法律声明

本课件包括：演示文稿，示例，代码，题库，视频和声音等，小象学院拥有完全知识产权的权利；只限于善意学习者在本课程使用，不得在课程范围外向任何第三方散播。任何其他人或机构不得盗版、复制、仿造其中的创意，我们将保留一切通过法律手段追究违反者的权利。

关注 小象学院

《进阶！量化交易实战：迭代式的量化策略研发》第3期

第8课：均值回复型策略的设计与开发

主　讲：宋战江

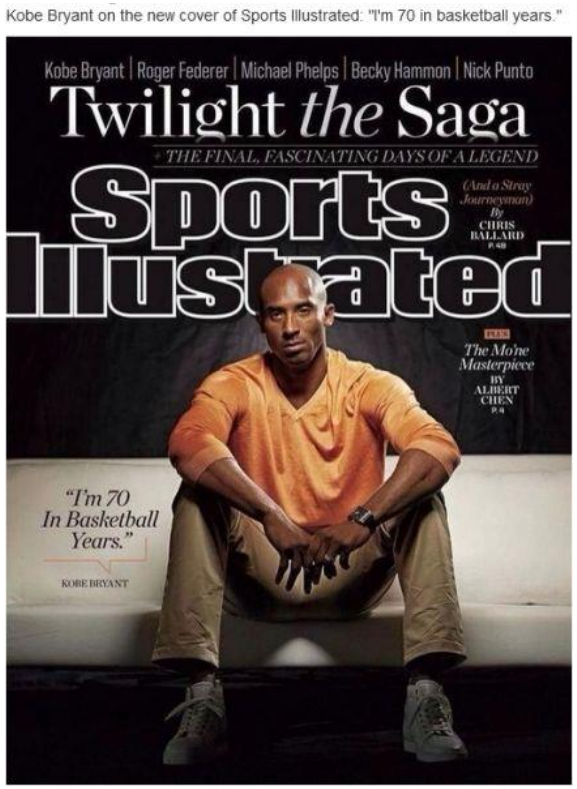
内容介绍

- 均值回复型策略原理
- 编写一个均值回复型股票量化策略
- 编写多空组合策略进行策略改进
- 策略对交易成本的敏感范围检测
- 在策略中实现头寸管理进行策略优化

天下大势 分久必合 合久必分

均值回复型策略原理

《体育画报》的厄运



一个运动员的照片如果出现在杂志封面，那他注定要在接下来的赛季中表现不佳。

- 丹尼尔·卡尼曼 2011

一个运动员的表现可以被认为是围绕均值随机分布的

理查德·塞勒的基金的理念

At the individual stock level, we search for events that suggest investor misbehavior.

Investors Make Mistakes. We Look For Them.

There are two kinds of mistakes that produce buying opportunities: over-reaction and under-reaction.

Other investors may over-react to bad news and losses (e.g., panic). Or they may under-react to good news (e.g., not pay attention).

Over-reaction
...to historical, negative information

Under-reaction
...to new, positive information

行为金融学指导下的交易策略

- Seeks to capitalize on behavioral biases that may cause the market to
  over-react to old, negative information or
  under-react to new, positive information.

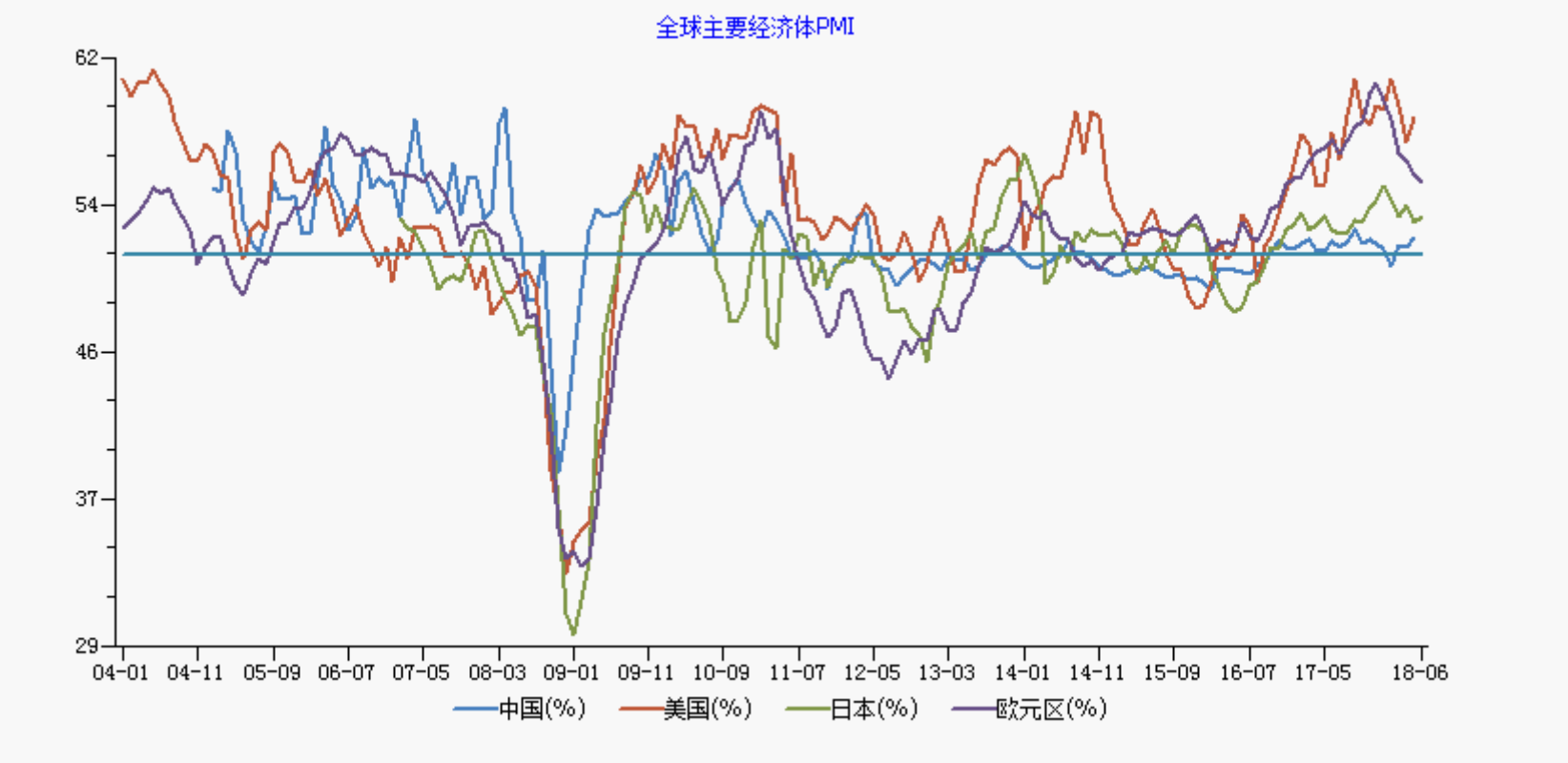
- Looks for companies with one or both of:
  - significant insider buying or stock repurchases (over-reaction)
  - large earnings surprises (under-reaction)

巴菲特的投资密码

- 别人贪婪时我恐惧，
  别人恐惧时我贪婪

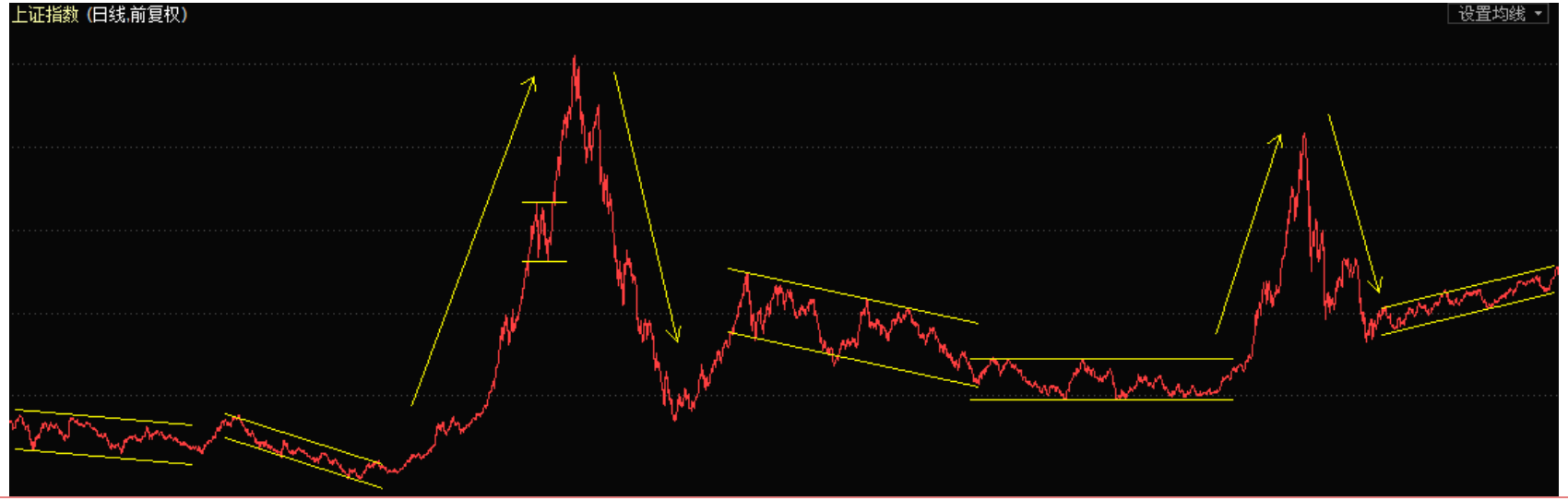
- 用平常的价钱买一家
  很棒的公司远远强过
  用很棒的价钱买一家
  平常的公司

经济规律或周期


全球主要经济体PMI
62
54
46
37
29
04-01
04-11
05-09
06-07
07-05
08-03
09-01
09-11
10-09
11-07
12-05
13-03
14-01
14-11
15-09
16-07
17-05
18-06
中国(%)
美国(%)
日本(%)
欧元区(%)

做趋势 vs. 做震荡

- 大尺度：趋势+反转
- 中尺度：趋势跟随 + 均值回复
- 小尺度：均值回复
- 微小尺度：随机游走



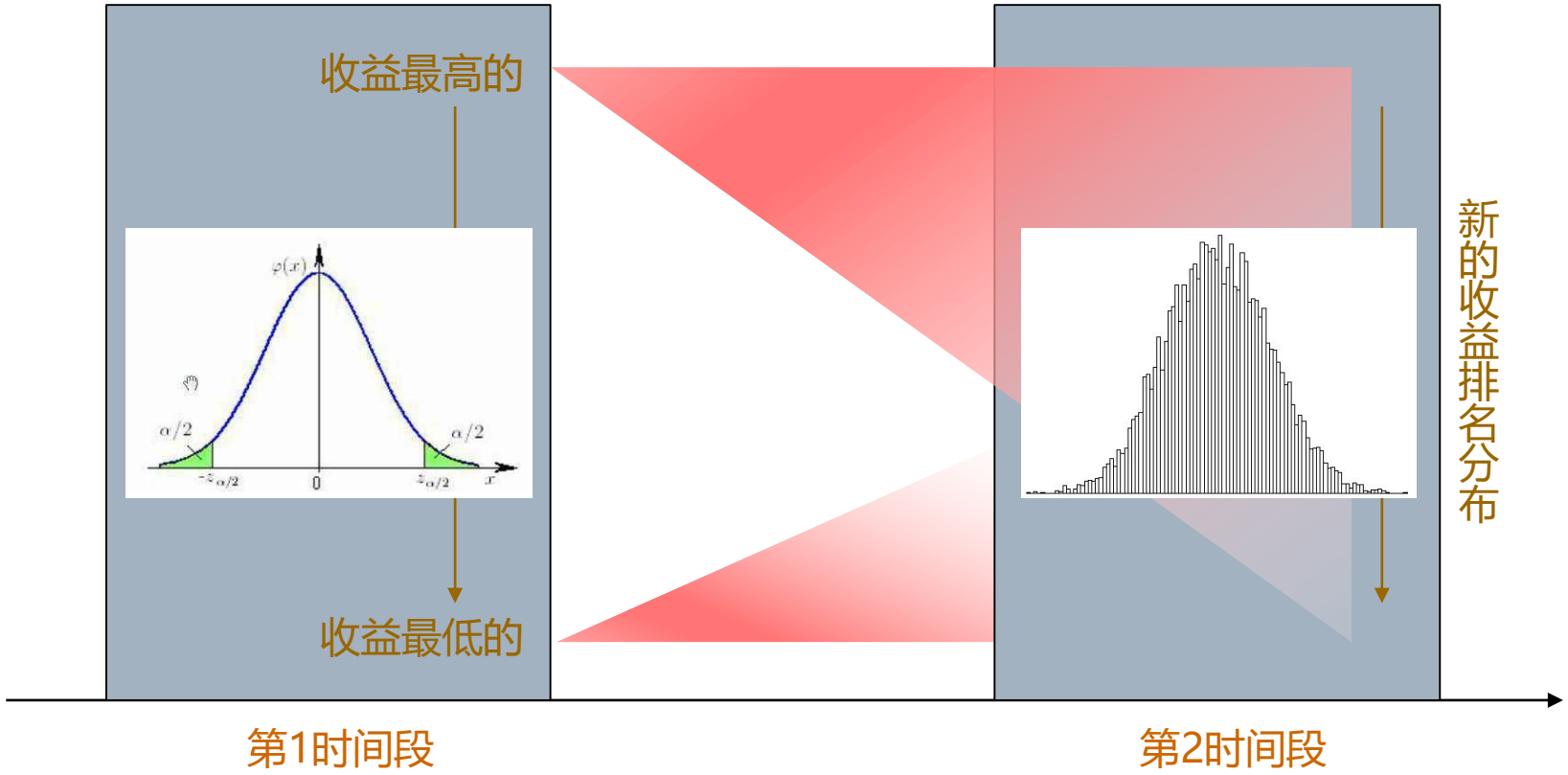
A股市场有没有稳定的均值回复效应？

Song's Hypothesis

- 构造两个组合：

	组合1	组合2
成分股选取方法	按收益率，Top-10% @ 时段1	按收益率，Bottom-10% @ 时段1
时段1(收益排名)	{1, 2, 3, …, 300} Average: 0.05	{2701, 2702, …, 3000} Average: 0.95
时段2(对应的收益排名)	e.g.: {2, 3, 7, 10, …, 2998}	e.g.: {4, 12, 13, 21, …, 2995}
检验指标	Average: ~0.5?	Average: ~0.5?

如何检验该假设？

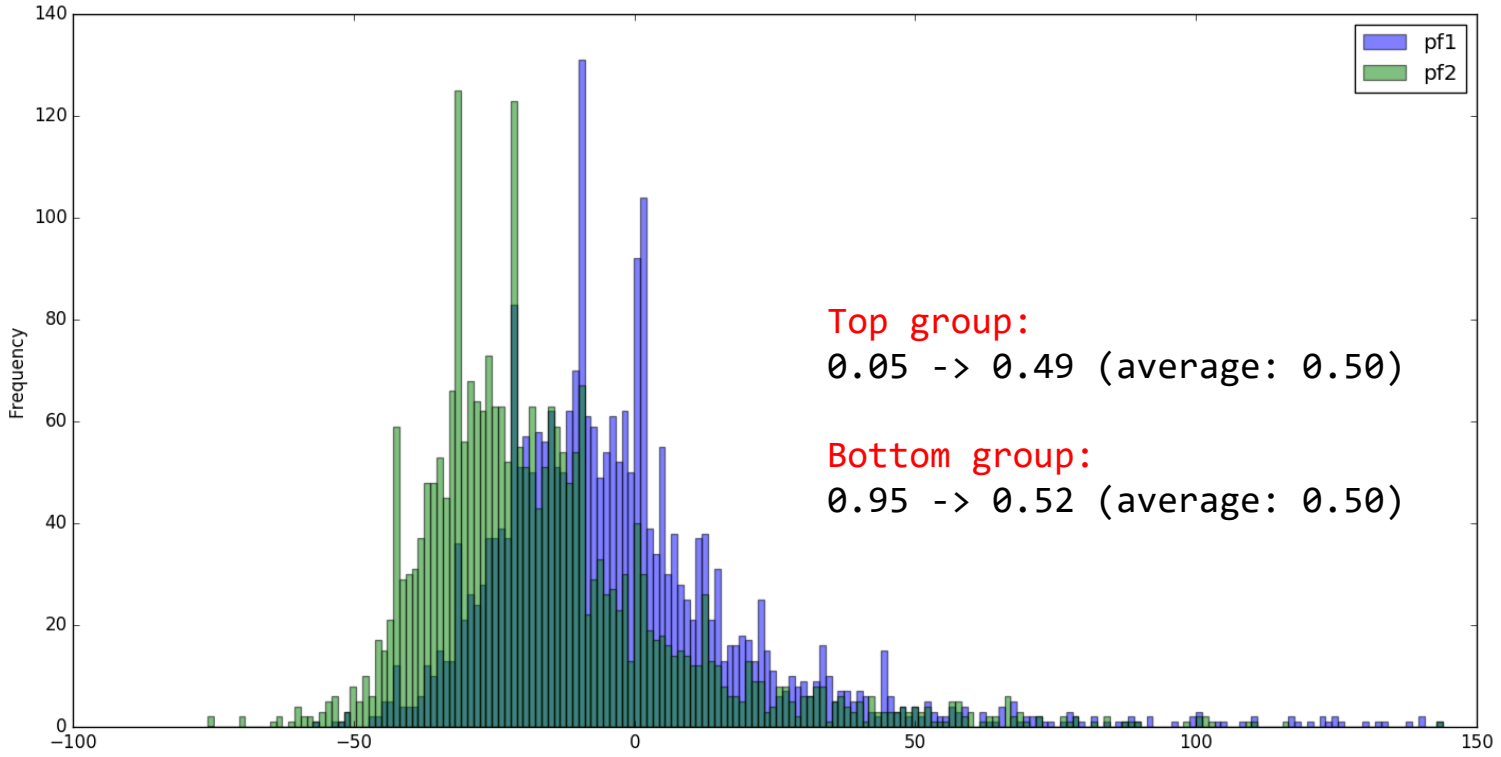


```
# -*- coding: utf-8 -*-

import tushare as ts
import pandas as pd
import matplotlib.pyplot as plt

start_date = "2016-01-01"
end_date = "2018-05-31"
obs_percent = 0.3  # 时间分3段，用首尾两个时间段来比较收益排名均值的变化
cmp_percent = 0.1  # 取收益最高的和最低的各10%，计算收益排名的均值

stock_list = ts.get_stock_basics()["name"]  # 全市场股票信息列表
pf_list = []

cnt = 0
for code, name in stock_list.iteritems():
    print(cnt, code, name)
    if name.find("ST") != -1:  # 排除ST和*ST股票
        continue
    df = ts.get_k_data(code, start=start_date, end=end_date, ktype="D",
                       autype="qfq")  # 读日K行情数据
    if len(df) < 400:  # 如果有较长的停牌时间，则忽略它
        continue
    df.index = df.pop("date")
    win_size = int(len(df) * obs_percent)  # 用于计算收益率的时间窗口宽度
```

```
        # 第1段时间内的收益率
        profit_1 = int((df["close"][win_size-1]/df["close"][0] - 1.0) * 100)
        # 第2段时间内的收益率
        profit_2 = int((df["close"][-1]/df["close"][-win_size] - 1.0) * 100)

        # 为便于观察分布图，忽略一些特别大或特别小的值
        if profit_1 > 150 or profit_2 > 150:
            continue

        # 保存每只股票的2个收益率的值
        pf_list.append(dict(code=code, pf1=profit_1, pf2=profit_2))

        cnt += 1

    # 先按第1段时间内的收益率排名（从高往低）
    pf_list.sort(key=lambda x: x["pf1"], reverse=True)
    for idx, doc in enumerate(pf_list):
        doc["rank_1"] = idx  # 记录下每只个股在第1段时间内的收益率排名

    num_cands = int(len(pf_list)*cmp_percent)  # 用于统计的最高（和最低）收益的股票个数
    top_cands = pf_list[:num_cands]   # 第1段时间内最高收益的股票信息列表
    btm_cands = pf_list[-num_cands:]  # 第1段时间内最低收益的股票信息列表
    top_group_codes = set([doc["code"] for doc in top_cands])  # 最高收益股票代码集合
    btm_group_codes = set([doc["code"] for doc in btm_cands])  # 最低收益股票代码集合
```

```
avg_top_idx = round(sum([doc["rank_1"] for doc in top_cands])
                    /num_cands/len(pf_list), 2)  # 最高收益股票排名均值
avg_btm_idx = round(sum([doc["rank_1"] for doc in btm_cands])
                    /num_cands/len(pf_list), 2)  # 最低收益股票排名均值
# 所有股票收益排名的均值，等差数列均值（中值），直接计算即可
avg_mid_idx = (0 + len(pf_list)-1) / 2 / len(pf_list)

# 再按第2段时间内的收益率排名（从高往低）
pf_list.sort(key=lambda x: x["pf2"], reverse=True)
for idx, doc in enumerate(pf_list):
    doc["rank_2"] = idx  # 记录下每只个股在第2段时间内的收益率排名

cmp_avg_top_idx = round(sum([doc["rank_2"] for doc in pf_list if doc["code"]
                             in top_group_codes])/num_cands/len(pf_list), 2)
cmp_avg_btm_idx = round(sum([doc["rank_2"] for doc in pf_list if doc["code"]
                             in btm_group_codes])/num_cands/len(pf_list), 2)

print("Top group:", avg_top_idx, "->", cmp_avg_top_idx,
      "(avg: %.2f)" % avg_mid_idx)
print("Btm group:", avg_btm_idx, "->", cmp_avg_btm_idx,
      "(avg: %.2f)" % avg_mid_idx)

profits = pd.DataFrame({"pf1": [doc["pf1"] for doc in pf_list],
                        "pf2": [doc["pf2"] for doc in pf_list]})
profits.plot.hist(bins=200, alpha=0.5)  # 绘制两段时间内全市场收益率分布的直方图
plt.show()
```

收益排名分布变化的启示



思考

- 时段1和时段2间隔时间长度的影响
  - 对应：均值回复的尺度
- 不同起始点的影响
  - 对应：在什么样的大趋势下有此特征

思路 – 短期市场反转

建立一个股票池：

过去3（或1）个月表现最差的N只股票构成的组合

再平衡周期：
- 1个月

头寸管理：
- 所有入选股票均仓
- 按照市值加权

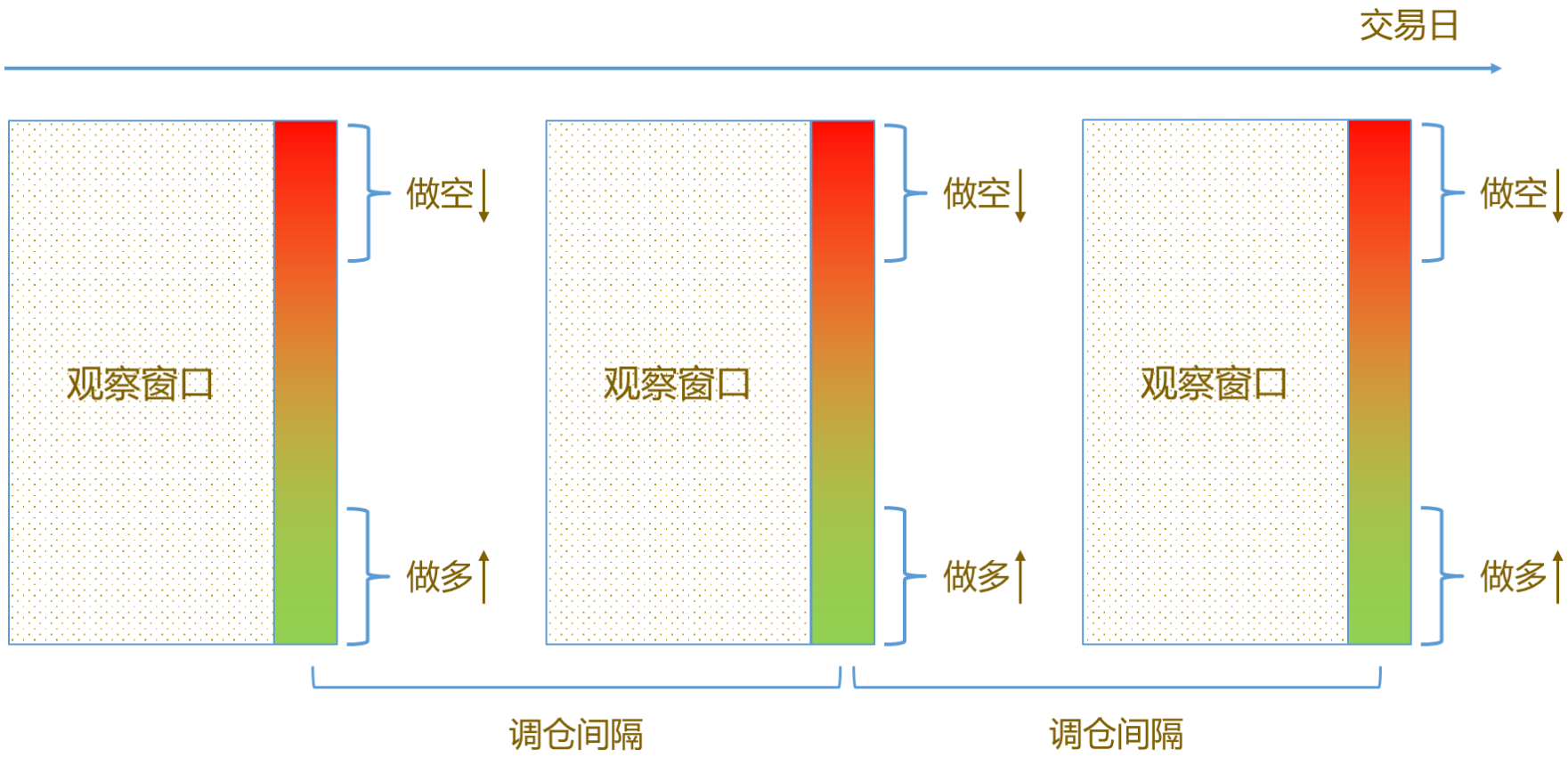
多空组合：

做多表现最差组合

同时

做空表现最好组合

均值回复策略示意图



从思想到实践 逐步求精的过程

在聚宽上编写一个均值回复型股票量化策略

策略编写的考虑

- 投资标的选择
  - 投资组合的容量
  - 期货 VS. 股票

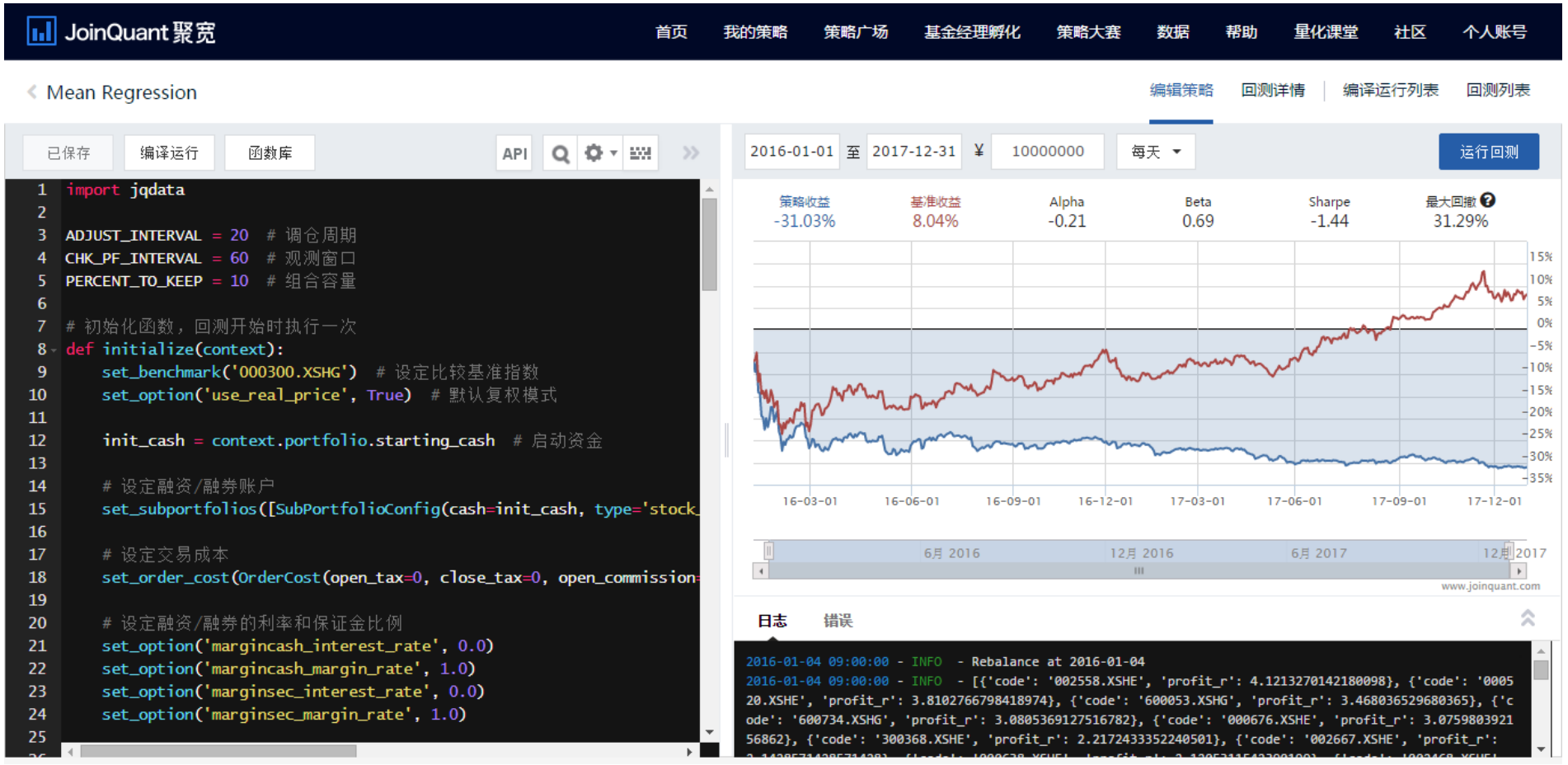
A股

融资/融券

Python

- 多空双向交易
  - 纯多头
  - 做多/做空机制

编程平台工作界面1


JoinQuant 聚宽
首页
我的策略
策略广场
基金经理孵化
策略大赛
数据
帮助
量化课堂
社区
个人账号
Mean Regression
编辑策略
回测详情
编译运行列表
回测列表
已保存
编译运行
函数库
API
2016-01-01
至
2017-12-31
¥
10000000
每天
运行回测
策略收益
-31.03%
基准收益
8.04%
Alpha
-0.21
Beta
0.69
Sharpe
-1.44
最大回撤
31.29%
import jqdata
ADJUST_INTERVAL = 20 # 调仓周期
CHK_PF_INTERVAL = 60 # 观测窗口
PERCENT_TO_KEEP = 10 # 组合容量
# 初始化函数，回测开始时执行一次
def initialize(context):
set_benchmark('000300.XSHG') # 设定比较基准指数
set_option('use_real_price', True) # 默认复权模式
init_cash = context.portfolio.starting_cash # 启动资金
# 设定融资/融券账户
set_subportfolios([SubPortfolioConfig(cash=init_cash, type='stock
设定交易成本
set_order_cost(OrderCost(open_tax=0, close_tax=0, open_commission=
# 设定融资/融券的利率和保证金比例
set_option('margincash_interest_rate', 0.0)
set_option('margincash_margin_rate', 1.0)
set_option('marginsec_interest_rate', 0.0)
set_option('marginsec_margin_rate', 1.0)
www.joinquant.com
日志
错误
2016-01-04 09:00:00 - INFO - Rebalance at 2016-01-04
2016-01-04 09:00:00 - INFO - [{'code': '002558.XSHE', 'profit_r': 4.1213270142180098}, {'code': '0005
20.XSHE', 'profit_r': 3.8102766798418974}, {'code': '600053.XSHG', 'profit_r': 3.468036529680365}, {'c
ode': '600734.XSHG', 'profit_r': 3.0805369127516782}, {'code': '000676.XSHE', 'profit_r': 3.0759803921
56862}, {'code': '300368.XSHE', 'profit_r': 2.2172433352240501}, {'code': '002667.XSHE', 'profit_r':

编程平台工作界面2


JoinQuant 聚宽
首页
我的策略
策略广场
基金经理孵化
策略大赛
数据
帮助
量化课堂
社区
个人账号
Mean Regression
编辑策略
回测详情
编译运行列表
回测列表
设置： 2016-01-01 到 2017-12-31， ¥10000000， 每天
状态：
49.2%
取 消
模拟交易
归因分析
分享到社区
查看代码
导出
收益概述
交易详情
每日持仓&收益
日志输出
性能分析
策略收益
基准收益
Alpha
Beta
Sharpe
Sortino
收益概述
策略收益
策略年化收益
基准收益
Alpha
Beta
Sharpe
胜率
盈亏比
最大回撤
其他指标
缩放：
1个月
1年
全部
策略收益
基准收益
超额收益
普通轴
对数轴
超额收益
时间：
2016-01-04
2017-12-29
-2.5%
-5%
-7.5%
-10%
-12.5%
-15%
-17.5%
-20%
-22.5%
-25%
-27.5%
收益
0k
-500k

初步尝试

- 纯多头方案
  - 定时调仓：卖掉旧持仓，构建新组合
  - 只做多前期观察窗口内表现最差的集合
  - 为入选组合的个股等额分配资金

- 作为后续的比较基准

环境准备

- 登录聚宽 => 我的策略 => 策略列表
- 新建策略 => 融资融券 => 重命名策略
- 回测区间：2016-01-01 ~ 2017-12-31
- 初始资金：10,000,000，回测周期：每天

初始化参数和变量

```
1  import jqdata
2
3  ADJUST_INTERVAL = 20  # 调仓周期
4  CHK_PF_INTERVAL = 60  # 观测窗口
5  PERCENT_TO_KEEP = 10  # 组合容量
6
7  # 初始化函数，回测开始时执行一次
8  def initialize(context):
9      set_benchmark('000300.XSHG')  # 设定比较基准指数
10     set_option('use_real_price', True)  # 默认复权模式
11
12     init_cash = context.portfolio.starting_cash  # 启动资金
13
14     # 设定融资/融券账户
15     set_subportfolios([SubPortfolioConfig(cash=init_cash, type='stock_margin')])
16
17     # 设定交易成本
18     set_order_cost(OrderCost(open_tax=0, close_tax=0, open_commission=0,
           close_commission=0, close_today_commission=0, min_commission=0), type='stock')
19
20     # 设定融资/融券的利率和保证金比例
21     set_option('margincash_interest_rate', 0.0)
22     set_option('margincash_margin_rate', 1.0)
23     set_option('marginsec_interest_rate', 0.0)
24     set_option('marginsec_margin_rate', 1.0)
25
```

```
26      # 设置定时运行函数
27      run_daily(before_market_open, time='before_open', reference_security='000300.XSHG')
28      run_daily(market_open, time='open', reference_security='000300.XSHG')
29      run_daily(after_market_close, time='after_close', reference_security='000300.XSHG')
30
31      # 记录交易日期及其索引
32      g.all_tdays = jqdata.get_all_trade_days()
33      g.date_xlat = dict([(doc[1], doc[0]) for doc in enumerate(g.all_tdays)])
34
35      # 设定若干全局变量
36      g.elapsed_days = 0
37      g.rebalance_today = False
38      g.new_long_pos = {}
39      g.new_short_pos = {}
40
```

用于快速定位交易日期的数据结构：

2018-05-07	2018-05-08	2018-05-09	2018-05-10	2018-05-11	2018-05-14	2018-05-15	2018-05-16	...
143	144	145	146	147	148	149	150	...

```
# 导入函数库
import jqdata

ADJUST_INTERVAL = 20  # 调仓周期
CHK_PF_INTERVAL = 60  # 观测窗口
PERCENT_TO_KEEP = 10  # 组合容量

# 初始化函数，回测开始时执行一次
def initialize(context):
    set_benchmark('000300.XSHG')  # 设定比较基准指数
    set_option('use_real_price', True)  # 默认复权模式

    # 设定融资/融券账户
    set_subportfolios([SubPortfolioConfig(cash=context.portfolio.cash,
                      type='stock_margin')])

    # 设定交易成本
    set_order_cost(OrderCost(open_tax=0, close_tax=0.000, open_commission=0.0000,
                   close_commission=0.0000, close_today_commission=0,
                   min_commission=0), type='stock')

    # 设定融资/融券的利率和保证金比例
    set_option('margincash_interest_rate', 0.00)
    set_option('margincash_margin_rate', 1.0)
    set_option('marginsec_interest_rate', 0.00)
    set_option('marginsec_margin_rate', 1.0)
```

```
    # 设置定时运行函数
    run_daily(before_market_open, time='before_open', reference_security='000300.XSHG')
    run_daily(market_open, time='open', reference_security='000300.XSHG')
    run_daily(after_market_close, time='after_close', reference_security='000300.XSHG')

    # 记录交易日期及其索引
    g.all_tdays = jqdata.get_all_trade_days()
    g.date_xlat = dict([(doc[1], doc[0]) for doc in enumerate(g.all_tdays)])

    # 设定若干全局变量
    g.elapsed_days = 0
    g.rebalance_today = False
    g.new_long_pos = {}

# 在每个交易日开盘前运行
def before_market_open(context):
    cur_dt = context.current_dt.date()  # 当前日期
    if g.elapsed_days % ADJUST_INTERVAL == 0:  # 是否是调仓日
        log.info("Rebalance at %s", cur_dt)
        g.rebalance_today = True
        idx = g.date_xlat[cur_dt]

        # 观察窗口首末日期
        tail_date = g.all_tdays[idx-1]
        head_date = g.all_tdays[idx-1-CHK_PF_INTERVAL]
```

```
# 调仓日前一天的股票列表
candidates = list(get_all_securities(["stock"], tail_date).index)

# 观察窗口末尾日期的股票价格
tail_prices = get_price(candidates, tail_date, tail_date, frequency="1d",
                        fields=["close"])
g.tail_values = dict(tail_prices["close"].iloc[0])

# 观察窗口起始日期的股票价格
head_prices = get_price(candidates, head_date, head_date, frequency="1d",
                        fields=["close"])
g.head_values = dict(head_prices["close"].iloc[0])

# 在观察窗口内按收益率对股票排序（从小到大）
merged_list = []
for code, tail_value in g.tail_values.items():
    if math.isnan(tail_value):
        continue
    if code not in g.head_values:
        continue
    head_value = g.head_values[code]
    if math.isnan(head_value):
        continue

    # 计算相对收益率
    profit_r = tail_value/head_value - 1.0
```

```
            # 保存中间结果
            merged_list.append({
                "code": code,
                "profit_r": profit_r,
                "price": tail_value
            })

        # 从首尾分别取出一定比例的股票，构造新的多空投资组合
        merged_list.sort(key=lambda x: x["profit_r"], reverse=False)

        num_to_keep = len(merged_list) * PERCENT_TO_KEEP // 100
        g.new_long_pos  = {doc["code"]:doc for doc in merged_list[0:num_to_keep]}

    else:
        g.rebalance_today = False

    g.elapsed_days += 1

# 在每个交易日开盘时运行
def market_open(context):
    p = context.portfolio.subportfolios[0]  # 融资/融券保证金账户
    if g.rebalance_today:
        # 再平衡步骤1：平掉原有多空仓位
        prev_long_pos = p.long_positions

        for code, pos in prev_long_pos.items():
            margincash_close(code, pos.closeable_amount)
```

```
            # 再平衡步骤2：开立新的多空仓位
            each_long_cash = round(p.available_margin / len(g.new_long_pos), 2)

            for code, doc in g.new_long_pos.items():
                num_to_buy = each_long_cash / g.tail_values[code] // 100 * 100
                margincash_open(code, num_to_buy)

        else:
            # 非调仓日，可以增加止盈/止损等额外操作……
            pass

# 在每个交易日收盘后运行
def after_market_close(context):
    # 查看融资融券账户相关相关信息(更多请见API-对象-SubPortfolio)
    p = context.portfolio.subportfolios[0]
    log.info('- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -')
    log.info('查看融资融券账户相关相关信息(更多请见API-对象-SubPortfolio)：')
    log.info('总资产：', p.total_value)
    log.info('净资产：', p.net_value)
    log.info('总负债：', p.total_liability)
    log.info('融资负债：', p.cash_liability)
    log.info('融券负债：', p.sec_liability)
    log.info('利息总负债：', p.interest)
    log.info('可用保证金：', p.available_margin)
    log.info('维持担保比例：', p.maintenance_margin_rate)
    log.info('账户所属类型：', p.type)
    log.info('###############################################################')
```

回测结果

- 仅做多表现最差的：

思考

- 刚才的结果是仅做多表现最差的，
- 如果仅做多表现最好的，又如何？

```
# 从首尾分别取出一定比例的股票，构造新的多空投资组合
merged_list.sort(key=lambda x: x["profit_r"], reverse=False)



# 从首尾分别取出一定比例的股票，构造新的多空投资组合
merged_list.sort(key=lambda x: x["profit_r"], reverse=True)

num_to_keep = len(merged_list) * PERCENT_TO_KEEP // 100
g.new_long_pos  = {doc["code"]:doc for doc in merged_list[0:num_to_keep]}
```

回测结果

- 仅做多表现最好的：（反向指标）

为什么不直接做空？

从“纯多头”到“多空双向”

编写多空组合策略进行策略改进

纯多头 vs. 多空双向

- 代码的增量修改部分：

```
g.new_long_pos = {}
g.new_short_pos = {}

num_to_keep = len(merged_list) * PERCENT_TO_KEEP / 100 // 2

g.new_long_pos  = {doc["code"]:doc for doc in merged_list[0:num_to_keep]}
g.new_short_pos = {doc["code"]:doc for doc in merged_list[-num_to_keep:]}

prev_long_pos = p.long_positions
prev_short_pos = p.short_positions
```

```
for code, pos in prev_long_pos.items():
    margincash_close(code, pos.closeable_amount)

for code, pos in prev_short_pos.items():
    marginsec_close(code, pos.closeable_amount)


each_long_cash = round(p.available_margin * 0.5 / len(g.new_long_pos), 2)
each_short_cash = round(p.available_margin * 0.5 / len(g.new_short_pos), 2)


for code, doc in g.new_long_pos.items():
    num_to_buy = each_long_cash / g.tail_values[code] // 100 * 100
    margincash_open(code, num_to_buy)

for code, doc in g.new_short_pos.items():
    num_to_sellshort = each_short_cash / g.tail_values[code] // 100 * 100
    marginsec_open(code, num_to_sellshort)
```

回测结果

- 多空双向：震荡市 vs. HS300

回测结果

- 多空双向：震荡市 vs. 上证指数

回测结果

- 多空双向：牛熊市 vs. HS300

融券卖出是保证金比例好像起不到作用

华鱼辛雨 发布于 3天前 53 4 0

导入函数库

import jqdata

初始化函数，设定基准等等

def initialize(context):

```
# 设定沪深300作为基准
set_benchmark('510300.XSHG')
# 只设定一个账号
# 获取初始资金
init_cash = context.portfolio.starting_cash
# 设置账户为融资融券账户
set_subportfolios([SubPortfolioConfig(cash=init_cash, type='s
# 设置融资利率
set_option('margincash_interest_rate', 0.02)
# 设置融券利率
set_option('marginsec_interest_rate', 0.02)
# 设置融资保证金比例
set_option('margincash_margin_rate', 1)
# 设置融券保证金比例
set_option('marginsec_margin_rate', 1)
```

Supercritical-JoinQuant聚宽

@华鱼辛雨

收到您的问题，我们核查下！

发布于 2018-06-29 16:42:38 回复

华鱼辛雨

@Supercritical-JoinQuant聚宽 怎么样找到问题所在了没

发布于 2018-07-02 10:40:09 回复

Supercritical-JoinQuant聚宽

@华鱼辛雨

我们正在处理中，有结果答复您，请耐心等候。

发布于 2018-07-02 11:33:39 回复

https://www.joinquant.com/post/13391?tag=algorithm

使用第三方平台遇到问题怎么办？

- 当效果不符合预期时
  - 从自身找原因
  - 看平台有没有异常
  - 用其它平台交叉验证
- 当效果特别好的时候
  - 同样参考上面三条
- Actions
  - 向平台反映情况，等待官方解决
  - 换个平台试试，或者自建平台

休息一下

5分钟后回来

滑点和成本 绝对不可忽视的因素

策略对交易成本的敏感范围检测

压力测试 - 考虑各种成本

- 各类交易费用
  - 交易所手续费
  - 券商佣金
- 保证金账户设置
  - 融资/融券利率
  - 保证金比率
- 滑点；冲击成本

保证金分配比例的计算：

	交易额	保证金
多头（融资）	X	1 * X
空头（融券）	X	1.5 * X

1 * X + 1.5 * X = 1.0 (100%)

多空资金分别占比：X = 0.4 (40%)

set_slippage(...) # 固定值 / 百分比

```
17      # 设定交易成本
18      # set_order_cost(OrderCost(open_tax=0, close_tax=0, open_commission=0,
        close_commission=0, close_today_commission=0, min_commission=0), type='stock')
19      set_order_cost(OrderCost(open_tax=0, close_tax=0.001, open_commission=0.0003,
            close_commission=0.0003, close_today_commission=0, min_commission=5), type='stock')
20
21      # 设定融资/融券的利率和保证金比例
22      """
23      set_option('margincash_interest_rate', 0.0)
24      set_option('margincash_margin_rate', 1.0)
25      set_option('marginsec_interest_rate', 0.0)
26      set_option('marginsec_margin_rate', 1.0)
27      """
28      set_option('margincash_interest_rate', 0.0)
29      set_option('margincash_margin_rate', 1.0)
30      set_option('marginsec_interest_rate', 0.10)
31      set_option('marginsec_margin_rate', 1.5)
32
```

```
111         # 再平衡步骤2：开立新的多空仓位
112         """
113         each_long_cash = round(p.available_margin * 0.5 / len(g.new_long_pos), 2)
114         each_short_cash = round(p.available_margin * 0.5 / len(g.new_short_pos), 2)
115         """
116         each_long_cash = round(p.available_margin * 0.4 / len(g.new_long_pos), 2)
117         each_short_cash = round(p.available_margin * 0.4 / len(g.new_short_pos), 2)
118
```

回测结果

- 多空双向，加上交易成本和两融利率

思考

- 与之前的结果相比，年化收益减少5%左右
- 为什么是这个数？尝试给出合理的解释...

还记得交易系统的核心要素吗?

在策略中实现头寸管理进行策略优化

头寸管理

- 分配资金方式：均仓（现有方式） vs. 按波动率
- 波动率增大，意味着风险增加，分配的仓位就小，反之亦然
- 平均真实波幅ATR = MA(TR, N)
  - TR = 当日的真实波动幅度
    = Max（H-L, |H-PDC|, |PDC-L|）
  - 其中：H=当日最高价，L=当日最低价，
    PDC=前一日收盘价
- 头寸规模单位
  = 账户的%风险 /（ATR * 每一最小交易单位）

计算公式

- 假设：R%是每只股票承担的%风险，C是可用资金，Pi是每只股票的最新价格，Vi是波动率，假如全仓买入所有候选股票，则：

$$\sum_{i=1}^{N} \frac{C \times R\%}{V_i} \times P_i = C$$

$$R\% = \frac{1}{\sum_{i=1}^{N} \frac{P_i}{V_i}}$$ 个股承担的%风险

$$K_i = Int(\frac{C \times R\%}{V_i \times 100}) \times 100$$ 个股的头寸

等波动率分配仓位

```
78              if code in g.head_values:
79                  head_value = g.head_values[code]
80                  if math.isnan(head_value):
81                      continue
82
83                  # 计算相对收益率
84                  profit_r = tail_value/head_value - 1.0
85
86                  # 计算ATR指标
87                  df = get_price(code, end_date=tail_date, count=ATR_WINDOW_SIZE+1, frequency
                        ="1d", skip_paused=True, fields=["high", "low", "close"])
88                  df["pdc"] = df["close"].shift(1)
89                  tr = df.apply(lambda x: max(x["high"]-x["low"], abs(x["high"]-x["pdc"]),
                        abs(x["pdc"]-x["low"])), axis=1)
90                  atr = tr[-ATR_WINDOW_SIZE:].mean()
91                  if math.isnan(atr):
92                      continue
93
94                  # 保存中间结果
95                  merged_list.append({
96                      "code": code,
97                      "profit_r": profit_r,
98                      "price": tail_value,
99                      "atr": atr
100                 })
```

```
3  import functools
4  ATR_WINDOW_SIZE = 20  # 计算ATR的窗口宽度
```

```
102        # 从首尾分别取出一定比例的股票，构造新的多空投资组合
103        merged_list.sort(key=lambda x: x["profit_r"], reverse=False)
104        num_to_keep = len(merged_list) * PERCENT_TO_KEEP / 100 // 2
105        g.new_long_pos  = {doc["code"]:doc for doc in merged_list[0:num_to_keep]}
106        g.new_short_pos = {doc["code"]:doc for doc in merged_list[-num_to_keep:]}
107
108        # 计算每只个股承担的风险百分比
109        sum = functools.reduce(lambda x, y: x + y["price"]/y["atr"], g.new_long_pos.values(
               ), 0)
110        sum += functools.reduce(lambda x, y: x + y["price"]/y["atr"], g.new_short_pos.
               values(), 0)
111        g.each_risk = 1 / sum
```

```
140        total_cash = p.available_margin
141
142        for code, doc in g.new_long_pos.items():
143            # num_to_buy = each_long_cash / g.tail_values[code] // 100 * 100
144            num_to_buy = total_cash * g.each_risk / doc["atr"] // 100 * 100
145            margincash_open(code, num_to_buy)
146
147        for code, doc in g.new_short_pos.items():
148            # num_to_sellshort = each_short_cash / g.tail_values[code] // 100 * 100
149            num_to_sellshort = total_cash * g.each_risk / doc["atr"] // 100 * 100
150            marginsec_open(code, num_to_sellshort)
```

```
# 导入函数库
import jqdata
import functools

ADJUST_INTERVAL = 20  # 调仓周期
CHK_PF_INTERVAL = 60  # 观测窗口
PERCENT_TO_KEEP = 10  # 组合容量
ATR_WINDOW_SIZE = 20  # 计算ATR的窗口宽度

# 初始化函数，回测开始时执行一次
def initialize(context):
    set_benchmark('000300.XSHG')  # 设定比较基准指数
    set_option('use_real_price', True)  # 默认复权模式

    # 设定融资/融券账户
    set_subportfolios([SubPortfolioConfig(cash=context.portfolio.cash,
                      type='stock_margin')])

    # 设定交易成本
    set_order_cost(OrderCost(open_tax=0, close_tax=0.000, open_commission=0.0000,
                             close_commission=0.0000, close_today_commission=0,
                             min_commission=0), type='stock')

    # 设定融资/融券的利率和保证金比例
    set_option('margincash_interest_rate', 0.00)
    set_option('margincash_margin_rate', 1.0)
    set_option('marginsec_interest_rate', 0.00)
    set_option('marginsec_margin_rate', 1.0)
```

```
    # 设置定时运行函数
    run_daily(before_market_open, time='before_open', reference_security='000300.XSHG')
    run_daily(market_open, time='open', reference_security='000300.XSHG')
    run_daily(after_market_close, time='after_close', reference_security='000300.XSHG')

    # 记录交易日期及其索引
    g.all_tdays = jqdata.get_all_trade_days()
    g.date_xlat = dict([(doc[1], doc[0]) for doc in enumerate(g.all_tdays)])

    # 设定若干全局变量
    g.elapsed_days = 0
    g.rebalance_today = False
    g.new_long_pos = {}
    g.new_short_pos = {}

# 在每个交易日开盘前运行
def before_market_open(context):
    cur_dt = context.current_dt.date()  # 当前日期
    if g.elapsed_days % ADJUST_INTERVAL == 0:  # 是否是调仓日
        log.info("Rebalance at %s", cur_dt)
        g.rebalance_today = True
        idx = g.date_xlat[cur_dt]

        # 观察窗口首末日期
        tail_date = g.all_tdays[idx-1]
        head_date = g.all_tdays[idx-1-CHK_PF_INTERVAL]
```

```
# 调仓日前一天的股票列表
candidates = list(get_all_securities(["stock"], tail_date).index)

# 观察窗口末尾日期的股票价格
tail_prices = get_price(candidates, tail_date, tail_date,
                        frequency="1d", fields=["close"])
g.tail_values = dict(tail_prices["close"].iloc[0])

# 观察窗口起始日期的股票价格
head_prices = get_price(candidates, head_date, head_date,
                        frequency="1d", fields=["close"])
g.head_values = dict(head_prices["close"].iloc[0])

# 在观察窗口内按收益率对股票排序（从小到大）
merged_list = []
for code, tail_value in g.tail_values.items():
    if math.isnan(tail_value):
        continue
    if code not in g.head_values:
        continue
    head_value = g.head_values[code]
    if math.isnan(head_value):
        continue

    # 计算相对收益率
    profit_r = tail_value/head_value - 1.0
```

```
        # 计算ATR指标
        df = get_price(code, end_date=tail_date, count=ATR_WINDOW_SIZE+1, frequency="1d",
                       skip_paused=True, fields=["high", "low", "close"])
        df["pdc"] = df["close"].shift(1)
        tr = df.apply(lambda x: max(x["high"]-x["low"], abs(x["high"]-x["pdc"]),
                      abs(x["pdc"]-x["low"])), axis=1)
        atr = tr[-ATR_WINDOW_SIZE:].mean()
        if math.isnan(atr):
            continue

        # 保存中间结果
        merged_list.append({
            "code": code,
            "profit_r": profit_r,
            "price": tail_value,
            "atr": atr
        })

    # 从首尾分别取出一定比例的股票，构造新的多空投资组合
    merged_list.sort(key=lambda x: x["profit_r"], reverse=False)

    # num_to_keep = len(merged_list) * PERCENT_TO_KEEP / 100 // 2
    num_to_keep = len(merged_list) * PERCENT_TO_KEEP // 100

    g.new_long_pos  = {doc["code"]:doc for doc in merged_list[0:num_to_keep]}
    # g.new_short_pos = {doc["code"]:doc for doc in merged_list[-num_to_keep:]}
```

```
        # 计算每只个股承担的风险百分比
        sum = functools.reduce(lambda x, y: x + y["price"]/y["atr"],
                               g.new_long_pos.values(), 0)
        # sum += functools.reduce(lambda x, y: x + y["price"]/y["atr"],
                                  g.new_short_pos.values(), 0)
        g.each_risk = 1 / sum

    else:
        g.rebalance_today = False

    g.elapsed_days += 1

# 在每个交易日开盘时运行
def market_open(context):
    p = context.portfolio.subportfolios[0]  # 融资/融券保证金账户
    if g.rebalance_today:
        # 再平衡步骤1：平掉原有多空仓位
        prev_long_pos = p.long_positions
        # prev_short_pos = p.short_positions

        for code, pos in prev_long_pos.items():
            margincash_close(code, pos.closeable_amount)

        """
        for code, pos in prev_short_pos.items():
            marginsec_close(code, pos.closeable_amount)
        """
```

```
# 再平衡步骤2：开立新的多空仓位
"""
each_long_cash = round(p.available_margin * 0.5 / len(g.new_long_pos), 2)
each_short_cash = round(p.available_margin * 0.5 / len(g.new_short_pos), 2)
"""
"""
each_long_cash = round(p.available_margin * 0.4 / len(g.new_long_pos), 2)
each_short_cash = round(p.available_margin * 0.4 / len(g.new_short_pos), 2)
"""
each_long_cash = round(p.available_margin / len(g.new_long_pos), 2)
# each_short_cash = round(p.available_margin / len(g.new_short_pos), 2)

total_cash = p.available_margin

for code, doc in g.new_long_pos.items():
    # num_to_buy = each_long_cash / g.tail_values[code] // 100 * 100
    num_to_buy = total_cash * g.each_risk / doc["atr"] // 100 * 100
    margincash_open(code, num_to_buy)

"""
for code, doc in g.new_short_pos.items():
    # num_to_sellshort = each_short_cash / g.tail_values[code] // 100 * 100
    num_to_sellshort = total_cash * g.each_risk / doc["atr"] // 100 * 100
    marginsec_open(code, num_to_sellshort)
"""
```

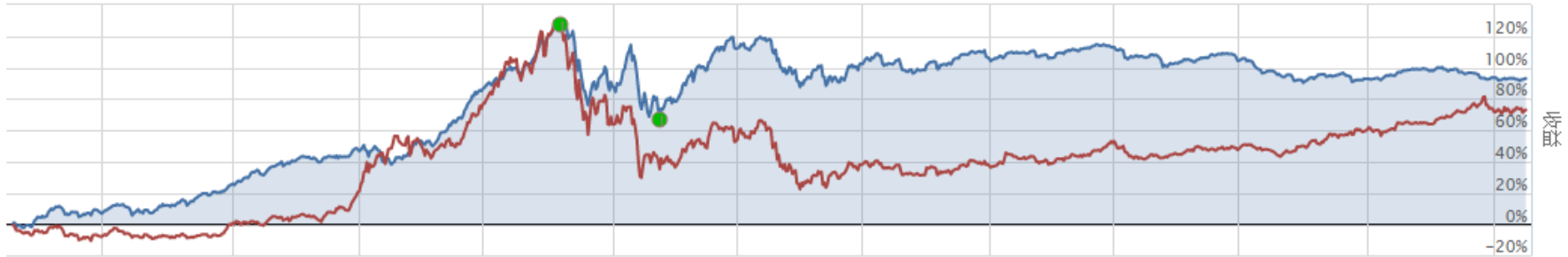
```
        else:
            # 非调仓日，可以增加止盈/止损等额外操作……
            pass

# 在每个交易日收盘后运行
def after_market_close(context):
    # 查看融资融券账户相关相关信息(更多请见API-对象-SubPortfolio)
    p = context.portfolio.subportfolios[0]
    log.info('- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -')
    log.info('查看融资融券账户相关相关信息(更多请见API-对象-SubPortfolio): ')
    log.info('总资产: ', p.total_value)
    log.info('净资产: ', p.net_value)
    log.info('总负债: ', p.total_liability)
    log.info('融资负债: ', p.cash_liability)
    log.info('融券负债: ', p.sec_liability)
    log.info('利息总负债: ', p.interest)
    log.info('可用保证金: ', p.available_margin)
    log.info('维持担保比例: ', p.maintenance_margin_rate)
    log.info('账户所属类型: ', p.type)
    log.info('##############################################################')
```

均仓分配资金

收益概述

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	胜率	盈亏比	最大回撤	
93.11%	18.34%	73.00%	0.088	0.497	0.882	0.545	1.509	27.186%	其他指标

缩放： 1个月 1年 全部 ■ 策略收益 ■ 基准收益 ■ 超额收益 ◉ 普通轴 ○ 对数轴 □ 超额收益 时间： 2013-12-30 - 2017-12-29



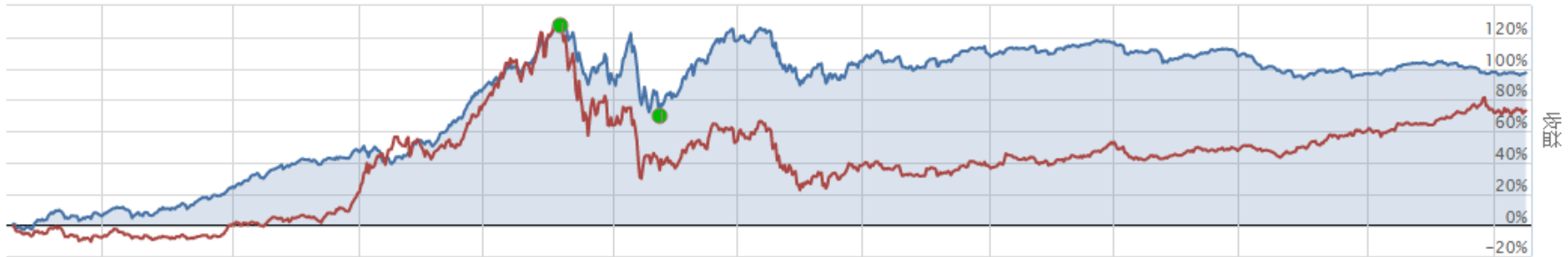
【多头/不加成本】

波动率（ATR）

收益概述

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	胜率	盈亏比	最大回撤	
97.27%	18.99%	73.00%	0.095	0.500	0.922	0.547	1.537	25.850%	其他指标

缩放： 1个月 1年 全部 ■ 策略收益 ■ 基准收益 ■ 超额收益 ◉ 普通轴 ○ 对数轴 □ 超额收益 时间： 2013-12-30 - 2017-12-29

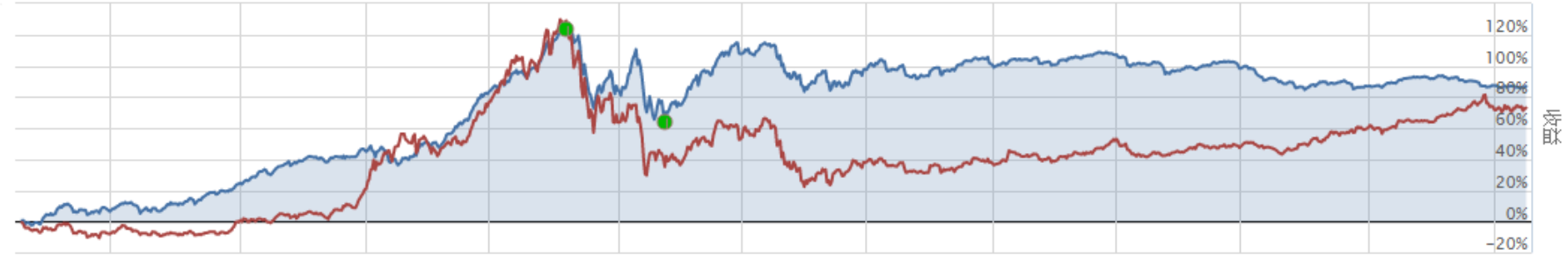


均仓分配资金

收益概述

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	胜率	盈亏比	最大回撤	
86.92%	17.36%	73.00%	0.079	0.496	0.823	0.545	1.520	27.265%	其他指标

缩放： 1个月 1年 全部 ■ 策略收益 ■ 基准收益 ■ 超额收益 ◉ 普通轴 ○ 对数轴 □ 超额收益 时间： 2013-12-30 - 2017-12-29



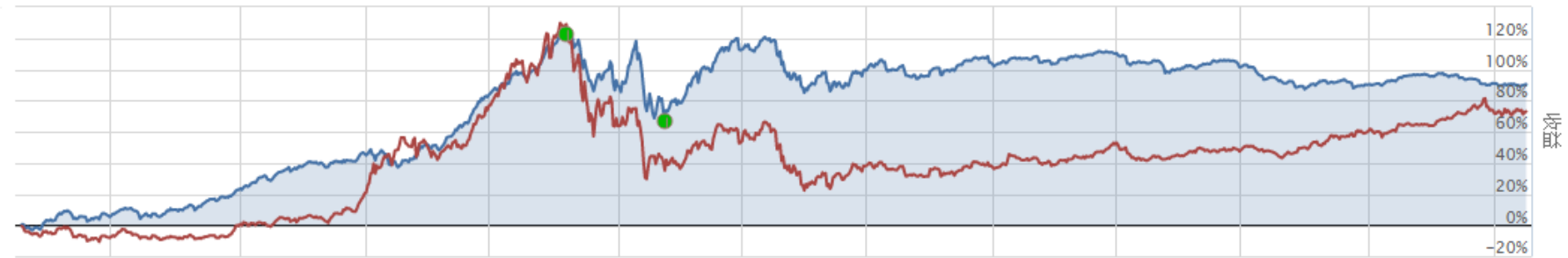
【多头/考虑成本】

波动率（ATR）

收益概述

策略收益	策略年化收益	基准收益	Alpha	Beta	Sharpe	胜率	盈亏比	最大回撤	
90.51%	17.93%	73.00%	0.084	0.499	0.858	0.546	1.547	25.963%	其他指标

缩放： 1个月 1年 全部 ■ 策略收益 ■ 基准收益 ■ 超额收益 ◉ 普通轴 ○ 对数轴 □ 超额收益 时间： 2013-12-30 - 2017-12-29



更多优化方向

- 调仓时，卖旧买新的逻辑如何更接近实战
- 大小周期，加择时，止盈止损
- 借鉴海龟交易法的动态头寸管理
- 行业板块占比均衡
- 成分股筛选和权重调整
- 参数优化（但要避免过拟合）

总结

- 均值回复策略的原理，思路验证
- 纯多头和多空双向的实现和比较
- 评估交易成本的影响
- 通过头寸管理方法进行策略优化

作业1

- 尝试对该策略进行改进，思路可以参考（但不限于）以下几点：
  - 观察区间和调仓频率的影响
  - 加择时信号
  - 加止盈止损
  - 其它更多优化方向的思路

作业2

- 对均值回复特性验证的实验（Song's Hypothesis），进行更多维度的测试：
  - 时段1和时段2间隔时间长度的影响
    - 对应：均值回复的尺度
  - 不同起始点的影响
    - 对应：在什么样的大趋势下有此特征

下节课预告

- 题目：配对型交易策略编写

  - 编程语言和运行平台：
    Multicharts（MC）- http://multicharts.cn

  - 准备工作：
    - 提前下载、安装MC软件，注册试用账号
    - 了解MC的用法、工作原理、编程语法和函数等

问答互动

在所报课的课程页面，
1、点击“全部问题”显示本课程所有学员提问的问题。
2、点击“提问”即可向该课程的老师和助教提问问题。

联系我们

小象学院：互联网新技术在线教育领航者

– 微信公众号：**小象学院**

THANKS