1.25/9

注意到

E(X)=\frac{1}{4}E(X|0)+\frac{3}{4}E(X|1)

E(X|1) = 2

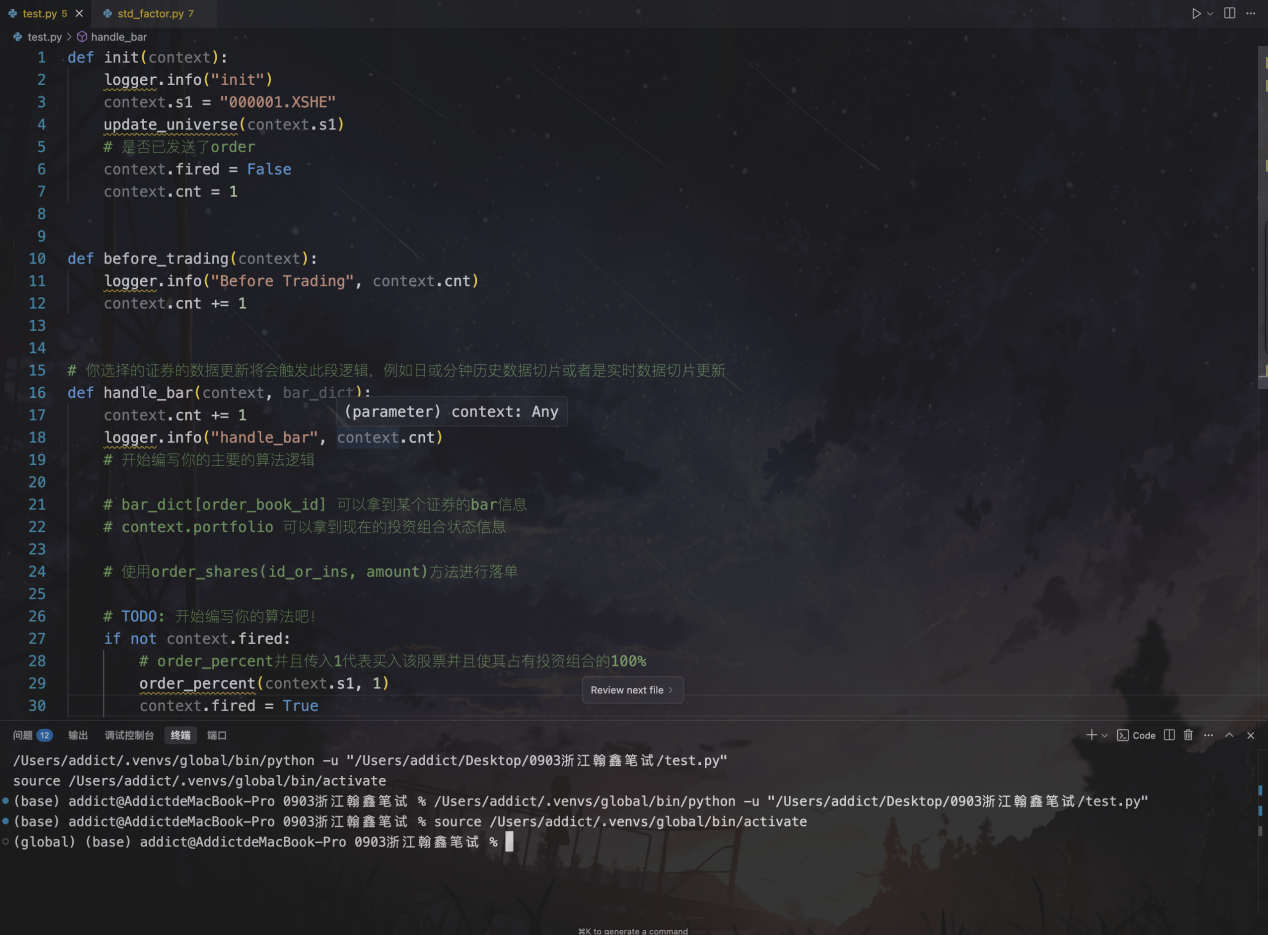
E(X|0) = \frac{1}{4}E(X|00) + \frac{3}{4}E(X|01)

E(X|00) = E(X|0) + 1

E(X|01) = E(X) + 2

解得E(X) = 25/9

2.



参数选择：重新调整周期为3天（周期2-6天都较稳定），测试时间2016-01-01到2025-08-31

注意到，因子在2021年之后显著失效。

import talib

import numpy as np

import pandas as pd

def init(context):

# 设置股票池:选择固定的100支沪深300成份股

context.stocks = [

"000001.XSHE", "000002.XSHE", "000063.XSHE", "000066.XSHE", "000100.XSHE",

"000157.XSHE", "000166.XSHE", "000301.XSHE", "000338.XSHE", "000425.XSHE",

"000538.XSHE", "000568.XSHE", "000596.XSHE", "000625.XSHE", "000627.XSHE",

"000661.XSHE", "000703.XSHE", "000708.XSHE", "000723.XSHE", "000725.XSHE",

"000728.XSHE", "000732.XSHE", "000733.XSHE", "000735.XSHE", "000738.XSHE",

"000750.XSHE", "000768.XSHE", "000776.XSHE", "000783.XSHE", "000786.XSHE",

"000800.XSHE", "000858.XSHE", "000876.XSHE", "000895.XSHE", "000938.XSHE",

"000961.XSHE", "000977.XSHE", "000983.XSHE", "000999.XSHE", "001979.XSHE",

"002001.XSHE", "002007.XSHE", "002027.XSHE", "002142.XSHE", "002230.XSHE",

"002241.XSHE", "002271.XSHE", "002304.XSHE", "002415.XSHE", "002475.XSHE",

"002594.XSHE", "002714.XSHE", "002841.XSHE", "300003.XSHE", "300015.XSHE",

"300059.XSHE", "300122.XSHE", "300124.XSHE", "300142.XSHE", "300274.XSHE",

"300347.XSHE", "300408.XSHE", "300433.XSHE", "300498.XSHE", "300601.XSHE",

"300750.XSHE", "300760.XSHE", "300782.XSHE", "300896.XSHE", "300957.XSHE",

"300999.XSHE", "600000.XSHG", "600009.XSHG", "600016.XSHG", "600019.XSHG",

"600028.XSHG", "600030.XSHG", "600036.XSHG", "600048.XSHG", "600050.XSHG",

"600104.XSHG", "600111.XSHG", "600276.XSHG", "600309.XSHG", "600340.XSHG",

"600436.XSHG", "600519.XSHG", "600585.XSHG", "600588.XSHG", "600690.XSHG",

"600703.XSHG", "600745.XSHG", "600760.XSHG", "600809.XSHG", "600887.XSHG",

"600900.XSHG", "600905.XSHG", "600918.XSHG", "600941.XSHG", "600958.XSHG",

"600989.XSHG", "600999.XSHG", "601012.XSHG", "601088.XSHG", "601138.XSHG",

"601166.XSHG", "601318.XSHG", "601398.XSHG", "601601.XSHG", "601668.XSHG",

"601766.XSHG", "601788.XSHG", "601816.XSHG", "601857.XSHG", "601888.XSHG",

"601899.XSHG", "601919.XSHG", "601988.XSHG", "601989.XSHG", "601995.XSHG",

"601998.XSHG", "603259.XSHG", "603288.XSHG", "603501.XSHG", "603986.XSHG",

"603993.XSHG", "688005.XSHG", "688012.XSHG", "688111.XSHG", "688363.XSHG",

"688396.XSHG", "688981.XSHG"

]

# 策略参数

context.VOLUME\_PERIOD = 20 # 成交量标准差计算周期

context.TOP\_PERCENT = 0.1 # 选择前10%的股票

context.REBALANCE\_FREQ = 3 # 重新平衡（3个交易日）

context.ORDER\_PERCENT = 0.9 # 每次买入使用90%的可用资金

context.last\_rebalance\_day = 0

context.current\_positions = set()

context.last\_total\_value = context.portfolio.total\_value

context.strategy\_start\_time = context.now

def calculate\_volume\_std\_factor(stock, context):

try:

# 获取历史成交量数据

volume\_data = history\_bars(stock, context.VOLUME\_PERIOD + 1, '1d', 'volume')

if len(volume\_data) < context.VOLUME\_PERIOD:

return 0

volume\_std = np.std(volume\_data[-context.VOLUME\_PERIOD:])

volume\_mean = np.mean(volume\_data[-context.VOLUME\_PERIOD:])

# std与均值的比，避免只会选到大市值股票

if volume\_mean > 0:

factor\_value = volume\_std / volume\_mean

else:

factor\_value = 0

return factor\_value

except Exception as e:

return 0

def select\_top\_stocks(context):

stock\_factors = []

for stock in context.stocks:

factor\_value = calculate\_volume\_std\_factor(stock, context)

if factor\_value > 0: # 只考虑有效的因子值

stock\_factors.append((stock, factor\_value))

stock\_factors.sort(key=lambda x: x[1], reverse=True)

top\_count = max(1, int(len(stock\_factors) \* context.TOP\_PERCENT))

top\_stocks = [stock for stock, \_ in stock\_factors[:top\_count]]

return top\_stocks

def rebalance\_portfolio(context, selected\_stocks):

for stock in list(context.current\_positions):

if stock not in selected\_stocks:

position = get\_position(stock)

if position and position.quantity > 0:

order\_target\_value(stock, 0)

context.current\_positions.remove(stock)

if selected\_stocks:

total\_target\_value = context.portfolio.total\_value \* context.ORDER\_PERCENT

current\_positions\_value = sum([get\_position(stock).market\_value for stock in selected\_stocks if get\_position(stock)])

additional\_investment = total\_target\_value - current\_positions\_value

if additional\_investment > 0:

target\_value\_per\_stock = additional\_investment / len(selected\_stocks)

for stock in selected\_stocks:

current\_position = get\_position(stock)

current\_value = current\_position.market\_value if current\_position else 0

if current\_value < target\_value\_per\_stock \* 0.9: # 允许10%的误差

try:

order\_value(stock, target\_value\_per\_stock)

context.current\_positions.add(stock)

except Exception as e:

return

else:

return

def handle\_bar(context, bar\_dict):

if context.now.day - context.last\_rebalance\_day >= context.REBALANCE\_FREQ:

selected\_stocks = select\_top\_stocks(context)

rebalance\_portfolio(context, selected\_stocks)

context.last\_rebalance\_day = context.now.day

def after\_trading\_end(context):

pass



3.def find\_best\_opportunity(bids, asks):

# 寻找所有的套利机会，按利润排序，选利润最大的

opportunities = []

for bid\_key, bid\_data in bids.items():

bid\_price, bid\_quantity = bid\_data

bid\_exchange, bid\_fee\_str = bid\_key.split('-')

bid\_fee = float(bid\_fee\_str)

for ask\_key, ask\_data in asks.items():

ask\_price, ask\_quantity = ask\_data

ask\_exchange, ask\_fee\_str = ask\_key.split('-')

ask\_fee = float(ask\_fee\_str)

if bid\_exchange == ask\_exchange:

continue

buy\_total = bid\_price + bid\_fee

sell\_total = ask\_price - ask\_fee

if buy\_total < sell\_total:

profit = sell\_total - buy\_total

max\_quantity = min(bid\_quantity, ask\_quantity)

total\_profit = profit \* max\_quantity

opportunities.append([total\_profit,bid\_exchange + "-" + bid\_fee\_str,ask\_exchange + "-" + ask\_fee\_str,max\_quantity])

opportunities.sort(reverse=True)

if opportunities:

return opportunities[0]

else:

return None

def execute\_arbitrage(bids, asks, opp):

# 执行套利

asks[opp[2]][1]-=opp[3]

bids[opp[1]][1]-=opp[3]

if asks[opp[2]][1]==0:

del asks[opp[2]]

if bids[opp[1]][1]==0:

del bids[opp[1]]

return opp[0]

opp = find\_best\_opportunity(bids,asks)

overall\_profit = 0

count = 0

while opp:

overall\_profit+=execute\_arbitrage(bids,asks,opp)

count+=1

opp = find\_best\_opportunity(bids,asks)

print(f"共进行{count}次套利，总收益为{overall\_profit}")

print(asks)

print(bids)

4.

取一支固定股票（记i），我们研究与它相似的股票。r为给定参数

每月的指标作Z-score处理：即对于每个指标，对其dataframe操作，增加两列，一列是均值\mu\_t，一列是标准差\sigma\_t，将该指标转换为标准正态分布（即对原dataframe中的每个元素X\_t,变为(X\_t - \mu\_t) / (\sigma\_t).保存到新的dataframe里

再用处理后的指标计算每支股票与i的欧式距离，除了i以外，其他股票每个指标与i中的相应指标相减再平方，得到的值求和开根，得到每支股票与i的欧式距离，这成为一个dataframe，记作dataframe0.

在dataframe0中，每月取出距离i最近的r支股票，按市值加权求和前一个月的收益率得到SIM\_t(i)

所以，对i我们构建了一个因子叫SIM\_t(i)，再减去i过去一月的收益率r\_{t-1}(i)，就得到

\Delta ER\_i(t) = SIM\_t(i) - r\_t(i)

可能的改进：

1.在计算市值的SIM\_t(i)时研报中按市值加权求和，但这难以反映已经被挑选出的r支股票与i的远近关系，相当于把它们视为和i一样近，可能更好的方式是按距离的倒数加权求和。

2.在定义相似指标时可以考虑业务的相似性，我们可以用一个指标来定义两支股票业务的相似性，同行业的股票此指标就较高，不同行业则较低。这样的方法可以让同行业的股票更“近”，但缺点是在同行竞争时会出现此消彼长的状况，这与我们想要构建的“相似”不符。

3.定义距离不一定需要明确公式，可能更好的角度是learning的角度，用机器学习聚类的方式有希望提升对相似性的刻画程度