★★★

严谨　认真　持续

量化策略制作：用Python写一个自定义因子并进行检验

量化投资实战交易体系必修课

系列4 -快速开发一个量化策略

普量学院 PLOUTO-QUANTS

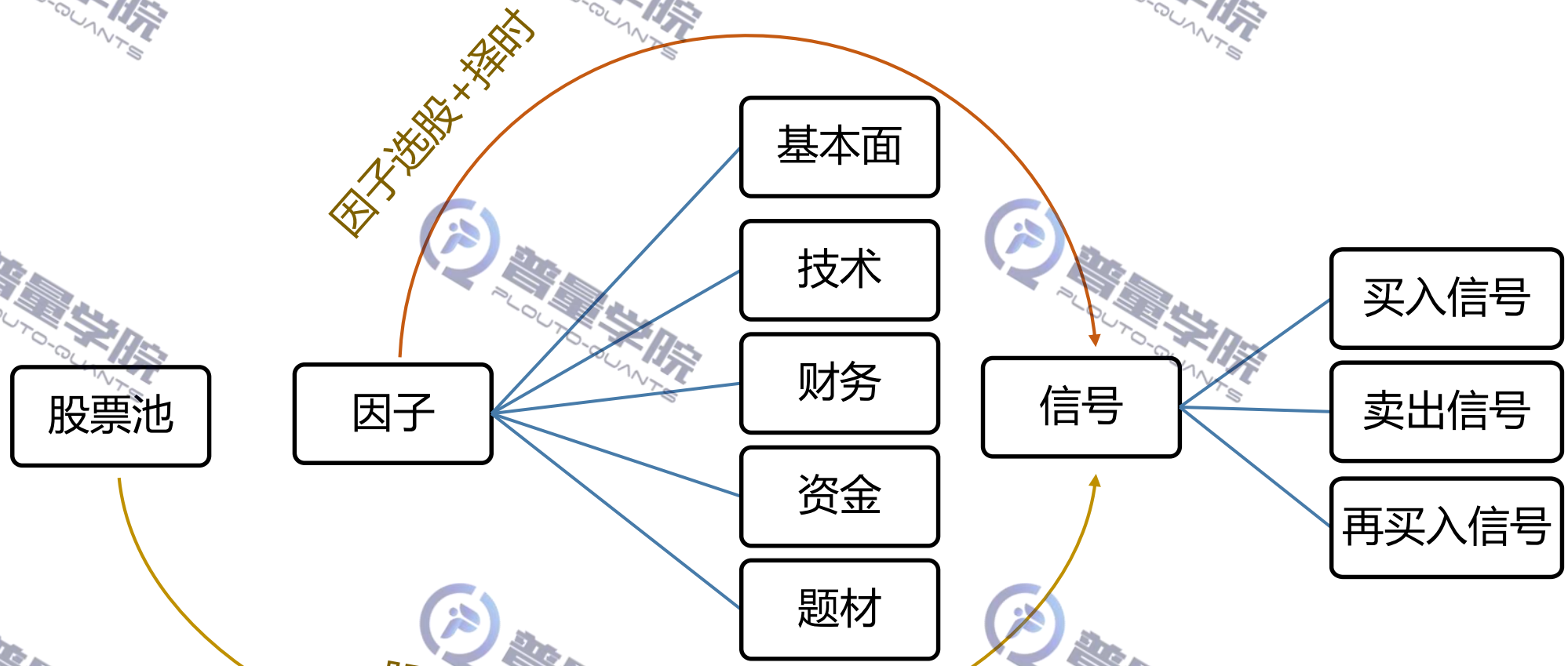
目录

CONTENT

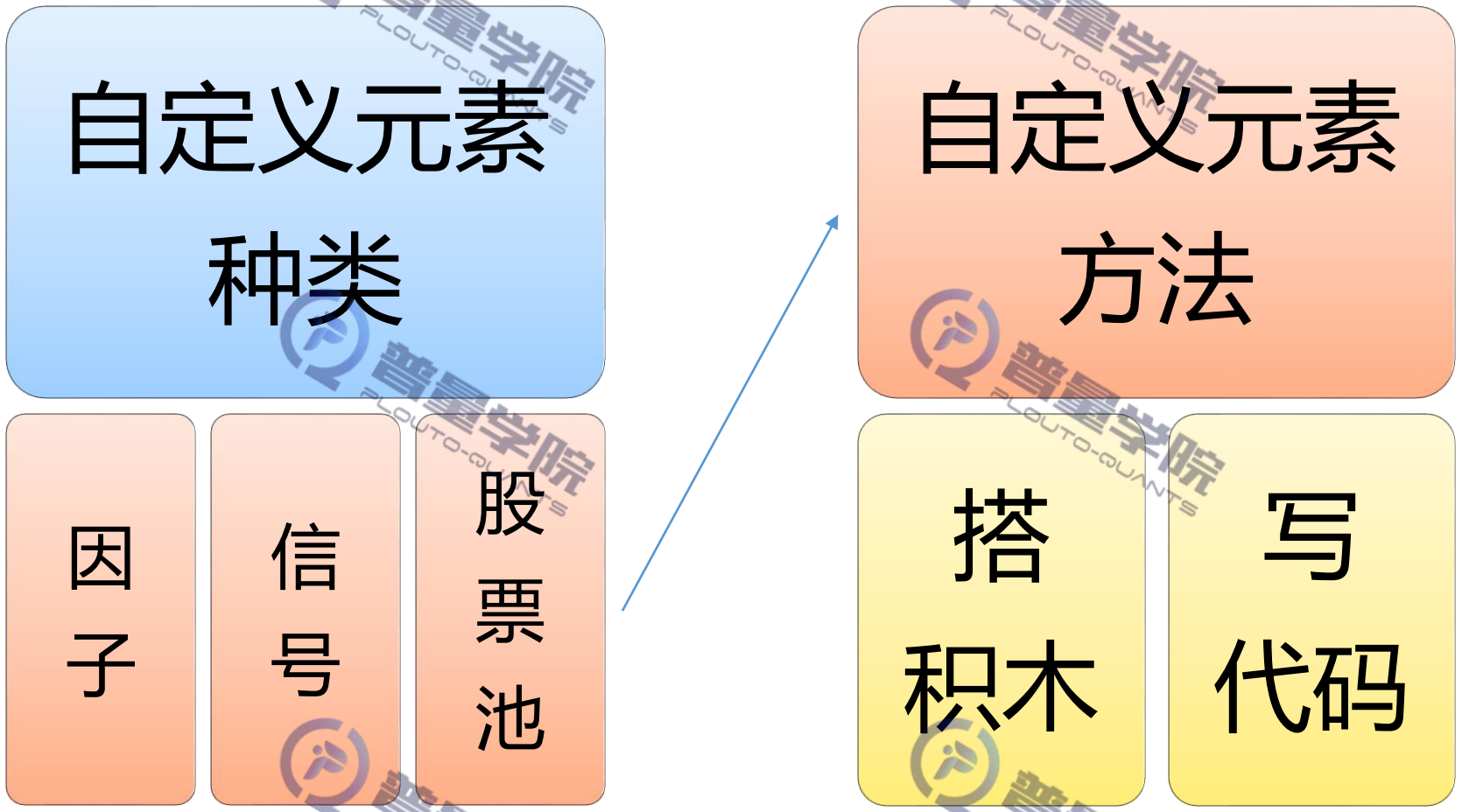
自定义因子

PART ONE

回顾一下



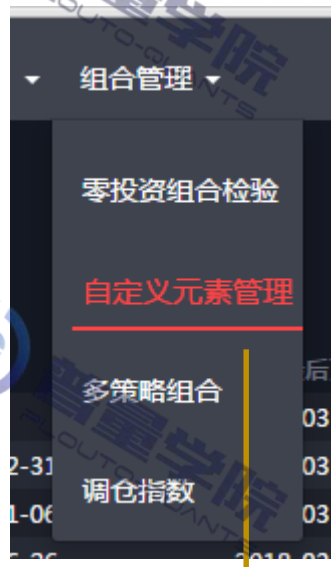
自定义元素



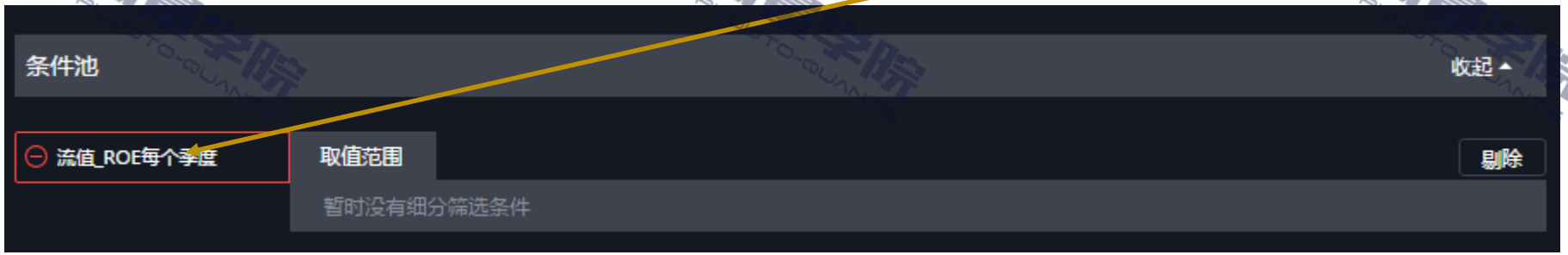
搭积木 – 自定义股票池


股票池 - 流值_ROE每个季度
选股条件
交易规则
查看规律
保存为策略
保存为指数
保存为股票池
保存为信号
保存为因子
股票池选股
因子选股
特色兵器库
形态类
短线高手
排行榜
条件设置
基本指标
财务指标
技术指标
资金指标
题材
K线形态检测
低位蓄势待发
优势形态精选
三重筑底反弹
长期筑底
近期筑底
主力拉升ing
反转突破
洗盘追踪
价格区间高低位
超跌共振反弹_60+8+3
超跌共振反弹_60+5+0
强势股_60日前200_8日前200
条件池
收起
净资产收益率ROE(四季度)
取值范围
特殊条件
删除
小于0
0~5%
5%~15%
15%~20%
20%~30%
大于30%
自定义
20 %~ %
净资产收益率ROE(三季度)
15 %~ %
净资产收益率ROE(二季度)
10 %~ %
净资产收益率ROE(一季度)
5 %~ %
流通市值
小于20亿
20亿~50亿
50亿~100亿
100亿~400亿
大于400亿
100 亿~ 1000 亿

自定义股票池


组合管理
零投资组合检验
自定义元素管理
多策略组合
调仓指数


Plouto-Quants
创建策略
我的策略
数据分析
监控
实盘交易
组合管理
新策略
选股条件
交易规则
查看规律
保存为策略
保存为指数
保存为股票池
保存为信号
保存为因子
股票池选股
流值_ROE每个季度
08-halfylast2%120D
08-halfylast2%60D
08-halfylast2%30D
08-1ylast2%250D
08-1ylast2%120D
流值_ROE每个季度
起始日期：2013-03-01，调整周期：1个交易日。
加入条件池
条件池
收起


条件池
收起
流值_ROE每个季度
取值范围
剔除
暂时没有细分筛选条件


自定义元素管理
新 增
名称
显示名称
类别
是否发布
已提取数据
最后更新时间
操作
流值_ROE每个季度
流值_ROE每个季度
股票池
是
2013-03-01 -- 2018-03-14
2018-03-15 10:33:46
编辑 删除 查看数据 查看规律 取消发布

搭积木 – 自定义信号


信号 - PE大于40
选股条件
交易规则
查看规律
保存为策略
保存为指数
保存为股票池
保存为信号
保存为因子
股票池选股
因子选股
特色兵器库
条件池
收起
市盈率(滚动)
取值范围
剔除
小于0
0~30
30~70
70~300
大于300
自定义
40
倍~
倍
形态类
短线高手
排行榜
条件设置
基本指标
K线形态检测
低位蓄势待发
优势形态精选
三重筑底反弹
长期筑底
近期筑底
主力拉升ing
反转突破

自定义信号

选股条件

交易规则

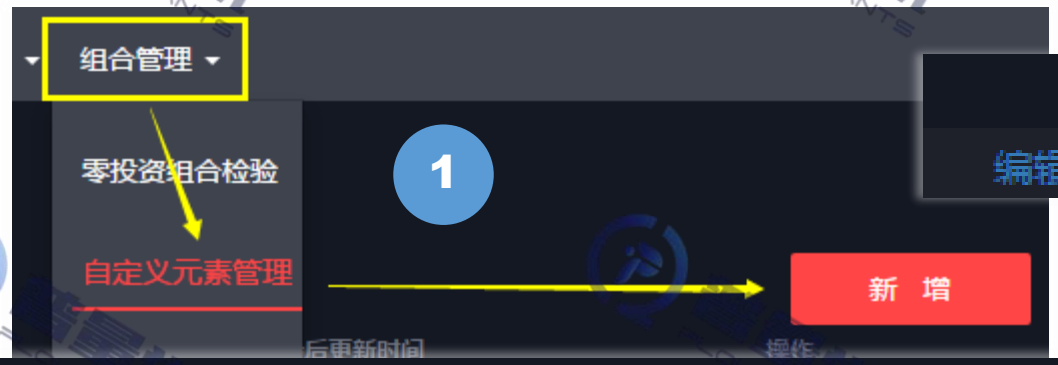
类别	必选条件	可选条件	
买入信号	开盘买入		
卖出信号	信号触发卖出	PE大于40[1分钟]	
持仓设置		同时持仓股票数上限10只	资金占比上限100%

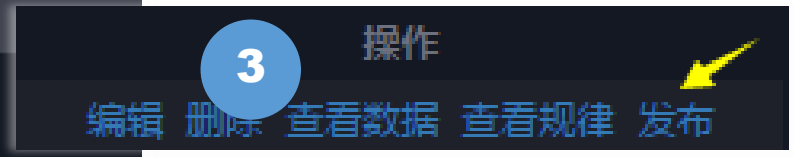
搭积木 - 自定义因子


因子 - 扣非ROE大于30%
选股条件
交易规则
查看规律
保存为策略
保存为指数
保存为股票池
保存为信号
保存为因子
股票池选股
因子选股
特色兵器库
条件池
收起
净资产收益率ROE
取值范围
特殊条件
剔除
小于0
0~5%
5%~15%
15%~20%
20%~30%
大于30%
自定义
形态类
K线形态检测
短线高手
低位蓄势待发
优势形态精选
排行榜
三重筑底反弹

自定义因子

写代码 - DIY自定义元素





配置文件-自定义因子/信号/股票池

```
# 类别：stock_pool-股票池；signal-信号；factor-股票因子
# type =
# 名称，英文标识，不同类别下的名称可以一样，但是同一个类别下的不能相同，或者会被覆盖
# name =
# 用来显示的中文名称，正式保存后，会显示普量云的页面上
# display =
# 生成的数据要保存的数据表，在下面的代码中需要将计算的数据保存到该collection中
# collection =
# 因子值对应的字段，如果为空，则默认使用value
# value_field =
# 其他参数名称，如果为空则表明没有其他参数
# params=
# 信号的特有参数，信号对应的买卖方向，buy-买入；sell-卖出
# direction =
# 信号的特有参数，时间级别，1、5、15、30、60-分钟级别；D-日，W-周，M-月, 5D-5日
# signal_period =
```

数据接口（API）

- get_stocks()
  - 获取：股票代码、名称、所属行业、交易所、上市日期
- get_daily(code, from, to)
  - 获取：股票代码、名称、日期、开高低收、前收、涨跌幅、成交量、成交额、PE、交易状态、复权因子等
- get_factor(factor, from, to)
  - 获取：股票代码、日期、因子数值

自定义元素条目

- 信号
  - {code，date，price}
- 因子
  - {code，date，[value_field]}
- 股票池
  - {code，date}

自定义因子举例

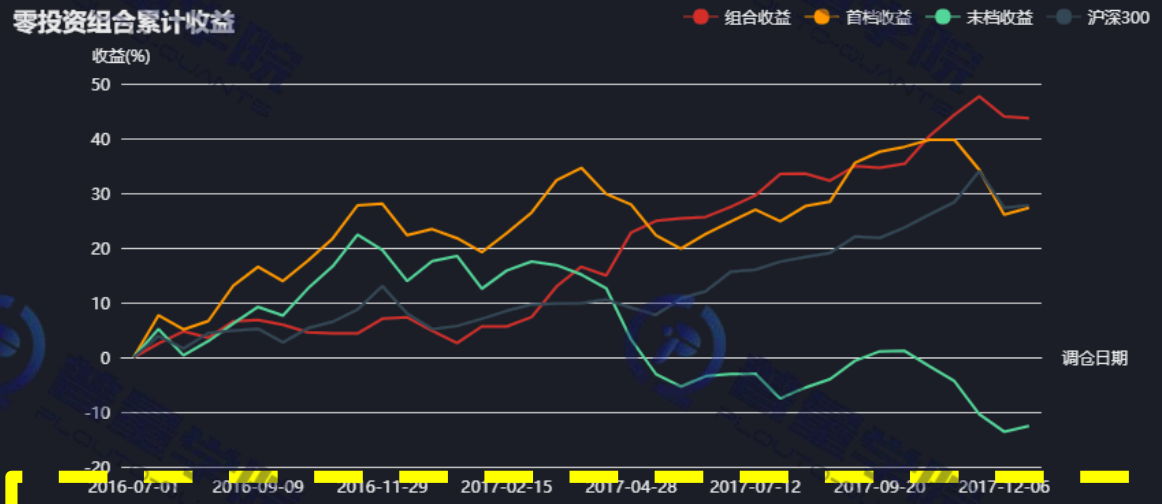
改进的ROE

零投资组合检验

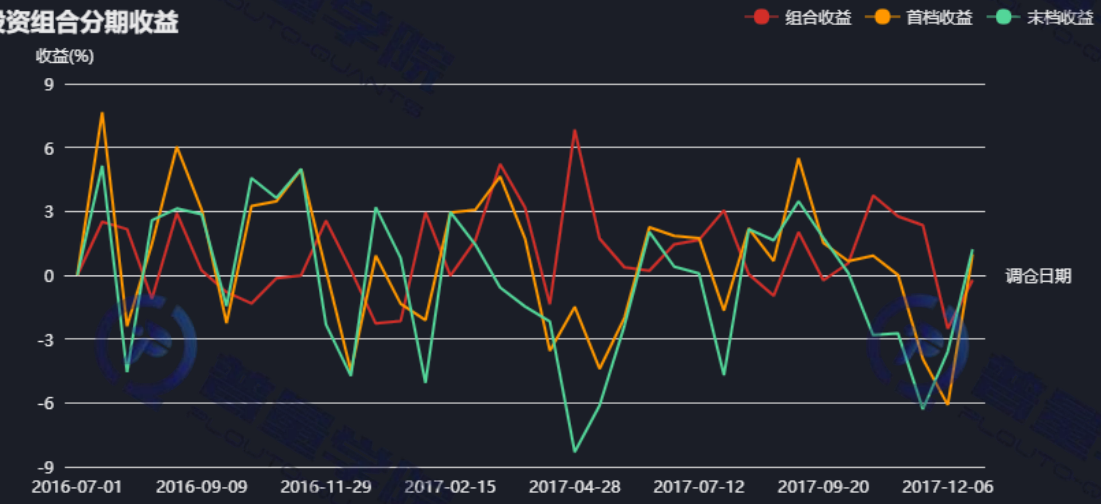
类别 盈利能力指标　　因子 净资产收益率ROE(加权)　　检测跨度 2016-07-01 - 2017-12-31

档位划分 10　　排序方式 倒序（从大到小）　　调仓频率(天) 10　　计算

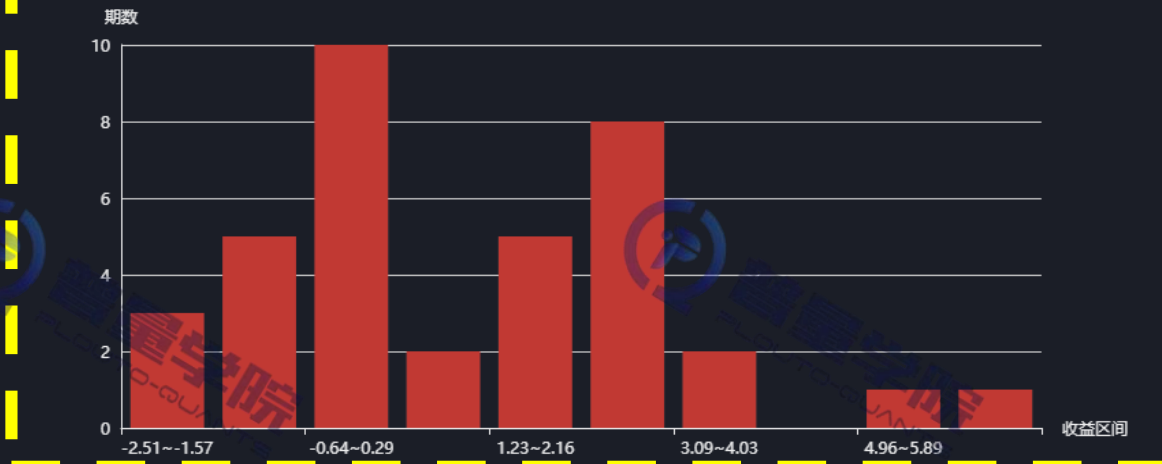
零投资组合累计收益



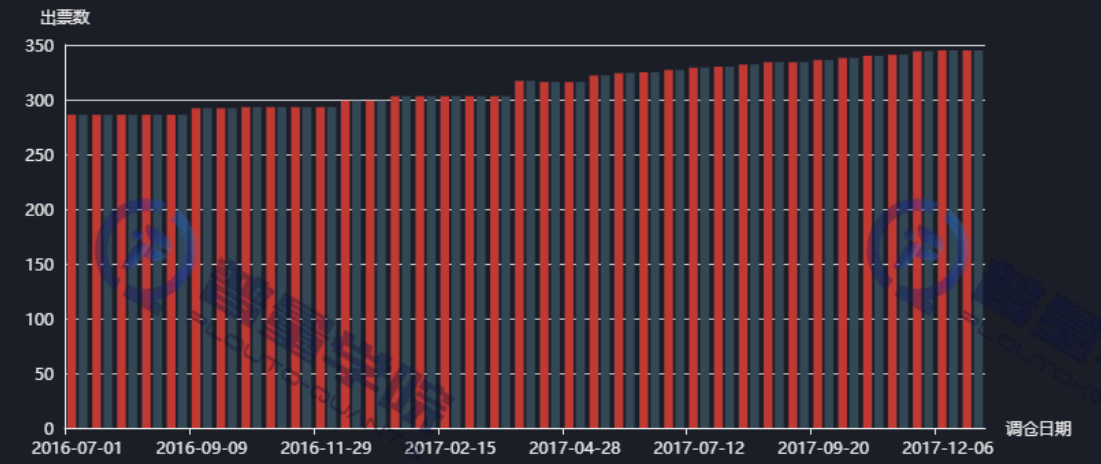
零投资组合分期收益



收益分布



多空数量对比 - 总期数：37



ROE的杜邦分解

权益乘数的升高反映了企业杠杆率的提升，并不代表企业竞争优势的提升，在某种程度上反被视为是风险的聚集！

Action：
尝试去掉它！

企业定价权提升，以及成本和费用管理能力提升

(相对于同行业其它公司)资产使用和管理效率的提升

(来源于企业资本结构的变化)反映了企业的财务杠杆水平

ROE分解

$$\text{ROE} = \frac{\text{净利润}}{\text{所有者权益}}$$

$$= \frac{\text{净利润}}{\text{营业收入}} \times \frac{\text{营业收入}}{\text{总资产}} \times \frac{\text{总资产}}{\text{所有者权益}}$$

$$= \text{净利润率} \times \text{资产周转率} \times \text{权益乘数}$$

$$\text{改进的ROE} = \text{净利润率} \times \text{资产周转率}$$

$$= \frac{\text{ROE}}{\text{权益乘数}}$$

get_factor() → roe_weighting
get_factor() → equity_multiple

代码 - 1

```
# 类别：stock_pool - 股票池，signal - 信号，factor - 股票因子
# type = factor
# 名称，英文标识，不同类别下的名称可以一样，但是同一个类别下的不能相同，或者会被覆盖
# name = improved_roe
# 用来显示的中文名称，正式保存后，会显示普量云的页面上
# display = 改进的ROE
# 生成的数据要保存的数据表，在下面的代码中需要将计算的数据保存到该collection中
# collection = improved_roe
# 因子值对应的字段，如果为空，则默认使用value
# value_field = i_roe
# 其他参数名称，如果为空则表明没有其他参数
# params=
# 信号的特有参数，信号对应的买卖方向，buy - 买入， sell - 卖出
# direction =
# 信号的特有参数，时间级别，1、5、15、30、60 - 分钟级别，D - 日，W - 周， M - 月，5D - 5日
# signal_period =
# 股票池的特有参数：调整周期，单位是交易日
# period =
# 股票池的特有参数：开始日期，即股票开始起作用的日期，如果指定的为非交易日，则从最近的一个交易开始起作用
# start_date =

# PLEASE DON'T REMOVE
# CODE START
```

代码 - 2

```
from datetime import datetime, timedelta
# 每天会被外部执行器定时调用的函数
def compute_daily():
    _today = datetime.now().strftime("%Y-%m-%d")
    return compute_one_day(_today)

# 计算某一天的数据，盘后执行
def compute_one_day(_date):
    # 获取该日所有股票的"ROE"值
    results = get_factor("roe_weighting", _date, _date)
    if not results["success"]:
        return None
    all_roes = dict([(doc["code"], doc["value"]) for doc in results["data"]["factors"]])

    # 获取该日所有股票的"权益乘数"
    results = get_factor("equity_multiple", _date, _date)
    if not results["success"]:
        return None
    all_multiples = dict([(doc["code"], doc["value"]) for doc in results["data"]["factors"]])
```

代码 - 3

```
    factors = []
    for code, roe in all_roes.items():
        if not is_stock(code): # 确保该code是股票代码，而不是指数
            continue

        # 取得该code对应的权益乘数
        multiple = all_multiples.get(code)
        if multiple is not None:
            # 只有在roe和权益乘数都存在的前提下，才计算改进的roe
            improved_roe = roe / multiple
            # 保存该自定义因子
            record = dict(
                code = code,
                date = _date,
                i_roe = improved_roe
                )
            factors.append(record)

    return factors
```

代码 - 4

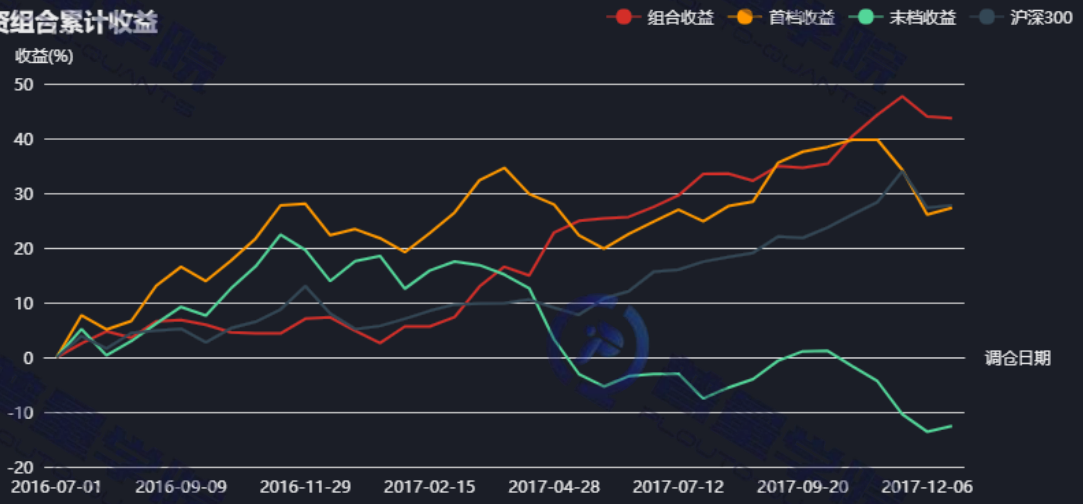
```
def is_stock(code):
    return code is not None and len(code) == 6 and code[0:2] in ["00", "30", "60"]

# 用于计算历史数据的方法，该方法重算历史数据时使用需要计算历史
def compute_history():
    _date = datetime(2016, 7, 1, 0, 0, 0)
    one_day = timedelta(days=1)
    _now = datetime.now()
    factors = []
    while _date <= _now:
        _date_str = _date.strftime("%Y-%m-%d")
        print("正在提取的日期: %s" % _date_str, flush=True)
        one_day_factors = compute_one_day(_date_str)
        if one_day_factors is not None and len(one_day_factors) > 0:
            factors += one_day_factors
        _date += one_day

    return factors
```

零投资组合检验

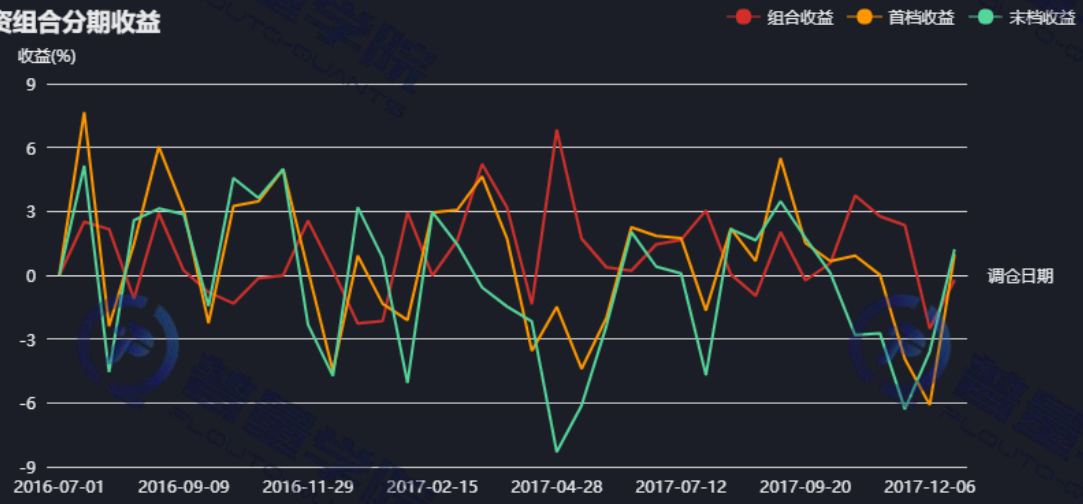
类别 盈利能力指标　　因子 净资产收益率ROE(加权)　　检测跨度 2016-07-01 - 2017-12-31

档位划分 10　　排序方式 倒序（从大到小）　　调仓频率(天) 10　　计算

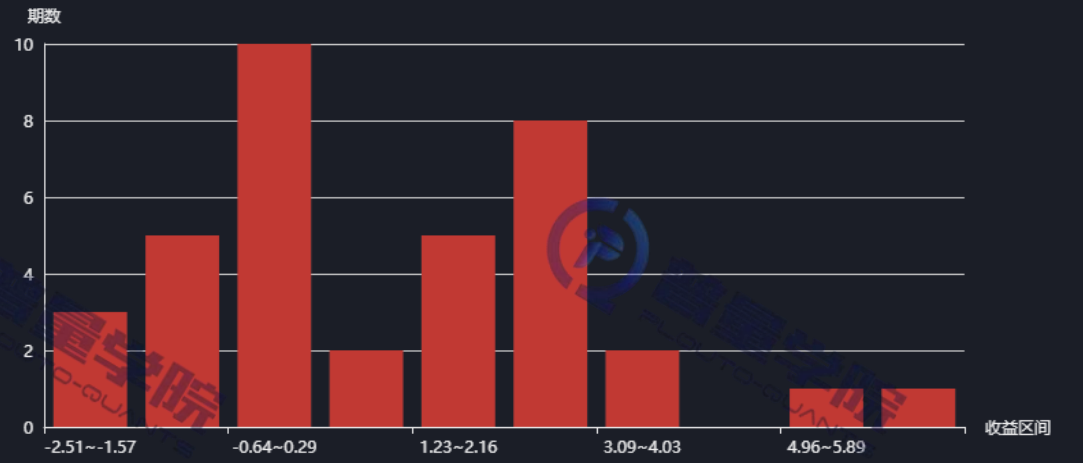
零投资组合累计收益



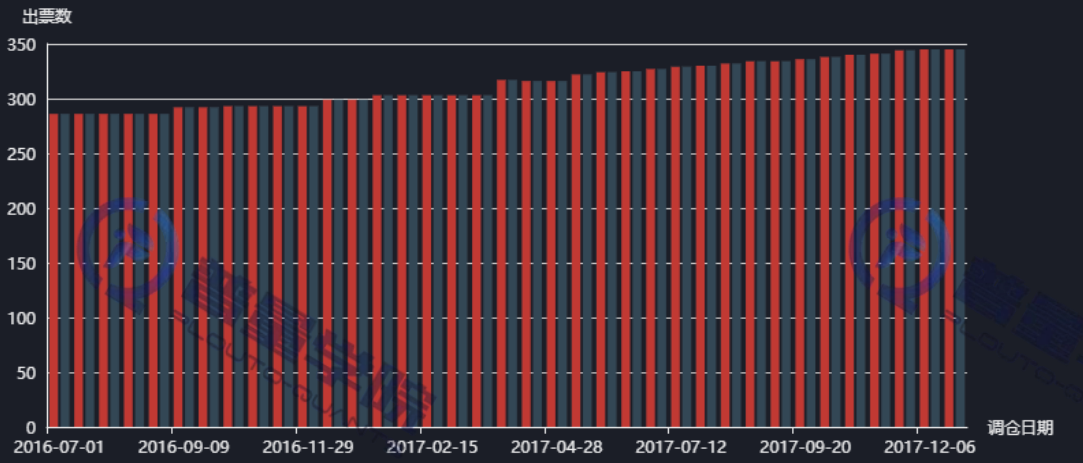
零投资组合分期收益



收益分布



多空数量对比 - 总期数：37

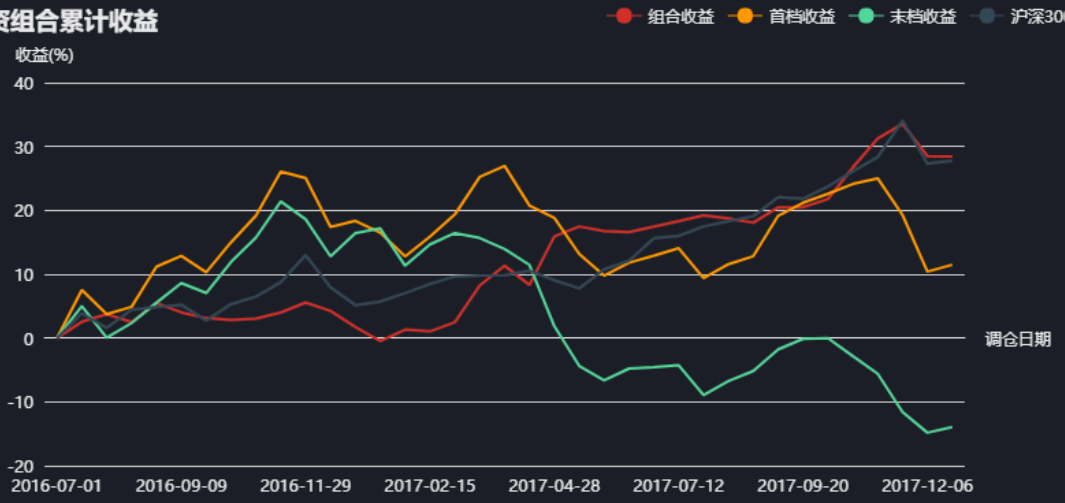


零投资组合检验

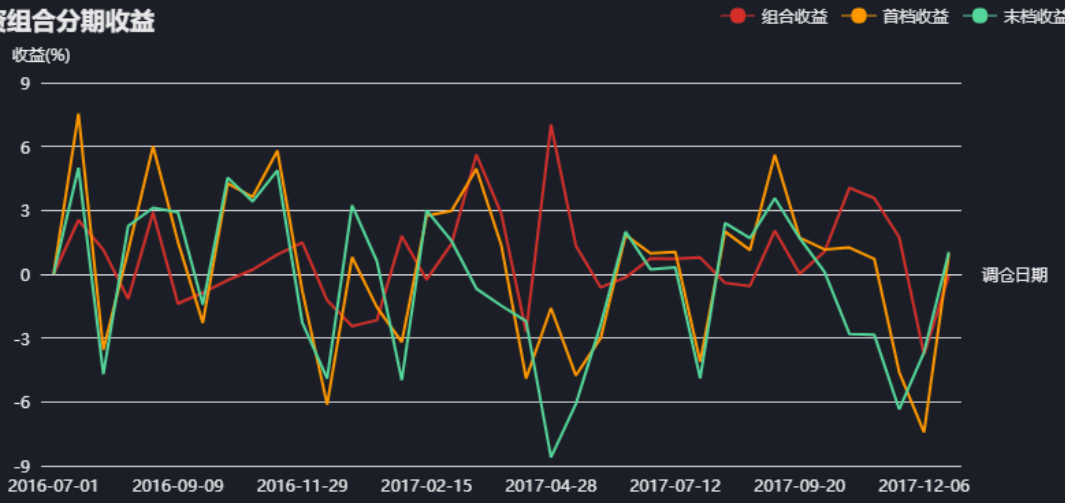
类别: 自定义因子　　因子: 改进的ROE　　检测跨度: 2016-07-01 - 2017-12-31

档位划分: 10　　排序方式: 倒序（从大到小）　　调仓频率(天): 10　　计算

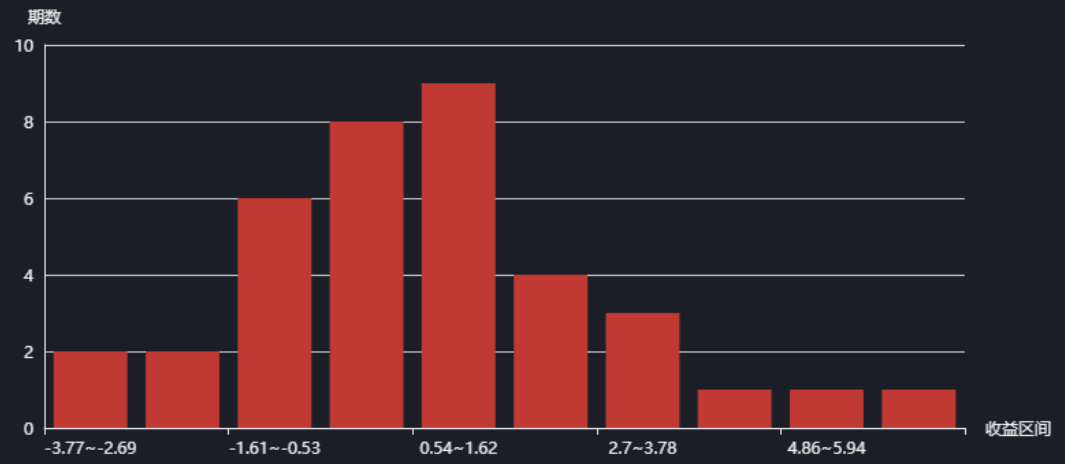
零投资组合累计收益



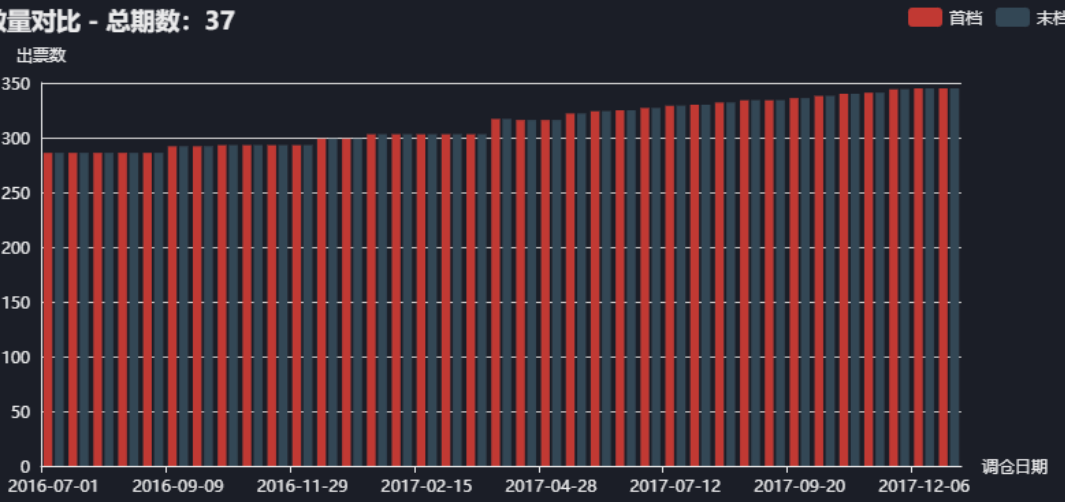
零投资组合分期收益



收益分布



多空数量对比 - 总期数：37



因子的有效性检验

PART TWO

Step1

- 选定研究因子，并使用每一时期的该因子敞口对股票进行排序，对排序结果进行五（或十）分为划割

Step2

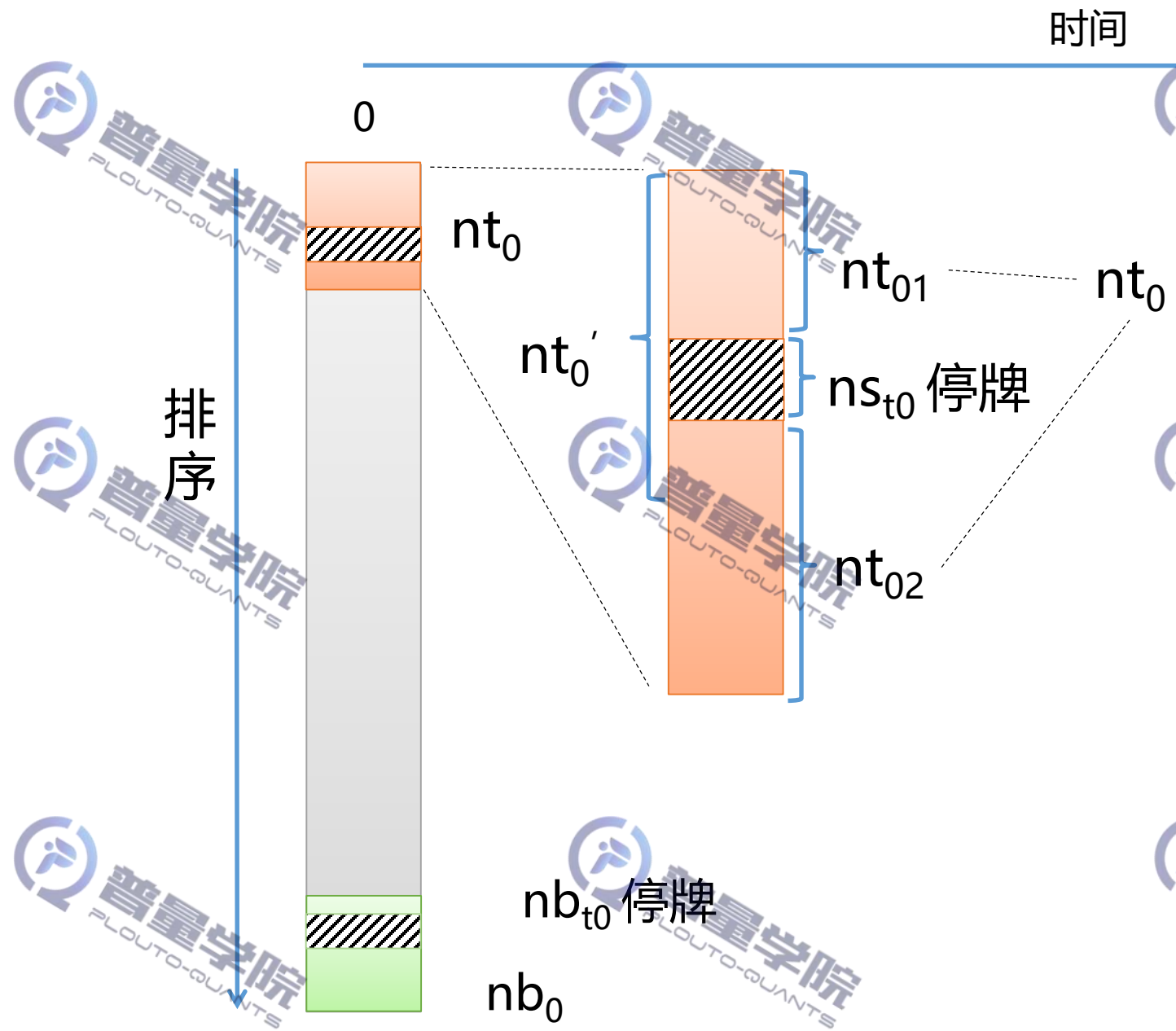
- 在第一个五分位里构造相同权重的股票组合，同时在最后一个五分位里相同权重的股票组合，第一个五分位是根据该因子排序的前1/5的股票，最后一个五分位是根据该因子的最后1/5的股票

Step3

- 计算两个股票组合的月度收益，并计算两个组合收益率的差值，该差值即为零投资组合的收益。这里的零投资组合是通过买进最顶层划分中的投资组合并卖空最底层划分中的投资组合得到的，它被称为零投资组合是因为从理论上讲构造该组合不需要资本。

Step4

- 在计算过零投资组合的收益后，我们进行统计检验，以考察等权重股票组合与零投资组合收益是否存在显著不同



当期数量

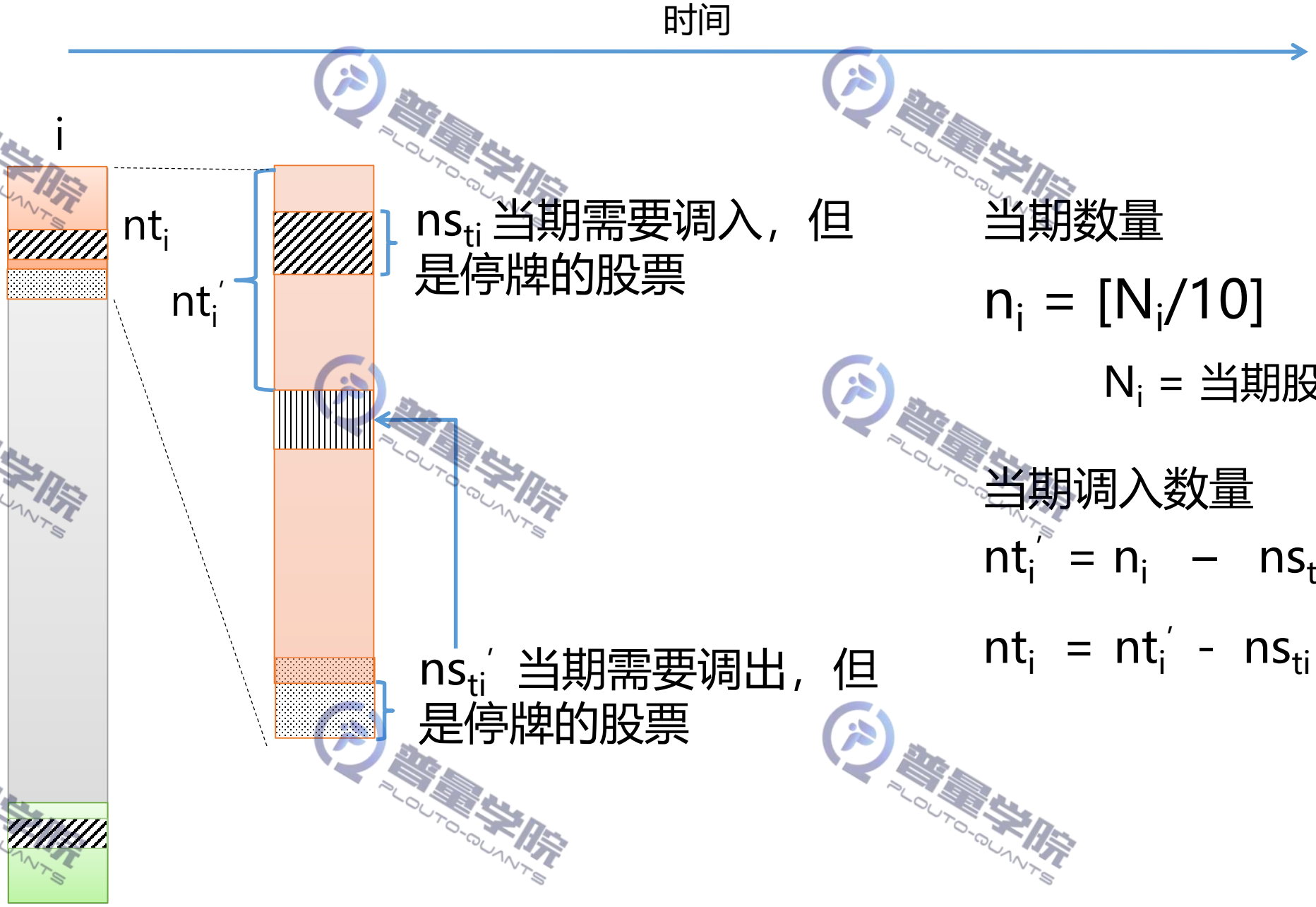
$n_0 = [N_0/10]$ N_0 = 当期股票总数

当期调入数量

$nt_0 = nb_0 = n_0$

$nt_0 = nt_0' - ns_{t0}$

$nb_0 = nb_0' - nb_{t0}$



当期数量

$n_i = [N_i/10]$

N_i = 当期股票总数

当期调入数量

$nt_i' = n_i - ns_{ti}'$

$nt_i = nt_i' - ns_{ti}$

计算收益

当期首末档收益

① $$P_i = \frac{1}{n}\sum_{m=0}^{n}\left(\frac{Close_{mi}}{Close_{m(i-1)}} - 1\right)$$

组合收益

② $$P_{pi} = P_{ti} - P_{bi}$$

累计收益（复利）

③ $$P = \prod_{i=1}^{m}(1+P_i) - 1,\ m \in \{1,2,3,\ldots,n\}$$

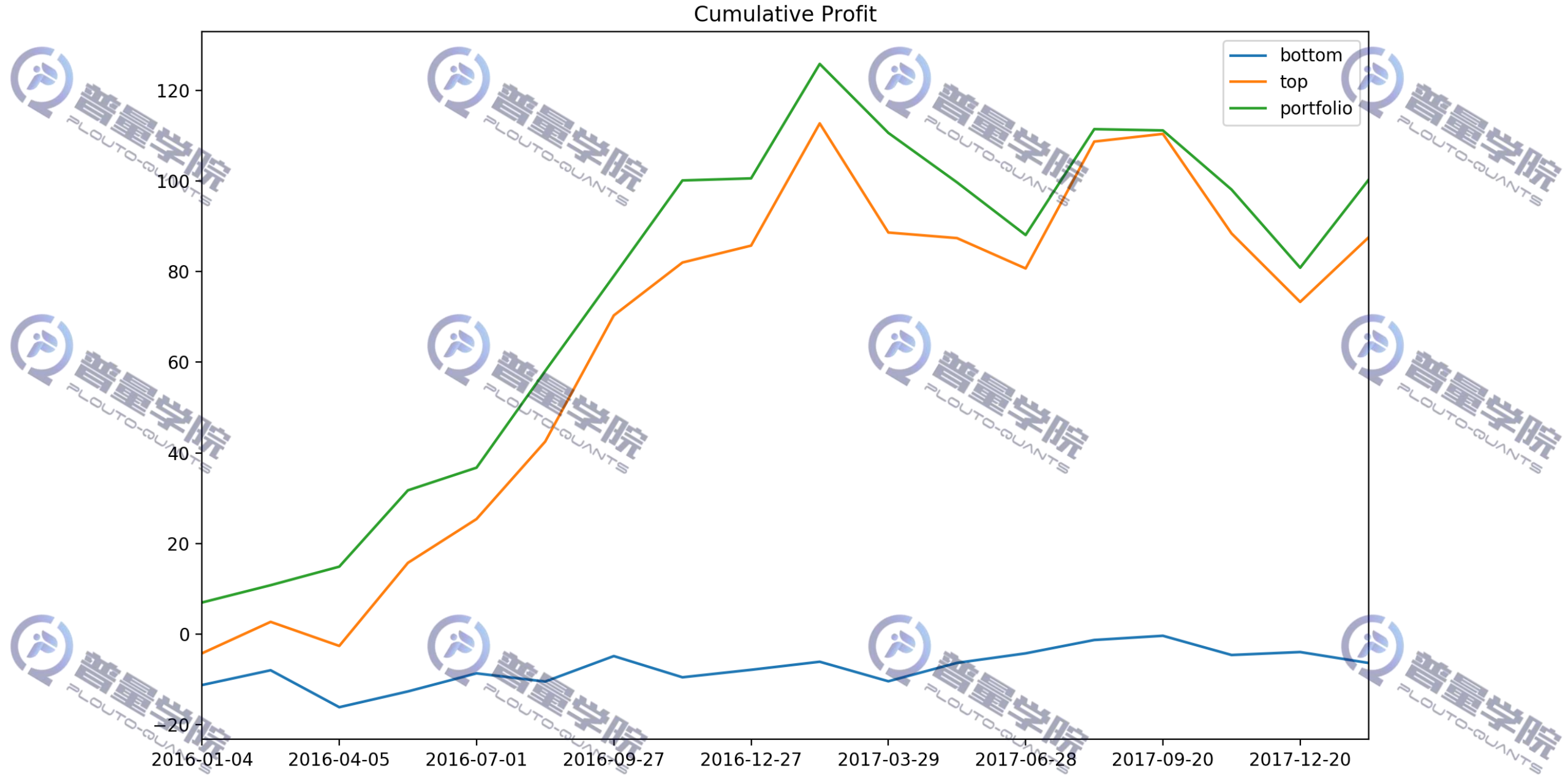
只有当第n期恰好结束时，m = n

单期收益

	bottom	top	portfolio
2016-01-04	-11.21	-4.21	7.00
2016-02-22	3.67	7.24	3.57
2016-04-05	-8.85	-5.16	3.69
2016-05-18	4.15	18.81	14.66
2016-07-01	4.56	8.35	3.79
2016-08-12	-1.98	13.62	15.60
2016-09-27	6.28	19.54	13.26
2016-11-15	-4.92	6.85	11.77
2016-12-27	1.82	2.05	0.23
2017-02-15	1.93	14.52	12.59
2017-03-29	-4.58	-11.32	-6.74
2017-05-15	4.53	-0.65	-5.18
2017-06-28	2.24	-3.58	-5.82
2017-08-09	3.08	15.50	12.42
2017-09-20	0.94	0.81	-0.13
2017-11-08	-4.26	-10.44	-6.18
2017-12-20	0.69	-8.02	-8.71
2018-02-01	-2.52	8.24	10.76

累积收益

	bottom	top	portfolio
2016-01-04	-11.21	-4.21	7.00
2016-02-22	-7.95	2.73	10.82
2016-04-05	-16.10	-2.58	14.91
2016-05-18	-12.62	15.75	31.75
2016-07-01	-8.63	25.42	36.75
2016-08-12	-10.44	42.50	58.08
2016-09-27	-4.82	70.34	79.04
2016-11-15	-9.50	82.01	100.12
2016-12-27	-7.85	85.74	100.58
2017-02-15	-6.07	112.71	125.83
2017-03-29	-10.38	88.63	110.61
2017-05-15	-6.32	87.40	99.70
2017-06-28	-4.22	80.70	88.08
2017-08-09	-1.27	108.70	111.44
2017-09-20	-0.34	110.39	111.16
2017-11-08	-4.58	88.43	98.11
2017-12-20	-3.93	73.32	80.86
2018-02-01	-6.35	87.60	100.32



```
# 构建股票代码和后复权价格相对应的dictionary
code_post_close_dict = dict()
for daily in dailies:
    code_post_close_dict[daily['code']] = daily['close'] * daily['adjfactor']

#计算收益
self.compute_profit(last_adjust_date, code_post_close_dict, top_dailies, bottom_dailies, adjust_date)

# 计算每当包含的股票数
total_size = len(dailies)
single_position_count = int(total_size/10)

#调整首档组合
self.adjust_top_position(top_dailies, dailies, single_position_count)

#调整末档组合
self.adjust_bottom_position(bottom_dailies, dailies, single_position_count)
```

计算上期收益

调整组合

```python
def adjust_top_position(self, top_dailies, dailies, single_position_count):
    # 移除首档非停牌股票
    self.remove_stocks(top_dailies, dailies)

    # 首档股票，保留后复权的价格
    top_size = len(top_dailies.keys())
    for daily in dailies:
        # 可能已经存在股票，所以需要先判断是否已经满足了数量要求
        if top_size == single_position_count:
            break

        # 只有是交易状态的股票才被纳入组合
        code = daily['code']
        if daily['trade_status'] == '交易':
            top_dailies[code] = daily['close'] * daily['adjfactor']
            top_size += 1
```

调整首档组合

- 移除上期调入且当前非停牌股票
- 加入当期应跳入且当前非停牌股票
- 加入时，应按照排序从头逐个加入，直到满足数量要求

```
"""
移除当期中非停牌的股票
"""
def remove_stocks(self, position_dailies, dailies):
    # 首期时，还不存在股票
    if len(position_dailies.keys()) == 0:
        pass

    # 只有非停牌的股票，才会被移除
    for daily in dailies:
        code = daily['code']
        if daily['trade_status'] == '交易' and code in position_dailies:
            del position_dailies[code]
```

```python
def adjust_bottom_position(self, bottom_dailies, dailies, single_position_count):
    # 移除首档非停牌股票
    self.remove_stocks(bottom_dailies, dailies)
    # 末档股票，保留后复权的价格
    bottom_size = len(bottom_dailies.keys())
    # 所有股票数
    total_size = len(dailies)
    for index in range(1, total_size):
        # 可能已经存在股票，所以需要先判断是否已经满足了数量要求
        if bottom_size == single_position_count:
            break

        # 末档要从最后一个向前回溯
        daily = dailies[-index]
        code = daily['code']
        # 只有是交易状态的股票才被纳入组合
        if daily['trade_status'] == '交易':
            bottom_dailies[code] = daily['close'] * daily['adjfactor']
            bottom_size += 1
```

调整末档组合

- 移除上期调入且当前非停牌股票
- 加入当期应跳入且当前非停牌股票
- 加入时，应按照排序从末尾逐个加入，直到满足数量要求

```
# 计算累积收益（复利方式）
# 首档
top_cumulative_profit = round((self.last_top_net_value * (1 + top_profit[0] / 100) - 1) * 100, 2)
self.last_top_net_value *= (1 + top_profit[0] / 100)
# 末档
bottom_cumulative_profit = round((self.last_bottom_net_value * (1 + bottom_profit[0] / 100) - 1) * 100, 2)
self.last_bottom_net_value *= (1 + bottom_profit[0] / 100)
#组合
portfolio_cumulative_profit = round((self.last_portfolio_net_value * (1 + portfolio_profit / 100) - 1) * 100, 2)
self.last_portfolio_net_value *= (1 + portfolio_profit / 100)
```

$$P = \prod_{i=1}^{m} (1 + P_i) - 1,\quad m \in \{1,2,3,\ldots,n\}$$

```python
def compute_average_profit(self, code_post_close_dict, position_code_close_dict, _date):
    # 提取股票列表
    codes = list(position_code_close_dict.keys())

    # 只有存在股票时，才进行计算
    if len(codes) > 0 :
        # 所有股票的累计收益
        profit_sum = 0
        # 实际参与统计的股票数
        count = 0
        # 计算所有股票的收益
        for code in codes:
            count += 1
            buy_close = position_code_close_dict[code]
            if code in code_post_close_dict:
                profit_sum += (code_post_close_dict[code] - buy_close)/buy_close
            else:
                # 有些停牌的股票没有日线，因此向前查找停牌前的最近一个时间
                last_daily = db.daily.find_one(
                    {'code':code, 'time': {'$lte': _date}},
                    projection={'close': True, 'adjfactor': True, '_id': False})
                if last_daily is not None:
                    profit_sum += (last_daily['close'] * last_daily['adjfactor'] - buy_close)/buy_close

        # 计算单期平均收益
        return (round(profit_sum * 100/count, 2), count)
```

后复权价格

答疑热线

牛小秘

微信：niuxiaomi1

普量学院
PLOUTO-QUANTS