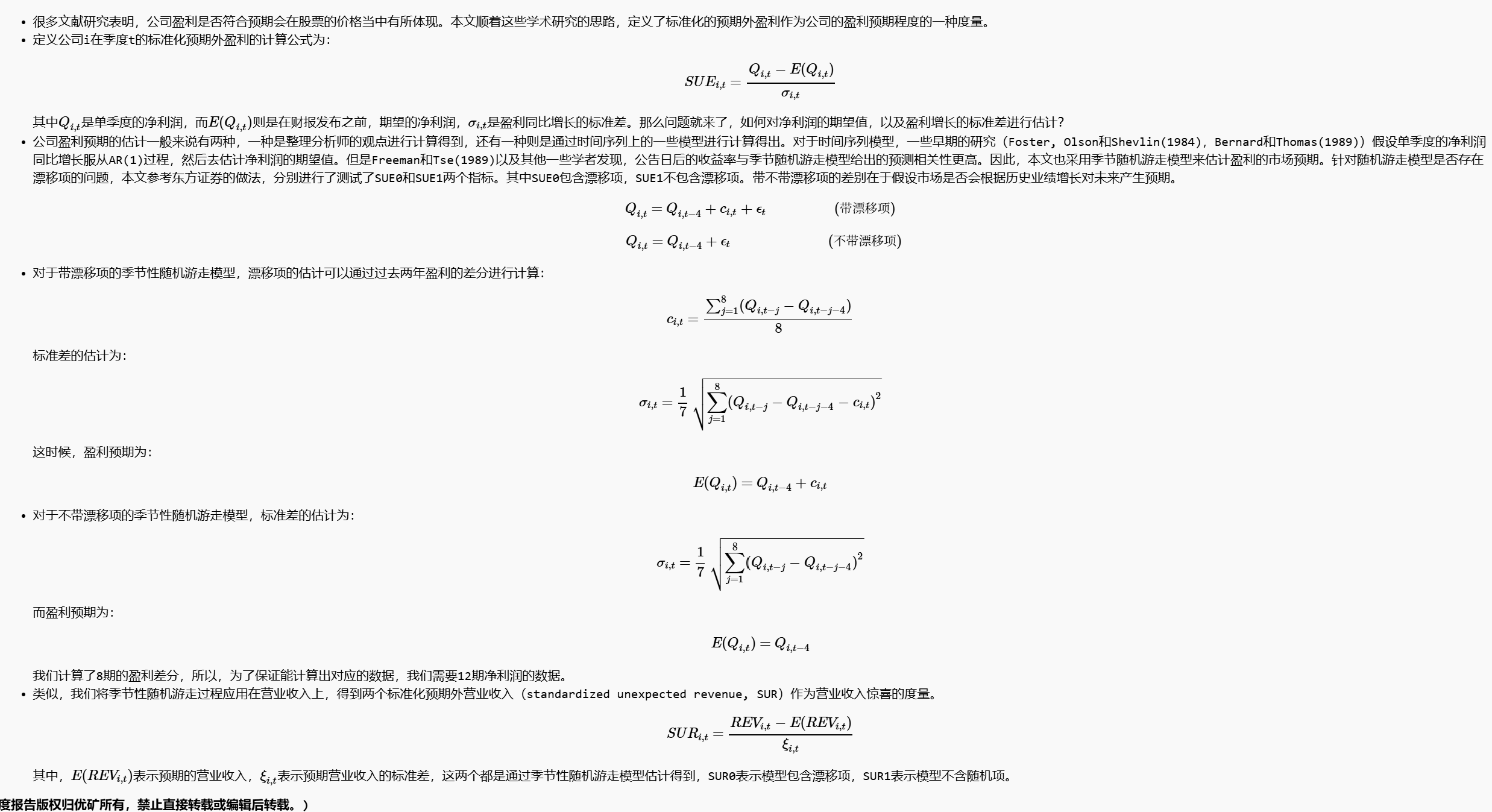
* 研究目的：
  + 1968年，Ball和Brown在美国市场首次发现了盈余公告后的价格偏移现象，即对于盈利超预期（Positive Eanings Surprise）的股票在公告后有持续正向的异常收益，相反，盈利低于预期的股票后期有持续负向的异常收益。自Ball 和Brown(1968)之后，大量研究学者在不同的国家、采用不同的研究或度量方法均发现了类似的结论，早期的研究主要集中于对净利润的研究，Ertimur 等（2003）和Jegadeesh 等（2006）的研究则将关注点从净利润转移到营业收入上，结果表明营业收入的预期外部分在公告后也有显著的异常收益，而且这部分异常收益并不能被盈利的预期外部分解释，说明营业收入相对于盈利有额外的信息。
  + 本文结合优矿底层提供的相关财务数据，参考了东方证券《业绩超预期类因子——因子选股系列研究之三十九》，计算了业绩惊喜相关因子，并对相关因子进行了测试。
* 研究结论：
  + 本文对业绩惊喜相关因子数据进行了测试，从单因子分析的结果来看，财报公告后，业绩超预期的股票有着显著的正向超额收益，而不及预期的股票有着负向的收益。
  + 同时，从IC均值，多空收益来看，基于净利润的业绩惊喜因子要好于基于营业收入的业绩惊喜因子。
  + 四个因子的原始值有着明显的选股效果，但中性化后的因子选股稳定性更高，预测能力也更强。
  + 本文对业绩惊喜因子进行了信息增量分析，研究发现，剔除了其他大类因子后，业绩惊喜因子仍具有不错的选股能力，而成长因子在剥离了业绩惊喜因子后，几乎没有选股效果，该因子可以作为成长因子的替代
* 文章结构：
  + 数据及工具函数的准备：该部分主要是一些工具函数的编写，包括交易日历的获取，因子的计算，因子分析的代码以及分析报告的绘图展示等等。（45分钟）
  + 业绩惊喜因子介绍：介绍了本文4个业绩惊喜因子的计算逻辑和计算方式
  + 单因子有效性分析：从因子的角度来分析业绩惊喜因子能否带来超额收益 （<5分钟）
  + 信息增量分析：分析业绩惊喜因子与大类因子之间的相关性，与成长因子的替代性 （<5分钟）
  + 指数增强策略：比较了使用业绩惊喜因子与成长因子的指数增强策略的业绩表现 (10分钟)
  + 总结：根据分析的结果，对业绩惊喜因子进行总结



import pandas as pd

import numpy as np

import scipy.stats as st

import gevent

import pandas as pd

import numpy as np

import os

import time

import multiprocessing

import copy

import seaborn as sns

import matplotlib.pyplot as plt

from datetime import datetime

from multiprocessing import Pool

from multiprocessing.dummy import Pool as ThreadPool

def get\_dates(start\_date, end\_date, frequency='daily'):

"""

输入起始日期和频率，即可获得日期列表（daily包括起始日，其余的都是位于起始日中间的）

输入：

start\_date，开始日期，'YYYYMMDD'形式

end\_date，截止日期，'YYYYMMDD'形式

frequency，频率，daily为所有交易日，weekly为每周最后一个交易日，monthly为每月最后一个交易日，quarterly为每季最后一个交易日

返回：

获得list型日期列表，以'YYYYMMDD'形式存储

"""

data = DataAPI.TradeCalGet(exchangeCD=u"XSHG", beginDate=start\_date, endDate=end\_date,

field=u"calendarDate,isOpen,isWeekEnd,isMonthEnd,isQuarterEnd", pandas="1")

if frequency == 'daily':

data = data[data['isOpen'] == 1]

elif frequency == 'weekly':

data = data[data['isWeekEnd'] == 1]

elif frequency == 'monthly':

data = data[data['isMonthEnd'] == 1]

elif frequency == 'quarterly':

data = data[data['isQuarterEnd'] == 1]

else:

raise ValueError('调仓频率必须为daily/daily1/weekly/weekly2/monthly/quarterly！！！')

date\_list = map(lambda x: x[0:4] + x[5:7] + x[8:10], data['calendarDate'].values.tolist())

return date\_list

def shift\_date(date, n, direction='back'):

"""

日期平移函数，获取date向前/向后移动Ndays个交易日所对应的交易日

输入：

date: str， 'YYYYMMDD'各式

n: int，长度不要超过700

direction:str, 可选'back'或者'forward'

返回：

date：str，'YYYYMMDD'格式

"""

last\_two\_year = str(int(date[:4]) - 2) + '0101'

forward\_two\_year = str(int(date[:4]) + 2) + '1231'

if direction == 'back':

date\_list = get\_dates(last\_two\_year, date, 'daily')

return date\_list[len(date\_list) - 1 - n]

elif direction == 'forward':

date\_list = get\_dates(date, forward\_two\_year, 'daily')

if len(date\_list) <= n: # 当未来数据长度不够用时，返回最后一个能取到的交易日

return date\_list[-1]

else:

return date\_list[n]

else:

raise ValueError('direction should be back/forward！！！')

def get\_st\_tickers(date):

"""

获取历史上某一天的ST股票

输入:

date: str, 'YYYYMMDD'格式

返回：

list: 元素为股票ticker

"""

data = DataAPI.SecSTGet(beginDate=date, endDate=date, field='')

return data['secID'].tolist()

def get\_all\_tickers(date):

"""

给定日期，获取这一天上市时间不低于60天的股票（参照中证全指指数编制）

输入：

date: str， 'YYYYMMDD'格式

返回：

list： 元素为股票ticker

"""

universe = DataAPI.EquGet(equTypeCD=u"A", listStatusCD="L,S,DE", field=u"secID,listDate,delistDate")

universe['listdate'] = universe['listDate'].apply(lambda x: x.replace('-', ''))

universe['delistdate'] = universe['delistDate'].apply(

lambda x: x.replace('-', '') if isinstance(x, unicode) else '99999999')

list\_d\_need = shift\_date(date, 60, 'back')

universe = universe[(universe['listdate'] <= list\_d\_need) & (universe['delistdate'] > date)]

tickers = list(set(universe['secID'].tolist()) - set(get\_st\_tickers(date)))

return tickers

def get\_idx\_cons(idx, date):

"""

获取某天指数成分股ticker列表

输入:

idx:str，指数代码

date:str，'YYYYMMDD'格式

返回：

list:指数成份股的ticker

"""

if idx != 'A':

try:

data = DataAPI.IdxConsGet(ticker=idx, intoDate=date, field='', pandas="1")['consID']

except Exception as e:

raise ValueError(u'DataAPI.IdxConsGet出错了！！！: %s' % e)

if len(data) < 50:

raise ValueError('{0}该日指数成分股API取出来的成分股数不足50个！！！'.format(date))

else:

universe = get\_all\_tickers(date)

st\_stk = get\_st\_tickers(date)

return list(set(universe) - set(st\_stk))

return list(set(data))

def pretreat\_factor(factor\_df, neu=True):

"""

去极值，中性化，标准化

"""

pretreat\_data = factor\_df.copy(deep=True)

for dt in pretreat\_data.index:

try:

factor\_dt = pretreat\_data.ix[dt].dropna()

factor\_dt\_dict = factor\_dt.to\_dict()

if neu:

pretreat\_data.ix[dt] = pd.Series(standardize(neutralize(winsorize(factor\_df.ix[dt].to\_dict()), target\_date=''.join(dt.split('-')), industry\_type='SW1', exclude\_style\_list=['BETA', 'RESVOL', 'MOMENTUM', 'EARNYILD', 'BTOP', 'SIZENL', 'GROWTH', 'LEVERAGE', 'LIQUIDTY'])))

else:

pretreat\_data.ix[dt] = pd.Series(standardize(winsorize(factor\_dt\_dict)))

except Exception as excp:

print dt

print excp

continue

return pretreat\_data

def fill\_surprise\_factor(df, name):

"""

处理业绩增长相关因子数据的格式

"""

df = df.pivot(index='publishDate', columns='secID', values='signal').loc[trade\_date\_list, :].fillna(method='ffill').loc[month\_date\_list, :].unstack().reset\_index()

df.columns = ['secID', 'publishDate', name]

return df

def select\_universe(x, universe):

try:

return x[x['secID'].isin(universe.loc[x['tradeDate'].iloc[0], :].dropna().tolist())]

except Exception as e:

print x

raise Exception(e)

def get\_factor\_by\_day(args):

"""

根据日期，获取当天的因子值

输入：

args：tuple:(list, str)，第一个元素为因子名称列表，第二个元素为'YYYYMMDD'格式的字符串，表示时间

返回：

DataFrame：因子数据

"""

factor\_names, date = args

cnt = 0

while True:

try:

x = DataAPI.MktStockFactorsOneDayProGet(tradeDate=date,

secID=u"",

field=['secID', 'tradeDate'] + factor\_names,

pandas="1")

return x

except Exception as e:

logging.info(e)

cnt += 1

if cnt >= 3:

print('error get factor data: ', date)

break

def get\_multi\_factor(factor\_names, trade\_date\_list):

"""

取多个因子数据

输入：

factor\_names: list, 因子名

trade\_date\_list: list, 交易日列表

返回：

DataFrame：记录多个因子数据

"""

pool = ThreadPool(processes=4)

factor\_names = [factor\_names] \* len(trade\_date\_list)

frame\_list = pool.map(get\_factor\_by\_day, zip(factor\_names, trade\_date\_list))

pool.close()

pool.join()

factor = pd.concat(frame\_list, axis=0)

factor['tradeDate'] = factor['tradeDate'].str.replace('-', '')

return factor

def winsorize\_by\_date(cdate\_input):

'''

按照[dm+5\*dm1, dm-5\*dm1]进行winsorize

参数:

cdate\_input: 某一期的因子值的dataframe

返回:

DataFrame, 去极值后的因子值

'''

cdate\_input = cdate\_input.copy()

dm = cdate\_input.median()

dm1 = (cdate\_input - dm).abs().median()

upper = dm + 5 \* dm1

lower = dm - 5 \* dm1

cdate\_input[cdate\_input > upper] = upper

cdate\_input[cdate\_input < lower] = lower

return cdate\_input

def nafill\_by\_sw1(cdate\_input):

"""

缺失值填充，使用用申万一级行业中位数

输入：

cdate\_input: 因子值，DataFrame

返回：

DataFrame, 填充缺失值后的因子值

"""

func\_input = cdate\_input.copy()

func\_input = func\_input.merge(sw\_map\_frame[['secID', 'industryName1']], on=['secID'], how='left')

func\_input.loc[:, factor\_name] = func\_input.loc[:, factor\_name].fillna(

func\_input.groupby('industryName1')[factor\_name].transform("median"))

return func\_input.fillna(0.0)

def winsorize\_fillna\_date(tdate):

"""

对某一天的数据进行去极值，填充缺失值

输入：

tdate： str， 'YYYYMMDD'格式

返回：

DataFrame, 去极值，填充缺失值后的因子值

"""

cnt = 0

while True:

try:

# 缺失值填充, 用同行业的均值

cdate\_input = input\_frame[input\_frame.tradeDate == tdate]

cdate\_input = nafill\_by\_sw1(cdate\_input)

cdate\_input.set\_index('secID', inplace=True)

# print("####Running single\_date for %s" % tdate)

# winsorize

cdate\_input = winsorize\_by\_date(cdate\_input)

return cdate\_input

except Exception as e:

cnt += 1

if cnt >= 3:

cdate\_input = input\_frame[input\_frame.tradeDate == tdate]

# 缺失值填充, 用同行业的均值

cdate\_input = nafill\_by\_sw1(cdate\_input)

cdate\_input.set\_index('secID', inplace=True)

return cdate\_input

def standardize(x):

return (x - x.mean()) / x.std()

def standardize\_neutralize\_factor(input\_data):

"""

进行中性化和标准化

输入：

input\_data：tuple, 传入的是(因子值，时间)。因子值为DataFrame

返回：

DataFrame, 行业标准化后的因子值

"""

cdate\_input, tdate = input\_data

for a\_factor in factor\_name + ['sue0', 'sue1', 'sur0', 'sur1', 'sales\_growth\_yoy', 'profit\_growth\_yoy']:

cnt = 0

while True:

try:

cdate\_input.loc[:, a\_factor] = standardize(neutralize(cdate\_input.loc[:, a\_factor],

target\_date=tdate,

exclude\_style\_list=['BETA', 'RESVOL', 'MOMENTUM', 'EARNYILD', 'BTOP',

'GROWTH', 'LEVERAGE', 'LIQUIDTY', 'SIZENL']))

break

except Exception as e:

cnt += 1

if cnt >= 3:

break

return cdate\_input

def standardize\_factor(input\_data):

"""

进行行业内标准化

输入：

input\_data：tuple, 传入的是(因子值，时间)。因子值为DataFrame

返回：

DataFrame, 行业标准化后的因子值

"""

cdate\_input, tdate = input\_data

for a\_factor in factor\_name:

cnt = 0

while True:

try:

cdate\_input.loc[:, a\_factor] =standardize(cdate\_input[a\_factor])

break

except Exception as e:

cnt += 1

if cnt >= 3:

break

return cdate\_input

if \_\_name\_\_ == "\_\_main\_\_":

# 设置起始时间和结束时间

begin\_date = '2007-01-01'

end\_date = '2018-06-30'

factor\_name = ['OperatingRevenueGrowRate', 'NetProfitGrowRate', 'PE', 'PB', 'PS', 'PCF', 'CETOP',

'ROE', 'ROA', 'EPS', 'ROIC', 'GrossIncomeRatio', 'VOL20', 'DAVOL20', 'ILLIQUIDITY',

'REVS20', 'REVS60']

trade\_date\_list = DataAPI.TradeCalGet(exchangeCD='XSHG', beginDate=begin\_date, endDate=end\_date)

trade\_date\_list = trade\_date\_list[trade\_date\_list['isMonthEnd'] == 1]['calendarDate'].tolist()

# 因子数据的获取

input\_frame = get\_multi\_factor(factor\_name, trade\_date\_list)

# 部分因子方向的调整

input\_frame['PB'] = 1. / input\_frame['PB']

input\_frame['PE'] = 1. / input\_frame['PE']

input\_frame['PS'] = 1. / input\_frame['PS']

input\_frame['PCF'] = 1. / input\_frame['PCF']

input\_frame['REVS20'] = -1 \* input\_frame['REVS20']

input\_frame['REVS60'] = -1 \* input\_frame['REVS60']

input\_frame['DAVOL20'] = -1 \* input\_frame['DAVOL20']

input\_frame['VOL20'] = -1 \* input\_frame['VOL20']

class SurpriseFactor(object):

"""

计算业绩惊喜相关因子

"""

def \_\_init\_\_(self, income\_statement, end\_date\_list):

self.income\_statement = income\_statement

self.end\_date\_list = end\_date\_list

@classmethod

def cal\_signal(cls, df, columns, end\_date\_list, shift=True):

df1 = df.copy()

df1.sort\_values(by=['publishDate', 'endDate'], ascending=False, inplace=True)

df2 = df.set\_index('publishDate')

date\_list = df1['publishDate'].unique()

date\_list.sort()

for date in sorted(date\_list):

tmp = df1[df1.publishDate <= date]

tmp.drop\_duplicates(subset=['secID', 'endDate'], inplace=True, keep='first')

tmp = tmp.sort\_values(by='endDate', ascending=False).set\_index('endDate')

report\_end\_date = tmp.index[0]

report\_date\_list = end\_date\_list[end\_date\_list <= report\_end\_date][-13:][::-1]

tmp = tmp.reindex(report\_date\_list).head(13)

tmp[columns] = tmp[columns].diff(-4)

sigma = tmp[columns][1:].std() if len(tmp.dropna()) >= 4 else np.NaN

if shift:

df2.loc[date, 'signal'] = (tmp[columns].iloc[0] - tmp[columns].iloc[1:].mean()) / sigma

else:

df2.loc[date, 'signal'] = tmp[columns].iloc[0] / sigma

df2 = df2.reset\_index()

return df2

@classmethod

def cal\_yoy\_signal(cls, df, columns, end\_date\_list):

df1 = df.copy()

df1.sort\_values(by=['publishDate', 'endDate'], ascending=False, inplace=True)

df2 = df.set\_index('publishDate')

date\_list = df1['publishDate'].unique()

date\_list.sort()

for date in sorted(date\_list):

tmp = df1[df1.publishDate <= date]

tmp.drop\_duplicates(subset=['secID', 'endDate'], inplace=True, keep='first')

tmp = tmp.sort\_values(by='endDate', ascending=False).set\_index('endDate')

report\_end\_date = tmp.index[0]

report\_date\_list = end\_date\_list[end\_date\_list <= report\_end\_date][-5:][::-1]

tmp = tmp.reindex(report\_date\_list).head(5)

df2.loc[date, 'signal'] = (tmp[columns].iloc[0] - tmp[columns].iloc[-1]) / np.abs(tmp[columns].iloc[-1])

df2 = df2.reset\_index()

return df2

def cal\_sue0(self):

"""

带漂移项的净利润业绩惊喜因子

:return: DataFrame, 公告日发布后计算的因子数据

"""

sue0 = self.income\_statement.groupby(by='secID').apply(lambda x: SurpriseFactor.cal\_signal(x,

'NIncomeAttrP',

self.end\_date\_list,

True))

sue0 = sue0.drop('secID', axis=1).reset\_index()

sue0.drop\_duplicates(subset=['secID', 'publishDate'], inplace=True, keep='first')

sue0 = sue0[['secID', 'publishDate', 'signal']].dropna()

return sue0

def cal\_sue1(self):

"""

不带漂移项的营业收入业绩惊喜因子

:return: DataFrame, 公告日发布后计算的因子数据

"""

sue1 = self.income\_statement.groupby(by='secID').apply(lambda x: SurpriseFactor.cal\_signal(x,

'NIncomeAttrP',

self.end\_date\_list,

False))

sue1 = sue1.drop('secID', axis=1).reset\_index()

sue1.drop\_duplicates(subset=['secID', 'publishDate'], inplace=True, keep='first')

sue1 = sue1[['secID', 'publishDate', 'signal']].dropna()

return sue1

def cal\_sur0(self):

"""

带漂移项的营业收入业绩惊喜因子

:return: DataFrame, 公告日发布后计算的因子数据

"""

sur0 = self.income\_statement.groupby(by='secID').apply(lambda x: SurpriseFactor.cal\_signal(x,

'revenue',

self.end\_date\_list,

True))

sur0 = sur0.drop('secID', axis=1).reset\_index()

sur0.drop\_duplicates(subset=['secID', 'publishDate'], inplace=True, keep='first')

sur0 = sur0[['secID', 'publishDate', 'signal']].dropna()

return sur0

def cal\_sur1(self):

"""

带漂移项的净利润业绩惊喜因子

:return: DataFrame, 公告日发布后计算的因子数据

"""

sur1 = self.income\_statement.groupby(by='secID').apply(lambda x: SurpriseFactor.cal\_signal(x,

'revenue',

self.end\_date\_list,

False))

sur1 = sur1.drop('secID', axis=1).reset\_index()

sur1.drop\_duplicates(subset=['secID', 'publishDate'], inplace=True, keep='first')

sur1 = sur1[['secID', 'publishDate', 'signal']].dropna()

return sur1

def cal\_profit\_growth\_yoy(self):

"""

计算净利润增长率单季度同比因子

:return:

"""

profit\_growth\_yoy = self.income\_statement.groupby(by='secID').apply(lambda x: SurpriseFactor.cal\_yoy\_signal(x,

'NIncomeAttrP',

self.end\_date\_list))

profit\_growth\_yoy.drop\_duplicates(subset=['secID', 'publishDate'], inplace=True, keep='first')

profit\_growth\_yoy = profit\_growth\_yoy[['secID', 'publishDate', 'signal']].dropna()

profit\_growth\_yoy.reset\_index(drop=True, inplace=True)

return profit\_growth\_yoy

def cal\_sales\_growth\_yoy(self):

"""

计算营业收入增长率单季度同比因子

:return:

"""

sales\_growth\_yoy = self.income\_statement.groupby(by='secID').apply(lambda x: SurpriseFactor.cal\_yoy\_signal(x,

'revenue',

self.end\_date\_list))

sales\_growth\_yoy.drop\_duplicates(subset=['secID', 'publishDate'], inplace=True, keep='first')

sales\_growth\_yoy = sales\_growth\_yoy[['secID', 'publishDate', 'signal']].dropna()

sales\_growth\_yoy.reset\_index(drop=True, inplace=True)

return sales\_growth\_yoy

**进行业绩惊喜相关因子的计算（耗时约40分钟）**

**if \_\_name\_\_ == '\_\_main\_\_':**

**# 全A投资域**

**a\_universe\_list = DataAPI.EquGet(equTypeCD=u"A",listStatusCD=u"L,S,DE",field=u"secID",pandas="1")['secID'].tolist()**

**a\_universe\_list.remove('DY600018.XSHG')**

**data = DataAPI.FdmtISQPITGet(secID=a\_universe\_list, field=u"secID,publishDate,endDate,NIncomeAttrP,revenue",pandas="1")**

**data = data[data['secID'].str[0].isin(['0', '3', '6'])]**

**data.to\_csv('income\_statement.csv')**

**data = pd.read\_csv('income\_statement.csv', index\_col=0)**

**date\_list = np.array(sorted(data['endDate'].unique()))**

**surprise\_factor = SurpriseFactor(data, date\_list)**

**sue0 = surprise\_factor.cal\_sue0()**

**sue0.to\_csv(path + 'sue0.csv')**

**sue1 = surprise\_factor.cal\_sue1()**

**sue1.to\_csv(path + 'sue1.csv')**

**sur0 = surprise\_factor.cal\_sur0()**

**sur0.to\_csv(path + 'sur0.csv')**

**sur1 = surprise\_factor.cal\_sur1()**

**sur1.to\_csv(path + 'sur1.csv')**

**profit\_growth\_yoy = surprise\_factor.cal\_profit\_growth\_yoy()**

**profit\_growth\_yoy.to\_csv(path + 'profit\_growth\_yoy.csv')**

**sales\_growth\_yoy = surprise\_factor.cal\_sales\_growth\_yoy()**

**sales\_growth\_yoy.to\_csv(path + 'sales\_growth\_yoy.csv')**

4.1 因子说明

* 我们重新回顾因子的构建逻辑，发现该因子与成长因子的结构比较相似。对于成长因子来说，为了不同股票之间的可比性，一般来说会除以上一期的值，但当上一期净利润为负时，这时候就会出现问题，简单起见，本文此处做法采用的是对分母取绝对值的方式。与成长因子不同，业绩惊喜因子则是除以波动进行调整（类似于标准化的做法）。但不管怎样，直观上来看，这两类因子分子相同，因此它们之间的相关性应该较高。这时候，我们需要先考察它是否给我们的模型带来额外的增量信息。
* 首先，为了降低问题复杂度，我们主要考虑大类因子之间的相关性。我们首先将这四个业绩惊喜因子进行等权合成，得到业绩惊喜这个大类因子，然后我们从当前的因子库当中，挑选一些比较成熟的因子，进行大类的合成（等权）。如下所示：

