

# 因子评估全流程详解

00

## 序

首先，这是一篇值得收藏的干货文。基本上覆盖到了因子评估的每个方面每个细节，小白友好型，很长，慢慢看。

关于因子评估，很早前写过三篇单因子测试的三篇文章：[上](#)、[中](#)、[下](#)，分别写了因子中性化、回归测试和分组测试，也可以参考这三篇。但是鉴于这篇写的过于早，现在再回过头来看，很多代码的效率非常低，有些说法非常不成熟，所以今天重新梳理一遍因子评估的全流程。











另一方面，小白入门最难的是找不到可以实操的例子，只看不练，其实也不太能学会，所以这篇也提供代码和数据，可以自己拿着练一练，相信会有很大的收获，也可以对比之前三篇的代码，找找差异，[获取方式见文末](#)。

因子评估，一般可以分为单因子的评估和多因子两部分，单因子是多因子的基础。一个新的因子要加入原有的模型，必须对这个因子做细致全面的分析，本文给出各种单因子分析的方法和理论。包括**因子收益**、**因子稳定性**、**因子行业表现**等等各方面。为了便于阅读和理解，每部分单独给出相应的数据和代码，以及图表结果。

01

## 数据说明

先对用到的数据做一个简单说明，本文以三个月动量（反转）因子为例，进行测试。因子定义为过去三个月的收益率，不做更多的处理，因子效果也不用太在意，本文的目的不在于找一个好因子，只是给出单因子评估全流程。数据总览如下

 mktcap	2020/3/25 11:58	Microsoft Excel ...	14,708 KB
 price	2020/3/25 11:58	Microsoft Excel ...	12,650 KB
 ST	2020/3/25 11:59	Microsoft Excel ...	27 KB
 股票上市日期	2020/3/25 12:23	Microsoft Excel ...	111 KB
 沪深300成分股	2020/3/25 12:03	Microsoft Excel ...	1,103 KB
 沪深300价格	2020/3/25 12:00	Microsoft Excel ...	129 KB
 中信行业代码表	2020/3/25 16:23	Microsoft Excel ...	1 KB
 中信一级行业	2020/3/25 12:02	Microsoft Excel ...	13,801 KB
 中证500成分股	2020/3/25 21:18	Microsoft Excel ...	1,612 KB
 中证500价格	2020/3/25 21:19	Microsoft Excel ...	107 KB

mtkcap是企业的市值，数据格式如下

tradedate	stockcode	mktcap
2000/1/28	000001. SZ	2870917. 12
2000/1/28	000002. SZ	575542. 0425
2000/1/28	000003. SZ	191724. 3108
2000/1/28	000004. SZ	112528. 7566
2000/1/28	000005. SZ	427364. 4375

price是股票的复权收盘价

tradedate	stockcode	price
2000/1/28	000001. SZ	400. 74
2000/1/28	000002. SZ	94. 9
2000/1/28	000003. SZ	22. 04

ST是股票ST记录，三列分别为股票代码、被ST日期和去除ST日期

stockcode	entry_dt	remove_dt
000511. SZ	1998/4/28	1999/4/28
000548. SZ	1998/4/29	1999/5/31
000558. SZ	1998/4/29	1999/5/26

股票上市日期格式如下

stockcode	stockname	ipo_date
000001. SZ	平安银行	1991/4/3
000002. SZ	万科A	1991/1/29
000004. SZ	国农科技	1991/1/14

沪深300成分股和中证500成分股为每个日期下的成分股列表，格式如下

stockcode	tradedate
000001. SZ	2005/4/29
000002. SZ	2005/4/29
000009. SZ	2005/4/29
000012. SZ	2005/4/29

沪深300价格和中证500价格为价格序列，格式如下

indexcode	tradedate	price
399300. SZ	2002/1/4	1316. 455
399300. SZ	2002/1/7	1302. 084
399300. SZ	2002/1/8	1292. 714
399300. SZ	2002/1/9	1272. 645

中信一级行业为给定日期下，股票所属的中信行业代码，行业代码的前四位表示一级行业

stockcode	classname	tradedate
600259. SH	b106020100	2000/5/31
600259. SH	b106020100	2000/6/30
600259. SH	b106020100	2000/7/31
600259. SH	b106020100	2000/8/31

中信行业代码表为行业代码和行业名称的对应关系

classname	classcode
汽车	b10d
非银行金融	b10m
房地产	b10n
银行	b10l
家电	b10g

在python中读入数据

```
454 # ===== 读数据 =====
455
456 mkt = pd.read_csv('mktcap.csv')
457 price = pd.read_csv('price.csv')
458 ST = pd.read_csv('ST.csv')
459
460 ipodate = pd.read_csv('股票上市日期.csv',encoding = 'gbk')
461 classcode = pd.read_csv('中信行业代码表.csv',encoding = 'gbk')
462 classname = pd.read_csv('中信一级行业.csv',encoding = 'gbk')
463
464 hs300 = pd.read_csv('沪深300成分股.csv',encoding = 'gbk')
465 zz500 = pd.read_csv('中证500成分股.csv',encoding = 'gbk')
466
467 price300 = pd.read_csv('沪深300价格.csv')
468 price500 = pd.read_csv('中证500价格.csv')
469
470
471 # 调整日期格式
472 ipodate['ipo_date'] = ipodate.ipo_date.apply(lambda x:pd.Timestamp(x).date())
473 ST['entry_dt'] = ST.entry_dt.apply(lambda x:pd.Timestamp(x).date())
474 ST['remove_dt'] = ST.remove_dt.apply(lambda x:pd.Timestamp(x).date())
475
476 mkt['tradedate'] = mkt.tradedate.apply(lambda x:pd.Timestamp(x).date())
477 price300['tradedate'] = price300.tradedate.apply(lambda x:pd.Timestamp(x).date())
478
479
480 price['tradedate'] = price.tradedate.apply(lambda x:pd.Timestamp(x).date())
481 classname['tradedate'] = classname.tradedate.apply(lambda x:pd.Timestamp(x).date())
482 hs300['tradedate'] = hs300.tradedate.apply(lambda x:pd.Timestamp(x).date())
483 zz500['tradedate'] = zz500.tradedate.apply(lambda x:pd.Timestamp(x).date())
484
485 classname['classcode'] = classname.classname.apply(lambda x:x[:4])
486 classname = classname.drop(['classname'],axis = 1)
487 classname = pd.merge(classname,classcode,left_on = ['classcode'],right_on = ['classcode'])
488 classname = classname.drop(['classcode'],axis = 1)
489
490
491 price500['tradedate'] = price500.tradedate.apply(lambda x:pd.Timestamp(x).date())
492 price300['tradedate'] = price300.tradedate.apply(lambda x:pd.Timestamp(x).date())
```

接下来进行各种因子分析。

02

因子定义和预处理

因子定义前文已经提到，三个月的动量（反转）因子，A股没有动量，都是反转。

```
# 计算三个月动量因子
f = price.pivot(index = 'tradedate',columns = 'stockcode',values = 'price').pct_change(3).fillna(0)
f = f.stack().reset_index()
f = f.rename(columns = {f.columns[2]: 'mom'})
```

因子的预处理对因子的效果有非常明显的影响，一般对因子的预处理包括缺失值填补、异常值处理等。

缺失值填补，一般填0或者填横截面均值，财务数据一般向前填充。动量因子没有缺失值，不涉及填补的问题。

异常值处理包括**异常样本的处理**和**离群值的处理**，异常样本包括新股、ST、PT等。比如新股可以定义为上市未满一年的股票，踢掉每一期的新股和PT、ST等股票，本文只踢ST股，有时也会考虑ST摘帽不满一年也踢掉。离群值为一些非常大或者非常小远超出正常范围的值，异常值产生的原因千奇百怪，这里不做总结，这里对离群值处理直接做**winsor**，即超出5%和95%分位数的点，拉到这两个分位点。

```
# 根据上市时间,剔除新股, 新股定义为上市未满一年
ipodate['entry_date'] = ipodate.ipo_date.apply(lambda x:x + datetime.timedelta(365))

f = pd.merge(f,ipodate,left_on = 'stockcode',right_on = 'stockcode',how = 'left')
f = f.loc[f.tradedate>=f.entry_date].reset_index(drop = True)
f = f.drop(['stockname','ipo_date','entry_date'],axis = 1)

# 根据ST列表, 剔除ST股, 剔除对应时间段为ST的股票

res =[]

for dates in f.tradedate.unique(): # dates = f.tradedate.unique()[10]
    fuse = f.loc[f.tradedate == dates]

    st_use = ST.loc[ST.entry_dt <= dates]
    st_use = ST.loc[ST.remove_dt > dates ]

    scode_notst = set(fuse.stockcode).difference(set(st_use.stockcode))

    fuse = fuse.set_index(['stockcode']).loc[scode_notst].reset_index()
    res.append(fuse)

res = pd.concat(res,axis = 0)

f = res.copy()
```

03

因子统计描述

因子统计描述包括因子的分布、因子自相关性、因子行业分布等。

因子分布

对因子按年划分绘制分布图（怎么划分根据需要，这里就看看），分别给出做winsor和不做winsor的结果，实现过程见函数plotFactor

```
def plotFactor(data,if_winsor):
    # data = allfactors.copy();savepath= output_picture
    """
    画因子分布图, hist, 先做winsor
    """

    fname = data.columns[2]

    data = f.copy()
    data['year'] = data.tradedate.apply(lambda x:x.year)
    data = data.fillna(0)

    if if_winsor:

        data = data.set_index(['tradedate','stockcode'])
        data = data.groupby('year').apply(lambda x:winsor(x[fname]) )
        data = data.reset_index()
    else:
        pass

    allyear = data.year.unique()
    n = len(allyear)
    ncol = int(n/3)+1

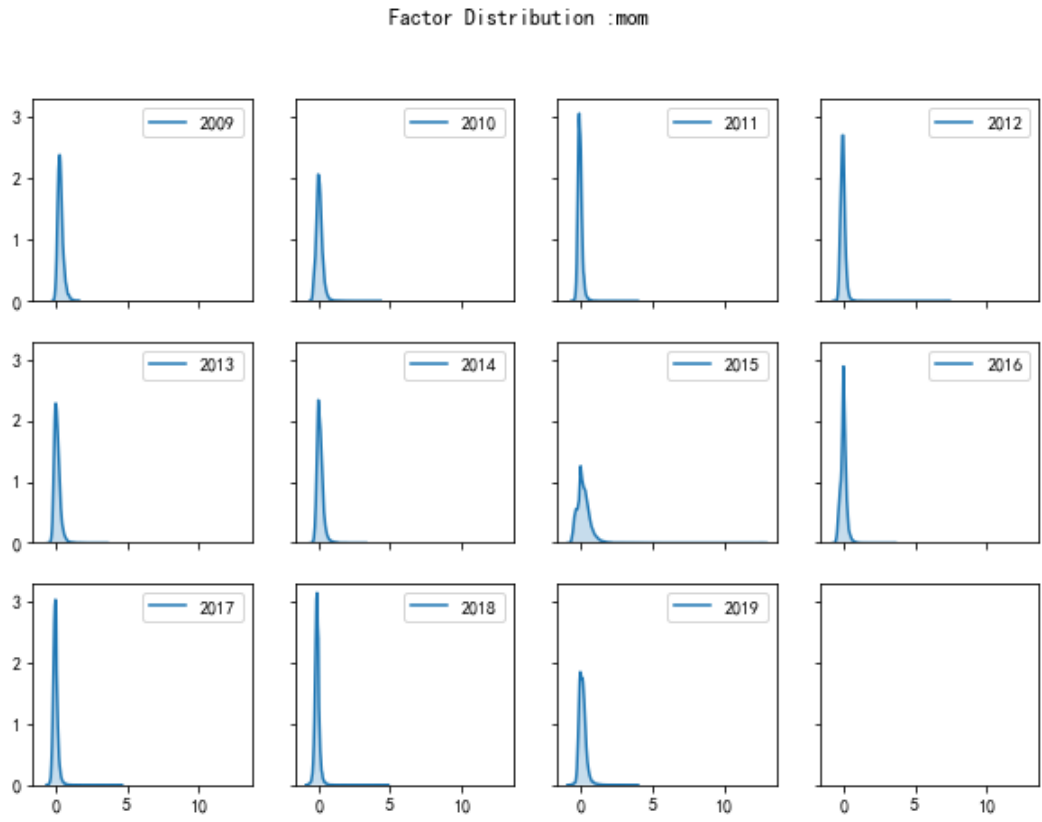
    fig,axes = plt.subplots(3, ncol, figsize=(10, 7), sharex=True, sharey=True)

    for i in range(n):

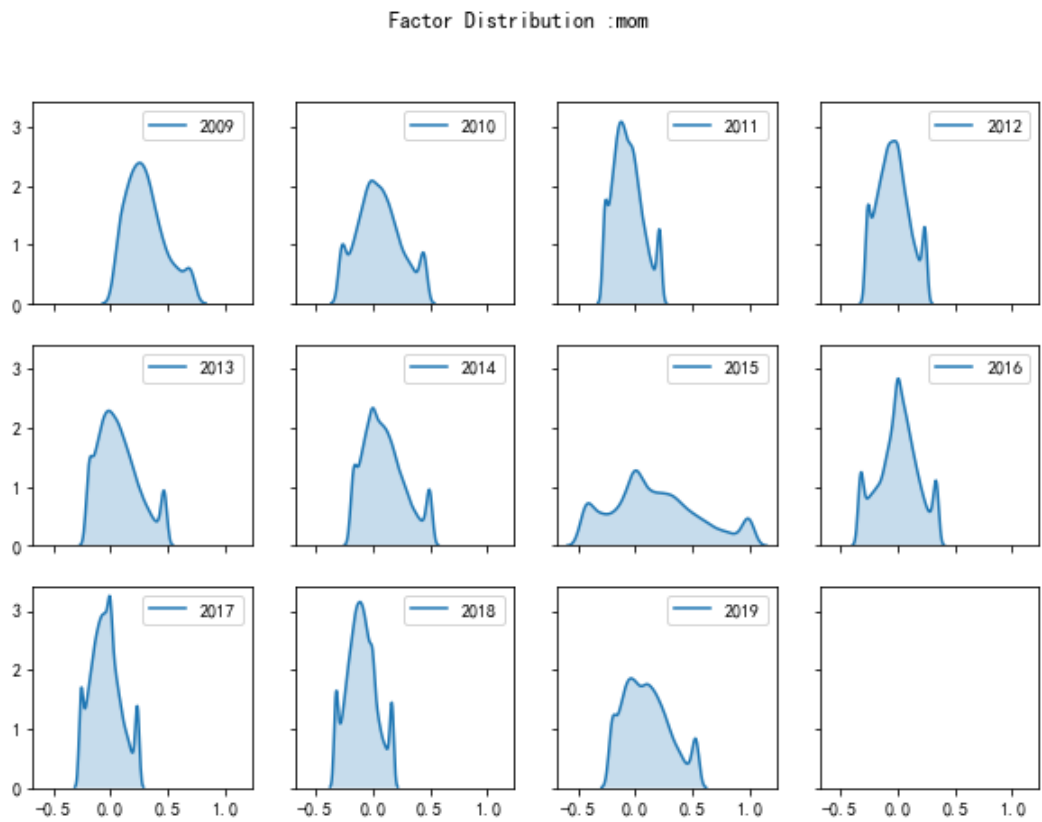
        sns.kdeplot(data.loc[data.year ==allyear[i],fname],shade=True,ax=axes[int(i/ncol),i%ncol],label = allyear[i])

    plt.suptitle('Factor Distribution :'+ fname)
```

不做winsor的结果



不做winsor的结果来看，因为有离群值，明显尖峰。做winsor的结果如下



做winsor的结果来看要正常很多，有些因子有季节效应，因子的分布图上会有明显的特征。

## 自相关分析

计算每一期因子和上一期的相关系数，实现过程见函数getFactorADF和plotADF

```
def getFactorADF(factors):
    # factors = fnormall;ret = retdata
    """
    计算因子自相关性
    """

    factors = factors.copy()
    factors = factors.set_index(['tradedate','stockcode'])
    fadf = []
    for i in factors.columns:
        s = factors.loc[:,i].unstack().T.corr()
        fadf.append(pd.DataFrame(np.diag(s,1),columns = [i],index = s.index[1:]))

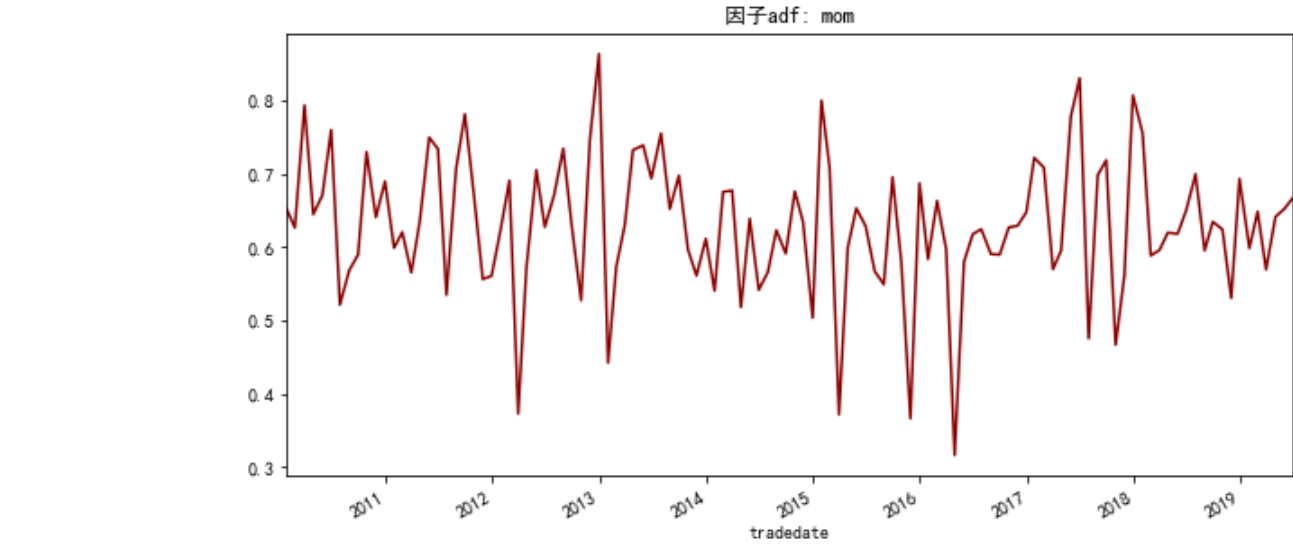
    fadf = pd.concat(fadf,axis = 1)
    return fadf

def plotADF(fadf):
    # outputpath = output_picture
    """
    因子ADF作图
    """

    for f in fadf.columns: # f = fadf.columns[0]
        fig = plt.figure(figsize = (10,5))
        fadf[f].plot(legend = False,color = 'darkred')
        plt.title('因子adf: ' + f)
```

这部分如果做的更细致，可以算滞后更多期的因子自相关系数，因子的自相关性主要是反映因子整体的稳定性，或者说因子的动量特征，如果因子的正自相关性很高，说明这个因子的持续性会很好，强者恒强，可以合理预期组合的换手会很低，负相关性很高则反之。当然这种方式的换手估计会很粗糙，之后会直接计算换手率。用上述两个函数计算自相关系数并作图结果如下





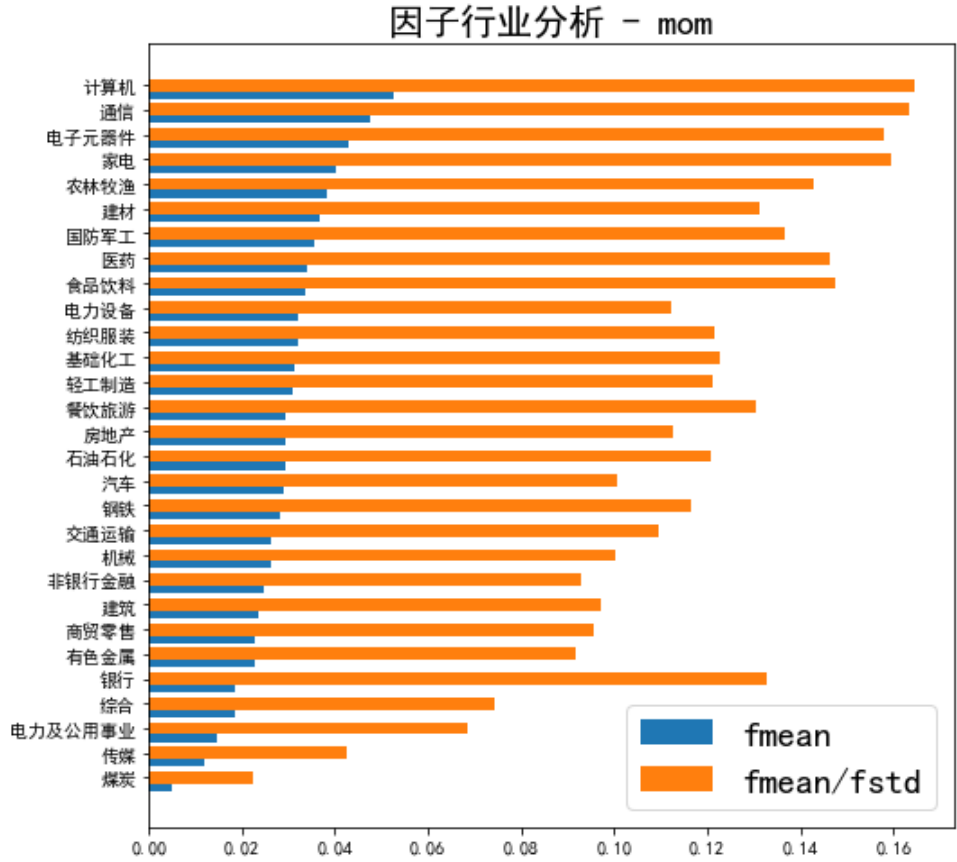
可以看出，因子的取值上，整体是呈现动量特征，即过去三个月跌的多的，下一个月三个月大概率也跌的多，这也符合常识。

## 因子的行业分析

因子的行业分析也非常重要，可以直观的看到哪些行业的因子暴露比较高，集中度比较高，更进一步可以预期构建的组合会倾向于哪些行业，从行业层面分析因子有效的逻辑。这部分直接算每个行业的因子暴露均值和因子暴露标准化均值，即均值除以标准差，这样可以反映整个行业因子暴露的一致性，即集中度。这部分通过函数plotIndf实现。

```
def plotIndf(f):  
  
    fmean = f.groupby('classname').mean()  
    fstd = f.groupby('classname').std()  
  
    fname = fmean.columns[-1]  
    fnorm = pd.Series(fmean[fname]/fstd[fname])  
  
    # 按照f大小进行排序  
    fmean = fmean[fname].sort_values()  
    fnorm = fnorm.loc[fmean.index]  
  
    plt.figure(figsize = (8,6))  
    plt.barh(y = np.arange(fmean.shape[0]),height = 0.5,tick_label = fmean.index,width = fmean,label = 'fmean')  
    plt.barh(y = np.arange(fnorm.shape[0]) + 0.3,height = 0.5,tick_label = fnorm.index,width = fnorm,label = 'fmean/fstd')  
    plt.title('因子行业分析 - ' + fname,fontsize = 20)  
    plt.legend(fontsize = 20)
```

调用函数绘制行业均值和标准化均值的柱状图如下



这张图包含了很多信息，蓝色为因子的行业均值，黄色线为标准化均值，结合因子的定义，平均来看，三个月平均涨幅最高的行业为计算机、通信、家电等等，同时稳定性也很高。三个月平均涨幅最低的行业包括煤炭、传媒、银行等等，并且可以看出，银行虽然平均涨幅很低，但标准化涨幅很高，说明这个行业涨幅的稳定性非常好，是非常低波动的行业。如果有一些行业研究的背景，可以对这些现象做一些更细致的分析。

上述都是因子自身的分析，当然除了行业，也可以从更多风格上分析因子的取值，比如看不同市值上的因子取值等等。接下来做一些因子和股票未来收益的分析，这部分也是大家最关注的部分，做之前首先做中性化处理。

### 04

#### 因子中性化

因子中性化主要是为了剔除因子在行业 and 市值上的特异性，让因子在行业 and 市值上可比。比如从上面的分析也可以明显看出，每个行业的因子大小和稳定性是有明显差异的，所以如果直接构建组合，结果会明显集中于部分行业。有时除了这两个之外，也会考虑更多因素，做一些风格中性化，或者说正交化。

中性化的原理为用因子值对市值和行业虚拟变量做回归，取回归的残差，因为回归残差和自变量之间是相互正交的。另外也可以证明，对行业虚拟变量回归去残差和在行业内减均值除以标准差的效果是一样的。

这部分通多函数norm和NormFactor实现

```
def olsResid(y,x):  
    df = pd.concat([y,x],axis = 1)  
  
    # print(df)  
    if df.dropna().shape[0]>0:  
        resid = sm.OLS(y,x,missing='drop').fit().resid  
        return resid.reindex(df.index)  
    else:  
        return y  
  
def norm(data,if_neutral):  
  
    data = data.copy()  
    """  
    数据预处理，标准化  
    """  
    # 判断有无缺失值，若有缺失值，drop，若都缺失，返回原值  
  
    datax = data.copy()  
    if data.shape[0] != 0:  
        classname = data['classname']
```

```
mkt = data['mktcap']
data = data.drop(['classname','mktcap'],axis = 1)

## 去极值
data = data.apply(lambda x:winsor(x),axis = 0)

## 中性化
if if_neutral: # 是否中性

    class_var = pd.get_dummies(classname,columns=['classname'],prefix='classname',
                                prefix_sep="_", dummy_na=False, drop_first=True)

    class_var['mktcap'] = np.log(mkt)
    class_var['Intercept'] = 1
    x = class_var
# 每个因子对所有自变量做回归，得到残差值
data = data.apply(func = OlsResid, args = (x,), axis = 0)

## zscore
data1 = (data - data.mean())/data.std()

# 缺失部分补进去
data1 = data1.reindex(datax.index)
else:
    data1 = data
return data1

"""
调用norm中性化所有因子
"""
def NormFactors(datas,if_neutral):
    # datas = factorall.copy()
    fnormal1 = []

    dates = datas.tradedate.unique()

    for dateuse in dates: # dateuse = dates[0]
        datause = datas.loc[datas.tradedate == dateuse]

        stockname = datause[['tradedate','stockcode']]
        fnorm = norm(datause.drop(['tradedate','stockcode'],axis = 1) ,if_neutral)
        fnormal1.append(pd.concat([stockname,fnorm],axis = 1))
        print('{}中性化完成! '.format(dateuse))
    fnormal1 = pd.concat(fnormal1,axis = 0)
    fnormal1 = fnormal1.sort_values(by = ['tradedate','stockcode'])
    return fnormal1.reset_index(drop = True)

def winsor(x):
    if x.dropna().shape[0] != 0:
        x.loc[x < np.percentile(x.dropna(),5)] = np.percentile(x.dropna(),5)
        x.loc[x > np.percentile(x.dropna(),95)] = np.percentile(x.dropna(),95)
    else:
        x = x.fillna(0)
    return x
```

这部分的结果没有图表呈现，当然也可以再去看看中性化后因子统计描述的结果，不做赘述。

05

因子IC分析

因子IC定义为因子值和股票未来一期收益率的相关系数，这里月频调仓的话也就是和未来一个月收益率的相关系数，也是很好理解的，如果相关性很高，并且时序上一直很高，即波动很小，说明因子预测能力很好，稳定性也很好，是一个比较有效的因子。

对于因子的IC值，一般比较关注的指标为因子IC的均值和ICIR，即IC均值除以IC标准差再年化。除此外，也会关注IC的衰减情况，半衰期，分层的IC，比如分行业、分市值的IC取值。有时也会关注IC的胜率，IC绝对值等等，也是为了分析因子的稳定性，本文略过。

IC和ICIR

IC和ICIR通过函数getICSeries和plotIC实现

```
def getICSeries(factors,ret,method):
    # method = 'spearman';factors = fnorm;

    icall = pd.DataFrame()
    fall = pd.merge(factors,ret,left_on = ['tradedate','stockcode'],right_on = ['tradedate','stockcode'])
    icall = fall.groupby('tradedate').apply(lambda x:x.corr(method = method)['ret']).reset_index()
    icall = icall.dropna().drop(['ret'],axis = 1).set_index('tradedate')

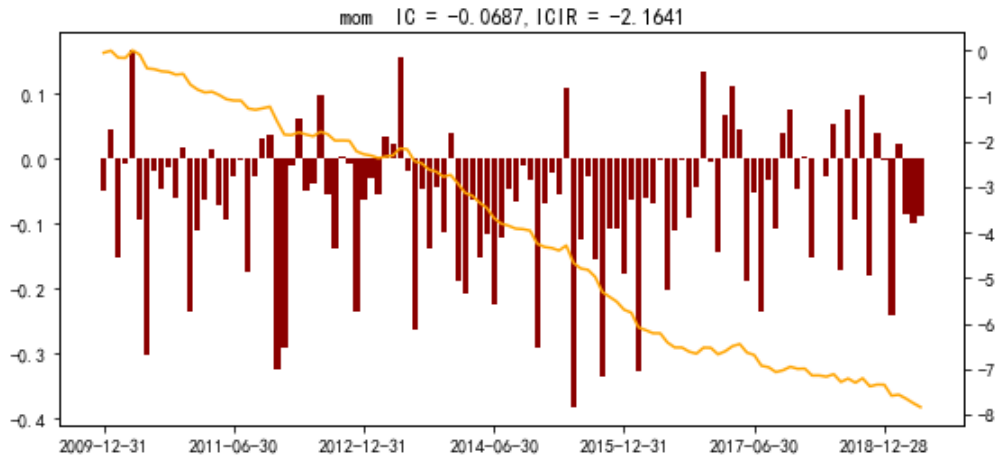
    return icall

def plotIC(ic_f):
    """
    IC作图
    """

    fname = ic_f.columns[0]

    fig = plt.figure(figsize = (9,4))
    ax = plt.axes()
    xtick = np.arange(0,ic_f.shape[0],18)
    xticklabel = pd.Series(ic_f.index[xtick])
    plt.bar(np.arange(ic_f.shape[0]),ic_f[fname],color = 'darkred')
    ax1 = plt.twinx()
    ax1.plot(np.arange(ic_f.shape[0]),ic_f.cumsum(),color = 'orange')
    ax.set_xticks(xtick)
    ax.set_xticklabels(xticklabel)
    plt.title(fname + ' IC = {},ICIR = {}'.format(round(ic_f[fname].mean(),4),round(ic_f[fname].mean()/ic_f[fname].std(),4)))
```

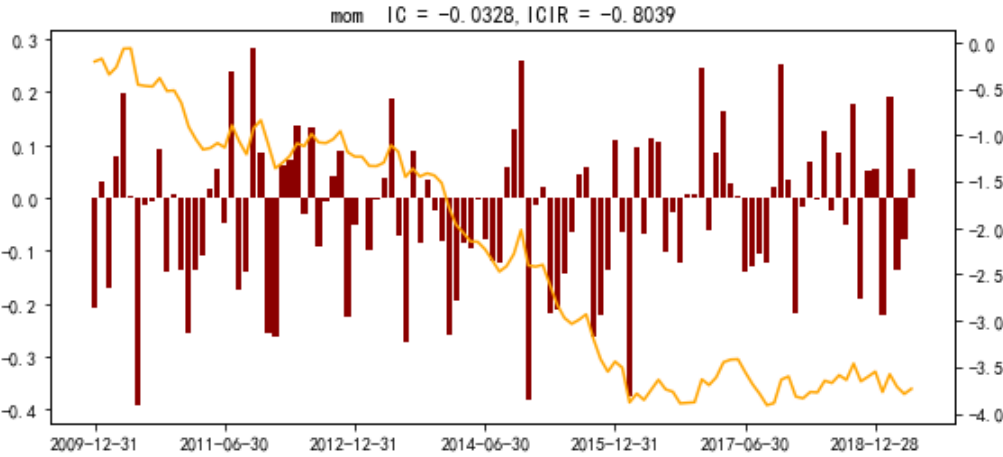
作图结果如下



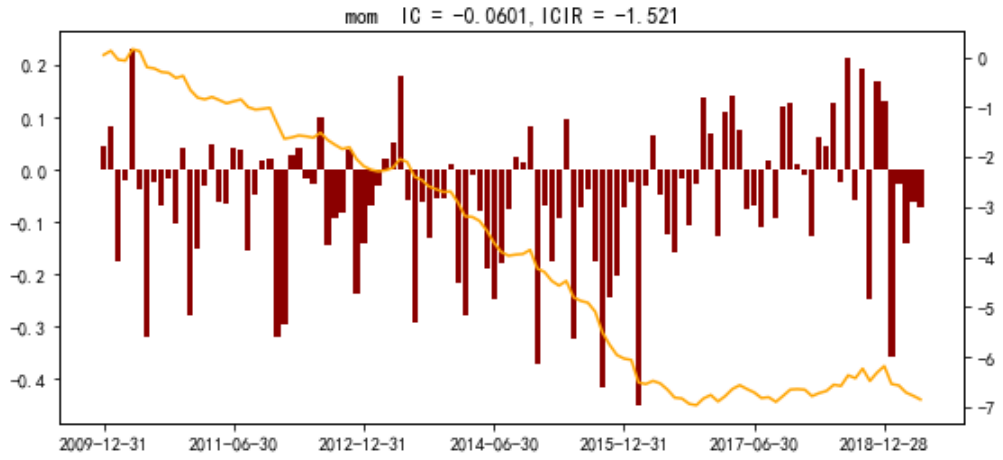
从结果来看，首先是明显的反转效应，IC = -6%，其次因子稳定性也很高，ICIR <-2，关于ICIR，可以类比T值来看。

除此外，如果要分析的更细致一些，也可以看看在各种指数成分股中的IC和ICIR，比如300、500、800、1000里，本文给出在沪深300和中证500成分股中IC、ICIR的结果如下。

300结果如下



稳定性明显要差一些，但IC仍有-3%，还是有效的，但近几年太平了，效果很差。500的结果如下



500的IC和ICIR都要更高一些，表现更好，但近几年也很平，也许是最近几年这个因子被用的太多了，谁知道呢。

## IC衰减

IC看的是相关性的高低，IC衰减看的是相关性的稳定性，如果衰减很慢，那么说明这个因子很稳定，可以做的比较长期一点，如果衰减很快，比较适合做短线。

因子衰减一般看IC的半衰期，首先计算因子值和未来一期、两期、三期等等的IC值(显然是越往后相关性越低的)，这里不累计，半衰期定义为IC值衰减到一半所用的时间。这部分通过函数getHalfValue、plotHalfIC和calhalfic实现。

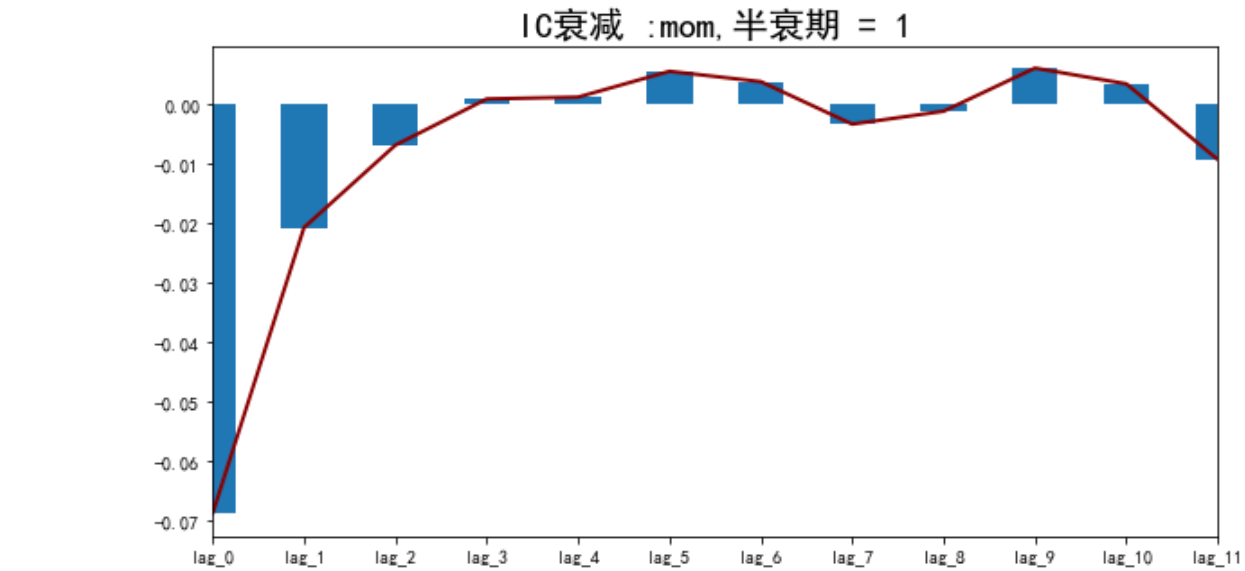
```
def getHalfValue(factors,method,ret):
    """
    计算因子半衰期
    """
    # factors = fnormal;ret = retdata
    halfic = []
    for i in range(12):
        ret1 = ret.pivot(index = 'tradedate',columns = 'stockcode',values = 'ret').shift(-i).stack()
        ret1 = ret1.rename(columns = {ret1.columns[-1]:'ret'})
        ic = getICSeries(factors,ret1,method)
        print('滞后{}期IC完成! '.format(i))
        halfic.append(pd.DataFrame(ic.mean(),columns = ['lag_' + str(i)]))
    halfic = pd.concat(halfic,axis = 1)

    halficvalue = halfic.apply(calhalfic,axis = 1)
    return halfic,halficvalue

def plotHalfIC(halfic):
    halficvalue = halfic.apply(calhalfic,axis = 1)
    for i in range(halfic.shape[0]): # i= 1
        fig = plt.figure(figsize = (10,5))
        halfic.iloc[i].plot(kind = 'bar')
        halfic.iloc[i].plot(kind = 'line',color = 'darkred',linewidth = 2)
        plt.title('IC衰减 :{},半衰期 = {}'.format(halfic.index[i],str(halficvalue[i]).strip('>')),font
```

```
def calhalfic(x):
    """
    计算因子半衰期的函数
    """
    target = abs(x[0]/2)
    position = np.where(x.abs() < target)
    if len(position[0]) >0:
        return position[0][0]
    else:
        return '>{}'.format(len(x))
```

结果如下图



半衰期是1，衰减很快，一般量价因子都衰减很快，财务类的比较慢。

## 分层IC

分层IC主要是看因子在行业、市值等其他风格上的分层表现，严格意义上已经不算单因子的分析了，算是多个因子的分析，如果要做的更细致，可以做[doublesor](#)和[fama-macbeth](#)回归，之前也写过，可以点开看看。本文只做按因子值分层和行业分组的IC。

按因子值的分层即安因子值大小等分为若干份，计算每一份的IC，不同因子取值下的IC值可能会有差异，也能看到很多信息。这部分通过函数[getGroupICSeries](#)实现

```
def getGroupICSeries(factors,ret,method,groups):
    # method = 'spearman';factors = fnorm.copy();ret = ret
    icall = pd.DataFrame()

    dates = factors.tradedate.unique()
    ret = ret.pivot(index = 'tradedate',columns = 'stockcode',values = 'ret')
    for dateuse in dates: # dateuse = dates[0]
        fic = pd.DataFrame()

        fdata = factors.loc[factors.tradedate == dateuse,factors.columns[1:]].set_index('stockcode')
        rt = ret.loc[dateuse]
        for f in fdata.columns: # f = fdata.columns[0]

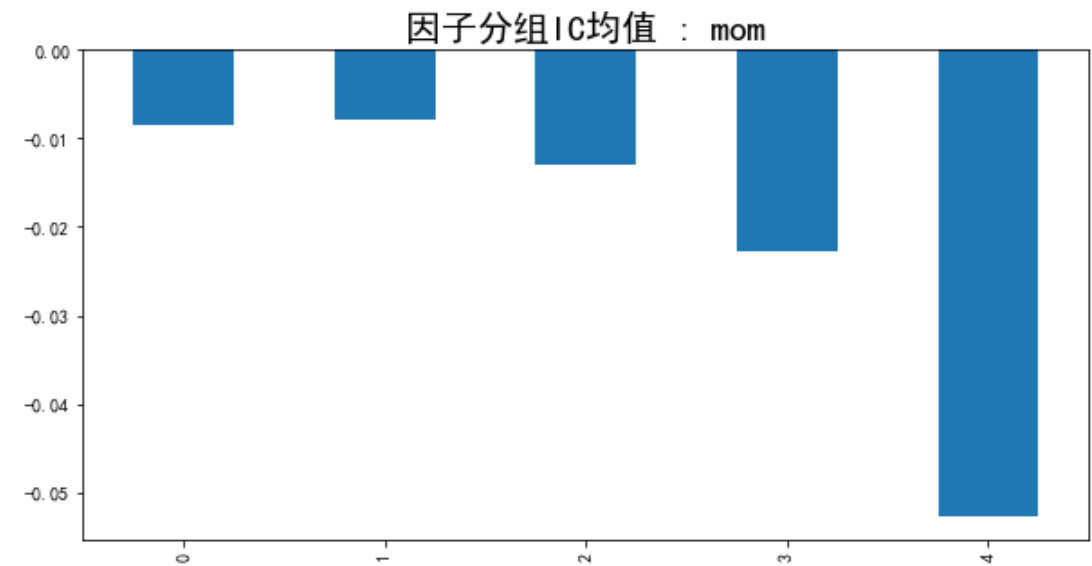
            IC = getGroupIC(fdata[f],rt,method,groups)
            IC.insert(0,'factor',f)

        fic = pd.concat([fic,IC],axis = 0)
        icall = pd.concat([icall,fic],axis = 0)

    return icall

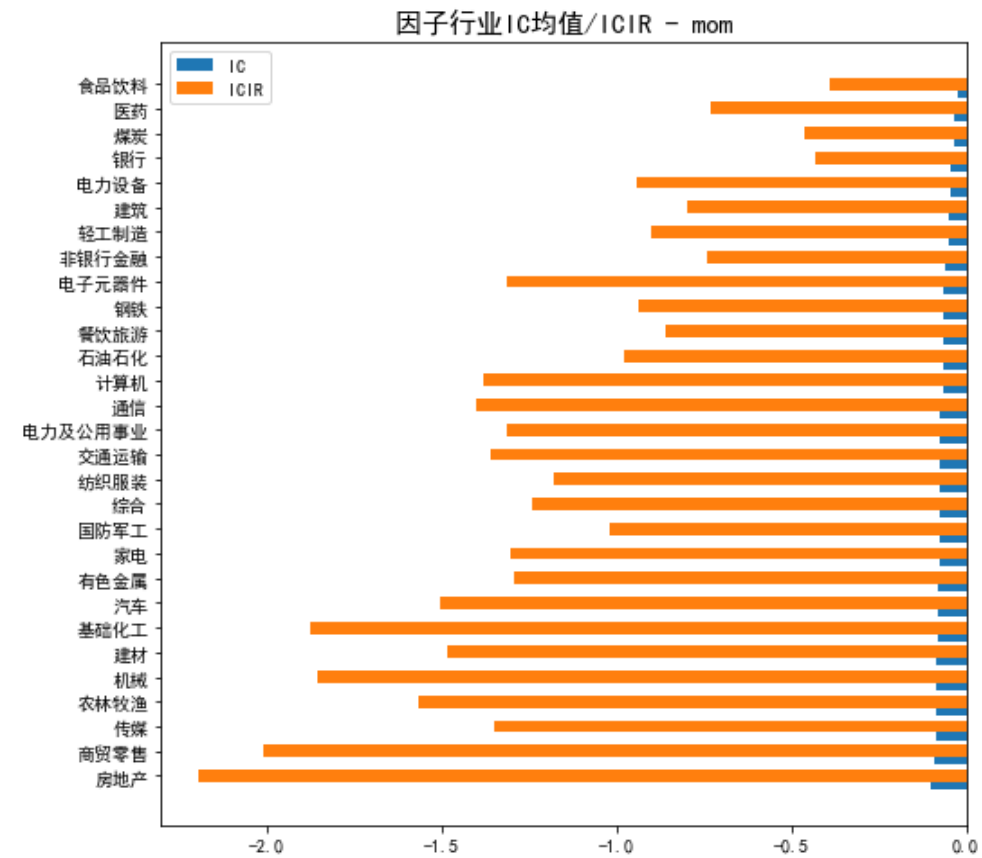
def plotGroupIC(groupIC):
    """
    分组IC作图
    """
    for f in groupIC.factor.unique():
        fig = plt.figure(figsize = (10,5))
        groupIC.loc[groupIC.factor == f,groupIC.columns[1:]].mean(axis =0).plot(kind = 'bar')
        plt.title('因子分组IC均值 : ' + f,fontsize = 20)
```

等分五份结果如下



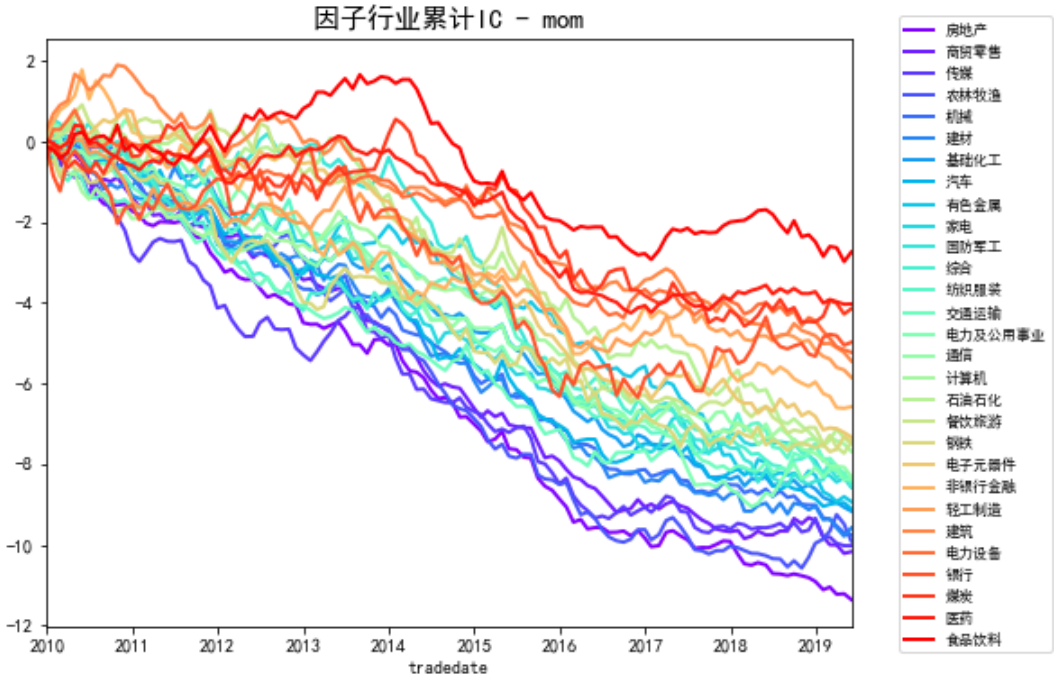
从0到1，因子值逐渐增大，可以明显看出，因子值越大，因子的IC绝对值越高，即相关性越高。仔细想想，这并不是一个好的现象，因为这是一个负向的因子，所以构建组合多头选的是取值最小的那部分股票，但这部分的IC反而是最小的，而相关性最高的这部分实际上是空头部分，A股不能做空，所以是拿不到的。很多量价因子都会有这样的现象，收益集中在空头。

接下来看分行业的IC情况，这部分的意义和前面分行业因子值差不多，可以分析这个因子在哪些行业上相关性更高，毕竟单纯的因子暴露高没有用，还要和未来收益率高才可能有效。代码和之前的差不多，不放了，需要自己取附件，计算每个行业上的IC和ICIR如下





这里可以和前面因子行业取值做一个交叉分析，有的行业虽然因子值很高，但是相关性很低，有的相关性很高，但是取值不在头部，也有一些很有意思的现象，当然这里因为已经做过了中性化，实际上可以认为在每个行业上的暴露是差不多的。下面是一张每个行业的累计IC曲线，看看就好。



做完了IC，接下来算因子的收益率，对于因子收益率的计算，一般有两种算法，一种是回归法，算回归的beta，Barra最开始采用这种办法，另一种是模拟组合法，也就是常说的分层测试，——说明。

06

因子收益率—回归法

回归法计算收益率的逻辑是，用因子未来一期的收益率和当期的因子暴露做OLS回归，取回归的beta作为因子收益率的估计值，这种方法可以一次回归一个因子，也可以一次把多个因子放在一起进行回归。本文只有一个因子进行回归。通过函数getFretSeries实现，这里两种回归方式的都写了，通过参数ftype进行控制。

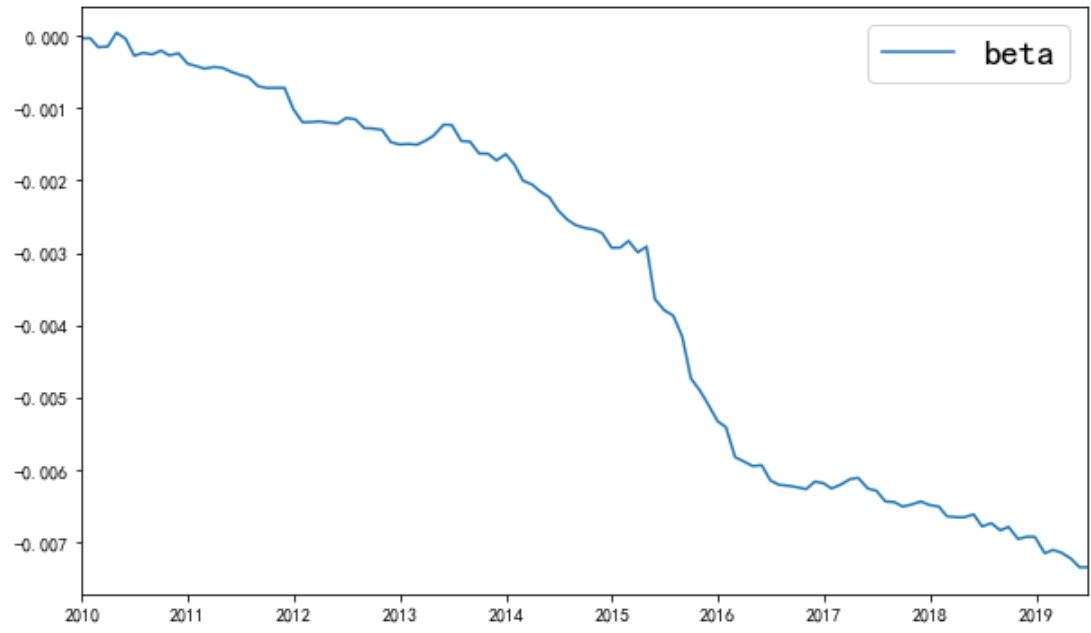
```
def getFretSeries(factors,ret,ftype):
    # method = 'spearman'
    fretall = pd.DataFrame()
    dates = factors.tradedate.unique()
    ret = ret.pivot(index = 'tradedate',columns = 'stockcode',values = 'ret')
    for dateuse in dates:

        fdata = factors.loc[factors.tradedate == dateuse,factors.columns[1:]].set_index('stockcode')

        rt = ret.loc[dateuse]
        if rt.dropna().shape[0] >0:
            fret = getFret(fdata,rt,ftype)
            fretall = pd.concat([fretall,fret],axis = 0)

#    fretall = fretall.reset_index(drop = True)
    return fretall
```

从OLS的beta表达式上，可以预期一个因子回归结果和IC的差异不会太大，这里做出因子收益率的累计曲线如下



如果你想更细，可以再做一个300、500、800里的结果出来看看。

最后要说明的一点是，回归法有一个很明显的问题，估计出来因子收益率的值的大小，没有什么实际的含义，因子的预处理方法上稍微有一点差异，最后回归出来的结果值都会差很多，接下来给出模拟组合法。

07

因子收益率-模拟组合法

模拟组合法，顾名思义，根据因子值构建投资组合，用投资组合的实际收益率，作为因子收益率的估计值。这里一般使用纯多头或者多空的收益率作为因子收益率的估计值，其实和常用的分层测试法差不多。本文首先给出分层测试+多空的結果，再给出TopN多头的结果。

分层+多空

每期按照因子的值将股票等分为5份或10份，计算每一份的投资收益，看收益曲线，好的因子应该会有区分明显、并且单调分层的曲线。多空即为最大组 - 最小组，本文因子是一个负向的因子，所以多空为最小组 - 最大组。这部分通过代码GroupTestAllFactors实现。



```
def GroupTestAllFactors(factors,ret,groups):
    # factors = fnorm.copy();groups = 10
    """
    一次性测试多个因子
    """
    fnames = factors.columns
    fall = pd.merge(factors,ret,left_on = ['stockcode','tradedate'],right_on = ['stockcode','tradedate'])
    Groupret = []
    Groupturnover = []
    for f in fnames: # f= fnames[2]
        if ((f != 'stockcode')&(f != 'tradedate')):
            fuse = fall[['stockcode','tradedate','ret',f]]
            fuse['groups'] = pd.qcut(fuse[f],q = groups,labels = False)
            fuse['groups'] = fuse[f].groupby(fuse.tradedate).apply(lambda x:np.ceil(x.rank()/(len(x)/groups)))
            result = fuse.groupby(['tradedate','groups']).apply(lambda x:x.ret.mean())
            result = result.unstack().reset_index()
            if result.iloc[:,-1].mean() > result.iloc[:,-groups].mean():
                result['L-S'] = result.iloc[:,-1] - result.iloc[:,-groups]
                stock_l = fuse.loc[fuse.groups == 1]
            else:
                result['L-S'] = result.iloc[:,-groups] - result.iloc[:,-1]
                stock_l = fuse.loc[fuse.groups == groups]

            result.insert(0,'factor',f)
            Groupret.append(result)

            turnovers = getturnover(stock_l)
            turnovers.columns = [f]
            Groupturnover.append(turnovers)

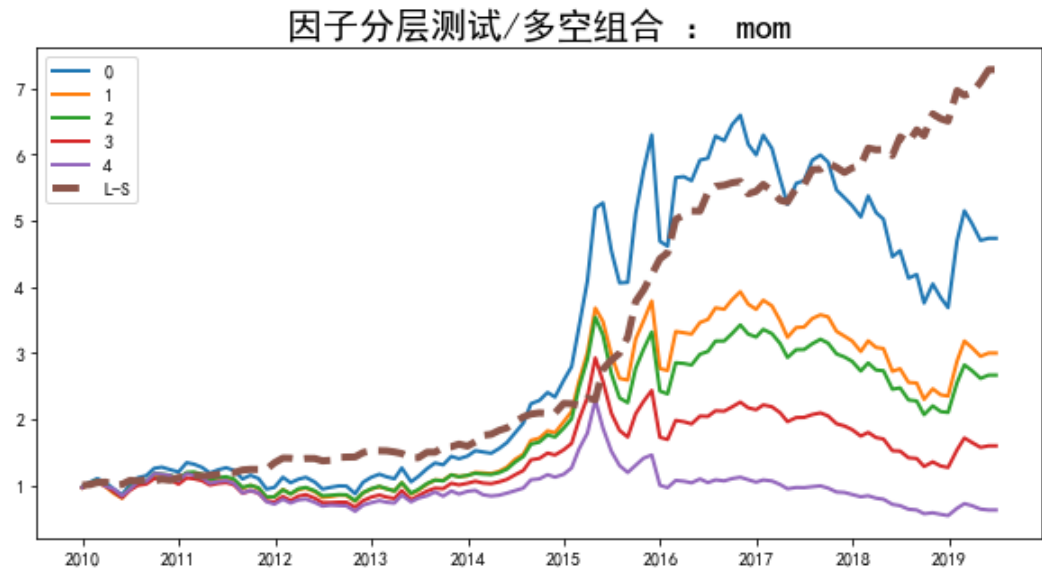
    Groupret = pd.concat(Groupret,axis = 0).reset_index(drop = True)

    Groupnav = Groupret.iloc[:,2:].groupby(Groupret.factor).apply(lambda x:(1 + x).cumprod())
    Groupnav = pd.concat([Groupret[['tradedate','factor']],Groupnav],axis = 1)

    Groupturnover = pd.concat(Groupturnover,axis = 1).reset_index()

    return Groupnav,Groupturnover
```

分层测试结果如下



这里L-S为多头减去空头，即多空的结果，分组从0到4，因子值逐渐增大，可以看出，是明显单调的，并且0组表现最好。关于换手率，在下部分说明。

除此外，也可以在300，500，800内做分层。

## TopN组合

取暴露最大（小）的前N只股票构建组合，评估组合的表现。通过函数**PortfolioTopN**实现。

对于组合表现的评估上，本文除了给出净值曲线，也给出每个组合每期的换手率情况，换手率越高，调仓成本会越高，换手率定义好像没有统一的，**这里用每期和上一期比的调出股票占上一期股票的比例定义**。也有用这一期股票做分母的。换手率通过函数**getturnover**实现。

此外，给出分年统计的效果，如果因子不是每年都表现很好，那说明因子稳定性不是很好。分年统计通过函数**performance**实现。

```
def portfolioTopN(factors,rt,benchmark,N,if_max = True):
    # rt = ret1.copy();benchmark = price300.copy();factors = fnormall.copy()

    if if_max:
        factors = factors.sort_values(by = factors.columns[2],ascending = 0)
        stocklist = factors.groupby('tradedate').head(N).reset_index(drop = True)
    else:
        factors = factors.sort_values(by = factors.columns[2],ascending = 1)
        stocklist = factors.groupby('tradedate').head(N).reset_index()
    stockret = pd.merge(stocklist,rt,left_on = ['tradedate','stockcode'],right_on = ['tradedate','stockcode'])
    stockret = stockret.sort_values(by = ['tradedate','stockcode'])

    result = stockret.ret.groupby(stockret.tradedate).mean()
    result = pd.DataFrame(result).shift(1).fillna(0)
    result.columns = ['Portfolio']
    result['Portfolio'] = (result.Portfolio+1).cumprod()

    benchmark['ym'] = pd.Series(benchmark.tradedate).apply(lambda x:x.year*100 + x.month)
    benchmark = benchmark.groupby('ym').last()
    benchmark = benchmark.set_index('tradedate')[['price']]
    benchmark = benchmark.reindex(result.index)
    benchmark['nav'] = benchmark['price']/benchmark.price[0]

    result['benchmark'] = benchmark.nav.values
    result['RS'] = result.Portfolio/result.benchmark

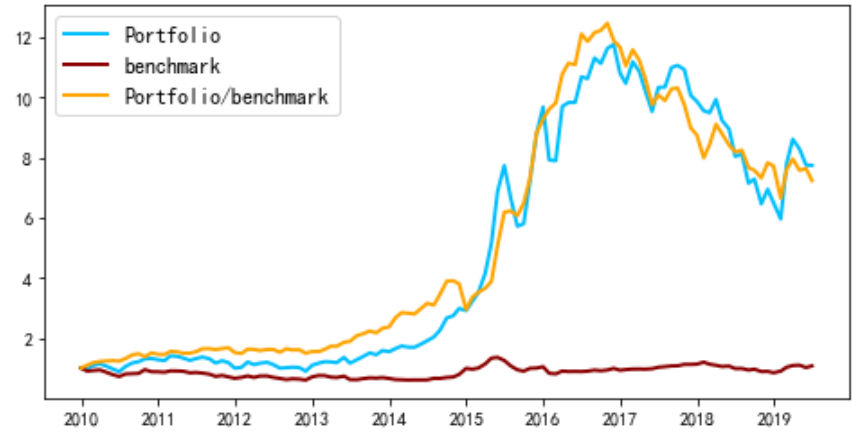
    plt.figure(figsize = (8,4))
    plt.plot(result.Portfolio,linewidth = 2,c = 'deepskyblue',label = 'Portfolio')
    plt.plot(result.benchmark,linewidth = 2,c = 'darkred',label = 'benchmark')
    plt.plot(result.RS,linewidth = 2,c = 'orange',label = 'Portfolio/benchmark')
    plt.legend(fontsize = 12)

    return stockret.reset_index(drop = True),result

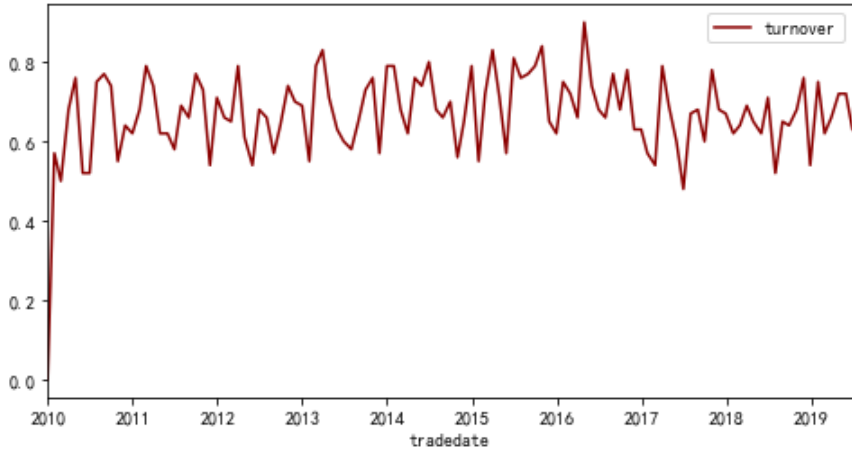
# 分年统计
def performance(stocklist,result):
    # result = res_300
    tunover1 = getturnover(stocklist)
    result['y'] = pd.Series(result.index).apply(lambda x:x.year).values
    result_y = result.groupby('y').last()/result.groupby('y').first()-1
    result_y['超额收益'] = result_y.iloc[:,0] - result_y.iloc[:,1]
    result_y = result_y.T
```

return tunover1,result\_y

取100只股票，在全A股中的结果如下，基准为沪深300。



换手率如下



分年表现如下

Index	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019
Portfolio	0	0.301883	-0.205105	0.0671446	0.327699	0.766511	2.00361	0.364161	-0.0571718	-0.32036	0.299521
benchmark	0	-0.0236861	-0.237531	0.0238173	-0.132814	0.604443	0.0863661	0.12355	0.189758	-0.295902	0.194886
RS	0	0.333468	0.0425274	0.0423193	0.531043	0.101012	1.76483	0.214152	-0.207547	-0.0347378	0.0075691
超额收益	0	0.325569	0.0324258	0.0433273	0.460513	0.162068	1.91725	0.240611	-0.24693	-0.0244588	0.104635

可以看出，相比于基准，除2017、2018年外，都有正的超额收益。源码里还给出了在300、500中的结果，限于篇幅，这里不写了。

08

还可以做什么

以上，对于小白，如果都能搞清楚原理，并且实现一遍，找一个金工实习不难的。当然也可以想想，还可以做什么？对于多因子，可以做的事情很多了，这里还是只说对于单因子，除了这些，还能做些什么？

- 1、除了TopN组合，还可以构建行业TopN组合，强制在每个行业取等量股票，看效果。
- 2、改变调仓日期、调仓频率看效果，月频换周频、季度频，调仓从月末改成月初等等。
- 3、关于组合评估，可以计算更多的指标，比如Sharp、最大回撤、索提诺比率比率，组合收益率计算上，可以对比等权 and 市值加权等等
- 4、最重要的一点，要结合以上的结果，从逻辑上分析，为什么这个因子是有效的，收益的来源是什么？

这些基本都写过，工作量不大，有兴趣的童鞋开源再加上去。

- End -

最后，最重要的，获取源码和数据，请加入知识星球



量化小白

星主：量化小白



 知识星球

微信扫码预览星球详情

如果对代码、数据或全文任何步骤有问题，也欢迎在星球中向我提问。



欢迎加入微信群，一起交流量化，分享信息

量化小白1群(193)

量化只是方法

量化也有花式亏钱法哈哈

你一定非要让方法带来金钱，这个逻辑本来就是不成立的

星期六 下午9:11

国内量化工作坑位少吧🤔

🤔研究不就是为了赚钱吗

量化小白1群(193)

话说群里有工作搞量化的吗？感觉量化工作不好找

@仙哥要认真 非常不好找

星期六 下午9:01

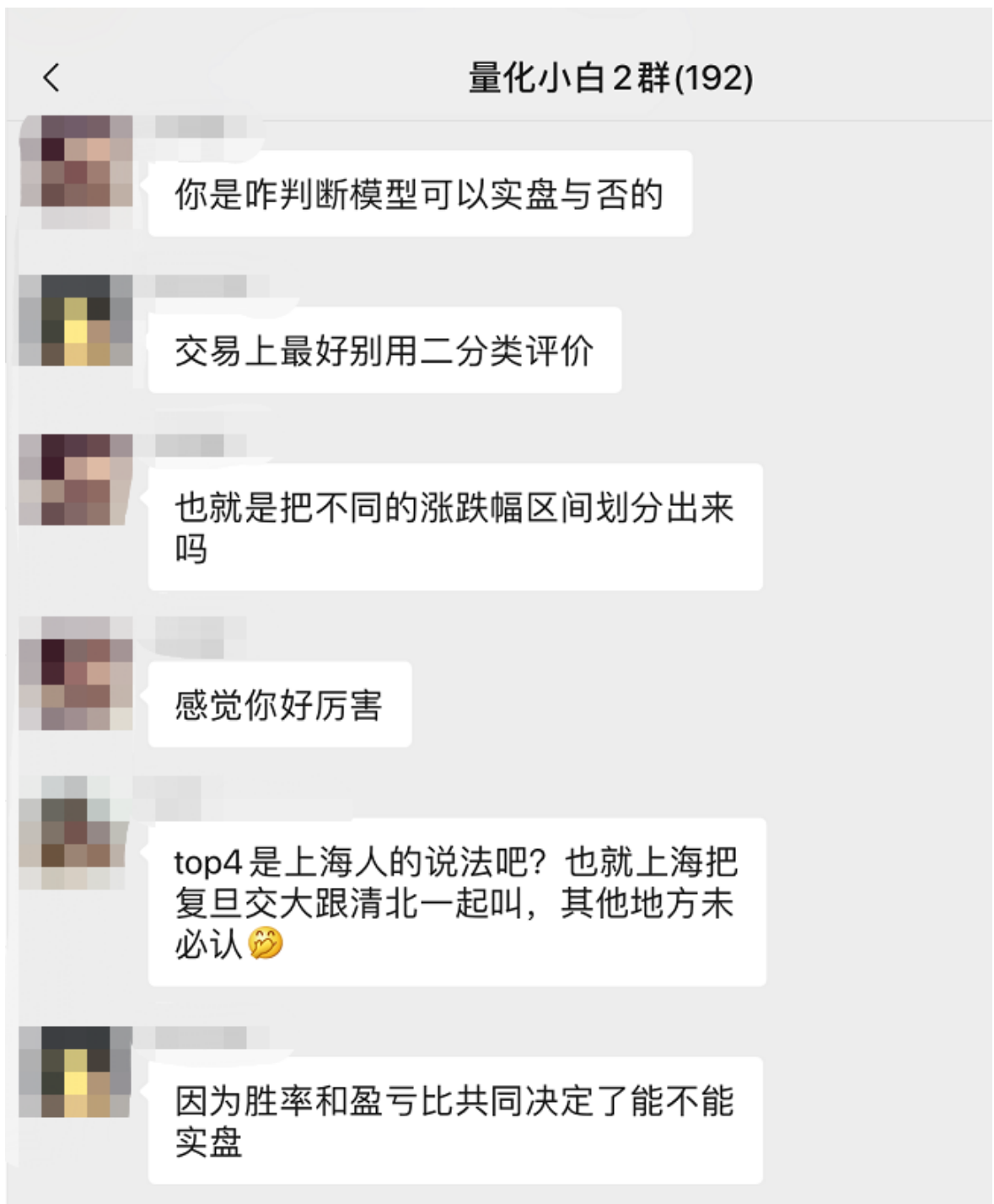
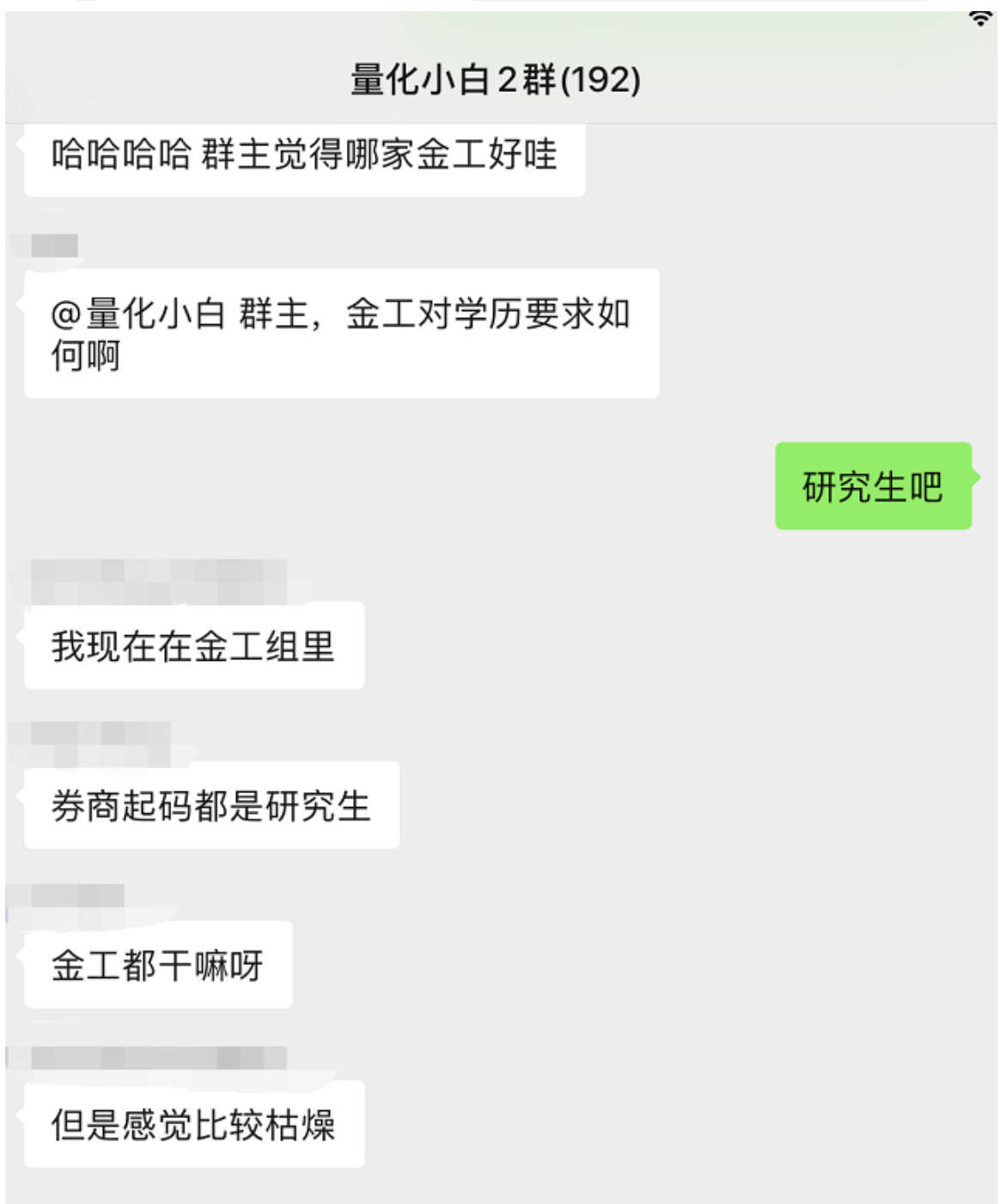
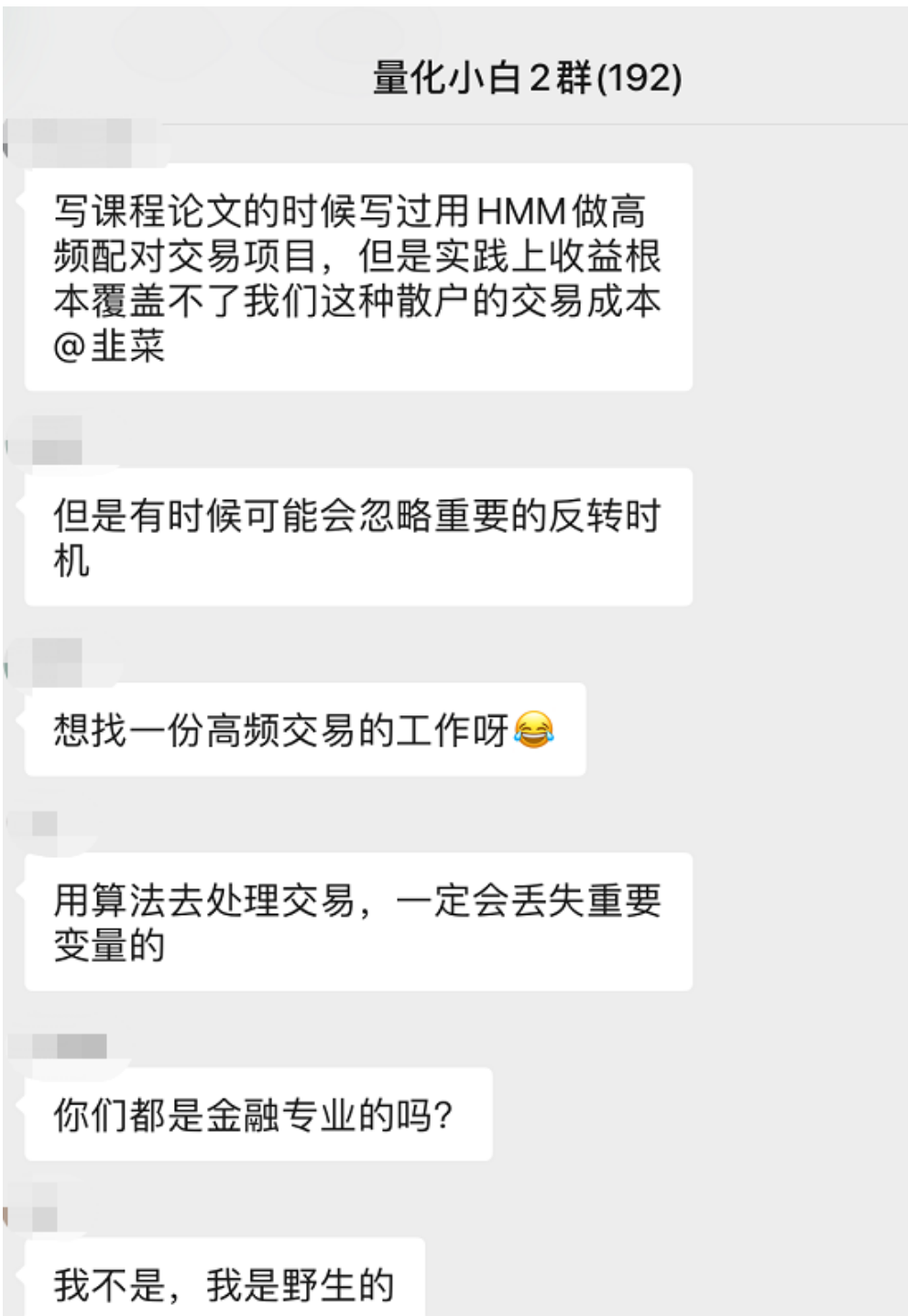
搞量化都是闷声发大财的吗

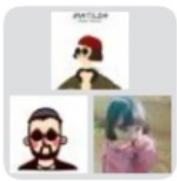
或者默默穷着吧

不过讲道理，量化岗位普遍还是招人的

猜想是上亿的资金肯定是找大牛







量化小白 3群



该二维码 7 天内 (4月2日前) 有效，重新进入将更新

如果人数已满，后台回复“微信群”获取最新群聊二维码

觉得不错就点个赞吧????