UQER

Python 量化交易教程

Table of Contents

Python 量化交易教程

整理：mushroomqiu、飞龙

第一部分 新手入门

一 量化投资视频学习课程

> 来源：https://uqer.io/community/share/569c7068228e5b8ffc744fb3

优矿团队呕心力作的量化投资视频学习课程（第六课时已更新）

视频内容抢先看

- 优矿界面介绍，使用向导
- Python编程引导上
- Python编程引导下
- DataAPI使用介绍
- Quartz写策略介绍
- 如何参加大赛（模拟交易），如何在社区分享策略
- 社区精华帖介绍（上、下）

我们萌萌哒的量化分析师和美女经理都会在视频中以音频的形式跟大家打招呼，大家有什么问题和建议都可以在帖子下方留言，优矿小秘书欢迎大家随时来勾搭。

亲爱的优客们量化视频中的声音指标，你能分析出她是社区里的哪位大神嘛？

二 Python 手把手教学

量化分析师的Python日记【第1天：谁来给我讲讲Python？】

> 来源：https://uqer.io/community/share/54c89443f9f06c276f651a52

“谁来给我讲讲Python？”

作为无基础的初学者，只想先大概了解一下Python，随便编个小程序，并能看懂一般的程序，那些什么JAVA啊、C啊、继承啊、异常啊通通不懂怎么办，于是我找了很多资料，写成下面这篇日记，希望以完全初学者的角度入手来认识Python这个在量化领域日益重要的语言

一，熟悉基本

在正式介绍python之前，了解下面两个基本操作对后面的学习是有好处的：

1）基本的输入输出 可以在Python中使用`+`、`-`、`*`、`/`直接进行四则运算。

```
1+3*3

10
```

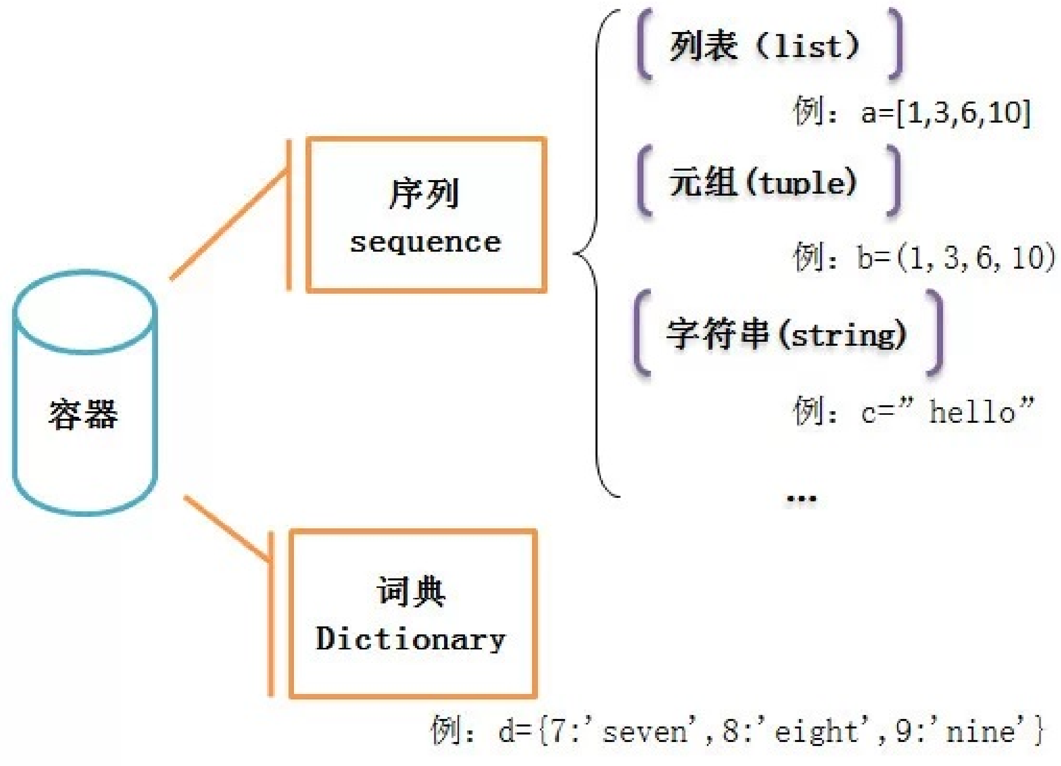
（2）导入模块 使用`import`可以导入模块，导入之后，就可以使用这个模块下面的函数了。 比如导入`math`模块，然后使用`math`模块下面的`sqrt`函数：

```
from math import sqrt
sqrt(9)

3.0
32.0
```

二，容器

1，什么是容器

开始学Python时，被它的数据结构，什么字典、序列、元组等等搞的很混乱，估计有跟我一样的初学者，所以我梳理了一下留存： 首先要从容器说起，Python中有一种名为容器的数据结构，顾名思义，容器，就是装数据的器具，它主要包括序列和词典，其中序列又主要包括列表、元组、字符串等（见下面那张图）。



列表的基本形式比如：`[1,3,6,10]` 或者 `[‘yes’,’no’,’OK’]`

元组的基本形式比如：`(1,3,6,10)` 或者 `(‘yes’,’no’,’OK’)`

字符串的基本形式比如：`’hello’`

以上几种属于序列，序列中的每一个元素都被分配一个序号——即元素的位置，也称为“索引”，第一个索引，即第一个元素的位置是0，第二个是1，依次类推。列表和元组的区别主要在于，列表可以修改，而元组不能（注意列表用中括号而元组用括号）。序列的这个特点，使得我们可以利用索引来访问序列中的某个或某几个元素，比如：

```
a=[1,3,6,10]
a[2]

6
```

```
b=(1,3,6,10)
b[2]

6
```

```
c='hello'
c[0:3]

'hel'
```

而与序列对应的“字典”则不一样，它是一个无序的容器，

它的基本形式比如： `d={7:'seven',8:'eight',9:'nine'}`

这是一个“键—值”映射的结构，因此字典不能通过索引来访问其中的元素，而要根据键来访问其中的元素：

```
d={7:'seven',8:'eight',9:'nine'}
d[8]

'eight'
```

2、序列的一些通用操作

除了上面说到的索引，列表、元组、字符串等这些序列还有一些共同的操作。

（1）索引（补充上面）

序列的最后一个元素的索引，也可以是-1，倒数第二个也可以用-2，依次类推：

```
a=[1,3,6,10]
print a[3]
print a[-1]

10
10
```

（2）分片

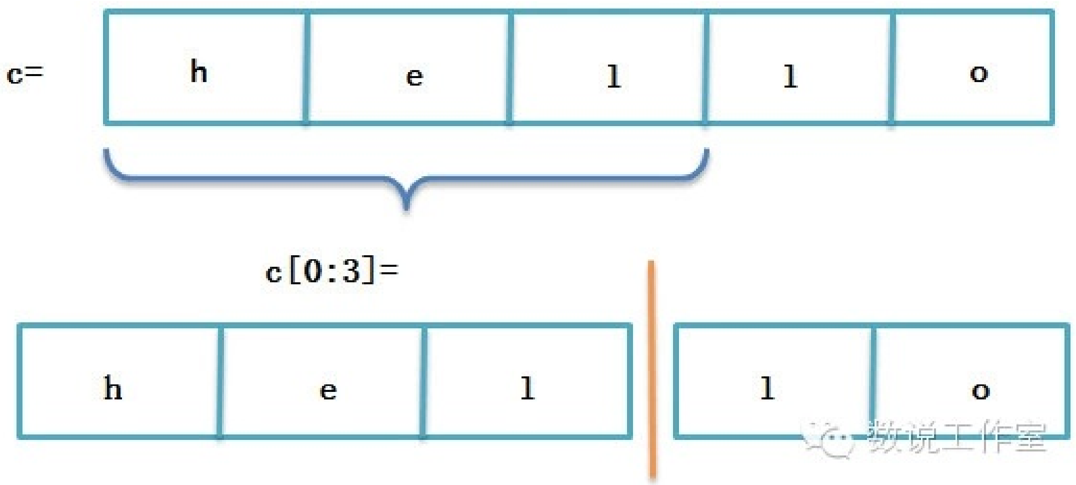
使用分片操作来访问一定范围内的元素，它的格式为：

```
a[开始索引:结束索引:步长]
```

那么访问的是，从开始索引号的那个元素，到结束索引号-1的那个元素，每间隔步长个元素访问一次，步长可以忽略，默认步长为1。

```
c='hello'
c[0:3]

'hel'
```

这个就好像把一个序列给分成几片几片的，所以叫做“分片”



（3）序列相加

即两种序列合并在一起，两种相同类型的序列才能相加

```
[1,2,3]+[4,5,6]

[1, 2, 3, 4, 5, 6]
```

```
'hello,'+'world!'

'hello,world!'
```

（4）成员资格

为了检查一个值是否在序列中，可以用 `in` 运算符

```
a='hello'
print 'o' in a
print 't' in a

True
False
```

3、列表操作

以上是序列共有的一些操作，列表也有一些自己独有的操作，这是其他序列所没有的

（1） `List` 函数

可以通过 `list` (序列)函数把一个序列转换成一个列表：

```
list('hello')

['h', 'e', 'l', 'l', 'o']
```

（2）元素赋值、删除

元素删除—— `del a[索引号]`

元素赋值—— `a[索引号]=值`

```
a

'hello'
```

```
b=list(a)
b

['h', 'e', 'l', 'o']
```

```
b[2]='t'
b

['h', 'e', 't', 'o']
```

分片赋值—— `a[开始索引号:结束索引号]=list(值)`

为列表的某一范围内的元素赋值，即在开始索引号到结束索引号-1的区间几个元素赋值，比如，利用上面语句，如何把hello变成heyyo？

```
b=list('hello')
b

['h', 'e', 'l', 'l', 'o']
```

```
b[2:4]=list('yy')
b

['h', 'e', 'y', 'y', 'o']
```

注意虽然 `“ll”` 处于 `“hello”` 这个单词的第2、3号索引的位置，但赋值时是用 `b[2:4]` 而不是 `b[2:3]` ，另外注意 `list()` 用小括号。

（3）列表方法

上面说过 `list` 函数，函数这个东西在很多语言中都有，比如excel里面的 `if` 函数、 `vlookup` 函数，SQL里面的 `count` 函数，以及各种语言中都有的 `sqrt` 函数等等，python中也有很多函数。 Python中的方法，是一个“与某些对象有紧密联系的”函数，所以列表方法，就是属于列表的函数，它可以对列表实现一些比较深入的操作，方法这样调用：

```
对象.方法(参数)
```

那么列表方法的调用就理所当然是：

```
列表.方法(参数)
```

常用的列表方法这么几个，以 `a=['h','e','l','l','o']` 为例：

```
a=['h','e','l','l','o']
a

['h', 'e', 'l', 'l', 'o']
```

给列表 `a` 的 `n` 索引位置插入一个元素 `m` ： `a.insert(n,m)`

```
a.insert(2,'t')
a

['h', 'e', 't', 'l', 'l', 'o']
```

给列表的最后添加元素 `m` ： `a.append(m)`

```
a.append('q')
a

['h', 'e', 't', 'l', 'l', 'o', 'q']
```

返回 `a` 列表中，元素 `m` 第一次出现的索引位置: `a.index(m)`

```
a.index('e')

1
```

删除 `a` 中的第一个 `m` 元素: `a.remove(m)`

```
a.remove('e')
a

['h', 't', 'l', 'l', 'o', 'q']
```

将列表 `a` 从大到小排列: `a.sort()`

```
a.sort()
a

['h', 'l', 'l', 'o', 'q', 't']
```

4、字典操作

（1） `dict` 函数

`dict` 函数可以通过关键字参数来创建字典，格式为：

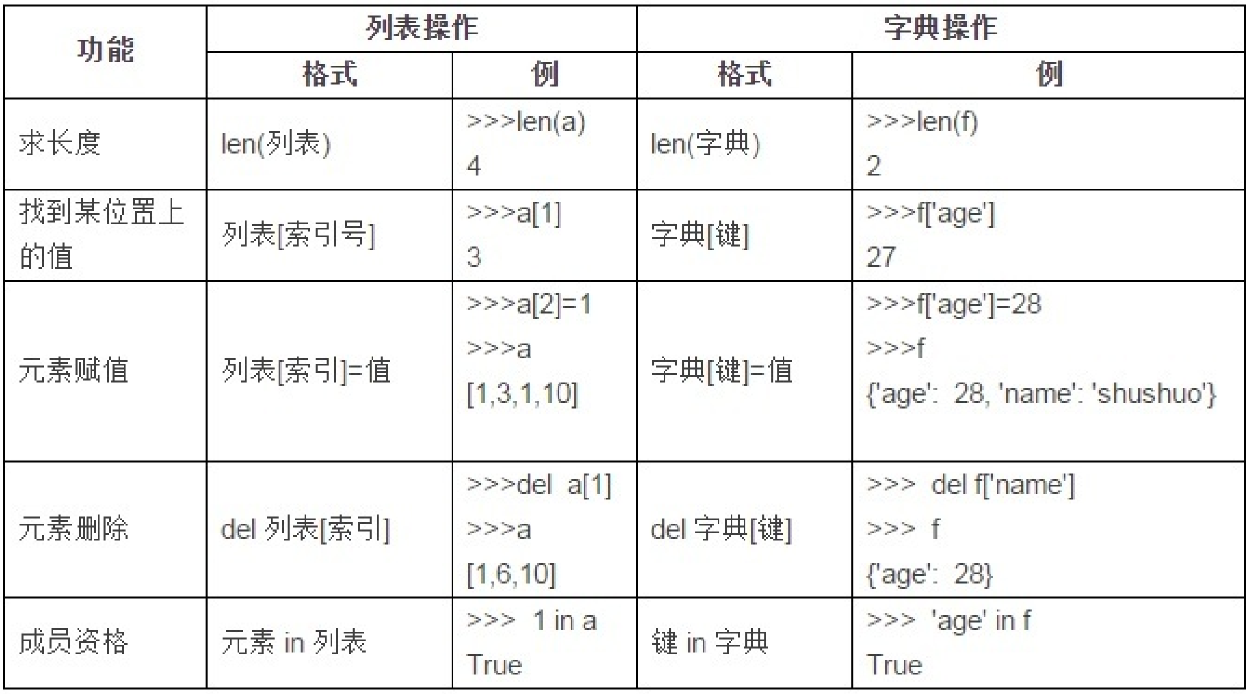
```
dict(参数1=值1,参数2=值2, …)={参数1:值1, 参数2=值2, …}
```

比如，如何创建一个名字 `name` 为 `jiayounet` ，年龄 `age` 为 `28` 的字典？

```
dict(name='jiayounet',age=27)

{'age': 27, 'name': 'jiayounet'}
```

（2）基本操作

字典的基本行为与列表在很多地方都相似，下面的例子以序列 `a=[1,3,6,10]` ，字典 `f={'age': 27, 'name': 'shushuo'}` 为例



功能	列表操作		字典操作	
	格式	例	格式	例
求长度	len(列表)	>>>len(a) 4	len(字典)	>>>len(f) 2
找到某位置上的值	列表[索引号]	>>>a[1] 3	字典[键]	>>>f['age'] 27
元素赋值	列表[索引]=值	>>>a[2]=1 >>>a [1,3,1,10]	字典[键]=值	>>>f['age']=28 >>>f {'age': 28, 'name': 'shushuo'}

元素删除	del 列表[索引]	>>>del a[1]
		>>>a
		[1,6,10]
			del 字典[键]	>>> del f['name']
				>>> f
				{'age': 28}

成员资格	元素 in 列表	>>> 1 in a True	键 in 字典	>>> 'age' in f True

日记小结： 今天学习了Python的基本页面、操作，以及几种主要的容器类型，天还要学习Python的函数、循环和条件、类，然后才算是对Python有一个大致的了解。

量化分析师的Python日记【第2天：再接着介绍一下Python呗】

> 来源：https://uqer.io/community/share/54c8af17f9f06c276f651a54

“谁来给我讲讲Python？”

第一天学习了Python的基本操作，以及几种主要的容器类型，今天学习python的函数、循环和条件、类，这样才算对Python有一个大致的了解。今天的学习大纲如下：

三、函数

1、定义函数

四、循环与条件

1、`if` 语句

2、`while true/break` 语句

3、`for` 语句

4、列表推导式

五、类

1、闲说类与对象

2、定义一个类

三，函数

1、定义函数

（1）定义规则

介绍列表方法的时候已经大概说过函数，学过数学的人都知道函数，给一个参数返回一个值。函数也可以自己定义。用如下的格式：

```
def 函数名(参数): 输入函数代码
```

函数代码中，`return` 表示返回的值。比如定义一个平方函数 `square(x)` ，输入参数 `x` ，返回 `x` 的平方：

```
def square(x):return x*x
square(9)

81
```

（2）定义变参数函数

有时需要定义参数个数可变的函数，有几个方法可以做到：

给参数指定默认值 比如，定义参数 `f(a,b=1,c=’hehe’)` ，那么在调用的时候，后面两个参数可以定义也可以不定义，不定义的话默认为 `b=1，c=’hehe’` ，因此如下调用都可以：

```
F(‘dsds’);
F(‘dsds’,2);
F(‘dsds’,2,’hdasda’);
```

参数关键字 上面的方法等于固定了参数的位置，第一个值就是第一个参数的赋值。而“参数关键字”方法，其实是固定了参数关键字，比如仍然定义参数 `f(a,b=1,c=’hehe’)` ，调用的时候可以用关键字来固定：

```
F(b=2,a=11)
```

位置可以动，只要参数关键指出来就可以了。

四、循环与条件

注意Python是用缩进来标识出哪一段属于本循环。

1、 `if` 语句

也是注意一是缩进，二是条件后面有冒号：

```
j=2.67
if j<3:
    print 'j<3'

j<3
```

对于多条件，注意的是elseif要写成elif，标准格式为：

```
if 条件1:
    执行语句1
elif 条件2:
    执行语句2
else:
    执行语句3
```

注意 `if…elif…else` 三个是并列的，不能有缩进：

```
t=3
if t<3:
    print 't<3'
elif t==3:
    print 't=3'
else:
    print 't>3'

t=3
```

2、 `while true/break` 语句

该语句的格式为

```
while true即条件为真:
    执行语句
    if中断语句条件 : break
```

看个例子：

```
a=3
while a<10:
    a=a+1
    print a
    if a==8: break

4
5
6
7
8
```

虽然 `while` 后面的条件是 `a<10` ，即 `a` 小于10的时候一直执行，但是 `if` 条件中规定了 `a` 为8时就 `break` 掉，因此，输出只能输到8。

3、`for` 语句

不多说了，可以遍历一个序列/字典等。

```
a=[1,2,3,4,5]
for i in a:
    print i

1
2
3
4
5
```

5、列表推导式：轻量级循环

列表推导式，是利用其它列表来创建一个新列表的方法，工作方式类似于 `for` 循环，格式为：

```
[输出值 for 条件]
```

当满足条件时，输出一个值，最终形成一个列表：

```
[x*x for x in range(10)]

[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
[x*x for x in range(10) if x%3==0]

[0, 9, 36, 81]
```

六、类

作为第二天的Python学习，先对类有一个大致的印象吧。

1、闲说类与对象

类是一个抽象的概念，它不存在于现实中的时间/空间里，类只是为所有的对象定义了抽象的属性与行为。就好像“Person（人）”这个类，它虽然可以包含很多个体，但它本身不存在于现实世界上。

而对象，是类的一个具体。它是一个实实在在存在的东西。如果上面说的“人”是一个抽象的类，那么你自己，就是这个类里一个具体的对象。

一个类的对象，也叫一个类的实例。再打个比方，类好比一个模具，对象就是用这个模具造出来的具有相同属性和方法的具体事物，俗话说：“他俩真像，好像一个模子刻出来的”，就是指的这个意思。 那么用这个模具造一个具体事物，就叫类的实例化。下面看一个具体的类：

2、定义一个类

```
class boy:
    gender='male'
    interest='girl'
    def say(self):
        return 'i am a boy'
```

上面的语句定义了一个类 `boy` ，我们来根据这儿类的模型构造一个具体的对象：

```
peter=boy()
```

现在来看看 `peter` 这个具体的实例有哪些属性和方法。

“什么叫属性和方法？”

它们都是“类”的两种表现，静态的叫属性，动态的叫方法。比如“人”类的属性有姓名、性别、身高、年龄、体重等等，“人”类的方法有走、跑、跳等等。

```
peter.gender

'male'
```

```
peter.interest

'girl'
```

```
peter.say()

'i am a boy
```

这里 `gender` 和 `interest` 是 `peter` 的属性，而 `say` 是他的方法。如果再实例化另一个对象比如 `sam` ：

```
sam=boy()
```

那么`sam`和`peter`有一样的属性和方法，可以说，“他们眞是一个模子刻出来的！”

量化分析师的Python日记【第3天：一大波金融Library来袭之numpy篇】

> 来源：https://uqer.io/community/share/54ca15f9f9f06c276f651a56

接下来要给大家介绍的系列中包含了Python在量化金融中运用最广泛的几个Library:

- numpy
- scipy
- pandas
- matplotlib

会给初学者一一介绍

NumPy 简介

一、NumPy是什么？

量化分析的工作涉及到大量的数值运算，一个高效方便的科学计算工具是必不可少的。Python语言一开始并不是设计为科学计算使用的语言，随着越来越多的人发现Python的易用性，逐渐出现了关于Python的大量外部扩展，NumPy (Numeric Python)就是其中之一。NumPy提供了大量的数值编程工具，可以方便地处理向量、矩阵等运算，极大地便利了人们在科学计算方面的工作。另一方面，Python是免费，相比于花费高额的费用使用Matlab，NumPy的出现使Python得到了更多人的青睐。

我们可以简单看一下如何开始使用NumPy：

```
import numpy
numpy.version.full_version

'1.8.0'
```

我们使用了 `import` 命令导入了NumPy，并使用 `numpy.version.full_version` 查出了量化实验室里使用的NumPy版本为1.8.0。在往后的介绍中，我们将大量使用NumPy中的函数，每次都添加 `numpy` 在函数前作为前缀比较费劲，在之前的介绍中，我们提及了引入外部扩展模块时的小技巧，可以使用 `from numpy import *` 解决这一问题。

那么问题解决了？慢！Python的外部扩展成千上万，在使用中很可能会 `import` 好几个外部扩展模块，如果某个模块包含的属性和方法与另一个模块同名，就必须使用 `import module` 来避免名字的冲突。即所谓的名字空间（namespace）混淆了，所以这前缀最好还是带上。

那有没有简单的办法呢？有的，我们可以在 `import` 扩展模块时添加模块在程序中的别名，调用时就不必写成全名了，例如，我们使用 `np` 作为别名并调用 `version.full_version` 函数：

```
import numpy as np
np.version.full_version

'1.8.0'
```

二、初窥NumPy对象：数组

NumPy中的基本对象是同类型的多维数组（homogeneous multidimensional array），这和C++中的数组是一致的，例如字符型和数值型就不可共存于同一个数组中。先上例子：

```
a = np.arange(20)
```

这里我们生成了一个一维数组 `a` ，从0开始，步长为1，长度为20。Python中的计数是从0开始的，R和Matlab的使用者需要小心。可以使用 `print` 查看：

```
print a

numpy.ndarray
```

通过函数 `reshape` ，我们可以重新构造一下这个数组，例如，我们可以构造一个 `4*5` 的二维数组，其中 `reshape` 的参数表示各维度的大小，且按各维顺序排列（两维时就是按行排列，这和R中按列是不同的）：

```
a = a.reshape(4, 5)
print a

[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]]
```

构造更高维的也没问题:

```
a = a.reshape(2, 2, 5)
print a

[[[ 0  1  2  3  4]
  [ 5  6  7  8  9]]

 [[10 11 12 13 14]
  [15 16 17 18 19]]]
```

既然 `a` 是 `array` ，我们还可以调用 `array` 的函数进一步查看 `a` 的相关属性： `ndim` 查看维度； `shape` 查看各维度的大小； `size` 查看全部的元素个数，等于各维度大小的乘积； `dtype` 可查看元素类型； `dsize` 查看元素占位（bytes）大小。

```
a.ndim

3
```

```
a.shape

(2, 2, 5)
```

```
a.size

20
```

```
a.dtype

dtype('int64')
```

三、创建数组

数组的创建可通过转换列表实现，高维数组可通过转换嵌套列表实现：

```
raw = [0,1,2,3,4]
a = np.array(raw)
a

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

```
raw = [[0,1,2,3,4], [5,6,7,8,9]]
b = np.array(raw)
b

array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
```

一些特殊的数组有特别定制的命令生成，如 `4*5` 的全零矩阵：

```
d = (4, 5)
np.zeros(d)

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

默认生成的类型是浮点型，可以通过指定类型改为整型：

```
d = (4, 5)
np.ones(d, dtype=int)

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

`[0, 1)` 区间的随机数数组：

```
np.random.rand(5)

array([ 0.93807818,  0.45307847,  0.90732828,  0.36099623,  0.71
981451])
```

四、数组操作

简单的四则运算已经重载过了，全部的 `+` ，`-` ，`*` ，`/` 运算都是基于全部的数组元素的，以加法为例：

```
a = np.array([[1.0, 2], [2, 4]])
print "a:"
print a
b = np.array([[3.2, 1.5], [2.5, 4]])
print "b:"
print b
print "a+b:"
print a+b

a:
[[ 1.  2.]
 [ 2.  4.]]
b:
[[ 3.2  1.5]
 [ 2.5  4. ]]
a+b:
[[ 4.2  3.5]
 [ 4.5  8. ]]
```

这里可以发现， `a` 中虽然仅有一个与元素是浮点数，其余均为整数，在处理中Python会自动将整数转换为浮点数（因为数组是同质的），并且，两个二维数组相加要求各维度大小相同。当然，NumPy里这些运算符也可以对标量和数组操作，结果是数组的全部元素对应这个标量进行运算，还是一个数组：

```
print "3 * a:"
print 3 * a
print "b + 1.8:"
print b + 1.8

3 * a:
[[  3.   6.]
 [  6.  12.]]
b + 1.8:
[[ 5.   3.3]
 [ 4.3  5.8]]
```

类似C++， `+=` 、 `-=` 、 `*=` 、 `/=` 操作符在NumPy中同样支持：

```
a /= 2
print a

[[ 0.5  1. ]
 [ 1.   2. ]]
```

开根号求指数也很容易：

```
print "a:"
print a
print "np.exp(a):"
print np.exp(a)
print "np.sqrt(a):"
print np.sqrt(a)
print "np.square(a):"
print np.square(a)
print "np.power(a, 3):"
print np.power(a, 3)

a:
[[ 0.5  1. ]
 [ 1.   2. ]]
np.exp(a):
[[ 1.64872127  2.71828183]
 [ 2.71828183  7.3890561 ]]
np.sqrt(a):
[[ 0.70710678  1.        ]
 [ 1.          1.41421356]]
np.square(a):
[[ 0.25  1.  ]
 [ 1.    4.  ]]
np.power(a, 3):
[[ 0.125  1.   ]
 [ 1.     8.   ]]
```

需要知道二维数组的最大最小值怎么办？想计算全部元素的和、按行求和、按列求和怎么办？`for`循环吗？不，NumPy的`ndarray`类已经做好函数了：

```
a = np.arange(20).reshape(4,5)
print "a:"
print a
print "sum of all elements in a: " + str(a.sum())
print "maximum element in a: " + str(a.max())
print "minimum element in a: " + str(a.min())
print "maximum element in each row of a: " + str(a.max(axis=1))
print "minimum element in each column of a: " + str(a.min(axis=0
))

a:
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]]
sum of all elements in a: 190
maximum element in a: 19
minimum element in a: 0
maximum element in each row of a: [ 4  9 14 19]
minimum element in each column of a: [0 1 2 3 4]
```

科学计算中大量使用到矩阵运算，除了数组，NumPy同时提供了矩阵对象（ `matrix` ）。矩阵对象和数组的主要有两点差别：一是矩阵是二维的，而数组的可以是任意正整数维；二是矩阵的 `*` 操作符进行的是矩阵乘法，乘号左侧的矩阵列和乘号右侧的矩阵行要相等，而在数组中 `*` 操作符进行的是每一元素的对应相乘，乘号两侧的数组每一维大小需要一致。数组可以通过 `asmatrix` 或者 `mat` 转换为矩阵，或者直接生成也可以：

```
a = np.arange(20).reshape(4, 5)
a = np.asmatrix(a)
print type(a)

b = np.matrix('1.0 2.0; 3.0 4.0')
print type(b)

<class 'numpy.matrixlib.defmatrix.matrix'>
<class 'numpy.matrixlib.defmatrix.matrix'>
```

再来看一下矩阵的乘法，这使用 `arange` 生成另一个矩阵 `b` ， `arange` 函数还可以通过 `arange(起始，终止，步长)` 的方式调用生成等差数列，注意含头不含尾。

```
b = np.arange(2, 45, 3).reshape(5, 3)
b = np.mat(b)
print b

[[ 2  5  8]
 [11 14 17]
 [20 23 26]
 [29 32 35]
 [38 41 44]]
```

有人要问了， `arange` 指定的是步长，如果想指定生成的一维数组的长度怎么办？好办， `linspace` 就可以做到：

```
np.linspace(0, 2, 9)

array([ 0.  ,  0.25,  0.5 ,  0.75,  1.  ,  1.25,  1.5 ,  1.75,
2.  ])
```

回到我们的问题，矩阵 `a` 和 `b` 做矩阵乘法：

```
print "matrix a:"
print a
print "matrix b:"
print b
c = a * b
print "matrix c:"
print c

print c
查看全部
matrix a:
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]]
matrix b:
[[ 2  5  8]
 [11 14 17]
 [20 23 26]
 [29 32 35]
 [38 41 44]]
matrix c:
[[ 290  320  350]
 [ 790  895 1000]
 [1290 1470 1650]
 [1790 2045 2300]]
```

五、数组元素访问

数组和矩阵元素的访问可通过下标进行，以下均以二维数组（或矩阵）为例：

```
a = np.array([[3.2, 1.5], [2.5, 4]])
print a[0][1]
print a[0, 1]

1.5
1.5
```

可以通过下标访问来修改数组元素的值：

```
b = a
a[0][1] = 2.0
print "a:"
print a
print "b:"
print b

a:
[[ 3.2  2. ]
 [ 2.5  4. ]]
b:
[[ 3.2  2. ]
 [ 2.5  4. ]]
```

现在问题来了，明明改的是 `a[0][1]` ，怎么连 `b[0][1]` 也跟着变了？这个陷阱在Python编程中很容易碰上，其原因在于Python不是真正将 `a` 复制一份给 `b` ，而是将 `b` 指到了 `a` 对应数据的内存地址上。想要真正的复制一份 `a` 给 `b` ，可以使用 `copy` ：

```
a = np.array([[3.2, 1.5], [2.5, 4]])
b = a.copy()
a[0][1] = 2.0
print "a:"
print a
print "b:"
print b

a:
[[ 3.2  2. ]
 [ 2.5  4. ]]
b:
[[ 3.2  1.5]
 [ 2.5  4. ]]
```

若对 `a` 重新赋值，即将 `a` 指到其他地址上， `b` 仍在原来的地址上：

```
a = np.array([[3.2, 1.5], [2.5, 4]])
b = a
a = np.array([[2, 1], [9, 3]])
print "a:"
print a
print "b:"
print b

a:
[[2 1]
 [9 3]]
b:
[[ 3.2  1.5]
 [ 2.5  4. ]]
```

利用 `:` 可以访问到某一维的全部数据，例如取矩阵中的指定列：

```
a = np.arange(20).reshape(4, 5)
print "a:"
print a
print "the 2nd and 4th column of a:"
print a[:,[1,3]]

a:
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]]
the 2nd and 4th column of a:
[[ 1  3]
 [ 6  8]
 [11 13]
 [16 18]]
```

稍微复杂一些，我们尝试取出满足某些条件的元素，这在数据的处理中十分常见，通常用在单行单列上。下面这个例子是将第一列大于5的元素（10和15）对应的第三列元素（12和17）取出来：

```
a[:, 2][a[:, 0] > 5]

array([12, 17])
```

可使用 `where` 函数查找特定值在数组中的位置：

```
loc = numpy.where(a==11)
print loc
print a[loc[0][0], loc[1][0]]

(array([2]), array([1]))
11
```

六、数组操作

还是拿矩阵（或二维数组）作为例子，首先来看矩阵转置：

```
a = np.random.rand(2,4)
print "a:"
print a
a = np.transpose(a)
print "a is an array, by using transpose(a):"
print a
b = np.random.rand(2,4)
b = np.mat(b)
print "b:"
print b
print "b is a matrix, by using b.T:"
print b.T

a:
[[ 0.17571282  0.98510461  0.94864387  0.50078988]
 [ 0.09457965  0.70251658  0.07134875  0.43780173]]
a is an array, by using transpose(a):
[[ 0.17571282  0.09457965]
 [ 0.98510461  0.70251658]
 [ 0.94864387  0.07134875]
 [ 0.50078988  0.43780173]]
b:
[[ 0.09653644  0.46123468  0.50117363  0.69752578]
 [ 0.60756723  0.44492537  0.05946373  0.4858369 ]]
b is a matrix, by using b.T:
[[ 0.09653644  0.60756723]
 [ 0.46123468  0.44492537]
 [ 0.50117363  0.05946373]
 [ 0.69752578  0.4858369 ]]
```

矩阵求逆：

```
import numpy.linalg as nlg
a = np.random.rand(2,2)
a = np.mat(a)
print "a:"
print a
ia = nlg.inv(a)
print "inverse of a:"
print ia
print "a * inv(a)"
print a * ia

a:
[[ 0.86211266  0.6885563 ]
 [ 0.28798536  0.70810425]]
inverse of a:
[[ 1.71798445 -1.6705577 ]
 [-0.69870271  2.09163573]]
a * inv(a)
[[ 1.  0.]
 [ 0.  1.]]
```

求特征值和特征向量

```
a = np.random.rand(3,3)
eig_value, eig_vector = nlg.eig(a)
print "eigen value:"
print eig_value
print "eigen vector:"
print eig_vector

eigen value:
[ 1.35760609  0.43205379 -0.53470662]
eigen vector:
[[-0.76595379 -0.88231952 -0.07390831]
 [-0.55170557  0.21659887 -0.74213622]
 [-0.33005418  0.41784829  0.66616169]]
```

按列拼接两个向量成一个矩阵：

```
a = np.array((1,2,3))
b = np.array((2,3,4))
print np.column_stack((a,b))

[[1 2]
 [2 3]
 [3 4]]
```

在循环处理某些数据得到结果后，将结果拼接成一个矩阵是十分有用的，可以通过 `vstack` 和 `hstack` 完成：

```
a = np.random.rand(2,2)
b = np.random.rand(2,2)
print "a:"
print a
print "b:"
print a
c = np.hstack([a,b])
d = np.vstack([a,b])
print "horizontal stacking a and b:"
print c
print "vertical stacking a and b:"
print d

a:
[[ 0.6738195   0.4944045 ]
 [ 0.25702675  0.15422012]]
b:
[[ 0.6738195   0.4944045 ]
 [ 0.25702675  0.15422012]]
horizontal stacking a and b:
[[ 0.6738195   0.4944045   0.28058267  0.0967197 ]
 [ 0.25702675  0.15422012  0.55191041  0.04694485]]
vertical stacking a and b:
[[ 0.6738195   0.4944045 ]
 [ 0.25702675  0.15422012]
 [ 0.28058267  0.0967197 ]
 [ 0.55191041  0.04694485]]
```

七、缺失值

缺失值在分析中也是信息的一种，NumPy提供 `nan` 作为缺失值的记录，通过 `isnan` 判定。

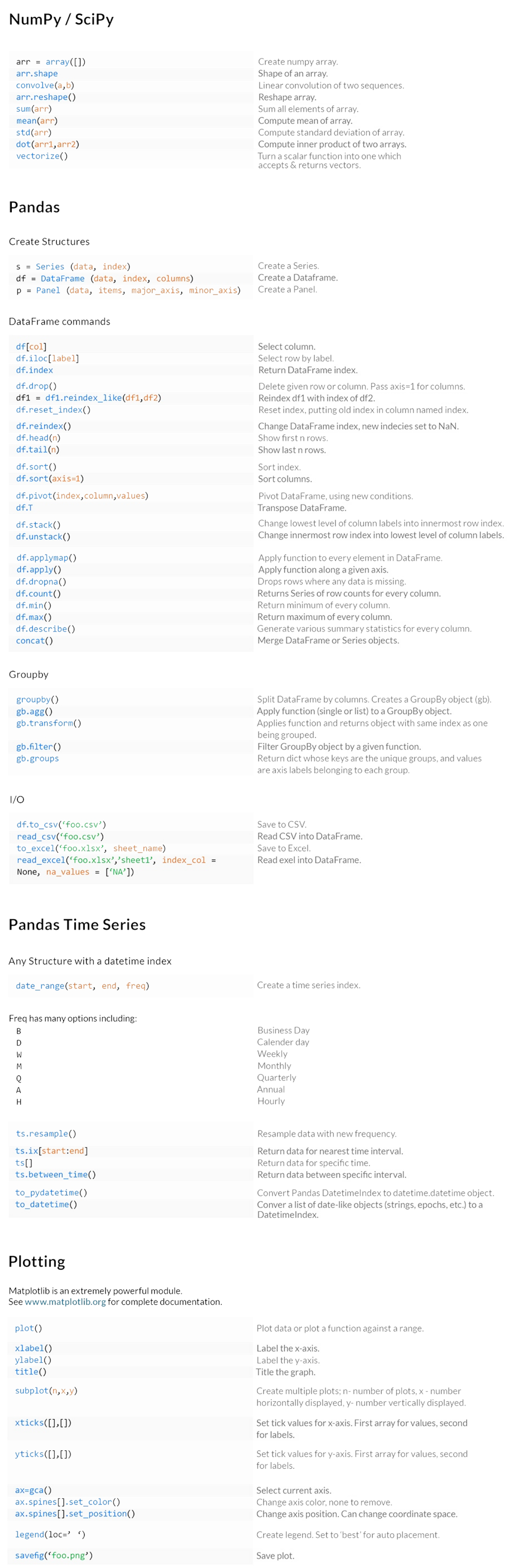
```
a = np.random.rand(2,2)
a[0, 1] = np.nan
print np.isnan(a)

[[False  True]
 [False False]]
```

`nan_to_num` 可用来将 `nan` 替换成0，在后面会介绍到的更高级的模块 `pandas` 时，我们将看到 `pandas` 提供能指定 `nan` 替换值的函数。

```
print np.nan_to_num(a)

[[ 0.58144238  0.        ]
 [ 0.26789784  0.48664306]]
```

NumPy还有很多的函数，想详细了解可参考链接 http://wiki.scipy.org/Numpy_Example_List 和 http://docs.scipy.org/doc/numpy

最后献上NumPy SciPy Pandas Cheat Sheet



NumPy / SciPy

`arr = array([])`	Create numpy array.
`arr.shape`	Shape of an array.
`convolve(a,b)`	Linear convolution of two sequences.
`arr.reshape()`	Reshape array.
`sum(arr)`	Sum all elements of array.
`mean(arr)`	Compute mean of array.
`std(arr)`	Compute standard deviation of array.
`dot(arr1,arr2)`	Compute inner product of two arrays.
`vectorize()`	Turn a scalar function into one which accepts & returns vectors.

Pandas

Create Structures

`s = Series (data, index)`	Create a Series.
`df = DataFrame (data, index, columns)`	Create a Dataframe.
`p = Panel (data, items, major_axis, minor_axis)`	Create a Panel.

DataFrame commands

`df[col]`	Select column.
`df.iloc[label]`	Select row by label.
`df.index`	Return DataFrame index.
`df.drop()`	Delete given row or column. Pass axis=1 for columns.
`df1 = df1.reindex_like(df1,df2)`	Reindex df1 with index of df2.
`df.reset_index()`	Reset index, putting old index in column named index.
`df.reindex()`	Change DataFrame index, new indecies set to NaN.
`df.head(n)`	Show first n rows.
`df.tail(n)`	Show last n rows.
`df.sort()`	Sort index.
`df.sort(axis=1)`	Sort columns.
`df.pivot(index,column,values)`	Pivot DataFrame, using new conditions.
`df.T`	Transpose DataFrame.
`df.stack()`	Change lowest level of column labels into innermost row index.
`df.unstack()`	Change innermost row index into lowest level of column labels.
`df.applymap()`	Apply function to every element in DataFrame.
`df.apply()`	Apply function along a given axis.
`df.dropna()`	Drops rows where any data is missing.
`df.count()`	Returns Series of row counts for every column.
`df.min()`	Return minimum of every column.
`df.max()`	Return maximum of every column.
`df.describe()`	Generate various summary statistics for every column.
`concat()`	Merge DataFrame or Series objects.

Groupby

`groupby()`	Split DataFrame by columns. Creates a GroupBy object (gb).
`gb.agg()`	Apply function (single or list) to a GroupBy object.
`gb.transform()`	Applies function and returns object with same index as one being grouped.
`gb.filter()`	Filter GroupBy object by a given function.
`gb.groups`	Return dict whose keys are the unique groups, and values are axis labels belonging to each group.

I/O

`df.to_csv('foo.csv')`	Save to CSV.
`read_csv('foo.csv')`	Read CSV into DataFrame.
`to_excel('foo.xlsx', sheet_name)`	Save to Excel.
`read_excel('foo.xlsx','sheet1', index_col = None, na_values = ['NA'])`	Read exel into DataFrame.

Pandas Time Series

Any Structure with a datetime index

`date_range(start, end, freq)`	Create a time series index.

Freq has many options including:

B	Business Day
D	Calender day
W	Weekly
M	Monthly
Q	Quarterly
A	Annual
H	Hourly

`ts.resample()`	Resample data with new frequency.
`ts.ix[start:end]`	Return data for nearest time interval.
`ts[]`	Return data for specific time.
`ts.between_time()`	Return data between specific interval.
`to_pydatetime()`	Convert Pandas DatetimeIndex to datetime.datetime object.
`to_datetime()`	Conver a list of date-like objects (strings, epochs, etc.) to a DatetimeIndex.

Plotting

Matplotlib is an extremely powerful module.
See www.matplotlib.org for complete documentation.

`plot()`	Plot data or plot a function against a range.
`xlabel()`	Label the x-axis.
`ylabel()`	Label the y-axis.
`title()`	Title the graph.
`subplot(n,x,y)`	Create multiple plots; n- number of plots, x - number horizontally displayed, y- number vertically displayed.
`xticks([],[])`	Set tick values for x-axis. First array for values, second for labels.
`yticks([],[])`	Set tick values for y-axis. First array for values, second for labels.
`ax=gca()`	Select current axis.
`ax.spines[].set_color()`	Change axis color, none to remove.
`ax.spines[].set_position()`	Change axis position. Can change coordinate space.
`legend(loc=' ')`	Create legend. Set to 'best' for auto placement.
`savefig('foo.png')`	Save plot.

量化分析师的Python日记【第4天：一大波金融Library来袭之scipy篇】

> 来源：https://uqer.io/community/share/54d83bb3f9f06c276f651a6e

上一篇介绍了`numpy`,本篇中着重介绍一下另一个量化金融中常用的库`scipy`

一、SciPy概述

前篇已经大致介绍了NumPy，接下来让我们看看SciPy能做些什么。NumPy替我们搞定了向量和矩阵的相关操作，基本上算是一个高级的科学计算器。SciPy基于NumPy提供了更为丰富和高级的功能扩展，在统计、优化、插值、数值积分、时频转换等方面提供了大量的可用函数，基本覆盖了基础科学计算相关的问题。

在量化分析中，运用最广泛的是统计和优化的相关技术，本篇重点介绍SciPy中的统计和优化模块，其他模块在随后系列文章中用到时再做详述。

本篇会涉及到一些矩阵代数，如若感觉不适，可考虑跳过第三部分或者在理解时简单采用一维的标量代替高维的向量。

首先还是导入相关的模块，我们使用的是SciPy里面的统计和优化部分：

```
import numpy as np
import scipy.stats as stats
import scipy.optimize as opt
```

二、统计部分

2.1 生成随机数

我们从生成随机数开始，这样方便后面的介绍。生成n个随机数可用`rv_continuous.rvs(size=n)`或`rv_discrete.rvs(size=n)`，其中`rv_continuous`表示连续型的随机分布，如均匀分布（uniform）、正态分布（norm）、贝塔分布（beta）等；`rv_discrete`表示离散型的随机分布，如伯努利分布（bernoulli）、几何分布（geom）、泊松分布（poisson）等。我们生成10个`[0, 1]`区间上的随机数和10个服从参数`a=4`，`b=2`的贝塔分布随机数：

```
rv_unif = stats.uniform.rvs(size=10)
print rv_unif
rv_beta = stats.beta.rvs(size=10, a=4, b=2)
print rv_beta

[ 0.6419336   0.48403001  0.89548809  0.73837498  0.65744886  0.
41845577
  0.3823512   0.0985301   0.66785949  0.73163835]
[ 0.82164685  0.69563836  0.74207073  0.94348192  0.82979411  0.
87013796
  0.78412952  0.47508183  0.29296073  0.52551156]
```

在每个随机分布的生成函数里，都内置了默认的参数，如均匀分布的上下界默认是0和1。可是一旦需要修改这些参数，每次生成随机都要敲这么老长一串有点麻烦，能不能简单点？SciPy里头有一个Freezing的功能，可以提供简便版本的命令。 `SciPy.stats` 支持定义出某个具体的分布的对象，我们可以做如下的定义，让 `beta` 直接指代具体参数 `a=4` 和 `b=2` 的贝塔分布。为让结果具有可比性，这里指定了随机数的生成种子，由NumPy提供。

```
np.random.seed(seed=2015)
rv_beta = stats.beta.rvs(size=10, a=4, b=2)
print "method 1:"
print rv_beta

np.random.seed(seed=2015)
beta = stats.beta(a=4, b=2)
print "method 2:"
print beta.rvs(size=10)

method 1:
[ 0.43857338  0.9411551   0.75116671  0.92002864  0.62030521  0.
56585548
  0.41843548  0.5953096   0.88983036  0.94675351]
method 2:
[ 0.43857338  0.9411551   0.75116671  0.92002864  0.62030521  0.
56585548
  0.41843548  0.5953096   0.88983036  0.94675351]
```

2.2 假设检验

好了，现在我们生成一组数据，并查看相关的统计量（相关分布的参数可以在这里查到）：

```
norm_dist = stats.norm(loc=0.5, scale=2)
n = 200
dat = norm_dist.rvs(size=n)
print "mean of data is: " + str(np.mean(dat))
print "median of data is: " + str(np.median(dat))
print "standard deviation of data is: " + str(np.std(dat))

mean of data is: 0.383309149888
median of data is: 0.394980561217
standard deviation of data is: 2.00589851641
```

假设这个数据是我们获取到的实际的某些数据，如股票日涨跌幅，我们对数据进行简单的分析。最简单的是检验这一组数据是否服从假设的分布，如正态分布。这个问题是典型的单样本假设检验问题，最为常见的解决方案是采用K-S检验（Kolmogorov-Smirnov test）。单样本K-S检验的原假设是给定的数据来自和原假设分布相同的分布，在SciPy中提供了 `kstest` 函数，参数分别是数据、拟检验的分布名称和对应的参数：

```
mu = np.mean(dat)
sigma = np.std(dat)
stat_val, p_val = stats.kstest(dat, 'norm', (mu, sigma))
print 'KS-statistic D = %6.3f p-value = %6.4f' % (stat_val, p_va
l)

KS-statistic D =  0.037 p-value = 0.9428
```

假设检验的p-value值很大（在原假设下，p-value是服从 `[0, 1]` 区间上的均匀分布的随机变量，可参考 http://en.wikipedia.org/wiki/P-value ），因此我们接受原假设，即该数据通过了正态性的检验。在正态性的前提下，我们可进一步检验这组数据的均值是不是0。典型的方法是t检验（t-test），其中单样本的t检验函数为 `ttest_1samp` ：

```
stat_val, p_val = stats.ttest_1samp(dat, 0)
print 'One-sample t-statistic D = %6.3f, p-value = %6.4f' % (sta
t_val, p_val)

One-sample t-statistic D =  2.696, p-value = 0.0076
```

我们看到 `p-value<0.05` ，即给定显著性水平0.05的前提下，我们应拒绝原假设：数据的均值为0。我们再生成一组数据，尝试一下双样本的t检验（ `ttest_ind` ）：

```
norm_dist2 = stats.norm(loc=-0.2, scale=1.2)
dat2 = norm_dist2.rvs(size=n/2)
stat_val, p_val = stats.ttest_ind(dat, dat2, equal_var=False)
print 'Two-sample t-statistic D = %6.3f, p-value = %6.4f' % (sta
t_val, p_val)

Two-sample t-statistic D =  3.572, p-value = 0.0004
```

注意，这里我们生成的第二组数据样本大小、方差和第一组均不相等，在运用t检验时需要使用Welch's t-test，即指定 `ttest_ind` 中的 `equal_var=False` 。我们同样得到了比较小的 `p-value$` ，在显著性水平0.05的前提下拒绝原假设，即认为两组数据均值不等。

`stats` 还提供其他大量的假设检验函数，如 `bartlett` 和 `levene` 用于检验方差是否相等； `anderson_ksam` p用于进行Anderson-Darling的K-样本检验等。

2.3 其他函数

有时需要知道某数值在一个分布中的分位，或者给定了一个分布，求某分位上的数值。这可以通过 `cdf` 和 `ppf` 函数完成：

```
g_dist = stats.gamma(a=2)
print "quantiles of 2, 4 and 5:"
print g_dist.cdf([2, 4, 5])
print "Values of 25%, 50% and 90%:"
print g_dist.pdf([0.25, 0.5, 0.95])

quantiles of 2, 4 and 5:
[ 0.59399415  0.90842181  0.95957232]
Values of 25%, 50% and 90%:
[ 0.1947002   0.30326533  0.36740397]
```

对于一个给定的分布，可以用 `moment` 很方便的查看分布的矩信息，例如我们查看 `N(0,1)` 的六阶原点矩：

```
stats.norm.moment(6, loc=0, scale=1)

15.000000000895332
```

`describe` 函数提供对数据集的统计描述分析，包括数据样本大小，极值，均值，方差，偏度和峰度：

```
norm_dist = stats.norm(loc=0, scale=1.8)
dat = norm_dist.rvs(size=100)
info = stats.describe(dat)
print "Data size is: " + str(info[0])
print "Minimum value is: " + str(info[1][0])
print "Maximum value is: " + str(info[1][1])
print "Arithmetic mean is: " + str(info[2])
print "Unbiased variance is: " + str(info[3])
print "Biased skewness is: " + str(info[4])
print "Biased kurtosis is: " + str(info[5])

Data size is: 100
Minimum value is: -5.73556523159
Maximum value is: 3.77439818033
Arithmetic mean is: -0.00559348382755
Unbiased variance is: 3.64113204268
Biased skewness is: -0.600615731841
Biased kurtosis is: 0.432147856587
```

当我们知道一组数据服从某些分布的时候，可以调用 `fit` 函数来得到对应分布参数的极大似然估计（MLE, maximum-likelihood estimation）。以下代码示例了假设数据服从正态分布，用极大似然估计分布参数：

```
norm_dist = stats.norm(loc=0, scale=1.8)
dat = norm_dist.rvs(size=100)
mu, sigma = stats.norm.fit(dat)
print "MLE of data mean:" + str(mu)
print "MLE of data standard deviation:" + str(sigma)

MLE of data mean:0.00712958665203
MLE of data standard deviation:1.71228079199
```

`pearsonr` 和 `spearmanr` 可以计算 `Pearson` 和 `Spearman` 相关系数，这两个相关系数度量了两组数据的相互线性关联程度：

```
norm_dist = stats.norm()
dat1 = norm_dist.rvs(size=100)
exp_dist = stats.expon()
dat2 = exp_dist.rvs(size=100)
cor, pval = stats.pearsonr(dat1, dat2)
print "Pearson correlation coefficient: " + str(cor)
cor, pval = stats.pearsonr(dat1, dat2)
print "Spearman's rank correlation coefficient: " + str(cor)

Pearson correlation coefficient: -0.0345336831321
Spearman's rank correlation coefficient: -0.0345336831321
```

其中的p-value表示原假设（两组数据不相关）下，相关系数的显著性。

最后，在分析金融数据中使用频繁的线性回归在SciPy中也有提供，我们来看一个例子：

```
x = stats.chi2.rvs(3, size=50)
y = 2.5 + 1.2 * x + stats.norm.rvs(size=50, loc=0, scale=1.5)
slope, intercept, r_value, p_value, std_err = stats.linregress(x
, y)
print "Slope of fitted model is:" , slope
print "Intercept of fitted model is:", intercept
print "R-squared:", r_value**2

Slope of fitted model is: 1.20010505908
Intercept of fitted model is: 2.04778311819
R-squared: 0.781316678233
```

在前面的链接中，可以查到大部分 `stat` 中的函数，本节权作简单介绍，挖掘更多功能的最好方法还是直接读原始的文档。另外，StatsModels（ http://statsmodels.sourceforge.net ）模块提供了更为专业，更多的统计相关函数。若在SciPy没有满足需求，可以采用StatsModels。

三、优化部分

优化问题在投资中可谓是根本问题，如果手上有众多可选的策略，应如何从中选择一个“最好”的策略进行投资呢？这时就需要用到一些优化技术针对给定的指标进行寻优。随着越来越多金融数据的出现，机器学习逐渐应用在投资领域，在机器学习中，优化也是十分重要的一个部分。以下介绍一些常见的优化方法，虽然例子是人工生成的，不直接应用于实际金融数据，我们希望读者在后面遇到优化问题时，能够从这些简单例子迅速上手解决。

3.1 无约束优化问题

所谓的无约束优化问题指的是一个优化问题的寻优可行集合是目标函数自变量的定义域，即没有外部的限制条件。例如，求解优化问题

$$\text{minimize} \quad f(x) = x^2 - 4.8x + 1.2$$

就是一个无约束优化问题，而求解

$$\begin{array}{ll} \text{minimize} & f(x) = x^2 - 4.8x + 1.2 \\ \text{subject to} & x \geq 0 \end{array}$$

则是一个带约束的优化问题。更进一步，我们假设考虑的问题全部是凸优化问题，即目标函数是凸函数，其自变量的可行集是凸集。（详细定义可参考斯坦福大学Stephen Boyd教授的教材convex optimization，下载链

接：http://stanford.edu/~boyd/cvxbook ）

我们以Rosenbrock函数

$$f(\mathbf{x}) = \sum_{i=1}^{N-1} 100(x_i - x_{i-1}^2)^2 + (1 - x_{i-1})^2$$

作为寻优的目标函数来简要介绍在SciPy中使用优化模块 `scipy.optimize` 。

首先需要定义一下这个Rosenbrock函数：

```
def rosen(x):
    """The Rosenbrock function"""
    return sum(100.0*(x[1:]-x[:-1]**2.0)**2.0 + (1-x[:-1])**2.0)
```

3.1.1 Nelder-Mead单纯形法

单纯形法是运筹学中介绍的求解线性规划问题的通用方法，这里的Nelder-Mead单纯形法与其并不相同，只是用到单纯形的概念。设定起始点 `x0=(1.3,0.7,0.8,1.9,1.2)` ，并进行最小化的寻优。这里 `xtol` 表示迭代收敛的容忍误差上界：

```
x_0 = np.array([0.5, 1.6, 1.1, 0.8, 1.2])
res = opt.minimize(rosen, x_0, method='nelder-mead', options={'x
tol': 1e-8, 'disp': True})
print "Result of minimizing Rosenbrock function via Nelder-Mead
Simplex algorithm:"
print res

Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 436
         Function evaluations: 706
Result of minimizing Rosenbrock function via Nelder-Mead Simplex
 algorithm:
  status: 0
    nfev: 706
 success: True
     fun: 1.6614969876635003e-17
       x: array([ 1.,  1.,  1.,  1.,  1.])
 message: 'Optimization terminated successfully.'
     nit: 436
```

Rosenbrock函数的性质比较好，简单的优化方法就可以处理了，还可以在 `minimize` 中使用 `method='powell'` 来指定使用Powell's method。这两种简单的方法并不使用函数的梯度，在略微复杂的情形下收敛速度比较慢，下面让我们来看一下用到函数梯度进行寻优的方法。

3.1.2 Broyden-Fletcher-Goldfarb-Shanno法

Broyden-Fletcher-Goldfarb-Shanno（BFGS）法用到了梯度信息，首先求一下Rosenbrock函数的梯度：

$$\begin{aligned}\frac{\partial f}{\partial x_j} &= \sum_{i=1}^{N} 200(x_i - x_{i-1}^2)(\delta_{i,j} - 2x_{i-1}\delta_{i-1,j}) - 2(1 - x_{i-1})\delta_{i-1,j} \\ &= 200(x_j - x_{j-1}^2) - 400x_j(x_{j+1} - x_j^2) - 2(1 - x_j)\end{aligned}$$

其中当 `i=j` 时，`δi,j=1` ，否则 `δi,j=0` 。

边界的梯度是特例，有如下形式：

$$\begin{aligned}\frac{\partial f}{\partial x_0} &= -400x_0(x_1 - x_0^2) - 2(1 - x_0), \\ \frac{\partial f}{\partial x_{N-1}} &= 200(x_{N-1} - x_{N-2}^2)\end{aligned}$$

我们可以如下定义梯度向量的计算函数了：

```
def rosen_der(x):
    xm = x[1:-1]
    xm_m1 = x[:-2]
    xm_p1 = x[2:]
    der = np.zeros_like(x)
    der[1:-1] = 200*(xm-xm_m1**2) - 400*(xm_p1 - xm**2)*xm - 2*(1
-xm)
    der[0] = -400*x[0]*(x[1]-x[0]**2) - 2*(1-x[0])
    der[-1] = 200*(x[-1]-x[-2]**2)
    return der
```

梯度信息的引入在 `minimize` 函数中通过参数 `jac` 指定：

```
res = opt.minimize(rosen, x_0, method='BFGS', jac=rosen_der, opt
ions={'disp': True})
print "Result of minimizing Rosenbrock function via Broyden-Flet
cher-Goldfarb-Shanno algorithm:"
print res

Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 52
         Function evaluations: 63
         Gradient evaluations: 63
Result of minimizing Rosenbrock function via Broyden-Fletcher-Go
ldfarb-Shanno algorithm:
   status: 0
  success: True
     njev: 63
     nfev: 63
 hess_inv: array([[ 0.00726515,  0.01195827,  0.0225785 ,  0.044
60906,  0.08923649],
       [ 0.01195827,  0.02417936,  0.04591135,  0.09086889,  0.1
8165604],
       [ 0.0225785 ,  0.04591135,  0.09208689,  0.18237695,  0.3
6445491],
       [ 0.04460906,  0.09086889,  0.18237695,  0.36609277,  0.7
3152922],
       [ 0.08923649,  0.18165604,  0.36445491,  0.73152922,  1.4
6680958]])
      fun: 3.179561068096293e-14
        x: array([ 1.        ,  0.99999998,  0.99999996,  0.9999
9992,  0.99999983])
  message: 'Optimization terminated successfully.'
      jac: array([  4.47207141e-06,   1.30357917e-06,  -1.864542
07e-07,
        -2.00564982e-06,   4.98799446e-07])
```

3.1.3 牛顿共轭梯度法（Newton-Conjugate-Gradient algorithm）

用到梯度的方法还有牛顿法，牛顿法是收敛速度最快的方法，其缺点在于要求Hessian矩阵（二阶导数矩阵）。牛顿法大致的思路是采用泰勒展开的二阶近似：

$$f(\mathbf{x}) \approx f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_0)^T\mathbf{H}(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0)$$

其中 `H(x0)` 表示二阶导数矩阵。若Hessian矩阵是正定的，函数的局部最小值可以通过使上面的二次型的一阶导数等于0来获取，我们有：

$$\mathbf{x}_{\mathrm{opt}} = \mathbf{x}_0 - \mathbf{H}^{-1}\nabla f$$

这里可使用共轭梯度近似Hessian矩阵的逆矩阵。下面给出Rosenbrock函数的Hessian矩阵元素通式：

$$\begin{aligned} H_{i,j} = \frac{\partial^2 f}{\partial x_i \partial x_j} &= 200(\delta_{i,j} - 2x_{i-1}\delta_{i-1,j}) - 400x_i(\delta_{i+1,j} - 2x_i\delta_{i,j}) - 400\delta_{i,j}(x_{i+1} - x_i^2) + 2\delta_{i,j}, \\ &= (202 + 1200x_i^2 - 400x_{i+1})\delta_{i,j} - 400x_i\delta_{i+1,j} - 400x_{i-1}\delta_{i-1,j} \end{aligned}$$

其中 `i,j∈[1,N-2]` 。其他边界上的元素通式为：

$$\begin{aligned} \frac{\partial^2 f}{\partial x_0^2} &= 1200x_0^2 - 400x_1 + 2, \\ \frac{\partial^2 f}{\partial x_0 \partial x_1} = \frac{\partial^2 f}{\partial x_1 \partial x_0} &= -400x_0, \\ \frac{\partial^2 f}{\partial x_{N-1} \partial x_{N-2}} = \frac{\partial^2 f}{\partial x_{N-2} \partial x_{N-1}} &= -400x_{N-2}, \\ \frac{\partial^2 f}{\partial x_{N-1}^2} &= 200. \end{aligned}$$

例如，当 `N=5` 时的Hessian矩阵为：

$$\mathbf{H} = \begin{bmatrix} 1200x_0^2 - 400x_1 + 2 & -400x_0 & 0 & 0 & 0 \\ -400x_0 & 202 + 1200x_1^2 - 400x_2 & -400x_1 & 0 & 0 \\ 0 & -400x_1 & 202 + 1200x_2^2 - 400x_3 & -400x_2 & 0 \\ 0 & 0 & -400x_2 & 202 + 1200x_3^2 - 400x_4 & -400x_3 \\ 0 & 0 & 0 & -400x_3 & 200 \end{bmatrix}$$

为使用牛顿共轭梯度法，我们需要提供一个计算Hessian矩阵的函数：

```
def rosen_hess(x):
    x = np.asarray(x)
    H = np.diag(-400*x[:-1],1) - np.diag(400*x[:-1],-1)
    diagonal = np.zeros_like(x)
    diagonal[0] = 1200*x[0]**2-400*x[1]+2
    diagonal[-1] = 200
    diagonal[1:-1] = 202 + 1200*x[1:-1]**2 - 400*x[2:]
    H = H + np.diag(diagonal)
    return H
```

```
res = opt.minimize(rosen, x_0, method='Newton-CG', jac=rosen_der
, hess=rosen_hess, options={'xtol': 1e-8, 'disp': True})
print "Result of minimizing Rosenbrock function via Newton-Conju
gate-Gradient algorithm (Hessian):"
print res

Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 20
         Function evaluations: 22
         Gradient evaluations: 41
         Hessian evaluations: 20
Result of minimizing Rosenbrock function via Newton-Conjugate-Gr
adient algorithm:
  status: 0
 success: True
    njev: 41
    nfev: 22
     fun: 1.47606641102778e-19
       x: array([ 1.,  1.,  1.,  1.,  1.])
 message: 'Optimization terminated successfully.'
    nhev: 20
     jac: array([ -3.62847530e-11,   2.68148992e-09,   1.1663736
2e-08,
         4.81693414e-08,  -2.76999090e-08])
```

对于一些大型的优化问题，Hessian矩阵将异常大，牛顿共轭梯度法用到的仅是Hessian矩阵和一个任意向量的乘积，为此，用户可以提供两个向量，一个是Hessian矩阵和一个任意向量 `p` 的乘积，另一个是向量 `p` ，这就减少了存储的开销。记向量 `p=(p1,…,pN−1)` ，可有

$$\mathbf{H(x)p} = \begin{bmatrix} (1200x_0^2 - 400x_1 + 2)p_0 - 400x_0p_1 \\ \vdots \\ -400x_{i-1}p_{i-1} + (202 + 1200x_i^2 - 400x_{i+1})p_i - 400x_ip_{i+1} \\ \vdots \\ -400x_{N-2}p_{N-2} + 200p_{N-1} \end{bmatrix}$$

我们定义如下函数并使用牛顿共轭梯度方法寻优：

```
def rosen_hess_p(x, p):
    x = np.asarray(x)
    Hp = np.zeros_like(x)
    Hp[0] = (1200*x[0]**2 - 400*x[1] + 2)*p[0] - 400*x[0]*p[1]
    Hp[1:-1] = -400*x[:-2]*p[:-2]+(202+1200*x[1:-1]**2-400*x[2:]
)*p[1:-1] \
               -400*x[1:-1]*p[2:]
    Hp[-1] = -400*x[-2]*p[-2] + 200*p[-1]
    return Hp

res = opt.minimize(rosen, x_0, method='Newton-CG', jac=rosen_der
, hessp=rosen_hess_p, options={'xtol': 1e-8, 'disp': True})
print "Result of minimizing Rosenbrock function via Newton-Conju
gate-Gradient algorithm (Hessian times arbitrary vector):"
print res

Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 20
         Function evaluations: 22
         Gradient evaluations: 41
         Hessian evaluations: 58
Result of minimizing Rosenbrock function via Newton-Conjugate-Gr
adient algorithm (Hessian times arbitrary vector):
  status: 0
 success: True
    njev: 41
    nfev: 22
     fun: 1.47606641102778e-19
       x: array([ 1.,  1.,  1.,  1.,  1.])
 message: 'Optimization terminated successfully.'
    nhev: 58
     jac: array([ -3.62847530e-11,   2.68148992e-09,   1.1663736
2e-08,
         4.81693414e-08,  -2.76999090e-08])
```

3.2. 约束优化问题

无约束优化问题的一种标准形式为：

$$\begin{array}{ll} \text{minimize} & f(\mathbf{x}) \\ \text{subject to} & g_i(\mathbf{x}) \leq 0, \quad i = 1, \ldots, m \\ & \mathbf{A}\mathbf{x} = \mathbf{b} \end{array}$$

其中$g_0, \ldots, g_m : \mathbb{R}^n \rightarrow \mathbb{R}$为$\mathbb{R}^n$空间上的二次可微的凸函数；`A` 为 `p×n` 矩阵且秩 `rankA=p<n` 。

我们考察如下一个例子：

$$\begin{aligned} \text{minimize} \quad & f(x,y) = 2xy + 2x - x^2 - 2y^2 \\ \text{subject to} \quad & x^3 - y = 0 \\ & y - 1 \geq 0 \end{aligned}$$

定义目标函数及其导数为：

```
def func(x, sign=1.0):
    """ Objective function """
    return sign*(2*x[0]*x[1] + 2*x[0] - x[0]**2 - 2*x[1]**2)

def func_deriv(x, sign=1.0):
    """ Derivative of objective function """
    dfdx0 = sign*(-2*x[0] + 2*x[1] + 2)
    dfdx1 = sign*(2*x[0] - 4*x[1])
    return np.array([ dfdx0, dfdx1 ])
```

其中 `sign` 表示求解最小或者最大值，我们进一步定义约束条件：

```
cons = ({'type': 'eq',  'fun': lambda x: np.array([x[0]**3 - x[1
]]), 'jac': lambda x: np.array([3.0*(x[0]**2.0), -1.0])},
      {'type': 'ineq', 'fun': lambda x: np.array([x[1] - 1]), 'j
ac': lambda x: np.array([0.0, 1.0])})
```

最后我们使用SLSQP（Sequential Least SQuares Programming optimization algorithm）方法进行约束问题的求解（作为比较，同时列出了无约束优化的求解）：

```
res = opt.minimize(func, [-1.0, 1.0], args=(-1.0,), jac=func_der
iv, method='SLSQP', options={'disp': True})
print "Result of unconstrained optimization:"
print res
res = opt.minimize(func, [-1.0, 1.0], args=(-1.0,), jac=func_der
iv, constraints=cons, method='SLSQP', options={'disp': True})
print "Result of constrained optimization:"
print res

Optimization terminated successfully.    (Exit mode 0)
            Current function value: -2.0
            Iterations: 4
            Function evaluations: 5
            Gradient evaluations: 4
Result of unconstrained optimization:
  status: 0
 success: True
    njev: 4
    nfev: 5
     fun: -1.9999999999999996
       x: array([ 2.,  1.])
 message: 'Optimization terminated successfully.'
     jac: array([ -2.22044605e-16,  -0.00000000e+00,   0.0000000
0e+00])
     nit: 4
Optimization terminated successfully.    (Exit mode 0)
            Current function value: -1.00000018311
            Iterations: 9
            Function evaluations: 14
            Gradient evaluations: 9
Result of constrained optimization:
  status: 0
 success: True
    njev: 9
    nfev: 14
     fun: -1.0000001831052137
       x: array([ 1.00000009,  1.        ])
 message: 'Optimization terminated successfully.'
     jac: array([-1.99999982,  1.99999982,  0.        ])
     nit: 9
```

和统计部分一样，Python也有专门的优化扩展模块，CVXOPT（ http://cvxopt.org ）专门用于处理凸优化问题，在约束优化问题上提供了更多的备选方法。CVXOPT是著名的凸优化教材convex optimization的作者之一，加州大学洛杉矶分校Lieven Vandenberghe教授的大作，是处理优化问题的利器。

SciPy中的优化模块还有一些特殊定制的函数，专门处理能够转化为优化求解的一些问题，如方程求根、最小方差拟合等，可到SciPy优化部分的指引页面查看。

量化分析师的Python日记【第5天：数据处理的瑞士军刀pandas】

> 来源：https://uqer.io/community/share/54ffd96ef9f06c276f651aac

第一篇：基本数据结构介绍

一、Pandas介绍

终于写到了作者最想介绍，同时也是Python在数据处理方面功能最为强大的扩展模块了。在处理实际的金融数据时，一个条数据通常包含了多种类型的数据，例如，股票的代码是字符串，收盘价是浮点型，而成交量是整型等。在C++中可以实现为一个给定结构体作为单元的容器，如向量（`vector`，C++中的特定数据结构）。在Python中，`pandas`包含了高级的数据结构`Series`和`DataFrame`，使得在Python中处理数据变得非常方便、快速和简单。

`pandas`不同的版本之间存在一些不兼容性，为此，我们需要清楚使用的是哪一个版本的`pandas`。现在我们就查看一下量化实验室的`pandas`版本：

```
import pandas as pd
pd.__version__

'0.14.1'
```

`pandas`主要的两个数据结构是`Series`和`DataFrame`，随后两节将介绍如何由其他类型的数据结构得到这两种数据结构，或者自行创建这两种数据结构，我们先导入它们以及相关模块：

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

二、Pandas数据结构：`Series`

从一般意义上来讲，`Series`可以简单地被认为是一维的数组。`Series`和一维数组最主要的区别在于`Series`类型具有索引（`index`），可以和另一个编程中常见的数据结构哈希（Hash）联系起来。

2.1 创建`Series`

创建一个 `Series` 的基本格式是 `s = Series(data, index=index, name=name)` ，以下给出几个创建 `Series` 的例子。首先我们从数组创建 `Series` ：

```
a = np.random.randn(5)
print "a is an array:"
print a
s = Series(a)
print "s is a Series:"
print s

a is an array:
[-1.24962807 -0.85316907  0.13032511 -0.19088881  0.40475505]
s is a Series:
0   -1.249628
1   -0.853169
2    0.130325
3   -0.190889
4    0.404755
dtype: float64
```

可以在创建 `Series` 时添加 `index` ，并可使用 `Series.index` 查看具体的 `index` 。需要注意的一点是，当从数组创建 `Series` 时，若指定 `index` ，那么 `index` 长度要和 `data` 的长度一致：

```
s = Series(np.random.randn(5), index=['a', 'b', 'c', 'd', 'e'])
print s
s.index

a    0.509906
b   -0.764549
c    0.919338
d   -0.084712
e    1.896407
dtype: float64
Index([u'a', u'b', u'c', u'd', u'e'], dtype='object')
```

创建 `Series` 的另一个可选项是 `name` ，可指定 `Series` 的名称，可用 `Series.name` 访问。在随后的 `DataFrame` 中，每一列的列名在该列被单独取出来时就成了 `Series` 的名称：

```
s = Series(np.random.randn(5), index=['a', 'b', 'c', 'd', 'e'],
name='my_series')
print s
print s.name

a   -1.898245
b    0.172835
c    0.779262
d    0.289468
e   -0.947995
Name: my_series, dtype: float64
my_series
```

`Series` 还可以从字典（ `dict` ）创建：

```
d = {'a': 0., 'b': 1, 'c': 2}
print "d is a dict:"
print d
s = Series(d)
print "s is a Series:"
print s

d is a dict:
{'a': 0.0, 'c': 2, 'b': 1}
s is a Series:
a    0
b    1
c    2
dtype: float64
```

让我们来看看使用字典创建 `Series` 时指定 `index` 的情形（ `index` 长度不必和字典相同）：

```
Series(d, index=['b', 'c', 'd', 'a'])

b      1
c      2
d    NaN
a      0
dtype: float64
```

我们可以观察到两点：一是字典创建的 `Series` ，数据将按 `index` 的顺序重新排列；二是 `index` 长度可以和字典长度不一致，如果多了的话， `pandas` 将自动为多余的 `index` 分配 `NaN` （not a number， `pandas` 中数据缺失的标准记号)，当然 `index` 少的话就截取部分的字典内容。

如果数据就是一个单一的变量，如数字4，那么 `Series` 将重复这个变量：

```
Series(4., index=['a', 'b', 'c', 'd', 'e'])

a    4
b    4
c    4
d    4
e    4
dtype: float64
```

2.2 `Series` 数据的访问

访问 `Series` 数据可以和数组一样使用下标，也可以像字典一样使用索引，还可以使用一些条件过滤：

```
s = Series(np.random.randn(10),index=['a', 'b', 'c', 'd', 'e', '
f', 'g', 'h', 'i', 'j'])
s[0]

1.4328106520571824
```

```
s[:2]

a    1.432811
b    0.120681
dtype: float64
```

```
s[[2,0,4]]

c    0.578146
a    1.432811
e    1.327594
dtype: float64
```

```
s[['e', 'i']]

e    1.327594
i   -0.634347
dtype: float64
```

```
s[s > 0.5]

a    1.432811
c    0.578146
e    1.327594
g    1.850783
dtype: float64
```

```
'e' in s

True
```

三、Pandas数据结构：DataFrame

在使用DataFrame之前，我们说明一下DataFrame的特性。DataFrame是将数个Series按列合并而成的二维数据结构，每一列单独取出来是一个Series，这和SQL数据库中取出的数据是很类似的。所以，按列对一个DataFrame进行处理更为方便，用户在编程时注意培养按列构建数据的思维。DataFrame的优势在于可以方便地处理不同类型的列，因此，就不要考虑如何对一个全是浮点数的DataFrame求逆之类的问题了，处理这种问题还是把数据存成NumPy的matrix类型比较便利一些。

3.1 创建DataFrame

首先来看如何从字典创建DataFrame。DataFrame是一个二维的数据结构，是多个Series的集合体。我们先创建一个值是Series的字典，并转换为DataFrame：

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

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

可以指定所需的行和列，若字典中不含有对应的元素，则置为NaN：

```
df = DataFrame(d, index=['r', 'd', 'a'], columns=['two', 'three'
])
print df

   two three
r  NaN   NaN
d    4   NaN
a    1   NaN
```

可以使用 `dataframe.index` 和 `dataframe.columns` 来查看 `DataFrame` 的行和列， `dataframe.values` 则以数组的形式返回 `DataFrame` 的元素：

```
print "DataFrame index:"
print df.index
print "DataFrame columns:"
print df.columns
print "DataFrame values:"
print df.values

DataFrame index:
Index([u'alpha', u'beta', u'gamma', u'delta', u'eta'], dtype='ob
ject')
DataFrame columns:
Index([u'a', u'b', u'c', u'd', u'e'], dtype='object')
DataFrame values:
[[  0.   0.   0.   0.   0.]
 [  1.   2.   3.   4.   5.]
 [  2.   4.   6.   8.  10.]
 [  3.   6.   9.  12.  15.]
 [  4.   8.  12.  16.  20.]]
```

`DataFrame` 也可以从值是数组的字典创建，但是各个数组的长度需要相同：

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

   one  two
a    1    4
b    2    3
c    3    2
d    4    1
```

值非数组时，没有这一限制，并且缺失值补成 `NaN` ：

```
d= [{'a': 1.6, 'b': 2}, {'a': 3, 'b': 6, 'c': 9}]
df = DataFrame(d)
print df

     a  b   c
0  1.6  2 NaN
1  3.0  6   9
```

在实际处理数据时，有时需要创建一个空的 `DataFrame` ，可以这么做：

```
df = DataFrame()
print df

Empty DataFrame
Columns: []
Index: []
```

另一种创建 `DataFrame` 的方法十分有用，那就是使用 `concat` 函数基于 `Series` 或者 `DataFrame` 创建一个 `DataFrame`

```
a = Series(range(5))
b = Series(np.linspace(4, 20, 5))
df = pd.concat([a, b], axis=1)
print df

   0   1
0  0   4
1  1   8
2  2  12
3  3  16
4  4  20
```

其中的 `axis=1` 表示按列进行合并， `axis=0` 表示按行合并，并且， `Series` 都处理成一列，所以这里如果选 `axis=0` 的话，将得到一个 `10×1` 的 `DataFrame` 。下面这个例子展示了如何按行合并 `DataFrame` 成一个大的 `DataFrame` ：

```
df = DataFrame()
index = ['alpha', 'beta', 'gamma', 'delta', 'eta']
for i in range(5):
    a = DataFrame([np.linspace(i, 5*i, 5)], index=[index[i]])
    df = pd.concat([df, a], axis=0)
print df

       0  1   2   3   4
alpha  0  0   0   0   0
beta   1  2   3   4   5
gamma  2  4   6   8  10
delta  3  6   9  12  15
eta    4  8  12  16  20
```

3.2 `DataFrame` 数据的访问

首先，再次强调一下 `DataFrame` 是以列作为操作的基础的，全部操作都想象成先从 `DataFrame` 里取一列，再从这个 `Series` 取元素即可。可以用 `dataffrae.column_name` 选取列，也可以使用 `dataframe[]` 操作选取列，我们可以马上发现前一种方法只能选取一列，而后一种方法可以选择多列。若 `DataFrame` 没有列名， `[]` 可以使用非负整数，也就是“下标”选取列；若有列名，则必须使用列名选取，另外 `datafrae.column_name` 在没有列名的时候是无效的：

```
print df[1]
print type(df[1])
df.columns = ['a', 'b', 'c', 'd', 'e']
print df['b']
print type(df['b'])
print df.b
print type(df.b)
print df[['a', 'd']]
print type(df[['a', 'd']])

alpha    0
beta     2
gamma    4
delta    6
eta      8
Name: 1, dtype: float64
<class 'pandas.core.series.Series'>
alpha    0
beta     2
gamma    4
delta    6
eta      8
Name: b, dtype: float64
<class 'pandas.core.series.Series'>
alpha    0
beta     2
gamma    4
delta    6
eta      8
Name: b, dtype: float64
<class 'pandas.core.series.Series'>
       a   d
alpha  0   0
beta   1   4
gamma  2   8
delta  3  12
eta    4  16
<class 'pandas.core.frame.DataFrame'>
```

以上代码使用了 `dataframe.columns` 为 `DataFrame` 赋列名，并且我们看到单独取一列出来，其数据结构显示的是 `Series` ，取两列及两列以上的结果仍然是 `DataFrame` 。访问特定的元素可以如 `Series` 一样使用下标或者是索引:

```
print df['b'][2]
print df['b']['gamma']

4.0
4.0
```

若需要选取行，可以使用 `dataframe.iloc` 按下标选取，或者使用 `dataframe.loc` 按索引选取：

```
print df.iloc[1]
print df.loc['beta']

a    1
b    2
c    3
d    4
e    5
Name: beta, dtype: float64
a    1
b    2
c    3
d    4
e    5
Name: beta, dtype: float64
```

选取行还可以使用切片的方式或者是布尔类型的向量：

```
print "Selecting by slices:"
print df[1:3]
bool_vec = [True, False, True, True, False]
print "Selecting by boolean vector:"
print df[bool_vec]

Selecting by slices:
       a  b  c  d   e
beta   1  2  3  4   5
gamma  2  4  6  8  10
Selecting by boolean vector:
       a  b  c   d   e
alpha  0  0  0   0   0
gamma  2  4  6   8  10
delta  3  6  9  12  15
```

行列组合起来选取数据：

```
print df[['b', 'd']].iloc[[1, 3]]
print df.iloc[[1, 3]][['b', 'd']]
print df[['b', 'd']].loc[['beta', 'delta']]
print df.loc[['beta', 'delta']][['b', 'd']]

       b   d
beta   2   4
delta  6  12
       b   d
beta   2   4
delta  6  12
       b   d
beta   2   4
delta  6  12
       b   d
beta   2   4
delta  6  12
```

如果不是需要访问特定行列，而只是某个特殊位置的元素的话， `dataframe.at` 和 `dataframe.iat` 是最快的方式，它们分别用于使用索引和下标进行访问：

```
print df.iat[2, 3]
print df.at['gamma', 'd']

8.0
8.0
```

`dataframe.ix` 可以混合使用索引和下标进行访问，唯一需要注意的地方是行列内部需要一致，不可以同时使用索引和标签访问行或者列，不然的话，将会得到意外的结果：

```
print df.ix['gamma', 4]
print df.ix[['delta', 'gamma'], [1, 4]]
print df.ix[[1, 2], ['b', 'e']]
print "Unwanted result:"
print df.ix[['beta', 2], ['b', 'e']]
print df.ix[[1, 2], ['b', 4]]

10.0
       b   e
delta  6  15
gamma  4  10
       b   e
beta   2   5
gamma  4  10
Unwanted result:
       b   e
beta   2   5
2    NaN NaN
       b   4
beta   2 NaN
gamma  4 NaN
```

量化分析师的Python日记【第6天：数据处理的瑞士军刀pandas下篇

> 来源：https://uqer.io/community/share/5514bb11f9f06c12790415b2

第二篇：快速进阶

在上一篇中我们介绍了如何创建并访问 `pandas` 的 `Series` 和 `DataFrame` 型的数据，本篇将介绍如何对 `pandas` 数据进行操作，掌握这些操作之后，基本可以处理大多数的数据了。首先，导入本篇中使用到的模块：

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

为了看数据方便一些，我们设置一下输出屏幕的宽度

```
pd.set_option('display.width', 200)
```

一、数据创建的其他方式

数据结构的创建不止是上篇中介绍的标准形式，本篇再介绍几种。例如，我们可以创建一个以日期为元素的 `Series` ：

```
dates = pd.date_range('20150101', periods=5)
print dates

<class 'pandas.tseries.index.DatetimeIndex'>
[2015-01-01, ..., 2015-01-05]
Length: 5, Freq: D, Timezone: None
```

将这个日期 `Series` 作为索引赋给一个 `DataFrame` ：

```
df = pd.DataFrame(np.random.randn(5, 4),index=dates,columns=list('ABCD'))
print df

                   A         B         C         D
2015-01-01 -0.168870  0.191945 -0.906788 -1.295211
2015-01-02 -0.985849  0.312378 -1.513870 -0.876869
2015-01-03 -0.241945  1.437763  0.209494  0.061032
2015-01-04  0.139199  0.124118 -0.204801 -1.745327
2015-01-05  0.243644 -0.373126  0.333583  2.640955
```

只要是能转换成`Series`的对象，都可以用于创建`DataFrame`：

```
df2 = pd.DataFrame({ 'A' : 1., 'B': pd.Timestamp('20150214'), 'C' : pd.Series(1.6,index=list(range(4)),dtype='float64'), 'D' : np.array([4] * 4, dtype='int64'), 'E' : 'hello pandas!' })
print df2

   A          B    C  D              E
0  1 2015-02-14  1.6  4  hello pandas!
1  1 2015-02-14  1.6  4  hello pandas!
2  1 2015-02-14  1.6  4  hello pandas!
3  1 2015-02-14  1.6  4  hello pandas!
```

二、数据的查看

在多数情况下，数据并不由分析数据的人员生成，而是通过数据接口、外部文件或者其他方式获取。这里我们通过量化实验室的数据接口获取一份数据作为示例：

```
stock_list = ['000001.XSHE', '000002.XSHE', '000568.XSHE', '000625.XSHE', '000768.XSHE', '600028.XSHG', '600030.XSHG', '601111.XSHG', '601390.XSHG', '601998.XSHG']
raw_data = DataAPI.MktEqudGet(secID=stock_list, beginDate='20150101', endDate='20150131', pandas='1')
df = raw_data[['secID', 'tradeDate', 'secShortName', 'openPrice', 'highestPrice', 'lowestPrice', 'closePrice', 'turnoverVol']]
```

以上代码获取了2015年一月份全部的交易日内十支股票的日行情信息，首先我们来看一下数据的大小：

```
print df.shape

(200, 8)
```

我们可以看到有200行，表示我们获取到了200条记录，每条记录有8个字段，现在预览一下数据， `dataframe.head()` 和 `dataframe.tail()` 可以查看数据的头五行和尾五行，若需要改变行数，可在括号内指定：

```
print "Head of this DataFrame:"
print df.head()
print "Tail of this DataFrame:"
print df.tail(3)

Head of this DataFrame:
         secID   tradeDate secShortName  openPrice  highestPrice
  lowestPrice  closePrice  turnoverVol
0  000001.XSHE  2015-01-05         平安银行      15.99         16
.28         15.60        16.02    286043643
1  000001.XSHE  2015-01-06         平安银行      15.85         16
.39         15.55        15.78    216642140
2  000001.XSHE  2015-01-07         平安银行      15.56         15
.83         15.30        15.48    170012067
3  000001.XSHE  2015-01-08         平安银行      15.50         15
.57         14.90        14.96    140771421
4  000001.XSHE  2015-01-09         平安银行      14.90         15
.87         14.71        15.08    250850023
Tail of this DataFrame:
           secID   tradeDate secShortName  openPrice  highestPri
ce  lowestPrice  closePrice  turnoverVol
197  601998.XSHG  2015-01-28         中信银行       7.04
7.32         6.95        7.15     163146128
198  601998.XSHG  2015-01-29         中信银行       6.97
7.05         6.90        7.01      93003445
199  601998.XSHG  2015-01-30         中信银行       7.10
7.14         6.92        6.95      68146718
```

`dataframe.describe()` 提供了 `DataFrame` 中纯数值数据的统计信息：

```
print df.describe()

       openPrice  highestPrice  lowestPrice  closePrice   turnov
erVol
count  200.00000    200.000000    200.00000  200.000000  2.00000
0e+02
mean    15.17095     15.634000     14.86545   15.242750  2.38481
1e+08
std      7.72807      7.997345      7.56136    7.772184  2.33051
0e+08
min      6.14000      6.170000      6.02000    6.030000  1.24218
3e+07
25%      8.09500      8.250000      7.98750    8.127500  7.35700
2e+07
50%     13.96000     14.335000     13.75500   13.925000  1.55456
9e+08
75%     19.95000     20.500000     19.46250   20.012500  3.35861
7e+08
max     36.40000     37.250000     34.68000   36.150000  1.31085
5e+09
```

对数据的排序将便利我们观察数据， `DataFrame` 提供了两种形式的排序。一种是按行列排序，即按照索引（行名）或者列名进行排序，可调用 `dataframe.sort_index` ，指定 `axis=0` 表示按索引（行名）排序， `axis=1` 表示按列名排序，并可指定升序或者降序：

```
print "Order by column names, descending:"
print df.sort_index(axis=1, ascending=False).head()

Order by column names, descending:
   turnoverVol   tradeDate secShortName        secID  openPrice
 lowestPrice  highestPrice  closePrice
0    286043643  2015-01-05         平安银行  000001.XSHE      15.
99         15.60          16.28        16.02
1    216642140  2015-01-06         平安银行  000001.XSHE      15.
85         15.55          16.39        15.78
2    170012067  2015-01-07         平安银行  000001.XSHE      15.
56         15.30          15.83        15.48
3    140771421  2015-01-08         平安银行  000001.XSHE      15.
50         14.90          15.57        14.96
4    250850023  2015-01-09         平安银行  000001.XSHE      14.
90         14.71          15.87        15.08
```

第二种排序是按值排序，可指定列名和排序方式，默认的是升序排序：

```
print "Order by column value, ascending:"
print df.sort(columns='tradeDate').head()
print "Order by multiple columns value:"
df = df.sort(columns=['tradeDate', 'secID'], ascending=[False, T
rue])
print df.head()

Order by column value, ascending:
          secID   tradeDate secShortName  openPrice  highestPric
e  lowestPrice  closePrice  turnoverVol
0   000001.XSHE  2015-01-05         平安银行      15.99          1
6.28         15.60        16.02     286043643
20  000002.XSHE  2015-01-05          万科A      14.39          15.
29          14.22        14.91     656083570
40  000568.XSHE  2015-01-05         泸州老窖      20.50          2
1.99         20.32        21.90      59304755
60  000625.XSHE  2015-01-05         长安汽车      16.40          1
8.07         16.32        18.07      82087982
80  000768.XSHE  2015-01-05         中航飞机      18.76          1
9.88         18.41        19.33      84199357
Order by multiple columns value:
          secID   tradeDate secShortName  openPrice  highestPric
e  lowestPrice  closePrice  turnoverVol
19  000001.XSHE  2015-01-30         平安银行      13.93          1
4.12         13.76        13.93      93011669
39  000002.XSHE  2015-01-30          万科A      13.09          13.
49          12.80        13.12     209624706
59  000568.XSHE  2015-01-30         泸州老窖      19.15          1
9.51         19.11        19.12      14177179
79  000625.XSHE  2015-01-30         长安汽车      19.16          1
9.45         18.92        19.18      21233495
99  000768.XSHE  2015-01-30         中航飞机      25.38          2
5.65         24.28        24.60      59550293
```

三、数据的访问和操作

3.1 再谈数据的访问

上篇中已经介绍了使用 `loc` 、 `iloc` 、 `at` 、 `iat` 、 `ix` 以及 `[]` 访问 `DataFrame` 数据的几种方式，这里再介绍一种方法，使用 `:` 来获取部行或者全部列：

```
print df.iloc[1:4][:]

          secID   tradeDate secShortName  openPrice  highestPric
e  lowestPrice  closePrice  turnoverVol
39  000002.XSHE  2015-01-30          万科A      13.09         13.
49        12.80       13.12    209624706
59  000568.XSHE  2015-01-30         泸州老窖      19.15          1
9.51        19.11       19.12     14177179
79  000625.XSHE  2015-01-30         长安汽车      19.16          1
9.45        18.92       19.18     21233495
```

我们可以扩展上篇介绍的使用布尔类型的向量获取数据的方法，可以很方便地过滤数据，例如，我们要选出收盘价在均值以上的数据：

```
print df[df.closePrice > df.closePrice.mean()].head()

           secID   tradeDate secShortName  openPrice  highestPri
ce  lowestPrice  closePrice  turnoverVol
59   000568.XSHE  2015-01-30         泸州老窖      19.15
19.51        19.11       19.12     14177179
79   000625.XSHE  2015-01-30         长安汽车      19.16
19.45        18.92       19.18     21233495
99   000768.XSHE  2015-01-30         中航飞机      25.38
25.65        24.28       24.60     59550293
139  600030.XSHG  2015-01-30         中信证券      28.50
28.72        27.78       27.86    304218245
58   000568.XSHE  2015-01-29         泸州老窖      19.04
19.23        19.00       19.15     12421826
```

`isin()` 函数可方便地过滤 `DataFrame` 中的数据：

```
print df[df['secID'].isin(['601628.XSHG', '000001.XSHE', '600030
.XSHG'])].head()
print df.shape

           secID   tradeDate secShortName  openPrice  highestPri
ce  lowestPrice  closePrice  turnoverVol
19   000001.XSHE  2015-01-30         平安银行      13.93
14.12        13.76       13.93     93011669
139  600030.XSHG  2015-01-30         中信证券      28.50
28.72        27.78       27.86    304218245
18   000001.XSHE  2015-01-29         平安银行      13.82
14.01        13.75       13.90    101675329
138  600030.XSHG  2015-01-29         中信证券      28.10
28.58        27.81       28.18    386310957
17   000001.XSHE  2015-01-28         平安银行      13.87
14.30        13.80       14.06    124087755
(200, 8)
```

3.2 处理缺失数据

在访问数据的基础上，我们可以更改数据，例如，修改某些元素为缺失值：

```
df['openPrice'][df['secID'] == '000001.XSHE'] = np.nan
df['highestPrice'][df['secID'] == '601111.XSHG'] = np.nan
df['lowestPrice'][df['secID'] == '601111.XSHG'] = np.nan
df['closePrice'][df['secID'] == '000002.XSHE'] = np.nan
df['turnoverVol'][df['secID'] == '601111.XSHG'] = np.nan
print df.head(10)
```

```
           secID   tradeDate secShortName  openPrice  highestPri
ce  lowestPrice  closePrice  turnoverVol
19   000001.XSHE  2015-01-30         平安银行        NaN
14.12        13.76       13.93     93011669
39   000002.XSHE  2015-01-30          万科A      13.09         13
.49       12.80         NaN    209624706
59   000568.XSHE  2015-01-30         泸州老窖      19.15
19.51        19.11       19.12     14177179
79   000625.XSHE  2015-01-30         长安汽车      19.16
19.45        18.92       19.18     21233495
99   000768.XSHE  2015-01-30         中航飞机      25.38
25.65        24.28       24.60     59550293
119  600028.XSHG  2015-01-30         中国石化       6.14
6.17         6.02        6.03    502445638
139  600030.XSHG  2015-01-30         中信证券      28.50
28.72        27.78       27.86    304218245
159  601111.XSHG  2015-01-30         中国国航       7.92
  NaN          NaN        7.69          NaN
179  601390.XSHG  2015-01-30         中国中铁       8.69
8.69         8.12        8.14    352357431
199  601998.XSHG  2015-01-30         中信银行       7.10
7.14         6.92        6.95     68146718
```

原始数据的中很可能存在一些数据的缺失，就如同现在处理的这个样例数据一样，处理缺失数据有多种方式。通常使用 `dataframe.dropna()` ， `dataframe.dropna()` 可以按行丢弃带有 `nan` 的数据；若指定 `how='all'` （默认是 `'any'` ），则只在整行全部是 `nan` 时丢弃数据；若指定 `thresh` ，则表示当某行数据非缺失列数超过指定数值时才保留；要指定根据某列丢弃可以通过 `subset` 完成。

```
print "Data size before filtering:"
print df.shape

print "Drop all rows that have any NaN values:"
print "Data size after filtering:"
print df.dropna().shape
print df.dropna().head(10)
print "Drop only if all columns are NaN:"
print "Data size after filtering:"
print df.dropna(how='all').shape
print df.dropna(how='all').head(10)

print "Drop rows who do not have at least six values that are no
t NaN"
print "Data size after filtering:"
print df.dropna(thresh=6).shape
print df.dropna(thresh=6).head(10)

print "Drop only if NaN in specific column:"
print "Data size after filtering:"
print df.dropna(subset=['closePrice']).shape
print df.dropna(subset=['closePrice']).head(10)

Data size before filtering:
(200, 8)
Drop all rows that have any NaN values:
Data size after filtering:
(140, 8)
           secID   tradeDate secShortName  openPrice  highestPri
ce  lowestPrice  closePrice  turnoverVol
59   000568.XSHE  2015-01-30         泸州老窖      19.15
19.51        19.11       19.12     14177179
79   000625.XSHE  2015-01-30         长安汽车      19.16
19.45        18.92       19.18     21233495
99   000768.XSHE  2015-01-30         中航飞机      25.38
25.65        24.28       24.60     59550293
119  600028.XSHG  2015-01-30         中国石化       6.14
6.17         6.02        6.03    502445638
139  600030.XSHG  2015-01-30         中信证券      28.50
28.72        27.78       27.86    304218245
179  601390.XSHG  2015-01-30         中国中铁       8.69
8.69         8.12        8.14    352357431
199  601998.XSHG  2015-01-30         中信银行       7.10
7.14         6.92        6.95     68146718
58   000568.XSHE  2015-01-29         泸州老窖      19.04
19.23        19.00       19.15     12421826
78   000625.XSHE  2015-01-29         长安汽车      19.60
19.64        18.90       19.24     25546060
98   000768.XSHE  2015-01-29         中航飞机      24.65
25.63        24.53       24.98     67095945
Drop only if all columns are NaN:
Data size after filtering:
(200, 8)
           secID   tradeDate secShortName  openPrice  highestPri
ce  lowestPrice  closePrice  turnoverVol
19   000001.XSHE  2015-01-30         平安银行        NaN
14.12        13.76       13.93     93011669
39   000002.XSHE  2015-01-30          万科A      13.09         13
.49        12.80         NaN    209624706
59   000568.XSHE  2015-01-30         泸州老窖      19.15
```

```
19.51        19.11       19.12     14177179
79   000625.XSHE  2015-01-30         长安汽车       19.16
19.45        18.92       19.18     21233495
99   000768.XSHE  2015-01-30         中航飞机       25.38
25.65        24.28       24.60     59550293
119  600028.XSHG  2015-01-30         中国石化        6.14
6.17         6.02        6.03     502445638
139  600030.XSHG  2015-01-30         中信证券       28.50
28.72        27.78       27.86    304218245
159  601111.XSHG  2015-01-30         中国国航        7.92
  NaN          NaN         7.69             NaN
179  601390.XSHG  2015-01-30         中国中铁        8.69
8.69         8.12        8.14     352357431
199  601998.XSHG  2015-01-30         中信银行        7.10
7.14         6.92        6.95      68146718
Drop rows who do not have at least six values that are not NaN
Data size after filtering:
(180, 8)
           secID   tradeDate secShortName  openPrice  highestPri
ce  lowestPrice  closePrice  turnoverVol
19   000001.XSHE  2015-01-30         平安银行         NaN
14.12        13.76       13.93     93011669
39   000002.XSHE  2015-01-30          万科A       13.09          13
.49        12.80         NaN     209624706
59   000568.XSHE  2015-01-30         泸州老窖       19.15
19.51        19.11       19.12     14177179
79   000625.XSHE  2015-01-30         长安汽车       19.16
19.45        18.92       19.18     21233495
99   000768.XSHE  2015-01-30         中航飞机       25.38
25.65        24.28       24.60     59550293
119  600028.XSHG  2015-01-30         中国石化        6.14
6.17         6.02        6.03     502445638
139  600030.XSHG  2015-01-30         中信证券       28.50
28.72        27.78       27.86    304218245
179  601390.XSHG  2015-01-30         中国中铁        8.69
8.69         8.12        8.14     352357431
199  601998.XSHG  2015-01-30         中信银行        7.10
7.14         6.92        6.95      68146718
18   000001.XSHE  2015-01-29         平安银行         NaN
14.01        13.75       13.90    101675329
Drop only if NaN in specific column:
Data size after filtering:
(180, 8)
           secID   tradeDate secShortName  openPrice  highestPri
ce  lowestPrice  closePrice  turnoverVol
19   000001.XSHE  2015-01-30         平安银行         NaN
14.12        13.76       13.93     93011669
59   000568.XSHE  2015-01-30         泸州老窖       19.15
19.51        19.11       19.12     14177179
79   000625.XSHE  2015-01-30         长安汽车       19.16
19.45        18.92       19.18     21233495
99   000768.XSHE  2015-01-30         中航飞机       25.38
25.65        24.28       24.60     59550293
```

```
119   600028.XSHG   2015-01-30         中国石化       6.14
6.17          6.02         6.03     502445638
139   600030.XSHG   2015-01-30         中信证券      28.50
28.72         27.78        27.86     304218245
159   601111.XSHG   2015-01-30         中国国航       7.92
  NaN           NaN         7.69           NaN
179   601390.XSHG   2015-01-30         中国中铁       8.69
8.69          8.12         8.14     352357431
199   601998.XSHG   2015-01-30         中信银行       7.10
7.14          6.92         6.95      68146718
18    000001.XSHE   2015-01-29         平安银行        NaN
14.01         13.75        13.90     101675329
```

有数据缺失时也未必是全部丢弃，`dataframe.fillna(value=value)`可以指定填补缺失值的数值

```
print df.fillna(value=20150101).head()

          secID   tradeDate secShortName    openPrice  highestPr
ice  lowestPrice   closePrice  turnoverVol
19  000001.XSHE  2015-01-30         平安银行  20150101.00
14.12        13.76        13.93     93011669
39  000002.XSHE  2015-01-30          万科A        13.09        1
3.49        12.80  20150101.00    209624706
59  000568.XSHE  2015-01-30         泸州老窖        19.15
19.51        19.11        19.12     14177179
79  000625.XSHE  2015-01-30         长安汽车        19.16
19.45        18.92        19.18     21233495
99  000768.XSHE  2015-01-30         中航飞机        25.38
25.65        24.28        24.60     59550293
```

3.3 数据操作

`Series`和`DataFrame`的类函数提供了一些函数，如`mean()`、`sum()`等，指定0按列进行，指定1按行进行：

```
df = raw_data[['secID', 'tradeDate', 'secShortName', 'openPrice'
, 'highestPrice', 'lowestPrice', 'closePrice', 'turnoverVol']]
print df.mean(0)

openPrice       1.517095e+01
highestPrice    1.563400e+01
lowestPrice     1.486545e+01
closePrice      1.524275e+01
turnoverVol     2.384811e+08
dtype: float64
```

value_counts 函数可以方便地统计频数：

```
print df['closePrice'].value_counts().head()

6.58     3
13.12    2
9.13     2
8.58     2
6.93     2
dtype: int64
```

在 panda 中， Series 可以调用 map 函数来对每个元素应用一个函数， DataFrame 可以调用 apply 函数对每一列（行）应用一个函数， applymap 对每个元素应用一个函数。这里面的函数可以是用户自定义的一个lambda函数，也可以是已有的其他函数。下例展示了将收盘价调整到 [0, 1] 区间：

```
print df[['closePrice']].apply(lambda x: (x - x.min()) / (x.max(
) - x.min())).head()

   closePrice
0    0.331673
1    0.323705
2    0.313745
3    0.296481
4    0.300465
```

使用 append 可以在 Series 后添加元素，以及在 DataFrame 尾部添加一行：

```
dat1 = df[['secID', 'tradeDate', 'closePrice']].head()
dat2 = df[['secID', 'tradeDate', 'closePrice']].iloc[2]
print "Before appending:"
print dat1
dat = dat1.append(dat2, ignore_index=True)
print "After appending:"
print dat

Before appending:
         secID   tradeDate  closePrice
0  000001.XSHE  2015-01-05       16.02
1  000001.XSHE  2015-01-06       15.78
2  000001.XSHE  2015-01-07       15.48
3  000001.XSHE  2015-01-08       14.96
4  000001.XSHE  2015-01-09       15.08
After appending:
         secID   tradeDate  closePrice
0  000001.XSHE  2015-01-05       16.02
1  000001.XSHE  2015-01-06       15.78
2  000001.XSHE  2015-01-07       15.48
3  000001.XSHE  2015-01-08       14.96
4  000001.XSHE  2015-01-09       15.08
5  000001.XSHE  2015-01-07       15.48
```

`DataFrame` 可以像在SQL中一样进行合并，在上篇中，我们介绍了使用 `concat` 函数创建 `DataFrame` ，这就是一种合并的方式。另外一种方式使用 `merge` 函数，需要指定依照哪些列进行合并，下例展示了如何根据security ID和交易日合并数据：

```
dat1 = df[['secID', 'tradeDate', 'closePrice']]
dat2 = df[['secID', 'tradeDate', 'turnoverVol']]
dat = dat1.merge(dat2, on=['secID', 'tradeDate'])
print "The first DataFrame:"
print dat1.head()
print "The second DataFrame:"
print dat2.head()
print "Merged DataFrame:"
print dat.head()

The first DataFrame:
         secID   tradeDate  closePrice
0  000001.XSHE  2015-01-05       16.02
1  000001.XSHE  2015-01-06       15.78
2  000001.XSHE  2015-01-07       15.48
3  000001.XSHE  2015-01-08       14.96
4  000001.XSHE  2015-01-09       15.08
The second DataFrame:
         secID   tradeDate  turnoverVol
0  000001.XSHE  2015-01-05    286043643
1  000001.XSHE  2015-01-06    216642140
2  000001.XSHE  2015-01-07    170012067
3  000001.XSHE  2015-01-08    140771421
4  000001.XSHE  2015-01-09    250850023
Merged DataFrame:
         secID   tradeDate  closePrice  turnoverVol
0  000001.XSHE  2015-01-05       16.02    286043643
1  000001.XSHE  2015-01-06       15.78    216642140
2  000001.XSHE  2015-01-07       15.48    170012067
3  000001.XSHE  2015-01-08       14.96    140771421
4  000001.XSHE  2015-01-09       15.08    250850023
```

`DataFrame` 另一个强大的函数是 `groupby` ，可以十分方便地对数据分组处理，我们对2015年一月内十支股票的开盘价，最高价，最低价，收盘价和成交量求平均值：

```
df_grp = df.groupby('secID')
grp_mean = df_grp.mean()
print grp_mean

             openPrice  highestPrice  lowestPrice  closePrice  t
urnoverVol
secID

000001.XSHE    14.6550       14.9840      14.4330     14.6650
154710615
000002.XSHE    13.3815       13.7530      13.0575     13.4100
277459431
000568.XSHE    19.7220       20.1015      19.4990     19.7935
  29199107
000625.XSHE    19.4915       20.2275      19.1040     19.7170
  42633332
000768.XSHE    22.4345       23.4625      21.8830     22.6905
  92781199
600028.XSHG     6.6060        6.7885       6.4715      6.6240
531966632
600030.XSHG    31.1505       32.0825      30.4950     31.2325
611544509
601111.XSHG     8.4320        8.6520       8.2330      8.4505
104143358
601390.XSHG     8.4060        8.6625       8.2005      8.4100
362831455
601998.XSHG     7.4305        7.6260       7.2780      7.4345
177541066
```

如果希望取每只股票的最新数据，应该怎么操作呢？ `drop_duplicates` 可以实现这个功能，首先对数据按日期排序，再按security ID去重：

```
df2 = df.sort(columns=['secID', 'tradeDate'], ascending=[True, F
alse])
print df2.drop_duplicates(subset='secID')

           secID   tradeDate secShortName  openPrice  highestPri
ce  lowestPrice  closePrice  turnoverVol
19   000001.XSHE  2015-01-30         平安银行      13.93
14.12        13.76       13.93     93011669
39   000002.XSHE  2015-01-30          万科A      13.09         13
.49        12.80       13.12    209624706
59   000568.XSHE  2015-01-30         泸州老窖      19.15
19.51        19.11       19.12     14177179
79   000625.XSHE  2015-01-30         长安汽车      19.16
19.45        18.92       19.18     21233495
99   000768.XSHE  2015-01-30         中航飞机      25.38
25.65        24.28       24.60     59550293
119  600028.XSHG  2015-01-30         中国石化       6.14
6.17         6.02        6.03    502445638
139  600030.XSHG  2015-01-30         中信证券      28.50
28.72        27.78       27.86    304218245
159  601111.XSHG  2015-01-30         中国国航       7.92
8.03         7.65        7.69     61877792
179  601390.XSHG  2015-01-30         中国中铁       8.69
8.69         8.12        8.14    352357431
199  601998.XSHG  2015-01-30         中信银行       7.10
7.14         6.92        6.95     68146718
```

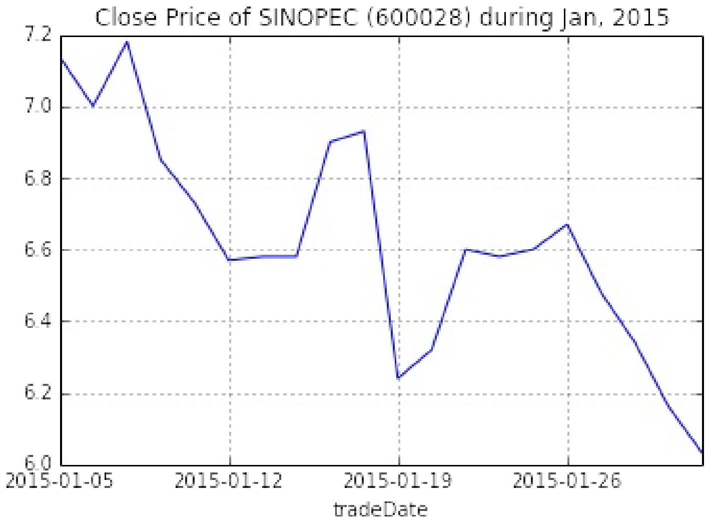
若想要保留最老的数据，可以在降序排列后取最后一个记录，通过指定 take_last=True （默认值为 False ，取第一条记录）可以实现：

```
print df2.drop_duplicates(subset='secID', take_last=True)

           secID   tradeDate secShortName  openPrice  highestPri
ce  lowestPrice  closePrice  turnoverVol
0    000001.XSHE  2015-01-05         平安银行      15.99
16.28        15.60       16.02    286043643
20   000002.XSHE  2015-01-05          万科A      14.39         15
.29         14.22       14.91    656083570
40   000568.XSHE  2015-01-05         泸州老窖      20.50
21.99        20.32       21.90     59304755
60   000625.XSHE  2015-01-05         长安汽车      16.40
18.07        16.32       18.07     82087982
80   000768.XSHE  2015-01-05         中航飞机      18.76
19.88        18.41       19.33     84199357
100  600028.XSHG  2015-01-05         中国石化       6.59
7.14         6.45        7.14   1186499645
120  600030.XSHG  2015-01-05         中信证券      33.90
35.25        33.01       34.66    698627215
140  601111.XSHG  2015-01-05         中国国航       7.98
8.62         7.98        8.62    231611758
160  601390.XSHG  2015-01-05         中国中铁       9.37
9.37         8.90        9.13    469902172
180  601998.XSHG  2015-01-05         中信银行       8.15
8.33         7.91        8.16    337368242
```

四、数据可视化

pandas 数据直接可以绘图查看，下例中我们采用中国石化一月的收盘价进行绘图，其中 set_index('tradeDate')['closePrice'] 表示将 DataFrame 的 'tradeDate' 这一列作为索引，将 'closePrice' 这一列作为 Series 的值，返回一个 Series 对象，随后调用 plot 函数绘图，更多的参数可以在 matplotlib 的文档中查看。

```
dat = df[df['secID'] == '600028.XSHG'].set_index('tradeDate')['c
losePrice']
dat.plot(title="Close Price of SINOPEC (600028) during Jan, 2015"
)

<matplotlib.axes.AxesSubplot at 0x49b6510>
```


Close Price of SINOPEC (600028) during Jan, 2015
7.2
7.0
6.8
6.6
6.4
6.2
6.0
2015-01-05
2015-01-12
2015-01-19
2015-01-26
tradeDate

量化分析师的Python日记【第7天：Q Quant 之初出江湖】

> 来源：https://uqer.io/community/share/5514fc98f9f06c8f33904449
>
> 通过前几日的学习，我们已经熟悉了Python中一些常用数值计算库的用法。本篇中，作为Quant中的Q宗（P Quant 和 Q Quant 到底哪个是未来？），我们将尝试把之前的介绍的工具串联起来，小试牛刀。
>
> 您将可以体验到：
>
> 1. 如何使用python内置的数学函数计算期权的价格；
> 2. 利用 `numpy` 加速数值计算；
> 3. 利用 `scipy` 进行仿真模拟；
> 4. 使用 `scipy` 求解器计算隐含波动率；
>
> 穿插着，我们也会使用 `matplotlib` 绘制精美的图标。

1. 关心的问题

我们想知道下面的一只期权的价格：

- 当前价 `spot` : 2.45
- 行权价 `strike` : 2.50
- 到期期限 `maturity` : 0.25
- 无风险利率 `r` : 0.05
- 波动率 `vol` : 0.25

关于这样的简单欧式期权的定价，有经典的Black - Scholes [1] 公式：

$$\mathrm{Call}(S,K,r,\tau,\sigma) = SN(d_1) - Ke^{-r\tau}N(d_2),$$

$$d_1 = \frac{\ln(S/K) + (r + \frac{1}{2}\sigma^2)\tau}{\sigma\sqrt{\tau}},$$

$$d_2 = d_1 - \sigma\sqrt{\tau}.$$

其中 `S` 为标的价格， `K` 为执行价格， `r` 为无风险利率， `τ=T-t` 为剩余到期时间。 `N(x)` 为标准正态分布的累积概率密度函数。 `Call(S,K,r,τ,σ)` 为看涨期权的价格。

```
# 参数
spot = 2.45
strike = 2.50
maturity = 0.25
r = 0.05
vol = 0.25
```

观察上面的公式，需要使用一些数学函数，我们把它分为两部分：

- `log`，`sqrt`，`exp`，这三个函数我们可以从标准库`math`中找到
- 标准正态分布的累计概率密度函数，我们使用`scipy`库中的`stats.norm.cdf`函数

```
# 基于Black - Scholes 公式的期权定价公式
from math import log, sqrt, exp
from scipy.stats import norm

def call_option_pricer(spot, strike, maturity, r, vol):

    d1 = (log(spot/strike) + (r + 0.5 * vol *vol) * maturity) / vol / sqrt(maturity)
    d2 = d1 - vol * sqrt(maturity)

    price = spot * norm.cdf(d1) - strike * exp(-r*maturity) * norm.cdf(d2)
    return price
```

我们可以使用这个函数计算我们关注期权的结果：

```
print '期权价格 : %.4f' % call_option_pricer(spot, strike, maturity, r, vol)

期权价格 : 0.1133
```

2. 使用numpy加速批量计算

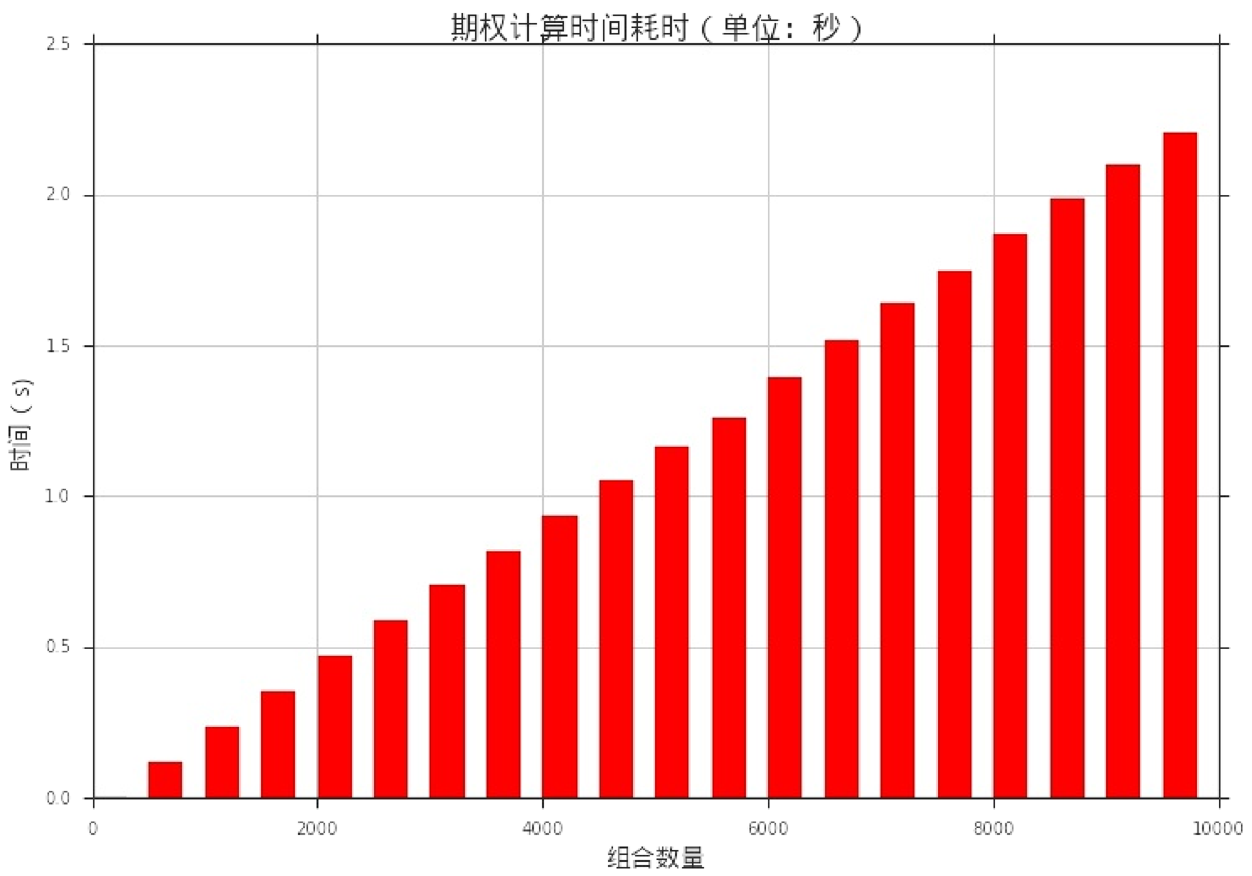
大部分的时候，我们不止关心一个期权的价格，而是关心一个组合（成千上万）的期权。我们想知道，随着期权组合数量的增长，我们计算时间的增长会有多块？

2.1 使用循环的方式

```
import time
import numpy as np

portfolioSize = range(1, 10000, 500)
timeSpent = []

for size in portfolioSize:
    now = time.time()
    strikes = np.linspace(2.0,3.0,size)
    for i in range(size):
        res = call_option_pricer(spot, strikes[i], maturity, r, vol)
    timeSpent.append(time.time() - now)
```

从下图中可以看出，计算时间的增长可以说是随着组合规模的增长线性上升。

```
from matplotlib import pylab
import seaborn as sns
font.set_size(15)
sns.set(style="ticks")
pylab.figure(figsize = (12,8))
pylab.bar(portfolioSize, timeSpent, color = 'r', width =300)
pylab.grid(True)
pylab.title(u'期权计算时间耗时（单位：秒）', fontproperties = font, fontsize = 18)
pylab.ylabel(u'时间（s)', fontproperties = font, fontsize = 15)
pylab.xlabel(u'组合数量', fontproperties = font, fontsize = 15)

<matplotlib.text.Text at 0xdbad950>
```



2.2 使用 numpy 向量计算

numpy 的内置数学函数可以天然的运用于向量：

```
sample = np.linspace(1.0,100.0,5)
np.exp(sample)

array([  2.71828183e+00,   1.52434373e+11,   8.54813429e+21,
         4.79357761e+32,   2.68811714e+43])
```

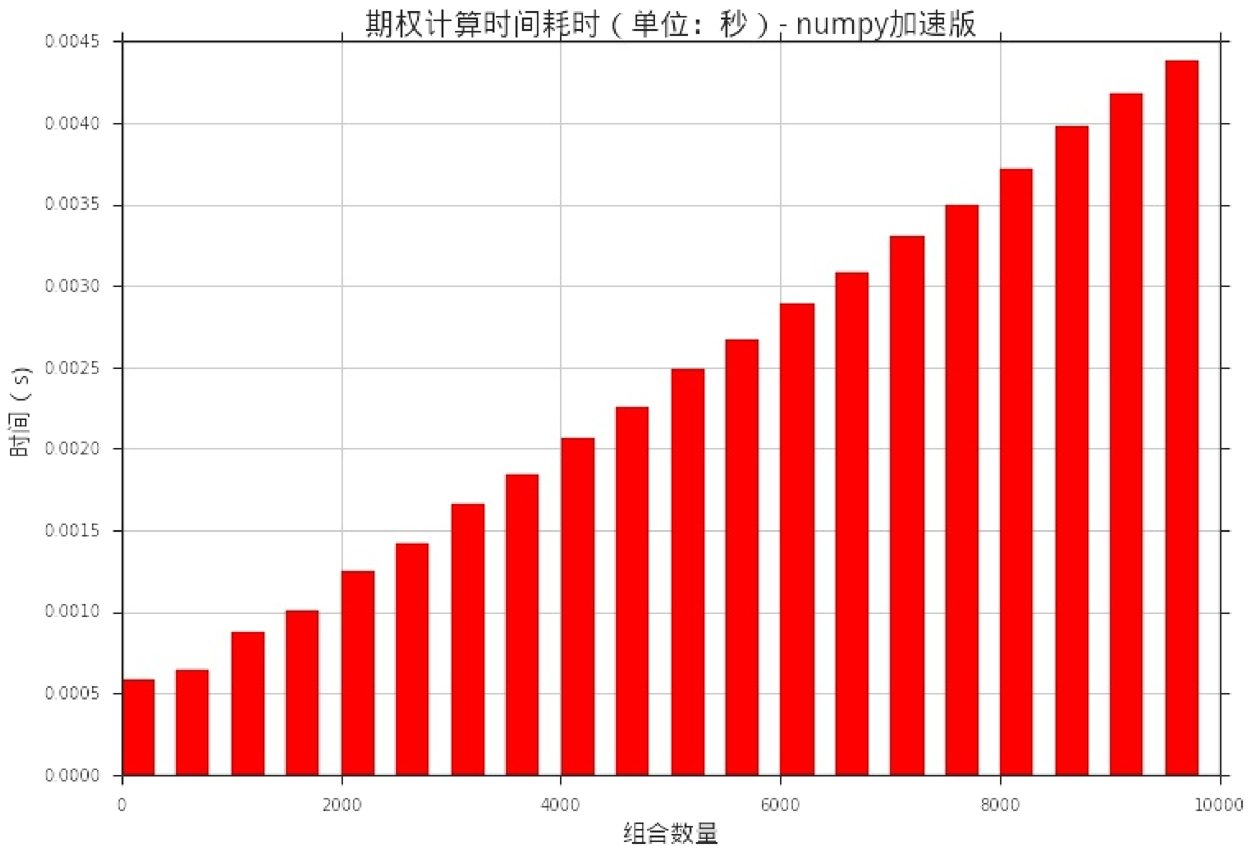
利用 numpy 的数学函数，我们可以重写原先的计算公式 call_option_pricer ，使得它接受向量参数。

```
# 使用numpy的向量函数重写Black - Scholes公式
def call_option_pricer_nunmpy(spot, strike, maturity, r, vol):

    d1 = (np.log(spot/strike) + (r + 0.5 * vol *vol) * maturity)
 / vol / np.sqrt(maturity)
    d2 = d1 - vol * np.sqrt(maturity)

    price = spot * norm.cdf(d1) - strike * np.exp(-r*maturity) *
 norm.cdf(d2)
    return price
```

```
timeSpentNumpy = []
for size in portfolioSize:
    now = time.time()
    strikes = np.linspace(2.0,3.0, size)
    res = call_option_pricer_nunmpy(spot, strikes, maturity, r,
vol)
    timeSpentNumpy.append(time.time() - now)
```

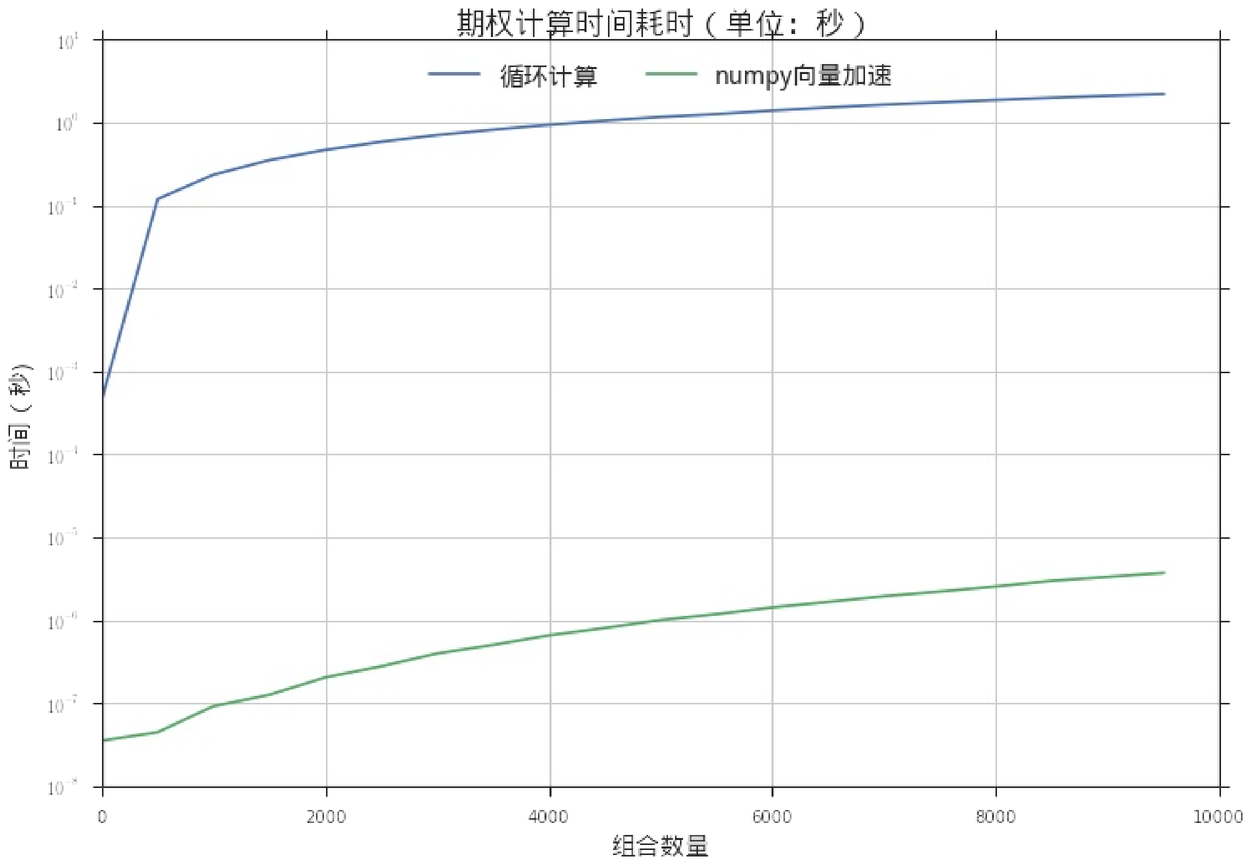
再观察一下计算耗时，虽然时间仍然是随着规模的增长线性上升，但是增长的速度要慢许多：

```
pylab.figure(figsize = (12,8))
pylab.bar(portfolioSize, timeSpentNumpy, color = 'r', width = 300
)
pylab.grid(True)
pylab.title(u'期权计算时间耗时（单位：秒）- numpy加速版', fontproperti
es = font, fontsize = 18)
pylab.ylabel(u'时间（s)', fontproperties = font, fontsize = 15)
pylab.xlabel(u'组合数量', fontproperties = font, fontsize = 15)

<matplotlib.text.Text at 0xe0ba090>
```



让我们把两次计算时间进行比对，更清楚的了解 `numpy` 计算效率的提升！

```
fig = pylab.figure(figsize = (12,8))
ax = fig.gca()
pylab.plot(portfolioSize, np.log10(timeSpent), portfolioSize, np
.log(timeSpentNumpy))
pylab.grid(True)
from matplotlib.ticker import FuncFormatter
def millions(x, pos):
    'The two args are the value and tick position'
    return '$10^{%.0f}$' % (x)
formatter = FuncFormatter(millions)
ax.yaxis.set_major_formatter(formatter)
pylab.title(u'期权计算时间耗时（单位：秒）', fontproperties = font, f
ontsize = 18)
pylab.legend([u'循环计算', u'numpy向量加速'], prop = font, loc = 'u
pper center', ncol = 2)
pylab.ylabel(u'时间（秒)', fontproperties = font, fontsize = 15)
pylab.xlabel(u'组合数量', fontproperties = font, fontsize = 15)

<matplotlib.text.Text at 0xe0b6390>
```



3. 使用 `scipy` 做仿真计算

期权价格的计算方法中有一类称为 蒙特卡洛 方法。这是利用随机抽样的方法，模拟标的股票价格随机游走，计算期权价格（未来的期望）。假设股票价格满足以下的随机游走：

$$\mathrm{d}S = rS\mathrm{d}t + \sigma S\mathrm{d}W(t).$$

仿真的方法可以模拟到期日的股票价格

$$S_T = S_0\exp((r - \frac{1}{2}\sigma^2)T + z\sigma\sqrt{T})$$

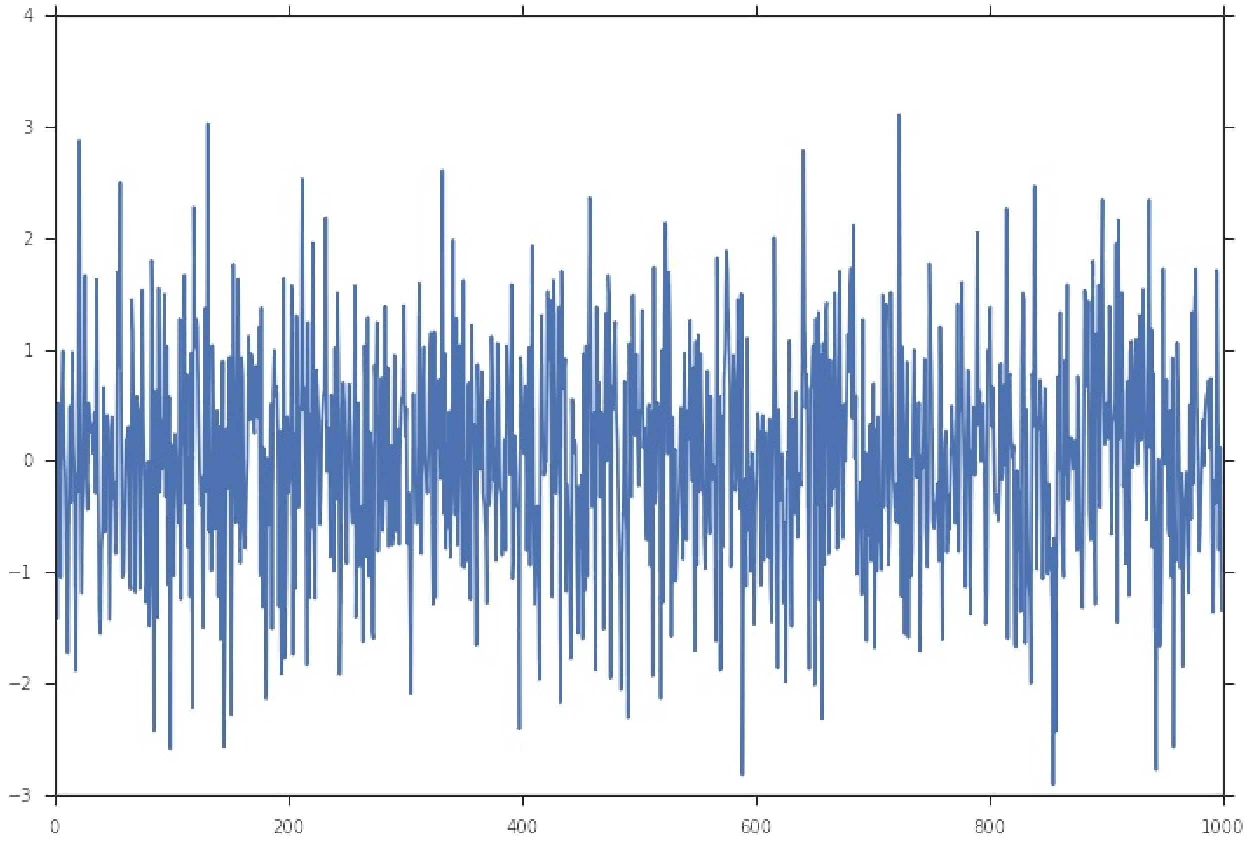
这里的 `z` 是一个符合标准正态分布的随机数。这样我们可以计算最后的期权价格：

$$\mathrm{price} = \exp(-rT)\sum_{i=1}^{N}\max(S_{T,i} - K, 0)$$

标准正态分布的随机数获取，可以方便的求助于 `scipy` 库：

```
import scipy
scipy.random.randn(10)

array([ 0.36802702,  1.09560268, -1.0235275 ,  0.15722882,  0.83
718188,
       -0.27193135, -0.03485659,  1.02705248,  0.69479874, -0.35
967107])
```

```
pylab.figure(figsize = (12,8))
randomSeries = scipy.random.randn(1000)
pylab.plot(randomSeries)
print u'均  值：%.4f' % randomSeries.mean()
print u'标准差：%.4f' % randomSeries.std()

均  值：0.0336
标准差：0.9689
```



结合 `scipy` `numpy` 我们可以定义基于蒙特卡洛的期权定价算法。

```
# 期权计算的蒙特卡洛方法
def call_option_pricer_monte_carlo(spot, strike, maturity, r, vo
l, numOfPath = 5000):
    randomSeries = scipy.random.randn(numOfPath)
    s_t = spot * np.exp((r - 0.5 * vol * vol) * maturity + rando
mSeries * vol * sqrt(maturity))
    sumValue = np.maximum(s_t - strike, 0.0).sum()
    price = exp(-r*maturity) * sumValue / numOfPath
    return price
```

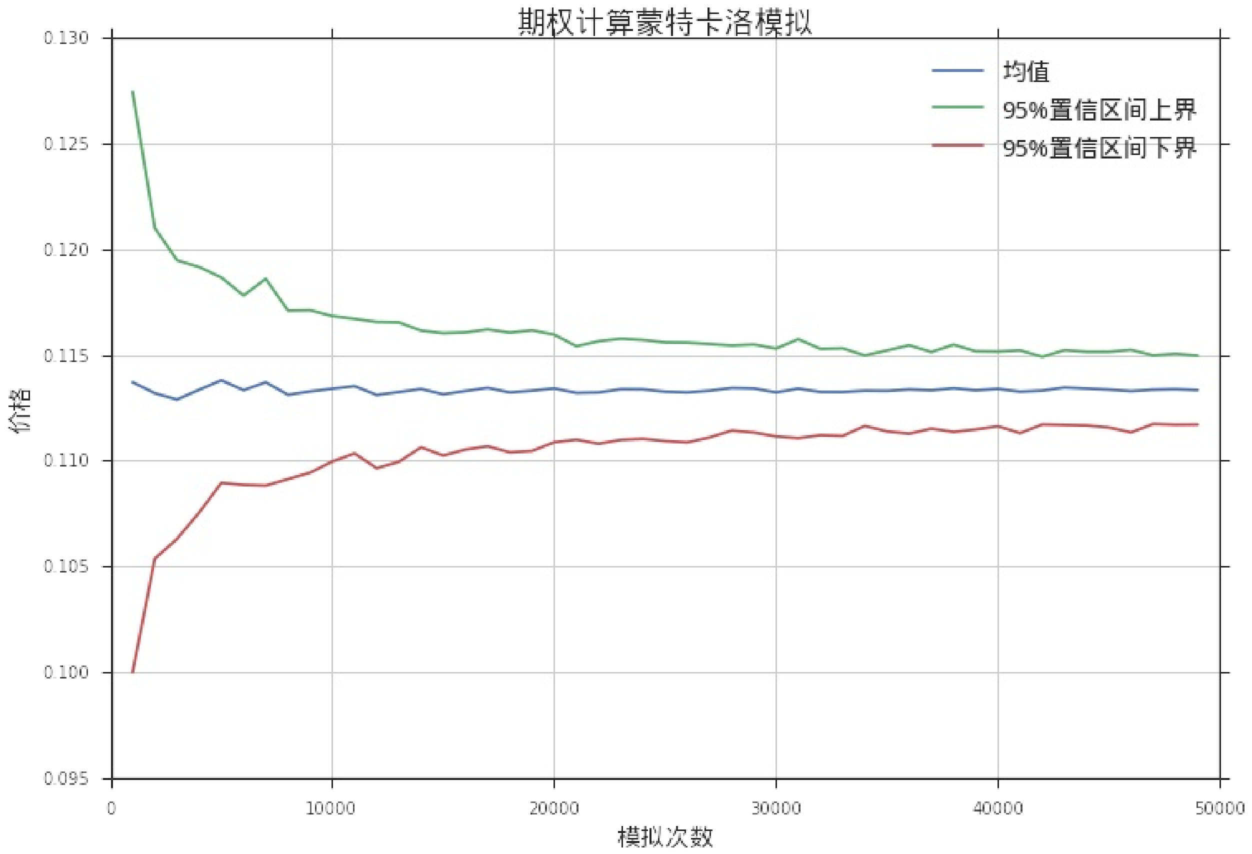
```
print '期权价格（蒙特卡洛） : %.4f' % call_option_pricer_monte_carl
o(spot, strike, maturity, r, vol)

期权价格（蒙特卡洛） : 0.1102
```

我们这里实验从1000次模拟到50000次模拟的结果，每次同样次数的模拟运行100遍。

```
pathScenario = range(1000, 50000, 1000)
numberOfTrials = 100

confidenceIntervalUpper = []
confidenceIntervalLower = []
means = []

for scenario in pathScenario:
    res = np.zeros(numberOfTrials)
    for i in range(numberOfTrials):
        res[i] = call_option_pricer_monte_carlo(spot, strike, ma
turity, r, vol, numOfPath = scenario)
    means.append(res.mean())
    confidenceIntervalUpper.append(res.mean() + 1.96*res.std())
    confidenceIntervalLower.append(res.mean() - 1.96*res.std())
```

蒙特卡洛方法会有收敛速度的考量。这里我们可以看到随着模拟次数的上升，仿眞结果的置信区间也在逐渐收敛。

```
pylab.figure(figsize = (12,8))
tabel = np.array([means,confidenceIntervalUpper,confidenceInterv
alLower]).T
pylab.plot(pathScenario, tabel)
pylab.title(u'期权计算蒙特卡洛模拟', fontproperties = font, fontsiz
e = 18)
pylab.legend([u'均值', u'95%置信区间上界', u'95%置信区间下界'], prop
= font)
pylab.ylabel(u'价格', fontproperties = font, fontsize = 15)
pylab.xlabel(u'模拟次数', fontproperties = font, fontsize = 15)
pylab.grid(True)
```



4. 计算隐含波动率

作为BSM期权定价最重要的参数，波动率 `σ` 是标的资产本身的波动率。是我们更关心的是当时的报价所反映的市场对波动率的估计，这个估计的波动率称为隐含波动率（Implied Volatility）。这里的过程实际上是在BSM公式中，假设另外4个参数确定，期权价格已知，反解 `σ` ：

$$f(\sigma) = \mathrm{Call}(S, K, r, \tau, \sigma) = S\mathrm{N}(d_1) - Ke^{-r\tau}\mathrm{N}(d_2),$$

$$\mathrm{IF}\ f(\sigma) = V, \mathrm{Then}\ \sigma = f^{-1}(V)\ ?$$

由于对于欧式看涨期权而言，其价格为对应波动率的单调递增函数，所以这个求解过程是稳定可行的。一般来说我们可以类似于试错法来实现。在 `scipy` 中已经有很多高效的算法可以为我们所用，例如Brent算法:

```
# 目标函数，目标价格由target确定
class cost_function:
    def __init__(self, target):
        self.targetValue = target

    def __call__(self, x):
        return call_option_pricer(spot, strike, maturity, r, x)
- self.targetValue

# 假设我们使用vol初值作为目标
target = call_option_pricer(spot, strike, maturity, r, vol)
cost_sampel = cost_function(target)

# 使用Brent算法求解
impliedVol = brentq(cost_sampel, 0.01, 0.5)

print u'眞实波动率： %.2f' % (vol*100,) + '%'
print u'隐含波动率： %.2f' % (impliedVol*100,) + '%'

眞实波动率： 25.00%
隐含波动率： 25.00%
```

量化分析师的Python日记【第8天 Q Quant兵器谱之函数插值】

> 来源：https://uqer.io/community/share/551cfa1ff9f06c8f339044ff
>
> 在本篇中，我们将介绍Q宽客常用工具之一：函数插值。接着将函数插值应用于一个实际的金融建模场景中：波动率曲面构造。
>
> 通过本篇的学习您将学习到：
>
> 1. 如何在 `scipy` 中使用函数插值模块： `interpolate` ；
> 2. 波动率曲面构造的原理；
> 3. 将 `interpolate` 运用于波动率曲面构造。

1. 如何使用 `scipy` 做函数插值

函数插值，即在离散数据的基础上补插连续函数，估算出函数在其他点处的近似值的方法。在 `scipy` 中，所有的与函数插值相关的功能都在 `scipy.interpolate` 模块中

```
from scipy import interpolate
dir(interpolate)[:5]

['Akima1DInterpolator',
 'BPoly',
 'BarycentricInterpolator',
 'BivariateSpline',
 'CloughTocher2DInterpolator']
```

作为介绍性质的本篇，我们将只关注 `interpolate.spline` 的使用，即样条插值方法：

- `xk` 离散的自变量值，为序列
- `yk` 对应 `xk` 的函数值，为与 `xk` 长度相同的序列
- `xnew` 需要进行插值的自变量值序列
- `order` 样条插值使用的函数基德阶数，为1时使用线性函数

```
print interpolate.spline.__doc__

Interpolate a curve at new points using a spline fit

Parameters
----------
xk, yk : array_like
    The x and y values that define the curve.
xnew : array_like
    The x values where spline should estimate the y values.
order : int
    Default is 3.
kind : string
    One of {'smoothest'}
conds : Don't know
    Don't know

Returns
-------
spline : ndarray
    An array of y values; the spline evaluated at the positions
`xnew`.
```

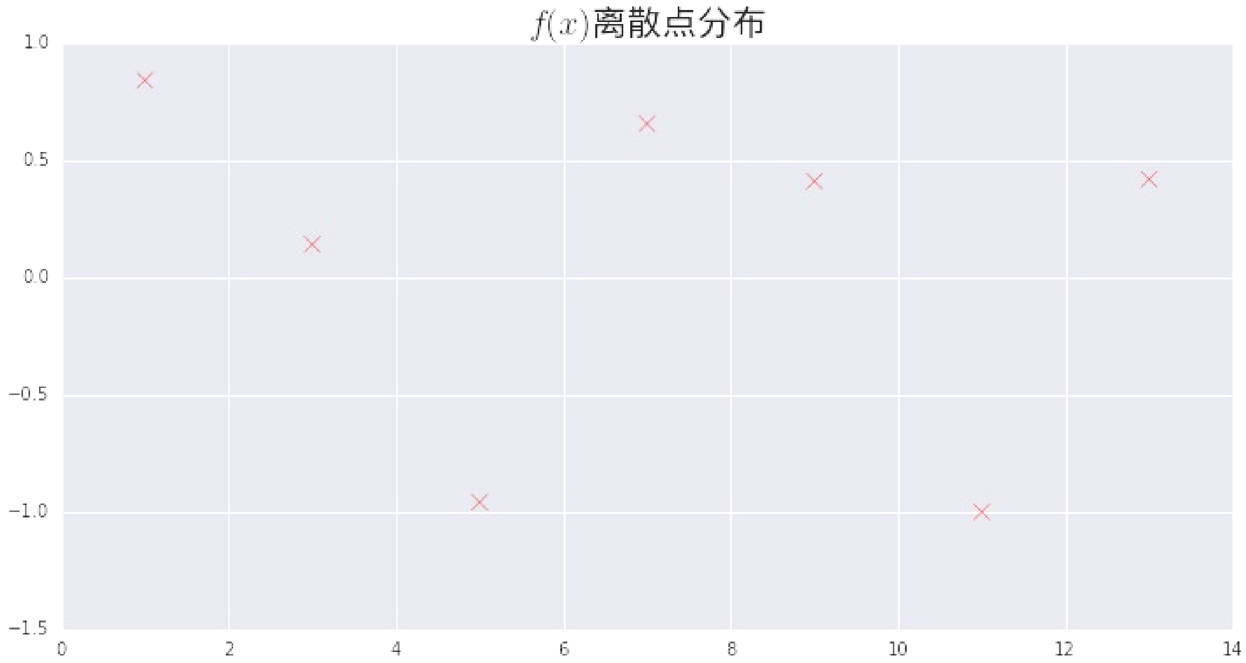
1.1 三角函数（ `np.sin` ）插值

一例胜千言！让我们这里用实际的一个示例，来说明如何在 `scipy` 中使用函数插值。这里的目标函数是三角函数：

$$f(x) = \sin(x)$$

假设我们已经观测到的 `f(x)` 在离散点 `x=(1,3,5,7,9,11,13)` 的值：

```
import numpy as np
from matplotlib import pylab
import seaborn as sns
font.set_size(20)
x = np.linspace(1.0, 13.0, 7)
y = np.sin(x)
pylab.figure(figsize = (12,6))
pylab.scatter(x,y, s = 85, marker='x', color = 'r')
pylab.title(u'$f(x)$离散点分布', fontproperties = font)

<matplotlib.text.Text at 0x142cafd0>
```



首先我们使用最简单的线性插值算法，这里面只要将 `spline` 的参数 `order` 设置为1即可：

```
xnew = np.linspace(1.0,13.0,500)
ynewLinear = interpolate.spline(x,y,xnew,order = 1)
ynewLinear[:5]

array([ 0.84147098,  0.83304993,  0.82462888,  0.81620782,  0.80
778677])
```

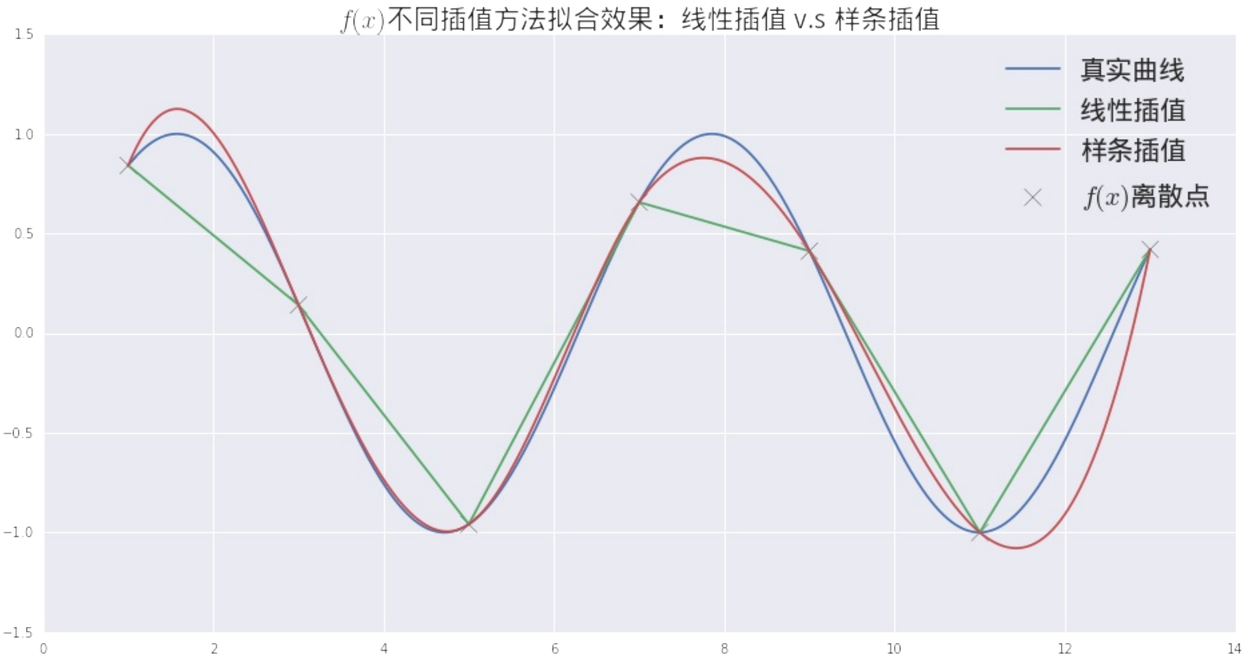
复杂一些的，也是 `spline` 函数默认的方法，即为样条插值，将 `order` 设置为3即可：

最后我们获得真实的 `sin(x)` 的值：

```
ynewReal = np.sin(xnew)
ynewReal[:5]

array([ 0.84147098,  0.85421967,  0.86647437,  0.87822801,  0.88
947378])
```

让我们把所有的函数画到一起，看一下插值的效果。对于我们这个例子中的目标函数而言，由于本身目标函数是光滑函数，则越高阶的样条插值的方法，插值效果越好。

```
pylab.figure(figsize = (16,8))
pylab.plot(xnew,ynewReal)
pylab.plot(xnew,ynewLinear)
pylab.plot(xnew,ynewCubicSpline)
pylab.scatter(x,y, s = 160, marker='x', color = 'k')
pylab.legend([u'眞实曲线', u'线性插值', u'样条插值', u'$f(x)$离散点']
, prop = font)
pylab.title(u'$f(x)$不同插值方法拟合效果：线性插值 v.s 样条插值', font
properties = font)

<matplotlib.text.Text at 0x1424cd50>
```



2. 函数插值应用 —— 期权波动率曲面构造

市场上期权价格一般以隐含波动率的形式报出，一般来讲在市场交易时间，交易员可以看到类似的波动率矩阵（Volatilitie Matrix):

```
import pandas as pd
pd.options.display.float_format = '{:,>.2f}'.format
dates = [Date(2015,3,25), Date(2015,4,25), Date(2015,6,25), Date(2015,9,25)]
strikes = [2.2, 2.3, 2.4, 2.5, 2.6]
blackVolMatrix = np.array([[ 0.32562851,  0.29746885,  0.29260648,  0.27679993],
                          [ 0.28841840,  0.29196629,  0.27385023,  0.26511898],
                          [ 0.27659511,  0.27350773,  0.25887604,  0.25283775],
                          [ 0.26969754,  0.25565971,  0.25803327,  0.25407669],
                          [ 0.27773032,  0.24823248,  0.27340796,  0.24814975]])
table = pd.DataFrame(blackVolMatrix * 100, index = strikes, columns = dates, )
table.index.name = u'行权价'
table.columns.name = u'到期时间'
print u'2015年3月3日10时波动率矩阵'
table

2015年3月3日10时波动率矩阵
```

到期时间	March 25th, 2015	April 25th, 2015	June 25th, 2015	September 25th, 2015
行权价				
2.20	32.56	29.75	29.26	27.68
2.30	28.84	29.20	27.39	26.51
2.40	27.66	27.35	25.89	25.28
2.50	26.97	25.57	25.80	25.41
2.60	27.77	24.82	27.34	24.81

交易员可以看到市场上离散值的信息，但是如果可以获得一些隐含的信息更好：例如，在2015年6月25日以及2015年9月25日之间，波动率的形状会是怎么样的？

2.1 方差曲面插值

我们并不是直接在波动率上进行插值，而是在方差矩阵上面进行插值。方差和波动率的关系如下：

$$\mathrm{Var}(K,T) = \sigma(K,T)^2 T$$

所以下面我们将通过处理，获取方差矩阵（Variance Matrix):

```
evaluationDate = Date(2015,3,3)
ttm = np.array([(d - evaluationDate) / 365.0 for d in dates])
varianceMatrix = (blackVolMatrix**2) * ttm
varianceMatrix

array([[ 0.00639109,  0.0128489 ,  0.02674114,  0.04324205],
       [ 0.0050139 ,  0.01237794,  0.02342277,  0.03966943],
       [ 0.00461125,  0.01086231,  0.02093128,  0.03607931],
       [ 0.00438413,  0.0094909 ,  0.02079521,  0.03643376],
       [ 0.00464918,  0.00894747,  0.02334717,  0.03475378]])
```

这里的值 `varianceMatrix` 就是变换而得的方差矩阵。

下面我们将在行权价方向以及时间方向同时进行线性插值，具体地，行权价方向：

$$\mathrm{Var}(K,t) = \frac{K_2 - K}{K_2 - K_1}\mathrm{Var}(K_1,t) + \frac{K - K_1}{K_2 - K_1}\mathrm{Var}(K_2,t)$$

时间方向：

$$\mathrm{Var}(K) = \frac{t_2 - t}{t_2 - t_1}\mathrm{Var}(K,t_1) + \frac{t - t_1}{t_2 - t_1}\mathrm{Var}(K,t_2)$$

这个过程在 `scipy` 中可以直接通过 `interpolate` 模块下 `interp2d` 来实现：

- `ttm` 时间方向离散点
- `strikes` 行权价方向离散点
- `varianceMatrix` 方差矩阵，列对应时间维度；行对应行权价维度
- `kind = 'linear'` 指示插值以线性方式进行

```
interp = interpolate.interp2d(ttm, strikes, varianceMatrix, kind
 = 'linear')
```

返回的 `interp` 对象可以用于获取任意点上插值获取的方差值：

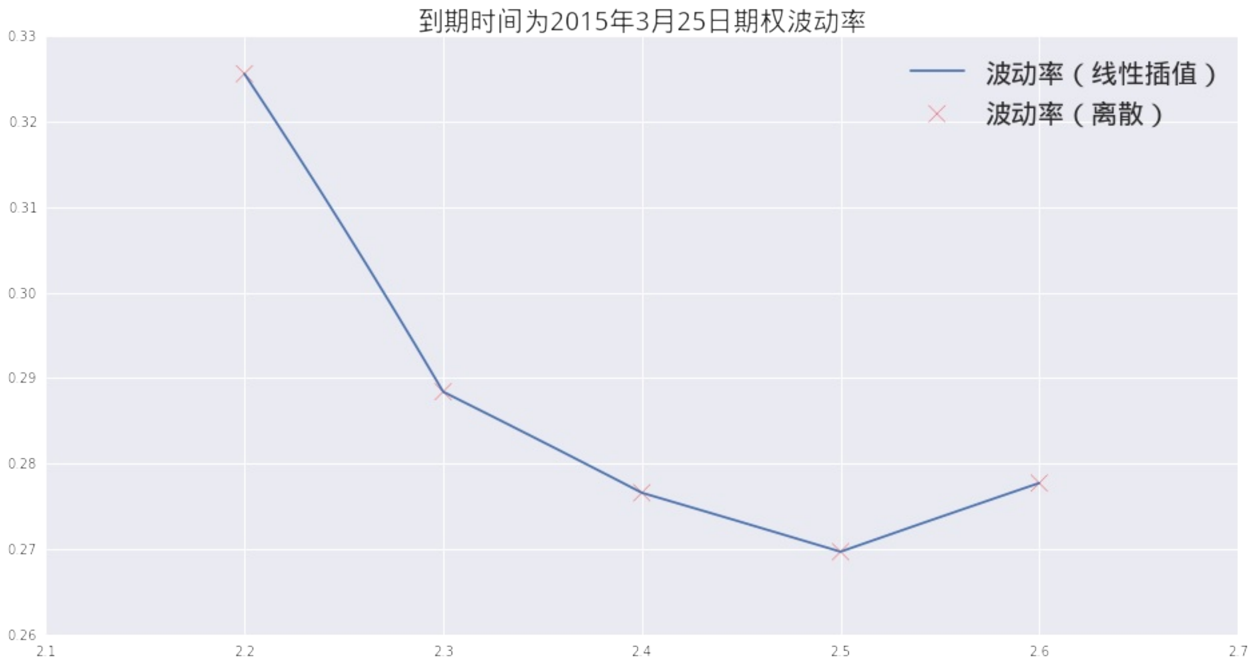
```
interp(ttm[0], strikes[0])

array([ 0.00639109])
```

最后我们获取整个平面上所有点的方差值，再转换为波动率曲面。

```
sMeshes = np.linspace(strikes[0], strikes[-1], 400)
tMeshes = np.linspace(ttm[0], ttm[-1], 200)
interpolatedVarianceSurface = np.zeros((len(sMeshes), len(tMeshe
s)))
for i, s in enumerate(sMeshes):
    for j, t in enumerate(tMeshes):
        interpolatedVarianceSurface[i][j] = interp(t,s)

interpolatedVolatilitySurface = np.sqrt((interpolatedVarianceSur
face / tMeshes))
print u'行权价方向网格数：', np.size(interpolatedVolatilitySurface,
0)
print u'到期时间方向网格数：', np.size(interpolatedVolatilitySurfac
e, 1)

行权价方向网格数： 400
到期时间方向网格数： 200
```

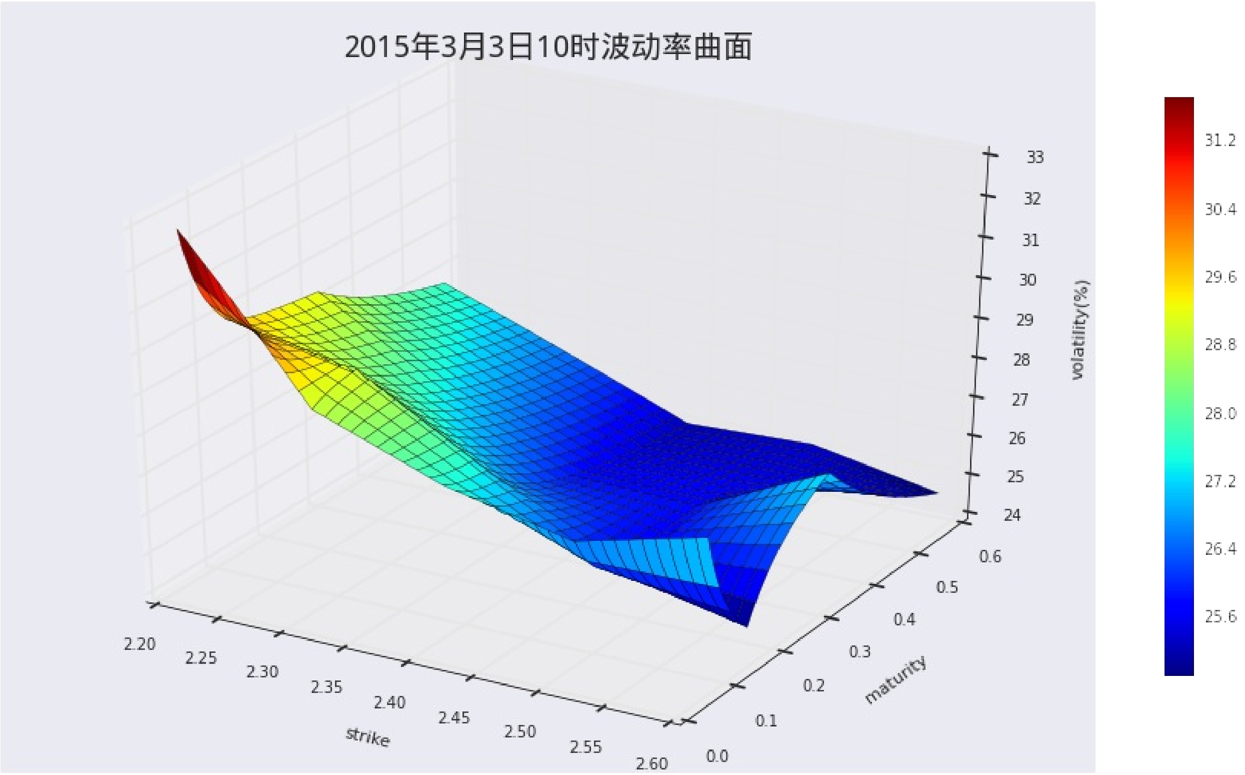
选取某一个到期时间上的波动率点，看一下插值的效果。这里我们选择到期时间最近的点：2015年3月25日：

```
pylab.figure(figsize = (16,8))
pylab.plot(sMeshes, interpolatedVolatilitySurface[:, 0])
pylab.scatter(x = strikes, y = blackVolMatrix[:,0], s = 160,mark
er = 'x', color = 'r')
pylab.legend([u'波动率（线性插值）', u'波动率（离散）'], prop = font)
pylab.title(u'到期时间为2015年3月25日期权波动率', fontproperties = f
ont)

<matplotlib.text.Text at 0xea27f90>
```



最终，我们把整个曲面的图像画出来看看：

```
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm

maturityMesher, strikeMesher = np.meshgrid(tMeshes, sMeshes)
pylab.figure(figsize = (16,9))
ax = pylab.gca(projection = '3d')
surface = ax.plot_surface(strikeMesher, maturityMesher, interpol
atedVolatilitySurface*100, cmap = cm.jet)
pylab.colorbar(surface,shrink=0.75)
pylab.title(u'2015年3月3日10时波动率曲面', fontproperties = font)
pylab.xlabel("strike")
pylab.ylabel("maturity")
ax.set_zlabel(r"volatility(%)")

<matplotlib.text.Text at 0x14e03050>
```


2015年3月3日10时波动率曲面
strike
2.20
2.25
2.30
2.35
2.40
2.45
2.50
2.55
2.60
maturity
0.0
0.1
0.2
0.3
0.4
0.5
0.6
volatility(%)
24
25
26
27
28
29
30
31
32
33
31.2
30.4
29.6
28.8
28.0
27.2
26.4
25.6

量化分析师的Python日记【第9天 Q Quant兵器谱之二叉树】

> 来源：https://uqer.io/community/share/5523a4a1f9f06c8f3390453b

通过之前几天的学习，Q Quant们应该已经熟悉了Python的基本语法，也了解了Python中常用数值库的算法。到这里为止，小Q们也许早就对之前简单的例子不满意，希望能在Python里面玩票大的！Ok，我们这里引入一个不怎么像玩具的模型——二叉树算法。我们仍然以期权为例子，教会大家：

1. 如何利用Python的控制语句与基本内置计算方法，构造一个二叉树模型；
2. 如何使用类封装的方式，抽象二叉树算法，并进行扩展；
3. 利用继承的方法为已有二叉树算法增加美式期权算法。

```
import numpy as np
import math
import seaborn as sns
from matplotlib import pylab
font.set_size(15)
```

1. 小Q的第一棵“树”——二叉树算法Python描述

我们这边只会简单的描述二叉树的算法，不会深究其原理，感兴趣的读者可以很方便的从公开的文献中获取细节。

我们这里仍然考虑基础的 Black - Scholes 模型：

$$\mathrm{d}S = (r - d)S\mathrm{d}t + \sigma S\mathrm{d}W_t$$

这里各个字母的含义如之前介绍，多出来的 r 代表股息率。

之所以该算法被称为 二叉树，因为这个算法的基础结构是一个逐层递增的树杈式结构：

一个基本的二叉树机构由以下三个参数决定：

1. `up` 标的资产价格向上跳升的比例， `up` 必然大于1 （对应上图中的 `u`)
2. `down` 标的资产价格向下跳升的比例， `dow` n必然小于1 (对应上图中的 `d`)
3. `upProbability` 标的资产价格向上跳升的概率

这里我们用一个具体的例子，使用Python实现二叉树算法。以下为具体参数：

- `ttm` 到期时间，单位年
- `tSteps` 时间方向步数
- `r` 无风险利率
- `d` 标的股息率

- `sigma` 波动率
- `strike` 期权行权价
- `spot` 标的现价

这里我们只考虑看涨期权。

```
# 设置基本参数
ttm = 3.0
tSteps = 25
r = 0.03
d = 0.02
sigma = 0.2
strike = 100.0
spot = 100.0
```

我们这里用作例子的树结构被称为 Jarrow - Rudd 树，其中：

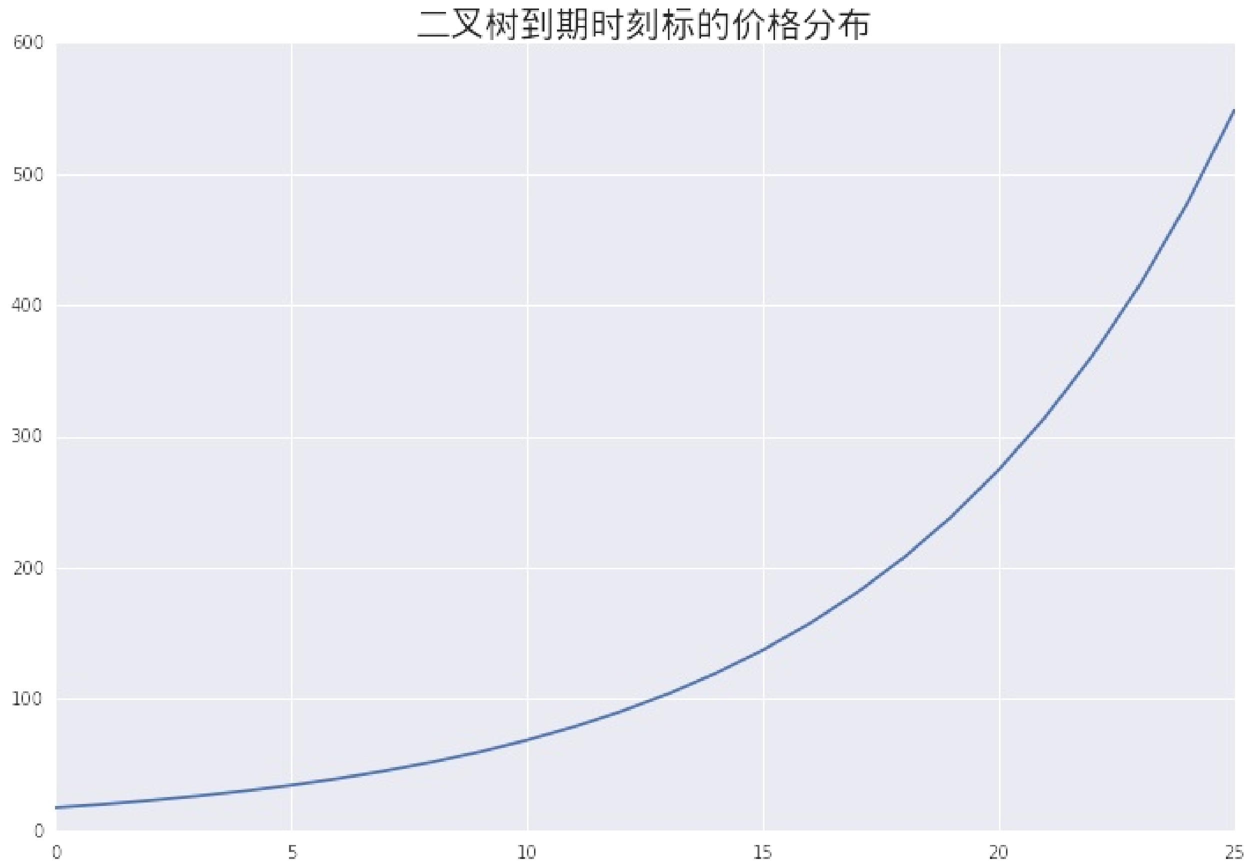
$$up = \exp((r - d - \frac{1}{2}\sigma^2)\Delta t + \sigma\sqrt{\Delta t})$$

$$down = \exp((r - d - \frac{1}{2}\sigma^2)\Delta t + \sigma\sqrt{\Delta t})$$

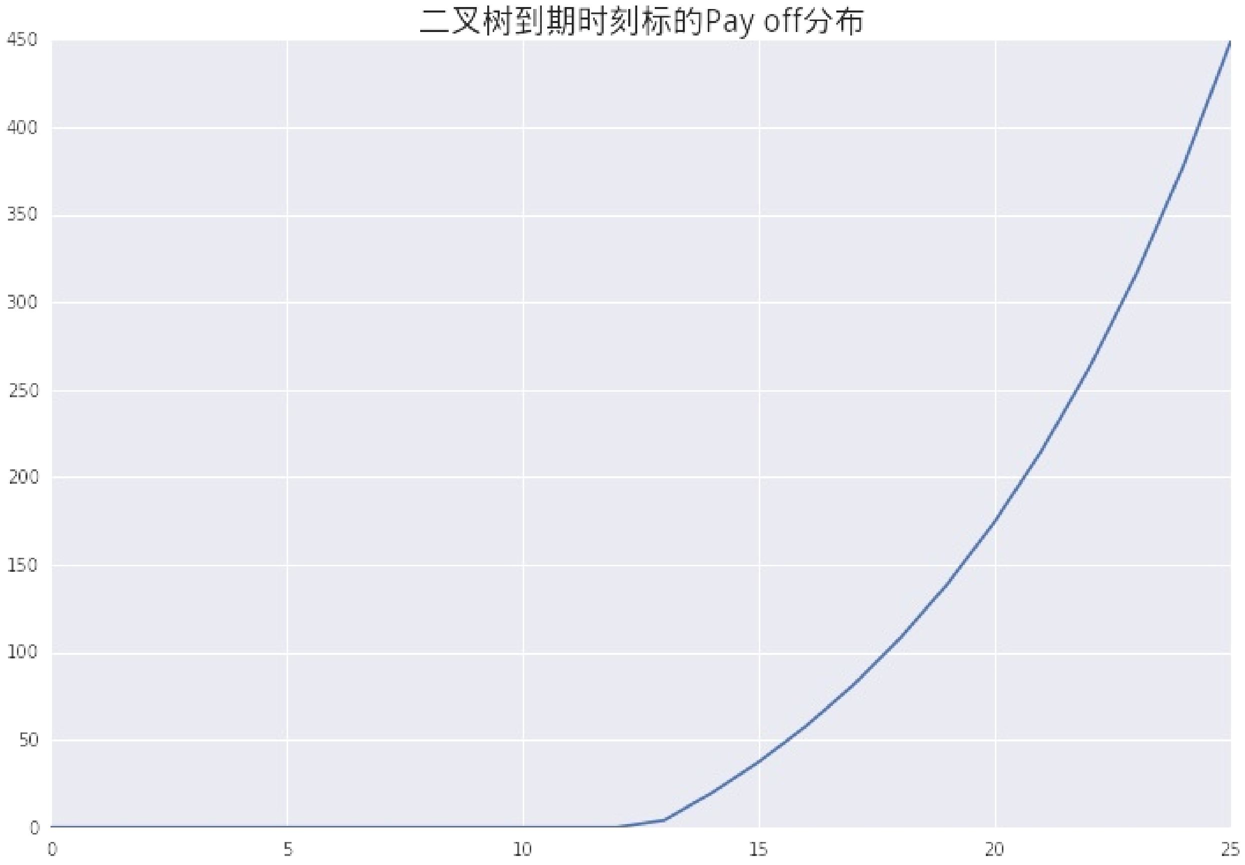
$$upProb = 0.5$$

```
dt = ttm / tSteps
up = math.exp((r - d - 0.5*sigma*sigma)*dt + sigma*math.sqrt(dt)
)
down = math.exp((r - d - 0.5*sigma*sigma)*dt - sigma*math.sqrt(d
t))
discount = math.exp(-r*dt)
```

```
pylab.figure(figsize = (12,8))
pylab.plot(lattice[tSteps])
pylab.title(u'二叉树到期时刻标的价格分布', fontproperties = font, fo
ntsize = 20)

<matplotlib.text.Text at 0x16bb2290>
```



```
# 在节点上计算payoff
def call_payoff(spot):
    global strike
    return max(spot - strike, 0.0)

pylab.figure(figsize = (12,8))
pylab.plot(map(call_payoff, lattice[tSteps]))
pylab.title(u'二叉树到期时刻标的Pay off分布', fontproperties = font,
 fontsize = 18)

<matplotlib.text.Text at 0x16bc4210>
```



在我们从树最茂盛的枝叶向根部回溯的时候，第 `i` 层节点与第 `i+1` 层节点的关系满足：

$$\mathrm{Lattice}[i][j] = discount \times (upProb \times \mathrm{Lattice}[i+1][j+1] + (1 - upProb) \times \mathrm{Lattice}[i+1][j])$$

```
# 反方向回溯整棵树
for i in range(tSteps,0,-1):
    for j in range(i,0,-1):
        if i == tSteps:
            lattice[i-1][j-1] = 0.5 * discount * (call_payoff(la
ttice[i][j]) + call_payoff(lattice[i][j-1]))
        else:
            lattice[i-1][j-1] = 0.5 * discount * (lattice[i][j]
+ lattice[i][j-1])
```

```
print u'二叉树价格： %.4f' % lattice[0][0]
print u'解析法价格： %.4f' % BSMPrice(1, strike, spot, r, d, sigma
, ttm, rawOutput= True)[0]

二叉树价格： 14.2663
解析法价格： 14.1978
```

2. 从“树”到“森林”—— 面向对象方式实现二叉树算法

之前的部分展示了一个树算法的基本结构。但是现在的实现由很多缺点：

- 没有明确接口，作为用户优雅简洁的使用既有算法；
- 没有完整封装，十分不利于算法的扩展；

下面我们将给出一个基于Python类的二叉树算法实现，实际上我们通过上面的实验性探索，发现整个程序可以拆成三个互相独立的功能模块：

- 二叉树框架

  树的框架结构，包括节点数以及基本参数的保存；

- 二叉树类型描述

  具体数算法的参数，例如上例中的 Jarrow Rudd树；

- 偿付函数

  到期的偿付形式，即为Payoff Function。

2.1 二叉树框架（`BinomialTree`）

这个类负责二叉树框架的构造，也是基本的二叉树算法的调用入口。它有三个成员：

- 构造函数（`__init__`）

  负责接受用户定义的具体参数，例如：`spot` 等；真正二叉树的构造方法，由私有方法 `_build_lattice` 以及传入参数 `treeTraits` 共同完成；

- 树构造细节（`_build_lattice`）

  接手具体的树构造过程，这里需要依赖根据 `treeTraits` 获取的参数例如：`up` , `down` 。

- 树回溯（`roll_back`）

  从树的最茂盛枝叶节点向根节点回溯的过程。最终根节点的值即为期权的价值。这里它要求的参数是一个 `pay_off` 函数。

```
# 二叉树框架（可以通过传入不同的treeTraits类型，设计不同的二叉树结构）
class BinomialTree:
    def __init__(self, spot, riskFree, dividend, tSteps, maturity, sigma, treeTraits):
        self.dt = maturity / tSteps
        self.spot = spot
        self.r = riskFree
        self.d = dividend
        self.tSteps = tSteps
        self.discount = math.exp(-self.r*self.dt)
        self.v = sigma
        self.up = treeTraits.up(self)
        self.down = treeTraits.down(self)
        self.upProbability = treeTraits.upProbability(self)
        self.downProbability = 1.0 - self.upProbability
        self._build_lattice()

    def _build_lattice(self):
        '''
        完成构造二叉树的工作
        '''
        self.lattice = np.zeros((self.tSteps+1, self.tSteps+1))
        self.lattice[0][0] = self.spot
        for i in range(self.tSteps):
            for j in range(i+1):
                self.lattice[i+1][j+1] = self.up * self.lattice[i][j]
            self.lattice[i+1][0] = self.down * self.lattice[i][0]

    def roll_back(self, payOff):
        '''
        节点计算，并反向倒推
        '''
        for i in range(self.tSteps,0,-1):
            for j in range(i,0,-1):
                if i == self.tSteps:
                    self.lattice[i-1][j-1] = self.discount * (self.upProbability * payOff(self.lattice[i][j]) + self.downProbability * payOff(self.lattice[i][j-1]))
                else:
                    self.lattice[i-1][j-1] = self.discount * (self.upProbability *  self.lattice[i][j] + self.downProbability * self.lattice[i][j-1])
```

2.2 二叉树类型描述（ `Tree Traits` ）

正像我们之前描述的那样，任意的树只要描述三个方面的特征就可以。所以我们设计的 `Tree Traits` 类只要通过它的静态成员返回这些特征就可以：

- `up` 返回向上跳升的比例；
- `down` 返回向下调降的比例；
- `upProbability` 返回向上跳升的概率

下面的类定义了 Jarrow - Rudd 树的描述：

```
class JarrowRuddTraits:
    @staticmethod
    def up(tree):
        return math.exp((tree.r - tree.d - 0.5*tree.v*tree.v)*tree.dt + tree.v*math.sqrt(tree.dt))

    @staticmethod
    def down(tree):
        return math.exp((tree.r - tree.d - 0.5*tree.v*tree.v)*tree.dt - tree.v*math.sqrt(tree.dt))

    @staticmethod
    def upProbability(tree):
        return 0.5
```

我们这里再给出另一个 Cox - Ross - Rubinstein 树的描述：

```
class CRRTraits:
    @staticmethod
    def up(tree):
        return math.exp(tree.v * math.sqrt(tree.dt))

    @staticmethod
    def down(tree):
        return math.exp(-tree.v * math.sqrt(tree.dt))

    @staticmethod
    def upProbability(tree):
        return 0.5 + 0.5 * (tree.r - tree.d - 0.5 * tree.v*tree.v) * tree.dt / tree.v / math.sqrt(tree.dt)
```

2.3 偿付函数（ `pay_off` ）

这部分很简单，就是一元函数，输入为标的价格，输出的偿付收益，对于看涨期权来说就是：

$$pay = \max(S - K, 0)$$

```
def pay_off(spot):
    global strike
    return max(spot - strike, 0.0)
```

2.4 组装

让我们三部分组装起来，现在整个调用过程变得什么清晰明了，同时最后的结果和第一部分是完全一致的。

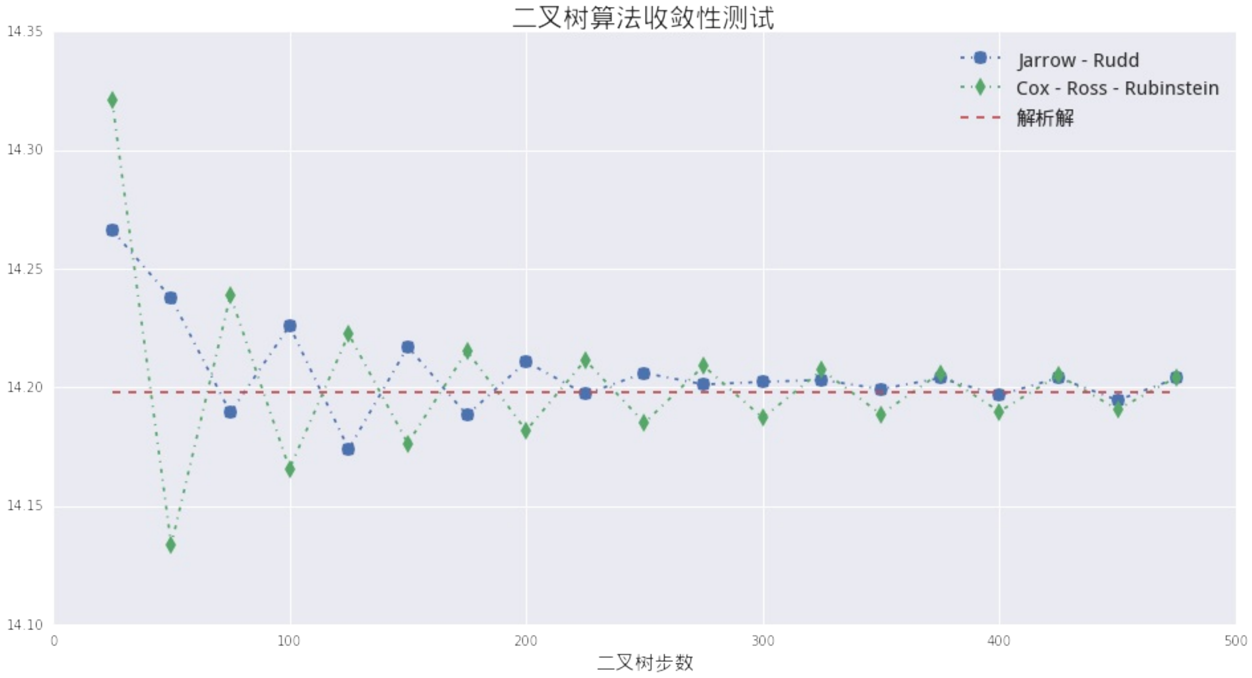
```
testTree = BinomialTree(spot, r, d, tSteps, ttm, sigma, JarrowRu
ddTraits)
testTree.roll_back(pay_off)
print u'二叉树价格： %.4f' % testTree.lattice[0][0]

二叉树价格： 14.2663
```

这里我们想更进一步，用我们现在的算法框架来测试二叉树的收敛性。这里我们用来作比较的算法即为之前描述的 Jarrow - Rudd 以及 Cox - Ross - Rubinstein 树：

```
stepSizes = range(25, 500,25)
jrRes = []
crrRes = []
for tSteps in stepSizes:
    # Jarrow - Rudd 结果
    testTree = BinomialTree(spot, r, d, tSteps, ttm, sigma, Jarr
owRuddTraits)
    testTree.roll_back(pay_off)
    jrRes.append(testTree.lattice[0][0])

    # Cox - Ross - Rubinstein 结果
    testTree = BinomialTree(spot, r, d, tSteps, ttm, sigma, CRRT
raits)
    testTree.roll_back(pay_off)
    crrRes.append(testTree.lattice[0][0])
```

我们可以绘制随着步数的增加，两种二叉树算法逐渐向真实值收敛的过程。

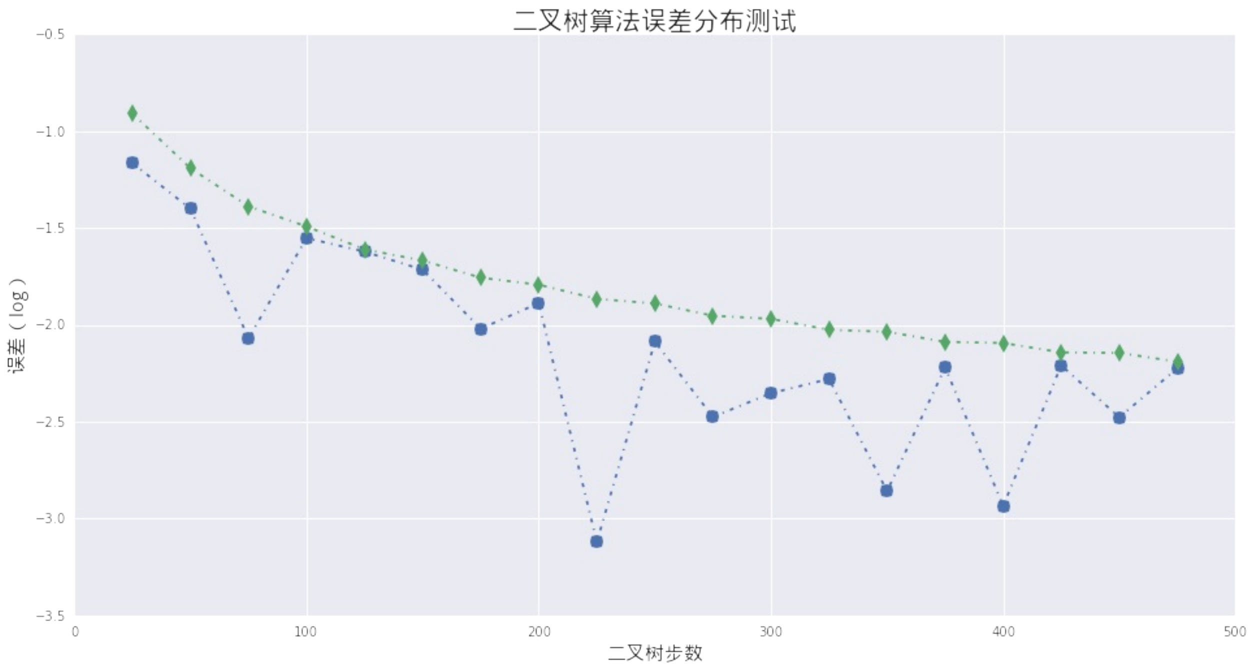
```
anyRes = [BSMPrice(1, strike, spot, r, d, sigma, ttm, rawOutput=
True)[0]] * len(stepSizes)

pylab.figure(figsize = (16,8))
pylab.plot(stepSizes, jrRes, '-.', marker = 'o', markersize = 10
)
pylab.plot(stepSizes, crrRes, '-.', marker = 'd', markersize = 10
)
pylab.plot(stepSizes, anyRes, '--')
pylab.legend(['Jarrow - Rudd', 'Cox - Ross - Rubinstein', u'解析
解'], prop = font)
pylab.xlabel(u'二叉树步数', fontproperties = font)
pylab.title(u'二叉树算法收敛性测试', fontproperties = font, fontsiz
e = 20)

<matplotlib.text.Text at 0x15e46490>
```



我们也可以绘制两种算法的误差随着步长下降的过程。

```
jrErr = np.array(jrRes) - np.array(anyRes)
crrErr = np.array(crrRes) - np.array(anyRes)
jrErr = np.log10(np.abs(jrErr))
crrErr = np.log10(np.abs(crrErr))
```

```
pylab.figure(figsize = (16,8))
pylab.plot(stepSizes, jrErr, '-.', marker = 'o', markersize = 10)
pylab.plot(stepSizes, crrErr, '-.', marker = 'd', markersize = 10)
pylab.xlabel(u'二叉树步数', fontproperties = font)
pylab.ylabel(u'误差（log）', fontproperties = font)
pylab.title(u'二叉树算法误差分布测试', fontproperties = font, fontsize = 20)

<matplotlib.text.Text at 0x172b06d0>
```



3. 新想法 —— 美式期权？

有小Q要问了，既然我们已经有解析算法了，为什么还要多此一举的去种“树”呢？是的，如果只是普通欧式期权的话，二叉树就是多此一举的做法。但是由于二叉树天然的反向回溯的特性，使得它特别适合处理有提前行权结构的期权产品。这里我们将以美式期权为例。

美式期权的行权结构在二叉树结构下处理起来特别简单，要做的只是在每个节点上做这样的比较：

$$\text{Lattice}[i][j] = \max(ExerciseValue, EuropeanValue)$$

这里的 `ExerciseValue` 就是立即行权的价值， `EuropeanValue` 为对应节点的欧式价值。

为了实现上面的比较，我们需要扩展原先的算法，这个我们可以通过Python的类继承在原先的类之上添加新功能：

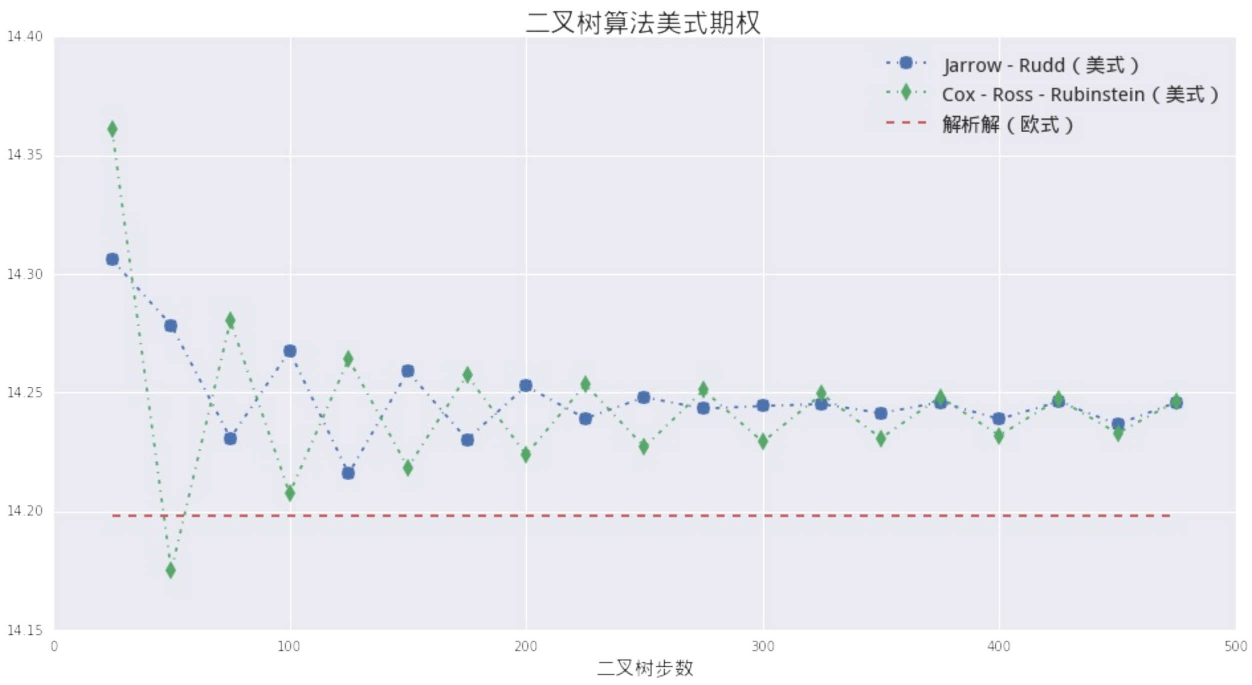
```
class ExtendBinomialTree(BinomialTree):

    def roll_back_american(self, payOff):
        '''
        节点计算，并反向倒推
        '''
        for i in range(self.tSteps,0,-1):
            for j in range(i,0,-1):
                if i == self.tSteps:
                    europeanValue = self.discount * (self.upProb
ability * payOff(self.lattice[i][j]) + self.downProbability * pa
yOff(self.lattice[i][j-1]))
                else:
                    europeanValue = self.discount * (self.upProb
ability *  self.lattice[i][j] + self.downProbability * self.latt
ice[i][j-1])
                # 处理美式行权
                exerciseValue = payOff(self.lattice[i-1][j-1])
                self.lattice[i-1][j-1] = max(europeanValue, exer
ciseValue)
```

我们将使用同样的参数测试美式期权算法的实现：

```
stepSizes = range(25, 500,25)
jrRes = []
crrRes = []
for tSteps in stepSizes:
    # Jarrow - Rudd 结果
    testTree = ExtendBinomialTree(spot, r, d, tSteps, ttm, sigma
, JarrowRuddTraits)
    testTree.roll_back_american(pay_off)
    jrRes.append(testTree.lattice[0][0])

    # Cox - Ross - Rubinstein 结果
    testTree = ExtendBinomialTree(spot, r, d, tSteps, ttm, sigma
, CRRTraits)
    testTree.roll_back_american(pay_off)
    crrRes.append(testTree.lattice[0][0])
```

我们画出美式期权价格的收敛图，价格始终高于欧式期权的价格，符合预期。

```
anyRes = [BSMPrice(1, strike, spot, r, d, sigma, ttm, rawOutput=
True)[0]] * len(stepSizes)

pylab.figure(figsize = (16,8))
pylab.plot(stepSizes, jrRes, '-.', marker = 'o', markersize = 10
)
pylab.plot(stepSizes, crrRes, '-.', marker = 'd', markersize = 10
)
pylab.plot(stepSizes, anyRes, '--')
pylab.legend([u'Jarrow - Rudd（美式）', u'Cox - Ross - Rubinstein
（美式）', u'解析解（欧式）'], prop = font)
pylab.xlabel(u'二叉树步数', fontproperties = font)
pylab.title(u'二叉树算法美式期权', fontproperties = font, fontsize
= 20)

<matplotlib.text.Text at 0x17aae2d0>
```



量化分析师的Python日记【第10天 Q Quant兵器谱 -之偏微分方程1】

> 来源：https://uqer.io/community/share/5530d9f1f9f06c8f3390465a
>
> 从今天开始我们将进入一个系列 —— 偏微分方程。作为这一系列的开篇，我们以热传导方差为引子，引出：
>
> 1. 如何提一个偏微分方程的初边值问题；
> 2. 利用差分格式将偏微分方程离散化；
> 3. 显示差分格式；
> 4. 显示差分格式的条件稳定性。
>
> 最后一点将作为伏笔，引出我们下一天的学习：无条件稳定格式。

1. 热传导方程

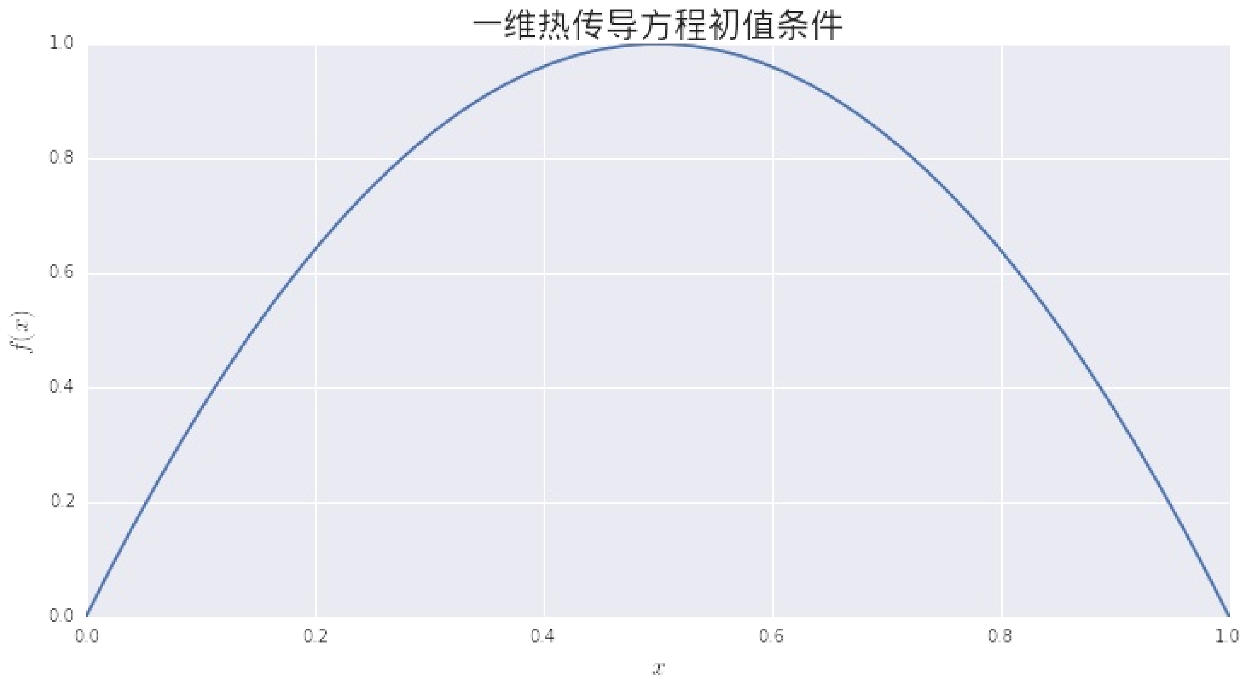
$$\begin{cases} u_\tau - \kappa u_{xx} & = 0, \quad 0 \le x \le 1 \quad [1] \\ u(x,0) & = 4x(1-x), \quad 0 \le x \le 1 \quad [2] \\ u(0,\tau) & = 0, \quad \tau \ge 0 \quad [3] \\ u(1,\tau) & = 0, \quad \tau \ge 0 \quad [4] \end{cases}$$

其中：

- κ 称为热传导系数
- [2] 称为方程的初值条件（Initial Condition)
- [3][4] 称为方程的边值条件（Boundaries Condition）。这里我们使用Dirichlet条件

我们可以看一下初值条件的形状:

```
from matplotlib import pylab
import seaborn as sns
import numpy as np
font.set_size(20)

def initialCondition(x):
    return 4.0*(1.0 - x) * x

xArray = np.linspace(0,1.0,50)
yArray = map(initialCondition, xArray)
pylab.figure(figsize = (12,6))
pylab.plot(xArray, yArray)
pylab.xlabel('$x$', fontsize = 15)
pylab.ylabel('$f(x)$', fontsize = 15)
pylab.title(u'一维热传导方程初值条件', fontproperties = font)

<matplotlib.text.Text at 0x12523810>
```



2. 显式差分格式

这里的基本思想是用差分格式替换对应的微分形式，并且期盼两种格式的"误差"在网格足够密的情况下会趋于0。我们分别在时间方向以及空间方向做差分格式：

$$\frac{\partial u(x_j,\tau_k)}{\partial \tau} = \frac{u_{j,k+1}-u_{j,k}}{\Delta\tau} + O(\Delta\tau)$$

$$\frac{\partial^2 u(x_j,\tau_k)}{\partial x^2} = \frac{u_{j-1,k}-2u_{j,k}+u_{j+1,k}}{\Delta x^2} + O(\Delta x^2)$$

合并在一起，我们就得到了原始微分方程的差分格式：

$$u_\tau(x_j,\tau_k) - \kappa u_{xx}(x_j,\tau_k) = 0$$

$$\frac{u_{j,k+1}-u_{j,k}}{\Delta\tau} - \kappa\frac{u_{j-1,k}-2u_{j,k}+u_{j+1,k}}{\Delta x^2} = O(\Delta\tau) + O(\Delta x^2)$$

这里我们使用差分网格上的近似值`Uj,k`代替`uj,k`，得到新的方程：

$$\begin{aligned}
\frac{U_{j,k+1}-U_{j,k}}{\Delta\tau} - \kappa\frac{U_{j-1,k}-2U_{j,k}+U_{j+1,k}}{\Delta x^2} &= 0,\\
\Rightarrow\quad U_{j,k+1}-U_{j,k}-\frac{\kappa\Delta\tau}{\Delta x^2}(U_{j-1,k}-2U_{j,k}+U_{j+1,k}) &= 0,\\
\Rightarrow\quad U_{j,k+1}-U_{j,k}-\rho(U_{j-1,k}-2U_{j,k}+U_{j+1,k}) &= 0.
\end{aligned}$$

到这里我们得到一个迭代方程组：

$$U_{j,k+1} = \rho U_{j-1,k} + (1-2\rho)U_{j,k} + \rho U_{j+1,k},\quad 1\le j\le N-1,\quad 0\le k\le M-1$$

其中，$\rho = \frac{\kappa\Delta\tau}{\Delta x^2}$。下面我们使用Python代码实现上面的过程。

首先定义基本变量：

- `N` 空间方向的网格数
- `M` 时间方向的网格数
- `T` 最大时间期限
- `X` 最大空间范围
- `U` 用来存储差分网格点上值得矩阵

```
N = 25    # x方向网格数
M = 2500  # t方向网格数

T = 1.0
X = 1.0

xArray = np.linspace(0,X,N+1)
yArray = map(initialCondition, xArray)

starValues = yArray
U = np.zeros((N+1,M+1))
U[:,0] = starValues
```

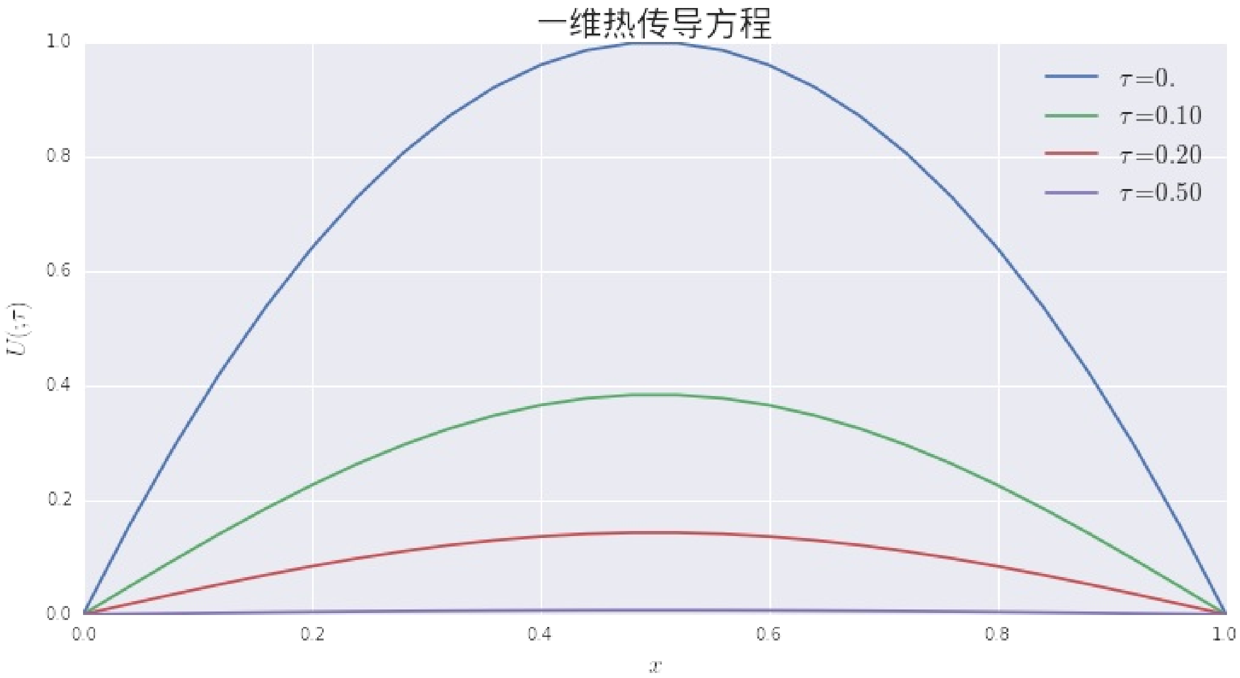
```
dx = X / N
dt = T / M
kappa = 1.0
rho = kappa * dt / dx / dx
```

这里我们做正向迭代：迭代时 `k=0,1...M-1` ，代表我们从0时刻运行至 `T`

```
for k in range(0, M):
    for j in range(1, N):
        U[j][k+1] = rho * U[j-1][k] + (1. - 2*rho) * U[j][k] + r
ho * U[j+1][k]
    U[0][k+1] = 0.
    U[N][k+1] = 0.
```

我们可以画出不同时间点 `U(,˙τk)` 的结果：

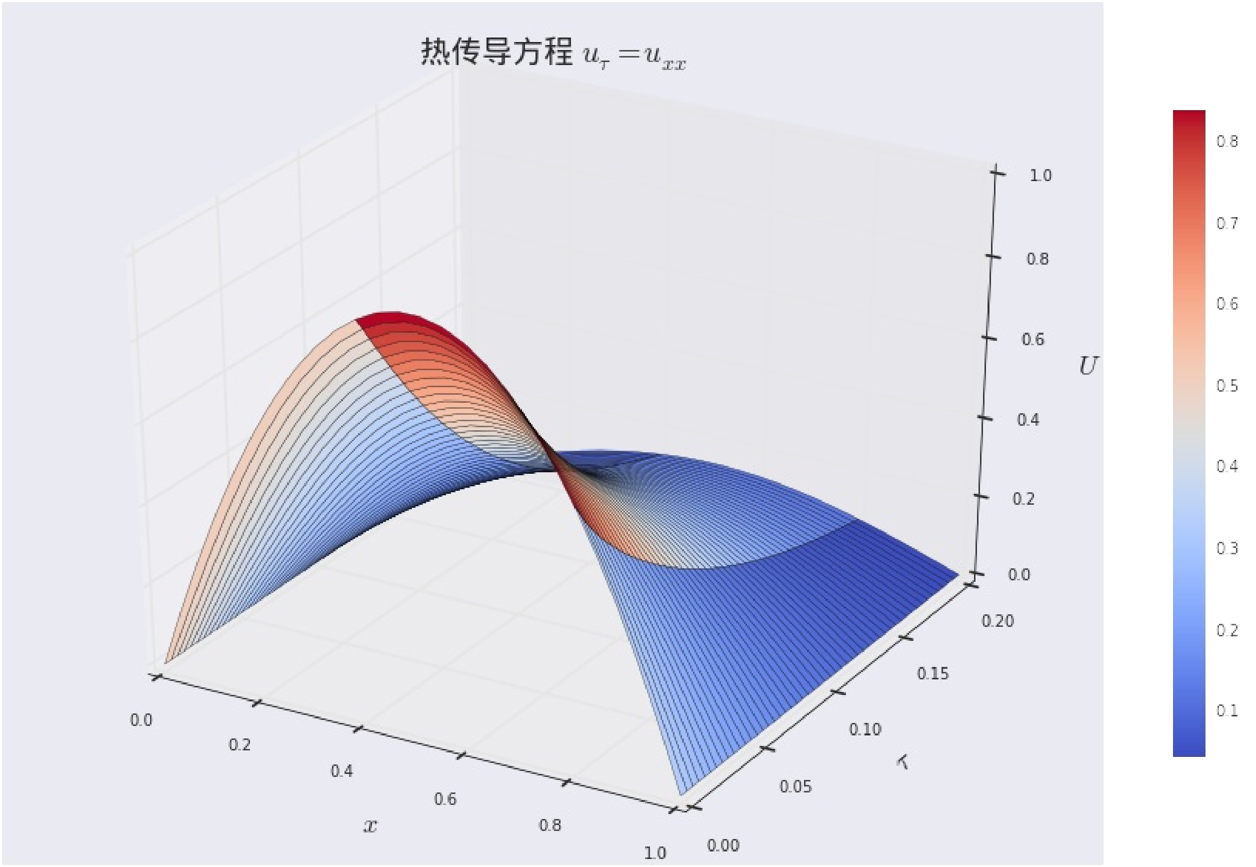
```
pylab.figure(figsize = (12,6))
pylab.plot(xArray, U[:,0])
pylab.plot(xArray, U[:, int(0.10/ dt)])
pylab.plot(xArray, U[:, int(0.20/ dt)])
pylab.plot(xArray, U[:, int(0.50/ dt)])
pylab.xlabel('$x$', fontsize = 15)
pylab.ylabel(r'$U(\dot, \tau)$', fontsize = 15)
pylab.title(u'一维热传导方程', fontproperties = font)
pylab.legend([r'$\tau = 0.$', r'$\tau = 0.10$', r'$\tau = 0.20$'
, r'$\tau = 0.50$'], fontsize = 15)

<matplotlib.legend.Legend at 0x12577cd0>
```



也可以通过三维立体图看一下整体的热传导过程：

```
tArray = np.linspace(0, 0.2, int(0.2 / dt) + 1)
xGrids, tGrids = np.meshgrid(xArray, tArray)
```

```
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm

fig= pylab.figure(figsize = (16,10))
ax = fig.add_subplot(1, 1, 1, projection = '3d')
surface = ax.plot_surface(xGrids, tGrids, U[:,:int(0.2 / dt) + 1
].T, cmap=cm.coolwarm)
ax.set_xlabel("$x$", fontdict={"size":18})
ax.set_ylabel(r"$\tau$", fontdict={"size":18})
ax.set_zlabel(r"$U$", fontdict={"size":18})
ax.set_title(u"热传导方程 $u_\\tau = u_{xx}$" , fontproperties = f
ont)
fig.colorbar(surface,shrink=0.75)

<matplotlib.colorbar.Colorbar instance at 0xf6eb878>
```



3. 组装起来

就像在前一天二叉树建模中介绍的一样，我们这里会以面向对象的方式重新封装分散的代码，方便复用。首先是方程的描述：

```
class HeatEquation:
    def __init__(self, kappa, X, T,
             initialConstion = lambda x:4.0*x*(1.0-x), boundaryConditionL = lambda x: 0, boundaryCondtionR = lambda x:0):
        self.kappa = kappa
        self.ic = initialConstion
        self.bcl = boundaryConditionL
        self.bcr = boundaryCondtionR
        self.X = X
        self.T = T
```

下面的是显式差分格式的描述：

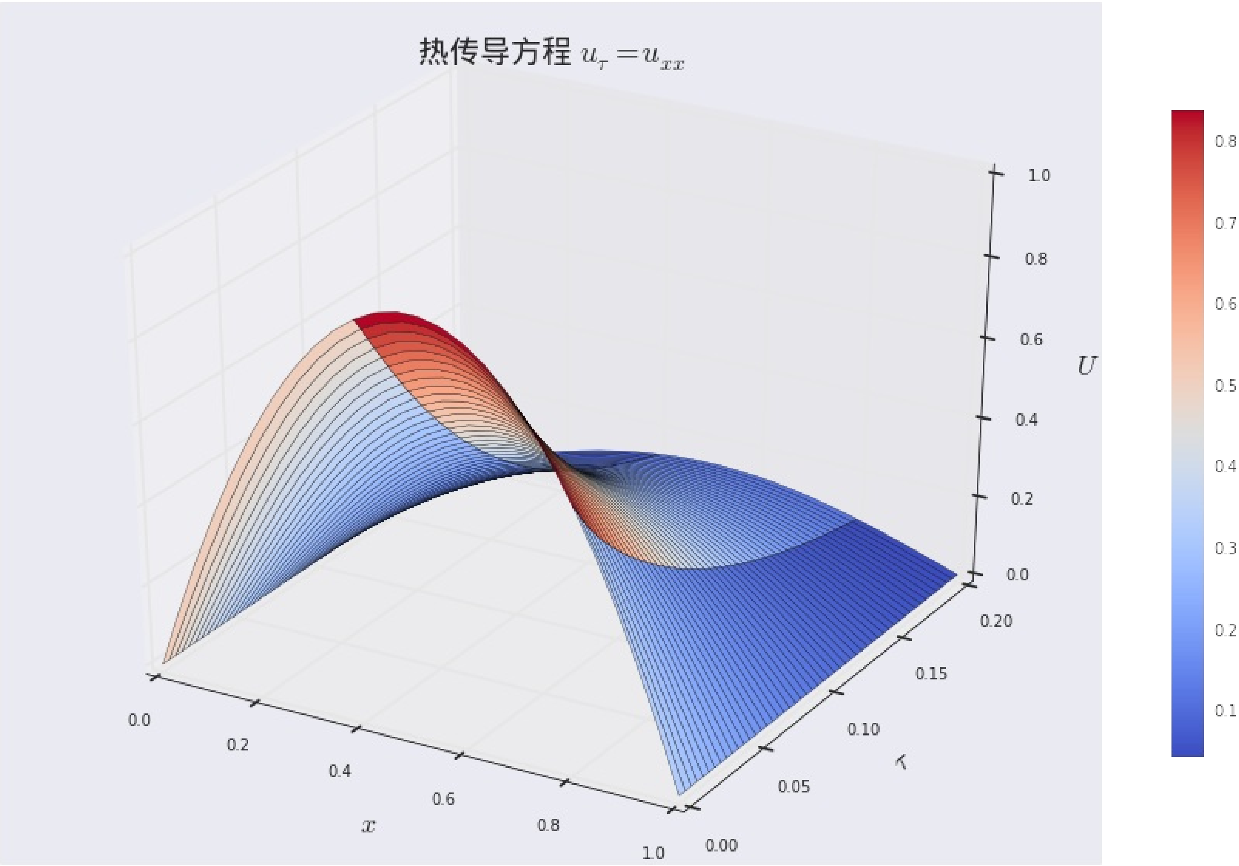
```
class ExplicitEulerScheme:
    def __init__(self, M, N, equation):
        self.eq = equation
        self.dt = self.eq.T / M
        self.dx = self.eq.X / N
        self.U = np.zeros((N+1, M+1))
        self.xArray = np.linspace(0,self.eq.X,N+1)
        self.U[:,0] = map(self.eq.ic, self.xArray)
        self.rho = self.eq.kappa * self.dt / self.dx / self.dx
        self.M = M
        self.N = N

    def roll_back(self):
        for k in range(0, self.M):
            for j in range(1, self.N):
                self.U[j][k+1] = self.rho * self.U[j-1][k] + (1.
 - 2*self.rho) * self.U[j][k] + self.rho * self.U[j+1][k]
        self.U[0][k+1] = self.eq.bcl(self.xArray[0])
        self.U[N][k+1] = self.eq.bcr(self.xArray[-1])

    def mesh_grids(self):
        tArray = np.linspace(0, self.eq.T, M+1)
        tGrids, xGrids = np.meshgrid(tArray, self.xArray)
        return tGrids, xGrids
```

有了以上的部分，现在整个过程可以简单的通过初始化和一行关于 `roll_back` 的调用完成：

```
ht = HeatEquation(1.,1.,1.)
scheme = ExplicitEulerScheme(2500,25, ht)
scheme.roll_back()
```

我们可以获取与之前相同的图像：

```
tGrids, xGrids = scheme.mesh_grids()
fig= pylab.figure(figsize = (16,10))
ax = fig.add_subplot(1, 1, 1, projection = '3d')
cutoff = int(0.2 / scheme.dt) + 1
surface = ax.plot_surface(xGrids[:,:cutoff], tGrids[:,:cutoff],
scheme.U[:,:cutoff], cmap=cm.coolwarm)
ax.set_xlabel("$x$", fontdict={"size":18})
ax.set_ylabel(r"$\tau$", fontdict={"size":18})
ax.set_zlabel(r"$U$", fontdict={"size":18})
ax.set_title(u"热传导方程 $u_\\tau = u_{xx}$" , fontproperties = f
ont)
fig.colorbar(surface,shrink=0.75)

<matplotlib.colorbar.Colorbar instance at 0x12d69e60>
```



4. 什么时候显式格式会失败？

显式格式不能任意取时间和空间的网格点数，即 `M` 与 `N` 不能随意取值。我们称显式格式为条件稳定。特别地，需要满足所谓CFL条件（Courant–Friedrichs–Lewy）：

$$0 \le \rho = \frac{\kappa \Delta \tau}{\Delta x^2} \le 0.5$$

例如：

- `M` = 2500
- `N` = 25

则：

$$\rho = \frac{\kappa \Delta \tau}{\Delta x^2} = 0.25 \le 0.5$$

- `M` = 1200
- `N` = 25

则：

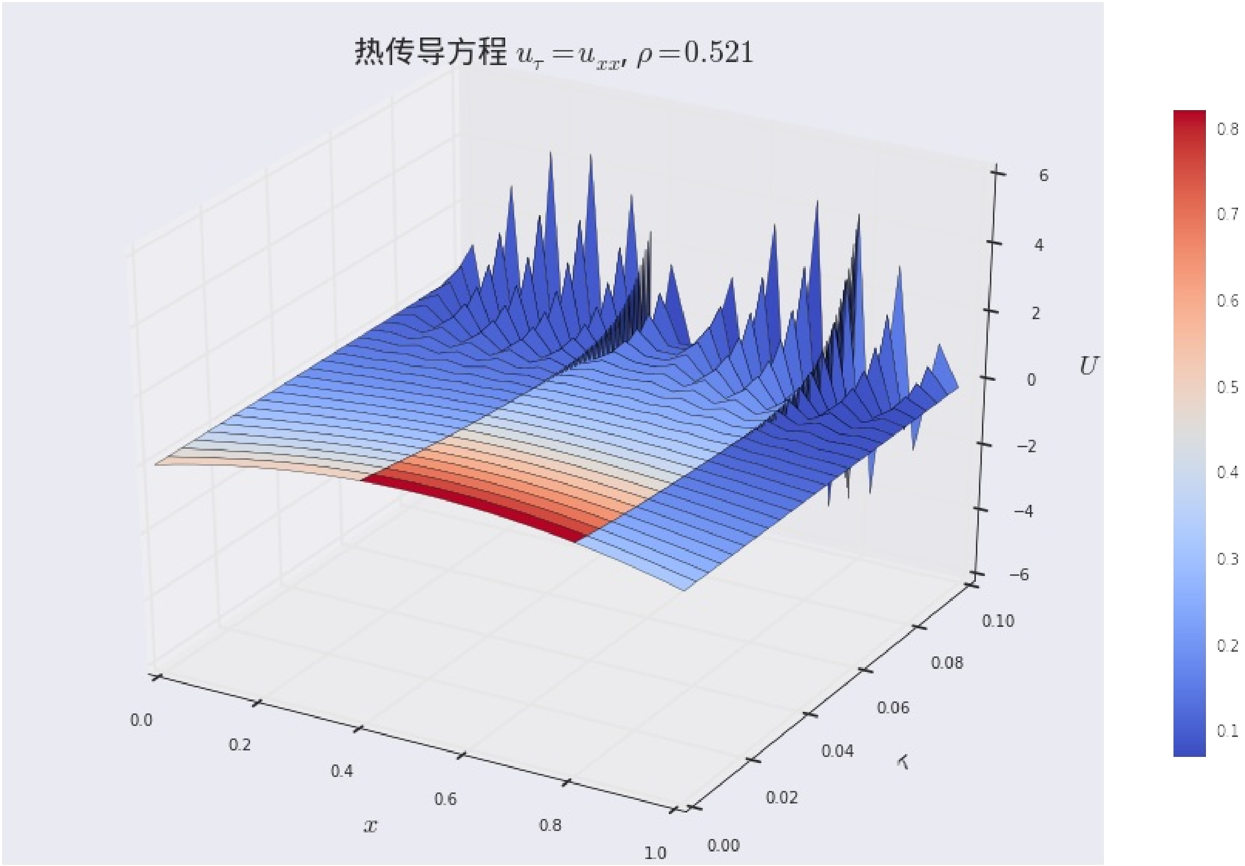
$$\rho = \frac{\kappa \Delta \tau}{\Delta x^2} = 0.521 \geq 0.5$$

下面的代码计算在第二种情形下的网格点计算过程：

```
ht = HeatEquation(1.,1.,1.)
scheme = ExplicitEulerScheme(1200,25, ht)
scheme.roll_back()
```

我们可以通过下图看到，在CFL条件无法满足的情况下，数值误差累计的结果（特别注意后面的锯齿）：

```
tGrids, xGrids = scheme.mesh_grids()
fig= pylab.figure(figsize = (16,10))
ax = fig.add_subplot(1, 1, 1, projection = '3d')
cutoff = int(0.2 / scheme.dt) + 1
surface = ax.plot_surface(xGrids[:,:cutoff], tGrids[:,:cutoff],
scheme.U[:,:cutoff], cmap=cm.coolwarm)
ax.set_xlabel("$x$", fontdict={"size":18})
ax.set_ylabel(r"$\tau$", fontdict={"size":18})
ax.set_zlabel(r"$U$", fontdict={"size":18})
ax.set_title(u"热传导方程 $u_\\tau = u_{xx}$, $\\rho = 0.521$" , f
ontproperties = font)
fig.colorbar(surface,shrink=0.75)

<matplotlib.colorbar.Colorbar instance at 0x10f51b48>
```



今天的日记到此为止，这个问题我们会在下一篇中进行讨论，引出无条件稳定格式：隐式差分格式（Implicit）。

量化分析师的Python日记【第11天 Q Quant兵器谱之偏微分方程2】

> 来源：https://uqer.io/community/share/5534ad3ff9f06c8f33904689
>
> 这是量化分析师的偏微分方程系列的第二篇，在这一篇中我们将解决上一篇显式格式留下的稳定性问题。本篇将引入隐式差分算法，读者可以学到：
>
> 1. 隐式差分格式描述
> 2. 三对角矩阵求解
> 3. 如何使用 `scipy` 加速算法实现
>
> 在完成两天的基础学习之后，在下一天中，我们将把已经学到的知识运用到金融定价领域最重要的方程之一：Black - Shcoles - Merton 偏微分方差

```
from matplotlib import pylab
import seaborn as sns
import numpy as np
np.set_printoptions(precision = 4)
font.set_size(20)

def initialCondition(x):
    return 4.0*(1.0 - x) * x
```

1. 隐式差分格式

像上一天一样，我们从差分格式的数学表述开始。隐式格式与显式格式的区别，在于我们时间方向选择的基准点。显式格式使用 `k` ，而隐式格式选择 `k+1` :

$$\frac{\partial u(x_j,\tau_{k+1})}{\partial \tau} = \frac{u_{j,k+1}-u_{j,k}}{\Delta\tau} + O(\Delta\tau)$$

$$\frac{\partial^2 u(x_j,\tau_{k+1})}{\partial x^2} = \frac{u_{j-1,k+1}-2u_{j,k+1}+u_{j+1,k+1}}{\Delta x^2} + O(\Delta x^2)$$

剩下的推到过程我完全一样，我们看到无论隐式格式还是显式格式，它们的截断误差是一样的：

$$u_\tau(x_j,\tau_{k+1}) - \kappa u_{xx}(x_j,\tau_{k+1}) = 0$$

$$\frac{u_{j,k+1}-u_{j,k}}{\Delta\tau} - \kappa\frac{u_{j-1,k+1}-2u_{j,k+1}+u_{j+1,k+1}}{\Delta x^2} = O(\Delta\tau)+O(\Delta x^2)$$

用离散值`Uj,k`替换`uj,k`，我们得到差分方程：

$$
\begin{aligned}
\frac{U_{j,k+1}-U_{j,k}}{\Delta\tau}-\kappa\frac{U_{j-1,k+1}-2U_{j,k+1}+U_{j+1,k+1}}{\Delta x^2} &= 0, \\
\Rightarrow\quad U_{j,k+1}-U_{j,k}-\frac{\kappa\Delta\tau}{\Delta x^2}\left(U_{j-1,k+1}-2U_{j,k+1}+U_{j+1,+1k}\right) &= 0, \\
\Rightarrow\quad U_{j,k+1}-U_{j,k}-\rho(U_{j-1,k+1}-2U_{j,k+1}+U_{j+1,k+1}) &= 0.
\end{aligned}
$$

最后，到这里我们得到一个迭代方程组：

$$
-\rho U_{j-1,k+1}+(1+2\rho)U_{j,k+1}-\rho U_{j+1,k+1}=U_{j,k},\quad 1\leq j\leq N-1,\quad 0\leq k\leq M-1
$$

其中 $\rho=\frac{\kappa\Delta\tau}{\Delta x^2}$ 。

```
N = 500  # x方向网格数
M = 500  # t方向网格数

T = 1.0
X = 1.0

xArray = np.linspace(0,X,N+1)
yArray = map(initialCondition, xArray)

starValues = yArray
U = np.zeros((N+1,M+1))
U[:,0] = starValues
```

```
dx = X / N
dt = T / M
kappa = 1.0
rho = kappa * dt / dx / dx
```

1.1 矩阵求解(`TridiagonalSystem`)

虽然看上去形式只是变了一点，但是求解的问题有很大的变化。在每个时间点上，我们需要求解如下的一个线性方程组：

$$
\mathbf{A}U_{k+1}=U_k
$$

这里`A`为：

$$\mathbf{A} = \begin{pmatrix} 1+2\rho & -\rho & \cdots & 0 \\ -\rho & 1+2\rho & -\rho & \cdots \\ & \ddots & \ddots & -\rho \\ 0 & \cdots & -\rho & 1+2\rho \end{pmatrix}.$$

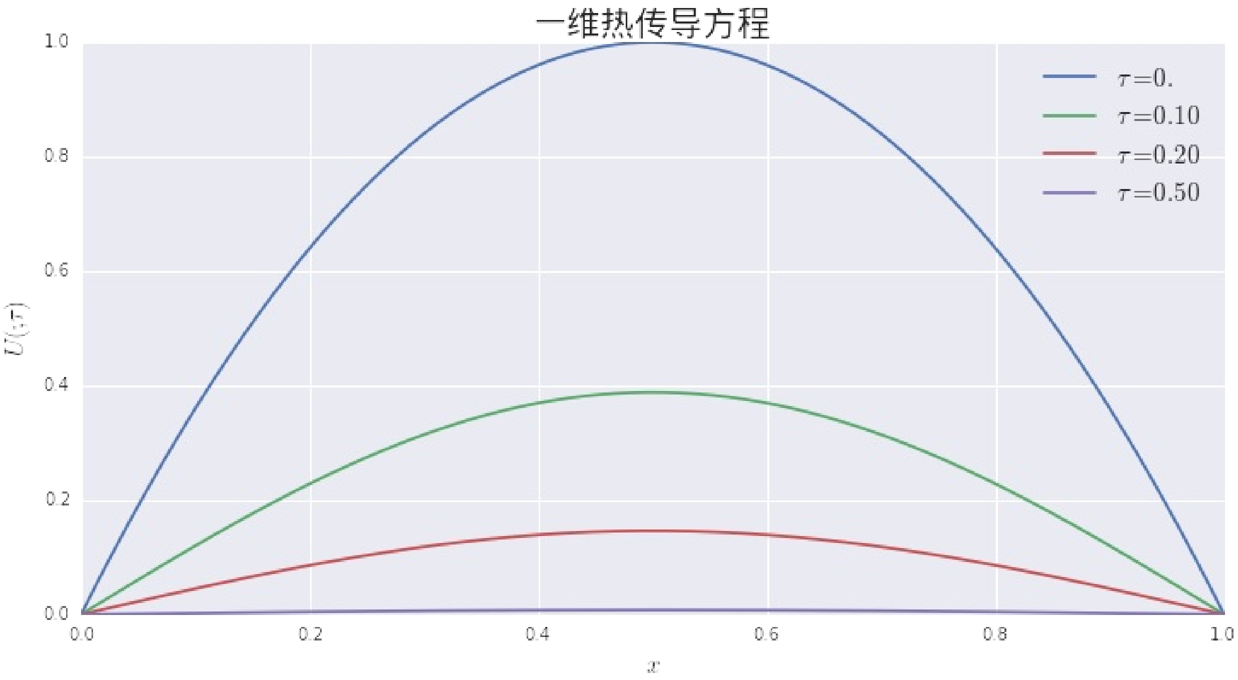
幸运的是，这个是个三对角矩阵，可以很简单的利用Gauss消去法求解。我们这里不会详细讨论算法的描述，细节都可以在下面的python类 `TridiagonalSystem` 中了解到：

```
class TridiagonalSystem:
    def __init__(self, udiag, cdiag, ldiag):
        '''
        三对角矩阵：
        udiag -- 上对角线
        cdiag -- 对角线
        ldiag -- 下对角线
        '''
        assert len(udiag) == len(cdiag)
        assert len(cdiag) == len(ldiag)
        self.udiag = udiag
        self.cdiag = cdiag
        self.ldiag = ldiag
        self.length = len(self.cdiag)

    def solve(self, rhs):
        '''
        求解以下方程组
        A \ dot x = rhs
        '''
        assert len(rhs) == len(self.cdiag)
        udiag = self.udiag.copy()
        cdiag = self.cdiag.copy()
        ldiag = self.ldiag.copy()
        b = rhs.copy()

        # 消去下对角元
        for i in range(1, self.length):
            cdiag[i] -=  udiag[i-1] * ldiag[i] / cdiag[i-1]
            b[i] -= b[i-1] * ldiag[i] / cdiag[i-1]

        # 从最后一个方程开始求解
        x = np.zeros(self.length)
        x[self.length-1] = b[self.length - 1] / cdiag[self.lengt
h - 1]
        for i in range(self.length - 2, -1, -1):
            x[i] = (b[i] - udiag[i]*x[i+1]) / cdiag[i]
        return x

    def multiply(self, x):
        '''
        矩阵乘法：
        rhs = A \dot x
        '''
        assert len(x) == len(self.cdiag)
        rhs = np.zeros(self.length)
        rhs[0] = x[0] * self.cdiag[0] + x[1] * self.udiag[0]
        for i in range(1, self.length - 1):
            rhs[i] =  x[i-1] * self.ldiag[i] + x[i] * self.cdiag
[i] + x[i+1] * self.udiag[i]
        rhs[self.length - 1] = x[self.length - 2] * self.ldiag[s
elf.length - 1] + x[self.length - 1] * self.cdiag[self.length - 1
]
        return rhs
```

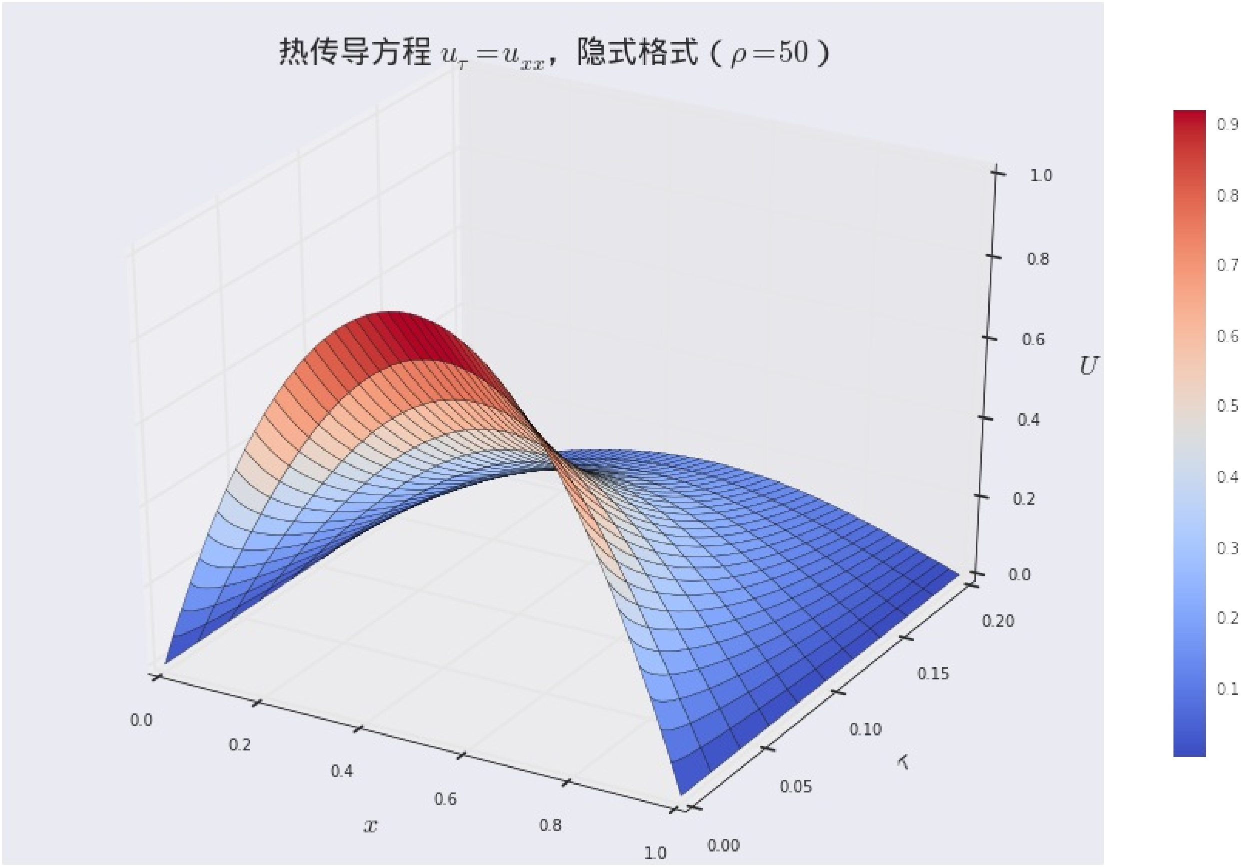
1.2 隐式格式求解

```
for k in range(0, M):
    udiag = - np.ones(N-1) * rho
    ldiag =  - np.ones(N-1) * rho
    cdiag =  np.ones(N-1) * (1.0 + 2. * rho)

    mat = TridiagonalSystem(udiag, cdiag, ldiag)
    rhs = U[1:N,k]
    x = mat.solve(rhs)
    U[1:N, k+1] = x
    U[0][k+1] = 0.
    U[N][k+1] = 0.
```

```
from lib.utilities import plotLines
plotLines([U[:,0], U[:, int(0.10/ dt)], U[:, int(0.20/ dt)], U[:
, int(0.50/ dt)]], xArray, title = u'一维热传导方程', xlabel = '$x
$',
          ylabel = r'$U(\dot, \tau)$', legend = [r'$\tau = 0.$',
r'$\tau = 0.10$', r'$\tau = 0.20$', r'$\tau = 0.50$'])
```



```
from lib.utilities import plotSurface
tArray = np.linspace(0, 0.2, int(0.2 / dt) + 1)
tGrids, xGrids = np.meshgrid(tArray, xArray)

plotSurface(xGrids, tGrids, U[:,:int(0.2 / dt) + 1], title = u"
热传导方程 $u_\\tau = u_{xx}$，隐式格式（$\\rho = 50$）", xlabel = "
$x$", ylabel = r"$\tau$", zlabel = r"$U$")
```



2. 继续组装

像我们在显示格式那一节介绍的同样做法，我们把之前的代码整合起来，归集与一个完整的类 `ImplicitEulerScheme` 中：

```python
from lib.utilities import HeatEquation
```

上面的代码（使用 `library` 功能，关于该功能的具体介绍请见帮助 — Library是干什么的)导入我们在上一期中已经定义过的类 `HeatEquation` ，避免代码重复。

```python
class ImplicitEulerScheme:
    def __init__(self, M, N, equation):
        self.eq = equation
        self.dt = self.eq.T / M
        self.dx = self.eq.X / N
        self.U = np.zeros((N+1, M+1))
        self.xArray = np.linspace(0,self.eq.X,N+1)
        self.U[:,0] = map(self.eq.ic, self.xArray)
        self.rho = self.eq.kappa * self.dt / self.dx / self.dx
        self.M = M
        self.N = N

    def roll_back(self):
        for k in range(0, self.M):
            udiag = - np.ones(self.N-1) * self.rho
            ldiag =  - np.ones(self.N-1) * self.rho
            cdiag =  np.ones(self.N-1) * (1.0 + 2. * self.rho)

            mat = TridiagonalSystem(udiag, cdiag, ldiag)
            rhs = self.U[1:self.N,k]
            x = mat.solve(rhs)
            self.U[1:self.N, k+1] = x
            self.U[0][k+1] = self.eq.bcl(self.xArray[0])
            self.U[self.N][k+1] = self.eq.bcr(self.xArray[-1])

    def mesh_grids(self):
        tArray = np.linspace(0, self.eq.T, M+1)
        tGrids, xGrids = np.meshgrid(tArray, self.xArray)
        return tGrids, xGrids
```

然后我们可以使用下面的三行简单调用完成功能：

```
ht = HeatEquation(1.,X, T)
scheme = ImplicitEulerScheme(M,N, ht)
scheme.roll_back()
scheme.U

array([[  0.0000e+00,   0.0000e+00,   0.0000e+00, ...,   0.0000e
+00,
          0.0000e+00,   0.0000e+00],
       [  7.9840e-03,   7.2843e-03,   6.9266e-03, ...,   3.8398e
-07,
          3.7655e-07,   3.6926e-07],
       [  1.5936e-02,   1.4567e-02,   1.3852e-02, ...,   7.6795e
-07,
          7.5308e-07,   7.3851e-07],
       ...,
       [  1.5936e-02,   1.4567e-02,   1.3852e-02, ...,   7.6795e
-07,
          7.5308e-07,   7.3851e-07],
       [  7.9840e-03,   7.2843e-03,   6.9266e-03, ...,   3.8398e
-07,
          3.7655e-07,   3.6926e-07],
       [  0.0000e+00,   0.0000e+00,   0.0000e+00, ...,   0.0000e
+00,
          0.0000e+00,   0.0000e+00]])
```

3. 使用 `scipy` 加速

软件工程行业里有句老话，叫做："不要重复发明轮子！"。实际上，之前的代码里面，我们就造了自己的轮子： `TridiagonalSystem` 。三对角矩阵作为最最常见的稀疏矩阵，关于它的线性方程组求解算法实际上早已为业界熟知，也已经有很多库内置了工业级别强度实现。这里我们取 `scipy` 作为例子，来展示使用外源库实现的好处：

- 更加稳健的算法： 知名库算法由于使用者广泛，有更大的概率发现一些极端情形下的bug。库作者可以根据用户反馈，及时调整算法；
- 更高的性能： 由于库的使用更为广泛，库作者有更大的动力去使用各种技术去提高算法的性能：例如使用更高效的语言实现，例如C。scipy中的情形就是一例。
- 持续的维护： 库的受众范围广，社区的力量会推动库作者持续维护。

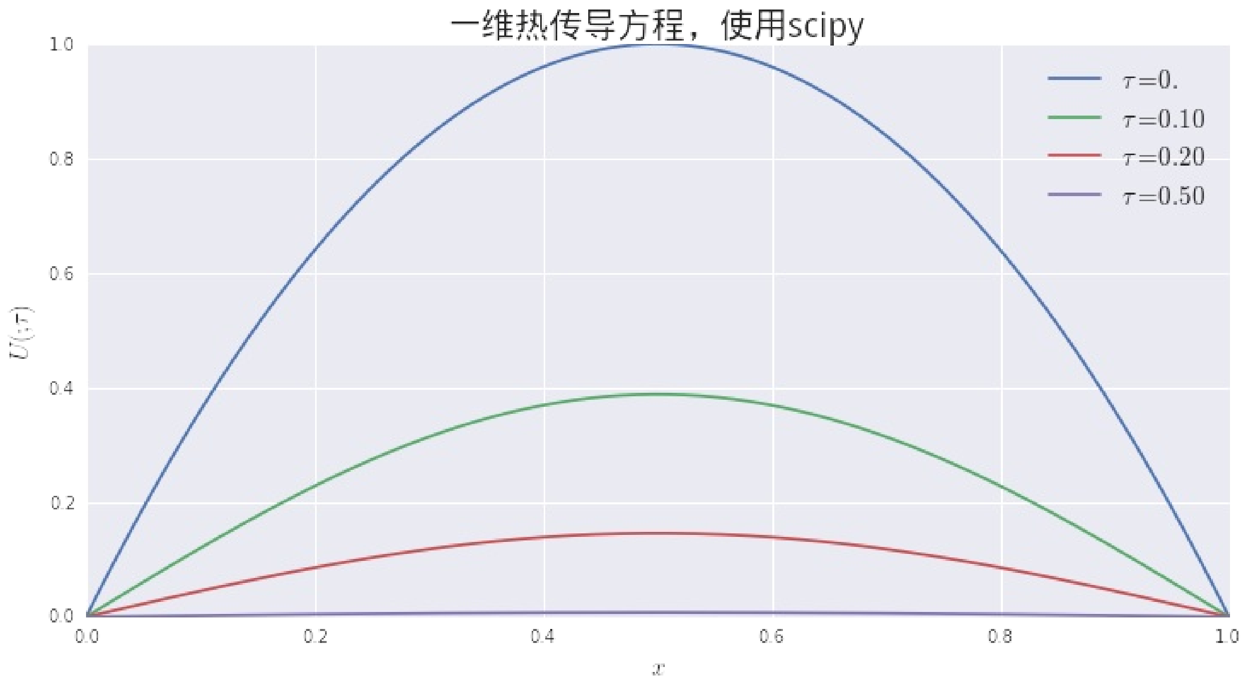
下面的代码展示，如何使用 `scipy` 中的 `solve_banded` 算法求解三对角矩阵：

```
import scipy as sp
from scipy.linalg import solve_banded

A = np.zeros((3, 5))
A[0, :] = np.ones(5) * 1.   # 上对角线
A[1, :] = np.ones(5) * 3.   # 对角线
A[2, :] = np.ones(5) * (-1.) # 下对角线

b = [1.,2.,3.,4.,5.]
x = solve_banded ((1,1), A,b)
print 'x = A^-1b = ',x

x = A^-1b =  [ 0.1833  0.45    0.8333  0.95    1.9833]
```

我们使用上面的算法替代我们之前的 `TridiagonalSystem` ，

```
import scipy as sp
from scipy.linalg import solve_banded

for k in range(0, M):
    udiag = - np.ones(N-1) * rho
    ldiag =  - np.ones(N-1) * rho
    cdiag =  np.ones(N-1) * (1.0 + 2. * rho)
    mat = np.zeros((3,N-1))
    mat[0,:] = udiag
    mat[1,:] = cdiag
    mat[2,:] = ldiag
    rhs = U[1:N,k]
    x = solve_banded ((1,1), mat,rhs)
    U[1:N, k+1] = x
    U[0][k+1] = 0.
    U[N][k+1] = 0.
```

```
plotLines([U[:,0], U[:, int(0.10/ dt)], U[:, int(0.20/ dt)], U[:, int(0.50/ dt)]], xArray, title = u'一维热传导方程，使用scipy', xlabel = '$x$',
          ylabel = r'$U(\dot, \tau)$', legend = [r'$\tau = 0.$', r'$\tau = 0.10$', r'$\tau = 0.20$', r'$\tau = 0.50$'])
```



同样的我们定义一个新类`ImplicitEulerSchemeWithScipy`使用`scipy`的算法：

```
class ImplicitEulerSchemeWithScipy:
    def __init__(self, M, N, equation):
        self.eq = equation
        self.dt = self.eq.T / M
        self.dx = self.eq.X / N
        self.U = np.zeros((N+1, M+1))
        self.xArray = np.linspace(0,self.eq.X,N+1)
        self.U[:,0] = map(self.eq.ic, self.xArray)
        self.rho = self.eq.kappa * self.dt / self.dx / self.dx
        self.M = M
        self.N = N

    def roll_back(self):
        for k in range(0, self.M):
            udiag = - np.ones(self.N-1) * self.rho
            ldiag =  - np.ones(self.N-1) * self.rho
            cdiag =  np.ones(self.N-1) * (1.0 + 2. * self.rho)

            mat = np.zeros((3,self.N-1))
            mat[0,:] = udiag
            mat[1,:] = cdiag
            mat[2,:] = ldiag
            rhs = self.U[1:self.N,k]
            x = solve_banded((1,1), mat, rhs)
            self.U[1:self.N, k+1] = x
            self.U[0][k+1] = self.eq.bcl(self.xArray[0])
            self.U[self.N][k+1] = self.eq.bcr(self.xArray[-1])

    def mesh_grids(self):
        tArray = np.linspace(0, self.eq.T, M+1)
        tGrids, xGrids = np.meshgrid(tArray, self.xArray)
        return tGrids, xGrids
```

下面的代码，比较了两种做法的性能。可以看到仅仅简单的替代三对角矩阵算法，我们就获得了接近8倍的性能提升

```
import time
startTime = time.time()
loop_round = 10

# 不使用scipy
for k in range(loop_round):
    ht = HeatEquation(1.,X, T)
    scheme = ImplicitEulerScheme(M,N, ht)
    scheme.roll_back()
endTime = time.time()
print '{0:<40}{1:.4f}'.format('执行时间(s) -- 不使用scipy.linalg: '
, endTime - startTime)

# 使用scipy
startTime = time.time()
for k in range(loop_round):
    ht = HeatEquation(1.,X, T)
    scheme = ImplicitEulerSchemeWithScipy(M,N, ht)
    scheme.roll_back()
endTime = time.time()
print '{0:<40}{1:.4f}'.format('执行时间(s) -- 使用scipy.linalg: ',
 endTime - startTime)

执行时间(s) -- 不使用scipy.linalg: 12.1589
执行时间(s) -- 使用scipy.linalg:  1.6224
```

4. 尾声

到这里为止，我们已经结束了偏微分方差差分格式的基础学习。这是一个很大的学科，这两天也只能做到“管中窥豹”。但是有了以上的基础知识，读者已经有了足够的积累，可以处理一些金融工程中会实际遇到的方程。在下一天中，我们将把这两天学习到的知识运用到金融工程史上最重要的方程：Black - Scholes - Merton 偏微分方程。

量化分析师的Python日记【第12天：量化入门进阶之葵花宝典：因子如何产生和回测】

> 来源：https://uqer.io/community/share/5538c7d4f9f06c3c92306684

0 预备知识

预备知识包括：数学，计算机，投资学

数学方面至少包括微积分，线性代数，优化理论，概率统计基础，线性回归等知识点。数学出生最佳，一般理工科都基本满足要求，即使有所欠缺，花点时间也就自学补上了

计算机主要有两点：一是会编程；二是会数据分析，相信在量化实验室看到截止今天日记的同学都已经满足了

投资学方面只要通过大学的《投资学》课程就好，像William Sharpe等3人合著的《投资学》，要是能够通过CFA那就最好，知识面更广

1 入门阶段

Barra USE3 handbook

Barra是量化投资技术提供商,是量化投资先驱。其经典的美国股票风险模型第3版（USE3）手册，详细介绍了股票市场多因子模型的理论框架和实证细节。手册共几十页，不太长，描述规范清晰，不陷入无意义的细节，非常适合于入门, 点此下载

2 系统学习阶段

系统学习1：Quantitative Equity Portfolio Management（QEPM），Ludwig Chincarini 偏学术风格。

偏学术界的作者撰写的关于量化股票组合投资的系统教程。尤其是前几章概述部分写得非常精彩、易懂、准确。把该领域的各个方面高屋建瓴地串讲了一遍。后面部分的章节似乎略有些学术了，但也值得一读。由于其较高的可读性，适于初学者学习。

系统学习2：Active Portfolio Management（APM），Grinold & Kahn 偏业界风格。

业界先驱所著，作者均曾任Barra公司的研究总监。本书深度相对较深，描述也偏实践，介绍了许多深刻的真知。并且书中很多论述精彩而透彻。该书被奉为量化组合投资业界圣经。不过该书有些章节撰写得深度不一，初学者容易感到阅读起来有

点困难。所以推荐：首次阅读不必纠结看不懂的细节，只要不影响后续阅读就跳过具体细节；有一定基础后，建议经常反复阅读本书。

系统学习3：Quantitative Equity Portfolio Management（QEPM）， Qian & Hua & Sorensen APM的补充

业界人士所著。针对性地对APM没有展开讲的一些topic做了很好的深入探讨。建议在APM之后阅读。该书风格比较数学，不过对数学专业背景的人并不太难。撰写文字也比较流畅。

注：修行上述3本葵花宝典是否要割舍些什么？主要是与亲友坐在一起聊天喝茶的时光、一些睡觉的时间以及购书需要上千元钱（建议读英文原著）；好消息是，练成之后，不仅钱可以赚回来，空闲时间也会多起来。

3 实践阶段

券商卖方金工研究报告：多因子模型、选股策略、择时策略

系统学习上面的材料之后，你已经有了分辨能力，这是看数量众多的券商卖方金工研究报告，就可以庖丁解牛，分辨真伪，总能筛选出优质信息积累下来了。

值得总结的是数学、计算机、分析框架等工具都只是量化投资的形，优质投资想法才是灵魂。所以在修炼上述量化投资的基本功的同时，请不要忘记向有洞察力、有独立思考的其它派系的投资专家学习，无论他/她是价值投资、成长投资、涨停板敢死队、技术分析、主题投资、逆向投资、各类套利。将你自己想出的或者从别人那里习得的投资想法，用量化框架验证、改进、去伪存真，并最终上实盘创造价值。

最推荐的入行过程：学习上述材料的同时，在通联量化实验室利用海量数据编程实现，理论付诸实践！

4 实战操作示例

在关于 `pandas` 的前两篇介绍中，我们已经接触了不少关于 `Series` 和 `DataFrame` 的操作以及函数。本篇将以实际的例子来介绍 `pandas` 在处理实际金融数据时的应用。

因子选股是股票投资中最常用的一种分析手段，利用量化计算的因子从成百上千的股票中进行快速筛选，帮助投资者从海量的数据中快速确定符合要求的目标，以下我们以量化因子计算过程的实例来展示如何利用pandas处理数据。

首先，我们依然是导入需要的一些外部模块：

```
import numpy as np
import pandas as pd
import datetime as dt
from pandas import Series, DataFrame, isnull
from datetime import timedelta, datetime
from CAL.PyCAL import *

pd.set_option('display.width', 200)
```

接着我们定义股票池和计算时所需要的时间区间参数。通常而言，计算某个因子是基于全A股的，这里作为示例，以HS300作为股票池。以计算市净率（PB）为例，我们取近一年的数据：

```
universe = set_universe('HS300')

today = Date.todaysDate()
start_date = (today - Period('1Y')).toDateTime().strftime('%Y%m%d')
end_date = today.toDateTime().strftime('%Y%m%d')
print 'start_date'
print start_date
print 'end_date'
print end_date

start_date
20150714
end_date
20160714
```

市净率是每股市价(Price)和每股净资产(Book Value)的比值，计算时通常使用总市值和归属于母公司所有者权益合计之比得到。前者通过访问股票日行情数据可以获得，后者在资产负债表上能够查到。在量化实验室中提供了访问股票日行情和资产负债表的API，可以获得相应数据。需要注意的一点是在获取财务报表数据时，因为只能指定一种类型的财报（季报，半年报，年报），需要做一个循环查询，并将获取到的 `DataFrame` 数据按垂直方向拼接，这里使用了 `concat` 函数：

```
market_capital = DataAPI.MktEqudGet(secID=universe, field=['secID', 'tradeDate', 'marketValue', 'negMarketValue'], beginDate=start_date, endDate=end_date, pandas='1')

equity = DataFrame()
for rpt_type in ['Q1', 'S1', 'Q3', 'A']:
    try:
        tmp = DataAPI.FdmtBSGet(secID=universe, field=['secID', 'endDate', 'publishDate', 'TEquityAttrP'], beginDate=start_date, publishDateEnd=end_date,  reportType=rpt_type)
    except:
        tmp = DataFrame()
    equity = pd.concat([equity, tmp], axis=0)

print 'Data of TEquityAttrP:'
print equity.head()
print 'Data of marketValue:'
print market_capital.head()
```

```
Data of TEquityAttrP:
         secID     endDate publishDate  TEquityAttrP
0  000001.XSHE  2016-03-31  2016-04-21  1.875690e+11
1  000002.XSHE  2016-03-31  2016-04-28  1.006367e+11
2  000009.XSHE  2016-03-31  2016-04-29  4.467273e+09
3  000039.XSHE  2016-03-31  2016-04-29  5.668081e+09
4  000060.XSHE  2016-03-31  2016-04-28  7.340159e+09
Data of marketValue:
         secID   tradeDate   marketValue  negMarketValue
0  000001.XSHE  2015-07-14  1.984613e+11    1.637222e+11
1  000001.XSHE  2015-07-15  1.943118e+11    1.602991e+11
2  000001.XSHE  2015-07-16  1.945980e+11    1.605351e+11
3  000001.XSHE  2015-07-17  1.977459e+11    1.631320e+11
4  000001.XSHE  2015-07-20  1.945980e+11    1.605351e+11
```

对于市值的数据，每个交易日均有提供，实际上我们多取了数据，我们只需要最新的市值数据即可。为此，我们将数据按股票代码和交易日进行排序，并按股票代码丢弃重复数据。以下代码表示按股票代码和交易日进行升序排序，并在丢弃重复值时，保留最后一个（默认是第一个）：

```
market_capital = market_capital.sort(columns=['secID', 'tradeDate'], ascending=[True, True])
market_capital = market_capital.drop_duplicates(subset='secID', take_last=True)
```

并非所有的数据都是完美的，有时候也会出现数据的缺失。我们在计算时无法处理缺失的数据，需要丢弃。下面这一行代码使用了 `isnull` 函数检查数据中总市值的缺失值，返回的是一个等长的逻辑 `Series` ，若数据缺失则为 `True` 。为尽可能多利用数据，我们考虑在总市值缺失的情况下，若流通市值有数值，则使用流通市

值替换总市值，仅在两者皆缺失的情况下丢弃数据（虽然多数情况下是流通市值缺失，有总市值的数据，这一处理方式在其它使用到流通市值计算的情形中可以参考）：

```
market_capital['marketValue'][isnull(market_capital['marketValue'])] = market_capital['negMarketValue'][isnull(market_capital['marketValue'])]
```

以下代码使用 `drop` 函数舍去了流通市值这一列，使用 `dropna` 函数丢弃缺失值，并使用 `rename` 函数将列 `marketValue` 重命名为 `numerator` ：

```
market_capital = market_capital.drop('negMarketValue', axis=1)
numerator = market_capital.dropna()
numerator.rename(columns={'marketValue': 'numerator'}, inplace=True)
```

我们可以看一下处理好的分子：

```
print numerator

             secID   tradeDate     numerator
244    000001.XSHE  2016-07-13  1.543620e+11
489    000002.XSHE  2016-07-13  2.022373e+11
734    000009.XSHE  2016-07-13  2.257608e+10
49244  000027.XSHE  2016-07-13  2.755322e+10
979    000039.XSHE  2016-07-13  4.434777e+10
49489  000046.XSHE  2016-07-13  5.222182e+10
1224   000060.XSHE  2016-07-13  2.739233e+10
1469   000061.XSHE  2016-07-13  2.161932e+10
1714   000063.XSHE  2016-07-13  6.184518e+10
1959   000069.XSHE  2016-07-13  5.743977e+10
24744  000100.XSHE  2016-07-13  4.335857e+10
24989  000156.XSHE  2016-07-13  2.833737e+10
2204   000157.XSHE  2016-07-13  3.287913e+10
49734  000166.XSHE  2016-07-13  1.291645e+11
25234  000333.XSHE  2016-07-13  1.750697e+11
2449   000338.XSHE  2016-07-13  3.266872e+10
2694   000402.XSHE  2016-07-13  3.000886e+10
49979  000413.XSHE  2016-07-13  3.831165e+10
50224  000415.XSHE  2016-07-13  4.217844e+10
2939   000423.XSHE  2016-07-13  3.700454e+10
3184   000425.XSHE  2016-07-13  2.249481e+10
50469  000503.XSHE  2016-07-13  3.908079e+10
3429   000538.XSHE  2016-07-13  7.392896e+10
50714  000540.XSHE  2016-07-13  3.147026e+10
50959  000559.XSHE  2016-07-13  3.634171e+10
3674   000568.XSHE  2016-07-13  4.571343e+10
25479  000623.XSHE  2016-07-13  2.298707e+10
```

```
3919   000625.XSHE  2016-07-13  7.134216e+10
4164   000630.XSHE  2016-07-13  2.753465e+10
4409   000651.XSHE  2016-07-13  1.065987e+11
...            ...         ...           ...
45079  601669.XSHG  2016-07-13  8.307799e+10
45324  601688.XSHG  2016-07-13  1.519940e+11
71425  601718.XSHG  2016-07-13  3.189739e+10
71670  601727.XSHG  2016-07-13  1.034922e+11
45569  601766.XSHG  2016-07-13  2.573330e+11
71915  601788.XSHG  2016-07-13  7.094565e+10
45814  601800.XSHG  2016-07-13  1.788926e+11
23764  601808.XSHG  2016-07-13  6.083780e+10
46059  601818.XSHG  2016-07-13  1.750466e+11
24009  601857.XSHG  2016-07-13  1.356185e+12
46304  601866.XSHG  2016-07-13  4.977011e+10
72160  601872.XSHG  2016-07-13  2.771617e+10
46549  601888.XSHG  2016-07-13  4.459454e+10
46794  601898.XSHG  2016-07-13  7.623731e+10
47039  601899.XSHG  2016-07-13  8.466495e+10
47284  601901.XSHG  2016-07-13  6.478664e+10
72405  601919.XSHG  2016-07-13  5.527004e+10
47529  601928.XSHG  2016-07-13  2.842653e+10
47774  601933.XSHG  2016-07-13  3.644512e+10
48019  601939.XSHG  2016-07-13  1.265056e+12
48264  601958.XSHG  2016-07-13  2.913624e+10
72650  601985.XSHG  2016-07-13  1.092693e+11
24254  601988.XSHG  2016-07-13  9.832552e+11
48509  601989.XSHG  2016-07-13  1.327548e+11
72895  601991.XSHG  2016-07-13  5.430495e+10
48754  601992.XSHG  2016-07-13  4.399242e+10
24499  601998.XSHG  2016-07-13  2.788525e+11
73140  603000.XSHG  2016-07-13  2.080911e+10
73385  603885.XSHG  2016-07-13  3.755758e+10
48999  603993.XSHG  2016-07-13  7.734337e+10

[300 rows x 3 columns]
```

接下来处理分母数据。同样，为保留最新数据，对权益数据按股票代码升序，报表日期和发布日期按降序排列。随后丢弃缺失数据并按照股票代码去掉重复项，更改列名 `TEquityAttrP` 为 `denominator` ：

```
equity = equity.sort(columns=['secID', 'endDate', 'publishDate']
, ascending=[True, False, False])
equity = equity.dropna()
equity = equity.drop_duplicates(cols='secID')
denominator = equity
denominator.rename(columns={"TEquityAttrP": "denominator"}, inpl
ace=True)
```

处理好的分母：

```
print denominator

              secID     endDate publishDate   denominator
0     000001.XSHE  2016-03-31  2016-04-21  1.875690e+11
1     000002.XSHE  2016-03-31  2016-04-28  1.006367e+11
2     000009.XSHE  2016-03-31  2016-04-29  4.467273e+09
203   000027.XSHE  2016-03-31  2016-04-30  2.139399e+10
3     000039.XSHE  2016-03-31  2016-04-29  5.668081e+09
204   000046.XSHE  2016-03-31  2016-04-28  1.632725e+10
4     000060.XSHE  2016-03-31  2016-04-28  7.340159e+09
5     000061.XSHE  2016-03-31  2016-04-30  4.845586e+09
6     000063.XSHE  2016-03-31  2016-04-29  3.902424e+10
7     000069.XSHE  2016-03-31  2016-04-29  3.836175e+10
100   000100.XSHE  2016-03-31  2016-04-21  2.461872e+10
101   000156.XSHE  2016-03-31  2016-04-25  9.396250e+09
8     000157.XSHE  2016-03-31  2016-04-23  3.929794e+10
205   000166.XSHE  2016-03-31  2016-04-30  5.003656e+10
102   000333.XSHE  2016-03-31  2016-04-30  5.385758e+10
9     000338.XSHE  2016-03-31  2016-04-30  3.198088e+10
10    000402.XSHE  2016-03-31  2016-04-30  2.626535e+10
206   000413.XSHE  2016-03-31  2016-04-29  1.466822e+10
207   000415.XSHE  2016-03-31  2016-04-30  2.720240e+10
11    000423.XSHE  2016-03-31  2016-04-20  7.558454e+09
12    000425.XSHE  2016-03-31  2016-04-26  2.057239e+10
208   000503.XSHE  2016-03-31  2016-04-29  1.347180e+09
13    000538.XSHE  2016-03-31  2016-04-27  1.405493e+10
209   000540.XSHE  2016-03-31  2016-04-29  1.271475e+10
210   000559.XSHE  2016-03-31  2016-04-20  4.417438e+09
14    000568.XSHE  2016-03-31  2016-04-28  1.083373e+10
103   000623.XSHE  2016-03-31  2016-04-29  1.743589e+10
15    000625.XSHE  2016-03-31  2016-04-30  3.702576e+10
16    000630.XSHE  2016-03-31  2016-04-30  1.374468e+10
17    000651.XSHE  2016-03-31  2016-04-30  5.069274e+10
..            ...         ...         ...           ...
185   601669.XSHG  2016-03-31  2016-04-30  5.714574e+10
186   601688.XSHG  2016-03-31  2016-04-29  8.060570e+10
295   601718.XSHG  2016-03-31  2016-04-19  1.303635e+10
296   601727.XSHG  2016-03-31  2016-04-30  3.758142e+10
187   601766.XSHG  2016-03-31  2016-04-28  9.868600e+10
297   601788.XSHG  2016-03-31  2016-04-26  3.993967e+10
188   601800.XSHG  2016-03-31  2016-04-27  1.479940e+11
96    601808.XSHG  2016-03-31  2016-04-29  4.578060e+10
189   601818.XSHG  2016-03-31  2016-04-30  2.323020e+11
97    601857.XSHG  2016-03-31  2016-04-29  1.169389e+12
190   601866.XSHG  2016-03-31  2016-04-29  1.801931e+10
298   601872.XSHG  2016-03-31  2016-04-22  1.404631e+10
191   601888.XSHG  2016-03-31  2016-04-23  1.188385e+10
192   601898.XSHG  2016-03-31  2016-04-28  8.306575e+10
193   601899.XSHG  2016-03-31  2016-04-30  2.722604e+10
194   601901.XSHG  2016-03-31  2016-04-29  3.514534e+10
300   601919.XSHG  2016-03-31  2016-04-29  2.253836e+10
195   601928.XSHG  2016-03-31  2016-04-30  1.093871e+10
```

```
196  601933.XSHG  2016-03-31  2016-05-06  1.269974e+10
198  601939.XSHG  2016-03-31  2016-04-30  1.499405e+12
199  601958.XSHG  2016-03-31  2016-04-29  1.271524e+10
301  601985.XSHG  2016-03-31  2016-04-29  3.860421e+10
98   601988.XSHG  2016-03-31  2016-04-27  1.348157e+12
200  601989.XSHG  2016-03-31  2016-04-29  5.696661e+10
302  601991.XSHG  2016-03-31  2016-04-28  4.568534e+10
201  601992.XSHG  2016-03-31  2016-04-27  3.824465e+10
99   601998.XSHG  2016-03-31  2016-04-28  3.286200e+11
303  603000.XSHG  2016-03-31  2016-04-29  2.734087e+09
304  603885.XSHG  2016-03-31  2016-04-21  3.820480e+09
202  603993.XSHG  2016-03-31  2016-04-29  1.757461e+10

[300 rows x 4 columns]
```

分子分母处理好之后，我们将两个 `DataFrame` 使用 `merge` 函数合并，使用参数 `how='inner'` 保留在两者中均存在的股票。

```
dat_info = numerator.merge(denominator, on='secID', how='inner')
```

作为比值，分母不可以为零，这里我们通过设置分母绝对值大于一个很小的数来过滤不符合要求的数据。随后直接通过 `DataFrame['Column_name']` 的复制添加一列PB：

```
dat_info = dat_info[abs(dat_info['denominator']) >= 1e-8]
dat_info['PB'] = dat_info['numerator'] / dat_info['denominator']
```

```
pb_signal = dat_info[['secID', 'PB']]
pb_signal = pb_signal.set_index('secID')['PB']
print pb_signal

secID
000001.XSHE     0.822961
000002.XSHE     2.009577
000009.XSHE     5.053661
000027.XSHE     1.287895
000039.XSHE     7.824125
000046.XSHE     3.198446
000060.XSHE     3.731845
000061.XSHE     4.461653
000063.XSHE     1.584789
000069.XSHE     1.497319
000100.XSHE     1.761203
000156.XSHE     3.015817
000157.XSHE     0.836663
000166.XSHE     2.581403
000333.XSHE     3.250605
```

```
000338.XSHE     1.021508
000402.XSHE     1.142527
000413.XSHE     2.611881
000415.XSHE     1.550541
000423.XSHE     4.895781
000425.XSHE     1.093446
000503.XSHE    29.009320
000538.XSHE     5.260001
000540.XSHE     2.475100
000559.XSHE     8.226875
000568.XSHE     4.219546
000623.XSHE     1.318376
000625.XSHE     1.926825
000630.XSHE     2.003295
000651.XSHE     2.102841
                  ...
601669.XSHG     1.453791
601688.XSHG     1.885648
601718.XSHG     2.446804
601727.XSHG     2.753812
601766.XSHG     2.607594
601788.XSHG     1.776320
601800.XSHG     1.208783
601808.XSHG     1.328899
601818.XSHG     0.753530
601857.XSHG     1.159739
601866.XSHG     2.762044
601872.XSHG     1.973199
601888.XSHG     3.752535
601898.XSHG     0.917795
601899.XSHG     3.109705
601901.XSHG     1.843392
601919.XSHG     2.452265
601928.XSHG     2.598709
601933.XSHG     2.869754
601939.XSHG     0.843705
601958.XSHG     2.291442
601985.XSHG     2.830503
601988.XSHG     0.729333
601989.XSHG     2.330397
601991.XSHG     1.188674
601992.XSHG     1.150289
601998.XSHG     0.848556
603000.XSHG     7.610988
603885.XSHG     9.830593
603993.XSHG     4.400859
Name: PB, dtype: float64
```

好了接下来我们把以上PB因子计算过程变成一个函数，使得它可以计算回测开始时间到结束时间的PB值，这样我们可以在通联的多因子信号分析工具RDP中方便的测试

```
def str2date(date_str):
    date_obj = dt.datetime(int(date_str[0:4]), int(date_str[4:6]), int(date_str[6:8]))
    return Date.fromDateTime(date_obj)

def signal_pb_calc(universe, current_date):
    today = str2date(current_date)
    start_date = (today - Period('1Y')).toDateTime().strftime('%Y%m%d')
    end_date = today.toDateTime().strftime('%Y%m%d')
    # dealing with the numerator
    market_capital = DataAPI.MktEqudGet(secID=universe, field=['secID', 'tradeDate', 'marketValue', 'negMarketValue', 'turnoverVol'], beginDate=start_date, endDate=end_date, pandas='1')
    market_capital = market_capital[market_capital['turnoverVol'] > 0]
    market_capital = market_capital.sort(columns=['secID', 'tradeDate'], ascending=[True, True])
    market_capital = market_capital.drop_duplicates(subset='secID', take_last=True)
    market_capital['marketValue'][isnull(market_capital['marketValue'])] = market_capital['negMarketValue'][isnull(market_capital['marketValue'])]
    market_capital = market_capital.drop('negMarketValue', axis=1)
    numerator = market_capital.dropna()
    numerator.rename(columns={'marketValue': 'numerator'}, inplace=True)
    # dealing with the denominator
    equity = DataFrame()
    for rpt_type in ['Q1', 'S1', 'Q3', 'A']:
        try:
            tmp = DataAPI.FdmtBSGet(secID=universe, field=['secID', 'endDate', 'publishDate', 'TEquityAttrP'], beginDate=start_date, publishDateEnd=end_date,  reportType=rpt_type)
        except:
            tmp = DataFrame()
        equity = pd.concat([equity, tmp], axis=0)

    equity = equity.sort(columns=['secID', 'endDate', 'publishDate'], ascending=[True, False, False])
    equity = equity.dropna()
    equity = equity.drop_duplicates(cols='secID')
    denominator = equity
    denominator.rename(columns={"TEquityAttrP": "denominator"}, inplace=True)
    # merge two dataframe and calculate price-to- book ratio
    dat_info = numerator.merge(denominator, on='secID', how='inner')
    dat_info = dat_info[abs(dat_info['denominator']) >= 1e-8]
    dat_info['PB'] = dat_info['numerator'] / dat_info['denominator']
    pb_signal = dat_info[['secID', 'PB']]
    pb_signal["secID"] = pb_signal["secID"].apply(lambda x:x[:6]
)
    return pb_signal
```

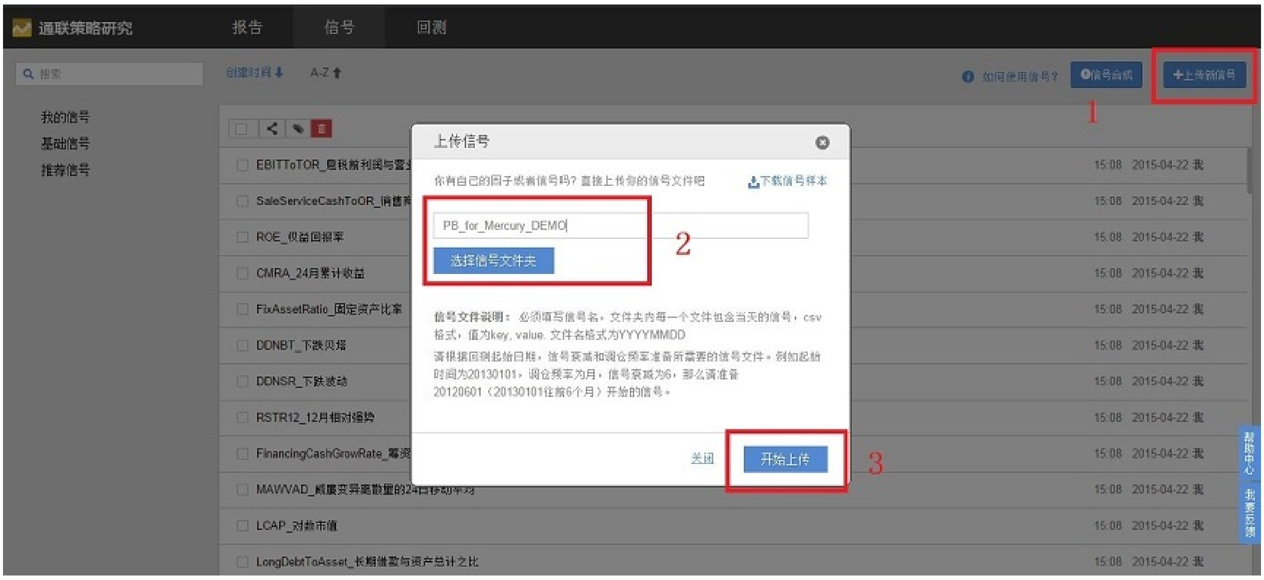
此代码完成的功能是：

- 计算沪深300成分股在一段时间内的PB值作为信号
- 把这些PB数据按照天存储为csv文件
- 把csv文件打包成zip

可以将这些文件下载到本地，解压到一个文件夹（比如 `PB_for_Mercury_DEMO` ），然后上传到RDP（通联策略研究）中当做信号使用。

```
start = datetime(2015, 1, 1)
end = datetime(2015, 4, 23)

univ = set_universe('HS300')
cal = Calendar('China.SSE')

all_files = []
today = start
while((today - end).days < 0):
    today_CAL = Date.fromDateTime(today)
    if(cal.isBizDay(today_CAL)):
        today_str = today.strftime("%Y%m%d")
        print "Calculating PB values on " + today_str
        pb_value = signal_pb_calc(univ, today_str)
        file_name = today_str + '.csv'
        pb_value.to_csv(file_name, index=False, header=False)
        all_files.append(file_name)
    today = today + timedelta(days=1)

# exporting all *.csv files to PB.zip
zip_files("PB"+ "_" + start.strftime("%Y%m%d") + "_" + end.strft
ime("%Y%m%d"), all_files)

# delete all *.csv
delete_files(all_files)
```

第一步：解压点击‘上传新信号’，选择信号文件夹并为信号命名，然后，开始上传；



上传信号

你有自己的因子或者信号吗？直接上传你的信号文件吧　下载信号样本

PB_for_Mercury_DEMO

选择信号文件夹

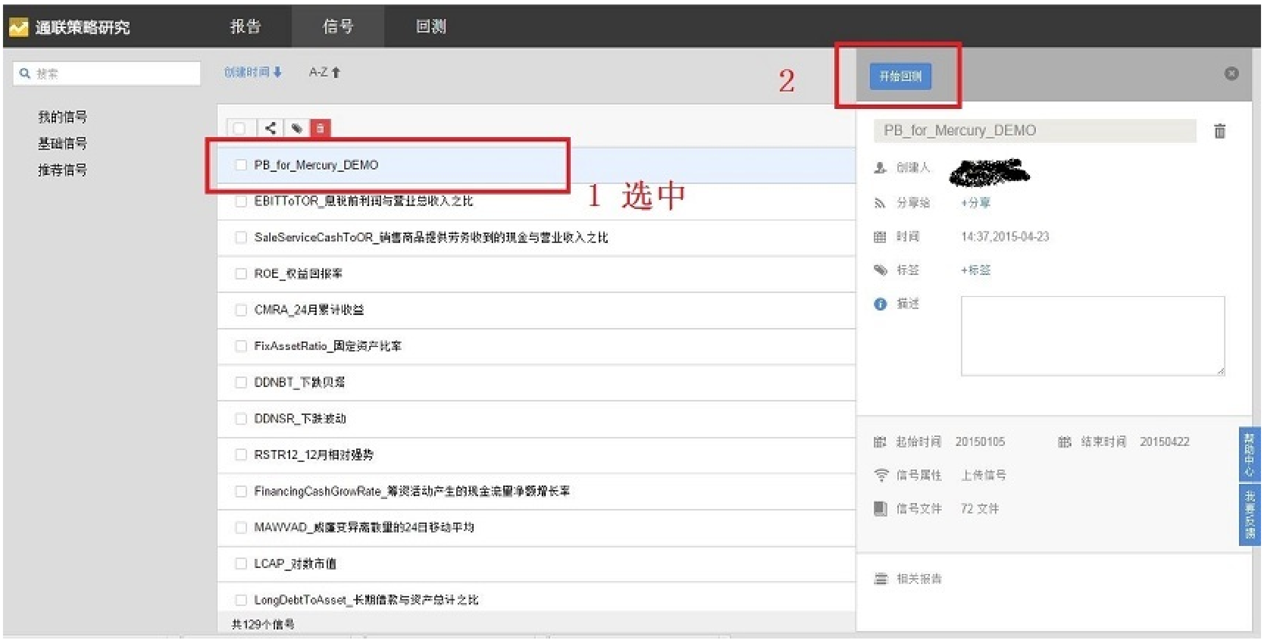
信号文件说明：必须填写信号名，文件夹内每一个文件包含当天的信号，csv格式，值为key, value. 文件名格式为YYYYMMDD

请根据回测起始日期，信号衰减和调仓频率准备所需要的信号文件。例如起始时间为20130101，调仓频率为月，信号衰减为6，那么请准备20120601（20130101往前6个月）开始的信号。

关闭　开始上传

- EBITToTOR_息税前利润与营业… 15:08 2015-04-22
- SaleServiceCashToOR_销售商… 15:08 2015-04-22
- ROE_权益回报率 15:08 2015-04-22
- CMRA_24月累计收益 15:08 2015-04-22
- FixAssetRatio_固定资产比率 15:08 2015-04-22
- DDNBT_下跌贝塔 15:08 2015-04-22
- DDNSR_下跌波动 15:08 2015-04-22
- RSTR12_12月相对强势 15:08 2015-04-22
- FinancingCashGrowRate_筹资… 15:08 2015-04-22
- MAWVAD_成交量变异离散量的24日移动平均 15:08 2015-04-22
- LCAP_对数市值 15:08 2015-04-22
- LongDebtToAsset_长期借款与资产总计之比 15:08 2015-04-22

第二步：选中上传的新信号，点击‘开始回测’；



- PB_for_Mercury_DEMO
- EBITToTOR_息税前利润与营业总收入之比
- SaleServiceCashToOR_销售商品提供劳务收到的现金与营业收入之比
- ROE_权益回报率
- CMRA_24月累计收益
- FixAssetRatio_固定资产比率
- DDNBT_下跌贝塔
- DDNSR_下跌波动
- RSTR12_12月相对强势
- FinancingCashGrowRate_筹资活动产生的现金流量净额增长率
- MAWVAD_成交量变异离散量的24日移动平均
- LCAP_对数市值
- LongDebtToAsset_长期借款与资产总计之比

共129个信号

PB_for_Mercury_DEMO

创建人 [redacted]

分享给 +分享

时间 14:37,2015-04-23

标签 +标签

描述

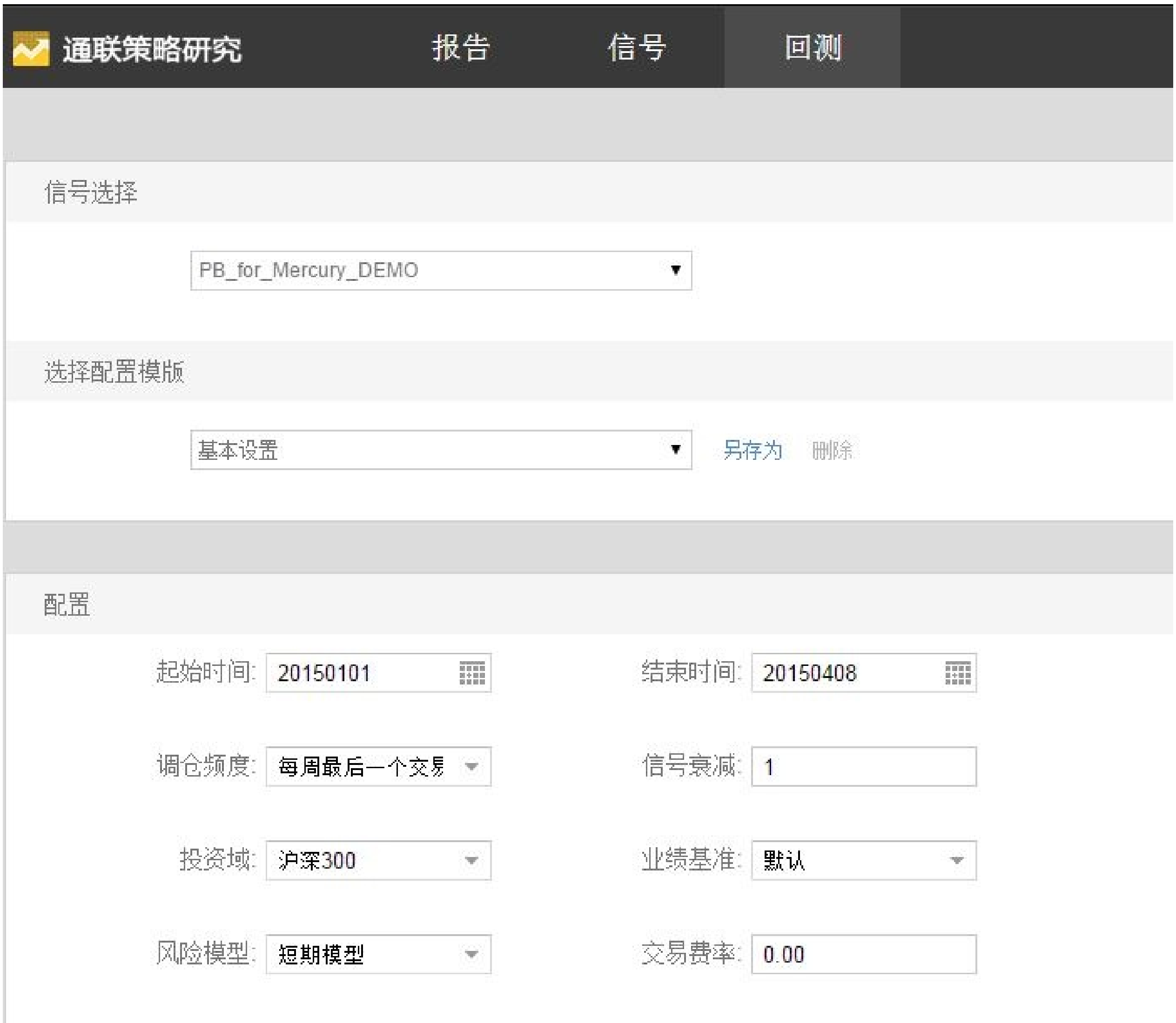
起始时间 20150105　结束时间 20150422

信号属性 上传信号

信号文件 72 文件

相关报告

第三步：进行回测的各种配置；



信号选择

PB_for_Mercury_DEMO

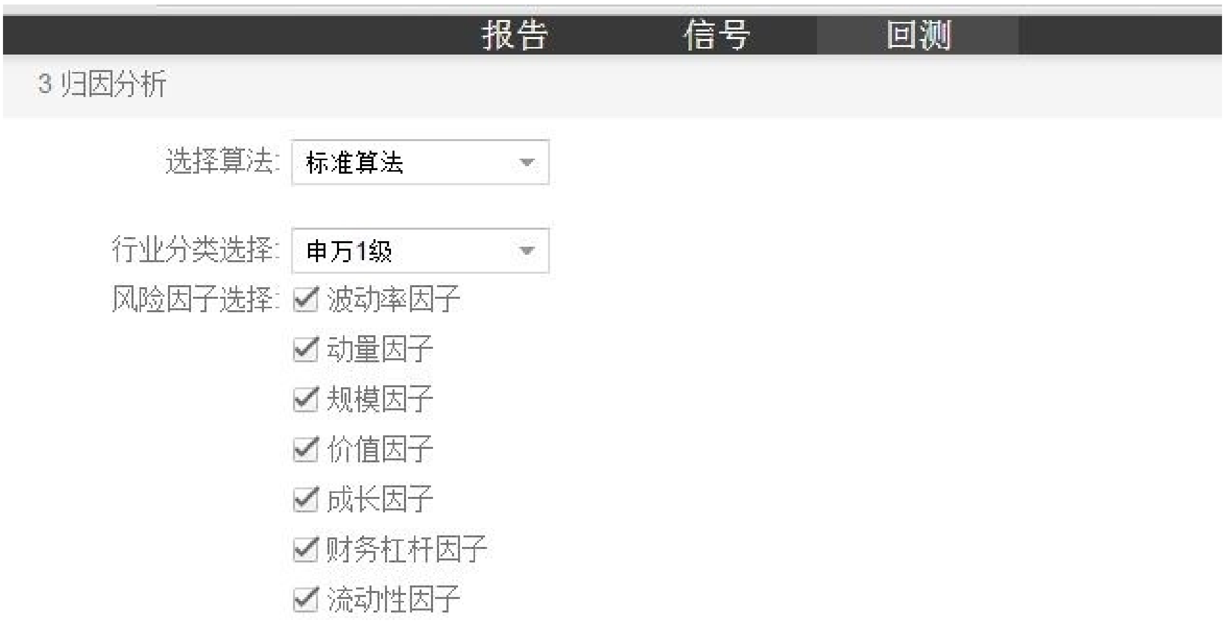
选择配置模版

基本设置 另存为 删除

配置

起始时间:	20150101	结束时间:	20150408
调仓频度:	每周最后一个交易	信号衰减:	1
投资域:	沪深300	业绩基准:	默认
风险模型:	短期模型	交易费率:	0.00

第四步：开始回测，回测完成后，点击报告的链接，查看回测结果；

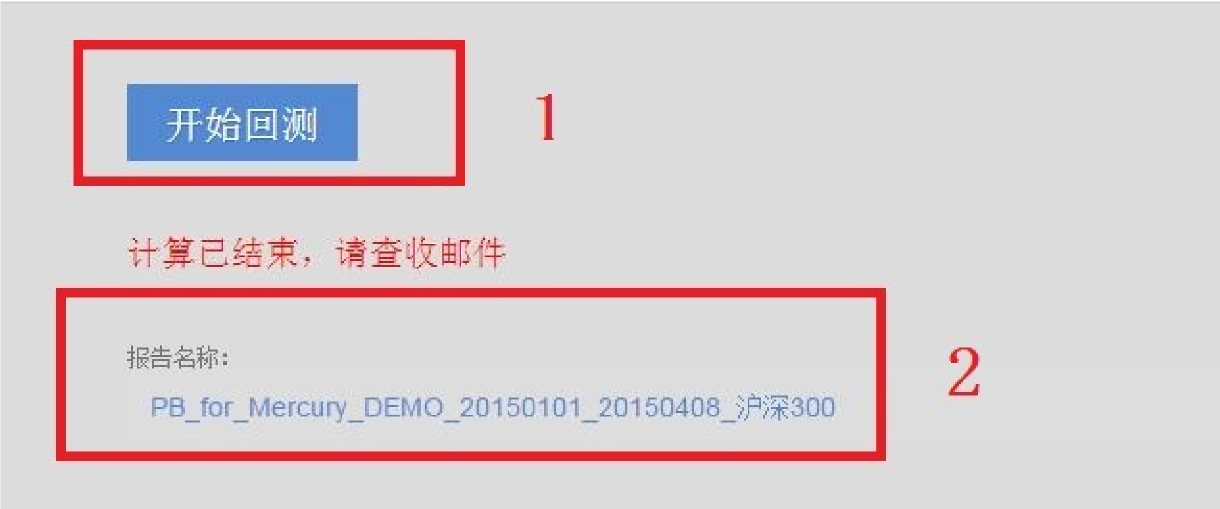


3 归因分析

选择算法: 标准算法

行业分类选择: 申万1级

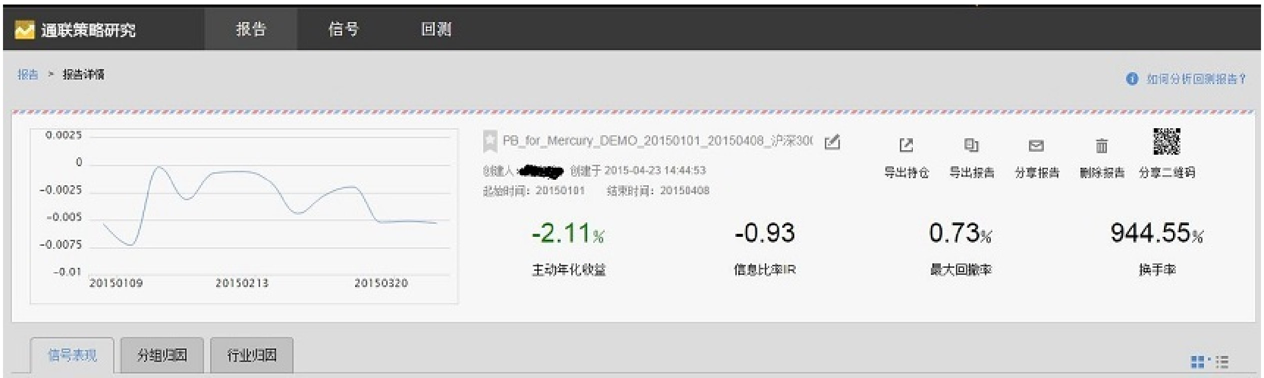
风险因子选择:
☑ 波动率因子
☑ 动量因子
☑ 规模因子
☑ 价值因子
☑ 成长因子
☑ 财务杠杆因子
☑ 流动性因子



计算已结束，请查收邮件

报告名称:
PB_for_Mercury_DEMO_20150101_20150408_沪深300

第五步：查看回测结果。

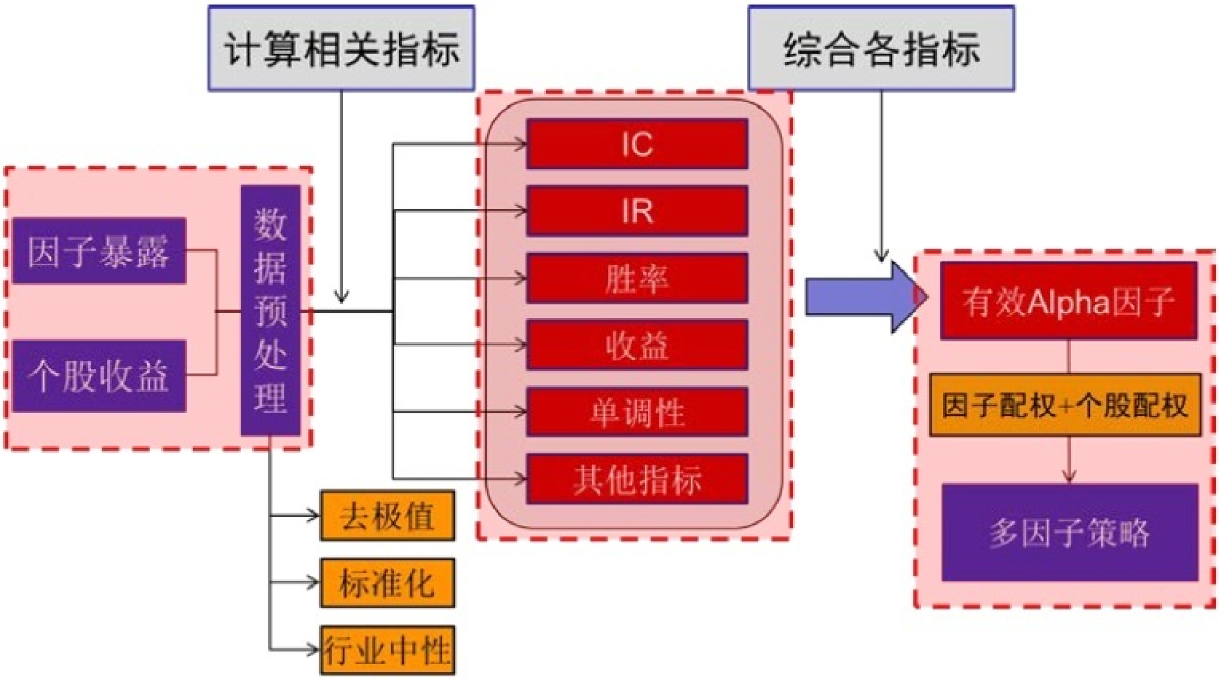


PB_for_Mercury_DEMO_20150101_20150408_沪深30(

创建人 [redacted] 创建于 2015-04-23 14:44:53
起始时间：20150101 结束时间：20150408

主动年化收益	信息比率IR	最大回撤率	换手率
-2.11%	-0.93	0.73%	944.55%

以上研究过程演示了单个因子的产生和快速回测的过程，后面会介绍多因子的策略框架


计算相关指标
综合各指标
因子暴露
个股收益
数据预处理
去极值
标准化
行业中性
IC
IR
胜率
收益
单调性
其他指标
有效Alpha因子
因子配权+个股配权
多因子策略

量化分析师的Python日记【第13天 Q Quant兵器谱之偏微分方程3】

> 来源：https://uqer.io/community/share/555dc9e8f9f06c6c7404f96e
>
> 欢迎来到 Black - Scholes — Merton 的世界！本篇中我们将使用在第11天学习到的知识应用到这个金融学的具体方程之上！

```
import numpy as np
import math
import seaborn as sns
from matplotlib import pylab
font.set_size(15)
```

1. 问题的提出

BSM模型可以设置为如下的偏微分方差初值问题：

$$\begin{cases} \frac{\partial C(S,t)}{\partial t} + rS\frac{\partial C(S,t)}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 C(S,t)}{\partial S^2} - rC(S,t) = 0, \quad 0 \le t \le T \\ C(S,T) = \max(S-K,0) \end{cases}$$

做变量替换 `τ=T-t` ，并且设置上下边界条件:

$$\begin{cases} \frac{\partial C(x,\tau)}{\partial \tau} & = (r - \frac{1}{2}\sigma^2)\frac{\partial C(x,\tau)}{\partial x} + \frac{1}{2}\sigma^2 \frac{\partial^2 C(x,\tau)}{\partial x^2} - rC(x,\tau) = 0, \quad 0 \le \tau \le T \\ C(x,0) & = \max(e^x - K, 0), \\ C(x_{max},\tau) & = e^{x_{max}} - Ke^{-\tau}, \\ C(x_{min},\tau) & = 0 \end{cases}$$

2. 算法

按照之前介绍的隐式差分格式的方法，用离散差分格式代替连续微分：

$$\frac{C_{j,k+1} - C_{j,k}}{\Delta\tau} = (r - \frac{1}{2}\sigma^2)\frac{C_{j+1,k+1} - C_{j-1,k+1}}{2\Delta x} + \frac{1}{2}\sigma^2\frac{C_{j+1,k+1} - 2C_{j,k+1} + C_{j-1,k+1}}{\Delta x^2} - rC_{j,k+1},$$

$$\Rightarrow \quad C_{j,k+1} - C_{j,k} - (r - \frac{1}{2}\sigma^2)\frac{\Delta\tau}{2\Delta x}(C_{j+1,k+1} - C_{j-1,k+1}) - \frac{1}{2}\sigma^2\frac{\Delta\tau}{\Delta x^2}(C_{j+1,k+1} - 2C_{j,k+1} + C_{j-1,k+1}) + r\Delta\tau C_{j,k+1} = 0,$$

$$\Rightarrow \quad -(\frac{1}{2}(r - \frac{1}{2}\sigma^2)\frac{\Delta\tau}{\Delta x} + \frac{1}{2}\sigma^2\frac{\Delta\tau}{\Delta x^2})C_{j+1,k+1} + (1 + \sigma^2\frac{\Delta\tau}{\Delta x^2} + r\Delta\tau)C_{j,k+1} - (\frac{1}{2}\sigma^2\frac{\Delta\tau}{\Delta x^2} - \frac{1}{2}(r - \frac{1}{2}\sigma^2)\frac{\Delta\tau}{\Delta x})C_{j-1,k+1} = C_{j,k},$$

$$\Rightarrow \quad l_jC_{j-1,k+1} + c_jC_{j,k+1} + u_jC_{j+1,k+1} = C_{j,k}$$

其中

$$l_j = -(\frac{1}{2}\sigma^2\frac{\Delta\tau}{\Delta x^2} - \frac{1}{2}(r - \frac{1}{2}\sigma^2)\frac{\Delta\tau}{\Delta x}),$$

$$c_j = 1 + \sigma^2\frac{\Delta\tau}{\Delta x^2} + r\Delta\tau,$$

$$u_j = -(\frac{1}{2}(r - \frac{1}{2}\sigma^2)\frac{\Delta\tau}{\Delta x} + \frac{1}{2}\sigma^2\frac{\Delta\tau}{\Delta x^2})$$

以上即为差分方程组。

这里有些细节需要处理，就是左右边界条件，我们这里使用Dirichlet边界条件，则：

$$C_{0,k} = C(x_{min}, \tau),$$

$$C_{N,k} = C(x_{max}, \tau)$$

3.实现

```
import scipy as sp
from scipy.linalg import solve_banded
```

描述BSM方程结构的类：`BSMModel`

```
class BSMModel:

    def __init__(self, s0, r, sigma):
        self.s0 = s0
        self.x0 = math.log(s0)
        self.r = r
        self.sigma = sigma

    def log_expectation(self, T):
        return self.x0 + (self.r - 0.5 * self.sigma * self.sigma
) * T

    def expectation(self, T):
        return math.exp(self.log_expectation(T))

    def x_max(self, T):
        return self.log_expectation(T) + 4.0 * self.sigma * math
.sqrt(T)

    def x_min(self, T):
        return self.log_expectation(T) - 4.0 * self.sigma * math
.sqrt(T)
```

描述我们这里设计到的产品的类： `CallOption`

```
class CallOption:

    def __init__(self, strike):
        self.k = strike

    def ic(self, spot):
        return max(spot - self.k, 0.0)

    def bcl(self, spot, tau, model):
        return 0.0

    def bcr(self, spot, tau, model):
        return spot - math.exp(-model.r*tau) * self.k
```

完整的隐式格式： `BSMScheme`

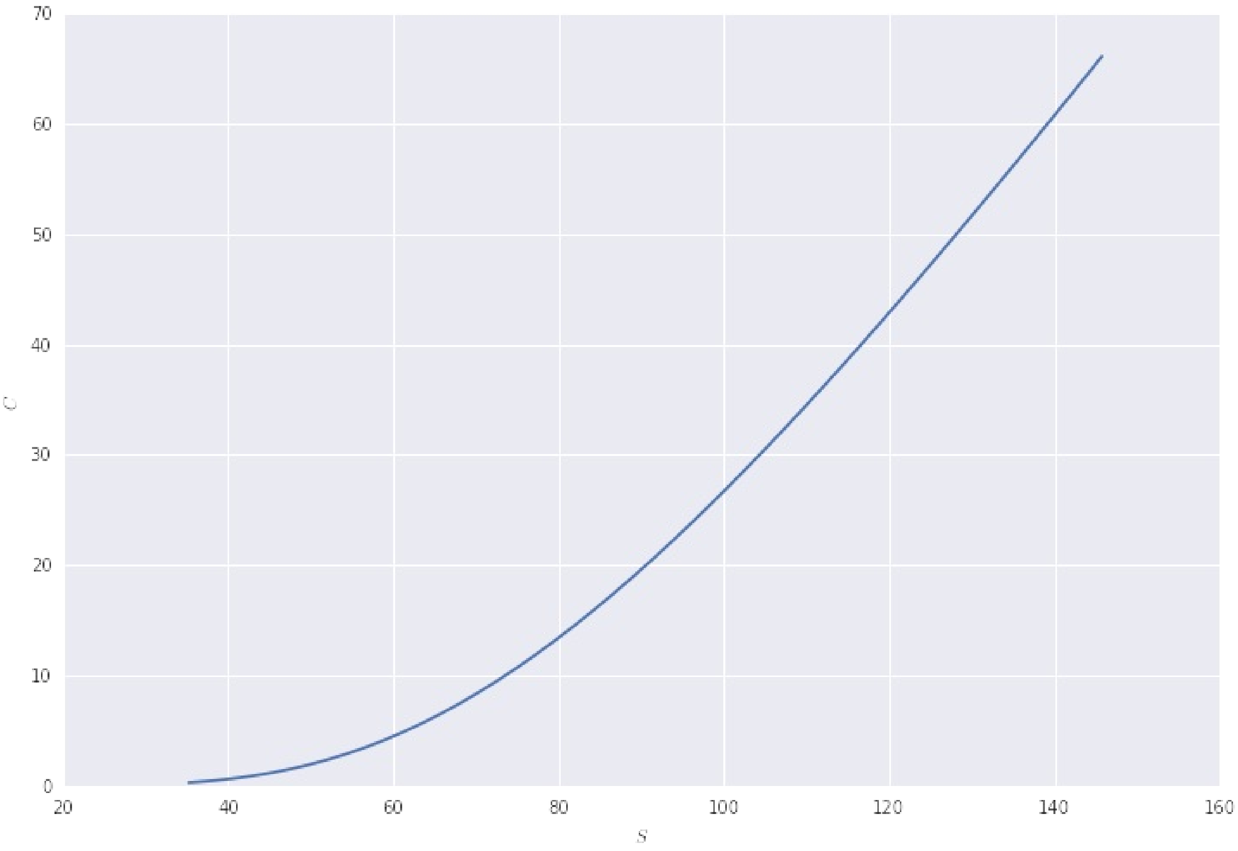
```
class BSMScheme:
    def __init__(self, model, payoff, T, M, N):
        self.model = model
        self.T = T
        self.M = M
        self.N = N
        self.dt = self.T / self.M
        self.payoff = payoff
        self.x_min = model.x_min(self.T)
        self.x_max = model.x_max(self.T)
        self.dx = (self.x_max - self.x_min) / self.N
        self.C = np.zeros((self.N+1, self.M+1)) # 全部网格
        self.xArray = np.linspace(self.x_min, self.x_max, self.N+1)
        self.C[:,0] = map(self.payoff.ic, np.exp(self.xArray))

        sigma_square = self.model.sigma*self.model.sigma
        r = self.model.r

        self.l_j = -(0.5*sigma_square*self.dt/self.dx/self.dx - 0.5 * (r - 0.5 * sigma_square)*self.dt/self.dx)
        self.c_j = 1.0 + sigma_square*self.dt/self.dx/self.dx + r*self.dt
        self.u_j = -(0.5*sigma_square*self.dt/self.dx/self.dx + 0.5 * (r - 0.5 * sigma_square)*self.dt/self.dx)

    def roll_back(self):

        for k in range(0, self.M):
            udiag = np.ones(self.N-1) * self.u_j
            ldiag =  np.ones(self.N-1) * self.l_j
            cdiag =  np.ones(self.N-1) * self.c_j

            mat = np.zeros((3,self.N-1))
            mat[0,:] = udiag
            mat[1,:] = cdiag
            mat[2,:] = ldiag
            rhs = np.copy(self.C[1:self.N,k])

            # 应用左端边值条件
            v1 = self.payoff.bcl(math.exp(self.x_min), (k+1)*self.dt, self.model)
            rhs[0] -= self.l_j * v1

            # 应用右端边值条件
            v2 = self.payoff.bcr(math.exp(self.x_max), (k+1)*self.dt, self.model)
            rhs[-1] -= self.u_j * v2

            x = solve_banded((1,1), mat, rhs)
            self.C[1:self.N, k+1] = x
            self.C[0][k+1] = v1
            self.C[self.N][k+1] = v2

    def mesh_grids(self):
        tArray = np.linspace(0, self.T, self.M+1)
        tGrids, xGrids = np.meshgrid(tArray, self.xArray)
        return tGrids, xGrids
```

应用在一起：

```
model = BSMModel(100.0, 0.05, 0.2)
payoff = CallOption(105.0)
scheme = BSMScheme(model, payoff, 5.0, 100, 300)
```

```
scheme.roll_back()
```

```
from matplotlib import pylab
pylab.figure(figsize=(12,8))
pylab.plot(np.exp(scheme.xArray)[50:170], scheme.C[50:170,-1])
pylab.xlabel('$S$')
pylab.ylabel('$C$')

<matplotlib.text.Text at 0x76ea7d0>
```



4. 收敛性测试

首先使用BSM模型的解析解获得精确解：

```
analyticPrice = BSMPrice(1, 105., 100., 0.05, 0.0, 0.2, 5.)
analyticPrice
```

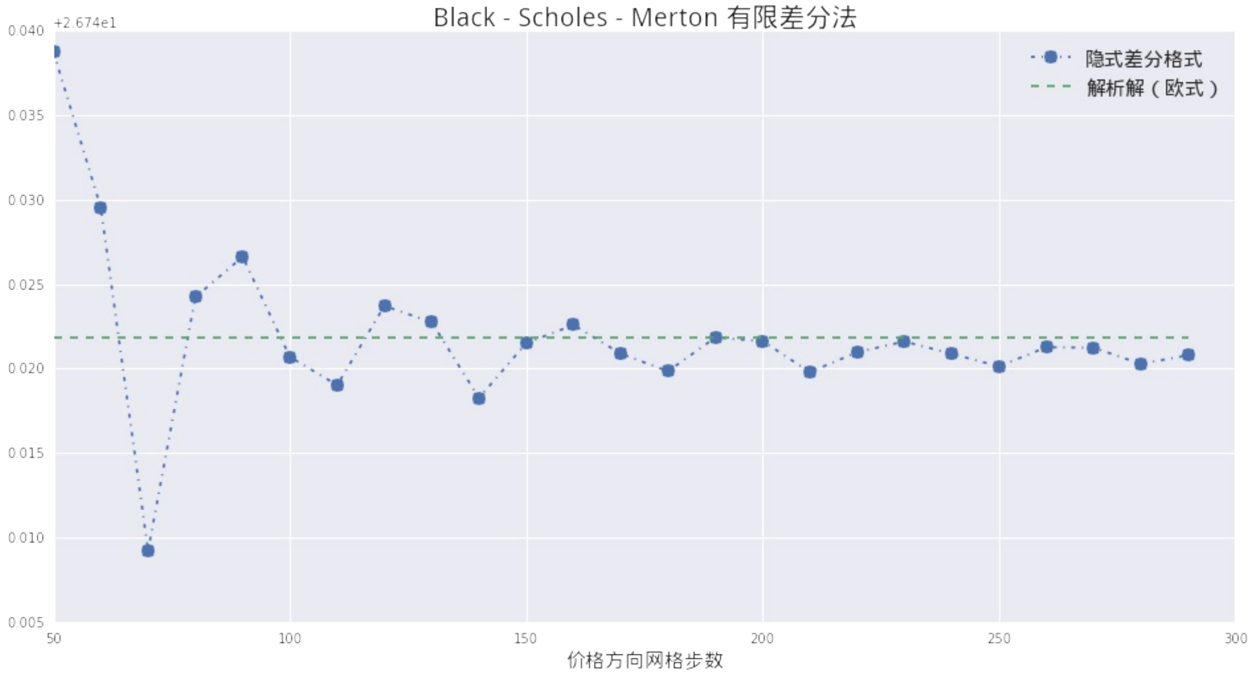
price	delta	gamma	vega	rho	theta	
1	26.761844	0.749694	0.00711	71.10319	241.037549	-3.83

我们固定时间方向网格数为3000，分别计算不同S网格数情形下的结果：

```
xSteps = range(50,300,10)
finiteResult = []
for xStep in xSteps:
    model = BSMModel(100.0, 0.05, 0.2)
    payoff = CallOption(105.0)
    scheme = BSMScheme(model, payoff, 5.0, 3000, xStep)
    scheme.roll_back()

    interp = CubicNaturalSpline(np.exp(scheme.xArray), scheme.C[
:,-1])
    price = interp(100.0)
    finiteResult.append(price)
```

我们可以画下收敛图：

```
anyRes = [analyticPrice['price'][1]] * len(xSteps)

pylab.figure(figsize = (16,8))
pylab.plot(xSteps, finiteResult, '-.', marker = 'o', markersize
= 10)
pylab.plot(xSteps, anyRes, '--')
pylab.legend([u'隐式差分格式', u'解析解（欧式）'], prop = font)
pylab.xlabel(u'价格方向网格步数', fontproperties = font)
pylab.title(u'Black - Scholes - Merton 有限差分法', fontproperties
 = font, fontsize = 20)

<matplotlib.text.Text at 0x7857bd0>
```


Black - Scholes - Merton 有限差分法
+2.674e1
0.040
0.035
0.030
0.025
0.020
0.015
0.010
0.005
50
100
150
200
250
300
价格方向网格步数
隐式差分格式
解析解（欧式）

量化分析师的Python日记【第14天：如何在优矿上做Alpha对冲模型】

> 来源：https://uqer.io/community/share/55e662f9f9f06c1ea481f9cf

上篇第12天讲了单因子如何产生和回测，本篇主要用具体的实例来介绍如何在优矿上做Alpha对冲模型，分以下四个部分展开：

- Alpha对冲模型简介
- 优矿“三剑客”简介
- 如何在优矿上做Alpha冲对模型 (多信号合成)

关于大赛

1、Alpha对冲模型简介

A、假设市场完全有效，那么根据CAPM模型有， `Rs=Rf+βs*(Rm-Rf)` 。式中， `Rs` 表示股票收益， `Rf` 表示无风险收益率， `Rm` 表示市场收益， `βs` 表示股票相比于市场的波动程度，用以衡量股票的系统性风险。

B、遗憾的是，市场并非完全有效，个股仍存在alpha（超额收益）。根据Jensen's alpha的定义： `αs=Rs-[Rf+βs*(Rm-Rf)]` ，除掉被市场解释的部分，超越市场基准的收益即为个股alpha。

C、实际中，股票的收益是受多方面因素影响的，比如经典的Fama French三因素就告诉我们，市值大小、估值水平、以及市场因子就能解释股票收益，而且低市值、低估值能够获取超额收益。那么，我们就可以通过寻找能够获取alpha的驱动因子来构建组合。

D、假设我们已经知道了哪些因子能够获取超额收益，那么我们根据这些因子构建股票组合（比如持有低市值、低估值的股票）。那么组合的收益理论上是能够获取超额收益的，简单来讲就是，组合的累计收益图应该是在基准（比如沪深300）累计收益图之上的，而且两者的差应该是扩大的趋势。

E、由于组合的涨跌我们是不知道的，我们能够确保的是组合与基准的收益差在不断扩大，那么持有组合，做空基准，对冲获取稳定的差额收益（alpha收益）,这就是传说中的市场中性策略

2、优矿“三剑客”

针对上述研究流程，优矿提供全程服务，从金融大数据，模型的研究开发到实盘交易和组合管理：

- DataAPI：提供近300个高质量的因子数据（基本面因子,技术面因子和大数据因子），为模型提供充足的原材料和让用户自己研究因子提供了基础
- RDP：提供标准的因子到信号的处理函数（去极值、中性化、标准化）同时，还提供了功能强大的组合构建函数
- Quartz：提供标准的、更贴近实际的回测框架，一键查看对冲模型历史表现

3、实例：优矿上的对冲模型

回测框架&基础工作简介：

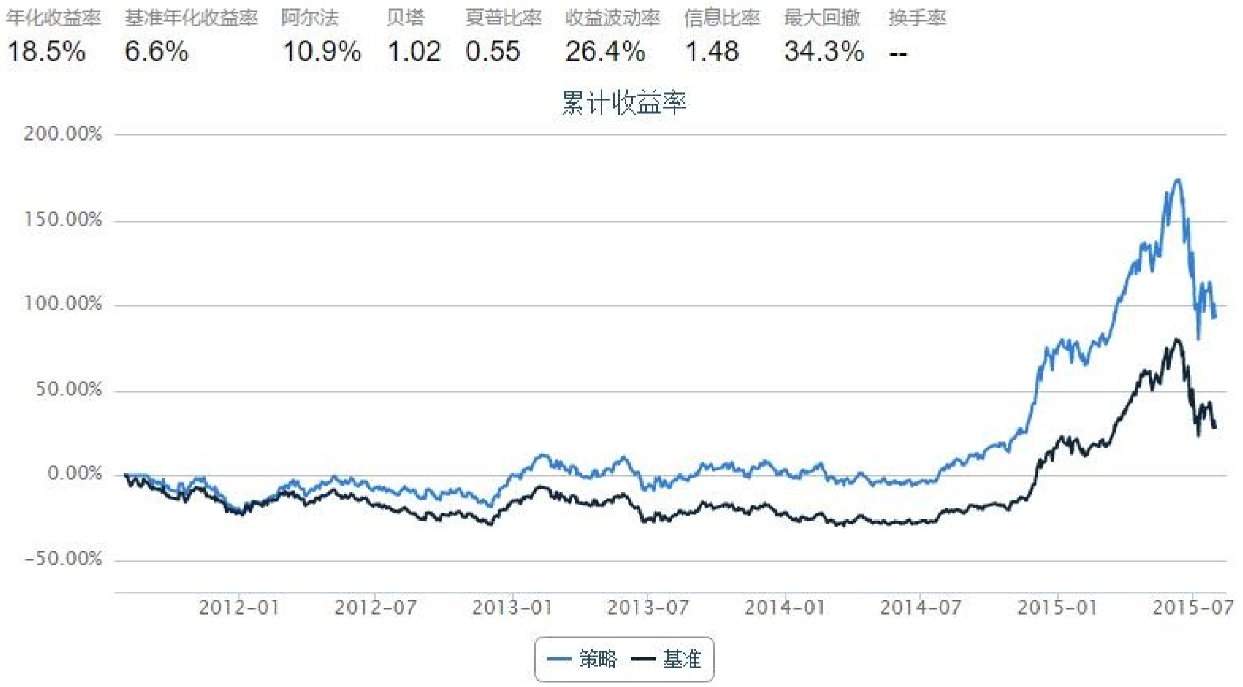
- 回测区间从2011年8月1日~2015年8月1日，基准为沪深300，策略每月第一个交易日开盘之后建仓
- 因子选取：净利润增长率（NetProfitGrowRate）、权益收益率（ROE）、相对强弱指标（RSI）
- 因子到信号的处理：用到了去极值（winsorize）、中性化（neutralize）、标准化（standardize）处理
- 组合构建：用到了RDP里的simple_long_only()

PS：关于函数的详细使用说明，可以新建 `cell` 输入 `函数名+?` ，运行得到API使用文档。比如，运行下面的代码便可以得到 `simple_long_only` 的使用说明。

```
simple_long_only?
```

```
end = '2015-08-01'                                  # 回测结束时间
benchmark = 'HS300'                                 # 策略参考标准
universe = set_universe('HS300')  # 证券池，支持股票和基金
capital_base = 10000000                               # 起始资金
freq = 'd'                                          # 策略类型，'d'表示日间
策略使用日线回测，'m'表示日内策略使用分钟线回测
refresh_rate = 1                                    # 调仓频率，表示执行hand
le_data的时间间隔，若freq = 'd'时间间隔的单位为交易日，若freq = 'm'时间
间隔为分钟

# 构建日期列表
data=DataAPI.TradeCalGet(exchangeCD=u"XSHG",beginDate=u"20110801"
,endDate=u"20150801",field=['calendarDate','isMonthEnd'],pandas=
"1")
data = data[data['isMonthEnd'] == 1]
date_list = data['calendarDate'].values.tolist()

cal = Calendar('China.SSE')
period = Period('-1B')

def initialize(account):                            # 初始化虚拟账户状态
```

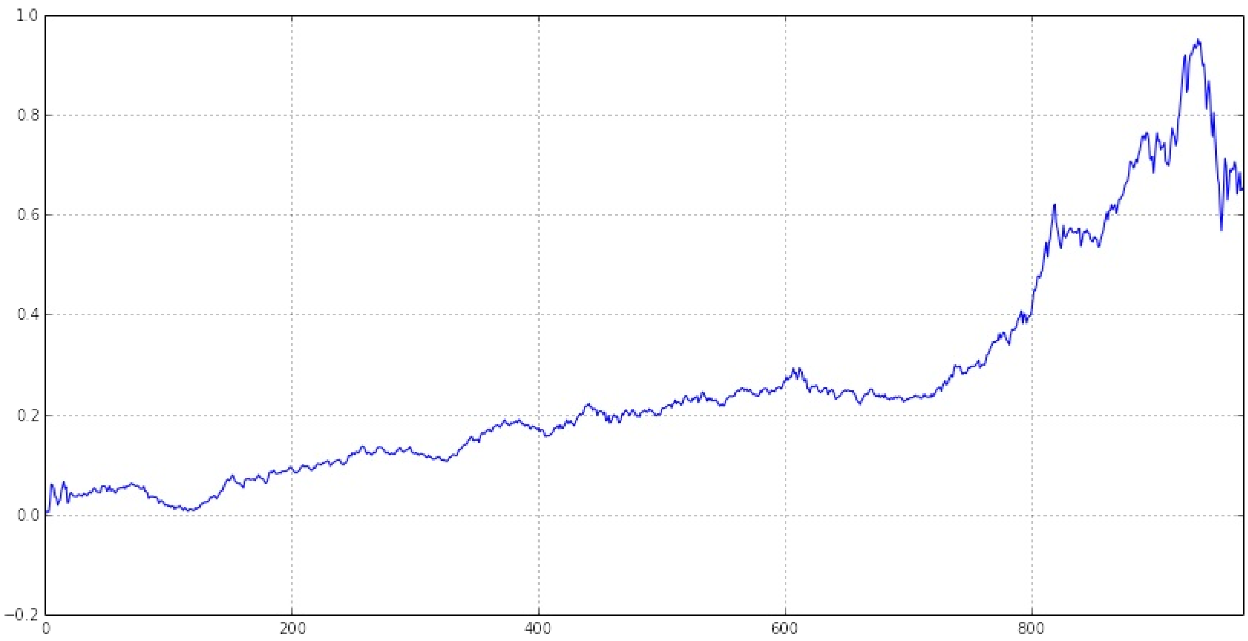
```
    pass

def handle_data(account):                  # 每个交易日的买入卖出指令


    today = account.current_date
    today = Date.fromDateTime(account.current_date)  # 向前移动一
个工作日
    yesterday = cal.advanceDate(today, period)
    yesterday = yesterday.toDateTime()

    if yesterday.strftime('%Y-%m-%d') in date_list:

        # 净利润增长率
        NetProfitGrowRate =DataAPI.MktStockFactorsOneDayGet(trad
eDate=yesterday.strftime('%Y%m%d'),secID=account.universe,field=
u"secID,NetProfitGrowRate",pandas="1")
        NetProfitGrowRate.columns = ['secID','NetProfitGrowRate'
]
        NetProfitGrowRate['ticker'] = NetProfitGrowRate['secID']
.apply(lambda x: x[0:6])
        NetProfitGrowRate.set_index('ticker',inplace=True)
        ep = NetProfitGrowRate['NetProfitGrowRate'].dropna().to_
dict()
        signal_NetProfitGrowRate = standardize(neutralize(winsor
ize(ep),yesterday.strftime('%Y%m%d')))  # 对因子进行去极值、中性化、
标准化处理得信号

        # 权益收益率
        ROE = DataAPI.MktStockFactorsOneDayGet(tradeDate=yesterd
ay.strftime('%Y%m%d'),secID=account.universe,field=u"secID,ROE",
pandas="1")
        ROE.columns = ['secID','ROE']
        ROE['ticker'] = ROE['secID'].apply(lambda x: x[0:6])
        ROE.set_index('ticker',inplace=True)
        ep = ROE['ROE'].dropna().to_dict()
        signal_ROE = standardize(neutralize(winsorize(ep),yester
day.strftime('%Y%m%d')))  # 对因子进行去极值、中性化、标准化处理得信号

        # RSI
        RSI = DataAPI.MktStockFactorsOneDayGet(tradeDate=yesterd
ay.strftime('%Y%m%d'),secID=account.universe,field=u"secID,RSI",
pandas="1")
        RSI.columns = ['secID','RSI']
        RSI['ticker'] = RSI['secID'].apply(lambda x: x[0:6])
        RSI.set_index('ticker',inplace=True)
        ep = RSI['RSI'].dropna().to_dict()
        if len(ep) == 0 :
            return
        signal_RSI = standardize(neutralize(winsorize(ep),yester
day.strftime('%Y%m%d'))) # 对因子进行去极值、中性化、标准化处理得信号

        # 构建组合score矩阵
        weight = np.array([0.4, 0.3, 0.3])   #  信号合成，各因子权重
        Total_Score = DataFrame(index=RSI.index, columns=['NetProfitGrowRate','ROE','RSI'], data=0)
        Total_Score['NetProfitGrowRate'][signal_NetProfitGrowRate.keys()] = signal_NetProfitGrowRate.values()
        Total_Score['ROE'][signal_ROE.keys()] = signal_ROE.values()
        Total_Score['RSI'][signal_RSI.keys()] = signal_RSI.values()
        Total_Score['total_score'] = np.dot(Total_Score, weight)

        total_score = Total_Score['total_score'].to_dict()
        wts = simple_long_only(total_score, today.strftime('%Y%m%d'))   # 调用组合构建函数，组合构建综合考虑各因子大小，行业配置等因素，默认返回前30%的股票

        # 找载体，将ticker转化为secID
        RSI['wts'] = np.nan
        RSI['wts'][wts.keys()] = wts.values()
        RSI = RSI[~np.isnan(RSI['wts'])]
        RSI.set_index('secID', inplace=True)
        RSI.drop('RSI', axis=1, inplace=True)

        # 先卖出
        sell_list = account.valid_secpos
        for stk in sell_list:
            order_to(stk, 0)

        # 再买入
        buy_list = RSI.index
        total_money = account.referencePortfolioValue
        prices = account.referencePrice
        for stk in buy_list:
            if np.isnan(prices[stk]) or prices[stk] == 0:  # 停牌或是还没有上市等原因不能交易
                continue
            order(stk, int(total_money * RSI.loc[stk]['wts'] / prices[stk] /100)*100)
    else:
        return
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
18.5%	6.6%	10.9%	1.02	0.55	26.4%	1.48	34.3%	--

接下来，绘制组合和基准的累计收益图之差，得到alpha收益，看看效果如何。

```
((bt['portfolio_value']/bt['portfolio_value'][0] - 1) - ((1 + bt[
'benchmark_return']).cumprod() - 1)).plot(figsize=(14,7))

<matplotlib.axes.AxesSubplot at 0x34eef90>
```



可以看到，在如上三个因子驱动下的alpha收益相对来说还是比较稳定的，由于有对冲，策略是市场中性的，不论市场涨跌对我们的收益是不受影响的（当然排除一些极端情况，比如所有股票收益没有任何差异性，例如流动性危机）。

4、关于大赛

大赛的一些规则设计：

- 单子股票持股不超过10%：alpha收益并不是全仓某一只股票，然后涨停；可以看到，由于有股指期货的对冲，那么组合的持仓也应该像股指期货一样各行业配置很均匀，组合构建函数simple_long_only()充分考虑了行业配置、各因子alpha贡献等因素
- 股票仓位在任意三个以上收盘日低于80%则不达标：因为alpha收益已经非常稳健，那么增加本金的投入只会带来更多的收益，何乐而不为呢？
- 相对HS300，强制平仓线90%：近似于强平线为0.9，当所选因子不能持续带来alpha收益时，有必要对因子要仔细考虑了。
- 不能投资流动性过差以及刚上市的股票：alpha收益追求的是稳定性，没有必要去承受额外的流动性风险以及其他风险。

5、后话

优矿提供了将近300个基础因子(包含价值/动量/质量/成长/情绪等维度)供用户研究和合成：

可以在如下帮助页面找到这些因子
https://uqer.io/help/api/search/MktStockFactors?page=1

在投资研究寻找新的因子就是专业量化研究员日常的工作，在研究过程中希望找到：有故事的因子，符合经济学原理，有投资逻辑

希望优矿用户在这里开启你的众包版对冲基金之旅！！！

量化分析师的Python日记【第15天：如何在优矿上搞一个wealthfront出来】

> 来源：https://uqer.io/community/share/5670da3c228e5b8d81f00a87

本篇结合wealthfront投资白皮书，详细介绍并开源了wealthfront的资产配置方法

目前国内也出来很多创业团队做这块，其实没有太多神秘的黑科技，优矿瞬间搞定

结合我国实情，在本篇中给出一个中国版的wealthfront实例

具体wealthfront投资白皮书，参见链接 https://research.wealthfront.com/whitepapers/investment-methodology/

wealthfront介绍

- wealthfront是美国知名的在线资产管理平台，目前其管理的资产总额已超过25亿美元https://www.wealthfront.com/
- 以ETF为标的，资产配置为理念，根据客户不同的风险偏好构建不同的投资组合
- 实时跟踪用户组合持仓，给出健康评分，同时根据市场情况和客户风险偏好变化帮用户调整到最优持仓

投资理念

- 价值投资（长线投资）：享受经济增长带来的资本增值，并非每个人都有时间看盘，短线投资太累不靠谱
- 被动投资：国内外众多研究表明，长期来看，主动型投资的收益不一定跑得过被动型投资，同时被动投资更容易分散风险
- 资产配置：不要把鸡蛋放在同一个篮子里，做好资产配置，分散掉没有价值非系统性风险

下面，将按照完整的投资步骤详细描述（主要包括选取资产大类，相关性矩阵，构建有效前沿，资产配置方法，组合监控和动态调仓）

并结合中国实情，以具体的例子展开上述过程

1 选取资产大类

- 所选取的资产大类要尽可能涵盖整个市场，而且不同收益特征的都要包括进来，大致可以分为：权益类，债券类和货币类
- 对于每一大类资产，结合我国实情又可以细分很多小类，小类数量不在于多，在于彼此间能够有效地分散掉非系统性风险，使efficient frontier最优

- 最后，选取出来七类资产：国内股市（大盘股、中盘股、小盘股）、国外股市（美股）、国内债券（国债、企业债）、货币基金
- 由于是被动投资，考虑历史数据长短问题，上述七类资产分别以沪深300、中证500、创业板、标普500、上证国债、上证企业债、博时现金收益A为代表

不失一般性，下面以过去三年的历史数据计算标的的相关指标，需要特别关注的是相关性系数矩阵，因为需要寻找的是相关性不强甚至是负相关的标的

```
#  数据准备
import numpy as np
import pandas as pd
from pandas import DataFrame, Series
from matplotlib import pyplot as plt

startdate = '20120101'
enddate = '20150101'

secIDs = ['000300.ZICN','000905.ZICN','399006.ZICN','SPX.ZIUS','000012.ZICN','000013.ZICN','050003.OFCN']   # 七类资产的secID
rtns = DataFrame()
for i in range(len(secIDs)-1):
    cp = DataAPI.JY.MktIdxdJYGet(indexID=secIDs[i],startDate=startdate,endDate=enddate,field=u"secShortName,tradeDate,closeIndex",pandas="1")
    cp.sort(columns = 'tradeDate', inplace = True)
    cp.columns = ['secShortName','tradeDate','return']
    cp['return'][1:] = 1.0 * cp['return'][1:].values / cp['return'][:-1].values - 1
    cp['return'][:1] = 0
    rtns = pd.concat([rtns,cp],axis = 0)  #  dataframe连接操作
cp = DataAPI.JY.FundNavJYGet(secID=secIDs[len(secIDs)-1],beginDate=startdate,endDate=enddate,field=u"secShortName,endDate,dailyProfit",pandas="1")
cp.columns = ['secShortName','tradeDate','return']
cp['return'] = cp['return'].values / 10000
rtns = pd.concat([rtns,cp],axis = 0)
rtn_table = pd.crosstab(rtns['tradeDate'],rtns['secShortName'], values = rtns['return'], aggfunc = sum)  #  一维表变为二维表
rtn_table = rtn_table[[6,2,3,5,1,0,4]]
rtn_table.fillna(0, inplace = True)  #  将NaN置换为0
```

运行上述代码，便可以看到整理好的日度收益数据如下所示

```
rtn_table.head(20)
```

secShortName	沪深300	创业板指	博时现金A	标普500	企债指

tradeDate					
2012-01-03 00:00:00	0.000000	0.000000	0.000391	0.000000	0.00000
2012-01-04 00:00:00	0.000000	0.000000	0.000139	0.000188	0.00000
2012-01-05 00:00:00	-0.009727	-0.056851	0.000121	0.002944	-0.0006
2012-01-06 00:00:00	0.006242	0.003164	0.000120	-0.002537	0.00006
2012-01-08 00:00:00	0.000000	0.000000	0.000236	0.000000	0.00000
2012-01-09 00:00:00	0.034039	0.034977	0.000122	0.002262	0.00040
2012-01-10 00:00:00	0.033261	0.034704	0.000126	0.008886	0.00006
2012-01-11 00:00:00	-0.004797	0.002080	0.000128	0.000310	0.00020
2012-01-12 00:00:00	-0.000160	-0.011213	0.000128	0.002337	0.00067
2012-01-13 00:00:00	-0.016791	-0.061714	0.000130	-0.004948	0.00006
2012-01-15 00:00:00	0.000000	0.000000	0.000259	0.000000	0.00000
2012-01-16 00:00:00	-0.020331	-0.048298	0.000127	0.000000	0.00000
2012-01-17 00:00:00	0.049006	0.045401	0.000119	0.003553	-0.0000
2012-01-18 00:00:00	-0.015610	-0.057010	0.000116	0.011108	-0.0001
2012-01-19 00:00:00	0.019057	0.012626	0.000125	0.004939	0.00040
2012-01-20 00:00:00	0.014479	0.021460	0.000128	0.000669	0.00114
2012-01-23 00:00:00	0.000000	0.000000	0.000000	0.000471	0.00000
2012-01-24 00:00:00	0.000000	0.000000	0.000000	-0.001026	0.00000

2012-01-25 00:00:00	0.000000	0.000000	0.000000	0.008679	0.00000
2012-01-26 00:00:00	0.000000	0.000000	0.000000	-0.005754	0.00000

先随便计算一下指标，年化收益率，年化标准差

```
rtn_table.mean() * 250

secShortName
沪深300     0.132476
创业板指      0.229035
博时现金A     0.034695
标普500     0.134380
企债指数      0.053748
中证500     0.157495
国债指数      0.027760
dtype: float64
```

```
rtn_table.std() * np.sqrt(250)

secShortName
沪深300     0.181934
创业板指      0.249659
博时现金A     0.001477
标普500     0.105316
企债指数      0.006232
中证500     0.197669
国债指数      0.006012
dtype: float64
```

接下来计算我们关心的相关系数矩阵

```
rtn_table.corr()
```

secShortName	**沪深300**	**创业板指**	**博时现金A**	**标普500**	**企债指**
secShortName					
沪深300	1.000000	0.570628	0.002318	0.063094	0.0743
创业板指	0.570628	1.000000	-0.018372	0.022396	0.1180
博时现金A	0.002318	-0.018372	1.000000	-0.013068	-0.0909
标普500	0.063094	0.022396	-0.013068	1.000000	0.0357
企债指数	0.074392	0.118028	-0.090991	0.035720	1.0000
中证500	0.835496	0.834778	-0.005413	0.043377	0.1293
国债指数	-0.024434	-0.046782	-0.017517	0.001724	0.2097

从上面可以看到：

- 收益相对稳定的债券和货币与其他类的资产相关性都比较低，一方面通过配置可以分散非系统性风险，另一方面在市场不好时可以提供相对稳健的收益
- 标普和国内股市相关性弱，这在进行权益类配置时特别有效，比如在12-14年我国股市表现不佳时，标普500却走出了一波慢牛

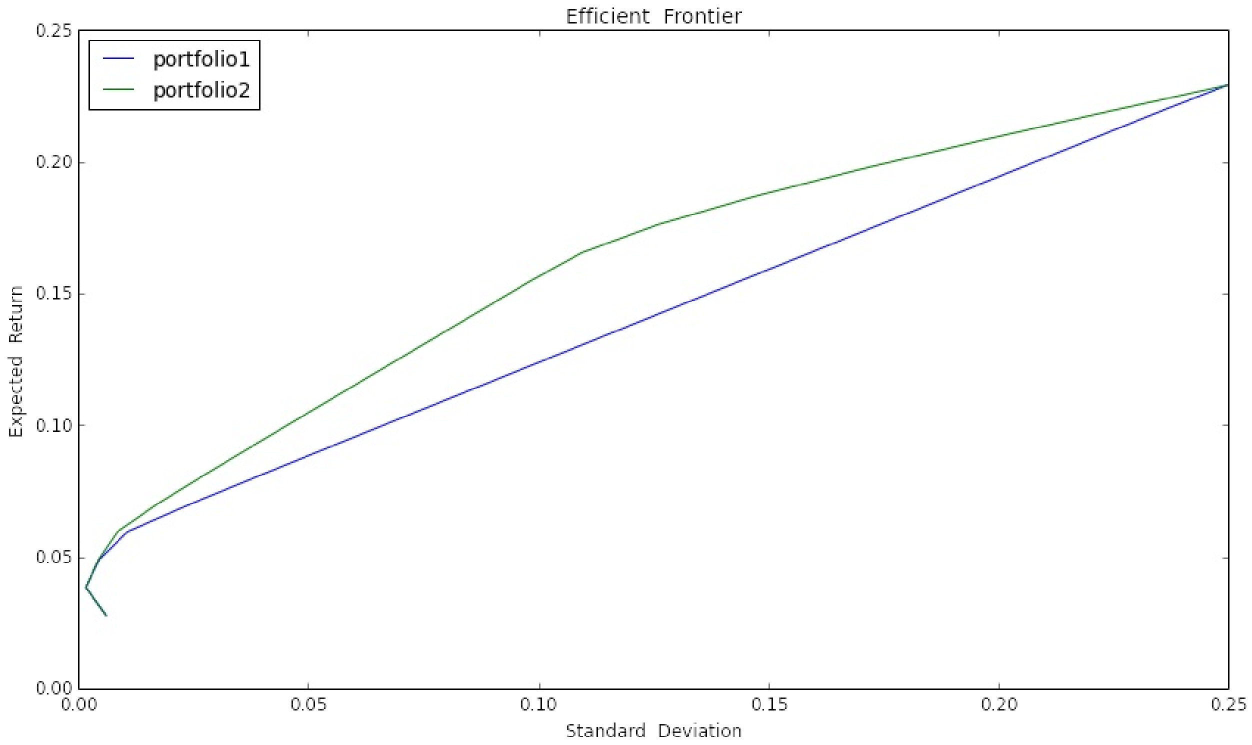
接下来，就来对比绘制efficient frontier，从实际中直观感知资产多元化带来的风险分散效果

- 构建两个组合作为对比，组合一仅包含沪深300、中证500、创业板、国债、货币，组合二则包含了组合一、标普500、企业债
- 绘制effiecient frontier用到了凸优化包cvxopt，关于cvxopt的用法详细介绍，参见。。。。。
- 在构建efficient frontier中，预期收益采取市场中性原则，用过去三年的平均收益

```
from cvxopt import matrix, solvers

portfolio1 = [0,1,2,4,6]
portfolio2 = range(7)
cov_mat = rtn_table.cov() * 250   # 协方差矩阵
exp_rtn = rtn_table.mean() * 250   # 标的预期收益

def cal_efficient_frontier(portfolio):
    #简单的容错处理
    if len(portfolio) <= 2 or len(portfolio) > 7:
        raise Exception('portfolio必须为长度大于2小于7的list！')
    # 数据准备
    cov_mat1 = cov_mat.iloc[portfolio][portfolio]
    exp_rtn1 = exp_rtn.iloc[portfolio]
    max_rtn = max(exp_rtn1)
    min_rtn = min(exp_rtn1)
    risks = []
    returns = []
    # 均匀选取20个点来作图
    for level_rtn in np.linspace(min_rtn, max_rtn, 20):
        sec_num = len(portfolio)
        P = 2*matrix(cov_mat1.values)
        q = matrix(np.zeros(sec_num))
        G = matrix(np.diag(-1 * np.ones(sec_num)))
        h = matrix(0.0, (sec_num,1))
        A = matrix(np.matrix([np.ones(sec_num),exp_rtn1.values])
)
        b = matrix([1.0,level_rtn])
        solvers.options['show_progress'] = False
        sol = solvers.qp(P,q, G, h, A, b)
        risks.append(sol['primal objective'])
        returns.append(level_rtn)
    return np.sqrt(risks), returns

#  计算画图数据
risk1, return1 = cal_efficient_frontier(portfolio1)
risk2, return2 = cal_efficient_frontier(portfolio2)
```

在上述准备好数据之后，接下来就构建组合一(沪深300、中证500、创业板、国债、货币)和组合二(组合一 + 标普500、企业债)的efficient frontier

```
fig = plt.figure(figsize = (14,8))
ax1 = fig.add_subplot(111)
ax1.plot(risk1,return1)
ax1.plot(risk2,return2)
ax1.set_title('Efficient  Frontier', fontsize = 14)
ax1.set_xlabel('Standard  Deviation', fontsize = 12)
ax1.set_ylabel('Expected  Return', fontsize = 12)
ax1.tick_params(labelsize = 12)
ax1.legend(['portfolio1','portfolio2'], loc = 'best', fontsize =
14)

<matplotlib.legend.Legend at 0x5e10990>
```



从上图可以很直观地看到：

- 组合一所包含的标的较少，相关性也较高，所以efficient frontier基本为一条直线，分散风险作用不明显
- 组合二引入了和其他资产相关性都不高的标普500，使得efficient frontier得到了很大程度的优化
- 由此也可以知晓，当加入某个标的之后能够使得efficient frontier得到改进的话，那么加入该资产到组合中是非常有必要的

接下来，给定预期收益，得到最优权重

- 如上分析，在得到最优的efficient frontier之后（本例中为组合二），便可以在资产池中进行资产配置

- 假定某投资者的风险厌恶系数为3（系数越大，表明越厌恶风险，投资更保守），那么就可以借鉴均方差优化来计算自由的资产配置权重

附：均值方差优化简介

- 均值方差模型可以理解成是一个效用函数的最大化，目标效用 = 预期收益带来的正效用 - 承担风险带来的负效用，用公式表示如下：

$$max(u' * w - \lambda/2 * w' * \Sigma * w)$$

$$\sum w = 1$$

$$0 \leq w \leq 1$$

上式中：u为资产的预期收益率，`w`为资产权重，`λ`为投资者风险厌恶系数，`Σ`为方差协方差矩阵

- 一般情况下，通过给定`u`、`λ`、`Σ`，就可以计算最优的资产配置权重w
- 上式表明，我们仅考虑long only时的情况

```
risk_aversion = 3
P = risk_aversion * matrix(cov_mat.values)
q = -1 * matrix(exp_rtn.values)
G = matrix(np.vstack((np.diag(np.ones(len(exp_rtn))),np.diag(-np
.ones(len(exp_rtn))))))
h = matrix(np.array([np.ones(len(exp_rtn)),np.zeros(len(exp_rtn)
)]).reshape(len(exp_rtn)*2,1))
A = matrix(np.ones(len(exp_rtn)),(1,len(exp_rtn)))
b = matrix([1.0])
solvers.options['show_progress'] = False
sol = solvers.qp(P,q, G, h, A, b)
DataFrame(index=exp_rtn.index,data = np.round(sol['x'],2), colum
ns = ['weight'])  # 权重精确到小数点后两位
```

weight	
secShortName	
沪深300	0.00
创业板指	0.58
博时现金A	0.00
标普500	0.42
企债指数	0.00
中证500	0.00
国债指数	0.00

- 如上所示，在我们的实例中，最优权重配置为58%的创业板，42%的标普500，只配置了两个标的，而且都是权益类的，相对风险较大，这主要是因为风险厌恶系数给定值较小的缘故
- 对于如上配置过程只是一个范例，除此之外，我们还可以定义很多个性化的东西，比如：wealthfront为了保证配置的均匀性，要求每一大类的配置比例都不得超过35%，这些个性化的条件，只用简单的加在优化函数的限制条件里就实现了，读者可以自行实践

最后，组合监控和动态调仓（rebalance）

承接上文，在构建好组合之后，

以上是对wealthfront投资方法的整体介绍，同时详细介绍了我国版的实例，后期优矿可以让大家自己产生这样的策略在优矿上跑，比其他创业产品透明的多喔。

第二部分 股票量化相关

一 基本面分析

1.1 alpha 多因子模型

破解Alpha对冲策略——观《量化分析师Python日记第14天》有感

> 来源：https://uqer.io/community/share/55ff6ce9f9f06c597265ef04

写在最前面：

> 1. 不知不觉逛社区快半年了，通过优矿平台认识了很多大牛，真心获益匪浅，不管是编程方面还是金融方面，在此真心感谢优矿平台，为你们的分享精神点个赞！
> 2. 再来说说写作目的吧，估计自己还算是个社区活跃用户，之前也分享过一些实用的帖子，然后某一天系统就发通知说感谢我对优矿的支持，内存已经帮我加到1GB，有效期1个月，从此妈妈再也不用担心我跑策略out of memory了，嘿嘿~所以呢，一方面传承优矿分享交流精神，另一方面也希望通过多为社区做贡献获得1GB内存更长时间，来个永久版最好啦！

本篇缘由：

> 1. 最近市场的起起伏伏真是惊心动被迫，股指期货投机交易也被狠狠的限制了，各种公募私募产品清盘处理。。。
> 2. 我开始思考，是市场的问题还是投资者的问题，究竟怎样的策略才能成为常胜将军，如果可以选择，我宁可做市场上的寿星而不是明星。
> 3. 优矿给了我这个启示，感谢社区大牛薛昆Kelvin的帖子量化分析师的Python日志第14天，告诉我该如何去做Alpha对冲策略
> 4. 如果你读到这里，强烈建议你先去认真读完上述帖子，然后接着往下看。
> 5. 上述帖子，对Alpha对冲策略从理论原理再到代码实现都进行了详细的讲解，但是对于其中提到的一些新的函数（而且是特别重要的函数讲述的不是特别清楚，只是说了一下大体方向），于是乎，笔者就顺藤摸瓜，探探究竟。本篇就是讲述自己对上述帖子的一些测试和自我体会，当然也秉承分享精神，展示笔者得意的Fama-french三因子策略。。。老舍不得的了，一定要给我加内存啊！！

关于本篇：

> 1. 本篇首先对帖子中出现的不太清晰的函数进行相关猜测与测试，包括因子信号处理函数:去极值（`winsorize`）、中性化（`neutralize`）、标准化（`standardize`）
> 2. 随后，对组合构建函数`simple_long_only`进行猜测
> 3. 最后，以Fama-French三因子构建策略进行回测展示

首先来看三个因子处理函数，笔者结合各家券商研究报告中提到的类似处理进行大胆猜测，并进行测试

> 首先是去极值函数`winsorize`，大量券商研究报告都提到了这个方法，业内常用所谓的“3σ”原则，也就是先根据因子样本计算出标准差，然后将其中大于`u+3σ`的置换为`u+3σ`，将小于`u-3σ`的置换为`u-3σ`，这样做的好处是可以消除因子极值对因子实际效果造成的不必要影响，下面举例来说明

Tips：读者可以首先在code模式下输入`winsorize`？然后运行便可以得到该函数的说明文档。

```
import numpy as np
import pandas as pd

universe = set_universe('SH50')  # 以上证50市盈率因子进行说明
data = DataAPI.MktStockFactorsOneDayGet(tradeDate='20150916', se
cID=universe, field='ticker,PE',pandas='1').set_index('ticker')
data = data['PE'].to_dict()  # winsorize之前数据
new_data = winsorize(data)  # winsorize之后数据
df = pd.DataFrame(data=0, index=map(lambda x: x[:6], universe),
columns=['before winsorize','after winsorize'])
df['before winsorize'][data.keys()] = data.values()  # 对比两者数
据进行展示
df['after winsorize'][new_data.keys()] = new_data.values()
df.reset_index(inplace=True)
```

接下来，我们可以看看`winsorize`前后数据的变化

```
df
```

	index	before winsorize	after winsorize
0	600000	5.9624	5.962400
1	600104	7.0826	7.082600
2	600050	33.0181	33.018100
3	600036	7.7177	7.717700
4	600030	8.8612	8.861200

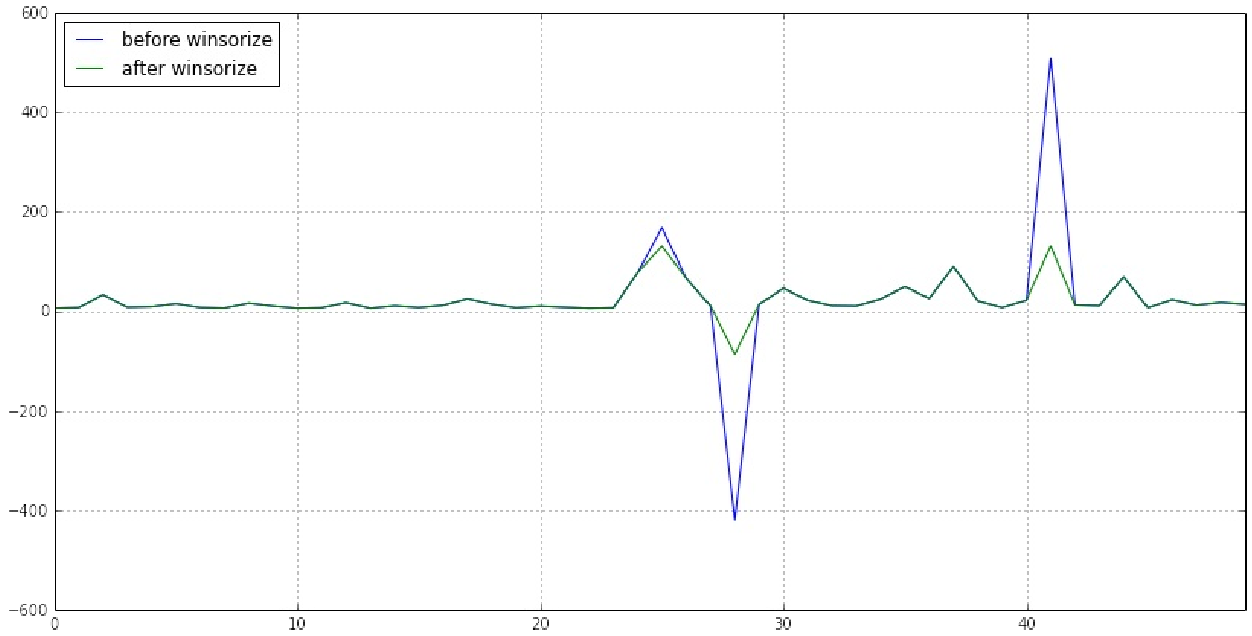
5	600028	14.9741	14.974100
6	600016	7.1609	7.160900
7	600015	6.1384	6.138400
8	600519	16.0515	16.051500
9	601006	10.1080	10.108000
10	601398	5.8342	5.834200
11	600048	6.7609	6.760900
12	601628	17.1175	17.117500
13	601166	5.7557	5.755700
14	601318	10.6240	10.624000
15	601328	7.1577	7.157700
16	601088	11.7254	11.725400
17	601857	24.8742	24.874200
18	601601	13.9766	13.976600
19	601169	6.7160	6.716000
20	600837	10.0223	10.022300
21	601668	7.6202	7.620200
22	601288	5.7281	5.728100
23	601818	6.6163	6.616300
24	600111	77.2052	77.205200
25	601989	167.9813	130.981297
26	601766	65.9834	65.983400
27	600585	10.3818	10.381800
28	600010	-419.8877	-86.645491
29	601901	13.5459	13.545900
30	600256	45.9370	45.937000
31	600887	21.8070	21.807000
32	601688	10.9483	10.948300
33	600999	10.3119	10.311900
34	600518	24.2080	24.208000

35	600406	49.2598	49.259800
36	600018	25.5060	25.506000
37	600637	89.8189	89.818900
38	600089	20.5017	20.501700
39	601998	7.0787	7.078700
40	600109	21.8898	21.889800
41	600150	508.1892	130.981297
42	600690	12.3600	12.360000
43	600583	10.8489	10.848900
44	600893	68.7643	68.764300
45	601988	6.7982	6.798200
46	601390	23.0430	23.043000
47	600958	12.0603	12.060300
48	601186	17.4490	17.449000
49	601800	13.6934	13.693400

可以很明显看到，大部分值都没变，第25、28、41行所在股票的PE值得到了处理，过大或者过小都会被视为极值，会得到调整。笔者也计算了 `u+3σ` ，发现调整结果并没有完全按照 `3σ` 原则，但是 `winsorize` 的作用已经得到了测试检验。

下面绘制一个对比图可以更明显看到 `winsorize` 前后数据的变化

```
df.plot(figsize=(14,7))

<matplotlib.axes.AxesSubplot at 0x42bfa10>
```



接下来是中性化函数neutralize

> `neutralize` 函数不太好做测试，但是根据 `neutralize` 的说明文档，可以猜个大概出来。该函数的定义形式是 `neutralize(raw_data, target_date, risk_module='short', industry_` ，可以看到函数需要选择风险模型、行业分类，由此不难推测出，输入原始因子数据，由于原始因子数据是所有行业的，这里可能按照行业分类，对因子进行了行业中性处理（大概可以理解为将因子间的行业差异消除了，比如互联网行业和银行之间的PE本来就不在一个level上， `neutralize` 之后可能就消除了这个因素，有点像对季节数据进行季节平滑处理）

再来看看标准化函数 `standardize`

> 这个函数应该非常好理解，也非常好测试，很多券商的研究报告都有提到过该处理方法，简单来讲就是 `（因子值 - 因子均值）/ 因子标准差` ，下面接前面的例子对 `standardize` 进行测试

```
data1 = standardize(data)
df1 = pd.DataFrame(data=0, index=map(lambda x: x[:6], universe),
 columns=['raw data','standardize function','standardize myself'
])
df1['raw data'][data.keys()] = data.values()  # 原始数据
df1['standardize function'][data1.keys()] = data1.values()  # 通
过standardize函数计算的值
df1['standardize myself'] = (df1['raw data'] - df1['raw data'].m
ean()) / df1['raw data'].std() # 自己计算的值
df1
```

	raw data	standardize function	standardize myself
600000	5.9624	-0.178463	-0.178463

600104	7.0826	-0.167042	-0.167042
600050	33.0181	0.097395	0.097395
600036	7.7177	-0.160566	-0.160566
600030	8.8612	-0.148907	-0.148907
600028	14.9741	-0.086580	-0.086580
600016	7.1609	-0.166243	-0.166243
600015	6.1384	-0.176669	-0.176669
600519	16.0515	-0.075595	-0.075595
601006	10.1080	-0.136195	-0.136195
601398	5.8342	-0.179770	-0.179770
600048	6.7609	-0.170322	-0.170322
601628	17.1175	-0.064726	-0.064726
601166	5.7557	-0.180571	-0.180571
601318	10.6240	-0.130934	-0.130934
601328	7.1577	-0.166276	-0.166276
601088	11.7254	-0.119704	-0.119704
601857	24.8742	0.014361	0.014361
601601	13.9766	-0.096751	-0.096751
601169	6.7160	-0.170779	-0.170779
600837	10.0223	-0.137069	-0.137069
601668	7.6202	-0.161560	-0.161560
601288	5.7281	-0.180852	-0.180852
601818	6.6163	-0.171796	-0.171796
600111	77.2052	0.547924	0.547924
601989	167.9813	1.473472	1.473472
601766	65.9834	0.433507	0.433507
600585	10.3818	-0.133403	-0.133403
600010	-419.8877	-4.520406	-4.520406
601901	13.5459	-0.101142	-0.101142
600256	45.9370	0.229116	0.229116

600887	21.8070	-0.016912	-0.016912
601688	10.9483	-0.127627	-0.127627
600999	10.3119	-0.134116	-0.134116
600518	24.2080	0.007568	0.007568
600406	49.2598	0.262995	0.262995
600018	25.5060	0.020802	0.020802
600637	89.8189	0.676533	0.676533
600089	20.5017	-0.030221	-0.030221
601998	7.0787	-0.167081	-0.167081
600109	21.8898	-0.016068	-0.016068
600150	508.1892	4.942212	4.942212
600690	12.3600	-0.113234	-0.113234
600583	10.8489	-0.128641	-0.128641
600893	68.7643	0.461861	0.461861
601988	6.7982	-0.169941	-0.169941
601390	23.0430	-0.004310	-0.004310
600958	12.0603	-0.116289	-0.116289
601186	17.4490	-0.061346	-0.061346
601800	13.6934	-0.099638	-0.099638

可以看到，猜测完全正确，得到的结果一模一样！！

好了，三个因子处理函数已经猜完了，再来看看大头吧，组合构建函数 `simple_long_only` ，同样，结合帮助文档来看。

> 在《量化分析师日记》中对该函数的说明是：“组合构建综合考虑各因子大小，行业配置等因素，默认返回前30%的股票”。给我的直观理解是，倘若给定100个股票，那么函数就根据股票的因子值以及行业分类选出其中最好的30%只股票，也就是30只股票以及他们各自的建仓权重。至于内部怎么实现的，我也只能猜测，估计选出来的30只股票行业配置要比较均匀，而且要因子值要优于没有被选中的股票，比如我要选低估值的股票，那么就优先选择低PE的，但是又不能直接选PE排名30%以下的那30只股票，因为还要考虑到行业配置均匀的问题，不然选出来的很可能都是同一个行业的（比如银行、钢铁之类的），所以，个人猜测组合构建函数就是在因子值和行业配置均匀之间进行博弈，求得一个最优组合。。。下面，还是写出猜想过程。

```
factor = standardize(neutralize(winsorize(data),'20150916'))   #
  将原始数据进行处理，得到最终因子值
weight = simple_long_only(factor, '20150915')  # 根据因子构建组合，
获得权重
df_factor = pd.DataFrame(data=np.nan, index=map(lambda x: x[:6],
 universe), columns=['factor','weight'])  # 将因子值和最后的持仓权重
对比
df_factor['factor'][factor.keys()] = factor.values()
df_factor['weight'][weight.keys()] = weight.values()
df_factor
```

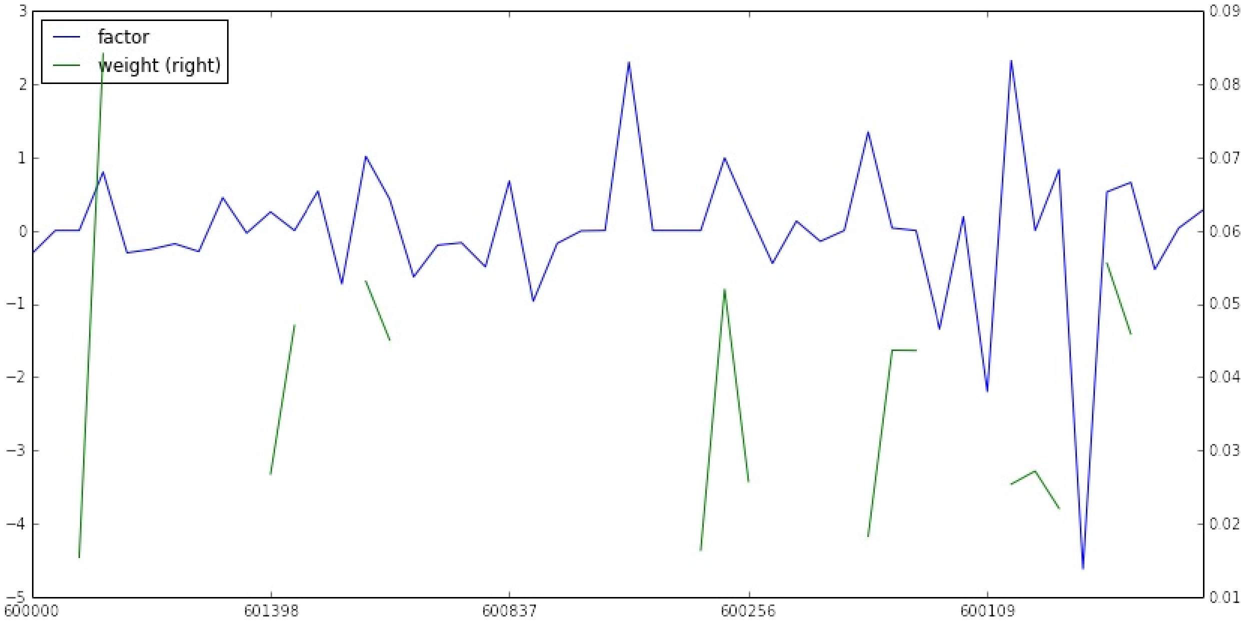
| | factor | weight | } --- | --- | | 600000 | -3.236122e-01 | NaN | | 600104 | -7.364325e-15 | NaN | | 600050 | 3.671396e-15 | 0.015365 | | 600036 | 7.982552e-01 | 0.084137 | | 600030 | -3.042384e-01 | NaN | | 600028 | -2.572782e-01 | NaN | | 600016 | -1.811580e-01 | NaN | | 600015 | -2.869542e-01 | NaN | | 600519 | 4.488652e-01 | 0.044730 | | 601006 | -3.457882e-02 | NaN | | 601398 | 2.537750e-01 | 0.026748 | | 600048 | 6.228453e-15 | 0.047034 | | 601628 | 5.374184e-01 | NaN | | 601166 | -7.274821e-01 | NaN | | 601318 | 1.013404e+00 | 0.053078 | | 601328 | 4.274117e-01 | 0.045050 | | 601088 | -6.332870e-01 | NaN | | 601857 | -2.000576e-01 | NaN | | 601601 | -1.666304e-01 | NaN | | 601169 | -4.956554e-01 | NaN | | 600837 | 6.762916e-01 | 0.035421 | | 601668 | -9.654556e-01 | NaN | | 601288 | -1.756114e-01 | NaN | | 601818 | -5.174820e-03 | NaN | | 600111 | -4.047149e-14 | NaN | | 601989 | 2.298816e+00 | NaN | | 601766 | -1.463175e-14 | NaN | | 600585 | -2.109168e-14 | NaN | | 600010 | 3.274110e-14 | 0.016346 | | 601901 | 9.923986e-01 | 0.051978 | | 600256 | 2.572782e-01 | 0.025743 | | 600887 | -4.488652e-01 | NaN | | 601688 | 1.289595e-01 | NaN | | 600999 | -1.478773e-01 | NaN | | 600518 | -2.324500e-14 | NaN | | 600406 | 1.344937e+00 | 0.018264 | | 600018 | 3.457882e-02 | 0.043647 | | 600637 | 1.551461e-14 | 0.043608 | | 600089 | -1.344937e+00 | NaN | | 601998 | 1.893061e-01 | NaN | | 600109 | -2.198574e+00 | NaN | | 600150 | 2.320845e+00 | 0.025369 | | 600690 | 2.358953e-14 | 0.027154 | | 600583 | 8.333446e-01 | 0.022058 | | 600893 | -4.619661e+00 | NaN | | 601988 | 5.269000e-01 | 0.055536 | | 601390 | 6.569516e-01 | 0.045871 | | 600958 | -5.311522e-01 | NaN | | 601186 | 3.286287e-02 | NaN | | 601800 | 2.756411e-01 | NaN |

从上面的对比可以看到：

总共50只股票，最后只选取了19只，比较接近30%的比例，证明了之前的猜测 由于我们假设的是要买高PE的，所以可以看到，最后选出的19只股票的因子值（PE）相对没有选中的都比较高，而且绝大多数权重都和因子值呈比例出现，至于没有呈现比例的应该是基于行业配置均匀的考虑，所以说之前的猜想还是非常靠谱的，有兴趣的读者可以自行进一步研究。

同样，也给出对比分析图

```
df_factor.plot(secondary_y='weight',figsize=(14,7))

<matplotlib.axes.AxesSubplot at 0x403e590>
```



终于写到最后一部分，内心是无比的纠结。。

有了因子处理以及组合构建之后，我们就可以自己找因子来构建组合了，大赛方还专门有获取因子数据的DataAPI，真心赞一个！

那么，我就要开始分享我的策略了。。。优矿工作人员看到的话一定要给我加内存，或者什么VIP账号啊啊啊！！！

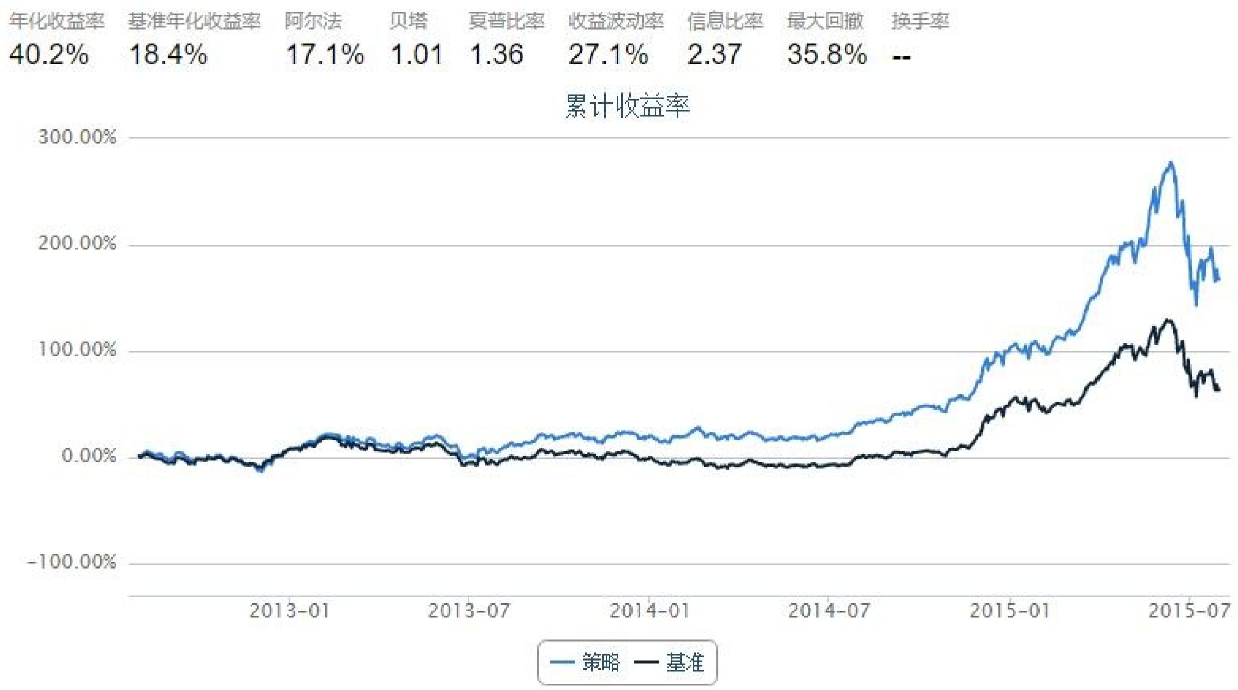
策略思路来源就是经典的Fama-French三因子模型，三因子模型告诉我们，股票的收益可以由这三个因子来解释：市场beta、股票市值、股票估值；同时，低估值、低市值的股票能够获得超额收益

那么，估值可以用市盈率来衡量(PE)，市值可以用流通市值来衡量（LFLO），下面就给出策略回测效果

PS：回测区间从2012年8月1日~2015年8月1日，股票池为中证800，每月第一个交易日建仓

```
# 导入包
from CAL.PyCAL import *
import numpy as np
import pandas as pd

# 构建日期列表，以保证每月第一个交易日建仓
data=DataAPI.TradeCalGet(exchangeCD=u"XSHG",beginDate=u"20120731"
,field=['calendarDate','isWeekEnd','isMonthEnd'],pandas="1")
data = data[data['isMonthEnd'] == 1]
date_list = map(lambda x: x[0:4]+x[5:7]+x[8:10], data['calendarD
ate'].values.tolist())

start = '2012-08-01'                       # 回测起始时间
end = '2015-08-01'      # 回测结束时间
universe = set_universe('HS300') + set_universe('ZZ500')   # 股票池

benchmark = 'HS300'                       # 策略参考标准
capital_base = 10000000                     # 起始资金
freq = 'd'                              # 策略类型，'d'表示日间策略
使用日线回测，'m'表示日内策略使用分钟线回测
refresh_rate = 1                          # 调仓频率
commission = Commission(buycost=0.0008, sellcost=0.0008)

# 日期处理相关
cal = Calendar('China.SSE')
period = Period('-1B')

def initialize(account):                   # 初始化虚拟账户状态
    pass

def handle_data(account):                  # 每个交易日的买入卖出指令


    today = account.current_date
    today = Date.fromDateTime(account.current_date)  # 向前移动一
个工作日
    yesterday = cal.advanceDate(today, period)
    yesterday = yesterday.toDateTime().strftime('%Y%m%d')
    if yesterday in date_list:

        Factor1 = DataAPI.MktStockFactorsOneDayGet(tradeDate=yes
terday,secID=account.universe[:400],field=['secID','PE','LFLO'],
pandas="1")
        Factor2 = DataAPI.MktStockFactorsOneDayGet(tradeDate=yes
terday,secID=account.universe[400:],field=['secID','PE','LFLO'],
pandas="1")
        Factor = pd.concat([Factor1, Factor2])
        Factor['ticker'] = Factor['secID'].apply(lambda x: x[0:6
])
        Factor.set_index('ticker',inplace=True)

        # 市盈率PE
        Factor['PE'] = 1.0 / Factor['PE']  # 低市盈率
        factor = Factor['PE'].dropna().to_dict()
        signal_PE = standardize(neutralize(winsorize(factor),yes
terday))   # 因子处理

        # 对数流通市值LFLO
        Factor['LFLO'] = 1.0 / Factor['LFLO']  # 低市值
        factor = Factor['LFLO'].dropna().to_dict()
        signal_LFLO = standardize(neutralize(winsorize(factor),y
esterday))  # 因子处理
```

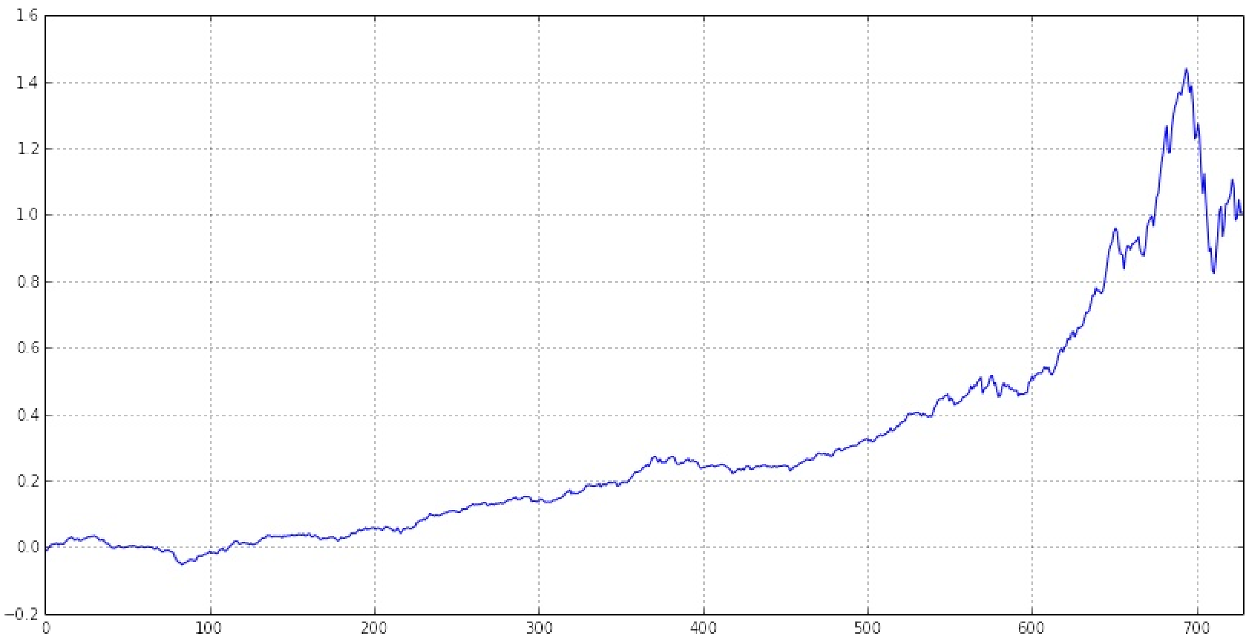
```
        # 构建组合score矩阵
        Total_Score = pd.DataFrame(index=Factor.index, columns=factor_name, data=0)
        Total_Score['PE'][signal_PE.keys()] = signal_PE.values()
        Total_Score['LFLO'][signal_LFLO.keys()] = signal_LFLO.values()
        Total_Score['total_score'] = np.dot(Total_Score, np.array([0.5, 0.5]))  # 综合两个因子的大小，不失一般性，等权求和
        total_score = Total_Score['total_score'].to_dict()

        wts = simple_long_only(total_score,yesterday)  # 组合构建函数
        Factor['wts'] = np.nan
        Factor['wts'][wts.keys()] = wts.values()
        Factor = Factor[~np.isnan(Factor['wts'])]
        Factor.set_index('secID', inplace=True)
        Factor.drop(factor_name, axis=1, inplace=True)

        # 先卖出
        sell_list = account.valid_secpos
        for stk in sell_list:
            order_to(stk, 0)

        # 再买入
        buy_list = Factor.index
        total_money = account.referencePortfolioValue
        prices = account.referencePrice
        for stk in buy_list:
            if np.isnan(prices[stk]) or prices[stk] == 0:  # 停牌或是还没有上市等原因不能交易
                continue
            order(stk, int(total_money * Factor.loc[stk]['wts'] / prices[stk] /100)*100)
    else:
        return
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
40.2%	18.4%	17.1%	1.01	1.36	27.1%	2.37	35.8%	--

接下来，用组合累计收益减去基准累计收益就得到alpha收益，如下所示：

```
((bt['portfolio_value']/bt['portfolio_value'][0] - 1) - ((1 + bt[
'benchmark_return']).cumprod() - 1)).plot(figsize=(14,7))

<matplotlib.axes.AxesSubplot at 0x25067110>
```



可以看到，将Fama-French三因子运用到中国市场，可以得到非常稳健的Alpha收益！

写在最后面：

重要的事情说三遍，希望可以给点内存，给点内存，给点内存。。我是不是太直接了？？？欢迎交流~~~

熔断不要怕， alpha model 为你保驾护航！

> 来源：https://uqer.io/community/share/568df2a3228e5b18e4ba296e

事件回顾

- 2016年1月7日，A股开盘半个小时收盘，A股周内第二次熔断，这次只花了15分钟。。。
    - 上证综指收报3115.89点，跌幅7.32%，成交额780亿元。
    - 深证成指收报10745.74点，跌幅8.35%，成交额1080亿元。
    - 创业板指收报2254.52点，跌幅8.66%，成交额261亿元。

贪婪与恐惧

- 在宽松预期落空、宏观数据不佳、外盘不振、熔断机制的磁吸效应等众多因素的综合作用下，2016年迎来了开门黑天鹅

- 抄底的抄底、加仓的加仓，无不演绎着资本市场的两大特色：贪婪与恐惧

- 好在有段子手，喝上一碗鸡汤，看看周围人亏损情形也都一样，于是呵呵一笑，继续演绎着贪婪与恐惧

理性与思考

- 虽然有着边际效应递减规律，但每次大跌带给我的痛是愈发厉害，因为真的痛了所以我才能真正静下心来思考与总结

- 动荡的A股行情让我开始思考对投资风险的控制，单边持有多头/空头显然并不太适合A股，拿今天来说，倘若仅仅持有股票，那么亏损基本就在7%以上

- 那么如何控制风险呢？是否需要考虑对冲风险？不追求超高的收益，只求稳稳当当，在A股频发黑天鹅的现状下只求稳健收益、睡个好觉

阿尔法对冲，穿越牛熊

- 同时持有空头和多头，通过一定手段保证总体收益为正

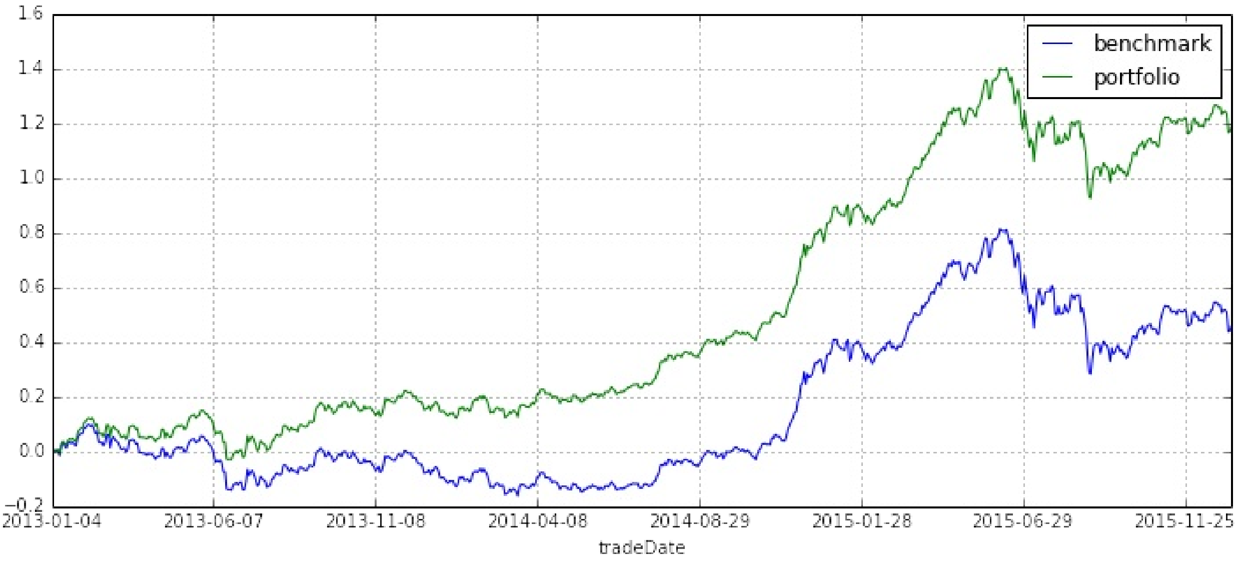
- 实际中，持有多头股票组合同时卖空股指期货，当大盘上涨时只要保证股票组合的收益大于股指期货的亏损就能实现整体盈利；同理，当大盘下跌时，保证股票的亏损小于股指期货的盈利就能实现整体正的收益

- 长时间的累积，就能实现稳健的收益，无惧黑天鹅

- 以今天为例，假设我股票多头亏损了6.9%，但期货端收益7%，从而整体我的收益是0.1%（7%-6.9%），试想，在别人亏损7%的时候我能实现盈利0.1%。。。

如下的例子，假设股票组合每天跑赢沪深300指数0.1%，看看长时间累积下来的情况如何

```
# 举例说明相对收益
data = DataAPI.MktIdxdGet(ticker='000300', beginDate='20130101',
 field='tradeDate,CHGPct', pandas='1').set_index('tradeDate').re
name(columns={'CHGPct':'benchmark'})
data['portfolio'] = data['benchmark'] + 0.001  # 每天跑赢基准0.1%
data.cumsum().plot(figsize=(12,5))

<matplotlib.axes.AxesSubplot at 0x4089c50>
```



如上图所示

- 长时间累积下来，组合相比沪深300指数的超额收益是非常可观的，而且也是非常稳健的，2年时间的超额收益将近80%！！！
- 而实际投资的阿尔法策略就是根据经济、金融理论，运用数学统计的方法，构建投资组合使其能够稳健跑赢基准指数（比如沪深300）
- 在实际操盘中，也是买入股票，卖出股指期货

如何做阿尔法模型

- 说了这么多，如何从研究到实盘，真正的做出一个阿尔法模型

- 优矿社区里已经有手把手的教程量化分析师的Python日记 第14天：如何在优矿上做Alpha对冲模型

- 每个月还有500万实盘大赛！！！

总结：受不来A股的跌宕起伏，只想睡个安稳的觉、做个安静的美男子，专心研究alpha model吧！

寻找 alpha 之: alpha 设计

> 来源：https://uqer.io/community/share/56893bb1228e5b67159beb38

一.寻找alpha, 回测时相关误区:

1. insight back 过去没有现在这样全面的分析方法
2. data back 过去没有现在这样全面整理的数据来分析
3. computational power and technology 过去没有现在的计算能力和技术

以上三个因素只是影响预测准确率的众多因素之一.

二.寻找alpha, 如何确认某策略的规则失败:

1. 回撤大于正常水平
2. 夏普下降
3. 与其他已发现的有效规则冲突

三.寻找alpha,

多策略同时使用: 某策略在50%时间能正确预测价格, 假如有10条相同准确率的策略, 把它们同时应用起来会比使用单一更好.

四.策略分类

1. 日内alphas有以下两种:
    1. 一定时间间隔的再平衡, 比如1/5/15分钟
    2. 事件驱动再平衡
2. 日间alphas:以天为最小间隔的再平衡
    1. 基于N天内数据
    2. 基于当前的快照数据
    3. 开盘/收盘 集合竞价时的交易
3. 周/月度的alphas

五 开发alpha

基于公开的信息, 找到其中的信号/模式, 这些数据处理过程越有效, 则alpha越好. 公开数据可以分为下面5个分类:

1. 量/价 使用技术分析对 量/价 做 预测/回归.
2. 基本面数据

3. 宏观数据
4. 各种文本数据:交易所公告, 公司公告, 报纸, 杂志, 新闻, 甚至社交媒体上的内容
5. 多媒体数据:音视频数据

有些数据并不能直接用来生成alpha, 但可以用来提高alpha的表现. 有下面3个例子:

1 风险因子模型: 通过控制风险因子甚至消除风险因子可以提高alpha的表现 2 关系模型: 不同票据之间在某种程度上有着关联, 在价格变动时有些是领跑的, 有些是被拖着跑的, 这些关联创造了套利的机会 3 微观模型: 提高真实交易执行中的表现 现在不是数据不够, 而是各种各样的数据太多了, 如何对数据消除噪音, 提取出我们要的信号. 问题的解决空间为非凸集, 断续, 动态. 我们可以对解决空间的范围的缩小就是通过不断使用已消化的结果来对新的数据进行处理.

六 评估alpha策略

1. 信息比率(Information Ratio) 持续表现怎样
2. 边际收益(Margin) =alpha带来的收益/交易次数, 可以得知交易次数对收效的影响, 值越高越好, 说明交易次数对收益影响越小.
3. 唯一性 与策略池里其他alpha相关性越低越好

当然还有其他测试方式来评估, 比如从票据流动性的好还是差还评估. 但记住, 对alpha的参数调整要以未来为目标, 因为就算你调整参数后, 对历史数据回测有多大提高也是没有用处的. 如果调整参数后对未来的预测贡献很小甚至负数, 但对历史数据却贡献很大, 那很可能是过度拟合了, 这时要用”样本外数据(out-of-sample data)”(机器学习里面有定义)来对alpha做评估, 而不能只是用”样本内数据(in-sample data)”

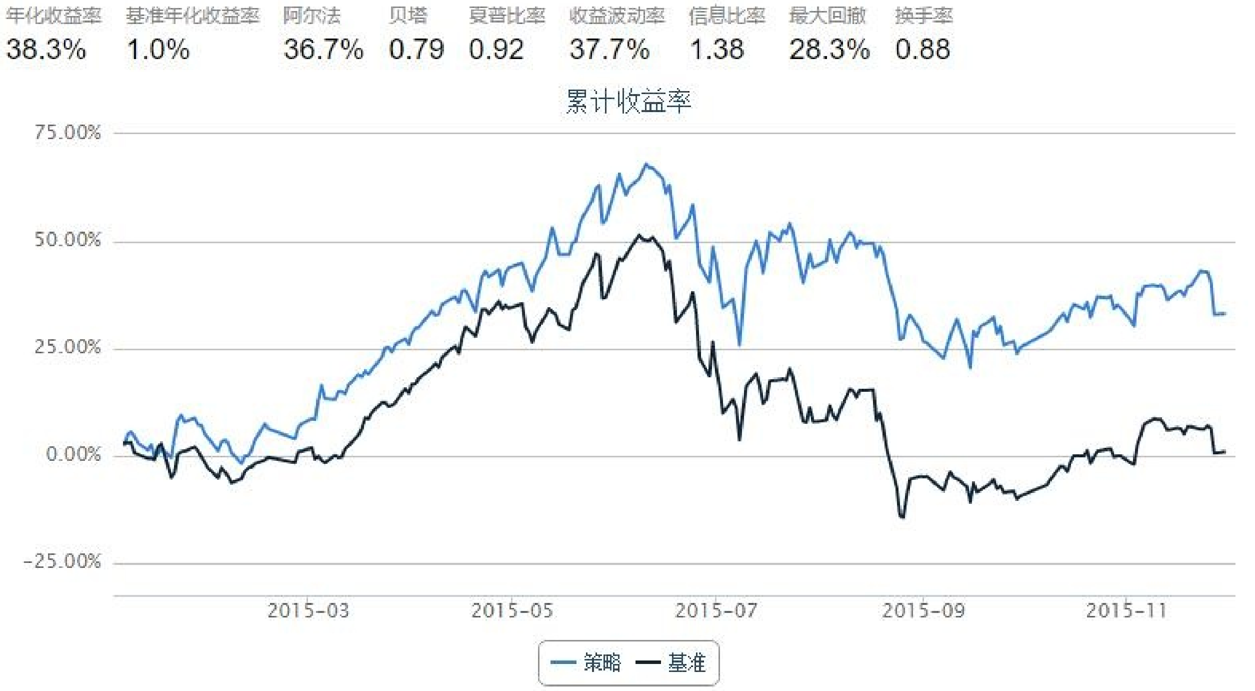
1.2 基本面因子选股

Porfolio（现金比率+负债现金+现金保障倍数）+市盈率

> 来源：https://uqer.io/community/share/566a896bf9f06c6c8a91cae7

```
?DataAPI.MktStockFactorsOneDayGet
```

```
import numpy as np
import pandas as pd
start = '2015-01-01'                       # 回测起始时间
end = '2015-11-30'                         # 回测结束时间
benchmark = 'HS300'                        # 策略参考标准
universe = set_universe('HS300')            # 证券池，支持股票和基金
capital_base = 100000
# 起始资金
freq = 'd'                             # 策略类型，'d'表示日间策略使
用日线回测，'m'表示日内策略使用分钟线回测
refresh_rate = 1   # 调仓频率，表示执行handle_data的时间间隔，若freq =
 'd'时间间隔的单位为交易日，若freq = 'm'时间间隔为分钟



def initialize(account):                   # 初始化虚拟账户状态
    pass

def handle_data(account):                  # 每个交易日的买入卖出指令


    market_val = DataAPI.MktEqudGet(tradeDate=account.current_da
te,field=u"secID,negMarketValue",pandas="1")     #获取所有股票的市值
    factor = DataAPI.MktStockFactorsOneDayGet(tradeDate=account.
current_date,field='secID,ROE,ROA,CashRateOfSales,FinancialExpen
seRate,CashToCurrentLiability,OperCashInToCurrentLiability,Gross
IncomeRatio,NetProfitRatio,PE,PB',pandas="1")     #获取所有股票的相
关因子
#    print factor
    factor.set_index('secID',inplace=True);
    sec_val_mkt = {'symbol':[], 'factor_value':[], 'market_value'
:[]}
    x='CashToCurrentLiability'
    y='OperCashInToCurrentLiability'
    z='PE'
    for stock in account.universe:
        sec_val_mkt['symbol'].append(stock)
        factor_va=float(1/3*factor.ix[stock][x]+1/3*factor.ix[st
ock][y]+1/3*factor.ix[stock][z]);
        sec_val_mkt['factor_value'].append(factor_va)
        sec_val_mkt['market_value'].append(float(market_val.negMarketValue[market_val.secID==stock]))

    sec_val_mkt = pd.DataFrame(sec_val_mkt).sort(columns='factor_value',ascending=True).reset_index()
    sec_val_mkt = sec_val_mkt[:int(len(sec_val_mkt)*0.1)]
    #排序并选择前10%

    buylist = list(sec_val_mkt.symbol)
    #买入股票列表
    sum_market_val = sum(sec_val_mkt.market_value)
    position = np.array(sec_val_mkt.market_value)/sum_market_val*account.cash
    for stock in account.valid_secpos:
        if stock not in buylist:
            order_to(stock, 0)
    for stock in buylist:
        if stock not in account.valid_secpos:
            order(stock, position[buylist.index(stock)])
    return
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
38.3%	1.0%	36.7%	0.79	0.92	37.7%	1.38	28.3%	0.88

```
bt.blotter
```

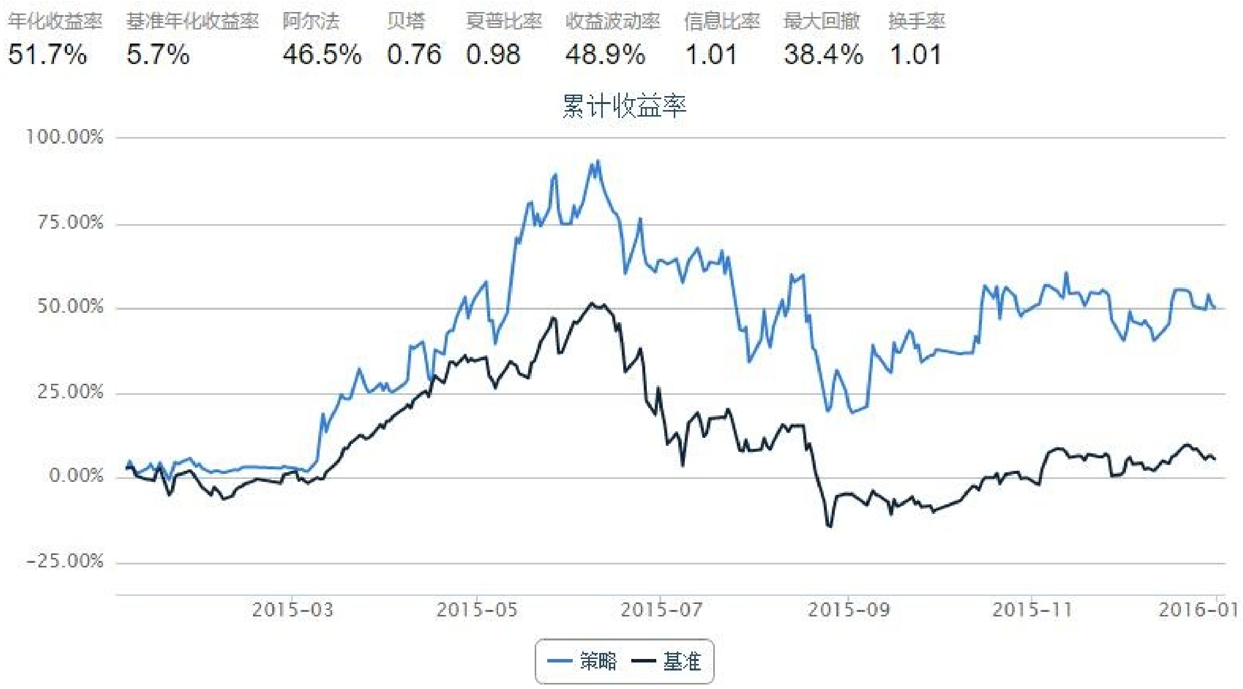
```
None
```

ROE选股指标

> 来源：https://uqer.io/community/share/5668533af9f06c6c8a91b688

简单的ROE选股：按ROE排序选前10%的股票，等权重买入

```
import numpy as np
import pandas as pd
start = '2015-01-01'                       # 回测起始时间
end = '2016-01-01'                         # 回测结束时间
benchmark = 'HS300'                        # 策略参考标准
universe = set_universe('HS300')            # 证券池，支持股票和基金
capital_base = 100000                      # 起始资金
freq = 'd'                             # 策略类型，'d'表示日间策略使
用日线回测，'m'表示日内策略使用分钟线回测
refresh_rate = 20   # 调仓频率，表示执行handle_data的时间间隔，若freq
 = 'd'时间间隔的单位为交易日，若freq = 'm'时间间隔为分钟



def initialize(account):                   # 初始化虚拟账户状态
    pass

def handle_data(account):                  # 每个交易日的买入卖出指令


    factor = DataAPI.MktStockFactorsOneDayGet(secID=account.univ
erse,tradeDate=account.previous_date,field='secID,ROE',pandas="1"
).dropna()    #获取所有股票的相关因子

    sec_val = {'symbol':[], 'factor_value':[]}

    for index, row in factor.iterrows():
        sec_val['symbol'].append(row['secID'])
        sec_val['factor_value'].append(row['ROE'])

    sec_val = pd.DataFrame(sec_val).sort(columns='factor_value')
.reset_index()
    sec_val = sec_val[int(len(sec_val)*0.9):]             #排序并选
择前10%

    buylist = list(sec_val.symbol)             #买入股票列表

    for stock in account.valid_secpos:
        if stock not in buylist:
            order_to(stock, 0)
    for stock in buylist:
        if stock not in account.valid_secpos:
            order(stock, account.cash/len(buylist))
    return
```



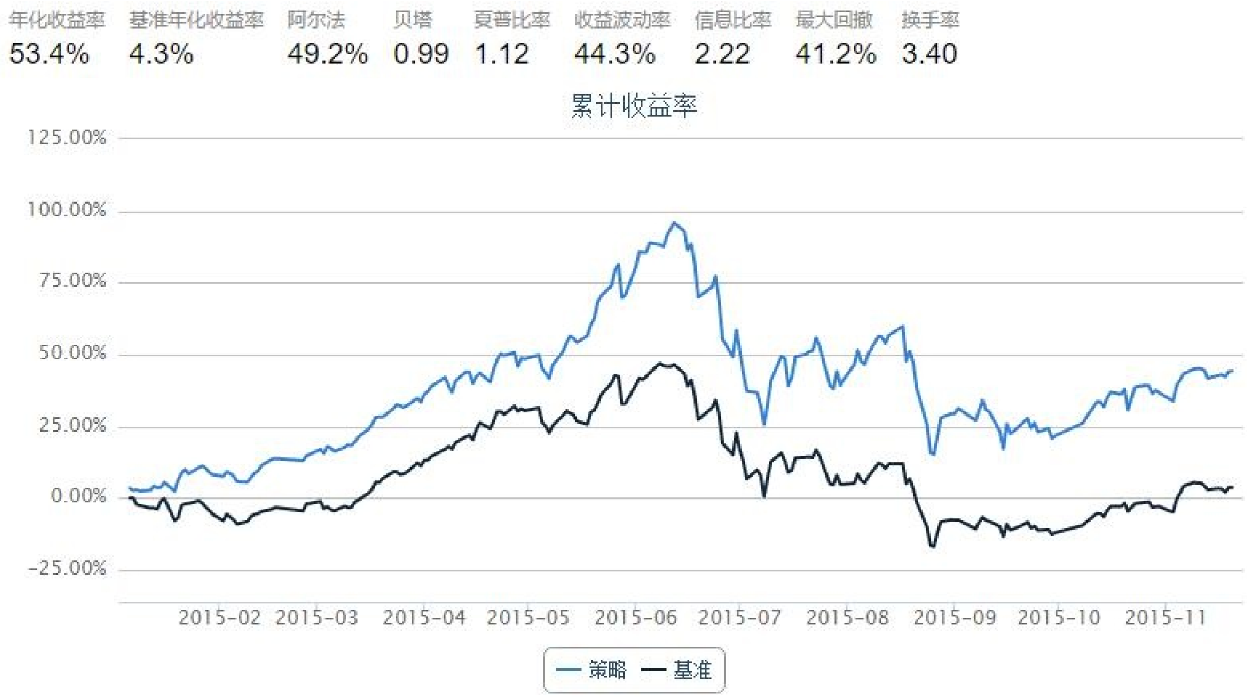
年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
51.7%	5.7%	46.5%	0.76	0.98	48.9%	1.01	38.4%	1.01

成交量因子

> 来源：https://uqer.io/community/share/5652bf20f9f06c4446b48636

投资于沪深300成份股，每月调仓，每月调入成交量最低的成份股

```
start = '2015-1-1'                       #  回测起始时间
end = '2015-11-22'                         #  回测结束时间
benchmark = 'HS300'                        #  策略参考标准
universe = set_universe('HS300')  #  证券池，支持股票和基金
capital_base = 10000000                      #  起始资金
refresh_rate = 20                          #  调仓频率，表示执行handle_data的时间间隔，若freq = 'd'时间间隔的单位为交易日，若freq = 'm'时间间隔为分钟
stk_num = 60


def initialize(account):                   #  初始化虚拟账户状态
    pass

def handle_data(account):                    #  每个交易日的买入卖出指令
    #取出每日成交量
    hist = account.get_attribute_history('turnoverValue',1)
    #print account.current_date
    #print hist

    cjl = {}
    #  排序
    for stock in account.universe:
        cjl[stock] = hist[stock][0]
    dd = sorted(cjl.iteritems(),key = lambda d:d[1],reverse = False)
    #print dd

    cash = account.cash
    #  卖出持有的
    for s,a in account.valid_secpos.items():
        order_to(s,0)
        cash += a*account.referencePrice[s]

    #  买入
    i = 0
    for s in dd:
        if i < stk_num :
            i = i + 1
            order(s[0],cash/60/account.referencePrice[s[0]])
            #print s[0]
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
53.4%	4.3%	49.2%	0.99	1.12	44.3%	2.22	41.2%	3.40

```
bt
```

	tradeDate	cash	security_position	portfolio_value	b
0	2015-01-06	980827.6468	{u'002252.XSHE': {u'amount': 100, u'cost': 22....	1000650.1468	-(
1	2015-01-07	980827.6468	{u'002252.XSHE': {u'amount': 100, u'cost': 22....	1000573.2468	0
2	2015-01-08	980827.6468	{u'002252.XSHE': {u'amount': 100, u'cost': 22....	1000781.2468	-(
3	2015-01-09	980827.6468	{u'002252.XSHE': {u'amount': 100, u'cost': 22....	1000992.8468	-(
4	2015-01-12	980827.6468	{u'002252.XSHE': {u'amount': 100, u'cost': 22....	1001146.3468	-(
5	2015-01-13	980827.6468	{u'002252.XSHE': {u'amount': 100, u'cost': 22....	1001283.9468	0
6	2015-01-14	980827.6468	{u'002252.XSHE': {u'amount': 100,	1001027.0468	-(

	14		u'cost': 22....		
7	2015-01-15	980827.6468	{u'002252.XSHE': {u'amount': 100, u'cost': 22....	1001017.4468	0
8	2015-01-16	980827.6468	{u'002252.XSHE': {u'amount': 100, u'cost': 22....	1001257.6468	0
9	2015-01-19	980827.6468	{u'002252.XSHE': {u'amount': 100, u'cost': 22....	1000788.4468	-0
10	2015-01-20	980827.6468	{u'002252.XSHE': {u'amount': 100, u'cost': 22....	1001703.5468	0
11	2015-01-21	980827.6468	{u'002252.XSHE': {u'amount': 100, u'cost': 22....	1002298.9468	0
12	2015-01-22	980827.6468	{u'002252.XSHE': {u'amount': 100, u'cost': 22....	1002567.5468	0
13	2015-01-23	980827.6468	{u'002252.XSHE': {u'amount': 100, u'cost': 22....	1002212.6468	0
14	2015-01-26	980827.6468	{u'002252.XSHE': {u'amount': 100, u'cost': 22....	1002646.4468	0
15	2015-01-27	980827.6468	{u'002252.XSHE': {u'amount': 100, u'cost': 22....	1003116.3468	-0
16	2015-01-28	980827.6468	{u'002252.XSHE': {u'amount': 100, u'cost': 22....	1002968.9468	-0
17	2015-01-29	980827.6468	{u'002252.XSHE': {u'amount': 100, u'cost': 22....	1002741.4468	-0
18	2015-01-30	980827.6468	{u'002252.XSHE': {u'amount': 100, u'cost': 22....	1002465.6468	-0
19	2015-02-	980827.6468	{u'002252.XSHE': {u'amount': 100,	1002310.2468	-0

20	2015-02-03	979846.9505	{u'601158.XSHG': {u'amount': 400, u'cost': 7.9...	1002687.7505	0
21	2015-02-04	979846.9505	{u'601158.XSHG': {u'amount': 400, u'cost': 7.9...	1002527.6505	-(
22	2015-02-05	979846.9505	{u'601158.XSHG': {u'amount': 400, u'cost': 7.9...	1002320.0505	-(
23	2015-02-06	979846.9505	{u'601158.XSHG': {u'amount': 400, u'cost': 7.9...	1001810.1505	-(
24	2015-02-09	979846.9505	{u'601158.XSHG': {u'amount': 400, u'cost': 7.9...	1002149.1505	0
25	2015-02-10	979846.9505	{u'601158.XSHG': {u'amount': 400, u'cost': 7.9...	1002526.5505	0
26	2015-02-11	979846.9505	{u'601158.XSHG': {u'amount': 400, u'cost': 7.9...	1002846.5505	0
27	2015-02-12	979846.9505	{u'601158.XSHG': {u'amount': 400, u'cost': 7.9...	1003098.6505	0
28	2015-02-13	979846.9505	{u'601158.XSHG': {u'amount': 400, u'cost': 7.9...	1003569.5505	0
29	2015-02-16	979846.9505	{u'601158.XSHG': {u'amount': 400, u'cost': 7.9...	1004382.4505	0
...	...	...	...	...	..
184	2015-10-12	979810.0847	{u'601158.XSHG': {u'amount': 400, u'cost': 8.2...	1011749.0847	0
185	2015-10-13	979810.0847	{u'601158.XSHG': {u'amount': 400, u'cost': 8.2...	1011658.0847	-(
186	2015-10-14	979810.0847	{u'601158.XSHG': {u'amount': 400,	1011279.0847	-(

186	2015-10-14	979810.0847	{u'amount': 400, u'cost': 8.2...	1011279.0847	-0
187	2015-10-15	979810.0847	{u'601158.XSHG': {u'amount': 400, u'cost': 8.2...	1011943.0847	0
188	2015-10-16	979810.0847	{u'601158.XSHG': {u'amount': 400, u'cost': 8.2...	1012667.0847	0
189	2015-10-19	979810.0847	{u'601158.XSHG': {u'amount': 400, u'cost': 8.2...	1012613.0847	0
190	2015-10-20	979810.0847	{u'601158.XSHG': {u'amount': 400, u'cost': 8.2...	1012848.0847	0
191	2015-10-21	979810.0847	{u'601158.XSHG': {u'amount': 400, u'cost': 8.2...	1011284.0847	-0
192	2015-10-22	979810.0847	{u'601158.XSHG': {u'amount': 400, u'cost': 8.2...	1012610.0847	0
193	2015-10-23	979810.0847	{u'601158.XSHG': {u'amount': 400, u'cost': 8.2...	1013730.0847	0
194	2015-10-26	979810.0847	{u'601158.XSHG': {u'amount': 400, u'cost': 8.2...	1013604.0847	0
195	2015-10-27	979810.0847	{u'601158.XSHG': {u'amount': 400, u'cost': 8.2...	1013456.0847	0
196	2015-10-28	979810.0847	{u'601158.XSHG': {u'amount': 400, u'cost': 8.2...	1012937.0847	-0
197	2015-10-29	979810.0847	{u'601158.XSHG': {u'amount': 400, u'cost': 8.2...	1013093.0847	0
198	2015-10-30	979810.0847	{u'601158.XSHG': {u'amount': 400, u'cost': 8.2...	1012739.0847	0
	2015-11-		{u'601158.XSHG':		

200	2015-11-03	984251.4817	{u'600011.XSHG': {u'amount': 400, u'cost': 9.0...	1012266.4817	-0
201	2015-11-04	984251.4817	{u'600011.XSHG': {u'amount': 400, u'cost': 9.0...	1013422.4817	0
202	2015-11-05	984251.4817	{u'600011.XSHG': {u'amount': 400, u'cost': 9.0...	1014069.4817	0
203	2015-11-06	984251.4817	{u'600011.XSHG': {u'amount': 400, u'cost': 9.0...	1014599.4817	0
204	2015-11-09	984251.4817	{u'600011.XSHG': {u'amount': 400, u'cost': 9.0...	1015194.4817	0
205	2015-11-10	984251.4817	{u'600011.XSHG': {u'amount': 400, u'cost': 9.0...	1015158.4817	-0
206	2015-11-11	984251.4817	{u'600011.XSHG': {u'amount': 400, u'cost': 9.0...	1015091.4817	0
207	2015-11-12	984251.4817	{u'600011.XSHG': {u'amount': 400, u'cost': 9.0...	1014932.4817	-0
208	2015-11-13	984251.4817	{u'600011.XSHG': {u'amount': 400, u'cost': 9.0...	1014318.4817	-0
209	2015-11-16	984251.4817	{u'600011.XSHG': {u'amount': 400, u'cost': 9.0...	1014373.4817	0
210	2015-11-17	984251.4817	{u'600011.XSHG': {u'amount': 400, u'cost': 9.0...	1014520.4817	-0
211	2015-11-18	984251.4817	{u'600011.XSHG': {u'amount': 400, u'cost': 9.0...	1014182.4817	-0
212	2015-11-19	984251.4817	{u'600011.XSHG': {u'amount': 400, u'cost': 9.0...	1014563.4817	0

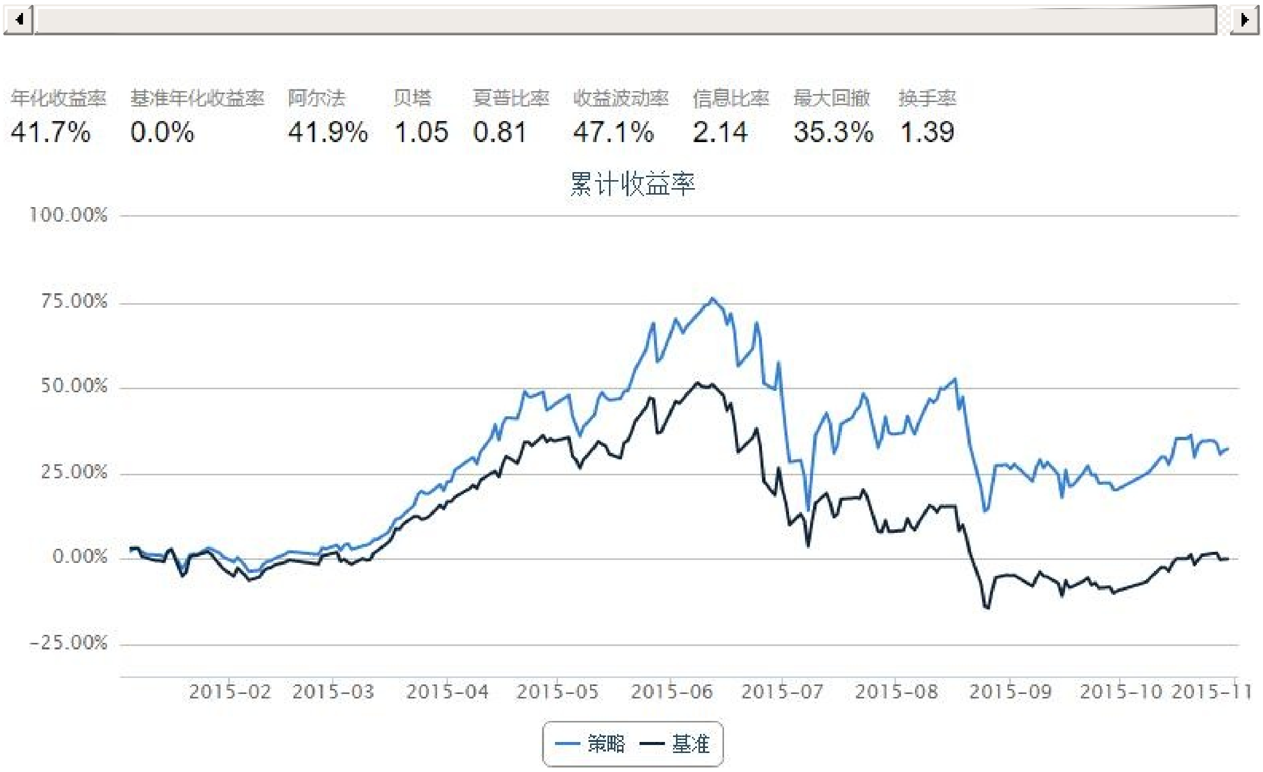
	19		u'cost': 9.0...		
213	2015-11-20	984251.4817	{u'600011.XSHG': {u'amount': 400, u'cost': 9.0...	1014545.4817	-(

```
214 rows × 6 columns
```

ROIC&cashROIC

> 来源：https://uqer.io/community/share/564d30eff9f06c4446b483db

```
from CAL.PyCAL import *
import numpy as np
from pandas import DataFrame , Series

start = '2015-01-01'                       # 回测起始时间
end = '2015-11-01'                         # 回测结束时间
benchmark = 'HS300'                        # 策略参考标准
universe = set_universe('HS300')           # 证券池，支持股票和基金
capital_base = 100000                      # 起始资金
freq = 'd'                                 # 策略类型，'d'表示日间
策略使用日线回测，'m'表示日内策略使用分钟线回测
refresh_rate = 30                          # 调仓频率，表示执行hand
le_data的时间间隔，若freq = 'd'时间间隔的单位为交易日，若freq = 'm'时间
间隔为分钟

cal = Calendar('China.SSE')

def initialize(account):                   # 初始化虚拟账户状态
    pass

def handle_data(account):                  # 每个交易日的买入卖出指令


    today = account.current_date.strftime('%Y%m%d')
    yesterday = cal.advanceDate(account.current_date, '-1B', Biz
DayConvention.Following).strftime('%Y%m%d')

    ROIC = {}
    cashROIC = {}

    # EBITToTOR
    EBITToTOR = DataAPI.MktStockFactorsOneDayGet(tradeDate=yeste
rday,secID=account.universe,field=u"secID,EBITToTOR",pandas="1")
    EBITToTOR.drop_duplicates('secID' , inplace= True)
    EBITToTOR.set_index('secID' , inplace = True)

    # tRevenue : 营业收入
    tRevenue = DataAPI.FdmtISAllLatestGet(secID=account.universe
,endDate=u"20151101",beginDate=u"20150101",field=u"secID,tRevenu
e",pandas="1")
    tRevenue.drop_duplicates('secID' , inplace= True)
    tRevenue.set_index('secID' , inplace = True)

    # 自由现金流
    freeCF = DataAPI.FdmtCFAllLatestGet(secID=account.universe,e
ndDate=u"20151101",beginDate=u"20150101",field=u"secID,NCFOperateA,purFixAssetsOth,dispFixAssetsOth",pandas="1")
    freeCF.drop_duplicates('secID' , inplace= True)
    freeCF.set_index('secID' , inplace= True)

    # IC : 投入资本
    IC = DataAPI.FdmtBSAllLatestGet(secID=account.universe,endDate=u"20151101",beginDate=u"20150101",field=u"secID,TShEquity,LTBorr",pandas="1")
    IC.drop_duplicates('secID' , inplace= True)
    IC.set_index('secID' , inplace = True)

    for s in account.universe:
        ROIC[s] = (EBITToTOR['EBITToTOR'][s] * tRevenue['tRevenue'][s] * (1-0.25)) / (IC['TShEquity'][s] + IC['LTBorr'][s])
        cashROIC[s] = (freeCF['NCFOperateA'][s] - (freeCF['purFixAssetsOth'][s] - freeCF['dispFixAssetsOth'][s])) / (IC['TShEquity'][s] + IC['LTBorr'][s])

    # ROIC
    ROIC = Series(ROIC)
    ROIC.sort(ascending = False)
    ROIC.dropna(inplace = True)
    ROIC = ROIC[0:60]

    buylist1 = list(ROIC.index)

    # cashROIC
    cashROIC = Series(cashROIC)
    cashROIC.sort(ascending = False)
    cashROIC.dropna(inplace = True)
    cashROIC = cashROIC[0:60]

    buylist2 = list(cashROIC.index)

    # buylist为buylist1与buylist2的交集
    buylist = list(set(buylist1)&set(buylist2))
    print 'buylist', len(buylist)
    print buylist

    sell_list = [x for x in account.valid_secpos if x not in buylist]
    for s in sell_list :
        order_to(s,0)

    total_money = account.referencePortfolioValue
    for s in buylist:
        if s in account.valid_secpos:
            pass
        else:
            order(s,total_money/(len(buylist)-len(set(buylist)&set(account.valid_secpos)))/account.referencePrice[s])
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
41.7%	0.0%	41.9%	1.05	0.81	47.1%	2.14	35.3%	1.39

```
buylist 24
['601601.XSHG', '601899.XSHG', '000895.XSHE', '000539.XSHE', '60
0276.XSHG', '600011.XSHG', '600887.XSHG', '000712.XSHE', '600863
.XSHG', '600018.XSHG', '000538.XSHE', '600900.XSHG', '002470.XSH
E', '600600.XSHG', '600153.XSHG', '002304.XSHE', '600104.XSHG',
'000333.XSHE', '600873.XSHG', '600578.XSHG', '000999.XSHE', '002
294.XSHE', '601006.XSHG', '600177.XSHG']
buylist 24
['600027.XSHG', '000559.XSHE', '601899.XSHG', '000895.XSHE', '00
0876.XSHE', '000539.XSHE', '600276.XSHG', '600011.XSHG', '601006
.XSHG', '000538.XSHE', '600900.XSHG', '002470.XSHE', '600600.XSH
G', '600153.XSHG', '600332.XSHG', '002304.XSHE', '600104.XSHG',
'000333.XSHE', '600873.XSHG', '600578.XSHG', '000999.XSHE', '002
294.XSHE', '000712.XSHE', '600177.XSHG']
buylist 21
['600276.XSHG', '600900.XSHG', '600104.XSHG', '600011.XSHG', '60
0887.XSHG', '600873.XSHG', '600027.XSHG', '601006.XSHG', '600663
.XSHG', '600018.XSHG', '000538.XSHE', '601899.XSHG', '600863.XSH
G', '002470.XSHE', '002304.XSHE', '000999.XSHE', '000895.XSHE',
'000876.XSHE', '600153.XSHG', '600177.XSHG', '000712.XSHE']
buylist 22
['601601.XSHG', '600027.XSHG', '000876.XSHE', '600276.XSHG', '60
0011.XSHG', '600887.XSHG', '600863.XSHG', '000538.XSHE', '002470
.XSHE', '600332.XSHG', '002304.XSHE', '601377.XSHG', '600104.XSH
G', '000333.XSHE', '600873.XSHG', '601628.XSHG', '002294.XSHE',
'600663.XSHG', '000712.XSHE', '600886.XSHG', '601318.XSHG', '600
177.XSHG']
buylist 22
['601216.XSHG', '600027.XSHG', '000895.XSHE', '000876.XSHE', '60
0011.XSHG', '600887.XSHG', '000538.XSHE', '002470.XSHE', '600649
.XSHG', '600332.XSHG', '002304.XSHE', '601377.XSHG', '600104.XSH
G', '000333.XSHE', '600873.XSHG', '600578.XSHG', '600741.XSHG',
'601628.XSHG', '000999.XSHE', '600795.XSHG', '600663.XSHG', '601
318.XSHG']
buylist 26
['601216.XSHG', '600340.XSHG', '000876.XSHE', '600276.XSHG', '60
0011.XSHG', '600887.XSHG', '601006.XSHG', '600018.XSHG', '000538
.XSHE', '002470.XSHE', '000069.XSHE', '600332.XSHG', '002304.XSH
E', '601377.XSHG', '600104.XSHG', '000333.XSHE', '600578.XSHG',
'601628.XSHG', '000999.XSHE', '002294.XSHE', '600795.XSHG', '600
663.XSHG', '600863.XSHG', '600060.XSHG', '601318.XSHG', '600177.
XSHG']
buylist 28
['002304.XSHE', '601216.XSHG', '600027.XSHG', '601111.XSHG', '60
1899.XSHG', '000878.XSHE', '600018.XSHG', '000895.XSHE', '600276
.XSHG', '600887.XSHG', '601006.XSHG', '601601.XSHG', '601088.XSH
G', '002470.XSHE', '000069.XSHE', '600332.XSHG', '600597.XSHG',
'601377.XSHG', '600873.XSHG', '600578.XSHG', '600649.XSHG', '601
628.XSHG', '002294.XSHE', '600795.XSHG', '600663.XSHG', '600863.
XSHG', '601318.XSHG', '600177.XSHG']
```

【国信金工】资产周转率选股模型

> 来源：https://uqer.io/community/share/565012a0f9f06c4446b48545

资产周转率选股模型

一、背景介绍

根据研究报告20151113-国信证券-国信alpha选股系列：资产周转率选股模型构建。

二、周转率选股模型

这里，研究报告中使用周转率以及相关基本面因子去预测公司 ROE 的改善，实证检验表明， ROE 的改善大概率会随着未来半年业绩的兑现而传导到股票收益上。综合考虑，研究报告用以下方式构建周转率选股模型：

1. 总资产周转率同比改善；
2. 营业收入同比增长；
3. ROE 在全市场处于较低水平；
4. 营业利润同比增长。

一年调仓两次，分别在每年 5 月初、 11 月初调仓。每期选股数量在 50 至 140只之间。平均每期股票 80 只左右。

使用“业绩快报”与“合并利润表”API或缺营业收入与营业利润只有年报，1月初发行的。所以5月与11月调仓使用的是同意的数据。

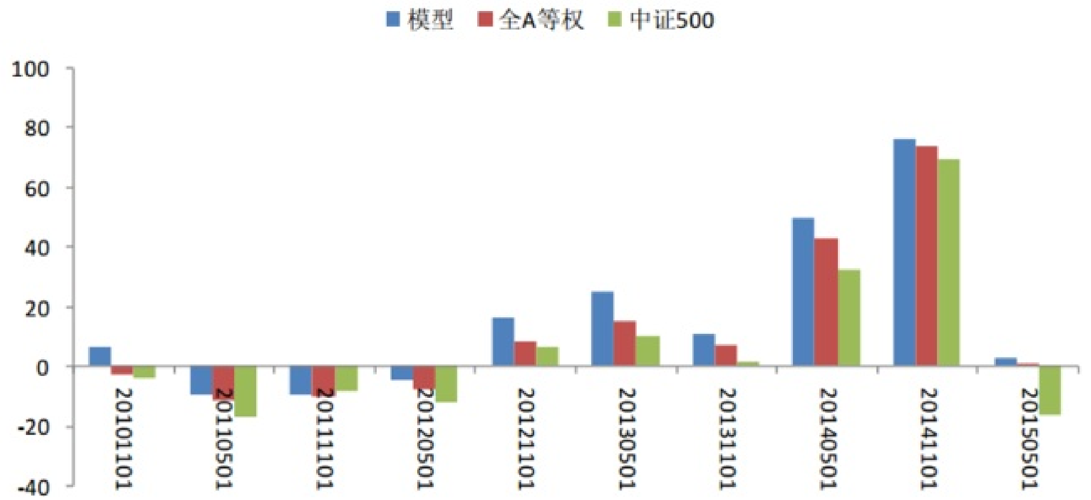
所以本策略对原模型稍加改变，使用了‘营业利润增长率’、‘营业收入增长率’代替‘营业利润’与‘营业收入’。

PS:其实是懒 - - ！ 没找其他的API，坐等大神实现原策略！^_^

三、研究报告回测结果

模型回测结果如下图所示:

图 2： 2011 年至 2015 年 10 月，周转率选股模型每期的绝对收益

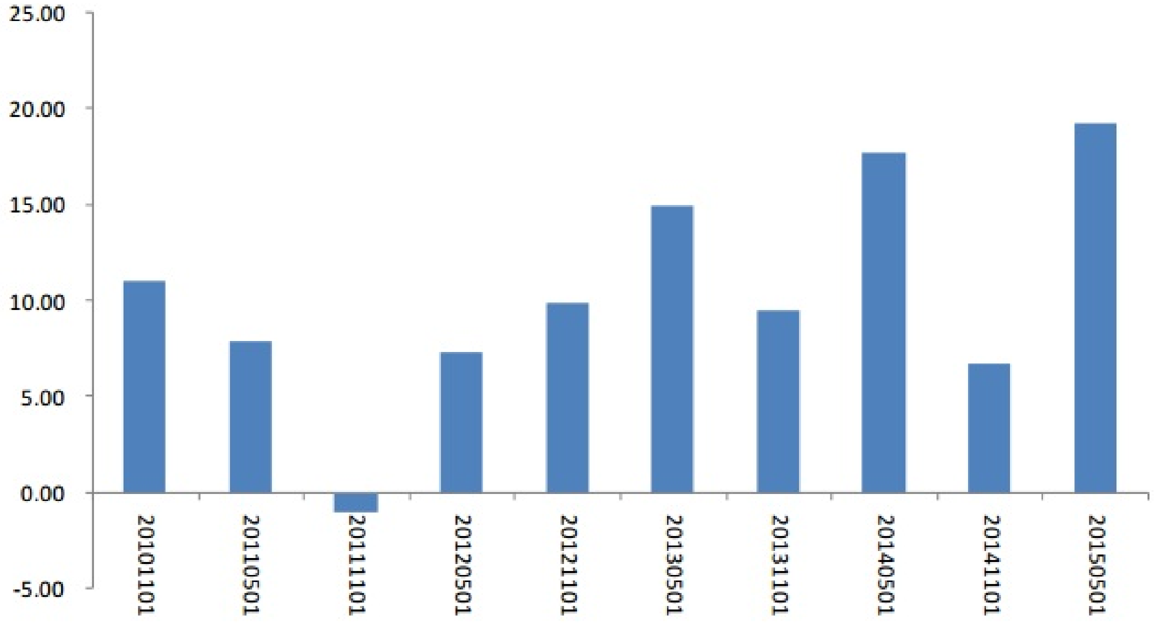


资料来源：WIND，国信证券经济研究所整理

上图显示，从绝对收益角度来看，周转率选股模型在 2011 年 11 月至 2013 年 4月这三期当中绝对收益为负；2011 年 5~10 月，虽然整体市场下跌，但模型组合绝对收益为正；2015 年 5~10 月，市场先扬后抑，但模型组合仍斩获正收益。

在 2012 年 11 月至 2015 年 4 月这五期当中，市场经历了一波牛市，模型组合不但收益为正，而且每期都跑赢了全 A 等权和中证 500。最近 10 期中有 7 期绝对收益为正，而全 A 等权 10 期中有 6 期收益为正，中证 500 在近 10 期中仅有 5 期收益为正。

图 4： 2011 年至 2015 年 10 月，周转率选股组合每期相对中证 500 指数的超额收益

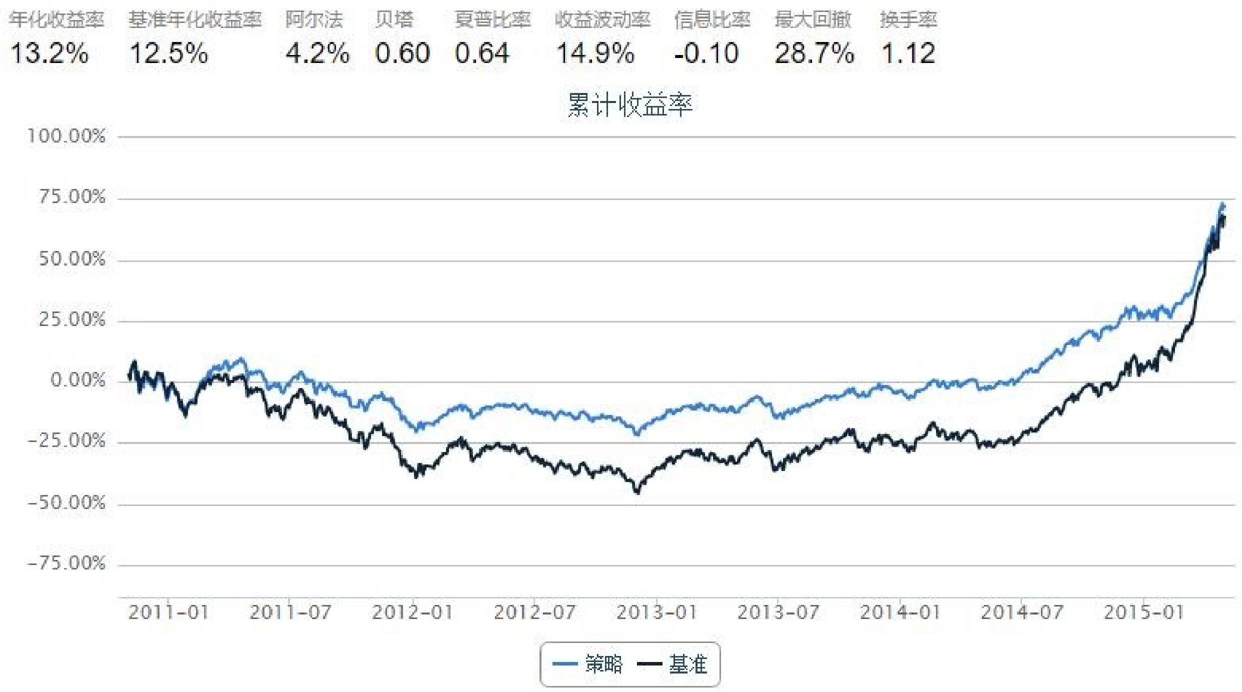


资料来源：WIND，国信证券经济研究所整理

在最近 10 期当中，除 2011 年报期之外，模型组合有 9 期跑赢中证 500。 其中，在 2011 年 11 月年报期，模型的相对收益最低，为-1.03%；在 2015 年 5 月中报期，模型的相对收益最高，有 19.30%。平均年化超额收益 20.68%。

从绝对收益角度考虑，由于全 A 等权并无实际对冲标的，而且模型相对中证 500的超额收益较高，所以该模型在中证 500 指数的基础上做绝对收益不失为一个不错的选择。

```
from CAL.PyCAL import *

start = '2010-11-01'                       # 回测起始时间
end = '2015-05-01'                         # 回测结束时间
benchmark = 'ZZ500'                        # 策略参考标准
universe = set_universe('ZZ500')# + set_universe('HS300')  # 证券池，支持股票和基金
capital_base = 10000000                     # 起始资金
freq = 'd'                                 # 策略类型，'d'表示日间策略使用日线回测，'m'表示日内策略使用分钟线回测
refresh_rate = 122                          # 调仓频率，表示执行handle_data的时间间隔，若freq = 'd'时间间隔的单位为交易日，若freq = 'm'时间间隔为分钟
cal = Calendar('China.SSE')
stocknum = 80

def initialize(account):                   # 初始化虚拟账户状态
    pass

def handle_data(account):                  # 每个交易日的买入卖出指令

    global stocknum

    # 获取时间
    today = Date.fromDateTime(account.current_date).strftime('%Y%m%d')     # 当天日期
    lasttime = cal.advanceDate(today,'-1Y',BizDayConvention.Preceding).strftime('%Y%m%d')
    print '调仓日期:' + str(account.current_date)

    # 根据条件获取buylist
    tf1 = DataAPI.MktStockFactorsOneDayGet(tradeDate=today,secID=account.universe,ticker=u"",field=['secID','OperatingProfitGrowRate','OperatingRevenueGrowRate','TotalAssetsTRate','ROE'],pandas="1")
    tf1.set_index('secID',inplace=True)
    tf1 = tf1[(tf1.OperatingProfitGrowRate>0.0)&(tf1.OperatingRevenueGrowRate>0.0)&(tf1.TotalAssetsTRate>0.0)&(tf1.ROE>0.0)].dropna()

    tf2 = DataAPI.MktStockFactorsOneDayGet(tradeDate=lasttime,secID=account.universe,ticker=u"",field=['secID','OperatingProfitGrowRate','OperatingRevenueGrowRate','TotalAssetsTRate','ROE'],pandas="1")
    tf2.set_index('secID',inplace=True)
    tf2 = tf2[(tf2.OperatingProfitGrowRate>0.0)&(tf2.OperatingRevenueGrowRate>0.0)&(tf2.TotalAssetsTRate>0.0)&(tf2.ROE>0.0)].dropna()

    per_buylist=[]
    for stock in list(tf1.index):
        if stock in list(tf2.index):
            if((tf1['OperatingProfitGrowRate'][stock]>tf2['OperatingProfitGrowRate'][stock])&(tf1['OperatingRevenueGrowRate'][stock]>tf2['OperatingRevenueGrowRate'][stock])&(tf1['TotalAssetsTRate'][stock]>tf2['TotalAssetsTRate'][stock])):
                per_buylist.append(stock)

    by = DataAPI.MktStockFactorsOneDayGet(tradeDate=today,secID=account.universe,ticker=u"",field=['secID','ROE'],pandas="1")
    by.set_index('secID',inplace=True)
    by = by[by.ROE>=0.0].dropna().sort(columns='ROE',ascending=True)
    by = by.head(80)
    buylist = list(by.index)

    # 卖出股票
    for stock in account.valid_secpos:
        if stock not in buylist:
            order_to(stock, 0)
        else:
            pass

    # 买入股票
    for stock in buylist:
        if stock not in account.valid_secpos:
            order(stock, account.cash/account.referencePrice[stock]/stocknum)
    # print list(account.valid_secpos.keys())
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
13.2%	12.5%	4.2%	0.60	0.64	14.9%	-0.10	28.7%	1.12

```
调仓日期:2010-11-01 00:00:00
调仓日期:2011-05-03 00:00:00
调仓日期:2011-10-31 00:00:00
调仓日期:2012-05-04 00:00:00
调仓日期:2012-10-31 00:00:00
调仓日期:2013-05-08 00:00:00
调仓日期:2013-11-08 00:00:00
调仓日期:2014-05-12 00:00:00
调仓日期:2014-11-07 00:00:00
```

【基本面指标】Cash Cow

> 来源：https://uqer.io/community/share/55418287f9f06c1c3d687fde

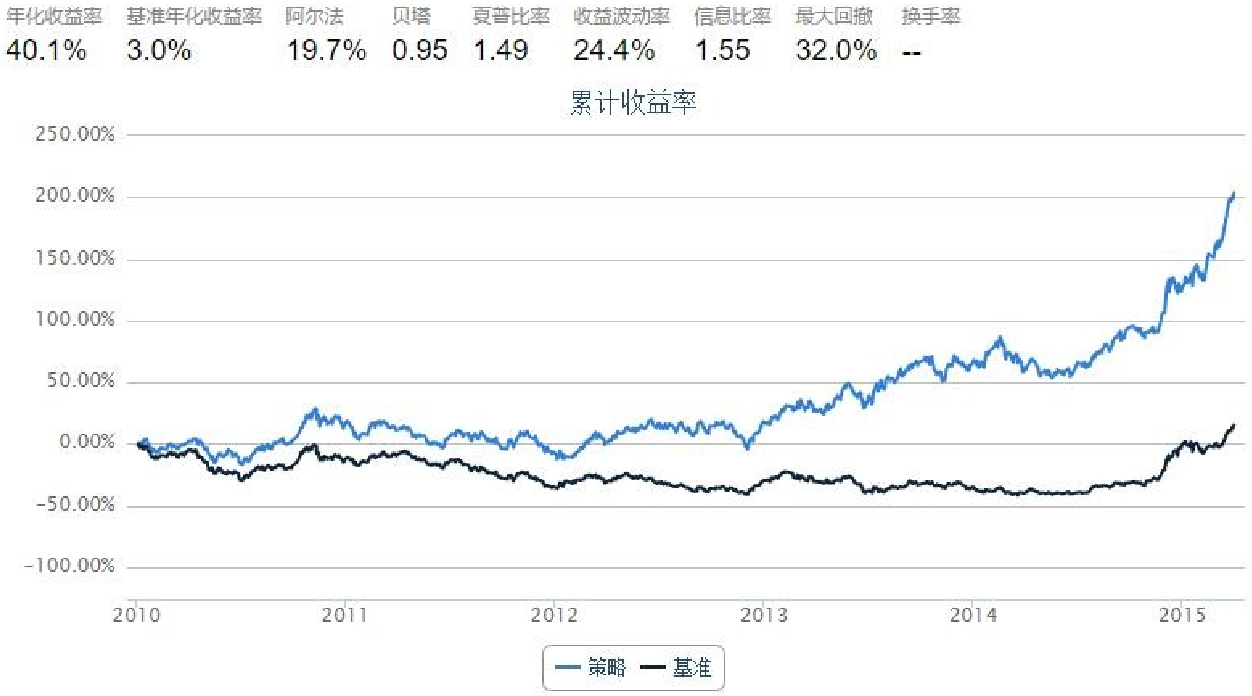
策略思路

每个季度，计算沪深300成分股资产负债表中的 现金及现金等价物/总资产 ，数值以最近一次披露的财报为准

清仓，选出该比率最大的前30只股票，将资金分成30份，分别买入

每60个交易日调仓一次

```
from heapq import nlargest
from datetime import timedelta

start = '2010-01-01'
end   = '2015-04-01'
benchmark = 'HS300'
universe = set_universe('HS300')
capital_base = 500000
refresh_rate = 60

def initialize(account):
    pass

def handle_data(account):
    cashpct = getCashPct(account.universe, account.current_date)
    buylist = nlargest(30, cashpct, key=cashpct.get)

    for stock in account.valid_secpos:
        order_to(stock, 0)

    for stock in buylist:
        order(stock, int(account.referencePortfolioValue/len(buy
list)/account.referencePrice[stock]/100)*100)

def getCashPct(universe, date):
    start, end = (date - timedelta(weeks=26)).strftime('%Y%m%d')
, date.strftime('%Y%m%d')
    N = len(universe)
    if N == 0:
        return None
    elif N <= 45:
        batches = [universe]
    else:
        batches = [universe[i:i+45] for i in range(0, N, 45)]

    CashPct = {}
    for sub in batches:
        df = DataAPI.FdmtBSGet(secID=','.join(sub), publishDateB
egin=start, publishDateEnd=end, field=['secID', 'cashCEquiv', 'T
Assets'])
        for stock in sub:
            try:
                df_sub = df[df.secID==stock]
                df_sub['pct'] = df_sub['cashCEquiv'] / df_sub['T
Assets']
                CashPct[stock] = df_sub['pct'].mean()
            except:
                pass
    return CashPct
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
40.1%	3.0%	19.7%	0.95	1.49	24.4%	1.55	32.0%	--

量化因子选股——净利润／营业总收入

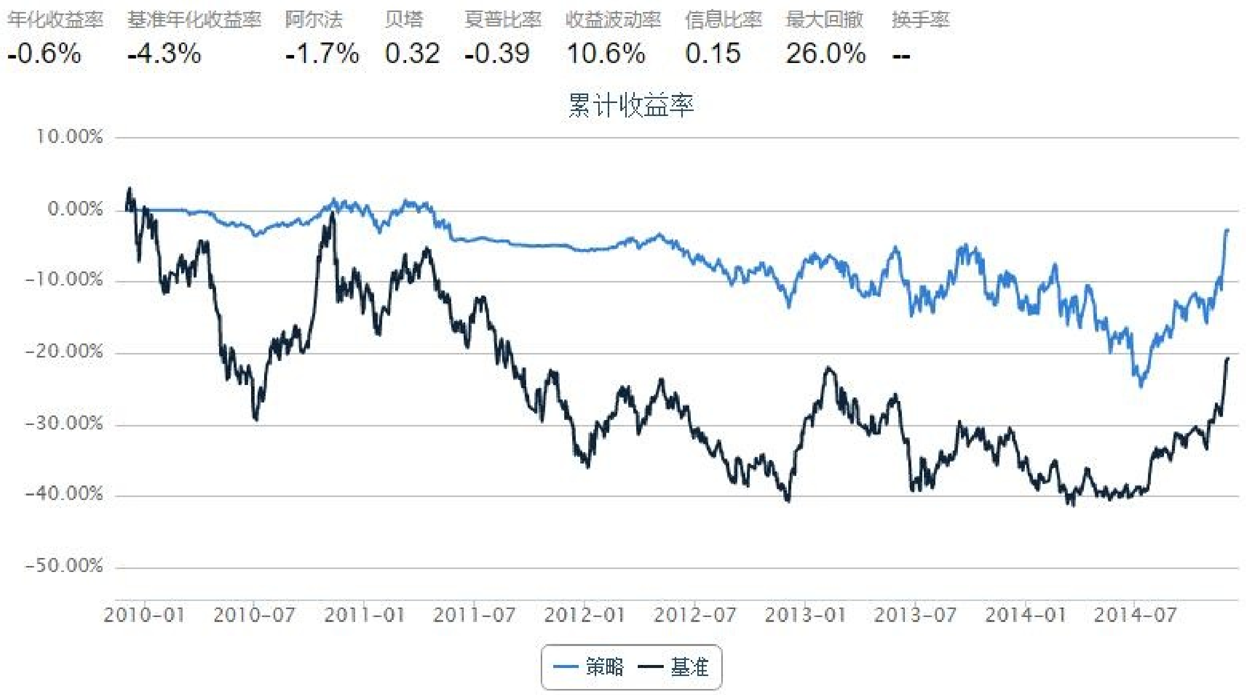
> 来源：https://uqer.io/community/share/548aac7af9f06c31c3950caf

量化因子选股：净利润与营业总收入之比

策略实现

按季度调仓，若某股票对应因子值在均值以上，则买入，反之卖出。

```
from CAL.PyCAL import *
from numpy import *

start = datetime(2009, 12, 1)
end   = datetime(2014, 12, 1)
benchmark = 'HS300'
universe = set_universe('SH180')
capital_base = 1e6
refresh_rate = 60


tickers = universe[:]
for i in range(len(universe)):
    tickers[i] = universe[i][0:6]

def initialize(account):
    pass

def handle_data(account, data):
    today = account.current_date
    today_str = today.strftime("%Y%m%d")
    print today_str

    factor = DataAPI.MktStockFactorsOneDayGet(stockID=tickers, d
ate=today_str, field='NPToTOR', pandas='1')
    if(len(factor) == 0):
        return
    factor = factor.dropna()
    cutoff = np.mean(factor['NPToTOR'])

    for stock in universe:
        stk = stock[0:6]
        try:
            val = factor[factor.ticker == stk]['NPToTOR'].iloc[0
]
        except:
            return

        if val > cutoff:
            order(stock, 100)
        else:
            order_to(stock, 0)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
-0.6%	-4.3%	-1.7%	0.32	-0.39	10.6%	0.15	26.0%	--

```
20100303
/market/getStockFactorsOneDay.csv?stockID=600000,600008,600009,6
00010,600011,600015,600016,600018,600019,600027,600028,600029,60
0030,600031,600036,600048,600050,600058,600060,600064,600066,600
067,600085,600089,600094,600100,600104,600108,600109,600111,6001
18,600123,600141,600143,600150,600157,600158,600160,600162,60016
6,600170,600177,600188,600196,600199,600208,600216,600221,600239
,600240,600252,600256,600259,600266,600267,600271,600276,600300,
600309,600315,600316,600325,600332,600340,600348,600352,600362,6
00366,600369,600372,600376,600383,600395,600406,600415,600418,60
0489,600497,600498,600503,600516,600518,600519,600535,600546,600
547,600549,600583,600585,600588,600597,600598,600600,600633,6006
37,600639,600643,600649,600663,600674,600675,600684,600690,60070
2,600703,600705,600736,600739,600741,600747,600748,600759,600765
,600770,600773,600795,600804,600809,600816,600823,600832,600837,
600875,600880,600886,600887,600893,600895,600900,600970,600999,6
01006,601009,601088,601099,601111,601117,601118,601166,601168,60
1169,601186,601216,601288,601299,601318,601328,601336,601377,601
390,601398,601555,601588,601600,601601,601607,601628,601633,6016
68,601669,601688,601699,601717,601766,601800,601808,601818,60185
7,601888,601899,601901,601928,601939,601958,601988,601989,601992
,601998,603000,603993&date=20100303&field=NPToTOR
20100528
/market/getStockFactorsOneDay.csv?stockID=600000,600008,600009,6
00010,600011,600015,600016,600018,600019,600027,600028,600029,60
0030,600031,600036,600048,600050,600058,600060,600064,600066,600
067,600085,600089,600094,600100,600104,600108,600109,600111,6001
18,600123,600141,600143,600150,600157,600158,600160,600162,60016
6,600170,600177,600188,600196,600199,600208,600216,600221,600239
,600240,600252,600256,600259,600266,600267,600271,600276,600300,
600309,600315,600316,600325,600332,600340,600348,600352,600362,6
00366,600369,600372,600376,600383,600395,600406,600415,600418,60
```

```
0489,600497,600498,600503,600516,600518,600519,600535,600546,600
547,600549,600583,600585,600588,600597,600598,600600,600633,6006
37,600639,600643,600649,600663,600674,600675,600684,600690,60070
2,600703,600705,600736,600739,600741,600747,600748,600759,600765
,600770,600773,600795,600804,600809,600816,600823,600832,600837,
600875,600880,600886,600887,600893,600895,600900,600970,600999,6
01006,601009,601088,601099,601111,601117,601118,601166,601168,60
1169,601186,601216,601288,601299,601318,601328,601336,601377,601
390,601398,601555,601588,601600,601601,601607,601628,601633,6016
68,601669,601688,601699,601717,601766,601800,601808,601818,60185
7,601888,601899,601901,601928,601939,601958,601988,601989,601992
,601998,603000,603993&date=20100528&field=NPToTOR
20100825
/market/getStockFactorsOneDay.csv?stockID=600000,600008,600009,6
00010,600011,600015,600016,600018,600019,600027,600028,600029,60
0030,600031,600036,600048,600050,600058,600060,600064,600066,600
067,600085,600089,600094,600100,600104,600108,600109,600111,6001
18,600123,600141,600143,600150,600157,600158,600160,600162,60016
6,600170,600177,600188,600196,600199,600208,600216,600221,600239
,600240,600252,600256,600259,600266,600267,600271,600276,600300,
600309,600315,600316,600325,600332,600340,600348,600352,600362,6
00366,600369,600372,600376,600383,600395,600406,600415,600418,60
0489,600497,600498,600503,600516,600518,600519,600535,600546,600
547,600549,600583,600585,600588,600597,600598,600600,600633,6006
37,600639,600643,600649,600663,600674,600675,600684,600690,60070
2,600703,600705,600736,600739,600741,600747,600748,600759,600765
,600770,600773,600795,600804,600809,600816,600823,600832,600837,
600875,600880,600886,600887,600893,600895,600900,600970,600999,6
01006,601009,601088,601099,601111,601117,601118,601166,601168,60
1169,601186,601216,601288,601299,601318,601328,601336,601377,601
390,601398,601555,601588,601600,601601,601607,601628,601633,6016
68,601669,601688,601699,601717,601766,601800,601808,601818,60185
7,601888,601899,601901,601928,601939,601958,601988,601989,601992
,601998,603000,603993&date=20100825&field=NPToTOR
20101129
/market/getStockFactorsOneDay.csv?stockID=600000,600008,600009,6
00010,600011,600015,600016,600018,600019,600027,600028,600029,60
0030,600031,600036,600048,600050,600058,600060,600064,600066,600
067,600085,600089,600094,600100,600104,600108,600109,600111,6001
18,600123,600141,600143,600150,600157,600158,600160,600162,60016
6,600170,600177,600188,600196,600199,600208,600216,600221,600239
,600240,600252,600256,600259,600266,600267,600271,600276,600300,
600309,600315,600316,600325,600332,600340,600348,600352,600362,6
00366,600369,600372,600376,600383,600395,600406,600415,600418,60
0489,600497,600498,600503,600516,600518,600519,600535,600546,600
547,600549,600583,600585,600588,600597,600598,600600,600633,6006
37,600639,600643,600649,600663,600674,600675,600684,600690,60070
2,600703,600705,600736,600739,600741,600747,600748,600759,600765
,600770,600773,600795,600804,600809,600816,600823,600832,600837,
600875,600880,600886,600887,600893,600895,600900,600970,600999,6
01006,601009,601088,601099,601111,601117,601118,601166,601168,60
1169,601186,601216,601288,601299,601318,601328,601336,601377,601
390,601398,601555,601588,601600,601601,601607,601628,601633,6016
```

```
68,601669,601688,601699,601717,601766,601800,601808,601818,60185
7,601888,601899,601901,601928,601939,601958,601988,601989,601992
,601998,603000,603993&date=20101129&field=NPToTOR
20110301
/market/getStockFactorsOneDay.csv?stockID=600000,600008,600009,6
00010,600011,600015,600016,600018,600019,600027,600028,600029,60
0030,600031,600036,600048,600050,600058,600060,600064,600066,600
067,600085,600089,600094,600100,600104,600108,600109,600111,6001
18,600123,600141,600143,600150,600157,600158,600160,600162,60016
6,600170,600177,600188,600196,600199,600208,600216,600221,600239
,600240,600252,600256,600259,600266,600267,600271,600276,600300,
600309,600315,600316,600325,600332,600340,600348,600352,600362,6
00366,600369,600372,600376,600383,600395,600406,600415,600418,60
0489,600497,600498,600503,600516,600518,600519,600535,600546,600
547,600549,600583,600585,600588,600597,600598,600600,600633,6006
37,600639,600643,600649,600663,600674,600675,600684,600690,60070
2,600703,600705,600736,600739,600741,600747,600748,600759,600765
,600770,600773,600795,600804,600809,600816,600823,600832,600837,
600875,600880,600886,600887,600893,600895,600900,600970,600999,6
01006,601009,601088,601099,601111,601117,601118,601166,601168,60
1169,601186,601216,601288,601299,601318,601328,601336,601377,601
390,601398,601555,601588,601600,601601,601607,601628,601633,6016
68,601669,601688,601699,601717,601766,601800,601808,601818,60185
7,601888,601899,601901,601928,601939,601958,601988,601989,601992
,601998,603000,603993&date=20110301&field=NPToTOR
20110527
/market/getStockFactorsOneDay.csv?stockID=600000,600008,600009,6
00010,600011,600015,600016,600018,600019,600027,600028,600029,60
0030,600031,600036,600048,600050,600058,600060,600064,600066,600
067,600085,600089,600094,600100,600104,600108,600109,600111,6001
18,600123,600141,600143,600150,600157,600158,600160,600162,60016
6,600170,600177,600188,600196,600199,600208,600216,600221,600239
,600240,600252,600256,600259,600266,600267,600271,600276,600300,
600309,600315,600316,600325,600332,600340,600348,600352,600362,6
00366,600369,600372,600376,600383,600395,600406,600415,600418,60
0489,600497,600498,600503,600516,600518,600519,600535,600546,600
547,600549,600583,600585,600588,600597,600598,600600,600633,6006
37,600639,600643,600649,600663,600674,600675,600684,600690,60070
2,600703,600705,600736,600739,600741,600747,600748,600759,600765
,600770,600773,600795,600804,600809,600816,600823,600832,600837,
600875,600880,600886,600887,600893,600895,600900,600970,600999,6
01006,601009,601088,601099,601111,601117,601118,601166,601168,60
1169,601186,601216,601288,601299,601318,601328,601336,601377,601
390,601398,601555,601588,601600,601601,601607,601628,601633,6016
68,601669,601688,601699,601717,601766,601800,601808,601818,60185
7,601888,601899,601901,601928,601939,601958,601988,601989,601992
,601998,603000,603993&date=20110527&field=NPToTOR
20110822
/market/getStockFactorsOneDay.csv?stockID=600000,600008,600009,6
00010,600011,600015,600016,600018,600019,600027,600028,600029,60
0030,600031,600036,600048,600050,600058,600060,600064,600066,600
067,600085,600089,600094,600100,600104,600108,600109,600111,6001
18,600123,600141,600143,600150,600157,600158,600160,600162,60016
```

```
6,600170,600177,600188,600196,600199,600208,600216,600221,600239
,600240,600252,600256,600259,600266,600267,600271,600276,600300,
600309,600315,600316,600325,600332,600340,600348,600352,600362,6
00366,600369,600372,600376,600383,600395,600406,600415,600418,60
0489,600497,600498,600503,600516,600518,600519,600535,600546,600
547,600549,600583,600585,600588,600597,600598,600600,600633,6006
37,600639,600643,600649,600663,600674,600675,600684,600690,60070
2,600703,600705,600736,600739,600741,600747,600748,600759,600765
,600770,600773,600795,600804,600809,600816,600823,600832,600837,
600875,600880,600886,600887,600893,600895,600900,600970,600999,6
01006,601009,601088,601099,601111,601117,601118,601166,601168,60
1169,601186,601216,601288,601299,601318,601328,601336,601377,601
390,601398,601555,601588,601600,601601,601607,601628,601633,6016
68,601669,601688,601699,601717,601766,601800,601808,601818,60185
7,601888,601899,601901,601928,601939,601958,601988,601989,601992
,601998,603000,603993&date=20110822&field=NPToTOR
20111122
/market/getStockFactorsOneDay.csv?stockID=600000,600008,600009,6
00010,600011,600015,600016,600018,600019,600027,600028,600029,60
0030,600031,600036,600048,600050,600058,600060,600064,600066,600
067,600085,600089,600094,600100,600104,600108,600109,600111,6001
18,600123,600141,600143,600150,600157,600158,600160,600162,60016
6,600170,600177,600188,600196,600199,600208,600216,600221,600239
,600240,600252,600256,600259,600266,600267,600271,600276,600300,
600309,600315,600316,600325,600332,600340,600348,600352,600362,6
00366,600369,600372,600376,600383,600395,600406,600415,600418,60
0489,600497,600498,600503,600516,600518,600519,600535,600546,600
547,600549,600583,600585,600588,600597,600598,600600,600633,6006
37,600639,600643,600649,600663,600674,600675,600684,600690,60070
2,600703,600705,600736,600739,600741,600747,600748,600759,600765
,600770,600773,600795,600804,600809,600816,600823,600832,600837,
600875,600880,600886,600887,600893,600895,600900,600970,600999,6
01006,601009,601088,601099,601111,601117,601118,601166,601168,60
1169,601186,601216,601288,601299,601318,601328,601336,601377,601
390,601398,601555,601588,601600,601601,601607,601628,601633,6016
68,601669,601688,601699,601717,601766,601800,601808,601818,60185
7,601888,601899,601901,601928,601939,601958,601988,601989,601992
,601998,603000,603993&date=20111122&field=NPToTOR
20120223
/market/getStockFactorsOneDay.csv?stockID=600000,600008,600009,6
00010,600011,600015,600016,600018,600019,600027,600028,600029,60
0030,600031,600036,600048,600050,600058,600060,600064,600066,600
067,600085,600089,600094,600100,600104,600108,600109,600111,6001
18,600123,600141,600143,600150,600157,600158,600160,600162,60016
6,600170,600177,600188,600196,600199,600208,600216,600221,600239
,600240,600252,600256,600259,600266,600267,600271,600276,600300,
600309,600315,600316,600325,600332,600340,600348,600352,600362,6
00366,600369,600372,600376,600383,600395,600406,600415,600418,60
0489,600497,600498,600503,600516,600518,600519,600535,600546,600
547,600549,600583,600585,600588,600597,600598,600600,600633,6006
37,600639,600643,600649,600663,600674,600675,600684,600690,60070
2,600703,600705,600736,600739,600741,600747,600748,600759,600765
,600770,600773,600795,600804,600809,600816,600823,600832,600837,
600875,600880,600886,600887,600893,600895,600900,600970,600999,6
01006,601009,601088,601099,601111,601117,601118,601166,601168,60
1169,601186,601216,601288,601299,601318,601328,601336,601377,601
390,601398,601555,601588,601600,601601,601607,601628,601633,6016
68,601669,601688,601699,601717,601766,601800,601808,601818,60185
7,601888,601899,601901,601928,601939,601958,601988,601989,601992
,601998,603000,603993&date=20120223&field=NPToTOR
20120524
/market/getStockFactorsOneDay.csv?stockID=600000,600008,600009,6
00010,600011,600015,600016,600018,600019,600027,600028,600029,60
0030,600031,600036,600048,600050,600058,600060,600064,600066,600
067,600085,600089,600094,600100,600104,600108,600109,600111,6001
18,600123,600141,600143,600150,600157,600158,600160,600162,60016
6,600170,600177,600188,600196,600199,600208,600216,600221,600239
,600240,600252,600256,600259,600266,600267,600271,600276,600300,
600309,600315,600316,600325,600332,600340,600348,600352,600362,6
00366,600369,600372,600376,600383,600395,600406,600415,600418,60
0489,600497,600498,600503,600516,600518,600519,600535,600546,600
547,600549,600583,600585,600588,600597,600598,600600,600633,6006
37,600639,600643,600649,600663,600674,600675,600684,600690,60070
2,600703,600705,600736,600739,600741,600747,600748,600759,600765
,600770,600773,600795,600804,600809,600816,600823,600832,600837,
600875,600880,600886,600887,600893,600895,600900,600970,600999,6
01006,601009,601088,601099,601111,601117,601118,601166,601168,60
1169,601186,601216,601288,601299,601318,601328,601336,601377,601
390,601398,601555,601588,601600,601601,601607,601628,601633,6016
68,601669,601688,601699,601717,601766,601800,601808,601818,60185
7,601888,601899,601901,601928,601939,601958,601988,601989,601992
,601998,603000,603993&date=20120524&field=NPToTOR
20120817
/market/getStockFactorsOneDay.csv?stockID=600000,600008,600009,6
00010,600011,600015,600016,600018,600019,600027,600028,600029,60
0030,600031,600036,600048,600050,600058,600060,600064,600066,600
067,600085,600089,600094,600100,600104,600108,600109,600111,6001
18,600123,600141,600143,600150,600157,600158,600160,600162,60016
6,600170,600177,600188,600196,600199,600208,600216,600221,600239
,600240,600252,600256,600259,600266,600267,600271,600276,600300,
600309,600315,600316,600325,600332,600340,600348,600352,600362,6
00366,600369,600372,600376,600383,600395,600406,600415,600418,60
0489,600497,600498,600503,600516,600518,600519,600535,600546,600
547,600549,600583,600585,600588,600597,600598,600600,600633,6006
37,600639,600643,600649,600663,600674,600675,600684,600690,60070
2,600703,600705,600736,600739,600741,600747,600748,600759,600765
,600770,600773,600795,600804,600809,600816,600823,600832,600837,
600875,600880,600886,600887,600893,600895,600900,600970,600999,6
01006,601009,601088,601099,601111,601117,601118,601166,601168,60
1169,601186,601216,601288,601299,601318,601328,601336,601377,601
390,601398,601555,601588,601600,601601,601607,601628,601633,6016
68,601669,601688,601699,601717,601766,601800,601808,601818,60185
7,601888,601899,601901,601928,601939,601958,601988,601989,601992
,601998,603000,603993&date=20120817&field=NPToTOR
20121116
/market/getStockFactorsOneDay.csv?stockID=600000,600008,600009,6
```

```
00010,600011,600015,600016,600018,600019,600027,600028,600029,60
0030,600031,600036,600048,600050,600058,600060,600064,600066,600
067,600085,600089,600094,600100,600104,600108,600109,600111,6001
18,600123,600141,600143,600150,600157,600158,600160,600162,60016
6,600170,600177,600188,600196,600199,600208,600216,600221,600239
,600240,600252,600256,600259,600266,600267,600271,600276,600300,
600309,600315,600316,600325,600332,600340,600348,600352,600362,6
00366,600369,600372,600376,600383,600395,600406,600415,600418,60
0489,600497,600498,600503,600516,600518,600519,600535,600546,600
547,600549,600583,600585,600588,600597,600598,600600,600633,6006
37,600639,600643,600649,600663,600674,600675,600684,600690,60070
2,600703,600705,600736,600739,600741,600747,600748,600759,600765
,600770,600773,600795,600804,600809,600816,600823,600832,600837,
600875,600880,600886,600887,600893,600895,600900,600970,600999,6
01006,601009,601088,601099,601111,601117,601118,601166,601168,60
1169,601186,601216,601288,601299,601318,601328,601336,601377,601
390,601398,601555,601588,601600,601601,601607,601628,601633,6016
68,601669,601688,601699,601717,601766,601800,601808,601818,60185
7,601888,601899,601901,601928,601939,601958,601988,601989,601992
,601998,603000,603993&date=20121116&field=NPToTOR
20130220
/market/getStockFactorsOneDay.csv?stockID=600000,600008,600009,6
00010,600011,600015,600016,600018,600019,600027,600028,600029,60
0030,600031,600036,600048,600050,600058,600060,600064,600066,600
067,600085,600089,600094,600100,600104,600108,600109,600111,6001
18,600123,600141,600143,600150,600157,600158,600160,600162,60016
6,600170,600177,600188,600196,600199,600208,600216,600221,600239
,600240,600252,600256,600259,600266,600267,600271,600276,600300,
600309,600315,600316,600325,600332,600340,600348,600352,600362,6
00366,600369,600372,600376,600383,600395,600406,600415,600418,60
0489,600497,600498,600503,600516,600518,600519,600535,600546,600
547,600549,600583,600585,600588,600597,600598,600600,600633,6006
37,600639,600643,600649,600663,600674,600675,600684,600690,60070
2,600703,600705,600736,600739,600741,600747,600748,600759,600765
,600770,600773,600795,600804,600809,600816,600823,600832,600837,
600875,600880,600886,600887,600893,600895,600900,600970,600999,6
01006,601009,601088,601099,601111,601117,601118,601166,601168,60
1169,601186,601216,601288,601299,601318,601328,601336,601377,601
390,601398,601555,601588,601600,601601,601607,601628,601633,6016
68,601669,601688,601699,601717,601766,601800,601808,601818,60185
7,601888,601899,601901,601928,601939,601958,601988,601989,601992
,601998,603000,603993&date=20130220&field=NPToTOR
20130522
/market/getStockFactorsOneDay.csv?stockID=600000,600008,600009,6
00010,600011,600015,600016,600018,600019,600027,600028,600029,60
0030,600031,600036,600048,600050,600058,600060,600064,600066,600
067,600085,600089,600094,600100,600104,600108,600109,600111,6001
18,600123,600141,600143,600150,600157,600158,600160,600162,60016
6,600170,600177,600188,600196,600199,600208,600216,600221,600239
,600240,600252,600256,600259,600266,600267,600271,600276,600300,
600309,600315,600316,600325,600332,600340,600348,600352,600362,6
00366,600369,600372,600376,600383,600395,600406,600415,600418,60
0489,600497,600498,600503,600516,600518,600519,600535,600546,600
```

```
547,600549,600583,600585,600588,600597,600598,600600,600633,6006
37,600639,600643,600649,600663,600674,600675,600684,600690,60070
2,600703,600705,600736,600739,600741,600747,600748,600759,600765
,600770,600773,600795,600804,600809,600816,600823,600832,600837,
600875,600880,600886,600887,600893,600895,600900,600970,600999,6
01006,601009,601088,601099,601111,601117,601118,601166,601168,60
1169,601186,601216,601288,601299,601318,601328,601336,601377,601
390,601398,601555,601588,601600,601601,601607,601628,601633,6016
68,601669,601688,601699,601717,601766,601800,601808,601818,60185
7,601888,601899,601901,601928,601939,601958,601988,601989,601992
,601998,603000,603993&date=20130522&field=NPToTOR
20130819
/market/getStockFactorsOneDay.csv?stockID=600000,600008,600009,6
00010,600011,600015,600016,600018,600019,600027,600028,600029,60
0030,600031,600036,600048,600050,600058,600060,600064,600066,600
067,600085,600089,600094,600100,600104,600108,600109,600111,6001
18,600123,600141,600143,600150,600157,600158,600160,600162,60016
6,600170,600177,600188,600196,600199,600208,600216,600221,600239
,600240,600252,600256,600259,600266,600267,600271,600276,600300,
600309,600315,600316,600325,600332,600340,600348,600352,600362,6
00366,600369,600372,600376,600383,600395,600406,600415,600418,60
0489,600497,600498,600503,600516,600518,600519,600535,600546,600
547,600549,600583,600585,600588,600597,600598,600600,600633,6006
37,600639,600643,600649,600663,600674,600675,600684,600690,60070
2,600703,600705,600736,600739,600741,600747,600748,600759,600765
,600770,600773,600795,600804,600809,600816,600823,600832,600837,
600875,600880,600886,600887,600893,600895,600900,600970,600999,6
01006,601009,601088,601099,601111,601117,601118,601166,601168,60
1169,601186,601216,601288,601299,601318,601328,601336,601377,601
390,601398,601555,601588,601600,601601,601607,601628,601633,6016
68,601669,601688,601699,601717,601766,601800,601808,601818,60185
7,601888,601899,601901,601928,601939,601958,601988,601989,601992
,601998,603000,603993&date=20130819&field=NPToTOR
20131120
/market/getStockFactorsOneDay.csv?stockID=600000,600008,600009,6
00010,600011,600015,600016,600018,600019,600027,600028,600029,60
0030,600031,600036,600048,600050,600058,600060,600064,600066,600
067,600085,600089,600094,600100,600104,600108,600109,600111,6001
18,600123,600141,600143,600150,600157,600158,600160,600162,60016
6,600170,600177,600188,600196,600199,600208,600216,600221,600239
,600240,600252,600256,600259,600266,600267,600271,600276,600300,
600309,600315,600316,600325,600332,600340,600348,600352,600362,6
00366,600369,600372,600376,600383,600395,600406,600415,600418,60
0489,600497,600498,600503,600516,600518,600519,600535,600546,600
547,600549,600583,600585,600588,600597,600598,600600,600633,6006
37,600639,600643,600649,600663,600674,600675,600684,600690,60070
2,600703,600705,600736,600739,600741,600747,600748,600759,600765
,600770,600773,600795,600804,600809,600816,600823,600832,600837,
600875,600880,600886,600887,600893,600895,600900,600970,600999,6
01006,601009,601088,601099,601111,601117,601118,601166,601168,60
1169,601186,601216,601288,601299,601318,601328,601336,601377,601
390,601398,601555,601588,601600,601601,601607,601628,601633,6016
68,601669,601688,601699,601717,601766,601800,601808,601818,60185
```

```
7,601888,601899,601901,601928,601939,601958,601988,601989,601992
,601998,603000,603993&date=20131120&field=NPToTOR
20140220
/market/getStockFactorsOneDay.csv?stockID=600000,600008,600009,6
00010,600011,600015,600016,600018,600019,600027,600028,600029,60
0030,600031,600036,600048,600050,600058,600060,600064,600066,600
067,600085,600089,600094,600100,600104,600108,600109,600111,6001
18,600123,600141,600143,600150,600157,600158,600160,600162,60016
6,600170,600177,600188,600196,600199,600208,600216,600221,600239
,600240,600252,600256,600259,600266,600267,600271,600276,600300,
600309,600315,600316,600325,600332,600340,600348,600352,600362,6
00366,600369,600372,600376,600383,600395,600406,600415,600418,60
0489,600497,600498,600503,600516,600518,600519,600535,600546,600
547,600549,600583,600585,600588,600597,600598,600600,600633,6006
37,600639,600643,600649,600663,600674,600675,600684,600690,60070
2,600703,600705,600736,600739,600741,600747,600748,600759,600765
,600770,600773,600795,600804,600809,600816,600823,600832,600837,
600875,600880,600886,600887,600893,600895,600900,600970,600999,6
01006,601009,601088,601099,601111,601117,601118,601166,601168,60
1169,601186,601216,601288,601299,601318,601328,601336,601377,601
390,601398,601555,601588,601600,601601,601607,601628,601633,6016
68,601669,601688,601699,601717,601766,601800,601808,601818,60185
7,601888,601899,601901,601928,601939,601958,601988,601989,601992
,601998,603000,603993&date=20140220&field=NPToTOR
20140520
/market/getStockFactorsOneDay.csv?stockID=600000,600008,600009,6
00010,600011,600015,600016,600018,600019,600027,600028,600029,60
0030,600031,600036,600048,600050,600058,600060,600064,600066,600
067,600085,600089,600094,600100,600104,600108,600109,600111,6001
18,600123,600141,600143,600150,600157,600158,600160,600162,60016
6,600170,600177,600188,600196,600199,600208,600216,600221,600239
,600240,600252,600256,600259,600266,600267,600271,600276,600300,
600309,600315,600316,600325,600332,600340,600348,600352,600362,6
00366,600369,600372,600376,600383,600395,600406,600415,600418,60
0489,600497,600498,600503,600516,600518,600519,600535,600546,600
547,600549,600583,600585,600588,600597,600598,600600,600633,6006
37,600639,600643,600649,600663,600674,600675,600684,600690,60070
2,600703,600705,600736,600739,600741,600747,600748,600759,600765
,600770,600773,600795,600804,600809,600816,600823,600832,600837,
600875,600880,600886,600887,600893,600895,600900,600970,600999,6
01006,601009,601088,601099,601111,601117,601118,601166,601168,60
1169,601186,601216,601288,601299,601318,601328,601336,601377,601
390,601398,601555,601588,601600,601601,601607,601628,601633,6016
68,601669,601688,601699,601717,601766,601800,601808,601818,60185
7,601888,601899,601901,601928,601939,601958,601988,601989,601992
,601998,603000,603993&date=20140520&field=NPToTOR
20140813
/market/getStockFactorsOneDay.csv?stockID=600000,600008,600009,6
00010,600011,600015,600016,600018,600019,600027,600028,600029,60
0030,600031,600036,600048,600050,600058,600060,600064,600066,600
067,600085,600089,600094,600100,600104,600108,600109,600111,6001
18,600123,600141,600143,600150,600157,600158,600160,600162,60016
6,600170,600177,600188,600196,600199,600208,600216,600221,600239
```

```
,600240,600252,600256,600259,600266,600267,600271,600276,600300,
600309,600315,600316,600325,600332,600340,600348,600352,600362,6
00366,600369,600372,600376,600383,600395,600406,600415,600418,60
0489,600497,600498,600503,600516,600518,600519,600535,600546,600
547,600549,600583,600585,600588,600597,600598,600600,600633,6006
37,600639,600643,600649,600663,600674,600675,600684,600690,60070
2,600703,600705,600736,600739,600741,600747,600748,600759,600765
,600770,600773,600795,600804,600809,600816,600823,600832,600837,
600875,600880,600886,600887,600893,600895,600900,600970,600999,6
01006,601009,601088,601099,601111,601117,601118,601166,601168,60
1169,601186,601216,601288,601299,601318,601328,601336,601377,601
390,601398,601555,601588,601600,601601,601607,601628,601633,6016
68,601669,601688,601699,601717,601766,601800,601808,601818,60185
7,601888,601899,601901,601928,601939,601958,601988,601989,601992
,601998,603000,603993&date=20140813&field=NPToTOR
20141113
/market/getStockFactorsOneDay.csv?stockID=600000,600008,600009,6
00010,600011,600015,600016,600018,600019,600027,600028,600029,60
0030,600031,600036,600048,600050,600058,600060,600064,600066,600
067,600085,600089,600094,600100,600104,600108,600109,600111,6001
18,600123,600141,600143,600150,600157,600158,600160,600162,60016
6,600170,600177,600188,600196,600199,600208,600216,600221,600239
,600240,600252,600256,600259,600266,600267,600271,600276,600300,
600309,600315,600316,600325,600332,600340,600348,600352,600362,6
00366,600369,600372,600376,600383,600395,600406,600415,600418,60
0489,600497,600498,600503,600516,600518,600519,600535,600546,600
547,600549,600583,600585,600588,600597,600598,600600,600633,6006
37,600639,600643,600649,600663,600674,600675,600684,600690,60070
2,600703,600705,600736,600739,600741,600747,600748,600759,600765
,600770,600773,600795,600804,600809,600816,600823,600832,600837,
600875,600880,600886,600887,600893,600895,600900,600970,600999,6
01006,601009,601088,601099,601111,601117,601118,601166,601168,60
1169,601186,601216,601288,601299,601318,601328,601336,601377,601
390,601398,601555,601588,601600,601601,601607,601628,601633,6016
68,601669,601688,601699,601717,601766,601800,601808,601818,60185
7,601888,601899,601901,601928,601939,601958,601988,601989,601992
,601998,603000,603993&date=20141113&field=NPToTOR
```

```
min(bt.cash)
```

```
0.59175999897206566
```

营业收入增长率+市盈率

> 来源：https://uqer.io/community/share/568cd66e228e5b960b7fd252

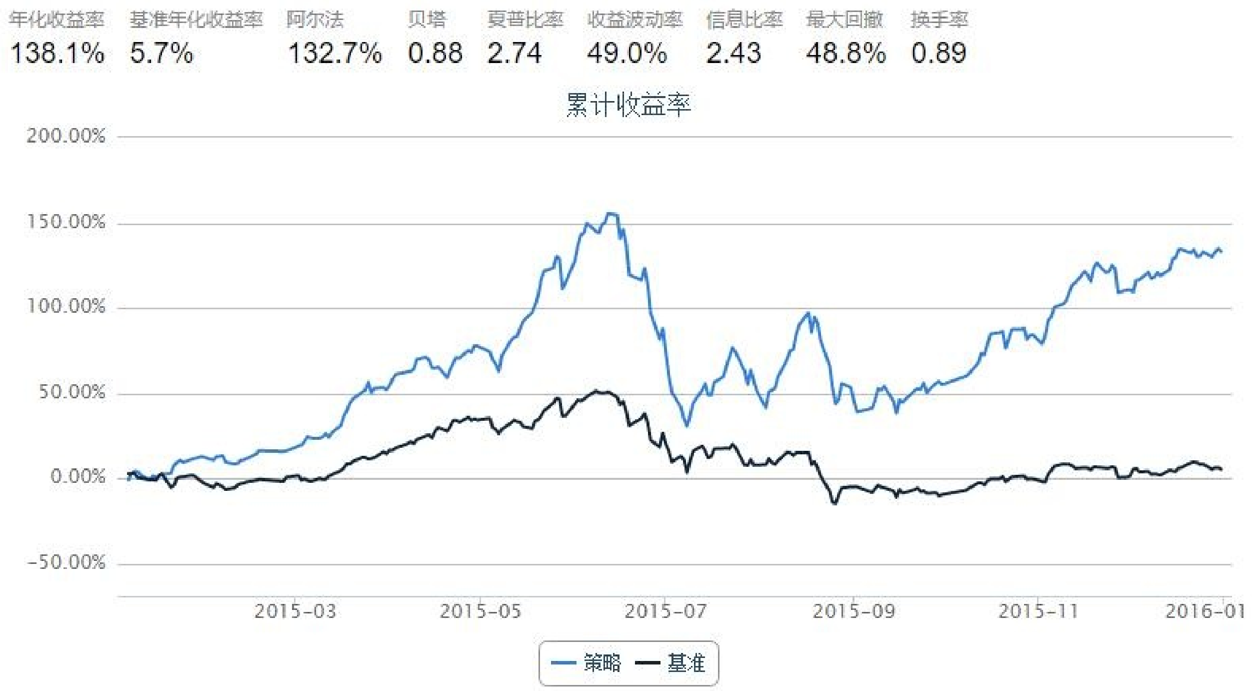
策略思路：

买入A股中同时满足以下条件的股票：

- 营业收入增长率最大的200只股票
- 市盈率最低的200只股票

实际操作中，总是持有满足上述两个条件的股票集合的交集，按月调仓

```
start = '2015-01-01'                       # 回测起始时间
end = '2016-01-01'                         # 回测结束时间
benchmark = 'HS300'                        # 策略参考标准
capital_base = 100000                      # 起始资金
freq = 'd'                               # 策略类型，'d'表示日间策略使用日线回
测，'m'表示日内策略使用分钟线回测
refresh_rate = 20                          # 调仓频率，表示执行handle_data的
时间间隔，若freq = 'd'时间间隔的单位为交易日，若freq = 'm'时间间隔为分钟

# 证券池，支持股票和基金
# 股票池设为营业收入增长率最大的200只股票与市盈率最低的200只股票的交集
universe = StockScreener(Factor.OperatingRevenueGrowRate.nlarge(
200) & Factor.PE.nsmall(200))

def initialize(account):                        # 初始化虚拟账户状态
    pass

def handle_data(account):                       # 每个交易日的买入卖出指令

    buylist = {stk:0 for stk in account.universe}

    # 不满足条件的股票，清仓
    for s in account.valid_secpos:
        if s not in buylist:
            order_to(s, 0)

    # 满足条件的股票，不管多少只，都等仓位买入
    v = account.referencePortfolioValue / len(buylist)    # 每只
股票买入金额
    for s in buylist:
        # 计算每只股票买卖股数： 正数为买入，负数为卖出
        buylist[s] = v / account.referencePrice[s] - account.val
id_secpos.get(s, 0)

    for s in sorted(buylist, key=buylist.get):
        order(s, buylist[s])
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
138.1%	5.7%	132.7%	0.88	2.74	49.0%	2.43	48.8%	0.89

1.3 财报阅读 • [米缸量化读财报] 资产负债表-投资相关资产

> 来源：https://uqer.io/community/share/56487f83f9f06c4446b48179

众所周知，股权投资最重要的是，对投资标的的了解。而对投资标的的了解，莫过于阅读上市公司的财报。

近来重读雪球大V唐朝的《手把手教你读财报》，温故而知新，希望从中有更多的收获。

唐朝的这本书，以贵州茅台的财报为例子，深入浅出的讲述了符合《企业会计准则》的财报的阅读方式和思维结构。

唐朝提出的思想和我的想法不谋而合。我作为证伪主义投资者，当读到“财报是用来排除企业的”这句副标题，会心一笑，掏钱把书买回家好好品读。

唐朝的这本书讲的很棒，但有几点不足

- 一是没有将多份研报横向对比，单独读一份研报，有一些数据无法挖掘的更深。行业内横向对比是必要的。
- 二是没有全A股范围的排序和分析。当我们将同样的一个数据放在全市场视角上来看，一定会看出不同的内涵。
- 三是讲了方法论，但没有给出结论，比如第四章，将股票分为8类，只给出了8种公司的特征，却没有给出每类公司的列表。

当我看到优矿平台和通联数据的免费海量金融数据时，欣喜若狂，如获至宝。

我本着继续深入学习财务知识的目的，开始重读唐老的这本“财报圣经”。尝试将其中的观点，以真实完整的全市场数据进行展示和验证，希望从中能够学到更深入的知识，从而加深对中国证券市场结构的理解。

其中的个人观点，大家请一看而过。

一、 随便聊聊

1.1 资产负债表

财报的内容很多，最主要，占篇幅最大的，莫过于三张表。而三张表根据最新的会计准则，又添加了很多条新的科目，便于投资者、管理者进行细节的考察和判断。

我们的讨论从最重要的资产负债表开始。如果你经历了2015年的股灾，你一定对资产、负债的概念不陌生。

股灾中，经常提到的强制平仓线，其实就是通过资产负债比来确定的。如果你的资产负债比小于130%，那么Margin Call和“强平”就会到来。

而我们看资产负债表，看的是什么？其实就是在看企业是否会被“强平”。因为大部分企业都是在借助“杠杆”来经营呢。

如果一个企业资产大大多于负债，那他的经营能力还是不错的呢。

1.2 资产

那我们就从资产说起。

资产是什么？根据一般性的定义，你拥有一个东西，它能够产生正向的现金流，那么它就是一个资产。

也就是说，你拥有这个东西，你不需要卖掉它，它也会给你带来现金流，那它就是资产无疑。投资投资，就是投入资产。

比如说，你买个房子，租出去，那每个月都会有租金的收入，那房子就是资产无疑。

如果你买个股票，每年都会给你现金分红，那这个股票也是资产无疑。

而如果一个东西你需要买了再卖，才能实现现金流，那这个就叫speculation啦。

1.3 投资相关资产

在唐老的书中，将资产分为“货币资金、经营相关资产、生产相关资产、投资相关资产”四类，甚是精妙。

我从最后一类开始分析。为什么呢？

马克思曾经说过，资本家分为商业资本家和金融资本家。商业资本家通过“剥削”普通劳动者来实现剩余价值，而金融资本家通过“剥削”商业资本家来获取剩余价值。

恭喜你，你作为某个上市公司的股东，你也是“金融资本家”的一份子了。XD

上市公司不仅仅有生产经营性活动，又有对外投资活动，所以又是“商业资本家”，又是“金融资本家”。

我们可以从“投资相关资产”看出企业的投资选择，从而学习其投资思想，判断其投资能力。

除此之外，你作为股东，拥有某个公司，而这个公司又控股其他子公司或者参股其他公司。你也就是那些子公司的间接控制者了。

而这些控股关系，也可以从投资相关资产这部分科目里看出来的。

所以，我们优先看看“投资相关资产”，从这里你可以看到整个市场的控股关系。

由于现在已经是完全的资本运作时代，很多公司的盈利并不是通过传统的“销售存货，获得现金”这种老套的模式获得的了。

而是通过兼并重组，投资其他公司，或者投资其他金融产品获得的。

之前房地产疯狂的时候，你发现很多公司，不管是哪个行业，做什么的，都转型房地产企业了。

而这部分房地产投资，恰恰就记在了投资相关资产的“投资性房地产”科目里。

从这里开始，我们开始正式的财报学习和研究。因为我也是财报世界的初学者，所以希望各位同学能够提供更多的建议和帮助。

二、 数据准备

再次表达一下优矿给我带来的惊喜。免费金融数据，可自定义的研究平台，很好的量化交流社区。

数据准备，我通过通联数据DataAPI获取全A股所有公司的最近一次财报，也就是2015年Q3的财报。

根据唐老的分类，共有6个科目属于投资相关资产。分别是'交易性金融资产', '持有至到期投资', '可供出售金融资产', '长期股权投资', ' 买入返售金融资产', '投资性房地产'。

```python
import pandas as pd

universe = set_universe('A')

whole_set = pd.DataFrame()
for stock in universe:
    try:
        data = DataAPI.FdmtBSGet(ticker=u"",secID=stock,reportType=u"",endDate=u"20150930",beginDate=u"20150830",publishDateEnd=u"",publishDateBegin=u"",endDateRep="",beginDateRep="",beginYear="",endYear="",fiscalPeriod="",field=u"ticker,secShortName,endDate,reportType,tradingFA,htmInvest,availForSaleFa,LTEquityInvest,purResaleFa,investRealEstate",pandas="1")
        whole_set = whole_set.append(data, ignore_index=True)
    except Exception:
        # print stock
        pass

whole_set = whole_set.set_index(['ticker'], inplace=False).fillna(0)
whole_set.columns = ['名称', '期末日期', '类型', '交易性金融资产', '持有至到期投资', '可供出售金融资产', '长期股权投资', '买入返售金融资产', '投资性房地产']

whole_set.to_csv('invest_asset.csv', encoding='GB18030')
```

三、 一项项看过来

3.1 交易性金融资产

交易性金融资产（Financial assets held for trading）是指企业为了近期内出售而持有的债券投资、股票投资和基金投资。如以赚取差价为目的从二级市场购买的股票、债券、基金等。如以赚取差价为目的从二级市场购买的股票、债券、基金等。[1]

“交易性金融资产的特点是不需要计提折旧减值，直接以持有期间的公允价值变动，作为该项资产的当期损益。”

也就是说，这部分资产，会直接影响公司的利润。那这部分价值的涨跌会影响公司的净资产变动，进而影响估值，进而影响股价。所以，可以关注一下，交易性金融资产对公司股价的影响。

交易性金融资产和利润的关联，我们在学习到利润表的时候，再分析。现在只看这一科目。

下面，我们来观察一下全A股，这一科目的排名。

```python
whole_set.sort('交易性金融资产', ascending=False).head(20)
```

	名称	期末日期	类型	交易性金融资产	持有至到期投资	可供出售金融资产
ticker						
601398	工商银行	2015-09-30	Q3	4.657340e+11	2.815399e+12	1.388838e+
601288	农业银行	2015-09-30	Q3	4.127890e+11	2.268457e+12	1.015701e+
601939	建设银行	2015-09-30	Q3	1.594500e+11	2.592316e+12	9.462150e+
601328	交通银行	2015-09-30	Q3	1.455020e+11	8.433740e+11	2.532930e+
601988	中国银行	2015-09-30	Q3	1.211650e+11	1.691177e+12	9.805820e+

601688	华泰证券	2015-09-30	Q3	1.103375e+11	5.000000e+06	2.535281e+1
000166	申万宏源	2015-09-30	Q3	7.485884e+10	1.162228e+08	2.234452e+1
600999	招商证券	2015-09-30	Q3	6.016705e+10	0.000000e+00	3.493192e+1
601318	中国平安	2015-09-30	Q3	4.988100e+10	8.719430e+11	4.233710e+1
600036	招商银行	2015-09-30	Q3	4.819000e+10	3.430530e+11	2.929180e+1
600000	浦发银行	2015-09-30	Q3	4.730400e+10	2.215700e+11	2.141730e+1
601628	中国人寿	2015-09-30	Q3	4.702600e+10	5.051350e+11	7.290770e+1
601166	兴业银行	2015-09-30	Q3	4.174800e+10	2.030120e+11	3.752570e+1
601901	方正证券	2015-09-30	Q3	3.725984e+10	1.408641e+09	1.956258e+1
600016	民生银行	2015-09-30	Q3	3.090100e+10	2.339070e+11	1.705880e+1
	光					

601788	大证券	2015-09-30	Q3	2.629547e+10	1.268603e+08	1.702767e+1
000686	东北证券	2015-09-30	Q3	2.564473e+10	0.000000e+00	5.195486e+0
601998	中信银行	2015-09-30	Q3	2.328700e+10	1.814820e+11	3.600010e+1
601009	南京银行	2015-09-30	Q3	2.209831e+10	7.358643e+10	9.241657e+1
601601	中国太保	2015-09-30	Q3	2.160100e+10	3.116450e+11	1.939390e+1

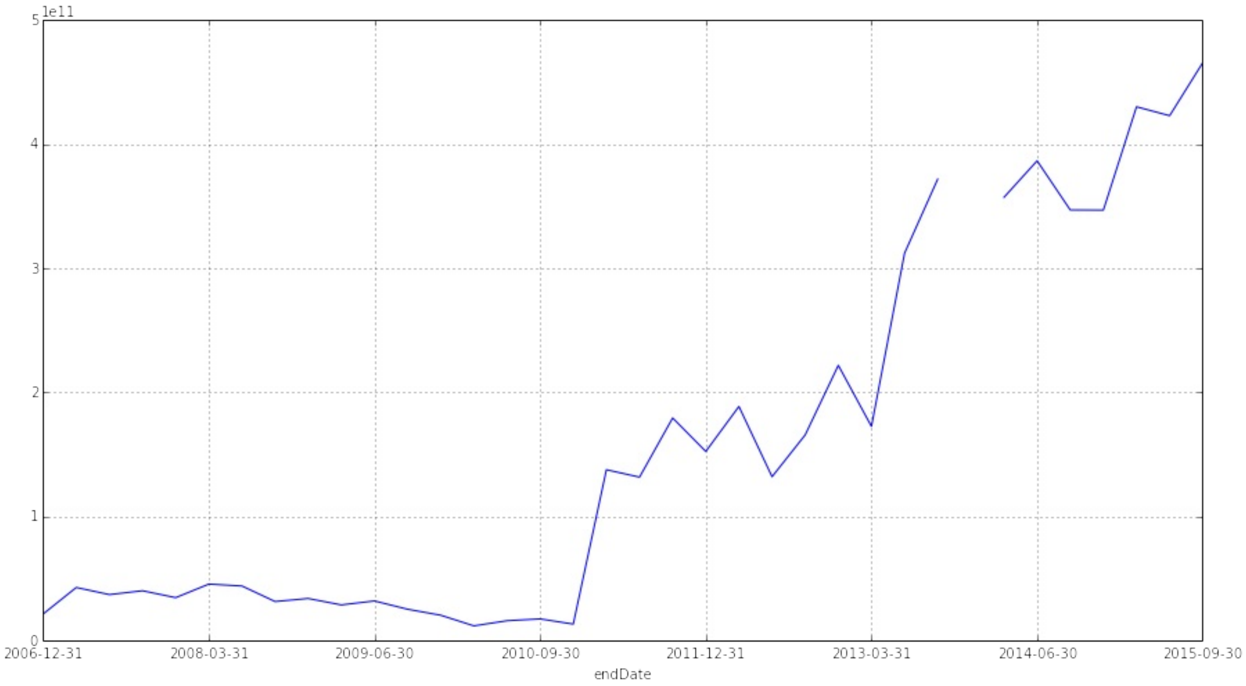
一眼望去都是证券、银行、保险，俗称“金三胖”的公司。

工商银行拥有4657亿元(`4.657340e+11`)的交易性金融资产，是所有上市公司最多的。

查看工商银行2014年年度财报，在财报附注中，提到工商银行参与的得金融资产类型，包括债券投资，比如国债，以及存放同业及其他金融机构款项等。这些主要和其关联方进行的交易。

下面画图看看工商银行这几年，交易性金融资产的变动情况。[2]

```
data = DataAPI.FdmtBSAllLatestGet(ticker=u"601398",secID=u"",reportType=u"",endDate=u"",beginDate=u"",field=u"ticker,secShortName,endDate,reportType,tradingFA,htmInvest,availForSaleFa,LTEquityInvest,purResaleFa,investRealEstate",pandas="1")
data = data.set_index('endDate', inplace=False).sort_index()

data['tradingFA'].plot(figsize=(15,8))

<matplotlib.axes.AxesSubplot at 0x3b97590>
```



2010年底，工商银行，突然持有大量可交易金融资产。这是为什么？谁能告诉我？

工商银行的主要关联方，有两个，财政部，中央汇金公司。随便一猜，2015年9月Q3财报，持有增多。多出来的那部分交易性金融资产，是借给汇金公司的吧。

3.2 持有至到期投资

持有至到期投资是指到期日固定、回收金额固定或可确定，且企业有明确意图和能力持有至到期的非衍生金融资产。通常情况下，包括企业持有的、在活跃市场上有公开报价的国债、企业债券、金融债券等。[3]

顾名思义，就是一直持有，不交易的投资。一般就是固定收益类投资了。

```
whole_set.sort('持有至到期投资', ascending=False).head(20)
```

	名称	期末日期	类型	交易性金融资产	持有至到期投资	可供出售金融资产
ticker						
601398	工商银行	2015-09-30	Q3	4.657340e+11	2.815399e+12	1.388838e+
601939	建设银行	2015-09-30	Q3	1.594500e+11	2.592316e+12	9.462150e+

601288	农业银行	2015-09-30	Q3	4.127890e+11	2.268457e+12	1.015701e+
601988	中国银行	2015-09-30	Q3	1.211650e+11	1.691177e+12	9.805820e+
601318	中国平安	2015-09-30	Q3	4.988100e+10	8.719430e+11	4.233710e+
601328	交通银行	2015-09-30	Q3	1.455020e+11	8.433740e+11	2.532930e+
601628	中国人寿	2015-09-30	Q3	4.702600e+10	5.051350e+11	7.290770e+
600036	招商银行	2015-09-30	Q3	4.819000e+10	3.430530e+11	2.929180e+
601601	中国太保	2015-09-30	Q3	2.160100e+10	3.116450e+11	1.939390e+
000001	平安银行	2015-09-30	Q3	1.759700e+10	2.600220e+11	7.660000e+
600016	民生银行	2015-09-30	Q3	3.090100e+10	2.339070e+11	1.705880e+
600000	浦发银行	2015-09-30	Q3	4.730400e+10	2.215700e+11	2.141730e+
601166	兴业	2015-	Q3	4.174800e+10	2.030120e+11	3.752570e+

601166	银行	09-30	Q3	4.174800e+10	2.030120e+11	3.752570e+1
601998	中信银行	2015-09-30	Q3	2.328700e+10	1.814820e+11	3.600010e+1
601336	新华保险	2015-09-30	Q3	1.883300e+10	1.766500e+11	1.816840e+1
600015	华夏银行	2015-09-30	Q3	9.595000e+09	1.741240e+11	7.099300e+1
601818	光大银行	2015-09-30	Q3	8.604000e+09	1.536820e+11	2.080620e+1
601169	北京银行	2015-09-30	Q3	1.702700e+10	1.468580e+11	1.125920e+1
601009	南京银行	2015-09-30	Q3	2.209831e+10	7.358643e+10	9.241657e+1
002142	宁波银行	2015-09-30	Q3	9.975815e+09	3.046485e+10	1.968860e+1

一眼望去，又是金三胖。我们跳过金三胖，向下看，20名到30名。

```
whole_set.sort('持有至到期投资', ascending=False)[20:30]
```

	名称	期末日期	类型	交易性金融资产	持有至到期投资	可供出售金融资产
ticker						
	西水	2015-				

	份					
600606	绿地控股	2015-09-30	Q3	2.754451e+09	2.573969e+09	6.689355e+0
600019	宝钢股份	2015-09-30	Q3	2.631434e+07	1.455592e+09	1.005997e+1
601901	方正证券	2015-09-30	Q3	3.725984e+10	1.408641e+09	1.956258e+1
000563	陕国投A	2015-09-30	Q3	1.885250e+07	1.313211e+09	7.481469e+0
600958	东方证券	2015-09-30	Q3	2.139055e+10	1.213959e+09	5.849011e+1
002498	汉缆股份	2015-09-30	Q3	5.640295e+08	1.049600e+09	5.400000e+0
600602	仪电电子	2015-09-30	Q3	0.000000e+00	9.620000e+08	2.411250e+0
000060	中金岭南	2015-09-30	Q3	9.590232e+05	7.655839e+08	1.680425e+0
002181	粤传媒	2015-09-30	Q3	0.000000e+00	7.250039e+08	1.490160e+0

西水股份排在前面。

西水股份全称内蒙古西水创业股份有限公司，经营范围是矿产品、建材产品、化工产品、机器设备、五金产品、电子产品的销售；机械设备租赁；软件开发。

是内蒙古地区最大的水泥生产企业。不过顺带做做软件开发是什么鬼？

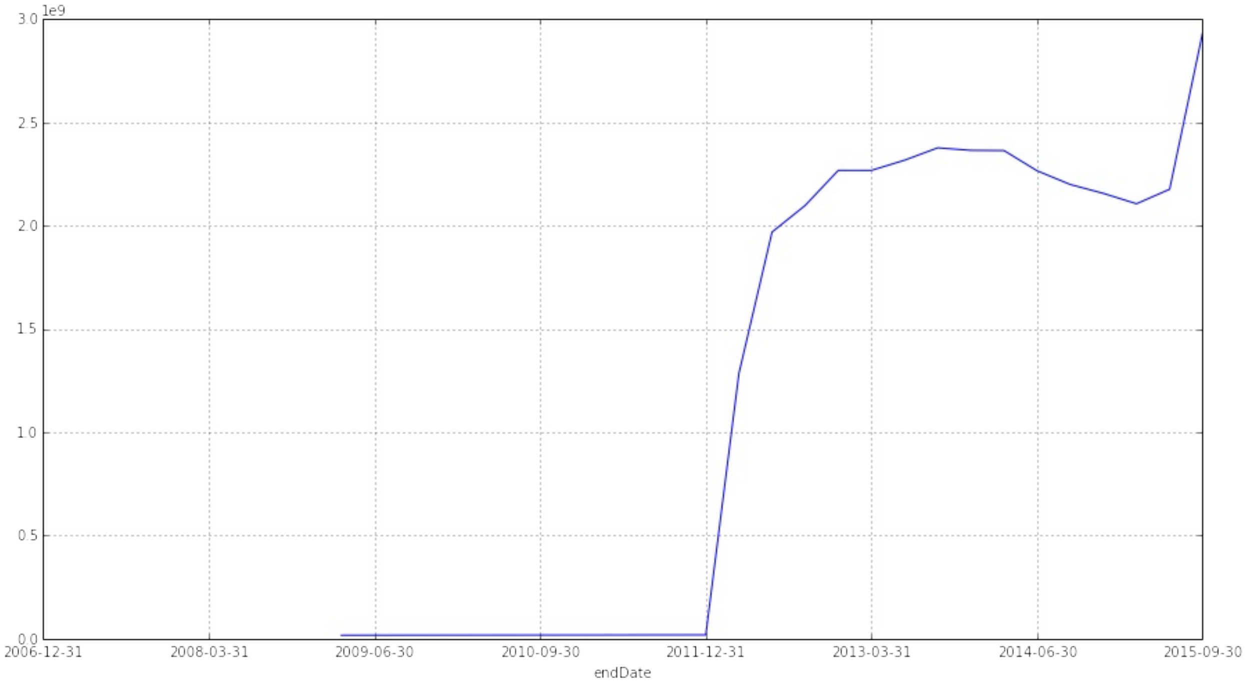
他的持有至到期投资有29亿(`2.937122e+09`)。这么多都干嘛去了？

我们来看下西水股份的2015年半年报[4]，在第79页第15小项，也就是财报附注中，我们看到西水股份的持有到期投资有三项：政府债券、金融债券、企业债券，其中百分之九十以上是企业债券。

以2015年半年报来看，持有到期投资为21.9亿。就是说，六月到九月的三个月，多了7个亿的固定收益投资。钱从哪儿来的？

我们来看下西水股份历年的‘持有到期投资’情况。

```
data = DataAPI.FdmtBSAllLatestGet(ticker=u"600291",secID=u"",reportType=u"",endDate=u"",beginDate=u"",field=u"ticker,secShortName,endDate,reportType,tradingFA,htmInvest,availForSaleFa,LTEquityInvest,purResaleFa,investRealEstate",pandas="1")
data = data.set_index('endDate', inplace=False).sort_index()

data['htmInvest'].plot(figsize=(15,8))

<matplotlib.axes.AxesSubplot at 0x3f2d5d0>
```



从曲线上看，最近三个月突然多了很多。这个和我上面在财报上对数值的观察一致。

唐老在书中，讲到要格外关注“持有到期投资”大额减值，因为该项可以用来操控利润。

那西水股份这种大额增值是不是也需要关注呢？

3.3 可供出售金融资产

通常是指企业初始确认时即被指定为可供出售的非衍生金融资产，以及没有划分为以公允价值计量且其变动计入当期损益的金融资产、持有至到期投资、贷款和应收款项的金融资产。[5]

用唐老的话，就是管理层没想好这部分资产，是属于交易性，还是持有到期，干脆放在“可供出售金融资产”科目中。

```
whole_set.sort('可供出售金融资产', ascending=False).head(20)
```

	名称	期末日期	类型	交易性金融资产	持有至到期投资	可供出售金融资产
ticker						
601398	工商银行	2015-09-30	Q3	4.657340e+11	2.815399e+12	1.388838e+1
601288	农业银行	2015-09-30	Q3	4.127890e+11	2.268457e+12	1.015701e+1
601988	中国银行	2015-09-30	Q3	1.211650e+11	1.691177e+12	9.805820e+1
601939	建设银行	2015-09-30	Q3	1.594500e+11	2.592316e+12	9.462150e+1
601628	中国人寿	2015-09-30	Q3	4.702600e+10	5.051350e+11	7.290770e+1
601318	中国平安	2015-09-30	Q3	4.988100e+10	8.719430e+11	4.233710e+1
601166	兴业银行	2015-09-30	Q3	4.174800e+10	2.030120e+11	3.752570e+1
	中					

	行					
601998	中信银行	2015-09-30	Q3	2.328700e+10	1.814820e+11	3.600010e+
600036	招商银行	2015-09-30	Q3	4.819000e+10	3.430530e+11	2.929180e+
601328	交通银行	2015-09-30	Q3	1.455020e+11	8.433740e+11	2.532930e+
600000	浦发银行	2015-09-30	Q3	4.730400e+10	2.215700e+11	2.141730e+
601818	光大银行	2015-09-30	Q3	8.604000e+09	1.536820e+11	2.080620e+
002142	宁波银行	2015-09-30	Q3	9.975815e+09	3.046485e+10	1.968860e+
601601	中国太保	2015-09-30	Q3	2.160100e+10	3.116450e+11	1.939390e+
601336	新华保险	2015-09-30	Q3	1.883300e+10	1.766500e+11	1.816840e+
600016	民生银行	2015-09-30	Q3	3.090100e+10	2.339070e+11	1.705880e+
601169	北京银行	2015-09-30	Q3	1.702700e+10	1.468580e+11	1.125920e+

	银行					
600291	西水股份	2015-09-30	Q3	0.000000e+00	2.937122e+09	7.327862e+1
600015	华夏银行	2015-09-30	Q3	9.595000e+09	1.741240e+11	7.099300e+1

看上面的表，不得不说，西水股份挺厉害的，在金三胖里面也数上号了。万绿丛中一点红。

顺着排名继续看。

```
whole_set.sort('可供出售金融资产', ascending=False)[20:30]
```

	名称	期末日期	类型	交易性金融资产	持有至到期投资	可供出售金融资产
ticker						
600030	中信证券	2015-09-30	Q3	0.000000e+00	0.000000e+00	7.060994e+1
000776	广发证券	2015-09-30	Q3	0.000000e+00	0.000000e+00	6.921736e+1
600958	东方证券	2015-09-30	Q3	2.139055e+10	1.213959e+09	5.849011e+1
600104	上汽集团	2015-09-30	Q3	3.239876e+08	0.000000e+00	5.687617e+1
600999	招商证券	2015-09-30	Q3	6.016705e+10	0.000000e+00	3.493192e+1

600177	雅戈尔	2015-09-30	Q3	0.000000e+00	0.000000e+00	2.650836e+1
601688	华泰证券	2015-09-30	Q3	1.103375e+11	5.000000e+06	2.535281e+1
000166	申万宏源	2015-09-30	Q3	7.485884e+10	1.162228e+08	2.234452e+1
601198	东兴证券	2015-09-30	Q3	0.000000e+00	0.000000e+00	2.198282e+1
002736	国信证券	2015-09-30	Q3	0.000000e+00	0.000000e+00	2.180174e+1

上汽集团上榜。他的“可供出售金融资产”有568亿(`5.687617e+10`)。

其实我一直挺奇怪的，做汽车的，做水泥的，要这么多债券干什么？

不好好造汽车，都搞投资去了么？

上汽集团全称上海汽车集团股份有限公司。主营业务：汽车，摩托车，拖拉机等各种机动车整车，机械设备，总成及零部件的生产、销售，国内贸易。

还挺靠谱，他是沪港通，个股期权等多个交易品种的测试用例。关注度相当高。

我们来看上汽集团2015年半年报[6]。

在第8页，(四)1这一项下，说明了可供出售的金融资内涵。原来是持有了招商银行的股份，股份来源是定向增发、配股。不过不多，80亿。

在第23页，利润表中，看到“可供出售金融资产公允价值变动损益”这一项，增加了20亿。大家懂得，牛市大家都有的赚。也说明了该科目对利润的影响。

在60页，第13点，我们详细看到了上汽集团，到底持有了那些“可供出售金融资产”。分为可供出售权益工具，和可供出售债务工具。其中可供出售权益工具占绝大多数。

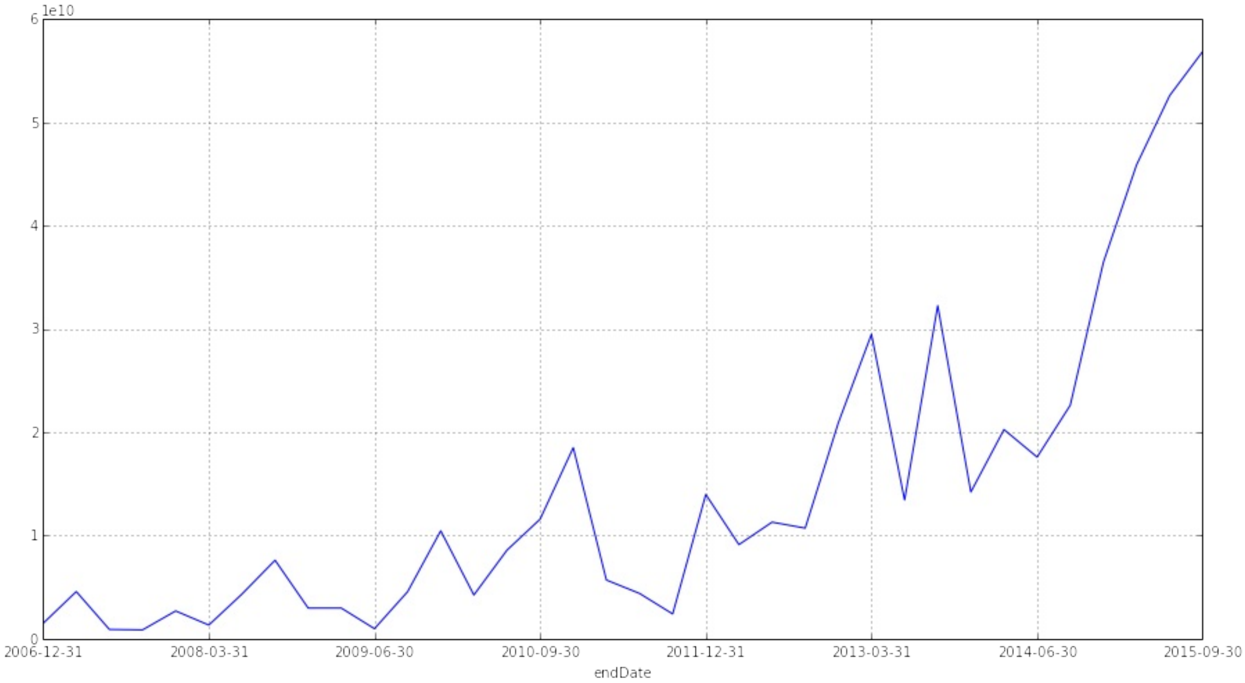
简单说，就是上汽集团也“炒股”。

其中包含几个公司的股份，包含“通用汽车韩国公司”、“上海国际信托有限公司”、“国汽(北京)汽车轻量化技术研究院有限公司”、“天津雷沃重工有限公司”。

其中通用汽车韩国公司占可供出售金融资产的比例最大，4.9亿(494,566,740.00)，不过，该项已经进入了减值准备一栏，极有可能上汽集团要出售该部分股份。

需要密切关注。

```
data = DataAPI.FdmtBSAllLatestGet(ticker=u"600104",secID=u"",rep
ortType=u"",endDate=u"",beginDate=u"",field=u"ticker,secShortNam
e,endDate,reportType,tradingFA,htmInvest,availForSaleFa,LTEquity
Invest,purResaleFa,investRealEstate",pandas="1")
data = data.set_index('endDate', inplace=False).sort_index()

data['availForSaleFa'].plot(figsize=(15,8))

<matplotlib.axes.AxesSubplot at 0x4bf9090>
```



上汽集团的“可供出售金融资产”一直保持增长。尤其2014年6月30日之后，突飞猛进。

再次印证，上汽集团在牛市里挣到钱了。

在财报附注，83页，55小项，提到“可供出售金融资产取得的投资收益”，共有12亿(1,215,319,222.49)。要想和其他科目对应上，还要考虑汇率变动带来的收益等，具体之后再详细研究。

总的来说，上汽集团的投资能力不错。

唐老在书中提到了，通过分析“持有其他上市公司”情况这节，看是不是持有“可供出售金融资产”，可以知道是发现金矿，还是踏入雷区。

这一点在上面已经提到，财报第8页，(四)。招商银行是以“可供出售金融资产”列入的，期末期初差额为9.2亿(920,428,156.35)。

分析这个资产对公司的影响，用期末余额粗略计算，收益率11%左右。还可以。算是金矿。买买买！

3.4 长期股权投资

长期股权投资（Long-term investment on stocks）是指通过投资取得被投资单位的股份。企业对其他单位的股权投资，通常视为长期持有，以及通过股权投资达到控制被投资单位，或对被投资单位施加重大影响，或为了与被投资单位建立密切关系，以分散经营风险。[7]

简单说，就是公司之间相互持股。包括控制、合营、联营多种关系。

这里相关的概念，就是《合并资产负债表》和《母公司资产负债表》。前者是母公司加子公司，所有科目全都合并；后者是母公司独有，子公司的影响仅限于分红。

下面看下全A股范围内的“长期股权投资”。

```
whole_set.sort('长期股权投资', ascending=False).head(20)
```

	名称	期末日期	类型	交易性金融资产	持有至到期投资	可供出售金融资产
ticker						
601857	中国石油	2015-09-30	Q3	0.000000e+00	0.000000e+00	2.040000e+
600028	中国石化	2015-09-30	Q3	0.000000e+00	0.000000e+00	1.281000e+
600104	上汽集团	2015-09-30	Q3	3.239876e+08	0.000000e+00	5.687617e+
601628	中国人寿	2015-09-30	Q3	4.702600e+10	5.051350e+11	7.290770e+
601398	工商银行	2015-09-30	Q3	4.657340e+11	2.815399e+12	1.388838e+
	中					

601668	国建筑	2015-09-30	Q3	5.927100e+08	0.000000e+00	6.000893e+0
000002	万科A	2015-09-30	Q3	0.000000e+00	0.000000e+00	1.854680e+0
601318	中国平安	2015-09-30	Q3	4.988100e+10	8.719430e+11	4.233710e+1
600011	华能国际	2015-09-30	Q3	0.000000e+00	0.000000e+00	5.331335e+0
600018	上港集团	2015-09-30	Q3	0.000000e+00	3.300000e+06	2.408377e+0
601238	广汽集团	2015-09-30	Q3	1.292453e+08	7.982648e+07	1.745842e+0
601006	大秦铁路	2015-09-30	Q3	0.000000e+00	0.000000e+00	6.515870e+0
601988	中国银行	2015-09-30	Q3	1.211650e+11	1.691177e+12	9.805820e+1
600795	国电电力	2015-09-30	Q3	0.000000e+00	0.000000e+00	1.906443e+0
600674	川投能源	2015-09-30	Q3	0.000000e+00	0.000000e+00	1.950043e+0
600023	浙能电力	2015-09-30	Q3	0.000000e+00	0.000000e+00	5.427675e+0

000625	长安汽车	2015-09-30	Q3	0.000000e+00	0.000000e+00	4.100911e+0
601111	中国国航	2015-09-30	Q3	2.413600e+07	8.000000e+07	6.972800e+0
600221	海南航空	2015-09-30	Q3	0.000000e+00	0.000000e+00	3.702220e+0
601991	大唐发电	2015-09-30	Q3	0.000000e+00	0.000000e+00	4.773750e+0

终于不是金三胖了。中国石油排在第一，是1120亿(1.120260e+11)。果然土豪。

我们来看中国石油2015年半年报。第32页，说明了“长期股权投资”的内涵。

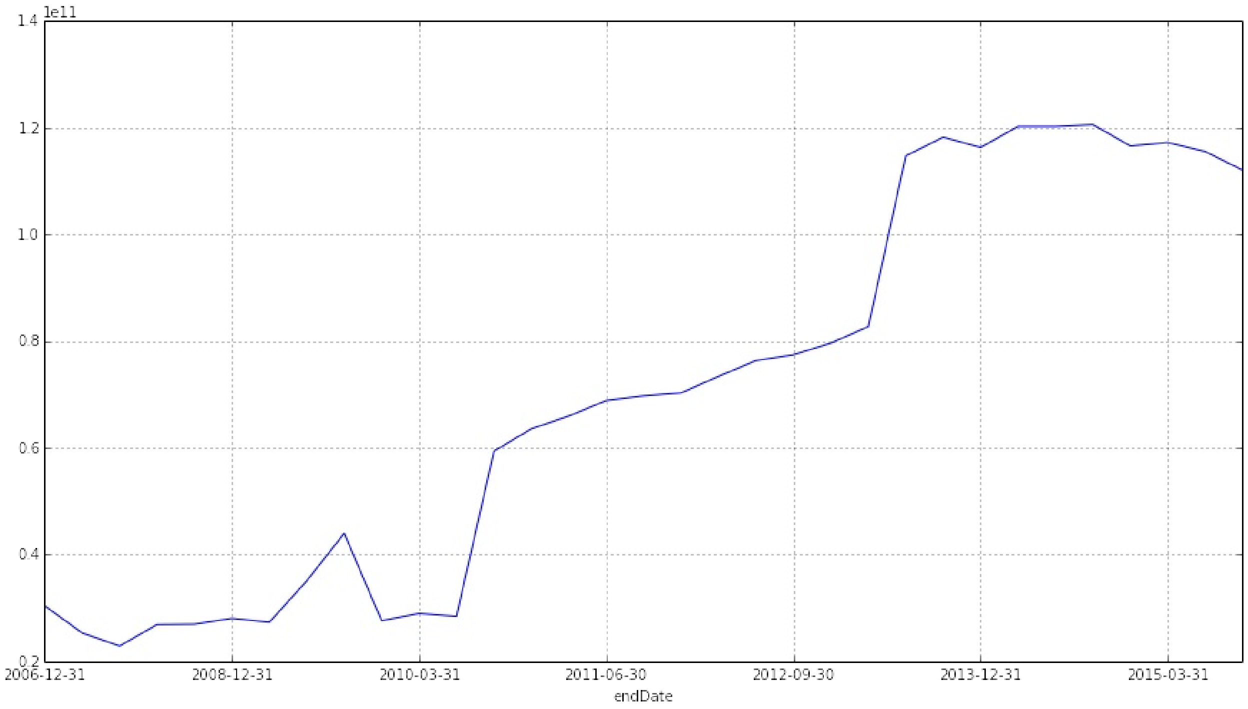
中国石油通过港股二级市场，收购了昆仑能源58.33的股份，账面价值2575.8亿港币(25,758百万元)

除了昆仑能源，还投资了中油财务有限责任公司和中石油专属财产保险股份有限公司。

在第71页，还提到了其他的合营公司，比如注册资本为1澳元的Arrow Energy Holdings Pty Ltd.

单独看下中石油这几年的“长期股权投资”。

```
data = DataAPI.FdmtBSAllLatestGet(ticker=u"601857",secID=u"",reportType=u"",endDate=u"",beginDate=u"",field=u"ticker,secShortName,endDate,reportType,tradingFA,htmInvest,availForSaleFa,LTEquityInvest,purResaleFa,investRealEstate",pandas="1")
data = data.set_index('endDate', inplace=False).sort_index()

data['LTEquityInvest'].plot(figsize=(15,8))

<matplotlib.axes.AxesSubplot at 0x4fa58d0>
```



这么看，从2013年到今天，基本没有再有设立子公司，兼并其他公司股份之类的较大举措。

唐老提到一种财务造假的手段，就是把同一控制公司当成是非同一控制公司进行交易，从而产生“利润”。这种手法不好量化，所以不具体谈了。

3.5 买入返售金融资产

买入返售金融资产是指公司按返售协议约定先买入再按固定价格返售的证券等金融资产所融出的资金。[9]

如果没理解错，应该是回购和逆回购。

买出返售金融资产，主要交易参与者是银行。在中国因为存贷比的约束，很多业务没法做。通过买入返售金融资产可以变相的进行借贷。

下面看看那些企业持有这类资产比较多。

```
whole_set.sort('买入返售金融资产', ascending=False).head(20)
```

	名称	期末日期	类型	交易性金融资产	持有至到期投资	可供出售金融资产
ticker						
601398	工商银行	2015-09-30	Q3	4.657340e+11	2.815399e+12	1.388838e+

601288	农业银行	2015-09-30	Q3	4.127890e+11	2.268457e+12	1.015701e+1
600016	民生银行	2015-09-30	Q3	3.090100e+10	2.339070e+11	1.705880e+1
601166	兴业银行	2015-09-30	Q3	4.174800e+10	2.030120e+11	3.752570e+1
601318	中国平安	2015-09-30	Q3	4.988100e+10	8.719430e+11	4.233710e+1
601939	建设银行	2015-09-30	Q3	1.594500e+11	2.592316e+12	9.462150e+1
600036	招商银行	2015-09-30	Q3	4.819000e+10	3.430530e+11	2.929180e+1
000001	平安银行	2015-09-30	Q3	1.759700e+10	2.600220e+11	7.660000e+0
600015	华夏银行	2015-09-30	Q3	9.595000e+09	1.741240e+11	7.099300e+1
601169	北京银行	2015-09-30	Q3	1.702700e+10	1.468580e+11	1.125920e+1
601328	交通银行	2015-09-30	Q3	1.455020e+11	8.433740e+11	2.532930e+1
	光大	2015-				

	银行	09-30				
600000	浦发银行	2015-09-30	Q3	4.730400e+10	2.215700e+11	2.141730e+1
601998	中信银行	2015-09-30	Q3	2.328700e+10	1.814820e+11	3.600010e+1
600837	海通证券	2015-09-30	Q3	0.000000e+00	8.112264e+07	2.051629e+1
601988	中国银行	2015-09-30	Q3	1.211650e+11	1.691177e+12	9.805820e+1
601211	国泰君安	2015-09-30	Q3	0.000000e+00	0.000000e+00	1.705721e+1
600030	中信证券	2015-09-30	Q3	0.000000e+00	0.000000e+00	7.060994e+1
601009	南京银行	2015-09-30	Q3	2.209831e+10	7.358643e+10	9.241657e+1
002736	国信证券	2015-09-30	Q3	0.000000e+00	0.000000e+00	2.180174e+1

果不其然，主要是银行。具体不作深入研究了

3.5 投资性房地产

投资性房地产，是指为赚取租金或资本增值，或两者兼有而持有的房地产。投资性房地产应当能够单独计量和出售。投资性房地产主要包括：已出租的土地使用权、持有并准备增值后转让的土地使用权和已出租的建筑物。

下列各项不属于投资性房地产的有：（1）自用房地产，即为生产商品、提供劳务或者经营管理而持有的房地产；（2）作为存货的房地产。 投资性房地产属于正常经常性活动，形成的租金收入或转让增值收益确认为企业的主营业务收入但对于大部分企业而言，是与经营性活动相关的其他经营活动。[10]

来看看哪些企业拥有的投资性房地产比较多。

```
whole_set.sort('投资性房地产', ascending=False).head(20)
```

	名称	期末日期	类型	交易性金融资产	持有至到期投资	可供出售金融资产
ticker						
601668	中国建筑	2015-09-30	Q3	5.927100e+08	0.000000e+00	6.000893e+0
601318	中国平安	2015-09-30	Q3	4.988100e+10	8.719430e+11	4.233710e+1
601988	中国银行	2015-09-30	Q3	1.211650e+11	1.691177e+12	9.805820e+1
600823	世茂股份	2015-09-30	Q3	0.000000e+00	0.000000e+00	1.933917e+0
600663	陆家嘴	2015-09-30	Q3	0.000000e+00	0.000000e+00	7.568074e+0
600606	绿地控股	2015-09-30	Q3	2.754451e+09	2.573969e+09	6.689355e+0
000402	金融街	2015-09-30	Q3	0.000000e+00	0.000000e+00	1.375968e+0
601992	金隅股	2015-09-30	Q3	0.000000e+00	0.000000e+00	1.106113e+0

	份					
600383	金地集团	2015-09-30	Q3	0.000000e+00	0.000000e+00	1.519697e+
600221	海南航空	2015-09-30	Q3	0.000000e+00	0.000000e+00	3.702220e+
600648	外高桥	2015-09-30	Q3	8.405400e+02	0.000000e+00	6.644758e+
000002	万科A	2015-09-30	Q3	0.000000e+00	0.000000e+00	1.854680e+
000056	皇庭国际	2015-09-30	Q3	0.000000e+00	0.000000e+00	0.000000e+
600048	保利地产	2015-09-30	Q3	0.000000e+00	0.000000e+00	6.995542e+
601601	中国太保	2015-09-30	Q3	2.160100e+10	3.116450e+11	1.939390e+
600007	中国国贸	2015-09-30	Q3	0.000000e+00	0.000000e+00	0.000000e+
601328	交通银行	2015-09-30	Q3	1.455020e+11	8.433740e+11	2.532930e+
000043	中航地产	2015-09-30	Q3	0.000000e+00	0.000000e+00	0.000000e+
000024	招商	2015-	Q3	0.000000e+00	0.000000e+00	0.000000e+

000024	地产	09-30	Q3	0.000000e+00	0.000000e+00	0.000000e+0
000926	福星股份	2015-09-30	Q3	3.915421e+08	0.000000e+00	4.713240e+0

排名中，不出意外的，以房地产企业为主。当然不排除，有些企业把存货的房地产，转移到了投资性房地产。

福星股份股份上榜。福星股份本来是做"福星牌"钢丝绳的。经营范围是金属丝、绳及其制品的制造、销售和出口业务;商品房销售;高新技术的开发与高新技术项目投资;创业投资、主营业务以外的其他项目投资。但2007年变成了以房地产为主的企业。

过去几年，有太多的企业都变成了“房地产企业”。这架势不把实体经济抽空才怪。

我们来看看他的2015年半年报。[11]

看62页13项，福星股份的投资性房地产主要以公允价值计量。这个和一般的使用成本法不太一样。当然，房价涨这么厉害，公允价值当然大于成本咯。如果想估值，可以适当减少部分。

那福星股份的房地产投资能力如何呢。看第79页第10项，本期的公允价值变动为-3636万(-36,369,051.00)，说明，房地产价值在缩水。

在117页，可以看到投资性房地产进行公允价值计量时的三个层次。福星股份都是以第二、第三层次计量。

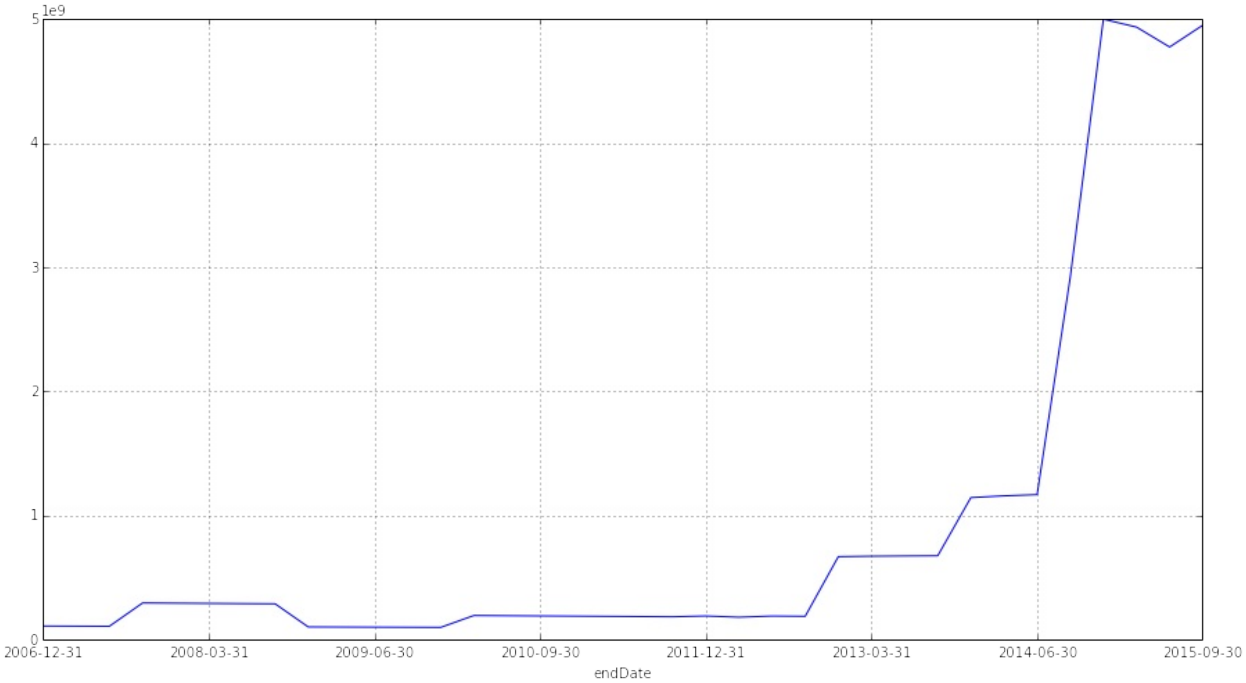
“公允价值在计量时为三个层次。第一层次是企业在计量日能获得相同资产或负债在活跃市场上报价的，以该报价为依据确定公允价值；第二层次是企业在计量日能获得类似资产或负债在活跃市场上的报价，或相同或类似资产或负债在非活跃市场上的报价的，以该报价为依据做必要调整确定公允价值；第三层次是企业无法获得相同或类似资产可比市场交易价格的，以其他反映市场参与者对资产或负债定价时所使用的参数为依据确定公允价值。”[12]

第三层次计量和财报中提到的选择公允价值计量的原因(即有活跃的交易市场)有明显的不一致。

这种计量方式，唐老有特别提到，可能会出现“纸上富贵”。

下图展示了福星股份的历年情况。

```
data = DataAPI.FdmtBSAllLatestGet(ticker=u"000926",secID=u"",rep
ortType=u"",endDate=u"",beginDate=u"",field=u"ticker,secShortNam
e,endDate,reportType,tradingFA,htmInvest,availForSaleFa,LTEquity
Invest,purResaleFa,investRealEstate",pandas="1")
data = data.set_index('endDate', inplace=False).sort_index()

data['investRealEstate'].plot(figsize=(15,8))

<matplotlib.axes.AxesSubplot at 0x3c2c390>
```



在2014年底，有较大的增长。

四、 总结

本文是整个系列的第一篇。以资产负债表中的投资相关资产为主题，逐个科目地将全A股的公司进行排名，并选择个别公司进行深入阅读。

其中涉及公司有

- 工商银行
- 西水股份
- 上汽集团
- 中国石油
- 福星股份

本文以研报阅读学习为主，其中提到的关于上述公司的论述，并没有进行深入准确的考证，从而得到的结论，以调侃为主。并不构成正式结论和投资建议。

其中所有数据都在参考文献中，指明了出处，请读者自行分析。作者才疏学浅，对其中的阅读难免出现理解上的偏差，请读者踊跃提出。

1.4 股东分析

技术分析入门【2】—— 大家抢筹码（06年至12年版）

> 来源：https://uqer.io/community/share/5541d8a4f9f06c1c3d687fef

在本篇中，我们将使用流通股份的集中程度作为指标，为大家开发如何机智的抢筹码策略！

股市里面总是有这样的一种说法： 大股东总是会快小散一步，悄悄地进村，放枪的不要。大股东会在建仓期吸收世面上的廉价筹码，然后放出利好，逢高出货。所以大股东的建仓期，正是小散们入场分汤的好时机！

1. 数据准备

好了，说了这些原理，到底灵不灵呢？来，一试便知！这里我们首先要定义什么叫大股东呢？这里我们借助中诚信的数据，获取前十大流通股东的持股比例：

数据API： `CCXE.EquMainshFCCXEGet` 获取财报中十大流通股股东的持股比例（本API需要在数据商城购买）

下面的语句查询 `600000.XSHG` 浦发银行在2014年9月30日到2014年12月31日的十大流通股股东持股情况：

```
import datetime as dt

data = DataAPI.CCXE.EquMainshFCCXEGet('600000.XSHG', endDateStart='20140930', endDateEnd='20141231')
data.head()
```

	secID	ticker	exchangeCD	secShortName	secShortN
0	600000.XSHG	600000	XSHG	浦发银行	NaN
1	600000.XSHG	600000	XSHG	浦发银行	NaN
2	600000.XSHG	600000	XSHG	浦发银行	NaN
3	600000.XSHG	600000	XSHG	浦发银行	NaN
4	600000.XSHG	600000	XSHG	浦发银行	NaN

我们按照报表日进行合并，并计算前十大流通股股东持股总比例：

```
data.groupby('endDate').sum()
```

	secShortNameEn	shNum	shRank	holdVol	holdPc
endDate					
2014-09-30 00:00:00	NaN	55	55	5550603395	29.76
2014-12-31 00:00:00	NaN	55	55	5455478743	29.25

可以看到，2014年年报中流通股集中度是下降的，相对于上一个季报，持股总比例从29.76%降到了29.25%。看来他的大股东没啥动静，小散们先按兵不动！

2. 策略思路

有一句俗话：不要在一棵树上吊死！小散们可以“海选PK”，择优录取！我们以上证50成分股为例，挑选出满足以下条件的股票：

- 2015年一季度季报中10大流通股股东持股比例相对于去年年报上升10%

这就是我们认定的大股东吸筹码的标志：

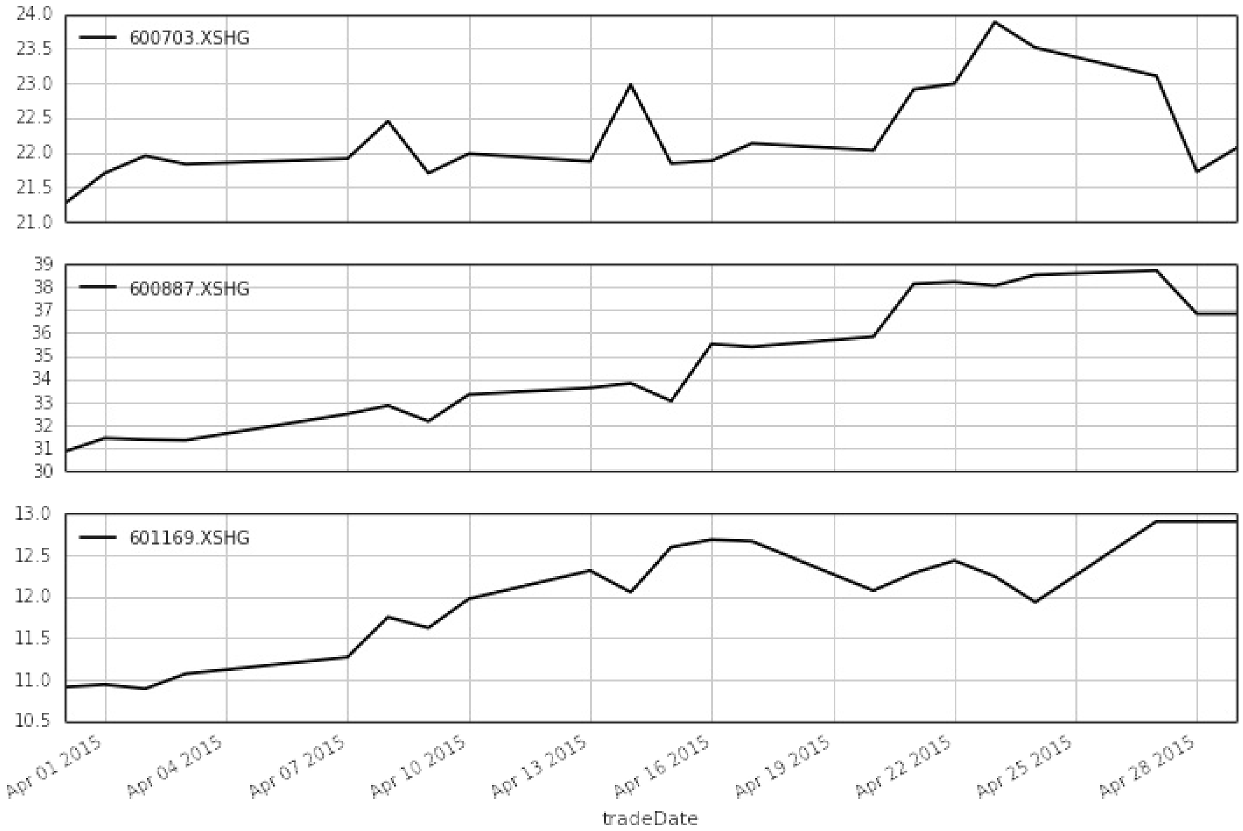
```
from quartz.api import set_universe
import datetime as dt

universe = set_universe('SH50')

for stock in universe:
    try:
        data = DataAPI.CCXE.EquMainshFCCXEGet(stock, endDateStart='20141231', endDateEnd='20150331')
    except:
        continue
    res = data.groupby('endDate').sum()[-2:]
    if len(res.index) == 2 and res.index[1] == '2015-03-31 00:00:00':
        chg = res['holdPct'].values[1] / res['holdPct'].values[0] - 1.0
        if chg > 0.1:
            print '%s: %.4f' % (stock, chg)

601169.XSHG: 0.1236
600887.XSHG: 0.1211
600703.XSHG: 0.1231
```

选出来有三只股票满足： `601169.XSHG` ， `600887.XSHG` ， `600703.XSHG`

下面的股价走势图来看，这样的股票总体还是上升的。但是按照这样投钱真的靠谱吗？

```
import pandas as pd
stock1 = DataAPI.MktEqudAdjGet(['601169.XSHG'], beginDate='20150331', endDate='20150429', field = ['closePrice', 'tradeDate'])
stock2 = DataAPI.MktEqudAdjGet(['600887.XSHG'], beginDate='20150331', endDate='20150429', field = ['closePrice', 'tradeDate'])
stock3 = DataAPI.MktEqudAdjGet(['600703.XSHG'], beginDate='20150331', endDate='20150429', field = ['closePrice', 'tradeDate'])
```

```
import seaborn as sns
sns.set_style('white')

total = pd.DataFrame({'601169.XSHG':stock1.closePrice.values, '6
00887.XSHG':stock2.closePrice.values, '600703.XSHG':stock3.close
Price.values})
total.index = stock1.tradeDate.apply(lambda x: dt.datetime.strpt
ime(x, '%Y-%m-%d'))
total.plot(subplots=True, figsize=(12,8))

array([<matplotlib.axes.AxesSubplot object at 0x53fa0d0>,
       <matplotlib.axes.AxesSubplot object at 0x57ab9d0>,
       <matplotlib.axes.AxesSubplot object at 0x57d0550>], dtype
=object)
```



3. 完整策略

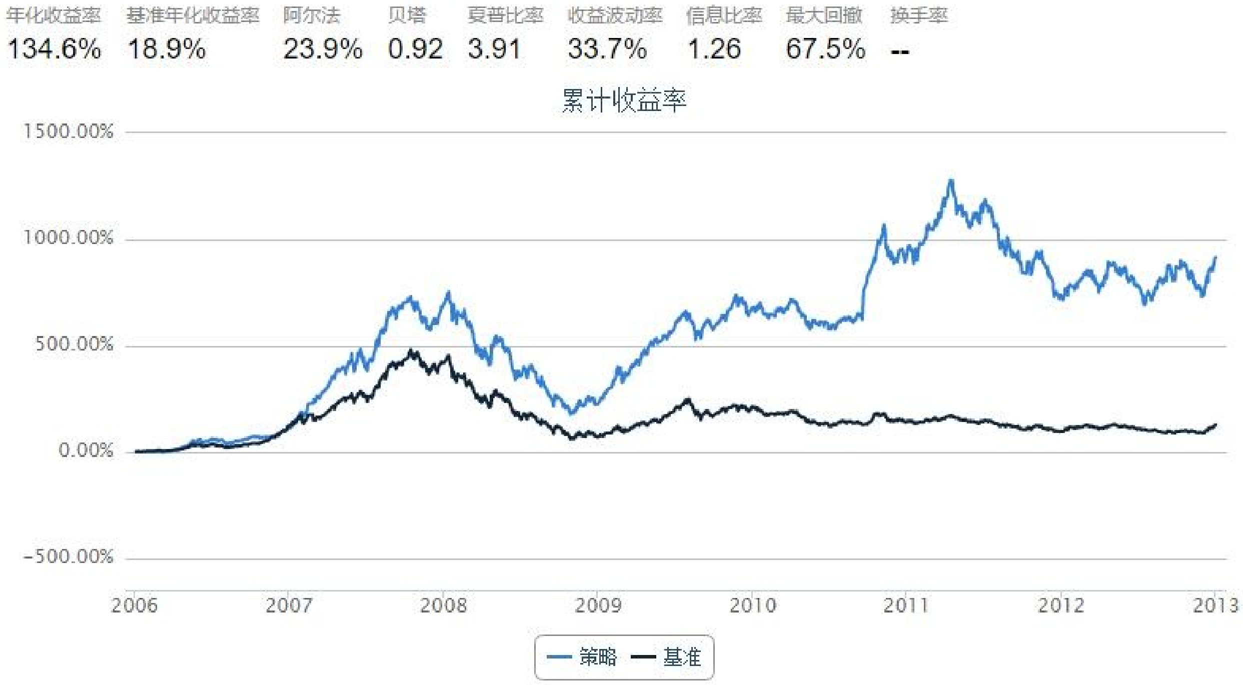
我们来吧上面的想法系统化，来看这个策略效率：

- 投资域 ：上证50成分股
- 业绩基准 ：上证50指数
- 调仓频率 ：3个月
- 调仓日期 ：每年的2月28日，5月31日，8月30日，11月30日，遇到节假日的话向后顺延
- 开仓信号 ：十大流通股股东持股比例集中度上升10%

- 清仓信号 ：每个调仓日前一个工作日，清空当前仓位
- 买入方式 ：等比例买入
- 回测周期 ：2006年1月1日至2015年4月28日

这里的调仓日期的设置，是满足每期报表结束日后的两个月，这样我们有比较大的把握，可以确实拿到当前的报表数据。

```
import datetime as dt

start = '2006-01-01'                       # 回测起始时间
end = '2012-12-31'                        # 回测结束时间
benchmark = 'SH50'                        # 策略参考标准
universe = set_universe('SH50')               # 证券池，支持股票和
基金
capital_base = 100000                      # 起始资金
longest_history = 1                       # handle_data 函数中可以
使用的历史数据最长窗口长度
refresh_rate = 1                         # 调仓频率，即每 refresh_
rate 个交易日执行一次 handle_data() 函数

def initialize(account):                   # 初始化虚拟账户状态
    account.reportingPair = [('0930', '1231'), ('1231', '0331'),
 ('0331', '0630'), ('0630', '0930')]

def handle_data(account):            # 每个交易日的买入卖出指令
    hist = account.get_history(longest_history)
    today = account.current_date
    year = today.year
    rebalance_dates = [dt.datetime(year, 2, 28), dt.datetime(yea
r, 5,31), dt.datetime(year, 8, 30), dt.datetime(year, 11,30)]
    cal = Calendar('China.SSE')
    rebalance_dates = [cal.adjustDate(d, BizDayConvention.Follow
ing) for d in rebalance_dates]

    rebalanceFlag = False
    period = -1
    for i, d in enumerate(rebalance_dates):
        # 判断是否是调仓日
        if today == d.toDateTime():
            rebalanceFlag = True
            period = i
            break
        # 调仓日前一个交易日，清空所有的仓位
        elif today == cal.advanceDate(d, '-1B').toDateTime():
            for stock in account.valid_secpos:
                order_to(stock,0)


    if rebalanceFlag:
        if period == 0:
            year -= 1
        # 确定当前调仓日对应需要查询的报表日期
        if account.reportingPair[period][0] < account.reportingPair[period][1]:
            endDateStart = str(year) + account.reportingPair[period][0]
        else:
            endDateStart = str(year-1) + account.reportingPair[period][0]
        endDateEnd = str(year) + account.reportingPair[period][1]

        buyList = []
        # 确定哪些股票满足“筹码”集中要求
        for stock in account.universe:
            try:
                data = DataAPI.CCXE.EquMainshFCCXEGet(stock, endDateStart=endDateStart, endDateEnd=endDateEnd)
            except:
                continue
            res = data.groupby('endDate').sum()[-2:]
            tmp = account.reportingPair[period][1]
            if len(res.index) == 2 and res.index[1] == str(year) + '-' + tmp[:2] + '-' + tmp[2:]+ ' 00:00:00':
                chg = res['holdPct'].values[1] / res['holdPct'].values[0] - 1.0
                if chg > 0.1:
                    buyList.append(stock)


        print u"%s 买入 : %s" % (today, buyList)

        # 等权重买入
        if len(buyList) != 0:
            singleCash = account.cash / len(buyList)
            for stock in buyList:
                approximationAmount = int(singleCash / hist[stock]['closePrice'][-1]/100.0) * 100
                order(stock, approximationAmount)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
134.6%	18.9%	23.9%	0.92	3.91	33.7%	1.26	67.5%	--

```
2006-02-28 00:00:00 买入 : ['600050.XSHG', '600016.XSHG', '600104
.XSHG', '600010.XSHG', '600518.XSHG', '600030.XSHG', '600150.XSH
G']
2006-05-31 00:00:00 买入 : ['600036.XSHG', '600111.XSHG', '600089
.XSHG', '600690.XSHG', '600104.XSHG', '600010.XSHG', '600030.XSH
G']
2006-08-30 00:00:00 买入 : ['600050.XSHG', '600196.XSHG', '600000
.XSHG', '600703.XSHG', '600089.XSHG', '600104.XSHG', '600637.XSH
G', '600837.XSHG', '600150.XSHG']
2006-11-30 00:00:00 买入 : ['600050.XSHG', '600036.XSHG', '600000
.XSHG', '600111.XSHG', '600372.XSHG', '600519.XSHG', '600016.XSH
G', '600703.XSHG', '600690.XSHG', '600518.XSHG', '600030.XSHG',
'600832.XSHG']
2007-02-28 00:00:00 买入 : ['600196.XSHG', '600000.XSHG', '600111
.XSHG', '601006.XSHG', '600406.XSHG', '600690.XSHG', '600048.XSH
G', '600015.XSHG', '600518.XSHG', '600887.XSHG', '600150.XSHG']
2007-05-31 00:00:00 买入 : ['600111.XSHG', '600256.XSHG', '601166
.XSHG', '600104.XSHG', '600015.XSHG', '600637.XSHG', '600837.XSH
G']
2007-08-30 00:00:00 买入 : ['600000.XSHG', '600372.XSHG', '600519
.XSHG', '600256.XSHG', '600690.XSHG', '600332.XSHG', '601166.XSH
G', '600015.XSHG', '600109.XSHG', '600887.XSHG', '601318.XSHG']
2007-11-30 00:00:00 买入 : ['600050.XSHG', '600196.XSHG', '600111
.XSHG', '600372.XSHG', '601006.XSHG', '600256.XSHG', '600406.XSH
G', '600048.XSHG', '600104.XSHG', '600015.XSHG', '600837.XSHG',
'600030.XSHG', '600832.XSHG']
2008-02-28 00:00:00 买入 : ['601328.XSHG', '600050.XSHG', '600196
.XSHG', '600000.XSHG', '600018.XSHG', '600016.XSHG', '601006.XSH
G', '600406.XSHG', '600104.XSHG', '600028.XSHG', '600518.XSHG',
'600837.XSHG', '601169.XSHG', '601398.XSHG']
2008-06-02 00:00:00 买入 : ['600196.XSHG', '601006.XSHG', '600690
.XSHG', '601166.XSHG', '600010.XSHG', '600518.XSHG', '601318.XSH
G']
2008-09-01 00:00:00 买入 : ['601328.XSHG', '600050.XSHG', '600196
.XSHG', '601601.XSHG', '600036.XSHG', '600000.XSHG', '600519.XSH
G', '600016.XSHG', '600089.XSHG', '600256.XSHG', '600332.XSHG',
'600015.XSHG', '601998.XSHG', '600637.XSHG', '600150.XSHG']
2008-12-01 00:00:00 买入 : ['601601.XSHG', '600372.XSHG', '600703
.XSHG', '600690.XSHG', '600104.XSHG', '600837.XSHG', '601169.XSH
G', '600030.XSHG', '600832.XSHG']
2009-03-02 00:00:00 买入 : ['601601.XSHG', '600372.XSHG', '600406
.XSHG', '600104.XSHG', '600028.XSHG', '600518.XSHG', '600887.XSH
G', '600837.XSHG']
2009-06-01 00:00:00 买入 : ['600036.XSHG', '600111.XSHG', '600703
.XSHG', '600585.XSHG', '600048.XSHG', '600109.XSHG', '600887.XSH
G']
2009-08-31 00:00:00 买入 : ['600050.XSHG', '600196.XSHG', '600000
.XSHG', '600111.XSHG', '600519.XSHG', '600703.XSHG', '600089.XSH
G', '600256.XSHG', '600332.XSHG', '600015.XSHG', '600010.XSHG',
'600887.XSHG', '601766.XSHG', '601398.XSHG', '600150.XSHG']
2009-11-30 00:00:00 买入 : ['600016.XSHG', '601006.XSHG', '600048
.XSHG', '600887.XSHG']
2010-03-01 00:00:00 买入 : ['601601.XSHG', '600018.XSHG', '600016
.XSHG', '601668.XSHG', '600585.XSHG', '600406.XSHG', '600104.XSH
G', '601998.XSHG', '600028.XSHG', '601398.XSHG']
2010-05-31 00:00:00 买入 : ['601299.XSHG', '600111.XSHG', '600256
.XSHG', '600999.XSHG', '601628.XSHG', '601318.XSHG']
2010-08-30 00:00:00 买入 : ['601328.XSHG', '600196.XSHG', '601299
.XSHG', '600111.XSHG', '600585.XSHG', '601688.XSHG', '601998.XSH
G', '600999.XSHG', '600109.XSHG', '601989.XSHG', '600837.XSHG']
2010-11-30 00:00:00 买入 : ['600372.XSHG', '600703.XSHG', '600010
.XSHG', '601989.XSHG', '601169.XSHG', '600150.XSHG']
2011-02-28 00:00:00 买入 : ['601601.XSHG', '601857.XSHG', '601299
.XSHG', '600372.XSHG', '601288.XSHG', '601668.XSHG', '601088.XSH
G', '600256.XSHG', '600999.XSHG', '601989.XSHG', '600837.XSHG']
2011-05-31 00:00:00 买入 : ['601118.XSHG', '601668.XSHG', '601688
.XSHG', '600010.XSHG', '600109.XSHG']
2011-08-30 00:00:00 买入 : ['600196.XSHG', '601299.XSHG', '601118
.XSHG', '600690.XSHG', '600010.XSHG', '600887.XSHG']
2011-11-30 00:00:00 买入 : ['601299.XSHG', '600372.XSHG', '601118
.XSHG', '600703.XSHG', '601288.XSHG', '601818.XSHG', '601766.XSH
G']
2012-02-28 00:00:00 买入 : ['600015.XSHG', '600030.XSHG', '601901
.XSHG']
2012-05-31 00:00:00 买入 : ['600372.XSHG', '601989.XSHG']
2012-08-30 00:00:00 买入 : ['601118.XSHG', '600837.XSHG', '601901
.XSHG']
2012-11-30 00:00:00 买入 : ['601118.XSHG', '601668.XSHG', '601901
.XSHG']
```

技术分析入门【2】——大家抢筹码（06年至12年版）— 更新版

> 来源：https://uqer.io/community/share/568e6f54228e5b18e5ba296e

从社区李大大以前的帖子，稍作修改，适合现在的uqer版本，感谢李大大的无私分享！

原帖地址：

https://uqer.io/community/share/5541d8a4f9f06c1c3d687fef

在本篇中，我们将使用流通股份的集中程度作为指标，为大家开发如何机智的抢筹码策略！

股市里面总是有这样的一种说法：大股东总是会快小散一步，悄悄地进村，放枪的不要。大股东会在建仓期吸收世面上的廉价筹码，然后放出利好，逢高出货。所以大股东的建仓期，正是小散们入场分汤的好时机！

1. 数据准备

好了，说了这些原理，到底灵不灵呢？来，一试便知！这里我们首先要定义什么叫大股东呢？这里我们借助中诚信的数据，获取前十大流通股东的持股比例：

数据API：`CCXE.EquMainshFCCXEGet` 获取财报中十大流通股股东的持股比例（本API需要在数据商城购买）

下面的语句查询`600000.XSHG`浦发银行在2014年9月30日到2014年12月31日的十大流通股股东持股情况：

```
import datetime as dt
from CAL.PyCAL import *

data = DataAPI.CCXE.EquMainshFCCXEGet('600000.XSHG', endDateStar
t='20140930', endDateEnd='20141231')
data.head()
```

	secID	ticker	exchangeCD	secShortName	secShortN
0	600000.XSHG	600000	XSHG	浦发银行	NaN
1	600000.XSHG	600000	XSHG	浦发银行	NaN
2	600000.XSHG	600000	XSHG	浦发银行	NaN
3	600000.XSHG	600000	XSHG	浦发银行	NaN
4	600000.XSHG	600000	XSHG	浦发银行	NaN

我们按照报表日进行合并，并计算前十大流通股股东持股总比例：

```
data.groupby('endDate').sum()
```

可以看到，2014年年报中流通股集中度是下降的，相对于上一个季报，持股总比例从29.76%降到了29.25%。看来他的大股东没啥动静，小散们先按兵不动！

2. 策略思路

有一句俗话：不要在一棵树上吊死！小散们可以“海选PK”，择优录取！我们以上证50成分股为例，挑选出满足以下条件的股票：

- 2015年一季度季报中10大流通股股东持股比例相对于去年年报上升10%

这就是我们认定的大股东吸筹码的标志：

```
from quartz.api import set_universe
import datetime as dt

universe = set_universe('SH50')

for stock in universe:
    try:
        data = DataAPI.CCXE.EquMainshFCCXEGet(stock, endDateStar
t='20141231', endDateEnd='20150331')
    except:
        continue
    res = data.groupby('endDate').sum()[-2:]
    if len(res.index) == 2 and res.index[1] == '2015-03-31 00:00
:00':
        chg = res['holdPct'].values[1] / res['holdPct'].values[0
] - 1.0
        if chg > 0.1:
            print '%s: %.4f' % (stock, chg)
```

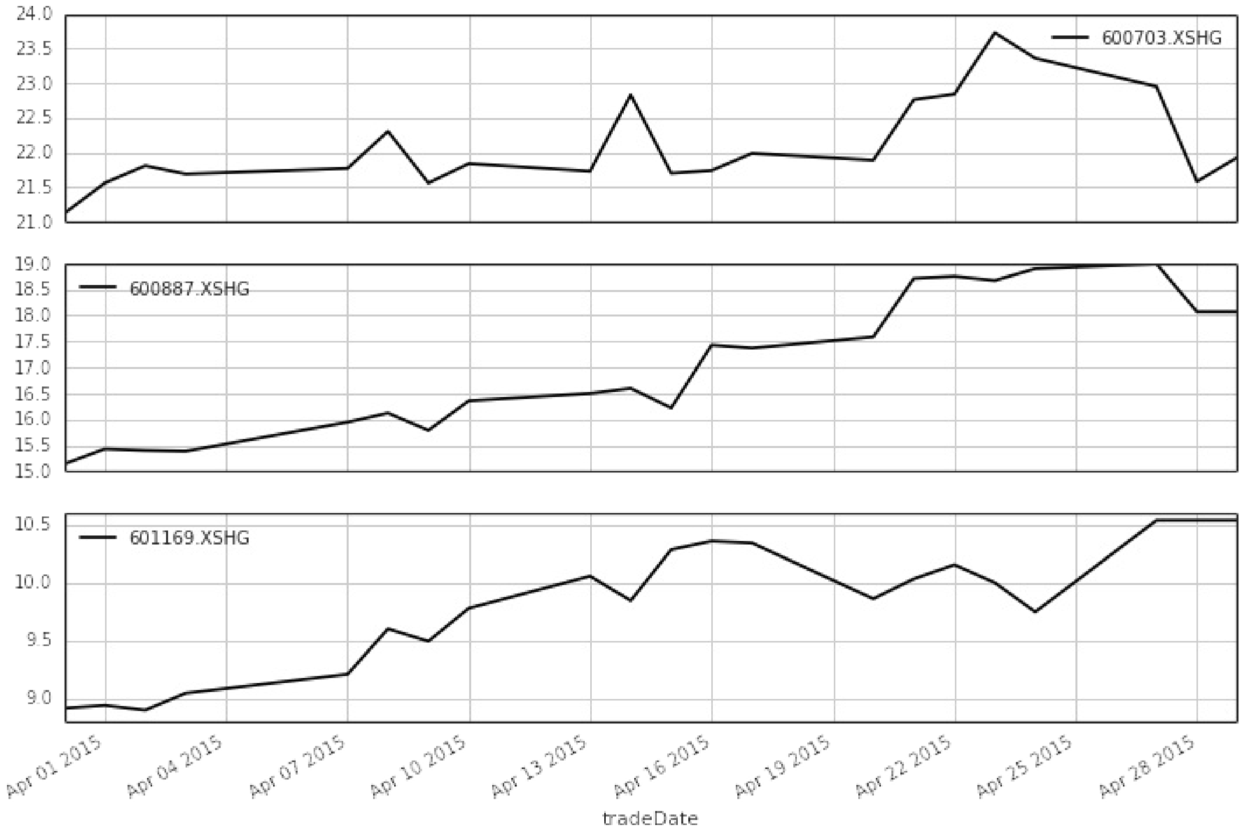
选出来有三只股票满足： `601169.XSHG` , `600887.XSHG` , `600703.XSHG`

下面的股价走势图来看，这样的股票总体还是上升的。但是按照这样投钱真的靠谱吗？

```
import pandas as pd
stock1 = DataAPI.MktEqudAdjGet(secID=['601169.XSHG'], beginDate=
'20150331', endDate='20150429', field = ['closePrice', 'tradeDat
e'])
stock2 = DataAPI.MktEqudAdjGet(secID=['600887.XSHG'], beginDate=
'20150331', endDate='20150429', field = ['closePrice', 'tradeDat
e'])
stock3 = DataAPI.MktEqudAdjGet(secID=['600703.XSHG'], beginDate=
'20150331', endDate='20150429', field = ['closePrice', 'tradeDat
e'])
```

```
import seaborn as sns
sns.set_style('white')

total = pd.DataFrame({'601169.XSHG':stock1.closePrice.values, '600887.XSHG':stock2.closePrice.values, '600703.XSHG':stock3.closePrice.values})
total.index = stock1.tradeDate.apply(lambda x: dt.datetime.strptime(x, '%Y-%m-%d'))
total.plot(subplots=True, figsize=(12,8))
```

```
array([<matplotlib.axes.AxesSubplot object at 0x5543d10>,
       <matplotlib.axes.AxesSubplot object at 0x5572850>,
       <matplotlib.axes.AxesSubplot object at 0x56a62d0>], dtype=object)
```



3. 完整策略

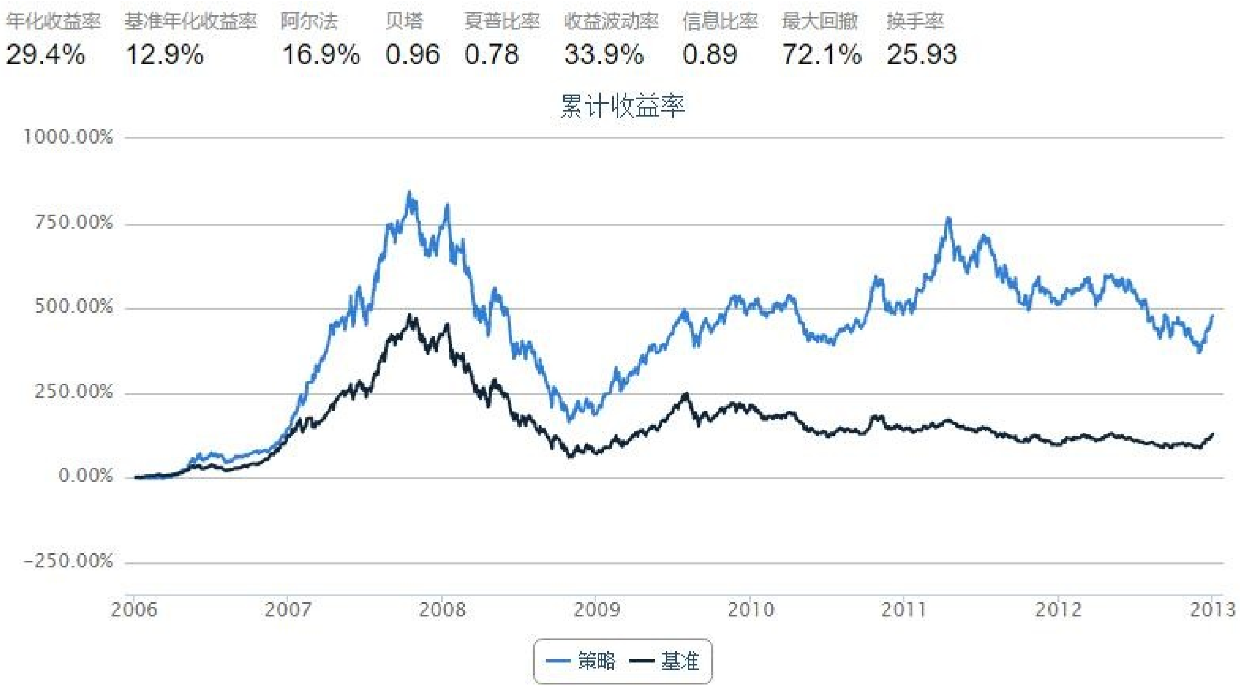
我们来吧上面的想法系统化，来看这个策略效率：

- 投资域 ：上证50成分股
- 业绩基准 ：上证50指数
- 调仓频率 ：3个月
- 调仓日期 ：每年的2月28日，5月31日，8月30日，11月30日，遇到节假日的话向后顺延
- 开仓信号 ：十大流通股股东持股比例集中度上升10%

- 清仓信号 ：每个调仓日前一个工作日，清空当前仓位
- 买入方式 ：等比例买入
- 回测周期 ：2006年1月1日至2015年4月28日

这里的调仓日期的设置，是满足每期报表结束日后的两个月，这样我们有比较大的把握，可以确实拿到当前的报表数据。

```
import datetime as dt

start = '2006-01-01'                           # 回测起始时间
end = '2012-12-31'                            # 回测结束时间
benchmark = 'SH50'                            # 策略参考标准
universe = set_universe('SH50')                   # 证券池，支持股票和
基金
capital_base = 100000                          # 起始资金
longest_history = 1                           # handle_data 函数中可以
使用的历史数据最长窗口长度
refresh_rate = 1                             # 调仓频率，即每 refresh_
rate 个交易日执行一次 handle_data() 函数

def initialize(account):                       # 初始化虚拟账户状态
    account.reportingPair = [('0930', '1231'), ('1231', '0331'),
 ('0331', '0630'), ('0630', '0930')]

def handle_data(account):                # 每个交易日的买入卖出指令
    hist = account.get_history(longest_history)
    today = account.current_date
    year = today.year
    rebalance_dates = [dt.datetime(year, 2, 28), dt.datetime(yea
r, 5,31), dt.datetime(year, 8, 30), dt.datetime(year, 11,30)]
    cal = Calendar('China.SSE')
    rebalance_dates = [cal.adjustDate(d, BizDayConvention.Follow
ing) for d in rebalance_dates]

    rebalanceFlag = False
    period = -1
    for i, d in enumerate(rebalance_dates):
        # 判断是否是调仓日
        if today == d.toDateTime():
            rebalanceFlag = True
            period = i
            break
        # 调仓日前一个交易日，清空所有的仓位
        elif today == cal.advanceDate(d, '-1B').toDateTime():
            for stock in account.valid_secpos:
                order_to(stock,0)


    if rebalanceFlag:
        if period == 0:
            year -= 1
        # 确定当前调仓日对应需要查询的报表日期
        if account.reportingPair[period][0] < account.reportingPair[period][1]:
            endDateStart = str(year) + account.reportingPair[period][0]
        else:
            endDateStart = str(year-1) + account.reportingPair[period][0]
        endDateEnd = str(year) + account.reportingPair[period][1]

        buyList = []
        # 确定哪些股票满足“筹码”集中要求
        for stock in account.universe:
            try:
                data = DataAPI.CCXE.EquMainshFCCXEGet(stock, endDateStart=endDateStart, endDateEnd=endDateEnd)
            except:
                continue
            res = data.groupby('endDate').sum()[-2:]
            tmp = account.reportingPair[period][1]
            if len(res.index) == 2 and res.index[1] == str(year) + '-' + tmp[:2] + '-' + tmp[2:]+ ' 00:00:00':
                chg = res['holdPct'].values[1] / res['holdPct'].values[0] - 1.0
                if chg > 0.1:
                    buyList.append(stock)


        print u"%s 买入 : %s" % (today, buyList)

        # 等权重买入
        if len(buyList) != 0:
            singleCash = account.cash / len(buyList)
            for stock in buyList:
                approximationAmount = int(singleCash / hist[stock]['closePrice'][-1]/100.0) * 100
                order(stock, approximationAmount)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
29.4%	12.9%	16.9%	0.96	0.78	33.9%	0.89	72.1%	25.93

```
2006-02-28 00:00:00 买入 : ['600050.XSHG', '600893.XSHG', '600016
.XSHG', '600104.XSHG', '600010.XSHG', '600518.XSHG', '600030.XSH
G', '600150.XSHG']
2006-05-31 00:00:00 买入 : ['600036.XSHG', '600111.XSHG', '600104
.XSHG', '600010.XSHG', '600030.XSHG']
2006-08-30 00:00:00 买入 : ['600050.XSHG', '600893.XSHG', '600000
.XSHG', '600104.XSHG', '600637.XSHG', '600837.XSHG', '600150.XSH
G']
2006-11-30 00:00:00 买入 : ['600050.XSHG', '600795.XSHG', '600036
.XSHG', '600000.XSHG', '600111.XSHG', '600519.XSHG', '600016.XSH
G', '600518.XSHG', '601988.XSHG', '600030.XSHG']
2007-02-28 00:00:00 买入 : ['600000.XSHG', '600111.XSHG', '601006
.XSHG', '600048.XSHG', '600015.XSHG', '600518.XSHG', '600887.XSH
G', '600150.XSHG']
2007-05-31 00:00:00 买入 : ['600795.XSHG', '600111.XSHG', '601166
.XSHG', '600104.XSHG', '600015.XSHG', '600637.XSHG', '600837.XSH
G']
2007-08-30 00:00:00 买入 : ['600000.XSHG', '600519.XSHG', '601166
.XSHG', '600015.XSHG', '600109.XSHG', '600887.XSHG', '601318.XSH
G']
2007-11-30 00:00:00 买入 : ['600050.XSHG', '600795.XSHG', '600111
.XSHG', '601006.XSHG', '600048.XSHG', '600104.XSHG', '600015.XSH
G', '600837.XSHG', '601988.XSHG', '600030.XSHG']
2008-02-28 00:00:00 买入 : ['601328.XSHG', '600050.XSHG', '600795
.XSHG', '600000.XSHG', '600018.XSHG', '600016.XSHG', '601006.XSH
G', '600104.XSHG', '600028.XSHG', '600518.XSHG', '600837.XSHG',
'601169.XSHG', '601988.XSHG', '601398.XSHG']
2008-06-02 00:00:00 买入 : ['601006.XSHG', '601166.XSHG', '600010
.XSHG', '600518.XSHG', '601318.XSHG']
2008-09-01 00:00:00 买入 : ['601328.XSHG', '600050.XSHG', '601601
.XSHG', '600036.XSHG', '600000.XSHG', '600519.XSHG', '600016.XSH
G', '601998.XSHG', '600015.XSHG', '600637.XSHG', '600150.XSHG']
```

```
2008-12-01 00:00:00 买入 : ['601601.XSHG', '600795.XSHG', '600104
.XSHG', '600837.XSHG', '601169.XSHG', '600030.XSHG']
2009-03-02 00:00:00 买入 : ['601601.XSHG', '601390.XSHG', '600104
.XSHG', '600028.XSHG', '600518.XSHG', '600887.XSHG', '600837.XSH
G', '601988.XSHG']
2009-06-01 00:00:00 买入 : ['600893.XSHG', '600036.XSHG', '600111
.XSHG', '600585.XSHG', '600048.XSHG', '600109.XSHG', '600887.XSH
G', '601988.XSHG']
2009-08-31 00:00:00 买入 : ['600050.XSHG', '600893.XSHG', '600000
.XSHG', '600111.XSHG', '600519.XSHG', '600015.XSHG', '600010.XSH
G', '600887.XSHG', '601766.XSHG', '601398.XSHG', '600150.XSHG']
2009-11-30 00:00:00 买入 : ['600795.XSHG', '600893.XSHG', '600016
.XSHG', '601006.XSHG', '600048.XSHG', '600887.XSHG', '601988.XSH
G']
2010-03-01 00:00:00 买入 : ['601601.XSHG', '600893.XSHG', '600018
.XSHG', '600016.XSHG', '601668.XSHG', '600585.XSHG', '601998.XSH
G', '600104.XSHG', '600028.XSHG', '601398.XSHG']
2010-05-31 00:00:00 买入 : ['600111.XSHG', '600999.XSHG', '601628
.XSHG', '601318.XSHG']
2010-08-30 00:00:00 买入 : ['601328.XSHG', '600893.XSHG', '600111
.XSHG', '600585.XSHG', '601998.XSHG', '601688.XSHG', '600999.XSH
G', '600109.XSHG', '601989.XSHG', '600837.XSHG']
2010-11-30 00:00:00 买入 : ['600010.XSHG', '601989.XSHG', '601169
.XSHG', '600150.XSHG']
2011-02-28 00:00:00 买入 : ['601601.XSHG', '601857.XSHG', '601390
.XSHG', '601288.XSHG', '601668.XSHG', '601088.XSHG', '600999.XSH
G', '601989.XSHG', '600837.XSHG']
2011-05-31 00:00:00 买入 : ['600893.XSHG', '601668.XSHG', '601688
.XSHG', '600010.XSHG', '600109.XSHG']
2011-08-30 00:00:00 买入 : ['600010.XSHG', '600887.XSHG']
2011-11-30 00:00:00 买入 : ['601288.XSHG', '601818.XSHG', '601766
.XSHG']
2012-02-28 00:00:00 买入 : ['600893.XSHG', '600015.XSHG', '600030
.XSHG', '601669.XSHG', '601901.XSHG']
2012-05-31 00:00:00 买入 : ['601336.XSHG', '601989.XSHG', '601669
.XSHG']
2012-08-30 00:00:00 买入 : ['601336.XSHG', '600837.XSHG', '601901
.XSHG']
2012-11-30 00:00:00 买入 : ['601668.XSHG', '601901.XSHG']
```

谁是中国A股最有钱的自然人

> 来源：https://uqer.io/community/share/5523b45ef9f06c8f3390453e

运行此代码，便可知道，在当前日期，谁是A股最有钱的自然人！

股东数据来自于恒生聚源，选择的是类型为“自然人”的股东，有可能有脏数据！

```
import pandas as pd
import numpy as np
from datetime import datetime,timedelta
from CAL.PyCAL import *
cal = Calendar('China.SSE')
```

```
def GetSecID(tk_list,**kargs):    #获得partyID
    num = 100
    cnt_num = len(tk_list)/num
    if cnt_num > 0:
        df = pd.DataFrame({})
        for i in range(cnt_num):
            sub_df = DataAPI.SecIDGet(ticker=tk_list[i*num:(i+1)
*num],**kargs)
            df = pd.concat([df,sub_df])
        if (i+1)*num != len(tk_list):
            sub_df = DataAPI.SecIDGet(ticker=tk_list[(i+1)*num:]
,**kargs)
            df = pd.concat([df,sub_df])
    else:
        df = DataAPI.SecIDGet(ticker=tk_list,**kargs)
    return df

def CountTime():    #获取最近的一个交易日，返回的是datetime格式
    today = datetime.today()
    today_str = today.strftime("%Y%m%d")
    cal_date = Date.fromDateTime(today)
    time1=" 15:05:00"
    ben_time = datetime.strptime(today_str+time1,"%Y%m%d %H:%M:%
S")
    if cal.isBizDay(cal_date) & (today>ben_time):    #如果是交易日
，则判断当天是不是在15点前
        date = today
    else:    #如果当天不是交易日，则获得前一个交易日
        cal_wd = cal.advanceDate(cal_date, '-1B', BizDayConventi
on.Following)    #Date格式
        date = cal_wd.toDateTime()    #datetime格式
    return date
def GetMktEqud(tk_list,**kargs):    #获得最新市场信息快照，即最新价格
信息
```

```
    num = 50
    cnt_num = len(tk_list)/num
    if cnt_num > 0:
        df = pd.DataFrame({})
        for i in range(cnt_num):
            sub_df = DataAPI.MktEqudGet(ticker=tk_list[i*num:(i+1
)*num],**kargs)
            df = pd.concat([df,sub_df])
        if (i+1)*num != len(tk_list):
            sub_df = DataAPI.MktEqudGet(ticker=tk_list[(i+1)*num
:],**kargs)
            df = pd.concat([df,sub_df])
    else:
        df = DataAPI.MktEqudGet(ticker=tk_list,**kargs)
    return df

def add_nm_money(sub_info):    #将个股名称与金额拼接，方便做展示
    add_info_list = []
    sub_info_1 = sub_info.sort(columns='hold_money',ascending=Fa
lse)
    for i in range(len(sub_info_1)):
        add_info = sub_info_1['secshortNM'].iloc[i] + str(round(
sub_info_1['hold_money'].iloc[i]/1e8,2))+'亿'
        add_info_list.append(add_info)
    return add_info_list
```

```
#获得全A股的partyID

universe = DataAPI.EquGet(equTypeCD='A')['secID'].tolist()    #
获得全A股的secID
All_A_tks_list = map(lambda x:x[0:6],universe)                #
根据色此ID获得A股的所有ticker，因为要获得partyID需要输入ticker
party_id_info = GetSecID(tk_list=All_A_tks_list,field=['secShort
Name','ticker','partyID'])    #由ticker获得该个股的partyID
party2tk_dic = dict(zip(party_id_info['partyID'],party_id_info['
ticker']))        #获得partyID与ticker的对应字典；注意，party_id_i
nfo的partyID是int型
party2nm_dic = dict(zip(party_id_info['partyID'],party_id_info['
secShortName']))    #获得partyID与secShortName的对应字典
party_id_list = map(lambda x:str(x),party_id_info['partyID'].tol
ist())     #获得partyID的list，返回的party_id_info的‘partyID’是int
型，而EquMainshJYGet输入的partyID需要str型，所以这里做个转换
field1 = ['partyID','publishDate','shName','shChar','holdVol']
  #分别对应的是[公司代码，公告日、信息类别、股东名称、股东性质、持股数]
```

```
#获得所有个股的自然人股东姓名，以及持有的股票数目
All_info_df = pd.DataFrame({})
for party_id in party_id_list:
    hold_info = DataAPI.JY.EquMainshJYGet(partyID=party_id,field=field1)
    last_publishDate = hold_info['publishDate'].iloc[-1]
    hold_info = hold_info[(hold_info['publishDate']==last_publishDate)&(hold_info['shChar']=='自然人')]     #获取最新的自然人股东信息
    hold_info_gp = hold_info.groupby('shName')
    #由于EquMainshJYGet这个API获得的是十大股东和十大流通股，只要出现过，不管是哪种都要记录；有可能出现两次，也只记录一次。
    for nm,sub_info in hold_info_gp:
        if len(sub_info)>1:    #既是十大股东之一也是十大流通股东之一，只记录其中之一
            All_info_df = pd.concat([All_info_df,sub_info[0:1]])
        else:    #是十大股东或十大流通股东，记录下来
            All_info_df = pd.concat([All_info_df,sub_info])
All_info_df['ticker'] = All_info_df['partyID'].apply(lambda x:party2tk_dic[x])     #获得partyID对应的ticker
```

```
#获得个股的行情数据
tklist_1 = All_info_df['ticker'].tolist()     #获得有自然人持股的个股ticker
tklist_1 = list(set(tklist_1))     #去重
endDate = CountTime().strftime('%Y%m%d')     #获得最近一个交易日的日期

Mkt_info = GetMktEqud(tklist_1,beginDate=endDate,endDate=endDate,field = ['ticker','closePrice'])     #获取最近一个交易日的行情数据
```

```
tk2price = dict(zip(Mkt_info['ticker'],Mkt_info['closePrice']))   #获得个股ticker与价格的字典
All_info_df['closePrice'] = All_info_df['ticker'].apply(lambda x:tk2price[x])     #添加closePrice到All_info_df中
All_info_df['secshortNM'] = All_info_df['partyID'].apply(lambda x:party2nm_dic[x])     #添加secshortNM即个股的简称到All_info_df中
All_info_df['hold_money'] = All_info_df['holdVol']*All_info_df['closePrice']     #添加hold_money即个股的持有金额到All_info_df中
All_info_df_gp = All_info_df.groupby('shName')     #根据股东的姓名来分类
```

```
#统计自然人股东的总资产，并按照总资产大小由大到小排序
final_info_dic = {'name':[],'total_money':[],'stk_money':[]}
for personNM,sub_info in All_info_df_gp:
    total_money = sub_info['hold_money'].sum()
    stk_money = add_nm_money(sub_info)     #获得个股名称和金额的拼接结果
    final_info_dic['name'].append(personNM)
    final_info_dic['total_money'].append(total_money)
    final_info_dic['stk_money'].append(stk_money)
final_info_df = pd.DataFrame(final_info_dic)
```

```
#为了展示，做一些处理
All_info_df_sort = final_info_df.sort(columns='total_money',ascending=False).reset_index(drop=True)
All_info_df_sort['total_money'] = np.round(All_info_df_sort['total_money']/1e8,2).astype(str)+'亿'
All_info_df_sort.columns = ['自然人名称','持有的股票及资产','总资产']
print '谁是A股最有钱的自然人股东？（附注：’恒生聚源‘的数据库显示为自然人，则该股东定为自然人股东，可能存在脏数据）'
All_info_df_sort
```

谁是A股最有钱的自然人股东？（附注：'恒生聚源‘的数据库显示为自然人，则该股东定为自然人股东，可能存在脏数据）

	自然人名称	持有的股票及资产	总资产
0	财政部	[工商银行6165.82亿, 农业银行4801.54亿, 交通银行1290.53亿]	12257.89亿
1	淡马锡	[建设银行908.96亿]	908.96亿
2	王靖	[信威集团480.9亿]	480.9亿
3	王传福	[比亚迪344.61亿]	344.61亿
4	张长虹	[大智慧340.28亿, *ST路翔0.51亿]	340.79亿
5	贾跃亭	[乐视网327.9亿]	327.9亿
6	张近东	[苏宁云商264.86亿]	264.86亿
7	中国第一重型机械集团公司	[中国一重250.14亿]	250.14亿
8	李仲初	[石基信息242.46亿]	242.46亿
9	龚虹嘉	[海康威视238.03亿]	238.03亿
10	傅利泉	[大华股份201.11亿]	201.11亿
11	肖文革	[印纪传媒175.97亿, 西部证券16.58亿]	192.55亿

12	杜江涛	[内蒙君正157.7亿, 博晖创新16.3亿]	174.0亿
13	梁允超	[汤臣倍健162.02亿]	162.02亿
14	孙清焕	[木林森161.08亿]	161.08亿
15	蔡东青	[奥飞动漫147.74亿]	147.74亿
16	吕向阳	[比亚迪144.47亿]	144.47亿
17	帅放文	[尔康制药128.99亿]	128.99亿
18	何巧女	[东方园林128.18亿]	128.18亿
19	易峥	[同花顺128.11亿]	128.11亿
20	田明	[美亚光电127.27亿, 西北轴承0.21亿]	127.48亿
21	姜伟	[贵州百灵126.89亿, 安泰科技0.41亿]	127.3亿
22	王俊民	[海思科125.16亿]	125.16亿
23	王伟	[朗玛信息98.87亿, 盛达矿业6.57亿, 火炬电子5.8亿, 凯发电气3.95亿, 新...	118.54亿
24	阙文彬	[恒康医疗116.81亿]	116.81亿
25	敖小强	[雪迪龙115.3亿]	115.3亿
26	庄敏	[中达股份107.71亿]	107.71亿
27	王海鹏	[美盈森105.31亿]	105.31亿
28	周亚辉	[昆仑万维98.11亿]	98.11亿
29	张轩松	[永辉超市94.87亿]	94.87亿
...	...	...	...
11186	罗篾涵	[金莱特0.03亿]	0.03亿
11187	翟振国	[北特科技0.03亿]	0.03亿
11188	高春成	[欣泰电气0.03亿]	0.03亿
11189	熊小华	[禾丰牧业0.03亿]	0.03亿
11190	冯月季	[联明股份0.03亿]	0.03亿
11191	张艳红	[金轮股份0.03亿]	0.03亿
11192	瞿斌	[联明股份0.03亿]	0.03亿
11193	宋建平	[光洋股份0.03亿]	0.03亿
11194	牛帅	[登云股份0.03亿]	0.03亿
11195	李国虎	[金莱特0.03亿]	0.03亿

11196	林薇薇	[北特科技0.03亿]	0.03亿
11197	陈天国	[雄韬股份0.03亿]	0.03亿
11198	宗长丽	[金轮股份0.03亿]	0.03亿
11199	刘燕华	[跃岭股份0.03亿]	0.03亿
11200	吴小金	[登云股份0.03亿]	0.03亿
11201	张铁立	[联明股份0.03亿]	0.03亿
11202	吴国军	[北特科技0.03亿]	0.03亿
11203	韩泉富	[联明股份0.03亿]	0.03亿
11204	余晓玲	[北特科技0.03亿]	0.03亿
11205	侯玉辉	[登云股份0.03亿]	0.03亿
11206	朱素焕	[登云股份0.03亿]	0.03亿
11207	陈茂铸	[天保重装0.03亿]	0.03亿
11208	陈兆国	[天保重装0.02亿]	0.02亿
11209	王利琼	[登云股份0.02亿]	0.02亿
11210	陈行飞	[天保重装0.02亿]	0.02亿
11211	戴晓斐	[天保重装0.02亿]	0.02亿
11212	赵丽	[天保重装0.02亿]	0.02亿
11213	陆建	[登云股份0.02亿]	0.02亿
11214	陈玉芬	[天保重装0.02亿]	0.02亿
11215	卢旭东	[红阳能源0.0亿]	0.0亿

```
11216 rows × 3 columns
```

1.5 宏观研究

【干货包邮】手把手教你做宏观择时

写贴缘由:

首先，纵观市场现有的常见策略，大致分为四大类：

1. 股票多因子alpha
2. 期货CTA
3. 市场择时：包括宏观择时、行业轮动、风格轮动等
4. 统计套利：期现套利、跨期套利、跨品种套利等

结合现在市场行情，期现套利可能在去年大赚，但今年可能没法进行（程序化被封+股指期货长期深度贴水）；期货CTA更多运用在多元化资产配置上，往往在市场大幅波动时表现较好；而市场择时则是每一位投资经理都想做好的事，当然也不是那么容易的事；在笔者目前浅短的目光下，多因子alpha或许是未来，一方面收益稳健、波动小，另一方面则体现在市场容量大，只是现在市场贴水严重，导致多因子alpha也寸步难行。

但出现了症状是不是就坐以待毙呢？市场是活的，市场的机会也是无处不在的，所以做投资就必须绞尽脑汁的想出各种法子来获取收益，比如就诞生了当下特别流行的分级基金套利（相比两年前这个策略大伙都还不他熟悉吧）。

回到正题，假设如笔者所想，多因子alpha是市场长远的未来，那么如何将其做到完美呢？从策略本身角度讲，alpha model更多关注一些基本面的东西，但实际中诸如宏观、市场情绪等很多因素都很难统一纳入到alpha model里面。从量化的本职工作来讲，将尽可能多的信息量化成实际可用的模型，这些宏观的、情绪的、政策预期、国际因素等是否也可以单独构建一个模型呢？结合alpha model，两者是否可以发挥更大的价值呢？这就是本文的干货点——宏观择时！

PS：很多时候做量化比的就是精细度，但是做到完美必然需要很多人的参与（从不同的角度去审视投资逻辑、投资细节），但现实是，量化的知识产权受限了交流的广度，各家私募都是闭门练功！笔者希望从我开始，从干货开始，希望在社区里形成交流分享的好习惯：不求作者已经在实盘赚钱的策略，比如一些好的投资想法、宏观研究、或是对一些投资细节的探讨等。。。三个臭皮匠顶个诸葛亮，希望在优矿社区上我们这帮矿工都能进步成长，分享投资观点、提出建议，一起构建好的策略。 最后，市场不缺钱，更何况优矿每个月还有500万呢，总之，共同进步、共同成长吧！

本贴内容提纲

- 为什么要择时，投资逻辑在哪
- 券商报告常见的宏观择时方法介绍
- 优矿上的宏观择时研究

1、为什么要择时，投资逻辑在哪？

A、为什么要择时：简单来讲，择时对了可以赚钱啊！但从量化角度看，择时可以将一些没有被alpha model包含的市场信心融入到模型中，涵盖面更广，组合策略更稳健、更可靠。

B、投资逻辑：从长远角度来看，股市是跟着经济走的，所谓股市是经济的晴雨表，投资者投资股票也是为了分享企业发展的红利（从宏观角度看，短线投资者只是增加了波动）；那么，在这个大经济体内，通过一些宏观指标的在某种程度上能够感知到经济的走势，或许会有预测能力；说一个现象，在每个月公布PMI数据的时候，市场总是会有很大的反应，至于什么原因读者自己去理解，只说一句：PMI被公认为是宏观经济最具有代表性的先行指标之一。

2、券商报告常见的宏观择时方法介绍

这一部分不打算多说，因为大家的投资逻辑都一样，但是使用的方法略有差异。在现有的券商报告中常见的宏观择时方法就是“逐项回归法”，简单来讲，就是先选取很多宏观经济变量（比如PMI、CPI、M1和M2的增速差。。。），然后将这些变量作为备选自变量，上证综指为因变量，进行逐项回归测试，找出几个显著的指标来预测下月大盘走势；然后动态进行，以此类推。

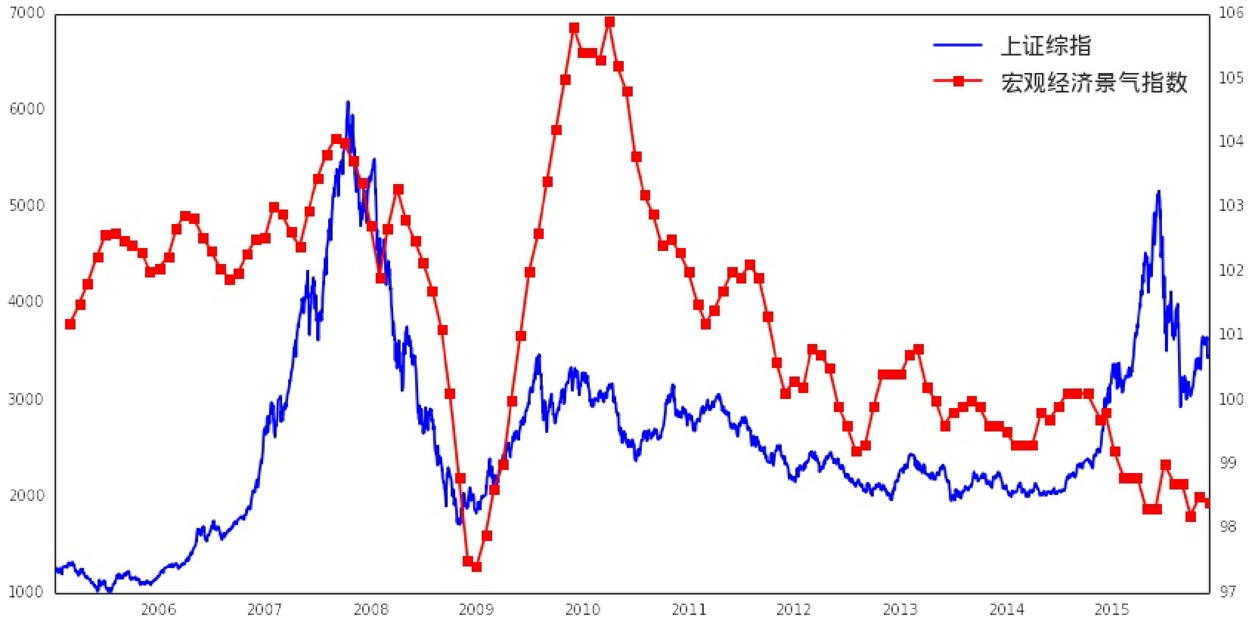
这个方法有理论基础，也有清晰的投资逻辑，具体感性的可以参见这篇报告《基于择时功效的股市宏观多因素预测模型》，在网上应该都能搜索得到的。

3、优矿上的宏观择时研究

做宏观择时研究第一出发点就是要找到能够预测大盘走势的宏观指标，试想，倘若某个指标和大盘走势非常接近，那么是不是可以说这个指标有很强的预测能力呢？废话不多说，先来一组图，选用的指标是：宏观经济景气指数（先行指数）

```python
import pandas as pd
import numpy as np
from CAL.PyCAL import *
from matplotlib import pyplot as plt
from datetime import datetime as dt

start_date = '20050101'
end_date = '20151201'
sz_index = DataAPI.MktIdxdGet(ticker=u"000001",beginDate=start_date,endDate=end_date,field=u"tradeDate,closeIndex",pandas="1").sort(columns='tradeDate').reset_index(drop=True)
sz_index['tradeDate'] = sz_index['tradeDate'].apply(lambda x: dt.strptime(x, '%Y-%m-%d'))
macro_data = DataAPI.ChinaDataECIGet(indicID=u"M020000005",beginDate=start_date,endDate=end_date,field=u"publishTime,dataValue",pandas="1").sort(columns='publishTime').reset_index(drop=True)
macro_data['publishTime'] = macro_data['publishTime'].apply(lambda x: dt.strptime(x[0:10], '%Y-%m-%d'))

# 绘图
fig = plt.figure(figsize=(14,7))
ax1 = fig.add_subplot(111)
ax1.plot(sz_index['tradeDate'], sz_index['closeIndex'], 'b-', label=u'上证综指')
ax2 = ax1.twinx()
ax2.plot(macro_data['publishTime'], macro_data['dataValue'], 'r-s', label=u'宏观经济景气指数')
lines1, labels1 = ax1.get_legend_handles_labels()  # 设置legend
lines2, labels2 = ax2.get_legend_handles_labels()
font.set_size(16)
ax1.legend(lines1 + lines2, labels1+labels2, loc='best',prop = font)
```

```
<matplotlib.legend.Legend at 0xc838b90>
```



如上图所示：

- 蓝线代表上证综指（日度数据）、红线代表宏观经济景气指数（月度数据）
- 从2005年至今的数据来看，红线和蓝线保持了高度的一致性，除了14年的牛市出现了背离，关于这一点的原因下文会有分析
- 试想，拿宏观经济景气指数走势来预测大盘，结果会如何呢？

需要说明的细节：

- 宏观数据有两个重要属性：数据日期和发布日期，由于数据只有在发布之后才知道，所以我们这里取宏观数据用的是发布日期
- 按照日期大小将日度数据和月度数据绘制在一张表上，由于宏观数据用的是发布日期，所以没有用到一点未来数据

找到一个好的宏观经济指标之后要干啥呢？当然是要进行回测，根据当前指标的大小、趋势来对下个月的上证综指做一个判断，那么就会有两个问题：

1. 怎么根据指标的值量化多空信号，或者怎么定义当前指标所给出的宏观经济趋势
2. 上证综指收益怎么计算，直接计算每个月的吗？

我们一个个来分析：

1. 从图上可以看到，上证综指的趋势和宏观经济景气指数的趋势非常接近，简单来讲，就是说，倘若这个月宏观经济景气指数大于上个月的，那么就可以认为下个月上证综指的收益为负；当然，根据数据值定义趋势、产生信号的过程有很多种方式，读者可以参考这篇研报《光大证券_20110616_2011年中期金融工程投资策略专题-宏观因子择时分析方法》，当然也欢迎大家自行发挥、多多交流（这也是社区的最大功效哈，集众人的智慧！！！）

2. 关于上证综指收益的计算：由于我们是根据宏观经济景气指数的信号来构建组合，那么我们计算的时点是不是应该是景气指数的公布时间呢？比如2015-01-01公布了景气指数，我们通过某种算法得到了对下个月的观点（多或空），那么我们的收益就应该是从2015-01-01起至下一次公布该指数的时间点。所以最真实的收益计算应该按照宏观指标的公布时间来，分别计算公布时间之间的累计收益，具体计算代码如下。

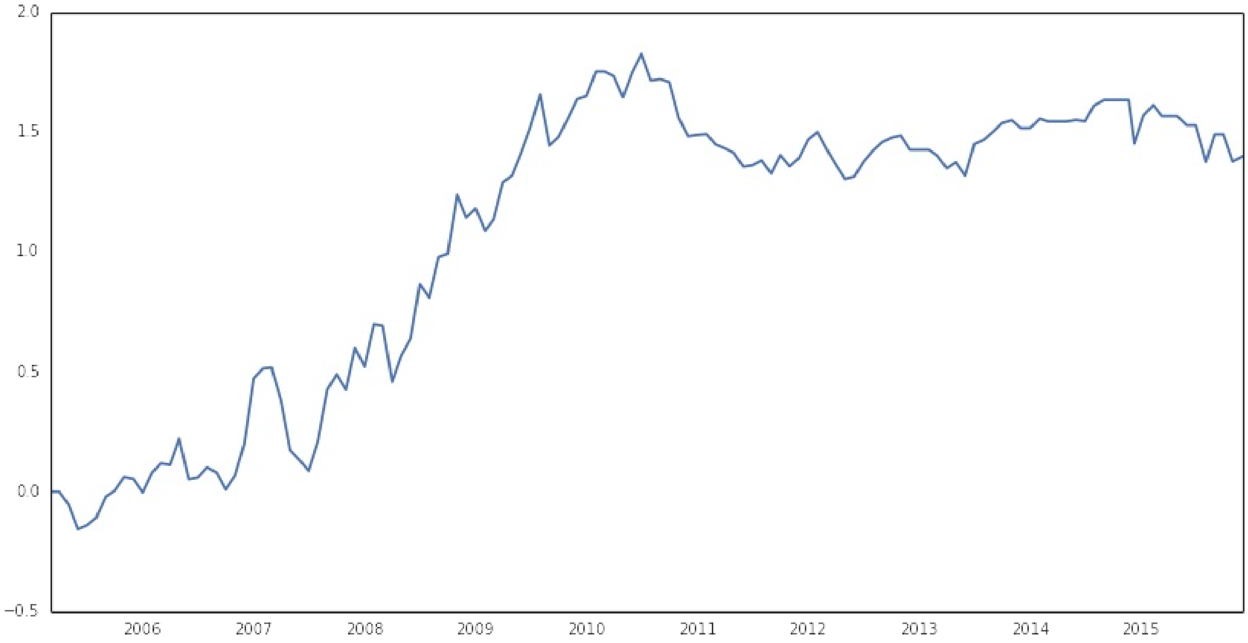
```
# 上证综指收益等数据都存放在macro_data里
macro_data['sz_return'] = 0.0
macro_data['dataValue change'] = 0.0
for i in macro_data['publishTime']:
    tmp = sz_index[sz_index['tradeDate'] <= i]
    macro_data.loc[macro_data['publishTime']==i,'sz_return'] = tmp.iloc[tmp.shape[0]-1,1]
macro_data.loc[1:, 'sz_return'] = macro_data['sz_return'][1:].values / macro_data['sz_return'][:-1].values - 1
macro_data.loc[0, 'sz_return'] = 0
macro_data.loc[1:,'dataValue change'] = macro_data['dataValue'].diff()[1:].values
macro_data.head()
```

	publishTime	dataValue	sz_return	dataValue change
0	2005-03-01	101.20	0.000000	0.00
1	2005-04-01	101.48	-0.061259	0.28
2	2005-05-01	101.82	-0.052649	0.34
3	2005-06-01	102.23	-0.103489	0.41
4	2005-07-01	102.58	0.015788	0.35

如上所示：

- 计算了上证综指的月度收益率以及宏观经济景气指数变动的值
- 那么就可以构建择时策略：当变动大于零时，做多；当变动小于零时，做空；当变动为零时，观望。
- 需要注意的一个细节：当期数值预测的是下一期的收益，这个在代码里有体现，回测结果如下所示。

```
macro_data['daily return'] = 0.0
for i in macro_data.index[:-1]:
    if macro_data.loc[i,'dataValue change'] > 0:
        macro_data.loc[i+1,'daily return'] = macro_data.loc[i+1,'sz_return']
    elif macro_data.loc[i,'dataValue change'] < 0:
        macro_data.loc[i+1,'daily return'] = -macro_data.loc[i+1,'sz_return']
macro_data['cumulative return'] = macro_data['daily return'].cumsum()  # 由于是择时，所以这里假设每期等额去投资，没有以复利形式计算
plt.figure(figsize=(14,7))
plt.plot(macro_data['publishTime'], macro_data['cumulative return'])
print '胜率为:' , np.round(1.0 * sum(macro_data['daily return'] > 0)/ sum(macro_data['daily return'] != 0),3)

胜率为: 0.595
```

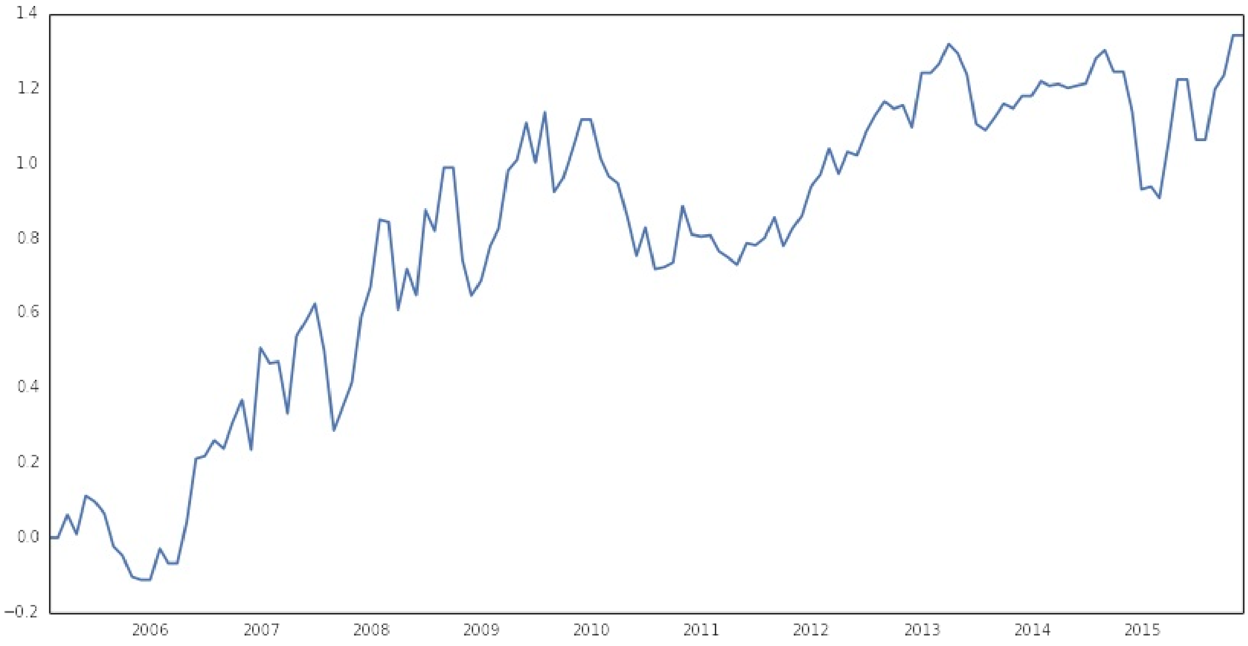


结果分析：

- 从累计收益图上看，效果还是不错的；而且胜率为59.5%，显著高于50%
- 不足之处在于2010年之后效果没有得到持续，而且如前文提到的，在去年的大牛市中，也没有预测准，不过估计去年从宏观来看应该很难判断到牛市的到来吧
- 从图表上看，2010年之后，两者的走势还是非常一致的，至于效果为什么没有预期的好，读者可以自行研究，我这里就不再进行更细致的分析了

接下来，将相同的分析运用到另一个代表性很强的宏观指标上：PMI

```
PMI = DataAPI.ChinaDataPMIGet(indicID=u"M020000008",beginDate=start_date,endDate=end_date,field=u"publishTime,dataValue",pandas="1").sort(columns='publishTime').reset_index(drop=True)
PMI['publishTime'] = PMI['publishTime'].apply(lambda x: dt.strptime(x[0:10], '%Y-%m-%d'))

PMI['sz_return'] = 0.0
PMI['dataValue change'] = 0.0
for i in PMI['publishTime']:
    tmp = sz_index[sz_index['tradeDate'] <= i]
    PMI.loc[PMI['publishTime']==i,'sz_return'] = tmp.iloc[tmp.shape[0]-1,1]
PMI.loc[1:, 'sz_return'] = PMI['sz_return'][1:].values / PMI['sz_return'][:-1].values - 1
PMI.loc[0, 'sz_return'] = 0
PMI.loc[1:,'dataValue change'] = PMI['dataValue'].diff()[1:].values

PMI['daily return'] = 0.0
for i in PMI.index[:-1]:
    if PMI.loc[i,'dataValue change'] > 0:
        PMI.loc[i+1,'daily return'] = PMI.loc[i+1,'sz_return']
    elif PMI.loc[i,'dataValue change'] < 0:
        PMI.loc[i+1,'daily return'] = -PMI.loc[i+1,'sz_return']
PMI['cumulative return'] = PMI['daily return'].cumsum()  # 由于是择时，所以这里假设每期等额去投资，没有以复利形式计算
plt.figure(figsize=(14,7))
plt.plot(PMI['publishTime'], PMI['cumulative return'])
print '胜率为：' , np.round(1.0 * sum(PMI['daily return'] > 0)/ sum(PMI['daily return'] != 0),3)

胜率为： 0.571
```



结果分析：

- 从图表上看，PMI的择时收益还是不错的，而且相对来说，持续性更稳健，尽管在某些时段最大回撤比较大
- 从胜率来看，57%的胜率也是蛮高的

总结全文

本文从初始投资逻辑入手，手把手讲述了如何做宏观择时，以及在实际处理中会遇到的一些细节问题。

从实际回测角度看，效果还是不错的，说明宏观经济变量还是具有非常好的前瞻性；但需要明白的是，宏观择时只是让模型加入了更多的市场信息，但实际也并非完全和模型一致，比如去年的大牛市，各种宏观指标都指向经济下行，但是大盘就是被拉到了5100；从事后来看，有很多的故事来解释去年的现象，这里也不一一累赘，只需要明白模型不是万能的，它只是帮我们总结过去、展望未来，省去了很多重复的工作。

最后，作为写作初衷，也和社区很多矿友一样，希望在优矿社区和大家进行交流和分享，共同进步！！！

宏观研究：从估值角度看当前市场

> 来源：https://uqer.io/community/share/5609f54af9f06c597065ef46

1、写在最前面

本贴相关内容受启发于姜超一文从今天起不再悲观，但与原文侧重点完全不一样。

本贴内容属于宏观研究范畴，从实用性角度看适用于中长线投资，对于喜好短线操作者可以参见笔者之前写的帖子追寻“国家队”的足迹，该贴自发出到现在国家队建仓基本完毕，笔者每周都会在评论部分进行数据更新，现在回头来看短线预测效果还是不错的。

2、关于本贴

本贴首先用最通俗的语言来解释何为估值，估值有何参考价值

其次，计算中证800历史PB、PE值，并介绍框架下的一些细节问题

最后，从估值的角度来审视当前市场

3、何为估值

简单讲，估值就是你舍得为某个东西花多少钱，那么在你做决策的过程中最需要考虑的因素应该是：它现在值多少钱（本身的价值）+ 未来能给我带来多少钱（它的发展潜力）。这也就是常见的的估值指标：市净率（PB）和市盈率（PE），拿到某个股票上来讲，市净率就是每一股的价格/每一股所占公司净资产的值，市盈率就是每一股的价格/每一股的盈利。应该来说还是很好理解的，下面对这两个指标来进行简单分析：

- 市净率PB：等于公司总市值/所有者权益，投资该公司看中的是其所有者权益增加部分，而所有者权益的增加也有赖于净利润的增加，所以和市盈率多少有些类似
- 市盈率PE：等于公司总市值/净利润，看中公司的利润水平以及其未来增长能力，一般来说对于高科技企业其市盈率都比较高

4、计算历史PB、PE值

教科书版的PB、PE计算是非常简单的，但是实际中会遇到很多特殊情况，先介绍计算框架：

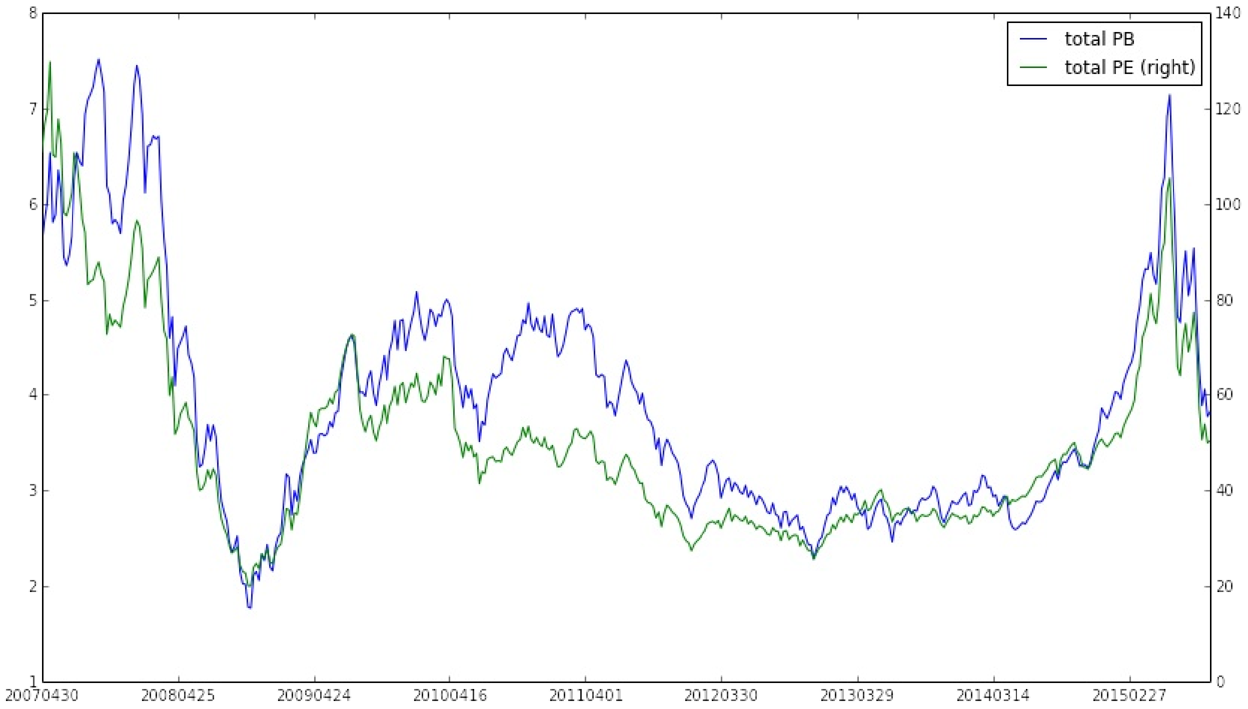
- 中证800：中证800对整个A股市场代表性强，且财务报表质量相对有保障，扣除掉中证800之后的股票所占市值非常小，在以市值加权情况下对整体结果影响不大
- 权重：首先考虑的是市值加权，但是考虑银行类权重和其他股市值差别过大，所用对数市值进行加权
- 时间：考虑股改因素以及数据的质量问题，计算的历史PB、PE从2007年4月30日至今，每周计算一次

接下来就重点介绍细节处理：

- TTM处理：在计算指标中，净利润采用TTM（过去12个月净利润）计算，数据更合理
- 负值：当公司净利润为负或者所有者权益为负时会照成PB、PE出现负值，结合前面介绍的指标计算公式，负值一般会非常大；处理方式是将负值赋值为0
- 极值：当公司某一年利润或者所有者权益特别小时，会导致PB、PE出现非常大的极值情况，采用`winsorize`处理（`winsorize`原理参见笔者前一篇帖子破解Alpha对冲策略）

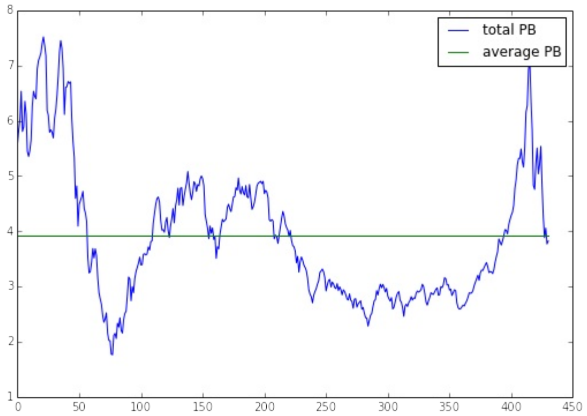
```
# 导入包
import pandas as pd
import numpy as np
from datetime import datetime, timedelta
import matplotlib.pyplot as plt

start_date = '20070430'  # 开始日期
end_date = (datetime.today() - timedelta(days=1)).strftime('%Y%m
%d')  # 截止日期
data = DataAPI.TradeCalGet(exchangeCD=u"XSHG",beginDate=start_da
te,endDate=end_date,field=u"calendarDate,isWeekEnd",pandas="1")
data = data[data['isWeekEnd'] == 1]
Dates = map(lambda x: x[0:4]+x[5:7]+x[8:10], data['calendarDate'
].values.tolist())

PB_PE = pd.DataFrame(index=Dates, data=0, columns=['total PB','t
otal PE'])
for date in Dates:
    universe = set_universe('HS300', date) + set_universe('ZZ500'
, date)
    data1 = DataAPI.MktStockFactorsOneDayGet(tradeDate=date,secI
D=universe[:400],field=['ticker','LCAP','PB','PE'],pandas="1").s
et_index('ticker')
    data2 = DataAPI.MktStockFactorsOneDayGet(tradeDate=date,secI
D=universe[400:],field=['ticker','LCAP','PB','PE'],pandas="1").s
et_index('ticker')
    total = pd.concat([data1, data2])
    length_total = total.shape[0]
    total.dropna(inplace=True)

    # 数据量不足80%时，不具有代表性
    if float(total.shape[0]) / length_total < 0.8:
        continue

    # 细节处理：负值极值处理
    total['PB'][total['PB'] < 0] = 0
    total['PE'][total['PE'] < 0] = 0
    pb = winsorize(total['PB'].to_dict())
    pe = winsorize(total['PE'].to_dict())
    total['PB'][pb.keys()] = pb.values()
    total['PE'][pe.keys()] = pe.values()

    # 权重 + 绘图
    total['wts'] = total['LCAP'] / sum(total['LCAP'])
    PB_PE.loc[date] = [sum(total['wts'] * total['PB']), sum(tota
l['wts'] * total['PE'])]
PB_PE[['total PB','total PE']].plot(secondary_y='total PE',figsi
ze=(14,8))
```

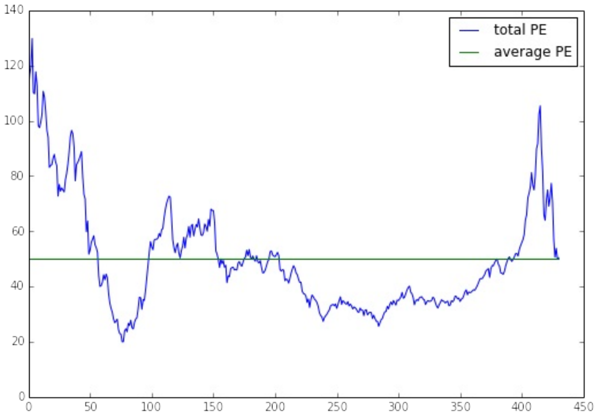
```
<matplotlib.axes.AxesSubplot at 0x4b4a710>
```



- 从上图可以很清晰的看到从2007年4月30以来，中证800PB、PE的历史走势图，估值水平和大盘走势完全一致！（毕竟指标的分子是股价）
- 那么，当前的估值水平相比历史来看如何呢？
- 接下来，分别将PB、PE和历史平均PB、PE来进行对比，并进行简单分析

```
PB_PE['average PB'] = PB_PE['total PB'].mean()
PB_PE['average PE'] = PB_PE['total PE'].mean()

# 绘图
fig = plt.figure(figsize=(19,6))
ax1 = fig.add_subplot(1,2,1)
ax2 = fig.add_subplot(1,2,2)
ax1.plot(PB_PE[['total PB','average PB']])
ax2.plot(PB_PE[['total PE','average PE']])
ax1.legend(['total PB','average PB'])
ax2.legend(['total PE','average PE'])

<matplotlib.legend.Legend at 0x7c82210>
```





- 从估值的角度来看，当前的市场处于历史平均估值的水平，是一个相对合理的点位，至少对于长线投资者持有中证800还是比较放心的
- 但是，是否说明大盘就不会下跌呢？那就看读者对估值的理解以及当前市场环境的判断
- 笔者以为，估值合理更适合从长远的角度来看，短期则更多受投资者情绪、资金面情况、海外市场等因素的影响；此外，结合历史也可以看到，急涨之后伴随着急跌，而且急跌往往会跌破合理估值水平，这就有点像物理里的惯性，说不定惯性也是股市的固有的属性呢，又或者说投资者风险偏好也是具有惯性这一固有属性呢？

追寻“国家队”的足迹

> 来源：https://uqer.io/community/share/55d44816f9f06c5221156a3d

背景介绍

- 还记得证金公司借道基金公司2000亿的救市资金吗？
- 简单来讲就是证金公司出钱给基金公司成立新的基金，用以提升市场流动性、维稳市场
- 2000亿总共分给了5只基金,具体各只基金的信息详见下文
- 另外，关于这2000亿的救市基金的相关报道很多，本文也是受《华尔街见闻》时芳胜的一篇文章启发，想着在UQER上做下简单的研究试试
- 链接是新浪财经的报道，有兴趣的可以看看 http://finance.sina.com.cn/money/fund/20150807/105122904602.shtml

研究意义

- 首先看一下，证金公司效应：8月4日，"梅雁吉祥"发布公告，证金公司已成为公司第一大股东，截止发稿日（8月17日），其已累计上涨129%
- 由此可见，国家队的直接效应还是很高的，那么，国家队借道基金公司的钱也应该值得重视
- 一方面是选股，这些基金的重仓股可以给我们一些投资参考（遗憾的是，这些基金刚成立，有的还没建仓完成，这一点可以之后再进行跟踪）
- 另一方面是择时，这些基金的建仓时点可以给一些市场的参考，假如我们推算的基金建仓的大盘点位，那么从长远来看这个点位至少是比较安全的点位
- 选股方面依赖于基金的季度公告，但择时我们可以细细分析，因为这些基金目前并没有建仓完全

先熟悉一下这5只基金

```
DataAPI.FundGet(ticker=u"001620,001683,001772,001773,001769",field=u"ticker,secShortName,establishDate,managementFullName,investField,perfBenchmark",pandas="1")
```

	ticker	secShortName	establishDate	managementFullName

0	001620	嘉实新机遇灵活配置混合	2015-07-13	嘉实基金管理有限公司
1	001683	华夏新经济灵活配置混合	2015-07-13	华夏基金管理有限公司
2	001769	易方达瑞惠灵活配置混合	2015-07-31	易方达基金管理有限公司
3	001772	南方消费活力灵活配置混合	2015-07-31	南方基金管理有限公司
4	001773	招商丰庆灵活配	2015-07-31	招商基金管理有限公司

4	001773	置混合-A	2015-07-31	招商基金管理有限公司	

可见，5只基金中都是最近才成立的，而且类型都为混合型，其业绩的比较基准也比较接近：50.0%×上海证券交易所国债指数+50.0%×沪深300指数

推断建仓时点逻辑

- 从基金的净值来推算基金的建仓情况
- 假设基金已完全建仓，那么其收益情况应该和基准的收益差不多
- 那么，通过对比5只基金的收益与基准的收益情况，从偏差可以大致推测基金的建仓情况；同时，在未来可以通过这种方法来监测5只基金的仓位变化，比如，当5只基金建仓完成后，若出现基准和基金收益相差较大的情况，很大可能是由于基金降低了仓位，这可以作为大盘短期的预警点位
- 略有遗憾的是，目前5只基金公布的净值是周度数据，没有日度数据，这样我们只能周度进行比较
- 笔者从嘉实基金官网看到，001620，在产品概况一栏有涨跌幅（日度），但这个数据只有一天的数据，有兴趣的可以每天去官网看看这5个基金的日度收益情况

下面，就做出对比收益图（周度收益，并非累计收益），直观看到变化（自动更新，只用运行代码即可）

```
import pandas as pd
import numpy as np

# 5只基金行情数据
funds = DataAPI.FundNavGet(ticker=u"001620,001683,001772,001773,001769",beginDate=u"20150701",field=u"ticker,endDate,ACCUM_NAV",pandas="1").pivot(index= 'endDate',columns='ticker',values='ACCUM_NAV')
# 基准数据：上证国债指数 + 沪深300
benchmark = DataAPI.MktIdxdGet(ticker=u"000300,000012",beginDate=u"20150701",field=u"ticker,tradeDate,closeIndex",pandas="1").pivot(index= 'tradeDate',columns='ticker',values='closeIndex')

table = pd.merge(funds,benchmark, left_index=True, right_index=True, how = 'inner')
table[1:] = table[1:].values / table[:-1].values - 1
table[0:1] = 0
table.fillna(0, inplace= True)
table['benchmark'] = table['000012'] * 0.5 + table['000300'] * 0.5
table.drop(['000012','000300'],axis = 1,inplace = True)
table.plot(figsize = (14,8))
```

~~~
![](img/wPDGGBLVtN+jQAAAABJRU5ErkJggg==.png)

+ 上图中，图例中的前2个001620、001683为7月13日成立的基金，可以看出，两只基金走势和基准最贴近，这一点也间接印证了笔者前面的分析，可以看到，这两只基金在8月7日已基本完成了建仓，建仓时点应该在8月1日~8月7日，对应的上证综指位于3600~3800之间

+ 001772似乎也在8月1日~8月7日完成了建仓，也就是刚成立就完成了建仓，相比前两个基金来的更迅速（前两个是7月13日就成立了），笔者的直观理解是，3600~3800点应该是基金公司所谓的短期市场相对低点

+ 回头看看，7月13日之后的行情，而刚开始成立的两只基金并没有急于建仓，大部分时间都在3800点以上，这一点读者自行理解吧

+ 最后看看另外两只基金，001769和001773，似乎还没有看到建仓的迹象，因为收益很低，比较好的解释是仓位很低，那么这两只基金是在等建仓时机吗？今天是8月17日，本周的第一天，大盘点位3993

+ 或许等到这周结束我们可以看到这两只基金是否建仓完成，但是从时效上来说是有点滞后的

+ 笔者又进一步查看了两只基金的官网，发现001773（招商丰庆灵活配置混合-A）是有净值单日变动的，也就是说可以查看到日度的收益率变动，看到这点笔者都有点小激动了。。

那么接下来的事情简单了：

+ 每日收盘后去招商基金官网，看一下001773的日度净值变动，然后和基准日度收益进行对比（运行下面的代码，得到最近几日的日度基准收益），便可以大致分析出001773的建仓情况(http://www.cmfchina.com/main/001773/fundinfo.shtml)

+ 每周末运行上面的代码，对比5只基金和基准的周度收益，看是否有显著变化；在今后，若全部建仓完成之后收益率偏差还出现显著变化，则有可能是该基金降低了仓位，可以作为一个预警信息

```py
bench = DataAPI.MktIdxdGet(ticker=u"000300,000012",beginDate=u"20150801",field=u"ticker,tradeDate,closeIndex",pandas="1").pivot(index= 'tradeDate',columns='ticker',values='closeIndex')
bench[1:] = bench[1:].values / bench[:-1].values - 1
bench[0:1] = 0
bench['benchmark'] = 0.5 * bench['000012'] + 0.5 * bench['000300']
bench.tail(2)
~~~

	ticker	000012	000300	benchmark
tradeDate				
2015-08-14	-0.000080	-0.000472	-0.000276	
2015-08-17	0.000246	0.001063	0.000655	

二 套利

2.1 配对交易

HS300ETF套利（上）

> 来源：https://uqer.io/community/share/56599d1ef9f06c6c8a91ada4

新股民入市时，一般都会收到一句忠告：“买ETF吧！”。

对于大部分散户而言，这句话十分刺耳，但是却无比正确。

假设来了：如果定投HS300ETF，那么从510300ETF基金上市以来，收益如何呢？

```
df = DataAPI.MktFunddAdjGet(ticker='510300',field='tradeDate,closePrice')
print '涨幅：%s%%' %(100*df.closePrice.iloc[-1] / df.closePrice.iloc[0]-100)
df.head().append(df.tail())

涨幅：45.1420890937%
```

	tradeDate	closePrice
0	2012-05-28	2.6040
1	2012-05-29	2.6440
2	2012-05-30	2.6360
3	2012-05-31	2.6300
4	2012-06-01	2.6300
848	2015-11-23	3.9869
849	2015-11-24	3.9901
850	2015-11-25	4.0142
851	2015-11-26	3.9953
852	2015-11-27	3.7795

可以看到，三年多涨幅高达45%，这还是在经历了股灾后的收益。

不费心不费力，就可以大幅跑赢宝宝。

不过值得注意的是，如果买在牛市高点，那就要套牢了。

也许你要问了，这个东西，确实是很省心，而且也享受了大盘上涨带来的红利，但是这个收益，咱还能不能再提高点呢？

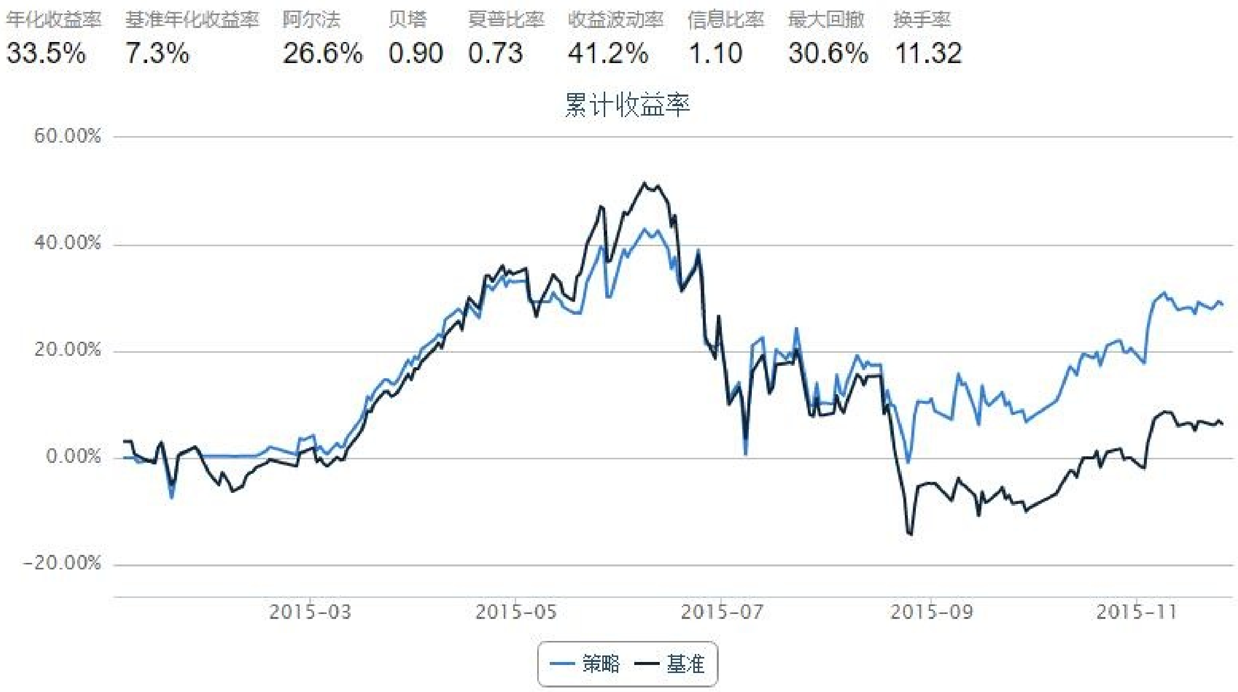
答案当然是肯定的。

目前市场上，挂钩HS300指数的ETF基金有好几款，其中流通性最好的就是沪市的510300和深市的159919。我的策略思路是：

两只ETF均挂钩HS300指数，估值透明，当A折价大于B时，卖出B买入A，反之同理。

直接上代码：

```
from CAL.PyCAL import *
start = '2015-01-01'                        # 回测起始时间
end = '2015-11-26'                          # 回测结束时间
benchmark = 'HS300'                         # 策略参考标准
universe = ['510300.XSHG', '159919.XSHE']  # 证券池，支持股票和基金
sh300, sz300 = universe
capital_base = 100000                       # 起始资金
freq = 'd'                                  # 策略类型，'d'表示日间策略使用日线回测，'m'表示日内策略使用分钟线回测
commission = Commission(buycost=0.00015, sellcost=0.00015)
refresh_rate = 1                            # 调仓频率，表示执行handle_data的时间间隔，若freq = 'd'时间间隔的单位为交易日，若freq = 'm'时间间隔为分钟
cal = Calendar('China.SSE')

def initialize(account):                    # 初始化虚拟账户状态
    pass

def handle_data(account):                   # 每个交易日的买入卖出指令

    # 有停牌的话，今天就跳过。
    if len(account.universe) < 2: return

    last_date = cal.advanceDate(account.current_date, '-1B').strftime('%Y%m%d')
    try:
        # 获取两只基金昨日收盘时的折价率
        sh300_df = DataAPI.MktFunddGet(ticker=u"510300",beginDate=last_date,endDate=last_date,field=u"discountRatio",pandas="1")
        sz300_df = DataAPI.MktFunddGet(ticker=u"159919",beginDate=last_date,endDate=last_date,field=u"discountRatio",pandas="1")
        discount_sh = sh300_df.discountRatio[0]
        discount_sz = sz300_df.discountRatio[0]
    except:
        return
    # 搬搬搬
    if discount_sh - discount_sz > 0.002:
        order_pct_to(sh300, 0.99)
        order_pct_to(sz300, 0)
    elif discount_sz - discount_sh > 0.002:
        order_pct_to(sh300, 0)
        order_pct_to(sz300, .99)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
33.5%	7.3%	26.6%	0.90	0.73	41.2%	1.10	30.6%	11.32

效果看着还可以吧。

这个策略还比较粗糙，而且市场容量有限，权当抛砖引玉。

有心人可以再研究交流。

有机会会再写一篇(下)，做一些更细节的测试。

本文不做任何买入建议，后果概不负责！！！

【统计套利】配对交易

> 来源：https://uqer.io/community/share/559c85baf9f06cb5614f190d

策略思路

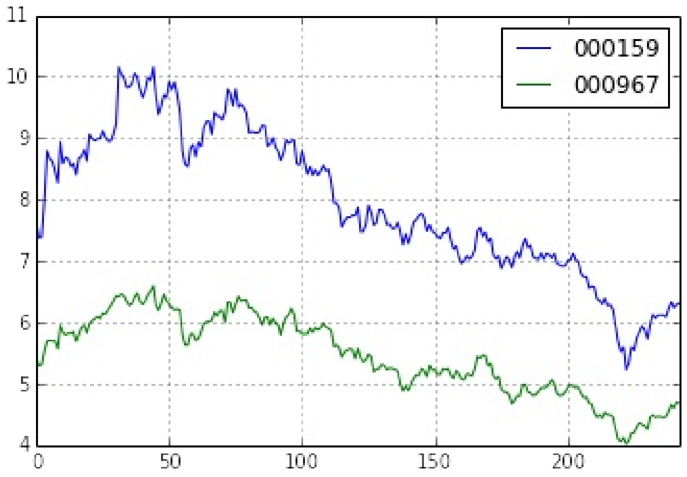
寻找走势相关且股价相近的一对股票，根据其价格变动买卖

策略实现

- 首先，历史前五日的Pearson相关系数若大于给定的阈值
- 如果两只股票走势趋同，则按上涨（下跌）趋势买入（卖出）股票
- 如果两只股票走势背离，则买入下跌股票，卖出上涨股票

首先需要在A股中寻找走势相关性很大的股票，这是一项很繁复的工作。为简单起见，这里直接使用了一个现成的结果：000159和000967在2012年的走势十分相似，这一点可以通过复权收盘价曲线来验证

```
<matplotlib.axes.AxesSubplot at 0x6415790>
```

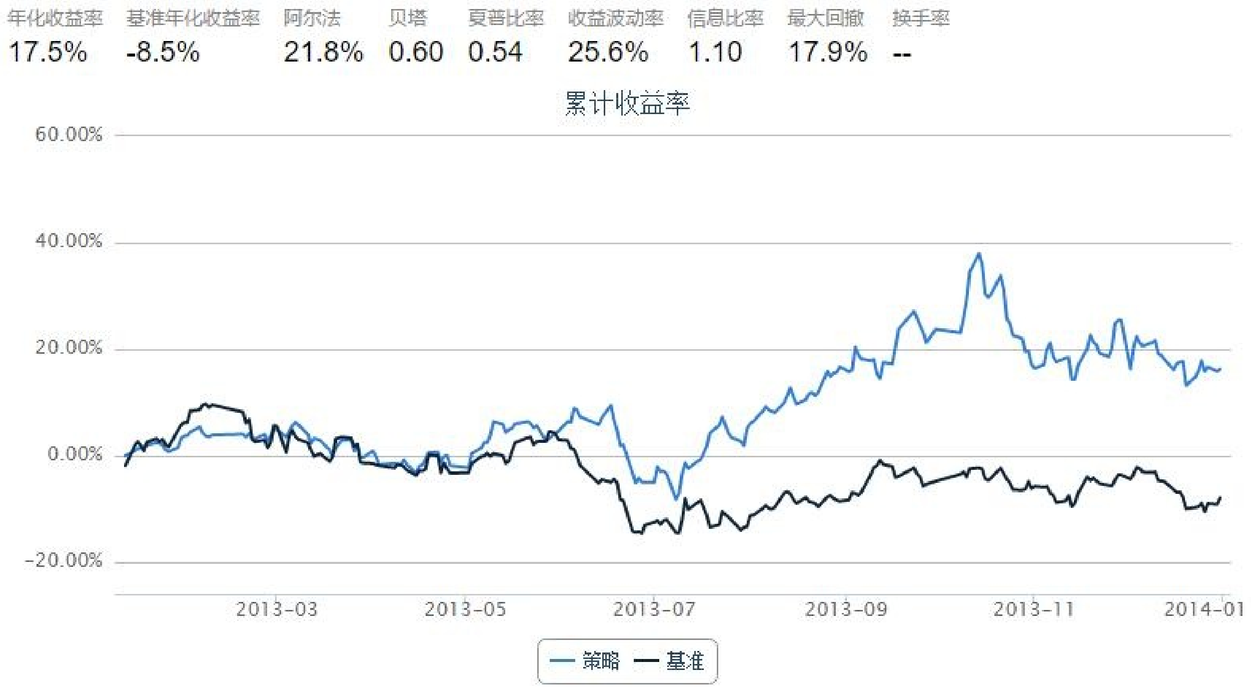


接下来我们对2013年全年的这两只股票进行配对交易策略的回测：

```
from scipy.stats.stats import pearsonr

start = '2013-01-01'
end   = '2014-01-01'
benchmark = 'HS300'
universe = ['000159.XSHE', '000967.XSHE']
capital_base = 1000000
```

```
def initialize(account):
    account.cutoff = 0.9
    account.prev_prc1 = 0
    account.prev_prc2 = 0
    account.prev_prcb = 0

def handle_data(account):
    if len(account.universe) < 2: return

    clsp = account.get_attribute_history('closePrice', 5)
    stk1, stk2 = universe
    px1, px2 = clsp[stk1], clsp[stk2]

    prc1, prc2 = px1[-1], px2[-1]
    prcb = account.get_symbol_history('benchmark', 1)['return'][0
]

    if account.prev_prc1 == 0:
        account.prev_prc1 = prc1
        account.prev_prc2 = prc2
        account.prev_prcb = prcb
        return

    corval, pval = pearsonr(px1, px2)

    if abs(corval) < account.cutoff:
        return

    mov1, mov2 = adj(prc1, prc2, prcb, account.prev_prc1, accoun
t.prev_prc2, account.prev_prcb)

    amount = 100000 / prc2
    if mov1 > 0:
        order(stk2, amount)
    elif mov1 < 0:
        if account.valid_secpos.get(stk2, 0) > amount:
            order(stk2, -amount)
        else:
            order_to(stk2, 0)

    amount = 100000 / prc1
    if mov2 > 0:
        order(stk1, amount)
    elif mov2 < 0:
        if account.valid_secpos.get(stk1, 0) > amount:
            order(stk1, -amount)
        else:
            order_to(stk1, 0)

    account.prev_prc1 = prc1
    account.prev_prc2 = prc2
    account.prev_prcb = prcb
```

```
def adj(x, y, base, prev_x, prev_y, prev_base):
    dhs = base / prev_base - 1
    dx = x / prev_x - 1 - dhs
    dy = y / prev_y - 1 - dhs
    return dx, dy
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
17.5%	-8.5%	21.8%	0.60	0.54	25.6%	1.10	17.9%	--

```
bt
```

	tradeDate	cash	security_position	portfolio_value
0	2013-01-11	1000000.000000	{}	1000000.000000
1	2013-01-14	802627.092268	{u'000159.XSHE': 16139, u'000967.XSHE': 21500}	1008718.298268
2	2013-01-15	802627.092268	{u'000159.XSHE': 16139, u'000967.XSHE': 21500}	1013493.762268
3	2013-01-16	802627.092268	{u'000159.XSHE': 16139, u'000967.XSHE': 21500}	1013887.243268
			{u'000159.XSHE':	

4	2013-01-17	802627.092268	16139, u'000967.XSHE': 21500}	1017193.274268
5	2013-01-18	601751.321172	{u'000159.XSHE': 31203, u'000967.XSHE': 41512}	1019875.334172
6	2013-01-21	601751.321172	{u'000159.XSHE': 31203, u'000967.XSHE': 41512}	1024320.333172
7	2013-01-22	601751.321172	{u'000159.XSHE': 31203, u'000967.XSHE': 41512}	1022956.509172
8	2013-01-23	601751.321172	{u'000159.XSHE': 31203, u'000967.XSHE': 41512}	1020170.513172
9	2013-01-24	601751.321172	{u'000159.XSHE': 31203, u'000967.XSHE': 41512}	1010615.301172
10	2013-01-25	601751.321172	{u'000159.XSHE': 31203, u'000967.XSHE': 41512}	1007214.174172
11	2013-01-28	601751.321172	{u'000159.XSHE': 31203, u'000967.XSHE': 41512}	1015317.846172
12	2013-01-29	601751.321172	{u'000159.XSHE': 31203, u'000967.XSHE': 41512}	1032167.466172
13	2013-01-30	601751.321172	{u'000159.XSHE': 31203, u'000967.XSHE': 41512}	1037040.199172
14	2013-01-31	601751.321172	{u'000159.XSHE': 31203, u'000967.XSHE':	1037199.250172

			41512}	
15	2013-02-01	601751.321172	{u'000159.XSHE': 31203, u'000967.XSHE': 41512}	1044424.173172
16	2013-02-04	601751.321172	{u'000159.XSHE': 31203, u'000967.XSHE': 41512}	1053929.951172
17	2013-02-05	601751.321172	{u'000159.XSHE': 31203, u'000967.XSHE': 41512}	1042352.810172
18	2013-02-06	601751.321172	{u'000159.XSHE': 31203, u'000967.XSHE': 41512}	1036748.235172
19	2013-02-07	601751.321172	{u'000159.XSHE': 31203, u'000967.XSHE': 41512}	1035219.288172
20	2013-02-08	601751.321172	{u'000159.XSHE': 31203, u'000967.XSHE': 41512}	1038390.029172
21	2013-02-18	601751.321172	{u'000159.XSHE': 31203, u'000967.XSHE': 41512}	1040366.776172
22	2013-02-19	601751.321172	{u'000159.XSHE': 31203, u'000967.XSHE': 41512}	1034837.773172
23	2013-02-20	601751.321172	{u'000159.XSHE': 31203, u'000967.XSHE': 41512}	1041832.310172
24	2013-02-21	601751.321172	{u'000159.XSHE': 31203, u'000967.XSHE': 41512}	1030863.855172

25	2013-02-22	601751.321172	{u'000159.XSHE': 31203, u'000967.XSHE': 41512}	1031881.589172
26	2013-02-25	601751.321172	{u'000159.XSHE': 31203, u'000967.XSHE': 41512}	1039886.408172
27	2013-02-26	601751.321172	{u'000159.XSHE': 31203, u'000967.XSHE': 41512}	1027334.880172
28	2013-02-27	601751.321172	{u'000159.XSHE': 31203, u'000967.XSHE': 41512}	1036669.463172
29	2013-02-28	601751.321172	{u'000159.XSHE': 31203, u'000967.XSHE': 41512}	1049123.518172
...	...	...	...	...
203	2013-11-20	4.001066	{u'000159.XSHE': 58955.0, u'000967.XSHE': 1104...	1213819.901066
204	2013-11-21	4.001066	{u'000159.XSHE': 58955.0, u'000967.XSHE': 1104...	1207170.725066
205	2013-11-22	4.001066	{u'000159.XSHE': 58955.0, u'000967.XSHE': 1104...	1191941.099066
206	2013-11-25	4.001066	{u'000159.XSHE': 58955.0, u'000967.XSHE': 1104...	1185756.570066
207	2013-11-26	4.001066	{u'000159.XSHE': 58955.0, u'000967.XSHE': 1104...	1202201.685066

208	2013-11-27	4.001066	{u'000159.XSHE': 58955.0, u'000967.XSHE': 1104...	1247942.107066
209	2013-11-28	4.001066	{u'000159.XSHE': 58955.0, u'000967.XSHE': 1104...	1254278.937066
210	2013-11-29	4.001066	{u'000159.XSHE': 58955.0, u'000967.XSHE': 1104...	1254572.047066
211	2013-12-02	4.001066	{u'000159.XSHE': 58955.0, u'000967.XSHE': 1104...	1162773.529066
212	2013-12-03	205057.003206	{u'000159.XSHE': 46102.0, u'000967.XSHE': 9480...	1203266.619206
213	2013-12-04	7547.120636	{u'000159.XSHE': 58573.0, u'000967.XSHE': 1098...	1223800.796636
214	2013-12-05	7547.120636	{u'000159.XSHE': 58573.0, u'000967.XSHE': 1098...	1211481.369636
215	2013-12-06	7547.120636	{u'000159.XSHE': 58573.0, u'000967.XSHE': 1098...	1205769.229636
216	2013-12-09	7547.120636	{u'000159.XSHE': 58573.0, u'000967.XSHE': 1098...	1212153.686636
217	2013-12-10	7547.120636	{u'000159.XSHE': 58573.0, u'000967.XSHE': 1098...	1216462.619636
	2013-12-		{u'000159.XSHE': 58573.0,	

	11		u'000967.XSHE': 1098...	
219	2013-12-12	7547.120636	{u'000159.XSHE': 58573.0, u'000967.XSHE': 1098...	1187601.796636
220	2013-12-13	7547.120636	{u'000159.XSHE': 58573.0, u'000967.XSHE': 1098...	1180581.953636
221	2013-12-16	7547.120636	{u'000159.XSHE': 58573.0, u'000967.XSHE': 1098...	1160961.657636
222	2013-12-17	7547.120636	{u'000159.XSHE': 58573.0, u'000967.XSHE': 1098...	1172930.532636
223	2013-12-18	7547.120636	{u'000159.XSHE': 58573.0, u'000967.XSHE': 1098...	1175604.134636
224	2013-12-19	7547.120636	{u'000159.XSHE': 58573.0, u'000967.XSHE': 1098...	1176775.594636
225	2013-12-20	7547.120636	{u'000159.XSHE': 58573.0, u'000967.XSHE': 1098...	1132028.254636
226	2013-12-23	7547.120636	{u'000159.XSHE': 58573.0, u'000967.XSHE': 1098...	1147914.771636
227	2013-12-24	7547.120636	{u'000159.XSHE': 58573.0, u'000967.XSHE': 1098...	1159575.888636
228	2013-12-25	7547.120636	{u'000159.XSHE': 58573.0, u'000967.XSHE': 1098...	1178451.287636

			1098...	
229	2013-12-26	7547.120636	{u'000159.XSHE': 58573.0, u'000967.XSHE': 1098...	1158503.363636
230	2013-12-27	7547.120636	{u'000159.XSHE': 58573.0, u'000967.XSHE': 1098...	1166499.850636
231	2013-12-30	7547.120636	{u'000159.XSHE': 58573.0, u'000967.XSHE': 1098...	1158181.598636
232	2013-12-31	7547.120636	{u'000159.XSHE': 58573.0, u'000967.XSHE': 1098...	1162504.538636

```
233 rows × 6 columns
```

根据 `bt` 的表现，在仓位控制和组合管理方面应该还有着不少进一步优化的空间

相似公司股票搬砖

> 来源：https://uqer.io/community/share/5555cbfaf9f06c6c7404f8ae

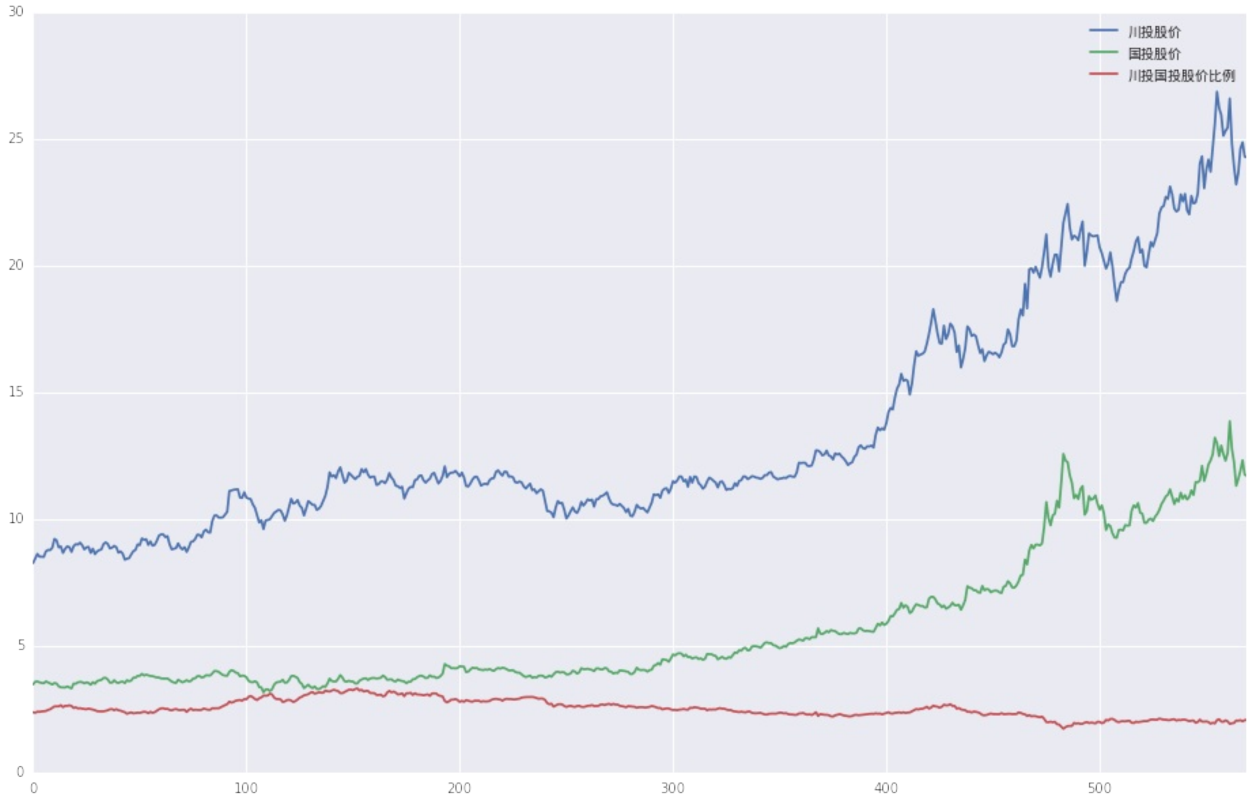
自从牛市来了,小老弟也去开了个户,平时常在雪球上找找票.

前段时间听到两只股票:川投能源,国投电力.

这两个公司是一对好基友, 他们的主要资产是一家水电公司的48%,52%股份, 两只股价常在两倍之间变动.

于是出现很多人在这两只股票之间搬砖, 本帖就来验证一下搬砖的效果, 看看结果会不会令人大吃一惊呢?

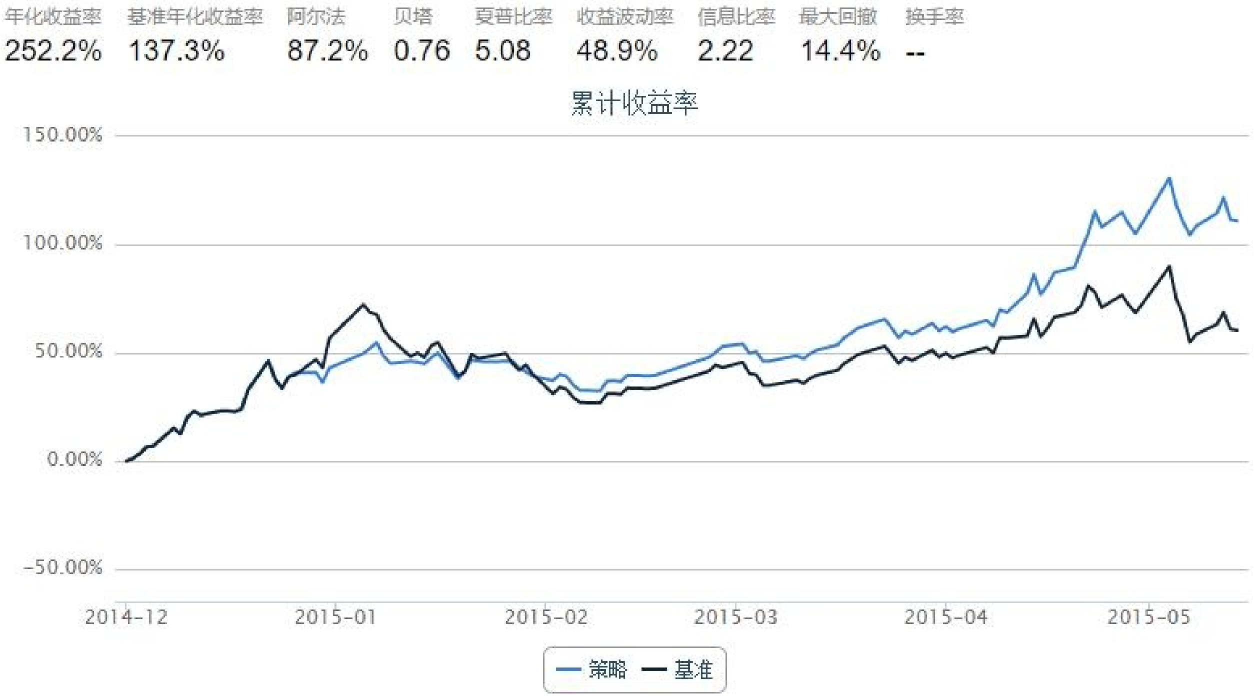
```
# 首先我们简单看下两只股票的走势.

import seaborn as sns
import pandas as pd
import matplotlib.pylab as pylab

begin_date, end_date = '20130101', '20150514'
ct,gt = '600674', '600886'
ct = DataAPI.MktEqudAdjGet(ticker=ct, beginDate=begin_date, endD
ate=end_date)
gt = DataAPI.MktEqudAdjGet(ticker=gt, beginDate=begin_date, endD
ate=end_date)

liangtou =pd.DataFrame()
liangtou['川投股价'] = ct.closePrice
liangtou['国投股价'] = gt.closePrice
liangtou['川投国投股价比例'] =  ct.closePrice / gt.closePrice
liangtou.plot( figsize=(16,10))
pylab.legend([u'川投股价',u'国投股价', u'川投国投股价比例'], prop=fon
t)

<matplotlib.legend.Legend at 0x754ce90>
```



重点来了，利用量化实验室的strategy模式，咱来编写一个策略.

- 起始建仓：全仓600886
- 起始日期：2014-12-01
- 结束日期：2015-05-14
- 起始资金：10w
- 调仓频率：1天
- 调仓信号：川投能源收盘价/国投能源收盘价*100%-200% 之差如果大于1%，卖出886买入774，如果小于-1%，则卖出774买入886

```
start = '2014-12-01'                              # 回测起始时间
end = '2015-05-14'                                # 回测结束时间
benchmark = '600886.XSHG'                           # 策略参考标准
universe = ['600674.XSHG', '600886.XSHG']              # 证券池
capital_base = 100000                               # 起始资金
refresh_rate = 1                                   # 调仓频率，即每 refresh_
rate 个交易日执行一次 handle_data() 函数
ct_stk, gt_stk = universe

def initialize(account):                            # 初始化虚拟账户状态
    pass

def handle_data(account):                           # 每天执行一次

    if len(account.universe) < 2: # 有停牌的话, 就跳过.
        return

    ct_price, gt_price = account.referencePrice[ct_stk], account
.referencePrice[gt_stk]
    percent = int(100*ct_price/gt_price-200)

    #第一次, 满仓买入600886
    if not account.valid_secpos:
        order(gt_stk, capital_base/gt_price)
        return

    if percent>1 and account.secpos.get(ct_stk, 0): #买886, 卖774
        amount = account.secpos.get(ct_stk, 0)
        print account.current_date, '买886,数量:%s, 卖774,数量:%s'
 %(amount*ct_price/gt_price, amount)
        order(ct_stk, -amount)
        order(gt_stk, amount*ct_price/gt_price)

    elif percent<-1 and account.secpos.get(gt_stk, 0): #卖886, 买
774
        amount = account.secpos.get(gt_stk, 0)
        print account.current_date, '卖886,数量:%s, 买774,数量:%s'
 %(amount, amount*ct_price/gt_price)
        order(gt_stk, -amount)
        order(ct_stk, amount*gt_price/ct_price)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
252.2%	137.3%	87.2%	0.76	5.08	48.9%	2.22	14.4%	--

```
2014-12-26 00:00:00 卖886,数量:13698, 买774,数量:27112.3136095
2015-02-03 00:00:00 买886,数量:14327.3103448, 卖774,数量:6897.0
2015-02-26 00:00:00 卖886,数量:14308.0, 买774,数量:28103.5159729
2015-03-05 00:00:00 买886,数量:14717.8178257, 卖774,数量:7284
2015-04-10 00:00:00 卖886,数量:14717, 买774,数量:28881.7914485
2015-04-14 00:00:00 买886,数量:15404.6484375, 卖774,数量:7385.0
2015-04-21 00:00:00 卖886,数量:15390.0, 买774,数量:29629.81316
2015-04-24 00:00:00 买886,数量:16527.4811393, 卖774,数量:7993
2015-05-05 00:00:00 卖886,数量:16527, 买774,数量:31706.5606061
2015-05-08 00:00:00 买886,数量:17669.7435897, 卖774,数量:8614
```

这结果真是让老弟目瞪口呆啊, 赶紧搬起.

该策略只是比较的昨日收盘价, 如果考虑日间搬砖, 简直不敢想了[口水].

Paired trading

> 来源：https://uqer.io/community/share/54895a8df9f06c31c3950ca0

配对交易

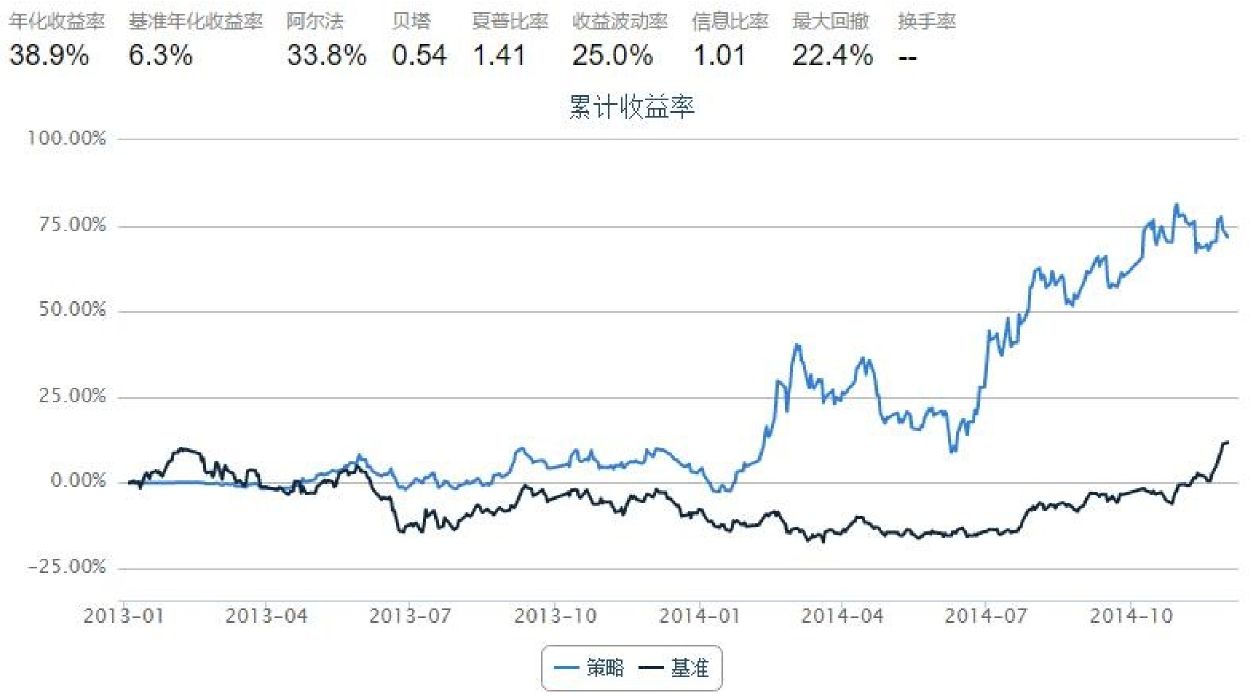
策略思路

寻找走势相关且股价相近的一对股票，根据其价格变动买卖

策略实现

历史前五日的Pearson相关系数若大于给定的阈值则触发买卖操作

```
from scipy.stats.stats import pearsonr

start = datetime(2013, 1, 1)
end   = datetime(2014, 12, 1)
benchmark = 'HS300'
universe = ['000559.XSHE', '600126.XSHG']
capital_base = 1e6

corlen = 5

def initialize(account):
    add_history('hist', corlen)
    account.cutoff = 0.9
    account.prev_prc1 = 0
    account.prev_prc2 = 0
    account.prev_prcb = 0

def handle_data(account, data):
    stk1 = universe[0]
    stk2 = universe[1]

    prc1 = data[stk1]['closePrice']
    prc2 = data[stk2]['closePrice']
    prcb = data['HS300']['return']

    px1 = account.hist[stk1]['closePrice'].values
    px2 = account.hist[stk2]['closePrice'].values
    pxb = account.hist['HS300']['return'].values

    corval, pval = pearsonr(px1, px2)

    mov1, mov2 = adj(prc1, prc2, prcb, account.prev_prc1, account.prev_prc2, account.prev_prcb)
    amount =1e4 / prc2
    if (mov1 > 0) and (abs(corval) > account.cutoff):
        order(stk2, amount)
    elif (mov1 < 0) and (abs(corval) > account.cutoff):
        if (account.position.stkpos.get(stk2, 0) > amount):
            order(stk2, -amount)
        else:
            order_to(stk2, 0)

    amount =1e4 / prc1
    if (mov2 > 0) and (abs(corval) > account.cutoff):
        order(stk1, amount)
    elif (mov2 < 0) and (abs(corval) > account.cutoff):
        if (account.position.stkpos.get(stk1, 0) > amount):
            order(stk1, -amount)
        else:
            order_to(stk1, 0)

    account.prev_prc1 = prc1
    account.prev_prc2 = prc2
    account.prev_prcb = prcb


def dmv(curr, prev):
    delta = curr / prev - 1
    return delta

def adj(x, y, base, prev_x, prev_y, prev_base):
    dhs = dmv(base, prev_base)
    dx = dmv(x, prev_x) - dhs
    dy = dmv(y, prev_y) - dhs
    return (dx, dy)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
38.9%	6.3%	33.8%	0.54	1.41	25.0%	1.01	22.4%	--

```
min(bt.cash)

232096.85369499651
```

```
import pandas as pd
import numpy as np
from datetime import datetime

import quartz
import quartz.backtest as qb
import quartz.performance as qp
from quartz.api import *

from scipy.stats.stats import pearsonr

start = datetime(2013, 1, 1)                      # 回测起始时间
end   = datetime(2014, 12, 1)                      # 回测结束时间
benchmark = 'HS300'                                  # 使用沪深 300 作为
参考标准
capital_base = 1e6                                # 起始资金

corlen = 5

def initialize(account):                          # 初始化虚拟账户状态
    add_history('hist', corlen)
    account.cutoff = 0.9
    account.prev_prc1 = 0
    account.prev_prc2 = 0
    account.prev_prcb = 0
```

```
def handle_data(account, data):                  # 每个交易日的买入卖
出指令
    stk1 = universe[0]
    stk2 = universe[1]

    prc1 = data[stk1]['closePrice']
    prc2 = data[stk2]['closePrice']
    prcb = data['HS300']['return']

    px1 = account.hist[stk1]['closePrice'].values
    px2 = account.hist[stk2]['closePrice'].values
    pxb = account.hist['HS300']['return'].values

    corval, pval = pearsonr(px1, px2)

    mov1, mov2 = adj(prc1, prc2, prcb, account.prev_prc1, accoun
t.prev_prc2, account.prev_prcb)

    #amount = int( 0.08 * capital_base / prc2)
    amount =1e4 / prc2
    if (mov1 > 0) and (abs(corval) > account.cutoff):
        order(stk2, amount)
    elif (mov1 < 0) and (abs(corval) > account.cutoff):
        if (account.position.stkpos.get(stk2, 0) > amount):
            order(stk2, -amount)
        else:
            order_to(stk2, 0)

    #amount = int(0.08 * capital_base / prc1)
    amount =1e4 / prc1
    if (mov2 > 0) and (abs(corval) > account.cutoff):
        order(stk1, amount)
    elif (mov2 < 0) and (abs(corval) > account.cutoff):
        if (account.position.stkpos.get(stk1, 0) > amount):
            order(stk1, -amount)
        else:
            order_to(stk1, 0)

    account.prev_prc1 = prc1
    account.prev_prc2 = prc2
    account.prev_prcb = prcb


def dmv(curr, prev):
    delta = curr / prev - 1
    return delta

def adj(x, y, base, prev_x, prev_y, prev_base):
    dhs = dmv(base, prev_base)
    dx = dmv(x, prev_x) - dhs
    dy = dmv(y, prev_y) - dhs
    return (dx, dy)
```

```
pool_raw = pd.read_csv("po.pair.2012.csv")
pool = []
for i in range(len(pool_raw)):
    s1, s2 = pool_raw.loc[i].tolist()
    if [s2, s1] not in pool:
        pool.append([s1, s2])

outfile = []
for i, universe in enumerate(pool):
    print i
    try:
        bt = qb.backtest(start, end, benchmark, universe, capita
l_base, initialize = initialize, handle_data = handle_data)
        perf = qp.perf_parse(bt)
        outfile.append(universe + [perf["annualized_return"], pe
rf["sharpe"]])
    except:
        pass

keys = ['stock1', 'stock2', 'annualized_return', 'sharpe']
outdict = {}
outfile =  zip(*sorted(outfile, key=lambda x:x[2], reverse=True)
)
for i,k in enumerate(keys):
    outdict[k] = outfile[i]
outdict = pd.DataFrame(outdict).loc[:, keys]
outdict

['000066.XSHE', '000707.XSHE']
['000066.XSHE', '600117.XSHG']
['000066.XSHE', '600126.XSHG']
['000066.XSHE', '600819.XSHG']
['000089.XSHE', '600035.XSHG']
['000089.XSHE', '600037.XSHG']
['000089.XSHE', '600595.XSHG']
['000159.XSHE', '000967.XSHE']
['000159.XSHE', '600595.XSHG']
['000417.XSHE', '000541.XSHE']
['000417.XSHE', '000685.XSHE']
['000417.XSHE', '600875.XSHG']
['000425.XSHE', '000528.XSHE']
['000507.XSHE', '600391.XSHG']
['000541.XSHE', '000987.XSHE']
['000541.XSHE', '600330.XSHG']
['000541.XSHE', '600883.XSHG']
['000554.XSHE', '000707.XSHE']
['000559.XSHE', '600026.XSHG']
['000559.XSHE', '600126.XSHG']
['000559.XSHE', '600477.XSHG']
['000559.XSHE', '600581.XSHG']
['000559.XSHE', '601666.XSHG']
['000635.XSHE', '000707.XSHE']
```

```
 ['000635.XSHE', '600068.XSHG']
 ['000635.XSHE', '600117.XSHG']
 ['000635.XSHE', '600188.XSHG']
 ['000635.XSHE', '600295.XSHG']
 ['000635.XSHE', '600550.XSHG']
 ['000635.XSHE', '600819.XSHG']
 ['000635.XSHE', '601168.XSHG']
 ['000635.XSHE', '601233.XSHG']
 ['000650.XSHE', '600261.XSHG']
 ['000683.XSHE', '000936.XSHE']
 ['000683.XSHE', '600595.XSHG']
 ['000685.XSHE', '000988.XSHE']
 ['000685.XSHE', '601101.XSHG']
 ['000698.XSHE', '000949.XSHE']
 ['000707.XSHE', '000911.XSHE']
 ['000707.XSHE', '000969.XSHE']
 ['000707.XSHE', '000987.XSHE']
 ['000707.XSHE', '600117.XSHG']
 ['000707.XSHE', '600295.XSHG']
 ['000707.XSHE', '600550.XSHG']
 ['000707.XSHE', '600831.XSHG']
 ['000707.XSHE', '601168.XSHG']
 ['000707.XSHE', '601233.XSHG']
 ['000708.XSHE', '600327.XSHG']
 ['000709.XSHE', '601107.XSHG']
 ['000709.XSHE', '601618.XSHG']
 ['000717.XSHE', '600282.XSHG']
 ['000717.XSHE', '600307.XSHG']
 ['000717.XSHE', '600808.XSHG']
 ['000761.XSHE', '600320.XSHG']
 ['000761.XSHE', '600548.XSHG']
 ['000822.XSHE', '600117.XSHG']
 ['000830.XSHE', '600068.XSHG']
 ['000830.XSHE', '600320.XSHG']
 ['000830.XSHE', '600550.XSHG']
 ['000877.XSHE', '601519.XSHG']
 ['000898.XSHE', '600022.XSHG']
 ['000898.XSHE', '600808.XSHG']
 ['000911.XSHE', '600550.XSHG']
 ['000916.XSHE', '600033.XSHG']
 ['000916.XSHE', '600035.XSHG']
 ['000916.XSHE', '600126.XSHG']
 ['000930.XSHE', '600026.XSHG']
 ['000932.XSHE', '600569.XSHG']
 ['000933.XSHE', '600348.XSHG']
 ['000933.XSHE', '600595.XSHG']
 ['000936.XSHE', '600477.XSHG']
 ['000937.XSHE', '600348.XSHG']
 ['000937.XSHE', '600508.XSHG']
 ['000937.XSHE', '600997.XSHG']
 ['000937.XSHE', '601001.XSHG']
 ['000939.XSHE', '600819.XSHG']
 ['000967.XSHE', '600879.XSHG']
 ['000969.XSHE', '600831.XSHG']
 ['000973.XSHE', '600460.XSHG']
 ['000987.XSHE', '600636.XSHG']
 ['000987.XSHE', '600827.XSHG']
 ['000987.XSHE', '601001.XSHG']
 ['600008.XSHG', '600035.XSHG']
 ['600012.XSHG', '600428.XSHG']
 ['600020.XSHG', '600033.XSHG']
 ['600020.XSHG', '600035.XSHG']
 ['600026.XSHG', '600068.XSHG']
 ['600026.XSHG', '600089.XSHG']
 ['600026.XSHG', '600126.XSHG']
 ['600026.XSHG', '600307.XSHG']
 ['600026.XSHG', '600331.XSHG']
 ['600026.XSHG', '600375.XSHG']
 ['600026.XSHG', '600581.XSHG']
 ['600026.XSHG', '600963.XSHG']
 ['600026.XSHG', '601666.XSHG']
 ['600026.XSHG', '601898.XSHG']
 ['600033.XSHG', '600035.XSHG']
 ['600035.XSHG', '600126.XSHG']
 ['600035.XSHG', '600269.XSHG']
 ['600035.XSHG', '600307.XSHG']
 ['600035.XSHG', '600586.XSHG']
 ['600037.XSHG', '600327.XSHG']
 ['600068.XSHG', '600126.XSHG']
 ['600068.XSHG', '600269.XSHG']
 ['600068.XSHG', '600320.XSHG']
 ['600068.XSHG', '600550.XSHG']
 ['600068.XSHG', '601001.XSHG']
 ['600068.XSHG', '601666.XSHG']
 ['600089.XSHG', '600581.XSHG']
 ['600100.XSHG', '600117.XSHG']
 ['600117.XSHG', '600295.XSHG']
 ['600117.XSHG', '600339.XSHG']
 ['600117.XSHG', '601168.XSHG']
 ['600117.XSHG', '601233.XSHG']
 ['600126.XSHG', '600282.XSHG']
 ['600126.XSHG', '600327.XSHG']
 ['600126.XSHG', '600569.XSHG']
 ['600126.XSHG', '600581.XSHG']
 ['600126.XSHG', '600808.XSHG']
 ['600126.XSHG', '600963.XSHG']
 ['600160.XSHG', '600449.XSHG']
 ['600160.XSHG', '601216.XSHG']
 ['600160.XSHG', '601311.XSHG']
 ['600188.XSHG', '600295.XSHG']
 ['600188.XSHG', '601001.XSHG']
 ['600231.XSHG', '600282.XSHG']
 ['600269.XSHG', '601618.XSHG']
 ['600282.XSHG', '600307.XSHG']
 ['600282.XSHG', '600569.XSHG']
 ['600282.XSHG', '600808.XSHG']
```

```
['600282.XSHG', '600963.XSHG']
['600307.XSHG', '600581.XSHG']
['600307.XSHG', '600808.XSHG']
['600307.XSHG', '600963.XSHG']
['600320.XSHG', '600548.XSHG']
['600320.XSHG', '601600.XSHG']
['600330.XSHG', '600883.XSHG']
['600330.XSHG', '601268.XSHG']
['600331.XSHG', '600581.XSHG']
['600348.XSHG', '600508.XSHG']
['600348.XSHG', '600997.XSHG']
['600348.XSHG', '601001.XSHG']
['600368.XSHG', '600527.XSHG']
['600375.XSHG', '600581.XSHG']
['600391.XSHG', '601100.XSHG']
['600449.XSHG', '601311.XSHG']
['600449.XSHG', '601519.XSHG']
['600460.XSHG', '601908.XSHG']
['600477.XSHG', '600581.XSHG']
['600508.XSHG', '600546.XSHG']
['600508.XSHG', '600997.XSHG']
['600522.XSHG', '600973.XSHG']
['600550.XSHG', '600831.XSHG']
['600569.XSHG', '600808.XSHG']
['600569.XSHG', '600963.XSHG']
['600581.XSHG', '600963.XSHG']
['600581.XSHG', '601001.XSHG']
['600581.XSHG', '601168.XSHG']
['600581.XSHG', '601666.XSHG']
['600586.XSHG', '601268.XSHG']
['600595.XSHG', '601001.XSHG']
['600595.XSHG', '601168.XSHG']
['600595.XSHG', '601666.XSHG']
['600688.XSHG', '600871.XSHG']
['600785.XSHG', '600827.XSHG']
['600808.XSHG', '600963.XSHG']
['600827.XSHG', '601001.XSHG']
['600875.XSHG', '601001.XSHG']
['600883.XSHG', '601268.XSHG']
['601001.XSHG', '601101.XSHG']
['601001.XSHG', '601168.XSHG']
['601001.XSHG', '601666.XSHG']
['601101.XSHG', '601666.XSHG']
['601168.XSHG', '601666.XSHG']
```

	stock1	stock2	annualized_return	sharpe
0	000761.XSHE	600548.XSHG	0.489473	2.411514
1	000708.XSHE	600327.XSHG	0.447337	2.021270
2	600126.XSHG	600327.XSHG	0.438380	1.946916

3	000554.XSHE	000707.XSHE	0.431123	1.331038
4	000939.XSHE	600819.XSHG	0.409471	1.919758
5	600026.XSHG	600963.XSHG	0.408791	1.681338
6	600037.XSHG	600327.XSHG	0.395624	1.691877
7	600808.XSHG	600963.XSHG	0.391988	1.724114
8	000559.XSHE	600126.XSHG	0.389043	1.413595
9	000761.XSHE	600320.XSHG	0.384325	1.807262
10	600126.XSHG	600963.XSHG	0.378064	1.662569
11	600126.XSHG	600808.XSHG	0.375825	1.513791
12	000936.XSHE	600477.XSHG	0.375135	1.707097
13	000930.XSHE	600026.XSHG	0.372924	1.524350
14	600320.XSHG	600548.XSHG	0.372499	2.083496
15	000507.XSHE	600391.XSHG	0.365637	1.813873
16	000559.XSHE	601666.XSHG	0.350235	0.925901
17	600012.XSHG	600428.XSHG	0.327834	1.722317
18	000916.XSHE	600033.XSHG	0.327795	1.406093
19	600035.XSHG	600126.XSHG	0.326167	1.442674
20	600827.XSHG	601001.XSHG	0.322705	0.957791
21	000717.XSHE	600808.XSHG	0.320737	1.293439
22	000559.XSHE	600477.XSHG	0.306670	1.218095
23	000685.XSHE	000988.XSHE	0.302593	1.692933
24	000683.XSHE	000936.XSHE	0.301804	1.550496
25	000559.XSHE	600026.XSHG	0.295510	1.279449
26	600269.XSHG	601618.XSHG	0.294215	1.486413
27	600026.XSHG	600126.XSHG	0.293884	1.441490
28	600068.XSHG	600126.XSHG	0.289457	1.261351
29	000159.XSHE	600595.XSHG	0.288982	0.946365
30	600020.XSHG	600033.XSHG	0.288243	1.489764
31	600126.XSHG	600569.XSHG	0.287607	1.371374
32	000635.XSHE	600819.XSHG	0.285135	1.364688

33	600068.XSHG	600320.XSHG	0.273513	1.262845
34	600785.XSHG	600827.XSHG	0.272658	0.842093
35	000089.XSHE	600595.XSHG	0.269903	1.256524
36	000898.XSHE	600808.XSHG	0.269717	1.074201
37	000717.XSHE	600282.XSHG	0.267478	1.270872
38	600282.XSHG	600808.XSHG	0.266402	1.181157
39	000916.XSHE	600035.XSHG	0.264325	1.079520
40	000089.XSHE	600037.XSHG	0.264201	1.467101
41	600026.XSHG	600068.XSHG	0.263959	1.107977
42	600026.XSHG	600331.XSHG	0.261025	0.977858
43	600020.XSHG	600035.XSHG	0.260176	1.119975
44	600569.XSHG	600963.XSHG	0.260006	1.154372
45	600307.XSHG	600963.XSHG	0.258488	1.322409
46	000898.XSHE	600022.XSHG	0.258246	1.100292
47	600282.XSHG	600963.XSHG	0.257496	1.175741
48	600307.XSHG	600808.XSHG	0.256071	1.062023
49	600126.XSHG	600282.XSHG	0.255657	1.318676
50	600033.XSHG	600035.XSHG	0.255634	1.055682
51	000709.XSHE	601618.XSHG	0.253129	1.062565
52	600026.XSHG	600307.XSHG	0.253119	0.985825
53	600026.XSHG	600375.XSHG	0.250793	1.063874
54	000066.XSHE	600126.XSHG	0.247493	1.469341
55	000830.XSHE	600320.XSHG	0.247001	1.370327
56	600320.XSHG	601600.XSHG	0.246534	0.966634
57	000717.XSHE	600307.XSHG	0.245805	1.202750
58	000417.XSHE	000685.XSHE	0.245031	1.189700
59	600330.XSHG	600883.XSHG	0.243437	1.086147
...	...	...	...	

```
174 rows × 4 columns
```

```
a = list(outfile[2])
'percentage of outperform HS300: %f' % (1.*len([x for x in a if x>0.117]) / len(a))

'percentage of outperform HS300: 0.741379'
```

2.2 期现套利 • 通过股指期货的期现差与 ETF 对冲套利

> 来源：https://uqer.io/community/share/55aa5aa8f9f06c56de1b53bb

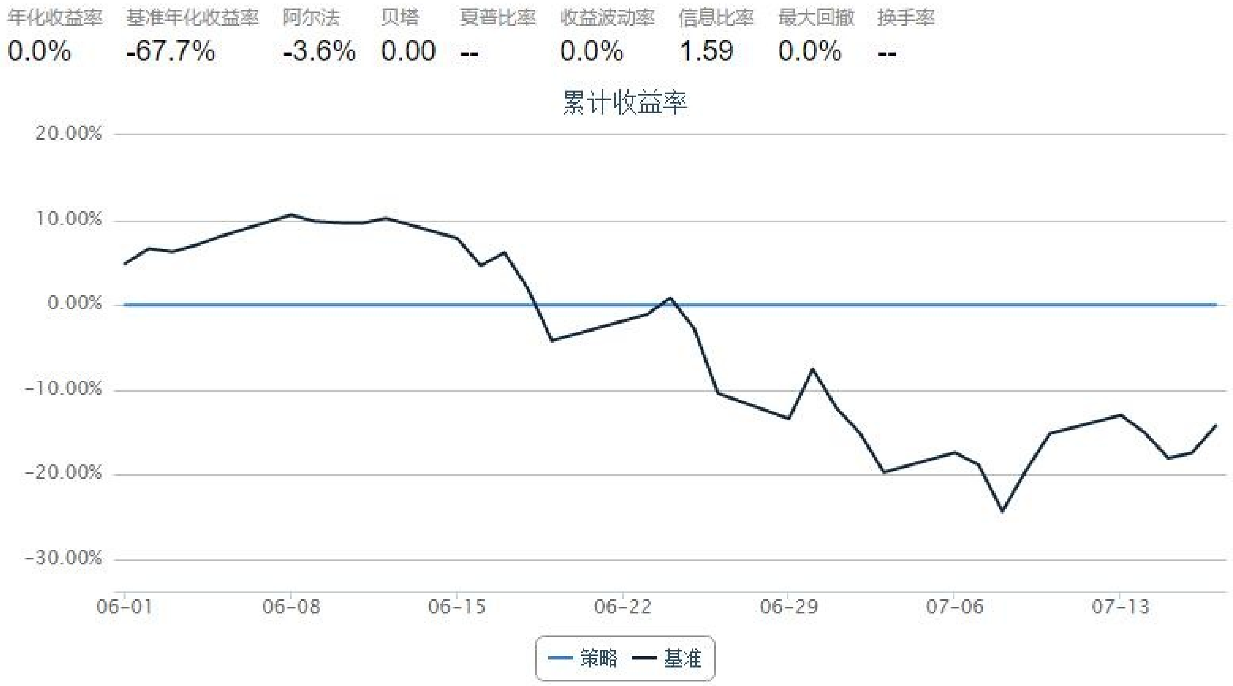
通过股指期货（IF1507）的期现差会围绕沪深300指数上下波动的原理，当期现差扩大到一定程度后，使用股指期货和ETF向对冲，以套取期现差的稳定利润。

只通过IF1507验证了一个基本可能性。还有很多细节需要完善。

还有，不知道如何通过order方法卖空股指期货和etf，了解的大神可以帮我解决一下~~

```
import pandas as pd

start = datetime(2015, 6, 1)                     # 回测起始时间
end   = datetime(2015, 7, 17)                     # 回测结束时间
benchmark = 'HS300'                                 # 策略参考标准
universe = ['510300.XSHG','IF1507.CCFX']      # 股票池
capital_base = 1000000                           # 起始资金


maxQxc = 0.0 #最大期现差
isOrder = False #是否已经买入对冲
direction = False # 买入方向
buyPosition = 0 # 买入点位
total = 0
def initialize(account):                         # 初始化虚拟账户状态

    pass

def handle_data(account):                        # 每个交易日的买入卖出指令

#     log.debug(account.current_date)
#     lowToNow = DataAPI.MktEqudAdjGet(secID = '510300', field =
 ['closePrice'],beginDate = lowDate, endDate = nowDate)
    global maxQxc,isOrder,direction,buyPosition,total

#     获取股指期货的行情数据
    if1507 = DataAPI.MktFutdGet(ticker="IF1507",
                               beginDate=account.current_dat
e.strftime("%Y%m%d"),endDate=account.current_date.strftime("%Y%m
%d"))
#     获取etf基金的行情数据
    etf300 =  DataAPI.MktFunddGet(ticker = '510300',
                                beginDate=account.current_da
te.strftime("%Y%m%d"),endDate=account.current_date.strftime("%Y%
m%d"))
#     获取沪深300的行情数据
```

```
    hs300 = DataAPI.MktIdxdGet(ticker="000300",
                       beginDate=account.current_date.strftime("
%Y%m%d"),endDate=account.current_date.strftime("%Y%m%d"))

#     计算期现差
    qxc = if1507['closePrice'].iloc[0] - hs300['closeIndex'].ilo
c[0]

#     log.debug((hs300['indexID']) + "/" + (if1507['secID']) + "
/" + (etf300['secID']))
#     log.debug(str(qxc) + "/" + str(hs300['closeIndex'].iloc[0]
) + "/" + str(if1507['closePrice'].iloc[0]) + "/" + str(etf300['
closePrice'].iloc[0]))
#     保存期现差最大值，为后面判断买点时用
    if(abs(qxc) > abs(maxQxc)):
        maxQxc = qxc
#     判断期现差绝对值大于100，并开始从高点回落，同时账户是空仓，就买入
    if(abs(maxQxc) > 100 and abs(qxc) < abs(maxQxc) and not isOr
der ):
        direction = qxc > 0
        buyPosition = qxc
        isOrder = True
#         if direction:
#             正期现差时
#             order(account.universe[1],-1) # 卖空一手期指
#             order(account.universe[0],3000) # 买入30万基金
#         else:
#             负期现差时
#             order(account.universe[1],1) # 卖空一手期指
#             order(account.universe[0],-3000) # 买入30万基金

        log.debug("买入点位 = "+ str(if1507['closePrice'].iloc[0]
) + "，期现差 = " + str(qxc))

#     判断已经持仓，并期现差已经反转，就卖出
    if(isOrder):
        if(direction):
            if(qxc < 0):
                earnings = buyPosition - qxc
                isOrder = False
                maxQxc = 0
#                 order(account.universe[1],1) # 卖空一手期指
#                 order(account.universe[0],-3000) # 买入30万基金
                account.cash = account.cash + abs(earnings)* 300
                log.debug("卖出盈利 = " + str(abs(earnings)* 300)
)


        else:
            if(qxc > 0):
                earnings = buyPosition - qxc
                isOrder = False
                maxQxc = 0
#                 order(account.universe[1],-1) # 卖空一手期指
#                 order(account.universe[0],3000) # 买入30万基金
                account.cash = account.cash + abs(earnings)* 300
                log.debug("卖出盈利 = " + str(abs(earnings) * 300
 ) + "，卖出时期现差 = " + str(qxc))

#     for stock in account.universe:
#         log.debug(stock)
#     log.debug(account.cash)
    return
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
0.0%	-67.7%	-3.6%	0.00	--	0.0%	1.59	0.0%	--

```
2015-06-01 [DEBUG] 1000000.0
2015-06-02 [DEBUG] 买入点位 = 5269.6，期现差 = 107.73
2015-06-02 [DEBUG] 1000000.0
2015-06-03 [DEBUG] 1000000.0
2015-06-04 [DEBUG] 1000000.0
2015-06-05 [DEBUG] 1000000.0
2015-06-08 [DEBUG] 卖出盈利 = 38004.0
2015-06-08 [DEBUG] 1038004.0
2015-06-09 [DEBUG] 1000000.0
2015-06-10 [DEBUG] 1000000.0
2015-06-11 [DEBUG] 1000000.0
2015-06-12 [DEBUG] 1000000.0
2015-06-15 [DEBUG] 1000000.0
2015-06-16 [DEBUG] 1000000.0
2015-06-17 [DEBUG] 1000000.0
2015-06-18 [DEBUG] 1000000.0
2015-06-19 [DEBUG] 1000000.0
2015-06-23 [DEBUG] 1000000.0
2015-06-24 [DEBUG] 1000000.0
2015-06-25 [DEBUG] 1000000.0
2015-06-26 [DEBUG] 1000000.0
2015-06-29 [DEBUG] 1000000.0
2015-06-30 [DEBUG] 买入点位 = 4381.4，期现差 = -91.598
2015-06-30 [DEBUG] 1000000.0
2015-07-01 [DEBUG] 1000000.0
2015-07-02 [DEBUG] 卖出盈利 = 34080.6，卖出时期现差 = 22.004
2015-07-02 [DEBUG] 1034080.6
2015-07-03 [DEBUG] 1000000.0
2015-07-06 [DEBUG] 1000000.0
2015-07-07 [DEBUG] 1000000.0
2015-07-08 [DEBUG] 买入点位 = 3463.4，期现差 = -199.638
2015-07-08 [DEBUG] 1000000.0
2015-07-09 [DEBUG] 1000000.0
2015-07-10 [DEBUG] 卖出盈利 = 77904.6，卖出时期现差 = 60.044
2015-07-10 [DEBUG] 1077904.6
2015-07-13 [DEBUG] 1000000.0
2015-07-14 [DEBUG] 买入点位 = 4001.6，期现差 = -110.549
2015-07-14 [DEBUG] 1000000.0
2015-07-15 [DEBUG] 1000000.0
2015-07-16 [DEBUG] 卖出盈利 = 36357.9，卖出时期现差 = 10.644
2015-07-16 [DEBUG] 1036357.9
2015-07-17 [DEBUG] 1000000.0
```

三 事件驱动

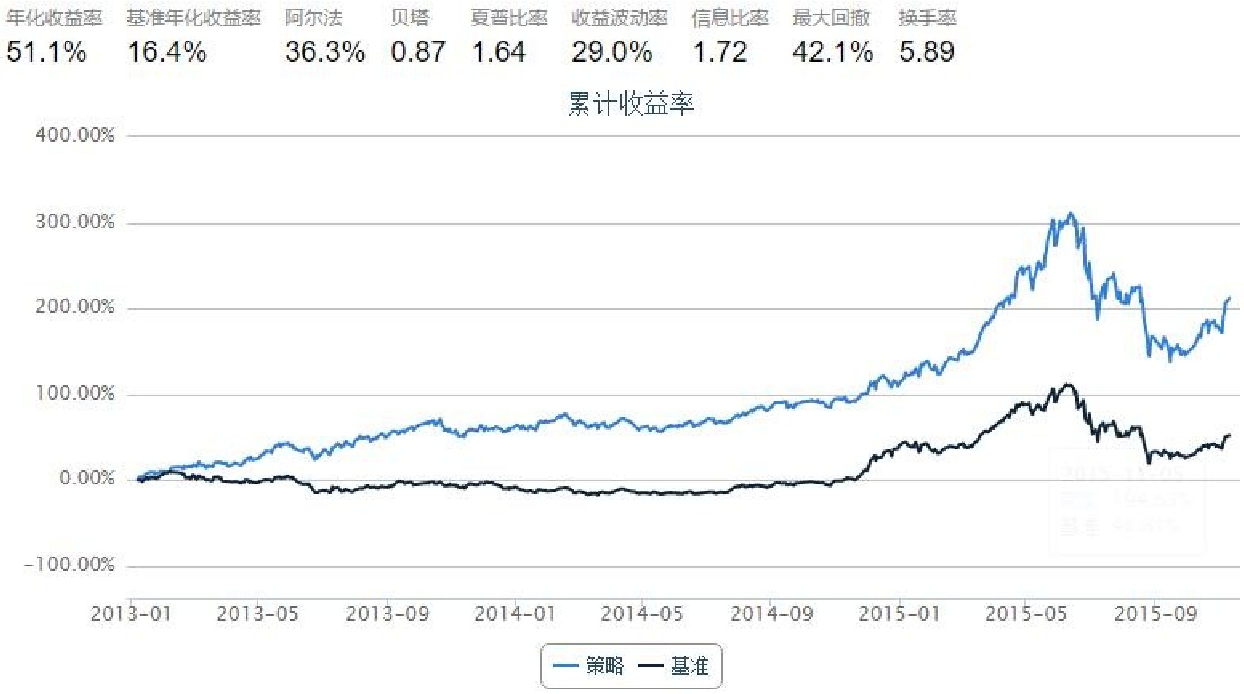
3.1 盈利预增

盈利预增事件

> 来源：https://uqer.io/community/share/5642c69af9f06c78612190b5

每次建仓等权重买入上季度净利润预增(盈利且盈利增加)的股票

```
import pandas as pd
import numpy as np
from CAL.PyCAL import *
from pandas import DataFrame,Series
from datetime import datetime, timedelta
start = '2013-01-01'                       # 回测起始时间
end = (datetime.today() - timedelta(days=1)).strftime('%Y%m%d')
# 截止日期
benchmark = 'HS300'                        # 策略参考标准
universe = set_universe('HS300')             # 证券池，支持股票和基金

capital_base = 1000000                     # 起始资金
freq = 'd'                                 # 策略类型，'d'表示日间
策略使用日线回测，'m'表示日内策略使用分钟线回测
refresh_rate = 63                          # 调仓频率，表示执行hand
le_data的时间间隔，若freq = 'd'时间间隔的单位为交易日，若freq = 'm'时间
间隔为分钟
cal = Calendar('China.SSE')

def initialize(account):                   # 初始化虚拟账户状态
    pass

def handle_data(account):                  # 每个交易日的买入卖出指令


    today = account.current_date.strftime('%Y%m%d')
    yesterday = cal.advanceDate(account.current_date, '-1B', Biz
DayConvention.Following).strftime('%Y%m%d')
    yester_refresh_day = cal.advanceDate(account.current_date, '
-62B' , BizDayConvention.Following).strftime('%Y%m%d')
    total_money = account.referencePortfolioValue
    prices = account.referencePrice
    # 去除新上市或复牌的股票
    opn = account.get_attribute_history('openPrice', 1)
    account.universe = [s for s in account.universe if not (np.i
snan(opn.get(s, 0)[0]) or opn.get(s, 0)[0] == 0 )]

    buylist =[]
    for s in  account.universe :
        try :
            temp=DataAPI.FdmtEfGet(secID = s,forecastType ='22',
publishDateBegin= yester_refresh_day , publishDateEnd = yesterda
y,field=['secID','publishDate','NIncAPChgrLL', 'NIncAPChgrUPL'],
pandas="1")
            buylist.append(s)
        except :
            continue
    sell_list = [x for x in account.valid_secpos if x not in buy
list]
    for s in sell_list :
        order_to(s,0)
    for s in buylist :
        order_to(s, int(total_money*0.99/len(buylist)/prices[s]/
100)*100)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
51.1%	16.4%	36.3%	0.87	1.64	29.0%	1.72	42.1%	5.89

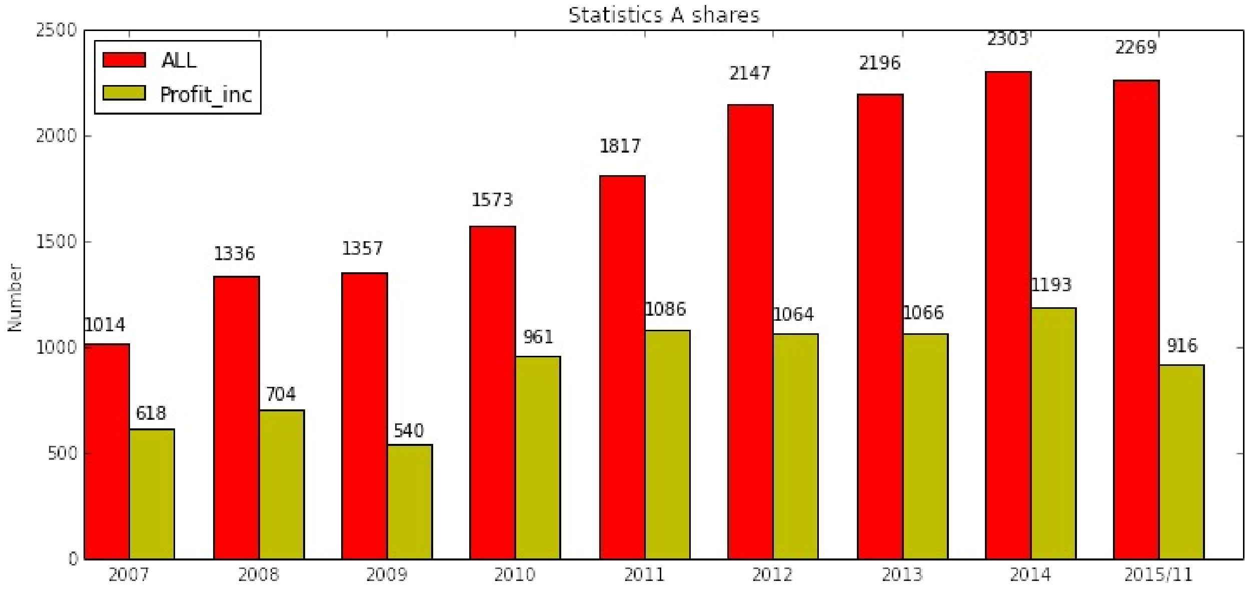
```
# 统计代码 不用看啰
from pandas import DataFrame
data = DataFrame()
for s in range(len(set_universe('A'))/100 + 1) :
    if s == len(set_universe('A'))/100 :
        temp_list = set_universe('A')[s*100:]
    else :
        temp_list = set_universe('A')[s*100:(s+1)*100]
    try:
        if not temp_list == 0 :
            data_temp = DataAPI.FdmtEfGet(secID = temp_list,fiel
d=['secID','publishDate','NIncAPChgrLL', 'NIncAPChgrUPL'],pandas=
"1")
    except :
        print '错误！'
    data = pd.concat([data,data_temp])
data['publishDate'] = pd.to_datetime(data['publishDate'])
list1 = []
```

```
data07 = data[data['publishDate'] < '20080101']
data07.drop_duplicates('secID' , inplace = True)
list1.append(len(data07))
data08 = data[(data['publishDate'] < '20090101') & (data['publis
hDate'] >= '20080101')]
data08.drop_duplicates('secID' , inplace = True)
list1.append(len(data08))
data09 = data[(data['publishDate'] < '20100101') & (data['publis
hDate'] >= '20090101')]
data09.drop_duplicates('secID' , inplace = True)
list1.append(len(data09))
data10 = data[(data['publishDate'] < '20110101') & (data['publis
hDate'] >= '20100101')]
data10.drop_duplicates('secID' , inplace = True)
list1.append(len(data10))
data11 = data[(data['publishDate'] < '20120101') & (data['publis
hDate'] >= '20110101')]
data11.drop_duplicates('secID' , inplace = True)
list1.append(len(data11))
data12 = data[(data['publishDate'] < '20130101') & (data['publis
hDate'] >= '20120101')]
data12.drop_duplicates('secID' , inplace = True)
list1.append(len(data12))
data13 = data[(data['publishDate'] < '20140101') & (data['publis
hDate'] >= '20130101')]
data13.drop_duplicates('secID' , inplace = True)
list1.append(len(data13))
data14 = data[(data['publishDate'] < '20150101') & (data['publis
hDate'] >= '20140101')]
data14.drop_duplicates('secID' , inplace = True)
list1.append(len(data14))
data15 = data[(data['publishDate'] < '20160101') & (data['publis
hDate'] >= '20150101')]
data15.drop_duplicates('secID' , inplace = True)
list1.append(len(data15))

data2 = DataFrame()
for s in range(len(set_universe('A'))/100 + 1) :
    if s == len(set_universe('A'))/100 :
        temp_list = set_universe('A')[s*100:]
    else :
        temp_list = set_universe('A')[s*100:(s+1)*100]
    try:
        if not temp_list == 0 :
            data_temp = DataAPI.FdmtEfGet(secID = temp_list,fore
castType= '22' ,field=['secID','publishDate','NIncAPChgrLL', 'NI
ncAPChgrUPL'],pandas="1")
    except :
        print '错误！'
    data2 = pd.concat([data2,data_temp])
data2['publishDate'] = pd.to_datetime(data2['publishDate'])
list2 = []
data07 = data2[data2['publishDate'] < '20080101']
```

```
data07.drop_duplicates('secID' , inplace = True)
list2.append(len(data07))
data08 = data2[(data2['publishDate'] < '20090101') & (data2['publishDate'] >= '20080101')]
data08.drop_duplicates('secID' , inplace = True)
list2.append(len(data08))
data09 = data2[(data2['publishDate'] < '20100101') & (data2['publishDate'] >= '20090101')]
data09.drop_duplicates('secID' , inplace = True)
list2.append(len(data09))
data10 = data2[(data2['publishDate'] < '20110101') & (data2['publishDate'] >= '20100101')]
data10.drop_duplicates('secID' , inplace = True)
list2.append(len(data10))
data11 = data2[(data2['publishDate'] < '20120101') & (data2['publishDate'] >= '20110101')]
data11.drop_duplicates('secID' , inplace = True)
list2.append(len(data11))
data12 = data2[(data2['publishDate'] < '20130101') & (data2['publishDate'] >= '20120101')]
data12.drop_duplicates('secID' , inplace = True)
list2.append(len(data12))
data13 = data2[(data2['publishDate'] < '20140101') & (data2['publishDate'] >= '20130101')]
data13.drop_duplicates('secID' , inplace = True)
list2.append(len(data13))
data14 = data2[(data2['publishDate'] < '20150101') & (data2['publishDate'] >= '20140101')]
data14.drop_duplicates('secID' , inplace = True)
list2.append(len(data14))
data15 = data2[(data2['publishDate'] < '20160101') & (data2['publishDate'] >= '20150101')]
data15.drop_duplicates('secID' , inplace = True)
list2.append(len(data15))
```

- 红柱表示年间发布过业绩预告的公司数量
- 黄柱表示年间发布过盈利预增的公司数量

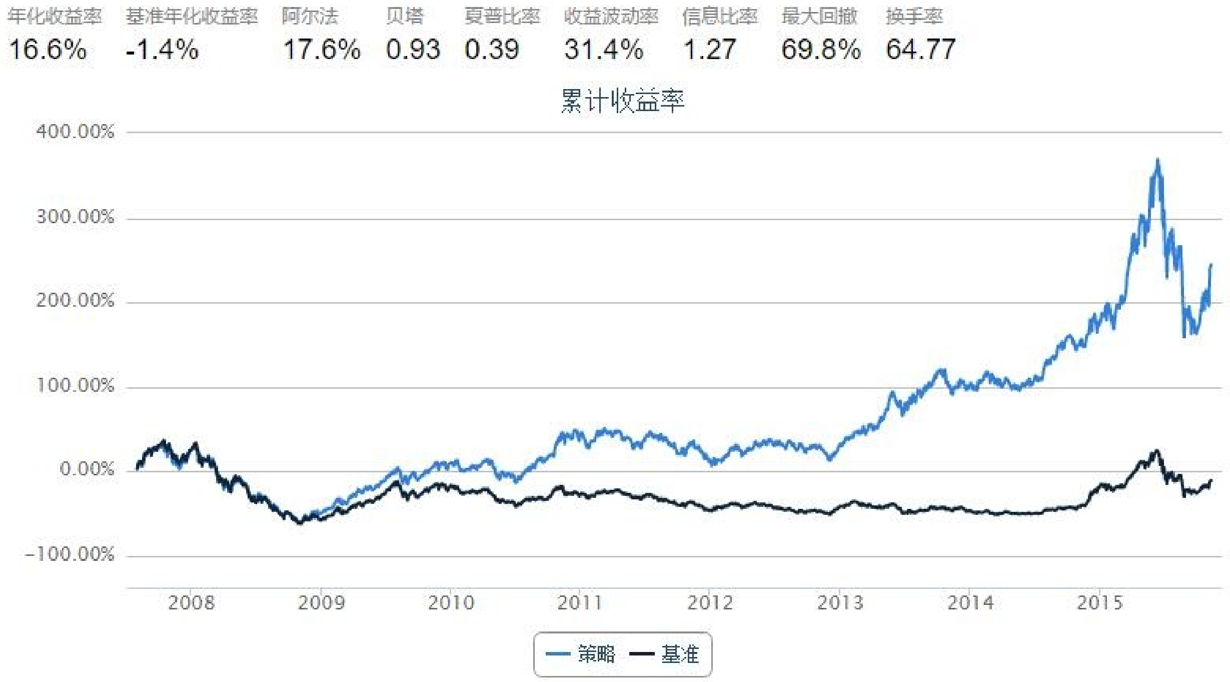
```
# plot Statistics
import numpy as np
import matplotlib.pyplot as plt
N = 9
ind = np.arange(N)
width = 0.35
fig = plt.figure(figsize=(12,12))
ax = fig.add_subplot(211)
rects1 = ax.bar(ind, list1, width, color='r')
rects2 = ax.bar(ind+width, list2, width, color='y')
# add some
ax.set_ylabel('Number')
ax.set_title('Statistics A shares')
ax.set_xticks(ind+width)
ax.set_xticklabels( ('2007', '2008', '2009', '2010', '2011','201
2','2013','2014','2015/11') )
ax.legend((rects1[0], rects2[0]), ('ALL', 'Profit_inc') , loc =
'1')
def autolabel(rects):
    # attach some text labels
    for rect in rects:
        height = rect.get_height()
        ax.text(rect.get_x()+rect.get_width()/2., 1.05*height, '
%d'%int(height),
                ha='center', va='bottom')
autolabel(rects1)
autolabel(rects2)
plt.show()
```



- 构造一个事件驱动策略看下盈利预增效应是否长期有效
- 策略思想 ： 以企业净利润预增预报为事件驱动，卖出同期持仓（上个换仓期到本次换仓）中收益率最low的股票，买入盈利预增的股票。

- 回测区间 ： 07年 —— 今

```
import pandas as pd
import numpy as np
from CAL.PyCAL import *
from pandas import DataFrame,Series
from datetime import datetime, timedelta
start = '20070801'                       # 回测起始时间
end = (datetime.today() - timedelta(days=1)).strftime('%Y%m%d')
# 截止日期

benchmark = 'HS300'                         # 策略参考标准
universe = set_universe('HS300')            # 证券池，支持股票和基金
capital_base = 1000000                      # 起始资金
freq = 'd'                                  # 策略类型，'d'表示日间
策略使用日线回测，'m'表示日内策略使用分钟线回测
refresh_rate = 1                            # 调仓频率，表示执行hand
le_data的时间间隔，若freq = 'd'时间间隔的单位为交易日，若freq = 'm'时间
间隔为分钟
cal = Calendar('China.SSE')

def initialize(account):                    # 初始化虚拟账户状态
    account.last_refreshtime = start

def handle_data(account):                   # 每个交易日的买入卖出指令


    today = account.current_date.strftime('%Y%m%d')
    yesterday = cal.advanceDate(account.current_date, '-1B', Biz
DayConvention.Following).strftime('%Y%m%d')
    last_day  = (account.current_date - timedelta(days=1)).strft
ime('%Y%m%d')
    yester_refresh_day = cal.advanceDate(account.current_date, '
-62B' , BizDayConvention.Following).strftime('%Y%m%d')
    total_money = account.referencePortfolioValue
    prices = account.referencePrice
    buylist =[]
    # 去除新上市或复牌的股票
    opn = account.get_attribute_history('openPrice', 1)
    account.universe = [s for s in account.universe if not (np.i
snan(opn.get(s, 0)[0]) or opn.get(s, 0)[0] == 0 )]

    # 初始建仓(选当前净利润最高的20只股票)：
    if len(account.valid_secpos) == 0 :
        # 净利润增长率
        NetProfitGrowRate = DataAPI.MktStockFactorsOneDayGet(tra
deDate=yesterday,secID=account.universe,field=u"secID,NetProfitG
rowRate",pandas="1")
        NetProfitGrowRate = NetProfitGrowRate.sort('NetProfitGro
wRate',ascending = False).drop_duplicates('secID')
        buylist = list(NetProfitGrowRate['secID'].values[0:20])
        for s in buylist :
            order_to(s, int(total_money*0.99/len(buylist)/prices[s]/100)*100)
        account.last_refreshtime = today
        return

    # 获取业绩预增的股票,最多取20只
    try :
        temp = DataAPI.FdmtEfGet(secID = account.universe , forecastType ='22',publishDateBegin= yesterday , publishDateEnd = last_day , field=['secID','publishDate','NIncAPChgrLL', 'NIncAPChgrUPL'],pandas="1")
        temp['meanGrowRate'] = (temp['NIncAPChgrLL'] + temp['NIncAPChgrUPL']) / 2
        temp.sort('meanGrowRate', ascending=False).drop_duplicates('secID' ,inplace = True)
        buylist = list(temp['secID'].values[0:20])
    except :
        return

    change_stock = [x for x in account.valid_secpos if x not in buylist]
    buylist = [x for x in buylist if x not in account.valid_secpos]
    # 换仓
    price1 = DataAPI.MktEqudAdjGet(secID=change_stock, tradeDate=account.last_refreshtime , field=u"secID,openPrice",pandas="1")
    price2 = DataAPI.MktEqudAdjGet(secID=change_stock, tradeDate=yesterday , field=u"secID,closePrice",pandas="1")
    # 计算持仓股这段时间的涨幅
    price1['stock_returns'] = price2['closePrice'] / price1['openPrice']
    price1.sort('stock_returns',ascending = True,inplace = True)
    # 剔除上个换仓日到现在最挫的股票  :
    sell_list = price1['secID'].values[0:len(buylist)]
    for s in sell_list :
        account.cash += prices[s] * account.valid_secpos.get(s)
        order_to(s,0)
    for s in buylist :
        order(s, int(account.cash / len(buylist)/prices[s]/100)*100)
    #更变最新调仓日期
    account.last_refreshtime = today
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
16.6%	-1.4%	17.6%	0.93	0.39	31.4%	1.27	69.8%	64.77

- 从回测结果来看业绩预增事件在熊市中效应并不显著，12年以前跑得也没之后的好。
- 从统计数据来看，07年后逐年有更多的上市公司以业绩预告的方式与公司股东进行互动交流,股民朋友们也越来越注重业绩预告，并在二级市场给予及时的回应。
- 呀！才发现回测有新功能了——回测详情，有更多统计信息了！！！！好棒！
- PS：从回测详情看好多业绩预增的公司报告发布后第二天开板涨停啊！看回测详情数据~新技能get.

事件驱动策略示例——盈利预增

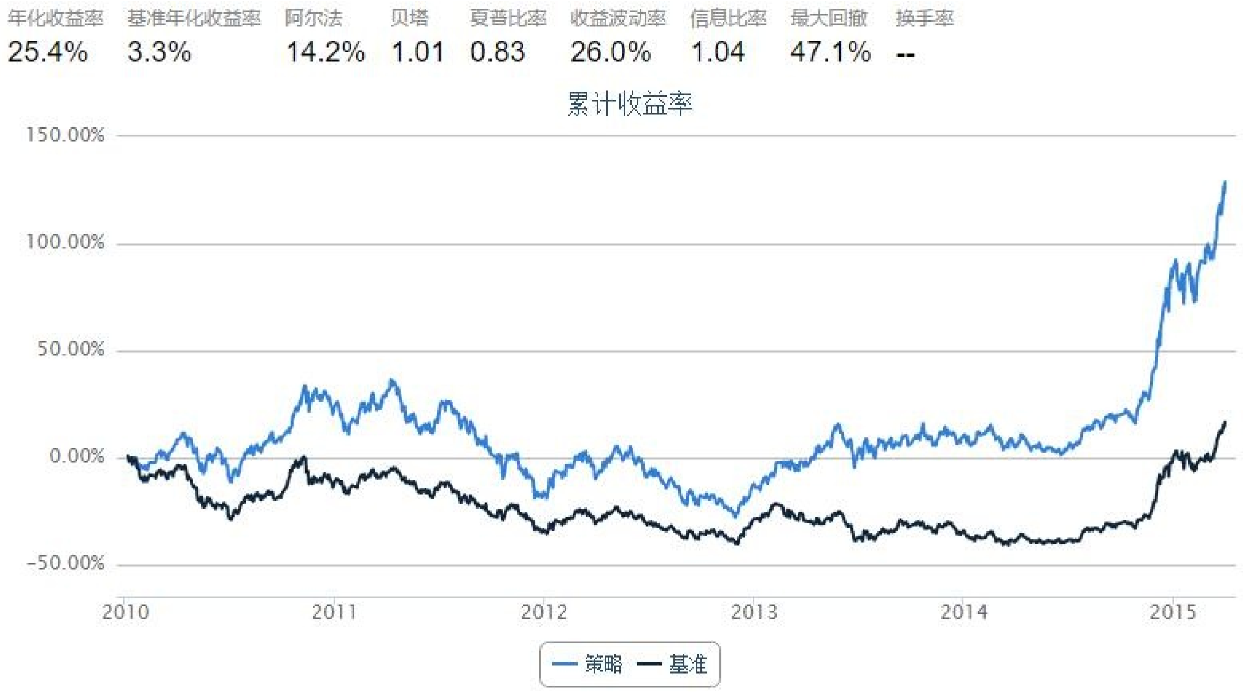
> 来源：https://uqer.io/community/share/54d972c1f9f06c276f651a72

策略思路

- 从DataAPI中获取沪深300成分股的盈利预增事件数据
- 每个交易日，将昨天发布盈利预增事件公司加入买入列表
- 根据调仓限制和买入列表进行调仓
- 调仓限制
    - (1) 股票持有不超过50只
    - (2) 一旦买入，持有40个交易日
    - (3) 仅当持有数量低于50只时才买入股票，补满50只

```
import pandas as pd
from datetime import datetime
from functools import partial

fields_ef = ['secID', 'publishDate']
get_data = partial(DataAPI.FdmtEfGet, forecastType = 22, field = fields_ef)  # forecastType 22: 盈利预增

data_ef = []
for stock in set_universe('HS300'):
    try:
        if len(data_ef):
            data_ef = data_ef.append(get_data(secID = stock))
        else:
            data_ef = get_data(secID = stock)
    except:
        pass

data_ef['publishDate'] = pd.to_datetime(data_ef['publishDate'])
data_ef = data_ef.sort(columns = 'publishDate')
data_ef = data_ef[data_ef.publishDate >= datetime(2010, 1, 1)]
```

```
start = '2010-01-01'
end   = '2015-04-01'
benchmark = 'HS300'
universe = set_universe('HS300')
capital_base = 1000000
longest_history = 1

max_t   = 40       # 持仓时间
max_n   = 50       # 持仓数量

def initialize(account):
    account.hold_period = {}

def handle_data(account):
    yesterday = account.get_symbol_history('tradeDate', 1)[0]
    data_sub = data_ef[data_ef.publishDate == yesterday]

    if len(data_sub):
        buylist = [s for s in data_sub['secID'].tolist() if s in account.universe]
        rebalance(account, buylist)

def rebalance(account, buylist):
    n = 0
    for stock, t in account.hold_period.items():
        if t == max_t:
            order_to(stock, 0)
            del account.hold_period[stock]
        else:
            account.hold_period[stock] += 1
            n += 1
    if n == max_n or buylist == []:
        return

    b = max_n - n
    buylist = [s for s in buylist if s not in account.hold_period]
    for stock in buylist[:b]:
        order(stock, account.referencePortfolioValue / b / account.referencePrice[stock])
        account.hold_period[stock] = 0
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
25.4%	3.3%	14.2%	1.01	0.83	26.0%	1.04	47.1%	--

3.2 分析师推荐 • 分析师的金手指？

> 在我们的观点中，分析师对股票的评级以及EPS的估计，更多的是对该之股票过去一段时间表现的总结，并没有明确的预测未来的能力。鉴于分析师估计的延迟特点，在我们的策略中我们将分析师估计作为反向指标使用。粗略的说，在固定的期限内，我们买入分析师调低预期的股票，卖出分析师调高预期的股票。

本策略的参数如下：

- 起始日期：2011年1月1日
- 结束日期：2015年3月19日
- 股票池：沪深300
- 业绩基准：沪深300
- 起始资金：100000元
- 调仓周期：3个月

本策略使用的主要数据API有：

这里我们使用了来自于第三方朝阳永续的数据API（需要在数据商城中购买）

- `CGRDReportGGGet` 获取朝阳永续分析师一致评级
- `CESTReportGGGet` 获取朝阳永续分析师一致预期

朝阳永续分析师分析数据相关链接

```
import pandas as pd

start = datetime(2011,1, 1)                # 回测起始时间
end   = datetime(2015, 3, 19)                # 回测结束时间
benchmark = 'HS300'                            # 策略参考标准
universe = set_universe('HS300')    # 股票池
#universe = ['600000.XSHG', '000001.XSHE']
capital_base = 100000                       # 起始资金
commission = Commission(0.0,0.0)
longest_history = 1

def CGRDwithBatch(universe, batch, startDate, endDate):
    res = pd.DataFrame()

    totalLength = len(universe)
    count = 0
    while totalLength > batch:
        tmp = DataAPI.GG.CGRDReportGGGet(secID = universe[count
* batch : (count + 1) * batch], BeginPubDate = startDate, EndPub
Date = endDate)
        count += 1
        totalLength -= batch
        res = res.append(tmp)

    tmp = DataAPI.GG.CGRDReportGGGet(secID = universe[(count * b
atch):], BeginPubDate = startDate, EndPubDate = endDate)
    res = res.append(tmp)

    return res

def CESTwithBatch(universe, batch, startDate, endDate):
    res = pd.DataFrame()

    totalLength = len(universe)
    count = 0
    while totalLength > batch:
        tmp = DataAPI.GG.CESTReportGGGet(secID = universe[count
* batch : (count + 1) * batch], BeginPubDate = startDate, EndPub
Date = endDate)
        count += 1
        totalLength -= batch
        res = res.append(tmp)

    tmp = DataAPI.GG.CGRDReportGGGet(secID = universe[(count * b
atch):], BeginPubDate = startDate, EndPubDate = endDate)
    res = res.append(tmp)

    return res

def MktEqudwithBatch(universe, batch, startDate, endDate):
    res = pd.DataFrame()

    totalLength = len(universe)
    count = 0
    while totalLength > batch:
        tmp = DataAPI.MktEqudGet(secID = universe[count * batch
: (count + 1) * batch], beginDate = startDate, endDate = endDate
)
        count += 1
        totalLength -= batch
        res = res.append(tmp)

    tmp = DataAPI.MktEqudGet(secID = universe[count * batch : (c
ount + 1) * batch], beginDate = startDate, endDate = endDate)
    res = res.append(tmp)

    return res


def regressionTesting(universe, startDate, endDate):
    import statsmodels.api as sm

    res1 = CGRDwithBatch(universe, 50, startDate, endDate).sort(
'publishDate')
    res2 = CESTwithBatch(universe, 50, startDate, endDate).sort(
'publishDate')
    res1 = res1[res1.RatingType == 1]
    res2 = res2[res2.PnetprofitType == 1]

    # got expRating change
    lastRating = res1.groupby('secID').last()
    firstRating = res1.groupby('secID').first()
    lastRating['previousRating'] = firstRating.Rating
    lastRating['chg_exp'] = lastRating.Rating / firstRating.Rati
ng - 1.0
    lowerP = lastRating['chg_exp'].quantile(0.05)
    highP = lastRating['chg_exp'].quantile(0.95)
    lastRating = lastRating[(lastRating['chg_exp']>lowerP) & (la
stRating['chg_exp']<highP)]
    lastRating['chg_exp'] = (lastRating.chg_exp  - lastRating.ch
g_exp.mean())/lastRating.chg_exp.std()
    expRating = lastRating[['secShortName', 'publishDate', 'Rati
ng', 'previousRating', 'chg_exp']]

    # got expEps change
    lastEps = res2.groupby('secID').last()
    firstEps = res2.groupby('secID').first()
    lastEps['previousEps'] = firstEps.EPS_con
    lastEps['chg_eps'] = lastEps.EPS_con / firstEps.EPS_con - 1.0

    lowerP = lastEps['chg_eps'].quantile(0.05)
    highP = lastEps['chg_eps'].quantile(0.95)
    lastEps = lastEps[(lastEps['chg_eps']>lowerP) & (lastEps['ch
g_eps']<highP)]
    lastEps['chg_eps'] = (lastEps.chg_eps  - lastEps.chg_eps.mea
n())/lastEps.chg_eps.std()
    expEps = lastEps[['secShortName', 'publishDate', 'EPS_con',
'previousEps', 'chg_eps']]

    # Weighted Average Ranking
    rankRes = expEps.copy()
    rankRes['chg_exp'] = expRating.chg_exp
    rankRes['ranking'] = expEps.chg_eps + expRating.chg_exp

    # Current period return
    mktDate = MktEqudwithBatch(universe, 50, startDate, endDate)
    group = mktDate.groupby('secID')
    returnRes = group.last().closePrice / group.first().closePri
ce - 1.0
    rankRes['currentReturn'] = (returnRes - returnRes.mean()) /
returnRes.std()
    rankRes.dropna(inplace=True)
    # Do linear regression for current return
    x = rankRes[['chg_eps','chg_exp']].values
    y = rankRes.currentReturn.values
    x = sm.add_constant(x)
    model = sm.OLS(y, x)
    results = model.fit()


    rankRes['resid'] = results.resid

    return rankRes

def initialize(account):                      # 初始化虚拟账户状态
    account.traded = False
    account.universe = universe
    account.tradingMonth = set([1,4,7,10])
    account.currentTradedMonth = 0
    account.previousRatingExp = None
    account.previousEpsExp = None
    account.holdings = set()
    account.first = True
    account.chosen = 0.05

def handle_data(account):                     # 每个交易日的买入卖出指令

    today = Date(account.current_date.year, account.current_date
.month, account.current_date.day)
    if today.month() in account.tradingMonth and not account.tra
ded:
        hist = account.get_history(1)
        account.traded = True
        account.currentTradedMonth = today.month()
        endDate = today
        startDate = endDate - '3m'

        endStr = ''.join(endDate.toISO().split('-'))
        startStr = ''.join(startDate.toISO().split('-'))
        res = regressionTesting(account.universe, startStr, endS
tr)
        chosenNumber = int(account.chosen * len(res))

        secids = res.sort('resid')[:chosenNumber].index.values
        print today.toISO() + ' ' + str(chosenNumber) + u' 股票被
选择：' + str(secids)

        # clean current position
        c = account.cash
        for s in account.holdings:
            c += hist[s]['closePrice'][-1] * account.secpos.get(
s, 0)
            order_to(s, 0)

        equalAmount = c / chosenNumber
        # order equal amount
        for s in secids:
            approximationAmount = int(equalAmount / hist[s]['clo
sePrice'][-1])
            order(s, approximationAmount)

        account.holdings = secids


    if today.month() != account.currentTradedMonth:

        account.traded = False
```

!{}(img/20160730104832.jpg)

```
2011-01-05 8 股票被选择：['002252.XSHE' '000338.XSHE' '600031.XSHG'
 '600741.XSHG' '002024.XSHE'
 '000869.XSHE' '600027.XSHG' '600588.XSHG']
2011-04-01 9 股票被选择：['600406.XSHG' '300024.XSHE' '002081.XSHE'
 '000776.XSHE' '002310.XSHE'
 '002375.XSHE' '601933.XSHG' '600570.XSHG' '002065.XSHE']
2011-07-01 9 股票被选择：['600873.XSHG' '600415.XSHG' '002344.XSHE'
 '002400.XSHE' '300133.XSHE'
 '002415.XSHE' '601166.XSHG' '002422.XSHE' '600887.XSHG']
2011-10-10 8 股票被选择：['600085.XSHG' '000598.XSHE' '002594.XSHE'
 '000157.XSHE' '600999.XSHG'
 '600208.XSHG' '600252.XSHG' '600585.XSHG']
2012-01-04 9 股票被选择：['600516.XSHG' '601901.XSHG' '600348.XSHG'
 '600395.XSHG' '601928.XSHG'
 '600352.XSHG' '600827.XSHG' '000629.XSHE' '600547.XSHG']
2012-04-05 9 股票被选择：['601929.XSHG' '300146.XSHE' '002450.XSHE'
 '300133.XSHE' '002603.XSHE'
 '600050.XSHG' '600252.XSHG' '601800.XSHG' '600267.XSHG']
2012-07-02 9 股票被选择：['002230.XSHE' '600143.XSHG' '002310.XSHE'
 '000729.XSHE' '600157.XSHG'
 '601258.XSHG' '600170.XSHG' '300133.XSHE' '002385.XSHE']
2012-10-08 9 股票被选择：['000869.XSHE' '002146.XSHE' '000338.XSHE'
 '601169.XSHG' '601336.XSHG'
 '000729.XSHE' '600031.XSHG' '002594.XSHE' '600115.XSHG']
2013-01-04 9 股票被选择：['002007.XSHE' '002065.XSHE' '601928.XSHG'
 '000858.XSHE' '600633.XSHG'
 '600519.XSHG' '600406.XSHG' '002603.XSHE' '603000.XSHG']
2013-04-01 9 股票被选择：['600809.XSHG' '000568.XSHE' '000060.XSHE'
 '000069.XSHE' '600549.XSHG'
 '000858.XSHE' '601377.XSHG' '002653.XSHE' '000338.XSHE']
2013-07-01 9 股票被选择：['600157.XSHG' '002475.XSHE' '000001.XSHE'
 '600886.XSHG' '002344.XSHE'
 '600028.XSHG' '600535.XSHG' '002429.XSHE' '600188.XSHG']
2013-10-08 9 股票被选择：['600372.XSHG' '600010.XSHG' '002146.XSHE'
 '002051.XSHE' '000999.XSHE'
 '600519.XSHG' '600518.XSHG' '000024.XSHE' '601117.XSHG']
2014-01-02 8 股票被选择：['300251.XSHE' '600880.XSHG' '600633.XSHG'
 '601928.XSHG' '002416.XSHE'
 '600637.XSHG' '600332.XSHG' '300058.XSHE']
2014-04-01 8 股票被选择：['002344.XSHE' '600880.XSHG' '002385.XSHE'
 '002310.XSHE' '600597.XSHG'
 '600315.XSHG' '600188.XSHG' '002415.XSHE']
2014-07-01 8 股票被选择：['300146.XSHE' '000413.XSHE' '002065.XSHE'
 '002456.XSHE' '300058.XSHE'
 '600633.XSHG' '000024.XSHE' '000400.XSHE']
2014-10-08 7 股票被选择：['600887.XSHG' '600863.XSHG' '300017.XSHE'
 '002292.XSHE' '002594.XSHE'
 '601169.XSHG' '000400.XSHE']
2015-01-05 8 股票被选择：['600880.XSHG' '002653.XSHE' '300017.XSHE'
 '603000.XSHG' '002456.XSHE'
 '002292.XSHE' '000963.XSHE' '300133.XSHE']
```

3.3 牛熊转换

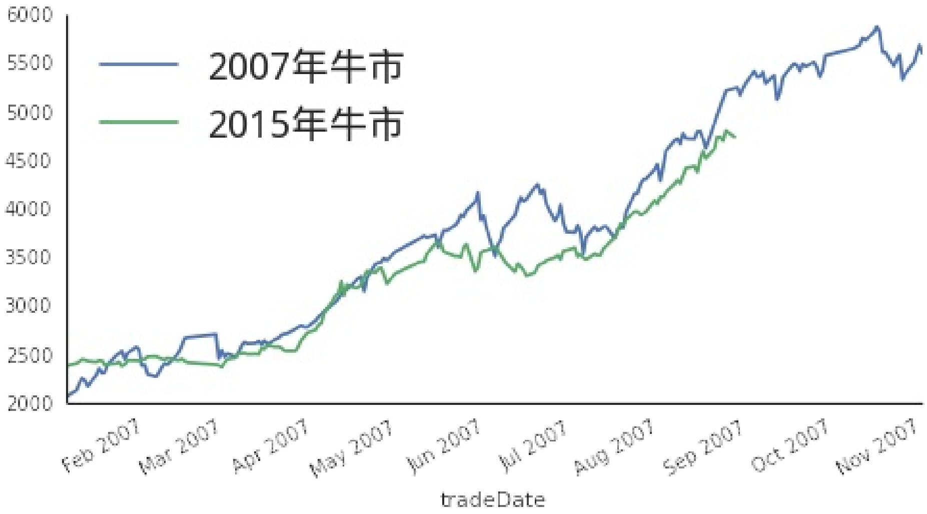
历史总是相似 牛市还在延续

> 来源：https://uqer.io/community/share/5541f07af9f06c1c3d687ffa

07年与15年的牛市时如此相似，你准备好继续all in 了吗？

话不多说，图见真章!

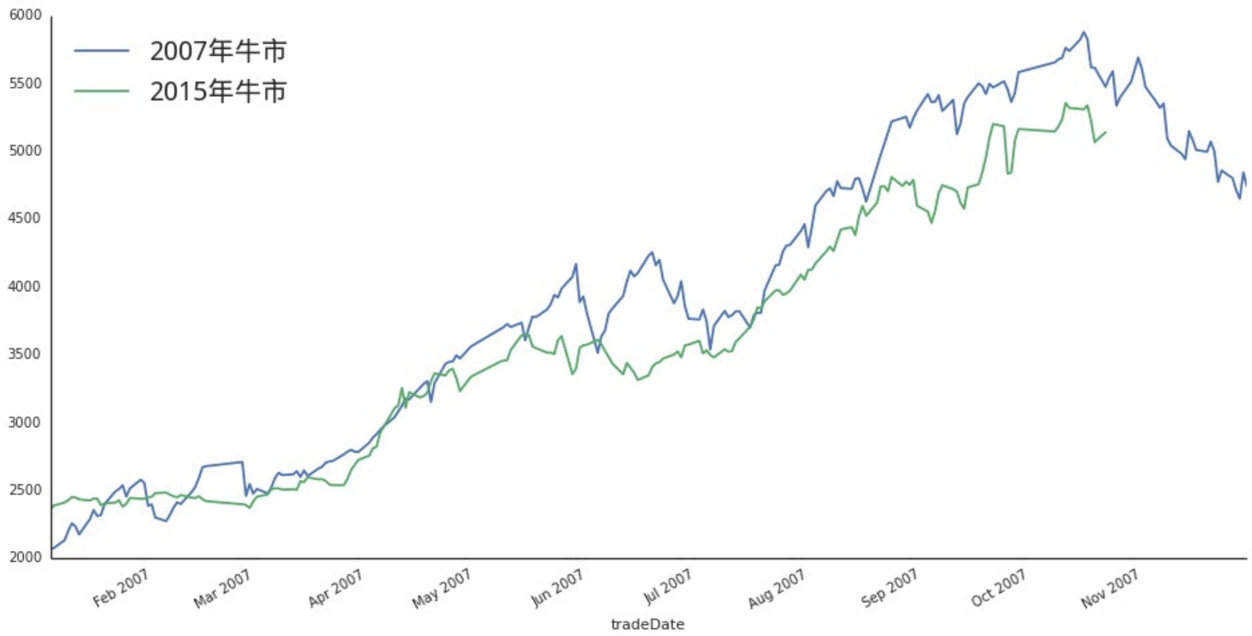
```
import datetime as dt
import numpy as np
import seaborn as sns
sns.set_style('white')
from matplotlib import pyplot as plt
from CAL.PyCAL import *
font.set_size(20)

index = '000300'
data = DataAPI.MktIdxdGet(ticker = index, beginDate='20070101',
endDate='20071101')
data.index = data.tradeDate.apply(lambda x: dt.datetime.strptime
(x, '%Y-%m-%d'))

data2 = DataAPI.MktIdxdGet(ticker =index, beginDate='20140830',
endDate='20150428')
data2.index = data2.tradeDate.apply(lambda x: dt.datetime.strpti
me(x, '%Y-%m-%d'))

data['2006 - 2008'] = data['closeIndex']
data = data[['2006 - 2008']]
data['2014 - 2015'] = np.nan
data['2014 - 2015'][:len(data2.closeIndex.values)] = data2.close
Index.values
data = data[['2006 - 2008', '2014 - 2015']]
data.plot(figsize=(8,4), grid = False)
plt.legend([u'2007年牛市', u'2015年牛市'], prop = font, loc = 'bes
t')
sns.despine()
```


6000
5500
5000
4500
4000
3500
3000
2500
2000
2007年牛市
2015年牛市
Feb 2007
Mar 2007
Apr 2007
May 2007
Jun 2007
Jul 2007
Aug 2007
Sep 2007
Oct 2007
Nov 2007
tradeDate

历史总是相似 牛市已经见顶？

> 来源：https://uqer.io/community/share/55814e7ef9f06c6519ad1522

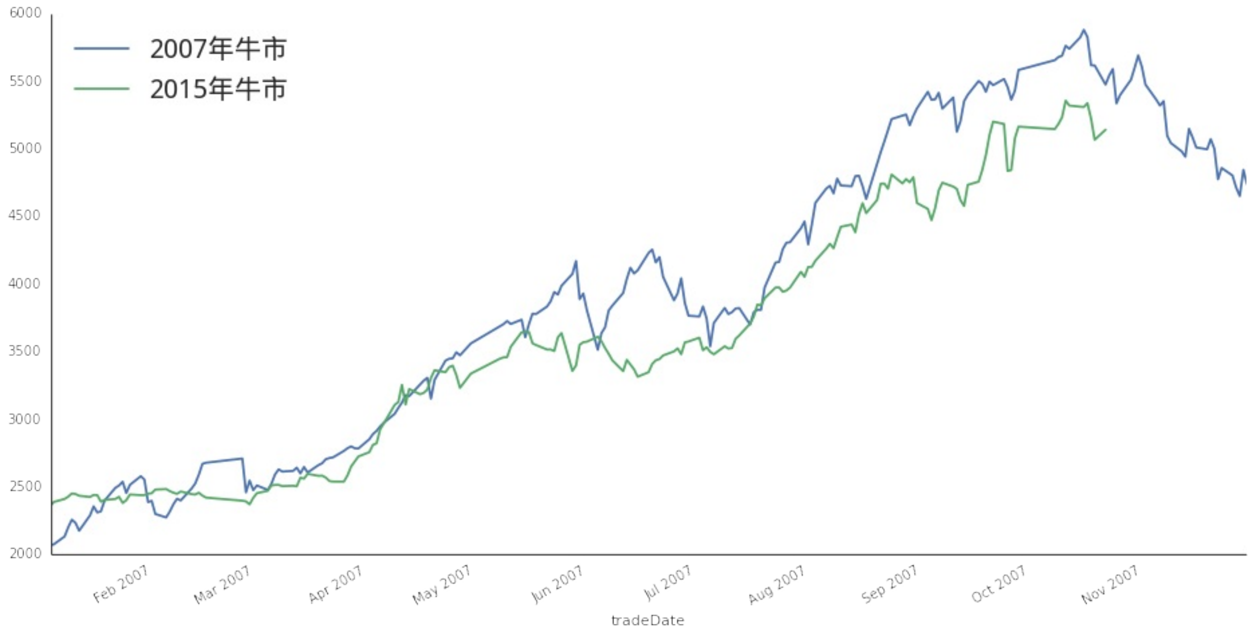
本文是前文《历史总是相似 牛市还在延续》的续篇， 此文可点击下面的链接：历史总是相似 牛市还在延续



话不多说，图见眞章!

这次我们把比较的周期从2015年4月28日延续到2015年6月17日。现在两个月过去了，看到大盘的走势和07年那波牛市是相似的。按照相同的时间点，07年的大牛已经见顶，15年呢？

```
import datetime as dt
import numpy as np
import seaborn as sns
sns.set_style('white')
from matplotlib import pylab
from CAL.PyCAL import *
font.set_size(20)

index = '000300'
data = DataAPI.MktIdxdGet(ticker = index, beginDate='20070101', endDate='20071201')
data.index = data.tradeDate.apply(lambda x: dt.datetime.strptime(x, '%Y-%m-%d'))

data2 = DataAPI.MktIdxdGet(ticker =index, beginDate='20140830', endDate='20150617')
data2.index = data2.tradeDate.apply(lambda x: dt.datetime.strptime(x, '%Y-%m-%d'))

data['2006 - 2008'] = data['closeIndex']
data = data[['2006 - 2008']]
data['2014 - 2015'] = np.nan
data['2014 - 2015'][:len(data2.closeIndex.values)] = data2.closeIndex.values
data = data[['2006 - 2008', '2014 - 2015']]
data.plot(figsize=(16,8), grid = False)
pylab.legend([u'2007年牛市', u'2015年牛市'], prop = font, loc = 'best')
sns.despine()
```



3.4 熔断机制 • 股海拾贝之 [熔断错杀股]

> 来源：https://uqer.io/community/share/568e4c78228e5b18e0ba2962

新年伊始，本是普天同庆之时，A股门前却风声鹤唳；熔断机制推出4天，触发-7%熔断两次；千古跌停中，有不少标的遭遇恐慌性抛售，有人称之为熔断机制的磁石效应！

本文中，我们试图以简单的逻辑，来找到今天即7日有可能遭遇恐慌性抛售的熔断错杀股。

```
import pandas as pd
import numpy as np
from matplotlib import pylab
import matplotlib.pyplot as plt
import seaborn
```

1. 今日错杀股

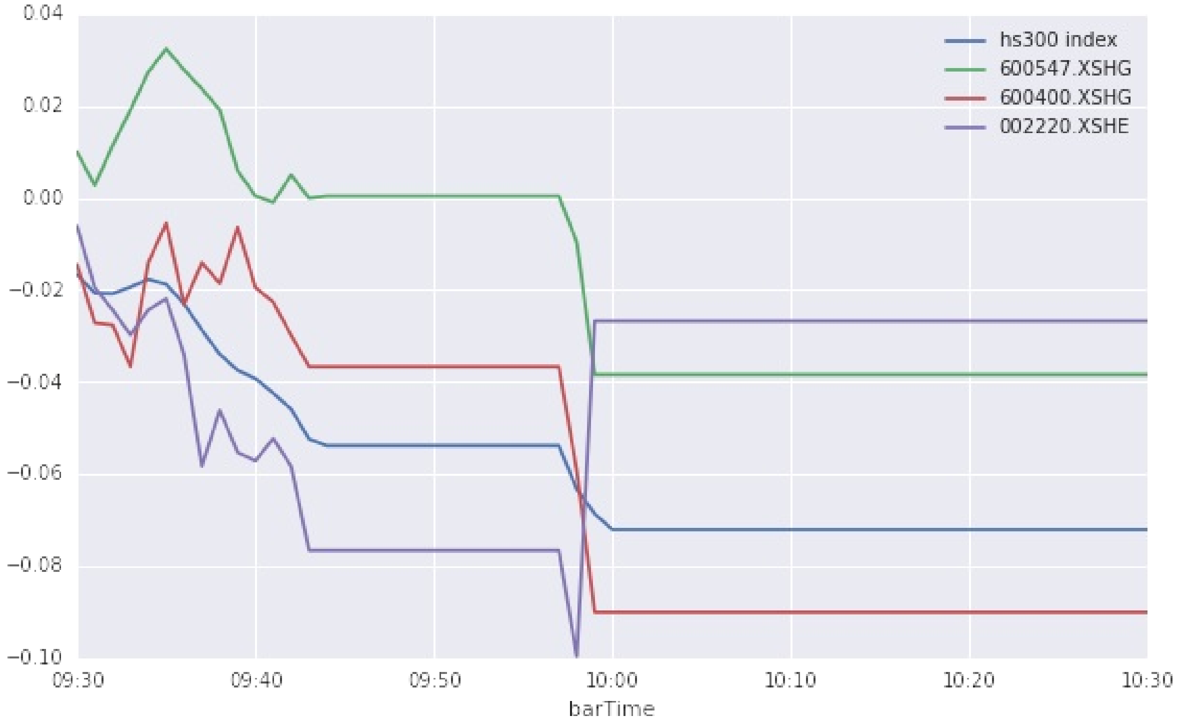
```
from quartz.api import set_universe

univ = set_universe('A')
univ.remove('002778.XSHE')  # 删除2016-01-06上市新股
```

以最浅显地理解，错杀股票可能存在于以下情况中：

- 今日开盘涨势不错，到首次熔断时仍为红盘，首次熔断结束后杀跌；
- 今日开盘处于跌势，首次熔断之前跌的少，而首次熔断到二次熔断之间杀跌；
- 首次熔断结束后开始交易，先杀跌，然后有明显拉升迹象，使得二次熔断时价格高于首次熔断时价格

三种情况下的走势如下图所示：

```
# 沪深300指数今日走势
hs300PreClose = DataAPI.MktIdxdGet(tradeDate='20160106',ticker='399300').closeIndex.values[0]
hs300 = DataAPI.MktBarRTIntraDayGet(securityID='399300.XSHE',startTime='09:30',endTime='10:30')
hs300['hs300 index'] = hs300.closePrice/hs300PreClose - 1
hs300 = hs300[['barTime','hs300 index']]

# 第一种情况，例如'山东黄金'
case1PrePrice = DataAPI.MktEqudAdjGet(tradeDate='20160106',ticker='600547').closePrice.values[0]
case1 = DataAPI.MktBarRTIntraDayGet(securityID='600547.XSHG',startTime='09:30',endTime='10:30')
case1['600547.XSHG'] = case1.closePrice/case1PrePrice - 1
case1 = case1[['barTime','600547.XSHG']]

# 第二种情况，例如'红豆股份'
case2PrePrice = DataAPI.MktEqudAdjGet(tradeDate='20160106',ticker='600400').closePrice.values[0]
case2 = DataAPI.MktBarRTIntraDayGet(securityID='600400.XSHG',startTime='09:30',endTime='10:30')
case2['600400.XSHG'] = case2.closePrice/case2PrePrice - 1
case2 = case2[['barTime','600400.XSHG']]

# 第三种情况，例如'天宝股份'
case3PrePrice = DataAPI.MktEqudAdjGet(tradeDate='20160106',ticker='002220').closePrice.values[0]
case3 = DataAPI.MktBarRTIntraDayGet(securityID='002220.XSHE',startTime='09:30',endTime='10:30')
case3['002220.XSHE'] = case3.closePrice/case3PrePrice - 1
case3 = case3[['barTime','002220.XSHE']]

for case in (case1,case2,case3):
    hs300 = pd.merge(hs300,case)

hs300 = hs300.set_index('barTime')
hs300.plot(figsize=(10,6))

<matplotlib.axes.AxesSubplot at 0x6e01210>
```



根据以上分析，我们拿取今日行情数据，对全A股中的股票做分析，得到表格如下，其中：

- `preClose` ： 昨日收盘价
- `1stPrice` ： 首次熔断时价格
- `2ndPrice` ： 二次熔断时价格
- `1stCollapse` ： 首次熔断时股价跌幅
- `2ndCollapse` ： 首次熔断结束，之后股价继续下跌的幅度，即首次熔断后今日股价又下跌了这么多
- `collapseRatio` ：首次熔断之后的跌幅和首次熔断之前的跌幅的比例
- `lowBetCollapse` ：首次熔断和二次熔断之间的股价低点
- `closeToLow` ： 二次熔断时的价格和lowBetCollapse的比例

```
cols = ['preClose','1stPrice','2ndPrice','1stCollapse','2ndCollapse','collapseRatio','lowBetCollapse','closeToLow']
collapse0107 = pd.DataFrame(0.0,index=univ,columns=cols)

# 股票前收盘价
collapse0107['preClose'] = DataAPI.MktEqudGet(tradeDate='20160106',secID=univ,field='secID,closePrice',pandas='1').set_index('secID')

# 股票在-5%和-7%熔断时候的价格
for stk in collapse0107.index:
    price = DataAPI.MktBarRTIntraDayGet(securityID=stk,startTime='09:56',endTime='10:03').closePrice.values
    collapse0107['1stPrice'][stk] = price[0]
    collapse0107['2ndPrice'][stk] = price[-1]
    collapse0107['lowBetCollapse'][stk] = np.min(price)

# 两次熔断前的股票跌幅
collapse0107['1stCollapse'] = 1-collapse0107['1stPrice']/collapse0107['preClose']
collapse0107['2ndCollapse'] = 1-collapse0107['2ndPrice']/collapse0107['preClose'] - collapse0107['1stCollapse']

# 二次熔断跌幅和一次熔断跌幅比
collapse0107['collapseRatio'] = collapse0107['2ndCollapse']/collapse0107['1stCollapse']

# 第二次熔断时价格与两次熔断之间的最低价的比值
collapse0107['closeToLow'] = collapse0107['2ndPrice']/collapse0107['lowBetCollapse']

collapse0107.sort(columns='collapseRatio',inplace=True)
collapse0107 = collapse0107[~np.isnan(collapse0107.collapseRatio)]

collapse0107.tail()
```

	preClose	1stPrice	2ndPrice	1stCollapse	2ndColl
603398.XSHG	123.01	121.00	110.71	0.016340	0.08365
002750.XSHE	40.59	40.05	37.00	0.013304	0.07514
600569.XSHG	3.58	3.55	3.31	0.008380	0.06703
002768.XSHE	71.17	71.00	66.00	0.002389	0.07025
600782.XSHG	5.81	5.80	5.31	0.001721	0.08433

按照以上数据，综合前面的简单逻辑，我们利用以下条件来选择错杀股：

- `collapseRatio` > 1.5
- `collapseRatio` < -0.9
- `closeToLow` > 1.02

得到约40只股票如下

```
good = collapse0107[(collapse0107.collapseRatio>1.5) | (collapse
0107.collapseRatio<-0.9) | (collapse0107.closeToLow>1.02)].index
good_stks = DataAPI.MktEqudGet(secID=good,tradeDate='20160107',f
ield='secID,secShortName,tradeDate,preClosePrice,closePrice')
good_stks
```

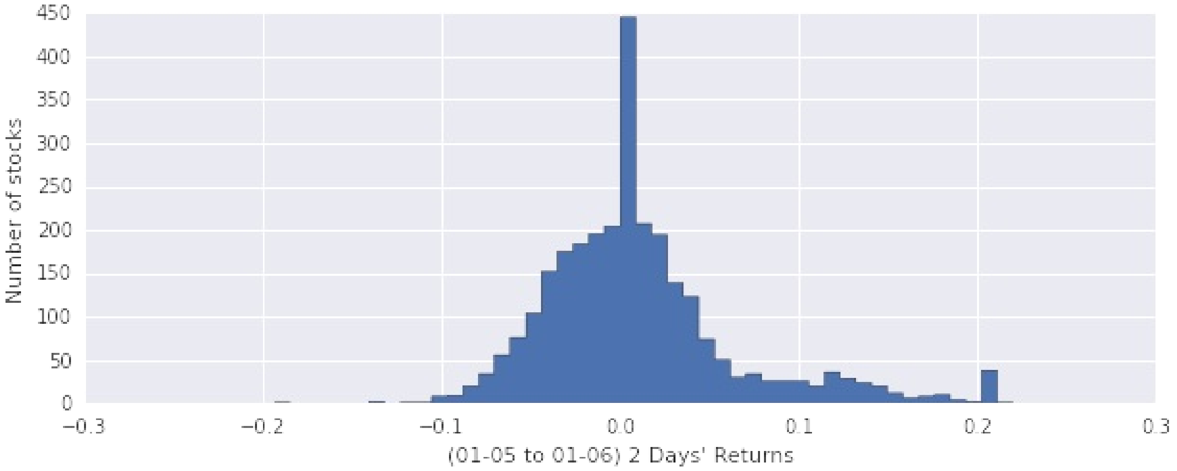
	secID	secShortName	tradeDate	preClosePrice	close
0	000726.XSHE	鲁泰A	2016-01-07	13.67	13.05
1	000766.XSHE	通化金马	2016-01-07	14.04	12.68
2	000838.XSHE	财信发展	2016-01-07	53.34	49.92
3	000856.XSHE	冀东装备	2016-01-07	12.91	12.00
4	000937.XSHE	冀中能源	2016-01-07	5.37	5.14
5	002155.XSHE	湖南黄金	2016-01-07	9.39	9.14
6	002220.XSHE	天宝股份	2016-01-07	16.46	15.15
7	002251.XSHE	步步高	2016-01-07	14.62	13.62
8	002283.XSHE	天润曲轴	2016-01-07	19.47	18.35
9	002291.XSHE	星期六	2016-01-07	14.24	13.03
10	002355.XSHE	兴民钢圈	2016-01-07	18.83	16.95
11	002444.XSHE	巨星科技	2016-01-07	19.44	17.85
12	002506.XSHE	协鑫集成	2016-01-07	9.85	9.66

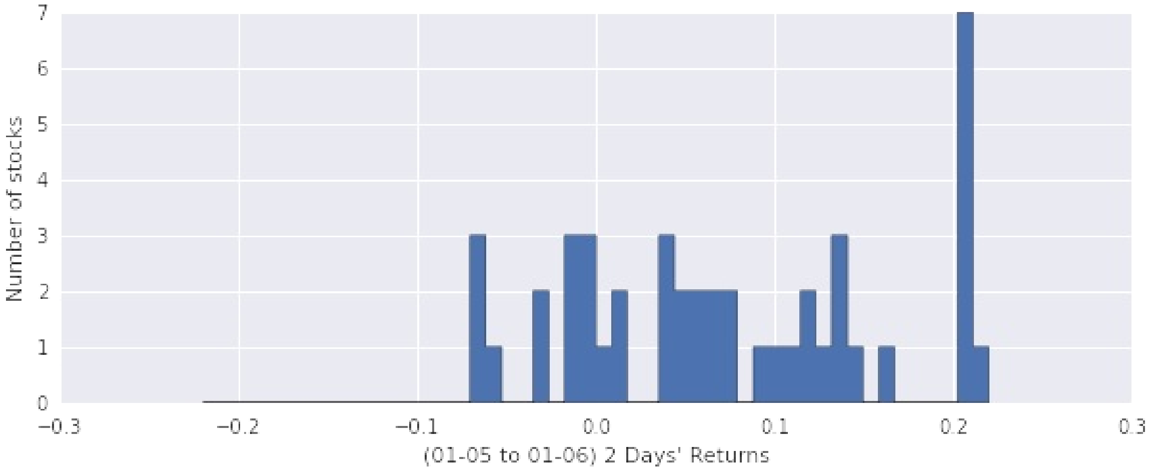
13	002517.XSHE	泰亚股份	2016-01-07	57.98	52.93
14	002575.XSHE	群兴玩具	2016-01-07	17.67	16.87
15	002588.XSHE	史丹利	2016-01-07	31.15	28.13
16	002615.XSHE	哈尔斯	2016-01-07	29.16	26.64
17	002617.XSHE	露笑科技	2016-01-07	23.17	21.12
18	002621.XSHE	三垒股份	2016-01-07	20.91	20.00
19	002640.XSHE	跨境通	2016-01-07	31.68	31.42
20	002702.XSHE	海欣食品	2016-01-07	24.59	23.05
21	002750.XSHE	龙津药业	2016-01-07	40.59	37.96
22	002768.XSHE	国恩股份	2016-01-07	71.17	67.03
23	002779.XSHE	中坚科技	2016-01-07	91.20	82.62
24	300013.XSHE	新宁物流	2016-01-07	17.50	16.38
25	300148.XSHE	天舟文化	2016-01-07	23.43	21.48
26	300179.XSHE	四方达	2016-01-07	9.79	9.18
27	300320.XSHE	海达股份	2016-01-07	12.88	11.95
28	600005.XSHG	武钢股份	2016-01-07	3.66	3.30
29	600057.XSHG	象屿股份	2016-01-07	12.01	11.25
30	600262.XSHG	北方股份	2016-01-07	35.69	33.50

31	600265.XSHG	ST景谷	2016-01-07	29.81	28.36
32	600291.XSHG	西水股份	2016-01-07	26.11	24.16
33	600298.XSHG	安琪酵母	2016-01-07	30.30	29.47
34	600328.XSHG	兰太实业	2016-01-07	13.63	12.54
35	600395.XSHG	盘江股份	2016-01-07	9.10	8.19
36	600448.XSHG	华纺股份	2016-01-07	9.01	8.60
37	600547.XSHG	山东黄金	2016-01-07	21.89	21.10
38	600569.XSHG	安阳钢铁	2016-01-07	3.58	3.38
39	600671.XSHG	天目药业	2016-01-07	32.98	32.82
40	600732.XSHG	*ST新梅	2016-01-07	8.51	8.46
41	600782.XSHG	新钢股份	2016-01-07	5.81	5.54
42	600874.XSHG	创业环保	2016-01-07	9.63	9.67
43	601001.XSHG	大同煤业	2016-01-07	5.78	5.75
44	603398.XSHG	邦宝益智	2016-01-07	123.01	111.3

我们还想看一下上述股票在过去的05、06两个交易日的表现：

```
fig = plt.figure(figsize=(10,8))

ax = fig.add_subplot(211)
fullA = DataAPI.MktEqudAdjGet(secID=univ, beginDate='20160104',
endDate='20160106', field='secID,tradeDate,closePrice', pandas='
1')
fullA = pd.DataFrame(fullA.groupby('secID').last().closePrice/fu
llA.groupby('secID').first().closePrice - 1)
ax = pylab.hist(fullA.closePrice,bins=50,histtype='stepfilled',r
ange=(-0.22,0.22))
pylab.xlabel("(01-05 to 01-06) 2 Days' Returns")
pylab.ylabel('Number of stocks')

ax = fig.add_subplot(212)
good_stk_data = DataAPI.MktEqudAdjGet(secID=good, beginDate='201
60104', endDate='20160106', field='secID,tradeDate,closePrice',
pandas='1')
good_stk_data = pd.DataFrame(good_stk_data.groupby('secID').last
().closePrice/good_stk_data.groupby('secID').first().closePrice
- 1)
ax = pylab.hist(good_stk_data.closePrice,bins=50,histtype='stepf
illed',range=(-0.22,0.22))
pylab.xlabel("(01-05 to 01-06) 2 Days' Returns")
pylab.ylabel('Number of stocks')

<matplotlib.text.Text at 0x7335050>
```





图中，上图为全A股在过去的5、6日两个交易日收益表现分布；下图为我们选出来的今日错杀股在5、6两个交易日的收益表现分布；

明显地，我们看出选出来的股票在过去两天表现比较出色；当然，过去两天的表现好不代表它们今天被错杀

2. 上次熔断时的4日被错杀股

利用上节中的选股条件，我们选出来4日熔断错杀股，来验证我们的逻辑

```
cols = ['preClose','1stPrice','2ndPrice','1stCollapse','2ndCollapse','collapseRatio','lowBetCollapse','closeToLow']
collapse0104 = pd.DataFrame(0.0,index=univ,columns=cols)

# 股票前收盘价
collapse0104['preClose'] = DataAPI.MktEqudGet(tradeDate='20151231',secID=univ,field='secID,closePrice',pandas='1').set_index('secID')

# 股票在-5%和-7%熔断时候的价格
for stk in collapse0104.index:
    price = DataAPI.MktBarHistOneDayGet(securityID=stk,date='20160104',startTime='13:25',endTime='13:40').closePrice.values
    collapse0104['1stPrice'][stk] = price[0]
    collapse0104['2ndPrice'][stk] = price[-1]
    collapse0104['lowBetCollapse'][stk] = np.min(price)

# 两次熔断前的股票跌幅
collapse0104['1stCollapse'] = 1-collapse0104['1stPrice']/collapse0104['preClose']
collapse0104['2ndCollapse'] = 1-collapse0104['2ndPrice']/collapse0104['preClose'] - collapse0104['1stCollapse']

collapse0104['closeToLow'] = collapse0104['2ndPrice']/collapse0104['lowBetCollapse']

# 二次熔断跌幅和一次熔断跌幅比
collapse0104['collapseRatio'] = collapse0104['2ndCollapse']/collapse0104['1stCollapse']

collapse0104.sort(columns='collapseRatio',inplace=True)
collapse0104 = collapse0104[~np.isnan(collapse0104.collapseRatio)]

collapse0104.tail()
```

	preClose	1stPrice	2ndPrice	1stCollapse	2ndColl
600836.XSHG	31.31	31.28	30.37	0.000958	0.02906
600178.XSHG	11.23	11.23	10.60	0.000000	0.05610
600822.XSHG	15.01	15.01	14.28	0.000000	0.04863
002686.XSHE	17.62	17.62	16.90	0.000000	0.04086
002009.XSHE	20.75	20.75	19.71	0.000000	0.05012

按照之前的选股条件，我们选出来了4日熔断被错杀的股票如下：

```
good = collapse0104[(collapse0104.collapseRatio>1.5) | (collapse
0104.collapseRatio<-0.9) | (collapse0104.closeToLow>1.02)].index
good_stks = DataAPI.MktEqudGet(secID=good,tradeDate='20160107',f
ield='secID,secShortName,tradeDate,preClosePrice,closePrice')
good_stks
```

	secID	secShortName	tradeDate	preClosePrice	clos
0	000040.XSHE	宝安地产	2016-01-07	15.45	13.9
1	000048.XSHE	康达尔	2016-01-07	44.68	40.2
2	000517.XSHE	荣安地产	2016-01-07	6.20	5.58
3	000519.XSHE	江南红箭	2016-01-07	19.20	17.2
4	000520.XSHE	长航凤凰	2016-01-07	13.11	11.8
5	000547.XSHE	航天发展	2016-01-07	18.81	16.9
6	000552.XSHE	靖远煤电	2016-01-07	9.85	8.87
7	000597.XSHE	东北制药	2016-01-07	11.50	10.3
8	000667.XSHE	美好集团	2016-01-07	5.62	5.06
9	000708.XSHE	大冶特钢	2016-01-07	13.20	11.8
10	000709.XSHE	河北钢铁	2016-01-07	3.67	3.30
11	000723.XSHE	美锦能源	2016-01-07	14.19	12.7
12	000757.XSHE	浩物股份	2016-01-07	10.80	9.72
13	000767.XSHE	漳泽电力	2016-01-07	6.29	5.66
14	000795.XSHE	太原刚玉	2016-01-07	17.12	15.4

15	000801.XSHE	四川九洲	2016-01-07	30.26	27.2
16	000898.XSHE	鞍钢股份	2016-01-07	5.19	4.68
17	000932.XSHE	华菱钢铁	2016-01-07	3.85	4.14
18	000952.XSHE	广济药业	2016-01-07	21.11	19.0
19	000990.XSHE	诚志股份	2016-01-07	24.59	22.1
20	002009.XSHE	天奇股份	2016-01-07	22.85	20.5
21	002013.XSHE	中航机电	2016-01-07	24.02	21.6
22	002025.XSHE	航天电器	2016-01-07	25.42	22.8
23	002045.XSHE	国光电器	2016-01-07	18.10	16.2
24	002149.XSHE	西部材料	2016-01-07	29.70	26.7
25	002157.XSHE	正邦科技	2016-01-07	21.13	19.3
26	002179.XSHE	中航光电	2016-01-07	37.56	34.1
27	002191.XSHE	劲嘉股份	2016-01-07	16.71	15.0
28	002200.XSHE	云投生态	2016-01-07	27.99	25.1
29	002220.XSHE	天宝股份	2016-01-07	16.46	15.1
...	...	...	...	...	...
104	600655.XSHG	豫园商城	2016-01-07	15.44	14.2
105	600662.XSHG	强生控股	2016-01-07	17.13	15.4
106	600663.XSHG	陆家嘴	2016-01-07	50.94	45.9

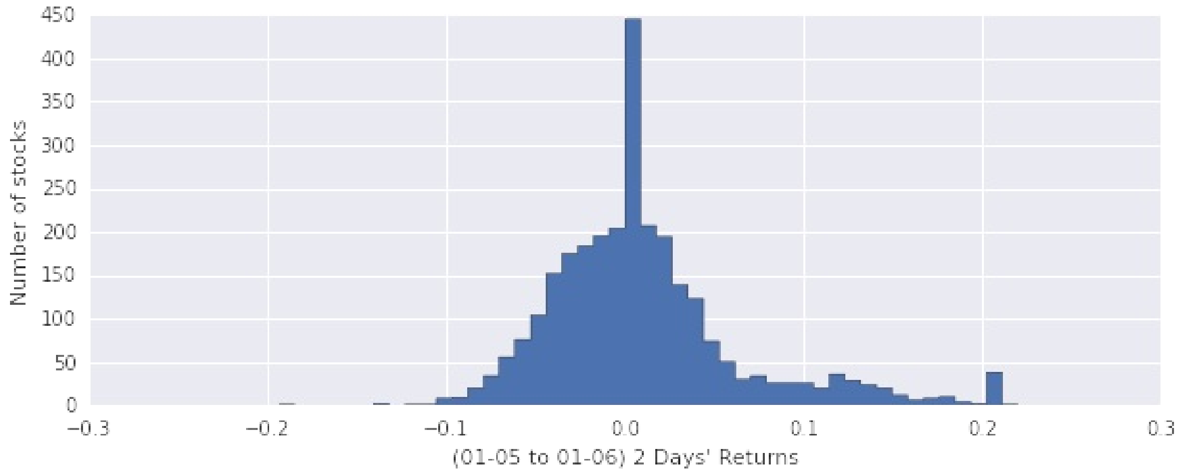
107	600685.XSHG	中船防务	2016-01-07	41.41	37.2
108	600734.XSHG	实达集团	2016-01-07	24.90	22.4
109	600755.XSHG	厦门国贸	2016-01-07	8.24	7.43
110	600760.XSHG	中航黑豹	2016-01-07	14.60	13.1
111	600774.XSHG	汉商集团	2016-01-07	29.00	26.1
112	600822.XSHG	上海物贸	2016-01-07	15.97	14.3
113	600826.XSHG	兰生股份	2016-01-07	36.16	32.5
114	600833.XSHG	第一医药	2016-01-07	18.55	16.7
115	600834.XSHG	申通地铁	2016-01-07	20.30	18.2
116	600836.XSHG	界龙实业	2016-01-07	34.32	30.8
117	600841.XSHG	上柴股份	2016-01-07	18.28	16.4
118	600855.XSHG	航天长峰	2016-01-07	45.30	40.7
119	600860.XSHG	京城股份	2016-01-07	11.38	10.2
120	600868.XSHG	梅雁吉祥	2016-01-07	7.61	6.89
121	600879.XSHG	航天电子	2016-01-07	18.94	17.0
122	600893.XSHG	中航动力	2016-01-07	42.87	38.5
123	600965.XSHG	福成五丰	2016-01-07	14.85	13.6
124	601069.XSHG	西部黄金	2016-01-07	23.71	21.9

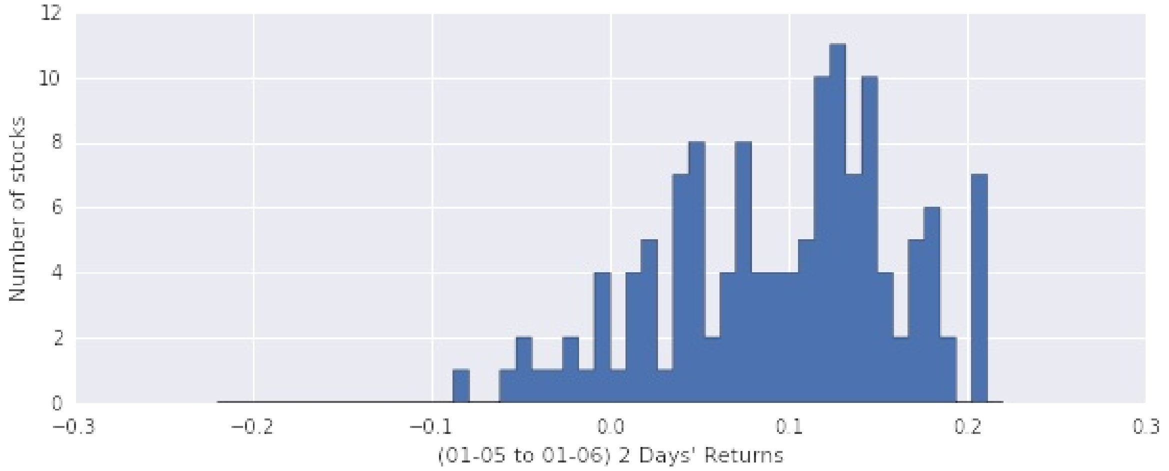
125	601233.XSHG	桐昆股份	2016-01-07	12.37	11.1
126	601636.XSHG	旗滨集团	2016-01-07	5.18	4.70
127	601700.XSHG	风范股份	2016-01-07	10.80	9.72
128	601888.XSHG	中国国旅	2016-01-07	56.49	51.9
129	601890.XSHG	亚星锚链	2016-01-07	13.43	12.0
130	601989.XSHG	中国重工	2016-01-07	9.40	8.48
131	601998.XSHG	中信银行	2016-01-07	6.84	6.39
132	603696.XSHG	安记食品	2016-01-07	64.11	57.7
133	603901.XSHG	永创智能	2016-01-07	40.84	36.7

```
134 rows × 5 columns
```

我们选出来的4日熔断被错杀的股票，在后面的5、6两日的表现究竟如何呢？请看下图

```
fig = plt.figure(figsize=(10,8))

ax = fig.add_subplot(211)
fullA = DataAPI.MktEqudAdjGet(secID=univ, beginDate='20160104',
endDate='20160106', field='secID,tradeDate,closePrice', pandas='
1')
fullA = pd.DataFrame(fullA.groupby('secID').last().closePrice/fu
llA.groupby('secID').first().closePrice - 1)
ax = pylab.hist(fullA.closePrice,bins=50,histtype='stepfilled',r
ange=(-0.22,0.22))
pylab.xlabel("(01-05 to 01-06) 2 Days' Returns")
pylab.ylabel('Number of stocks')

ax = fig.add_subplot(212)
good_stk_data = DataAPI.MktEqudAdjGet(secID=good, beginDate='201
60104', endDate='20160106', field='secID,tradeDate,closePrice',
pandas='1')
good_stk_data = pd.DataFrame(good_stk_data.groupby('secID').last
().closePrice/good_stk_data.groupby('secID').first().closePrice
- 1)
ax = pylab.hist(good_stk_data.closePrice,bins=50,histtype='stepf
illed',range=(-0.22,0.22))
pylab.xlabel("(01-05 to 01-06) 2 Days' Returns")
pylab.ylabel('Number of stocks')

<matplotlib.text.Text at 0x74df810>
```





图中，上图为全A股在过去的5、6日两个交易日收益表现分布；下图为我们选出来的4日熔断错杀股在5、6两个交易日的收益表现分布，可以看出选出的错杀股在过去两天均有10%左右的涨幅；

明显地，我们看出选出来的4日熔断错杀股在后面的两天表现出色

3. 结论

2节中对于4日熔断错杀股在5、6两个交易日的数据，似乎能够支持我们在第1节中的错杀股选股逻辑；对于1节中选出来的今天即7日错杀股，搬个板凳看后面走势究竟如何

PS：股市风险大，投资需谨慎；本文仅是研究之用，不构成任何荐股观点

3.5 暴涨暴跌 • [实盘感悟] 遇上暴跌我该怎么做？

> 来源：https://uqer.io/community/share/565d47e3f9f06c6c8a91af49

写贴缘由：

- 经历了上周五的大跌以及昨天的深V反转，这行情真是吓死宝宝了。。。
- 好在这次还算淡定，没有在低点出货，算是理智战胜了恐惧吧（具体我相信其不会继续暴跌的原因详见后面）
- 所以痛定思痛，好好反思总结，来说说暴跌我该怎么办吧

回首历史：

记得在7,8月份大跌时，我做过一个简单的统计，统计了上证综指历史上所有周度的涨跌幅，统计后发现，大部分的涨跌幅都位于（-5%，5%），一些极端情况会出现涨跌10%左右，跌的最多的好像也就15%左右，而且即使在熊市，跌多了也会短期反弹一点，用专业的话讲，应该算是下跌趋势中的短期反转吧（momentum&reverse）。

所以，当时面对当时的下跌，我并没有太绝望，我在等的也就是反转的机会，而且不承担选股风险，就买带杠杆的分级B，快进快出，这样熊市下来我并没有亏损太多，甚至略微有盈利。

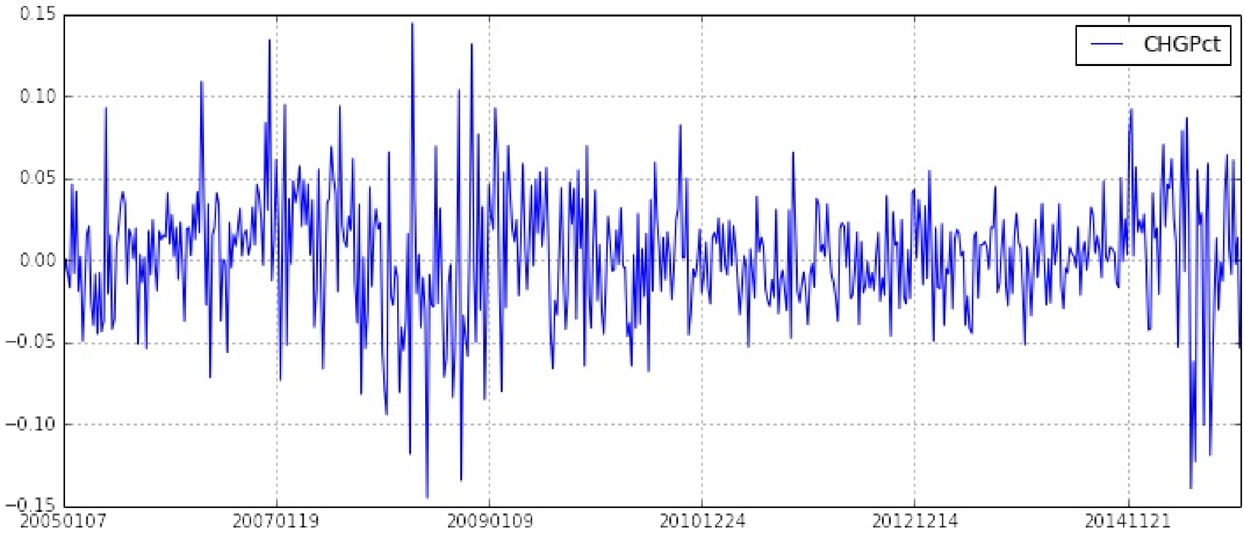
那么问题来了，怎么判定反转呢？不然这些都是大白话。我的想法也非常简单，既然历史（当时主要是07,08年的）统计表明跌幅超过15%之后再继续下跌的概率不高，那么一旦跌幅超过15%我在入场抄底，反转的机会是不是很高呢？说起来很简单吧，但做起来呢？

做起来就没那么简单了，人毕竟是感性的，遇上满仓跌停你的第一感觉就是将亏损和工资or奖金匹配了，这个时候都是恐惧，所以才出现了巨幅的跳水，因为大部分人都已经没法正常思考了（比如你当时想买某个分级B，你绝对不太会像之前一样理性，先要看看分级基金整体折溢价如何。。）

但这个时候就是真正的机会，别人恐惧时你要贪婪。想在想想当时7、8月份的行情，有很多曾经盘中一度跌幅超过15%的吧，这其实都是很好的抄底时机，但在熊市里，抄底要注意快进快出！

先来看看上证综指的历史周度收益吧

```
import lib.BGI as BGI
import pandas as pd
import numpy as np

quotes_daily = DataAPI.MktIdxdGet(ticker='000001', beginDate='20050101', field='tradeDate,CHGPct', pandas='1').set_index('tradeDate')
quotes_daily.index = map(lambda x: x.replace('-', ''), quotes_daily.index)
quotes_weekly = BGI.daily2weekly(quotes_daily, 'sum') # 周度行情
quotes_weekly.plot(figsize=(12,5))

<matplotlib.axes.AxesSubplot at 0x4d19f50>
```



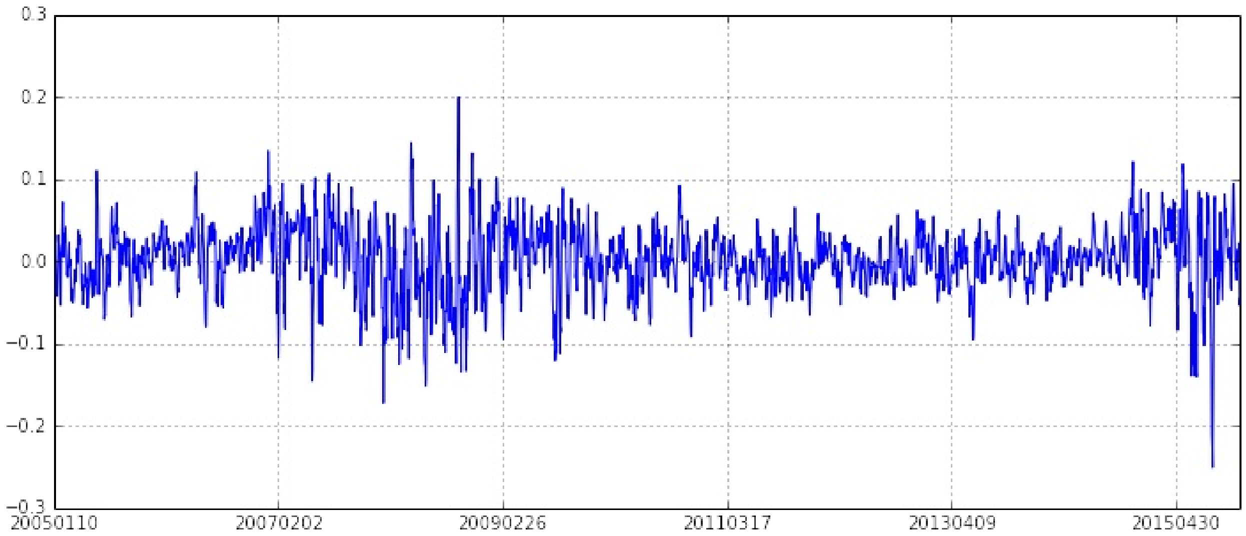
怎么样，大部分都是位于（-5%，5%）吧，只有在熊市里才有超过10%的，但都没有超过15%。所以，在某一周中，盘中跌幅超过15%，那抄底还是挺自信的，更何况，现在的行情已经有企稳迹象，并非像7、8月份的熊市，怕什么呢？看看上周四、周五、再加昨天盘中的最大跌幅，刚刚好10%，敢问各位，抄底了吗？

当然，我也好奇的统计了，大盘过去N天的跌幅情况，以及随后M天的反弹情况，统计了收益为正的概率等等（统计规律没太多投资逻辑，仅供参考）

```
# 输入原始daily行情，获取行情统计信息，统计过去before_days的收益，以及随后after_days的收益
def get_quotes(row_quotes, before_days, after_days):
    quotes_Ndays = pd.DataFrame(index=row_quotes.index[before_days-1:], columns=['before '+str(before_days)+'days return','after '+str(after_days)+'days return'], data=0.0)
    for i in range(before_days-1,row_quotes.shape[0]):
        quotes_Ndays.iloc[i-before_days+1,:] = [row_quotes.iloc[i-before_days+1:i+1]['CHGPct'].values.sum(),row_quotes.iloc[i+1:i+1+after_days]['CHGPct'].values.sum()]
    return quotes_Ndays
```

下图是上证综指过去五天的累计收益图（没有按照周来划分），可以看到，结果差不多，连续五天下跌幅度超过10%的非常少，只有在8月份的熊市里，才出现过一次20%！

```
quotes_Ndays = get_quotes(quotes_daily, 5, 1)
quotes_Ndays[quotes_Ndays.columns[0]].plot(figsize=(12,5))

<matplotlib.axes.AxesSubplot at 0x4d0c310>
```



接下来，就简单统计一下，上证综指在连续五天分别下跌10%、15%、20%，在随后的1天、2天、3天、4天、5天内行情的均值收益以及收益为正的概率。

```
res = pd.DataFrame(index=['drawdown 10%','drawdown 15%','drawdow
n 20%'], columns=['following 1 day','following 2 day','following
 3 day','following 4 day','following 5 day'])
for i in range(1,6):
    tmp_quote = get_quotes(quotes_daily, 5, i)
    tmp1 = tmp_quote[tmp_quote['before 5days return'] <= -0.1]['
after '+str(i)+'days return']
    tmp2 = tmp_quote[tmp_quote['before 5days return'] <= -0.15][
'after '+str(i)+'days return']
    tmp3 = tmp_quote[tmp_quote['before 5days return'] <= -0.2]['
after '+str(i)+'days return']
    res.loc['drawdown 10%',res.columns[i-1]] = (np.round(tmp1.me
an(), 4), np.round(float(sum(tmp1>0))/len(tmp1), 3))
    res.loc['drawdown 15%',res.columns[i-1]] = (np.round(tmp2.me
an(), 4), np.round(float(sum(tmp2>0))/len(tmp2), 3))
    res.loc['drawdown 20%',res.columns[i-1]] = (np.round(tmp3.me
an(), 4), np.round(float(sum(tmp3>0))/len(tmp3),3))
res
```

	following 1 day	following 2 day	following 3 day	following 4 day	following 5 day
drawdown 10%	(-0.001, 0.524)	(0.0067, 0.595)	(0.0078, 0.524)	(0.0081, 0.5)	(0.0077, 0.548)
drawdown 15%	(0.0076, 0.667)	(0.017, 0.667)	(0.0408, 0.833)	(0.0223, 0.667)	(0.0208, 0.667)
drawdown 20%	(-0.0119, 0.333)	(0.0178, 0.667)	(0.0489, 0.667)	(0.0582, 1.0)	(0.0507, 1.0)

如上表所示：

- 每个单元格是一个tuple，第一个值为收益率的均值，第二个值为收益率为正的概率，举例来说，第一个元素(-0.001, 0.524)表示上证综指过去5天下跌超过10%的情况下，在随后1天的行情中，所有收益率的均值为-0.001，所有收益率中收益为正的概率为0.524
- 可以看到，从收益均值角度看，基本上都能获得正的收益，而且多持有几天，获得均值正收益越多，上证综指3%左右还是很不错的收益
- 从概率的角度看，都能明显高于50%

当然，以上仅仅是一些历史统计规律，具有一定的参考性，个人可根据实际情况来进行抉择！

回到现在：

关于这一波下跌，市场也并没有给出一个明确的原因（或许是券商收益互换？美联储加息？？？。。。）

总之，现在还没有到7,8月份时候清理两融、清配资来得直接，此外现在点位相对来说不算太高，顶多算是给前期涨幅太多降降温；而且随着人民币“入篮”，今天的PMI数据也有企稳迹象，总总迹象都表明政府主导的结构性改革正在有序进行。

也许大盘还会继续跌（当然啦，我是不相信会跌到3000点的，而且个人觉得接下来的行情会不错~），因为北上的沪港通已经帮我建立一个牢靠的成本线（参见社区帖子https://uqer.io/community/share/564a9754f9f06c4446b48253，强烈建议去看一下），你不抄底，他们会抄底，不信的话请看这两天的沪股通流入额！

总结：

- 单单通过历史统计规律来决定是否抄底确实有点草率，但历史规律性的东西还是很具有参考价值的，好歹技术指标的一大前提就是相信历史会重演
- 实际运用中，还有结合当时的市场环境，是下跌趋势还是企稳震荡又或是上涨，当然还可以分析一下国内环境、国际环境、又或是一些其他指标（比如这里提到的沪港通）

- 总之，遇上暴跌不要恐慌，做一名理性的投资者！从本文的角度来看，暴跌不正是千载难逢的好机会吗？

3.6 兼并重组、举牌收购 • 宝万战-大戏开幕

> 来源：https://uqer.io/community/share/56767dd7228e5bab36c978eb

宝能系通过在二级市场上不断买入万科A，已经成为万科的大股东，上周四王石发表了《不欢迎宝能系成为第一大股东》，宝万大战已然开始。

周五下午万科停牌，加上宝万周末两天的准备，周一必将精彩。

野蛮人叩门，叩什么样的门，宝万大战已经告诉我们了。

作为普通投资者，在观战同时，切忌过于沉迷讨论事件里谁对谁错，因为有更重要的事情等着我们。下面我列出了第三季度公布的沪深300成分股的大股东不足30%的排名列表。

不管最终结果如何，这次事件一定是中国金融市场历史上浓重的一笔。

宝能系通过在二级市场上不断买入万科A，已经成为万科的大股东，上周四王石发表了《不欢迎宝能系成为第一大股东》，宝万大战已然开始。

周五下午万科停牌，加上宝万周末两天的准备，周一必将精彩。

野蛮人叩门，叩什么样的门，宝万大战已经告诉我们了。

作为普通投资者，在观战同时，切忌过于沉迷讨论事件里谁对谁错，因为有更重要的事情等着我们。下面我列出了第三季度公布的沪深300成分股的大股东不足30%的排名列表。

不管最终结果如何，这次事件一定是中国金融市场历史上浓重的一笔。

```
import pandas as pd

universe = set_universe('HS300')
data = pd.DataFrame()

for stock in universe:
    stock_df = DataAPI.CCXE.EquMainshCCXEGet(secID=stock,
                                              endDateStart=u"20150930",
                                              field=u"secShortName,endDate,s
hName,holdPct,shRank",
                                              pandas="1")
    data = data.append(stock_df)
```

```
print data[(data.shRank==1)&(data.holdPct<30)].sort(['holdPct'])
.reset_index(drop=True).to_html()
```

	secShortName	endDate	shName	holdPct

0	蓝色光标	2015-09-30 00:00:00	赵文权	7.0900
1	华闻传媒	2015-09-30 00:00:00	国广环球资产管理有限公司	7.1400
2	TCL集团	2015-09-30 00:00:00	惠州市投资控股有限公司	7.1900
3	鹏博士	2015-09-30 00:00:00	深圳鹏博实业集团有限公司	7.6300
4	伊利股份	2015-09-30 00:00:00	呼和浩特投资有限责任公司	8.7000
5	辽宁成大	2015-09-30 00:00:00	辽宁成大集团有限公司	11.1100
6	京东方A	2015-09-30 00:00:00	北京国有资本经营管理中心	11.5600
7	特变电工	2015-09-30 00:00:00	新疆特变电工集团有限公司	11.6200
8	中国宝安	2015-09-30 00:00:00	深圳市富安控股有限公司	11.9100
9	浙江龙盛	2015-09-30 00:00:00	阮水龙	11.9800
10	太平洋	2015-09-30 00:00:00	北京华信六合投资有限公司	12.7500
11	神州泰岳	2015-09-30 00:00:00	李力	12.8700
12	北京银行	2015-09-30 00:00:00	ING BANK N.V.	13.6400
		2015-		

13	科大讯飞	09-30 00:00:00	中国移动通信有限公司	13.9400
14	东旭光电	2015-09-30 00:00:00	东旭集团有限公司	14.6500
15	长江证券	2015-09-30 00:00:00	青岛海尔投资发展有限公司	14.7200
16	南京银行	2015-09-30 00:00:00	法国巴黎银行	14.8700
17	万科A	2015-09-30 00:00:00	华润股份有限公司	15.2300
18	康得新	2015-12-15 00:00:00	康得投资集团有限公司	15.2900
19	东软集团	2015-09-30 00:00:00	东北大学科技产业集团有限公司	15.6564
20	二三四五	2015-09-30 00:00:00	浙富控股集团股份有限公司	16.4600
21	电广传媒	2015-09-30 00:00:00	湖南广播电视产业中心	16.5800
22	康得新	2015-09-30 00:00:00	康得投资集团有限公司	17.1100
23	大族激光	2015-09-30 00:00:00	大族控股集团有限公司	17.8200
24	华兰生物	2015-09-30 00:00:00	安康	17.8400
25	兴业银行	2015-09-30 00:00:00	福建省财政厅	17.8600
26	招商银行	2015-09-30	香港中央结算(代理人)有限公司	18.0000

		00:00:00	司	
27	江苏有线	2015-09-30 00:00:00	江苏省广播电视信息网络投资有限公司	18.0300
28	国金证券	2015-09-30 00:00:00	长沙九芝堂(集团)有限公司	18.0900
29	金风科技	2015-09-30 00:00:00	香港中央结算(代理人)有限公司	18.2200
30	格力电器	2015-09-30 00:00:00	珠海格力集团有限公司	18.2200
31	中联重科	2015-09-30 00:00:00	香港中央结算(代理人)有限公司(HKSCC NOMINEES LIMITED)	18.5300
32	宁波银行	2015-09-30 00:00:00	新加坡华侨银行有限公司	18.5800
33	中信证券	2015-09-30 00:00:00	香港中央结算(代理人)有限公司	18.8000
34	民生银行	2015-09-30 00:00:00	香港中央结算(代理人)有限公司	18.9100
35	广联达	2015-09-30 00:00:00	刁志中	19.0200
36	汇川技术	2015-09-30 00:00:00	深圳市汇川投资有限公司	19.5100
37	永辉超市	2015-09-30 00:00:00	牛奶有限公司	19.9900
38	启迪桑德	2015-09-30 00:00:00	启迪科技服务有限公司	19.9900
39	浦发银行	2015-09-30	中国移动通信集团广东有限公司	20.0000

		00:00:00		
40	兴业证券	2015-09-30 00:00:00	福建省财政厅	20.0800
41	福耀玻璃	2015-09-30 00:00:00	HKSCC NOMINEES LIMITED	20.1500
42	威孚高科	2015-09-30 00:00:00	无锡产业发展集团有限公司	20.2200
43	华夏银行	2015-09-30 00:00:00	首钢总公司	20.2800
44	欧菲光	2015-09-30 00:00:00	深圳市欧菲投资控股有限公司	20.3200
45	东华软件	2015-09-30 00:00:00	北京东华诚信电脑科技发展有限公司	20.3300
46	青岛海尔	2015-09-30 00:00:00	海尔电器国际股份有限公司	20.5600
47	恒生电子	2015-09-30 00:00:00	杭州恒生电子集团有限公司	20.7200
48	华谊兄弟	2015-09-30 00:00:00	王忠军	20.8200
49	庞大集团	2015-09-30 00:00:00	庞庆华	21.0300
50	海格通信	2015-09-30 00:00:00	广州无线电集团有限公司	21.2200
51	网宿科技	2015-09-30 00:00:00	陈宝珍	21.3000
52	东旭光电	2015-12-17 00:00:00	东旭集团有限公司	21.6400

53	金地集团	2015-09-30 00:00:00	富德生命人寿保险股份有限公司-万能H	21.7700
54	碧水源	2015-09-30 00:00:00	文剑平	21.8200
55	乐普医疗	2015-09-30 00:00:00	中国船舶重工集团公司第七二五研究所(洛阳船舶材料研究所)	21.9000
56	国元证券	2015-09-30 00:00:00	安徽国元控股(集团)有限责任公司	21.9900
57	广发证券	2015-09-30 00:00:00	香港中央结算(代理人)有限公司	22.3100
58	国海证券	2015-09-30 00:00:00	广西投资集团有限公司	22.3400
59	杰瑞股份	2015-09-30 00:00:00	孙伟杰	22.3600
60	掌趣科技	2015-09-30 00:00:00	姚文彬	22.5200
61	中恒集团	2015-09-30 00:00:00	广西中恒实业有限公司	22.5200
62	双鹭药业	2015-09-30 00:00:00	徐明波	22.5500
63	东阿阿胶	2015-09-30 00:00:00	华润东阿阿胶有限公司	23.1400
64	四川长虹	2015-09-30 00:00:00	四川长虹电子控股集团有限公司	23.2000
65	中航动控	2015-09-30 00:00:00	西安航空动力控制有限责任公司	23.3400
		2015-		

66	光大银行	2015-09-30 00:00:00	中国光大集团股份公司	23.6900
67	华泰证券	2015-09-30 00:00:00	香港中央结算(代理人)有限公司	23.9500
68	五矿稀土	2015-09-30 00:00:00	五矿稀土集团有限公司	23.9800
69	潍柴动力	2015-09-30 00:00:00	香港中央结算代理人有限公司	24.2300
70	恒瑞医药	2015-09-30 00:00:00	江苏恒瑞医药集团有限公司	24.3000
71	金螳螂	2015-09-30 00:00:00	苏州金螳螂企业(集团)有限公司	24.7000
72	招商证券	2015-09-30 00:00:00	深圳市招融投资控股有限公司	24.7100
73	吉林敖东	2015-09-30 00:00:00	敦化市金诚实业有限责任公司	25.0200
74	机器人	2015-11-19 00:00:00	中国科学院沈阳自动化研究所	25.2700
75	西部证券	2015-09-30 00:00:00	陕西省电力建设投资开发公司	25.3300
76	同方股份	2015-09-30 00:00:00	清华控股有限公司	25.4200
77	歌尔声学	2015-09-30 00:00:00	潍坊歌尔集团有限公司	25.5700
78	国泰君安	2015-09-30 00:00:00	上海国有资产经营有限公司	25.6300
		2015-		

79	东吴证券	09-30 00:00:00	苏州国际发展集团有限公司	25.6800
80	九州通	2015-09-30 00:00:00	上海弘康实业投资有限公司	26.3000
81	科伦药业	2015-09-30 00:00:00	刘革新	26.3300
82	苏宁云商	2015-09-30 00:00:00	张近东	26.4400
83	泸州老窖	2015-09-30 00:00:00	泸州老窖集团有限责任公司	26.4800
84	交通银行	2015-09-30 00:00:00	中华人民共和国财政部	26.5300
85	华策影视	2015-11-27 00:00:00	傅梅城	26.7200
86	紫金矿业	2015-09-30 00:00:00	香港中央结算代理人有限公司	26.7300
87	农产品	2015-09-30 00:00:00	深圳市人民政府国有资产监督管理委员会	26.7600
88	西南证券	2015-09-30 00:00:00	重庆渝富资产经营管理集团有限公司	26.9900
89	福田汽车	2015-09-30 00:00:00	北京汽车集团有限公司	27.0700
90	上海家化	2015-09-30 00:00:00	上海家化(集团)有限公司	27.0700
91	机器人	2015-09-30 00:00:00	中国科学院沈阳自动化研究所	27.3700
92	梅花生物	2015-09-30	孟庆山	27.4800

		00:00:00		
93	金融街	2015-09-30 00:00:00	北京金融街投资(集团)有限公司	27.5200
94	海虹控股	2015-09-30 00:00:00	中海恒实业发展有限公司	27.6200
95	方正证券	2015-09-30 00:00:00	北大方正集团有限公司	27.7500
96	比亚迪	2015-09-30 00:00:00	HKSCCNOMINEESLIMITED	27.8200
97	上海医药	2015-09-30 00:00:00	HKSCC NOMINEES LIMITED	27.8200
98	东方财富	2015-09-30 00:00:00	其实	27.9400
99	中环股份	2015-12-18 00:00:00	天津中环电子信息集团有限公司	27.9500
100	海南航空	2015-09-30 00:00:00	大新华航空有限公司	28.1900
101	中直股份	2015-09-30 00:00:00	哈尔滨航空工业(集团)有限公司	28.2100
102	用友网络	2015-09-30 00:00:00	北京用友科技有限公司	28.5200
103	新希望	2015-09-30 00:00:00	南方希望实业有限公司	29.4100
104	中金岭南	2015-09-30 00:00:00	广东省广晟资产经营有限公司	29.5800
105	海通证券	2015-09-30 00:00:00	香港中央结算(代理人)有限公司	29.6400

		00:00:00	司	
106	光大证券	2015-09-30 00:00:00	中国光大集团股份公司	29.6800
107	华策影视	2015-09-30 00:00:00	傅梅城	29.7000
108	三安光电	2015-12-14 00:00:00	厦门三安电子有限公司	29.7600

可以看到伊利，格力等等现金牛企业，以及被不断举牌的浦发等企业，目前大股东的股份占比均小于30%。

以上仅供参考，不作为任何买与卖的建议！

祝您投资愉快！

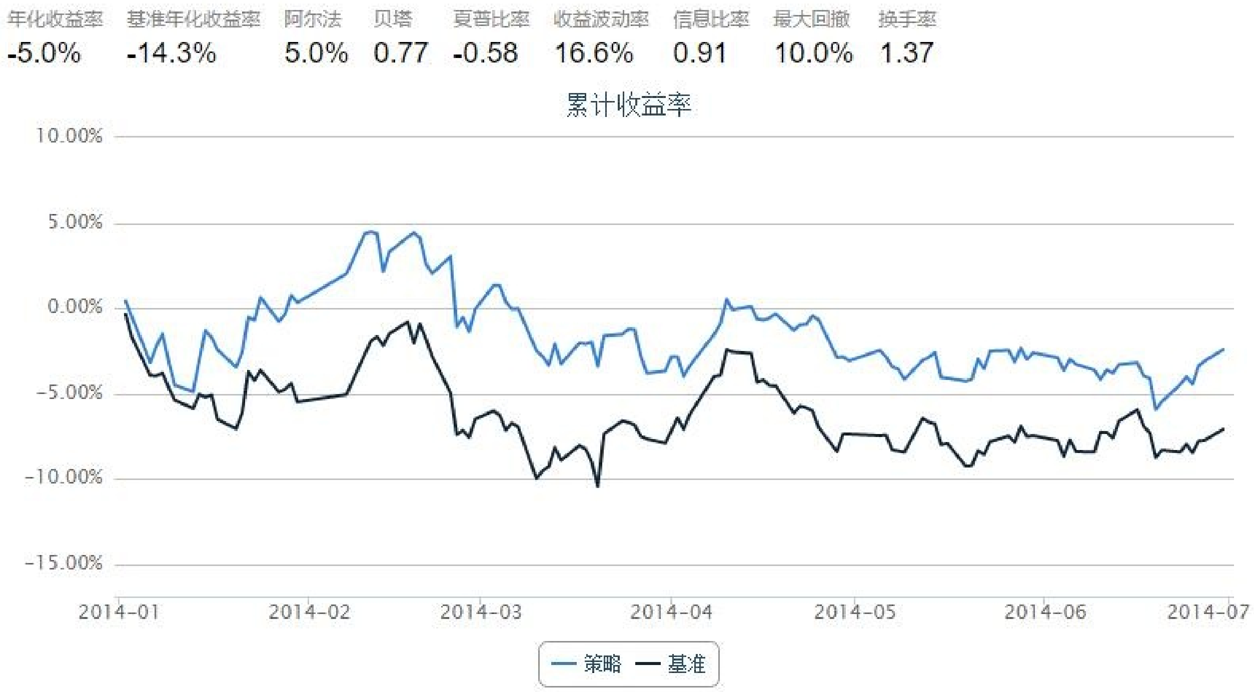
四 技术分析

4.1 布林带

布林带交易策略

> 来源：https://uqer.io/community/share/5673d440228e5bab38c97782

```
from CAL.PyCAL import *
from datetime import *
import time
import pandas as pd
import numpy  as np
import math

start = datetime(2014, 1, 1)
end   = datetime(2014, 6, 30)
benchmark = 'HS300'
universe = set_universe('HS300')
capital_base = 100000
refresh_rate = 1
##########################################
t1 = 50    #MA周期
t2 = 30    #ROC周期
MaxBar = 0.75 #持仓周期最大时，首次卖出系数
##########################################
T =  pd.Series(data=[t1],index = universe)
commission = Commission(buycost=0.0003, sellcost=0.0003)
def initialize(account):
    pass

def handle_data(account):
    #print account.current_date
    cal = Calendar('China.SSE')
    last_dayt1 = cal.advanceDate(account.current_date, str(-t1)+
'B', BizDayConvention.Preceding).toDateTime()           #计算出前
t1个交易日
    buylist = []
    selllist = []

    #取出当前交易日至前t1交易日之间的收盘价格数据
    cp = DataAPI.MktEqudGet(secID=account.universe,ticker=u"",tr
adeDate=u"",beginDate=last_dayt1,endDate=account.current_date,fi
eld=[u"tradeDate",u"secID",u"closePrice"],pandas="1")

    #根据持仓周期，更新MA计算周期T，持仓周期增加1，MA计算周期就减少1，最小
为10
    for stock in account.avail_secpos:
        T[stock] =  T[stock] - 1
        if(T[stock] < 10):
            T[stock] = 10
    #计算t1周期MA
    cpg = cp['closePrice'].groupby(cp['secID'])
    ma = cpg.mean()
    std = cpg.std()

    #当股票当前价格突破布林线上轨，且ROC值大于0，买入
    for stock in account.universe:
        upband = ma[stock] + std[stock]
        if(len(cp[cp.secID == stock]) > t2):
            roc = (account.referencePrice[stock] - float(cp[cp.secID == stock][-t2:-t2+1]['closePrice'])) / float(cp[cp.secID == stock][-t2:-t2+1]['closePrice'])
            if account.referencePrice[stock] > upband and roc > 0:
                buylist.append(stock)
    #当股票当前价格突破布林线中轨，且ROC值小于0，卖出
    for stock in account.avail_secpos:
        #根据股票的持仓周期计算，MA周期
        MAT = np.mean(cp[cp.secID == stock][-T[stock]:]['closePrice'])
        stdT = np.std(cp[cp.secID == stock][-T[stock]:]['closePrice'])
        midband = MAT
        downband = MAT - stdT
        if(len(cp[cp.secID == stock]) > t2):
            roc = (account.referencePrice[stock] - float(cp[cp.secID == stock][-t2:-t2+1]['closePrice'])) / float(cp[cp.secID == stock][-t2:-t2+1]['closePrice'])
            if account.referencePrice[stock] < midband  and roc < 0:
                selllist.append(stock)
    #买入策略，虚拟账户剩余金额按可买股票平均买入，0.95为成功成交系数
    for i in buylist:
        order(i, account.cash / len(buylist) / account.referencePrice[i] * 0.95)
    #卖出策略，按持仓周期逐步卖出，持仓周期越长，第一次卖出越多，最多为3/4仓位，以10天为单位递减
    for i in selllist:
        x = (account.avail_secpos[i] * MaxBar*math.pow(0.5,T[i]// 10 -1) // 100) * 100
        order(i,-x)
        if(x == 100):
             T[i] = t1
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
-5.0%	-14.3%	5.0%	0.77	-0.58	16.6%	0.91	10.0%	1.37

布林带回调系统-日内

> 来源：https://uqer.io/community/share/566929a4f9f06c6c8a91b6e6

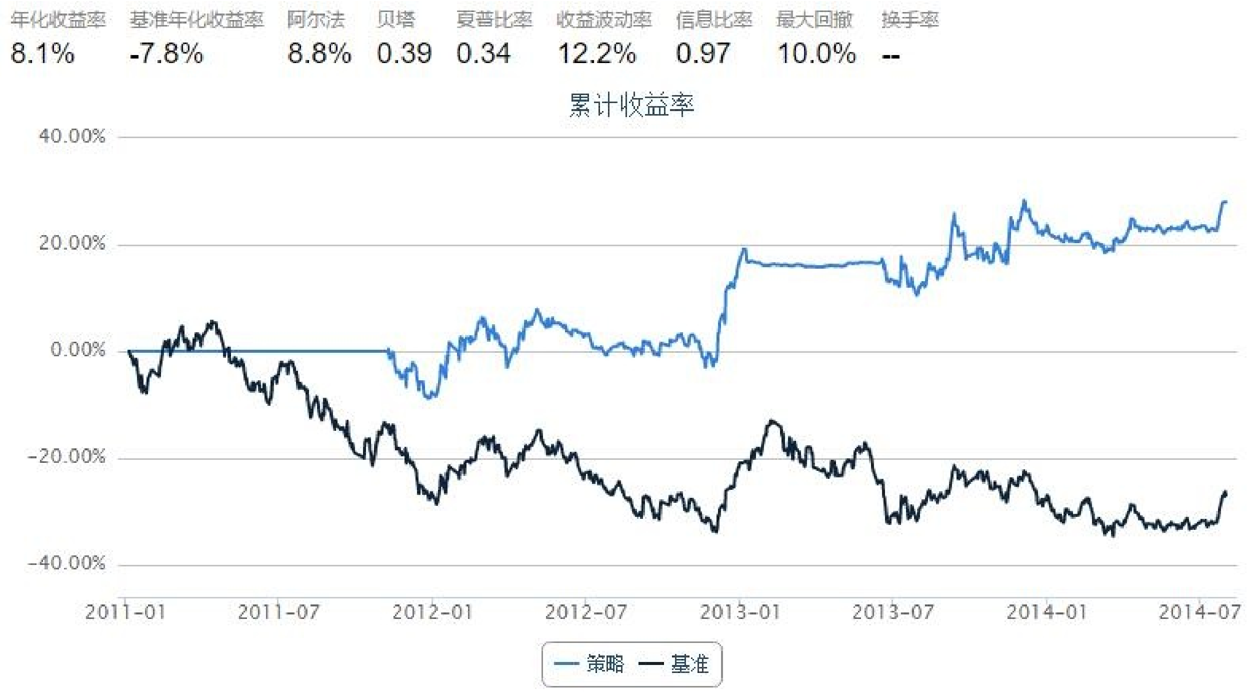
```
import numpy as np
import pandas as pd
from pandas import DataFrame
start = '2014-01-01'                       # 回测起始时间
end = '2015-01-01'                         # 回测结束时间
benchmark = 'HS300'                        # 策略参考标准
universe = set_universe('HS300')           # 证券池，支持股票和基金
capital_base = 100000                      # 起始资金
freq = 'm'                                 # 策略类型，'d'表示日间
策略使用日线回测，'m'表示日内策略使用分钟线回测
refresh_rate = 239                         # 调仓频率，表示执行hand
le_data的时间间隔，若freq = 'd'时间间隔的单位为交易日，若freq = 'm'时间
间隔为分钟
period = 10
multiple=1.5
threshold=-0.1
boll=pd.DataFrame(index=universe,columns = ['mean_cp','high_chan
nel','low_channel'])

def initialize(account):                   # 初始化虚拟账户状态
    pass

def handle_data(account):                  # 每个交易日的买入卖出指令

    if(account.current_minute=='09:30'):
        close_prices = account.get_daily_attribute_history('clos
ePrice', period)
        for s in account.universe:
            mean_cp = close_prices[s].mean()
            bias = multiple*np.std(close_prices[s])
            high_channel = mean_cp + bias
            low_channel = mean_cp - bias
            boll.at[s,'high_channel']=high_channel
            boll.at[s,'low_channel']=low_channel
            boll.at[s,'mean_cp']=mean_cp
    elif(account.current_minute=='14:50'):
        print account.current_date,",",account.valid_secpos
    else:
        for s in account.valid_secpos:     #清仓
            if account.referencePrice[s]>=boll.at[s,'mean_cp'] :
                order_to(s, 0)
        buylist=[]
        c = account.referencePortfolioValue
        for s in account.universe:
            if ((account.referencePrice[s]-boll.at[s,'low_channe
l'])/boll.at[s,'low_channel'])<=threshold:
                buylist.append(s)
        if (len(buylist)==0):
            return
        else:
            w=min(0.2,1.0/len(buylist))# 最大仓位1/5
            for s in buylist:
                p=account.referencePrice[s]*1.01
                num=int(c * w / p)
                order(s, num)
```

Conservative Bollinger Bands

> 来源：https://uqer.io/community/share/548575def9f06c8e77336728

```
import quartz
import quartz.backtest    as qb
import quartz.performance as qp
from   quartz.api         import *

import pandas as pd
import numpy  as np
from datetime   import datetime
from matplotlib import pylab

import talib
```

```
start = datetime(2011, 1, 1)
end = datetime(2014, 8, 1)
benchmark = 'HS300'
universe = ['601398.XSHG', '600028.XSHG', '601988.XSHG', '600036.XSHG', '600030.XSHG',
            '601318.XSHG', '600000.XSHG', '600019.XSHG', '600519.XSHG', '601166.XSHG']
capital_base = 1000000
refresh_rate = 5
window = 200

def initialize(account):
    account.amount = 10000
    account.universe = universe
    add_history('hist', window)

def handle_data(account, data):

    for stk in account.universe:
        prices = account.hist[stk]['closePrice']
        if prices is None:
            return

        mu = prices.mean()
        sd = prices.std()

        upper = mu + 1*sd
        middle = mu
        lower = mu - 1*sd


        cur_pos = account.position.stkpos.get(stk, 0)
        cur_prc = prices[-1]
        if cur_prc > upper and cur_pos >= 0:
            order_to(stk, 0)
        if cur_prc < lower and cur_pos <= 0:
            order(stk, account.amount)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
8.1%	-7.8%	8.8%	0.39	0.34	12.2%	0.97	10.0%	--

```
bt
```

	tradeDate	cash	stock_position	portfolio_value	benchma
0	2011-01-04	1000000	{}	1000000	0.000000
1	2011-01-05	1000000	{}	1000000	-0.004395
2	2011-01-06	1000000	{}	1000000	-0.005044
3	2011-01-07	1000000	{}	1000000	0.002209
4	2011-01-10	1000000	{}	1000000	-0.018454
5	2011-01-11	1000000	{}	1000000	0.005384
6	2011-01-12	1000000	{}	1000000	0.005573
7	2011-01-13	1000000	{}	1000000	-0.000335
8	2011-01-14	1000000	{}	1000000	-0.015733
9	2011-01-	1000000	{}	1000000	-0.038007

9	17	1000000	{}	1000000	-0.038007
10	2011-01-18	1000000	{}	1000000	0.001109
11	2011-01-19	1000000	{}	1000000	0.022569
12	2011-01-20	1000000	{}	1000000	-0.032888
13	2011-01-21	1000000	{}	1000000	0.013157
14	2011-01-24	1000000	{}	1000000	-0.009795
15	2011-01-25	1000000	{}	1000000	-0.005273
16	2011-01-26	1000000	{}	1000000	0.013536
17	2011-01-27	1000000	{}	1000000	0.016128
18	2011-01-28	1000000	{}	1000000	0.003393
19	2011-01-31	1000000	{}	1000000	0.013097
20	2011-02-01	1000000	{}	1000000	0.000252
21	2011-02-09	1000000	{}	1000000	-0.011807
22	2011-02-10	1000000	{}	1000000	0.020788
23	2011-02-11	1000000	{}	1000000	0.005410
24	2011-02-14	1000000	{}	1000000	0.031461
25	2011-02-15	1000000	{}	1000000	-0.000457
26	2011-02-16	1000000	{}	1000000	0.009590
27	2011-02-	1000000	{}	1000000	-0.000807

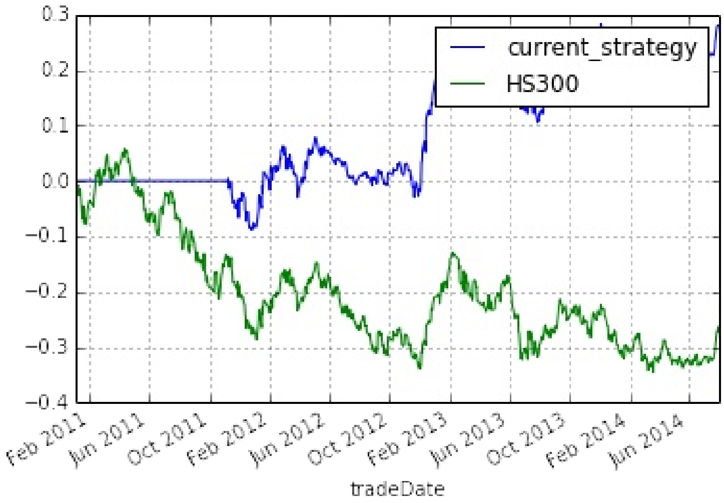
28	2011-02-18	1000000	{}	1000000	-0.010484
29	2011-02-21	1000000	{}	1000000	0.014332
30	2011-02-22	1000000	{}	1000000	-0.028954
31	2011-02-23	1000000	{}	1000000	0.003529
32	2011-02-24	1000000	{}	1000000	0.005101
33	2011-02-25	1000000	{}	1000000	0.002094
34	2011-02-28	1000000	{}	1000000	0.013117
35	2011-03-01	1000000	{}	1000000	0.004733
36	2011-03-02	1000000	{}	1000000	-0.003562
37	2011-03-03	1000000	{}	1000000	-0.006654
38	2011-03-04	1000000	{}	1000000	0.015193
39	2011-03-07	1000000	{}	1000000	0.019520
40	2011-03-08	1000000	{}	1000000	0.000884
41	2011-03-09	1000000	{}	1000000	0.000420
42	2011-03-10	1000000	{}	1000000	-0.017551
43	2011-03-11	1000000	{}	1000000	-0.010025
44	2011-03-14	1000000	{}	1000000	0.004787
45	2011-03-15	1000000	{}	1000000	-0.018069

46	2011-03-16	1000000	{}	1000000	0.013806
47	2011-03-17	1000000	{}	1000000	-0.015730
48	2011-03-18	1000000	{}	1000000	0.005813
49	2011-03-21	1000000	{}	1000000	-0.002667
50	2011-03-22	1000000	{}	1000000	0.004942
51	2011-03-23	1000000	{}	1000000	0.013021
52	2011-03-24	1000000	{}	1000000	-0.004155
53	2011-03-25	1000000	{}	1000000	0.013263
54	2011-03-28	1000000	{}	1000000	-0.001188
55	2011-03-29	1000000	{}	1000000	-0.009905
56	2011-03-30	1000000	{}	1000000	-0.000583
57	2011-03-31	1000000	{}	1000000	-0.010071
58	2011-04-01	1000000	{}	1000000	0.015339
59	2011-04-06	1000000	{}	1000000	0.011714
...	...	...	...	...	...

```
868 rows × 6 columns
```

```
perf = qp.perf_parse(bt)
out_keys = ['annualized_return', 'volatility', 'information',
            'sharpe', 'max_drawdown', 'alpha', 'beta']

for k in out_keys:
    print '%s: %s' % (k, perf[k])

annualized_return: 0.0806072460858
volatility: 0.121542243584
information: 0.967129870018
sharpe: 0.344919139631
max_drawdown: 0.100359317734
alpha: 0.0876204656402
beta: 0.392712356147
```

```
perf['cumulative_return'].plot()
perf['benchmark_cumulative_return'].plot()
pylab.legend(['current_strategy','HS300'])
```

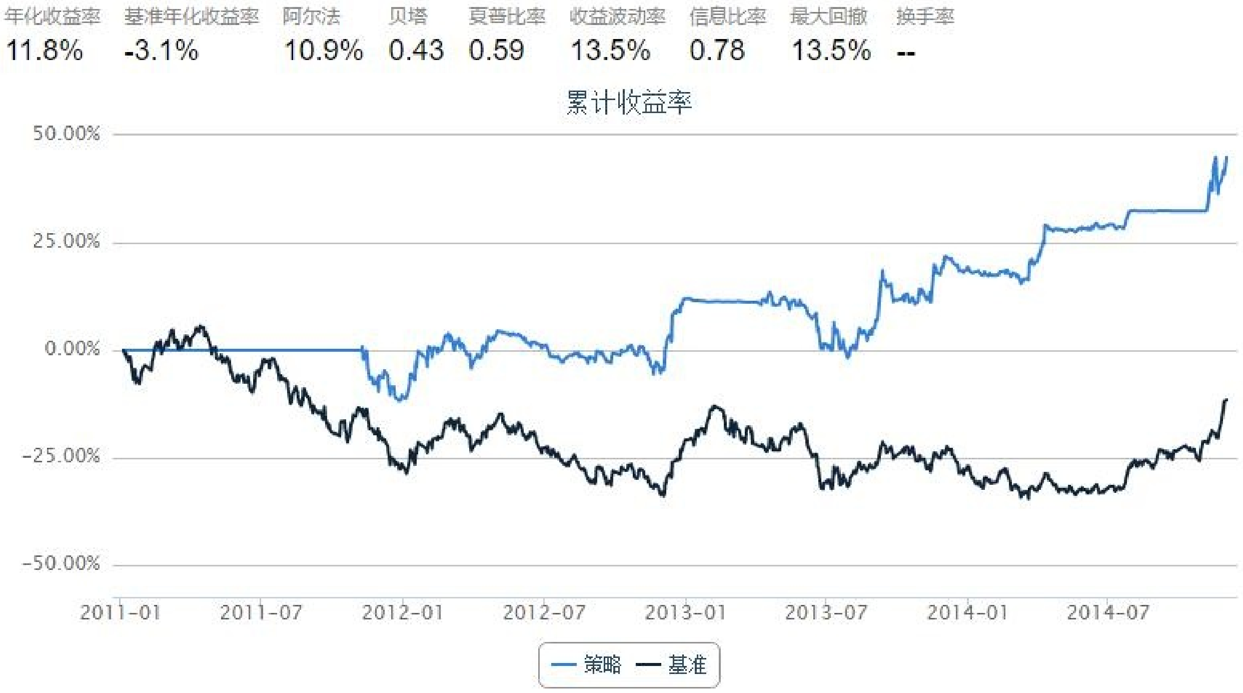


Even More Conservative Bollinger Bands

> 来源：https://uqer.io/community/share/54859edff9f06c8e77336729

```
import quartz
import quartz.backtest    as qb
import quartz.performance as qp
from   quartz.api         import *

import pandas as pd
import numpy  as np
from datetime   import datetime
from matplotlib import pylab

import talib
```

```
start = datetime(2011, 1, 1)
end = datetime(2014, 12, 1)
benchmark = 'HS300'
universe = ['601398.XSHG', '600028.XSHG', '601988.XSHG', '600036
.XSHG', '600030.XSHG',
            '601318.XSHG', '600000.XSHG', '600019.XSHG', '600519
.XSHG', '601166.XSHG']
capital_base = 1000000
refresh_rate = 5
window = 200

def initialize(account):
    account.amount = 10000
    account.universe = universe
    add_history('hist', window)

def handle_data(account, data):

    for stk in account.universe:
        prices = account.hist[stk]['closePrice']
        if prices is None:
            return

        mu = prices.mean()
        sd = prices.std()

        upper = mu + .5*sd
        middle = mu
        lower = mu - .5*sd


        cur_pos = account.position.stkpos.get(stk, 0)
        cur_prc = prices[-1]
        if cur_prc > upper and cur_pos >= 0:
            order_to(stk, 0)
        if cur_prc < lower and cur_pos <= 0:
            order(stk, account.amount)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
11.8%	-3.1%	10.9%	0.43	0.59	13.5%	0.78	13.5%	--

```
bt
```

	tradeDate	cash	stock_position	portfolio_value	benchma
0	2011-01-04	1000000	{}	1000000	0.000000
1	2011-01-05	1000000	{}	1000000	-0.004395
2	2011-01-06	1000000	{}	1000000	-0.005044
3	2011-01-07	1000000	{}	1000000	0.002209
4	2011-01-10	1000000	{}	1000000	-0.018454
5	2011-01-11	1000000	{}	1000000	0.005384
6	2011-01-12	1000000	{}	1000000	0.005573
7	2011-01-13	1000000	{}	1000000	-0.000335
8	2011-01-14	1000000	{}	1000000	-0.015733
	2011-01-				

	17				
10	2011-01-18	1000000	{}	1000000	0.001109
11	2011-01-19	1000000	{}	1000000	0.022569
12	2011-01-20	1000000	{}	1000000	-0.032888
13	2011-01-21	1000000	{}	1000000	0.013157
14	2011-01-24	1000000	{}	1000000	-0.009795
15	2011-01-25	1000000	{}	1000000	-0.005273
16	2011-01-26	1000000	{}	1000000	0.013536
17	2011-01-27	1000000	{}	1000000	0.016128
18	2011-01-28	1000000	{}	1000000	0.003393
19	2011-01-31	1000000	{}	1000000	0.013097
20	2011-02-01	1000000	{}	1000000	0.000252
21	2011-02-09	1000000	{}	1000000	-0.011807
22	2011-02-10	1000000	{}	1000000	0.020788
23	2011-02-11	1000000	{}	1000000	0.005410
24	2011-02-14	1000000	{}	1000000	0.031461
25	2011-02-15	1000000	{}	1000000	-0.000457
26	2011-02-16	1000000	{}	1000000	0.009590
27	2011-02-17	1000000	{}	1000000	-0.000807

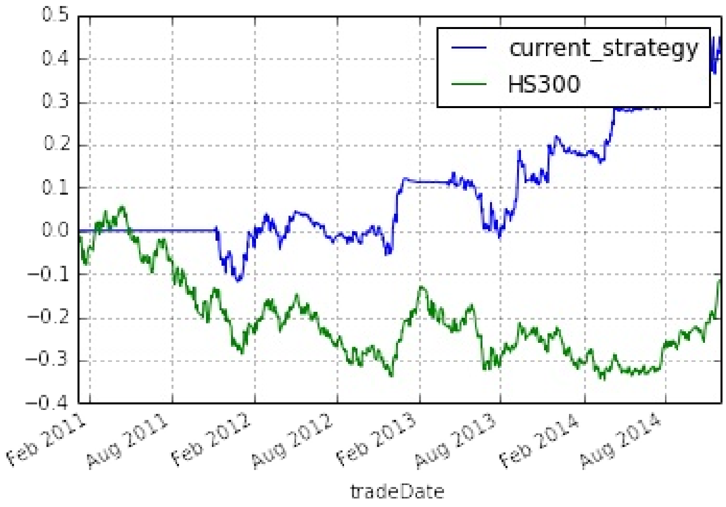
	17				
28	2011-02-18	1000000	{}	1000000	-0.010484
29	2011-02-21	1000000	{}	1000000	0.014332
30	2011-02-22	1000000	{}	1000000	-0.028954
31	2011-02-23	1000000	{}	1000000	0.003529
32	2011-02-24	1000000	{}	1000000	0.005101
33	2011-02-25	1000000	{}	1000000	0.002094
34	2011-02-28	1000000	{}	1000000	0.013117
35	2011-03-01	1000000	{}	1000000	0.004733
36	2011-03-02	1000000	{}	1000000	-0.003562
37	2011-03-03	1000000	{}	1000000	-0.006654
38	2011-03-04	1000000	{}	1000000	0.015193
39	2011-03-07	1000000	{}	1000000	0.019520
40	2011-03-08	1000000	{}	1000000	0.000884
41	2011-03-09	1000000	{}	1000000	0.000420
42	2011-03-10	1000000	{}	1000000	-0.017551
43	2011-03-11	1000000	{}	1000000	-0.010025
44	2011-03-14	1000000	{}	1000000	0.004787
45	2011-03-15	1000000	{}	1000000	-0.018069

46	2011-03-16	1000000	{}	1000000	0.013806
47	2011-03-17	1000000	{}	1000000	-0.015730
48	2011-03-18	1000000	{}	1000000	0.005813
49	2011-03-21	1000000	{}	1000000	-0.002667
50	2011-03-22	1000000	{}	1000000	0.004942
51	2011-03-23	1000000	{}	1000000	0.013021
52	2011-03-24	1000000	{}	1000000	-0.004155
53	2011-03-25	1000000	{}	1000000	0.013263
54	2011-03-28	1000000	{}	1000000	-0.001188
55	2011-03-29	1000000	{}	1000000	-0.009905
56	2011-03-30	1000000	{}	1000000	-0.000583
57	2011-03-31	1000000	{}	1000000	-0.010071
58	2011-04-01	1000000	{}	1000000	0.015339
59	2011-04-06	1000000	{}	1000000	0.011714
...	...	...	...	...	...

```
948 rows × 6 columns
```

```
perf = qp.perf_parse(bt)
out_keys = ['annualized_return', 'volatility', 'information',
            'sharpe', 'max_drawdown', 'alpha', 'beta']

for k in out_keys:
    print '%s: %s' % (k, perf[k])

annualized_return: 0.118291633101
volatility: 0.134550735738
information: 0.776689524517
sharpe: 0.591647698281
max_drawdown: 0.135222029922
alpha: 0.109380091075
beta: 0.429849284472
```

```
perf['cumulative_return'].plot()
perf['benchmark_cumulative_return'].plot()
pylab.legend(['current_strategy','HS300'])

<matplotlib.legend.Legend at 0x49c0b10>
```



Simple Bollinger Bands

> 来源：https://uqer.io/community/share/54b5c1b4f9f06c276f651a15

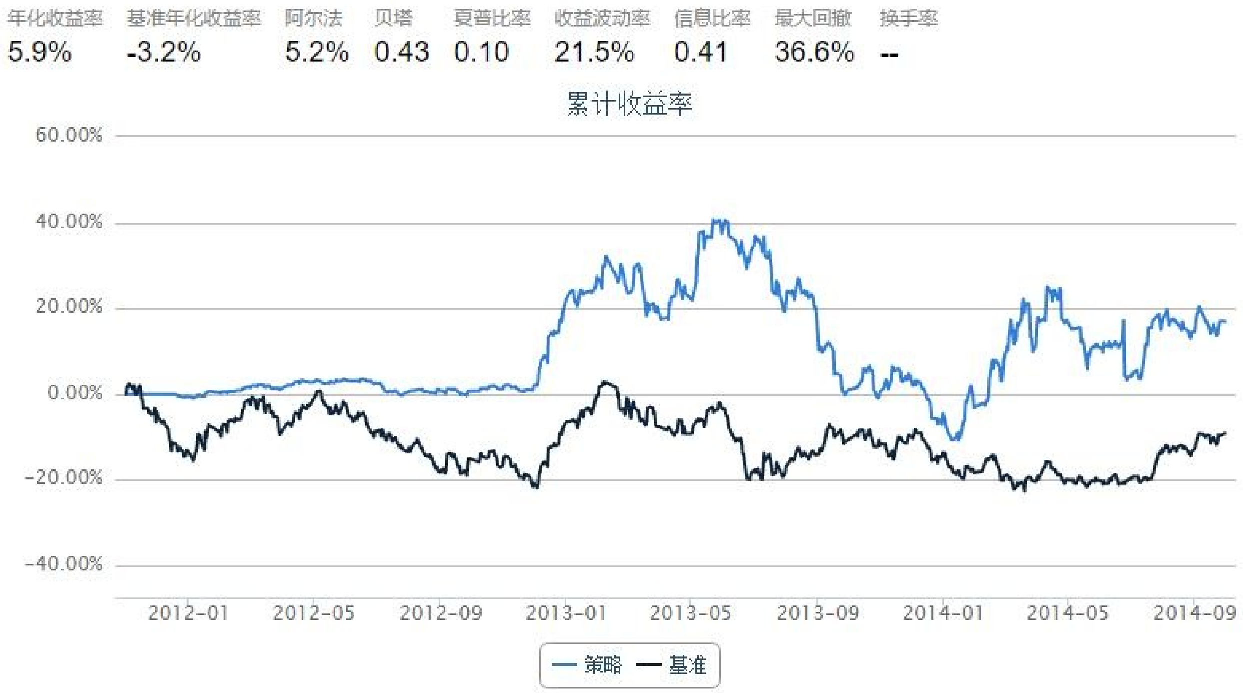
策略思路： 在一段时间内，股票价格可以认为是在一定水平上下波动。例如，可以简单认为股价在过去100天的平均值基础上，上下波动几个标准差。

策略实现： (1) 股价高于这个波动区间，说明股价虚高，故卖出 (2) 股价低于这个波动区间，说明股价虚低，故买入

PS：本策略适用于股市平稳波动时期，在大牛市或者大熊市不太适用

```
import quartz
import quartz.backtest    as qb
import quartz.performance as qp
from   quartz.api         import *

import pandas as pd
import numpy  as np
from datetime   import datetime
from matplotlib import pylab

import talib
```

```
start = datetime(2011, 1, 1)
end = datetime(2014, 10, 1)
benchmark = 'HS300'
universe = ['601398.XSHG', '600028.XSHG', '601988.XSHG', '600036.XSHG', '600030.XSHG',
            '601318.XSHG', '600000.XSHG', '600019.XSHG', '600519.XSHG', '601166.XSHG']
capital_base = 1000000
refresh_rate = 5
window = 200

def initialize(account):
    account.amount = 10000
    account.universe = universe
    add_history('hist', window)

def handle_data(account):

    for stk in account.universe:
        prices = account.hist[stk]['closePrice']
        if prices is None:
            return

        mu = prices.mean()
        sd = prices.std()

        upper = mu + 2*sd
        middle = mu
        lower = mu - 2*sd

        cur_pos = account.position.stkpos.get(stk, 0)
        cur_prc = prices[-1]
        if cur_prc > upper and cur_pos >= 0:
            order_to(stk, 0)
        if cur_prc < lower and cur_pos <= 0:
            order(stk, account.amount)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
5.9%	-3.2%	5.2%	0.43	0.10	21.5%	0.41	36.6%	--

```
bt
```

	tradeDate	cash	stock_position	portfolio_value
0	2011-11-01	1000000.00000	{}	1000000.00000
1	2011-11-02	1000000.00000	{}	1000000.00000
2	2011-11-03	1000000.00000	{}	1000000.00000
3	2011-11-04	1000000.00000	{}	1000000.00000
4	2011-11-07	1000000.00000	{}	1000000.00000
5	2011-11-08	1000000.00000	{}	1000000.00000
6	2011-11-09	1000000.00000	{}	1000000.00000
7	2011-11-10	1000000.00000	{}	1000000.00000
8	2011-11-11	1000000.00000	{}	1000000.00000
	2011-11-			

9	14	1000000.00000	{}	1000000.00000
10	2011-11-15	1000000.00000	{}	1000000.00000
11	2011-11-16	1000000.00000	{}	1000000.00000
12	2011-11-17	1000000.00000	{}	1000000.00000
13	2011-11-18	1000000.00000	{}	1000000.00000
14	2011-11-21	1000000.00000	{}	1000000.00000
15	2011-11-22	1000000.00000	{}	1000000.00000
16	2011-11-23	1000000.00000	{}	1000000.00000
17	2011-11-24	1000000.00000	{}	1000000.00000
18	2011-11-25	1000000.00000	{}	1000000.00000
19	2011-11-28	1000000.00000	{}	1000000.00000
20	2011-11-29	1000000.00000	{}	1000000.00000
21	2011-11-30	1000000.00000	{}	1000000.00000
22	2011-12-01	1000000.00000	{}	1000000.00000
23	2011-12-02	1000000.00000	{}	1000000.00000
24	2011-12-05	1000000.00000	{}	1000000.00000
25	2011-12-06	1000000.00000	{}	1000000.00000
26	2011-12-07	1000000.00000	{}	1000000.00000
27	2011-12-	1000000.00000	{}	1000000.00000

28	2011-12-09	1000000.00000	{}	1000000.00000
29	2011-12-12	1000000.00000	{}	1000000.00000
...	...	...	...	...
679	2014-08-19	96907.47989	{u'601166.XSHG': 3877.0, u'600036.XSHG': 0, u'...	1149017.27989
680	2014-08-20	96907.47989	{u'601166.XSHG': 3877.0, u'600036.XSHG': 0, u'...	1168551.04989
681	2014-08-21	96907.47989	{u'601166.XSHG': 3877.0, u'600036.XSHG': 0, u'...	1155719.88989
682	2014-08-22	96907.47989	{u'601166.XSHG': 3877.0, u'600036.XSHG': 0, u'...	1152026.33989
683	2014-08-25	96907.47989	{u'601166.XSHG': 3877.0, u'600036.XSHG': 0, u'...	1148250.45989
684	2014-08-26	96907.47989	{u'601166.XSHG': 3877.0, u'600036.XSHG': 0, u'...	1141866.53989
685	2014-08-27	96907.47989	{u'601166.XSHG': 3877.0, u'600036.XSHG': 0, u'...	1133729.60989
686	2014-08-28	96907.47989	{u'601166.XSHG': 3877.0, u'600036.XSHG': 0, u'...	1130309.43989
687	2014-08-29	96907.47989	{u'601166.XSHG': 3877.0, u'600036.XSHG':	1144849.97989

	29		u'600036.XSHG': 0, u'...	
688	2014-09-01	96907.47989	{u'601166.XSHG': 3877.0, u'600036.XSHG': 0, u'...	1146127.24989
689	2014-09-02	96907.47989	{u'601166.XSHG': 3877.0, u'600036.XSHG': 0, u'...	1171994.81989
690	2014-09-03	96907.47989	{u'601166.XSHG': 3877.0, u'600036.XSHG': 0, u'...	1186363.06989
691	2014-09-04	96907.47989	{u'601166.XSHG': 3877.0, u'600036.XSHG': 0, u'...	1188453.82989
692	2014-09-05	96907.47989	{u'601166.XSHG': 3877.0, u'600036.XSHG': 0, u'...	1203626.03989
693	2014-09-09	96907.47989	{u'601166.XSHG': 3877.0, u'600036.XSHG': 0, u'...	1182303.76989
694	2014-09-10	96907.47989	{u'601166.XSHG': 3877.0, u'600036.XSHG': 0, u'...	1180321.58989
695	2014-09-11	96907.47989	{u'601166.XSHG': 3877.0, u'600036.XSHG': 0, u'...	1168537.55989
696	2014-09-12	96907.47989	{u'601166.XSHG': 3877.0, u'600036.XSHG': 0, u'...	1171787.86989
697	2014-09-15	96907.47989	{u'601166.XSHG': 3877.0, u'600036.XSHG': 0, u'...	1153269.96989

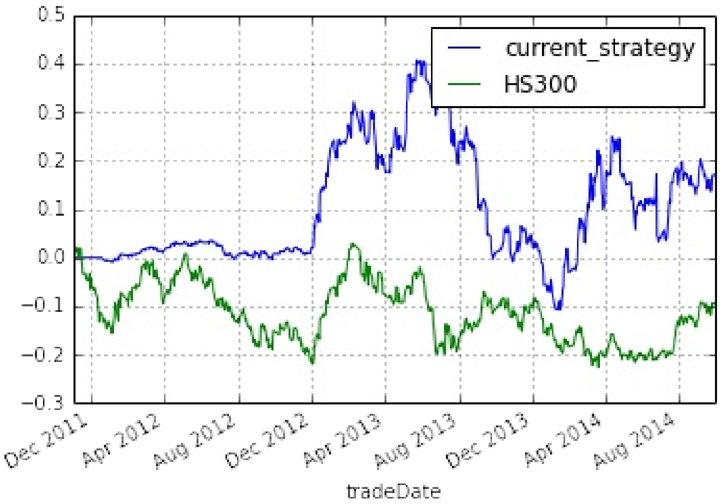
			0, u'...	
698	2014-09-16	96907.47989	{u'601166.XSHG': 3877.0, u'600036.XSHG': 0, u'...	1141194.90989
699	2014-09-17	96907.47989	{u'601166.XSHG': 3877.0, u'600036.XSHG': 0, u'...	1145778.58989
700	2014-09-18	96907.47989	{u'601166.XSHG': 3877.0, u'600036.XSHG': 0, u'...	1152235.30989
701	2014-09-19	96907.47989	{u'601166.XSHG': 3877.0, u'600036.XSHG': 0, u'...	1162000.35989
702	2014-09-22	96907.47989	{u'601166.XSHG': 3877.0, u'600036.XSHG': 0, u'...	1136441.48989
703	2014-09-23	96907.47989	{u'601166.XSHG': 3877.0, u'600036.XSHG': 0, u'...	1139988.07989
704	2014-09-24	96907.47989	{u'601166.XSHG': 3877.0, u'600036.XSHG': 0, u'...	1157613.46989
705	2014-09-25	96907.47989	{u'601166.XSHG': 3877.0, u'600036.XSHG': 0, u'...	1169367.30989
706	2014-09-26	96907.47989	{u'601166.XSHG': 3877.0, u'600036.XSHG': 0, u'...	1169791.29989
707	2014-09-29	96907.47989	{u'601166.XSHG': 3877.0, u'600036.XSHG': 0, u'...	1170503.49989

708	2014-09-30	96907.47989	3877.0, u'600036.XSHG': 0, u'...	1168692.07989

```
709 rows × 6 columns
```

```
perf = qp.perf_parse(bt)
out_keys = ['annualized_return', 'volatility', 'information',
            'sharpe', 'max_drawdown', 'alpha', 'beta']

for k in out_keys:
    print '%s: %s' % (k, perf[k])

annualized_return: 0.0594823977045
volatility: 0.214571802928
information: 0.407163653335
sharpe: 0.102806600884
max_drawdown: 0.36585509048
alpha: 0.052294592526
beta: 0.434097117249
```

```
perf['cumulative_return'].plot()
perf['benchmark_cumulative_return'].plot()
pylab.legend(['current_strategy','HS300'])

<matplotlib.legend.Legend at 0x568fe90>
```



4.2 均线系统

技术分析入门 —— 双均线策略

> 来源：https://uqer.io/community/share/554051bbf9f06c1c3d687fac
>
> 本篇中，我们将通过技术分析流派中经典的“双均线策略”，向大家展现如何在量化实验室中使用Python测试自己的想法，并最终将它转化为策略！

1. 准备工作

一大波Python库需要在使用之前被导入：

- `matplotlib` 用于绘制图表
- `numpy` 时间序列的计算
- `pandas` 处理结构化的表格数据
- `DataAPI` 通联数据提供的数据API
- `seaborn` 用于美化matplotlib图表

```
from matplotlib import pylab
import numpy as np
import pandas as pd
import DataAPI
import seaborn as sns
sns.set_style('white')
```

我们的关注点是关于一只ETF基金的投资：华夏上证50ETF，代码： `510050.XSHG` 。我们考虑的回测周期：

- 起始：2008年1月1日
- 结束：2015年4月23日

这里我们使用数据API函数 `MktFunddGe` t获取基金交易价格的日线数据，最后获得 `security` 是 `pandas` 下的 `DataFrame` 对象：

```
secID = '510050.XSHG'
start = '20080101'
end = '20150423'

security = DataAPI.MktFunddGet(secID, beginDate=start, endDate=end, field=['tradeDate', 'closePrice'])
security['tradeDate'] = pd.to_datetime(security['tradeDate'])
security = security.set_index('tradeDate')
security.info()

<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1775 entries, 2008-01-02 00:00:00 to 2015-04-23 00:00:00
Data columns (total 1 columns):
closePrice    1775 non-null float64
dtypes: float64(1)
```

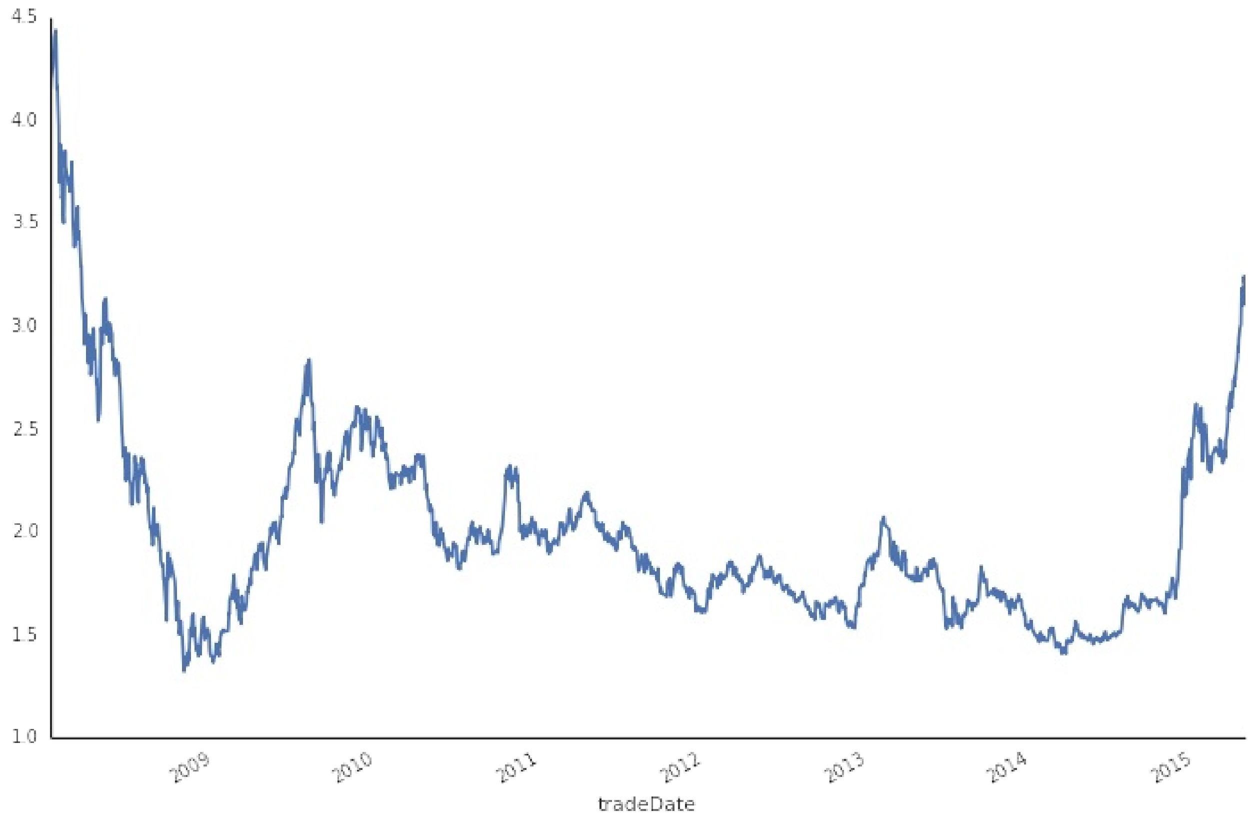
最近5天的收盘价如下：

```
security.tail()
```

	closePrice
tradeDate	
2015-04-17	3.185
2015-04-20	3.103
2015-04-21	3.141
2015-04-22	3.241
2015-04-23	3.212

适当的图表可以帮助研究人员直观的了解标的的历史走势，这里我们直接借助 `DataFrame` 的 `plot` 成员：

```
security['closePrice'].plot(grid=False, figsize=(12,8))
sns.despine()
```



2. 策略描述

> 这里我们以经典的“双均线”策略为例，讲述如何使用量化实验室进行分析研究。

这里我们使用的均线定义为：

- 短期均线： `window_short = 20` ，相当于月均线
- 长期均线： `window_long = 120` ，相当于半年线
- 偏离度阈值： `SD = 5%` ，区间宽度，这个会在后面有详细解释

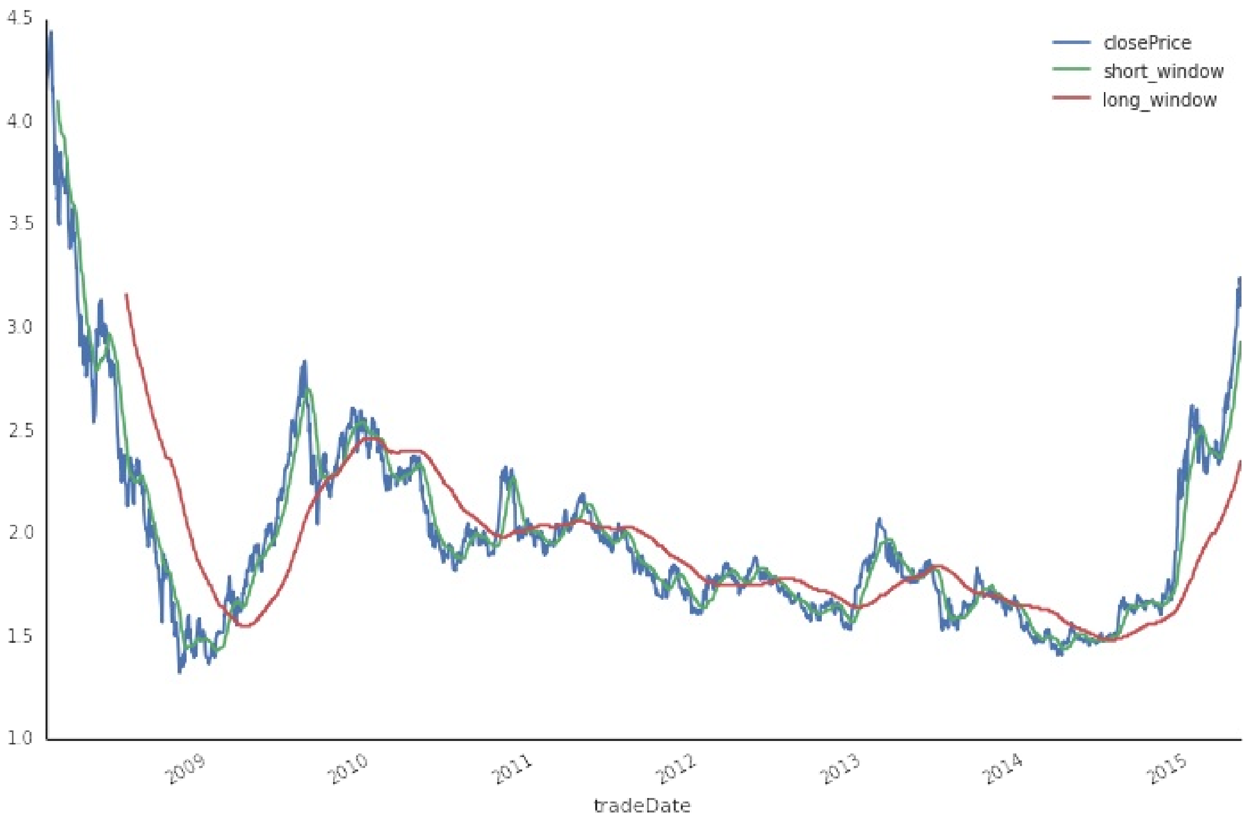
计算均值我们借助了 `numpy` 的内置移动平均函数： `rolling_mean`

```
window_short = 20
window_long = 120
SD = 0.05

security['short_window'] = np.round(pd.rolling_mean(security['closePrice'], window=window_short), 2)
security['long_window'] = np.round(pd.rolling_mean(security['closePrice'], window=window_long), 2)
security[['closePrice', 'short_window', 'long_window']].tail()
```

	closePrice	short_window	long_window
tradeDate			
2015-04-17	3.185	2.82	2.30
2015-04-20	3.103	2.85	2.31
2015-04-21	3.141	2.87	2.33
2015-04-22	3.241	2.90	2.34
2015-04-23	3.212	2.93	2.35

仍然地，我们可以把包含收盘价的三条线画到一张图上，看看有没有什么启发？

```
security[['closePrice', 'short_window', 'long_window']].plot(grid=False, figsize=(12,8))
sns.despine()
```



2.1 定义信号

> 买入信号： 短期均线高于长期日均线，并且超过 `SD` 个点位；
>
> 卖出信号： 不满足买入信号的所有情况；

我们首先计算短期均线与长期均线的差 `s-l` ，这样的向量级运算，在 `pandas` 中可以像普通标量一样计算：

```
security['s-l'] = security['short_window'] - security['long_wind
ow']
security['s-l'].tail()

tradeDate
2015-04-17    0.52
2015-04-20    0.54
2015-04-21    0.54
2015-04-22    0.56
2015-04-23    0.58
Name: s-l, dtype: float64
```

根据 `s-l` 的值，我们可以定义信号：

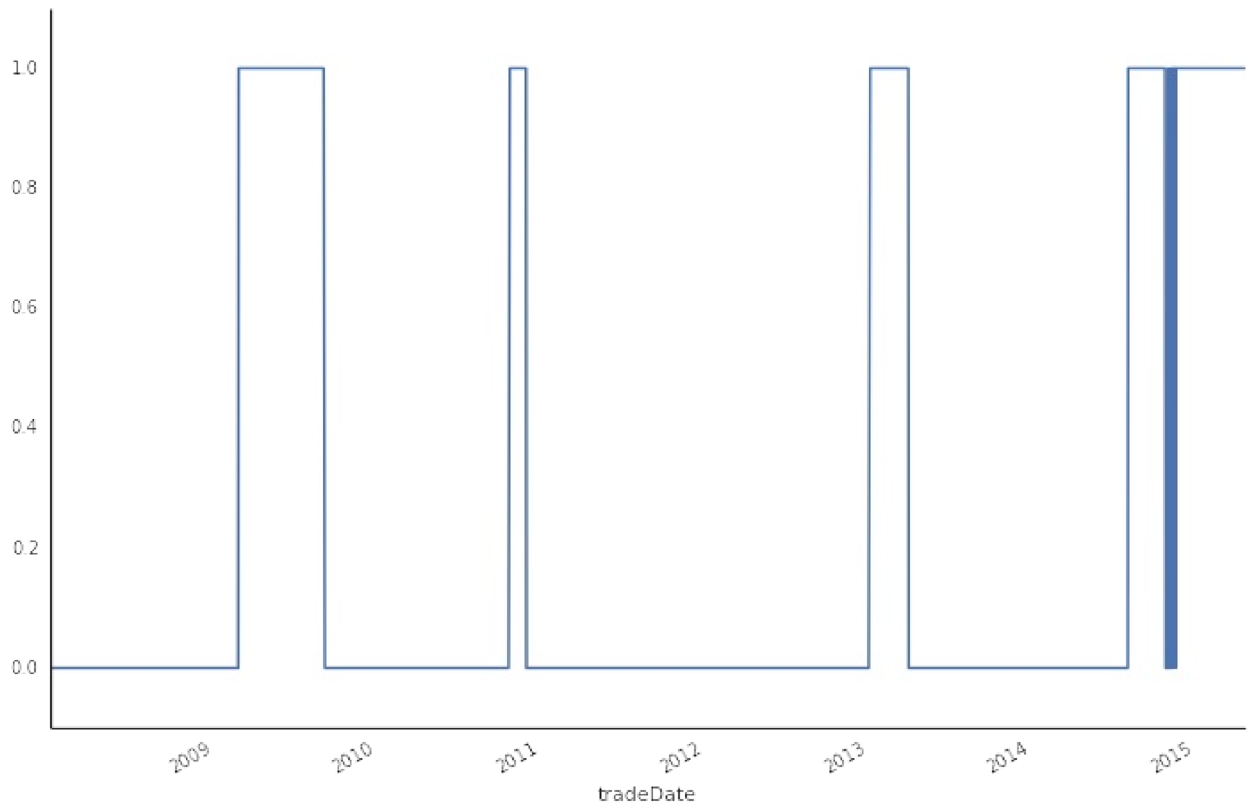
- `s−l>SD×long_window` ，支持买入，定义 `Regime` 为 `True`
- 其他情形下，卖出信号，定义 `Regime` 为 `False`

```
security['Regime'] = np.where(security['s-l'] > security['long_w
indow'] * SD, 1, 0)
security['Regime'].value_counts()

0    1394
1     381
dtype: int64
```

上面的统计给出了总共有多少次买入信号，多少次卖出信号。

下图给出了信号的时间分布：

```
security['Regime'].plot(grid=False, lw=1.5, figsize=(12,8))
pylab.ylim((-0.1,1.1))
sns.despine()
```



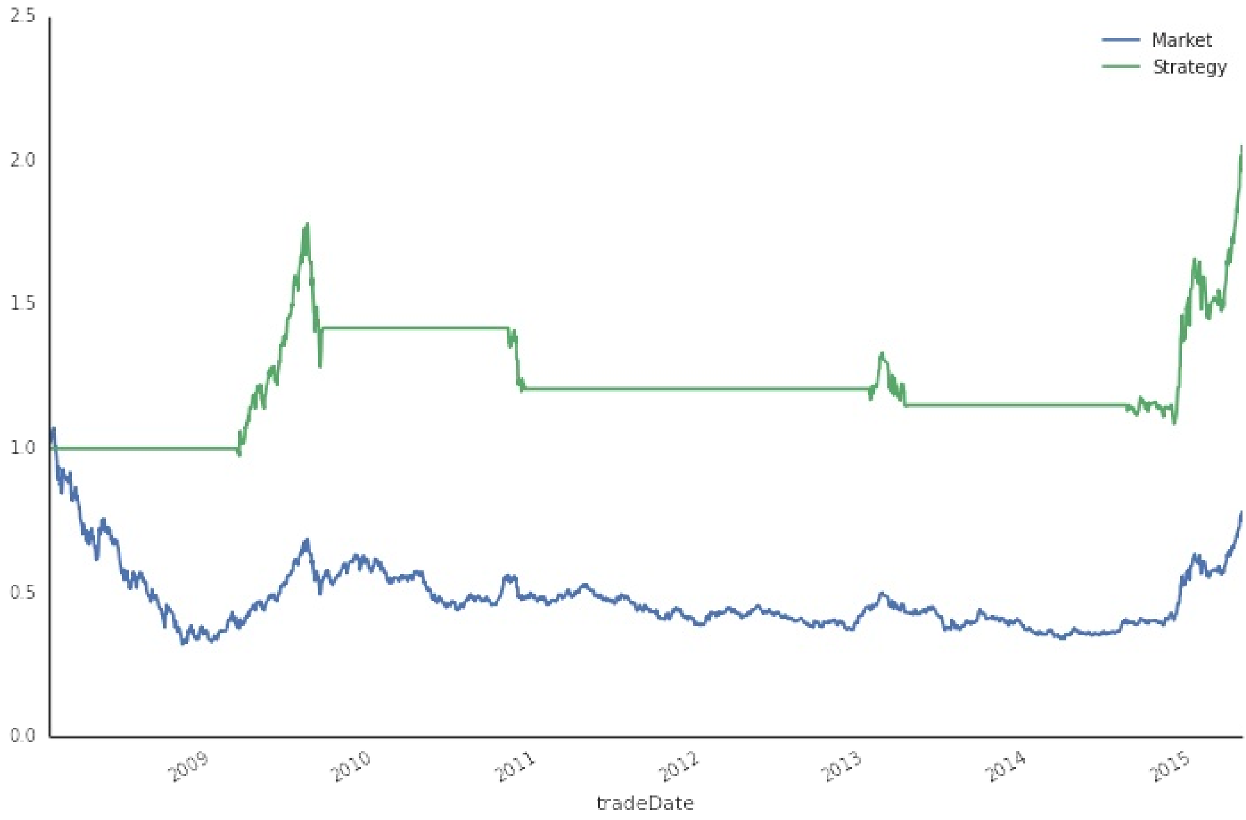
我们可以在有了信号之后执行买入卖出操作，然后根据操作计算每日的收益。这里注意，我们计算策略收益的时候，使用的是当天的信号乘以次日的收益率。这是因为我们的决定是当天做出的，但是能享受到的收益只可能是第二天的（如果用当天信号乘以当日的收益率，那么这里面就有使用未来数据的问题）。

```
security['Market'] = np.log(security['closePrice'] / security['closePrice'].shift(1))
security['Strategy'] = security['Regime'].shift(1) * security['Market']
security[['Market', 'Strategy', 'Regime']].tail()
```

	Market	Strategy	Regime
tradeDate			
2015-04-17	0.012638	0.012638	1
2015-04-20	-0.026083	-0.026083	1
2015-04-21	0.012172	0.012172	1
2015-04-22	0.031341	0.031341	1
2015-04-23	-0.008988	-0.008988	1

最后我们把每天的收益率求和就得到了最后的累计收益率（这里因为我们使用的是指数收益率，所以将每日收益累加是合理的），这个累加的过程也可以通过 `DataFrame` 的内置函数 `cumsum` 轻松完成：

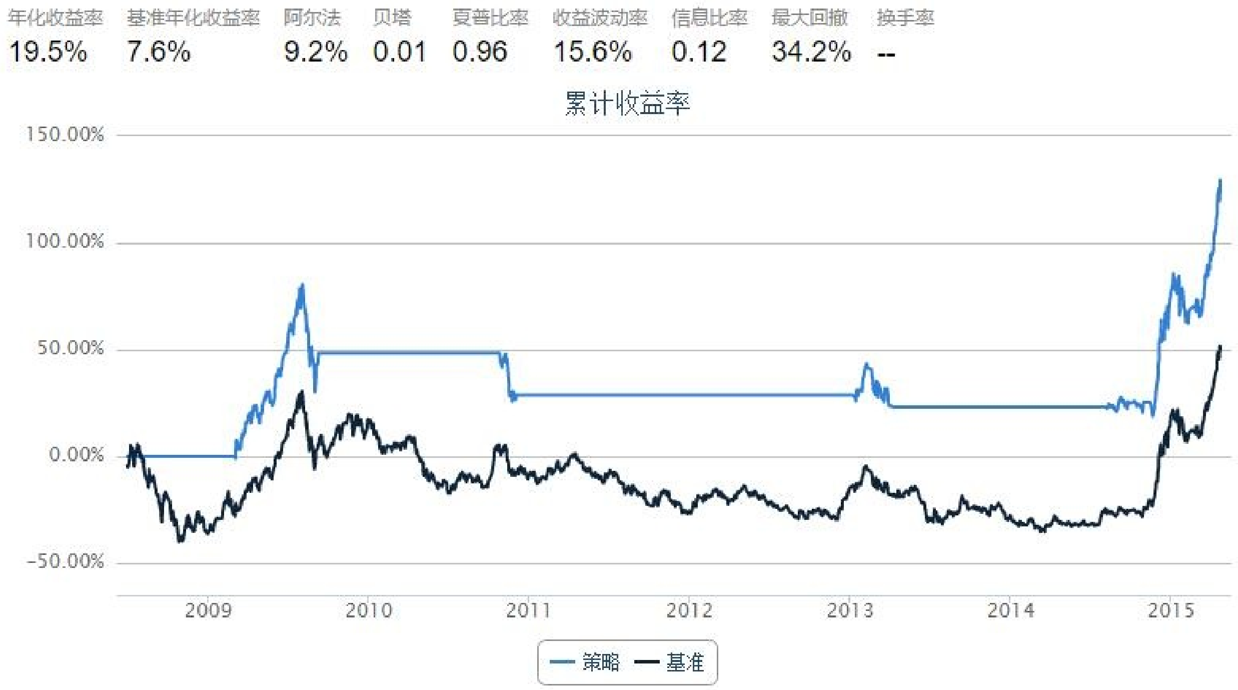
```
security[['Market', 'Strategy']].cumsum().apply(np.exp).plot(gri
d=False, figsize=(12,8))
sns.despine()
```



3 使用 `quartz` 实现策略

上面的部分介绍了从数据出发，在量化实验室内研究策略的流程。实际上我们可以直接用量化实验室内置的 `quartz` 框架。 `quartz` 框架为用户隐藏了数据获取、数据清晰以及回测逻辑。用户可以更加专注于策略逻辑的描述：

```
start = datetime(2008, 1, 1)                      # 回测起始时间
end  = datetime(2015, 4, 23)                      # 回测结束时间
benchmark = 'SH50'                                  # 策略参考标准
universe = ['510050.XSHG']    # 股票池
capital_base = 100000      # 起始资金
commission = Commission(0.0,0.0)

window_short = 20
window_long = 120
longest_history = window_long
SD = 0.05

def initialize(account):                          # 初始化虚拟账户状态
    account.fund = universe[0]
    account.SD = SD
    account.window_short = window_short
    account.window_long = window_long

def handle_data(account):                   # 每个交易日的买入卖出指令
    hist = account.get_history(longest_history)
    fund = account.fund
    short_mean = np.mean(hist[fund]['closePrice'][-account.windo
w_short:]) # 计算短均线值
    long_mean = np.mean(hist[fund]['closePrice'][-account.window
_long:])   #计算长均线值

    # 计算买入卖出信号
    flag = True if (short_mean - long_mean) > account.SD * long_
mean else False
    if flag:
        if account.position.secpos.get(fund, 0) == 0:
            # 空仓时全仓买入，买入股数为100的整数倍
            approximationAmount = int(account.cash / hist[fund][
'closePrice'][-1]/100.0) * 100
            order(fund, approximationAmount)
    else:
        # 卖出时，全仓清空
        if account.position.secpos.get(fund, 0) >= 0:
            order_to(fund, 0)
```

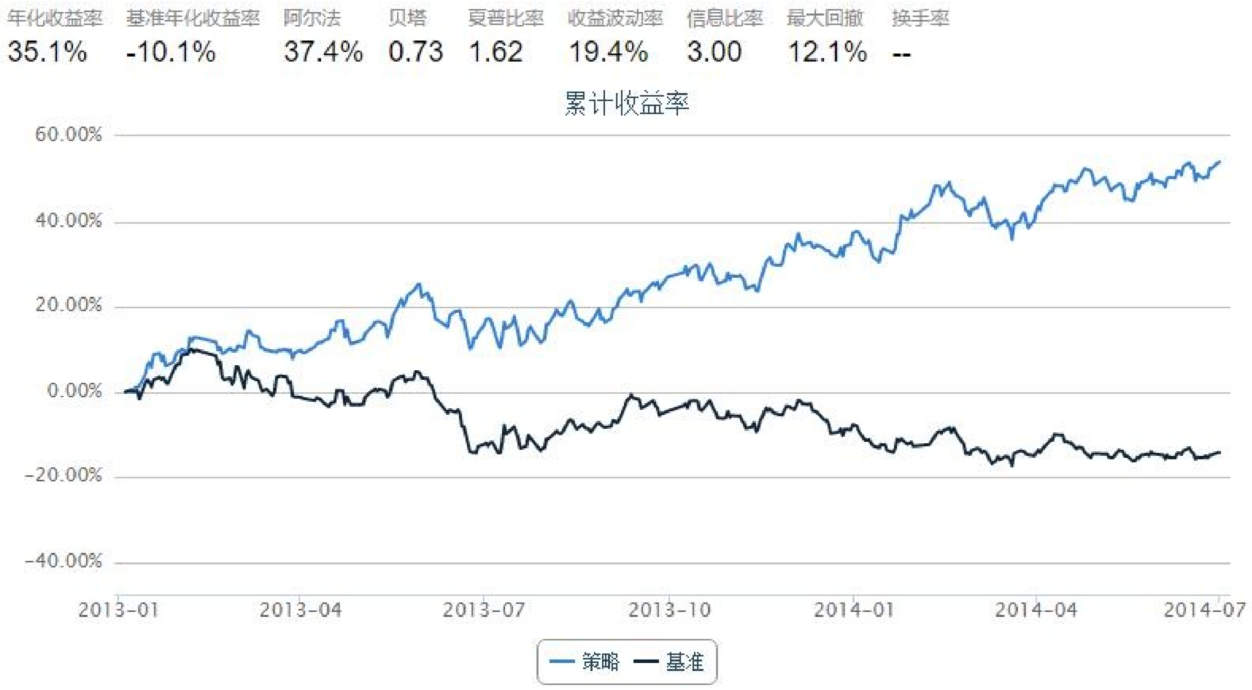


年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
19.5%	7.6%	9.2%	0.01	0.96	15.6%	0.12	34.2%	--

5日线10日线交易策略

> 来源：https://uqer.io/community/share/566526b3f9f06c6c8a91b45b

```
from CAL.PyCAL import Date
from CAL.PyCAL import Calendar
from CAL.PyCAL import BizDayConvention
start = '2013-01-01'                       # 回测起始时间
end = '2014-07-01'                         # 回测结束时间
benchmark = 'HS300'                        # 策略参考标准
universe = set_universe('HS300')  # 证券池，支持股票和基金
capital_base = 100000                      # 起始资金
freq = 'd'                                 # 策略类型，'d'表示日间策略使用日线回测，'m'表示日内策略使用分钟线回测
refresh_rate = 1                           # 调仓频率，表示执行handle_data的时间间隔，若freq = 'd'时间间隔的单位为交易日，若freq = 'm'时间间隔为分钟
commission = Commission(buycost=0.0003, sellcost=0.002, unit='perValue')

def initialize(account):                   # 初始化虚拟账户状态
    pass

def handle_data(account):                  # 每个交易日的买入卖出指令

    buylist=[]
    selist=[]
    dt = Date.fromDateTime(account.current_date)
    cal = Calendar('China.SSE')
    lastTDay = cal.advanceDate(dt,'-1B',BizDayConvention.Preceding)
    current_date=dt.strftime('%Y%m%d')
    last_date=lastTDay.strftime('%Y%m%d')

    getData_current_date=DataAPI.MktStockFactorsOneDayGet(tradeDate=current_date,secID=account.universe,field=['secID','MA5','MA10','MA20','NetProfitGrowRate'],pandas="1")
    getData_current_date.set_index('secID',inplace=True)
    getData_current_date=getData_current_date[getData_current_date.NetProfitGrowRate>=1.0].dropna()
    getData_current_date=getData_current_date.sort(columns='NetProfitGrowRate',ascending=False)
    getData_current_date=getData_current_date.head(20)
    #print  account.current_date,getData_current_date

    getData_last_date=DataAPI.MktStockFactorsOneDayGet(tradeDate=last_date,secID=list(getData_current_date.index),field=['secID','MA5','MA10','MA20','NetProfitGrowRate'],pandas="1")
    getData_last_date.set_index('secID',inplace=True)
    for stock in list(getData_current_date.index):
        if((getData_current_date['MA5'][stock]>getData_current_date['MA10'][stock])&(getData_last_date['MA5'][stock]<=getData_last_date['MA10'][stock])):
            buylist.append(stock)

    for stock in list(getData_current_date.index):
        if((getData_current_date['MA5'][stock]<=getData_current_date['MA10'][stock])&(getData_last_date['MA5'][stock]>getData_last_date['MA10'][stock])):
            selist.append(stock)


    for stock in selist:
        if(stock in account.valid_secpos):
            order_to(stock,0)
        else:
            pass

    for stock in buylist:
        if(stock in account.valid_secpos):
            pass
        else:
            if(len(buylist)>=5):
                order(stock,account.cash/account.referencePrice[stock]/len(buylist))
            else:
                order(stock,account.cash/account.referencePrice[stock]/5)

    print "日期：",account.current_date,",持仓：",account.valid_secpos
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
35.1%	-10.1%	37.4%	0.73	1.62	19.4%	3.00	12.1%	--

```
日期： 2013-01-04 00:00:00 ,持仓： {}
日期： 2013-01-07 00:00:00 ,持仓： {}
日期： 2013-01-08 00:00:00 ,持仓： {}
日期： 2013-01-09 00:00:00 ,持仓： {'002008.XSHE': 2489}
日期： 2013-01-10 00:00:00 ,持仓： {'002008.XSHE': 2489}
日期： 2013-01-11 00:00:00 ,持仓： {'002008.XSHE': 2489}
日期： 2013-01-14 00:00:00 ,持仓： {'002008.XSHE': 2489}
日期： 2013-01-15 00:00:00 ,持仓： {'002450.XSHE': 1546, '002008.X
SHE': 2489}
日期： 2013-01-16 00:00:00 ,持仓： {'002450.XSHE': 1546, '002008.X
SHE': 2489, '600108.XSHG': 1893}
日期： 2013-01-17 00:00:00 ,持仓： {'002450.XSHE': 1546, '002008.X
SHE': 2489, '600108.XSHG': 1893}
日期： 2013-01-18 00:00:00 ,持仓： {'002450.XSHE': 1546, '002008.X
SHE': 2489, '600108.XSHG': 1893}
日期： 2013-01-21 00:00:00 ,持仓： {'002450.XSHE': 1546, '002008.X
SHE': 2489, '600108.XSHG': 1893}
日期： 2013-01-22 00:00:00 ,持仓： {'002450.XSHE': 1546, '002008.X
SHE': 2489, '000046.XSHE': 2058, '600108.XSHG': 1893}
日期： 2013-01-23 00:00:00 ,持仓： {'002450.XSHE': 1546, '002008.X
SHE': 2489, '000046.XSHE': 2058, '600108.XSHG': 1893}
日期： 2013-01-24 00:00:00 ,持仓： {'002450.XSHE': 1546, '002008.X
SHE': 2489, '000046.XSHE': 2058, '600108.XSHG': 1893}
日期： 2013-01-25 00:00:00 ,持仓： {'002450.XSHE': 1546, '002008.X
SHE': 2489, '600108.XSHG': 1893, '000046.XSHE': 2058, '600383.XS
HG': 1248}
日期： 2013-01-28 00:00:00 ,持仓： {'002450.XSHE': 1546, '002008.X
SHE': 2489, '600108.XSHG': 1893}
日期： 2013-01-29 00:00:00 ,持仓： {'002008.XSHE': 2489, '600383.X
SHG': 1530}
日期： 2013-01-30 00:00:00 ,持仓： {'002450.XSHE': 1210, '002008.X
SHE': 2489, '600383.XSHG': 1530}
日期： 2013-01-31 00:00:00 ,持仓： {'002450.XSHE': 1210, '600383.X
```

```
SHG': 1530}
日期： 2013-02-01 00:00:00 ,持仓： {'002450.XSHE': 1210, '600011.X
SHG': 2988, '600383.XSHG': 1530}
日期： 2013-02-04 00:00:00 ,持仓： {'002450.XSHE': 1210, '002008.X
SHE': 1303, '600383.XSHG': 1530, '600011.XSHG': 2988, '000046.XS
HE': 2834}
日期： 2013-02-05 00:00:00 ,持仓： {'002450.XSHE': 1210, '002008.X
SHE': 1303, '600383.XSHG': 1530, '600011.XSHG': 2988, '000046.XS
HE': 2834}
日期： 2013-02-06 00:00:00 ,持仓： {'002450.XSHE': 1210, '000883.X
SHE': 2298, '600011.XSHG': 2988, '002008.XSHE': 1303, '000046.XS
HE': 2834, '600383.XSHG': 1530}
日期： 2013-02-07 00:00:00 ,持仓： {'002008.XSHE': 1303, '000883.X
SHE': 2298, '600383.XSHG': 1530, '600011.XSHG': 2988, '000046.XS
HE': 2834}
日期： 2013-02-08 00:00:00 ,持仓： {'002008.XSHE': 1303, '000883.X
SHE': 2298, '600383.XSHG': 1530, '600011.XSHG': 2988, '000046.XS
HE': 2834}
日期： 2013-02-18 00:00:00 ,持仓： {'002008.XSHE': 1303, '000883.X
SHE': 2298, '600383.XSHG': 1530, '600011.XSHG': 2988, '000046.XS
HE': 2834}
日期： 2013-02-19 00:00:00 ,持仓： {'002450.XSHE': 747, '000883.XS
HE': 2298, '600011.XSHG': 2988, '002008.XSHE': 1303, '000046.XSH
E': 2834, '000413.XSHE': 2260, '600383.XSHG': 1530}
日期： 2013-02-20 00:00:00 ,持仓： {'002450.XSHE': 747, '000883.XS
HE': 2298, '600011.XSHG': 2988, '002008.XSHE': 1303, '000046.XSH
E': 2834, '000413.XSHE': 2260}
日期： 2013-02-21 00:00:00 ,持仓： {'002450.XSHE': 747, '600108.XS
HG': 1123, '000883.XSHE': 2298, '002008.XSHE': 1303, '000046.XSH
E': 2834, '000413.XSHE': 2260}
日期： 2013-02-22 00:00:00 ,持仓： {'002450.XSHE': 747, '600108.XS
HG': 1123, '000413.XSHE': 2260, '002008.XSHE': 1303}
日期： 2013-02-25 00:00:00 ,持仓： {'002450.XSHE': 747, '600108.XS
HG': 1123, '002008.XSHE': 1303}
日期： 2013-02-26 00:00:00 ,持仓： {'002450.XSHE': 747, '600108.XS
HG': 1123, '002008.XSHE': 1303}
日期： 2013-02-27 00:00:00 ,持仓： {'002450.XSHE': 747, '600108.XS
HG': 1123}
日期： 2013-02-28 00:00:00 ,持仓： {'002450.XSHE': 747, '600108.XS
HG': 1123}
日期： 2013-03-01 00:00:00 ,持仓： {'002450.XSHE': 747, '600108.XS
HG': 1123}
日期： 2013-03-04 00:00:00 ,持仓： {'002450.XSHE': 747, '600108.XS
HG': 1123, '002008.XSHE': 1626}
日期： 2013-03-05 00:00:00 ,持仓： {'002450.XSHE': 747, '002008.XS
HE': 1626}
日期： 2013-03-06 00:00:00 ,持仓： {'002450.XSHE': 747, '002008.XS
HE': 1626, '600011.XSHG': 2825, '000413.XSHE': 4178}
日期： 2013-03-07 00:00:00 ,持仓： {'002450.XSHE': 747, '600108.XS
HG': 1380, '000883.XSHE': 3007, '600011.XSHG': 2825, '002008.XSH
E': 1626, '000413.XSHE': 4178}
日期： 2013-03-08 00:00:00 ,持仓： {'002450.XSHE': 747, '600108.XS
HG': 1380, '000883.XSHE': 3007, '600011.XSHG': 2825, '002008.XSH
```

```
E': 1626, '000413.XSHE': 4178}
日期： 2013-03-11 00:00:00 ,持仓： {'002450.XSHE': 747, '600108.XS
HG': 1380, '000883.XSHE': 3007, '600011.XSHG': 2825, '002008.XSH
E': 1626, '000413.XSHE': 4178}
日期： 2013-03-12 00:00:00 ,持仓： {'002450.XSHE': 747, '600108.XS
HG': 1380, '000883.XSHE': 3007, '600011.XSHG': 2825, '002008.XSH
E': 1626, '000413.XSHE': 4178}
日期： 2013-03-13 00:00:00 ,持仓： {'002450.XSHE': 747, '600108.XS
HG': 1380, '000883.XSHE': 3007, '600011.XSHG': 2825, '002008.XSH
E': 1626, '000413.XSHE': 4178}
日期： 2013-03-14 00:00:00 ,持仓： {'002450.XSHE': 747, '000883.XS
HE': 3007, '600011.XSHG': 2825, '000413.XSHE': 4178}
日期： 2013-03-15 00:00:00 ,持仓： {'002450.XSHE': 747, '000883.XS
HE': 3007}
日期： 2013-03-18 00:00:00 ,持仓： {'002450.XSHE': 747, '000883.XS
HE': 3007}
日期： 2013-03-19 00:00:00 ,持仓： {'002450.XSHE': 747, '000883.XS
HE': 3007}
日期： 2013-03-20 00:00:00 ,持仓： {'002450.XSHE': 747}
日期： 2013-03-21 00:00:00 ,持仓： {'002450.XSHE': 747, '600011.XS
HG': 3208}
日期： 2013-03-22 00:00:00 ,持仓： {'002450.XSHE': 747, '002008.XS
HE': 1327, '600383.XSHG': 2530, '600011.XSHG': 3208, '000046.XSH
E': 3572}
日期： 2013-03-25 00:00:00 ,持仓： {'002450.XSHE': 747, '002008.XS
HE': 1327, '600383.XSHG': 2530, '600011.XSHG': 3208, '000046.XSH
E': 3572}
日期： 2013-03-26 00:00:00 ,持仓： {'002450.XSHE': 747, '600108.XS
HG': 955, '600011.XSHG': 3208, '002008.XSHE': 1327, '000046.XSHE
': 3572, '600383.XSHG': 2530}
日期： 2013-03-27 00:00:00 ,持仓： {'002450.XSHE': 747, '600108.XS
HG': 955, '600011.XSHG': 3208, '002008.XSHE': 1327, '000046.XSHE
': 3572, '000413.XSHE': 1276, '600383.XSHG': 2530}
日期： 2013-03-28 00:00:00 ,持仓： {'002450.XSHE': 747, '600108.XS
HG': 955, '000883.XSHE': 1215, '600011.XSHG': 3208, '002008.XSHE
': 1327, '000046.XSHE': 3572, '000413.XSHE': 1276, '600383.XSHG'
: 2530}
日期： 2013-03-29 00:00:00 ,持仓： {'002450.XSHE': 747, '002008.XS
HE': 1327, '000883.XSHE': 1215, '600383.XSHG': 2530, '000413.XSH
E': 1276}
日期： 2013-04-01 00:00:00 ,持仓： {'002450.XSHE': 747, '002008.XS
HE': 1327, '600383.XSHG': 2530, '000413.XSHE': 1276}
日期： 2013-04-02 00:00:00 ,持仓： {'002450.XSHE': 747, '002008.XS
HE': 1327, '600383.XSHG': 2530, '000413.XSHE': 1276}
日期： 2013-04-03 00:00:00 ,持仓： {'002008.XSHE': 1327, '600383.X
SHG': 2530}
日期： 2013-04-08 00:00:00 ,持仓： {'002008.XSHE': 1327}
日期： 2013-04-09 00:00:00 ,持仓： {'002008.XSHE': 1327, '000046.X
SHE': 4207, '600383.XSHG': 2968}
日期： 2013-04-10 00:00:00 ,持仓： {'002008.XSHE': 1327, '000046.X
SHE': 4207, '600383.XSHG': 2968}
日期： 2013-04-11 00:00:00 ,持仓： {'002008.XSHE': 1327, '601991.X
SHG': 2759, '000046.XSHE': 4207, '600383.XSHG': 2968}
```

```
日期： 2013-04-12 00:00:00 ,持仓： {'600027.XSHG': 2288, '601991.X
SHG': 2759, '002008.XSHE': 1327, '000046.XSHE': 4207, '000413.XS
HE': 2315, '600383.XSHG': 2968}
日期： 2013-04-15 00:00:00 ,持仓： {'600027.XSHG': 2288, '601991.X
SHG': 2759, '002008.XSHE': 1327, '000046.XSHE': 4207, '000413.XS
HE': 2315, '600383.XSHG': 2968}
日期： 2013-04-16 00:00:00 ,持仓： {'600027.XSHG': 2288, '601991.X
SHG': 2759, '002008.XSHE': 1327, '000046.XSHE': 4207, '000413.XS
HE': 2315, '600383.XSHG': 2968}
日期： 2013-04-17 00:00:00 ,持仓： {'600583.XSHG': 881, '600027.XS
HG': 2288, '601991.XSHG': 2759, '002008.XSHE': 1327, '000046.XSH
E': 4207, '000413.XSHE': 2315, '000895.XSHE': 229, '600383.XSHG'
: 2968}
日期： 2013-04-18 00:00:00 ,持仓： {'002450.XSHE': 239, '600583.XS
HG': 881, '601991.XSHG': 2759, '002008.XSHE': 1327, '000046.XSHE
': 4207, '000413.XSHE': 2315, '000895.XSHE': 229, '600383.XSHG':
 2968}
日期： 2013-04-19 00:00:00 ,持仓： {'002450.XSHE': 239, '601991.XS
HG': 2759, '002008.XSHE': 1327, '000046.XSHE': 4207, '000413.XSH
E': 2315, '000895.XSHE': 229, '600383.XSHG': 2968}
日期： 2013-04-22 00:00:00 ,持仓： {'002450.XSHE': 239, '002008.XS
HE': 1327, '601991.XSHG': 2759, '000712.XSHE': 748, '000046.XSHE
': 4207, '000413.XSHE': 2315, '000895.XSHE': 229, '600383.XSHG':
 2968}
日期： 2013-04-23 00:00:00 ,持仓： {'002450.XSHE': 239, '600108.XS
HG': 673, '000712.XSHE': 748, '002008.XSHE': 1327, '000046.XSHE'
: 4207, '000413.XSHE': 2315, '000895.XSHE': 229, '002673.XSHE':
620, '600383.XSHG': 2968}
日期： 2013-04-24 00:00:00 ,持仓： {'002450.XSHE': 239, '600108.XS
HG': 673, '000712.XSHE': 748, '002008.XSHE': 1327, '000046.XSHE'
: 4207, '002673.XSHE': 620, '000895.XSHE': 229, '600383.XSHG': 2
968}
日期： 2013-04-25 00:00:00 ,持仓： {'002450.XSHE': 239, '600108.XS
HG': 673, '600011.XSHG': 1175, '000712.XSHE': 748, '002008.XSHE'
: 1327, '000046.XSHE': 4207, '002673.XSHE': 620, '000895.XSHE':
229, '600383.XSHG': 2968}
日期： 2013-04-26 00:00:00 ,持仓： {'600108.XSHG': 673, '000883.XS
HE': 1727, '600011.XSHG': 1175, '002008.XSHE': 1327, '000712.XSH
E': 748, '300251.XSHE': 541, '000046.XSHE': 4207, '002673.XSHE':
 620, '000895.XSHE': 229, '600383.XSHG': 2968, '600886.XSHG': 15
11}
日期： 2013-05-02 00:00:00 ,持仓： {'600108.XSHG': 673, '600011.XS
HG': 1175, '002008.XSHE': 1327, '000712.XSHE': 748, '300251.XSHE
': 541, '000046.XSHE': 4207, '002673.XSHE': 620, '000895.XSHE':
229, '600383.XSHG': 2968, '600886.XSHG': 1511}
日期： 2013-05-03 00:00:00 ,持仓： {'600108.XSHG': 673, '600011.XS
HG': 1175, '002008.XSHE': 1327, '600795.XSHG': 1589, '000712.XSH
E': 748, '300251.XSHE': 541, '000046.XSHE': 4207, '002456.XSHE':
 169, '000895.XSHE': 229, '600383.XSHG': 2968, '600886.XSHG': 15
11}
日期： 2013-05-06 00:00:00 ,持仓： {'600108.XSHG': 673, '600583.XS
HG': 511, '002008.XSHE': 1327, '600027.XSHG': 848, '600369.XSHG'
: 784, '600795.XSHG': 1589, '000712.XSHE': 748, '300251.XSHE': 5
```

```
41, '000046.XSHE': 4207, '002456.XSHE': 169, '600886.XSHG': 1511
, '000895.XSHE': 229, '600383.XSHG': 2968, '600011.XSHG': 1175}
日期： 2013-05-07 00:00:00 ,持仓： {'600583.XSHG': 511, '002008.XS
HE': 1327, '600027.XSHG': 848, '600369.XSHG': 784, '600795.XSHG'
: 1589, '000712.XSHE': 748, '300251.XSHE': 541, '000046.XSHE': 4
207, '002456.XSHE': 169, '002673.XSHE': 181, '000895.XSHE': 229,
 '600383.XSHG': 2968, '600886.XSHG': 1511, '600011.XSHG': 1175}
日期： 2013-05-08 00:00:00 ,持仓： {'601991.XSHG': 452, '600583.XS
HG': 511, '002008.XSHE': 1327, '600027.XSHG': 848, '600369.XSHG'
: 784, '600795.XSHG': 1589, '000712.XSHE': 748, '000625.XSHE': 1
71, '000046.XSHE': 4207, '000895.XSHE': 229, '300251.XSHE': 541,
 '002673.XSHE': 181, '600383.XSHG': 2968, '600886.XSHG': 1511, '
600011.XSHG': 1175}
日期： 2013-05-09 00:00:00 ,持仓： {'601991.XSHG': 452, '600583.XS
HG': 511, '002008.XSHE': 1327, '600027.XSHG': 848, '600369.XSHG'
: 784, '600795.XSHG': 1589, '000712.XSHE': 748, '000625.XSHE': 1
71, '000046.XSHE': 4207, '000895.XSHE': 229, '300251.XSHE': 541,
 '002673.XSHE': 181, '600383.XSHG': 2968, '000539.XSHE': 466, '6
00886.XSHG': 1511, '600011.XSHG': 1175}
日期： 2013-05-10 00:00:00 ,持仓： {'002450.XSHE': 93, '601991.XSH
G': 452, '600583.XSHG': 511, '000712.XSHE': 748, '600027.XSHG':
848, '000625.XSHE': 171, '600369.XSHG': 784, '600795.XSHG': 1589
, '600157.XSHG': 482, '002008.XSHE': 1327, '300251.XSHE': 541, '
000046.XSHE': 4207, '600316.XSHG': 93, '000895.XSHE': 229, '0026
73.XSHE': 181, '600383.XSHG': 2968, '000539.XSHE': 466, '600886.
XSHG': 1511, '600011.XSHG': 1175}
日期： 2013-05-13 00:00:00 ,持仓： {'000046.XSHE': 4207, '601991.X
SHG': 452, '600583.XSHG': 511, '000712.XSHE': 748, '600027.XSHG'
: 848, '000625.XSHE': 171, '600369.XSHG': 784, '600795.XSHG': 15
89, '600157.XSHG': 482, '002008.XSHE': 1327, '300251.XSHE': 541,
 '600316.XSHG': 93, '002456.XSHE': 24, '000413.XSHE': 162, '0008
95.XSHE': 229, '002673.XSHE': 181, '600383.XSHG': 2968, '000539.
XSHE': 466, '600886.XSHG': 1511, '600011.XSHG': 1175}
日期： 2013-05-14 00:00:00 ,持仓： {'000046.XSHE': 4207, '600108.X
SHG': 105, '601991.XSHG': 452, '600583.XSHG': 511, '000712.XSHE'
: 748, '600027.XSHG': 848, '000625.XSHE': 171, '600369.XSHG': 78
4, '600795.XSHG': 1589, '600157.XSHG': 482, '002008.XSHE': 1327,
 '300251.XSHE': 541, '600316.XSHG': 93, '002456.XSHE': 24, '0004
13.XSHE': 162, '000895.XSHE': 229, '002673.XSHE': 181, '600383.X
SHG': 2968, '000539.XSHE': 466, '600886.XSHG': 1511, '600011.XSH
G': 1175}
日期： 2013-05-15 00:00:00 ,持仓： {'000046.XSHE': 4207, '600108.X
SHG': 105, '600583.XSHG': 511, '600027.XSHG': 848, '601991.XSHG'
: 452, '600795.XSHG': 1589, '600157.XSHG': 482, '002008.XSHE': 1
327, '300251.XSHE': 541, '600316.XSHG': 93, '002456.XSHE': 24, '
000413.XSHE': 162, '000895.XSHE': 229, '002673.XSHE': 181, '6003
83.XSHG': 2968, '000539.XSHE': 466, '600886.XSHG': 1511, '600011
.XSHG': 1175}
日期： 2013-05-16 00:00:00 ,持仓： {'600108.XSHG': 105, '600011.XS
HG': 1175, '601991.XSHG': 452, '600795.XSHG': 1589, '002008.XSHE
': 1327, '300251.XSHE': 541, '600316.XSHG': 93, '002456.XSHE': 2
4, '000413.XSHE': 162, '000895.XSHE': 229, '002673.XSHE': 181, '
600383.XSHG': 2968, '000539.XSHE': 466, '600886.XSHG': 1511, '60
```

```
0583.XSHG': 511}
日期： 2013-05-17 00:00:00 ,持仓： {'600108.XSHG': 105, '600011.XS
HG': 1175, '601991.XSHG': 452, '600795.XSHG': 1589, '002008.XSHE
': 1327, '300251.XSHE': 541, '600316.XSHG': 93, '002456.XSHE': 2
4, '000413.XSHE': 162, '000895.XSHE': 229, '002673.XSHE': 181, '
600383.XSHG': 2968, '000539.XSHE': 466, '600886.XSHG': 1511, '60
0583.XSHG': 511}
日期： 2013-05-20 00:00:00 ,持仓： {'002450.XSHE': 441, '600108.XS
HG': 105, '600011.XSHG': 1175, '000712.XSHE': 945, '601991.XSHG'
: 452, '600795.XSHG': 1589, '002008.XSHE': 1327, '300251.XSHE':
541, '600316.XSHG': 93, '002456.XSHE': 24, '000413.XSHE': 162, '
000895.XSHE': 229, '002673.XSHE': 181, '600383.XSHG': 2968, '600
886.XSHG': 1511, '600583.XSHG': 511}
日期： 2013-05-21 00:00:00 ,持仓： {'002450.XSHE': 441, '600108.XS
HG': 105, '601991.XSHG': 452, '600583.XSHG': 511, '000712.XSHE':
 945, '600369.XSHG': 1143, '600795.XSHG': 1589, '002008.XSHE': 1
327, '300251.XSHE': 541, '000046.XSHE': 956, '002456.XSHE': 24,
'600316.XSHG': 93, '000895.XSHE': 229, '002673.XSHE': 181, '6003
83.XSHG': 2968, '600886.XSHG': 1511, '601901.XSHG': 663, '600011
.XSHG': 1175}
日期： 2013-05-22 00:00:00 ,持仓： {'600583.XSHG': 511, '002456.XS
HE': 24, '000413.XSHE': 541, '000895.XSHE': 229, '000539.XSHE':
484, '002450.XSHE': 441, '600011.XSHG': 1175, '000783.XSHE': 440
, '600369.XSHG': 1143, '002008.XSHE': 1327, '300251.XSHE': 541,
'600316.XSHG': 93, '000046.XSHE': 956, '002673.XSHE': 181, '6003
83.XSHG': 2968, '600108.XSHG': 105, '601991.XSHG': 452, '600795.
XSHG': 1589, '600157.XSHG': 651, '000712.XSHE': 945, '600886.XSH
G': 1511, '601901.XSHG': 663}
日期： 2013-05-23 00:00:00 ,持仓： {'002450.XSHE': 441, '600108.XS
HG': 105, '601991.XSHG': 452, '600583.XSHG': 511, '000712.XSHE':
 945, '000783.XSHE': 440, '600369.XSHG': 1143, '600795.XSHG': 15
89, '000046.XSHE': 956, '002008.XSHE': 1327, '300251.XSHE': 541,
 '600316.XSHG': 93, '002456.XSHE': 24, '000413.XSHE': 541, '6001
57.XSHG': 651, '000895.XSHE': 229, '002673.XSHE': 181, '600383.X
SHG': 2968, '000539.XSHE': 484, '601901.XSHG': 663, '600011.XSHG
': 1175}
日期： 2013-05-24 00:00:00 ,持仓： {'002450.XSHE': 441, '600108.XS
HG': 105, '600583.XSHG': 511, '000712.XSHE': 945, '000783.XSHE':
 440, '600369.XSHG': 1143, '600795.XSHG': 1589, '600157.XSHG': 6
51, '002008.XSHE': 1327, '300251.XSHE': 541, '600316.XSHG': 93,
'002456.XSHE': 24, '000413.XSHE': 541, '000895.XSHE': 229, '0026
73.XSHE': 181, '600383.XSHG': 2968, '000539.XSHE': 484, '601901.
XSHG': 663, '000046.XSHE': 956}
日期： 2013-05-27 00:00:00 ,持仓： {'002450.XSHE': 441, '600108.XS
HG': 105, '600583.XSHG': 511, '000712.XSHE': 945, '000783.XSHE':
 440, '000625.XSHE': 317, '600369.XSHG': 1143, '600795.XSHG': 15
89, '600157.XSHG': 651, '002008.XSHE': 1327, '300251.XSHE': 541,
 '600316.XSHG': 93, '002456.XSHE': 24, '000413.XSHE': 541, '0008
95.XSHE': 229, '002673.XSHE': 181, '600383.XSHG': 2968, '000539.
XSHE': 484, '601901.XSHG': 663, '000046.XSHE': 956}
日期： 2013-05-28 00:00:00 ,持仓： {'002450.XSHE': 441, '600108.XS
HG': 105, '600583.XSHG': 511, '000712.XSHE': 945, '000783.XSHE':
 440, '000625.XSHE': 317, '600369.XSHG': 1143, '600795.XSHG': 15
```

```
89, '600157.XSHG': 651, '002008.XSHE': 1327, '300251.XSHE': 541,
 '600316.XSHG': 93, '002456.XSHE': 24, '000413.XSHE': 541, '0008
95.XSHE': 229, '002673.XSHE': 181, '600383.XSHG': 2968, '000046.
XSHE': 956}
日期： 2013-05-29 00:00:00 ,持仓： {'002450.XSHE': 441, '600108.XS
HG': 105, '600583.XSHG': 511, '000712.XSHE': 945, '000783.XSHE':
 440, '000625.XSHE': 317, '600369.XSHG': 1143, '600795.XSHG': 15
89, '600157.XSHG': 651, '002008.XSHE': 1327, '300251.XSHE': 541,
 '600316.XSHG': 93, '000413.XSHE': 541, '000895.XSHE': 229, '002
673.XSHE': 181, '600383.XSHG': 2968, '601901.XSHG': 551, '000046
.XSHE': 956}
日期： 2013-05-30 00:00:00 ,持仓： {'002450.XSHE': 441, '600108.XS
HG': 105, '600583.XSHG': 511, '000712.XSHE': 945, '000783.XSHE':
 440, '000625.XSHE': 317, '600369.XSHG': 1143, '600795.XSHG': 15
89, '600157.XSHG': 651, '002008.XSHE': 1327, '300251.XSHE': 541,
 '600316.XSHG': 93, '000413.XSHE': 541, '000895.XSHE': 229, '002
673.XSHE': 181, '600383.XSHG': 2968, '601901.XSHG': 551, '000046
.XSHE': 956}
日期： 2013-05-31 00:00:00 ,持仓： {'002450.XSHE': 441, '600108.XS
HG': 105, '600583.XSHG': 511, '000712.XSHE': 945, '000783.XSHE':
 440, '600369.XSHG': 1143, '600795.XSHG': 1589, '002008.XSHE': 1
327, '000625.XSHE': 317, '600316.XSHG': 93, '000413.XSHE': 541,
'000895.XSHE': 229, '002673.XSHE': 181, '600383.XSHG': 2968, '60
1901.XSHG': 551, '000046.XSHE': 956}
日期： 2013-06-03 00:00:00 ,持仓： {'002450.XSHE': 441, '600108.XS
HG': 105, '601991.XSHG': 1109, '600583.XSHG': 511, '000712.XSHE'
: 945, '000783.XSHE': 440, '600369.XSHG': 1143, '600795.XSHG': 1
589, '600157.XSHG': 1552, '002008.XSHE': 1327, '000625.XSHE': 31
7, '600316.XSHG': 93, '000413.XSHE': 541, '000895.XSHE': 229, '0
02673.XSHE': 181, '600383.XSHG': 2968, '601901.XSHG': 551, '0000
46.XSHE': 956}
日期： 2013-06-04 00:00:00 ,持仓： {'600108.XSHG': 105, '601991.XS
HG': 1109, '600583.XSHG': 511, '000712.XSHE': 945, '000783.XSHE'
: 440, '600369.XSHG': 1143, '600795.XSHG': 1589, '600157.XSHG':
1552, '002008.XSHE': 1327, '600316.XSHG': 93, '000413.XSHE': 541
, '000895.XSHE': 229, '002673.XSHE': 181, '600383.XSHG': 2968, '
000539.XSHE': 730, '601901.XSHG': 551, '000046.XSHE': 956}
日期： 2013-06-05 00:00:00 ,持仓： {'600108.XSHG': 105, '601991.XS
HG': 1109, '600583.XSHG': 511, '000712.XSHE': 945, '000783.XSHE'
: 440, '600369.XSHG': 1143, '600795.XSHG': 1589, '600157.XSHG':
1552, '002008.XSHE': 1327, '600316.XSHG': 93, '000413.XSHE': 541
, '000895.XSHE': 229, '600383.XSHG': 2968, '000539.XSHE': 730, '
600886.XSHG': 1210, '601901.XSHG': 551, '000046.XSHE': 956}
日期： 2013-06-06 00:00:00 ,持仓： {'000712.XSHE': 945, '600369.XS
HG': 1143, '600795.XSHG': 1589, '600157.XSHG': 1552, '002008.XSH
E': 1327, '000413.XSHE': 541, '000046.XSHE': 956, '600316.XSHG':
 93, '000895.XSHE': 229, '000539.XSHE': 730, '601901.XSHG': 551,
 '600383.XSHG': 2968}
日期： 2013-06-07 00:00:00 ,持仓： {'000712.XSHE': 945, '600795.XS
HG': 1589, '600157.XSHG': 1552, '002008.XSHE': 1327, '000046.XSH
E': 956, '000413.XSHE': 541, '000895.XSHE': 229, '000539.XSHE':
730, '600383.XSHG': 2968}
日期： 2013-06-13 00:00:00 ,持仓： {'000712.XSHE': 945, '600795.XS
```

```
HG': 1589, '600157.XSHG': 1552, '002008.XSHE': 1327, '000046.XSH
E': 956, '000413.XSHE': 541, '000895.XSHE': 229, '000539.XSHE':
730, '600383.XSHG': 2968}
日期： 2013-06-14 00:00:00 ,持仓： {'600157.XSHG': 1552, '000895.X
SHE': 229, '600383.XSHG': 2968, '600795.XSHG': 1589, '002008.XSH
E': 1327}
日期： 2013-06-17 00:00:00 ,持仓： {'600795.XSHG': 1589, '600157.X
SHG': 1552, '002008.XSHE': 1327, '300251.XSHE': 1065, '002456.XS
HE': 541, '000413.XSHE': 3121, '000895.XSHE': 229, '600383.XSHG'
: 2968}
日期： 2013-06-18 00:00:00 ,持仓： {'600795.XSHG': 1589, '600157.X
SHG': 1552, '002008.XSHE': 1327, '300251.XSHE': 1065, '002456.XS
HE': 541, '000413.XSHE': 3121, '000895.XSHE': 229, '600383.XSHG'
: 2968}
日期： 2013-06-19 00:00:00 ,持仓： {'601991.XSHG': 1160, '600795.X
SHG': 1589, '600157.XSHG': 1552, '002008.XSHE': 1327, '300251.XS
HE': 1065, '002456.XSHE': 541, '000413.XSHE': 3121, '000895.XSHE
': 229, '600383.XSHG': 2968}
日期： 2013-06-20 00:00:00 ,持仓： {'002450.XSHE': 232, '601991.XS
HG': 1160, '600795.XSHG': 1589, '600157.XSHG': 1552, '002008.XSH
E': 1327, '300251.XSHE': 1065, '000413.XSHE': 3121, '000895.XSHE
': 229, '600383.XSHG': 2968}
日期： 2013-06-21 00:00:00 ,持仓： {'002450.XSHE': 232, '601991.XS
HG': 1160, '600795.XSHG': 1589, '600157.XSHG': 1552, '002008.XSH
E': 1327, '300251.XSHE': 1065, '000413.XSHE': 3121, '000895.XSHE
': 229, '600383.XSHG': 2968}
日期： 2013-06-24 00:00:00 ,持仓： {'002450.XSHE': 232, '601991.XS
HG': 1160, '600795.XSHG': 1589, '600157.XSHG': 1552, '002008.XSH
E': 1327, '300251.XSHE': 1065, '000413.XSHE': 3121, '000895.XSHE
': 229, '600383.XSHG': 2968}
日期： 2013-06-25 00:00:00 ,持仓： {'002450.XSHE': 232, '601991.XS
HG': 1160, '600795.XSHG': 1589, '600157.XSHG': 1552, '002008.XSH
E': 1327, '300251.XSHE': 1065, '000413.XSHE': 3121, '000895.XSHE
': 229, '600383.XSHG': 2968}
日期： 2013-06-26 00:00:00 ,持仓： {'002450.XSHE': 232, '601991.XS
HG': 1160, '600795.XSHG': 1589, '600157.XSHG': 1552, '002008.XSH
E': 1327, '300251.XSHE': 1065, '000413.XSHE': 3121, '000895.XSHE
': 229, '600383.XSHG': 2968}
日期： 2013-06-27 00:00:00 ,持仓： {'601991.XSHG': 1160, '600795.X
SHG': 1589, '600157.XSHG': 1552, '002008.XSHE': 1327, '300251.XS
HE': 1065, '000895.XSHE': 229, '600383.XSHG': 2968}
日期： 2013-06-28 00:00:00 ,持仓： {'601991.XSHG': 1160, '600795.X
SHG': 1589, '600157.XSHG': 1552, '002008.XSHE': 1327, '000895.XS
HE': 229, '600383.XSHG': 2968}
日期： 2013-07-01 00:00:00 ,持仓： {'601991.XSHG': 1160, '600795.X
SHG': 1589, '600157.XSHG': 1552, '002008.XSHE': 1327, '000895.XS
HE': 229, '600383.XSHG': 2968}
日期： 2013-07-02 00:00:00 ,持仓： {'002450.XSHE': 664, '601991.XS
HG': 1160, '600795.XSHG': 1589, '600157.XSHG': 1552, '002008.XSH
E': 1327, '000625.XSHE': 1355, '002456.XSHE': 521, '000895.XSHE'
: 229, '600383.XSHG': 2968}
日期： 2013-07-03 00:00:00 ,持仓： {'002450.XSHE': 664, '000712.XS
HE': 541, '601991.XSHG': 1160, '600795.XSHG': 1589, '600157.XSHG
```

```
': 1552, '002008.XSHE': 1327, '000625.XSHE': 1355, '600886.XSHG'
: 1438, '002456.XSHE': 521, '000895.XSHE': 229, '600383.XSHG': 2
968}
日期： 2013-07-04 00:00:00 ,持仓： {'002450.XSHE': 664, '600108.XS
HG': 431, '600011.XSHG': 622, '000712.XSHE': 541, '601991.XSHG':
 1160, '600795.XSHG': 1589, '600157.XSHG': 1552, '002008.XSHE':
1327, '600886.XSHG': 1438, '002456.XSHE': 521, '000895.XSHE': 22
9, '600383.XSHG': 2968}
日期： 2013-07-05 00:00:00 ,持仓： {'002450.XSHE': 664, '600108.XS
HG': 431, '600011.XSHG': 622, '000712.XSHE': 541, '601991.XSHG':
 1160, '600795.XSHG': 1589, '600157.XSHG': 1552, '002008.XSHE':
1327, '300251.XSHE': 291, '600886.XSHG': 1438, '002456.XSHE': 52
1, '600316.XSHG': 241, '000895.XSHE': 229, '600383.XSHG': 2968}
日期： 2013-07-08 00:00:00 ,持仓： {'002450.XSHE': 664, '600108.XS
HG': 431, '601991.XSHG': 1160, '600011.XSHG': 622, '000712.XSHE'
: 541, '000783.XSHE': 676, '600369.XSHG': 657, '600795.XSHG': 15
89, '600157.XSHG': 1552, '002008.XSHE': 1327, '300251.XSHE': 291
, '600316.XSHG': 241, '002456.XSHE': 521, '000413.XSHE': 567, '0
00895.XSHE': 229, '002673.XSHE': 435, '600383.XSHG': 2968, '6008
86.XSHG': 1438, '600583.XSHG': 371.0}
日期： 2013-07-09 00:00:00 ,持仓： {'002450.XSHE': 664, '600108.XS
HG': 431, '601991.XSHG': 1160, '600011.XSHG': 622, '000712.XSHE'
: 541, '600369.XSHG': 657, '600157.XSHG': 1552, '002008.XSHE': 1
327, '300251.XSHE': 291, '600316.XSHG': 241, '002456.XSHE': 521,
 '000413.XSHE': 567, '000895.XSHE': 229, '002673.XSHE': 435, '60
0383.XSHG': 2968, '600886.XSHG': 1438, '600583.XSHG': 371.0}
日期： 2013-07-10 00:00:00 ,持仓： {'002450.XSHE': 664, '600108.XS
HG': 431, '600583.XSHG': 371.0, '000712.XSHE': 541, '601991.XSHG
': 1160, '600157.XSHG': 1552, '002008.XSHE': 1327, '300251.XSHE'
: 291, '600316.XSHG': 241, '002456.XSHE': 521, '000413.XSHE': 56
7, '000895.XSHE': 229, '600886.XSHG': 1438, '600383.XSHG': 2968}
日期： 2013-07-11 00:00:00 ,持仓： {'002450.XSHE': 664, '600108.XS
HG': 431, '600583.XSHG': 371.0, '600369.XSHG': 729, '600157.XSHG
': 1552, '002008.XSHE': 1327, '300251.XSHE': 291, '600316.XSHG':
 241, '002456.XSHE': 521, '000413.XSHE': 567, '000895.XSHE': 229
, '600886.XSHG': 1438, '600383.XSHG': 2968}
日期： 2013-07-12 00:00:00 ,持仓： {'002450.XSHE': 664, '600583.XS
HG': 371.0, '000783.XSHE': 1123, '600369.XSHG': 729, '600157.XSH
G': 1552, '002008.XSHE': 1327, '000625.XSHE': 461, '600886.XSHG'
: 1438, '000895.XSHE': 229, '600383.XSHG': 2968, '600011.XSHG':
863}
日期： 2013-07-15 00:00:00 ,持仓： {'002450.XSHE': 664, '600583.XS
HG': 371.0, '000783.XSHE': 1123, '600369.XSHG': 729, '600157.XSH
G': 1552, '002008.XSHE': 1327, '000625.XSHE': 461, '600886.XSHG'
: 1438, '000895.XSHE': 229, '600383.XSHG': 2968, '000539.XSHE':
1798, '600011.XSHG': 863}
日期： 2013-07-16 00:00:00 ,持仓： {'002450.XSHE': 664, '600108.XS
HG': 507, '600583.XSHG': 371.0, '000712.XSHE': 374, '000783.XSHE
': 1123, '600027.XSHG': 1252, '000625.XSHE': 461, '600369.XSHG':
 729, '600795.XSHG': 1535.0, '600157.XSHG': 1552, '002008.XSHE':
 1327, '300251.XSHE': 242, '600886.XSHG': 1438, '000413.XSHE': 7
81, '000895.XSHE': 229, '002673.XSHE': 583, '600383.XSHG': 2968,
 '000539.XSHE': 1798, '601901.XSHG': 608, '600011.XSHG': 863}
```

```
日期： 2013-07-17 00:00:00 ,持仓： {'002450.XSHE': 664, '600108.XS
HG': 507, '600583.XSHG': 371.0, '000712.XSHE': 374, '000783.XSHE
': 1123, '600027.XSHG': 1252, '000625.XSHE': 461, '600369.XSHG':
 729, '600795.XSHG': 1535.0, '600157.XSHG': 1552, '002008.XSHE':
 1327, '300251.XSHE': 242, '600886.XSHG': 1438, '000413.XSHE': 7
81, '000895.XSHE': 229, '002673.XSHE': 583, '600383.XSHG': 2968,
 '000539.XSHE': 1798, '601901.XSHG': 608, '600011.XSHG': 863}
日期： 2013-07-18 00:00:00 ,持仓： {'002450.XSHE': 664, '600108.XS
HG': 507, '600583.XSHG': 371.0, '000712.XSHE': 374, '000783.XSHE
': 1123, '600027.XSHG': 1252, '000625.XSHE': 461, '600369.XSHG':
 729, '600795.XSHG': 1535.0, '600157.XSHG': 1552, '002008.XSHE':
 1327, '300251.XSHE': 242, '600886.XSHG': 1438, '000413.XSHE': 7
81, '000895.XSHE': 229, '002673.XSHE': 583, '600383.XSHG': 2968,
 '000539.XSHE': 1798, '601901.XSHG': 608, '600011.XSHG': 863}
日期： 2013-07-19 00:00:00 ,持仓： {'002450.XSHE': 664, '600108.XS
HG': 507, '600583.XSHG': 371.0, '000712.XSHE': 374, '000783.XSHE
': 1123, '600027.XSHG': 1252, '000625.XSHE': 461, '600369.XSHG':
 729, '600795.XSHG': 1535.0, '600157.XSHG': 1552, '002008.XSHE':
 1327, '300251.XSHE': 242, '600886.XSHG': 1438, '000413.XSHE': 7
81, '000895.XSHE': 229, '002673.XSHE': 583, '600383.XSHG': 2968,
 '000539.XSHE': 1798, '601901.XSHG': 608, '600011.XSHG': 863}
日期： 2013-07-22 00:00:00 ,持仓： {'002450.XSHE': 664, '600108.XS
HG': 507, '600583.XSHG': 371.0, '000712.XSHE': 374, '000783.XSHE
': 1123, '600027.XSHG': 1252, '600369.XSHG': 729, '600795.XSHG':
 1535.0, '600157.XSHG': 1552, '002008.XSHE': 1327, '300251.XSHE'
: 242, '000413.XSHE': 781, '000895.XSHE': 229, '002673.XSHE': 58
3, '600383.XSHG': 2968, '000539.XSHE': 1798, '601901.XSHG': 608,
 '600011.XSHG': 863}
日期： 2013-07-23 00:00:00 ,持仓： {'002450.XSHE': 664, '600583.XS
HG': 371.0, '000712.XSHE': 374, '000783.XSHE': 1123, '600369.XSH
G': 729, '600157.XSHG': 1552, '002008.XSHE': 1327, '300251.XSHE'
: 242, '000413.XSHE': 781, '000895.XSHE': 229, '002673.XSHE': 58
3, '601901.XSHG': 608, '600383.XSHG': 2968}
日期： 2013-07-24 00:00:00 ,持仓： {'002450.XSHE': 664, '000413.XS
HE': 781, '002008.XSHE': 1327, '300251.XSHE': 242, '002673.XSHE'
: 583, '000895.XSHE': 229, '600383.XSHG': 2968}
日期： 2013-07-25 00:00:00 ,持仓： {'002450.XSHE': 664, '002008.XS
HE': 1327, '300251.XSHE': 242, '000413.XSHE': 781, '000895.XSHE'
: 229, '600383.XSHG': 2968}
日期： 2013-07-26 00:00:00 ,持仓： {'002450.XSHE': 664, '600583.XS
HG': 1459, '002008.XSHE': 1327, '300251.XSHE': 242, '000413.XSHE
': 781, '000895.XSHE': 229, '002673.XSHE': 1716, '600383.XSHG':
2968}
日期： 2013-07-29 00:00:00 ,持仓： {'002450.XSHE': 664, '600108.XS
HG': 920, '600583.XSHG': 1459, '000712.XSHE': 678, '002008.XSHE'
: 1327, '300251.XSHE': 242, '000413.XSHE': 781, '000895.XSHE': 2
29, '002673.XSHE': 1716, '601901.XSHG': 1041, '600383.XSHG': 296
8}
日期： 2013-07-30 00:00:00 ,持仓： {'002450.XSHE': 664, '600108.XS
HG': 920, '600583.XSHG': 1459, '000712.XSHE': 678, '600369.XSHG'
: 629, '002008.XSHE': 1327, '300251.XSHE': 242, '000413.XSHE': 7
81, '000895.XSHE': 229, '002673.XSHE': 1716, '601901.XSHG': 1041
, '600383.XSHG': 2968}
```

```
日期： 2013-07-31 00:00:00 ,持仓： {'002450.XSHE': 664, '600108.XS
HG': 920, '600583.XSHG': 1459, '000712.XSHE': 678, '600369.XSHG'
: 629, '002008.XSHE': 1327, '300251.XSHE': 242, '000413.XSHE': 7
81, '000895.XSHE': 229, '002673.XSHE': 1716, '601901.XSHG': 1041
, '600383.XSHG': 2968}
日期： 2013-08-01 00:00:00 ,持仓： {'002450.XSHE': 664, '600108.XS
HG': 920, '600583.XSHG': 1459, '000712.XSHE': 678, '600369.XSHG'
: 629, '002008.XSHE': 1327, '000413.XSHE': 781, '000895.XSHE': 2
29, '002673.XSHE': 1716, '600383.XSHG': 2968, '601901.XSHG': 104
1, '600011.XSHG': 421}
日期： 2013-08-02 00:00:00 ,持仓： {'002450.XSHE': 664, '601991.XS
HG': 489, '600583.XSHG': 1459, '000712.XSHE': 678, '600369.XSHG'
: 629, '002008.XSHE': 1327, '000413.XSHE': 781, '000895.XSHE': 2
29, '002673.XSHE': 1716, '600383.XSHG': 2968, '601901.XSHG': 104
1, '600011.XSHG': 421}
日期： 2013-08-05 00:00:00 ,持仓： {'002450.XSHE': 664, '601991.XS
HG': 489, '600011.XSHG': 421, '000712.XSHE': 678, '600369.XSHG':
 629, '002008.XSHE': 1327, '000413.XSHE': 781, '000895.XSHE': 22
9, '002673.XSHE': 1716, '600383.XSHG': 2968}
日期： 2013-08-06 00:00:00 ,持仓： {'002450.XSHE': 664, '600108.XS
HG': 530, '601991.XSHG': 489, '600583.XSHG': 492, '000712.XSHE':
 678, '000783.XSHE': 915, '600027.XSHG': 1342, '600369.XSHG': 62
9, '600663.XSHG': 352, '600157.XSHG': 1571, '002008.XSHE': 1327,
 '000046.XSHE': 735, '000413.XSHE': 781, '000895.XSHE': 229, '00
2673.XSHE': 1716, '600383.XSHG': 2968, '000539.XSHE': 992.0, '60
0886.XSHG': 1104, '600011.XSHG': 421}
日期： 2013-08-07 00:00:00 ,持仓： {'002450.XSHE': 664, '600108.XS
HG': 530, '601991.XSHG': 489, '600583.XSHG': 492, '000712.XSHE':
 678, '000783.XSHE': 915, '600027.XSHG': 1342, '600369.XSHG': 62
9, '600663.XSHG': 352, '600157.XSHG': 1571, '002008.XSHE': 1327,
 '000046.XSHE': 735, '000413.XSHE': 781, '000895.XSHE': 229, '00
2673.XSHE': 1716, '600383.XSHG': 2968, '000539.XSHE': 992.0, '60
0886.XSHG': 1104, '600011.XSHG': 421}
日期： 2013-08-08 00:00:00 ,持仓： {'002450.XSHE': 664, '600108.XS
HG': 530, '601991.XSHG': 489, '600583.XSHG': 492, '000712.XSHE':
 678, '000783.XSHE': 915, '600027.XSHG': 1342, '600369.XSHG': 62
9, '600663.XSHG': 352, '600157.XSHG': 1571, '002008.XSHE': 1327,
 '000046.XSHE': 735, '000413.XSHE': 781, '000895.XSHE': 229, '00
2673.XSHE': 1716, '600383.XSHG': 2968, '000539.XSHE': 992.0, '60
0886.XSHG': 1104, '600011.XSHG': 421}
日期： 2013-08-09 00:00:00 ,持仓： {'002450.XSHE': 664, '600108.XS
HG': 530, '601991.XSHG': 489, '600583.XSHG': 492, '000712.XSHE':
 678, '000783.XSHE': 915, '600027.XSHG': 1342, '600369.XSHG': 62
9, '600663.XSHG': 352, '600157.XSHG': 1571, '002008.XSHE': 1327,
 '000046.XSHE': 735, '000413.XSHE': 781, '000895.XSHE': 229, '00
2673.XSHE': 1716, '600383.XSHG': 2968, '000539.XSHE': 992.0, '60
0886.XSHG': 1104, '600011.XSHG': 421}
日期： 2013-08-12 00:00:00 ,持仓： {'002450.XSHE': 664, '600108.XS
HG': 530, '601991.XSHG': 489, '600583.XSHG': 492, '000712.XSHE':
 678, '000783.XSHE': 915, '600027.XSHG': 1342, '600369.XSHG': 62
9, '600663.XSHG': 352, '600157.XSHG': 1571, '002008.XSHE': 1327,
 '000046.XSHE': 735, '000413.XSHE': 781, '000895.XSHE': 229, '00
2673.XSHE': 1716, '600383.XSHG': 2968, '000539.XSHE': 992.0, '60
```

```
0886.XSHG': 1104, '600011.XSHG': 421}
日期： 2013-08-13 00:00:00 ,持仓： {'002450.XSHE': 664, '600108.XS
HG': 530, '600583.XSHG': 492, '000712.XSHE': 678, '000783.XSHE':
 915, '600027.XSHG': 1342, '600369.XSHG': 629, '600663.XSHG': 35
2, '600157.XSHG': 1571, '002008.XSHE': 1327, '000046.XSHE': 735,
 '000413.XSHE': 781, '000895.XSHE': 229, '002673.XSHE': 1716, '6
00383.XSHG': 2968, '000539.XSHE': 992.0, '600886.XSHG': 1104, '6
00011.XSHG': 421}
日期： 2013-08-14 00:00:00 ,持仓： {'002450.XSHE': 664, '600108.XS
HG': 530, '600011.XSHG': 421, '000712.XSHE': 678, '000783.XSHE':
 915, '600027.XSHG': 1342, '600369.XSHG': 629, '600663.XSHG': 35
2, '600157.XSHG': 1571, '002008.XSHE': 1327, '000046.XSHE': 735,
 '000413.XSHE': 781, '000895.XSHE': 229, '002673.XSHE': 1716, '0
00539.XSHE': 992.0, '600886.XSHG': 1104, '600383.XSHG': 2968}
日期： 2013-08-15 00:00:00 ,持仓： {'002450.XSHE': 664, '600108.XS
HG': 530, '600011.XSHG': 421, '000712.XSHE': 678, '000783.XSHE':
 915, '600027.XSHG': 1342, '600369.XSHG': 629, '600663.XSHG': 35
2, '600157.XSHG': 1571, '002008.XSHE': 1327, '000046.XSHE': 735,
 '002673.XSHE': 1716, '000895.XSHE': 229, '000539.XSHE': 992.0,
'600886.XSHG': 1104, '600383.XSHG': 2968}
日期： 2013-08-16 00:00:00 ,持仓： {'002450.XSHE': 664, '600108.XS
HG': 530, '600011.XSHG': 421, '000712.XSHE': 678, '000783.XSHE':
 915, '600027.XSHG': 1342, '600369.XSHG': 629, '600663.XSHG': 35
2, '600157.XSHG': 1571, '002008.XSHE': 1327, '000046.XSHE': 735,
 '002673.XSHE': 1716, '000895.XSHE': 229, '000539.XSHE': 992.0,
'600886.XSHG': 1104, '600383.XSHG': 2968}
日期： 2013-08-19 00:00:00 ,持仓： {'002450.XSHE': 664, '600108.XS
HG': 530, '000712.XSHE': 678, '000783.XSHE': 915, '600027.XSHG':
 1342, '600369.XSHG': 629, '600663.XSHG': 352, '600157.XSHG': 15
71, '002008.XSHE': 1327, '000046.XSHE': 735, '002673.XSHE': 1716
, '000895.XSHE': 229, '000539.XSHE': 992.0, '600886.XSHG': 1104,
 '600383.XSHG': 2968}
日期： 2013-08-20 00:00:00 ,持仓： {'000783.XSHE': 915, '600369.XS
HG': 629, '600663.XSHG': 352, '600157.XSHG': 1571, '002008.XSHE'
: 1327, '600886.XSHG': 1104, '002673.XSHE': 1716, '000895.XSHE':
 229, '000539.XSHE': 992.0, '600383.XSHG': 2968}
日期： 2013-08-21 00:00:00 ,持仓： {'000783.XSHE': 915, '600369.XS
HG': 629, '600663.XSHG': 352, '600157.XSHG': 1571, '002008.XSHE'
: 1327, '000413.XSHE': 1486, '000895.XSHE': 229, '002673.XSHE':
1716, '000539.XSHE': 992.0, '600383.XSHG': 2968}
日期： 2013-08-22 00:00:00 ,持仓： {'000783.XSHE': 915, '000413.XS
HE': 1486, '600663.XSHG': 352, '600157.XSHG': 1571, '002008.XSHE
': 1327, '002673.XSHE': 1716, '000895.XSHE': 229, '000539.XSHE':
 992.0, '600383.XSHG': 2968}
日期： 2013-08-23 00:00:00 ,持仓： {'000783.XSHE': 915, '600663.XS
HG': 352, '002008.XSHE': 1327, '300251.XSHE': 573, '000413.XSHE'
: 1486, '000895.XSHE': 229, '002673.XSHE': 1716, '000539.XSHE':
992.0, '600383.XSHG': 2968}
日期： 2013-08-26 00:00:00 ,持仓： {'000783.XSHE': 915, '300027.XS
HE': 356, '600663.XSHG': 352, '002008.XSHE': 1327, '300251.XSHE'
: 573, '000413.XSHE': 1486, '000895.XSHE': 229, '002673.XSHE': 1
716, '000539.XSHE': 992.0, '600383.XSHG': 2968}
日期： 2013-08-27 00:00:00 ,持仓： {'000783.XSHE': 915, '300027.XS
```

```
HE': 356, '600663.XSHG': 352, '002008.XSHE': 1327, '300251.XSHE'
: 573, '002456.XSHE': 243, '000413.XSHE': 1486, '000895.XSHE': 2
29, '002673.XSHE': 1716, '000539.XSHE': 992.0, '600383.XSHG': 29
68}
日期： 2013-08-28 00:00:00 ,持仓： {'000783.XSHE': 915, '300027.XS
HE': 356, '600663.XSHG': 352, '002008.XSHE': 1327, '300251.XSHE'
: 573, '000046.XSHE': 951, '002456.XSHE': 243, '000413.XSHE': 14
86, '000895.XSHE': 229, '002673.XSHE': 1716, '000539.XSHE': 992.
0, '600252.XSHG': 952, '600383.XSHG': 2968}
日期： 2013-08-29 00:00:00 ,持仓： {'002450.XSHE': 150, '000783.XS
HE': 915, '000625.XSHE': 279, '601555.XSHG': 382, '600663.XSHG':
 352, '002008.XSHE': 1327, '300027.XSHE': 356, '300251.XSHE': 57
3, '000046.XSHE': 951, '002456.XSHE': 243, '000413.XSHE': 1486,
'000895.XSHE': 229, '002673.XSHE': 1716, '000539.XSHE': 992.0, '
600886.XSHG': 786, '600252.XSHG': 952, '600383.XSHG': 2968}
日期： 2013-08-30 00:00:00 ,持仓： {'002450.XSHE': 150, '600583.XS
HG': 79, '000712.XSHE': 57, '000783.XSHE': 915, '000625.XSHE': 2
79, '601555.XSHG': 382, '600663.XSHG': 352, '002008.XSHE': 1327,
 '300027.XSHE': 356, '300251.XSHE': 573, '000046.XSHE': 951, '00
2456.XSHE': 243, '000413.XSHE': 1486, '000895.XSHE': 229, '00267
3.XSHE': 1716, '000539.XSHE': 992.0, '600886.XSHG': 786, '600252
.XSHG': 952, '600383.XSHG': 2968}
日期： 2013-09-02 00:00:00 ,持仓： {'600583.XSHG': 79, '000712.XSH
E': 57, '000783.XSHE': 915, '000625.XSHE': 279, '300027.XSHE': 3
56, '600663.XSHG': 352, '002008.XSHE': 1327, '300251.XSHE': 573,
 '000046.XSHE': 951, '002456.XSHE': 243, '601555.XSHG': 382, '00
0895.XSHE': 229, '002673.XSHE': 1716, '000539.XSHE': 992.0, '600
886.XSHG': 786, '600252.XSHG': 952, '600383.XSHG': 2968}
日期： 2013-09-03 00:00:00 ,持仓： {'600108.XSHG': 349, '600583.XS
HG': 79, '000712.XSHE': 57, '000783.XSHE': 915, '600027.XSHG': 8
92, '000625.XSHE': 279, '300027.XSHE': 356, '600663.XSHG': 352,
'002008.XSHE': 1327, '300251.XSHE': 573, '600886.XSHG': 786, '60
1555.XSHG': 382, '000895.XSHE': 229, '002673.XSHE': 1716, '60038
3.XSHG': 2968, '000539.XSHE': 992.0, '600252.XSHG': 952, '600011
.XSHG': 508}
日期： 2013-09-04 00:00:00 ,持仓： {'600108.XSHG': 349, '600583.XS
HG': 79, '000783.XSHE': 915, '600027.XSHG': 892, '300027.XSHE':
356, '600663.XSHG': 352, '002008.XSHE': 1327, '000625.XSHE': 279
, '600886.XSHG': 786, '601555.XSHG': 382, '000895.XSHE': 229, '0
02673.XSHE': 1716, '600383.XSHG': 2968, '000539.XSHE': 992.0, '6
00011.XSHG': 508}
日期： 2013-09-05 00:00:00 ,持仓： {'600108.XSHG': 349, '600583.XS
HG': 79, '000712.XSHE': 544, '000783.XSHE': 915, '600027.XSHG':
892, '300027.XSHE': 356, '600663.XSHG': 352, '002008.XSHE': 1327
, '000625.XSHE': 279, '600886.XSHG': 786, '601555.XSHG': 382, '0
00895.XSHE': 229, '002673.XSHE': 1716, '600383.XSHG': 2968, '000
539.XSHE': 992.0, '600011.XSHG': 508}
日期： 2013-09-06 00:00:00 ,持仓： {'600108.XSHG': 349, '600583.XS
HG': 79, '000712.XSHE': 544, '000783.XSHE': 915, '600027.XSHG':
892, '300027.XSHE': 356, '600663.XSHG': 352, '002008.XSHE': 1327
, '000625.XSHE': 279, '600886.XSHG': 786, '601555.XSHG': 382, '0
00895.XSHE': 229, '002673.XSHE': 1716, '600383.XSHG': 2968, '000
539.XSHE': 992.0, '600011.XSHG': 508}
```

```
日期： 2013-09-09 00:00:00 ,持仓： {'600108.XSHG': 349, '601991.XS
HG': 1024, '000712.XSHE': 544, '000783.XSHE': 915, '600027.XSHG'
: 892, '000625.XSHE': 279, '300027.XSHE': 356, '600663.XSHG': 35
2, '002008.XSHE': 1327, '300251.XSHE': 281, '600886.XSHG': 786,
'002456.XSHE': 187, '601555.XSHG': 382, '000895.XSHE': 229, '002
673.XSHE': 1716, '000539.XSHE': 992.0, '600383.XSHG': 2968}
日期： 2013-09-10 00:00:00 ,持仓： {'600108.XSHG': 349, '000503.XS
HE': 126, '601991.XSHG': 1024, '000712.XSHE': 544, '000783.XSHE'
: 915, '600027.XSHG': 892, '000625.XSHE': 279, '300027.XSHE': 35
6, '600663.XSHG': 352, '002008.XSHE': 1327, '300251.XSHE': 281,
'000046.XSHE': 507, '002456.XSHE': 187, '601555.XSHG': 382, '000
895.XSHE': 229, '002673.XSHE': 1716, '000539.XSHE': 992.0, '6008
86.XSHG': 786, '600383.XSHG': 2968}
日期： 2013-09-11 00:00:00 ,持仓： {'002450.XSHE': 83, '000503.XSH
E': 126, '601991.XSHG': 1024, '600583.XSHG': 202, '000712.XSHE':
 544, '000783.XSHE': 915, '600027.XSHG': 892, '000625.XSHE': 279
, '300027.XSHE': 356, '600663.XSHG': 352, '600108.XSHG': 349, '0
02008.XSHE': 1327, '300251.XSHE': 281, '000046.XSHE': 507, '0024
56.XSHE': 187, '601555.XSHG': 382, '000895.XSHE': 229, '002673.X
SHE': 1716, '000539.XSHE': 992.0, '600886.XSHG': 786, '600383.XS
HG': 2968}
日期： 2013-09-12 00:00:00 ,持仓： {'600583.XSHG': 202, '600027.XS
HG': 892, '002456.XSHE': 187, '000895.XSHE': 229, '000539.XSHE':
 992.0, '600011.XSHG': 174, '000783.XSHE': 915, '002008.XSHE': 1
327, '300251.XSHE': 281, '601555.XSHG': 382, '000725.XSHE': 409,
 '000503.XSHE': 126, '000046.XSHE': 507, '002673.XSHE': 1716, '6
00383.XSHG': 2968, '600108.XSHG': 349, '300027.XSHE': 356, '6019
91.XSHG': 1024, '600663.XSHG': 352, '000712.XSHE': 544, '000625.
XSHE': 279, '600886.XSHG': 786}
日期： 2013-09-13 00:00:00 ,持仓： {'600108.XSHG': 349, '000503.XS
HE': 126, '601991.XSHG': 1024, '600583.XSHG': 202, '600383.XSHG'
: 2968, '000783.XSHE': 915, '600027.XSHG': 892, '000625.XSHE': 2
79, '300027.XSHE': 356, '000712.XSHE': 544, '600663.XSHG': 352,
'002008.XSHE': 1327, '300251.XSHE': 281, '000046.XSHE': 507, '60
1555.XSHG': 382, '000895.XSHE': 229, '002673.XSHE': 1716, '00072
5.XSHE': 409, '000539.XSHE': 992.0, '600886.XSHG': 786, '600011.
XSHG': 174}
日期： 2013-09-16 00:00:00 ,持仓： {'600583.XSHG': 202, '600027.XS
HG': 892, '000895.XSHE': 229, '000539.XSHE': 992.0, '600011.XSHG
': 174, '600674.XSHG': 288, '000783.XSHE': 915, '002008.XSHE': 1
327, '300251.XSHE': 281, '601555.XSHG': 382, '000725.XSHE': 409,
 '000503.XSHE': 126, '000046.XSHE': 507, '002673.XSHE': 1716, '6
00383.XSHG': 2968, '600108.XSHG': 349, '300027.XSHE': 356, '6019
91.XSHG': 1024, '600663.XSHG': 352, '000712.XSHE': 544, '000625.
XSHE': 279, '600886.XSHG': 786}
日期： 2013-09-17 00:00:00 ,持仓： {'600583.XSHG': 202, '600027.XS
HG': 892, '000413.XSHE': 244, '000895.XSHE': 229, '000539.XSHE':
 992.0, '600011.XSHG': 174, '600674.XSHG': 288, '000783.XSHE': 9
15, '002008.XSHE': 1327, '300251.XSHE': 281, '601555.XSHG': 382,
 '000725.XSHE': 409, '000503.XSHE': 126, '000046.XSHE': 507, '00
2673.XSHE': 1716, '600383.XSHG': 2968, '600108.XSHG': 349, '3000
27.XSHE': 356, '601991.XSHG': 1024, '600663.XSHG': 352, '000712.
XSHE': 544, '000625.XSHE': 279}
```

```
日期： 2013-09-18 00:00:00 ,持仓： {'600108.XSHG': 349, '000503.XS
HE': 126, '601991.XSHG': 1024, '600583.XSHG': 202, '600674.XSHG'
: 288, '000783.XSHE': 915, '600027.XSHG': 892, '600383.XSHG': 29
68, '601555.XSHG': 382, '600663.XSHG': 352, '002008.XSHE': 1327,
 '300027.XSHE': 356, '300251.XSHE': 281, '000046.XSHE': 507, '00
0413.XSHE': 244, '000895.XSHE': 229, '002673.XSHE': 1716, '00072
5.XSHE': 409, '000539.XSHE': 992.0, '600011.XSHG': 174}
日期： 2013-09-23 00:00:00 ,持仓： {'000503.XSHE': 126, '601991.XS
HG': 1024, '600011.XSHG': 174, '600674.XSHG': 288, '000783.XSHE'
: 915, '600027.XSHG': 892, '601555.XSHG': 382, '600663.XSHG': 35
2, '002008.XSHE': 1327, '300027.XSHE': 356, '300251.XSHE': 281,
'000413.XSHE': 244, '000895.XSHE': 229, '002673.XSHE': 1716, '00
0725.XSHE': 409, '000539.XSHE': 992.0, '600383.XSHG': 2968}
日期： 2013-09-24 00:00:00 ,持仓： {'000503.XSHE': 126, '600011.XS
HG': 174, '600674.XSHG': 288, '000783.XSHE': 915, '300027.XSHE':
 356, '600663.XSHG': 352, '002008.XSHE': 1327, '300251.XSHE': 28
1, '000413.XSHE': 244, '000895.XSHE': 229, '002673.XSHE': 1716,
'000725.XSHE': 409, '000539.XSHE': 992.0, '600252.XSHG': 1019, '
600383.XSHG': 2968}
日期： 2013-09-25 00:00:00 ,持仓： {'000503.XSHE': 126, '000783.XS
HE': 915, '300027.XSHE': 356, '600663.XSHG': 352, '002008.XSHE':
 1327, '300251.XSHE': 281, '000413.XSHE': 244, '000895.XSHE': 22
9, '002673.XSHE': 1716, '000539.XSHE': 992.0, '600252.XSHG': 101
9, '600383.XSHG': 2968}
日期： 2013-09-26 00:00:00 ,持仓： {'002450.XSHE': 385, '000783.XS
HE': 915, '300027.XSHE': 356, '600663.XSHG': 352, '002008.XSHE':
 1327, '300251.XSHE': 281, '000413.XSHE': 244, '000895.XSHE': 22
9, '002673.XSHE': 1716, '000539.XSHE': 992.0, '600252.XSHG': 101
9, '600383.XSHG': 2968}
日期： 2013-09-27 00:00:00 ,持仓： {'000783.XSHE': 915, '300027.XS
HE': 356, '600663.XSHG': 352, '002008.XSHE': 1327, '300251.XSHE'
: 281, '000413.XSHE': 244, '000895.XSHE': 229, '002673.XSHE': 17
16, '000539.XSHE': 992.0, '600252.XSHG': 1019, '600383.XSHG': 29
68}
日期： 2013-09-30 00:00:00 ,持仓： {'000783.XSHE': 915, '300027.XS
HE': 356, '600663.XSHG': 352, '002008.XSHE': 1327, '300251.XSHE'
: 281, '002673.XSHE': 1716, '000895.XSHE': 229, '000539.XSHE': 9
92.0, '600252.XSHG': 1019, '600383.XSHG': 2968}
日期： 2013-10-08 00:00:00 ,持仓： {'600108.XSHG': 863, '000783.XS
HE': 915, '300027.XSHE': 356, '600663.XSHG': 352, '002008.XSHE':
 1327, '300251.XSHE': 281, '002673.XSHE': 1716, '000895.XSHE': 2
29, '000539.XSHE': 992.0, '600252.XSHG': 1019, '600383.XSHG': 29
68}
日期： 2013-10-09 00:00:00 ,持仓： {'600108.XSHG': 863, '000783.XS
HE': 915, '300027.XSHE': 356, '600663.XSHG': 352, '002008.XSHE':
 1327, '300251.XSHE': 281, '002673.XSHE': 1716, '000895.XSHE': 2
29, '000539.XSHE': 992.0, '600252.XSHG': 1019, '600383.XSHG': 29
68}
日期： 2013-10-10 00:00:00 ,持仓： {'600108.XSHG': 863, '600583.XS
HG': 778, '000783.XSHE': 915, '300027.XSHE': 356, '600663.XSHG':
 352, '002008.XSHE': 1327, '300251.XSHE': 281, '002673.XSHE': 17
16, '000895.XSHE': 229, '000539.XSHE': 992.0, '600252.XSHG': 101
9, '600383.XSHG': 2968}
```

```
日期： 2013-10-11 00:00:00 ,持仓： {'600108.XSHG': 863, '000503.XS
HE': 214, '600583.XSHG': 778, '600674.XSHG': 821, '000783.XSHE':
 915, '000625.XSHE': 425, '002008.XSHE': 1327, '300251.XSHE': 28
1, '000413.XSHE': 762, '000895.XSHE': 229, '002673.XSHE': 1716,
'000539.XSHE': 992.0, '600252.XSHG': 1019, '600383.XSHG': 2968}
日期： 2013-10-14 00:00:00 ,持仓： {'600108.XSHG': 863, '000503.XS
HE': 214, '600583.XSHG': 778, '600674.XSHG': 821, '000783.XSHE':
 915, '000625.XSHE': 425, '002008.XSHE': 1327, '300251.XSHE': 28
1, '600886.XSHG': 1476, '000413.XSHE': 762, '000895.XSHE': 229,
'002673.XSHE': 1716, '000539.XSHE': 992.0, '600252.XSHG': 1019,
'600383.XSHG': 2968}
日期： 2013-10-15 00:00:00 ,持仓： {'002450.XSHE': 221.0, '000503.
XSHE': 214, '000625.XSHE': 425, '600583.XSHG': 778, '600674.XSHG
': 821, '000783.XSHE': 915, '600027.XSHG': 1257, '600383.XSHG':
2968, '601555.XSHG': 465, '600108.XSHG': 863, '002008.XSHE': 132
7, '300251.XSHE': 281, '000046.XSHE': 745, '000413.XSHE': 762, '
000895.XSHE': 229, '002673.XSHE': 1716, '000725.XSHE': 1637, '00
0539.XSHE': 992.0, '600886.XSHG': 1476, '600252.XSHG': 1019, '60
0011.XSHG': 710}
日期： 2013-10-16 00:00:00 ,持仓： {'600108.XSHG': 863, '000503.XS
HE': 214, '600583.XSHG': 778, '600674.XSHG': 821, '000783.XSHE':
 915, '600027.XSHG': 1257, '600383.XSHG': 2968, '601555.XSHG': 4
65, '002008.XSHE': 1327, '000625.XSHE': 425, '000046.XSHE': 745,
 '000413.XSHE': 762, '000895.XSHE': 229, '002673.XSHE': 1716, '0
00725.XSHE': 1637, '000539.XSHE': 992.0, '600886.XSHG': 1476, '6
00011.XSHG': 710}
日期： 2013-10-17 00:00:00 ,持仓： {'600108.XSHG': 863, '000503.XS
HE': 214, '600583.XSHG': 778, '600674.XSHG': 821, '000783.XSHE':
 915, '600027.XSHG': 1257, '600383.XSHG': 2968, '002008.XSHE': 1
327, '000625.XSHE': 425, '000046.XSHE': 745, '000413.XSHE': 762,
 '000895.XSHE': 229, '002673.XSHE': 1716, '000725.XSHE': 1637, '
000539.XSHE': 992.0, '600886.XSHG': 1476, '600011.XSHG': 710}
日期： 2013-10-18 00:00:00 ,持仓： {'600108.XSHG': 863, '000503.XS
HE': 214, '600583.XSHG': 778, '600674.XSHG': 821, '000783.XSHE':
 915, '600027.XSHG': 1257, '600383.XSHG': 2968, '600663.XSHG': 1
56, '002008.XSHE': 1327, '000625.XSHE': 425, '000046.XSHE': 745,
 '000413.XSHE': 762, '000895.XSHE': 229, '002673.XSHE': 1716, '0
00725.XSHE': 1637, '000539.XSHE': 992.0, '600886.XSHG': 1476, '6
00011.XSHG': 710}
日期： 2013-10-21 00:00:00 ,持仓： {'600108.XSHG': 863, '000503.XS
HE': 214, '600583.XSHG': 778, '600674.XSHG': 821, '000783.XSHE':
 915, '600027.XSHG': 1257, '002008.XSHE': 1327, '000625.XSHE': 4
25, '000046.XSHE': 745, '002456.XSHE': 121, '000413.XSHE': 762,
'000895.XSHE': 229, '002673.XSHE': 1716, '600383.XSHG': 2968, '0
00539.XSHE': 992.0, '600886.XSHG': 1476, '600011.XSHG': 710}
日期： 2013-10-22 00:00:00 ,持仓： {'000712.XSHE': 334, '000503.XS
HE': 214, '600011.XSHG': 710, '600674.XSHG': 821, '000783.XSHE':
 915, '600027.XSHG': 1257, '002008.XSHE': 1327, '000625.XSHE': 4
25, '000046.XSHE': 745, '002456.XSHE': 121, '000413.XSHE': 762,
'000895.XSHE': 229, '002673.XSHE': 1716, '000539.XSHE': 992.0, '
600886.XSHG': 1476, '600383.XSHG': 2968}
日期： 2013-10-23 00:00:00 ,持仓： {'000712.XSHE': 334, '000503.XS
HE': 214, '600011.XSHG': 710, '600674.XSHG': 821, '000783.XSHE':
 915, '600027.XSHG': 1257, '002008.XSHE': 1327, '000625.XSHE': 4
25, '000046.XSHE': 745, '002456.XSHE': 121, '000413.XSHE': 762,
'000895.XSHE': 229, '002673.XSHE': 1716, '000539.XSHE': 992.0, '
600886.XSHG': 1476, '600383.XSHG': 2968}
日期： 2013-10-24 00:00:00 ,持仓： {'002450.XSHE': 339, '000503.XS
HE': 214, '600583.XSHG': 721, '600674.XSHG': 821, '000783.XSHE':
 915, '600027.XSHG': 1257, '000712.XSHE': 334, '002008.XSHE': 13
27, '000625.XSHE': 425, '002456.XSHE': 121, '000413.XSHE': 762,
'000895.XSHE': 229, '002673.XSHE': 1716, '600383.XSHG': 2968, '0
00539.XSHE': 992.0, '600011.XSHG': 710}
日期： 2013-10-25 00:00:00 ,持仓： {'002450.XSHE': 339, '000503.XS
HE': 214, '600583.XSHG': 721, '600674.XSHG': 821, '000783.XSHE':
 915, '000712.XSHE': 334, '002008.XSHE': 1327, '002456.XSHE': 12
1, '000413.XSHE': 762, '000895.XSHE': 229, '002673.XSHE': 1716,
'600383.XSHG': 2968, '000539.XSHE': 992.0, '600011.XSHG': 710}
日期： 2013-10-28 00:00:00 ,持仓： {'002450.XSHE': 339, '600583.XS
HG': 721, '600674.XSHG': 821, '000783.XSHE': 915, '000712.XSHE':
 334, '002008.XSHE': 1327, '002456.XSHE': 121, '000413.XSHE': 76
2, '000895.XSHE': 229, '002673.XSHE': 1716, '000725.XSHE': 3262,
 '000539.XSHE': 992.0, '600383.XSHG': 2968}
日期： 2013-10-29 00:00:00 ,持仓： {'002450.XSHE': 339, '600583.XS
HG': 721, '600674.XSHG': 821, '000783.XSHE': 915, '000712.XSHE':
 334, '002008.XSHE': 1327, '002456.XSHE': 121, '000413.XSHE': 76
2, '000895.XSHE': 229, '002673.XSHE': 1716, '000725.XSHE': 3262,
 '000539.XSHE': 992.0, '600383.XSHG': 2968}
日期： 2013-10-30 00:00:00 ,持仓： {'002450.XSHE': 339, '600583.XS
HG': 721, '600674.XSHG': 821, '000783.XSHE': 915, '000712.XSHE':
 334, '002008.XSHE': 1327, '600886.XSHG': 1941, '002456.XSHE': 1
21, '000413.XSHE': 762, '000895.XSHE': 229, '002673.XSHE': 1716,
 '000725.XSHE': 3262, '000539.XSHE': 992.0, '600383.XSHG': 2968}
日期： 2013-10-31 00:00:00 ,持仓： {'002450.XSHE': 339, '600583.XS
HG': 721, '600674.XSHG': 821, '000783.XSHE': 915, '600027.XSHG':
 1955, '002008.XSHE': 1327, '600886.XSHG': 1941, '002456.XSHE':
121, '000413.XSHE': 762, '000895.XSHE': 229, '002673.XSHE': 1716
, '000725.XSHE': 3262, '000539.XSHE': 992.0, '600383.XSHG': 2968
}
日期： 2013-11-01 00:00:00 ,持仓： {'002450.XSHE': 339, '601628.XS
HG': 393, '600583.XSHG': 721, '600674.XSHG': 821, '000783.XSHE':
 915, '600027.XSHG': 1955, '002008.XSHE': 1327, '600886.XSHG': 1
941, '002456.XSHE': 121, '000413.XSHE': 762, '000895.XSHE': 229,
 '002673.XSHE': 1716, '000725.XSHE': 3262, '000539.XSHE': 992.0,
 '600383.XSHG': 2968}
日期： 2013-11-04 00:00:00 ,持仓： {'002450.XSHE': 339, '601628.XS
HG': 393, '600369.XSHG': 967, '600583.XSHG': 721, '600674.XSHG':
 821, '601601.XSHG': 259, '600027.XSHG': 1955, '600383.XSHG': 29
68, '000783.XSHE': 915, '002008.XSHE': 1327, '000046.XSHE': 907,
 '002456.XSHE': 121, '000413.XSHE': 762, '000895.XSHE': 229, '00
2673.XSHE': 1716, '000725.XSHE': 3262, '000539.XSHE': 992.0, '60
0886.XSHG': 1941, '600011.XSHG': 815}
日期： 2013-11-05 00:00:00 ,持仓： {'600583.XSHG': 721, '601601.XS
HG': 259, '600027.XSHG': 1955, '002456.XSHE': 121, '000413.XSHE'
: 762, '000895.XSHE': 229, '000539.XSHE': 992.0, '002450.XSHE':
339, '600011.XSHG': 815, '600674.XSHG': 821, '000783.XSHE': 915,
 '600369.XSHG': 967, '002008.XSHE': 1327, '000686.XSHE': 106, '0
00725.XSHE': 3262, '000046.XSHE': 907, '002673.XSHE': 1716, '600
383.XSHG': 2968, '601628.XSHG': 393, '000625.XSHE': 70, '600886.
XSHG': 1941, '300017.XSHE': 51}
日期： 2013-11-06 00:00:00 ,持仓： {'600583.XSHG': 721, '601601.XS
HG': 259, '600027.XSHG': 1955, '002456.XSHE': 121, '000413.XSHE'
: 762, '000895.XSHE': 229, '000539.XSHE': 992.0, '002450.XSHE':
339, '600011.XSHG': 815, '600674.XSHG': 821, '000783.XSHE': 915,
 '600369.XSHG': 967, '002008.XSHE': 1327, '000686.XSHE': 106, '0
00725.XSHE': 3262, '000046.XSHE': 907, '002673.XSHE': 1716, '600
383.XSHG': 2968, '601628.XSHG': 393, '000625.XSHE': 70, '600886.
XSHG': 1941, '300017.XSHE': 51}
日期： 2013-11-07 00:00:00 ,持仓： {'002450.XSHE': 339, '601628.XS
HG': 393, '600369.XSHG': 967, '600583.XSHG': 721, '600674.XSHG':
 821, '601601.XSHG': 259, '600027.XSHG': 1955, '600352.XSHG': 59
, '000783.XSHE': 915, '002008.XSHE': 1327, '600383.XSHG': 2968,
'000625.XSHE': 70, '600886.XSHG': 1941, '002456.XSHE': 121, '000
413.XSHE': 762, '300017.XSHE': 51, '000895.XSHE': 229, '002673.X
SHE': 1716, '000725.XSHE': 3262, '000539.XSHE': 992.0, '600011.X
SHG': 815}
日期： 2013-11-08 00:00:00 ,持仓： {'002450.XSHE': 339, '601628.XS
HG': 393, '600369.XSHG': 967, '600583.XSHG': 721, '600674.XSHG':
 821, '601601.XSHG': 259, '600027.XSHG': 1955, '600352.XSHG': 59
, '000783.XSHE': 915, '002008.XSHE': 1327, '600383.XSHG': 2968,
'000625.XSHE': 70, '600886.XSHG': 1941, '002456.XSHE': 121, '000
413.XSHE': 762, '300017.XSHE': 51, '000895.XSHE': 229, '002673.X
SHE': 1716, '000725.XSHE': 3262, '000539.XSHE': 992.0, '600011.X
SHG': 815}
日期： 2013-11-11 00:00:00 ,持仓： {'002450.XSHE': 339, '600583.XS
HG': 721, '600674.XSHG': 821, '000783.XSHE': 915, '002202.XSHE':
 162, '600369.XSHG': 967, '002008.XSHE': 1327, '600383.XSHG': 29
68, '000625.XSHE': 70, '600886.XSHG': 1941, '002456.XSHE': 121,
'000413.XSHE': 762, '300017.XSHE': 51, '000895.XSHE': 229, '0026
73.XSHE': 1716, '000725.XSHE': 3262, '000539.XSHE': 992.0, '6000
11.XSHG': 815}
日期： 2013-11-12 00:00:00 ,持仓： {'002450.XSHE': 339, '600583.XS
HG': 721, '000783.XSHE': 915, '002202.XSHE': 162, '002008.XSHE':
 1327, '600886.XSHG': 1941, '002456.XSHE': 121, '000413.XSHE': 7
62, '300017.XSHE': 51, '000895.XSHE': 229, '002673.XSHE': 1716,
'000725.XSHE': 3262, '000539.XSHE': 992.0, '600383.XSHG': 2968}
日期： 2013-11-13 00:00:00 ,持仓： {'002450.XSHE': 339, '600583.XS
HG': 721, '000783.XSHE': 915, '002008.XSHE': 1327, '002456.XSHE'
: 121, '000413.XSHE': 762, '000895.XSHE': 229, '002673.XSHE': 17
16, '000725.XSHE': 3262, '000539.XSHE': 992.0, '600383.XSHG': 29
68}
日期： 2013-11-14 00:00:00 ,持仓： {'002450.XSHE': 339, '600583.XS
HG': 721, '000783.XSHE': 915, '600648.XSHG': 234, '002008.XSHE':
 1327, '002456.XSHE': 121, '000413.XSHE': 762, '000895.XSHE': 22
9, '002673.XSHE': 1716, '000725.XSHE': 3262, '000539.XSHE': 992.
0, '600383.XSHG': 2968}
日期： 2013-11-15 00:00:00 ,持仓： {'002450.XSHE': 339, '000783.XS
HE': 915, '002008.XSHE': 1327, '002456.XSHE': 121, '000413.XSHE'
: 762, '000895.XSHE': 229, '002673.XSHE': 1716, '000725.XSHE': 3
```

```
262, '000539.XSHE': 992.0, '600383.XSHG': 2968}
日期： 2013-11-18 00:00:00 ,持仓： {'002450.XSHE': 339, '601628.XS
HG': 752, '601601.XSHG': 611, '000783.XSHE': 915, '002008.XSHE':
 1327, '002456.XSHE': 121, '000413.XSHE': 762, '000895.XSHE': 22
9, '002673.XSHE': 1716, '000725.XSHE': 3262, '000539.XSHE': 992.
0, '600383.XSHG': 2968}
日期： 2013-11-19 00:00:00 ,持仓： {'002450.XSHE': 339, '000917.XS
HE': 393, '601628.XSHG': 752, '000712.XSHE': 573, '601601.XSHG':
 611, '600352.XSHG': 1021.0, '000783.XSHE': 915, '002008.XSHE':
1327, '300017.XSHE': 360, '002456.XSHE': 121, '000413.XSHE': 762
, '601099.XSHG': 1772, '000895.XSHE': 229, '002673.XSHE': 1716,
'000725.XSHE': 3262, '000539.XSHE': 992.0, '600383.XSHG': 2968}
日期： 2013-11-20 00:00:00 ,持仓： {'002450.XSHE': 339, '000917.XS
HE': 393, '601628.XSHG': 752, '000712.XSHE': 573, '601601.XSHG':
 611, '600352.XSHG': 1021.0, '000783.XSHE': 915, '002008.XSHE':
1327, '300017.XSHE': 360, '002456.XSHE': 121, '000413.XSHE': 762
, '601099.XSHG': 1772, '000895.XSHE': 229, '002673.XSHE': 1716,
'000725.XSHE': 3262, '000539.XSHE': 992.0, '600383.XSHG': 2968}
日期： 2013-11-21 00:00:00 ,持仓： {'002450.XSHE': 339, '000917.XS
HE': 393, '601628.XSHG': 752, '000712.XSHE': 573, '601601.XSHG':
 611, '600352.XSHG': 1021.0, '000783.XSHE': 915, '002008.XSHE':
1327, '300017.XSHE': 360, '002456.XSHE': 121, '000413.XSHE': 762
, '601099.XSHG': 1772, '000895.XSHE': 229, '002673.XSHE': 1716,
'000725.XSHE': 3262, '000539.XSHE': 992.0, '600383.XSHG': 2968}
日期： 2013-11-22 00:00:00 ,持仓： {'002450.XSHE': 339, '000917.XS
HE': 393, '601628.XSHG': 752, '000712.XSHE': 573, '601601.XSHG':
 611, '600352.XSHG': 1021.0, '000783.XSHE': 915, '002008.XSHE':
1327, '300017.XSHE': 360, '002456.XSHE': 121, '000413.XSHE': 762
, '601099.XSHG': 1772, '000895.XSHE': 229, '002673.XSHE': 1716,
'000725.XSHE': 3262, '000539.XSHE': 992.0, '600383.XSHG': 2968}
日期： 2013-11-25 00:00:00 ,持仓： {'002450.XSHE': 339, '000917.XS
HE': 393, '601628.XSHG': 752, '000712.XSHE': 573, '601601.XSHG':
 611, '600352.XSHG': 1021.0, '000783.XSHE': 915, '002008.XSHE':
1327, '300017.XSHE': 360, '002456.XSHE': 121, '000413.XSHE': 762
, '601099.XSHG': 1772, '000895.XSHE': 229, '002673.XSHE': 1716,
'000725.XSHE': 3262, '000539.XSHE': 992.0, '600383.XSHG': 2968}
日期： 2013-11-26 00:00:00 ,持仓： {'002450.XSHE': 339, '000917.XS
HE': 393, '601628.XSHG': 752, '000712.XSHE': 573, '601601.XSHG':
 611, '600352.XSHG': 1021.0, '000783.XSHE': 915, '002008.XSHE':
1327, '002456.XSHE': 121, '000413.XSHE': 762, '601099.XSHG': 177
2, '000895.XSHE': 229, '002673.XSHE': 1716, '000725.XSHE': 3262,
 '000539.XSHE': 992.0, '600383.XSHG': 2968}
日期： 2013-11-27 00:00:00 ,持仓： {'002450.XSHE': 339, '000917.XS
HE': 393, '601628.XSHG': 752, '000712.XSHE': 573, '601601.XSHG':
 611, '600352.XSHG': 1021.0, '000783.XSHE': 915, '002008.XSHE':
1327, '002456.XSHE': 121, '000413.XSHE': 762, '601099.XSHG': 177
2, '000895.XSHE': 229, '002673.XSHE': 1716, '000725.XSHE': 3262,
 '000539.XSHE': 992.0, '600252.XSHG': 266, '600383.XSHG': 2968}
日期： 2013-11-28 00:00:00 ,持仓： {'002450.XSHE': 339, '601628.XS
HG': 752, '601601.XSHG': 611, '600352.XSHG': 1021.0, '000783.XSH
E': 915, '002008.XSHE': 1327, '002456.XSHE': 121, '000413.XSHE':
 762, '601099.XSHG': 1772, '000895.XSHE': 229, '002673.XSHE': 17
16, '000725.XSHE': 3262, '000539.XSHE': 992.0, '600252.XSHG': 26
```

```
6, '600383.XSHG': 2968}
日期： 2013-11-29 00:00:00 ,持仓： {'002450.XSHE': 339, '601628.XS
HG': 752, '601601.XSHG': 611, '600352.XSHG': 1021.0, '000783.XSH
E': 915, '002008.XSHE': 1327, '002456.XSHE': 121, '000413.XSHE':
 762, '601099.XSHG': 1772, '000895.XSHE': 229, '002673.XSHE': 17
16, '000725.XSHE': 3262, '000539.XSHE': 992.0, '600252.XSHG': 26
6, '600383.XSHG': 2968}
日期： 2013-12-02 00:00:00 ,持仓： {'002450.XSHE': 339, '300017.XS
HE': 188, '601628.XSHG': 752, '000783.XSHE': 915, '600352.XSHG':
 1021.0, '002008.XSHE': 1327, '002456.XSHE': 121, '000413.XSHE':
 762, '601099.XSHG': 1772, '000895.XSHE': 229, '002673.XSHE': 17
16, '000725.XSHE': 3262, '000539.XSHE': 992.0, '600252.XSHG': 26
6, '600383.XSHG': 2968}
日期： 2013-12-03 00:00:00 ,持仓： {'002450.XSHE': 339, '000917.XS
HE': 308, '601628.XSHG': 752, '000783.XSHE': 915, '600352.XSHG':
 1021.0, '002008.XSHE': 1327, '300017.XSHE': 188, '002456.XSHE':
 121, '000413.XSHE': 762, '601099.XSHG': 1772, '000895.XSHE': 22
9, '002673.XSHE': 1716, '000725.XSHE': 3262, '000539.XSHE': 992.
0, '600252.XSHG': 266, '600383.XSHG': 2968}
日期： 2013-12-04 00:00:00 ,持仓： {'002450.XSHE': 339, '000917.XS
HE': 308, '601628.XSHG': 752, '600011.XSHG': 736.0, '600674.XSHG
': 672, '000783.XSHE': 915, '600027.XSHG': 1324, '002202.XSHE':
485, '002673.XSHE': 1716, '000712.XSHE': 357, '002008.XSHE': 132
7, '300017.XSHE': 188, '002456.XSHE': 121, '000413.XSHE': 762, '
601099.XSHG': 1772, '000895.XSHE': 229, '600352.XSHG': 1021.0, '
000725.XSHE': 3262, '000539.XSHE': 992.0, '600252.XSHG': 266, '6
00383.XSHG': 2968}
日期： 2013-12-05 00:00:00 ,持仓： {'002450.XSHE': 339, '000917.XS
HE': 308, '601628.XSHG': 752, '600011.XSHG': 736.0, '600674.XSHG
': 672, '000783.XSHE': 915, '600027.XSHG': 1324, '002202.XSHE':
485, '002673.XSHE': 1716, '000712.XSHE': 357, '002008.XSHE': 132
7, '300017.XSHE': 188, '002456.XSHE': 121, '000413.XSHE': 762, '
601099.XSHG': 1772, '000895.XSHE': 229, '600352.XSHG': 1021.0, '
000725.XSHE': 3262, '000539.XSHE': 992.0, '600252.XSHG': 266, '6
00383.XSHG': 2968}
日期： 2013-12-06 00:00:00 ,持仓： {'002450.XSHE': 339, '000917.XS
HE': 308, '601628.XSHG': 752, '600011.XSHG': 736.0, '600674.XSHG
': 672, '000783.XSHE': 915, '600027.XSHG': 1324, '002202.XSHE':
485, '002673.XSHE': 1716, '000712.XSHE': 357, '002008.XSHE': 132
7, '300017.XSHE': 188, '002456.XSHE': 121, '000413.XSHE': 762, '
601099.XSHG': 1772, '000895.XSHE': 229, '600352.XSHG': 1021.0, '
000725.XSHE': 3262, '000539.XSHE': 992.0, '600252.XSHG': 266, '6
00383.XSHG': 2968}
日期： 2013-12-09 00:00:00 ,持仓： {'002450.XSHE': 339, '000917.XS
HE': 308, '601628.XSHG': 752, '600011.XSHG': 736.0, '600674.XSHG
': 672, '000783.XSHE': 915, '600027.XSHG': 1324, '002202.XSHE':
485, '002673.XSHE': 1716, '000712.XSHE': 357, '002008.XSHE': 132
7, '300017.XSHE': 188, '002456.XSHE': 121, '000413.XSHE': 762, '
601099.XSHG': 1772, '000895.XSHE': 229, '600352.XSHG': 1021.0, '
000725.XSHE': 3262, '000539.XSHE': 992.0, '600252.XSHG': 266, '6
00383.XSHG': 2968}
日期： 2013-12-10 00:00:00 ,持仓： {'002450.XSHE': 339, '000917.XS
HE': 308, '601628.XSHG': 752, '600011.XSHG': 736.0, '600674.XSHG
```

```
': 672, '000783.XSHE': 915, '600027.XSHG': 1324, '002202.XSHE':
485, '002673.XSHE': 1716, '000712.XSHE': 357, '002008.XSHE': 132
7, '002456.XSHE': 121, '000413.XSHE': 762, '601099.XSHG': 1772,
'000895.XSHE': 229, '600352.XSHG': 1021.0, '000725.XSHE': 3262,
'000539.XSHE': 992.0, '600252.XSHG': 266, '600383.XSHG': 2968}
日期： 2013-12-11 00:00:00 ,持仓： {'002450.XSHE': 339, '000917.XS
HE': 308, '601628.XSHG': 752, '600011.XSHG': 736.0, '600674.XSHG
': 672, '601601.XSHG': 30, '600027.XSHG': 1324, '002202.XSHE': 4
85, '000783.XSHE': 915, '000712.XSHE': 357, '002008.XSHE': 1327,
 '002456.XSHE': 121, '000413.XSHE': 762, '000895.XSHE': 229, '00
2673.XSHE': 1716, '000725.XSHE': 3262, '000539.XSHE': 992.0, '60
0252.XSHG': 266, '600383.XSHG': 2968}
日期： 2013-12-12 00:00:00 ,持仓： {'002450.XSHE': 339, '000917.XS
HE': 308, '601628.XSHG': 752, '600674.XSHG': 672, '000783.XSHE':
 915, '600027.XSHG': 1324, '002202.XSHE': 485, '000712.XSHE': 35
7, '002008.XSHE': 1327, '002456.XSHE': 121, '000413.XSHE': 762,
'000895.XSHE': 229, '002673.XSHE': 1716, '000725.XSHE': 3262, '0
00539.XSHE': 992.0, '600252.XSHG': 266, '600383.XSHG': 2968}
日期： 2013-12-13 00:00:00 ,持仓： {'002450.XSHE': 339, '000917.XS
HE': 308, '000712.XSHE': 357, '000783.XSHE': 915, '002202.XSHE':
 485, '002008.XSHE': 1327, '002456.XSHE': 121, '000413.XSHE': 76
2, '000895.XSHE': 229, '002673.XSHE': 1716, '000725.XSHE': 3262,
 '000539.XSHE': 992.0, '600252.XSHG': 266, '600383.XSHG': 2968}
日期： 2013-12-16 00:00:00 ,持仓： {'002450.XSHE': 339, '000917.XS
HE': 308, '000783.XSHE': 915, '002008.XSHE': 1327, '002456.XSHE'
: 121, '000413.XSHE': 762, '000895.XSHE': 229, '002673.XSHE': 17
16, '000725.XSHE': 3262, '000539.XSHE': 992.0, '600252.XSHG': 26
6, '600383.XSHG': 2968}
日期： 2013-12-17 00:00:00 ,持仓： {'002450.XSHE': 339, '000917.XS
HE': 308, '000783.XSHE': 915, '002008.XSHE': 1327, '002456.XSHE'
: 121, '000413.XSHE': 762, '300017.XSHE': 558, '000895.XSHE': 22
9, '002673.XSHE': 1716, '000725.XSHE': 3262, '000539.XSHE': 992.
0, '600383.XSHG': 2968}
日期： 2013-12-18 00:00:00 ,持仓： {'002450.XSHE': 339, '000783.XS
HE': 915, '002008.XSHE': 1327, '002456.XSHE': 121, '000413.XSHE'
: 762, '300017.XSHE': 558, '000895.XSHE': 229, '002673.XSHE': 17
16, '000725.XSHE': 3262, '000539.XSHE': 992.0, '600383.XSHG': 29
68}
日期： 2013-12-19 00:00:00 ,持仓： {'002450.XSHE': 339, '000783.XS
HE': 915, '002008.XSHE': 1327, '002456.XSHE': 121, '000413.XSHE'
: 762, '300017.XSHE': 558, '000895.XSHE': 229, '002673.XSHE': 17
16, '000725.XSHE': 3262, '000539.XSHE': 992.0, '600383.XSHG': 29
68}
日期： 2013-12-20 00:00:00 ,持仓： {'002450.XSHE': 339, '000783.XS
HE': 915, '002008.XSHE': 1327, '002456.XSHE': 121, '000413.XSHE'
: 762, '300017.XSHE': 558, '000895.XSHE': 229, '002673.XSHE': 17
16, '000725.XSHE': 3262, '000539.XSHE': 992.0, '600383.XSHG': 29
68}
日期： 2013-12-23 00:00:00 ,持仓： {'002450.XSHE': 339, '000783.XS
HE': 915, '002008.XSHE': 1327, '002456.XSHE': 121, '000413.XSHE'
: 762, '300017.XSHE': 558, '000895.XSHE': 229, '002673.XSHE': 17
16, '000725.XSHE': 3262, '000539.XSHE': 992.0, '600383.XSHG': 29
68}
```

```
日期： 2013-12-24 00:00:00 ,持仓： {'002450.XSHE': 339, '000783.XS
HE': 915, '002008.XSHE': 1327, '002456.XSHE': 121, '000413.XSHE'
: 762, '300017.XSHE': 558, '000895.XSHE': 229, '002673.XSHE': 17
16, '000725.XSHE': 3262, '000539.XSHE': 992.0, '600383.XSHG': 29
68}
日期： 2013-12-25 00:00:00 ,持仓： {'002450.XSHE': 339, '000783.XS
HE': 915, '002008.XSHE': 1327, '002456.XSHE': 121, '000413.XSHE'
: 762, '300017.XSHE': 558, '000895.XSHE': 229, '002673.XSHE': 17
16, '000725.XSHE': 3262, '000539.XSHE': 992.0, '600383.XSHG': 29
68}
日期： 2013-12-26 00:00:00 ,持仓： {'002450.XSHE': 339, '000783.XS
HE': 915, '002008.XSHE': 1327, '002456.XSHE': 121, '000413.XSHE'
: 762, '300017.XSHE': 558, '000895.XSHE': 229, '002673.XSHE': 17
16, '000725.XSHE': 3262, '000539.XSHE': 992.0, '600383.XSHG': 29
68}
日期： 2013-12-27 00:00:00 ,持仓： {'002450.XSHE': 339, '000783.XS
HE': 915, '002008.XSHE': 1327, '002456.XSHE': 121, '000413.XSHE'
: 762, '300017.XSHE': 558, '000895.XSHE': 229, '002673.XSHE': 17
16, '000725.XSHE': 3262, '000539.XSHE': 992.0, '600252.XSHG': 20
01, '600383.XSHG': 2968}
日期： 2013-12-30 00:00:00 ,持仓： {'002450.XSHE': 339, '000917.XS
HE': 434, '000783.XSHE': 915, '600352.XSHG': 1137, '002008.XSHE'
: 1327, '000046.XSHE': 1569, '002456.XSHE': 121, '000413.XSHE':
762, '300017.XSHE': 558, '000895.XSHE': 229, '002673.XSHE': 1716
, '000725.XSHE': 3262, '000539.XSHE': 992.0, '600252.XSHG': 2001
, '600383.XSHG': 2968}
日期： 2013-12-31 00:00:00 ,持仓： {'002450.XSHE': 339, '000917.XS
HE': 434, '601628.XSHG': 189, '000783.XSHE': 915, '600352.XSHG':
 1137, '002008.XSHE': 1327, '000625.XSHE': 244, '000046.XSHE': 1
569, '002456.XSHE': 121, '000413.XSHE': 762, '300017.XSHE': 558,
 '000895.XSHE': 229, '002673.XSHE': 1716, '000725.XSHE': 3262, '
000539.XSHE': 992.0, '600252.XSHG': 2001, '600383.XSHG': 2968}
日期： 2014-01-02 00:00:00 ,持仓： {'002456.XSHE': 121, '000413.XS
HE': 762, '601099.XSHG': 368, '000895.XSHE': 229, '000539.XSHE':
 992.0, '600252.XSHG': 2001, '002450.XSHE': 339, '000783.XSHE':
915, '002202.XSHE': 179, '600369.XSHG': 295, '002008.XSHE': 1327
, '000686.XSHE': 183, '000725.XSHE': 3262, '000917.XSHE': 434, '
600352.XSHG': 1137, '000046.XSHE': 1569, '002673.XSHE': 1716, '6
00383.XSHG': 2968, '000009.XSHE': 183, '601628.XSHG': 189, '0007
12.XSHE': 129, '000625.XSHE': 244, '300017.XSHE': 558}
日期： 2014-01-03 00:00:00 ,持仓： {'002456.XSHE': 121, '000413.XS
HE': 762, '601099.XSHG': 368, '000895.XSHE': 229, '000539.XSHE':
 992.0, '600252.XSHG': 2001, '002450.XSHE': 339, '000783.XSHE':
915, '002202.XSHE': 179, '600369.XSHG': 295, '002008.XSHE': 1327
, '000686.XSHE': 183, '000725.XSHE': 3262, '000917.XSHE': 434, '
600352.XSHG': 1137, '000046.XSHE': 1569, '002673.XSHE': 1716, '6
00383.XSHG': 2968, '000009.XSHE': 183, '601628.XSHG': 189, '0007
12.XSHE': 129, '000625.XSHE': 244, '300017.XSHE': 558}
日期： 2014-01-06 00:00:00 ,持仓： {'002456.XSHE': 121, '000413.XS
HE': 762, '601099.XSHG': 368, '000895.XSHE': 229, '000539.XSHE':
 992.0, '600252.XSHG': 2001, '002450.XSHE': 339, '000783.XSHE':
915, '002202.XSHE': 179, '600369.XSHG': 295, '002008.XSHE': 1327
, '000686.XSHE': 183, '000725.XSHE': 3262, '000917.XSHE': 434, '
```

```
600352.XSHG': 1137, '000046.XSHE': 1569, '002673.XSHE': 1716, '6
00383.XSHG': 2968, '000009.XSHE': 183, '601628.XSHG': 189, '0007
12.XSHE': 129, '000625.XSHE': 244, '300017.XSHE': 558}
日期： 2014-01-07 00:00:00 ,持仓： {'002456.XSHE': 121, '000413.XS
HE': 762, '601099.XSHG': 368, '000895.XSHE': 229, '000539.XSHE':
 992.0, '600252.XSHG': 2001, '002450.XSHE': 339, '000783.XSHE':
915, '002202.XSHE': 179, '600369.XSHG': 295, '002008.XSHE': 1327
, '000686.XSHE': 183, '000725.XSHE': 3262, '000917.XSHE': 434, '
600352.XSHG': 1137, '002673.XSHE': 1716, '600383.XSHG': 2968, '0
00009.XSHE': 183, '601628.XSHG': 189, '000712.XSHE': 129, '00062
5.XSHE': 244, '300017.XSHE': 558}
日期： 2014-01-08 00:00:00 ,持仓： {'002450.XSHE': 339, '000009.XS
HE': 183, '300017.XSHE': 558, '000712.XSHE': 129, '000783.XSHE':
 915, '002202.XSHE': 179, '002673.XSHE': 1716, '002008.XSHE': 13
27, '000686.XSHE': 183, '000625.XSHE': 244, '002456.XSHE': 121,
'000413.XSHE': 762, '601099.XSHG': 368, '000895.XSHE': 229, '600
352.XSHG': 1137, '000725.XSHE': 3262, '000539.XSHE': 992.0, '600
252.XSHG': 2001, '600383.XSHG': 2968}
日期： 2014-01-09 00:00:00 ,持仓： {'002450.XSHE': 339, '300017.XS
HE': 558, '000783.XSHE': 915, '002202.XSHE': 179, '002008.XSHE':
 1327, '000686.XSHE': 183, '000625.XSHE': 244, '002456.XSHE': 12
1, '000413.XSHE': 762, '601099.XSHG': 368, '000895.XSHE': 229, '
002673.XSHE': 1716, '000725.XSHE': 3262, '000539.XSHE': 992.0, '
600252.XSHG': 2001, '600383.XSHG': 2968}
日期： 2014-01-10 00:00:00 ,持仓： {'002450.XSHE': 339, '300017.XS
HE': 558, '000783.XSHE': 915, '002202.XSHE': 179, '002008.XSHE':
 1327, '000625.XSHE': 244, '002456.XSHE': 121, '000413.XSHE': 76
2, '601099.XSHG': 368, '000895.XSHE': 229, '002673.XSHE': 1716,
'000725.XSHE': 3262, '000539.XSHE': 992.0, '600252.XSHG': 2001,
'600383.XSHG': 2968}
日期： 2014-01-13 00:00:00 ,持仓： {'002450.XSHE': 339, '000783.XS
HE': 915, '002202.XSHE': 179, '002008.XSHE': 1327, '000625.XSHE'
: 244, '002456.XSHE': 121, '000413.XSHE': 762, '300017.XSHE': 55
8, '000895.XSHE': 229, '002673.XSHE': 1716, '000725.XSHE': 3262,
 '000539.XSHE': 992.0, '600383.XSHG': 2968}
日期： 2014-01-14 00:00:00 ,持仓： {'002450.XSHE': 339, '600583.XS
HG': 962, '000783.XSHE': 915, '002008.XSHE': 1327, '000625.XSHE'
: 244, '002456.XSHE': 121, '000413.XSHE': 762, '300017.XSHE': 55
8, '000895.XSHE': 229, '002673.XSHE': 1716, '000725.XSHE': 3262,
 '000539.XSHE': 992.0, '600383.XSHG': 2968}
日期： 2014-01-15 00:00:00 ,持仓： {'002450.XSHE': 339, '600583.XS
HG': 962, '000783.XSHE': 915, '600648.XSHG': 196, '002008.XSHE':
 1327, '000625.XSHE': 244, '002456.XSHE': 121, '000413.XSHE': 76
2, '300017.XSHE': 558, '000895.XSHE': 229, '002673.XSHE': 1716,
'000725.XSHE': 3262, '000539.XSHE': 992.0, '600383.XSHG': 2968}
日期： 2014-01-16 00:00:00 ,持仓： {'002450.XSHE': 339, '600583.XS
HG': 962, '000783.XSHE': 915, '600648.XSHG': 196, '600663.XSHG':
 307, '002008.XSHE': 1327, '000625.XSHE': 244, '002456.XSHE': 12
1, '000413.XSHE': 762, '300017.XSHE': 558, '000895.XSHE': 229, '
002673.XSHE': 1716, '000725.XSHE': 3262, '000539.XSHE': 992.0, '
600383.XSHG': 2968}
日期： 2014-01-17 00:00:00 ,持仓： {'002450.XSHE': 339, '600583.XS
HG': 962, '600383.XSHG': 2968, '000783.XSHE': 915, '600648.XSHG'
```

```
: 196, '002202.XSHE': 482, '600663.XSHG': 307, '002008.XSHE': 13
27, '000625.XSHE': 244, '000046.XSHE': 981, '002456.XSHE': 121,
'000413.XSHE': 762, '300017.XSHE': 558, '000895.XSHE': 229, '002
673.XSHE': 1716, '000725.XSHE': 3262, '000539.XSHE': 992.0, '000
712.XSHE': 388, '600011.XSHG': 937}
日期： 2014-01-20 00:00:00 ,持仓： {'600583.XSHG': 962, '600027.XS
HG': 307, '600648.XSHG': 196, '002456.XSHE': 121, '000413.XSHE':
 762, '601099.XSHG': 200, '000895.XSHE': 229, '000539.XSHE': 992
.0, '002450.XSHE': 339, '600011.XSHG': 937, '600674.XSHG': 153,
'000783.XSHE': 915, '002202.XSHE': 482, '002008.XSHE': 1327, '00
0725.XSHE': 3262, '000046.XSHE': 981, '002673.XSHE': 1716, '6003
83.XSHG': 2968, '600663.XSHG': 307, '000712.XSHE': 388, '000625.
XSHE': 244, '300017.XSHE': 558}
日期： 2014-01-21 00:00:00 ,持仓： {'600583.XSHG': 962, '601601.XS
HG': 19, '600027.XSHG': 307, '600648.XSHG': 196, '002456.XSHE':
121, '000413.XSHE': 762, '601099.XSHG': 200, '000895.XSHE': 229,
 '000539.XSHE': 992.0, '002450.XSHE': 339, '600674.XSHG': 153, '
000783.XSHE': 915, '002202.XSHE': 482, '002008.XSHE': 1327, '000
725.XSHE': 3262, '000046.XSHE': 981, '002673.XSHE': 1716, '60038
3.XSHG': 2968, '600663.XSHG': 307, '000712.XSHE': 388, '000625.X
SHE': 244, '300017.XSHE': 558}
日期： 2014-01-22 00:00:00 ,持仓： {'002450.XSHE': 339, '300017.XS
HE': 558, '601628.XSHG': 77, '600583.XSHG': 962, '000712.XSHE':
388, '601601.XSHG': 19, '600027.XSHG': 307, '002202.XSHE': 482,
'000783.XSHE': 915, '002008.XSHE': 1327, '000625.XSHE': 244, '00
2456.XSHE': 121, '000413.XSHE': 762, '601099.XSHG': 200, '000895
.XSHE': 229, '002673.XSHE': 1716, '000725.XSHE': 3262, '000539.X
SHE': 992.0, '600383.XSHG': 2968}
日期： 2014-01-23 00:00:00 ,持仓： {'600583.XSHG': 962, '601601.XS
HG': 19, '600027.XSHG': 307, '002456.XSHE': 121, '000413.XSHE':
762, '601099.XSHG': 200, '000895.XSHE': 229, '000539.XSHE': 992.
0, '002450.XSHE': 339, '000783.XSHE': 915, '002202.XSHE': 482, '
600369.XSHG': 891, '002008.XSHE': 1327, '000686.XSHE': 521, '000
725.XSHE': 3262, '600352.XSHG': 712, '002673.XSHE': 1716, '60038
3.XSHG': 2968, '601628.XSHG': 77, '000712.XSHE': 388, '000625.XS
HE': 244, '300017.XSHE': 558}
日期： 2014-01-24 00:00:00 ,持仓： {'600583.XSHG': 962, '601601.XS
HG': 19, '600027.XSHG': 307, '002456.XSHE': 121, '000413.XSHE':
762, '601099.XSHG': 200, '000895.XSHE': 229, '000539.XSHE': 992.
0, '002450.XSHE': 339, '000783.XSHE': 915, '002202.XSHE': 482, '
600369.XSHG': 891, '002008.XSHE': 1327, '000686.XSHE': 521, '000
725.XSHE': 3262, '000917.XSHE': 107, '600352.XSHG': 712, '000046
.XSHE': 375, '002673.XSHE': 1716, '600383.XSHG': 2968, '601628.X
SHG': 77, '000712.XSHE': 388, '000625.XSHE': 244, '300017.XSHE':
 558}
日期： 2014-01-27 00:00:00 ,持仓： {'600583.XSHG': 962, '601601.XS
HG': 19, '600027.XSHG': 307, '600648.XSHG': 28, '002456.XSHE': 1
21, '000413.XSHE': 762, '601099.XSHG': 200, '000895.XSHE': 229,
'000539.XSHE': 992.0, '002450.XSHE': 339, '600011.XSHG': 213, '0
00783.XSHE': 915, '002202.XSHE': 482, '600369.XSHG': 891, '00200
8.XSHE': 1327, '000686.XSHE': 521, '000725.XSHE': 3262, '000917.
XSHE': 107, '600352.XSHG': 712, '000046.XSHE': 375, '002673.XSHE
': 1716, '600383.XSHG': 2968, '000009.XSHE': 128, '601628.XSHG':
 77, '600663.XSHG': 58, '000712.XSHE': 388, '000625.XSHE': 244,
'300017.XSHE': 558}
日期： 2014-01-28 00:00:00 ,持仓： {'600583.XSHG': 962, '600027.XS
HG': 307, '600648.XSHG': 28, '002456.XSHE': 121, '000413.XSHE':
762, '601099.XSHG': 200, '000895.XSHE': 229, '000539.XSHE': 992.
0, '600252.XSHG': 49, '002450.XSHE': 339, '600011.XSHG': 213, '0
00783.XSHE': 915, '002202.XSHE': 482, '600369.XSHG': 891, '00200
8.XSHE': 1327, '000686.XSHE': 521, '000725.XSHE': 3262, '000917.
XSHE': 107, '600352.XSHG': 712, '000046.XSHE': 375, '002673.XSHE
': 1716, '600383.XSHG': 2968, '000009.XSHE': 128, '600663.XSHG':
 58, '000712.XSHE': 388, '000625.XSHE': 244, '300017.XSHE': 558}
日期： 2014-01-29 00:00:00 ,持仓： {'600583.XSHG': 962, '600027.XS
HG': 307, '600648.XSHG': 28, '002456.XSHE': 121, '000413.XSHE':
762, '601099.XSHG': 200, '000895.XSHE': 229, '000539.XSHE': 992.
0, '600252.XSHG': 49, '002450.XSHE': 339, '600011.XSHG': 213, '0
00783.XSHE': 915, '002202.XSHE': 482, '600369.XSHG': 891, '00200
8.XSHE': 1327, '000686.XSHE': 521, '000725.XSHE': 3262, '000917.
XSHE': 107, '600352.XSHG': 712, '000046.XSHE': 375, '002673.XSHE
': 1716, '600383.XSHG': 2968, '000009.XSHE': 128, '600663.XSHG':
 58, '000712.XSHE': 388, '000625.XSHE': 244, '600886.XSHG': 113,
 '300017.XSHE': 558}
日期： 2014-01-30 00:00:00 ,持仓： {'600583.XSHG': 962, '600027.XS
HG': 307, '600648.XSHG': 28, '002456.XSHE': 121, '000413.XSHE':
762, '601099.XSHG': 200, '000895.XSHE': 229, '000539.XSHE': 992.
0, '600252.XSHG': 49, '002450.XSHE': 339, '600011.XSHG': 213, '6
00674.XSHG': 67, '000783.XSHE': 915, '002202.XSHE': 482, '600369
.XSHG': 891, '002008.XSHE': 1327, '000686.XSHE': 521, '000725.XS
HE': 3262, '000917.XSHE': 107, '600352.XSHG': 712, '000046.XSHE'
: 375, '002673.XSHE': 1716, '600383.XSHG': 2968, '000009.XSHE':
128, '600663.XSHG': 58, '000712.XSHE': 388, '000625.XSHE': 244,
'600886.XSHG': 113, '300017.XSHE': 558}
日期： 2014-02-07 00:00:00 ,持仓： {'600583.XSHG': 962, '600027.XS
HG': 307, '600648.XSHG': 28, '002456.XSHE': 121, '000413.XSHE':
762, '601099.XSHG': 200, '000895.XSHE': 229, '000539.XSHE': 992.
0, '600252.XSHG': 49, '002450.XSHE': 339, '600011.XSHG': 213, '6
00674.XSHG': 67, '000783.XSHE': 915, '002202.XSHE': 482, '600369
.XSHG': 891, '002008.XSHE': 1327, '000686.XSHE': 521, '000725.XS
HE': 3262, '000917.XSHE': 107, '600352.XSHG': 712, '000046.XSHE'
: 375, '002673.XSHE': 1716, '600383.XSHG': 2968, '000009.XSHE':
128, '600663.XSHG': 58, '000712.XSHE': 388, '000625.XSHE': 244,
'600886.XSHG': 113, '300017.XSHE': 558}
日期： 2014-02-10 00:00:00 ,持仓： {'600583.XSHG': 962, '600027.XS
HG': 307, '600648.XSHG': 28, '002456.XSHE': 121, '000413.XSHE':
762, '601099.XSHG': 200, '000895.XSHE': 229, '000539.XSHE': 992.
0, '600252.XSHG': 49, '002450.XSHE': 339, '600011.XSHG': 213, '6
00674.XSHG': 67, '000783.XSHE': 915, '002202.XSHE': 482, '002008
.XSHE': 1327, '000686.XSHE': 521, '000725.XSHE': 3262, '000917.X
SHE': 107, '600352.XSHG': 712, '002673.XSHE': 1716, '600383.XSHG
': 2968, '000009.XSHE': 128, '600663.XSHG': 58, '000712.XSHE': 3
88, '000625.XSHE': 244, '600886.XSHG': 113, '300017.XSHE': 558}
日期： 2014-02-11 00:00:00 ,持仓： {'600583.XSHG': 962, '600027.XS
HG': 307, '600648.XSHG': 28, '002456.XSHE': 121, '000413.XSHE':
762, '601099.XSHG': 200, '000895.XSHE': 229, '000539.XSHE': 992.
```

```
0, '600252.XSHG': 49, '002450.XSHE': 339, '600674.XSHG': 67, '00
0783.XSHE': 915, '002202.XSHE': 482, '002008.XSHE': 1327, '00068
6.XSHE': 521, '000725.XSHE': 3262, '000917.XSHE': 107, '002673.X
SHE': 1716, '600383.XSHG': 2968, '000009.XSHE': 128, '600663.XSH
G': 58, '000712.XSHE': 388, '600886.XSHG': 113, '300017.XSHE': 5
58}
日期： 2014-02-12 00:00:00 ,持仓： {'600583.XSHG': 962, '600027.XS
HG': 307, '600648.XSHG': 28, '002456.XSHE': 121, '000413.XSHE':
762, '601099.XSHG': 200, '000895.XSHE': 229, '000539.XSHE': 992.
0, '600252.XSHG': 49, '002450.XSHE': 339, '600674.XSHG': 67, '00
0783.XSHE': 915, '002202.XSHE': 482, '002008.XSHE': 1327, '00068
6.XSHE': 521, '000725.XSHE': 3262, '000917.XSHE': 107, '002673.X
SHE': 1716, '600383.XSHG': 2968, '000009.XSHE': 128, '600663.XSH
G': 58, '000712.XSHE': 388, '600886.XSHG': 113, '300017.XSHE': 5
58}
日期： 2014-02-13 00:00:00 ,持仓： {'600583.XSHG': 962, '600027.XS
HG': 307, '600648.XSHG': 28, '002456.XSHE': 121, '000413.XSHE':
762, '601099.XSHG': 200, '000895.XSHE': 229, '000539.XSHE': 992.
0, '600252.XSHG': 49, '002450.XSHE': 339, '600674.XSHG': 67, '00
0783.XSHE': 915, '002202.XSHE': 482, '002008.XSHE': 1327, '00068
6.XSHE': 521, '000725.XSHE': 3262, '000917.XSHE': 107, '600352.X
SHG': 492, '002673.XSHE': 1716, '600383.XSHG': 2968, '000009.XSH
E': 128, '600663.XSHG': 58, '000712.XSHE': 388, '600886.XSHG': 1
13, '300017.XSHE': 558}
日期： 2014-02-14 00:00:00 ,持仓： {'600583.XSHG': 962, '600027.XS
HG': 307, '600648.XSHG': 28, '002456.XSHE': 121, '000413.XSHE':
762, '601099.XSHG': 200, '000895.XSHE': 229, '000539.XSHE': 992.
0, '600252.XSHG': 49, '002450.XSHE': 339, '600674.XSHG': 67, '00
0783.XSHE': 915, '002202.XSHE': 482, '600369.XSHG': 494, '002008
.XSHE': 1327, '000686.XSHE': 521, '000725.XSHE': 3262, '000917.X
SHE': 107, '600352.XSHG': 492, '002673.XSHE': 1716, '600383.XSHG
': 2968, '000009.XSHE': 128, '600663.XSHG': 58, '000712.XSHE': 3
88, '600886.XSHG': 113, '300017.XSHE': 558}
日期： 2014-02-17 00:00:00 ,持仓： {'600583.XSHG': 962, '600027.XS
HG': 307, '600648.XSHG': 28, '002456.XSHE': 121, '000413.XSHE':
762, '000895.XSHE': 229, '000539.XSHE': 992.0, '600252.XSHG': 49
, '002450.XSHE': 339, '600011.XSHG': 416, '600674.XSHG': 67, '00
0783.XSHE': 915, '002202.XSHE': 482, '600369.XSHG': 494, '002008
.XSHE': 1327, '000686.XSHE': 521, '000725.XSHE': 3262, '000917.X
SHE': 107, '600352.XSHG': 492, '000046.XSHE': 431, '002673.XSHE'
: 1716, '600383.XSHG': 2968, '000009.XSHE': 128, '601628.XSHG':
136, '600663.XSHG': 58, '000712.XSHE': 388, '000625.XSHE': 170,
'600886.XSHG': 113, '300017.XSHE': 558}
日期： 2014-02-18 00:00:00 ,持仓： {'600583.XSHG': 962, '600027.XS
HG': 307, '600648.XSHG': 28, '002456.XSHE': 121, '000413.XSHE':
762, '000895.XSHE': 229, '000539.XSHE': 992.0, '600252.XSHG': 49
, '002450.XSHE': 339, '600011.XSHG': 416, '600674.XSHG': 67, '00
0783.XSHE': 915, '002202.XSHE': 482, '600369.XSHG': 494, '002008
.XSHE': 1327, '000686.XSHE': 521, '000725.XSHE': 3262, '000917.X
SHE': 107, '600352.XSHG': 492, '000046.XSHE': 431, '002673.XSHE'
: 1716, '600383.XSHG': 2968, '000009.XSHE': 128, '601628.XSHG':
136, '600663.XSHG': 58, '000712.XSHE': 388, '000625.XSHE': 170,
'600886.XSHG': 113, '300017.XSHE': 558}
```

```
日期： 2014-02-19 00:00:00 ,持仓： {'600583.XSHG': 962, '600648.XS
HG': 28, '002456.XSHE': 121, '000413.XSHE': 762, '000895.XSHE':
229, '000539.XSHE': 992.0, '600252.XSHG': 49, '002450.XSHE': 339
, '600011.XSHG': 416, '600674.XSHG': 67, '000783.XSHE': 915, '00
2202.XSHE': 482, '600369.XSHG': 494, '002008.XSHE': 1327, '00072
5.XSHE': 3262, '000917.XSHE': 107, '600352.XSHG': 492, '000046.X
SHE': 431, '002673.XSHE': 1716, '600383.XSHG': 2968, '000009.XSH
E': 128, '601628.XSHG': 136, '600663.XSHG': 58, '000712.XSHE': 3
88, '000625.XSHE': 170, '300017.XSHE': 558}
日期： 2014-02-20 00:00:00 ,持仓： {'002450.XSHE': 339, '000009.XS
HE': 128, '601628.XSHG': 136, '600011.XSHG': 416, '600674.XSHG':
 67, '000783.XSHE': 915, '600648.XSHG': 28, '002673.XSHE': 1716,
 '002202.XSHE': 482, '600663.XSHG': 58, '002008.XSHE': 1327, '00
0046.XSHE': 431, '002456.XSHE': 121, '000413.XSHE': 762, '000895
.XSHE': 229, '600352.XSHG': 492, '000725.XSHE': 3262, '000539.XS
HE': 992.0, '000712.XSHE': 388, '600252.XSHG': 49, '600383.XSHG'
: 2968}
日期： 2014-02-21 00:00:00 ,持仓： {'002450.XSHE': 339, '000009.XS
HE': 128, '601628.XSHG': 136, '600011.XSHG': 416, '600674.XSHG':
 67, '000783.XSHE': 915, '600648.XSHG': 28, '002673.XSHE': 1716,
 '600663.XSHG': 58, '002008.XSHE': 1327, '000046.XSHE': 431, '00
2456.XSHE': 121, '000413.XSHE': 762, '000895.XSHE': 229, '600352
.XSHG': 492, '000725.XSHE': 3262, '000539.XSHE': 992.0, '600886.
XSHG': 1951, '600252.XSHG': 49, '600383.XSHG': 2968}
日期： 2014-02-24 00:00:00 ,持仓： {'002450.XSHE': 339, '000009.XS
HE': 128, '601628.XSHG': 136, '600674.XSHG': 67, '000783.XSHE':
915, '600352.XSHG': 492, '002008.XSHE': 1327, '000046.XSHE': 431
, '002456.XSHE': 121, '000413.XSHE': 762, '300017.XSHE': 277, '0
00895.XSHE': 229, '002673.XSHE': 1716, '000725.XSHE': 3262, '000
539.XSHE': 992.0, '600886.XSHG': 1951, '600252.XSHG': 49, '60038
3.XSHG': 2968}
日期： 2014-02-25 00:00:00 ,持仓： {'002450.XSHE': 339, '000009.XS
HE': 128, '601628.XSHG': 136, '600674.XSHG': 67, '000783.XSHE':
915, '600352.XSHG': 492, '002008.XSHE': 1327, '000046.XSHE': 431
, '002456.XSHE': 121, '000413.XSHE': 762, '300017.XSHE': 277, '0
00895.XSHE': 229, '002673.XSHE': 1716, '000725.XSHE': 3262, '000
539.XSHE': 992.0, '600886.XSHG': 1951, '600383.XSHG': 2968}
日期： 2014-02-26 00:00:00 ,持仓： {'002450.XSHE': 339, '600674.XS
HG': 67, '000783.XSHE': 915, '600352.XSHG': 492, '002008.XSHE':
1327, '002456.XSHE': 121, '000413.XSHE': 762, '000895.XSHE': 229
, '002673.XSHE': 1716, '000725.XSHE': 3262, '000539.XSHE': 992.0
, '600383.XSHG': 2968}
日期： 2014-02-27 00:00:00 ,持仓： {'002450.XSHE': 339, '600674.XS
HG': 67, '000783.XSHE': 915, '600352.XSHG': 492, '002008.XSHE':
1327, '002456.XSHE': 121, '000413.XSHE': 762, '000895.XSHE': 229
, '002673.XSHE': 1716, '000725.XSHE': 3262, '000539.XSHE': 992.0
, '600383.XSHG': 2968}
日期： 2014-02-28 00:00:00 ,持仓： {'002450.XSHE': 339, '000712.XS
HE': 767, '000783.XSHE': 915, '002008.XSHE': 1327, '002456.XSHE'
: 121, '000413.XSHE': 762, '000895.XSHE': 229, '002673.XSHE': 17
16, '000725.XSHE': 3262, '000539.XSHE': 992.0, '600383.XSHG': 29
68}
日期： 2014-03-03 00:00:00 ,持仓： {'002450.XSHE': 339, '000712.XS
```

```
HE': 767, '000783.XSHE': 915, '002008.XSHE': 1327, '002456.XSHE'
: 121, '000413.XSHE': 762, '000895.XSHE': 229, '002673.XSHE': 17
16, '000725.XSHE': 3262, '000539.XSHE': 992.0, '600383.XSHG': 29
68}
日期： 2014-03-04 00:00:00 ,持仓： {'002450.XSHE': 339, '000712.XS
HE': 767, '000783.XSHE': 915, '002008.XSHE': 1327, '002456.XSHE'
: 121, '000413.XSHE': 762, '000895.XSHE': 229, '002673.XSHE': 17
16, '000725.XSHE': 3262, '000539.XSHE': 992.0, '600383.XSHG': 29
68}
日期： 2014-03-05 00:00:00 ,持仓： {'002450.XSHE': 339, '000009.XS
HE': 1072, '000712.XSHE': 767, '000783.XSHE': 915, '600027.XSHG'
: 3602, '002202.XSHE': 1012, '002008.XSHE': 1327, '002456.XSHE':
 121, '000413.XSHE': 762, '000895.XSHE': 229, '002673.XSHE': 171
6, '000725.XSHE': 3262, '000539.XSHE': 992.0, '600383.XSHG': 296
8}
日期： 2014-03-06 00:00:00 ,持仓： {'002450.XSHE': 339, '000009.XS
HE': 1072, '600011.XSHG': 906, '000712.XSHE': 767, '000783.XSHE'
: 915, '600027.XSHG': 3602, '002202.XSHE': 1012, '002008.XSHE':
1327, '002456.XSHE': 121, '000413.XSHE': 762, '000895.XSHE': 229
, '002673.XSHE': 1716, '000725.XSHE': 3262, '000539.XSHE': 992.0
, '600383.XSHG': 2968}
日期： 2014-03-07 00:00:00 ,持仓： {'002450.XSHE': 339, '000009.XS
HE': 1072, '601628.XSHG': 242, '600011.XSHG': 906, '000712.XSHE'
: 767, '000783.XSHE': 915, '600027.XSHG': 3602, '002202.XSHE': 1
012, '002008.XSHE': 1327, '000686.XSHE': 448, '600886.XSHG': 817
, '002456.XSHE': 121, '000413.XSHE': 762, '601099.XSHG': 664, '0
00895.XSHE': 229, '002673.XSHE': 1716, '000725.XSHE': 3262, '000
539.XSHE': 992.0, '600383.XSHG': 2968}
日期： 2014-03-10 00:00:00 ,持仓： {'600027.XSHG': 3602, '002456.X
SHE': 121, '000413.XSHE': 762, '601099.XSHG': 664, '000895.XSHE'
: 229, '000539.XSHE': 992.0, '002450.XSHE': 339, '600011.XSHG':
906, '000783.XSHE': 915, '002202.XSHE': 1012, '002008.XSHE': 132
7, '000725.XSHE': 3262, '000917.XSHE': 43, '600352.XSHG': 97, '0
00046.XSHE': 151, '002673.XSHE': 1716, '600383.XSHG': 2968, '000
009.XSHE': 1072, '601628.XSHG': 242, '000712.XSHE': 767, '000625
.XSHE': 70, '600886.XSHG': 817}
日期： 2014-03-11 00:00:00 ,持仓： {'002450.XSHE': 339, '000917.XS
HE': 43, '300017.XSHE': 29, '000009.XSHE': 1072, '000712.XSHE':
767, '000783.XSHE': 915, '600027.XSHG': 3602, '002202.XSHE': 101
2, '002673.XSHE': 1716, '002008.XSHE': 1327, '000625.XSHE': 70,
'000046.XSHE': 151, '002456.XSHE': 121, '000413.XSHE': 762, '601
099.XSHG': 664, '000895.XSHE': 229, '600352.XSHG': 97, '000725.X
SHE': 3262, '000539.XSHE': 992.0, '600886.XSHG': 817, '600383.XS
HG': 2968}
日期： 2014-03-12 00:00:00 ,持仓： {'002450.XSHE': 339, '000917.XS
HE': 43, '300017.XSHE': 29, '000712.XSHE': 767, '000783.XSHE': 9
15, '002202.XSHE': 1012, '002673.XSHE': 1716, '002008.XSHE': 132
7, '000625.XSHE': 70, '000046.XSHE': 151, '002456.XSHE': 121, '0
00413.XSHE': 762, '601099.XSHG': 664, '000895.XSHE': 229, '60035
2.XSHG': 97, '000725.XSHE': 3262, '000539.XSHE': 992.0, '600383.
XSHG': 2968}
日期： 2014-03-13 00:00:00 ,持仓： {'002450.XSHE': 339, '300017.XS
HE': 29, '000783.XSHE': 915, '002202.XSHE': 1012, '002673.XSHE':
 1716, '002008.XSHE': 1327, '000625.XSHE': 70, '000046.XSHE': 15
1, '002456.XSHE': 121, '000413.XSHE': 762, '601099.XSHG': 664, '
000895.XSHE': 229, '600352.XSHG': 97, '000725.XSHE': 3262, '0005
39.XSHE': 992.0, '600383.XSHG': 2968}
日期： 2014-03-14 00:00:00 ,持仓： {'002450.XSHE': 339, '300017.XS
HE': 29, '000783.XSHE': 915, '600352.XSHG': 97, '002008.XSHE': 1
327, '000625.XSHE': 70, '000046.XSHE': 151, '002456.XSHE': 121,
'000413.XSHE': 762, '601099.XSHG': 664, '000895.XSHE': 229, '002
673.XSHE': 1716, '000725.XSHE': 3262, '000539.XSHE': 992.0, '600
383.XSHG': 2968}
日期： 2014-03-17 00:00:00 ,持仓： {'002450.XSHE': 339, '000783.XS
HE': 915, '600352.XSHG': 97, '002008.XSHE': 1327, '000625.XSHE':
 70, '002456.XSHE': 121, '000413.XSHE': 762, '300017.XSHE': 29,
'000895.XSHE': 229, '002673.XSHE': 1716, '000725.XSHE': 3262, '0
00539.XSHE': 992.0, '600383.XSHG': 2968}
日期： 2014-03-18 00:00:00 ,持仓： {'002450.XSHE': 339, '600583.XS
HG': 1496, '600674.XSHG': 2204, '000783.XSHE': 915, '600352.XSHG
': 97, '002008.XSHE': 1327, '000686.XSHE': 1500, '000625.XSHE':
70, '600886.XSHG': 2867, '002456.XSHE': 121, '000413.XSHE': 762,
 '300017.XSHE': 29, '000895.XSHE': 229, '002673.XSHE': 1716, '00
0725.XSHE': 3262, '000539.XSHE': 992.0, '600383.XSHG': 2968}
日期： 2014-03-19 00:00:00 ,持仓： {'002450.XSHE': 339, '000009.XS
HE': 244, '600583.XSHG': 1496, '600674.XSHG': 2204, '000783.XSHE
': 915, '600352.XSHG': 97, '002008.XSHE': 1327, '000686.XSHE': 1
500, '000625.XSHE': 70, '000046.XSHE': 544, '002456.XSHE': 121,
'000413.XSHE': 762, '300017.XSHE': 29, '000895.XSHE': 229, '0026
73.XSHE': 1716, '000725.XSHE': 3262, '000539.XSHE': 992.0, '6008
86.XSHG': 2867, '600383.XSHG': 2968}
日期： 2014-03-20 00:00:00 ,持仓： {'002450.XSHE': 339, '000009.XS
HE': 244, '000625.XSHE': 70, '600583.XSHG': 1496, '600674.XSHG':
 2204, '000783.XSHE': 915, '600027.XSHG': 507, '600352.XSHG': 97
, '002008.XSHE': 1327, '000686.XSHE': 1500, '600867.XSHG': 129,
'000046.XSHE': 544, '002456.XSHE': 121, '000413.XSHE': 762, '300
017.XSHE': 29, '000895.XSHE': 229, '002673.XSHE': 1716, '000725.
XSHE': 3262, '000539.XSHE': 992.0, '600886.XSHG': 2867, '600383.
XSHG': 2968}
日期： 2014-03-21 00:00:00 ,持仓： {'600583.XSHG': 1496, '600027.X
SHG': 507, '002456.XSHE': 121, '000413.XSHE': 762, '000895.XSHE'
: 229, '000539.XSHE': 992.0, '002450.XSHE': 339, '600674.XSHG':
2204, '000783.XSHE': 915, '002202.XSHE': 83, '002008.XSHE': 1327
, '000686.XSHE': 1500, '000725.XSHE': 3262, '600352.XSHG': 97, '
600867.XSHG': 129, '000046.XSHE': 544, '002673.XSHE': 1716, '600
383.XSHG': 2968, '000009.XSHE': 244, '601628.XSHG': 64, '300027.
XSHE': 31, '600886.XSHG': 2867, '300017.XSHE': 29}
日期： 2014-03-24 00:00:00 ,持仓： {'600583.XSHG': 1496, '600027.X
SHG': 507, '002456.XSHE': 121, '000413.XSHE': 762, '000895.XSHE'
: 229, '000539.XSHE': 992.0, '002450.XSHE': 339, '600674.XSHG':
2204, '000783.XSHE': 915, '002202.XSHE': 83, '002008.XSHE': 1327
, '000686.XSHE': 1500, '002594.XSHE': 9, '000725.XSHE': 3262, '0
00917.XSHE': 33, '600352.XSHG': 97, '600867.XSHG': 129, '000046.
XSHE': 544, '002673.XSHE': 1716, '600383.XSHG': 2968, '000009.XS
HE': 244, '601628.XSHG': 64, '300027.XSHE': 31, '600886.XSHG': 2
867, '300017.XSHE': 29}
```

```
日期： 2014-03-25 00:00:00 ,持仓： {'601601.XSHG': 18, '600027.XSH
G': 507, '002456.XSHE': 121, '000413.XSHE': 762, '000895.XSHE':
229, '000539.XSHE': 992.0, '002450.XSHE': 339, '600674.XSHG': 22
04, '000783.XSHE': 915, '002202.XSHE': 83, '002008.XSHE': 1327,
'000686.XSHE': 1500, '002594.XSHE': 9, '000725.XSHE': 3262, '600
352.XSHG': 97, '600867.XSHG': 129, '000046.XSHE': 544, '002673.X
SHE': 1716, '600383.XSHG': 2968, '000009.XSHE': 244, '601628.XSH
G': 64, '300027.XSHE': 31, '600886.XSHG': 2867, '300017.XSHE': 2
9}
日期： 2014-03-26 00:00:00 ,持仓： {'601601.XSHG': 18, '600027.XSH
G': 507, '600648.XSHG': 90, '002456.XSHE': 121, '000413.XSHE': 7
62, '000895.XSHE': 229, '000539.XSHE': 992.0, '002450.XSHE': 339
, '600674.XSHG': 2204, '000783.XSHE': 915, '002202.XSHE': 83, '0
02008.XSHE': 1327, '000686.XSHE': 1500, '000725.XSHE': 3262, '60
0352.XSHG': 97, '600867.XSHG': 129, '000046.XSHE': 544, '002673.
XSHE': 1716, '600383.XSHG': 2968, '601628.XSHG': 64, '600886.XSH
G': 2867, '300017.XSHE': 29}
日期： 2014-03-27 00:00:00 ,持仓： {'601601.XSHG': 18, '600027.XSH
G': 507, '600648.XSHG': 90, '002456.XSHE': 121, '000413.XSHE': 7
62, '000895.XSHE': 229, '000539.XSHE': 992.0, '002450.XSHE': 339
, '600674.XSHG': 2204, '000783.XSHE': 915, '002202.XSHE': 83, '0
02008.XSHE': 1327, '000686.XSHE': 1500, '000725.XSHE': 3262, '60
0352.XSHG': 97, '600867.XSHG': 129, '000046.XSHE': 544, '002673.
XSHE': 1716, '600383.XSHG': 2968, '601628.XSHG': 64, '600886.XSH
G': 2867, '300017.XSHE': 29}
日期： 2014-03-28 00:00:00 ,持仓： {'601601.XSHG': 18, '600027.XSH
G': 507, '600648.XSHG': 90, '002456.XSHE': 121, '000413.XSHE': 7
62, '000895.XSHE': 229, '000539.XSHE': 992.0, '002450.XSHE': 339
, '600674.XSHG': 2204, '000783.XSHE': 915, '002202.XSHE': 83, '0
02008.XSHE': 1327, '000686.XSHE': 1500, '000725.XSHE': 3262, '60
0352.XSHG': 97, '600867.XSHG': 129, '000046.XSHE': 544, '002673.
XSHE': 1716, '600383.XSHG': 2968, '601628.XSHG': 64, '000712.XSH
E': 178, '600804.XSHG': 191, '600886.XSHG': 2867, '300017.XSHE':
 29}
日期： 2014-03-31 00:00:00 ,持仓： {'601601.XSHG': 18, '600027.XSH
G': 507, '600648.XSHG': 90, '002456.XSHE': 121, '000413.XSHE': 7
62, '000895.XSHE': 229, '000539.XSHE': 992.0, '002450.XSHE': 339
, '600674.XSHG': 2204, '000783.XSHE': 915, '002008.XSHE': 1327,
'000686.XSHE': 1500, '000725.XSHE': 3262, '600352.XSHG': 97, '60
0867.XSHG': 129, '000046.XSHE': 544, '002673.XSHE': 1716, '60038
3.XSHG': 2968, '601628.XSHG': 64, '000712.XSHE': 178, '600886.XS
HG': 2867, '300017.XSHE': 29}
日期： 2014-04-01 00:00:00 ,持仓： {'601601.XSHG': 18, '600027.XSH
G': 507, '600648.XSHG': 90, '002456.XSHE': 121, '000413.XSHE': 7
62, '000895.XSHE': 229, '000539.XSHE': 992.0, '002450.XSHE': 339
, '600674.XSHG': 2204, '000783.XSHE': 915, '002008.XSHE': 1327,
'000686.XSHE': 1500, '000725.XSHE': 3262, '600352.XSHG': 97, '60
0867.XSHG': 129, '000046.XSHE': 544, '002673.XSHE': 1716, '60038
3.XSHG': 2968, '601628.XSHG': 64, '000712.XSHE': 178, '600804.XS
HG': 172, '000625.XSHE': 249, '600886.XSHG': 2867, '300017.XSHE'
: 29}
日期： 2014-04-02 00:00:00 ,持仓： {'601601.XSHG': 18, '600027.XSH
G': 507, '002456.XSHE': 121, '000413.XSHE': 762, '000895.XSHE':
```

```
229, '000539.XSHE': 992.0, '002450.XSHE': 339, '600674.XSHG': 22
04, '000783.XSHE': 915, '002008.XSHE': 1327, '000686.XSHE': 1500
, '000725.XSHE': 3262, '600352.XSHG': 97, '600867.XSHG': 129, '0
00046.XSHE': 544, '002673.XSHE': 1716, '600383.XSHG': 2968, '601
628.XSHG': 64, '000712.XSHE': 178, '000625.XSHE': 249, '600886.X
SHG': 2867, '300017.XSHE': 29}
日期： 2014-04-03 00:00:00 ,持仓： {'002450.XSHE': 339, '601628.XS
HG': 64, '600674.XSHG': 2204, '601601.XSHG': 18, '600027.XSHG':
507, '600352.XSHG': 97, '000783.XSHE': 915, '000712.XSHE': 178,
'002008.XSHE': 1327, '000686.XSHE': 1500, '000625.XSHE': 249, '0
00046.XSHE': 544, '002456.XSHE': 121, '000413.XSHE': 762, '30001
7.XSHE': 29, '000895.XSHE': 229, '002673.XSHE': 1716, '000725.XS
HE': 3262, '000539.XSHE': 992.0, '600886.XSHG': 2867, '600383.XS
HG': 2968}
日期： 2014-04-04 00:00:00 ,持仓： {'002450.XSHE': 339, '601628.XS
HG': 64, '600674.XSHG': 2204, '601601.XSHG': 18, '600027.XSHG':
507, '600352.XSHG': 97, '000783.XSHE': 915, '000712.XSHE': 178,
'002008.XSHE': 1327, '000625.XSHE': 249, '000046.XSHE': 544, '00
2456.XSHE': 121, '000413.XSHE': 762, '300017.XSHE': 29, '000895.
XSHE': 229, '002673.XSHE': 1716, '000725.XSHE': 3262, '000539.XS
HE': 992.0, '600886.XSHG': 2867, '600383.XSHG': 2968}
日期： 2014-04-08 00:00:00 ,持仓： {'002450.XSHE': 339, '601628.XS
HG': 64, '600674.XSHG': 2204, '601601.XSHG': 18, '600027.XSHG':
507, '600352.XSHG': 97, '000783.XSHE': 915, '000712.XSHE': 178,
'002008.XSHE': 1327, '000686.XSHE': 641, '000625.XSHE': 249, '00
0046.XSHE': 544, '002456.XSHE': 121, '000413.XSHE': 762, '300017
.XSHE': 29, '000895.XSHE': 229, '002673.XSHE': 1716, '000725.XSH
E': 3262, '000539.XSHE': 992.0, '600886.XSHG': 2867, '600383.XSH
G': 2968}
日期： 2014-04-09 00:00:00 ,持仓： {'600583.XSHG': 531, '601601.XS
HG': 18, '600027.XSHG': 507, '002456.XSHE': 121, '000413.XSHE':
762, '000895.XSHE': 229, '000539.XSHE': 992.0, '002450.XSHE': 33
9, '600674.XSHG': 2204, '000783.XSHE': 915, '002008.XSHE': 1327,
 '000686.XSHE': 641, '000725.XSHE': 3262, '600352.XSHG': 97, '00
0046.XSHE': 544, '002673.XSHE': 1716, '600383.XSHG': 2968, '6016
28.XSHG': 64, '000712.XSHE': 178, '000625.XSHE': 249, '600886.XS
HG': 2867, '300017.XSHE': 29}
日期： 2014-04-10 00:00:00 ,持仓： {'600583.XSHG': 531, '601601.XS
HG': 18, '600027.XSHG': 507, '002456.XSHE': 121, '000413.XSHE':
762, '601099.XSHG': 734, '000895.XSHE': 229, '000539.XSHE': 992.
0, '002450.XSHE': 339, '600674.XSHG': 2204, '000783.XSHE': 915,
'002008.XSHE': 1327, '000686.XSHE': 641, '002594.XSHE': 64, '000
725.XSHE': 3262, '000917.XSHE': 231, '600352.XSHG': 97, '000046.
XSHE': 544, '002673.XSHE': 1716, '600383.XSHG': 2968, '000009.XS
HE': 358, '601628.XSHG': 64, '000712.XSHE': 178, '000625.XSHE':
249, '600886.XSHG': 2867, '300017.XSHE': 29}
日期： 2014-04-11 00:00:00 ,持仓： {'600583.XSHG': 531, '601601.XS
HG': 18, '600027.XSHG': 507, '600648.XSHG': 22, '002456.XSHE': 1
21, '000413.XSHE': 762, '601099.XSHG': 734, '000895.XSHE': 229,
'000539.XSHE': 992.0, '002450.XSHE': 339, '600674.XSHG': 2204, '
000783.XSHE': 915, '002008.XSHE': 1327, '000686.XSHE': 641, '002
594.XSHE': 64, '000725.XSHE': 3262, '000917.XSHE': 231, '600352.
XSHG': 97, '601179.XSHG': 177, '600867.XSHG': 61, '000046.XSHE':
 544, '002673.XSHE': 1716, '600383.XSHG': 2968, '000009.XSHE': 3
58, '601628.XSHG': 64, '300027.XSHE': 26, '000712.XSHE': 178, '6
00804.XSHG': 46, '000625.XSHE': 249, '600886.XSHG': 2867, '30001
7.XSHE': 29}
日期： 2014-04-14 00:00:00 ,持仓： {'600583.XSHG': 531, '601601.XS
HG': 18, '600027.XSHG': 507, '600648.XSHG': 22, '002456.XSHE': 1
21, '000413.XSHE': 762, '601099.XSHG': 734, '000895.XSHE': 229,
'000539.XSHE': 992.0, '002450.XSHE': 339, '600674.XSHG': 2204, '
000783.XSHE': 915, '002202.XSHE': 1, '002008.XSHE': 1327, '00068
6.XSHE': 641, '002594.XSHE': 64, '000725.XSHE': 3262, '000917.XS
HE': 231, '600352.XSHG': 97, '601179.XSHG': 177, '600867.XSHG':
61, '000046.XSHE': 544, '002673.XSHE': 1716, '600383.XSHG': 2968
, '000009.XSHE': 358, '601628.XSHG': 64, '300027.XSHE': 26, '000
712.XSHE': 178, '600804.XSHG': 46, '000625.XSHE': 249, '600886.X
SHG': 2867, '300017.XSHE': 29}
日期： 2014-04-15 00:00:00 ,持仓： {'600583.XSHG': 531, '601601.XS
HG': 18, '600027.XSHG': 507, '600648.XSHG': 22, '002456.XSHE': 1
21, '000413.XSHE': 762, '601099.XSHG': 734, '000895.XSHE': 229,
'000539.XSHE': 992.0, '002450.XSHE': 339, '600674.XSHG': 2204, '
000783.XSHE': 915, '002202.XSHE': 1, '002008.XSHE': 1327, '00068
6.XSHE': 641, '002594.XSHE': 64, '000725.XSHE': 3262, '000917.XS
HE': 231, '600352.XSHG': 97, '601179.XSHG': 177, '600867.XSHG':
61, '000046.XSHE': 544, '002673.XSHE': 1716, '600383.XSHG': 2968
, '000009.XSHE': 358, '601628.XSHG': 64, '300027.XSHE': 26, '000
712.XSHE': 178, '600804.XSHG': 46, '000625.XSHE': 249, '600886.X
SHG': 2867, '300017.XSHE': 29}
日期： 2014-04-16 00:00:00 ,持仓： {'600583.XSHG': 531, '601601.XS
HG': 18, '600027.XSHG': 507, '600648.XSHG': 22, '002456.XSHE': 1
21, '000413.XSHE': 762, '601099.XSHG': 734, '000895.XSHE': 229,
'000539.XSHE': 992.0, '002450.XSHE': 339, '600674.XSHG': 2204, '
000783.XSHE': 915, '002202.XSHE': 1, '002008.XSHE': 1327, '00068
6.XSHE': 641, '002594.XSHE': 64, '000725.XSHE': 3262, '000917.XS
HE': 231, '600352.XSHG': 97, '601179.XSHG': 177, '600867.XSHG':
61, '000046.XSHE': 544, '002673.XSHE': 1716, '600383.XSHG': 2968
, '000009.XSHE': 358, '601628.XSHG': 64, '300027.XSHE': 26, '000
712.XSHE': 178, '600804.XSHG': 46, '000625.XSHE': 249, '600886.X
SHG': 2867, '300017.XSHE': 29}
日期： 2014-04-17 00:00:00 ,持仓： {'600583.XSHG': 531, '601601.XS
HG': 18, '600027.XSHG': 507, '600648.XSHG': 22, '002456.XSHE': 1
21, '000413.XSHE': 762, '601099.XSHG': 734, '000895.XSHE': 229,
'000539.XSHE': 992.0, '002450.XSHE': 339, '600674.XSHG': 2204, '
000783.XSHE': 915, '002008.XSHE': 1327, '002594.XSHE': 64, '0007
25.XSHE': 3262, '000917.XSHE': 231, '600352.XSHG': 97, '601179.X
SHG': 177, '600867.XSHG': 61, '000046.XSHE': 544, '002673.XSHE':
 1716, '600383.XSHG': 2968, '000009.XSHE': 358, '601628.XSHG': 6
4, '300027.XSHE': 26, '000712.XSHE': 178, '600804.XSHG': 46, '00
0625.XSHE': 249, '600886.XSHG': 2867, '300017.XSHE': 29}
日期： 2014-04-18 00:00:00 ,持仓： {'601601.XSHG': 18, '600027.XSH
G': 507, '600648.XSHG': 22, '002456.XSHE': 121, '000413.XSHE': 7
62, '601099.XSHG': 734, '000895.XSHE': 229, '000539.XSHE': 992.0
, '002450.XSHE': 339, '600674.XSHG': 2204, '000783.XSHE': 915, '
002008.XSHE': 1327, '000725.XSHE': 3262, '000917.XSHE': 231, '60
0352.XSHG': 97, '601179.XSHG': 177, '600867.XSHG': 61, '000046.X
```

```
SHE': 544, '002673.XSHE': 1716, '600383.XSHG': 2968, '000009.XSH
E': 358, '601628.XSHG': 64, '000712.XSHE': 178, '600804.XSHG': 4
6, '000625.XSHE': 249, '600886.XSHG': 2867}
日期： 2014-04-21 00:00:00 ,持仓： {'601601.XSHG': 18, '600027.XSH
G': 507, '600648.XSHG': 22, '002456.XSHE': 121, '000413.XSHE': 7
62, '601099.XSHG': 734, '000895.XSHE': 229, '000539.XSHE': 992.0
, '002450.XSHE': 339, '600674.XSHG': 2204, '000783.XSHE': 915, '
002008.XSHE': 1327, '000725.XSHE': 3262, '000917.XSHE': 231, '60
0352.XSHG': 97, '601179.XSHG': 177, '600867.XSHG': 61, '000046.X
SHE': 544, '002673.XSHE': 1716, '600383.XSHG': 2968, '000712.XSH
E': 178, '600804.XSHG': 46, '000625.XSHE': 249, '300017.XSHE': 1
13}
日期： 2014-04-22 00:00:00 ,持仓： {'002450.XSHE': 339, '000917.XS
HE': 231, '300017.XSHE': 113, '000625.XSHE': 249, '000712.XSHE':
 178, '601601.XSHG': 18, '600648.XSHG': 22, '002673.XSHE': 1716,
 '000783.XSHE': 915, '002008.XSHE': 1327, '600804.XSHG': 46, '60
0867.XSHG': 61, '000046.XSHE': 544, '002456.XSHE': 121, '000413.
XSHE': 762, '601099.XSHG': 734, '000895.XSHE': 229, '600352.XSHG
': 97, '000725.XSHE': 3262, '000539.XSHE': 992.0, '600383.XSHG':
 2968}
日期： 2014-04-23 00:00:00 ,持仓： {'002450.XSHE': 339, '000917.XS
HE': 231, '300017.XSHE': 113, '000712.XSHE': 178, '601601.XSHG':
 18, '600648.XSHG': 22, '002673.XSHE': 1716, '000783.XSHE': 915,
 '002008.XSHE': 1327, '600804.XSHG': 46, '000625.XSHE': 249, '00
0046.XSHE': 544, '002456.XSHE': 121, '000413.XSHE': 762, '601099
.XSHG': 734, '000895.XSHE': 229, '600352.XSHG': 97, '000725.XSHE
': 3262, '000539.XSHE': 992.0, '600383.XSHG': 2968}
日期： 2014-04-24 00:00:00 ,持仓： {'002450.XSHE': 339, '000917.XS
HE': 231, '000712.XSHE': 178, '601601.XSHG': 18, '600648.XSHG':
22, '002673.XSHE': 1716, '000783.XSHE': 915, '002008.XSHE': 1327
, '600804.XSHG': 46, '000625.XSHE': 249, '000046.XSHE': 544, '00
2456.XSHE': 121, '000413.XSHE': 762, '300017.XSHE': 113, '000895
.XSHE': 229, '600352.XSHG': 97, '000725.XSHE': 3262, '000539.XSH
E': 992.0, '600383.XSHG': 2968}
日期： 2014-04-25 00:00:00 ,持仓： {'002450.XSHE': 339, '000917.XS
HE': 231, '600674.XSHG': 1602, '601601.XSHG': 18, '600648.XSHG':
 22, '002673.XSHE': 1716, '000783.XSHE': 915, '000712.XSHE': 178
, '002008.XSHE': 1327, '600804.XSHG': 46, '000625.XSHE': 249, '0
00046.XSHE': 544, '002456.XSHE': 121, '000413.XSHE': 762, '30001
7.XSHE': 113, '000895.XSHE': 229, '600352.XSHG': 97, '000725.XSH
E': 3262, '000539.XSHE': 992.0, '600383.XSHG': 2968}
日期： 2014-04-28 00:00:00 ,持仓： {'002450.XSHE': 339, '000917.XS
HE': 231, '000712.XSHE': 178, '601601.XSHG': 18, '600648.XSHG':
22, '002673.XSHE': 1716, '000783.XSHE': 915, '002008.XSHE': 1327
, '600804.XSHG': 46, '000046.XSHE': 544, '002456.XSHE': 121, '00
0413.XSHE': 762, '300017.XSHE': 113, '000895.XSHE': 229, '600352
.XSHG': 97, '000725.XSHE': 3262, '000539.XSHE': 992.0, '600383.X
SHG': 2968}
日期： 2014-04-29 00:00:00 ,持仓： {'002450.XSHE': 339, '000917.XS
HE': 231, '000712.XSHE': 178, '601601.XSHG': 18, '600648.XSHG':
22, '000783.XSHE': 915, '002008.XSHE': 1327, '600804.XSHG': 46,
'000046.XSHE': 544, '002456.XSHE': 121, '000413.XSHE': 762, '300
017.XSHE': 113, '000895.XSHE': 229, '002673.XSHE': 1716, '000725
```

```
.XSHE': 3262, '000539.XSHE': 992.0, '600383.XSHG': 2968}
日期： 2014-04-30 00:00:00 ,持仓： {'002450.XSHE': 339, '000917.XS
HE': 231, '000712.XSHE': 178, '601601.XSHG': 18, '002202.XSHE':
1162, '000783.XSHE': 915, '002008.XSHE': 1327, '600804.XSHG': 46
, '000046.XSHE': 544, '002456.XSHE': 121, '000413.XSHE': 762, '3
00017.XSHE': 113, '000895.XSHE': 229, '002673.XSHE': 1716, '0007
25.XSHE': 3262, '000539.XSHE': 992.0, '600383.XSHG': 2968}
日期： 2014-05-05 00:00:00 ,持仓： {'002450.XSHE': 339, '000917.XS
HE': 231, '600369.XSHG': 1811, '000712.XSHE': 178, '601601.XSHG'
: 18, '002202.XSHE': 1162, '000783.XSHE': 915, '002008.XSHE': 13
27, '600804.XSHG': 46, '000046.XSHE': 544, '002456.XSHE': 121, '
000413.XSHE': 762, '300017.XSHE': 113, '000895.XSHE': 229, '0026
73.XSHE': 1716, '000725.XSHE': 3262, '000539.XSHE': 992.0, '6003
83.XSHG': 2968}
日期： 2014-05-06 00:00:00 ,持仓： {'002450.XSHE': 339, '000917.XS
HE': 231, '600369.XSHG': 1811, '000712.XSHE': 178, '601601.XSHG'
: 18, '002202.XSHE': 1162, '000728.XSHE': 686, '000783.XSHE': 91
5, '002008.XSHE': 1327, '600804.XSHG': 46, '000625.XSHE': 554, '
000046.XSHE': 544, '002456.XSHE': 121, '000413.XSHE': 762, '3000
17.XSHE': 113, '000895.XSHE': 229, '002673.XSHE': 1716, '000725.
XSHE': 3262, '000539.XSHE': 992.0, '600886.XSHG': 1406, '600383.
XSHG': 2968}
日期： 2014-05-07 00:00:00 ,持仓： {'601601.XSHG': 18, '002456.XSH
E': 121, '000413.XSHE': 762, '000895.XSHE': 229, '000539.XSHE':
992.0, '000728.XSHE': 686, '002450.XSHE': 339, '600887.XSHG': 21
5, '000783.XSHE': 915, '002202.XSHE': 1162, '600369.XSHG': 1811,
 '002008.XSHE': 1327, '000725.XSHE': 3262, '000917.XSHE': 231, '
000046.XSHE': 544, '002673.XSHE': 1716, '600383.XSHG': 2968, '00
0712.XSHE': 178, '600804.XSHG': 46, '000625.XSHE': 554, '600886.
XSHG': 1406, '300017.XSHE': 113}
日期： 2014-05-08 00:00:00 ,持仓： {'601601.XSHG': 18, '600027.XSH
G': 721, '002456.XSHE': 121, '000413.XSHE': 762, '000895.XSHE':
229, '000539.XSHE': 992.0, '000728.XSHE': 686, '002450.XSHE': 33
9, '600887.XSHG': 215, '000783.XSHE': 915, '002202.XSHE': 1162,
'600369.XSHG': 1811, '002008.XSHE': 1327, '000725.XSHE': 3262, '
000917.XSHE': 231, '000046.XSHE': 544, '002673.XSHE': 1716, '600
383.XSHG': 2968, '600674.XSHG': 361, '000712.XSHE': 178, '600804
.XSHG': 46, '000625.XSHE': 554, '600886.XSHG': 1406, '300017.XSH
E': 113, '300058.XSHE': 152}
日期： 2014-05-09 00:00:00 ,持仓： {'601601.XSHG': 18, '600027.XSH
G': 721, '002456.XSHE': 121, '000413.XSHE': 762, '000895.XSHE':
229, '000539.XSHE': 992.0, '002450.XSHE': 339, '600887.XSHG': 21
5, '000783.XSHE': 915, '002202.XSHE': 1162, '600369.XSHG': 1811,
 '002008.XSHE': 1327, '600649.XSHG': 133, '000725.XSHE': 3262, '
000917.XSHE': 231, '000046.XSHE': 544, '002673.XSHE': 1716, '600
383.XSHG': 2968, '600633.XSHG': 68, '600674.XSHG': 361, '000728.
XSHE': 686, '000712.XSHE': 178, '600804.XSHG': 46, '000625.XSHE'
: 554, '600886.XSHG': 1406, '300017.XSHE': 113, '300058.XSHE': 1
52}
日期： 2014-05-12 00:00:00 ,持仓： {'601601.XSHG': 18, '600027.XSH
G': 721, '002456.XSHE': 121, '000413.XSHE': 762, '000895.XSHE':
229, '000728.XSHE': 686, '002450.XSHE': 339, '600887.XSHG': 215,
 '000783.XSHE': 915, '000539.XSHE': 992.0, '600369.XSHG': 1811,
'002008.XSHE': 1327, '000725.XSHE': 3262, '000917.XSHE': 231, '0
00046.XSHE': 544, '002673.XSHE': 1716, '600383.XSHG': 2968, '600
633.XSHG': 68, '600674.XSHG': 361, '000712.XSHE': 178, '600804.X
SHG': 46, '000625.XSHE': 554, '600886.XSHG': 1406, '300017.XSHE'
: 113, '300058.XSHE': 152}
日期： 2014-05-13 00:00:00 ,持仓： {'600583.XSHG': 367, '601601.XS
HG': 18, '600027.XSHG': 721, '600111.XSHG': 201, '002456.XSHE':
121, '000413.XSHE': 762, '000895.XSHE': 229, '000539.XSHE': 992.
0, '002450.XSHE': 339, '600674.XSHG': 361, '000783.XSHE': 915, '
000728.XSHE': 686, '600369.XSHG': 1811, '002008.XSHE': 1327, '00
0725.XSHE': 3262, '000917.XSHE': 231, '000046.XSHE': 544, '00267
3.XSHE': 1716, '600383.XSHG': 2968, '600633.XSHG': 68, '600887.X
SHG': 215, '300027.XSHE': 119, '600166.XSHG': 532, '000712.XSHE'
: 178, '600804.XSHG': 46, '600886.XSHG': 1406, '300017.XSHE': 11
3, '300058.XSHE': 152}
日期： 2014-05-14 00:00:00 ,持仓： {'600583.XSHG': 367, '601601.XS
HG': 18, '600027.XSHG': 721, '600111.XSHG': 201, '002456.XSHE':
121, '000413.XSHE': 762, '000895.XSHE': 229, '000728.XSHE': 686,
 '002450.XSHE': 339, '000783.XSHE': 915, '000539.XSHE': 992.0, '
600369.XSHG': 1811, '002008.XSHE': 1327, '000725.XSHE': 3262, '0
00917.XSHE': 231, '000046.XSHE': 544, '002673.XSHE': 1716, '6003
83.XSHG': 2968, '600633.XSHG': 68, '300027.XSHE': 119, '600166.X
SHG': 532, '000712.XSHE': 178, '600804.XSHG': 46, '300058.XSHE':
 152}
日期： 2014-05-15 00:00:00 ,持仓： {'600583.XSHG': 367, '601601.XS
HG': 18, '600027.XSHG': 721, '600111.XSHG': 201, '002456.XSHE':
121, '000413.XSHE': 762, '000895.XSHE': 229, '000728.XSHE': 686,
 '002450.XSHE': 339, '000783.XSHE': 915, '000539.XSHE': 992.0, '
600369.XSHG': 1811, '002008.XSHE': 1327, '000725.XSHE': 3262, '0
00917.XSHE': 231, '000046.XSHE': 544, '002673.XSHE': 1716, '6003
83.XSHG': 2968, '600633.XSHG': 68, '300027.XSHE': 119, '600166.X
SHG': 532, '000712.XSHE': 178, '600804.XSHG': 46, '300058.XSHE':
 152}
日期： 2014-05-16 00:00:00 ,持仓： {'600583.XSHG': 367, '601601.XS
HG': 18, '600027.XSHG': 721, '600111.XSHG': 201, '002456.XSHE':
121, '000413.XSHE': 762, '000895.XSHE': 229, '000728.XSHE': 686,
 '002450.XSHE': 339, '000783.XSHE': 915, '000539.XSHE': 992.0, '
600369.XSHG': 1811, '002008.XSHE': 1327, '000725.XSHE': 3262, '0
00917.XSHE': 231, '600352.XSHG': 548, '000046.XSHE': 544, '00267
3.XSHE': 1716, '600383.XSHG': 2968, '600633.XSHG': 68, '300027.X
SHE': 119, '600166.XSHG': 532, '000712.XSHE': 178, '600804.XSHG'
: 46}
日期： 2014-05-19 00:00:00 ,持仓： {'600583.XSHG': 367, '601601.XS
HG': 18, '600027.XSHG': 721, '600111.XSHG': 201, '002456.XSHE':
121, '000413.XSHE': 762, '000895.XSHE': 229, '000539.XSHE': 992.
0, '002450.XSHE': 339, '000783.XSHE': 915, '000728.XSHE': 686, '
600369.XSHG': 1811, '002008.XSHE': 1327, '600839.XSHG': 1287, '6
00649.XSHG': 643, '000725.XSHE': 3262, '000917.XSHE': 231, '6003
52.XSHG': 548, '000046.XSHE': 544, '002673.XSHE': 1716, '600383.
XSHG': 2968, '600633.XSHG': 68, '600166.XSHG': 532, '000712.XSHE
': 178, '600804.XSHG': 46}
日期： 2014-05-20 00:00:00 ,持仓： {'002450.XSHE': 339, '000917.XS
HE': 231, '600633.XSHG': 68, '000712.XSHE': 178, '601601.XSHG':
```

```
18, '600027.XSHG': 721, '000728.XSHE': 686, '000783.XSHE': 915,
'002008.XSHE': 1327, '600111.XSHG': 201, '600804.XSHG': 46, '000
046.XSHE': 544, '002456.XSHE': 121, '000413.XSHE': 762, '000895.
XSHE': 229, '002673.XSHE': 1716, '000539.XSHE': 992.0, '600383.X
SHG': 2968}
日期： 2014-05-21 00:00:00 ,持仓： {'002450.XSHE': 339, '000917.XS
HE': 231, '000712.XSHE': 178, '601601.XSHG': 18, '600027.XSHG':
721, '002202.XSHE': 1096, '002673.XSHE': 1716, '000783.XSHE': 91
5, '002008.XSHE': 1327, '600804.XSHG': 46, '000046.XSHE': 544, '
002456.XSHE': 121, '000413.XSHE': 762, '000895.XSHE': 229, '6003
52.XSHG': 1198, '000539.XSHE': 992.0, '600383.XSHG': 2968}
日期： 2014-05-22 00:00:00 ,持仓： {'002450.XSHE': 339, '000917.XS
HE': 231, '000712.XSHE': 178, '601601.XSHG': 18, '600027.XSHG':
721, '000783.XSHE': 915, '002008.XSHE': 1327, '600804.XSHG': 46,
 '000046.XSHE': 544, '002456.XSHE': 121, '000413.XSHE': 762, '00
0895.XSHE': 229, '002673.XSHE': 1716, '000539.XSHE': 992.0, '600
886.XSHG': 1673, '600383.XSHG': 2968}
日期： 2014-05-23 00:00:00 ,持仓： {'002450.XSHE': 339, '000917.XS
HE': 231, '000712.XSHE': 178, '601601.XSHG': 18, '600027.XSHG':
721, '000783.XSHE': 915, '002008.XSHE': 1327, '600804.XSHG': 46,
 '000046.XSHE': 544, '002456.XSHE': 121, '000413.XSHE': 762, '00
0895.XSHE': 229, '002673.XSHE': 1716, '000725.XSHE': 4514, '0005
39.XSHE': 992.0, '600886.XSHG': 1673, '600383.XSHG': 2968}
日期： 2014-05-26 00:00:00 ,持仓： {'002450.XSHE': 339, '000917.XS
HE': 231, '000712.XSHE': 178, '601601.XSHG': 18, '600027.XSHG':
721, '002202.XSHE': 913, '002673.XSHE': 1716, '000783.XSHE': 915
, '002008.XSHE': 1327, '600804.XSHG': 46, '000046.XSHE': 544, '0
02456.XSHE': 121, '000413.XSHE': 762, '000895.XSHE': 229, '60064
9.XSHG': 1232, '000539.XSHE': 992.0, '600886.XSHG': 1673, '60038
3.XSHG': 2968}
日期： 2014-05-27 00:00:00 ,持仓： {'601601.XSHG': 18, '600027.XSH
G': 721, '002456.XSHE': 121, '000413.XSHE': 762, '000895.XSHE':
229, '000539.XSHE': 992.0, '002450.XSHE': 339, '600674.XSHG': 97
0, '000783.XSHE': 915, '002202.XSHE': 913, '002008.XSHE': 1327,
'002594.XSHE': 131, '600649.XSHG': 1232, '000725.XSHE': 2559, '0
00917.XSHE': 231, '600352.XSHG': 688, '000046.XSHE': 544, '00267
3.XSHE': 1716, '600383.XSHG': 2968, '000712.XSHE': 178, '600804.
XSHG': 46, '000625.XSHE': 494, '600886.XSHG': 1673, '300058.XSHE
': 376.0}
日期： 2014-05-28 00:00:00 ,持仓： {'601601.XSHG': 18, '600027.XSH
G': 721, '002456.XSHE': 121, '000413.XSHE': 762, '000895.XSHE':
229, '000539.XSHE': 992.0, '002450.XSHE': 339, '600674.XSHG': 97
0, '000783.XSHE': 915, '002202.XSHE': 913, '002008.XSHE': 1327,
'002594.XSHE': 131, '600649.XSHG': 1232, '000725.XSHE': 2559, '0
00917.XSHE': 231, '600352.XSHG': 688, '000046.XSHE': 544, '00267
3.XSHE': 1716, '600383.XSHG': 2968, '000712.XSHE': 178, '600804.
XSHG': 46, '000625.XSHE': 494, '600886.XSHG': 1673, '300058.XSHE
': 376.0}
日期： 2014-05-29 00:00:00 ,持仓： {'601601.XSHG': 18, '600027.XSH
G': 721, '002456.XSHE': 121, '000413.XSHE': 762, '000895.XSHE':
229, '000539.XSHE': 992.0, '002450.XSHE': 339, '600674.XSHG': 97
0, '000783.XSHE': 915, '002202.XSHE': 913, '002008.XSHE': 1327,
'002594.XSHE': 131, '600649.XSHG': 1232, '000725.XSHE': 2559, '0
```

```
00917.XSHE': 231, '600352.XSHG': 688, '000046.XSHE': 544, '00267
3.XSHE': 1716, '600383.XSHG': 2968, '000712.XSHE': 178, '600804.
XSHG': 46, '000625.XSHE': 494, '600886.XSHG': 1673, '300058.XSHE
': 376.0}
日期： 2014-05-30 00:00:00 ,持仓： {'601601.XSHG': 18, '002456.XSH
E': 121, '000413.XSHE': 762, '000895.XSHE': 229, '000539.XSHE':
992.0, '002450.XSHE': 339, '600674.XSHG': 970, '000783.XSHE': 91
5, '002202.XSHE': 913, '002008.XSHE': 1327, '002594.XSHE': 131,
'600649.XSHG': 1232, '000725.XSHE': 2559, '000917.XSHE': 231, '6
00352.XSHG': 688, '000046.XSHE': 544, '002673.XSHE': 1716, '6003
83.XSHG': 2968, '000712.XSHE': 178, '600804.XSHG': 46, '000625.X
SHE': 494, '600886.XSHG': 1673, '300058.XSHE': 376.0}
日期： 2014-06-03 00:00:00 ,持仓： {'601601.XSHG': 18, '002456.XSH
E': 121, '000413.XSHE': 762, '000895.XSHE': 229, '000539.XSHE':
992.0, '002450.XSHE': 339, '600674.XSHG': 970, '000783.XSHE': 91
5, '002202.XSHE': 913, '002008.XSHE': 1327, '002594.XSHE': 131,
'600649.XSHG': 1232, '000725.XSHE': 2559, '000917.XSHE': 231, '0
00046.XSHE': 544, '002673.XSHE': 1716, '600383.XSHG': 2968, '000
712.XSHE': 178, '600804.XSHG': 46, '000625.XSHE': 494, '600886.X
SHG': 1673, '300017.XSHE': 16}
日期： 2014-06-04 00:00:00 ,持仓： {'601601.XSHG': 18, '002456.XSH
E': 121, '000413.XSHE': 762, '000895.XSHE': 229, '000539.XSHE':
992.0, '002450.XSHE': 339, '600674.XSHG': 970, '000783.XSHE': 91
5, '002202.XSHE': 913, '002008.XSHE': 1327, '002594.XSHE': 131,
'000725.XSHE': 2559, '000917.XSHE': 231, '000046.XSHE': 544, '00
2673.XSHE': 1716, '600383.XSHG': 2968, '600633.XSHG': 181, '0007
12.XSHE': 178, '600804.XSHG': 46, '000625.XSHE': 494, '600886.XS
HG': 1673, '300017.XSHE': 16}
日期： 2014-06-05 00:00:00 ,持仓： {'002450.XSHE': 339, '000917.XS
HE': 231, '600633.XSHG': 181, '600674.XSHG': 970, '601601.XSHG':
 18, '002202.XSHE': 913, '000783.XSHE': 915, '000712.XSHE': 178,
 '002008.XSHE': 1327, '600804.XSHG': 46, '000625.XSHE': 494, '00
2594.XSHE': 131, '002456.XSHE': 121, '000413.XSHE': 762, '300017
.XSHE': 16, '000895.XSHE': 229, '002673.XSHE': 1716, '600383.XSH
G': 2968, '000539.XSHE': 992.0, '600886.XSHG': 1673, '000046.XSH
E': 544}
日期： 2014-06-06 00:00:00 ,持仓： {'002450.XSHE': 339, '000917.XS
HE': 231, '600674.XSHG': 970, '601601.XSHG': 18, '002202.XSHE':
913, '000783.XSHE': 915, '000712.XSHE': 178, '002008.XSHE': 1327
, '600804.XSHG': 46, '000625.XSHE': 494, '002594.XSHE': 131, '00
2456.XSHE': 121, '000413.XSHE': 762, '300017.XSHE': 16, '000895.
XSHE': 229, '002673.XSHE': 1716, '600383.XSHG': 2968, '000539.XS
HE': 992.0, '600886.XSHG': 1673, '000046.XSHE': 544}
日期： 2014-06-09 00:00:00 ,持仓： {'002450.XSHE': 339, '000917.XS
HE': 231, '600674.XSHG': 970, '601601.XSHG': 18, '600027.XSHG':
1705, '002202.XSHE': 913, '000783.XSHE': 915, '000712.XSHE': 178
, '002008.XSHE': 1327, '600804.XSHG': 46, '000046.XSHE': 544, '0
02456.XSHE': 121, '000413.XSHE': 762, '300017.XSHE': 16, '000895
.XSHE': 229, '002673.XSHE': 1716, '000539.XSHE': 992.0, '600886.
XSHG': 1673, '600383.XSHG': 2968}
日期： 2014-06-10 00:00:00 ,持仓： {'002450.XSHE': 339, '000917.XS
HE': 231, '600674.XSHG': 970, '601601.XSHG': 18, '600027.XSHG':
1705, '002202.XSHE': 913, '000783.XSHE': 915, '000712.XSHE': 178
```

```
, '002008.XSHE': 1327, '600804.XSHG': 46, '000046.XSHE': 544, '0
02456.XSHE': 121, '000413.XSHE': 762, '000895.XSHE': 229, '00267
3.XSHE': 1716, '000539.XSHE': 992.0, '600886.XSHG': 1673, '60038
3.XSHG': 2968}
日期： 2014-06-11 00:00:00 ,持仓： {'002450.XSHE': 339, '000917.XS
HE': 231, '600633.XSHG': 466, '000712.XSHE': 178, '601601.XSHG':
 18, '600027.XSHG': 1705, '002202.XSHE': 913, '000783.XSHE': 915
, '002008.XSHE': 1327, '600804.XSHG': 46, '000625.XSHE': 557, '0
00046.XSHE': 544, '002456.XSHE': 121, '000413.XSHE': 762, '00089
5.XSHE': 229, '002673.XSHE': 1716, '000539.XSHE': 992.0, '600886
.XSHG': 1673, '600383.XSHG': 2968}
日期： 2014-06-12 00:00:00 ,持仓： {'002450.XSHE': 339, '000917.XS
HE': 231, '600633.XSHG': 466, '000712.XSHE': 178, '601601.XSHG':
 18, '600027.XSHG': 1705, '002202.XSHE': 913, '000783.XSHE': 915
, '002008.XSHE': 1327, '600111.XSHG': 387, '600804.XSHG': 46, '0
00625.XSHE': 557, '000046.XSHE': 544, '002456.XSHE': 121, '00041
3.XSHE': 762, '000895.XSHE': 229, '002673.XSHE': 1716, '000539.X
SHE': 992.0, '600886.XSHG': 1673, '600383.XSHG': 2968}
日期： 2014-06-13 00:00:00 ,持仓： {'002450.XSHE': 339, '000917.XS
HE': 231, '600633.XSHG': 466, '600674.XSHG': 684, '601601.XSHG':
 18, '600027.XSHG': 1705, '002202.XSHE': 913, '000783.XSHE': 915
, '000712.XSHE': 178, '002008.XSHE': 1327, '600111.XSHG': 387, '
600804.XSHG': 46, '000625.XSHE': 557, '000046.XSHE': 544, '00245
6.XSHE': 121, '000413.XSHE': 762, '000895.XSHE': 229, '002673.XS
HE': 1716, '000539.XSHE': 992.0, '600886.XSHG': 1673, '600383.XS
HG': 2968}
日期： 2014-06-16 00:00:00 ,持仓： {'601601.XSHG': 18, '600027.XSH
G': 1705, '600111.XSHG': 387, '002456.XSHE': 121, '000413.XSHE':
 762, '000895.XSHE': 229, '000539.XSHE': 992.0, '002450.XSHE': 3
39, '600887.XSHG': 300, '000783.XSHE': 915, '002202.XSHE': 913,
'002008.XSHE': 1327, '002594.XSHE': 70, '600649.XSHG': 507, '000
725.XSHE': 1478, '000917.XSHE': 231, '000046.XSHE': 544, '002673
.XSHE': 1716, '600383.XSHG': 2968, '600633.XSHG': 466, '600674.X
SHG': 684, '000712.XSHE': 178, '600804.XSHG': 46, '000625.XSHE':
 557, '600886.XSHG': 1673}
日期： 2014-06-17 00:00:00 ,持仓： {'600583.XSHG': 90, '601601.XSH
G': 18, '600027.XSHG': 1705, '600111.XSHG': 387, '002456.XSHE':
121, '000413.XSHE': 762, '000895.XSHE': 229, '000539.XSHE': 992.
0, '002450.XSHE': 339, '600674.XSHG': 684, '000783.XSHE': 915, '
002202.XSHE': 913, '002008.XSHE': 1327, '600839.XSHG': 200, '002
594.XSHE': 70, '600649.XSHG': 507, '000725.XSHE': 1478, '000917.
XSHE': 231, '000046.XSHE': 544, '002673.XSHE': 1716, '600383.XSH
G': 2968, '600633.XSHG': 466, '600887.XSHG': 300, '600166.XSHG':
 129, '000728.XSHE': 64, '000712.XSHE': 178, '600804.XSHG': 46,
'000625.XSHE': 557, '600886.XSHG': 1673}
日期： 2014-06-18 00:00:00 ,持仓： {'600583.XSHG': 90, '601601.XSH
G': 18, '600111.XSHG': 387, '002456.XSHE': 121, '000413.XSHE': 7
62, '000895.XSHE': 229, '000539.XSHE': 992.0, '002450.XSHE': 339
, '600674.XSHG': 684, '000783.XSHE': 915, '002202.XSHE': 913, '0
02008.XSHE': 1327, '600839.XSHG': 200, '002594.XSHE': 70, '60064
9.XSHG': 507, '000725.XSHE': 1478, '000917.XSHE': 231, '600867.X
SHG': 11, '000046.XSHE': 544, '002673.XSHE': 1716, '600383.XSHG'
: 2968, '600633.XSHG': 466, '600887.XSHG': 300, '600166.XSHG': 1
```

```
29, '000728.XSHE': 64, '000712.XSHE': 178, '600804.XSHG': 46, '0
00625.XSHE': 557, '600886.XSHG': 1673}
日期： 2014-06-19 00:00:00 ,持仓： {'600583.XSHG': 90, '601601.XSH
G': 18, '002456.XSHE': 121, '000413.XSHE': 762, '000895.XSHE': 2
29, '000539.XSHE': 992.0, '002450.XSHE': 339, '600674.XSHG': 684
, '000783.XSHE': 915, '002202.XSHE': 913, '002008.XSHE': 1327, '
600839.XSHG': 200, '002594.XSHE': 70, '000725.XSHE': 1478, '0009
17.XSHE': 231, '600867.XSHG': 11, '000046.XSHE': 544, '002673.XS
HE': 1716, '600383.XSHG': 2968, '600633.XSHG': 466, '600887.XSHG
': 300, '000712.XSHE': 178, '600804.XSHG': 46, '000625.XSHE': 55
7, '600886.XSHG': 1673}
日期： 2014-06-20 00:00:00 ,持仓： {'600583.XSHG': 90, '601601.XSH
G': 18, '002456.XSHE': 121, '000413.XSHE': 762, '000895.XSHE': 2
29, '000539.XSHE': 992.0, '002450.XSHE': 339, '600674.XSHG': 684
, '000783.XSHE': 915, '002202.XSHE': 913, '002008.XSHE': 1327, '
002594.XSHE': 70, '000725.XSHE': 1478, '000917.XSHE': 231, '6008
67.XSHG': 11, '000046.XSHE': 544, '002673.XSHE': 1716, '600383.X
SHG': 2968, '600633.XSHG': 466, '600887.XSHG': 300, '000712.XSHE
': 178, '600804.XSHG': 46, '000625.XSHE': 557, '600886.XSHG': 16
73, '300017.XSHE': 118}
日期： 2014-06-23 00:00:00 ,持仓： {'002450.XSHE': 339, '000917.XS
HE': 231, '000625.XSHE': 557, '600583.XSHG': 90, '600383.XSHG': 
2968, '601601.XSHG': 18, '002202.XSHE': 913, '000783.XSHE': 915,
 '000712.XSHE': 178, '002008.XSHE': 1327, '600804.XSHG': 46, '60
0867.XSHG': 11, '002594.XSHE': 70, '002456.XSHE': 121, '000413.X
SHE': 762, '300017.XSHE': 118, '000895.XSHE': 229, '002673.XSHE'
: 1716, '000725.XSHE': 1478, '000539.XSHE': 992.0, '000046.XSHE'
: 544}
日期： 2014-06-24 00:00:00 ,持仓： {'002450.XSHE': 339, '000917.XS
HE': 231, '600383.XSHG': 2968, '601601.XSHG': 18, '000783.XSHE':
 915, '000712.XSHE': 178, '002008.XSHE': 1327, '600804.XSHG': 46
, '600867.XSHG': 11, '002594.XSHE': 70, '002456.XSHE': 121, '000
413.XSHE': 762, '300017.XSHE': 118, '000895.XSHE': 229, '002673.
XSHE': 1716, '000725.XSHE': 1478, '000539.XSHE': 992.0, '000046.
XSHE': 544}
日期： 2014-06-25 00:00:00 ,持仓： {'002450.XSHE': 339, '000917.XS
HE': 231, '600583.XSHG': 1373, '600383.XSHG': 2968, '601601.XSHG
': 18, '000783.XSHE': 915, '000712.XSHE': 178, '002008.XSHE': 13
27, '600111.XSHG': 767, '600804.XSHG': 46, '600867.XSHG': 11, '0
02594.XSHE': 70, '002456.XSHE': 121, '000413.XSHE': 762, '300017
.XSHE': 118, '000895.XSHE': 229, '002673.XSHE': 1716, '000725.XS
HE': 1478, '000539.XSHE': 992.0, '000046.XSHE': 544}
日期： 2014-06-26 00:00:00 ,持仓： {'002450.XSHE': 339, '000917.XS
HE': 231, '600583.XSHG': 1373, '601601.XSHG': 18, '000783.XSHE':
 915, '002008.XSHE': 1327, '600111.XSHG': 767, '600804.XSHG': 46
, '600867.XSHG': 11, '002594.XSHE': 70, '002456.XSHE': 121, '000
413.XSHE': 762, '300017.XSHE': 118, '000895.XSHE': 229, '002673.
XSHE': 1716, '600383.XSHG': 2968, '000539.XSHE': 992.0, '000046.
XSHE': 544}
日期： 2014-06-27 00:00:00 ,持仓： {'002450.XSHE': 339, '000917.XS
HE': 231, '600583.XSHG': 1373, '601601.XSHG': 18, '000783.XSHE':
 915, '002008.XSHE': 1327, '600111.XSHG': 767, '600804.XSHG': 46
, '600867.XSHG': 11, '002594.XSHE': 70, '002456.XSHE': 121, '000
```

```
413.XSHE': 762, '300017.XSHE': 118, '000895.XSHE': 229, '002673.
XSHE': 1716, '600383.XSHG': 2968, '000539.XSHE': 992.0, '000046.
XSHE': 544}
日期： 2014-06-30 00:00:00 ,持仓： {'002450.XSHE': 339, '000917.XS
HE': 231, '600583.XSHG': 1373, '600887.XSHG': 665, '601601.XSHG'
: 18, '000783.XSHE': 915, '002008.XSHE': 1327, '600111.XSHG': 76
7, '600804.XSHG': 46, '600867.XSHG': 11, '002594.XSHE': 70, '002
456.XSHE': 121, '000413.XSHE': 762, '300017.XSHE': 118, '000895.
XSHE': 229, '002673.XSHE': 1716, '600383.XSHG': 2968, '000539.XS
HE': 992.0, '000046.XSHE': 544}
日期： 2014-07-01 00:00:00 ,持仓： {'600583.XSHG': 1373, '601601.X
SHG': 18, '600027.XSHG': 1602, '600111.XSHG': 767, '002456.XSHE'
: 121, '000413.XSHE': 762, '000895.XSHE': 229, '000539.XSHE': 99
2.0, '002450.XSHE': 339, '600674.XSHG': 813, '000783.XSHE': 915,
 '002008.XSHE': 1327, '600839.XSHG': 1500.0, '002594.XSHE': 70,
'000917.XSHE': 231, '600352.XSHG': 615, '600867.XSHG': 11, '0000
46.XSHE': 544, '002673.XSHE': 1716, '600383.XSHG': 2968, '600887
.XSHG': 665, '600166.XSHG': 968, '600804.XSHG': 46, '000625.XSHE
': 399, '300017.XSHE': 118}
```

用5日均线和10日均线进行判断 --- 改进版

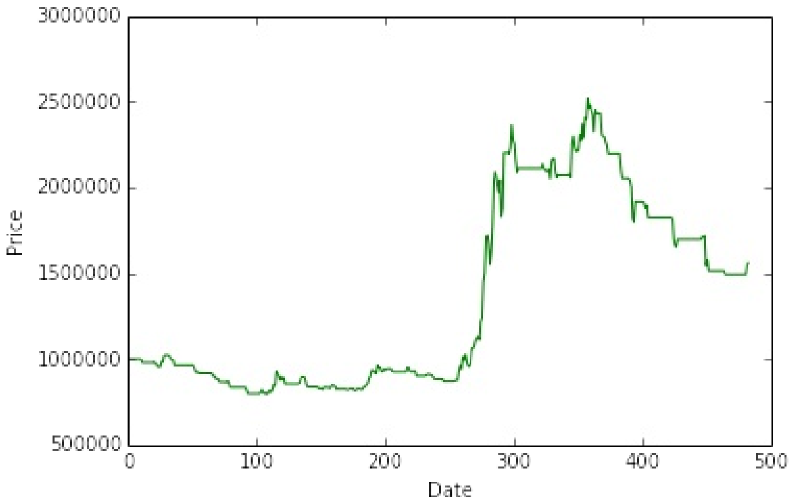
> 来源：https://uqer.io/community/share/561e3a65f9f06c4ca82fb5ec

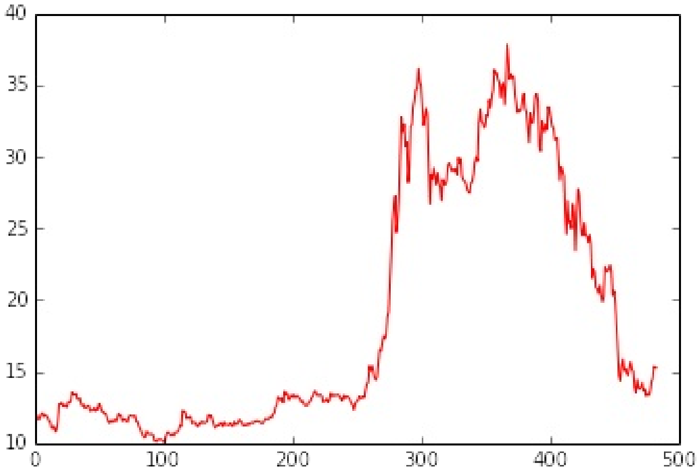
1、 修改Adobe同学的代码

```
data=DataAPI.MktEqudGet(ticker="600030",beginDate="20131001")
#选取600030股票

a = data.closePrice
B = []
n = len(a)
for i in range(10, n):
    x5 = a[i-5:i].mean()                 #5日均线值
    x10 = a[i-10:i].mean()               #10日均线值
    B.append(x5 > x10)
```

```python
import matplotlib.pyplot as plt

o = data.openPrice
m = len(B)
w = 0                       #利润

cash = 1000000              #操作金额1亿，但考虑买的份额为100的整数，取1百万

amount = 0

PL = []                  #利润w的数组
for i in range(1, m):
    k = i + 10
    if B[i-1] == 0 and B[i] == 1 and not amount:
        amount = cash // o[k]      #买入份额
        cash -= o[k] * amount
    elif B[i-1] == 1 and B[i]==0 and amount:
        cash += o[k] * amount       #卖出的金额
        amount = 0
#     print cash, amount

    PL.append(cash + o[k] * amount)

print("利润：{}".format(PL[-1]))

plt.plot(PL,color="green",label="Profit and Loss")
plt.xlabel("Date")
plt.ylabel("Price")
plt.show()

plt.plot(a[10:], color="red",label="Profit and Loss")
plt.show()
```

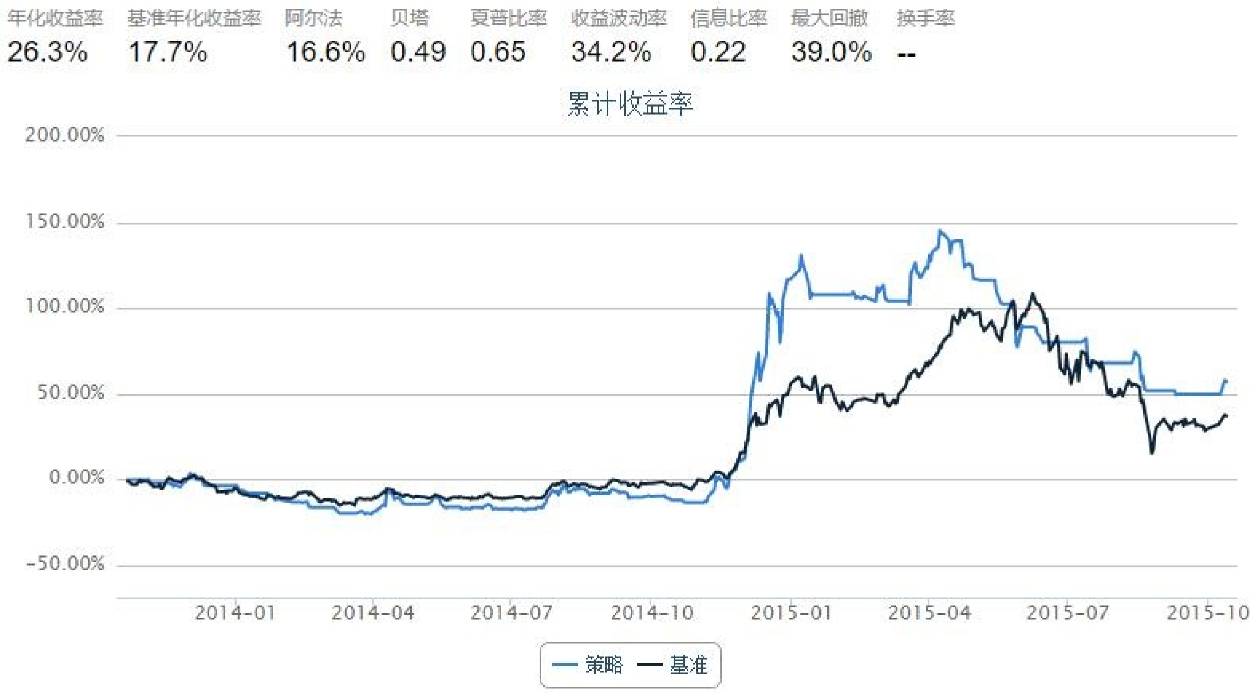
```
利润：1559132.78
```





2、 Uqer框架相同策略

```
import numpy as np

start = '2013-10-01'                # 回测起始时间
end   =  '2015-10-13'               # 回测结束时间
benchmark = 'SH50'                            # 策略参考标准
universe = ['600030.XSHG']                    # 股票池   中信证券
capital_base = 100000                       # 起始资金
commission = Commission(0.0,0.0)

window_short = 5   # 短均线周期
window_long = 10   # 长均线周期

def initialize(account):                    # 初始化虚拟账户状态
    account.fund = universe[0]

def handle_data(account):             # 每个交易日的买入卖出指令
    cp_hist = account.get_attribute_history('closePrice', window
_long)[account.fund]
    short_mean = np.mean(cp_hist[-window_short:]) # 计算短均线值
    long_mean = np.mean(cp_hist[-window_long:])   #计算长均线值

    # 计算买入卖出信号
    if short_mean - long_mean > 0:
        if account.fund not in account.valid_secpos:
            # 空仓时全仓买入，买入股数为100的整数倍
            approximationAmount = int(account.cash / account.ref
erencePrice[account.fund] / 100) * 100
            order(account.fund, approximationAmount)
    else:
        # 卖出时，全仓清空
        if account.fund in account.valid_secpos:
            order_to(account.fund, 0)
```

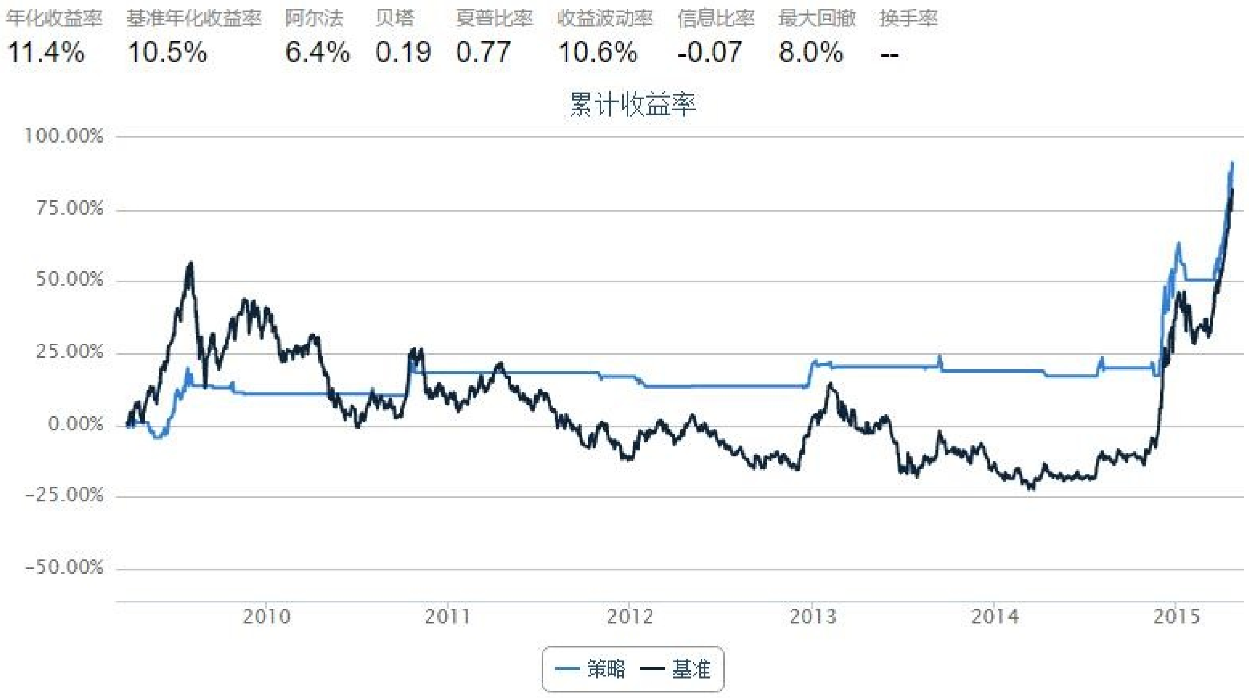


年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
26.3%	17.7%	16.6%	0.49	0.65	34.2%	0.22	39.0%	--

macross

> 来源：https://uqer.io/community/share/55a92cdcf9f06c57a11b53be

```
#第一次写大家帮看看有问题么
#策略  ：  日收盘价高于ma20 买入；  低于ma20 卖出清仓
from matplotlib import pylab
import numpy as np
import pandas as pd
import DataAPI
import seaborn as sns
sns.set_style('white')
################

start = datetime(2008, 1, 1)                # 回测起始时间
end  = datetime(2015, 4, 23)                # 回测结束时间
benchmark = 'SH50'                            # 策略参考标准
universe = ['510050.XSHG']    # 股票池
#benchmark = 'HS300'
#universe = ['510300.XSHG']
capital_base = 100000     # 起始资金
commission = Commission(0.0,0.0)

window_short = 20
window_long = 300
longest_history = window_long
#longest_history = window_short
SD = 0.05

def initialize(account):                    # 初始化虚拟账户状态
    account.fund = universe[0]
    account.SD = SD
    account.window_short = window_short
    account.window_long = window_long

def handle_data(account):             # 每个交易日的买入卖出指令
    hist = account.get_history(longest_history)
    fund = account.fund
    short_mean = np.mean(hist[fund]['closePrice'][-account.windo
w_short:]) # 计算短均线值
    long_mean = np.mean(hist[fund]['closePrice'][-account.window
_long:])   #计算长均线值
    now_price = hist[fund]['closePrice'][-1:]
    #print len(short_mean)
    #print type(now_price)
    #print(now_price)

    #now_price.plot
    #all_close_prices = account.get_attribute_history('closePric
e', 1)

    # 计算买入卖出信号
 #   flag = True if (short_mean - long_mean) > account.SD * long
_mean else False
    flag = True if (now_price - short_mean) > account.SD * short
_mean else False
    if flag:
        if account.position.secpos.get(fund, 0) == 0:
            # 空仓时全仓买入，买入股数为100的整数倍
            approximationAmount = int(account.cash / hist[fund][
'closePrice'][-1]/100.0) * 100
            order(fund, approximationAmount)
    else:
        # 卖出时，全仓清空
        if account.position.secpos.get(fund, 0) >= 0:
            order_to(fund, 0)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
11.4%	10.5%	6.4%	0.19	0.77	10.6%	-0.07	8.0%	--

4.3 MACD

Simple MACD

> 来源：https://uqer.io/community/share/560a3007f9f06c597665ef61

MACD 公式算法:

- 短期EMA： 短期（例如12日）的收盘价指数移动平均值（Exponential Moving Average）
- 长期EMA： 长期（例如26日）的收盘价指数移动平均值（Exponential Moving Average）
- DIF线： （Difference）短期EMA和长期EMA的离差值
- DEA线： （Difference Exponential Average）DIF线的M日指数平滑移动平均线
- MACD线： DIF线与DEA线的差

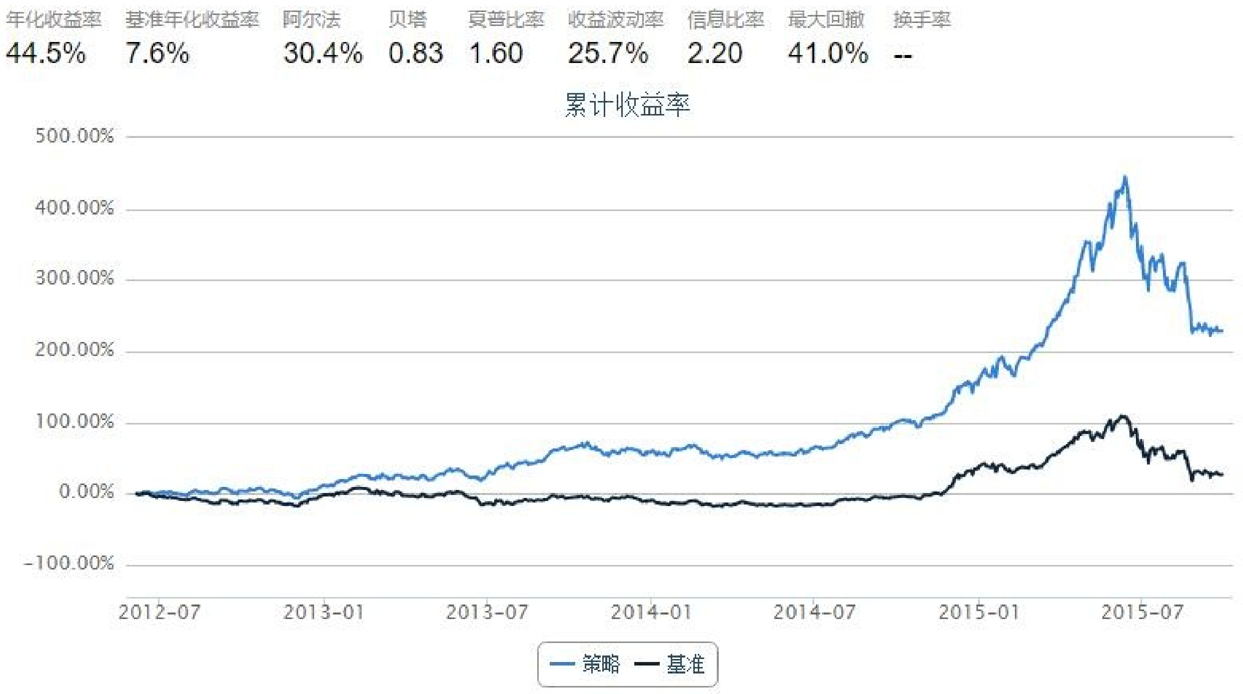
策略实现：

- DIF从下而上穿过DEA，买进；
- 相反，如DIF从上往下穿过DEA，卖出。

策略中使用 `talib` 计算MACD

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

start = '2012-01-01'
end = '2015-09-28'
benchmark = 'HS300'
universe = set_universe('HS300')
capital_base = 1000000
refresh_rate = 5

## 使用talib计算MACD的参数
short_win = 12    # 短期EMA平滑天数
long_win  = 26    # 长期EMA平滑天数
macd_win  = 20     # DEA线平滑天数

stk_num = 20      # 持仓股票数量

longest_history = 100
def initialize(account):
    account.universe = universe

def handle_data(account):
    all_close_prices = account.get_attribute_history('closePrice'
, longest_history)
```

```
    long_bucket = []
    short_bucket = []
    for stk in account.universe:
        prices = all_close_prices[stk]
        if prices is None:
            continue
        try:
            # talib计算MACD
            macd_tmp = talib.MACD(prices, fastperiod=short_win, slowperiod=long_win, signalperiod=macd_win)
            DIF = macd_tmp[0]
            DEA = macd_tmp[1]
            MACD = macd_tmp[2]
        except:
            continue

        # 判断MACD走向
        if MACD[-1] > 0 and MACD[-4] < 0:
            long_bucket.append(stk)
        elif MACD[-1] < 0 and MACD[-4] > 0:
            short_bucket.append(stk)

    hold = []
    # 处理持仓中的股票
    for stk in account.valid_secpos:
        # 在short_bucket中的，卖出
        if stk in short_bucket:
            order_to(stk, 0)
        # 不在short_bucket中的，留着
        else:
            hold.append(stk)

    buy_list = hold
    for stk in long_bucket:
        if stk not in hold:
            buy_list.append(stk)

    if len(buy_list) > 0:
        # 无论buy_list中有多少只股票，都将仓位分成stk_num份，每份买入一只股票
        amount_per_stk = account.referencePortfolioValue/stk_num
        for stk in buy_list:
            amount = int(amount_per_stk/account.referencePrice[stk] / 100.0) * 100
            order_to(stk, amount)
```

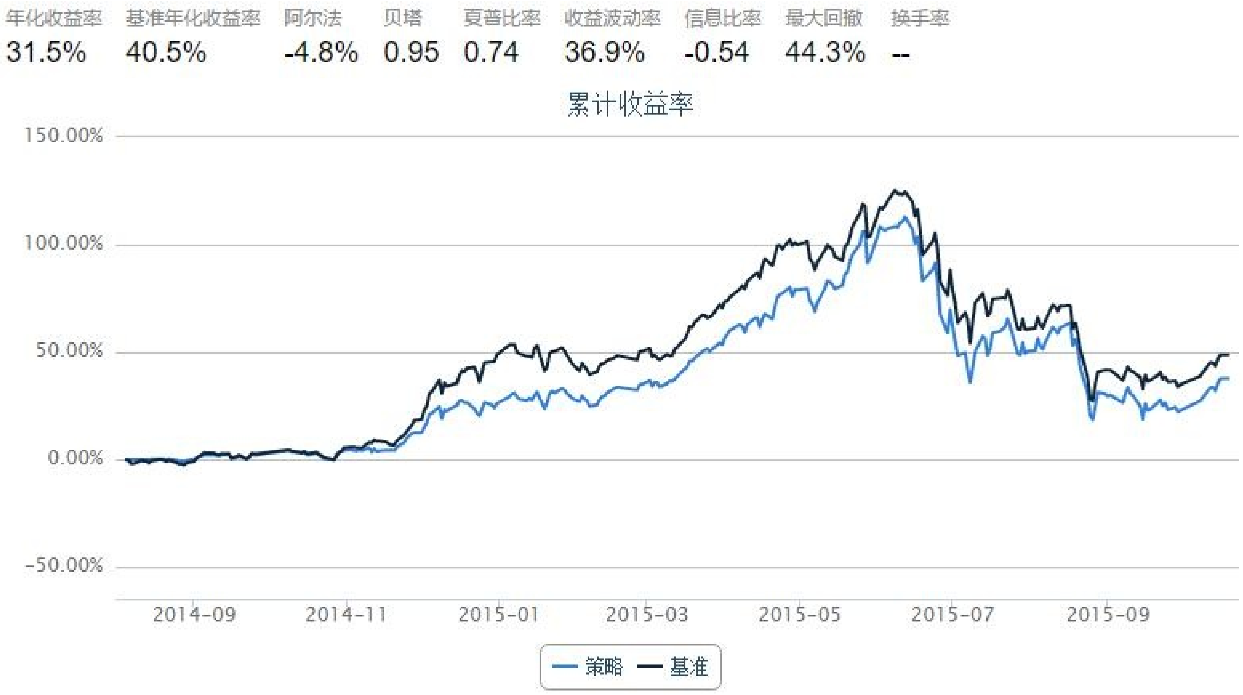


年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
44.5%	7.6%	30.4%	0.83	1.60	25.7%	2.20	41.0%	--

MACD quantization trade

> 来源：https://uqer.io/community/share/56265ed3f9f06c4ca62fb618

```
import talib
import copy
from numpy import arange, array, isnan

#本策略的目的在于对常见的MACD指标策略进行验证
start = '2014-01-01'  #start time
end = '2015-10-19' #end time
benchmark = 'HS300'
#universe = ['000001.XSHE', '600000.XSHG']
universe = set_universe('HS300')           # 证券池，支持股票和基金
capital_base = 1000000
freq = 'd'
refresh_rate = 1

def dea_deviate_from_k_line(k_line, dea):
    #判断k线和dea走势是否背离，主要思路是
    #从最新一天开始往回寻找几个极值点，然后
    #进行极值比较
    #然后是从当前点到第一个极值点进行比对
    #最后按照权重返回对应数值
    #对数组进行翻转
    k_line_re = k_line[::-1]
    dea_re = dea[::-1]
    k_line_extremum = []
    #获取极值点的索引，只统计近似区间的情况
    k_line_extremum_inx = {}
    dea_extremum = []
    dea_extremum_inx = {}
    for c in xrange(1, len(k_line_re) - 1):
        if not isnan(k_line_re[c - 1]) and not isnan(k_line_re[c
]) and not isnan(k_line_re[c + 1]):
            if k_line_re[c - 1] < k_line_re[c] and k_line_re[c +
1] < k_line_re[c]:
                #获取到极值点
                k_line_extremum.append(k_line_re[c])
                k_line_extremum_inx[k_line_re[c]] = c
    for c in xrange(1, len(dea_re) - 1):
        if not isnan(dea_re[c - 1]) and not isnan(dea_re[c]) and
not isnan(dea_re[c + 1]):
            if dea_re[c - 1] < dea_re[c] and dea_re[c + 1] < dea
_re[c]:
                dea_extremum.append(dea_re[c])
                dea_extremum_inx[dea_re[c]] = c
    #对极值点进行筛选判断趋势
    sig_extremum = 0
    if len(k_line_extremum) >= 2 and len(dea_extremum)  >= 2:
        k_line_first_deviate = k_line_extremum[0] - k_line_extre
mum[1]
        dea_first_deviate = dea_extremum[0] - dea_extremum[1]
        k_and_dea_start_deviate = k_line_extremum_inx[k_line_ext
remum[0]] - dea_extremum_inx[dea_extremum[0]]
        k_and_dea_end_deviate = k_line_extremum_inx[k_line_extre
mum[1]] - dea_extremum_inx[dea_extremum[1]]
        #k线和dea的同时启动差距要小于２天
        if abs(k_and_dea_start_deviate) < 2:
            #超过３天的背离趋势被确定为趋势
            k_deviate_dates = abs(k_line_extremum_inx[k_line_ext
remum[0]]) - abs(k_line_extremum_inx[k_line_extremum[1]])
            dea_deviate_date = abs(dea_extremum_inx[dea_extremum[
0]]) - abs(dea_extremum_inx[dea_extremum[1]])
            #最好是能同时停止，如果k线和dea线持续时间差值过大，说明应该以
大的线为主要趋势
            dea_k_deviate = abs(k_deviate_dates) - abs(dea_devia
te_date)
            if abs(k_deviate_dates) >= 3 and abs(dea_deviate_dat
e) >= 3 and dea_k_deviate < 3:
                if k_line_first_deviate < 0 and dea_first_deviat
e > 0 :
                    #k线下降，dea上升说明是变盘上涨信号
                    sig_extremum = 1;
                elif k_line_first_deviate > 0 and dea_first_devi
ate < 0:
                    #k线上升，dea下降说明是变盘下跌信号
                    sig_extremum = -1
            else:
                #需要看是以那个线为准
                if abs(k_deviate_dates) > abs(dea_deviate_date):
                    if k_line_first_deviate < 0:
                        sig_extremum = -1
                    else:
                        sig_extremum = 1
                else:
                    if dea_first_deviate < 0:
                        sig_extremum = -1
                    else:
                        sig_extremum = 1
    return sig_extremum

def initialize(account):
    pass

def handle_data(account):
    fibonacci_sequence_date1 = 144
    fibonacci_sequence_date2 = 143
    hist = account.get_attribute_history('closePrice', fibonacci
_sequence_date1)
    for stock in account.universe:
        s_num = 0
        for c in hist[stock]:
            if isnan(c):
                s_num += 1
        if s_num >= len(hist[stock]) - 10:
            #说明数据太少直接放过
            continue
        macd, macdsignal, macdhist = talib.MACD(hist[stock])
        trade_signal = 0
        for c in range(fibonacci_sequence_date2, len(macd) - 1):
            #1.DIFF、DEA均为正，DIFF向上突破DEA，买入信号
            if not isnan(macd[c]) and not isnan(macdsignal[c]) and macd[c] > 0 and macdsignal[c] > 0:
                if not isnan(macd[c - 1]) and not isnan(macdsignal[c - 1]) and macd[c - 1] < macdsignal[c - 1]:
                    if not isnan(macd[c + 1]) and not isnan(macdsignal[c + 1]) and macd[c + 1] > macdsignal[c + 1]:
                        trade_signal += 1
            #2.DIFF、DEA均为负，DIFF向下跌破DEA，卖出信号
            if not isnan(macd[c]) and not isnan(macdsignal[c]) and macd[c] < 0 and macdsignal[c] < 0:
                if not isnan(macd[c - 1]) and not isnan(macdsignal[c - 1]) and macd[c - 1] > macdsignal[c - 1]:
                    if not isnan(macd[c + 1]) and not isnan(macdsignal[c + 1]) and macd[c + 1] < macdsignal[c + 1]:
                        trade_signal -= 1
            #4.分析MACD柱状线，由负变正，买入信号
            if not isnan(macdhist[c - 1]) and not isnan(macdhist[c]) and not isnan(macdhist[c + 1]):
                if macdhist[c] > macdhist[c - 1] and macdhist[c + 1] > macdhist[c] and macdhist[c] > 0 and macdhist[c - 1] < 0:
                    trade_signal += 1
            #5.分析MACD柱状线，由正变负，卖出信号
            if not isnan(macdhist[c - 1]) and not isnan(macdhist[c]) and not isnan(macdhist[c + 1]):
                if macdhist[c] < macdhist[c - 1] and macdhist[c + 1] < macdhist[c] and macdhist[c] < 0 and macdhist[c - 1] > 0:
                    trade_signal -= 1
        #3.DEA线与K线发生背离，行情反转信号
        #如果返回0则代表不对信号进行叠加
        trade_signal += dea_deviate_from_k_line(hist[stock], macdsignal)
        if trade_signal > 0 and trade_signal <= 1:
            order(stock, 100)
        elif trade_signal > 1:
            order(stock, 200)
        elif trade_signal < 0 and trade_signal >= -1:
            order(stock, -100)
        elif trade_signal < -1:
            order(stock, -200)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
31.5%	40.5%	-4.8%	0.95	0.74	36.9%	-0.54	44.3%	--

MACD平滑异同移动平均线方法

> 来源：https://uqer.io/community/share/55a4e581f9f06c6dd0e17efa

策略思路：

MACD（Moving Average Convergence and Divergence)是Geral Appel 于1979年提出的，利用收盘价的短期（常用为12日）指数移动平均线与长期（常用为26日）指数移动平均线之间的聚合与分离状况，对买进、卖出时机作出研判的技术指标。

公式算法:

- 短期EMA： 短期（例如12日）的收盘价指数移动平均值（Exponential Moving Average）
- 长期EMA： 长期（例如26日）的收盘价指数移动平均值（Exponential Moving Average）
- DIF线： （Difference）短期EMA和长期EMA的离差值
- DEA线： （Difference Exponential Average）DIF线的M日指数平滑移动平均线
- MACD线： DIF线与DEA线的差

参数：SHORT(短期)、LONG(长期)、M天数，一般为12、26、9。指数加权平滑系数为：

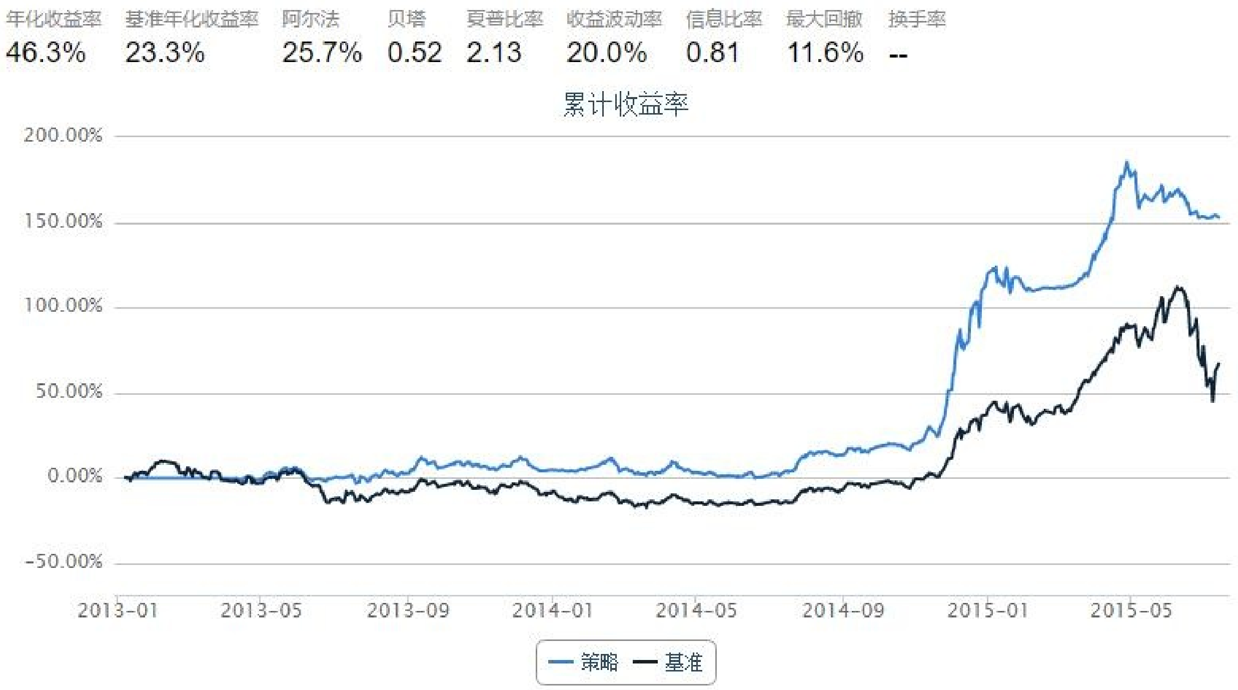
- 短期EMA平滑系数： 2/(SHORT+1)
- 长期EMA平滑系数： 2/(LONG+1)
- DEA线平滑系数： 2/(M+1)

策略实现：

- DIF从下而上穿过DEA，买进；
- 相反，如DIF从上往下穿过DEA，卖出。

```
import pandas as pd

start = datetime(2013, 1, 1)
end = datetime(2015, 7, 13)
benchmark = 'HS300'
#universe = ['601398.XSHG', '600028.XSHG', '601988.XSHG', '600036.XSHG', '600030.XSHG',
            #'601318.XSHG', '600000.XSHG', '600019.XSHG', '600519.XSHG', '601166.XSHG']
universe = set_universe('SH50')
capital_base = 200000
refresh_rate = 1
window = 1

initMACD = -10000.0
histMACD = pd.DataFrame(initMACD, index = universe, columns = ['
preShortEMA', 'preLongEMA', 'preDIF', 'preDEA'])
shortWin = 26    # 短期EMA平滑天数
longWin  = 52    # 长期EMA平滑天数
macdWin  = 15    # DEA线平滑天数

longest_history = window

def initialize(account):
    account.amount = 10000
    account.universe = universe
    account.days = 0

def handle_data(account):
    account.days = account.days+1

    for stk in account.universe:
        all_close_prices = account.get_attribute_history('closeP
rice', 1)
        prices = all_close_prices[stk]
        if prices is None:
            continue

        preShortEMA = histMACD.at[stk, 'preShortEMA']
        preLongEMA = histMACD.at[stk, 'preLongEMA']
        preDIF = histMACD.at[stk, 'preDIF']
        preDEA = histMACD.at[stk, 'preDEA']
        if preShortEMA == initMACD or preLongEMA == initMACD:
            histMACD.at[stk, 'preShortEMA'] = prices[-1]
            histMACD.at[stk, 'preLongEMA'] = prices[-1]
            histMACD.at[stk, 'preDIF'] = 0
            histMACD.at[stk, 'preDEA'] = 0
            return

        shortEMA = preShortEMA*1.0*(shortWin-1)/(shortWin+1) + p
rices[-1]*2.0/(shortWin+1)
        longEMA = preLongEMA*1.0*(longWin-1)/(longWin+1) + price
s[-1]*2.0/(longWin+1)
        DIF = shortEMA - longEMA
        DEA = preDEA*1.0*(macdWin-1)/(macdWin+1) + DIF*2.0/(macd
Win+1)

        histMACD.at[stk, 'preShortEMA'] = shortEMA
        histMACD.at[stk, 'preLongEMA'] = longEMA
        histMACD.at[stk, 'preDIF'] = DIF
        histMACD.at[stk, 'preDEA'] = DEA

        if account.days > longWin and account.days%1 == 0:
            #if DIF > 0 and DEA > 0 and preDIF > preDEA and DIF
< DEA:
            if preDIF > preDEA and DIF < DEA:
                order_to(stk, 0)
            #if DIF < 0 and DEA < 0 and preDIF < preDEA and DIF
> DEA:
```

```
            if preDIF < preDEA and DIF > DEA:
                amount = account.amount/prices[-1]
                order_to(stk, amount)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
46.3%	23.3%	25.7%	0.52	2.13	20.0%	0.81	11.6%	--

4.4 阿隆指标 • 技术指标阿隆（ Aroon ）全解析

> 来源：https://uqer.io/community/share/55e69497f9f06c1eaa81f9d4

一、阿隆指标(Aroon)简介

阿隆指标（Aroon）是由图莎尔·钱德（Tushar Chande)1995 年发明的，它通过计算自价格达到近期最高值和最低值以来所经过的期间数，帮助投资者预测证券价格从趋势到区域、区域或反转的变化。在技术分析领域中，有一个说法，一个指标使用的人越多，其效力越低。这个技术指标还挺冷门的，我们一同来看看它的效果。

```
from CAL.PyCAL import *
import numpy as np
import pandas as pd
from pandas import DataFrame
from heapq import nlargest
from heapq import nsmallest
```

二、Aroon计算方法

Aroon指标分为两个具体指标，分别 `AroonUp` 和 `AroonDown` 。其具体计算方式为：

- `AroonUp = [(计算期天数-最高价后的天数)/计算期天数]*100`
- `AroonDown = [(计算期天数-最低价后的天数)/计算期天数]*100`
- `AroonOsc = AroonUp - AroonDown`

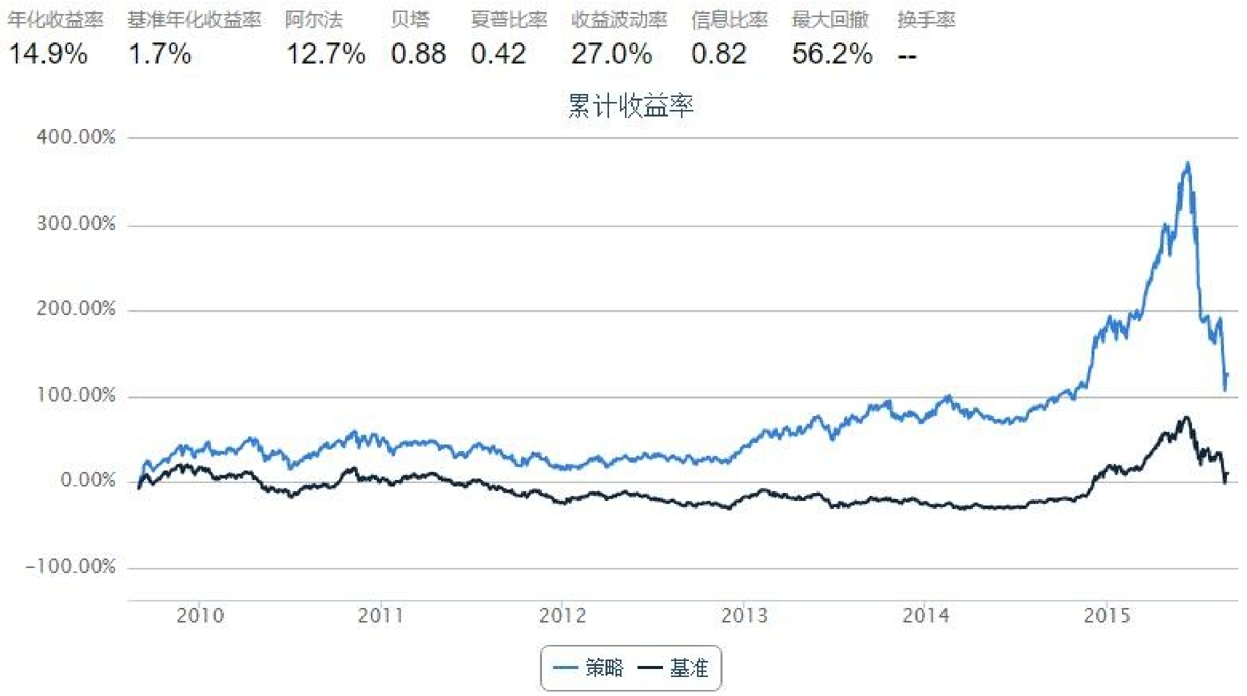
计算期天数通常取20天

```
def aroonUp(account,timeLength=20):
    #运用heapq包的nlargest函数，可以轻松获得：计算期天数-最高价后的天数
    eq_AroonUp = {}
    history = account.get_attribute_history('closePrice',timeLength)
    for stk in account.universe:
        priceSeries = pd.Series(history[stk])
        eq_AroonUp[stk] = (nlargest(1,range(len(priceSeries)),key=priceSeries.get)[0]+1)*100/timeLength # eq_AroonUp[stk]范围在[5,100]之间
    return eq_AroonUp

def aroonDown(account,timeLength=20):
    #运用heapq包的nsmallest函数，可以轻松获得:计算期天数-最低价后的天数
    eq_AroonDown = {}
    history = account.get_attribute_history('closePrice',timeLength)
    for stk in account.universe:
        priceSeries = pd.Series(history[stk])
        eq_AroonDown[stk] = (nsmallest(1,range(len(priceSeries)),key=priceSeries.get)[0]+1)*100/timeLength # eq_AroonDown[stk]范围在[5,100]之间
    return eq_AroonDown
```

三、Aroon指标的基本用法

- 当 `AroonUp` 指标向下跌破50 时，表示向上的趋势正在失去动力；当 `AroonDown` 指标向下跌破50时，表示向下的趋势正在失去动力；如果两个指标都在低位，表示股价没有明确的趋势；如果指标在70 以上，表示趋势十分强烈；如果在30 以下，表明相反的趋势正在酝酿。通常来说， `AroonOsc` 在0附近时，是典型的无趋势特征，股票处于盘整阶段。
- 参考研报《技术指标系列（三）——加入“二次确认”的AROON 阿隆优化指标》中的方法，我们买入 `AroonOsc > 50` 的股票。

```
start = '2009-08-01'                             # 回测起始时间
end = '2015-08-31'                               # 回测结束时间
benchmark = 'HS300'                              # 策略参考标准
universe = set_universe('HS300')                   # 证券池，支持股票和基金

capital_base = 100000                            # 起始资金
freq = 'd'                                       # 策略类型，'d'表示日间
策略使用日线回测，'m'表示日内策略使用分钟线回测
refresh_rate = 10                                # 调仓频率，表示执行hand
le_data的时间间隔，若freq = 'd'时间间隔的单位为交易日，若freq = 'm'时间
间隔为分钟

def initialize(account):                         # 初始化虚拟账户状态

    pass

def handle_data(account):                        # 每个交易日的买入卖出指令

    eq_AroonUp = aroonUp(account,20)
    eq_AroonDown = aroonDown(account,20)
    buyList = []
    for stk in account.valid_secpos:
            order_to(stk, 0)

    for stk in account.universe:
        if  eq_AroonUp[stk] - eq_AroonDown[stk] > 50:
            buyList.append(stk)

    for stk in buyList[:]:
        if stk not in account.universe or account.referencePrice
[stk] == 0 or np.isnan(account.referencePrice[stk]):
            buyList.remove(stk)

    for stk in buyList:
        order(stk, account.referencePortfolioValue/account.refer
encePrice[stk]/len(buyList))
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
14.9%	1.7%	12.7%	0.88	0.42	27.0%	0.82	56.2%	--

可以看出，策略在股市处于震荡市和牛市中，表现很好；而在熊市和暴跌中，表现的非常差，最大回撤很大。这从阿隆指标的构造中，就可以理解，阿隆指标是一个跟踪趋势的指标，在震荡市和牛市中，都能精选出股票，超越指数；然而在暴跌中，处于上升趋势的股票可能跌的更惨，倾巢之下，焉有完卵。。。

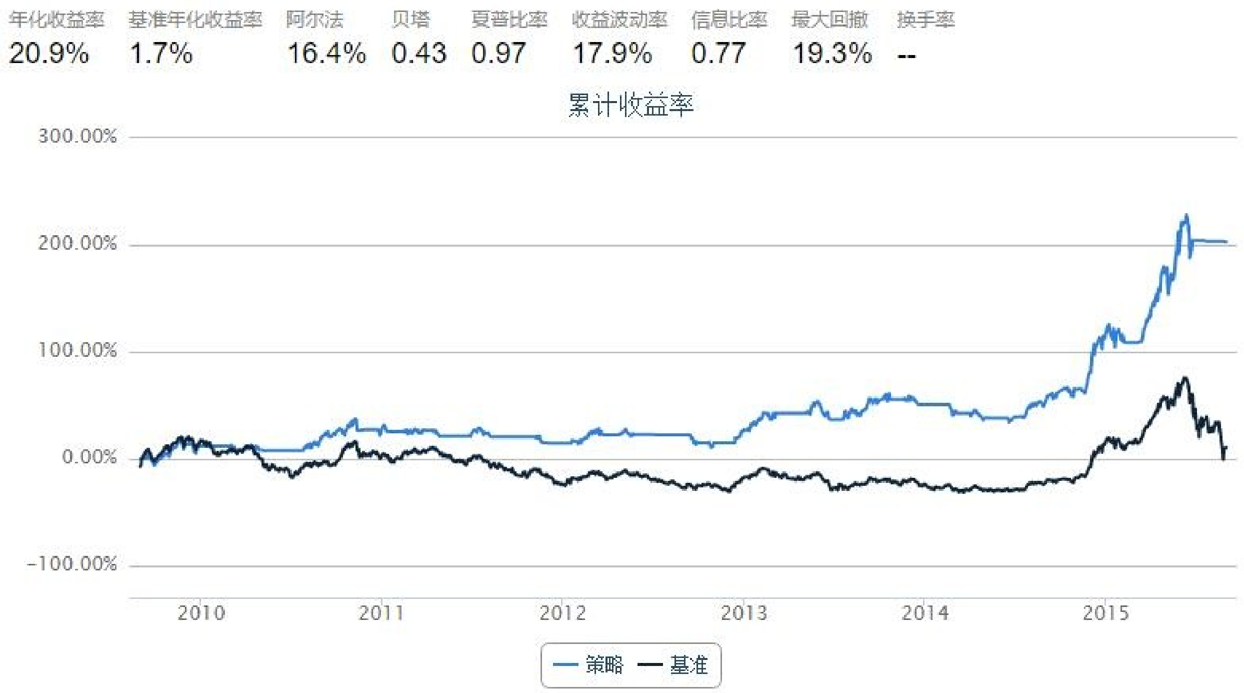
四、运用Aroon指标来择时

前文说到阿隆指标是一个跟踪趋势的指标，既然如此，我们为什么不把它用来择时呢？

```
def aroonIndex(account,timeLength=20):
    #构建指数阿隆指标
    indexSeries = pd.Series(account.get_symbol_history('benchmark', timeLength)['closeIndex'])
    indexAronUp = (nlargest(1,range(len(indexSeries)),key=indexSeries.get)[0]+1)*100/timeLength
    indexAronDown = (nsmallest(1,range(len(indexSeries)),key=indexSeries.get)[0]+1)*100/timeLength
    indexOsc = indexAronUp - indexAronDown
    return indexOsc
```

当 `indexOsc > 0` 时，我们大致认为现在的市场环境没有那么差，可以考虑开仓，编写如下策略。

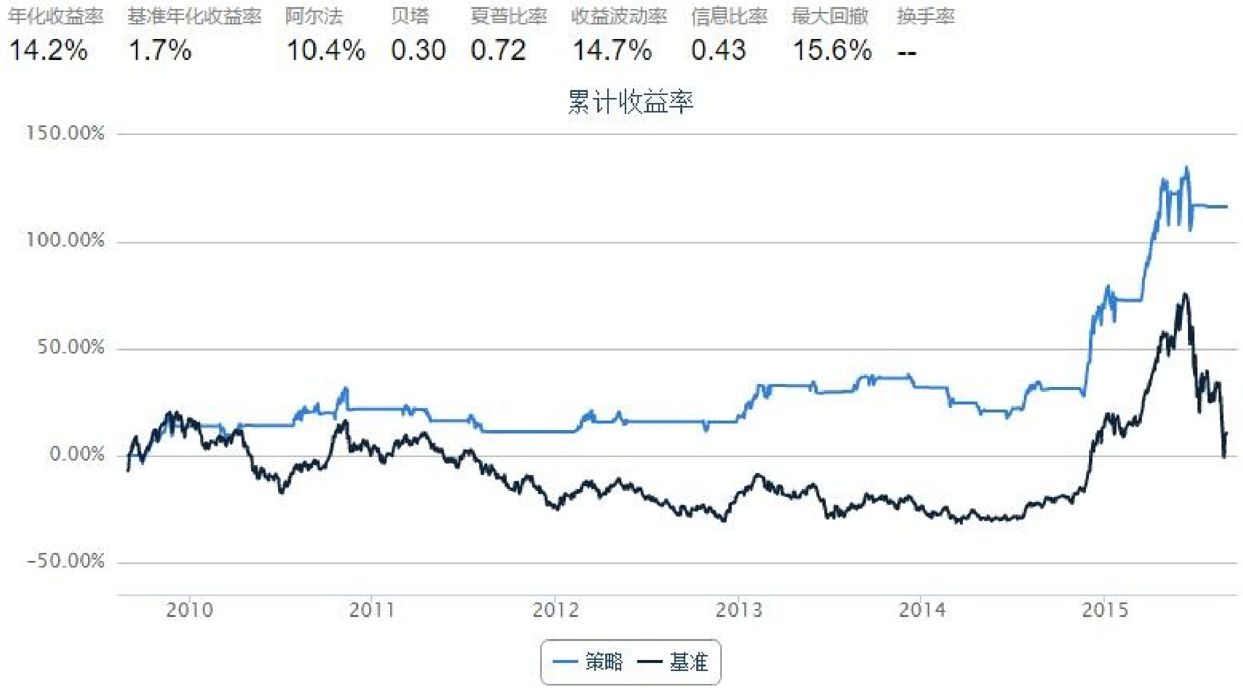
```
start = '2009-08-01'                           # 回测起始时间
end = '2015-08-31'                             # 回测结束时间
benchmark = 'HS300'                            # 策略参考标准
universe = set_universe('HS300')                 # 证券池，支持股票和基金

capital_base = 100000                          # 起始资金
freq = 'd'                                     # 策略类型，'d'表示日间
策略使用日线回测，'m'表示日内策略使用分钟线回测
refresh_rate = 10                              # 调仓频率，表示执行hand
le_data的时间间隔，若freq = 'd'时间间隔的单位为交易日，若freq = 'm'时间
间隔为分钟

def initialize(account):                       # 初始化虚拟账户状态

    pass

def handle_data(account):                      # 每个交易日的买入卖出指令

    eq_AroonUp = aroonUp(account,20)
    eq_AroonDown = aroonDown(account,20)
    index_osc = aroonIndex(account,20)
    buyList = []
    for stk in account.valid_secpos:
            order_to(stk, 0)

    if index_osc > 0:
        for stk in account.universe:
            if  eq_AroonUp[stk] - eq_AroonDown[stk] > 50:
                buyList.append(stk)

        for stk in buyList[:]:
            if stk not in account.universe or account.referenceP
rice[stk] == 0 or np.isnan(account.referencePrice[stk]):
                buyList.remove(stk)

        for stk in buyList:
            order(stk, account.referencePortfolioValue/account.r
eferencePrice[stk]/len(buyList))
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
20.9%	1.7%	16.4%	0.43	0.97	17.9%	0.77	19.3%	--

可以看出运用阿隆指标来择时的效果还是不错的，震荡市能跑赢指数，牛市的收益基本可以吃到，暴跌也几乎完美的规避了！缺点就是最大回测还是偏大，可以考虑让条件更严格，让 `indexOsc > 50` 。

```
start = '2009-08-01'                           # 回测起始时间
end = '2015-08-31'                             # 回测结束时间
benchmark = 'HS300'                            # 策略参考标准
universe = set_universe('HS300')                 # 证券池，支持股票和基金

capital_base = 100000                          # 起始资金
freq = 'd'                                     # 策略类型，'d'表示日间
策略使用日线回测，'m'表示日内策略使用分钟线回测
refresh_rate = 10                              # 调仓频率，表示执行hand
le_data的时间间隔，若freq = 'd'时间间隔的单位为交易日，若freq = 'm'时间
间隔为分钟

def initialize(account):                       # 初始化虚拟账户状态

    pass

def handle_data(account):                      # 每个交易日的买入卖出指令

    eq_AroonUp = aroonUp(account,20)
    eq_AroonDown = aroonDown(account,20)
    index_osc = aroonIndex(account,20)
    buyList = []
    for stk in account.valid_secpos:
            order_to(stk, 0)

    if index_osc > 50:
        for stk in account.universe:
            if  eq_AroonUp[stk] - eq_AroonDown[stk] > 50:
                buyList.append(stk)

        for stk in buyList[:]:
            if stk not in account.universe or account.referenceP
rice[stk] == 0 or np.isnan(account.referencePrice[stk]):
                buyList.remove(stk)

        for stk in buyList:
            order(stk, account.referencePortfolioValue/account.r
eferencePrice[stk]/len(buyList))
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
14.2%	1.7%	10.4%	0.30	0.72	14.7%	0.43	15.6%	--

将择时条件设置更严格后，最大回撤果然有所下降，但年化收益率也有大幅下降。从回测图形中，也可以明显看到，指标具有很强的滞后性，往往是指数开始涨了一段时间，策略才开始开仓买入。将 `indexOsc` 条件设置的越严格，滞后性表现的就越明显，这样虽然可以提高正确率，减小最大回撤，但有许多收益也错过了。

4.5 CCI • CCI 顺势指标探索

> 来源：https://uqer.io/community/share/55f28179f9f06c1ea681fa40

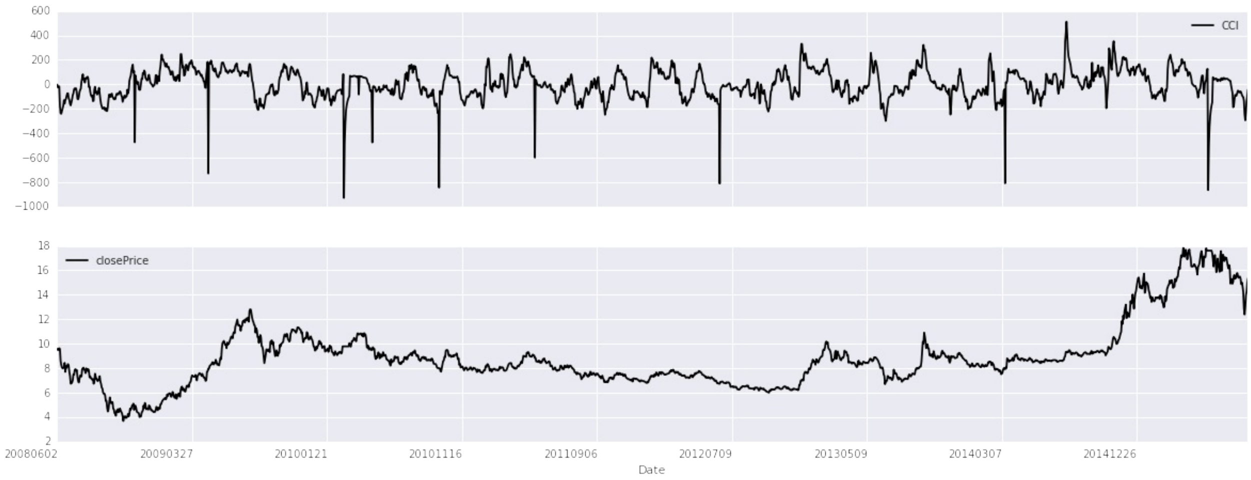
一、CCI指标简介与构造

顺势指标CCI由唐纳德拉姆伯特所创，是通过测量股价的波动是否已超出其正常范围，来预测股价变化趋势的技术分析指标。计算方法参考《技术指标系列（五）——CCI的顺势而为》。

下面描绘出CCI与股价时序图走势

```
def cci(stock,start_date,end_date,windows):   #设置股票，起始时间，以及CCI指标多少日
    import pandas as pd
    import numpy as np
    from CAL.PyCAL import *
    Alpha = 0.015
    eq_TP = {}
    eq_MATP = {}
    eq_meanDev = {}
    eq_CCI = {}
    cal = Calendar('China.SSE')
    windows = '-'+str(windows)+'B'
    start_date = Date.strptime(start_date,"%Y%m%d")
    end_date = Date.strptime(end_date,"%Y%m%d")
    timeLength = cal.bizDatesList(start_date, end_date)

    for i in xrange(len(timeLength)):

        begin_date = cal.advanceDate(timeLength[i],windows,BizDayConvention.Unadjusted)
        begin_date =begin_date.strftime("%Y%m%d")
        timeLength[i] = timeLength[i].strftime("%Y%m%d")
        eq_static = DataAPI.MktEqudAdjGet(secID=stock,beginDate=begin_date,endDate=timeLength[i],field=['secID','highestPrice','lowestPrice','closePrice'],pandas="1")
        for stk in stock:
            try:
                eq_TP[stk] = np.array(eq_static[eq_static['secID'] == stk].mean(axis=1))
                eq_MATP[stk] = sum(eq_TP[stk])/len(eq_TP[stk])
                eq_meanDev[stk] = sum(abs(eq_TP[stk] - eq_MATP[stk]))/len(eq_TP[stk])
                eq_CCI[stk].append((eq_TP[stk][-1] - eq_MATP[stk])/(Alpha * eq_meanDev[stk]))
            except:
                eq_CCI[stk] = []

    Date = pd.DataFrame(timeLength)
    eq_CCI = pd.DataFrame(eq_CCI)
    cciSeries = pd.concat([Date,eq_CCI],axis =1)
    cciSeries.columns = ['Date','CCI']
    return cciSeries

def cci_price_Plot(stock,start_date,end_date,windows):
    cciSeries = cci(stock,start_date,end_date,windows)
    closePrice = DataAPI.MktEqudAdjGet(secID=stock,beginDate=start_date,endDate=end_date,field=['closePrice'],pandas="1")
    table = pd.merge(cciSeries,closePrice, left_index=True, right_index=True, how = 'inner')
    return table
```

```
import pandas as pd
import numpy as np
from CAL.PyCAL import *
cal = Calendar('China.SSE')
table = cci_price_Plot(['600000.XSHG'],'20080531','20150901',30)
  #绘制浦发银行的CCI与股价对比图
tableDate = table.set_index('Date')
tableDate.plot(figsize=(20,8),subplots = 1)


array([<matplotlib.axes.AxesSubplot object at 0x60037d0>,
       <matplotlib.axes.AxesSubplot object at 0x602fa90>], dtype
=object)
```



二、CCI指标简单应用

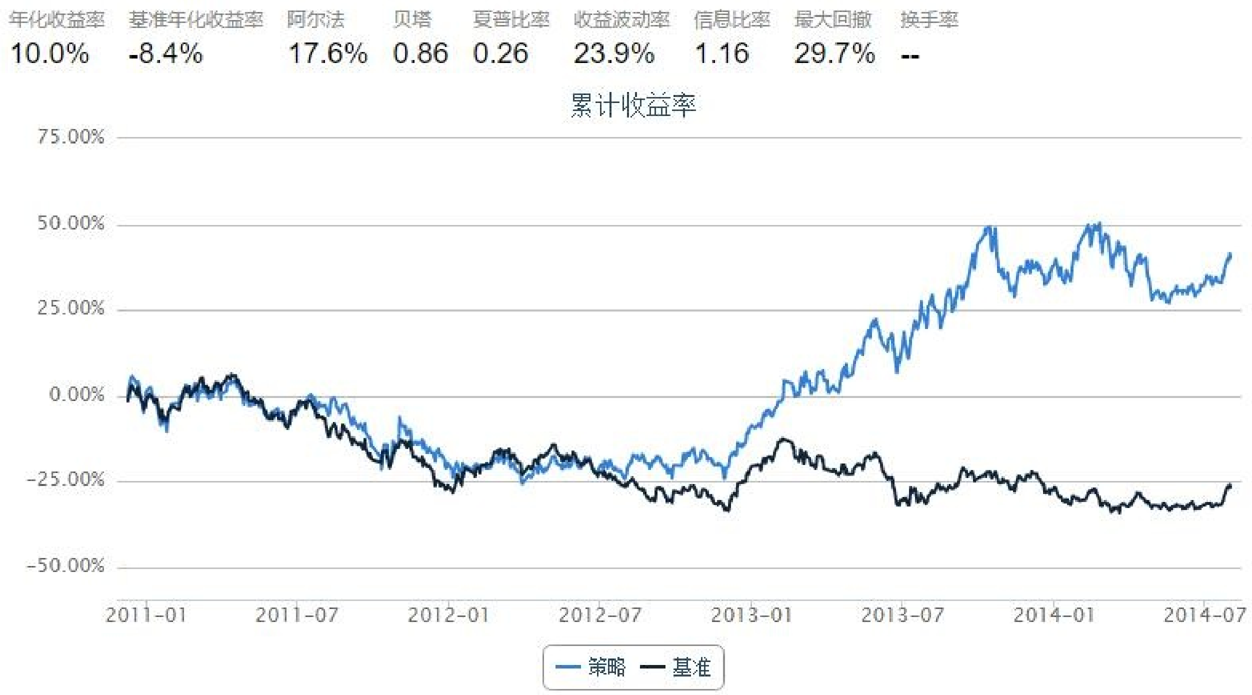
选取CCI处于100和150之间，开始处于上涨趋势的股票。关于windows，我们用quick_backtest做一个简单的优化

```
def cci(account,N=20):
    Alpha = 0.015
    eq_TP = {}
    eq_MATP = {}
    eq_meanDev = {}
    eq_CCI = {}
    eq_highPrice = account.get_attribute_history('highPrice',N)
    eq_closePrice = account.get_attribute_history('closePrice',N
)
    eq_lowPrice = account.get_attribute_history('lowPrice',N)
    for stk in account.universe:
        eq_TP[stk] = (eq_highPrice[stk] + eq_closePrice[stk] + e
q_lowPrice[stk])/3
        eq_MATP[stk] = sum(eq_TP[stk])/len(eq_TP[stk])
        eq_meanDev[stk] = sum(abs(eq_TP[stk] - eq_MATP[stk]))/le
n(eq_TP[stk])
        eq_CCI[stk] = (eq_TP[stk][-1] - eq_MATP[stk])/(Alpha * e
q_meanDev[stk])
    return eq_CCI
```

```
start = '2010-08-01'                     # 回测起始时间
end = '2014-08-01'                       # 回测结束时间
benchmark = 'HS300'                      # 策略参考标准
universe = set_universe('HS300')         # 证券池，支持股票和基金
capital_base = 100000                    # 起始资金
freq = 'd'                               # 策略类型，'d'表示日间
策略使用日线回测，'m'表示日内策略使用分钟线回测
refresh_rate = 20                        # 调仓频率，表示执行hand
le_data的时间间隔，若freq = 'd'时间间隔的单位为交易日，若freq = 'm'时间
间隔为分钟

sim_params = quartz.sim_condition.env.SimulationParameters(start
, end, benchmark, universe, capital_base)
idxmap_all, data_all = quartz.sim_condition.data_generator.get_d
aily_data(sim_params)
```

```
from CAL.PyCAL import *
import pandas as pd
import numpy as np


def initialize(account):                 # 初始化虚拟账户状态
    pass

def handle_data(account):                # 每个交易日的买入卖出指
令
    eq_CCI = cci(account,window)
    buylist = []
    for stk in account.universe:
        try:
            if eq_CCI[stk] > 100 and eq_CCI[stk] < 150:
                buylist.append(stk)
        except:
            pass

    for stk in account.valid_secpos:
        order_to(stk, 0)

    for stk in buylist[:]:
            if stk not in account.universe or account.referenceP
rice[stk] == 0 or np.isnan(account.referencePrice[stk]):
                bulist.remove(stk)

    for stk in buylist:
        order(stk, account.referencePortfolioValue/account.refer
encePrice[stk]/len(buylist))
print 'window   annualized_return   sharpe   max_drawdown'
for window in range(10, 100, 5):
    strategy = quartz.sim_condition.strategy.TradingStrategy(ini
tialize, handle_data)
    bt_test, acct = quartz.quick_backtest(sim_params, strategy,
idxmap_all, data_all,refresh_rate = refresh_rate)
    perf = quartz.perf_parse(bt_test, acct)
    print '  {0:2d}        {1:>7.4f}          {2:>7.4f}    {3:>7
.4f}'.format(window, perf['annualized_return'], perf['sharpe'],
perf['max_drawdown'])

window   annualized_return   sharpe   max_drawdown
  10         0.0186          -0.0610     0.4161
  15        -0.0367          -0.2818     0.5448
  20         0.0753           0.1734     0.4531
  25         0.0268          -0.0254     0.3098
  30        -0.0440          -0.3198     0.5640
  35         0.0481           0.0599     0.4794
  40         0.1117           0.3270     0.4057
  45         0.0619           0.1176     0.2353
  50        -0.0425          -0.3442     0.4226
  55         0.0227          -0.0577     0.3355
  60         0.0513           0.0540     0.4461
  65         0.0860           0.1969     0.2304
  70         0.0434           0.0218     0.3005
  75         0.0126          -0.1176     0.3672
  80         0.0891           0.2084     0.3728
  85         0.1002           0.2554     0.2971
  90         0.0768           0.1687     0.2710
  95         0.0243          -0.0588     0.3461
```

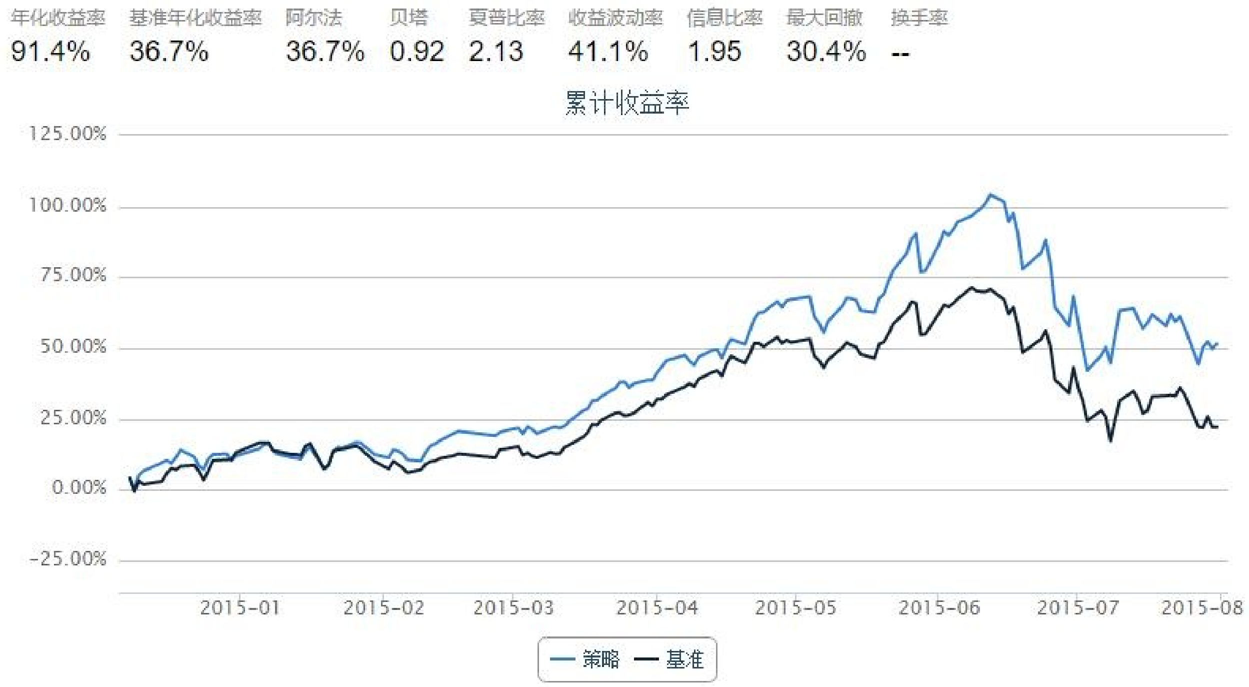
```
from CAL.PyCAL import *
import pandas as pd
import numpy as np

start = '2010-08-01'                       # 回测起始时间
end = '2014-08-01'                         # 回测结束时间
benchmark = 'HS300'                        # 策略参考标准
universe = set_universe('HS300')           # 证券池，支持股票和基金
capital_base = 100000                      # 起始资金
freq = 'd'                                 # 策略类型，'d'表示日间
策略使用日线回测，'m'表示日内策略使用分钟线回测
refresh_rate = 20                          # 调仓频率，表示执行hand
le_data的时间间隔，若freq = 'd'时间间隔的单位为交易日，若freq = 'm'时间
间隔为分钟

def initialize(account):                   # 初始化虚拟账户状态
    pass

def handle_data(account):                  # 每个交易日的买入卖出指
令
    eq_CCI = cci(account,85)
    buylist = []
    for stk in account.universe:
        try:
            if eq_CCI[stk] > 100 and eq_CCI[stk] < 150:
                buylist.append(stk)
        except:
            pass


    for stk in account.valid_secpos:
        order_to(stk, 0)

    for stk in buylist[:]:
            if stk not in account.universe or account.referenceP
rice[stk] == 0 or np.isnan(account.referencePrice[stk]):
                bulist.remove(stk)

    for stk in buylist:
        order(stk, account.referencePortfolioValue/account.refer
encePrice[stk]/len(buylist))
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
10.0%	-8.4%	17.6%	0.86	0.26	23.9%	1.16	29.7%	--

样本外测试

```
from CAL.PyCAL import *
import pandas as pd
import numpy as np

start = '2014-08-01'                       # 回测起始时间
end = '2015-08-01'                         # 回测结束时间
benchmark = 'HS300'                        # 策略参考标准
universe = set_universe('HS300')           # 证券池，支持股票和基金
capital_base = 100000                      # 起始资金
freq = 'd'                                 # 策略类型，'d'表示日间
策略使用日线回测，'m'表示日内策略使用分钟线回测
refresh_rate = 20                          # 调仓频率，表示执行hand
le_data的时间间隔，若freq = 'd'时间间隔的单位为交易日，若freq = 'm'时间
间隔为分钟

def initialize(account):                   # 初始化虚拟账户状态
    pass

def handle_data(account):                  # 每个交易日的买入卖出指
令
    eq_CCI = cci(account,85)
    buylist = []
    for stk in account.universe:
        try:
            if eq_CCI[stk] > 100 and eq_CCI[stk] < 150:
                buylist.append(stk)
        except:
            pass


    for stk in account.valid_secpos:
        order_to(stk, 0)

    for stk in buylist[:]:
            if stk not in account.universe or account.referenceP
rice[stk] == 0 or np.isnan(account.referencePrice[stk]):
                bulist.remove(stk)

    for stk in buylist:
        order(stk, account.referencePortfolioValue/account.refer
encePrice[stk]/len(buylist))
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
91.4%	36.7%	36.7%	0.92	2.13	41.1%	1.95	30.4%	--

4.6 RSI

重写 rsi

> 来源：https://uqer.io/community/share/560dfd84f9f06c4ca82fb597

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

start = '2014-01-01'                       # 回测起始时间
end = '2015-01-01'                        # 回测结束时间
benchmark = 'HS300'                       # 策略参考标准
universe = set_universe('HS300')               # 证券池，支持股票
和基金
capital_base = 10000000                    # 起始资金
freq = 'd'                              # 策略类型，'d'表示日间策
略使用日线回测
refresh_rate = 1                          # 调仓频率，表示执行hand
le_data的时间间隔，由于freq = 'd'，时间间隔的单位为交易日
pieces=10                               #每个标的最多买1/10

def initaialize(account):
    pass


def handle_data(account):
    prices=account.get_attribute_history('closePrice',100)
    for s in universe:
        cun_price=price[s][-1]
        cun_amount[s]=account.secpos.get(s,0)
        RSI=ta.RSI(prices[s],9)
        buy_flag=RSI[-1]>RSI[-2] and RSI[-1]>30     #计算买入条件
        sell_flag = RSI[-1]<RSI[-2] and RSI[-1]<70   #计算卖出条件
        amount_max=int((0.1*capital_base)/cun_price)  #计算单支仓
位上限
        amount = min(int(2500000/cun_price),amount_max-cun-amoun
t[s]) #计算下单量
        if buy_flag and (cun_amount[s]<amount_max):
            order(s,amount)
        elif sell_flag and (cun_amount[s]>0):
            order_to(s,0)

----------------------------------------------------------------
-----------
ValueError                                Traceback (most recent
 call last)
<mercury-input-6-72d5ab71705d> in <module>()
     61         perf = quartz.perf_parse(bt, quartz_acct)
     62     elif QUARTZ_CACHE.get('start', 0) == sim_params.firs
```

```
t_trading_day and          QUARTZ_CACHE.get('end', 0) == sim_par
ams.last_trading_day and          QUARTZ_CACHE.get('benchmark',
0) == benchmark and          QUARTZ_CACHE.get('universe', 0) ==
sim_params.universe:
---> 63         strategy = quartz.sim_condition.strategy.Trading
Strategy(initialize, handle_data)
     64         bt, quartz_acct = quartz.quick_backtest_generato
r(sim_params = QUARTZ_CACHE['sim_params'],
     65
  strategy = strategy,

python2.7/site-packages/quartz/sim_condition/strategy.pyc in __i
nit__(self, initialize, handle_data)
     19     def __init__(self, initialize=None, handle_data=None
):
     20         if not hasattr(initialize, '__call__'):
---> 21             raise ValueError('initialize must be a funct
ion!')
     22         else:
     23             self._initialize = initialize

ValueError: initialize must be a function!
```

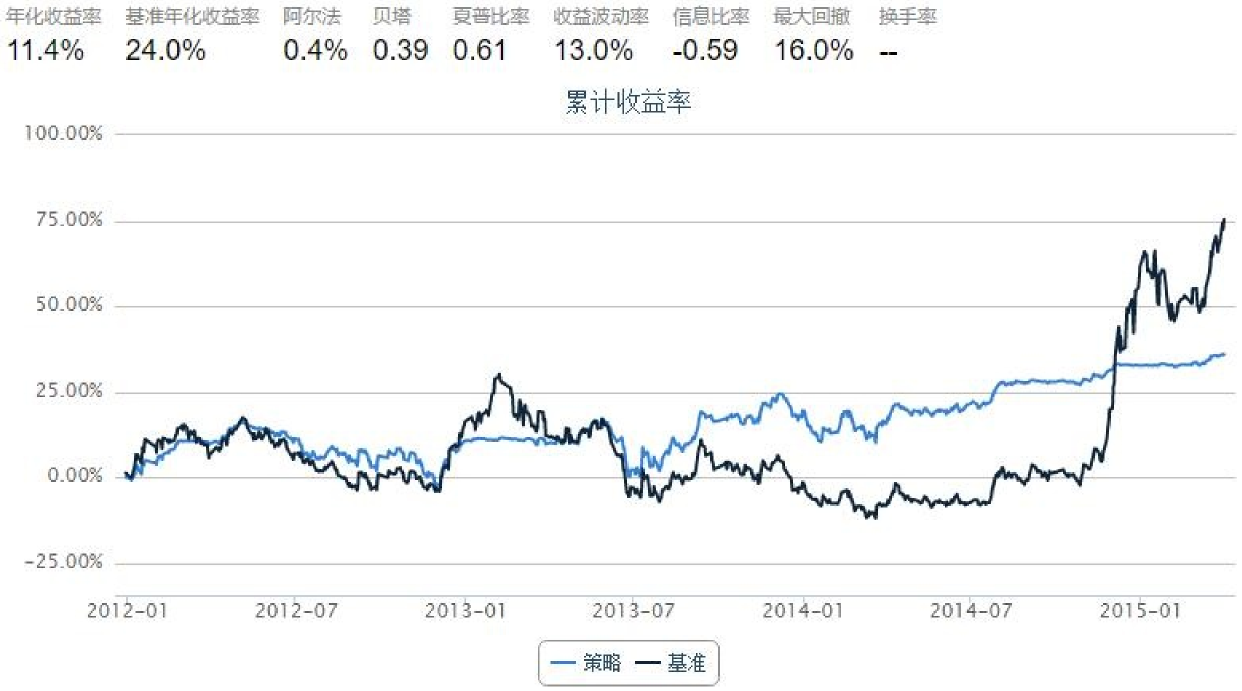
RSI指标策略

> 来源：https://uqer.io/community/share/549ccfd2f9f06c4bb886323d

策略思路

- 使用talib中的RSI函数计算每只股票过去20天的rsi
- 当rsi低于30是买入，高于70时卖出
- 每只股票仓位最多不超过总资金的10%

```
import talib as ta

start = '2011-12-01'
end   = '2015-04-01'

benchmark = 'SH50'
universe = set_universe('SH50')
capital_base = 5000000
longest_history = 21

def initialize(account):
    account.lower_rsi = 30
    account.upper_rsi = 70

def handle_data(account):
    all_close_prices = account.get_attribute_history('closePrice', longest_history)

    rsi, c_price, c_amount = {}, {}, {}
    for stock in account.universe:
        rsi[stock] = ta.RSI(all_close_prices[stock], longest_history-1)[-1]
        c_amount[stock] = account.secpos.get(stock, 0)

    for stock in account.universe:
        max_amount = int(0.1 * account.referencePortfolioValue / account.referencePrice[stock])
        amount = min(int(25000./account.referencePrice[stock]), max_amount - c_amount[stock])
        if   (rsi[stock] < account.lower_rsi) and (c_amount[stock] < max_amount):
            order(stock, amount)
        elif (rsi[stock] > account.upper_rsi) and (c_amount[stock] >  0):
            order_to(stock, 0)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
11.4%	24.0%	0.4%	0.39	0.61	13.0%	-0.59	16.0%	--

4.7 DMI • DMI 指标体系的构建及简单应用

> 来源：https://uqer.io/community/share/561727aaf9f06c4ca92fb5a4

DMI指标又叫动向指标或趋向指标，是通过分析股票价格在涨跌过程中买卖双方力量均衡点的变化情况，即多空双方的力量的变化受价格波动的影响而发生由均衡到失衡的循环过程，从而提供对趋势判断依据的一种技术指标。其由美国技术分析大师威尔斯·威尔德（Wells Wilder）所创造，是一种中长期股市技术分析方法。

DMI指标体系的构建：

```
TR = SUM(MAX(MAX(HIGH - LOW, ABS(HIGH-REF(CLOSE,1))), ABS(LOW -
REF(CLOSE, 1))), N)
HD = HIGH - REF(HIGH, 1)
LD = REF(LOW, 1) - LOW
DMP = SUM(IF(HD>0 AND HD>LD, HD, 0), N)
DMM = SUM(IF(LD>0 AND LD>HD, LD, 0), N)
PDI = DMP*100/TR
MDI = DMM*100/TR
DX = ABS(MDI - PDI)/(MDI + PDI)*100
ADX = MA(ABS(MDI - PDI)/(MDI + PDI)*100, M)
```

其中变量与函数定义如下：

- `CLOSE` ：引用收盘价(在盘中指最新价)
- `HIGH` ：引用最高价
- `LOW` ：引用最低价
- `REF(X, N)` ：引用X在N个周期前的值
- `ABS(X)` ：求X的绝对值
- `MAX(A, B)` ：求A，B中的较大者
- `SUM(X, N)` ：得到X在N周期内的总和
- `IF(C, A, B)` ：如果C成立返回A，否则返回B

此外， `PDI` 简记为 `+DI` ， `MDI` 简记为 `-DI` ；参数： `N=14` (默认)， `M=14` (默认)。

实际上从数学上看， `DX` 或 `ADX` 的构建并不一定需要 `PDI` 与 `MDI` ，有 `DMP` 和 `DMM` 就行了。计算 `PDI` 与 `MDI` 是将指标数值控制在0到100之间。

```
#计算某一天的股票DMP与DMM值
def eq_DMPandDMM(stk_list,current_date,N=14):

    cal = Calendar('China.SSE')
    period = '-' + str(N+1) + 'B'
    begin_date = cal.advanceDate(current_date,period,BizDayConve
ntion.Unadjusted)
    end_date = cal.advanceDate(current_date,'-1B',BizDayConventi
on.Unadjusted)

    eq_hd = {}
    eq_ld = {}
    dmp_sum = 0
    dmm_sum = 0
    eq_dmp = {}
    eq_dmm = {}

    eq_Price = DataAPI.MktEqudAdjGet(secID=stk_list,beginDate=be
gin_date.strftime('%Y%m%d'),endDate=end_date.strftime('%Y%m%d'),
field=['secID','highestPrice','lowestPrice'],pandas="1")

    avaiable_list = eq_Price['secID'].drop_duplicates().tolist()

    eq_Price.set_index('secID',inplace=True)

    for stk in avaiable_list:
        if len(eq_Price.ix[stk]) == (N+1):
            eq_hd[stk] = np.array(eq_Price.ix[stk]['highestPrice'
][1:] - eq_Price.ix[stk]['highestPrice'][:-1])
            eq_ld[stk] = np.array(eq_Price.ix[stk]['lowestPrice'
][:-1] - eq_Price.ix[stk]['lowestPrice'][1:])
            for i in xrange(len(eq_ld[stk])):

                if eq_hd[stk][i] > 0 and eq_hd[stk][i] > eq_ld[s
tk][i]:
                    dmp_sum = dmp_sum + eq_hd[stk][i]
                if eq_ld[stk][i] > 0 and eq_ld[stk][i] > eq_hd[s
tk][i]:
                    dmm_sum = dmm_sum + eq_ld[stk][i]

            eq_dmp[stk] = dmp_sum
            eq_dmm[stk] = dmm_sum
            dmm_sum = 0
            dmp_sum = 0
    return eq_dmp,eq_dmm
```

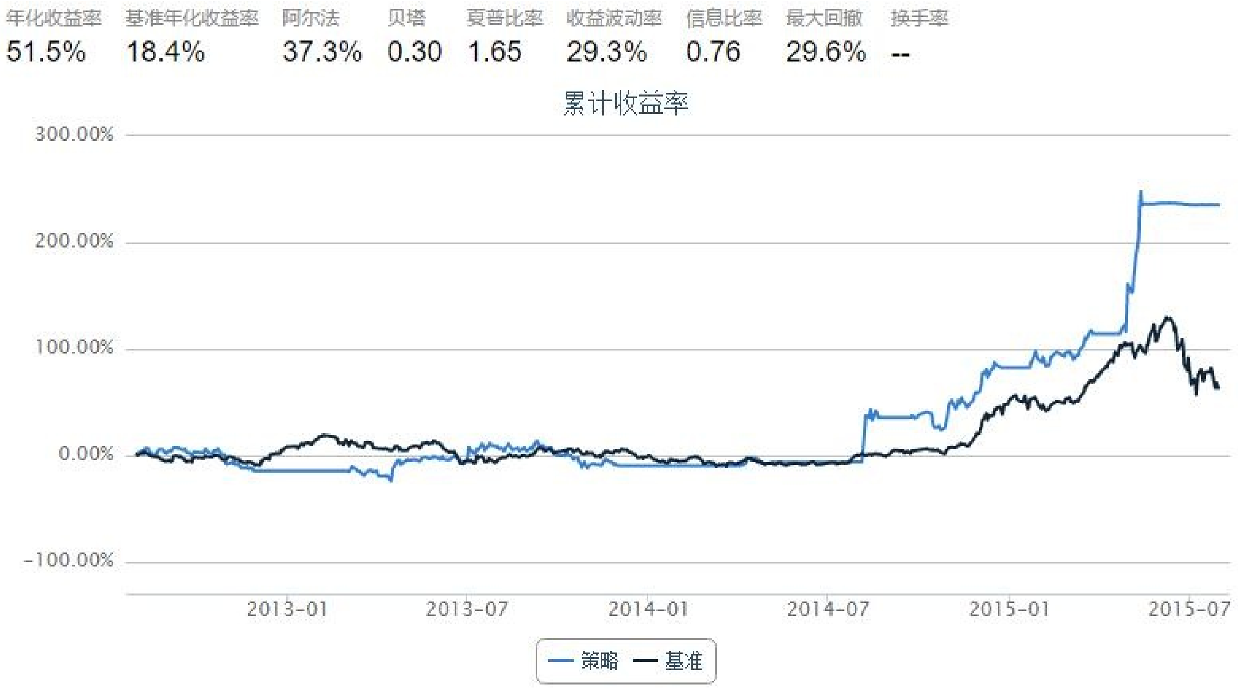
```
#计算某一天股票的TR值
def eq_TR(stk_list,current_date,N=14):

    cal = Calendar('China.SSE')
    period = '-' + str(N+1) + 'B'
    begin_date = cal.advanceDate(current_date,period,BizDayConvention.Unadjusted)
    end_date = cal.advanceDate(current_date,'-1B',BizDayConvention.Unadjusted)

    eq_hl = {} #HIGH - LOW
    eq_hc = {} #HIGH - CLOSE
    eq_lc = {} #LOW - CLOSE
    eq_tr = {}
    tr_sum = 0

    eq_Price = DataAPI.MktEqudAdjGet(secID=stk_list,beginDate=begin_date.strftime('%Y%m%d'),endDate=end_date.strftime('%Y%m%d'),field=['secID','highestPrice','lowestPrice','closePrice'],pandas="1")

    avaiable_list = eq_Price['secID'].drop_duplicates().tolist()

    eq_Price.set_index('secID',inplace=True)

    for stk in avaiable_list:
        if len(eq_Price.ix[stk]) == (N+1):
            eq_hl[stk] = np.array(eq_Price.ix[stk]['highestPrice'][1:] - eq_Price.ix[stk]['lowestPrice'][1:])
            eq_hc[stk] = np.array(eq_Price.ix[stk]['highestPrice'][1:] - eq_Price.ix[stk]['closePrice'][:-1])
            eq_lc[stk] = np.array(eq_Price.ix[stk]['lowestPrice'][:-1] - eq_Price.ix[stk]['closePrice'][1:])
            for i in xrange(len(eq_hl[stk])):

                tr_sum = tr_sum + max(max(eq_hl[stk][i],abs(eq_hc[stk][i])),abs(eq_lc[stk][i]))

            eq_tr[stk] = tr_sum
            tr_sum = 0
    return eq_tr
```

```
#计算某一天股票的ADX
def eq_ADX(stk_list,current_date,N=14):
    cal = Calendar('China.SSE')
    period = '-' + str(N) + 'B'
    begin_date = cal.advanceDate(current_date,period,BizDayConve
ntion.Unadjusted)
    end_date = cal.advanceDate(current_date,'-1B',BizDayConventi
on.Unadjusted)

    timeSeries = cal.bizDatesList(begin_date,end_date)

    eq_adx = {}
    adx_sum = 0

    #初始化eq_adx
    eq_Price = DataAPI.MktEqudAdjGet(secID=stk_list,beginDate=be
gin_date.strftime('%Y%m%d'),endDate=end_date.strftime('%Y%m%d'),
field=['secID','highestPrice','lowestPrice'],pandas="1")

    avaiable_list = eq_Price['secID'].drop_duplicates().tolist()

    eq_Price.set_index('secID',inplace=True)

    for stk in avaiable_list:
        if len(eq_Price.ix[stk]) == N:
            eq_adx[stk] = 0

    #计算ADX
    for i in xrange(len(timeSeries)):
        eq_dmp,eq_dmm = eq_DMPandDMM(stk_list,timeSeries[i],N)
        for stk in eq_dmp:
            if eq_dmp[stk] == 0 and eq_dmm[stk] == 0: #当DMP与DMM
都为零时，认为无趋势DX=0
                pass
            else:
                eq_adx[stk] = eq_adx[stk] + abs(eq_dmp[stk] - eq
_dmm[stk])/(eq_dmp[stk] + eq_dmm[stk])*100

    for stk in eq_adx:
        eq_adx[stk] = eq_adx[stk] / len(timeSeries)
    return eq_adx
```

简单应用：

当 DMP 上穿 DMM 时，意味着，上涨倾向强于下跌倾向，一个买入信号生成。反之则反。而 ADX 用于反映趋向变动的程度，在买入信号时， ADX 伴随上升，则预示股价的涨势可能更强劲。

```
import numpy as np
import pandas as pd
from CAL.PyCAL import *
start = '2012-08-01'                       # 回测起始时间
end = '2015-08-01'                         # 回测结束时间
benchmark = 'HS300'                        # 策略参考标准
universe = set_universe('HS300')  # 证券池，支持股票和基金
capital_base = 1000000                      # 起始资金
freq = 'd'                                 # 策略类型，'d'表示日间
策略使用日线回测，'m'表示日内策略使用分钟线回测
refresh_rate = 20                           # 调仓频率，表示执行han
dle_data的时间间隔，若freq = 'd'时间间隔的单位为交易日，若freq = 'm'时
间间隔为分钟
cal = Calendar('China.SSE')

def initialize(account):                   # 初始化虚拟账户状态
    pass

def handle_data(account):                  # 每个交易日的买入卖出指令


    eq_dmp_now,eq_dmm_now = eq_DMPandDMM(account.universe,accoun
t.current_date,14)
    eq_adx = eq_ADX(account.universe,account.current_date,14)

    yestoday = cal.advanceDate(account.current_date,'-1B',BizDay
Convention.Unadjusted)

    eq_dmp_before,eq_dmm_before = eq_DMPandDMM(account.universe,
yestoday,14)
    eq_adx_before = eq_ADX(account.universe,yestoday,14)

    long_bucket = []
    short_bucket = []

    for stk in account.universe:
        try:
            if eq_dmp_now[stk] > eq_dmm_now[stk] and eq_dmp_befo
re[stk] < eq_dmm_before[stk] and eq_adx[stk] > eq_adx_before[stk
]:
                long_bucket.append(stk)
            else:
                short_bucket.append(stk)
        except:
            pass


    #调仓逻辑是调仓时将所有满足条件的股票等权
    stk_num = len(account.valid_secpos) + len(long_bucket)
    for stk in account.valid_secpos:
        if stk in short_bucket:
            order_to(stk,0)
            stk_num = stk_num - 1
    for stk in account.valid_secpos:
        if stk not in short_bucket:
            order_to(stk,account.referencePortfolioValue/account
.referencePrice[stk]/stk_num)

    for stk in long_bucket:
        if stk not in account.avail_secpos:
            order_to(stk,account.referencePortfolioValue/account
.referencePrice[stk]/stk_num)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
51.5%	18.4%	37.3%	0.30	1.65	29.3%	0.76	29.6%	--

4.8 EMV • EMV 技术指标的构建及应用

简易波动指标（EMV），是为数不多的考虑价量关系的技术指标。它刻画了股价在下跌的过程当中，由于买气不断的萎靡退缩，致使成交量逐渐的减少，EMV 数值也因而尾随下降，直到股价下跌至某一个合理支撑区，捡便宜货的买单促使成交量再度活跃，EMV 数值于是作相对反应向上攀升，当EMV 数值由负值向上趋近于零时，表示部分信心坚定的资金，成功的扭转了股价的跌势，行情不断反转上扬，并且形成另一次的买进讯号。

计算方法：

第一步

$$MID=\frac{TH+TL}{2}-\frac{YH+YL}{2}$$

这里 `TH` 为当天最高价， `TL` 为当天最低价， `YH` 为前日最高价， `YL` 为前日最低价。 `MID > 0` 意味着今天的平均价高于昨天的平均价。

第二步

$$BRO=\frac{VOL}{H-L}$$

其中 `VOL` 代表交易量， `H` 、 `L` 代表同一天的最高价与最低价

第三步

$$EM=\frac{MID}{BRO}$$

第四步

`EMV = EM` 的N日简单移动平均

第五步

`MAEMV = EMV` 的M日简单移动平均

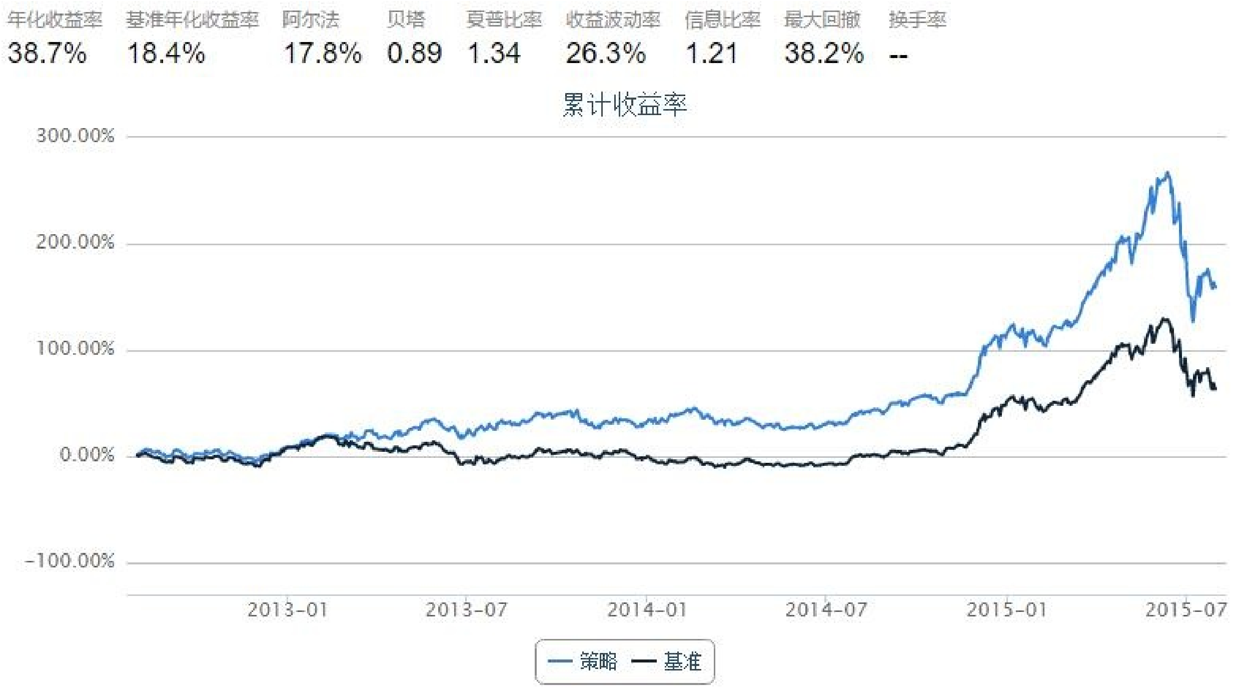
```
def emv(stk_list,current_date,N=14):

    cal = Calendar('China.SSE')
    period = '-' + str(N+1) + 'B'
    begin_date = cal.advanceDate(current_date,period,BizDayConvention.Unadjusted)
    end_date = cal.advanceDate(current_date,'-1B',BizDayConvention.Unadjusted)

    eq_emv = {}
    eq_mid = {}
    eq_bro = {}

    eq_Market = DataAPI.MktEqudAdjGet(secID=stk_list,beginDate=begin_date.strftime('%Y%m%d'),endDate=end_date.strftime('%Y%m%d'),field=['secID','highestPrice','lowestPrice','turnoverVol'],pandas="1")

    avaiable_list = eq_Market['secID'].drop_duplicates().tolist()

    eq_Market.set_index('secID',inplace=True)

    for stk in avaiable_list:
        if len(eq_Market.ix[stk]) == (N+1):
            eq_mid[stk] = (np.array(eq_Market.ix[stk]['highestPrice'][1:] + eq_Market.ix[stk]['lowestPrice'][1:]) - np.array(eq_Market.ix[stk]['highestPrice'][:-1] + eq_Market.ix[stk]['lowestPrice'][:-1]))/2
            eq_bro[stk] = np.array(eq_Market.ix[stk]['turnoverVol'][1:])/np.array(eq_Market.ix[stk]['highestPrice'][1:] + eq_Market.ix[stk]['lowestPrice'][1:])
            eq_emv[stk] = np.mean(eq_mid[stk]/eq_bro[stk])

    return eq_emv
```

```
def maemv(stk_list,current_date,N=14):
    cal = Calendar('China.SSE')
    period = '-' + str(N+1) + 'B'
    end_date = cal.advanceDate(current_date,'-1B',BizDayConvention.Unadjusted)
    start_date =  cal.advanceDate(current_date,period,BizDayConvention.Unadjusted)
    timeSeries = cal.bizDatesList(start_date, end_date)
    eq_maemv = {}
    #初始化eq_maemv字典
    eq_emv = emv(stk_list,end_date,N)
    for stk in eq_emv:
        eq_maemv[stk] = 0
    #仅调用N次emv函数
    for i in xrange(len(timeSeries)):
        eq_emv = emv(stk_list,timeSeries[i],N)
        for stk in eq_emv:
            eq_maemv[stk] = eq_maemv[stk] + eq_emv[stk]

    for stk in eq_maemv:
        eq_maemv[stk] = eq_maemv[stk]/N

    return eq_maemv
```

`EMV` 指标基本用法

`EMV` 在0 以下表示弱势，在0 以上表示强势； `EMV` 由负转正应买进，由正转负应卖出。

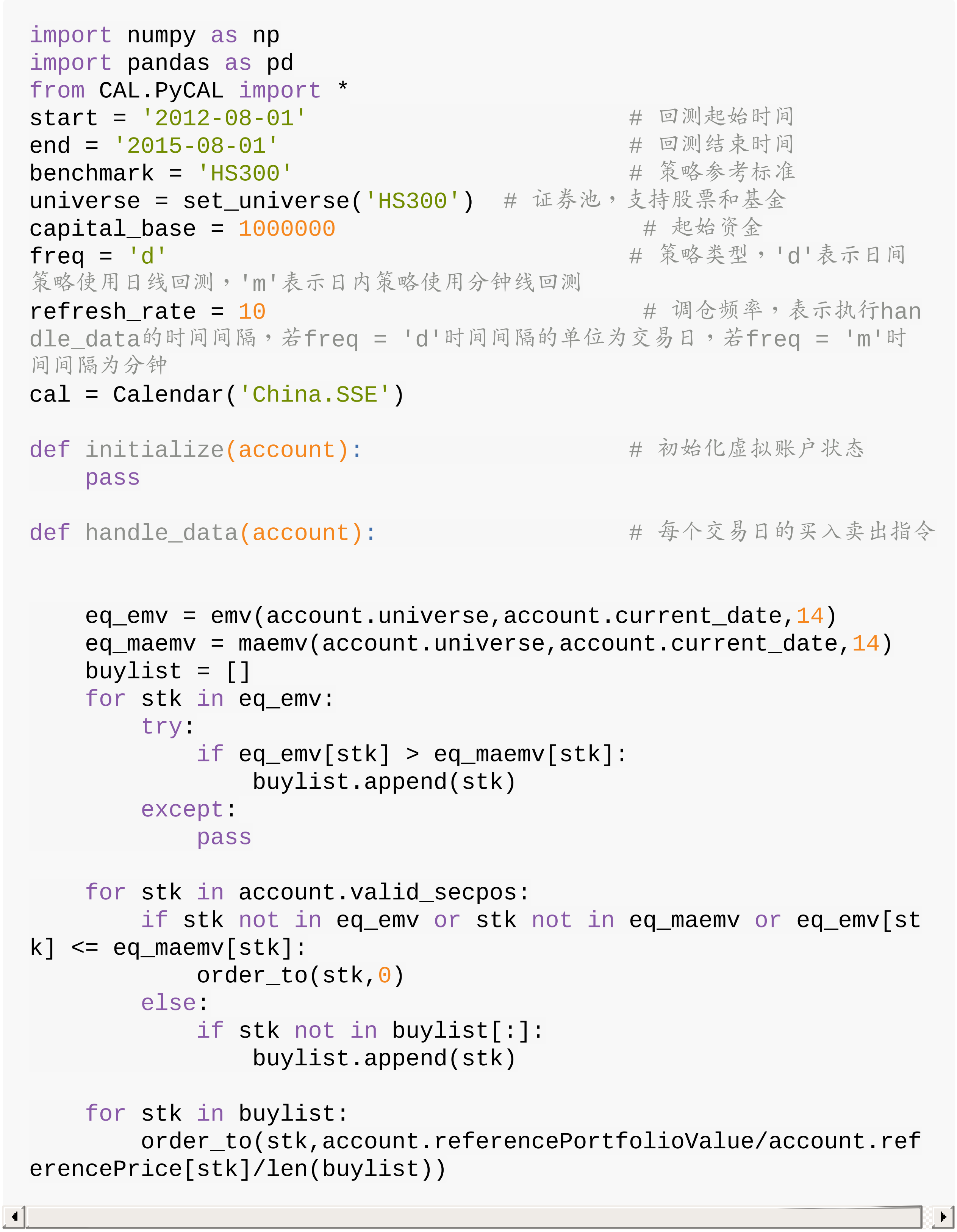
```
import numpy as np
import pandas as pd
from CAL.PyCAL import *
start = '2012-08-01'                       # 回测起始时间
end = '2015-08-01'                         # 回测结束时间
benchmark = 'HS300'                        # 策略参考标准
universe = set_universe('HS300')  # 证券池，支持股票和基金
capital_base = 1000000                      # 起始资金
freq = 'd'                                 # 策略类型，'d'表示日间策略使用日线回测，'m'表示日内策略使用分钟线回测
refresh_rate = 10                           # 调仓频率，表示执行handle_data的时间间隔，若freq = 'd'时间间隔的单位为交易日，若freq = 'm'时间间隔为分钟
cal = Calendar('China.SSE')

def initialize(account):                   # 初始化虚拟账户状态
    pass

def handle_data(account):                  # 每个交易日的买入卖出指令


    eq_emv = emv(account.universe,account.current_date,N=14)
    buylist = []
    for stk in eq_emv:
        if eq_emv[stk] > 0:
            buylist.append(stk)

    for stk in account.valid_secpos:
        if stk not in eq_emv or eq_emv[stk] <= 0:
            order_to(stk,0)
        else:
            if stk not in buylist[:]:
                buylist.append(stk)

    for stk in buylist:
        order_to(stk,account.referencePortfolioValue/account.referencePrice[stk]/len(buylist))
```



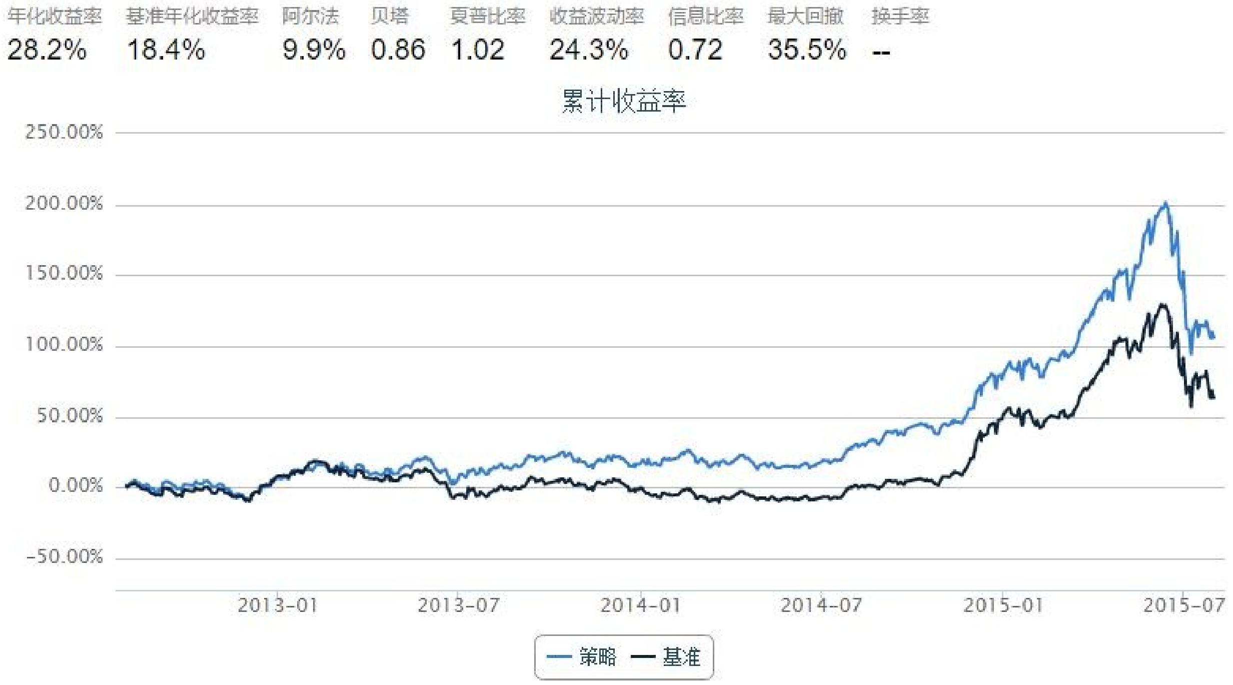
年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
38.7%	18.4%	17.8%	0.89	1.34	26.3%	1.21	38.2%	--

`EMV` 结合 `MAEMV` 使用

`EMV` 上穿 `MAEMV` 则买入，`EMV` 下穿 `MAEMV` 则卖出。



```
import numpy as np
import pandas as pd
from CAL.PyCAL import *
start = '2012-08-01'                       # 回测起始时间
end = '2015-08-01'                         # 回测结束时间
benchmark = 'HS300'                        # 策略参考标准
universe = set_universe('HS300')  # 证券池，支持股票和基金
capital_base = 1000000                      # 起始资金
freq = 'd'                                 # 策略类型，'d'表示日间
策略使用日线回测，'m'表示日内策略使用分钟线回测
refresh_rate = 10                           # 调仓频率，表示执行han
dle_data的时间间隔，若freq = 'd'时间间隔的单位为交易日，若freq = 'm'时
间间隔为分钟
cal = Calendar('China.SSE')

def initialize(account):                   # 初始化虚拟账户状态
    pass

def handle_data(account):                  # 每个交易日的买入卖出指令


    eq_emv = emv(account.universe,account.current_date,14)
    eq_maemv = maemv(account.universe,account.current_date,14)
    buylist = []
    for stk in eq_emv:
        try:
            if eq_emv[stk] > eq_maemv[stk]:
                buylist.append(stk)
        except:
            pass

    for stk in account.valid_secpos:
        if stk not in eq_emv or stk not in eq_maemv or eq_emv[st
k] <= eq_maemv[stk]:
            order_to(stk,0)
        else:
            if stk not in buylist[:]:
                buylist.append(stk)

    for stk in buylist:
        order_to(stk,account.referencePortfolioValue/account.ref
erencePrice[stk]/len(buylist))
```

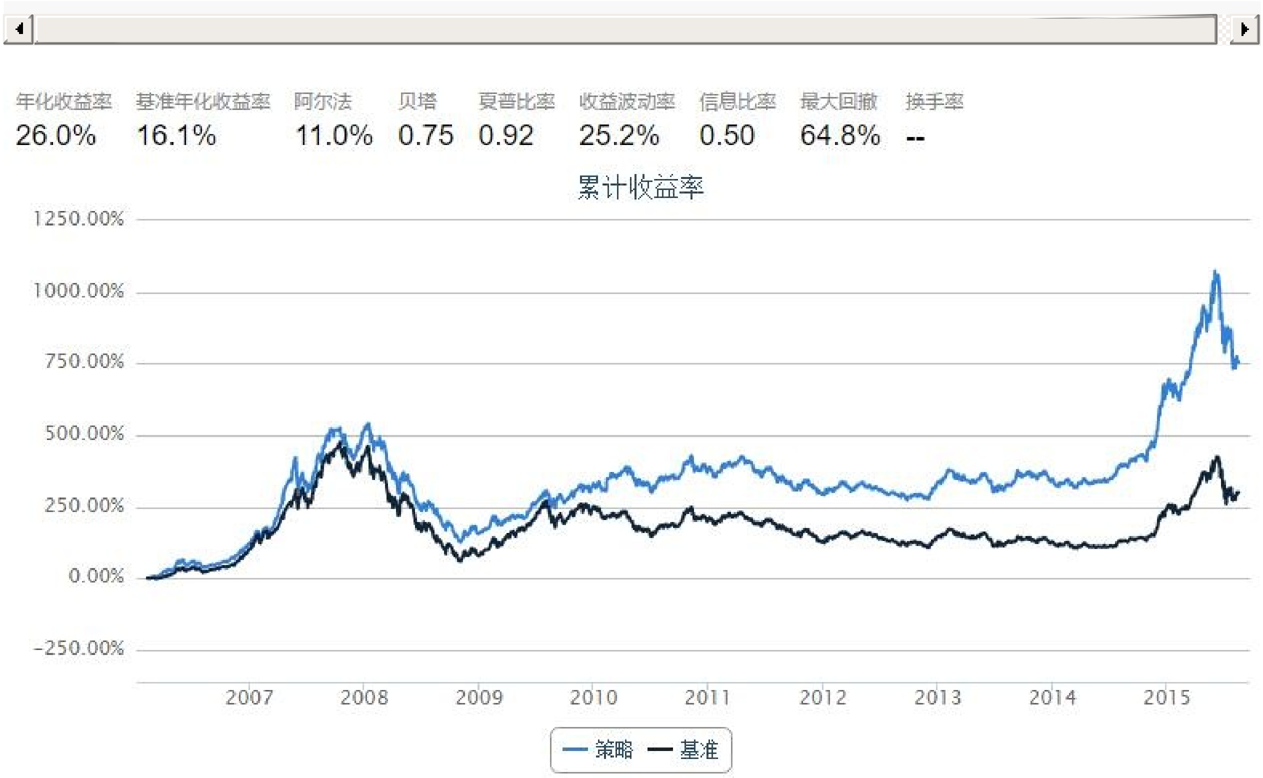


年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
28.2%	18.4%	9.9%	0.86	1.02	24.3%	0.72	35.5%	--

4.9 KDJ • KDJ 策略

> 来源：https://uqer.io/community/share/55d20b3bf9f06c91f818c6ac

```
import numpy as np
import pandas as pd
from pandas import DataFrame
import talib as ta

start = '2006-01-01'                      # 回测起始时间
end = '2015-08-17'                        # 回测结束时间
benchmark = 'HS300'                       # 策略参考标准
universe = set_universe('HS300')
capital_base = 100000                      # 起始资金
refresh_rate = 1                         # 调仓频率，即每 refres
h_rate 个交易日执行一次 handle_data() 函数
longest_history=20
MA=[5,10,20,30,60,120]                     #移动均线参数

def initialize(account):
    account.kdj=[]

def handle_data(account):

    # 每个交易日的买入卖出指令

    sell_pool=[]
    hist = account.get_history(longest_history)
        #data=DataFrame(hist['600006.XSHG'])
    stock_pool,all_data=Get_all_indicators(hist)
    pool_num=len(stock_pool)
    if account.secpos==None:
        print 'null'
        for i in stock_pool:
            buy_num=int(float(account.cash/pool_num)/account.ref
erencePrice[i]/100.0)*100
            order(i, buy_num)
    else:

        for x in account.valid_secpos:
            if all_data[x].iloc[-1]['closePrice']<all_data[x].il
oc[-1]['ma1'] and (all_data[x].iloc[-1]['ma1']-all_data[x].iloc[
-1]['closePrice'])/all_data[x].iloc[-1]['ma1']>0.05 :
                sell_pool.append(x)
                order_to(x, 0)



        if account.cash>500 and pool_num>0:
            try:
                sim_buy_money=float(account.cash)/pool_num
                for l in stock_pool:
                    #print sim_buy_money,account.referencePrice[
l]

                    buy_num=int(sim_buy_money/account.referenceP
rice[l]/100.0)*100

                    #buy_num=10000
                    order(l, buy_num)
            except Exception as e:
                #print e
                pass



def Get_kd_ma(data):
    indicators={}
    #计算kd指标
    indicators['k'],indicators['d']=ta.STOCH(np.array(data['high
Price']),np.array(data['lowPrice']),np.array(data['closePrice'])
,\
    fastk_period=9,slowk_period=3,slowk_matype=0,slowd_period=3,
slowd_matype=0)
    indicators['ma1']=pd.rolling_mean(data['closePrice'], MA[0])
    indicators['ma2']=pd.rolling_mean(data['closePrice'], MA[1])
    indicators['ma3']=pd.rolling_mean(data['closePrice'], MA[2])
    indicators['ma4']=pd.rolling_mean(data['closePrice'], MA[3])
    indicators['ma5']=pd.rolling_mean(data['closePrice'], MA[4])
    indicators['closePrice']=data['closePrice']
    indicators=pd.DataFrame(indicators)
    return indicators

def Get_all_indicators(hist):
    stock_pool=[]
    all_data={}
    for i in hist:
        try:
            indicators=Get_kd_ma(hist[i])
            all_data[i]=indicators
        except Exception as e:
            #print 'error:%s'%e
            pass
        if indicators.iloc[-2]['k']<indicators.iloc[-2]['d'] and
 indicators.iloc[-1]['k']>indicators.iloc[-2]['d']:
            stock_pool.append(i)
        elif indicators.iloc[-1]['k']>=10 and indicators.iloc[-1
]['d']<=20 and indicators.iloc[-1]['k']>indicators.iloc[-2]['k']
and indicators.iloc[-2]['k']<indicators.iloc[-3]['k']:
            stock_pool.append(i)
    return stock_pool,all_data
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
26.0%	16.1%	11.0%	0.75	0.92	25.2%	0.50	64.8%	--

4.10 CMO

CMO 策略模仿练习 1

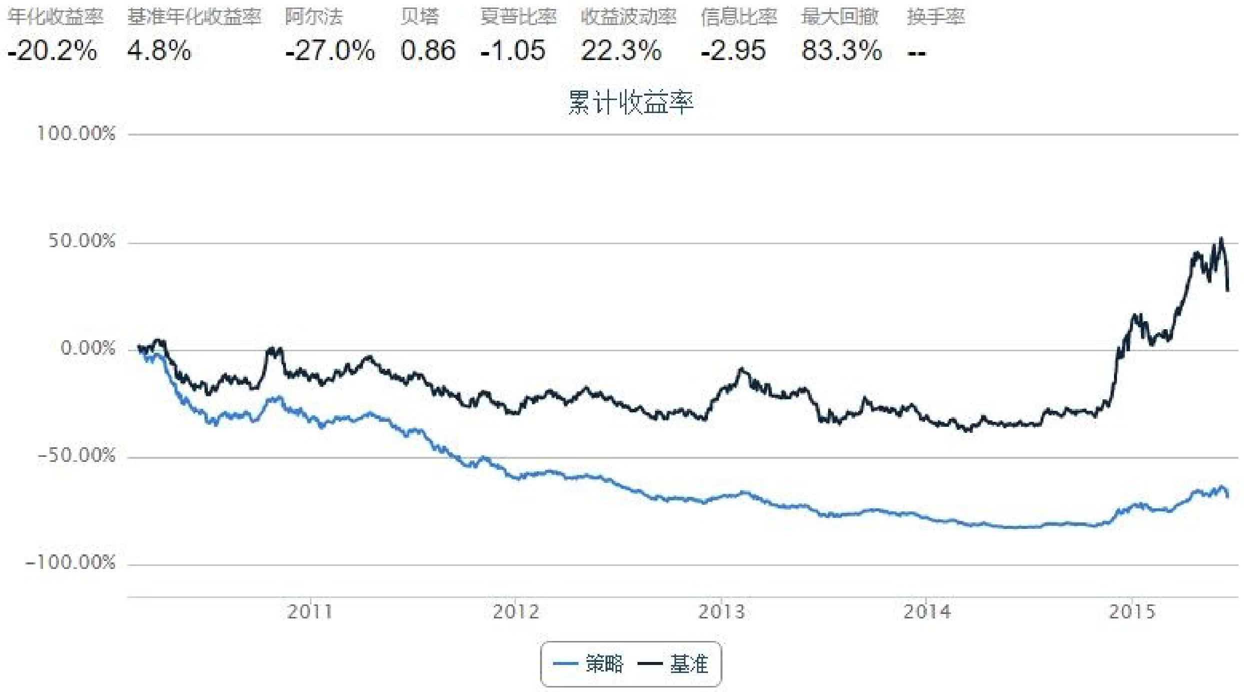
> 来源：https://uqer.io/community/share/55b4c44ef9f06c91f818c5ea

```
import numpy as np

start='2010-01-01'
end='2015-06-20'
benchmark='sh50'
universe=set_universe('SH50')
capital_base=1000000
window=35   # 参数，CMO指标计算周期
def initialize(account):
    pass
def handle_data(account):
    clp=account.get_attribute_history("closeprice",window)
    prc=account.get_attribute_history("precloseprice",window)
    p=account.referenceprice
    # 计算CMO
    CMO= {}
    for s in account.universe:
        diff=clp(s)-prc(s)
        u=sum(n for n in diff if n>0)
        d=sum(-n for n in diff if n<0)
        if u+d==0: continue
        CMO[s]=(u-d)/(u+d)*100
    # 根据CMO卖出目前持有股票
    v=account.cash
    for s,a in account.valid_secpos.items():
        if cmo.get(s,0)<0 and s in account.universe:
            order_to(s,0)
            v+=a*p[s]
    # 根据CMO确定买入列表
    buylist= []
    for s in account_universe:
        if cmo.get(s,0)<0 and not np.isnan(p[s]) and s not in ac
count.valid_secpos:
            buylist.append(s)
    if v > account.referencePortfolioValue * 0.33: # 为了避免调仓
过于频繁，仅当可用现金超过账户市值1/3时买入
        for s in buylist:
            order(s, v/len(buylist)/ p[s])

----------------------------------------------------------------
-----------
ValueError                                Traceback (most recent
 call last)
<mercury-input-8-189e24327e9d> in <module>()
     54                             slippage        = slippage,
     55                             refresh_rate    = refresh_ra
te,
---> 56                             freq            = freq)
     57     perf = quartz.perf_parse(bt, quartz_acct)
     58     perf_temp = {}

python2.7/site-packages/quartz/backtest.py in backtest_generator
(start, end, benchmark, universe, capital_base, initialize, hand
le_data, csvs, security_base, commission, slippage, refresh_rate
, freq, *args, **kwargs)
    279         sim_params = env.SimulationParameters(start, end
, benchmark, universe, capital_base, security_base, csvs)
    280
--> 281         idxmap_all, data_all = data_generator.get_daily_
data(sim_params)
    282         data_gen = data_generator.get_daily_data_generat
or(data_all)
    283         account = env.Account(sim_params, strg, idxmap_a
ll, data_all, commission, slippage)

python2.7/site-packages/quartz/sim_condition/data_generator.py i
n get_daily_data(sim_params, fq)
     26     trading_days = sim_params.trading_days
     27     idxmap_date = dict(zip(trading_days, range(len(tradi
ng_days))))
---> 28     idxmap_bm, data_bm = load_benchmark_data(sim_params.
benchmark, trading_days)
     29
     30     stocks, funds = univ_divide(sim_params.universe)

python2.7/site-packages/quartz/data/benchmarks.py in load_benchm
ark_data(symbol, trading_days)
    135                 data.append(line)
    136     else:
--> 137         raise ValueError("Please verify your benchmark I
D!")
    138
    139     # 数据缺失

ValueError: Please verify your benchmark ID!
```

CMO策略模仿练习2

> 来源：https://uqer.io/community/share/55b4e523f9f06c91f918c5db

```
import numpy as np

start='2010-01-01'
end='2015-06-20'
benchmark='SH50'
universe=set_universe('SH50')
capital_base=1000000
window=35    # 参数，CMO指标计算周期
def initialize(account):
    pass
def handle_data(account):
    clp=account.get_attribute_history("closePrice",window)
    prc=account.get_attribute_history("preClosePrice",window)
    p=account.referencePrice
    # 计算CMO
    CMO= {}
    for s in account.universe:
        diff=clp[s]-prc[s]
        u=sum(n for n in diff if n>0)
        d=sum(-n for n in diff if n<0)
        if u+d==0: continue
        CMO[s]=(u-d)/(u+d)*100
    # 根据CMO卖出目前持有股票
    v=account.cash
    for s,a in account.valid_secpos.items():
        if CMO.get(s,0)<0 and s in account.universe:
            order_to(s,0)
            v+=a*p[s]
    # 根据CMO确定买入列表
    buylist= []
    for s in account.universe:
        if CMO.get(s,0)<0 and not np.isnan(p[s]) and s not in account.valid_secpos:
            buylist.append(s)
    if v > account.referencePortfolioValue * 0.33: # 为了避免调仓过于频繁，仅当可用现金超过账户市值1/3时买入
        for s in buylist:
            order(s, v/len(buylist)/ p[s])
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
-20.2%	4.8%	-27.0%	0.86	-1.05	22.3%	-2.95	83.3%	--

[技术指标] CMO

> 来源：https://uqer.io/community/share/5590d8bdf9f06cb5604f1881

指标介绍

- CMO(Chande Momentum Oscillator)动量震荡指标是由Tushar S. Chande提出的类似于RSI的指标
- `CMOn` 是一个n天滚动指标，在这n天中的第i天计算每天的 收盘价 - 前收盘价 ，如果为正则赋给 `upi` （ `dni` 为0），为负则将绝对值赋给 `dni` （ `upi` 为0）
- 其计算公式为：

$$CMO_n = \frac{\sum_{i=1}^{n} up_i - \sum_{i=1}^{n} dn_i}{\sum_{i=1}^{n} up_i + \sum_{i=1}^{n} dn_i} * 100$$

策略思路

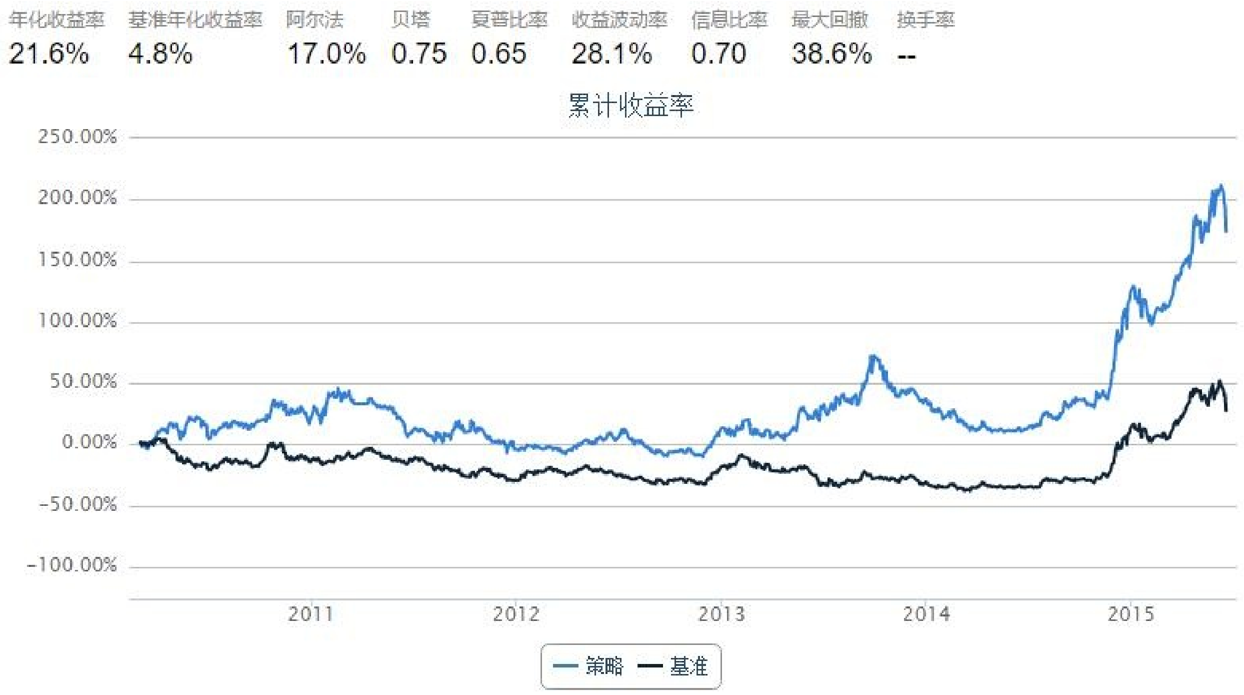
- 计算上证50成分股当中所有股票过去n天的CMO
- CMO大于0时买入，小于0时卖出
- 根据一定的调仓原则进行调仓，细节见代码

可进一步挖掘的点

- 考虑CMO的形态，如上/下穿0线作为买卖信号
- 扩大股票池范围，观察CMO与股票池的关系，比如区分大小盘股观察CMO的有效性
- 股票权重的分配方式
- 其他调仓原则

先来看看最简单的情况

```
import numpy as np

start = '2010-01-01'
end = '2015-06-20'
benchmark = 'SH50'
universe = set_universe('SH50')
capital_base = 1000000

window = 35    # 参数，CMO指标计算周期

def initialize(account):
    pass

def handle_data(account):
    clp = account.get_attribute_history('closePrice', window)
    prc = account.get_attribute_history('preClosePrice', window)
    p = account.referencePrice

    # 计算CMO
    CMO = {}
    for s in account.universe:
        diff = clp[s] - prc[s]
        u = sum(n for n in diff if n > 0)
        d = sum(-n for n in diff if n < 0)
        if u + d == 0: continue
        CMO[s] = (u - d) / (u + d) * 100

    # 根据CMO卖出目前持有股票
    v = account.cash
    for s,a in account.valid_secpos.items():
        if CMO.get(s, 0) < 0 and s in account.universe:
            order_to(s, 0)
            v += a * p[s]

    # 根据CMO确定买入列表
    buylist = []
    for s in account.universe:
        if CMO.get(s, 0) > 0 and not np.isnan(p[s]) and s not in
 account.valid_secpos:
            buylist.append(s)

    # 根据买入列表和可用现金买入股票
    if v > account.referencePortfolioValue * 0.33: # 为了避免调仓
过于频繁，仅当可用现金超过账户市值1/3时买入
        for s in buylist:
            order(s, v / len(buylist) / p[s])
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
21.6%	4.8%	17.0%	0.75	0.65	28.1%	0.70	38.6%	--

上面的策略实现了策略思路所表述的意思，使用了非常简单的调仓原则，其表现还不错

但是，其中的关键参数 `window` 为什么设置为 35 呢？

这当然不是拍脑袋拍出来的，而是通过参数调试出来的:

```
window   annualized_return   sharpe   max_drawdown
  10         0.0052          -0.1377     0.4944
  15         0.1817           0.5284     0.3388
  20         0.1925           0.5850     0.3404
  25         0.1914           0.5835     0.2956
  30         0.2094           0.6053     0.4516
  35         0.2156           0.6468     0.3856
  40         0.0610           0.0970     0.5264
  45         0.1980           0.5728     0.4872
  50         0.1730           0.4632     0.5166
```

从上面的调试结果中可以看到，当 `window = 35` 时，夏普和最大回撤相对而言最好，因此有了最上面的那个策略

然而调试完了 `window` ，这个策略就没有优化空间了吗？不，我们还可以根据这个策略的表现来分析一下这个策略的缺陷在哪里，并加以改进

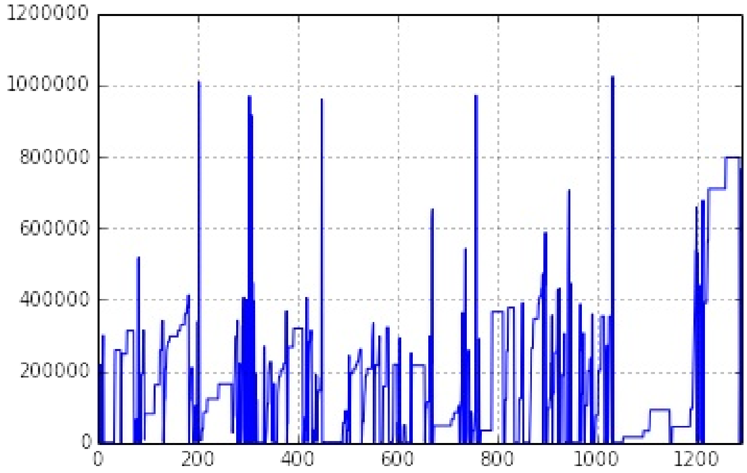
因为该策略的调仓原则是买入所有产生的信号，并没有对持仓进行限制，这会造成两个方面的影响：

1. 仓位中的股票可能会有很多只，这样资金会比较分散，削弱信号的效果
2. 如果买入信号比较少，而卖出信号比较多的话，现金的利用率会比较低

那么到底是否存在上述问题呢？我们可以通过最大持仓数量和现金走势来加以判断

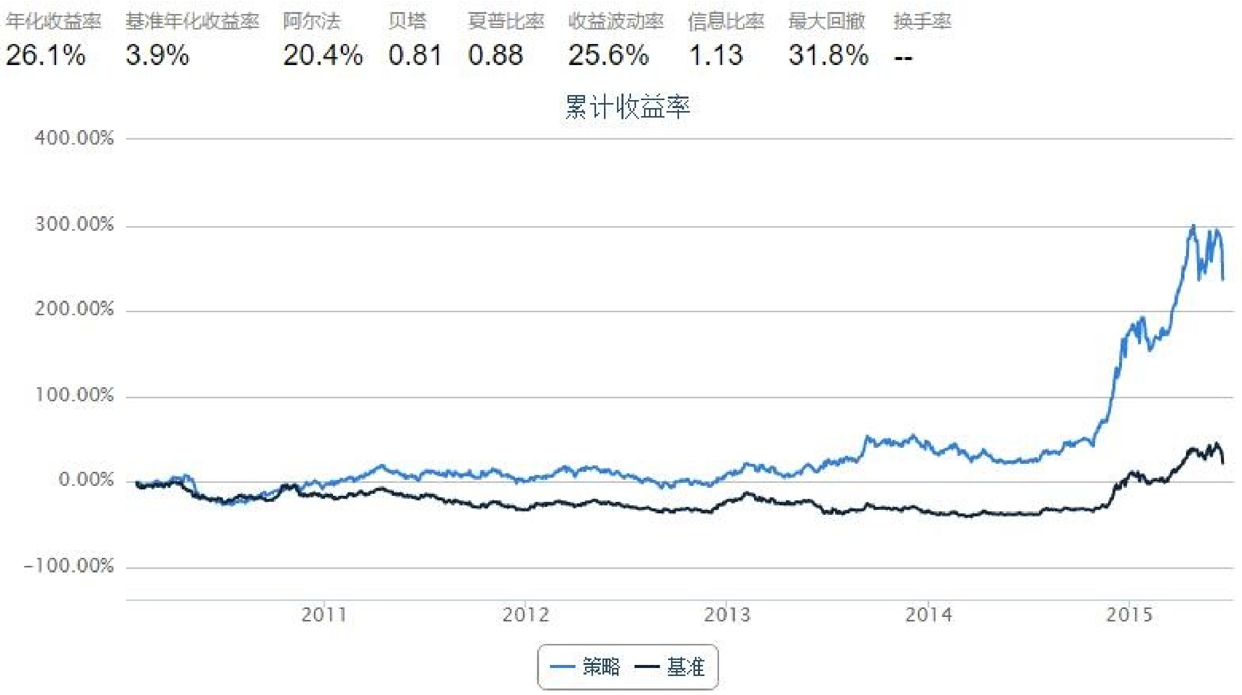
```
x = map(len, bt['security_position'].tolist())
max(x)

42
```

```
bt.cash.plot()

<matplotlib.axes.AxesSubplot at 0x6053cd0>
```



从上面的两个cell中可以看出这两个问题还是比较明显的。为了解决这两个问题，我们对策略进行优化：一是限制最大持仓位10只股票，二是每次卖出的现金都平均分配给目前仓位中的股票和即将买入的股票

```
import numpy as np
from heapq import nlargest

start = '2010-01-01'
end = '2015-06-20'
benchmark = 'SH50'
universe = set_universe('SH50')
capital_base = 1000000

max_n  = 10     # 参数，最大持仓数量
window = 15     # 参数，CMO指标计算周期

def initialize(account):
    pass

def handle_data(account):
    clp = account.get_attribute_history('closePrice', window)
    prc = account.get_attribute_history('preClosePrice', window)
    p = account.referencePrice
    # 计算CMO
    CMO = {}
    for s in account.universe:
        diff = clp[s] - prc[s]
        u = sum(n for n in diff if n > 0)
        d = sum(-n for n in diff if n < 0)
        if u + d == 0: continue
        CMO[s] = (u - d) / (u + d) * 100

    # 根据CMO卖出目前持有股票
    n = len(account.valid_secpos)
    sellist = []
    for s,a in account.valid_secpos.items():
        if CMO.get(s, 0) < 0 and s in account.universe:
            order_to(s, 0)
            n -= 1
            sellist.append(s)

    if n >= max_n: # 如果超过最大持仓，则不买入
        return

    # 根据CMO确定买入列表
    buylist = []
    for s in account.universe:
        if CMO.get(s, 0) > 0 and not np.isnan(p[s]) and s not in
 account.valid_secpos:
            buylist.append(s)

    # 根据最大持仓数量确定买入列表数量，按CMO排序选较大的部分
    if len(buylist) + n > max_n:
        buylist = nlargest(max_n - n, buylist, key=CMO.get)

    # 将资金重新分配到新买入的与已持有的股票中
    buylist += [s for s in account.valid_secpos if s not in sell
ist]
    amount = {}
    for s in buylist:
        amount[s] = account.referencePortfolioValue / len(buylis
t) / p[s] - account.valid_secpos.get(s, 0)

    # 根据应调数量买卖股票，先卖出后买入
    for s in sorted(amount, key=amount.get):
        order(s, amount[s])
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
26.1%	3.9%	20.4%	0.81	0.88	25.6%	1.13	31.8%	--

```
window    annualized_return    sharpe    max_drawdown
  10           0.0830           0.1789      0.4101
  15           0.2606           0.8783      0.3182
  20           0.1726           0.5180      0.3689
  25           0.2190           0.6762      0.3508
  30           0.2067           0.6376      0.3725
  35           0.1676           0.5040      0.3550
  40           0.1416           0.4086      0.4478
  45           0.1927           0.5926      0.4001
  50           0.1183           0.3215      0.4030
```

从上面的图表可以看出其表现相比最初的策略有了不少改善。其中 `window = 15` 是最适合目前调仓原则的参数。

但是这些优化的初衷是为了解决股票数量和资金利用率的问题，我们仍然通过最大持仓数量和现金走势来判断

```
x = map(len, bt['security_position'].tolist())
max(x)

10
```

```
bt.cash.plot()

<matplotlib.axes.AxesSubplot at 0x5a48fd0>
```

!](img/8DjEAXQMXIa20AAAAASUVORK5CYII=.png)

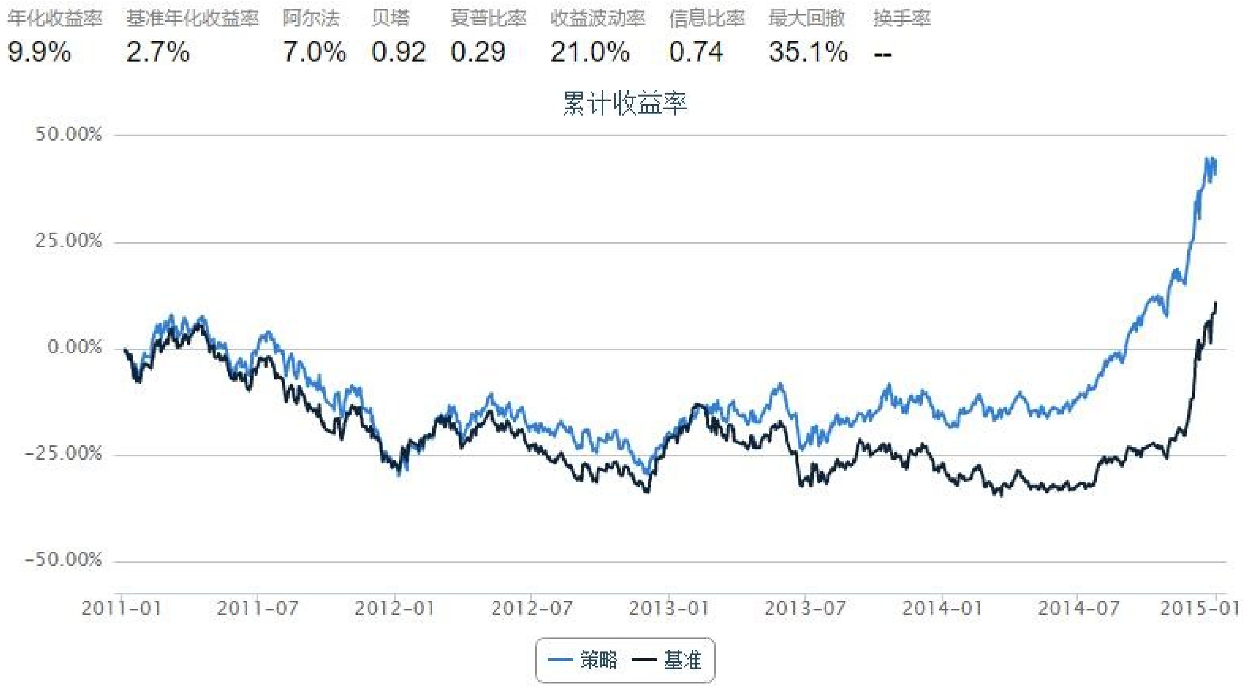
以上是对CMO这个技术指标进行的一些简单的回测，并且针对策略本身的特点进行了一定的优化。在最前面列出了一些可挖掘的点，如果想进行更深入的研究还是有很多东西可以做的。

4.11 FPC • FPC 指标选股

> 来源：https://uqer.io/community/share/55a4bcb4f9f06c6dd1e17f19

```
import pandas as pd
fields = ['tradeDate', 'secID', 'OperatingProfitRatio', 'ROE', '
InventoryTRate', 'ARTRate', 'TotalAssetsTRate', 'OperatingRevenu
eGrowRate', 'OperatingProfitGrowRate', 'NetProfitGrowRate', 'Net
AssetGrowRate', 'DebtsAssetRatio', 'CurrentRatio', 'QuickRatio']

weights = [15.0, 20.0, 0.07, 0.07, 0.06, 8.0, 8.0, 8.0, 8.0, 0.07
, 0.07, 0.07]

def FPCCalculator(x):
    sumRes = 0
    for name, weight in zip(fields[2:], weights):
        sumRes += x[name] * weight
    return sumRes

start = '2011-01-01'                       # 回测起始时间
end = '2015-01-01'                         # 回测结束时间
benchmark = 'HS300'                        # 策略参考标准
universe = set_universe('HS300', start)  # 证券池，回测支持股票和基金

capital_base = 100000000                  # 起始资金
refresh_rate = 60                         # 调仓频率，即每 refresh
_rate 个交易日执行一次 handle_data() 函数

def initialize(account):                   # 初始化虚拟账户状态
    pass

def handle_data(account):                  # 每个交易日的买入卖出指令


    hist = account.get_attribute_history('closePrice',1)
    print account.current_date
    today = account.current_date
    cal = Calendar('China.SSE')
    referenceDate = cal.advanceDate(today, '-1b', BizDayConventi
on.Preceding).strftime('%Y%m%d')
    res = pd.DataFrame()
    for stock in account.universe:
        data = DataAPI.MktStockFactorsDateRangeGet(secID=stock,
beginDate=referenceDate, endDate=referenceDate, field=fields)
        res = res.append(data.dropna())
    res = res.reset_index(drop=True)
    res = res.set_index('secID')
    ind = DataAPI.EquIndustryGet(secID=res.index.values, industr
yVersionCD='010303', intoDate=referenceDate)
```

```
    ind = ind.set_index('secID')
    res['industry'] = ind.industryName1
    res['score'] = res.apply(FPCCalculator, 1)
    grouped = res[['industry', 'score']].groupby('industry')

    buyList = []
    # 只选行业中股票个数大于5个行业分类
    for key, group in grouped:
        if len(group) >= 5:
            group['score'] = group['score'] - group['score'].mea
n()
            group = group.sort('score', ascending=False)
            buyList.append((key, group.index[0], group.score[0])
)

    # 先卖出
    for stock in account.valid_secpos:
        order_to(stock, 0)

    # 再买入
    totalValue = account.referencePortfolioValue
    cashForEachStock = totalValue / (len(buyList) + 1)

    for key, name, score in buyList:
        amount = int(cashForEachStock / hist[name] / 100) * 100
        print "{0:<20s}: {1:s}, {2:f}, {3:d}".format(key, name,
score, amount)
        order(name, amount)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
9.9%	2.7%	7.0%	0.92	0.29	21.0%	0.74	35.1%	--

```
2011-01-05 00:00:00
```

```
交运设备        : 600418.XSHG, 277.876645, 640800
交通运输        : 600350.XSHG, 33.614389, 1559300
信息服务        : 600271.XSHG, 19.515397, 249400
公用事业        : 600863.XSHG, 261.666837, 1671100
农林牧渔        : 000061.XSHE, 15.004222, 658100
化工              : 000422.XSHE, 187.995211, 475200
医药生物        : 002007.XSHE, 67.818000, 135500
商业贸易        : 002024.XSHE, 12.686739, 495900
家用电器        : 000651.XSHE, 6.061078, 822600
建筑建材        : 600528.XSHG, 33.795187, 709800
房地产           : 600266.XSHG, 44.695256, 637600
有色金属        : 600547.XSHG, 417.523861, 120400
机械设备        : 000039.XSHE, 35.414604, 295900
采掘              : 000933.XSHE, 46.406670, 398200
食品饮料        : 000858.XSHE, 19.308545, 203000
2011-04-08 00:00:00
交运设备        : 000927.XSHE, 128.086565, 783700
交通运输        : 601111.XSHG, 40.021770, 609200
信息服务        : 600271.XSHG, 18.897618, 284900
公用事业        : 600863.XSHG, 264.298228, 1622300
农林牧渔        : 000061.XSHE, 15.004222, 649600
化工              : 000422.XSHE, 11.132274, 421800
医药生物        : 600216.XSHG, 15.584448, 335300
商业贸易        : 002024.XSHE, 10.290995, 481100
家用电器        : 000651.XSHE, 5.837902, 623700
建筑建材        : 601117.XSHG, 19.892445, 838900
房地产           : 000897.XSHE, 32.173353, 1113200
有色金属        : 600547.XSHG, 949.914380, 118200
机械设备        : 000157.XSHE, 30.560292, 526900
采掘              : 600188.XSHG, 19.870549, 179800
钢铁              : 000709.XSHE, 130.212328, 1520200
食品饮料        : 600300.XSHG, 100.285020, 1076800
2011-07-05 00:00:00
交运设备        : 600104.XSHG, 30.278670, 359900
交通运输        : 600029.XSHG, 159.996161, 804300
信息服务        : 600271.XSHG, 18.554824, 256400
公用事业        : 600900.XSHG, 30.288765, 964700
农林牧渔        : 000876.XSHE, 16.770247, 314800
化工              : 000059.XSHE, 52.574421, 440800
医药生物        : 600518.XSHG, 14.594103, 904600
商业贸易        : 002024.XSHE, 10.591486, 474500
家用电器        : 000651.XSHE, 10.057383, 595800
建筑建材        : 600585.XSHG, 19.600357, 231400
房地产           : 000631.XSHE, 40.536041, 1370900
有色金属        : 600111.XSHG, 109.240872, 255400
机械设备        : 000039.XSHE, 57.594695, 290000
采掘              : 601001.XSHG, 15.730951, 350000
钢铁              : 600516.XSHG, 407.299490, 551700
食品饮料        : 600132.XSHG, 17.648762, 105700
2011-09-28 00:00:00
交运设备        : 600104.XSHG, 21.918292, 381800
交通运输        : 600221.XSHG, 63.193351, 1851600
信息服务        : 600271.XSHG, 14.206672, 211700
```

```
公用事业          : 600674.XSHG, 25.243519, 1662700
农林牧渔          : 000876.XSHE, 24.967820, 279200
化工                : 000059.XSHE, 25.787124, 568800
医药生物          : 600518.XSHG, 15.576307, 793800
商业贸易          : 002024.XSHE, 8.403357, 528100
家用电器          : 000651.XSHE, 9.330746, 631900
建筑建材          : 600585.XSHG, 306.156492, 316400
房地产             : 000631.XSHE, 31.578629, 1362600
有色金属          : 600111.XSHG, 200.004115, 290700
机械设备          : 000039.XSHE, 53.399231, 348500
采掘                : 600188.XSHG, 28.178807, 194500
钢铁                : 000898.XSHE, 2738.223747, 1032200
食品饮料          : 600809.XSHG, 18.768502, 156500
2011-12-28 00:00:00
交运设备          : 600104.XSHG, 17.063323, 385900
交通运输          : 000089.XSHE, 35.753107, 1126000
信息服务          : 600588.XSHG, 14.781967, 473400
公用事业          : 000685.XSHE, 46.858354, 805300
农林牧渔          : 000876.XSHE, 26.901091, 281600
化工                : 000422.XSHE, 26.961147, 393900
医药生物          : 600518.XSHG, 16.171002, 839200
商业贸易          : 600694.XSHG, 34.616035, 158900
建筑建材          : 600585.XSHG, 34.849863, 325600
房地产             : 000631.XSHE, 36.914683, 1842100
有色金属          : 000807.XSHE, 263.555174, 958000
机械设备          : 600031.XSHG, 29.957499, 410900
采掘                : 601001.XSHG, 12.862060, 383400
钢铁                : 600307.XSHG, 26.649657, 1276300
食品饮料          : 600300.XSHG, 25.273663, 1180800
2012-03-30 00:00:00
交运设备          : 600104.XSHG, 18.909886, 426800
交通运输          : 000089.XSHE, 35.166259, 1278400
信息服务          : 600271.XSHG, 16.981212, 314400
公用事业          : 000685.XSHE, 48.707469, 871200
化工                : 000422.XSHE, 28.601842, 413000
医药生物          : 600518.XSHG, 15.576615, 867900
商业贸易          : 600694.XSHG, 32.895471, 212700
建筑建材          : 600585.XSHG, 20.132624, 371300
房地产             : 000402.XSHE, 36.743106, 936400
有色金属          : 000612.XSHE, 3797.404448, 760600
机械设备          : 600031.XSHG, 27.121428, 466400
采掘                : 601001.XSHG, 11.694587, 432800
钢铁                : 600010.XSHG, 81.043473, 1722100
食品饮料          : 600300.XSHG, 24.462883, 813200
2012-07-02 00:00:00
交运设备          : 600066.XSHG, 28.154927, 670000
交通运输          : 000089.XSHE, 39.622036, 1276300
信息服务          : 600588.XSHG, 34.751230, 504100
公用事业          : 000685.XSHE, 35.456503, 856900
化工                : 600251.XSHG, 24.002774, 567000
医药生物          : 600196.XSHG, 14.593144, 512900
商业贸易          : 600694.XSHG, 17.596119, 168100
家用电器          : 000100.XSHE, 162.404051, 2648600
```

```
建筑建材      : 600585.XSHG, 22.135445, 368200
房地产        : 600663.XSHG, 59.464286, 431300
有色金属      : 600456.XSHG, 186.015637, 274800
机械设备      : 600031.XSHG, 28.231175, 399400
采掘          : 600123.XSHG, 8.183554, 307700
钢铁          : 600307.XSHG, 31.022309, 1541900
食品饮料      : 600519.XSHG, 23.296690, 25300
2012-09-24 00:00:00
交运设备      : 600150.XSHG, 86.708491, 234500
交通运输      : 000089.XSHE, 39.383480, 1313100
信息服务      : 600271.XSHG, 80.422510, 337800
公用事业      : 000685.XSHE, 35.061598, 963300
化工          : 000422.XSHE, 20.686581, 413900
医药生物      : 600518.XSHG, 18.695275, 634200
商业贸易      : 600694.XSHG, 34.618859, 136600
家用电器      : 000651.XSHE, 14.583746, 537700
建筑建材      : 601117.XSHG, 11.177778, 773300
房地产        : 000046.XSHE, 907.969889, 1432000
有色金属      : 600456.XSHG, 57.602414, 290200
机械设备      : 000157.XSHE, 27.342064, 635600
采掘          : 600123.XSHG, 9.754217, 290300
钢铁          : 000825.XSHE, 43.268790, 1516400
食品饮料      : 002304.XSHE, 19.892648, 63800
2012-12-24 00:00:00
交运设备      : 600066.XSHG, 26.100632, 614600
交通运输      : 601107.XSHG, 116.863379, 1556200
信息服务      : 600271.XSHG, 48.157093, 378300
公用事业      : 600008.XSHG, 58.545444, 1219100
化工          : 000422.XSHE, 16.611969, 460200
医药生物      : 600518.XSHG, 25.176992, 777700
商业贸易      : 600694.XSHG, 24.983091, 170800
家用电器      : 000651.XSHE, 18.691066, 433900
建筑建材      : 601117.XSHG, 15.471286, 647900
房地产        : 600895.XSHG, 294.473466, 754600
有色金属      : 601899.XSHG, 147.629408, 1435300
机械设备      : 600312.XSHG, 325.920975, 680700
采掘          : 600123.XSHG, 9.751122, 266000
钢铁          : 000825.XSHE, 88.335062, 1438800
食品饮料      : 600519.XSHG, 21.102387, 25900
2013-03-28 00:00:00
交运设备      : 600150.XSHG, 38.141684, 255100
交通运输      : 601107.XSHG, 124.017399, 1667100
信息服务      : 600271.XSHG, 37.765484, 379400
公用事业      : 600011.XSHG, 70.219799, 848200
农林牧渔      : 000876.XSHE, 126.213409, 422400
化工          : 600309.XSHG, 20.922440, 309400
医药生物      : 600518.XSHG, 23.752800, 585800
商业贸易      : 002024.XSHE, 17.212139, 796100
家用电器      : 000651.XSHE, 17.939662, 408700
建筑建材      : 601117.XSHG, 15.867114, 552900
房地产        : 600895.XSHG, 294.490930, 811000
有色金属      : 601899.XSHG, 144.656562, 1639000
机械设备      : 600312.XSHG, 304.909865, 581300
```

```
采掘              : 600583.XSHG, 55.129969, 802800
钢铁              : 000825.XSHE, 123.683741, 1554900
食品饮料        : 600519.XSHG, 21.180904, 36000
2013-07-02 00:00:00
交运设备        : 000625.XSHE, 114.535214, 552700
交通运输        : 000089.XSHE, 41.398885, 1385600
信息服务        : 600804.XSHG, 7.150407, 422300
公用事业        : 601139.XSHG, 271.722329, 581200
化工              : 600309.XSHG, 34.933967, 311000
医药生物        : 600518.XSHG, 8.240444, 498800
商业贸易        : 600694.XSHG, 25.571651, 184100
家用电器        : 000651.XSHE, 11.454882, 447500
建筑建材        : 601668.XSHG, 20.426365, 1637400
房地产           : 000046.XSHE, 35.341020, 1079900
有色金属        : 601899.XSHG, 139.891359, 2220300
机械设备        : 600312.XSHG, 97.669502, 512400
采掘              : 600583.XSHG, 33.244083, 748900
钢铁              : 600019.XSHG, 93.875140, 1296600
食品饮料        : 000895.XSHE, 17.037947, 193200
2013-09-26 00:00:00
交运设备        : 000625.XSHE, 142.122140, 558000
交通运输        : 000089.XSHE, 44.337186, 1395800
信息服务        : 600804.XSHG, 13.150377, 303700
公用事业        : 600886.XSHG, 36.863455, 1607000
化工              : 600309.XSHG, 25.442594, 367000
医药生物        : 600196.XSHG, 11.686152, 438500
商业贸易        : 600655.XSHG, 6.632367, 651100
建筑建材        : 600528.XSHG, 25.614124, 1033400
房地产           : 002244.XSHE, 48.302567, 1420900
有色金属        : 600489.XSHG, 876.140851, 584300
机械设备        : 600312.XSHG, 118.116765, 585800
采掘              : 600583.XSHG, 36.158939, 793600
钢铁              : 000825.XSHE, 10.851036, 2157800
食品饮料        : 600519.XSHG, 13.133058, 47600
2013-12-26 00:00:00
交运设备        : 000625.XSHE, 89.622325, 495200
交通运输        : 000089.XSHE, 41.127229, 1326900
信息服务        : 600804.XSHG, 18.757929, 406600
公用事业        : 600027.XSHG, 111.988911, 2078600
化工              : 600352.XSHG, 27.120396, 908000
医药生物        : 600161.XSHG, 20.161769, 262600
商业贸易        : 600500.XSHG, 9.151041, 810200
建筑建材        : 600585.XSHG, 40.048519, 347000
房地产           : 000046.XSHE, 31.540756, 1305800
有色金属        : 600497.XSHG, 26.848059, 602300
机械设备        : 600312.XSHG, 49.129871, 573500
采掘              : 600583.XSHG, 38.696328, 755100
钢铁              : 000825.XSHE, 1.620364, 2231100
食品饮料        : 600519.XSHG, 15.203512, 50900
2014-03-28 00:00:00
交通运输        : 000089.XSHE, 57.025247, 1288900
公用事业        : 600886.XSHG, 29.920482, 1177500
化工              : 600352.XSHG, 34.851128, 663100
```

```
医药生物          : 600664.XSHG, 8.847229, 861800
商业贸易          : 600694.XSHG, 10.188886, 203400
国防军工          : 000768.XSHE, 15.368459, 584700
建筑装饰          : 600820.XSHG, 47.412512, 994900
房地产             : 000046.XSHE, 38.909941, 1148300
有色金属          : 600497.XSHG, 28.310791, 573100
机械设备          : 600835.XSHG, 16.508934, 257400
汽车                : 000625.XSHE, 86.397775, 552900
电气设备          : 600312.XSHG, 35.927563, 402100
计算机             : 600100.XSHG, 8.826704, 530600
采掘                : 600583.XSHG, 58.070124, 669400
钢铁                : 000825.XSHE, 1.617123, 2131300
食品饮料          : 600519.XSHG, 16.279505, 35700
2014-06-26 00:00:00
交通运输          : 000089.XSHE, 37.823741, 1315200
公用事业          : 600674.XSHG, 34.934140, 842600
化工                : 600352.XSHG, 29.599356, 680000
医药生物          : 002007.XSHE, 15.935101, 196000
商业贸易          : 000061.XSHE, 858.871568, 512100
国防军工          : 600118.XSHG, 6.488620, 284900
家用电器          : 600839.XSHG, 71.112703, 1583000
建筑装饰          : 601186.XSHG, 8.365248, 1104900
房地产             : 600239.XSHG, 564.094438, 1540200
有色金属          : 600111.XSHG, 29.080872, 375500
机械设备          : 000528.XSHE, 15.807318, 841400
汽车                : 000951.XSHE, 1596.494204, 421700
电气设备          : 002202.XSHE, 12.785842, 534100
计算机             : 000021.XSHE, 14.081832, 839100
采掘                : 600583.XSHG, 48.024740, 672600
钢铁                : 601003.XSHG, 28.696162, 2283900
食品饮料          : 600519.XSHG, 22.505981, 34000
2014-09-19 00:00:00
交通运输          : 000089.XSHE, 37.736841, 1403600
公用事业          : 600674.XSHG, 25.079282, 787100
化工                : 600688.XSHG, 188.636319, 1673400
医药生物          : 000623.XSHE, 17.549571, 363500
商业贸易          : 000061.XSHE, 123.977950, 555100
国防军工          : 600118.XSHG, 6.782743, 273900
建筑装饰          : 601186.XSHG, 5.159384, 1216400
房地产             : 600239.XSHG, 6068.206329, 1379400
有色金属          : 000969.XSHE, 46.723918, 616600
机械设备          : 000039.XSHE, 14.453034, 414400
汽车                : 000625.XSHE, 25.819080, 486400
电气设备          : 002202.XSHE, 33.252376, 570900
计算机             : 000021.XSHE, 36.036769, 946700
采掘                : 601001.XSHG, 52.806971, 961500
钢铁                : 600005.XSHG, 40.192879, 2705800
食品饮料          : 600519.XSHG, 26.104634, 39300
2014-12-19 00:00:00
交通运输          : 000089.XSHE, 27.584352, 1245200
公用事业          : 600674.XSHG, 53.839932, 800900
化工                : 600352.XSHG, 40.059302, 891300
医药生物          : 000623.XSHE, 15.455563, 234000
```

```
商业贸易    : 000061.XSHE, 52.779162, 640700
国防军工    : 600879.XSHG, 3.075249, 484500
家用电器    : 000100.XSHE, 9.096006, 2101200
建筑装饰    : 600068.XSHG, 6.882331, 929000
房地产       : 600239.XSHG, 67.767563, 1517300
有色金属    : 601168.XSHG, 132.989485, 894800
机械设备    : 000039.XSHE, 19.252980, 348100
汽车          : 000625.XSHE, 21.574376, 476200
电气设备    : 002202.XSHE, 47.553600, 613000
计算机       : 600588.XSHG, 10.384352, 336200
采掘          : 600583.XSHG, 41.950386, 818900
钢铁          : 600005.XSHG, 46.425391, 2349100
食品饮料    : 600132.XSHG, 30.739120, 472600
```

4.12 Chaikin Volatility

嘉庆离散指标测试

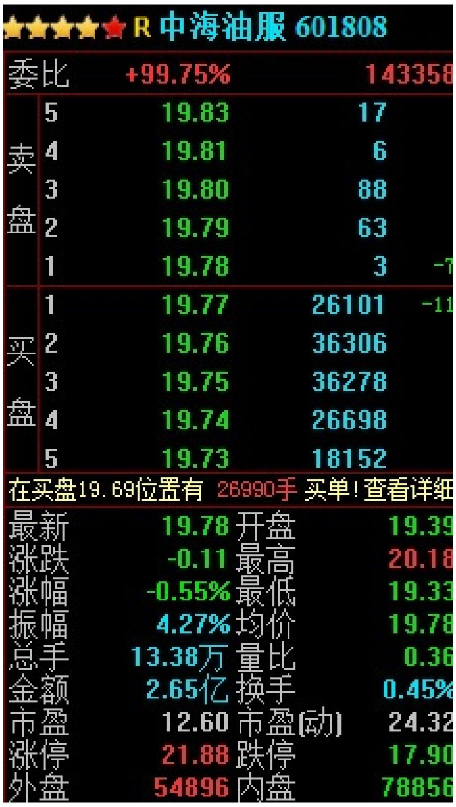
> 来源：https://uqer.io/community/share/55595229f9f06c6c7104f8b3

```
import numpy as np
start = datetime(2011, 1, 1)
end   = datetime(2015, 4, 27)
benchmark = 'HS300'
universe  = set_universe('HS300')
capital_base = 100000
short_history = 30
longest_history = 60

pos_pieces = 10
enter_window = 20
exit_window = 10
N = 4

def initialize(account):
    account.postion_size_hold = {}
    for stk in universe:
        account.postion_size_hold[stk] = 0

def handle_data(account):
    for stock in account.universe:
        cnt_price = account.referencePrice[stock]
        a1 = account.get_attribute_history('closePrice', longest_history)[stock]-account.get_attribute_history('lowPrice', longest_history)[stock]
        b1 = account.get_attribute_history('closePrice', longest_history)[stock]-account.get_attribute_history('highPrice',longest_history)[stock]
        c1 = account.get_attribute_history('highPrice',longest_history)[stock]-account.get_attribute_history('lowPrice',longest_history)[stock]
        d1 = account.get_attribute_history('turnoverVol', longest_history)[stock]
        adl = ((((a1)-(b1))/(c1)))*d1
        a2 = account.get_attribute_history('closePrice', short_history)[stock]-account.get_attribute_history('lowPrice', short_history)[stock]
        b2 = account.get_attribute_history('closePrice', short_history)[stock]-account.get_attribute_history('highPrice',short_history)[stock]
        c2 = account.get_attribute_history('highPrice',short_history)[stock]-account.get_attribute_history('lowPrice',short_history)[stock]
        d2 = account.get_attribute_history('turnoverVol', short_history)[stock]
        ads = ((((a2)-(b2))/(c2)))*d2

        mean_cp1 = adl.mean()
        mean_cp2 = ads.mean()

        flag = mean_cp1 - mean_cp2
        if flag > 0 and account.postion_size_hold[stock]<N:
            order_to(stock, capital_base/pos_pieces/cnt_price/N)
            account.postion_size_hold[stock] += 1
        elif flag < 0 :
            order_to(stock, 0)
            account.postion_size_hold[stock] = 0
```

4.13 委比 • 实时计算委比

> 来源：https://uqer.io/community/share/55bf0426f9f06c91fc18c62f

最近几个交易日，大盘又是一片惨淡，然而我们发现郭嘉队并没有猛拉指数，转而是对部分股票托底。比如中海油服：



★★★★★ R 中海油服 601808

委比		+99.75%	143358	
卖盘	5	19.83	17	
	4	19.81	6	
	3	19.80	88	
	2	19.79	63	
	1	19.78	3	-7
买盘	1	19.77	26101	-11
	2	19.76	36306	
	3	19.75	36278	
	4	19.74	26698	
	5	19.73	18152	

在买盘19.69位置有 26990手 买单！查看详细

最新	19.78	开盘	19.39
涨跌	-0.11	最高	20.18
涨幅	-0.55%	最低	19.33
振幅	4.27%	均价	19.78
总手	13.38万	量比	0.36
金额	2.65亿	换手	0.45%
市盈	12.60	市盈(动)	24.32
涨停	21.88	跌停	17.90
外盘	54896	内盘	78856

风格的转变，意味着郭嘉队转攻为守，让我们看看都有哪些股票值得郭嘉队托底

```
# 常量准备
import pandas as pd
from datetime import datetime as dt
from pandas import DataFrame, Series
today = dt.today().strftime('%Y%m%d')    # 获得今天的日期

# DataAPI取所有A股
stocks = DataAPI.EquGet(equTypeCD='A',listStatusCD='L',field='secID,nonrestfloatA',pandas="1")
universe = stocks['secID'].tolist()     # 转变为list格式，以便和DataAPI中的格式符合

# 取所有A股的最新行情
bidask_fields = ['bidBook_volume%s' %i for i in xrange(1, 6)] + ['askBook_volume%s' %i for i in xrange(1, 6)]
fields = ['shortNM','lastPrice','bidBook','askBook','suspension']
```

```
def get_data():
    data = DataFrame()
    for i in range(0,len(universe),300):    # 原则上可以性取完的，但是试验中作者发现会报错，估计是运算量太大，所以这里分批次取，每次300个
        t = DataAPI.MktTickRTSnapshotGet(securityID=universe[i:min(i+300,len(universe))],field=fields,pandas="1")
        tmp = DataFrame()
        tmp['secID'] = t['ticker']+'.'+t['exchangeCD']
        tmp[['shortNM','suspension'] + bidask_fields] =t[['shortNM', 'suspension']+bidask_fields]
        data = pd.concat([data,tmp],axis=0)    # 数据拼接

    # 去掉当日停牌的股票
    data['nonrestfloatA'] = stocks['nonrestfloatA']
    data = data[data['suspension']==0]

    data = data[(data['bidBook_volume1']>0).values & (data['askBook_volume1']>0).values]


    # 去掉没有涨停板的股票
    data['bidBook_volume'] = sum([data['bidBook_volume%s' %i] for i in xrange(1,6)])
    data['askBook_volume'] = sum([data['askBook_volume%s' %i] for i in xrange(1,6)])

    # 计算委比
    data['rate'] = data['bidBook_volume']/(data['askBook_volume']+data['bidBook_volume'])*100   #百分之几
    data = data.sort(columns='rate',ascending=False).reset_index()
    data.drop('index',axis=1,inplace=True)

    # 重命名
    data = data[['secID', 'shortNM', 'bidBook_volume', 'askBook_volume', 'rate']]
    data.columns = ['代码','简称','买量','卖量','委比']
    return data

get_data().head(50)
```

	代码	简称	买量	卖量	委比
0	601336.XSHG	新华保险	14310780	24700	99.827700
1	600485.XSHG	信威集团	2086200	4100	99.803856
2	600188.XSHG	兖州煤业	38015538	98746	99.740921
3	601808.XSHG	中海油服	16845821	44700	99.735355
4	600362.XSHG	江西铜业	20343390	56935	99.720911

5	600958.XSHG	东方证券	4069182	12972	99.682227
6	600340.XSHG	华夏幸福	12797790	54800	99.573627
7	000951.XSHE	中国重汽	1964684	9600	99.513748
8	002304.XSHE	洋河股份	2943430	15200	99.486249
9	600276.XSHG	恒瑞医药	2230015	12500	99.442590
10	000538.XSHE	云南白药	1798287	10200	99.435993
11	002143.XSHE	印纪传媒	2813406	16335	99.422739
12	000800.XSHE	一汽轿车	9174799	56199	99.391193
13	601958.XSHG	金钼股份	20460168	127809	99.379206
14	000513.XSHE	丽珠集团	881306	5600	99.368591
15	603288.XSHG	海天味业	6049140	39340	99.353862
16	300202.XSHE	聚龙股份	1145291	7600	99.340788
17	601800.XSHG	中国交建	29915478	208100	99.309179
18	600403.XSHG	大有能源	8768385	61500	99.303502
19	601225.XSHG	陕西煤业	57038023	406943	99.291595
20	601098.XSHG	中南传媒	7902047	58400	99.266373
21	601226.XSHG	华电重工	4604601	35200	99.241347
22	600600.XSHG	青岛啤酒	1887900	15673	99.176654
23	601688.XSHG	华泰证券	11673583	97299	99.173392
24	000895.XSHE	双汇发展	17718527	154080	99.137899
25	601555.XSHG	东吴证券	14508868	127897	99.126194
26	300003.XSHE	乐普医疗	2355404	20971	99.117521
27	600429.XSHG	三元股份	6761469	61600	99.097180
28	000333.XSHE	美的集团	9583724	89633	99.073403
29	600004.XSHG	白云机场	3333050	32000	99.049048
30	600637.XSHG	东方明珠	2020400	19420	99.047955
31	600741.XSHG	华域汽车	15412841	150590	99.032411
32	600166.XSHG	福田汽车	14485200	146445	98.999121
33	002594.XSHE	比亚迪	2961400	29967	98.998217

34	000750.XSHE	国海证券	13826732	146800	98.949442
35	600664.XSHG	哈药股份	7654217	81506	98.946369
36	601186.XSHG	中国铁建	24479602	265911	98.925417
37	600519.XSHG	贵州茅台	894254	9741	98.922450
38	601901.XSHG	方正证券	11906346	132861	98.896431
39	002081.XSHE	金螳螂	6156885	69107	98.890024
40	600660.XSHG	福耀玻璃	3706300	44600	98.810952
41	000869.XSHE	张裕A	3073749	37483	98.795236
42	002431.XSHE	棕榈园林	1556637	19100	98.787869
43	002221.XSHE	东华能源	2483500	30600	98.782865
44	000685.XSHE	中山公用	8005948	100200	98.763901
45	002122.XSHE	天马股份	5874886	74210	98.752584
46	000600.XSHE	建投能源	8612256	109140	98.748595
47	002663.XSHE	普邦园林	7862104	101141	98.729902
48	600823.XSHG	世茂股份	7544949	97251	98.727448
49	002252.XSHE	上海莱士	1819900	23600	98.719826

可以看到里面不乏贵州茅台，美的集团，张裕A等十几倍市盈率的现金牛企业。这也就反映了个股正处在严重两极分化的过程，郭嘉队和主力更亲睐于优质蓝筹。

由于这里计算委比时，只是针对买卖五档行情，所以有一定的失眞，而且在行情软件上按照委比排序，效果更直观。那为什么还要搞得这么复杂呢？

因为不想错过发生过的委比结果，因为还想对当天整个的委比情况做分析。

下面的代码通过一个 `while` 循环，每隔10秒打印一次委比前30的股票名称：

```
import time
import datetime

full_data = DataFrame()

while 1:
    now = datetime.datetime.now()
    if now.hour>=15:
        break

    data = DataFrame()
    time_str =  '%2s%2s%2s' %(now.hour, now.minute, now.second)
    try:
        data = get_data()[:30]
        data['time'] = time_str
    except Exception,e:
        print e

    if data.empty:
        continue

    full_data = full_data.append(data)
    print now, ', '.join([ e for e in data['简称'].values])

    time.sleep(10)

2015-08-03 14:01:36.452456 广发证券, 张　裕Ａ, 江铃汽车, 比亚迪, 兖州
煤业, 云南白药, 海天味业, 信威集团, 东华能源, 中集集团, 美邦服饰, 万华化学
, 华电国际, 申万宏源, 陕国投Ａ, 环旭电子, 华域汽车, 一汽轿车, 哈药股份, 陕
西煤业, 聚龙股份, 日出东方, 誉衡药业, 格力电器, 深圳燃气, 徐工机械, 新 和
成, 建投能源, 康恩贝, 驰宏锌锗
2015-08-03 14:01:48.711193 广发证券, 兖州煤业, 张　裕Ａ, 江铃汽车, 云
南白药, 中集集团, 海天味业, 格力电器, 中国西电, 比亚迪, 贵人鸟, 美邦服饰,
东华能源, 万华化学, 哈药股份, 华电国际, 申万宏源, 贵州茅台, 陕国投Ａ, 华域
汽车, 陕西煤业, 日出东方, 誉衡药业, 聚龙股份, 新 和 成, 深圳燃气, 建投能源
, 驰宏锌锗, 一汽轿车, 国海证券
2015-08-03 14:02:01.886604 广发证券, 兖州煤业, 张　裕Ａ, 中集集团, 东
吴证券, 江铃汽车, 华电国际, 海天味业, 云南白药, 中国西电, 万华化学, 申万宏
源, 国海证券, 比亚迪, 东华能源, 互动娱乐, 哈药股份, 日出东方, 华域汽车, 环
旭电子, 双汇发展, 誉衡药业, 格力电器, 陕国投Ａ, 深圳燃气, 东方能源, 新 和
成, 一汽轿车, 驰宏锌锗, 聚龙股份
2015-08-03 14:02:12.446392 中集集团, 广发证券, 云南白药, 兖州煤业, 张
 裕Ａ, 东吴证券, 江铃汽车, 丽珠集团, 信威集团, 金 融 街, 海天味业, 华电国
际, 双汇发展, 互动娱乐, 哈药股份, 万华化学, 上海医药, 华域汽车, 新华保险,
日出东方, 国海证券, 贵人鸟, 冠农股份, 申万宏源, 驰宏锌锗, 誉衡药业, 深圳燃
气, 上海莱士, 徐工机械, 一汽轿车
2015-08-03 14:02:25.053561 张　裕Ａ, 云南白药, 中集集团, 东吴证券, 兖
州煤业, 广发证券, 丽珠集团, 首航节能, 双汇发展, 金 融 街, 海天味业, 万华化
学, 华电国际, 东方能源, 互动娱乐, 哈药股份, 日出东方, 中煤能源, 上海医药,
环旭电子, 华域汽车, 国海证券, 誉衡药业, 申万宏源, 贝因美, 深圳燃气, 聚龙股
份, 浙大网新, 一汽轿车, 上海莱士
2015-08-03 14:02:36.258301 中集集团, 张　裕Ａ, 云南白药, 兖州煤业, 广
发证券, 万华化学, 丽珠集团, 海天味业, 中文传媒, 互动娱乐, 中国西电, 一汽轿
车, 东华能源, 哈药股份, 东吴证券, 华电国际, 格力电器, 金 融 街, 日出东方,
 华域汽车, 国海证券, 誉衡药业, 申万宏源, 深圳燃气, 科力远, 陕国投Ａ, 二三
四五, 上海莱士, 粤电力Ａ, 驰宏锌锗
2015-08-03 14:02:46.940336 中集集团, 张　裕Ａ, 云南白药, 兖州煤业, 中
文传媒, 广发证券, 万华化学, 丽珠集团, 贵州茅台, 尚荣医疗, 日出东方, 一汽轿
车, 东吴证券, 东方能源, 康恩贝, 哈药股份, 金 融 街, 海天味业, 中国神华,
华域汽车, 中国西电, 互动娱乐, 誉衡药业, 上海医药, 闰土股份, 申万宏源, 深圳
燃气, 科力远, 聚龙股份, 二三四五
----------------------------------------------------------------
-----------
KeyboardInterrupt                          Traceback (most recent
 call last)
```

```
<mercury-input-27-f8d02897ce95> in <module>()
     24     print now, ', '.join([ e for e in data['简称'].values
])
     25 
---> 26     time.sleep(10)
     27 

KeyboardInterrupt: 
```

本贴的实现参考于社区高人 @明轩 @jiang.wei 的涨停板帖子。

4.14 封单量

按照封单跟流通股本比例排序，剔除6月上市新股，前50

> 来源：https://uqer.io/community/share/559e5315f9f06c6dd1e17ed6

```
import pandas as pd

today = Date.todaysDate()
cal = Calendar('China.SSE')
yesterday = cal.advanceDate(today, '-1B',  BizDayConvention.Following).strftime('%Y%m%d')
allSecList = DataAPI.EquGet(equTypeCD = 'A', field = ['secID'])['secID'].tolist()
precls = {}
uplimit, dnlimit = [], []


for i in range(0, len(allSecList), 200):
    sub = allSecList[i:min(len(allSecList), i+200)]
    df_precls = DataAPI.MktEqudGet(secID = sub, beginDate = yesterday, endDate = yesterday, field = 'secID,closePrice')
    df_lasprx = DataAPI.MktTickRTSnapshotGet(securityID = sub, field = 'shortNM,lastPrice,bidBook,askBook')
    for j in range(len(df_precls)):
        precls[df_precls.at[j,'secID']] = df_precls.at[j,'closePrice']
    for j in range(len(df_lasprx)):
        if df_lasprx.at[j,'lastPrice'] > 0:
            sec = df_lasprx.at[j,'ticker']+'.'+df_lasprx.at[j,'exchangeCD']
            if df_lasprx.at[j,'bidBook_volume1'] == 0 and df_lasprx.at[j,'askBook_volume1'] > 0:
                bang = df_lasprx.at[j,'askBook_volume1'] * df_lasprx.at[j,'askBook_price1']
                dnlimit.append([df_lasprx.at[j,'shortNM'], sec, precls[sec], df_lasprx.at[j,'lastPrice'],df_lasprx.at[j,'askBook_volume1'], bang/10000])
            if df_lasprx.at[j,'askBook_volume1'] == 0 and df_lasprx.at[j,'bidBook_volume1'] > 0:
                bang = df_lasprx.at[j,'bidBook_volume1'] * df_lasprx.at[j,'bidBook_price1']
                uplimit.append([df_lasprx.at[j,'shortNM'], sec, precls[sec], df_lasprx.at[j,'lastPrice'],df_lasprx.at[j,'bidBook_volume1'], bang/10000])
```

```
name, sec, precls, latprx, bangvol, bang = zip(*uplimit)
df_uplimit = pd.DataFrame({'简称': name, '代码': sec, '前收': prec
ls, '最新价': latprx,'封单量': bangvol, '封单金额（万）': bang}).sor
t(columns='封单金额（万）').reset_index()
df_uplimit = df_uplimit.loc[:, ['简称', '代码','前收','最新价','封单
量','封单金额（万）']]
print '涨停股票数量：', len(uplimit) , '个， 收盘封单总金额：', df_upl
imit.sum()['封单金额（万）']/10000 , '亿'
df_uplimit
```

```
涨停股票数量： 1280 个， 收盘封单总金额： 644.57280353 亿
```

	简称	代码	前收	最新价	封单量	封单金额（万）
0	科达股份	600986.XSHG	24.86	27.35	8095	22.139825
1	*ST夏利	000927.XSHE	7.11	7.47	61912	46.248264
2	宝诚股份	600892.XSHG	35.60	39.16	14200	55.607200
3	*ST古汉	000590.XSHE	17.97	18.87	35228	66.475236
4	永艺股份	603600.XSHG	96.55	106.21	7500	79.657500
5	法拉电子	600563.XSHG	27.52	30.27	27485	83.197095
6	天润乳业	600419.XSHG	37.46	41.21	30361	125.117681
	光大					

	证券					
8	S佳通	600182.XSHG	19.59	20.57	80503	165.594671
9	*ST华锦	000059.XSHE	7.57	7.95	220596	175.373820
10	鄂尔多斯	600295.XSHG	8.40	9.24	248700	229.798800
11	富煌钢构	002743.XSHE	24.32	26.75	91387	244.460225
12	良信电器	002706.XSHE	52.81	58.09	47519	276.037871
13	洋河股份	002304.XSHE	64.26	70.69	41128	290.733832
14	*ST水井	600779.XSHG	9.76	10.25	284700	291.817500
15	滨海能源	000695.XSHE	14.06	15.47	189000	292.383000
16	中钢国际	000928.XSHE	18.03	19.83	167224	331.605192
17	金地集团	600383.XSHG	12.64	13.90	273600	380.304000
18	百花村	600721.XSHG	10.21	11.23	361831	406.336213

	村					
19	XD吉林森	600189.XSHG	9.85	10.73	392689	421.355297
20	兖州煤业	600188.XSHG	9.74	10.71	422093	452.061603
21	时代万恒	600241.XSHG	11.14	12.25	381174	466.938150
22	银座股份	600858.XSHG	9.19	10.11	480700	485.987700
23	金山开发	600679.XSHG	11.39	12.53	393526	493.088078
24	远程电缆	002692.XSHE	16.67	18.34	271064	497.131376
25	苏泊尔	002032.XSHE	21.88	24.07	206600	497.286200
26	万向德农	600371.XSHG	15.93	17.52	292926	513.206352
27	洛阳玻璃	600876.XSHG	10.60	11.66	448678	523.158548
28	东方证券	600958.XSHG	23.18	25.50	205700	524.535000
	东					

29	证券	601198.XSHG	20.52	22.57	236032	532.724224
...	...	...	...	...	...	...
1250	金龙汽车	600686.XSHG	14.42	15.86	15154357	24034.810202
1251	浩物股份	000757.XSHE	7.43	8.17	30249462	24713.810454
1252	湖北能源	000883.XSHE	5.03	5.53	45428493	25121.956629
1253	上海莱士	002252.XSHE	70.82	77.90	3266870	25448.917300
1254	保利地产	600048.XSHG	9.38	10.32	24812739	25606.746648
1255	贵州百灵	002424.XSHE	46.51	51.16	5007138	25616.518008
1256	掌趣科技	300315.XSHE	9.70	10.67	25399058	27100.794886
1257	万向钱潮	000559.XSHE	11.65	12.82	21612652	27707.419864
1258	东软集团	600718.XSHG	12.09	13.30	21087016	28045.731280
	时					

1259	代新材	600458.XSHG	16.97	18.67	15189697	28359.164299
1260	东方财富	300059.XSHE	45.99	50.59	5819640	29441.558760
1261	紫光股份	000938.XSHE	56.69	62.25	4756189	29607.276525
1262	泛海控股	000046.XSHE	11.00	12.10	24553068	29709.212280
1263	隆平高科	000998.XSHE	18.15	19.97	15137564	30229.715308
1264	五粮液	000858.XSHE	21.92	24.11	12672663	30553.790493
1265	洛阳钼业	603993.XSHG	10.21	11.23	28763738	32301.677774
1266	川投能源	600674.XSHG	8.91	9.80	34759105	34063.922900
1267	大族激光	002008.XSHE	20.09	22.10	16645752	36787.111920
1268	机器人	300024.XSHE	78.20	86.02	4285905	36867.354810
1269	上海电气	601727.XSHG	10.34	11.37	33204217	37753.194729

1270	大唐电信	600198.XSHG	19.88	21.87	17856149	39051.397863
1271	招商证券	600999.XSHG	22.07	24.28	17547853	42606.187084
1272	新大陆	000997.XSHE	17.52	19.27	22280471	42934.467617
1273	同方股份	600100.XSHG	11.48	12.63	37915652	47887.468476
1274	中航动力	600893.XSHG	28.34	31.17	17871582	55705.721094
1275	国泰君安	601211.XSHG	25.88	28.47	19942615	56776.624905
1276	中国重工	601989.XSHG	8.53	9.38	67338975	63163.958550
1277	格力电器	000651.XSHE	19.64	21.60	30817884	66566.629440
1278	中国核电	601985.XSHG	8.20	9.02	76664230	69151.135460
1279	兴业银行	601166.XSHG	15.53	17.08	45921158	78433.337864

```
1280 rows × 6 columns
```

```
1280 rows × 6 columns
```

```
nfList = DataAPI.EquGet(equTypeCD = 'A', field = ['secID','nonrestfloatA','listDate'])
nf = {}
nl = {}
for j in range(len(nfList)):
    nf[nfList.at[j,'secID']] = nfList.at[j,'nonrestfloatA']
    nl[nfList.at[j,'secID']] = nfList.at[j,'listDate']
```

```
name, sec, precls, latprx, bangvol, bang = zip(*uplimit)
nflist = []
nllist = []
nfrate = []
for j in range(len(sec)):
   nflist.append(nf[sec[j]])
   nllist.append(nl[sec[j]])
   if nf[sec[j]] > 0:
       nfrate.append(bangvol[j]*1000/nf[sec[j]])
   else:
       nfrate.append(0)

print "按照封单跟流通股本比例排序，剔除6月上市新股，前50"
df_uplimit1 = pd.DataFrame({'简称': name, '代码': sec, '前收': precls, '最新价': latprx,'封单量': bangvol, '封单金额（万）': bang, '总流通股本': tuple(nflist), '封单总股本比(‰)': tuple(nfrate),'listDate': tuple(nllist)}).sort(columns='封单总股本比(‰)',ascending=False).reset_index()
df_uplimit1 = df_uplimit1.loc[:, ['简称', '代码','前收','最新价','封单量','封单金额（万）','总流通股本','封单总股本比(‰)','listDate']]
df_uplimit1[df_uplimit1.listDate < '2015-06'][:50]
```

```
按照封单跟流通股本比例排序，剔除6月上市新股，前50
```

	简称	代码	前收	最新价	封单量	封单金额（万）
4	山河药辅	300452.XSHE	46.62	51.28	1231062	6312.885936

	份					
7	康斯特	300445.XSHE	58.53	64.38	899439	5790.588282
14	三变科技	002112.XSHE	9.63	10.59	13039855	13809.206445
17	石大胜华	603026.XSHG	15.09	16.60	3422222	5680.888520
18	三毛派神	000779.XSHE	9.93	10.92	12593170	13751.741640
22	威帝股份	603023.XSHG	31.06	34.17	1292499	4416.469083
23	普丽盛	300442.XSHE	40.09	44.10	1587575	7001.205750
27	立霸股份	603519.XSHG	45.17	49.69	1217375	6049.136375
33	道氏技术	300409.XSHE	39.34	43.27	1348689	5835.777303
35	金石东方	300434.XSHE	35.04	38.54	913700	3521.399800
37	苏试试验	300416.XSHE	45.27	49.80	831929	4143.006420
38	南华	300417.XSHE	48.16	52.98	531826	2817.614148

38	南华仪器	300417.XSHE	48.16	52.98	531826	2817.614148
39	美尔雅	600107.XSHG	7.82	8.60	18644466	16034.240760
40	海联讯	300277.XSHE	15.25	16.78	2678747	4494.937466
41	华铁科技	603300.XSHG	16.81	18.49	2535770	4688.638730
42	先导股份	300450.XSHE	84.45	92.90	835200	7759.008000
45	亚星客车	600213.XSHG	7.13	7.84	10399809	8153.450256
47	鲍斯股份	300441.XSHE	31.55	34.71	972250	3374.679750
49	盛洋科技	603703.XSHG	30.03	33.03	1037229	3425.967387
50	中光防雷	300414.XSHE	50.46	55.51	910413	5053.702563
51	清水源	300437.XSHE	47.30	52.03	727611	3785.760033
52	博通股	600455.XSHG	26.21	28.83	2131135	6144.062205

53	股份	603100.XSHG	13.04	14.34	4233692	6071.114328
54	永东股份	002753.XSHE	26.64	29.30	1018901	2985.379930
55	华天酒店	000428.XSHE	5.95	6.55	29409137	19262.984735
60	惠天热电	000692.XSHE	5.23	5.75	20215598	11623.968850
61	全信股份	300447.XSHE	50.22	55.24	767600	4240.222400
62	康跃科技	300391.XSHE	14.69	16.16	1497500	2419.960000
63	莫高股份	600543.XSHG	8.19	9.01	11235969	10123.608069
64	宝色股份	300402.XSHE	12.61	13.87	1762286	2444.290682
65	金龙汽车	600686.XSHG	14.42	15.86	15154357	24034.810202
66	大唐电信	600198.XSHG	19.88	21.87	17856149	39051.397863
67	山东华	603021.XSHG	26.64	29.30	882565	2585.915450

67	山东华鹏	603021.XSHG	26.64	29.30	882565	2585.915450
68	高能环境	603588.XSHG	42.31	46.54	1296851	6035.544554
69	三鑫医疗	300453.XSHE	35.23	38.75	649389	2516.382375
70	中来股份	300393.XSHE	40.45	44.50	759245	3378.640250
71	仙坛股份	002746.XSHE	15.21	16.73	1310986	2193.279578
73	济民制药	603222.XSHG	17.89	19.68	1302174	2562.678432
74	兰州黄河	000929.XSHE	14.69	16.16	6088560	9839.112960
75	迅游科技	300467.XSHE	176.12	193.73	328574	6365.464102
76	北京城乡	600861.XSHG	10.22	11.24	10072835	11321.866540
77	中泰股份	300435.XSHE	34.94	38.43	621982	2390.276826
	轴研					

	技					
79	四通新材	300428.XSHE	34.41	37.85	609600	2307.336000
80	田中精机	300461.XSHE	29.96	32.96	510040	1681.091840
81	运达科技	300440.XSHE	42.54	46.79	847050	3963.346950
83	迈克生物	300463.XSHE	57.51	63.26	1140788	7216.624888
84	华通医药	002758.XSHE	38.64	42.50	429400	1824.950000
86	鼎泰新材	002352.XSHE	27.66	30.43	1520600	4627.185800

涨停股票封单统计

> 来源：https://uqer.io/community/share/559e49d5f9f06c6dd1e17ed5

```
import pandas as pd

today = Date.todaysDate()
cal = Calendar('China.SSE')
yesterday = cal.advanceDate(today, '-1B',  BizDayConvention.Following).strftime('%Y%m%d')
allSecList = DataAPI.EquGet(equTypeCD = 'A', field = ['secID'])['secID'].tolist()
precls = {}
uplimit, dnlimit = [], []


for i in range(0, len(allSecList), 200):
    sub = allSecList[i:min(len(allSecList), i+200)]
    df_precls = DataAPI.MktEqudGet(secID = sub, beginDate = yesterday, endDate = yesterday, field = 'secID,closePrice')
    df_lasprx = DataAPI.MktTickRTSnapshotGet(securityID = sub, field = 'shortNM,lastPrice,bidBook,askBook')
    for j in range(len(df_precls)):
        precls[df_precls.at[j,'secID']] = df_precls.at[j,'closePrice']
    for j in range(len(df_lasprx)):
        if df_lasprx.at[j,'lastPrice'] > 0:
            sec = df_lasprx.at[j,'ticker']+'.'+df_lasprx.at[j,'exchangeCD']
            if df_lasprx.at[j,'bidBook_volume1'] == 0 and df_lasprx.at[j,'askBook_volume1'] > 0:
                bang = df_lasprx.at[j,'askBook_volume1'] * df_lasprx.at[j,'askBook_price1']
                dnlimit.append([df_lasprx.at[j,'shortNM'], sec, precls[sec], df_lasprx.at[j,'lastPrice'],df_lasprx.at[j,'askBook_volume1'], bang/10000])
            if df_lasprx.at[j,'askBook_volume1'] == 0 and df_lasprx.at[j,'bidBook_volume1'] > 0:
                bang = df_lasprx.at[j,'bidBook_volume1'] * df_lasprx.at[j,'bidBook_price1']
                uplimit.append([df_lasprx.at[j,'shortNM'], sec, precls[sec], df_lasprx.at[j,'lastPrice'],df_lasprx.at[j,'bidBook_volume1'], bang/10000])
```

```
name, sec, precls, latprx, bangvol, bang = zip(*uplimit)
df_uplimit = pd.DataFrame({'简称': name, '代码': sec, '前收': precls, '最新价': latprx,'封单量': bangvol, '封单金额（万）': bang}).sort(columns='封单金额（万）').reset_index()
df_uplimit = df_uplimit.loc[:, ['简称', '代码','前收','最新价','封单量','封单金额（万）']]
print '涨停股票数量：', len(uplimit) , '个， 收盘封单总金额：', df_uplimit.sum()['封单金额（万）']/10000 , '亿'
df_uplimit

涨停股票数量： 1280 个， 收盘封单总金额： 644.57280353 亿
```

	简称	代码	前收	最新价	封单量	封单金额（万）
0	科达股份	600986.XSHG	24.86	27.35	8095	22.139825
1	*ST夏利	000927.XSHE	7.11	7.47	61912	46.248264
2	宝诚股份	600892.XSHG	35.60	39.16	14200	55.607200
3	*ST古汉	000590.XSHE	17.97	18.87	35228	66.475236
4	永艺股份	603600.XSHG	96.55	106.21	7500	79.657500
5	法拉电子	600563.XSHG	27.52	30.27	27485	83.197095
6	天润乳业	600419.XSHG	37.46	41.21	30361	125.117681
7	光大	601788.XSHG	20.09	22.09	65049	143.758290

7	证券	601788.XSHG	20.09	22.09	65049	143.758290
8	S佳通	600182.XSHG	19.59	20.57	80503	165.594671
9	*ST华锦	000059.XSHE	7.57	7.95	220596	175.373820
10	鄂尔多斯	600295.XSHG	8.40	9.24	248700	229.798800
11	富煌钢构	002743.XSHE	24.32	26.75	91387	244.460225
12	良信电器	002706.XSHE	52.81	58.09	47519	276.037871
13	洋河股份	002304.XSHE	64.26	70.69	41128	290.733832
14	*ST水井	600779.XSHG	9.76	10.25	284700	291.817500
15	滨海能源	000695.XSHE	14.06	15.47	189000	292.383000
16	中钢国际	000928.XSHE	18.03	19.83	167224	331.605192
17	金地集团	600383.XSHG	12.64	13.90	273600	380.304000
18	百花	600721.XSHG	10.21	11.23	361831	406.336213

19	XD吉林森	600189.XSHG	9.85	10.73	392689	421.355297
20	兖州煤业	600188.XSHG	9.74	10.71	422093	452.061603
21	时代万恒	600241.XSHG	11.14	12.25	381174	466.938150
22	银座股份	600858.XSHG	9.19	10.11	480700	485.987700
23	金山开发	600679.XSHG	11.39	12.53	393526	493.088078
24	远程电缆	002692.XSHE	16.67	18.34	271064	497.131376
25	苏泊尔	002032.XSHE	21.88	24.07	206600	497.286200
26	万向德农	600371.XSHG	15.93	17.52	292926	513.206352
27	洛阳玻璃	600876.XSHG	10.60	11.66	448678	523.158548
28	东方证券	600958.XSHG	23.18	25.50	205700	524.535000
	东兴					

29	兴证券	601198.XSHG	20.52	22.57	236032	532.724224
...	...	...	...	...	...	...
1250	金龙汽车	600686.XSHG	14.42	15.86	15154357	24034.810202
1251	浩物股份	000757.XSHE	7.43	8.17	30249462	24713.810454
1252	湖北能源	000883.XSHE	5.03	5.53	45428493	25121.956629
1253	上海莱士	002252.XSHE	70.82	77.90	3266870	25448.917300
1254	保利地产	600048.XSHG	9.38	10.32	24812739	25606.746648
1255	贵州百灵	002424.XSHE	46.51	51.16	5007138	25616.518008
1256	掌趣科技	300315.XSHE	9.70	10.67	25399058	27100.794886
1257	万向钱潮	000559.XSHE	11.65	12.82	21612652	27707.419864
1258	东软集团	600718.XSHG	12.09	13.30	21087016	28045.731280
	时					

1259	代新材	600458.XSHG	16.97	18.67	15189697	28359.164299
1260	东方财富	300059.XSHE	45.99	50.59	5819640	29441.558760
1261	紫光股份	000938.XSHE	56.69	62.25	4756189	29607.276525
1262	泛海控股	000046.XSHE	11.00	12.10	24553068	29709.212280
1263	隆平高科	000998.XSHE	18.15	19.97	15137564	30229.715308
1264	五粮液	000858.XSHE	21.92	24.11	12672663	30553.790493
1265	洛阳钼业	603993.XSHG	10.21	11.23	28763738	32301.677774
1266	川投能源	600674.XSHG	8.91	9.80	34759105	34063.922900
1267	大族激光	002008.XSHE	20.09	22.10	16645752	36787.111920
1268	机器人	300024.XSHE	78.20	86.02	4285905	36867.354810
1269	上海电气	601727.XSHG	10.34	11.37	33204217	37753.194729

1270	大唐电信	600198.XSHG	19.88	21.87	17856149	39051.397863
1271	招商证券	600999.XSHG	22.07	24.28	17547853	42606.187084
1272	新大陆	000997.XSHE	17.52	19.27	22280471	42934.467617
1273	同方股份	600100.XSHG	11.48	12.63	37915652	47887.468476
1274	中航动力	600893.XSHG	28.34	31.17	17871582	55705.721094
1275	国泰君安	601211.XSHG	25.88	28.47	19942615	56776.624905
1276	中国重工	601989.XSHG	8.53	9.38	67338975	63163.958550
1277	格力电器	000651.XSHE	19.64	21.60	30817884	66566.629440
1278	中国核电	601985.XSHG	8.20	9.02	76664230	69151.135460
1279	兴业银行	601166.XSHG	15.53	17.08	45921158	78433.337864

```
nfList = DataAPI.EquGet(equTypeCD = 'A', field = ['secID','nonrestfloatA'])
nf = {}
for j in range(len(nfList)):
    nf[nfList.at[j,'secID']] = nfList.at[j,'nonrestfloatA']
```

```
name, sec, precls, latprx, bangvol, bang = zip(*uplimit)
nflist = []
nfrate = []
for j in range(len(sec)):
   nflist.append(nf[sec[j]])
   if nf[sec[j]] > 0:
       nfrate.append(bangvol[j]*1000/nf[sec[j]])
   else:
       nfrate.append(0)

print "按照封单跟流通股本比例排序"
df_uplimit1 = pd.DataFrame({'简称': name, '代码': sec, '前收': precls, '最新价': latprx,'封单量': bangvol, '封单金额（万）': bang, '总流通股本': tuple(nflist), '封单总股本比(千分之n)': tuple(nfrate)}).sort(columns='封单总股本比(千分之n)',ascending=False).reset_index()
df_uplimit1 = df_uplimit1.loc[:, ['简称', '代码','前收','最新价','封单量','封单金额（万）','总流通股本','封单总股本比(千分之n)']]
df_uplimit1
```

```
按照封单跟流通股本比例排序
```

	简称	代码	前收	最新价	封单量	封单金额（万）
0	光力科技	300480.XSHE	15.35	16.89	5979987	10100.198043
1	万孚生物	300482.XSHE	40.81	44.89	4077275	18302.887475

1	孚生物	300482.XSHE	40.81	44.89	4077275	18302.887475
2	真视通	002771.XSHE	35.85	39.44	3170548	12504.641312
3	恒锋工具	300488.XSHE	39.87	43.86	1950131	8553.274566
4	山河药辅	300452.XSHE	46.62	51.28	1231062	6312.885936
5	浩物股份	000757.XSHE	7.43	8.17	30249462	24713.810454
6	蓝晓科技	300487.XSHE	31.28	34.41	1894400	6518.630400
7	康斯特	300445.XSHE	58.53	64.38	899439	5790.588282
8	天成自控	603085.XSHG	18.55	20.41	2189087	4467.926567
9	中飞股份	300489.XSHE	40.73	44.80	997400	4468.352000
10	杭州高新	300478.XSHE	30.02	33.02	1414810	4671.702620
11	濮阳惠成	300481.XSHE	22.51	24.76	1484372	3675.305072

13	汇洁股份	002763.XSHE	21.98	24.18	3901200	9433.101600
14	三变科技	002112.XSHE	9.63	10.59	13039855	13809.206445
15	康弘药业	002773.XSHE	35.30	38.83	3163996	12285.796468
16	凤形股份	002760.XSHE	23.81	26.19	1531650	4011.391350
17	石大胜华	603026.XSHG	15.09	16.60	3422222	5680.888520
18	三毛派神	000779.XSHE	9.93	10.92	12593170	13751.741640
19	万林股份	603117.XSHG	13.62	14.98	4056942	6077.299116
20	四通股份	603838.XSHG	17.27	19.00	2204115	4187.818500
21	普路通	002769.XSHE	65.41	71.95	1226834	8827.070630
22	威帝股份	603023.XSHG	31.06	34.17	1292499	4416.469083
	普					

22	股份	603023.XSHG	31.06	34.17	1292499	4416.469083
23	普丽盛	300442.XSHE	40.09	44.10	1587575	7001.205750
24	科迪乳业	002770.XSHE	12.63	13.89	4330037	6014.421393
25	音飞储存	603066.XSHG	21.68	23.85	1558372	3716.717220
26	沃施股份	300483.XSHE	29.04	31.94	947482	3026.257508
27	立霸股份	603519.XSHG	45.17	49.69	1217375	6049.136375
28	东杰智能	300486.XSHE	20.40	22.44	2050203	4600.655532
29	口子窖	603589.XSHG	24.64	27.10	3444728	9335.212880
...	...	...	...	...	...	...
1250	太平洋	601099.XSHG	9.55	10.51	1577651	1658.111201
1251	科达股份	600986.XSHG	24.86	27.35	8095	22.139825
1252	中国神华	601088.XSHG	18.84	20.72	266241	551.651352

1254	华泰证券	601688.XSHG	20.54	22.59	1029106	2324.750454
1255	际华集团	601718.XSHG	10.10	11.11	1550394	1722.487734
1256	美的集团	000333.XSHE	30.81	33.89	883331	2993.608759
1257	万华化学	600309.XSHG	18.45	20.30	282697	573.874910
1258	西南证券	600369.XSHG	15.81	17.39	1456800	2533.375200
1259	宁沪高速	600377.XSHG	7.05	7.76	2667199	2069.746424
1260	金地集团	600383.XSHG	12.64	13.90	273600	380.304000
1261	天润乳业	600419.XSHG	37.46	41.21	30361	125.117681
1262	荣盛石化	002493.XSHE	13.85	15.24	941374	1434.653976
1263	立讯精密	002475.XSHE	30.01	33.01	551275	1819.758775
	金					

1263	讯精密	002475.XSHE	30.01	33.01	551275	1819.758775
1264	金隅股份	601992.XSHG	7.51	8.26	2923312	2414.655712
1265	大唐发电	601991.XSHG	5.84	6.42	3382468	2171.544456
1266	大秦铁路	601006.XSHG	10.08	11.09	3236323	3589.082207
1267	中国汽研	601965.XSHG	8.90	9.79	878170	859.728430
1268	中国中车	601766.XSHG	16.09	17.70	10114023	17901.820710
1269	深高速	600548.XSHG	7.57	8.33	1113968	927.935344
1270	中国远洋	601919.XSHG	7.10	7.81	5954095	4650.148195
1271	海信科龙	000921.XSHE	9.76	10.74	693024	744.307776
1272	方正证券	601901.XSHG	9.70	10.67	1558703	1663.136101
1273	中煤能源	601898.XSHG	6.89	7.58	4670154	3539.976732

1275	招商轮船	601872.XSHG	6.83	7.51	4215059	3165.509309
1276	中海集运	601866.XSHG	5.56	6.12	7863060	4812.192720
1277	中国交建	601800.XSHG	15.49	17.04	680950	1160.338800
1278	光大证券	601788.XSHG	20.09	22.09	65049	143.758290
1279	法拉电子	600563.XSHG	27.52	30.27	27485	83.197095

```
1280 rows × 8 columns
```

实时计算涨停板股票的封单资金与总流通市值的比例

> 来源：https://uqer.io/community/share/55a8cf52f9f06c57a11b53b9

申明： 本贴的思路来源于社区高人 @明轩 的启发 说明： 本人之前一直用Matlab/R研究策略，最近发现了“优矿”这个策略研究平台，感觉非常的方便，社区大神也很多，社区里也有很多优秀的帖子， 再加上一直想学python的这种冲动（毕竟python在量化界还是很有地位的），所以就萌生在此发帖写策略的冲动。现在看来，这一切都是值的。

策略介绍： 过去的几个星期里中国股市很不太平，市场基本就只有三种状态：涨停、跌停、停牌。在这种特殊行情下，一些平时不会考虑到的因素可能在 这段时间里很有效。论坛里 @明轩 有一个很不错的想法，计算涨停板股票的封单资金与流通市值的比例，用以反映该股票的热度，这一点在直观上 也是很好理解的，本篇做的也就是这件事情，只需要克隆并运行下面的代码，就可以得到实时的“封单资金流通市值比例”。 在@明轩的帖子中，代码实现的大部分都是C的思想，过程显得比较复杂，而且对于程序也没有太多的注释说明，对初学者来说还是有一些压力的。 作者也是上周才看到“优矿”社区，学python也就一周的时间，初看 @明轩 的代码还是很吃力的，由于作者是matlab出身，所以就用matlab的思想来 实现这个想法（主要是矩阵运算，代码中尽量不要用到for循环），同时对于代码也做到注释详尽，以便和我一样的初学者共同学习进步。 作者新人，初次发帖，难免会有疏忽，还望论坛各位大神予以指正。

```
# 常量准备
import pandas as pd
from datetime import datetime as dt
from pandas import DataFrame, Series
today = dt.today().strftime('%Y%m%d')   # 获得今天的日期

# DataAPI取所有A股
stocks = DataAPI.EquGet(equTypeCD='A',listStatusCD='L',field='secID,nonrestfloatA',pandas="1")
universe = stocks['secID'].tolist()     # 转变为list格式，以便和DataAPI中的格式符合

# 取所有A股的最新行情
fields = ['shortNM','lastPrice','bidBook','askBook','suspension']
data = DataFrame()
for i in range(0,len(universe),300):    # 原则上可以性取完的，但是试验中作者发现会报错，估计是运算量太大，所以这里分批次取，每次300个
    t = DataAPI.MktTickRTSnapshotGet(securityID=universe[i:min(i+300,len(universe))],field=fields,pandas="1")
    tmp = DataFrame()
    tmp['secID'] = t['ticker']+'.'+t['exchangeCD']
    tmp[['shortNM','lastPrice','bidBook_price1','bidBook_volume1'
,'askBook_volume1','suspension']] =t[['shortNM','lastPrice','bidBook_price1','bidBook_volume1','askBook_volume1','suspension']]
    data = pd.concat([data,tmp],axis=0)    # 数据拼接

# 去掉当日停牌的股票
data['nonrestfloatA'] = stocks['nonrestfloatA']
data = data[data['suspension']==0]

# 去掉没有涨停板的股票
data['suspension'][(data['bidBook_volume1']>0).values & (data['askBook_volume1']==0).values] = 1  # 若涨停盘，suspension则赋值1
data = data[data['suspension']==1]
data.drop(['suspension','askBook_volume1'],axis=1,inplace=True)

# 计算封停板资金量、流通市值、两者比值
data['stop_money'] = data['bidBook_price1'].values * data['bidBook_volume1'].values
data['float_value'] = data['bidBook_price1'].values * data['nonrestfloatA'].values
data['rate'] = data['stop_money']/data['float_value']*100   #百分之几
data = data.sort(columns='rate',ascending=False).reset_index()
data.drop('index',axis=1,inplace=True)

# 重命名
data.columns = ['代码','简称','最新成交价','买一价','买一量','非受限流通股','封停板资金','流通市值','封停资金流通市值比（百分之几）']
data.head(30)
```

	代码	简称	最新成交价	买一价	买一量	非受限流通股	封停
0	600127.XSHG	金健米业	8.64	8.64	31034932	138756240	2.68
1	002465.XSHE	海格通信	26.49	26.49	48681212	232841128	1.28
2	002451.XSHE	摩恩电气	10.15	10.15	20255795	99720453	2.05
		酒					

3	600307.XSHG	钢宏兴	4.97	4.97	41841200	223174819	2.07
4	601216.XSHG	内蒙君正	10.97	10.97	47936330	256155318	5.25
5	002220.XSHE	天宝股份	9.53	9.53	36358180	203638446	3.46
6	000153.XSHE	丰原药业	8.91	8.91	43825338	296645200	3.90
7	600363.XSHG	联创光电	11.98	11.98	39827219	287985798	4.77
8	600881.XSHG	亚泰集团	10.32	10.32	39219948	324000000	4.04
9	601989.XSHG	中国重工	12.90	12.90	35504690	309043600	4.58
10	002578.XSHE	闽发铝业	8.61	8.61	18732903	169374479	1.61
11	002314.XSHE	雅致股份	8.51	8.51	29786294	331060900	2.53
12	002596.XSHE	海南瑞泽	19.77	19.77	13188288	150360840	2.60
13	300245.XSHE	天玑科	14.99	14.99	30609263	358143846	4.58

		技					
14	600725.XSHG	云维股份	7.21	7.21	10355583	131323745	7.466
15	600635.XSHG	大众公用	8.45	8.45	15693824	203285722	1.326
16	000005.XSHE	世纪星源	5.96	5.96	65653281	913743007	3.912
17	000908.XSHE	景峰医药	13.72	13.72	22259100	318306848	3.053
18	002392.XSHE	北京利尔	6.60	6.60	18238316	272720782	1.203
19	300315.XSHE	掌趣科技	15.47	15.47	8467640	143055590	1.309
20	000733.XSHE	振华科技	17.52	17.52	25394193	429158874	4.449
21	002492.XSHE	恒基达鑫	10.15	10.15	15039065	258580938	1.526
22	000662.XSHE	索芙特	13.52	13.52	16284361	287985798	2.201
23	002070.XSHE	众和股份	9.57	9.57	16355886	301076280	1.565
		好					

24	002582.XSHE	想你	18.94	18.94	19100773	400080405	3.61
25	600818.XSHG	中路股份	67.12	67.12	9203792	213594042	6.17
26	600439.XSHG	瑞贝卡	6.90	6.90	17411170	429158874	1.20
27	002385.XSHE	大北农	16.10	16.10	8450558	221721339	1.36
28	600839.XSHG	四川长虹	8.07	8.07	21128839	560561743	1.70
29	002176.XSHE	江特电机	18.48	18.48	13745498	387357522	2.54

应用： 只用将上述代码块拷到自己的notebook下运行，便可以得到实时的涨停板资金总流通市值比例。 后文： 得到上述比例表格可能暂时没有实际的用处，顶多指导一下明天投资的优先顺序。所以，接下来作者也会写一下更具实际价值的研究工具， 比如：实时更新昨天封停资金流通市值比例前20,但今天没有涨停的股票（对于市场遇到急跌的情况，一旦这些股票开板了，是非常好的投资机会）。 也希望通过多发一些帖子来进一步熟悉python和quartz。

4.15 成交量•决战之地， IF1507 ！

> 来源：https://uqer.io/community/share/559a44a1f9f06cb5604f1902
>
> 本文试图通过股指期货主力合约的成交量价分析，找出6月下旬以来暴跌的线索。本文的思路受癫狂的IF1507成日交易额巨量金融怪兽 远超现货的启发，但是并不局限于此。
>
> 链接的文章中，没有考虑期货合约主力切换时对成交量的变化。同时原文中，未能将期指7月合约的成交量与过往主力合约进行对比，使得结论的立据不足。
>
> 结论仍然含有很多作者本人的猜测。欲得出更可靠的结果，需要精细的分析买卖盘口等深度数据。

```
import pandas as pd
import seaborn as sns
sns.set_style('white')
from CAL.PyCAL import *
from matplotlib import pylab
```

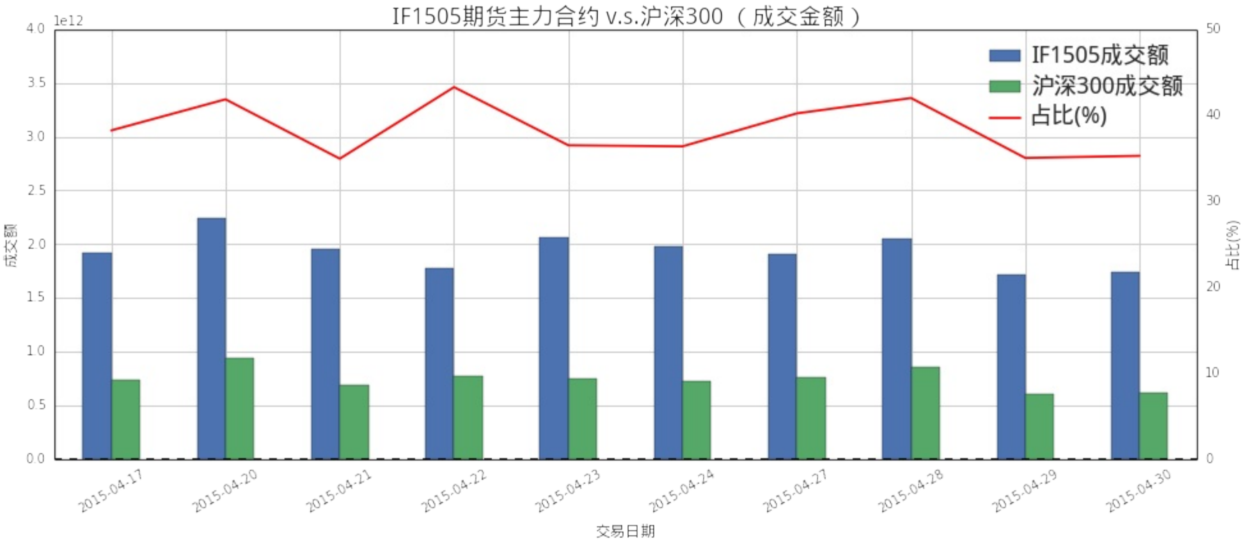
1. 期货主力合约成交变化趋势

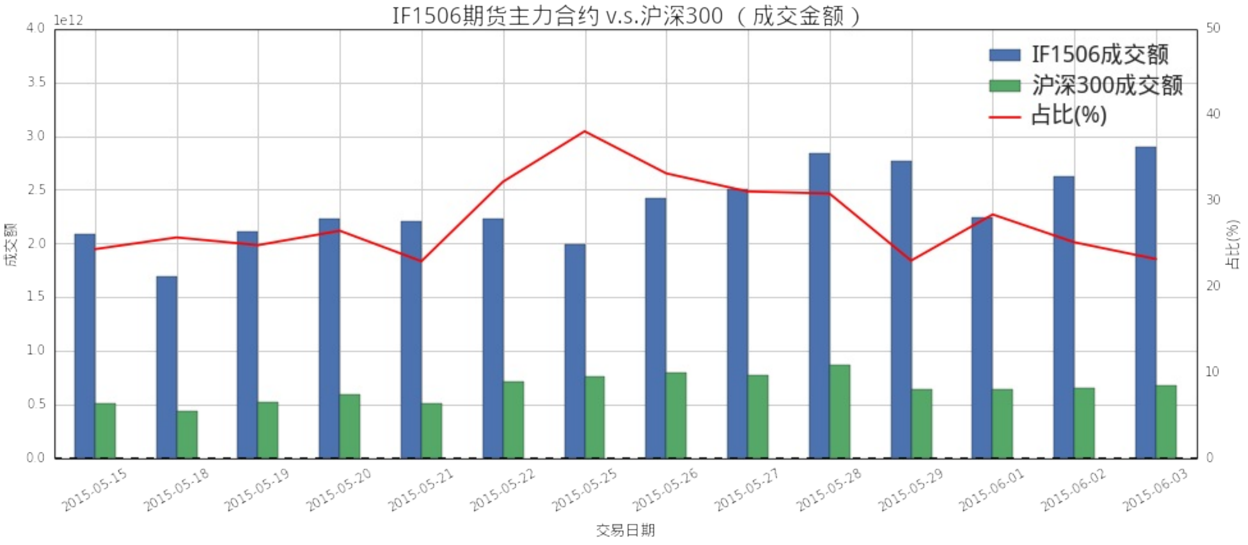
市场参与者猜测在6月下旬开始的暴跌中，股指期货主力合约是罪魁祸首。天量的期货空头，压制了多头的艰难上攻，引发了现货市场一泻千里的惨象。这里我们并不打算讨论这一运行逻辑，只打算从数据出发，分析是否真的在6月市场上出现了期货空头集中出击的局面。

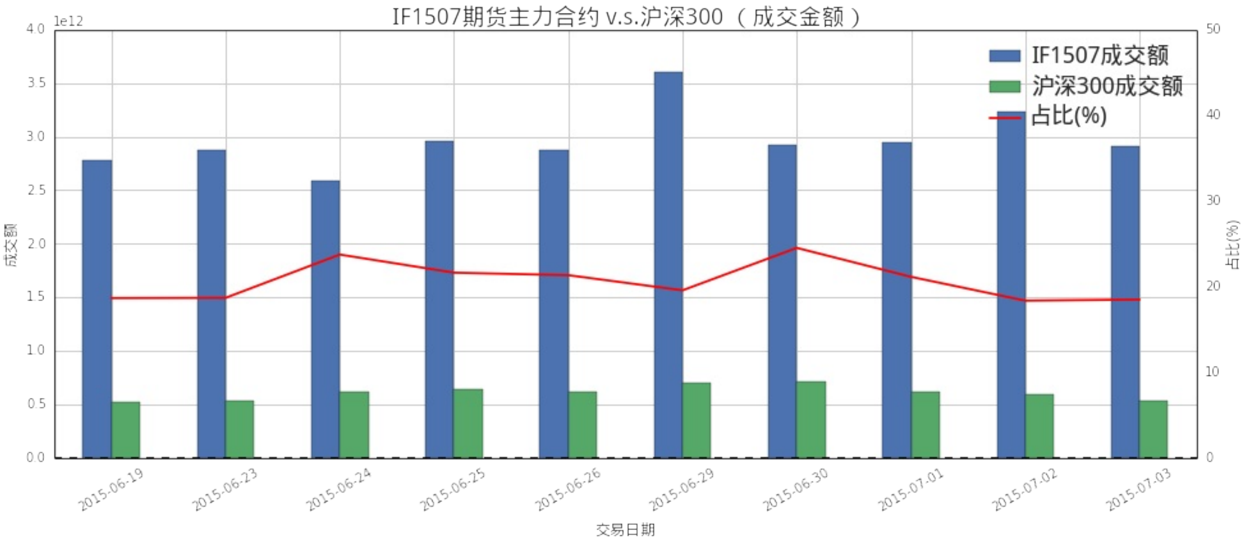
下图中，我们绘制了从2015年4月开始到2015年6月为止的期货主力合约成交金额与现货成交金额对比图：

- IF150X成交额：当月期货主力合约成交金额；
- 沪深300成交额：沪深300成分股总成交金额，用于指示现货市场的成交额；
- 占比(%)：现货市场成交金额占期货市场成交金额比例。

```
contractMonths = [5,6,7]
allRes = []
for cMonth in contractMonths:
    contractName = 'IF150' + unicode(cMonth)
    beginDate = Date.NthWeekDay(3,Friday,cMonth-1,2015).strftime('%Y%m%d')
    endDate = Date(2015,cMonth,3).strftime('%Y%m%d')
    futRes = DataAPI.MktFutdGet(ticker=contractName, beginDate=beginDate, endDate = endDate, field=['tradeDate','closePrice' ,'turnoverVol','turnoverValue'])
    futRes = futRes.set_index(['tradeDate'])
    idxRes = DataAPI.MktIdxdGet(ticker='000300', beginDate=beginDate, endDate = endDate,field=['tradeDate','closeIndex','turnoverVol','turnoverValue'])
    idxRes = idxRes.set_index(['tradeDate'])
    totalRes = pd.merge(futRes, idxRes, left_index=True, right_index=True)
    totalRes.columns = [contractName, contractName + ' turn over vol', contractName + u' turn over value', u'HS300', u'HS300 turn over vol', u'HS300 turn over value',]
    totalRes[u'Per.(%)'] = totalRes[u'HS300 turn over value'] / totalRes[contractName + u' turn over value'] * 100
    font.set_size(12)
    pylab.figure(figsize = (16,6))
    ax1 = totalRes.plot(y=['Per.(%)'], secondary_y= ['Per.(%)'], style='r')
    ax2 = totalRes.plot(y=[contractName + ' turn over value', 'HS300 turn over value'], kind='bar',legend=False, rot=30)
    patches2, labels2 = ax2.get_legend_handles_labels()
    ax2.set_ylabel(u'成交额', fontproperties=font)
    ax2.set_ylim((0.0,4.0e12))
    ax1.set_ylim((0.0, 50.0))
    ax2.set_xlabel(u'交易日期', fontproperties=font)
    ax1.set_ylabel(u'占比(%)', fontproperties=font)
    patches1, labels1 = ax1.get_legend_handles_labels()
    ax1.legend(patches2 + patches1, [contractName + u'成交额', u'沪深300成交额', u'占比(%)'], prop = font, loc='best')
    font.set_size(18)
    pylab.title(contractName + u'期货主力合约 v.s.沪深300（成交金额）', fontproperties=font)
    allRes.append(totalRes)
```







可以观察到，在上面3张系列图中，有以下值得关注的变化：

- 期货主力成交金额显著上升，从日均2万亿上升至日均3万亿；
- 现货成交量较为稳定，日均在5千亿到1万亿之间；
- 现货成交与期货成交占比显著下降，从4月份的40%左右，下降至5月份的30%

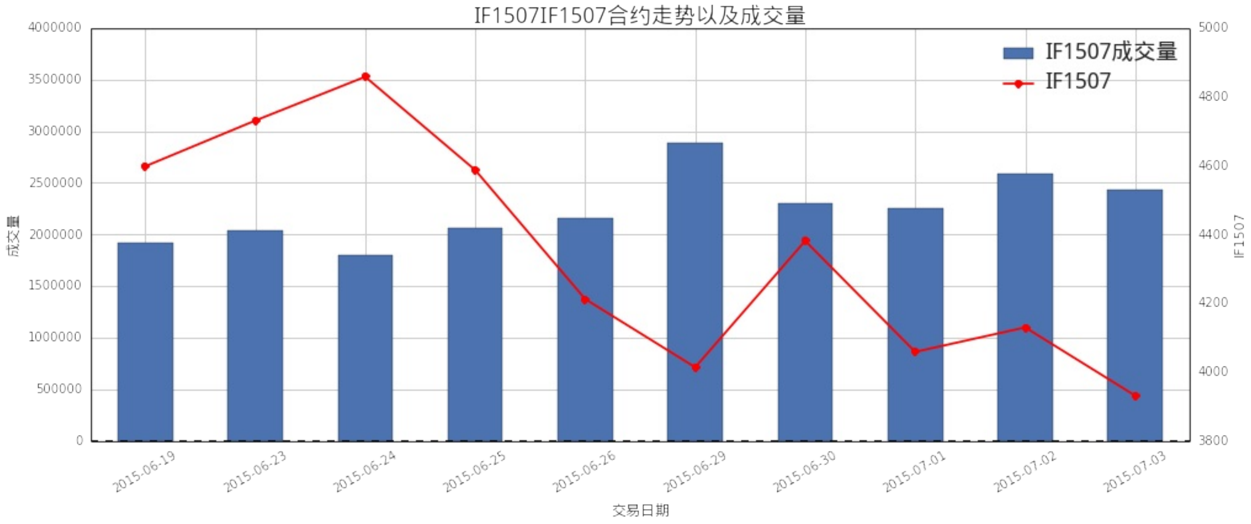
左右，直至6月份的20%左右。

从上面的分析确实可以得出，特别是在6月份，以现货市场的交易为基准，期货主力合约交易显著放大。如果我们假设原先市场是整体中性的，即现货市场与期货市场整体对冲均衡。那么这两个月以来的期货新加入者，很有可能是在6月下旬以来，通过裸卖空等手段，做空市场的主力。

2. IF1507期货价量变化趋势

下面的图简单描述了IF1507合约自成为主力合约以来的价量变化趋势。整体来说，IF1507一直呈下跌趋势，而这一趋势中也伴随着整体成交量的上升。

```
font.set_size(12)
pylab.figure(figsize = (16,6))
ax2 = allRes[-1].plot(y=['IF1507'], secondary_y=['IF1507'], styl
e='ro-', )
ax1 = allRes[-1].plot(y=['IF1507 turn over vol'], kind='bar', ro
t =30)
ax1.set_ylabel(u'成交量', fontproperties=font)
ax2.set_ylabel('IF1507')
patches, labels = ax1.get_legend_handles_labels()
patches2, labels2 = ax2.get_legend_handles_labels()
ax1.legend(patches + patches2, [u'IF1507成交量', u'IF1507'], prop
 = font, loc='best')
ax1.set_ylim((0.0,4000000))
ax1.set_xlabel(u'交易日期', fontproperties=font)
font.set_size(18)
pylab.title(contractName + u'IF1507合约走势以及成交量', fontpropert
ies=font)
```



3. 结语

本文简单的通过趋势分析，得出来市场的空头6月下旬以来通过期货主力合约IF1507，集中出击，打压大盘的结论。但是这里面也含有作者本人的很多猜测。事实上，我们无法得出新增加的期货参与者，到底是空头还是多头。更准确的市场多空力量角斗图，需要精细化的分析，例如买卖盘口等深度数据。

4.16 K 线分析 • 寻找夜空中最亮的星

> 来源：https://uqer.io/community/share/557c4527f9f06c6518ad1543

星蜡烛线（简称星线）的实体较小,并且在它的实体与它前面的蜡烛线的实体之间形成了价格跳空。只要星线的实体与前一个实体没有任何重叠,那么这个星蜡烛线就是成立的。星线对形态反转有着重要意义。

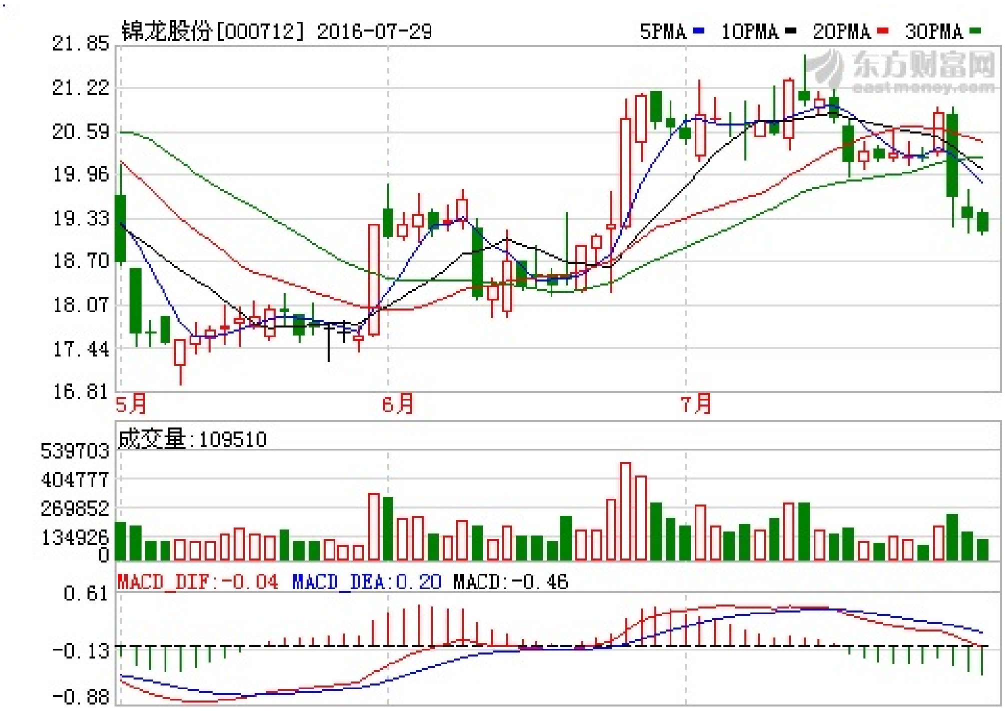
- step 1. 获取股票池。
- step 2. 股票池筛选：最近趋势为跌-->换手率>2%-->近两天出现过星线。
- step 3. 输出这些股票的最近k线图。

```python
from lib.kbase import K

def trend(closePrices, before=5):
    '''趋势
    @param closePrices: 收盘价格列表，由先到后顺序
    @return: 上涨趋势返回1，下跌趋势返回-1, 趋势不明返回0
    '''
    if len(closePrices) < before: return 0

    up = 0
    down = 0
    for i, price in enumerate(closePrices[-before:]):
        if i == 0:
            pre_price = price
            continue
        if pre_price < price:
            up += 1
        elif pre_price > price:
            down += 1
        pre_price = price
    if up > down and closePrices[-2] > closePrices[before-1]:
        return 1
    elif down > up and closePrices[-2] < closePrices[before-1]:
        return -1
    else:
        return 0

def is_star_line(infos):
    '''
    @param infos: [(openPrice，lowestPrice，highestPrice，closePrice), ...]
    '''
    ks = [K(e) for e in infos]
    stars = []
    for k in ks[-3:]:
        if k.length > 0 and (k.entityLen/k.length) <= 0.1:
            stars.append(k)
    for star in stars:
        idx = ks.index(star)
        pre = ks[idx-1]
        if pre.isRed:
            if star.openPrice >= pre.closePrice and star.closePr
ice >= pre.closePrice:
                return True
            elif star.openPrice <= pre.openPrice and star.closeP
rice <= pre.openPrice:
                return True
        else:
            if star.openPrice >= pre.openPrice and star.closePri
ce >= pre.openPrice:
                return True
            elif star.openPrice <= pre.closePrice and star.close
Price <= pre.closePrice:
                return True
    return False
```

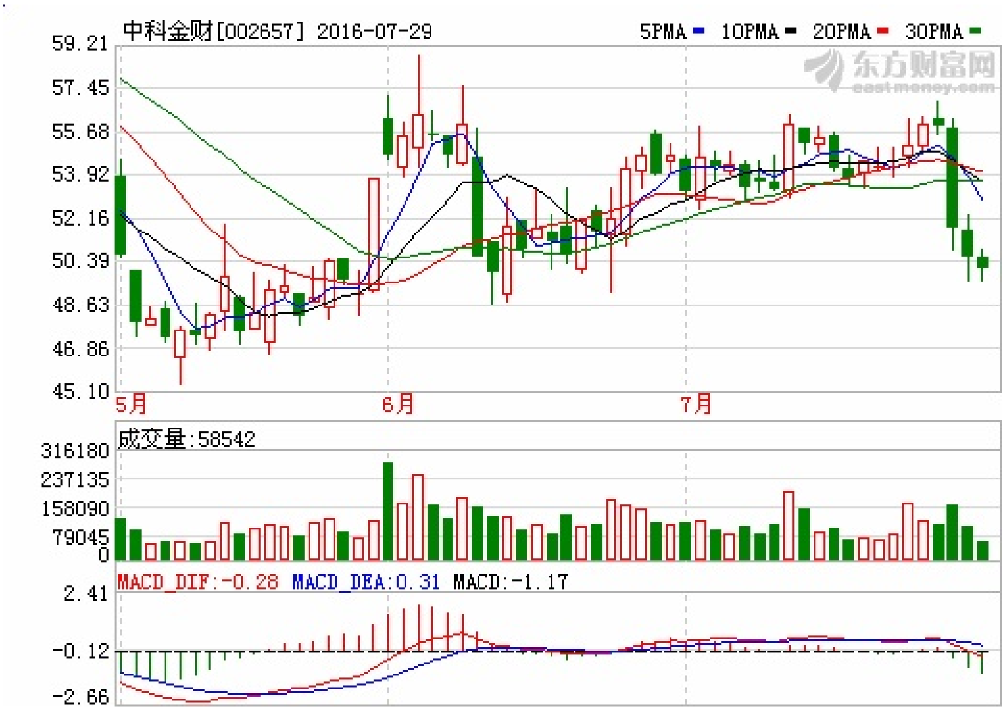
```
def get_k_pic(tid, ecd):
    if ecd == 'XSHE':
        pic = '![%s](http://hqpick.eastmoney.com/EM_Quote2010Pic
tureProducter/Index.aspx?ImageType=KXL&ID=%s%s&EF=&Formula=MACD)'
 % \
              (tid, tid, 2)
    else:
        pic = '![%s](http://hqpick.eastmoney.com/EM_Quote2010Pic
tureProducter/Index.aspx?ImageType=KXL&ID=%s%s&EF=&Formula=MACD)'
 % \
              (tid, tid, 1)
    return pic
```

```
from lib.kbase import K

today = '20150612'
beginDate = '20150601'

def get_active_tickers():
    tickers = DataAPI.EquGet(equTypeCD="A", listStatusCD="L", fi
eld=['ticker', 'ListSector'])
    tickers = tickers[tickers['ListSector'] != '创业板']
    tickers = [val[0] for val in tickers.values]
    return tickers

def get_last_with_star_line_tickers(count):
    tickers = []
    all_tickers = get_active_tickers()
    for ticker in all_tickers:
        infos = DataAPI.MktEqudAdjGet(ticker=ticker, beginDate=b
eginDate,
                                      field=["ticker", "secShort
Name", "exchangeCD", "tradeDate", "openPrice",
                                             "lowestPrice", "hig
hestPrice", "closePrice"])
        infos = infos[infos['openPrice'] > 0]
        closePrices = [val[-1] for val in infos.values]
        if not closePrices: continue
        if trend(closePrices) != -1: continue
        vol5 = DataAPI.MktStockFactorsOneDayGet(tradeDate=today,
ticker=infos.values[0][0],field=["ticker","VOL5"])
        vol5 = vol5.values[0][1]
        if vol5 < 0.02: continue
        _infos = [val[-4:] for val in infos.values]
        if is_star_line(_infos):
            tickers.append(infos)
        if len(tickers) >= count: break
    return tickers

if __name__ == '__main__':
    tickers = get_last_with_star_line_tickers(1000)
    for t in tickers:
        print '###%s(%s)\n' % (t.values[0][1], t.values[0][0])
        print get_k_pic(t.values[0][0], t.values[0][2])
        print '\n'
```

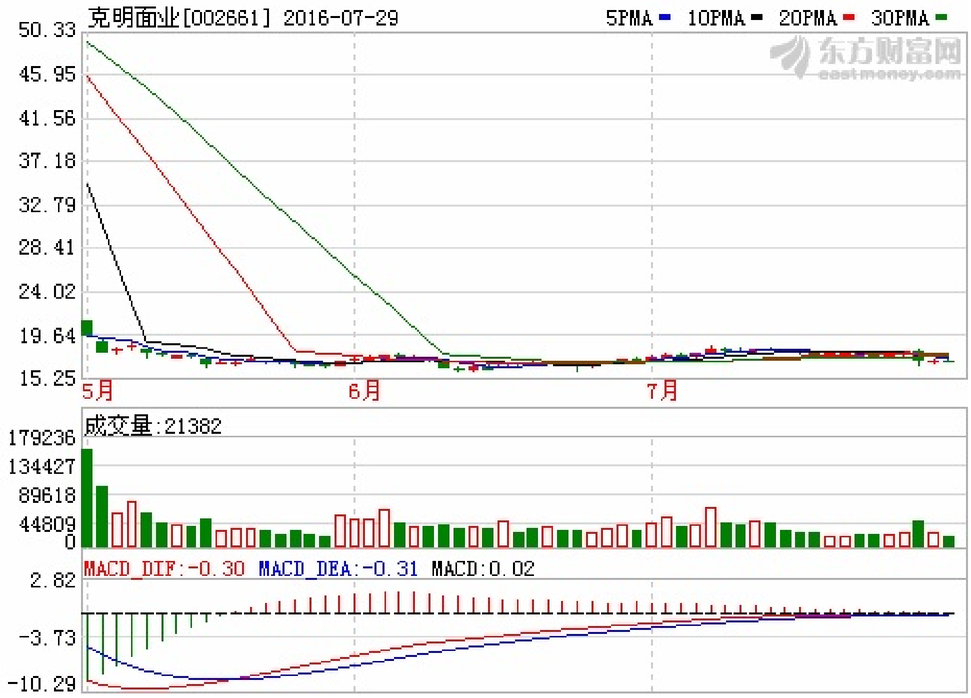
锦龙股份(000712)



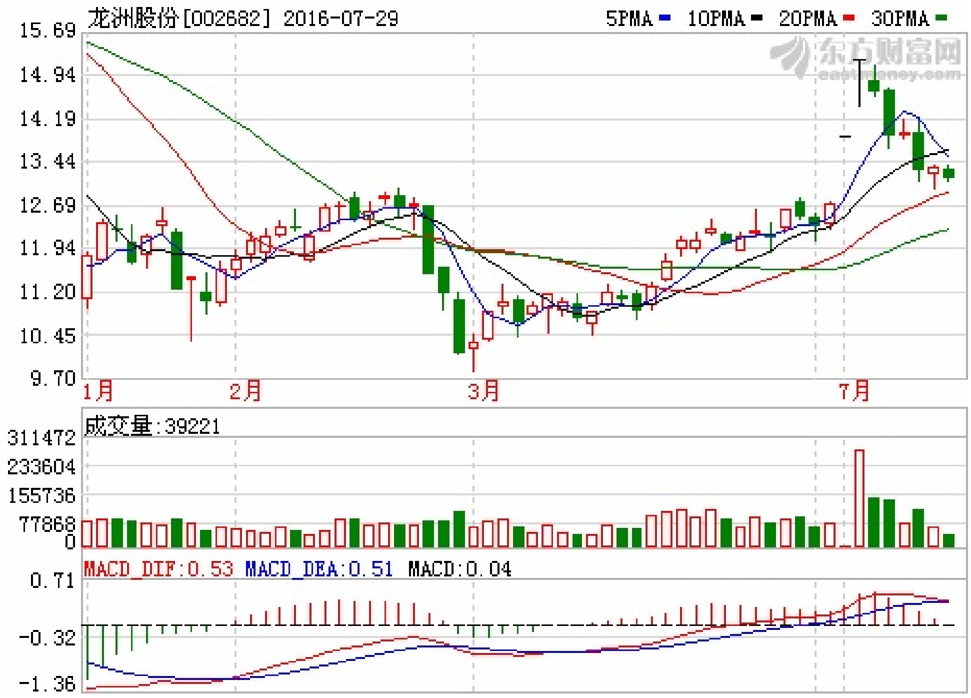
中科金财(002657)



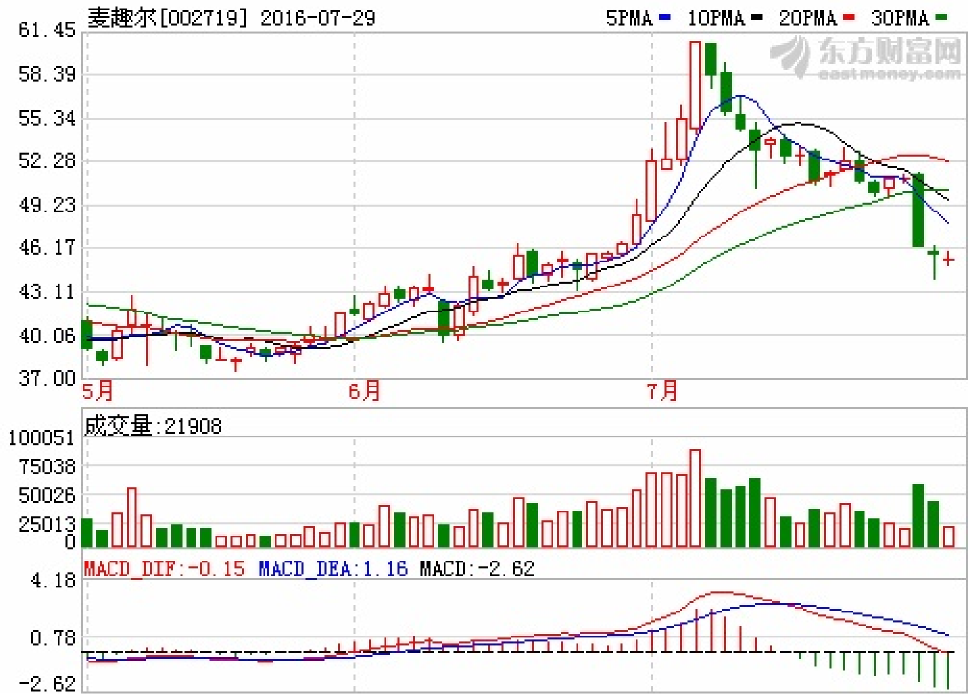
克明面业(002661)



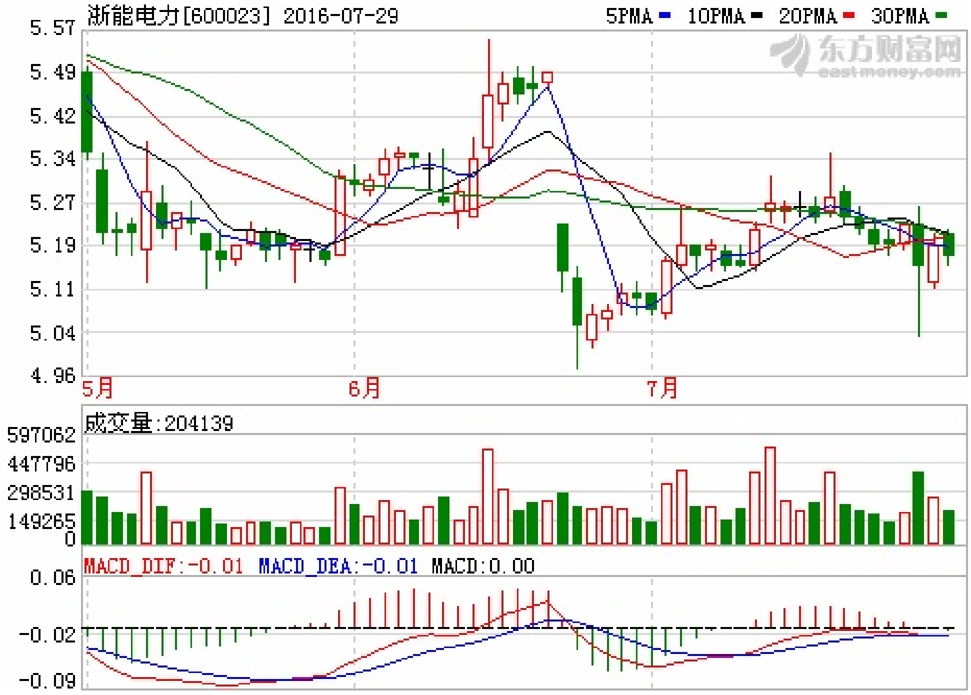
龙洲股份(002682)



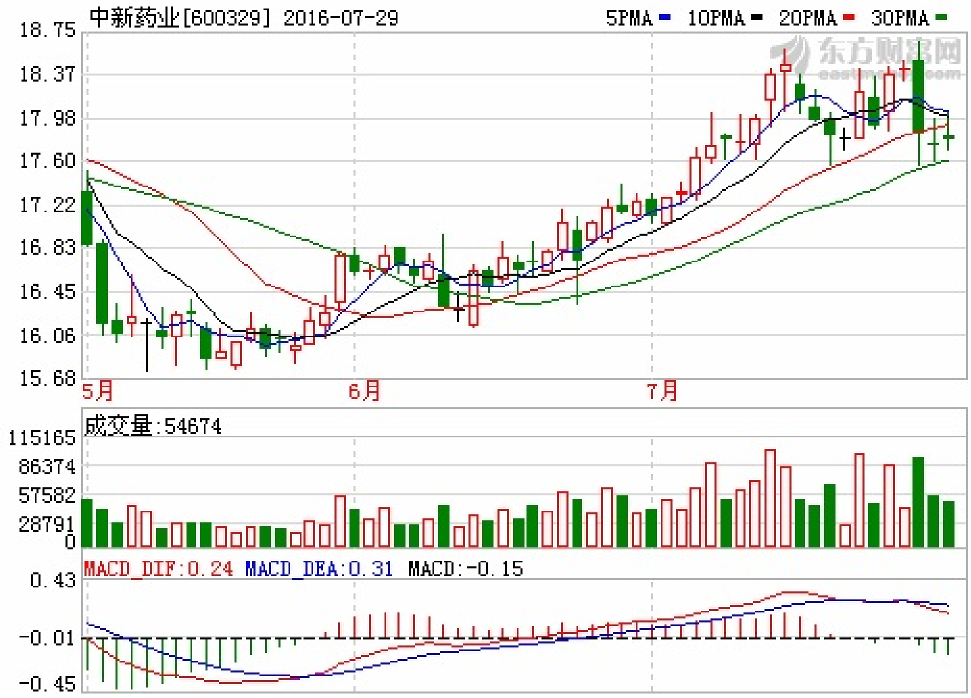
麦趣尔(002719)



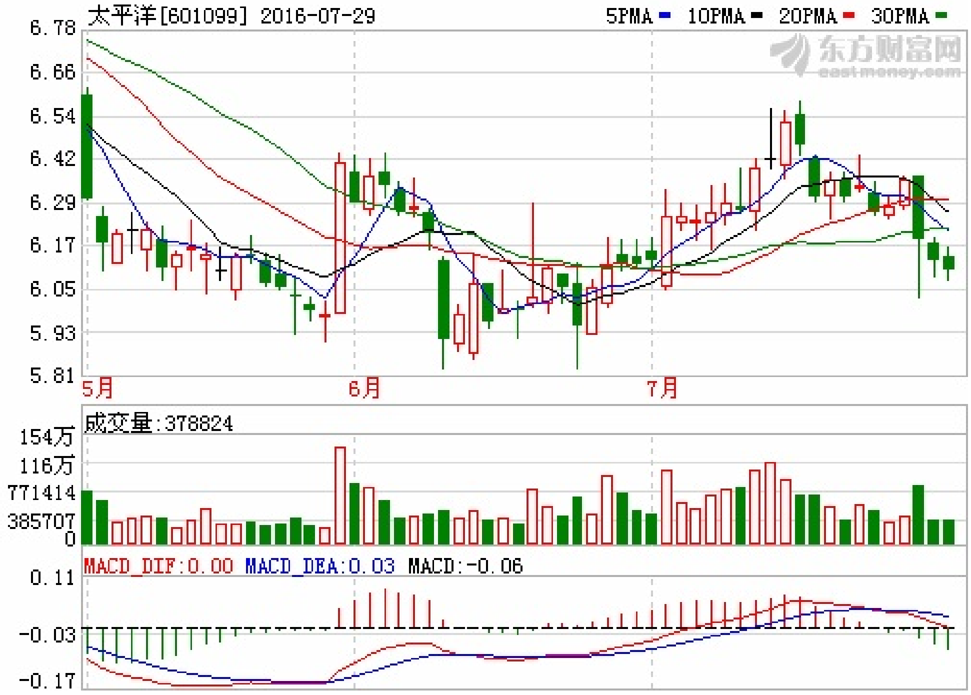
浙能电力(600023)



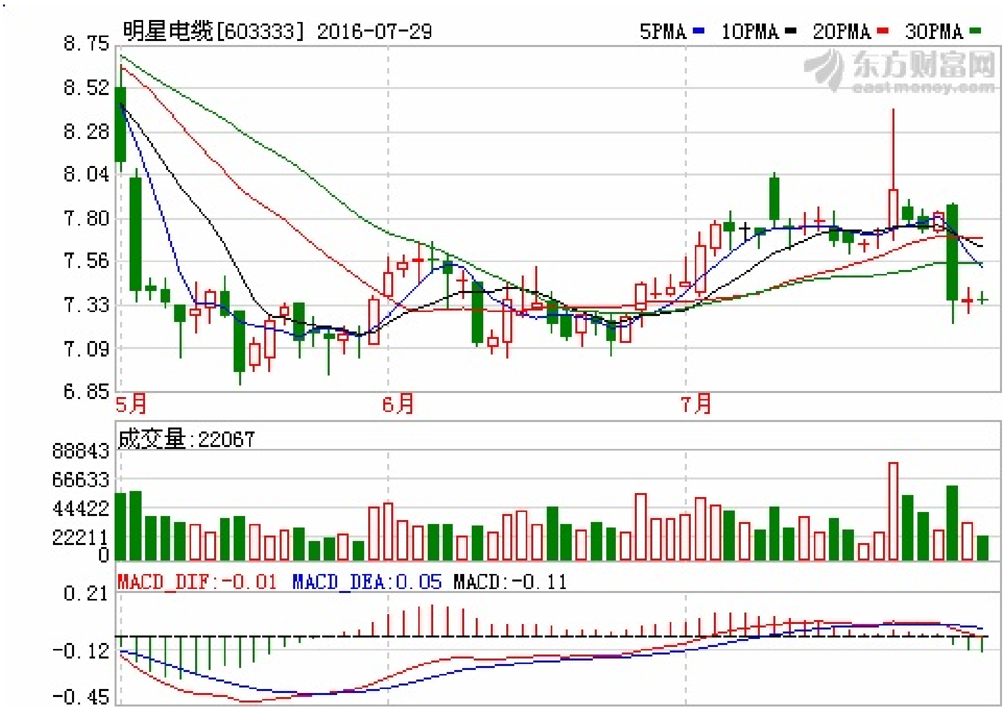
中新药业(600329)



太平洋(601099)



明星电缆(603333)


明星电缆[603333] 2016-07-29
5PMA
10PMA
20PMA
30PMA
东方财富网
eastmoney.com
8.75
8.52
8.28
8.04
7.80
7.56
7.33
7.09
6.85
5月
6月
7月
成交量:22067
88843
66633
44422
22211
0
MACD_DIF:-0.01
MACD_DEA:0.05
MACD:-0.11
0.21
-0.12
-0.45

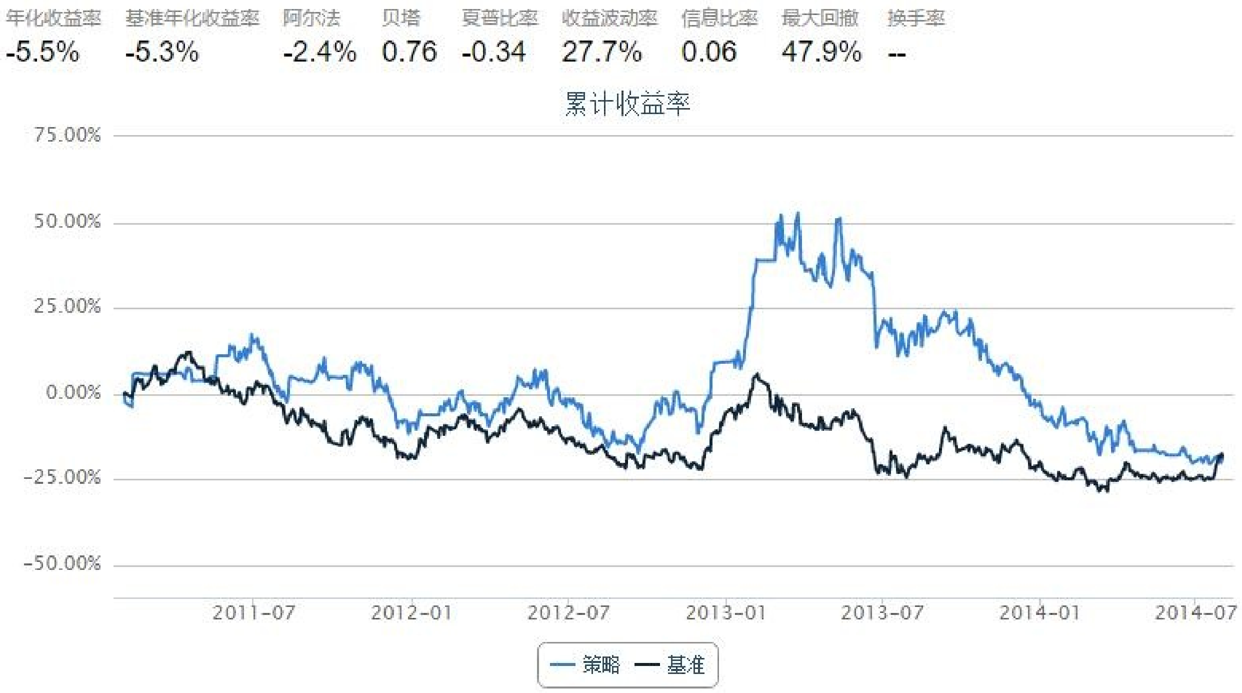
五 量化模型

5.1 动量模型

Momentum策略

> 来源：https://uqer.io/community/share/5526191ef9f06c8f33904571

```python
import pandas as pd
from pandas import Series, DataFrame

start = datetime(2011, 1, 1)
end = datetime(2014, 8, 1)
benchmark = 'SH50'
universe = set_universe('SH50')
capital_base = 100000
refresh_rate = 10

window = 20

def initialize(account):
    account.amount = 300
    add_history('hist', window)


def handle_data(account):
    momentum = {'symbol':[], 'c_ret':[]}
    for stk in account.universe:
        momentum = pd.DataFrame(momentum)
        momentum = momentum.append([{'symbol':stk,'c_ret':account.hist[stk]['closePrice'].iloc[-1]/account.hist[stk]['closePrice'].iloc[0]}])
        momentum = momentum.sort(columns='c_ret').reset_index(drop=True)
        momentum = momentum[len(momentum)*4/5:len(momentum)]
        buylist = momentum['symbol'].tolist()

        for stk in account.stkpos:
            if (stk not in buylist) and (account.stkpos[stk]>0):
                order_to(stk, 0)
        for stk in buylist:
            if account.stkpos.get(stk,0) == 0:
                order_to(stk, account.amount)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
-5.5%	-5.3%	-2.4%	0.76	-0.34	27.7%	0.06	47.9%	--

【小散学量化】-2-动量模型的简单实践

> 来源：https://uqer.io/community/share/5673a22b228e5b8d5bf017ba

本帖将对之前提到的动量模型进行回测，顺便科普一下动量模型中用到的数学工具，包括以下内容：

- 1、线性回归
- 2、动量模型回测
- 3、小散的思考

1.线性回归

线性回归是利用称为线性回归方程的最小平方函数对一个或多个自变量和因变量之间关系进行建模的一种回归分析。

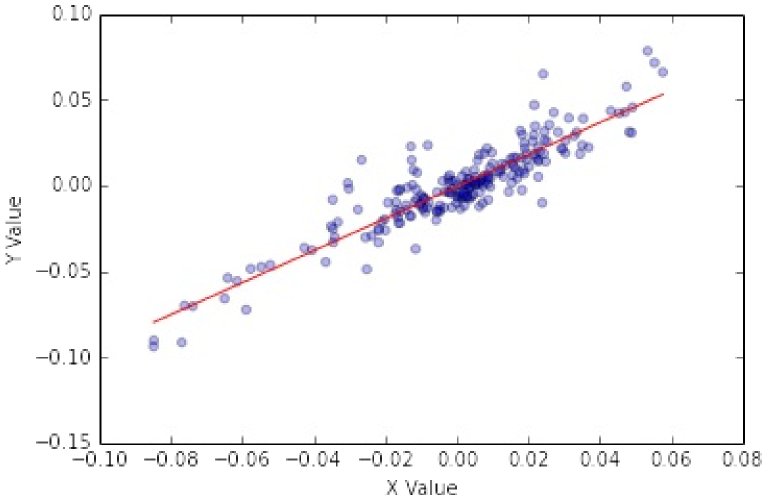
对于自变量X和因变量Y,线性回归方法用Y=α+βX 的形式来解释X,Y之间的关系。例如，小散想知道上证50指数与上证指数之间的变化，该怎么办呢？

首先，定义一个函数来执行线性回归操作，并返回回归结果。

```
import numpy as np
from statsmodels import regression
import statsmodels.api as sm
import matplotlib.pyplot as plt
import math
def linreg(X,Y):
    # Running the linear regression
    X = sm.add_constant(X)
    model = regression.linear_model.OLS(Y, X).fit()
    a = model.params[0]
    b = model.params[1]
    X = X[:, 1]
    # Return summary of the regression and plot results
    X2 = np.linspace(X.min(), X.max(), 100)
    Y_hat = X2 * b + a
    plt.scatter(X, Y, alpha=0.3) # Plot the raw data
    plt.plot(X2, Y_hat, 'r', alpha=0.9);  # Add the regression l
ine, colored in red
    plt.xlabel('X Value')
    plt.ylabel('Y Value')
    return model.summary()
```

接着，应用上面定义的函数，以2015年的数据为例，验证上证指数（000001）和上证50指数（000016）之间的关系。

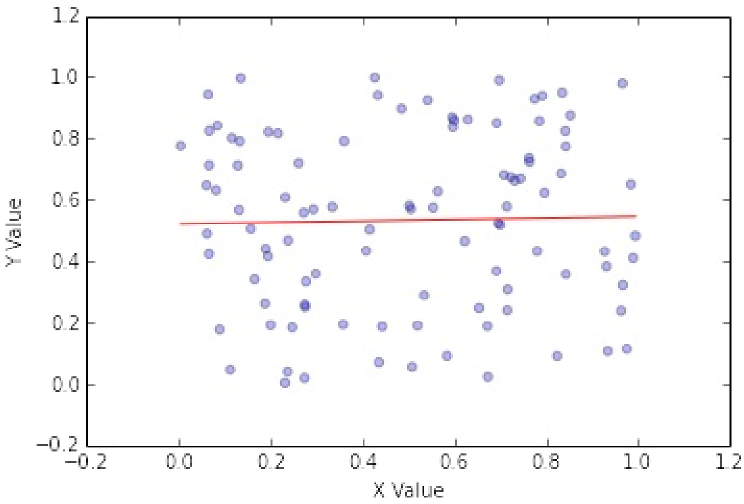
```
data1=DataAPI.MktIdxdGet(ticker='000016',beginDate='20150101',en
dDate='20151201')
asset=data1['closeIndex']
data2=DataAPI.MktIdxdGet(ticker='000001',beginDate='20150101',en
dDate='20151201')
benchmark=data2['closeIndex']
r_a = asset.pct_change()[1:]
r_b = benchmark.pct_change()[1:]
linreg(r_b.values, r_a.values)

OLS Regression Results
Dep. Variable:    y      R-squared:                0.823
Model:    OLS      Adj. R-squared:          0.823
Method:    Least Squares      F-statistic:              1021.
Date:    Fri, 18 Dec 2015      Prob (F-statistic):    2.11e-84
Time:    13:40:10      Log-Likelihood:          685.41
No. Observations:      221      AIC:                     -1367.
Df Residuals:       219      BIC:                     -1360.
Df Model:         1
Covariance Type:    nonrobust

coef    std err    t    P>|t|    [95.0% Conf. Int.]
const        -0.0006         0.001         -0.877      0.381        -0
.002     0.001
x1         0.9317         0.029         31.952      0.000         0.874
    0.989

Omnibus:    43.308       Durbin-Watson:            1.857
Prob(Omnibus):      0.000       Jarque-Bera (JB):         85.122
Skew:     0.968       Prob(JB):               3.28e-19
Kurtosis:      5.344       Cond. No.               39.6
```



小散并不知道OLS结果怎么看，只知道这里的R-sq用来解释线性关系的可靠性，0.823表示自变量可以解释82.3%的因变量变化。小散是一个好奇心很强的人，这个82.3%到底是大还是小呢？

这里使用随机生成X,Y来观察一下。

```
X = np.random.rand(100)
Y = np.random.rand(100)
linreg(X, Y)

OLS Regression Results
Dep. Variable:     y       R-squared:                 0.001
Model:     OLS      Adj. R-squared:          -0.010
Method:     Least Squares       F-statistic:            0.06682
Date:     Fri, 18 Dec 2015       Prob (F-statistic):      0.797
Time:     13:40:15       Log-Likelihood:          -15.737
No. Observations:        100       AIC:                    35.47
Df Residuals:         98       BIC:                      40.68
Df Model:          1
Covariance Type:     nonrobust

coef     std err     t     P>|t|     [95.0% Conf. Int.]
const          0.5214         0.056         9.245      0.000         0
.410       0.633
x1         0.0253         0.098         0.258      0.797        -0.169
     0.219

Omnibus:     22.647       Durbin-Watson:            2.366
Prob(Omnibus):      0.000       Jarque-Bera (JB):         5.557
Skew:     -0.164      Prob(JB):                0.0621
Kurtosis:      1.893       Cond. No.                4.32
```



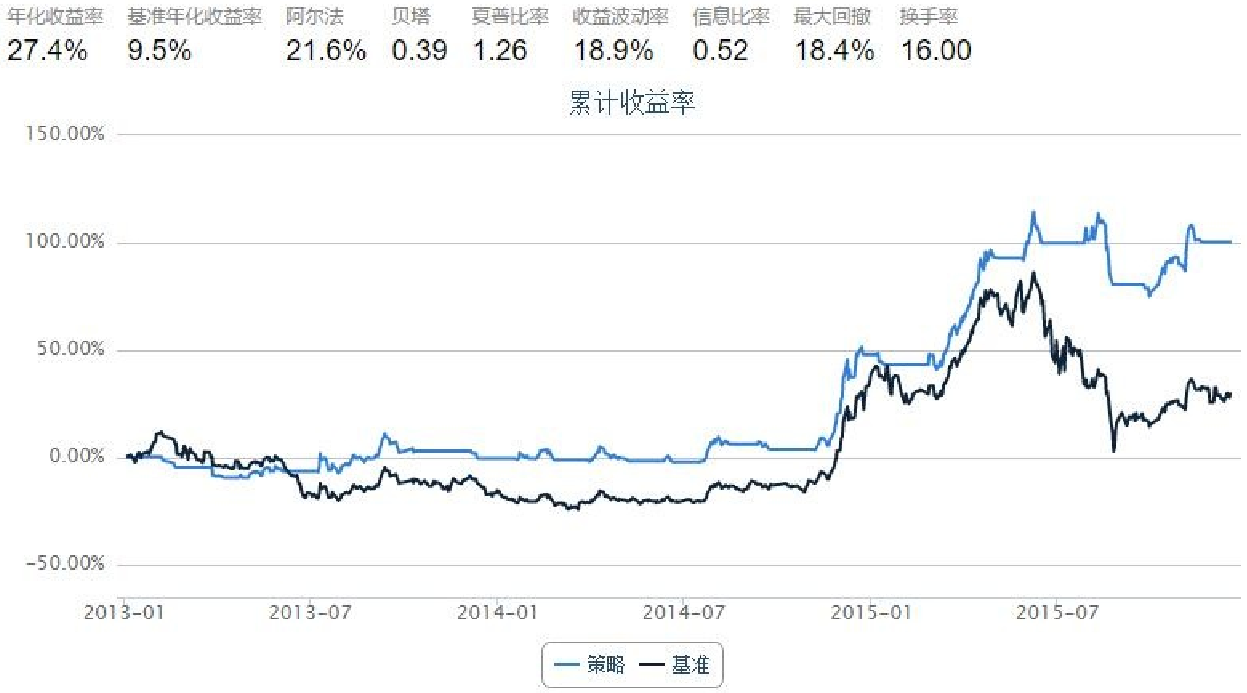
2 动量模型回测

针对小伙伴之前提出的回测效果问题，小散选择了上证50指数作为参考，50ETF作为投资标的，上证指数（000001）作为动量模型的择时工具进行了回测。

首先，定义动量信号函数，函数返回值为回归模型的Y值，同期上证指数值及标准差。

```
from statsmodels import regression
import statsmodels.api as sm
import scipy.stats as stats
import scipy.spatial.distance as distance
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from CAL.PyCAL import *
from datetime import datetime, timedelta
import numpy as np
cal = Calendar('China.SSE')
def signal_gzs_calc(today,before):
    start= cal.advanceDate(today, Period('-30B'))
    start=start.toDateTime()
    end = cal.advanceDate(today, before)
    end=end.toDateTime()
    asset=DataAPI.MktIdxdGet(ticker='000001',beginDate=start,endDate=end,field=['tradeDate','closeIndex'],pandas="1").set_index('tradeDate')['closeIndex']

    # 对收盘价数据进行直线拟合
    X = np.arange(len(asset))
    x = sm.add_constant(X)
    model = regression.linear_model.OLS(asset, x).fit() # 使用OLS计算y=ax+b对应的a和b
    a = model.params[0]
    b = model.params[1]
    Y_fit = X * b + a
    return Y_fit[-1],asset[-1],np.std(asset - Y_fit)
```

```
start = '2013-01-01'                           # 回测起始时间
today = datetime.today()
delta = timedelta(days = -1)
end = (today+delta).strftime('%Y-%m-%d')
benchmark = 'SH50'                                # 策略参考标准
universe = ['510050.XSHG']     # 股票池                            # 策略
参考标准
refresh_rate = 1                              # 调仓频率，即每 refresh
_rate 个交易日执行一次 handle_data() 函数
def initialize(account):                       # 初始化虚拟账户状态
    pass

def handle_data(account):                      # 每个交易日的买入卖出指
令
    today=account.current_date
    a_1,b_1,c_1=signal_gzs_calc(today,Period('-2B'))
    a_2,b_2,c_2=signal_gzs_calc(today,Period('-1B'))
    for stk in account.universe:
        if (a_1-b_1)< -c_1 and (a_2-b_2)> -c_2 and  not stk in a
ccount.valid_secpos:
            order(stk, 200000)

        if (a_1-b_1)<c_1 and (a_2-b_2)>c_2 and stk in account.va
lid_secpos:
            order(stk, -200000)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
27.4%	9.5%	21.6%	0.39	1.26	18.9%	0.52	18.4%	16.00

3、小散的思考

从回测效果来看，本模型在捕捉上升趋势和下降趋势时均有延迟，可以通过修改period来调整（初步设定为30）。

单纯依靠反转信号作为卖出信号带来的回撤较大，可以考虑增加止损条件(无止损设定)。

3年回测期内，共发出买卖信号11次，最近一次空仓信号发出在11月19日，产生的信号较少，可以考虑修改两个if条件增加信号量（小散设定为单位标准差）。

策略收益主要是靠空仓避难得到的，小伙伴可以将这个模型推广应用到多个个股，看看有没有挖掘价值。

参考文献：

1、量化分析师日记，新手必看。

2、zilong.li https://uqer.io/community/share/561e3a65f9f06c4ca82fb5ec 用5日均线和10日均线进行判断--改进版。

本次【小散学量化】到此结束， 韭菜的观点仅供参考，切记切记~

一个追涨的策略(修正版)

> 来源：https://uqer.io/community/share/56677b30f9f06c6c8a91b5e8

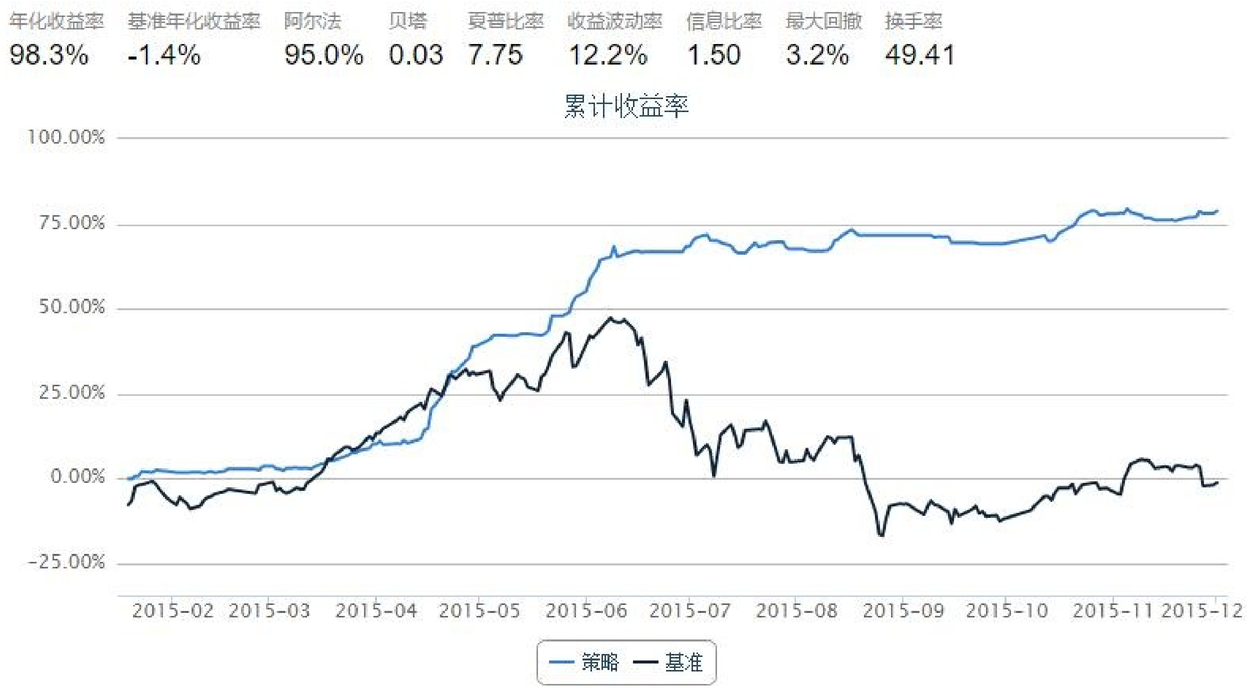
基本思想

每天收盘前一分钟看一下，如果是强势股。则买入。 第二天开盘卖出。 强势股的判断标准：1. 开盘价高于昨日收盘价，也就是有缺口。2. 收盘价接近于当日最高价，也就是收光头阳线 3.收盘价创10日新高

通达信策略公式：

```
ENTERLONG:O>REF(H,1) AND C>H*0.99 AND H=HHV(H,10) AND O/REF(C,1)
<1.09;
EXITLONG:REF(ENTERLONG,1);

{多头买入:开盘价>1日前的最高价 AND 收盘价>最高价*0.99 AND 最高价=10日内
最高价的最高値 AND 开盘价/1日前的收盘价<1.09}
{多头卖出:1日前的ENTERLONG}
```

```
# 第一步：设置基本参数
start = '2015-01-01'
end   = '2015-12-01'
capital_base = 1000000
refresh_rate = 239
benchmark = 'HS300'
freq = 'm'


# 第二步：选择主题，设置股票池
universe = set_universe('HS300')

def initialize(account):                   # 初始化虚拟账户状态
    pass

def handle_data(account):                  # 每个交易日的买入卖出指令

    #print 'start:',account.current_date,account.current_minute
    #print account.current_date
    if(account.current_minute=='09:30'):
        #print '开盘了'
        #print account.referencePrice
        for s in account.valid_secpos:     #清仓
            order_to(s, 0)
    else:
        #print '收盘了'
        c = account.referencePortfolioValue
        today_minutes=account.get_history(238) #今天
        stocks=[]
        for s in account.universe:
            #print s
            open_price=today_minutes[s]['openPrice'][0]  #09:31分钟开盘价
            pre_close_price=today_minutes[s]['closePrice'][-1] #15:28分钟收盘价
            high_price=today_minutes[s]['highPrice'].max() #当日最高价
            last_10day = account.get_daily_history(10) #最近10日
            close_last_day=last_10day[s]['closePrice'][-1] #昨日收盘价
            max_10day=last_10day[s]['highPrice'].max() #最近10日最高价
            #print s,close_last_day,open_price,pre_close_price,high_price
            if open_price>close_last_day and pre_close_price >= max_10day and  pre_close_price> high_price*0.99 :
                stocks.append(s)
                #print s,close_last_day,open_price,pre_close_price,high_price
        # print stocks
        if len(stocks)==0:
            return
        w=min(0.1,1.0/len(stocks))# 最大仓位1/10
        # print w
        for s in stocks:
            p=today_minutes[s]['closePrice'][-1] #15:28分钟收盘价
            num=int(c * w / p)
            order(s, num)
```



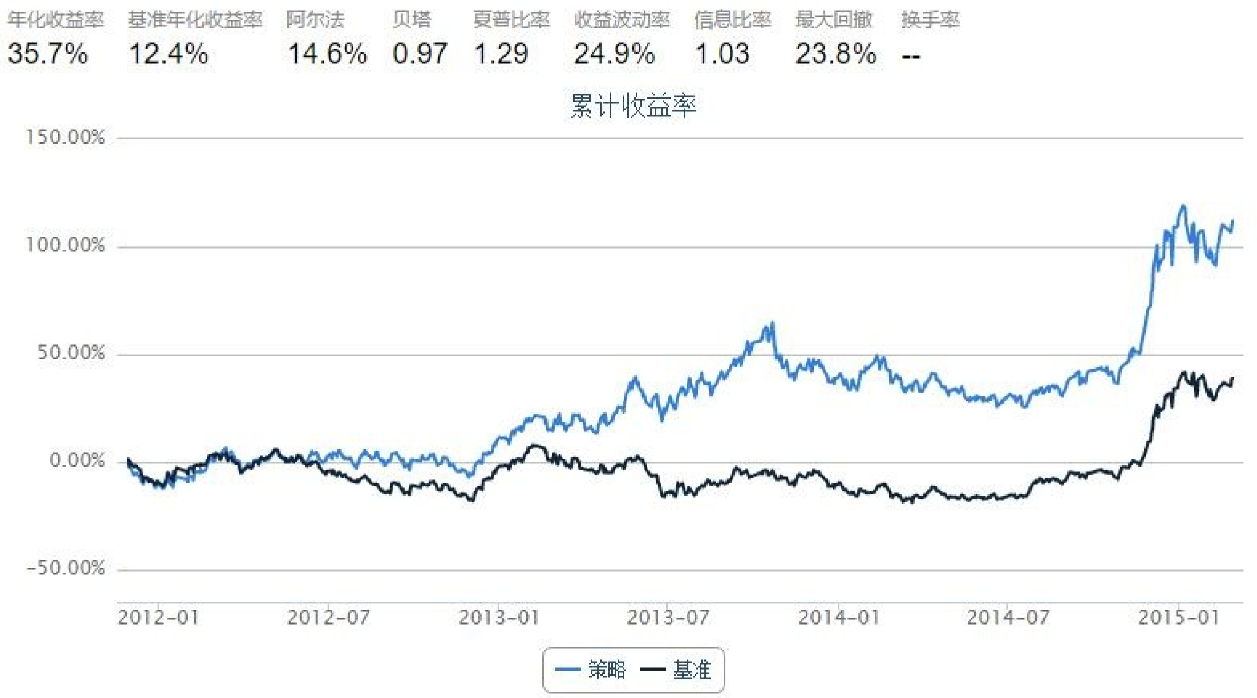
年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
98.3%	-1.4%	95.0%	0.03	7.75	12.2%	1.50	3.2%	49.41

这个模式还没实操过。手动下单在最后1分钟有点匆忙。请教各位，有没有可以程序化下单或者设定条件快速下单的软件？

动量策略（momentum driven）

> 来源：https://uqer.io/community/share/555ed9eef9f06c6c7304f9a9

```
import pandas as pd

start = '2011-11-01'
end   = '2015-03-01'
benchmark = 'HS300'
universe = set_universe('HS300')   # 股票池为沪深300
capital_base = 10000000
refresh_rate = 10

def initialize(account):
    pass

def handle_data(account):
    history = account.get_attribute_history('closePrice', 20)
    momentum = {'symbol':[], 'c_ret':[]}
    for stk in account.universe:
        momentum['symbol'].append(stk)
        momentum['c_ret'].append(history[stk][-1]/history[stk][0
])

    # 按照过去20日收益率排序，并且选择前20%的股票作为买入候选
    momentum = pd.DataFrame(momentum).sort(columns='c_ret').rese
t_index()
    momentum = momentum[len(momentum)*4/5:len(momentum)]   # 选择
    buylist = momentum['symbol'].tolist()
    for stk in account.valid_secpos:
        if stk not in buylist:
            order_to(stk, 0)

    # 等权重买入所选股票
    portfolio_value = account.referencePortfolioValue
    for stk in buylist:
        if stk not in account.valid_secpos:
            order_to(stk, int(portfolio_value / account.referenc
ePrice[stk] / 100.0 / len(buylist))*100)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
35.7%	12.4%	14.6%	0.97	1.29	24.9%	1.03	23.8%	--

```
bt
```

	tradeDate	cash	security_position	portfolio_value	be
0	2011-11-29	1.627636	{u'600809.XSHG': 5700, u'600597.XSHG': 21100, ...	10057034.029636	0.0
1	2011-11-30	1.627636	{u'600809.XSHG': 5700, u'600597.XSHG': 21100, ...	9744501.409636	-0.(
2	2011-12-01	1.627636	{u'600809.XSHG': 5700, u'600597.XSHG': 21100, ...	9865655.500636	0.0
3	2011-12-02	1.627636	{u'600809.XSHG': 5700, u'600597.XSHG': 21100, ...	9708455.312636	-0.(
4	2011-12-05	1.627636	{u'600809.XSHG': 5700, u'600597.XSHG': 21100, ...	9406180.945636	-0.(
			{u'600809.XSHG':		

5	2011-12-06	1.627636	5700, u'600597.XSHG': 21100, ...	9415313.949636	-0.0
6	2011-12-07	1.627636	{u'600809.XSHG': 5700, u'600597.XSHG': 21100, ...	9454383.554636	0.0
7	2011-12-08	1.627636	{u'600809.XSHG': 5700, u'600597.XSHG': 21100, ...	9492264.755636	-0.0
8	2011-12-09	1.627636	{u'600809.XSHG': 5700, u'600597.XSHG': 21100, ...	9410892.851636	-0.0
9	2011-12-12	1.627636	{u'600809.XSHG': 5700, u'600597.XSHG': 21100, ...	9280515.559636	-0.0
10	2011-12-13	6.144010	{u'000423.XSHE': 4200, u'600036.XSHG': 16300, ...	9116051.541010	-0.0
11	2011-12-14	6.144010	{u'000423.XSHE': 4200, u'600036.XSHG': 16300, ...	9042660.489010	-0.0
12	2011-12-15	6.144010	{u'000423.XSHE': 4200, u'600036.XSHG': 16300, ...	8973191.009010	-0.0
13	2011-12-16	6.144010	{u'000423.XSHE': 4200, u'600036.XSHG': 16300, ...	9203671.207010	0.0
14	2011-12-19	6.144010	{u'000423.XSHE': 4200, u'600036.XSHG': 16300, ...	9272914.609010	-0.0
15	2011-12-20	6.144010	{u'000423.XSHE': 4200, u'600036.XSHG':	9245375.546010	-0.0

			16300, ...		
16	2011-12-21	6.144010	{u'000423.XSHE': 4200, u'600036.XSHG': 16300, ...	9134328.126010	-0.(
17	2011-12-22	6.144010	{u'000423.XSHE': 4200, u'600036.XSHG': 16300, ...	9174823.752010	0.0
18	2011-12-23	6.144010	{u'000423.XSHE': 4200, u'600036.XSHG': 16300, ...	9185929.702010	0.0
19	2011-12-26	6.144010	{u'000423.XSHE': 4200, u'600036.XSHG': 16300, ...	9054507.546010	-0.(
20	2011-12-27	9.172263	{u'601328.XSHG': 41600, u'600036.XSHG': 16300,...	8855733.701263	-0.(
21	2011-12-28	9.172263	{u'601328.XSHG': 41600, u'600036.XSHG': 16300,...	8820469.780263	0.0
22	2011-12-29	9.172263	{u'601328.XSHG': 41600, u'600036.XSHG': 16300,...	8836530.533263	0.0
23	2011-12-30	9.172263	{u'601328.XSHG': 41600, u'600036.XSHG': 16300,...	8976704.526263	0.0
24	2012-01-04	9.172263	{u'601328.XSHG': 41600, u'600036.XSHG': 16300,...	8858664.267263	-0.(
25	2012-01-05	9.172263	{u'601328.XSHG': 41600, u'600036.XSHG': 16300,...	8785873.316263	-0.(

26	2012-01-06	9.172263	{u'601328.XSHG': 41600, u'600036.XSHG': 16300,...	8790219.626263	0.0
27	2012-01-09	9.172263	{u'601328.XSHG': 41600, u'600036.XSHG': 16300,...	9032347.125263	0.0
28	2012-01-10	9.172263	{u'601328.XSHG': 41600, u'600036.XSHG': 16300,...	9247829.118263	0.0
29	2012-01-11	9.172263	{u'601328.XSHG': 41600, u'600036.XSHG': 16300,...	9186427.159263	-0.0
...	...	...	...	...	...
755	2015-01-12	1.942737	{u'600036.XSHG': 23600, u'000776.XSHE': 22100,...	20280828.092737	-0.0
756	2015-01-13	1.942737	{u'600036.XSHG': 23600, u'000776.XSHE': 22100,...	20204218.432737	0.0
757	2015-01-14	1.942737	{u'600036.XSHG': 23600, u'000776.XSHE': 22100,...	20223885.602737	-0.0
758	2015-01-15	1.942737	{u'600036.XSHG': 23600, u'000776.XSHE': 22100,...	20962393.212737	0.0
759	2015-01-16	1.942737	{u'600036.XSHG': 23600, u'000776.XSHE': 22100,...	21076614.722737	0.0
760	2015-01-19	3.731317	{u'601328.XSHG': 49500, u'600066.XSHG': 21600,...	19298994.131317	-0.0

761	2015-01-20	3.731317	{u'601328.XSHG': 49500, u'600066.XSHG': 21600,...	19641436.921317	0.0
762	2015-01-21	3.731317	{u'601328.XSHG': 49500, u'600066.XSHG': 21600,...	20548813.131317	0.0
763	2015-01-22	3.731317	{u'601328.XSHG': 49500, u'600066.XSHG': 21600,...	20654164.321317	0.0
764	2015-01-23	3.731317	{u'601328.XSHG': 49500, u'600066.XSHG': 21600,...	20704527.401317	0.0
765	2015-01-26	3.731317	{u'601328.XSHG': 49500, u'600066.XSHG': 21600,...	20741863.411317	0.0
766	2015-01-27	3.731317	{u'601328.XSHG': 49500, u'600066.XSHG': 21600,...	20501439.751317	-0.0
767	2015-01-28	3.731317	{u'601328.XSHG': 49500, u'600066.XSHG': 21600,...	20072450.291317	-0.0
768	2015-01-29	3.731317	{u'601328.XSHG': 49500, u'600066.XSHG': 21600,...	19879912.781317	-0.0
769	2015-01-30	3.731317	{u'601328.XSHG': 49500, u'600066.XSHG': 21600,...	19612742.771317	-0.0
770	2015-02-02	6.734377	{u'002153.XSHE': 4400, u'600498.XSHG': 19400, ...	19457712.474377	-0.0
	2015-02-		{u'002153.XSHE': 4400,		

	03		u'600498.XSHG': 19400, ...		
772	2015-02-04	6.734377	{u'002153.XSHE': 4400, u'600498.XSHG': 19400, ...	19714886.724377	-0.
773	2015-02-05	6.734377	{u'002153.XSHE': 4400, u'600498.XSHG': 19400, ...	19679598.144377	-0.
774	2015-02-06	6.734377	{u'002153.XSHE': 4400, u'600498.XSHG': 19400, ...	19297569.294377	-0.
775	2015-02-09	6.734377	{u'002153.XSHE': 4400, u'600498.XSHG': 19400, ...	19131675.244377	0.0
776	2015-02-10	6.734377	{u'002153.XSHE': 4400, u'600498.XSHG': 19400, ...	19621103.094377	0.0
777	2015-02-11	6.734377	{u'002153.XSHE': 4400, u'600498.XSHG': 19400, ...	20007845.234377	0.0
778	2015-02-12	6.734377	{u'002153.XSHE': 4400, u'600498.XSHG': 19400, ...	20139049.804377	0.0
779	2015-02-13	6.734377	{u'002153.XSHE': 4400, u'600498.XSHG': 19400, ...	20416511.184377	0.0
780	2015-02-16	5.417277	{u'002153.XSHE': 4400, u'600498.XSHG': 19400, ...	21015638.057277	0.0
781	2015-02-17	5.417277	{u'002153.XSHE': 4400, u'600498.XSHG': 19400, ...	20991534.387277	0.0

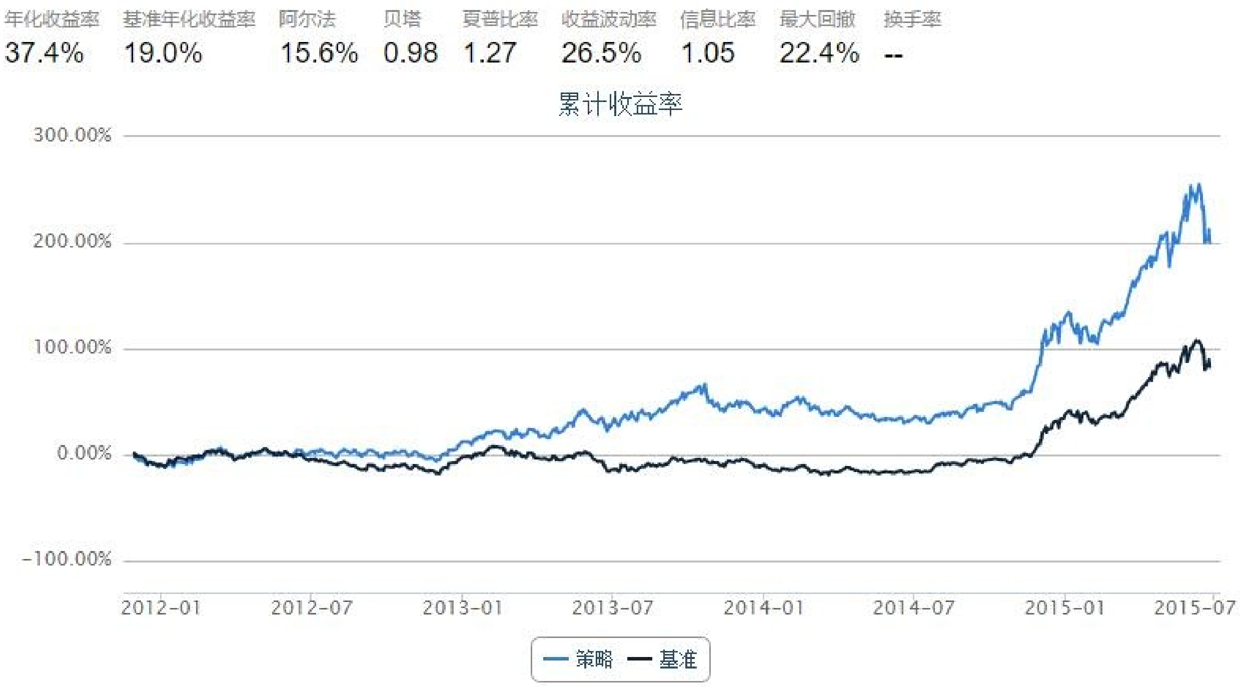
			19400, ...		
782	2015-02-25	5.417277	{u'002153.XSHE': 4400, u'600498.XSHG': 19400, ...	20681973.077277	-0.0
783	2015-02-26	5.417277	{u'002153.XSHE': 4400, u'600498.XSHG': 19400, ...	20936634.347277	0.0
784	2015-02-27	5.417277	{u'002153.XSHE': 4400, u'600498.XSHG': 19400, ...	21204596.757277	0.0

```
785 rows × 6 columns
```

动量策略（momentum driven）——修正版

> 来源：https://uqer.io/community/share/5590a4d4f9f06cb5604f187e

```
import pandas as pd

start = '2011-11-01'
end   = '2015-06-25'
benchmark = 'HS300'
universe = set_universe('HS300')    # 股票池为沪深300
capital_base = 10000000
refresh_rate = 10

def initialize(account):
    pass

def handle_data(account):
    history = account.get_attribute_history('closePrice', 20)
    momentum = {'symbol':[], 'c_ret':[]}
    for stk in account.universe:
        momentum['symbol'].append(stk)
        momentum['c_ret'].append(history[stk][-1]/history[stk][0
])

    # 按照过去20日收益率排序，并且选择前20%的股票作为买入候选
    momentum = pd.DataFrame(momentum).sort(columns='c_ret').rese
t_index()
    momentum = momentum[len(momentum)*4/5:len(momentum)]   # 选择
    buylist = momentum['symbol'].tolist()
    for stk in account.valid_secpos:
        if stk not in buylist:
            order_to(stk, 0)

    # 等权重买入所选股票
    portfolio_value = account.referencePortfolioValue

    filteredBuylist = []
    for stk in buylist:
        if not np.isnan(account.referencePrice[stk]):
            filteredBuylist.append(stk)

    print account.current_date, filteredBuylist

    for stk in filteredBuylist:
        if stk not in account.valid_secpos:
            order_to(stk, int(portfolio_value / account.referenc
ePrice[stk] / 100.0 / len(buylist))*100)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
37.4%	19.0%	15.6%	0.98	1.27	26.5%	1.05	22.4%	--

```
2011-11-29 00:00:00 ['000963.XSHE', '000061.XSHE', '601106.XSHG'
, '600108.XSHG', '300251.XSHE', '000400.XSHE', '600519.XSHG', '6
00100.XSHG', '600157.XSHG', '000999.XSHE', '002065.XSHE', '00089
5.XSHE', '600177.XSHG', '002008.XSHE', '002230.XSHE', '000858.XS
HE', '600578.XSHG', '000729.XSHE', '002353.XSHE', '002465.XSHE',
 '000009.XSHE', '600256.XSHG', '600998.XSHG', '002385.XSHE', '30
0133.XSHE', '002007.XSHE', '002450.XSHE', '000581.XSHE', '600535
.XSHG', '300003.XSHE', '600008.XSHG', '000503.XSHE', '600406.XSH
G', '600315.XSHG', '000413.XSHE', '600809.XSHG', '002292.XSHE',
'600085.XSHG', '000538.XSHE', '600718.XSHG', '000423.XSHE', '300
027.XSHE', '600600.XSHG', '300070.XSHE', '600276.XSHG', '600703.
XSHG', '600066.XSHG', '600597.XSHG', '300146.XSHE', '000539.XSHE
', '300058.XSHE', '600485.XSHG', '600373.XSHG', '600637.XSHG', '
002570.XSHE']
2011-12-13 00:00:00 ['000895.XSHE', '601318.XSHG', '002142.XSHE'
, '600028.XSHG', '601766.XSHG', '000963.XSHE', '601398.XSHG', '6
01166.XSHG', '600383.XSHG', '601328.XSHG', '000423.XSHE', '00235
3.XSHE', '000001.XSHE', '000538.XSHE', '002304.XSHE', '600315.XS
HG', '002385.XSHE', '601988.XSHG', '000002.XSHE', '600276.XSHG',
 '600519.XSHG', '000046.XSHE', '601169.XSHG', '601288.XSHG', '00
0858.XSHE', '600108.XSHG', '002570.XSHE', '002292.XSHE', '601939
.XSHG', '600535.XSHG', '002038.XSHE', '601628.XSHG', '600177.XSH
G', '600893.XSHG', '600048.XSHG', '000999.XSHE', '600637.XSHG',
'000069.XSHE', '600867.XSHG', '600015.XSHG', '300058.XSHE', '601
333.XSHG', '600016.XSHG', '000024.XSHE', '600036.XSHG', '002146.
XSHE', '601888.XSHG', '000539.XSHE', '002065.XSHE', '601009.XSHG
', '600066.XSHG', '002236.XSHE', '000738.XSHE', '601555.XSHG', '
601928.XSHG']
2011-12-27 00:00:00 ['600000.XSHG', '600633.XSHG', '002007.XSHE'
, '600028.XSHG', '600100.XSHG', '600863.XSHG', '600741.XSHG', '6
00660.XSHG', '600153.XSHG', '002142.XSHE', '000738.XSHE', '60064
2.XSHG', '000402.XSHE', '600066.XSHG', '601328.XSHG', '601988.XS
```

```
HG', '000963.XSHE', '601398.XSHG', '600104.XSHG', '601166.XSHG',
 '002292.XSHE', '300017.XSHE', '601933.XSHG', '600016.XSHG', '60
0019.XSHG', '601601.XSHG', '601857.XSHG', '600867.XSHG', '600690
.XSHG', '601288.XSHG', '002065.XSHE', '600036.XSHG', '601991.XSH
G', '601009.XSHG', '002252.XSHE', '000581.XSHE', '601633.XSHG',
'600027.XSHG', '601333.XSHG', '600795.XSHG', '000069.XSHE', '600
893.XSHG', '000539.XSHE', '000002.XSHE', '600015.XSHG', '600011.
XSHG', '002415.XSHE', '600383.XSHG', '600048.XSHG', '000024.XSHE
', '002236.XSHE', '000046.XSHE', '002146.XSHE', '601555.XSHG', '
601336.XSHG', '601928.XSHG']
2012-01-12 00:00:00 ['002450.XSHE', '600170.XSHG', '000024.XSHE'
, '600036.XSHG', '601699.XSHG', '600115.XSHG', '600886.XSHG', '6
00642.XSHG', '600048.XSHG', '600795.XSHG', '600028.XSHG', '00093
7.XSHE', '000157.XSHE', '600015.XSHG', '601601.XSHG', '601992.XS
HG', '600066.XSHG', '600340.XSHG', '600372.XSHG', '601111.XSHG',
 '601808.XSHG', '600369.XSHG', '000651.XSHE', '600578.XSHG', '60
1898.XSHG', '000970.XSHE', '600031.XSHG', '300146.XSHE', '000425
.XSHE', '002594.XSHE', '000100.XSHE', '600348.XSHG', '000629.XSH
E', '601933.XSHG', '600690.XSHG', '601633.XSHG', '000539.XSHE',
'002292.XSHE', '600309.XSHG', '002146.XSHE', '601377.XSHG', '600
383.XSHG', '600060.XSHG', '000625.XSHE', '000046.XSHE', '601991.
XSHG', '600415.XSHG', '600741.XSHG', '000800.XSHE', '600010.XSHG
', '600027.XSHG', '600011.XSHG', '600100.XSHG', '600104.XSHG', '
601336.XSHG']
2012-02-02 00:00:00 ['600317.XSHG', '600157.XSHG', '600383.XSHG'
, '000783.XSHE', '600030.XSHG', '002202.XSHE', '600585.XSHG', '6
01866.XSHG', '601166.XSHG', '000623.XSHE', '600028.XSHG', '00087
8.XSHE', '601168.XSHG', '600309.XSHG', '601336.XSHG', '601318.XS
HG', '000009.XSHE', '600104.XSHG', '000800.XSHE', '600153.XSHG',
 '000039.XSHE', '601958.XSHG', '600837.XSHG', '000625.XSHE', '60
1808.XSHG', '600188.XSHG', '600340.XSHG', '000937.XSHE', '600166
.XSHG', '000060.XSHE', '002008.XSHE', '601699.XSHG', '000425.XSH
E', '600741.XSHG', '000100.XSHE', '600031.XSHG', '002129.XSHE',
'601899.XSHG', '600010.XSHG', '600362.XSHG', '000157.XSHE', '600
739.XSHG', '000776.XSHE', '600547.XSHG', '600549.XSHG', '600497.
XSHG', '000970.XSHE', '600100.XSHG', '600348.XSHG', '600489.XSHG
', '000630.XSHE', '601377.XSHG', '600111.XSHG', '600060.XSHG', '
000960.XSHE', '002653.XSHE']
2012-02-16 00:00:00 ['601186.XSHG', '601899.XSHG', '000623.XSHE'
, '002241.XSHE', '002385.XSHE', '002304.XSHE', '600718.XSHG', '6
01958.XSHG', '000400.XSHE', '002008.XSHE', '601607.XSHG', '00208
1.XSHE', '600372.XSHG', '600893.XSHG', '600497.XSHG', '600837.XS
HG', '600549.XSHG', '600030.XSHG', '000738.XSHE', '601318.XSHG',
 '600111.XSHG', '000009.XSHE', '600153.XSHG', '000776.XSHE', '60
0271.XSHG', '601106.XSHG', '600157.XSHG', '600108.XSHG', '000060
.XSHE', '600166.XSHG', '300017.XSHE', '600809.XSHG', '000750.XSH
E', '601117.XSHG', '600362.XSHG', '600516.XSHG', '601866.XSHG',
'000157.XSHE', '002001.XSHE', '600547.XSHG', '600277.XSHG', '002
375.XSHE', '601989.XSHG', '002230.XSHE', '601118.XSHG', '000630.
XSHE', '600489.XSHG', '002456.XSHE', '600637.XSHG', '601919.XSHG
', '600060.XSHG', '000960.XSHE', '002129.XSHE', '600340.XSHG', '
601258.XSHG', '002653.XSHE']
2012-03-01 00:00:00 ['000883.XSHE', '002001.XSHE', '000738.XSHE'
```

```
, '600703.XSHG', '000063.XSHE', '300017.XSHE', '002081.XSHE', '6
01633.XSHG', '002304.XSHE', '000625.XSHE', '300146.XSHE', '60025
6.XSHG', '600873.XSHG', '002465.XSHE', '000060.XSHE', '002422.XS
HE', '600690.XSHG', '300251.XSHE', '000768.XSHE', '600570.XSHG',
 '002385.XSHE', '600893.XSHG', '002008.XSHE', '002450.XSHE', '60
0157.XSHG', '600340.XSHG', '600406.XSHG', '600166.XSHG', '000686
.XSHE', '000712.XSHE', '600398.XSHG', '600804.XSHG', '600372.XSH
G', '000917.XSHE', '600271.XSHG', '002146.XSHE', '600208.XSHG',
'000750.XSHE', '600252.XSHG', '600839.XSHG', '002310.XSHE', '601
992.XSHG', '600111.XSHG', '002375.XSHE', '600809.XSHG', '601555.
XSHG', '300059.XSHE', '000046.XSHE', '002230.XSHE', '600060.XSHG
', '600637.XSHG', '300104.XSHE', '002456.XSHE', '601258.XSHG', '
601929.XSHG', '601231.XSHG']
2012-03-15 00:00:00 ['002465.XSHE', '002304.XSHE', '601633.XSHG'
, '600383.XSHG', '600549.XSHG', '600066.XSHG', '002353.XSHE', '6
00485.XSHG', '000651.XSHE', '600809.XSHG', '600369.XSHG', '00056
8.XSHE', '002415.XSHE', '000625.XSHE', '600256.XSHG', '601699.XS
HG', '601688.XSHG', '000750.XSHE', '600010.XSHG', '601216.XSHG',
 '002230.XSHE', '000917.XSHE', '000728.XSHE', '600208.XSHG', '00
2252.XSHE', '002024.XSHE', '000883.XSHE', '000800.XSHE', '601788
.XSHG', '600157.XSHG', '002310.XSHE', '000783.XSHE', '600703.XSH
G', '300251.XSHE', '000712.XSHE', '601555.XSHG', '300059.XSHE',
'002146.XSHE', '000686.XSHE', '000776.XSHE', '000046.XSHE', '000
970.XSHE', '300146.XSHE', '002456.XSHE', '600690.XSHG', '600998.
XSHG', '002375.XSHE', '600739.XSHG', '600150.XSHG', '600108.XSHG
', '300104.XSHE', '600060.XSHG', '600111.XSHG', '601929.XSHG', '
601231.XSHG', '601800.XSHG']
2012-03-29 00:00:00 ['000712.XSHE', '300104.XSHE', '000776.XSHE'
, '600369.XSHG', '002410.XSHE', '601333.XSHG', '601098.XSHG', '6
01958.XSHG', '000729.XSHE', '600900.XSHG', '600519.XSHG', '00257
0.XSHE', '600276.XSHG', '600109.XSHG', '600518.XSHG', '002146.XS
HE', '000800.XSHE', '002294.XSHE', '002236.XSHE', '600196.XSHG',
 '300058.XSHE', '000963.XSHE', '002415.XSHE', '002310.XSHE', '60
0588.XSHG', '002385.XSHE', '002594.XSHE', '000728.XSHE', '002304
.XSHE', '002252.XSHE', '600108.XSHG', '000413.XSHE', '000839.XSH
E', '000999.XSHE', '000778.XSHE', '600352.XSHG', '002450.XSHE',
'002153.XSHE', '300251.XSHE', '600597.XSHG', '600150.XSHG', '002
353.XSHE', '300017.XSHE', '002344.XSHE', '000883.XSHE', '000970.
XSHE', '600332.XSHG', '002024.XSHE', '600739.XSHG', '600998.XSHG
', '600010.XSHG', '600549.XSHG', '600170.XSHG', '000750.XSHE', '
600111.XSHG', '601800.XSHG']
2012-04-17 00:00:00 ['300058.XSHE', '600741.XSHG', '000970.XSHE'
, '600597.XSHG', '601958.XSHG', '000883.XSHE', '600637.XSHG', '6
01336.XSHG', '600398.XSHG', '601933.XSHG', '000686.XSHE', '60111
7.XSHG', '600104.XSHG', '000002.XSHE', '000009.XSHE', '002410.XS
HE', '600157.XSHG', '601688.XSHG', '601888.XSHG', '000625.XSHE',
 '600153.XSHG', '002344.XSHE', '300017.XSHE', '000651.XSHE', '00
0783.XSHE', '000413.XSHE', '601788.XSHG', '000069.XSHE', '000728
.XSHE', '600208.XSHG', '600519.XSHG', '600585.XSHG', '600648.XSH
G', '600549.XSHG', '600999.XSHG', '000776.XSHE', '002450.XSHE',
'601866.XSHG', '002236.XSHE', '600649.XSHG', '600111.XSHG', '600
837.XSHG', '600030.XSHG', '600633.XSHG', '600066.XSHG', '600048.
XSHG', '002146.XSHE', '600383.XSHG', '600010.XSHG', '000629.XSHE
```

```
', '300146.XSHE', '002153.XSHE', '600170.XSHG', '002594.XSHE', '
000750.XSHE']
2012-05-03 00:00:00 ['600999.XSHG', '000712.XSHE', '000878.XSHE'
, '600362.XSHG', '600372.XSHG', '601318.XSHG', '600383.XSHG', '0
00425.XSHE', '000783.XSHE', '000623.XSHE', '600030.XSHG', '00214
2.XSHE', '600663.XSHG', '600570.XSHG', '600369.XSHG', '600648.XS
HG', '000039.XSHE', '601699.XSHG', '601628.XSHG', '600809.XSHG',
 '600150.XSHG', '601601.XSHG', '600115.XSHG', '600348.XSHG', '60
1099.XSHG', '601118.XSHG', '601117.XSHG', '600031.XSHG', '000625
.XSHE', '000024.XSHE', '600166.XSHG', '000157.XSHE', '000630.XSH
E', '300059.XSHE', '600048.XSHG', '601336.XSHG', '000776.XSHE',
'000629.XSHE', '000686.XSHE', '000937.XSHE', '601633.XSHG', '601
688.XSHG', '000970.XSHE', '600170.XSHG', '600157.XSHG', '600208.
XSHG', '600649.XSHG', '000983.XSHE', '601901.XSHG', '601377.XSHG
', '600109.XSHG', '002500.XSHE', '601555.XSHG', '000750.XSHE', '
603000.XSHG']
2012-05-17 00:00:00 ['600010.XSHG', '002415.XSHE', '002353.XSHE'
, '002475.XSHE', '600256.XSHG', '600208.XSHG', '002236.XSHE', '6
00369.XSHG', '600600.XSHG', '601688.XSHG', '600783.XSHG', '00062
9.XSHE', '601179.XSHG', '601607.XSHG', '601377.XSHG', '600570.XS
HG', '601336.XSHG', '600315.XSHG', '300059.XSHE', '601727.XSHG',
 '000061.XSHE', '600703.XSHG', '000400.XSHE', '300003.XSHE', '60
0048.XSHG', '000999.XSHE', '600196.XSHG', '002294.XSHE', '600998
.XSHG', '000826.XSHE', '002146.XSHE', '000963.XSHE', '600166.XSH
G', '601808.XSHG', '000883.XSHE', '300146.XSHE', '600690.XSHG',
'600674.XSHG', '000878.XSHE', '601633.XSHG', '002241.XSHE', '002
422.XSHE', '601901.XSHG', '600372.XSHG', '300070.XSHE', '300024.
XSHE', '600170.XSHG', '600109.XSHG', '600252.XSHG', '601555.XSHG
', '600111.XSHG', '000970.XSHE', '000750.XSHE', '600332.XSHG', '
002673.XSHE', '603000.XSHG']
2012-05-31 00:00:00 ['000402.XSHE', '000917.XSHE', '000046.XSHE'
, '000539.XSHE', '000157.XSHE', '002310.XSHE', '002422.XSHE', '6
00060.XSHG', '601179.XSHG', '600383.XSHG', '002146.XSHE', '60168
8.XSHG', '600674.XSHG', '000800.XSHE', '600170.XSHG', '600048.XS
HG', '002653.XSHE', '600600.XSHG', '600690.XSHG', '600340.XSHG',
 '600315.XSHG', '300146.XSHE', '000024.XSHE', '300024.XSHE', '60
1788.XSHG', '600166.XSHG', '601633.XSHG', '600252.XSHG', '300133
.XSHE', '002475.XSHE', '600256.XSHG', '000883.XSHE', '600485.XSH
G', '300124.XSHE', '600157.XSHG', '600549.XSHG', '000400.XSHE',
'000625.XSHE', '300104.XSHE', '002241.XSHE', '600038.XSHG', '002
236.XSHE', '600415.XSHG', '002410.XSHE', '000826.XSHE', '002051.
XSHE', '600111.XSHG', '002456.XSHE', '600703.XSHG', '600369.XSHG
', '002375.XSHE', '002081.XSHE', '002673.XSHE', '300070.XSHE', '
000970.XSHE', '600332.XSHG']
2012-06-14 00:00:00 ['601186.XSHG', '000423.XSHE', '002475.XSHE'
, '002001.XSHE', '300027.XSHE', '600703.XSHG', '600547.XSHG', '6
00157.XSHG', '002456.XSHE', '300058.XSHE', '601991.XSHG', '00205
1.XSHE', '600519.XSHG', '601688.XSHG', '000027.XSHE', '600221.XS
HG', '600085.XSHG', '600578.XSHG', '600549.XSHG', '000826.XSHE',
 '600332.XSHG', '000725.XSHE', '600600.XSHG', '600315.XSHG', '00
0999.XSHE', '002038.XSHE', '600276.XSHG', '000538.XSHE', '002146
.XSHE', '000157.XSHE', '300070.XSHE', '002241.XSHE', '601633.XSH
G', '300146.XSHE', '002422.XSHE', '600048.XSHG', '000970.XSHE',
'000539.XSHE', '600535.XSHG', '000750.XSHE', '600383.XSHG', '600
011.XSHG', '002415.XSHE', '300002.XSHE', '002236.XSHE', '000625.
XSHE', '000024.XSHE', '000069.XSHE', '600340.XSHG', '600027.XSHG
', '002410.XSHE', '300104.XSHE', '002310.XSHE', '600369.XSHG', '
002375.XSHE', '002081.XSHE', '002653.XSHE']
2012-06-29 00:00:00 ['601633.XSHG', '600827.XSHG', '000712.XSHE'
, '600863.XSHG', '600048.XSHG', '601668.XSHG', '002422.XSHE', '0
00069.XSHE', '601288.XSHG', '601601.XSHG', '002065.XSHE', '00075
0.XSHE', '002051.XSHE', '300015.XSHE', '300017.XSHE', '601628.XS
HG', '002294.XSHE', '000027.XSHE', '601888.XSHG', '002456.XSHE',
 '300024.XSHE', '300104.XSHE', '000963.XSHE', '000568.XSHE', '00
2081.XSHE', '000895.XSHE', '002385.XSHE', '600642.XSHG', '600315
.XSHG', '601318.XSHG', '600886.XSHG', '300058.XSHE', '600998.XSH
G', '002410.XSHE', '000792.XSHE', '300027.XSHE', '600276.XSHG',
'002353.XSHE', '600795.XSHG', '600085.XSHG', '600578.XSHG', '601
991.XSHG', '000999.XSHE', '002038.XSHE', '300146.XSHE', '600535.
XSHG', '000538.XSHE', '002475.XSHE', '002415.XSHE', '002310.XSHE
', '002241.XSHE', '000539.XSHE', '600011.XSHG', '600518.XSHG', '
600027.XSHG', '002653.XSHE']
2012-07-13 00:00:00 ['002142.XSHE', '300133.XSHE', '000598.XSHE'
, '000895.XSHE', '600809.XSHG', '600029.XSHG', '300027.XSHE', '0
00002.XSHE', '002415.XSHE', '601111.XSHG', '300146.XSHE', '60051
9.XSHG', '600340.XSHG', '600048.XSHG', '600535.XSHG', '002007.XS
HE', '601888.XSHG', '601929.XSHG', '000712.XSHE', '000069.XSHE',
 '000538.XSHE', '000970.XSHE', '002310.XSHE', '300017.XSHE', '60
0583.XSHG', '002304.XSHE', '002081.XSHE', '002294.XSHE', '600011
.XSHG', '002450.XSHE', '002375.XSHE', '002038.XSHE', '600867.XSH
G', '601607.XSHG', '000539.XSHE', '000963.XSHE', '600196.XSHG',
'600115.XSHG', '002456.XSHE', '000568.XSHE', '002241.XSHE', '002
236.XSHE', '600315.XSHG', '300058.XSHE', '002051.XSHE', '002292.
XSHE', '002353.XSHE', '000792.XSHE', '000858.XSHE', '600085.XSHG
', '002475.XSHE', '600518.XSHG', '600027.XSHG', '600252.XSHG', '
002385.XSHE', '002653.XSHE', '600332.XSHG']
2012-07-27 00:00:00 ['002292.XSHE', '601117.XSHG', '300017.XSHE'
, '000728.XSHE', '000712.XSHE', '601888.XSHG', '600177.XSHG', '6
00637.XSHG', '601601.XSHG', '600085.XSHG', '002456.XSHE', '60180
8.XSHG', '300070.XSHE', '002007.XSHE', '000651.XSHE', '600276.XS
HG', '300024.XSHE', '600373.XSHG', '002465.XSHE', '600804.XSHG',
 '601928.XSHG', '600535.XSHG', '000792.XSHE', '600010.XSHG', '00
2385.XSHE', '002415.XSHE', '600867.XSHG', '000876.XSHE', '000970
.XSHE', '002375.XSHE', '601098.XSHG', '300104.XSHE', '600011.XSH
G', '600108.XSHG', '600519.XSHG', '601628.XSHG', '601607.XSHG',
'002304.XSHE', '000625.XSHE', '002038.XSHE', '601186.XSHG', '002
051.XSHE', '000858.XSHE', '000963.XSHE', '600196.XSHG', '002475.
XSHE', '600633.XSHG', '002570.XSHE', '000883.XSHE', '002470.XSHE
', '300133.XSHE', '600332.XSHG', '300058.XSHE', '600315.XSHG', '
002450.XSHE', '600252.XSHG', '002236.XSHE']
2012-08-10 00:00:00 ['601989.XSHG', '601998.XSHG', '000876.XSHE'
, '601231.XSHG', '601699.XSHG', '000783.XSHE', '601992.XSHG', '6
00489.XSHG', '002310.XSHE', '000400.XSHE', '000061.XSHE', '60118
6.XSHG', '000728.XSHE', '600089.XSHG', '600050.XSHG', '600406.XS
HG', '600271.XSHG', '002038.XSHE', '002475.XSHE', '601899.XSHG',
 '002241.XSHE', '600277.XSHG', '600549.XSHG', '000963.XSHE', '00
```

```
2470.XSHE', '002410.XSHE', '601258.XSHG', '600998.XSHG', '600741
.XSHG', '002051.XSHE', '000826.XSHE', '600516.XSHG', '300070.XSH
E', '000559.XSHE', '600547.XSHG', '000999.XSHE', '600867.XSHG',
'600739.XSHG', '300002.XSHE', '600535.XSHG', '002415.XSHE', '000
686.XSHE', '300024.XSHE', '300027.XSHE', '300251.XSHE', '600372.
XSHG', '300017.XSHE', '300133.XSHE', '002456.XSHE', '000503.XSHE
', '002236.XSHE', '601117.XSHG', '600703.XSHG', '600010.XSHG', '
002450.XSHE', '600637.XSHG', '000883.XSHE']
2012-08-24 00:00:00 ['000793.XSHE', '600518.XSHG', '600588.XSHG'
, '600277.XSHG', '601933.XSHG', '002422.XSHE', '600739.XSHG', '0
02051.XSHE', '000559.XSHE', '600489.XSHG', '600170.XSHG', '60008
5.XSHG', '000061.XSHE', '600703.XSHG', '300027.XSHE', '600332.XS
HG', '600373.XSHG', '601607.XSHG', '002310.XSHE', '000581.XSHE',
 '600674.XSHG', '600547.XSHG', '000963.XSHE', '300133.XSHE', '00
0629.XSHE', '002470.XSHE', '000009.XSHE', '600998.XSHG', '002475
.XSHE', '600867.XSHG', '600315.XSHG', '002456.XSHE', '600516.XSH
G', '000999.XSHE', '000826.XSHE', '600809.XSHG', '600535.XSHG',
'002241.XSHE', '300251.XSHE', '002065.XSHE', '300070.XSHE', '300
002.XSHE', '000400.XSHE', '002236.XSHE', '002570.XSHE', '002653.
XSHE', '002038.XSHE', '600372.XSHG', '600010.XSHG', '002450.XSHE
', '600549.XSHG', '600637.XSHG', '000503.XSHE', '000883.XSHE', '
300017.XSHE', '002230.XSHE', '601231.XSHG']
2012-09-07 00:00:00 ['600276.XSHG', '000725.XSHE', '600109.XSHG'
, '600688.XSHG', '300027.XSHE', '000999.XSHE', '000651.XSHE', '6
00690.XSHG', '600998.XSHG', '600597.XSHG', '600739.XSHG', '60089
3.XSHG', '002470.XSHE', '300251.XSHE', '601929.XSHG', '000826.XS
HE', '600633.XSHG', '600108.XSHG', '002236.XSHE', '601933.XSHG',
 '601928.XSHG', '600208.XSHG', '600050.XSHG', '002594.XSHE', '00
2456.XSHE', '002024.XSHE', '002450.XSHE', '600489.XSHG', '000503
.XSHE', '300070.XSHE', '300124.XSHE', '600373.XSHG', '600066.XSH
G', '002065.XSHE', '002465.XSHE', '600019.XSHG', '300024.XSHE',
'601607.XSHG', '600372.XSHG', '600887.XSHG', '600867.XSHG', '300
017.XSHE', '000793.XSHE', '002570.XSHE', '000413.XSHE', '600547.
XSHG', '600703.XSHG', '600316.XSHG', '600038.XSHG', '000738.XSHE
', '603000.XSHG', '600516.XSHG', '601231.XSHG', '002230.XSHE', '
000400.XSHE', '002008.XSHE', '600705.XSHG']
2012-09-21 00:00:00 ['601186.XSHG', '002353.XSHE', '000999.XSHE'
, '000651.XSHE', '000793.XSHE', '000728.XSHE', '601788.XSHG', '6
01989.XSHG', '300024.XSHE', '000960.XSHE', '000917.XSHE', '60051
8.XSHG', '002236.XSHE', '000783.XSHE', '601899.XSHG', '601688.XS
HG', '600867.XSHG', '601628.XSHG', '600362.XSHG', '000630.XSHE',
 '600150.XSHG', '600585.XSHG', '300070.XSHE', '000503.XSHE', '60
1555.XSHG', '300017.XSHE', '600406.XSHG', '002230.XSHE', '002594
.XSHE', '600690.XSHG', '600352.XSHG', '600804.XSHG', '600703.XSH
G', '600893.XSHG', '002385.XSHE', '000738.XSHE', '000400.XSHE',
'601117.XSHG', '600104.XSHG', '600038.XSHG', '600256.XSHG', '600
118.XSHG', '603000.XSHG', '601633.XSHG', '002024.XSHE', '002422.
XSHE', '002465.XSHE', '000725.XSHE', '600019.XSHG', '600547.XSHG
', '002294.XSHE', '600489.XSHG', '600060.XSHG', '600109.XSHG', '
600170.XSHG', '002008.XSHE', '600705.XSHG']
2012-10-12 00:00:00 ['600535.XSHG', '600089.XSHG', '600690.XSHG'
, '000960.XSHE', '600118.XSHG', '000568.XSHE', '601699.XSHG', '6
01788.XSHG', '601117.XSHG', '002008.XSHE', '000060.XSHE', '00223
```

```
0.XSHE', '000768.XSHE', '600019.XSHG', '002450.XSHE', '000100.XS
HE', '600837.XSHG', '600570.XSHG', '000538.XSHE', '600352.XSHG',
 '601186.XSHG', '000503.XSHE', '000963.XSHE', '601555.XSHG', '00
0895.XSHE', '600809.XSHG', '002344.XSHE', '600309.XSHG', '000630
.XSHE', '000413.XSHE', '600104.XSHG', '601929.XSHG', '600585.XSH
G', '000783.XSHE', '002038.XSHE', '600369.XSHG', '000063.XSHE',
'002385.XSHE', '000725.XSHE', '002422.XSHE', '002456.XSHE', '600
741.XSHG', '600547.XSHG', '600060.XSHG', '002399.XSHE', '601377.
XSHG', '300058.XSHE', '000625.XSHE', '000423.XSHE', '600170.XSHG
', '600489.XSHG', '600109.XSHG', '601633.XSHG', '000776.XSHE', '
600256.XSHG', '002294.XSHE', '603993.XSHG']
2012-10-26 00:00:00 ['601390.XSHG', '600827.XSHG', '002252.XSHE'
, '600166.XSHG', '601766.XSHG', '002304.XSHE', '601727.XSHG', '0
02236.XSHE', '601888.XSHG', '603000.XSHG', '300058.XSHE', '60001
6.XSHG', '601186.XSHG', '002422.XSHE', '601800.XSHG', '600886.XS
HG', '000686.XSHE', '000157.XSHE', '600369.XSHG', '600089.XSHG',
 '600585.XSHG', '000729.XSHE', '300251.XSHE', '600118.XSHG', '00
0423.XSHE', '600315.XSHG', '601992.XSHG', '600383.XSHG', '600157
.XSHG', '600109.XSHG', '002415.XSHE', '000069.XSHE', '601866.XSH
G', '600048.XSHG', '300124.XSHE', '601258.XSHG', '601669.XSHG',
'601929.XSHG', '000581.XSHE', '002375.XSHE', '000725.XSHE', '600
705.XSHG', '000539.XSHE', '000338.XSHE', '300133.XSHE', '002024.
XSHE', '002450.XSHE', '600010.XSHG', '601377.XSHG', '600340.XSHG
', '600256.XSHG', '002146.XSHE', '000024.XSHE', '600809.XSHG', '
002456.XSHE', '002294.XSHE', '603993.XSHG', '000156.XSHE']
2012-11-09 00:00:00 ['601169.XSHG', '000793.XSHE', '601866.XSHG'
, '300024.XSHE', '002594.XSHE', '601166.XSHG', '601933.XSHG', '6
01669.XSHG', '600104.XSHG', '000598.XSHE', '600010.XSHG', '60128
8.XSHG', '601988.XSHG', '000738.XSHE', '000002.XSHE', '000895.XS
HE', '600028.XSHG', '600597.XSHG', '601939.XSHG', '601633.XSHG',
 '600015.XSHG', '002252.XSHE', '600518.XSHG', '300070.XSHE', '60
0221.XSHG', '002415.XSHE', '600383.XSHG', '600887.XSHG', '601186
.XSHG', '601390.XSHG', '601800.XSHG', '000538.XSHE', '000100.XSH
E', '601766.XSHG', '000539.XSHE', '000069.XSHE', '600016.XSHG',
'600085.XSHG', '600048.XSHG', '600674.XSHG', '600578.XSHG', '600
809.XSHG', '000651.XSHE', '000581.XSHE', '300133.XSHE', '600011.
XSHG', '002146.XSHE', '002422.XSHE', '600886.XSHG', '000503.XSHE
', '300124.XSHE', '000725.XSHE', '000046.XSHE', '600705.XSHG', '
600315.XSHG', '600340.XSHG', '002294.XSHE', '000156.XSHE']
2012-11-23 00:00:00 ['601009.XSHG', '600089.XSHG', '600153.XSHG'
, '600018.XSHG', '600660.XSHG', '601919.XSHG', '601998.XSHG', '3
00133.XSHE', '300124.XSHE', '600015.XSHG', '000001.XSHE', '60059
7.XSHG', '601166.XSHG', '300027.XSHE', '600585.XSHG', '601988.XS
HG', '002051.XSHE', '000002.XSHE', '600038.XSHG', '600741.XSHG',
 '601006.XSHG', '600016.XSHG', '002142.XSHE', '600008.XSHG', '00
0024.XSHE', '000069.XSHE', '600383.XSHG', '601992.XSHG', '600104
.XSHG', '601633.XSHG', '601288.XSHG', '601117.XSHG', '002353.XSH
E', '000725.XSHE', '000538.XSHE', '000046.XSHE', '601169.XSHG',
'000598.XSHE', '600340.XSHG', '600048.XSHG', '000651.XSHE', '600
674.XSHG', '000581.XSHE', '600398.XSHG', '000100.XSHE', '000338.
XSHE', '000625.XSHE', '600109.XSHG', '601390.XSHG', '600886.XSHG
', '601766.XSHG', '601186.XSHG', '002594.XSHE', '600108.XSHG', '
002146.XSHE', '000503.XSHE', '600111.XSHG', '600277.XSHG']
```

```
2012-12-07 00:00:00 ['600497.XSHG', '600009.XSHG', '601628.XSHG'
, '601328.XSHG', '000027.XSHE', '000800.XSHE', '601111.XSHG', '6
00705.XSHG', '002236.XSHE', '600208.XSHG', '600150.XSHG', '60064
2.XSHG', '000069.XSHE', '600660.XSHG', '601601.XSHG', '601009.XS
HG', '000651.XSHE', '000581.XSHE', '600166.XSHG', '601006.XSHG',
 '600157.XSHG', '600108.XSHG', '601390.XSHG', '601919.XSHG', '60
0000.XSHG', '600674.XSHG', '000100.XSHE', '601998.XSHG', '601766
.XSHG', '002241.XSHE', '600741.XSHG', '601800.XSHG', '600585.XSH
G', '601668.XSHG', '600104.XSHG', '002081.XSHE', '600060.XSHG',
'600153.XSHG', '600016.XSHG', '601166.XSHG', '601992.XSHG', '600
340.XSHG', '601186.XSHG', '600048.XSHG', '600011.XSHG', '600383.
XSHG', '000712.XSHE', '600886.XSHG', '601169.XSHG', '000002.XSHE
', '600005.XSHG', '000024.XSHE', '600277.XSHG', '002146.XSHE', '
600027.XSHG', '000338.XSHE', '601117.XSHG']
2012-12-21 00:00:00 ['600030.XSHG', '601186.XSHG', '601117.XSHG'
, '600031.XSHG', '000157.XSHE', '600038.XSHG', '002594.XSHE', '0
00709.XSHE', '000776.XSHE', '000413.XSHE', '600029.XSHG', '60001
5.XSHG', '603993.XSHG', '601377.XSHG', '000338.XSHE', '600157.XS
HG', '600585.XSHG', '600111.XSHG', '601318.XSHG', '000001.XSHE',
 '600783.XSHG', '600060.XSHG', '000024.XSHE', '600340.XSHG', '60
0690.XSHG', '600999.XSHG', '600383.XSHG', '601669.XSHG', '600166
.XSHG', '600208.XSHG', '000970.XSHE', '002081.XSHE', '601668.XSH
G', '601800.XSHG', '000629.XSHE', '601788.XSHG', '600153.XSHG',
'601601.XSHG', '601009.XSHG', '601633.XSHG', '600036.XSHG', '600
000.XSHG', '000937.XSHE', '600104.XSHG', '300251.XSHE', '000400.
XSHE', '600549.XSHG', '601166.XSHG', '601111.XSHG', '600016.XSHG
', '000800.XSHE', '601169.XSHG', '601699.XSHG', '000061.XSHE', '
601336.XSHG', '601992.XSHG', '000750.XSHE']
2013-01-09 00:00:00 ['601788.XSHG', '600827.XSHG', '002252.XSHE'
, '600060.XSHG', '600208.XSHG', '000983.XSHE', '600663.XSHG', '0
00629.XSHE', '300002.XSHE', '600016.XSHG', '600863.XSHG', '30002
7.XSHE', '600633.XSHG', '000800.XSHE', '601699.XSHG', '603993.XS
HG', '600340.XSHG', '600809.XSHG', '600383.XSHG', '601166.XSHG',
 '000999.XSHE', '601633.XSHG', '002500.XSHE', '002007.XSHE', '00
2146.XSHE', '000768.XSHE', '600153.XSHG', '600030.XSHG', '000024
.XSHE', '600000.XSHG', '002594.XSHE', '002292.XSHE', '000039.XSH
E', '000970.XSHE', '002673.XSHE', '600109.XSHG', '600873.XSHG',
'600036.XSHG', '002653.XSHE', '601336.XSHG', '000937.XSHE', '002
570.XSHE', '000623.XSHE', '600332.XSHG', '600783.XSHG', '600118.
XSHG', '600705.XSHG', '000046.XSHE', '600893.XSHG', '600648.XSHG
', '600372.XSHG', '002465.XSHE', '600316.XSHG', '000400.XSHE', '
000750.XSHE', '000156.XSHE', '300251.XSHE']
2013-01-23 00:00:00 ['600036.XSHG', '600663.XSHG', '000423.XSHE'
, '600153.XSHG', '000623.XSHE', '002456.XSHE', '000503.XSHE', '6
00252.XSHG', '601166.XSHG', '600208.XSHG', '000046.XSHE', '00089
5.XSHE', '000876.XSHE', '300017.XSHE', '000063.XSHE', '002450.XS
HE', '600066.XSHG', '000999.XSHE', '600016.XSHG', '002570.XSHE',
 '002653.XSHE', '300146.XSHE', '601633.XSHG', '600873.XSHG', '30
0104.XSHE', '600157.XSHG', '300024.XSHE', '000400.XSHE', '600109
.XSHG', '601901.XSHG', '603993.XSHG', '600089.XSHG', '002594.XSH
E', '000002.XSHE', '600038.XSHG', '600867.XSHG', '600570.XSHG',
'000937.XSHE', '300251.XSHE', '002294.XSHE', '002292.XSHE', '000
039.XSHE', '002038.XSHE', '600887.XSHG', '002008.XSHE', '002146.
```

```
XSHE', '600118.XSHG', '000738.XSHE', '600372.XSHG', '000768.XSHE
', '300058.XSHE', '002353.XSHE', '000001.XSHE', '600316.XSHG', '
600648.XSHG', '000156.XSHE', '600705.XSHG']
2013-02-06 00:00:00 ['600516.XSHG', '000895.XSHE', '600348.XSHG'
, '601788.XSHG', '600383.XSHG', '600030.XSHG', '000425.XSHE', '0
02142.XSHE', '600369.XSHG', '600038.XSHG', '002344.XSHE', '60011
8.XSHG', '600633.XSHG', '600000.XSHG', '002294.XSHE', '600497.XS
HG', '300058.XSHE', '600015.XSHG', '000960.XSHE', '002500.XSHE',
 '603993.XSHG', '002038.XSHE', '300024.XSHE', '000559.XSHE', '60
1998.XSHG', '601099.XSHG', '601633.XSHG', '601818.XSHG', '000738
.XSHE', '600887.XSHG', '601166.XSHG', '600316.XSHG', '000002.XSH
E', '002353.XSHE', '000937.XSHE', '000783.XSHE', '600315.XSHG',
'300027.XSHE', '000623.XSHE', '600089.XSHG', '600837.XSHG', '002
570.XSHE', '002008.XSHE', '600867.XSHG', '601688.XSHG', '000060.
XSHE', '600031.XSHG', '600705.XSHG', '600157.XSHG', '000768.XSHE
', '000625.XSHE', '000581.XSHE', '600352.XSHG', '600999.XSHG', '
000001.XSHE', '601901.XSHG', '600372.XSHG', '600016.XSHG']
2013-02-27 00:00:00 ['601216.XSHG', '600886.XSHG', '000738.XSHE'
, '600372.XSHG', '601158.XSHG', '600373.XSHG', '601098.XSHG', '0
02470.XSHE', '000060.XSHE', '600535.XSHG', '002294.XSHE', '00099
9.XSHE', '300104.XSHE', '600867.XSHG', '002236.XSHE', '600027.XS
HG', '002353.XSHE', '002385.XSHE', '600837.XSHG', '002230.XSHE',
 '601607.XSHG', '002653.XSHE', '601929.XSHG', '300124.XSHE', '00
0625.XSHE', '601688.XSHG', '000623.XSHE', '300058.XSHE', '601258
.XSHG', '002570.XSHE', '300015.XSHE', '002410.XSHE', '601901.XSH
G', '002065.XSHE', '002475.XSHE', '300133.XSHE', '002038.XSHE',
'300070.XSHE', '002310.XSHE', '600633.XSHG', '600518.XSHG', '002
252.XSHE', '000826.XSHE', '600332.XSHG', '600277.XSHG', '300017.
XSHE', '000581.XSHE', '600008.XSHG', '600999.XSHG', '600038.XSHG
', '300027.XSHE', '600352.XSHG', '300059.XSHE', '000503.XSHE', '
600597.XSHG', '002344.XSHE', '000831.XSHE']
2013-03-13 00:00:00 ['600804.XSHG', '002008.XSHE', '300002.XSHE'
, '002344.XSHE', '601117.XSHG', '600516.XSHG', '600196.XSHG', '6
00027.XSHG', '600795.XSHG', '601231.XSHG', '600600.XSHG', '60193
3.XSHG', '601216.XSHG', '600028.XSHG', '300003.XSHE', '600867.XS
HG', '002230.XSHE', '300133.XSHE', '601258.XSHG', '000100.XSHE',
 '601633.XSHG', '300251.XSHE', '000729.XSHE', '002252.XSHE', '60
1901.XSHG', '002475.XSHE', '002450.XSHE', '600703.XSHG', '002065
.XSHE', '000826.XSHE', '600597.XSHG', '000063.XSHE', '600688.XSH
G', '002241.XSHE', '600038.XSHG', '300070.XSHE', '600535.XSHG',
'601158.XSHG', '300015.XSHE', '600256.XSHG', '000738.XSHE', '002
415.XSHE', '600809.XSHG', '000598.XSHE', '300017.XSHE', '000503.
XSHE', '002038.XSHE', '002410.XSHE', '002236.XSHE', '000793.XSHE
', '300104.XSHE', '002456.XSHE', '600648.XSHG', '300059.XSHE', '
600277.XSHG', '600008.XSHG', '000831.XSHE']
2013-03-27 00:00:00 ['002008.XSHE', '600583.XSHG', '600027.XSHG'
, '600795.XSHG', '002065.XSHE', '600535.XSHG', '000792.XSHE', '0
02230.XSHE', '000963.XSHE', '600516.XSHG', '300070.XSHE', '60115
8.XSHG', '601888.XSHG', '000686.XSHE', '600648.XSHG', '600221.XS
HG', '600804.XSHG', '600315.XSHG', '600873.XSHG', '600256.XSHG',
 '000423.XSHE', '000538.XSHE', '600060.XSHG', '601998.XSHG', '60
0588.XSHG', '601117.XSHG', '000895.XSHE', '601633.XSHG', '601231
.XSHG', '600352.XSHG', '000831.XSHE', '600277.XSHG', '300104.XSH
```

```
E', '002038.XSHE', '600252.XSHG', '002450.XSHE', '000793.XSHE',
'600867.XSHG', '002353.XSHE', '300059.XSHE', '000009.XSHE', '600
597.XSHG', '600332.XSHG', '002410.XSHE', '601901.XSHG', '002415.
XSHE', '000063.XSHE', '600887.XSHG', '300017.XSHE', '002236.XSHE
', '002241.XSHE', '300251.XSHE', '002570.XSHE', '000712.XSHE', '
002456.XSHE', '000598.XSHE', '600008.XSHG']
2013-04-12 00:00:00 ['600887.XSHG', '000831.XSHE', '600340.XSHG'
, '000024.XSHE', '600085.XSHG', '600690.XSHG', '601118.XSHG', '0
00895.XSHE', '600585.XSHG', '600118.XSHG', '000100.XSHE', '00053
8.XSHE', '000826.XSHE', '600518.XSHG', '600372.XSHG', '601800.XS
HG', '600153.XSHG', '300104.XSHE', '002653.XSHE', '600066.XSHG',
 '600315.XSHG', '601117.XSHG', '600383.XSHG', '600674.XSHG', '60
0008.XSHG', '002230.XSHE', '300146.XSHE', '300070.XSHE', '600804
.XSHG', '002375.XSHE', '300251.XSHE', '002241.XSHE', '600316.XSH
G', '002236.XSHE', '600309.XSHG', '002146.XSHE', '300133.XSHE',
'600637.XSHG', '600252.XSHG', '600221.XSHG', '600597.XSHG', '601
231.XSHG', '300017.XSHE', '000625.XSHE', '002310.XSHE', '600783.
XSHG', '600332.XSHG', '600352.XSHG', '300002.XSHE', '600705.XSHG
', '002008.XSHE', '002570.XSHE', '600170.XSHG', '600060.XSHG', '
002353.XSHE', '000598.XSHE', '000712.XSHE', '002456.XSHE']
2013-04-26 00:00:00 ['600383.XSHG', '300070.XSHE', '000793.XSHE'
, '300059.XSHE', '601336.XSHG', '000598.XSHE', '600703.XSHG', '0
02344.XSHE', '002008.XSHE', '600309.XSHG', '600535.XSHG', '00223
0.XSHE', '600600.XSHG', '300146.XSHE', '002051.XSHE', '000338.XS
HE', '600570.XSHG', '000503.XSHE', '002570.XSHE', '600839.XSHG',
 '600519.XSHG', '000783.XSHE', '300002.XSHE', '600352.XSHG', '60
0221.XSHG', '002353.XSHE', '000046.XSHE', '000400.XSHE', '002375
.XSHE', '000826.XSHE', '603000.XSHG', '002294.XSHE', '002653.XSH
E', '002292.XSHE', '601633.XSHG', '000100.XSHE', '002450.XSHE',
'600648.XSHG', '002081.XSHE', '601258.XSHG', '000917.XSHE', '002
236.XSHE', '000625.XSHE', '600332.XSHG', '300027.XSHE', '002310.
XSHE', '300017.XSHE', '300133.XSHE', '600060.XSHG', '600637.XSHG
', '300124.XSHE', '300104.XSHE', '600804.XSHG', '600406.XSHG', '
300058.XSHE', '000800.XSHE', '300251.XSHE', '002456.XSHE']
2013-05-15 00:00:00 ['601901.XSHG', '600108.XSHG', '600519.XSHG'
, '300070.XSHE', '600085.XSHG', '002038.XSHE', '600485.XSHG', '0
00598.XSHE', '000826.XSHE', '300133.XSHE', '002410.XSHE', '00267
3.XSHE', '002129.XSHE', '000970.XSHE', '000009.XSHE', '300015.XS
HE', '600783.XSHG', '000686.XSHE', '002236.XSHE', '600867.XSHG',
 '601928.XSHG', '600637.XSHG', '002344.XSHE', '300003.XSHE', '00
0559.XSHE', '000917.XSHE', '300058.XSHE', '600373.XSHG', '000063
.XSHE', '600535.XSHG', '600066.XSHG', '002294.XSHE', '002450.XSH
E', '000503.XSHE', '002065.XSHE', '300017.XSHE', '600406.XSHG',
'600570.XSHG', '002241.XSHE', '600648.XSHG', '600703.XSHG', '300
124.XSHE', '603000.XSHG', '300024.XSHE', '600332.XSHG', '002456.
XSHE', '601633.XSHG', '002653.XSHE', '601231.XSHG', '002292.XSHE
', '000400.XSHE', '300059.XSHE', '300104.XSHE', '300251.XSHE', '
300027.XSHE', '000800.XSHE', '600804.XSHG', '002594.XSHE']
2013-05-29 00:00:00 ['603000.XSHG', '002465.XSHE', '600369.XSHG'
, '000768.XSHE', '600588.XSHG', '000876.XSHE', '000839.XSHE', '6
00066.XSHG', '002081.XSHE', '000024.XSHE', '000970.XSHE', '00091
7.XSHE', '000800.XSHE', '300058.XSHE', '600271.XSHG', '600739.XS
HG', '002450.XSHE', '002146.XSHE', '300070.XSHE', '600118.XSHG',
 '300003.XSHE', '600705.XSHG', '000156.XSHE', '002202.XSHE', '60
0674.XSHG', '601258.XSHG', '600372.XSHG', '002292.XSHE', '600316
.XSHG', '600485.XSHG', '002153.XSHE', '002475.XSHE', '300059.XSH
E', '601231.XSHG', '600637.XSHG', '600867.XSHG', '600406.XSHG',
'000559.XSHE', '000009.XSHE', '300104.XSHE', '601928.XSHG', '601
117.XSHG', '600340.XSHG', '600583.XSHG', '000712.XSHE', '002241.
XSHE', '300027.XSHE', '600783.XSHG', '002375.XSHE', '000400.XSHE
', '600089.XSHG', '002594.XSHE', '000061.XSHE', '600703.XSHG', '
002129.XSHE', '002653.XSHE', '600804.XSHG']
2013-06-17 00:00:00 ['000963.XSHE', '600600.XSHG', '600739.XSHG'
, '600718.XSHG', '002292.XSHE', '000800.XSHE', '600518.XSHG', '0
00725.XSHE', '000917.XSHE', '600256.XSHG', '600867.XSHG', '60011
8.XSHG', '000826.XSHE', '300015.XSHE', '002146.XSHE', '600583.XS
HG', '600633.XSHG', '600157.XSHG', '600406.XSHG', '002241.XSHE',
 '002051.XSHE', '601928.XSHG', '002008.XSHE', '600369.XSHG', '00
0046.XSHE', '600998.XSHG', '000156.XSHE', '601117.XSHG', '002353
.XSHE', '300251.XSHE', '600252.XSHG', '300070.XSHE', '600089.XSH
G', '601098.XSHG', '000559.XSHE', '300058.XSHE', '002081.XSHE',
'600674.XSHG', '000839.XSHE', '000061.XSHE', '601258.XSHG', '600
340.XSHG', '000009.XSHE', '600637.XSHG', '000712.XSHE', '002653.
XSHE', '000413.XSHE', '002230.XSHE', '600649.XSHG', '600703.XSHG
', '000793.XSHE', '600783.XSHG', '300104.XSHE', '002129.XSHE', '
002375.XSHE', '300002.XSHE']
2013-07-01 00:00:00 ['000729.XSHE', '601818.XSHG', '600256.XSHG'
, '002410.XSHE', '600340.XSHG', '600570.XSHG', '000538.XSHE', '0
02375.XSHE', '002353.XSHE', '600535.XSHG', '000413.XSHE', '60027
6.XSHG', '300058.XSHE', '300024.XSHE', '600600.XSHG', '601288.XS
HG', '600703.XSHG', '002450.XSHE', '300070.XSHE', '300015.XSHE',
 '002038.XSHE', '000839.XSHE', '601988.XSHG', '601098.XSHG', '60
1336.XSHG', '000895.XSHE', '002570.XSHE', '600519.XSHG', '600998
.XSHG', '600649.XSHG', '002241.XSHE', '600066.XSHG', '002399.XSH
E', '600547.XSHG', '600867.XSHG', '000963.XSHE', '300251.XSHE',
'600518.XSHG', '601398.XSHG', '000063.XSHE', '002129.XSHE', '600
315.XSHG', '600373.XSHG', '601991.XSHG', '300017.XSHE', '300124.
XSHE', '000793.XSHE', '002230.XSHE', '600887.XSHG', '300104.XSHE
', '601216.XSHG', '600637.XSHG', '300002.XSHE', '600893.XSHG', '
002292.XSHE', '600633.XSHG', '300059.XSHE']
2013-07-15 00:00:00 ['002410.XSHE', '600900.XSHG', '600011.XSHG'
, '002236.XSHE', '601991.XSHG', '000970.XSHE', '601888.XSHG', '0
02038.XSHE', '002081.XSHE', '601933.XSHG', '601098.XSHG', '60060
0.XSHG', '600718.XSHG', '600309.XSHG', '002594.XSHE', '601928.XS
HG', '000793.XSHE', '600352.XSHG', '000750.XSHE', '600886.XSHG',
 '000400.XSHE', '002465.XSHE', '000651.XSHE', '002456.XSHE', '60
0315.XSHG', '002252.XSHE', '601633.XSHG', '600518.XSHG', '002230
.XSHE', '600637.XSHG', '600383.XSHG', '600535.XSHG', '000024.XSH
E', '300146.XSHE', '002146.XSHE', '300024.XSHE', '002153.XSHE',
'000538.XSHE', '300104.XSHE', '600570.XSHG', '600804.XSHG', '603
000.XSHG', '600867.XSHG', '600373.XSHG', '300124.XSHE', '600893.
XSHG', '600340.XSHG', '002570.XSHE', '600597.XSHG', '300017.XSHE
', '000831.XSHE', '600887.XSHG', '600633.XSHG', '002292.XSHE', '
601216.XSHG', '300059.XSHE']
2013-07-29 00:00:00 ['002653.XSHE', '000069.XSHE', '000623.XSHE'
, '600100.XSHG', '600839.XSHG', '601766.XSHG', '000898.XSHE', '6
```

```
01390.XSHG', '600585.XSHG', '000598.XSHE', '600485.XSHG', '60058
3.XSHG', '600535.XSHG', '600837.XSHG', '300070.XSHE', '600637.XS
HG', '600867.XSHG', '603000.XSHG', '000826.XSHE', '600252.XSHG',
 '002415.XSHE', '000917.XSHE', '601231.XSHG', '600271.XSHG', '00
2024.XSHE', '000750.XSHE', '600588.XSHG', '300024.XSHE', '600783
.XSHG', '601929.XSHG', '300251.XSHE', '601216.XSHG', '000538.XSH
E', '601928.XSHG', '000063.XSHE', '300059.XSHE', '601186.XSHG',
'300015.XSHE', '000413.XSHE', '002230.XSHE', '000503.XSHE', '002
007.XSHE', '600597.XSHG', '600633.XSHG', '600570.XSHG', '600060.
XSHG', '002410.XSHE', '300017.XSHE', '600718.XSHG', '600276.XSHG
', '000793.XSHE', '600804.XSHG', '300027.XSHE', '002065.XSHE', '
002153.XSHE', '000831.XSHE', '000156.XSHE']
2013-08-12 00:00:00 ['601766.XSHG', '000069.XSHE', '002465.XSHE'
, '601699.XSHG', '601186.XSHG', '600415.XSHG', '600008.XSHG', '0
00625.XSHE', '600600.XSHG', '600578.XSHG', '002008.XSHE', '00059
8.XSHE', '600674.XSHG', '002294.XSHE', '002470.XSHE', '600867.XS
HG', '000898.XSHE', '601888.XSHG', '002241.XSHE', '000651.XSHE',
 '600485.XSHG', '600703.XSHG', '600585.XSHG', '000712.XSHE', '30
0146.XSHE', '002153.XSHE', '600276.XSHG', '002475.XSHE', '600111
.XSHG', '002007.XSHE', '002653.XSHE', '600060.XSHG', '000063.XSH
E', '601179.XSHG', '002385.XSHE', '002450.XSHE', '000503.XSHE',
'601231.XSHG', '000831.XSHE', '600718.XSHG', '600315.XSHG', '002
065.XSHE', '600873.XSHG', '601928.XSHG', '600588.XSHG', '601633.
XSHG', '600570.XSHG', '600804.XSHG', '002001.XSHE', '300133.XSHE
', '600352.XSHG', '000793.XSHE', '000413.XSHE', '002024.XSHE', '
000156.XSHE', '300027.XSHE']
2013-08-26 00:00:00 ['601888.XSHG', '600827.XSHG', '600570.XSHG'
, '000063.XSHE', '000937.XSHE', '600016.XSHG', '600674.XSHG', '6
00804.XSHG', '600009.XSHG', '000001.XSHE', '600585.XSHG', '60031
5.XSHG', '002252.XSHE', '600100.XSHG', '601166.XSHG', '600256.XS
HG', '002153.XSHE', '000831.XSHE', '000983.XSHE', '601006.XSHG',
 '000559.XSHE', '600018.XSHG', '600663.XSHG', '600406.XSHG', '00
2001.XSHE', '600177.XSHG', '600157.XSHG', '300003.XSHE', '601699
.XSHG', '000876.XSHE', '600718.XSHG', '300017.XSHE', '601928.XSH
G', '000960.XSHE', '600873.XSHG', '000413.XSHE', '601808.XSHG',
'601633.XSHG', '000800.XSHE', '603000.XSHG', '300146.XSHE', '300
124.XSHE', '002570.XSHE', '002375.XSHE', '002241.XSHE', '600497.
XSHG', '600111.XSHG', '000625.XSHE', '600741.XSHG', '000400.XSHE
', '601231.XSHG', '600352.XSHG', '300133.XSHE', '600588.XSHG', '
002475.XSHE', '002024.XSHE']
2013-09-09 00:00:00 ['300058.XSHE', '601018.XSHG', '000156.XSHE'
, '600317.XSHG', '002570.XSHE', '600038.XSHG', '002024.XSHE', '6
01727.XSHG', '002252.XSHE', '002001.XSHE', '000559.XSHE', '00212
9.XSHE', '000400.XSHE', '002410.XSHE', '000917.XSHE', '600718.XS
HG', '600373.XSHG', '300017.XSHE', '600741.XSHG', '600827.XSHG',
 '600177.XSHG', '601919.XSHG', '600118.XSHG', '600000.XSHG', '60
1166.XSHG', '300059.XSHE', '002415.XSHE', '600115.XSHG', '600637
.XSHG', '600170.XSHG', '600089.XSHG', '000039.XSHE', '600570.XSH
G', '601006.XSHG', '300015.XSHE', '300124.XSHE', '601231.XSHG',
'600108.XSHG', '601928.XSHG', '300251.XSHE', '600588.XSHG', '600
316.XSHG', '300133.XSHE', '300003.XSHE', '601866.XSHG', '601333.
XSHG', '600009.XSHG', '600717.XSHG', '603000.XSHG', '601118.XSHG
', '600633.XSHG', '300027.XSHE', '300002.XSHE', '600398.XSHG', '
```

```
600648.XSHG', '600663.XSHG', '600018.XSHG']
2013-09-25 00:00:00 ['000001.XSHE', '600642.XSHG', '601600.XSHG'
, '600177.XSHG', '600875.XSHG', '000768.XSHE', '002129.XSHE', '3
00003.XSHE', '000917.XSHE', '601098.XSHG', '600660.XSHG', '00259
4.XSHE', '600150.XSHG', '300133.XSHE', '601933.XSHG', '300059.XS
HE', '600170.XSHG', '600597.XSHG', '601106.XSHG', '601727.XSHG',
 '000581.XSHE', '600108.XSHG', '000895.XSHE', '600089.XSHG', '00
2001.XSHE', '600705.XSHG', '002202.XSHE', '601919.XSHG', '300058
.XSHE', '600000.XSHG', '601333.XSHG', '600887.XSHG', '600717.XSH
G', '600649.XSHG', '601888.XSHG', '601607.XSHG', '600115.XSHG',
'300251.XSHE', '000039.XSHE', '002344.XSHE', '600783.XSHG', '600
633.XSHG', '300002.XSHE', '603000.XSHG', '600827.XSHG', '600415.
XSHG', '300027.XSHE', '600637.XSHG', '601989.XSHG', '601866.XSHG
', '601118.XSHG', '002024.XSHE', '600398.XSHG', '600663.XSHG', '
600018.XSHG', '600648.XSHG', '000333.XSHE']
2013-10-16 00:00:00 ['000581.XSHE', '601992.XSHG', '600352.XSHG'
, '600998.XSHG', '600588.XSHG', '600717.XSHG', '600256.XSHG', '3
00027.XSHE', '600887.XSHG', '600277.XSHG', '002410.XSHE', '60125
8.XSHG', '600873.XSHG', '600804.XSHG', '300058.XSHE', '000503.XS
HE', '000895.XSHE', '300015.XSHE', '300146.XSHE', '601933.XSHG',
 '600108.XSHG', '000876.XSHE', '000729.XSHE', '600597.XSHG', '60
0703.XSHG', '600177.XSHG', '600739.XSHG', '300124.XSHE', '300059
.XSHE', '002344.XSHE', '002129.XSHE', '300251.XSHE', '600783.XSH
G', '002385.XSHE', '600089.XSHG', '000917.XSHE', '600690.XSHG',
'300017.XSHE', '600415.XSHG', '601600.XSHG', '600637.XSHG', '600
649.XSHG', '600663.XSHG', '000061.XSHE', '600718.XSHG', '601607.
XSHG', '600827.XSHG', '300104.XSHE', '601989.XSHG', '600196.XSHG
', '002202.XSHE', '601118.XSHG', '002153.XSHE', '601216.XSHG', '
002024.XSHE', '600648.XSHG', '600485.XSHG', '000333.XSHE']
2013-10-30 00:00:00 ['000826.XSHE', '002129.XSHE', '600674.XSHG'
, '600027.XSHG', '002008.XSHE', '600108.XSHG', '600578.XSHG', '0
00539.XSHE', '600277.XSHG', '000423.XSHE', '000100.XSHE', '00235
3.XSHE', '600583.XSHG', '000625.XSHE', '600795.XSHG', '002470.XS
HE', '601166.XSHG', '002142.XSHE', '300015.XSHE', '600886.XSHG',
 '600177.XSHG', '000333.XSHE', '600741.XSHG', '601258.XSHG', '60
0863.XSHG', '600703.XSHG', '000778.XSHE', '002153.XSHE', '000729
.XSHE', '600717.XSHG', '300124.XSHE', '002410.XSHE', '600600.XSH
G', '000783.XSHE', '002294.XSHE', '601766.XSHG', '600873.XSHG',
'000061.XSHE', '600739.XSHG', '600089.XSHG', '600718.XSHG', '300
017.XSHE', '000413.XSHE', '002202.XSHE', '601992.XSHG', '601118.
XSHG', '000598.XSHE', '000001.XSHE', '600690.XSHG', '002385.XSHE
', '002292.XSHE', '600256.XSHG', '601808.XSHG', '300104.XSHE', '
600196.XSHG', '601216.XSHG', '600839.XSHG', '600485.XSHG']
2013-11-13 00:00:00 ['002456.XSHE', '600118.XSHG', '601988.XSHG'
, '600690.XSHG', '600519.XSHG', '002304.XSHE', '000712.XSHE', '0
02465.XSHE', '002038.XSHE', '601668.XSHG', '600674.XSHG', '60022
1.XSHG', '601800.XSHG', '601009.XSHG', '601166.XSHG', '601818.XS
HG', '600863.XSHG', '601939.XSHG', '000581.XSHE', '600000.XSHG',
 '000100.XSHE', '601390.XSHG', '601628.XSHG', '601288.XSHG', '60
0739.XSHG', '002142.XSHE', '601006.XSHG', '002450.XSHE', '601158
.XSHG', '000783.XSHE', '000333.XSHE', '600717.XSHG', '600795.XSH
G', '601318.XSHG', '002310.XSHE', '600196.XSHG', '600104.XSHG',
'000568.XSHE', '600309.XSHG', '600518.XSHG', '600015.XSHG', '002
```

```
008.XSHE', '002252.XSHE', '600809.XSHG', '601998.XSHG', '600398.
XSHG', '000970.XSHE', '002353.XSHE', '600028.XSHG', '000651.XSHE
', '601929.XSHG', '600886.XSHG', '600583.XSHG', '000001.XSHE', '
000598.XSHE', '601808.XSHG', '601766.XSHG', '600839.XSHG']
2013-11-27 00:00:00 ['600783.XSHG', '600028.XSHG', '600009.XSHG'
, '600519.XSHG', '002304.XSHE', '300024.XSHE', '601928.XSHG', '0
02456.XSHE', '000063.XSHE', '601633.XSHG', '002065.XSHE', '60192
9.XSHG', '600690.XSHG', '002475.XSHE', '002450.XSHE', '000876.XS
HE', '600369.XSHG', '000883.XSHE', '600717.XSHG', '002153.XSHE',
 '600588.XSHG', '600276.XSHG', '601231.XSHG', '000970.XSHE', '60
1318.XSHG', '601992.XSHG', '603000.XSHG', '600309.XSHG', '600585
.XSHG', '600485.XSHG', '601628.XSHG', '600827.XSHG', '002465.XSH
E', '002008.XSHE', '600649.XSHG', '000039.XSHE', '601607.XSHG',
'000768.XSHE', '600271.XSHG', '601601.XSHG', '600038.XSHG', '600
809.XSHG', '300003.XSHE', '002252.XSHE', '601117.XSHG', '600100.
XSHG', '600839.XSHG', '000503.XSHE', '300027.XSHE', '600893.XSHG
', '600150.XSHG', '002470.XSHE', '601018.XSHG', '000738.XSHE', '
600118.XSHG', '600109.XSHG', '600316.XSHG', '600372.XSHG']
2013-12-11 00:00:00 ['000651.XSHE', '600875.XSHG', '600497.XSHG'
, '601018.XSHG', '601118.XSHG', '600485.XSHG', '000623.XSHE', '0
00917.XSHE', '002500.XSHE', '600588.XSHG', '000783.XSHE', '00267
3.XSHE', '000581.XSHE', '600873.XSHG', '002375.XSHE', '601099.XS
HG', '600398.XSHG', '600741.XSHG', '600150.XSHG', '600703.XSHG',
 '002008.XSHE', '000728.XSHE', '601628.XSHG', '600585.XSHG', '60
1318.XSHG', '000776.XSHE', '600649.XSHG', '000039.XSHE', '601117
.XSHG', '002385.XSHE', '600309.XSHG', '000738.XSHE', '600837.XSH
G', '002081.XSHE', '600038.XSHG', '000793.XSHE', '600030.XSHG',
'601633.XSHG', '600018.XSHG', '601555.XSHG', '600100.XSHG', '600
118.XSHG', '600783.XSHG', '601607.XSHG', '000876.XSHE', '600839.
XSHG', '600369.XSHG', '600372.XSHG', '600893.XSHG', '002470.XSHE
', '000750.XSHE', '600690.XSHG', '600999.XSHG', '600827.XSHG', '
300027.XSHE', '600316.XSHG', '600109.XSHG', '002252.XSHE']
2013-12-25 00:00:00 ['601919.XSHG', '601888.XSHG', '600660.XSHG'
, '002653.XSHE', '600875.XSHG', '000917.XSHE', '600166.XSHG', '6
00497.XSHG', '002294.XSHE', '600100.XSHG', '600867.XSHG', '60099
8.XSHG', '600309.XSHG', '000400.XSHE', '000625.XSHE', '002470.XS
HE', '600352.XSHG', '600600.XSHG', '600597.XSHG', '000709.XSHE',
 '600315.XSHG', '600066.XSHG', '000729.XSHE', '002236.XSHE', '60
0150.XSHG', '002344.XSHE', '002422.XSHE', '600383.XSHG', '002375
.XSHE', '600741.XSHG', '300003.XSHE', '000895.XSHE', '600535.XSH
G', '000712.XSHE', '002081.XSHE', '002385.XSHE', '000651.XSHE',
'000876.XSHE', '300024.XSHE', '600588.XSHG', '600999.XSHG', '600
018.XSHG', '600406.XSHG', '600276.XSHG', '600196.XSHG', '600703.
XSHG', '000581.XSHE', '600085.XSHG', '000333.XSHE', '002007.XSHE
', '002038.XSHE', '000963.XSHE', '600690.XSHG', '600688.XSHG', '
300017.XSHE', '000883.XSHE', '002252.XSHE', '600023.XSHG']
2014-01-09 00:00:00 ['600588.XSHG', '002236.XSHE', '600600.XSHG'
, '002500.XSHE', '000413.XSHE', '601098.XSHG', '600804.XSHG', '6
01099.XSHG', '600372.XSHG', '600089.XSHG', '601933.XSHG', '00259
4.XSHE', '002294.XSHE', '000729.XSHE', '002456.XSHE', '600637.XS
HG', '600633.XSHG', '600315.XSHG', '300251.XSHE', '000039.XSHE',
 '300003.XSHE', '600406.XSHG', '600196.XSHG', '600150.XSHG', '00
0400.XSHE', '002353.XSHE', '002153.XSHE', '002008.XSHE', '603000
```

```
.XSHG', '600867.XSHG', '002065.XSHE', '002422.XSHE', '300133.XSH
E', '600109.XSHG', '600276.XSHG', '002465.XSHE', '300070.XSHE',
'600535.XSHG', '300124.XSHE', '002007.XSHE', '002038.XSHE', '300
015.XSHE', '002653.XSHE', '300024.XSHE', '300146.XSHE', '600010.
XSHG', '601231.XSHG', '600998.XSHG', '002410.XSHE', '000963.XSHE
', '300058.XSHE', '600703.XSHG', '300017.XSHE', '300002.XSHE', '
300104.XSHE', '002475.XSHE', '002292.XSHE', '600023.XSHG']
2014-01-23 00:00:00 ['600648.XSHG', '601318.XSHG', '000999.XSHE'
, '600893.XSHG', '000895.XSHE', '601929.XSHG', '600535.XSHG', '6
00703.XSHG', '300027.XSHE', '600100.XSHG', '600015.XSHG', '60051
9.XSHG', '002230.XSHE', '000963.XSHE', '002024.XSHE', '002465.XS
HE', '300059.XSHE', '002353.XSHE', '002653.XSHE', '600804.XSHG',
 '002038.XSHE', '002304.XSHE', '300017.XSHE', '600690.XSHG', '60
1933.XSHG', '600373.XSHG', '600637.XSHG', '600583.XSHG', '600588
.XSHG', '600718.XSHG', '600398.XSHG', '000793.XSHE', '600010.XSH
G', '002202.XSHE', '000039.XSHE', '000400.XSHE', '600633.XSHG',
'300251.XSHE', '600839.XSHG', '002008.XSHE', '300124.XSHE', '300
024.XSHE', '000413.XSHE', '300015.XSHE', '002292.XSHE', '601231.
XSHG', '601099.XSHG', '002475.XSHE', '603000.XSHG', '300002.XSHE
', '600998.XSHG', '601216.XSHG', '300058.XSHE', '300133.XSHE', '
600109.XSHG', '300104.XSHE', '002410.XSHE', '002153.XSHE']
2014-02-13 00:00:00 ['000738.XSHE', '601928.XSHG', '002024.XSHE'
, '002252.XSHE', '600008.XSHG', '000839.XSHE', '600648.XSHG', '6
00108.XSHG', '600518.XSHG', '000100.XSHE', '601098.XSHG', '00247
0.XSHE', '600118.XSHG', '002129.XSHE', '600340.XSHG', '000156.XS
HE', '002375.XSHE', '600570.XSHG', '600649.XSHG', '000413.XSHE',
 '300124.XSHE', '000400.XSHE', '600717.XSHG', '002594.XSHE', '60
1888.XSHG', '600705.XSHG', '002465.XSHE', '002153.XSHE', '600100
.XSHG', '600690.XSHG', '600633.XSHG', '300015.XSHE', '000970.XSH
E', '300059.XSHE', '600718.XSHG', '601998.XSHG', '002230.XSHE',
'601118.XSHG', '300024.XSHE', '000009.XSHE', '600998.XSHG', '002
008.XSHE', '300251.XSHE', '600398.XSHG', '002202.XSHE', '600839.
XSHG', '002410.XSHE', '601099.XSHG', '601216.XSHG', '600893.XSHG
', '600109.XSHG', '000712.XSHE', '600373.XSHG', '300017.XSHE', '
600588.XSHG', '601929.XSHG', '000559.XSHE', '601225.XSHG', '6032
88.XSHG']
2014-02-27 00:00:00 ['000598.XSHE', '600016.XSHG', '601179.XSHG'
, '601333.XSHG', '000738.XSHE', '000027.XSHE', '000503.XSHE', '6
01888.XSHG', '600518.XSHG', '601118.XSHG', '002008.XSHE', '60035
2.XSHG', '600519.XSHG', '002450.XSHE', '600109.XSHG', '600008.XS
HG', '002252.XSHE', '601169.XSHG', '002051.XSHE', '000839.XSHE',
 '601899.XSHG', '600570.XSHG', '600256.XSHG', '600893.XSHG', '00
0970.XSHE', '600028.XSHG', '300124.XSHE', '600153.XSHG', '300003
.XSHE', '000400.XSHE', '600489.XSHG', '600718.XSHG', '300017.XSH
E', '600547.XSHG', '300015.XSHE', '000725.XSHE', '002399.XSHE',
'600717.XSHG', '600688.XSHG', '600373.XSHG', '600166.XSHG', '000
100.XSHE', '000009.XSHE', '601098.XSHG', '600271.XSHG', '002304.
XSHE', '000960.XSHE', '600406.XSHG', '600398.XSHG', '000712.XSHE
', '002065.XSHE', '600196.XSHG', '601929.XSHG', '600588.XSHG', '
601998.XSHG', '002594.XSHE', '000559.XSHE', '601225.XSHG', '6032
88.XSHG']
2014-03-13 00:00:00 ['601169.XSHG', '600863.XSHG', '601929.XSHG'
, '000423.XSHE', '601225.XSHG', '300104.XSHE', '600489.XSHG', '6
```

```
00208.XSHG', '600660.XSHG', '000709.XSHE', '600010.XSHG', '60088
6.XSHG', '600535.XSHG', '300002.XSHE', '000778.XSHE', '000839.XS
HE', '600166.XSHG', '000539.XSHE', '601857.XSHG', '000009.XSHE',
 '600600.XSHG', '600050.XSHG', '000725.XSHE', '600153.XSHG', '00
2475.XSHE', '600177.XSHG', '600516.XSHG', '000027.XSHE', '300015
.XSHE', '002399.XSHE', '600068.XSHG', '000002.XSHE', '002594.XSH
E', '601919.XSHG', '600352.XSHG', '600875.XSHG', '600383.XSHG',
'600271.XSHG', '002146.XSHE', '601727.XSHG', '000568.XSHE', '600
547.XSHG', '600398.XSHG', '601098.XSHG', '600340.XSHG', '600406.
XSHG', '000960.XSHE', '002304.XSHE', '600028.XSHG', '601998.XSHG
', '300003.XSHE', '000858.XSHE', '600578.XSHG', '601179.XSHG', '
600196.XSHG', '600519.XSHG', '600688.XSHG', '000559.XSHE']
2014-03-27 00:00:00 ['600863.XSHG', '000839.XSHE', '002500.XSHE'
, '601668.XSHG', '000876.XSHE', '002001.XSHE', '601727.XSHG', '3
00027.XSHE', '600703.XSHG', '601766.XSHG', '600000.XSHG', '60078
3.XSHG', '600585.XSHG', '600109.XSHG', '002375.XSHE', '600886.XS
HG', '600315.XSHG', '600018.XSHG', '600060.XSHG', '600398.XSHG',
 '601901.XSHG', '600887.XSHG', '601377.XSHG', '000568.XSHE', '60
0177.XSHG', '600741.XSHG', '600519.XSHG', '600317.XSHG', '601919
.XSHG', '000937.XSHE', '601117.XSHG', '600875.XSHG', '600516.XSH
G', '000333.XSHE', '002202.XSHE', '600048.XSHG', '300017.XSHE',
'600663.XSHG', '000686.XSHE', '601633.XSHG', '300059.XSHE', '002
129.XSHE', '000061.XSHE', '000402.XSHE', '000858.XSHE', '601258.
XSHG', '000709.XSHE', '600383.XSHG', '600578.XSHG', '601929.XSHG
', '600549.XSHG', '600352.XSHG', '000002.XSHE', '000024.XSHE', '
600827.XSHG', '601992.XSHG', '002146.XSHE', '600340.XSHG']
2014-04-11 00:00:00 ['600674.XSHG', '600018.XSHG', '600549.XSHG'
, '000063.XSHE', '000728.XSHE', '601800.XSHG', '600015.XSHG', '6
01668.XSHG', '600999.XSHG', '601333.XSHG', '600060.XSHG', '60070
3.XSHG', '600863.XSHG', '002344.XSHE', '000333.XSHE', '600027.XS
HG', '600837.XSHG', '000709.XSHE', '601628.XSHG', '600019.XSHG',
 '000402.XSHE', '600839.XSHG', '600383.XSHG', '600886.XSHG', '60
0741.XSHG', '000024.XSHE', '601601.XSHG', '600048.XSHG', '601088
.XSHG', '601318.XSHG', '000898.XSHE', '601166.XSHG', '601766.XSH
G', '000937.XSHE', '600000.XSHG', '601688.XSHG', '600011.XSHG',
'000338.XSHE', '601390.XSHG', '000686.XSHE', '600104.XSHG', '300
059.XSHE', '600030.XSHG', '002129.XSHE', '002146.XSHE', '000156.
XSHE', '000046.XSHE', '601377.XSHG', '600827.XSHG', '002001.XSHE
', '601186.XSHG', '600352.XSHG', '600340.XSHG', '601992.XSHG', '
600585.XSHG', '000061.XSHE', '600570.XSHG']
2014-04-25 00:00:00 ['601601.XSHG', '000831.XSHE', '000001.XSHE'
, '603000.XSHG', '002570.XSHE', '600600.XSHG', '601933.XSHG', '0
00876.XSHE', '000858.XSHE', '002353.XSHE', '300024.XSHE', '00053
8.XSHE', '600597.XSHG', '002129.XSHE', '000063.XSHE', '002465.XS
HE', '000402.XSHE', '601901.XSHG', '601318.XSHG', '600048.XSHG',
 '600893.XSHG', '000960.XSHE', '000917.XSHE', '600011.XSHG', '30
0059.XSHE', '601186.XSHG', '600150.XSHG', '600585.XSHG', '601166
.XSHG', '600315.XSHG', '000728.XSHE', '600352.XSHG', '002415.XSH
E', '600739.XSHG', '600030.XSHG', '000046.XSHE', '600372.XSHG',
'000581.XSHE', '600317.XSHG', '600256.XSHG', '002475.XSHE', '600
519.XSHG', '300070.XSHE', '002008.XSHE', '000651.XSHE', '000333.
XSHE', '600104.XSHG', '002304.XSHE', '002241.XSHE', '000783.XSHE
', '600827.XSHG', '000156.XSHE', '002252.XSHE', '601231.XSHG', '
```

```
000625.XSHE', '600570.XSHG', '600383.XSHG']
2014-05-13 00:00:00 ['600863.XSHG', '002450.XSHE', '000425.XSHE'
, '601555.XSHG', '002153.XSHE', '600009.XSHG', '600549.XSHG', '0
02465.XSHE', '600783.XSHG', '600277.XSHG', '601699.XSHG', '60190
1.XSHG', '000060.XSHE', '002310.XSHE', '000538.XSHE', '300058.XS
HE', '000027.XSHE', '600050.XSHG', '601788.XSHG', '601618.XSHG',
 '002038.XSHE', '002500.XSHE', '600348.XSHG', '600383.XSHG', '00
0876.XSHE', '300070.XSHE', '002241.XSHE', '601898.XSHG', '002008
.XSHE', '000581.XSHE', '002415.XSHE', '002353.XSHE', '002304.XSH
E', '300059.XSHE', '600369.XSHG', '000983.XSHE', '000625.XSHE',
'601225.XSHG', '000831.XSHE', '000629.XSHE', '601168.XSHG', '000
963.XSHE', '000630.XSHE', '000402.XSHE', '300017.XSHE', '600188.
XSHG', '000728.XSHE', '000825.XSHE', '601958.XSHG', '000783.XSHE
', '600827.XSHG', '002475.XSHE', '000413.XSHE', '002252.XSHE', '
603993.XSHG', '601231.XSHG', '600317.XSHG']
2014-05-27 00:00:00 ['000027.XSHE', '000826.XSHE', '600535.XSHG'
, '000778.XSHE', '601168.XSHG', '601186.XSHG', '603000.XSHG', '6
00369.XSHG', '600050.XSHG', '000792.XSHE', '002294.XSHE', '60058
8.XSHG', '600011.XSHG', '600660.XSHG', '000768.XSHE', '600340.XS
HG', '002292.XSHE', '000825.XSHE', '603288.XSHG', '601788.XSHG',
 '000728.XSHE', '000983.XSHE', '002001.XSHE', '000729.XSHE', '60
0177.XSHG', '002065.XSHE', '300017.XSHE', '300024.XSHE', '600271
.XSHG', '600886.XSHG', '600276.XSHG', '600516.XSHG', '000629.XSH
E', '000630.XSHE', '300058.XSHE', '601699.XSHG', '600741.XSHG',
'000002.XSHE', '600867.XSHG', '002153.XSHE', '002410.XSHE', '601
958.XSHG', '600600.XSHG', '000963.XSHE', '002051.XSHE', '000402.
XSHE', '000831.XSHE', '300002.XSHE', '600633.XSHG', '300059.XSHE
', '601231.XSHG', '000413.XSHE', '002008.XSHE', '603993.XSHG', '
600188.XSHG', '300104.XSHE', '300133.XSHE', '600317.XSHG']
2014-06-11 00:00:00 ['601933.XSHG', '000651.XSHE', '601288.XSHG'
, '000503.XSHE', '601169.XSHG', '600196.XSHG', '000402.XSHE', '6
01628.XSHG', '601601.XSHG', '601336.XSHG', '600157.XSHG', '60192
8.XSHG', '601098.XSHG', '600317.XSHG', '600332.XSHG', '600485.XS
HG', '000333.XSHE', '000729.XSHE', '600900.XSHG', '000156.XSHE',
 '600718.XSHG', '002008.XSHE', '002051.XSHE', '000581.XSHE', '00
0826.XSHE', '600271.XSHG', '603000.XSHG', '000625.XSHE', '000024
.XSHE', '300024.XSHE', '601398.XSHG', '600633.XSHG', '002146.XSH
E', '000069.XSHE', '002410.XSHE', '002594.XSHE', '600867.XSHG',
'600276.XSHG', '600886.XSHG', '600383.XSHG', '002202.XSHE', '600
398.XSHG', '600048.XSHG', '300015.XSHE', '000413.XSHE', '000002.
XSHE', '000712.XSHE', '002465.XSHE', '300027.XSHE', '002292.XSHE
', '002129.XSHE', '002065.XSHE', '002024.XSHE', '600588.XSHG', '
300133.XSHE', '300002.XSHE', '300104.XSHE']
2014-06-25 00:00:00 ['600271.XSHG', '000423.XSHE', '600100.XSHG'
, '601118.XSHG', '600570.XSHG', '600739.XSHG', '600535.XSHG', '6
00549.XSHG', '600867.XSHG', '000883.XSHE', '600276.XSHG', '60087
3.XSHG', '600104.XSHG', '000858.XSHE', '600383.XSHG', '600038.XS
HG', '600109.XSHG', '600804.XSHG', '000831.XSHE', '600519.XSHG',
 '002465.XSHE', '002065.XSHE', '600372.XSHG', '000333.XSHE', '30
0059.XSHE', '601808.XSHG', '600718.XSHG', '601098.XSHG', '601231
.XSHG', '300070.XSHE', '601928.XSHG', '002038.XSHE', '002450.XSH
E', '000963.XSHE', '300104.XSHE', '600373.XSHG', '600315.XSHG',
'601601.XSHG', '600497.XSHG', '603993.XSHG', '600893.XSHG', '601
```

```
958.XSHG', '300251.XSHE', '002081.XSHE', '002129.XSHE', '002153.
XSHE', '300133.XSHE', '300027.XSHE', '000581.XSHE', '000768.XSHE
', '002292.XSHE', '002230.XSHE', '000712.XSHE', '000559.XSHE', '
300024.XSHE', '600516.XSHG', '000060.XSHE', '600317.XSHG']
2014-07-09 00:00:00 ['600718.XSHG', '600406.XSHG', '600570.XSHG'
, '600783.XSHG', '002344.XSHE', '600998.XSHG', '601118.XSHG', '6
01098.XSHG', '002310.XSHE', '000825.XSHE', '600027.XSHG', '30000
2.XSHE', '002594.XSHE', '000503.XSHE', '002465.XSHE', '600705.XS
HG', '601377.XSHG', '600150.XSHG', '000039.XSHE', '600317.XSHG',
 '000898.XSHE', '600873.XSHG', '000625.XSHE', '601989.XSHG', '60
1991.XSHG', '601766.XSHG', '600315.XSHG', '600804.XSHG', '600118
.XSHG', '601958.XSHG', '300124.XSHE', '601933.XSHG', '000423.XSH
E', '600372.XSHG', '600893.XSHG', '600038.XSHG', '603993.XSHG',
'600316.XSHG', '000400.XSHE', '600383.XSHG', '000963.XSHE', '002
410.XSHE', '000738.XSHE', '600497.XSHG', '600066.XSHG', '300024.
XSHE', '000768.XSHE', '000917.XSHE', '600485.XSHG', '601231.XSHG
', '002081.XSHE', '002230.XSHE', '600373.XSHG', '000839.XSHE', '
300017.XSHE', '600516.XSHG', '000060.XSHE', '000559.XSHE']
2014-07-23 00:00:00 ['600019.XSHG', '000423.XSHE', '300002.XSHE'
, '000825.XSHE', '601006.XSHG', '600674.XSHG', '600585.XSHG', '6
01336.XSHG', '601958.XSHG', '002129.XSHE', '000917.XSHE', '60015
0.XSHG', '600705.XSHG', '600900.XSHG', '600166.XSHG', '000999.XS
HE', '600887.XSHG', '000046.XSHE', '000898.XSHE', '000027.XSHE',
 '601377.XSHG', '000800.XSHE', '002146.XSHE', '601766.XSHG', '00
0060.XSHE', '600886.XSHG', '002001.XSHE', '000778.XSHE', '000002
.XSHE', '600153.XSHG', '600588.XSHG', '601555.XSHG', '600340.XSH
G', '000039.XSHE', '600406.XSHG', '002304.XSHE', '600549.XSHG',
'600519.XSHG', '600741.XSHG', '000876.XSHE', '000400.XSHE', '000
402.XSHE', '000858.XSHE', '000970.XSHE', '600066.XSHG', '601600.
XSHG', '000024.XSHE', '601633.XSHG', '601933.XSHG', '600048.XSHG
', '600011.XSHG', '000425.XSHE', '600809.XSHG', '601866.XSHG', '
000839.XSHE', '600485.XSHG', '000559.XSHE']
2014-08-06 00:00:00 ['000778.XSHE', '601901.XSHG', '000768.XSHE'
, '600109.XSHG', '601818.XSHG', '601117.XSHG', '000024.XSHE', '6
00519.XSHG', '600030.XSHG', '002001.XSHE', '000983.XSHE', '00006
1.XSHE', '600048.XSHG', '600309.XSHG', '000898.XSHE', '601688.XS
HG', '600188.XSHG', '600863.XSHG', '600348.XSHG', '000338.XSHE',
 '000559.XSHE', '601099.XSHG', '000046.XSHE', '000800.XSHE', '00
0883.XSHE', '002304.XSHE', '601225.XSHG', '601555.XSHG', '000876
.XSHE', '601377.XSHG', '601788.XSHG', '600111.XSHG', '600010.XSH
G', '600660.XSHG', '600690.XSHG', '601328.XSHG', '000568.XSHE',
'601600.XSHG', '000728.XSHE', '601958.XSHG', '000686.XSHE', '000
069.XSHE', '000878.XSHE', '601992.XSHG', '000937.XSHE', '000831.
XSHE', '600549.XSHG', '601699.XSHG', '600887.XSHG', '600809.XSHG
', '600741.XSHG', '002385.XSHE', '601633.XSHG', '000970.XSHE', '
601216.XSHG', '000783.XSHE', '600157.XSHG']
2014-08-20 00:00:00 ['002500.XSHE', '002241.XSHE', '000568.XSHE'
, '600688.XSHG', '002456.XSHE', '300059.XSHE', '601328.XSHG', '3
00015.XSHE', '002385.XSHE', '002230.XSHE', '002353.XSHE', '30005
8.XSHE', '600256.XSHG', '300070.XSHE', '000061.XSHE', '000768.XS
HE', '000878.XSHE', '601929.XSHG', '000156.XSHE', '002081.XSHE',
 '600516.XSHG', '002024.XSHE', '002450.XSHE', '000629.XSHE', '60
0718.XSHG', '600588.XSHG', '000937.XSHE', '600369.XSHG', '601377
```

```
.XSHG', '600485.XSHG', '600839.XSHG', '601699.XSHG', '600827.XSH
G', '300146.XSHE', '000792.XSHE', '300251.XSHE', '601118.XSHG',
'000970.XSHE', '000686.XSHE', '600348.XSHG', '000413.XSHE', '600
108.XSHG', '601888.XSHG', '000883.XSHE', '601216.XSHG', '600277.
XSHG', '601099.XSHG', '603000.XSHG', '600373.XSHG', '600208.XSHG
', '000783.XSHE', '000728.XSHE', '000009.XSHE', '600633.XSHG', '
600010.XSHG', '000960.XSHE', '600157.XSHG']
2014-09-03 00:00:00 ['601929.XSHG', '601231.XSHG', '600108.XSHG'
, '300015.XSHE', '000970.XSHE', '601888.XSHG', '601258.XSHG', '6
00588.XSHG', '601808.XSHG', '600900.XSHG', '002465.XSHE', '60010
0.XSHG', '600717.XSHG', '002230.XSHE', '000883.XSHE', '600718.XS
HG', '600118.XSHG', '300003.XSHE', '000156.XSHE', '000413.XSHE',
 '000826.XSHE', '600886.XSHG', '002456.XSHE', '000960.XSHE', '00
2475.XSHE', '002304.XSHE', '300070.XSHE', '603288.XSHG', '600316
.XSHG', '600023.XSHG', '600038.XSHG', '000728.XSHE', '601099.XSH
G', '600157.XSHG', '600839.XSHG', '300024.XSHE', '600256.XSHG',
'000768.XSHE', '600998.XSHG', '600674.XSHG', '601928.XSHG', '600
372.XSHG', '002202.XSHE', '002081.XSHE', '600570.XSHG', '600373.
XSHG', '002008.XSHE', '600893.XSHG', '000503.XSHE', '300059.XSHE
', '000738.XSHE', '600485.XSHG', '600633.XSHG', '603000.XSHG', '
000009.XSHE', '600317.XSHG']
2014-09-18 00:00:00 ['600038.XSHG', '600717.XSHG', '600578.XSHG'
, '601390.XSHG', '600900.XSHG', '600795.XSHG', '000629.XSHE', '0
02422.XSHE', '600023.XSHG', '600583.XSHG', '002236.XSHE', '60010
8.XSHG', '600406.XSHG', '600372.XSHG', '601377.XSHG', '601158.XS
HG', '002024.XSHE', '600221.XSHG', '601169.XSHG', '600398.XSHG',
 '601800.XSHG', '000598.XSHE', '300059.XSHE', '600893.XSHG', '30
0003.XSHE', '600029.XSHG', '600633.XSHG', '601866.XSHG', '601989
.XSHG', '601919.XSHG', '000046.XSHE', '000738.XSHE', '600115.XSH
G', '600886.XSHG', '601118.XSHG', '000503.XSHE', '600118.XSHG',
'600570.XSHG', '600060.XSHG', '000883.XSHE', '002570.XSHE', '600
415.XSHG', '002202.XSHE', '600008.XSHG', '603000.XSHG', '600674.
XSHG', '002375.XSHE', '300024.XSHE', '601618.XSHG', '000768.XSHE
', '601018.XSHG', '601106.XSHG', '600150.XSHG', '600485.XSHG', '
600316.XSHG', '600317.XSHG']
2014-10-09 00:00:00 ['600535.XSHG', '002230.XSHE', '601117.XSHG'
, '000425.XSHE', '600867.XSHG', '000999.XSHE', '600718.XSHG', '0
02007.XSHE', '600886.XSHG', '002422.XSHE', '000783.XSHE', '60040
6.XSHG', '600008.XSHG', '600276.XSHG', '002450.XSHE', '600317.XS
HG', '000623.XSHE', '600068.XSHG', '000598.XSHE', '600010.XSHG',
 '600115.XSHG', '002399.XSHE', '300059.XSHE', '601800.XSHG', '60
1158.XSHG', '601258.XSHG', '600029.XSHG', '601099.XSHG', '601555
.XSHG', '601377.XSHG', '601919.XSHG', '600060.XSHG', '002470.XSH
E', '600108.XSHG', '600415.XSHG', '601179.XSHG', '600570.XSHG',
'002570.XSHE', '601669.XSHG', '601118.XSHG', '600316.XSHG', '002
673.XSHE', '601225.XSHG', '601989.XSHG', '600485.XSHG', '600674.
XSHG', '002153.XSHE', '601727.XSHG', '002375.XSHE', '000061.XSHE
', '600717.XSHG', '000883.XSHE', '601618.XSHG', '002252.XSHE', '
601018.XSHG', '600150.XSHG', '601106.XSHG']
2014-10-23 00:00:00 ['601688.XSHG', '000100.XSHE', '000898.XSHE'
, '601618.XSHG', '600999.XSHG', '601179.XSHG', '000999.XSHE', '6
00271.XSHG', '600674.XSHG', '300059.XSHE', '600109.XSHG', '60002
3.XSHG', '601766.XSHG', '600588.XSHG', '300058.XSHE', '000917.XS
```

```
HE', '601336.XSHG', '601111.XSHG', '600315.XSHG', '002007.XSHE',
 '601788.XSHG', '600153.XSHG', '300133.XSHE', '601258.XSHG', '60
0867.XSHG', '002294.XSHE', '000839.XSHE', '000783.XSHE', '000800
.XSHE', '000738.XSHE', '000625.XSHE', '000629.XSHE', '601989.XSH
G', '000686.XSHE', '600570.XSHG', '600038.XSHG', '002230.XSHE',
'600372.XSHG', '601225.XSHG', '000623.XSHE', '000728.XSHE', '002
153.XSHE', '601555.XSHG', '002673.XSHE', '601099.XSHG', '601727.
XSHG', '600717.XSHG', '002399.XSHE', '002470.XSHE', '000883.XSHE
', '600150.XSHG', '000825.XSHE', '002252.XSHE', '601669.XSHG', '
601106.XSHG']
2014-11-06 00:00:00 ['601998.XSHG', '600170.XSHG', '601377.XSHG'
, '000539.XSHE', '600031.XSHG', '600900.XSHG', '002252.XSHE', '6
00068.XSHG', '000750.XSHE', '601333.XSHG', '600588.XSHG', '00077
6.XSHE', '000629.XSHE', '002294.XSHE', '600999.XSHG', '002142.XS
HE', '000800.XSHE', '600109.XSHG', '600005.XSHG', '000425.XSHE',
 '300059.XSHE', '000686.XSHE', '000709.XSHE', '000898.XSHE', '00
2007.XSHE', '300002.XSHE', '601018.XSHG', '000039.XSHE', '002673
.XSHE', '600867.XSHG', '600795.XSHG', '000623.XSHE', '600153.XSH
G', '601669.XSHG', '601919.XSHG', '000027.XSHE', '600663.XSHG',
'601111.XSHG', '601009.XSHG', '000100.XSHE', '600029.XSHG', '600
717.XSHG', '601336.XSHG', '601688.XSHG', '600221.XSHG', '600839.
XSHG', '601788.XSHG', '000825.XSHE', '601800.XSHG', '600115.XSHG
', '601186.XSHG', '601555.XSHG', '002051.XSHE', '000728.XSHE', '
000883.XSHE', '601390.XSHG']
2014-11-20 00:00:00 ['600741.XSHG', '601618.XSHG', '600068.XSHG'
, '600809.XSHG', '601668.XSHG', '000425.XSHE', '600648.XSHG', '6
01216.XSHG', '601118.XSHG', '600009.XSHG', '600795.XSHG', '60100
9.XSHG', '600999.XSHG', '002007.XSHE', '002500.XSHE', '600027.XS
HG', '600415.XSHG', '600578.XSHG', '000623.XSHE', '000783.XSHE',
 '000712.XSHE', '600109.XSHG', '600875.XSHG', '000686.XSHE', '60
1179.XSHG', '000503.XSHE', '600010.XSHG', '600170.XSHG', '601111
.XSHG', '000039.XSHE', '000709.XSHE', '000728.XSHE', '601992.XSH
G', '601727.XSHG', '600208.XSHG', '601006.XSHG', '000027.XSHE',
'600873.XSHG', '600340.XSHG', '600029.XSHG', '601186.XSHG', '600
100.XSHG', '601018.XSHG', '601919.XSHG', '600221.XSHG', '600827.
XSHG', '601555.XSHG', '600717.XSHG', '600867.XSHG', '002051.XSHE
', '601390.XSHG', '601377.XSHG', '300059.XSHE', '601800.XSHG', '
600115.XSHG', '600663.XSHG']
2014-12-04 00:00:00 ['600739.XSHG', '000651.XSHE', '601158.XSHG'
, '600000.XSHG', '300027.XSHE', '601225.XSHG', '600547.XSHG', '6
01958.XSHG', '000709.XSHE', '600015.XSHG', '600809.XSHG', '60001
0.XSHG', '600588.XSHG', '000402.XSHE', '600016.XSHG', '600048.XS
HG', '600827.XSHG', '601318.XSHG', '600783.XSHG', '600340.XSHG',
 '601992.XSHG', '000728.XSHE', '601555.XSHG', '601998.XSHG', '00
0878.XSHE', '603993.XSHG', '600570.XSHG', '000712.XSHE', '601111
.XSHG', '601328.XSHG', '600188.XSHG', '000024.XSHE', '601818.XSH
G', '000623.XSHE', '600109.XSHG', '600029.XSHG', '601336.XSHG',
'600115.XSHG', '000750.XSHE', '600030.XSHG', '601628.XSHG', '600
663.XSHG', '601099.XSHG', '000046.XSHE', '002673.XSHE', '000686.
XSHE', '600415.XSHG', '000783.XSHE', '300059.XSHE', '002500.XSHE
', '600208.XSHG', '000776.XSHE', '600999.XSHG', '601901.XSHG', '
601377.XSHG', '601688.XSHG', '601788.XSHG']
2014-12-18 00:00:00 ['600019.XSHG', '000425.XSHE', '600153.XSHG'
```

```
, '601158.XSHG', '600010.XSHG', '601939.XSHG', '600036.XSHG', '6
00221.XSHG', '600188.XSHG', '601998.XSHG', '601336.XSHG', '00000
1.XSHE', '000878.XSHE', '000898.XSHE', '000709.XSHE', '000027.XS
HE', '600016.XSHG', '600048.XSHG', '601318.XSHG', '601989.XSHG',
 '600109.XSHG', '601818.XSHG', '601628.XSHG', '601618.XSHG', '00
0623.XSHE', '600008.XSHG', '601186.XSHG', '600570.XSHG', '000728
.XSHE', '601669.XSHG', '601668.XSHG', '601555.XSHG', '000046.XSH
E', '600415.XSHG', '601111.XSHG', '600029.XSHG', '000024.XSHE',
'600068.XSHG', '000402.XSHE', '300059.XSHE', '000686.XSHE', '601
390.XSHG', '000750.XSHE', '002500.XSHE', '601800.XSHG', '601377.
XSHG', '601688.XSHG', '601099.XSHG', '002673.XSHE', '600837.XSHG
', '000776.XSHE', '600369.XSHG', '601788.XSHG', '601901.XSHG', '
600030.XSHG', '000783.XSHE', '600999.XSHG', '601969.XSHG']
2015-01-05 00:00:00 ['000002.XSHE', '000728.XSHE', '600583.XSHG'
, '600999.XSHG', '600019.XSHG', '600153.XSHG', '601998.XSHG', '6
00016.XSHG', '601601.XSHG', '600036.XSHG', '601288.XSHG', '00006
9.XSHE', '601727.XSHG', '600118.XSHG', '601333.XSHG', '601099.XS
HG', '600008.XSHG', '601111.XSHG', '000539.XSHE', '601166.XSHG',
 '600011.XSHG', '600352.XSHG', '601318.XSHG', '601117.XSHG', '00
0157.XSHE', '601600.XSHG', '600170.XSHG', '600578.XSHG', '600863
.XSHG', '601939.XSHG', '601919.XSHG', '601866.XSHG', '000024.XSH
E', '601179.XSHG', '601628.XSHG', '000783.XSHE', '000402.XSHE',
'600048.XSHG', '600027.XSHG', '600031.XSHG', '000425.XSHE', '002
673.XSHE', '601991.XSHG', '601989.XSHG', '600795.XSHG', '600886.
XSHG', '600837.XSHG', '601668.XSHG', '600068.XSHG', '600030.XSHG
', '601618.XSHG', '601186.XSHG', '601390.XSHG', '601800.XSHG', '
601669.XSHG', '600705.XSHG', '600369.XSHG', '002736.XSHE', '6019
69.XSHG']
2015-01-19 00:00:00 ['300002.XSHE', '600188.XSHG', '600875.XSHG'
, '601998.XSHG', '600348.XSHG', '601818.XSHG', '000651.XSHE', '6
00015.XSHG', '600332.XSHG', '000069.XSHE', '601898.XSHG', '60079
5.XSHG', '601169.XSHG', '600741.XSHG', '601939.XSHG', '601216.XS
HG', '600028.XSHG', '000002.XSHE', '601899.XSHG', '000858.XSHE',
 '600703.XSHG', '601166.XSHG', '600383.XSHG', '600066.XSHG', '00
2304.XSHE', '601398.XSHG', '600000.XSHG', '601328.XSHG', '601318
.XSHG', '600489.XSHG', '600547.XSHG', '002415.XSHE', '600309.XSH
G', '000625.XSHE', '000712.XSHE', '601088.XSHG', '600873.XSHG',
'000024.XSHE', '601727.XSHG', '000728.XSHE', '601288.XSHG', '600
886.XSHG', '601186.XSHG', '601601.XSHG', '600340.XSHG', '601857.
XSHG', '600048.XSHG', '002081.XSHE', '600588.XSHG', '600271.XSHG
', '600352.XSHG', '300104.XSHE', '300017.XSHE', '601988.XSHG', '
601628.XSHG', '600705.XSHG', '601766.XSHG', '002736.XSHE']
2015-02-02 00:00:00 ['002415.XSHE', '600332.XSHG', '600352.XSHG'
, '600100.XSHG', '002353.XSHE', '600038.XSHG', '000800.XSHE', '0
00831.XSHE', '603288.XSHG', '002456.XSHE', '300251.XSHE', '00242
2.XSHE', '000156.XSHE', '300124.XSHE', '002241.XSHE', '600705.XS
HG', '000400.XSHE', '601231.XSHG', '002450.XSHE', '600718.XSHG',
 '002024.XSHE', '000792.XSHE', '603993.XSHG', '600398.XSHG', '60
0570.XSHG', '600010.XSHG', '000503.XSHE', '002292.XSHE', '600703
.XSHG', '600271.XSHG', '300015.XSHE', '002230.XSHE', '600060.XSH
G', '600839.XSHG', '002475.XSHE', '300058.XSHE', '000793.XSHE',
'000712.XSHE', '600547.XSHG', '002153.XSHE', '300002.XSHE', '002
081.XSHE', '002653.XSHE', '002470.XSHE', '600804.XSHG', '601216.
```

```
XSHG', '000768.XSHE', '601727.XSHG', '002410.XSHE', '002065.XSHE
', '300017.XSHE', '300059.XSHE', '600588.XSHG', '300104.XSHE', '
601766.XSHG', '002736.XSHE', '000166.XSHE', '601021.XSHG']
2015-02-16 00:00:00 ['600068.XSHG', '002465.XSHE', '600019.XSHG'
, '002252.XSHE', '002310.XSHE', '601555.XSHG', '000568.XSHE', '3
00017.XSHE', '600999.XSHG', '002653.XSHE', '000559.XSHE', '60031
6.XSHG', '600398.XSHG', '300251.XSHE', '600373.XSHG', '601628.XS
HG', '000800.XSHE', '000503.XSHE', '600005.XSHG', '600642.XSHG',
 '603000.XSHG', '300146.XSHE', '000712.XSHE', '000768.XSHE', '30
0003.XSHE', '002065.XSHE', '600718.XSHG', '002470.XSHE', '600570
.XSHG', '600177.XSHG', '000792.XSHE', '600100.XSHG', '600705.XSH
G', '000100.XSHE', '600739.XSHG', '002344.XSHE', '002410.XSHE',
 '600170.XSHG', '600518.XSHG', '601231.XSHG', '000970.XSHE', '600
010.XSHG', '002230.XSHE', '000793.XSHE', '603993.XSHG', '601633.
XSHG', '600415.XSHG', '002024.XSHE', '600038.XSHG', '000831.XSHE
', '000839.XSHE', '600998.XSHG', '000156.XSHE', '002153.XSHE', '
000046.XSHE', '300104.XSHE', '600060.XSHG', '000166.XSHE', '6010
21.XSHG']
2015-03-09 00:00:00 ['600398.XSHG', '002065.XSHE', '002653.XSHE'
, '000960.XSHE', '000970.XSHE', '601618.XSHG', '600170.XSHG', '0
00046.XSHE', '000559.XSHE', '002024.XSHE', '601958.XSHG', '00225
2.XSHE', '600038.XSHG', '601377.XSHG', '603000.XSHG', '000839.XS
HE', '002470.XSHE', '002081.XSHE', '601929.XSHG', '000686.XSHE',
 '601098.XSHG', '300017.XSHE', '300146.XSHE', '600373.XSHG', '30
0133.XSHE', '300002.XSHE', '300003.XSHE', '603993.XSHG', '600050
.XSHG', '601600.XSHG', '601800.XSHG', '600177.XSHG', '000009.XSH
E', '601669.XSHG', '002153.XSHE', '601555.XSHG', '000917.XSHE',
'000425.XSHE', '601021.XSHG', '002673.XSHE', '600005.XSHG', '600
688.XSHG', '600415.XSHG', '000876.XSHE', '002230.XSHE', '601258.
XSHG', '600109.XSHG', '300251.XSHE', '600998.XSHG', '000100.XSHE
', '600570.XSHG', '600060.XSHG', '002594.XSHE', '000156.XSHE', '
000413.XSHE', '601216.XSHG', '300104.XSHE', '600518.XSHG']
2015-03-23 00:00:00 ['000060.XSHE', '000027.XSHE', '601225.XSHG'
, '601919.XSHG', '002007.XSHE', '600642.XSHG', '002292.XSHE', '6
01928.XSHG', '000425.XSHE', '600549.XSHG', '300017.XSHE', '60086
3.XSHG', '601688.XSHG', '000100.XSHE', '002375.XSHE', '002399.XS
HE', '600804.XSHG', '601166.XSHG', '002051.XSHE', '000963.XSHE',
 '000876.XSHE', '601866.XSHG', '000960.XSHE', '601600.XSHG', '60
1766.XSHG', '601018.XSHG', '000629.XSHE', '601390.XSHG', '601800
.XSHG', '600277.XSHG', '300146.XSHE', '601607.XSHG', '002142.XSH
E', '002038.XSHE', '600010.XSHG', '601186.XSHG', '000725.XSHE',
'601618.XSHG', '600177.XSHG', '601669.XSHG', '601111.XSHG', '600
570.XSHG', '600153.XSHG', '002129.XSHE', '000156.XSHE', '600518.
XSHG', '601258.XSHG', '002385.XSHE', '600688.XSHG', '600221.XSHG
', '600415.XSHG', '600115.XSHG', '000413.XSHE', '601216.XSHG', '
600109.XSHG', '600029.XSHG']
2015-04-07 00:00:00 ['002375.XSHE', '601933.XSHG', '002008.XSHE'
, '600633.XSHG', '601899.XSHG', '600008.XSHG', '601607.XSHG', '0
02081.XSHE', '601555.XSHG', '601669.XSHG', '600999.XSHG', '00077
6.XSHE', '000839.XSHE', '600340.XSHG', '600583.XSHG', '000686.XS
HE', '600369.XSHG', '600089.XSHG', '000725.XSHE', '000917.XSHE',
 '600837.XSHG', '300003.XSHE', '600739.XSHG', '000400.XSHE', '60
1111.XSHG', '002673.XSHE', '002230.XSHE', '600115.XSHG', '002153
```

```
.XSHE', '600718.XSHG', '601788.XSHG', '600153.XSHG', '300146.XSH
E', '601800.XSHG', '002202.XSHE', '601992.XSHG', '002344.XSHE',
'000559.XSHE', '601186.XSHG', '000629.XSHE', '600690.XSHG', '000
709.XSHE', '300058.XSHE', '000027.XSHE', '002410.XSHE', '600406.
XSHG', '002236.XSHE', '601688.XSHG', '601225.XSHG', '601390.XSHG
', '002146.XSHE', '600415.XSHG', '002129.XSHE', '600570.XSHG', '
600029.XSHG', '600958.XSHG']
2015-04-21 00:00:00 ['600276.XSHG', '600005.XSHG', '600863.XSHG'
, '600406.XSHG', '300002.XSHE', '600875.XSHG', '000778.XSHE', '0
02375.XSHE', '000063.XSHE', '002202.XSHE', '601006.XSHG', '60003
1.XSHG', '601225.XSHG', '600015.XSHG', '000709.XSHE', '600795.XS
HG', '000039.XSHE', '600048.XSHG', '601117.XSHG', '601600.XSHG',
 '002673.XSHE', '000001.XSHE', '601899.XSHG', '000738.XSHE', '60
0085.XSHG', '600029.XSHG', '000060.XSHE', '600153.XSHG', '601808
.XSHG', '300059.XSHE', '600068.XSHG', '601333.XSHG', '600873.XSH
G', '601179.XSHG', '000826.XSHE', '600089.XSHG', '600050.XSHG',
'601018.XSHG', '600018.XSHG', '601618.XSHG', '601668.XSHG', '600
221.XSHG', '600150.XSHG', '600583.XSHG', '600893.XSHG', '601919.
XSHG', '601727.XSHG', '601186.XSHG', '601989.XSHG', '601800.XSHG
', '601106.XSHG', '601866.XSHG', '600157.XSHG', '601390.XSHG', '
600958.XSHG', '601766.XSHG']
2015-05-06 00:00:00 ['600048.XSHG', '000825.XSHE', '002450.XSHE'
, '600028.XSHG', '600271.XSHG', '000009.XSHE', '601991.XSHG', '0
00651.XSHE', '600519.XSHG', '600839.XSHG', '600642.XSHG', '00200
7.XSHE', '601021.XSHG', '600252.XSHG', '002292.XSHE', '600150.XS
HG', '002500.XSHE', '000063.XSHE', '002129.XSHE', '601117.XSHG',
 '600018.XSHG', '601600.XSHG', '601333.XSHG', '000539.XSHE', '60
0637.XSHG', '601808.XSHG', '601179.XSHG', '600019.XSHG', '600221
.XSHG', '600068.XSHG', '000826.XSHE', '600863.XSHG', '000738.XSH
E', '600027.XSHG', '600893.XSHG', '600005.XSHG', '600011.XSHG',
'601186.XSHG', '002673.XSHE', '300002.XSHE', '601018.XSHG', '601
727.XSHG', '601668.XSHG', '300104.XSHE', '600050.XSHG', '600118.
XSHG', '600795.XSHG', '601989.XSHG', '601669.XSHG', '600688.XSHG
', '600157.XSHG', '601390.XSHG', '601766.XSHG', '601618.XSHG', '
601866.XSHG', '601919.XSHG', '601106.XSHG', '300059.XSHE']
2015-05-20 00:00:00 ['601600.XSHG', '000738.XSHE', '600637.XSHG'
, '600485.XSHG', '600316.XSHG', '601098.XSHG', '600827.XSHG', '6
00795.XSHG', '600111.XSHG', '600998.XSHG', '600893.XSHG', '00223
0.XSHE', '000559.XSHE', '601618.XSHG', '300027.XSHE', '600252.XS
HG', '600663.XSHG', '002310.XSHE', '002038.XSHE', '002236.XSHE',
 '002465.XSHE', '300133.XSHE', '002415.XSHE', '600688.XSHG', '00
0156.XSHE', '002410.XSHE', '600100.XSHG', '300070.XSHE', '601258
.XSHG', '000793.XSHE', '600588.XSHG', '002500.XSHE', '601727.XSH
G', '600038.XSHG', '002007.XSHE', '601669.XSHG', '000061.XSHE',
'601021.XSHG', '000503.XSHE', '601216.XSHG', '600060.XSHG', '600
373.XSHG', '002450.XSHE', '000917.XSHE', '300024.XSHE', '002153.
XSHE', '300124.XSHE', '600867.XSHG', '600804.XSHG', '002456.XSHE
', '002024.XSHE', '002129.XSHE', '600271.XSHG', '601106.XSHG', '
600118.XSHG', '000883.XSHE', '300059.XSHE', '300104.XSHE']
2015-06-03 00:00:00 ['300027.XSHE', '000061.XSHE', '002570.XSHE'
, '600166.XSHG', '002399.XSHE', '000878.XSHE', '000826.XSHE', '0
02450.XSHE', '600085.XSHG', '002065.XSHE', '600157.XSHG', '00200
8.XSHE', '601727.XSHG', '600153.XSHG', '300003.XSHE', '600060.XS
```

```
HG', '601258.XSHG', '002001.XSHE', '600642.XSHG', '600893.XSHG',
 '300015.XSHE', '300124.XSHE', '601216.XSHG', '601933.XSHG', '00
2470.XSHE', '600783.XSHG', '000060.XSHE', '000738.XSHE', '300058
.XSHE', '600118.XSHG', '002415.XSHE', '000768.XSHE', '000839.XSH
E', '002410.XSHE', '600170.XSHG', '601106.XSHG', '002653.XSHE',
'300251.XSHE', '600485.XSHG', '002375.XSHE', '000559.XSHE', '600
038.XSHG', '600100.XSHG', '002465.XSHE', '601018.XSHG', '000027.
XSHE', '002456.XSHE', '600271.XSHG', '002153.XSHE', '000503.XSHE
', '300133.XSHE', '600373.XSHG', '600023.XSHG', '600663.XSHG', '
300024.XSHE', '600588.XSHG', '000883.XSHE']
2015-06-17 00:00:00 ['601106.XSHG', '300017.XSHE', '000538.XSHE'
, '600008.XSHG', '000792.XSHE', '002422.XSHE', '600256.XSHG', '0
00858.XSHE', '000970.XSHE', '600221.XSHG', '000999.XSHE', '00015
7.XSHE', '000839.XSHE', '600188.XSHG', '300251.XSHE', '300003.XS
HE', '601929.XSHG', '002653.XSHE', '000983.XSHE', '600100.XSHG',
 '600415.XSHG', '600518.XSHG', '300015.XSHE', '000768.XSHE', '60
0873.XSHG', '000629.XSHE', '600085.XSHG', '000060.XSHE', '601969
.XSHG', '600642.XSHG', '002001.XSHE', '000937.XSHE', '300027.XSH
E', '002008.XSHE', '600863.XSHG', '002570.XSHE', '601333.XSHG',
'601018.XSHG', '600663.XSHG', '002146.XSHE', '600597.XSHG', '600
153.XSHG', '000069.XSHE', '601933.XSHG', '600703.XSHG', '601328.
XSHG', '601888.XSHG', '000027.XSHE', '600029.XSHG', '600166.XSHG
', '601898.XSHG', '600157.XSHG', '601258.XSHG', '600023.XSHG', '
600115.XSHG', '000559.XSHE', '600839.XSHG']
```

最经典的Momentum和Contrarian在中国市场的测试

> 来源：https://uqer.io/community/share/549b5bc8f9f06c4bb8863237

Momentum

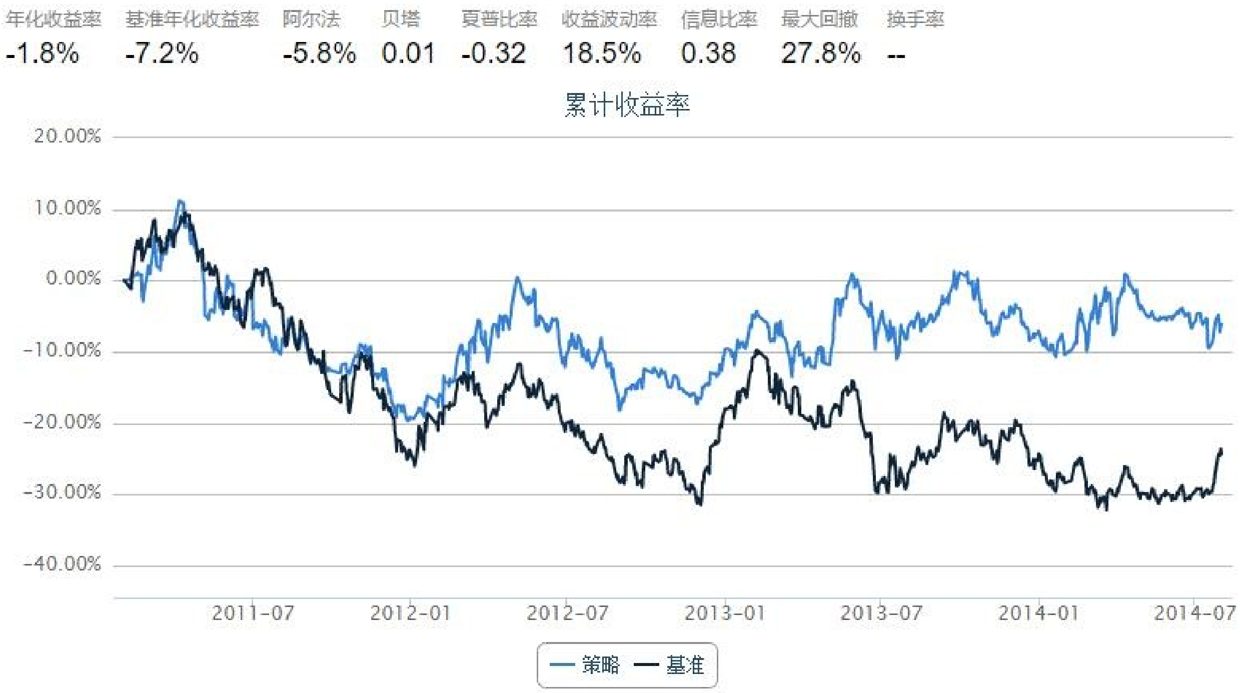
策略思路

- Momentum：业绩好的股票会继续保持其上涨的势头，业绩差的股票会保持其下跌的势头

策略实现

- Momentum：每次调仓将股票按照前一段时间的累计收益率排序并分组，买入历史累计收益 最高 的那一组

```
start = datetime(2011, 1, 1)                      # 回测起始时间
end   = datetime(2014, 8, 1)                      # 回测结束时间
benchmark = 'HS300'                                  # 使用沪深 300 作为
参考标准
universe = set_universe('SH50')                   # 股票池，上证50
capital_base = 100000                             # 起始资金
refresh_rate = 10
window = 20

def initialize(account):                          # 初始化虚拟账户状态
    account.amount = 300
    account.universe = universe
    add_history('hist', window)

def handle_data(account, data):                      # 每个交易日的买入卖
出指令
    momentum = {'symbol':[], 'c_ret':[]}
    for stk in account.hist:
        if 'closePrice' in account.hist[stk].columns:
            momentum['symbol'].append(stk)
            momentum['c_ret'].append(account.hist[stk].iloc[wind
ow-1,:]['closePrice']/account.hist[stk].iloc[0,:]['closePrice'])
    momentum = pd.DataFrame(momentum).sort(columns='c_ret').rese
t_index()
    momentum = momentum[len(momentum)*4/5:len(momentum)]
    buylist = momentum['symbol'].tolist()
    for stk in account.position.stkpos:
        if (stk not in buylist) and (account.position.stkpos[stk
]>0):
            order_to(stk, 0)
    for stk in buylist:
        if account.position.stkpos.get(stk, 0)==0:
            order_to(stk, account.amount)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
-1.8%	-7.2%	-5.8%	0.01	-0.32	18.5%	0.38	27.8%	--

Contrarian

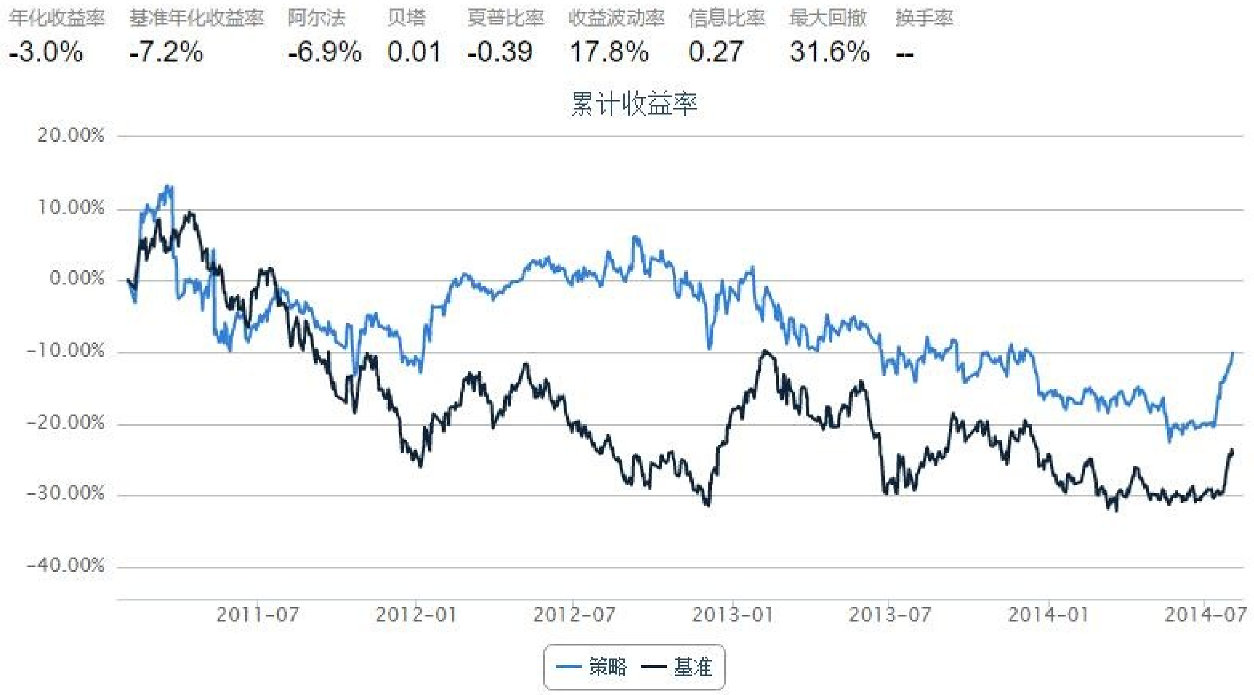
策略思路

- Contrarian：股票在经过一段时间的上涨之后会出现回落，一段时间的下跌之后会出现反弹

策略实现

- Contrarian：每次调仓将股票按照前一段时间的累计收益率排序并分组，买入历史累计收益 最低 的那一组

```
start = datetime(2011, 1, 1)                      # 回测起始时间
end   = datetime(2014, 8, 1)                      # 回测结束时间
benchmark = 'HS300'                                  # 使用沪深 300 作为
参考标准
universe = set_universe('SH50')                   # 股票池，上证50
capital_base = 100000                             # 起始资金
refresh_rate = 10
window = 20

def initialize(account):                          # 初始化虚拟账户状态
    account.amount = 300
    account.universe = universe
    add_history('hist', window)

def handle_data(account, data):                      # 每个交易日的买入卖
出指令
    contrarian = {'symbol':[], 'c_ret':[]}
    for stk in account.hist:
        if 'closePrice' in account.hist[stk].columns:
            contrarian['symbol'].append(stk)
            contrarian['c_ret'].append(account.hist[stk].iloc[wi
ndow-1,:]['closePrice']/account.hist[stk].iloc[0,:]['closePrice'
])
    contrarian = pd.DataFrame(contrarian).sort(columns='c_ret').
reset_index()
    contrarian = contrarian[:len(contrarian)/5]
    buylist = contrarian['symbol'].tolist()
    for stk in account.position.stkpos:
        if (stk not in buylist) and (account.position.stkpos[stk
]>0):
            order_to(stk, 0)
    for stk in buylist:
        if account.position.stkpos.get(stk, 0)==0:
            order_to(stk, account.amount)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
-3.0%	-7.2%	-6.9%	0.01	-0.39	17.8%	0.27	31.6%	--

最经典的Momentum和Contrarian在中国市场的测试-yanheven改进

> 来源：https://uqer.io/community/share/5663f369f9f06c6c8a91b3af

Momentum

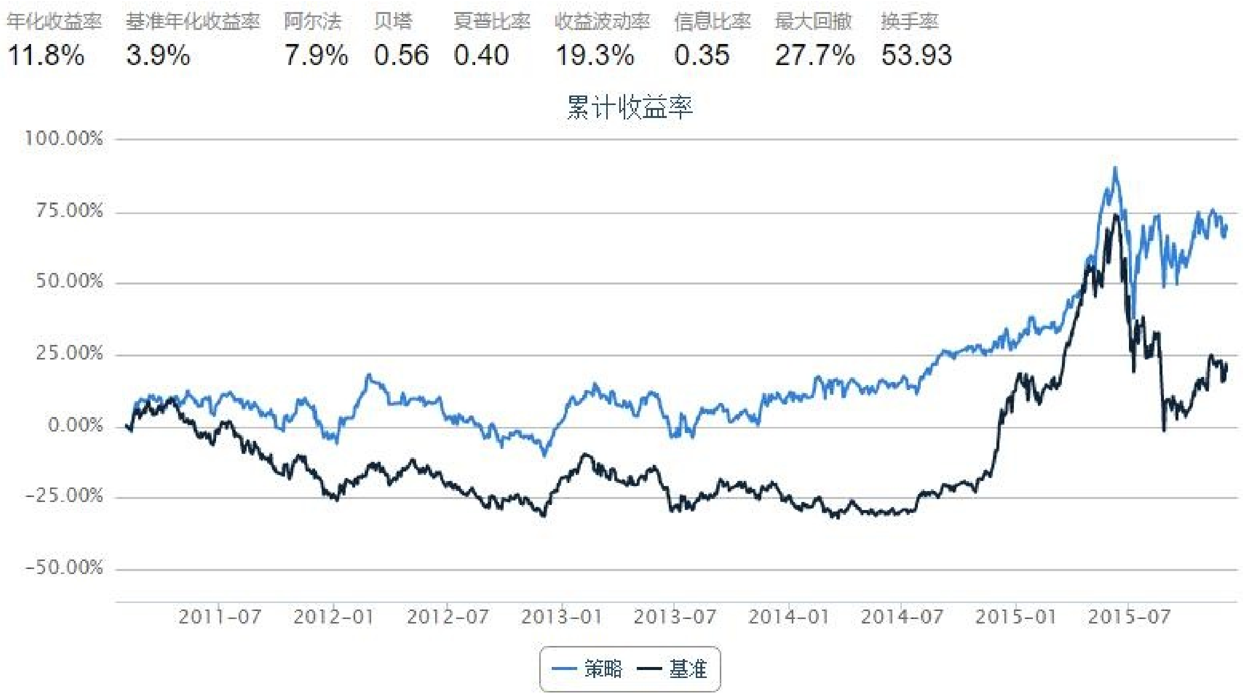
策略思路

- Momentum：业绩好的股票会继续保持其上涨的势头，业绩差的股票会保持其下跌的势头

策略实现

- Momentum：每次调仓将股票按照前一段时间的累计收益率排序并分组，买入历史累计收益 最高 的那一组

```
start = datetime(2011, 1, 1)                            # 回测起始时间
end   = datetime(2015, 12, 5)                            # 回测结束时间
benchmark = 'HS300'                                        # 使用沪深 300 作为
参考标准
universe = set_universe('HS300')                         # 股票池，沪深 300
capital_base = 100000                                   # 起始资金
refresh_rate = 10


def initialize(account):                                # 初始化虚拟账户状态
    pass


def handle_data(account):                            # 每个交易日的买入卖出指令
    history = account.get_attribute_history('closePrice', 20)
    momentum = []
    holding = account.valid_secpos
    for stk in history:
        if stk in account.universe:
            his = history[stk]
            change = his[-1] / his[0]
            momentum.append((stk, change))
    momentum = sorted(momentum, key=lambda x: x[1])
    momentum = momentum[:10]
    momentum_list = [i[0] for i in momentum]
    buy_list = set(momentum_list) - set(holding)
    sell_list = set(holding) - set(momentum_list)
    for i in buy_list:
        try:
            order_pct_to(i, 0.1)
        except Exception as e:
            log.warn(i + str(e))
    for i in sell_list:
        order_to(i, 0)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
11.8%	3.9%	7.9%	0.56	0.40	19.3%	0.35	27.7%	53.93

Contrarian

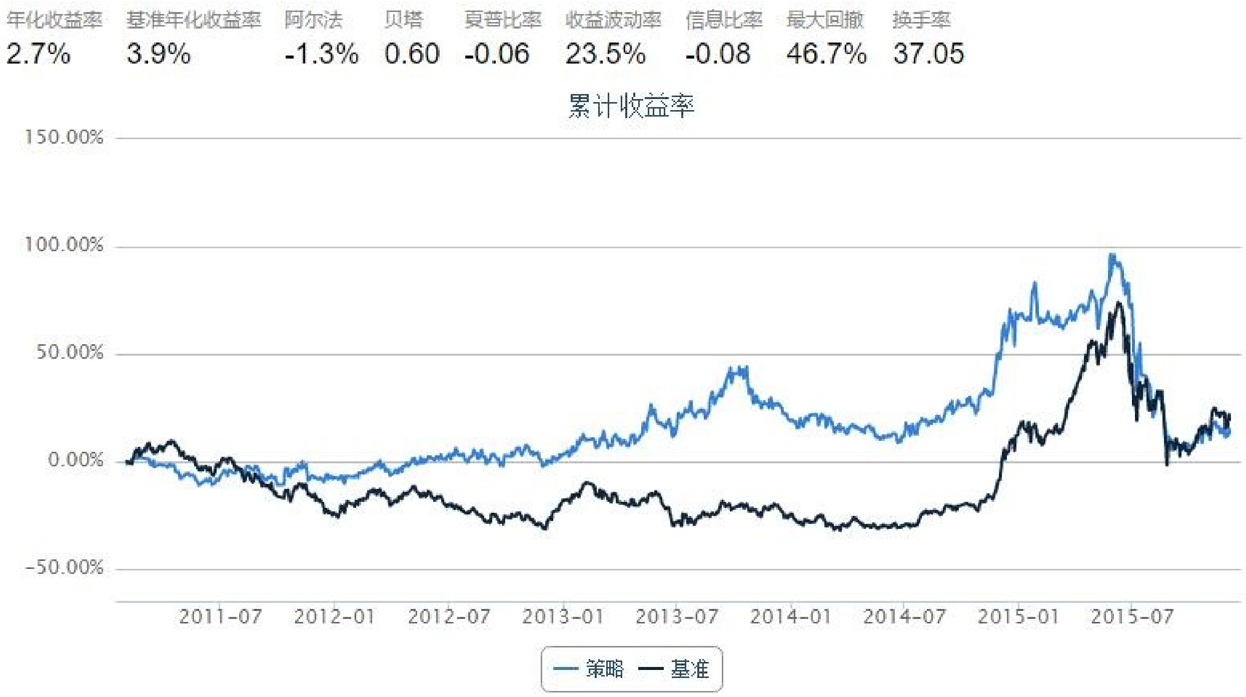
策略思路

- Contrarian：股票在经过一段时间的上涨之后会出现回落，一段时间的下跌之后会出现反弹

策略实现

- Contrarian：每次调仓将股票按照前一段时间的累计收益率排序并分组，买入历史累计收益 最低 的那一组

```
start = datetime(2011, 1, 1)                     # 回测起始时间
end   = datetime(2015, 12, 5)                     # 回测结束时间
benchmark = 'HS300'                                 # 使用沪深 300 作为
参考标准
universe = set_universe('HS300')                  # 股票池，沪深 300
capital_base = 100000                            # 起始资金
refresh_rate = 10


def initialize(account):                         # 初始化虚拟账户状态
    pass


def handle_data(account):                     # 每个交易日的买入卖出指令
    history = account.get_attribute_history('closePrice', 20)
    momentum = []
    holding = account.valid_secpos
    for stk in history:
        if stk in account.universe:
            his = history[stk]
            change = his[-1] / his[0]
            momentum.append((stk, change))
    momentum = sorted(momentum, key=lambda x: x[1], reverse=True
)
    # if momentum[-1][1] < 1:
    #     log.info(holding)
    #     for i in holding:
    #         order_to(i, 0)
    #     # return
    momentum = momentum[:10]
    momentum_list = [i[0] for i in momentum]
    buy_list = set(momentum_list) - set(holding)
    sell_list = set(holding) - set(momentum_list)
    for i in buy_list:
        try:
            order_pct_to(i, 0.1)
        except Exception as e:
            log.warn(i + str(e))
    for i in sell_list:
        order_to(i, 0)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
2.7%	3.9%	-1.3%	0.60	-0.06	23.5%	-0.08	46.7%	37.05

［策略］基于胜率的趋势交易策略

> 来源：https://uqer.io/community/share/565bbfa4f9f06c6c8a91ae7a

策略说明

简单构建了一个基于胜率的趋势交易策略。认为过去一段时间（N天）内胜率较高、信息比率较高的股票会在紧随其后的几天有较好的表现

1）先根据胜率要求筛选出过去N天胜率高的股票作为预选股票（benchmark可以是定义的确定阈值，或者是某个指数相应的收益率），用aprior算法进行快速筛选。第i只股票胜率的计算方式如下:

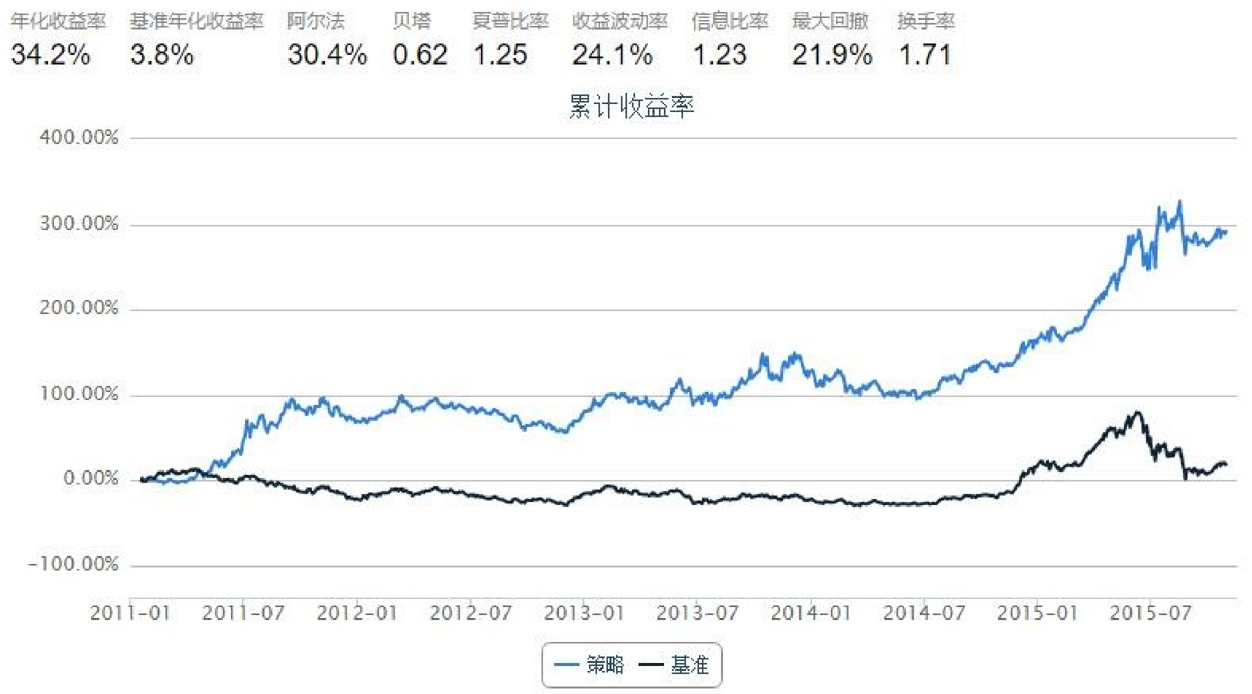
```
winRate(i) = sum([sign(ret(i,t)-ret(bm,t))==1]/N)|t~(t-N,t)
    ＊ret(i,t): i股票在第t天的收益率；
    ＊ret（bm,t）: benchmark在第t天的收益率；
```

2）从筛选出的股票中选择过去N天信息比率（收益率／波动率）高的部分股票构建备选投资组合； 3）依据被选投资组合做买入操作，使用可用资金的50%～70%；4）设定股票止损位在收益下跌至0.95，止损时将仓位调整至原仓位的40%～60%； 5）调仓频率为5天，股票池为沪深300。

```
import numpy as np
from CAL.PyCAL import *
################################################################
################
#    Back Test Functions
################################################################
################
def initialize(account):                    # 初始化虚拟账户状态
    pass

####init the univese of the choosen stock
def universeInit():
    stockComponent = DataAPI.MktTickRTSnapshotIndexGet(securityI
D=u"000300.XSHG",field=u"lastPrice,shortNM")
    stockCount = len(stockComponent)
    stockTicker = stockComponent['ticker']
    stockExchgID = stockComponent['exchangeCD']
    stockID = []
    for index in range(stockCount):
        stockID.append(stockTicker[index] + '.' + stockExchgID[i
ndex])
    return stockID

####deal with the trading signals
```

```
def handle_data(account):                     # 每个交易日的买入卖出指令

    ####Presettings
    histLength = 10
    stockDataThres = 0


    ####Dictionary of the return Rate
    closePrice = account.get_attribute_history('closePrice',histLength)
    retRate = {}
    for index in account.universe:
        retRate[index] = ((closePrice[index][1:] - closePrice[index][:-1])/closePrice[index][:-1]).tolist()


    ###ret list of the benchmark
    calendar = Calendar('China.SSE')
    startDate = calendar.advanceDate(account.current_date,'-'+str(histLength)+'B').toDateTime()
    benchmark = DataAPI.MktIdxdGet(ticker = "000300",
                field = "closeIndex",
                beginDate = startDate,
                endDate = account.current_date,pandas = '1')
    bmClose = benchmark['closeIndex'].tolist()
    bmRet = []
    for index in range(len(bmClose)-1):
        bmRet.append((bmClose[1:][index]-bmClose[:-1][index])/bmClose[:-1][index])

    ####List of transactions
    transactions = []
    for index in range(histLength-1):
        tmpt = []
        for stock in account.universe:
            if retRate[stock][index] > stockDataThres:
            # if retRate[stock][index] > bmRet[index]:
                tmpt.append(stock)
        transactions.append(tmpt)

    ####List of hot stocks
    hotStock = []
    hotStockDict,hotStockList = apriori(transactions,0.95)
    for index in hotStockList:
        for stock in index:
            if stock not in hotStock:
                hotStock.append(stock)

    ####List of the portfolio
    retRate = {}
    fluctRate = {}
    sharpRate = {}
    for index in hotStock:
        retRate[index] = ((closePrice[index][-1] - closePrice[in
dex][0])/closePrice[index][0])
        fluctRate[index] = np.std(closePrice[index])
        sharpRate[index] = retRate[index]/fluctRate[index]
    portfolio = [index[0] for index in sorted(sharpRate.items(),
key = lambda sharpRate:sharpRate[1])[-len(sharpRate)/2:]]


    ####Stop loss at -0.05
    validSecHist = account.get_attribute_history('closePrice',2)
    for index in account.valid_secpos:
        if (validSecHist[index][-1] - validSecHist[index][0])/va
lidSecHist[index][0] < -0.05:
            order_to(index,0.45*account.valid_secpos[index])

    ####Buy portfolio
    for index in portfolio:
        amount = 0.65*account.cash/len(hotStock)/account.referen
cePrice[index]
        order(index,amount)
    return


################################################################
########################
#    Aprior algorithm
################################################################
########################
def elementsDet(datasets):
    if type(datasets) == list:
        elements = {}
        for index in datasets:
            for index1 in index:
                if elements.has_key(index1) == False:
                    elements[index1] = 1
                else:
                    elements[index1] += 1
        return elements
    if type(datasets) == dict:
        elements = {}
        for index in datasets:
            if type(index) == tuple:
                index = list(index)
                for index1 in index:
                    if elements.has_key(index1) == False:
                        elements[index1] = 0
            else:
                elements[index] = 0
        return elements
    pass

def checkAssociation(subset,objset):
    for index in subset:
        if index not in objset:
            return False
    return True
    pass

def support(subset,datasets):
    count = 0
    for transaction in datasets:
        if checkAssociation(subset,transaction) == True:
            count += 1
    return 1.0*count/len(datasets)
    pass

def apriori(datasets,minsup):
    candidateIterator = []
    electIterator = []
    length = len(datasets)
    ##init part
    #the candidate
    elements = elementsDet(datasets)
    candidate = {}
    for index in elements:
            candidate[index] = 1.0*elements[index]/length
    candidateIterator.append(candidate)
    #the elect
    elect = {}
    for index in candidate:
        if candidate[index] > minsup:
            elect[index] = candidate[index]
    electIterator.append(elect)

    ##the update part
    itera = 1
    while(len(electIterator[-1]) != 0):

        candidateOld = candidateIterator[-1]
        electOld = electIterator[-1]
        elementsOld = elementsDet(electOld)
        # print elementsOld
        candidate = {}

        ##the candidate
        for index in electOld:
            for index1 in elementsOld:
                if type(index) != list and type(index) != tuple:
                    if index1 != index:
                        tmp = []
                        tmp.append(index)
                        tmp.append(index1)
                        tmp.sort()
                        if candidate.has_key(tuple(tmp)) == False
:
                            candidate[tuple(tmp)] = 0
                if type(index) == tuple:
                    tmp = list(index)
                    if tmp.count(index1) == False:
                        tmp1 = tmp
                        tmp1.append(index1)
                        tmp1.sort()
                        if candidate.has_key(tuple(tmp1)) == Fal
se:
                            candidate[tuple(tmp1)] = 0
        candidateIterator.append(candidate)

        ##the elect
        elect = {}
        for index in candidate:
            candidate[index] = support(index,datasets)

        for index in candidate:
            if candidate[index] > minsup:
                elect[index] = candidate[index]
        electIterator.append(elect)

        # print 'iteartion ' + str(itera) + ' is finished!'
        itera += 1

    ##the elected frequency sets dictionary: the value is the ke
y's support
    electedDict = {}
    for index in electIterator:
        for index1 in index:
            electedDict[index1] = index[index1]

    ##the elected frequency sets lists
    electedList = []
    for index in electIterator:
        tmp = []
        for index1 in index:
            if type(index1) == tuple:
                tmp1 = []
                for ele in index1:
                    tmp1.append(ele)
                tmp.append(tmp1)
            else:
                tmp.append([str(index1)])
        tmp.sort()
        for index1 in tmp:
            electedList.append(index1)

    return electedDict,electedList


################################################################
################
```

```
#    Back Test Presetting
################################################################################
start = '2011-01-01'                       # 回测起始时间
end = '2015-11-01'                         # 回测结束时间
benchmark = 'HS300'                        # 策略参考标准
universe = set_universe('HS300')
# universe = universeInit()                # 证券池，支持股票和基金
capital_base = 100000                      # 起始资金
freq = 'd'                                 # 策略类型，'d'表示日间策略使用日线回测，'m'表示日内策略使用分钟线回测
refresh_rate = 5                           # 调仓频率，表示执行handle_data的时间间隔，若freq = 'd'时间间隔的单位为交易日，若freq = 'm'时间间隔为分钟
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
34.2%	3.8%	30.4%	0.62	1.25	24.1%	1.23	21.9%	1.71

策略表现

- 策略能产生一定的alpha；
- 策略表现与起点强相关，sharpRatio不稳定；
- 策略表现会受到自身参数设定的影响，例如胜率选择周期、筛选阈值、调仓频率、建仓头寸、止损仓位等，需要依据表现对其进行优化；
- 策略在2011年4月至12月、2015年6月到11月有相对好的表现，可见其相对较适用于趋势下跌的市场环境。

问题探讨

因子选股模型的流程应该是怎样的？

小编认为构建因子选股的模型需要有如下过程：

1. 大类配置：根据宏观判断市场，进行市场判断（根据不同市场选择不同因子）、资产配置（不同风险性证券的配比选择➡不同热度的行业配比选择）和策略选择（市场中性、单边做多等）；
2. 选股－alpha端：对选股因子进行有效性分析，包括单因子的预测性、因子间相关性，构建多因子模型使得选股有尽可能高的alpha；
3. 选股－风险端：对alpha端的多因子模型进行风险评估，根据风险因子优化模型，使模型尽可能达到有效边界；
4. 择时－买卖时点：对根据因子模型选出的股票进行择时分析，进一步筛选投资组合中的股票及判断作何操作；

因子选股中比较basic的问题，欢迎社区的小伙伴们发表看法、评论和拍醒~

[量化策略] `Sharpe_Momentum` (夏普率动量策略)

> 来源：https://uqer.io/community/share/5656bd13f9f06c4446b48875

1. Introduction

众所周知，动量策略是量化选股中非常经典的模型。

思路是：从股票池中选取过去一段时间表现最好（收益率最高）的部分股票，等权重买入，到下一个调仓日清仓。然后周而复始。

与之相应的还有一个叫做“反转策略”，即认为投资者有抄底心态，过去表现最不好的股票将否极泰来。

2. Parameters

这类模型有以下几个参数可以调整：

1）换仓频率

2）过去表现的周期设定（即设定之前的多少天作为衡量基准）

3）股票池的选取：全部A股？或是弹性更高的中证500？或是流动性最好的上证50？

4）按照过去表现排序后，选择其中哪些股票？（Top 20%？Top 10%？或是中间的一部分？）

可以说，这类模型思路简单，逻辑明晰，参数变量少，是很棒的入门级策略

3. Development

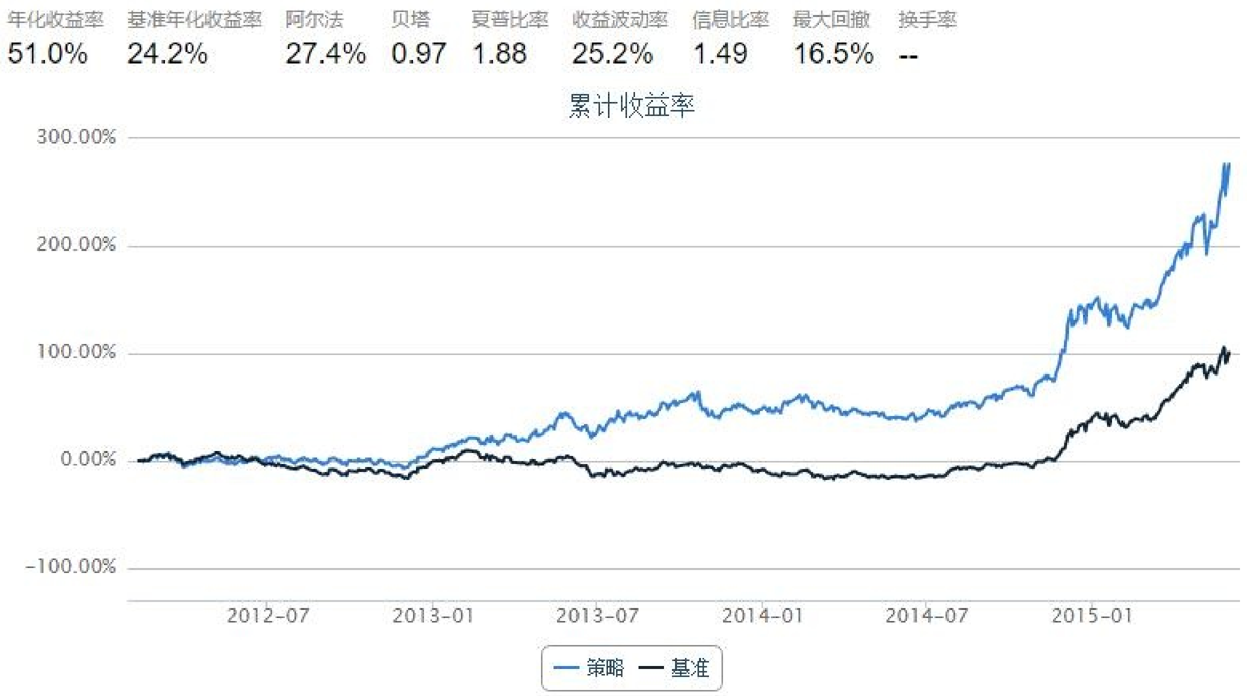
无论动量策略或是反转策略，都只考虑收益的变化，而对该收益率所承担的风险不闻不问。

笔者从夏普率的维度去看待动量策略，直觉上认为前一段时间夏普率高的股票将延续这一势头。（此处不考虑无风险利率）

PS：关于夏普率如何计算，可参考笔者的”[量化基础] 如何计算夏普率“一文

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

```
start = '2012-01-01'
end   = '2015-06-01'
benchmark = 'HS300'
universe = set_universe('HS300')
capital_base = 10000000
refresh_rate = 10

def initialize(account):
    pass

def handle_data(account):
    window = 20  #回望表现周期
    history = account.get_attribute_history('closePrice', window+
1) #多取一天收盘价，为了计算window个收益率
    history = pd.DataFrame(history)
    sharpe = {'symbol':[], 'ratio':[]} #设置一个字典
    for stk in account.universe:
        sharpe['symbol'].append(stk)  #字典中的symbol段 储存股票代码
        ret = history[stk].pct_change() #之前讲history转化成DataFr
ame结构，方便计算
        ratio = ret.mean() / ret.std() #夏普率简化为平均收益／收益波
动率，也不年化了，反正排序后效果是一致的
        sharpe['ratio'].append(ratio)


    # 按照过去window日收益率排序，并且选择前20%的股票作为买入候选
    sharpe = pd.DataFrame(sharpe).sort(columns = 'ratio').reset_
index()
    sharpe = sharpe[len(sharpe)*4/5:len(sharpe)]
    buylist = sharpe['symbol'].tolist()
    for stk in account.valid_secpos:
        if stk not in buylist:
            order_to(stk, 0)

    # 等权重买入所选股票
    portfolio_value = account.referencePortfolioValue

    filteredBuylist = []
    for stk in buylist:
        if not np.isnan(account.referencePrice[stk]):
            filteredBuylist.append(stk)

    #print account.current_date, filteredBuylist

    for stk in filteredBuylist:
        if stk not in account.valid_secpos:
            order_to(stk, int(portfolio_value / account.referenc
ePrice[stk] / 100.0 / len(buylist))*100)
```

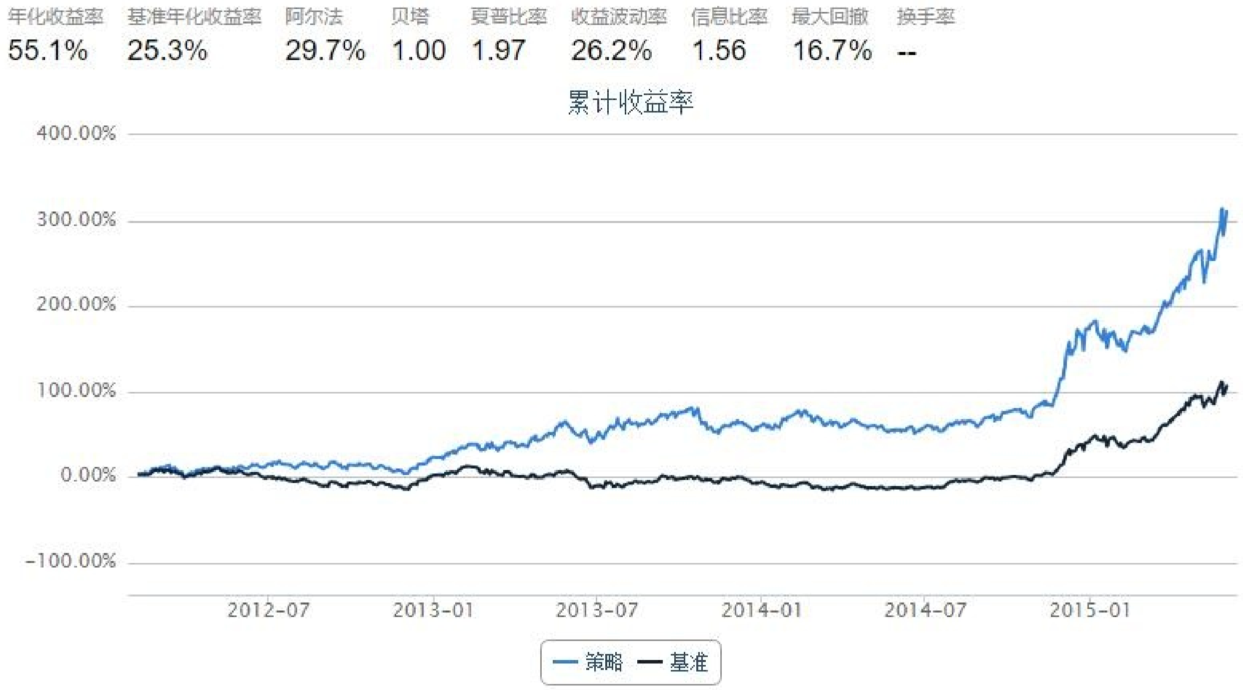


年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
51.0%	24.2%	27.4%	0.97	1.88	25.2%	1.49	16.5%	--

4. Comparison

同样的设定我们与下面的动量策略做比较

```
import numpy as np
import pandas as pd

start = '2012-01-01'
end   = '2015-06-01'
benchmark = 'HS300'
universe = set_universe('HS300')   # 股票池为沪深300
capital_base = 10000000
refresh_rate = 10

def initialize(account):
    pass

def handle_data(account):
    history = account.get_attribute_history('closePrice', 20)
    momentum = {'symbol':[], 'c_ret':[]}
    for stk in account.universe:
        momentum['symbol'].append(stk)
        momentum['c_ret'].append(history[stk][-1]/history[stk][0
])

    # 按照过去20日收益率排序，并且选择前20%的股票作为买入候选
    momentum = pd.DataFrame(momentum).sort(columns='c_ret').rese
t_index()
    momentum = momentum[len(momentum)*4/5:len(momentum)]   # 选择
    buylist = momentum['symbol'].tolist()
    for stk in account.valid_secpos:
        if stk not in buylist:
            order_to(stk, 0)

    # 等权重买入所选股票
    portfolio_value = account.referencePortfolioValue

    filteredBuylist = []
    for stk in buylist:
        if not np.isnan(account.referencePrice[stk]):
            filteredBuylist.append(stk)

    #print account.current_date, filteredBuylist

    for stk in filteredBuylist:
        if stk not in account.valid_secpos:
            order_to(stk, int(portfolio_value / account.referenc
ePrice[stk] / 100.0 / len(buylist))*100)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
55.1%	25.3%	29.7%	1.00	1.97	26.2%	1.56	16.7%	--

5. Futhur Discussion

无论是夏普率动量策略，或是收益率动量策略，表现看上去都是那么完美

但是！

但是！

但是！

一旦把2015年6月开始股灾放进回测区间，那么效果瞬间大打折扣

该策略还有许多值得改进之处，各位朋友发挥脑力吧。

疑问：比较两个策略，回测周期一致，基准一致，那么为什么基准年化收益率会不同？

策略探讨（更新）：价量结合+动量反转

> 来源：https://uqer.io/community/share/55b6f4d2f9f06c91fb18c5cd

前篇简介:

- 在前一篇文章中，对策略理念和流程大致走了一遍
- 对quartz中的参数优化过程以及简单策略分析也做了简单示例

仍存在的问题:

- 前一篇中代码细节问题修正
- 停牌数据问题以及其他小细节
- 收益波动性太大、熊市不抗跌

下面就结合实际情况，对上述问题进行逐个分析

- 没有看前篇的，点此链接 https://uqer.io/community/share/55b1f886f9f06c91f918c5d1

```
# step1：前篇代码细节修正
# 如前篇描述，（-inf，negative2,negative1,postive1,positive2,inf）将整个数轴分成了5个区间段，从左到右，每个区间段分别代表（long,short,nothing,long,short）操作
# 而在前篇的代码中忽略掉了小细节，将修正后的部分代码简单展示如下：
    signal = closeprice[stk][-1]/wp - 1
    if stk in account.valid_secpos and (signal > positive2 or negative2 < signal < negative1):
        selllist.append(stk)
    elif stk not in account.valid_secpos and (positive2 > signal > positive1 or signal < negative2):
        buylist.append(stk)
```

step2：数据质量问题再分析

- 首先分析一下数据细节问题，策略中唯一要计算的指标是加权平均价，要用到过去一段时间的收盘价和成交量，倘若过去时间里有停牌呢？
- 之前的简单处理是，若全部停牌会导致wp计算出来为nan，然后把它舍弃掉，但是倘若过去所用的时间段内刚巧只有一天是没停牌的呢？
- 查看DataAPI原始数据发现，停牌时，收盘价是filldown处理的，当日成交量是为0的
- 所以要保证所计算出来的加权平均价wp的相对合理性，必须要保证所计算的过去数据的有效性，这里的处理是：若停牌天数超过window/2时，则不进行任何操作

```
# step2对应的代码修改部分（非运行代码，只做展示说明，下同）
....
for stk in account.universe:
  if sum(volumn[stk] > 0) <= window/2: continue
      ....
```

step3：收益波动性太大

- 如前篇描述，（-inf，negative2,negative1,postive1,positive2,inf）将整个数轴分成了5个区间段，从左到右，每个区间段分别代表（long,short,nothing,long,short）操作
- 结合实际考虑，股市里目前没有实际的short操作，回测的策略只是LongOnly，对应的short信号只是卖掉持有的股票，所以要结合实际对信号做一些更改
- 整体思路是：买入的信号不变，short的信号是卖掉手中股票的信号，而并不是开空仓的信号，因为即使预测对了下跌也是赚不到钱的
- 改进点是： negative1变为0

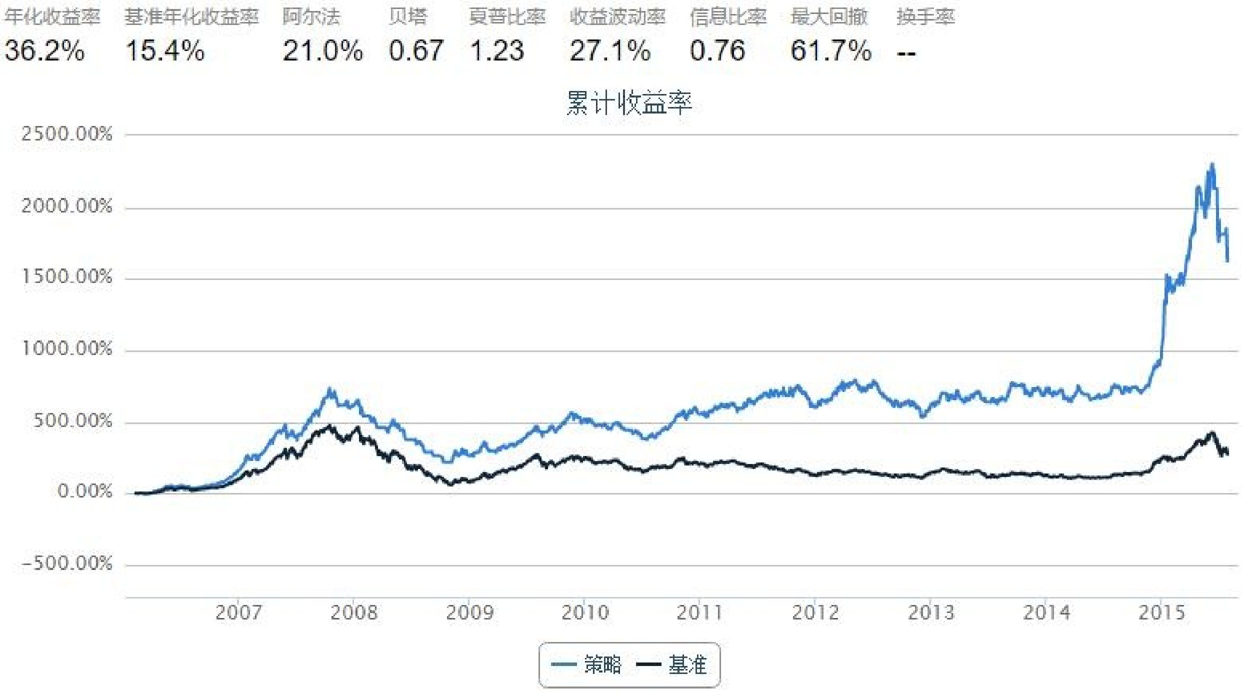
```
# step3：对应要修改的代码
negative1 = 0
```

step4：熊市不抗跌

- 遇到熊市时，可以考虑设置止损线，虽然不能做空，但是降低仓位可以逃过一部分损失
- 止损条件有两点考虑：一是现有组合市值短期亏损比较大就止损；二是大盘过去一段时间里下跌很多，要止损（近似于宏观择时操作）
- 止损条件1：当沪深300在过去3周累计下跌超过10%时，下一期不持仓
- 止损条件2：当组合净值在过去3周累计缩水超过10%或者在过去一周累计缩水5%，下一期不持仓

```
from datetime import datetime, timedelta
import numpy as np

start = '2006-01-01'                       # 回测起始时间
# end = '2015-01-01'
today = datetime.today()
delta = timedelta(days = -1)
end = (today+delta).strftime('%Y-%m-%d')
benchmark = 'HS300'                        # 策略参考标准
universe = set_universe('SH50')  # 证券池，回测支持股票和基金
capital_base = 10000000                     # 起始资金
refresh_rate = 5                          # 调仓频率，即每 refresh
_rate 个交易日执行一次 handle_data() 函数

window = 20   # 取过去行情数据
positive2 = 0.1   # 信号突破比例阈值
negative2 = -0.1
```

```
positive1 = 0.01
negative1 = 0
stoploss_percent = 0


def initialize(account):                        # 初始化虚拟账户状态
    account.portfoliovalue = []  #记录组合市值

def handle_data(account):                       # 每个交易日的买入卖出指令


    # 止损判断
    hs300 = account.get_symbol_history('benchmark',window)['closeIndex']
    account.portfoliovalue.append(account.referencePortfolioValue)
    if hs300[-1]/hs300[-16] - 1 < -0.1:  # 止损条件1
        for stk in account.valid_secpos:
            order_to(stk,0)
        return
    if len(account.portfoliovalue) < 4:  # 止损条件2
        pass
    elif account.portfoliovalue[-1]/account.portfoliovalue[-4] - 1 <= -0.1 or account.portfoliovalue[-1]/account.portfoliovalue[-2] - 1 <= -0.05:
        for stk in account.valid_secpos:
            order_to(stk,0)
        return

    # 取行情
    closeprice = account.get_attribute_history('closePrice',window)
    volumn = account.get_attribute_history('turnoverVol',window)
    reference_p = account.referencePrice    # 参考价
    cash = account.cash   # 剩余现金

    # 计算买入卖出
    buylist = []
    selllist = []
    for stk in account.universe:
        if sum(volumn[stk] > 0) <= window/2: continue
        wp = np.dot(closeprice[stk], volumn[stk]/sum(volumn[stk]))   #  成交量加权价
        signal = closeprice[stk][-1]/wp - 1
        if stk in account.valid_secpos and (signal > positive2 or negative2 < signal < negative1):
            selllist.append(stk)
        elif positive2 > signal > positive1 or signal < negative2:
            buylist.append(stk)

    # 卖出下单
    for stk in selllist:
        order_to(stk,0)
        cash = cash + account.valid_secpos[stk] * reference_p[stk]

    # 买单分解为需要新买的+已有持仓的
    holds = [stk for stk in account.valid_secpos if stk not in selllist]
    new_buy = [stk for stk in buylist if stk not in holds]

    # 买入操作
    for stk in new_buy:
        order(stk,int(cash/len(new_buy)/reference_p[stk]))
```



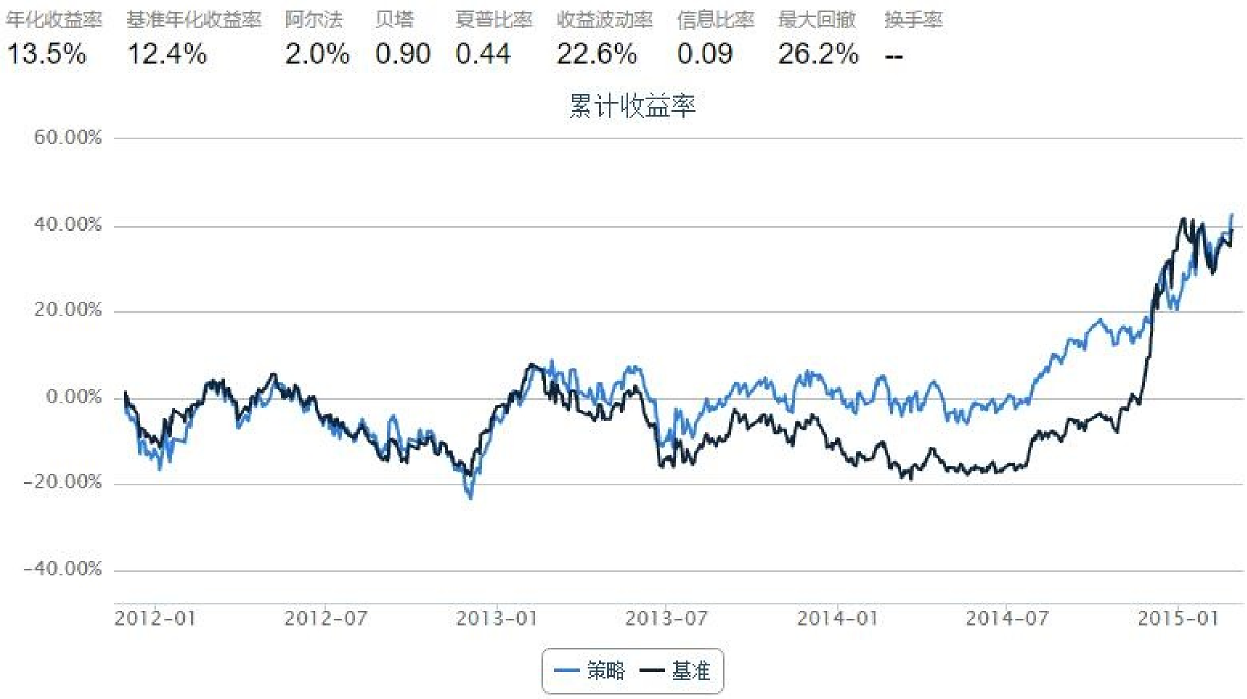
年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
36.2%	15.4%	21.0%	0.67	1.23	27.1%	0.76	61.7%	--

- 可以看到，加入简单的止损后，策略效果有明显改善，尤其是在07年~08年熊市阶段，躲过了很多下跌
- 同时也看到，在最近的大跌行情中表现一般，主要原因是周度策略，一周换一次仓，周中的大跌是难以避免的，一个好的做法是考虑周中止损，这一点可以自行尝试
- 除了直接的止损外，还可以考虑加入择时，比如预测涨就全仓，预测跌就1/3仓位等，上面展示的只是最简单的想法，更细致的东西还有待去研究

反向动量策略（reverse momentum driven）

> 来源：https://uqer.io/community/share/5562c046f9f06c6c7404f9e3

```
import pandas as pd

start = '2011-11-01'
end   = '2015-03-01'
benchmark = 'HS300'
universe = set_universe('HS300')    # 股票池为沪深300
capital_base = 10000000
refresh_rate = 10

def initialize(account):
    pass

def handle_data(account):
    history = account.get_attribute_history('closePrice', 20)
    momentum = {'symbol':[], 'c_ret':[]}
    for stk in account.universe:
        momentum['symbol'].append(stk)
        momentum['c_ret'].append(history[stk][-1]/history[stk][0
])

    # 按照过去20日收益率排序，并且选择前20%的股票作为买入候选
    momentum = pd.DataFrame(momentum).sort(columns='c_ret', asce
nding=False).reset_index()
    momentum = momentum[len(momentum)*4/5:len(momentum)]    # 选择
    buylist = momentum['symbol'].tolist()
    for stk in account.valid_secpos:
        if stk not in buylist:
            order_to(stk, 0)

    # 等权重买入所选股票
    portfolio_value = account.referencePortfolioValue
    for stk in buylist:
        if stk not in account.valid_secpos:
            order_to(stk, int(portfolio_value / account.referenc
ePrice[stk] / 100.0 / len(buylist))*100)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
13.5%	12.4%	2.0%	0.90	0.44	22.6%	0.09	26.2%	--

```
bt
```

	tradeDate	cash	security_position	portfolio_value	bei
0	2011-11-29	1.627636	{u'600809.XSHG': 5700, u'600597.XSHG': 21100, ...	10057034.029636	0.0
1	2011-11-30	1.627636	{u'600809.XSHG': 5700, u'600597.XSHG': 21100, ...	9744501.409636	-0.(
2	2011-12-01	1.627636	{u'600809.XSHG': 5700, u'600597.XSHG': 21100, ...	9865655.500636	0.0
3	2011-12-02	1.627636	{u'600809.XSHG': 5700, u'600597.XSHG': 21100, ...	9708455.312636	-0.(
4	2011-12-05	1.627636	{u'600809.XSHG': 5700, u'600597.XSHG': 21100, ...	9406180.945636	-0.(
			{u'600809.XSHG':		

5	2011-12-06	1.627636	5700, u'600597.XSHG': 21100, ...	9415313.949636	-0.0
6	2011-12-07	1.627636	{u'600809.XSHG': 5700, u'600597.XSHG': 21100, ...	9454383.554636	0.0
7	2011-12-08	1.627636	{u'600809.XSHG': 5700, u'600597.XSHG': 21100, ...	9492264.755636	-0.0
8	2011-12-09	1.627636	{u'600809.XSHG': 5700, u'600597.XSHG': 21100, ...	9410892.851636	-0.0
9	2011-12-12	1.627636	{u'600809.XSHG': 5700, u'600597.XSHG': 21100, ...	9280515.559636	-0.0
10	2011-12-13	6.144010	{u'000423.XSHE': 4200, u'600036.XSHG': 16300, ...	9116051.541010	-0.0
11	2011-12-14	6.144010	{u'000423.XSHE': 4200, u'600036.XSHG': 16300, ...	9042660.489010	-0.0
12	2011-12-15	6.144010	{u'000423.XSHE': 4200, u'600036.XSHG': 16300, ...	8973191.009010	-0.0
13	2011-12-16	6.144010	{u'000423.XSHE': 4200, u'600036.XSHG': 16300, ...	9203671.207010	0.0
14	2011-12-19	6.144010	{u'000423.XSHE': 4200, u'600036.XSHG': 16300, ...	9272914.609010	-0.0
15	2011-12-20	6.144010	{u'000423.XSHE': 4200, u'600036.XSHG':	9245375.546010	-0.0

			16300, ...		
16	2011-12-21	6.144010	{u'000423.XSHE': 4200, u'600036.XSHG': 16300, ...	9134328.126010	-0.0
17	2011-12-22	6.144010	{u'000423.XSHE': 4200, u'600036.XSHG': 16300, ...	9174823.752010	0.0
18	2011-12-23	6.144010	{u'000423.XSHE': 4200, u'600036.XSHG': 16300, ...	9185929.702010	0.0
19	2011-12-26	6.144010	{u'000423.XSHE': 4200, u'600036.XSHG': 16300, ...	9054507.546010	-0.0
20	2011-12-27	9.172263	{u'601328.XSHG': 41600, u'600036.XSHG': 16300,...	8855733.701263	-0.0
21	2011-12-28	9.172263	{u'601328.XSHG': 41600, u'600036.XSHG': 16300,...	8820469.780263	0.0
22	2011-12-29	9.172263	{u'601328.XSHG': 41600, u'600036.XSHG': 16300,...	8836530.533263	0.0
23	2011-12-30	9.172263	{u'601328.XSHG': 41600, u'600036.XSHG': 16300,...	8976704.526263	0.0
24	2012-01-04	9.172263	{u'601328.XSHG': 41600, u'600036.XSHG': 16300,...	8858664.267263	-0.0
25	2012-01-05	9.172263	{u'601328.XSHG': 41600, u'600036.XSHG': 16300,...	8785873.316263	-0.0

26	2012-01-06	9.172263	{u'601328.XSHG': 41600, u'600036.XSHG': 16300,...	8790219.626263	0.0
27	2012-01-09	9.172263	{u'601328.XSHG': 41600, u'600036.XSHG': 16300,...	9032347.125263	0.0
28	2012-01-10	9.172263	{u'601328.XSHG': 41600, u'600036.XSHG': 16300,...	9247829.118263	0.0
29	2012-01-11	9.172263	{u'601328.XSHG': 41600, u'600036.XSHG': 16300,...	9186427.159263	-0.0
...	...	...	...	...	...
755	2015-01-12	1.942737	{u'600036.XSHG': 23600, u'000776.XSHE': 22100,...	20280828.092737	-0.0
756	2015-01-13	1.942737	{u'600036.XSHG': 23600, u'000776.XSHE': 22100,...	20204218.432737	0.0
757	2015-01-14	1.942737	{u'600036.XSHG': 23600, u'000776.XSHE': 22100,...	20223885.602737	-0.0
758	2015-01-15	1.942737	{u'600036.XSHG': 23600, u'000776.XSHE': 22100,...	20962393.212737	0.0
759	2015-01-16	1.942737	{u'600036.XSHG': 23600, u'000776.XSHE': 22100,...	21076614.722737	0.0
760	2015-01-19	3.731317	{u'601328.XSHG': 49500, u'600066.XSHG': 21600,...	19298994.131317	-0.0

761	2015-01-20	3.731317	{u'601328.XSHG': 49500, u'600066.XSHG': 21600,...	19641436.921317	0.0
762	2015-01-21	3.731317	{u'601328.XSHG': 49500, u'600066.XSHG': 21600,...	20548813.131317	0.0
763	2015-01-22	3.731317	{u'601328.XSHG': 49500, u'600066.XSHG': 21600,...	20654164.321317	0.0
764	2015-01-23	3.731317	{u'601328.XSHG': 49500, u'600066.XSHG': 21600,...	20704527.401317	0.0
765	2015-01-26	3.731317	{u'601328.XSHG': 49500, u'600066.XSHG': 21600,...	20741863.411317	0.0
766	2015-01-27	3.731317	{u'601328.XSHG': 49500, u'600066.XSHG': 21600,...	20501439.751317	-0.0
767	2015-01-28	3.731317	{u'601328.XSHG': 49500, u'600066.XSHG': 21600,...	20072450.291317	-0.0
768	2015-01-29	3.731317	{u'601328.XSHG': 49500, u'600066.XSHG': 21600,...	19879912.781317	-0.0
769	2015-01-30	3.731317	{u'601328.XSHG': 49500, u'600066.XSHG': 21600,...	19612742.771317	-0.0
770	2015-02-02	6.734377	{u'002153.XSHE': 4400, u'600498.XSHG': 19400, ...	19457712.474377	-0.0
	2015-02-		{u'002153.XSHE': 4400,		

771	03	6.734377	u'600498.XSHG': 19400, ...	19862476.714377	0.0
772	2015-02-04	6.734377	{u'002153.XSHE': 4400, u'600498.XSHG': 19400, ...	19714886.724377	-0.
773	2015-02-05	6.734377	{u'002153.XSHE': 4400, u'600498.XSHG': 19400, ...	19679598.144377	-0.
774	2015-02-06	6.734377	{u'002153.XSHE': 4400, u'600498.XSHG': 19400, ...	19297569.294377	-0.
775	2015-02-09	6.734377	{u'002153.XSHE': 4400, u'600498.XSHG': 19400, ...	19131675.244377	0.0
776	2015-02-10	6.734377	{u'002153.XSHE': 4400, u'600498.XSHG': 19400, ...	19621103.094377	0.0
777	2015-02-11	6.734377	{u'002153.XSHE': 4400, u'600498.XSHG': 19400, ...	20007845.234377	0.0
778	2015-02-12	6.734377	{u'002153.XSHE': 4400, u'600498.XSHG': 19400, ...	20139049.804377	0.0
779	2015-02-13	6.734377	{u'002153.XSHE': 4400, u'600498.XSHG': 19400, ...	20416511.184377	0.0
780	2015-02-16	5.417277	{u'002153.XSHE': 4400, u'600498.XSHG': 19400, ...	21015638.057277	0.0
781	2015-02-17	5.417277	{u'002153.XSHE': 4400, u'600498.XSHG':	20991534.387277	0.0

782	2015-02-25	5.417277	{u'002153.XSHE': 4400, u'600498.XSHG': 19400, ...	20681973.077277	-0.0
783	2015-02-26	5.417277	{u'002153.XSHE': 4400, u'600498.XSHG': 19400, ...	20936634.347277	0.0
784	2015-02-27	5.417277	{u'002153.XSHE': 4400, u'600498.XSHG': 19400, ...	21204596.757277	0.0

```
785 rows × 6 columns
```

轻松跑赢大盘 - 主题Momentum策略

> 来源：https://uqer.io/community/share/551d02c2f9f06c8f33904502

策略原理

在我们的观点中，Momentum这样的思路在主题中也有体现，业绩好的主题会继续保持其上涨的势头，本策略测试了这个思路，具体实现方式为：按20个交易日为一个调仓周期，在调仓日卖出以前所有持仓，买入选出的最好的20个主题中最好的5只股票总共100只股票 本策略的回测参数如下：

- 起始日期： 2014年12月24日
- 结束日期： 2015年4月24日
- 股票池： 3月25日所有活跃主题关联的所有股票
- 业绩基准： 沪深300
- 起始资金： 1000万元
- 调仓周期： 20个交易日

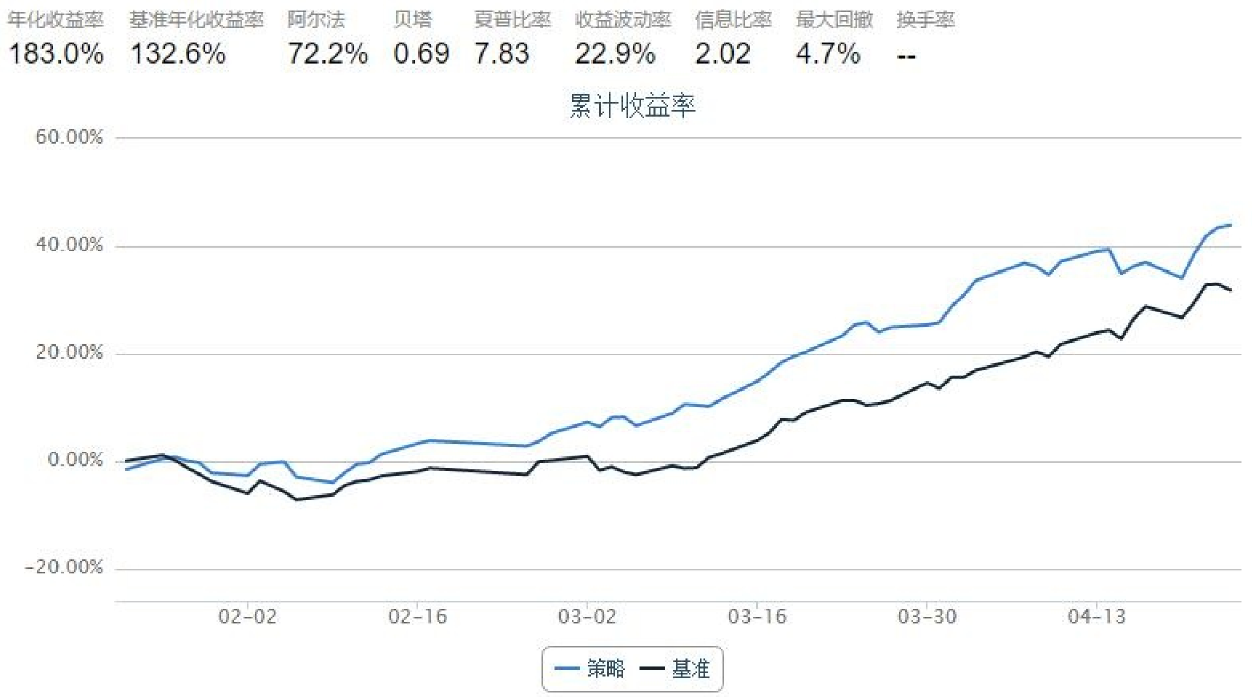
本策略使用的主要数据API有：

- `DataAPI.ActiveThemesGet` 获取某天活跃的主题数据，输入一个日期，获取在该日期活跃的主题。
- `DataAPI.TickersByThemesGet` 获取主题关联的证券

还有很多可以根据基本面调优的办法，读者可以自己尝试

```
from datetime import datetime,timedelta
from heapq import nlargest

#ticker转换为id
tk2id=lambda x: x+'.XSHG' if x[0]=='6'else x+'.XSHE'

#获取3月25日活跃主题对应的所有股票
themeList = DataAPI.ActiveThemesGet('20150325').themeID.tolist()
sa = []
for t in themeList:
    sa += DataAPI.TickersByThemesGet(themeID=str(t)).ticker.toli
st()
sa = list(set(sa))

#以下为回测参数
start = '2014-12-24'
end = '2015-04-24'
```

```python
benchmark = 'HS300'
universe = map(tk2id, sa)
capital_base = 10000000
refresh_rate = 20
longest_history = 20

def initialize(account):
    pass

def handle_data(account):
    # 获取调仓日活跃主题相关股票tickers
    themeList = DataAPI.ActiveThemesGet(account.current_date.strftime('%Y%m%d')).themeID.tolist()
    ta = {}
    for t in themeList:
        ta[t] = DataAPI.TickersByThemesGet(themeID=str(t)).ticker.tolist()

    # 获取过去20个交易日的收盘价
    p = account.get_attribute_history('closePrice', 20)

    #找调仓日内按照等权return加和最大的20个主题
    sa = {}
    for stock in account.universe:
        sa[stock] = p[stock][-1] / p[stock][0] -1 #从调仓日之前20天的return

    tb = {}
    for t in ta:
        tb[t] = sum(sa.get(s,0) for s in ta[t])
    tb = nlargest(20,tb,tb.get)

    # 找这最好的20个主题中最好的5只股票
    sc = []
    for t in tb:
        sc += nlargest(5,[s for s in map(tk2id,ta[t]) if s in sa],sa.get)
    sc = list(set(sc))

    for stock in account.valid_secpos: # 卖出目前所有持有的股票
        order_to(stock, 0)

    for stock in sc: # 买进新选出的100只股票
        order(stock, account.referencePortfolioValue / len(sc) / p[stock][-1])
```



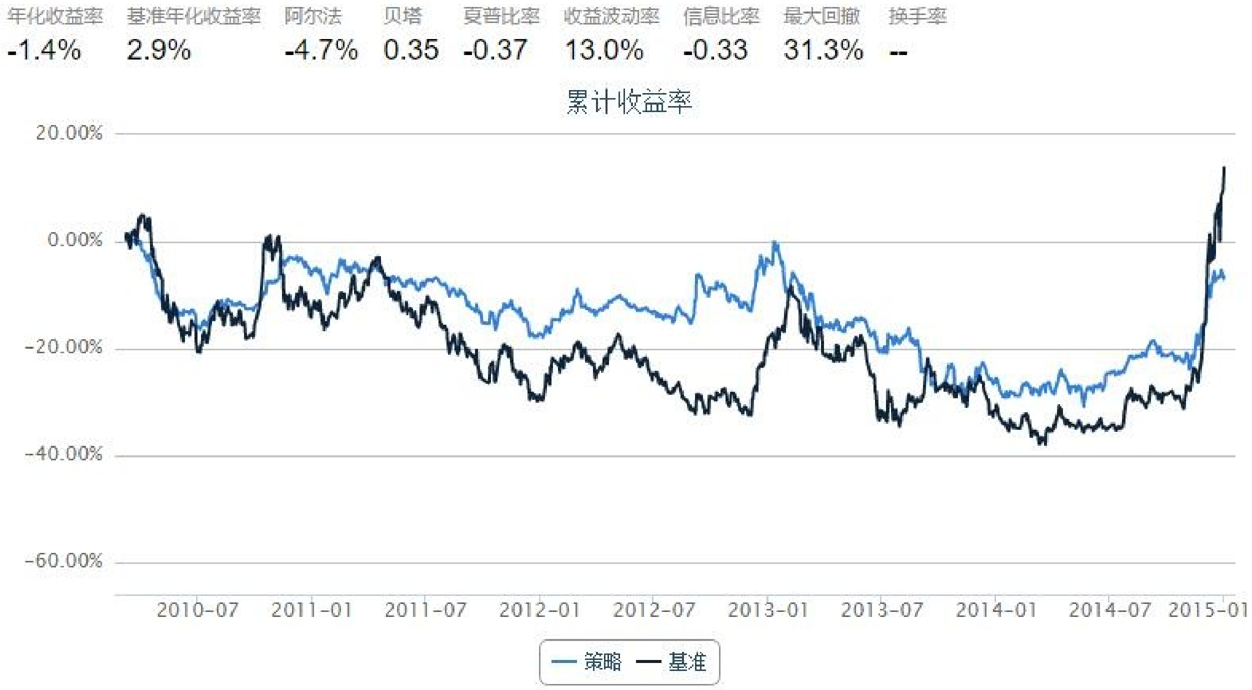
年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
183.0%	132.6%	72.2%	0.69	7.83	22.9%	2.02	4.7%	--

Contrarian strategy

> 来源：https://uqer.io/community/share/5545ff8df9f06c1c3d68802f

Contrarian strategy similar with Momentum strategy

```
import pandas as pd
start = '2010-01-01'                          # 回测起始时间
end = '2015-01-01'                            # 回测结束时间
benchmark = 'SH50'                           # 策略参考标准
universe = set_universe('SH50')
capital_base = 100000                         # 起始资金
longest_history = 40                           # handle_data 函数中
可以使用的历史数据最长窗口长度
refresh_rate = 1                              # 调仓频率，即每 refres
h_rate 个交易日执行一次 handle_data() 函数


def initialize(account):                      # 初始化虚拟账户状态
    pass

def handle_data(account):                     # 每个交易日的买入卖出指令

    returndata = {'symbol':[], 'ret':[]}
    history_data = account.get_attribute_history('closePrice',40
)
    for s in account.universe:
        returndata['symbol'].append(s)
        returndata['ret'].append(history_data[s][-1] / history_d
ata[s][0])
    returndatanew = pd.DataFrame(returndata).sort(columns = 'ret'
).reset_index()
    returndatanew = returndatanew[0:len(returndatanew)/5]
    buylist = returndatanew['symbol'].tolist()

    for cur in account.valid_secpos:
        if cur not in buylist:
            order_to(cur,0)
    for sym in buylist:
        if sym not in account.valid_secpos:
            order_to(sym,300)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
-1.4%	2.9%	-4.7%	0.35	-0.37	13.0%	-0.33	31.3%	--

worse than Momentum strategy

5.2 Joseph Piotroski 9 F-Score Value Investing Model · 基本面选股系统：Piotroski F-Score ranking system

> 来源：https://uqer.io/community/share/56710b1d228e5b8d84f00ac7

```
from CAL.PyCAL import *
import numpy as np
from pandas import DataFrame , Series

start = '2014-01-01'                       # 回测起始时间
end = '2015-01-01'                         # 回测结束时间
benchmark = 'HS300'                        # 策略参考标准
universe = set_universe('HS300')           # 证券池，支持股票和基金
capital_base = 100000                      # 起始资金
csvs = []
security_base = {}
commission = Commission(buycost=0.0008, sellcost=0.0018)  # 佣金万八
slippage = Slippage()
freq = 'd'                                 # 策略类型，'d'表示日间策略使用日线回测，'m'表示日内策略使用分钟线回测
refresh_rate = 1                           # 调仓频率，表示执行handle_data的时间间隔，若freq = 'd'时间间隔的单位为交易日，若freq = 'm'时间间隔为分钟
cal = Calendar('China.SSE')

def initialize(account):                   # 初始化虚拟账户状态
    pass

def handle_data(account):                  # 每个交易日的买入卖出指令


    today = account.current_date.strftime('%Y%m%d')
    yesterday = cal.advanceDate(account.current_date, '-1B', BizDayConvention.Following).strftime('%Y%m%d')
    lastyear = cal.advanceDate(account.current_date, '-1Y', BizDayConvention.Following).strftime('%Y%m%d')

    # 去除ST股
    try:
        STlist = DataAPI.SecSTGet(secID=account.universe, beginDate=yesterday, endDate=yesterday, field=['secID']).tolist()
        account.universe = [s for s in account.universe if s not in STlist]
    except:
        pass
    # 去除流动性差的股票
    tv = account.get_attribute_history('turnoverValue', 20)
    mtv = {sec: sum(tvs)/20. for sec,tvs in tv.items()}
    account.universe = [s for s in account.universe if mtv.get(s
, 0) >= 10**7]

    # 去除新上市或复牌的股票
    opn = account.get_attribute_history('openPrice', 1)
    account.universe = [s for s in account.universe if not (np.i
snan(opn.get(s, 0)[0]) or opn.get(s, 0)[0] == 0)]

    # 调仓部分注意仓位控制，尽量满足80%股票仓位和单只股票不超过10%的条件
    #return

    buylist = []
    selllist = []
    getData_yesterday = Series()
    getData_lastyear = Series()

    #取上一个交易日的数据，用于指标打分
    getData_yesterday = DataAPI.MktStockFactorsOneDayGet(tradeDa
te=yesterday,secID=account.universe,field=['secID','LFLO','ROA',
'OperCashGrowRate','CurrentRatio','DebtEquityRatio','GrossIncome
Ratio','TotalAssetsTRate'],pandas="1")
    getData_yesterday.drop_duplicates('secID', inplace = True)
    getData_yesterday = getData_yesterday.sort('LFLO', ascending=
True)[0:100]
    getData_yesterday.set_index('secID',inplace=True)
    getData_yesterday.dropna(inplace = True)

    #取一年前的数据，用于指标打分
    getData_lastyear = DataAPI.MktStockFactorsOneDayGet(tradeDat
e=lastyear,secID=account.universe,field=['secID','LFLO','ROA','O
perCashGrowRate','CurrentRatio','DebtEquityRatio','GrossIncomeRa
tio','TotalAssetsTRate'],pandas="1")
    getData_lastyear.drop_duplicates('secID', inplace = True)
    getData_lastyear.set_index('secID',inplace=True)
    getData_lastyear.dropna(inplace = True)


    totallist = list(set(getData_yesterday.index)&set(getData_la
styear.index))

    for s in totallist:
        ROA1 = getData_yesterday[s]['ROA']>0
        ROA2 = getData_yesterday[s]['ROA']>getData_lastyear[s]['
ROA']
        OperCashGrowRate = getData_yesterday[s]['CurrentRatio']>0

        CurrentRatio = getData_yesterday[s]['CurrentRatio']>getD
ata_lastyear[s]['CurrentRatio']
        DebtEquityRatio = getData_yesterday[s]['DebtEquityRatio'
]<getData_lastyear[s]['DebtEquityRatio']
```

```
        GrossIncomeRatio = getData_yesterday[s]['GrossIncomeRatio']>getData_lastyear[s]['GrossIncomeRatio']
        TotalAssetsTRate = getData_yesterday[s]['TotalAssetsTRate']>getData_lastyear[s]['TotalAssetsTRate']

        Scores = int(ROA1)+int(ROA2)+int(OperCashGrowRate)+int(CurrentRatio)+int(DebtEquityRatio)+int(GrossIncomeRatio)+int(TotalAssetsTRate)
        if Scores>=6:
            buylist.append(s)

    for s in account.valid_secpos:
        if s not in buylist:
            order_to(s, 0)

    for s in buylist:
        if len(buylist)>=10:
            order(s, account.referencePortfolioValue/len(buylist)/account.referencePrice[s])
        else:
            order_pct_to(s, 0.1)

    for s in account.valid_secpos:
        if account.referencePrice[s] * account.valid_secpos[s] / account.referencePortfolioValue >0.1:
            order_pct_to(s, 0.1)
```

5.3 SVR · 使用SVR预测股票开盘价 v1.0

> 来源：https://uqer.io/community/share/5646f635f9f06c4446b48126

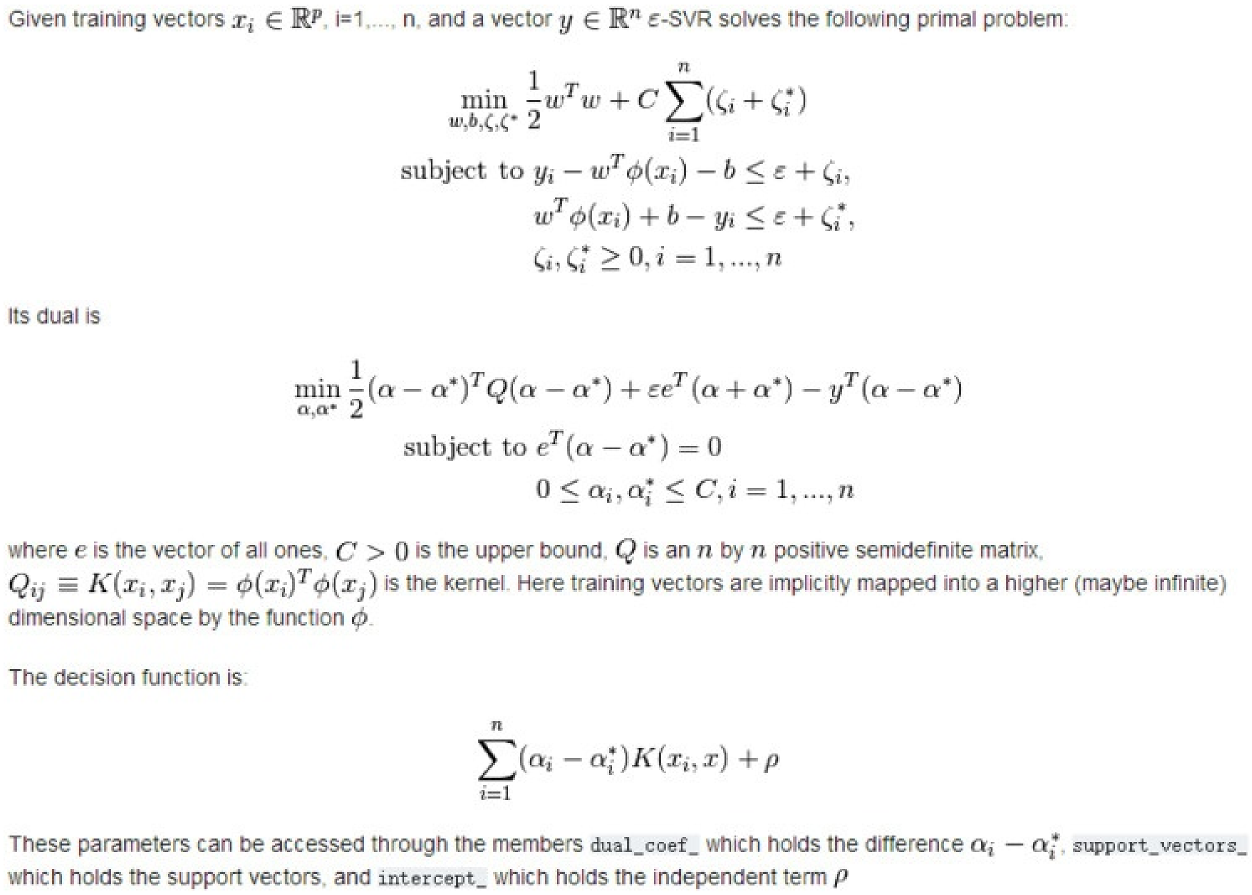
一、策略概述

本策略主旨思想是利用SVR建立的模型对股票每日开盘价进行回归拟合,即把前一日的 `['openPrice','highestPrice','lowestPrice','closePrice','turnoverVol` 作为当日 `'openPrice'` 的自变量，当日 `'openPrice'` 作为因变量。SVR的实现使用第三方库scikit-learn。

二、SVR

SVR详情

SVR参考文献见下方



Given training vectors $x_i \in \mathbb{R}^p$, i=1,..., n, and a vector $y \in \mathbb{R}^n$ ε-SVR solves the following primal problem:

$$\min_{w,b,\zeta,\zeta^*} \frac{1}{2} w^T w + C \sum_{i=1}^{n} (\zeta_i + \zeta_i^*)$$

$$\begin{aligned} \text{subject to } & y_i - w^T \phi(x_i) - b \leq \varepsilon + \zeta_i, \\ & w^T \phi(x_i) + b - y_i \leq \varepsilon + \zeta_i^*, \\ & \zeta_i, \zeta_i^* \geq 0, i = 1, ..., n \end{aligned}$$

Its dual is

$$\min_{\alpha,\alpha^*} \frac{1}{2} (\alpha - \alpha^*)^T Q (\alpha - \alpha^*) + \varepsilon e^T (\alpha + \alpha^*) - y^T (\alpha - \alpha^*)$$

$$\begin{aligned} \text{subject to } & e^T (\alpha - \alpha^*) = 0 \\ & 0 \leq \alpha_i, \alpha_i^* \leq C, i = 1, ..., n \end{aligned}$$

where e is the vector of all ones, $C > 0$ is the upper bound, Q is an n by n positive semidefinite matrix, $Q_{ij} \equiv K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$ is the kernel. Here training vectors are implicitly mapped into a higher (maybe infinite) dimensional space by the function ϕ.

The decision function is:

$$\sum_{i=1}^{n} (\alpha_i - \alpha_i^*) K(x_i, x) + \rho$$

These parameters can be accessed through the members `dual_coef_` which holds the difference $\alpha_i - \alpha_i^*$, `support_vectors_` which holds the support vectors, and `intercept_` which holds the independent term ρ

SVM-Regression

The method of Support Vector Classification can be extended to solve regression problems. This method is called Support Vector Regression.

The model produced by support vector classification (as described above) depends only on a subset of the training data, because the cost function for building the model does not care about training points that lie beyond the margin. Analogously, the model produced by Support Vector Regression depends only on a subset of the training data, because the cost function for building the model ignores any training data close to the model prediction.

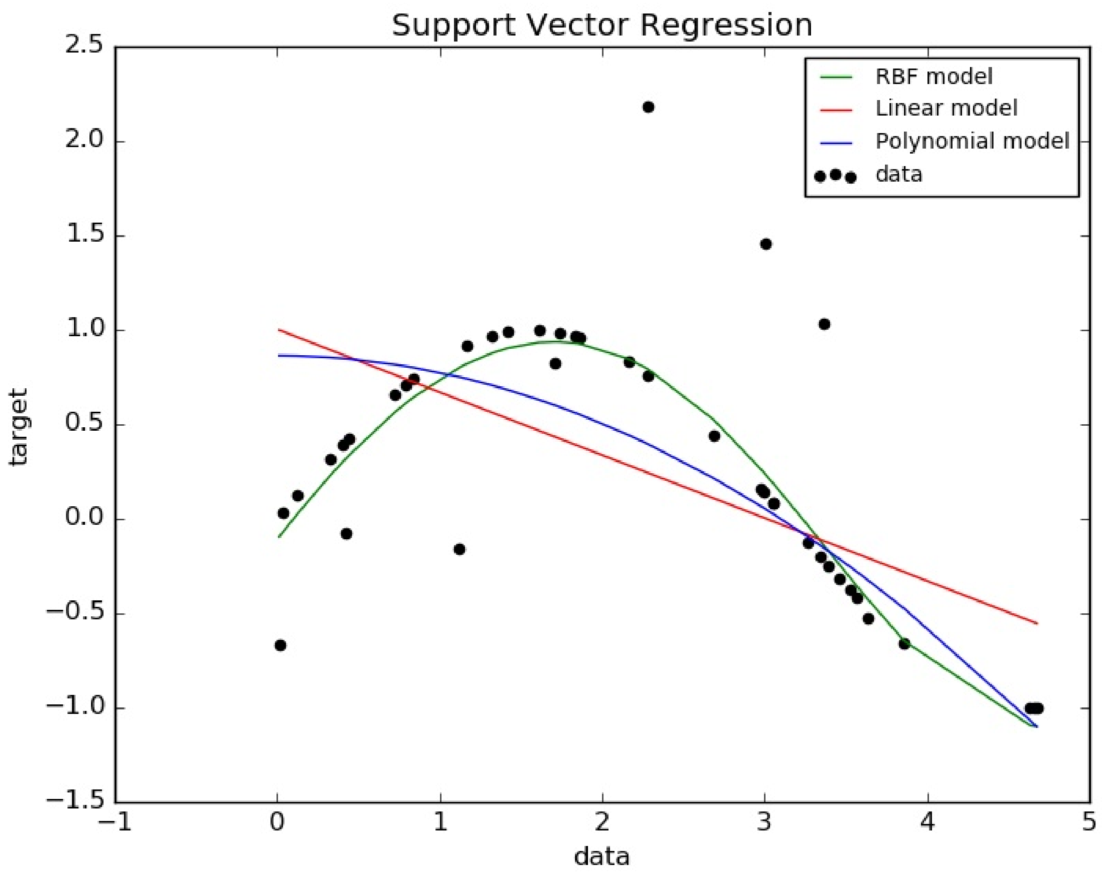
There are three different implementations of Support Vector Regression: SVR, NuSVR and LinearSVR. LinearSVR provides a faster implementation than SVR but only considers linear kernels, while NuSVR implements a slightly different formulation than SVR and LinearSVR.

As with classification classes, the fit method will take as argument vectors X, y, only that in this case y is expected to have floating point values instead of integer values:

```
>>> from sklearn import svm
>>> X = [[0, 0], [2, 2]]
>>> y = [0.5, 2.5]
>>> clf = svm.SVR()
>>> clf.fit(X, y)
SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='auto',
    kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
>>> clf.predict([[1, 1]])
array([ 1.5])
```

Support Vector Regression (SVR) using linear and non-linear kernels:

```
import numpy as np
from sklearn.svm import SVR
import matplotlib.pyplot as plt

################################################################
###############
# Generate sample data
X = np.sort(5 * np.random.rand(40, 1), axis=0)
y = np.sin(X).ravel()

################################################################
###############
# Add noise to targets
y[::5] += 3 * (0.5 - np.random.rand(8))

################################################################
###############
# Fit regression model
svr_rbf = SVR(kernel='rbf', C=1e3, gamma=0.1)
svr_lin = SVR(kernel='linear', C=1e3)
svr_poly = SVR(kernel='poly', C=1e3, degree=2)
y_rbf = svr_rbf.fit(X, y).predict(X)
y_lin = svr_lin.fit(X, y).predict(X)
y_poly = svr_poly.fit(X, y).predict(X)

################################################################
###############
# look at the results
plt.scatter(X, y, c='k', label='data')
plt.plot(X, y_rbf, c='g', label='RBF model')
plt.plot(X, y_lin, c='r', label='Linear model')
plt.plot(X, y_poly, c='b', label='Polynomial model')
plt.xlabel('data')
plt.ylabel('target')
plt.title('Support Vector Regression')
plt.legend()
plt.show()
```



三、PS

原本使用前一天数据预测当天的，但在 Quartz 中，交易策略被具体化为根据一定的规则，判断每个交易日以开盘价买入多少数量的何种股票。回测不影响，但在使模拟盘时无法获取当天的closePrice等，所以将程序改为用地n-2个交易日的数据作为自变量，第n个交易日的openPrice作为因变量。

股票筛选的方法还很欠缺，本程序只用了'去除流动性差的股票'和'净利润增长率大于1的前N支股票'分别进行股票筛选测试，个人感觉都不很理想，还希望大牛们能提供一些有效的筛选方法。

对于股票指数来说，大多数时候都无法对其进行精确的预测，本策略只做参考。

期间发现通过 get_attribute_history 与 DataAPI.MktEqudGet 获取的数据中，有些股票的数据存在一些差异。

关于止损，同样的止损策略，在其他平台可以明显看到，但在Uqer感觉并不起作用，不知是不是代码编写存在错误？还望大牛指正。

程序写的有点乱七八糟的，还望大家见谅，多有不足还望指导！

References:

“A Tutorial on Support Vector Regression” Alex J. Smola, Bernhard Schölkopf - Statistics and Computing archive Volume 14 Issue 3, August 2004, p. 199-222

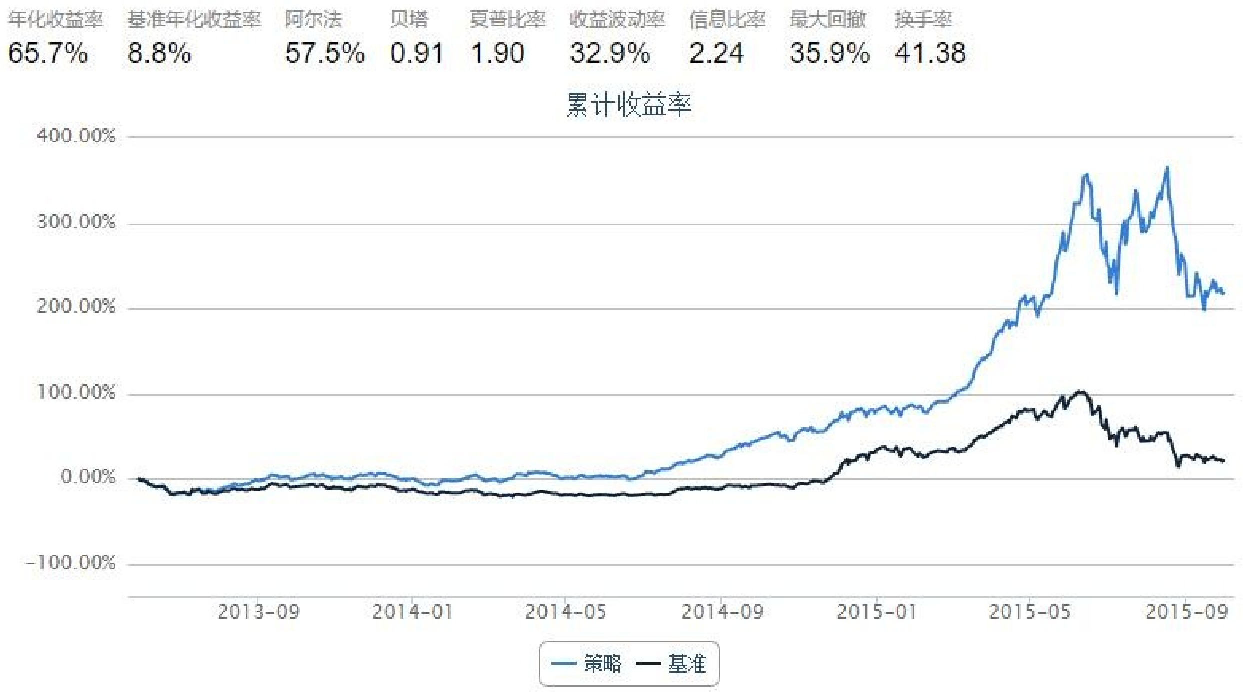
```
# 定义SVR预测函数
def svr_predict(tickerlist,strattime_trainX,endtime_trainX,strat
time_trainY,endtime_trainY,time_testX):
    from sklearn import svm

    # Get train data
    Per_Train_X = DataAPI.MktEqudGet(secID=tickerlist,beginDate=
strattime_trainX,endDate=endtime_trainX,field=['openPrice','high
estPrice','lowestPrice','closePrice','turnoverVol','turnoverValu
e'],pandas="1")
    Train_X = []
    for i in xrange(len(Per_Train_X)):
        Train_X.append(list(Per_Train_X.iloc[i]))

    # Get train label
    Train_label = DataAPI.MktEqudGet(secID=tickerlist,beginDate=
strattime_trainY,endDate=endtime_trainY,field='openPrice',pandas=
"1")
    Train_label = list(Train_label['openPrice'])

    # Get test data
    if len(Train_X) == len(Train_label):

        Per_Test_X = DataAPI.MktEqudGet(secID=tickerlist,tradeDa
te=time_testX,field=['openPrice','highestPrice','lowestPrice','c
losePrice','turnoverVol','turnoverValue'],pandas="1")
        Test_X= []
        for i in xrange(len(Per_Test_X)):
            Test_X.append(list(Per_Test_X.iloc[i]))

        # Fit regression model
        clf = svm.SVR()
        clf.fit(Train_X, Train_label)
        # print clf.fit(Train_X, Train_label)
        PRY = clf.predict(Test_X)
        return '%.2f' %PRY[0]
        # retunr rount(PRY[0],2)
    else:
        pass
```

```
from CAL.PyCAL import *
from heapq import nsmallest
import pandas as pd

start = '2013-05-01'                       # 回测起始时间
end = '2015-10-01'                         # 回测结束时间
benchmark = 'HS300'                        # 策略参考标准
universe =  set_universe('ZZ500') #+ set_universe('SH180')  + se
t_universe('HS300') # 证券池，支持股票和基金
# universe = StockScreener(Factor('LCAP').nsmall(300))  #先用筛选
```

```
器选择出市值最小的N只股票
capital_base = 1000000                     # 起始资金
freq = 'd'                                 # 策略类型，'d'表示日间
策略使用日线回测，'m'表示日内策略使用分钟线回测
refresh_rate = 1                           # 调仓频率，表示执行hand
le_data的时间间隔，若freq = 'd'时间间隔的单位为交易日，若freq = 'm'时间
间隔为分钟
commission = Commission(buycost=0.0008, sellcost=0.0018) # 佣金万八

cal = Calendar('China.SSE')
stocknum = 50

def initialize(account):                   # 初始化虚拟账户状态
    pass

def handle_data(account):                  # 每个交易日的买入卖出指令

    global stocknum

    # 获得日期
    today = Date.fromDateTime(account.current_date).strftime('%Y
%m%d')      # 当天日期
    strattime_trainY = cal.advanceDate(today,'-100B',BizDayConve
ntion.Preceding).strftime('%Y%m%d')
    endtime_trainY = time_testX = cal.advanceDate(today,'-1B',Bi
zDayConvention.Preceding).strftime('%Y%m%d')
    strattime_trainX = cal.advanceDate(strattime_trainY,'-2B',Bi
zDayConvention.Preceding).strftime('%Y%m%d')
    endtime_trainX = cal.advanceDate(endtime_trainY,'-2B',BizDay
Convention.Preceding).strftime('%Y%m%d')
    history_start_time = cal.advanceDate(today,'-2B',BizDayConve
ntion.Preceding).strftime('%Y%m%d')
    history_end_time = cal.advanceDate(today,'-1B',BizDayConvent
ion.Preceding).strftime('%Y%m%d')

    ############################################################
############
    # # 获取当日净利润增长率大于1的前N支股票,由于API的读取数量限制，分批运
行API。
    # getData_today = pd.DataFrame()
    # for i in xrange(300,len(account.universe),300):
    #     tmp = DataAPI.MktStockFactorsOneDayGet(secID=account.u
niverse[i-300:i],tradeDate=today,field=['secID','MA5','MA10','Ne
tProfitGrowRate'],pandas="1")
    #     getData_today = pd.concat([getData_today,tmp],axis = 0)

    # i = (len(account.universe) / 300)*300
    # tmp = DataAPI.MktStockFactorsOneDayGet(secID=account.unive
rse[i:],tradeDate=today,field=['secID','NetProfitGrowRate'],pand
as="1")
    # getData_today = pd.concat([getData_today,tmp],axis = 0)
    # getData_today=getData_today[getData_today.NetProfitGrowRat
e>=1.0].dropna()
```

```
    # getData_today=getData_today.sort(columns='NetProfitGrowRat
e',ascending=False)
    # getData_today=getData_today.head(100)
    # buylist = list(getData_today['secID'])
    ############################################################
###########
    # 去除流动性差的股票
    tv = account.get_attribute_history('turnoverValue', 20)
    mtv = {sec: sum(tvs)/20. for sec,tvs in tv.items()}
    per_butylist = [s for s in account.universe if mtv.get(s, 0)
 >= 10**7]
    bucket = {}
    for stock in per_butylist:
        bucket[stock] = account.referencePrice[stock]
    buylist = nsmallest(stocknum, bucket, key=bucket.get)
    ############################################################
#############

    history = pd.DataFrame()
    for i in xrange(300,len(account.universe),300):
        tmp = DataAPI.MktEqudGet(secID=account.universe[i-300:i]
,beginDate=history_start_time,endDate=history_end_time,field=u"s
ecID,closePrice",pandas="1")
        history = pd.concat([history,tmp],axis = 0)
    i = (len(account.universe) / 300)*300
    tmp = DataAPI.MktEqudGet(secID=account.universe[i:],beginDat
e=history_start_time,endDate=history_end_time,field=u"secID,clos
ePrice",pandas="1")
    history = pd.concat([history,tmp],axis = 0)
    # history = account.get_attribute_history('closePrice', 2)
    # history = DataAPI.MktEqudGet(secID=account.universe,beginD
ate=history_start_time,endDate=history_end_time,field=u"secID,cl
osePrice",pandas="1")
    history.columns = ['secID','closePrice']
    keys = list(history['secID'])
    history.set_index('secID',inplace=True)
    ############################################################
############

    # Sell&止损
    for stock in account.valid_secpos:
        if stock in keys:
            PRY = svr_predict(stock,strattime_trainX,endtime_tra
inX,strattime_trainY,endtime_trainY,time_testX)
            if (PRY < (list(history['closePrice'][stock])[-1]))
or (((list(history['closePrice'][stock])[-1]/list(history['close
Price'][stock])[0])-1) <= -0.05):
                order_to(stock, 0)

    # Buy
    for stock in buylist:
        N = stocknum - len(account.valid_secpos)
        if (stock in keys) and (N > 0):
```

```
        if stock not in account.valid_secpos:
          PRY = svr_predict(stock,strattime_trainX,end
time_trainX,strattime_trainY,endtime_trainY,time_testX)
          if (PRY > list(history['closePrice'][stock])[
-1]):
            amount = (account.cash/N)/account.refere
ncePrice[stock]
            order(stock, amount)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
65.7%	8.8%	57.5%	0.91	1.90	32.9%	2.24	35.9%	41.38

5.4 决策树、随机树

决策树模型（固定模型）

> 来源：https://uqer.io/community/share/568dce2d228e5b18e2ba296e

楼主上学时学的是机器学习，现在在BAT做数据挖掘，一直对将机器学习的知识应用到金融领域比较感兴趣。

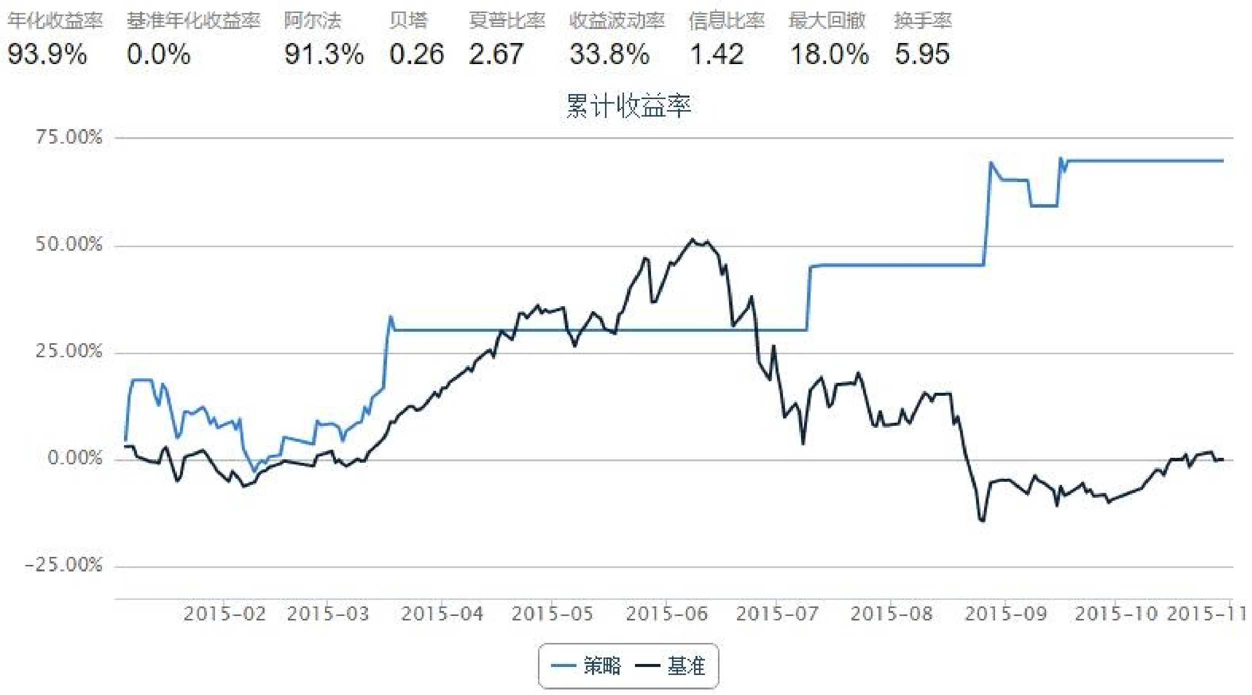
最近发现了优矿这个平台之后，有点着迷了，通过看大家的策略，也学到些知识。

因为楼主对金融投资认识不多，所以写的策略比较简单粗暴，希望向大家多多学习~

策略： 1、不预测具体股价，只预测次日收盘价相比今日是涨是跌； 2、如果预测为涨，则全部买入或持有；如果预测为跌，则全部卖出。

方法： 基于某只股票的历史数据，采用机器学习的方法，挖掘其中规律，预测该只股票次日收盘价是涨还是跌

```
import numpy as np
from CAL.PyCAL import *
from sklearn.cross_validation import train_test_split
from sklearn.externals import joblib
import pandas as pd

cal = Calendar('China.SSE')

# 第一步：设置基本参数
start = '2015-01-01'
end   = '2015-11-01'
capital_base = 1000000
refresh_rate = 1
benchmark = 'HS300'

##HS300
freq = 'd'
#601872.XSHG   HS300
# 第二步：选择主题，设置股票池
universe = ['601872.XSHG', ]


##训练模型
def model_train(begin_date,end_date):

    data1=DataAPI.MktEqudGet(secID=u"601872.XSHG",beginDate=begi
n_date,endDate=end_date,field=['tradeDate','highestPrice','lowes
tPrice','openPrice','closePrice','turnoverVol','turnoverRate'],p
andas="1")

    data2=DataAPI.MktStockFactorsDateRangeGet(secID=u"601872.XSH
G",beginDate=begin_date,endDate=end_date,field=['tradeDate','DAV
OL5','EMA5','EMA10','MA5','MA20','RSI','VOL5','VOL10','MACD'],pa
ndas="1")

    df_data=pd.merge(data1,data2,on='tradeDate')

    tmp=[]
    for i in range(len(df_data.values)):
        mark_1=0
        for j in range(len(df_data.values[i])):
            if str(df_data.values[i][j])=='nan':
                mark_1=1
        if mark_1==0:
            a=list(df_data.values[i])
            a.append(df_data.values[i][4]-df_data.values[i][10])
            a.append(df_data.values[i][4]-df_data.values[i][11])
            tmp.append(a)
    data=tmp
    print len(data)
    x=[]
    y=[]
    for i in range(len(data)-1):
        if data[i][4]<data[i+1][4]:
            y.append(1)
        else:
            y.append(0)
        x.append(data[i][1:])

    x_train, x_test, y_train, y_test = train_test_split(x, y, te
st_size=0.0, random_state=42)

    ##训练模型
    from sklearn import tree
    clf = tree.DecisionTreeClassifier( max_depth =3 )
    clf.fit(x_train,y_train)
    y_predict=clf.predict(x_train)
    n_1=0
    for i in range(len(y_predict)):
        if y_train[i]==y_predict[i]:
            n_1=n_1+1
    n_2=0
    for i in range(len(y_predict)):
        if y_train[i]==y_predict[i] and y_predict[i]==1:
            n_2=n_2+1
    joblib.dump(clf, 'clf.model')
    return clf,float(n_1)/float( len(y_predict) ),float(n_2)/flo
at( int(sum(y_train)) ) ,float(sum(y_train))/float(len(y_train))

def initialize(account):
    ##使用2015年2月1日之前800个交易日的数据进行训练
    today='20150201'
    train_begin_date = cal.advanceDate(today,'-800B',BizDayConve
ntion.Preceding).strftime('%Y%m%d')
    train_end_date = cal.advanceDate(today,'-1B',BizDayConventio
n.Preceding).strftime('%Y%m%d')

    model,acc_rate,recall_rate,balance=model_train(train_begin_d
ate,train_end_date)
    print acc_rate,recall_rate,balance  ##正确率、召回率、正负样本均
衡度

def handle_data(account):
    # 本策略将使用account的以下属性：
    # account.referencePortfolioValue表示根据前收计算的当前持有证券市
场价值与现金之和。
    # account.universe表示当天，股票池中可以进行交易的证券池，剔除停牌退
市等股票。
    # account.referencePrice表示股票的参考价，一般使用的是上一日收盘价。

    # account.valid_secpos字典，键为证券代码，值为虚拟账户中当前所持有该
股票的数量。

    c = account.referencePortfolioValue

    today = account.current_date.strftime('%Y-%m-%d')

    begin_date = cal.advanceDate(today,'-1B',BizDayConvention.Pr
eceding).strftime('%Y%m%d')
    end_date = cal.advanceDate(today,'-1B',BizDayConvention.Prec
eding).strftime('%Y%m%d')

    data1=DataAPI.MktEqudGet(secID=u"601872.XSHG",beginDate=begi
n_date,endDate=end_date,field=['tradeDate','highestPrice','lowes
tPrice','openPrice','closePrice','turnoverVol','turnoverRate'],p
andas="1")

    data2=DataAPI.MktStockFactorsDateRangeGet(secID=u"601872.XSH
G",beginDate=begin_date,endDate=end_date,field=['tradeDate','DAV
OL5','EMA5','EMA10','MA5','MA20','RSI','VOL5','VOL10','MACD'],pa
ndas="1")

    df_data=pd.merge(data1,data2,on='tradeDate')

    a=list(df_data.values[0])
    a.append(df_data.values[0][4]-df_data.values[0][10])
    a.append(df_data.values[0][4]-df_data.values[0][11])

    x_predict=a[1:]

    for i in range(len(x_predict)):
        if str(x_predict[i])=='nan':
            x_predict[i]=10000000

    clf = joblib.load('clf.model')
    y_predict=clf.predict(x_predict)
    # 计算调仓数量
    change = {}
    for stock in account.universe:
        if y_predict>0 and stock not in account.valid_secpos:
            p = account.referencePrice[stock]
            order(stock,int(c / p))
        if y_predict==0 and stock in account.valid_secpos:
            order_to(stock,0)
    #print today,x_predict[3],y_predict
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
93.9%	0.0%	91.3%	0.26	2.67	33.8%	1.42	18.0%	5.95

```
713
0.580056179775 0.334384858044 0.445224719101
```

This is an empty markdown cell

基于Random Forest的决策策略

> 来源：https://uqer.io/community/share/54a10ef8f9f06c4bb886324b

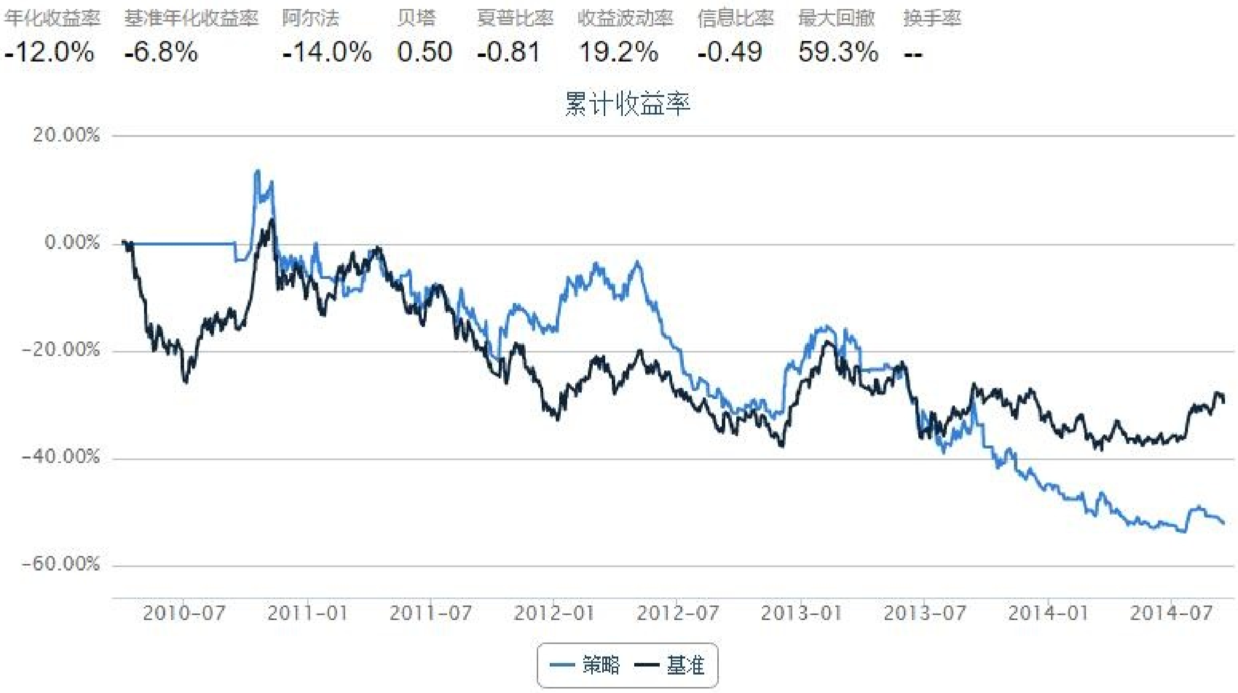
版本：1.0

作者：李丞

联系：cheng.li@datayes.com

利用随机树分类算法，通过历史价格的上升状态变化规律，预测下一日股价变动的方向。预测上涨则买入，下跌则卖出（如果可以的话）；

```
from sklearn.ensemble import RandomForestClassifier
from collections import deque
import pandas as pd
import numpy as np

start = pd.datetime(2010, 4, 1)
end   = pd.datetime(2014, 9, 16)
longest_history = 1

bm = 'HS300'

universe = ['600000.XSHG']
csvs = []

capital_base = 1e5
refresh_rate = 1
window_length = 10

def initialize(account):
    account.security = universe[0]
    account.window_length = window_length

    account.classifier = RandomForestClassifier()

    # 先进先出的deque序列，设定了最长的长度，在序列超过最长长度的时候，会将头部序列移出
    account.recent_prices = deque(maxlen=account.window_length+2) # 保存最近的股价
    account.X = deque(maxlen=100) # 自变量
    account.Y = deque(maxlen=100) # 应变量

    account.prediction = 0 # 保存最近的预测值

def handle_data(account):
    hist = account.get_history(1)
    if account.security in hist:
        account.recent_prices.append(hist[account.security]['clo
sePrice'][0]) # 更新最近的股价
    if len(account.recent_prices) >= account.window_length+2
: # 如果我们已经获取了足够的股价
      RecentPrice=list(account.recent_prices)   # 将deque转
换为对应的list
      # 制作一组1和0，标记股价是否相对于上一日价格上升。
      changes = np.diff(RecentPrice) > 0

      account.X.append(RecentPrice[1:-1])
      account.Y.append(changes[-1])

      if len(account.Y) >= 100: # 已经拥有足够的数据im

        account.classifier.fit(account.X, account.Y) #
设定模型

        account.prediction = account.classifier.predict(
changes[1:]) # 预测

        # 如果过大0.5，买入；小于0.5，卖出

        if  account.prediction > 0.5:
          buyAmount = int(account.position.cash / hist
[account.security]['closePrice'][0])
          order(account.security, buyAmount)
        else:
          order_to(account.security, 0)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
-12.0%	-6.8%	-14.0%	0.50	-0.81	19.2%	-0.49	59.3%	--

5.5 钟摆理论 · 钟摆理论的简单实现——完美躲过股灾和精准抄底

今天给大家简单介绍一种稳健的投资体系——钟摆理论的量化模型实现。这里要感谢@进化论一平 的雪球分享：http://xueqiu.com/8510627167/29759691

其中的核心思想是：（1）从买入操作来看：通过找到有效价格区间，要求在价格低估时买入，但这同时又要满足另一个条件，那就是趋势必须向上。二者缺一不可。（2）从卖出操作来看：必须顺势而为，在价格超过有效价格区间以后，如果趋势不变，不要急于卖出，直到趋势改变，价格高估再卖出。二者同样缺一不可。

那么核心问题就是两个：（1）如何找到有效的价格区间？也就是，如何给出个股的估值？（2）怎样判断趋势？

为了尽量追求简单，避免太复杂的优化。我这里直接给出两个问题的简单判定方法。

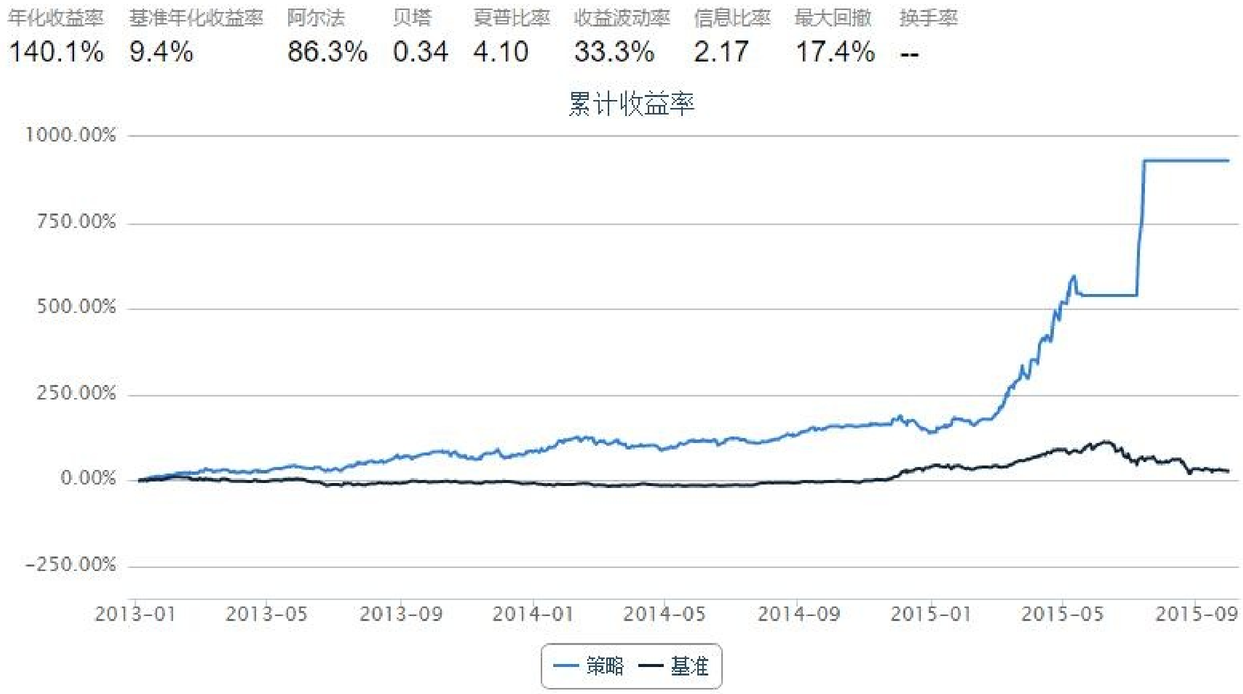
（1）根据格雷厄姆的成长价值公式进行估值，并且根据A股的实际情况或者市场情绪给予一定溢价或者折价。价值=当期(正常)利润×(8.5 + 两倍的预期年增长率)，其中的当期利润使用每股收益EPS进行衡量，预期年增长率使用EGRO/5表示，其中EGRO的计算方法为5年收益关于时间（年）进行线性回归的回归系数/5年收益均值的绝对值

（2）判断趋势有两种途径结合，一种是趋势已经向上，比较简单判断方法是五日线在十日线之上（这种判断方法犯错的几率较大，读者可以自行改进），另外一种是趋势由下向上逆转，即出现明显的底部形态。关于后者，我给出的判断标准为：股价相对于近期高点大幅下跌超过downPercent（例如30%），并且收盘价在五日线十日线之下，并且收红或者收星，跌幅小于7%

接下来就是具体实现了。

```
def preceding_date(date):
    cal = DataAPI.TradeCalGet(exchangeCD=u"XSHG",beginDate='2011
0101',endDate=date,field=['calendarDate','isOpen'],pandas="1")
    cal = cal[cal['isOpen']==1]
    date = cal['calendarDate'].values[-2].replace('-','')
    return date

def duotou_5_10(date, stockList, precedingDate=True):
    if precedingDate:
        date = preceding_date(date)
    duotou = {}
    if stockList is None or len(stockList) == 0:
        return duotou
    kLine = DataAPI.MktStockFactorsOneDayGet(tradeDate=date,secI
D=stockList,field=['secID','MA5','MA10'],pandas="1")
    kLine = kLine.dropna()
    for stock, ma5, ma10 in zip(kLine['secID'].values, kLine['MA
5'].values, kLine['MA10'].values):
        if ma5 > ma10:
            duotou[stock] = True
        else:
            duotou[stock] = False
    return duotou

def spreadRateByIntrinsicValue(account, overflow=0.0, precedingD
ate=True):
    stock_list = account.universe
    current_date = account.current_date
    date = current_date.strftime('%Y%m%d')
    if precedingDate:
        date = preceding_date(date)
    eq_EPS_EGRO = DataAPI.MktStockFactorsOneDayGet(tradeDate=dat
e,secID=stock_list,field=['secID','EPS','EGRO'],pandas="1")
    eq_EPS_EGRO['Value'] = eq_EPS_EGRO['EPS']*(8.5+2*eq_EPS_EGRO[
'EGRO']/5)
    eq_EPS_EGRO = eq_EPS_EGRO.dropna()
    spread_rate = []
    for stock, intrinsic_value in zip(eq_EPS_EGRO['secID'].value
s, eq_EPS_EGRO['Value'].values):
        intrinsic_value = intrinsic_value*(1+overflow)
        reference_price = account.referencePrice[stock]
        if reference_price > 0 and reference_price < intrinsic_v
alue:
            spread_rate.append((stock, (intrinsic_value-referenc
e_price)/reference_price))
    return sorted(spread_rate, key=lambda k: k[-1], reverse=True
)

'''
判断是否为底部形态，判断标准为股价相对于近期高点大幅下跌超过downPercent，并
且收盘价在五日线十日线之下，并且收红或者收星，跌幅小于7%
'''
def isButtom(date, stockList, precedingDate=True, downPercent=0.3
):
    cal = DataAPI.TradeCalGet(exchangeCD=u"XSHG",beginDate='2011
0101',endDate=date,field=u"prevTradeDate",pandas="1")
    daysAhead = cal['prevTradeDate'].values[-20].replace('-','')
    if precedingDate:
        date = cal['prevTradeDate'].values[-1].replace('-','')
    rs = {}
    if stockList is None or len(stockList) == 0:
        return rs
    dayInfo = DataAPI.MktEqudAdjGet(secID=stockList, beginDate=d
aysAhead, endDate=date ,field=['secID', 'openPrice', 'closePrice'
, 'preClosePrice'],pandas="1")
    dayInfo.dropna()
    for stock in stockList:
        stockDayInfo = dayInfo[dayInfo['secID']==stock]
        closePrices = stockDayInfo['closePrice'].values
        ma5 = np.mean(closePrices[-5:])
        ma10 = np.mean(closePrices[-10:])
        closePrice = closePrices[-1]
        maxClosePrice = np.max(closePrices)
        openPrice = stockDayInfo['openPrice'].values[-1]
        preClosePrice = stockDayInfo['preClosePrice'].values[-1]
        if (maxClosePrice-closePrice)/maxClosePrice > downPercent and closePrice < ma5 and ma5 < ma10 and (closePrice > openPrice or abs(closePrice-openPrice)/openPrice < 0.02) and abs(closePrice-preClosePrice)/preClosePrice<0.07:
            rs[stock] = True
        else:
            rs[stock] = False
    return rs
```

```
import numpy as np

start = '2013-01-01'                       # 回测起始时间
end = '2015-10-01'                         # 回测结束时间
benchmark = 'HS300'                       # 策略参考标准
commission = Commission(buycost=0.0008, sellcost=0.0018)  # 佣金万八
universe = set_universe('CYB',date=end)           # Very Important Here!! 选股很重要！不要玩大烂臭！估值再低也别玩！
capital_base = 1000000                      # 起始资金
freq = 'd'                                 # 策略类型，'d'表示日间策略使用日线回测，'m'表示日内策略使用分钟线回测
refresh_rate = 1                           # 调仓频率，表示执行handle_data的时间间隔，若freq = 'd'时间间隔的单位为交易日，若freq = 'm'时间间隔为分钟

max_percent_of_a_stock = 1.0 # 单支股的最大仓位

def initialize(account):                   # 初始化虚拟账户状态
    pass

def handle_data(account):                  # 每个交易日的买入卖出指令

    global max_percent_of_a_stock
    buylist = []
    selist = []
    current_date = account.current_date
    current_date = current_date.strftime('%Y%m%d')

    overflow = 0.15 # 根据情况给予一定的溢价（例如0.1)或者折价(例如-0.1)，也可以根据市场风险程度进行动态调节（此处读者可以自行发挥）

    spread_rate = dict(spreadRateByIntrinsicValue(account, overflow=overflow, precedingDate=True))

    referencePortfolioValue = account.referencePortfolioValue
    # 获取用来计算多头形态的股票列表
    stock_set_for_duotou = []
    stock_set_for_duotou.extend(account.avail_secpos.keys())
    stock_set_for_duotou.extend(spread_rate.keys())
    stock_set_for_duotou = list(set(stock_set_for_duotou))

    duotou_5_10_Map = duotou_5_10(current_date, stock_set_for_du
otou, precedingDate=True)
    isButtom_Map = isButtom(current_date, stock_set_for_duotou,
precedingDate=True, downPercent=0.3)

    for stock in account.avail_secpos.keys():
        if stock not in spread_rate and not duotou_5_10_Map.get(
stock, False):
            selist.append(stock)

    for stock in selist:
        sell_value = account.referencePrice[stock]*account.valid
_secpos[stock]
        order_to(stock, 0)

    for stock in spread_rate.keys():
        if stock not in account.valid_secpos:
            buylist.append(stock)

    for stock in buylist:
        # 满足以下条件之一买入：（1）5日线在10日线之上；（2）出现底部特征
        if duotou_5_10_Map.get(stock, False) or isButtom_Map.get
(stock, False):
            buy_value = min(referencePortfolioValue*max_percent_
of_a_stock, account.cash/len(buylist))
            if buy_value/referencePortfolioValue >= 0.0001:
                order_pct(stock, buy_value/referencePortfolioVal
ue)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
140.1%	9.4%	86.3%	0.34	4.10	33.3%	2.17	17.4%	--

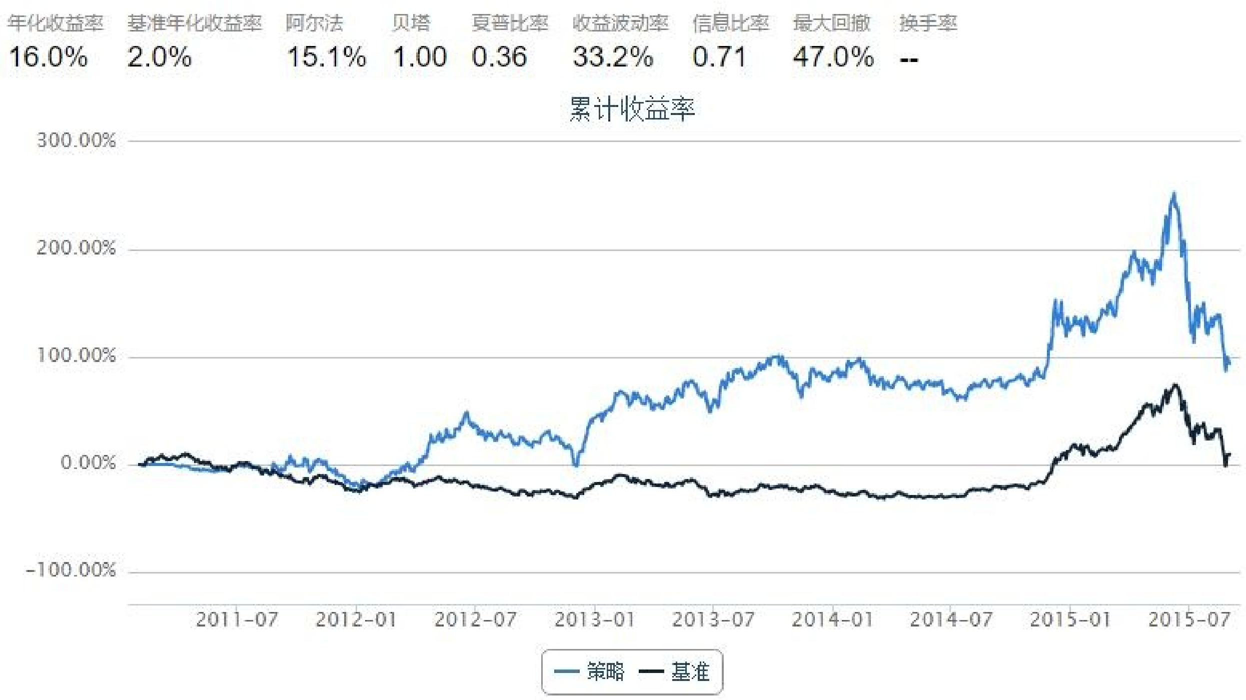
回测可以发现，这种投资体系能够完美躲过股灾，并且能在股灾中精准抄底获利：-)

5.6 海龟模型

simple turtle

> 来源：https://uqer.io/community/share/55fe8f58f9f06c597165ef13

```
start = '2011-01-01'                       # 回测起始时间
end = '2015-09-01'                         # 回测结束时间
benchmark = 'HS300'                        # 策略参考标准
universe = set_universe('HS300')  # 证券池，支持股票和基金
capital_base = 100000                      # 起始资金
freq = 'd'                                 # 策略类型，'d'表示日间策略使用日线回测，'m'表示日内策略使用分钟线回测
refresh_rate = 1                           # 调仓频率，表示执行handle_data的时间间隔，若freq = 'd'时间间隔的单位为交易日，若freq = 'm'时间间隔为分钟
longest_history=60
pos_pieces=10
window=20

def initialize(account):                   # 初始化虚拟账户状态
    pass

def handle_data(account):                  # 每个交易日的买入卖出指令

    highest_price=account.get_attribute_history('highPrice',window)
    lowest_price=account.get_attribute_history('lowPrice',window)
    for stock in account.universe:
        current_price=account.referencePrice[stock]
        if current_price > highest_price[stock].max() and account.position.secpos.get(stock,0)==0:
            order_to(stock,capital_base/pos_pieces/current_price)
        elif current_price < lowest_price[stock].min():
            order_to(stock,0)
    return
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
16.0%	2.0%	15.1%	1.00	0.36	33.2%	0.71	47.0%	--

侠之大者 一起赚钱

> 来源：https://uqer.io/community/share/554048dff9f06c1c3d687fa5

在阔别七年的又一轮牛市里,炒股已经成为人们每天讨论的话题.

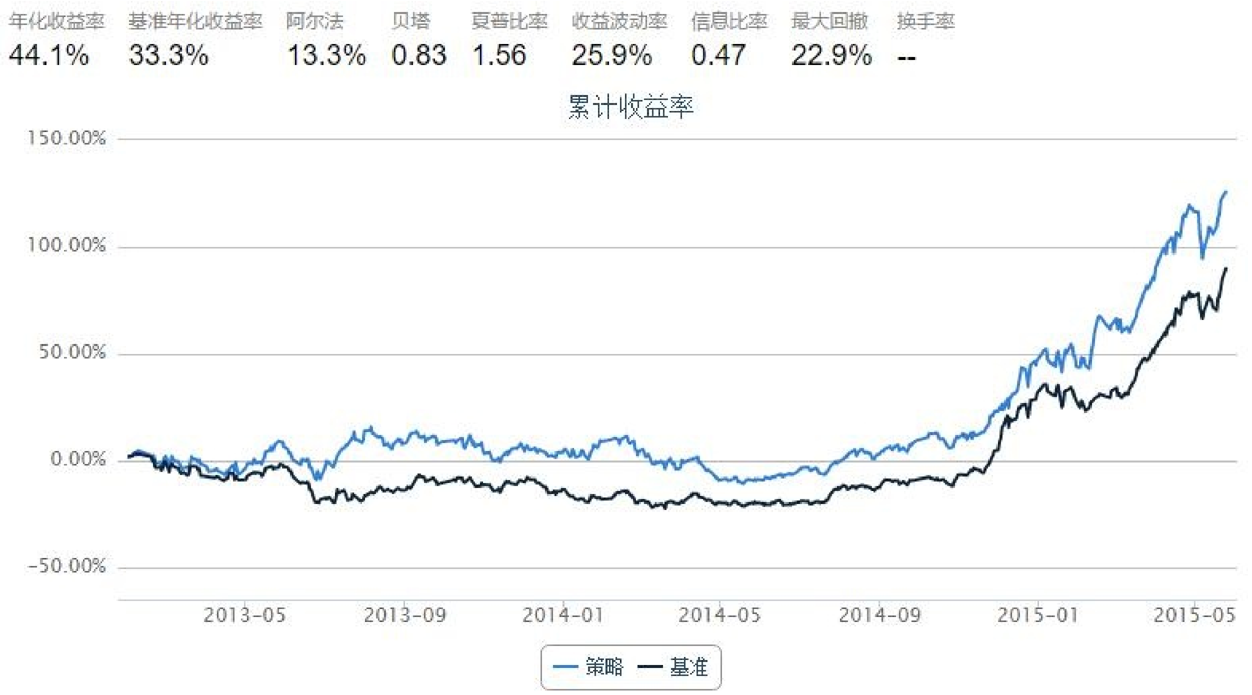
小老弟一直以为:"侠之大者,一起赚钱,一起嗨". 故借宝地献出珍藏多年的交易秘籍.

首先讲述一下策略思路:

- 标的: 流通性较好,深受大妈喜爱的沪深300成分股, 乃策略标的最佳选择.
- 买卖点: 追涨杀跌是本策略的核心思路. 在股价,成交量向上突破最近20日最高价格(量)时买入. 在股价向下突破最近10日最低价格卖出.
- 头寸规模：每只股票最多占1/10仓位.

话不多说, 上代码:

```
start = datetime(2013, 1, 1)
end   = datetime(2015, 5, 25)
benchmark = 'HS300'
universe  = set_universe('HS300')
capital_base = 100000

pos_pieces = 10
enter_window = 20
exit_window = 10

def initialize(account):
    pass

def handle_data(account):
    highest_price = account.get_attribute_history('highPrice', enter_window)
    lowest_price  = account.get_attribute_history('lowPrice', exit_window)
    close_price  = account.get_attribute_history('closePrice', exit_window)
    turnover_vol = account.get_attribute_history('turnoverVol', enter_window)
    for stock in account.universe:
        cnt_price = close_price[stock][-1] #account.referencePrice[stock]
        cnt_turnover = turnover_vol[stock][-1]
        if cnt_price > highest_price[stock][:-1].max() and cnt_turnover > turnover_vol[stock][:-1].max() and account.position.secpos.get(stock, 0)==0:
            order(stock, capital_base/pos_pieces/cnt_price)
        elif cnt_price < lowest_price[stock][:-1].min():

            order_to(stock, 0)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
44.1%	33.3%	13.3%	0.83	1.56	25.9%	0.47	22.9%	--

也许已经有人发现，其实这就是海龟交易系统.

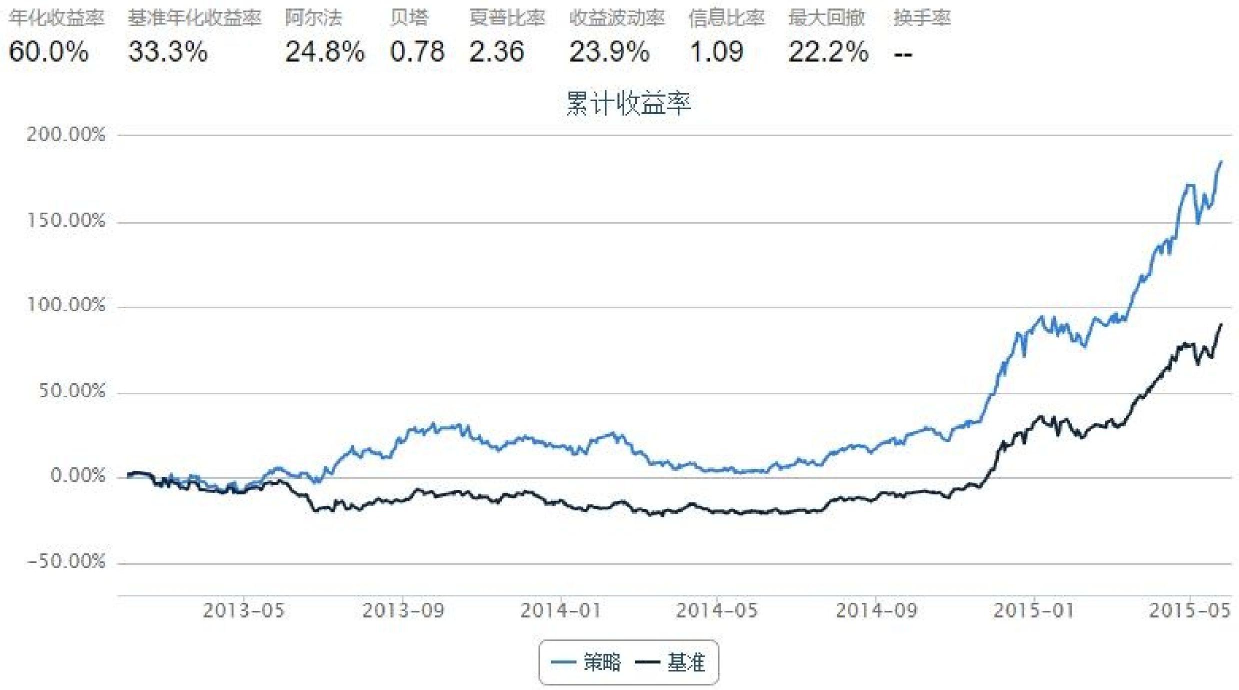
海龟交易系统是一个完整的交易系统,它有一个完整的交易系统所应该有的所有成分，涵盖了成功交易中的每一个必要决策：

- 市场：买卖什么？
- 头寸规模：买卖多少？
- 入市：什么时候买卖？
- 止损：什么时候放弃一个亏损的头寸？
- 退出：什么时候退出一个盈利的头寸？
- 战术：怎么买卖？

在上面的策略中，每只股票的头寸规模为1/10的初始资金.

《海龟交易法则》介绍了一种头寸规模控制方法，将头寸分为一个个单位，下面的策略将展示将头寸分为N个单位，每次产生买入信号时，仅买入一个单位.

```
start = datetime(2013, 1, 1)
end   = datetime(2015, 5, 25)
benchmark = 'HS300'
universe  = set_universe('HS300')
capital_base = 100000

pos_pieces = 10
enter_window = 20
exit_window = 10
N = 4

def initialize(account):
    account.postion_size_hold = {}
    for stk in universe:
        account.postion_size_hold[stk] = 0

def handle_data(account):
    highest_price = account.get_attribute_history('highPrice', enter_window)
    lowest_price  = account.get_attribute_history('lowPrice', exit_window)
    close_price  = account.get_attribute_history('closePrice', exit_window)
    turnover_vol = account.get_attribute_history('turnoverVol', enter_window)
    for stock in account.universe:
        cnt_price = close_price[stock][-1] #account.referencePrice[stock]
        cnt_turnover = turnover_vol[stock][-1]
        if cnt_price > highest_price[stock][:-1].max() and cnt_turnover > turnover_vol[stock][:-1].max() and account.postion_size_hold[stock]<N:
            order(stock, capital_base/pos_pieces/cnt_price/N)
            account.postion_size_hold[stock] += 1
        elif cnt_price < lowest_price[stock][:-1].min():
            order_to(stock, 0)
            account.postion_size_hold[stock] = 0
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
60.0%	33.3%	24.8%	0.78	2.36	23.9%	1.09	22.2%	--

我们发现回撤和波动率有所下降,而收益率竟然上升了. 其实原因很简单, 分N次买入时, 如果信号正确, 可能会提高一定的持仓成本,降低收益率; 反之如果信号有误, 也能够快速止损, 减少回撤.

也就是说, 头寸规模有效的控制了风险.

以上两个策略属于唐安奇趋势系统, 结束之前, 再介绍一下布林格突破系统.

布林线定义：

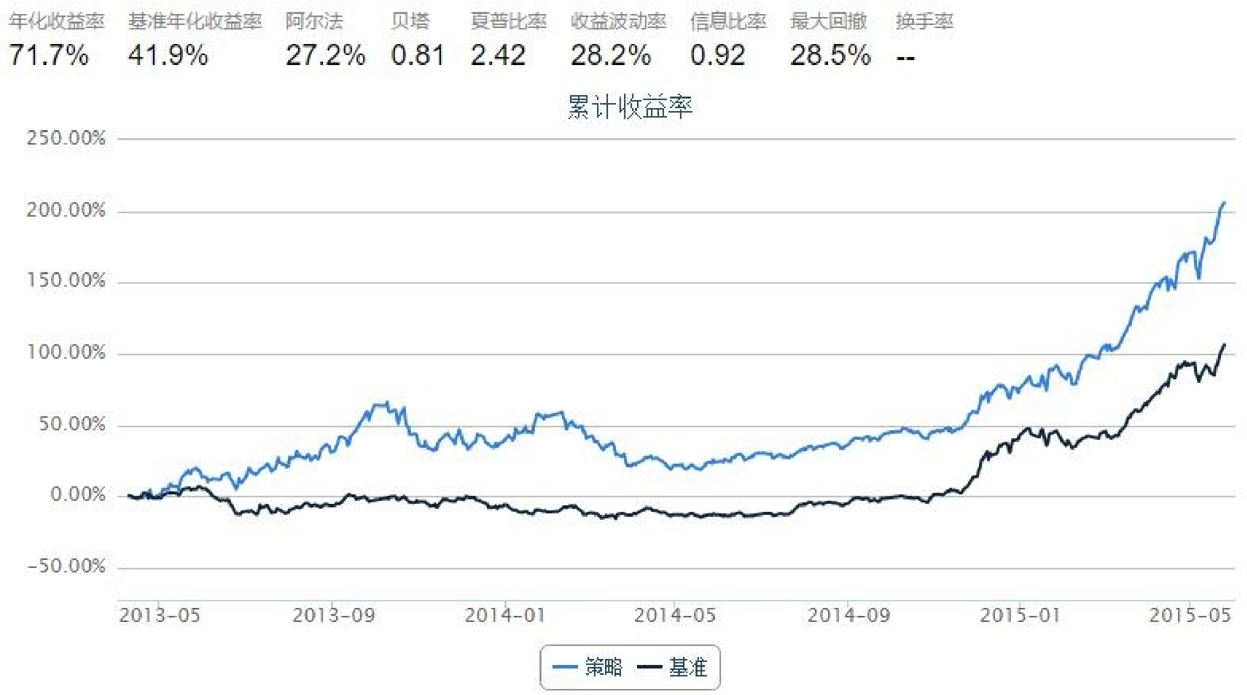
布林线是通过350日平均收盘加减2.5倍标准差得到的。

布林线方法：

- 如果前一日的收盘价穿越了通道的顶部，则开盘做多
- 如果前一日的收盘价跌破了通道的底部，则开盘做空

在我们的这个股票策略里,我们以60日平均收盘加减2.5倍标准差作为波幅通道.

```
import numpy as np
start = datetime(2013, 1, 1)
end   = datetime(2015, 5, 25)
benchmark = 'HS300'
universe  = set_universe('HS300')
capital_base = 100000
longest_history = 60

pos_pieces = 10
enter_window = 20
exit_window = 10
N = 4

def initialize(account):
    account.postion_size_hold = {}
    for stk in universe:
        account.postion_size_hold[stk] = 0

def handle_data(account):
    close_prices = account.get_attribute_history('closePrice', longest_history)
    for stock in account.universe:
        cnt_price = close_prices[stock][-1] #account.referencePrice[stock]
        mean_cp = close_prices[stock].mean()
        bias = 2.5*np.std(close_prices[stock])
        high_channel = mean_cp + bias
        low_channel = mean_cp - bias
        if cnt_price >= high_channel and account.postion_size_hold[stock]<N:
            order(stock, capital_base/pos_pieces/cnt_price/N)
            account.postion_size_hold[stock] += 1
        elif cnt_price <= low_channel:
            order_to(stock, 0)
            account.postion_size_hold[stock] = 0
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
71.7%	41.9%	27.2%	0.81	2.42	28.2%	0.92	28.5%	--

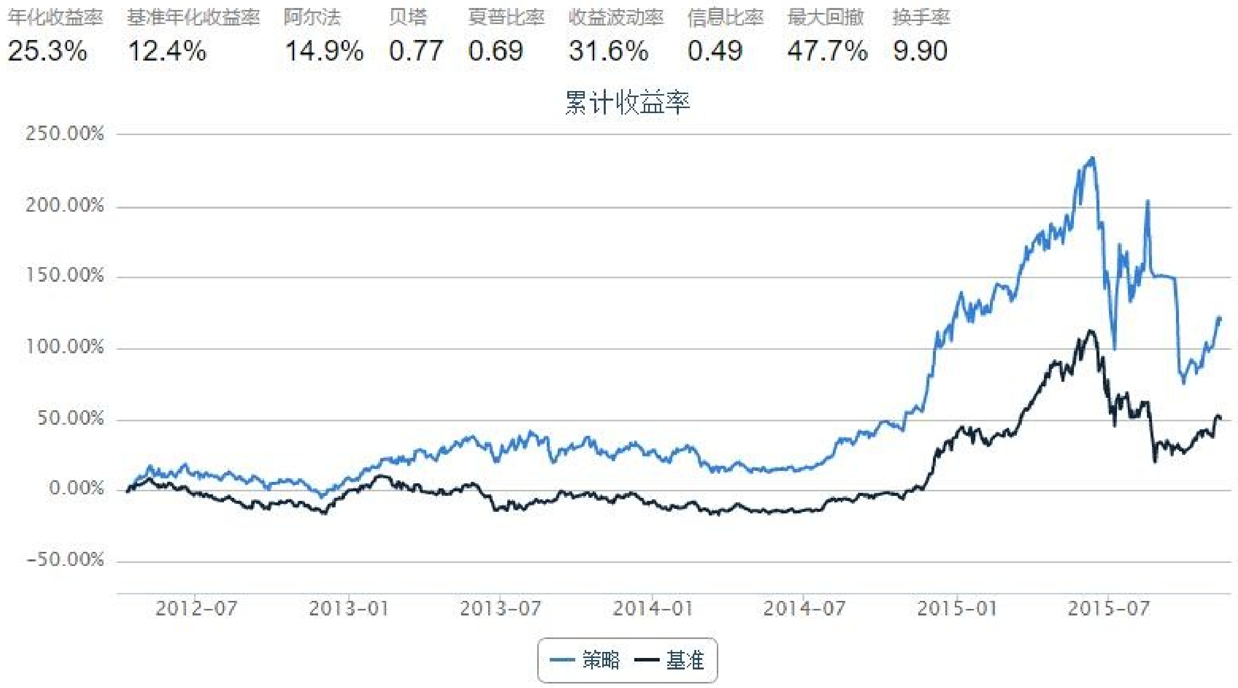
参考自:《海龟交易法则》 作者: 柯蒂斯·费思

5.7 5217 策略 · 白龙马的新手策略

> 来源：https://uqer.io/community/share/56458dbcf9f06c4446b480ec

- 买入信号 ：价格创出60个交易日新高，第二天买入
- 卖出信号 ：价格从最高点下跌17%，则第二天卖出
- 卖出金额平均分配到新高的股价上

```
import numpy as np
from datetime import datetime, timedelta

start = '20120101'                        #  回测起始时间
end = (datetime.today() - timedelta(days=1)).strftime('%Y%m%d')
#  截止日期
benchmark = 'HS300'                         #  策略参考标准
universe = set_universe('HS300')
capital_base = 1000000                      #  起始资金
freq = 'd'                                  #  策略类型，'d'表示日间
策略使用日线回测，'m'表示日内策略使用分钟线回测
refresh_rate = 1                            #  调仓频率，表示执行hand
le_data的时间间隔，若freq = 'd'时间间隔的单位为交易日，若freq = 'm'时间
间隔为分钟

MAX_PRICE = {}

def initialize(account):                    #  初始化虚拟账户状态
    pass

def handle_data(account):                   #  每个交易日的买入卖出指令


    today = account.current_date.strftime('%Y%m%d')
    hist = account.get_attribute_history('closePrice' , 60)
    cash = account.cash
    buylist = []

    #记录持仓股票的最高价（卖出判断指标）
    for s in account.valid_secpos :
        MAX_PRICE[s] = max(MAX_PRICE[s],np.max(hist[s]))

    #  备选买入股票,已经在股票池的股票不再重复购买,创60日新高则入选购买
    option = [x for x in account.universe if x not in account.va
lid_secpos]

    for s in option :
        if np.max(hist[s]) == hist[s][-1] :
            buylist.append(s)
            MAX_PRICE[s] = hist[s][-1]
    # 从最高点下跌17%，卖出
    for s in account.valid_secpos :

        if hist[s][-1] <= MAX_PRICE[s] * (1 - 0.17) :
            cash += hist[s][-1] * account.valid_secpos.get(s)
            order_to(s , 0)
            # 最高价清零
            MAX_PRICE[s] = 0

    # 买入
    for s in buylist :
        order( s, cash / len(buylist) / hist[s][-1] )
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
25.3%	12.4%	14.9%	0.77	0.69	31.6%	0.49	47.7%	9.90

5.8 SMIA · 基于历史状态空间相似性匹配的行业配置 SMIA 模型—取交集

> 来源：https://uqer.io/community/share/55e00e38f9f06c521f156a86

行业轮动分析

本文主要参考广发金工的研报《基于历史状态空间相似性匹配的行业配置SMIA 模型》中所用的方法，通过量化实验室平台完成实证的过程。

1.策略思想

随机过程可以划分为有记忆性特征和无记忆性特征两种。行业轮动具有记忆性，即当期行业的相对表现会影响下一期行业的相对表现，这是由行业之间错综复杂的经济关系所决定的。我们通过寻找历史状态空间中与当期行业收益率排名相似的一些时间点，观察这些时间点之后一期行业轮动的变化特征，从中选取统计上表现较好的行业，作为当前时间点下一期的推荐超配行业，从而实现相似性匹配行业配置（Similarity Matching Industry Allocation,缩写为SMIA）量化模型的构建。

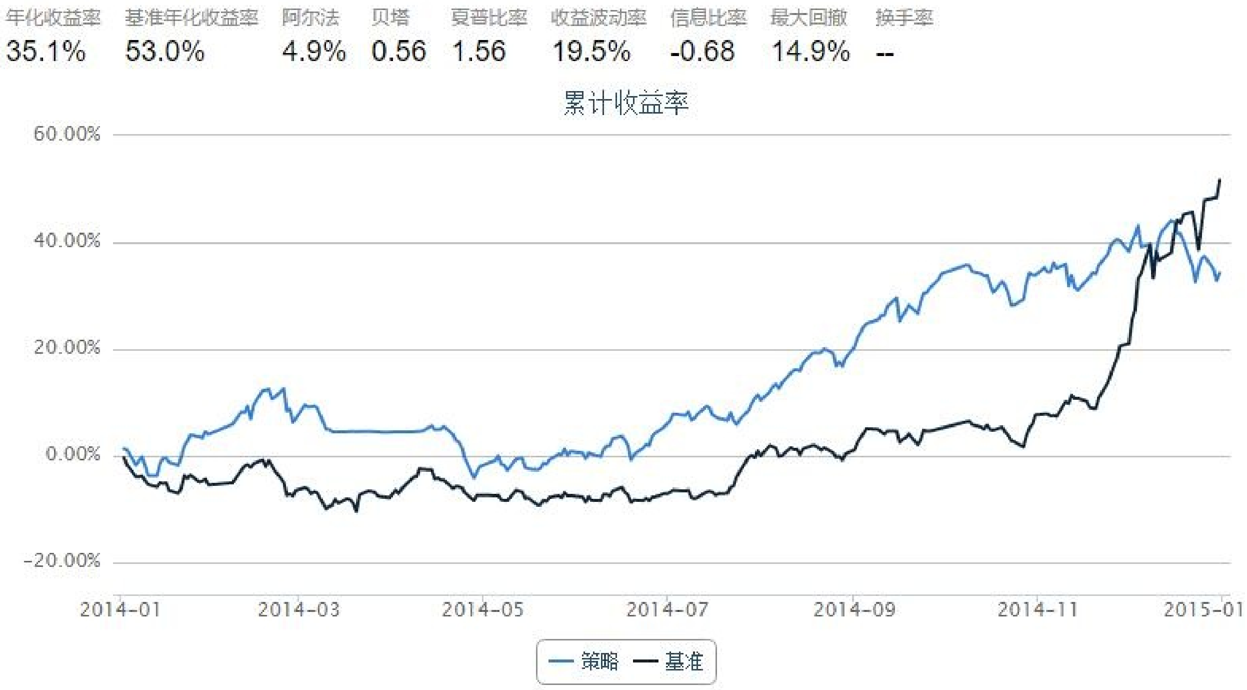
2.实证过程

```
def distanceGet(history, test):
    #计算欧式距离
    distance = sum((history - test).values**2)
    return distance
```

下面的策略作了一些改变，变成行业之间取交集。需要测试别的距离的效果，也可以在上面的函数直接改哦。

```
start = '2014-01-01'                       # 回测起始时间
end = '2015-01-01'                         # 回测结束时间
benchmark = 'HS300'                        # 策略参考标准
universe = universeGet(getIndustryInfo())  # 证券池，支持股票和基金
capital_base = 1000000                     # 起始资金
freq = 'd'                                 # 策略类型，'d'表示日间
策略使用日线回测，'m'表示日内策略使用分钟线回测
refresh_rate = 22                          # 调仓频率，表示执行hand
le_data的时间间隔，若freq = 'd'时间间隔的单位为交易日，若freq = 'm'时间
间隔为分钟

Select_Match = 3                           # 选取三个最相似的行业市
```

```
场排名状态
Select_Order = 14                          # 选取收益率最高的十四个行业

fixYield = DataFrame(fixInd_meanYield).rank(axis=1)   #将收益率矩阵变为序号矩阵

def initialize(account):                   # 初始化虚拟账户状态
    pass

#总体思路是根据调仓时期，将最近22个交易日的的收益率按行业排序，拿之前所有数据源进行匹配，用欧式距离找出最接近的排序的3期，选取它们的下一期，再挑选出这些期行业收益率前两个名的行业。
def handle_data(account): # 每个交易日的买入卖出指令

    today = account.current_date

    if today.strftime("%Y-%m-%d") == '2014-01-02' : #当回测开始调仓时（2014-01-01元旦不交易，顺延到1月2号），不需要调用addYield(current_date)函数

        fixYield_history = fixYield.ix[:len(fixYield)-2] #将最后一期之前的数据，作为匹配数据源
        fixYield_test = fixYield.ix[len(fixYield)-1] #选取最后一期数据来和其他数据源做匹配

        distance = pd.Series(np.zeros(len(fixYield_history ))) #初始化
        for i in xrange(len(fixYield_history)):
            distance[i] = distanceGet(fixYield_history.ix[i],fixYield_test )

        sort_distance = distance.order() #按distance中的值排序，然而其索引值仍保留
        indexSort = sort_distance.index #distance的索引并没有改变，我们得到了它的索引，就可以反过来在fixYield中找到最匹配的源数据

        #industryList = []
        industryList = set()
        intersectionList = set()
        for i in xrange(Select_Match):
            Match_Period = dict(fixYield.ix[indexSort[i]+1]) #取distance值在前3的索引，将这个索引加1，得到最匹配的数据源的下一期
            #industryList = industryList + nlargest(Select_Order, Match_Period, key=Match_Period.get) #取收益率最高的四个索引（行业）
            industryList = set(nlargest(Select_Order, Match_Period, key=Match_Period.get))

            if i == 0:
                intersectionList = industryList
            else:
                intersectionList = intersectionList & industryList
        #industryList = list(set(industryList)) #得到我们选出的行业
        intersectionList = list(intersectionList)
    else:
        addInd_meanYield = addYield(account.current_date) #除去最开始调仓，我们都需要调用addYield(current_date)得到更多的数据源
        addInd_meanYield = DataFrame(addInd_meanYield).rank(axis=1) #将收益率矩阵变序号矩阵
        addInd_meanYield_history = addInd_meanYield.ix[:len(addInd_meanYield)-2] #将这些数据也作为匹配数据源
        addInd_meanYield_test = addInd_meanYield.ix[len(addInd_meanYield)-1] #选取最后一期数据来和其他数据源做匹配

        distance = pd.Series(np.zeros(len(addInd_meanYield_history)+len(fixYield )))
        for i in xrange(len(fixYield)):
            distance[i] = distanceGet(fixYield.ix[i],addInd_meanYield_test)
        for i in xrange(len(fixYield),len(fixYield) + len(addInd_meanYield_history)):
            distance[i] = distanceGet(addInd_meanYield_history.ix[i-len(fixYield)],addInd_meanYield_test)

        sort_distance = distance.order()
        indexSort = sort_distance.index

        #industryList = []
        industryList = set()
        intersectionList = set()
        for i in xrange(Select_Match): #选取3个匹配

            if indexSort[i]+1 < len(fixYield): #若找的日期不超过fixYield的范围，直接计算。
                Match_Period = dict(fixYield.ix[indexSort[i]+1])
                #industryList = industryList + nlargest(Select_Order, Match_Period, key=Match_Period.get)
                industryList = set(nlargest(Select_Order, Match_Period, key=Match_Period.get))

            elif indexSort[i]+1 < len(distance) : #若找到的日期不超过所有数据期数减一，但超过了fixYield的范围，则匹配在addInd_meanYield_history中
                Match_Period = dict(addInd_meanYield_history.ix[indexSort[i]+1-len(fixYield)])
                #industryList = industryList + nlargest(Select_Order, Match_Period, key=Match_Period.get)
                industryList = set(nlargest(Select_Order, Match_Period, key=Match_Period.get))

            else:                             #这里其实就是找到了本身，其匹配源数据找到了上一期
                Match_Period = dict(addInd_meanYield_test)
                #industryList = industryList + nlargest(Select_Order, Match_Period, key=Match_Period.get)
                industryList = set(nlargest(Select_Order, Match_
Period, key=Match_Period.get))

            if i == 0:
                intersectionList = industryList
            else:
                intersectionList = intersectionList & industryLi
st
        #industryList = list(set(industryList))
        intersectionList = list(intersectionList)


    buyList = []
    for key in indContent:
        if key in intersectionList:
            buyList = buyList + indContent[key]

    #除去不在account.universe以及停牌和新股
    for stk in buyList[:]:
        if stk not in account.universe or account.referencePrice
[stk] == 0 or np.isnan(account.referencePrice[stk]):
            buyList.remove(stk)

    for stk in account.valid_secpos:
        order_to(stk, 0)

    for stk in buyList:
        order(stk, account.referencePortfolioValue/account.refer
encePrice[stk]/len(buyList))
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
35.1%	53.0%	4.9%	0.56	1.56	19.5%	-0.68	14.9%	--

5.9 神经网络

神经网络交易的训练部分

> 来源：https://uqer.io/community/share/55b8af12f9f06c91f818c607

```
import pybrain as brain
training_set = ("20050101", "20130101")        # 训练集（六年）
testing_set  = ("20150101", "20150525")        # 测试集（2015上半年
数据）
universe     = ['000001']
                                               # 目标股票池

HISTORY      = 10                              # 通过前十日数据预测
```

```
from pybrain.datasets import SupervisedDataSet
### 建立数据集
def make_training_data():
    ds = SupervisedDataSet(HISTORY, 1)
    for ticker in universe: # 遍历每支股票
        raw_data = DataAPI.MktEqudGet(ticker=ticker, beginDate=t
raining_set[0], endDate=training_set[1], field=[
                'tradeDate', 'closePrice'    # 敏感字段
            ], pandas="1")
        plist = list(raw_data['closePrice'])
        for idx in range(1, len(plist) - HISTORY - 1):
            sample = []
            for i in range(HISTORY):
                sample.append(plist[idx + i - 1] / plist[idx + i
] - 1)
            answer = plist[idx + HISTORY - 1] / plist[idx + HIST
ORY] - 1

            ds.addSample(sample, answer)
    return ds

### 建立测试集
def make_testing_data():
    ds = SupervisedDataSet(HISTORY, 1)
    for ticker in universe: # 遍历每支股票
        raw_data = DataAPI.MktEqudGet(ticker=ticker, beginDate=t
esting_set[0], endDate=testing_set[1], field=[
                'tradeDate', 'closePrice'    # 敏感字段
            ], pandas="1")
        plist = list(raw_data['closePrice'])
        for idx in range(1, len(plist) - HISTORY - 1):
            sample = []
            for i in range(HISTORY):
                sample.append(plist[idx + i - 1] / plist[idx + i
] - 1)
            answer = plist[idx + HISTORY - 1] / plist[idx + HIST
ORY] - 1

            ds.addSample(sample, answer)
    return ds
```

```
from pybrain.supervised.trainers import BackpropTrainer
### 构造BP训练实例
def make_trainer(net, ds, momentum = 0.1, verbose = True, weight
decay = 0.01): # 网络, 训练集, 训练参数
    trainer = BackpropTrainer(net, ds, momentum = momentum, verb
ose = verbose, weightdecay = weightdecay)
    return trainer
```

```
### 开始训练
def start_training(trainer, epochs = 15): # 迭代次数
    trainer.trainEpochs(epochs)

def start_testing(net, dataset):
    return net.activateOnDataset(dataset)
```

```
### 保存参数
from pybrain.tools.customxml import NetworkWriter
def save_arguments(net):
    NetworkWriter.writeToFile(net, 'huge_data.csv')
    print 'Arguments save to file net.csv'
```

```
from pybrain.tools.shortcuts import buildNetwork
### 初始化神经网络
fnn = buildNetwork(HISTORY, 15, 7, 1)

training_dataset = make_training_data()
testing_dataset  = make_testing_data()
trainer = make_trainer(fnn, training_dataset)
start_training(trainer, 5)
save_arguments(fnn)
print start_testing(fnn, testing_dataset)

Total error:  0.00226884924246
Total error:  0.00058242191557
Total error:  0.00058089738079
Total error:  0.000581061747831
Total error:  0.000580708420341
Arguments save to file net.csv
[[-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
```

```
  [-0.00055257]
  [-0.00055257]
  [-0.00055257]
  [-0.00055257]
  [-0.00055257]
  [-0.00055257]
  [-0.00055257]
  [-0.00055257]
  [-0.00055257]
  [-0.00055257]
  [-0.00055257]
  [-0.00055257]
  [-0.00055257]
  [-0.00055257]
  [-0.00055257]
  [-0.00055257]
  [-0.00055257]
  [-0.00055257]
  [-0.00055257]
  [-0.00055257]
  [-0.00055257]
  [-0.00055257]
  [-0.00055257]
  [-0.00055257]
  [-0.00055257]
  [-0.00055257]
  [-0.00055257]
  [-0.00055257]
  [-0.00055257]
  [-0.00055257]
  [-0.00055257]
  [-0.00055257]
  [-0.00055257]
  [-0.00055257]
  [-0.00055257]
  [-0.00055257]
  [-0.00055257]
  [-0.00055257]
  [-0.00055257]
  [-0.00055257]
  [-0.00055257]
  [-0.00055257]
  [-0.00055257]
  [-0.00055257]
  [-0.00055257]
  [-0.00055257]
  [-0.00055257]
  [-0.00055257]
  [-0.00055257]
  [-0.00055257]
  [-0.00055257]
  [-0.00055257]
  [-0.00055257]
```

```
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
```

```
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
```

```
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
 [-0.00055257]
```

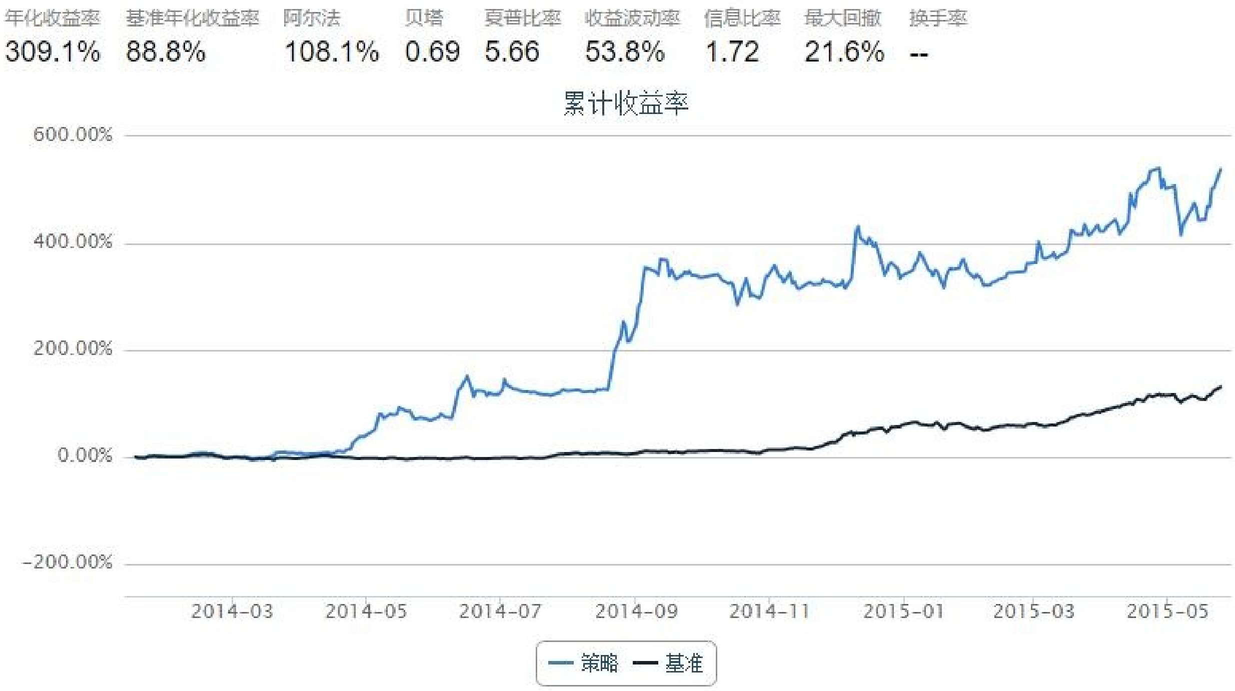
```
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
   [-0.00055257]
```

```
 [-0.00055257]]
```

通过神经网络进行交易

> 来源：https://uqer.io/community/share/55b8acbaf9f06c91fa18c5ce

```
start = '2014-01-01'                        # 回测起始时间
end = '2015-05-25'                          # 回测结束时间
benchmark = 'HS300'                         # 策略参考标准
universe = set_universe('HS300')            # 证券池，支持股票和基金
capital_base = 1000000                      # 起始资金
freq = 'd'                                  # 策略类型，'d'表示日间
策略使用日线回测，'m'表示日内策略使用分钟线回测
refresh_rate = 1                            # 调仓频率，表示执行hand
le_data的时间间隔，若freq = 'd'时间间隔的单位为交易日，若freq = 'm'时间
间隔为分钟

import pybrain as brain
from pybrain.tools.shortcuts import buildNetwork
from pybrain.tools.customxml import NetworkReader
HISTORY      = 10                              # 通过前十日数据预测
fnn = buildNetwork(HISTORY, 15, 7, 1)          # 初始化神经网络

def initialize(account):                       # 初始化虚拟账户状态
    fnn = NetworkReader.readFrom('net.csv')

def handle_data(account):                      # 每个交易日的买入卖
出指令
    hist = account.get_attribute_history('closePrice', 10)
    bucket = []
    for s in account.universe:
        sample = hist[s]
        possibility = fnn.activate(sample)
        bucket.append((possibility, s))

        if possibility < 0 and s in account.valid_secpos:
            order_to(s, 0)

    bucket = sorted(bucket, key=lambda x: x[0], reverse=True)
    print bucket[0][0]

    if bucket[0][0] < 0:
        raise Exception('Network Error')

    for s in bucket[:10]:
        if s[0] > 0.5 and s[1] not in account.valid_secpos:
            order(s[1], 10000 * s[0] * 80000)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
309.1%	88.8%	108.1%	0.69	5.66	53.8%	1.72	21.6%	--

```
[ 1.44446298]
[ 1.57722526]
[ 1.44509945]
[ 1.44829344]
[ 1.48584942]
[ 1.60968867]
[ 1.61088618]
[ 1.43639898]
[ 1.43767639]
[ 1.43911414]
[ 1.43768517]
[ 1.43585662]
[ 1.43720968]
[ 1.43317016]
[ 1.43301566]
[ 1.42953404]
[ 1.42678559]
[ 1.43098489]
[ 1.42855878]
[ 1.42709837]
[ 1.42726163]
[ 1.42585347]
[ 1.42973957]
[ 1.42980622]
[ 1.43599317]
[ 1.44286782]
[ 1.57580564]
[ 1.59120978]
[ 1.6130606]
[ 1.59582232]
[ 1.5850841]
[ 1.61084701]
```

```
 [ 1.59595849]
 [ 1.52961191]
 [ 1.50583099]
 [ 1.46038687]
 [ 1.44689328]
 [ 1.5432668]
 [ 1.55312445]
 [ 1.44337678]
 [ 1.44056972]
 [ 1.50173311]
 [ 1.59748366]
 [ 1.4267731]
 [ 1.47709901]
 [ 1.62105239]
 [ 1.60780394]
 [ 1.53541989]
 [ 1.60721757]
 [ 1.58754631]
 [ 1.5909996]
 [ 1.60486746]
 [ 1.48532045]
 [ 1.56199286]
 [ 1.42685994]
 [ 1.42218871]
 [ 1.42513733]
 [ 1.42560821]
 [ 1.42627889]
 [ 1.42422753]
 [ 1.42382572]
 [ 1.42222283]
 [ 1.41752142]
 [ 1.41257471]
 [ 1.41516891]
 [ 1.41390184]
 [ 1.58426403]
 [ 1.53824457]
 [ 1.45517987]
 [ 1.500387]
 [ 1.48309551]
 [ 1.51026016]
 [ 1.52573794]
 [ 1.53639431]
 [ 1.35975534]
 [ 1.3949126]
 [ 1.41854269]
 [ 1.5371124]
 [ 1.5318818]
 [ 1.61626035]
 [ 1.46463971]
 [ 1.35377736]
 [ 1.3781526]
 [ 1.36485304]
 [ 1.35738739]
 [ 1.35879235]
 [ 1.35848317]
 [ 1.35674074]
 [ 1.35842602]
 [ 1.35549472]
 [ 1.40440556]
 [ 1.35685947]
 [ 1.35700859]
 [ 1.44201184]
 [ 1.43235995]
 [ 1.37015535]
 [ 1.35396728]
 [ 1.35545512]
 [ 1.35623892]
 [ 1.39545221]
 [ 1.35725555]
 [ 1.52999178]
 [ 1.52399418]
 [ 1.39365249]
 [ 1.36779515]
 [ 1.35482391]
 [ 1.40293755]
 [ 1.37213596]
 [ 1.35738371]
 [ 1.35808458]
 [ 1.35662849]
 [ 1.35528448]
 [ 1.35510845]
 [ 1.35379783]
 [ 1.35430934]
 [ 1.35312843]
 [ 1.35581243]
 [ 1.36879701]
 [ 1.41158962]
 [ 1.44027263]
 [ 1.44380821]
 [ 1.48272708]
 [ 1.51507127]
 [ 1.46605994]
 [ 1.61084145]
 [ 1.58922279]
 [ 1.46771218]
 [ 1.40289457]
 [ 1.34716878]
 [ 1.35043834]
 [ 1.35590544]
 [ 1.37653415]
 [ 1.34764272]
 [ 1.34831244]
 [ 1.34689904]
 [ 1.34150245]
 [ 1.33927252]
 [ 1.33978952]
 [ 1.3470568]
 [ 1.34433552]
 [ 1.34484056]
 [ 1.34160806]
 [ 1.3407761]
 [ 1.3424078]
 [ 1.3433431]
 [ 1.34328446]
 [ 1.33992925]
 [ 1.34388204]
 [ 1.34802088]
 [ 1.3453579]
 [ 1.3428265]
 [ 1.34329775]
 [ 1.34191156]
 [ 1.34611248]
 [ 1.37349663]
 [ 1.34815805]
 [ 1.34014992]
 [ 1.34521152]
 [ 1.34456372]
 [ 1.34089661]
 [ 1.34023757]
 [ 1.3410812]
 [ 1.33807578]
 [ 1.33572014]
 [ 1.34433535]
 [ 1.33505861]
 [ 1.33827504]
 [ 1.33755043]
 [ 1.38559783]
 [ 1.35527351]
 [ 1.33053597]
 [ 1.33701674]
 [ 1.33273647]
 [ 1.33668717]
 [ 1.33941937]
 [ 1.34060378]
 [ 1.3372182]
 [ 1.61340736]
 [ 1.59055412]
 [ 1.33505241]
 [ 1.60308339]
 [ 1.51156137]
 [ 1.35797843]
 [ 1.34580909]
 [ 1.48117895]
 [ 1.44494812]
 [ 1.35293003]
 [ 1.35665647]
 [ 1.37410369]
 [ 1.35666235]
 [ 1.33729064]
 [ 1.45931719]
 [ 1.55375605]
 [ 1.48339986]
 [ 1.35060715]
 [ 1.36146995]
 [ 1.34245541]
 [ 1.35342592]
 [ 1.35796042]
 [ 1.37098111]
 [ 1.34045319]
 [ 1.42147708]
 [ 1.365122]
 [ 1.4076879]
 [ 1.39762825]
 [ 1.34262013]
 [ 1.38706403]
 [ 1.33523713]
 [ 1.33186205]
 [ 1.33077059]
 [ 1.3324637]
 [ 1.33112122]
 [ 1.32952302]
 [ 1.33383435]
 [ 1.32954544]
 [ 1.33443469]
 [ 1.33090967]
 [ 1.33522262]
 [ 1.33175321]
 [ 1.49987289]
 [ 1.51376666]
 [ 1.4208718]
 [ 1.49241705]
 [ 1.36766608]
 [ 1.36990194]
 [ 1.33322159]
 [ 1.34836793]
 [ 1.34669257]
 [ 1.36690579]
 [ 1.37890552]
 [ 1.59037649]
 [ 1.60582728]
 [ 1.61743431]
 [ 1.62123338]
 [ 1.61336502]
 [ 1.60121318]
 [ 1.62107838]
 [ 1.41357384]
 [ 1.61966948]
 [ 1.51775743]
 [ 1.33704794]
 [ 1.37279934]
 [ 1.34484306]
 [ 1.3705884]
 [ 1.41262748]
 [ 1.44408315]
 [ 1.52046936]
 [ 1.38814136]
 [ 1.38882472]
 [ 1.35596408]
 [ 1.52776999]
 [ 1.55767315]
 [ 1.33500518]
 [ 1.33840795]
 [ 1.34727997]
 [ 1.43367698]
 [ 1.35595655]
 [ 1.34698186]
 [ 1.59583696]
 [ 1.374913]
 [ 1.60214431]
 [ 1.53554784]
 [ 1.49221176]
 [ 1.59822169]
 [ 1.35287993]
 [ 1.34985064]
 [ 1.34512204]
 [ 1.33554636]
 [ 1.33612458]
 [ 1.32905663]
 [ 1.32990288]
 [ 1.36225504]
 [ 1.59836396]
 [ 1.32984726]
 [ 1.33153792]
 [ 1.39786779]
 [ 1.3416728]
 [ 1.3547156]
 [ 1.3417874]
 [ 1.33787953]
 [ 1.42237594]
 [ 1.32939148]
 [ 1.34560785]
 [ 1.33542025]
 [ 1.32921129]
 [ 1.32924703]
 [ 1.32956219]
 [ 1.32953676]
 [ 1.32962066]
 [ 1.33064464]
 [ 1.32916515]
 [ 1.32946366]
 [ 1.33199463]
 [ 1.32940815]
 [ 1.33035788]
 [ 1.33158764]
 [ 1.33103393]
 [ 1.3312874]
 [ 1.32907548]
 [ 1.33131474]
 [ 1.33113065]
 [ 1.33056411]
 [ 1.54542979]
 [ 1.43053565]
 [ 1.44441014]
 [ 1.55239121]
 [ 1.37602661]
 [ 1.62125583]
 [ 1.36640902]
 [ 1.56636469]
 [ 1.33713086]
 [ 1.33348418]
 [ 1.33584004]
 [ 1.35366715]
 [ 1.39788942]
 [ 1.41189411]
 [ 1.57317611]
 [ 1.40385926]
 [ 1.61962342]
 [ 1.55777659]
 [ 1.5813632]
 [ 1.52487439]
 [ 1.44917861]
 [ 1.35809968]
 [ 1.35031112]
 [ 1.34328138]
 [ 1.3453355]
 [ 1.36096032]
 [ 1.34087397]
```

5.10 PAMR · PAMR ： 基于均值反转的投资组合选择策略 - 修改版

> 来源：https://uqer.io/community/share/55a4c52bf9f06c6dd3e17f0f

策略思路：

该策略的主要思想是用一个损失函数反映均值反转性质，即如果基于前一期相对价格的预期收益值大于一定阈值，损失值将线性增长；否则，损失为0

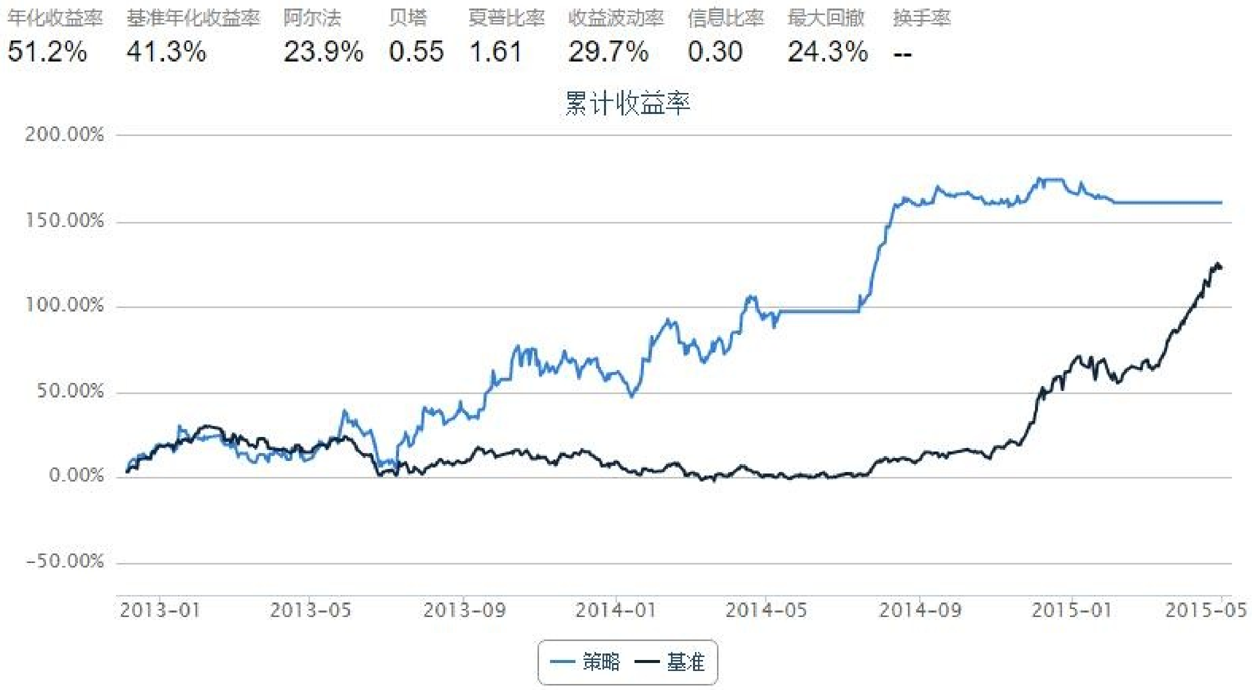
策略实现

m个资产每日调仓：对每个资产，收益高于总资产平均收益者，减持；收益低于总资产平均收益者，增持

具体参见文献： http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.421.579&rep=rep1&type=pdf

```python
from CAL.PyCAL import *
from numpy import *
import pandas as pd
import numpy as np
from pandas import DataFrame

import cvxopt
from cvxopt import matrix
from cvxopt.blas import dot
import cvxopt.solvers as cs

# parameters used in updatePAMR
sensitivity = 0.8
C = 600

start = datetime(2012, 12, 1)
end   = datetime(2015, 5, 1)
benchmark = 'HS300'
universe = set_universe('SH180')
capital_base = 1e8
refresh_rate = 1
window = 1

tickers = [stk[0:6] for stk in universe]
portfolio = DataFrame(1.0, index = universe, columns = ['prePosition', 'position', 'relative_price'])

def initialize(account):
    account.amount = capital_base
    account.universe = universe
    account.days = 0
```

```python
def handle_data(account):
    today = account.current_date
    today_str = today.strftime("%Y%m%d")
    for stk in universe:
        hist_close = account.get_attribute_history('closePrice', 2)
        hist_pre_close = account.get_attribute_history('preClosePrice', 2)
        try:
            portfolio['relative_price'][stk] = hist_close[stk][-1]/hist_pre_close[stk][-1]
            #print stk, today_str, portfolio['relative_price'][stk]
        except:
            continue
    portfolio['relative_price'] = portfolio['relative_price'].fillna(1.0)

    portfolio['prePosition'] = portfolio['position']
    a = portfolio['prePosition']
    b = portfolio['relative_price']
    portfolio['position'] = normalizePortfolio(updatePAMR(a, b, sensitivity, C))

    for stk in portfolio.index:
        try:
            stk_amount = capital_base*portfolio['position'][stk]/hist_close[stk][-1]
            order_to(stk, stk_amount)
        except:
            continue

def lossFunction(portfolio, relative_price, sensitivity):
    # define a e-insensitive loss function
    # portfolio vector: b
    # price relative vector: x
    # sensitivity parameter: e
    # then: loss = max(0, dot(x,b) - e)
    portfolio_return = portfolio.transpose().dot(relative_price)
    if portfolio_return < sensitivity:
        return 0
    else:
        return portfolio_return - sensitivity

def normalizePortfolio(portfolio):
    # original portfolio vector: b_origin
    # find b = argmin(|b - b_origin|^2) under condition:
    # sum(b_i) = 1 and b_i > 0 for all i

    # solve the problems using Quadratic Programming Method:
    # http://abel.ee.ucla.edu/cvxopt/userguide/coneprog.html#quadratic-programming
    n = portfolio.shape[0]
    S = cvxopt.matrix(0.0, (n,n))
    S[::n+1] = 1.0
    S = S.T*S

    pbar = cvxopt.matrix(portfolio.values).T*(S + S.T)
    pbar = pbar.T
    G = cvxopt.matrix(0.0, (n,n))
    G[::n+1] = -1.0
    h = cvxopt.matrix(0.0, (n,1))
    A = cvxopt.matrix(1.0, (1,n))
    b = cvxopt.matrix(1.0)

    cvxopt.solvers.options['show_progress'] = False
    x = cs.qp(S, -pbar, G, h, A, b)['x']
    b = portfolio.copy()
    for i in range(0, n):
        b.ix[b.index[i]] = x[i]
    return b

def updatePAMR(portfolio, relative_price, sensitivity, C):
    # update portfolio by PAMR2 methods:
    # PAMR: Passive Aggressive Mean Reversion Strategy for Portfolio Selection.
    # Bin Li, Peilin Zhao, Steven C.H. Hoi, and V. Gopalkrishnan.

    # Machine Learning, 2012, 87(2), 221 - 258.

    loss = lossFunction(portfolio, relative_price, sensitivity)
    avg_ret = relative_price.sum()/relative_price.shape[0]
    tmp = ((relative_price - avg_ret)**2).sum() + 1.0/2/C
    tau = loss/tmp

    return portfolio - tau*(relative_price - avg_ret)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
51.2%	41.3%	23.9%	0.55	1.61	29.7%	0.30	24.3%	--

5.11 Fisher Transform · Using Fisher Transform Indicator

> 来源：https://uqer.io/community/share/54b5c288f9f06c276f651a16

策略思路：

在技术分析中，很多时候，人们都把股价数据当作正态分布的数据来分析。但是，其实股价数据分布并不符合正态分布。Fisher Transformation是一个可以把股价数据变为类似于正态分布的方法。

Fisher Transformation将市场数据的走势平滑化，去掉了一些尖锐的短期振荡；利用今日和前一日该指标的交错可以给出交易信号；

例如，对于沪深300指数使用Fisher变换的结果见本文后面的具体讨论。

Fisher Transformation

- 定义今日中间价：

  ```
  mid=(low+high)/2
  ```

- 确定计算周期，例如可使用10日为周期。计算周期内最高价和最低价：

  ```
  lowestLow=周期内最低价，    highestHigh=周期内最高价
  ```

- 定义价变参数（其中的 `ratio` 为0-1之间常数，例如可取0.5或0.33）：

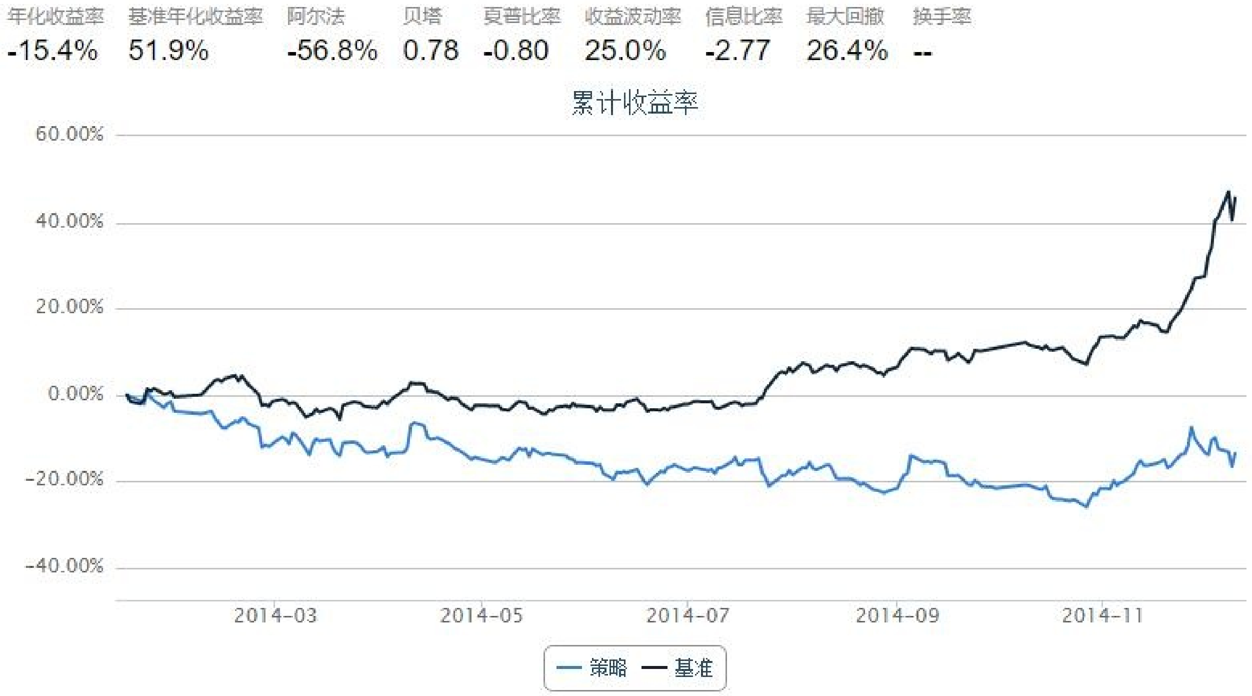
$$x = \text{ratio} \times 2 \times \left(\frac{\text{mid} - \text{lowestLow}}{\text{highestHigh} - \text{lowestLow}} - \frac{1}{2} \right) + (1 - \text{ratio}) \times \text{前一日的}x$$

- 对价变参数 `x` 使用Fisher变换，得到Fisher指标：

$$\text{fish} = 0.5 \times \log \frac{1+x}{1-x} + 0.5 \times \text{前一日的 fish}$$

```
import quartz
import quartz.backtest    as qb
import quartz.performance as qp
from   quartz.api         import *

import pandas as pd
import numpy  as np
from datetime   import datetime
from matplotlib import pylab
```

```
start = datetime(2014, 1, 1)                # 回测起始时间
end   = datetime(2014, 12, 10)                # 回测结束时间
benchmark = 'HS300'                            # 使用沪深 300 作为
参考标准
universe = set_universe('SH50')    # 股票池
capital_base = 100000                       # 起始资金


refresh_rate = 1
window = 10

# 本策略对于window非常非常敏感！！！

histFish = pd.DataFrame(0.0, index = universe, columns = ['preDi
ff', 'preFish', 'preState'])

def initialize(account):                    # 初始化虚拟账户状态
    account.amount = 10000
    account.universe = universe
    add_history('hist', window)


def handle_data(account):                # 每个交易日的买入卖出指令

    for stk in account.universe:
        prices = account.hist[stk]
        if prices is None:
            return

        preDiff = histFish.at[stk, 'preDiff']
        preFish = histFish.at[stk, 'preFish']
        preState = histFish.at[stk, 'preState']

        diff, fish = FisherTransIndicator(prices, preDiff, preFi
sh)
        if fish > preFish:
            state = 1
        elif fish < preFish:
            state = -1
        else:
            state = 0

        if state == 1 and preState == -1:
            #stkAmount = int(account.amount / prices.iloc[-1]['openPrice'])
            order(stk, account.amount)
        elif state == -1 and preState == 1:
            order_to(stk, 0)

        histFish.at[stk, 'preDiff'] = diff
        histFish.at[stk, 'preFish'] = fish
        histFish.at[stk, 'preState'] = state


def FisherTransIndicator(windowData, preDiff, preFish):
    # This function calculate the Fisher Transform indicator based on the data
    # in the windowData.
    minLowPrice = min(windowData['lowPrice'])
    maxHghPrice = max(windowData['highPrice'])
    tdyMidPrice = (windowData.iloc[-1]['lowPrice'] + windowData.iloc[-1]['highPrice'])/2.0

    diffRatio = 0.33
    # 本策略对于diffRatio同样非常敏感！！！

    diff = (tdyMidPrice - minLowPrice)/(maxHghPrice - minLowPrice) - 0.5
    diff = 2 * diff
    diff = diffRatio * diff + (1.0 - diffRatio) * preDiff

    if diff > 0.99:
        diff = 0.999
    elif diff < -0.99:
        diff = -0.999

    fish = np.log((1.0 + diff)/(1.0 - diff))
    fish = 0.5 * fish + 0.5 * fish

    return diff, fish
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
-15.4%	51.9%	-56.8%	0.78	-0.80	25.0%	-2.77	26.4%	--

沪深300指数上使用Fisher Transformation

- 对最近半年的沪深300进行Fisher变换，得到的指标能够比较温和准确反映出指数的变化

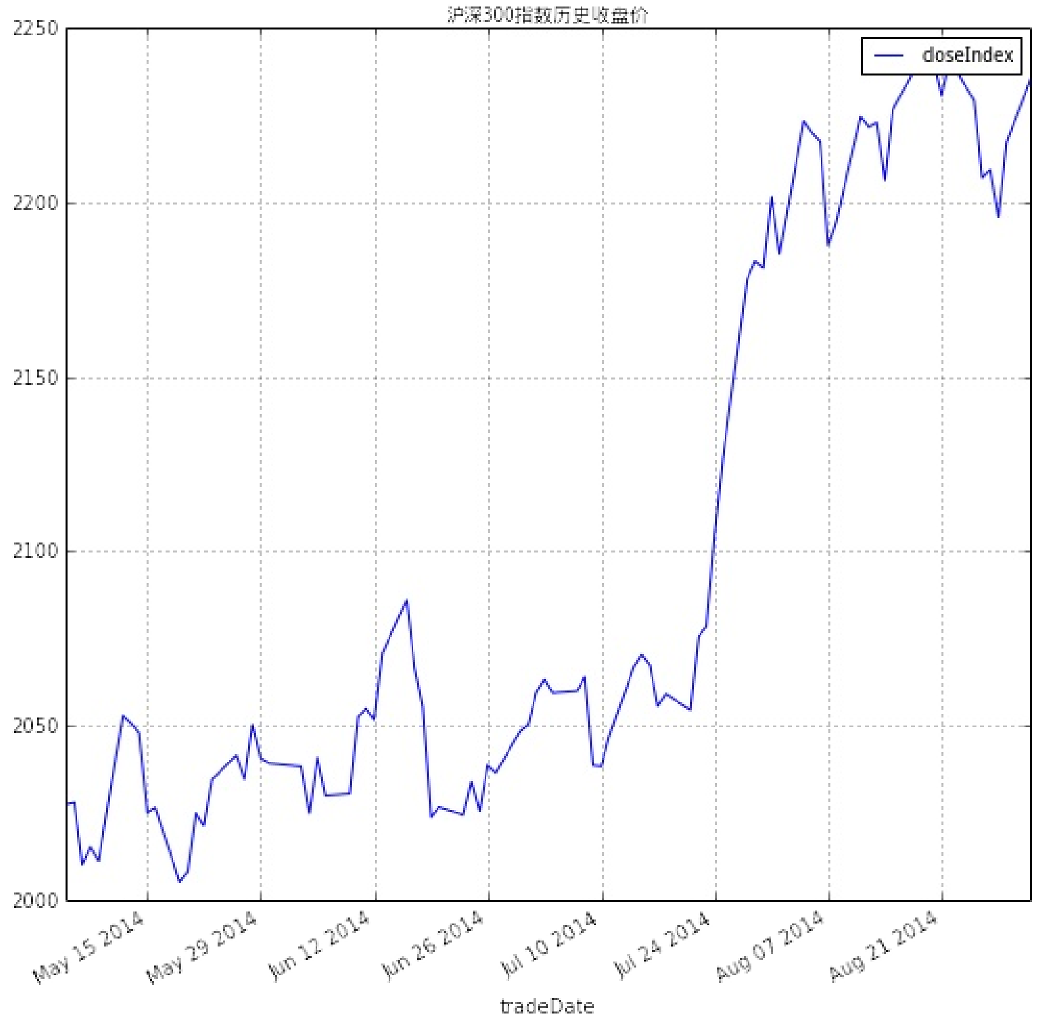
```
from CAL.PyCAL import *

# DataAPI.MktIdxdGet返回pandas.DataFrame格式
index =  DataAPI.MktIdxdGet(indexID = "000001.ZICN", beginDate =
"20140501", endDate = "20140901")
```

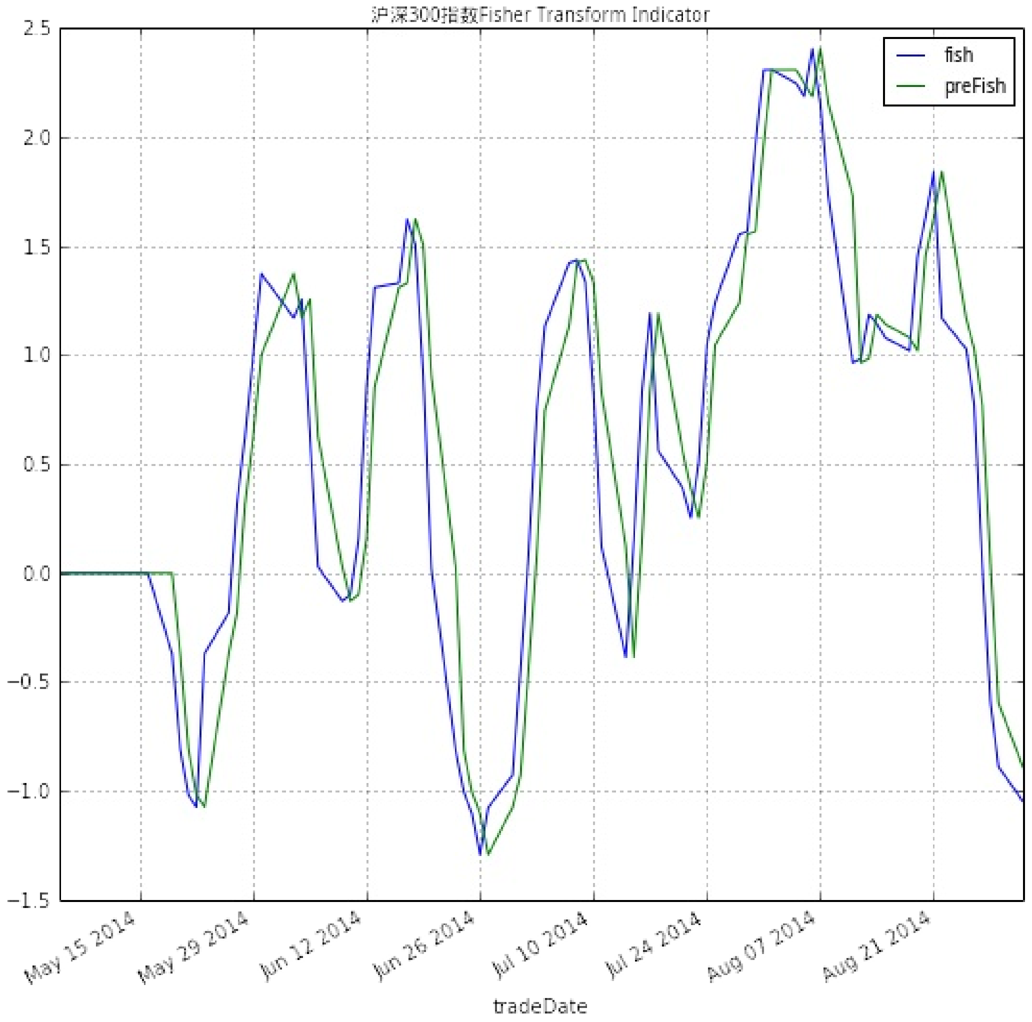
```
index.head()
```

	indexID	tradeDate	ticker	secShortName	exchangeCD
0	000001.ZICN	2014-05-05	1	上证综指	XSHG
1	000001.ZICN	2014-05-06	1	上证综指	XSHG
2	000001.ZICN	2014-05-07	1	上证综指	XSHG
3	000001.ZICN	2014-05-08	1	上证综指	XSHG
4	000001.ZICN	2014-05-09	1	上证综指	XSHG

```
def FisherTransIndicator(windowData, preDiff, preFish, state):
    # This function calculate the Fisher Transform indicator bas
ed on the data
    # in the windowData.
    minLowPrice = min(windowData['lowestIndex'])
    maxHghPrice = max(windowData['highestIndex'])
    tdyMidPrice = (windowData.iloc[-1]['lowestIndex'] + windowDa
ta.iloc[-1]['highestIndex'])/2.0

    diffRatio = 0.5

    diff = (tdyMidPrice - minLowPrice)/(maxHghPrice - minLowPric
e) - 0.5
    diff = 2 * diff

    if state == 1:
        diff = diffRatio * diff + (1 - diffRatio) * preDiff

    if diff > 0.995:
        diff = 0.999
    elif diff < -0.995:
        diff = -0.999

    fish = np.log((1 + diff)/(1 - diff))
    if state == 1:
        fish = 0.5 * fish + 0.5 * fish

    return diff, fish
```

```
window = 10

index['diff'] = 0.0
index['fish'] = 0.0
index['preFish'] = 0.0

for i in range(window, index.shape[0]):
    windowData = index.iloc[i-window : i]
    if i == window:
        diff, fish = FisherTransIndicator(windowData, 0, 0, 1)
        index.at[i,'preFish'] = 0
        index.at[i,'diff'] = diff
        index.at[i,'fish'] = fish
    else:
        preDiff = index.iloc[i-1]['diff']
        preFish = index.iloc[i-1]['fish']
        diff, fish = FisherTransIndicator(windowData, preDiff, p
reFish, 1)
        index.at[i,'preFish'] = preFish
        index.at[i,'diff'] = diff
        index.at[i,'fish'] = fish


Plot(index, settings = {'x':'tradeDate','y':'closeIndex', 'title'
:u'沪深300指数历史收盘价'})
Plot(index, settings = {'x':'tradeDate','y':['fish', 'preFish'],
'title':u'沪深300指数Fisher Transform Indicator'})
```


沪深300指数历史收盘价
doseIndex
2250
2200
2150
2100
2050
2000
May 15 2014
May 29 2014
Jun 12 2014
Jun 26 2014
Jul 10 2014
Jul 24 2014
Aug 07 2014
Aug 21 2014
tradeDate



- 上图中的蓝色曲线表示Fisher指标，绿色曲线表示前一日的Fisher指标，两个指标的交错可以给出沪深300指数涨跌情况的信号

5.12 分型假说， Hurst 指数 · 分形市场假说，一个听起来很美的假说

> 来源：https://uqer.io/community/share/564c3bc2f9f06c4446b48393

写在前面

- 9月的时候说想把arch包加进去，昨儿发现优矿已经加好了，由于优矿暂时没有开放历史高频接口，我索性就分享一个冷冷的小知识：分形市场假说（FMH），分析中玩的是低频数据（日线，或者分钟线）。
- 所谓分形市场假说，就是人们发现有效市场假说的种种不合理后，提出的一种假说，我曾经有仔细关注过这一块，因为这个假说真是太「中国特色」了：

它有几个主要论点：

1. 当市场是由各种投资期限的投资者组成时，市场是稳定的（长期投资者和短期投资者），当投资者单一时，则市场会出流动性问题；
2. 信息集对基本分析和技术分析来讲短期影响比长期影响要大；
3. 当某一事件的出现使得基础分析的有效性值得怀疑时，长期投资者或者停止入市操作或者基于短期信息进行买卖；
4. 价格是短期技术分析和长期基础分析的综合反应；
5. 如果某种证券与经济周期无关，那么它本身就不存在长期趋势。此时，交易行为、市场流动性和短期信息将占主导地位。

总之就是一个具有「正反馈、非线性、分形、混沌、耗散」等等很牛逼的概念，深深吸引着曾经学过物理学的我。。。

关于Hurst指数以及MF-DFA

- 现在对于分形市场假说的主要方法论就是 Hurst指数，通过MF-DFA（Multifractal detrended fluctuation analysis）来计算， 具体的可以维基百科一下，大体就是当hurst>0.5时时间序列是一个persistent的过程，当hurst>0.5时时间序列是一个anti-persistent的过程，当hurst=0.5时间序列是一个不存在记忆的随机游走过程。
-
- 而在实际计算中，不会以理论值0.5作为标准（一般会略大于0.5）

写在最后

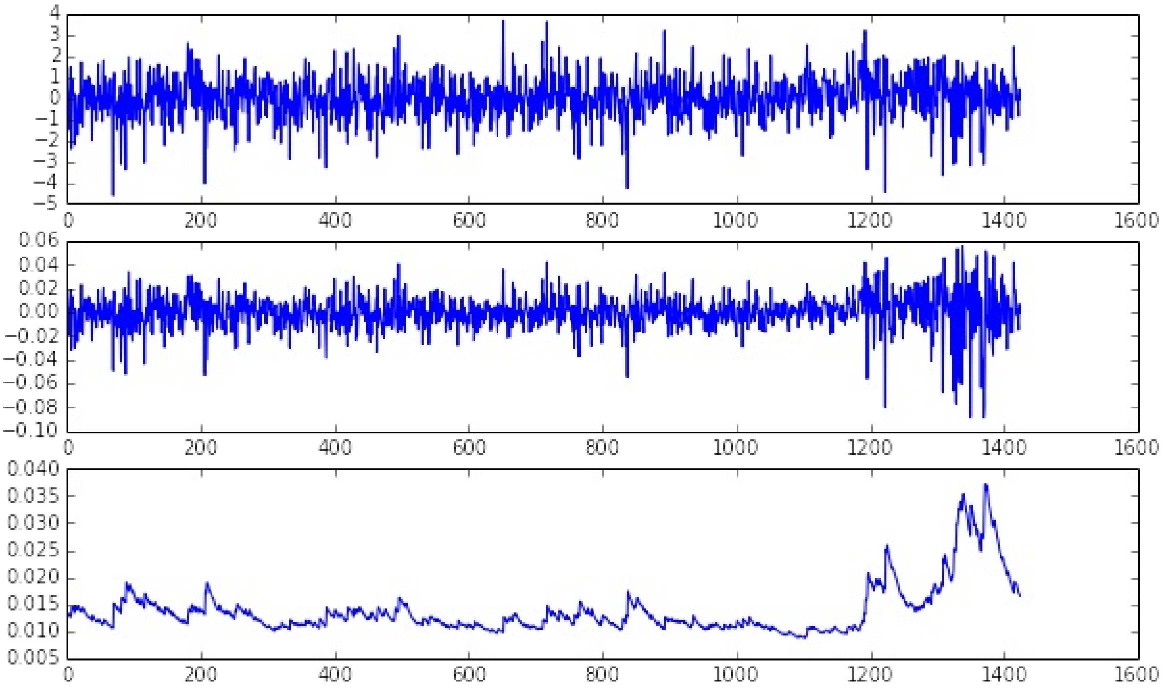
- 这份工作来自于LADISLAV KRISTOUFEK这位教授在12年的工作，论文名叫做RACTAL MARKETS HYPOTHESIS AND THE GLOBAL FINANCIAL CRISIS: SCALING, INVESTMENT HORIZONS AND LIQUIDITY

- 这位教授后来在13年把这项工作强化了一下（加了点小波的方法），把论文的图画得美美哒，竟然发表在了Nature的子刊Scientific Report上。当年我的导师发了一篇SR可是全校通报表扬啊，虽然现在我以前在物理系的导师说今年有4篇SR发表。。
- 总之，如果谁对这个感兴趣，或者想在Nature上水一篇文章，可以研究研究。
- 这个方法对设计策略有没有什么用？好像没有用哎，所以我发表在「研究」板块里了哈。不过10年海通有研究员测试过根据这个方法写的策略，据说alpha还不错。
- 算法部分我用的是自己的library库。

```
import numpy as np
import pandas as pd
from arch import arch_model # GARCH(1,1)
from matplotlib import pyplot as plt
from datetime import timedelta
from CAL.PyCAL import *
from lib.Hurst import *
```

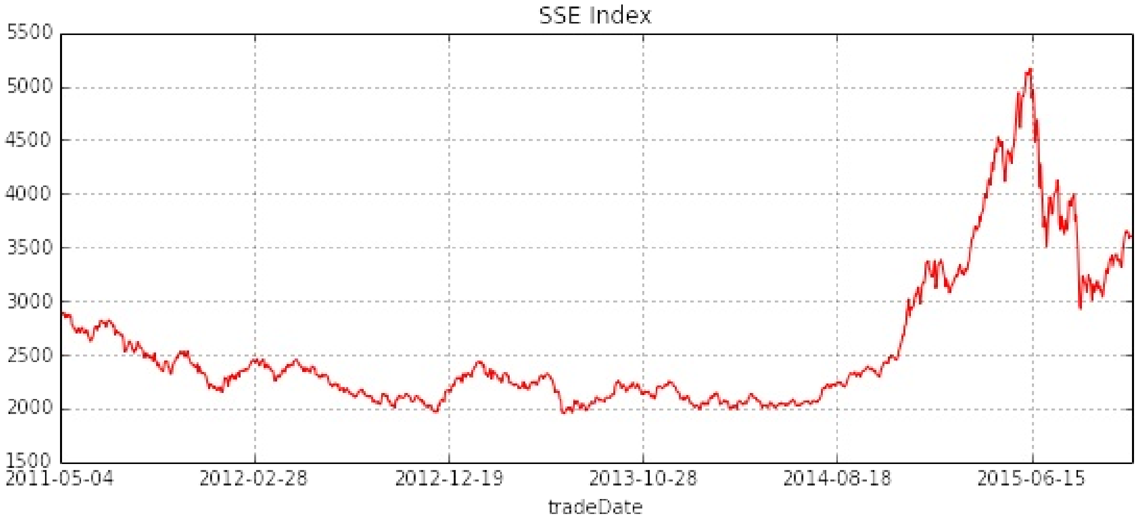
```
inter = 320 #滑动时间窗口
#设置时间
today = Date.todaysDate()
beginDate = '20100101'
endDate = today.toDateTime().strftime('%Y%m%d')

#设置指数类型
indexLabel = '000001' # SSE index
#indexLabel = '399006' # CYB index

#读取指数
indexPrice = DataAPI.MktIdxdGet(ticker=indexLabel,beginDate=begi
nDate,endDate=endDate,field=["tradeDate","closeIndex"],pandas="1"
)
price = np.array(indexPrice.loc[:,'closeIndex'])

#计算对数收益
back_price = np.append(price[0],price.copy())
back_price = back_price[:-1]
return_price = np.log(price) - np.log(back_price)

#计算波动率 from GARCH(1,1)
am = arch_model(return_price)
res = am.fit()
sqt_h = res.conditional_volatility

#去除波动性
f = return_price/sqt_h

#计算hurst指数,函数来自自定义library
hurst = Hurst(f,T=inter,step=1,q=2,Smin=10,Smax=50,Sintr=1)

indexPrice['Hurst'] = pd.DataFrame(np.array([0] * len(indexPrice
)))
indexPrice.loc[inter-1:,'Hurst'] = hurst
indexPrice.index = indexPrice['tradeDate']

Iteration:      1,   Func. Count:      6,   Neg. LLF: -4149.5646
3466
Optimization terminated successfully.    (Exit mode 0)
            Current function value: -4151.74496903
            Iterations: 1
            Function evaluations: 17
            Gradient evaluations: 1
```

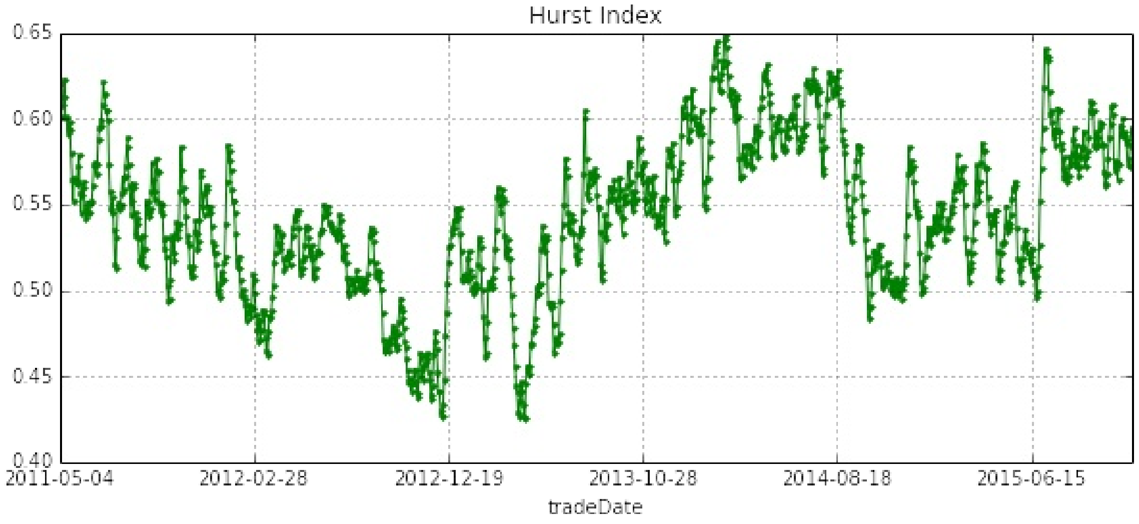
```
plt.figure(figsize=(10,6))
plt.subplot(3,1,1)
plt.plot(f)
plt.subplot(3,1,2)
plt.plot(return_price)
plt.subplot(3,1,3)
plt.plot(sqt_h)

[<matplotlib.lines.Line2D at 0x95065d0>]
```



上面的图能够看到 `garch(1,1)` 到底做了什么，它主要是对波动率进行了建模，在做分析时消去了这部分的影响。

```
plt.figure(1)
indexPrice['closeIndex'].tail(len(indexPrice)-inter).plot(figsiz
e=(10,4),color='red',title='SSE Index',linewidth=1)
plt.figure(2)
indexPrice['Hurst'].tail(len(indexPrice)-inter).plot(figsize=(10,
4),color='green',title='Hurst Index',linewidth=1,marker='.')

<matplotlib.axes.AxesSubplot at 0x95ae390>
```





- 看出了啥没？简单点说，就是hurst越大，越有可能延续之前的趋势（即动量），若hurst越小，则越有可能违反之前的趋势（即反转）。LADISLAV KRISTOUFEK这位教授的想法是通过极大极小值来判断，当然它分析的是美股啦。
- 再看看上面的图，是对上证指数的分析，取的是日线的数据（其实我喜欢用分钟线，因为A股波动辣么牛逼，日线颗粒度哪里够啊。。），可以得（meng）出这些结论：
    - 13年中旬hurst出现最小值，说明熊市的跌势要反转了，马上要进入牛市了？！
    - 15年中旬hurst出现最小值，说明牛市的涨势要反转了，马上要进入熊市了？！
- 算卦完毕。

5.13 变点理论 · 变点策略初步

> 来源：https://uqer.io/community/share/5667d533f9f06c6c8a91b61c

寻找变点

1.变点理论

变点理论是统计学中的一个经典分支，其基本定义是在一个序列或过程中，当某个统计特性（分布类型、分布参数）在某时间点受系统性因素而非偶然性因素影响发生变化，我们就称该时间点为变点。变点识别即利用统计量或统计方法将该变点位置估计出来。

CUSUM图作为工业应用检验变点的三大控制图之一，其利用假设检验和极大似然估计的相关统计原理，构建累积和统计量，不断累积观察值与基线水平的差值，将微小偏差累积，放大观察数据出现的波动，从而更加迅速敏感地探测到微小的异常情况，检验出变点位置。其最大的特点是对系统性变化的敏感性，不需要积累太多的样本，因而能较好的控制风险。

基于变点CUSUM图的基本原理，以股价对数收益率符合局部正态分布为基本原理，构建CUSUM的上下统计量，一旦统计量突破阈值即判断出现变点。以股价上升时对数收益率出现上升变点作为买入时机，以股价下降时对数收益率出现下降变点作为卖出时机。统计量中的两个参数，允偏量k设置为动态变化自适应的形式，阈值h则需要根据直观进行设定。

2.CUSUM原理

CUSUM控制图的设计思想是对信息加以累积，将过程的小偏移累加起来，达到放大的结果，从而提高检验小偏移的灵敏度。CUSUM作为一个统计量，其由来具有严格的数学推理，总的来说，是一个变点假设检验通过极大似然法推导得到的统计量。

令 `xi` （it）为独立的 `N(δ,1)` 同分布，其中 `t` 为未知变点，对于给定的观察序列 `xn` ，假设 `t=v` ，如此构成一个假设检验问题:

$$H_0 : X_i \text{-} N(0,1), i = 1,2,\ldots,n \tag{1}$$

$$H_1 : \exists v < \infty, s.t. X_i \text{-} N(0,1), i = 1,2,\ldots,n; X_i \text{-} N(\delta,1), i = v, v+1,\ldots,n \tag{2}$$

则似然比统计量为(以 `Φ(` 表示标准正态分布 `N(0,1)` 的分布密度函数）:

$$L_{n,v} = \frac{\prod_{i=1}^{v}\phi(x_i)\prod_{j=v+1}^{n}\phi(x_j-\delta)}{\prod_{i=1}^{n}\phi(x_i)} = \frac{\prod_{j=v+1}^{n}\phi(x_i-\delta)}{\prod_{i=v+1}^{n}\phi(x_i)} = \exp\left\{\delta\sum_{i=v+1}^{n}(x_i-\frac{\delta}{2})\right\}$$

对数化为：

$$\Lambda_{n,v} = \ln \mathrm{L}_{n,v} = \delta \sum_{i=v+1}^{n} (x_i - \frac{\delta}{2})$$

假设变量有偏移，则其对数似然统计量为:

$$\Lambda_n = \max_{1 \le v < n} \Lambda_{n,v} = \max\{\delta \sum_{i=v+1}^{n} (x_i - \frac{\delta}{2})$$

若我们检验的为向上偏移，即 `δ>0` ，上述的对数似然统计量等价于下面统计量

$$Z_n = \max_{1 \le v < n} \sum_{i=v+1}^{n} (x_i - \frac{\delta}{2})$$

设 `n-1` 个观测值没有均值偏移，即

$$Z_i \le h, i = 1, 2, \ldots, n-1$$

`h` 为门限。如果在时刻 `n` ，满足:

$$\left\{ \begin{array}{cr} x_n - \frac{\delta}{2} > h & \\ x_n + x_{n+1} - \delta > h & (3) \\ x_n + x_{n-1} + x_{n-2} - \frac{3\delta}{2} > h & (4) \\ \ldots & \\ \ldots & \\ x_n + x_{n-1} + \ldots + x_1 - \frac{n\delta}{2} > h & \end{array} \right\}$$

则这个过程发生了均值偏移。以下记号推导有:

$$\begin{aligned} & \widetilde{x}_i = x_i - \frac{\delta}{2}, \widetilde{x}_0 = 0, \bar{s}_k = \sum_{i=0}^{\kappa} \widetilde{x} && (5) \\ & Z_n - Z_{n-1} = \widetilde{x}_n - \min\{0, \bar{s}_n - \min_{0 \le v \le n-1} \bar{s}_v\} && (6) \\ & = max\{\widetilde{x}_n, \widetilde{x}_n - \bar{s}_n + \min_{0 \le v \le n-1} \bar{s}_v\} && (7) \\ & = \max\{\widetilde{x}_n, \min_{0 \le v \le n-1} \bar{s}_v - \bar{s}_n\} && (8) \\ & = \max\{\widetilde{x}_n, -Z_{n-1}\} && (9) \end{aligned}$$

用不定参数 `k` 代替 `δ/2` ，就得到了 `Zn` 的递推公式：

$$Z_n = \max\{0, Z_{n-1} + x_n - k\}, n = 1, 2, \ldots$$

若设定报警门限为 `h>0` ，如果在第 `n` 个观察点满足：

$$Z_n > h(Z_i \leq h, i = 1, 2, \ldots, n-1)|$$

则报警，确定在 `n` 以前的统计量发生了均值向上偏移，判断有系统性因素而非偶然性因素存在。 向下偏移也可通过类似的推导得到。

3.具体应用

由CUSUM的推导分析，我们得到CUSUM统计量，即：

$$S_i = \sum_{i=1}^{l} (X_i - k) = S_{i-1} + (X_i - k)|$$

其中 `X` 为金融序列，令 `yi=Xi-k` ，其中k为允偏量。上下预警指标分别为

$$\left\{ \begin{array}{l} C_i = \max(C_{i-1} + y_i, 0) \\ C'_i = \min(C'_{i-1} + y_i, 0) \end{array} \right\}|$$

若预警指标分别达到上下阈值 `h` ，则对应地产生上变点和下变点。

```
Ci≥h,C'i≤-h
```

4.参数设定

（1）允偏量k值

经过正态分布标准化后的假设检验，即变点前的数据服从 `N(0,1)` 分布，变点后的数据服从 `N(δ,1)` 分布。累积和控制图的算法设计是基于既定的观测值偏移量，即控制图参数 `k` ，理论上 `k=δ/2` 时控制图的效果最好，其中， `δ` 即为观测值 `X` 的偏移量。但是实际情形下， `δ` 是未知量，也是我们需要检测的值，同时 `δ` 会随采样时间 `t` 变化，其大小决定了累积和控制图参数 `k` 的取值，并通过 `k` 的变化影响控制图的统计性能，所以有必要对 `δ` 进行动态预测和更新。

这里基于已检测的历史数据和待检测的点作一个动态设定。运用已知历史序列拟合参数 `mu` 和 `sigma` ，再将历史数据和待检测点进行标准正态化，则历史数据的均值为0，而待测点的值为 `x` 。在某种程度上，可将标准化后的历史数据看作 `N(0,1)` 分布，而将待检测点看作 `N(x,1)` 分布，按照CUSUM的推导，设置 `k=x/2`

2）阈值 `h` 在参数检验中，为使得观测数据尽量复合假设，需要将观测到的数据正态标准化，所以阈值h的设置在一定程度上相当于标准正态分布参数的选定，参考布林带的做法，我们将 `h` 定为2（布林带标准差的倍数一般为2）。

5.算法步骤

a) 选取数据段 `x0` ，计算对数收益率 `r` ；

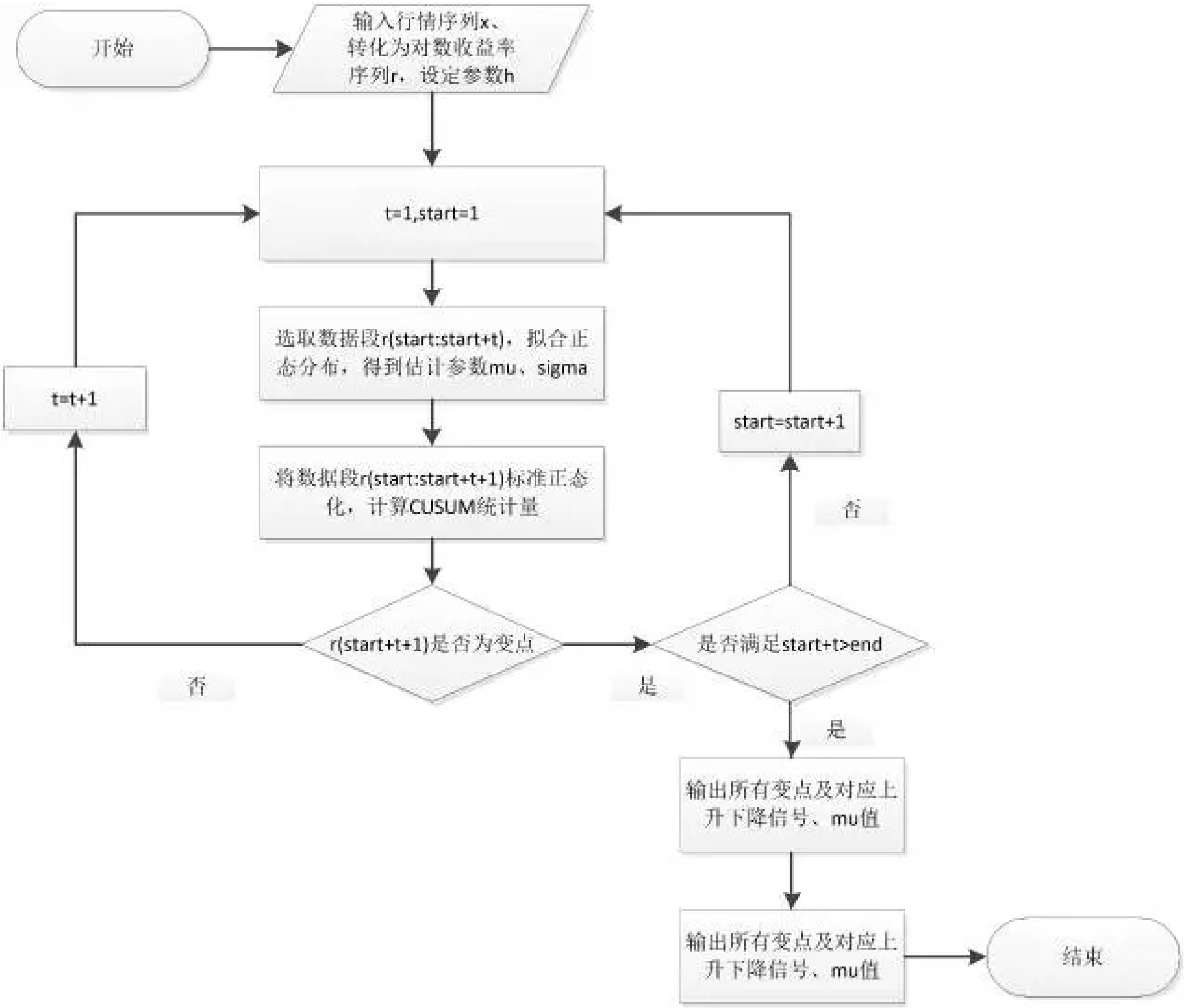
b) 从数据段 `x0` 的第一个数据开始，以第一第二个数据为初始数据段 `startdata` ，进行正态分布拟合，得到均值 `mu` 和标准差 `sigma` ，选择数据段外的第一个数据（整体样本的第三个数据），合成实验数据段data，并利用 `mu` 和 `sigma` 值进行归一化，得到标准正态分布序列（近似） `x` ；

c) 允偏量 `k` 采取动态变化，为归一化后序列 `x` 的最后一个数值的一半，计算上下CUSUM统计量，判断 `x` 序列最后一个数是否超出CUSUM阈值，若超出，对 `x` 序列最后一个点进行变点标记；若不超出，往后迭代，直至出现变点标记为止；

d) 从 `x0` 第二个点开始，重现选择数据段，依照a)、b)步骤标记变点，直至标记完所有的起点为止；

e) 若该点存在标记数，则判断该点为变点；

f) 在变点中选择处于股价上升路径的上升变点以及股价处于下降路径中的下降变点，以此确定拐点和拐点方向。 流程图如下：


开始
输入行情序列x、转化为对数收益率序列r，设定参数h
t=1,start=1
选取数据段r(start:start+t)，拟合正态分布，得到估计参数mu、sigma
将数据段r(start:start+t+1)标准正态化，计算CUSUM统计量
t=t+1
start=start+1
否
r(start+t+1)是否为变点
是否满足start+t>end
否
是
是
输出所有变点及对应上升下降信号、mu值
输出所有变点及对应上升下降信号、mu值
结束

6.初步程序

原理部分来源于长江金工（作者：刘胜利），更加全面的可见（http://mp.weixin.qq.com/s?__biz=MzA3ODIyNjMzNA==&mid=210838858&idx=1&sn=e850063b81844cac4ff96177354a9535&scene=1&srcid=1110LMZHGjtVUAIzMhUYGvKG#rd ）

根据上述原理初步编了一个寻找变点的程序（好几周之前了=_=,之前把所有的变点都成功得到过，只不过忘了加对于价格的判断），但是不会把它变成策略。不知道为什么，原来可以成功print出变点的程序现在运行不完，一直处于运行中。原来不加价格的判断运行一会儿interrupt后有部分结果，，加了价格判断一直只有一个结果。。。看到社区有关于变点策略的讨论，就把自己之前写的Po出来(关于元素的引用我自己绕了好久+1.-1...)，希望社区的老师们批评指正~

```
import numpy as np
import pandas as pd
import scipy.stats as stats
from pandas import Series, DataFrame
data1=DataAPI.MktEqudGet(ticker="600837",beginDate='20140101',en
dDate='20151120',field='ticker,secShortName,tradeDate,openPrice'
,pandas="1")
price = data1['openPrice'].values                         #取出价格
lnp = np.log(price)                                   #取对数
h = 2                                            #设定参数h
R = []                                            #建立一个空列表用于存放
对数收益率
i = 1
while i < len(lnp):
    r = lnp[i]-lnp[i-1]
    i = i+1
    R.append(r)                                         #得到对数收益率序列
start = 1
t = 1
while start+t <= len(R)-1:
    d = R[(start-1):(start-1+t+1)]
    mu,sigma = stats.norm.fit(d)                             #用前面的数据
进行正态分布拟合
    OR = (R[start-1:start-1+t+1+1]-mu)/sigma                      #对该数
据段进行标准化
    k = OR[-1]/2                                        #允偏量设定为归一化
后序列的最后一个数值的一半
    j = 1
    s = 0
    while j < t+2:                                      #(start-1+t+1+1)
-(start-1)=t+2
        s = s + OR[j-1]-k                                #计算CUSUM
        j = j+1
    if s>0:                                           #判断正负
        c1 = max(s,0)
        if price[start+t] > price[start+t-1] and c1 > h:          #
判断是否价格上升并冲破阈值，大于h则买入
```

```
            print 'c1 = ',c1,start+t                 #start+t即
为第(start+t)个变点
            start = start+1
        elif c1 < h:
            t = t+1                             #继续向后迭代直至出
现变点为止
    elif s<0:
        c2 = min(s,0)
        if price[start+t] < price[start+t-1] and c2 < -h:     #
判断是否价格下降并冲破阈值，小于-h则卖出
            print 'c2 = ',c2,start+t
            start = start+1
        elif c2 > -h:
            t = t+1

c2 =  -2.66368724271 3
```

5.14 Z-score Model

Zscore Model Tutorial

> 来源：https://uqer.io/community/share/54ab4407f9f06c276f6519ec

1. 什么是 Zscore Model

信用风险评分方法通常包括定性法、单变量法、多变量法，目前广泛采用多变量方法包括如判别分析(discriminant analysis)、逻辑回归(logit regression)和非线性模型如神经网络等。奥特曼博士于1968年发表的Z-score模型基于多变量判别分析方法。

Z-score模型基于各变量加权得分对企业是否破产进行判断，在Z-score原始模型中，得分高于2.99的属于“安全”区域、低于1.80的属于“困境”区域，两个得分之间的属于“灰色”区域。

Z-score模型在美国企业如柯达、通用汽车的破产预测得到了很好的结果。

2. 本模块提供的 Zscore Model

下面我们从原理，模型公式，划分区间三个方面来简单介绍一下本模块中的两个Z-score模型：

2.1 All Corporate Bonds

模型原理

- No equity prices are needed
- Uses discriminant analysis methodology
- Coefficients are obtained from China distressed firm dataset from historical periods

模型公式

```
ZScore = 0.517 - 0.460*TotalLiabilities/TotalAssets
           + 9.320*NetProfit/0.5*(TotalAssets+TotalAssets[last])
           + 0.388*WorkingCapital/TotalAssets
           + 1.158*RetainedEarnings/TotalAssets
         = 0.517 - 0.460*x1
           + 9.320*2/x2
           + 0.388*x3
           + 1.158*x4
```

划分区间

- `Z-score < 0.5`：已经违约
- `0.5 < Z-score < 0.9`：有违约的可能性
- `Z-score > 0.9`：财务健康，短期内不会出现违约情况

2.2 Corporate Bonds with Equity Listings

模型原理

- Uses Equity Prices: Information from equity set
- Uses discriminant analysis methodology
- Coefficients are obtained from China distressed firm dataset from historical periods

模型公式

```
ZScore = 0.2086*x1 + 4.3465*x2 + 4.9601*x3
             x1: market value/book value of TotalLiabilities
             x2: total sales/TotalAssets
             x3: (TotalAssets-TotalAssets[last])/TotalAssets[las
t]
coef = [0.2086, 4.3465, 4.9601]
```

划分区间

- `Z-score < 1.5408`：已经违约
- `Z-score > 1.5408`：财务健康，短期内不会出现违约情况

3. 如何实用本模块提供的 Zscore Model 接口

本模块目前提供了四个Z-score模型接口，分别如下：

3.1 `zscore_ACB`

```
---
Interface for calculating zscore using ACB.
parameter:
    ticker[string]: ticker code
parameter[opt]:
    begin[int]: year begin, default 2010
    end[int]: year end, default 2014
    coef[list of int]: coefficients for Zscore Model, default [0
.517, -0.460, 18.640, 0.388, 1.158]
---
when success, will return factors and status code, as a dict;
when failed for some reason, will return error message and error
 code, as a dict;
---
Model ACB:
    All Corporate Bonds
    ZScore = 0.517 - 0.460*TotalLiabilities/TotalAssets
             + 9.320*NetProfit/0.5*(TotalAssets+TotalAssets[last
])
             + 0.388*WorkingCapital/TotalAssets
             + 1.158*RetainedEarnings/TotalAssets
           = 0.517 - 0.460*x1
                + 9.320*2/x2
                + 0.388*x3
                + 1.158*x4
    coef = [0.517, -0.460, 18.640, 0.388, 1.158]
```

3.2 zscore_ACB_List

```
---
Interface for calculating zscore using ACB.
parameter:
    ticker[list]: ticker code
parameter[opt]:
    begin[int or list]: year begin, default 2010
    end[int or list]: year end, default 2014
    coef[list of int]: coefficients for Zscore Model, default [0
.517, -0.460, 18.640, 0.388, 1.158]
---
```

3.3 zscore_ACBEL

```
---
Interface for calculating zscore using ACBEL.
parameter:
    ticker[string]: ticker code
parameter[opt]:
    begin[int]: year begin, default 2010
    end[int]: year end, default 2014
    coef[list of int]: coefficients for model, default [0.2086,
4.3465, 4.9601]
---
when success, will return factors and status code, as a dict;
when failed for some reason, will return error message and error
 code, as a dict;
---
Model ACB:
    All Corporate Bonds with Equity Listings
    ZScore = 0.2086*x1 + 4.3465*x2 + 4.9601*x3
            x1: market value/book value of TotalLiabilities
            x2: total sales/TotalAssets
            x3: (TotalAssets-TotalAssets[last])/TotalAssets[last
]
    coef = [0.2086, 4.3465, 4.9601]
---
```

3.4 `zscore_ACBEL_List`

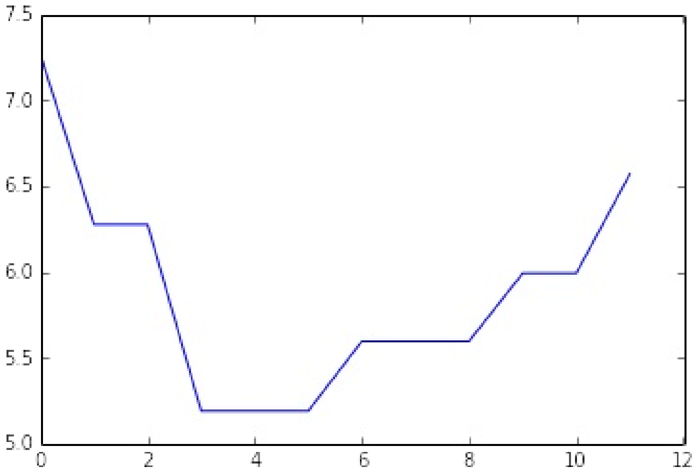
```
---
Interface for calculating zscore using ACBEL.
parameter:
    ticker[list]: ticker code
parameter[opt]:
    begin[int or list]: year begin, default 2010
    end[int or list]: year end, default 2014
    coef[list of int]: coefficients for Zscore Model, default [0
.2086, 4.3465, 4.9601]
---
```

4. 一个 Zscore Model 实例

相关步骤说明如下：

- 4.1 导出该模块
- 4.2 输入股票代码，可选择性输入计算时间区间，以年为单位
- 4.3 得到计算结果
- 4.4 作图分析

```
# 4.1 & 4.2
GZMT = zscore_ACB('600519')

# 4.3
factors = GZMT['factors']

# 4.4
from matplotlib.pylab import   plot
plot(factors['zscore'])

[<matplotlib.lines.Line2D at 0x4eda750>]
```



5. 本模块未来版本规划

为适应光大金融人士更加方便地使用 Z-Score Model，本模块将在以下几个方面对本模块的未来版本进行规划：

- 更加简单：提供上传模板，用户只需按照一定格式上传自己持仓的 excel 表格，即可在零编码的情况下得到相应持仓债券、股票发行人的 Z-Score 情况；
- 更加灵活：将会允许用户在我们定义的两个 Z-Score 模型的相关系数；
- 更加智能：预计提供给用户将近50个公司财务相关的因子，由用户自定义因子和相关的系数来计算 Z-Score 值；

信用债风险模型初探之：Z-Score Model

> 来源：https://uqer.io/community/share/568b73d6228e5b67159bee69

0. 引言

2015年3月4日晚间，ST超日（上海超日太阳能科技股份有限公司）董事会发布公告称，“11超日债”本期利息将无法于原定付息日2014年3月7日按期全额支付。至此，“11超日债”正式成为国内首例违约债券。

1. 什么是 Zscore Model

简单的说，zscore model 是一种用于估计债券发行人违约风险的信用风险模型。

2. 本文提供的 Zscore Model

下面我们首先从原理，模型公式，划分区间三个方面来简单介绍一下本模块中的两个 Z-score 模型； 然后我们从如何获取数据，清洗数据；如何计算债券发行人 Z-score 值；如何作图来操练这个模型；

2.1 All Corporate Bonds

模型原理

- 不需要发行人上市交易数据
- 离散分析方法

模型公式

```
ZScore = 0.517 - 0.460*x1 + 9.320*2/x2 + 0.388*x3 + 1.158*x4

x1: 负债合计/资产总计
x2: 净利润/0.5*(资产总计 + 资产总计[上期])
x3: 营运资本/资产总计
x4: 未分配利润/资产总计

coef=[0.517, -0.460, 18.640, 0.388, 1.158]
```

划分区间

- `Z-score < 0.5`：已经违约
- `0.5 < Z-score < 0.9`：有违约的可能性

- `Z-score > 0.9`：财务健康，短期内不会出现违约情况

2.2 Corporate Bonds with Equity Listings

模型原理

- 需要发行人上市交易数据
- 离散分析方法

模型公式

```
ZScore = 0.2086*x1 + 4.3465*x2 + 4.9601*x3

x1: 总市值/负债合计
x2: 营业总收入/资产总计
x3: (资产总计-资产总计[上期])/资产总计[上期]

coef = [0.2086, 4.3465, 4.9601]
```

划分区间

- `Z-score < 1.5408`：已经违约
- `Z-score > 1.5408`：财务健康，短期内不会出现违约情况

3. 未来

Z-Score 是一个比较基础，通用的模型，本文只是对其原理和实现的一个简单探索，要想真正 build 一个足够 robust 的模型还需要做很多工作。不过 uqer 提供了完善的财务数据，行情数据以及 100 多个相关因子，相信会给大家建立模型上节省不少时间。

```py

# built-in package

import time import json import random import datetime as dt

# third-party package

import numpy as np import pandas as pd pd.options.display.max_columns = 100 pd.options.display.max_rows = 100 from matplotlib.pyplot import * import seaborn
```

user-defined package

Const Variable

~~~
获取数据，清洗数据

```py
def data_for_acb(ticker="000001", tstart=2010, tend=2015):
    """获取，清洗 ACB 模型所需要的数据。

    ACB 模型需要数据：
        负债合计（TLiab）
        资产总计（TAssets）
        未分配利润（retainedEarnings）
        净利润（NIncome）
        营运资本 = 资产总计 - 负债合计
    """
    bs_data = DataAPI.FdmtBSGet(ticker=ticker, beginYear=tstart-
1, endYear=tend,
                                field=['secID', 'endDate', 'publ
ishDate', 'TLiab', 'TAssets', 'retainedEarnings'])
    is_data = DataAPI.FdmtISGet(ticker=ticker, beginYear=tstart-
1, endYear=tend,
                                field=['secID', 'endDate', 'publ
ishDate', 'NIncome'])
    bs_data = bs_data.drop_duplicates('endDate')
    is_data = is_data.drop_duplicates('endDate')

    data = is_data.merge(bs_data, on=['secID', 'endDate'])

    # calculate TAssets diff of current and last report
    pre_TAssets = []
    length = len(data)

    for index, number in enumerate(data.TAssets):
        if index + 1 == length:
            last_number = index
        else:
            last_number = index + 1
        pre_TAssets.append(data.TAssets[last_number])

    data['TAssetsPre'] = pre_TAssets

    return data

def data_for_acbel(ticker="000001", tstart=2010, tend=2015):
~~~

```
    """获取，清洗 ACB 模型所需要的数据。

    ACB 模型需要数据：
        负债合计（TLiab）
        资产总计（TAssets）
        未分配利润（retainedEarnings）
        净利润（NIncome）
        营业总收入（tRevenue）
        总市值（marketValue）
    """
    bs_data = DataAPI.FdmtBSGet(ticker=ticker, beginYear=tstart-
1, endYear=tend,
                                field=['secID', 'endDate', 'publ
ishDate', 'TLiab', 'TAssets', 'retainedEarnings'])
    is_data = DataAPI.FdmtISGet(ticker=ticker, beginYear=tstart-
1, endYear=tend,
                                field=['secID', 'endDate', 'publ
ishDate', 'NIncome', 'tRevenue'])
    market_data = DataAPI.MktEqudGet(ticker=ticker,  field=['sec
ID', 'tradeDate', 'marketValue'])
    market_data.rename(columns={'tradeDate': 'endDate'}, inplace
=True)
    bs_data = bs_data.drop_duplicates('endDate')
    is_data = is_data.drop_duplicates('endDate')

    data = is_data.merge(bs_data, on=['secID', 'endDate'])
    endDate = list(data.endDate)
    data = data.merge(market_data, on=['secID', 'endDate'], how=
'outer')
    data.marketValue = data.marketValue.fillna(method='ffill')
    data = data[data.endDate.isin(endDate)]
    # calculate TAssets diff of current and last report
    pre_TAssets = []
    length = len(data)

    for index, number in enumerate(data.TAssets):
        if index + 1 == length:
            last_number = index
        else:
            last_number = index + 1
        pre_TAssets.append(data.TAssets[last_number])

    data['TAssetsPre'] = pre_TAssets

    return data
```

测试：获取数据，清洗数据

```
data_for_acb("002056").head(5)
```

	secID	endDate	publishDate_x	NIncome	publishD
0	002056.XSHE	2015-09-30	2015-10-28	2.657156e+08	2015-10-
1	002056.XSHE	2015-06-30	2015-08-27	1.482967e+08	2015-08-
2	002056.XSHE	2015-03-31	2015-04-27	6.872527e+07	2015-04-
3	002056.XSHE	2014-12-31	2015-03-28	3.814308e+08	2015-10-
4	002056.XSHE	2014-09-30	2015-10-28	9.175491e+07	2014-10-

```
data_for_acbel("002056").head(5)
```

	secID	endDate	publishDate_x	NIncome	tRever
0	002056.XSHE	2015-09-30	2015-10-28	2.657156e+08	2.882585
1	002056.XSHE	2015-06-30	2015-08-27	1.482967e+08	1.800482
2	002056.XSHE	2015-03-31	2015-04-27	6.872527e+07	8.511258
3	002056.XSHE	2014-12-31	2015-03-28	3.814308e+08	3.668800
4	002056.XSHE	2014-09-30	2015-10-28	9.175491e+07	9.342650

计算 Z-score

```
def zscore_ACB(ticker=None, tstart=2010, tend=2015, coef=[0.517, -0.460, 18.640, 0.388, 1.158]):
    # step 1. get data and pre-calculate the factor
    ticker = data_for_acb(ticker, tstart, tend)
    ticker['x0'] = 1
    ticker['x1'] = ticker['TLiab'] / ticker['TAssets']
    ticker['x2'] = ticker['NIncome'] * 2 / (ticker['TAssets'] + ticker['TAssetsPre'])
    ticker['x3'] = (ticker['TAssets'] - ticker['TLiab']) / ticker['TAssets']
    ticker['x4'] = ticker['retainedEarnings'] / ticker['TAssets']
    # step 2. calculate zscore
    tmp = ticker[['x0', 'x1', 'x2', 'x3', 'x4']] * coef
    ticker['zscore'] = tmp.sum(axis=1)

    # step 3. build result
    ticker.sort('endDate', ascending=True, inplace=True)
    return ticker[['secID', 'endDate', 'NIncome', 'TLiab', 'TAss
ets',
                   'retainedEarnings', 'x0', 'x1', 'x2', 'x3', '
x4', 'zscore']]


def zscore_ACBEL(ticker=None, tstart=2010, tend=2015, coef=[0.20
86, 4.3465, 4.9601]):
    # step 1. get data and pre-calculate the factor
    ticker = data_for_acbel(ticker, tstart, tend)
    ticker['x0'] = ticker['marketValue'] / ticker['TLiab']
    ticker['x1'] = ticker['tRevenue'] / ticker['TAssets']
    ticker['x2'] = (ticker['TAssets'] - ticker['TAssetsPre']) /
ticker['TAssetsPre']

    # step 2. calculate zscore
    tmp = ticker[['x0', 'x1', 'x2']] * coef
    ticker['zscore'] = tmp.sum(axis=1)

    # step 3. build result
    ticker.sort('endDate', ascending=True, inplace=True)
    return ticker[['secID', 'endDate', 'NIncome', 'TLiab', 'tRev
enue', 'TAssets',
                   'retainedEarnings', 'marketValue', 'TAssetsPr
e', 'x0', 'x1', 'x2', 'zscore']]

def get_ticker(bond=None):
    """Get the ticker number of a bond.
    """
    # bondID -> partyID -> ticker
    partyID = None
    try:
        data = DataAPI.BondGet(ticker=bond)
        partyID = data['partyID'][0]
    except:
        return 'Cannot find this bond in DataAPI'

    ticker = None
    try:
        data = DataAPI.SecIDGet(partyID=str(partyID))
        ticker = data['ticker'][0]
    except:
        return 'Cannot find the ticker for this bond in DataAPI,
 maybe the issuer is not listed'

    return ticker
```

测试：计算 Z-score

```
zscore_ACB("002506").head(5)
```

	secID	endDate	NIncome	TLiab	TAss
21	002506.XSHE	2009-12-31	1.699573e+08	7.039600e+08	1.26261
20	002506.XSHE	2010-09-30	1.051847e+08	1.131338e+09	1.84896
19	002506.XSHE	2010-12-31	2.194191e+08	1.398571e+09	4.46659
18	002506.XSHE	2011-03-31	3.719796e+07	1.841932e+09	4.94695
17	002506.XSHE	2011-06-30	1.313367e+08	2.631518e+09	5.72884

```
zscore_ACBEL("002506").head(5)
```

	secID	endDate	NIncome	TLiab	tReve
21	002506.XSHE	2009-12-31	1.699573e+08	7.039600e+08	1.31824
20	002506.XSHE	2010-09-30	1.545466e+08	1.131338e+09	1.64622
19	002506.XSHE	2010-12-31	2.194191e+08	1.398571e+09	2.68664
18	002506.XSHE	2011-03-31	3.719796e+07	1.841932e+09	6.50264
17	002506.XSHE	2011-06-30	1.313367e+08	2.631518e+09	1.79788

```
a = zscore_ACBEL("002506")
a.head(3)
```

	secID	endDate	NIncome	TLiab	tReve
21	002506.XSHE	2009-12-31	1.699573e+08	7.039600e+08	1.318242
20	002506.XSHE	2010-09-30	1.051847e+08	1.131338e+09	6.18418
19	002506.XSHE	2010-12-31	2.194191e+08	1.398571e+09	2.68664

作图分析

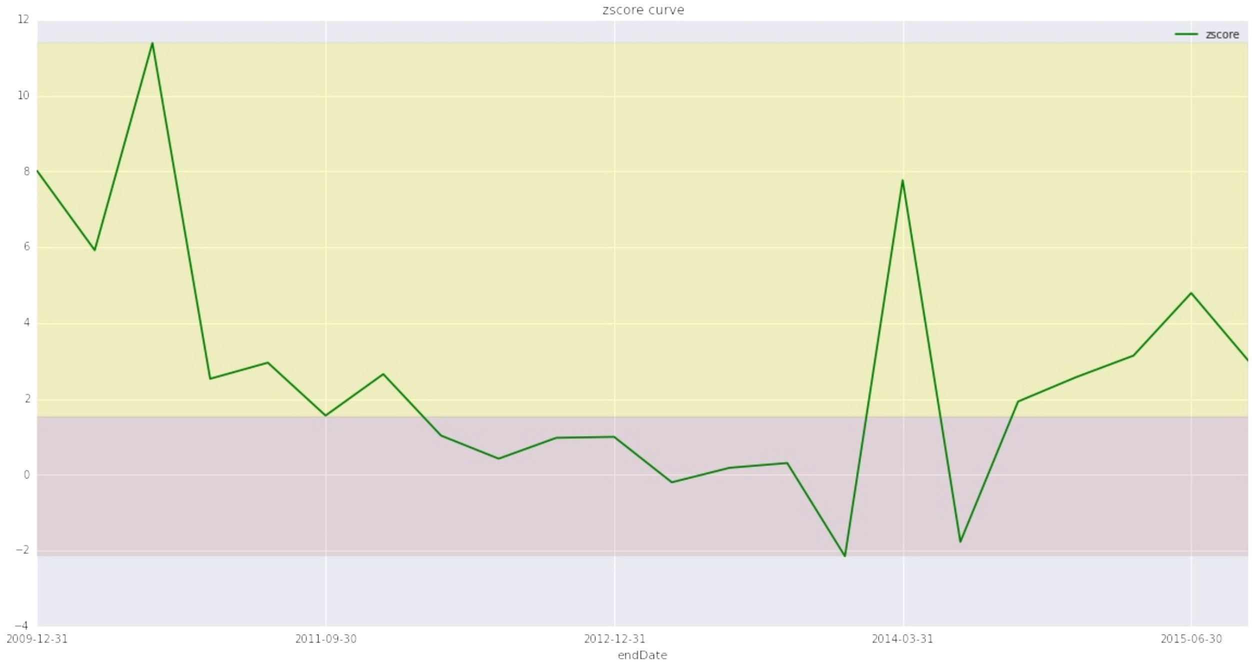
```
def zscore_plot(dataframe, upper_limit, low_limit):
    ax = dataframe.plot('endDate', ['zscore'], figsize=(20, 10),
 style='g-', title='zscore curve',)
    axhspan(low_limit, dataframe.zscore.min(), facecolor='maroon'
, alpha=0.1)
    axhspan(upper_limit, dataframe.zscore.max(), facecolor='yell
ow', alpha=0.2)
    ax.legend()
    return ax
```

测试：作图分析

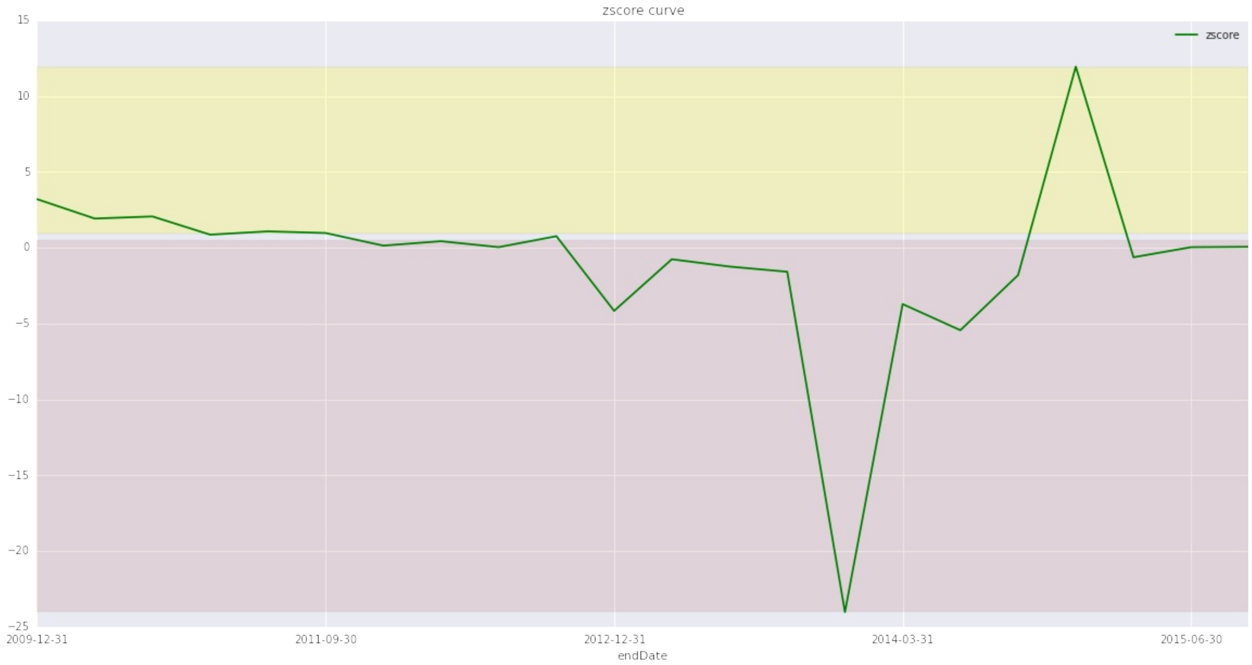
这里，我们以 11超日债〔112061〕来做测试，看看当前这个模型表现怎么样。

- step 1: 调用 `get_ticker` 函数通过债券代码获取发行人上市代码，在发行人已上市的前提下；
- step 2: 调用 `zscore_ACBEL` 或 `zscore_ACB` 计算发行人的 Z-score 值；
- step 3: 调用 `zscore_plot` 绘制 Z-score 曲线；

```
ticker = get_ticker("112061")
df = zscore_ACBEL(ticker)
zscore_plot(df, 1.5408, 1.5408)

<matplotlib.axes.AxesSubplot at 0x5220a10>
```



```
ticker = get_ticker("112061")
df = zscore_ACB(ticker)
zscore_plot(df, 0.9, 0.5)

<matplotlib.axes.AxesSubplot at 0x525c610>
```



5.15 机器学习 · Machine Learning 学习笔记（一） by OTreeWEN

> 来源：https://uqer.io/community/share/55bc40caf9f06c91f918c604

```
# 测试sklearn安装包是否已装好
import sklearn as sk
import numpy as np
import matplotlib.pylab as plt
# 加载数据
from sklearn import datasets
iris = datasets.load_iris()
X_iris, y_iris = iris.data, iris.target
# 将X 和 y 的shape 打印出来
print X_iris.shape, y_iris.shape
print 'X_iris =', X_iris[0]
print 'y_iris =',y_iris[0]

(150, 4) (150,)
X_iris = [ 5.1  3.5  1.4  0.2]
y_iris = 0
```

Any machine learning problem can be represented with the following three concepts:

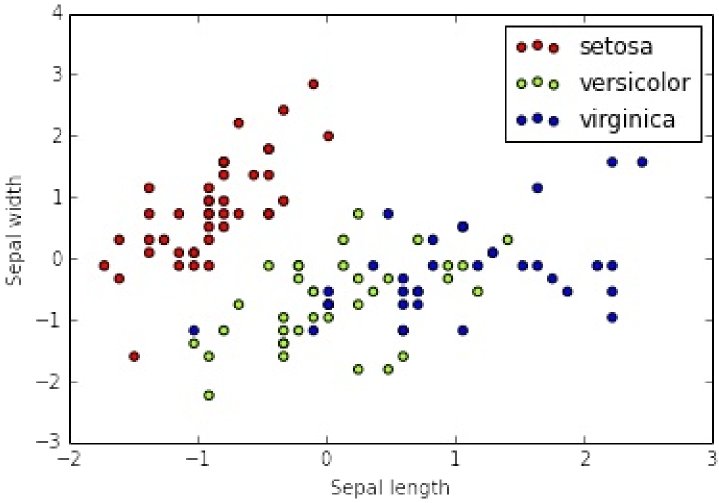
• We will have to learn to solve a task T. For example, build a spam filter that learns to classify e-mails as spam or ham.

• We will need some experience E to learn to perform the task. Usually, experience is represented through a dataset. For the spam filter, experience comes as a set of e-mails, manually classified by a human as spam or ham.

• We will need a measure of performance P to know how well we are solving the task and also to know whether after doing some modifications, our results are improving or getting worse. The percentage of e-mails that our spam filtering is correctly classifying as spam or ham could be P for our spam-filtering task.

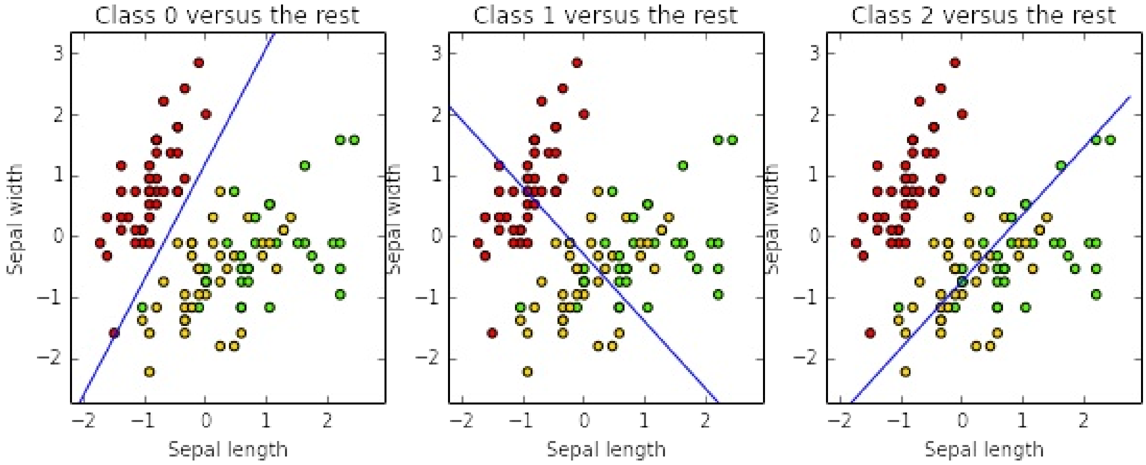
Our first machine learning method – linear classification

```
# ex1 linear classification
from sklearn.cross_validation import train_test_split
from sklearn import preprocessing
# Get dataset with only the first two attributes
X, y = X_iris[:, :2], y_iris
# Split the dataset into a training and a testing set
# Test set will be the 25% taken randomly
X_train, X_test, y_train, y_test = train_test_split(X, y,test_si
ze=0.25, random_state=33)
# The train_test_split function automatically builds the training

# and evaluation datasets, randomly selecting the samples.
scaler = preprocessing.StandardScaler().fit(X_train) # Standardi
ze the features
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

print 'X_train',X_train.shape # 75% of sample
print 'X_test',X_test.shape   # 25% of sample

import matplotlib.pyplot as plt
colors = ['red', 'greenyellow', 'blue']
for i in xrange(len(colors)):
    xs = X_train[:, 0][y_train == i]
    ys = X_train[:, 1][y_train == i]
    plt.scatter(xs, ys, c=colors[i])

plt.legend(iris.target_names)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')

# from sklearn.linear_modelsklearn._model import SGDClassifier
from sklearn.linear_model import SGDClassifier
# SGD stands for Stochastic Gradient Descent
clf = SGDClassifier()
clf.fit(X_train, y_train) # three-class problem
print 'shape of coef =',clf.coef_.shape
print 'shape of intercept =',clf.intercept_.shape

X_train (112, 2)
X_test (38, 2)
shape of coef = (3, 2)
shape of intercept = (3,)
```



The following code draws the three decision boundaries and lets us know if they worked as expected:

```
# 设定画图的刻度边界
x_min, x_max = X_train[:, 0].min() - .5, X_train[:, 0].max() + .5

y_min, y_max = X_train[:, 1].min() - .5, X_train[:, 1].max() + .5


xs = np.arange(x_min, x_max, 0.5)
fig, axes = plt.subplots(1, 3)
fig.set_size_inches(10, 6)

for i in [0, 1, 2]:

    # 设定图表i的标识、刻度等
    axes[i].set_aspect('equal')
    axes[i].set_title('Class '+ str(i) + ' versus the rest')
    axes[i].set_xlabel('Sepal length')
    axes[i].set_ylabel('Sepal width')
    axes[i].set_xlim(x_min, x_max)
    axes[i].set_ylim(y_min, y_max)

    plt.sca(axes[i]) # 选择图表 i
    # 画散点图
    plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=pl
t.cm.prism)
    # 计算分割线
    ys = (- clf.intercept_[i] - xs * clf.coef_[i, 0]) / clf.coef
_[i, 1]
     # 画分割线
    plt.plot(xs, ys, hold=True)
```



suppose that we have a new flower with a sepal width of 4.7 and a sepal length of 3.1, and we want to predict its class. We just have to apply our brand new classifier to it (after normalizing!).

```
print clf.predict(scaler.transform([[4.7, 3.1]]))[0]
print clf.decision_function(scaler.transform([[4.7, 3.1]]))

0
[[ 19.77232705   8.13983962 -28.65250296]]
```

We want to be a little more formal when we talk about a good classifier. What does that mean? The performance of a classifier is a measure of its effectiveness. The simplest performance measure is accuracy: given a classifier and an evaluation dataset, it measures the proportion of instances correctly classified by the classifier.

```
# Evaluating the results
from sklearn import metrics
y_train_pred = clf.predict(X_train)
print metrics.accuracy_score(y_train, y_train_pred)

0.821428571429
```

Probably, the most important thing you should learn from this chapter is that measuring accuracy on the training set is really a bad idea. You have built your model using this data, and it is possible that your model adjusts well to them but performs poorly in future (previously unseen data), which is its purpose. This phenomenon is called overfitting, and you will see it now and again while you read this book. If you measure based on your training data, you will never detect overfitting. So, never measure based on your training data. This is why we have reserved part of the original dataset (the testing partition)—we want to evaluate performance on previously unseen data. Let's check the accuracy again, now on the evaluation set (recall that it was already scaled):

```
y_pred = clf.predict(X_test)
print metrics.accuracy_score(y_test, y_pred)

0.684210526316
```

Precision: This computes the proportion of instances predicted as positives that were correctly evaluated (it measures how right our classifier is when it says that an instance is positive).

Recall: This counts the proportion of positive instances that were correctly evaluated (measuring how right our classifier is when faced with a positive instance).

F1-score: This is the harmonic mean of precision and recall, and tries to combine both in a single number.

```
print metrics.classification_report(y_test, y_pred, target_names
=iris.target_names)

             precision    recall  f1-score   support

     setosa       1.00      1.00      1.00         8
 versicolor       0.43      0.27      0.33        11
  virginica       0.65      0.79      0.71        19

avg / total       0.66      0.68      0.66        38
```

Another useful metric (especially for multi-class problems) is the confusion matrix: in its (i, j) cell, it shows the number of class instances i that were predicted to be in class j. A good classifier will accumulate the values on the confusion matrix diagonal, where correctly classified instances belong.

```
print metrics.confusion_matrix(y_test, y_pred)
```

To finish our evaluation process, we will introduce a very useful method known as cross-validation. As we explained before, we have to partition our dataset into a training set and a testing set. However, partitioning the data, results such that there are fewer instances to train on, and also, depending on the particular partition we make (usually made randomly), we can get either better or worse results. Cross-validation allows us to avoid this particular case, reducing result variance and producing a more realistic score for our models. The usual steps for k-fold cross-validation are the following:

1. Partition the dataset into k different subsets.
2. Create k different models by training on k-1 subsets and testing on the remaining subset.

3. Measure the performance on each of the k models and take the average measure.

```
from sklearn.cross_validation import cross_val_score, KFold
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
# create a composite estimator made by a pipeline of the
# standarization and the linear model
clf = Pipeline([('scaler', StandardScaler()),('linear_model', SG
DClassifier())])
# create a k-fold cross validation iterator of k=5 folds
cv = KFold(X.shape[0], 5, shuffle=True, random_state=33)
# by default the score used is the one returned by score
# method of the estimator (accuracy)
scores = cross_val_score(clf, X, y, cv=cv)
print scores

[ 0.73333333  0.63333333  0.73333333  0.66666667  0.6        ]
```

We obtained an array with the k scores. We can calculate the mean and the standard error to obtain a final figure:

```
from scipy.stats import sem
def mean_score(scores):
    return ("Mean score: {0:.3f} (+/-{1:.3f})").format(np.mean(s
cores), sem(scores))
print mean_score(scores)

Mean score: 0.673 (+/-0.027)
```

Machine learning categories

Classification is only one of the possible machine learning problems that can be addressed with scikit-learn. We can organize them in the following categories:

• In the previous example, we had a set of instances (that is, a set of data collected from a population) represented by certain features and with a particular target attribute. Supervised learning algorithms try to build a model from this data, which lets us predict the target attribute for new instances, knowing only these instance features. When the target class belongs to a discrete set (such as a list of flower species), we are facing a classification problem.

• Sometimes the class we want to predict, instead of belonging to a discrete set, ranges on a continuous set, such as the real number line. In this case, we are trying to solve a regression problem (the term was coined by Francis Galton, who observed that the heights of tall ancestors tend to regress down towards a normal value, the average human height). For example, we could try to predict the petal width based on the other three features. We will see that the methods used for regression are quite different from those used for classification.

- Another different type of machine learning problem is that of unsupervised learning. In this case, we do not have a target class to predict but instead want to group instances according to some similarity measure based on the available set of features. For example, suppose you have a dataset composed of e-mails and want to group them by their main topic (the task of grouping instances is called clustering). We can use it as features, for example, the different words used in each of them.

5.16 DualTrust 策略和布林强盗策略

> 来源：https://uqer.io/community/share/564737ddf9f06c4446b48133

谁能够帮忙实现DualTrust策略和布林强盗策略(BollingerBandit)？@薛昆Kelvin

@lookis：

DualTrust：

```
start = '2014-01-01'                       # 回测起始时间
end = '2015-01-01'                         # 回测结束时间
benchmark = 'HS300'                        # 策略参考标准
universe = set_universe("CYB") # 证券池，支持股票和基金
capital_base = 100000                      # 起始资金
freq = 'm'                                 # 策略类型，'d'表示日间
策略使用日线回测，'m'表示日内策略使用分钟线回测
refresh_rate = 1                           # 调仓频率，表示执行hand
le_data的时间间隔，若freq = 'd'时间间隔的单位为交易日，若freq = 'm'时间
间隔为分钟

def initialize(account):                   # 初始化虚拟账户状态
    account.k1 = 0.7
    account.k2 = 0.7
    account.cache = {}
    account.holding_max = 10
    account.holding = 0
    account.buy_sell_line = {}
    pass

def handle_data(account):                  # 每个交易日的买入卖出指令

    #准备数据
    if not account.current_date.strftime('%Y%m%d') in account.ca
che:
        account.cache = {}
        account.cache[account.current_date.strftime('%Y%m%d')] =
 account.get_daily_history(1)
    if account.current_minute == "09:30":
        return
    #每天画一次线
    if account.current_minute == "09:31":
        account.buy_sell_line = {}
        for stock in account.cache[account.current_date.strftime(
'%Y%m%d')]:
            if stock in account.universe:
                close = account.cache[account.current_date.strft
ime('%Y%m%d')][stock]["closePrice"][0]
                low = account.cache[account.current_date.strftim
e('%Y%m%d')][stock]["lowPrice"][0]
```

```
                high = account.cache[account.current_date.strfti
me('%Y%m%d')][stock]["highPrice"][0]
                o = account.referencePrice[stock]
                r = max(high - low, close - low)
                account.buy_sell_line[stock] = {"buy": o + accou
nt.k1 * r, "sell": o - account.k2 * r}
    else:
        #每天剩余的时间根据画线买卖
        for stock in account.buy_sell_line:
            if stock in account.universe and stock in account.re
ferencePrice and stock in account.valid_secpos:
                if account.referencePrice[stock] < account.buy_s
ell_line[stock]["sell"]:
                    order_to(stock, 0)
                    account.holding -= 1
        for stock in account.buy_sell_line:
            if stock in account.universe and stock in account.re
ferencePrice and not stock in account.valid_secpos:
                if account.holding < account.holding_max and acc
ount.referencePrice[stock] > account.buy_sell_line[stock]["buy"]
:
                    account.holding += 1
                    order_pct(stock, 1.0/account.holding_max)
    return
```

回测看效果不是特别好...... LZ自己调一下参数吧

@JasonYichuan：

BollingerBandit很一般，不过没怎么调参数，看着办吧

```
import numpy as np
import pandas as pd

start = '2015-01-01'                       # 回测起始时间
end = '2015-11-26'                         # 回测结束时间
benchmark = 'HS300'                        # 策略参考标准
universe = set_universe('HS300')  # 证券池，支持股票和基金
capital_base = 100000                      # 起始资金
#commission = Commission(buycost=0.00025,sellcost=0.00025)  # 佣金

freq = 'd'                                 # 策略类型，'d'表示日间
策略使用日线回测，'m'表示日内策略使用分钟线回测
refresh_rate = 1                           # 调仓频率

# 全局参数
## Boll线参数
N = 20
k = 2
## ROC变动率参数
M = 20
```

```
## 平仓参数
E = 20

def initialize(account):                      # 初始化虚拟账户状态
    # 持股代码以及持股时间
    account.duration = pd.DataFrame(np.array([0]*len(universe)),
 index=universe, columns=['duration'])
    account.amount = 400

def handle_data(account):                     # 每个交易日的买入卖出指令

    hist = account.get_attribute_history('closePrice',50)
    ticker_name = []                          # 符合买入要求股票代码
    for stk in account.universe:              # 遍历股票池内所有股票，
选出符合要求的股票
        if np.isnan(account.referencePrice[stk]) or account.refe
rencePrice[stk] == 0:  # 停牌或是还没有上市等原因不能交易
            continue

        # 计算股票的BOLL线上下轨
        ## 计算MA
        MA = np.mean(hist[stk][-N:])
        ## 计算标准差MD
        MD = np.sqrt((sum(hist[stk][-N:] - MA)**2) / N)
        ## 计算MB、UP、DN线
        MB =np.mean(hist[stk][-(N-1):])
        UP = MB + k * MD
        DN = MB - k * MD

        # 计算股票的ROC
        ROC = float(hist[stk][-1] - hist[stk][-M])/float(hist[st
k][-M])

        # 开仓条件
        if (hist[stk][-1] > UP) and (ROC > 0):
            ticker_name.append(stk)
    # 若股票符合开仓条件且尚未持有，则买入
    for stk in ticker_name:
        if stk not in account.valid_secpos:
            order(stk,account.amount)
            account.duration.loc[stk]['duration'] = 1
    # 对于持有的股票，若股票不符合平仓条件，则将持仓时间加1，否则卖出，并删
除该持仓时间记录
    for stk in account.valid_secpos:
        T = max(E - account.duration.loc[stk]['duration'],10)
        if hist[stk][-1] > np.mean(hist[stk][-T:]):
            account.duration.loc[stk]['duration'] = account.dura
tion.loc[stk]['duration'] + 1
        else:
            order_to(stk,0)
            account.duration.loc[stk]['duration'] = 0
    return
```

5.17 卡尔曼滤波

> 来源：https://uqer.io/community/share/56324662f9f06c06acdb4762

有没有朋友懂如何用卡尔曼滤波进行金融数据分析的？

近来看了一些金融数据分析的资料。 其中有提到用卡尔曼滤波进行数据处理。

比如下面的文章:

http://jonathankinlay.com/?p=1185

由于是EE背景，对滤波很有感情，所以看到卡尔曼滤波的处理方法，感觉很是兴奋。

但是文章没太懂，想找懂这块的朋友相互交流。

如果有相关书籍能够推荐就太好了。

@llhe：

应该很好理解吧，配对交易很重要的是计算配对的比例，而比例是时变的，这就涉及到估计对冲系数的问题，卡尔曼滤波就是用来干这个的。我猜你简单的用滑动平均可能也可以的。刚开始学习，纸上谈兵多交流

5.18 LPPL anti-bubble model

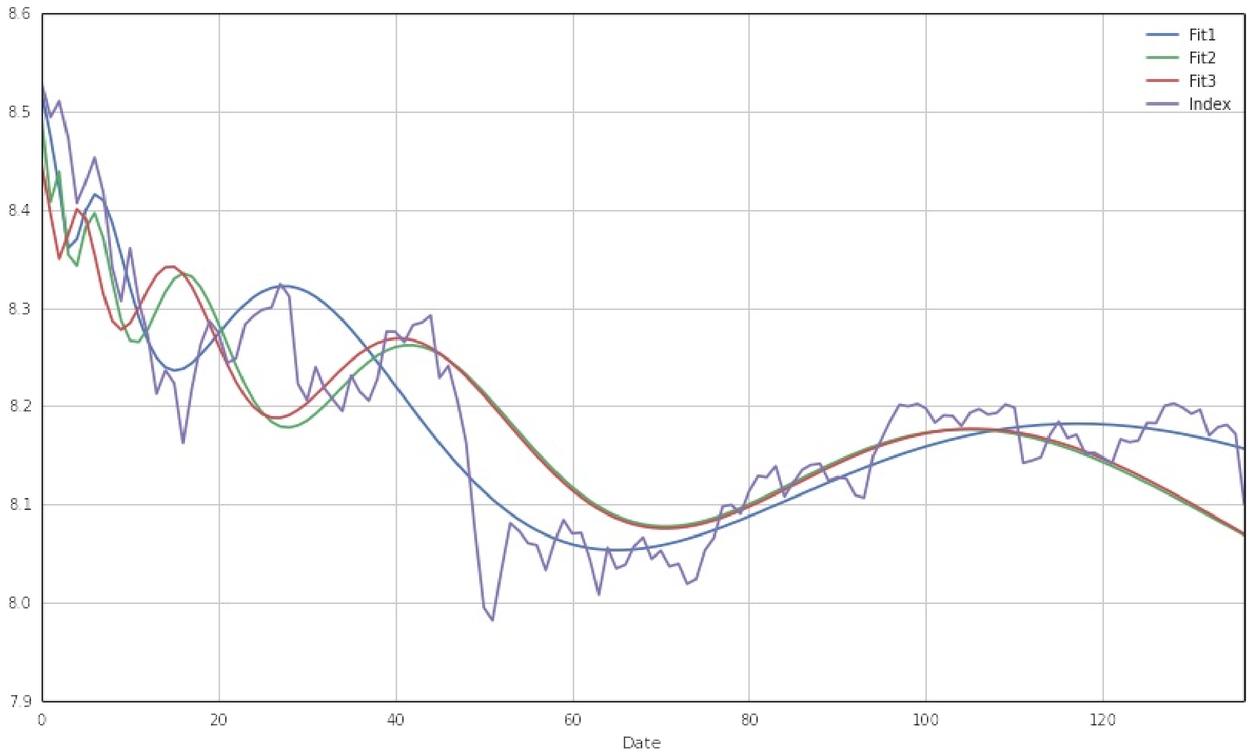
今天大盘熔断大跌，后市如何—— based on LPPL anti-bubble model

> 来源：https://uqer.io/community/share/568a65e0228e5b67159becfc

- 今天指数两次熔断，沪深300大跌7%，很恐怖，我节前空仓，今天也就索性全部放逆回购了，这样后天才能赎回，也是为了防止明天万一想抄底剁手
- 反正也是现成的算法调了调参数跑一跑，LPPL的anti-bubble model，感谢优矿平台
- 今天大跌我想说一下我的想法，大跌原因不在熔断机制，而在于股价太贵了,没有足够的流动性支撑，熔断机制只是催化剂罢了。没有熔断，指数照样要冲不出3600，就想没有国家救市，指数照样能在2500点以上反弹一样（当然后期平台不会这么久）。

```
import lib.relppltool as relppltool
from matplotlib import pyplot as plt
import datetime
import numpy as np
import pandas as pd
import seaborn as sns
sns.set_style('white')

limits = ([8.5, 8.6], [-0.25, -0.12], [-3, 3], [.15,.4], [0.05,0
.1], [4,8], [0, 2*np.pi])
x = relppltool.Population(limits, 20, 0.3, 1.5, .05, 4)
for i in range (3):
    x.Fitness()
    x.Eliminate()
    x.Mate()
    x.Mutate()

x.Fitness()
values = x.BestSolutions(3)
for xx in values:
    print xx.PrintIndividual()
```

```
data = pd.DataFrame({'Date':values[0].getDataSeries()[0],'Index'
:values[0].getDataSeries()[1],'Fit1':values[0].getExpData(),'Fit
2':values[1].getExpData(),'Fit3':values[2].getExpData()})
data = data.set_index('Date')
data.plot(figsize=(14,8))

<matplotlib.axes.AxesSubplot at 0x63dff50>
```



模型大家看看就好，权当参考。重点不在预测，而在资金管理。

破解股市泡沫之谜——对数周期幂率（LPPL）模型

> 来源：https://uqer.io/community/share/567a4fbd228e5b344568810f

引言

虽然离开物理专业有好几年了，但一直有些念念不忘，码着代码，写着开题报告，又闲不住想来讲一个没有好奇心的物理学家不是好金融学家的故事。

发现金融泡沫并预测到其何时破裂是很多从事金融行业的人的梦想。如今中国股市也成为了热门的话题，然而，资本狂欢之后是股灾，多少人因此从千万富翁炒股变成百万富翁，预测泡沫是所有人的梦想。

我们的主角Sornette教授登场了。Didier Sornette是一位受过培训的统计物理学家和地球物理学家，目前在瑞士联邦理工学院苏黎世分校(Swiss Federal Institute of Technology in Zurich)任金融学教授，主讲创业风险。他似乎并没有因外界对这种综合学科研究方法的热情有所减弱而感到烦恼。相反，他还在做自己大部分职业生涯一直在做的事：不仅在主要物理期刊上发表文章，还在领先的金融期刊上发表文章。

Sornette教授开始尝试着解答这个问题——不是通过传统的金融学方法，而是将物理学思想引入其中。作为2004年出版的《股市为什么会崩盘》(Why Stock Markets Crash)一书的作者，Sornette教授实质上是希望更深刻地理解泡沫的形成和发展。在对复杂体系的分析中，他独自——或者是和极少数几个人一起——引领着三个并行领域：纯物理学、应用经济学和计量经济学，以及市场从业人员。

在《股市为什么会崩盘》一书中，Sornette教授全面分析了一个由其提出的预测市场泡沫的模型——对数周期幂律（LPPL）模型。该模型对之后许多次市场泡沫都进行了准确的预测，由于该模型由Johansen，Ledoit和Sornette共同提出并完善，因此也被称为JLS模型。我们来聊一聊它。

什么是对数周期幂率模型？

作为纯物理学家的Sornettee教授不甘于仅仅在物理学领域有所建树，他还看到了金光闪闪的华尔街，在那里，各类炼金术师在寻找各种允许少数人持续获利的方法。于是，Sornettee教授在金融领域的跨界之旅开始了。他脑洞大开，想将物理学模型延伸到金融学领域中，而他找到的第一把金光闪闪的钥匙叫做易辛模型——一种描述物质铁磁性的经典模型。简单地说，易辛模型认为单个原子的磁矩只可能有两种状态，+1（自旋向上）或者-1（自旋向下），原子以某种规则排列着，并存在着交互作用，使得相邻之间的原子的自旋互相影响。

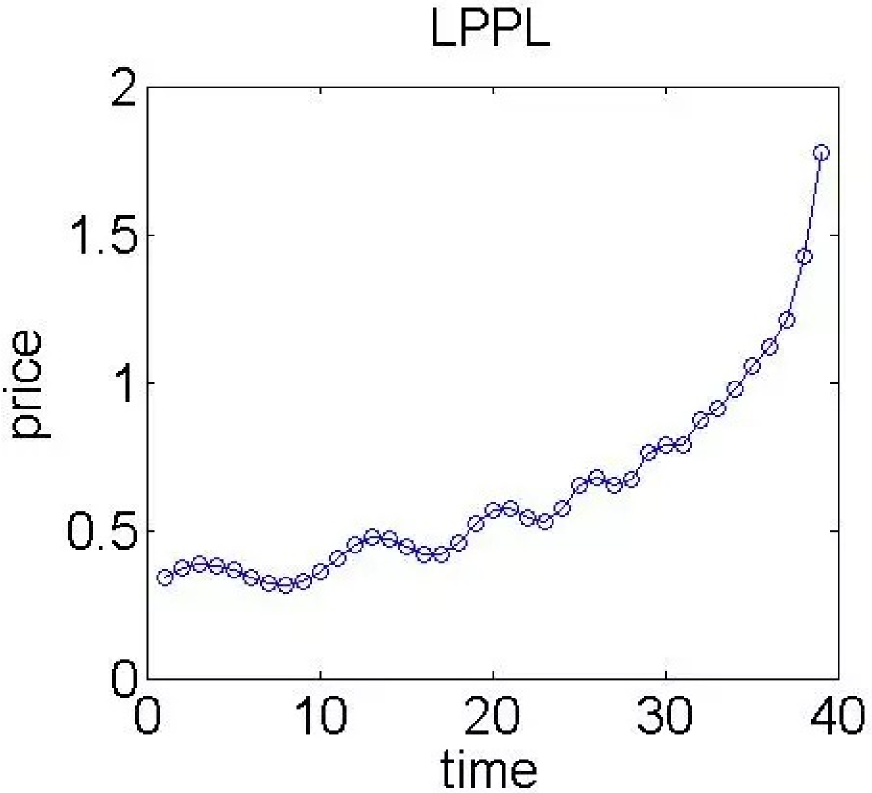
Sornette教授的眼睛仿佛一下子充满了光芒，他仿佛看到了美元纸币上的林肯在向他招手。受易辛模型启发，Sornette教授认为在金融市场中，投资者也只具有两种状态，即买或者卖。同时，投资者的交易行为取决于其他投资者的决策及外部因素的影响，这与易辛模型是多么的相似！

假想我们处于这样的一个市场中：资产没有派息、银行利率为零、市场极度厌恶风险，并且市场有着充足的流动性。显然，在这个市场中的金融资产没有任何价值，也就是其基础价值为零。在这样的框架内，市场中出现两类投资者，如上文所说，一类是理性投资者，一类是非理性的噪声投资者。后者具有羊群效应，使得金融资产价格偏离其基础价值，在没有足够的做空机制下，该结果导致理性投资者也不得不跟随噪声投资者的行为，通过享受泡沫来获得收益。最终当趋势达到某一临界值时，大量投资者没有足够的头寸维持该趋势，于是手中的卖单导致了市场的崩盘。

那么这是一个怎样的趋势呢？Sornette教授考虑了自激励的正反馈过程的思想，而该过程会导致大量交易者的行为方式逐渐趋于一致。在经过一些推导之后，Sornette教授发现该趋势是按对数周期幂律（LPPL）增长，这里给出唯一的也是最重要的公式。

$$\ln[p(t)] \approx A + B(t_c - t)^{\beta}\{1 + C\cos[\omega\ln(t_c - t) + \phi]\}$$

这里不去探讨该公式的具体意义，让我们看一下它的样子。



可以看到，随着时间增长，资产价格有着近似指数增长的特点，但同时也伴随着不断的振荡，随着时间越来越接近临界时间，振荡的幅度逐渐减小，增长速度逐渐增大，进入超指数的增长状态，最终市场在临界时间点附近崩盘。通过该模型，人们可以提前获知可能的临界时间点来规避风险。该模型曾成功预测了2008年的石油泡沫，美国房地产泡沫，以及2009 年中国股市泡沫等。

然而，试图根据泡沫迹象采取行动的交易员，现在或许会非常失望。正如Sornette教授自己承认的，其理论实际上旨在估计这种泡沫的存在时间，而过早退出市场将是个错误，很可能会损失大量资金，然而离开过晚就不是几个人失去工作的事了。

事实上，该模型并没有考虑交易者以外的因素，比如政策层面或者市场情绪等因素，但将金融系统认为是一个复杂系统并加以研究的思想是深远的。现在也有许多将语义情绪分析等类似机器学习的方法应用于模型中来对市场状态进行分析。

毫无疑问，更多地了解泡沫的形成和发展，价值是无法估量的。关注经济和金融以外的领域有助于拓展思路，但不要指望找到一个指导市场交易的万能公式。

LPPL模型收到的批评与收到的赞扬一样多，有不少人认为该模型没有操作价值。如果你想了解更多关于它的信息，可以仔细详读Everything You Always Wanted to Know about Log Periodic Power Laws for Bubble Modelling but Were Afraid to Ask。

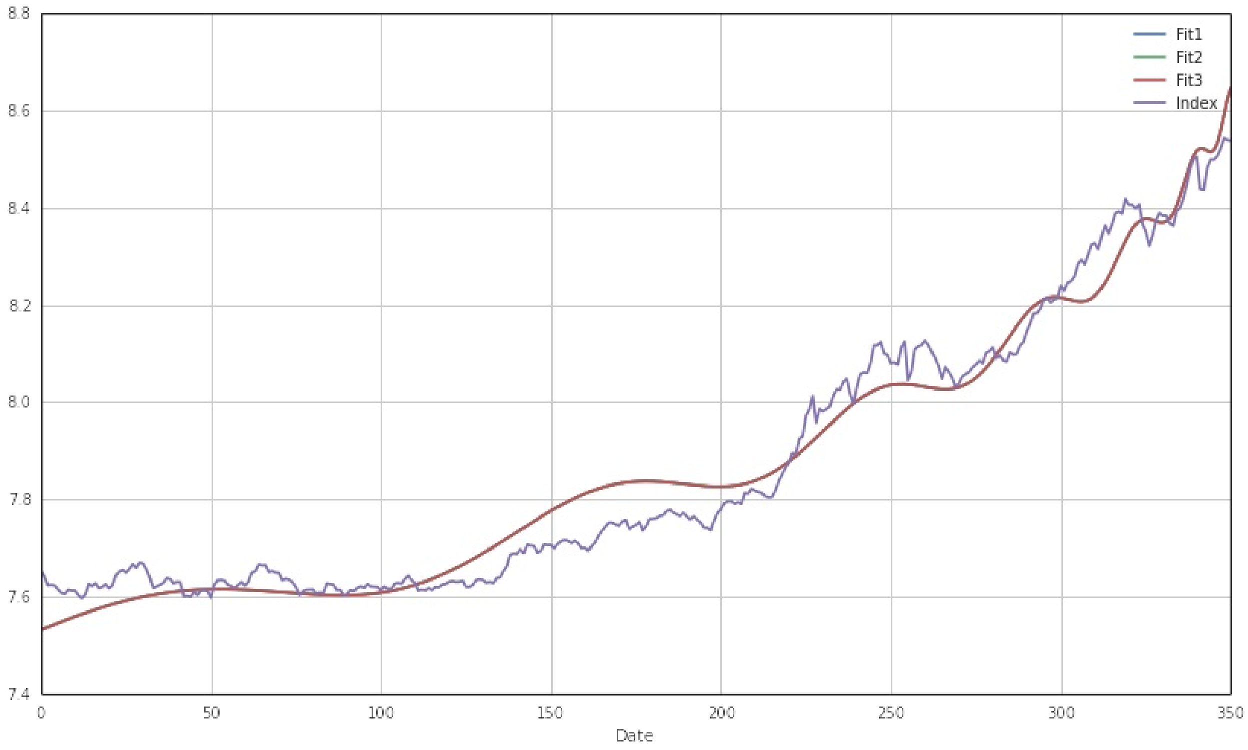
最后，如果你有好奇心，那么你一定想知道LPPL模型的实战结果到底如何。下面我用优矿再现了LPPL模型预测2015年夏天A股市场的泡沫。

P.S 这次股灾虽然我逃顶了，也许我马后炮了~ :)

使用 `DataAPI.MktIdxdGet()` 函数获取上证指数2014年1月1日至2015年6月10日的指数信息（股灾发生在约一星期后）。 `lib` 库我也放在了文章最后，大家可以尝试着使用。特别感谢jd8001，用于拟合的GA算法的核心代码框架来自于他，我对参数设置做了细致的调整，以让它更好的符合A股市场，最后我友好地加上了一些注释。

```
import lib.lppltool as lppltool
from matplotlib import pyplot as plt
import datetime
import numpy as np
import pandas as pd
import seaborn as sns
sns.set_style('white')

limits = ([8.4, 8.8], [-1, -0.1], [350, 400], [.1,.9], [-1,1], [
12,18], [0, 2*np.pi])
x = lppltool.Population(limits, 20, 0.3, 1.5, .05, 4)
for i in range (2):
    x.Fitness()
    x.Eliminate()
    x.Mate()
    x.Mutate()

x.Fitness()
values = x.BestSolutions(3)
for x in values:
    print x.PrintIndividual()

Fitness Evaluating: 0 of 20
Fitness Evaluating: 1 of 20
Fitness Evaluating: 2 of 20
Fitness Evaluating: 3 of 20
Fitness Evaluating: 4 of 20
Fitness Evaluating: 5 of 20
Fitness Evaluating: 6 of 20
Fitness Evaluating: 7 of 20
Fitness Evaluating: 8 of 20
Fitness Evaluating: 9 of 20
Fitness Evaluating: 10 of 20
Fitness Evaluating: 11 of 20
Fitness Evaluating: 12 of 20
Fitness Evaluating: 13 of 20
Fitness Evaluating: 14 of 20
Fitness Evaluating: 15 of 20
Fitness Evaluating: 16 of 20
Fitness Evaluating: 17 of 20
Fitness Evaluating: 18 of 20
Fitness Evaluating: 19 of 20
 fitness out size: 20 0
Eliminate: 14
Mate Loop complete: 25
Mutate: 2
Fitness Evaluating: 0 of 31
Fitness Evaluating: 1 of 31
Fitness Evaluating: 2 of 31
Fitness Evaluating: 3 of 31
Fitness Evaluating: 4 of 31
Fitness Evaluating: 5 of 31
Fitness Evaluating: 6 of 31
Fitness Evaluating: 7 of 31
Fitness Evaluating: 8 of 31
Fitness Evaluating: 9 of 31
Fitness Evaluating: 10 of 31
Fitness Evaluating: 11 of 31
Fitness Evaluating: 12 of 31
Fitness Evaluating: 13 of 31
Fitness Evaluating: 14 of 31
Fitness Evaluating: 15 of 31
Fitness Evaluating: 16 of 31
Fitness Evaluating: 17 of 31
Fitness Evaluating: 18 of 31
Fitness Evaluating: 19 of 31
Fitness Evaluating: 20 of 31
Fitness Evaluating: 21 of 31
Fitness Evaluating: 22 of 31
Fitness Evaluating: 23 of 31
Fitness Evaluating: 24 of 31
Fitness Evaluating: 25 of 31
Fitness Evaluating: 26 of 31
Fitness Evaluating: 27 of 31
Fitness Evaluating: 28 of 31
Fitness Evaluating: 29 of 31
Fitness Evaluating: 30 of 31
 fitness out size: 31 0
Eliminate: 25
Mate Loop complete: 25
Mutate: 0
Fitness Evaluating: 0 of 31
Fitness Evaluating: 1 of 31
Fitness Evaluating: 2 of 31
Fitness Evaluating: 3 of 31
Fitness Evaluating: 4 of 31
Fitness Evaluating: 5 of 31
Fitness Evaluating: 6 of 31
Fitness Evaluating: 7 of 31
Fitness Evaluating: 8 of 31
Fitness Evaluating: 9 of 31
Fitness Evaluating: 10 of 31
Fitness Evaluating: 11 of 31
Fitness Evaluating: 12 of 31
Fitness Evaluating: 13 of 31
Fitness Evaluating: 14 of 31
Fitness Evaluating: 15 of 31
Fitness Evaluating: 16 of 31
Fitness Evaluating: 17 of 31
Fitness Evaluating: 18 of 31
Fitness Evaluating: 19 of 31
Fitness Evaluating: 20 of 31
Fitness Evaluating: 21 of 31
Fitness Evaluating: 22 of 31
Fitness Evaluating: 23 of 31
Fitness Evaluating: 24 of 31
Fitness Evaluating: 25 of 31
Fitness Evaluating: 26 of 31
Fitness Evaluating: 27 of 31
Fitness Evaluating: 28 of 31
Fitness Evaluating: 29 of 31
Fitness Evaluating: 30 of 31
 fitness out size: 31 0
fitness: 0.99612688166
A: 9.817B: -0.681Critical Time: 365.323m: 0.207c: -0.023omega: 12.241phi: 4.25
fitness: 0.99612688166
A: 9.817B: -0.681Critical Time: 365.323m: 0.207c: -0.023omega: 12.241phi: 4.25
fitness: 0.99653502204
A: 9.8B: -0.667Critical Time: 365.405m: 0.209c: -0.023omega: 12.267phi: 4.105
```

```
data = pd.DataFrame({'Date':values[0].getDataSeries()[0],'Index':values[0].getDataSeries()[1],'Fit1':values[0].getExpData(),'Fit2':values[1].getExpData(),'Fit3':values[2].getExpData()})
data = data.set_index('Date')
data.plot(figsize=(14,8))

<matplotlib.axes.AxesSubplot at 0x663c250>
```



模型预测的临界时间（Critical Time）为365，即为6月10日（350）之后的15个交易日左右。实际股灾时间为6月15日（353），比实际结果晚10个交易日左右。

`lib` 库代码，请保存并命名为 `lppltool`

```python
#code created by jd8001
#reference: https://github.com/jd8001/LPPL
#kindly thank jd8001!

import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import fmin_tnc
import random
import pandas as pd
from pandas import Series, DataFrame
import datetime
import itertools

SP = DataAPI.MktIdxdGet(ticker='000001',beginDate='20140101',end
Date='20150610',field=["tradeDate","closeIndex"],pandas="1")
global date = SP.tradeDate
time = np.linspace(0, len(SP)-1, len(SP))
close = [np.log(SP.closeIndex[i]) for i in range(len(SP))]
global DataSeries
DataSeries = [time, close]

def lppl (t,x): #return fitting result using LPPL parameters
    a = x[0]
    b = x[1]
    tc = x[2]
    m = x[3]
    c = x[4]
    w = x[5]
    phi = x[6]
    return a + (b*np.power(tc - t, m))*(1 + (c*np.cos((w *np.log(tc-t))+phi)))

def func(x):
    delta = [lppl(t,x) for t in DataSeries[0]] #生成lppl时间序列
    delta = np.subtract(delta, DataSeries[1]) #将生成的lppl时间序列减去对数指数序列
    delta = np.power(delta, 2)
    return np.sum(delta) #返回拟合均方差


class Individual:
    'base class for individuals'


    def __init__ (self, InitValues):
        self.fit = 0
        self.cof = InitValues

    def fitness(self): #
        try:
            cofs, nfeval, rc = fmin_tnc(func, self.cof, fprime=None,approx_grad=True, messages=0) #基于牛顿梯度下山的寻找函数最小值
            self.fit = func(cofs)
            self.cof = cofs

        except:

            #does not converge
            return False



    def mate(self, partner): #交配
        reply = []
        for i in range(0, len(self.cof)): # 遍历所以的输入参数
            if (random.randint(0,1) == 1): # 交配，0.5的概率自身的参数保留，0.5的概率留下partner的参数，即基因交换
                reply.append(self.cof[i])
            else:
                reply.append(partner.cof[i])

        return Individual(reply)
    def mutate(self): #突变
        for i in range(0, len(self.cof)-1):
            if (random.randint(0,len(self.cof)) <= 2):
                #print "Mutate" + str(i)
                self.cof[i] += random.choice([-1,1]) * .05 * i #突变
    def PrintIndividual(self): #打印结果
        #t, a, b, tc, m, c, w, phi
        cofs = "A: " + str(round(self.cof[0], 3))
        cofs += "B: " + str(round(self.cof[1],3))
        cofs += "Critical Time: " + str(round(self.cof[2], 3))
        cofs += "m: " + str(round(self.cof[3], 3))
        cofs += "c: " + str(round(self.cof[4], 3))
        cofs += "omega: " + str(round(self.cof[5], 3))
        cofs += "phi: " + str(round(self.cof[6], 3))

        return "fitness: " + str(self.fit) +"\n" + cofs
        #return str(self.cof) + " fitness: " + str(self.fit)
    def getDataSeries(self):
        return DataSeries
    def getExpData(self):
        return [lppl(t,self.cof) for t in DataSeries[0]]
    def getTradeDate(self):
        return date


def fitFunc(t, a, b, tc, m, c, w, phi):
    return a - (b*np.power(tc - t, m))*(1 + (c*np.cos((w *np.log
(tc-t))+phi)))


class Population:
    'base class for a population'
    LOOP_MAX = 1000

    def __init__ (self, limits, size, eliminate, mate, probmutat
e, vsize):
        'seeds the population'
        'limits is a tuple holding the lower and upper limits of
 the cofs'
        'size is the size of the seed population'
        self.populous = []
        self.eliminate = eliminate
        self.size = size
        self.mate = mate
        self.probmutate = probmutate
        self.fitness = []

        for i in range(size):
            SeedCofs = [random.uniform(a[0], a[1]) for a in limi
ts]
            self.populous.append(Individual(SeedCofs))

    def PopulationPrint(self):
        for x in self.populous:
            print x.cof
    def SetFitness(self):
        self.fitness = [x.fit for x in self.populous]
    def FitnessStats(self):
        #returns an array with high, low, mean
        return [np.amax(self.fitness), np.amin(self.fitness), np
.mean(self.fitness)]
    def Fitness(self):
        counter = 0
        false = 0
        for individual in list(self.populous):
            print('Fitness Evaluating: ' + str(counter) +  " of "
 + str(len(self.populous)) + "        \r"),
            state = individual.fitness()
            counter += 1

            if ((state == False)):
                false += 1
                self.populous.remove(individual)
        self.SetFitness()
        print "\n fitness out size: " + str(len(self.populous))
+ " " + str(false)
    def Eliminate(self):
        a = len(self.populous)
        self.populous.sort(key=lambda ind: ind.fit)
        while (len(self.populous) > self.size * self.eliminate):
            self.populous.pop()
        print "Eliminate: " + str(a- len(self.populous))
    def Mate(self):
        counter = 0
        while (len(self.populous) <= self.mate * self.size):
            counter += 1
            i = self.populous[random.randint(0, len(self.populou
s)-1)]
            j = self.populous[random.randint(0, len(self.populou
s)-1)]
            diff = abs(i.fit-j.fit)
            if (diff < random.uniform(np.amin(self.fitness), np.
amax(self.fitness) - np.amin(self.fitness))):
                self.populous.append(i.mate(j))

            if (counter > Population.LOOP_MAX):
                print "loop broken: mate"
                while (len(self.populous) <= self.mate * self.si
ze):
                    i = self.populous[random.randint(0, len(self
.populous)-1)]
                    j = self.populous[random.randint(0, len(self
.populous)-1)]
                    self.populous.append(i.mate(j))


        print "Mate Loop complete: " + str(counter)

    def Mutate(self):
        counter = 0
        for ind in self.populous:
            if (random.uniform(0, 1) < self.probmutate):
                ind.mutate()
                ind.fitness()
                counter +=1
        print "Mutate: " + str(counter)
        self.SetFitness()

    def BestSolutions(self, num):
        reply = []
        self.populous.sort(key=lambda ind: ind.fit)
        for i in range(num):
            reply.append(self.populous[i])
        return reply;

    random.seed()
```

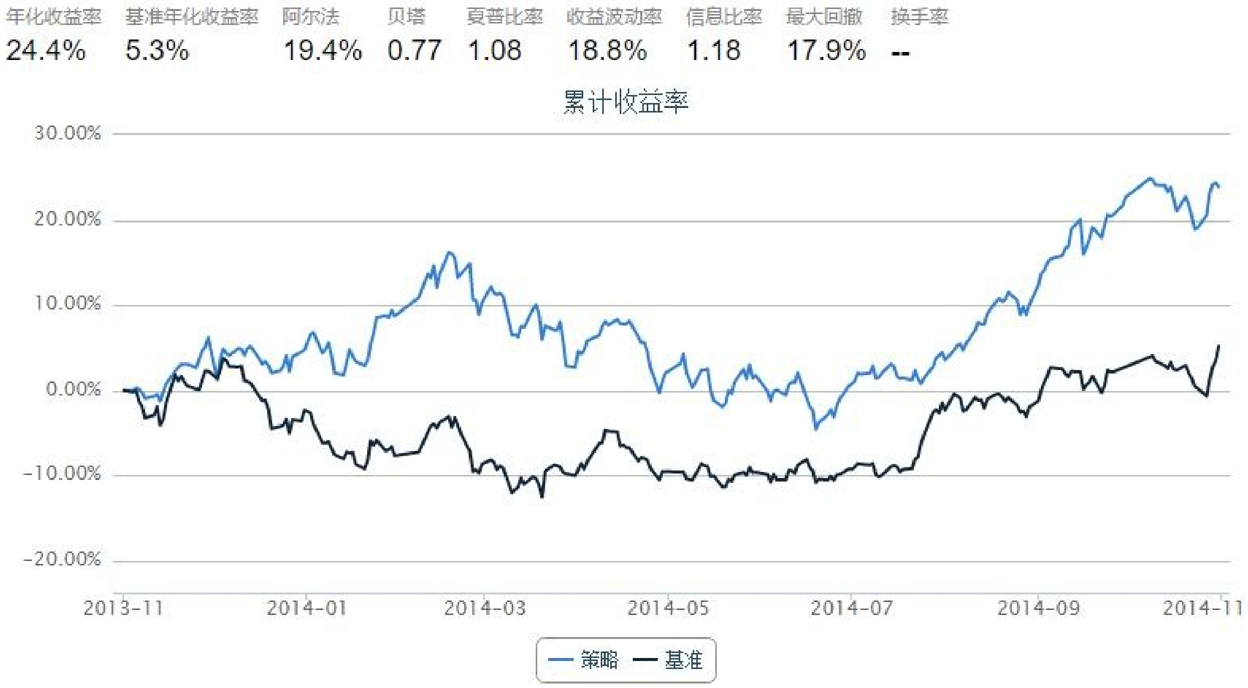
六 大数据模型

6.1 市场情绪分析

通联情绪指标策略

> 来源：https://uqer.io/community/share/548111daf9f06c8e7733670a

```
start = pd.datetime(2013, 11, 1)
end   = pd.datetime(2014, 11, 1)
benchmark = 'HS300'
universe = read('3b_ticker.txt').split(',')
capital_base = 100000
csvs = ['3b_news.csv']

def initialize(account):
    add_history('hist1', 1)

def handle_data(account, data):
    for stock in universe:
        if (stock not in account.hist1) or ('emotion' not in acc
ount.hist1[stock].columns):
            continue

        sig = account.hist1[stock].iloc[0,:]['emotion']
        if sig > 0.2 and account.position.stkpos.get(stock, 0)==0
:
            order(stock, 100)
        elif sig < 0 and account.position.stkpos.get(stock, 0)>0
:
            order_to(stock, 0)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
24.4%	5.3%	19.4%	0.77	1.08	18.8%	1.18	17.9%	--

互联网+量化投资 大数据指数手把手

> 来源：https://uqer.io/community/share/55263359f9f06c8f3390457b

策略简介

从公司基本面、市场驱动指标、市场情绪等多维度验证拥有“天时、地利、人和”的大牛股，让每个人都能生产符合自己投资理念的大数据指数。实现中参考了水星社区中的牛人@吴宇笛的因子计分卡策略。

本策略的参数如下：

- 起始日期：2014年1月1日
- 结束日期：2016年5月18日
- 股票池：上证50
- 业绩基准：上证50
- 起始资金：100000元
- 调仓周期：1个月

策略参数获取：

- 十日移动均线(MA10) 60日移动均线(MA60) 资产回报率(ROA) 市盈率(PE) 对数市值(LCAP) 波幅中位数(DHILO) 净利润/营业总收入(NPToTOR) 产权比率(DebtEquityRatio) 营业利润同比增长(OperatingProfitGrowRate) 总资产同比增长(TotalAssetGrowRate) 均可以通过 `DataAPI.MktStockFactorsDateRangeGet` 获得
- 市场新闻热度指标可以通过 `DataAPI.NewsHeatIndexGet` 获得
- 市场情绪指标可以通过 `DataAPI.NewsSentimentIndexGet` 获得；与新闻热度指标一样，都是DataYes利用大数据分析从海量关联新闻中提取出来的

调仓策略

(1) 对每只股票获取之前的120个交易日的收盘价，计算20日累计收益，共得到100个收益率数据

(2) 获取该股票同期的100个交易日的基本面、市场驱动指标和市场热度、情绪指标，分别计算均值、标准差，并进行中心化

(3) 以该股票20日累计收益率为因变量，基本面、市场驱动指标和市场热度、情绪指标为自变量进行弹性网（ElasticNet）回归

(4) 获取该股票前一日的基本面、市场驱动指标和市场热度、情绪指标

(5) 对该股票前一日的基本面、市场驱动指标和市场热度、情绪指标，依据前100个交易日的均值和标准差，置相对大小为 （前一日值 - 均值）/ 标准差 并四舍五入，作为在该项因子上的得分

(6) 根据之前计算出的权重对这些得分进行加总，得到该股票的得分，并以此为指数进行股票筛选

(7) 根据指数得分排序，选取总分最高的前五支股票作为买入列表

(8) 根据买入列表调仓

```
import pandas as pd
import numpy  as np
import statsmodels.api as sm
import statsmodels.regression.linear_model as lm
from sklearn.linear_model import ElasticNet
from CAL.PyCAL import *

used_factors = ['MA10', 'MA60', 'ROA', 'PE', 'LCAP', 'DHILO', 'DebtEquityRatio', 'OperatingProfitGrowRate', 'TotalAssetGrowRate', 'NPToTOR']

#used_factors = ['ASSI', 'EBITToTOR', 'ETP5', 'MA60', 'HSIGMA', 'PE', 'VOL60', 'SUE', 'DAVOL20', 'TotalAssetGrowRate']

def StockFactorsGet(universe, trading_days):
    data_all = {}
    for i,stock in enumerate(universe):
        try:
            data = DataAPI.MktStockFactorsDateRangeGet(secID = stock, beginDate = trading_days[0], endDate = trading_days[-1], field = ['tradeDate'] + used_factors)
            # data['tradeDate'] = pd.to_datetime(data['tradeDate'])
        except Exception, e:
            print e

        try:
            news_data = DataAPI.NewsHeatIndexGet(secID = stock, beginDate = trading_days[0], endDate = trading_days[-1])
            heatIndex = news_data.set_index('newsPublishDate').sort_index().reset_index()[['heatIndex','newsPublishDate']]
            heatIndex['flag'] = heatIndex['newsPublishDate'].apply(lambda x: True if x in data.tradeDate.values else False)
            heatIndex = heatIndex[heatIndex.flag].reset_index()
            data = pd.merge(data, heatIndex, how = 'inner', left_index = 'tradeDate', right_index = 'newsPublishDate').drop(['index','newsPublishDate','flag'], 1)
        except Exception, e:
            data['heatIndex'] = 0

        try:
            emotion_data = DataAPI.NewsSentimentIndexGet(secID = stock, beginDate = trading_days[0], endDate = trading_days[-1])
            emotionIndex = emotion_data.set_index('newsPublishDate').sort_index().reset_index()[['sentimentIndex','newsPublishDa
te']]
            emotionIndex['flag'] = emotionIndex['newsPublishDate'].apply(lambda x: True if x in data.tradeDate.values else False)
            emotionIndex = emotionIndex[emotionIndex.flag].reset_index()
            data = pd.merge(data, emotionIndex, how = 'inner', left_index = 'tradeDate', right_index = 'newsPublishDate').drop(['index','newsPublishDate','flag'], 1)
        except Exception, e:
            # print 'emotion', stock, e
            data['sentimentIndex'] = 0

        data['news_emotion'] = data['heatIndex'] * data['sentimentIndex']

        data_all[stock] = data
    return data_all

def StockRegDataGet(stock, trading_days, factors, shift = 20):
    start = trading_days[0]
    end   = trading_days[-1]
    data  = factors[(factors.tradeDate >= start.strftime('%Y-%m-%d')) & (factors.tradeDate <= end.strftime('%Y-%m-%d'))][:-shift]

    ret = DataAPI.MktEqudGet(secID = stock, beginDate = start.strftime('%Y%m%d'), endDate = end.strftime('%Y%m%d'), field = ['tradeDate', 'closePrice'])
    ret['fwdPrice'] = ret['closePrice'].shift(-shift)
    ret['return'] = ret['fwdPrice'] / ret['closePrice'] - 1.
    ret = ret[:-shift]

    data = data.merge(ret, how = 'inner', left_on = ['tradeDate'], right_on = ['tradeDate'])
    data = data.loc[:, ['return', 'heatIndex', 'sentimentIndex', 'news_emotion'] + used_factors]
    return data

def GetRegressionResult(data):
    data = data.dropna()

    all_factors = ['heatIndex', 'sentimentIndex', 'news_emotion'] + used_factors
    for f in all_factors:
        if data[f].std() == 0:
            continue
        data[f] = (data[f] - data[f].mean()) / data[f].std()

    y = np.array(data['return'].tolist())
    x = []
    for f in all_factors:
        x.append(data[f].tolist())
    x = np.column_stack(tuple(x))
    x = np.array( [ np.append(v,1) for v in x] )

    en = ElasticNet(fit_intercept=True, alpha=0)
    en.fit(x, y)
    res = en.coef_[:-1]
    w = dict(zip(all_factors, res))
    return w

def preparing(universe, date, factors_all):
    date = Date(date.year, date.month, date.day)

    cal = Calendar('China.SSE')
    start = cal.advanceDate(date, '-120B', BizDayConvention.Foll
owing)
    end   = cal.advanceDate(date, '-1B',   BizDayConvention.Foll
owing)

    start = datetime(start.year(), start.month(), start.dayOfMon
th())
    end   = datetime(  end.year(),   end.month(),   end.dayOfMon
th())

    trading_days = quartz.utils.tradingcalendar.get_trading_days
(start, end)
    datas, means, vols, weights = {}, {}, {}, {}
    for i,stock in enumerate(universe):
        try:
            datas[stock]   = StockRegDataGet(stock, trading_days
, factors_all[stock])
            means[stock]   = dict(datas[stock].mean())
            vols[stock]    = dict(datas[stock].std())
            weights[stock] = GetRegressionResult(datas[stock])
        except Exception, e:
            pass
    return means, vols, weights
```

```
from datetime import datetime
end   = datetime(2016, 5, 18)
f_start = datetime(2014, 1, 1)
universe = set_universe('SH50')
f_days = quartz.utils.tradingcalendar.get_trading_days(f_start,
end)
factors_all = StockFactorsGet(universe, f_days)
```

```
from datetime import datetime
start = datetime(2014, 6, 1)
end   = datetime(2016, 5, 18)
benchmark = 'SH50'
universe = set_universe('SH50')
```

```
capital_base = 100000
refresh_rate = 20

# f_start = datetime(2012, 6, 1)
# f_days = quartz.utils.tradingcalendar.get_trading_days(f_start
, end)
# factors_all = StockFactorsGet(universe, f_days)

def initialize(account):
    pass

def handle_data(account):
    print account.current_date
    means, vols, weights = preparing(account.universe, account.c
urrent_date, factors_all)

    cal  = Calendar('China.SSE')
    date = Date(account.current_date.year, account.current_date.
month, account.current_date.day)
    date = cal.advanceDate(date, '-1B', BizDayConvention.Followi
ng)
    date = datetime(date.year(), date.month(), date.dayOfMonth()
)

    factors_cur = StockFactorsGet(account.universe, [date])

    score = {}
    all_factors = ['heatIndex', 'sentimentIndex', 'news_emotion'
] + used_factors
    for stock in account.universe:
        if stock not in weights:
            continue

        fac = factors_cur[stock]
        s = 0
        for f in all_factors:
            try:
                x = fac[f].iloc[-1]
                x = (x - means[stock][f])/vols[stock][f]
                s += weights[stock][f] * int(round(x))
            except:
                pass
        score[stock] = s

    buylist = sorted(score.keys(), key = lambda x: score[x])[-5:
]
    rebalance(account, buylist)

def rebalance(account, buylist):
    for stock in account.valid_secpos:
        if stock not in buylist:
            order_to(stock, 0)
    for stock in buylist:
        order(stock, account.referencePortfolioValue / len(buyli
st) / account.referencePrice[stock])

2014-06-03 00:00:00
2014-07-01 00:00:00
2014-07-29 00:00:00
2014-08-26 00:00:00
2014-09-24 00:00:00
2014-10-29 00:00:00
2014-11-26 00:00:00
2014-12-24 00:00:00
2015-01-23 00:00:00
2015-02-27 00:00:00
2015-03-27 00:00:00
2015-04-27 00:00:00
2015-05-26 00:00:00
2015-06-24 00:00:00
2015-07-22 00:00:00
2015-08-19 00:00:00
2015-09-18 00:00:00
2015-10-23 00:00:00
```

6.2 新闻热点

如何使用优矿之“新闻热点”？

> 来源：https://uqer.io/community/share/55fa68a0f9f06cb1199d44c6

本期讲解如何使用优矿的新闻热点相关API，以及一个“然并卵”的示例策略。

包括：

- 股票新闻热点获取：`NewsHeatIndexGet`
- 股票新闻情感获取：`NewsSentimentIndexGet`
- 股票相关新闻获取：`NewsByTickersGet`

本篇中，我们只研究沪深300成分股。

```
from quartz.api import set_universe
universe = set_universe("HS300")
```

1. 获取新闻热点

使用：`NewsHeatIndexGet`

```
Type:        function
Definition: DataAPI.NewsHeatIndexGet(exchangeCD='', ticker='', s
ecShortName='', beginDate='', endDate='', secID='', field='', pa
ndas='1')
Docstring:
包含证券相关的新闻热度指数数据，输入一个或多个证券交易代码、起止日期，获取该
证券一段时间内的新闻热度指数(即证券当天关联新闻数量占当天新闻总量的百分比(%)
)。每天更新。（注：1、2014/1/1起新闻来源众多、指数统计有效，2013年及之前的
网站来源不全、数据波动大，数据自2004/10/28始；2、新闻量的统计口径为经算法处
理后证券关联到的所有常规新闻；3、数据按日更新。)
```

关键的参数：

- `secID:` 证券代码列表
- `beginDate`：新闻搜索开始日期
- `endDate`：新闻搜索结束日期

```
data = DataAPI.NewsHeatIndexGet(secID=universe, beginDate="20150
916", endDate="20150916")
data.sort('heatIndex', ascending=False).head()
```

	secID	exchangeCD	exchangeName	ticker	secSho
125	600030.XSHG	XSHG	上海证券交易所	600030	中信证
205	600837.XSHG	XSHG	上海证券交易所	600837	海通证
251	601688.XSHG	XSHG	上海证券交易所	601688	华泰证
241	601398.XSHG	XSHG	上海证券交易所	601398	工商银
269	601939.XSHG	XSHG	上海证券交易所	601939	建设银

获取的数据列表中，每一行就是对应的证券在某一天的新闻热度（heatIndex)。可以看到9月16日，中信证券（600030）荣登热度排行榜榜首！

2. 获取新闻情感

光知道新闻热度的话不够，我们还需要这道整体的新闻情感（正面or负面？）。

使用：`NewsSentimentIndexGet`

```
Type:       function
Definition: DataAPI.NewsSentimentIndexGet(exchangeCD='', ticker=
'', secShortName='', beginDate='', endDate='', secID='', field='
', pandas='1')
Docstring:
包含证券相关的新闻情感指数数据，输入一个或多个证券交易代码、起止日期，获取该
证券一段时间内的新闻情感指数(即当天证券关联新闻的情感均值)。（注：1、2014/1
/1起新闻来源众多、指数统计有效，2013年及之前的网站来源不全、数据波动大，数据
自2004/10/28始；2、新闻量的统计口径为经算法处理后证券关联到的所有常规新闻；
3、数据按日更新。)
```

关键的参数：

- `secID`：证券代码列表
- `beginDate`：新闻搜索开始日期
- `endDate`：新闻搜索结束日期

```
data = DataAPI.NewsSentimentIndexGet(secID=universe, beginDate="
20150916", endDate="20150916")
data.sort('sentimentIndex', ascending=True).head()
```

	secID	exchangeCD	exchangeName	ticker	secSh
49	000831.XSHE	XSHE	深圳证券交易所	000831	五矿稀
125	600030.XSHG	XSHG	上海证券交易所	600030	中信证
171	600489.XSHG	XSHG	上海证券交易所	600489	中金黄
231	601225.XSHG	XSHG	上海证券交易所	601225	陕西煤
95	002653.XSHE	XSHE	深圳证券交易所	002653	海思科

获取的数据列表中，每一行就是对应的证券在某一天的新闻情感（ `heatIndex` ），负数代表负面情感，正数代表正面情感。可以看到9月16日，中信证券（600030）在新闻情感指数榜上排名倒数第二！

3. 股票详细新闻获取

用户如果想更深度的剖析个别新闻对某只证券的影响，可以通过API获取详细的新闻分析列表：

使用： `NewsByTickersGet`

```
Type:        function
Definition: DataAPI.NewsByTickersGet(ticker='', secShortName='',
 secID='', exchangeCD='', beginDate='', endDate='', field='', pa
ndas='1')
Docstring:
包含证券相关的新闻数据，同时可获取针对不同证券的新闻情感数据。输入证券代码或
简称、查询的新闻发布起止时间，同时可输入证券交易所代码，获取相关新闻数据，如
：新闻ID、新闻标题、发布来源、发布时间、入库时间等。(注：1、自2014/1/1起新
闻来源众多、新闻量日均4万左右，2013年及之前的网站来源少、新闻数据量少；2、数
据实时更新。)
```

关键的参数：

- `secID` ：证券代码列表
- `beginDate` ：新闻搜索开始日期
- `endDate` ：新闻搜索结束日期

我们来试着获取2015年9月16日当天中信证券的相关新闻：

```
data = DataAPI.NewsByTickersGet(secID='600030.XSHG', beginDate='
20150916', endDate='20150916')
data.sort('relatedScore', ascending=False).head(10)
```

	secID	exchangeCD	exchangeName	ticker	secSho
71	600030.XSHG	XSHG	上海证券交易所	600030	中信证
106	600030.XSHG	XSHG	上海证券交易所	600030	中信证
118	600030.XSHG	XSHG	上海证券交易所	600030	中信证
31	600030.XSHG	XSHG	上海证券交易所	600030	中信证
19	600030.XSHG	XSHG	上海证券交易所	600030	中信证

33	600030.XSHG	XSHG	上海证券交易所	600030	中信证
40	600030.XSHG	XSHG	上海证券交易所	600030	中信证
61	600030.XSHG	XSHG	上海证券交易所	600030	中信证
129	600030.XSHG	XSHG	上海证券交易所	600030	中信证
53	600030.XSHG	XSHG	上海证券交易所	600030	中信证

4. 使用新闻数据编写简单策略

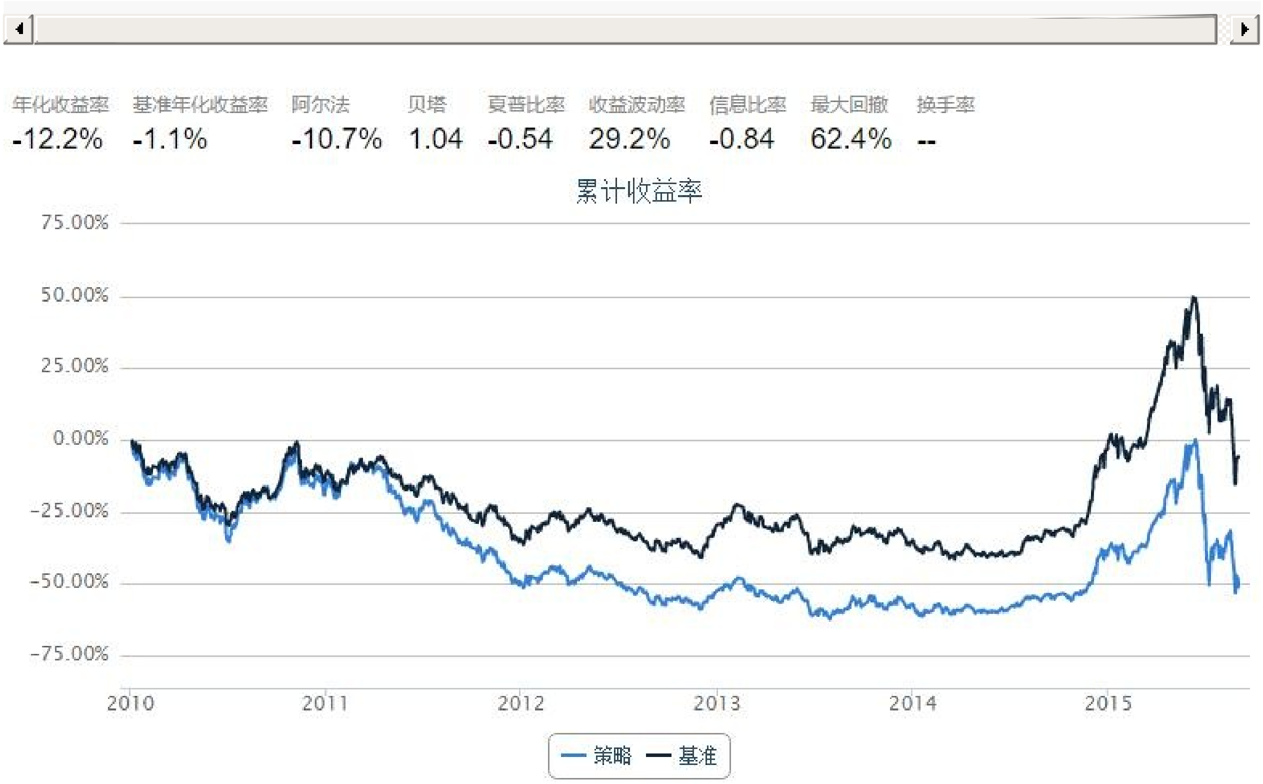
策略的指导想法是买入市场关为热点，并且新闻情感为正面的股票。

策略参数：

- 开始日期：2010年1月1日
- 结束日期：2015年9月1日
- 选择域：沪深成分股（2010年1月1日采样）
- 调仓周期：10个交易日
- 买入方法：等权重买入
- 规则：选取热度最高的100支股票，从中再选取情感最高并且为正的20支。

```
from CAL.PyCAL import *
```

```
start = '2010-01-01'
end = '2015-09-01'
benchmark = 'HS300'
universe = set_universe('HS300', start)
capital_base = 1000000
freq = 'd'
refresh_rate = 10

def initialize(account):
    pass

def handle_data(account):
    cal = Calendar('China.SSE')
    endDate = cal.advanceDate(account.current_date, '-1b', BizDayConvention.Preceding)
    beginDate = cal.advanceDate(endDate, '-10b', BizDayConvention.Preceding)

    # 获取当前参考期内股票热度
    data = DataAPI.NewsHeatIndexGet(secID=account.universe, beginDate=beginDate.strftime("%Y%m%d"), endDate=endDate.strftime("%Y%m%d"))

    # 只选取热度排名前100的股票
    sortedHeatIndex = data.groupby('secID')[['secID', 'heatIndex']].mean()
    choosenStocks = list(sortedHeatIndex.sort('heatIndex', ascending=False).index[:100].values)

    # 获取选取的50支股票的情感指数
    data = DataAPI.NewsSentimentIndexGet(secID=choosenStocks, beginDate=beginDate.strftime("%Y%m%d"), endDate=endDate.strftime("%Y%m%d"))

    # 只选取正面情感最高的20支股票
    data = data.groupby('secID')[['secID', 'sentimentIndex']].mean()
    sortedSentimentIndex = data.sort('sentimentIndex', ascending=False)
    sortedSentimentIndex = sortedSentimentIndex[sortedSentimentIndex['sentimentIndex'] > 0]
    choosenStocks = list(sortedSentimentIndex.index[:20].values)

    estimtedPortfolioValue = account.referencePortfolioValue

    # 卖出当前持仓
    for s in account.valid_secpos:
        order_to(s, 0)

    # 等比例买入选择股票
    for s in choosenStocks:
        order(s, int(estimtedPortfolioValue / len(choosenStocks) / account.referencePrice[s] / 100.)*100)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
-12.2%	-1.1%	-10.7%	1.04	-0.54	29.2%	-0.84	62.4%	--

看来这么简单的想法确实是“然并卵”！

技术分析【3】—— 众星拱月，众口铄金?

> 来源：https://uqer.io/community/share/55498c0af9f06c1c3d68806e

很多股民都习惯于看新闻，根据新闻中对某只股票的评价（或好或坏），进行买卖操作。这里新闻对于股票的评价我们称之为新闻情感。本篇中，我们将做一个小实验，看看这样的操作手法是否合理。

1. 数据准备

在我们的量化实验室中，用户可以通过数据API: `NewsSentimentIndexGet` 获取某只股票对应的新闻情感。

```
res = DataAPI.NewsSentimentIndexGet(secID = '600000.XSHG',field=[
'secID', 'newsPublishDate', 'sentimentIndex'])
res.tail()
```

	secID	newsPublishDate	sentimentIndex
26	600000.XSHG	2015-05-02	-0.008371
27	600000.XSHG	2015-05-03	-0.016820
28	600000.XSHG	2015-05-04	-0.013082
29	600000.XSHG	2015-05-05	0.004557
30	600000.XSHG	2015-05-06	-0.026943

上面的API调用，获得了最近的浦发银行的每日新闻情感：

- `secID` 证券代码
- `newsPublishDate` 交易日
- `sentimentIndex` 当时交易日的总体新闻情感指标，正的表示评价总体正面，负值表示评价总体负面

默认情况下，会获取最近30天的情感指标。

2. 操作手法

我们这里使用程序化的方法，执行如下的操作手法：

- 获取上证50成分股的最近30日新闻情感
- 将新闻情感按照 `secID` 分组，每组取平均；即获取每只股票最近30日情感

均值
- 取情感最正面的5只股票

```
from quartz.api import set_universe
universe = set_universe('SH50')

res = DataAPI.NewsSentimentIndexGet(secID=universe, field=['secI
D', 'newsPublishDate', 'sentimentIndex'])
res = res.groupby('secID')
res.mean().sort('sentimentIndex', ascending=False).head(5)
```

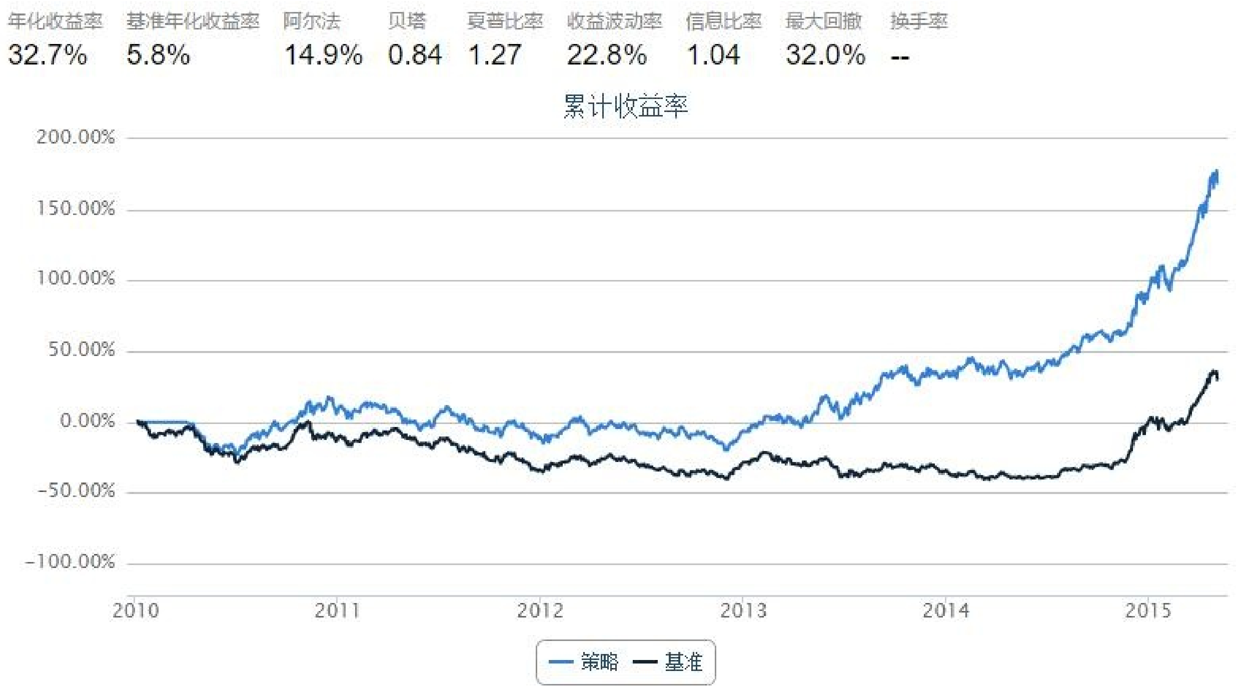
	sentimentIndex
secID	
600406.XSHG	0.153961
600372.XSHG	0.130670
600018.XSHG	0.119349
600887.XSHG	0.116333
600196.XSHG	0.108185

3. 策略实现

- 投资域 ：沪深300成分股
- 业绩基准 ：沪深300指数
- 调仓频率 ：60个交易日
- 开仓信号 ：评价最正面的10%股票
- 清仓信号 ：每个调仓日前一个工作日，清空当前仓位
- 买入方式 ：等比例买入
- 回测周期 ：2010年1月1日至2015年4月28日

```
from CAL.PyCAL import Date

start = '2010-01-01'                       # 回测起始时间
end = '2015-05-05'                         # 回测结束时间
benchmark = 'HS300'                        # 策略参考标准
universe = set_universe('HS300')               # 证券池，支持股票和
基金
capital_base = 1000000                      # 起始资金
longest_history = 0                        # handle_data 函数中可
以使用的历史数据最长窗口长度
refresh_rate = 1                          # 调仓频率，即每 refresh
_rate 个交易日执行一次 handle_data() 函数
longest_history = 1
```

```
def initialize(account):                     # 初始化虚拟账户状态
    account.isBuyPeriod = False
    account.dayCount = 0

def handle_data(account):                    # 每个交易日的买入卖出指令

    account.dayCount += 1
    if account.isBuyPeriod:                  # 每60个工作日（3个月）调仓
        hist = account.get_history(longest_history)
        endDate = Date.fromDateTime(account.current_date)
        startDate = endDate - 30
        res =  DataAPI.NewsSentimentIndexGet(secID=account.universe, field=['secID', 'newsPublishDate', 'sentimentIndex'], beginDate=startDate.strftime('%Y%m%d'),endDate=endDate.strftime('%Y%m%d'))
        res = res.groupby('secID')

        # top 10%
        top10 = res.mean().sort('sentimentIndex', ascending=False).head(int(0.1*len(res)))
        buyList = list(top10.index)
        print u"%s 买入 : %s" % (endDate, buyList)

        # 等权重买入
        if len(buyList) != 0:
            singleCash = account.cash / len(buyList)
            for stock in buyList:
                approximationAmount = int(singleCash / hist[stock]['closePrice'][-1]/100.0) * 100
                order(stock, approximationAmount)

        account.isBuyPeriod = False
        account.dayCount = 0
    elif account.dayCount == 59:            # 调仓日前一日清空当前仓位
        for stock in account.valid_secpos:
            order_to(stock,0)
        account.isBuyPeriod = True
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
32.7%	5.8%	14.9%	0.84	1.27	22.8%	1.04	32.0%	--

```
2010-04-06 买入 : ['601888.XSHG', '000338.XSHE', '000401.XSHE', '
000425.XSHE', '600880.XSHG', '601179.XSHG', '600066.XSHG', '6012
99.XSHG', '000983.XSHE', '601186.XSHG', '600010.XSHG', '601866.X
SHG', '002146.XSHE', '000630.XSHE']
2010-07-05 买入 : ['601006.XSHG', '600660.XSHG', '600188.XSHG', '
600498.XSHG', '600585.XSHG', '601168.XSHG', '000878.XSHE', '6017
66.XSHG', '600060.XSHG', '000630.XSHE', '000625.XSHE']
2010-09-30 买入 : ['600718.XSHG', '600703.XSHG', '000400.XSHE', '
600583.XSHG', '002230.XSHE', '601766.XSHG', '601808.XSHG', '6004
06.XSHG', '600795.XSHG', '600875.XSHG', '600066.XSHG', '600196.X
SHG', '000559.XSHE', '601018.XSHG']
2010-12-30 买入 : ['000883.XSHE', '600642.XSHG', '600998.XSHG', '
600276.XSHG', '002008.XSHE', '600066.XSHG', '600880.XSHG', '0022
92.XSHE', '600100.XSHG', '600741.XSHG', '601766.XSHG', '300070.X
SHE', '002146.XSHE', '600018.XSHG', '601299.XSHG', '000400.XSHE'
]
2011-04-01 买入 : ['600583.XSHG', '601179.XSHG', '000402.XSHE', '
600058.XSHG', '600900.XSHG', '601299.XSHG', '000623.XSHE', '6017
66.XSHG', '600118.XSHG', '601117.XSHG', '601006.XSHG', '600038.X
SHG', '600893.XSHG', '000559.XSHE', '000937.XSHE', '000876.XSHE'
]
2011-06-30 买入 : ['600252.XSHG', '600276.XSHG', '600066.XSHG', '
601117.XSHG', '600079.XSHG', '600583.XSHG', '601808.XSHG', '0009
63.XSHE', '600535.XSHG', '600741.XSHG', '600068.XSHG', '600196.X
SHG', '600688.XSHG', '600585.XSHG', '002353.XSHE', '601299.XSHG'
, '601933.XSHG']
2011-09-23 买入 : ['600079.XSHG', '600660.XSHG', '600276.XSHG', '
000581.XSHE', '002375.XSHE', '002465.XSHE', '600066.XSHG', '0020
81.XSHE', '600170.XSHG', '600009.XSHG', '600267.XSHG', '600588.X
SHG', '600893.XSHG', '600648.XSHG', '002400.XSHE', '600655.XSHG'
, '000869.XSHE', '000999.XSHE', '600741.XSHG', '600637.XSHG', '3
00017.XSHE', '000778.XSHE', '600196.XSHG', '601888.XSHG']
```

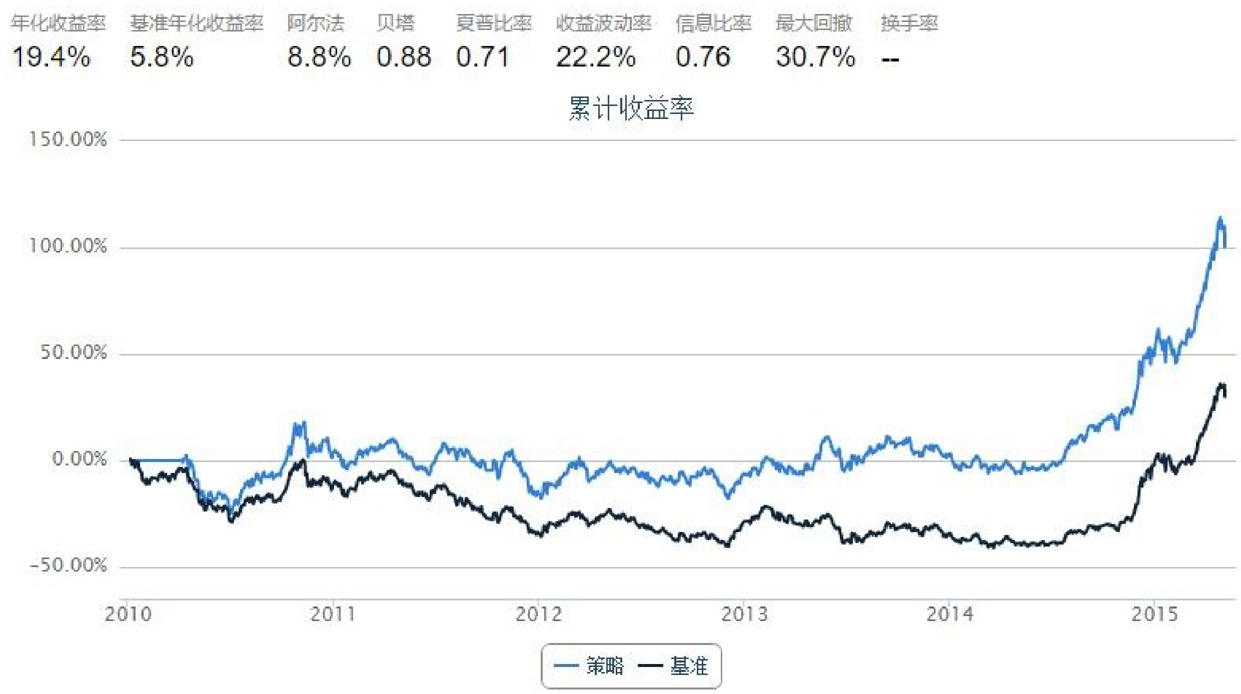
```
2011-12-23 买入 : ['600079.XSHG', '600863.XSHG', '000963.XSHE', '
000581.XSHE', '600741.XSHG', '000400.XSHE', '600660.XSHG', '0024
75.XSHE', '600060.XSHG', '600271.XSHG', '002415.XSHE', '002081.X
SHE', '600256.XSHG', '600009.XSHG', '002465.XSHE', '600166.XSHG'
, '000338.XSHE', '600068.XSHG', '600674.XSHG', '000630.XSHE', '6
00066.XSHG', '002422.XSHE', '000999.XSHE', '600340.XSHG']
2012-03-27 买入 : ['000338.XSHE', '002450.XSHE', '600893.XSHG', '
601098.XSHG', '600741.XSHG', '601179.XSHG', '300015.XSHE', '0023
53.XSHE', '601299.XSHG', '600060.XSHG', '600348.XSHG', '002375.X
SHE', '600066.XSHG', '600863.XSHG', '002470.XSHE', '600588.XSHG'
, '600655.XSHG', '000826.XSHE', '002065.XSHE', '600570.XSHG', '0
02230.XSHE', '300133.XSHE', '600880.XSHG', '000400.XSHE', '60015
7.XSHG']
2012-06-27 买入 : ['600660.XSHG', '600316.XSHG', '601333.XSHG', '
002353.XSHE', '002400.XSHE', '600741.XSHG', '600372.XSHG', '3002
51.XSHE', '002470.XSHE', '600089.XSHG', '002038.XSHE', '002310.X
SHE', '002603.XSHE', '601216.XSHG', '601669.XSHG', '601117.XSHG'
, '601766.XSHG', '601299.XSHG', '002252.XSHE', '000883.XSHE', '0
00027.XSHE', '600893.XSHG', '000963.XSHE', '600038.XSHG', '30013
3.XSHE']
2012-09-19 买入 : ['002465.XSHE', '002470.XSHE', '600660.XSHG', '
600718.XSHG', '600583.XSHG', '600079.XSHG', '600633.XSHG', '0009
63.XSHE', '000338.XSHE', '000826.XSHE', '600570.XSHG', '600372.X
SHG', '002146.XSHE', '600436.XSHG', '600867.XSHG', '600832.XSHG'
, '600498.XSHG', '601231.XSHG', '000400.XSHE', '300017.XSHE', '6
00271.XSHG', '002051.XSHE', '002450.XSHE', '600588.XSHG', '60115
8.XSHG', '002129.XSHE', '000792.XSHE']
2012-12-19 买入 : ['600170.XSHG', '300124.XSHE', '002475.XSHE', '
600741.XSHG', '002292.XSHE', '600718.XSHG', '601766.XSHG', '0024
65.XSHE', '002241.XSHE', '002081.XSHE', '002400.XSHE', '600166.X
SHG', '000826.XSHE', '600633.XSHG', '600373.XSHG', '601231.XSHG'
, '000725.XSHE', '000338.XSHE', '601299.XSHG', '600649.XSHG', '6
00535.XSHG', '601118.XSHG', '600547.XSHG', '600340.XSHG', '60063
7.XSHG', '002065.XSHE', '300133.XSHE']
2013-03-25 买入 : ['600880.XSHG', '601158.XSHG', '000963.XSHE', '
600867.XSHG', '600316.XSHG', '600718.XSHG', '002294.XSHE', '6002
71.XSHG', '600372.XSHG', '601928.XSHG', '600340.XSHG', '002410.X
SHE', '002292.XSHE', '601098.XSHG', '002465.XSHE', '002385.XSHE'
, '000598.XSHE', '600498.XSHG', '002146.XSHE', '002603.XSHE', '6
00373.XSHG', '600886.XSHG', '600633.XSHG', '600118.XSHG', '00091
7.XSHE', '600535.XSHG', '600038.XSHG']
2013-06-27 买入 : ['600741.XSHG', '600660.XSHG', '600900.XSHG', '
600008.XSHG', '600018.XSHG', '002400.XSHE', '600066.XSHG', '6013
33.XSHG', '002292.XSHE', '000826.XSHE', '002465.XSHE', '600703.X
SHG', '601299.XSHG', '000401.XSHE', '600583.XSHG', '600276.XSHG'
, '000917.XSHE', '600079.XSHG', '601098.XSHG', '002475.XSHE', '3
00124.XSHE', '600633.XSHG', '300015.XSHE', '600827.XSHG', '60180
0.XSHG', '600373.XSHG', '600637.XSHG']
2013-09-23 买入 : ['300015.XSHE', '000963.XSHE', '600660.XSHG', '
601098.XSHG', '600741.XSHG', '600066.XSHG', '600886.XSHG', '6007
03.XSHG', '600373.XSHG', '002465.XSHE', '601231.XSHG', '601299.X
SHG', '300133.XSHE', '601158.XSHG', '000623.XSHE', '600170.XSHG'
, '600009.XSHG', '002008.XSHE', '600827.XSHG', '600863.XSHG', '6
01928.XSHG', '600516.XSHG', '002051.XSHE', '000778.XSHE', '60040
6.XSHG', '300146.XSHE', '000826.XSHE', '300124.XSHE']
2013-12-23 买入 : ['600741.XSHG', '002400.XSHE', '300015.XSHE', '
002292.XSHE', '601929.XSHG', '600660.XSHG', '600900.XSHG', '0001
56.XSHE', '601299.XSHG', '600886.XSHG', '000963.XSHE', '600066.X
SHG', '600633.XSHG', '600018.XSHG', '600578.XSHG', '600498.XSHG'
, '000338.XSHE', '601098.XSHG', '600372.XSHG', '600583.XSHG', '0
02450.XSHE', '600703.XSHG', '600170.XSHG', '601179.XSHG', '60040
6.XSHG', '002465.XSHE', '600079.XSHG', '601766.XSHG']
2014-03-25 买入 : ['600900.XSHG', '601299.XSHG', '002292.XSHE', '
600886.XSHG', '300133.XSHE', '002400.XSHE', '600741.XSHG', '0024
65.XSHE', '600497.XSHG', '000963.XSHE', '600018.XSHG', '300015.X
SHE', '600633.XSHG', '601231.XSHG', '000338.XSHE', '600066.XSHG'
, '600585.XSHG', '600583.XSHG', '601928.XSHG', '600578.XSHG', '6
00703.XSHG', '600170.XSHG', '600498.XSHG', '002146.XSHE', '00224
1.XSHE', '601607.XSHG', '600372.XSHG', '601929.XSHG']
2014-06-23 买入 : ['600578.XSHG', '600741.XSHG', '600170.XSHG', '
600660.XSHG', '601158.XSHG', '600886.XSHG', '002292.XSHE', '0021
46.XSHE', '600066.XSHG', '600703.XSHG', '600485.XSHG', '300015.X
SHE', '002465.XSHE', '000598.XSHE', '600718.XSHG', '601231.XSHG'
, '601098.XSHG', '002400.XSHE', '600018.XSHG', '601179.XSHG', '6
00079.XSHG', '600498.XSHG', '600783.XSHG', '600089.XSHG', '00082
6.XSHE', '600497.XSHG', '600583.XSHG', '000400.XSHE']
2014-09-16 买入 : ['600741.XSHG', '300015.XSHE', '600578.XSHG', '
002400.XSHE', '600886.XSHG', '600079.XSHG', '002465.XSHE', '6001
70.XSHG', '002292.XSHE', '600660.XSHG', '601158.XSHG', '600703.X
SHG', '601299.XSHG', '600633.XSHG', '600718.XSHG', '000963.XSHE'
, '600066.XSHG', '601179.XSHG', '600900.XSHG', '601929.XSHG', '0
00598.XSHE', '601231.XSHG', '600585.XSHG', '002146.XSHE', '60008
9.XSHG', '002241.XSHE', '600583.XSHG', '601098.XSHG']
2014-12-16 买入 : ['600741.XSHG', '002400.XSHE', '601929.XSHG', '
600578.XSHG', '600079.XSHG', '002292.XSHE', '600170.XSHG', '3000
15.XSHE', '601158.XSHG', '600718.XSHG', '600783.XSHG', '601098.X
SHG', '600900.XSHG', '600583.XSHG', '600703.XSHG', '000963.XSHE'
, '601179.XSHG', '600018.XSHG', '600660.XSHG', '601800.XSHG', '6
00585.XSHG', '600886.XSHG', '600066.XSHG', '002146.XSHE', '00033
8.XSHE', '600497.XSHG', '002465.XSHE', '601607.XSHG']
2015-03-19 买入 : ['601098.XSHG', '600718.XSHG', '000400.XSHE', '
600066.XSHG', '600900.XSHG', '600018.XSHG', '600886.XSHG', '0009
63.XSHE', '600089.XSHG', '600583.XSHG', '600373.XSHG', '002051.X
SHE', '002292.XSHE', '002400.XSHE', '002465.XSHE', '600703.XSHG'
, '600316.XSHG', '600153.XSHG', '600118.XSHG', '300124.XSHE', '0
02450.XSHE', '600485.XSHG', '601158.XSHG', '600893.XSHG', '60192
9.XSHG', '600498.XSHG', '600276.XSHG', '000598.XSHE']
```

我们还可以试一下反向操作，选取评价最负面的10%

- 投资域 ：沪深300成分股
- 业绩基准 ：沪深300指数
- 调仓频率 ：60个交易日
- 开仓信号 ：评价最负面的10%股票
- 清仓信号 ：每个调仓日前一个工作日，清空当前仓位

- 买入方式 ：等比例买入
- 回测周期 ：2010年1月1日至2015年4月28日

```
from CAL.PyCAL import Date

start = '2010-01-01'                       # 回测起始时间
end = '2015-05-05'                         # 回测结束时间
benchmark = 'HS300'                        # 策略参考标准
universe = set_universe('HS300')               # 证券池，支持股票和
基金
capital_base = 1000000                      # 起始资金
longest_history = 0                        # handle_data 函数中可
以使用的历史数据最长窗口长度
refresh_rate = 1                          # 调仓频率，即每 refresh
_rate 个交易日执行一次 handle_data() 函数
longest_history = 1


def initialize(account):                   # 初始化虚拟账户状态
    account.isBuyPeriod = False
    account.dayCount = 0

def handle_data(account):                  # 每个交易日的买入卖出指令

    account.dayCount += 1
    if account.isBuyPeriod:                # 每60个工作日（3个月）
调仓
        hist = account.get_history(longest_history)
        endDate = Date.fromDateTime(account.current_date)
        startDate = endDate - 30
        res =  DataAPI.NewsSentimentIndexGet(secID=account.unive
rse, field=['secID', 'newsPublishDate', 'sentimentIndex'], begin
Date=startDate.strftime('%Y%m%d'),endDate=endDate.strftime('%Y%m
%d'))
        res = res.groupby('secID')

        # Bottom 10%
        top10 = res.mean().sort('sentimentIndex', ascending=True
).head(int(0.1*len(res)))
        buyList = list(top10.index)
        print u"%s 买入 : %s" % (endDate, buyList)

        # 等权重买入
        if len(buyList) != 0:
            singleCash = account.cash / len(buyList)
            for stock in buyList:
                approximationAmount = int(singleCash / hist[stoc
k]['closePrice'][-1]/100.0) * 100
                order(stock, approximationAmount)

        account.isBuyPeriod = False
        account.dayCount = 0
    elif account.dayCount == 59:          # 调仓日前一日清空当前仓位
        for stock in account.valid_secpos:
            order_to(stock,0)
        account.isBuyPeriod = True
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
19.4%	5.8%	8.8%	0.88	0.71	22.2%	0.76	30.7%	--

```
2010-04-06 买入 : ['600143.XSHG', '600588.XSHG', '600900.XSHG', '
300017.XSHE', '000825.XSHE', '600276.XSHG', '600839.XSHG', '0005
03.XSHE', '601898.XSHG', '600497.XSHG', '000878.XSHE', '601166.X
SHG', '000876.XSHE', '600036.XSHG']
2010-07-05 买入 : ['600649.XSHG', '601333.XSHG', '000425.XSHE', '
000423.XSHE', '002304.XSHE', '002310.XSHE', '000069.XSHE', '0007
93.XSHE', '601898.XSHG', '600029.XSHG', '300024.XSHE']
2010-09-30 买入 : ['600497.XSHG', '000960.XSHE', '002310.XSHE', '
601899.XSHG', '600060.XSHG', '300017.XSHE', '601006.XSHG', '3000
27.XSHE', '002399.XSHE', '600118.XSHG', '000878.XSHE', '600519.X
SHG', '600315.XSHG', '000776.XSHE']
2010-12-30 买入 : ['601006.XSHG', '600208.XSHG', '600497.XSHG', '
600436.XSHG', '601899.XSHG', '000793.XSHE', '000783.XSHE', '6008
04.XSHG', '600583.XSHG', '600369.XSHG', '000858.XSHE', '601166.X
SHG', '000538.XSHE', '600597.XSHG', '600011.XSHG', '600058.XSHG'
]
2011-04-01 买入 : ['601888.XSHG', '300058.XSHE', '601899.XSHG', '
600886.XSHG', '600415.XSHG', '000060.XSHE', '000009.XSHE', '6006
74.XSHG', '002292.XSHE', '600663.XSHG', '600362.XSHG', '601998.X
SHG', '601601.XSHG', '000686.XSHE', '000768.XSHE', '601939.XSHG'
]
2011-06-30 买入 : ['002385.XSHE', '000009.XSHE', '600085.XSHG', '
300017.XSHE', '600485.XSHG', '300015.XSHE', '000858.XSHE', '6006
64.XSHG', '002500.XSHE', '601899.XSHG', '600497.XSHG', '000630.X
SHE', '000895.XSHE', '300070.XSHE', '600036.XSHG', '600089.XSHG'
```

```
, '000778.XSHE']
2011-09-23 买入 : ['600415.XSHG', '600783.XSHG', '600832.XSHG', '
002007.XSHE', '601179.XSHG', '002470.XSHE', '002399.XSHE', '6018
99.XSHG', '000009.XSHE', '601866.XSHG', '000792.XSHE', '601216.X
SHG', '600153.XSHG', '600664.XSHG', '600867.XSHG', '002429.XSHE'
, '002008.XSHE', '601098.XSHG', '600108.XSHG', '601958.XSHG', '0
00629.XSHE', '601898.XSHG', '600436.XSHG', '600583.XSHG']
2011-12-23 买入 : ['002456.XSHE', '000793.XSHE', '600362.XSHG', '
600583.XSHG', '600348.XSHG', '600588.XSHG', '000895.XSHE', '0022
52.XSHE', '000402.XSHE', '002399.XSHE', '002429.XSHE', '601866.X
SHG', '600352.XSHG', '002603.XSHE', '600031.XSHG', '002385.XSHE'
, '601899.XSHG', '000970.XSHE', '000002.XSHE', '600369.XSHG', '0
00725.XSHE', '000060.XSHE', '000009.XSHE', '600649.XSHG']
2012-03-27 买入 : ['600832.XSHG', '601899.XSHG', '000009.XSHE', '
600998.XSHG', '600315.XSHG', '000536.XSHE', '600664.XSHG', '0024
22.XSHE', '002399.XSHE', '002400.XSHE', '600485.XSHG', '000060.X
SHE', '600029.XSHG', '002570.XSHE', '002603.XSHE', '000623.XSHE'
, '601699.XSHG', '000538.XSHE', '600221.XSHG', '000895.XSHE', '6
01818.XSHG', '600068.XSHG', '000002.XSHE', '600398.XSHG', '60015
3.XSHG']
2012-06-27 买入 : ['000793.XSHE', '600827.XSHG', '002416.XSHE', '
002570.XSHE', '000858.XSHE', '000400.XSHE', '600348.XSHG', '6016
07.XSHG', '300015.XSHE', '000536.XSHE', '002385.XSHE', '600369.X
SHG', '000009.XSHE', '000895.XSHE', '002594.XSHE', '002001.XSHE'
, '600880.XSHG', '603000.XSHG', '002007.XSHE', '600398.XSHG', '0
00983.XSHE', '000725.XSHE', '600015.XSHG', '300017.XSHE', '60123
1.XSHG']
2012-09-19 买入 : ['600516.XSHG', '600398.XSHG', '002399.XSHE', '
000869.XSHE', '000060.XSHE', '000960.XSHE', '601258.XSHG', '6018
66.XSHG', '600809.XSHG', '300124.XSHE', '600369.XSHG', '002024.X
SHE', '600315.XSHG', '002416.XSHE', '601888.XSHG', '002304.XSHE'
, '002410.XSHE', '002385.XSHE', '600029.XSHG', '600008.XSHG', '6
00518.XSHG', '000858.XSHE', '600519.XSHG', '603000.XSHG', '60011
5.XSHG', '600221.XSHG', '601933.XSHG']
2012-12-19 买入 : ['000869.XSHE', '000895.XSHE', '600369.XSHG', '
603993.XSHG', '601258.XSHG', '300146.XSHE', '600352.XSHG', '0008
58.XSHE', '601333.XSHG', '600398.XSHG', '600519.XSHG', '600015.X
SHG', '601888.XSHG', '600415.XSHG', '600031.XSHG', '600809.XSHG'
, '601555.XSHG', '000157.XSHE', '002304.XSHE', '002673.XSHE', '6
00029.XSHG', '603000.XSHG', '000728.XSHE', '000568.XSHE', '00006
0.XSHE', '600221.XSHG', '000960.XSHE']
2013-03-25 买入 : ['000623.XSHE', '000536.XSHE', '002375.XSHE', '
600406.XSHG', '601699.XSHG', '600143.XSHG', '600015.XSHG', '0023
10.XSHE', '600832.XSHG', '000869.XSHE', '000858.XSHE', '002399.X
SHE', '600549.XSHG', '600518.XSHG', '600348.XSHG', '002416.XSHE'
, '002051.XSHE', '000792.XSHE', '603000.XSHG', '600519.XSHG', '6
01258.XSHG', '601168.XSHG', '002653.XSHE', '000400.XSHE', '60189
8.XSHG', '601888.XSHG', '300124.XSHE']
2013-06-27 买入 : ['000839.XSHE', '002653.XSHE', '000629.XSHE', '
601899.XSHG', '600031.XSHG', '002422.XSHE', '600085.XSHG', '0008
25.XSHE', '600395.XSHG', '603000.XSHG', '600998.XSHG', '600570.X
SHG', '603993.XSHG', '000568.XSHE', '000858.XSHE', '300146.XSHE'
, '600362.XSHG', '600315.XSHG', '601398.XSHG', '000060.XSHE', '6
```

```
00369.XSHG', '600519.XSHG', '601600.XSHG', '601988.XSHG', '00015
7.XSHE', '600839.XSHG', '600348.XSHG']
2013-09-23 买入 : ['600873.XSHG', '600642.XSHG', '600348.XSHG', '
002422.XSHE', '601818.XSHG', '600188.XSHG', '000960.XSHE', '0023
99.XSHE', '000878.XSHE', '000858.XSHE', '002153.XSHE', '600177.X
SHG', '600519.XSHG', '600031.XSHG', '600029.XSHG', '601111.XSHG'
, '603000.XSHG', '600489.XSHG', '601898.XSHG', '000536.XSHE', '6
00398.XSHG', '603993.XSHG', '002304.XSHE', '600395.XSHG', '60160
1.XSHG', '000568.XSHE', '000983.XSHE', '600518.XSHG']
2013-12-23 买入 : ['600348.XSHG', '600664.XSHG', '600315.XSHG', '
600489.XSHG', '002653.XSHE', '601899.XSHG', '000960.XSHE', '0007
09.XSHE', '000536.XSHE', '601600.XSHG', '000858.XSHE', '002422.X
SHE', '601866.XSHG', '000629.XSHE', '603000.XSHG', '000825.XSHE'
, '600519.XSHG', '600518.XSHG', '601111.XSHG', '600188.XSHG', '6
00177.XSHG', '000983.XSHE', '601888.XSHG', '601618.XSHG', '60001
1.XSHG', '600383.XSHG', '601258.XSHG', '000878.XSHE']
2014-03-25 买入 : ['600348.XSHG', '000060.XSHE', '600664.XSHG', '
000825.XSHE', '000536.XSHE', '000568.XSHE', '600489.XSHG', '6036
99.XSHG', '601225.XSHG', '601111.XSHG', '600362.XSHG', '601166.X
SHG', '000983.XSHE', '000858.XSHE', '603000.XSHG', '002422.XSHE'
, '600115.XSHG', '600398.XSHG', '600029.XSHG', '601398.XSHG', '6
00036.XSHG', '601618.XSHG', '600177.XSHG', '601628.XSHG', '60018
8.XSHG', '603993.XSHG', '000709.XSHE', '601939.XSHG']
2014-06-23 买入 : ['600315.XSHG', '600489.XSHG', '601225.XSHG', '
601899.XSHG', '603288.XSHG', '600108.XSHG', '600348.XSHG', '0024
22.XSHE', '600436.XSHG', '600664.XSHG', '601111.XSHG', '000629.X
SHE', '002653.XSHE', '601628.XSHG', '601555.XSHG', '603000.XSHG'
, '000568.XSHE', '601601.XSHG', '000858.XSHE', '601600.XSHG', '0
00728.XSHE', '000825.XSHE', '600115.XSHG', '600383.XSHG', '00230
4.XSHE', '600188.XSHG', '601258.XSHG', '002153.XSHE']
2014-09-16 买入 : ['600348.XSHG', '601225.XSHG', '600489.XSHG', '
600664.XSHG', '000825.XSHE', '002653.XSHE', '000536.XSHE', '0009
83.XSHE', '000060.XSHE', '603000.XSHG', '600362.XSHG', '601111.X
SHG', '600518.XSHG', '000858.XSHE', '601898.XSHG', '600177.XSHG'
, '601899.XSHG', '600143.XSHG', '601600.XSHG', '600115.XSHG', '0
00568.XSHE', '600519.XSHG', '603288.XSHG', '600029.XSHG', '60031
5.XSHG', '601618.XSHG', '600383.XSHG', '000800.XSHE']
2014-12-16 买入 : ['600348.XSHG', '600664.XSHG', '600489.XSHG', '
600315.XSHG', '000983.XSHE', '600518.XSHG', '000060.XSHE', '0005
36.XSHE', '600362.XSHG', '600873.XSHG', '600739.XSHG', '603000.X
SHG', '601225.XSHG', '000825.XSHE', '000568.XSHE', '603993.XSHG'
, '601618.XSHG', '000709.XSHE', '601899.XSHG', '600519.XSHG', '0
00858.XSHE', '601168.XSHG', '601166.XSHG', '000869.XSHE', '60011
5.XSHG', '601111.XSHG', '601601.XSHG', '603288.XSHG']
2015-03-19 买入 : ['600348.XSHG', '000060.XSHE', '601225.XSHG', '
601898.XSHG', '601088.XSHG', '000568.XSHE', '603000.XSHG', '0024
22.XSHE', '601899.XSHG', '603288.XSHG', '601166.XSHG', '601111.X
SHG', '600315.XSHG', '600739.XSHG', '000709.XSHE', '600489.XSHG'
, '600519.XSHG', '600873.XSHG', '601699.XSHG', '600016.XSHG', '6
01398.XSHG', '601939.XSHG', '601009.XSHG', '000536.XSHE', '00089
5.XSHE', '601988.XSHG', '600015.XSHG', '600188.XSHG']
```

嗯？主动收益率还是正的。。。。看来这样的简单正面负面指标还不足以进行有效的区分。

七 排名选股系统

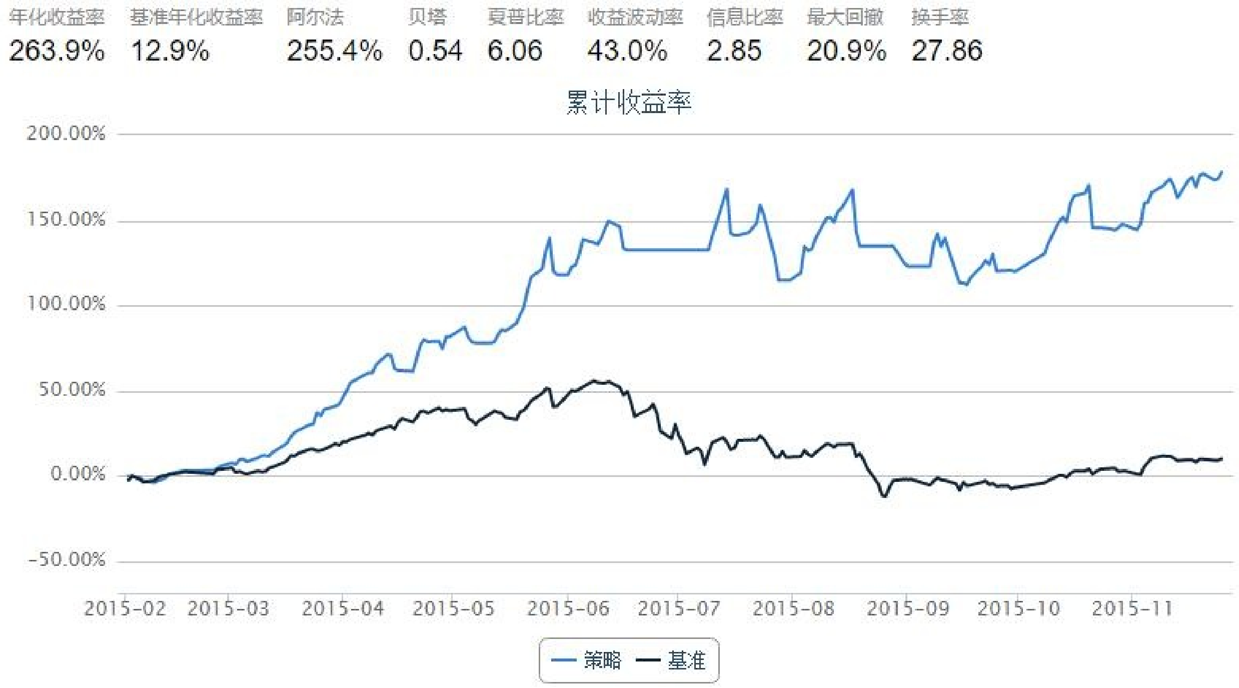
7.1 小市值投资法

学习笔记：可模拟（小市值+便宜 的修改版）

> 来源：https://uqer.io/community/share/566e867d228e5b7b41cfaf01

```
#小市值，低股价可模拟策略
import numpy as np
from heapq import nlargest, nsmallest
from CAL.PyCAL import *
import operator
start = '2015-01-01'
end  = '2015-11-25'
benchmark = 'HS300'                               # 策略参考标准
#以沪深300、中证500、创业板的并集为股票池（中间存在一定交叉，因此需要去掉重复项）
universe = list(set(set_universe('HS300')+set_universe('ZZ500')+set_universe('CYB')))
capital_base = 10000
stk_num = 10      # 持仓股票数量
refresh_rate = 1

def initialize(account):
    pass

def handle_data(account):
    cal = Calendar('China.SSE')
    # ----------------- 清洗universe -------------------------------
    date = account.current_date #类型为datetime   Date.fromDateTime(datetime) 将datetime转为Date，反过来 Date.toDateTime()将Date转为datetime
    yesterday = cal.advanceDate(date, '-1B', BizDayConvention.Following)
    yesterday = datetime(yesterday.year(), yesterday.month(), yesterday.dayOfMonth()).strftime('%Y%m%d')
    fivedays =  cal.advanceDate(date, '-5B', BizDayConvention.Following)
    fivedays = datetime(fivedays.year(), fivedays.month(), fivedays.dayOfMonth()).strftime('%Y%m%d')
    # 选出可用的300只市值最小的股票（如过用 universe = StockScreener(Factor('LCAP').nsmall(300))则不能进行模拟）
    # MktStockFactorsOneDayGet函数支持的股票池长度有限，所以分两次合成Dataframe
    LCAP = DataAPI.MktStockFactorsOneDayGet(tradeDate=yesterday,secID=account.universe[0:len(account.universe)/2],field=['LCAP','secID'])
    LCAP = LCAP.append(DataAPI.MktStockFactorsOneDayGet(tradeDate=yesterday,secID=account.universe[len(account.universe)/2:],field=['LCAP','secID']))
    LCAP = LCAP.sort_index(by = 'LCAP')
    #这里我们将股票池转移到自己定义的my_universe中，不能修改account.universe，因为一旦修改则会导致模拟无法正常进行
    my_universe =[i for i in LCAP['secID']][0:300]
    # 去除ST股
    try:
        STlist = DataAPI.SecSTGet(secID=my_universe, beginDate=yesterday, endDate=yesterday, field=['secID']).tolist()
        my_universe = [s for s in my_universe if s not in STlist]
    except:
        pass
    # 去除流动性差的股票
    tv = account.get_attribute_history('turnoverValue', 20)
    mtv = {sec: sum(tvs)/20. for sec,tvs in tv.items()}
    my_universe = [s for s in my_universe if mtv.get(s, 0) >= 10000000]
    # 去除新上市或复牌的股票
    opn = account.get_attribute_history('openPrice', 1)
    my_universe = [s for s in my_universe if not (np.isnan(opn.get(s, 0)[0]) or opn.get(s, 0)[0] == 0)]
    # 去除弱势股票
    hist_prices = account.get_attribute_history('closePrice', 5)
    hist_returns = {sec: hist_prices[sec][-1]/hist_prices[sec][0] for sec in hist_prices.keys()}
    my_universe = [s for s in my_universe if hist_returns.get(s, 0) > 0.96]
    #选出价格最小的stk_num*2只股票
    bucket = {}
    for stk in my_universe:
        bucket[stk] = account.referencePrice[stk]
    '''这里我们其实取了股价最低的 stk_num*2 只，原因在于：如果取stk_num只，
    那么一旦遇到涨停停牌等买不进的情况，就跪了；所以我们拿stk_num*2 数量的股票，
    但是却将仓位分成stk_num份，买进可以交易的前stk_num只股票'''
    buy_list = nsmallest(stk_num*2, bucket, key=bucket.get)

    # ----------------- 调仓逻辑 ---------------------------------
    clo = account.get_attribute_history('closePrice', 5)
    target_increase1 = sum(clo[stk][-1] for stk in buy_list)/sum(clo[stk][-2] for stk in buy_list)
    target_increase2 = sum(clo[stk][-2] for stk in buy_list)/sum(clo[stk][-3] for stk in buy_list)
    target_increase5 = sum(clo[stk][-1] for stk in buy_list)/sum(clo[stk][0] for stk in buy_list)
    dapan = DataAPI.MktIdxdGet(ticker=u"000300",beginDate=fivedays,endDate=yesterday,field=['closeIndex'],pandas="1")
    dapan_increase = dapan['closeIndex'][4] / dapan['closeIndex'][0]
    #止损逻辑，主要根据：最近两天的合计涨跌幅、上一天与五天前的合计涨跌幅、大盘的5天涨跌幅来作为限制条件
    #满足条件则买入股票
    if  dapan_increase >= 0.963 and target_increase1 >= 0.963 and
```

```
 target_increase2 >= 0.963 and target_increase5 >= 0.963:
    # 目前持仓中不在buy_list中的股票，清仓
        for stk in account.valid_secpos:
            if stk not in buy_list:
                order_to(stk, 0)

        money = account.referencePortfolioValue / stk_num
        for stk in buy_list:
            #不够一手最少买一手
            order_to(stk, max(int(money / account.referencePrice
[stk] / 100),1) * 100)
    #不满止损条件则清仓
    else:
        for stk in account.valid_secpos:
            order_to(stk,0)
    return
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
263.9%	12.9%	255.4%	0.54	6.06	43.0%	2.85	20.9%	27.86

本文主要是为了分享学习心得，希望能对和我一样的新人有所帮助。 写本文的动机主要是： 一、自己也是近期在优矿上开始模拟研究策略，觉得优矿提供的接口非常全面，并且注释详尽，让我这个新手对如何用python编写策略很快有了一个直观的认识（当然并不深入，但这已经非常好了），在量化这个本身就比较高门槛的领域，优矿能让新人有这种感觉和体验是十分难得和至关重要的，尤其在学习过程中，这是必不可少的一部分。 二、“为策略写代码注释是一个不错的学习方式”，这是一位朋友和我推荐的方法，这里也用自己的亲身体验和大家分享一下这个方法（大神请忽略我哈），确实对刚开是学习有很大帮助。

社区中有一篇“小市值+便宜就是Alpha”的策略，但是因为接口的原因不能模拟，刚好就以此为例：自己在给策略添加注释的同时，也将这个策略进行了一下修改，最终可以实现模拟运行，并可以考虑根据每天的交易信号实盘跟单。

本人是新人菜鸟，不足之处请大家多多见谅，多多批评指正，谢谢。

市值最小300指数

> 来源：https://uqer.io/community/share/5604fbe6f9f06c597665ef37

刷爆沪深300

策略名称： 市值最小３００指数

回测时间：２０１３－０１-０１ 到　２０１５－０９－２４

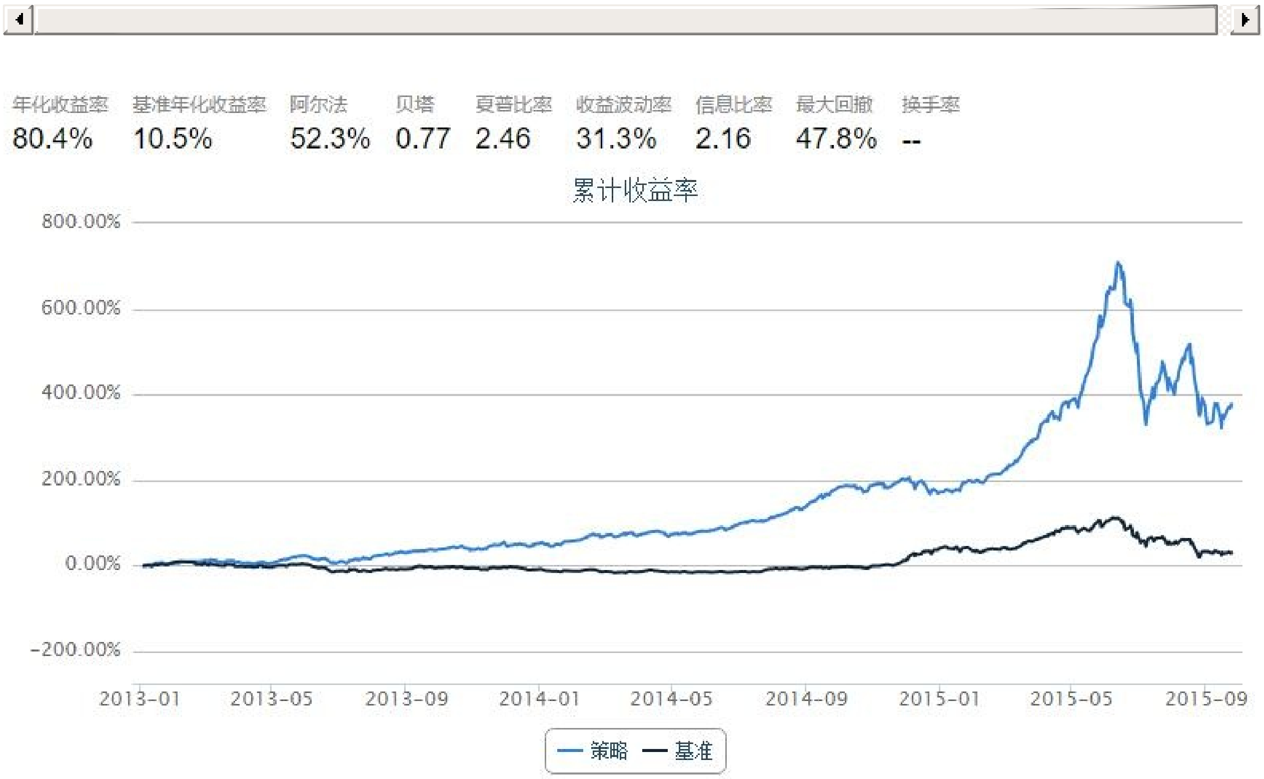
调仓期　：２０交易日

策略思想：找A股市场市值最小的300只股票，等权重构建最小３００指数

注意 ：

- 内存不够请自行缩短回测时间或universe
- 此贴有4个！！！

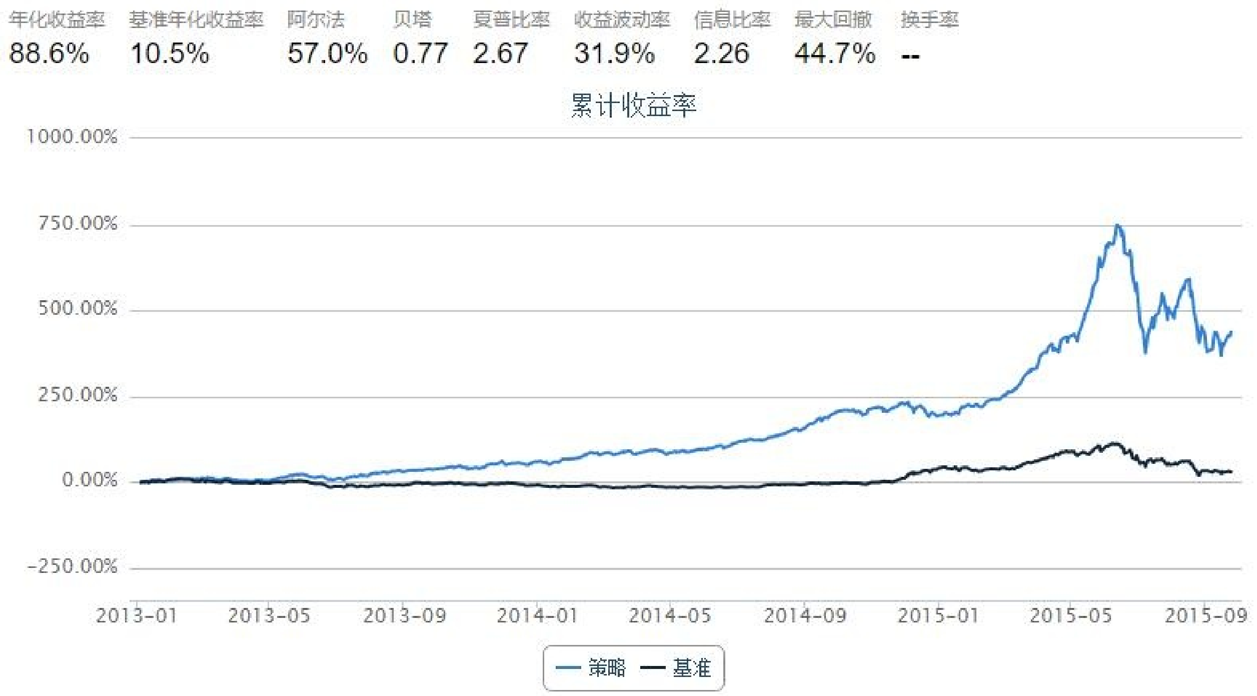
```
import pandas as pd
import numpy as np
from pandas import Series, DataFrame
start = '2013-01-01'                              # 回测起始时间
end = '2015-09-24'                                # 回测结束时间
benchmark = 'HS300'                               # 策略参考标准

universe0 = set_universe('A')                      # 证券池，支持股票和基金
universe1 = set_universe('HS300')
universe2 = set_universe('ZZ500')
universe = list(set(universe0).difference(set(universe1+universe
2)))   #最小市值股一定不在中证500和沪深300 pass

capital_base = 100000000                          # 起始资金
freq = 'd'                                        # 策略类型，'d'表示日间
策略使用日线回测，'m'表示日内策略使用分钟线回测

refresh_rate = 20                                 # 调仓频率，表示执行hand
le_data的时间间隔，若freq = 'd'时间间隔的单位为交易日，若freq = 'm'时间
间隔为分钟

def initialize(account):                          # 初始化虚拟账户状态
    pass

def handle_data(account):                         # 每个交易日的买入卖出指令


    total_money = account.referencePortfolioValue

    prices = account.referencePrice
    buylist = []

    marketValue = DataFrame()

    today = account.current_date.strftime('%Y%m%d')

    for s in range(len(account.universe)/40 + 1):

        if s == len(account.universe)/40:
            temp_list = account.universe[s*40:]
        else :
            temp_list = account.universe[s*40:(s+1)*40]

#MktEqudGet接口一次最多选50个

        try:  #排除最后一次temp_list为零的可能
            marketValue_temp = DataAPI.MktEqudGet(secID = temp_l
ist,tradeDate= today, field=u"secID,marketValue",pandas="1")

        except :
            pass
        marketValue = pd.concat([marketValue,marketValue_temp])

    marketValue = marketValue.sort('marketValue',ascending=True)
.drop_duplicates('secID')

    marketValue.set_index('secID',inplace=True)

    marketValue = marketValue.dropna()

    #排除新股发行日
    for s in list(marketValue.index) :

        if not (np.isnan(prices[s]) or prices[s] == 0) :

            buylist.append(s)

        if len(buylist) >= 300 :

            break

    sell_list = [x for x in account.valid_secpos if x not in buy
list]

    for stk in sell_list:
        order_to(stk, 0)



    for stk in buylist:

        order_to(stk, int(total_money/300/prices[stk]/100)*100)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
80.4%	10.5%	52.3%	0.77	2.46	31.3%	2.16	47.8%	--

更改权重比例 ：加权流通市值倒数（越小越买） 买买买！

```
import pandas as pd
import numpy as np
from pandas import Series, DataFrame
start = '2013-01-01'                       # 回测起始时间
end = '2015-09-24'                         # 回测结束时间
benchmark = 'HS300'                        # 策略参考标准

universe0 = set_universe('A')               # 证券池，支持股票和基金
universe1 = set_universe('HS300')
universe2 = set_universe('ZZ500')
universe = list(set(universe0).difference(set(universe1+universe
2)))

capital_base = 100000000                   # 起始资金
freq = 'd'                                 # 策略类型，'d'表示日间
策略使用日线回测，'m'表示日内策略使用分钟线回测

refresh_rate = 20                          # 调仓频率，表示执行hand
le_data的时间间隔，若freq = 'd'时间间隔的单位为交易日，若freq = 'm'时间
间隔为分钟

def initialize(account):                   # 初始化虚拟账户状态
    pass

def handle_data(account):                  # 每个交易日的买入卖出指令


    total_money = account.referencePortfolioValue
    prices = account.referencePrice

    buylist = []

    marketValue = DataFrame()

    today = account.current_date.strftime('%Y%m%d')

    for s in range(len(account.universe)/40 + 1):

        if s == len(account.universe)/40:
            temp_list = account.universe[s*40:]
        else :
            temp_list = account.universe[s*40:(s+1)*40]

#MktEqudGet接口一次最多选50个

        try:  #排除最后一次temp_list为零的可能
            marketValue_temp = DataAPI.MktEqudGet(secID = temp_list,tradeDate= today, field=u"secID,marketValue,negMarketValue",pandas="1")
        except :
            pass
        marketValue = pd.concat([marketValue,marketValue_temp])

    marketValue = marketValue.sort('marketValue',ascending=True).drop_duplicates('secID')

    marketValue.set_index('secID',inplace=True)

    marketValue = marketValue.dropna()

    #排除新股发行日
    for s in list(marketValue.index) :

        if not (np.isnan(prices[s]) or prices[s] == 0) :

            buylist.append(s)

        if len(buylist) >= 300 :

            break

    sell_list = [x for x in account.valid_secpos if x not in buylist]

    for stk in sell_list:
        order_to(stk, 0)

    #加权流通市值倒数购买
    weight_list = []

    for stk in buylist:
        weight_list.append(1.0/marketValue['negMarketValue'][stk
])

    temp_sum = 0
    for temp in weight_list:
            temp_sum += temp

    weight_list = [x/temp_sum for x in weight_list]

    i = 0
    for stk in buylist:
        order_to(stk, int(total_money*weight_list[i]/prices[stk]/
100)*100)
        i += 1
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
88.6%	10.5%	57.0%	0.77	2.67	31.9%	2.26	44.7%	--

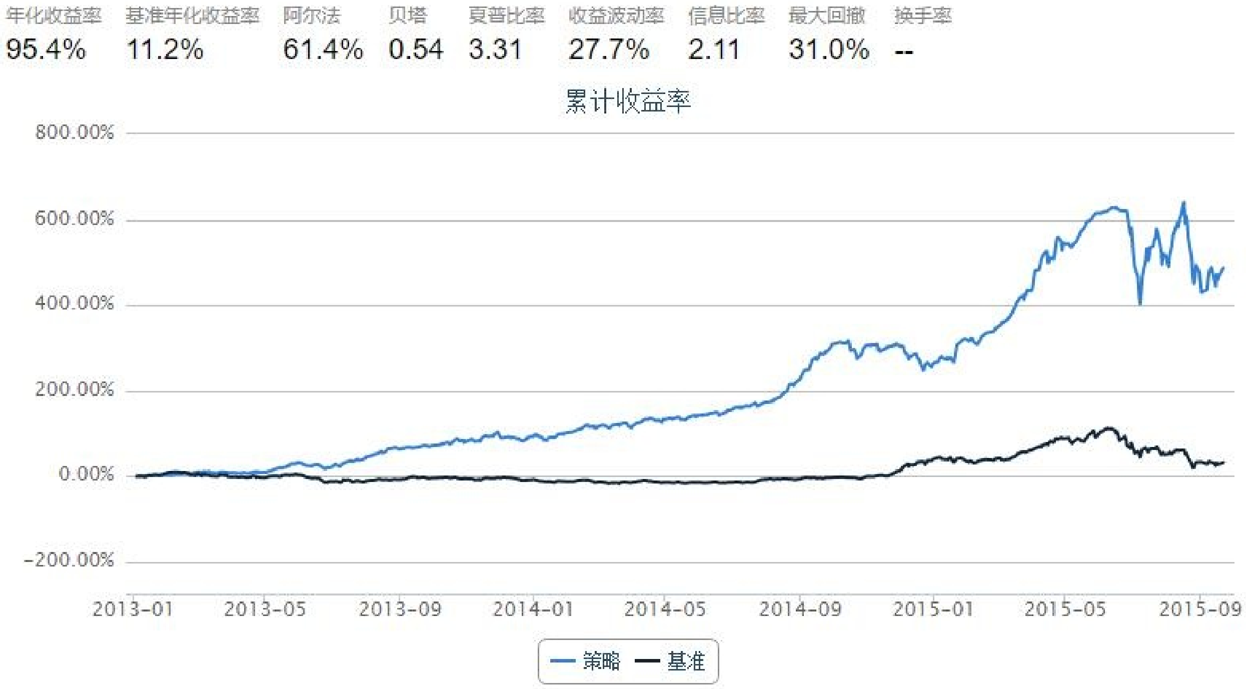
是不是在想赚钱分分钟了？　您有买３００只个股的毛爷爷吗，啊!　○_○-..

木有？我会告诉你买十只也很叼吗？？？

```
import pandas as pd
import numpy as np
from pandas import Series, DataFrame
start = '2013-01-01'                          # 回测起始时间
end = '2015-09-24'                            # 回测结束时间
benchmark = 'HS300'                           # 策略参考标准

universe0 = set_universe('A')                  # 证券池，支持股票和基金
universe1 = set_universe('HS300')
universe2 = set_universe('ZZ500')
universe = list(set(universe0).difference(set(universe1+universe
2)))
```

```
capital_base = 100000000                         # 起始资金
freq = 'd'                                       # 策略类型，'d'表示日间策略使用日线回测，'m'表示日内策略使用分钟线回测

refresh_rate = 20                                # 调仓频率，表示执行handle_data的时间间隔，若freq = 'd'时间间隔的单位为交易日，若freq = 'm'时间间隔为分钟

def initialize(account):                         # 初始化虚拟账户状态
    pass

def handle_data(account):                        # 每个交易日的买入卖出指令


    total_money = account.referencePortfolioValue

    prices = account.referencePrice

    buylist = []

    marketValue = DataFrame()

    today = account.current_date.strftime('%Y%m%d')

    for s in range(len(account.universe)/40 + 1):

        if s == len(account.universe)/40:
            temp_list = account.universe[s*40:]
        else :
            temp_list = account.universe[s*40:(s+1)*40]

#MktEqudGet接口一次最多选50个

        try:  #排除最后一次temp_list为零的可能
            marketValue_temp = DataAPI.MktEqudGet(secID = temp_list,tradeDate= today, field=u"secID,marketValue",pandas="1")

        except :
            pass
        marketValue = pd.concat([marketValue,marketValue_temp])

    marketValue = marketValue.sort('marketValue',ascending=True).drop_duplicates('secID')

    marketValue.set_index('secID',inplace=True)

    marketValue = marketValue.dropna()

    #排除新股发行日
    for s in list(marketValue.index) :

        if not (np.isnan(prices[s]) or prices[s] == 0) :
```

```
      buylist.append(s)

    if len(buylist) >= 10 :

      break

  sell_list = [x for x in account.valid_secpos if x not in buy
list]

  for stk in sell_list:
    order_to(stk, 0)

  #只买最优十只

  for stk in buylist:

    order_to(stk, int(total_money/10/prices[stk]/100)*100)
```

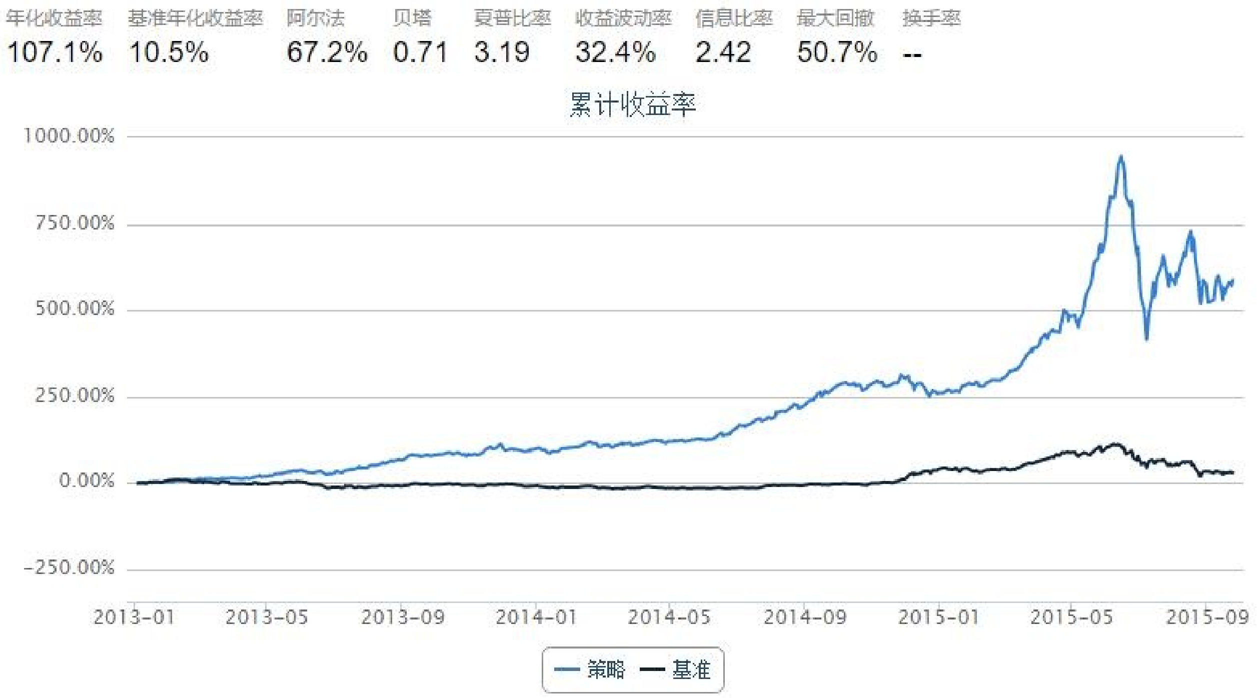


年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
95.4%	11.2%	61.4%	0.54	3.31	27.7%	2.11	31.0%	--

我会告诉你有庄家（机构持股）的股票更容易飞 ？？？ 小注：此策略中 `DataAPI.JY.EquInstShJYGet` (恒生聚源接口)暂不开放！！！活跃用户自行申请

```
import pandas as pd
import numpy as np
from pandas import Series, DataFrame
start = '2013-01-01'                       # 回测起始时间
end = '2015-09-24'                         # 回测结束时间
benchmark = 'HS300'                        # 策略参考标准
```

```
universe0 = set_universe('A')               # 证券池，支持股票和基金
universe1 = set_universe('HS300')
universe2 = set_universe('ZZ500')
universe = list(set(universe0).difference(set(universe1+universe
2)))

capital_base = 100000000                   # 起始资金
freq = 'd'                                 # 策略类型，'d'表示日间
策略使用日线回测，'m'表示日内策略使用分钟线回测

refresh_rate = 20                           # 调仓频率，表示执行han
dle_data的时间间隔，若freq = 'd'时间间隔的单位为交易日，若freq = 'm'时
间间隔为分钟

def initialize(account):                   # 初始化虚拟账户状态
    pass

def handle_data(account):                  # 每个交易日的买入卖出指令


    total_money = account.referencePortfolioValue

    prices = account.referencePrice

    buylist = []

    marketValue = DataFrame()

    today = account.current_date.strftime('%Y%m%d')

    for s in range(len(account.universe)/40 + 1):

        if s == len(account.universe)/40:
            temp_list = account.universe[s*40:]
        else :
            temp_list = account.universe[s*40:(s+1)*40]

#MktEqudGet接口一次最多选50个

        try:  #排除最后一次temp_list为零的可能
            marketValue_temp = DataAPI.MktEqudGet(secID = temp_l
ist,tradeDate= today, field=u"secID,marketValue",pandas="1")

        except :
            pass
        marketValue = pd.concat([marketValue,marketValue_temp])

    marketValue = marketValue.sort('marketValue',ascending=True)
.drop_duplicates('secID')

    marketValue.set_index('secID',inplace=True)

    marketValue = marketValue.dropna()
    # 机构持股  非第一天上市新股
    for s in list(marketValue.index) :

        try :
            # 处理巨源的数据接口没有此股
            temp =  DataAPI.JY.EquInstShJYGet ( secID = s , fiel
d = u"instNrfaPct" , pandas = "1" )

        except :
            print account.current_date.strftime('%Y-%m-%d'),' ',
s,' ','DataAPI.JY.EquInstShJYGet get wrong'
            continue

            #有机构持股 > 30%
        if temp['instNrfaPct'][0] > 30 and not (np.isnan(prices[
s]) or prices[s] == 0) :

            buylist.append(s)

        if len(buylist) >= 10 :

            break

    sell_list = [x for x in account.valid_secpos if x not in buy
list]

    for stk in sell_list:
        order_to(stk, 0)

    for stk in buylist:

        order_to(stk, int(total_money/10/prices[stk]/100)*100)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
107.1%	10.5%	67.2%	0.71	3.19	32.4%	2.42	50.7%	--

呵呵，居然看完了，还不赶紧点赞克隆去赚钱！！！内存不够的还不赶紧签到　！！！

流通市值最小股票（新筛选器版）

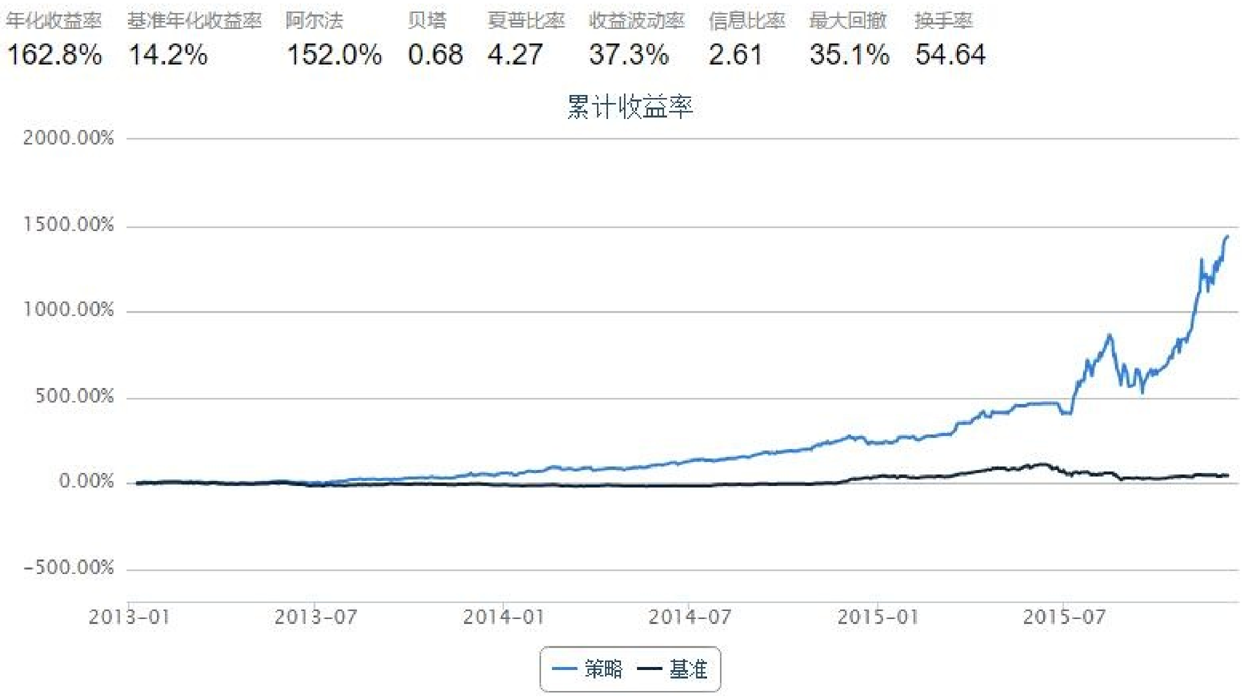
> 来源：https://uqer.io/community/share/56ee3c99228e5b887fe50dc2

策略思路

总是持有流通市值最小的10只股票

```
import numpy as np
from CAL.PyCAL import *

start = '2013-01-05'
end = '2015-12-07'
benchmark = 'HS300'
universe = StockScreener(Factor.LFLO.nsmall(20))     # LFLO 为流通市值对数，LCAP 为总市值对数
capital_base = 1000000
refresh_rate = 1
stk_num = 10     # 持仓股票数量

def initialize(account):
    pass

def handle_data(account):
    open_price = account.get_attribute_history('openPrice', 1)   # 前一日开盘价
    close_price = account.referencePrice                        # 前一日收盘价

    #  获取回测当日的前一天日期
    dt = Date.fromDateTime(account.current_date)
    cal = Calendar('China.SSE')
    last_day = cal.advanceDate(dt,'-1B',BizDayConvention.Preceding)            #计算出倒数第一个交易日
    last_day_str = last_day.strftime("%Y%m%d")

    # 市值排序
    mkt_value = DataAPI.MktEqudGet(secID=account.universe,tradeDate=last_day_str,field="secID,negMarketValue",pandas="1")
    univ_sorted = mkt_value.sort('negMarketValue').secID

    buylist = {}
    for s in univ_sorted:
        if not (np.isnan(close_price[s]) or close_price[s] == 0 or np.isnan(open_price[s]) or open_price[s] == 0):
            buylist[s] = 0

        if len(buylist) >= stk_num :     # 只持有stk_num数目的股票
```

```
      break

  for s in account.valid_secpos:
    if s not in buylist:
      order_to(s, 0)

  v = account.referencePortfolioValue / len(buylist)
  for s in buylist:
    buylist[s] = v / close_price[s] - account.valid_secpos.g
et(s, 0)

  for s in sorted(buylist, key=buylist.get):
    order(s, buylist[s])
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
162.8%	14.2%	152.0%	0.68	4.27	37.3%	2.61	35.1%	54.64

回测详情或者 `bt` 打印出来，都可以看到我们维护了一个总是持有10只最小流通市值股票的持仓；偶尔持仓股票数目会有11或者12只

```
bt
```

	tradeDate	cash	security_position	portfolio_value
0	2013-01-08	3352.2479	{u'300237.XSHE': {u'amount': 56400, u'cost': 1...	1017415.8479
1	2013-01-09	11778.5117	{u'300237.XSHE': {u'amount': 56900, u'cost': 1...	1028379.7117
	2013-01-		{u'300237.XSHE':	

2	2013-01-10	3589.4794	{u'amount': 59000, u'cost': 1...	1031625.0794
3	2013-01-11	3536.7632	{u'300237.XSHE': {u'amount': 59100, u'cost': 1...	1033944.0632
4	2013-01-14	220.7088	{u'300280.XSHE': {u'amount': 12700, u'cost': 8...	1071566.8088
5	2013-01-15	2933.8669	{u'300280.XSHE': {u'amount': 12800, u'cost': 8...	1084159.1669
6	2013-01-16	1784.7565	{u'300280.XSHE': {u'amount': 12700, u'cost': 8...	1072728.6565
7	2013-01-17	66.0136	{u'300280.XSHE': {u'amount': 12700, u'cost': 8...	1058278.1136
8	2013-01-18	80.2938	{u'300237.XSHE': {u'amount': 58600, u'cost': 1...	1068224.0938
9	2013-01-21	3050.0950	{u'300237.XSHE': {u'amount': 58600, u'cost': 1...	1068048.8950
10	2013-01-22	2609.2459	{u'300237.XSHE': {u'amount': 61900, u'cost': 1...	1036051.4459
11	2013-01-23	2045.4473	{u'300237.XSHE': {u'amount': 61000, u'cost': 1...	1026266.2473
12	2013-01-24	2926.7991	{u'300237.XSHE': {u'amount': 60700, u'cost': 1...	1000829.7991
13	2013-01-25	328.4654	{u'300237.XSHE': {u'amount': 60600, u'cost': 1...	999702.1654
14	2013-01-28	286.7192	{u'300237.XSHE': {u'amount': 60500, u'cost': 1...	1019742.9192
	2013-01-		{u'300237.XSHE':	

	29		u'cost': 1...	
16	2013-01-30	171.7371	{u'300237.XSHE': {u'amount': 61100, u'cost': 1...	1012637.1371
17	2013-01-31	386.9216	{u'300237.XSHE': {u'amount': 60700, u'cost': 1...	996104.6216
18	2013-02-01	668.2563	{u'300237.XSHE': {u'amount': 60500, u'cost': 1...	1016779.9563
19	2013-02-04	2638.0595	{u'300237.XSHE': {u'amount': 60500, u'cost': 1...	1027883.2595
20	2013-02-05	422.8302	{u'300237.XSHE': {u'amount': 61100, u'cost': 1...	1037631.5302
21	2013-02-06	102645.2421	{u'300237.XSHE': {u'amount': 61100, u'cost': 1...	1036665.5421
22	2013-02-07	350.1216	{u'300237.XSHE': {u'amount': 61000, u'cost': 1...	1040688.9216
23	2013-02-08	66.3084	{u'300237.XSHE': {u'amount': 61600, u'cost': 1...	1052094.3084
24	2013-02-18	356.0140	{u'300237.XSHE': {u'amount': 60600, u'cost': 1...	1059799.6140
25	2013-02-19	347.5874	{u'300237.XSHE': {u'amount': 60900, u'cost': 1...	1043617.1874
26	2013-02-20	1524.6353	{u'300237.XSHE': {u'amount': 60900, u'cost': 1...	1064335.0353
27	2013-02-21	5.6970	{u'300237.XSHE': {u'amount': 61100, u'cost': 1...	1055551.1970
28	2013-02-22	427.2956	{u'300237.XSHE': {u'amount': 60600,	1065969.2956

28	22	427.2956	u'cost': 1...	1065969.2956
29	2013-02-25	128.1500	{u'300237.XSHE': {u'amount': 61100, u'cost': 1...	1080711.1500
...	...	...	...	...
677	2015-10-27	2335.2216	{u'300405.XSHE': {u'amount': 21700, u'cost': 3...	9409754.2216
678	2015-10-28	3639.3976	{u'300405.XSHE': {u'amount': 21900, u'cost': 3...	9225569.3976
679	2015-10-29	2647.5406	{u'300405.XSHE': {u'amount': 21900, u'cost': 3...	9463491.5406
680	2015-10-30	3303.6906	{u'300405.XSHE': {u'amount': 21900, u'cost': 3...	9688757.6906
681	2015-11-02	11034.8026	{u'300405.XSHE': {u'amount': 23100, u'cost': 3...	9996076.8026
682	2015-11-03	29889.7366	{u'002735.XSHE': {u'amount': 26300, u'cost': 3...	10474014.7366
683	2015-11-04	3009.8936	{u'002735.XSHE': {u'amount': 26700, u'cost': 3...	10880432.8936
684	2015-11-05	1079.4096	{u'002735.XSHE': {u'amount': 26800, u'cost': 3...	10870617.4096
685	2015-11-06	104.6466	{u'002735.XSHE': {u'amount': 27600, u'cost': 3...	11390740.6466
686	2015-11-09	8962.8616	{u'002735.XSHE': {u'amount': 27500, u'cost': 3...	12093281.8616
687	2015-11-10	22992.6176	{u'002735.XSHE': {u'amount': 27400, u'cost': 3...	12110893.6176

688	11	1201562.4166	{u'amount': 28000, u'cost': 3...	13030169.4166
689	2015-11-12	113031.2706	{u'002735.XSHE': {u'amount': 28200, u'cost': 3...	14037309.2706
690	2015-11-13	5102.3686	{u'002735.XSHE': {u'amount': 29400, u'cost': 3...	12922427.3686
691	2015-11-16	21313.6426	{u'002735.XSHE': {u'amount': 29100, u'cost': 3...	13183437.6426
692	2015-11-17	1401.0406	{u'002735.XSHE': {u'amount': 29600, u'cost': 3...	12937004.0406
693	2015-11-18	173.1346	{u'002735.XSHE': {u'amount': 29700, u'cost': 3...	12168156.1346
694	2015-11-19	2455.0746	{u'002735.XSHE': {u'amount': 29400, u'cost': 3...	12936911.0746
695	2015-11-20	7478.6776	{u'002735.XSHE': {u'amount': 29700, u'cost': 3...	13013394.6776
696	2015-11-23	3168.3376	{u'002735.XSHE': {u'amount': 29700, u'cost': 3...	12619590.3376
697	2015-11-24	15418.3816	{u'002735.XSHE': {u'amount': 29400, u'cost': 3...	13665856.3816
698	2015-11-25	70214.3646	{u'002735.XSHE': {u'amount': 29400, u'cost': 3...	13704250.3646
699	2015-11-26	16885.0396	{u'002735.XSHE': {u'amount': 29600, u'cost': 3...	13902736.0396
700	2015-11-27	47391.8976	{u'002735.XSHE': {u'amount': 29100, u'cost': 3...	13367121.8976
	2015-11-		{u'002735.XSHE': {u'amount': 30100,	

701	2015-11-30	31493.4806	{u'002735.XSHE': {u'amount': 30100, u'cost': 3...	14149036.4806
702	2015-12-01	2339.8786	{u'002761.XSHE': {u'amount': 40700, u'cost': 3...	13996263.8786
703	2015-12-02	1396517.7026	{u'002761.XSHE': {u'amount': 40200, u'cost': 3...	13965267.7026
704	2015-12-03	1439379.3926	{u'002735.XSHE': {u'amount': 30900, u'cost': 3...	14861814.3926
705	2015-12-04	1479225.7226	{u'002761.XSHE': {u'amount': 41600, u'cost': 3...	15134121.7226
706	2015-12-07	1552999.2546	{u'002735.XSHE': {u'amount': 31600, u'cost': 3...	15369296.2546

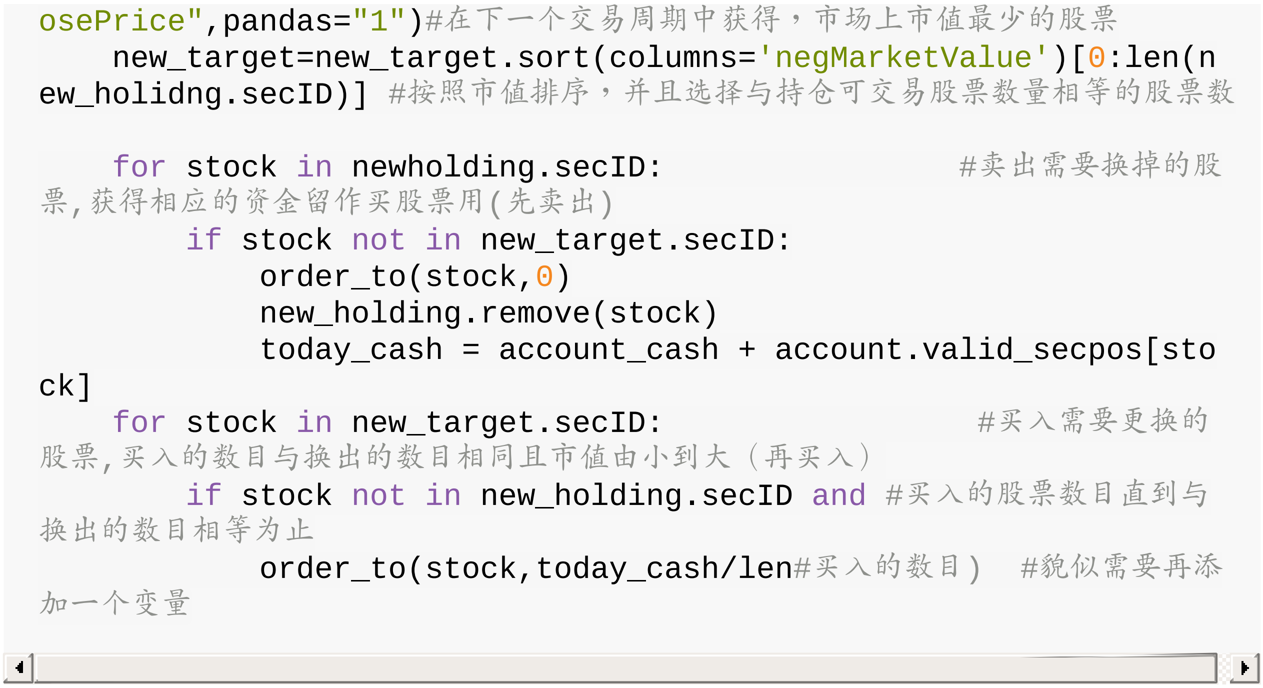
```
707 rows × 6 columns
```

持有市值最小的10只股票

> 来源：https://uqer.io/community/share/5666456ef9f06c6c8a91b52d

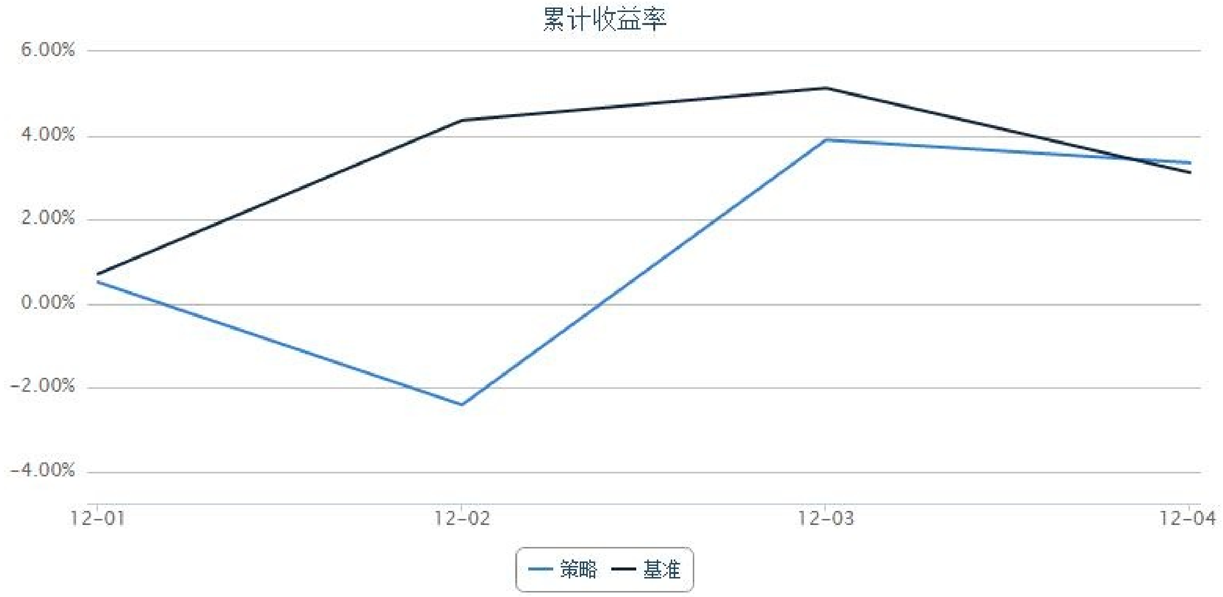
策略是一直持有沪深当中流通市值最少的10只股票。分割线前部分是第一次买入10只市值最少的股票。 分割线后部分是每次换股票的策略。1.因为持仓中可能有停牌，所以希望能把停牌的股票先排除在外，形成new持仓，就是需要换的股票，假如这里还有8只。（这里buylist应该如何传递之前的10只信息？）2.获得现在市场上市值最小的8只股票成为target，取8只是因为极端的情况下就是我把自己持仓的全部都卖出了，换成新的8只。3.判断，卖出，如果旧有的持仓中股票不在新的target里面，证明持有的市值大于target里面的，就卖出，获得资金；买入，若此时卖出了3只，则剩余5只，那么用资金平均买入target里面市值最小的3只。形成新的8只，加上原来停牌的2只，则一共10只。 分割线后部分不知道如何写，希望指导，非常感谢。

```
start = '2015-12-01'
end = '2015-12-06'
benchmark = 'HS300'
universe = StockScreener(Factor.LFLO.nsmall(20))
capital_base = 100000
freq = 'd'
refresh_rate = 1                           # 调仓频率，表示执行handle_data的时间间隔，若freq = 'd'时间间隔的单位为交易日，若freq = 'm'时间间隔为分钟
n=10
def initialize(account):                   # 初始化虚拟账户状态
    pass

def handle_data(account):                  # 每个交易日的买入卖出指令

    print "current_date = %s"%(account.current_date.strftime('%Y%m%d'))
    buy_list=[]
    holding_first=DataAPI.MktEqudGet(secID=account.universe,tradeDate=account.current_date,field="tradeDate,secID,negMarketValue,closePrice",pandas="1")#获得市场上市值最少的10只股票
    holding_first=holding_first.sort(columns='negMarketValue')[0:n] #选出第一次能交易的十只
    print holding_first
    for stk in holding_first.secID:
        if len(buy_list)<10:
            order(stk,capital_base/n)
            buy_list.append(stk)
    print buy_list   # ——————————————————————————分割线
    new_holding=DataAPI.MktEqudGet(secID='buy_list',tradeDate=account.current_date,field="tradeDate,secID,negMarketValue,closePrice",pandas="1")#在下一个交易周期中获得，当前持仓中能交易的个股信息
    new_target=DataAPI.MktEqudGet(secID=account.universe,tradeDate=account.current_date,field="tradeDate,secID,negMarketValue,cl
```



```
osePrice",pandas="1")#在下一个交易周期中获得，市场上市值最少的股票
    new_target=new_target.sort(columns='negMarketValue')[0:len(n
ew_holidng.secID)] #按照市值排序，并且选择与持仓可交易股票数量相等的股票数

    for stock in newholding.secID:                  #卖出需要换掉的股
票,获得相应的资金留作买股票用(先卖出)
        if stock not in new_target.secID:
            order_to(stock,0)
            new_holding.remove(stock)
            today_cash = account_cash + account.valid_secpos[sto
ck]
    for stock in new_target.secID:                   #买入需要更换的
股票,买入的数目与换出的数目相同且市值由小到大（再买入）
        if stock not in new_holding.secID and #买入的股票数目直到与
换出的数目相等为止
            order_to(stock,today_cash/len#买入的数目)  #貌似需要再添
加一个变量
```

年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
685.6%	581.2%	954.5%	-0.47	10.85	62.9%	0.30	2.9%	0.00

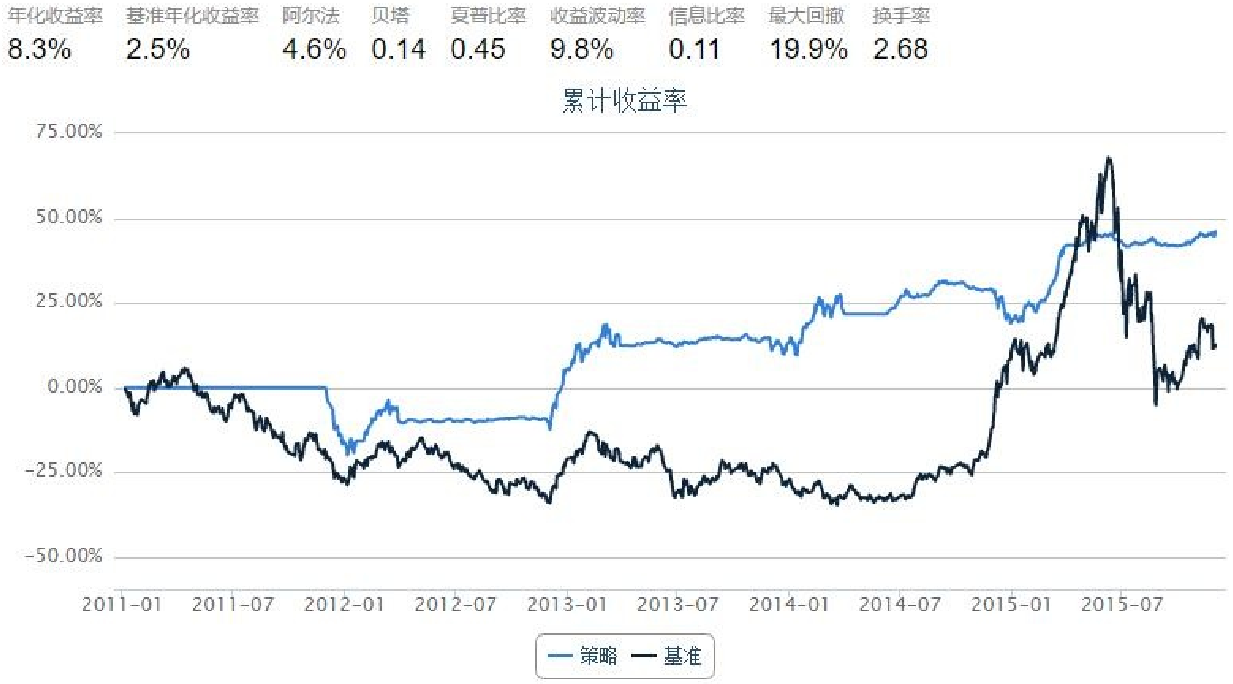


```
current_date = 20151201
     tradeDate        secID  negMarketValue  closePrice
15  2015-12-01  300466.XSHE       600200000       30.01
2   2015-12-01  002735.XSHE       929800000       46.49
17  2015-12-01  300483.XSHE       974175000       62.85
3   2015-12-01  002743.XSHE       977251400       32.21
14  2015-12-01  300464.XSHE       985959000       47.70
16  2015-12-01  300472.XSHE       988197600       59.28
13  2015-12-01  300461.XSHE       998798400       59.88
5   2015-12-01  002755.XSHE      1004984400       39.66
6   2015-12-01  002760.XSHE      1019700000       46.35
7   2015-12-01  002761.XSHE      1044600000       34.82
['300466.XSHE', '002735.XSHE', '300483.XSHE', '002743.XSHE', '30
0464.XSHE', '300472.XSHE', '300461.XSHE', '002755.XSHE', '002760
.XSHE', '002761.XSHE']
current_date = 20151202
     tradeDate        secID  negMarketValue  closePrice
13  2015-12-02  300466.XSHE       660200000       33.01
2   2015-12-02  002735.XSHE       902000000       45.10
12  2015-12-02  300464.XSHE       939658200       45.46
3   2015-12-02  002743.XSHE       945091000       31.15
16  2015-12-02  300483.XSHE       947205000       61.11
4   2015-12-02  002755.XSHE       964440400       38.06
14  2015-12-02  300472.XSHE       997032700       59.81
1   2015-12-02  002734.XSHE       999050000       30.74
10  2015-12-02  300423.XSHE      1012220000       46.01
17  2015-12-02  603009.XSHG      1026528300       38.49
['300466.XSHE', '002735.XSHE', '300464.XSHE', '002743.XSHE', '30
0483.XSHE', '002755.XSHE', '300472.XSHE', '002734.XSHE', '300423
.XSHE', '603009.XSHG']
current_date = 20151203
     tradeDate        secID  negMarketValue  closePrice
13  2015-12-03  300466.XSHE       726200000       36.31
1   2015-12-03  002735.XSHE       961200000       48.06
2   2015-12-03  002743.XSHE       979678600       32.29
4   2015-12-03  002755.XSHE      1010559200       39.88
12  2015-12-03  300464.XSHE      1020684600       49.38
0   2015-12-03  002734.XSHE      1031550000       31.74
15  2015-12-03  300483.XSHE      1041910000       67.22
5   2015-12-03  002761.XSHE      1070700000       35.69
8   2015-12-03  300391.XSHE      1071222750       20.73
16  2015-12-03  603009.XSHG      1082001900       40.57
['300466.XSHE', '002735.XSHE', '002743.XSHE', '002755.XSHE', '30
0464.XSHE', '002734.XSHE', '300483.XSHE', '002761.XSHE', '300391
.XSHE', '603009.XSHG']
current_date = 20151204
     tradeDate        secID  negMarketValue  closePrice
12  2015-12-04  300466.XSHE       798800000       39.94
2   2015-12-04  002743.XSHE       994241800       32.77
3   2015-12-04  002755.XSHE      1028804000       40.60
0   2015-12-04  002734.XSHE      1031875000       31.75
11  2015-12-04  300464.XSHE      1051482900       50.87
14  2015-12-04  300483.XSHE      1059735000       68.37
15  2015-12-04  603009.XSHG      1064399700       39.91
1   2015-12-04  002735.XSHE      1070048000       47.77
16  2015-12-04  603022.XSHG      1088400000       54.42
10  2015-12-04  300423.XSHE      1095160000       49.78
['300466.XSHE', '002743.XSHE', '002755.XSHE', '002734.XSHE', '30
0464.XSHE', '300483.XSHE', '603009.XSHG', '002735.XSHE', '603022
.XSHG', '300423.XSHE']
```

10% smallest cap stock

> 来源：https://uqer.io/community/share/5663e2f4f9f06c6c8a91b391

```
import numpy as np
start = '2011-01-05'                       # 回测起始时间
end = '2015-12-01'                         # 回测结束时间
benchmark = 'HS300'                        # 策略参考标准
universe = StockScreener(Factor.LCAP.nsmall(40))
capital_base = 100000                      # 起始资金
freq = 'd'                                 # 策略类型，'d'表示日间
策略使用日线回测，'m'表示日内策略使用分钟线回测
refresh_rate = 1                           # 调仓频率，表示执行hand
le_data的时间间隔，若freq = 'd'时间间隔的单位为交易日，若freq = 'm'时间
间隔为分钟

def initialize(account):                   # 初始化虚拟账户状态
    account.empty = True

def handle_data(account):                  # 每个交易日的买入卖出指令

    today = account.current_date
    if today.month == 12 and account.empty:
        account.empty = False
        for stock in account.universe:
            p = account.referencePrice.get(stock, 0)
            if np.isnan(p) or p == 0:
                continue
            order_pct_to(stock, 0.025)
    elif today.month == 4 and not account.empty:
        account.empty = True
        for stock in account.universe:
            if stock in account.valid_secpos:
                order_to(stock,0)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
8.3%	2.5%	4.6%	0.14	0.45	9.8%	0.11	19.9%	2.68

7.2 羊驼策略

羊驼策略

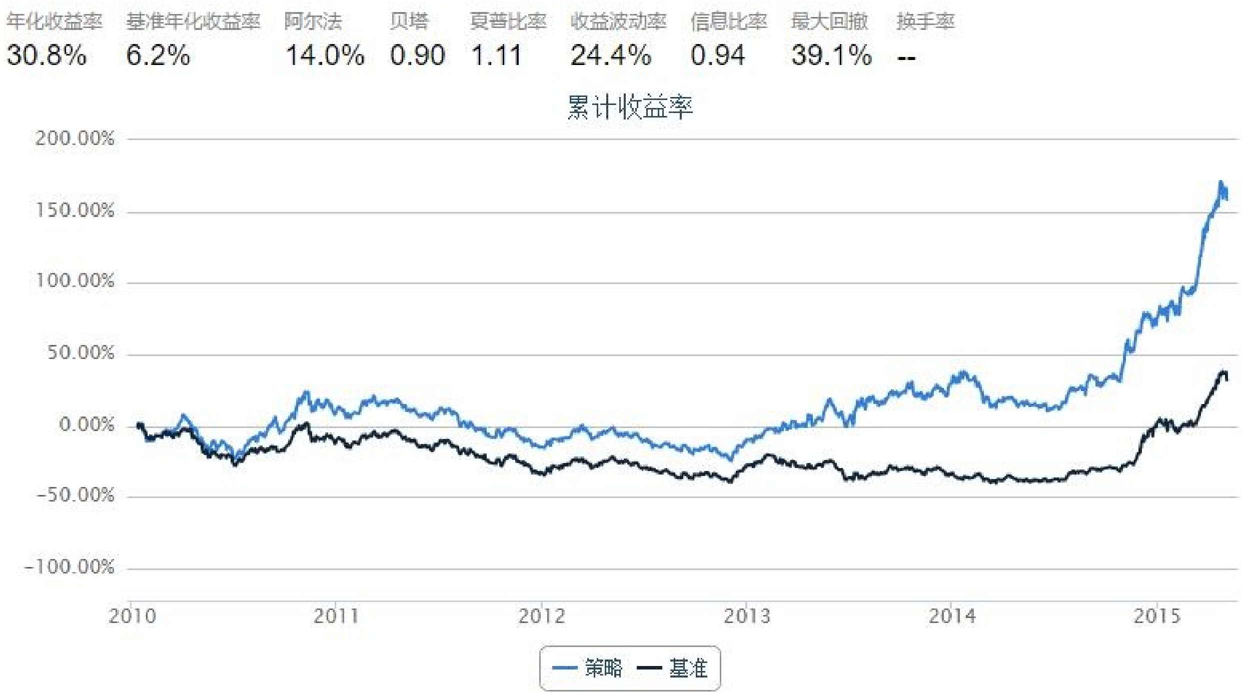
策略实现

羊驼做为上古十大神兽之一，选股祥瑞，名号响亮，本策略由一个羊驼类负责每周生成买入卖出信号，验证羊驼是否名实相符.

- 投资域 ：沪深300成分股
- 业绩基准 ：沪深300指数
- 调仓频率 ：5个交易日
- 买入卖出信号 ：初始时任意买10只羊驼,每次调仓时,剔除收益最差的一只羊驼,再任意买一只羊驼.
- 回测周期 ：2014年1月1日 至2015年5月5日

```
import numpy as np
import operator
from datetime import datetime

start = datetime(2010, 1, 1)
end   = datetime(2015, 5, 5)
benchmark = 'HS300'
universe  = set_universe('HS300')
capital_base = 100000
longest_history = 10
refresh_rate = 5
```

```
def initialize(account):
    account.stocks_num = 10

def handle_data(account):
    hist_prices = account.get_attribute_history('closePrice', 5)

    yangtuos = list(YangTuo(set(account.universe)-set(account.valid_secpos.keys()), account.stocks_num))
    cash = account.cash

    if account.stocks_num == 1:
        hist_returns = {}
        for stock in account.valid_secpos:
            hist_returns[stock] = hist_prices[stock][-1]/hist_prices[stock][0]

        sorted_returns = sorted(hist_returns.items(), key=operator.itemgetter(1))
        sell_stock = sorted_returns[0][0]

        cash = account.cash + hist_prices[sell_stock][-1]*account.valid_secpos.get(sell_stock)
        order_to(sell_stock, 0)
    else:
        account.stocks_num = 1

    for stock in yangtuos:
        order(stock, cash/len(yangtuos)/hist_prices[stock][-1])


class YangTuo:
    def __init__(self, caoyuan=[], count=10):
        self.count = count
        self.i = 0
        self.caoyuan = list(caoyuan)

    def __iter__(self):
        return self

    def next(self):
        if self.i < self.count:
            self.i += 1
            return self.caoyuan.pop(np.random.randint(len(self.caoyuan)))
        else:
            raise StopIteration()
```



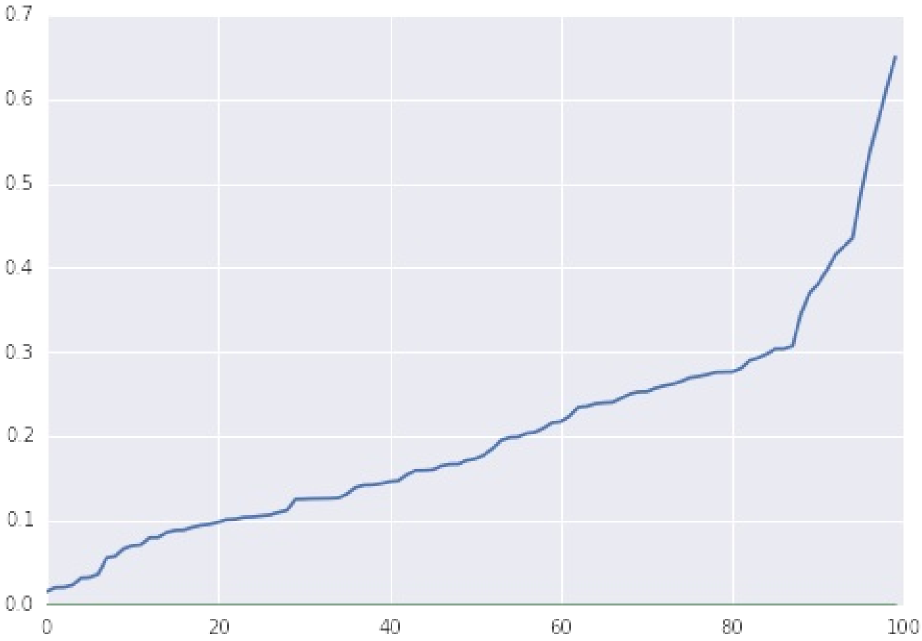
年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
30.8%	6.2%	14.0%	0.90	1.11	24.4%	0.94	39.1%	--

也许你会说,这只是运气好,并不能说明羊驼的厉害啊!好,接下来我们运行100次,看看羊驼的威力.

```
start = datetime(2010, 1, 1)
end   = datetime(2015, 5, 5)
benchmark = 'HS300'
universe  = set_universe('HS300')
capital_base = 100000

sim_params = quartz.sim_condition.env.SimulationParameters(start
, end, benchmark, universe, capital_base)
idxmap_all, data_all = quartz.sim_condition.data_generator.get_d
aily_data(sim_params)
```

```
import numpy as np
import operator

longest_history = 10
refresh_rate = 5

def initialize(account):
    account.stocks_num = 10

def handle_data(account):
    hist_prices = account.get_attribute_history('closePrice', 5)

    yangtuos = list(YangTuo(set(account.universe)-set(account.va
lid_secpos.keys()), account.stocks_num))
    cash = account.cash
    if account.stocks_num == 1:
        hist_returns = {}
        for stock in account.valid_secpos:
            hist_returns[stock] = hist_prices[stock][-1]/hist_pr
ices[stock][0]

        sorted_returns = sorted(hist_returns.items(), key=operat
or.itemgetter(1))
        sell_stock = sorted_returns[0][0]

        cash = account.cash + hist_prices[sell_stock][-1]*accoun
t.valid_secpos.get(sell_stock)
        order_to(sell_stock, 0)
    else:
        account.stocks_num = 1

    for stock in yangtuos:
        order(stock, cash/len(yangtuos)/hist_prices[stock][-1])


class YangTuo:
    def __init__(self, caoyuan=[], count=10):
        self.count = count
        self.i = 0
        self.caoyuan = list(caoyuan)

    def __iter__(self):
        return self

    def next(self):
        if self.i < self.count:
            self.i += 1
            return self.caoyuan.pop(np.random.randint(len(self.c
aoyuan)))
        else:
            raise StopIteration()

strategy = quartz.sim_condition.strategy.TradingStrategy(initial
ize, handle_data)
perfs = []
for i in xrange(100):
    bt, acct = quartz.quick_backtest(sim_params, strategy, idxma
p_all, data_all, refresh_rate = refresh_rate, longest_history=lo
ngest_history)
    perf = quartz.perf_parse(bt, acct)
    perfs.append(perf)
```

```
from matplotlib import pylab
import seaborn
x = sorted([p['annualized_return']-p['benchmark_annualized_retur
n'] for p in perfs])
pylab.plot(x)
pylab.plot([0]*len(x))

[<matplotlib.lines.Line2D at 0x7702a10>]
```



100%的胜率! 大家闭着眼睛,跟着羊驼买就行了!

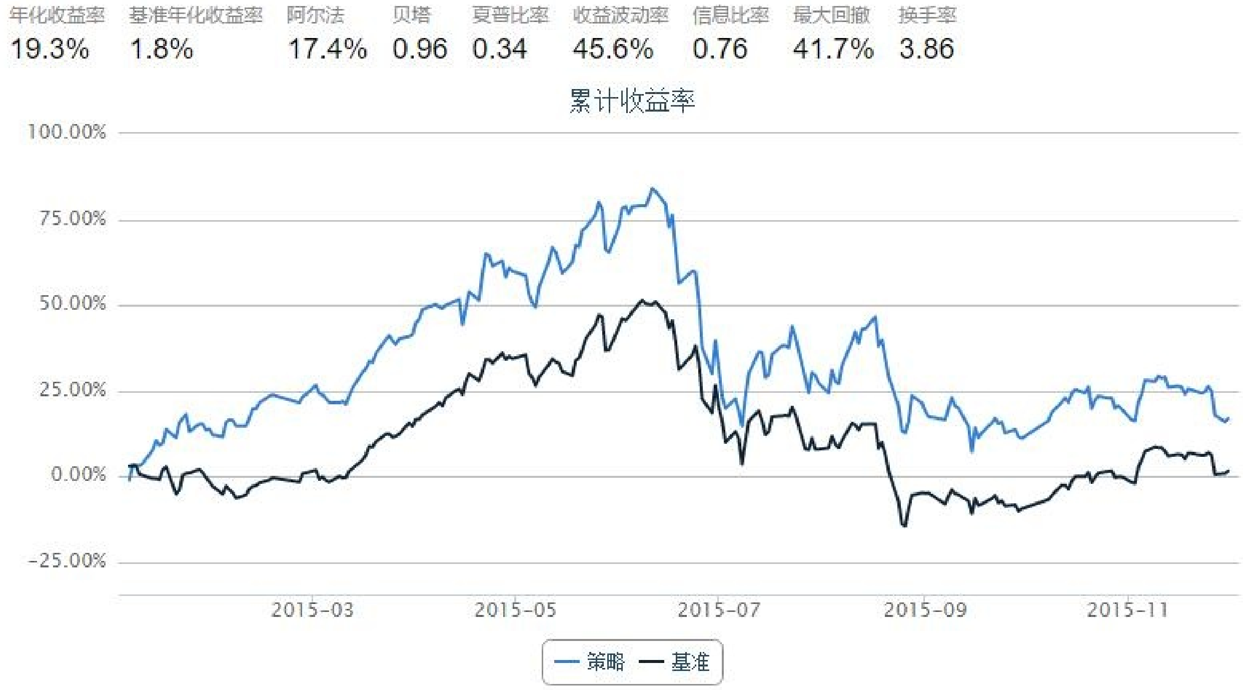
接下来的工作:

由于指数并没有分红等概念, 直接拿HS300指数做benchmark, 对HS300并不公平. 所以接下来考虑把benchmark换成某只指数基金, 再做对比.

羊驼反转策略（修改版）

> 来源：https://uqer.io/community/share/566c0e3cf9f06c6c8a91ceec

```
# 第一步：设置基本参数
start = '2015-01-01'                             # 回测起始时间
end   = '2015-12-01'                             # 回测结束时间
capital_base = 1000000                           # 起始资金
refresh_rate = 5                                 # 调仓频率
benchmark = 'HS300'                              # 策略参考标准
freq = 'd'                                     # 策略类型，'d'表示日间
策略使用日线回测

# 第二步：选择主题，设置股票池
universe = set_universe('HS300')                                  #
股票池

import numpy as np
import pandas as pd

def initialize(account):                       # 初始化虚拟账户状态
    account.stocks_num=10


def handle_data(account):                      # 每个交易日的买入卖出指令

    if account.stocks_num==10:                                #第一天
交易使用buylist
        account.stocks_num=1
        keylist=[]
        data=DataAPI.MktStockFactorsOneDayGet(tradeDate=account.
current_date,secID=account.universe,ticker=u"",field=['secID','R
EVS10'],pandas="1")    #获取start前一日股票池中十日收益
        keylist=data.dropna().sort(columns='REVS10',ascending=Fa
lse).tail(10)['secID'].values.tolist()                #将十日
收益最差的十只股票组成list
        #hist_prices = account.get_attribute_history('closePrice
', 1)
        for i in keylist:
            order(i,100000/account.referencePrice[i])
    else:
        sellist=[]
        replacelist=[]
        keylist=[]
        for key in account.valid_secpos.keys():
            keylist.append(key)

        sell=DataAPI.MktStockFactorsOneDayGet(tradeDate=account.
current_date,secID=keylist,ticker=u"",field=['secID','REVS10'],p
andas="1") #获得十日账户中所有股票的收益
        sellist.append(sell.min()['secID'])
                                  #找出收益最差的股票加入sellist

        replace=DataAPI.MktStockFactorsOneDayGet(tradeDate=accou
nt.current_date,secID=universe,ticker=u"",field=['secID','REVS10'
],pandas="1")   #获得股票池中十日以来
        replace=replace.set_index('secID').drop(keylist).dropna(
)
        replace=replace.sort(columns='REVS10',ascending=False).t
ail(1).reset_index()['secID'].values.tolist()     #获得收益最差的股
票作为账户中新的代替股票

        keylist.remove(sellist[0])
        replacelist=replacelist+replace
        keylist.append(replacelist[0])
        #print keylist
        for stk in sellist:
            order_to(stk, 0)
        for stk in replacelist:
            order(stk,account.cash/account.referencePrice[stk])
        #print account.valid_secpos
```

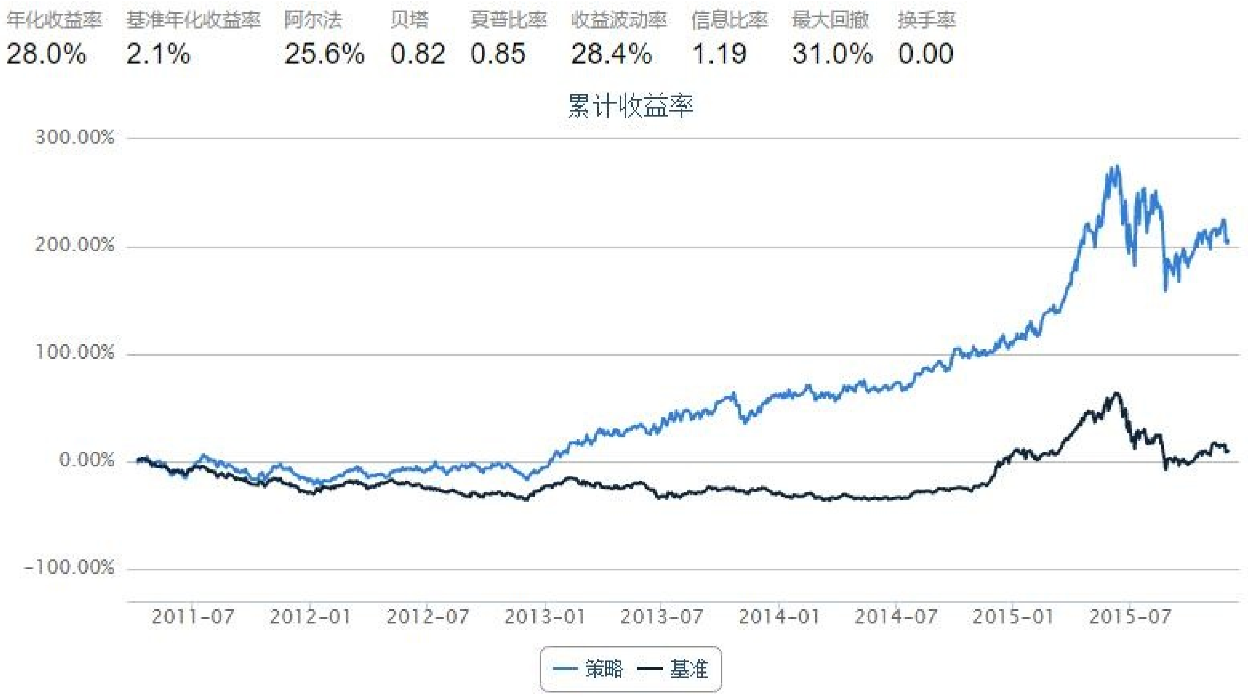


年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
19.3%	1.8%	17.4%	0.96	0.34	45.6%	0.76	41.7%	3.86

羊驼反转策略

> 来源：https://uqer.io/community/share/566982a4f9f06c6c8a91b7a2

```
# 第一步：设置基本参数
start = '2011-04-01'                             # 回测起始时间
end   = '2015-12-01'                             # 回测结束时间
capital_base = 1000000                           # 起始资金
refresh_rate = 10                                 # 调仓频率
benchmark = 'HS300'                              # 策略参考标准
freq = 'd'                                     # 策略类型，'d'表示日间
策略使用日线回测

# 第二步：选择主题，设置股票池
universe = set_universe('HS300')                                  #
股票池

import numpy as np
import pandas as pd


data=DataAPI.MktStockFactorsOneDayGet(tradeDate='20110331',secID
=universe,ticker=u"",field=['secID','REVS10'],pandas="1")     #获
取start前一日股票池中十日收益
buylist=data.dropna().sort(columns='REVS10',ascending=False).tai
l(10)['secID'].values.tolist()                          #将十日收益最差的
十只股票组成list


def initialize(account):                       # 初始化虚拟账户状态
    account.stocks_num=10


def handle_data(account):                      # 每个交易日的买入卖出指令

    if account.stocks_num==10:                                #第一天
交易使用buylist
        global buylist
        account.universe=buylist
        hist_prices = account.get_attribute_history('closePrice'
, 1)
        for i in account.universe:
            order(i,100000/hist_prices[i][0])

    account.stocks_num=1                                   #之后为非第
一天交易策略

    sellist=[]
    replacelist=[]
    sell=DataAPI.MktStockFactorsOneDayGet(tradeDate=account.current_date,secID=account.universe,ticker=u"",field=['secID','REVS10'],pandas="1")        #获得十日以来账户中所有股票的收益
    sellist.append(sell.min()['secID'])                                                                      #找出收益最差的股票加入sellist

    replace=DataAPI.MktStockFactorsOneDayGet(tradeDate=account.current_date,secID=universe,ticker=u"",field=['secID','REVS10'],pandas="1")           #获得股票池中十日以来所有股票的收益
    replace=replace.set_index('secID').drop(buylist)
    replace=replace.dropna().sort(columns='REVS10',ascending=False).tail(1).reset_index()['secID'].values.tolist()                #获得收益最差的股票作为账户中新的代替股票
    replacelist.append(replace)

    account.universe.remove(sell.min()['secID'])
    account.universe=account.universe+replacelist[0]

    hist_prices = account.get_attribute_history('closePrice', 1)                                            #获取前一个交易日账户股票价格

    for stk in sellist:
        order_to(stk,0)
    for stk in account.universe:
        order(stk,account.cash/10/hist_prices[stk][0])
```

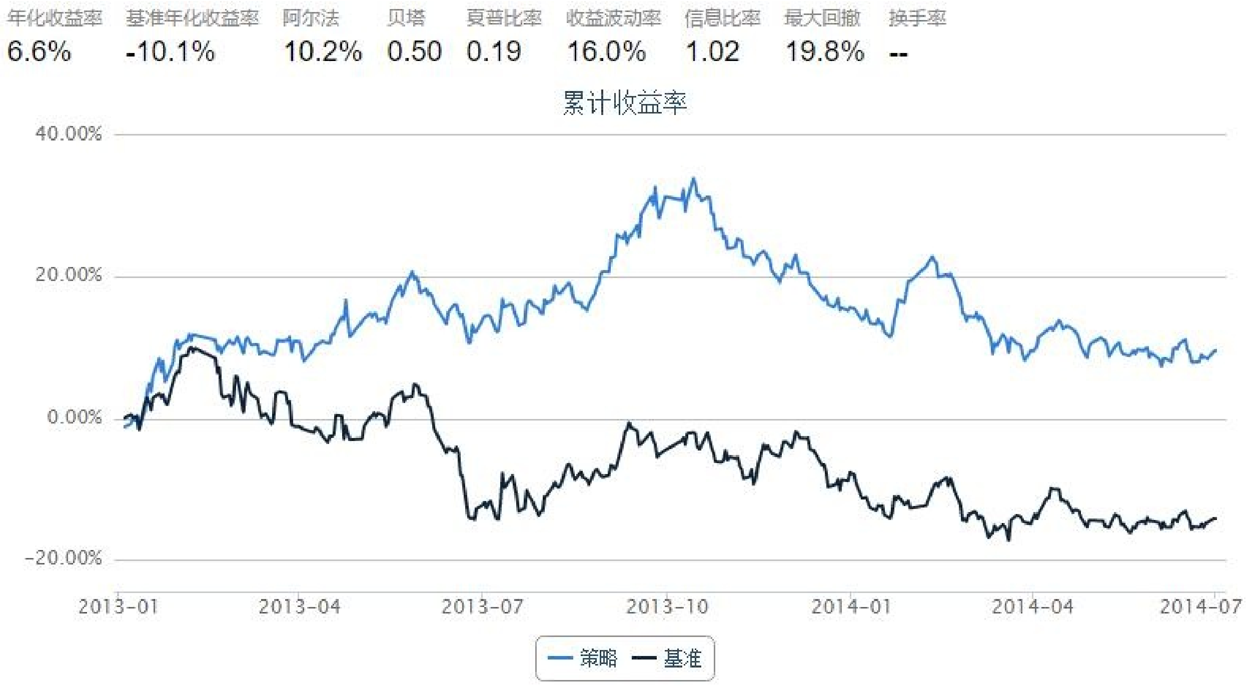


年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
28.0%	2.1%	25.6%	0.82	0.85	28.4%	1.19	31.0%	0.00

我的羊驼策略，选5只股无脑轮替

> 来源：https://uqer.io/community/share/561290f2f9f06c4ca72fb5ab

```
from CAL.PyCAL import Date
from CAL.PyCAL import Calendar
from CAL.PyCAL import BizDayConvention
import numpy as np

start = '2013-01-01'                       # 回测起始时间
end = '2014-07-01'                         # 回测结束时间
benchmark = 'HS300'                        # 策略参考标准
universe = set_universe('HS300')    # 证券池，支持股票和基金
capital_base = 100000                      # 起始资金
freq = 'd'                                 # 策略类型，'d'表示日间策略使用日线回测，'m'表示日内策略使用分钟线回测
refresh_rate = 5                       # 调仓频率，表示执行handle_data的时间间隔，若freq = 'd'时间间隔的单位为交易日，若freq = 'm'时间间隔为分钟

#EGRO 5年收益增长率
#GrossIncomeRatio 毛利率
#NetProfitGrowRate 净利润同比增长
#DEGM 毛利率增长，去年同期相比
#OperatingRevenueGrowRate 营业收入同比增长
#ROE 权益回报率
#DebtsAssetRatio 负债资产率
#EPS 每股收益

def initialize(account):                   # 初始化虚拟账户状态
    pass

def handle_data(account):                  # 每个交易日的买入卖出指令

    buylist=[]
    selist=[]

    dt = Date.fromDateTime(account.current_date)
    cal = Calendar('China.SSE')
    lastTDay = cal.advanceDate(dt,'-1B',BizDayConvention.Preceding)
    last_date=lastTDay.strftime('%Y%m%d')
    getData=DataAPI.MktStockFactorsOneDayGet(tradeDate=last_date,secID=account.universe,field=['secID','NetProfitGrowRate'],pandas="1")
    getData.set_index('secID',inplace=True)
    getData=getData[getData.NetProfitGrowRate>=1.0].dropna()
    getData=getData.sort(columns='NetProfitGrowRate',ascending=False)
    getData=getData.head(20)
    getData['profitRatio']=np.nan

    try:
        for stock in list(getData.index):
            getData['profitRatio'][stock]=(account.get_symbol_history(stock,refresh_rate)['closePrice'][-1]/account.get_symbol_history(stock,refresh_rate)['closePrice'][0])
    except:
        print 'GET PROFITRATIO ERROR!!!'

    getData=getData.sort(columns='profitRatio',ascending=False)
    getData=getData.head(5)
    print "getData:",getData

    for stock in list(getData.index):
        buylist.append(stock)
    print "buylist:",buylist

    for stock in account.valid_secpos:
        if(stock in buylist):
            pass
        else:
            order_to(stock,0)


    for stock in buylist:
        if(stock in account.valid_secpos):
            pass
        else:
            order(stock,account.cash/account.referencePrice[stock]/len(buylist))


    print "日期：",account.current_date,"，持仓：",account.valid_secpos
```



```
GET PROFITRATIO ERROR!!!
getData:              NetProfitGrowRate  profitRatio
secID
000883.XSHE            31.0717          NaN
000413.XSHE            10.4768          NaN
000046.XSHE             2.3237          NaN
600157.XSHG             2.1672          NaN
002450.XSHE             1.7749          NaN
buylist: ['000883.XSHE', '000413.XSHE', '000046.XSHE', '600157.X
SHG', '002450.XSHE']
日期： 2013-01-04 00:00:00 ,持仓： {}
getData:              NetProfitGrowRate  profitRatio
secID
002008.XSHE             1.1794     1.087628
300058.XSHE             1.2371     1.056438
600157.XSHG             2.1672     1.041454
000883.XSHE            31.0717     1.034817
002450.XSHE             1.7749     1.032616
buylist: ['002008.XSHE', '300058.XSHE', '600157.XSHG', '000883.X
SHE', '002450.XSHE']
日期： 2013-01-11 00:00:00 ,持仓： {'600157.XSHG': 5822, '002450.X
SHE': 1824.0, '000883.XSHE': 6011, '000046.XSHE': 3943, '000413.
XSHE': 5056}
getData:              NetProfitGrowRate  profitRatio
secID
600157.XSHG             2.1672     1.113170
002450.XSHE             1.7749     1.103866
002008.XSHE             1.1794     1.092182
300058.XSHE             1.2371     1.083742
600108.XSHG             1.1371     1.081443
buylist: ['600157.XSHG', '002450.XSHE', '002008.XSHE', '300058.X
SHE', '600108.XSHG']
```

```
日期： 2013-01-18 00:00:00 ,持仓： {'600157.XSHG': 5822, '002450.X
SHE': 1824.0, '000883.XSHE': 6011}
getData:              NetProfitGrowRate  profitRatio
secID
600157.XSHG             2.1672     1.024451
300058.XSHE             1.2371     1.011067
600383.XSHG             1.0580     1.000000
600108.XSHG             1.1371     0.984742
002450.XSHE             1.7749     0.957582
buylist: ['600157.XSHG', '300058.XSHE', '600383.XSHG', '600108.X
SHG', '002450.XSHE']
日期： 2013-01-25 00:00:00 ,持仓： {'002450.XSHE': 1824.0, '600108
.XSHG': 1104, '600157.XSHG': 5822, '300058.XSHE': 1315, '002008.
XSHE': 824}
getData:              NetProfitGrowRate  profitRatio
secID
600157.XSHG             2.1672     1.112213
600011.XSHG             1.5759     1.078102
600383.XSHG             1.0580     1.075537
002008.XSHE             1.1794     1.037387
300058.XSHE             1.2371     1.028319
buylist: ['600157.XSHG', '600011.XSHG', '600383.XSHG', '002008.X
SHE', '300058.XSHE']
日期： 2013-02-01 00:00:00 ,持仓： {'002450.XSHE': 1824.0, '600108
.XSHG': 1104, '600157.XSHG': 5822, '300058.XSHE': 1315, '600383.
XSHG': 1080}
getData:              NetProfitGrowRate  profitRatio
secID
002008.XSHE             1.1794     1.099718
000046.XSHE             2.3237     1.076794
600383.XSHG             1.0580     1.069296
000413.XSHE            10.4768     1.015821
600011.XSHG             1.5759     1.004408
buylist: ['002008.XSHE', '000046.XSHE', '600383.XSHG', '000413.X
SHE', '600011.XSHG']
日期： 2013-02-08 00:00:00 ,持仓： {'600157.XSHG': 5822, '002008.X
SHE': 727, '300058.XSHE': 1315, '600011.XSHG': 1257, '600383.XSH
G': 1080}
getData:              NetProfitGrowRate  profitRatio
secID
300058.XSHE             1.2371     1.022028
600108.XSHG             1.1371     0.997116
002450.XSHE             1.7749     0.971011
000413.XSHE            10.4768     0.962690
002008.XSHE             1.1794     0.955920
buylist: ['300058.XSHE', '600108.XSHG', '002450.XSHE', '000413.X
SHE', '002008.XSHE']
日期： 2013-02-22 00:00:00 ,持仓： {'002008.XSHE': 727, '000413.XS
HE': 2549, '600011.XSHG': 1257, '600383.XSHG': 1080, '000046.XSH
E': 2003}
getData:              NetProfitGrowRate  profitRatio
secID
002008.XSHE             1.1794     1.050923
```

```
002450.XSHE             1.7749     1.050195
600383.XSHG             1.0580     1.047712
000046.XSHE             2.3237     1.032878
600011.XSHG             1.5759     1.023239
buylist: ['002008.XSHE', '002450.XSHE', '600383.XSHG', '000046.X
SHE', '600011.XSHG']
日期： 2013-03-01 00:00:00 ,持仓： {'002450.XSHE': 1122, '600108.X
SHG': 1941, '000413.XSHE': 2549, '300058.XSHE': 1855, '002008.XS
HE': 727}
getData:              NetProfitGrowRate  profitRatio
secID
600011.XSHG             1.5759     1.057045
000883.XSHE            31.0717     1.044418
002450.XSHE             1.7749     1.041435
000413.XSHE            10.4768     1.032852
002008.XSHE             1.1794     1.030952
buylist: ['600011.XSHG', '000883.XSHE', '002450.XSHE', '000413.X
SHE', '002008.XSHE']
日期： 2013-03-08 00:00:00 ,持仓： {'002450.XSHE': 1122, '002008.X
SHE': 727, '600383.XSHG': 1466, '600011.XSHG': 1769, '000046.XSH
E': 2043}
getData:              NetProfitGrowRate  profitRatio
secID
002456.XSHE            14.5041     1.082176
002450.XSHE             1.7749     1.055441
000883.XSHE            31.0717     1.009892
000413.XSHE            10.4768     0.966968
600011.XSHG             1.5759     0.958285
buylist: ['002456.XSHE', '002450.XSHE', '000883.XSHE', '000413.X
SHE', '600011.XSHG']
日期： 2013-03-15 00:00:00 ,持仓： {'002450.XSHE': 1122, '002008.X
SHE': 727, '000883.XSHE': 3341, '600011.XSHG': 1769, '000413.XSH
E': 2801}
getData:              NetProfitGrowRate  profitRatio
secID
002456.XSHE            14.5041     1.092847
600383.XSHG             1.0580     1.081583
002008.XSHE             1.1794     1.072461
000046.XSHE             2.3237     1.053841
600108.XSHG             1.1371     1.044520
buylist: ['002456.XSHE', '600383.XSHG', '002008.XSHE', '000046.X
SHE', '600108.XSHG']
日期： 2013-03-22 00:00:00 ,持仓： {'002450.XSHE': 1122, '000883.X
SHE': 3341, '600011.XSHG': 1769, '002456.XSHE': 697, '000413.XSH
E': 2801}
getData:              NetProfitGrowRate  profitRatio
secID
000895.XSHE             1.0951     1.100220
002456.XSHE            14.5041     1.010301
000413.XSHE            10.4768     0.996685
600383.XSHG             1.0580     0.987662
002450.XSHE             2.2432     0.977696
buylist: ['000895.XSHE', '002456.XSHE', '000413.XSHE', '600383.X
SHG', '002450.XSHE']
日期： 2013-03-29 00:00:00 ,持仓： {'600108.XSHG': 1490, '002008.X
SHE': 827, '000046.XSHE': 2286, '002456.XSHE': 697, '600383.XSHG
': 1615}
getData:              NetProfitGrowRate  profitRatio
secID
000046.XSHE             2.3237     1.040085
002456.XSHE            14.5041     1.020039
600108.XSHG             1.1371     1.017244
002008.XSHE             1.1794     1.016968
601991.XSHG             1.0666     1.006950
buylist: ['000046.XSHE', '002456.XSHE', '600108.XSHG', '002008.X
SHE', '601991.XSHG']
日期： 2013-04-09 00:00:00 ,持仓： {'002450.XSHE': 837, '000895.XS
HE': 471, '600383.XSHG': 1615, '002456.XSHE': 697, '000413.XSHE'
: 3024}
getData:              NetProfitGrowRate  profitRatio
secID
000895.XSHE             1.0951     1.062802
000413.XSHE            10.4768     1.023791
601991.XSHG             1.0666     1.020423
002456.XSHE            14.5041     1.019788
000046.XSHE             2.3237     1.008401
buylist: ['000895.XSHE', '000413.XSHE', '601991.XSHG', '002456.X
SHE', '000046.XSHE']
日期： 2013-04-16 00:00:00 ,持仓： {'600108.XSHG': 1630, '002008.X
SHE': 821, '601991.XSHG': 2584, '000046.XSHE': 2391, '002456.XSH
E': 697}
getData:              NetProfitGrowRate  profitRatio
secID
002450.XSHE             2.2432     1.225398
002456.XSHE            14.5041     1.178291
600108.XSHG             2.9211     1.085501
000895.XSHE             1.0951     1.081940
000712.XSHE            21.2835     1.048451
buylist: ['002450.XSHE', '002456.XSHE', '600108.XSHG', '000895.X
SHE', '000712.XSHE']
日期： 2013-04-23 00:00:00 ,持仓： {'000895.XSHE': 458, '601991.XS
HG': 2584, '000046.XSHE': 2391, '002456.XSHE': 697, '000413.XSHE
': 2722}
getData:              NetProfitGrowRate  profitRatio
secID
300251.XSHE             1.0538     1.225936
601901.XSHG             1.6801     1.086260
000783.XSHE             1.8956     1.083919
000046.XSHE             2.3047     1.049812
000712.XSHE            21.2835     1.031657
buylist: ['300251.XSHE', '601901.XSHG', '000783.XSHE', '000046.X
SHE', '000712.XSHE']
日期： 2013-05-03 00:00:00 ,持仓： {'002450.XSHE': 636, '600108.XS
HG': 1598, '000895.XSHE': 458, '002456.XSHE': 697, '000712.XSHE'
: 1434}
getData:              NetProfitGrowRate  profitRatio
```

```
secID
300251.XSHE             1.0538     1.119771
002673.XSHE             1.4960     1.059101
600011.XSHG             2.9428     1.047790
000413.XSHE             3.1256     1.047541
600316.XSHG             1.3699     1.046952
buylist: ['300251.XSHE', '002673.XSHE', '600011.XSHG', '000413.X
SHE', '600316.XSHG']
日期： 2013-05-10 00:00:00 ,持仓： {'000712.XSHE': 1434, '300251.X
SHE': 985, '000046.XSHE': 2278, '601901.XSHG': 1461, '000783.XSH
E': 2336}
getData:              NetProfitGrowRate  profitRatio
secID
600316.XSHG             1.3699     1.152960
600108.XSHG             2.4674     1.081311
002450.XSHE             2.2106     1.076391
000712.XSHE            21.2835     1.073935
002456.XSHE             9.8722     1.072468
buylist: ['600316.XSHG', '600108.XSHG', '002450.XSHE', '000712.X
SHE', '002456.XSHE']
日期： 2013-05-17 00:00:00 ,持仓： {'002673.XSHE': 1604, '600316.X
SHG': 704, '300251.XSHE': 985, '600011.XSHG': 1945, '000413.XSHE
': 3086}
getData:              NetProfitGrowRate  profitRatio
secID
002450.XSHE             2.2106     1.099246
600583.XSHG             1.3907     1.075360
300251.XSHE             1.0538     1.060257
000413.XSHE             3.1256     1.053806
000625.XSHE             4.4493     1.043534
buylist: ['002450.XSHE', '600583.XSHG', '300251.XSHE', '000413.X
SHE', '000625.XSHE']
日期： 2013-05-24 00:00:00 ,持仓： {'002450.XSHE': 643, '600108.XS
HG': 1574, '600316.XSHG': 704, '002456.XSHE': 405, '000712.XSHE'
: 1378}
getData:              NetProfitGrowRate  profitRatio
secID
000712.XSHE            21.2835     1.166762
000046.XSHE             2.3047     1.123654
000413.XSHE             3.1256     1.075840
601901.XSHG             1.6801     1.056115
600316.XSHG             1.3699     1.052770
buylist: ['000712.XSHE', '000046.XSHE', '000413.XSHE', '601901.X
SHG', '600316.XSHG']
日期： 2013-05-31 00:00:00 ,持仓： {'002450.XSHE': 643, '300251.XS
HE': 923, '000625.XSHE': 1099, '600583.XSHG': 1550, '000413.XSHE
': 2882}
getData:              NetProfitGrowRate  profitRatio
secID
000046.XSHE             2.3047     1.039187
002456.XSHE             9.8722     1.019868
000413.XSHE             3.1256     1.014214
000712.XSHE            21.2835     1.011627
```

```
000539.XSHE             1.0140     1.005502
buylist: ['000046.XSHE', '002456.XSHE', '000413.XSHE', '000712.X
SHE', '000539.XSHE']
日期： 2013-06-07 00:00:00 ,持仓： {'000712.XSHE': 1158, '600316.X
SHG': 558, '000046.XSHE': 2108, '601901.XSHG': 1550, '000413.XSH
E': 2882}
getData:              NetProfitGrowRate  profitRatio
secID
300251.XSHE             1.0538     1.146265
000413.XSHE             3.1256     1.122355
601991.XSHG             1.0182     1.096403
002450.XSHE             2.2106     1.079312
000712.XSHE            21.2835     1.018607
buylist: ['300251.XSHE', '000413.XSHE', '601991.XSHG', '002450.X
SHE', '000712.XSHE']
日期： 2013-06-19 00:00:00 ,持仓： {'000712.XSHE': 1158, '000539.X
SHE': 2804, '000046.XSHE': 2108, '002456.XSHE': 481, '000413.XSH
E': 2882}
getData:              NetProfitGrowRate  profitRatio
secID
601991.XSHG             1.0182     0.993971
600011.XSHG             2.9428     0.961163
000625.XSHE             4.4493     0.949968
002450.XSHE             2.2106     0.938177
300251.XSHE             1.0538     0.930702
buylist: ['601991.XSHG', '600011.XSHG', '000625.XSHE', '002450.X
SHE', '300251.XSHE']
日期： 2013-06-26 00:00:00 ,持仓： {'002450.XSHE': 633, '000712.XS
HE': 1158, '601991.XSHG': 2463, '300251.XSHE': 827, '000413.XSHE
': 2882}
getData:              NetProfitGrowRate  profitRatio
secID
600108.XSHG             2.4674     1.115134
002456.XSHE             9.8722     1.111125
000712.XSHE            21.2835     1.094753
600886.XSHG             2.7866     1.082971
600011.XSHG             2.9428     1.066637
buylist: ['600108.XSHG', '002456.XSHE', '000712.XSHE', '600886.X
SHG', '600011.XSHG']
日期： 2013-07-03 00:00:00 ,持仓： {'002450.XSHE': 633, '000625.XS
HE': 1270, '601991.XSHG': 2463, '300251.XSHE': 827, '600011.XSHG
': 2487}
getData:              NetProfitGrowRate  profitRatio
secID
600011.XSHG             2.9428     1.045534
000625.XSHE             4.4493     1.035993
000046.XSHE             2.3047     1.032779
600795.XSHG             1.0106     0.995629
600886.XSHG             2.7866     0.994578
buylist: ['600011.XSHG', '000625.XSHE', '000046.XSHE', '600795.X
SHG', '600886.XSHG']
日期： 2013-07-10 00:00:00 ,持仓： {'600108.XSHG': 1624, '000712.X
SHE': 1215, '600011.XSHG': 2487, '002456.XSHE': 421, '600886.XSH
G': 3186}
getData:              NetProfitGrowRate  profitRatio
secID
600583.XSHG             1.3907     1.123528
601901.XSHG             1.6801     1.116944
002673.XSHE             1.4960     1.102686
600369.XSHG             1.5169     1.101620
002450.XSHE             2.2106     1.100006
buylist: ['600583.XSHG', '601901.XSHG', '002673.XSHE', '600369.X
SHG', '002450.XSHE']
日期： 2013-07-17 00:00:00 ,持仓： {'600886.XSHG': 3186, '000625.X
SHE': 1317, '600795.XSHG': 5623, '600011.XSHG': 2487, '000046.XS
HE': 2541}
getData:              NetProfitGrowRate  profitRatio
secID
000413.XSHE             3.1256     1.117993
002450.XSHE             2.2106     1.036790
000712.XSHE            21.2835     1.035480
600583.XSHG             1.3907     1.034635
600108.XSHG             2.4674     1.030835
buylist: ['000413.XSHE', '002450.XSHE', '000712.XSHE', '600583.X
SHG', '600108.XSHG']
日期： 2013-07-24 00:00:00 ,持仓： {'002450.XSHE': 601, '002673.XS
HE': 1755, '600369.XSHG': 2597, '600583.XSHG': 1511, '601901.XSH
G': 1775}
getData:              NetProfitGrowRate  profitRatio
secID
600011.XSHG             2.9428     1.040415
601991.XSHG             1.0182     1.030665
600369.XSHG             1.5169     1.003435
000413.XSHE             3.1256     1.001363
002673.XSHE             1.4960     0.988043
buylist: ['600011.XSHG', '601991.XSHG', '600369.XSHG', '000413.X
SHE', '002673.XSHE']
日期： 2013-07-31 00:00:00 ,持仓： {'002450.XSHE': 601, '600108.XS
HG': 1756, '600583.XSHG': 1511, '000712.XSHE': 1274, '000413.XSH
E': 2480}
getData:              NetProfitGrowRate  profitRatio
secID
000625.XSHE             4.4493     1.201249
000712.XSHE            21.2835     1.140688
600886.XSHG             2.7866     1.110534
600011.XSHG             3.1450     1.097059
600027.XSHG             2.9527     1.088622
buylist: ['000625.XSHE', '000712.XSHE', '600886.XSHG', '600011.X
SHG', '600027.XSHG']
日期： 2013-08-07 00:00:00 ,持仓： {'601991.XSHG': 2276, '002673.X
SHE': 1864, '600369.XSHG': 2787, '600011.XSHG': 2318, '000413.XS
HE': 2480}
getData:              NetProfitGrowRate  profitRatio
secID
600157.XSHG             2.3238     1.161850
600886.XSHG             2.7866     1.064370
```

```
601555.XSHG             1.3076     1.059994
600369.XSHG             1.5169     1.050995
600108.XSHG             2.4674     1.039207
buylist: ['600157.XSHG', '600886.XSHG', '601555.XSHG', '600369.X
SHG', '600108.XSHG']
日期： 2013-08-14 00:00:00 ,持仓： {'600027.XSHG': 3865, '000712.X
SHE': 1035, '000625.XSHE': 1120, '600011.XSHG': 2318, '600886.XS
HG': 3080}
getData:              NetProfitGrowRate  profitRatio
secID
000413.XSHE             3.1256     1.096319
600663.XSHG             1.4512     1.011244
000725.XSHE             1.5732     1.000000
002456.XSHE             9.8722     0.992576
600027.XSHG             2.9527     0.980969
buylist: ['000413.XSHE', '600663.XSHG', '000725.XSHE', '002456.X
SHE', '600027.XSHG']
日期： 2013-08-21 00:00:00 ,持仓： {'600108.XSHG': 1660, '600157.X
SHG': 4363, '600369.XSHG': 2671, '600886.XSHG': 3080, '601555.XS
HG': 1635}
getData:              NetProfitGrowRate  profitRatio
secID
600663.XSHG             1.4512     1.251894
300027.XSHE             1.2203     1.251205
300251.XSHE             1.0400     1.080901
600252.XSHG             1.1888     1.048242
000625.XSHE             4.4493     1.046591
buylist: ['600663.XSHG', '300027.XSHE', '300251.XSHE', '600252.X
SHG', '000625.XSHE']
日期： 2013-08-28 00:00:00 ,持仓： {'600027.XSHG': 4114, '000725.X
SHE': 5254, '600663.XSHG': 1021, '002456.XSHE': 501, '000413.XSH
E': 1901}
getData:              NetProfitGrowRate  profitRatio
secID
600648.XSHG             1.3325     1.331161
600108.XSHG             2.1869     1.234514
600663.XSHG             1.4512     1.162914
300027.XSHE             1.2203     1.082302
000625.XSHE             3.9408     1.055561
buylist: ['600648.XSHG', '600108.XSHG', '600663.XSHG', '300027.X
SHE', '000625.XSHE']
日期： 2013-09-04 00:00:00 ,持仓： {'000625.XSHE': 1139, '300251.X
SHE': 747, '600663.XSHG': 1021, '600252.XSHG': 2250, '300027.XSH
E': 467}
getData:              NetProfitGrowRate  profitRatio
secID
600648.XSHG             1.3325     1.464392
300027.XSHE             1.2203     1.132654
000503.XSHE             2.4690     1.097778
600663.XSHG             1.4512     1.078274
601555.XSHG             1.3076     1.073362
buylist: ['600648.XSHG', '300027.XSHE', '000503.XSHE', '600663.X
SHG', '601555.XSHG']
```

```
日期： 2013-09-11 00:00:00 ,持仓： {'600108.XSHG': 1324, '000625.X
SHE': 1139, '600663.XSHG': 1021, '300027.XSHE': 467}
getData:              NetProfitGrowRate  profitRatio
secID
600648.XSHG             1.3325     1.395300
300251.XSHE             1.0400     1.215395
300027.XSHE             1.2203     1.177448
000413.XSHE             1.5996     1.096738
000503.XSHE             2.4690     1.028571
buylist: ['600648.XSHG', '300251.XSHE', '300027.XSHE', '000413.X
SHE', '000503.XSHE']
日期： 2013-09-18 00:00:00 ,持仓： {'000503.XSHE': 690, '300027.XS
HE': 467, '600663.XSHG': 1021, '601555.XSHG': 1685}
getData:              NetProfitGrowRate  profitRatio
secID
600663.XSHG             1.4512     1.309473
600648.XSHG             1.3325     1.197671
300251.XSHE             1.0400     1.085617
600108.XSHG             2.1869     1.072006
000712.XSHE             1.9401     1.006185
buylist: ['600663.XSHG', '600648.XSHG', '300251.XSHE', '600108.X
SHG', '000712.XSHE']
日期： 2013-09-27 00:00:00 ,持仓： {'600648.XSHG': 294, '000503.XS
HE': 690, '300027.XSHE': 467, '300251.XSHE': 698, '000413.XSHE':
 2201}
getData:              NetProfitGrowRate  profitRatio
secID
600108.XSHG             2.1869     1.243266
000503.XSHE             2.4690     1.068320
600583.XSHG             1.1630     1.048205
000625.XSHE             3.9408     1.047890
600674.XSHG             1.8199     1.046938
buylist: ['600108.XSHG', '000503.XSHE', '600583.XSHG', '000625.X
SHE', '600674.XSHG']
日期： 2013-10-11 00:00:00 ,持仓： {'600108.XSHG': 1302, '600648.X
SHG': 294, '300251.XSHE': 698, '600663.XSHG': 439, '000712.XSHE'
: 1111}
getData:              NetProfitGrowRate  profitRatio
secID
002456.XSHE             2.6541     1.189799
000503.XSHE             2.4690     1.133425
000712.XSHE             1.9401     1.042925
600027.XSHG             7.8319     1.029016
600011.XSHG             3.1450     1.029006
buylist: ['002456.XSHE', '000503.XSHE', '000712.XSHE', '600027.X
SHG', '600011.XSHG']
日期： 2013-10-18 00:00:00 ,持仓： {'600108.XSHG': 1302, '000503.X
SHE': 610, '000625.XSHE': 1227, '600583.XSHG': 1738, '600674.XSH
G': 2257}
getData:              NetProfitGrowRate  profitRatio
secID
000413.XSHE             1.5996     1.045447
600252.XSHG             1.3699     1.039856
```

```
002450.XSHE             1.1200     1.035005
600648.XSHG             1.3325     1.027141
600583.XSHG             1.1630     1.021851
buylist: ['000413.XSHE', '600252.XSHG', '002450.XSHE', '600648.X
SHG', '600583.XSHG']
日期： 2013-10-25 00:00:00 ,持仓： {'600027.XSHG': 4383, '000503.X
SHE': 610, '600011.XSHG': 2477, '002456.XSHE': 554, '000712.XSHE
': 1206}
getData:              NetProfitGrowRate  profitRatio
secID
600886.XSHG             2.6808     1.124354
300017.XSHE             1.5261     1.084772
600011.XSHG             1.6585     1.068460
600583.XSHG             1.3300     1.058399
600027.XSHG             5.8031     1.051499
buylist: ['600886.XSHG', '300017.XSHE', '600011.XSHG', '600583.X
SHG', '600027.XSHG']
日期： 2013-11-01 00:00:00 ,持仓： {'002450.XSHE': 781, '600648.XS
HG': 281, '600583.XSHG': 1667, '600252.XSHG': 2785, '000413.XSHE
': 2018}
getData:              NetProfitGrowRate  profitRatio
secID
600583.XSHG             1.3300     1.132449
002202.XSHE             4.3413     1.046205
000917.XSHE             1.5781     1.006599
600027.XSHG             5.8031     1.006038
000712.XSHE             1.7032     1.001928
buylist: ['600583.XSHG', '002202.XSHE', '000917.XSHE', '600027.X
SHG', '000712.XSHE']
日期： 2013-11-08 00:00:00 ,持仓： {'600027.XSHG': 4345, '600886.X
SHG': 3134, '300017.XSHE': 795, '600583.XSHG': 1667, '600011.XSH
G': 2470}
getData:              NetProfitGrowRate  profitRatio
secID
600352.XSHG             1.4342     1.051812
000917.XSHE             1.5781     1.040933
600252.XSHG             1.3699     1.038871
300017.XSHE             1.5261     1.022195
002202.XSHE             4.3413     1.020151
buylist: ['600352.XSHG', '000917.XSHE', '600252.XSHG', '300017.X
SHE', '002202.XSHE']
日期： 2013-11-15 00:00:00 ,持仓： {'600027.XSHG': 4345, '002202.X
SHE': 1490, '000917.XSHE': 782, '600583.XSHG': 1667, '000712.XSH
E': 1136}
getData:              NetProfitGrowRate  profitRatio
secID
600648.XSHG             1.8864     1.102044
601628.XSHG             1.9948     1.096096
600583.XSHG             1.3300     1.069177
600663.XSHG             1.6233     1.064922
600369.XSHG             1.4694     1.048993
buylist: ['600648.XSHG', '601628.XSHG', '600583.XSHG', '600663.X
SHG', '600369.XSHG']
```

```
日期： 2013-11-22 00:00:00 ,持仓： {'000917.XSHE': 782, '002202.XS
HE': 1490, '300017.XSHE': 762, '600252.XSHG': 2921, '600352.XSHG
': 2187}
getData:              NetProfitGrowRate  profitRatio
secID
600352.XSHG             1.4342     1.119820
601099.XSHG             2.3074     1.114664
300017.XSHE             1.5261     1.111971
000625.XSHE             2.8365     1.037890
600369.XSHG             1.4694     1.035814
buylist: ['600352.XSHG', '601099.XSHG', '300017.XSHE', '000625.X
SHE', '600369.XSHG']
日期： 2013-11-29 00:00:00 ,持仓： {'600648.XSHG': 314, '601628.XS
HG': 828, '600369.XSHG': 2632, '600583.XSHG': 1416, '600663.XSHG
': 656}
getData:              NetProfitGrowRate  profitRatio
secID
000712.XSHE             1.7032     1.151983
600369.XSHG             1.4694     1.084413
000625.XSHE             2.8365     1.078332
601628.XSHG             1.9948     1.041333
002202.XSHE             4.3413     1.039846
buylist: ['000712.XSHE', '600369.XSHG', '000625.XSHE', '601628.X
SHG', '002202.XSHE']
日期： 2013-12-06 00:00:00 ,持仓： {'601099.XSHG': 3117, '600352.X
SHG': 1859, '300017.XSHE': 705, '600369.XSHG': 2632, '000625.XSH
E': 1045}
getData:              NetProfitGrowRate  profitRatio
secID
300017.XSHE             1.5261     1.105913
601099.XSHG             2.3074     1.019100
600663.XSHG             1.6233     1.004301
600352.XSHG             1.4342     1.001420
600674.XSHG             1.8361     0.997585
buylist: ['300017.XSHE', '601099.XSHG', '600663.XSHG', '600352.X
SHG', '600674.XSHG']
日期： 2013-12-13 00:00:00 ,持仓： {'002202.XSHE': 1431, '601628.X
SHG': 754, '600369.XSHG': 2632, '000625.XSHE': 1045, '000712.XSH
E': 1013}
getData:              NetProfitGrowRate  profitRatio
secID
300017.XSHE             1.5261     1.108336
000009.XSHE             1.6400     1.017611
000686.XSHE            43.8777     0.989797
600674.XSHG             1.8361     0.982998
600648.XSHG             1.8864     0.974786
buylist: ['300017.XSHE', '000009.XSHE', '000686.XSHE', '600674.X
SHG', '600648.XSHG']
日期： 2013-12-20 00:00:00 ,持仓： {'601099.XSHG': 3012, '600352.X
SHG': 1849, '300017.XSHE': 689, '600663.XSHG': 636, '600674.XSHG
': 2028}
getData:              NetProfitGrowRate  profitRatio
secID
```

```
000046.XSHE             1.8622     1.062081
000917.XSHE             1.5781     1.052427
600352.XSHG             1.4342     1.022568
600252.XSHG             1.3699     1.022152
601628.XSHG             1.9948     1.020006
buylist: ['000046.XSHE', '000917.XSHE', '600352.XSHG', '600252.X
SHG', '601628.XSHG']
日期： 2013-12-27 00:00:00 ,持仓： {'000009.XSHE': 1550, '600648.X
SHG': 350, '000686.XSHE': 1535, '300017.XSHE': 689, '600674.XSHG
': 2028}
getData:              NetProfitGrowRate  profitRatio
secID
300017.XSHE             1.5261     1.064353
002202.XSHE             4.3413     1.056975
600252.XSHG             1.3699     1.055956
601099.XSHG             2.3074     1.038871
000686.XSHE            43.8777     1.018868
buylist: ['300017.XSHE', '002202.XSHE', '600252.XSHG', '601099.X
SHG', '000686.XSHE']
日期： 2014-01-06 00:00:00 ,持仓： {'000917.XSHE': 715, '600352.XS
HG': 1872, '601628.XSHG': 799, '000046.XSHE': 2584, '600252.XSHG
': 2727}
getData:              NetProfitGrowRate  profitRatio
secID
600583.XSHG             1.3300     1.132102
600648.XSHG             1.8864     1.067373
600663.XSHG             1.6233     1.048503
600886.XSHG             2.6808     1.033741
600011.XSHG             1.6585     1.018681
buylist: ['600583.XSHG', '600648.XSHG', '600663.XSHG', '600886.X
SHG', '600011.XSHG']
日期： 2014-01-13 00:00:00 ,持仓： {'601099.XSHG': 2949, '002202.X
SHE': 1390, '000686.XSHE': 1479, '300017.XSHE': 568, '600252.XSH
G': 2727}
getData:              NetProfitGrowRate  profitRatio
secID
601099.XSHG             2.3074     1.126824
002202.XSHE             4.3413     1.113215
000712.XSHE             1.7032     1.053295
600352.XSHG             1.4342     1.031516
601628.XSHG             1.9948     1.030327
buylist: ['601099.XSHG', '002202.XSHE', '000712.XSHE', '600352.X
SHG', '601628.XSHG']
日期： 2014-01-20 00:00:00 ,持仓： {'600886.XSHG': 2992, '600648.X
SHG': 359, '600583.XSHG': 1437, '600663.XSHG': 705, '600011.XSHG
': 2573}
getData:              NetProfitGrowRate  profitRatio
secID
300017.XSHE             1.5261     1.209802
601099.XSHG             2.3074     1.148589
002202.XSHE             4.3413     1.141289
000712.XSHE             1.7032     1.115776
600648.XSHG             1.8864     1.096859
```

```
buylist: ['300017.XSHE', '601099.XSHG', '002202.XSHE', '000712.X
SHE', '600648.XSHG']
日期： 2014-01-27 00:00:00 ,持仓： {'601099.XSHG': 2640, '002202.X
SHE': 1306, '601628.XSHG': 793, '000712.XSHE': 1016, '600352.XSH
G': 1968}
getData:              NetProfitGrowRate  profitRatio
secID
300017.XSHE             1.5261     1.170863
000712.XSHE             1.7032     1.146359
601099.XSHG             2.3074     1.086993
000917.XSHE             1.5781     1.066609
000686.XSHE            43.8777     1.049759
buylist: ['300017.XSHE', '000712.XSHE', '601099.XSHG', '000917.X
SHE', '000686.XSHE']
日期： 2014-02-10 00:00:00 ,持仓： {'601099.XSHG': 2640, '600648.X
SHG': 332, '300017.XSHE': 450, '000712.XSHE': 1016, '002202.XSHE
': 1306}
getData:              NetProfitGrowRate  profitRatio
secID
600352.XSHG             1.4342     1.073845
000009.XSHE             1.6400     1.041815
600663.XSHG             1.6233     1.039189
000046.XSHE             1.8622     1.031897
600252.XSHG             1.3699     1.023154
buylist: ['600352.XSHG', '000009.XSHE', '600663.XSHG', '000046.X
SHE', '600252.XSHG']
日期： 2014-02-17 00:00:00 ,持仓： {'601099.XSHG': 2640, '000917.X
SHE': 702, '000686.XSHE': 1389, '300017.XSHE': 450, '000712.XSHE
': 1016}
getData:              NetProfitGrowRate  profitRatio
secID
600674.XSHG             1.8361     1.024902
300017.XSHE             1.5261     1.012560
000046.XSHE             1.8622     1.006401
000009.XSHE             1.6400     1.000000
600886.XSHG             2.6808     1.000000
buylist: ['600674.XSHG', '300017.XSHE', '000046.XSHE', '000009.X
SHE', '600886.XSHG']
日期： 2014-02-24 00:00:00 ,持仓： {'000009.XSHE': 1264, '600352.X
SHG': 1792, '000046.XSHE': 2652, '600252.XSHG': 2601, '600663.XS
HG': 632}
getData:              NetProfitGrowRate  profitRatio
secID
000009.XSHE             1.6400     1.057629
600369.XSHG             1.4694     1.039398
000712.XSHE             1.7032     1.037474
600027.XSHG             5.8031     1.027129
000686.XSHE            43.8777     1.010130
buylist: ['000009.XSHE', '600369.XSHG', '000712.XSHE', '600027.X
SHG', '000686.XSHE']
日期： 2014-03-03 00:00:00 ,持仓： {'000009.XSHE': 1264, '300017.X
SHE': 410, '000046.XSHE': 2652, '600674.XSHG': 2260, '600886.XSH
G': 3073}
```

```
getData:              NetProfitGrowRate  profitRatio
secID
601099.XSHG             2.3074     1.114637
600352.XSHG             1.4342     1.074520
300017.XSHE             1.5261     1.069434
000712.XSHE             1.7032     1.047295
002202.XSHE             4.3413     1.024708
buylist: ['601099.XSHG', '600352.XSHG', '300017.XSHE', '000712.X
SHE', '002202.XSHE']
日期： 2014-03-10 00:00:00 ,持仓： {'600027.XSHG': 4278, '000009.X
SHE': 1264, '600369.XSHG': 2679, '000686.XSHE': 1580, '000712.XS
HE': 761}
getData:              NetProfitGrowRate  profitRatio
secID
000686.XSHE             2.1949     1.110835
000009.XSHE             1.6400     1.049752
600886.XSHG             2.6808     1.045168
600583.XSHG             1.3300     1.032905
600674.XSHG             1.8361     1.032297
buylist: ['000686.XSHE', '000009.XSHE', '600886.XSHG', '600583.X
SHG', '600674.XSHG']
日期： 2014-03-17 00:00:00 ,持仓： {'601099.XSHG': 2167, '002202.X
SHE': 1110, '300017.XSHE': 424, '000712.XSHE': 761, '600352.XSHG
': 1650}
getData:              NetProfitGrowRate  profitRatio
secID
002202.XSHE             4.3413     1.042749
000686.XSHE             2.1949     1.035724
600027.XSHG             5.8031     1.034172
300017.XSHE             1.2855     1.021615
601601.XSHG             1.2722     1.016992
buylist: ['002202.XSHE', '000686.XSHE', '600027.XSHG', '300017.X
SHE', '601601.XSHG']
日期： 2014-03-24 00:00:00 ,持仓： {'000009.XSHE': 1248, '000686.X
SHE': 1492, '600583.XSHG': 1487, '600674.XSHG': 2191, '600886.XS
HG': 2851}
getData:              NetProfitGrowRate  profitRatio
secID
000625.XSHE             2.8365     1.066057
600886.XSHG             2.6808     1.047739
600352.XSHG             1.4342     1.046220
600674.XSHG             1.8361     1.017640
600867.XSHG             2.0088     1.000000
buylist: ['000625.XSHE', '600886.XSHG', '600352.XSHG', '600674.X
SHG', '600867.XSHG']
日期： 2014-03-31 00:00:00 ,持仓： {'600027.XSHG': 3955, '002202.X
SHE': 1094, '000686.XSHE': 1492, '300017.XSHE': 369, '601601.XSH
G': 730}
getData:              NetProfitGrowRate  profitRatio
secID
000625.XSHE             2.8365     1.088728
600352.XSHG             1.4342     1.059077
000750.XSHE             1.0884     1.043022
```

```
600648.XSHG             1.8864     1.032962
002594.XSHE             2.6444     1.032853
buylist: ['000625.XSHE', '600352.XSHG', '000750.XSHE', '600648.X
SHG', '002594.XSHE']
日期： 2014-04-08 00:00:00 ,持仓： {'600352.XSHG': 1466, '000625.X
SHE': 1192, '600867.XSHG': 1049, '600886.XSHG': 2667, '600674.XS
HG': 2073}
getData:              NetProfitGrowRate  profitRatio
secID
000625.XSHE             2.8365     1.104711
000009.XSHE             1.6400     1.039688
601628.XSHG             1.2186     1.028232
600839.XSHG             1.7718     1.021942
600648.XSHG             1.8864     1.020342
buylist: ['000625.XSHE', '000009.XSHE', '601628.XSHG', '600839.X
SHG', '600648.XSHG']
日期： 2014-04-15 00:00:00 ,持仓： {'600648.XSHG': 379, '000750.XS
HE': 1054, '000625.XSHE': 1192, '002594.XSHE': 213, '600352.XSHG
': 1466}
getData:              NetProfitGrowRate  profitRatio
secID
300017.XSHE             1.2855     1.055075
000917.XSHE             1.5781     1.044945
601179.XSHG             2.3206     1.013215
002594.XSHE             2.6444     1.010078
600648.XSHG             1.8864     1.006202
buylist: ['300017.XSHE', '000917.XSHE', '601179.XSHG', '002594.X
SHE', '600648.XSHG']
日期： 2014-04-22 00:00:00 ,持仓： {'000009.XSHE': 1191, '600648.X
SHG': 379, '601628.XSHG': 788, '600839.XSHG': 2982, '000625.XSHE
': 1192}
getData:              NetProfitGrowRate  profitRatio
secID
002202.XSHE             1.3557     1.018616
300017.XSHE             1.6739     1.004767
600369.XSHG             1.4618     0.998828
600027.XSHG             1.9501     0.987302
000728.XSHE             1.4491     0.986435
buylist: ['002202.XSHE', '300017.XSHE', '600369.XSHG', '600027.X
SHG', '000728.XSHE']
日期： 2014-04-29 00:00:00 ,持仓： {'000917.XSHE': 776, '600648.XS
HG': 379, '300017.XSHE': 453, '601179.XSHG': 2964, '002594.XSHE'
: 227}
getData:              NetProfitGrowRate  profitRatio
secID
300017.XSHE             1.6739     1.077084
300058.XSHE             1.0727     1.072137
600633.XSHG             1.1869     1.033699
600886.XSHG             1.2837     1.030234
600887.XSHG             1.1054     1.027954
buylist: ['300017.XSHE', '300058.XSHE', '600633.XSHG', '600886.X
SHG', '600887.XSHG']
日期： 2014-05-08 00:00:00 ,持仓： {'600027.XSHG': 4050, '002202.X
SHE': 1328, '300017.XSHE': 453, '600369.XSHG': 2660, '000728.XSH
E': 1218}
getData:              NetProfitGrowRate  profitRatio
secID
000728.XSHE             1.4491     1.083973
600867.XSHG             1.3611     1.066896
600352.XSHG             1.1168     1.051762
600633.XSHG             1.1869     1.046675
600111.XSHG             1.5238     1.033323
buylist: ['000728.XSHE', '600867.XSHG', '600352.XSHG', '600633.X
SHG', '600111.XSHG']
日期： 2014-05-15 00:00:00 ,持仓： {'300017.XSHE': 453, '600633.XS
HG': 862, '600886.XSHG': 2311, '600887.XSHG': 898, '300058.XSHE'
: 781}
getData:              NetProfitGrowRate  profitRatio
secID
600886.XSHG             1.2837     1.056149
600649.XSHG             1.2412     1.037091
600027.XSHG             1.2327     1.025260
600674.XSHG             1.3748     1.024103
000725.XSHE             1.5186     1.014085
buylist: ['600886.XSHG', '600649.XSHG', '600027.XSHG', '600674.X
SHG', '000725.XSHE']
日期： 2014-05-22 00:00:00 ,持仓： {'600352.XSHG': 1407, '600111.X
SHG': 873, '600633.XSHG': 862, '600867.XSHG': 1113, '000728.XSHE
': 1089}
getData:              NetProfitGrowRate  profitRatio
secID
002594.XSHE             1.2435     1.133788
300017.XSHE             1.6739     1.121024
600867.XSHG             1.3611     1.091230
002202.XSHE             1.3557     1.079545
600633.XSHG             1.1869     1.068456
buylist: ['002594.XSHE', '300017.XSHE', '600867.XSHG', '002202.X
SHE', '600633.XSHG']
日期： 2014-05-29 00:00:00 ,持仓： {'600027.XSHG': 3653, '600649.X
SHG': 1697, '000725.XSHE': 5011, '600886.XSHG': 2329, '600674.XS
HG': 1915}
getData:              NetProfitGrowRate  profitRatio
secID
300027.XSHE             1.3626     1.133186
600886.XSHG             1.2837     1.034307
000712.XSHE             3.9370     1.029255
600839.XSHG             1.1077     1.026230
600867.XSHG             1.3611     1.024717
buylist: ['300027.XSHE', '600886.XSHG', '000712.XSHE', '600839.X
SHG', '600867.XSHG']
日期： 2014-06-06 00:00:00 ,持仓： {'300017.XSHE': 412, '002202.XS
HE': 1200, '600633.XSHG': 822, '600867.XSHG': 993, '002594.XSHE'
: 231}
getData:              NetProfitGrowRate  profitRatio
secID
600633.XSHG             1.1869     1.056702
```

```
000712.XSHE             3.9370     1.052317
300027.XSHE             1.3626     1.046564
600886.XSHG             1.2837     1.035149
002202.XSHE             1.3557     1.022519
buylist: ['600633.XSHG', '000712.XSHE', '300027.XSHE', '600886.X
SHG', '002202.XSHE']
日期： 2014-06-13 00:00:00 ,持仓： {'000712.XSHE': 819, '600839.XS
HG': 3482, '600867.XSHG': 993, '600886.XSHG': 2231, '300027.XSHE
': 452}
getData:              NetProfitGrowRate  profitRatio
secID
000725.XSHE             1.5186     1.023041
600867.XSHG             1.3611     1.004021
600583.XSHG             1.6443     0.995563
000625.XSHE             1.6681     0.989348
600111.XSHG             1.5238     0.984946
buylist: ['000725.XSHE', '600867.XSHG', '600583.XSHG', '000625.X
SHE', '600111.XSHG']
日期： 2014-06-20 00:00:00 ,持仓： {'002202.XSHE': 1156, '600633.X
SHG': 752, '600886.XSHG': 2231, '000712.XSHE': 819, '300027.XSHE
': 452}
getData:              NetProfitGrowRate  profitRatio
secID
300017.XSHE             1.6739     1.094995
002594.XSHE             1.2435     1.062227
600886.XSHG             1.2837     1.034318
600867.XSHG             1.3611     1.027216
600111.XSHG             1.5238     1.022082
buylist: ['300017.XSHE', '002594.XSHE', '600886.XSHG', '600867.X
SHG', '600111.XSHG']
日期： 2014-06-27 00:00:00 ,持仓： {'000625.XSHE': 907, '600111.XS
HG': 858, '000725.XSHE': 4859, '600867.XSHG': 960, '600583.XSHG'
: 1550}
```

7.3 低价策略

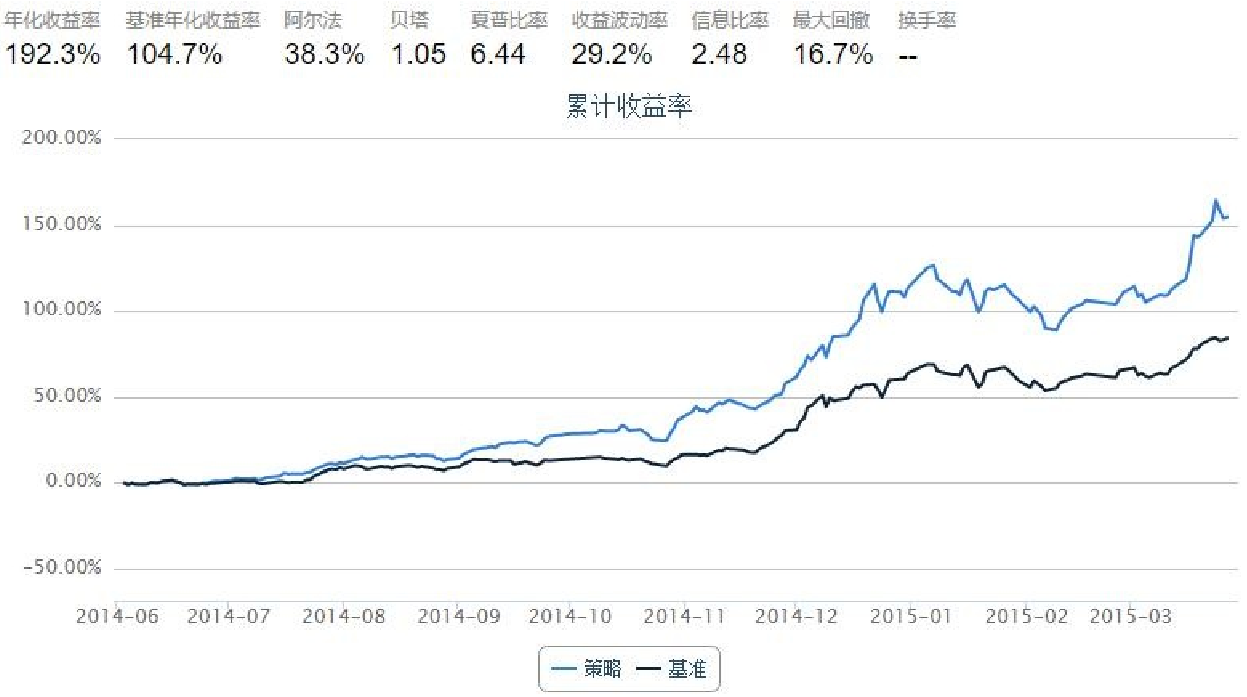
专捡便宜货(新版quartz)

> 来源：https://uqer.io/community/share/55152e9ff9f06c8f33904450

策略原理

策略基本思路是：买入低于X元的股票，持有到1.25X元以上则卖出.

```
start = datetime(2014, 6, 1)
end   = datetime(2015, 3, 27)
benchmark = 'HS300'                                 # 策略参考标准
universe = set_universe('HS300')
capital_base = 1000000


def initialize(account):
    account.buy_price_flag = 4
    account.sell_price_flag = account.buy_price_flag*1.25

def handle_data(account):
    signals = []
    acc_cash = account.cash

    for stock in account.universe:

        p = account.referencePrice[stock]
        if p < account.buy_price_flag:
            #满足买入条件，加入signals列表中
            signals.append(stock)
        elif p >= account.sell_price_flag and account.secpos.get(stock, 0) > 0:
            #将卖出股票所得现金加入到本次的可用现金
            acc_cash += account.referencePrice[stock]*account.secpos.get(stock,0)
            order_to(stock, 0)

    for stock in signals:
        # 平均买入signals列表中的股票
        amount = acc_cash/len(signals)/account.referencePrice[stock]
        order(stock, amount)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
192.3%	104.7%	38.3%	1.05	6.44	29.2%	2.48	16.7%	--

便宜就是 alpha

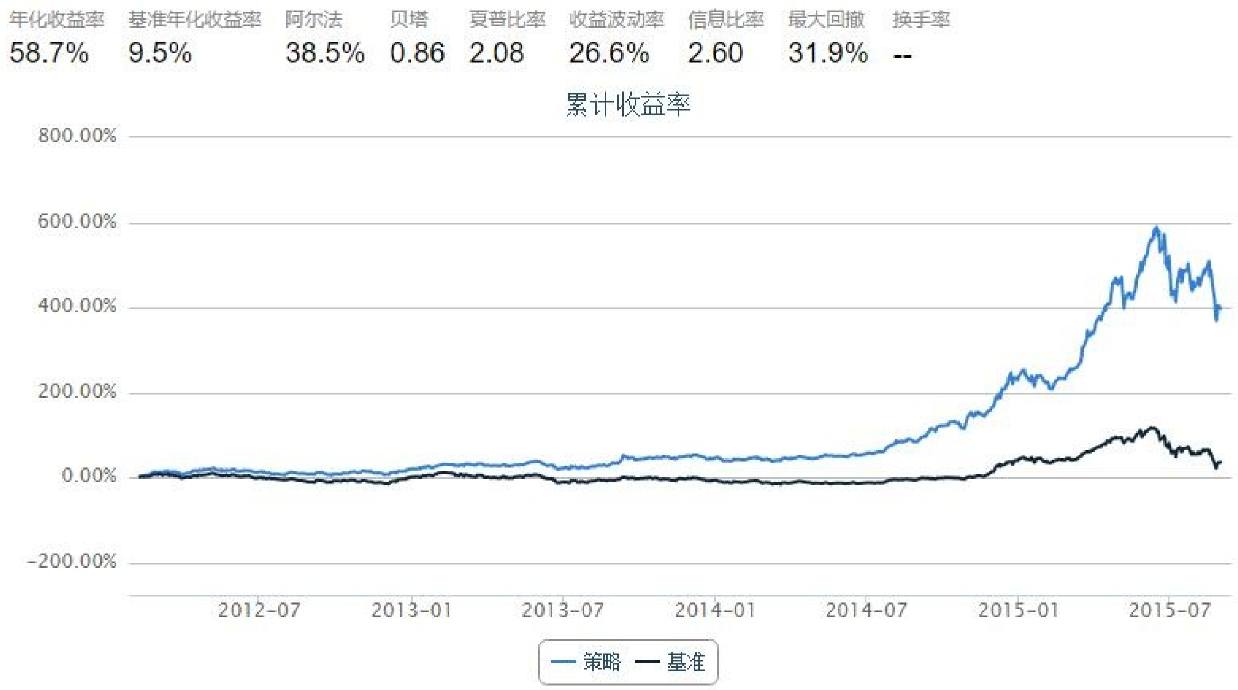
> 来源：https://uqer.io/community/share/5609169cf9f06c597565ef13

参考社区的策略，练手一个简单的调仓逻辑

策略思路

- 一星期或者一个月一换仓
- 每次换仓时，取股价最低的15只等权重买入

```
import numpy as np
from heapq import nlargest, nsmallest
from CAL.PyCAL import *

start = datetime(2012, 1, 1)
end   = datetime(2015, 8, 31)
benchmark = 'HS300'                              # 策略参考标准
universe = set_universe('ZZ500') + set_universe('HS300')
capital_base = 1000000
stk_num = 15       # 持仓股票数量
refresh_rate = 5

def initialize(account):
    pass

def handle_data(account):
    cal = Calendar('China.SSE')

    # ----------------- 清洗universe -------------------------------
    date = account.current_date
    yesterday = cal.advanceDate(date, '-1B', BizDayConvention.Following)
    yesterday = datetime(yesterday.year(), yesterday.month(), yesterday.dayOfMonth()).strftime('%Y%m%d'),
    # 去除ST股
    try:
        STlist = DataAPI.SecSTGet(secID=account.universe, beginDate=yesterday, endDate=yesterday, field=['secID']).tolist()
        account.universe = [s for s in account.universe if s not in STlist]
    except:
        pass
    # 去除流动性差的股票
    tv = account.get_attribute_history('turnoverValue', 20)
    mtv = {sec: sum(tvs)/20. for sec,tvs in tv.items()}
    account.universe = [s for s in account.universe if mtv.get(s
, 0) >= 10**7]
    # 去除新上市或复牌的股票
    opn = account.get_attribute_history('openPrice', 1)
    account.universe = [s for s in account.universe if not (np.i
snan(opn.get(s, 0)[0]) or opn.get(s, 0)[0] == 0)]

    # ----------------- 调仓逻辑 -------------------------------
    bucket = {}
    for stk in account.universe:
        bucket[stk] = account.referencePrice[stk]

    # 以前面计算得到的turnover_delta对股票池中股票排序，并取前stk_num只
，力图满足比赛要求
    # 注意：
    # 这里我们其实取了股价最低的 stk_num*2 只，原因在于：为了满足参赛要求
，调仓时候我们必须
    # 达到一定仓位，如果取stk_num只，那么一旦遇到涨停停牌等买不进的情况，
就跪了；所以我们拿
    # stk_num*2 数量的股票，但是却将仓位分成stk_num份，每份买进一只，这
样有一只买不进，就
    # 买后面的，参赛调仓是不是保险了许多啊
    buy_list = nsmallest(stk_num*2, bucket, key=bucket.get)

    # 目前持仓中不在buy_list中的股票，清仓
    for stk, amount in account.valid_secpos.items():
        if stk not in buy_list:
            order_to(stk, 0)

    # buy_list中的股票，等权重买入
    position_per_stk = account.referencePortfolioValue/stk_num
# 将仓位分成stk_num份
    for stk in buy_list:
        if account.referencePrice[stk] > 0:
            amount = int(position_per_stk/account.referencePrice
[stk]/100.0) * 100
            order_to(stk, amount)
    return
```

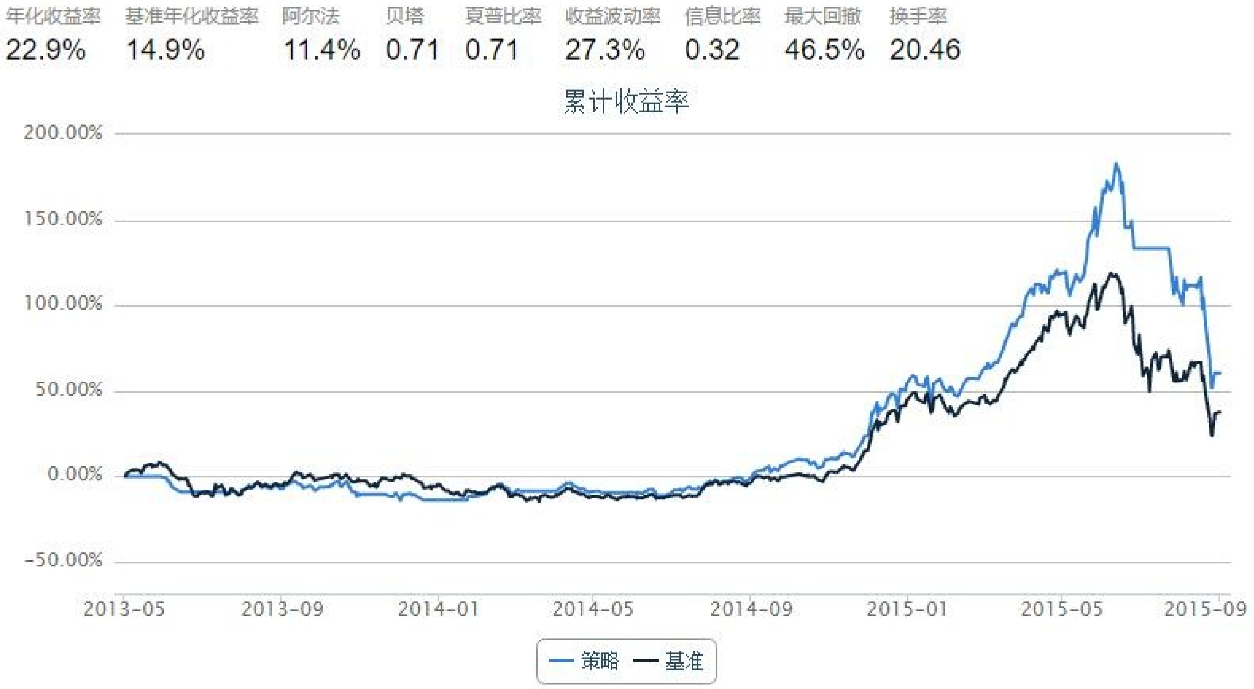


年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
58.7%	9.5%	38.5%	0.86	2.08	26.6%	2.60	31.9%	--

八 轮动模型

8.1 大小盘轮动 · 新手上路 -- 二八ETF择时轮动策略2.0

```
start = '2013-05-01'                       # 回测起始时间
end = '2015-09-01'                         # 回测结束时间
benchmark = 'HS300'                        # 策略参考标准
universe = ['510300.XSHG','510500.XSHG']        # 证券池，支持股票和基金
HS300,SZ500=universe
capital_base = 100000                      # 起始资金
freq = 'd'                             # 策略类型，'d'表示日间策略使用日线回测
refresh_rate =1                          # 调仓频率，表示执行handle_data的时间间隔，由于freq = 'd'，时间间隔的单位为交易日
max_retracement = 0.01                     # 最大回撤比例

def initialize(account):                   # 初始化虚拟账户状态
    pass

def handle_data(account):                  # 每个交易日的买入卖出指令


    # 周末进行交换
    if  account.current_date.weekday() != 4 :
        return

    # 有停牌的话，今天就跳过。
    if len(account.universe) < 2: return

    hist = account.get_attribute_history('closePrice', 19)
    if  len(hist) < 2:
        return


    # 如果HS300四周内涨幅大于SZ500
    if hist[HS300][-1]/hist[HS300][0] > hist[SZ500][-1]/hist[ SZ500][0]:
      # 且为正收益
      if hist[HS300][-1]/hist[HS300][0] > 1:
        if account.avail_secpos.has_key(SZ500):
            order_pct_to(SZ500, 0)
        order_pct_to(HS300, 0.99)

      elif hist[HS300][-1]/hist[HS300][0] < 1- max_retracement:
        # 负收益，清盘
        if account.avail_secpos.has_key(SZ500):
            order_pct_to(SZ500, 0)
        if account.avail_secpos.has_key(HS300):
            order_pct_to(HS300, 0)



    # 如果HS300四周内涨幅小于SZ500
    elif hist[HS300][-1]/hist[HS300][0] < hist[SZ500][-1]/hist[SZ500][0]:
     # 且为正收益
     if hist[SZ500][-1]/hist[SZ500][0] > 1:
        if account.avail_secpos.has_key(HS300):
            order_pct_to(HS300, 0)
        order_pct_to(SZ500, 0.99)

     elif hist[SZ500][-1]/hist[SZ500][0] < 1- max_retracement:
        # 负收益，清盘
        if account.avail_secpos.has_key(SZ500):
            order_pct_to(SZ500, 0)
        if account.avail_secpos.has_key(HS300):
            order_pct_to(HS300, 0)
```



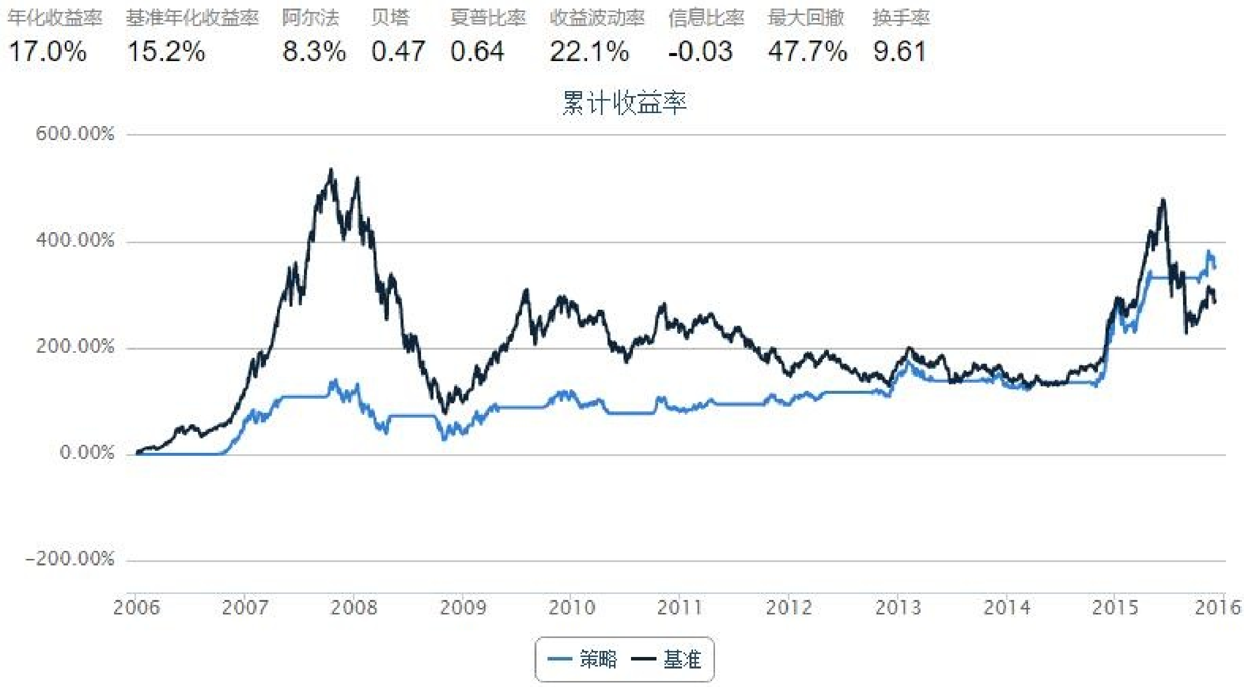
年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
22.9%	14.9%	11.4%	0.71	0.71	27.3%	0.32	46.5%	20.46

8.2 季节性策略

Halloween Cycle

> 来源：https://uqer.io/community/share/5663db06f9f06c6c8a91b37c

```
import numpy as np
start = '2006-01-01'                       # 回测起始时间
end = '2015-12-01'                         # 回测结束时间
benchmark = 'HS300'                        # 策略参考标准
universe = ['601398.XSHG','600028.XSHG', '601988.XSHG', '600036.XSHG','600030.XSHG','601318.XSHG', '600000.XSHG', '600019.XSHG', '600519.XSHG', '601166.XSHG']
capital_base = 100000                      # 起始资金
freq = 'd'                                 # 策略类型，'d'表示日间策略使用日线回测，'m'表示日内策略使用分钟线回测
refresh_rate = 1                           # 调仓频率，表示执行handle_data的时间间隔，若freq = 'd'时间间隔的单位为交易日，若freq = 'm'时间间隔为分钟

def initialize(account):                   # 初始化虚拟账户状态
    pass

def handle_data(account):                  # 每个交易日的买入卖出指令

    today = account.current_date
    for stock in account.universe:
        if(today.month == 10):
            p = account.referencePrice.get(stock, 0)
            if np.isnan(p) or p == 0:
                continue
            order_pct_to(stock, 0.1)
        elif today.month == 5 and stock in account.valid_secpos:
            order_to(stock,0)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
17.0%	15.2%	8.3%	0.47	0.64	22.1%	-0.03	47.7%	9.61

Halloween cycle 2

> 来源：https://uqer.io/community/share/54ae4117f9f06c276f651a07

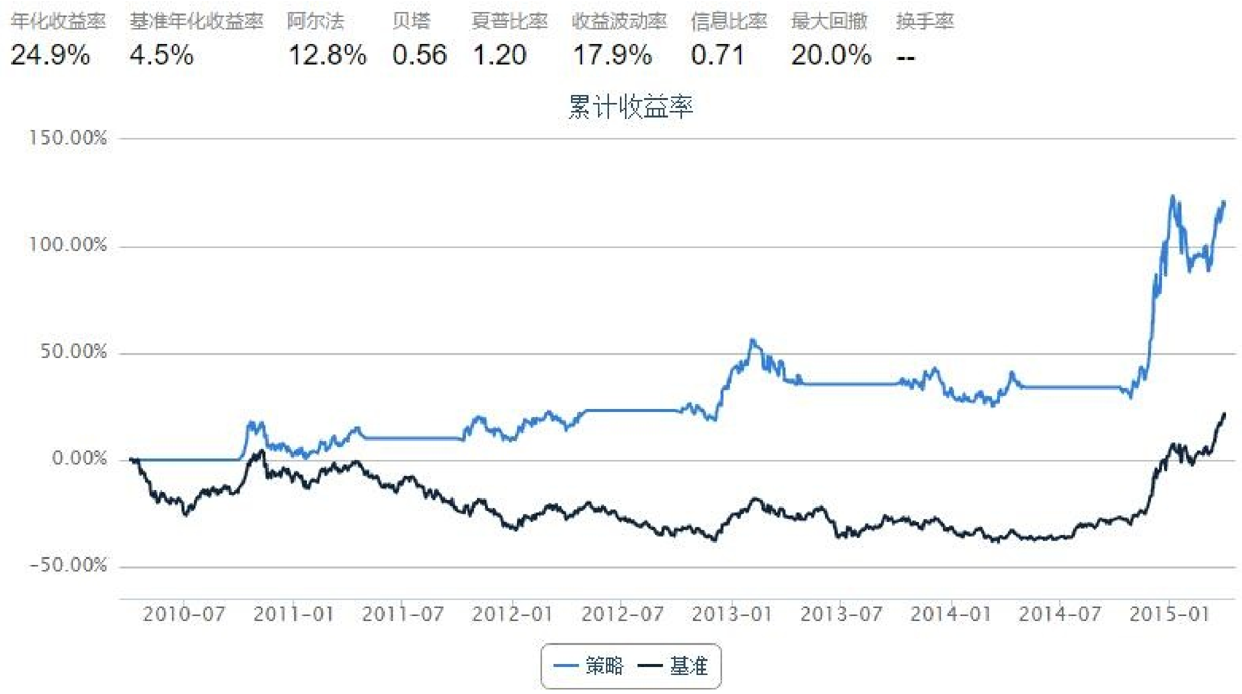
策略思路

“万圣节效应”：每年10月到次年5月，股票市场会出现上涨的趋势

策略实现

- 股票池：流动性充足的10只个股，包括工商银行、中国石化等
- 每年10月，将账户中现金平均分成10份，分别买入相应的10只个股，满仓；次年5月全部抛出，空仓

```
start = '2010-04-01'                       # 回测起始时间
end = '2015-04-01'                         # 回测结束时间
benchmark = 'HS300'                        # 策略参考标准
# 证券池,流动性充足的10只个股
universe = ['601398.XSHG', '600028.XSHG', '601988.XSHG', '600036.XSHG', '600030.XSHG', '601318.XSHG', '600000.XSHG', '600019.XSHG', '600519.XSHG', '601166.XSHG']
capital_base = 100000                      # 起始资金
longest_history = 1                       # handle_data 函数中可以使用的历史数据最长窗口长度
refresh_rate = 1                           # 调仓频率，即每 refresh_rate 个交易日执行一次 handle_data() 函数

def initialize(account):                   # 初始化虚拟账户状态
    pass

def handle_data(account):                  # 每个交易日的买入卖出指令

    yesterday = account.get_attribute_history('closePrice',1)
    for stock in account.universe:
        today = account.current_date
        if stock not in account.valid_secpos and (today.month == 10):  # 10月买入
            # 现金平均分成10份，买入
            amount = int(account.cash/len(account.universe)/ yesterday[stock][0])
            order(stock, amount)

        elif stock in account.valid_secpos and (today.month == 5):   # 5月卖出
            order_to(stock, 0)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
24.9%	4.5%	12.8%	0.56	1.20	17.9%	0.71	20.0%	--

夏买电，东买煤？

> 来源：https://uqer.io/community/share/563c8050f9f06c713ddfeb6c

冬吃萝卜夏吃姜？冬炒煤来夏炒电 ？

行业分类 ：申万二级行业

行业涨幅 ：行业成分股市值加权

验证时间 ：冬天(12,1,2)、夏天(7,8,9)..O、O .... 不懂什么夏至，春分，冬至啊/........

主观预测 ：然并卵好吗。!!!....唱唱 炒股歌 都能攥钱还要我们混吗？

参考 ：Uqer社区牛人李杰关于行业涨幅统计的贴子

```
#获得行业信息
def GetEquIndustry(universe,field):
    num = 100
    cnt_num = len(universe)/num
    if cnt_num > 0:
        df = pd.DataFrame({})
        for i in range(cnt_num) :
            sub_df = DataAPI.EquIndustryGet(secID=universe[i*num
:(i+1)*num],field=field)
            df = pd.concat([df,sub_df])
        if (i+1)*num != len(universe):
            sub_df = DataAPI.EquIndustryGet(secID=universe[(i+1)
*num:],field=field)
            df = pd.concat([df,sub_df])
    else:
        df = DataAPI.EquIndustryGet(secID=universe,field=field)
    return df
```

选取行业，将行业所属股票加入 `universe`

```
from CAL.PyCAL import *
import pandas as pd
cal = Calendar('China.SSE')
universe = DataAPI.EquGet(equTypeCD='A')['secID'].tolist()    # 获得全A股的secID
id2nm = lambda x:x[0:6]
tk_list_A = map(id2nm,universe)    #获得全A股的ticker

Ind_info = GetEquIndustry(universe=universe,field=['ticker','secShortName','industryName2'])    #获得个股的申万行业分类
Ind_info_gp = Ind_info.groupby('industryName2')#按照行业分组

universe = []
Ind_tks_dic = {}    #获得每个行业包含的股票
for ind_nm,sub_info in Ind_info_gp:
    if ind_nm in ['电力' , '煤炭开采'] :
        Ind_tks_dic[ind_nm] = sub_info.drop_duplicates('ticker')['ticker'].tolist()
        universe += Ind_tks_dic[ind_nm]
# print len(universe)
```

获取所需各段时间段的数据，放入 `Data_time` 中

```
from pandas import DataFrame,Series
from CAL.PyCAL import *
cal = Calendar('China.SSE')
data = DataFrame()
field = ['ticker','secShortName','tradeDate','preClosePrice','closePrice','turnoverValue']
#时间轴(开始时间)
time = ['20150630','20140630','20130630','20120630','20110630','20141130', '20131130', '20121130', '20111130', '20101130']
#保存各个时间段的数据
Data_time = {}
#保存各个时间段的股票名字
tk_nm_dic ={}
for s in time :
    Data_time[s] = DataFrame()
    data_temp = DataAPI.MktEqudAdjGet( ticker = universe , field =field , beginDate = s , endDate = cal.advanceDate(s,'3M', BizDayConvention.Following).strftime('%Y%m%d'))
    data_temp['marketValue']  = DataAPI.MktEqudGet(ticker = universe ,field ='marketValue' , beginDate = s , endDate = cal.advanceDate(s,'3M', BizDayConvention.Following).strftime('%Y%m%d'))
    Data_time[s] = pd.concat([Data_time[s],data_temp])
    tk_nm_dic[s] = dict(zip(Data_time[s]['ticker'],Data_time[s]['secShortName']))                                   # 获得个股ticker与名称的对应字典

for s in Data_time.values() :
    s['tradeDate'] = pd.to_datetime(s['tradeDate'])                            # 将tradeDate这一列的格式由string改为datetime
    s['increase'] = s['closePrice']/s['preClosePrice']                             # 获得个股每天的收益
```

```
#股票数据统计
Stock_Data = {}
for s in Data_time.keys() :
    Stock_dict = {'ticker':[],'income':[],'turnoverValue':[] ,'m
arketValue' :[]}
    # 获得每个时间段的Data计算个股的收益和平均市值
    for tk,sub_info in Data_time[s].groupby('ticker') :
        income = sub_info['increase'].prod()-1
   # 获得在这段时间内该股的涨幅
        mkt_value = sub_info['marketValue'].sum()/len(sub_info)

        turnoverValue_avg = sub_info['turnoverValue'].sum()/len(
sub_info)
        Stock_dict['ticker'].append(tk)
        Stock_dict['income'].append(income)
        Stock_dict['marketValue'].append(mkt_value)
        Stock_dict['turnoverValue'].append(turnoverValue_avg)
    # 返回时间为Key的个股数据
    Stock_Data[s] = pd.DataFrame(Stock_dict)
```

```
#行业数据统计
Output_dicy = {}
Output_dicy['industry'] = []
Output_dicy['Num'] = []
Output_dicy['bigstk_Summer15'] = []
Output_dicy['bigstk_Winter14'] = []
for ind,tks in Ind_tks_dic.items() :
    for table in Stock_Data.keys() :
        if not table in Output_dicy.keys() :
            Output_dicy[table] = []
        sub_Industry = Stock_Data[table][Stock_Data[table]['tick
er'].isin(tks)]
        # 行业指数收益
        rtn_Industry = (sub_Industry['income']*sub_Industry['mar
ketValue']).sum()/sub_Industry['marketValue'].sum()
        # 成交量
        bigstk = sub_Industry.sort(columns='turnoverValue',ascen
ding=False)['ticker'][0:3].tolist()

        Output_dicy[table].append(rtn_Industry)
        # 计算成交量
        if table == '20150630' :
            Output_dicy['bigstk_Summer15'].append(map(lambda x:t
k_nm_dic['20150630'][x],bigstk))
        if table == '20141130' :
            Output_dicy['bigstk_Winter14'].append(map(lambda x:t
k_nm_dic['20141130'][x],bigstk))
    #最新行业成分数量
    Output_dicy['Num'].append(len(sub_Industry))
    Output_dicy['industry'].append(ind)

# 计算上证指数同期涨幅
for s in time :
    temp = DataAPI.MktIdxdGet(ticker= '000001' ,  beginDate = s
, endDate = cal.advanceDate(s,'3M', BizDayConvention.Following).
strftime('%Y%m%d'),field=u"secShortName,closeIndex",pandas="1")

    SH_rtn = temp['closeIndex'].values[-1] / temp['closeIndex'].
values[0] - 1
    Output_dicy[s].append(SH_rtn)
Output_dicy['Num'].append(1)
Output_dicy['industry'].append('上证指数')
Output_dicy['bigstk_Winter14'].append(None)
Output_dicy['bigstk_Summer15'].append(None)

Output_table = pd.DataFrame(Output_dicy)
```

```
# 夏天统计
Out_put = Output_table.loc[:,['industry','Num','20110630','20120630','20130630','20140630','20150630','bigstk_Summer15']]
Out_put.columns = [u'行业名称',u'该行业成分股数目(15年)',u'2011年夏天收益',u'2012年夏天收益',u'2013年夏天收益',u'2014年夏天收益',u'2015年夏天收益',u'2015年夏天成交量前三']
# print u'一共有%d个申万二级行业'%len(Out_put),u'，1年内行业涨幅'
Out_put.sort(u'2015年夏天收益' , ascending = False)
```

	行业名称	该行业成分股数目(15年)	2011年夏天收益	2012年夏天收益	2013年夏天收益	2014年夏天收益	2015年夏天收益
0	电力	65	-0.164885	-0.080913	0.047399	0.311616	-0.277611
2	上证指数	1	-0.145853	-0.068142	0.089925	0.154049	-0.286270
1	煤炭开采	44	-0.095143	-0.029945	0.070714	0.244293	-0.341531

14年夏天电力还不错。。。

```
# 冬天统计
Out_put = Output_table.loc[:,['industry','Num','20101130','20111130','20121130','20131130','20141130','bigstk_Winter14']]
Out_put.columns = [u'行业名称',u'该行业成分股数目(15年)',u'2010年冬天收益',u'2011年冬天收益',u'2012年冬天收益',u'2013年冬天收益',u'2014年冬天收益',u'2014年冬天成交量前三']
# print u'一共有%d个申万二级行业'%len(Out_put),u'，1年内行业涨幅'
Out_put.sort(u'2014年冬天收益' , ascending = False)
```

	行业名称	该行业成分股数目(15年)	2010年冬天收益	2011年冬天收益	2012年冬天收益	2013年冬天收益	2014年冬天收益
2	上证指数	1	0.030095	0.040744	0.194673	-0.068438	0.244808
0	电力	65	-0.018729	-0.003542	0.161399	-0.043282	0.210642
1	煤炭开采	44	0.059724	0.017702	0.157182	-0.209695	0.140903

冬买煤、冬买煤、冬买煤，呵呵！赶紧买...保证亏不死！ O.....~

历史的十一月板块涨幅

> 来源：https://uqer.io/community/share/563c8065f9f06c713ddfeb6d

大盘又开始疯长了，还记嘚去年十一月吗 ？

```
#获得行业信息
def GetEquIndustry(universe,field):
    num = 100
    cnt_num = len(universe)/num
    if cnt_num > 0:
        df = pd.DataFrame({})
        for i in range(cnt_num) :
            sub_df = DataAPI.EquIndustryGet(secID=universe[i*num
:(i+1)*num],field=field)
            df = pd.concat([df,sub_df])
        if (i+1)*num != len(universe):
            sub_df = DataAPI.EquIndustryGet(secID=universe[(i+1)
*num:],field=field)
            df = pd.concat([df,sub_df])
    else:
        df = DataAPI.EquIndustryGet(secID=universe,field=field)
    return df
```

```
from CAL.PyCAL import *
import pandas as pd
cal = Calendar('China.SSE')
universe = DataAPI.EquGet(equTypeCD='A')['secID'].tolist()    #
获得全A股的secID
id2nm = lambda x:x[0:6]
tk_list_A = map(id2nm,universe)    #获得全A股的ticker

Ind_info = GetEquIndustry(universe = universe ,field=['ticker','
secShortName','industryName2'])    #获得个股的申万行业分类
Ind_info_gp = Ind_info.groupby('industryName2')#按照行业分组

Ind_tks_dic = {}    #获得每个行业包含的股票
for ind_nm,sub_info in Ind_info_gp:
    Ind_tks_dic[ind_nm] = sub_info.drop_duplicates('ticker')['ti
cker'].tolist()
```

```
from pandas import DataFrame,Series
from CAL.PyCAL import *
cal = Calendar('China.SSE')
field = ['ticker','secShortName','tradeDate','preClosePrice','closePrice','turnoverValue']
#时间轴(开始时间)
time = ['20141031', '20131031', '20121031', '20111031', '20101031']
#保存各个时间段的数据
Data_time = {}
#保存各个时间段的股票名字
tk_nm_dic ={}

# 时间稍慢
for s in time :
    Data_time[s] = DataFrame()
    for x in universe :
        try :
            data_temp = DataAPI.MktEqudAdjGet( secID = x , field =field , beginDate = s , endDate = cal.advanceDate(s,'1M', BizDayConvention.Following).strftime('%Y%m%d'))
            data_temp['marketValue']  = DataAPI.MktEqudGet(secID = x ,field ='marketValue' , beginDate = s , endDate = cal.advanceDate(s,'1M', BizDayConvention.Following).strftime('%Y%m%d'))
            Data_time[s] = pd.concat([Data_time[s],data_temp])
        except :
            continue
    tk_nm_dic[s] = dict(zip(Data_time[s]['ticker'],Data_time[s]['secShortName']))                        # 获得个股ticker与名称的对应字典

for s in Data_time.values() :
    s['tradeDate'] = pd.to_datetime(s['tradeDate'])                           # 将tradeDate这一列的格式由string改为datetime
    s['increase'] = s['closePrice']/s['preClosePrice']                           # 获得个股每天的收益
```

```
# 股票数据统计
Stock_Data = {}
for s in Data_time.keys() :
    Stock_dict = {'ticker':[],'income':[],'turnoverValue':[] ,'marketValue' :[]}
    # 获得每个时间段的Data计算个股的收益和平均市值
    for tk,sub_info in Data_time[s].groupby('ticker') :
        income = sub_info['increase'].prod()-1
   # 获得在这段时间内该股的涨幅
        mkt_value = sub_info['marketValue'].sum()/len(sub_info)

        turnoverValue_avg = sub_info['turnoverValue'].sum()/len(sub_info)
        Stock_dict['ticker'].append(tk)
        Stock_dict['income'].append(income)
        Stock_dict['marketValue'].append(mkt_value)
        Stock_dict['turnoverValue'].append(turnoverValue_avg)
    # 返回时间为Key的个股数据
    Stock_Data[s] = pd.DataFrame(Stock_dict)
```

```
# 行业数据统计
Output_dicy = {}
Output_dicy['industry'] = []
Output_dicy['Num'] = []
Output_dicy['Nov14'] = []

for ind,tks in Ind_tks_dic.items() :
    for table in Stock_Data.keys() :
        if not table in Output_dicy.keys() :
            Output_dicy[table] = []
        sub_Industry = Stock_Data[table][Stock_Data[table]['ticker'].isin(tks)]
         # 成交量前三
        bigstk = sub_Industry.sort(columns='turnoverValue',ascending=False)['ticker'][0:3].tolist()
        # 行业指数收益
        if not sub_Industry['marketValue'].sum() == 0 :
            rtn_Industry = (sub_Industry['income']*sub_Industry['marketValue']).sum()/sub_Industry['marketValue'].sum()
            Output_dicy[table].append(rtn_Industry)
            if table == '20141031' :
                Output_dicy['Nov14'].append(map(lambda x:tk_nm_dic['20141031'][x],bigstk))

    if not sub_Industry['marketValue'].sum() == 0 :
        #最新行业成分数量
        Output_dicy['Num'].append(len(sub_Industry))
        Output_dicy['industry'].append(ind)

Output_table = pd.DataFrame(Output_dicy)
```

去年十一月疯长的大盘

```
# 统计并显示
Out_put = Output_table.loc[:,['industry','Num','20101031','20111
031','20121031','20131031','20141031','Nov14']]
Out_put.columns = [u'行业名称',u'该行业成分股数目(15年)',u'2010年十一
月',u'2011年十一月',u'2012年十一月',u'2013年十一月',u'2014年十一月',u'
2014年板块成交量前三']
Out_put[u'平均涨幅'] = (Out_put[u'2010年十一月']+Out_put[u'2011年十
一月']+Out_put[u'2012年十一月']+Out_put[u'2013年十一月']+Out_put[u'
2014年十一月']) / 5
print u'一共有%d个申万二级行业'%len(Out_put),u'  :  '
Out_put.sort(u'2014年十一月' , ascending = False).head(20)
```

```
一共有208个申万二级行业  :
```

	行业名称	该行业成分股数目(15年)	2010年十一月	2011年十一月	2012年十一月	2013年十一月	201十-
97	资本市场服务	18	-0.141082	-0.152839	-0.133307	0.098754	0.51
22	证券	20	-0.142078	-0.150951	-0.129955	0.094950	0.49

107	综合金融	21	-0.139421	-0.148480	-0.132800	0.095897	0.47
2	航空运输	6	-0.179507	-0.123943	-0.094819	0.015406	0.40
149	航空运输业	10	-0.173271	-0.107966	-0.082920	0.015486	0.34
38	保险业	3	-0.095615	-0.017941	-0.035695	0.154288	0.32
153	保险	4	-0.095476	-0.018500	-0.035806	0.154006	0.32

92	房屋建筑业	1	0.000000	0.000000	-0.127193	0.202055	0.31
112	高速公路	23	-0.108908	-0.088902	-0.073154	0.052344	0.27
193	石油加工、炼焦和核燃料加工业	18	0.007286	-0.034520	-0.127721	0.005377	0.25
122	建筑安装业	1	-0.113664	-0.006221	-0.139037	0.020833	0.21
139	基础建设	22	-0.037736	-0.088644	0.009290	0.011860	0.21
202	农、林、牧、渔服务业	1	-0.013893	0.032037	-0.106501	0.055081	0.21
113	房屋建设	4	-0.046656	-0.087118	0.006878	0.009602	0.20

176	园区开发	18	-0.042658	-0.077632	-0.073093	-0.033278	0.20
39	租赁业	1	-0.149817	0.011525	-0.106517	0.043152	0.20
82	土木工程建筑业	45	-0.028200	-0.077634	-0.003929	0.004428	0.18
194	货币金融服务	16	-0.060532	-0.046490	0.017593	0.012884	0.18
101	银行	16	-0.060532	-0.046490	0.017593	0.012884	0.18
	燃气						

67	和供应业	9	-0.097077	0.016135	-0.038805	0.005440	0.17

五年间十一月平均涨幅

```
Out_put.sort(u'平均涨幅' , ascending = False).head(20)
```

	行业名称	该行业成分股数目(15年)	2010年十一月	2011年十一月	2012年十一月	2013年十一月	201 十-
92	房屋建筑业	1	0.000000	0.000000	-0.127193	0.202055	0.31
38	保险业	3	-0.095615	-0.017941	-0.035695	0.154288	0.32
153	保险	4	-0.095476	-0.018500	-0.035806	0.154006	0.32

35	林业	5	0.010244	0.117180	0.028619	0.012805	0.08
200	运输设备	7	0.116078	-0.056020	0.095749	0.071869	0.00
97	资本市场服务	18	-0.141082	-0.152839	-0.133307	0.098754	0.51
202	农、林、牧、渔服务业	1	-0.013893	0.032037	-0.106501	0.055081	0.21
22	证券	20	-0.142078	-0.150951	-0.129955	0.094950	0.49
107	综合金融	21	-0.139421	-0.148480	-0.132800	0.095897	0.47

123	铁路运输业	3	-0.074811	-0.032704	0.033269	0.058432	0.15
111	餐饮业	4	0.116088	0.030976	-0.115720	0.094488	0.00
188	互联网和相关服务	12	0.063492	0.059126	-0.123481	0.001113	0.12
182	电气自动化设备	30	0.181047	0.021313	-0.121661	0.000477	0.04
20	专业技术服务业	9	0.173828	-0.040446	-0.119783	0.046313	0.05

193	石油加工、炼焦和核燃料加工业	18	0.007286	-0.034520	-0.127721	0.005377	0.25
4	铁路运输	3	-0.074811	-0.032704	0.028185	0.057472	0.13
139	基础建设	22	-0.037736	-0.088644	0.009290	0.011860	0.21
194	货币金融服务	16	-0.060532	-0.046490	0.017593	0.012884	0.18
101	银行	16	-0.060532	-0.046490	0.017593	0.012884	0.18

44	食品制造业	22	0.150009	0.017352	-0.080203	-0.045608	0.05

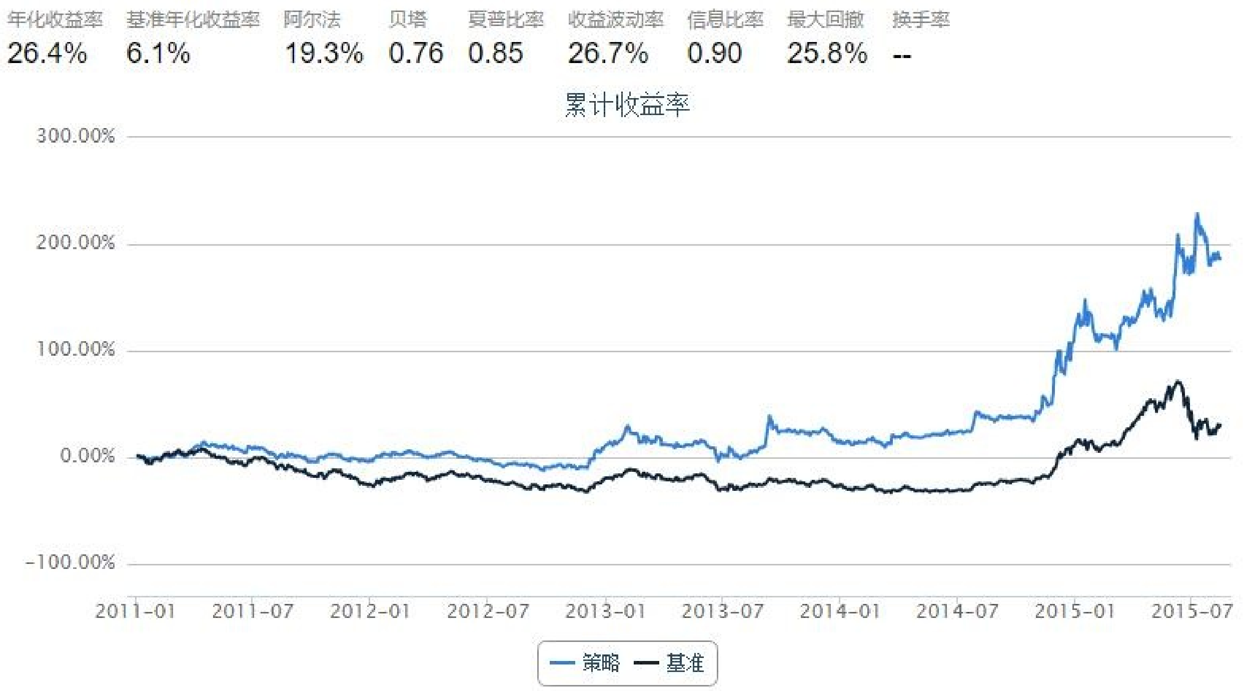
8.3 行业轮动

银行股轮动

> 来源：https://uqer.io/community/share/55d0314af9f06c915418c69a

策略说明：持续持有两支市净率最低银行股，每月换仓

```
import numpy as np
import pandas as pd
from pandas import DataFrame
import datetime

start = '2011-01-01'
end = '2015-08-14'
benchmark = 'HS300'
universe = ['000001.XSHE','002142.XSHE','600000.XSHG','600015.XSHG','600016.XSHG','600036.XSHG','601009.XSHG','601166.XSHG','601169.XSHG','601288.XSHG','601328.XSHG','601398.XSHG','601818.XSHG','601939.XSHG','601988.XSHG','601998.XSHG']

capital_base = 10000000
refresh_rate = 20

def initialize(account):
    pass

def handle_data(account):
    cal = Calendar('China.SSE')
    lastTDay = cal.advanceDate(account.current_date,'-1B',BizDayConvention.Preceding)
    today_str = lastTDay.strftime("%Y%m%d")

    tickers = []
    for stk in account.universe:
        if not np.isnan(account.referencePrice[stk]):
            tickers.append(stk[0:6])
    try:
        d=DataAPI.MktEqudGet(secID=u"",ticker=tickers,tradeDate=today_str,beginDate=u"",endDate=u"",field="secID,PB",pandas="1")

        d=d.sort(columns='PB',ascending=1)
        d=d.head(2)
        buylist = d['secID'].tolist()
        for stk in account.valid_secpos:
            if stk not in buylist:
                order_to(stk, 0)

         # 等权重买入所选股票
        portfolio_value = account.referencePortfolioValue


        for stk in buylist:
            if stk not in account.valid_secpos:
                order_to(stk, int(portfolio_value / account.referencePrice[stk] / 100.0 / len(buylist))*100)
    except:
        return
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
26.4%	6.1%	19.3%	0.76	0.85	26.7%	0.90	25.8%	--

申万二级行业在最近1年、3个月、5个交易日的涨幅统计

> 来源：https://uqer.io/community/share/55401d69f9f06c1c3d687fa1

功能：运行此代码可知最近1年、3个月、5个交易日各个行业的涨幅

注意1：行业是采用申万二级行业分类

注意2：行业涨幅是个股涨幅加权平均成交金额所得

```
def GetEquIndustry(universe,field):    #获得行业信息
    num = 100
    cnt_num = len(universe)/num
    if cnt_num > 0:
        df = pd.DataFrame({})
        for i in range(cnt_num):
            sub_df = DataAPI.EquIndustryGet(secID=universe[i*num:(i+1)*num],field=field)
            df = pd.concat([df,sub_df])
        if (i+1)*num != len(universe):
            sub_df = DataAPI.EquIndustryGet(secID=universe[(i+1)*num:],field=field)
            df = pd.concat([df,sub_df])
    else:
        df = DataAPI.EquIndustryGet(secID=universe,field=field)
    return df

def CountTime():    #获得可获得数据的最后一个交易日日期，返回的是datetime格式
    cal = Calendar('China.SSE')
    today = datetime.today()
    today_str = today.strftime("%Y%m%d")
    cal_date = Date.fromDateTime(today)
    time1=" 17:05:00"
    ben_time = datetime.strptime(today_str+time1,"%Y%m%d %H:%M:%S")
    if cal.isBizDay(cal_date) & (today>ben_time):    #如果是交易日，则判断当天是不是在15点前
        date = today
    else:    #如果当天不是交易日，则获得前一个交易日
        cal_wd = cal.advanceDate(cal_date, '-1B', BizDayConvention.Following)    #Date格式
        date = cal_wd.toDateTime()    #datetime格式
    return date

def GetMktEqud(tk_list,**kargs):
    num = 100
    cnt_num = len(tk_list)/num
    if cnt_num > 0:
        df = pd.DataFrame({})
        for i in range(cnt_num):
            sub_df = DataAPI.MktEqudGet(ticker=tk_list[i*num:(i+1
)*num],**kargs)
            df = pd.concat([df,sub_df])
        if (i+1)*num != len(tk_list):
            sub_df = DataAPI.MktEqudGet(ticker=tk_list[(i+1)*num
:],**kargs)
            df = pd.concat([df,sub_df])
    else:
        df = DataAPI.MktEqudGet(ticker=tk_list,**kargs)


    list_info = GetSecID(tk_list = tk_list,field=['ticker','list
Date'])    #获得上市日期信息，修改上市当天的数据中的preClosePrice为open
Price
    df.set_index(['ticker','tradeDate'],inplace=True)
    df_gp = df.groupby(level='ticker')
    df_new = pd.DataFrame({})    #储存新数据
    for tk,sub_info in df_gp:
        list_date = list_info['listDate'][list_info['ticker']==t
k].iloc[0]
        try:
            sub_info.loc[(tk,list_date),'preClosePrice'] = sub_i
nfo.loc[(tk,list_date),'openPrice']
        except:
            pass
        list_info = list_info[list_info['ticker']!=tk]
        df_new = pd.concat([df_new,sub_info])
    df_new.reset_index(inplace=True)
    return df_new
def GetSecID(tk_list,field):    #获取上市日期
    num = 100
    cnt_num = len(tk_list)/num
    if cnt_num > 0:
        df = pd.DataFrame({})
        for i in range(cnt_num):
            sub_df = DataAPI.SecIDGet(ticker=tk_list[i*num:(i+1)
*num],field=field)
            df = pd.concat([df,sub_df])
        if (i+1)*num != len(tk_list):
            sub_df = DataAPI.SecIDGet(ticker=tk_list[(i+1)*num:]
,field=field)
            df = pd.concat([df,sub_df])
    else:
        df = DataAPI.SecIDGet(ticker=tk_list,field=field)
    return df

def GetReturn(Mkt_Info_df):    #该函数是用来获得行业在一段时间内的收益
，以及个股在这段时间内的收益（先计算成分股在一段时间内的涨幅，再加权成交金额
得到主题的涨幅）
```

```
    Mkt_Info_df_gp = Mkt_Info_df.groupby('ticker')
    tk_inc_dic = {'ticker':[],'return':[],'turnoverValue':[]}
    for tk,sub_info in Mkt_Info_df_gp:
        rtn = sub_info['increase'].prod()-1
        tnv = sub_info['turnoverValue'].sum()/len(sub_info)     #获得平均成交金额
        tk_inc_dic['ticker'].append(tk)
        tk_inc_dic['return'].append(rtn)
        tk_inc_dic['turnoverValue'].append(tnv)
    tk_inc_df = pd.DataFrame(tk_inc_dic)
    tk_inc_df['secShortName'] = tk_inc_df['ticker'].apply(lambda x:tk2nm_dic[x])
    rtn_together = (tk_inc_df['return']*tk_inc_df['turnoverValue']).sum()/tk_inc_df['turnoverValue'].sum()     #获得该主题一段时间的涨幅，成交金额加权收益
    return rtn_together,tk_inc_df
```

```
import pandas as pd
cal = Calendar('China.SSE')

universe = DataAPI.EquGet(equTypeCD='A')['secID'].tolist()     #获得全A股的secID
id2nm = lambda x:x[0:6]
tk_list_A = map(id2nm,universe)     #获得全A股的ticker

Ind_info = GetEquIndustry(universe=universe,field=['ticker','secShortName','industryName2'])     #获得个股的申万行业分类
Ind_info_gp = Ind_info.groupby('industryName2')     #按照行业分组
Ind_tks_dic = {}     #获得每个行业包含的股票
for ind_nm,sub_info in Ind_info_gp:
    Ind_tks_dic[ind_nm] = sub_info['ticker'].tolist()
```

```
#获得统计的日期

#获得研究的结束时间，如果在当天收盘前，则为前一个交易日
endDate_dt = CountTime()
endDate_CAL = Date.fromDateTime(endDate_dt)

#前一季度的时间
beginDate_3M_CAL = cal.advanceDate(endDate_CAL,'-3M',BizDayConvention.Following)
beginDate_3M_dt = beginDate_3M_CAL.toDateTime()
#前5个交易日的时间
period_day = 5              ###################输入###################
period_CAL = '-'+str(period_day)+'B'
beginDate_5B_CAL = cal.advanceDate(endDate_CAL, period_CAL, BizDayConvention.Following)
beginDate_5B_dt = beginDate_5B_CAL.toDateTime()
```

```
#获得全A股在过去1年的市场行情

field = ['ticker','secShortName','tradeDate','preClosePrice','closePrice','turnoverValue','marketValue']
Mkt_Info_df_1Y = GetMktEqud(tk_list=tk_list_A,field =field)
#获取市场行情，省略了beginDate和endDate，则获取最近1年的行情
Mkt_Info_df_1Y['tradeDate'] = pd.to_datetime(Mkt_Info_df_1Y['tradeDate'])    #将tradeDate这一列的格式由string改为datetime
Mkt_Info_df_1Y['increase'] = Mkt_Info_df_1Y['closePrice']/Mkt_Info_df_1Y['preClosePrice']    #获得个股每天的收益

tk_nm_dic = dict(zip(Mkt_Info_df_1Y['ticker'],Mkt_Info_df_1Y['secShortName']))    #获得个股ticker与名称的对应字典

Mkt_Info_df_3M = Mkt_Info_df_1Y[Mkt_Info_df_1Y['tradeDate']>beginDate_3M_dt]    #最近3个月的信息
Mkt_Info_df_5B = Mkt_Info_df_1Y[Mkt_Info_df_1Y['tradeDate']>beginDate_5B_dt]    #最近5个交易日的信息
```

```
#统计得到个股以下信息：1年收益inc_1Y、3个月收益inc_3M、5个交易日收益inc_5B，1年平均市值mkv_1Y、3个月平均市值mkv_3M、5个交易日平均市值mkv_5B

#获得一年的涨幅、平均成交金额数据
Mkt_Info_df_1Y_gp = Mkt_Info_df_1Y.groupby('ticker')    #按照ticker分类
tk_rtn_dic_1Y = {'ticker':[],'inc_1Y':[],'tnv_1Y':[]}    #获得每个个股的收益和平均市值
for tk,sub_info in Mkt_Info_df_1Y_gp:
    inc_1Y = sub_info['increase'].prod()-1    #获得在这段时间内该股的涨幅
    tnv_avg_1Y = sub_info['turnoverValue'].sum()/len(sub_info)

    tk_rtn_dic_1Y['ticker'].append(tk)
    tk_rtn_dic_1Y['inc_1Y'].append(inc_1Y)
    tk_rtn_dic_1Y['tnv_1Y'].append(tnv_avg_1Y)
tk_rtn_df_1Y = pd.DataFrame(tk_rtn_dic_1Y)
#获得3个月的涨幅、平均成交金额数据
Mkt_Info_df_3M_gp = Mkt_Info_df_3M.groupby('ticker')    #按照ticker分类
tk_rtn_dic_3M = {'ticker':[],'inc_3M':[],'tnv_3M':[]}    #获得每个个股的收益和平均市值
for tk,sub_info in Mkt_Info_df_3M_gp:
    inc_3M = sub_info['increase'].prod()-1    #获得在这段时间内该股的涨幅
    tnv_avg_3M = sub_info['turnoverValue'].sum()/len(sub_info)

    tk_rtn_dic_3M['ticker'].append(tk)
    tk_rtn_dic_3M['inc_3M'].append(inc_3M)
    tk_rtn_dic_3M['tnv_3M'].append(tnv_avg_3M)
tk_rtn_df_3M = pd.DataFrame(tk_rtn_dic_3M)

#获得5个交易日的涨幅、平均成交金额数据
Mkt_Info_df_5B_gp = Mkt_Info_df_5B.groupby('ticker')    #按照ticker分类
tk_rtn_dic_5B = {'ticker':[],'inc_5B':[],'tnv_5B':[]}    #获得每个个股的收益和平均市值
for tk,sub_info in Mkt_Info_df_5B_gp:
    inc_5B = sub_info['increase'].prod()-1    #获得在这段时间内该股的涨幅
    tnv_avg_5B = sub_info['turnoverValue'].sum()/len(sub_info)

    tk_rtn_dic_5B['ticker'].append(tk)
    tk_rtn_dic_5B['inc_5B'].append(inc_5B)
    tk_rtn_dic_5B['tnv_5B'].append(tnv_avg_5B)
tk_rtn_df_5B = pd.DataFrame(tk_rtn_dic_5B)
```

```
#计算1年、3个月、5个交易日的行业涨幅

Ind_tks_dic
ind_rtn_dic = {'industry':[],'rtn_1Y':[],'rtn_3M':[],'rtn_5B':[],'bigstk_1Y':[],'bigstk_3M':[],'bigstk_5B':[],'Num':[]}
for ind,tks in Ind_tks_dic.items():
    sub_info_1Y = tk_rtn_df_1Y[tk_rtn_df_1Y['ticker'].isin(tks)]
    #先选出该行业的个股内容，包括收益和平均成交金额
    rtn_ind_1Y = (sub_info_1Y['inc_1Y']*sub_info_1Y['tnv_1Y']).sum()/sub_info_1Y['tnv_1Y'].sum()
    bigstk_1Y = sub_info_1Y.sort(columns='tnv_1Y',ascending=False)['ticker'][0:3].tolist()

    sub_info_3M = tk_rtn_df_3M[tk_rtn_df_3M['ticker'].isin(tks)]
    #先选出该行业的个股内容，包括收益和平均成交金额
    rtn_ind_3M = (sub_info_3M['inc_3M']*sub_info_3M['tnv_3M']).sum()/sub_info_3M['tnv_3M'].sum()
    bigstk_3M = sub_info_3M.sort(columns='tnv_3M',ascending=False)['ticker'][0:3].tolist()

    sub_info_5B = tk_rtn_df_5B[tk_rtn_df_5B['ticker'].isin(tks)]
    #先选出该行业的个股内容，包括收益和平均成交金额
    rtn_ind_5B = (sub_info_5B['inc_5B']*sub_info_5B['tnv_5B']).sum()/sub_info_5B['tnv_5B'].sum()
    bigstk_5B = sub_info_5B.sort(columns='tnv_5B',ascending=False)['ticker'][0:3].tolist()

    ind_rtn_dic['industry'].append(ind)
    ind_rtn_dic['Num'].append(len(sub_info_1Y))
    ind_rtn_dic['rtn_1Y'].append(rtn_ind_1Y)
    ind_rtn_dic['bigstk_1Y'].append(map(lambda x:tk_nm_dic[x],bigstk_1Y))
    ind_rtn_dic['rtn_3M'].append(rtn_ind_3M)
    ind_rtn_dic['bigstk_3M'].append(map(lambda x:tk_nm_dic[x],bigstk_3M))
    ind_rtn_dic['rtn_5B'].append(rtn_ind_5B)
    ind_rtn_dic['bigstk_5B'].append(map(lambda x:tk_nm_dic[x],bigstk_5B))

ind_rtn_df = pd.DataFrame(ind_rtn_dic)
```

```
ind_rtn_df_sort_1Y = ind_rtn_df.sort(columns='rtn_1Y',ascending=False).loc[:,['industry','rtn_1Y','bigstk_1Y','Num','rtn_3M','rtn_5B']]
ind_rtn_df_sort_1Y.columns = [u'行业名称',u'最近一年收益',u'一年内平均成交量最大的股票',u'该行业个股数目',u'最近3个月收益',u'最近5个交易日收益']
print u'一共有%d个申万二级行业'%len(ind_rtn_df_sort_1Y),u'，1年内行业涨幅'
ind_rtn_df_sort_1Y

一共有103个申万二级行业 ，1年内行业涨幅
```

	行业名称	最近一年收益	一年内平均成交量最大的股票	该行业个股数目	最近3个月收益	最近5个交易日收益
99	运输设备	5.303234	[中国南车, 中国北车, 晋西车轴]	8	1.173940	-0.014448
90	基础建设	4.691083	[中国中铁, 中国铁建, 中国交建]	25	0.921275	0.093444
1	航空运输	3.131124	[海南航空, 南方航空, 春秋航空]	6	0.876162	0.056124
8	航运	3.057806	[中海集运, 中国远洋, 中海发展]	12	1.016439	0.270692
15	计算机应用	2.754977	[用友网络, 恒生电子, 华胜天成]	96	0.860180	0.011083
98	互联网传媒	2.651445	[乐视网, 鹏博士, 东方财富]	24	0.802981	0.062447
39	专业工程	2.625073	[中国中冶, 中国化学, 航天工程]	20	0.915729	0.247647
53	房屋建设	2.582405	[中国建筑, 上海建工, 龙元建设]	4	0.629508	0.091970
			[同方股份,			

71	设备	2.366100	大华股份, 浪潮信息]	40	0.575884	0.014130
13	证券	2.361874	[中信证券, 东方证券, 海通证券]	23	0.523428	0.021711
34	通信运营	2.353681	[中国联通, 二六三]	2	1.195826	0.305338
11	仪器仪表	2.329112	[航天科技, 天和防务, 先河环保]	24	0.734938	0.069429
24	船舶制造	2.266239	[中国重工, 中国船舶, 广船国际]	8	0.602151	0.111529
7	专用设备	2.191821	[三一重工, 中国一重, 中联重科]	105	0.504312	0.083313
81	钢铁	2.183165	[包钢股份, 宝钢股份, 河北钢铁]	33	0.550343	0.035136
60	物流	2.115166	[怡亚通, 中储股份, 建发股份]	18	0.703845	0.008994
70	港口	2.063319	[天津港, 唐山港, 营口港]	17	0.389900	0.018574
33	农产品加工	1.938949	[中粮屯河, 中粮生化, 朗源股份]	17	0.646614	0.078169
94	通用机械	1.894176	[中核科技, 机器人, 晋亿实业]	96	0.604509	0.016274
52	高速公路	1.891902	[福建高速, 重庆路桥, 五洲交通]	18	0.645981	0.040706
23	化学纤维	1.821298	[中纺投资, 皖维高新, 华峰氨纶]	27	0.638380	0.085734
12	地面兵装	1.820928	[北方导航, 中航黑豹, 四创电子]	4	0.228236	-0.024187

			四创电子]			
44	服装家纺	1.805284	[雅戈尔, 际华集团, 探路者]	39	0.621356	0.002398
62	航空装备	1.793423	[中航飞机, 成飞集成, 洪都航空]	13	0.351965	0.005921
97	商业物业经营	1.791940	[小商品城, 农产品, 轻纺城]	15	0.638699	0.010417
36	多元金融	1.787572	[中航资本, 爱建股份, 安信信托]	11	0.330127	0.096928
41	景点	1.787031	[九华旅游, 宋城演艺, 长白山]	8	0.819598	0.030744
25	旅游综合	1.783006	[中青旅, 腾邦国际, 北部湾旅]	16	0.809576	0.005146
83	营销传播	1.779635	[蓝色光标, 省广股份, 腾信股份]	9	0.431896	-0.006486
56	电力	1.766792	[国电电力, 国投电力, 长江电力]	58	0.489520	0.050847
...	...	...	...	...	...	...
50	造纸	1.263068	[景兴纸业, 山鹰纸业, 冠豪高新]	23	0.480623	0.019398
95	化学制药	1.256582	[鲁抗医药, 恒瑞医药, 新和成]	62	0.478035	0.009671
69	综合	1.248448	[中国宝安, 中信国安, 东方集团]	53	0.444324	0.031787
65	公交	1.229096	[大众交通, 强生控股, 锦江投资]	8	0.451447	0.009324
			[新希望, 大			

75	饲料	1.216313	北农, 禾丰牧业]	11	0.403494	0.000612
58	半导体	1.206561	[国民技术, 长电科技, 华天科技]	22	0.471403	0.030992
32	畜禽养殖	1.177174	[罗牛山, 大康牧业, 仙坛股份]	16	0.529071	0.011948
26	机场	1.166775	[上海机场, 白云机场, 厦门空港]	4	0.275427	-0.039167
101	金属非金属新材料	1.131387	[烯碳新材, 方大炭素, 中科三环]	24	0.404832	0.009488
22	水泥制造	1.112418	[海螺水泥, 金隅股份, 亚泰集团]	19	0.411717	-0.003110
30	中药	1.090465	[康美药业, 吉林敖东, 云南白药]	59	0.506922	0.023478
28	医疗服务	1.086574	[恒康医疗, 爱尔眼科, 金陵药业]	8	0.770956	-0.007666
40	石油化工	1.062722	[中国石化, 广汇能源, 上海石化]	15	0.458994	0.155356
45	动物保健	1.061414	[中牧股份, 金宇集团, 升华拜克]	7	0.397987	0.002755
10	采掘服务	1.059690	[海油工程, 中海油服, 中矿资源]	11	0.391112	-0.026253
9	农业综合	1.047126	[大禹节水, 农发种业]	2	0.215880	-0.004511
49	银行	1.031171	[兴业银行, 浦发银行, 民生银行]	16	0.243698	0.031925
46	汽车服务	1.024574	[庞大集团, 申华控股, 国机汽车]	6	0.572989	0.112366

43	稀有金属	1.023880	[北方稀土, 五矿稀土, 厦门钨业]	21	0.326484	0.028529
79	石油开采	0.999020	[中国石油]	1	0.169796	0.103156
77	种植业	0.997333	[亚盛集团, 海南橡胶, 隆平高科]	14	0.255847	-0.008357
76	玻璃制造	0.988161	[南玻A, 金刚玻璃, 金晶科技]	9	0.495342	0.117769
6	汽车整车	0.976699	[上汽集团, 比亚迪, 长安汽车]	22	0.300511	0.031516
91	光学光电子	0.960364	[京东方A, 三安光电, 东旭光电]	42	0.389974	-0.029134
21	食品加工	0.873128	[伊利股份, 双汇发展, 梅花生物]	33	0.383895	-0.016259
63	园林工程	0.856421	[东方园林, 蒙草抗旱, 棕榈园林]	7	0.421860	-0.027685
38	渔业	0.778087	[大湖股份, 好当家, 獐子岛]	10	0.389029	0.018921
93	饮料制造	0.774735	[五粮液, 贵州茅台, 泸州老窖]	36	0.267515	-0.024910
2	其他轻工制造	0.742366	[易尚展示]	1	0.742366	0.742366
55	餐饮	0.552202	[中科云网, 西安饮食, 零七股份]	4	0.192203	0.013139

```
103 rows × 6 columns
```

```
ind_rtn_df_sort_3M = ind_rtn_df.sort(columns='rtn_3M',ascending=False).loc[:,['industry','rtn_3M','bigstk_3M','Num','rtn_1Y','rtn_5B']]
ind_rtn_df_sort_3M.columns = [u'行业名称',u'最近3个月收益',u'3个月内平均成交量最大的股票',u'该行业个股数目',u'最近一年收益',u'最近5个交易日收益']
print u'一共有%d个申万二级行业'%len(ind_rtn_df_sort_1Y),u'，3个月内行业涨幅'
ind_rtn_df_sort_3M
```

```
一共有103个申万二级行业 ，3个月内行业涨幅
```

	行业名称	最近3个月收益	3个月内平均成交量最大的股票	该行业个股数目	最近一年收益	最近5个交易日收益
34	通信运营	1.195826	[中国联通, 二六三]	2	2.353681	0.305338
99	运输设备	1.173940	[中国南车, 中国北车, 晋西车轴]	8	5.303234	-0.014448
8	航运	1.016439	[中海集运, 中国远洋, 中海发展]	12	3.057806	0.270692
90	基础建设	0.921275	[中国中铁, 中国铁建, 中国交建]	25	4.691083	0.093444
39	专业工程	0.915729	[中国中冶, 中国化学, 中工国际]	20	2.625073	0.247647
1	航空运输	0.876162	[南方航空, 海南航空, 中国国航]	6	3.131124	0.056124
15	计算机应用	0.860180	[大智慧, 用友网络, 恒生电子]	96	2.754977	0.011083
41	景点	0.819598	[宋城演艺, 九华旅游, 峨眉山A]	8	1.787031	0.030744
25	旅游综	0.809576	[中青旅, 腾邦国际, 号	16	1.783006	0.005146

			百控股]			
98	互联网传媒	0.802981	[乐视网, 掌趣科技, 鹏博士]	24	2.651445	0.062447
100	装修装饰	0.789720	[金螳螂, 洪涛股份, 亚厦股份]	12	1.460358	0.022219
28	医疗服务	0.770956	[迪安诊断, 恒康医疗, 爱尔眼科]	8	1.086574	-0.007666
51	电子制造	0.755818	[歌尔声学, 蓝思科技, 得润电子]	32	1.431560	-0.004972
2	其他轻工制造	0.742366	[易尚展示]	1	0.742366	0.742366
11	仪器仪表	0.734938	[航天科技, 先河环保, 雪迪龙]	24	2.329112	0.069429
57	环保工程及服务	0.722461	[万邦达, 碧水源, 桑德环境]	28	1.687707	0.021656
5	医疗器械	0.711224	[和佳股份, 千山药机, 新华医疗]	22	1.628250	0.044459
60	物流	0.703845	[怡亚通, 建发股份, 中储股份]	18	2.115166	0.008994
61	一般零售	0.669449	[文峰股份, 国际医学, 永辉超市]	49	1.501575	0.033214
89	电气自动化设备	0.647961	[国电南瑞, 许继电气, 汇川技术]	38	1.429676	0.017054
33	农产品加工	0.646614	[中粮屯河, 中粮生化, 东凌粮油]	17	1.938949	0.078169
52	高速公路	0.645981	[中原高速, 福建高速, 重庆路桥]	18	1.891902	0.040706

97	商业物业经营	0.638699	[小商品城, 农产品, 华联股份]	15	1.791940	0.010417
23	化学纤维	0.638380	[皖维高新, 中纺投资, 华峰氨纶]	27	1.821298	0.085734
29	包装印刷	0.637423	[劲嘉股份, 陕西金叶, 奥瑞金]	24	1.531204	0.042091
53	房屋建设	0.629508	[中国建筑, 上海建工, 龙元建设]	4	2.582405	0.091970
44	服装家纺	0.621356	[雅戈尔, 际华集团, 美邦服饰]	39	1.805284	0.002398
0	家用轻工	0.621297	[宜华木业, 威华股份, 明牌珠宝]	43	1.272965	0.081527
94	通用机械	0.604509	[中核科技, 机器人, 中航重机]	96	1.894176	0.016274
24	船舶制造	0.602151	[中国重工, 中国船舶, 广船国际]	8	2.266239	0.111529
...	...	...	...	...	...	...
63	园林工程	0.421860	[东方园林, 普邦园林, 蒙草抗旱]	7	0.856421	-0.027685
22	水泥制造	0.411717	[海螺水泥, 亚泰集团, 金隅股份]	19	1.112418	-0.003110
86	其他交运设备	0.411019	[隆鑫通用, 中国嘉陵, 深中华A]	7	1.294861	-0.028150
101	金属非金属新材料	0.404832	[烯碳新材, 中科三环, 沃尔核材]	24	1.131387	0.009488
75	饲料	0.403494	[新希望, 大北农, 海大	11	1.216313	0.000612

45	动物保健	0.397987	[中牧股份, 金宇集团, 瑞普生物]	7	1.061414	0.002755
92	贸易	0.392636	[辽宁成大, 厦门国贸, 五矿发展]	23	1.607619	0.026065
10	采掘服务	0.391112	[海油工程, 中海油服, 恒泰艾普]	11	1.059690	-0.026253
91	光学光电子	0.389974	[京东方A, 东旭光电, 三安光电]	42	0.960364	-0.029134
70	港口	0.389900	[上港集团, 宁波港, 唐山港]	17	2.063319	0.018574
38	渔业	0.389029	[大湖股份, 獐子岛, 好当家]	10	0.778087	0.018921
21	食品加工	0.383895	[伊利股份, 梅花生物, 双汇发展]	33	0.873128	-0.016259
62	航空装备	0.351965	[中航飞机, 中航动力, 洪都航空]	13	1.793423	0.005921
84	园区开发	0.349786	[张江高科, 陆家嘴, 浦东金桥]	9	1.700588	0.059708
85	燃气	0.332420	[深圳燃气, 陕天然气, 金鸿能源]	7	1.489302	-0.018759
36	多元金融	0.330127	[中航资本, 大众公用, 爱建股份]	11	1.787572	0.096928
43	稀有金属	0.326484	[北方稀土, 五矿稀土, 锡业股份]	21	1.023880	0.028529
14	电机	0.325072	[卧龙电气, 佳电股份, 通达动力]	8	1.455971	-0.022692

47	航天装备	0.323204	[中国卫星, 航天电子, 航天动力]	6	1.422559	0.032242
6	汽车整车	0.300511	[上汽集团, 比亚迪, 一汽轿车]	22	0.976699	0.031516
26	机场	0.275427	[上海机场, 深圳机场, 白云机场]	4	1.166775	-0.039167
93	饮料制造	0.267515	[五粮液, 贵州茅台, 泸州老窖]	36	0.774735	-0.024910
77	种植业	0.255847	[亚盛集团, 海南橡胶, 隆平高科]	14	0.997333	-0.008357
49	银行	0.243698	[兴业银行, 中国银行, 浦发银行]	16	1.031171	0.031925
12	地面兵装	0.228236	[北方导航, 中航黑豹, 四创电子]	4	1.820928	-0.024187
9	农业综合	0.215880	[大禹节水, 农发种业]	2	1.047126	-0.004511
55	餐饮	0.192203	[中科云网, 零七股份, 全聚德]	4	0.552202	0.013139
79	石油开采	0.169796	[中国石油]	1	0.999020	0.103156
18	林业	0.151489	[平潭发展, 吉林森工, 福建金森]	5	1.573602	-0.054994
72	保险	0.122277	[中国平安, 中国人寿, 中国太保]	5	1.480924	0.001178

```
103 rows × 6 columns
```

```
ind_rtn_df_sort_5B = ind_rtn_df.sort(columns='rtn_5B',ascending=False).loc[:,['industry','rtn_5B','bigstk_5B','Num','rtn_1Y','rtn_5B']]
ind_rtn_df_sort_5B.columns = [u'行业名称',u'最近5个交易日收益',u'5个交易日内平均成交量最大的股票',u'该行业个股数目',u'最近一年收益',u'最近3个月收益']
print u'一共有%d个申万二级行业'%len(ind_rtn_df_sort_1Y),u'，5个交易日内行业涨幅'
ind_rtn_df_sort_5B

一共有103个申万二级行业 ，5个交易日内行业涨幅
```

	行业名称	最近5个交易日收益	5个交易日内平均成交量最大的股票	该行业个股数目	最近一年收益	最近3个月收益
2	其他轻工制造	0.742366	[易尚展示]	1	0.742366	0.742366
34	通信运营	0.305338	[中国联通, 二六三]	2	2.353681	0.305338
8	航运	0.270692	[中国远洋, 中海集运, 中海发展]	12	3.057806	0.270692
39	专业工程	0.247647	[中国中冶, 中国化学, 中工国际]	20	2.625073	0.247647
40	石油化工	0.155356	[中国石化, 广汇能源, 上海石化]	15	1.062722	0.155356
76	玻璃制造	0.117769	[南玻A, 金晶科技, 耀皮玻璃]	9	0.988161	0.117769
46	汽车服务	0.112366	[庞大集团, 申华控股, 中国汽研]	6	1.024574	0.112366
24	船舶制造	0.111529	[中国重工, 中国船舶, 广船国际]	8	2.266239	0.111529
79	石油开采	0.103156	[中国石油]	1	0.999020	0.103156

36	多元金融	0.096928	[中航资本, 大众公用, 渤海租赁]	11	1.787572	0.096928
90	基础建设	0.093444	[中国中铁, 中国铁建, 中国交建]	25	4.691083	0.093444
53	房屋建设	0.091970	[中国建筑, 上海建工, 龙元建设]	4	2.582405	0.091970
35	酒店	0.091441	[华天酒店, 锦江股份, 大东海A]	7	1.469323	0.091441
23	化学纤维	0.085734	[石化油服, 中纺投资, 华西股份]	27	1.821298	0.085734
7	专用设备	0.083313	[中国一重, 三一重工, 中联重科]	105	2.191821	0.083313
0	家用轻工	0.081527	[宜华木业, 威华股份, 乐凯胶片]	43	1.272965	0.081527
33	农产品加工	0.078169	[中粮屯河, 中粮生化, 国投中鲁]	17	1.938949	0.078169
3	铁路运输	0.074405	[大秦铁路, 广深铁路, 铁龙物流]	3	1.552497	0.074405
11	仪器仪表	0.069429	[航天科技, 雪迪龙, 天和防务]	24	2.329112	0.069429
42	其他采掘	0.063680	[攀钢钒钛, 西藏矿业, 金岭矿业]	9	1.334411	0.063680
98	互联网传媒	0.062447	[乐视网, 掌趣科技, 鹏博士]	24	2.651445	0.062447
84	园区开发	0.059708	[张江高科, 陆家嘴, 南京高科]	9	1.700588	0.059708
			[海南航空, 南			

1	运输	0.056124	方航空, 中国国航]	6	3.131124	0.056124
73	文化传媒	0.055896	[百视通, 东方明珠, 电广传媒]	35	1.552490	0.055896
80	白色家电	0.053758	[格力电器, 青岛海尔, 美的集团]	44	1.321674	0.053758
16	化学原料	0.052725	[内蒙君正, 三友化工, 中泰化学]	23	1.363827	0.052725
56	电力	0.050847	[国电电力, 国投电力, 浙能电力]	58	1.766792	0.050847
68	工业金属	0.047487	[中国铝业, 南山铝业, 江西铜业]	42	1.603236	0.047487
5	医疗器械	0.044459	[千山药机, 新华医疗, 鱼跃医疗]	22	1.628250	0.044459
27	煤炭开采	0.042696	[中国神华, 中煤能源, 永泰能源]	42	1.308314	0.042696
...	...	...	...	...	...	...
65	公交	0.009324	[大众交通, 强生控股, 锦江投资]	8	1.229096	0.009324
60	物流	0.008994	[怡亚通, 建发股份, 中储股份]	18	2.115166	0.008994
66	其他建材	0.006497	[国栋建设, 国睿科技, 濮耐股份]	41	1.446100	0.006497
62	航空装备	0.005921	[中航飞机, 中航动力, 洪都航空]	13	1.793423	0.005921
25	旅游综合	0.005146	[号百控股, 中青旅, 腾邦国际]	16	1.783006	0.005146

45	动物保健	0.002755	[天康生物, 中牧股份, 金宇集团]	7	1.061414	0.002755
44	服装家纺	0.002398	[际华集团, 美邦服饰, 报喜鸟]	39	1.805284	0.002398
72	保险	0.001178	[中国平安, 中国太保, 中国人寿]	5	1.480924	0.001178
88	通信设备	0.001046	[中兴通讯, 网宿科技, 海格通信]	59	1.338543	0.001046
75	饲料	0.000612	[大北农, 新希望, 禾丰牧业]	11	1.216313	0.000612
59	橡胶	-0.002903	[黔轮胎A, 青岛双星, 赛轮金宇]	14	1.310560	-0.002903
22	水泥制造	-0.003110	[海螺水泥, 金隅股份, 冀东水泥]	19	1.112418	-0.003110
9	农业综合	-0.004511	[农发种业, 大禹节水]	2	1.047126	-0.004511
51	电子制造	-0.004972	[歌尔声学, 蓝思科技, 得润电子]	32	1.431560	-0.004972
83	营销传播	-0.006486	[蓝色光标, 省广股份, 华谊嘉信]	9	1.779635	-0.006486
28	医疗服务	-0.007666	[迪安诊断, 恒康医疗, 泰格医药]	8	1.086574	-0.007666
77	种植业	-0.008357	[亚盛集团, 海南橡胶, 隆平高科]	14	0.997333	-0.008357
99	运输设备	-0.014448	[中国南车, 中国北车, 晋西车轴]	8	5.303234	-0.014448
21	食品加工	-0.016259	[伊利股份, 梅花生物, 汤臣	33	0.873128	-0.016259

85	燃气	-0.018759	[国新能源, 长春燃气, 金鸿能源]	7	1.489302	-0.018759
14	电机	-0.022692	[江特电机, 卧龙电气, 佳电股份]	8	1.455971	-0.022692
12	地面兵装	-0.024187	[北方导航, 中航黑豹, 光电股份]	4	1.820928	-0.024187
17	视听器材	-0.024822	[TCL集团, 四川长虹, 海信电器]	9	1.427695	-0.024822
93	饮料制造	-0.024910	[贵州茅台, 五粮液, 泸州老窖]	36	0.774735	-0.024910
10	采掘服务	-0.026253	[海油工程, 中海油服, 恒泰艾普]	11	1.059690	-0.026253
63	园林工程	-0.027685	[东方园林, 普邦园林, 蒙草抗旱]	7	0.856421	-0.027685
86	其他交运设备	-0.028150	[中国嘉陵, 隆鑫通用, 深中华A]	7	1.294861	-0.028150
91	光学光电子	-0.029134	[京东方A, 三安光电, 东旭光电]	42	0.960364	-0.029134
26	机场	-0.039167	[上海机场, 厦门空港, 深圳机场]	4	1.166775	-0.039167
18	林业	-0.054994	[平潭发展, 永安林业, 吉林森工]	5	1.573602	-0.054994

```
103 rows × 6 columns
```

8.4 主题轮动

快速研究主题神器

> 来源：https://uqer.io/community/share/551e5160f9f06c8f33904513

用于快速研究某个主题，可以获得以下信息

- 主题相关的成分股
- 主题在最近1年、3个月、5个交易日内的涨幅
- 依据涨幅和成交量来获取在最近1年、3个月、5个交易日内的主题龙头股，并列出龙头股在这段时间区间内的涨幅
- 依据通联算法，获得与主题相关度最高的个股以及个股在最近1年、3个月、5个交易日内的涨幅

该代码用法

- step1：先在输入1处输入待研究的主题名称，如“新能源汽车”，运行“输入1”所在的cell，可以看到该主题所对应的主题id。有可能有多个主题包含了输入的主题名称，需要从中挑选自己想要研究的主题
- step2：确定了主题id，在“输入2”所在cell修改 `theme_id` ，注意格式是字符串
- step3：运行所有cell，便可获取与主题相关的信息了

```
#先通过主题名称获得主题id
themeName = u'生物医药'                         #######################输入1，在此处输入要研究的主题名称#######################
field1 = ['themeID','themeName']
thms_id = DataAPI.ThemesContentGet(themeName=themeName,field=field1)
thmid2nm_dic = dict(zip(thms_id['themeID'],thms_id['themeName']))    #获得主题id与主题名称的对应
thms_id
```

	themeID	themeName
0	4462	生物医药股
1	120419	生物医药
2	120420	生物医药产业

```
##这里是输入
theme_id = '120419'               ###################输入2，由上面
可获得主题id，在此处输入主题id，注意格式是字符串###################

field2 = ['themeID','themeName','ticker','secShortName','returnS
core','textContributionScore','industryScore']
thm_tks = DataAPI.TickersByThemesGet(themeID=theme_id,field=fiel
d2)    #获得该主题相关的证券，以及证券与主题的相关度
tk2nm_dic = dict(zip(thm_tks['ticker'],thm_tks['secShortName']))
```

```
import pandas as pd
from CAL.PyCAL import *
cal = Calendar('China.SSE')

def CountTime():    #返回的是datetime格式
    today = datetime.today()
    today_str = today.strftime("%Y%m%d")
    cal_date = Date.fromDateTime(today)
    time1=" 15:05:00"
    ben_time = datetime.strptime(today_str+time1,"%Y%m%d %H:%M:%
S")
    if cal.isBizDay(cal_date) & (today>ben_time):    #如果是交易日
，则判断当天是不是在15点前
        date = today
    else:    #如果当天不是交易日，则获得前一个交易日
        cal_wd = cal.adjustDate(cal_date,BizDayConvention.Preced
ing)    #Date格式
        date = cal_wd.toDateTime()    #datetime格式
    return date

def GetMktEqud(tk_list,**kargs):    #该函数是用来调取市场行情数据，由
于调取时有长度限制，如果查询的个股数太多，需要分批调取
    num = 100
    cnt_num = len(tk_list)/num
    if cnt_num > 0:
        df = pd.DataFrame({})
        for i in range(cnt_num):
            sub_df = DataAPI.MktEqudGet(ticker=tk_list[i*num:(i+1
)*num],**kargs)
            df = pd.concat([df,sub_df])
        if (i+1)*num != len(tk_list):
            sub_df = DataAPI.MktEqudGet(ticker=tk_list[(i+1)*num
:],**kargs)
            df = pd.concat([df,sub_df])
    else:
        df = DataAPI.MktEqudGet(ticker=tk_list,**kargs)
    return df

def GetReturn(Mkt_Info_df):    #该函数是用来获得主题在一段时间内的收益
，以及个股在这段时间内的收益（先计算成分股在一段时间内的涨幅，再加权成交金额
```

```
得到主题的涨幅）

    Mkt_Info_df_gp = Mkt_Info_df.groupby('ticker')
    tk_inc_dic = {'ticker':[],'return':[],'turnoverValue':[]}
    for tk,sub_info in Mkt_Info_df_gp:
        rtn = sub_info['increase'].prod()-1
        tnv = sub_info['turnoverValue'].sum()/len(sub_info)     #
获得平均成交金额
        tk_inc_dic['ticker'].append(tk)
        tk_inc_dic['return'].append(rtn)
        tk_inc_dic['turnoverValue'].append(tnv)
    tk_inc_df = pd.DataFrame(tk_inc_dic)
    tk_inc_df['secShortName'] = tk_inc_df['ticker'].apply(lambda
 x:tk2nm_dic[x])
    rtn_together = (tk_inc_df['return']*tk_inc_df['turnoverValue'
]).sum()/tk_inc_df['turnoverValue'].sum()     #获得该主题一段时间的涨
幅，成交金额加权收益
    return rtn_together,tk_inc_df
```

```
print '主题关联的个股'
thm_tks
```

```
主题关联的个股
```

	themeID	themeName	ticker	secShortName	returnScore
0	120419	生物医药	000004	国农科技	0.935363
1	120419	生物医药	000403	ST生化	0.927900
2	120419	生物医药	000513	丽珠集团	0.963505
3	120419	生物医药	000538	云南白药	0.985011
4	120419	生物医药	000597	东北制药	0.988989
5	120419	生物医药	000661	长春高新	0.938084
6	120419	生物医药	000739	普洛药业	0.954498
7	120419	生物医药	000790	华神集团	0.816360
8	120419	生物医药	000820	金城股份	0.630109
9	120419	生物医药	000931	中关村	0.927900
10	120419	生物医药	000963	华东医药	0.693950
11	120419	生物医药	002004	华邦颖泰	0.791938
12	120419	生物医药	002007	华兰生物	0.942944

13	120419	生物医药	002019	亿帆鑫富	0.982201
14	120419	生物医药	002020	京新药业	0.915740
15	120419	生物医药	002030	达安基因	0.142927
16	120419	生物医药	002038	双鹭药业	0.680201
17	120419	生物医药	002102	冠福股份	0.847786
18	120419	生物医药	002107	沃华医药	0.000000
19	120419	生物医药	002219	恒康医疗	0.930044
20	120419	生物医药	002286	保龄宝	0.904069
21	120419	生物医药	002287	奇正藏药	0.897739
22	120419	生物医药	002294	信立泰	0.785857
23	120419	生物医药	002317	众生药业	0.927900
24	120419	生物医药	002349	精华制药	0.927900
25	120419	生物医药	002432	九安医疗	0.804717
26	120419	生物医药	002462	嘉事堂	0.835883
27	120419	生物医药	002550	千红制药	0.961297
28	120419	生物医药	002581	万昌科技	0.772591
29	120419	生物医药	002653	海思科	0.900234
...	...	...	...	...	...
52	120419	生物医药	600220	江苏阳光	0.754740
53	120419	生物医药	600222	太龙药业	0.866747
54	120419	生物医药	600249	两面针	0.944427
55	120419	生物医药	600252	中恒集团	0.907264
56	120419	生物医药	600267	海正药业	0.967912
57	120419	生物医药	600272	开开实业	0.995495
58	120419	生物医药	600276	恒瑞医药	0.935974
59	120419	生物医药	600297	美罗药业	0.833323
60	120419	生物医药	600332	白云山	0.956238
61	120419	生物医药	600340	华夏幸福	0.881892
62	120419	生物医药	600381	贤成矿业	0.921978
63	120419	生物医药	600385	ST金泰	0.765946

64	120419	生物医药	600422	昆药集团	0.956965
65	120419	生物医药	600503	华丽家族	0.927900
66	120419	生物医药	600521	华海药业	0.982925
67	120419	生物医药	600535	天士力	0.983813
68	120419	生物医药	600557	康缘药业	0.988432
69	120419	生物医药	600587	新华医疗	0.967148
70	120419	生物医药	600594	益佰制药	0.836619
71	120419	生物医药	600624	复旦复华	0.977262
72	120419	生物医药	600645	中源协和	0.599070
73	120419	生物医药	600666	西南药业	0.831056
74	120419	生物医药	600783	鲁信创投	0.878917
75	120419	生物医药	600789	鲁抗医药	0.993466
76	120419	生物医药	600826	兰生股份	0.913197
77	120419	生物医药	600867	通化东宝	0.822112
78	120419	生物医药	600873	梅花生物	0.958417
79	120419	生物医药	600895	张江高科	0.627730
80	120419	生物医药	601607	上海医药	0.519610
81	120419	生物医药	603168	莎普爱思	0.994970

```
82 rows × 7 columns
```

```
#获得该主题的上涨幅度

#获得研究的结束时间，如果在当天收盘前，则为前一个交易日
endDate_dt = CountTime()
endDate_CAL = Date.fromDateTime(endDate_dt)

#前一季度的时间
beginDate_3M_CAL = cal.advanceDate(endDate_CAL,Period('-3M'),BizDayConvention.Following)
beginDate_3M_dt = beginDate_3M_CAL.toDateTime()
#前5个交易日的时间
period_day = 5              ###################输入####################
period_CAL = '-'+str(period_day)+'B'
beginDate_5B_CAL = cal.advanceDate(endDate_CAL, period_CAL, BizDayConvention.Following)
beginDate_5B_dt = beginDate_5B_CAL.toDateTime()
```

```
#获得主题在这一年、一季度、5个交易日内的涨幅

tk_list = thm_tks['ticker'].tolist()    #获得主题关联的证券代码列表
field = ['ticker','secShortName','tradeDate','preClosePrice','closePrice','turnoverValue','marketValue']

#计算主题在最近1年的涨幅
Mkt_Info_df_1Y = GetMktEqud(tk_list=tk_list,field =field)     #获取市场行情，省略了beginDate和endDate，则获取最近1年的行情
Mkt_Info_df_1Y['tradeDate'] = pd.to_datetime(Mkt_Info_df_1Y['tradeDate'])    #将tradeDate这一列的格式由string改为datetime
Mkt_Info_df_1Y['increase'] = Mkt_Info_df_1Y['closePrice']/Mkt_Info_df_1Y['preClosePrice']
(rtn_1Y,tk_rt_df_1Y) = GetReturn(Mkt_Info_df_1Y)

#计算主题在最近3个月的涨幅
Mkt_Info_df_3M = Mkt_Info_df_1Y[Mkt_Info_df_1Y['tradeDate']>beginDate_3M_dt]
(rtn_3M,tk_rt_df_3M) = GetReturn(Mkt_Info_df_3M)

#计算主题在最近5个交易日的涨幅
Mkt_Info_df_5B = Mkt_Info_df_1Y[Mkt_Info_df_1Y['tradeDate']>beginDate_5B_dt]
(rtn_5B,tk_rt_df_5B) = GetReturn(Mkt_Info_df_5B)
```

```
def add_nm_rtn(mkt_df):    #将个股名称与收益拼接，方便做展示
    add_info_list = []
    for i in range(len(mkt_df)):
        add_info = mkt_df['secShortName'].iloc[i] + str(round(mk
t_df['return'].iloc[i],3))
        add_info_list.append(add_info)
    return add_info_list
```

```
#获取主题在最近1年、3个月、5个交易日内的龙头股及其涨幅
df_list = [tk_rt_df_1Y,tk_rt_df_3M,tk_rt_df_5B]
bigstk_dic = {'bigstk_by_rtn':[],'bigstk_by_rnv':[]}
for df_i in df_list:
    df_sort_rtn = df_i.sort(columns='return',ascending=False)[0:3
]    #按照收益率对其排序，取前3
    df_sort_tnv = df_i.sort(columns='turnoverValue',ascending=Fa
lse)[0:3]    #按照成交量对其排序，取前3
    bigstk_rtn_list = add_nm_rtn(df_sort_rtn)
    bigstk_tnv_list = add_nm_rtn(df_sort_tnv)
    bigstk_dic['bigstk_by_rtn'].append(bigstk_rtn_list)
    bigstk_dic['bigstk_by_rnv'].append(bigstk_tnv_list)
bigstk_dic['thm_rtn'] = [round(rtn_1Y,3),round(rtn_3M,3),round(r
tn_5B,3)]
bigstk_df = pd.DataFrame(bigstk_dic)
bigstk_df = bigstk_df.loc[:,['thm_rtn','bigstk_by_rtn','bigstk_b
y_rnv']]
bigstk_df.index = [u'最近一年',u'最近3个月',u'最近5个交易日']
bigstk_df.columns = [u'主题涨幅',u'龙头股_按涨幅',u'龙头股_按成交量']
print '主题：',thmid2nm_dic[int(theme_id)]
bigstk_df
```

```
主题： 生物医药
```

	主题涨幅	龙头股_按涨幅	龙头股_按成交量
最近一年	0.983	[沃华医药5.498, 莎普爱思4.354, 达安基因3.13]	[云南白药0.268, 达安基因3.13, 白云山0.344]
最近3个月	0.518	[沃华医药1.938, 达安基因1.348, 博腾股份1.149]	[达安基因1.348, 张江高科0.548, 上海医药0.418]
最近5个交易日	0.091	[江苏阳光0.266, 恒康医疗0.221, 兰生股份0.22]	[达安基因0.198, 华夏幸福0.122, 上海医药0.088]

```
#按照相关度做研究，不同维度得到的最相关的个股，查看其收益率
```

```
tks_rtnscore = thm_tks.sort(columns='returnScore',ascending=False)[0:3]['ticker'].tolist()    #根据returnScore排序
tks_textscore = thm_tks.sort(columns='textContributionScore',ascending=False)[0:3]['ticker'].tolist()     #根据textContributionScore排序
tks_indscore = thm_tks.sort(columns='industryScore',ascending=False)[0:3]['ticker'].tolist()     #根据industryScore排序
tks_score_list = [tks_rtnscore,tks_textscore,tks_indscore]
bigstk_score_dic = {}

def noname(df,lt):    #将结果按照传入的list中的ticker顺序排列，而不是默认由市场行情获得的的那个dataframe的顺序，我说清楚了吗
    new_df = pd.DataFrame({})
    for i in lt:
        a = df[df['ticker']==i]
        new_df = pd.concat([new_df,a])
    return new_df


for i in range(3):
    tk_score_list = tks_score_list[i]
    #先获得1年、3个月、5个交易日的dataframe
    sub_mkt_1Y = noname(tk_rt_df_1Y,tk_score_list)
    add_info_1Y = add_nm_rtn(sub_mkt_1Y)
    sub_mkt_3M = noname(tk_rt_df_3M,tk_score_list)
    add_info_3M = add_nm_rtn(sub_mkt_3M)
    sub_mkt_5B = noname(tk_rt_df_5B,tk_score_list)
    add_info_5B = add_nm_rtn(sub_mkt_5B)

    if i == 0:
        bigstk_score_dic['rtn_score'] = [add_info_1Y,add_info_3M,add_info_5B]
    if i == 1:
        bigstk_score_dic['text_score'] = [add_info_1Y,add_info_3M,add_info_5B]
    if i == 2:
        bigstk_score_dic['ind_score'] = [add_info_1Y,add_info_3M,add_info_5B]

bigstk_score_dic['thm_rtn'] = [round(rtn_1Y,3),round(rtn_3M,3),round(rtn_5B,3)]
bigstk_score_df = pd.DataFrame(bigstk_score_dic)

bigstk_score_df = bigstk_score_df.loc[:,['thm_rtn','text_score','ind_score','rtn_score']]
bigstk_score_df.index = [u'最近一年',u'最近3个月',u'最近5个交易日']
bigstk_score_df.columns = [u'主题涨幅',u'最相关_文本',u'最相关_行业',u'最相关_收益']
bigstk_score_df
```

	主题涨幅	最相关_文本	最相关_行业	最相关_收益
最近一年	0.983	[中关村0.986, 恒瑞医药0.642, 达安基因3.13]	[国农科技1.028, 中恒集团0.599, 华邦颖泰1.034]	[开开实业0.697, 莎普爱思4.354, 鲁抗医药1.183]
最近3个月	0.518	[中关村0.35, 恒瑞医药0.258, 达安基因1.348]	[国农科技0.648, 中恒集团0.224, 华邦颖泰0.902]	[开开实业0.241, 莎普爱思0.487, 鲁抗医药0.612]
最近5个交易日	0.091	[中关村0.097, 恒瑞医药0.096, 达安基因0.198]	[国农科技0.073, 中恒集团0.028, 华邦颖泰0.148]	[开开实业0.086, 莎普爱思0.037, 鲁抗医药0.197]

```
thm_tks_text = thm_tks.sort(columns='textContributionScore',ascending=False)[0:5]
print '排名按照textContributionScore(文本贡献关联度，主题和证券在新闻文本中的相似度，取值范围[0，1]，值越大表示关联度越高)'
thm_tks_text
```

```
排名按照textContributionScore(文本贡献关联度，主题和证券在新闻文本中的相似度，取值范围[0，1]，值越大表示关联度越高)
```

	themeID	themeName	ticker	secShortName	returnScore
9	120419	生物医药	000931	中关村	0.927900
58	120419	生物医药	600276	恒瑞医药	0.935974
15	120419	生物医药	002030	达安基因	0.142927
72	120419	生物医药	600645	中源协和	0.599070
67	120419	生物医药	600535	天士力	0.983813

```
thm_tks_ind = thm_tks.sort(columns='industryScore',ascending=False)[0:5]
print '排名按照industryScore(行业关联度，主题和证券在行业分布上的相似度，取值范围[0，1]，值越大表示关联度越高)'
thm_tks_ind
```

```
排名按照industryScore(行业关联度，主题和证券在行业分布上的相似度，取值范围[0，1]，值越大表示关联度越高)
```

	themeID	themeName	ticker	secShortName	returnScore
0	120419	生物医药	000004	国农科技	0.935363
55	120419	生物医药	600252	中恒集团	0.907264
11	120419	生物医药	002004	华邦颖泰	0.791938
72	120419	生物医药	600645	中源协和	0.599070
13	120419	生物医药	002019	亿帆鑫富	0.982201

```
thm_tks_rtn = thm_tks.sort(columns='returnScore',ascending=False
)[0:5]
print '排名按照returnScore(收益关联程度，主题和证券在短期收益上的相似度，
取值范围[0，1]，值越大表示关联度越高)'
thm_tks_rtn

排名按照returnScore(收益关联程度，主题和证券在短期收益上的相似度，取值范围[
0，1]，值越大表示关联度越高)
```

	themeID	themeName	ticker	secShortName	returnScore
57	120419	生物医药	600272	开开实业	0.995495
81	120419	生物医药	603168	莎普爱思	0.994970
75	120419	生物医药	600789	鲁抗医药	0.993466
4	120419	生物医药	000597	东北制药	0.988989
68	120419	生物医药	600557	康缘药业	0.988432

recommendation based on subject

> 来源：https://uqer.io/community/share/549d0203f9f06c4bb8863242

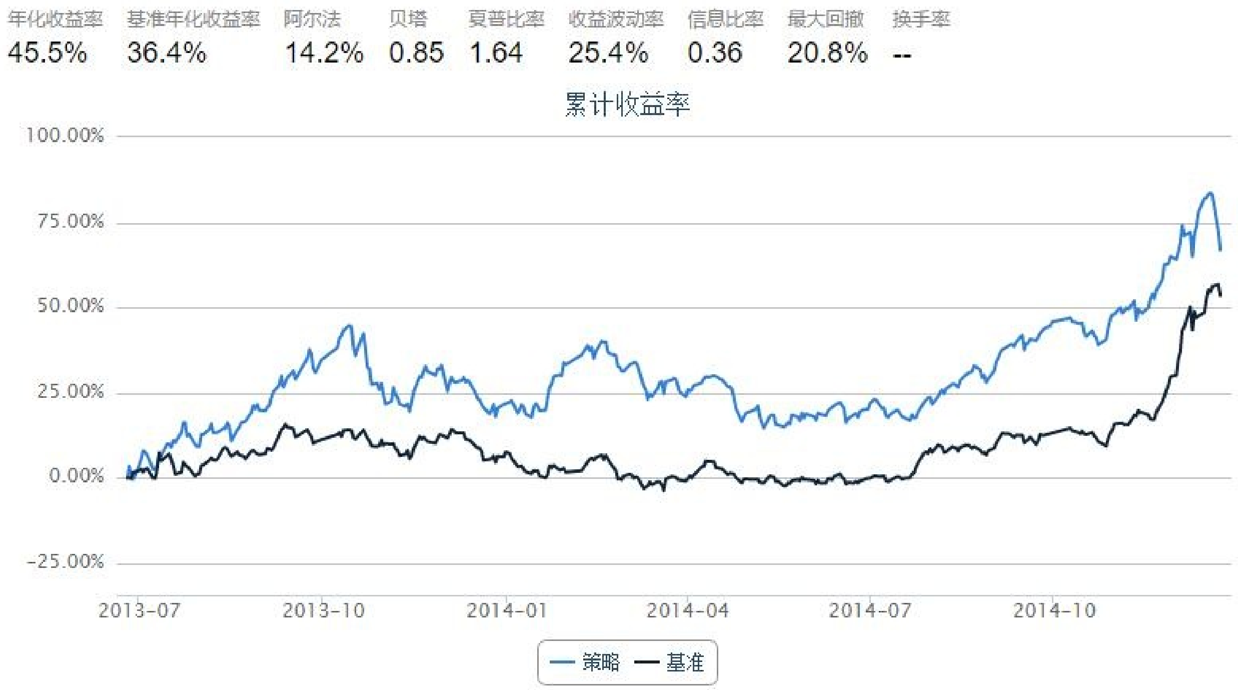
策略思路：

- step1：计算昨日所有主题的涨跌幅，根据涨跌幅排名，挑出涨幅最高的前 `n_sub` 个主题
- step2：根据昨日成交量挑选出每个主题的龙头股 `n_bigstk` 只
- 买入策略：昨日涨幅最高的前 `n_sub` 个主题，每个主题龙头股 `n_bigstk` 只，当日一共买入 `m*n` 只个股
- 卖出策略：持有固定天数 `hold_days` ，卖出。

此实验中， `n_sub=5` ， `n_bigstk=5` ， `hold_days=10`

文件 `sub_stk_info.txt` 是根据 `dataapi` 获得的文件，里面储存了多个主题及对应的股票列表，点击这里下载

```
a2=read('sub_stk_info.txt')
b2=a2.split('\r\n')
b2=b2[:-1]
sub_stk_dic={}
universe1=set([])
IDmap=lambda x:x +'.XSHG' if x[0]=='6' else x+'.XSHE'
for i2 in b2:
    i2=i2.split(':')
    i3=i2[1].split(',')
    sub_stk_dic[i2[0]]=map(IDmap,i3)
    universe1 |= set(i3)

start = datetime(2013, 6, 23)                # 回测起始时间
end   = datetime(2014, 12, 23)                # 回测结束时间
benchmark = 'HS300'                            # 使用沪深 300 作为
参考标准
universe = map(IDmap, list(universe1))
capital_base = 100000                       # 起始资金
#print len(universe)
hold_days=10
sell_stk_list=[]
for i in range(hold_days):
    sell_stk_list.append({})
j=hold_days
def initialize(account):                    # 初始化虚拟账户状态
    add_history('hist',1)

def handle_data(account):                # 每个交易日的买入卖出指令
    global sell_stk_list
    global j
    #计算昨日主题涨跌幅
    sub_increase_rate_dic={}
    sub_bigstk_dic={}
    #print 'today:',account.current_date
    for (subid,stkid_list) in sub_stk_dic.items():

        increase_rate_list=[]
        turnvol_list=[]
        #记录每只股票的成交量
        stk_turnvol_dic={}
        for stk in stkid_list:
            #停盘的情况
            if (stk not in account.universe) :
                continue
            close_price=account.hist[stk].iloc[0,3]
            pre_close_price=account.hist[stk].iloc[0,4]
            turnoverVol=account.hist[stk].iloc[0,5]
            stk_turnvol_dic[stk]=turnoverVol
            increase_rate_yes=(close_price-pre_close_price)*turnoverVol/pre_close_price
            increase_rate_list.append(increase_rate_yes)
            turnvol_list.append(turnoverVol)
        big_stk_list=sorted(stk_turnvol_dic.keys(),key=lambda x:stk_turnvol_dic[x], reverse=True)
        #买龙头股，每个主题买n只龙头股
        n_bigstk=5
        big_stk_list=big_stk_list[0:n_bigstk]
        sub_bigstk_dic[subid]=big_stk_list

        increase_rate_w=sum(increase_rate_list)/sum(turnvol_list)
        sub_increase_rate_dic[subid]=increase_rate_w
    sub_increase_rate_dic_sorted=sorted(sub_increase_rate_dic.keys(), key = lambda x:sub_increase_rate_dic[x], reverse = True)

    n_sub=5
    buy_subject_list=sub_increase_rate_dic_sorted[0:n_sub]
    buy_stk_list=[]
    for sub_id in buy_subject_list:
        buy_stk_list +=sub_bigstk_dic[sub_id]
    sell_next_dic={}
    for stk in buy_stk_list:
        if j>0:
            amount=int(account.position.cash/hold_days/len(buy_stk_list)/account.hist[stk].iloc[0,3])
            j -=1
        else:
            amount=int(account.position.cash/len(buy_stk_list)/account.hist[stk].iloc[0,3])
        order(stk,amount)
        sell_next_dic[stk]=amount
    sell_stk_list.insert(0,sell_next_dic)
    #print 'sell_stk_list:',sell_stk_list
    sell_today_dic=sell_stk_list.pop()
    #print 'sell_today_dic',sell_today_dic
    if sell_today_dic!={}:
        for (stk,amt) in sell_today_dic.items():
            #如果股票今天不能交易,就过hold_days再卖
            if stk not in account.universe:
                sell_stk_list[0][stk]=amt
            else:
                order(stk,-amt)
```



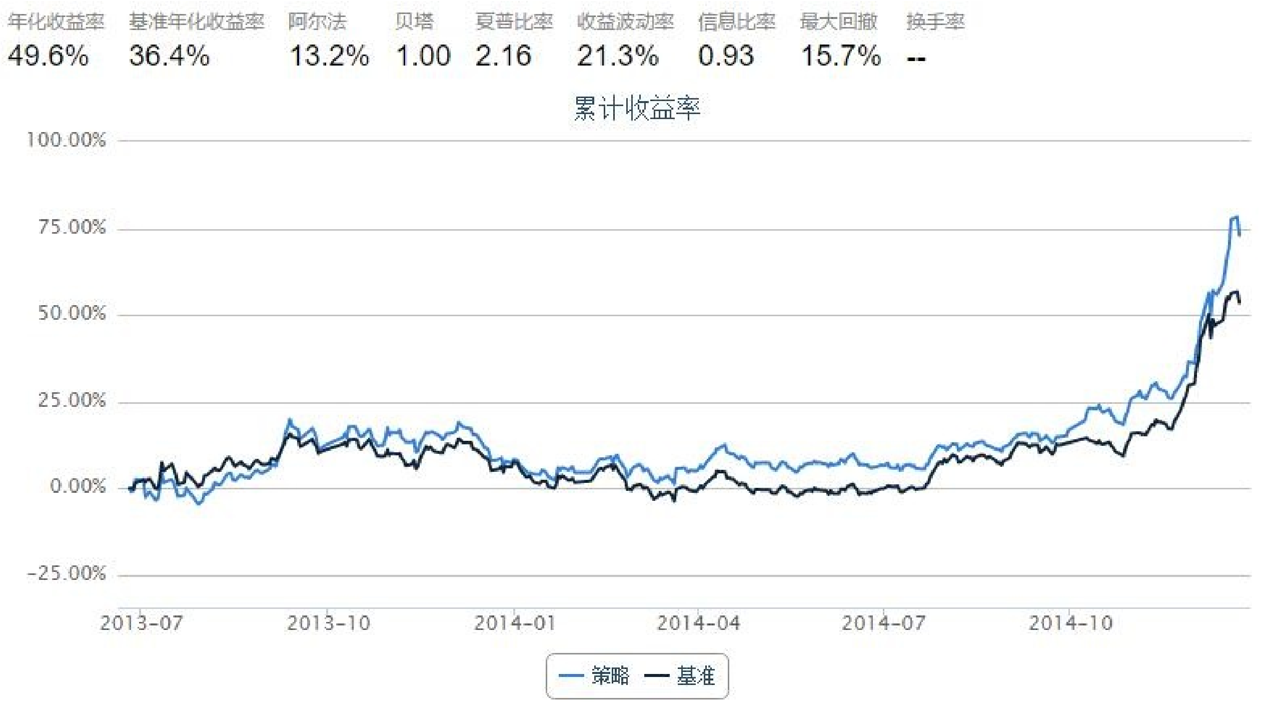
年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
45.5%	36.4%	14.2%	0.85	1.64	25.4%	0.36	20.8%	--

strategy7: recommendation based on theme

> 来源：https://uqer.io/community/share/54a3c0bff9f06c276f6519e7

```
universe=set_universe('HS300')
tickers=''
for stk in universe:
    tickers += stk[0:6]+','
tickers=tickers.strip(",")
theme_infos=DataAPI.ThemeThemesGet(ticker=tickers,beginDate='201
41201',endDate='20141230',field=['themeID','themeName'])
#建立一个字典theme_id_name_dic，储存themeID与themeName对应关系
#建立一个字典theme_stk_dic，储存主题与个股的对应关系
theme_id_name_dic={}
theme_stk_dic={}

for i in range(len(theme_infos)):
    ticker = theme_infos.loc[i,'ticker']
    if ticker[0]=='6':
        ticker += '.XSHG'
    else:
        ticker += '.XSHE'
    themeID = theme_infos.loc[i,'themeID']
    themeName = theme_infos.loc[i,'themeName']
    #建立themeID与themeName的对应关系
    if themeID not in theme_id_name_dic:
        theme_id_name_dic[themeID] = themeName
    #建立主题与个股的对应关系，其中的stock均不带后缀
    if themeID not in theme_stk_dic:
        theme_stk_dic[themeID] = [ticker]
    else:
        theme_stk_dic[themeID].append(ticker)

#过滤掉无效的主题
filter_theme_id_name_dic={}
filter_stk_theme_dic={}
filter_theme_stk_dic={}

#训练得到有效主题
for (theme_id,stk_list) in theme_stk_dic.items():
    if len(stk_list)>5:
        filter_theme_id_name_dic[theme_id] = theme_id_name_dic[t
heme_id]
        filter_theme_stk_dic[theme_id] = stk_list

for (theme_id,stk_list) in filter_theme_stk_dic.items():
    for stk in stk_list:
        if stk not in filter_stk_theme_dic:
            filter_stk_theme_dic[stk] = [theme_id]
        else:
            filter_stk_theme_dic[stk].append(theme_id)

filter_universe = filter_stk_theme_dic.keys()
```

```
start = datetime(2013, 6, 23)                # 回测起始时间
end   = datetime(2014, 12, 23)                # 回测结束时间
benchmark = 'HS300'                            # 策略参考标准
universe = filter_universe    # 股票池
capital_base = 100000                       # 起始资金
window=1
hold_days=2    #股票持有时间
sell_stk_list=[]
for i in range(hold_days):
    sell_stk_list.append({})

def initialize(account):                    # 初始化虚拟账户状态
    add_history('hist',window)

def handle_data(account):                   # 每个交易日的买入卖出指令

    theme_increase_raw={}
    theme_increase = {}
    n_theme = 6    #挑选涨得最好的几个主题
    n_bigstk1 = 0    #根据过去涨幅挑选龙头股
    n_bigstk2 = 2    #根据过去成交量大小挑选龙头股
    for stk in account.universe:
        theme_list = filter_stk_theme_dic[stk]
        close_price = account.hist[stk]['closePrice'].iloc[-1]
        pre_close_price = account.hist[stk]['preClosePrice'].ilo
c[0]
        increase_rate = (close_price-pre_close_price)*1.0/pre_cl
ose_price
        sum_turnoverVol = sum(list(account.hist[stk]['turnoverVo
l']))
        num_increase = increase_rate * sum_turnoverVol
        #计算主题涨幅
        for theme_id in theme_list:
            if theme_id not in theme_increase:
                theme_increase_raw[theme_id] = [num_increase,sum
_turnoverVol]
            else:
                theme_increase_raw[theme_id][0] += num_increase
                theme_increase_raw[theme_id][1] += sum_turnoverV
ol

    for (theme_id,theme_increase_list) in theme_increase_raw.ite
ms():
        theme_increase[theme_id] = theme_increase_raw[theme_id][0
]*1.0/theme_increase_raw[theme_id][1]
    #将主题按涨幅排序，排名靠前的挑选出来。
    theme_list_sort = sorted(theme_increase.keys(), key = lambda
 x:theme_increase[x], reverse=True)
    good_theme = theme_list_sort[0:n_theme]
    buy_list = []
    #print 'theme_list_sort:',theme_list_sort
    #print 'good_theme',good_theme
    #挑选涨幅好的主题对应的龙头股，涨的多就是龙头股
    for theme in good_theme:
        #print account.current_date,theme_id_name_dic[theme],theme_increase[theme]
        stk_list = filter_theme_stk_dic[theme]
        stk_increase_dic = {}
        for stk in stk_list:
            if stk not in account.universe: continue
            close_price = account.hist[stk]['closePrice'].iloc[-1]
            pre_close_price = account.hist[stk]['preClosePrice'].iloc[0]
            increase_rate = (close_price-pre_close_price)*1.0/pre_close_price
            sum_turnoverVol = account.hist[stk]['turnoverVol'].iloc[window-1]
            stk_increase_dic[stk] = [increase_rate,sum_turnoverVol]
        stk_list_sort1 = sorted(stk_increase_dic.keys(), key =lambda x:stk_increase_dic[x][0], reverse = True)
        stk_list_sort2 = sorted(stk_increase_dic.keys(), key =lambda x:stk_increase_dic[x][1], reverse = True)
        #print stk_list_sort[0:n_bigstk]
        buy_list += stk_list_sort1[0:n_bigstk1]+stk_list_sort2[0:n_bigstk2]
    #买入股票，并将买入信息写入到卖出股票中
    per_money=account.cash/len(buy_list)
    sell_next_dic = {}
    for stk in buy_list:
        amount = int(per_money/account.hist[stk]['closePrice'].iloc[window-1])
        order(stk,amount)
        sell_next_dic[stk]=amount
    sell_stk_list.append(sell_next_dic)
    sell_today_dic=sell_stk_list.pop(0)
    if sell_today_dic!={}:
        for (stk,amt) in sell_today_dic.items():
            #如果股票今天不能交易,就下一天再卖
            if stk not in account.universe:
                sell_stk_list[0][stk]=amt
            else:
                order(stk,-amt)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
49.6%	36.4%	13.2%	1.00	2.16	21.3%	0.93	15.7%	--

板块异动类

> 来源：https://uqer.io/community/share/54ccf06af9f06c276f651a5a

本代码主要实现以下功能

- 由 `DataAPI.EquIndustryGet` 获得每只个股的所属行业，这里采用的是申万二级分类；
- 根据个股行业获得所有行业的成分股
- 根据成分股的每天涨幅和市值，获得主题的加权涨幅，将其排序，即得到每日涨跌幅最大的行业前十
- 根据成分股出现的涨跌停次数，获得涨跌停比例最大的行业前十
- 根据成分股的换手率，获得换手率最大和最小的行业前十
- 将每个行业所包含的个股，储存到csv文件中，如果对某个行业感兴趣，可以进一步查看其成分股

此处定义了一些函数，使得代码功能更明确

```
def GetIndInfo(universe,field):    #获得行业数据
    num = 100
    count_num = len(universe)/num
    if count_num>0:
        indus_df = pd.DataFrame({})
        for i in range(count_num):
            sub_ind = DataAPI.EquIndustryGet(secID=universe[i*nu
m:(i+1)*num],field=field)
            indus_df = pd.concat([indus_df,sub_ind])
        sub_ind = DataAPI.EquIndustryGet(secID=universe[(i+1)*nu
m:],field=field)
        indus_df = pd.concat([indus_df,sub_ind])
    else:
        indus_df = DataAPI.EquIndustryGet(secID=universe,field=f
ield)
    filed_new = ['secID']+field
    indus_df = indus_df[filed_new]
    return indus_df

def GetMktInfo(secID,beginDate,endDate,field):     #获得市场行情数据
    num = 50
    count_num = len(secID)/num
    if count_num>0:
        MktInfo_df = pd.DataFrame({})
        for i in range(count_num):
            sub_info = DataAPI.MktEqudGet(secID=secID[i*num:(i+1
)*num],beginDate=beginDate,endDate=endDate,field=field)
            MktInfo_df = pd.concat([MktInfo_df,sub_info])
        sub_info = DataAPI.MktEqudGet(secID=secID[(i+1)*num:],be
ginDate=beginDate,endDate=endDate,field=field)
        MktInfo_df = pd.concat([MktInfo_df,sub_info])
    else:
        MktInfo_df = DataAPI.MktEqudGet(secID=secID,beginDate=beginDate,endDate=endDate,field=field)
    return MktInfo_df

def CountTime():
    today = datetime.today()
    cal_date = Date.fromDateTime(today)
    if cal.isBizDay(cal_date):    #如果是交易日，则判断当天是不是在15点前
        today_str = today.strftime("%Y%m%d")
        time1=" 15:05:00"
        ben_time = datetime.strptime(today_str+time1,"%Y%m%d %H:%M:%S")
        if today>ben_time:
            date = today_str
    else:    #如果当天不是交易日，则获得前一个交易日
        cal_wd = cal.adjustDate(cal_date,BizDayConvention.Preceding)    #Date格式
        dtime_wd = cal_wd.toDateTime()    #datetime格式
        date = dtime_wd.strftime("%Y%m%d")
    return date
```

获得个股的行情数据，并以此来计算主题的：涨幅、涨跌停比例、换手率

以下是将结果进行展示

```
from datetime import timedelta
cal = Calendar('China.SSE')

universe = set_universe('A')

indus_df = GetIndInfo(universe=universe,field =['secShortName','industryName2'])
cnt_date = CountTime()    #获得可用的时期
field_mkt = ['preClosePrice','openPrice','highestPrice','lowestPrice','closePrice','turnoverRate','marketValue']
MktInfo_df = GetMktInfo(secID=universe,beginDate=cnt_date,endDate=cnt_date,field=field_mkt)

ind_inc_dic = {}    #记录行业的涨幅
ind_gb_dic = {}     #记录行业的涨跌停数目
ind_turn_dic = {}    #记录行业的换手率
ind_tknm_dic = {}    #记录行业包含的个股

grouped = indus_df.groupby('industryName2')
for name,group in grouped:
    ind_tknm_dic[name] = list(group['secShortName'])

    stk_list = list(group['secID'])
    sub_mkt_info = MktInfo_df[MktInfo_df.secID.isin(stk_list)]
    #计算行业涨跌幅
    sub_mkt_info['inc_rate'] = (sub_mkt_info['closePrice']-sub_mkt_info['preClosePrice'])/sub_mkt_info['preClosePrice']    #获得每个个股的涨跌幅
    ind_inc = (sub_mkt_info['inc_rate']*sub_mkt_info['marketValue']).sum()/sub_mkt_info['marketValue'].sum()    #获得行业的涨跌幅，利用市值加权平均值计算
    ind_inc_dic[name] = ind_inc

    num_good = len(sub_mkt_info[((sub_mkt_info['closePrice']-sub_mkt_info['preClosePrice'])/sub_mkt_info['preClosePrice']).round(2)==0.1])    #涨停的个股
    num_bad  = len(sub_mkt_info[((sub_mkt_info['preClosePrice']-sub_mkt_info['closePrice'])/sub_mkt_info['preClosePrice']).round(2)==0.1])    #跌停的个股
    ind_gb_dic[name] = (num_good-num_bad)*1.0/len(group)

    turnover = sub_mkt_info['turnoverRate'].mean()    #计算行业的平均换手率
    ind_turn_dic[name] = turnover
```

以下是将结果进行展示

```
ind_turn_pd = pd.DataFrame.from_dict(ind_turn_dic,orient='index'
)
ind_turn_pd.rename(columns={0:u'换手率'},inplace=True)
ind_turn_pd = ind_turn_pd.sort(columns=u'换手率',ascending=False)
ind_turn_pd1 = ind_turn_pd.sort(columns=u'换手率',ascending=True)
print cnt_date+'换手率最大的行业前十：'
ind_turn_pd[0:10]

20150130换手率最大的行业前十：
```

	换手率
视听器材	0.046510
基础建设	0.042633
房屋建设	0.036725
计算机应用	0.036130
环保工程及服务	0.035021
营销传播	0.034763
畜禽养殖	0.034093
电力	0.033552
农业综合	0.032450
装修装饰	0.032230

```
print cnt_date+'换手率最小的行业前十：'
ind_turn_pd1[0:10]

20150130换手率最小的行业前十：
```

	换手率
石油开采	0.000900
银行	0.008894
机场	0.009800
航空运输	0.010020
饲料	0.010518
高速公路	0.010583
汽车整车	0.011491
煤炭开采	0.011964
其他交运设备	0.012071
餐饮	0.012150

```
ind_gb_pd = pd.DataFrame.from_dict(ind_gb_dic,orient='index')
ind_gb_pd.rename(columns={0:u'涨跌停比例'},inplace=True)
ind_gb_pd = ind_gb_pd.sort(columns=u'涨跌停比例',ascending=False)
ind_gb_pd1 = ind_gb_pd.sort(columns=u'涨跌停比例',ascending=True)
print cnt_date+'涨停比例最大的行业前十：'
ind_gb_pd[0:10]

20150130涨停比例最大的行业前十：
```

	涨跌停比例
视听器材	0.200000
贸易	0.086957
物流	0.055556
专业工程	0.055556
互联网传媒	0.045455
塑料	0.045455
房地产开发	0.029630
电力	0.017241
家用轻工	0.000000
保险	0.000000

```
print cnt_date+'跌停比例最大的行业前十：'
ind_gb_pd1[0:10]

20150130跌停比例最大的行业前十：
```

	涨跌停比例
旅游综合	-0.066667
计算机设备	-0.051282
电子制造	-0.032258
光学光电子	-0.024390
中药	-0.017857
化学制品	-0.006993
专用设备	0.000000
航运	0.000000
农业综合	0.000000
采掘服务	0.000000

```
ind_inc_pd = pd.DataFrame.from_dict(ind_inc_dic,orient='index')
ind_inc_pd = ind_inc_pd.sort(columns=0,ascending=False)
ind_inc_pd.rename(columns={0:u'涨跌幅'},inplace=True)
ind_inc_pd1 = ind_inc_pd.sort(columns=u'涨跌幅')
print cnt_date+'涨幅最大的行业前十：'
ind_inc_pd[0:10]
```

	涨跌幅
视听器材	0.036822
燃气	0.018286
种植业	0.015623
房地产开发	0.006603
农业综合	0.005786
水务	0.005265
餐饮	0.004425
动物保健	0.004262
饮料制造	0.003649
汽车服务	0.003630

```
print cnt_date+'跌幅最大的行业前十：'
ind_inc_pd1[:10]

20150130跌幅最大的行业前十：
```

	涨跌幅
运输设备	-0.071812
基础建设	-0.049886
多元金融	-0.041817
铁路运输	-0.040228
保险	-0.036876
房屋建设	-0.035251
计算机应用	-0.032599
石油开采	-0.028381
林业	-0.028153
航空运输	-0.025830

将行业包含的个股信息储存到csv文件中，可以进行更细致的查看行业信息

```
ind_tk_pd = pd.DataFrame({})
for ind_nm,tk_list in ind_tknm_dic.items():
    sub_pd = pd.DataFrame(tk_list)
    sub_pd[u'行业名称'] = ind_nm
    ind_tk_pd = pd.concat([ind_tk_pd,sub_pd])
ind_tk_pd.rename(columns={0:u'成分股'},inplace=True)
ind_tk_pd = ind_tk_pd.loc[:,[u'行业名称',u'成分股']]
ind_tk_pd.to_csv('ind_tk.csv',encoding='GBK',index=False)
```

风险因子（离散类）

> 来源：https://uqer.io/community/share/54d2cee9f9f06c276f651a67

本代码用于计算风险因子

- 先根据 `DataAPI.ThemeTickersGet` 得到每个主题相关的个股
- 计算个股在前7天的每天涨跌幅，从而计算主题的每天涨跌幅（市值加权）
- 计算个股前7天的涨跌停次数，计算主题涨跌停比例
- 对每个股票，按照股票市值占主题总市值的比例，计算涨跌幅和涨跌停比例（均为7日），将两个指标进行排名，个股有两个排名得分
- 再取两个排名得分的平均，对个股再次排名

排名越高，波动越大，风险越大

```
datetime.today()

datetime.datetime(2015, 2, 4, 22, 18, 57, 402881)
```

此处定义了几个函数，方便调用

```
def GetThemeInfo(thm_id_list):
#由于ThemeTickersGet对于数据量有限制，一次调用1000个主题数据
    num = 1000                                           #每一次
调取多少个主题的信息
    cnt_num = len(thm_id_list)/num          #一次调取num个主题，要
调用num次
    beginDate = '20140601'                           #开始时间
    endDate = '20150123'                              #结束时间
    if cnt_num>0:
        thm_tk_pd = pd.DataFrame({})
        for i in range(cnt_num):
            info_sub = DataAPI.ThemeTickersGet(beginDate=beginDa
te,endDate=endDate,themeID=thm_id_list[i*num:(i+1)*num])
#获取主题相关的个股
            thm_tk_pd = pd.concat([thm_tk_pd,info_sub])
                                          #将数据连接
        info_sub = DataAPI.ThemeTickersGet(beginDate=beginDate,e
ndDate=endDate,themeID=thm_id_list[(i+1)*num:])
        thm_tk_pd = pd.concat([thm_tk_pd,info_sub])
    else:
        thm_tk_pd = DataAPI.ThemeTickersGet(beginDate=beginDate,
endDate=endDate,themeID=thm_id_list)
    return thm_tk_pd

def GetMktInfo(tk_list,beginDate,endDate,field_mkt):    #获得个股
的日线行情数据
    num = 50
    cnt_num = len(tk_list)/num
    if cnt_num>0:
        tk_mkt_info = pd.DataFrame({})
        for i in range(cnt_num):
            sub_info = DataAPI.MktEqudGet(ticker=tk_list[i*num:(
i+1)*num],beginDate=beginDate,endDate=endDate,field=field_mkt)
            tk_mkt_info = pd.concat([tk_mkt_info,sub_info])
        sub_info = DataAPI.MktEqudGet(ticker=tk_list[(i+1)*num:]
,beginDate=beginDate,endDate=endDate,field=field_mkt)
        tk_mkt_info = pd.concat([tk_mkt_info,sub_info])
    else:
        tk_mkt_info = DataAPI.MktEqudGet(ticker=tk_list,beginDat
e=beginDate,endDate=endDate,field=field_mkt)
    return tk_mkt_info

def GetDate(n):     #获得最近7个交易日的日期
    cal = Calendar("China.SSE")
    today_cal = Date.todaysDate()
    today_dtime = datetime.today()
    if cal.isBizDay(today_cal):     #如果今天是交易日
        today_ymd = today_dtime.strftime("%Y%m%d")
        hms = " 15:05:00"
        ben_time = datetime.strptime(today_ymd+hms,"%Y%m%d %H:%M
:%S")
        if today_dtime>ben_time:     #如果当前时间晚于15：05分，则可以
获取到今日行情数据
            end_date = today_ymd
        else:
            cal_wd = cal.advanceDate(today_cal, '-1B', BizDayCon
vention.Preceding)     #获得前一个工作日Date格式
            end_date = cal_wd.toISO().replace('-','')     #转换成
字符串格式‘20140102’
    else:
        cal_wd = cal.advanceDate(today_cal, '-1B', BizDayConvent
ion.Preceding)     #获得前一个工作日Date格式
        end_date = cal_wd.toISO().replace('-','')     #转换成字符串
格式‘20140102’

    end_date_cal = Date.parseISO('-'.join([end_date[0:4],end_dat
e[4:6],end_date[6:8]]))     #更改日期格式为“2014-03-02”
    prd = '-'+str(n-1)+'B'     #起始日期和终止日期间隔的天数
    begin_date_cal = cal.advanceDate(end_date_cal, prd , BizDayC
onvention.Preceding)     #获得6天前的工作日
    begin_date = begin_date_cal.toISO().replace('-','')
    return begin_date,end_date
```

读取主题id文件，先对个股和主题进行筛选，然后获得个股的行情数据

```
#Main
import pandas as pd

f1 = read('20140601_20150203theme_list.txt')                 #从这个文档中读取所有的主题id
thm_id_list = f1.split(',')

thm_tk_pd = GetThemeInfo(thm_id_list=thm_id_list)     #获得主题对应个股的信息
thm_tk_pd = thm_tk_pd[(thm_tk_pd['ticker'].str.len()==6) & (thm_tk_pd['ticker'].apply(lambda x:x[0]=='0' or x[0]=='6'))]     #过滤港股和新三板，因为拿不到行情数据

grouped_thmid = thm_tk_pd.groupby('themeID')     #根据主题id分类，得到每个主题对应的个股
###对主题进行过滤如果该主题所包含的个股《5，则舍弃
fld_thmid_list = []
for name,group in grouped_thmid:
    if len(group)>=5:
        fld_thmid_list.append(name)
thm_tk_pd = thm_tk_pd[thm_tk_pd['themeID'].isin(fld_thmid_list)]


ThmId_Nm_dic = dict(zip(thm_tk_pd['themeID'],thm_tk_pd['themeName']))     #获得主题id与主题名称的对应
TkId_Nm_dic = dict(zip(thm_tk_pd['ticker'],thm_tk_pd['secShortName']))     #获得个股id与个股名称的对应
thm_tk_pd = thm_tk_pd[['themeID','ticker']]
tk_list = list(set(thm_tk_pd['ticker']))      #获得所有的个股
n_prd =7
beginDate,endDate = GetDate(n_prd)     #获取n_prd个交易日的具体日期
field_mkt = ['ticker','openPrice','closePrice','highestPrice','lowestPrice','marketValue','preClosePrice ']

tk_mktinfo_pd = GetMktInfo(tk_list,beginDate,endDate,field_mkt)    #获得所有个股的行情数据
tk_mktinfo_pd['return'] = (tk_mktinfo_pd['closePrice']-tk_mktinfo_pd['preClosePrice'])/tk_mktinfo_pd['preClosePrice']     #计算所有个股每天的涨跌幅
```

计算主题的涨跌幅（绝对值）和涨跌停比例

```
grouped_thmid = thm_tk_pd.groupby('themeID')    #根据主题id分类，得到每个主题对应的个股
grouped_tkid = thm_tk_pd.groupby('ticker')    #根据ticker分类，得到每个个股对应的主题
thm_rtn_dic, thm_gb_dic, thm_mkv_dic = {},{},{}    #主题的日涨幅，主题的日涨跌停比例，主题的市值
#获得主题的日收益的绝对值的平均
for thm,group_thm in grouped_thmid:
    sub_tk_list = list(group_thm['ticker'])
    sub_tk_mkt_pd = tk_mktinfo_pd[tk_mktinfo_pd['ticker'].isin(sub_tk_list)]    #获得该主题下个股的行情数据
    thm_rtn =  (sub_tk_mkt_pd['marketValue']*abs(sub_tk_mkt_pd['return'])).sum()/sub_tk_mkt_pd['marketValue'].sum()    #计算主题在这7天的平均每天绝对收益
    thm_rtn_dic[thm] = thm_rtn
    thm_mkv_dic[thm] = sub_tk_mkt_pd['marketValue'].sum()    #记录每个主题的市值（7天的和）
    num_gb = len(sub_tk_mkt_pd[(abs((sub_tk_mkt_pd['closePrice']-sub_tk_mkt_pd['preClosePrice']))/sub_tk_mkt_pd['preClosePrice']).round(2)==0.1])    #涨跌停的个股数目
    thm_gb_dic[thm] = num_gb/n_prd    #主题涨跌停比例7日均值
```

由主题涨跌幅和涨跌停比例，计算个股的涨跌幅和涨跌停比例

```
tk_inc_gb_dic = {}    #由主题计算的个股的涨幅和涨跌停比例
for tk,group_tk in grouped_tkid:
    tk_mkv = tk_mktinfo_pd['marketValue'][tk_mktinfo_pd['ticker']==tk].sum()    #得到个股市值（7天的和）
    thm_list = group_tk['themeID']
    inc,gb_ratio = 0,0
    for thm in thm_list:
        pro = tk_mkv/thm_mkv_dic[thm]    #个股占该主题的比例
        inc += thm_rtn_dic[thm]*pro
        gb_ratio += thm_gb_dic[thm]*pro
    tk_inc_gb_dic[tk] = (inc,gb_ratio)    #记录个股的涨幅和涨跌停比例
```

根据个股的涨跌幅和涨跌停比例进行排名，再将这两个排名进行平均，再排名

```
sort1 = sorted(tk_inc_gb_dic.keys(), key = lambda x:tk_inc_gb_dic[x][0], reverse=True)    #根据个股的涨幅排名，涨幅大的排名在前
sort2 = sorted(tk_inc_gb_dic.keys(), key = lambda x:tk_inc_gb_dic[x][1], reverse=True)    #根据个股的涨跌停比例排名，涨跌停比例高的排名在前
rank = lambda x:(sort1.index(x)+sort2.index(x))*1.0/2+1
id2name = lambda x:TkId_Nm_dic[x]
df = pd.DataFrame({'ticker':tk_list})
df['name'] = pd.Series(map(id2name,tk_list))
df['ranking_score'] = pd.Series(map(rank,tk_list))
df_sort = df.sort(columns=['ranking_score'],ascending = True)
df_sort.reset_index(inplace=True,drop=True)
print "最近个股风险因子排名："
df_sort
```

```
datetime.today()

datetime.datetime(2015, 2, 4, 22, 19, 15, 638752)
```

8.5 龙头轮动

Competitive Securities

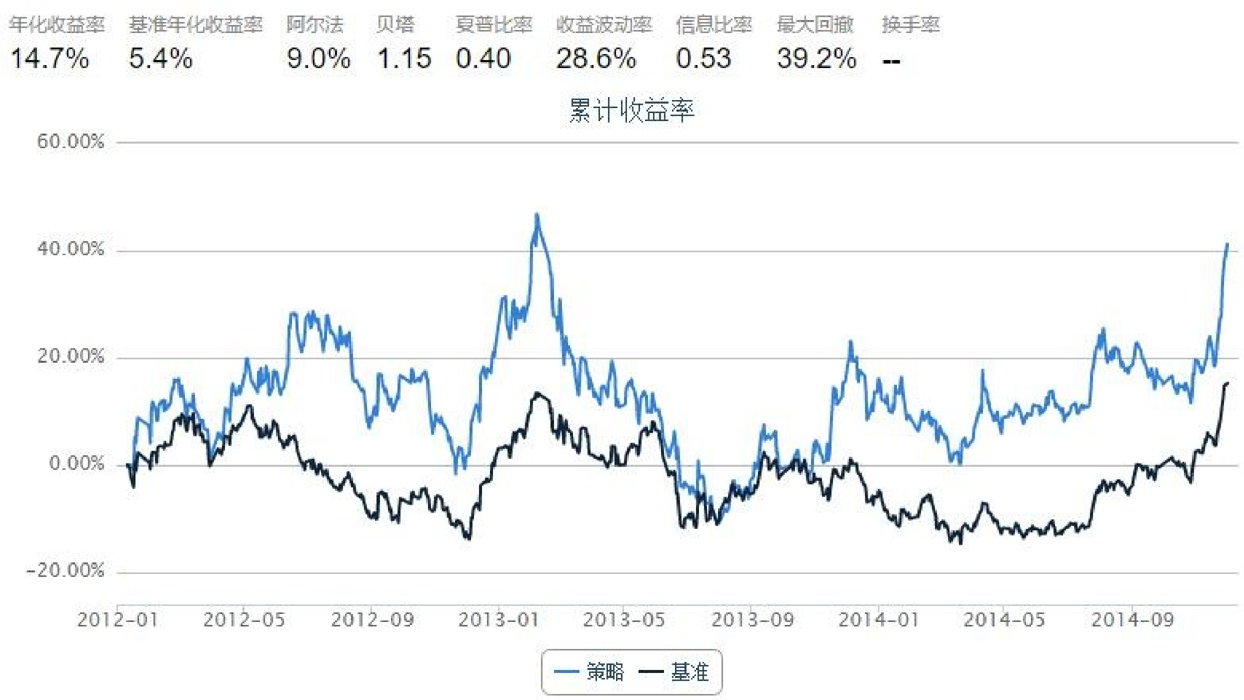
> 来源：https://uqer.io/community/share/54b5c373f9f06c276f651a18

策略实现：

- 计算三只同一行业股票过去4天内前3天的平均成交价（VWAP），这里选用的是中国平安 (601318.XSHG)、中国太保 (601601.XSHG)和中国人寿 (601628.XSHG)
- 当某两只股票的价格低于 `0.995 * VWAP` ，同时另一只股票价格高于VWAP时，买入后者
- 当某两只股票的价格高于 `1.025 * VWAP` ，同时另一只股票价格低于VWAP时，清空后者

```
import pandas as pd
import numpy as np
from datetime   import datetime
from matplotlib import pylab

import quartz
import quartz.backtest as qb
import quartz.performance as qp
from quartz.api import *
```

```
"Competitive Securities"

start = pd.datetime(2012, 1, 1)
end   = pd.datetime(2014, 12, 1)
bm = 'HS300'
universe = ['601601.XSHG', '601318.XSHG', '601628.XSHG']
csvs = []

capital_base = 5000
window = 4
threshold_dn = 0.995
threshold_up = 1.025
refresh_rate = 4

def initialize(account):
    account.amount = 1000
    account.universe = universe
    add_history('hist', window)

def handle_data(account):
    vwap3, price = {}, {}
    for stk in account.universe:
        if stk not in account.hist:
            continue

        vwap3[stk] = sum(account.hist[stk]['turnoverValue'][:3])
/sum(account.hist[stk]['turnoverVol'][:3])
        price[stk] = account.hist[stk].iloc[window-1,:]['closePr
ice']

    if len(vwap3)!=3:
        return

    stk_0 = account.universe[0]
    stk_1 = account.universe[1]
    stk_2 = account.universe[2]

    if price[stk_1] <= threshold_dn * vwap3[stk_1] and price[stk
_2] <= threshold_dn * vwap3[stk_2] and price[stk_0] > vwap3[stk_
0]:
        order(stk_0, account.amount)
    if price[stk_2] <= threshold_dn * vwap3[stk_2] and price[stk
_0] <= threshold_dn * vwap3[stk_0] and price[stk_1] > vwap3[stk_
1]:
        order(stk_1, account.amount)
    if price[stk_0] <= threshold_dn * vwap3[stk_0] and price[stk
_1] <= threshold_dn * vwap3[stk_1] and price[stk_2] > vwap3[stk_
2]:
        order(stk_2, account.amount)

    if price[stk_1] >= threshold_up * vwap3[stk_1] and price[stk
_2] >= threshold_up * vwap3[stk_2] and price[stk_0] < vwap3[stk_
0]:
        order_to(stk_0, 0)
    if price[stk_2] >= threshold_up * vwap3[stk_2] and price[stk
_0] >= threshold_up * vwap3[stk_0] and price[stk_1] < vwap3[stk_
1]:
        order_to(stk_1, 0)
    if price[stk_0] >= threshold_up * vwap3[stk_0] and price[stk
_1] >= threshold_up * vwap3[stk_1] and price[stk_2] < vwap3[stk_
2]:
        order_to(stk_2, 0)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
14.7%	5.4%	9.0%	1.15	0.40	28.6%	0.53	39.2%	--

```
perf = qp.perf_parse(bt)
out_keys = ['annualized_return', 'volatility', 'information',
            'sharpe', 'max_drawdown', 'alpha', 'beta']

for k in out_keys:
    print '%s: %s' % (k, perf[k])
```

```
annualized_return: 0.14708285
volatility: 0.285959506628
information: 0.525131029268
sharpe: 0.395275720443
max_drawdown: 0.391931712536
alpha: 0.089663482291
beta: 1.15117691695
```

```
perf['cumulative_return'].plot()
perf['benchmark_cumulative_return'].plot()
pylab.legend(['current_strategy','HS300'])
```

```
<matplotlib.legend.Legend at 0x55bf290>
```

Market Competitiveness

> 来源：https://uqer.io/community/share/54b5c2f1f9f06c276f651a17

来一个奇葩无厘头的市场竞争策略

策略思路

某一行业的几大龙头股票，在稳定时期此消彼长

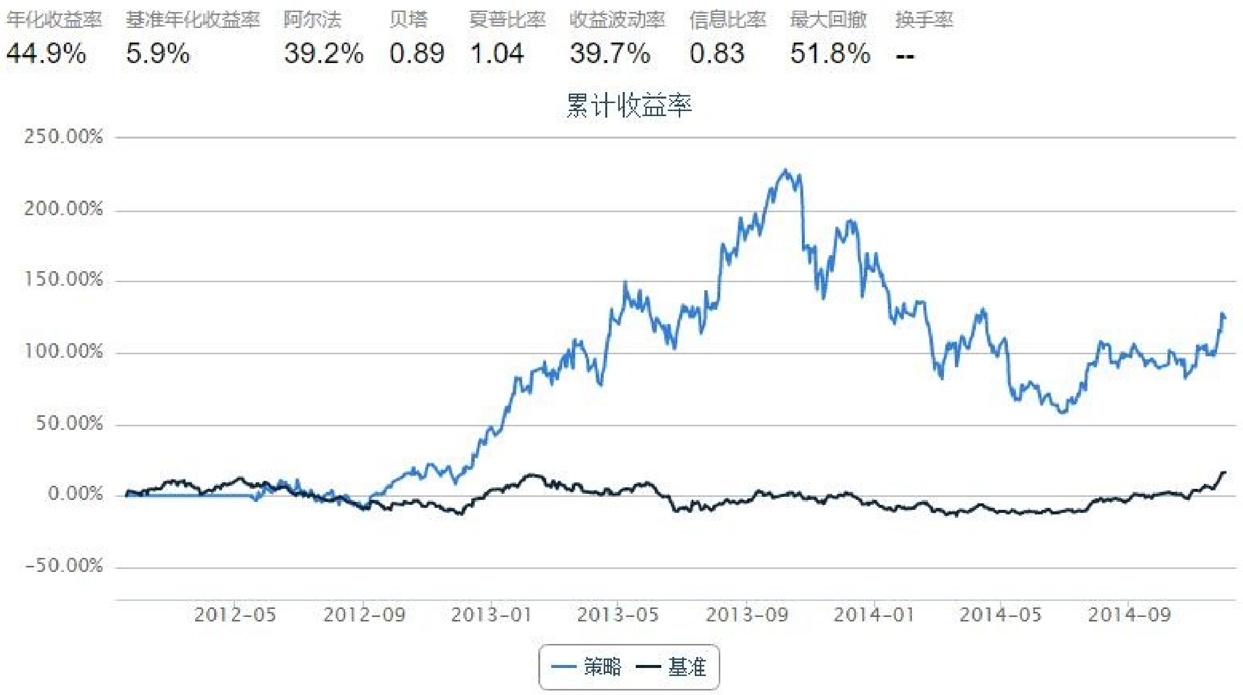
策略实现

- 股票池：选择一行业内的流动性比较好的龙头股票；例如三家自助品牌汽车，长安、比亚迪和长城，以下按照三只股票情况讨论
- 观察某一天时，股票价格和该股票在过去几天内平均值的关系
- 如果两只股票下跌，则预测另一只股票上涨；如果两只股票上涨，则预测另一只股票下跌
- 如果某天三只股票中的两只较其平均值有较大幅度下跌，而另一只股票较其平均值比较稳定不变，则买入后面这只比较稳定的股票
- 如果某天三只股票中的两只较平均值有较大幅度上涨，而另一只股票较其平均值比较稳定不变，则卖出后面这只比较稳定的股票

```
import quartz
import quartz.backtest    as qb
import quartz.performance as qp
from   quartz.api         import *

import pandas as pd
import numpy  as np
from datetime   import datetime
from matplotlib import pylab
```

```
start = datetime(2012, 1, 1)
end = datetime(2014, 12, 1)
benchmark = 'HS300'
universe = ['000625.XSHE', # 长安汽车
            '002594.XSHE', # 比亚迪汽车
            '601633.XSHG'  # 长城汽车
            ]

capital_base = 1000000
```

```
refresh_rate = 5
window = 10

def initialize(account):
    account.amount = 100000
    account.universe = universe
    add_history('hist', window)

def handle_data(account):

    stk_0 = universe[0]
    stk_1 = universe[1]
    stk_2 = universe[2]

    prices_0 = account.hist[stk_0]['closePrice']
    prices_1 = account.hist[stk_1]['closePrice']
    prices_2 = account.hist[stk_2]['closePrice']

    mu_0 = prices_0.mean()
    mu_1 = prices_1.mean()
    mu_2 = prices_2.mean()

    # 两只下跌较大幅度，一只较稳定，买入较稳定这只股票
    if prices_0[-1] > mu_0 and prices_1[-1] < 0.975 * mu_1 and prices_2[-1] < 0.975 * mu_2:
        order(stk_0, account.amount)
    if prices_1[-1] > mu_1 and prices_2[-1] < 0.975 * mu_2 and prices_0[-1] < 0.975 * mu_0:
        order(stk_1, account.amount)
    if prices_2[-1] > mu_2 and prices_0[-1] < 0.975 * mu_0 and prices_1[-1] < 0.975 * mu_1:
        order(stk_2, account.amount)

    # 两只上涨较大幅度，一只较稳定，卖出较稳定这只股票
    if prices_0[-1] < mu_0 and prices_1[-1] > 1.025 * mu_1 and prices_2[-1] > 1.025 * mu_2:
        order_to(stk_0, 0)
    if prices_1[-1] < mu_1 and prices_0[-1] > 1.025 * mu_0 and prices_2[-1] > 1.025 * mu_2:
        order_to(stk_1, 0)
    if prices_2[-1] < mu_2 and prices_0[-1] > 1.025 * mu_0 and prices_1[-1] > 1.025 * mu_1:
        order_to(stk_2, 0)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
44.9%	5.9%	39.2%	0.89	1.04	39.7%	0.83	51.8%	--

```
bt
```

	tradeDate	cash	stock_position	portfolio_value	
0	2012-01-18	1000000.00000	{}	1000000.00000	
1	2012-01-19	1000000.00000	{}	1000000.00000	
2	2012-01-20	1000000.00000	{}	1000000.00000	
3	2012-01-30	1000000.00000	{}	1000000.00000	
4	2012-01-31	1000000.00000	{}	1000000.00000	
5	2012-02-01	1000000.00000	{}	1000000.00000	
6	2012-02-02	1000000.00000	{}	1000000.00000	
7	2012-02-03	1000000.00000	{}	1000000.00000	
8	2012-02-06	1000000.00000	{}	1000000.00000	
9	2012-02-	1000000.00000	{}	1000000.00000	

9	07	1000000.00000	{}	1000000.00000
10	2012-02-08	1000000.00000	{}	1000000.00000
11	2012-02-09	1000000.00000	{}	1000000.00000
12	2012-02-10	1000000.00000	{}	1000000.00000
13	2012-02-13	1000000.00000	{}	1000000.00000
14	2012-02-14	1000000.00000	{}	1000000.00000
15	2012-02-15	1000000.00000	{}	1000000.00000
16	2012-02-16	1000000.00000	{}	1000000.00000
17	2012-02-17	1000000.00000	{}	1000000.00000
18	2012-02-20	1000000.00000	{}	1000000.00000
19	2012-02-21	1000000.00000	{}	1000000.00000
20	2012-02-22	1000000.00000	{}	1000000.00000
21	2012-02-23	1000000.00000	{}	1000000.00000
22	2012-02-24	1000000.00000	{}	1000000.00000
23	2012-02-27	1000000.00000	{}	1000000.00000
24	2012-02-28	1000000.00000	{}	1000000.00000
25	2012-02-29	1000000.00000	{}	1000000.00000
26	2012-03-01	1000000.00000	{}	1000000.00000
27	2012-03-	1000000.00000	{}	1000000.00000

28	2012-03-05	1000000.00000	{}	1000000.00000
29	2012-03-06	1000000.00000	{}	1000000.00000
...	...	...	...	...
664	2014-10-21	1.56401	{u'000625.XSHE': 1.0, u'601633.XSHG': 62476.0}	1913031.23401
665	2014-10-22	1.56401	{u'000625.XSHE': 1.0, u'601633.XSHG': 62476.0}	1933648.47401
666	2014-10-23	1.56401	{u'000625.XSHE': 1.0, u'601633.XSHG': 62476.0}	1953640.67401
667	2014-10-24	1.56401	{u'000625.XSHE': 1.0, u'601633.XSHG': 62476.0}	1823065.79401
668	2014-10-27	1.56401	{u'000625.XSHE': 1.0, u'601633.XSHG': 62476.0}	1859302.04401
669	2014-10-28	1.56401	{u'000625.XSHE': 1.0, u'601633.XSHG': 62476.0}	1863675.44401
670	2014-10-29	1.56401	{u'000625.XSHE': 1.0, u'601633.XSHG': 62476.0}	1871797.24401
671	2014-10-30	1.56401	{u'000625.XSHE': 1.0, u'601633.XSHG': 62476.0}	1883042.97401
672	2014-10-31	1.56401	{u'000625.XSHE': 1.0, u'601633.XSHG':	1913656.09401

	31		u'601633.XSHG': 62476.0}	
673	2014-11-03	1.56401	{u'000625.XSHE': 1.0, u'601633.XSHG': 62476.0}	1902410.64401
674	2014-11-04	1.56401	{u'000625.XSHE': 1.0, u'601633.XSHG': 62476.0}	1964886.42401
675	2014-11-05	1.56401	{u'000625.XSHE': 1.0, u'601633.XSHG': 62476.0}	2049228.79401
676	2014-11-06	1.56401	{u'000625.XSHE': 1.0, u'601633.XSHG': 62476.0}	2020489.70401
677	2014-11-07	1.56401	{u'000625.XSHE': 1.0, u'601633.XSHG': 62476.0}	2027362.02401
678	2014-11-10	1.56401	{u'000625.XSHE': 1.0, u'601633.XSHG': 62476.0}	2043606.06401
679	2014-11-11	1.56401	{u'000625.XSHE': 1.0, u'601633.XSHG': 62476.0}	2022988.64401
680	2014-11-12	1.56401	{u'000625.XSHE': 1.0, u'601633.XSHG': 62476.0}	2049228.91401
681	2014-11-13	1.56401	{u'000625.XSHE': 1.0, u'601633.XSHG': 62476.0}	2054226.61401
682	2014-11-14	1.56401	{u'000625.XSHE': 1.0, u'601633.XSHG':	1987377.18401

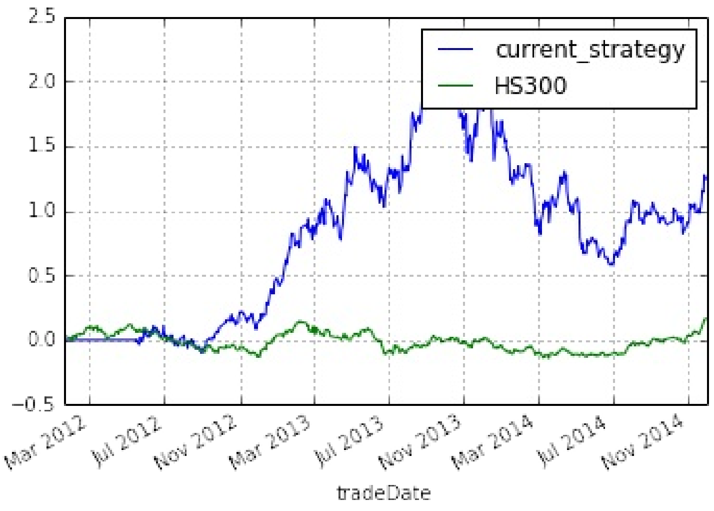
683	2014-11-17	1.56401	{u'000625.XSHE': 1.0, u'601633.XSHG': 62476.0}	1988626.85401
684	2014-11-18	1.56401	{u'000625.XSHE': 1.0, u'601633.XSHG': 62476.0}	2006744.97401
685	2014-11-19	1.56401	{u'000625.XSHE': 1.0, u'601633.XSHG': 62476.0}	2014241.95401
686	2014-11-20	1.56401	{u'000625.XSHE': 1.0, u'601633.XSHG': 62476.0}	1980504.81401
687	2014-11-21	1.56401	{u'000625.XSHE': 1.0, u'601633.XSHG': 62476.0}	1989251.55401
688	2014-11-24	1.56401	{u'000625.XSHE': 1.0, u'601633.XSHG': 62476.0}	2085464.99401
689	2014-11-25	1.56401	{u'000625.XSHE': 1.0, u'601633.XSHG': 62476.0}	2156687.78401
690	2014-11-26	1.56401	{u'000625.XSHE': 1.0, u'601633.XSHG': 62476.0}	2142942.92401
691	2014-11-27	1.56401	{u'000625.XSHE': 1.0, u'601633.XSHG': 62476.0}	2146691.26401
692	2014-11-28	1.56401	{u'000625.XSHE': 1.0, u'601633.XSHG': 62476.0}	2276016.94401
			{u'000625.XSHE':	

693	2014-12-01	1.56401	{u'000625.XSHE': 1.0, u'601633.XSHG': 62476.0}	2245404.03401	

```
694 rows × 6 columns
```

```
perf = qp.perf_parse(bt)
out_keys = ['annualized_return', 'volatility', 'information',
            'sharpe', 'max_drawdown', 'alpha', 'beta']

for k in out_keys:
    print '%s: %s' % (k, perf[k])

annualized_return: 0.448632577093
volatility: 0.397466535866
information: 0.825863671828
sharpe: 1.04326663926
max_drawdown: 0.518092986656
alpha: 0.392363999248
beta: 0.886220585368
```

```
perf['cumulative_return'].plot()
perf['benchmark_cumulative_return'].plot()
pylab.legend(['current_strategy','HS300'])

<matplotlib.legend.Legend at 0x4e27c50>
```



主题龙头类

> 来源：https://uqer.io/community/share/54c3348ff9f06c276f651a49

本代码用于挖掘主题的龙头股

- 先由通联提供的有关主题的API获得所有主题ID，储存为文档 `20140601_20150123theme_list.txt`
- 由API函数 `DataAPI.ThemeTickersGet` 获得所有各个主题对应个股
- 由相关个股的日涨幅和市值，计算主题的每天收益
- 滚动计算主题5天的涨幅，找到主题涨幅最高的时间区间
- 在这个时间区间内，计算主题相关个股的涨幅
- 找到涨幅最高的个股，即为该主题的龙头股

```
datetime.today()

datetime.datetime(2015, 1, 24, 13, 28, 42, 799154)
```

读取主题id文件，获得所有主题的相关个股，储存在 `info` 中

```
f1 = read('20140601_20150123theme_list.txt')                        #
从这个文档中读取所有的主题id
themeId_list = f1.split(',')
tk2id = lambda x:x+'.XSHG' if x[0]=='6' else x+'.XSHE'

#由于ThemeTickersGet对于数据量有限制，一次调用1000个主题数据
num_up = 1000                                                   #每一次调
取多少个主题的信息
thm_tk_dic = {}                                                    #储存
每个主题包含的个股
tk_list = set([])
#tk_list储存了所有相关个股
num = len(themeId_list)/num_up                  #一次调取num_up个主题，要调
用num次
beginDate = '20140601'                                      #开始时间
endDate = '20150123'                                         #结束时间
if num>0:
    info = pd.DataFrame({})
    for i in range(num):
        info_sub = DataAPI.ThemeTickersGet(beginDate=beginDate,e
ndDate=endDate,themeID=themeId_list[i*num_up:(i+1)*num_up])
   #获取主题相关的个股
        info = pd.concat([info,info_sub])
                                                #将数据连接
    info_sub = DataAPI.ThemeTickersGet(beginDate=beginDate,endDa
te=endDate,themeID=themeId_list[(i+1)*num_up:])
    info = pd.concat([info,info_sub])
else:
    info = DataAPI.ThemeTickersGet(beginDate=beginDate,endDate=e
ndDate,themeID=themeId_list)
```

将主题与个股对应， `thm_tk_dic` 储存了每个主题对应的个股， `key` 是主题名称， `value` 是与主题相关的个股列表

```
info = info[['themeName','ticker','secShortName']]
#只取这几列数据
group_info = info.groupby('themeName')
#根据主题名称分类
for theme,group in group_info:
    theme_tk = group['ticker'].tolist()
    if len(theme_tk[0])!=6:
        continue
    if len(theme_tk)>10:
        thm_tk_dic[theme] = theme_tk
       #获得某个主题下相关的个股
        tk_list |= set(theme_tk)
                   #所有的个股
```

利用 `DataAPI.SecIDGet` 获得个股对应的名称，便于最后展示

```
#获得个股代码与名称的对应
tk_list = list(tk_list)
num_name = len(tk_list)/1000
if num_name>0:
    tk_name_pd = pd.DataFrame({})
    for i in range(num_name):
        sub = DataAPI.SecIDGet(ticker=tk_list[1000*i:1000*(i+1)]
,field='secShortName')
#获取证券简称
        tk_name_pd = pd.concat([tk_name_pd,sub])
    sub = DataAPI.SecIDGet(ticker=tk_list[1000*(i+1):],field='se
cShortName')
    tk_name_pd = pd.concat([tk_name_pd,sub])
else:
    tk_name_pd = DataAPI.SecIDGet(ticker=tk_list,field='secShort
Name')

tk_name_dic = dict(zip(tk_name_pd.ticker,tk_name_pd.secShortName
))                                                         #获得个
股代码与名字对应的字典，便于最后展示
```

获得所有股票在一段日期内的日行情数据，便于计算个股每日涨幅和主题涨幅

```
#由于API对访问量有限制，每次只能调取50个股票的日线数据，故采用限制每次调用次
数，循环调用的方法
len_stk = 50                      #每次调用len_stk只个股
num_stk = len(tk_list)/len_stk   #要调用num_stk次
#获得tk_list中所有个股的日线信息
if num_stk>0:
    mkt_stk_info = pd.DataFrame({})
    for i in range(num_stk):
        info = DataAPI.MktEqudGet(ticker=tk_list[i*len_stk:(i+1)
*len_stk],beginDate=beginDate,endDate=endDate,field=['ticker','p
reClosePrice','closePrice','marketValue'])
        mkt_stk_info = pd.concat([mkt_stk_info,info])
    info = DataAPI.MktEqudGet(ticker=tk_list[(i+1)*len_stk:],beg
inDate=beginDate,endDate=endDate,field=['ticker','preClosePrice',
'closePrice','marketValue'])
    mkt_stk_info = pd.concat([mkt_stk_info,info])

else:
    mkt_stk_info = DataAPI.MktEqudGet(ticker=tk_list,beginDate=b
eginDate,endDate=endDate,field=['ticker','preClosePrice','closeP
rice','marketValue'])
mkt_stk_info = mkt_stk_info[['ticker','tradeDate','preClosePrice'
,'closePrice','marketValue']]
```

计算主题每日涨幅

```
thm_date_inc_dic = {}
                                                #记录了每个主题每天的涨
幅，key是日期，value是主题涨幅
for key in thm_tk_dic.keys():
    thm_date_inc_dic[key] = {}
for theme,stocks in thm_tk_dic.items():
    mkt_info = mkt_stk_info[mkt_stk_info['ticker'].isin(stocks)
]
    mkt_info['increase'] = (mkt_info['closePrice']-mkt_info['pre
ClosePrice'])/mkt_info['preClosePrice']     #计算主题涨幅
    gp_date = mkt_info.groupby('tradeDate')
    for date,group in gp_date:
        thm_inc = sum(group['marketValue']*group['increase'])/su
m(group['marketValue'])                         #某天的主题收益
        thm_date_inc_dic[theme][date] = thm_inc
```

计算主题滚动5日的涨幅之和，找到主题涨幅最大的时间区间

```
thm_date_inc_pd = pd.DataFrame(thm_date_inc_dic)
                     #生成dataframe，index是日期，columns是主题名称
window = 5

  #统计5天的主题收益之和
thm_5d_inc_pd = pd.rolling_sum(thm_date_inc_pd,window=window)
      #计算滚动和
id_max_list_begin = thm_5d_inc_pd.dropna().values.argmax(axis=0)
           #由于最初的window-1行值为NA，舍弃之后得到的下标便是最大和的开
始下标

pl2date = lambda x,:list(thm_5d_inc_pd.index)[x:x+window]
                    #由开始下标获得这段时间区间，即获得[2014-06-03,201
4-06-04,...]这样的列表
date_list = map(pl2date,id_max_list_begin)
max_date_periods = {}
                                                            #储存每个
主题获得最高收益的时间区间
for i in range(len(date_list)):
    theme = thm_date_inc_pd.columns[i]
                                              #主题名称
    date = date_list[i]

#对应的时间区间
    max_date_periods[theme] = date
```

计算在上述时间区间内个股的涨幅，涨幅最大的即为该主题的龙头股

```
thm_leadStk_dic = {}


    #记录每个主题的龙头股
for theme,stock_list in thm_tk_dic.items():
    date_list = max_date_periods[theme]
    flt_thm_stk_info = mkt_stk_info[(mkt_stk_info['ticker'].isin
(stock_list)) & (mkt_stk_info['tradeDate'].isin(date_list))]
#获取这些个股在这段时期内的日线数据
    grouped = flt_thm_stk_info.groupby('ticker')
    stk_inc_dic = {}
    for stk,group in grouped:
        stk_inc = (group['closePrice'].iloc[-1]-group['preCloseP
rice'].iloc[0])/group['preClosePrice'].iloc[0]
     #获取个股在这段时间内的收益
        stk_inc_dic[stk] = stk_inc

    thm_leadStk_dic[theme] = sorted(stk_inc_dic.keys(),key=lambda
 x:stk_inc_dic[x],reverse = True)[0]                          #排
序，获得该主题的龙头股
```

将主题和龙头股写成 `dataframe` 形式，便于展示

```
thm_leadStk_pd = pd.DataFrame.from_dict(thm_leadStk_dic,orient='
index').reset_index()                         #由字典生成dataframe
thm_leadStk_pd.rename(columns={0:'ticker'},inplace=True)

        #重命名，便于下一步merge
lead_tk_list = list(thm_leadStk_pd['ticker'])
tk2nm = lambda x:tk_name_dic[x]
#tk2nm = lambda x:
lead_name_list = map(tk2nm,lead_tk_list)
name_pd = pd.DataFrame({'shortname':lead_name_list})
answer = pd.concat([thm_leadStk_pd,name_pd],axis=1)
answer.rename(columns={'index':u'主题名称','ticker':u'个股代码','sh
ortname':u'个股简称'},inplace=True)   #重命名
answer
```

	主题名称	个股代码	个股简称
0	金融机具股	601818	光大银行
1	银联	601818	光大银行
2	公路运输股	601939	建设银行
3	小额贷款股	601818	光大银行
4	LBS股	600118	中国卫星

5	智能电表	300085	银之杰
6	国资整合	601299	中国北车
7	硝酸铵股	002217	*ST合泰
8	镍氢电池股	600549	厦门钨业
9	国产手机	600050	中国联通
10	浦东新区	601901	方正证券
11	特高压	000709	河北钢铁
12	特高压股	300265	通光线缆
13	数字地图	600717	天津港
14	白色石墨烯股	000009	中国宝安
15	淘宝	000002	万科A
16	电子支付	002095	生意宝
17	PE(化工)	600028	中国石化
18	核电主设备股	300411	金盾股份
19	兽药	000826	桑德环境
20	沪港通股	601099	太平洋
21	甲基叔丁基醚股	000151	中成股份
22	体育文化	601901	方正证券
23	品牌服装	002503	搜于特
24	多晶硅股	600151	航天机电
25	丁二醇股	000151	中成股份
26	TVOS股	300079	数码视讯
27	光电子材料	002261	拓维信息
28	珠海航展	600990	四创电子
29	农用机械股	300159	新研股份
...	...	...	...
1301	德州本地股	601106	中国一重
1302	制冷剂股	000550	江铃汽车
1303	微信股	600109	国金证券
1304	保健品	600530	交大昂立

1305	抗寒	600188	兖州煤业
1306	社保股	002501	利源精制
1307	神舟十号	601988	中国银行
1308	WAPI	002439	启明星辰
1309	光电子材料股	002261	拓维信息
1310	广东自贸区股	600185	格力地产
1311	空调股	300411	金盾股份
1312	德州本地	601106	中国一重
1313	雅安地震	002314	雅致股份
1314	新疆建设股	000562	宏源证券
1315	甲醇股	600188	兖州煤业
1316	陕甘宁区	600185	格力地产
1317	电解铝	601600	中国铝业
1318	电子信息股	000901	航天科技
1319	油气	601857	中国石油
1320	机床	603011	合锻股份
1321	人工智能	002230	科大讯飞
1322	机制纸	000488	晨鸣纸业
1323	铝股	600595	中孚实业
1324	金属新材料股	600888	新疆众和
1325	指纹识别	300248	新开普
1326	小米概念	000333	美的集团
1327	地沟油检测	600028	中国石化
1328	江苏沿海地区	000425	徐工机械
1329	电线电缆	002692	远程电缆
1330	风电股	600163	福建南纸

```
1331 rows × 3 columns
```

```
datetime.today()

datetime.datetime(2015, 1, 24, 13, 33, 44, 786650)
```

九 组合投资

9.1 指数跟踪 · [策略] 指数跟踪低成本建仓策略

> 来源：https://uqer.io/community/share/56632448f9f06c6c8a91b337

指数跟踪

指数追踪通常是指利用某个股票组合复制某一现实指数或者虚拟指数的市场表现，来获取与指数相近的收益，试图最小化跟踪误差。为什么要跟踪指数呢？那要从指数广泛的用途说起：1）很多大型的公募基金都设有专门的指数投资部，通过跟踪特定的指数标的发行ETF、分级基金等产品，以供投资者使用；2）目前在A股市场上做期限套利者通常需要建仓SH50、HS300、ZZ500等指数的现货；3）有些投资者热衷于行业投资，因此其投资标的不是个股而是行业指数；4）近年流行的FOF，是以基金产品作为投资标的，构建基金产品的组合，基金产品也可以认为是代表基金经理投资特定的指数。如果直接购买指数产品有限制或者需要较低成本的建仓指数，那就需要一些策略啦！

通常指数跟踪分为两类：完全复制法和部分复制法。顾名思义，完全复制法即为尽可能完全按照指数的成分股和配比权重来调整组合，使之尽可能的跟踪所选的标的。这种方法对于一般的投资者来讲通常成本和操作难度比较大，这种方法通常仅限于大型公募基金指数部门所使用。部分复制法即为选择某个不同于指数成分股和权重的组合（通常是指数的子集）来尽可能的去跟踪指数。这就需要采用一些优化策略去做选股和配权重。本文参考了文献“Carrier Portfolios.--Steven Kusiak”中的思想，意在研究一种部分复制指数的方法。

Carrier Portfolios

1）模型： 指数本质上是由其成分股按照特定的权重线性组合而成，其成分股张成一个线性空间，每个成分股的权重即相当于指数在每个维度上的长度。我们希望从这个线性空间中找到一个子空间，并根据子空间的基去配以合适的权重，使得配比后的结果能够最大程度上的模拟原指数，思想就是主成分分析（PCA）。将其建模成一个优化问题，假设我们构建的资产组合为 `P = {w1,w2,...,wN}` ，`wi` 表示给于第i各成分股的权重。优化的目标便是使得P中权重的1范数的和 `∑|wi|,i = 1,2,3,...,N` 最小，其含义就是令投资组合的建仓成本达到尽可能小。如果我们只允许做多，那么问题便可以变成更为简单的使 `∑wi` 最小。然而如何保证让构建的P尽可能的跟踪指数走势呢？我们可以做这样一个简单合理的假设：如果P在过去一段时间窗口 `T` 日中能够拟合指数的走势（体现在收益率的一致性），那么 `P` 就能下一次调仓前跟踪指数的走势。如果令 `r(i,t)` 表示第 `i` 个成分股在t日的收益率， `R(t)` 表示待跟踪的标的指数在 `t` 日的收益率，那么： `∑r(i,t)*wi = R(t)` ，`t = 1,2,3,...,T` .那么我们要建模的优化问题便是：

```
min ∑wi,     i = 1,2,3,...,N
s.t. ∑r(i,t)*wi = R(t),        t = 1,2,3,...,T
    wi >= 0
```

2）分析：

- ①若 `N > T`：约束条件为欠定方程组，解空间会有无数的解。我们需要通过迭代运算从解空间中寻找满足优化目标的最优解，通常在边界取到；
- ②若 `N = T`：有唯一解，不具备优化的空间；
- ③若 `N < T`：约束条件为超定方程组，解空间无解（假设曲线每天的走势是不相关的）

3）求解：

通过分析我们知道应该在 `N > T` 的条件下去做优化，具体的步骤为：

- ①：不等式约束 `wi >= 0` 可通过添加log barrier惩罚项将其约束到优化目标中：`min∑wi-u*log(wi)`
- ②：可利用原对偶内点法去求解目标函数，算法原理可参考文献“Primal-Dual Interior Point algorithms for Linear Programming--George Tzallas-Regas”
- ③：依据②中求得的权重去配指数。由于近似最优解一般在边界上取到，因此必然最优 `w` 是一个降维后的结果，即有些成分股的权重配比小到无法操作，舍去即可

策略回测

1）策略目标

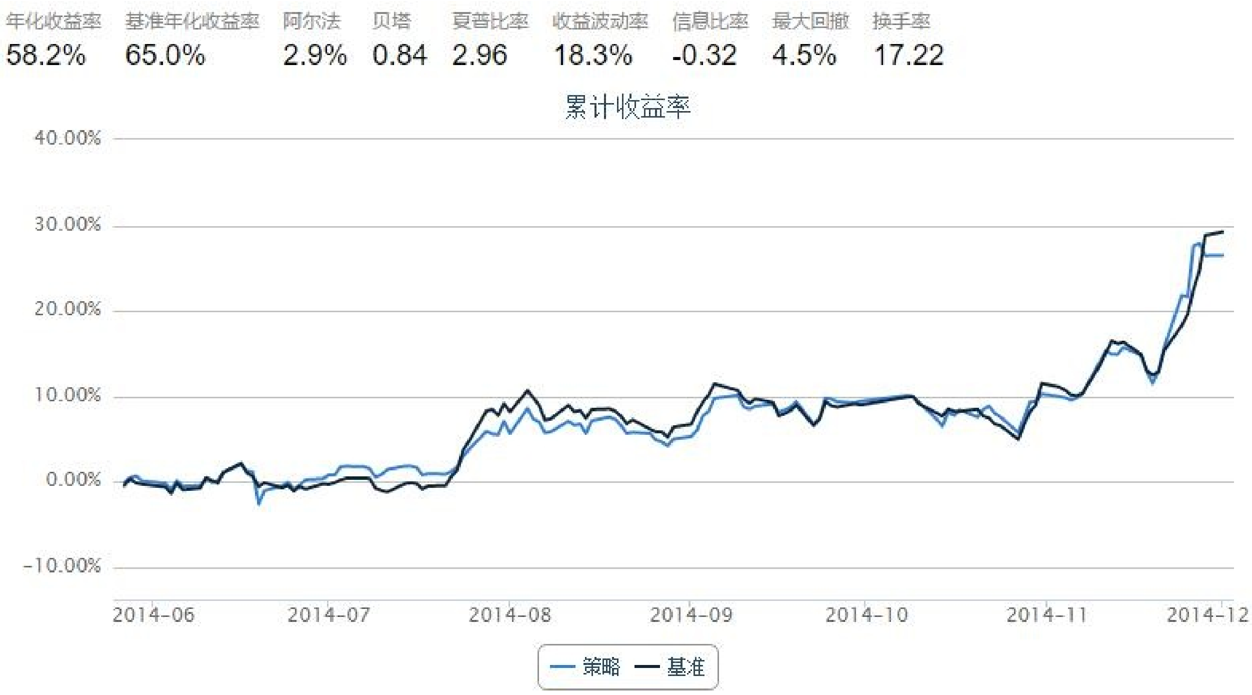
以上证50为例，我们希望选取部分上证50的成分股并配以一定的优化后的权重构建投资组合 `P = {w1,w2,...,w50}`，让 `P` 可以有效的跟踪上证50的走势，并且尽可能的选用部分成分股去构建组合。

2）仿真环境

- ①跟踪标的：上证50
- ②股票池：上证50成分股
- ③调仓：5天（由于只是研究策略，因此假设每次调仓都用新得的股票替换掉原有的，实际做的时候并不需要这样，只要交易当前组合和原组合的差额即可）

```
from CAL.PyCAL import *
import numpy as np
import copy as cp

start = '2014-05-01'                     #  回测起始时间
end = '2014-12-01'                         #  回测结束时间
benchmark = 'SH50'                       #  策略参考标准
universe = set_universe('SH50')  # 证券池，支持股票和基金
```

```
capital_base = 100000                      # 起始资金
freq = 'd'                                 # 策略类型，'d'表示日间
策略使用日线回测，'m'表示日内策略使用分钟线回测
refresh_rate = 5    # 调仓频率，表示执行handle_data的时间间隔，若freq
 = 'd'时间间隔的单位为交易日，若freq = 'm'时间间隔为分钟

def initialize(account):                   # 初始化虚拟账户状态
    account.portfolioNumList = []
    pass


def dict2list(dictionary):
    tmplist = []
    for index in dictionary:
        tmplist.append(dictionary[index])
    return tmplist

def handle_data(account):                  # 每个交易日的买入卖出指令

    histLength = 15
    stockLength = len(account.universe)

    ####get the return rate of the universe
    closePrice = account.get_attribute_history('closePrice',hist
Length+1)
    uniRetList = []
    for index in closePrice:
        uniRetList.append(((closePrice[index][1:]-closePrice[ind
ex][:-1])/closePrice[index][:-1]).tolist())
    uniRetMat = np.mat(uniRetList).T

    ####get the return rate of the benchmark
    calendar = Calendar('China.SSE')
    startDate = calendar.advanceDate(account.current_date,'-'+st
r(histLength+1)+'B').toDateTime()
    endDate = calendar.advanceDate(account.current_date,'-1B').t
oDateTime()
    benchmark = DataAPI.MktIdxdGet(ticker = "000016",
                field = "closeIndex",
                beginDate = startDate,
                endDate = endDate,pandas = '1')
    bmClose = benchmark['closeIndex'].tolist()
    bmRet = []
    for index in range(len(bmClose)-1):
        bmRet.append((bmClose[1:][index]-bmClose[:-1][index])/bm
Close[:-1][index])
    bmRetMat = np.mat(bmRet)

    ####initialization: constant
    ##ones: stockLength
    ones = np.ones(stockLength)

    ##unitMat: stockLength  *  stockLength
    unitMat = np.diag(ones)

    ##zeros: histLength
    zeros = np.zeros(histLength)

    ##zero: stockLength
    zero = np.zeros(stockLength)

    ##zeroMatMid: histLength * histLength
    zeroMatMid = np.diag(zeros)

    ##zeroMat: stockLength  *  stockLength
    zeroMat = np.diag(zero)


    ##zeroMatRes: histLength *  stockLength
    zeroMatRes = np.zeros((histLength,stockLength))

    ##initialization: variables
    ##w: stockLength
    # w = np.ones(stockLength)/(stockLength)
    w = np.ones(stockLength)/(stockLength)

    ##wmat: stockLength  *  stockLength
    wMat = np.diag(w)

    ##u: histLength
    u = np.ones(histLength)/histLength

    ##uMat: histLength  *  histLength
    uMat = np.diag(u)


    ##v: stockLength
    v = np.ones(stockLength)/(stockLength)

    ##vMat: stockLength  *  stockLength
    vMat = np.diag(v)

    ##R: histLength * stockLength
    R = uniRetMat

    ##Q: histLength  *  stockLength
    Q = R

    ##splMat: (stockLength + histLength + stockLength) * (stockL
ength + histLength + stockLength)
    splMatTmp1 = np.hstack([zeroMat,Q.T,unitMat])
    splMatTmp2 = np.hstack([Q,zeroMatMid,zeroMatRes])
    splMatTmp3 = np.hstack([vMat,zeroMatRes.T,wMat])
    splMatTmp = np.vstack([splMatTmp1,splMatTmp2,splMatTmp3])
    splMat = splMatTmp
    ##mulVec: length = stockLength + histLength + stockLength
    firstCol = np.subtract(ones,np.dot(Q.T,u))
    firstCol = np.subtract(firstCol,v)
    secondCol = np.subtract(bmRetMat,np.dot(Q,w))
    thirdCol = np.mat(np.subtract(np.dot(0.1,ones),np.dot(wMat,v)))


    mulVec = np.hstack([firstCol,secondCol,thirdCol])

    ####algorithm iteration part
    d = 1
    mu = 0.1
    itera = 0

    while itera < 300:
    # while d > 0.01:
        ##calculate the dirtaw, dirtau, dirtav
        temp = np.dot(splMat.I,mulVec.T).tolist()
        dirtaw = [index[0] for index in temp[:stockLength]]
        dirtau = [index[0] for index in temp[stockLength:stockLength+histLength]]
        dirtav = [index[0] for index in temp[stockLength+histLength:]]


        ##update the vector w, u, v and the matrix wmat, umat, vmat
        w = np.add(w,dirtaw)
        u = np.add(u,dirtau)
        v = np.add(v,dirtav)
        wMat = np.diag(w)
        uMat = np.diag(u)
        vMat = np.diag(v)

        ##init the matrix splmat: (stockLength + histLength + stockLength) * (stockLength + histLength + stockLength)
        splMatTmp1 = np.hstack([zeroMat,Q.T,unitMat])
        splMatTmp2 = np.hstack([Q,zeroMatMid,zeroMatRes])
        splMatTmp3 = np.hstack([vMat,zeroMatRes.T,wMat])
        splMatTmp = np.vstack([splMatTmp1,splMatTmp2,splMatTmp3])
        splMat = splMatTmp

        ##init the vector mulvec: length = stockLength + histLength + stockLength
        firstCol = np.subtract(ones,np.dot(Q.T,u))
        firstCol = np.subtract(firstCol,v)
        secondCol = np.subtract(bmRetMat,np.dot(Q,w))
        thirdCol = np.mat(np.subtract(np.dot(mu,ones),np.dot(wMat,v)))

        ##calculate the iteration condition variable d
        tmp1 = 0
        for index in dirtaw:
            tmp1 += index**2
        tmp2 = 0
        for index in dirtau:
            tmp2 += index**2
        d = tmp1 + tmp2

        ##update the itera and mu
        itera += 1
        mu = mu*(1-stockLength**(-0.5))**5


    ##weight of the components
    weight = w
    for index in range(stockLength):
        if weight[index]<0:
            weight[index] = 0
    weightSum = np.sum(weight)
    weightReg = [index/weightSum for index in weight]
    for index in range(stockLength):
        if weightReg[index]<10*10**(-3):
            weightReg[index] = 0
    count = 0
    for index in weightReg:
        if index != 0:
            count += 1
    account.portfolioNumList.append({account.current_date:count}
)

    ##Sell portfolio
    for index in account.valid_secpos:
        order_to(index,0)

    ####Buy portfolio
    portfolio = []
    for index in range(stockLength):
        amount = round(100*weightReg[index])*100
        if amount != 0:
            portfolio.append({account.universe[index]:amount})
        # amount = account.cash*weightReg[index]/account.referen
cePrice[account.universe[index]]
        order(account.universe[index],amount)
    print 'The portfolio at '+str(account.current_date) + ' has '
 + str(len(portfolio)) + ' stocks, which '
    print portfolio
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
58.2%	65.0%	2.9%	0.84	2.96	18.3%	-0.32	4.5%	17.22

```
The portfolio at 2014-05-27 00:00:00 has 15 stocks, which
[{'601601.XSHG': 500.0}, {'600111.XSHG': 100.0}, {'600018.XSHG':
 500.0}, {'600519.XSHG': 1400.0}, {'601668.XSHG': 100.0}, {'6010
88.XSHG': 1500.0}, {'601998.XSHG': 500.0}, {'600010.XSHG': 500.0
}, {'600637.XSHG': 500.0}, {'600999.XSHG': 300.0}, {'600887.XSHG
': 1100.0}, {'601169.XSHG': 1300.0}, {'601988.XSHG': 200.0}, {'6
01318.XSHG': 900.0}, {'601901.XSHG': 100.0}]
The portfolio at 2014-06-04 00:00:00 has 11 stocks, which
[{'600583.XSHG': 500.0}, {'600893.XSHG': 300.0}, {'601288.XSHG':
 1700.0}, {'600585.XSHG': 700.0}, {'600256.XSHG': 500.0}, {'6018
00.XSHG': 800.0}, {'601989.XSHG': 100.0}, {'600887.XSHG': 800.0}
, {'601398.XSHG': 3400.0}, {'600030.XSHG': 500.0}, {'600150.XSHG
': 800.0}]
The portfolio at 2014-06-11 00:00:00 has 15 stocks, which
[{'601601.XSHG': 600.0}, {'600893.XSHG': 300.0}, {'600111.XSHG':
 100.0}, {'600018.XSHG': 400.0}, {'601390.XSHG': 1500.0}, {'6012
88.XSHG': 800.0}, {'601668.XSHG': 700.0}, {'601818.XSHG': 500.0}
, {'600690.XSHG': 1400.0}, {'600010.XSHG': 700.0}, {'600999.XSHG
': 500.0}, {'601628.XSHG': 200.0}, {'600887.XSHG': 400.0}, {'601
766.XSHG': 400.0}, {'600030.XSHG': 1400.0}]
The portfolio at 2014-06-18 00:00:00 has 7 stocks, which
[{'600036.XSHG': 100.0}, {'600018.XSHG': 300.0}, {'600256.XSHG':
 500.0}, {'600406.XSHG': 7600.0}, {'600048.XSHG': 400.0}, {'6008
87.XSHG': 200.0}, {'600030.XSHG': 900.0}]
The portfolio at 2014-06-25 00:00:00 has 7 stocks, which
[{'600050.XSHG': 1200.0}, {'600089.XSHG': 900.0}, {'600256.XSHG'
: 900.0}, {'600104.XSHG': 1500.0}, {'601989.XSHG': 2000.0}, {'60
0837.XSHG': 1000.0}, {'601398.XSHG': 2200.0}]
The portfolio at 2014-07-02 00:00:00 has 9 stocks, which
[{'600036.XSHG': 800.0}, {'601390.XSHG': 1900.0}, {'600016.XSHG'
: 200.0}, {'601006.XSHG': 600.0}, {'601088.XSHG': 300.0}, {'6005
85.XSHG': 1300.0}, {'600690.XSHG': 1300.0}, {'601988.XSHG': 2900
```

```
.0}, {'601186.XSHG': 500.0}]
The portfolio at 2014-07-09 00:00:00 has 8 stocks, which
[{'601288.XSHG': 2100.0}, {'601668.XSHG': 400.0}, {'601088.XSHG'
: 2100.0}, {'600089.XSHG': 1400.0}, {'600690.XSHG': 900.0}, {'60
0010.XSHG': 1400.0}, {'601988.XSHG': 1100.0}, {'601398.XSHG': 40
0.0}]
The portfolio at 2014-07-16 00:00:00 has 6 stocks, which
[{'600583.XSHG': 1400.0}, {'601006.XSHG': 1400.0}, {'600256.XSHG
': 500.0}, {'601998.XSHG': 1700.0}, {'600048.XSHG': 400.0}, {'60
0518.XSHG': 4400.0}]
The portfolio at 2014-07-23 00:00:00 has 23 stocks, which
[{'600583.XSHG': 200.0}, {'601601.XSHG': 700.0}, {'600036.XSHG':
 400.0}, {'600018.XSHG': 300.0}, {'600519.XSHG': 100.0}, {'60139
0.XSHG': 1400.0}, {'601288.XSHG': 200.0}, {'601006.XSHG': 300.0}
, {'601088.XSHG': 500.0}, {'600256.XSHG': 500.0}, {'601998.XSHG'
: 100.0}, {'600015.XSHG': 100.0}, {'600028.XSHG': 500.0}, {'6000
10.XSHG': 300.0}, {'600999.XSHG': 400.0}, {'600109.XSHG': 100.0}
, {'601989.XSHG': 1100.0}, {'600887.XSHG': 200.0}, {'601766.XSHG
': 200.0}, {'601169.XSHG': 700.0}, {'601988.XSHG': 1100.0}, {'60
0030.XSHG': 200.0}, {'601901.XSHG': 300.0}]
The portfolio at 2014-07-30 00:00:00 has 8 stocks, which
[{'600893.XSHG': 200.0}, {'600104.XSHG': 300.0}, {'600015.XSHG':
 100.0}, {'601989.XSHG': 600.0}, {'600518.XSHG': 800.0}, {'60162
8.XSHG': 2000.0}, {'600887.XSHG': 300.0}, {'601318.XSHG': 5400.0
}]
The portfolio at 2014-08-06 00:00:00 has 7 stocks, which
[{'601328.XSHG': 2800.0}, {'600036.XSHG': 600.0}, {'600111.XSHG'
: 1200.0}, {'601088.XSHG': 700.0}, {'601800.XSHG': 1300.0}, {'60
0028.XSHG': 1900.0}, {'600518.XSHG': 1100.0}]
The portfolio at 2014-08-13 00:00:00 has 7 stocks, which
[{'601601.XSHG': 1400.0}, {'600018.XSHG': 100.0}, {'601818.XSHG'
: 600.0}, {'601088.XSHG': 2100.0}, {'600406.XSHG': 1600.0}, {'60
0999.XSHG': 2900.0}, {'600887.XSHG': 1300.0}]
The portfolio at 2014-08-20 00:00:00 has 6 stocks, which
[{'600000.XSHG': 1900.0}, {'600111.XSHG': 900.0}, {'600016.XSHG'
: 1600.0}, {'600690.XSHG': 300.0}, {'600150.XSHG': 100.0}, {'601
186.XSHG': 5100.0}]
The portfolio at 2014-08-27 00:00:00 has 8 stocks, which
[{'601328.XSHG': 2000.0}, {'600585.XSHG': 300.0}, {'600048.XSHG'
: 1300.0}, {'601800.XSHG': 1200.0}, {'600690.XSHG': 700.0}, {'60
0028.XSHG': 1400.0}, {'601989.XSHG': 1200.0}, {'601988.XSHG': 18
00.0}]
The portfolio at 2014-09-03 00:00:00 has 8 stocks, which
[{'600016.XSHG': 2200.0}, {'601818.XSHG': 1100.0}, {'600585.XSHG
': 2500.0}, {'600089.XSHG': 2000.0}, {'600048.XSHG': 400.0}, {'6
00104.XSHG': 1100.0}, {'600518.XSHG': 500.0}, {'600150.XSHG': 20
0.0}]
The portfolio at 2014-09-11 00:00:00 has 7 stocks, which
[{'601857.XSHG': 4300.0}, {'601088.XSHG': 700.0}, {'600048.XSHG'
: 1000.0}, {'600690.XSHG': 600.0}, {'600518.XSHG': 1000.0}, {'60
1169.XSHG': 800.0}, {'601901.XSHG': 1400.0}]
The portfolio at 2014-09-18 00:00:00 has 8 stocks, which
[{'601328.XSHG': 1700.0}, {'600111.XSHG': 500.0}, {'600016.XSHG'
```

```
: 3100.0}, {'600015.XSHG': 300.0}, {'600887.XSHG': 400.0}, {'600
030.XSHG': 1800.0}, {'601186.XSHG': 600.0}, {'601318.XSHG': 1700
.0}]
The portfolio at 2014-09-25 00:00:00 has 11 stocks, which
[{'600050.XSHG': 600.0}, {'600583.XSHG': 700.0}, {'600111.XSHG':
 3000.0}, {'600048.XSHG': 500.0}, {'600015.XSHG': 300.0}, {'6000
28.XSHG': 400.0}, {'600837.XSHG': 1400.0}, {'601169.XSHG': 100.0
}, {'601988.XSHG': 600.0}, {'601186.XSHG': 100.0}, {'601901.XSHG
': 1700.0}]
The portfolio at 2014-10-09 00:00:00 has 9 stocks, which
[{'600519.XSHG': 1400.0}, {'600585.XSHG': 3000.0}, {'600256.XSHG
': 300.0}, {'600048.XSHG': 2500.0}, {'600104.XSHG': 900.0}, {'60
0015.XSHG': 600.0}, {'600999.XSHG': 300.0}, {'600109.XSHG': 600.
0}, {'600887.XSHG': 100.0}]
The portfolio at 2014-10-16 00:00:00 has 8 stocks, which
[{'600519.XSHG': 800.0}, {'601166.XSHG': 6900.0}, {'600104.XSHG'
: 700.0}, {'600010.XSHG': 200.0}, {'600999.XSHG': 200.0}, {'6017
66.XSHG': 700.0}, {'601169.XSHG': 100.0}, {'600030.XSHG': 100.0}
]
The portfolio at 2014-10-23 00:00:00 has 7 stocks, which
[{'601390.XSHG': 400.0}, {'601668.XSHG': 1100.0}, {'600048.XSHG'
: 1100.0}, {'600104.XSHG': 200.0}, {'600837.XSHG': 4000.0}, {'60
1398.XSHG': 2200.0}, {'601318.XSHG': 800.0}]
The portfolio at 2014-10-30 00:00:00 has 6 stocks, which
[{'601857.XSHG': 800.0}, {'600256.XSHG': 600.0}, {'600015.XSHG':
 200.0}, {'600887.XSHG': 100.0}, {'601398.XSHG': 7600.0}, {'6000
30.XSHG': 500.0}]
The portfolio at 2014-11-06 00:00:00 has 8 stocks, which
[{'601328.XSHG': 4800.0}, {'600519.XSHG': 1000.0}, {'601818.XSHG
': 800.0}, {'601088.XSHG': 900.0}, {'600585.XSHG': 600.0}, {'600
406.XSHG': 900.0}, {'600104.XSHG': 500.0}, {'601901.XSHG': 400.0
}]
The portfolio at 2014-11-13 00:00:00 has 7 stocks, which
[{'600583.XSHG': 1300.0}, {'601601.XSHG': 2400.0}, {'600893.XSHG
': 400.0}, {'601818.XSHG': 1500.0}, {'600104.XSHG': 1700.0}, {'6
00109.XSHG': 200.0}, {'601318.XSHG': 2300.0}]
The portfolio at 2014-11-20 00:00:00 has 9 stocks, which
[{'601601.XSHG': 3800.0}, {'600018.XSHG': 300.0}, {'601006.XSHG'
: 600.0}, {'600089.XSHG': 1300.0}, {'601998.XSHG': 800.0}, {'600
015.XSHG': 1300.0}, {'600010.XSHG': 300.0}, {'601989.XSHG': 300.
0}, {'601988.XSHG': 1200.0}]
The portfolio at 2014-11-27 00:00:00 has 14 stocks, which
[{'601601.XSHG': 400.0}, {'600893.XSHG': 1700.0}, {'600111.XSHG'
: 600.0}, {'600519.XSHG': 300.0}, {'601288.XSHG': 800.0}, {'6010
88.XSHG': 100.0}, {'600585.XSHG': 1800.0}, {'600089.XSHG': 500.0
}, {'600015.XSHG': 500.0}, {'600999.XSHG': 200.0}, {'600109.XSHG
': 600.0}, {'600518.XSHG': 300.0}, {'601988.XSHG': 1900.0}, {'60
1318.XSHG': 100.0}]
```

回测结果

从上面的回测结果中我们可以看到，每个时点的持仓个股数在7-15只的范围内，然而组合依然可以较好的跟踪标的上证50指数。跟踪误差没有详细去计算，应该在400bps（4%）以内，考虑到大大减少了建仓的难度和成本，因此这个跟踪误差是可以接受的。

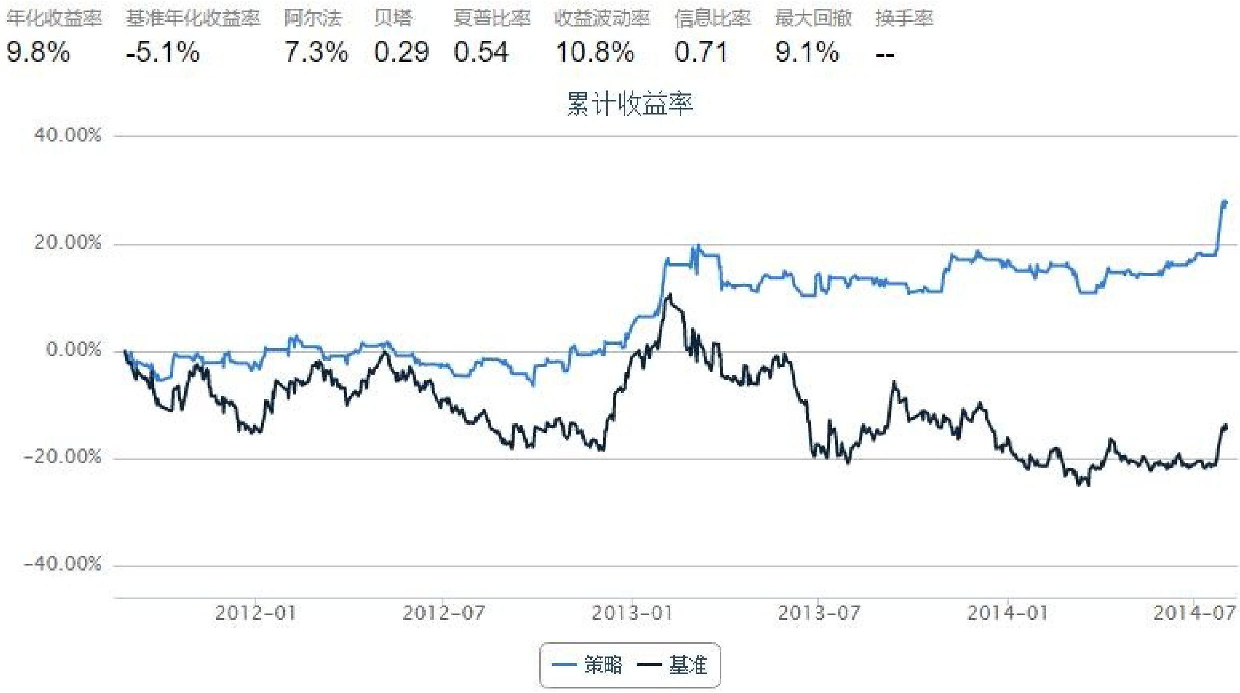
思考

- ① 考虑到我们上面仅仅回测了最简单的model，事实上我们完全可以在上面的优化问题中加入更多的限制条件，以获得更为进准的匹配效果。例如，如果我们限制了每只股票的持仓不得高于某个比例，则可以将条件 `wi >= 0` 改为 `xi >= wi >=0` 。如果考虑可以融券做空，则优化目标变为 `min ∑|wi|` 。如果考虑加入仓位限制，例如仓位控制在70%以上，则可加入 `∑wi > 70%` ，等等。
- ② 该模型可以在一定程度上挖掘任意指数、股票型基金曲线的持仓信息。以基金为例：只要是股票型基金，尤其是该基金的投资标的池已知，例如基金嘉实大盘研究精选的标的池很有可能就是HS300，那么便可以用HS300去跟踪该基金，得到的股票组合很有可能就是嘉实大盘研究精选的持仓。

9.2 GMVP · Global Minimum Variance Portfolio (GMVP)

> 来源：https://uqer.io/community/share/55461734f9f06c1c3d688030

```
import pandas as pd
import numpy as np


start = '2011-07-01'                            # 回测起始时间
end = '2014-08-01'                              # 回测结束时间
benchmark = 'SH50'                             # 策略参考标准
universe = ['601398.XSHG','600028.XSHG', '601988.XSHG', '600036.
XSHG','600030.XSHG','601318.XSHG', '600000.XSHG', '600019.XSHG',
'600519.XSHG', '601166.XSHG']
capital_base = 100000                          # 起始资金
longest_history = 40                             # handle_data 函数中
可以使用的历史数据最长窗口长度
refresh_rate = 10                                # 调仓频率，即每 refre
sh_rate 个交易日执行一次 handle_data() 函数


def initialize(account):                        # 初始化虚拟账户状态
    pass

def handle_data(account):                       # 每个交易日的买入卖出指令

    history_data = account.get_attribute_history('closePrice',40
)
    retmatrix = []
    for s in account.universe:
        retmatrix.append([history_data[s][i]/ history_data[s][i
- 1] for i in range(1,40) ])
    retmatrix = np.array(retmatrix)
    covmatrix =  np.cov(retmatrix, y=None, rowvar=1, bias=0, ddo
f=None)
    covmatrix = np.matrix(covmatrix)  # 不加这句执行矩阵求逆报错
    covinv = np.linalg.inv(covmatrix)
    one_row = np.matrix(np.ones(len(account.universe)))
    one_vector = np.matrix(np.ones(len(account.universe))).trans
pose()
    up = np.dot(covinv, one_vector)
    down = np.dot(np.dot(one_row, covinv), one_vector)
    weights = up/down
    weightsum = 0
    for a in weights:
        weightsum += a
    index= 0
    for s in account.universe:
        weigh = weights[index]/weightsum
        index = index + 1
        amount = account.cash * weigh / account.referencePrice[s
]
        order_to(s,amount)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
9.8%	-5.1%	7.3%	0.29	0.54	10.8%	0.71	9.1%	--

just implement the examples in the API doc

9.3 凸优化 · 如何在 Python 中利用 CVXOPT 求解二次规划问题

> 来源：https://uqer.io/community/share/55c9a55df9f06c91f818c675

问题描述：

在实际生活中，我们经常会遇到一些优化问题，简单的线性规划可以作图求解，但是对于目标函数包含二次项时，则需要另觅它法

在金融实践中，马科维茨均方差模型就有实际的二次优化需求

作为金融实践中常用的方法，本篇将对CVXOPT中求解二次规划的问题进行举例详细说明，关于该方法在均方差优化中的实践应用，参见后续发帖

1、二次规划问题的标准形式

$$
\begin{aligned}
min \quad & \frac{1}{2}x^TPx+q^Tx \\
s.t. \quad & Gx \le h \\
& Ax = b
\end{aligned}
$$

上式中，`x` 为所要求解的列向量，`xT` 表示 `x` 的转置

接下来，按步骤对上式进行相关说明：

- 上式表明，任何二次规划问题都可以转化为上式的结构，事实上用cvxopt的第一步就是将实际的二次规划问题转换为上式的结构，写出对应的 `P` 、 `q` 、 `G` 、 `h` 、 `A` 、 `b`
- 目标函数若为求 `max` ，可以通过乘以−1，将最大化问题转换为最小化问题
- `Gx≤b` 表示的是所有的不等式约束，同样，若存在诸如 `x≥0` 的限制条件，也可以通过乘以−1转换为 `≤` 的形式
- `Ax=b` 表示所有的等式约束

2、以一个标准的例子进行过程说明

$$min(x,y) \quad \frac{1}{2}x^2+3x+4y$$

$$s.t. \quad x,y \geq 0$$

$$x+3y \geq 15$$

$$2x+5y \leq 100$$

$$3x+4y \leq 80$$

例子中，需要求解的是 `x`，`y` ，我们可以把它写成向量的形式，同时，也需要将限制条件按照上述标准形式进行调整，用矩阵形式表示，如下所示：

$$min(x,y) \quad \frac{1}{2}\begin{bmatrix}x\\y\end{bmatrix}^T\begin{bmatrix}1&0\\0&0\end{bmatrix}\begin{bmatrix}x\\y\end{bmatrix}+\begin{bmatrix}3\\4\end{bmatrix}^T\begin{bmatrix}x\\y\end{bmatrix}$$

$$\begin{bmatrix}-1&0\\0&-1\\-1&-3\\2&5\\3&4\end{bmatrix}\begin{bmatrix}x\\y\end{bmatrix}\leq\begin{bmatrix}0\\0\\-15\\100\\80\end{bmatrix}$$

- 如上所示，目标函数和限制条件均转化成了二次规划的标准形式，这是第一步，也是最难的一步，接下来的事情就简单了
- 对比上式和标准形式，不难得出：

$$P=\begin{bmatrix}1&0\\0&0\end{bmatrix},q=\begin{bmatrix}3\\4\end{bmatrix},G=\begin{bmatrix}-1&0\\0&-1\\-1&-3\\2&5\\3&4\end{bmatrix},h=\begin{bmatrix}0\\0\\-15\\100\\80\end{bmatrix}$$

接下来就是几行简单的代码，目的是告诉计算机上面的参数具体是什么

```
from cvxopt  import solvers, matrix
P = matrix([[1.0,0.0],[0.0,0.0]])   # matrix里区分int和double，所以数字后面都需要加小数点
q = matrix([3.0,4.0])
G = matrix([[-1.0,0.0,-1.0,2.0,3.0],[0.0,-1.0,-3.0,5.0,4.0]])
h = matrix([0.0,0.0,-15.0,100.0,80.0])

sol = solvers.qp(P,q,G,h)   # 调用优化函数solvers.qp求解
print sol['x']  # 打印结果，sol里面还有很多其他属性，读者可以自行了解

     pcost       dcost       gap    pres   dres
 0:  1.0780e+02 -7.6366e+02  9e+02  1e-16  4e+01
 1:  9.3245e+01  9.7637e+00  8e+01  1e-16  3e+00
 2:  6.7311e+01  3.2553e+01  3e+01  6e-17  1e+00
 3:  2.6071e+01  1.5068e+01  1e+01  2e-16  7e-01
 4:  3.7092e+01  2.3152e+01  1e+01  2e-16  4e-01
 5:  2.5352e+01  1.8652e+01  7e+00  8e-17  3e-16
 6:  2.0062e+01  1.9974e+01  9e-02  6e-17  3e-16
 7:  2.0001e+01  2.0000e+01  9e-04  6e-17  3e-16
 8:  2.0000e+01  2.0000e+01  9e-06  9e-17  2e-16
Optimal solution found.
[ 7.13e-07]
[ 5.00e+00]
```

- 看了上面的代码，是不是觉得很简单。因为难点不在代码，而是在于将实际优化问题转化为标准形式的过程
- 在上面的例子中，并没有出现等号，当出现等式约束时，过程一样，找到 `A` , `b` ，然后运行代码 `sol = solvers.qp(P,q,G,h,A,b)` 即可求解

扩展：上述定义各个矩阵参数用的是最直接的方式，其实也可以结合Numpy来定义上述矩阵

```python
from cvxopt import solvers, matrix
import numpy as np

P = matrix(np.diag([1.0,0]))  #  对于一些特殊矩阵，用numpy创建会方便很多（在本例中可能感受不大）
q = matrix(np.array([3.0,4]))
G = matrix(np.array([[-1.0,0],[0,-1],[-1,-3],[2,5],[3,4]]))
h = matrix(np.array([0.0,0,-15,100,80]))
sol = solvers.qp(P,q,G,h)
```

```
     pcost       dcost       gap    pres   dres
 0:  1.0780e+02 -7.6366e+02  9e+02  1e-16  4e+01
 1:  9.3245e+01  9.7637e+00  8e+01  1e-16  3e+00
 2:  6.7311e+01  3.2553e+01  3e+01  6e-17  1e+00
 3:  2.6071e+01  1.5068e+01  1e+01  2e-16  7e-01
 4:  3.7092e+01  2.3152e+01  1e+01  2e-16  4e-01
 5:  2.5352e+01  1.8652e+01  7e+00  8e-17  3e-16
 6:  2.0062e+01  1.9974e+01  9e-02  6e-17  3e-16
 7:  2.0001e+01  2.0000e+01  9e-04  6e-17  3e-16
 8:  2.0000e+01  2.0000e+01  9e-06  9e-17  2e-16
Optimal solution found.
```

先写到这吧，关于二次规划在均方差优化中的实践应用，参见后续发帖，欢迎交流~~

十 波动率

10.1 波动率选股 · 风平浪静 风起猪飞

> 来源：https://uqer.io/community/share/550fe978f9f06c7a9ae9a557

策略基本思路是：挑选具有较低波动率品种以防市场大跌，同时持有较低涨幅品种，等待时机，以图崛起。

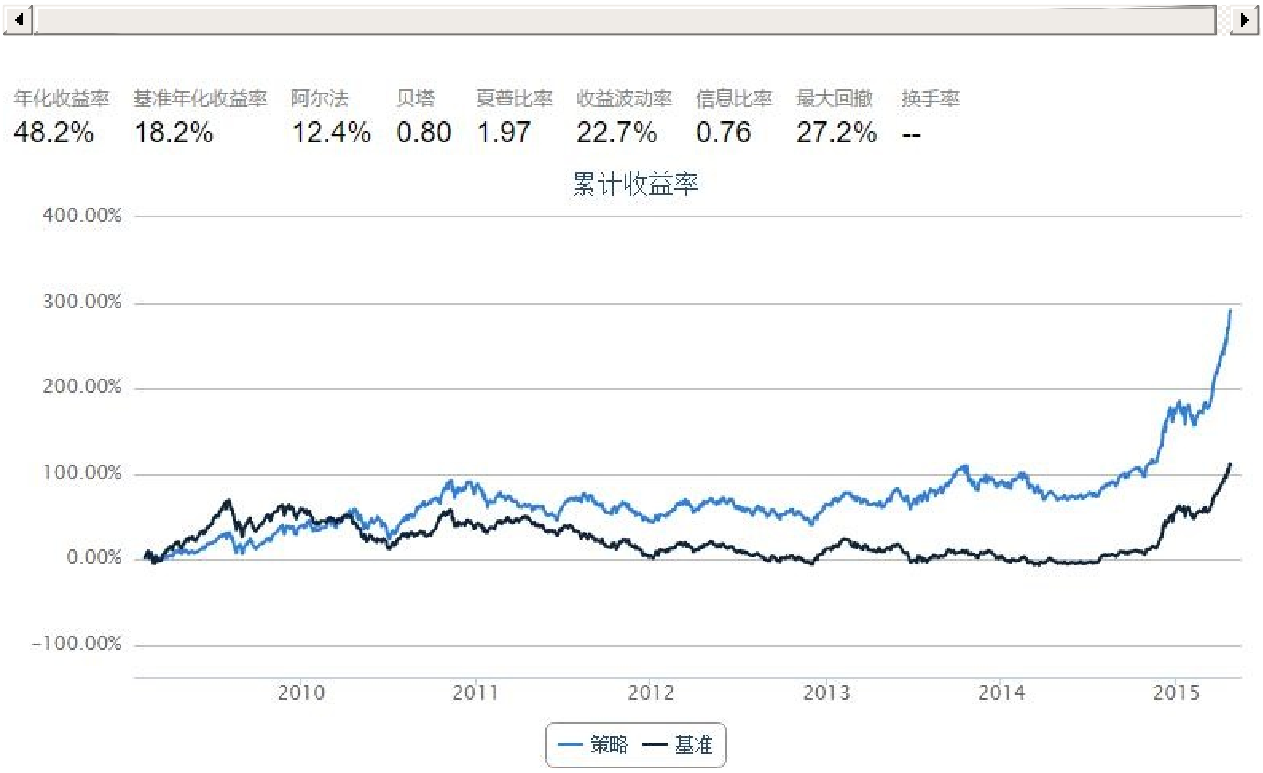
其中，较低波动率品种指标为股票近一年收盘价的标准差小于0.2；较低涨幅品种指股票年化收益率与无风险利率之差除以基准收益率与无风险利率之差小于0.5的股票。

本策略无风险利率特指：0.035

本策略的参数如下：

- 起始日期： 2009年1月1日
- 结束日期： 2015年3月20日
- 股票池： 沪深300
- 业绩基准： 沪深300
- 起始资金： 100000元
- 调仓周期： 3个月

```
import numpy as np
import pandas as pd
from datetime import timedelta, datetime

start    = datetime(2009, 1, 1)                 # 回测起始时间
end     = datetime(2015, 3, 20)                # 回测结束时间
benchmark = 'HS300'                             # 策略参考标准
universe  = set_universe('HS300')         # 股票池
capital_base = 10000000                 # 起始资金
refresh_rate = 60                    # 调仓频率60个交易日
longest_history = 20

def initialize(account):                      # 初始化虚拟账户状态
    pass

def handle_data(account):              # 每个交易日的买入卖出指令
    today = account.current_date

    stocks  = []
    changes = []
    vols   = []
    returns = []
    hist = account.get_history(20)
    # 基准收益率
    bm_return = hist['benchmark']['return'][-1]

    # 无风险利率
    rf_return = 0.035

    for stock in account.universe:
        # 计算股票的收益和收益波动率
        rt  = hist[stock]['closePrice'][-1]/hist[stock]['closePrice'][0] - 1
        rts = hist[stock]['closePrice']/np.roll(hist[stock]['closePrice'], 1) - 1
        vol = np.std(rts[1:])

        stocks.append(stock)
        changes.append(((rt-rf_return)/(bm_return-rf_return)))
        vols.append(vol)
        returns.append(rt)


    df = pd.DataFrame({'stocks':stocks, 'changes':changes, 'vols':vols, 'returns':returns})

    # 筛选符合条件的股票
    signal_stocks = list(df[df.vols<0.2][df.changes<0.5].stocks.values)

    c = account.cash

    # 卖出不在信号股票集合内的持仓，计算可用资金
    #print today
    #print '卖出'
    for stock in account.valid_secpos:
        if account.valid_secpos[stock] > 0 and stock not in signal_stocks:
            order_to(stock, 0)
            #print stock,
            c += account.valid_secpos[stock] * hist[stock]['closePrice'][-1]
        elif stock in signal_stocks:
            signal_stocks.remove(stock)

    # 买入股票
    #print
    #print '买入'
    for stock in signal_stocks:
        #print stock,
        order(stock, int(0.9*c/len(signal_stocks)/hist[stock]['closePrice'][-1])/100*100)

    #print
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
48.2%	18.2%	12.4%	0.80	1.97	22.7%	0.76	27.2%	--

10.2 波动率择时

基于 VIX 指数的择时策略

> 来源：https://uqer.io/community/share/55b6152ff9f06c91fa18c5c9

波动率VIX指数是跟踪市场波动性的指数，一般通过标的期权的隐含波动率计算得来，以芝加哥期权交易所的VIX指数为例，如标的期权的隐含波动率越高，则VIX指数相应越高，一般而言，该指数反映出投资者愿意付出多少成本去对冲投资风险。业内认为，当VIX越高时，表示市场参与者预期后市波动程度会更加激烈，同时也反映其不安的心理状态；相反，VIX越低时，则反映市场参与者预期后市波动程度会趋于缓和的心态。因此，VIX又被称为投资人恐慌指标（The Investor Fear Gauge）。

中国波指是由上证所发布，用于衡量上证50ETF未来30日的预期波动。该指数是根据方差互换的原理，结合50ETF期权的实际运作特点，并通过对上证所交易的50ETF期权价格的计算编制而得。网址为：http://www.sse.com.cn/assortment/derivatives/options/volatility/

本文中，基于优矿平台，自己尝试计算了日间的中国波指，并将其用在了华夏上证50的择时买卖上，以验证VIX指数对未来的预测性

由于上证所未发布其iVIX计算方法，所以此处的计算基于CBOE发布的方法，具体参见： http://www.cboe.com/micro/vix/part2.aspx

策略思路

- 当VIX指数快速上升时，表示市场恐慌情绪蔓延，产生卖出信号
- 当VIX指数快速下降时，恐慌情绪有所舒缓，产生买入信号
- 卖出买入信号均用来买卖华夏上证50ETF基金

注：国内唯一一只期权上证50ETF期权，跟踪标的为华夏上证50ETF(510050)基金

1. 计算历史VIX指数

```
from matplotlib import pylab
import numpy as np
import pandas as pd
import DataAPI
import seaborn as sns
sns.set_style('white')
```

```
from CAL.PyCAL import *
from pandas import Series, DataFrame, concat
import pandas as pd
import numpy as np
import seaborn as sns
sns.set_style('white')
from matplotlib import pylab
import time
import math

def getHistDayOptions(var, date):
    # 使用DataAPI.OptGet，拿到已退市和上市的所有期权的基本信息；
    # 同时使用DataAPI.MktOptdGet，拿到历史上某一天的期权成交信息；
    # 返回历史上指定日期交易的所有期权信息，包括：
    # optID  varSecID  contractType  strikePrice  expDate  trade
Date  closePrice
    # 以optID为index。
    vixDateStr = date.toISO().replace('-', '')
    optionsMkt = DataAPI.MktOptdGet(tradeDate = vixDateStr, fiel
d = [u"optID", "tradeDate", "closePrice"], pandas = "1")
    optionsMkt = optionsMkt.set_index(u"optID")
    optionsMkt.closePrice.name = u"price"

    optionsID = map(str, optionsMkt.index.values.tolist())
    fieldNeeded = ["optID", u"varSecID", u'contractType', u'stri
kePrice', u'expDate']
    optionsInfo = DataAPI.OptGet(optID=optionsID, contractStatus
 = [u"DE", u"L"], field=fieldNeeded, pandas="1")
    optionsInfo = optionsInfo.set_index(u"optID")
    options = concat([optionsInfo, optionsMkt], axis=1, join='in
ner').sort_index()
    return options[options.varSecID==var]

def getNearNextOptExpDate(options, vixDate):
    # 找到options中的当月和次月期权到期日；
    # 用这两个期权隐含的未来波动率来插值计算未来30隐含波动率，是为市场恐慌
指数VIX；
    # 如果options中的最近到期期权离到期日仅剩1天以内，则抛弃这一期权，改
    # 选择次月期权和次月期权之后第一个到期的期权来计算。
    # 返回的near和next就是用来计算VIX的两个期权的到期日
    optionsExpDate = Series(options.expDate.values.ravel()).uniq
ue().tolist()
    near = min(optionsExpDate)
    optionsExpDate.remove(near)
    if Date.parseISO(near) - vixDate < 1:
        near = min(optionsExpDate)
        optionsExpDate.remove(near)
    next = min(optionsExpDate)
    return near, next

def getStrikeMinCallMinusPutClosePrice(options):
    # options 中包括计算某日VIX的call和put两种期权，
    # 对每个行权价，计算相应的call和put的价格差的绝对值，
    # 返回这一价格差的绝对值最小的那个行权价，
    # 并返回该行权价对应的call和put期权价格的差
    call = options[options.contractType==u"CO"].set_index(u"stri
kePrice").sort_index()
```

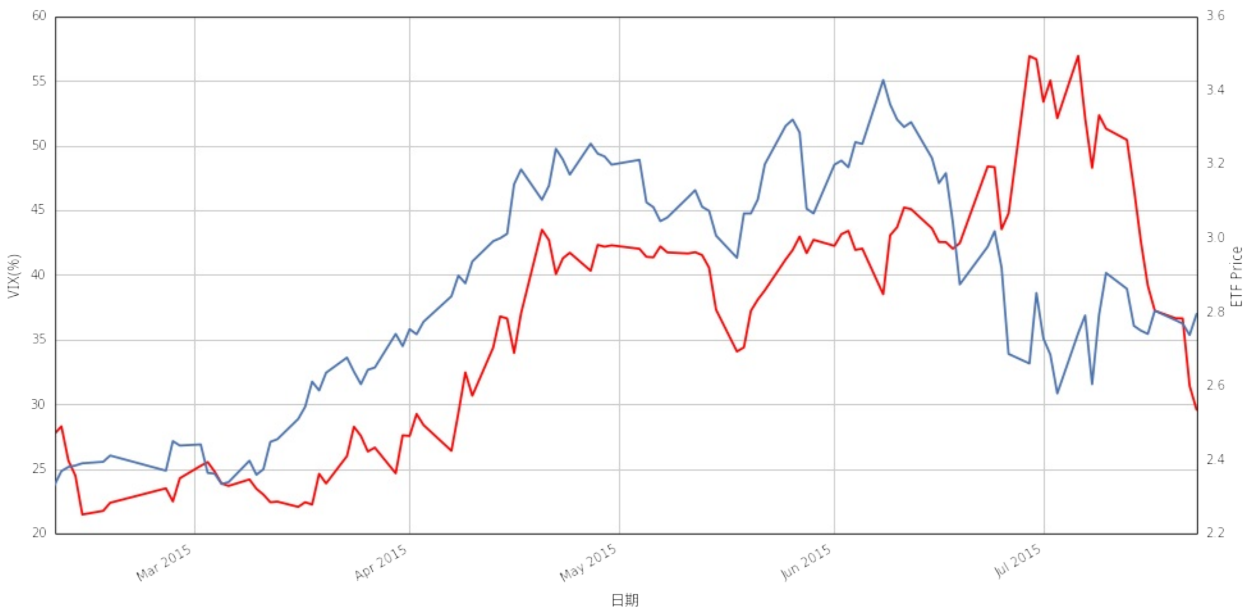
```
    put  = options[options.contractType==u"PO"].set_index(u"strikePrice").sort_index()
    callMinusPut = call.closePrice - put.closePrice
    strike = abs(callMinusPut).idxmin()
    priceDiff = callMinusPut[strike]
    return strike, priceDiff

def calSigmaSquare(options, FF, R, T):
    # 计算某个到期日期权对于VIX的贡献sigma；
    # 输入为期权数据options，FF为forward index price，
    # R为无风险利率， T为期权剩余到期时间
    callAll = options[options.contractType==u"CO"].set_index(u"strikePrice").sort_index()
    putAll  = options[options.contractType==u"PO"].set_index(u"strikePrice").sort_index()
    callAll['deltaK'] = 0.05
    putAll['deltaK']  = 0.05

    # Interval between strike prices
    index = callAll.index
    if len(index) < 3:
        callAll['deltaK'] = index[-1] - index[0]
    else:
        for i in range(1,len(index)-1):
            callAll['deltaK'].ix[index[i]] = (index[i+1]-index[i-1])/2.0
        callAll['deltaK'].ix[index[0]] = index[1]-index[0]
        callAll['deltaK'].ix[index[-1]] = index[-1] - index[-2]
    index = putAll.index
    if len(index) < 3:
        putAll['deltaK'] = index[-1] - index[0]
    else:
        for i in range(1,len(index)-1):
            putAll['deltaK'].ix[index[i]] = (index[i+1]-index[i-1])/2.0
        putAll['deltaK'].ix[index[0]] = index[1]-index[0]
        putAll['deltaK'].ix[index[-1]] = index[-1] - index[-2]

    call = callAll[callAll.index > FF]
    put  = putAll[putAll.index < FF]
    FF_idx = FF
    if not put.empty:
        FF_idx = put.index[-1]
        put['closePrice'].iloc[-1] = (putAll.ix[FF_idx].closePrice + callAll.ix[FF_idx].closePrice)/2.0

    callComponent = call.closePrice*call.deltaK/call.index/call.index
    putComponent  = put.closePrice*put.deltaK/put.index/put.index
    sigma = (sum(callComponent)+sum(putComponent))*np.exp(T*R)*2/T
    sigma = sigma - (FF/FF_idx - 1)**2/T
    return sigma

def calDayVIX(optionVarSecID, vixDate):
    # 利用CBOE的计算方法，计算历史某一日的未来30日期权波动率指数VIX

    # The risk-free interest rates
    R_near = 0.06
    R_next = 0.06
    # 拿取所需期权信息
    options = getHistDayOptions(optionVarSecID, vixDate)
    termNearNext = getNearNextOptExpDate(options, vixDate)
    optionsNearTerm = options[options.expDate == termNearNext[0]]
    optionsNextTerm = options[options.expDate == termNearNext[1]]
    # time to expiration
    T_near = (Date.parseISO(termNearNext[0]) - vixDate)/365.0
    T_next = (Date.parseISO(termNearNext[1]) - vixDate)/365.0
    # the forward index prices
    nearPriceDiff = getStrikeMinCallMinusPutClosePrice(optionsNearTerm)
    nextPriceDiff = getStrikeMinCallMinusPutClosePrice(optionsNextTerm)
    near_F = nearPriceDiff[0] + np.exp(T_near*R_near)*nearPriceDiff[1]
    next_F = nextPriceDiff[0] + np.exp(T_next*R_next)*nextPriceDiff[1]
    # 计算不同到期日期权对于VIX的贡献
    near_sigma = calSigmaSquare(optionsNearTerm, near_F, R_near, T_near)
    next_sigma = calSigmaSquare(optionsNextTerm, next_F, R_next, T_next)

    # 利用两个不同到期日的期权对VIX的贡献sig1和sig2，
    # 已经相应的期权剩余到期时间T1和T2；
    # 差值得到并返回VIX指数(%)
    w = (T_next - 30.0/365.0)/(T_next - T_near)
    vix = T_near*w*near_sigma + T_next*(1 - w)*next_sigma
    return 100*np.sqrt(vix*365.0/30.0)

def getHistVIX(beginDate, endDate):
    # 计算历史一段时间内的VIX指数并返回
    optionVarSecID = u"510050.XSHG"
    cal = Calendar('China.SSE')
    dates = cal.bizDatesList(beginDate, endDate)
    dates = map(Date.toDateTime, dates)
    histVIX = pd.DataFrame(0.0, index=dates, columns=['VIX'])
    histVIX.index.name = 'date'
    for date in histVIX.index:
        histVIX['VIX'][date] =  calDayVIX(optionVarSecID, Date.fromDateTime(date))
    return histVIX
```

```
def getDayVIX(date):
    optionVarSecID = u"510050.XSHG"
    return calDayVIX(optionVarSecID, date)
```

2. VIX指数与华夏上证50ETF基金的走势对比

```
secID = '510050.XSHG'
begin = Date(2015, 2, 9)
end = Date(2015, 7, 23)

# 历史VIX
histVIX = getHistVIX(begin, end)

# 华夏上证50ETF
etf = DataAPI.MktFunddGet(secID, beginDate=begin.toISO().replace('-', ''), endDate=end.toISO().replace('-', ''), field=['tradeDate', 'closePrice'])
etf['tradeDate'] = pd.to_datetime(etf['tradeDate'])
etf = etf.set_index('tradeDate')
```

```
font.set_size(12)
pylab.figure(figsize = (16,8))

ax1 = histVIX.plot(x=histVIX.index, y='VIX', style='r')
ax1.set_xlabel(u'日期', fontproperties=font)
ax1.set_ylabel(u'VIX(%)', fontproperties=font)

ax2 = ax1.twinx()
ax2.plot(etf.index,etf.closePrice)
ax2.set_ylabel(u'ETF Price', fontproperties=font)

<matplotlib.text.Text at 0x5a66390>
```



关于VIX，比较成熟的美国市场中，标普500指数和相应的VIX之间呈负相关性。具体可以参照CBOE的数据：http://www.cboe.com/micro/vix/part3.aspx

这可以理解为：

- 当VIX越高时，表示市场参与者预期后市波动程度会更加激烈，所以谨慎持仓，甚至逐渐减仓；
- 相反，VIX越低时，市场参与者预期后市波动程度会趋于缓和，开始放心投资股市。

上图中的中国市场VIX指数与华夏上证50ETF走势对比中，我们不难发现以下几点：

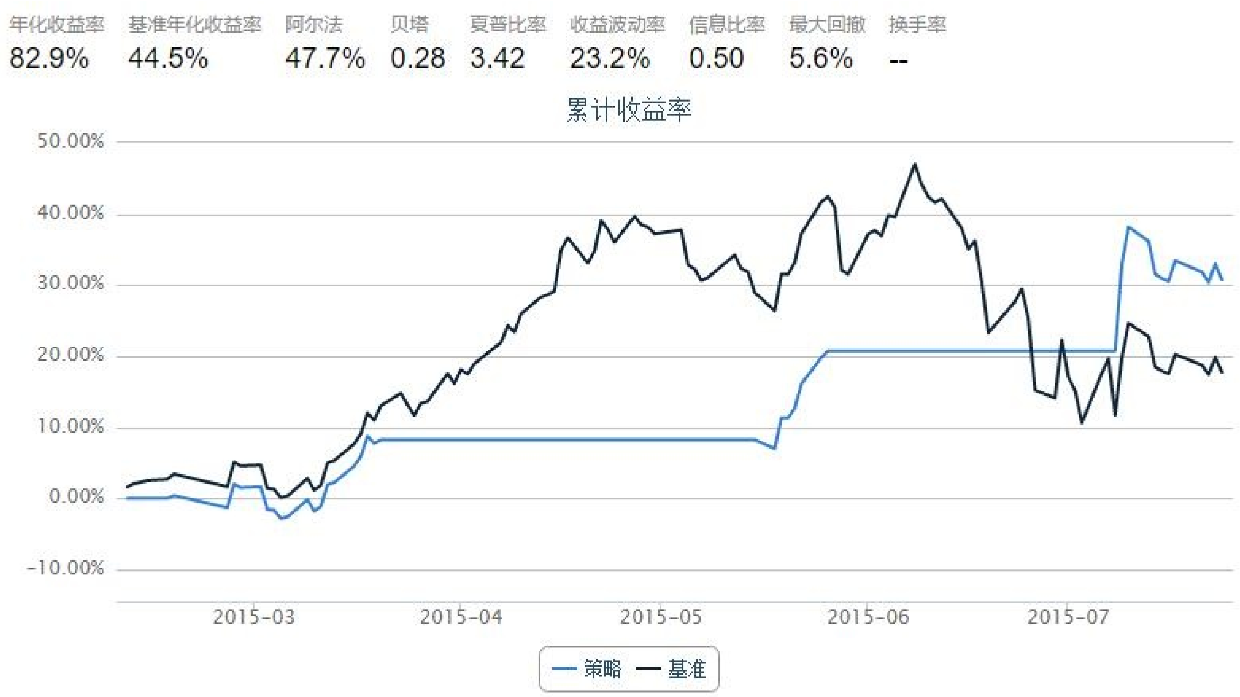
- 上证50ETF期权于2月9日上市，之后一个月VIX稳定在低位运行，同时市场也表现出稳定的态势
- 3月下旬到5月初一段时间，VIX指数显著上升，表示市场认为后期震荡会加剧，但这种恐慌淹没在牛市大潮中
- 5月到6月VIX高位运行，但似乎没有引起市场的足够重视
- 6月中的股市大跌开始后，VIX指数快速上升到接近60
- 7月时候，市场认可国家救市决心，VIX开始从高位迅速下降，股指也日趋稳定

可以看出，VIX指数在和股指的并驾齐驱中总是慢人一步，没法充分表现出股指在六月极高位时候市场的不安；实际上，国内期权市场建立不足半年，期权流动性并不够大，导致基于期权市场的VIX指数对于中国股市的预测并不如成熟市场一样流畅

3. 基于VIX指数的择时策略示例

```
start = datetime(2015, 2, 9)                  # 回测起始时间
end   = datetime(2015, 7, 26)                 # 回测结束时间
benchmark = '510050.XSHG'                      # 策略参考标准
universe = ['510050.XSHG']      # 股票池
```

```
capital_base = 100000     # 起始资金
commission = Commission(0.0,0.0)


window_short = 1
window_long = 5
longest_history = 1
SD = 0.08

histVIX['short_window'] = pd.rolling_mean(histVIX['VIX'], window
=window_short)
histVIX['long_window'] = pd.rolling_mean(histVIX['VIX'], window=
window_long)

def initialize(account):                      # 初始化虚拟账户状态
    account.fund = universe[0]

def handle_data(account):                # 每个交易日的买入卖出指令
    hist = account.get_history(longest_history)
    fund = account.fund

     #  获取回测当日的前一天日期
    dt = Date.fromDateTime(account.current_date)
    cal = Calendar('China.IB')
    lastTDay = cal.advanceDate(dt,'-1B',BizDayConvention.Precedi
ng) #计算出前一个交易日期
    last_day_str = lastTDay.strftime("%Y-%m-%d")

    # 计算买入卖出信号
    try:
        short_mean = histVIX['short_window'].loc[last_day_str] #
 计算短均线值
        long_mean = histVIX['long_window'].loc[last_day_str]   #
 计算长均线值
        long_flag = True if (short_mean - long_mean) < -SD * lon
g_mean else False
        short_flag = True if (short_mean - long_mean) > SD * lon
g_mean else False
    except:
        long_flag = False
        short_flag = False

    if long_flag:
        if account.position.secpos.get(fund, 0) == 0:
            # 空仓时全仓买入，买入股数为100的整数倍
            approximationAmount = int(account.cash / hist[fund][
'closePrice'][-1]/100.0) * 100
            order(fund, approximationAmount)
    elif short_flag:
        # 卖出时，全仓清空
        if account.position.secpos.get(fund, 0) >= 0:
            order_to(fund, 0)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
82.9%	44.5%	47.7%	0.28	3.42	23.2%	0.50	5.6%	--

可以看出：

- 基于VIX指数高位时空仓、低位时进场的策略，可以比较有效地避开股指大跌的风险
- 但由于国内期权市场流动性不足，VIX指数并不能有效反应市场的情绪，导致我们也错过了很多牛市的蛋糕

简单低波动率指数

> 来源：https://uqer.io/community/share/5566a9b8f9f06c6641e97aea

金融市场的波动性加剧，为了提供更好的下行保护，低波动率的Smart Beta策略受到了广泛的欢迎

代表指数

S&P 500 Low Volatility Index

目标指数

HS300

选股

计算目标指数股票池中样本股过去100个交易日中的历史波动率，并挑选其中波动率最低的50只股票作为指数的成分股

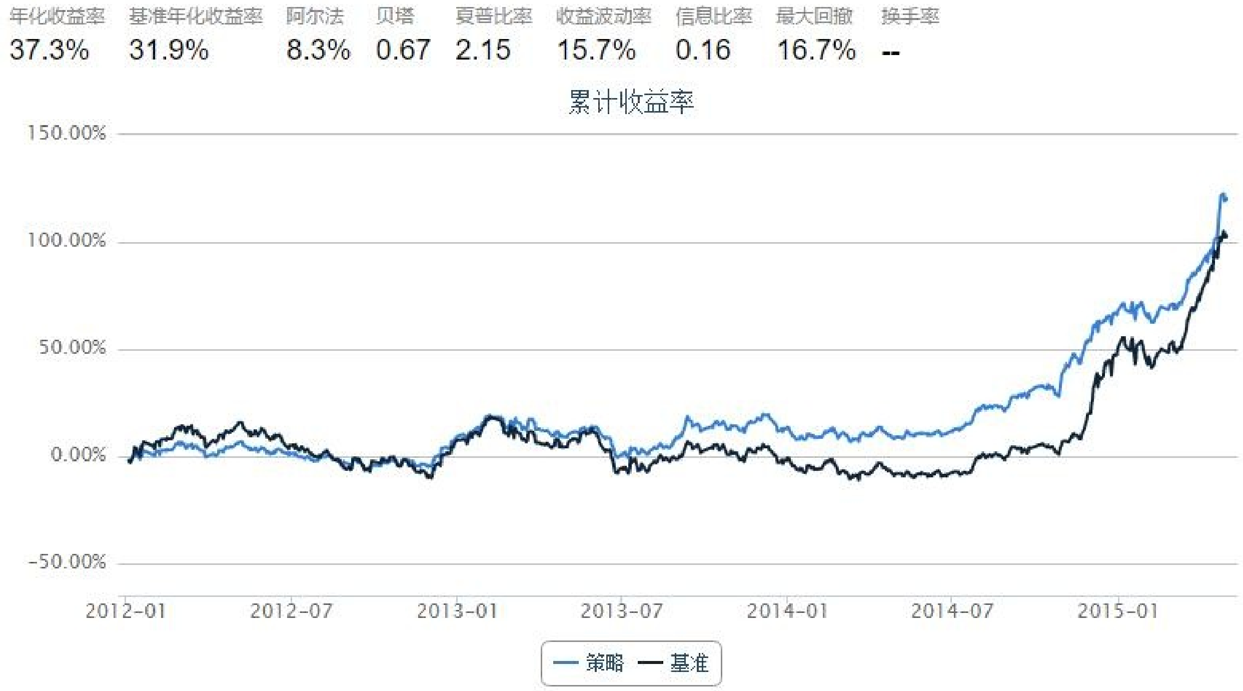
加权

与传统指数市值加权不同，本指数根据股票波动率倒数为个股权重

实现细节

通过 `DataAPI.EquRetudGet` 获取不考虑现金红利再投资情况下的每日收益率，波动率为调仓前100个交易日的日收益率标准差

```
import numpy as np
import pandas as pd
start = '2012-01-01'                       # 回测起始时间
end = '2015-05-01'                         # 回测结束时间
benchmark = 'HS300'                        # 策略参考标准
universe = set_universe('HS300')  # 证券池，回测支持股票和基金
capital_base = 10000000                      # 起始资金
refresh_rate = 100                           # 调仓频率，即每 refr
esh_rate 个交易日执行一次 handle_data() 函数

cal = Calendar('China.SSE')

def initialize(account):                   # 初始化虚拟账户状态
    pass

def handle_data(account):                  # 每个交易日的买入卖出指令

    volatility_res = {}
    cal_today = Date.fromDateTime(account.current_date)
    start_day = cal.advanceDate(cal_today, '-101B', BizDayConven
tion.Following)
    yesterday = cal.advanceDate(cal_today, '-1B', BizDayConventi
on.Following)

    for stk in universe:
        try:
            data = DataAPI.EquRetudGet(ticker=stk[:6], beginDate
=Date.toDateTime(start_day).strftime('%Y%m%d'), endDate=Date.toD
ateTime(yesterday).strftime('%Y%m%d'), field=['ticker',"dailyRet
urnNoReinv"])
            revenue = data['dailyReturnNoReinv']
            volatility_res[stk] = np.std(revenue)
        except:
            universe.remove(stk)

    res = pd.Series(volatility_res).order()[:50]
    temp = np.ones(50)
    res = np.divide(temp, res)
    weight_sum = res.values.sum()
    order_list = dict(res/weight_sum)

    for stk in account.valid_secpos:
        order_to(stk, 0)

    for s, weight in order_list.iteritems():
        if account.referencePrice[s] == 0:
            continue
        order(s, capital_base*weight/account.referencePrice[s])
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
37.3%	31.9%	8.3%	0.67	2.15	15.7%	0.16	16.7%	--

```
print "Benchmark Volatility : ", perf['benchmark_volatility']
print "Index Volatility : ", perf['volatility']

Benchmark Volatility :  0.213927304422
Index Volatility :  0.156413355501
```

结果分析

通过以上结果我们可以看到，该策略alpha极小，beta较大，并显著减小了波动率

10.3 Arch/Garch 模型 · 如何使用优矿进行 GARCH 模型分析

> 来源：https://uqer.io/community/share/56209478f9f06c4c5e2fb5f1

ARCH 建模示例

小弟近来学习波动率建模相关知识，正巧发现优矿中有ARCH包，所以通过翻译ARCH包文档中的示例，来学习相关函数用法。

翻译有不少不准确的地方，请大家指出，我会及时改进。

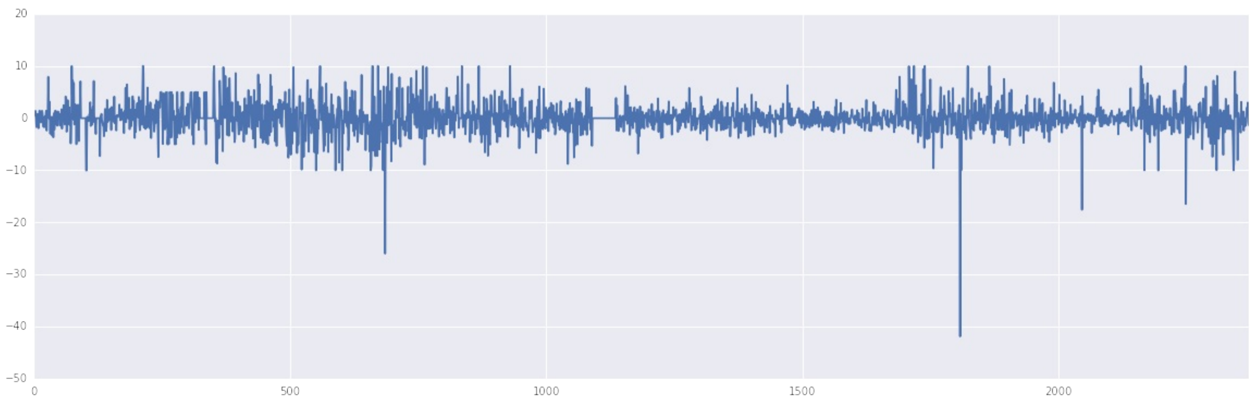
```
import numpy as np
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt

try:
   import seaborn
except ImportError:
    pass
```

一、准备工作

这个例子中使用通联DataAPI提供的行情数据。以平安银行为例。

下面画出了平安银行从1990年到2015年收益率情况。

```
import datetime as dt
st = dt.datetime(1990,1,1)
en = dt.datetime(2015,6,30)

data = DataAPI.MktEqudGet(secID=u"",ticker=u"000001",tradeDate=u"",beginDate=u"",endDate=u"",field=u"ticker,secShortName,tradeDate,closePrice,PE",pandas="1")

# data.index = data.tradeDate
returns = 100 * data['closePrice'].pct_change().dropna()
figure = returns.plot(figsize=(20,6))
```



二、实现常见模型

描述一个模型最简单的方法就是使用Python的类库 `arch.arch_model` 。使用这个类库可以实现大多数常见的模型。

简单的调用 `arch` 类库会得到一个均值恒定，误差符合正态分布，符合 `GARCH(1,1)` 波动率过程的模型。

$$r_t = \mu + \epsilon_t$$

$$\sigma_t^2 = \omega + \alpha\epsilon_{t-1}^2 + \beta\sigma_{t-1}^2$$

$$\epsilon_t = \sigma_t e_t, \ \ e_t \sim N(0,1)$$

通过调用 `fit` 方法，可以对这个模型进行估计。可选输入项 `update_freq` 控制优化器输出结果的频率， `disp` 控制是否返回收敛相关信息。返回结果直接提供了估计的参数值和相关数量，同时包含估计结果的摘要信息。

GARCH (均值恒定)

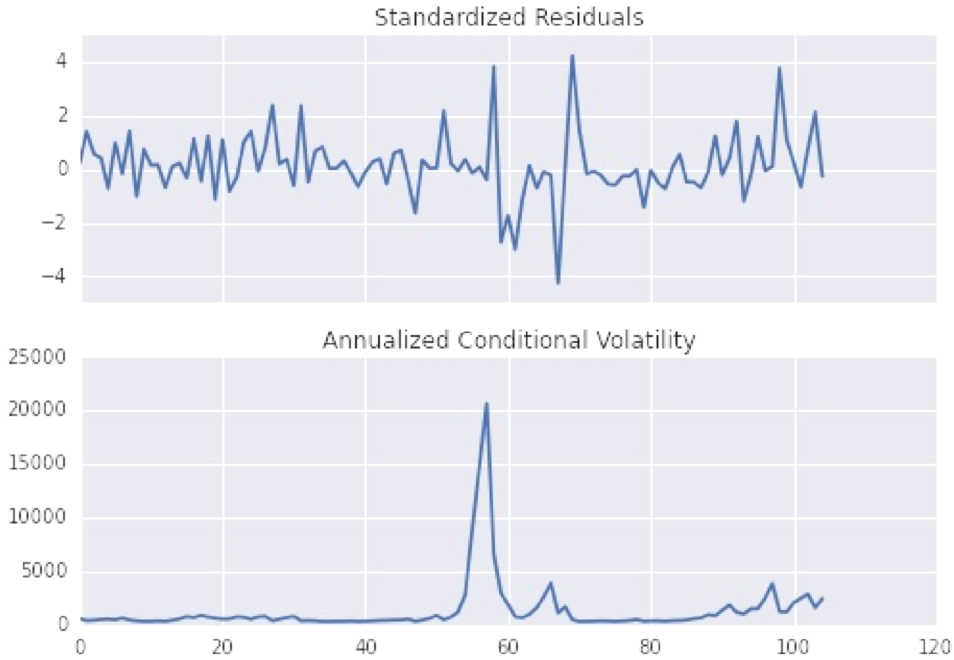
使用默认选项，可以生成一个均值恒定，误差符合正态分布，同时符合 `GARCH(1,1)` 条件方差的模型。

通过拟合获得模型的参数，下面展示拟合结果的摘要。

```
from arch import arch_model
am = arch_model(returns)
res = am.fit(update_freq=5)
print(res.summary())

Iteration:      5,   Func. Count:     38,   Neg. LLF: 5787.77526
93
Iteration:     10,   Func. Count:     75,   Neg. LLF: 5785.39088
499
Optimization terminated successfully.    (Exit mode 0)
            Current function value: 5785.0196556
            Iterations: 14
            Function evaluations: 101
            Gradient evaluations: 14
                     Constant Mean - GARCH Model Results

================================================================
==============
Dep. Variable:             closePrice   R-squared:
        -0.000
Mean Model:             Constant Mean   Adj. R-squared:
        -0.000
Vol Model:                      GARCH   Log-Likelihood:
      -5785.02
Distribution:                  Normal   AIC:
       11578.0
Method:            Maximum Likelihood   BIC:
       11601.1
                                        No. Observations:
          2373
Date:                Fri, Oct 16 2015   Df Residuals:
          2369
Time:                        11:08:36   Df Model:
             4
                                  Mean Model

================================================================
==============
                 coef    std err          t      P>|t|          95
.0% Conf. Int.
----------------------------------------------------------------
--------------
mu             0.0349  2.825e-03     12.348  4.985e-35   [2.935e
-02,4.042e-02]
                               Volatility Model

================================================================
==============
                 coef    std err          t      P>|t|          95
.0% Conf. Int.
----------------------------------------------------------------
--------------
omega          0.0383  2.763e-03     13.861  1.085e-43   [3.288e
-02,4.371e-02]
alpha[1]       0.0287  2.439e-04    117.761      0.000   [2.825e
-02,2.920e-02]
beta[1]        0.9694  4.516e-04   2146.801      0.000       [
0.969,  0.970]
================================================================
==============

Covariance estimator: robust
```

`plot()` 函数可以快速展示 标的的标准偏差和条件波动率。

```
fig = res.plot(annualize='D')
```



GJR-GARCH

`arch_model` 在构建模型时，还可以添加附加参数。在这个例子中，设置 `o` 为1， 即包含了非对称冲击的一阶滞后项，从而将原GARCH模型转换为一个GJR-GARCH模型。新的模型具有动态方差，由下面公式给出：

$$\sigma_t^2 = \omega + \alpha\epsilon_{t-1}^2 + \gamma\epsilon_{t-1}^2 I_{[\epsilon_{t-1}<0]} + \beta\sigma_{t-1}^2$$

其中， `I` 是一个指标函数，当它的输入参数为真时，返回为1.

对数似然函数值改进了非对称项引入的平稳性问题，从而使参数估计具有很高的显著性。

```
am = arch_model(returns, p=1, o=1, q=1)
res = am.fit(update_freq=5, disp='off')
print(res.summary())
```

TARCH/ZARCH

TARCH模型 (又称为 ZARCH模型) 是对波动率的绝对值进行建模. 使用该模型时，需要在 `arch_model` 建构函数中，设置 `power=1.0` 。因为默认的阶数为2，对应的是用平方项表示的方差变化过程。

TARCH model的波动率过程由下面公式给出：

$$\sigma_t = \omega + \alpha|\epsilon_{t-1}| + \gamma|\epsilon_{t-1}| I_{[\epsilon_{t-1}<0]} + \beta\sigma_{t-1}|$$

更一般的情况，模型的动态波动率代码 `κ` 阶。这种情况由下面的公式给出

$$\sigma_t^\kappa = \omega + \alpha|\epsilon_{t-1}|^\kappa + \gamma|\epsilon_{t-1}|^\kappa I_{[\epsilon_{t-1}<0]} + \beta\sigma_{t-1}^\kappa$$

其中，条件方差为$(\sigma_t^\kappa)^{2/\kappa}$

尽管对数似然函数值变动范围更小，TARCH 模型还是对拟合过程做了改进。

```
am = arch_model(returns, p=1, o=1, q=1, power=1.0)
res = am.fit(update_freq=5)
print(res.summary())

Iteration:      5,   Func. Count:     45,   Neg. LLF: 5765.36462
439
Iteration:     10,   Func. Count:     84,   Neg. LLF: 5758.70411
096
Iteration:     15,   Func. Count:    121,   Neg. LLF: 5758.63601
597
Optimization terminated successfully.    (Exit mode 0)
            Current function value: 5758.6360268
            Iterations: 15
            Function evaluations: 121
            Gradient evaluations: 15
                  Constant Mean - TARCH/ZARCH Model Results

================================================================
==============
Dep. Variable:             closePrice   R-squared:
        -0.000
Mean Model:             Constant Mean   Adj. R-squared:
        -0.000
Vol Model:                TARCH/ZARCH   Log-Likelihood:
      -5758.64
Distribution:                  Normal   AIC:
       11527.3
Method:            Maximum Likelihood   BIC:
       11556.1
                                        No. Observations:
          2373
Date:                Fri, Oct 16 2015   Df Residuals:
          2368
Time:                        12:50:24   Df Model:
             5
                                  Mean Model

================================================================
==============
                 coef    std err          t      P>|t|         95
.0% Conf. Int.
-----------------------------------------------------------------
--------------
mu             0.0625  4.323e-03     14.456  2.296e-47   [5.402e
-02,7.096e-02]
                             Volatility Model

=================================================================
==============
                 coef    std err          t      P>|t|         95
.0% Conf. Int.
-----------------------------------------------------------------
--------------
omega          0.0457  4.933e-03      9.257  2.107e-20   [3.599e
-02,5.533e-02]
alpha[1]       0.0594  1.012e-03     58.701      0.000   [5.742e
-02,6.139e-02]
gamma[1]      -0.0184  5.771e-04    -31.882 4.768e-223 [-1.953e-
02,-1.727e-02]
beta[1]        0.9498  2.510e-03    378.466      0.000        [
0.945,  0.955]
=================================================================
==============

Covariance estimator: robust
```

学生T分布误差

金融资产回报率的分布往往体现出肥尾现象，学生`T`分布是一种简单的方法，可以用来捕捉这种特性。在调用`arch_model`构建函数时，可以将概率分布从正态分布转换为学生T分布。

标准化的新息展示出，分布函数具有一个将近10个估计自由度的肥尾。 对数似然函数值同样有大的改善。

```
am = arch_model(returns, p=1, o=1, q=1, power=1.0, dist='Student
sT')
res = am.fit(update_freq=5)
print(res.summary())

Iteration:      5,   Func. Count:     48,   Neg. LLF: 5522.16193
119
Iteration:     10,   Func. Count:     93,   Neg. LLF: 5475.09377
571
Iteration:     15,   Func. Count:    138,   Neg. LLF: 5451.04968
458
Iteration:     20,   Func. Count:    179,   Neg. LLF: 5435.39156
625
Iteration:     25,   Func. Count:    223,   Neg. LLF: 5434.83467
```

```
797
Iteration:     30,   Func. Count:    269,   Neg. LLF: 5434.83086
355
Optimization terminated successfully.    (Exit mode 0)
            Current function value: 5434.83085822
            Iterations: 33
            Function evaluations: 304
            Gradient evaluations: 33
                     Constant Mean - TARCH/ZARCH Model Results

================================================================
====================
Dep. Variable:                   closePrice   R-squared:
              -0.001
Mean Model:                   Constant Mean   Adj. R-squared:
              -0.001
Vol Model:                      TARCH/ZARCH   Log-Likelihood:
            -5434.83
Distribution:      Standardized Student's t   AIC:
             10881.7
Method:                  Maximum Likelihood   BIC:
             10916.3
                                              No. Observations:
                2373
Date:                      Fri, Oct 16 2015   Df Residuals:
                2367
Time:                              13:04:54   Df Model:
                   6
                                   Mean Model

================================================================
===============
                  coef    std err          t      P>|t|        9
5.0% Conf. Int.
----------------------------------------------------------------
---------------
mu         -1.3518e-08  1.493e-10    -90.539      0.000 [-1.381e
-08,-1.323e-08]
                                Volatility Model

================================================================
===============
                  coef    std err          t      P>|t|        9
5.0% Conf. Int.
----------------------------------------------------------------
---------------
omega           0.0894      0.304      0.294      0.769       [
-0.507,  0.686]
alpha[1]        0.1185  9.143e-03     12.962  2.004e-38       [
 0.101,  0.136]
gamma[1]   -2.9293e-03  4.117e-03     -0.712      0.477  [-1.100
e-02,5.139e-03]
beta[1]         0.8829  9.652e-02      9.148  5.803e-20       [
 0.694,  1.072]
                                 Distribution

================================================================
==============
                 coef    std err          t      P>|t|        95
.0% Conf. Int.
----------------------------------------------------------------
--------------
nu             3.6773      0.142     25.919 4.047e-148       [
3.399,  3.955]
================================================================
==============

Covariance estimator: robust
```

使用固定参数

在一些场景下，相比估计出来的参数，使用固定参数可能更让人感兴趣。

使用 `arch_model` 的 `fix()` 可以同样生成一个模型。除了没有与推断相关的值(标准差，t统计量等)的差别之外，新的模型和通常的模型没有什么差别。

在这个例子中，将参数固定设置为之前估计模型的对称版本。

```
fixed_res = am.fix([0.0235, 0.01, 0.06, 0.0, 0.9382, 8.0])
print(fixed_res.summary())

                      Constant Mean - TARCH/ZARCH Model Results

================================================================
=====================
Dep. Variable:                    closePrice   R-squared:
                   --
Mean Model:                    Constant Mean   Adj. R-squared:
                   --
Vol Model:                       TARCH/ZARCH   Log-Likelihood:
             -5579.95
Distribution:       Standardized Student's t   AIC:
              11171.9
Method:            User-specified Parameters   BIC:
              11206.5
                                               No. Observations:
                 2373
Date:                       Fri, Oct 16 2015

Time:                               13:04:57

      Mean Model
=====================
                 coef
---------------------
mu             0.0235
   Volatility Model
=====================
                 coef
---------------------
omega          0.0100
alpha[1]       0.0600
gamma[1]       0.0000
beta[1]        0.9382
     Distribution
=====================
                 coef
---------------------
nu             8.0000
=====================

Results generated with user-specified parameters.
Since the model was not estimated, there are no std. errors.
```

```
import pandas as pd
df = pd.concat([res.conditional_volatility,fixed_res.conditional
_volatility],1)
df.columns = ['Estimated', 'Fixed']
df.plot()

<matplotlib.axes.AxesSubplot at 0x6b5cd90>
```

三、通过多个组件模块创建模型

模型同样可以使用代表三类模型的 `arch` 模块进行系统性的构建。

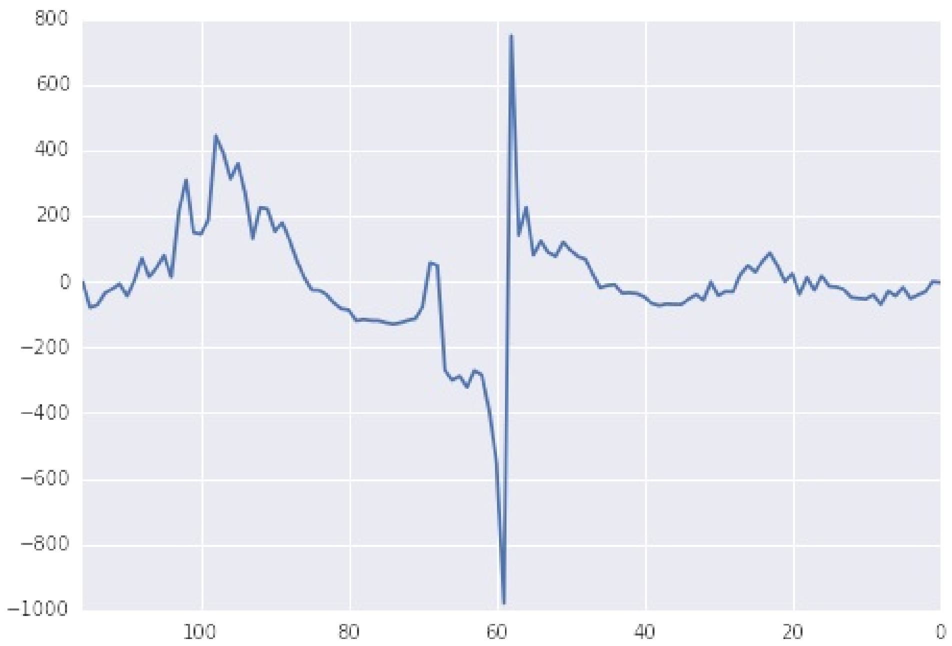
- A mean model (arch.mean)
  - Zero mean (ZeroMean) - useful if using residuals from a model estimated separately
  - Constant mean (ConstantMean) - common for most liquid financial assets
  - Autoregressive (ARX) with optional exogenous regressors
  - Heterogeneous (HARX) autoregression with optional exogenous regressors
  - Exogenous regressors only (LS)
- A volatility process (arch.volatility)
  - ARCH (ARCH)
  - GARCH (GARCH)
  - GJR-GARCH (GARCH using o argument)
  - TARCH/ZARCH (GARCH using power argument set to 1)
  - Power GARCH and Asymmetric Power GARCH (GARCH using power)
  - Heterogeneous ARCH (HARCH)
  - Parameterless Models
    - Exponentially Weighted Moving Average Variance, known as RiskMetrics (EWMAVariance)
    - Weighted averages of EWMAs, known as the RiskMetrics 2006 methodology (RiskMetrics2006)
- A distribution (arch.distribution)
  - Normal (Normal)
  - Standardized Students's T (StudentsT)

Mean Models 均值模型

第一种选择是使用均值模型。 对于很多流动性充足的金融资产来说， 恒定均值(甚至是0均值)的模型就足够了。

对于其他一些时间序列，若通货膨胀率数据，可能需要更加复杂的模型。 下面的例子使用了中国居民消费价格指数(CPI)数据。这些数据由通联DataAPI提供。

```
core_cpi = DataAPI.ChinaDataCPIGet(indicID=u"M030000003",indicNa
me=u"",beginDate=u"20050101",endDate=u"",field=u"",pandas="1")
ann_inflation = 100 * core_cpi.sort(columns='periodDate').dataVa
lue.pct_change(12).dropna()
fig = ann_inflation.plot()
fig

<matplotlib.axes.AxesSubplot at 0x6e6a090>
```



所有的均值模型都派生自恒定方差、正态分布误差的基础模型。

对于 `ARX` 模型， `lags` 输入参数制定了模型需要包括的滞后项阶数。

```
from arch.univariate import ARX
ar = ARX(ann_inflation, lags = [1, 3, 12])
print(ar.fit().summary())

                     AR - Constant Variance Model Results

================================================================
==============
Dep. Variable:              dataValue   R-squared:
         0.264
Mean Model:                        AR   Adj. R-squared:
         0.242
Vol Model:          Constant Variance   Log-Likelihood:
      -685.867
Distribution:                  Normal   AIC:
       1381.73
Method:            Maximum Likelihood   BIC:
       1395.00
                                        No. Observations:
```

```
           105
Date:                Fri, Oct 16 2015   Df Residuals:
           100
Time:                        13:21:29   Df Model:
             5
                                    Mean Model

================================================================
=================
                    coef    std err          t      P>|t|
 95.0% Conf. Int.
----------------------------------------------------------------
-----------------
Const            -1.4076    254.081 -5.540e-03      0.996  [-4.9
94e+02,4.966e+02]
dataValue[1]      0.4233      0.112      3.777  1.590e-04
[  0.204,  0.643]
dataValue[3]      0.1590  1.423e-02     11.173  5.523e-29
[  0.131,  0.187]
dataValue[12]    -0.0117  1.050e-03    -11.133  8.695e-29 [-1.37
4e-02,-9.628e-03]
                               Volatility Model

================================================================
==============
                 coef    std err          t      P>|t|         95
.0% Conf. Int.
----------------------------------------------------------------
--------------
sigma2     2.7619e+04  2.259e+08  1.223e-04      1.000  [-4.427e
+08,4.427e+08]
================================================================
==============

Covariance estimator: White's Heteroskedasticity Consistent Esti
mator
```

Volatility Processes 波动率过程

波动率过程可以通过在均值模型中添加 `volatility` 属性来实现。

下面的例子中将模型的波动率设置为 `ARCH(5)`，`update_freq` 和 `disp` 用来约束 `fit()` 进行估计时候的输出内容。

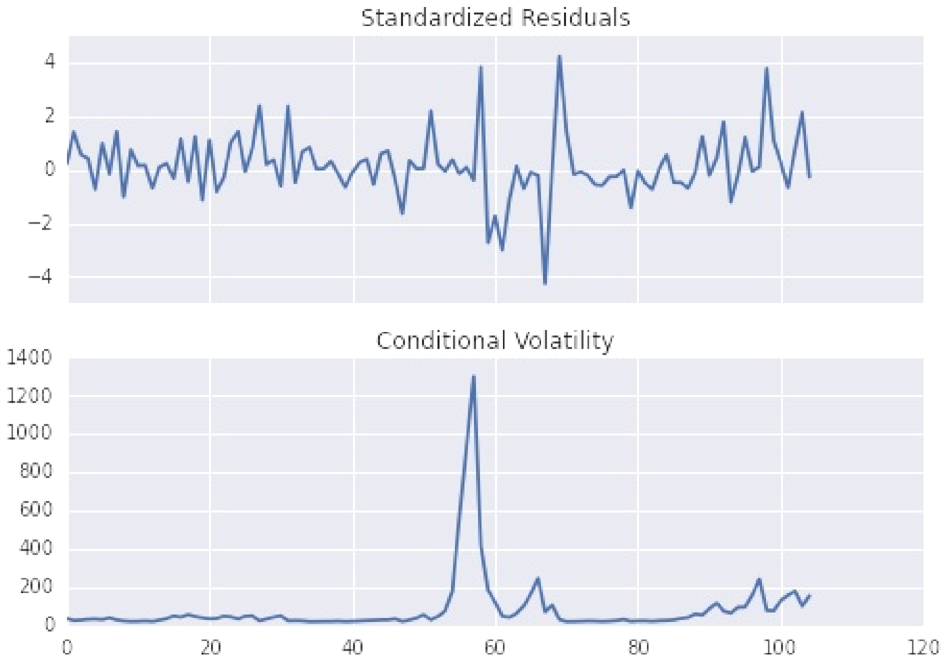
```
from arch.univariate import ARCH, GARCH
ar.volatility = ARCH(p=5)
res = ar.fit(update_freq=0, disp='off')
print(res.summary())

                           AR - ARCH Model Results
```

```
================================================================
==============
Dep. Variable:              dataValue   R-squared:
         0.104
Mean Model:                        AR   Adj. R-squared:
         0.077
Vol Model:                       ARCH   Log-Likelihood:
      -576.003
Distribution:                  Normal   AIC:
       1172.01
Method:            Maximum Likelihood   BIC:
       1198.55
                                        No. Observations:
           105
Date:                Fri, Oct 16 2015   Df Residuals:
            95
Time:                        13:25:52   Df Model:
            10
                                    Mean Model

================================================================
=================
                    coef     std err           t      P>|t|
95.0% Conf. Int.
----------------------------------------------------------------
-----------------
Const           -11.9096      29.775      -0.400      0.689       [
-70.268, 46.449]
dataValue[1]      0.8647   2.093e-02      41.313      0.000
[  0.824,  0.906]
dataValue[3]     -0.0296   9.925e-03      -2.978  2.904e-03 [-4.90
1e-02,-1.010e-02]
dataValue[12]    -0.0154   2.936e-04     -52.419      0.000 [-1.59
7e-02,-1.482e-02]
                                Volatility Model

================================================================
===============
                  coef     std err           t      P>|t|       9
5.0% Conf. Int.
----------------------------------------------------------------
---------------
omega         404.9890   1.164e+05   3.480e-03      0.997  [-2.277
e+05,2.285e+05]
alpha[1]        0.6348   5.176e-02      12.266  1.383e-34       [
0.533,  0.736]
alpha[2]        0.2690   1.767e-02      15.221  2.552e-52       [
0.234,  0.304]
alpha[3]        0.0962   1.161e-02       8.283  1.196e-16     [7.3
41e-02,  0.119]
alpha[4]    2.5205e-10   3.554e-04   7.093e-07      1.000  [-6.965
e-04,6.965e-04]
```

```
alpha[5]    -4.6303e-10  1.178e-05 -3.929e-05      1.000  [-2.310
e-05,2.309e-05]
================================================================
===============

Covariance estimator: robust
```

下面的图标，展示了标准化后的新息和条件波动率的情况。可以看出，虽然进行了标准化处理，一些位置还是出现了较大的误差(通过振幅来看)

```
fig = res.plot()
```



Distributions 概率分布情况

最后提一下，模型的概率分布可以通过设置 `distribution` 属性来修改，将默认的正态分布修改为学生T分布。

学生T分布改进了模型，模型的估计自由度大约在8左右。

备注 1 本文章是将Python ARCH类库的帮助文档进行翻译产生的。

原文地址：
http://nbviewer.ipython.org/github/bashtage/arch/blob/master/examples/univariate_volatility_modeling.ipynb

本文将原文中的数据源替换为优矿/通联DataAPI的数据，方便优矿用户进行参照和分析。

2 文中，多次提到 `iter` ，但这个疑为原作者笔误，故都改为 `update_freq`

3 原文最后一章‘WTI Crude’由于给出的例子和当前的 `arch` 包版本不兼容，无法正确运行，故没有引入

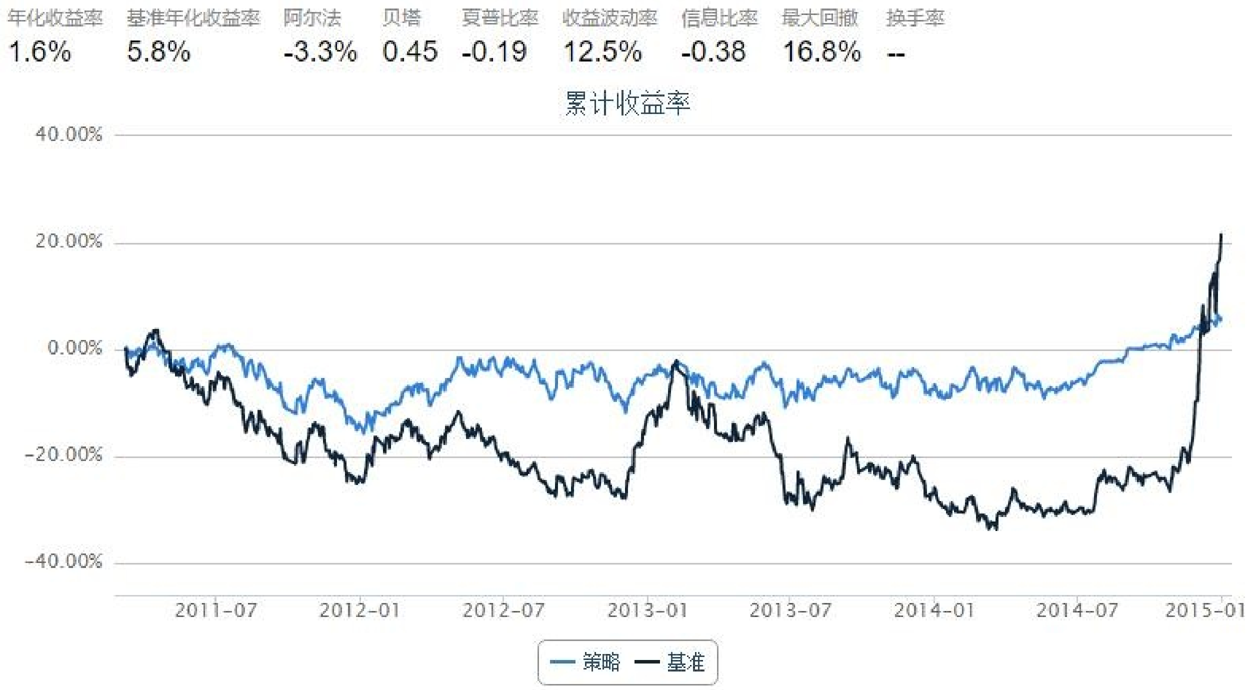
十一 算法交易

11.1 VWAP · Value-Weighted Average Price (VWAP)

> 来源：https://uqer.io/community/share/55462234f9f06c1c3d688033

You can find it in API doc

```
start = '2011-01-01'                           # 回测起始时间
end = '2015-01-01'                             # 回测结束时间
benchmark = 'SH50'                            # 策略参考标准
universe = set_universe('SH50')
capital_base = 100000                         # 起始资金
longest_history = 40                            # handle_data 函数中
可以使用的历史数据最长窗口长度
refresh_rate = 1                              # 调仓频率，即每 refresh
_rate 个交易日执行一次 handle_data() 函数

threshold = 0.03

def initialize(account):                       # 初始化虚拟账户状态
    pass

def handle_data(account):                      # 每个交易日的买入卖出指令

    for s in account.universe:
        try:
            inter = 20
            hist = account.get_symbol_history(s, inter)
        except:
            continue
        vwampvalue = sum(hist['turnoverValue'])/sum(hist['turnov
erVol'])
        if(hist['lowPrice'][-1] < vwampvalue*(1 - threshold)) and
 (s not in account.valid_secpos):
            order(s,100)
        if(hist['lowPrice'][-1] > vwampvalue) and (s in account.
valid_secpos):
            order_to(s,0)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
1.6%	5.8%	-3.3%	0.45	-0.19	12.5%	-0.38	16.8%	--

easy strategy.. bad results....

It's very difficult to get a good alpha......

十二 中高频交易

12.1 order book 分析 · 基于高频 limit order book 数据的短程价格方向预测—— via multi-class SVM

> 来源：https://uqer.io/community/share/5660665bf9f06c6c8a91b1a0

摘要：

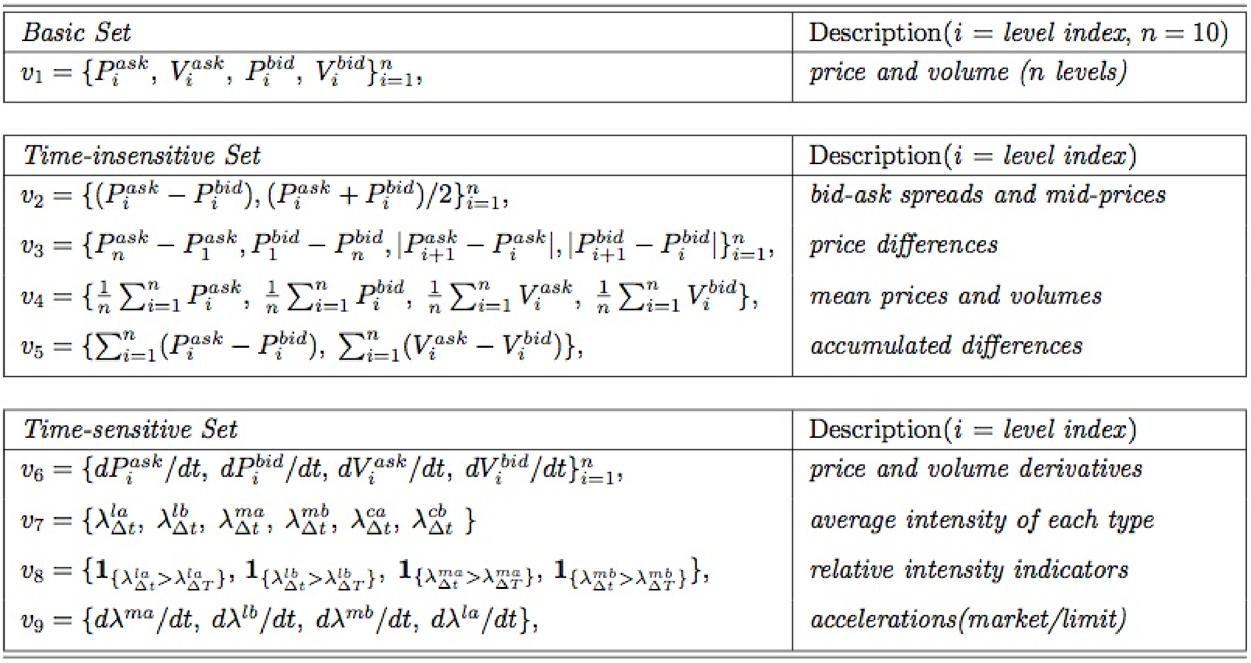
下面的内容是基于文献Modeling high-frequency limit order book dynamics with support vector machines的框架写的，由于高频数据粗粒度依然有限，只能实现了部分内容。若需要完整理解这个问题以及实现方法，请阅读上述的文献。下面我会简单介绍一下整个框架的内容。

模型构造

作者使用Message book以及Order book作为数据来源，通联没有前者的数据，因此后面的部分只涉及到level1买卖5档的order book数据作为模型的输入。这里我只实现了通过order book数据预测mid price的方向，包括向上，向下，以及不变。对于bid-ask spread crossing的方法相似，我暂时就不放上来了。

特征选择

对order book数据做处理后，可以提取到我们需要的特征向量。总的特征分为三类：基本、时间不敏感和时间敏感三类，这里我们能从数据中获得全部的基本和时间不敏感特征，以及部分时间敏感特征，具体的见图片，或者进一步阅读文献。



Basic Set	Description($i = level\ index$, $n = 10$)
$v_1 = \{P_i^{ask},\ V_i^{ask},\ P_i^{bid},\ V_i^{bid}\}_{i=1}^n,$	*price and volume (n levels)*

Time-insensitive Set	Description($i = level\ index$)
$v_2 = \{(P_i^{ask} - P_i^{bid}), (P_i^{ask} + P_i^{bid})/2\}_{i=1}^n,$	*bid-ask spreads and mid-prices*
$v_3 = \{P_n^{ask} - P_1^{ask}, P_1^{bid} - P_n^{bid}, \lvert P_{i+1}^{ask} - P_i^{ask}\rvert, \lvert P_{i+1}^{bid} - P_i^{bid}\rvert\}_{i=1}^n,$	*price differences*
$v_4 = \{\frac{1}{n}\sum_{i=1}^n P_i^{ask},\ \frac{1}{n}\sum_{i=1}^n P_i^{bid},\ \frac{1}{n}\sum_{i=1}^n V_i^{ask},\ \frac{1}{n}\sum_{i=1}^n V_i^{bid}\},$	*mean prices and volumes*
$v_5 = \{\sum_{i=1}^n (P_i^{ask} - P_i^{bid}),\ \sum_{i=1}^n (V_i^{ask} - V_i^{bid})\},$	*accumulated differences*

Time-sensitive Set	Description($i = level\ index$)
$v_6 = \{dP_i^{ask}/dt,\ dP_i^{bid}/dt,\ dV_i^{ask}/dt,\ dV_i^{bid}/dt\}_{i=1}^n,$	*price and volume derivatives*
$v_7 = \{\lambda_{\Delta t}^{la},\ \lambda_{\Delta t}^{lb},\ \lambda_{\Delta t}^{ma},\ \lambda_{\Delta t}^{mb},\ \lambda_{\Delta t}^{ca},\ \lambda_{\Delta t}^{cb}\ \}$	*average intensity of each type*
$v_8 = \{\mathbf{1}_{\{\lambda_{\Delta t}^{la} > \lambda_{\Delta T}^{la}\}},\ \mathbf{1}_{\{\lambda_{\Delta t}^{lb} > \lambda_{\Delta T}^{lb}\}},\ \mathbf{1}_{\{\lambda_{\Delta t}^{ma} > \lambda_{\Delta T}^{ma}\}},\ \mathbf{1}_{\{\lambda_{\Delta t}^{mb} > \lambda_{\Delta T}^{mb}\}}\},$	*relative intensity indicators*
$v_9 = \{d\lambda^{ma}/dt,\ d\lambda^{lb}/dt,\ d\lambda^{mb}/dt,\ d\lambda^{la}/dt\},$	*accelerations(market/limit)*

```
#importing package
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn import svm
from CAL.PyCAL import *

#global parameter for model
date = '20151130'
securityID = '000002.XSHE' #万科A
trainSetNum = 900
testSetNum = 600


#loading LOB data
dataSet = DataAPI.MktTicksHistOneDayGet(securityID=securityID, d
ate=date,pandas='1')

#Features representation
##Basic Set
###V1: price and volume (10 levels)
featV1 = dataSet[['askPrice1','askPrice2','askPrice3','askPrice4'
,'askPrice5','askVolume1','askVolume2','askVolume3','askVolume4',
'askVolume5','bidPrice1','bidPrice2','bidPrice3','bidPrice4','bi
dPrice5','bidVolume1','bidVolume2','bidVolume3','bidVolume4','bi
dVolume5']]
featV1 = np.array(featV1)

##Time-insensitive Set
###V2: bid-ask spread and mid-prices
temp1 = featV1[:,0:5] - featV1[:,10:15]
temp2 = (featV1[:,0:5] + featV1[:,10:15])*0.5
featV2 = np.zeros([temp1.shape[0],temp1.shape[1]+temp2.shape[1]]
)
featV2[:,0:temp1.shape[1]] = temp1
featV2[:,temp1.shape[1]:] = temp2

###V3: price differences
temp1 = featV1[:,4] - featV1[:,0]
temp2 = featV1[:,10] - featV1[:,14]
temp3 = abs(featV1[:,1:5] - featV1[:,0:4])
temp4 = abs(featV1[:,11:15] - featV1[:,10:14])
featV3 = np.zeros([temp1.shape[0],1+1+temp3.shape[1]+temp4.shape[
1]])
featV3[:,0] = temp1
featV3[:,1] = temp2
featV3[:,2:2+temp3.shape[1]] = temp3
featV3[:,2+temp3.shape[1]:] = temp4

###V4: mean prices and volumns
temp1 = np.mean(featV1[:,0:5],1)
temp2 = np.mean(featV1[:,10:15],1)
temp3 = np.mean(featV1[:,5:10],1)
temp4 = np.mean(featV1[:,15:],1)
featV4 = np.zeros([temp1.shape[0],1+1+1+1])
featV4[:,0] = temp1
featV4[:,1] = temp2
featV4[:,2] = temp3
featV4[:,3] = temp4

###V5: accumulated differences
temp1 = np.sum(featV2[:,0:5],1)
temp2 = np.sum(featV1[:,5:10] - featV1[:,15:],1)
featV5 = np.zeros([temp1.shape[0],1+1])
featV5[:,0] = temp1
featV5[:,1] = temp2

##Time-insensitive Set
###V6: price and volume derivatives
temp1 = featV1[1:,0:5] - featV1[:-1,0:5]
temp2 = featV1[1:,10:15] - featV1[:-1,10:15]
temp3 = featV1[1:,5:10] - featV1[:-1,5:10]
temp4 = featV1[1:,15:] - featV1[:-1,15:]
featV6 = np.zeros([temp1.shape[0]+1,temp1.shape[1]+temp2.shape[1
]+temp3.shape[1]+temp4.shape[1]]) #由于差分，少掉一个数据，此处补回
featV6[1:,0:temp1.shape[1]] = temp1
featV6[1:,temp1.shape[1]:temp1.shape[1]+temp2.shape[1]] = temp2
featV6[1:,temp1.shape[1]+temp2.shape[1]:temp1.shape[1]+temp2.sha
pe[1]+temp3.shape[1]] = temp3
featV6[1:,temp1.shape[1]+temp2.shape[1]+temp3.shape[1]:] = temp4

##combining the features
feat = np.zeros([featV1.shape[0],sum([featV1.shape[1],featV2.sha
pe[1],featV3.shape[1],featV4.shape[1],featV5.shape[1],featV6.sha
pe[1]])])
feat[:,:featV1.shape[1]] = featV1
```

```
feat[:,featV1.shape[1]:featV1.shape[1]+featV2.shape[1]] = featV2
feat[:,featV1.shape[1]+featV2.shape[1]:featV1.shape[1]+featV2.shape[1]+featV3.shape[1]] = featV3
feat[:,featV1.shape[1]+featV2.shape[1]+featV3.shape[1]:featV1.shape[1]+featV2.shape[1]+featV3.shape[1]+featV4.shape[1]] = featV4
feat[:,featV1.shape[1]+featV2.shape[1]+featV3.shape[1]+featV4.shape[1]:featV1.shape[1]+featV2.shape[1]+featV3.shape[1]+featV4.shape[1]+featV5.shape[1]] = featV5
feat[:,featV1.shape[1]+featV2.shape[1]+featV3.shape[1]+featV4.shape[1]+featV5.shape[1]:] = featV6

##normalizing the feature
numFeat = feat.shape[1]
meanFeat = feat.mean(axis=1)
meanFeat.shape = [meanFeat.shape[0],1]
stdFeat = feat.std(axis=1)
stdFeat.shape = [stdFeat.shape[0],1]
normFeat = (feat - meanFeat.repeat(numFeat,axis=1))/stdFeat.repeat(numFeat,axis=1)
#print(normFeat)

api.wmcloud.com 443
```

数据标注

选择时间间隔为通联能获取的最小时间间隔（3s），

- 若下一个单位时刻mid price大于此时的mid price，则标注为向上，
- 若下一个单位时刻mid price小于此时的mid price，则标注为向下，
- 若下一个单位时刻mid price等于此时的mid price，则标注为不变，

```
##mid-price trend of dataset:upward(0),downward(1) or stationary(2)
upY = featV2[1:,5] > featV2[:-1,5]
upY = np.append(upY,0)
numUp = sum(upY)
downY = featV2[1:,5] < featV2[:-1,5]
downY = np.append(downY,0)
numDown = sum(downY)
statY = featV2[1:,5] == featV2[:-1,5]
statY = np.append(statY,0)
numStat = sum(statY)

#Y = np.zeros([upY.shape[0],3])
#Y[:,0] = upY
#Y[:,1] = downY
#Y[:,2] = statY

pUp = np.where(upY==1)[0]
pDown = np.where(downY==1)[0]
pStat = np.where(statY==1)[0]
multiY = np.zeros([upY.shape[0],1])
multiY[pUp] = 0
multiY[pDown] = 1
multiY[pStat] = 2

##divide the dataset into trainSet, and testSst
numTrain = 1200
numTest = 500
#rebalance the radio of upward, downward and stationary data
numTrainUp = 250
numTrainDown = 250
numTrainStat = 400
pUpTrain = pUp[:numTrainUp]
pDownTrain = pDown[:numTrainDown]
pStatTrain = pStat[:numTrainStat]

pTrainTemp = np.append(pUpTrain,pDownTrain)
pTrain = np.append(pTrainTemp,pStatTrain)
trainSet = normFeat[pTrain,:]
#trainSet = normFeat[1:numTrain+1,:]
testSet = normFeat[numTrain+1:numTrain+numTest+1,:]

#trainY = Y[1:numTrain+1,:]
trainMultiYTemp = np.append(multiY[pUpTrain],multiY[pDownTrain])
trainMultiY = np.append(trainMultiYTemp,multiY[pStatTrain])
#trainMultiY = multiY[1:numTrain+1]
testMultiY = multiY[numTrain+1:numTrain+numTest+1]
```

分类模型

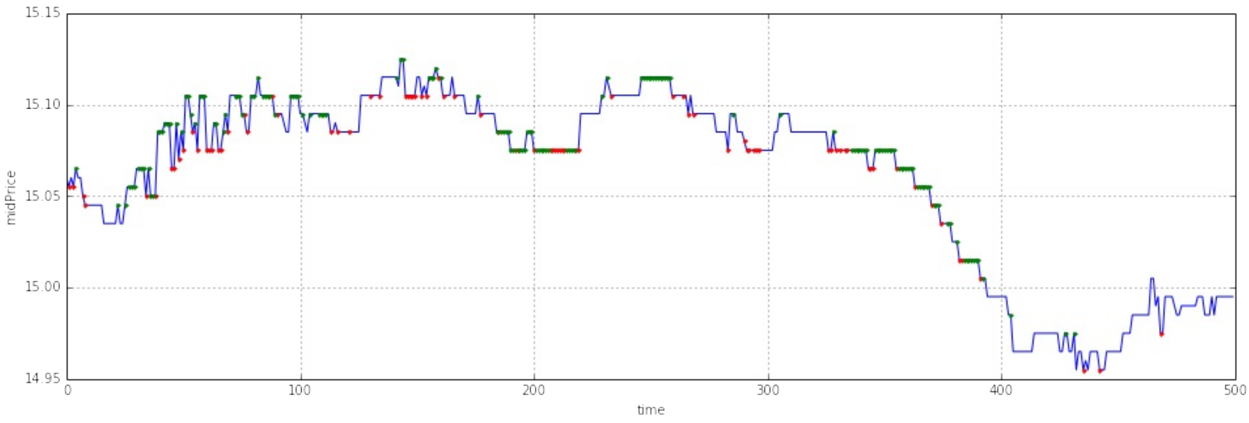
基于one vs all的multi-class SVM，这里我没有对参数做过多调整，因此看到的模型事实上非常简陋。有兴趣的话也可以用forest tree等ML方法尝试。

```
##training a multi-class svm model
Model = svm.LinearSVC(C=2.)
Model.fit(trainSet,trainMultiY)
pred = Model.predict(testSet)
ap = Model.score(testSet,testMultiY)
print(ap)

0.522
```

结果

我这里拿了11月30日的万科A作为数据来源来预测。之所以拿万科A，是因为我从11月上旬就开始看好这只股票，结果在中旬的时候没有拿住，低位没有补进，谁知道月底就起飞了，让我又爱又恨。我在最后画出了预测结果，蓝线是测试集中的mid price时间序列，红点表示模型预测下一时刻方向向上，绿点表示模型预测下一时刻方向向下，没有画点表示预测方向不变。

```
testMidPrice = featV2[numTrain+1:numTrain+numTest+1,5]
pUpTest = np.where(pred==0)[0]
pDownTest = np.where(pred==1)[0]
pStatTest = np.where(pred==2)[0]

plt.figure(figsize=(16,5))
plt.plot(range(numTest),testMidPrice,'b-',pUpTest,testMidPrice[p
UpTest],'r.',pDownTest,testMidPrice[pDownTest],'g.')
plt.grid()
plt.xlabel('time')
plt.ylabel('midPrice')

<matplotlib.text.Text at 0x6f8d2d0>
```



题外话

现在你看到的是一个极为粗糙的东西，原论文的框架远远比这个复杂，包括对训练集的交叉验证，以及数据的更新替代，bid-ask spread crossing，以及基于此的toy策略（当然这么高频的操作在平台上暂时也实现不了:)）等等等等都没有实现。这里我只是选取了前1200个数据作了normalization和rebalance后来预测后500个数据。我现在研二忙成狗，也只能晚上写一写，还得赶着发完论文以后赶紧找实习，所以以后有机会也许再放一个更精细的版本上来。最后感谢通联的朋友特地给我开了历史高频的接口~

12.2 日内交易 · 大盘日内走势 (for 择时)

> 来源：https://uqer.io/community/share/5649b64af9f06c4446b48202

上周统计过周一到周五的涨跌分布，后来又统计了一下股指交割周的周四，竟然只有33.33%上涨 。也是醉了。

统计完日间，再来看下日内，那么大盘日内走势是怎样呢？ 对日内操作有指导吗？

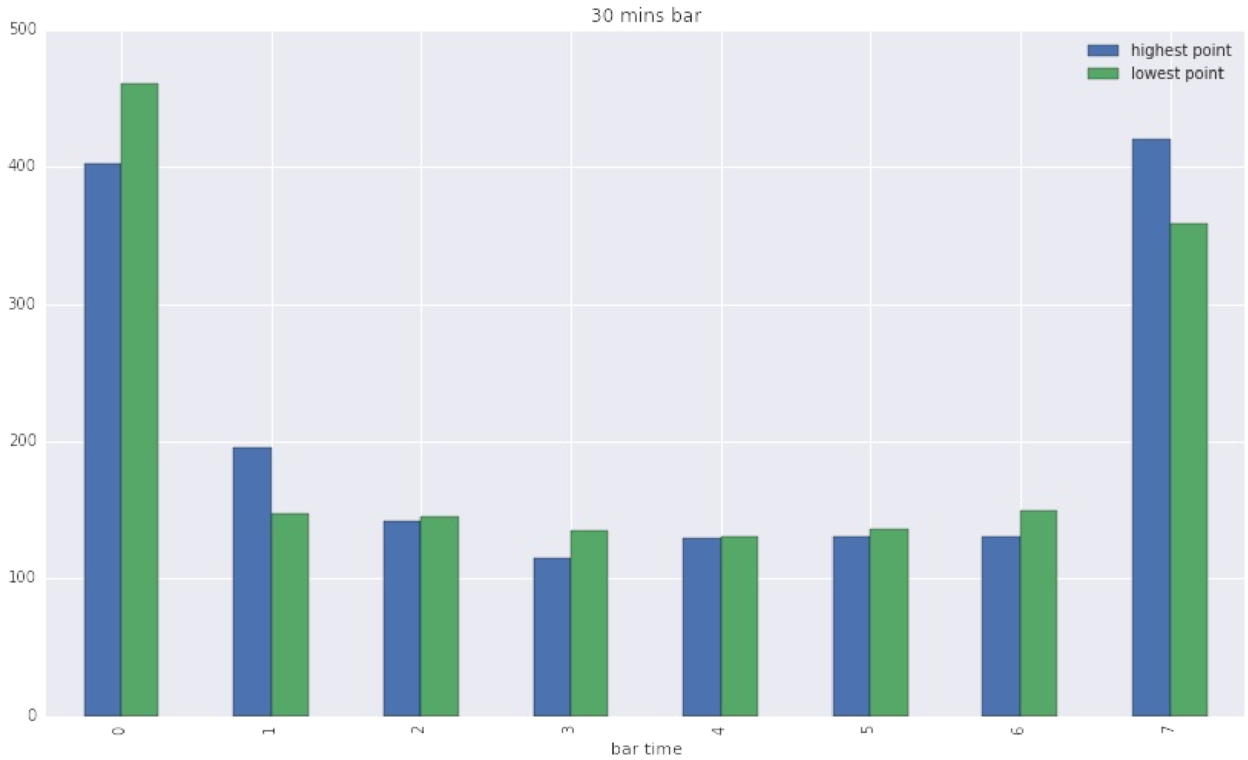
时间紧急，话不多说，上分析过程。

```
# 获取09年以来的上证交易日
import datetime
import seaborn
import pandas as pd

df = DataAPI.TradeCalGet(exchangeCD=u"XSHG",beginDate=u"20090101"
,endDate=datetime.datetime.now().strftime('%Y%m%d'),field=u"cale
ndarDate,isOpen",pandas="1")
trading_days = df[df.isOpen==1].calendarDate.apply(lambda x:x.re
place('-','')).values
trading_days

array(['20090105', '20090106', '20090107', ..., '20151112', '201
51113',
       '20151116'], dtype=object)
```

```
# 获取09年以来的上证指数的分钟线
df = None
for date in trading_days:
    try:
        temp_df = DataAPI.MktBarHistOneDayGet(securityID='000001
.XSHG',date=date, field='barTime,closePrice')[1:]
    except:
        print 'get data error at %s.' %date
        continue
    # 日内打分，1表示最高
    temp_df['rank'] = temp_df.closePrice.rank(ascending=False)
    temp_df['index'] = range(len(temp_df))
    if df is None:
        df = temp_df
    else:
        df = df.append(temp_df)
```

首先看一下30mins线，日内高点和低点的分布图。

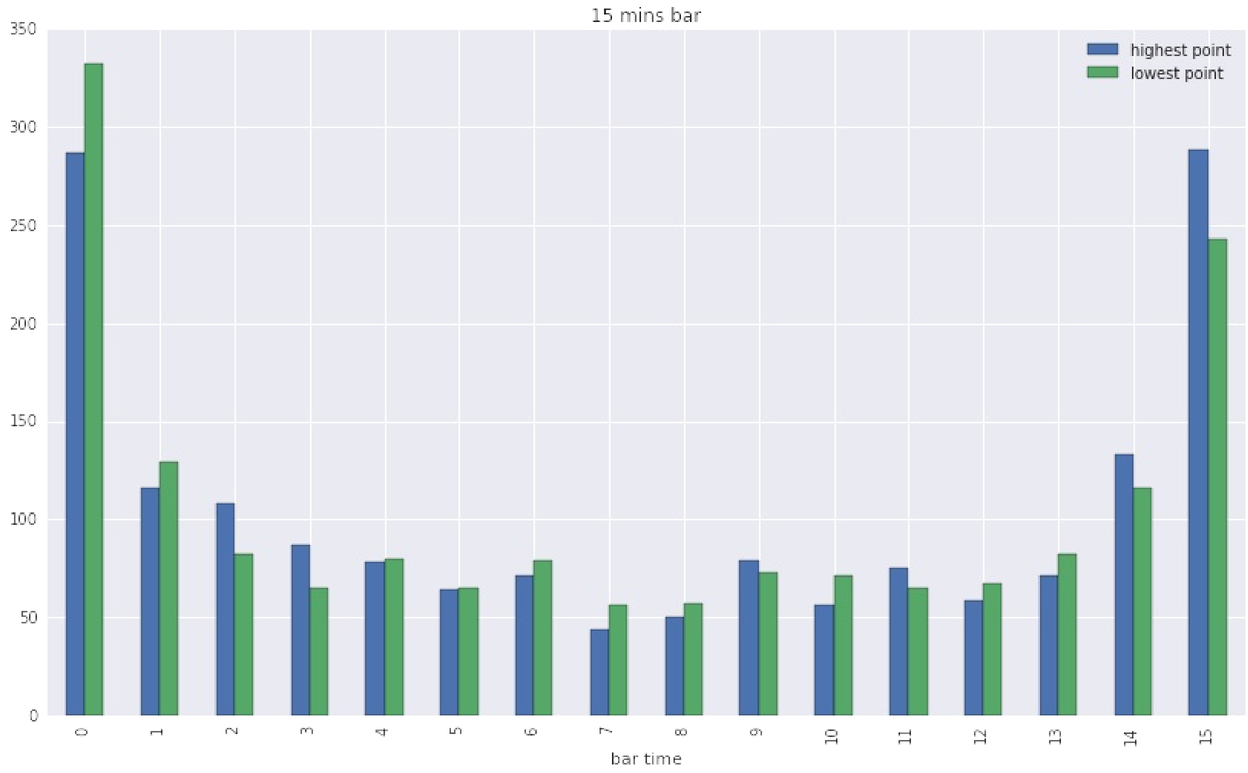
```
bar_length = 30 #30mins bar
def plot(bar_length):
    df['bar time'] = df['index'].apply(lambda x:x/bar_length)
    highest_count = df[df['rank'] == min(df['rank'])].groupby('bar time')['rank'].count()
    lowest_count = df[df['rank'] == max(df['rank'])].groupby('bar time')['rank'].count()

    pd.DataFrame({'highest point':highest_count,'lowest point':lowest_count}).plot(figsize=(14,8),kind='bar', title='%s mins bar' %bar_length)
plot(bar_length)
```



可以看到，日内的最高点和最低点在早盘和尾盘出现频率最高。实际上，确实很多人都会选择在早盘或者尾盘操作。

那15mins和5mins的情况呢？

```
plot(bar_length=15)
plot(bar_length=5)
```



5mins比15mins图更清晰。

越靠近开盘，出现日内低点概率越高；而越临近收盘，冲高概率也越高。极点微笑。

今天（20151116）的走势，正巧是低开高收。

对于日内需要调仓，或者做T，可以关注一下该现象。不做任何买卖建议哦。

完。

十三 Alternative Strategy

13.1 易经、传统文化 · 老黄历诊股

A：听说你今天满仓了？

B：是啊，听老专家说炒股要看黄历，我早上查了，今天宜开市。你呢，仓位如何？

A：策略有卖出信号，我昨天就空仓了。

S：怪不得今天跌停，原来是空头行动了。

最近小散发现了一个管用的择时方法:“看黄历”。4号忌交易，熔断；6号宜开市，红盘。7号宜开市，满仓杀入。然后就有了上面的对话。

等等，好像哪里不对。

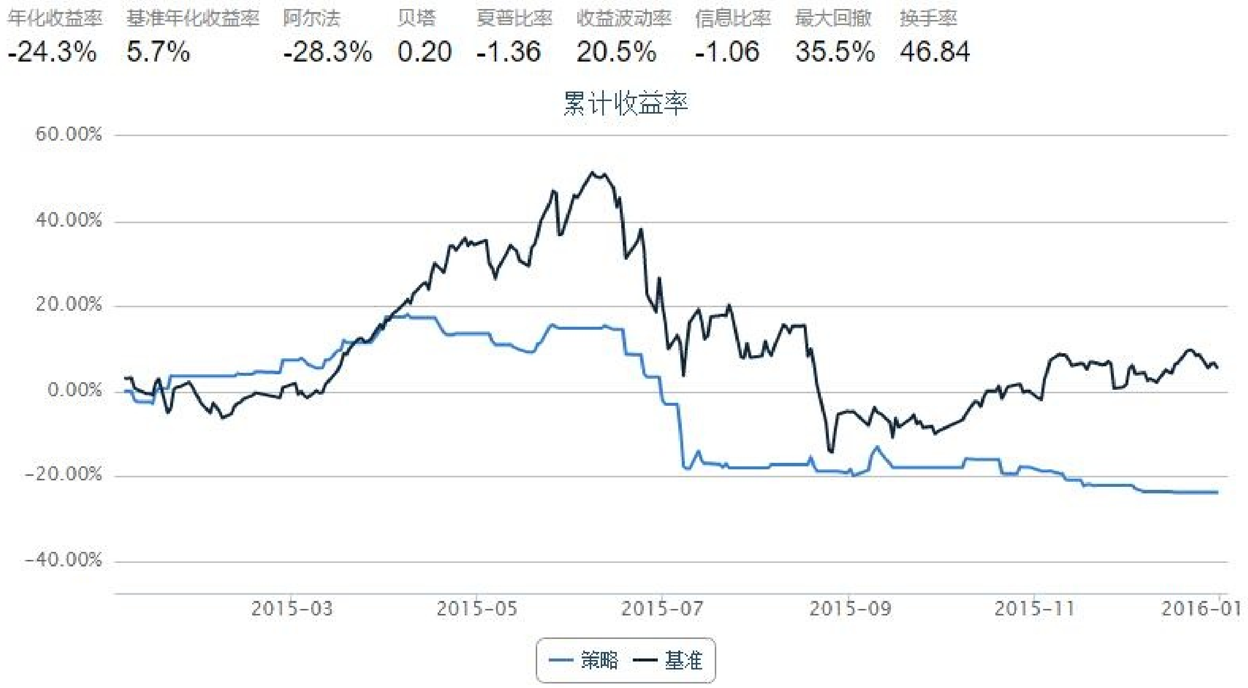
为了让大家科学的赚钱，小散决定验证一下这个策略。

第一步：读入老黄历data

```
import pandas as pd
data = pd.read_csv('老黄历诊股.csv',encoding='GB18030')
data = data[['calendarDate','isOpen']]
buydata = data[data['isOpen'] == 1]
selldata = data[data['isOpen'] == -1]
print buydata.head(5)
print selldata.head(5)

   calendarDate  isOpen
3    2015-01-08       1
7    2015-01-14       1
8    2015-01-15       1
11   2015-01-20       1
27   2015-02-11       1
   calendarDate  isOpen
1    2015-01-06      -1
10   2015-01-19      -1
13   2015-01-22      -1
15   2015-01-26      -1
18   2015-01-29      -1
```

第二步A，买卖HS300ETF。根据老黄历信号，宜开市开仓，否则平仓。

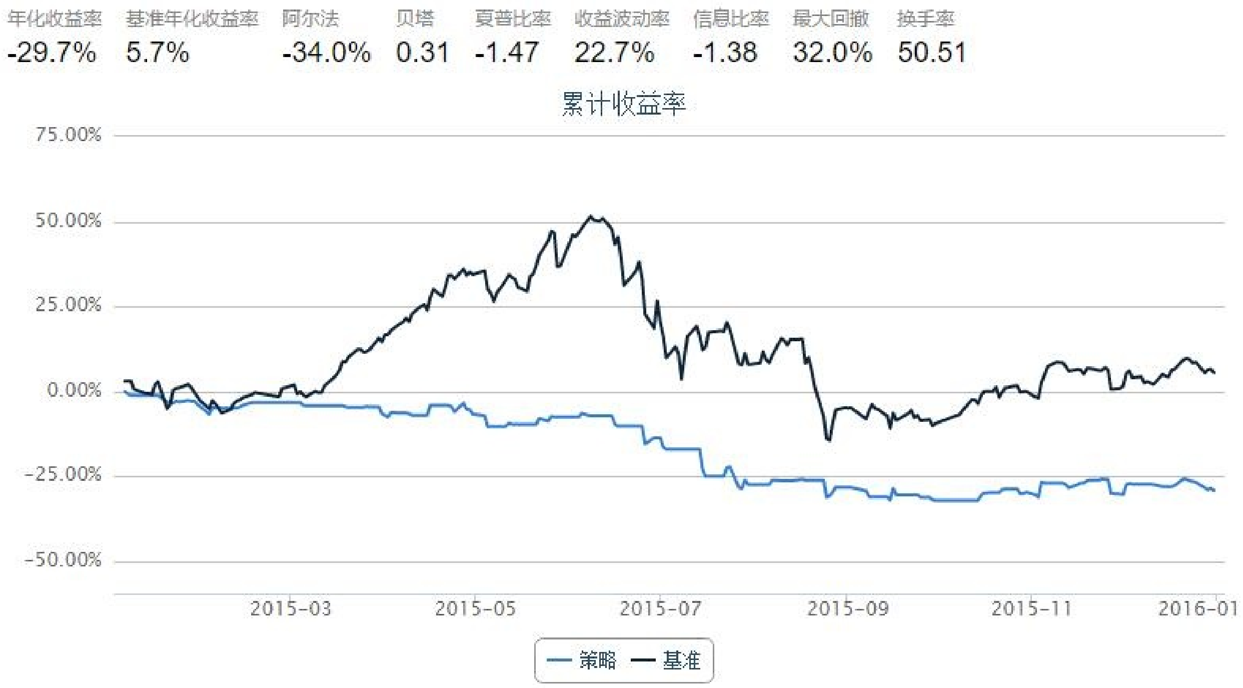
```
buy_date = map(lambda x: x[0:4]+x[5:7]+x[8:10], buydata['calendarDate'].values.tolist())
sell_date = map(lambda x: x[0:4]+x[5:7]+x[8:10], selldata['calendarDate'].values.tolist())
start = '2015-01-01'                       # 回测起始时间
end = '2015-12-31'      # 回测结束时间
benchmark = 'HS300'                            # 策略参考标准
universe = ['510310.XSHG']     # 股票池                            # 策略参考标准
refresh_rate = 1                          # 调仓频率，即每 refresh_rate 个交易日执行一次 handle_data() 函数
def initialize(account):                    # 初始化虚拟账户状态
    pass

def handle_data(account):                  # 每个交易日的买入卖出指令
    today=account.current_date.strftime('%Y%m%d')
    for stk in account.universe:
        if today in buy_date:
            order_pct_to(stk, 1)
        else:
            if stk in account.avail_secpos:
                order_to(stk, 0)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
-24.3%	5.7%	-28.3%	0.20	-1.36	20.5%	-1.06	35.5%	46.84

第二步B，买卖HS300ETF。根据老黄历信号，忌开市开仓，否则平仓。

```
start = '2015-01-01'                                    # 回测起始时间
end = '2015-12-31'      # 回测结束时间
benchmark = 'HS300'                                         # 策略参考标准
universe = ['510310.XSHG']    # 股票池                                 # 策略
参考标准
refresh_rate = 1                                       # 调仓频率，即每 refresh
_rate 个交易日执行一次 handle_data() 函数
def initialize(account):                                # 初始化虚拟账户状态
    pass

def handle_data(account):                               # 每个交易日的买入卖出指
令
    today=account.current_date.strftime('%Y%m%d')
    for stk in account.universe:
        if today in sell_date:
            order_pct_to(stk, 1)
        else:
            if stk in account.avail_secpos:
                order_to(stk, 0)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
-29.7%	5.7%	-34.0%	0.31	-1.47	22.7%	-1.38	32.0%	50.51

观察策略结果，虽然A，B都显著跑输大盘，但是A相对B仍然有5%的超额收益。真是应了那句老话：

本策略告诉小散：炒股有风险，新手慎入。

`老黄历诊股.csv` 数据来源：

1、 `DataAPI.TradeCalGet` 得到交易日历，下载到本地。

2、根据汉典老黄历开市日历为 `isOpen` 字段赋值，宜开市为1，忌开市为-1，无提示为0。

3、上传到Data使用。

```
print "Buydate:",buy_date
print "Selldate:",sell_date

Buydate: [u'20150108', u'20150114', u'20150115', u'20150120', u'
20150211', u'20150217', u'20150226', u'20150304', u'20150306', u
'20150311', u'20150312', u'20150316', u'20150318', u'20150330',
u'20150401', u'20150408', u'20150420', u'20150424', u'20150506',
u'20150513', u'20150515', u'20150520', u'20150521', u'20150525',
u'20150527', u'20150612', u'20150619', u'20150625', u'20150701',
u'20150707', u'20150708', u'20150713', u'20150714', u'20150720',
u'20150722', u'20150806', u'20150819', u'20150820', u'20150831',
u'20150901', u'20150902', u'20150908', u'20150910', u'20150914',
u'20151009', u'20151021', u'20151027', u'20151102', u'20151106',
u'20151110', u'20151111', u'20151117', u'20151120', u'20151204',
u'20151216']
Selldate: [u'20150106', u'20150119', u'20150122', u'20150126', u
'20150129', u'20150130', u'20150203', u'20150212', u'20150213',
u'20150216', u'20150305', u'20150319', u'20150326', u'20150331',
u'20150402', u'20150403', u'20150409', u'20150416', u'20150423',
u'20150427', u'20150428', u'20150430', u'20150505', u'20150512',
u'20150522', u'20150526', u'20150605', u'20150616', u'20150626',
u'20150702', u'20150715', u'20150723', u'20150727', u'20150729',
u'20150807', u'20150817', u'20150825', u'20150826', u'20150828',
u'20150907', u'20150915', u'20150916', u'20150917', u'20150923',
u'20150925', u'20150928', u'20150929', u'20151015', u'20151016',
u'20151022', u'20151028', u'20151029', u'20151030', u'20151103',
u'20151104', u'20151112', u'20151116', u'20151119', u'20151124',
u'20151127', u'20151201', u'20151202', u'20151203', u'20151211',
u'20151218', u'20151221', u'20151223', u'20151224', u'20151225',
u'20151228', u'20151230']
```

第三部分 基金、利率互换、固定收益类

一 分级基金

“优矿”集思录——分级基金专题

> 来源：https://uqer.io/community/share/561da7d9f9f06c4c5e2fb5cc

前言：

从14年7月份开始，伴随着牛市的到来，一个不怎么知名的投资标的大放光彩——分级基金

虽说分级基金是牛市产物，但对于短期的反弹行情其一样也是众多投资物中的一颗明珠，只要抓好反弹行情，随便吃几个涨停也是妥妥的

但分级基金相对复杂，要注意好分级基金的上下折、杠杆、折溢价等问题

本文假设读者已经对分级基金有一定的了解，只要运行代码就可以得到常见重要的分级基金相关指标，旨在成为投资者身边分级基金好工具

结合笔者平时的投资习惯，总结的指标主要有：母基金跟踪指数，整体溢价率，AB各自溢价率，B类价格杠杆，母基金下折需要跌多少，其他常见指标

用法：

运行下面的代码，即可以得到关于母基金、A份额、B份额，如上列举的所有指标

同时还可以选择输出结果排序规则等

有一定编程基础的读者，建议详细阅读函数说明，自行使用

```
def get_leverage_fund(show_type='M', order_by='discount_rate', order_method='desc', date=None):
    '''
    输入参数：
       show_type  str，展示/返回的数据，'T'为返回所有，'A'为返回A类相关，'B'为返回B类相关，'M'为返回母基金相关
       order_by  str，返回结果的排序属性列，可选的为'B_leverage'(B类价格杠杆),'ticker'(交易代码),'discount_rate'(整体溢价率)
       order_method  str，排序规则，降序（'desc'）,升序（'acd'）
    输出参数：
       计算好指标的dataframe，同时还将结果直接打印出来
    '''

    import pandas as pd
    import numpy as np
    from CAL.PyCAL import *

    if show_type not in ['T','A','B','M']:
        raise ValueError('show type 必须为T，A，B，M中的一个！')
    if order_by not in ['B_leverage','ticker','discount_rate']:
        raise ValueError('order_by 必须为B_leverage,ticker,discou
nt_rate中的一个！')

    if order_method not in ['desc','acd']:
        raise ValueError('order_method 必须为desc,acd中的一个！')

    # 日期默认为前一个工作日
    if date is None:
        date = Date.todaysDate()
        cal = Calendar('China.SSE')
        period = Period('-1B')
        date = cal.advanceDate(date, period)
        date = date.toDateTime().strftime('%Y%m%d')
    elif not (isinstance(date, str) and len(date) == 8):
        raise ValueError('date必须为xxxxxxxx字符串类型日期格式！')

    # 所有股票类分级基金ticker
    funds = DataAPI.FundLeverageInfoGet(exchangeCDLeverage=['XSH
G','XSHE'], field='ticker,secShortName,tickerLeverage,secShortNa
meLeverage,shareType,category,,shareProp,idxCn,splitNote,downThr
shold')
    funds_total = funds[funds['category']=='E']
    funds_total.drop('category', axis=1, inplace=True)
    funds_total.columns = ['母基金代码','母基金简称','子基金代码','子
基金简称','份额类别','分折比例A/B','跟踪指数','折算说明','下折B阈值']
    funds_total['子基金代码'] = funds_total['子基金代码'].apply(str
)
    funds_total['分折比例A/B'][funds_total['分折比例A/B'] == 1] = 5
.0

    # 替换基金简称
    codes = funds_total.drop_duplicates('母基金代码')['母基金代码']
.tolist()
    codes_leverage = map(str,funds_total['子基金代码'].tolist())
    short_names = DataAPI.FundGet(ticker=codes_leverage+codes, l
istStatusCd=['L','UN'], field='ticker,tradeAbbrName', pandas='1'
)
    tmp = pd.merge(funds_total, short_names, how='left', left_on=
'母基金代码', right_on='ticker')
    funds_total['母基金简称'] = tmp['tradeAbbrName']
    tmp = pd.merge(funds_total, short_names, how='left', left_on=
'子基金代码', right_on='ticker')
    funds_total['子基金简称'] = tmp['tradeAbbrName']

    # 取净值
    net_values = DataAPI.FundNavGet(ticker=codes+codes_leverage,
 dataDate=date, field='ticker,NAV', pandas='1')
    tmp = pd.merge(funds_total, net_values, how='left', left_on=
'母基金代码', right_on='ticker')
    funds_total['母基金净值'] = tmp['NAV']
    tmp = pd.merge(funds_total, net_values, how='left', left_on=
'子基金代码', right_on='ticker')
    funds_total['子基金净值'] = tmp['NAV']

    # 取行情
    prices = DataAPI.MktFunddGet(ticker=codes+codes_leverage, field='ticker,closePrice', tradeDate=date, pandas='1')
    tmp = pd.merge(funds_total, prices, how='left', left_on='子基金代码', right_on='ticker')
    funds_total['子基金价格'] = tmp['closePrice']
    funds_total['子基金溢价率'] = funds_total['子基金价格'] / funds_total['子基金净值'] - 1

    # 计算相关指标，合并dataframe
    funds_A = funds_total[funds_total['份额类别'] == 'A']
    funds_A.drop('份额类别',axis=1, inplace=True)
    funds_B = funds_total[funds_total['份额类别'] == 'B'][['母基金代码','子基金代码','子基金简称','子基金净值','子基金价格','子基金溢价率']]
    funds_B.columns = [['母基金代码','B类代码','B类简称','B类净值','B类价格','B类溢价率']]
    funds_leverage = pd.merge(funds_A, funds_B, how='left', on='母基金代码')
    funds_leverage.rename(columns={'子基金代码':'A类代码', '子基金简称':'A类简称', '子基金净值':'A类净值', '子基金价格':'A类价格', '子基金溢价率':'A类溢价率'}, inplace=True)
    funds_leverage['整体溢价率'] = (funds_leverage['A类价格'] * (funds_leverage['分折比例A/B'] / 10) + funds_leverage['B类价格'] * (1 - funds_leverage['分折比例A/B'] / 10)) / funds_leverage['母基金净值'] -1
    funds_leverage['B类价格杠杆'] = (funds_leverage['A类价格'] * funds_leverage['分折比例A/B'] / 10 + funds_leverage['B类价格'] * (1 - funds_leverage['分折比例A/B'] / 10)) / funds_leverage['B类价格'] / (1 - funds_leverage['分折比例A/B'] / 10)
    funds_leverage['下折母需跌'] = 1 - (funds_leverage['A类净值'] * funds_leverage['分折比例A/B'] / 10 + funds_leverage['下折B阈值'] * (1 - funds_leverage['分折比例A/B'] / 10)) / funds_leverage['母基金净值']
    funds_leverage = funds_leverage[['母基金代码','母基金简称','母基金净值','整体溢价率','跟踪指数','分折比例A/B','下折母需跌','A类代码','A类简称','A类净值','A类价格','A类溢价率','B类代码','B类简称','B类净值','B类价格','B类溢价率','B类价格杠杆','下折B阈值','折算说明']]
    funds_leverage['B类价格杠杆'] = np.round(funds_leverage['B类价格杠杆'], 2)
    funds_leverage.dropna(inplace=True)
    funds_leverage['整体溢价率'] = pd.Series(["{0:.1f}%".format(val * 100) for val in funds_leverage['整体溢价率']], index = funds_leverage.index)
    funds_leverage['A类溢价率'] = pd.Series(["{0:.1f}%".format(val * 100) for val in funds_leverage['A类溢价率']], index = funds_leverage.index)
    funds_leverage['B类溢价率'] = pd.Series(["{0:.1f}%".format(val * 100) for val in funds_leverage['B类溢价率']], index = funds_leverage.index)
    funds_leverage['下折母需跌'] = pd.Series(["{0:.1f}%".format(val * 100) for val in funds_leverage['下折母需跌']], index = funds_leverage.index)

    # 返回类型
    if show_type == 'T':
        columns = funds_leverage.columns
    elif show_type == 'A':
        columns = ['A类代码','A类简称','A类净值','A类价格','A类溢价率','整体溢价率','跟踪指数']
    elif show_type == 'B':
        columns = ['B类代码','B类简称','B类净值','B类价格','B类溢价率','B类价格杠杆','下折B阈值','整体溢价率','跟踪指数']
    else:
        columns = ['母基金代码','母基金简称','母基金净值','整体溢价率','跟踪指数','分折比例A/B','下折母需跌','折算说明']

    # 排序
    if order_by == 'B_leverage':
        order_by = 'B类价格杠杆'
    elif order_by == 'ticker':
        order_by = '母基金代码'
    else:
        order_by = '整体溢价率'

    if order_method == 'acd':
        res = funds_leverage.sort(columns=order_by, ascending=True)[columns]
    else:
        res = funds_leverage.sort(columns=order_by, ascending=False)[columns]
    res = res.reset_index().drop('index', axis=1)
    res
    return res
get_leverage_fund()
```

	母基金代码	母基金简称	母基金净值	整体溢价率	跟踪指数	分折比例A/B	下折母需跌	折算说
0	160420	华安50	0.6705	3.7%	创业板50	5	5.6%	每年12月日定折，节假日顺延，A类参与上折
1	160633	鹏华	0.6740	2.6%	证券	5	5.5%	每年11月一个工作

								定折
2	160634	鹏华环保分级	1.1280	2.5%	中证环保	5	44.5%	每年11月一个工作定折，B不参与定
3	161819	银华资源	1.0770	2.5%	内地资源	4	47.1%	每年第一工作日定
4	502020	国金50	1.1770	2.5%	上证50	5	46.6%	每年12月日定折，节假日顺
5	164821	工银新能源	0.8108	2.4%	中证新能	5	21.9%	每年1月一个工作定折(成不足6个月、下折足1个月外)
6	502023	钢铁分级	1.0230	2.4%	国证钢铁	5	38.5%	每年9月一个工作定折，距成立或上折不满3月除外，次折算后整约定利
7	502026	新丝路	1.0090	2.4%	新丝路	5	37.7%	每年9月一个工作定折，距成立或上折不满3月除外，次折算后整约定利
8	160628	鹏华地产	0.9890	2.3%	800地产	5	36.0%	每年第一工作日定
9	164820	工银高铁	0.7691	2.1%	高铁产业	5	17.9%	每年第一工作日定折，A类参与上折
								每年12月日定折，节假日顺

10	167301	方正保险	0.9270	2.0%	保险主题	5	32.0%	节假日顺延，基金同生效不六个月的外，上下不足1个除外
11	164809	工银500	1.4138	1.8%	中证500	4	59.7%	每年第一工作日定
12	161028	富国新能源	0.8540	1.7%	CS新能车	5	24.9%	12/15
13	165315	建信网金融	0.9880	1.3%	互联金融	5	36.1%	每年12月一个工作定折
14	162413	华宝1000	0.8673	1.2%	中证1000	5	27.0%	每年12月日定折，次定折或定折基准变更利率
15	165511	信诚500	1.1920	1.2%	中证500	4	53.7%	02/06，折日为非作日则提
16	502056	医疗分级	0.8919	0.9%	中证医疗	5	29.2%	每年12月日定折(节假日顺延，上下不足3个除外）
17	160637	鹏华创业板	0.7430	0.7%	创业板指	5	14.7%	每年12月一个工作定折
18	161628	融通军工	0.9780	0.7%	中证军工	5	35.2%	每年12月日定折，节假日提前，A类参与上折
19	161910	万家中创	1.1115	0.7%	创业成长	5	43.4%	每年第一工作日定
								每年12月日定折，假日顺延

								A类不参上折，B不参与定
21	162412	华宝医疗	0.8583	0.6%	中证医疗	5	26.3%	每年12月日定折，节假日提前，每次折或不定基准日变利率
22	168001	国寿养老	0.8770	0.5%	养老产业	5	27.8%	每年12月日定折，节假日提
23	160517	博时银行	0.8485	0.3%	中证银行	5	25.2%	每年12月一个工作定折，B不参与定
24	160629	鹏华传媒	1.1880	0.3%	中证传媒	5	47.1%	每年12月一个工作定折
25	160630	鹏华国防	1.2940	0.3%	中证国防	5	51.5%	每年12月一个工作定折
26	164819	工银环保	0.8604	0.3%	中证环保	5	26.4%	每年第一工作日定折，成立足半年可定折
27	160625	鹏华证保	1.1500	0.2%	800证保	5	45.3%	每年第一工作日定
28	161719	招商可转债	1.0690	0.2%	中信转债	7	21.0%	每年12月日定折，3年折为基金再折
29	162107	金鹰中证500	1.0987	0.2%	中证500	5	42.9%	每年第一工作日定
...	...	...	...	...	...	...	...	...
73	164401	前海健康	1.0530	-0.5%	健康产业	5	39.0%	每年第一工作日定

74	165521	信诚金融	0.9160	-0.5%	800金融	5	29.1%	12/15
75	167503	安信一带一路	0.9160	-0.5%	一带一路	5	31.1%	每年12月日定折节日提前
76	502016	长信一带一路	1.0350	-0.5%	一带一路	5	39.2%	每年12月日定折，节假日提前，距离立或上下不满3个除外
77	168203	中融钢铁	0.9230	-0.4%	国证钢铁	5	31.9%	每年12月日定折，节假日提
78	160127	南方消费	0.9730	-0.3%	内地消费	5	36.8%	03/13，折时A/B/全归1，顺延定折
79	161726	招商生物	1.0360	-0.3%	生物医药	5	39.0%	每年12月日定折，节假日顺
80	162010	长城久兆	1.1360	-0.3%	中小300P	4	51.4%	01/30，提前定折
81	162509	国联安双禧	1.3030	-0.3%	中证100	4	57.4%	2016/04/每三年定时A/B/M归1，定日为非工日则提前
82	164905	交银新能源	1.1900	-0.3%	国证新能	5	47.3%	每年第一工作日定
83	165515	信诚300	0.7410	-0.3%	沪深300	5	13.8%	12/15
84	167601	国金300	0.9048	-0.3%	沪深300	5	28.3%	每年第一工作日定
85	160417	华安300	1.2690	-0.2%	沪深300	5	46.8%	每年第一工作日定

86	161022	创业板	1.1030	-0.2%	板指	5	43.0%	工作日定
87	161030	富国体育	0.8940	-0.2%	中证体育	5	29.2%	每年12月一个工作定折
88	161721	招商300地产	0.7360	-0.2%	地产等权	5	13.6%	12/15
89	161724	招商煤炭	0.9560	-0.2%	煤炭等权	5	33.9%	每年12月日定折，节假日顺
90	163209	诺安中创	1.2210	-0.2%	创业成长	4	54.8%	每年第一工作日定
91	164811	工银100	1.2346	-0.2%	深证100P	5	48.7%	07/01
92	160219	国泰医药	0.8360	-0.1%	国证医药	5	23.3%	每年第一工作日定折，每次折或不定基准日变利率
93	160516	博时证保	1.0758	-0.1%	800证保	5	41.6%	每年12月一个工作定折
94	161031	富国工业4	1.0710	-0.1%	工业4.0	5	41.3%	每年12月一个工作定折，B不参与定
95	161718	招商300高贝	0.8140	-0.1%	300高贝	5	21.1%	12/15
96	161812	银华100	0.9530	-0.1%	深证100R	5	32.1%	每年第一工作日定
97	163109	申万深成	0.6555	-0.1%	深证成指	5	12.7%	每年第一工作日定折，无下折，A不与上折，值<1元无折

98	164508	国富100	0.8310	-0.1%	中证100	5	22.8%	每年第一工作日定
99	168204	中融煤炭	0.7310	-0.1%	中证煤炭	5	13.2%	每年12月日定折，节假日提
100	161029	富国银行	0.8940	-0.0%	中证银行	5	28.7%	每年12月日
101	161629	融通证券	0.7770	-0.0%	证券公司	5	18.7%	每年12月日定折，类不参与折
102	167701	德邦德信	1.0520	-0.0%	中高企债	7	20.9%	每次折算2周年定转为母基折分

```
103 rows × 8 columns
```

基于期权定价的分级基金交易策略

> 来源：https://uqer.io/community/share/548a6af2f9f06c31c3950ca7

版本：1.0

作者：李丞

联系：cheng.li@datayes.com

1. 分级基金中的期权结构

分级基金是中国金融市场化下创新的产物，多数是以AB端分级，A端获取相对保守收益，B端获取杠杆收益的结构。通俗的讲，在分级基金结构中，大多数情况下，B端优先承受市场风险损失，换取A端“借”给它钱投资的融资优势。

现在市场上大多数的指数型分级基金采取的收益分配模式为：A端获取固定的约定收益率，多半为一年期定存+x%；B端获取剩余的母基金净资产。这样的分级基金可以看做A端是一个固定利率债券，B端是一个看涨期权，其中的期权卖方恰恰是A端。在这里我们不会详细探讨这一类型的结构，关于这一类型分级基金的期权分析可以参考[1]。

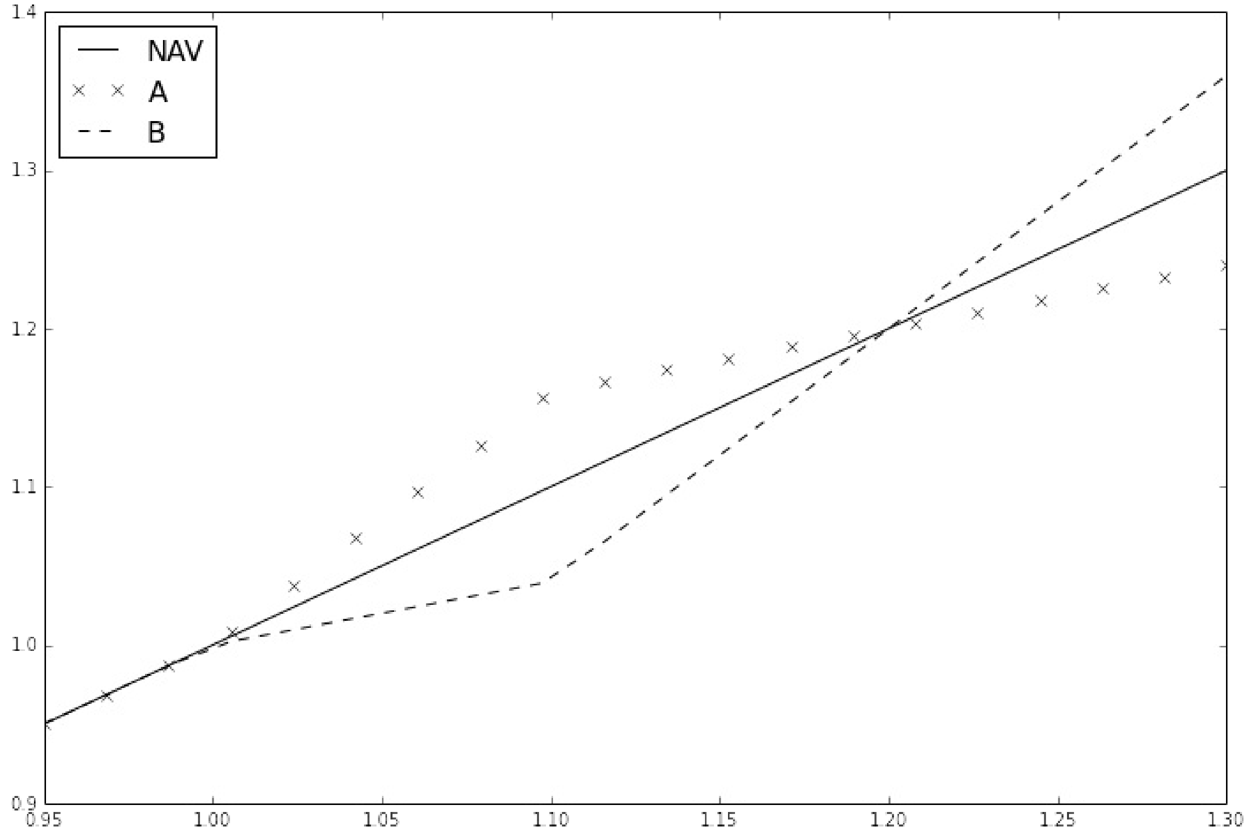
这里我们会看一个有趣的产品，在这个产品中，A、B端都是期权形式的[1]。这个产品就是国投瑞银瑞和沪深300分级证券投资基金。在它的招募说明书中，有这样的表述：

> (年阀值为10%)在任一运作周年内,如果瑞和 300 份额的基金份额净值大于 1.000 元,则在每份瑞和小康份额与每份瑞和远见份额各自获得 1.000 元净值的基础上,本基金将以年阀值为基准,将瑞和 300 份额的基金份额净值超出 1.000 元的部分划分成年阀值以内和年阀值以外的两个部分,与此相对应,对于每一对瑞和小康份额与瑞和远见份额的份额组合所包含的年阀值以内的部分,由一份瑞和小康份额与一份瑞和远见份额按 8:2 的比例分成;对于每一对瑞和小康份额与瑞和远见份额的份额组合所包含的年阀值以外的部分,由一份瑞和小康份额与一份瑞和远见份额按 2:8 的比例分成。 在我们下面的分析中你可以看到，这个是典型的期权结构，并且可以拆分成简单的看涨看跌期权的和。

2. 瑞和300期权结构分析

收益的结构最容易以一张图的形式表示出来：

```
from matplotlib import pyplot
def AReturn(base):
    if base < 1.0:
        return base
    elif base >=1 and base < 1.1:
        return 1.6 * base - 0.6
    else:
        return 0.4 * base + 0.72

def BReturn(base):
    if base < 1.0:
        return base
    elif base >=1 and base < 1.1:
        return 0.4 * base + 0.6
    else:
        return 1.6* base - 0.72

xspace = np.linspace(0.95, 1.3, 20)

aSeries = [AReturn(x) for x in xspace]
bSeries = [BReturn(x) for x in xspace]

pyplot.figure(figsize=(12,8))
pyplot.plot(xspace, xspace, '-k')
pyplot.plot(xspace, aSeries, 'xk')
pyplot.plot(xspace, bSeries, '--k')
pyplot.xlim((0.95,1.3))
pyplot.legend(['NAV', 'A', 'B'], loc = 'best', fontsize = 16)

<matplotlib.legend.Legend at 0x4ad7390>
```



收益的描述也可以用下式描述，其中 NAV 为母基金净值：

$$A=\begin{cases}\mathrm{NAV},\ \text{如果 } \mathrm{NAV}\le 1,\\ \\ 1.6\times\mathrm{NAV}-0.60,\ \text{如果 } 1\le\mathrm{NAV}\le 1.1,\\ \\ 0.4\times\mathrm{NAV}+0.72,\ \text{如果 } \mathrm{NAV}\ge 1.1\end{cases}$$

B的收益也有类似的式子：

$$B=\begin{cases}\mathrm{NAV},\ \text{如果 } \mathrm{NAV}\le 1,\\ \\ 0.4\times\mathrm{NAV}+0.60,\ \text{如果 } 1\le\mathrm{NAV}\le 1.1,\\ \\ 1.6\times\mathrm{NAV}-0.72,\ \text{如果 } \mathrm{NAV}\ge 1.1\end{cases}$$

实际上我们可以把它写成更明显的形式，展示它们的内在期权实质：

$$A=1-\mathrm{MAX}(0.0,1.0-\mathrm{NAV})+1.6\times\mathrm{MAX}(0.0,\mathrm{NAV}-1.0)-1.2\times\mathrm{MAX}(0.0,\mathrm{NAV}-1.1),$$

$$B=1-\mathrm{MAX}(0.0,1.0-\mathrm{NAV})+0.4\times\mathrm{MAX}(0.0,\mathrm{NAV}-1.0)+1.2\times\mathrm{MAX}(0.0,\mathrm{NAV}-1.1)$$

可以看到，这两个子基金的价值都是三个期权的组合，只是权重不同：

- 行权价为1.0的看跌期权

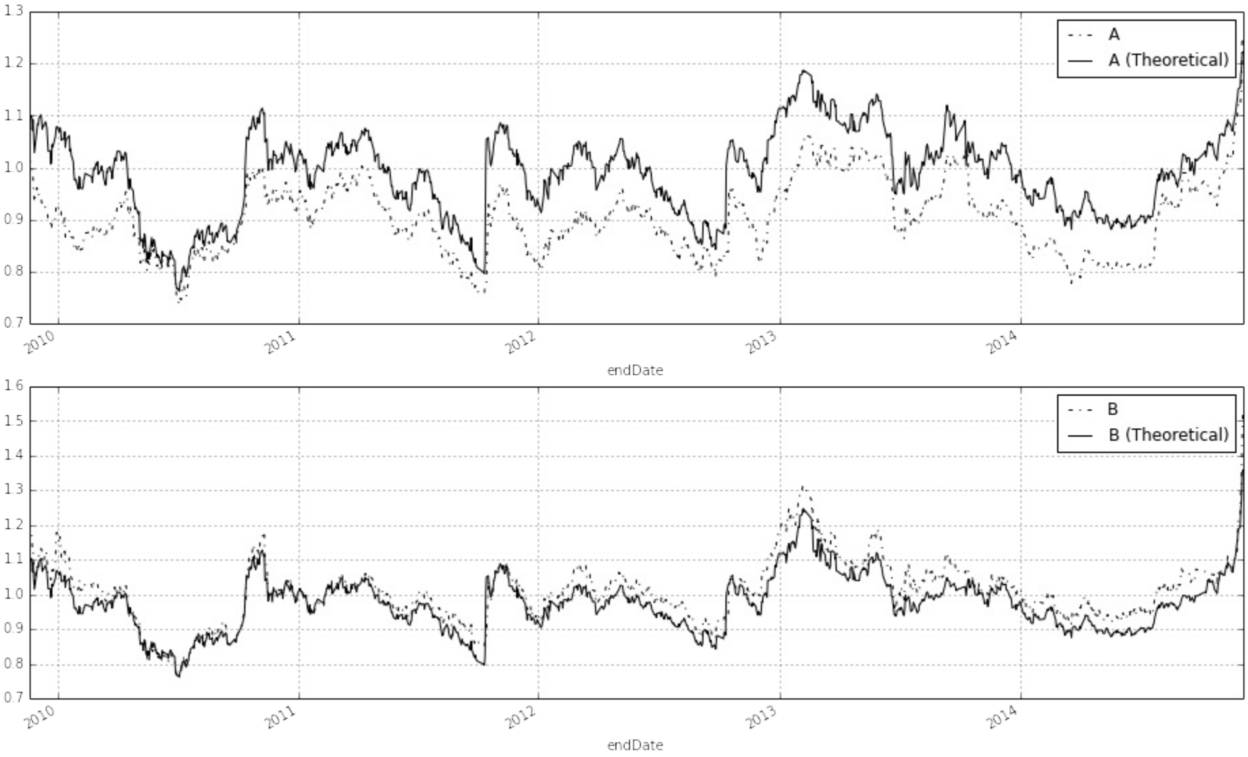
- 行权价为1.0的看涨期权
- 行权价为1.1的看涨期权

对于这些期权，我们可以假设标的即为母基金净值，期限为当前日期到下一个折算日（即为每年的10月12日，遇到节假日的话，向前调整至上一营业日），无风险利率使用3个月Shibor做简单的近似：

```
# 导入需要的模块
from CAL.PyCAL import *

# 读入外部行情数据
data = pd.read_excel(r'gtry_dat_300.xlsx','Sheet1')

riskFree = data['Shibor 3M'] / 100.0
maturity = data['Maturity']
spot = data['161207.OFCN']
ATarget = data['150008.XSHE']
BTarget = data['150009.XSHE']


def AOptionPrice(vol, riskFree, maturity, spot):
    price1 = BSMPrice(-1, 1.0, spot, riskFree, 0.0, vol[0], maturity, rawOutput = True)
    price2 = BSMPrice(1, 1.0, spot, riskFree, 0.0, vol[0], maturity, rawOutput = True)
    price3 = BSMPrice(1, 1.1, spot, riskFree, 0.0, vol[0], maturity, rawOutput = True)
    return  1.0 - price1[0] + price2[0] * 1.6 - price3[0] * 1.2

def BOptionPrice(vol, riskFree, maturity, spot):
    price1 = BSMPrice(-1, 1.0, spot, riskFree, 0.0, vol[0], maturity, rawOutput = True)
    price2 = BSMPrice(1, 1.0, spot, riskFree, 0.0, vol[0], maturity, rawOutput = True)
    price3 = BSMPrice(1, 1.1, spot, riskFree, 0.0, vol[0], maturity, rawOutput = True)
    return  1.0 - price1[0] + price2[0] * 0.4 + price3[0] * 1.2


aTheoreticalPrice = AOptionPrice([0.15], riskFree, maturity, spot)
bTheoreticalPrice = BOptionPrice([0.15], riskFree, maturity, spot)
```

我们分别看一下，AB端基金理论价格和实际收盘价之间的关系（上面的计算中假设波动率为15%）：可以看到，基本上理论价格和真实价格的变动是完全通向的，但是存在价差，A长期折价，B长期溢价。这个价差随着到期折算日的接近，收敛至0。这个是与期权的性质是完全一致的。

```
data['A (Theoretical)'] = aTheoreticalPrice
data['B (Theoretical)'] = bTheoreticalPrice
pyplot.figure(figsize = (16,10))
ax1 = pyplot.subplot('211')
data.plot('endDate', ['150008.XSHE','A (Theoretical)'], style = [
'-.k', '-k'])
ax1.legend(['A', 'A (Theoretical)'], loc = 'best')
ax2 = pyplot.subplot('212')
data.plot('endDate', ['150009.XSHE','B (Theoretical)'], style = [
'-.k', '-k'])
ax2.legend(['B', 'B (Theoretical)'], loc = 'best')

<matplotlib.legend.Legend at 0x5c64210>
```



3. 瑞和300的期权投资策略

根据上面的分析，似乎读者可以得到这样的一个印象：A端长期比较便宜，应该直接持有A端，真的是这样吗？这里面实际上有以下的问题：

- A端由于收益算法的原因，属于类固定收益产品，并且它在标的价格高企时，凸性为负；
- B端属于杠杆类型，在标的股价高企时，凸性为正；
- 市场可能会对凸性的不同，对于AB端分别进行折溢价调整；
- 15%是一个魔幻数（Magic Number)，真实市场波动率水平显然不应该是一个常值。

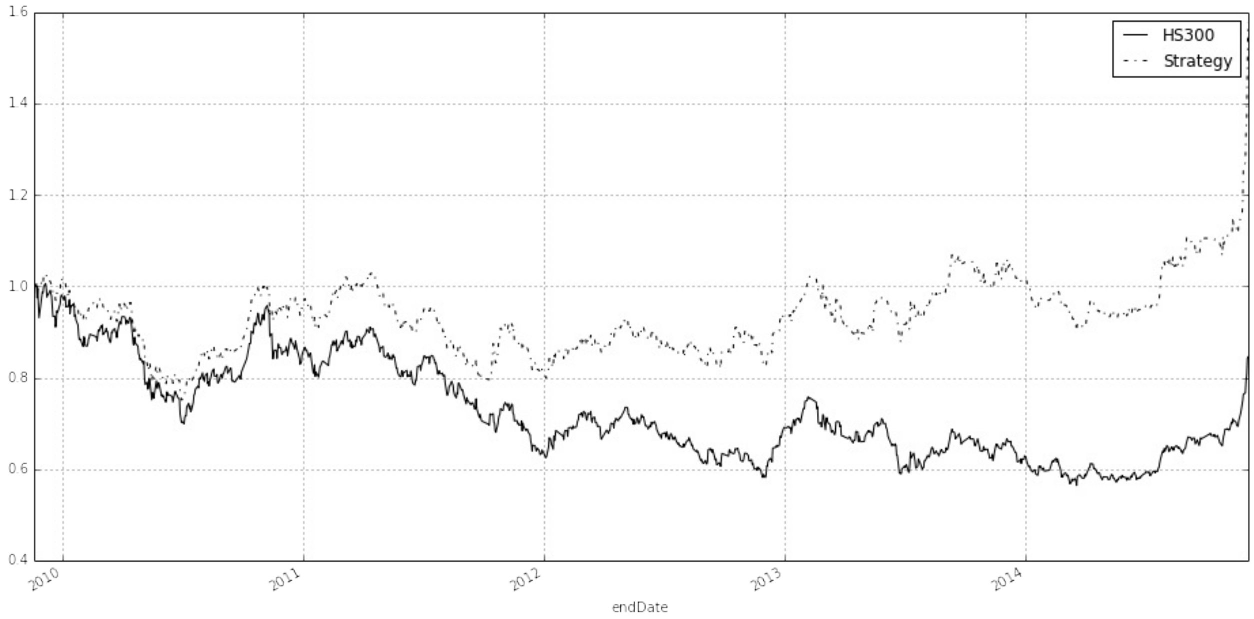
这里我们将涉及一个策略，试着解释最后一个问题。期权有一种估计期权的方法，称为“隐含波动率”。我们可以把类似的想法引入我们这个产品当中，每天收盘的时候，我们可以观察到A端和B端的价格（或者说两个期权组合的价格）。这时候，可以使用优化的方法，找到一个波动率水平使得理论价格在某种标准下与实际价格差异最小。我们把这个波动率水平称之为瑞和300的“隐含波动率”。

有了这个隐含波动率水平，我们可以再计算理论价格，这时候计算而得的理论价格，我们可以认为是“真实”市场估计下的理论期权价值。用这个价格作为评估的标准，比较A端与B端那个更加便宜，从而决定购买哪个产品。下面的策略中，我们即使用上面介绍的办法，每天调仓，根据估价的高低，分别购买A端（B端），同时卖出B端（A端）。这个策略只在二级市场中进行交易：

```
def processDate(record):

    riskFree = record['Shibor 3M'] / 100.0
    maturity = record['Maturity']
    spot = record['161207.OFCN']
    ATarget = record['150008.XSHE']
    BTarget = record['150009.XSHE']

    def errorFunction(vol):

        price1 = AOptionPrice(vol, riskFree, maturity, spot)
        price2 = BOptionPrice(vol, riskFree, maturity, spot)

        return (price1 - ATarget)**2 + (price2 - BTarget)**2

    out, fx, its, imode, smode = optimize.fmin_slsqp(errorFuncti
on, [0.15], bounds = [(0.01, 0.25)], epsilon = 1e-6, iter = 10000
, disp = False, full_output = True, acc = 1e-16)

    price1 = AOptionPrice(out, riskFree, maturity, spot)
    price2 = BOptionPrice(out, riskFree, maturity, spot)

    return price1 - ATarget, price2 - BTarget
```

```
import datetime as dt
from scipy import optimize

callDate = [dt.datetime(2010,10,12), dt.datetime(2011,10,12), dt
.datetime(2012,10,11), dt.datetime(2013,10,10), dt.datetime(2014,
10,10)]

class deque:

    def __init__(self, maxlen):
        self.maxlen = maxlen
        self.cont = []
    def append(self,vec):

        self.cont.append(vec)
        if len(self.cont)>100:
            self.cont = self.cont[len(self.cont) - 100:]

    def __item__(self, i):
        return self.cont[i]

    def average(self):
        sum = 0.0
        for i in xrange(len(self.cont)):
            sum += self.cont[i]
        return sum / float(len(self.cont))

class Account:

    def __init__(self, cash):

        self.aAmount = 0
        self.bAmount = 0
        self.cash = cash

    def order(self, amount, fundType, price):
        if fundType.upper() == 'A':
            self.aAmount += amount
            self.cash -= amount * price
        elif fundType.upper() == 'B':
            self.bAmount += amount
            self.cash -= amount * price

    def currentValue(self, aQuote, bQuote):
        return self.aAmount * aQuote + self.bAmount * bQuote + s
elf.cash

def BackTesting(data, window = 5, startAmount = 100000, tradeVol
 = 2000):

    account = Account(startAmount)

    aWindow = deque(maxlen = window)
    bWindow = deque(maxlen = window)
    performance = [startAmount]
    aVol = [0]
    bVol = [0]
    cash = [startAmount]
    for i in xrange(1, len(data)):
        previousDay = data.loc[i-1]
        aUnderEstimated, bUnderEstimated  = processDate(previous
Day)
        aWindow.append(aUnderEstimated)
        bWindow.append(bUnderEstimated)
        aAverage = aWindow.average()
        bAverage = bWindow.average()

        today = data.loc[i]
        aPrice = today['150008.XSHE']
        bPrice = today['150009.XSHE']
        if i >= window:
            # 如果分级A端相对于B端更便宜
            if aUnderEstimated - aAverage > bUnderEstimated - bA
verage:

                if account.cash > tradeVol:
                    account.order(tradeVol, 'A', aPrice)
                if account.bAmount >0:
                    account.order(-tradeVol, 'B', bPrice)

            # 如果分级B端相对于A端更便宜
            elif aUnderEstimated - aAverage < bUnderEstimated -
bAverage:
                if account.cash > tradeVol:
                    account.order(tradeVol, 'B', bPrice)
                if account.aAmount >0:
                    account.order(-tradeVol, 'A', aPrice)

            for calDate in callDate:
                if today['endDate'] == calDate:
                    account.order(-account.aAmount, 'A', aPrice)
                    account.order(-account.bAmount, 'B', bPrice)


        performance.append(account.currentValue(aPrice, bPrice))
        aVol.append(account.aAmount)
        bVol.append(account.bAmount)
        cash.append(account.cash)


    originalReturn = list(data['161207.OFCN'].values)
    start = originalReturn[0]
    originalReturn[0] = 1.0
    dates = data['endDate']
    scalar = 1.0
    count = 0
    for i in xrange(1, len(originalReturn)):
        if count < len(callDate) and dates[i-1] == callDate[coun
t]:
           start = originalReturn[i]
           originalReturn[i] =  originalReturn[i-1]
           count += 1
        else:
           scalar = originalReturn[i] / start
           start = originalReturn[i]
           originalReturn[i] = originalReturn[i-1] * scalar
    scalar = float(performance[0])
    performance = [p / scalar for p in performance]
    return pd.DataFrame({'Performance':performance, '150008.XSHE'
: aVol, '150009.XSHE': bVol, 'Cash': cash, '161207.OFCN': data['
161207.OFCN'].values, 'Benchmark Return':originalReturn } ,index
 = data.endDate)
```

```
bt = BackTesting(data, tradeVol = 20000)
bt.plot(y = ['Benchmark Return', 'Performance'], figsize = (16,8
), style = ['-k', '-.k'])
pyplot.legend( ['HS300', 'Strategy'], loc = 'best')

<matplotlib.legend.Legend at 0x4a70a10>
```



由上图可知，这样的策略是比较典型的指数增强型策略。本质上瑞和300母基金是沪深300指数的复制，该策略是捕捉A端、B端中的阿尔法因素，增强指数的表现。

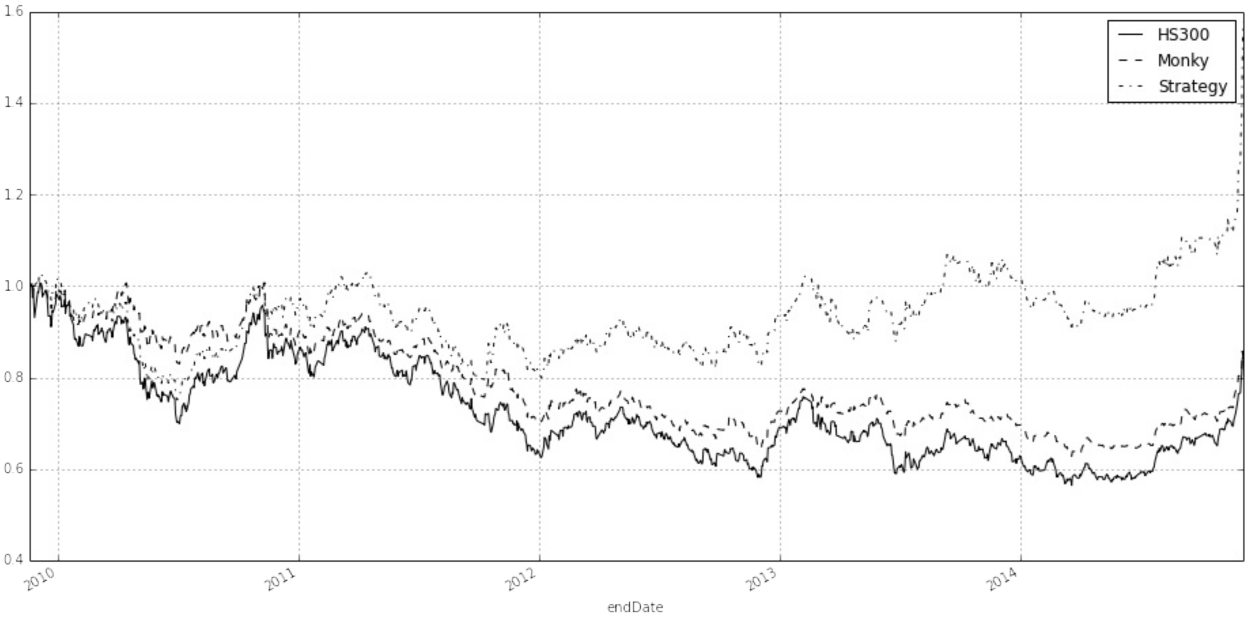
4. 我们是否能够比“猴子”做的更好？

作为和该策略的比较，我们可以使用一个随机投资的做法。让我们看看，和“猴子”(Monky Random Choice Strategy）比，我们是否能够做的更好？

```
def BackTesting2(data, window = 5, startAmount = 100000, tradeVo
l = 2000):

    account = Account(startAmount)

    performance = [startAmount]
    aVol = [0]
    bVol = [0]
    cash = [startAmount]
    s = MersenneTwister19937UniformRsg()
    for i in xrange(1, len(data)):
        previousDay = data.loc[i-1]
        aUnderEstimated, bUnderEstimated  = processDate(previous
Day)

        today = data.loc[i]
        aPrice = today['150008.XSHE']
        bPrice = today['150009.XSHE']
        if i >= window:
            # 如果随机数>0.5
            if s.nextSequence()[0] > 0.5:

                if account.cash > tradeVol:
                    account.order(tradeVol, 'A', aPrice)
                if account.bAmount >0:
                    account.order(-tradeVol, 'B', bPrice)

            # 如果随机数<0.5
            elif s.nextSequence()[0] < 0.5:
                if account.cash > tradeVol:
                    account.order(tradeVol, 'B', bPrice)
                if account.aAmount >0:
                    account.order(-tradeVol, 'A', aPrice)

            for calDate in callDate:
                if today['endDate'] == calDate:
                    account.order(-account.aAmount, 'A', aPrice)
                    account.order(-account.bAmount, 'B', bPrice)


        performance.append(account.currentValue(aPrice, bPrice))
        aVol.append(account.aAmount)
        bVol.append(account.bAmount)
        cash.append(account.cash)


    originalReturn = list(data['161207.OFCN'].values)
    start = originalReturn[0]
    originalReturn[0] = 1.0
    dates = data['endDate']
    scalar = 1.0
    count = 0
    for i in xrange(1, len(originalReturn)):
        if count < len(callDate) and dates[i-1] == callDate[coun
t]:
           start = originalReturn[i]
           originalReturn[i] =  originalReturn[i-1]
           count += 1
        else:
           scalar = originalReturn[i] / start
           start = originalReturn[i]
```

```
                originalReturn[i] = originalReturn[i-1] * scalar
        scalar = float(performance[0])
        performance = [p / scalar for p in performance]
        return pd.DataFrame({'Performance':performance, '150008.XSHE'
: aVol, '150009.XSHE': bVol, 'Cash': cash, '161207.OFCN': data['
161207.OFCN'].values, 'Benchmark Return':originalReturn } ,index
 = data.endDate)
```

```
bt1 = BackTesting(data, tradeVol = 20000)
bt2 = BackTesting2(data, tradeVol = 20000)
bt1['Monky'] = bt2['Performance']
bt1.plot(y = ['Benchmark Return', 'Monky', 'Performance'], figsi
ze = (16,8), style = ['-k', '--k', '-.k'])
pyplot.legend( ['HS300', 'Monky', 'Strategy'], loc = 'best')

<matplotlib.legend.Legend at 0x5c979d0>
```



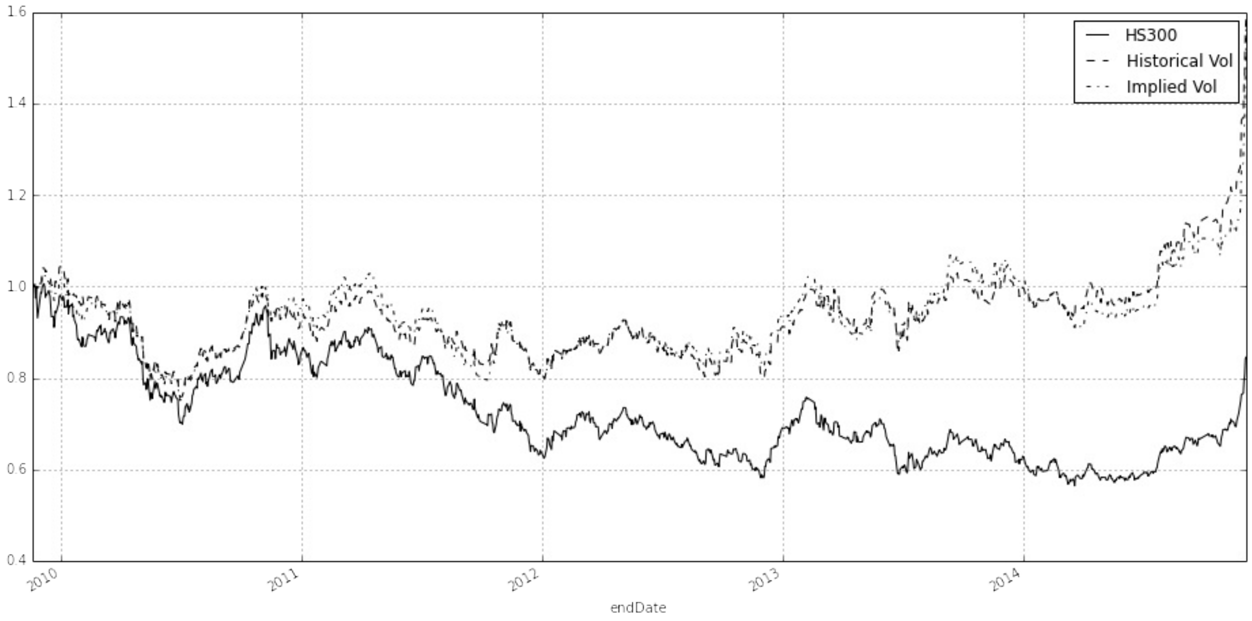
结果令人满意，我们的期权投资比随机选择的结果好的多。我们看到如果随机投资，“猴子”式的选择并不能显著的击败标的母基金。但是我们的期权投资策略还是可以保持的持续性的跑赢指数以及随机选择。

5. 历史波动率作为输入参数

这里我们给了一个使用历史波动率计算折溢价水平，与之前使用的隐含波动率方法进行比较。这里使用的历史波动率水平是20天年化收益标准差。结果上，我们无法显著区别这两种波动率算法在表现上面的区别。但是他们都可以显著的击败标的母基金。

```
def processDate2(record):
    riskFree = record['Shibor 3M'] / 100.0
    maturity = record['Maturity']
    spot = record['161207.OFCN']
    ATarget = record['150008.XSHE']
    BTarget = record['150009.XSHE']
    volatility = record['volatility']

    vol = [volatility]

    price1 = AOptionPrice(vol, riskFree, maturity, spot)
    price2 = BOptionPrice(vol, riskFree, maturity, spot)

    return price1 - ATarget, price2 - BTarget

def BackTesting3(data, window = 5, startAmount = 100000, tradeVo
l = 2000):

    account = Account(startAmount)

    aWindow = deque(maxlen = window)
    bWindow = deque(maxlen = window)
    performance = [startAmount]
    aVol = [0]
    bVol = [0]
    cash = [startAmount]
    for i in xrange(1, len(data)):
        previousDay = data.loc[i-1]
        aUnderEstimated, bUnderEstimated  = processDate2(previou
sDay)
        aWindow.append(aUnderEstimated)
        bWindow.append(bUnderEstimated)

        aAverage = aWindow.average()
        bAverage = bWindow.average()

        today = data.loc[i]
        aPrice = today['150008.XSHE']
        bPrice = today['150009.XSHE']
        if i >= window:
            # 如果分级A端相对于B端更便宜
            if aUnderEstimated - aAverage > bUnderEstimated - bA
verage:

                if account.cash > tradeVol:
                    account.order(tradeVol, 'A', aPrice)
                if account.bAmount >0:
                    account.order(-tradeVol, 'B', bPrice)

            # 如果分级B端相对于A端更便宜
            elif aUnderEstimated - aAverage < bUnderEstimated -
bAverage:
                if account.cash > tradeVol:
                    account.order(tradeVol, 'B', bPrice)
                if account.aAmount >0:
                    account.order(-tradeVol, 'A', aPrice)

            for calDate in callDate:
                if today['endDate'] == calDate:
                    account.order(-account.aAmount, 'A', aPrice)
                    account.order(-account.bAmount, 'B', bPrice)


        performance.append(account.currentValue(aPrice, bPrice))
        aVol.append(account.aAmount)
        bVol.append(account.bAmount)
        cash.append(account.cash)


    originalReturn = list(data['161207.OFCN'].values)
    start = originalReturn[0]
    originalReturn[0] = 1.0
    dates = data['endDate']
    scalar = 1.0
    count = 0
    for i in xrange(1, len(originalReturn)):
        if count < len(callDate) and dates[i-1] == callDate[coun
t]:
           start = originalReturn[i]
           originalReturn[i] =  originalReturn[i-1]
           count += 1
        else:
           scalar = originalReturn[i] / start
           start = originalReturn[i]
           originalReturn[i] = originalReturn[i-1] * scalar
    scalar = float(performance[0])
    performance = [p / scalar for p in performance]
    return pd.DataFrame({'Performance':performance, '150008.XSHE'
: aVol, '150009.XSHE': bVol, 'Cash': cash, '161207.OFCN': data['
161207.OFCN'].values, 'Benchmark Return':originalReturn } ,index
 = data.endDate)
```

```
bt3 = BackTesting3(data, tradeVol = 20000)
bt1['Historical (Vol)'] = bt3['Performance']
bt1.plot(y = ['Benchmark Return', 'Historical (Vol)', 'Performan
ce'], figsize = (16,8), style = ['-k', '--k', '-.k'])
pyplot.legend( ['HS300', 'Historical Vol', 'Implied Vol'], loc =
'best')

<matplotlib.legend.Legend at 0x6073850>
```



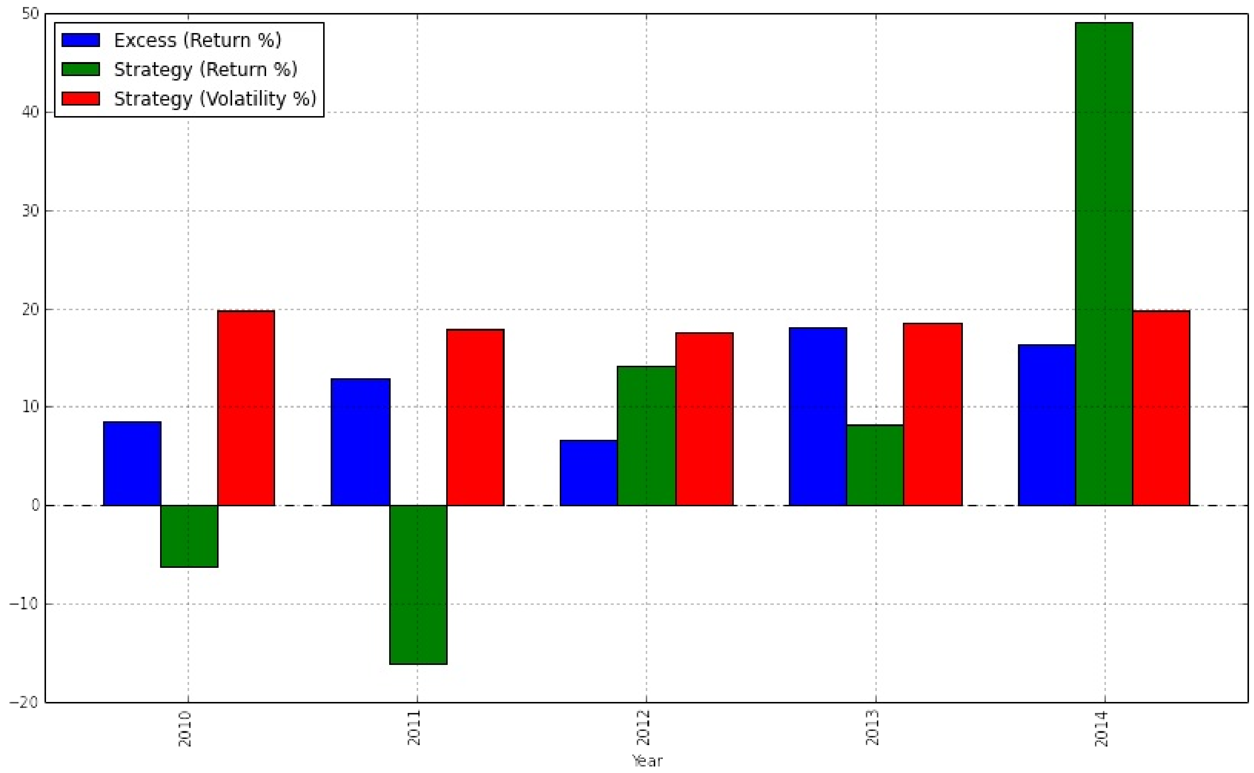
6. 风险收益分析

下面我们按照每个自然年评估策略的绩效（注意，这里2009年的时间比较短，所以并没有对它进行评估）。可以看到在5个自然年中，有两年策略的收益率为负的；但是与之相对的，相对于母基金的基准收益，超额收益始终为正的。最高的超额收益发生在2013年为18.06%，最低为为2012年6.54%。

```
value = bt1[['Performance', 'Benchmark Return']]
value['endDate'] = value.index.values
returnRes = [0]
tmp = np.log(value['Performance'][1:].values/ value['Performance'
][:-1].values)
returnRes.extend(tmp)
value['Per. Return'] = returnRes
returnRes = [0]
tmp = np.log(value['Benchmark Return'][1:].values/ value['Benchm
ark Return'][:-1].values)
returnRes.extend(tmp)
value['Benchmark Return'] = returnRes
year2010 = value[(value['endDate'] > Date(2010,1,1).toTimestamp(
)) & (value['endDate'] <= Date(2010,12,31).toTimestamp())]
year2011 = value[(value['endDate'] > Date(2011,1,1).toTimestamp(
)) & (value['endDate'] <= Date(2011,12,31).toTimestamp())]
year2012 = value[(value['endDate'] > Date(2012,1,1).toTimestamp(
)) & (value['endDate'] <= Date(2012,12,31).toTimestamp())]
year2013 = value[(value['endDate'] > Date(2013,1,1).toTimestamp(
)) & (value['endDate'] <= Date(2013,12,31).toTimestamp())]
year2014 = value[(value['endDate'] > Date(2014,1,1).toTimestamp(
)) & (value['endDate'] <= Date(2014,12,31).toTimestamp())]

days = 252
```

```
def perRes(yearRes):
    yearRes['Excess Return'] = yearRes['Per. Return'] - yearRes['Benchmark Return']
    mean = yearRes.mean() * days * 100
    std = yearRes.std() * np.sqrt(days) * 100

    return mean['Per. Return'], mean['Excess Return'], std['Per. Return']

res2010 = perRes(year2010)
res2011 = perRes(year2011)
res2012 = perRes(year2012)
res2013 = perRes(year2013)
res2014 = perRes(year2014)

perRet = []
exceRet= []
perStd = []

for res in [res2010, res2011, res2012, res2013, res2014]:
    perRet.append(res[0])
    exceRet.append(res[1])
    perStd.append(res[2])

resTable = pd.DataFrame({'Strategy (Return %)':perRet, 'Excess (Return %)':exceRet, 'Strategy (Volatility %)':perStd }, index = ['2010', '2011', '2012', '2013', '2014'])
resTable.index.name = 'Year'
resTable.plot(kind = 'bar', figsize = (14,8), legend = True)
```

```
<matplotlib.axes.AxesSubplot at 0x97a3050>
```



```
resTable
```

	Excess (Return %)	Strategy (Return %)	Strategy (Volatility %)
Year			
2010	8.481646	-6.327046	19.742343
2011	12.891790	-16.194009	17.954727
2012	6.545174	14.197018	17.604919
2013	18.062832	8.099492	18.518870
2014	16.344165	49.163501	19.841940

```
5 rows × 3 columns
```

7. 瑞和300期权投资策略的优势与缺陷

优势：

- 非常容易在实际中操作，只在二级市场买卖，不涉及申购赎回等复杂操作；
- 不需要配对交易，可以放大交易量
- 指数增强，正确捕获阿尔法；

- 策略参数少，只有一个时间窗口参数，很大程度上规避了过拟合问题。

劣势：

- 未使用对冲，无法降低原始指数的回撤以及波动率；
- 放弃了在到点折算日套利的机会。
- 以上的缺陷都是未来我们需要进一步研究的地方。

基于期权定价的兴全合润基金交易策略

> 来源：https://uqer.io/community/share/5534bc30f9f06c8f3390468f

版本：1.1

作者：李丞

联系：cheng.li@datayes.com

摘要

分级基金是中国金融市场化下创新的产物，多数是以AB端分级，A端获取相对保守收益，B端获取杠杆收益的结构。通俗的讲，在分级基金结构中，大多数情况下，B端优先承受市场风险损失，换取A端“借”给它钱投资的融资优势。

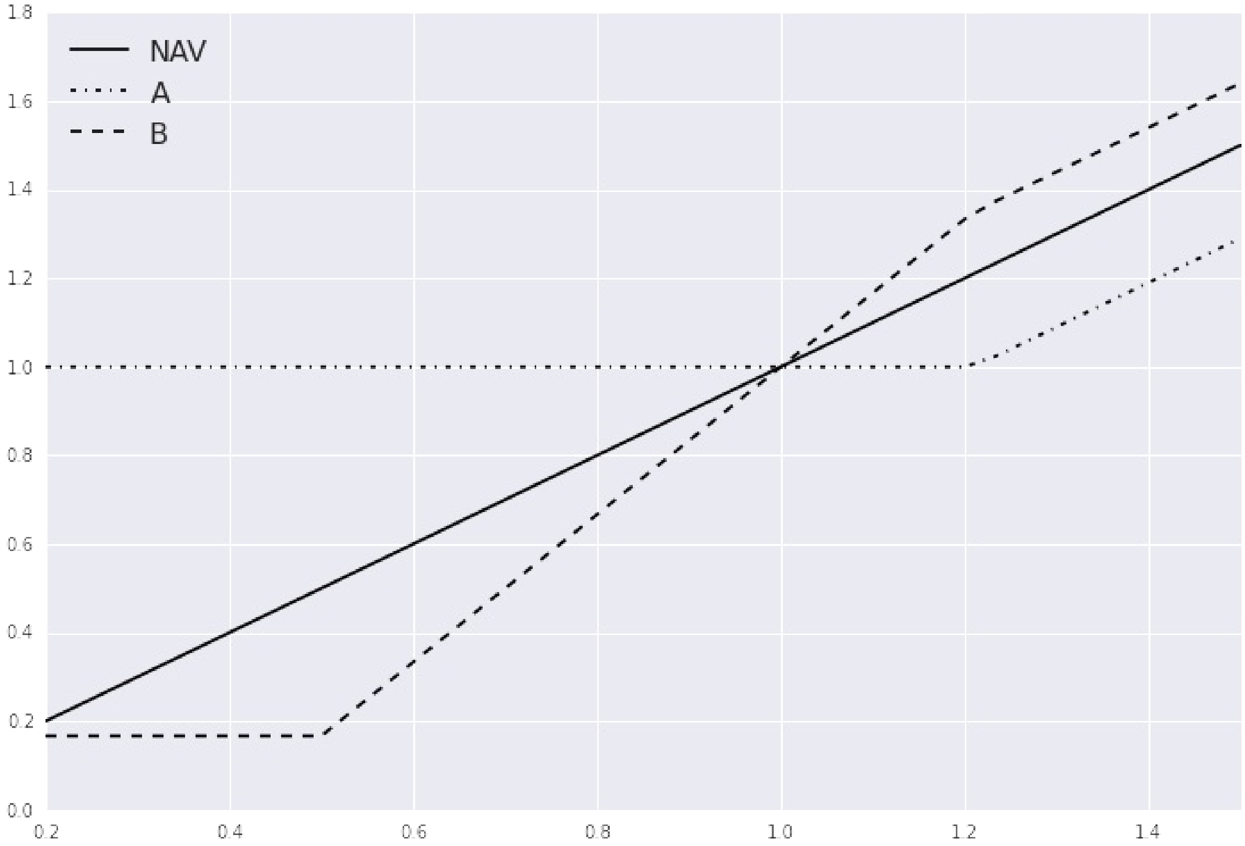
现在市场上大多数的指数型分级基金采取的收益分配模式为：A端获取固定的约定收益率，多半为一年期定存+x%；B端获取剩余的母基金净资产。这样的分级基金可以看做A端是一个固定利率债券，B端是一个看涨期权，其中的期权卖方恰恰是A端。在这里我们不会详细探讨这一类型的结构，关于这一类型分级基金的期权分析可以参考[1]。

这里我们会看一个有趣的产品，在这个产品中，A、B端都是期权形式的[1]。这个产品就是兴全合润分级基金.

1. 兴全合润期权结构分析

收益的结构最容易以一张图的形式表示出来：

```
from matplotlib import pyplot
import numpy as np
import pandas as pd
import seaborn as sns
def AReturn(base):
    if base < 1.21:
        return 1.0
    else:
        return base - 0.21

def BReturn(base):
    if base < 0.5:
        return 0.1 / 0.6
    elif base < 1.21:
        return (base - 0.4) / 0.6
    else:
        return base + 0.14

xspace = np.linspace(0.2, 1.5, 40)

aSeries = [AReturn(x) for x in xspace]
bSeries = [BReturn(x) for x in xspace]

pyplot.figure(figsize=(12,8))
pyplot.plot(xspace, xspace, '-k')
pyplot.plot(xspace, aSeries, '-.k')
pyplot.plot(xspace, bSeries, '--k')
pyplot.xlim((0.2,1.5))
pyplot.legend(['NAV', 'A', 'B'], loc = 'best', fontsize = 16)

<matplotlib.legend.Legend at 0x64f2d10>
```



收益的描述也可以用下式描述，其中 NAV 为母基金净值：

$$A = 1.0 + \mathrm{MAX}(0.0, \mathrm{NAV} - 1.21),$$

$$B = -\frac{2}{3} + \frac{5}{3} \times \mathrm{NAV} - \frac{2}{3} \times \mathrm{MAX}(0.0, \mathrm{NAV} - 1.21)$$

可以看到，这两个子基金的价值都是三个期权的组合，只是权重不同：

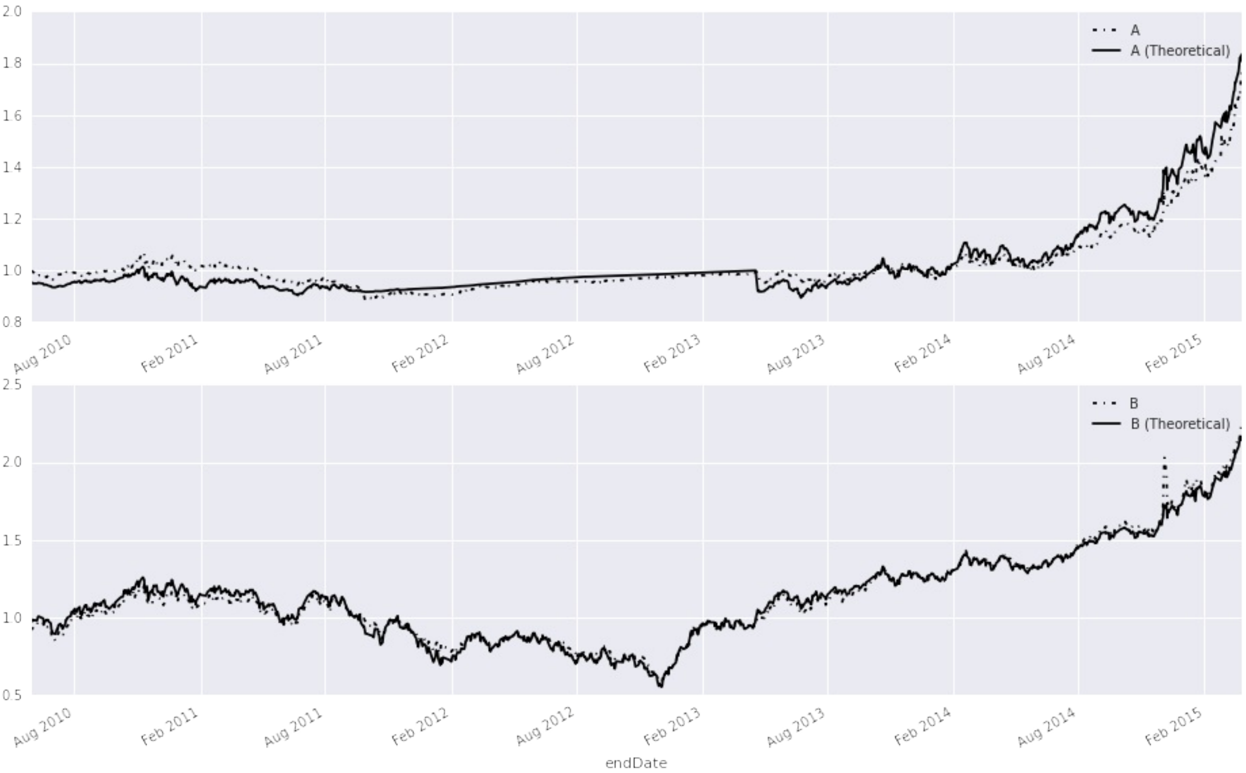
- A份额买入一份行权价为1.21元的期权
- B份额买入5/3份行权价为0的看涨期权，同时卖出2/3份行权价为1.21元的看涨期权

对于这些期权，我们可以假设标的即为母基金净值，期限为当前日期到下一个折算日（下一个折算日为2016年4月22日），无风险利率使用3个月Shibor做简单的近似：

```
# 导入需要的模块
from CAL.PyCAL import *

# 读入外部行情数据
data = pd.read_excel(r'xqhr.xlsx','Sheet1')

riskFree = data['Shibor 3M'] / 100.0
maturity = data['Maturity']
spot = data['163406.OFCN']
ATarget = data['150016.XSHE']
BTarget = data['150017.XSHE']

def AOptionPrice(vol, riskFree, maturity, spot):
    price1 = BSMPrice(1, 1.21, spot, riskFree, 0.0, vol[0], maturity, rawOutput = True)
    return  1.0*np.exp(-riskFree*maturity) + price1[0]

def BOptionPrice(vol, riskFree, maturity, spot):
    price1 = BSMPrice(1, 1.21, spot, riskFree, 0.0, vol[0], maturity, rawOutput = True)
    return  -2.0/3.0*np.exp(-riskFree*maturity) + 5.0/3.0 * spot - 2.0/3.0 * price1[0]

aTheoreticalPrice = AOptionPrice([0.09], riskFree, maturity, spot)
bTheoreticalPrice = BOptionPrice([0.09], riskFree, maturity, spot)
```

我们分别看一下，AB端基金理论价格和实际收盘价之间的关系（上面的计算中假设波动率为15%）：可以看到，基本上理论价格和真实价格的变动是完全同向的，但是存在价差，A长期折价，B长期溢价。这个价差随着到期折算日的接近，收敛至0。这个是与期权的性质是完全一致的。

```
data['A (Theoretical)'] = aTheoreticalPrice
data['B (Theoretical)'] = bTheoreticalPrice
pyplot.figure(figsize = (16,10))
ax1 = pyplot.subplot('211')
data.plot('endDate', ['150016.XSHE','A (Theoretical)'], style = ['-.k', '-k'])
ax1.legend(['A', 'A (Theoretical)'], loc = 'best')
ax2 = pyplot.subplot('212')
data.plot('endDate', ['150017.XSHE','B (Theoretical)'], style = ['-.k', '-k'])
ax2.legend(['B', 'B (Theoretical)'], loc = 'best')

<matplotlib.legend.Legend at 0x7134610>
```



2. 兴全合润的期权投资策略

根据上面的分析，似乎读者可以得到这样的一个印象：A端长期比较便宜，应该直接持有A端，眞的是这样吗？这里面实际上有以下的问题：

- A端由于收益算法的原因，属于类固定收益产品，并且它在标的价格高企时，凸性为负；
- B端属于杠杆类型，在标的股价高企时，凸性为正；
- 市场可能会对凸性的不同，对于AB端分别进行折溢价调整；
- 15%是一个魔幻数（Magic Number)，眞实市场波动率水平显然不应该是一个常値。

这里我们将涉及一个策略，试着解释最后一个问题。期权有一种估计期权的方法，称为“隐含波动率”。我们可以把类似的想法引入我们这个产品当中，每天收盘的时候，我们可以观察到A端和B端的价格（或者说两个期权组合的价格）。这时候，可以使用优化的方法，找到一个波动率水平使得理论价格在某种标准下与实际价格差异最小。我们把这个波动率水平称之为瑞和300的“隐含波动率”。

有了这个隐含波动率水平，我们可以再计算理论价格，这时候计算而得的理论价格，我们可以认为是“眞实”市场估计下的理论期权价値。用这个价格作为评估的标准，比较A端与B端那个更加便宜，从而决定购买哪个产品。下面的策略中，我们即使用上面介绍的办法，每天调仓，根据估价的高低，分别购买A端（B端），同时卖出B端（A端）。这个策略只在二级市场中进行交易。具体的参数如下：

本策略的参数如下：

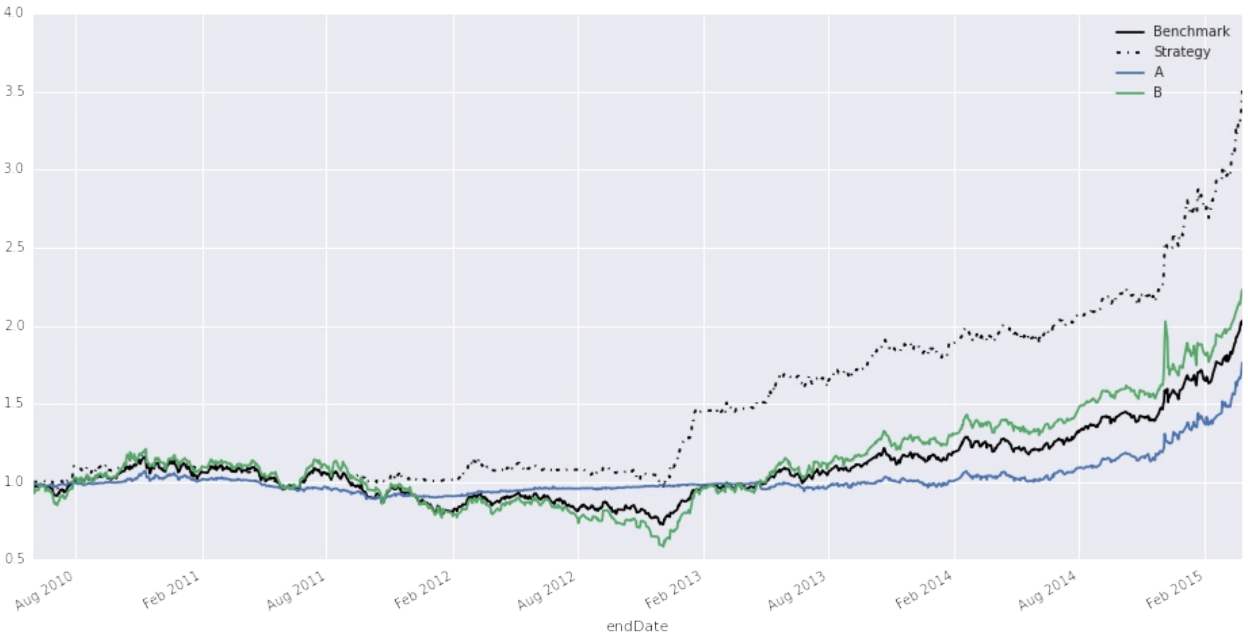
- 起始日期：2010年5月31日
- 结束日期：2015年3月27日

- 起始资金：100000元
- 调仓周期：每个交易日

```
prices1 = []
prices2 = []
aTarget = []
bTarget = []

def processDate(record):

    riskFree = record['Shibor 3M'] / 100.0
    maturity = record['Maturity']
    spot = record['163406.OFCN']
    ATarget = record['150016.XSHE']
    BTarget = record['150017.XSHE']

    def errorFunction(vol):

        price1 = AOptionPrice(vol, riskFree, maturity, spot)
        price2 = BOptionPrice(vol, riskFree, maturity, spot)

        return (price1 - ATarget)**2 + (price2 - BTarget)**2

    out, fx, its, imode, smode = optimize.fmin_slsqp(errorFunction, [0.15], bounds = [(0.01, 0.25)], epsilon = 1e-6, iter = 10000, disp = False, full_output = True, acc = 1e-16)

    price1 = AOptionPrice(out, riskFree, maturity, spot)
    price2 = BOptionPrice(out, riskFree, maturity, spot)

    prices1.append(price1)
    prices2.append(price2)
    aTarget.append(ATarget)
    bTarget.append(BTarget)

    return price1 - ATarget, price2 - BTarget
```

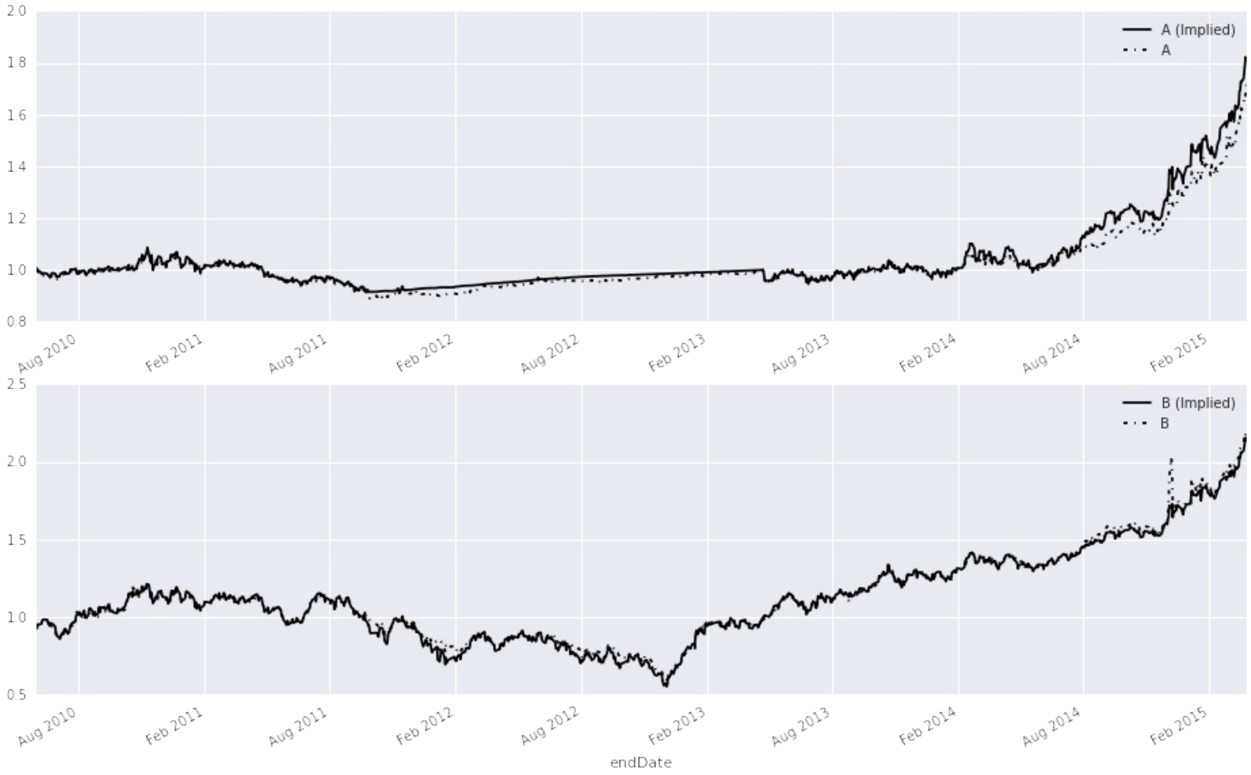
```
import datetime as dt
from scipy import optimize

callDate = [dt.datetime(2013,4,19)]

class deque:

    def __init__(self, maxlen):
        self.maxlen = maxlen
        self.cont = []

    def append(self,vec):
        self.cont.append(vec)
        if len(self.cont)>self.maxlen:
            self.cont = self.cont[len(self.cont) - self.maxlen:]

    def __item__(self, i):
        return self.cont[i]

    def average(self):
        sum = 0.0
        for i in xrange(len(self.cont)):
            sum += self.cont[i]
        return sum / float(len(self.cont))

class Account:

    def __init__(self, cash, commission = 0.0005):

        self.aAmount = 0
        self.bAmount = 0
        self.commission = commission
        self.cash = cash

    def order(self, amount, fundType, price):
        if fundType.upper() == 'A':
            self.aAmount += amount
            if amount> 0:
                self.cash -= amount * price * (1.0 + self.commis
sion)
            else:
                self.cash -= amount * price * (1.0 - self.commis
sion)
        elif fundType.upper() == 'B':
            self.bAmount += amount
            if amount> 0:
                self.cash -= amount * price * (1.0 + self.commis
sion)
            else:
                self.cash -= amount * price * (1.0 - self.commis
sion)

    def currentValue(self, aQuote, bQuote):
        return self.aAmount * aQuote + self.bAmount * bQuote + s
elf.cash

def BackTesting(data, window = 20, startAmount = 100000, tradeVo
l = 2000):

    account = Account(startAmount)

    aWindow = deque(maxlen = window)
    bWindow = deque(maxlen = window)
    performance = [startAmount]
    aVol = [0]
    bVol = [0]
    cash = [startAmount]
    for i in xrange(1, len(data)):
        previousDay = data.loc[i-1]
        aUnderEstimated, bUnderEstimated  = processDate(previousDay)
        aWindow.append(aUnderEstimated)
        bWindow.append(bUnderEstimated)

        aAverage = aWindow.average()
        bAverage = bWindow.average()

        today = data.loc[i]
        aPrice = today['150016.XSHE']
        bPrice = today['150017.XSHE']
        if i >= 5:
            # 如果分级A端相对于B端更便宜
            if aUnderEstimated - aAverage > bUnderEstimated - bAverage:

                if account.cash > tradeVol:
                    account.order(tradeVol, 'A', aPrice)
                if account.bAmount >0:
                    account.order(-tradeVol, 'B', bPrice)

            # 如果分级B端相对于A端更便宜
            elif aUnderEstimated - aAverage < bUnderEstimated - bAverage:
                if account.cash > tradeVol:
                    account.order(tradeVol, 'B', bPrice)
                if account.aAmount >0:
                    account.order(-tradeVol, 'A', aPrice)

            for calDate in callDate:
                if today['endDate'] == calDate:
                    account.order(-account.aAmount, 'A', aPrice)
                    account.order(-account.bAmount, 'B', bPrice)


        performance.append(account.currentValue(aPrice, bPrice))
        aVol.append(account.aAmount)
        bVol.append(account.bAmount)
        cash.append(account.cash)


    originalReturn = data[['150016.XSHE', '150017.XSHE', '163406.OFCN']].values
    start = originalReturn[0]
    originalReturn[0] = 1.0
    dates = data['endDate']
    scalar = 1.0
    count = 0
    for i in xrange(1, len(originalReturn)):
        if count < len(callDate) and dates[i-1] == callDate[coun
t]:
            start = originalReturn[i]
            originalReturn[i] =  originalReturn[i-1]
            count += 1
        else:
            scalar = originalReturn[i] / start
            start = originalReturn[i]
            originalReturn[i] = originalReturn[i-1] * scalar
    scalar = float(performance[0])
    performance = [p / scalar for p in performance]
    return pd.DataFrame({'Performance':performance, '150016.XSHE'
: aVol, '150017.XSHE': bVol, 'Cash': cash, '163406.OFCN': data['
163406.OFCN'].values, 'A Performance': originalReturn[:,0], 'B P
erformance': originalReturn[:,1],'Benchmark Return':originalRetu
rn[:,2]} ,index = data.endDate)
```

```
bt = BackTesting(data, tradeVol = 20000)
bt.plot(y = ['Benchmark Return', 'Performance', 'A Performance',
'B Performance'], figsize = (16,8), style = ['-k', '-.k'])
pyplot.legend( ['Benchmark', 'Strategy', 'A', 'B'], loc = 'best'
)

<matplotlib.legend.Legend at 0x7285510>
```



由上图可知，这样的策略是比较典型的指数增强型策略

```
res = pd.DataFrame({'A (Implied)': prices1, 'B (Implied)':prices
2, 'A': aTarget, 'B': bTarget}, index = data.endDate[1:])
pyplot.figure(figsize = (16,10))
ax1 = pyplot.subplot('211')
res.plot(y = ['A (Implied)', 'A'], style = ['-k', '-.k'])
pyplot.legend(['A (Implied)', 'A'])
ax1 = pyplot.subplot('212')
res.plot(y = ['B (Implied)', 'B'], style = ['-k', '-.k'])
pyplot.legend(['B (Implied)', 'B'])

<matplotlib.legend.Legend at 0x7804290>
```

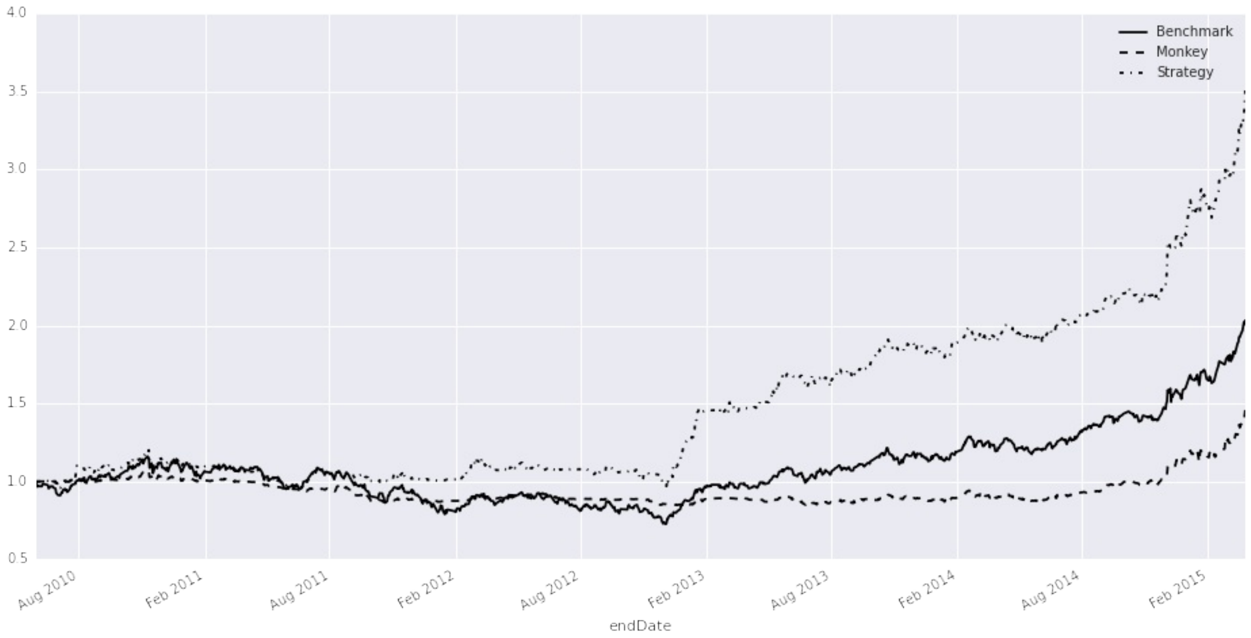


3. 我们是否能够比“猴子”做的更好？

作为和该策略的比较，我们可以使用一个随机投资的做法。让我们看看，和“猴子”(Monky Random Choice Strategy）比，我们是否能够做的更好？

```
def BackTesting2(data, window = 20, startAmount = 100000, tradeV
ol = 2000):

    account = Account(startAmount)

    performance = [startAmount]
    aVol = [0]
    bVol = [0]
    cash = [startAmount]
    s = MersenneTwister19937UniformRsg(seed = 1234)
    for i in xrange(1, len(data)):
        previousDay = data.loc[i-1]
        aUnderEstimated, bUnderEstimated  = processDate(previous
Day)

        today = data.loc[i]
        aPrice = today['150016.XSHE']
        bPrice = today['150017.XSHE']
        if i >= 5:
            # 如果随机数>0.5
            if s.nextSequence()[0] > 0.5:

                if account.cash > tradeVol:
                    account.order(tradeVol, 'A', aPrice)
                if account.bAmount >0:
                    account.order(-tradeVol, 'B', bPrice)

            # 如果随机数<0.5
            elif s.nextSequence()[0] < 0.5:
                if account.cash > tradeVol:
                    account.order(tradeVol, 'B', bPrice)
                if account.aAmount >0:
                    account.order(-tradeVol, 'A', aPrice)

            for calDate in callDate:
                if today['endDate'] == calDate:
                    account.order(-account.aAmount, 'A', aPrice)
                    account.order(-account.bAmount, 'B', bPrice)


        performance.append(account.currentValue(aPrice, bPrice))
        aVol.append(account.aAmount)
        bVol.append(account.bAmount)
        cash.append(account.cash)


    originalReturn = list(data['163406.OFCN'].values)
    start = originalReturn[0]
    originalReturn[0] = 1.0
    dates = data['endDate']
    scalar = 1.0
    count = 0
    for i in xrange(1, len(originalReturn)):
        if count < len(callDate) and dates[i-1] == callDate[coun
t]:
           start = originalReturn[i]
           originalReturn[i] =  originalReturn[i-1]
           count += 1
        else:
           scalar = originalReturn[i] / start
           start = originalReturn[i]
           originalReturn[i] = originalReturn[i-1] * scalar
    scalar = float(performance[0])
    performance = [p / scalar for p in performance]
    return pd.DataFrame({'Performance':performance, '150016.XSHE'
: aVol, '150017.XSHE': bVol, 'Cash': cash, '163406.OFCN': data['
163406.OFCN'].values, 'Benchmark Return':originalReturn } ,index
 = data.endDate)
```

```
bt1 = BackTesting(data, tradeVol = 20000)
bt2 = BackTesting2(data, tradeVol = 20000)
bt1['Monky'] = bt2['Performance']
bt1.plot(y = ['Benchmark Return', 'Monky', 'Performance'], figsi
ze = (16,8), style = ['-k', '--k', '-.k'])
pyplot.legend( ['Benchmark', 'Monkey', 'Strategy'], loc = 'best'
)

<matplotlib.legend.Legend at 0x7804150>
```



结果令人满意，我们的期权投资比随机选择的结果好的多。我们看到如果随机投资，“猴子”式的选择并不能显著的击败标的母基金。但是我们的期权投资策略还是可以保持的持续性的跑赢指数以及随机选择。

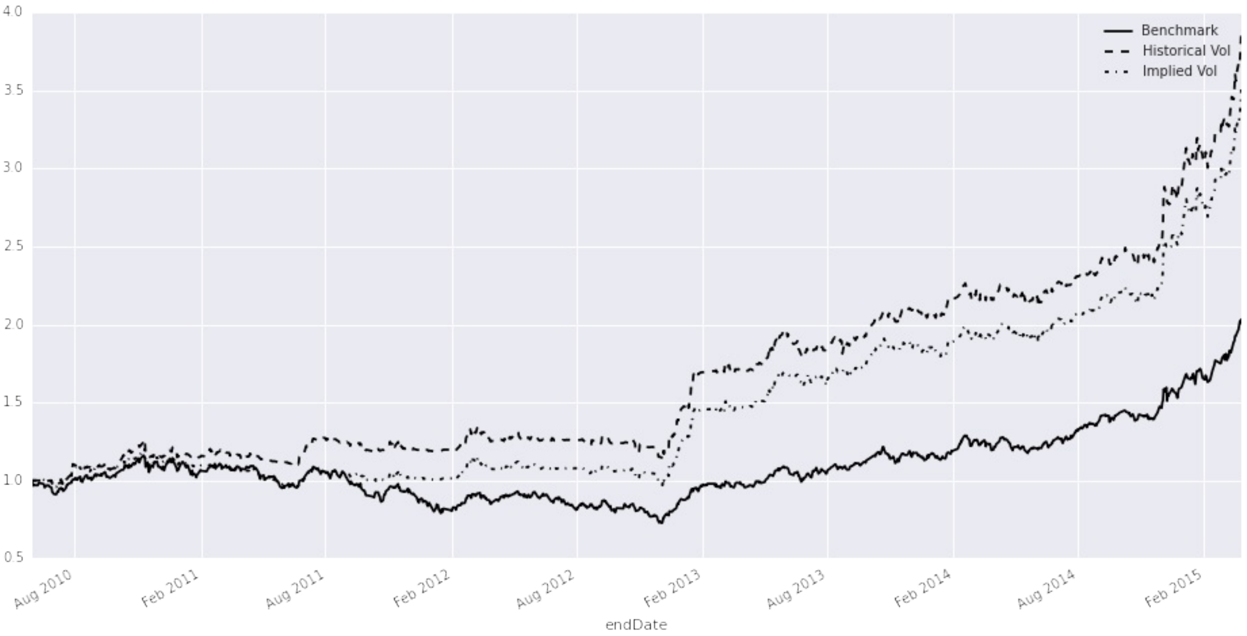
4. 使用历史波动率

这里我们给了一个使用历史波动率计算折溢价水平，与之前使用的隐含波动率方法进行比较。这里使用的历史波动率水平是20天年化收益标准差。结果上，我们无法显著区别这两种波动率算法在表现上面的区别。但是他们都可以显著的击败标的母基金。

```
def processDate2(record):

    riskFree = record['Shibor 3M'] / 100.0
    maturity = record['Maturity']
    spot = record['163406.OFCN']
    ATarget = record['150016.XSHE']
    BTarget = record['150017.XSHE']
    volatility = record['volatility']

    vol = [volatility]

    price1 = AOptionPrice(vol, riskFree, maturity, spot)
    price2 = BOptionPrice(vol, riskFree, maturity, spot)

    return price1 - ATarget, price2 - BTarget

def BackTesting3(data, window = 20, startAmount = 100000, tradeV
ol = 2000):

    account = Account(startAmount)

    aWindow = deque(maxlen = window)
    bWindow = deque(maxlen = window)
    performance = [startAmount]
    aVol = [0]
    bVol = [0]
    cash = [startAmount]
    for i in xrange(1, len(data)):
        previousDay = data.loc[i-1]
        aUnderEstimated, bUnderEstimated  = processDate2(previou
sDay)
        aWindow.append(aUnderEstimated)
        bWindow.append(bUnderEstimated)

        aAverage = aWindow.average()
        bAverage = bWindow.average()

        today = data.loc[i]
        aPrice = today['150016.XSHE']
        bPrice = today['150017.XSHE']
        if i >= 5:
            # 如果分级A端相对于B端更便宜
            if aUnderEstimated - aAverage > bUnderEstimated - bA
verage:

                if account.cash > tradeVol:
                    account.order(tradeVol, 'A', aPrice)
                if account.bAmount >0:
                    account.order(-tradeVol, 'B', bPrice)

            # 如果分级B端相对于A端更便宜
            elif aUnderEstimated - aAverage < bUnderEstimated -
```

```
bAverage:
                if account.cash > tradeVol:
                    account.order(tradeVol, 'B', bPrice)
                if account.aAmount >0:
                    account.order(-tradeVol, 'A', aPrice)

            for calDate in callDate:
                if today['endDate'] == calDate:
                    account.order(-account.aAmount, 'A', aPrice)
                    account.order(-account.bAmount, 'B', bPrice)


        performance.append(account.currentValue(aPrice, bPrice))
        aVol.append(account.aAmount)
        bVol.append(account.bAmount)
        cash.append(account.cash)


    originalReturn = list(data['163406.OFCN'].values)
    start = originalReturn[0]
    originalReturn[0] = 1.0
    dates = data['endDate']
    scalar = 1.0
    count = 0
    for i in xrange(1, len(originalReturn)):
        if count < len(callDate) and dates[i-1] == callDate[coun
t]:
           start = originalReturn[i]
           originalReturn[i] =  originalReturn[i-1]
           count += 1
        else:
           scalar = originalReturn[i] / start
           start = originalReturn[i]
           originalReturn[i] = originalReturn[i-1] * scalar
    scalar = float(performance[0])
    performance = [p / scalar for p in performance]
    return pd.DataFrame({'Performance':performance, '150016.XSHE'
: aVol, '150017.XSHE': bVol, 'Cash': cash, '163406.OFCN': data['
163406.OFCN'].values, 'Benchmark Return':originalReturn } ,index
 = data.endDate)
```

```
bt3 = BackTesting3(data, tradeVol = 20000)
bt1['Historical (Vol)'] = bt3['Performance']
bt1.plot(y = ['Benchmark Return', 'Historical (Vol)', 'Performan
ce'], figsize = (16,8), style = ['-k', '--k', '-.k'])
pyplot.legend( ['Benchmark', 'Historical Vol', 'Implied Vol'], l
oc = 'best')

<matplotlib.legend.Legend at 0x7841410>
```

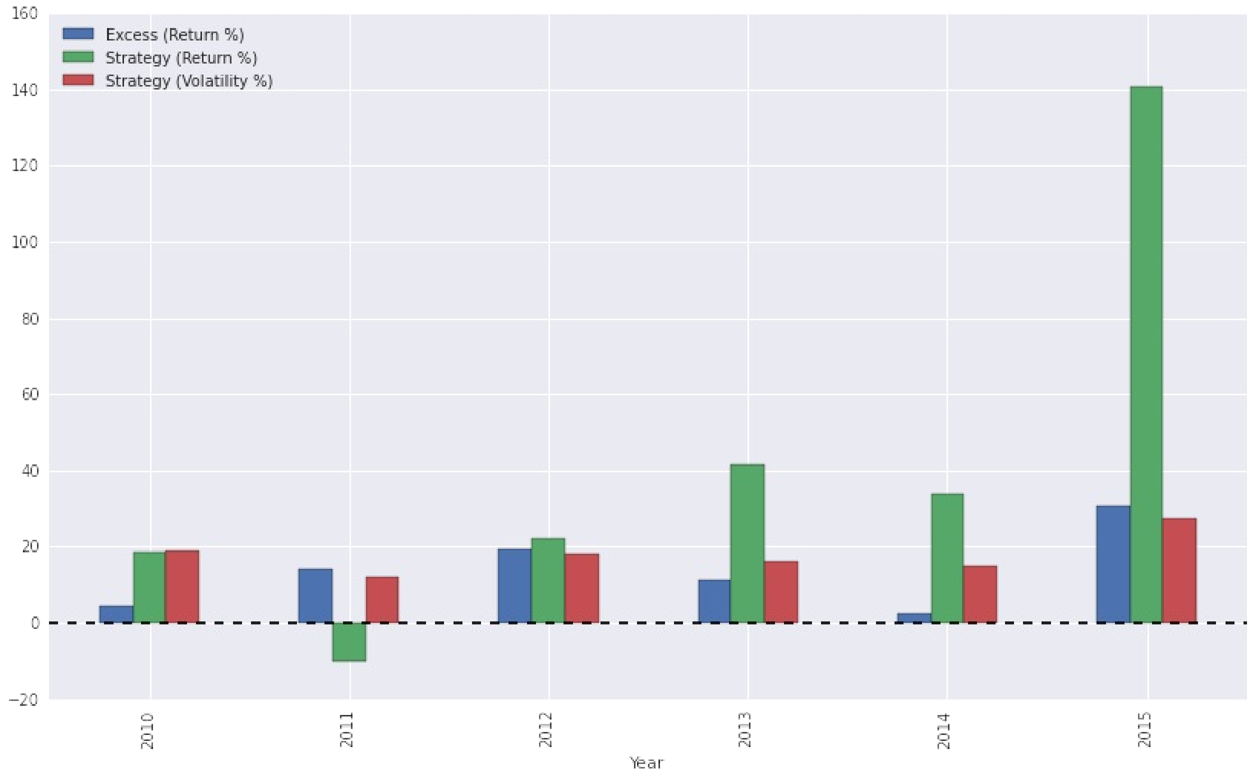


5. 风险收益分析

下面我们按照每个自然年评估策略的绩效。可以看到在5个自然年中，只有一年的收益为负；更值得关注的是，这个策略相对于原策略都录得了正的超额收益。

```
value = bt1[['Performance', 'Benchmark Return']]
value['endDate'] = value.index.values
returnRes = [0]
tmp = np.log(value['Performance'][1:].values/ value['Performance'
][:-1].values)
returnRes.extend(tmp)
value['Per. Return'] = returnRes
returnRes = [0]
tmp = np.log(value['Benchmark Return'][1:].values/ value['Benchm
ark Return'][:-1].values)
returnRes.extend(tmp)
value['Benchmark Return'] = returnRes
year2010 = value[(value['endDate'] > Date(2010,1,1).toTimestamp(
)) & (value['endDate'] <= Date(2010,12,31).toTimestamp())]
year2011 = value[(value['endDate'] > Date(2011,1,1).toTimestamp(
)) & (value['endDate'] <= Date(2011,12,31).toTimestamp())]
year2012 = value[(value['endDate'] > Date(2012,1,1).toTimestamp(
)) & (value['endDate'] <= Date(2012,12,31).toTimestamp())]
year2013 = value[(value['endDate'] > Date(2013,1,1).toTimestamp(
)) & (value['endDate'] <= Date(2013,12,31).toTimestamp())]
year2014 = value[(value['endDate'] > Date(2014,1,1).toTimestamp(
)) & (value['endDate'] <= Date(2014,12,31).toTimestamp())]
year2015 = value[(value['endDate'] > Date(2015,1,1).toTimestamp(
)) & (value['endDate'] <= Date(2015,12,31).toTimestamp())]

days = 252
```

```
def perRes(yearRes):
    yearRes['Excess Return'] = yearRes['Per. Return'] - yearRes[
'Benchmark Return']
    mean = yearRes.mean() * days * 100
    std = yearRes.std() * np.sqrt(days) * 100

    return mean['Per. Return'], mean['Excess Return'], std['Per.
 Return']

res2010 = perRes(year2010)
res2011 = perRes(year2011)
res2012 = perRes(year2012)
res2013 = perRes(year2013)
res2014 = perRes(year2014)
res2015 = perRes(year2015)

perRet = []
exceRet= []
perStd = []

for res in [res2010, res2011, res2012, res2013, res2014, res2015
]:
    perRet.append(res[0])
    exceRet.append(res[1])
    perStd.append(res[2])

resTable = pd.DataFrame({'Strategy (Return %)':perRet, 'Excess (
Return %)':exceRet, 'Strategy (Volatility %)':perStd }, index = [
'2010', '2011', '2012', '2013', '2014', '2015'])
resTable.index.name = 'Year'
resTable.plot(kind = 'bar', figsize = (14,8), legend = True)

<matplotlib.axes.AxesSubplot at 0x82f4b50>
```


Excess (Return %)
Strategy (Return %)
Strategy (Volatility %)
160
140
120
100
80
60
40
20
0
−20
2010
2011
2012
2013
2014
2015
Year

二 基金分析

Alpha 基金“黑天鹅事件” -- 思考以及原因

> 来源：https://uqer.io/community/share/54b39717f9f06c276f651a0d

0. 引言

2014年11月底至2014年12月初的那一周，在市场不断冲高的节奏下，alpha型对冲基金却遭遇了集体的滑铁卢，最高单周跌幅可以达到11%。这里面到底发生了什么？ 本文思想以及部分数据参考自[1]

1. 风格因子

基于Fama-French经典的因子模型，这里我们考虑代表三种不同投资风格的因子：“市场”、“规模”、“价值”。

市场

市场因子反映了市场当前的趋势，是代表最广泛的变动趋势，是全市场的“动量”方向，这里我们选取了中证800指数；

规模

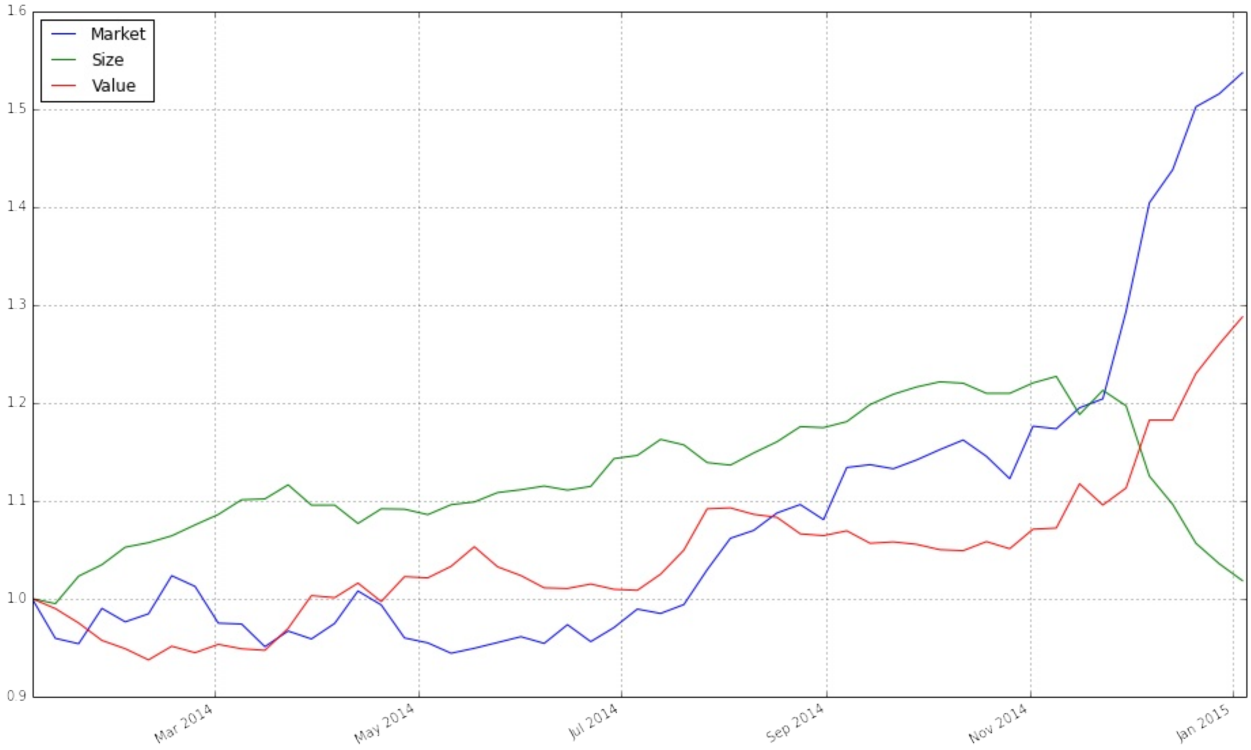
规模因子反映了市场对公司规模的折溢价观点。这里我们按照最初的Fama设想，买入小规模市值股票组合，卖出大规模市值股票组合。这里我们实际选取的组合依据是小盘风格指数以及大盘风格指数。

价值

价值因子反映了市场对公司估值的折溢价观点。这里我们按照最初的Fama设想，买入低估值股票组合，卖出高成长股票组合。这里我们实际选取的组合依据是价值风格指数以及成长风格指数

下图中我们可以看到这三种投资风格，2014年的整体走势。我们可以看到经过上半年的蛰伏，下半年市场因子异军突起，将规模和价值因子牢牢的甩在身后。当价值因子亦步亦趋的追赶市场的步伐的时候，规模因子在11月底12月初来了个高台跳水，丢失了上半年所有的成果。这一现象与12月后蓝筹起舞，小票低迷的市场现状是一致的。

```
from matplotlib import pyplot as plt
factorData = pd.read_excel('三因子数据.xlsx','Sheet1',index_col = 0
)
factorData.plot(figsize = (16,10))
plt.legend(['Market', 'Size', 'Value'], loc = 'best')

<matplotlib.legend.Legend at 0x594d0d0>
```



我们也可以看到这几个因子之间收益的相关性，显著的低于一般市场指数之间的相关性，确实体现了风格上的差别

```
factorData.pct_change()[1:].corr()
```

	市场收益	规模	价值
市场收益	1.000000	-0.437854	0.412471
规模	-0.437854	1.000000	-0.739627
价值	0.412471	-0.739627	1.000000

2. 风格分析

为了探究alpha基金在2014年11月末12月初这一周中“黑天鹅”事件的原因，我们选取了38只有每周净值数据的alpha型私募基金。选取的日期时间为2014年8月至2014年12月7日，在这段时间内以上基金都有数据。我们使用风格归因的方法，从这些基金的历史收益率情况猜测出他们的投资风格。

```
alphaData = pd.read_excel('alpha基金数据.xlsx','Sheet1', index_co
l = 0)
```

这些基金的名称如下：

```
for name in alphaData.columns.values[3:]:
    print name
```

```
安进1号大岩对冲
安进1号尊享K期
安进1号尊享L期
安进1号尊享O期
安进1号尊享P期
安进1号大岩对冲尊享C期
安进尊享F期
方正富邦基金-高程量化1号
龙旗扶翼量化对冲
盈融达量化对冲1期
盈融达量化对冲2期
盈融达量化对冲5期
盈融达量化对冲6期
盈融达量化对冲7期
杉杉青骓量化对冲1期
朱雀漂亮阿尔法
朱雀阿尔法7号
朱雀阿尔法8号
朱雀投资阿尔法2号
尊嘉ALPHA
尊嘉ALPHA尊享B期
宁聚爬山虎1期
宁聚稳进
宁聚量化对冲1期
金锝2号
金锝5号
金锝5号尊享A期
金锝5号尊享B期
金锝6号
金锝6号尊享A期
金锝量化
通和量化对冲2期
中信富享1期
中信富享2期
翼虎量化对冲
翼虎量化对冲2期
翼虎量化对冲3期
中钢投资套利优选
```

我们将他们的净值数据与前节中提到的因子数据合并起来：

```
alphaData
```

	市场	规模	价值	安进1 号大岩	安进1 号尊享	安进1 号尊享	安 号

日期							
2014-08-03	1.000000	1.000000	1.000000	0.9762	0.9686	0.9686	0.
2014-08-10	1.007588	1.010957	0.994005	0.9840	0.9764	0.9763	0.
2014-08-17	1.024355	1.020982	0.991207	0.9904	0.9827	0.9827	0.
2014-08-24	1.032469	1.034579	0.975795	0.9961	0.9884	0.9884	0.
2014-08-31	1.017946	1.033698	0.974094	0.9900	0.9823	0.9823	0.
2014-09-07	1.068116	1.039119	0.978613	0.9948	0.9871	0.9871	0.
2014-09-14	1.070794	1.054439	0.966962	1.0038	0.9960	0.9960	0.
2014-09-21	1.066904	1.063699	0.968235	1.0066	0.9988	0.9988	0.
2014-09-28	1.075369	1.070250	0.965948	1.0176	1.0097	1.0097	1.
2014-10-05	1.085320	1.074879	0.960948	1.0293	1.0213	1.0213	1.
2014-10-12	1.094552	1.073642	0.960053	1.0332	1.0252	1.0252	1.
2014-10-19	1.078820	1.064507	0.968506	1.0254	1.0174	1.0174	1.
2014-10-26	1.057308	1.064504	0.961861	1.0312	1.0232	1.0232	1.
2014-11-02	1.107830	1.073874	0.980116	1.0414	1.0333	1.0333	1.
2014-11-09	1.105356	1.079756	0.981184	1.0470	1.0389	1.0389	1.
2014-11-16	1.125537	1.045483	1.022599	1.0344	1.0264	1.0264	1.
2014-11-23	1.134135	1.067249	1.002712	1.0342	1.0262	1.0262	1.
2014-							

2014-11-30	1.218464	1.053291	1.018619	1.0450	1.0369	1.0369	1.
2014-12-07	1.322762	0.990118	1.082090	1.0040	0.9962	0.9962	0.

```
19 rows × 41 columns
```

将价格数据转换为收益率数据：这里我们将三个因子的收益率数据做了标准化处理，这样方便比较后面的因子权重。

```
returnData = alphaData.pct_change()
returnData = returnData[1:]

returnData[u'市场']  = returnData[u'市场'] / returnData[u'市场'].s
td() / 100.0
returnData[u'规模']  = returnData[u'规模'] / returnData[u'规模'].s
td() / 100.0
returnData[u'价值']  = returnData[u'价值'] / returnData[u'价值'].s
td() / 100.0
returnData
```

	市场	规模	价值	安进1号 大岩对冲	安进1号 尊享K期	安 尊
日期						
2014-08-10	0.002575	0.005755	-0.002978	0.007990	0.008053	0.0
2014-08-17	0.005647	0.005208	-0.001398	0.006504	0.006452	0.0
2014-08-24	0.002688	0.006995	-0.007724	0.005755	0.005800	0.0
2014-08-31	-0.004773	-0.000447	-0.000866	-0.006124	-0.006172	-0.0
2014-09-07	0.016724	0.002754	0.002304	0.004848	0.004886	0.0
2014-09-14	0.000851	0.007743	-0.005914	0.009047	0.009016	0.0
2014-09-21	-0.001233	0.004612	0.000654	0.002789	0.002811	0.0

09-28	0.002692	0.003235	-0.001173	0.010928	0.010913	0.0
2014-10-05	0.003140	0.002271	-0.002571	0.011498	0.011489	0.0
2014-10-12	0.002887	-0.000604	-0.000463	0.003789	0.003819	0.0
2014-10-19	-0.004877	-0.004469	0.004374	-0.007549	-0.007608	-0.0
2014-10-26	-0.006766	-0.000002	-0.003409	0.005656	0.005701	0.0
2014-11-02	0.016215	0.004623	0.009428	0.009891	0.009871	0.0
2014-11-09	-0.000758	0.002877	0.000542	0.005377	0.005420	0.0
2014-11-16	0.006195	-0.016671	0.020967	-0.012034	-0.012032	-0.0
2014-11-23	0.002592	0.010934	-0.009661	-0.000193	-0.000195	-0.0
2014-11-30	0.025231	-0.006869	0.007881	0.010443	0.010427	0.0
2014-12-07	0.029046	-0.031500	0.030952	-0.039234	-0.039252	-0.0

```
18 rows × 41 columns
```

在这里开始风格归因。我们使用的是经典回归分析的方法，数据截止到2014年11月30日。关于每个基金我们得到3个风格分别的权重，即为回归方程的系数：

```
R=β1×RMarket+β2×RSize+β3×RValue+α
```

例如：“安进1号大岩对冲”的三个系数为：`β1=0.4682`，`β2=0.3556`，`β3=-0.3487`

```
from sklearn import linear_model

cols = returnData.columns[3:]
x = returnData[[u'市场',u'规模',u'价值']][:-1]

market = []
size = []
value = []
intercept = []

for name in cols:

    clf = linear_model.LinearRegression()
    y = returnData[name][:-1]
    clf.fit(x,y)
    market.append(clf.coef_[0])
    size.append(clf.coef_[1])
    value.append(clf.coef_[2])
    intercept.append(clf.intercept_)
```

```
regression = pd.DataFrame({'Market':market, 'Size':size, u'Value'
:value, u'Intercept':intercept}, index = cols)
regression['Return'] = returnData[-1:].values.flatten()[3:]
regression['Name'] = regression.index
regression = regression.reindex(columns = ['Name', 'Return', 'Ma
rket', 'Size', 'Value', 'Intercept'])
regression
```

	Name	Return	Market	Size	Value	Inte
安进1号大岩对冲	安进1号大岩对冲	-0.039234	0.468212	0.355625	-0.348703	0.00
安进1号尊享K期	安进1号尊享K期	-0.039252	0.468560	0.355400	-0.349973	0.00
安进1号尊享L期	安进1号尊享L期	-0.039252	0.468794	0.355543	-0.349783	0.00
安进1号尊享O期	安进1号尊享O期	-0.039159	0.467231	0.360231	-0.343805	0.00
安进1号尊享	安进1号尊享	-0.039234	0.467421	0.353230	-0.349906	0.00

P期	P期					
安进1号大岩对冲尊享C期	安进1号大岩对冲尊享C期	-0.039234	0.468212	0.355625	-0.348703	0.00
安进尊享F期	安进尊享F期	-0.039252	0.468560	0.355400	-0.349973	0.00
方正富邦基金-高程量化1号	方正富邦基金-高程量化1号	0.069128	1.911136	0.266616	-0.004385	-0.00
龙旗扶翼量化对冲	龙旗扶翼量化对冲	-0.036871	0.296329	1.274064	-0.539870	0.00
盈融达量化对冲1期	盈融达量化对冲1期	-0.007880	0.199913	0.341597	-0.379266	0.00
盈融达量化对冲2期	盈融达量化对冲2期	-0.018233	0.018288	0.344163	-0.512896	0.00
盈融达量化对冲5期	盈融达量化对冲5期	-0.016643	0.164744	0.195826	-0.418823	0.00
盈融达量化对冲6期	盈融达量化对冲6期	-0.015752	0.162680	0.240728	-0.468615	0.00
盈融达量化对冲7期	盈融达量化对冲7期	-0.025421	0.047622	0.667812	-0.327798	0.00
杉杉青骓量化对冲1期	杉杉青骓量化对冲1期	-0.000085	0.208083	-0.472288	-0.796191	0.00
朱雀漂亮阿尔法	朱雀漂亮阿尔法	-0.036283	0.205736	0.455439	-0.441430	0.00
朱雀阿尔法7号	朱雀阿尔法7号	-0.041888	0.049495	0.645693	-0.192636	0.00

朱雀阿尔法8号	朱雀阿尔法8号	-0.051708	0.270032	0.097040	-0.881424	0.00
朱雀投资阿尔法2号	朱雀投资阿尔法2号	-0.037950	0.149797	0.388702	-0.478786	0.00
尊嘉ALPHA	尊嘉ALPHA	-0.077634	0.045722	2.030097	-0.015791	-0.00
尊嘉ALPHA尊享B期	尊嘉ALPHA尊享B期	-0.077672	0.045520	2.031095	-0.013126	-0.00
宁聚爬山虎1期	宁聚爬山虎1期	-0.113581	-0.386913	1.870564	-0.323395	0.00
宁聚稳进	宁聚稳进	-0.117121	-0.608006	2.888806	0.261654	0.00
宁聚量化对冲1期	宁聚量化对冲1期	-0.051896	-0.600376	1.973223	0.387574	0.00
金锝2号	金锝2号	-0.057131	-0.408465	0.997068	0.452534	0.00
金锝5号	金锝5号	-0.062897	-0.377828	0.988950	0.380185	0.00
金锝5号尊享A期	金锝5号尊享A期	-0.063031	-0.377760	0.990211	0.382639	0.00
金锝5号尊享B期	金锝5号尊享B期	-0.063167	-0.378665	0.994681	0.385604	0.00
金锝6号	金锝6号	-0.056269	-0.370573	1.040665	0.271547	0.00
金锝6号尊享A期	金锝6号尊享A期	-0.056212	-0.373603	1.037030	0.270727	0.00
金锝量化	金锝量化	-0.052331	-0.381568	0.838623	0.261113	0.00
通和量化对冲	通和量化对冲	-0.062994	0.056065	1.602601	0.487299	-0.00

2期	2期					
中信富享1期	中信富享1期	-0.063006	0.057517	1.596775	0.483519	-0.00
中信富享2期	中信富享2期	-0.063006	0.056753	1.602213	0.486687	-0.00
翼虎量化对冲	翼虎量化对冲	0.026735	0.463962	0.294367	-1.057744	0.00
翼虎量化对冲2期	翼虎量化对冲2期	0.016722	0.724630	-0.209773	-1.360643	0.00
翼虎量化对冲3期	翼虎量化对冲3期	0.027691	0.160118	0.981666	-0.509016	0.00
中钢投资套利优选	中钢投资套利优选	-0.091424	-0.215237	1.264767	-0.165220	0.00

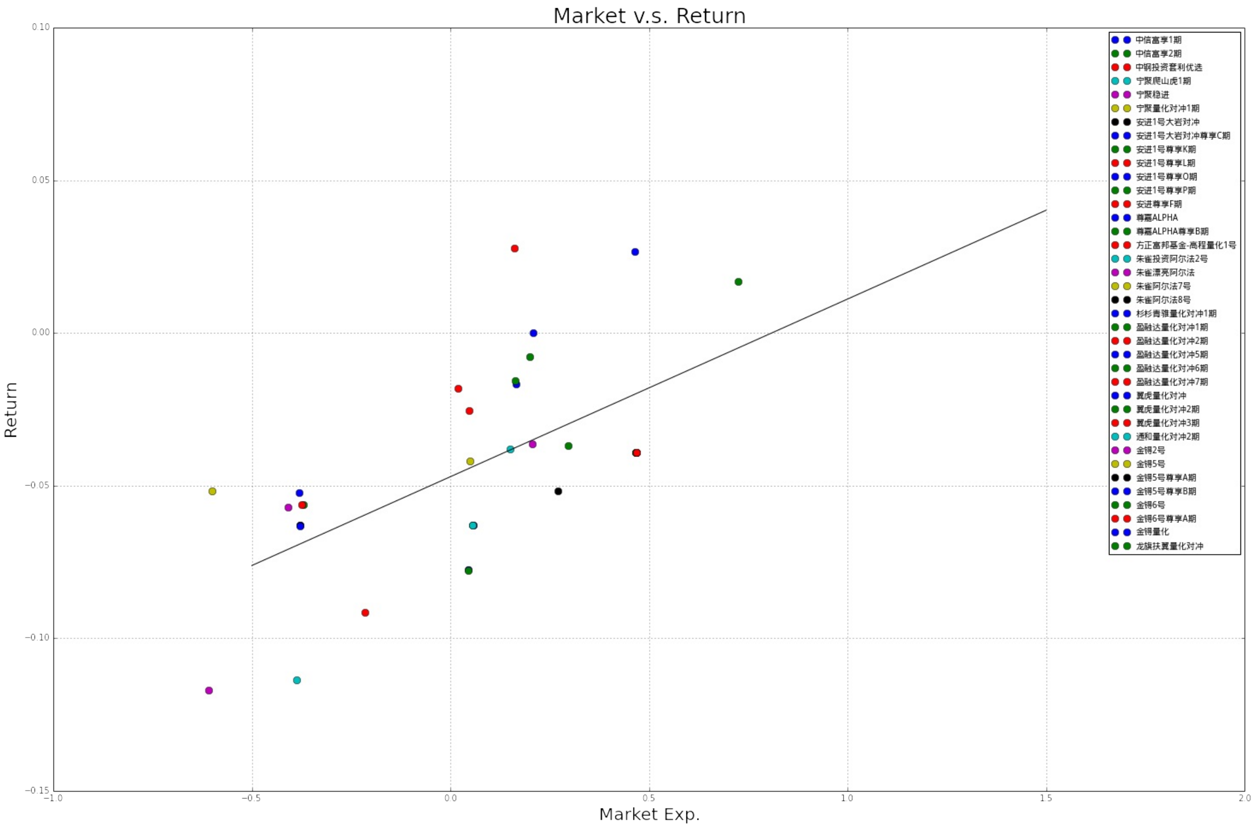
3. 风格收益分析

我们用上节得到的因子权重，与2014年12月初的那一周收益率进行对比。为了更加的一目了然，我们分别按照“市场 v.s. 收益”、“规模 v.s. 收益”、“价值 v.s. 收益”三个维度进行分析。通过散点图，很清楚的显示，市场因子在这一周对于alpha基金的收益的贡献是正向反馈效应；相反的，规模以及价值因子对于alpha基金的收益是负反馈。

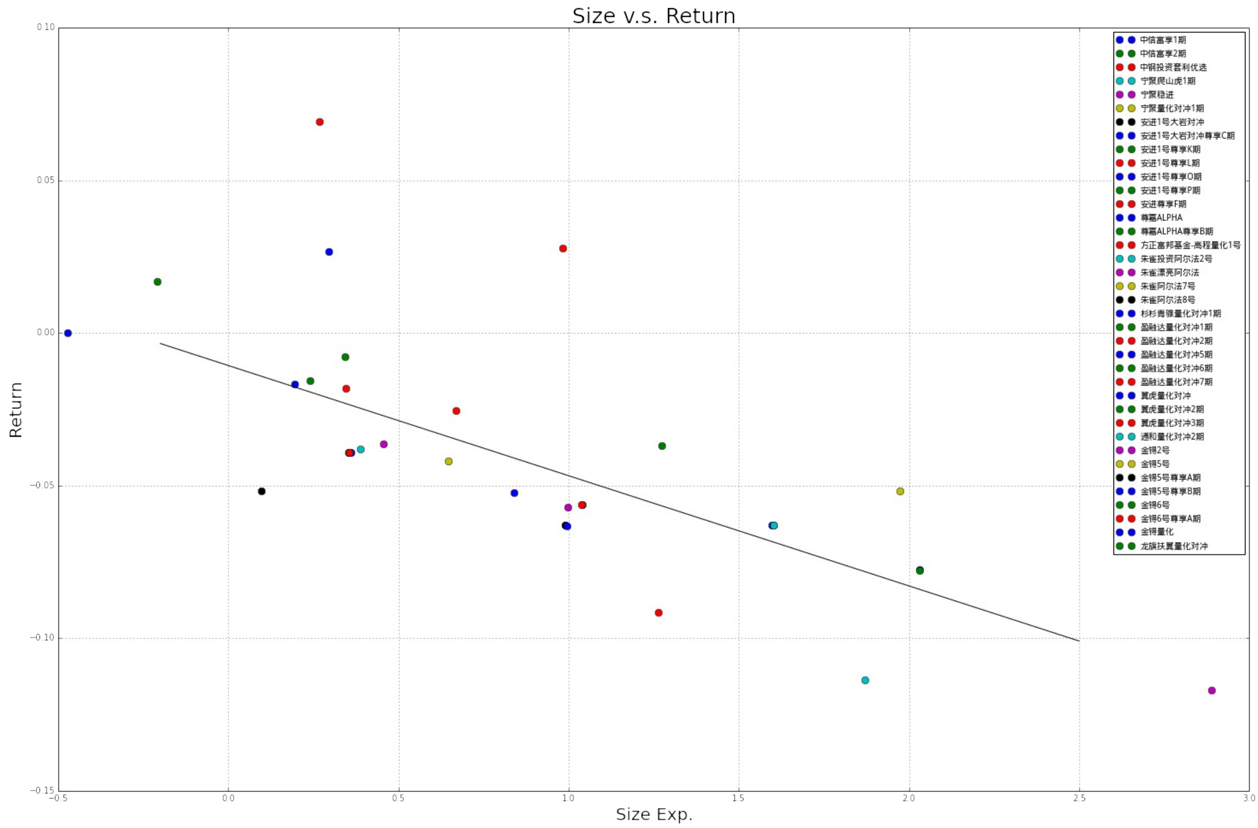
```
def func(beta, alpha):

    def inner(x):
        return beta*x + alpha

    return inner
```

```
groups = regression.groupby('Name')
fig, ax = plt.subplots(figsize = (25,16))
for name, group in groups:
    ax.plot(group.Market, group.Return, marker='o', linestyle=''
, ms=8, label=name)
    ax.grid(True)
ax.legend(prop = font)
ax.set_xlabel('Market Exp.', fontsize = 20)
ax.set_ylabel('Return', fontsize = 20)
ax.set_title('Market v.s. Return', fontsize = 25)

clf = linear_model.LinearRegression()
x = regression[['Market']]
y = regression['Return']
clf.fit(x,y)
beta = clf.coef_[0]
alpha = clf.intercept_

applyFunc = func(beta, alpha)
x = np.linspace( -0.5, 1.5, 100)
y = [applyFunc(v) for v in x]

plt.plot(x,y ,'k-')
```

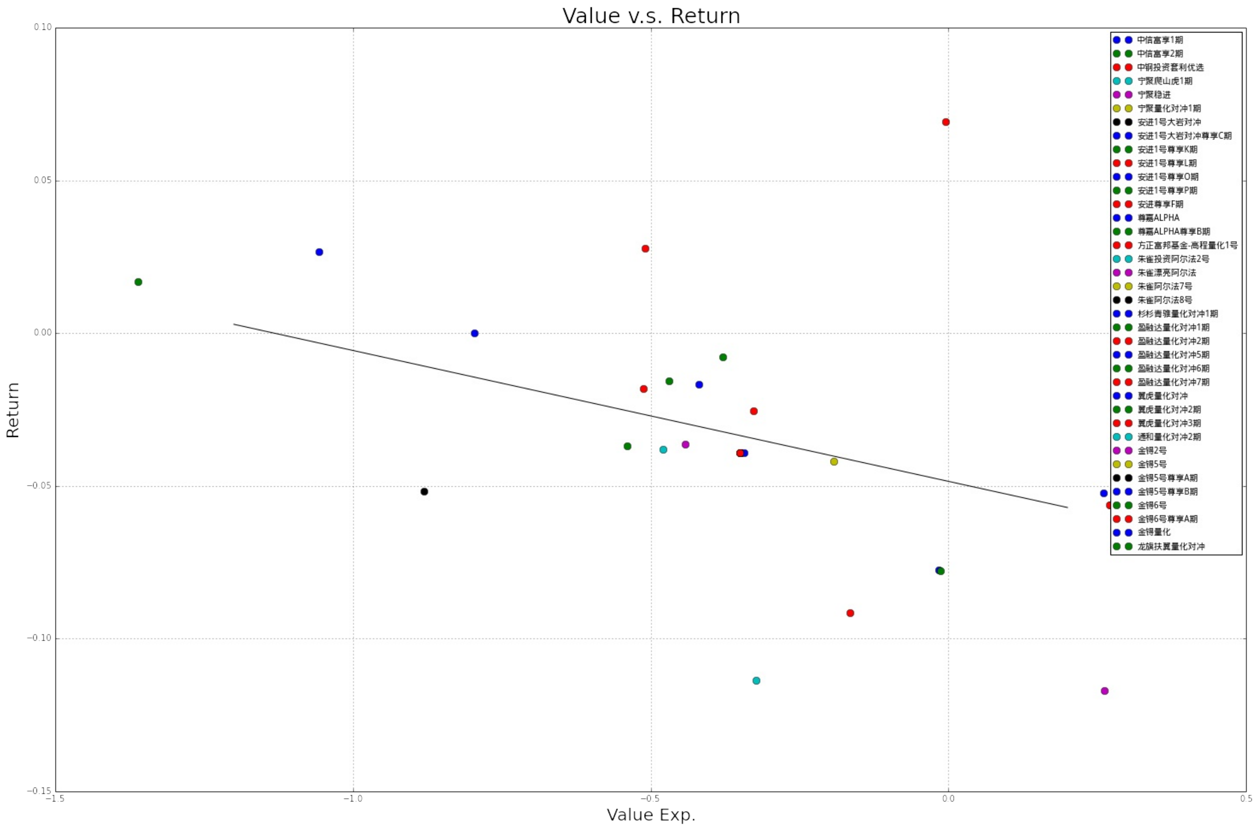
```
[<matplotlib.lines.Line2D at 0x6477550>]
```



```
groups = regression.groupby('Name')
fig, ax = plt.subplots(figsize = (25,16))
for name, group in groups:
    ax.plot(group.Size, group.Return, marker='o', linestyle='',
ms=8, label=name)
    ax.grid(True)
ax.legend(prop = font)
ax.set_xlabel('Size Exp.', fontsize = 20)
ax.set_ylabel('Return', fontsize = 20)
ax.set_title('Size v.s. Return', fontsize = 25)

clf = linear_model.LinearRegression()
x = regression[['Size']]
y = regression['Return']
clf.fit(x,y)
beta = clf.coef_[0]
alpha = clf.intercept_

applyFunc = func(beta, alpha)
x = np.linspace( -0.2, 2.5, 100)
y = [applyFunc(v) for v in x]

plt.plot(x,y ,'k-')

[<matplotlib.lines.Line2D at 0x70e9810>]
```



```
groups = regression.groupby('Name')
fig, ax = plt.subplots(figsize = (25,16))
for name, group in groups:
    ax.plot(group.Value, group.Return, marker='o', linestyle='',
 ms=8, label=name)
    ax.grid(True)
ax.legend(prop = font)
ax.set_xlabel('Value Exp.', fontsize = 20)
ax.set_ylabel('Return', fontsize = 20)
ax.set_title('Value v.s. Return', fontsize = 25)

clf = linear_model.LinearRegression()
x = regression[['Value']]
y = regression['Return']
clf.fit(x,y)
beta = clf.coef_[0]
alpha = clf.intercept_

applyFunc = func(beta, alpha)
x = np.linspace( -1.2, 0.2, 100)
y = [applyFunc(v) for v in x]

plt.plot(x,y ,'k-')

[<matplotlib.lines.Line2D at 0x7d43410>]
```



4. “黑天鹅”的原因

让我们再仔细看一下之前的各家基金的风格权重。

市场因子

我们可以看到所有的4个收益为正的基金都在市场权重最高的50%以内。并且市场因子最大的两个基金恰好都是收益为正的。

```
regression.sort(columns = ['Market'], ascending = False)[:19]
```

	Name	Return	Market	Size	Value	Intercept
方正富邦基金-高程量化1号	方正富邦基金-高程量化1号	0.069128	1.911136	0.266616	-0.004385	-0.002195
翼虎量化对冲2期	翼虎量化对冲2期	0.016722	0.724630	-0.209773	-1.360643	0.001874
安进1号尊享L期	安进1号尊享L期	-0.039252	0.468794	0.355543	-0.349783	0.001756
安进尊享	安进尊享F	-0.039252	0.468560	0.355400	-0.349973	0.001757

F期						
安进1号尊享K期	安进1号尊享K期	-0.039252	0.468560	0.355400	-0.349973	0.001757
安进1号大岩对冲	安进1号大岩对冲	-0.039234	0.468212	0.355625	-0.348703	0.001755
安进1号大岩对冲尊享C期	安进1号大岩对冲尊享C期	-0.039234	0.468212	0.355625	-0.348703	0.001755
安进1号尊享P期	安进1号尊享P期	-0.039234	0.467421	0.353230	-0.349906	0.001765
安进1号尊享O	安进1号尊享O期	-0.039159	0.467231	0.360231	-0.343805	0.001745

O期						
翼虎量化对冲	翼虎量化对冲	0.026735	0.463962	0.294367	-1.057744	0.002474
龙旗扶翼量化对冲	龙旗扶翼量化对冲	-0.036871	0.296329	1.274064	-0.539870	0.001011
朱雀阿尔法8号	朱雀阿尔法8号	-0.051708	0.270032	0.097040	-0.881424	0.002036
杉杉青骓量化对冲1期	杉杉青骓量化对冲1期	-0.000085	0.208083	-0.472288	-0.796191	0.008189
朱雀漂亮阿尔法	朱雀漂亮阿尔法	-0.036283	0.205736	0.455439	-0.441430	0.001541
盈融达量	盈融达量					

对冲1期	化对冲1期					
盈融达量化对冲5期	盈融达量化对冲5期	-0.016643	0.164744	0.195826	-0.418823	0.002362
盈融达量化对冲6期	盈融达量化对冲6期	-0.015752	0.162680	0.240728	-0.468615	0.003681
翼虎量化对冲3期	翼虎量化对冲3期	0.027691	0.160118	0.981666	-0.509016	0.001601
朱雀投资阿尔法2号	朱雀投资阿尔法2号	-0.037950	0.149797	0.388702	-0.478786	0.001898

规模

我们可以看到3个收益为正的基金在规模权重最低的50%以内。而且这3个基金的规模权重都在最低的前10名以内。

```
regression.sort(columns = ['Size'], ascending = True)[:19]
```

	Name	Return	Market	Size	Value	Intercept
杉杉青骓量化对冲1期	杉杉青骓量化对冲1期	-0.000085	0.208083	-0.472288	-0.796191	0.008189
翼虎量化对冲2期	翼虎量化对冲2期	0.016722	0.724630	-0.209773	-1.360643	0.001874
朱雀阿尔法8号	朱雀阿尔法8号	-0.051708	0.270032	0.097040	-0.881424	0.002036
盈融达量化对冲5期	盈融达量化对冲5期	-0.016643	0.164744	0.195826	-0.418823	0.002362
盈融达量化对	盈融达量化对	-0.015752	0.162680	0.240728	-0.468615	0.003681

冲6期						
方正富邦基金-高程量化1号	方正富邦基金-高程量化1号	0.069128	1.911136	0.266616	-0.004385	-0.002195
翼虎量化对冲	翼虎量化对冲	0.026735	0.463962	0.294367	-1.057744	0.002474
盈融达量化对冲1期	盈融达量化对冲1期	-0.007880	0.199913	0.341597	-0.379266	0.004520
盈融达量化对冲2期	盈融达量化对冲2期	-0.018233	0.018288	0.344163	-0.512896	0.005633
安进1号尊享	安进1号尊享P期	-0.039234	0.467421	0.353230	-0.349906	0.001765

享P期	享P期					
安进1号尊享K期	安进1号尊享K期	-0.039252	0.468560	0.355400	-0.349973	0.001757
安进尊享F期	安进尊享F期	-0.039252	0.468560	0.355400	-0.349973	0.001757
安进1号尊享L期	安进1号尊享L期	-0.039252	0.468794	0.355543	-0.349783	0.001756
安进1号大岩对冲	安进1号大岩对冲	-0.039234	0.468212	0.355625	-0.348703	0.001755
安进1号大岩对冲尊享C	安进1号大岩对冲尊享C期	-0.039234	0.468212	0.355625	-0.348703	0.001755

安进1号尊享O期	安进1号尊享O期	-0.039159	0.467231	0.360231	-0.343805	0.001745
朱雀投资阿尔法2号	朱雀投资阿尔法2号	-0.037950	0.149797	0.388702	-0.478786	0.001898
朱雀漂亮阿尔法	朱雀漂亮阿尔法	-0.036283	0.205736	0.455439	-0.441430	0.001541
朱雀阿尔法7号	朱雀阿尔法7号	-0.041888	0.049495	0.645693	-0.192636	0.001548

价值

我们可以看到3个收益为正的基金在价值权重最低的50%以内。而且这3个基金的价值权重都在最低的前10名以内。特别的，价值权重最低的两个基金恰好都为正收益。

```
regression.sort(columns = ['Value'], ascending = True)[:19]
```

	Name	Return	Market	Size	Value	Intercept
翼虎量	翼虎					

量化对冲2期	翼虎量化对冲2期	0.016722	0.724630	-0.209773	-1.360643	0.001874
翼虎量化对冲	翼虎量化对冲	0.026735	0.463962	0.294367	-1.057744	0.002474
朱雀阿尔法8号	朱雀阿尔法8号	-0.051708	0.270032	0.097040	-0.881424	0.002036
杉杉青骓量化对冲1期	杉杉青骓量化对冲1期	-0.000085	0.208083	-0.472288	-0.796191	0.008189
龙旗扶翼量化对冲	龙旗扶翼量化对冲	-0.036871	0.296329	1.274064	-0.539870	0.001011
盈融达量化对冲2	盈融达量化对冲2期	-0.018233	0.018288	0.344163	-0.512896	0.005633

翼虎量化对冲3期	翼虎量化对冲3期	0.027691	0.160118	0.981666	-0.509016	0.001601
朱雀投资阿尔法2号	朱雀投资阿尔法2号	-0.037950	0.149797	0.388702	-0.478786	0.001898
盈融达量化对冲6期	盈融达量化对冲6期	-0.015752	0.162680	0.240728	-0.468615	0.003681
朱雀漂亮阿尔法	朱雀漂亮阿尔法	-0.036283	0.205736	0.455439	-0.441430	0.001541
盈融达量化对冲5期	盈融达量化对冲5期	-0.016643	0.164744	0.195826	-0.418823	0.002362
盈融						

融达量化对冲1期	盈融达量化对冲1期	-0.007880	0.199913	0.341597	-0.379266	0.004520
安进1号尊享K期	安进1号尊享K期	-0.039252	0.468560	0.355400	-0.349973	0.001757
安进尊享F期	安进尊享F期	-0.039252	0.468560	0.355400	-0.349973	0.001757
安进1号尊享P期	安进1号尊享P期	-0.039234	0.467421	0.353230	-0.349906	0.001765
安进1号尊享L期	安进1号尊享L期	-0.039252	0.468794	0.355543	-0.349783	0.001756
安进1号大岩对	安进1号大岩对冲	-0.039234	0.468212	0.355625	-0.348703	0.001755

安进1号大岩对冲尊享C期	安进1号大岩对冲尊享C期	-0.039234	0.468212	0.355625	-0.348703	0.001755
安进1号尊享O期	安进1号尊享O期	-0.039159	0.467231	0.360231	-0.343805	0.001745

三 债券

债券报价中的小陷阱

> 来源：https://uqer.io/community/share/55361dc2f9f06c392662104e

投资者习惯于使用到期收益率作为衡量债券投资价值的标杆，倾向于买入收益率高的债券，卖出收益率低的债券。这里有一个隐含的假设，所有债券的到期收益率都是由同一算法计算而得。但是事实上，真的是这样吗？

上图是在2015年4月21日截取的中债登的实时行情。我们取其中一个债券做示例：

- 代码：080014
- 净价：100.363
- 全价：103.214
- 票息：4.23%
- 到期收益率：3.019%

这个债券是2015年8月18日到期，还有一次付息。存续期不超过1年，为119天，0.326027年

1. 债券的例子

```
testBond = BuildBond('080014.XIBE')
testBond.bondProfile()
```

080014.XIBE	
securityID	080014.XIBE
issuer	财政部
issueDate	2008-08-18
exchange	XIBE
shortName	08国债14
maturity	7Y
startDate	2008-08-18
maturityDate	2015-08-18
settlementDays	1
coupon	0.0423
frequency	1
dayCounter	Actual/Actual (ISMA)

```
cleanPrice = 100.363
settlementDate = Date(2015,4,21)
```

2. 简单利率算法

```
print u'到期收益率：%.4f' % (testBond.yieldFromCleanPrice(cleanPri
ce,'Actual/Actual (ISMA)', Compounding.Simple, Frequency.Annual,
 settlementDate)*100)
print u'应计利息  ：%.4f' % testBond.accruedAmount(Date(2015,4,21)
)

到期收益率：3.0196
应计利息  ：2.8509
```

3. 复利算法

```
print u'到期收益率：%.4f' % (testBond.yieldFromCleanPrice(cleanPrice,'Actual/Actual (ISMA)', Compounding.Compounded, Frequency.Annual, settlementDate)*100)
print u'应计利息   ：%.4f' % testBond.accruedAmount(settlementDate)
```

```
到期收益率：3.0504
应计利息   ：2.8509
```

4. 总结

现阶段大多数行情软件的报价也都是依照中债登类似的做法。存续期少于1年的债券使用简单利率算法，大于1年的债券使用复利算法。同样的债券价格，使用不同的算法，获得的到期收益率会有些微的差异。这个差异在某个债券即将到期是特别明显（可能差几十个bp甚至上一个百分点）。所以投资者在使用到期收益率作为债券投资价值评估标准的时候，要注意这些差异。

四 利率互换

Swap Curve Construction

> 来源：https://uqer.io/community/share/55c2d440f9f06c91fc18c648

在这个示例中，我们将指导用户如何使用平台的功能，完成从利率互换的市场报价完成收益率曲线的构造。

```
from CAL.PyCAL import *
SetEvaluationDate(Date(2015, 8, 6))
```

1. 构造收益率曲线

我们从一组市场标准化互换的市场报价中获取收益率曲线的信息：

- `swap_rates` ：标准互换对应的固定端利率
- `swap_tenor` ：标准互换对应的期限

```
swap_rates = [0.02, 0.03, 0.04 ,0.05, 0.055, 0.06, 0.065, 0.07]
swap_tenor = ['6M', '1Y', '2Y', '3Y', '4Y', '5Y',  '7Y', '10Y']
shiborIndex = Shibor('3M')

instruments = []
for rate, tenor in zip(swap_rates, swap_tenor):
    print('{0:3s} benchmark Shibor Swap fixed at: {1:.2f}%'.form
at(tenor, rate*100))
    rateHelper = ShiborSwapRateHelper(rate, Period(tenor), Frequ
ency.Quarterly, shiborIndex)
    instruments.append(rateHelper)

6M  benchmark Shibor Swap fixed at: 2.00%
1Y  benchmark Shibor Swap fixed at: 3.00%
2Y  benchmark Shibor Swap fixed at: 4.00%
3Y  benchmark Shibor Swap fixed at: 5.00%
4Y  benchmark Shibor Swap fixed at: 5.50%
5Y  benchmark Shibor Swap fixed at: 6.00%
7Y  benchmark Shibor Swap fixed at: 6.50%
10Y benchmark Shibor Swap fixed at: 7.00%
```

通过标准互换校正（calibration）收益率曲线：

```
calibratedCurve = CalibratedYieldCurve(EvaluationDate(), instrum
ents, 'Actual/365 (Fixed)')
```

收益率曲线的基本信息：

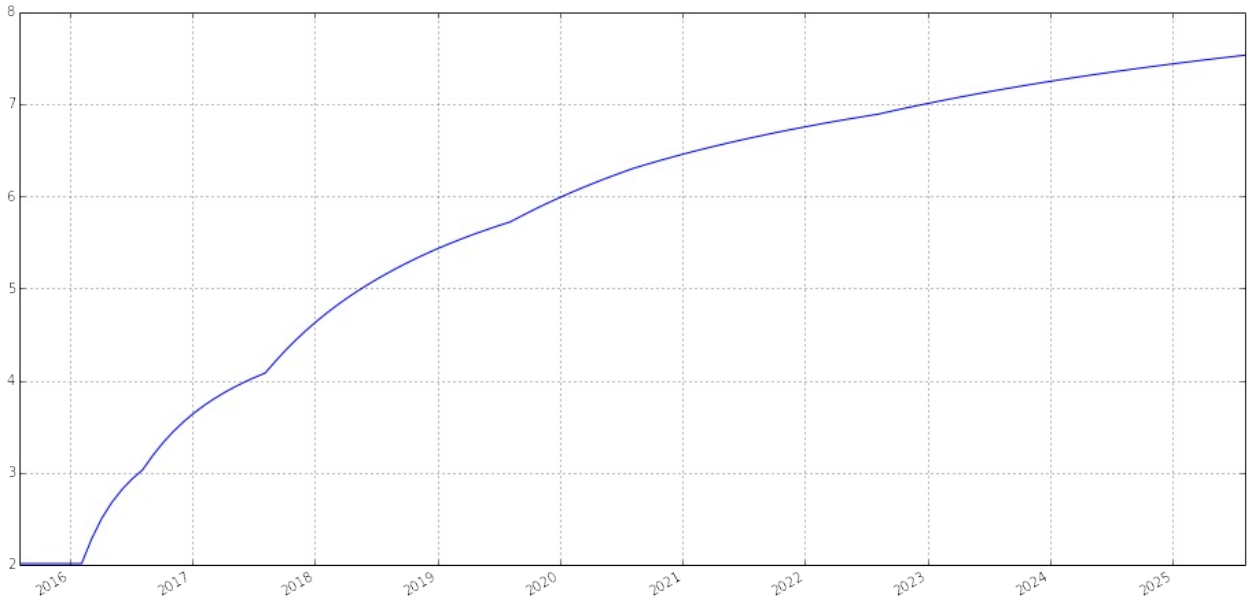
- `discount` ：折现因子
- `forward(%)` ：远期利率
- `zero(%)` ：零息利率

```
calibratedCurve.curveProfile().head(10)
```

	date	discount	forward(%)	zero(%)
2015-08-06	2015-08-06	1.000000	1.994947	2.014979
2015-09-06	2015-09-06	0.998307	1.994947	2.014979
2015-10-06	2015-10-06	0.996672	1.994947	2.014979
2015-11-06	2015-11-06	0.994984	1.994947	2.014979
2015-12-06	2015-12-06	0.993354	1.994947	2.014979
2016-01-06	2016-01-06	0.991672	1.994947	2.014979
2016-02-06	2016-02-06	0.989994	1.994947	2.014979
2016-03-06	2016-03-06	0.986950	4.014304	2.276446
2016-04-06	2016-04-06	0.983591	4.014304	2.505840
2016-05-06	2016-05-06	0.980351	4.014304	2.678750

我们可以画图来看：

```
calibratedCurve.curveProfile()['zero(%)'].plot(figsize=(16,8))

<matplotlib.axes.AxesSubplot at 0x6bbabd0>
```



2. 测试

首先可以看这条收益率曲线是否真的可以完美定价基准互换（perfectly pricing）：

```
cal = Calendar('China.IB')
startDate = cal.advanceDate(Date(2015, 8, 6), '1B', BizDayConven
tion.Following)
shiborIndex = Shibor('3M', calibratedCurve)
nominal = 100000000.
pricingEngine = DiscountingSwapEngine(calibratedCurve)
for rate, tenor in zip(swap_rates, swap_tenor):
    benchmarkSwap = ShiborSwap(SwapLegType.Payer, nominal, start
Date, Period(tenor), Period('3M'), rate, shiborIndex)
    benchmarkSwap.setPricingEngine(pricingEngine)
    print('{0:3s} benchmark Shibor Swap NPV: {1:>8.4f}'.format(t
enor, benchmarkSwap.NPV()))

6M  benchmark Shibor Swap NPV:   0.0000
1Y  benchmark Shibor Swap NPV:  -0.0000
2Y  benchmark Shibor Swap NPV:   0.0000
3Y  benchmark Shibor Swap NPV:   0.0000
4Y  benchmark Shibor Swap NPV:  -0.0000
5Y  benchmark Shibor Swap NPV:   0.0000
7Y  benchmark Shibor Swap NPV:   0.0000
10Y benchmark Shibor Swap NPV:   0.0000
```

然后我们取一个假设已经存在的互换（seasoned swap），通过这条收益率曲线估计它的现值：

```
startDate = Date(2015, 7, 15)
shiborIndex.addFixing(Date(2015, 7, 14), 0.045)
customizeSwap = ShiborSwap(SwapLegType.Receiver, nominal, startD
ate, Period('9Y'), Period('3M'), 0.06, shiborIndex)
customizeSwap.setPricingEngine(pricingEngine)
print('{0:3s} Shibor Swap fixed at {1:.2f}% NPV: {2:15.4f}'.form
at('9Y',6.00, customizeSwap.NPV()))

9Y  Shibor Swap fixed at 6.00% NPV:   -6308510.5573
```

```
customizeSwap.legAnalysis(0).head(10)
```

	AMOUNT	NOMINAL	ACCRUAL_START_DATE	A
PAYMENT_DATE				
2015-10-15	1512329	1e+08	2015-07-15	20
2016-01-15	1512329	1e+08	2015-10-15	20
2016-04-15	1495890	1e+08	2016-01-15	20
2016-07-15	1495890	1e+08	2016-04-15	20
2016-10-17	1545205	1e+08	2016-07-15	20
2017-01-16	1495890	1e+08	2016-10-17	20
2017-04-17	1495890	1e+08	2017-01-16	20
2017-07-17	1495890	1e+08	2017-04-17	20
2017-10-16	1495890	1e+08	2017-07-17	20
2018-01-15	1495890	1e+08	2017-10-16	20

```
customizeSwap.legAnalysis(1).head(10)
```

	AMOUNT	NOMINAL	ACCRUAL_START_DATE	A
PAYMENT_DATE				
2015-10-15	1150000	1e+08	2015-07-15	20
2016-01-15	504102.3	1e+08	2015-10-15	20
2016-04-15	871825.4	1e+08	2016-01-15	20
2016-07-15	1005852	1e+08	2016-04-15	20
2016-10-17	1234864	1e+08	2016-07-15	20
2017-01-16	1260227	1e+08	2016-10-17	20
2017-04-17	1260227	1e+08	2017-01-16	20
2017-07-17	1260227	1e+08	2017-04-17	20
2017-10-16	1668056	1e+08	2017-07-17	20
2018-01-15	1790725	1e+08	2017-10-16	20

中国 Repo 7D 互换的例子

> 来源：https://uqer.io/community/share/55c177cbf9f06c915418c641

下面的例子给出在量化实验室中如何为一个Repo 7D互换定价的例子

- `swapType` ：互换类型， `Payer` 代表付固定端利息，收浮动端利息；
- `nominal` ：互换面值
- `startDate` ：互换生效日
- `swapTenor` ：互换期限
- `paymentTenor` ：付息周期
- `fixedRate` ：固定端利息
- `rateSpread` ：浮动端息差
- `repoIndex` ：浮动端指数

这里我们使用一条平坦的收益率曲线作为远期曲线：

```
forwardingCurve = FlatForward(Date(2015, 8, 4), 0.05, 'Actual/36
0')
```

```
from CAL.PyCAL import *

SetEvaluationDate = Date(2015, 8, 4)

swapType = SwapLegType.Payer
nominal = 100000000.
startDate = Date(2015, 8, 7)
swapTenor = Period('10Y')
paymentTenor = Period('3M')
fixedRate = 0.055
rateSpread = 0.0
forwardingCurve = FlatForward(Date(2015, 8, 4), 0.05, 'Actual/36
0')
repoIndex = RepoChina('7D', yieldCurve)
```

组装成我们需要的 `RepoCompoundingSwap` ：

```
swap = RepoCompoundingSwap(swapType=swapType,
                           nominal=nominal,
                           startDate=startDate,
                           swapTenor=swapTenor,
                           paymentTenor=paymentTenor,
                           fixedRate=fixedRate,
                           rateSpread=rateSpread,
                           repoIndex=repoIndex)
```

继续的，为了计算 `swap` 的现值，我们需要定义 `DiscountingSwapEngine` 对象，这里我们同样使用一条平坦的收益率曲线：

```
discountingCurve = FlatForward(Date(2015, 8, 4), 0.065, 'Actual/
360')
pricingEngine = DiscountingSwapEngine(discountingCurve)

swap.setPricingEngine(pricingEngine)

print("NPV: {0:.4f}".format(swap.NPV()))
print("Fair rate: {0:.4f}".format(swap.fairRate()))

NPV: -2282521.8872
Fair rate: 0.0519
```

下面的是 `swap` 每条 `leg` 的具体现金流分析：在 `legAnalysis` 接受的参数中，0 代表固定端，1代表浮动端。

```
swap.legAnalysis(0).tail()
```

	AMOUNT	NOMINAL	ACCRUAL_START_DATE	A
PAYMENT_DATE				
2024-08-07	1386301	1e+08	2024-05-07	20
2024-11-07	1386301	1e+08	2024-08-07	20
2025-02-07	1386301	1e+08	2024-11-07	20
2025-05-07	1341096	1e+08	2025-02-07	20
2025-08-07	1386301	1e+08	2025-05-07	20

```
swap.legAnalysis(1).tail()
```

	AMOUNT	NOMINAL	ACCRUAL_START_DATE	A
PAYMENT_DATE				
2024-08-07	1306927	1e+08	2024-05-07	20
2024-11-07	1306927	1e+08	2024-08-07	20
2025-02-07	1309570	1e+08	2024-11-07	20
2025-05-07	1265808	1e+08	2025-02-07	20
2025-08-07	1306927	1e+08	2025-05-07	20

第四部分 衍生品相关

一 期权数据

如何获取期权市场数据快照

> 来源：https://uqer.io/community/share/550274e4f9f06c7a9ae9a535

在本文中，我们将通过实际的市场的例子，展示如何在量化实验室中计算和展示期权的隐含波动率微笑。

```
import pandas as pd
from matplotlib import pylab
pd.options.display.float_format = '{:,>.4f}'.format
```

1. 获取市场数据

在本节中，我们使用数据API获取数据，并进行一些必要的数据转换。这里我们获取的是实时报价，是本 notebook 运行时的市场快照。

- `dataDate` 交易日
- `dataTime` 快照时间戳
- `optionId` 期权代码
- `instrumentID` 期权交易代码
- `contractType` 期权类型，CO为看着，PO为看跌
- `strikePrice` 行权价
- `expDate` 到期日
- `lastPrice` 最新价

```
optionSnapShot = OptionsDataSnapShot()
optionSnapShot[optionSnapShot.expDate == Date(2015,9,23)]
```

	dataDate	dataTime	optionId	instrumentID	contract
30	2015-03-13	13:24:12	10000031	510050C1509M02200	CO
31	2015-03-13	13:24:17	10000032	510050C1509M02250	CO
32	2015-03-13	13:24:22	10000033	510050C1509M02300	CO
33	2015-03-13	13:24:27	10000034	510050C1509M02350	CO
34	2015-03-13	13:24:32	10000035	510050C1509M02400	CO
35	2015-03-13	13:24:36	10000036	510050P1509M02200	PO
36	2015-03-13	13:24:41	10000037	510050P1509M02250	PO
37	2015-03-13	13:24:47	10000038	510050P1509M02300	PO
38	2015-03-13	13:24:52	10000039	510050P1509M02350	PO
39	2015-03-13	13:24:58	10000040	510050P1509M02400	PO
46	2015-03-13	13:24:52	10000047	510050C1509M02450	CO
47	2015-03-13	13:24:58	10000048	510050P1509M02450	PO
54	2015-03-13	13:24:32	10000055	510050C1509M02500	CO
55	2015-03-13	13:24:36	10000056	510050P1509M02500	PO
62	2015-03-13	13:24:32	10000063	510050C1509M02550	CO
63	2015-03-13	13:24:36	10000064	510050P1509M02550	PO

2. 计算隐含波动率以及相关Greeks

接着我们可以方便的使用内置函数 BSMImpliedVolatity 计算期权的隐含波动率。

- `price` 市场报价或者模型价格
- `delta` 期权价格关于标的价格的一阶导数
- `gamma` 期权价格关于标的价格的二阶导数
- `rho` 期权价格关于无风险利率的一阶导数
- `theta` 期权价格关于到期时间的一阶导数（每日）
- `vega` 期权价格关于波动率的一阶导数

```
analyticResult = OptionsAnalyticResult()
analyticResult.loc[:10, ['optionId', 'contractType', 'strikePrice', 'expDate', 'lastPrice', 'vol', 'delta', 'gamma', 'rho', 'theta', 'vega']]
```

	optionId	contractType	strikePrice	expDate	lastPrice	vo
1	10000002	CO	2.2500	March 25th, 2015	0.2184	0.22
2	10000003	CO	2.3000	March 25th, 2015	0.1730	0.28
3	10000004	CO	2.3500	March 25th, 2015	0.1229	0.21
4	10000005	CO	2.4000	March 25th, 2015	0.0814	0.21
8	10000009	PO	2.3500	March 25th, 2015	0.0076	0.24
9	10000010	PO	2.4000	March 25th, 2015	0.0159	0.23
10	10000011	CO	2.2000	April 22nd, 2015	0.2778	0.27

3. 构造波动率曲面

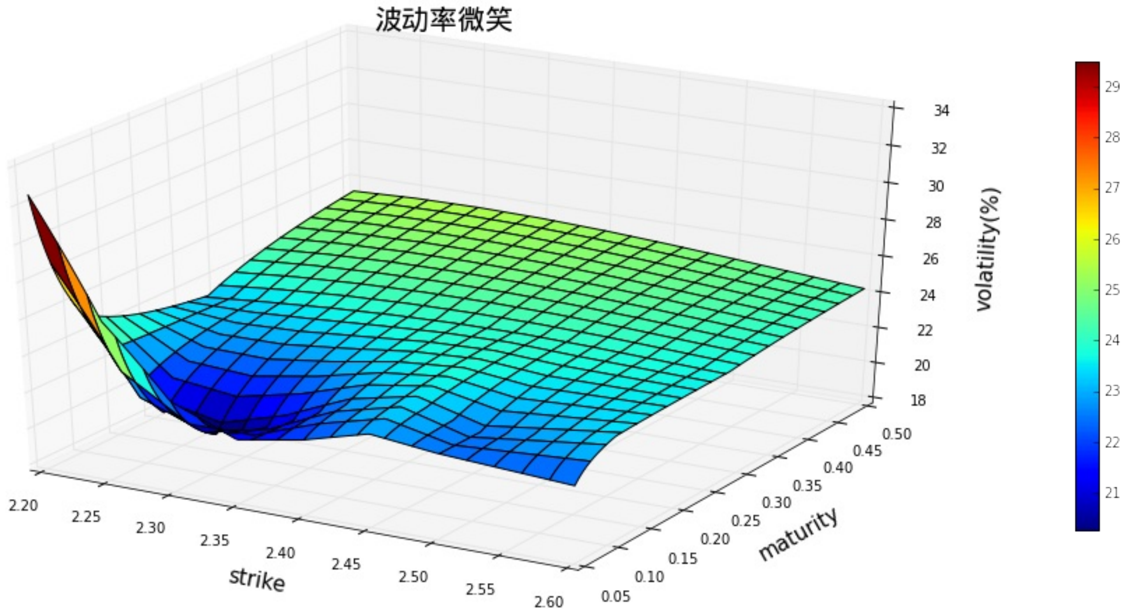
但是对于市场参与者而言，像刚才这样仅仅观察的线的结构不够。他们需要看到整个市场以到期时间，行权价为轴的波动率曲面（Volatility Surface）。除此之外，他们更想知道，波动率曲面上，那些并不是市场报价点的值，至少是个估计。这样的波动率曲面构造，往往需要依赖某种模型，或者某种插值方法。在这一节中，我们将介绍使用 CAL 中的波动率曲面构造函数。

以下的例子基于 CAL 函数: `VolatilitySurfaceSnapShot`

3.1 基于SABR模型的波动率曲面

```
volInterpolatorSABR = VolatilitySurfaceSnapShot(optionType = 'CALL', interpType = 'SABR')
volInterpolatorSABR.plotSurface(startStrike = 2.2,endStrike = 2.6)
volInterpolatorSABR.volalitltyProfileFromPeriods([2.2, 2.3, 2.4, 2.5, 2.6], ['1M', '2M', '3M', '6M', '9M'])
```

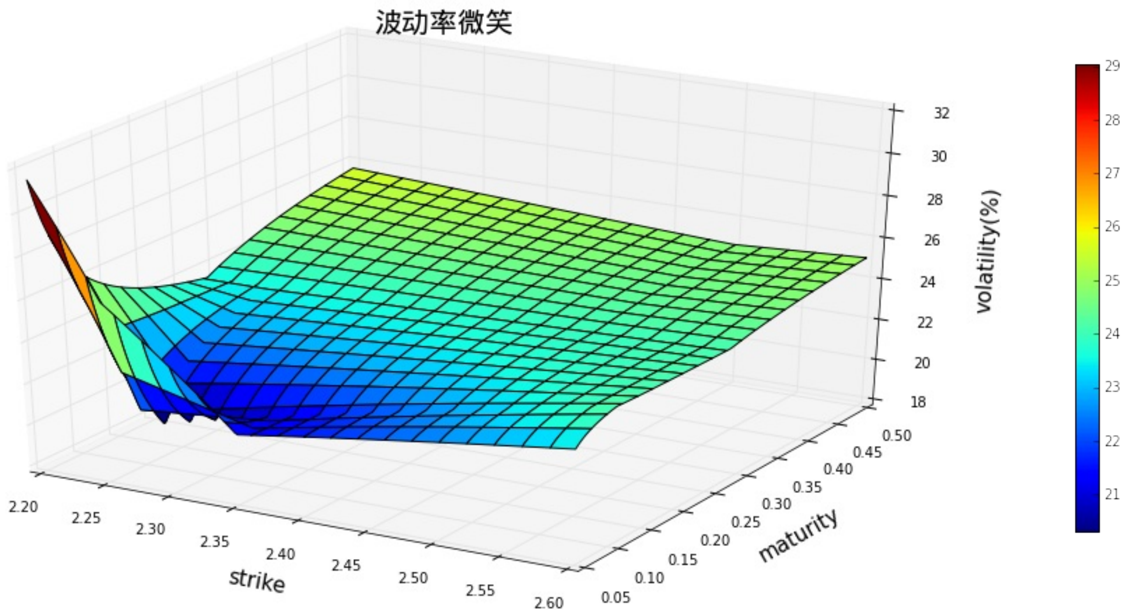
	1M	2M	3M	6M	9M
2.2000	0.2720	0.2406	0.2327	0.2531	0.2545
2.3000	0.2048	0.2207	0.2345	0.2546	0.2557
2.4000	0.2245	0.2341	0.2389	0.2525	0.2533
2.5000	0.2241	0.2328	0.2381	0.2479	0.2484
2.6000	0.2311	0.2356	0.2362	0.2425	0.2429



3.2 基于SVI模型的波动率曲面

```
volInterpolatorSVI = VolatilitySurfaceSnapShot(optionType = 'CALL', interpType = 'SVI')
volInterpolatorSVI.plotSurface(startStrike = 2.2,endStrike = 2.6)
volInterpolatorSVI.volalitltyProfileFromPeriods([2.2, 2.3, 2.4, 2.5, 2.6], ['1M', '2M', '3M', '6M', '9M'])
```

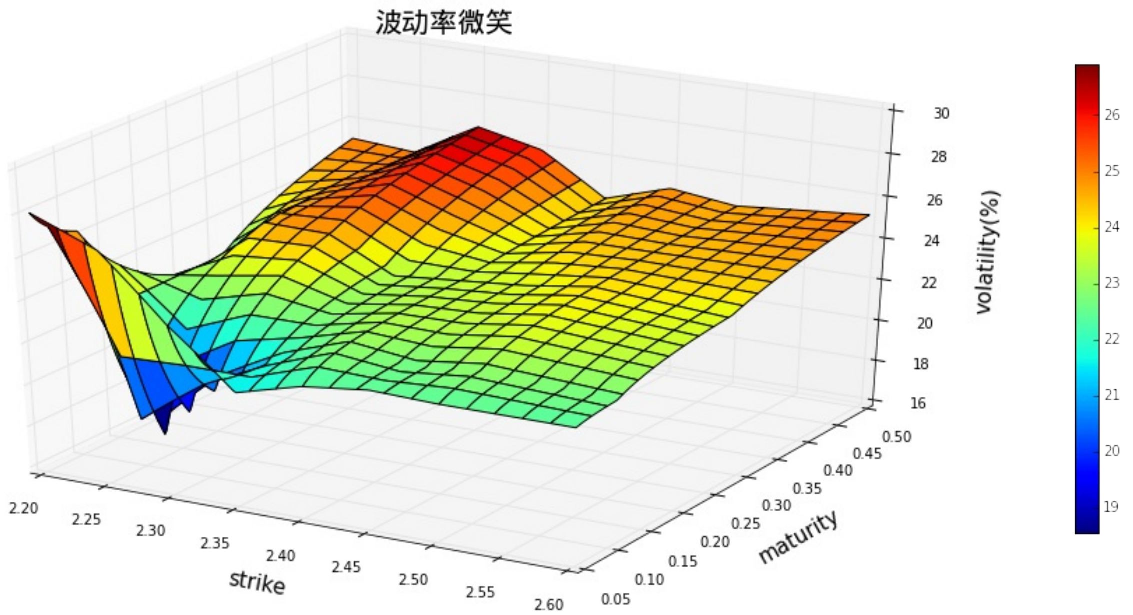
	1M	2M	3M	6M	9M
2.2000	0.2769	0.2476	0.2369	0.2566	0.2580
2.3000	0.2121	0.2223	0.2340	0.2535	0.2545
2.4000	0.2170	0.2292	0.2365	0.2504	0.2512
2.5000	0.2290	0.2357	0.2389	0.2474	0.2479
2.6000	0.2401	0.2417	0.2413	0.2508	0.2514



3.3 基于Balck波动率插值的波动率曲面

```
volInterpolatorVariance = VolatilitySurfaceSnapShot(optionType = 'CALL', interpType = 'BlackVariance')
volInterpolatorVariance.plotSurface(startStrike = 2.2,endStrike = 2.6)
volInterpolatorVariance.volalitltyProfileFromPeriods([2.2, 2.3, 2.4, 2.5, 2.6], ['1M', '2M', '3M', '6M', '9M'])
```

	1M	2M	3M	6M	9M
2.2000	0.2676	0.2380	0.2202	0.2516	0.2537
2.3000	0.2082	0.2270	0.2441	0.2660	0.2672
2.4000	0.2277	0.2325	0.2341	0.2404	0.2408
2.5000	0.2278	0.2363	0.2408	0.2463	0.2466
2.6000	0.2252	0.2324	0.2365	0.2517	0.2526



4. 组合计算

在本节中，我们假设客户已经拥有了自己的期权头寸，希望利用量化实验室的功能进行风险监控。我们假设有以下的期权头寸：

期权代码	数量	行权价（￥）	到期时间
10000004	-7000	2.35	2015-03-25
10000011	2000	2.20	2015-04-22
10000027	5000	2.25	2015-06-24
10000047	3000	2.45	2015-09-23

然后我们构造 `OptionBook` ：

```
optionIDs = ['10000011', '10000027', '10000004', '10000047']
amounts = [2000, 5000, -7000, 3000]
optBook = OptionBook(optionIDs, amounts)
print u'期权头寸：'
optBook.description()

期权头寸：
```

	dataDate	dataTime	optionId	instrumentID	contractT
0	2015-03-13	13:24:58	10000004	510050C1503M02350	CO
1	2015-03-13	13:24:32	10000011	510050C1504M02200	CO
2	2015-03-13	13:24:52	10000027	510050P1506M02250	PO
3	2015-03-13	13:24:52	10000047	510050C1509M02450	CO

4.1 使用Black插值模型计算组合风险

```
optBook.riskReport(volInterpolatorVariance)
```

	optionId	vol	price	delta	gamma	
0	10000004	0.2060	-860.3000	-6370.8417	-12316.8687	-
1	10000011	0.2634	555.6000	1828.2807	1456.7054	4
2	10000027	0.2484	220.5000	-1103.5286	4552.0335	-
3	10000047	0.2509	566.7000	1659.5347	2626.2983	1
portfolio	NaN	nan	482.5000	-3986.5549	-3681.8315	9

4.2 使用SABR模型组合风险

```
optBook.riskReport(volInterpolatorSABR)
```

	optionId	vol	price	delta	gamma	
0	10000004	0.2157	-865.4365	-6301.5462	-12703.7735	-
1	10000011	0.2523	552.8686	1845.2432	1405.9255	4
2	10000027	0.2347	194.1368	-1048.6009	4677.9921	-
3	10000047	0.2511	566.9933	1659.5667	2624.8517	1
portfolio	NaN	nan	448.5622	-3845.3372	-3995.0043	1

4.3 使用SVI模型组合风险

```
optBook.riskReport(volInterpolatorSVI)
```

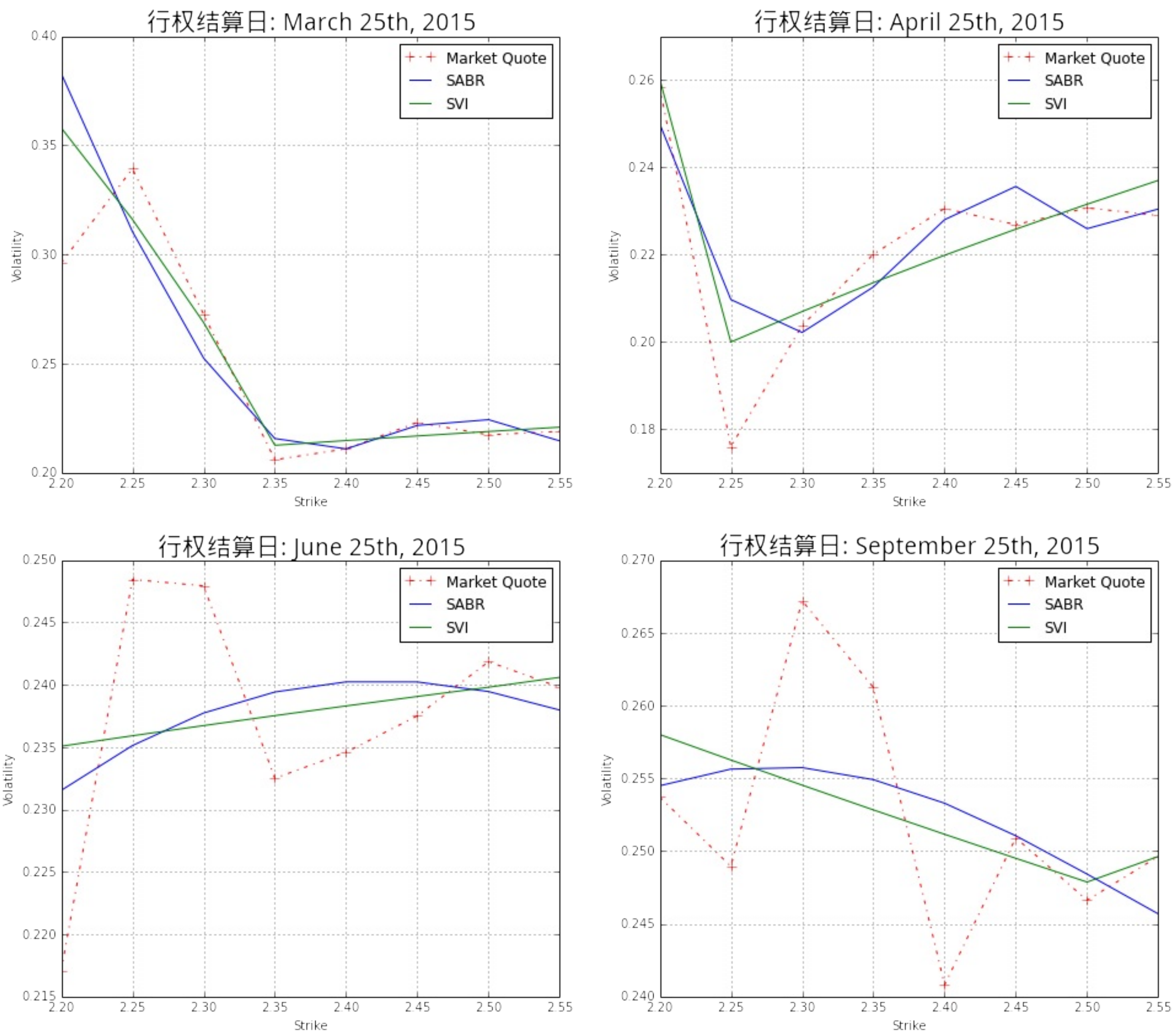
	optionId	vol	price	delta	gamma	
0	10000004	0.2126	-863.7639	-6323.4081	-12591.1718	-
1	10000011	0.2634	555.6000	1828.2807	1456.7054	4
2	10000027	0.2355	195.6318	-1051.9049	4670.9045	-
3	10000047	0.2495	563.6855	1659.2077	2641.2567	1
portfolio	NaN	nan	451.1534	-3887.8246	-3822.3052	1

5 比较不同模型的拟合市场数据的能力

这里我们比较不同的模型，对于市场数据的拟合能力。这里我们可以观察到单论你和能力 `BlackVarianceSurface > SviCalibratedVolSruface > SABRCalibratedVolS`。这里我们并不想下这样的结论：这些模型的优劣也有相同的排序。

另一个我们可以观察到的现象，对于近月合约（流动性最好），波动率微笑是最规则的。在这个期限上，三种模型的拟合都很到位。随着期限的上升，流动性的下降，买卖价差也随之扩大。这时候波动率微笑变得愈发不规则，这个时候一个完美拟合至市场的模型是否必要，是一个很大的问题：如果市场报价并不理性，一个优秀的模型应该可以指出这种不合理点，而不是简单的接受市场的非理性。

```
from matplotlib import pylab

strikes = sorted(analyticResult['strikePrice'].unique())
expiries = [Date(2015,3,25),Date(2015,4,25),Date(2015,6,25),Date(
2015,9,25)]
maturity = [(date - EvaluationDate())/ 365.0 for date in expirie
s]
volSurfaces = [volInterpolatorSABR, volInterpolatorSVI]

def plotModelFitting(index, volSurfaces, legends = ['Market Quot
e', 'SABR', 'SVI']):
    # Using Black variance surface to extrace the rar wolatility
    data = volInterpolatorVariance.volatility(strikes, maturity[
index], True)
    pylab.plot(strikes, data, 'r+-.',  markersize = 8)
    for s in volSurfaces:
        data = s.volatility(strikes, maturity[index], True)
        pylab.plot(strikes, data)
    pylab.xlabel('Strike')
    pylab.ylabel('Volatility')
    pylab.legend(legends, loc = 'best', fontsize = 12)
    pylab.title(u'行权结算日： ' + str(expiries[index]), fontproper
ties = font, fontsize = 20)
    pylab.grid(True)

pylab.subplots(2,2, figsize = (16,14))
for i in range(1,5):
    pylab.subplot('22' + str(i))
    plotModelFitting(i-1, volSurfaces)
```


行权结算日: March 25th, 2015
行权结算日: April 25th, 2015
行权结算日: June 25th, 2015
行权结算日: September 25th, 2015
Market Quote
SABR
SVI
Volatility
Strike
0.40
0.35
0.30
0.25
0.20
0.26
0.24
0.22
0.18
0.250
0.245
0.240
0.235
0.230
0.225
0.220
0.215
0.270
0.265
0.260
0.255
0.250
0.245
0.240
2.20
2.25
2.30
2.35
2.40
2.45
2.50
2.55

期权高频数据准备

> 来源：https://uqer.io/community/share/55027e68f9f06c7a9ae9a53b

本notebook根据指定的时间区间整理并保存 `option_data.csv` 文件，请与 期权市场一周纵览 notebook配合使用。

```
import pandas as pd
import numpy as np
pd.options.display.float_format = '{:,>.4f}'.format
```

```
calendar = Calendar('China.SSE')
class _format_checker:

    def __init__(self, calendar):
        self.calendar = calendar

    def _format_check(self, instrumentID):
        contractType = instrumentID[6] + 'O'
        contractYear = int(instrumentID[7:9]) + 2000
        contractMonth = int(instrumentID[9:11])
        contractExp = Date.NthWeekDay(4, Wednesday, contractMonth, contractYear)
        contractExp = self.calendar.adjustDate(contractExp, BizDayConvention.Following)
        contractStrike = float(instrumentID[-4:]) / 1000.0
        return contractType, contractExp, contractStrike

checker = _format_checker(calendar)
```

```
tradingDays = calendar.bizDatesList(Date(2015,3,5), Date(2015,3,12))
names, instrumentIDs = (OptionsDataSnapShot().optionId.unique(), OptionsDataSnapShot().instrumentID.unique())
data = pd.DataFrame(names, columns = ['optionId'])
instrumentIDs = pd.Series(instrumentIDs)
data = data.join(pd.DataFrame(list(instrumentIDs.apply(checker._format_check)), columns= ['contractType', 'expDate', 'strikePrice']))
data[:5]
```

	optionId	contractType	expDate	strikePrice
0	10000001	CO	March 25th, 2015	2.2000
1	10000002	CO	March 25th, 2015	2.2500
2	10000003	CO	March 25th, 2015	2.3000
3	10000004	CO	March 25th, 2015	2.3500
4	10000005	CO	March 25th, 2015	2.4000

```
tradingDaysStr = [''.join(date.toISO().split('-')) for date in t
radingDays]
tradingDaysStr

['20150305', '20150306', '20150309', '20150310', '20150311']
```

```
res = pd.DataFrame()
spotData = []
for day in tradingDaysStr:
    tmp = spotData
    try:
        spotData = DataAPI.MktTicksHistOneDayGet('510050.XSHG',
date = day, field = ['dataDate', 'datasTime', 'secOffset', 'last
Price'])
        spotData = spotData.drop(0)
    except Exception, e:
        print e
        spotData = tmp
    for opt in names:
        try:
            sample = DataAPI.MktOptionTicksHistOneDayGet(optionI
d = opt,date = day)#field = ['optionId', 'dataDate', 'dataTime'
'secOffset', 'lastPrice'])
            sample = sample.drop_duplicates(['secOffset'])
            spotPrice = np.zeros((len(sample),))
            j = 0
            index = spotData.index
            for i, secOffset in enumerate(sample.secOffset):
                currentSpotSecOffset = spotData.loc[index[j], 's
ecOffset']*1000
                while currentSpotSecOffset < secOffset and j < l
en(index)-1:
                    j = j + 1
                    currentSpotSecOffset = spotData.loc[index[j]
, 'secOffset']*1000
                if j>=1:
                    spotPrice[i] =  spotData.loc[index[j-1], 'la
stPrice']
                else:
```

```
          spotPrice[i] = spotData.loc[index[j], 'lastP
rice']
      sample['spotPrice'] = spotPrice
      res = res.append(sample)
    except Exception, e:
      print e
  print day + ' finished!'
```

```
20150305 finished!
-1:No Data Returned for request: /market/getOptionTicksHistOneDa
y.csv?field=&optionId=10000030&date=20150306&startSecOffset=&end
SecOffset=
-1:No Data Returned for request: /market/getOptionTicksHistOneDa
y.csv?field=&optionId=10000032&date=20150306&startSecOffset=&end
SecOffset=
-1:No Data Returned for request: /market/getOptionTicksHistOneDa
y.csv?field=&optionId=10000033&date=20150306&startSecOffset=&end
SecOffset=
-1:No Data Returned for request: /market/getOptionTicksHistOneDa
y.csv?field=&optionId=10000035&date=20150306&startSecOffset=&end
SecOffset=
-1:No Data Returned for request: /market/getOptionTicksHistOneDa
y.csv?field=&optionId=10000054&date=20150306&startSecOffset=&end
SecOffset=
-1:No Data Returned for request: /market/getOptionTicksHistOneDa
y.csv?field=&optionId=10000056&date=20150306&startSecOffset=&end
SecOffset=
20150306 finished!
20150309 finished!
-1:No Data Returned for request: /market/getOptionTicksHistOneDa
y.csv?field=&optionId=10000039&date=20150310&startSecOffset=&end
SecOffset=
-1:No Data Returned for request: /market/getOptionTicksHistOneDa
y.csv?field=&optionId=10000056&date=20150310&startSecOffset=&end
SecOffset=
-1:No Data Returned for request: /market/getOptionTicksHistOneDa
y.csv?field=&optionId=10000064&date=20150310&startSecOffset=&end
SecOffset=
20150310 finished!
-1:No Data Returned for request: /market/getOptionTicksHistOneDa
y.csv?field=&optionId=10000039&date=20150311&startSecOffset=&end
SecOffset=
-1:No Data Returned for request: /market/getOptionTicksHistOneDa
y.csv?field=&optionId=10000064&date=20150311&startSecOffset=&end
SecOffset=
20150311 finished!
```

```
res.optionId = res.optionId.astype('str')

res = res.merge(data, how = 'left', on = 'optionId')
```

```
dateData, idData, volumeData = res.dataDate, res.optionId, res['
volume']

previous = [dateData[0], idData[0], 0]
newVolume = np.zeros((len(dateData),))
count = 0
for date, ids, volume in zip(dateData, idData, volumeData ):
    if date == previous[0] and ids == previous[1]:
        newVolume[count] = volume - previous[2]
    else:
        newVolume[count] = volume
    previous[0] = date
    previous[1] = ids
    previous[2] = volume
    count = count + 1

res.volume = newVolume

res['pdDateTime'] = res.expDate.apply(lambda x: x.toDateTime())

optData = pd.DataFrame()
optData['contractType'] = res['contractType']
optData['valuationDate'] = res['dataDate']
optData['expDate'] = res['expDate']
optData['strikePrice'] = res['strikePrice']
optData['lastPrice'] = res['lastPrice']
optData['optionId'] = res['optionId'].astype('str')
optData['Type'] = Option.Call
optData['spotPrice'] = res.spotPrice
optData.loc[optData['contractType'] == 'PO','Type'] = Option.Put

optData['valuationDate'] = [Date(int(date.split('-')[0]),int(dat
e.split('-')[1]),int(date.split('-')[2])) for date in optData['v
aluationDate']]

dc = DayCounter('Actual/365 (Fixed)')
optData['ttm'] = [dc.yearFraction(date1, date2) for date1, date2
in zip(optData['valuationDate'], optData['expDate'])]

optData['lastPrice(vol)'] = BSMImpliedVolatity(optData['Type'],
optData['strikePrice'], optData['spotPrice'], 0.0, 0.0, optData[
'ttm'], optData['lastPrice'])
optData['bid1(vol)'] = BSMImpliedVolatity(optData['Type'], optDa
ta['strikePrice'], optData['spotPrice'], 0.0, 0.0, optData['ttm'
], res.bidPrice1)
optData['ask1(vol)'] = BSMImpliedVolatity(optData['Type'], optDa
ta['strikePrice'], optData['spotPrice'], 0.0, 0.0, optData['ttm'
], res.askPrice1)

res1 = res.merge(optData[[u'spotPrice', u'ttm', u'lastPrice(vol)'
, u'bid1(vol)', u'ask1(vol)']], left_index=True, right_index=True
)
```

```
res1 = res1.dropna(how = 'any')

res1['bidAskSpread(bps)'] = (res1.askPrice1 - res1.bidPrice1) * 10000
res1['bidAskSpread(vol bps)'] = (res1['ask1(vol)'] - res1['bid1(vol)']) * 10000

res1.to_csv('option_data.csv')
```

二 期权系列

[50ETF 期权] 1. 历史成交持仓和 PCR 数据

> 来源：https://uqer.io/community/share/5604937ff9f06c597665ef34

在本文中，我们将通过量化实验室提供的数据，计算上证50ETF期权的历史成交持仓和PCR数据，并在最后利用PCR建立一个简单的择时策略

```
from CAL.PyCAL import *
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import rc
rc('mathtext', default='regular')
import seaborn as sns
sns.set_style('white')
from matplotlib import dates
```

1. 期权数据接口

有关上证50ETF期权数据，量化实验室有三个接口，分别对应于不同的功能

- `DataAPI.OptGet` ：可以获取已退市和上市的所有期权的基本信息
- `DataAPI.MktOptdGet` ：拿到历史上某一天或某段时间的期权成交行情信息
- `DataAPI.MktTickRTSnapshotGet` ：此为高频数据，获取期权最新市场信息快照

在接下来对于期权的数据分析中，我们将使用这三个API提供的数据，以下为API使用示例，具体API的详情可以查看帮助文档

```
# 使用DataAPI.OptGet，拿到已退市和上市的所有期权的基本信息
opt_info = DataAPI.OptGet(optID='', contractStatus=[u"DE", u"L"]
, field='', pandas="1")
opt_info.head(3)
```

	secID	optID	secShortName	ticker
0	510050C1503M02200.XSHG	10000001	50ETF购3月 2200	510050C1
1	510050C1503M02250.XSHG	10000002	50ETF购3月 2250	510050C1
2	510050C1503M02300.XSHG	10000003	50ETF购3月 2300	510050C1

```
3 rows × 23 columns
```

```
#使用DataAPI.MktOptdGet，拿到历史上某一天的期权成交信息
opt_mkt = DataAPI.MktOptdGet(tradeDate='20150921', field='', pan
das="1")
opt_mkt.head(2)
```

	secID	optID	ticker	sec
0	510050C1512M02100.XSHG	10000368	510050C1512M02100	50E 210
1	510050P1512M01950.XSHG	10000369	510050P1512M01950	50E 195

```
# 获取期权最新市场信息快照
opt_mkt_snapshot = DataAPI.MktOptionTickRTSnapshotGet(optionId=u
"",field=u"",pandas="1")
opt_mkt_snapshot[opt_mkt_snapshot.dataDate=='2015-09-22'].head(2
)
```

	optionId	timestamp	auctionPrice	auctionQty	dataDate	data

```
0 rows × 37 columns
```

2. 期权历史成交持仓数据图

```
# 华夏上证50ETF收盘价数据
secID = '510050.XSHG'
begin = Date(2015, 2, 9)
end = Date.todaysDate()
fields = ['tradeDate', 'closePrice']
etf = DataAPI.MktFunddGet(secID, beginDate=begin.toISO().replace(
'-', ''), endDate=end.toISO().replace('-', ''), field=fields)
etf['tradeDate'] = pd.to_datetime(etf['tradeDate'])
etf = etf.set_index('tradeDate')
etf.tail(2)
```

	closePrice
tradeDate	
2015-09-23	2.180
2015-09-24	2.187

统计50ETF期权历史成交量和持仓量信息

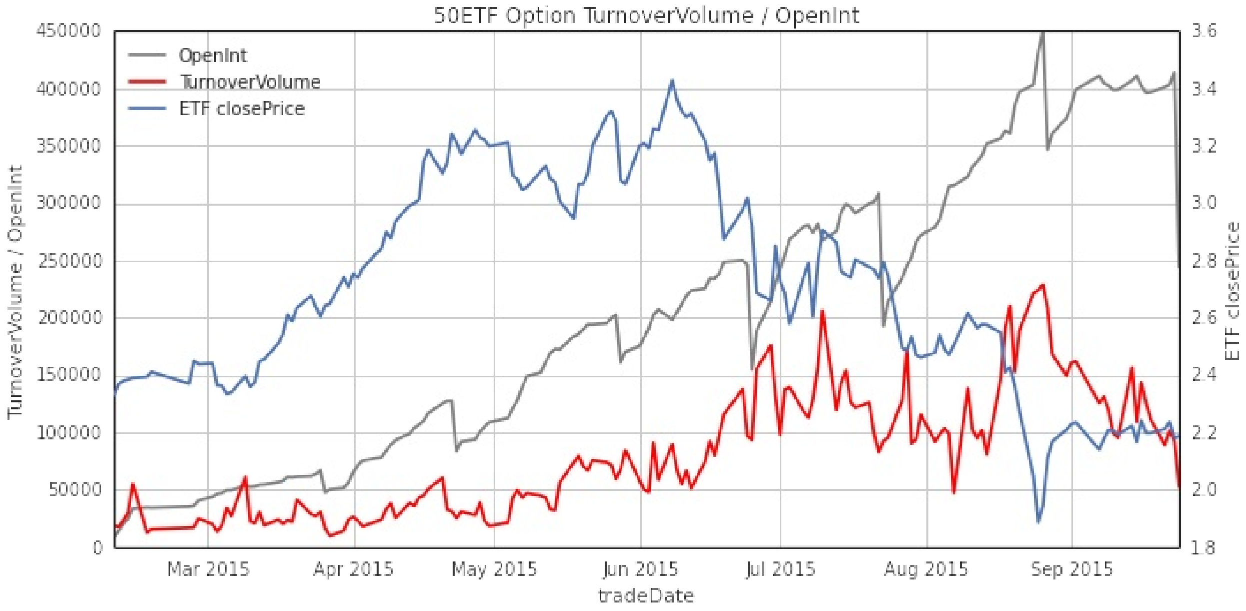
```
# 计算历史一段时间内的50ETF期权持仓量交易量数据
def getOptHistVol(beginDate, endDate):
    optionVarSecID = u"510050.XSHG"
    cal = Calendar('China.SSE')
    cal.addHoliday(Date(2015,9,3))
    cal.addHoliday(Date(2015,9,4))

    dates = cal.bizDatesList(beginDate, endDate)
    dates = map(Date.toDateTime, dates)
    columns = ['callVol', 'putVol', 'callValue',
               'putValue', 'callOpenInt', 'putOpenInt',
               'nearCallVol', 'nearPutVol', 'nearCallValue',
               'nearPutValue', 'nearCallOpenInt', 'nearPutOpenIn
t',
               'netVol', 'netValue', 'netOpenInt',
               'volPCR', 'valuePCR', 'openIntPCR',
               'nearVolPCR', 'nearValuePCR', 'nearOpenIntPCR']
    hist_opt = pd.DataFrame(0.0, index=dates, columns=columns)
    hist_opt.index.name = 'date'
    # 每一个交易日数据单独计算
    for date in hist_opt.index:
        date_str = Date.fromDateTime(date).toISO().replace('-',
'')
        try:
            opt_data = DataAPI.MktOptdGet(secID=u"", tradeDate=d
ate_str, field=u"", pandas="1")
        except:
            hist_opt = hist_opt.drop(date)
            continue

        opt_type = []
        exp_date = []
        for ticker in opt_data.secID.values:
            opt_type.append(ticker[6])
            exp_date.append(ticker[7:11])
        opt_data['optType'] = opt_type
        opt_data['expDate'] = exp_date
        near_exp = np.sort(opt_data.expDate.unique())[0]

        data = opt_data.groupby('optType')
        # 计算所有上市期权：看涨看跌交易量、看涨看跌交易额、看涨看跌持仓量
        hist_opt['callVol'][date] = data.turnoverVol.sum()['C']
        hist_opt['putVol'][date] = data.turnoverVol.sum()['P']
        hist_opt['callValue'][date] = data.turnoverValue.sum()['C']
        hist_opt['putValue'][date] = data.turnoverValue.sum()['P']
        hist_opt['callOpenInt'][date] = data.openInt.sum()['C']
        hist_opt['putOpenInt'][date] = data.openInt.sum()['P']

        near_data = opt_data[opt_data.expDate == near_exp]
        near_data = near_data.groupby('optType')
        # 计算近月期权(主力合约)： 看涨看跌交易量、看涨看跌交易额、看涨看跌持仓量
        hist_opt['nearCallVol'][date] = near_data.turnoverVol.sum()['C']
        hist_opt['nearPutVol'][date] = near_data.turnoverVol.sum()['P']
        hist_opt['nearCallValue'][date] = near_data.turnoverValue.sum()['C']
        hist_opt['nearPutValue'][date] = near_data.turnoverValue.sum()['P']
        hist_opt['nearCallOpenInt'][date] = near_data.openInt.sum()['C']
        hist_opt['nearPutOpenInt'][date] = near_data.openInt.sum()['P']

        # 计算所有上市期权： 总交易量、总交易额、总持仓量
        hist_opt['netVol'][date] = hist_opt['callVol'][date] + hist_opt['putVol'][date]
        hist_opt['netValue'][date] = hist_opt['callValue'][date] + hist_opt['putValue'][date]
        hist_opt['netOpenInt'][date] = hist_opt['callOpenInt'][date] + hist_opt['putOpenInt'][date]

        # 计算期权看跌看涨期权交易量(持仓量)的比率：
        # 交易量看跌看涨比率，交易额看跌看涨比率, 持仓量看跌看涨比率
        # 近月期权交易量看跌看涨比率，近月期权交易额看跌看涨比率, 近月期权持仓量看跌看涨比率
        # PCR = Put Call Ratio
        hist_opt['volPCR'][date] = round(hist_opt['putVol'][date]*1.0/hist_opt['callVol'][date], 4)
        hist_opt['valuePCR'][date] = round(hist_opt['putValue'][date]*1.0/hist_opt['callValue'][date], 4)
        hist_opt['openIntPCR'][date] = round(hist_opt['putOpenInt'][date]*1.0/hist_opt['callOpenInt'][date], 4)
        hist_opt['nearVolPCR'][date] = round(hist_opt['nearPutVol'][date]*1.0/hist_opt['nearCallVol'][date], 4)
        hist_opt['nearValuePCR'][date] = round(hist_opt['nearPutValue'][date]*1.0/hist_opt['nearCallValue'][date], 4)
        hist_opt['nearOpenIntPCR'][date] = round(hist_opt['nearPutOpenInt'][date]*1.0/hist_opt['nearCallOpenInt'][date], 4)
    return hist_opt
```

```
begin = Date(2015, 2, 9)
end = Date.todaysDate()

opt_hist = getOptHistVol(begin, end)
opt_hist.tail(2)
```

	callVol	putVol	callValue	putValue	callOpenInt	putOpe
date						
2015-09-23	50093	42910	37809117	41517121	269395	144256
2015-09-24	29352	23474	21696859	22161955	146224	98350

```
2 rows × 21 columns
```

```
## ----- 50ETF期权成交持仓数据图 -----
fig = plt.figure(figsize=(10,5))
fig.set_tight_layout(True)
ax = fig.add_subplot(111)
font.set_size(16)

lns1 = ax.plot(opt_hist.index, opt_hist.netOpenInt, 'grey', label = u'OpenInt')
lns2 = ax.plot(opt_hist.index, opt_hist.netVol, '-r', label = 'TurnoverVolume')
ax2 = ax.twinx()
lns3 = ax2.plot(etf.index, etf.closePrice, '-', label = 'ETF closePrice')

lns = lns1+lns2+lns3
labs = [l.get_label() for l in lns]
ax.legend(lns, labs, loc=2)
ax.grid()
ax.set_xlabel(u"tradeDate")
ax.set_ylabel(r"TurnoverVolume / OpenInt")
ax2.set_ylabel(r"ETF closePrice")
plt.title('50ETF Option TurnoverVolume / OpenInt')
plt.show()
```



从上图可以看出：

- 期权的交易量基本上是50ETF的反向指标
- 五月之前的疯牛中，期权日交易量处于低位
- 六月中下旬之后的暴跌时间段，期权日交易量高位运行，是不是创个新高
- 8月17日开始的这一周中，大盘风雨飘摇，50ETF探底时，期权交易量创了新高
- 目前来看，期权交易仍然活跃，但是交易量较之前数据有所回落，应该是大盘企稳的节奏

3. 期权的PCR比例

期权分看跌和看涨两种，买入两种不同的期权，代表着对于后市的不同看法，因此可以引进一个量化指标，来表示对后市看衰与看涨的力量的强弱：

- PCR = Put Call Ratio
- PCR可以是关于成交量的PCR，可以是持仓量的PCR，也可以是成交额的PCR

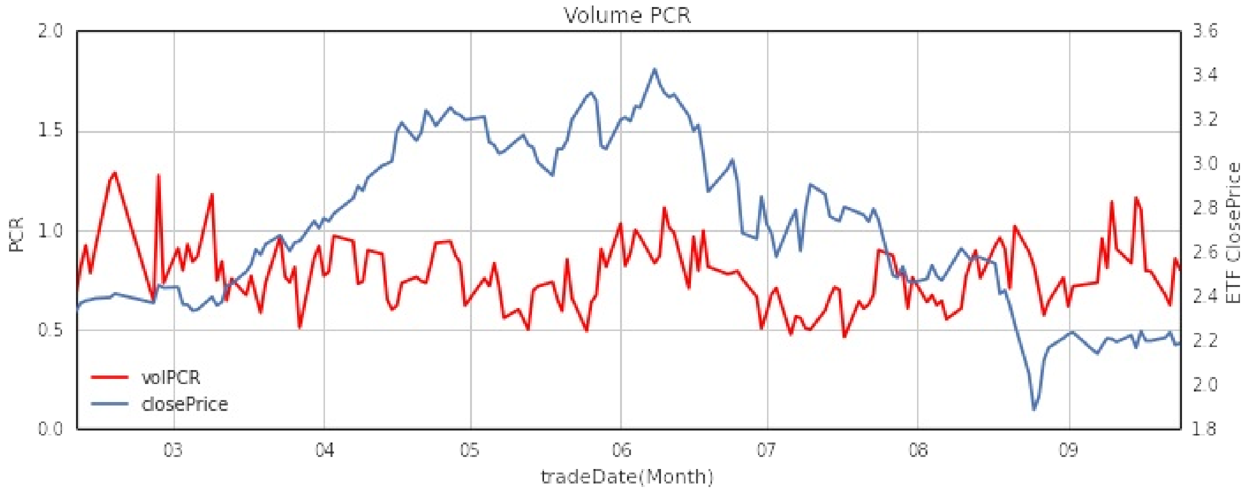
```
begin = Date(2015, 2, 9)
end = Date.todaysDate()

opt_hist = getOptHistVol(begin, end)
opt_hist.tail(2)
```

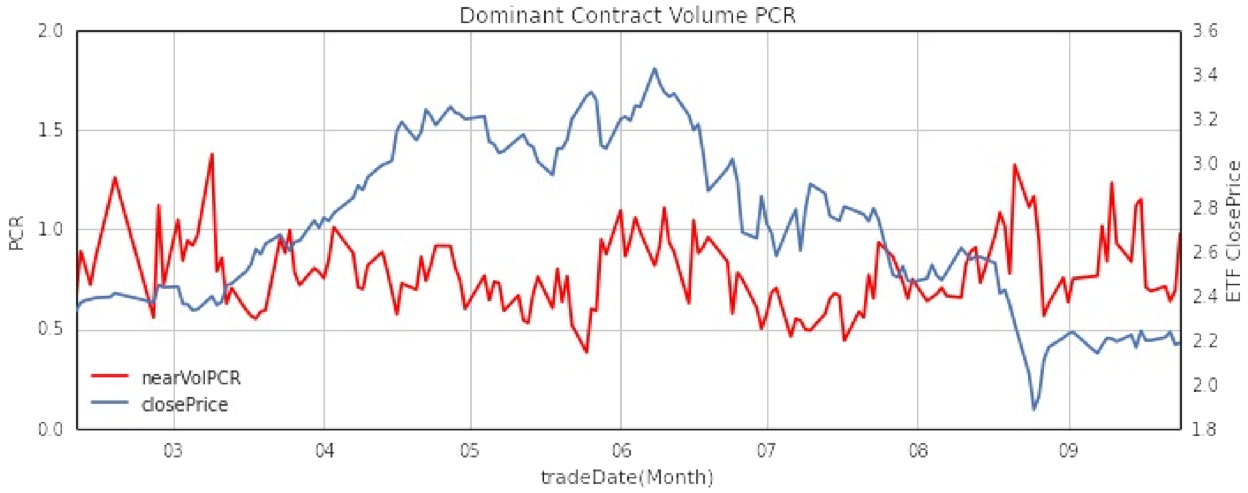
	callVol	putVol	callValue	putValue	callOpenInt	putOpe
date						
2015-09-23	50093	42910	37809117	41517121	269395	144256
2015-09-24	29352	23474	21696859	22161955	146224	98350

```
2 rows × 21 columns
```

首先，我们来看看成交量PCR和ETF价格走势的关系

```
## ----------------------------------------------
## 50ETF期权PC比例数据图
fig = plt.figure(figsize=(10,8))
fig.set_tight_layout(True)

# ------ 成交量PC比例 ------
ax = fig.add_subplot(211)
lns1 = ax.plot(opt_hist.index, opt_hist.volPCR, color='r', label
 = u'volPCR')
ax2 = ax.twinx()
lns2 = ax2.plot(etf.index, etf.closePrice, '-', label = 'closePr
ice')
lns = lns1+lns2
labs = [l.get_label() for l in lns]
ax.legend(lns, labs, loc=3)
ax.set_ylim(0, 2)

hfmt = dates.DateFormatter('%m')
ax.xaxis.set_major_formatter(hfmt)
ax.grid()
ax.set_xlabel(u"tradeDate(Month)")
ax.set_ylabel(r"PCR")
ax2.set_ylabel(r"ETF ClosePrice")
plt.title('Volume PCR')

# ------ 近月主力期权成交量PC比例 ------
ax = fig.add_subplot(212)
lns1 = ax.plot(opt_hist.index, opt_hist.nearVolPCR, color='r', l
abel = u'nearVolPCR')
ax2 = ax.twinx()
lns2 = ax2.plot(etf.index, etf.closePrice, '-', label = 'closePr
ice')
lns = lns1+lns2
labs = [l.get_label() for l in lns]
ax.legend(lns, labs, loc=3)
ax.set_ylim(0, 2)

hfmt = dates.DateFormatter('%m')
ax.xaxis.set_major_formatter(hfmt)
ax.grid()
ax.set_xlabel(u"tradeDate(Month)")
ax.set_ylabel(r"PCR")
ax2.set_ylabel(r"ETF ClosePrice")
plt.title('Dominant Contract Volume PCR')

<matplotlib.text.Text at 0x6470990>
```



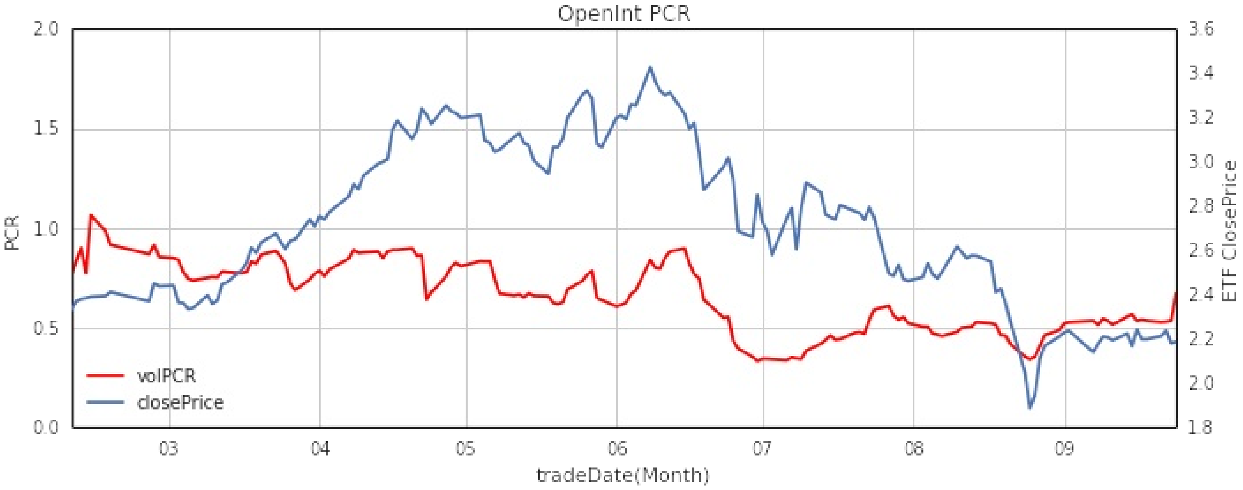


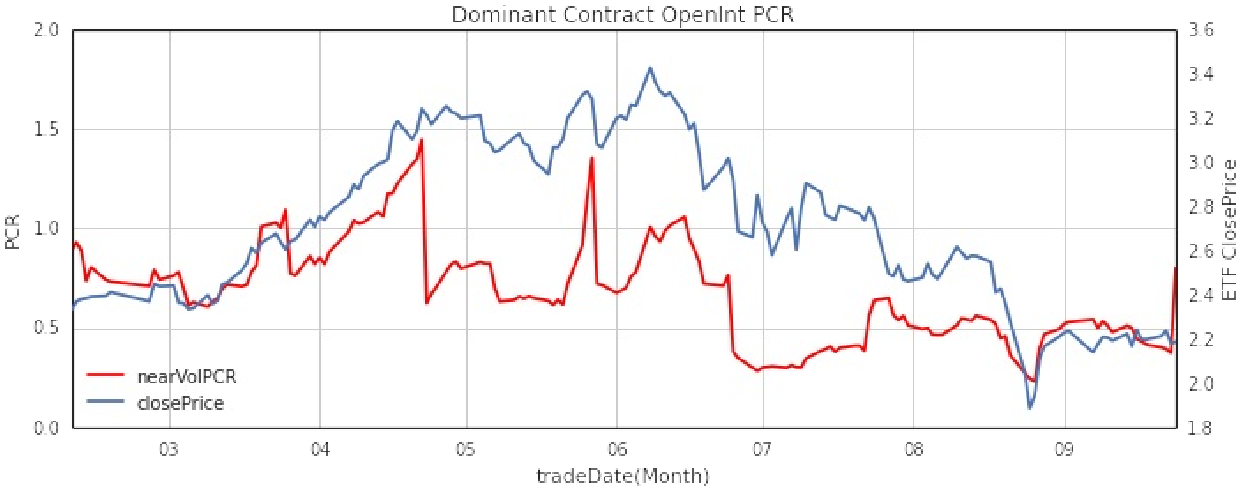
成交量数据图中，上图为全体期权的成交量PCR，下图为近月期权的成交量PCR：

- 上下两图中，PCR的曲线走势基本相似，因为期权交易中，近月期权最为活跃
- ETF价格走势，和PCR走势有比较明显的负相关性

其次，我们来看看持仓量PCR和ETF价格走势的关系

```
## -----------------------------------------------
## 50ETF期权PC比例数据图
fig = plt.figure(figsize=(10,8))
fig.set_tight_layout(True)

# ------ 持仓量PC比例 ------
ax = fig.add_subplot(211)
lns1 = ax.plot(opt_hist.index, opt_hist.openIntPCR, color='r', label = u'volPCR')
ax2 = ax.twinx()
lns2 = ax2.plot(etf.index, etf.closePrice, '-', label = 'closePrice')
lns = lns1+lns2
labs = [l.get_label() for l in lns]
ax.legend(lns, labs, loc=3)
ax.set_ylim(0, 2)

hfmt = dates.DateFormatter('%m')
ax.xaxis.set_major_formatter(hfmt)
ax.grid()
ax.set_xlabel(u"tradeDate(Month)")
ax.set_ylabel(r"PCR")
ax2.set_ylabel(r"ETF ClosePrice")
plt.title('OpenInt PCR')

# ------ 近月主力期权持仓量PC比例 ------
ax = fig.add_subplot(212)
lns1 = ax.plot(opt_hist.index, opt_hist.nearOpenIntPCR, color='r', label = u'nearVolPCR')
ax2 = ax.twinx()
lns2 = ax2.plot(etf.index, etf.closePrice, '-', label = 'closePrice')
lns = lns1+lns2
labs = [l.get_label() for l in lns]
ax.legend(lns, labs, loc=3)
ax.set_ylim(0, 2)

hfmt = dates.DateFormatter('%m')
ax.xaxis.set_major_formatter(hfmt)
ax.grid()
ax.set_xlabel(u"tradeDate(Month)")
ax.set_ylabel(r"PCR")
ax2.set_ylabel(r"ETF ClosePrice")
plt.title('Dominant Contract OpenInt PCR')

<matplotlib.text.Text at 0x69e5990>
```





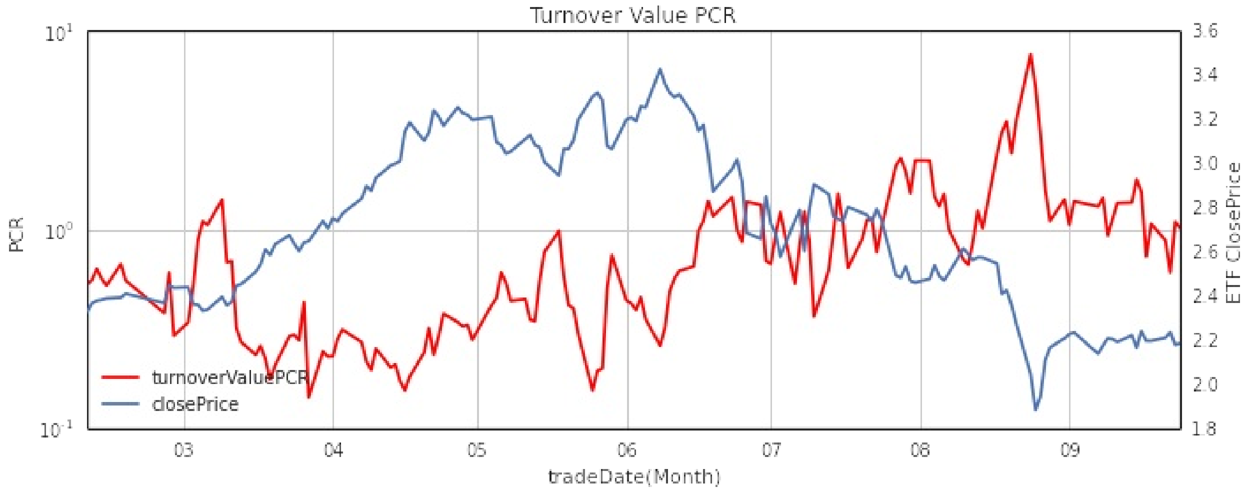
持仓量数据图中，上图为全体期权的持仓量PCR，下图为近月期权的持仓量PCR：

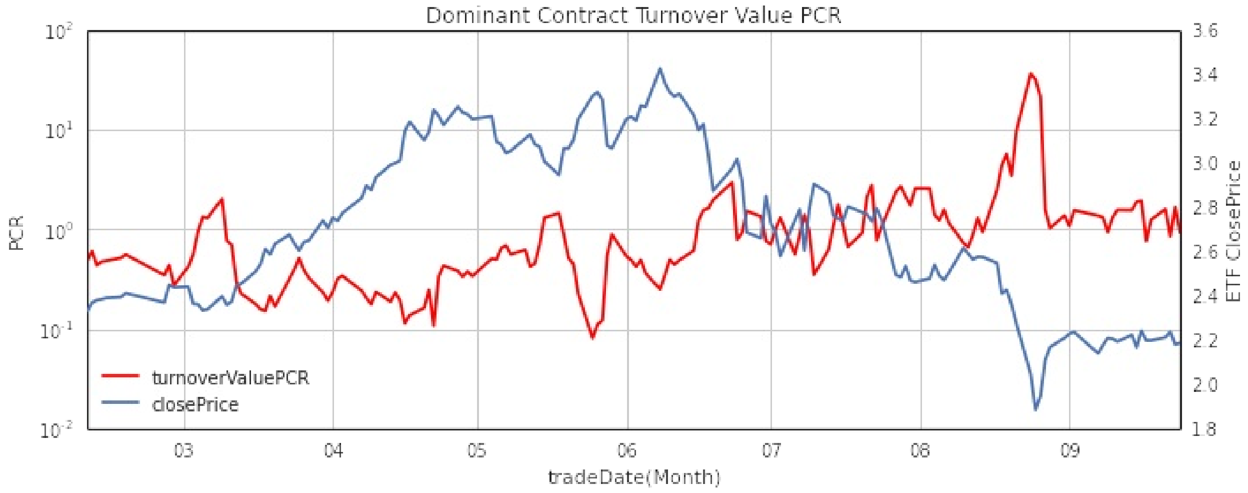
- 上下两图中，PCR的曲线走势基本相似，因为期权交易中，近月期权最为活跃
- 实际上，近月期权十分活跃，使得近月期权的PCR系数变动往往比整体期权PCR变化更剧烈
- ETF价格走势，和PCR走势并无明显的负相关性
- 相反，ETF价格的低点，往往PCR也处于低点，这其实说明：股价大跌之后大家会选择平仓看跌期权

最后，我们来看看成交额PCR和ETF价格走势的关系

```
## ---------------------------------------------
## 50ETF期权PC比例数据图
fig = plt.figure(figsize=(10,8))
fig.set_tight_layout(True)

# ------ 成交额PC比例 ------
ax = fig.add_subplot(211)
lns1 = ax.plot(opt_hist.index, opt_hist.valuePCR, color='r', lab
el = u'turnoverValuePCR')
ax2 = ax.twinx()
lns2 = ax2.plot(etf.index, etf.closePrice, '-', label = 'closePr
ice')
lns = lns1+lns2
labs = [l.get_label() for l in lns]
ax.legend(lns, labs, loc=3)
#ax.set_ylim(0, 2)
ax.set_yscale('log')

hfmt = dates.DateFormatter('%m')
ax.xaxis.set_major_formatter(hfmt)
ax.grid()
ax.set_xlabel(u"tradeDate(Month)")
ax.set_ylabel(r"PCR")
ax2.set_ylabel(r"ETF ClosePrice")
plt.title('Turnover Value PCR')

# ------ 近月主力期权成交额PC比例 ------
ax = fig.add_subplot(212)
lns1 = ax.plot(opt_hist.index, opt_hist.nearValuePCR, color='r',
 label = u'turnoverValuePCR')
ax2 = ax.twinx()
lns2 = ax2.plot(etf.index, etf.closePrice, '-', label = 'closePr
ice')
lns = lns1+lns2
labs = [l.get_label() for l in lns]
ax.legend(lns, labs, loc=3)
#ax.set_ylim(0, 2)
ax.set_yscale('log')

hfmt = dates.DateFormatter('%m')
ax.xaxis.set_major_formatter(hfmt)
ax.grid()
ax.set_xlabel(u"tradeDate(Month)")
ax.set_ylabel(r"PCR")
ax2.set_ylabel(r"ETF ClosePrice")
plt.title('Dominant Contract Turnover Value PCR')
```

```
<matplotlib.text.Text at 0x70ce890>
```





成交额数据图中，上图为全体期权的成交额PCR，下图为近月期权的成交额PCR：

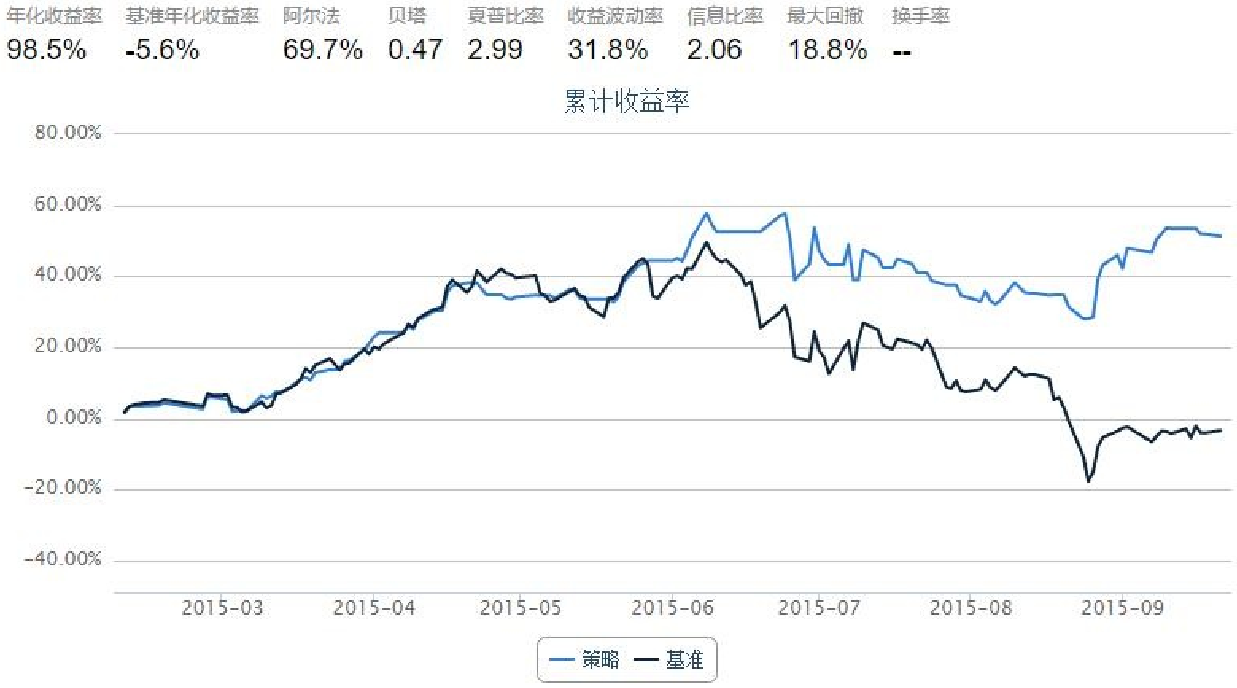
- 上下两图中，PCR的曲线走势基本相似，因为期权交易中，近月期权最为活跃
- 实际上，近月期权PCR指数十分活跃，使得近月期权的PCR系数变动往往比整体期权PCR变化更剧烈
- 相对于成交量和持仓量PCR指标，此处的成交额PCR指标峰值往往很高，上图中近月期权的成交额PCR最大值甚至接近30，这是由于市场恐慌时候，看跌期权成交量本身就大，而交易量大往往将看跌期权的价格大幅抬高
- ETF价格走势，和PCR走势具有明显的负相关性

- 基于期权成交额PCR的择时策略

根据成交额PCR和ETF价格走势明显的负相关性，我们建立一个非常简单的择时策略：

- PCR下降时，市场情绪趋稳定，全仓买入50ETF
- PCR上升时，恐慌情绪蔓延，清仓观望

```
start = datetime(2015, 2, 9)                # 回测起始时间
end  = datetime(2015, 9, 21)                # 回测结束时间

hist_pcr = getOptHistVol(start, end)

start = datetime(2015, 2, 9)                # 回测起始时间
end  = datetime(2015, 9, 21)                # 回测结束时间
benchmark = '510050.XSHG'                    # 策略参考标准
universe = ['510050.XSHG']                    # 股票池
capital_base = 100000                       # 起始资金
commission = Commission(0.0,0.0)
refresh_rate = 1

def initialize(account):                    # 初始化虚拟账户状态
    account.fund = universe[0]

def handle_data(account):             # 每个交易日的买入卖出指令
    fund = account.fund
    #  获取回测当日的前一天日期
    dt = Date.fromDateTime(account.current_date)
    cal = Calendar('China.IB')
    cal.addHoliday(Date(2015,9,3))
    cal.addHoliday(Date(2015,9,4))

    last_day = cal.advanceDate(dt,'-1B',BizDayConvention.Precedi
ng)            #计算出倒数第一个交易日
    last_last_day = cal.advanceDate(last_day,'-1B',BizDayConvent
ion.Preceding)  #计算出倒数第二个交易日
    last_day_str = last_day.strftime("%Y-%m-%d")
    last_last_day_str = last_last_day.strftime("%Y-%m-%d")

    # 计算买入卖出信号
    try:
        # 拿取PCR数据
        pcr_last = hist_pcr['valuePCR'].loc[last_day_str]
        pcr_last_last = hist_pcr['valuePCR'].loc[last_last_day_s
tr]
        long_flag = True if (pcr_last - pcr_last_last) < 0 else
False
    except:
        long_flag = True

    if long_flag:
        approximationAmount = int(account.cash / account.referen
cePrice[fund] / 100.0) * 100
        order(fund, approximationAmount)
    else:
        # 卖出时，全仓清空
        order_to(fund, 0)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
98.5%	-5.6%	69.7%	0.47	2.99	31.8%	2.06	18.8%	--

回测结果如上，需要注意的是：

- 期权挂牌时间较短，回测时间短，加上期权市场参与人数少，故而回测结果可能然并卵
- 但是严格根据PCR走势买卖50ETF，还是可以比较好的避开市场大跌的风险
- 不管怎样，PCR可以作为一个择时指标来讨论
- 除了成交额PCR，还可以通过成交量、持仓量、近月成交额等等PCR建立择时策略

【50ETF期权】 2. 历史波动率

> 来源：https://uqer.io/community/share/560493a4f9f06c597565ef03

在本文中，我们将通过量化实验室提供的数据，计算上证50ETF的历史波动率数据

```
from CAL.PyCAL import *
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import rc
rc('mathtext', default='regular')
import seaborn as sns
sns.set_style('white')
import math
from scipy.stats import mstats
```

50ETF收盘价

```
# 华夏上证50ETF
secID = '510050.XSHG'
begin = Date(2015, 2, 9)
end = Date.todaysDate()
fields = ['tradeDate', 'closePrice']
etf = DataAPI.MktFunddGet(secID, beginDate=begin.toISO().replace(
'-', ''), endDate=end.toISO().replace('-', ''), field=fields)
etf['tradeDate'] = pd.to_datetime(etf['tradeDate'])
etf = etf.set_index('tradeDate')
etf.tail(3)
```

	closePrice
tradeDate	
2015-09-22	2.237
2015-09-23	2.180
2015-09-24	2.187

1. EWMA模型计算历史波动率

EWMA(Exponentially Weighted Moving Average)指数加权移动平均计算历史波动率：

$$\sigma_n^2 = \lambda\sigma_{n-1}^2 + (1-\lambda)u_{n-1}^2$$

其中

$$u_n = \ln\frac{S_n}{S_{n-1}} \quad \text{或者} \quad u_n = \frac{S_n - S_{n-1}}{S_{n-1}}$$

上式中的 `Si` 为 `i` 天的收盘价， `λ` 为介于0和1之间的常数。也就是说，在第 `n-1` 天估算的第 `n` 天的波动率估计值 `σn` 由第 `n-1` 天的波动率估计值 `σn-1` 和收盘价在最近一天的变化百分比 `un-1` 决定。

计算周期为 `N` 天的波动率时， `λ` 可以取为：

$$\lambda = \frac{N-1}{N+1}$$

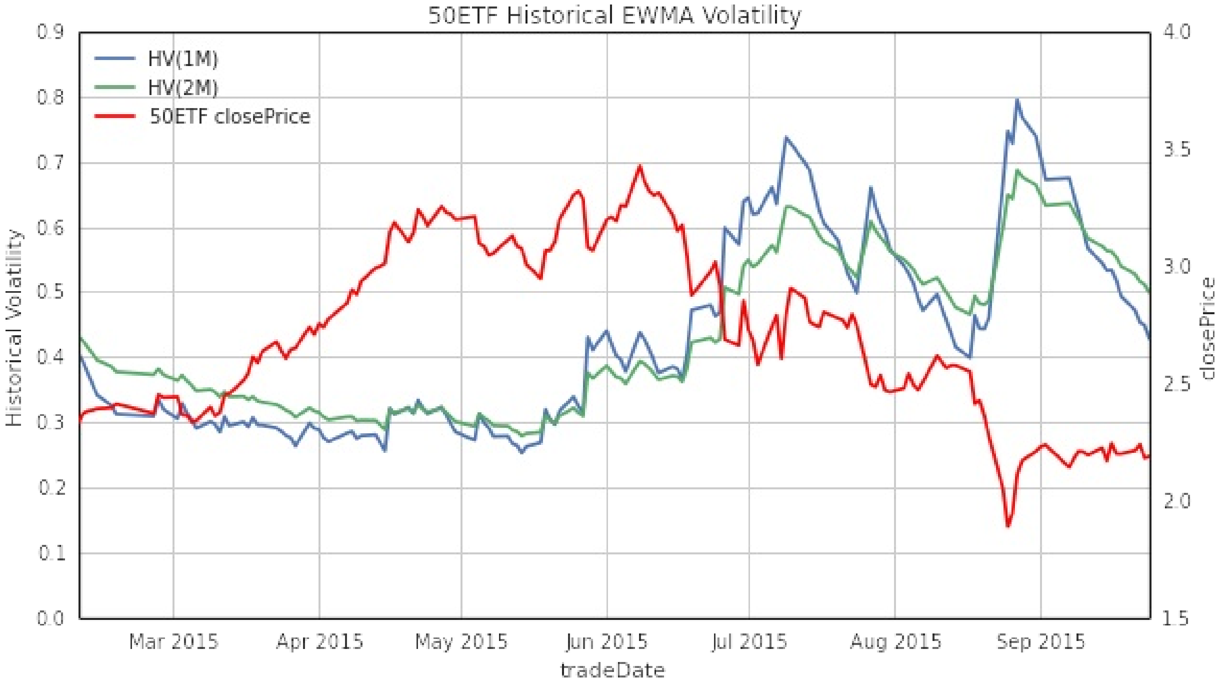
```
def getHistVolatilityEWMA(secID, beginDate, endDate):
    cal = Calendar('China.SSE')
    spotBeginDate = cal.advanceDate(beginDate,'-520B',BizDayConvention.Preceding)
    spotBeginDate = Date(2006, 1, 1)
    begin = spotBeginDate.toISO().replace('-', '')
    end = endDate.toISO().replace('-', '')

    fields = ['tradeDate', 'preClosePrice', 'closePrice', 'settlePrice', 'preSettlePrice']
    security = DataAPI.MktFunddGet(secID, beginDate=begin, endDate=end, field=fields)
    security['dailyReturn'] = security['closePrice']/security['preClosePrice']   # 日回报率
    security['u2'] = (np.log(security['dailyReturn']))**2      # u2为复利形式的日回报率平方
    # security['u2'] = (security['dailyReturn'] - 1.0)**2      # u2为日价格变化百分比的平方
    security['tradeDate'] = pd.to_datetime(security['tradeDate'])

    periods = {'hv1W': 5, 'hv2W': 10, 'hv1M': 21, 'hv2M': 41, 'hv3M': 62, 'hv4M': 83,
               'hv5M': 104, 'hv6M': 124, 'hv9M': 186, 'hv1Y': 249, 'hv2Y': 497}
    # 利用pandas中的ewma模型计算波动率
    for prd in periods.keys():
        # 此处的span实际上就是上面计算波动率公式中lambda表达式中的N
        security[prd] = np.round(np.sqrt(pd.ewma(security['u2'], span=periods[prd], adjust=False)), 5)*math.sqrt(252.0)

    security = security[security.tradeDate >= beginDate.toISO()]
    security = security.set_index('tradeDate')
    return security
```

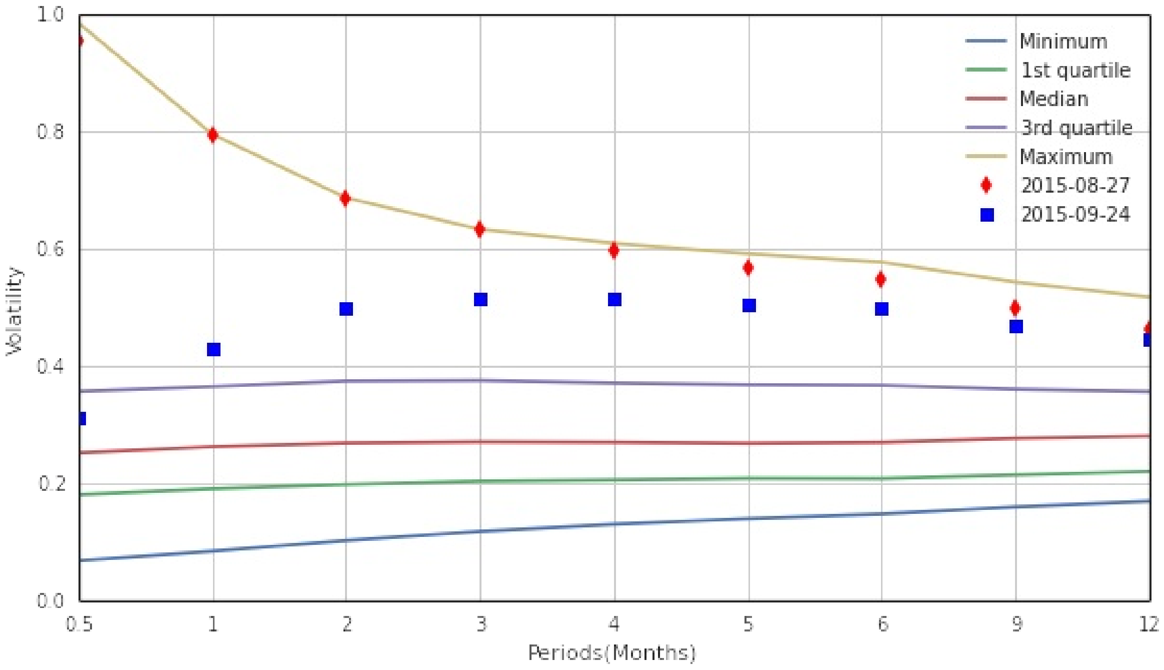
```
secID = '510050.XSHG'
start = Date(2015, 2, 9)
end = Date.todaysDate()
hist_HV = getHistVolatilityEWMA(secID, start, end)
hist_HV.tail(2)
```

	preClosePrice	closePrice	dailyReturn	u2	hv2
tradeDate					
2015-09-23	2.237	2.180	0.974519	0.000666	0.511
2015-09-24	2.180	2.187	1.003211	0.000010	0.499

```
secID = '510050.XSHG'
start = Date(2007, 1, 1)
end = Date.todaysDate()
hist_HV = getHistVolatilityEWMA(secID, start, end)

## ----- 50ETF历史波动率 -----
fig = plt.figure(figsize=(10,12))
ax = fig.add_subplot(211)
font.set_size(16)

hist_plot = hist_HV[hist_HV.index >= Date(2015,2,9).toISO()]
etf_plot = etf[etf.index >= Date(2015,2,9).toISO()]
lns1 = ax.plot(hist_plot.index, hist_plot.hv1M, '-', label = u'HV(1M)')
lns2 = ax.plot(hist_plot.index, hist_plot.hv2M, '-', label = u'HV(2M)')
ax2 = ax.twinx()
lns3 = ax2.plot(etf_plot.index, etf_plot.closePrice, '-r', label = '50ETF closePrice')

lns = lns1+lns2+lns3
labs = [l.get_label() for l in lns]
ax.legend(lns, labs, loc=2)
ax.grid()
ax.set_xlabel(u"tradeDate")
ax.set_ylabel(r"Historical Volatility")
ax2.set_ylabel(r"closePrice")
ax.set_ylim(0, 0.9)
ax2.set_ylim(1.5, 4)
plt.title('50ETF Historical EWMA Volatility')


## -----------------------------------
## ----- 50ETF历史波动率统计数据 -----
# 注意： 该统计数据基于07年以来将近九年的历史波动率得出
ax3 = fig.add_subplot(212)
font.set_size(16)

hist_plot = hist_HV[[u'hv2W', u'hv1M', u'hv2M', u'hv3M', u'hv4M', u'hv5M', u'hv6M', u'hv9M', u'hv1Y']]

# Calculate the quantiles column wise
```

```
quantiles = mstats.mquantiles(hist_plot, prob=[0.0, 0.25, 0.5, 0
.75, 1.0], axis=0)
labels = ['Minimum', '1st quartile', 'Median', '3rd quartile', '
Maximum']
for i, q in enumerate(quantiles):
    ax3.plot(q, label=labels[i])

# 在统计图中标出某一天的波动率
date = Date(2015,8,27)
last_day_HV = hist_plot.ix[date.toDateTime()].T
ax3.plot(last_day_HV.values, 'dr', label=date.toISO())

# 在统计图中标出最近一天的波动率
last_day_HV = hist_plot.tail(1).T
ax3.plot(last_day_HV.values, 'sb', label=Date.fromDateTime(last_
day_HV.columns[0]).toISO())

ax3.set_ylabel(r"Volatility")
plt.xticks((0,1,2,3,4,5,6,7,8),(0.5,1,2,3,4,5,6,9,12))
plt.xlabel('Periods(Months)')
plt.legend()
plt.grid()
```





波动率图中，上图表示50ETF收盘价格和历史波动率的走势关系：

- 显然，短周期波动率对于近期的波动更敏感
- 收盘价的下跌往往伴随着波动率的上升，两者的负相关性质明显

波动率图中，下图表示50ETF历史波动率的统计数据，图中给出了四分位波动率锥：

- 8月底时，各个周期历史波动率均处于历史高位
- 目前，短周期波动率已经有所回落

2. Close to Close 模型计算历史波动率

m 天周期的Close to Close波动率：

$$\sigma_n^2 = \frac{1}{m-1} \Sigma_{i=1}^{m} (u_{n-i} - \bar{u})$$

其中

$$\bar{u} = \frac{1}{m} \Sigma_{i=1}^{m} u_{n-i}$$

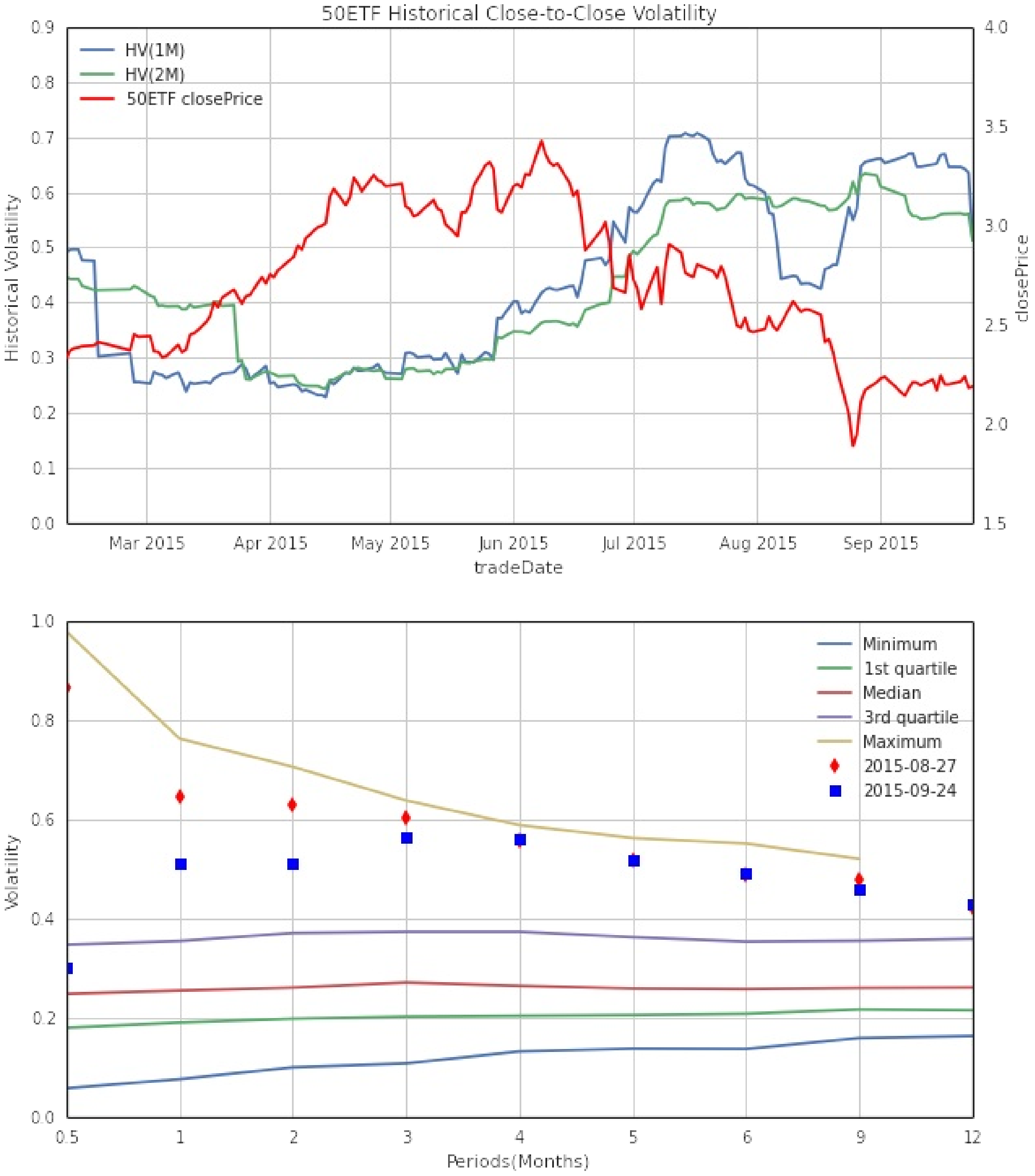
$$u_n = \ln \frac{S_n}{S_{n-1}} \quad \text{或者} \quad u_n = \frac{S_n - S_{n-1}}{S_{n-1}}$$

也就是说，在第 `n-1` 天估算的第 `n` 天的波动率估计值 `σn` 由前面 `m` 天的每日收盘价变化百分比 `ui` 的标准差决定。

```python
## 计算一段时间标的的历史波动率，返回值包括以下不同周期的波动率：
# 一周，半月，一个月，两个月，三个月，四个月，五个月，半年，九个月，一年，两年

def getHistVolatilityC2C(secID, beginDate, endDate):
    cal = Calendar('China.SSE')
    spotBeginDate = cal.advanceDate(beginDate,'-520B',BizDayConvention.Preceding)
    spotBeginDate = Date(2006, 1, 1)
    begin = spotBeginDate.toISO().replace('-', '')
    end = endDate.toISO().replace('-', '')

    fields = ['tradeDate', 'preClosePrice', 'closePrice', 'settlePrice', 'preSettlePrice']
    security = DataAPI.MktFunddGet(secID, beginDate=begin, endDate=end, field=fields)
    security['dailyReturn'] = security['closePrice']/security['preClosePrice']    # 日回报率
    security['u'] = np.log(security['dailyReturn'])       # u2为复利形式的日回报率
    security['tradeDate'] = pd.to_datetime(security['tradeDate'])

    periods = {'hv1W': 5, 'hv2W': 10, 'hv1M': 21, 'hv2M': 41, 'hv3M': 62, 'hv4M': 83,
               'hv5M': 104, 'hv6M': 124, 'hv9M': 186, 'hv1Y': 249, 'hv2Y': 497}
    # 利用方差模型计算波动率
    for prd in periods.keys():
        security[prd] = np.round(pd.rolling_std(security['u'], window=periods[prd]), 5)*math.sqrt(252.0)

    security = security[security.tradeDate >= beginDate.toISO()]
    security = security.set_index('tradeDate')
    return security
```

```
secID = '510050.XSHG'
start = Date(2007, 1, 1)
end = Date.todaysDate()
hist_HV = getHistVolatilityC2C(secID, start, end)

## ----- 50ETF历史波动率 -----
fig = plt.figure(figsize=(10,12))
ax = fig.add_subplot(211)
font.set_size(16)

hist_plot = hist_HV[hist_HV.index >= Date(2015,2,9).toISO()]
etf_plot = etf[etf.index >= Date(2015,2,9).toISO()]
lns1 = ax.plot(hist_plot.index, hist_plot.hv1M, '-', label = u'HV(1M)')
lns2 = ax.plot(hist_plot.index, hist_plot.hv2M, '-', label = u'HV(2M)')
ax2 = ax.twinx()
lns3 = ax2.plot(etf_plot.index, etf_plot.closePrice, '-r', label = '50ETF closePrice')

lns = lns1+lns2+lns3
labs = [l.get_label() for l in lns]
ax.legend(lns, labs, loc=2)
ax.grid()
ax.set_xlabel(u"tradeDate")
ax.set_ylabel(r"Historical Volatility")
ax2.set_ylabel(r"closePrice")
ax.set_ylim(0, 0.9)
ax2.set_ylim(1.5, 4)
plt.title('50ETF Historical Close-to-Close Volatility')


## -----------------------------------
## ----- 50ETF历史波动率统计数据 -----
# 注意： 该统计数据基于07年以来将近九年的历史波动率得出
ax3 = fig.add_subplot(212)
font.set_size(16)

hist_plot = hist_HV[[u'hv2W', u'hv1M', u'hv2M', u'hv3M', u'hv4M', u'hv5M', u'hv6M', u'hv9M', u'hv1Y']]

# Calculate the quantiles column wise
quantiles = mstats.mquantiles(hist_plot, prob=[0.0, 0.25, 0.5, 0.75, 1.0], axis=0)
labels = ['Minimum', '1st quartile', 'Median', '3rd quartile', 'Maximum']
for i, q in enumerate(quantiles):
    ax3.plot(q, label=labels[i])

# 在统计图中标出某一天的波动率
date = Date(2015,8,27)
last_day_HV = hist_plot.ix[date.toDateTime()].T
```

```
ax3.plot(last_day_HV.values, 'dr', label=date.toISO())

# 在统计图中标出最近一天的波动率
last_day_HV = hist_plot.tail(1).T
ax3.plot(last_day_HV.values, 'sb', label=Date.fromDateTime(last_
day_HV.columns[0]).toISO())

ax3.set_ylabel(r"Volatility")
plt.xticks((0,1,2,3,4,5,6,7,8),(0.5,1,2,3,4,5,6,9,12))
plt.xlabel('Periods(Months)')
plt.legend()
plt.grid()
```



波动率图中，上图表示50ETF收盘价格和历史波动率的走势关系：

- 显然，短周期波动率对于近期的波动更敏感
- 收盘价的下跌往往伴随着波动率的上升，两者的负相关性质明显

波动率图中，下图表示50ETF历史波动率的统计数据，图中给出了四分位波动率锥：

- 8月底时，各个周期历史波动率均处于历史高位
- 目前，短周期波动率已经有所回落

明显地，相对于EWMA计算的历史波动率，Close to Close波动率对于最近价格波动反应比较迟钝

【50ETF期权】 3. 中国波指 iVIX

> 来源：https://uqer.io/community/share/560493f7f9f06c590c65ef21

在本文中，我们将通过量化实验室提供的数据，计算基于50ETF期权的中国波指iVIX

波动率VIX指数是跟踪市场波动性的指数，一般通过标的期权的隐含波动率计算得来。当VIX越高，表示市场参与者预期后市波动程度会更加激烈，同时也反映其不安的心理状态；相反，VIX越低时，则反映市场参与者预期后市波动程度会趋于缓和。因此，VIX又被称为投资人恐慌指标（The Investor Fear Gauge）。

中国波指由上交所发布，用于衡量上证50ETF未来30日的预期波动。按照上交所网页描述：该指数是根据方差互换的原理，结合50ETF期权的实际运作特点，并通过对上证所交易的50ETF期权价格的计算编制而得。网址为：http://www.sse.com.cn/assortment/derivatives/options/volatility/ ，该网页中发布历史 iVIX 和当日日内 iVIX 数据

```
from CAL.PyCAL import *
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import rc
rc('mathtext', default='regular')
import seaborn as sns
sns.set_style('white')
from matplotlib import dates
from pandas import Series, DataFrame, concat
from scipy import interpolate
import math
import time
```

上证50ETF收盘价，用来和iVIX对比走势

```
# 华夏上证50ETF
secID = '510050.XSHG'
begin = Date(2015, 2, 9)
end = Date.todaysDate()
fields = ['tradeDate', 'closePrice']
etf = DataAPI.MktFunddGet(secID, beginDate=begin.toISO().replace(
'-', ''), endDate=end.toISO().replace('-', ''), field=fields)
etf['tradeDate'] = pd.to_datetime(etf['tradeDate'])
etf = etf.set_index('tradeDate')
etf.tail(2)
```

	closePrice
tradeDate	
2015-09-23	2.180
2015-09-24	2.187

上海银行间同业拆借利率 SHIBOR，用来作为无风险利率参考

```
## 银行间质押式回购利率
def getHistDayInterestRateInterbankRepo(date):
    cal = Calendar('China.SSE')
    period = Period('-10B')
    begin = cal.advanceDate(date, period)
    begin_str = begin.toISO().replace('-', '')
    date_str = date.toISO().replace('-', '')
    # 以下的indicID分别对应的银行间质押式回购利率周期为：
    # 1D, 7D, 14D, 21D, 1M, 3M, 4M, 6M, 9M, 1Y
    indicID = [u"M120000067", u"M120000068", u"M120000069", u"M1
20000070", u"M120000071",
               u"M120000072", u"M120000073", u"M120000074", u"M1
20000075", u"M120000076"]
    period = np.asarray([1.0, 7.0, 14.0, 21.0, 30.0, 90.0, 120.0
, 180.0, 270.0, 360.0]) / 360.0
    period_matrix = pd.DataFrame(index=indicID, data=period, col
umns=['period'])
    field = u"indicID,indicName,publishTime,periodDate,dataValue
,unit"
    interbank_repo = DataAPI.ChinaDataInterestRateInterbankRepoG
et(indicID=indicID,beginDate=begin_str,endDate=date_str,field=fi
eld,pandas="1")
    interbank_repo = interbank_repo.groupby('indicID').first()
    interbank_repo = concat([interbank_repo, period_matrix], axi
s=1, join='inner').sort_index()
    return interbank_repo

## 银行间同业拆借利率
def getHistDaySHIBOR(date):
    date_str = date.toISO().replace('-', '')
    # 以下的indicID分别对应的SHIBOR周期为：
    # 1D, 7D, 14D, 1M, 3M, 6M, 9M, 1Y
    indicID = [u"M120000057", u"M120000058", u"M120000059", u"M1
20000060",
               u"M120000061", u"M120000062", u"M120000063", u"M1
20000064"]
    period = np.asarray([1.0, 7.0, 14.0, 30.0, 90.0, 180.0, 270.0
, 360.0]) / 360.0
    period_matrix = pd.DataFrame(index=indicID, data=period, col
umns=['period'])
    field = u"indicID,indicName,publishTime,periodDate,dataValue
,unit"
```

```
    interest_shibor = DataAPI.ChinaDataInterestRateSHIBORGet(indicID=indicID,beginDate=date_str,endDate=date_str,field=field,pandas="1")
    interest_shibor = interest_shibor.set_index('indicID')
    interest_shibor = concat([interest_shibor, period_matrix], axis=1, join='inner').sort_index()
    return interest_shibor

## 插值得到给定的周期的无风险利率
def periodsSplineRiskFreeInterestRate(date, periods):
    # 此处使用SHIBOR来插值
    init_shibor = getHistDaySHIBOR(date)

    shibor = {}
    min_period = min(init_shibor.period.values)
    max_period = max(init_shibor.period.values)
    for p in periods.keys():
        tmp = periods[p]
        if periods[p] > max_period:
            tmp = max_period * 0.99999
        elif periods[p] < min_period:
            tmp = min_period * 1.00001
        sh = interpolate.spline(init_shibor.period.values, init_shibor.dataValue.values, [tmp], order=3)
        shibor[p] = sh[0]/100.0
    return shibor
```

50ETF历史波动率，用来和iVIX走势作对比

```
## 计算一段时间标的的历史波动率，返回值包括以下不同周期的波动率：
# 一周，半月，一个月，两个月，三个月，四个月，五个月，半年，九个月，一年，两年

def getHistVolatilityEWMA(secID, beginDate, endDate):
    cal = Calendar('China.SSE')
    spotBeginDate = cal.advanceDate(beginDate,'-520B',BizDayConvention.Preceding)
    spotBeginDate = Date(2006, 1, 1)
    begin = spotBeginDate.toISO().replace('-', '')
    end = endDate.toISO().replace('-', '')

    fields = ['tradeDate', 'preClosePrice', 'closePrice', 'settlePrice', 'preSettlePrice']
    security = DataAPI.MktFunddGet(secID, beginDate=begin, endDate=end, field=fields)
    security['dailyReturn'] = security['closePrice']/security['preClosePrice']   # 日回报率
    security['u2'] = (np.log(security['dailyReturn']))**2      # u2为复利形式的日回报率平方
    # security['u2'] = (security['dailyReturn'] - 1.0)**2      # u2为日价格变化百分比的平方
    security['tradeDate'] = pd.to_datetime(security['tradeDate'])

    periods = {'hv1W': 5, 'hv2W': 10, 'hv1M': 21, 'hv2M': 41, 'hv3M': 62, 'hv4M': 83,
               'hv5M': 104, 'hv6M': 124, 'hv9M': 186, 'hv1Y': 249, 'hv2Y': 497}
    # 利用pandas中的ewma模型计算波动率
    for prd in periods.keys():
        # 此处的span实际上就是上面计算波动率公式中lambda表达式中的N
        security[prd] = np.round(np.sqrt(pd.ewma(security['u2'], span=periods[prd], adjust=False)), 5)*math.sqrt(252.0)

    security = security[security.tradeDate >= beginDate.toISO()]
    security = security.set_index('tradeDate')
    return security
```

1. 计算历史每日 iVIX

计算方法参考CBOE的手册：http://www.cboe.com/micro/vix/part2.aspx

```
# 计算历史某一天的iVIX
def calDayVIX(date, opt_info):
    var_sec = u"510050.XSHG"
    # 使用DataAPI.MktOptdGet，拿到历史上某一天的期权行情信息
    date_str = date.toISO().replace('-', '')
    fields_mkt = [u"optID", "tradeDate", "closePrice", 'settlPri
ce']
    opt_mkt = DataAPI.MktOptdGet(tradeDate=date_str, field=field
s_mkt, pandas="1")
    opt_mkt = opt_mkt.set_index(u"optID")
    opt_mkt[u"price"] = opt_mkt['closePrice']

    # concat某一日行情和期权基本信息，得到所需数据
    opt = concat([opt_info, opt_mkt], axis=1, join='inner').sort
_index()
    opt = opt[opt.varSecID==var_sec]
    exp_dates = map(Date.parseISO, np.sort(opt.expDate.unique())
)
    trade_date = date
    exp_periods = {}
    for epd in exp_dates:
        exp_periods[epd] = (epd - date)*1.0/365.0
    risk_free = periodsSplineRiskFreeInterestRate(trade_date, ex
p_periods)

    sigma_square = {}
    for date in exp_dates:
        # 计算某一日的vix
        opt_date = opt[opt.expDate==date.toISO()]
        rf = risk_free[date]
        #rf = 0.05

        opt_call = opt_date[opt_date.contractType == 'CO'].set_i
ndex('strikePrice')
        opt_put = opt_date[opt_date.contractType == 'PO'].set_in
dex('strikePrice')
        opt_call_price = opt_call[[u'price']].sort_index()
        opt_put_price = opt_put[[u'price']].sort_index()
        opt_call_price.columns = [u'callPrice']
        opt_put_price.columns = [u'putPrice']
        opt_call_put_price = concat([opt_call_price, opt_put_pri
ce], axis=1, join='inner').sort_index()
        opt_call_put_price['diffCallPut'] = opt_call_put_price.c
allPrice - opt_call_put_price.putPrice

        strike = abs(opt_call_put_price['diffCallPut']).idxmin()
        price_diff = opt_call_put_price['diffCallPut'][strike]
        ttm = exp_periods[date]
        fw = strike + np.exp(ttm*rf) * price_diff
        strikes = np.sort(opt_call_put_price.index.values)

        delta_K_tmp = np.concatenate((strikes, strikes[-1:], str
ikes[-1:]))
        delta_K_tmp = delta_K_tmp - np.concatenate((strikes[0:1]
, strikes[0:1], strikes))
        delta_K = np.concatenate((delta_K_tmp[1:2], delta_K_tmp[2
:-2]/2, delta_K_tmp[-2:-1]))
        delta_K = pd.DataFrame(delta_K, index=strikes, columns=[
'deltaStrike'])
```

```
        # opt_otm = opt_out_of_money
        opt_otm = concat([opt_call[opt_call.index>fw], opt_put[o
pt_put.index<fw]], axis=0, join='inner')
        opt_otm = concat([opt_otm, delta_K], axis=1, join='inner'
).sort_index()

        # 计算VIX时，比forward price低的第一个行权价被设置为参考行权价，
参考值以上
        # 的call和以下的put均为虚值期权，所有的虚值期权被用来计算VIX，然
而计算中发
        # 现，有时候没有比forward price更低的行权价，例如2015-07-08，
故有以下关于
        # 参考行权价的设置
        strike_ref = fw
        if len((strikes[strikes < fw])) > 0:
            strike_ref = max([k for k in strikes[strikes < fw]])
            opt_otm['price'][strike_ref] = (opt_call['price'][st
rike_ref] + opt_call['price'][strike_ref])/2.0

        exp_rt = np.exp(rf*ttm)
        opt_otm['sigmaTerm'] = opt_otm.deltaStrike*opt_otm.price
/(opt_otm.index)**2
        sigma = opt_otm.sigmaTerm.sum()
        sigma = (sigma*2.0*exp_rt - (fw*1.0/strike_ref - 1.0)**2
)/ttm
        sigma_square[date] = sigma

    # d_one, d_two 将被用来计算VIX(30):
    if exp_periods[exp_dates[0]] >= 1.0/365.0:
        d_one = exp_dates[0]
        d_two = exp_dates[1]
    else:
        d_one = exp_dates[1]
        d_two = exp_dates[2]
    w = (exp_periods[d_two] - 30.0/365.0)/(exp_periods[d_two] -
exp_periods[d_one])
    vix30 = exp_periods[d_one]*w*sigma_square[d_one] + exp_perio
ds[d_two]*(1 - w)*sigma_square[d_two]
    vix30 = 100*np.sqrt(vix30*365.0/30.0)

    # d_one, d_two 将被用来计算VIX(60):
    d_one = exp_dates[1]
    d_two = exp_dates[2]
    w = (exp_periods[d_two] - 60.0/365.0)/(exp_periods[d_two] -
exp_periods[d_one])
    vix60 = exp_periods[d_one]*w*sigma_square[d_one] + exp_perio
ds[d_two]*(1 - w)*sigma_square[d_two]
    vix60 = 100*np.sqrt(vix60*365.0/60.0)

    return vix30, vix60

def getHistDailyVIX(beginDate, endDate):
    # 计算历史一段时间内的VIX指数并返回
    optionVarSecID = u"510050.XSHG"

    # 使用DataAPI.OptGet，一次拿取所有存在过的期权信息，以备后用
    fields_info = ["optID", u"varSecID", u'contractType', u'strikePrice', u'expDate']
    opt_info = DataAPI.OptGet(optID='', contractStatus=[u"DE", u"L"], field=fields_info, pandas="1")
    opt_info = opt_info.set_index(u"optID")

    cal = Calendar('China.SSE')
    cal.addHoliday(Date(2015,9,3))
    cal.addHoliday(Date(2015,9,4))

    dates = cal.bizDatesList(beginDate, endDate)
    histVIX = pd.DataFrame(0.0, index=map(Date.toDateTime, dates), columns=['VIX30','VIX60'])
    histVIX.index.name = 'tradeDate'
    for date in histVIX.index:
        try:
            vix30, vix60 =  calDayVIX(Date.fromDateTime(date), opt_info)
        except:
            histVIX = histVIX.drop(date)
            continue
        histVIX['VIX30'][date] = vix30
        histVIX['VIX60'][date] = vix60
    return histVIX

def getHistOneDayVIX(date):
    # 计算历史某天的VIX指数并返回
    optionVarSecID = u"510050.XSHG"

    # 使用DataAPI.OptGet，一次拿取所有存在过的期权信息，以备后用
    fields_info = ["optID", u"varSecID", u'contractType', u'strikePrice', u'expDate']
    opt_info = DataAPI.OptGet(optID='', contractStatus=[u"DE", u"L"], field=fields_info, pandas="1")
    opt_info = opt_info.set_index(u"optID")

    cal = Calendar('China.SSE')
    cal.addHoliday(Date(2015,9,3))
    cal.addHoliday(Date(2015,9,4))

    if cal.isBizDay(date):
        vix30, vix60 = 0.0, 0.0
        vix30, vix60 =  calDayVIX(date, opt_info)
        return vix30, vix60
    else:
        print date, "不是工作日"
```

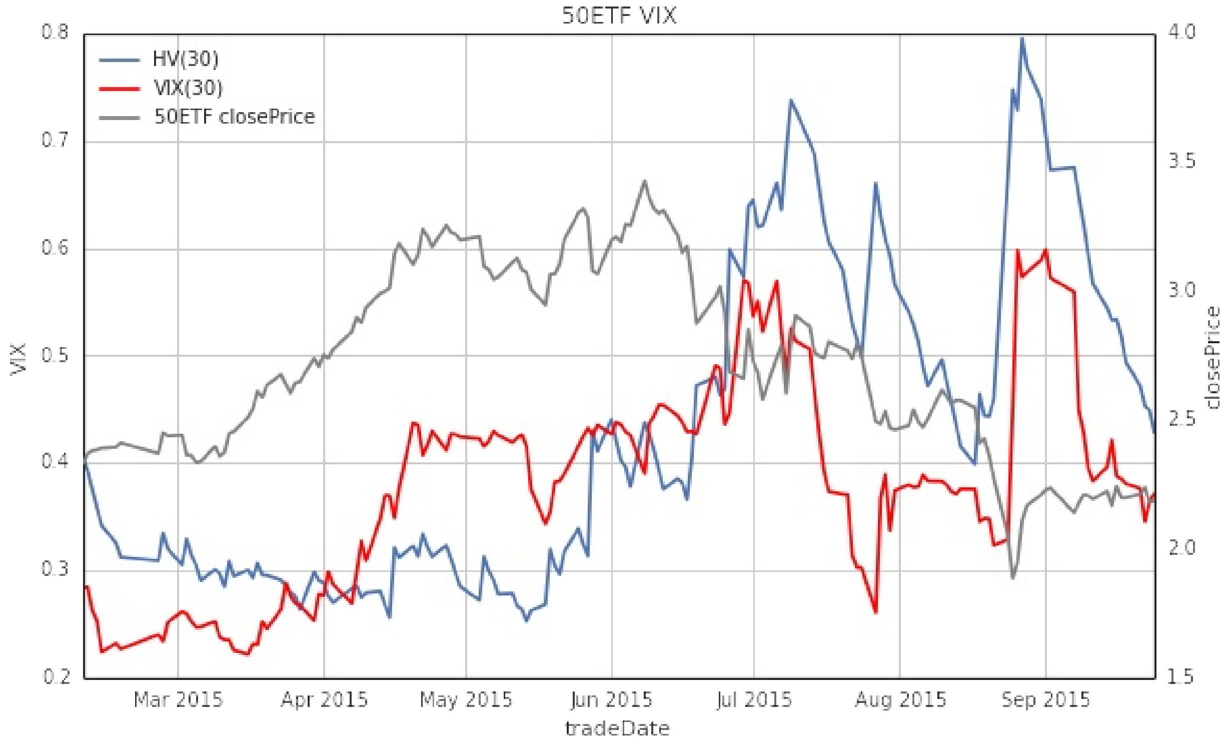
历史每日iVIX 数据

```
begin = Date(2015, 2, 9)    # 起始日
end = Date.todaysDate()     # 截至今天

hist_VIX = getHistDailyVIX(begin, end)
hist_VIX.tail()
```

	VIX30	VIX60
tradeDate		
2015-09-18	38.057648	39.074643
2015-09-21	37.610259	38.559095
2015-09-22	34.507456	36.788384
2015-09-23	36.413426	37.837454
2015-09-24	37.114348	24.346747

iVIX、50ETF收盘价、50ETF波动率比较

```
start = Date(2007, 1, 1)
end = Date.todaysDate()
secID = '510050.XSHG'
hist_HV = getHistVolatilityEWMA(secID, start, end)

## ----- 50ETF VIX指数和历史波动率比较 -----
fig = plt.figure(figsize=(10,6))
ax = fig.add_subplot(111)
font.set_size(16)

hist_HV_plot = hist_HV[hist_HV.index >= Date(2015,2,9).toISO()]
etf_plot = etf[etf.index >= Date(2015,2,9).toISO()]
lns1 = ax.plot(hist_HV_plot.index, hist_HV_plot.hv1M, '-', label
 = u'HV(30)')
lns2 = ax.plot(hist_VIX.index, hist_VIX.VIX30/100.0, '-r', label
 = u'VIX(30)')
#lns3 = ax.plot(hist_VIX.index, hist_VIX.VIX60/100.0, '-g', labe
l = u'VIX(60)')
ax2 = ax.twinx()
lns4 = ax2.plot(etf_plot.index, etf_plot.closePrice, 'grey', lab
el = '50ETF closePrice')

lns = lns1+lns2+lns4
labs = [l.get_label() for l in lns]
ax.legend(lns, labs, loc=2)
ax.grid()
ax.set_xlabel(u"tradeDate")
ax.set_ylabel(r"VIX")
ax2.set_ylabel(r"closePrice")
#ax.set_ylim(0, 0.80)
ax2.set_ylim(1.5, 4)
plt.title('50ETF VIX')

<matplotlib.text.Text at 0x5acec90>
```



2. 基于iVIX的择时策略

策略思路：

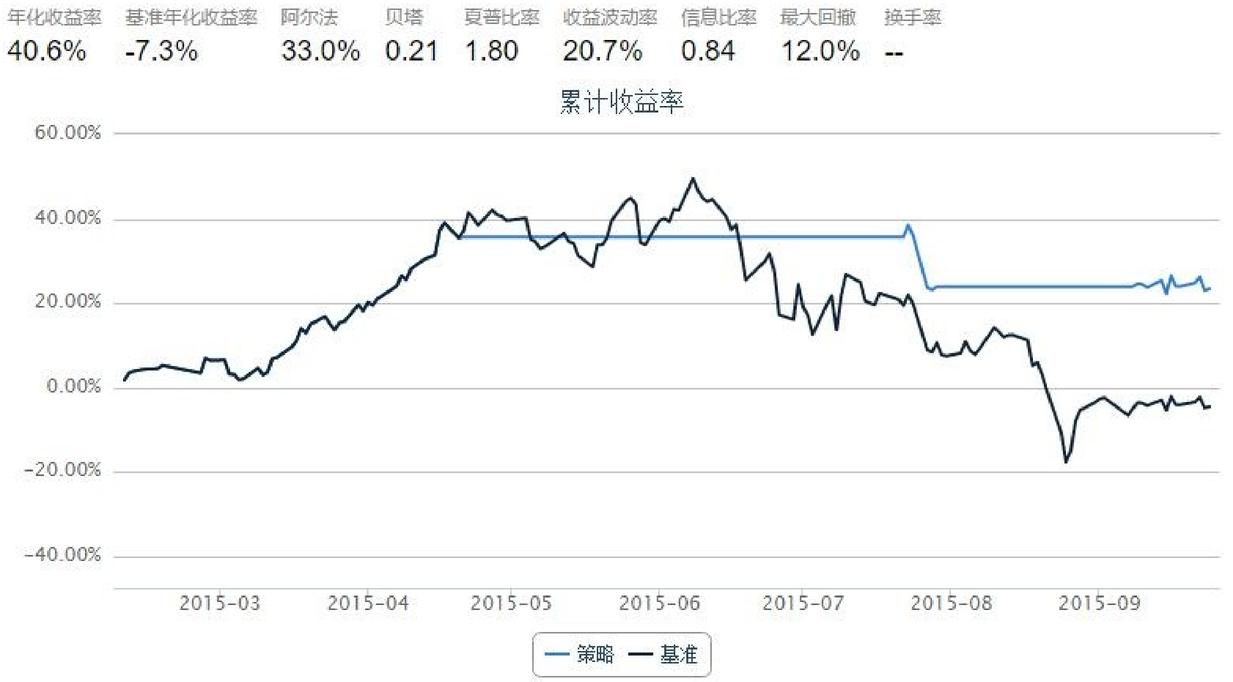
- 计算 VIX 三日均线
- 前一日 VIX 向上穿过三日均线一定比例，则卖出
- 前一日 VIX 向下穿过三日均线一定比例，则买入
- 只买卖50ETF

```
start = datetime(2015, 2, 9)                # 回测起始时间
end  = datetime(2015, 9, 24)                # 回测结束时间

hist_VIX = getHistDailyVIX(start, end)

hist_VIX.tail(2)
```

	VIX30	VIX60
tradeDate		
2015-09-23	36.413426	37.837454
2015-09-24	37.114348	24.346747

```
start = datetime(2015, 2, 9)                # 回测起始时间
end  = datetime(2015, 9, 24)                # 回测结束时间
benchmark = '510050.XSHG'                     # 策略参考标准
```

```
universe = ['510050.XSHG']                       # 股票池
capital_base = 100000                          # 起始资金
commission = Commission(0.0,0.0)

window_short = 1
window_long = 3
SD = 0.1

hist_VIX['short_window'] = pd.rolling_mean(hist_VIX['VIX30'], wi
ndow=window_short)
hist_VIX['long_window'] = pd.rolling_mean(hist_VIX['VIX30'], win
dow=window_long)

def initialize(account):                       # 初始化虚拟账户状态
    account.fund = universe[0]

def handle_data(account):                # 每个交易日的买入卖出指令
    fund = account.fund
    #  获取回测当日的前一天日期
    dt = Date.fromDateTime(account.current_date)
    cal = Calendar('China.IB')
    cal.addHoliday(Date(2015,9,3))
    cal.addHoliday(Date(2015,9,4))

    last_day = cal.advanceDate(dt,'-1B',BizDayConvention.Precedi
ng)            #计算出倒数第一个交易日
    last_last_day = cal.advanceDate(last_day,'-1B',BizDayConvent
ion.Preceding)  #计算出倒数第二个交易日
    last_day_str = last_day.strftime("%Y-%m-%d")
    last_last_day_str = last_last_day.strftime("%Y-%m-%d")

    # 计算买入卖出信号
    try:
        short_mean = hist_VIX['short_window'].loc[last_day_str]
# 短均线值
        long_mean = hist_VIX['long_window'].loc[last_day_str]
# 长均线值
        long_flag = True if (short_mean - long_mean) < - SD * lo
ng_mean else False
        short_flag = True if (short_mean - long_mean) > SD * lon
g_mean else False
    except:
        long_flag = True
        short_flag = True

    if long_flag:
        approximationAmount = int(account.cash / account.referen
cePrice[fund] / 100.0) * 100
        order(fund, approximationAmount)
    elif short_flag:
        # 卖出时，全仓清空
        order_to(fund, 0)
```



3. 日内跟踪计算 iVIX

计算方法和日间iVIX类似

```
def calSnapshotVIX(date, opt_info):
    var_sec = u"510050.XSHG"
    # 使用DataAPI.MktOptdGet，拿到历史上某一天的期权行情信息
    date_str = date.toISO().replace('-', '')
    fields_mkt = [u'optionId', u'dataDate', u'highPrice', u'last
Price', u'lowPrice', u'openPrice', u'preSettlePrice', u'bidBook_
price1', u'bidBook_volume1', u'askBook_price1', u'askBook_volume
1']
    # opt_mkt = DataAPI.MktOptdGet(tradeDate=date_str, field=fie
lds_mkt, pandas="1")
    opt_mkt = DataAPI.MktOptionTickRTSnapshotGet(optionId=u"", f
ield='', pandas="1")
    opt_mkt = opt_mkt[opt_mkt.dataDate == date.toISO()]
    opt_mkt['optID'] = map(int, opt_mkt['optionId'])
    opt_mkt = opt_mkt.set_index(u"optID")
    opt_mkt[u"price"] = (opt_mkt['bidBook_price1'] + opt_mkt['as
kBook_price1'])/2.0

    # concat某一日行情和期权基本信息，得到所需数据
    opt = concat([opt_info, opt_mkt], axis=1, join='inner').sort
_index()

    #opt = opt[opt.varSecID==var_sec]
    exp_dates = map(Date.parseISO, np.sort(opt.expDate.unique())
)
    trade_date = date
    exp_periods = {}
    for epd in exp_dates:
        exp_periods[epd] = (epd - date)*1.0/365.0
    risk_free = periodsSplineRiskFreeInterestRate(trade_date, ex
p_periods)

    sigma_square = {}
    for date in exp_dates:
        # 计算某一日的vix
        opt_date = opt[opt.expDate==date.toISO()]
        rf = risk_free[date]
        #rf = 0.05

        opt_call = opt_date[opt_date.contractType == 'CO'].set_i
ndex('strikePrice')
        opt_put = opt_date[opt_date.contractType == 'PO'].set_in
dex('strikePrice')
        opt_call_price = opt_call[[u'price']].sort_index()
        opt_put_price = opt_put[[u'price']].sort_index()
        opt_call_price.columns = [u'callPrice']
        opt_put_price.columns = [u'putPrice']
        opt_call_put_price = concat([opt_call_price, opt_put_pri
ce], axis=1, join='inner').sort_index()
        opt_call_put_price['diffCallPut'] = opt_call_put_price.c
allPrice - opt_call_put_price.putPrice

        strike = abs(opt_call_put_price['diffCallPut']).idxmin()
        price_diff = opt_call_put_price['diffCallPut'][strike]
        ttm = exp_periods[date]
        fw = strike + np.exp(ttm*rf) * price_diff
        strikes = np.sort(opt_call_put_price.index.values)

        delta_K_tmp = np.concatenate((strikes, strikes[-1:], str
ikes[-1:]))
        delta_K_tmp = delta_K_tmp - np.concatenate((strikes[0:1]
, strikes[0:1], strikes))
        delta_K = np.concatenate((delta_K_tmp[1:2], delta_K_tmp[2
:-2]/2, delta_K_tmp[-2:-1]))
        delta_K = pd.DataFrame(delta_K, index=strikes, columns=[
'deltaStrike'])

        # opt_otm = opt_out_of_money
        opt_otm = concat([opt_call[opt_call.index>fw], opt_put[o
pt_put.index<fw]], axis=0, join='inner')
        opt_otm = concat([opt_otm, delta_K], axis=1, join='inner'
).sort_index()

        # 计算VIX时，比forward price低的第一个行权价被设置为参考行权价，
参考值以上
        # 的call和以下的put均为虚值期权，所有的虚值期权被用来计算VIX，然
而计算中发
        # 现，有时候没有比forward price更低的行权价，例如2015-07-08，
故有以下关于
```

```
        # 参考行权价的设置
        strike_ref = fw
        if len((strikes[strikes < fw])) > 0:
            strike_ref = max([k for k in strikes[strikes < fw]])
            opt_otm['price'][strike_ref] = (opt_call['price'][st
rike_ref] + opt_call['price'][strike_ref])/2.0

        exp_rt = np.exp(rf*ttm)
        opt_otm['sigmaTerm'] = opt_otm.deltaStrike*opt_otm.price
/(opt_otm.index)**2
        sigma = opt_otm.sigmaTerm.sum()
        sigma = (sigma*2.0*exp_rt - (fw*1.0/strike_ref - 1.0)**2
)/ttm
        sigma_square[date] = sigma

    # d_one, d_two 将被用来计算VIX(30):
    if exp_periods[exp_dates[0]] >= 1.0/365.0:
        d_one = exp_dates[0]
        d_two = exp_dates[1]
    else:
        d_one = exp_dates[1]
        d_two = exp_dates[2]
    w = (exp_periods[d_two] - 30.0/365.0)/(exp_periods[d_two] -
exp_periods[d_one])
    vix30 = exp_periods[d_one]*w*sigma_square[d_one] + exp_perio
ds[d_two]*(1 - w)*sigma_square[d_two]
    vix30 = 100*np.sqrt(vix30*365.0/30.0)

    # d_one, d_two 将被用来计算VIX(60):
    d_one = exp_dates[1]
    d_two = exp_dates[2]
    w = (exp_periods[d_two] - 60.0/365.0)/(exp_periods[d_two] -
exp_periods[d_one])
    vix60 = exp_periods[d_one]*w*sigma_square[d_one] + exp_perio
ds[d_two]*(1 - w)*sigma_square[d_two]
    vix60 = 100*np.sqrt(vix60*365.0/60.0)
    return vix30, vix60


def getTodaySnapshotVIX():
    # 计算历史某天的VIX指数并返回
    optionVarSecID = u"510050.XSHG"
    date = Date.todaysDate()

    # 使用DataAPI.OptGet，一次拿取所有存在过的期权信息，以备后用
    fields_info = ["optID", u"varSecID", u'contractType', u'stri
kePrice', u'expDate']
    opt_info = DataAPI.OptGet(optID='', contractStatus=[u"DE", u
"L"], field=fields_info, pandas="1")
    opt_info = opt_info.set_index(u"optID")

    cal = Calendar('China.SSE')
    cal.addHoliday(Date(2015,9,3))
    cal.addHoliday(Date(2015,9,4))

    if cal.isBizDay(date):
        now_long = datetime.now()
        now = now_long.time().isoformat()
        if (now > '09:25:00' and now < '11:30:00') or (now > '13:00:00' and now < '15:00:00'):
            vix30, vix60 =  calSnapshotVIX(date, opt_info)
            vix = pd.DataFrame([[date, vix30, vix60]], index=[now_long], columns=['dataDate', 'VIX30', 'VIX60'])
            vix.index.name = 'time'
        else:
            vix = pd.DataFrame(0.0, index=[], columns=['dataDate', 'VIX30', 'VIX60'])
            vix.index.name = 'time'
        return vix
    else:
        print "今天： ", date, " 不是工作日"
```

计算即时的VIX

如果在工作日非交易时间运行计算函数，则得到一个空的 `dataframe`

```
getTodaySnapshotVIX()
```

	dataDate	VIX30	VIX60
time			

跟踪计算当日日内 VIX 走势

```
## 此函数跟踪计算并记录当日日内VIX走势，数据记录在：
# 文件 'VIX_intraday_' + Date.todaysDate().toISO() + '.csv' 中
# 该文件保存在登录uqer账号的 Data 空间中
# seconds 为跟踪计算间隔秒数
def trackTodayIntradayVIX(seconds):
    vix_file_str = 'VIX_intraday_' + Date.todaysDate().toISO() + '.csv'
    vix = pd.DataFrame(0.0, index=[], columns=['dataDate', 'VIX30', 'VIX60'])
    vix.index.name = 'time'
    vix.to_csv(vix_file_str)

    now = datetime.now().time()
    while now.isoformat() < '15:00:00':
        vix = pd.read_csv(vix_file_str).set_index('time')
        vix_now = getTodaySnapshotVIX()
        if vix_now.shape[0] > 0:
            vix = vix.append(vix_now)
            vix.to_csv(vix_file_str)
            # print vix_now.index[0], '\t', vix_now.VIX30[0], '\t', vix_now.VIX60[0]
        time.sleep(seconds)
        now = datetime.now().time()
```

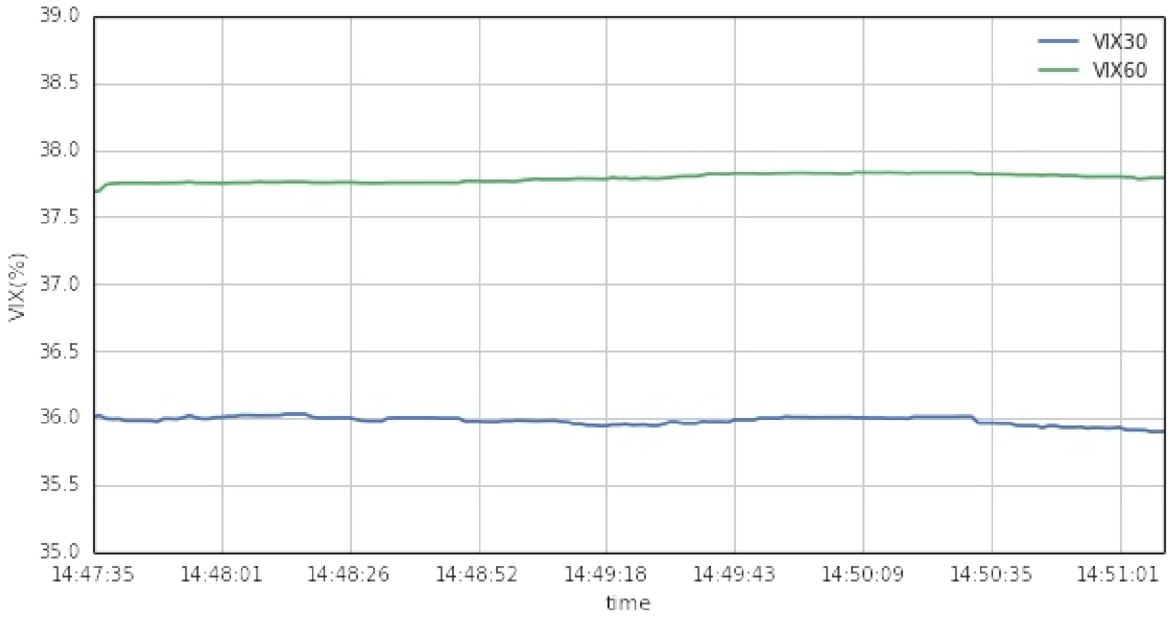
注意：

`trackTodayIntradayVIX` 函数一经运行，便持续到当日收盘时，除非手动终止运行

```
# 追踪当前iVIX走势，每隔60秒计算一次即时iVIX
time_interval = 60
trackTodayIntradayVIX(time_interval)

---------------------------------------------------------------
-----------
KeyboardInterrupt                         Traceback (most recent
 call last)
<mercury-input-20-3f8b5a5070f8> in <module>()
      1 # 追踪当前iVIX走势，每隔60秒计算一次即时iVIX
      2 time_interval = 60
----> 3 trackTodayIntradayVIX(time_interval)

<mercury-input-19-d53f12cb0e4a> in trackTodayIntradayVIX(seconds
)
     17               vix.to_csv(vix_file_str)
     18               # print vix_now.index[0], '\t', vix_now.VIX3
0[0], '\t', vix_now.VIX60[0]
---> 19           time.sleep(seconds)
     20           now = datetime.now().time()

KeyboardInterrupt:
```

将当日追踪到的iVIX日内走势作图，注意读取数据文件名和 trackTodayIntradayVIX 函数中的存储文件名一致

```
vix_file_str = 'VIX_intraday_2015-09-23-backup.csv'
vix = pd.read_csv(vix_file_str)
vix['time'] = [x[11:19] for x in vix.time]
vix = vix.set_index('time')

ax = vix.plot(figsize=(10,5))
ax.set_xlabel('time')
ax.set_ylabel('VIX(%)')
ax.set_ylim(35, 39)

(35, 39)
```


39.0
38.5
38.0
37.5
37.0
36.5
36.0
35.5
35.0
VIX(%)
14:47:35
14:48:01
14:48:26
14:48:52
14:49:18
14:49:43
14:50:09
14:50:35
14:51:01
time
VIX30
VIX60

【50ETF期权】 4. Greeks 和隐含波动率微笑

> 来源：https://uqer.io/community/share/560769faf9f06c597165ef75

在本文中，我们将通过量化实验室提供的数据，计算上证50ETF期权的隐含波动率微笑。

```
from CAL.PyCAL import *
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import rc
rc('mathtext', default='regular')
import seaborn as sns
sns.set_style('white')
import math
from scipy import interpolate
from scipy.stats import mstats
from pandas import Series, DataFrame, concat
import time
from matplotlib import dates
```

上海银行间同业拆借利率 SHIBOR，用来作为无风险利率参考

```
## 银行间质押式回购利率
def getHistDayInterestRateInterbankRepo(date):
    cal = Calendar('China.SSE')
    period = Period('-10B')
    begin = cal.advanceDate(date, period)
    begin_str = begin.toISO().replace('-', '')
    date_str = date.toISO().replace('-', '')
    # 以下的indicID分别对应的银行间质押式回购利率周期为：
    # 1D, 7D, 14D, 21D, 1M, 3M, 4M, 6M, 9M, 1Y
    indicID = [u"M120000067", u"M120000068", u"M120000069", u"M1
20000070", u"M120000071",
               u"M120000072", u"M120000073", u"M120000074", u"M1
20000075", u"M120000076"]
    period = np.asarray([1.0, 7.0, 14.0, 21.0, 30.0, 90.0, 120.0
, 180.0, 270.0, 360.0]) / 360.0
    period_matrix = pd.DataFrame(index=indicID, data=period, col
umns=['period'])
    field = u"indicID,indicName,publishTime,periodDate,dataValue
,unit"
    interbank_repo = DataAPI.ChinaDataInterestRateInterbankRepoG
et(indicID=indicID,beginDate=begin_str,endDate=date_str,field=fi
eld,pandas="1")
    interbank_repo = interbank_repo.groupby('indicID').first()
    interbank_repo = concat([interbank_repo, period_matrix], axi
s=1, join='inner').sort_index()
    return interbank_repo

## 银行间同业拆借利率
def getHistDaySHIBOR(date):
    cal = Calendar('China.SSE')
    period = Period('-10B')
    begin = cal.advanceDate(date, period)
    begin_str = begin.toISO().replace('-', '')
    date_str = date.toISO().replace('-', '')
    # 以下的indicID分别对应的SHIBOR周期为：
    # 1D, 7D, 14D, 1M, 3M, 6M, 9M, 1Y
    indicID = [u"M120000057", u"M120000058", u"M120000059", u"M120000060",
               u"M120000061", u"M120000062", u"M120000063", u"M120000064"]
    period = np.asarray([1.0, 7.0, 14.0, 30.0, 90.0, 180.0, 270.0, 360.0]) / 360.0
    period_matrix = pd.DataFrame(index=indicID, data=period, columns=['period'])
    field = u"indicID,indicName,publishTime,periodDate,dataValue,unit"
    interest_shibor = DataAPI.ChinaDataInterestRateSHIBORGet(indicID=indicID,beginDate=begin_str,endDate=date_str,field=field,pandas="1")
    interest_shibor = interest_shibor.groupby('indicID').first()
    interest_shibor = concat([interest_shibor, period_matrix], axis=1, join='inner').sort_index()
    return interest_shibor

## 插值得到给定的周期的无风险利率
def periodsSplineRiskFreeInterestRate(date, periods):
    # 此处使用SHIBOR来插值
    init_shibor = getHistDaySHIBOR(date)

    shibor = {}
    min_period = min(init_shibor.period.values)
    min_period = 10.0/360.0
    max_period = max(init_shibor.period.values)
    for p in periods.keys():
        tmp = periods[p]
        if periods[p] > max_period:
            tmp = max_period * 0.99999
        elif periods[p] < min_period:
            tmp = min_period * 1.00001
        sh = interpolate.spline(init_shibor.period.values, init_shibor.dataValue.values, [tmp], order=3)
        shibor[p] = sh[0]/100.0
    return shibor
```

1. Greeks 和 隐含波动率计算

本文中计算的Greeks包括：

- `delta` 期权价格关于标的价格的一阶导数
- `gamma` 期权价格关于标的价格的二阶导数
- `rho` 期权价格关于无风险利率的一阶导数
- `theta` 期权价格关于到期时间的一阶导数
- `vega` 期权价格关于波动率的一阶导数

注意：

- 计算隐含波动率，我们采用Black-Scholes-Merton模型，此模型在平台Python包CAL中已有实现
- 无风险利率使用SHIBOR
- 期权的时间价值为负时(此种情况在50ETF期权里时有发生)，没法通过BSM模型计算隐含波动率，故此时将期权隐含波动率设为0.0，实际上，此时的隐含波动率和各风险指标并无实际参考价值

```
## 使用DataAPI.OptGet, DataAPI.MktOptdGet拿到计算所需数据
def getOptDayData(opt_var_sec, date):
    date_str = date.toISO().replace('-', '')

    #使用DataAPI.OptGet，拿到已退市和上市的所有期权的基本信息
    info_fields = [u'optID', u'varSecID', u'varShortName', u'var
Ticker', u'varExchangeCD', u'varType',
                   u'contractType', u'strikePrice', u'contMultNu
m', u'contractStatus', u'listDate',
                   u'expYear', u'expMonth', u'expDate', u'lastTr
adeDate', u'exerDate', u'deliDate',
                   u'delistDate']
    opt_info = DataAPI.OptGet(optID='', contractStatus=[u"DE",u"
L"], field=info_fields, pandas="1")

    #使用DataAPI.MktOptdGet，拿到历史上某一天的期权成交信息
    mkt_fields = [u'ticker', u'optID', u'secShortName', u'exchan
geCD', u'tradeDate', u'preSettlePrice',
                  u'preClosePrice', u'openPrice', u'highestPrice'
, u'lowestPrice', u'closePrice',
                  u'settlPrice', u'turnoverVol', u'turnoverValue'
, u'openInt']
    opt_mkt = DataAPI.MktOptdGet(tradeDate=date_str, field=mkt_f
ields, pandas = "1")

    opt_info = opt_info.set_index(u"optID")
    opt_mkt = opt_mkt.set_index(u"optID")
    opt = concat([opt_info, opt_mkt], axis=1, join='inner').sort
_index()
    return opt

## 分析历史某一日的期权收盘价信息，得到隐含波动率微笑和期权风险指标
def getOptDayAnalysis(opt_var_sec, date):
    opt = getOptDayData(opt_var_sec, date)
    #使用DataAPI.MktFunddGet拿到期权标的的日行情
    date_str = date.toISO().replace('-', '')
    opt_var_mkt = DataAPI.MktFunddGet(secID=opt_var_sec,tradeDat
e=date_str,beginDate=u"",endDate=u"",field=u"",pandas="1")
    #opt_var_mkt = DataAPI.MktFunddAdjGet(secID=opt_var_sec,begi
nDate=date_str,endDate=date_str,field=u"",pandas="1")

    # 计算shibor
    exp_dates_str = opt.expDate.unique()
    periods = {}
    for date_str in exp_dates_str:
        exp_date = Date.parseISO(date_str)
        periods[exp_date] = (exp_date - date)/360.0
    shibor = periodsSplineRiskFreeInterestRate(date, periods)

    settle = opt.settlPrice.values         # 期权 settle price
    close = opt.closePrice.values          # 期权 close price
    strike = opt.strikePrice.values        # 期权 strike price
    option_type = opt.contractType.values  # 期权类型
    exp_date_str = opt.expDate.values      # 期权行权日期
    eval_date_str = opt.tradeDate.values   # 期权交易日期

    mat_dates = []
    eval_dates = []
    spot = []
    for epd, evd in zip(exp_date_str, eval_date_str):
        mat_dates.append(Date.parseISO(epd))
        eval_dates.append(Date.parseISO(evd))
        spot.append(opt_var_mkt.closePrice[0])
    time_to_maturity = [float(mat - eva + 1.0)/365.0 for (mat, e
va) in zip(mat_dates, eval_dates)]

    risk_free = []  # 无风险利率
    for s, mat, time in zip(spot, mat_dates, time_to_maturity):
        #rf = math.log(forward_price[mat] / s) / time
        rf = shibor[mat]
        risk_free.append(rf)

    opt_types = []   # 期权类型
    for t in option_type:
        if t == 'CO':
            opt_types.append(1)
        else:
            opt_types.append(-1)

    # 使用通联CAL包中 BSMImpliedVolatity 计算隐含波动率
    calculated_vol = BSMImpliedVolatity(opt_types, strike, spot,
 risk_free, 0.0, time_to_maturity, settle)
    calculated_vol = calculated_vol.fillna(0.0)

    # 使用通联CAL包中 BSMPrice 计算期权风险指标
    greeks = BSMPrice(opt_types, strike, spot, risk_free, 0.0, c
alculated_vol.vol.values, time_to_maturity)
    greeks.vega = greeks.vega #/ 100.0
    greeks.rho = greeks.rho #/ 100.0
    greeks.theta = greeks.theta #* 365.0 / 252.0 #/ 365.0

    opt['strike'] = strike
    opt['optType'] = option_type
    opt['expDate'] = exp_date_str
    opt['spotPrice'] = spot
    opt['riskFree'] = risk_free
    opt['timeToMaturity'] = np.around(time_to_maturity, decimals=4)
    opt['settle'] = np.around(greeks.price.values.astype(np.double), decimals=4)
    opt['iv'] = np.around(calculated_vol.vol.values.astype(np.double), decimals=4)
    opt['delta'] = np.around(greeks.delta.values.astype(np.double), decimals=4)
    opt['vega'] = np.around(greeks.vega.values.astype(np.double), decimals=4)
    opt['gamma'] = np.around(greeks.gamma.values.astype(np.double), decimals=4)
    opt['theta'] = np.around(greeks.theta.values.astype(np.double), decimals=4)
    opt['rho'] = np.around(greeks.rho.values.astype(np.double), decimals=4)

    fields = [u'ticker', u'contractType', u'strikePrice', u'expDate', u'tradeDate',
              u'closePrice', u'settlPrice', 'spotPrice', u'iv',
              u'delta', u'vega', u'gamma', u'theta',  u'rho']
    opt = opt[fields].reset_index().set_index('ticker').sort_index()
    #opt['iv'] = opt.iv.replace(to_replace=0.0, value=np.nan)
    return opt
```

尝试用 `getOptDayAnalysis` 计算 2015-09-24 这一天的风险指标

```
# Uqer 计算期权的风险数据
opt_var_sec = u"510050.XSHG"    # 期权标的
date = Date(2015, 9, 24)

option_risk = getOptDayAnalysis(opt_var_sec, date)
option_risk.head(2)
```

	optID	contractType	strikePrice	expDate
ticker				
510050C1510M01850	10000405	CO	1.85	2015-10-28
510050C1510M01900	10000406	CO	1.90	2015-10-28

进一步，我们和上交所给出的对应日期的风险指标参考数据对比一下

- 上交所的数据需要自行下载，注意选择日期下载相应csv文件，http://www.sse.com.cn/assortment/derivatives/options/risk/
- 下载完后，不做内容改动，请上传到UQER平台的 Data 中；文件名请相应修改，此处我设为了 `option_risk_sse_0924.csv`
- 为了避免冗余，下面我们仅仅对比近月期权的各个风险指标

```
# 读取上交所数据
def readRiskDataSSE(file_str):
    # 按照上交所下载到的risk数据排版格式，做以处理
    opt = pd.read_csv(file_str, encoding='gb2312').reset_index()
    opt.columns = [['tradeDate','optID','ticker','secShortName',
'delta','theta','gamma','vega','rho','margin']]
    opt = opt[['tradeDate','optID','ticker','delta','theta','gam
ma','vega','rho']]
    opt['ticker'] = [tic[1:-2] for tic in opt['ticker']]
    opt['tradeDate'] = [td[0:-1] for td in opt['tradeDate']]

    #使用DataAPI.OptGet，拿到已退市和上市的所有期权的基本信息
    info_fields = [u'optID', u'varSecID', u'varShortName', u'var
Ticker', u'varExchangeCD', u'varType',
                   u'contractType', u'strikePrice', u'contMultNu
m', u'contractStatus', u'listDate',
                   u'expYear', u'expMonth', u'expDate', u'lastTr
adeDate', u'exerDate', u'deliDate',
                   u'delistDate']
    opt_info = DataAPI.OptGet(optID='', contractStatus=[u"DE",u"
L"], field=info_fields, pandas="1")

    # 上交所的数据和期权基本信息合并，得到比较完整的期权数据
    opt_info = opt_info.set_index(u"optID")
    opt = opt.set_index(u"optID")
    opt = concat([opt_info, opt], axis=1, join='inner').sort_ind
ex()

    fields = [u'ticker', u'contractType', u'strikePrice', u'expD
ate', u'tradeDate',
              u'delta', u'vega', u'gamma', u'theta',  u'rho']
    opt = opt[fields].reset_index().set_index('ticker').sort_ind
ex()
    return opt
```

读取 2015-09-24 上交所数据

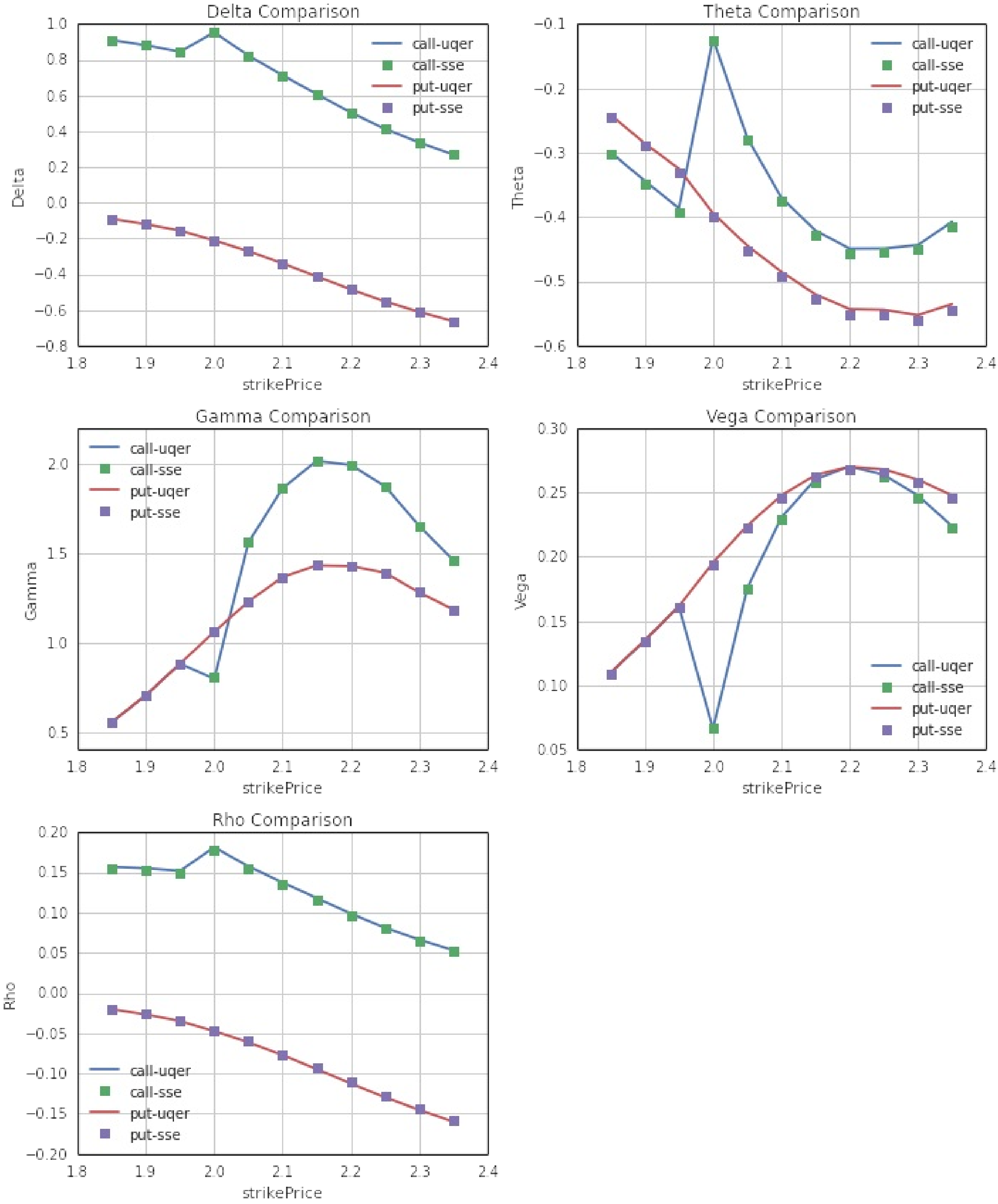
```
option_risk_sse = readRiskDataSSE('option_risk_sse_0924.csv')
option_risk_sse.head(2)
```

	optID	contractType	strikePrice	expDate
ticker				
510050C1510M01850	10000405	CO	1.85	2015-10-28
510050C1510M01900	10000406	CO	1.90	2015-10-28

`getOptDayAnalysis` 函数计算结果和上交所数据的对比

```
# 对比本文计算结果 option_risk 和上交所结果 option_risk_sse 中的近月期权风险指标

near_exp = np.sort(option_risk.expDate.unique())[0]    # 近月期权行权日

opt_call_uqer = option_risk[option_risk.expDate==near_exp][option_risk.contractType=='CO'].set_index('strikePrice')
opt_call_sse = option_risk_sse[option_risk_sse.expDate==near_exp][option_risk_sse.contractType=='CO'].set_index('strikePrice')
opt_put_uqer = option_risk[option_risk.expDate==near_exp][option_risk.contractType=='PO'].set_index('strikePrice')
opt_put_sse = option_risk_sse[option_risk_sse.expDate==near_exp][option_risk_sse.contractType=='PO'].set_index('strikePrice')

## ----------------------------------------------
## 风险指标对比
fig = plt.figure(figsize=(10,12))
fig.set_tight_layout(True)

# ------ Delta ------
ax = fig.add_subplot(321)
ax.plot(opt_call_uqer.index, opt_call_uqer['delta'], '-')
ax.plot(opt_call_sse.index, opt_call_sse['delta'], 's')
ax.plot(opt_put_uqer.index, opt_put_uqer['delta'], '-')
ax.plot(opt_put_sse.index, opt_put_sse['delta'], 's')
ax.legend(['call-uqer', 'call-sse', 'put-uqer', 'put-sse'])
ax.grid()
ax.set_xlabel(u"strikePrice")
ax.set_ylabel(r"Delta")
plt.title('Delta Comparison')

# ------ Theta ------
ax = fig.add_subplot(322)
ax.plot(opt_call_uqer.index, opt_call_uqer['theta'], '-')
ax.plot(opt_call_sse.index, opt_call_sse['theta'], 's')
ax.plot(opt_put_uqer.index, opt_put_uqer['theta'], '-')
ax.plot(opt_put_sse.index, opt_put_sse['theta'], 's')
ax.legend(['call-uqer', 'call-sse', 'put-uqer', 'put-sse'])
ax.grid()
ax.set_xlabel(u"strikePrice")
ax.set_ylabel(r"Theta")
plt.title('Theta Comparison')

# ------ Gamma ------
ax = fig.add_subplot(323)
ax.plot(opt_call_uqer.index, opt_call_uqer['gamma'], '-')
ax.plot(opt_call_sse.index, opt_call_sse['gamma'], 's')
ax.plot(opt_put_uqer.index, opt_put_uqer['gamma'], '-')
ax.plot(opt_put_sse.index, opt_put_sse['gamma'], 's')
ax.legend(['call-uqer', 'call-sse', 'put-uqer', 'put-sse'], loc=0
)
ax.grid()
ax.set_xlabel(u"strikePrice")
ax.set_ylabel(r"Gamma")
plt.title('Gamma Comparison')

# # ------ Vega ------
ax = fig.add_subplot(324)
ax.plot(opt_call_uqer.index, opt_call_uqer['vega'], '-')
ax.plot(opt_call_sse.index, opt_call_sse['vega'], 's')
ax.plot(opt_put_uqer.index, opt_put_uqer['vega'], '-')
ax.plot(opt_put_sse.index, opt_put_sse['vega'], 's')
ax.legend(['call-uqer', 'call-sse', 'put-uqer', 'put-sse'], loc=4
)
ax.grid()
ax.set_xlabel(u"strikePrice")
ax.set_ylabel(r"Vega")
plt.title('Vega Comparison')

# ------ Rho ------
ax = fig.add_subplot(325)
ax.plot(opt_call_uqer.index, opt_call_uqer['rho'], '-')
ax.plot(opt_call_sse.index, opt_call_sse['rho'], 's')
ax.plot(opt_put_uqer.index, opt_put_uqer['rho'], '-')
ax.plot(opt_put_sse.index, opt_put_sse['rho'], 's')
ax.legend(['call-uqer', 'call-sse', 'put-uqer', 'put-sse'], loc=3
)
ax.grid()
ax.set_xlabel(u"strikePrice")
ax.set_ylabel(r"Rho")
plt.title('Rho Comparison')
```

```
<matplotlib.text.Text at 0x535d0d0>
```



上述五张图中，对于近月期权，我们分别对比了五个Greeks风险指标： `Delta` ， `Theta` ， `Gamma` ， `Vega` ， `Rho` ：

- 每张图中， `Call` 和 `Put` 分开比较，横轴为行权价
- 可以看出，本文中的计算结果和上交所的参考数值符合的比较好
- 在接下来的50ETF期权分析中，我们将使用本文中的计算方法来计算期权隐含波动率和Greeks风险指标

把上面的数据整理整理，格式更简洁一点

```
# 每日期权分析数据整理
def getOptDayGreeksIV(date):
    # Uqer 计算期权的风险数据
    opt_var_sec = u"510050.XSHG"    # 期权标的
    opt = getOptDayAnalysis(opt_var_sec, date)

    # 整理数据部分
    opt.index = [index[-10:] for index in opt.index]
    opt = opt[['contractType','strikePrice','expDate','closePrice','iv','delta','theta','gamma','vega','rho']]
    opt_call = opt[opt.contractType=='CO']
    opt_put = opt[opt.contractType=='PO']
    opt_call.columns = pd.MultiIndex.from_tuples([('Call', c) for c in opt_call.columns])
    opt_call[('Call-Put', 'strikePrice')] = opt_call[('Call', 'strikePrice')]
    opt_put.columns = pd.MultiIndex.from_tuples([('Put', c) for c in opt_put.columns])
    opt = concat([opt_call, opt_put], axis=1, join='inner').sort_index()
    opt = opt.set_index(('Call','expDate')).sort_index()
    opt = opt.drop([('Call','contractType'), ('Call','strikePrice')], axis=1)
    opt = opt.drop([('Put','expDate'), ('Put','contractType'), ('Put','strikePrice')], axis=1)
    opt.index.name = 'expDate'
    ## 以上得到完整的历史某日数据，格式简洁明了
    return opt
```

```
date = Date(2015, 9, 24)

option_risk = getOptDayGreeksIV(date)
option_risk.head(10)
```

	Call	Call-Put	Put				
closePrice	iv	delta	theta	gamma	vega	rho	strik
expDate							
2015-10-28	0.3268	0.4317	0.9101	-0.2992	0.5550	0.1099	0.15
2015-10-28	0.2791	0.4161	0.8810	-0.3435	0.7058	0.1347	0.15
2015-10-28	0.2360	0.3990	0.8449	-0.3862	0.8823	0.1615	0.15
2015-10-28	0.1955	0.1811	0.9532	-0.1225	0.7980	0.0663	0.18
2015-10-28	0.1599	0.2453	0.8237	-0.2764	1.5588	0.1754	0.15
2015-10-28	0.1275	0.2698	0.7137	-0.3696	1.8625	0.2304	0.13
2015-10-28	0.0990	0.2814	0.6081	-0.4208	2.0162	0.2602	0.11
2015-10-28	0.0768	0.2955	0.5057	-0.4489	1.9934	0.2701	0.09
2015-10-28	0.0584	0.3068	0.4132	-0.4487	1.8746	0.2637	0.08
2015-10-28	0.0470	0.3264	0.3381	-0.4434	1.6538	0.2476	0.06

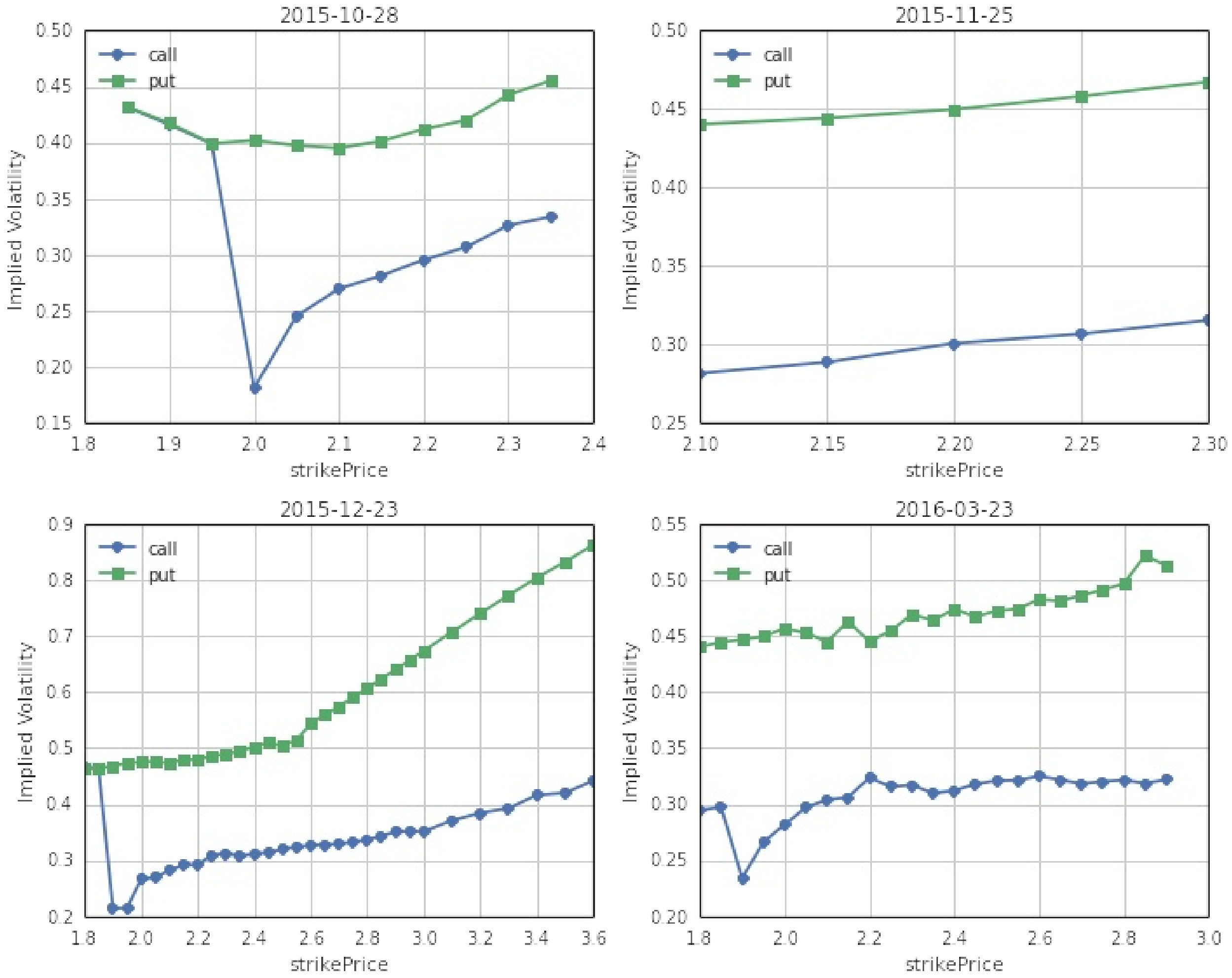
2. 隐含波动率微笑

利用上一小节的代码，给出隐含波动率微笑结构

隐含波动率微笑

```
# 做图展示某一天的隐含波动率微笑
def plotSmileVolatility(date):
    # Uqer 计算期权的风险数据
    opt = getOptDayGreeksIV(date)

    # 下面展示波动率微笑
    exp_dates = np.sort(opt.index.unique())
    ## ----------------------------------------------
    fig = plt.figure(figsize=(10,8))
    fig.set_tight_layout(True)

    for i in range(exp_dates.shape[0]):
        date = exp_dates[i]
        ax = fig.add_subplot(2,2,i+1)
        opt_date = opt[opt.index==date].set_index(('Call-Put', '
strikePrice'))
        opt_date.index.name = 'strikePrice'

        ax.plot(opt_date.index, opt_date[('Call', 'iv')], '-o')
        ax.plot(opt_date.index, opt_date[('Put', 'iv')], '-s')
        ax.legend(['call', 'put'], loc=0)
        ax.grid()
        ax.set_xlabel(u"strikePrice")
        ax.set_ylabel(r"Implied Volatility")
        plt.title(exp_dates[i])
```

```
plotSmileVolatility(Date(2015,9,24))
```

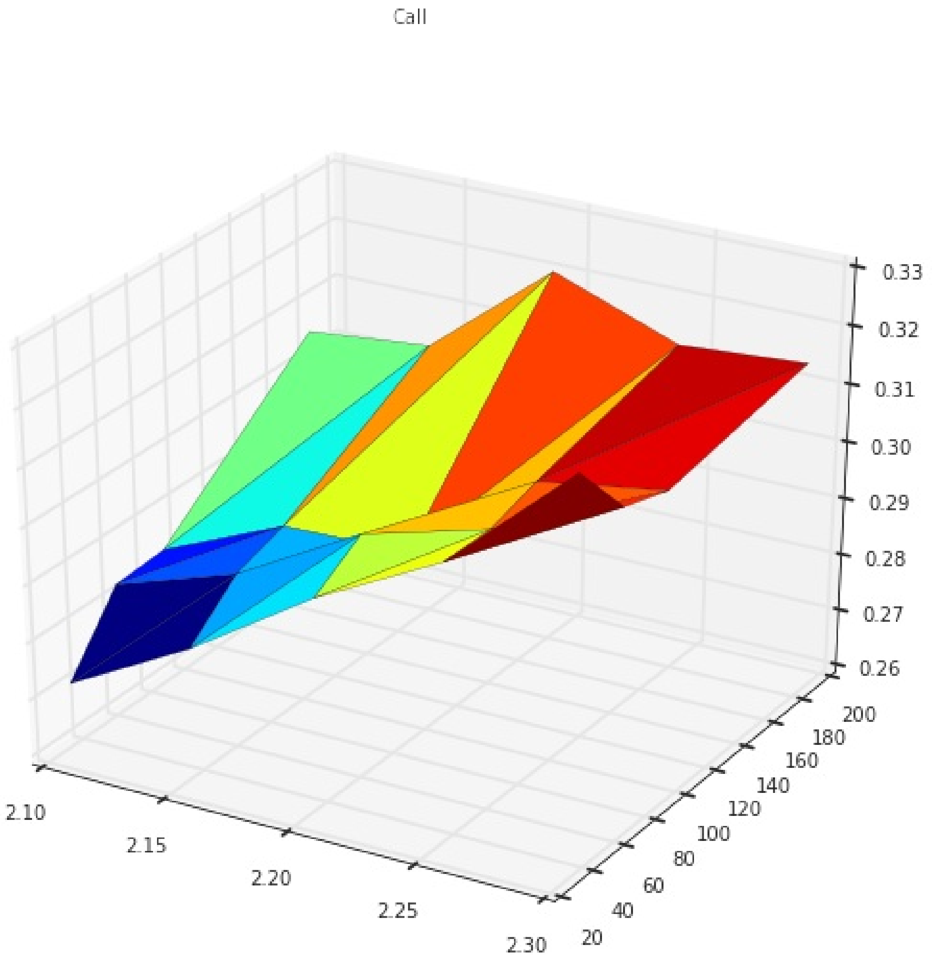


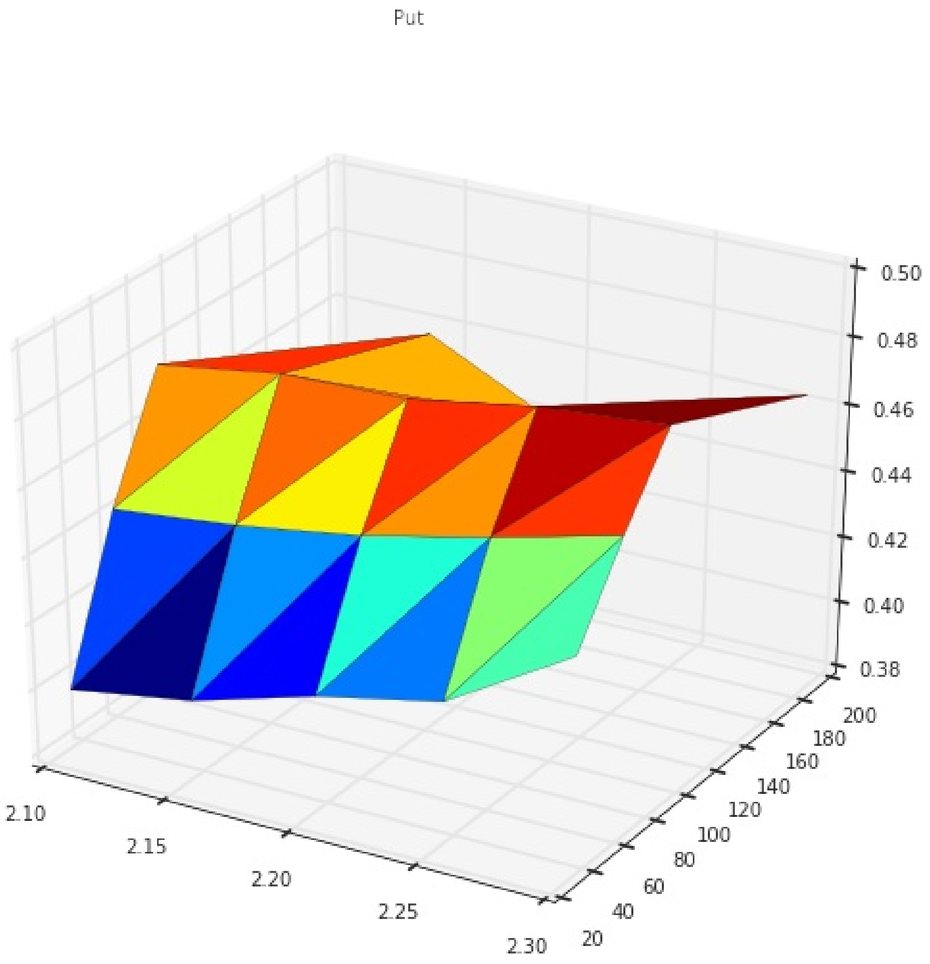
行权价和行权日期两个方向上的隐含波动率微笑

```
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm

# 做图展示某一天的隐含波动率结构
def plotSmileVolatilitySurface(date):
    # Uqer 计算期权的风险数据
    opt = getOptDayGreeksIV(date)

    # 下面展示波动率结构
    exp_dates = np.sort(opt.index.unique())
    strikes = np.sort(opt[('Call-Put', 'strikePrice')].unique())
    risk_mt = {'Call': pd.DataFrame(index=strikes),
               'Put': pd.DataFrame(index=strikes) }

    # 将数据整理成Call和Put分开来，分别的结构为：
    # 行为行权价，列为剩余到期天数（以自然天数计算）
    for epd in exp_dates:
        exp_days = Date.parseISO(epd) - date
        opt_date = opt[opt.index==epd].set_index(('Call-Put', 's
trikePrice'))
        opt_date.index.name = 'strikePrice'
        for cp in risk_mt.keys():
            risk_mt[cp][exp_days] = opt_date[(cp, 'iv')]
    for cp in risk_mt.keys():
        for strike in risk_mt[cp].index:
            if np.sum(np.isnan(risk_mt[cp].ix[strike])) > 0:
                risk_mt[cp] = risk_mt[cp].drop(strike)

    # Call和Put分开显示，行index为行权价，列index为剩余到期天数
    #print risk_mt

    # 画图
    for cp in ['Call', 'Put']:
        opt = risk_mt[cp]
        x = []
        y = []
        z = []
        for xx in opt.index:
            for yy in opt.columns:
                x.append(xx)
                y.append(yy)
                z.append(opt[yy][xx])
        fig = plt.figure(figsize=(10,8))
        fig.suptitle(cp)
        ax = fig.gca(projection='3d')
        ax.plot_trisurf(x, y, z, cmap=cm.jet, linewidth=0.2)
    return risk_mt
```

画出某一天的波动率微笑曲面结构

```
opt = plotSmileVolatilitySurface(Date(2015,9,24))
opt  # Call和Put分开显示，行index为行权价，列index为剩余到期天数

{'Call':          34      62      90      181
 2.10  0.2698  0.2817  0.2823  0.3042
 2.15  0.2814  0.2888  0.2916  0.3063
 2.20  0.2955  0.3008  0.2922  0.3237
 2.25  0.3068  0.3067  0.3093  0.3157
 2.30  0.3264  0.3155  0.3128  0.3172,
 'Put':          34      62      90      181
 2.10  0.3952  0.4403  0.4740  0.4449
 2.15  0.4013  0.4442  0.4794  0.4632
 2.20  0.4121  0.4498  0.4802  0.4451
 2.25  0.4200  0.4581  0.4863  0.4547
 2.30  0.4426  0.4673  0.4893  0.4691}
```





波动率曲面结构图中：

- 上图为Call，下图为Put，此处没有进行任何插值处理，所以略显粗糙
- Put的隐含波动率明显大于Call
- 期限结构来说，波动率呈现远高近低的特征

【50ETF期权】 5. 日内即时监控 Greeks 和隐含波动率微笑

> 来源：https://uqer.io/community/share/5615d10ff9f06c4ca72fb5be

和上一篇类似，计算上证50ETF期权的隐含波动率微笑，但这里通过DataAPI提供的高频数据，在交易时间实时监控Greeks 和隐含波动率变化！

```
from CAL.PyCAL import *
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import rc
rc('mathtext', default='regular')
import seaborn as sns
sns.set_style('white')
import math
from scipy import interpolate
from scipy.stats import mstats
from pandas import Series, DataFrame, concat
import time
from matplotlib import dates
```

上海银行间同业拆借利率 SHIBOR，用来作为无风险利率参考

```
## 银行间质押式回购利率
def getHistDayInterestRateInterbankRepo(date):
    cal = Calendar('China.SSE')
    period = Period('-10B')
    begin = cal.advanceDate(date, period)
    begin_str = begin.toISO().replace('-', '')
    date_str = date.toISO().replace('-', '')
    # 以下的indicID分别对应的银行间质押式回购利率周期为：
    # 1D, 7D, 14D, 21D, 1M, 3M, 4M, 6M, 9M, 1Y
    indicID = [u"M120000067", u"M120000068", u"M120000069", u"M120000070", u"M120000071",
               u"M120000072", u"M120000073", u"M120000074", u"M120000075", u"M120000076"]
    period = np.asarray([1.0, 7.0, 14.0, 21.0, 30.0, 90.0, 120.0, 180.0, 270.0, 360.0]) / 360.0
    period_matrix = pd.DataFrame(index=indicID, data=period, columns=['period'])
    field = u"indicID,indicName,publishTime,periodDate,dataValue,unit"
    interbank_repo = DataAPI.ChinaDataInterestRateInterbankRepoGet(indicID=indicID,beginDate=begin_str,endDate=date_str,field=field,pandas="1")
    interbank_repo = interbank_repo.groupby('indicID').first()
    interbank_repo = concat([interbank_repo, period_matrix], axis=1, join='inner').sort_index()
    return interbank_repo

## 银行间同业拆借利率
def getHistDaySHIBOR(date):
    cal = Calendar('China.SSE')
    period = Period('-10B')
    begin = cal.advanceDate(date, period)
    begin_str = begin.toISO().replace('-', '')
    date_str = date.toISO().replace('-', '')
    # 以下的indicID分别对应的SHIBOR周期为：
    # 1D, 7D, 14D, 1M, 3M, 6M, 9M, 1Y
    indicID = [u"M120000057", u"M120000058", u"M120000059", u"M120000060",
               u"M120000061", u"M120000062", u"M120000063", u"M120000064"]
    period = np.asarray([1.0, 7.0, 14.0, 30.0, 90.0, 180.0, 270.0, 360.0]) / 360.0
    period_matrix = pd.DataFrame(index=indicID, data=period, columns=['period'])
    field = u"indicID,indicName,publishTime,periodDate,dataValue,unit"
    interest_shibor = DataAPI.ChinaDataInterestRateSHIBORGet(indicID=indicID,beginDate=begin_str,endDate=date_str,field=field,pandas="1")
    interest_shibor = interest_shibor.groupby('indicID').first()
    interest_shibor = concat([interest_shibor, period_matrix], axis=1, join='inner').sort_index()
    return interest_shibor

## 插值得到给定的周期的无风险利率
def periodsSplineRiskFreeInterestRate(date, periods):
    # 此处使用SHIBOR来插值
    init_shibor = getHistDaySHIBOR(date)

    shibor = {}
    min_period = min(init_shibor.period.values)
    min_period = 10.0/360.0
    max_period = max(init_shibor.period.values)
    for p in periods.keys():
        tmp = periods[p]
        if periods[p] > max_period:
            tmp = max_period * 0.99999
        elif periods[p] < min_period:
            tmp = min_period * 1.00001
        sh = interpolate.spline(init_shibor.period.values, init_shibor.dataValue.values, [tmp], order=3)
        shibor[p] = sh[0]/100.0
    return shibor
```

1. Greeks 和 隐含波动率

本文中计算的Greeks包括：

- `delta` 期权价格关于标的价格的一阶导数
- `gamma` 期权价格关于标的价格的二阶导数
- `rho` 期权价格关于无风险利率的一阶导数
- `theta` 期权价格关于到期时间的一阶导数
- `vega` 期权价格关于波动率的一阶导数

注意：

- 计算隐含波动率，我们采用Black-Scholes-Merton模型，此模型在平台Python包CAL中已有实现
- 无风险利率使用SHIBOR
- 期权的时间价值为负时(此种情况在50ETF期权里时有发生)，没法通过BSM模型计算隐含波动率，故此时将期权隐含波动率设为0.0，实际上，此时的隐含波动率和各风险指标并无实际参考价值
- 此处计算即时隐含波动率和Greeks时候，我们使用买一和卖一的中间价作为期权价格

```
## 使用DataAPI.OptGet, DataAPI.MktOptionTickRTSnapshotGet 拿到计算所需数据
def getOptTickSnapshot(opt_var_sec, date):
    date_str = date.toISO().replace('-', '')

    #使用DataAPI.OptGet，拿到已退市和上市的所有期权的基本信息
    info_fields = [u'optID', u'varSecID', u'varShortName', u'varTicker', u'varExchangeCD', u'varType',
                   u'contractType', u'strikePrice', u'contMultNum', u'contractStatus', u'listDate',
                   u'expYear', u'expMonth', u'expDate', u'lastTradeDate', u'exerDate', u'deliDate',
                   u'delistDate']
    opt_info = DataAPI.OptGet(optID='', contractStatus=[u"DE",u"L"], field=info_fields, pandas="1")

    #使用DataAPI.MktOptionTickRTSnapshotGet，拿到期权实时成交数据
    date_str = date.toISO().replace('-', '')
    fields_mkt = ['instrumentID', u'optionId', u'dataDate', u'lastPrice', u'preSettlePrice',
                  u'bidBook_price1', u'bidBook_volume1', u'askBook_price1', u'askBook_volume1']
    opt_mkt = DataAPI.MktOptionTickRTSnapshotGet(optionId=u"", field='', pandas="1")
    opt_mkt = opt_mkt[opt_mkt.dataDate == date.toISO()]
    opt_mkt['optID'] = map(int, opt_mkt['optionId'])
    opt_mkt[u"price"] = (opt_mkt['bidBook_price1'] + opt_mkt['askBook_price1'])/2.0
    opt_info = opt_info.set_index(u"optID")
    opt_mkt = opt_mkt.set_index(u"optID")
    opt = concat([opt_info, opt_mkt], axis=1, join='inner').sort_index()
    return opt

## 分析期权即时交易信息，得到隐含波动率微笑和期权风险指标
def getOptAnalysisSnapshot(opt_var_sec):
    date = Date.todaysDate()
    opt = getOptTickSnapshot(opt_var_sec, date)

    #使用DataAPI.MktTickRTSnapshotGet 拿到期权标的的即时行情
    date_str = date.toISO().replace('-', '')
    opt_var_mkt = DataAPI.MktTickRTSnapshotGet(securityID=opt_var_sec,field=u"lastPrice",pandas="1")

    # 计算shibor
    exp_dates_str = opt.expDate.unique()
    periods = {}
    for date_str in exp_dates_str:
        exp_date = Date.parseISO(date_str)
        periods[exp_date] = (exp_date - date)/360.0
    shibor = periodsSplineRiskFreeInterestRate(date, periods)

    settle = opt.price.values         # 期权 settle price
    close = opt.price.values          # 期权 close price
    strike = opt.strikePrice.values        # 期权 strike price
    option_type = opt.contractType.values  # 期权类型
    exp_date_str = opt.expDate.values      # 期权行权日期
    eval_date_str = opt.dataDate.values   # 期权交易日期

    mat_dates = []
    eval_dates = []
    spot = []
    for epd, evd in zip(exp_date_str, eval_date_str):
        mat_dates.append(Date.parseISO(epd))
        eval_dates.append(Date.parseISO(evd))
        spot.append(opt_var_mkt.lastPrice[0])
    time_to_maturity = [float(mat - eva + 1.0)/365.0 for (mat, eva) in zip(mat_dates, eval_dates)]

    risk_free = []  # 无风险利率
    for s, mat, time in zip(spot, mat_dates, time_to_maturity):
        #rf = math.log(forward_price[mat] / s) / time
        rf = shibor[mat]
        risk_free.append(rf)

    opt_types = []   # 期权类型
    for t in option_type:
        if t == 'CO':
            opt_types.append(1)
        else:
            opt_types.append(-1)
    # 使用通联CAL包中 BSMImpliedVolatity 计算隐含波动率
    calculated_vol = BSMImpliedVolatity(opt_types, strike, spot,
 risk_free, 0.0, time_to_maturity, settle)
    calculated_vol = calculated_vol.fillna(0.0)

    # 使用通联CAL包中 BSMPrice 计算期权风险指标
    greeks = BSMPrice(opt_types, strike, spot, risk_free, 0.0, c
alculated_vol.vol.values, time_to_maturity)
    # vega、rho、theta 的计量单位参照上交所的数据，以求统一对比
    greeks.vega = greeks.vega #/ 100.0
    greeks.rho = greeks.rho #/ 100.0
    greeks.theta = greeks.theta #* 365.0 / 252.0 #/ 365.0

    opt['strike'] = strike
    opt['optType'] = option_type
    opt['expDate'] = exp_date_str
    opt['spotPrice'] = spot
    opt['riskFree'] = risk_free
    opt['timeToMaturity'] = np.around(time_to_maturity, decimals=
4)
    opt['settle'] = np.around(greeks.price.values.astype(np.doub
le), decimals=4)
    opt['iv'] = np.around(calculated_vol.vol.values.astype(np.do
uble), decimals=4)
    opt['delta'] = np.around(greeks.delta.values.astype(np.doubl
e), decimals=4)
    opt['vega'] = np.around(greeks.vega.values.astype(np.double)
, decimals=4)
    opt['gamma'] = np.around(greeks.gamma.values.astype(np.doubl
e), decimals=4)
    opt['theta'] = np.around(greeks.theta.values.astype(np.doubl
e), decimals=4)
    opt['rho'] = np.around(greeks.rho.values.astype(np.double),
decimals=4)

    fields = [u'instrumentID', u'contractType', u'strikePrice',
u'expDate', u'dataDate', u'dataTime',
              u'price', 'spotPrice', u'iv',
              u'delta', u'vega', u'gamma', u'theta',  u'rho']
    opt = opt[fields].reset_index().set_index('instrumentID').so
rt_index()
    #opt['iv'] = opt.iv.replace(to_replace=0.0, value=np.nan)
    return opt

# 期权分析数据整理
def getOptSnapshotGreeksIV():
    # Uqer 计算期权的风险数据
    opt_var_sec = u"510050.XSHG"    # 期权标的
    opt = getOptAnalysisSnapshot(opt_var_sec)

    # 整理数据部分
    opt.index = [index[-10:] for index in opt.index]
    opt = opt[['spotPrice', 'contractType','strikePrice','expDat
e','price','iv','delta','theta','gamma','vega','rho']]
    opt_call = opt[opt.contractType=='CO']
    opt_put = opt[opt.contractType=='PO']
    opt_call.columns = pd.MultiIndex.from_tuples([('Call', c) for
 c in opt_call.columns])
    opt_call[('Call-Put', 'spotPrice')] = opt_call[('Call', 'spo
tPrice')]
    opt_call[('Call-Put', 'strikePrice')] = opt_call[('Call', 's
trikePrice')]
    opt_put.columns = pd.MultiIndex.from_tuples([('Put', c) for
c in opt_put.columns])
    opt = concat([opt_call, opt_put], axis=1, join='inner').sort
_index()
    opt = opt.set_index(('Call','expDate')).sort_index()
    opt = opt.drop([('Call','contractType'), ('Call','strikePric
e'), ('Call','spotPrice')], axis=1)
    opt = opt.drop([('Put','expDate'), ('Put','contractType'), (
'Put','strikePrice'), ('Put','spotPrice')], axis=1)
    opt.index.name = 'expDate'
    ## 以上得到完整的历史某日数据，格式简洁明了
    return opt
```

即时风险数据计算，其中的 `price` 就是买一、卖一价格的平均

```
DataAPI.MktTickRTSnapshotGet(securityID="510050.XSHG",field=u"",
pandas="1").columns
```

```
Index([u'exchangeCD', u'ticker', u'timestamp', u'AggQty', u'ampl
itude', u'change', u'changePct', u'currencyCD', u'dataDate', u'd
ataTime', u'deal', u'extraLargeOrderValue', u'highPrice', u'IEP'
, u'largeOrderValue', u'lastPrice', u'localTimestamp', u'lowPric
e', u'mediumOrderValue', u'negMarketValue', u'nominalPrice', u'o
penPrice', u'orderType', u'prevClosePrice', u'shortNM', u'smallO
rderValue', u'staticPE', u'suspension', u'totalOrderValue', u'tr
adSessionID', u'tradSessionStatus', u'tradSessionSubID', u'tradS
tatus', u'tradType', u'turnoverRate', u'utcOffset', u'value', u'
volume', u'VWAP', u'Yield', u'bidBook_price1', u'bidBook_volume1'
, u'bidBook_price2', u'bidBook_volume2', u'bidBook_price3', u'bi
dBook_volume3', u'bidBook_price4', u'bidBook_volume4', u'bidBook
_price5', u'bidBook_volume5', u'askBook_price1', u'askBook_volum
e1', u'askBook_price2', u'askBook_volume2', u'askBook_price3', u
'askBook_volume3', u'askBook_price4', u'askBook_volume4', u'askB
ook_price5', u'askBook_volume5'], dtype='object')
```

```
opt = getOptSnapshotGreeksIV()
opt.head(20)
```

	Call	Call-Put	Put				
price	iv	delta	theta	gamma	vega	rho	spotP
expDate							
2015-10-28	0.35250	0.0000	NaN	NaN	NaN	NaN	NaN
2015-10-28	0.30390	0.0000	NaN	NaN	NaN	NaN	NaN
2015-10-28	0.25710	0.0000	NaN	NaN	NaN	NaN	NaN
2015-10-28	0.20990	0.0000	NaN	NaN	NaN	NaN	NaN
2015-10-28	0.16690	0.0000	NaN	NaN	NaN	NaN	NaN
2015-10-28	0.12500	0.1998	0.8793	-0.2390	1.8916	0.1067	0.104
2015-10-28	0.09155	0.2392	0.7182	-0.4171	2.6574	0.1794	0.086
2015-10-28	0.06285	0.2510	0.5679	-0.4908	2.9485	0.2089	0.068
2015-10-28	0.04160	0.2615	0.4241	-0.4994	2.8189	0.2081	0.051
2015-10-28	0.02795	0.2780	0.3064	-0.4697	2.3766	0.1865	0.037
2015-10-28	0.01655	0.2793	0.2049	-0.3791	1.9141	0.1509	0.025
2015-11-25	0.17915	0.1663	0.9149	-0.1371	1.1545	0.1264	0.248
2015-11-25	0.14805	0.2196	0.7751	-0.2490	1.6819	0.2433	0.210
2015-11-25	0.12025	0.2438	0.6649	-0.3116	1.8413	0.2957	0.181
2015-11-25	0.09455	0.2541	0.5657	-0.3391	1.9085	0.3194	0.155
2015-11-25	0.07355	0.2631	0.4721	-0.3475	1.8635	0.3230	0.130

2015-11-25	0.05525	0.2667	0.3849	-0.3335	1.7659	0.3102	0.107
2015-12-23	0.38610	0.0000	NaN	NaN	NaN	NaN	NaN
2015-12-23	0.34710	0.0000	NaN	NaN	NaN	NaN	NaN
2015-12-23	0.30415	0.0000	NaN	NaN	NaN	NaN	NaN

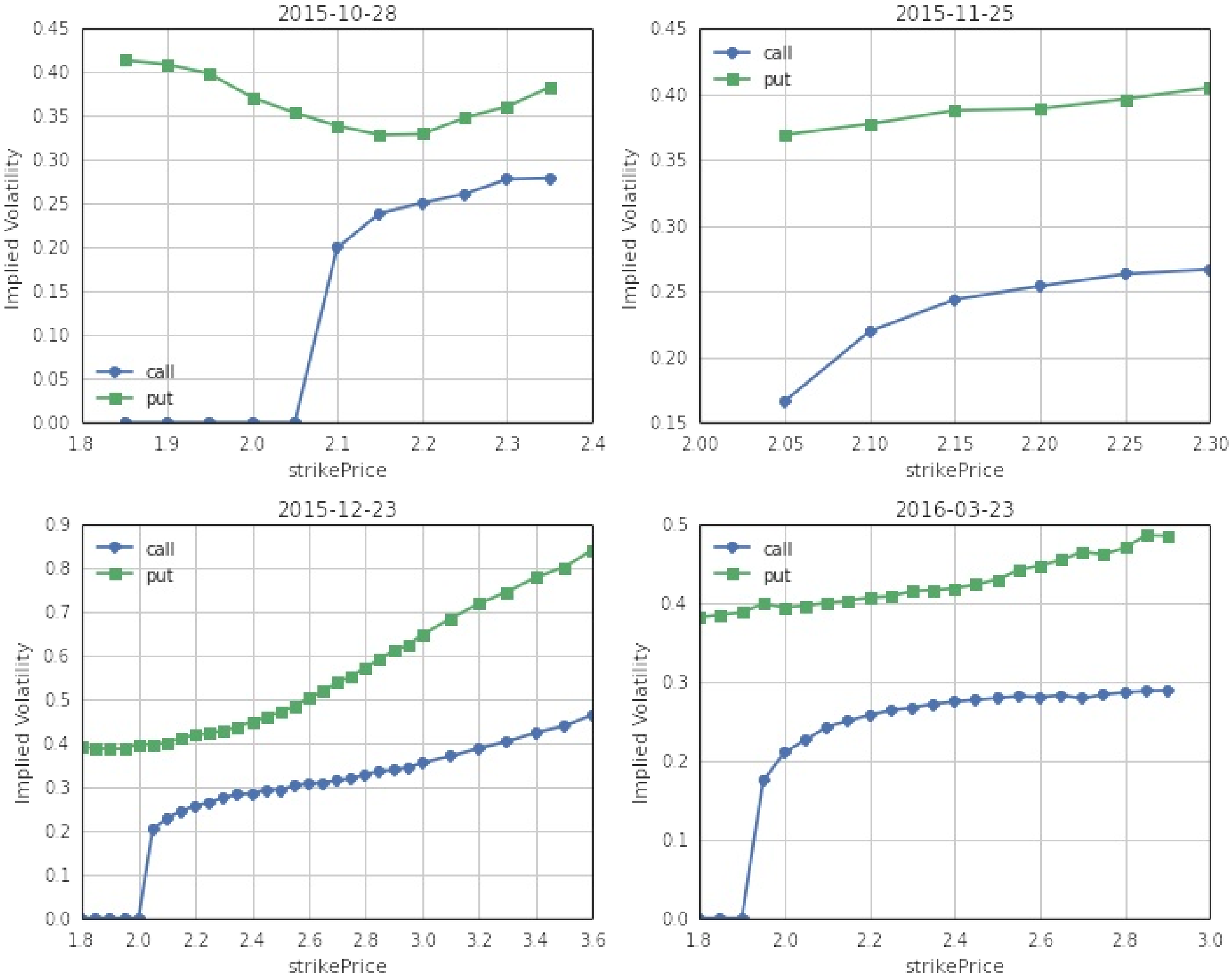
2. 隐含波动率微笑

利用上一小节的代码，给出隐含波动率微笑结构

隐含波动率微笑

```
# 做图展示某一天的隐含波动率微笑
def plotSnapshotSmileVolatility():
    # Uqer 计算期权的风险数据
    opt = getOptSnapshotGreeksIV()

    # 下面展示波动率微笑
    exp_dates = np.sort(opt.index.unique())
    ## ----------------------------------------------
    fig = plt.figure(figsize=(10,8))
    fig.set_tight_layout(True)

    for i in range(exp_dates.shape[0]):
        date = exp_dates[i]
        ax = fig.add_subplot(2,2,i+1)
        opt_date = opt[opt.index==date].set_index(('Call-Put', 'strikePrice'))
        opt_date.index.name = 'strikePrice'

        ax.plot(opt_date.index, opt_date[('Call', 'iv')], '-o')
        ax.plot(opt_date.index, opt_date[('Put', 'iv')], '-s')
        ax.legend(['call', 'put'], loc=0)
        ax.grid()
        ax.set_xlabel(u"strikePrice")
        ax.set_ylabel(r"Implied Volatility")
        plt.title(exp_dates[i])
```

```
plotSnapshotSmileVolatility()
```

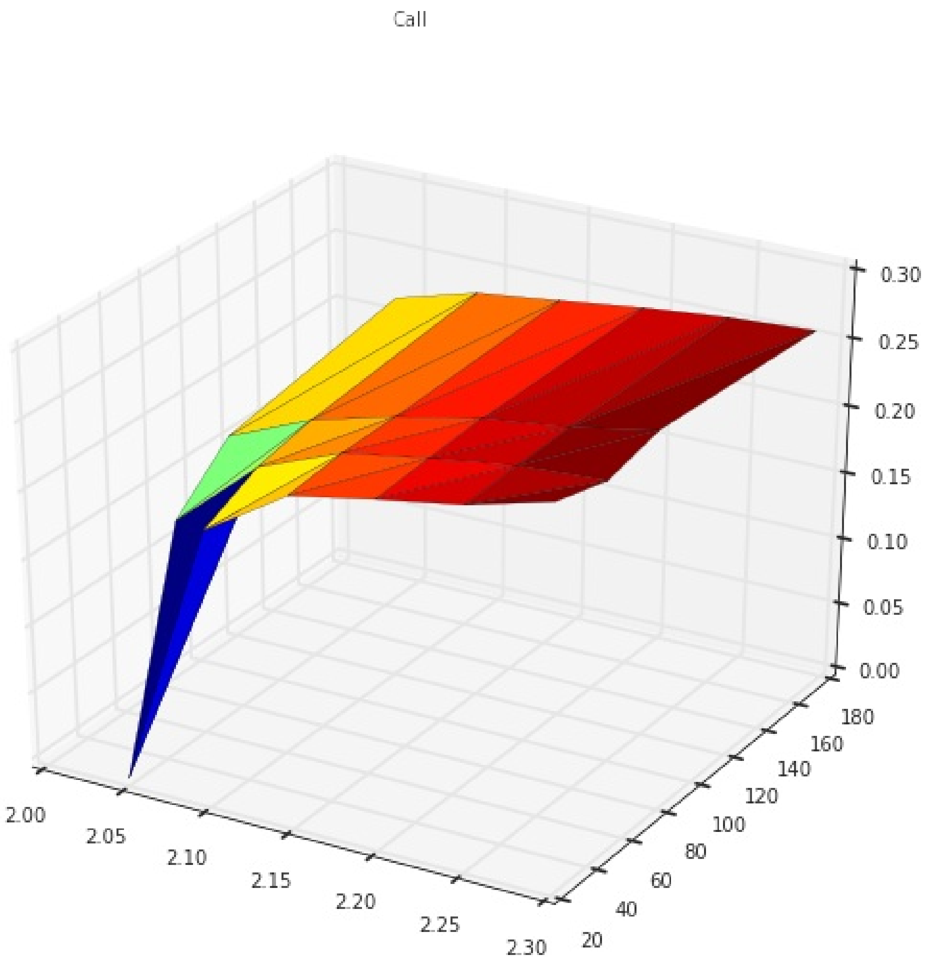


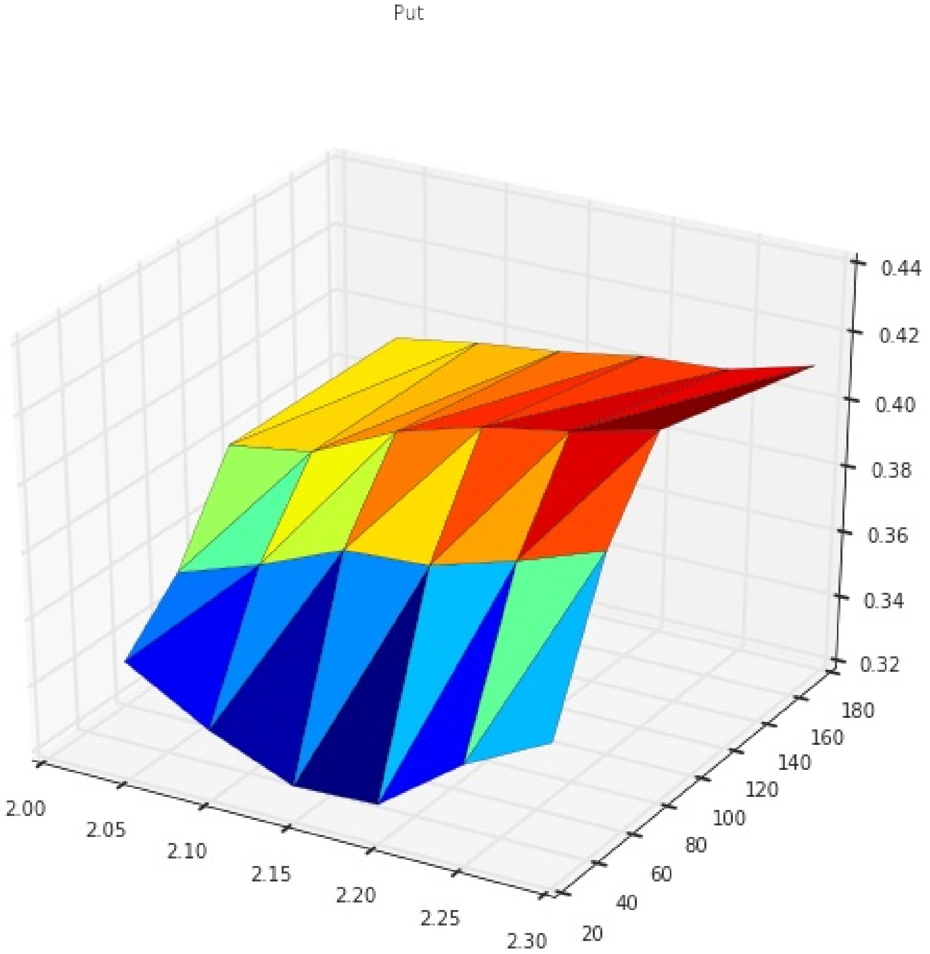
行权价和行权日期两个方向上的隐含波动率微笑

```
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm

# 做图展示某一天的隐含波动率结构
def plotSnapshotSmileVolatilitySurface():
    # Uqer 计算期权的风险数据
    date = Date.todaysDate()
    opt = getOptSnapshotGreeksIV()

    # 下面展示波动率结构
    exp_dates = np.sort(opt.index.unique())
    strikes = np.sort(opt[('Call-Put', 'strikePrice')].unique())
    risk_mt = {'Call': pd.DataFrame(index=strikes),
               'Put': pd.DataFrame(index=strikes) }

    # 将数据整理成Call和Put分开来，分别的结构为：
    # 行为行权价，列为剩余到期天数（以自然天数计算）
    for epd in exp_dates:
        exp_days = Date.parseISO(epd) - date
        opt_date = opt[opt.index==epd].set_index(('Call-Put', 's
trikePrice'))
        opt_date.index.name = 'strikePrice'
        for cp in risk_mt.keys():
            risk_mt[cp][exp_days] = opt_date[(cp, 'iv')]
    for cp in risk_mt.keys():
        for strike in risk_mt[cp].index:
            if np.sum(np.isnan(risk_mt[cp].ix[strike])) > 0:
                risk_mt[cp] = risk_mt[cp].drop(strike)

    # Call和Put分开显示，行index为行权价，列index为剩余到期天数
    #print risk_mt

    # 画图
    for cp in ['Call', 'Put']:
        opt = risk_mt[cp]
        x = []
        y = []
        z = []
        for xx in opt.index:
            for yy in opt.columns:
                x.append(xx)
                y.append(yy)
                z.append(opt[yy][xx])
        fig = plt.figure(figsize=(10,8))
        fig.suptitle(cp)
        ax = fig.gca(projection='3d')
        ax.plot_trisurf(x, y, z, cmap=cm.jet, linewidth=0.2)
    return risk_mt
```

画出某一天的波动率微笑曲面结构

```
opt = plotSnapshotSmileVolatilitySurface()
opt  # Call和Put分开显示，行index为行权价，列index为剩余到期天数

{'Call':         20      48      76     167
 2.05  0.0000  0.1663  0.2027  0.2263
 2.10  0.1998  0.2196  0.2283  0.2430
 2.15  0.2392  0.2438  0.2446  0.2502
 2.20  0.2510  0.2541  0.2570  0.2579
 2.25  0.2615  0.2631  0.2646  0.2639
 2.30  0.2780  0.2667  0.2763  0.2673,
 'Put':         20      48      76     167
 2.05  0.3535  0.3692  0.3965  0.3965
 2.10  0.3391  0.3775  0.4002  0.4002
 2.15  0.3287  0.3877  0.4116  0.4030
 2.20  0.3297  0.3891  0.4185  0.4069
 2.25  0.3483  0.3964  0.4228  0.4084
 2.30  0.3612  0.4052  0.4287  0.4149}
```





波动率曲面结构图中：

- 上图为Call，下图为Put，此处没有进行任何插值处理，略显粗糙
- Put的隐含波动率明显大于Call
- 期限结构来说，波动率呈现远高近低的特征
- 切记：所有的隐含波动率为0代表着期权的时间价值为负，此时的风险数据实际上并无多大参考意义！！

【50ETF期权】 5. 日内即时监控 Greeks 和隐含波动率微笑

> 来源：https://uqer.io/community/share/5615d10ff9f06c4ca72fb5be

和上一篇类似，计算上证50ETF期权的隐含波动率微笑，但这里通过DataAPI提供的高频数据，在交易时间实时监控Greeks 和隐含波动率变化！

```
from CAL.PyCAL import *
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import rc
rc('mathtext', default='regular')
import seaborn as sns
sns.set_style('white')
import math
from scipy import interpolate
from scipy.stats import mstats
from pandas import Series, DataFrame, concat
import time
from matplotlib import dates
```

上海银行间同业拆借利率 SHIBOR，用来作为无风险利率参考

```
## 银行间质押式回购利率
def getHistDayInterestRateInterbankRepo(date):
    cal = Calendar('China.SSE')
    period = Period('-10B')
    begin = cal.advanceDate(date, period)
    begin_str = begin.toISO().replace('-', '')
    date_str = date.toISO().replace('-', '')
    # 以下的indicID分别对应的银行间质押式回购利率周期为：
    # 1D, 7D, 14D, 21D, 1M, 3M, 4M, 6M, 9M, 1Y
    indicID = [u"M120000067", u"M120000068", u"M120000069", u"M120000070", u"M120000071",
               u"M120000072", u"M120000073", u"M120000074", u"M120000075", u"M120000076"]
    period = np.asarray([1.0, 7.0, 14.0, 21.0, 30.0, 90.0, 120.0, 180.0, 270.0, 360.0]) / 360.0
    period_matrix = pd.DataFrame(index=indicID, data=period, columns=['period'])
    field = u"indicID,indicName,publishTime,periodDate,dataValue,unit"
    interbank_repo = DataAPI.ChinaDataInterestRateInterbankRepoGet(indicID=indicID,beginDate=begin_str,endDate=date_str,field=field,pandas="1")
    interbank_repo = interbank_repo.groupby('indicID').first()
    interbank_repo = concat([interbank_repo, period_matrix], axis=1, join='inner').sort_index()
    return interbank_repo

## 银行间同业拆借利率
def getHistDaySHIBOR(date):
    cal = Calendar('China.SSE')
    period = Period('-10B')
    begin = cal.advanceDate(date, period)
    begin_str = begin.toISO().replace('-', '')
    date_str = date.toISO().replace('-', '')
    # 以下的indicID分别对应的SHIBOR周期为：
    # 1D, 7D, 14D, 1M, 3M, 6M, 9M, 1Y
    indicID = [u"M120000057", u"M120000058", u"M120000059", u"M120000060",
               u"M120000061", u"M120000062", u"M120000063", u"M120000064"]
    period = np.asarray([1.0, 7.0, 14.0, 30.0, 90.0, 180.0, 270.0, 360.0]) / 360.0
    period_matrix = pd.DataFrame(index=indicID, data=period, columns=['period'])
    field = u"indicID,indicName,publishTime,periodDate,dataValue,unit"
    interest_shibor = DataAPI.ChinaDataInterestRateSHIBORGet(indicID=indicID,beginDate=begin_str,endDate=date_str,field=field,pandas="1")
    interest_shibor = interest_shibor.groupby('indicID').first()
    interest_shibor = concat([interest_shibor, period_matrix], axis=1, join='inner').sort_index()
    return interest_shibor

## 插值得到给定的周期的无风险利率
def periodsSplineRiskFreeInterestRate(date, periods):
    # 此处使用SHIBOR来插值
    init_shibor = getHistDaySHIBOR(date)

    shibor = {}
    min_period = min(init_shibor.period.values)
    min_period = 10.0/360.0
    max_period = max(init_shibor.period.values)
    for p in periods.keys():
        tmp = periods[p]
        if periods[p] > max_period:
            tmp = max_period * 0.99999
        elif periods[p] < min_period:
            tmp = min_period * 1.00001
        sh = interpolate.spline(init_shibor.period.values, init_shibor.dataValue.values, [tmp], order=3)
        shibor[p] = sh[0]/100.0
    return shibor
```

1. Greeks 和 隐含波动率

本文中计算的Greeks包括：

- `delta` 期权价格关于标的价格的一阶导数
- `gamma` 期权价格关于标的价格的二阶导数
- `rho` 期权价格关于无风险利率的一阶导数
- `theta` 期权价格关于到期时间的一阶导数
- `vega` 期权价格关于波动率的一阶导数

注意：

- 计算隐含波动率，我们采用Black-Scholes-Merton模型，此模型在平台Python包CAL中已有实现
- 无风险利率使用SHIBOR
- 期权的时间价值为负时(此种情况在50ETF期权里时有发生)，没法通过BSM模型计算隐含波动率，故此时将期权隐含波动率设为0.0，实际上，此时的隐含波动率和各风险指标并无实际参考价值
- 此处计算即时隐含波动率和Greeks时候，我们使用买一和卖一的中间价作为期权价格

```python
## 使用DataAPI.OptGet, DataAPI.MktOptionTickRTSnapshotGet 拿到计算所需数据
def getOptTickSnapshot(opt_var_sec, date):
    date_str = date.toISO().replace('-', '')

    #使用DataAPI.OptGet，拿到已退市和上市的所有期权的基本信息
    info_fields = [u'optID', u'varSecID', u'varShortName', u'varTicker', u'varExchangeCD', u'varType',
                   u'contractType', u'strikePrice', u'contMultNum', u'contractStatus', u'listDate',
                   u'expYear', u'expMonth', u'expDate', u'lastTradeDate', u'exerDate', u'deliDate',
                   u'delistDate']
    opt_info = DataAPI.OptGet(optID='', contractStatus=[u"DE",u"L"], field=info_fields, pandas="1")

    #使用DataAPI.MktOptionTickRTSnapshotGet，拿到期权实时成交数据
    date_str = date.toISO().replace('-', '')
    fields_mkt = ['instrumentID', u'optionId', u'dataDate', u'lastPrice', u'preSettlePrice',
                  u'bidBook_price1', u'bidBook_volume1', u'askBook_price1', u'askBook_volume1']
    opt_mkt = DataAPI.MktOptionTickRTSnapshotGet(optionId=u"", field='', pandas="1")
    opt_mkt = opt_mkt[opt_mkt.dataDate == date.toISO()]
    opt_mkt['optID'] = map(int, opt_mkt['optionId'])
    opt_mkt[u"price"] = (opt_mkt['bidBook_price1'] + opt_mkt['askBook_price1'])/2.0
    opt_info = opt_info.set_index(u"optID")
    opt_mkt = opt_mkt.set_index(u"optID")
    opt = concat([opt_info, opt_mkt], axis=1, join='inner').sort
_index()
    return opt

## 分析期权即时交易信息，得到隐含波动率微笑和期权风险指标
def getOptAnalysisSnapshot(opt_var_sec):
    date = Date.todaysDate()
    opt = getOptTickSnapshot(opt_var_sec, date)

    #使用DataAPI.MktTickRTSnapshotGet 拿到期权标的的即时行情
    date_str = date.toISO().replace('-', '')
    opt_var_mkt = DataAPI.MktTickRTSnapshotGet(securityID=opt_va
r_sec,field=u"lastPrice",pandas="1")

    # 计算shibor
    exp_dates_str = opt.expDate.unique()
    periods = {}
    for date_str in exp_dates_str:
        exp_date = Date.parseISO(date_str)
        periods[exp_date] = (exp_date - date)/360.0
    shibor = periodsSplineRiskFreeInterestRate(date, periods)

    settle = opt.price.values         # 期权 settle price
    close = opt.price.values          # 期权 close price
    strike = opt.strikePrice.values        # 期权 strike price
    option_type = opt.contractType.values  # 期权类型
    exp_date_str = opt.expDate.values      # 期权行权日期
    eval_date_str = opt.dataDate.values   # 期权交易日期

    mat_dates = []
    eval_dates = []
    spot = []
    for epd, evd in zip(exp_date_str, eval_date_str):
        mat_dates.append(Date.parseISO(epd))
        eval_dates.append(Date.parseISO(evd))
        spot.append(opt_var_mkt.lastPrice[0])
    time_to_maturity = [float(mat - eva + 1.0)/365.0 for (mat, e
va) in zip(mat_dates, eval_dates)]

    risk_free = []  # 无风险利率
    for s, mat, time in zip(spot, mat_dates, time_to_maturity):
        #rf = math.log(forward_price[mat] / s) / time
        rf = shibor[mat]
        risk_free.append(rf)

    opt_types = []   # 期权类型
    for t in option_type:
        if t == 'CO':
            opt_types.append(1)
        else:
            opt_types.append(-1)
    # 使用通联CAL包中 BSMImpliedVolatity 计算隐含波动率
    calculated_vol = BSMImpliedVolatity(opt_types, strike, spot,
 risk_free, 0.0, time_to_maturity, settle)
    calculated_vol = calculated_vol.fillna(0.0)

    # 使用通联CAL包中 BSMPrice 计算期权风险指标
    greeks = BSMPrice(opt_types, strike, spot, risk_free, 0.0, c
alculated_vol.vol.values, time_to_maturity)
    # vega、rho、theta 的计量单位参照上交所的数据，以求统一对比
    greeks.vega = greeks.vega #/ 100.0
    greeks.rho = greeks.rho #/ 100.0
    greeks.theta = greeks.theta #* 365.0 / 252.0 #/ 365.0

    opt['strike'] = strike
    opt['optType'] = option_type
    opt['expDate'] = exp_date_str
    opt['spotPrice'] = spot
    opt['riskFree'] = risk_free
    opt['timeToMaturity'] = np.around(time_to_maturity, decimals=
4)
    opt['settle'] = np.around(greeks.price.values.astype(np.doub
le), decimals=4)
    opt['iv'] = np.around(calculated_vol.vol.values.astype(np.do
uble), decimals=4)
    opt['delta'] = np.around(greeks.delta.values.astype(np.doubl
e), decimals=4)
    opt['vega'] = np.around(greeks.vega.values.astype(np.double)
, decimals=4)
    opt['gamma'] = np.around(greeks.gamma.values.astype(np.doubl
e), decimals=4)
    opt['theta'] = np.around(greeks.theta.values.astype(np.doubl
e), decimals=4)
    opt['rho'] = np.around(greeks.rho.values.astype(np.double),
decimals=4)

    fields = [u'instrumentID', u'contractType', u'strikePrice',
u'expDate', u'dataDate', u'dataTime',
              u'price', 'spotPrice', u'iv',
              u'delta', u'vega', u'gamma', u'theta',  u'rho']
    opt = opt[fields].reset_index().set_index('instrumentID').so
rt_index()
    #opt['iv'] = opt.iv.replace(to_replace=0.0, value=np.nan)
    return opt

# 期权分析数据整理
def getOptSnapshotGreeksIV():
    # Uqer 计算期权的风险数据
    opt_var_sec = u"510050.XSHG"    # 期权标的
    opt = getOptAnalysisSnapshot(opt_var_sec)

    # 整理数据部分
    opt.index = [index[-10:] for index in opt.index]
    opt = opt[['spotPrice', 'contractType','strikePrice','expDate','price','iv','delta','theta','gamma','vega','rho']]
    opt_call = opt[opt.contractType=='CO']
    opt_put = opt[opt.contractType=='PO']
    opt_call.columns = pd.MultiIndex.from_tuples([('Call', c) for c in opt_call.columns])
    opt_call[('Call-Put', 'spotPrice')] = opt_call[('Call', 'spotPrice')]
    opt_call[('Call-Put', 'strikePrice')] = opt_call[('Call', 'strikePrice')]
    opt_put.columns = pd.MultiIndex.from_tuples([('Put', c) for c in opt_put.columns])
    opt = concat([opt_call, opt_put], axis=1, join='inner').sort_index()
    opt = opt.set_index(('Call','expDate')).sort_index()
    opt = opt.drop([('Call','contractType'), ('Call','strikePrice'), ('Call','spotPrice')], axis=1)
    opt = opt.drop([('Put','expDate'), ('Put','contractType'), ('Put','strikePrice'), ('Put','spotPrice')], axis=1)
    opt.index.name = 'expDate'
    ## 以上得到完整的历史某日数据，格式简洁明了
    return opt
```

即时风险数据计算，其中的price就是买一、卖一价格的平均

```
DataAPI.MktTickRTSnapshotGet(securityID="510050.XSHG",field=u"",pandas="1").columns

Index([u'exchangeCD', u'ticker', u'timestamp', u'AggQty', u'amplitude', u'change', u'changePct', u'currencyCD', u'dataDate', u'dataTime', u'deal', u'extraLargeOrderValue', u'highPrice', u'IEP', u'largeOrderValue', u'lastPrice', u'localTimestamp', u'lowPrice', u'mediumOrderValue', u'negMarketValue', u'nominalPrice', u'openPrice', u'orderType', u'prevClosePrice', u'shortNM', u'smallOrderValue', u'staticPE', u'suspension', u'totalOrderValue', u'tradSessionID', u'tradSessionStatus', u'tradSessionSubID', u'tradStatus', u'tradType', u'turnoverRate', u'utcOffset', u'value', u'volume', u'VWAP', u'Yield', u'bidBook_price1', u'bidBook_volume1', u'bidBook_price2', u'bidBook_volume2', u'bidBook_price3', u'bidBook_volume3', u'bidBook_price4', u'bidBook_volume4', u'bidBook_price5', u'bidBook_volume5', u'askBook_price1', u'askBook_volume1', u'askBook_price2', u'askBook_volume2', u'askBook_price3', u'askBook_volume3', u'askBook_price4', u'askBook_volume4', u'askBook_price5', u'askBook_volume5'], dtype='object')
```

```
opt = getOptSnapshotGreeksIV()
opt.head(20)
```

	Call	Call-Put	Put				
price	iv	delta	theta	gamma	vega	rho	spotP
expDate							
2015-10-28	0.35250	0.0000	NaN	NaN	NaN	NaN	NaN
2015-10-28	0.30390	0.0000	NaN	NaN	NaN	NaN	NaN
2015-10-28	0.25710	0.0000	NaN	NaN	NaN	NaN	NaN
2015-10-28	0.20990	0.0000	NaN	NaN	NaN	NaN	NaN
2015-10-28	0.16690	0.0000	NaN	NaN	NaN	NaN	NaN
2015-10-28	0.12500	0.1998	0.8793	-0.2390	1.8916	0.1067	0.104
2015-10-28	0.09155	0.2392	0.7182	-0.4171	2.6574	0.1794	0.086
2015-10-28	0.06285	0.2510	0.5679	-0.4908	2.9485	0.2089	0.068
2015-10-28	0.04160	0.2615	0.4241	-0.4994	2.8189	0.2081	0.051
2015-10-28	0.02795	0.2780	0.3064	-0.4697	2.3766	0.1865	0.037
2015-10-28	0.01655	0.2793	0.2049	-0.3791	1.9141	0.1509	0.025
2015-11-25	0.17915	0.1663	0.9149	-0.1371	1.1545	0.1264	0.248
2015-11-25	0.14805	0.2196	0.7751	-0.2490	1.6819	0.2433	0.210
2015-11-25	0.12025	0.2438	0.6649	-0.3116	1.8413	0.2957	0.181
2015-11-25	0.09455	0.2541	0.5657	-0.3391	1.9085	0.3194	0.155
2015-11-25	0.07355	0.2631	0.4721	-0.3475	1.8635	0.3230	0.130

2015-11-25	0.05525	0.2667	0.3849	-0.3335	1.7659	0.3102	0.107
2015-12-23	0.38610	0.0000	NaN	NaN	NaN	NaN	NaN
2015-12-23	0.34710	0.0000	NaN	NaN	NaN	NaN	NaN
2015-12-23	0.30415	0.0000	NaN	NaN	NaN	NaN	NaN

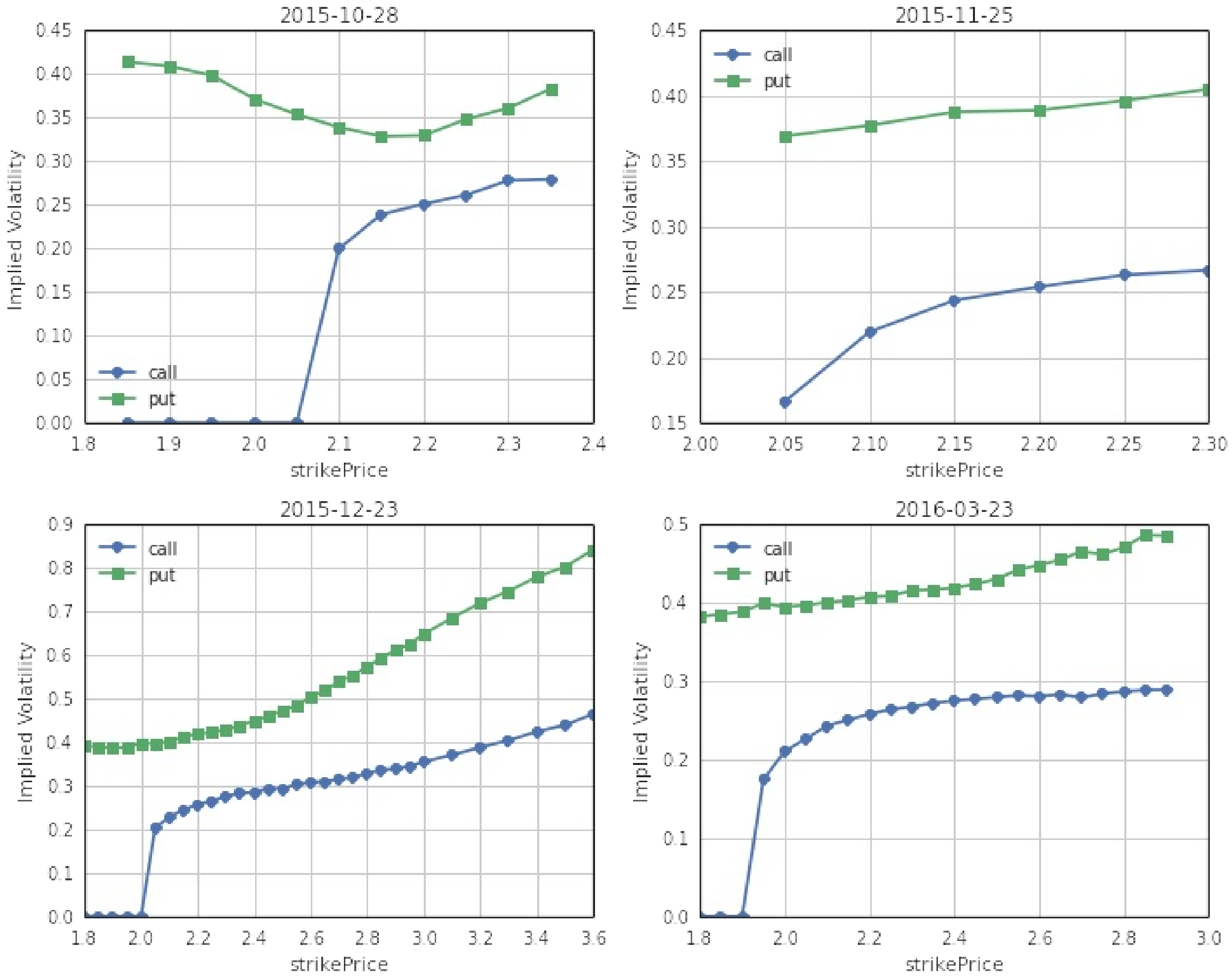
2. 隐含波动率微笑

利用上一小节的代码，给出隐含波动率微笑结构

隐含波动率微笑

```
# 做图展示某一天的隐含波动率微笑
def plotSnapshotSmileVolatility():
    # Uqer 计算期权的风险数据
    opt = getOptSnapshotGreeksIV()

    # 下面展示波动率微笑
    exp_dates = np.sort(opt.index.unique())
    ## ---------------------------------------------
    fig = plt.figure(figsize=(10,8))
    fig.set_tight_layout(True)

    for i in range(exp_dates.shape[0]):
        date = exp_dates[i]
        ax = fig.add_subplot(2,2,i+1)
        opt_date = opt[opt.index==date].set_index(('Call-Put', '
strikePrice'))
        opt_date.index.name = 'strikePrice'

        ax.plot(opt_date.index, opt_date[('Call', 'iv')], '-o')
        ax.plot(opt_date.index, opt_date[('Put', 'iv')], '-s')
        ax.legend(['call', 'put'], loc=0)
        ax.grid()
        ax.set_xlabel(u"strikePrice")
        ax.set_ylabel(r"Implied Volatility")
        plt.title(exp_dates[i])
```

```
plotSnapshotSmileVolatility()
```

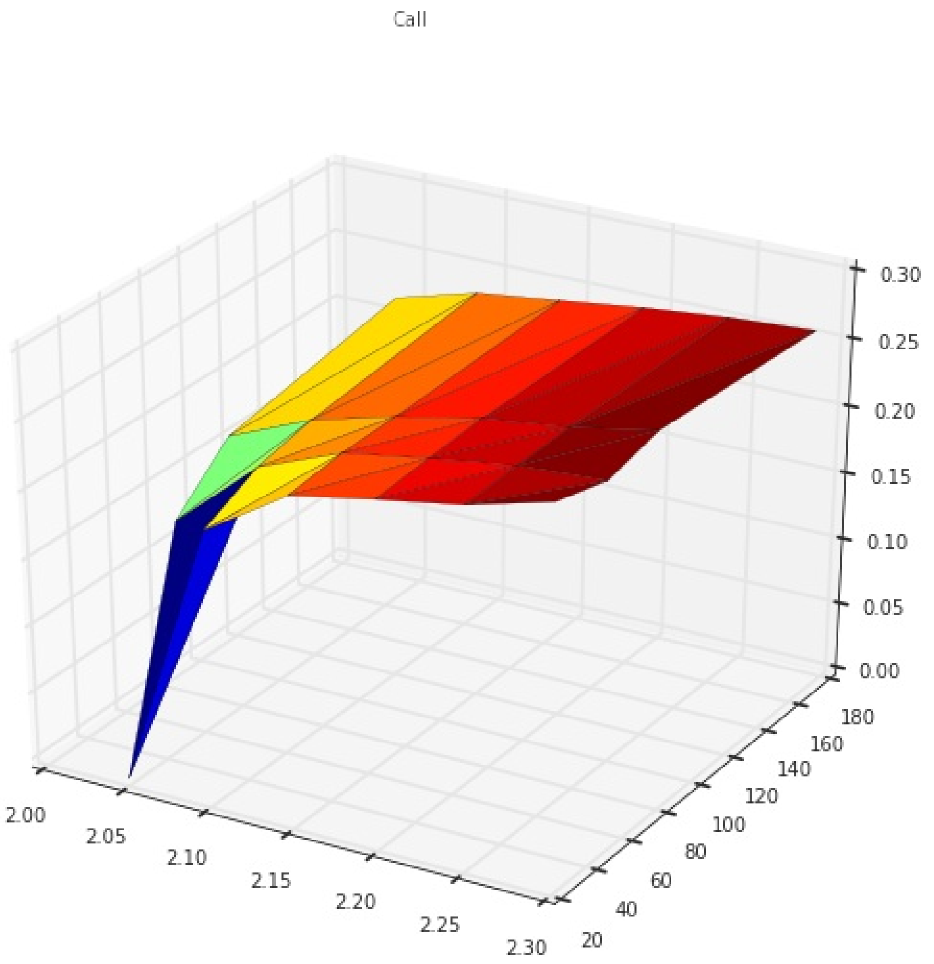


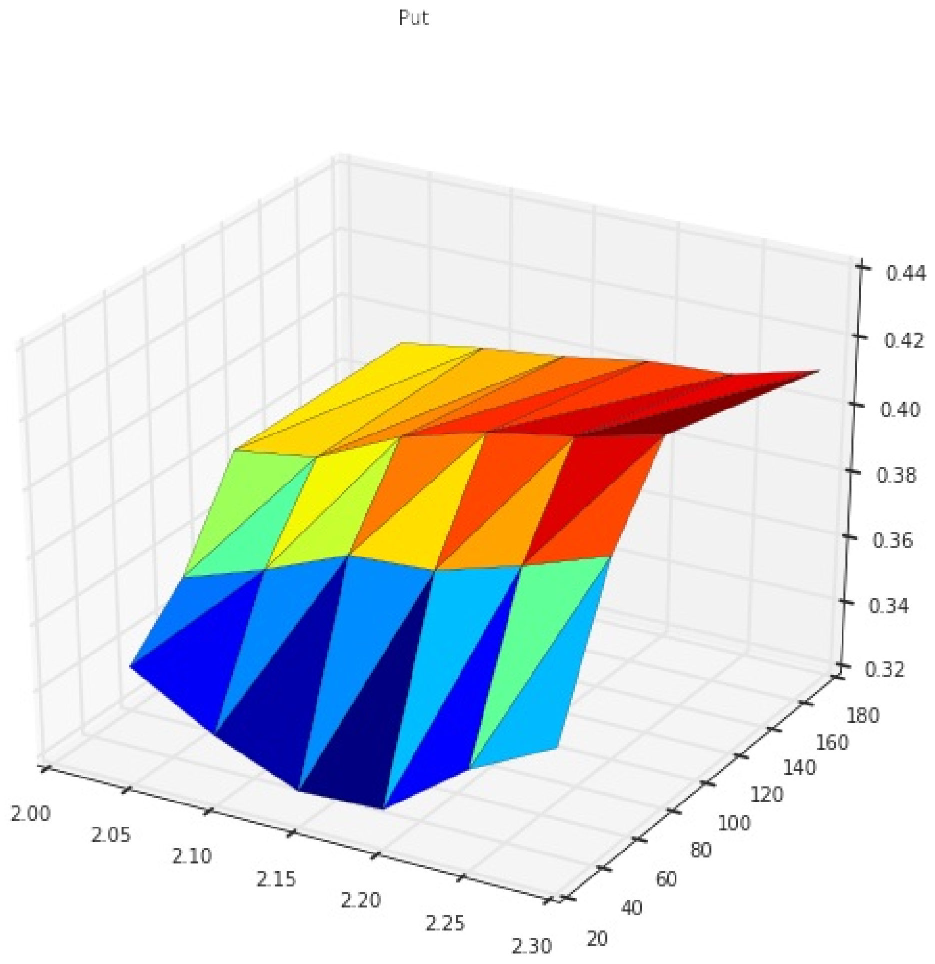
行权价和行权日期两个方向上的隐含波动率微笑

```
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm

# 做图展示某一天的隐含波动率结构
def plotSnapshotSmileVolatilitySurface():
    # Uqer 计算期权的风险数据
    date = Date.todaysDate()
    opt = getOptSnapshotGreeksIV()

    # 下面展示波动率结构
    exp_dates = np.sort(opt.index.unique())
    strikes = np.sort(opt[('Call-Put', 'strikePrice')].unique())
    risk_mt = {'Call': pd.DataFrame(index=strikes),
               'Put': pd.DataFrame(index=strikes) }

    # 将数据整理成Call和Put分开来，分别的结构为：
    # 行为行权价，列为剩余到期天数（以自然天数计算）
    for epd in exp_dates:
        exp_days = Date.parseISO(epd) - date
        opt_date = opt[opt.index==epd].set_index(('Call-Put', 's
trikePrice'))
        opt_date.index.name = 'strikePrice'
        for cp in risk_mt.keys():
            risk_mt[cp][exp_days] = opt_date[(cp, 'iv')]
    for cp in risk_mt.keys():
        for strike in risk_mt[cp].index:
            if np.sum(np.isnan(risk_mt[cp].ix[strike])) > 0:
                risk_mt[cp] = risk_mt[cp].drop(strike)

    # Call和Put分开显示，行index为行权价，列index为剩余到期天数
    #print risk_mt

    # 画图
    for cp in ['Call', 'Put']:
        opt = risk_mt[cp]
        x = []
        y = []
        z = []
        for xx in opt.index:
            for yy in opt.columns:
                x.append(xx)
                y.append(yy)
                z.append(opt[yy][xx])
        fig = plt.figure(figsize=(10,8))
        fig.suptitle(cp)
        ax = fig.gca(projection='3d')
        ax.plot_trisurf(x, y, z, cmap=cm.jet, linewidth=0.2)
    return risk_mt
```

画出某一天的波动率微笑曲面结构

```
opt = plotSnapshotSmileVolatilitySurface()
opt  # Call和Put分开显示，行index为行权价，列index为剩余到期天数

{'Call':           20       48       76      167
 2.05  0.0000  0.1663  0.2027  0.2263
 2.10  0.1998  0.2196  0.2283  0.2430
 2.15  0.2392  0.2438  0.2446  0.2502
 2.20  0.2510  0.2541  0.2570  0.2579
 2.25  0.2615  0.2631  0.2646  0.2639
 2.30  0.2780  0.2667  0.2763  0.2673,
 'Put':           20       48       76      167
 2.05  0.3535  0.3692  0.3965  0.3965
 2.10  0.3391  0.3775  0.4002  0.4002
 2.15  0.3287  0.3877  0.4116  0.4030
 2.20  0.3297  0.3891  0.4185  0.4069
 2.25  0.3483  0.3964  0.4228  0.4084
 2.30  0.3612  0.4052  0.4287  0.4149}
```





波动率曲面结构图中：

- 上图为Call，下图为Put，此处没有进行任何插值处理，略显粗糙
- Put的隐含波动率明显大于Call
- 期限结构来说，波动率呈现远高近低的特征
- 切记：所有的隐含波动率为0代表着期权的时间价值为负，此时的风险数据实际上并无多大参考意义！！

三 期权分析

【50ETF期权】 期权择时指数 1.0

> 来源：https://uqer.io/community/share/561c883df9f06c4ca72fb5f7

本文中，我们使用期权的日行情数据，计算期权情绪指标，并用以指导实战择时

初步讨论只包括两个指标

- 成交量(成交额) PCR：看跌看涨期权的成交量(成交额)比率
- PCIVD：Put Call Implied Volatility Difference 看跌看涨期权隐含波动率差

```
from CAL.PyCAL import *
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import rc
rc('mathtext', default='regular')
import seaborn as sns
sns.set_style('white')
from matplotlib import dates
from pandas import concat
from scipy import interpolate
import math
```

1. 看跌看涨成交量（成交额）比率 PCR

- 计算每日看跌看涨成交量或成交额的比率，即PCR
- 我们考虑PCR每日变化量与现货50ETF隔日收益率的关系
- 每日PCR变化量PCRD为：当日PCR减去前一日PCR得到的值，即对PCR做差分

```
def histVolumeOpt50ETF(beginDate, endDate):
    ## 计算历史一段时间内的50ETF期权持仓量交易量数据

    optionVarSecID = u"510050.XSHG"
    cal = Calendar('China.SSE')
    dates = cal.bizDatesList(beginDate, endDate)
    dates = map(Date.toDateTime, dates)
    columns = ['callVol', 'putVol', 'callValue',
               'putValue', 'callOpenInt', 'putOpenInt',
               'nearCallVol', 'nearPutVol', 'nearCallValue',
               'nearPutValue', 'nearCallOpenInt', 'nearPutOpenIn
t',
               'netVol', 'netValue', 'netOpenInt',
               'volPCR', 'valuePCR', 'openIntPCR',
               'nearVolPCR', 'nearValuePCR', 'nearOpenIntPCR']
    hist_opt = pd.DataFrame(0.0, index=dates, columns=columns)
    hist_opt.index.name = 'date'
    # 每一个交易日数据单独计算
    for date in hist_opt.index:
        date_str = Date.fromDateTime(date).toISO().replace('-',
'')
        try:
            opt_data = DataAPI.MktOptdGet(secID=u"", tradeDate=d
ate_str, field=u"", pandas="1")
        except:
            hist_opt = hist_opt.drop(date)
            continue

        opt_type = []
        exp_date = []
        for ticker in opt_data.secID.values:
            opt_type.append(ticker[6])
            exp_date.append(ticker[7:11])
        opt_data['optType'] = opt_type
        opt_data['expDate'] = exp_date
        near_exp = np.sort(opt_data.expDate.unique())[0]

        data = opt_data.groupby('optType')
        # 计算所有上市期权：看涨看跌交易量、看涨看跌交易额、看涨看跌持仓量
        hist_opt['callVol'][date] = data.turnoverVol.sum()['C']
        hist_opt['putVol'][date] = data.turnoverVol.sum()['P']
        hist_opt['callValue'][date] = data.turnoverValue.sum()['
C']
        hist_opt['putValue'][date] = data.turnoverValue.sum()['P'
]
        hist_opt['callOpenInt'][date] = data.openInt.sum()['C']
        hist_opt['putOpenInt'][date] = data.openInt.sum()['P']

        near_data = opt_data[opt_data.expDate == near_exp]
        near_data = near_data.groupby('optType')
        # 计算近月期权(主力合约)： 看涨看跌交易量、看涨看跌交易额、看涨看
跌持仓量
        hist_opt['nearCallVol'][date] = near_data.turnoverVol.su
m()['C']
        hist_opt['nearPutVol'][date] = near_data.turnoverVol.sum
()['P']
        hist_opt['nearCallValue'][date] = near_data.turnoverValu
e.sum()['C']
        hist_opt['nearPutValue'][date] = near_data.turnoverValue
.sum()['P']
        hist_opt['nearCallOpenInt'][date] = near_data.openInt.su
m()['C']
        hist_opt['nearPutOpenInt'][date] = near_data.openInt.sum
()['P']

        # 计算所有上市期权： 总交易量、总交易额、总持仓量
        hist_opt['netVol'][date] = hist_opt['callVol'][date] + h
ist_opt['putVol'][date]
```

```
        hist_opt['netValue'][date] = hist_opt['callValue'][date] + hist_opt['putValue'][date]
        hist_opt['netOpenInt'][date] = hist_opt['callOpenInt'][date] + hist_opt['putOpenInt'][date]

        # 计算期权看跌看涨期权交易量(持仓量)的比率：
        # 交易量看跌看涨比率，交易额看跌看涨比率, 持仓量看跌看涨比率
        # 近月期权交易量看跌看涨比率，近月期权交易额看跌看涨比率, 近月期权持仓量看跌看涨比率
        # PCR = Put Call Ratio
        hist_opt['volPCR'][date] = round(hist_opt['putVol'][date]*1.0/hist_opt['callVol'][date], 4)
        hist_opt['valuePCR'][date] = round(hist_opt['putValue'][date]*1.0/hist_opt['callValue'][date], 4)
        hist_opt['openIntPCR'][date] = round(hist_opt['putOpenInt'][date]*1.0/hist_opt['callOpenInt'][date], 4)
        hist_opt['nearVolPCR'][date] = round(hist_opt['nearPutVol'][date]*1.0/hist_opt['nearCallVol'][date], 4)
        hist_opt['nearValuePCR'][date] = round(hist_opt['nearPutValue'][date]*1.0/hist_opt['nearCallValue'][date], 4)
        hist_opt['nearOpenIntPCR'][date] = round(hist_opt['nearPutOpenInt'][date]*1.0/hist_opt['nearCallOpenInt'][date], 4)
    return hist_opt

def histPrice50ETF(beginDate, endDate):
    # 华夏上证50ETF收盘价数据
    secID = '510050.XSHG'
    begin = Date.fromDateTime(beginDate).toISO().replace('-', '')
    end = Date.fromDateTime(endDate).toISO().replace('-', '')
    fields = ['tradeDate', 'closePrice', 'preClosePrice']
    etf = DataAPI.MktFunddGet(secID, beginDate=begin, endDate=end, field=fields)
    etf['tradeDate'] = pd.to_datetime(etf['tradeDate'])
    etf['dailyReturn'] = etf['closePrice'] / etf['preClosePrice'] - 1.0
    etf = etf.set_index('tradeDate')
    return etf

def histPCR50ETF(beginDate, endDate):
    # PCRD: Put Call Ratio Diff
    # 计算每日PCR变化量：当日PCR减去前一日PCR得到的值，即对PCR做差分
    # 专注于某一项PCR，例如：成交额PCR --- valuePCR
    pcr_names = ['volPCR', 'valuePCR', 'openIntPCR',
                 'nearVolPCR', 'nearValuePCR', 'nearOpenIntPCR']
    pcr_diff_names = [pcr + 'Diff' for pcr in pcr_names]
    pcr = histVolumeOpt50ETF(beginDate, endDate)
    for pcr_name in pcr_names:
        pcr[pcr_name + 'Diff'] = pcr[pcr_name].diff()
    return pcr[pcr_names + pcr_diff_names]
```

计算PCR

- 期权自15年2月9号上市
- 此处计算得到的数据可以用在后面几条策略中

```
## PCRD计算示例

start = datetime(2015,2, 9)                     # 回测起始时间
end  = datetime(2015, 10, 13)                     # 回测结束时间

hist_pcrd = histPCR50ETF(start, end)          # 计算PCRD
hist_pcrd.tail()
```

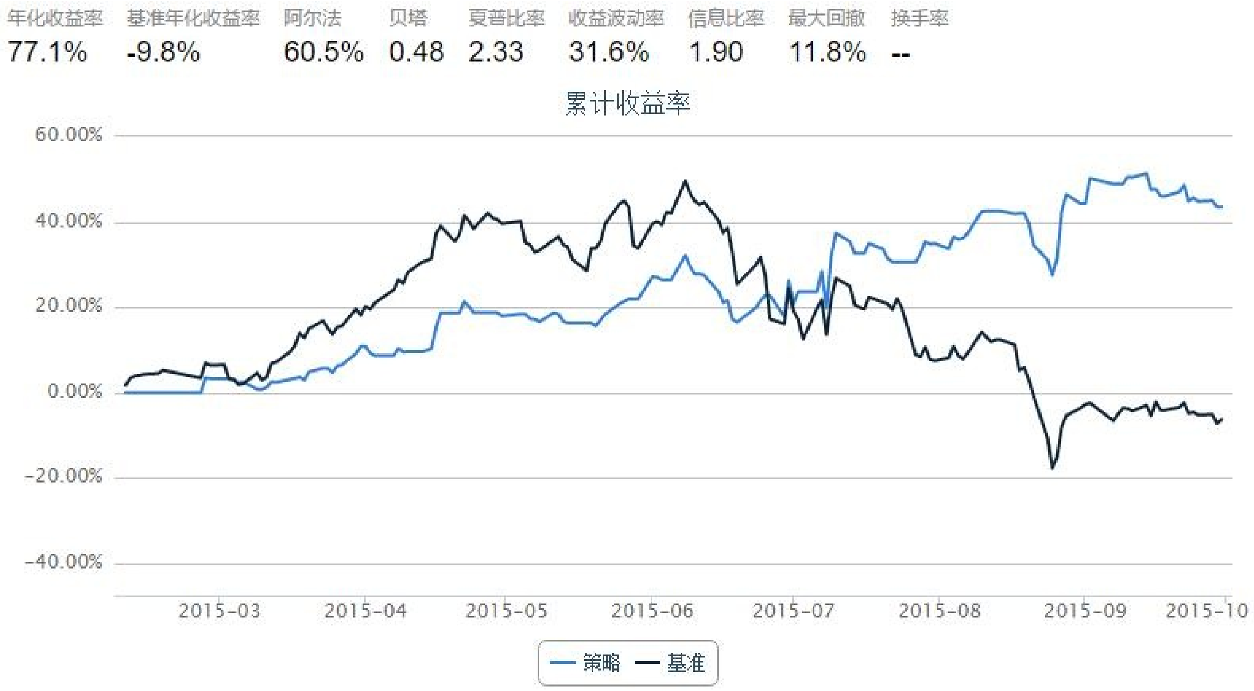
	volPCR	valuePCR	openIntPCR	nearVolPCR	nearValuePC
date					
2015-09-29	1.0863	1.5860	0.6680	1.2372	1.6552
2015-09-30	0.9664	1.1366	0.6709	1.1153	1.1460
2015-10-08	0.8997	0.5940	0.6726	0.9244	0.4646
2015-10-09	1.0979	0.7708	0.7068	1.1542	0.6672
2015-10-12	0.6494	0.2432	0.7713	0.6604	0.2002

1.1 使用基于成交量 PCR 日变化量的择时策略

策略思路：考虑成交量 PCR 日变化量 PCRD(volume)

- 前一日PCRD(volume)小于0，则今天全仓50ETF
- 否则，清仓观望
- 简单来说，就是PCR上升，空仓；PCR下降，买入

```
start = datetime(2015, 2, 9)                          # 回测起始时间
end  = datetime(2015, 10, 7)                          # 回测结束时间
benchmark = '510050.XSHG'                              # 策略参考标准
universe = ['510050.XSHG']                             # 股票池

capital_base = 100000                                 # 起始资金
commission = Commission(0.0,0.0)
refresh_rate = 1

# hist_pcrd = histPCR50ETF(start, end)                # 计算PCRD

def initialize(account):                              # 初始化虚拟账户状态
    account.fund = universe[0]

def handle_data(account):                       # 每个交易日的买入卖出指令
    fund = account.fund
    #  获取回测当日的前一天日期
    dt = Date.fromDateTime(account.current_date)
    cal = Calendar('China.IB')
    last_day = cal.advanceDate(dt,'-1B',BizDayConvention.Precedi
ng)           #计算出倒数第一个交易日
    last_day_str = last_day.strftime("%Y-%m-%d")

    # 计算买入卖出信号
    try:
        # 拿取PCRD数据
        pcrd_last_vol = hist_pcrd.volPCRDiff.loc[last_day_str]
        # PCRD(volumn)
        long_flag = True if pcrd_last_vol < 0 else False    # 调
仓条件
    except:
        long_flag = False

    if long_flag:
        # 买入时，全仓杀入
        try:
            approximationAmount = int(account.cash / account.ref
erencePrice[fund] / 100.0) * 100
            order(fund, approximationAmount)
        except:
            return
    else:
        # 卖出时，全仓清空
        order_to(fund, 0)
```



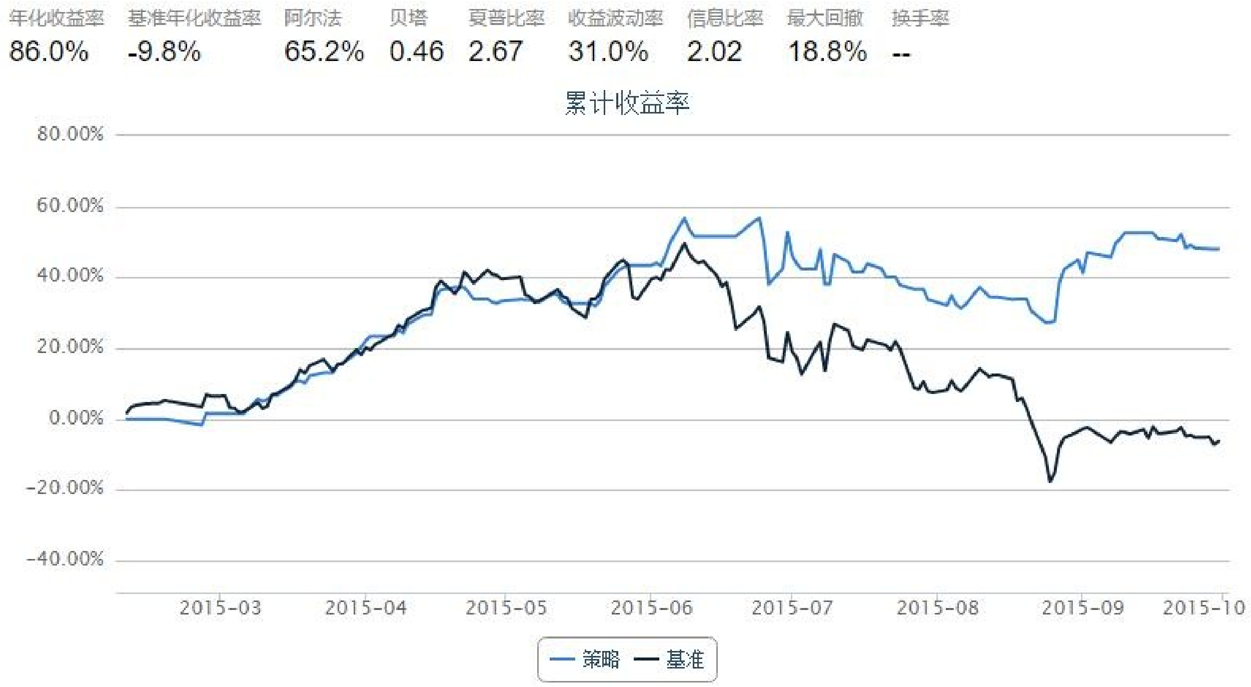
年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
77.1%	-9.8%	60.5%	0.48	2.33	31.6%	1.90	11.8%	--

1.2 使用基于成交额 PCR 日变化量的择时策略

策略思路：考虑成交额 PCR 日变化量 PCRD(value)

- 前一日PCRD(value)小于0，则今天全仓50ETF
- 否则，清仓观望
- 简单来说，就是PCR上升，空仓；PCR下降，买入

```
start = datetime(2015, 2, 9)                         # 回测起始时间
end  = datetime(2015, 10, 7)                         # 回测结束时间
benchmark = '510050.XSHG'                             # 策略参考标准
universe = ['510050.XSHG']                           # 股票池

capital_base = 100000                                # 起始资金
commission = Commission(0.0,0.0)
refresh_rate = 1

# hist_pcrd = histPCR50ETF(start, end)               # 计算PCRD

def initialize(account):                             # 初始化虚拟账户状态
    account.fund = universe[0]

def handle_data(account):                      # 每个交易日的买入卖出指令
    fund = account.fund
    #  获取回测当日的前一天日期
    dt = Date.fromDateTime(account.current_date)
    cal = Calendar('China.IB')
    last_day = cal.advanceDate(dt,'-1B',BizDayConvention.Precedi
ng)           #计算出倒数第一个交易日
    last_day_str = last_day.strftime("%Y-%m-%d")

    # 计算买入卖出信号
    try:
        # 拿取PCRD数据
        pcrd_last_value = hist_pcrd.valuePCRDiff.loc[last_day_st
r]      # PCRD(value)
        long_flag = True if pcrd_last_value < 0 else False    #
调仓条件
    except:
        long_flag = False

    if long_flag:
        # 买入时，全仓杀入
        try:
            approximationAmount = int(account.cash / account.ref
erencePrice[fund] / 100.0) * 100
            order(fund, approximationAmount)
        except:
            return
    else:
        # 卖出时，全仓清空
        order_to(fund, 0)
```



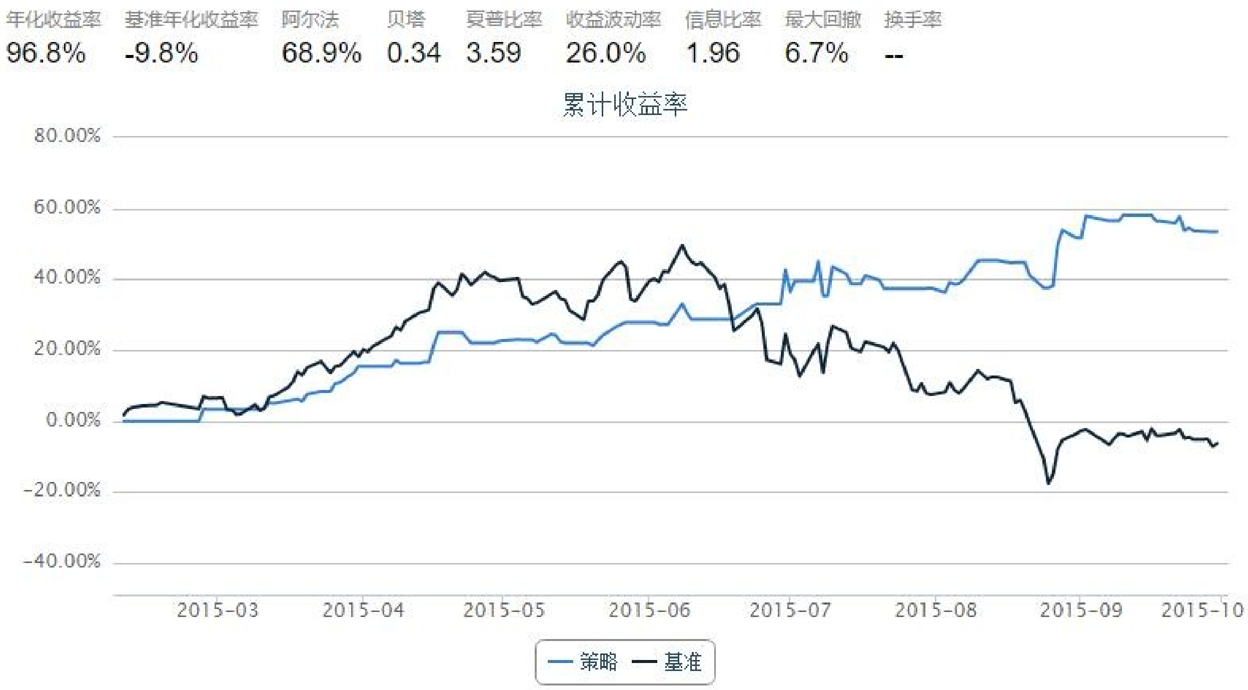
年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
86.0%	-9.8%	65.2%	0.46	2.67	31.0%	2.02	18.8%	--

1.3 结合使用成交量、成交额 PCR 日变化量的择时策略

策略思路：考虑成交量PCRD(volume) 和成交额PCRD(value)

- 前一日PCRD(volume)和PCRD(value)同时小于0，则今天全仓50ETF
- 否则，清仓观望

```
start = datetime(2015, 2, 9)                          # 回测起始时间
end  = datetime(2015, 10, 7)                          # 回测结束时间
benchmark = '510050.XSHG'                              # 策略参考标准
universe = ['510050.XSHG']                              # 股票池

capital_base = 100000                                 # 起始资金
commission = Commission(0.0,0.0)
refresh_rate = 1

hist_pcrd = histPCR50ETF(start, end)        # 计算PCRD

def initialize(account):                              # 初始化虚拟账户状态
    account.fund = universe[0]

def handle_data(account):                       # 每个交易日的买入卖出指令
    fund = account.fund
    #  获取回测当日的前一天日期
    dt = Date.fromDateTime(account.current_date)
    cal = Calendar('China.IB')
    last_day = cal.advanceDate(dt,'-1B',BizDayConvention.Precedi
ng)           #计算出倒数第一个交易日
    last_day_str = last_day.strftime("%Y-%m-%d")

    # 计算买入卖出信号
    try:
        # 拿取PCRD数据
        pcrd_last_value = hist_pcrd.valuePCRDiff.loc[last_day_st
r]      # PCRD(value)
        pcrd_last_vol = hist_pcrd.volPCRDiff.loc[last_day_str]
        # PCRD(volumn)
        long_flag = True if pcrd_last_value < 0.0 and pcrd_last_
vol < 0.0 else False    # 调仓条件
    except:
        long_flag = False

    if long_flag:
        # 买入时，全仓杀入
        try:
            approximationAmount = int(account.cash / account.ref
erencePrice[fund] / 100.0) * 100
            order(fund, approximationAmount)
        except:
            return
    else:
        # 卖出时，全仓清空
        order_to(fund, 0)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
96.8%	-9.8%	68.9%	0.34	3.59	26.0%	1.96	6.7%	--

2. 看跌看涨隐含波动率价差 PCIVD

- 相同到期日、行权价的看跌看涨期权，其隐含波动率会有差异
- 由于套保需要，一般看跌期权隐含波动率高于看涨期权
- 看跌、看涨期权隐含波动率之差 PCIVD 的每日变化可以用来指导实际操作
- 在计算中，我们使用平值附近的期权计算 PCIVD

```
## 银行间质押式回购利率
def histDayInterestRateInterbankRepo(date):
    cal = Calendar('China.SSE')
    period = Period('-10B')
    begin = cal.advanceDate(date, period)
    begin_str = begin.toISO().replace('-', '')
    date_str = date.toISO().replace('-', '')
    # 以下的indicID分别对应的银行间质押式回购利率周期为：
    # 1D, 7D, 14D, 21D, 1M, 3M, 4M, 6M, 9M, 1Y
    indicID = [u"M120000067", u"M120000068", u"M120000069", u"M120000070", u"M120000071",
               u"M120000072", u"M120000073", u"M120000074", u"M120000075", u"M120000076"]
    period = np.asarray([1.0, 7.0, 14.0, 21.0, 30.0, 90.0, 120.0, 180.0, 270.0, 360.0]) / 360.0
    period_matrix = pd.DataFrame(index=indicID, data=period, columns=['period'])
    field = u"indicID,indicName,publishTime,periodDate,dataValue,unit"
    interbank_repo = DataAPI.ChinaDataInterestRateInterbankRepoGet(indicID=indicID,beginDate=begin_str,endDate=date_str,field=field,pandas="1")
    interbank_repo = interbank_repo.groupby('indicID').first()
    interbank_repo = concat([interbank_repo, period_matrix], axi
s=1, join='inner').sort_index()
    return interbank_repo

## 银行间同业拆借利率
def histDaySHIBOR(date):
    cal = Calendar('China.SSE')
    period = Period('-10B')
    begin = cal.advanceDate(date, period)
    begin_str = begin.toISO().replace('-', '')
    date_str = date.toISO().replace('-', '')
    # 以下的indicID分别对应的SHIBOR周期为：
    # 1D, 7D, 14D, 1M, 3M, 6M, 9M, 1Y
    indicID = [u"M120000057", u"M120000058", u"M120000059", u"M1
20000060",
               u"M120000061", u"M120000062", u"M120000063", u"M1
20000064"]
    period = np.asarray([1.0, 7.0, 14.0, 30.0, 90.0, 180.0, 270.0
, 360.0]) / 360.0
    period_matrix = pd.DataFrame(index=indicID, data=period, col
umns=['period'])
    field = u"indicID,indicName,publishTime,periodDate,dataValue
,unit"
    interest_shibor = DataAPI.ChinaDataInterestRateSHIBORGet(ind
icID=indicID,beginDate=begin_str,endDate=date_str,field=field,pa
ndas="1")
    interest_shibor = interest_shibor.groupby('indicID').first()
    interest_shibor = concat([interest_shibor, period_matrix], a
xis=1, join='inner').sort_index()
    return interest_shibor

## 插值得到给定的周期的无风险利率
def periodsSplineRiskFreeInterestRate(date, periods):
    # 此处使用SHIBOR来插值
    init_shibor = histDaySHIBOR(date)

    shibor = {}
    min_period = min(init_shibor.period.values)
    min_period = 25.0/360.0
    max_period = max(init_shibor.period.values)
    for p in periods.keys():
        tmp = periods[p]
        if periods[p] > max_period:
            tmp = max_period * 0.99999
        elif periods[p] < min_period:
            tmp = min_period * 1.00001
        sh = interpolate.spline(init_shibor.period.values, init_
shibor.dataValue.values, [tmp], order=3)
        shibor[p] = sh[0]/100.0
    return shibor


## 使用DataAPI.OptGet， DataAPI.MktOptdGet拿到计算所需数据
```

```
def histDayDataOpt50ETF(date):
    date_str = date.toISO().replace('-', '')

    #使用DataAPI.OptGet，拿到已退市和上市的所有期权的基本信息
    info_fields = [u'optID', u'varSecID', u'varShortName', u'varTicker', u'varExchangeCD', u'varType',
                   u'contractType', u'strikePrice', u'contMultNum', u'contractStatus', u'listDate',
                   u'expYear', u'expMonth', u'expDate', u'lastTradeDate', u'exerDate', u'deliDate',
                   u'delistDate']
    opt_info = DataAPI.OptGet(optID='', contractStatus=[u"DE",u"L"], field=info_fields, pandas="1")

    #使用DataAPI.MktOptdGet，拿到历史上某一天的期权成交信息
    mkt_fields = [u'ticker', u'optID', u'secShortName', u'exchangeCD', u'tradeDate', u'preSettlePrice',
                  u'preClosePrice', u'openPrice', u'highestPrice', u'lowestPrice', u'closePrice',
                  u'settlPrice', u'turnoverVol', u'turnoverValue', u'openInt']
    opt_mkt = DataAPI.MktOptdGet(tradeDate=date_str, field=mkt_fields, pandas = "1")

    opt_info = opt_info.set_index(u"optID")
    opt_mkt = opt_mkt.set_index(u"optID")
    opt = concat([opt_info, opt_mkt], axis=1, join='inner').sort_index()
    return opt

# 旧版forward计算稍有差别
def histDayMktForwardPriceOpt50ETF(opt, risk_free):
    exp_dates_str = np.sort(opt.expDate.unique())
    trade_date = Date.parseISO(opt.tradeDate.values[0])

    forward = {}
    for date_str in exp_dates_str:
        opt_date = opt[opt.expDate == date_str]
        opt_call_date = opt_date[opt_date.contractType == 'CO']
        opt_put_date = opt_date[opt_date.contractType == 'PO']
        opt_call_date = opt_call_date[[u'strikePrice', u'price']].set_index('strikePrice').sort_index()
        opt_put_date = opt_put_date[[u'strikePrice', u'price']].set_index('strikePrice').sort_index()
        opt_call_date.columns = [u'callPrice']
        opt_put_date.columns = [u'putPrice']

        opt_date = concat([opt_call_date, opt_put_date], axis=1, join='inner').sort_index()
        opt_date['diffCallPut'] = opt_date.callPrice - opt_date.putPrice

        strike = abs(opt_date['diffCallPut']).idxmin()
        priceDiff = opt_date['diffCallPut'][strike]
        date = Date.parseISO(date_str)
        ttm = abs(float(date - trade_date + 1.0)/365.0)
        rf = risk_free[date]
        fw = strike + np.exp(ttm*rf) * priceDiff
        forward[date] = fw
    return forward

## 分析历史某一日的期权收盘价信息，得到隐含波动率微笑和期权风险指标
def histDayAnalysisOpt50ETF(date):
    opt_var_sec = u"510050.XSHG"    # 期权标的
    opt = histDayDataOpt50ETF(date)

    #使用DataAPI.MktFunddGet拿到期权标的的日行情
    date_str = date.toISO().replace('-', '')
    opt_var_mkt = DataAPI.MktFunddGet(secID=opt_var_sec,tradeDat
e=date_str,beginDate=u"",endDate=u"",field=u"",pandas="1")
    #opt_var_mkt = DataAPI.MktFunddAdjGet(secID=opt_var_sec,begi
nDate=date_str,endDate=date_str,field=u"",pandas="1")

    # 计算shibor
    exp_dates_str = opt.expDate.unique()
    periods = {}
    for date_str in exp_dates_str:
        exp_date = Date.parseISO(date_str)
        periods[exp_date] = (exp_date - date)/360.0
    shibor = periodsSplineRiskFreeInterestRate(date, periods)

    # 计算forward price
    opt_tmp = opt[[u'contractType', u'tradeDate', u'strikePrice'
, u'expDate', u'settlPrice']]
    opt_tmp.columns = [[u'contractType', u'tradeDate', u'strikeP
rice', u'expDate', u'price']]
    forward_price = histDayMktForwardPriceOpt50ETF(opt_tmp, shib
or)

    settle = opt.settlPrice.values         # 期权 settle price
    close = opt.closePrice.values          # 期权 close price
    strike = opt.strikePrice.values        # 期权 strike price
    option_type = opt.contractType.values  # 期权类型
    exp_date_str = opt.expDate.values      # 期权行权日期
    eval_date_str = opt.tradeDate.values   # 期权交易日期

    mat_dates = []
    eval_dates = []
    spot = []
    for epd, evd in zip(exp_date_str, eval_date_str):
        mat_dates.append(Date.parseISO(epd))
        eval_dates.append(Date.parseISO(evd))
        spot.append(opt_var_mkt.closePrice[0])
    time_to_maturity = [float(mat - eva + 1.0)/365.0 for (mat, e
va) in zip(mat_dates, eval_dates)]
```

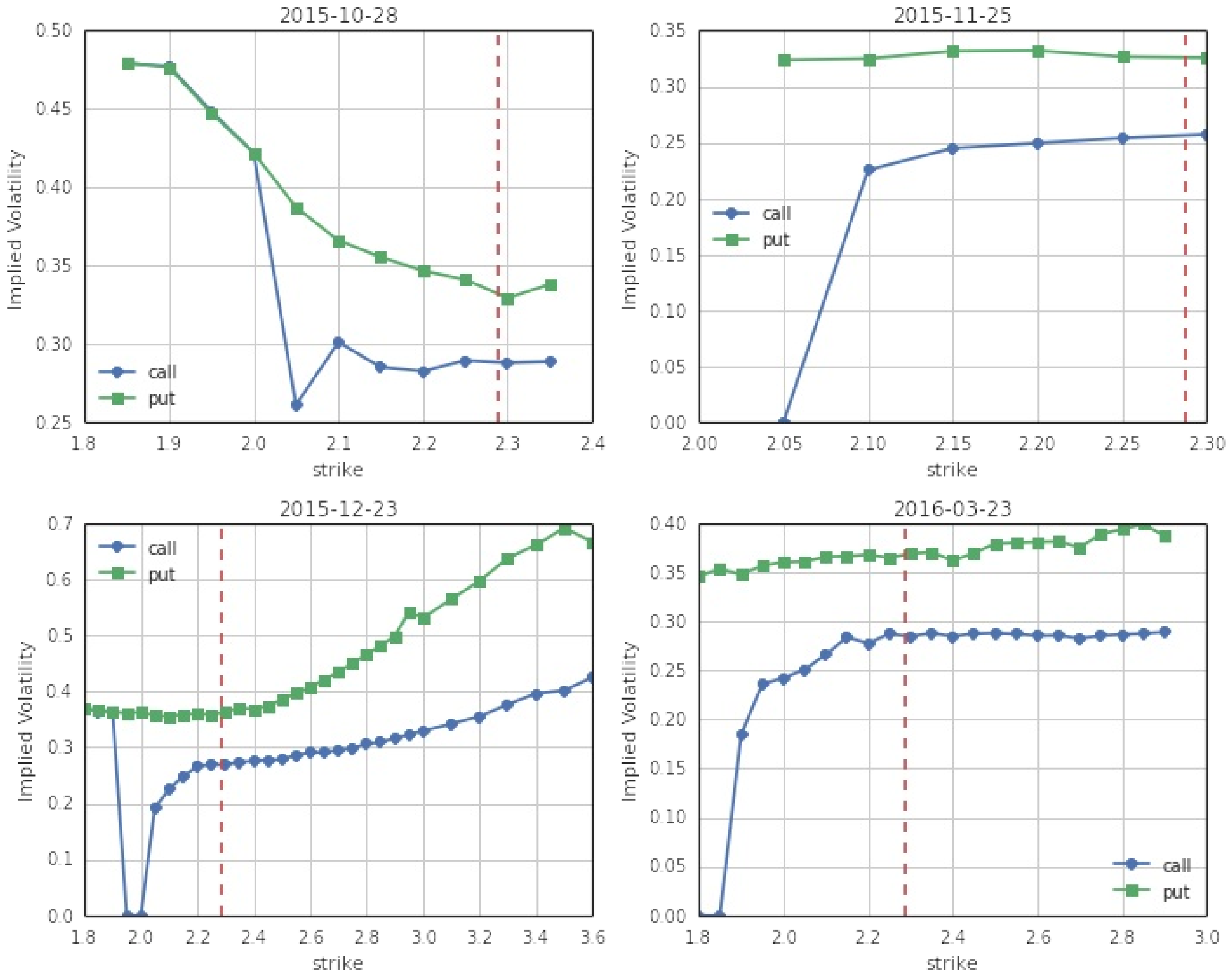
```
    risk_free = []  # 无风险利率
    forward = []    # 市场远期
    for s, mat, time in zip(spot, mat_dates, time_to_maturity):
        #rf = math.log(forward_price[mat] / s) / time
        rf = shibor[mat]
        risk_free.append(rf)
        forward.append(forward_price[mat])

    opt_types = []   # 期权类型
    for t in option_type:
        if t == 'CO':
            opt_types.append(1)
        else:
            opt_types.append(-1)

    # 使用通联CAL包中 BSMImpliedVolatity 计算隐含波动率
    calculated_vol = BSMImpliedVolatity(opt_types, strike, spot,
 risk_free, 0.0, time_to_maturity, settle)
    calculated_vol = calculated_vol.fillna(0.0)

    # 使用通联CAL包中 BSMPrice 计算期权风险指标
    greeks = BSMPrice(opt_types, strike, spot, risk_free, 0.0, c
alculated_vol.vol.values, time_to_maturity)
    # vega、rho、theta 的计量单位参照上交所的数据，以求统一对比
    greeks.vega = greeks.vega #/ 100.0
    greeks.rho = greeks.rho #/ 100.0
    greeks.theta = greeks.theta #* 365.0 / 252.0 #/ 365.0

    opt['strike'] = strike
    opt['forward'] = np.around(forward, decimals=3)
    opt['optType'] = option_type
    opt['expDate'] = exp_date_str
    opt['spotPrice'] = spot
    opt['riskFree'] = risk_free
    opt['timeToMaturity'] = np.around(time_to_maturity, decimals=
4)
    opt['settle'] = np.around(greeks.price.values.astype(np.doub
le), decimals=4)
    opt['iv'] = np.around(calculated_vol.vol.values.astype(np.do
uble), decimals=4)
    opt['delta'] = np.around(greeks.delta.values.astype(np.doubl
e), decimals=4)
    opt['vega'] = np.around(greeks.vega.values.astype(np.double)
, decimals=4)
    opt['gamma'] = np.around(greeks.gamma.values.astype(np.doubl
e), decimals=4)
    opt['theta'] = np.around(greeks.theta.values.astype(np.doubl
e), decimals=4)
    opt['rho'] = np.around(greeks.rho.values.astype(np.double),
decimals=4)

    fields = [u'ticker', u'contractType', u'strikePrice', 'forwa
rd', u'expDate', u'tradeDate',
              u'closePrice', u'settlPrice', 'spotPrice', u'iv',
              u'delta', u'vega', u'gamma', u'theta',  u'rho']
    opt = opt[fields].reset_index().set_index('ticker').sort_index()
    #opt['iv'] = opt.iv.replace(to_replace=0.0, value=np.nan)
    return opt


# 每日期权分析数据整理
def histDayGreeksIVOpt50ETF(date):
    # Uqer 计算期权的风险数据
    opt = histDayAnalysisOpt50ETF(date)

    # 整理数据部分
    opt.index = [index[-10:] for index in opt.index]
    opt = opt[['contractType','strikePrice','spotPrice','forward','expDate','closePrice','iv','delta','theta','gamma','vega','rho']]
    opt.columns = [['contractType','strike','spot','forward','expDate','close','iv','delta','theta','gamma','vega','rho']]
    opt_call = opt[opt.contractType=='CO']
    opt_put = opt[opt.contractType=='PO']
    opt_call.columns = pd.MultiIndex.from_tuples([('Call', c) for c in opt_call.columns])
    opt_call[('Call-Put', 'strike')] = opt_call[('Call', 'strike')]
    opt_call[('Call-Put', 'spot')] = opt_call[('Call', 'spot')]
    opt_call[('Call-Put', 'forward')] = opt_call[('Call', 'forward')]
    opt_put.columns = pd.MultiIndex.from_tuples([('Put', c) for c in opt_put.columns])
    opt = concat([opt_call, opt_put], axis=1, join='inner').sort_index()
    opt = opt.set_index(('Call','expDate')).sort_index()
    opt = opt.drop([('Call','contractType'), ('Call','strike'), ('Call','forward'), ('Call','spot')], axis=1)
    opt = opt.drop([('Put','expDate'), ('Put','contractType'), ('Put','strike'), ('Put','forward'), ('Put','spot')], axis=1)
    opt.index.name = 'expDate'
    ## 以上得到完整的历史某日数据，格式简洁明了
    return opt


# 做图展示某一天的隐含波动率微笑
def histDayPlotSmileVolatilityOpt50ETF(date):
    cal = Calendar('China.SSE')
    if not cal.isBizDay(date):
        print date, ' is not a trading day!'
        return

    # Uqer 计算期权的风险数据
    opt = histDayGreeksIVOpt50ETF(date)
    spot = opt[('Call-Put', 'spot')].values[0]
    # 下面展示波动率微笑
    exp_dates = np.sort(opt.index.unique())
    ## ---------------------------------------------
    fig = plt.figure(figsize=(10,8))
    fig.set_tight_layout(True)
    for i in range(exp_dates.shape[0]):
        date = exp_dates[i]
        ax = fig.add_subplot(2,2,i+1)
        opt_date = opt[opt.index==date].set_index(('Call-Put', 'strike'))
        opt_date.index.name = 'strike'
        ax.plot(opt_date.index, opt_date[('Call', 'iv')], '-o')
        ax.plot(opt_date.index, opt_date[('Put', 'iv')], '-s')
        (y_min, y_max) = ax.get_ylim()
        ax.plot([spot, spot], [y_min, y_max], '--')
        ax.set_ylim(y_min, y_max)
        ax.legend(['call', 'put'], loc=0)
        ax.grid()
        ax.set_xlabel(u"strike")
        ax.set_ylabel(r"Implied Volatility")
        plt.title(exp_dates[i])
```

```py
```

每日期权风险数据整理

histDayGreeksIVOpt50ETF(Date(2015,10,12)).head()

~~~
| | Call | Call-Put | Put |
| --- | -- |
| | close | iv | delta | theta | gamma | vega | rho | strike | s
pot | forward | close | iv | delta | theta | gamma | vega | rho
|
| expDate |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 2015-10-28 | 0.4331 | 0.4790 | 0.9830 | -0.1615 | 0.1779 | 0.0
208 | 0.0842 | 1.85 | 2.288 | 2.283 | 0.0015 | 0.4793 | -0.0170
| -0.1059 | 0.1783 | 0.0208 | -0.0019 |
| 2015-10-28 | 0.3821 | 0.4772 | 0.9692 | -0.2310 | 0.2949 | 0.0
343 | 0.0850 | 1.90 | 2.288 | 2.283 | 0.0029 | 0.4765 | -0.0306
| -0.1725 | 0.2939 | 0.0341 | -0.0034 |
| 2015-10-28 | 0.3335 | 0.4485 | 0.9568 | -0.2740 | 0.4144 | 0.0
453 | 0.0859 | 1.95 | 2.288 | 2.283 | 0.0040 | 0.4473 | -0.0428
| -0.2129 | 0.4124 | 0.0450 | -0.0048 |
| 2015-10-28 | 0.2874 | 0.4218 | 0.9381 | -0.3289 | 0.5861 | 0.0
603 | 0.0862 | 2.00 | 2.288 | 2.283 | 0.0058 | 0.4220 | -0.0620
| -0.2690 | 0.5866 | 0.0604 | -0.0069 |
| 2015-10-28 | 0.2420 | 0.2613 | 0.9773 | -0.1349 | 0.4175 | 0.0
266 | 0.0929 | 2.05 | 2.288 | 2.283 | 0.0077 | 0.3873 | -0.0849
| -0.3130 | 0.8130 | 0.0768 | -0.0094 |

期权的隐含波动率微笑

+ 下图中，竖直虚线表示当日的标的50ETF收盘价
+ 实际上计算PCIVD就是仅仅考虑竖直虚线附近的平值期权
+ 看跌看涨隐含波动率微笑曲线中间的 Gap 的变化，正是我们关注点

```py
histDayPlotSmileVolatilityOpt50ETF(Date(2015,10,12))
~~~



```
def histDayPCIVD50ETF(date):
    ## PCIVD: Put Call Implied Volatility Diff；
    ## 看跌看涨期权隐含波动率价差，选取平值附近的近月和次近月合约构建
    ## 看跌和看涨期权的隐含波动率指数，PCIVD即为两指数之差。
    # Uqer 计算期权的风险数据
    opt = histDayGreeksIVOpt50ETF(date)
    # 下面展示波动率微笑
    exp_dates = np.sort(opt.index.unique())[0:2]
    pcivd = pd.DataFrame(0.0, index=map(Date.toDateTime, [date]), columns=['nearPCIVD','nextPCIVD'])
    pcivd.index.name = 'date'

    ivd = []
    for epd in exp_dates:
        opt_epd = opt[opt.index==epd]
        opt_epd[('Call-Put', 'diffKF')] = np.abs(opt_epd[('Call-Put', 'strike')] - opt_epd[('Call-Put', 'spot')])
        opt_epd = opt_epd.set_index(('Call-Put', 'strike'))
        opt_epd.index.name = 'strike'
        opt_epd = opt_epd.sort([('Call-Put', 'diffKF')]).head(2)
        ivd_epd = opt_epd[('Put', 'iv')].mean() - opt_epd[('Call', 'iv')].mean()
        ivd.append(ivd_epd)
    pcivd.ix[Date.toDateTime(date)] = ivd
    return pcivd

def histDayPCIVD50ETF_check(date):
    ## PCIVD: Put Call Implied Volatility Diff；
    ## 看跌看涨期权隐含波动率价差，选取平值附近的近月和次近月合约构建
    ## 看跌和看涨期权的隐含波动率指数，PCIVD即为两指数之差。
    # Uqer 计算期权的风险数据
    opt = histDayGreeksIVOpt50ETF(date)
    # 下面展示波动率微笑
    exp_dates = np.sort(opt.index.unique())[0:2]
    pcivd = pd.DataFrame(0.0, index=map(Date.toDateTime, [date])
, columns=['nearPCIVD', 'nearPutIV', 'nearCallIV','nextPCIVD', '
nextPutIV', 'nextCallIV'])
    pcivd.index.name = 'date'

    ivd = []
    for epd in exp_dates:
        opt_epd = opt[opt.index==epd]
        opt_epd[('Call-Put', 'diffKF')] = np.abs(opt_epd[('Call-
Put', 'strike')] - opt_epd[('Call-Put', 'spot')])
        opt_epd = opt_epd.set_index(('Call-Put', 'strike'))
        opt_epd.index.name = 'strike'
        opt_epd = opt_epd.sort([('Call-Put', 'diffKF')]).head(2)
        ivd_epd = opt_epd[('Put', 'iv')].mean() - opt_epd[('Call'
, 'iv')].mean()
        ivd.append(ivd_epd)
        ivd.append(opt_epd[('Put', 'iv')].mean())
        ivd.append(opt_epd[('Call', 'iv')].mean())
    pcivd.ix[Date.toDateTime(date)] = ivd
    return pcivd

def histPCIVD50ETF(beginDate, endDate):
    begin = Date.fromDateTime(beginDate)
    end = Date.fromDateTime(endDate)

    cal = Calendar('China.SSE')
    dates = cal.bizDatesList(begin, end)
    pcivd = pd.DataFrame()
    for dt in dates:
        pcivd_dt = histDayPCIVD50ETF(dt)
        pcivd = concat([pcivd, pcivd_dt])
    pcivd['nearDiff'] = pcivd['nearPCIVD'].diff()
    pcivd['nextDiff'] = pcivd['nextPCIVD'].diff()
    return pcivd

def histPCIVD50ETF_check(beginDate, endDate):
    begin = Date.fromDateTime(beginDate)
    end = Date.fromDateTime(endDate)

    cal = Calendar('China.SSE')
    dates = cal.bizDatesList(begin, end)
    pcivd = pd.DataFrame()
    for dt in dates:
        pcivd_dt = histDayPCIVD50ETF_check(dt)
        pcivd = concat([pcivd, pcivd_dt])
    pcivd['nearPutDiff'] = pcivd['nearPutIV'].diff()
    pcivd['nearCallDiff'] = pcivd['nearCallIV'].diff()
    pcivd['nextPutDiff'] = pcivd['nextPutIV'].diff()
    pcivd['nextCallDiff'] = pcivd['nextCallIV'].diff()
    return pcivd
```

计算PCIVD

- 期权自15年2月9号上市
- 此处计算得到的数据可以用在后面几条策略中

结果中的列分别为：

- nearPCIVD：当月PCIVD
- nextPCIVD：次月PCIVD
- nearDiff：当月PCIVD与前一日值的变化量
- nextDiff：次月PCIVD与前一日值的变化量

```
## PCIVD计算示例

start = datetime(2015, 2, 9)                # 回测起始时间
end   = datetime(2015, 10, 12)               # 回测结束时间
pcivd = histPCIVD50ETF(start, end)
pcivd.tail()
```

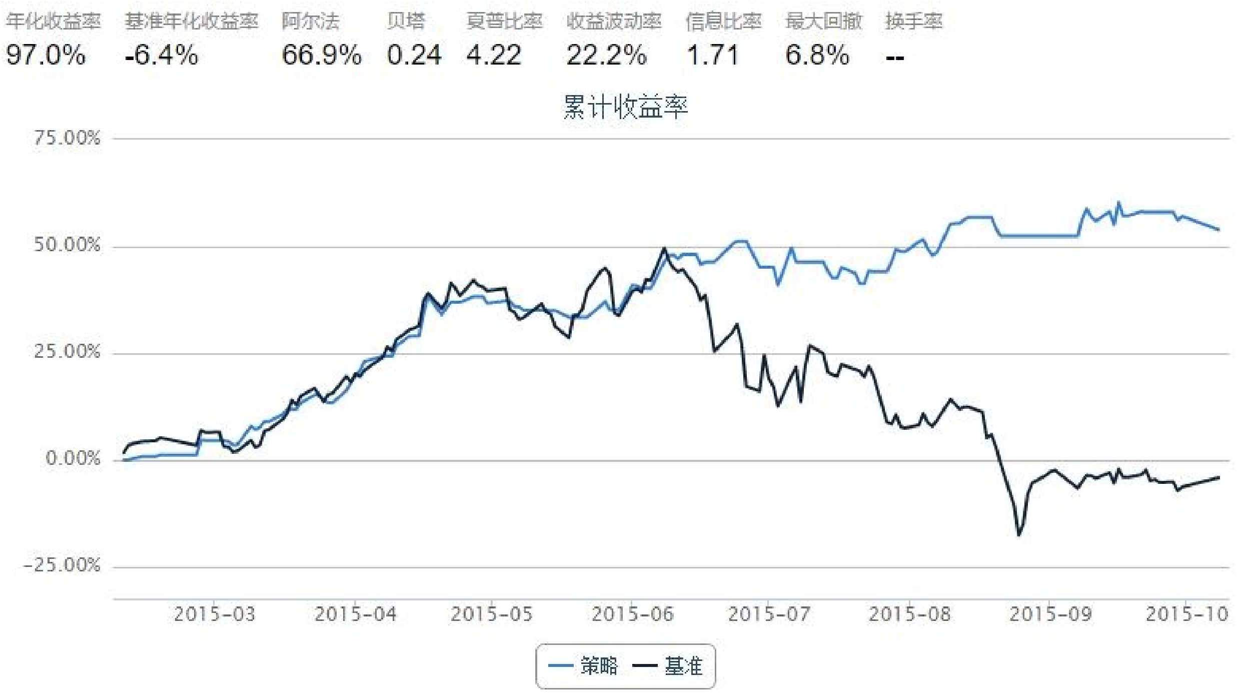
	nearPCIVD	nextPCIVD	nearDiff	nextDiff
date				
2015-09-29	0.15540	0.15915	0.02660	0.0073
2015-09-30	0.10205	0.14915	-0.05335	-0.0100
2015-10-08	0.08845	0.10645	-0.01360	-0.0427
2015-10-09	0.08320	0.10375	-0.00525	-0.0027
2015-10-12	0.04635	0.07065	-0.03685	-0.0331

2.1 结合使用当月、次月 PCIVD 的择时策略

策略思路：考虑当月 PCIVD 和 次月 PCIVD 的日变化量

- 当月 PCIVD 和 次月 PCIVD 同时变小（当月和次月的 PCIVDDiff 同时小于0），则今天全仓50ETF
- 否则，清仓观望

```
start = datetime(2015, 2, 9)                       # 回测起始时间
end  = datetime(2015, 10, 8)                       # 回测结束时间
benchmark = '510050.XSHG'                           # 策略参考标准
universe = ['510050.XSHG']                           # 股票池

capital_base = 100000                              # 起始资金
commission = Commission(0.0,0.0)
refresh_rate = 1

# pcivd = histPCIVD50ETF(start, end)

def initialize(account):                           # 初始化虚拟账户状态
    account.fund = universe[0]

def handle_data(account):                    # 每个交易日的买入卖出指令
    fund = account.fund
    #  获取回测当日的前一天日期
    dt = Date.fromDateTime(account.current_date)
    cal = Calendar('China.IB')
    last_day = cal.advanceDate(dt,'-1B',BizDayConvention.Preceding)             #计算出倒数第一个交易日
    last_day_str = last_day.strftime("%Y-%m-%d")

    # 计算买入卖出信号
    try:
        # 拿取PCIVD数据
        pcivd_near = pcivd.nearDiff.loc[last_day_str]
        pcivd_next = pcivd.nextDiff.loc[last_day_str]
        long_flag = True if pcivd_near < 0 and pcivd_next < 0 else False
    except:
        long_flag = False

    if long_flag:
        # 买入时，全仓杀入
        try:
            approximationAmount = int(account.cash / account.referencePrice[fund] / 100.0) * 100
            order(fund, approximationAmount)
        except:
            return
    else:
        # 卖出时，全仓清空
        order_to(fund, 0)
```



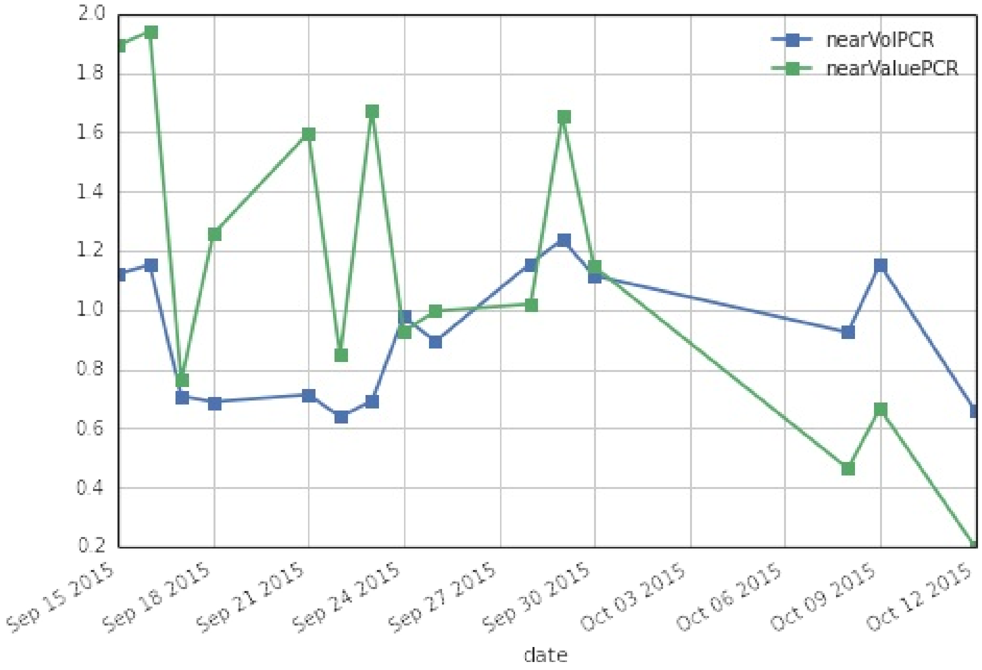
年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
97.0%	-6.4%	66.9%	0.24	4.22	22.2%	1.71	6.8%	--

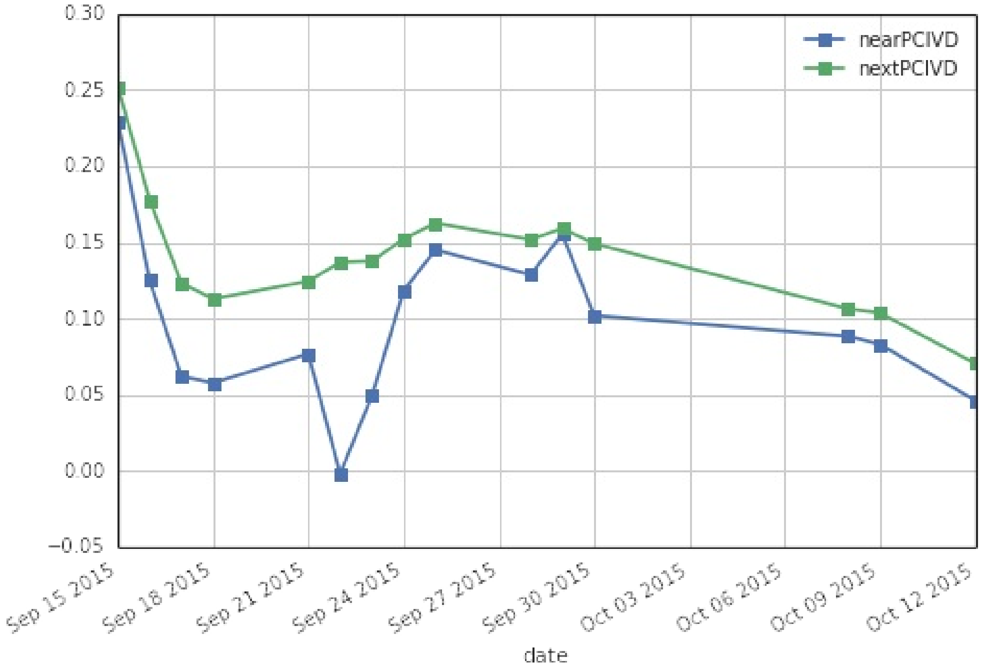
PCR 和 PCIVD 的良好择时效果表明，虽然回测时间短，但它们均可以通过期权市场的信息来给出在现货市场的买卖择时信号，必要时建议我们空仓

3. 监视最近 PCR 和 PCIVD 走势

- 每日监视 PCR 和 PCIVD 近期走势，指导次日操作
- 如果 PCR 和 PCIVD 的值降低，那么我们就在第二天买入

```
cal = Calendar('China.IB')

# Dates
end = Date.todaysDate()
end = cal.advanceDate(end,'-1B',BizDayConvention.Preceding)
# 这里结束点选择昨天，因为DataAPI的今日数据要到收盘后比较晚才能拿到；实际中
可以自己调整
start = cal.advanceDate(end,'-15B',BizDayConvention.Preceding)
# 开始点为七天前

## 计算 PCR 和 PCIVD
start = start.toDateTime()
end  = end.toDateTime()

hist_pcr = histPCR50ETF(start, end)       # 计算PCR
hist_pcivd = histPCIVD50ETF(start, end)  # 计算PCIVD

hist_pcr[['nearVolPCR', 'nearValuePCR']].plot(style='s-')
hist_pcivd[['nearPCIVD', 'nextPCIVD']].plot(style='s-')

<matplotlib.axes.AxesSubplot at 0x852ba90>
```





PCIVD 图中，近月期权的 PCIVD 在行权日为0，需要注意；行权日附近，可以以次近月期权的 PCIVD 走势为参考

期权头寸计算

> 来源：https://uqer.io/community/share/54c846cdf9f06c276f651a50

版本：1.0

作者：李丞

联系：cheng.li@datayes.com

在本篇中，我们假设客户已经拥有了自己的期权头寸，希望利用量化实验室的功能进行风险监控。

数据准备

用户的期权头寸数据保存在期权头寸.csv文件中(请点击链接下载），样例如下：

```
optionData = pd.read_csv(u'期权头寸.csv', encoding='gbk', parse_dates = [3], dtype = {u'代码':str})
pd.options.display.float_format = '{:,.2f}'.format
print(optionData)

        代码  方向   行权价         到期时间     头寸
0   000001  C 2.20 2015-02-19  1000
1   000002  C 2.10 2015-02-19  -500
2   000003  P 1.90 2015-03-19  1000
3   000004  P 1.80 2015-03-19  -500
```

计算方法

这里我们简单展示如何用内置函数，方便的计算基于Black-Scholes模型的价格以及风险值：

```
def processOptionBook(optionData):

    # 由于我们现在还无法获取市场数据，制作一些假的市场数据数据
    # 例如波动率水平，利率水平
    vol = 0.3 * np.random.random(len(optionData)) + 0.2
    spot = 2.0
    riskFree = 0.04
    dividend = 0.0
    evaluationDate = Date.todaysDate()

    # 根据用户的输入计算期权的价格以及各种greeks
    t2m = (optionData[u'到期时间'].apply(lambda x:Date(x.year,x.month,x.day)) - evaluationDate)/ 365.0
    optionType = optionData[u'方向'].apply(lambda x: x=='C' and 1 or -1)
    strike = optionData[u'行权价']

    calculateResult = BSMPrice(optionType,strike,spot,riskFree,dividend, vol,t2m)

    # 整理数据
    calculateResult = calculateResult.multiply(optionData[u'头寸'].values, axis = 0)
    calculateResult.index = optionData[u'代码']
    portfolio = pd.DataFrame(dict(np.sum(calculateResult)), index = ['portfolio'])
    calculateResult = calculateResult.append(portfolio)
    calculateResult.index.name = u'代码'
    calculateResult = calculateResult.reindex_axis(['price', 'delta', 'gamma', 'rho', 'theta', 'vega'], axis = 1)

    return calculateResult
```

```
res= processOptionBook(optionData)
print(res)
```

```
          price   delta     gamma    rho   theta   vega
代码
000001    27.36  222.15  1,317.92  26.27 -550.43 149.48
000002    -5.13  -94.07 -1,326.48 -11.53  116.11 -67.71
000003    24.81 -238.18  1,822.42 -70.03 -168.14 231.44
000004    -8.43   71.48   -514.95  21.15   82.48 -84.38
portfolio 38.62  -38.62  1,298.91 -34.13 -519.98 228.84
```

期权探秘1

> 来源：https://uqer.io/community/share/54b39784f9f06c276f651a0e

版本：1.0

作者：李丞

联系：cheng.li@datayes.com

1. 什么是期权

期权是期权买方与期权卖方达成的未来交割协议：期权的卖方承诺在未来的指定到期日向期权买方以指定的敲定价格出售（或者买入）指定的标的，这个合约是期权卖方的义务；同时是期权买方的权利。

- 当期权卖方有标的的卖出义务，期权买方有标的的买入权利时，这个期权称为看涨期权或者买入期权
- 当期权卖方有标的的买入义务，期权买方有标的的卖出权利时，这个期权称为看跌期权或者卖出期权

这样的期权可以在到期日之间在投资者之间自由的交易，这个价格的形成机制成为了期权投资者、套利者、套期保值者的研究核心。

首先我们从期权的买方（投资者）出发，研究买入期权的支付结构（Payoff):

- 如果到期日，标的资产价格高于敲定价格，那么投资者会很高兴行使买入的权利（Exercise）；
- 反之在到期日，标的资产价格低于敲定价格，投资者宁可选择从市场上直接买入标的而不是行权。

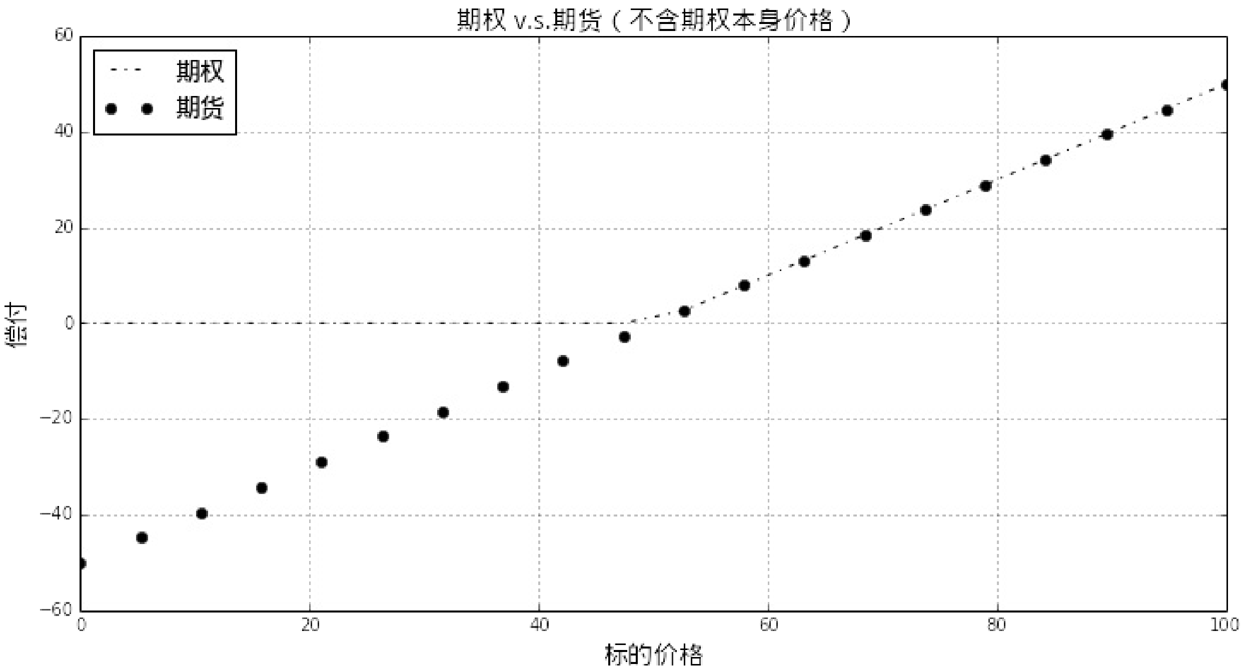
如果我们假设投资者在行权之后会直接在市场上卖出标的，那么投资者的最终到期收益很容易的用下面的式子表示：

```
Payoff=MAX(S-K,0)
```

这里面 `S` 就是到期日标的价格， `K` 是敲定价格。

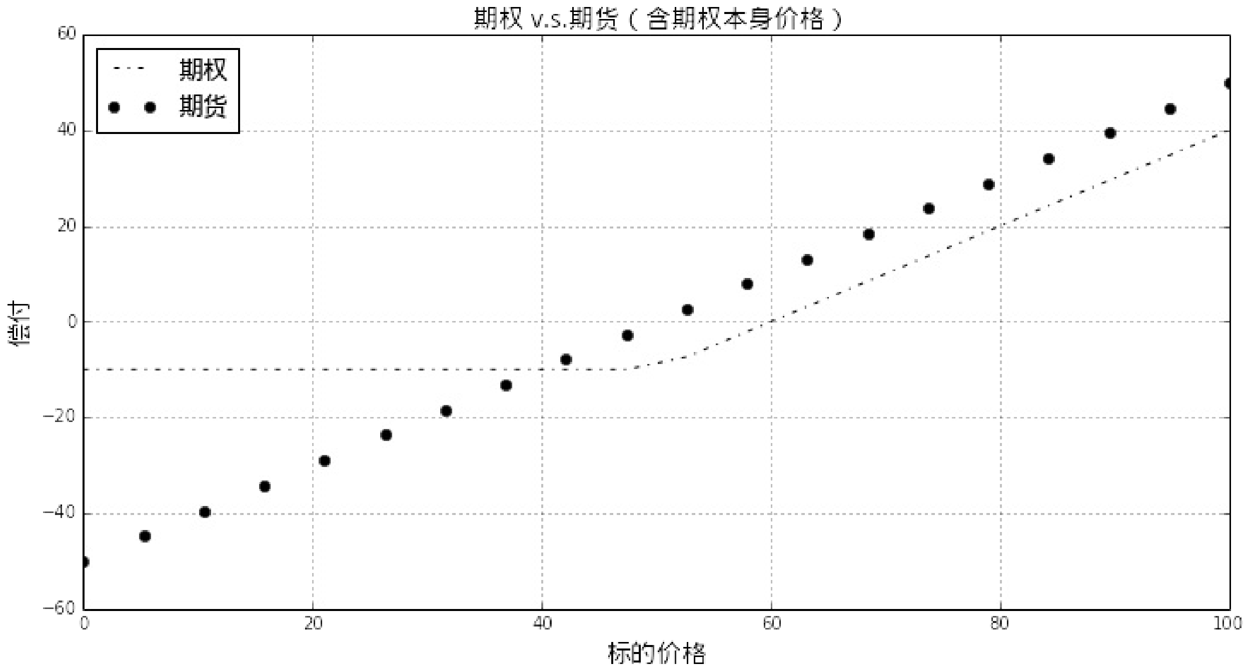
考虑敲定价格为50的情形，让我们绘制支付函数的具体形状：

```
import numpy as np
from matplotlib import pylab

def future(S):
    return S - 50.0

def call(S):
    return max(S - 50.0,0.0)

callfunc = np.frompyfunc(call, 1, 1)
futurefunc = np.frompyfunc(future, 1, 1)

spots = np.linspace(0,100,20)
pylab.figure(figsize=(12,6))
pylab.plot(spots, callfunc(spots), 'k-.')
pylab.plot(spots, futurefunc(spots), 'ko')
font.set_size(15)
pylab.legend([u'期权',u'期货'], prop = font, loc = 'best')
pylab.title(u'期权 v.s.期货（不含期权本身价格）', fontproperties = fo
nt)
pylab.xlabel(u'标的价格', fontproperties = font)
pylab.ylabel(u'偿付', fontproperties = font)
pylab.grid()
```



很容易的可以从上图看到，期权相对于期货，具有上涨的完全收益，同时规避了下跌的风险。当然这样的“权利”自然不会是免费的，期权的价格就是期权买方需要付出的代价。我们这里假设期权的价格为10：

```
def future(S):
    return S - 50.0

def call(S):
    return max(S - 50.0,0.0) - 10.0

callfunc = np.frompyfunc(call, 1, 1)

spots = np.linspace(0,100,20)
pylab.figure(figsize=(12,6))
pylab.plot(spots, callfunc(spots), 'k-.')
pylab.plot(spots, futurefunc(spots), 'ko')
pylab.legend([u'期权',u'期货'], prop = font, loc = 'best')
pylab.title(u'期权 v.s.期货（含期权本身价格）', fontproperties = fon
t)
pylab.xlabel(u'标的价格', fontproperties = font)
pylab.ylabel(u'偿付', fontproperties = font)
pylab.grid()
```



上图中可以清晰的看到，期权价格会直接影响到期的损益。在我们的这个例子中，除非标的价格到期涨到超过60元，客户不会有正的收益！

2. 影响期权价格的因素

这里我们暂不讨论期权的定价方法，这个我们将在后续的报告中讨论。这里我们首先讨论影响期权价格的几个显著要素。这里我们仍然选取上面的看涨期权的例子，先使用最常见的Black - Scholes模型[1]（关于这个模型的详细介绍会在以后给出）。这里例子的数据取自经典教材[2]

- 期权类型：看涨期权

- 敲定价格：50
- 标的价格：50
- 无风险利率： 5%
- 红利率：0%
- 波动率：30%
- 到期时间：1年

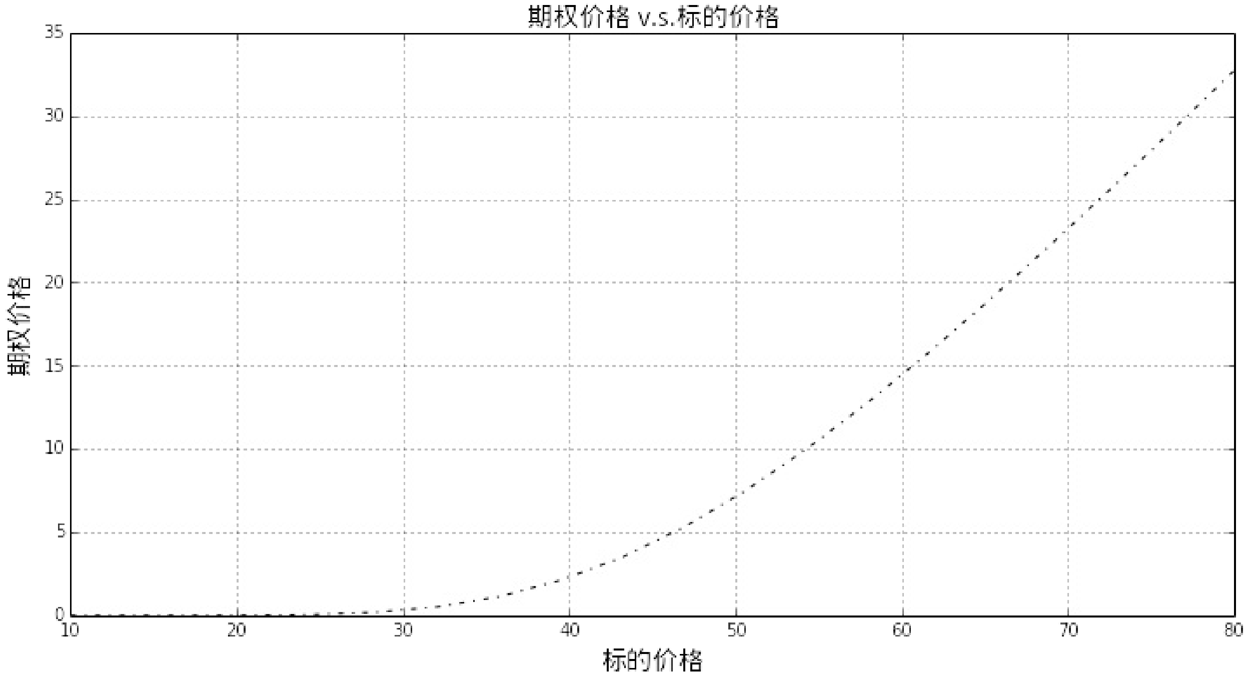
在这篇报告中我们将只讨论：标的价格、敲定价格以及到期时间的影响。剩余的因子我们会在以后讨论。

2.1 标的价格

显然的，随着标的价格的上升，客户手上的期权未来值得行权的可能性越高，并且行权以后获得的收益也越高。由此可知期权价格是标的价格单调增函数：

```
optinType = Option.Call
strike = 50.0
riskFree = 0.05
dividend = 0.0
volatility = 0.3
maturity = 1.0
```

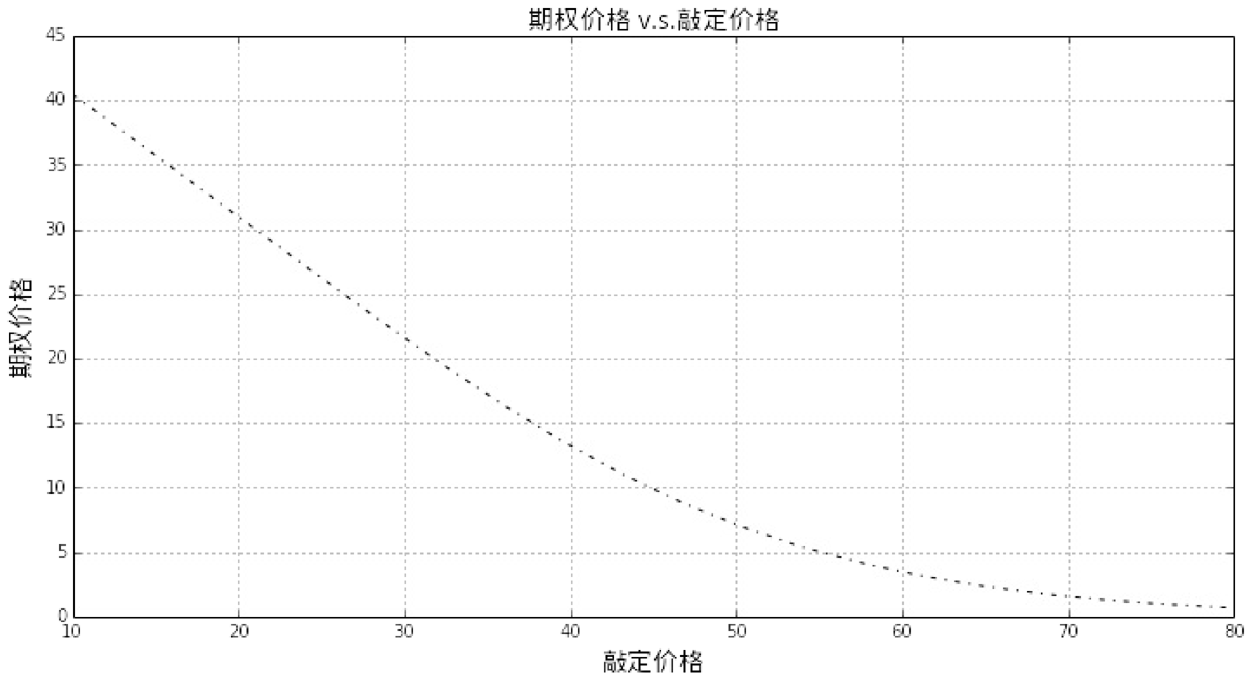
```
spots = np.linspace(10,80,20)
prices = BSMPrice(optinType, strike, spots, riskFree, dividend,
volatility, maturity)['price']
pylab.figure(figsize=(12,6))
pylab.plot(spots, prices, 'k-.')
pylab.title(u'期权价格 v.s.标的价格', fontproperties = font)
pylab.xlabel(u'标的价格', fontproperties = font)
pylab.ylabel(u'期权价格', fontproperties = font)
pylab.grid()
```



2.2 敲定价格

与敲定价格恰恰相反，敲定价格与看涨期权价格是反向关系：敲定价格越高，看涨期权行权的可能性越小；即使可以行权，获得收益也相对于低敲定价格的期权低

```
strikes = np.linspace(10,80,20)
spot = 50.0
prices = BSMPrice(optinType, strikes, spot, riskFree, dividend,
volatility, maturity)['price']
pylab.figure(figsize=(12,6))
pylab.plot(strikes, prices, 'k-.')
pylab.title(u'期权价格 v.s.敲定价格', fontproperties = font)
pylab.xlabel(u'敲定价格', fontproperties = font)
pylab.ylabel(u'期权价格', fontproperties = font)
pylab.grid()
```

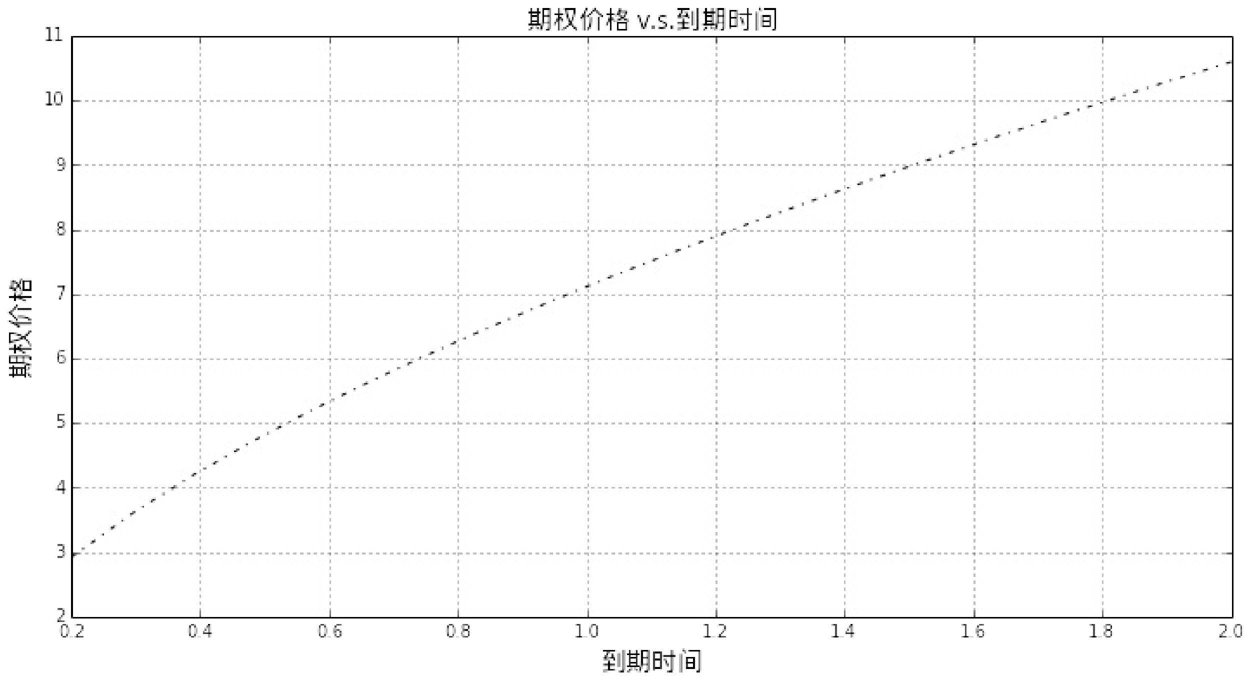


2.3 到期时间

到期时间的作用不像标的价格以及行权价格那样一目了然，但是在假设的情形下吗，仍然是可以分析的：

如果离到期时间还有距离，那么意味着即使现在的标的价格仍然低于到期价格，仍然有机会“咸鱼翻身”。这时候到期时间对于期权价格的贡献是正的。

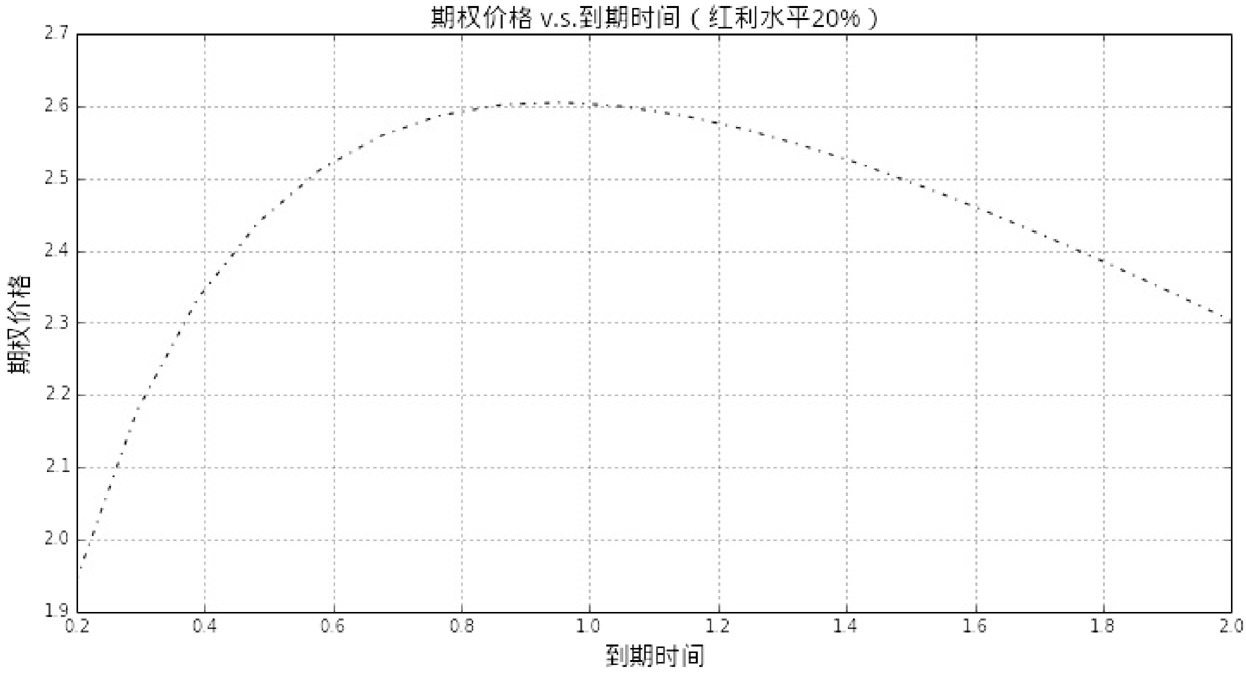
```
spot = 50
strike = 50
maturities = np.linspace(0.2,2.0,20)
prices = BSMPrice(optinType, strike, spot, riskFree, 0.0, volati
lity, maturities)['price']
pylab.figure(figsize=(12,6))
pylab.plot(maturities, prices, 'k-.')
pylab.title(u'期权价格 v.s.到期时间', fontproperties = font)
pylab.xlabel(u'到期时间', fontproperties = font)
pylab.ylabel(u'期权价格', fontproperties = font)
pylab.grid()
```



直接可以看到，期权的价格是到期时间单调增函数。

但是期权到期时间的影响没有那么简单，让我们看下面这张图：这里假设红利比率为20%，期权价格在1年左右达到最大值，随后开始下降。这里面的原因比较复杂，留待我们以后的报告中解释。

```
spot = 50
strike = 50
maturities = np.linspace(0.2,2.0,20)
prices = BSMPrice(optinType, strike, spot, riskFree, 0.2, volatility, maturities)['price']
pylab.figure(figsize=(12,6))
pylab.plot(maturities, prices, 'k-.')
pylab.title(u'期权价格 v.s.到期时间（红利水平20%）', fontproperties = font)
pylab.xlabel(u'到期时间', fontproperties = font)
pylab.ylabel(u'期权价格', fontproperties = font)
pylab.grid()
```


期权价格 v.s.到期时间（红利水平20%）
期权价格
到期时间
2.7
2.6
2.5
2.4
2.3
2.2
2.1
2.0
1.9
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
1.8
2.0

期权探秘2

> 来源：https://uqer.io/community/share/54c479f1f9f06c276f651a4a

版本：1.0

作者：李自龙

联系：zilong.li@datayes.com

上一篇中，李博简单介绍了期权和影响期权价格的各种因素。本篇中，我们重点介绍期权与不同资产的组合能够给我们带来什么样新颖的盈利方式，例如：

- 期权 + 零息债券
- 期权 + 标的资产
- 同一标的资产上的多个期权

如此通过构造包含期权的头寸组合，能够实现丰富多彩的未来收益形式，给我们的想象力提供了最大的舞台，请看下文！

```
import numpy as np
from matplotlib import pylab
```

1. 期权 + 零息债券 = 保本债券

期权推出后，银行可以向客户提供以下形式的价格1000元的理财产品：

- 面值为1000元的三年期零息债券
- 标的为股票组合的三年期欧式平值看涨期权

该产品三年到期后：如果股票组合的价值增长，通过行使期权，投资者将得到1000元债券收益外加1000元股票组合头寸对应的增值部分；如果股票组合下跌，期权便没有价值，但是投资者仍然可以得到1000元的债券收益。总的来看，投资者此时的1000元投资在三年后是保本的，即本金不会有任何风险，这就是保本债券（principle-protected notes）的由来。

对于银行来说，发行保本债券也不是赔本的买卖。如果三年期复利利率为5%，那么三年后1000元的今日帖现值为1000e−0.05×3=860.71元。也就是说，投资者今日花1000元买入该产品，在完全无风险的情况下，银行也可以拿出差额139.29元来购买该产品中的看涨期权。我们来看看该看涨期权在今日价值几何（利用BSM期权定价公式，假定股票组合的波动率为10%、收益率为1.5%，无风险利率为5%）：

```
price = BSMPrice(Option.Call, 1000, 1000, 0.05, 0.015, 0.1, 3)['price']
print "看涨期权价格:  ", price[1]

看涨期权价格:   121.470487448
```

这就是说，差额139.29元完全足够购买该理财产品中的必须的看涨期权，银行在推出该保本债券理财产品时，其成本是低于售价1000元的。聪明的投资者可能会想，比购买该理财产品更好地投资做法是：自己购买标的期权，并将剩余的本金投入到无风险投资上。但这个想法有点理想化：

- 普通投资者入市期权门槛太高，也会面临更大的买入卖出差价
- 剩余本金的投资利息比银行要低

因此银行推出这一产品可以给投资者带来收益，同时自己也能得到盈利。由于期权价格和标的资产波动率密切相关，如果波动率过高，差额139.29元可能不够购买这一看涨期权：

```
price = BSMPrice(Option.Call, 1000, 1000, 0.05, 0.015, 0.2, 3)['price']
print "波动率为20%时看涨期权价格:  ", price[1]

波动率为20%时看涨期权价格:   178.183351319
```

此时银行的可以通过购买更高行权价格的期权：

```
price = BSMPrice(Option.Call, 1100, 1000, 0.05, 0.015, 0.2, 3)['price']
print "波动率为20%、行权价为1100时的看涨期权价格:  ", price[1]

波动率为20%、行权价为1100时的看涨期权价格:   135.48518416
```

此时，三年后，只有该标的资产的价格增长超过10%时，投资者才能获利。

在此基础上（标的波动率为20%、期权行权价为1100），我们讨论投资者今日投入1000元、三年后到期的收益情况：如果标的组合三年期间价格增加30%，那么投资者到期收益1200元；这一收益仍大于将1000元直接存入银行的收益 `1000e ** 0.05*3=1161.83` 元。总之，投资者投资这一产品的收益和标的资产的价格增长情况息息相关；如果标的资产为上证50ETF，那么在接下来普遍看好的牛市中该产品应该可以带给投资者不错的收益，且投资该产品是保本的。

2. 股票与单一期权相组合的策略

前一节我们讨论了，单一期权与零息债券的组合，本节我们将讨论单一期权与股票的组合。

```
# 定义看涨看跌期权到期收益函数
def call(S):
    return max(S - 100.0,0.0) - 40.0

def put(S):
    return max(100.0 - S,0.0) - 40.0

callfunc = np.frompyfunc(call, 1, 1)
putfunc = np.frompyfunc(put, 1, 1)
```

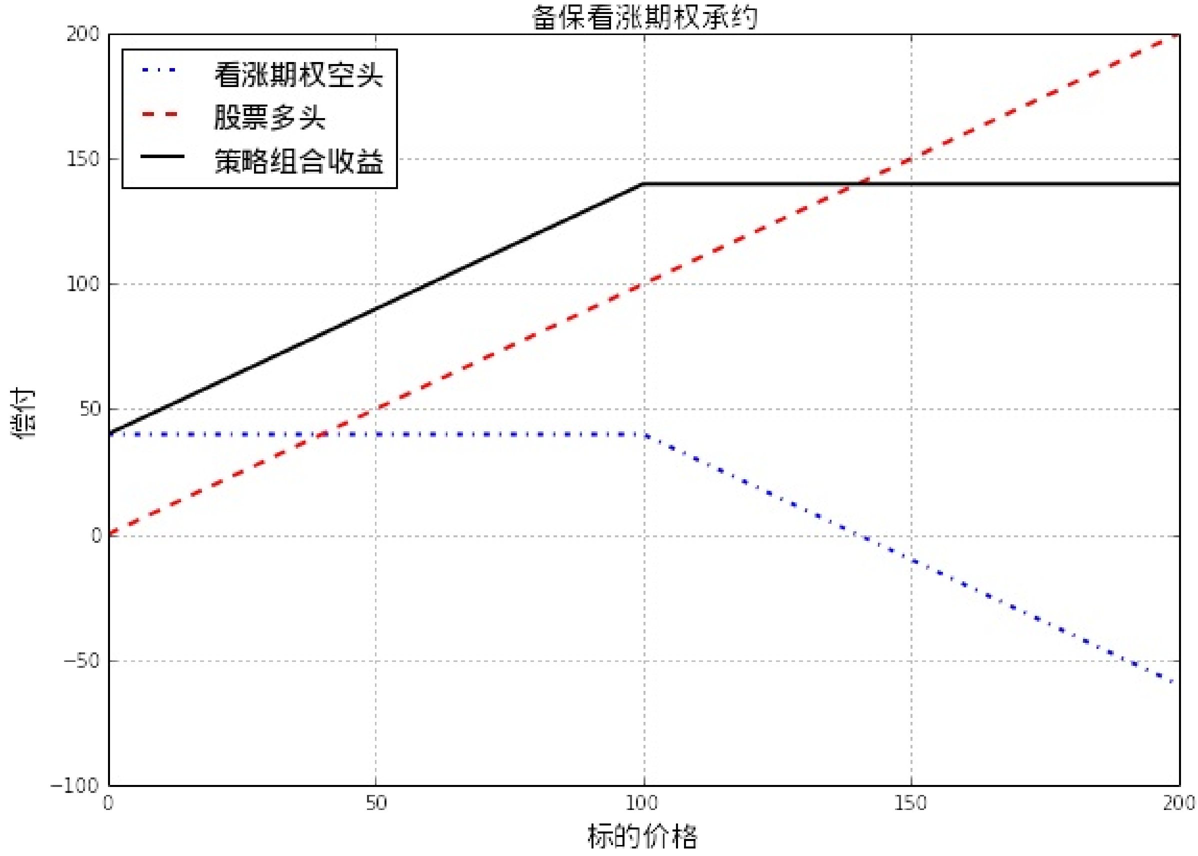
2.1 备保看涨期权承约（writing covered call）

策略构造：

- 卖出以股票为标的的看涨期权
- 持有相应于期权空头的该标的股票

策略到期收益和股票价格的关系如下图：

```
spots = np.linspace(0,200,21)

pylab.figure(figsize=(10,7))
pylab.plot(spots, -callfunc(spots), 'b-.',linewidth = 2)
pylab.plot(spots, spots, 'r--',linewidth = 2)
pylab.plot(spots, -callfunc(spots) + spots, 'k-',linewidth = 2)
font.set_size(15)
pylab.legend([u'看涨期权空头',u'股票多头',u'策略组合收益'], prop = fo
nt, loc = 'best')
pylab.title(u'备保看涨期权承约', fontproperties = font)
pylab.xlabel(u'标的价格', fontproperties = font)
pylab.ylabel(u'偿付', fontproperties = font)
pylab.grid()
```



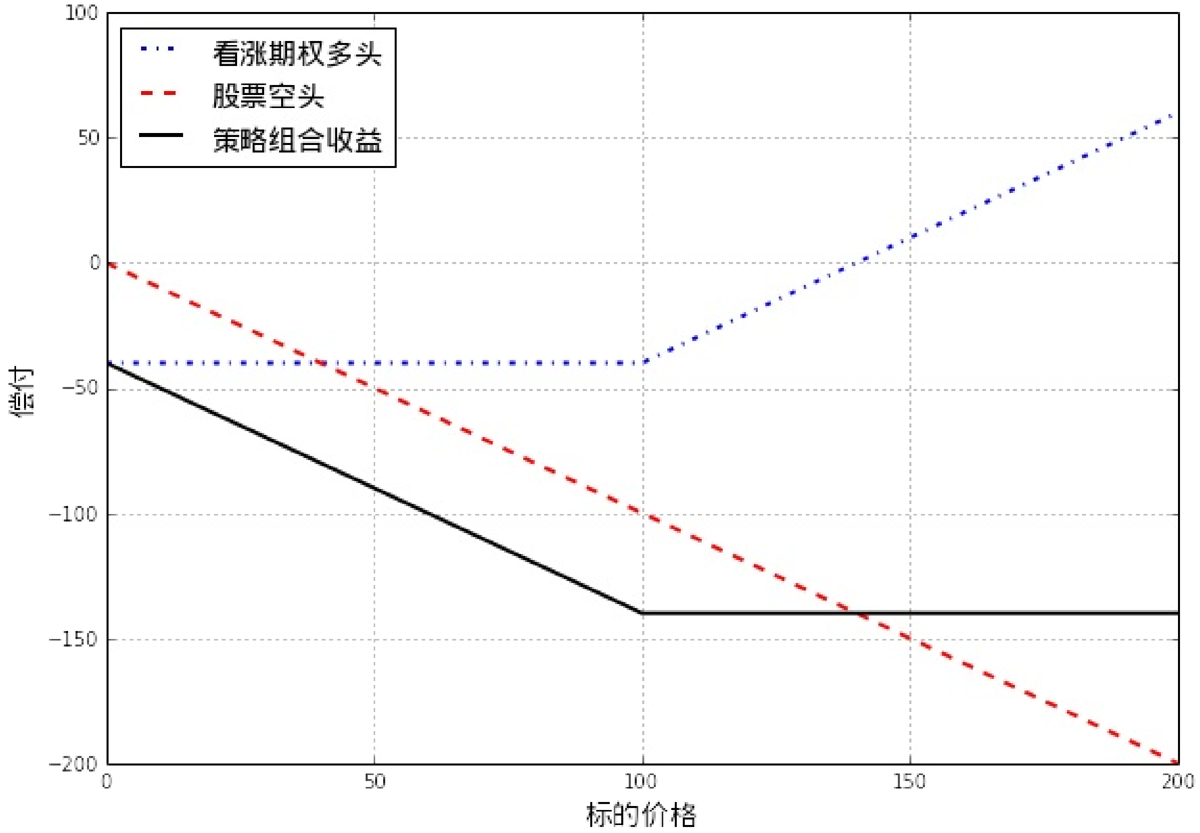
由上图中所见，在股票价格急剧上涨时，投资者持有的股票保护（cover）了其看涨期权空头将带来的损失。可以看出，采用Covered Call策略的投资者对未来该股票的表现持中性态度，认为该股票价格在未来一段时间内将保持在当前价格附件区间。

和Covered Call相反，如果对未来某股票的表现持中性或者看跌态度，投资者利用该股票看涨期权构建策略：

- 持有以股票为标的的看涨期权
- 卖空相应份额的该股票

策略到期收益和股票价格的关系如下图：

```
spots = np.linspace(0,200,21)

pylab.figure(figsize=(10,7))
pylab.plot(spots, callfunc(spots), 'b-.',linewidth = 2)
pylab.plot(spots, -spots, 'r--',linewidth = 2)
pylab.plot(spots, callfunc(spots) - spots, 'k-',linewidth = 2)
font.set_size(15)
pylab.legend([u'看涨期权多头',u'股票空头',u'策略组合收益'], prop = fo
nt, loc = 'best')
pylab.xlabel(u'标的价格', fontproperties = font)
pylab.ylabel(u'偿付', fontproperties = font)
pylab.grid()
```



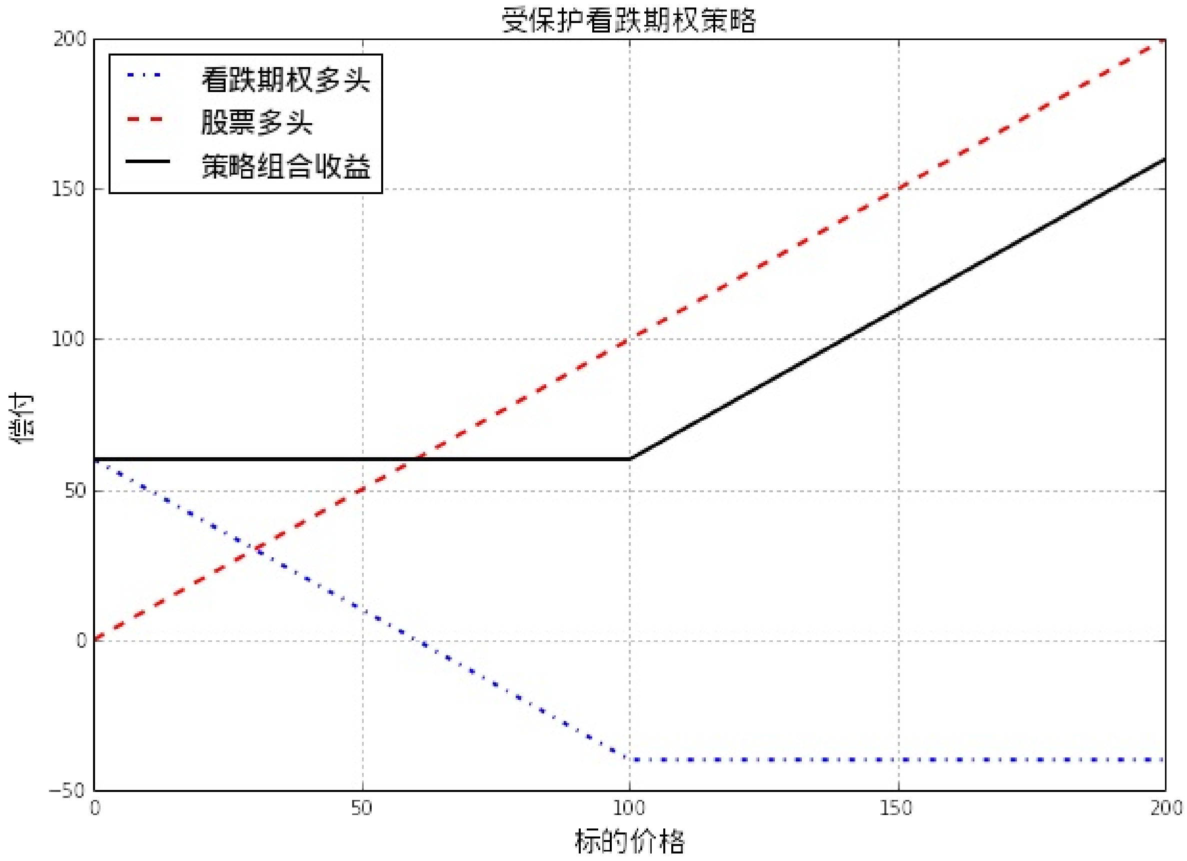
2.2 保护性看跌期权策略（Protective Put）

策略构造：

- 持有以某股票为标的的看跌期权
- 买入相应于看跌期权份额的该股票

策略到期收益和股票价格的关系如下图：

```
spots = np.linspace(0,200,21)

pylab.figure(figsize=(10,7))
pylab.plot(spots, putfunc(spots), 'b-.',linewidth = 2)
pylab.plot(spots, spots, 'r--',linewidth = 2)
pylab.plot(spots, putfunc(spots) + spots, 'k-',linewidth = 2)
font.set_size(15)
pylab.legend([u'看跌期权多头',u'股票多头',u'策略组合收益'], prop = fo
nt, loc = 'best')
pylab.title(u'受保护看跌期权策略', fontproperties = font)
pylab.xlabel(u'标的价格', fontproperties = font)
pylab.ylabel(u'偿付', fontproperties = font)
pylab.grid()
```



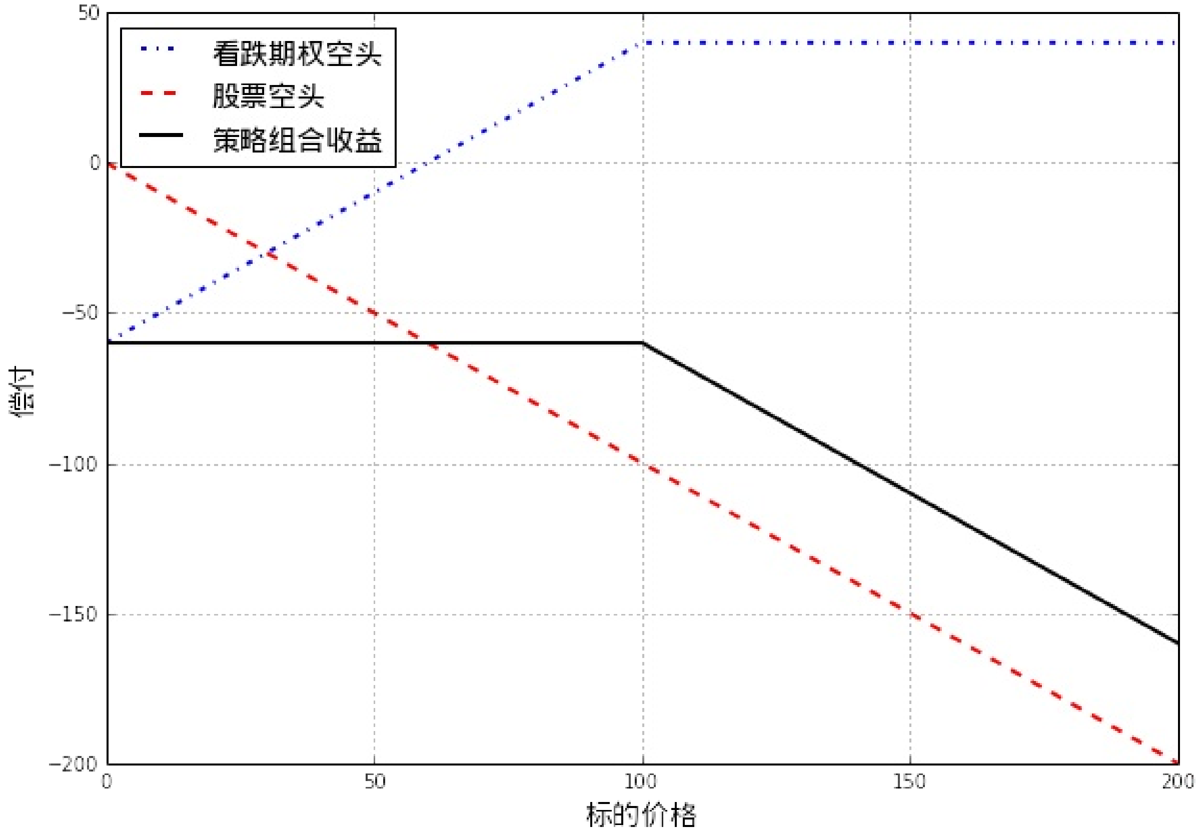
由上图可以看到，投资者对于自身持有的股票持看涨态度，但是又不愿意承担股票下跌的损失，所以采用保护性看跌期权策略（Protective Put），能够在获取股票升值收益的情况下，同时利用持有的看跌期权保护自己不受股票下跌的影响。

与保护性看跌期权策略相反，如果投资者对于股票持中性或者轻微看跌态度时，可以通过卖出看跌期权来构造如下策略：

- 卖出以某股票为标的的看跌期权
- 卖空相应于看跌期权份额的该股票

策略到期收益和股票价格的关系如下图：

```
spots = np.linspace(0,200,21)

pylab.figure(figsize=(10,7))
pylab.plot(spots, -putfunc(spots), 'b-.',linewidth = 2)
pylab.plot(spots, -spots, 'r--',linewidth = 2)
pylab.plot(spots, -putfunc(spots) - spots, 'k-',linewidth = 2)
font.set_size(15)
pylab.legend([u'看跌期权空头',u'股票空头',u'策略组合收益'], prop = fo
nt, loc = 'best')
pylab.xlabel(u'标的价格', fontproperties = font)
pylab.ylabel(u'偿付', fontproperties = font)
pylab.grid()
```



3. 差价策略（spread）

差价策略，是指将两个或者多个相同类型（同为看涨或同为看跌）期权组合在一起的期权交易策略。

```
def call(S, K, c):
    return max(S - K,0.0) - c

def put(S, K, p):
    return max(K - S,0.0) - p

callfunc = np.frompyfunc(call, 3, 1)
putfunc = np.frompyfunc(put, 3, 1)
```

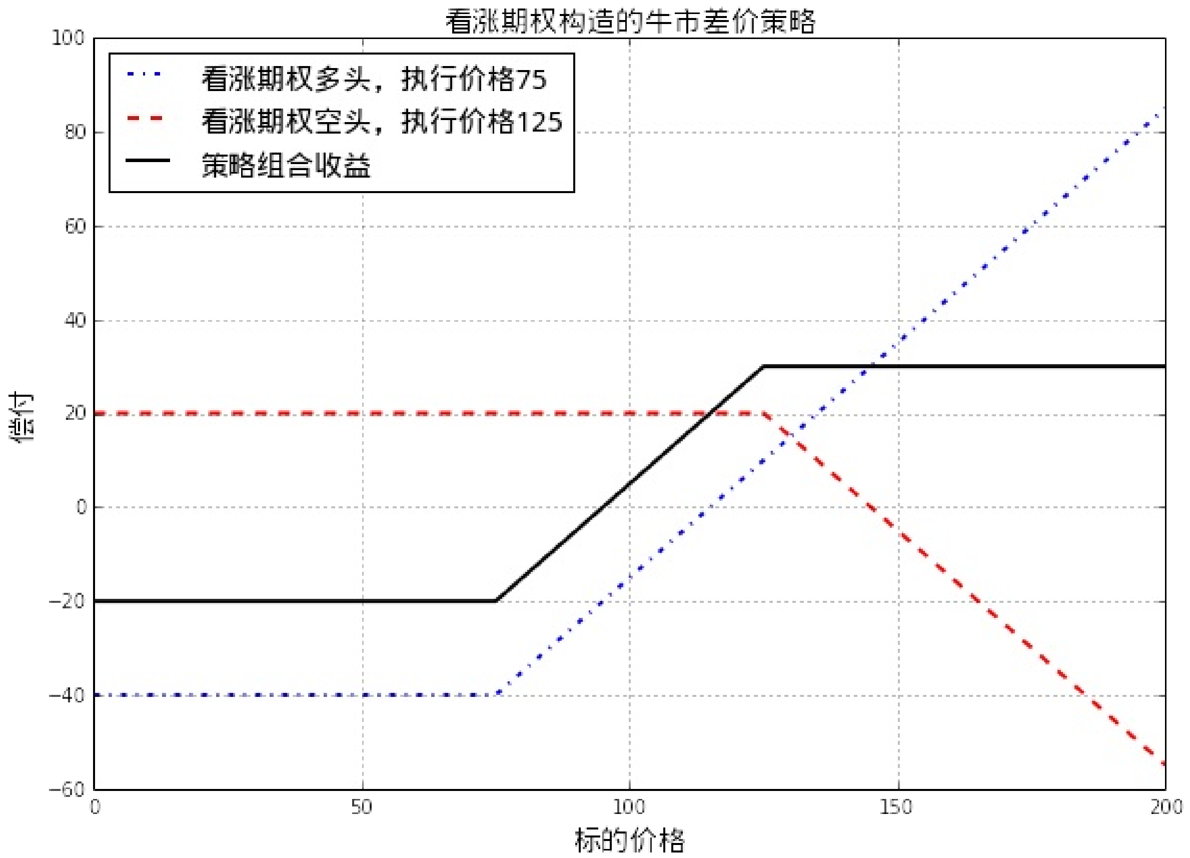
3.1 牛市差价（bull spread）

策略构造：

- 买入一份股票标的看涨期权
- 卖出同一股票标的、期权期限相同，但执行价格较高的看涨期权（对于看涨期权来说，执行价格高意味着期权费用较低）

策略到期收益和股票价格的关系如下图：

```
spots = np.linspace(0,200,41)

pylab.figure(figsize=(10,7))
pylab.plot(spots, callfunc(spots, 75, 40), 'b-.',linewidth = 2)
pylab.plot(spots, -callfunc(spots, 125, 20), 'r--',linewidth = 2
)
pylab.plot(spots, callfunc(spots, 75, 40) -  callfunc(spots, 125
, 20), 'k-',linewidth = 2)
font.set_size(15)
pylab.legend([u'看涨期权多头，执行价格75',u'看涨期权空头，执行价格125',
u'策略组合收益'], prop = font, loc = 'best')
pylab.title(u'看涨期权构造的牛市差价策略', fontproperties = font)
pylab.xlabel(u'标的价格', fontproperties = font)
pylab.ylabel(u'偿付', fontproperties = font)
pylab.grid()
```



如上图所示，牛市价差策略虽然限制了投资者在股票大涨时的收益，但也控制了股价下跌时候的损失幅度。换句话说，投资者持有一个执行价格75的看涨期权，同时卖出一个执行价格较高为125的看涨期权而放弃了股票上涨时候的潜在收益。可以看出，放弃获取这些潜在收益的补偿，就是卖出执行价格125的期权而得到的期权费用。

进一步的，市场上的牛市差价策略可以分成3种类型：

- 两个看涨期权均为虚值期权
- 持有的看涨期权为实值期权，卖出的期权为虚值期权

- 两个看涨期权均为实值期权

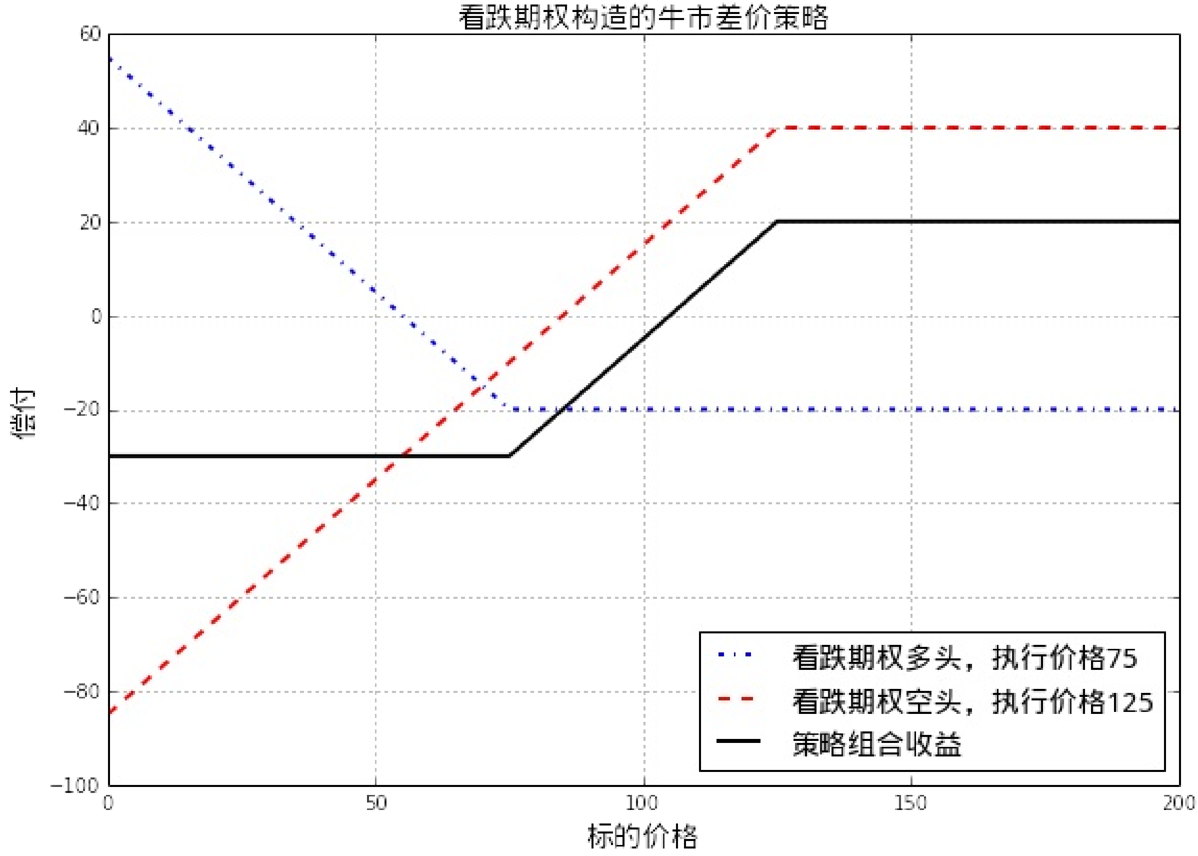
第一种牛市差价策略最为激进，这一策略的成本很低；最后一种牛市差价策略趋于保守。

实际上，利用看跌期权也可以构造牛市差价策略，策略的结构为：

- 买入一份股票标的看跌期权
- 卖出同一股票标的、期权期限相同，但执行价格较高的看跌期权（对于看跌期权来说，执行价格高意味着期权费用较高）

看跌期权构造的牛市差价策略在开始时带给投资者一个正的现金流，策略到期收益和股票价格的关系如下图：

```
spots = np.linspace(0,200,41)

pylab.figure(figsize=(10,7))
pylab.plot(spots, putfunc(spots, 75, 20), 'b-.',linewidth = 2)
pylab.plot(spots, -putfunc(spots, 125, 40), 'r--',linewidth = 2)
pylab.plot(spots, putfunc(spots, 75, 20) - putfunc(spots, 125, 40
), 'k-',linewidth = 2)
font.set_size(15)
pylab.legend([u'看跌期权多头，执行价格75',u'看跌期权空头，执行价格125',
u'策略组合收益'], prop = font, loc = 'best')
pylab.title(u'看跌期权构造的牛市差价策略', fontproperties = font)
pylab.xlabel(u'标的价格', fontproperties = font)
pylab.ylabel(u'偿付', fontproperties = font)
pylab.grid()
```



3.2 熊市差价（bear spread）

投资者构造3.1节中的牛市差价策略时，看好的是股票价格上涨；类似的，此处构造熊市差价策略时，投资者看好股票价格下跌。

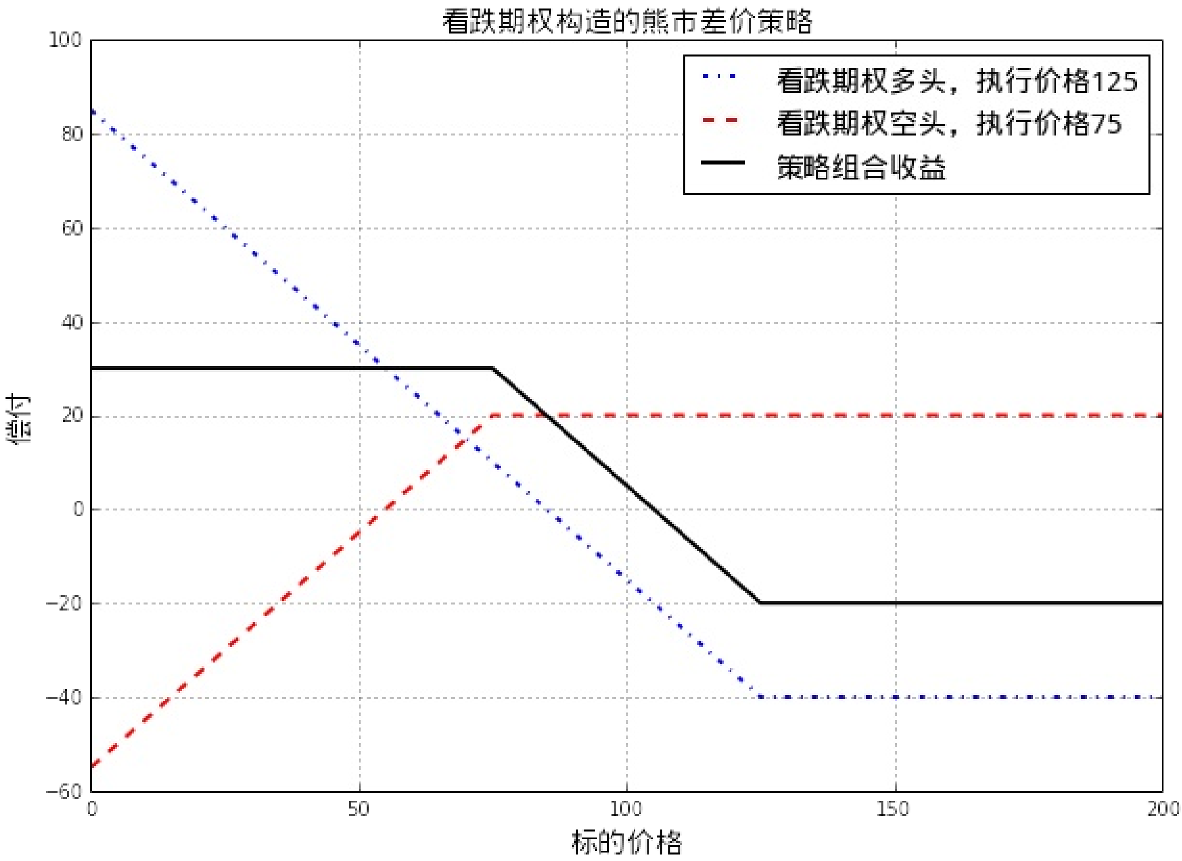
策略构造：

- 买入一份股票标的看跌期权
- 卖出同一股票标的、期权期限相同，但执行价格较低的看跌期权（对于看跌期权来说，执行价格低意味着期权费用较低）

与牛市差价策略中卖出较高执行价格看跌期权相反，熊市差价策略中我们总是卖出执行价格较低的期权。

策略到期收益和股票价格的关系如下图：

```
spots = np.linspace(0,200,41)

pylab.figure(figsize=(10,7))
pylab.plot(spots, putfunc(spots, 125, 40), 'b-.',linewidth = 2)
pylab.plot(spots, -putfunc(spots, 75, 20), 'r--',linewidth = 2)
pylab.plot(spots, putfunc(spots, 125, 40) - putfunc(spots, 75, 20
), 'k-',linewidth = 2)
font.set_size(15)
pylab.legend([u'看跌期权多头，执行价格125',u'看跌期权空头，执行价格75',
u'策略组合收益'], prop = font, loc = 'best')
pylab.title(u'看跌期权构造的熊市差价策略', fontproperties = font)
pylab.xlabel(u'标的价格', fontproperties = font)
pylab.ylabel(u'偿付', fontproperties = font)
pylab.grid()
```



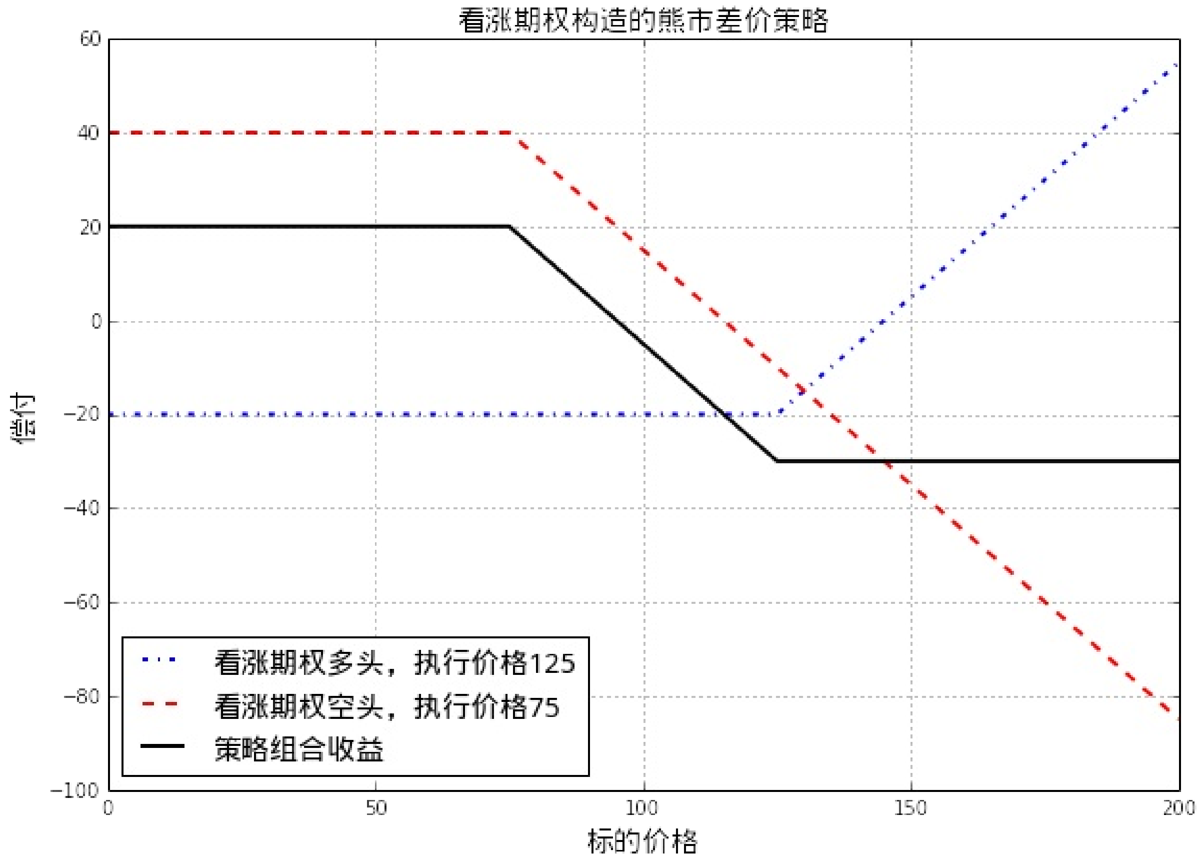
由上图可以看到，熊市差价策略限定了收益上限，也控制了损失幅度。投资者看好股票价格下跌，买入执行价格为125的看跌期权；同时，投资者卖出一份执行价格为75的看跌期权，虽然放弃了股票大幅下跌时候的额外收益，但也收取了卖出期权的期权费用。

和牛市差价策略类似，也可以通过看涨期权构造熊市差价策略：

- 买入一份股票标的看涨期权
- 卖出同一股票标的、期权期限相同，但执行价格较低的看涨期权（对于看涨期权来说，执行价格低意味着期权费用较高）

策略到期收益和股票价格的关系如下图：

```
spots = np.linspace(0,200,41)

pylab.figure(figsize=(10,7))
pylab.plot(spots, callfunc(spots, 125, 20), 'b-.',linewidth = 2)
pylab.plot(spots, -callfunc(spots, 75, 40), 'r--',linewidth = 2)
pylab.plot(spots, callfunc(spots, 125, 20) -  callfunc(spots, 75
, 40), 'k-',linewidth = 2)
font.set_size(15)
pylab.legend([u'看涨期权多头，执行价格125',u'看涨期权空头，执行价格75',
u'策略组合收益'], prop = font, loc = 'best')
pylab.title(u'看涨期权构造的熊市差价策略', fontproperties = font)
pylab.xlabel(u'标的价格', fontproperties = font)
pylab.ylabel(u'偿付', fontproperties = font)
pylab.grid()
```



3.3 盒式差价（box spread）

盒式差价是指由看涨期权构造的一个牛市差价和由看跌期权构造的一个熊市差价的组合。

利用上面的讨论，我们具体地构造这一策略：

- 牛市差价：价格 `c1` 买入执行价格为K1的看涨期权，同时卖出执行价格

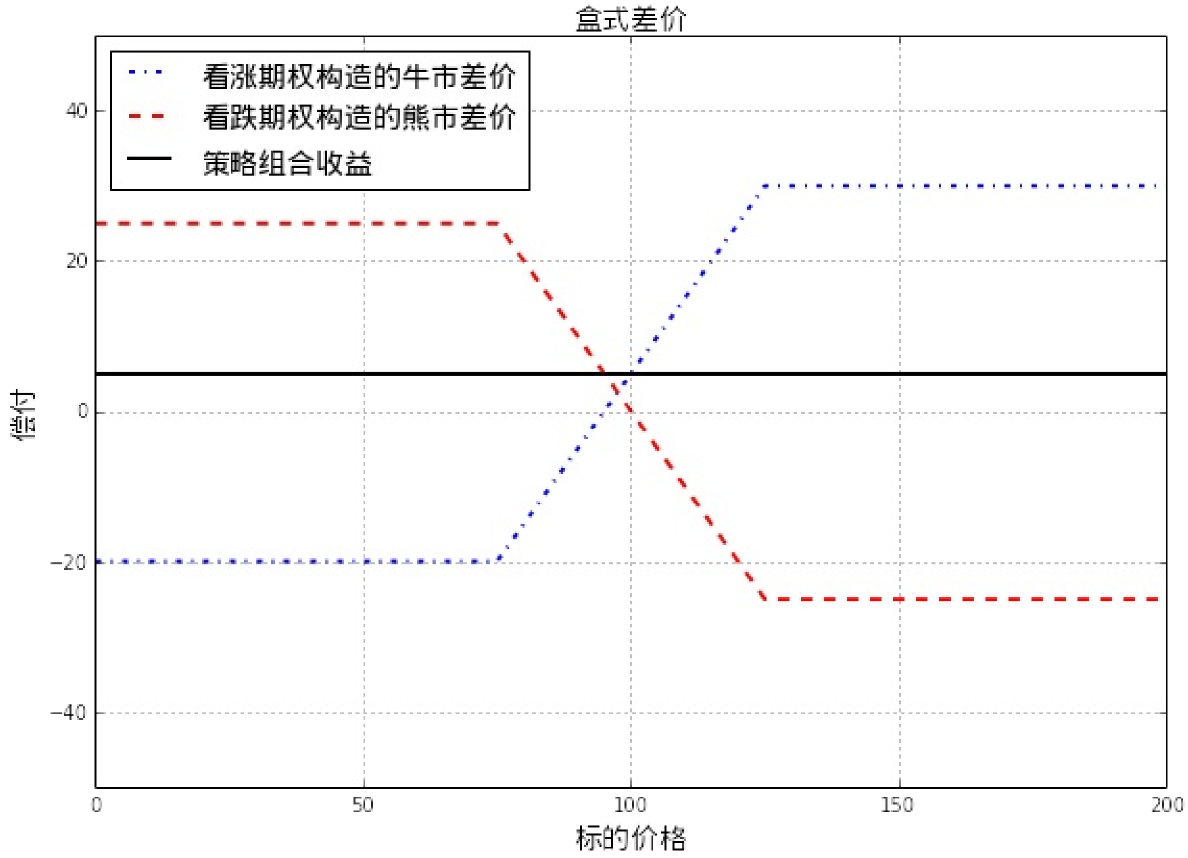
为 `K2` 的看涨期权并获得期权费用c2

- 熊市差价：价格 `p1` 买入执行价格为K1的看跌期权，同时卖出执行价格为 `K2` 的看跌期权并获得期权费用p2

可以看出，构造这一策略我们需要付出初始现金：`(c2-c1)+(p2-p1)` 。

策略到期收益和股票价格的关系示意如下图（相应参数为 `c1=40` , `c2=20` , `p1=45` , `p2=20` , `K1=75` , `K2=125` ）：

```
spots = np.linspace(0,200,41)

pylab.figure(figsize=(10,7))
pylab.plot(spots, callfunc(spots, 75, 40) -  callfunc(spots, 125, 20), 'b-.',linewidth = 2)
pylab.plot(spots, putfunc(spots, 125, 45) - putfunc(spots, 75, 20), 'r--',linewidth = 2)
pylab.plot(spots, callfunc(spots, 75, 40) -  callfunc(spots, 125, 20) + putfunc(spots, 125, 45) - putfunc(spots, 75, 20), 'k-',linewidth = 2)
font.set_size(15)
pylab.legend([u'看涨期权构造的牛市差价',u'看跌期权构造的熊市差价',u'策略组合收益'], prop = font, loc = 'best')
pylab.title(u'盒式差价', fontproperties = font)
pylab.xlabel(u'标的价格', fontproperties = font)
pylab.ylabel(u'偿付', fontproperties = font)
pylab.ylim(-50,50)
pylab.grid()
```



由上图可见，由四个期权构成的盒式差价策略，其期权到期日的收益为一常数；换句话说，盒式差价策略的收益是无风险的！

实际上，盒式差价策略的到期收益贴现值为 `K2-K1` 的贴现值$(K_2-K_1)e^{-rT}$（上面作图时，我们假定了无风险利率 `r=0` ）减去初始构建策略投资现金 `(c2-c1)+(p2-p1)` ：

$$R_{\text{box}}=(K_2-K_1)e^{-rT}-[(c_2-c_1)+(p_2-p_1)]$$

因为这一收益是无风险的，所以如果期权市场价格合理的话，该收益应该为0；也就是，盒式差价策略是一个套利策略，如果我们能够通过购买以上四个期权头寸而构建出 `Rbox>0` 的盒式差价组合的话，说明市场对期权定价不合理，可以通过盒式差价套利。

3.4 蝶式差价（butterfly spread）

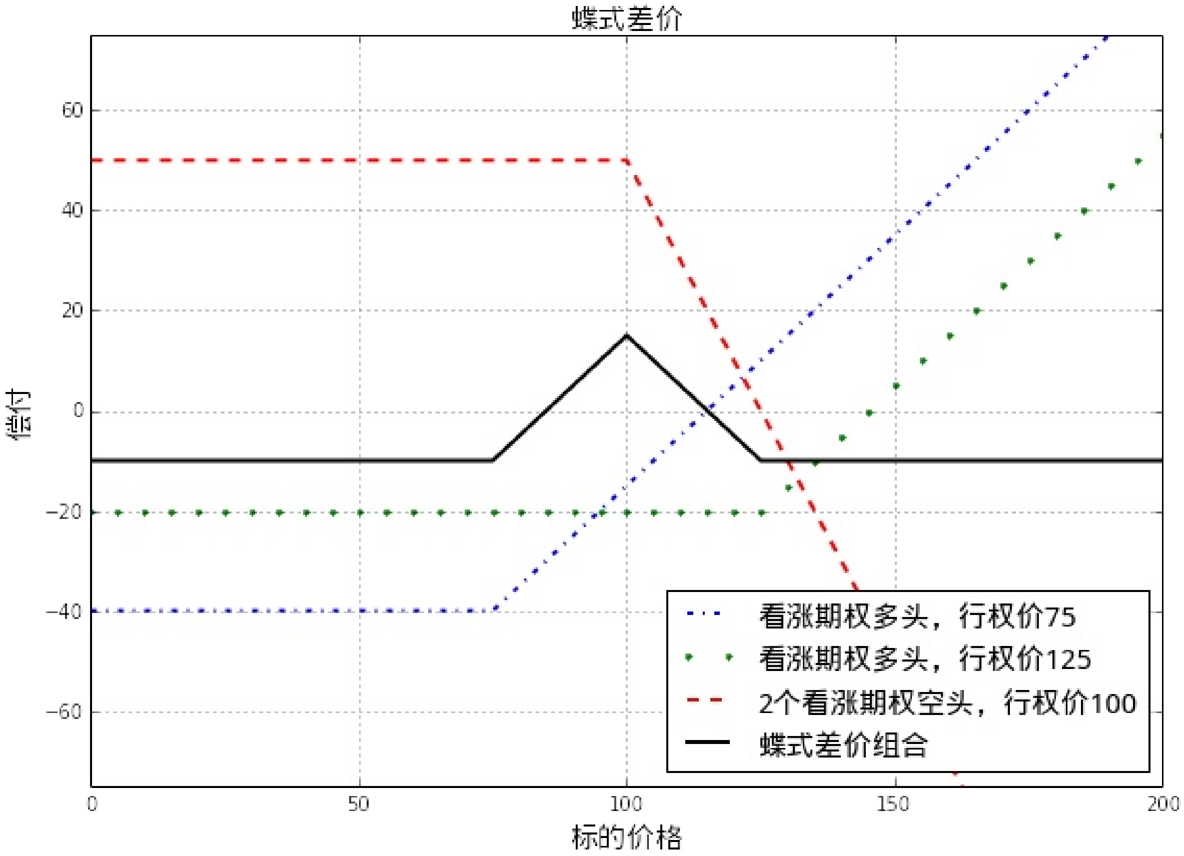
蝶式差价由三个相同标的，且到期日相同而行权价不同的看涨期权构成。

策略构造：

- 买入一个具有较低执行价格 `K1` 的欧式看涨期权
- 买入一个具有较高执行价格 `K3` 的欧式看涨期权
- 卖出两个具有中间执行价格K2的欧式看涨期权，其中 `K1<K2<K3` ，一般来讲，K2接近于当前标的股票价格

策略到期收益和股票价格的关系示意如下图：

```
spots = np.linspace(0,200,41)

pylab.figure(figsize=(10,7))
pylab.plot(spots, callfunc(spots, 75, 40), 'b-.',linewidth = 2)
pylab.plot(spots, callfunc(spots, 125, 20), 'g.',linewidth = 2)
pylab.plot(spots, -2*callfunc(spots, 100, 25), 'r--',linewidth =
2)
pylab.plot(spots, callfunc(spots, 75, 40) + callfunc(spots, 125,
20) - 2*callfunc(spots, 100, 25), 'k-',linewidth = 2)
font.set_size(15)
pylab.legend([u'看涨期权多头，行权价75',u'看涨期权多头，行权价125',u'2
个看涨期权空头，行权价100', u'蝶式差价组合'], prop = font, loc = 'best'
)
pylab.title(u'蝶式差价', fontproperties = font)
pylab.xlabel(u'标的价格', fontproperties = font)
pylab.ylabel(u'偿付', fontproperties = font)
pylab.ylim(-75,75)
pylab.grid()
```



如上图所示，构造蝶式差价时，投资者对于标的股票持中立态度，认为期权行权日前股价将会稳定在K2附近(也就是在今日股价附近)。

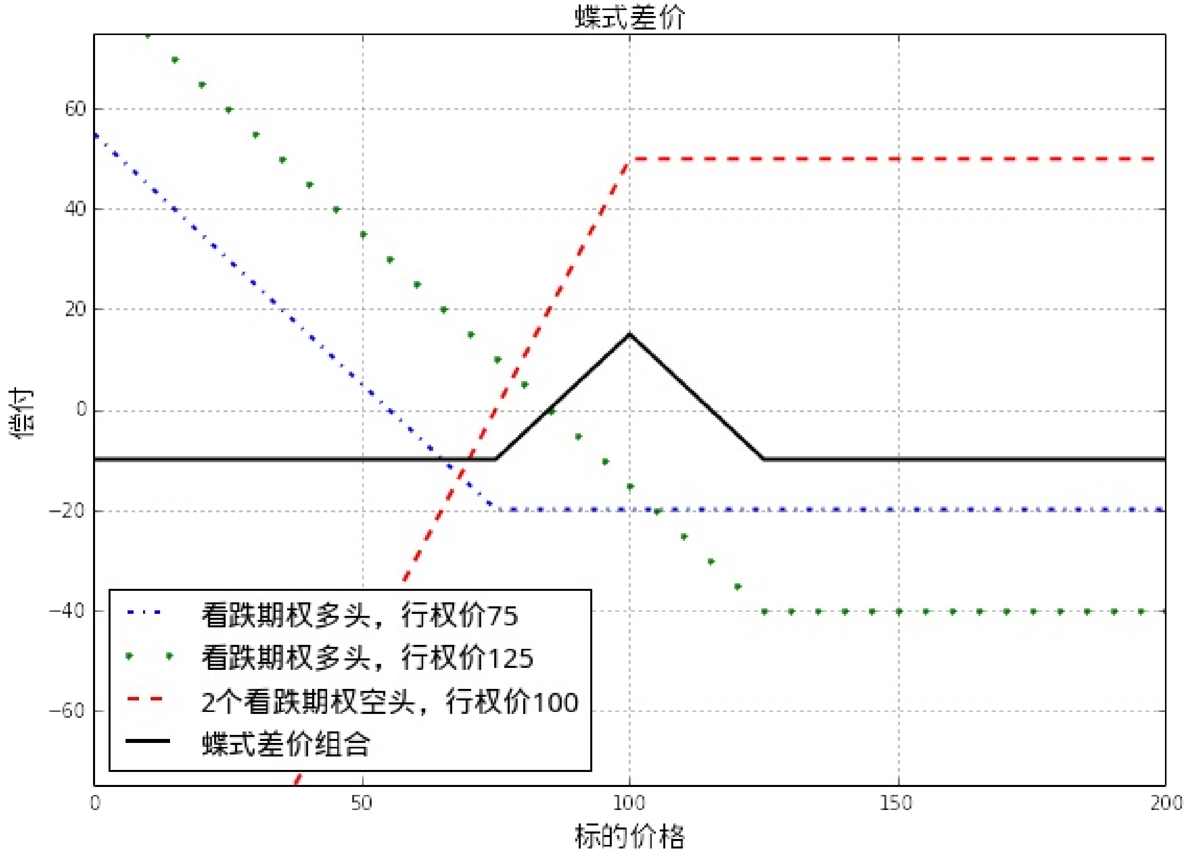
如果到期时候股价保持在 K2 附近，蝶式差价将会产生收益；如果到期时候股价远远偏离K2，投资者将损失少量的初始投资。

类似于看涨期权构造的蝶式差价，看跌期权也可以用来构造蝶式差价组合：

- 买入一个具有较低执行价格 `K1` 的欧式看跌期权
- 买入一个具有较高执行价格 `K3` 的欧式看跌期权
- 卖出两个具有中间执行价格 `K2` 的欧式看跌期权，其中 `K1<K2<K3` ，一般来讲， `K2` 接近于当前标的股票价格

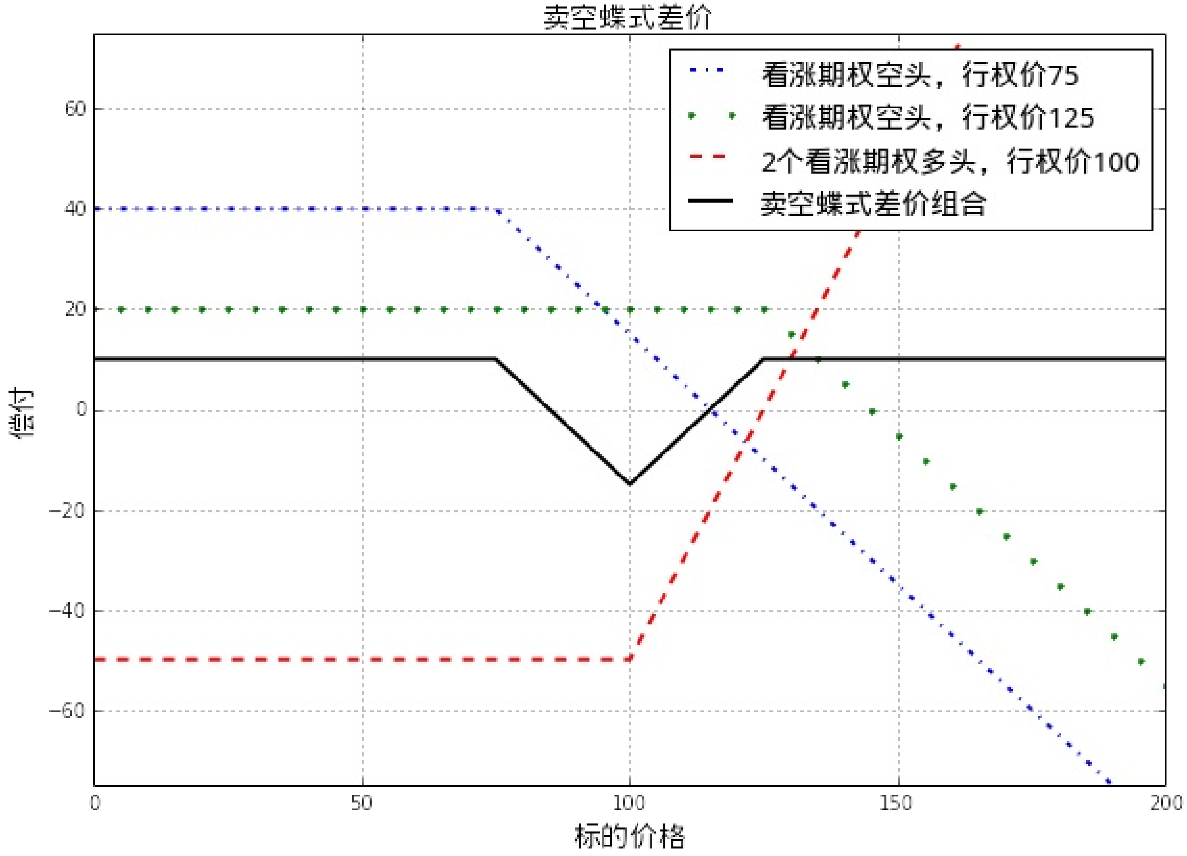
策略到期收益和股票价格的关系示意如下图：

```
spots = np.linspace(0,200,41)

pylab.figure(figsize=(10,7))
pylab.plot(spots, putfunc(spots, 75, 20), 'b-.',linewidth = 2)
pylab.plot(spots, putfunc(spots, 125, 40), 'g.',linewidth = 2)
pylab.plot(spots, -2*putfunc(spots, 100, 25), 'r--',linewidth = 2)
pylab.plot(spots, putfunc(spots, 75, 20) + putfunc(spots, 125, 40) - 2*putfunc(spots, 100, 25), 'k-',linewidth = 2)
font.set_size(15)
pylab.legend([u'看跌期权多头，行权价75',u'看跌期权多头，行权价125',u'2个看跌期权空头，行权价100', u'蝶式差价组合'], prop = font, loc = 'best')
pylab.title(u'蝶式差价', fontproperties = font)
pylab.xlabel(u'标的价格', fontproperties = font)
pylab.ylabel(u'偿付', fontproperties = font)
pylab.ylim(-75,75)
pylab.grid()
```



和蝶式差价组合相反，投资者如果看好股价在未来将有巨大的波动，但却不确定波动方向时，可以卖空蝶式差价，从而在股价大幅波动时赚取收益。见下图（卖空一个由看涨期权构成的蝶式差价组合）：

```
spots = np.linspace(0,200,41)

pylab.figure(figsize=(10,7))
pylab.plot(spots, -callfunc(spots, 75, 40), 'b-.',linewidth = 2)
pylab.plot(spots, -callfunc(spots, 125, 20), 'g.',linewidth = 2)
pylab.plot(spots, 2*callfunc(spots, 100, 25), 'r--',linewidth = 2
)
pylab.plot(spots, - callfunc(spots, 75, 40) - callfunc(spots, 125
, 20) + 2*callfunc(spots, 100, 25), 'k-',linewidth = 2)
font.set_size(15)
pylab.legend([u'看涨期权空头，行权价75',u'看涨期权空头，行权价125',u'2
个看涨期权多头，行权价100', u'卖空蝶式差价组合'], prop = font, loc = '
best')
pylab.title(u'卖空蝶式差价', fontproperties = font)
pylab.xlabel(u'标的价格', fontproperties = font)
pylab.ylabel(u'偿付', fontproperties = font)
pylab.ylim(-75,75)
pylab.grid()
```



3.5 日历差价（calendar spread）

目前为止，我们讨论的差价组合策略中，总是假定所有期权具有相同的到期日。现在，我们看一下构成组合的期权具有相同的行权价但期权到期日不同的情况，即是所谓的日历差价。

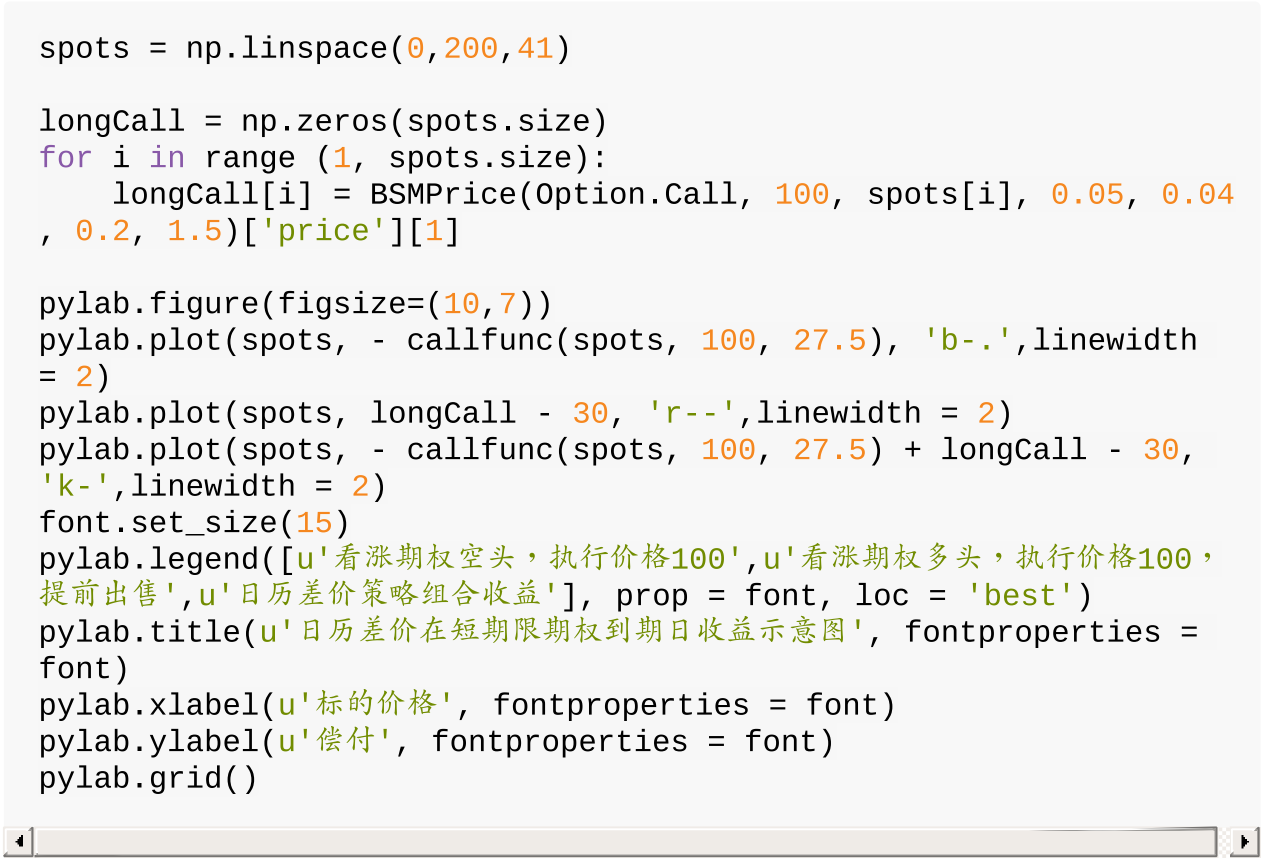
策略构造：

- 卖出一个具有较短期限的欧式看涨期权
- 买入一个具有较长期限，且行权价和上一个期权相同的看涨期权

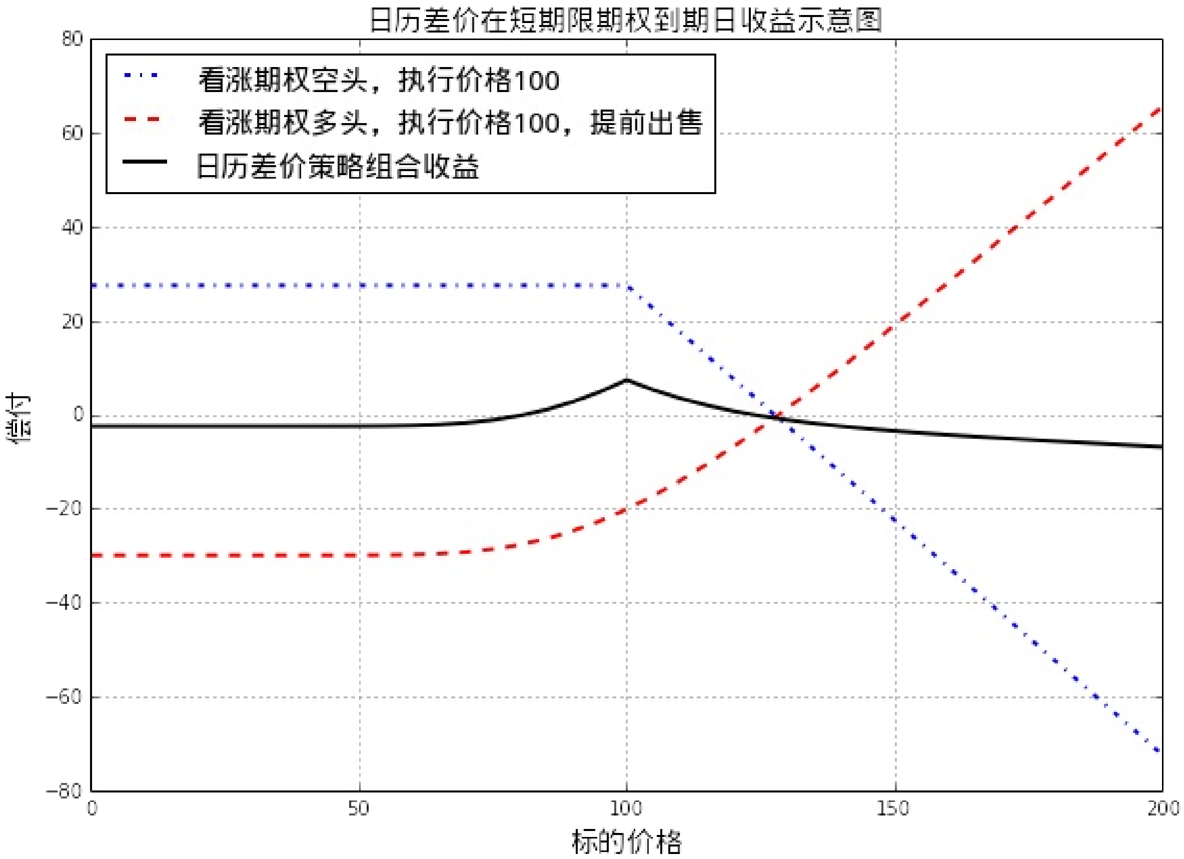
对于看涨期权来说，期限越长，期权价格越贵，所以构造日历差价需要一定的初始投资。

讨论日历价差的盈利曲线时候，我们假定盈利实现在短期限期权的到期日，届时长期限期权被出售。

较短期限期权到期日的策略收益和股票价格的关系示意如下图：



```
spots = np.linspace(0,200,41)

longCall = np.zeros(spots.size)
for i in range (1, spots.size):
    longCall[i] = BSMPrice(Option.Call, 100, spots[i], 0.05, 0.04
, 0.2, 1.5)['price'][1]

pylab.figure(figsize=(10,7))
pylab.plot(spots, - callfunc(spots, 100, 27.5), 'b-.',linewidth
= 2)
pylab.plot(spots, longCall - 30, 'r--',linewidth = 2)
pylab.plot(spots, - callfunc(spots, 100, 27.5) + longCall - 30,
'k-',linewidth = 2)
font.set_size(15)
pylab.legend([u'看涨期权空头，执行价格100',u'看涨期权多头，执行价格100，
提前出售',u'日历差价策略组合收益'], prop = font, loc = 'best')
pylab.title(u'日历差价在短期限期权到期日收益示意图', fontproperties =
font)
pylab.xlabel(u'标的价格', fontproperties = font)
pylab.ylabel(u'偿付', fontproperties = font)
pylab.grid()
```



如上图所示，与蝶式差价组合类似，在短期限期权到期日，如果股票价格接近于期权执行价格，则投资者可获利；相反，如果股票价格远离期权执行价格，则投资者要付出一定损失。

进一步讨论日历差价的收益，在短期限期权到期日：

- 股票价格较低时，两个期权都是看涨期权，故其价值均接近于0；投资者损失了构造组合的初始投资费用
- 股票价格很高时，短期限期权空头带来损失 `S-K` （其中 `S` 为股票价格， `K` 为行权价）；长期限期权价格接近于 `S-K` ；总体上，投资者也会损失构造组合的初始投资
- 股票价格接近行权价时，短期限期权带来损益接近于0；长期限期权因为未到期，仍具有潜在价值；投资者会获得一定收入

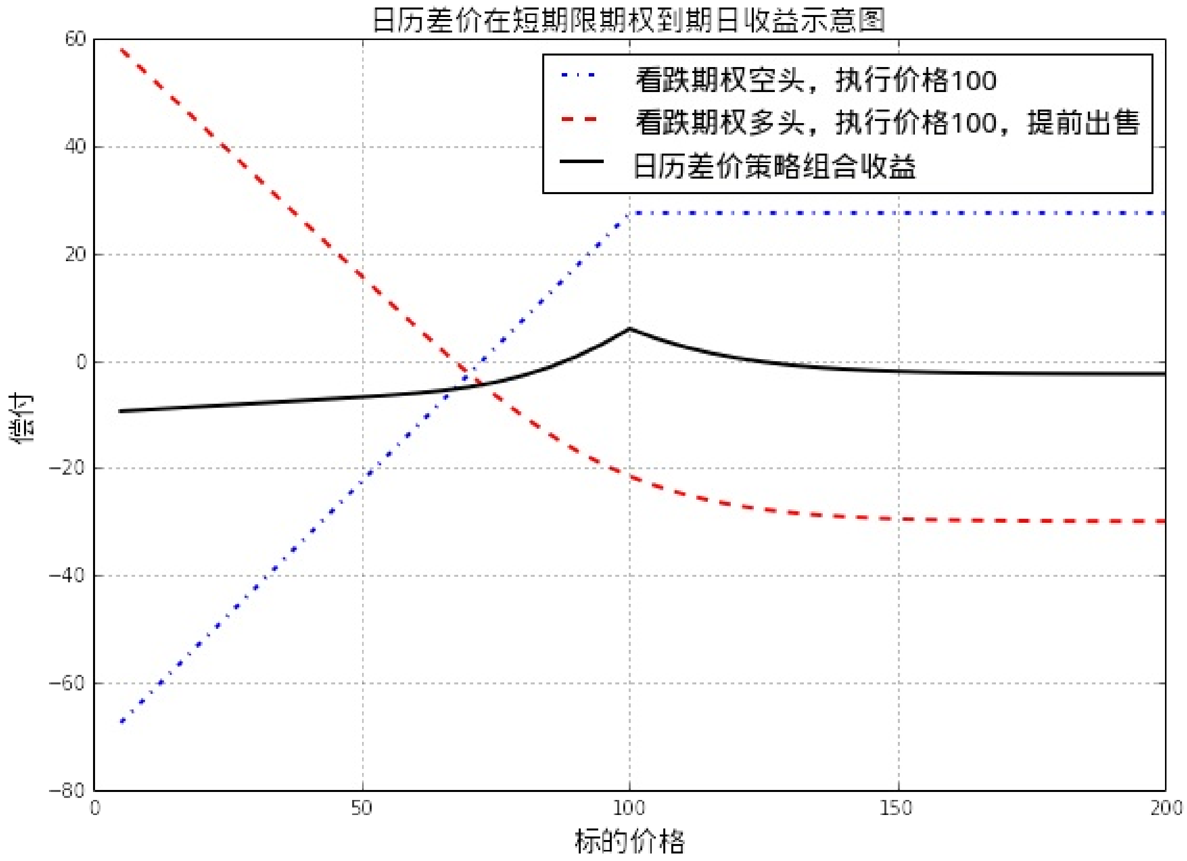
类似地，可以利用看跌期权构造日历差价组合：

- 卖出一个具有较短期限的欧式看跌期权
- 买入一个具有较长期限，且行权价和上一个期权相同的看跌期权

对于看跌期权来说，期限越长，期权价格越贵，所以利用看跌期权构造日历差价需要一定的初始投资。

较短期限期权到期日的策略收益和股票价格的关系示意如下图：

```
spots = np.linspace(5,200,40)

longPut = np.zeros(spots.size)
for i in range (0, spots.size):
    longPut[i] = BSMPrice(Option.Put, 100, spots[i], 0.05, 0.04, 0.2, 1.5)['price'][1]

pylab.figure(figsize=(10,7))
pylab.plot(spots, - putfunc(spots, 100, 27.5), 'b-.',linewidth = 2)
pylab.plot(spots, longPut - 30, 'r--',linewidth = 2)
pylab.plot(spots, - putfunc(spots, 100, 27.5) + longPut - 30, 'k-',linewidth = 2)
font.set_size(15)
pylab.legend([u'看跌期权空头，执行价格100',u'看跌期权多头，执行价格100，提前出售',u'日历差价策略组合收益'], prop = font, loc = 'best')
pylab.title(u'日历差价在短期限期权到期日收益示意图', fontproperties = font)
pylab.xlabel(u'标的价格', fontproperties = font)
pylab.ylabel(u'偿付', fontproperties = font)
pylab.grid()
```



根据构造日历差价时候选择的期权执行价不同，日历差价可以分成三种类型：

- 中性日历差价：执行价接近于股票当前价
- 牛市日历差价：执行价高于股票当前价
- 熊市日历差价：执行价低于股票当前价

和蝶式差价类似，如果投资者看好股价在未来会有很大的波动，可以选择卖空日历差价组合，达到股价大幅波动时获益的效果。

3.6 对角差价（diagonal spread）

牛市差价、熊市差价里面的期权长短头寸，都具有相同的到期日和不同的行权价；日历差价中的期权长短头寸，则具有相同的行权价和不同的到期日；进一步推广，可以构造对角差价组合：

对角差价中的期权长短头寸，到期日和行权价都不相同，会产生更加多样化的盈利曲线。 对角差价比较复杂，此处暂时略过。

4. 组合策略（combination）

相对于差价策略包括同一股票标的行权价不同的同类（看涨或看跌）期权，组合策略中包括同一种股票的看涨和看跌期权。

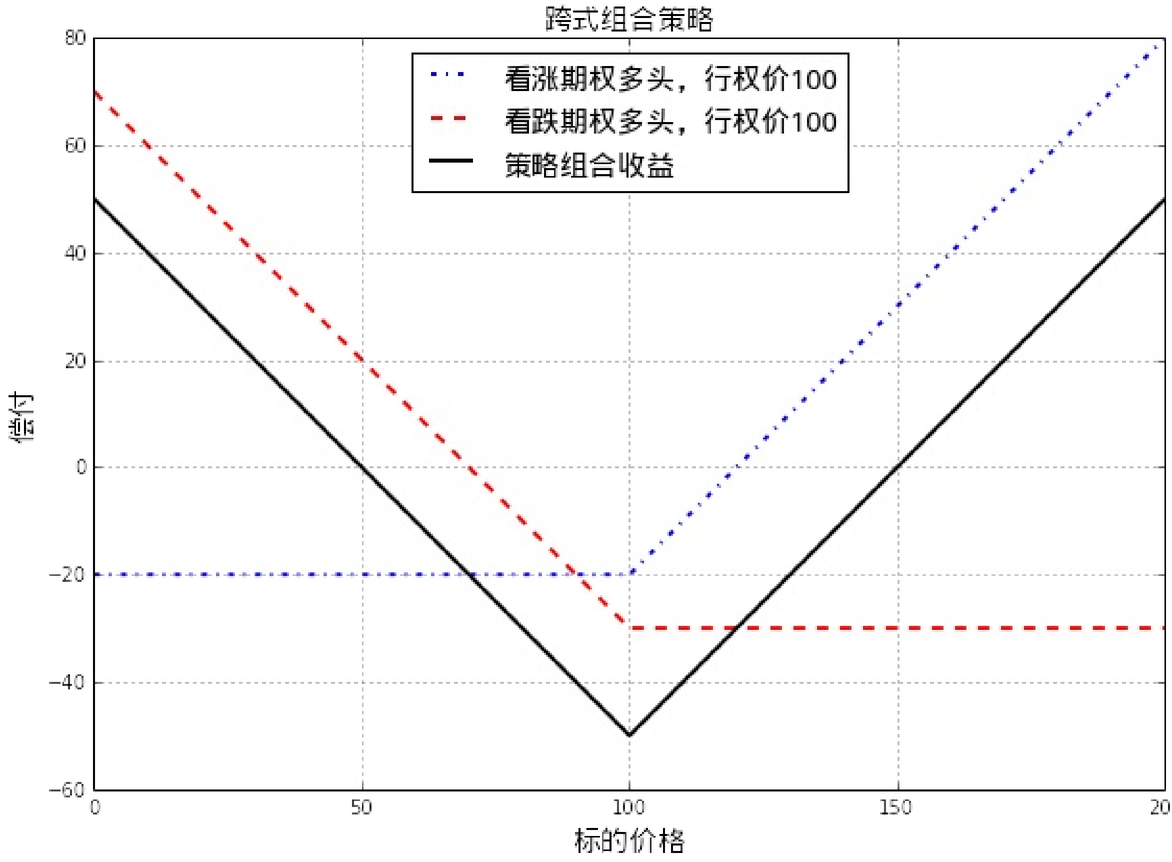
4.1 跨式组合（straddle combination）

策略构造：

- 买入一个看涨期权
- 买入一个标的相同、期限相同、行权价相同的看跌期权

跨式组合策略收益和股票价格的关系示意如下图：

```
spots = np.linspace(0,200,41)

pylab.figure(figsize=(10,7))
pylab.plot(spots, callfunc(spots, 100, 20), 'b-.',linewidth = 2)
pylab.plot(spots, putfunc(spots, 100, 30), 'r--',linewidth = 2)
pylab.plot(spots, callfunc(spots, 100, 20) + putfunc(spots, 100,
30), 'k-',linewidth = 2)
font.set_size(15)
pylab.legend([u'看涨期权多头，行权价100',u'看跌期权多头，行权价100',u'
策略组合收益'], prop = font, loc = 'best')
pylab.title(u'跨式组合策略', fontproperties = font)
pylab.xlabel(u'标的价格', fontproperties = font)
pylab.ylabel(u'偿付', fontproperties = font)
pylab.grid()
```



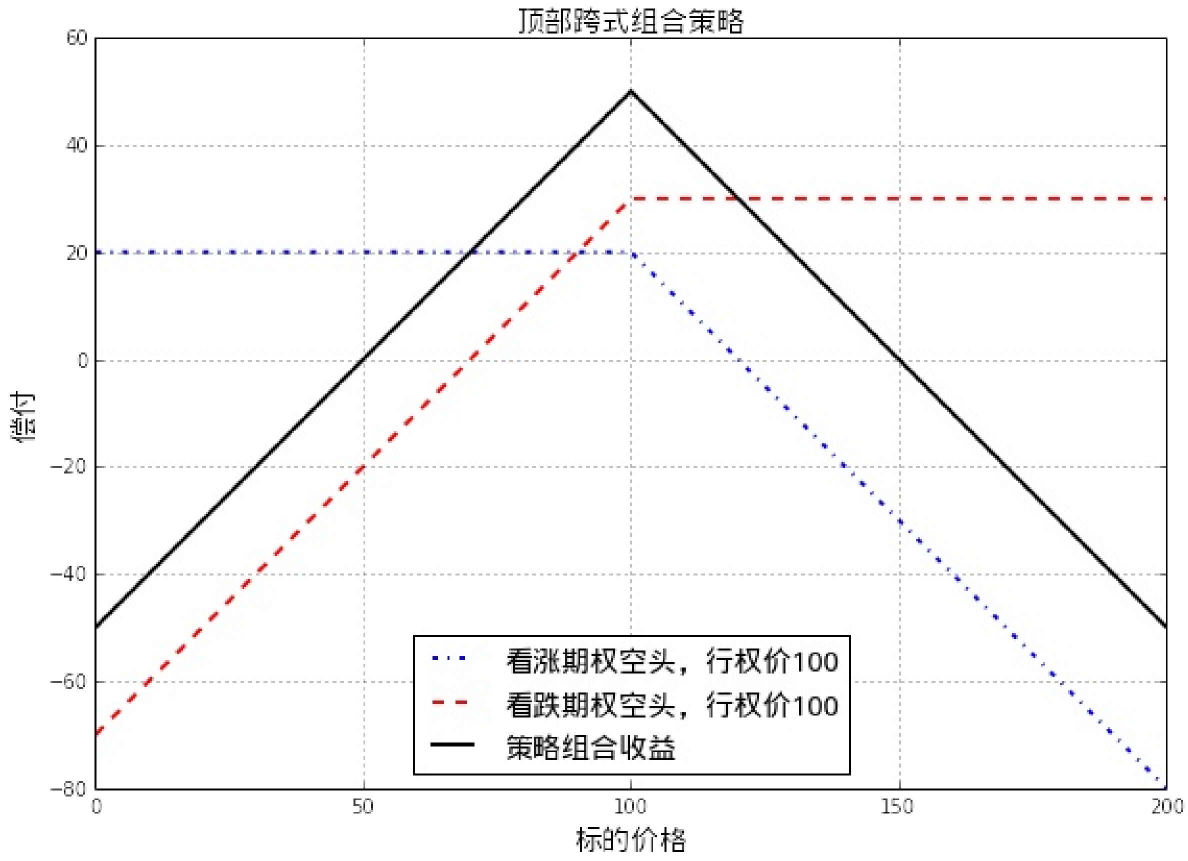
如上图所示，如果投资者认为股票价格会有很大的波动，但却不确定波动方向时候，可以选择跨式组合。

上图中的跨式组合有时被称为底部跨式组合（bottom straddle）或者买入跨式组合（straddle purchase）；与此相反，投资者可以构造顶部跨式组合（top straddle）或者卖出跨式组合（straddle write）：

- 卖出一个看涨期权
- 卖出一个标的相同、期限相同、行权价相同的看跌期权

顶部跨式组合策略收益和股票价格的关系示意如下图：

```
spots = np.linspace(0,200,41)

pylab.figure(figsize=(10,7))
pylab.plot(spots, -callfunc(spots, 100, 20), 'b-.',linewidth = 2
)
pylab.plot(spots, -putfunc(spots, 100, 30), 'r--',linewidth = 2)
pylab.plot(spots, -callfunc(spots, 100, 20) - putfunc(spots, 100
, 30), 'k-',linewidth = 2)
font.set_size(15)
pylab.legend([u'看涨期权空头，行权价100',u'看跌期权空头，行权价100',u'
策略组合收益'], prop = font, loc = 'best')
pylab.title(u'顶部跨式组合策略', fontproperties = font)
pylab.xlabel(u'标的价格', fontproperties = font)
pylab.ylabel(u'偿付', fontproperties = font)
pylab.grid()
```



如上图，当到期日股价接近于行权价时候，顶部跨式组合和蝶式差价类似，都是获得一定收益；但是，当股票价格在到期日变化剧烈时：

蝶式差价只损失一小部分构造费用，但是顶部跨式组合却可能让投资者血本无归，因为顶部跨式组合此时的损失是无限的！

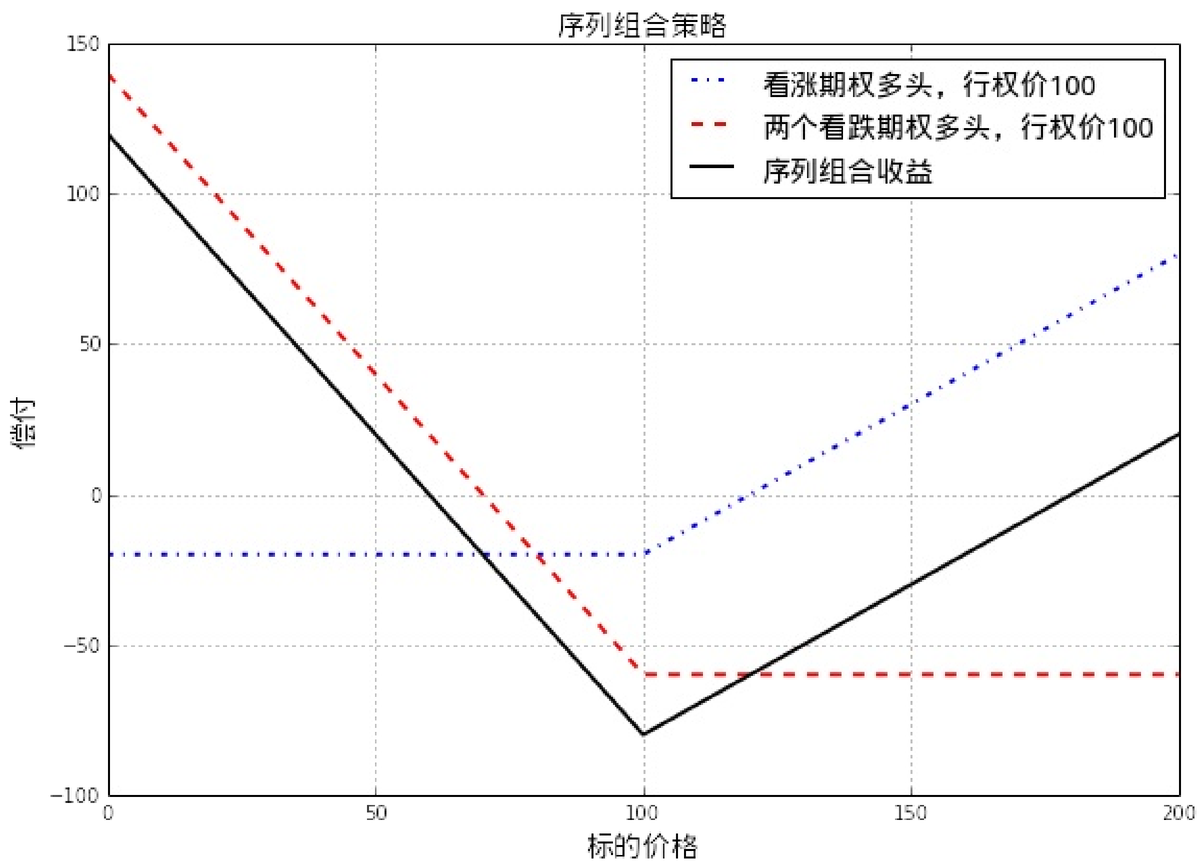
4.2 序列组合（strip）和带式组合（strap）

序列组合策略构造：

- 买入一个欧式看涨期权
- 买入两个标的相同、期限相同、行权价相同的看跌期权

跨式组合策略收益和股票价格的关系示意如下图：

```
spots = np.linspace(0,200,41)

pylab.figure(figsize=(10,7))
pylab.plot(spots, callfunc(spots, 100, 20), 'b-.',linewidth = 2)
pylab.plot(spots, 2*putfunc(spots, 100, 30), 'r--',linewidth = 2)
pylab.plot(spots, callfunc(spots, 100, 20) + 2*putfunc(spots, 100, 30), 'k-',linewidth = 2)
font.set_size(15)
pylab.legend([u'看涨期权多头，行权价100',u'两个看跌期权多头，行权价100',u'序列组合收益'], prop = font, loc = 'best')
pylab.title(u'序列组合策略', fontproperties = font)
pylab.xlabel(u'标的价格', fontproperties = font)
pylab.ylabel(u'偿付', fontproperties = font)
pylab.grid()
```

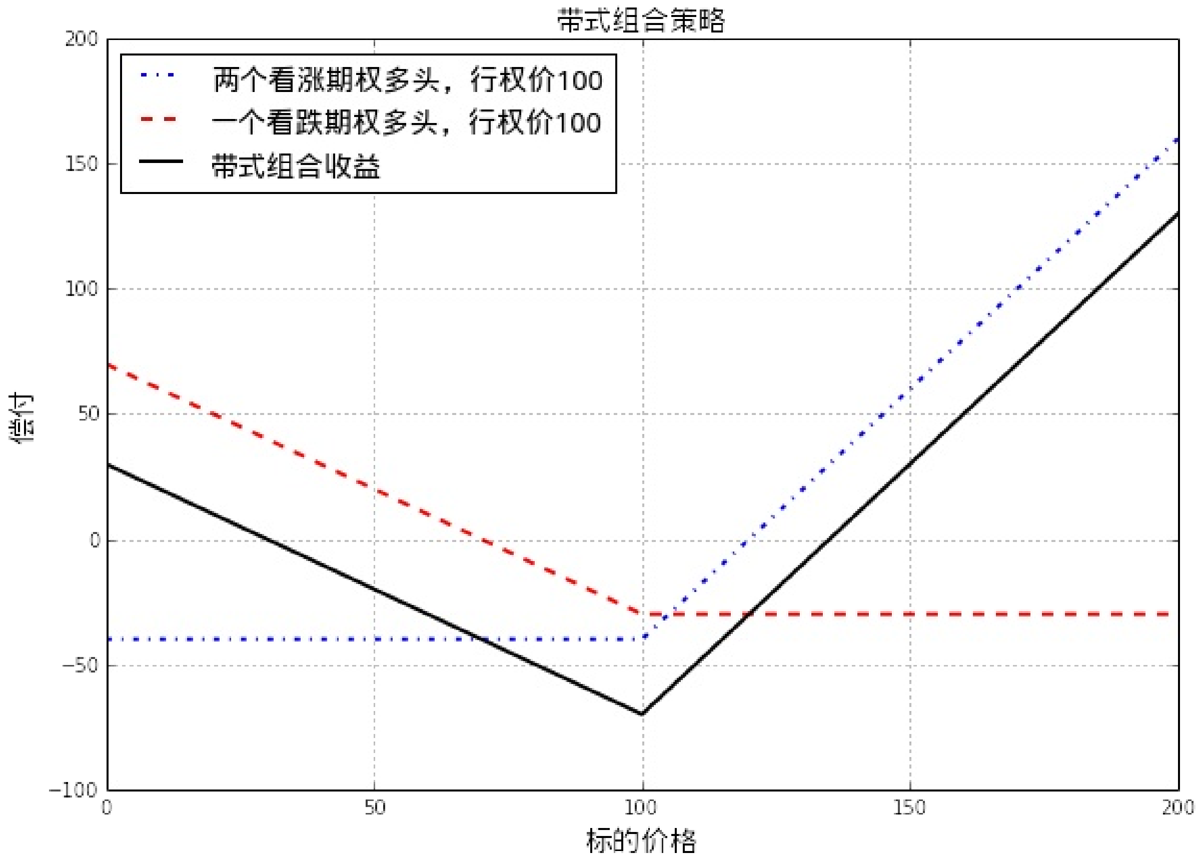


带式组合策略构造：

- 买入两个个欧式看涨期权
- 买入一个标的相同、期限相同、行权价相同的看跌期权

带式组合策略收益和股票价格的关系示意如下图：

```
spots = np.linspace(0,200,41)

pylab.figure(figsize=(10,7))
pylab.plot(spots, 2*callfunc(spots, 100, 20), 'b-.',linewidth = 2
)
pylab.plot(spots, putfunc(spots, 100, 30), 'r--',linewidth = 2)
pylab.plot(spots, 2*callfunc(spots, 100, 20) + putfunc(spots, 100
, 30), 'k-',linewidth = 2)
font.set_size(15)
pylab.legend([u'两个看涨期权多头，行权价100',u'一个看跌期权多头，行权价1
00',u'带式组合收益'], prop = font, loc = 'best')
pylab.title(u'带式组合策略', fontproperties = font)
pylab.xlabel(u'标的价格', fontproperties = font)
pylab.ylabel(u'偿付', fontproperties = font)
pylab.grid()
```



结合上述两张图可知，无论是序列组合还是带式组合，投资者都认为股票价格会有较大幅度的变化，区别在于：

- 序列组合中，投资者认为股价下降的可能性较大
- 带式组合中，投资者认为股价上涨的可能性较大

4.3 异价跨式组合（strangle combination）

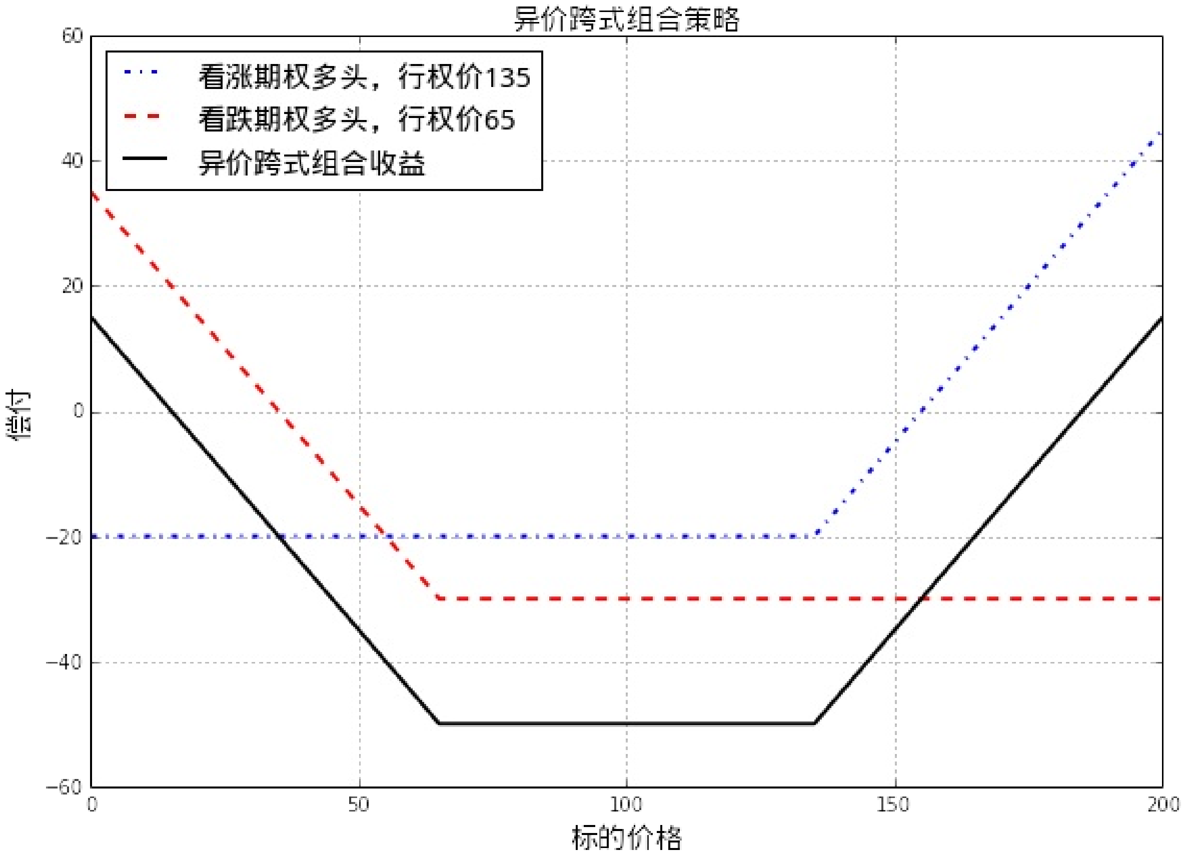
异价跨式组合有时被称为底部垂直组合（bottom vertical combination）。

策略构造：

- 买入一个看涨期权，行权价较高
- 买入一个标的相同、期限相同的看跌期权，行权价较低

异价跨式组合策略收益和股票价格的关系示意如下图：

```
spots = np.linspace(0,200,41)

pylab.figure(figsize=(10,7))
pylab.plot(spots, callfunc(spots, 135, 20), 'b-.',linewidth = 2)
pylab.plot(spots, putfunc(spots, 65, 30), 'r--',linewidth = 2)
pylab.plot(spots, callfunc(spots, 135, 20) + putfunc(spots, 65,
30), 'k-',linewidth = 2)
font.set_size(15)
pylab.legend([u'看涨期权多头，行权价135',u'看跌期权多头，行权价65',u'异
价跨式组合收益'], prop = font, loc = 'best')
pylab.title(u'异价跨式组合策略', fontproperties = font)
pylab.xlabel(u'标的价格', fontproperties = font)
pylab.ylabel(u'偿付', fontproperties = font)
pylab.grid()
```



如上图所示，与跨式组合类似，如果投资者认为股票价格会有很大的波动，但却不确定波动方向时候，可以选择异价跨式组合（底部垂直组合）。与跨式组合不同的是：

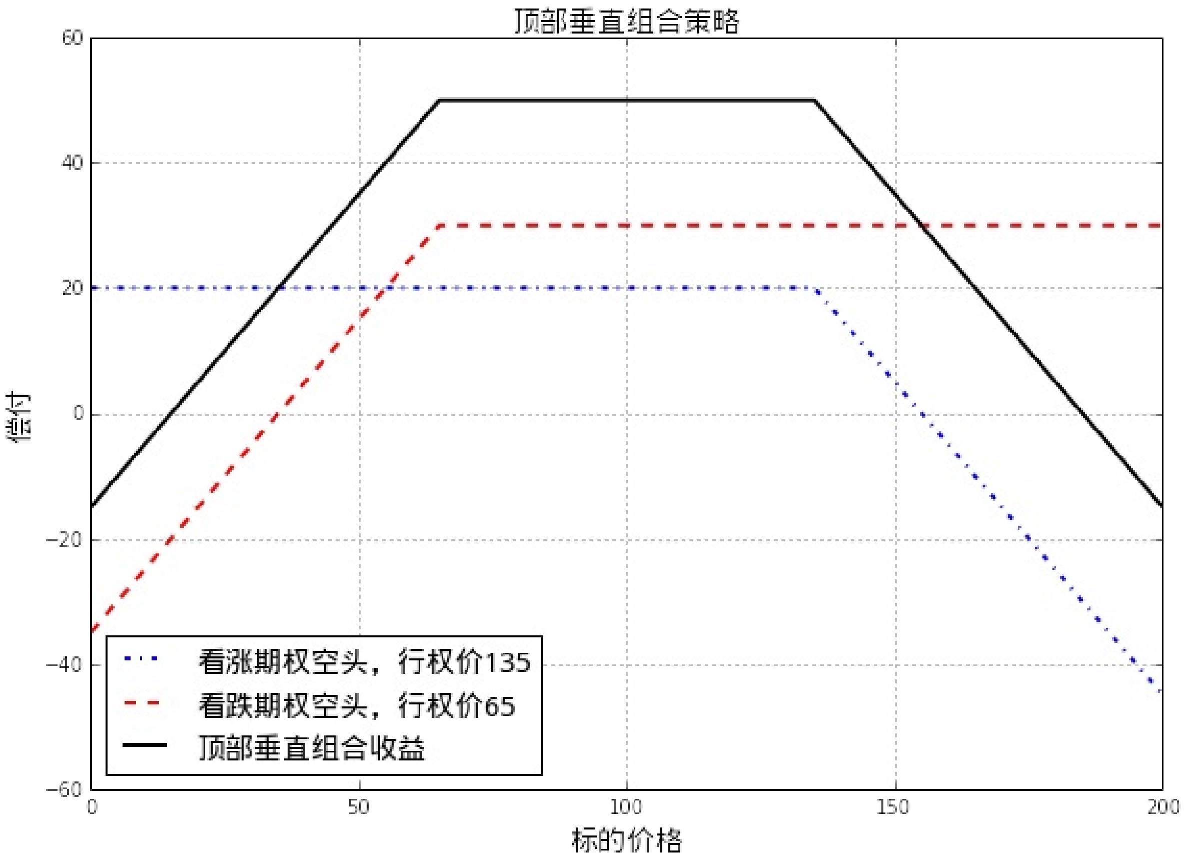
- 底部垂直组合需要股价变动更大才能盈利
- 同时，底部垂直组合当股价比较稳定时的损失却也比较小

相反地，可以构造顶部垂直组合（top vertical combination）：

- 卖出一个看涨期权，行权价较高
- 卖出一个标的相同、期限相同的看跌期权，行权价较低

顶部垂直组合策略收益和股票价格的关系示意如下图：

```
spots = np.linspace(0,200,41)

pylab.figure(figsize=(10,7))
pylab.plot(spots, -callfunc(spots, 135, 20), 'b-.',linewidth = 2)
pylab.plot(spots, -putfunc(spots, 65, 30), 'r--',linewidth = 2)
pylab.plot(spots, -callfunc(spots, 135, 20) - putfunc(spots, 65, 30), 'k-',linewidth = 2)
font.set_size(15)
pylab.legend([u'看涨期权空头，行权价135',u'看跌期权空头，行权价65',u'顶部垂直组合收益'], prop = font, loc = 'best')
pylab.title(u'顶部垂直组合策略', fontproperties = font)
pylab.xlabel(u'标的价格', fontproperties = font)
pylab.ylabel(u'偿付', fontproperties = font)
pylab.grid()
```



如上图可见，构造顶部垂直组合后，如果股价变动非常大，投资者的损失是无限的！

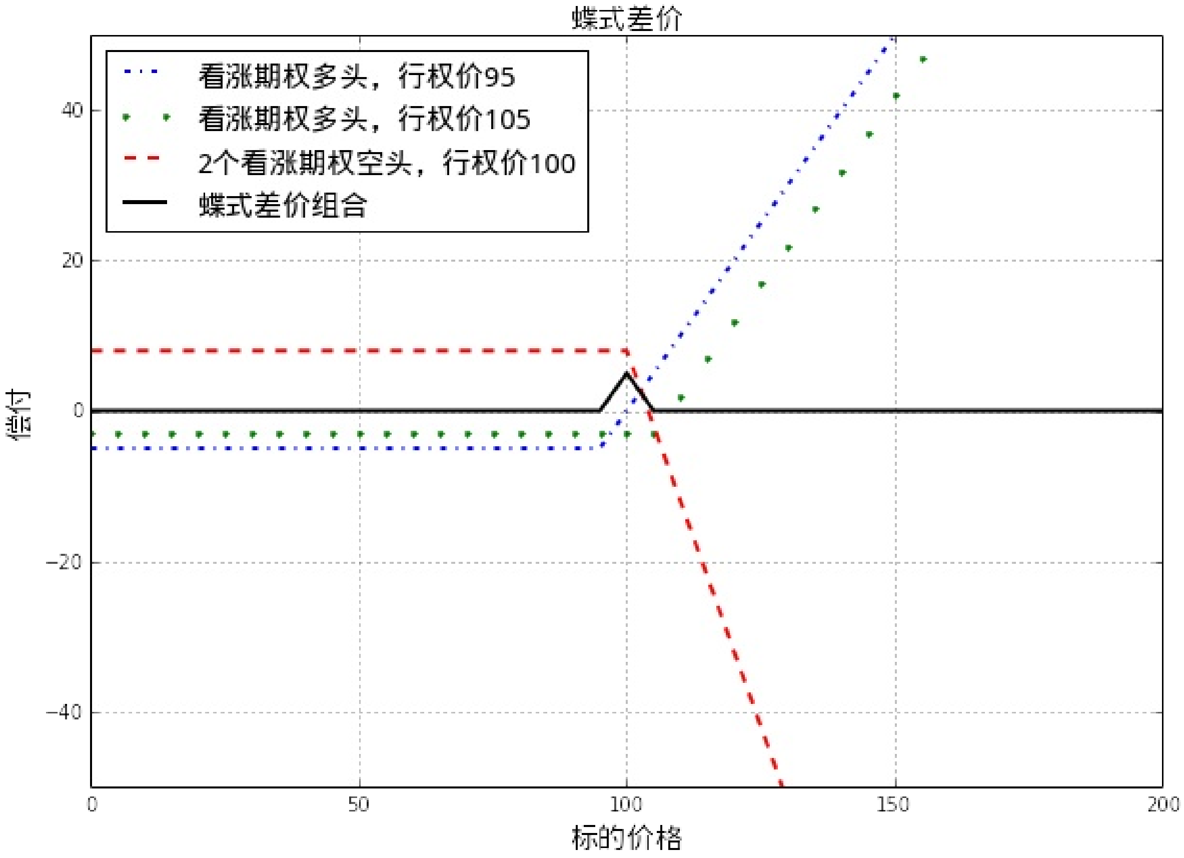
5. 具有任意收益形式的组合

上面的一系列策略组合中，我们看到利用期权可以产生很多有趣的盈利形式。实际上，如果：

- 对于到期日 `T` ，任意行权价的欧式期权均可以交易

那么，理论上讲，可以构造任何形式的到期日收益。这可以利用下图来说明：

```
spots = np.linspace(0,200,41)

pylab.figure(figsize=(10,7))
pylab.plot(spots, callfunc(spots, 95, 5), 'b-.',linewidth = 2)
pylab.plot(spots, callfunc(spots, 105, 3), 'g.',linewidth = 2)
pylab.plot(spots, -2*callfunc(spots, 100, 4), 'r--',linewidth = 2
)
pylab.plot(spots, callfunc(spots, 95, 5) + callfunc(spots, 105, 3
) - 2*callfunc(spots, 100, 4), 'k-',linewidth = 2)
font.set_size(15)
pylab.legend([u'看涨期权多头，行权价95',u'看涨期权多头，行权价105',u'2
个看涨期权空头，行权价100', u'蝶式差价组合'], prop = font, loc = 'best'
)
pylab.title(u'蝶式差价', fontproperties = font)
pylab.xlabel(u'标的价格', fontproperties = font)
pylab.ylabel(u'偿付', fontproperties = font)
pylab.ylim(-50,50)
pylab.grid()
```



上图是一个蝶式差价，其中三个行权价`K1`、`K2`、`K3`非常接近，所以我们构造了一个类似于小尖刺的收益形式。将很多的这样的尖刺组合起来，就可以构造成功任意形式的收益。

期权市场一周纵览

> 来源：https://uqer.io/community/share/55027679f9f06c7a9ae9a53a

本文档依赖的数据 `option_data.csv` 可以通过运行 期权高频数据准备 notebook 而获取。

```
from matplotlib import pylab
import pandas as pd
import seaborn as sns
sns.set(style="white", context="talk")

import pandas as pd
pd.options.display.float_format = '{:,>.4f}'.format
```

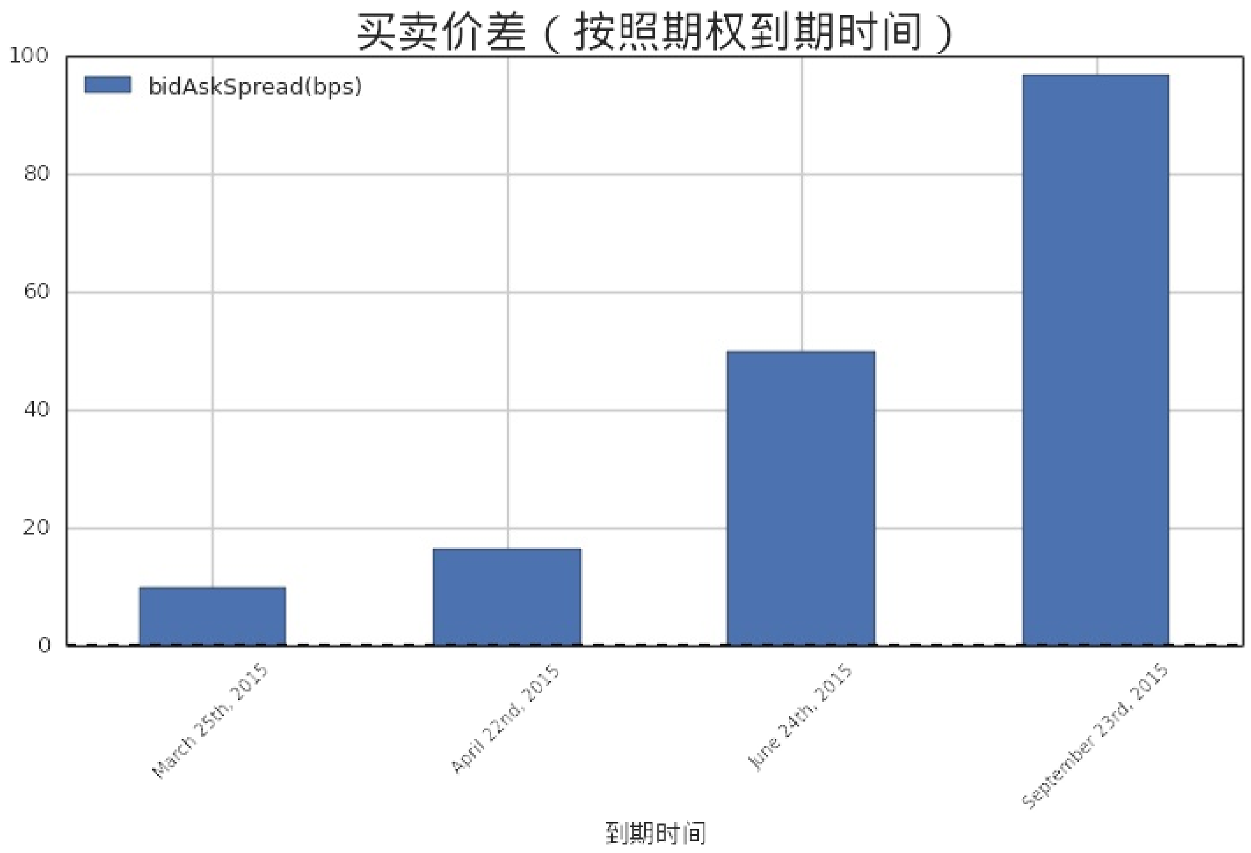
```
res = pd.read_csv('option_data.csv', parse_dates=['pdDateTime'])
res['timeStamp'] = res['dataDate'] + ' ' + res['dataTime']
res['timeStamp'] = pd.to_datetime(res['timeStamp'])
res.optionId = res.optionId.astype('str')
res = res.drop('Unnamed: 0', axis=1)
res.pdDateTime = res.pdDateTime.apply(lambda x:Date(x.year,x.mon
th,x.day))
print('开始日期： ' + res['dataDate'].iloc[0])
print('结束日期： ' + res['dataDate'].iloc[-1])
print('Market Sample: ')
res[['dataDate', 'dataTime', 'optionId', 'lastPrice', 'bidPrice1'
, 'askPrice1', 'lastPrice(vol)']].head()

开始日期： 2015-03-05
结束日期： 2015-03-09
Market Sample:
```

	dataDate	dataTime	optionId	lastPrice	bidPrice1	askPrice1
0	2015-03-05	09:30:00	10000001	0.1677	0.1717	0.1765
1	2015-03-05	09:30:14	10000001	0.1717	0.1717	0.1765
2	2015-03-05	09:30:15	10000001	0.1717	0.1610	0.1798
3	2015-03-05	09:30:16	10000001	0.1678	0.1610	0.1798
4	2015-03-05	09:30:18	10000001	0.1798	0.1641	0.1798

1. 买卖价差分析

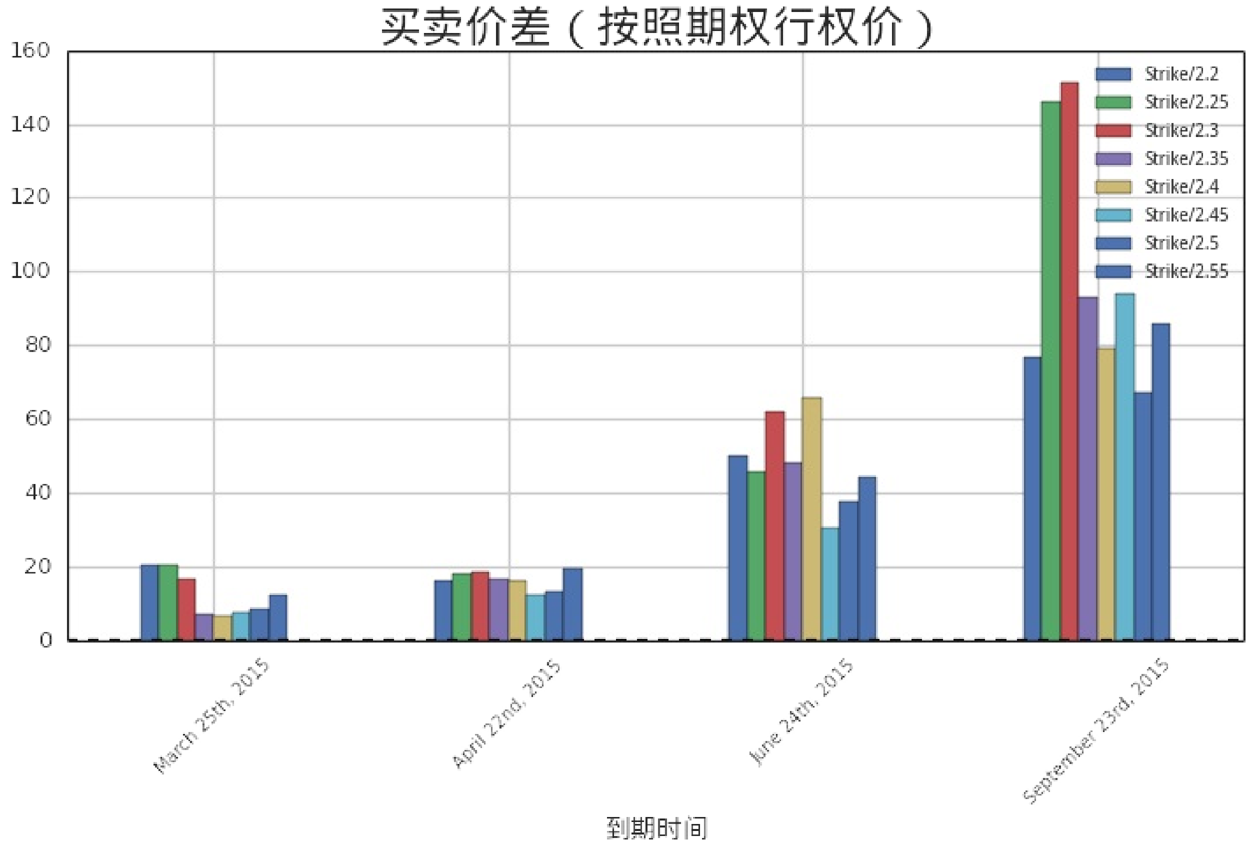
1.1 买卖价差（到期时间）

```
bidAskSample = res[[u'optionId', 'pdDateTime', 'dataDate', 'cont
ractType', 'strikePrice', 'bidAskSpread(bps)']]
bidAskSample.columns = ['optionId', 'maturity', 'tradingDate', '
contractType', 'strikePrice', 'bidAskSpread(bps)']

tmp = bidAskSample.groupby(['maturity'])[['bidAskSpread(bps)']]
ax = tmp.mean().plot(kind = 'bar', figsize = (12,6), rot = 45)
ax.set_title(u'买卖价差（按照期权到期时间）', fontproperties = font,
fontsize = 25)
ax.set_xlabel(u'到期时间', fontproperties = font, fontsize = 15)

<matplotlib.text.Text at 0x7798290>
```



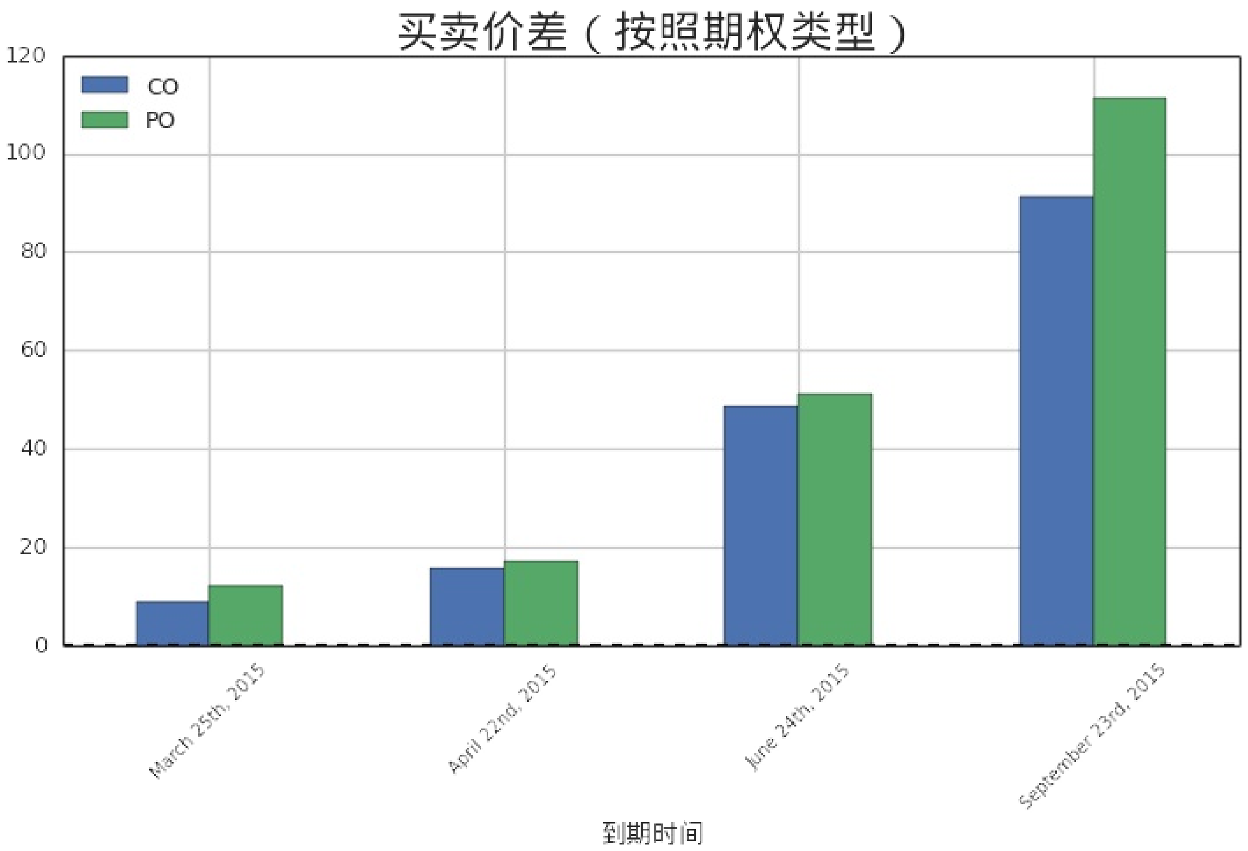
1.2 买卖价差（行权价）

```
tmp = bidAskSample.groupby(['maturity', 'strikePrice'])[['bidAskSpread(bps)']].mean().unstack()
ax = tmp.plot(kind = 'bar', figsize = (12,6), legend = True, rot = 45)
patches, labels = ax.get_legend_handles_labels()
labels = ['Strike/' + l.strip('()').split()[1] for l in labels]
ax.legend(patches, labels, loc='best', prop = font)
ax.set_title(u'买卖价差（按照期权行权价）', fontproperties = font, fontsize = 25)
ax.set_xlabel(u'到期时间', fontproperties = font, fontsize = 15)

<matplotlib.text.Text at 0x5bc08d0>
```



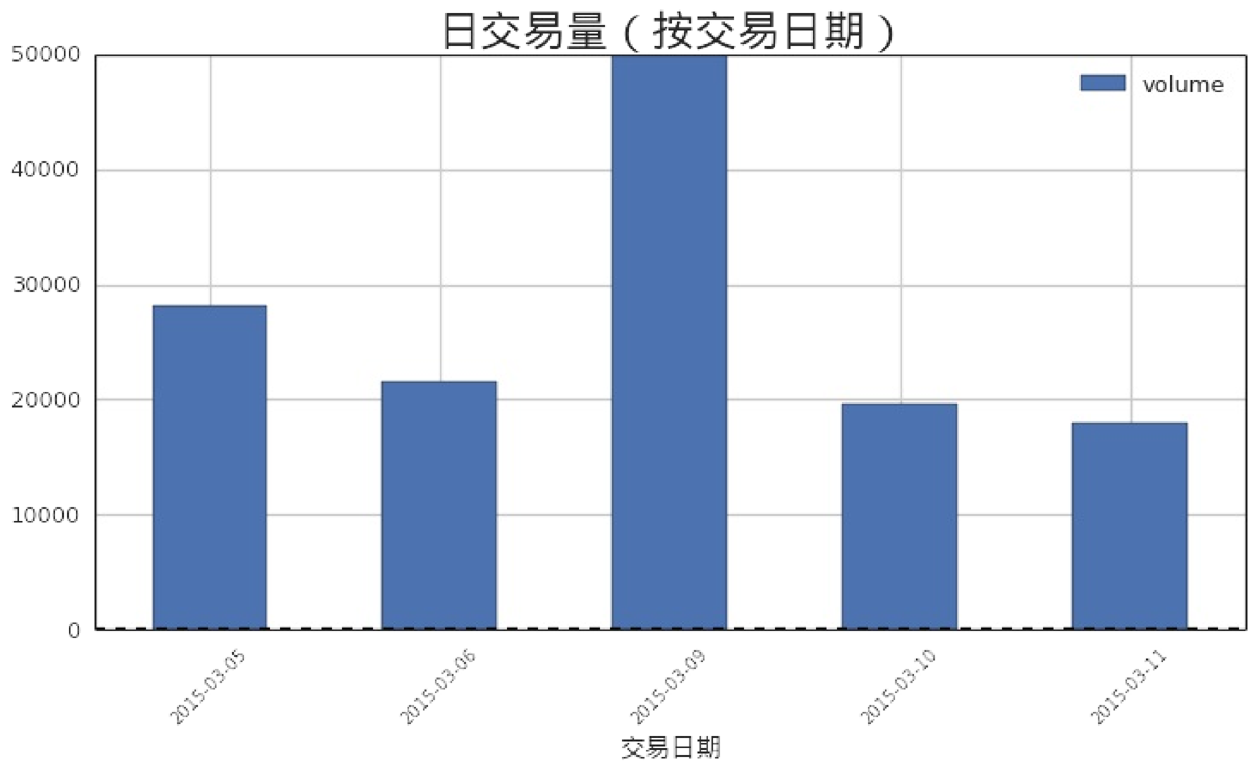
1.3 买卖价差（期权类型）

```
tmp = bidAskSample.groupby(['maturity', 'contractType'])[['bidAskSpread(bps)']].mean().unstack()
ax = tmp.plot(kind = 'bar', figsize = (12,6), rot = 45)
patches, labels = ax.get_legend_handles_labels()
labels = [l.strip('()').split()[1] for l in labels]
ax.legend(patches, labels, loc='best')
ax.set_title(u'买卖价差（按照期权类型）', fontproperties = font, fontsize = 25)
ax.set_xlabel(u'到期时间', fontproperties = font, fontsize = 15)

<matplotlib.text.Text at 0x7a8d7d0>
```



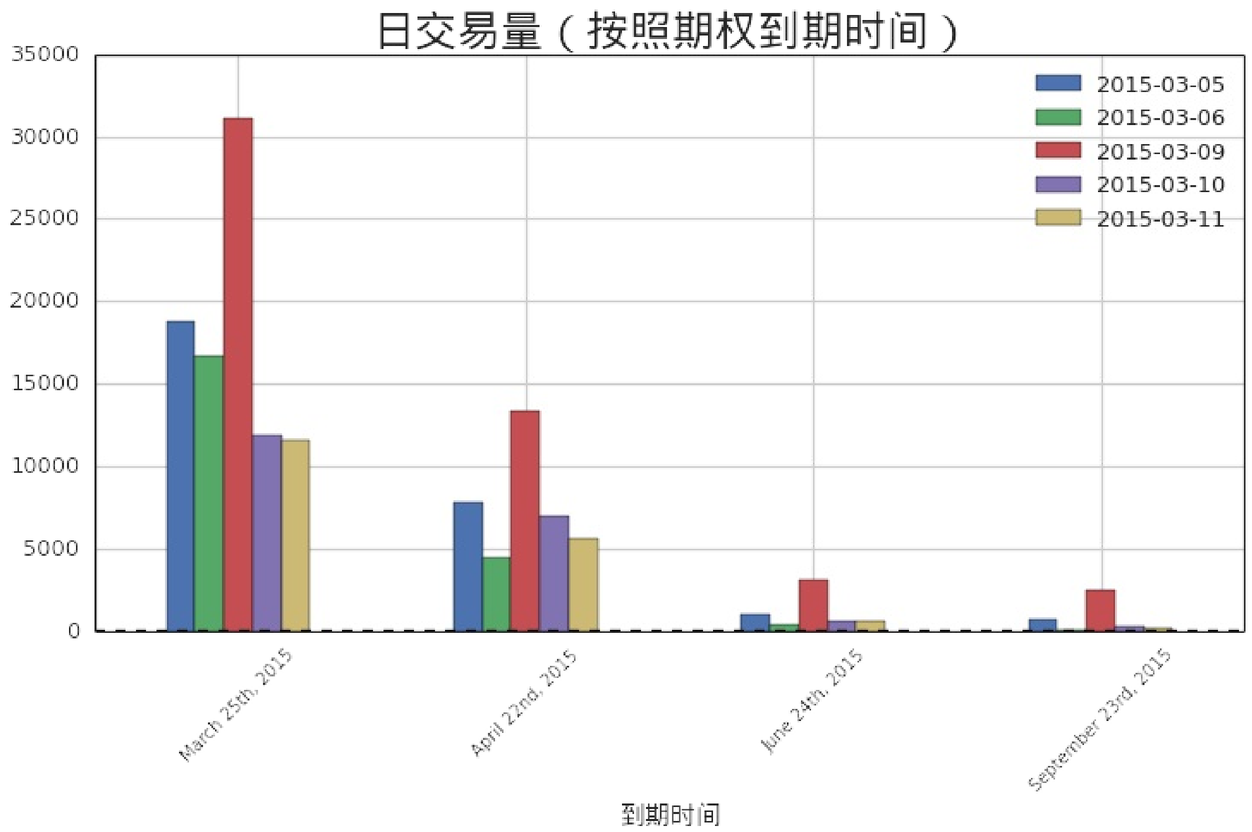
2. 日交易量分析

```
volumeSample = res[[u'optionId', 'pdDateTime', 'dataDate', 'cont
ractType', 'strikePrice', 'volume']]
volumeSample.columns = ['optionId', 'maturity', 'tradingDate', '
contractType', 'strikePrice', 'volume']
tmp = volumeSample.groupby(['tradingDate'])[['volume']].sum()
ax = tmp.plot(kind = 'bar', figsize = (12,6), rot = 45)
ax.set_title(u'日交易量（按交易日期）', fontproperties = font, fonts
ize = 25)
ax.set_xlabel(u'交易日期', fontproperties = font, fontsize = 15)

<matplotlib.text.Text at 0x7a72d90>
```



2.1 日交易量（到期时间）

```
tmp = volumeSample.groupby(['maturity', 'tradingDate'])[['volume'
]].sum().unstack()
ax = tmp.plot(kind = 'bar', figsize = (12,6), rot = 45)
patches, labels = ax.get_legend_handles_labels()
labels = [l.strip('()').split()[1] for l in labels]
ax.legend(patches, labels, loc='best')
ax.set_title(u'日交易量（按照期权到期时间）', fontproperties = font,
fontsize = 25)
ax.set_xlabel(u'到期时间', fontproperties = font, fontsize = 15)
```



每个交易日不同到期期限期权的交易量：

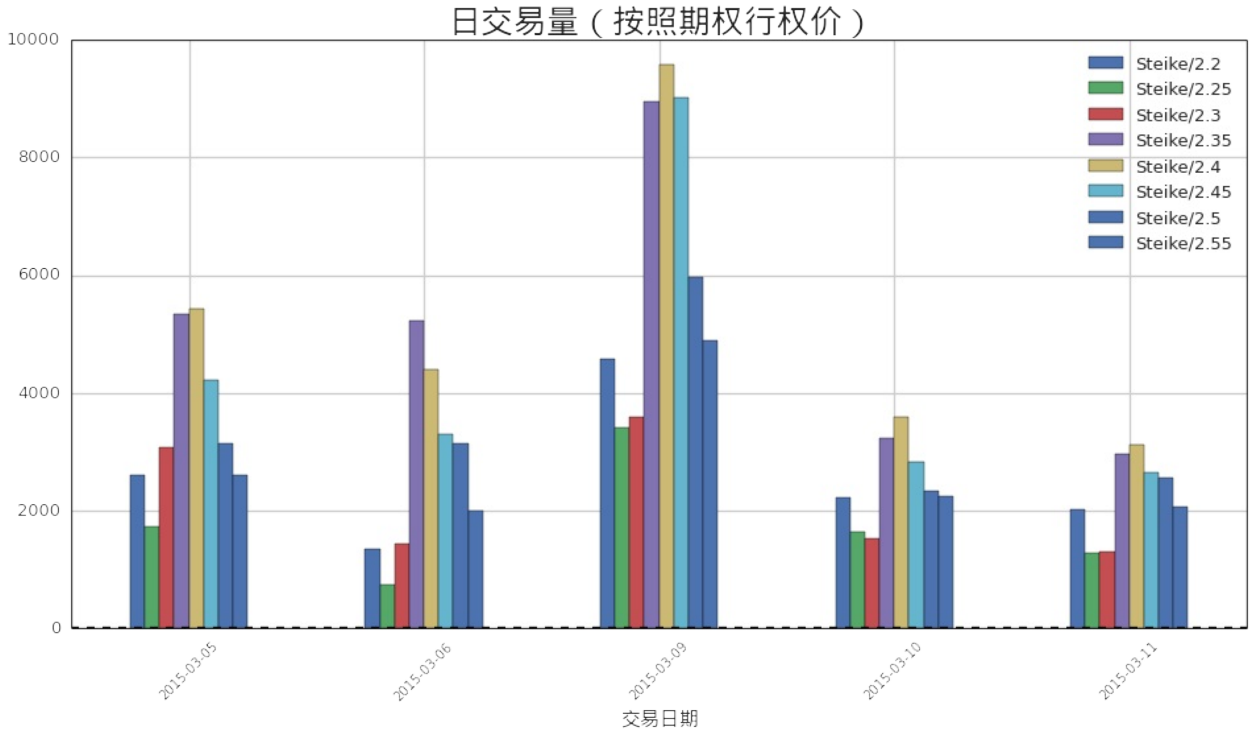
```
tmp
```

	volume				
tradingDate	2015-03-05	2015-03-06	2015-03-09	2015-03-10	2015
maturity					
March 25th, 2015	18767.0000	16704.0000	31115.0000	11888.0000	1156
April 22nd, 2015	7791.0000	4468.0000	13355.0000	6909.0000	5632
June 24th, 2015	965.0000	326.0000	3091.0000	619.0000	604.
September 23rd, 2015	635.0000	101.0000	2426.0000	240.0000	178.

2.2 日交易量（行权价）

```
tmp = volumeSample.groupby(['tradingDate','strikePrice'])[['volume']].sum().unstack()
ax = tmp.plot(kind = 'bar', figsize = (16,8),  rot = 45)
patches, labels = ax.get_legend_handles_labels()
labels = ['Strike/' + l.strip('()').split()[1] for l in labels]
ax.legend(patches, labels, loc='best')
ax.set_title(u'日交易量（按照期权行权价）', fontproperties = font, fontsize = 25)
ax.set_xlabel(u'交易日期', fontproperties = font, fontsize = 15)
```

```
<matplotlib.text.Text at 0x7fa5610>
```



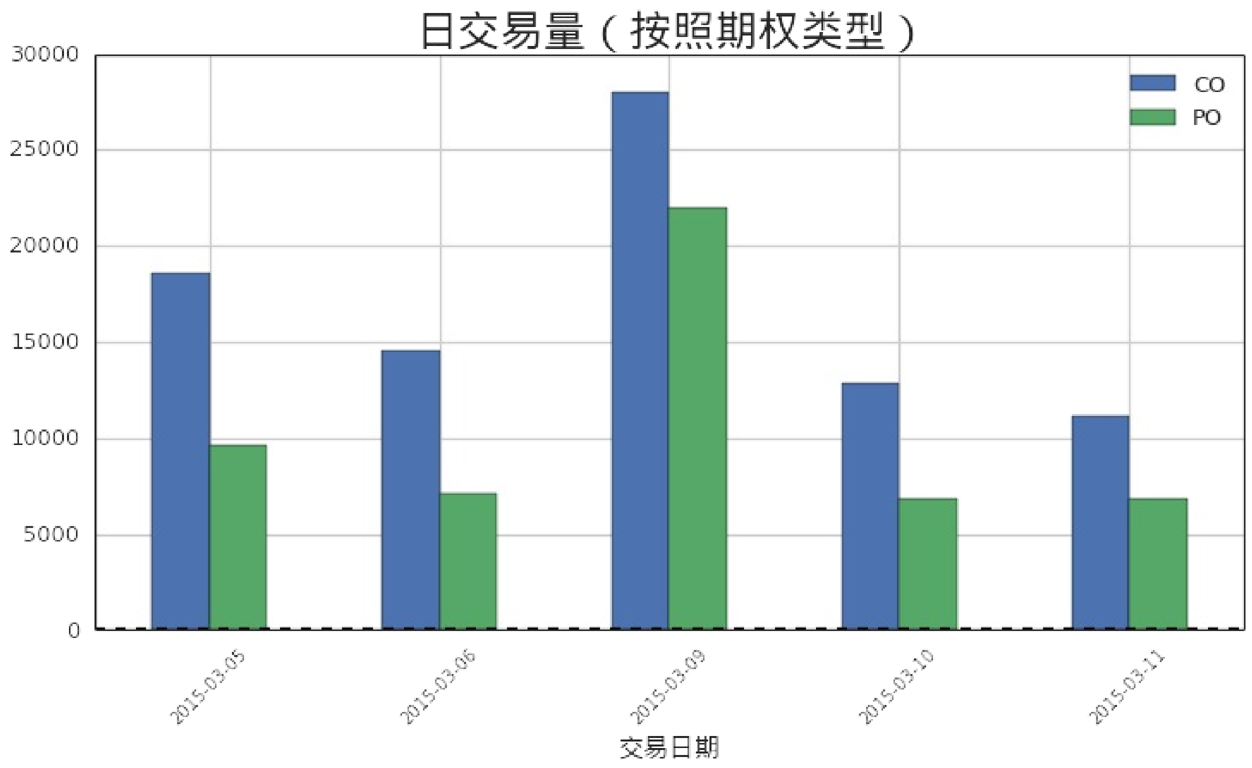
每个交易日不同行权价期权的交易量：

```
tmp
```

	volume				
strikePrice	2.2000	2.2500	2.3000	2.3500	2.4000
tradingDate					
2015-03-05	2597.0000	1725.0000	3077.0000	5351.0000	5430.000
2015-03-06	1352.0000	750.0000	1435.0000	5219.0000	4395.000
2015-03-09	4576.0000	3407.0000	3599.0000	8954.0000	9564.000
2015-03-10	2225.0000	1649.0000	1532.0000	3237.0000	3588.000
2015-03-11	2021.0000	1286.0000	1299.0000	2959.0000	3121.000

2.3 日交易量（期权类型）

```
tmp = volumeSample.groupby(['tradingDate','contractType'])[['vol
ume']].sum().unstack()
ax = tmp.plot(kind = 'bar', y = ['volume'],  figsize = (12,6), r
ot = 45)
patches, labels = ax.get_legend_handles_labels()
labels = [l.strip('()').split()[1] for l in labels]
ax.legend(patches, labels, loc='best')
ax.set_title(u'日交易量（按照期权类型）', fontproperties = font, fon
tsize = 25)
ax.set_xlabel(u'交易日期', fontproperties = font, fontsize = 15)

<matplotlib.text.Text at 0x8813e10>
```

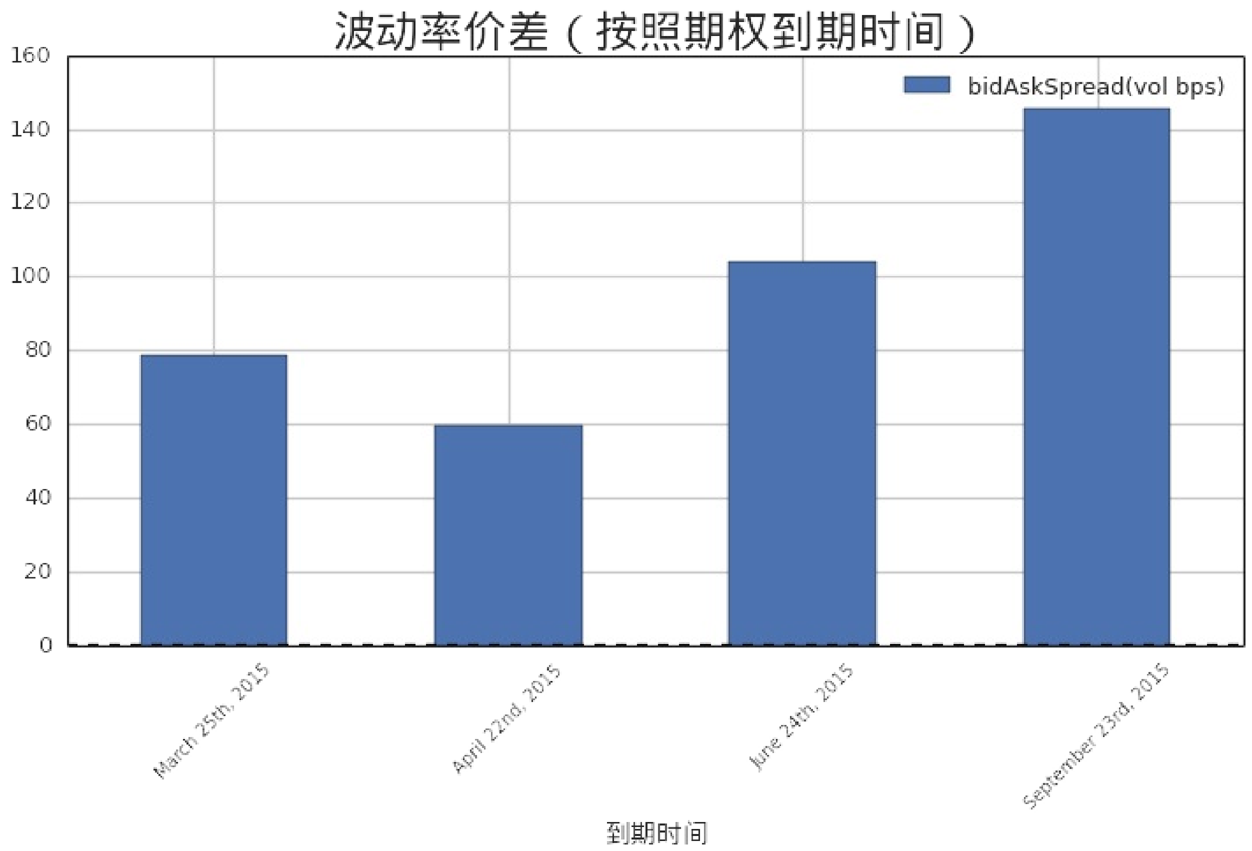


3. 波动率价差分析

```
bidAskVolSample = res[[u'optionId', 'pdDateTime', 'dataDate', 'c
ontractType', 'strikePrice', 'bidAskSpread(vol bps)']]
bidAskVolSample.columns = ['optionId', 'maturity', 'tradingDate'
, 'contractType', 'strikePrice', 'bidAskSpread(vol bps)']
```

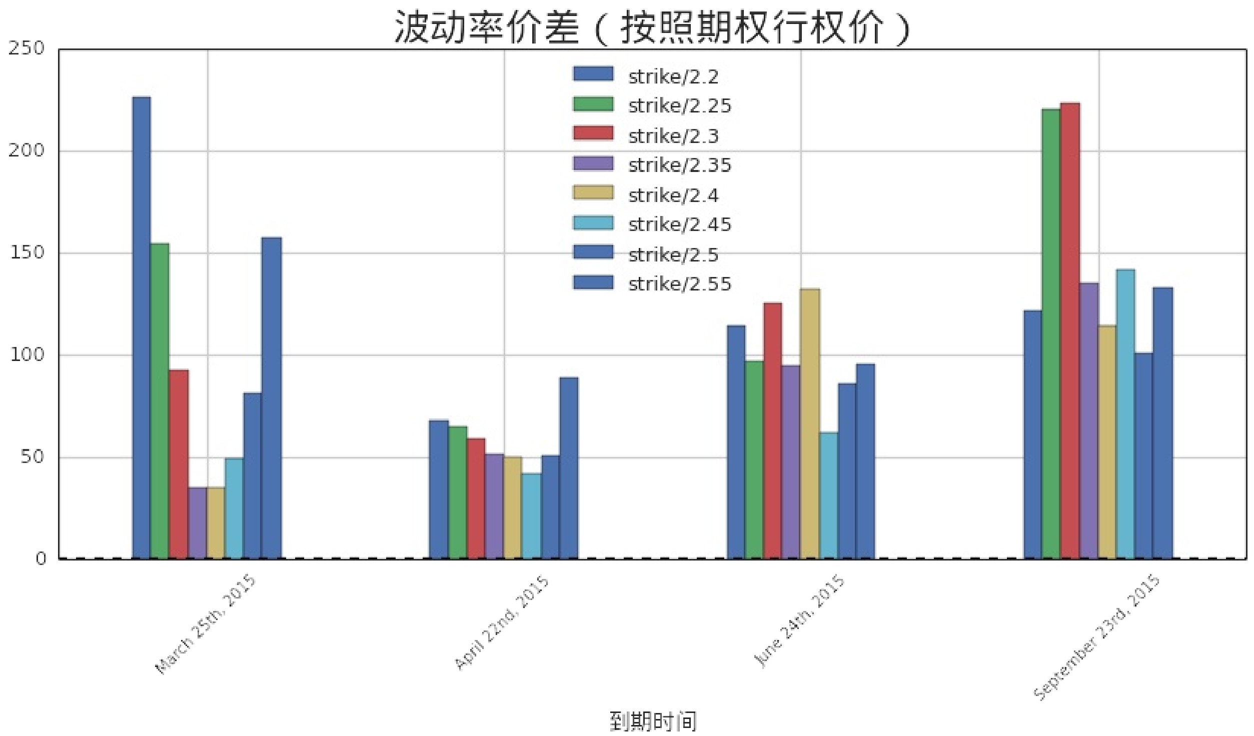
3.1 波动率价差（到期时间）

```
tmp = bidAskVolSample.groupby(['maturity'])[['bidAskSpread(vol b
ps)']]
ax = tmp.mean().plot(kind = 'bar', figsize = (12,6), rot = 45)
ax.set_title(u'波动率价差（按照期权到期时间）', fontproperties = font
, fontsize = 25)
ax.set_xlabel(u'到期时间', fontproperties = font, fontsize = 15)

<matplotlib.text.Text at 0x8c0b7d0>
```



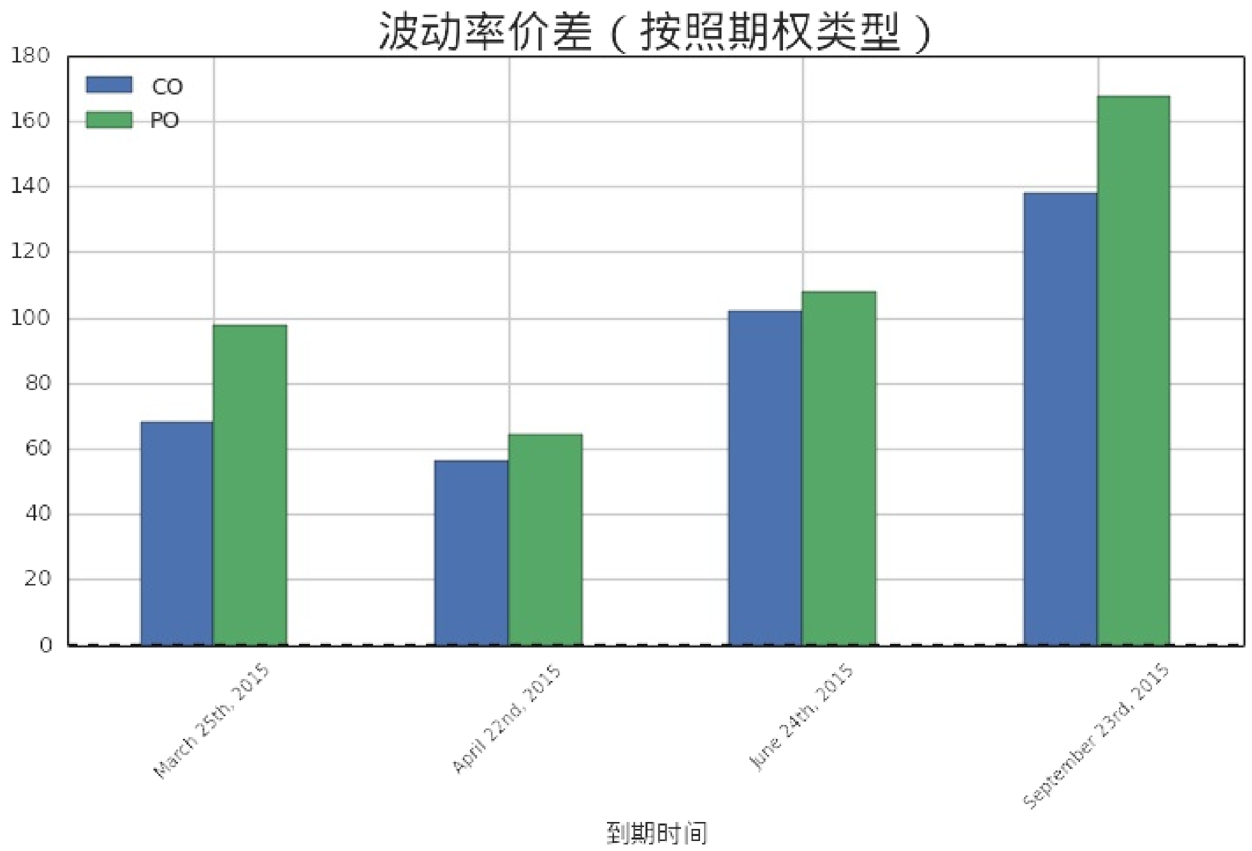
3.2 波动率价差（行权价）

```python
tmp = bidAskVolSample.groupby(['maturity', 'strikePrice'])[['bidAskSpread(vol bps)']].mean().unstack()
ax = tmp.plot(kind = 'bar', figsize = (14,6), legend = True, rot = 45)
patches, labels = ax.get_legend_handles_labels()
labels = ['strike/' + l.strip('()').split()[-1] for l in labels]
ax.legend(patches, labels, loc='best')
ax.set_title(u'波动率价差（按照期权行权价）', fontproperties = font, fontsize = 25)
ax.set_xlabel(u'到期时间', fontproperties = font, fontsize = 15)
```



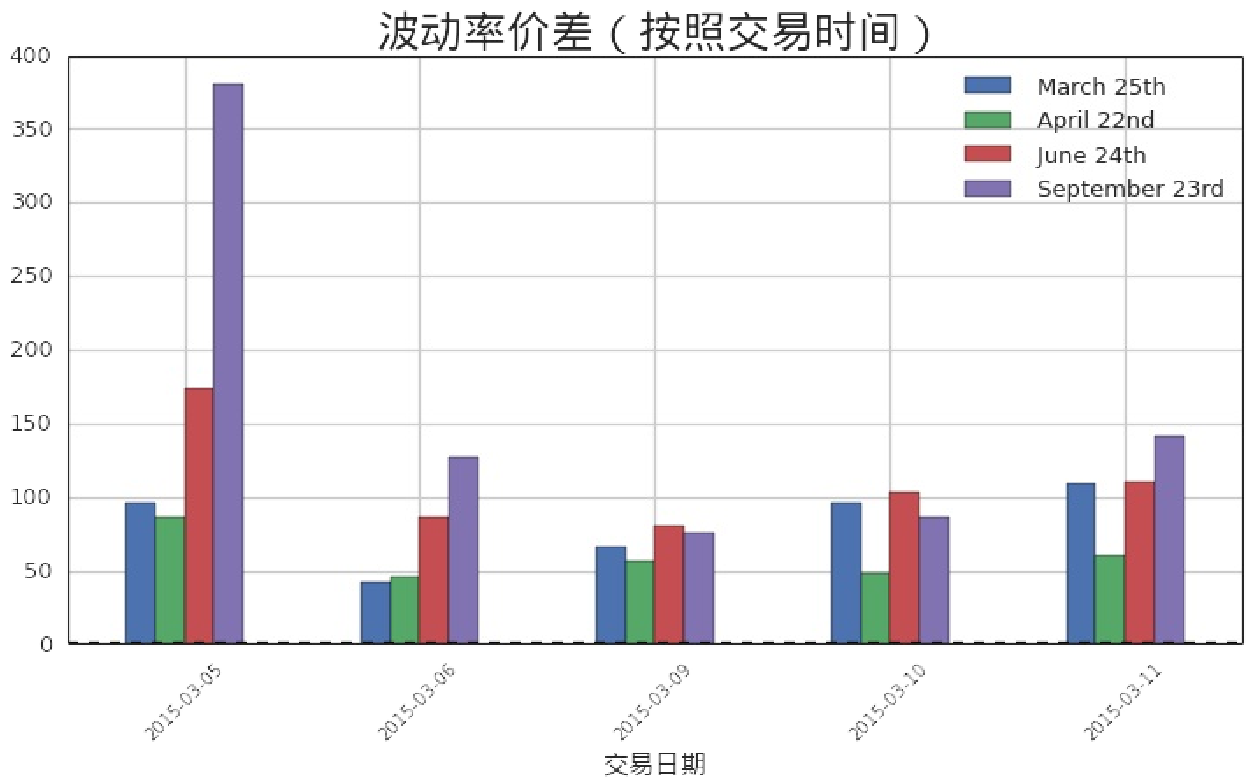
3.3 波动率价差（期权类型）

```
tmp = bidAskVolSample.groupby(['maturity', 'contractType'])[['bidAskSpread(vol bps)']].mean().unstack()
ax = tmp.plot(kind = 'bar', figsize = (12,6), rot = 45)
patches, labels = ax.get_legend_handles_labels()
labels = [l.split()[-1].strip('()') for l in labels]
ax.legend(patches, labels, loc='best')
ax.set_title(u'波动率价差（按照期权类型）', fontproperties = font, fontsize = 25)
ax.set_xlabel(u'到期时间', fontproperties = font, fontsize = 15)
```



3.4 波动率价差（交易时间）

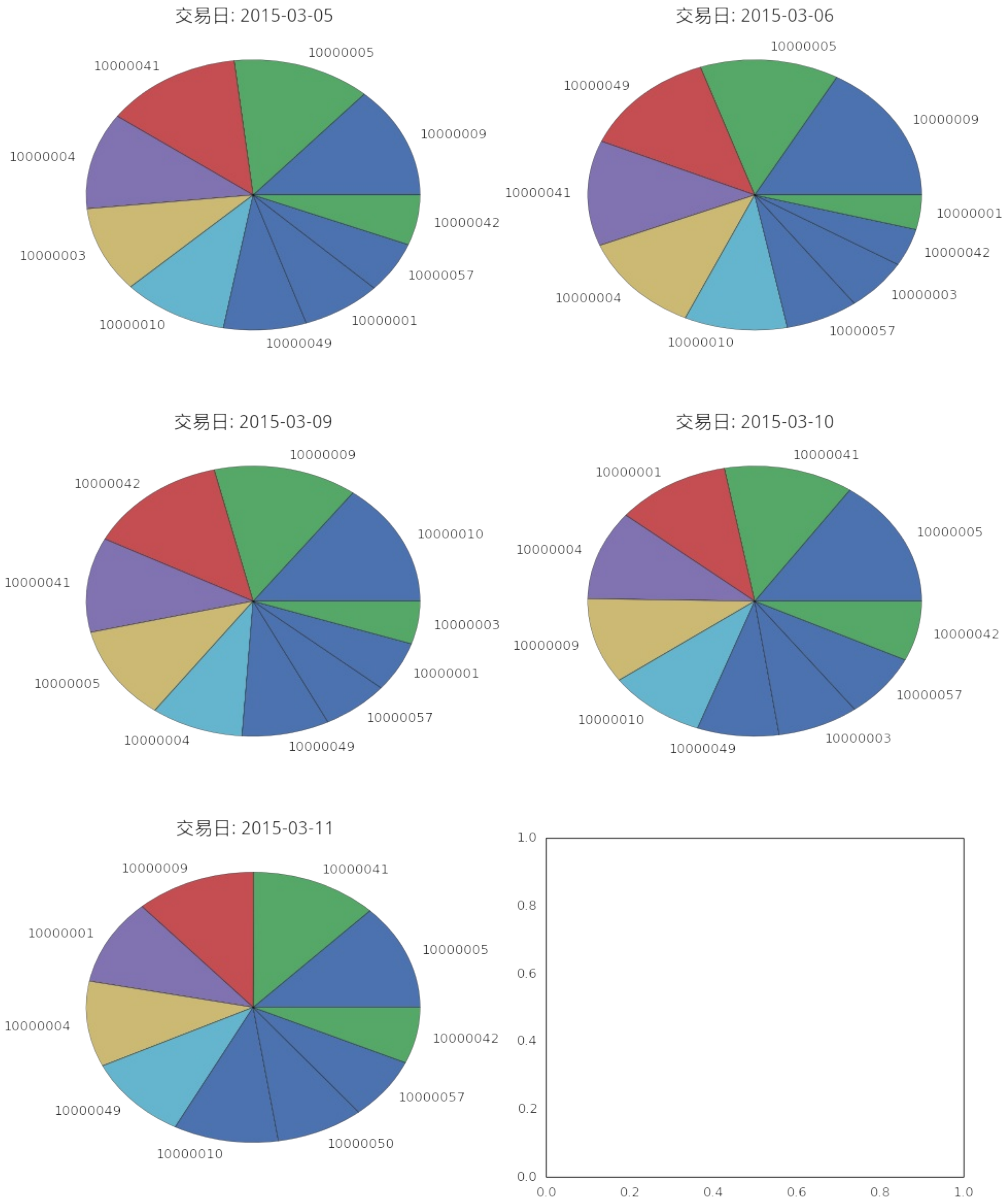
```
tmp = bidAskVolSample.groupby(['tradingDate', 'maturity'])[['bid
AskSpread(vol bps)']].mean().unstack()
ax = tmp.plot(kind = 'bar', figsize = (12,6), rot = 45)
patches, labels = ax.get_legend_handles_labels()
labels = [l.split(',')[1].strip('()') for l in labels]
ax.legend(patches, labels, loc='best')
ax.set_title(u'波动率价差（按照交易时间）', fontproperties = font, f
ontsize = 25)
ax.set_xlabel(u'交易日期', fontproperties = font, fontsize = 15)

<matplotlib.text.Text at 0x8d1fc50>
```



4. 个券分析

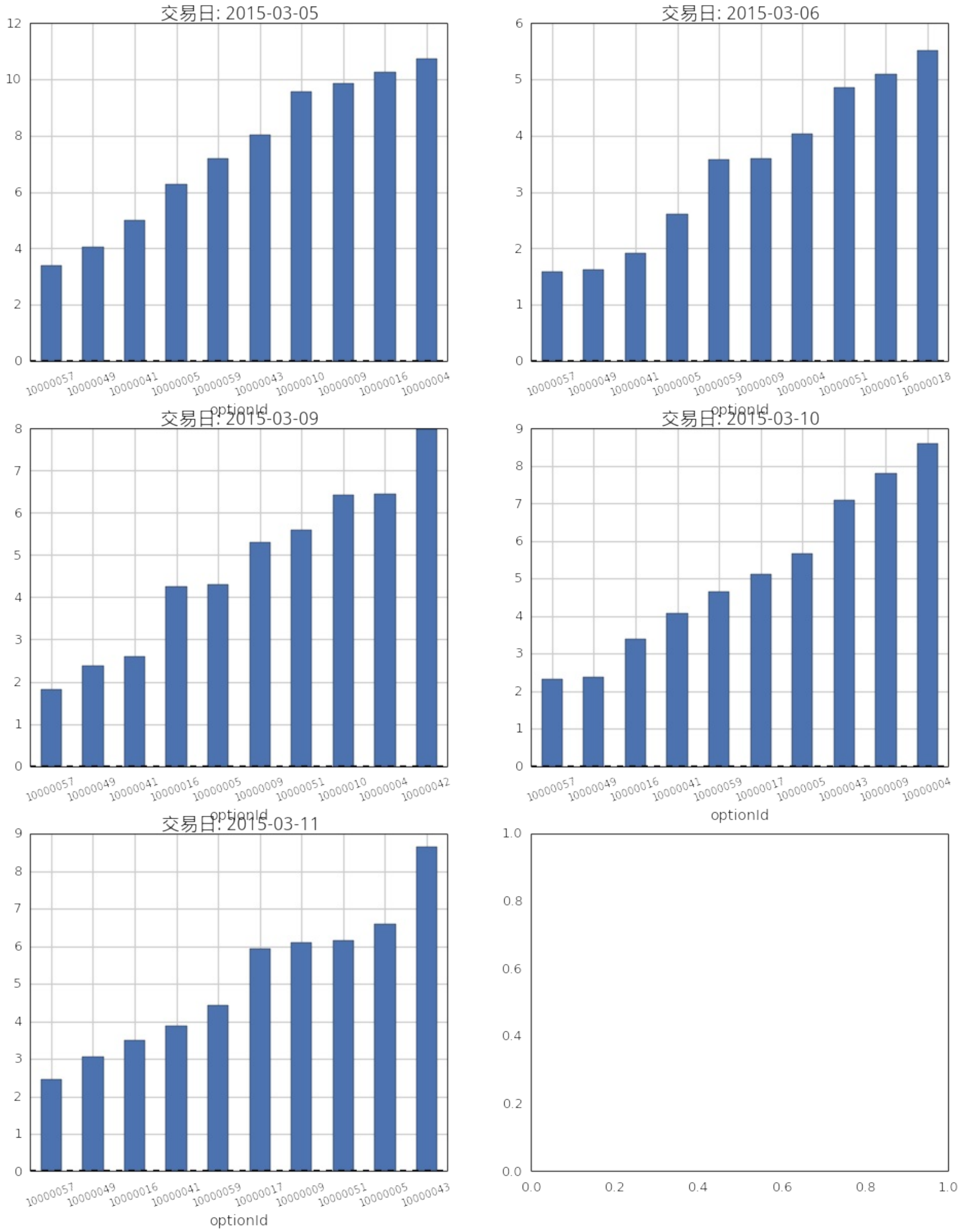
4.1 交易量

```
tmp = volumeSample.groupby(['tradingDate','optionId'])[['volume'
]].sum().unstack()
fig, axs = pylab.subplots(len(tmp)/2 + len(tmp)%2, 2, figsize = (
16,8 * len(tmp)/2))
for i in range(len(tmp)):
    sample = pd.DataFrame(tmp.iloc[i]['volume'])
    sample.columns = ['volume']
    sample = sample.sort('volume', ascending = False)
    sample = sample[:10]
    row = i / 2
    col = i % 2
    sample.plot(kind = 'PIE',y = 'volume', sharex= False, ax = a
xs[row][col], legend = False, rot = 45)
    axs[row][col].set_title(u'交易日： ' + str(tmp.index[i]), font
properties = font, fontsize = 18)
```


交易日: 2015-03-05
10000005
10000041
10000009
10000004
10000042
10000003
10000057
10000010
10000001
10000049
交易日: 2015-03-06
10000005
10000049
10000009
10000041
10000001
10000042
10000003
10000004
10000057
10000010
交易日: 2015-03-09
10000009
10000042
10000010
10000041
10000003
10000001
10000005
10000057
10000004
10000049
交易日: 2015-03-10
10000041
10000001
10000005
10000004
10000042
10000009
10000057
10000010
10000003
10000049
交易日: 2015-03-11
10000009
10000041
10000001
10000005
10000004
10000042
10000057
10000049
10000050
10000010
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

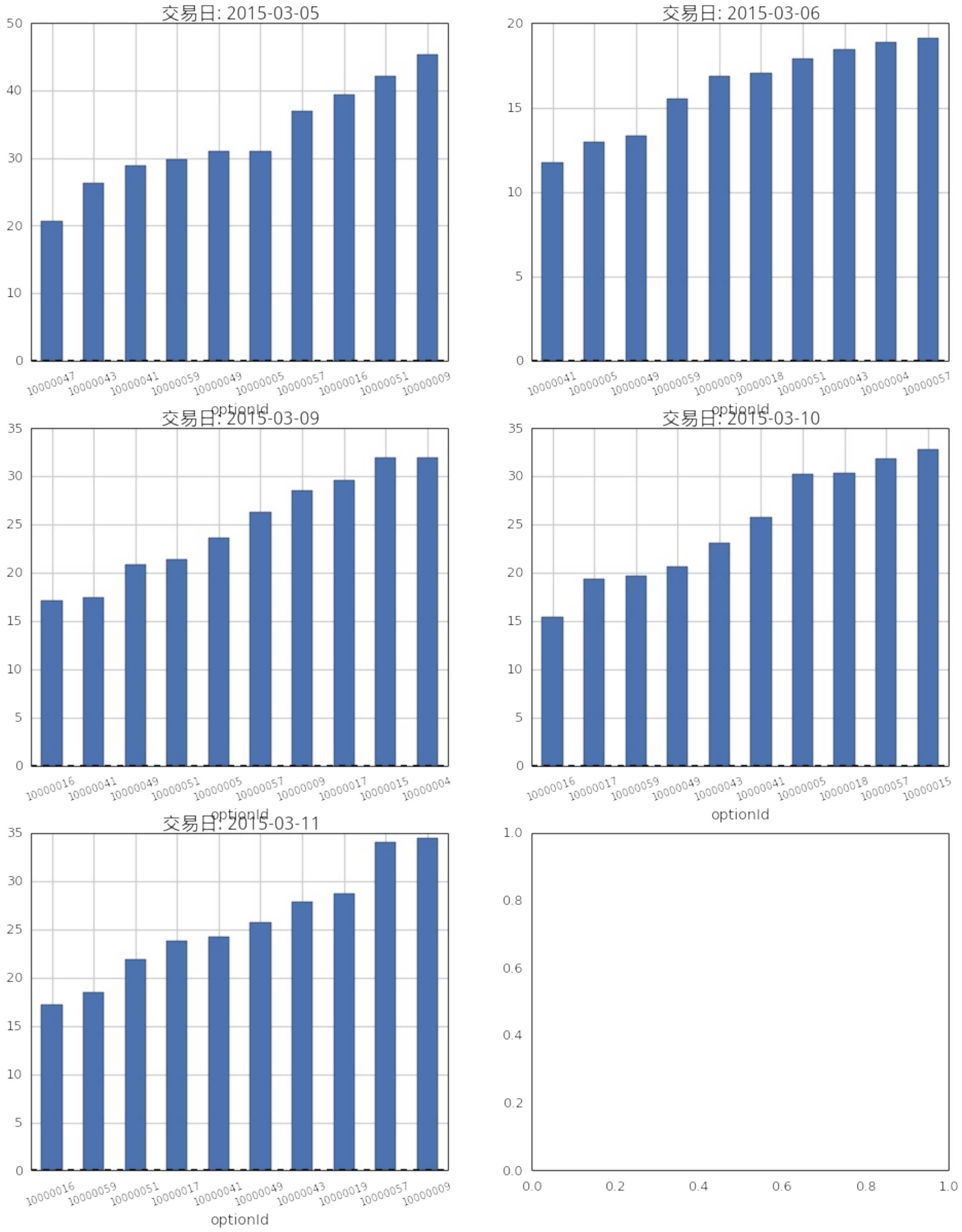
4.2 买卖价差

```
tmp = bidAskSample.groupby(['tradingDate','optionId'])[['bidAskSpread(bps)']].mean().unstack()
fig, axs = pylab.subplots(len(tmp)/2 + len(tmp)%2, 2, figsize = (16,8*len(tmp)/2))
for i in range(len(tmp)):
    sample = pd.DataFrame(tmp.iloc[i]['bidAskSpread(bps)'])
    sample.columns = ['bidAskSpread(bps)']
    sample = sample.sort('bidAskSpread(bps)')
    sample = sample[:10]
    row = i / 2
    col = i % 2
    sample.plot(kind = 'bar',y = 'bidAskSpread(bps)', sharex= False, ax = axs[row][col], legend = False, rot = 20)
    axs[row][col].set_title(u'交易日: ' + str(tmp.index[i]), fontproperties = font, fontsize = 18)
```


交易日: 2015-03-05
12
10
8
6
4
2
0
10000057
10000049
10000041
10000005
10000059
10000043
10000010
10000009
10000016
10000004
optionId
交易日: 2015-03-06
6
5
4
3
2
1
0
10000057
10000049
10000041
10000005
10000059
10000009
10000004
10000051
10000016
10000018
optionId
交易日: 2015-03-09
8
7
6
5
4
3
2
1
0
10000057
10000049
10000041
10000016
10000005
10000009
10000051
10000010
10000004
10000042
optionId
交易日: 2015-03-10
9
8
7
6
5
4
3
2
1
0
10000057
10000049
10000016
10000041
10000059
10000017
10000005
10000043
10000009
10000004
optionId
交易日: 2015-03-11
9
8
7
6
5
4
3
2
1
0
10000057
10000049
10000016
10000041
10000059
10000017
10000009
10000051
10000005
10000043
optionId
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

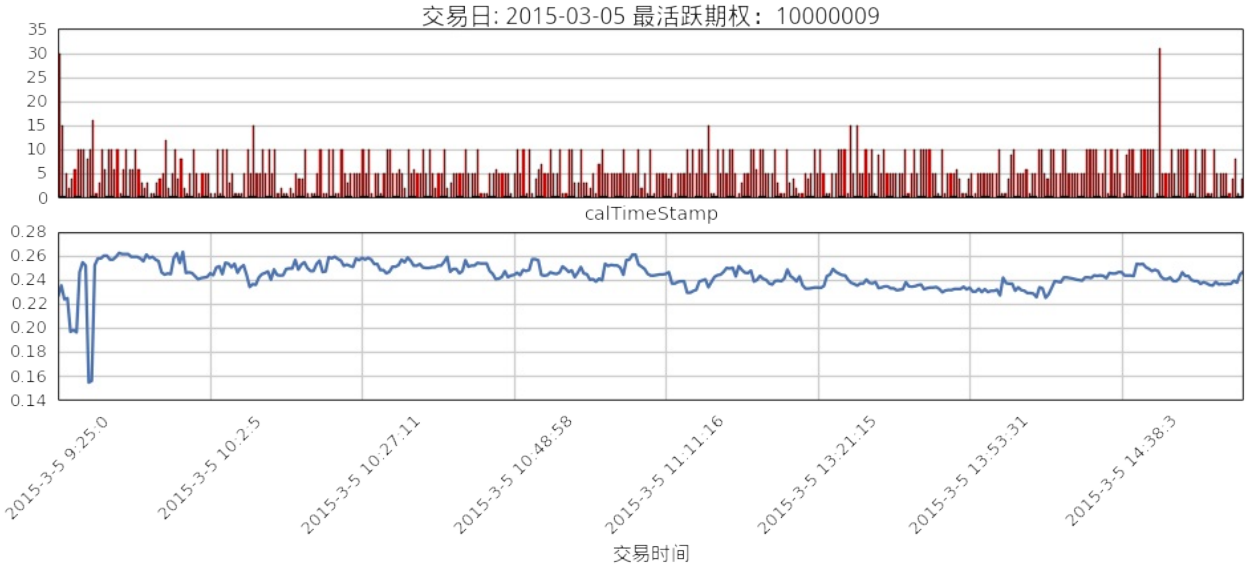
4.3 波动率价差

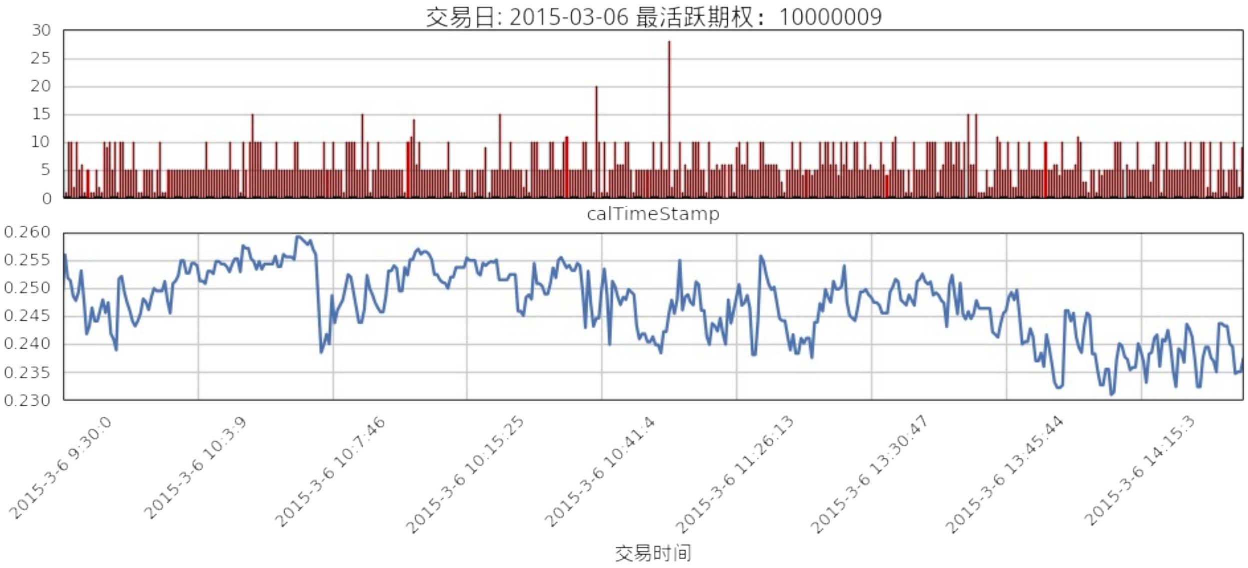
```
tmp = bidAskVolSample.groupby(['tradingDate','optionId'])[['bidAskSpread(vol bps)']].mean().unstack()
fig, axs = pylab.subplots(len(tmp)/2 + len(tmp)%2, 2, figsize = (16,8*len(tmp)/2))
for i in range(len(tmp)):
    sample = pd.DataFrame(tmp.iloc[i]['bidAskSpread(vol bps)'])
    sample.columns = ['bidAskSpread(vol bps)']
    sample = sample.sort('bidAskSpread(vol bps)')
    sample = sample[:10]
    row = i / 2
    col = i % 2
    sample.plot(kind = 'bar',y = 'bidAskSpread(vol bps)', sharex = False, ax = axs[row][col], legend = False, rot = 20)
    axs[row][col].set_title(u'交易日： ' + str(tmp.index[i]), fontproperties = font, fontsize = 18)
```

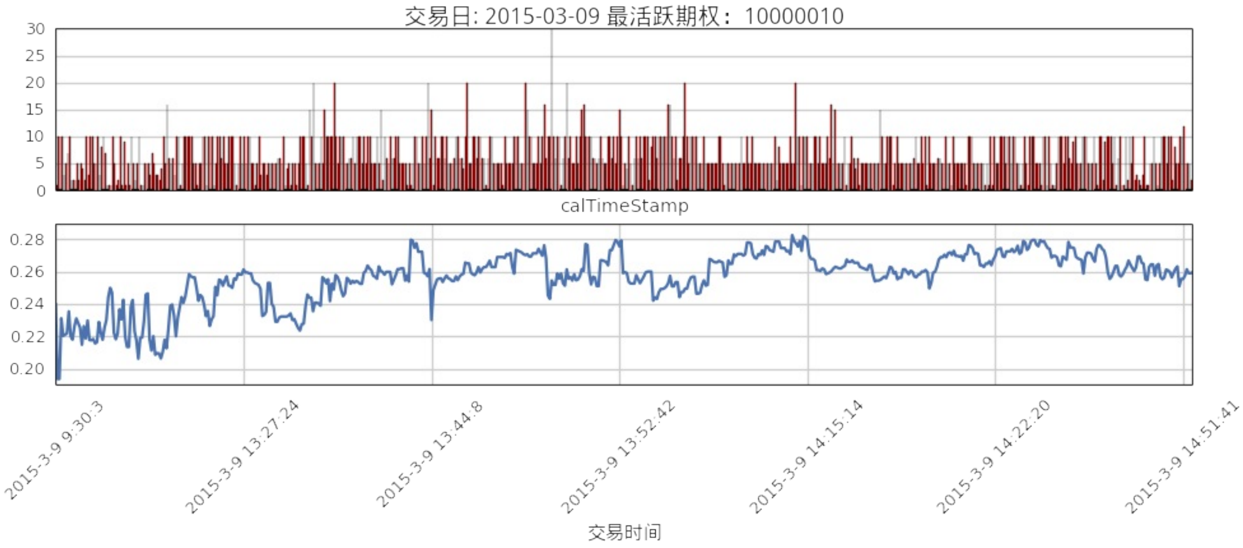

交易日: 2015-03-05
50
40
30
20
10
0
10000047
10000043
10000041
10000059
10000049
10000005
10000057
10000016
10000051
10000009
optionId
交易日: 2015-03-06
20
15
10
5
0
10000041
10000005
10000049
10000059
10000009
10000018
10000051
10000043
10000004
10000057
optionId
交易日: 2015-03-09
35
30
25
20
15
10
5
0
10000016
10000041
10000049
10000051
10000005
10000057
10000009
10000017
10000015
10000004
optionId
交易日: 2015-03-10
35
30
25
20
15
10
5
0
10000016
10000017
10000059
10000049
10000043
10000041
10000005
10000018
10000057
10000015
optionId
交易日: 2015-03-11
35
30
25
20
15
10
5
0
10000016
10000059
10000051
10000017
10000041
10000049
10000043
10000019
10000057
10000009
optionId
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

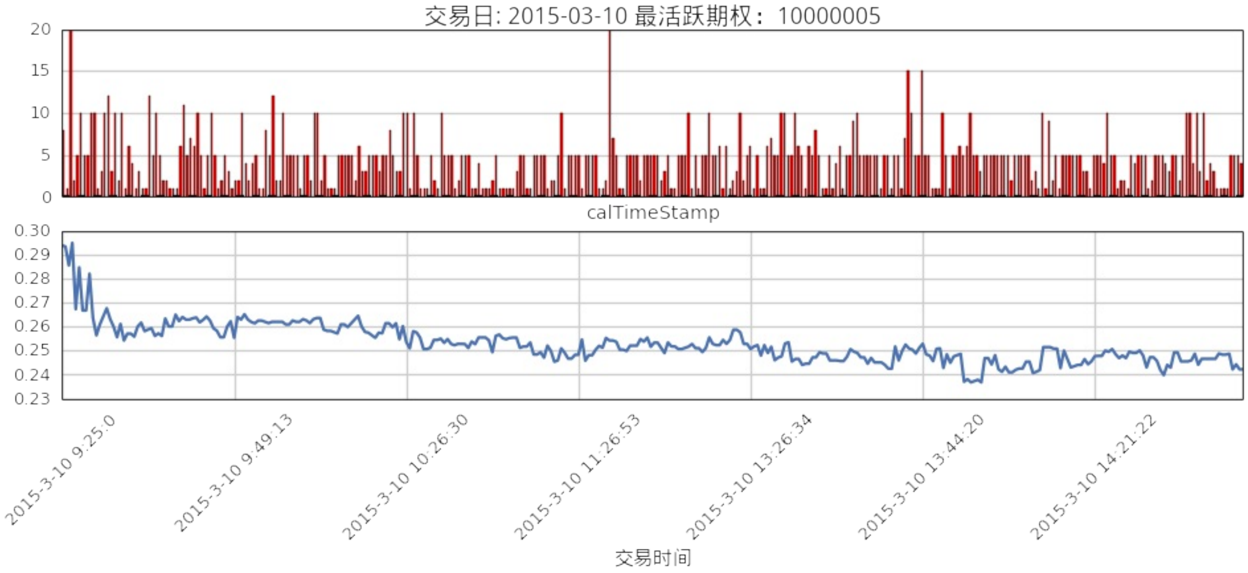
4.4 时间序列分析

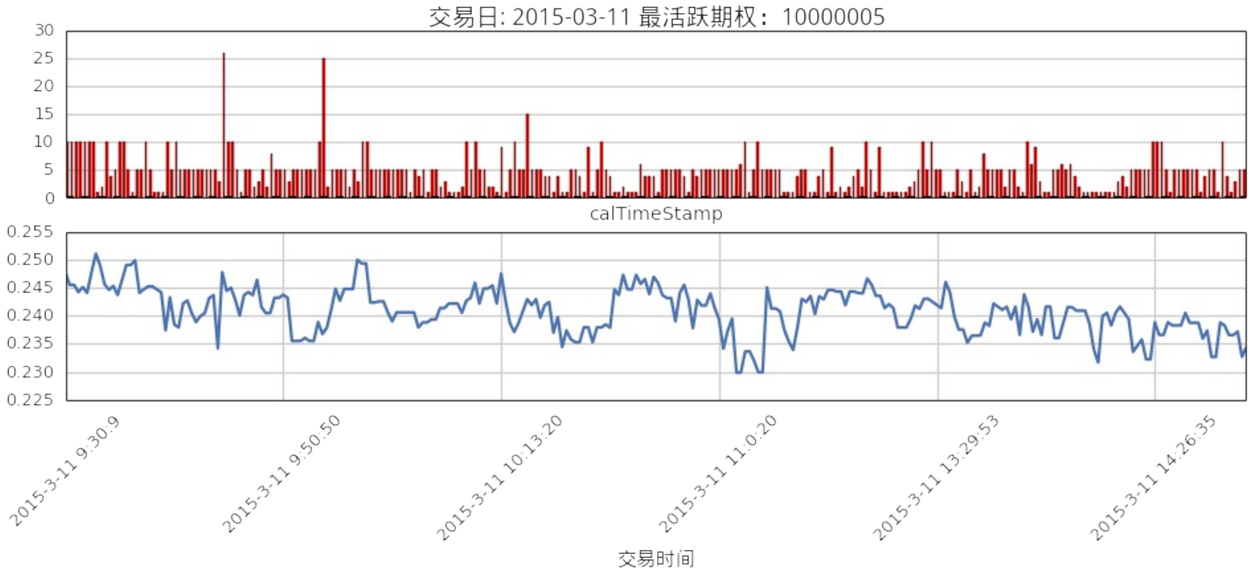
```
tmp = volumeSample.groupby(['tradingDate','optionId'])[['volume'
]].sum().unstack()

for i, d in enumerate(tmp.index):
    fig, axs = pylab.subplots(2, 1, figsize = (16,5))
    sample = tmp.loc(d)
    sample = sample[d]
    sample.sort('volume', ascending = False)

    base = res[res['dataDate'] == d]
    base = base[base.optionId == sample.index[0][1]]
    base.index = range(len(base))

    base['calTimeStamp'] = base.timeStamp.apply(lambda s: DateTi
me(s.year, s.month, s.day, s.hour, s.minute, s.second))
    ax = base.plot(x = 'calTimeStamp', y = ['volume'], kind = 'b
ar', sharex=True, xticks = [], color = 'r', ax = axs[0])
    ax.set_title(u'交易日： ' + unicode(d) + u' 最活跃期权：'+ unico
de(sample.index[0][1]), fontproperties = font, fontsize = 18)
    ax = base.plot(x= 'calTimeStamp', y = ['lastPrice(vol)'], sh
arex=True, legend = True,ax = axs[1], rot = 45)
    ax.set_xlabel(u'交易时间', fontproperties = font, fontsize =
15)
```




交易日: 2015-03-06 最活跃期权：10000009


交易日: 2015-03-09 最活跃期权：10000010


交易日: 2015-03-10 最活跃期权：10000005


交易日: 2015-03-11 最活跃期权：10000005
30
25
20
15
10
5
0
calTimeStamp
0.255
0.250
0.245
0.240
0.235
0.230
0.225
2015-3-11 9:30:9
2015-3-11 9:50:50
2015-3-11 10:13:20
2015-3-11 11:0:20
2015-3-11 13:29:53
2015-3-11 14:26:35
交易时间

基于期权PCR指数的择时策略

> 来源：https://uqer.io/community/share/55bedc1af9f06c91f818c62d

P/C作为市场情绪指标

计算方式

P/C比例作为一种反向情绪指标，是看跌期权的成交量（成交额，持仓量等）与看涨期权的成交量（持仓量）的比值。

指标含义

- 看跌期权的成交量可以作为市场看空力量多寡的衡量；
- 看涨期权的成交量可以描述市场看多力量。

指标应用

- 当P/C比例过小达到一个极端时，被视为市场过度乐观，此时市场将遏制原来的上涨趋势；
- 当P/C比例过大到达另一个极端时，被视为市场过度悲观，此时市场可能出现反弹。

策略思路

比较交易日之前两日的PCR(Put Call Ratio)指数：

- PCR上升时，市场恐慌情绪蔓延，卖出

- PCR下降时，恐慌情绪有所舒缓，买入

注：国内唯一一只期权上证50ETF期权，跟踪标的为华夏上证50ETF(510050)基金

1. 计算历史PCR指数

```
from matplotlib import pylab
import numpy as np
import pandas as pd
import DataAPI
import seaborn as sns
sns.set_style('white')
```

```
def getHistDayOptions(var, date):
    # 使用DataAPI.OptGet，拿到已退市和上市的所有期权的基本信息；
    # 同时使用DataAPI.MktOptdGet，拿到历史上某一天的期权成交信息；
    # 返回历史上指定日期交易的所有期权信息，包括：
    # optID  varSecID  contractType  strikePrice  expDate  tradeDate  closePrice turnoverValue
    # 以optID为index。
    dateStr = date.toISO().replace('-', '')
    optionsMkt = DataAPI.MktOptdGet(tradeDate = dateStr, field = [u"optID", "tradeDate", "closePrice", "turnoverValue"], pandas = "1")
    optionsMkt = optionsMkt.set_index(u"optID")
    optionsMkt.closePrice.name = u"price"

    optionsID = map(str, optionsMkt.index.values.tolist())
    fieldNeeded = ["optID", u"varSecID", u'contractType', u'strikePrice', u'expDate']
    optionsInfo = DataAPI.OptGet(optID=optionsID, contractStatus = [u"DE", u"L"], field=fieldNeeded, pandas="1")
    optionsInfo = optionsInfo.set_index(u"optID")
    options = concat([optionsInfo, optionsMkt], axis=1, join='inner').sort_index()
    return options[options.varSecID==var]

def calDayTurnoverValuePCR(optionVarSecID, date):
    # 计算历史每日的看跌看涨期权交易额的比值
    # PCR: put call ratio
    options = getHistDayOptions(optionVarSecID, date)
    call = options[options.contractType==u"CO"]
    put  = options[options.contractType==u"PO"]
    callTurnoverValue = call.turnoverValue.sum()
    putTurnoverValue = put.turnoverValue.sum()
    return 1.0 * putTurnoverValue / callTurnoverValue

def getHistPCR(beginDate, endDate):
    # 计算历史一段时间内的PCR指数并返回
    optionVarSecID = u"510050.XSHG"
    cal = Calendar('China.SSE')
    dates = cal.bizDatesList(beginDate, endDate)
    dates = map(Date.toDateTime, dates)
    histPCR = pd.DataFrame(0.0, index=dates, columns=['PCR'])
    histPCR.index.name = 'date'
    for date in histPCR.index:
        histPCR['PCR'][date] =  calDayTurnoverValuePCR(optionVarSecID, Date.fromDateTime(date))
    return histPCR

def getDayPCR(date):
    # 计算历史一段时间内的PCR指数并返回
    optionVarSecID = u"510050.XSHG"
    return calDayTurnoverValuePCR(optionVarSecID, date)
```

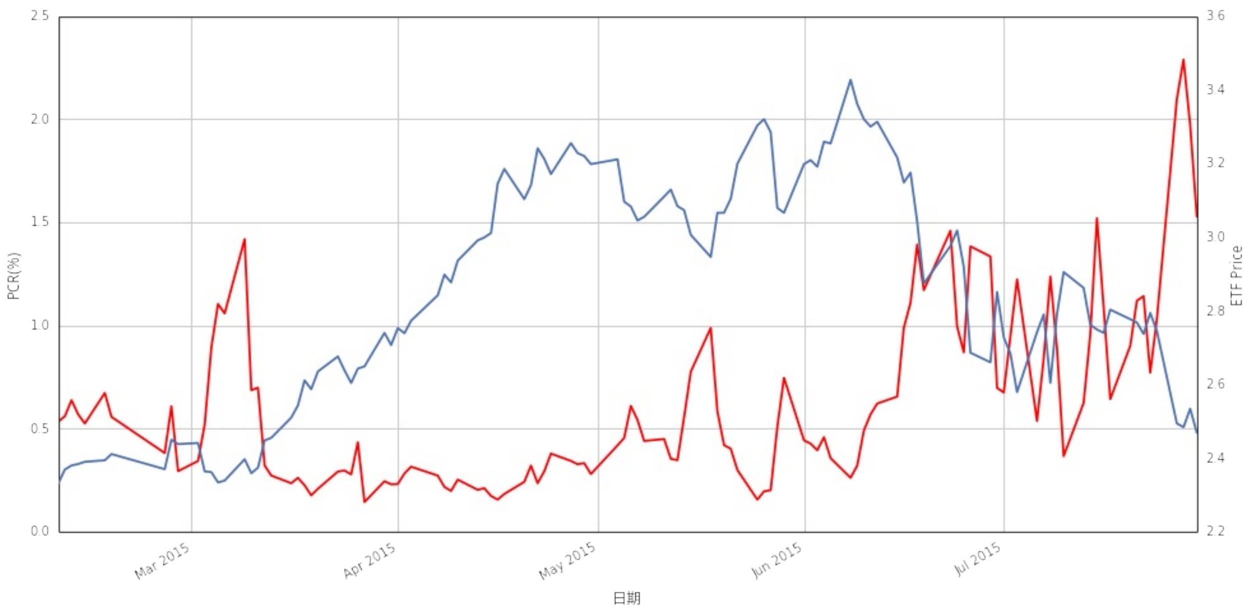
```
secID = '510050.XSHG'
begin = Date(2015, 2, 9)
end = Date(2015, 7, 30)

getHistPCR(begin, end).tail()
```

	PCR
date	
2015-07-24	1.032107
2015-07-27	2.097952
2015-07-28	2.288790
2015-07-29	1.971831
2015-07-30	1.527717

2. PCR指数与华夏上证50ETF基金的走势对比

```
secID = '510050.XSHG'
begin = Date(2015, 2, 9)
end = Date(2015, 7, 30)

# 历史PCR
histPCR = getHistPCR(begin, end)

# 华夏上证50ETF
etf = DataAPI.MktFunddGet(secID, beginDate=begin.toISO().replace(
'-', ''), endDate=end.toISO().replace('-', ''), field=['tradeDat
e', 'closePrice'])
etf['tradeDate'] = pd.to_datetime(etf['tradeDate'])
etf = etf.set_index('tradeDate')
```

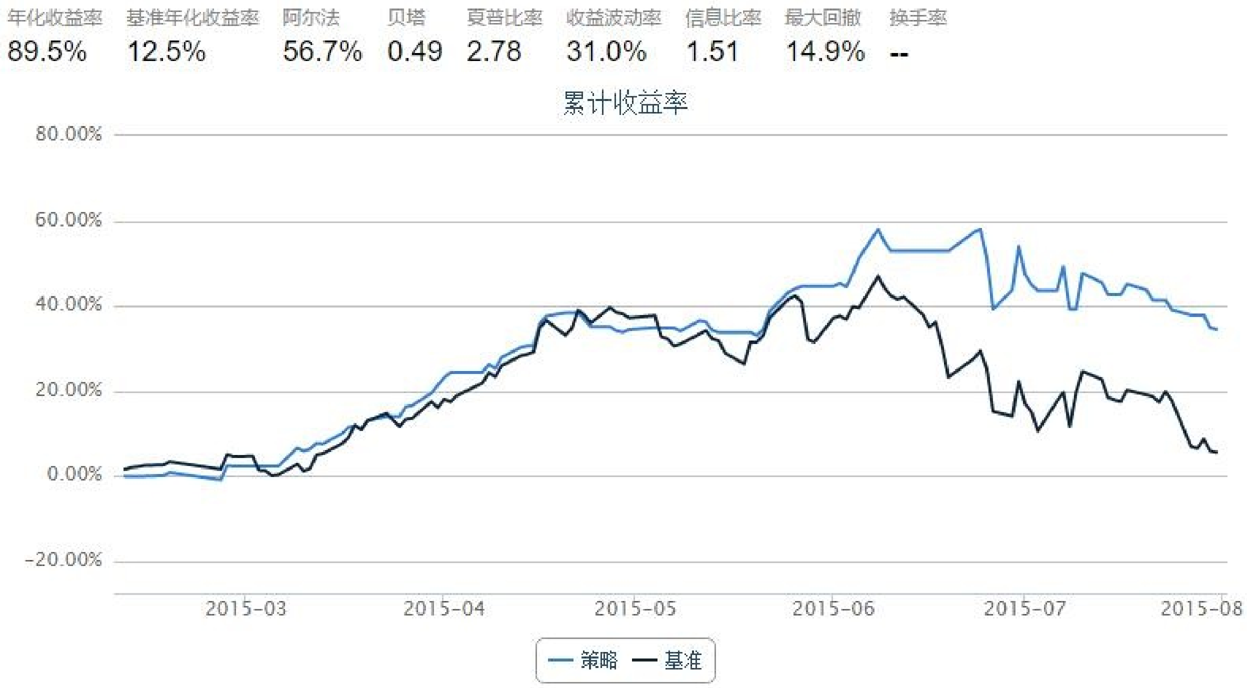
```
font.set_size(12)
pylab.figure(figsize = (16,8))

ax1 = histPCR.plot(x=histPCR.index, y='PCR', style='r')
ax1.set_xlabel(u'日期', fontproperties=font)
ax1.set_ylabel(u'PCR(%)', fontproperties=font)

ax2 = ax1.twinx()
ax2.plot(etf.index,etf.closePrice)
ax2.set_ylabel(u'ETF Price', fontproperties=font)

<matplotlib.text.Text at 0x78a4d90>
```



从上图可以看出，每次PC指标的上升都对应着标的价格的下挫

3. 基于PCR指数的择时策略示例

```
start = datetime(2015, 2, 9)                      # 回测起始时间
end  = datetime(2015, 7, 31)                      # 回测结束时间
benchmark = '510050.XSHG'                          # 策略参考标准
universe = ['510050.XSHG']    # 股票池
capital_base = 100000     # 起始资金
commission = Commission(0.0,0.0)

longest_history = 1
histPCR = getHistPCR(start, end)

def initialize(account):                          # 初始化虚拟账户状态
    account.fund = universe[0]

def handle_data(account):                   # 每个交易日的买入卖出指令
    hist = account.get_history(longest_history)
    fund = account.fund

     #  获取回测当日的前一天日期
    dt = Date.fromDateTime(account.current_date)
    cal = Calendar('China.IB')
    lastTDay = cal.advanceDate(dt,'-1B',BizDayConvention.Preceding)          #计算出倒数第一个交易日
    lastLastTDay = cal.advanceDate(lastTDay,'-1B',BizDayConvention.Preceding)  #计算出倒数第二个交易日
    last_day_str = lastTDay.strftime("%Y-%m-%d")
    last_last_day_str = lastLastTDay.strftime("%Y-%m-%d")

    # 计算买入卖出信号
    try:
        pcr_last = histPCR['PCR'].loc[last_day_str] # 计算短均线值
        pcr_last_last = histPCR['PCR'].loc[last_last_day_str] # 计算长均线值
        long_flag = True if (pcr_last - pcr_last_last) < 0 else False
    except:
        return

    if long_flag:
        if account.position.secpos.get(fund, 0) == 0:
            # 空仓时全仓买入，买入股数为100的整数倍
            approximationAmount = int(account.cash / hist[fund]['closePrice'][-1]/100.0) * 100
            order(fund, approximationAmount)
    else:
        # 卖出时，全仓清空
        if account.position.secpos.get(fund, 0) >= 0:
            order_to(fund, 0)
```



年化收益率	基准年化收益率	阿尔法	贝塔	夏普比率	收益波动率	信息比率	最大回撤	换手率
89.5%	12.5%	56.7%	0.49	2.78	31.0%	1.51	14.9%	--

基于PCR指数上升时空仓、下降时进场的策略来买卖标的，可以比较有效地降低标的大跌的风险

期权每日成交额PC比例计算

> 来源：https://uqer.io/community/share/55bed777f9f06c915418c62f

P/C作为市场情绪指标

计算方式

P/C比例作为一种反向情绪指标，是看跌期权的成交量（成交额，持仓量等）与看涨期权的成交量（持仓量）的比值。

指标含义

- 看跌期权的成交量可以作为市场看空力量多寡的衡量；
- 看涨期权的成交量可以描述市场看多力量。

指标应用

- 当P/C比例过小达到一个极端时，被视为市场过度乐观，此时市场将遏制原来的上涨趋势；
- 当P/C比例过大到达另一个极端时，被视为市场过度悲观，此时市场可能出现反弹。

```
from matplotlib import pylab
import numpy as np
import pandas as pd
import DataAPI
import seaborn as sns
sns.set_style('white')
```

1. 定义计算PCR的函数

此处计算看跌看涨期权每日成交额的比值

```
def getHistDayOptions(var, date):
    # 使用DataAPI.OptGet，拿到已退市和上市的所有期权的基本信息；
    # 同时使用DataAPI.MktOptdGet，拿到历史上某一天的期权成交信息；
    # 返回历史上指定日期交易的所有期权信息，包括：
    # optID  varSecID  contractType  strikePrice  expDate  trade
Date  closePrice turnoverValue
    # 以optID为index。
    vixDateStr = date.toISO().replace('-', '')
    optionsMkt = DataAPI.MktOptdGet(tradeDate = vixDateStr, fiel
d = [u"optID", "tradeDate", "closePrice", "turnoverValue"], pand
as = "1")
    optionsMkt = optionsMkt.set_index(u"optID")
    optionsMkt.closePrice.name = u"price"

    optionsID = map(str, optionsMkt.index.values.tolist())
    fieldNeeded = ["optID", u"varSecID", u'contractType', u'stri
kePrice', u'expDate']
    optionsInfo = DataAPI.OptGet(optID=optionsID, contractStatus
 = [u"DE", u"L"], field=fieldNeeded, pandas="1")
    optionsInfo = optionsInfo.set_index(u"optID")
    options = pd.concat([optionsInfo, optionsMkt], axis=1, join=
'inner').sort_index()
    return options[options.varSecID==var]

def calDayTurnoverValuePCR(optionVarSecID, date):
    # 计算历史每日的看跌看涨期权交易额的比值
    # PCR: put call ratio
    options = getHistDayOptions(optionVarSecID, date)
    call = options[options.contractType==u"CO"]
    put  = options[options.contractType==u"PO"]
    callTurnoverValue = call.turnoverValue.sum()
    putTurnoverValue = put.turnoverValue.sum()
    return 1.0 * putTurnoverValue / callTurnoverValue

def getHistPCR(beginDate, endDate):
    # 计算历史一段时间内的PCR指数并返回
    optionVarSecID = u"510050.XSHG"
    cal = Calendar('China.SSE')
    dates = cal.bizDatesList(beginDate, endDate)
    dates = map(Date.toDateTime, dates)
    histPCR = pd.DataFrame(0.0, index=dates, columns=['PCR'])
    histPCR.index.name = 'date'
    for date in histPCR.index:
        histPCR['PCR'][date] =  calDayTurnoverValuePCR(optionVar
SecID, Date.fromDateTime(date))
    return histPCR

def getDayPCR(date):
    # 计算历史某一天的PCR指数并返回
    optionVarSecID = u"510050.XSHG"
    return calDayTurnoverValuePCR(optionVarSecID, date)
```

2. 计算PCR指标

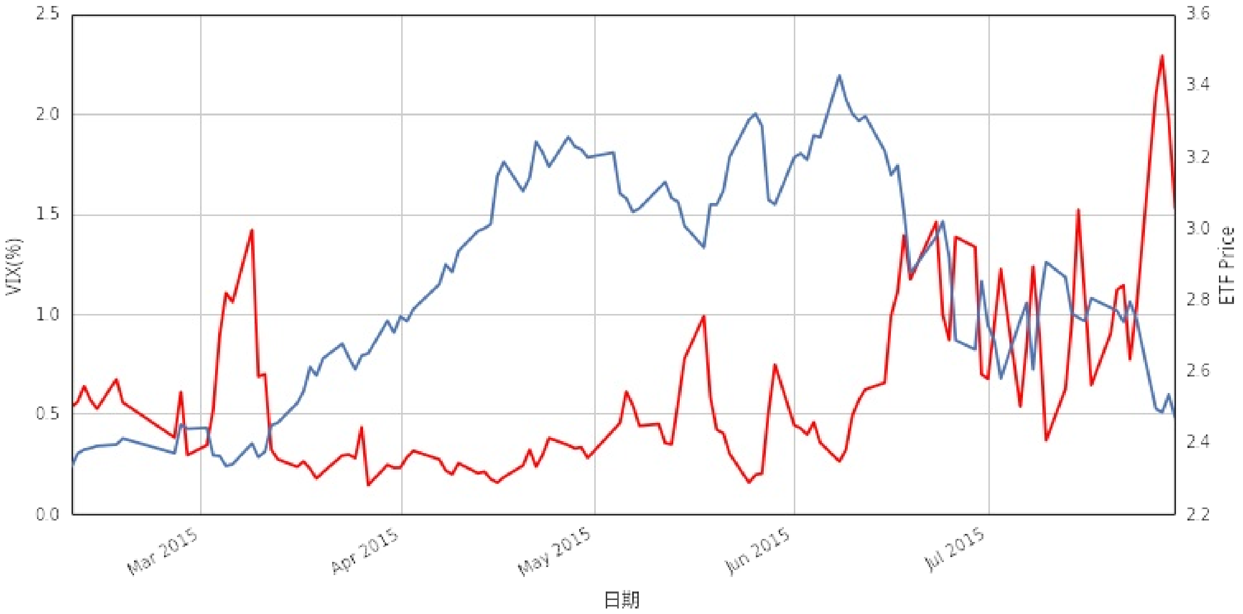
```
begin = Date(2015, 2, 9)
end = Date(2015, 7, 30)

getHistPCR(begin, end).tail()
```

	PCR
date	
2015-07-24	1.032107
2015-07-27	2.097952
2015-07-28	2.288790
2015-07-29	1.971831
2015-07-30	1.527717

```
date = Date(2015, 7, 30)

getDayPCR(date)

1.5277173819619587
```

3. PC指标历史走势

```
secID = '510050.XSHG'
begin = Date(2015, 2, 9)
end = Date(2015, 7, 30)

# 历史PCR
histPCR = getHistPCR(begin, end)

# 华夏上证50ETF
etf = DataAPI.MktFunddGet(secID, beginDate=begin.toISO().replace(
'-', ''), endDate=end.toISO().replace('-', ''), field=['tradeDat
e', 'closePrice'])
etf['tradeDate'] = pd.to_datetime(etf['tradeDate'])
etf = etf.set_index('tradeDate')

font.set_size(12)
pylab.figure(figsize = (12,6))

ax1 = histPCR.plot(x=histPCR.index, y='PCR', style='r')
ax1.set_xlabel(u'日期', fontproperties=font)
ax1.set_ylabel(u'VIX(%)', fontproperties=font)

ax2 = ax1.twinx()
ax2.plot(etf.index,etf.closePrice)
ax2.set_ylabel(u'ETF Price', fontproperties=font)

<matplotlib.text.Text at 0x53797d0>
```



从上图可以看出，每次PC指标的上升都对应着标的价格的下挫

四 期货分析

【前方高能！】Gifts from Santa Claus——股指期货趋势交易研究

来源：https://uqer.io/community/share/567ca9b1228e5b3444688438

股指期货的前世今生

股指期货，名曰期指，乃期货界的明珠，现货对应的是股票指数，其逼格比之大豆棉油不知道甩几条巷子。期指的上市为股票市场提供了很好的风险管理和做策略的工具，矿工们也觉得自己终于能吃上一碗带肉的康师傅牛肉面。正所谓侠之大者，为国背锅，这一年股指期货经历了不平凡的跌宕起伏。上半年市场一片开朗，10000点不是梦，期指也由原来的 'IF（沪深300）'一个品种增加到了‘IH（上证50）’，‘IF（沪深300）’，‘IC（中证500）’三个品种。“放开让孩儿们玩吧”，不愿透露姓名的领导心里想着，顺势掐灭了手上的烟头。不幸的是，市场下半年被玩坏了，从去杠杆到查做空、从基金经理被约谈到公安部出动、从券商砸奖金到救市基金强势入场，轰轰烈烈的市场拯救活动持续了一两个月，终于把市场给摁了下来。比较不幸的是，量化交易和衍生品受到了比较严重的打击和监管。股指期货的交易方式、保证金比例、交易频率都受到了严厉的限制，并成为众矢之的。 虽不知股指

期货的未来何去何从，但效率的提升才能代表未来的发展方向，投资管理变得更为理性化、程序化、智能化对于专业的投资机构来讲是大势所趋。我们相信期指的严冬期总会过去，我们有理想，我们有愿望，宣誓完毕！

股指期货的才艺

一个品种的股指期货有四份合约，分为当月合约、次月合约、第一季月合约和第二季月合约。作为衍生品，它为二级市场投资带来了丰富的投资方式，并在投资策略中担当着各种各样的角色。基于股指期货可做的事情例如：

- 1、构建alpha市场中性策略；
- 2、期限套利；
- 3、期指间跨期跨品种套利；
- 4、趋势交易；
- 5、高频交易（现在估计比较难）。

这次我们主要研究一种单品种的股指期货趋势交易策略，提供了一种基于市场情绪的择时指标，称之为ITS（Informed Trader Sentiment），参考文献为。

ITS——基于市场情绪的择时指标

中金所每日收盘后会公布股指期货的成交持仓量表，表单中会有三个项目的排名：成交量、持买单量和持卖单量，表单会列出前20名的会员单位。这些会员单位代表了期货市场上最大的机构投资者。我们将同时出现在三个表单排名中的单位视作VIP单位，他们是这个市场的中流砥柱，他们的投资情绪会影响到市场未来的走势，我们需要做的就是找到对的大佬，跟着大佬的情绪飞。那如何才能跟到对的大佬？我们基于这样一个逻辑：如果某位大佬更有预知市场的能力，那么他会在交易中更为坚定的选择某一个买卖方向，它由于买卖带来的成交量就会相对较小，进而（持仓量/成交量）数值相对会大，我们认定他是知情者（对的大佬）；若某个大佬的持仓量与前一位大佬相当但成交量明显偏大，说明这位大佬的交易有更多的不确定性，我们也就认定他不是一个知情者（没有缘分的大佬）。找到对的大佬之后，便开始估测一下大佬的情绪。我们用“对的大佬”们的（持买单量-持卖单量）/（持买单量+持卖单量）（此处的持买单和卖单量都是对的大佬们求和的总量）作为归一化的指标，根据指标是否大于某个阈值Thres来判定大佬对于市场是看多还是看空。那么我们的信号提取过程为：

- step1. 表单中筛选VIP单位（三个排名均上榜）→ 备选大佬；
- step2. 备选大佬中找到（持仓量/成交量）大于Avg的VIP单位 → 对的大佬；
- step3. ITS =（持买单量-持卖单量）/（持买单量+持卖单量）

ITS信号生成器

```
from CAL.PyCAL import *
import copy as cp
import numpy as np
```

```
################################################################
###########################################
#    generate the its signal
################################################################
###########################################
class itsFutSignal:
    def __init__(self,secID,currentDate):
        ####input
        self.secID = secID
        self.currentDate = currentDate
        ####output
        self.sigDate = self.lastDateGet()
        self.contract = self.currContractGet()
        self.posTable = self.posTableGet()
        self.vipDict = self.vipDictGet()
        self.sentiSig = self.sentiSigGet()
        self.inforedInvestor = self.inforedInvestorGet()
        self.itsSig = self.itsSigGet()

    def lastDateGet(self):
        calendar = Calendar('China.SSE')
        date = calendar.advanceDate(self.currentDate,'-'+str(1)+
'B').toDateTime()
        return date

    def currContractGet(self):
        future = DataAPI.MktMFutdGet(mainCon=u"1",contractMark=u
"",contractObject=u"",tradeDate=self.sigDate,startDate=u"",endDa
te=u"",field=u"",pandas="1")
        for index in future.ticker:
            if index[:2] == self.secID:
                return future[future.ticker==index].ticker.tolis
t()[0]

    def posTableGet(self):
        try:
            pos = DataAPI.MktFutMLRGet(secID=u"",ticker=self.con
tract,beginDate=self.sigDate,endDate=self.sigDate,field=u"",pand
as="1")
            neg = DataAPI.MktFutMSRGet(secID=u"",ticker=self.con
tract,beginDate=self.sigDate,endDate=self.sigDate,field=u"",pand
as="1")
            vol = DataAPI.MktFutMTRGet(secID=u"",ticker=self.con
tract,beginDate=self.sigDate,endDate=self.sigDate,field=u"",pand
as="1")
            return {'pos':pos,'neg':neg,'vol':vol}
        except:
            calendar = Calendar('China.SSE')
            self.sigDate = calendar.advanceDate(self.sigDate,'-'
+str(1)+'B').toDateTime()
            self.contract = self.currContractGet()
            return self.posTableGet()

    def list2Dict(self,list):
        keys = list[0]
        values = list[1]
        resultDict = {}
        for index in range(len(keys)):
            resultDict[keys[index]] = values[index]
        return resultDict

    def vipDictGet(self):
        ####get data
        longDict = self.list2Dict([self.posTable['pos'].partySho
rtName.tolist(),self.posTable['pos'].longVol])
        shortDict = self.list2Dict([self.posTable['neg'].partySh
ortName.tolist(),self.posTable['neg'].shortVol])
        volDict = self.list2Dict([self.posTable['vol'].partyShor
tName.tolist(),self.posTable['vol'].turnoverVol])

        ####get vip list
        vipList = []
        for index in longDict.keys():
            if index in shortDict.keys():
                if index in volDict.keys():
                    vipList.append(index)
        ####get vip dict
        vipDict = {}
        for index in vipList:
            vipDict[index] = [longDict[index],shortDict[index],v
olDict[index]]

        return vipDict

    def sentiSigGet(self):
        sentiSig = {}
        for index in self.vipDict:
            sentiSig[index] = sum(self.vipDict[index][:2])*1.0/s
elf.vipDict[index][-1]
        return sentiSig

    def inforedInvestorGet(self):
        if len(self.sentiSig) != 0:
            sentiAvg = sum(self.sentiSig.values())/len(self.sent
iSig)
            inforedInvestor = [index for index in self.sentiSig
if self.sentiSig[index] > sentiAvg]
            return inforedInvestor
        else:
            sentiAvg = 0
            return []

    def itsSigGet(self):
        totalBuy = 0
        totalSell = 0
        if len(self.inforedInvestor) != 0:
            for index in self.inforedInvestor:
                totalBuy += self.vipDict[index][0]
                totalSell += self.vipDict[index][1]
            if totalBuy + totalSell != 0:
                return 1.0*(totalBuy - totalSell)/(totalBuy + to
talSell)
            else:
                return 'null'
        else:
            return 'null'
```

ITS趋势交易信号

ITS信号代表了对的大佬们的情绪，那么如何利用它给出每日的交易信号呢？我们利用最为直观有效的阈值策略，设定某阈值 `Thres` ：

1） `ITS > Thres` ：大佬看多，买买买，交易信号为1；

2） `ITS < Thres` ：大佬看空，卖卖卖，交易信号为-1；

3） `ITS = Thres & DataErr` ：形势不明朗或找不到大佬，停止交易观望，交易信号为0

此处我们取 `Thres = -0.12` ？Why？其实是这样的，由于期现套利交易的存在，因此期指本身有一部分的空单是由于期限套利造成的，由于这部分资金通常会留存较久，因此通常情况下期指的持仓总量应该是空单偏多，而我们判断市场情绪的时候要把这部分期货市场上的“裸空单”给剔除掉，因此Thres应该设置为负。

IF趋势交易测试

由于目前Strategy部分还不好支持期指的交易回测，因此用脚本生成了测试数据。

1）交易标的：沪深300主力合约。用IF来测试的原因很简单，样本数多呀！

2）每日的交易信号：根据前一日收盘后的持仓量表单计算ITS后根据阈值产生；

3）每日收益率：我们假定在获取当日的信号后，在开盘的一段时间内以某个价格买入期指，持有至临近收盘后以某个价格卖出，做日内交易。那么买卖价如何界定？有三种方式来计算：①昨收-今收；②今开-今收；③昨结算-今结算。①和②都是时点价格，而③是均价。①必然不合理因为无法在昨日收盘前得到今日的交易信号，不具有可操作性；②是时点价格，可操作性也不强；对③来讲，由于结算价是一段时间的均价，我们认为拿这个均价作为买卖的期望价格是合理的。所以每日收益率的计算方式是③；

```
startDate = '20110101'
endDate = '20150801'
```

```
future = DataAPI.MktMFutdGet(mainCon=u"1",contractMark=u"",contractObject=u"",tradeDate=u'',startDate=startDate,endDate=endDate,field=u"",pandas="1")

# print future
closePrice = []
openPrice = []
preClosePrice = []
settlePrice = []
preSettlePrice = []
tradeDate = []
ticker = []
for index in future.ticker.tolist():
    if index[:2] == 'IF':
        if index not in ticker:
            ticker.append(index)

for index in ticker:
    closePrice += future[future.ticker==index]['closePrice'].tolist()
    openPrice += future[future.ticker==index]['openPrice'].tolist()
    preClosePrice += future[future.ticker==index]['preClosePrice'].tolist()
    settlePrice += future[future.ticker==index]['settlePrice'].tolist()
    preSettlePrice += future[future.ticker==index]['preSettlePrice'].tolist()
    tradeDate += future[future.ticker==index]['tradeDate'].tolist()

closePrice = np.array(closePrice)
openPrice = np.array(openPrice)
preClosePrice = np.array(preClosePrice)
settlePrice = np.array(settlePrice)
preSettlePrice = np.array(preSettlePrice)


closeRetRate = (closePrice - preClosePrice)/preClosePrice
settleRetRate = (settlePrice - preSettlePrice)/preSettlePrice
clopenRetRate = (closePrice - openPrice)/openPrice
tradeDate = tradeDate


itsValue = [itsFutSignal('IF',index).itsSig for index in tradeDate]
itsSignal = []
thres  = -0.12
for index in itsValue:
    if index != 'null':
        if index > thres:
            itsSignal.append(1)
        else:
            itsSignal.append(-1)
    else:
        itsSignal.append(0)


itsSignal = np.array(itsSignal)

benchMark = DataAPI.MktIdxdGet(tradeDate=u"",indexID=u"",ticker=
u"000300",beginDate=startDate,endDate=endDate,field=u"closeIndex"
,pandas="1")['closeIndex'].tolist()
benchMark = benchMark/benchMark[0]
```

回测展示

下面为回测结果展示，测算细节如下:

1) 每日收益率根据结算价来计算，前结算价作为买入的参考均价，结算价作为卖出的参考均价；

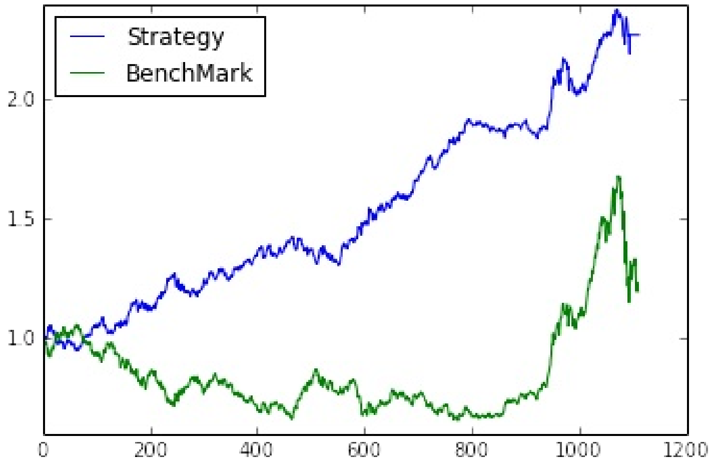
2）收益率曲线计算采用单利计算；

3）无风险利率取5%；

4）最后一小段曲线为平是由于股指期货受到限制导致交易停止

```
import matplotlib.pyplot as plt

####calculate the daily return
dailyRet = settleRetRate*itsSignal


####calculate the winRate
count = 0
denom = 0
for index in dailyRet:
    if index > 0:
        count += 1
    if index != 0:
        denom += 1
print '策略胜率 : ' + str(round((count*1.0/denom)*100,2)) + '%'


####calculate the curve
curve = [1]
position = 0.8
for index in dailyRet:
    curve.append(curve[-1] + index*position)
```

```
####calculate the location
print '策略仓位 : ' + str(position)

####calculate the max drawDown
maxDrawDown = []
for index in range(len(curve)):
    tmp = [ele/curve[index] for ele in curve[index:]]
    maxDrawDown.append(min(tmp))
print '最大回撤 : ' + str(round((1-min(maxDrawDown))*100,2)) + '%'



####calculate the sharpRatio
stDate = Date(int(startDate[:4]),int(startDate[4:6]),int(startDa
te[6:8]))
edDate = Date(int(endDate[:4]),int(endDate[4:6]),int(endDate[6:8
]))
duration = round((edDate-stDate)/365.0,1)

retYearly = curve[-1]/duration
interest = 0.05
fluctuation = np.std(curve)/np.sqrt(duration)

print '年化收益 : ' + str(round(retYearly,2)*100.0) + '%'
print '夏普比率 : ' + str(round((retYearly-interest)/fluctuation,2
))


####plot the figure
print '\n'
plt.plot(curve)
plt.plot(benchMark)
plt.legend(['Strategy','BenchMark'],0)

策略胜率 : 55.03%
策略仓位  :  0.8
最大回撤 : 10.06%
年化收益 : 49.0%
夏普比率 : 2.44

<matplotlib.legend.Legend at 0xed4cc50>
```



被曲线美哭了，不叨叨了，Merry Xmas！