**项目链接：**[Lancercxy/automatic\_transaction: 分析上证指数，并建立买卖规则，实现自动化交易 (github.com)](https://github.com/Lancercxy/automatic_transaction)

**准备工作：**

**1、导入相关模块**

import akshare as ak

import matplotlib.pyplot as plt

from pylab import mpl

import pandas as pd

import requests

from bs4 import BeautifulSoup

import re

from jqdatasdk import \*

import time

import datetime

**2、使用之前的项目收集上证A股的所有个股信息（这里为了实验只获取部分数据）**

**项目链接：**[Lancercxy/Finance\_crawler: 财经个股爬虫 (github.com)](https://github.com/Lancercxy/Finance_crawler)

**效果图：**



**3、编写get\_stock()函数该函数接收个股代码并通过akshare模块获取历史行情**

def get\_stock(code):

    #获取当前日期与30天的历史行情

    start\_date = (datetime.datetime.now() - datetime.timedelta(days = 30))

    start\_date = start\_date.strftime('%Y%m%d')

    end\_date = time.strftime('%Y%m%d')

    df = ak.stock\_zh\_a\_hist(symbol=code,start\_date=start\_date, end\_date=end\_date, adjust="")

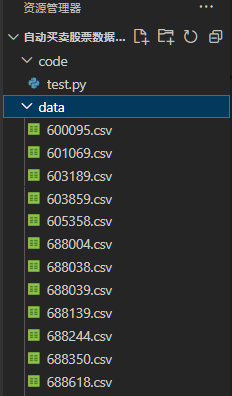
    #个股历史行情存储路径，以股票代码命名

    path = f'D:\office\Github\爬虫\自动买卖股票数据采集\data\{code}.csv'

    #以CSV格式写入信息

    df.to\_csv(path,index=False)

**效果图：**



1. **编写get\_sz()函数该函数使用网络爬虫获取上证指数的历史行情**

def get\_sz():

    #1、请求链接

    url = 'https://q.stock.sohu.com/hisHq?code=zs\_000001&stat=1&order=D&period=d&callback=historySearchHandler&rt=jsonp&0.4530586399394587'

    #2、请求头

    headers = {'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/109.0.0.0 Safari/537.36 Edg/109.0.1518.70',

               'Cookie':'gidinf=x099980109ee1586bb342f0390007640b6db6df607be; reqtype=pc; BIZ\_MyLBS=cn\_688039%2C%u5F53%u8679%u79D1%u6280%7Ccn\_000002%2C%u4E07%u79D1%uFF21%7Ccn\_601121%2C%u5B9D%u5730%u77FF%u4E1A; t=1678499703304'

               }

    #3、发送get请求

    response = requests.get(url, headers=headers)

    #4、获取响应中的字符

    sz = response.text

    #5、通过三步清洗获取T-1日涨跌幅

    sz01 = sz.split('[')[4].split(']')[0]

    sz02 = sz01.split(',')[4]

    sz03 = sz02.replace('%','').replace('"','')

    #6、判断涨跌幅是否大于等于0.5,满足条件返回True否则返回False

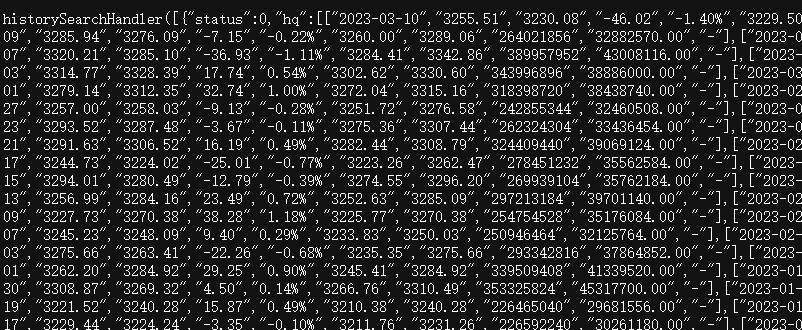
    if sz03 >= 0.5:

        return True

    else:

        return False

**上证指数历史行情：**



**5、编写calculation\_cci()函数该函数接收个股代码列表，设计计算CCI值算法筛选满足条件的股票，然后返回**

def calculation\_cci(stock\_list):

    #1、创建buy\_code\_list列表用于存储满足条件的个股代码

    buy\_code\_list = []

    #2、遍历代码

    for code in stock\_list:

        #3、根据传入的股票代码读取数据

        df = pd.read\_csv(f'D:\office\Github\爬虫\自动买卖股票数据采集\data\{code}.csv',sep=',')

        #4、获取T-1日涨跌幅

        df\_Rise\_fall = df[-2:].head(1)['涨跌幅']

        df\_Rise\_fall = float(df\_Rise\_fall)

        #5、创建cci\_list列表用于存储T-1与T-2的cci值

        cci\_list = []

        for j in range(1,3):

            #6、删去T日与T-1的数据然后计算cci

            df.drop(df.tail(1).index,inplace=True)

            #7、获取最后一行数据（当天数据）计算TP值

            TP = (df[-1:]['最高']+df[-1:]['最低']+df[-1:]['收盘'])/3

            # print("T-{0}的TP={1}".format(j,TP))

            #8、获取进14日的数据，并计算MA值

            df\_14 = df.tail(14)

            MA = (df\_14[['最高','最低','收盘']].sum(axis=1).sum(axis=0))/42

            # print("T-{0}的MA=".format(j,MA))

            #9、计算MD值

            md = df\_14[['最高','最低','收盘']].sum(axis=1)

            #10、参数add用于存储14天TP之和

            add = 0

            for i in md:

                add += abs(i/3 - MA)

            MD = add/14

            # print("T-{0}的MD=".format(j,MD))

            #11、计算cci数据

            cci = (TP - MA)/MD/0.015

            # print("T-{0}的cci值为:{1:.2f}".format(j,float(cci)))

            #12、将cci值添加入列表

            cci\_list.append(float(cci))

            # print(cci\_list)

        #13、根据要求进行判断

        # if cci\_list[1] >= -50 and cci\_list[1] <= 100 and cci\_list[0] >= 100 and cci\_list[0] <= 200 and df\_Rise\_fall >= 10:

        #为了测试降低了买入要求

        if cci\_list[1] >= -50 and cci\_list[1] <= 100 :

            print('可买股票',code)

            #14、将符合要求的个股添加入buy\_code\_list列表

            buy\_code\_list.append(code)

            # return code

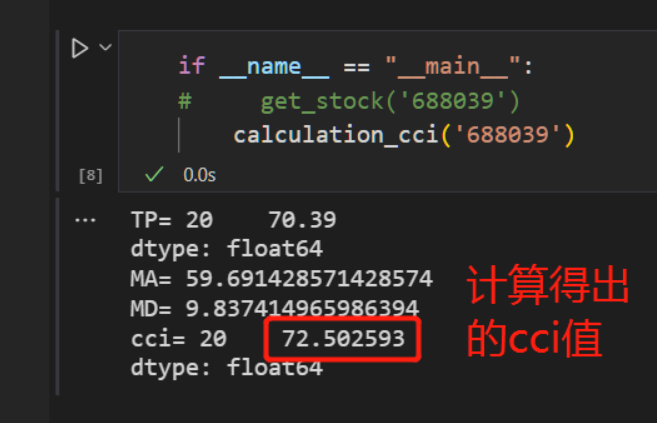
        else:

            print('不可买',code)

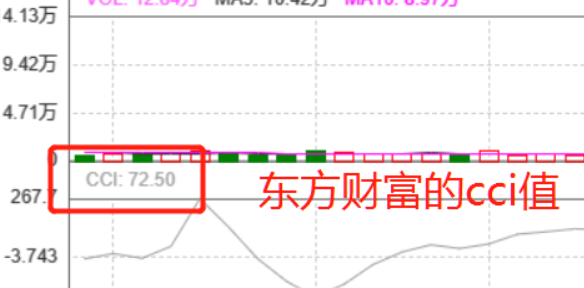
    #15、返回符合要求的代码列表

    return buy\_code\_list

**计算得到的CCI值：**



**实际CCI值：**



**6、编写high\_open()函数该函数用于接收个股代码进行高开判断**

def high\_open(code):

    #1、获取今天日期

    g.today = context.current\_dt.strftime('%Y-%m-%d')

    #2、获取上个交易日日期

    g.start = context.current\_dt - datetime.timedelta(day = 1)

    #3、以上个交易日为开始时间，今日为结束时间获取个股信息

    grid = get\_price(f'{code}.XSHE', start\_date=g.start, end\_date=g.today, fields=['open', 'high', 'low', 'close', 'high\_limit', 'paused'])

    #4、判断是否满足高开在0-2%之间

    if len(grid)>1 and grid.open[-1] > grid.close[-2] and grid.open[-1] / grid.close[-2] >= 1.00 and grid.open[-1] / grid.close[-2] <= 1.02 :

        return True

    else:

        return False

**7、编写change\_pct()函数该函数用于接收个股代码进行涨跌幅判断**

def change\_pct(code):

    #1、获取今天日期

    g.today = context.current\_dt.strftime('%Y-%m-%d')

    #2、获取上个交易日日期

    g.start = context.current\_dt - datetime.timedelta(day = 1)

    #3、获取涨跌幅

    money\_flow = get\_money\_flow(f'{code}.XSHG',start\_date=g.start, end\_date=g.today, fields="change\_pct")

    #4、判断是否满足高开在0-2%之间

    if len(money\_flow)>1 and money\_flow.change\_pct[-1] < 0.1 and money\_flow.change\_pct[-1] > -0.1 :

        return '0'

    elif money\_flow.change\_pct[-1] <= -0.1:

        return '-1'

    else:

        return '1'

**8、编写timeFun()函数该函用于定时执行某函数**

def timerFun(sell\_Timer,fun):

    #标识参数

    flag = 0

    while True:

        #获取当前时间并与传入时间对比

        now = datetime.datetime.now().strftime('%Y%m%d%H%M%S')

        if now == sell\_Timer.strftime('%m%d%H%M%S'):

            #若到了规定时间则执行函数

            # fun

            #并将flag参数设置为1

            flag = 1

        else:

            #若flag为1,意味着已经执行完目标函数，可将时间调整为下一天

            if flag == 1:

                sell\_Timer = sell\_Timer + datetime.timedelta(day=1)

                #将标识重新标为0

                flag = 0

**以上准备工作完毕**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**以下为程序执行顺序**

**一、开盘前执行before\_market\_open()函数该函数用于开盘前对数据进行收集和筛选出符合要求的个股使用buy\_code\_list参数接收返回值**

#该函数用于开盘前对数据进行收集和筛选出符合要求的个股

def before\_market\_open():

    #1、读取提前使用网络爬虫收集好的上证A股的所有个股信息

    df = pd.read\_csv('D:\office\Github\爬虫\财经\东方财富.csv',sep=',')

    #2、创建名为stock\_list的列表，用于存储个股代码

    stock\_list = []

    #3、为了方便测试只拿出10支股票进行操作

    for i in df['代码'][30:41]:

        #4、清洗数据中多余的字符，并添加进列表里

        stock\_list.append(i.replace('"',''))

    #5、遍历列表

    for j in stock\_list:

        #6、将列表中的个股代码逐个传入get\_stock()函数，以获取该股票的历史行情

        get\_stock(j)

    #7、判断T-1日上证指数涨幅是否大于0.5%，get\_sz()函数用于判断

    # if get\_sz():

    #为了方便测试条件设定为True

    if True:

        #8、满足T-1日上证指数涨幅大于0.5%,使用calculation\_cci()计算cci值，进行进一步筛选

        code = calculation\_cci(stock\_list)

        #9、判断code是否为空，不为空则返回

        if len(code):

            return code

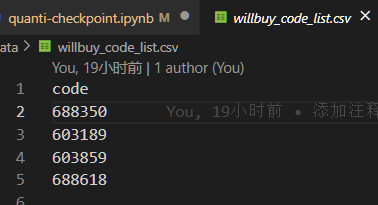
**将列表转化为DataFrame类型，再以CSV格式保存再本地**

pf = pd.DataFrame(buy\_code\_list)

    pf.columns=['code']

    pf.to\_csv('D:\office\Github\爬虫\自动买卖股票数据采集\willbuy\_code\_list.csv',index=False)

**满足题目条件的个股代码：**



**三、开盘时调用market\_open()函数执行买入操作**

def market\_open():

    # log.info('函数运行时间(market\_open):'+str(context.current\_dt.time()))

    #1、读取将要购入的个股代码

    df = pd.read\_csv('D:\office\Github\爬虫\自动买卖股票数据采集\data\willbuy\_code\_list.csv')

    #2、创建名为own\_stock的列表，用于存储以购买的个股代码

    own\_stock = []

    #3、取得当前的现金并平均分成n份（n为即将购买的个股数）

    cash = context.portfolio.available\_cash/len(df)

    #4、遍历数据

    for code in df['code']:

        # print(code)

        # g.security = f'{code}.XSHE'

        # security = g.security

        #5、调用high\_open()函数判断高开是否在0-2%之间

        if high\_open(f'{code}.XSHE'):

            #6、若满足要求用cash买入股票

            order\_value(code, cash)

            #7、将已经买入的股票代码保存入own\_stock列表

            own\_stock.append(code)

        else:

            # 若不满足则打印出来

            print("不买入",code)

    #8、将已经买入的股票代码以CSV格式保存入本地文件

    pf = pd.DataFrame(own\_stock)

    pf.columns=['code']

    pf.to\_csv('D:\office\Github\爬虫\自动买卖股票数据采集\data\own\_stock.csv',index=False)

**四、14:55执行卖出操作**

#获取T+1日14:55的时间

    tomorrow = (datetime.datetime.now() + datetime.timedelta(days = 1))

    sell\_Timer = datetime.datetime(tomorrow.year,tomorrow.month,tomorrow.day,14,55,0)

    #将规定时间和函数传入timerFun()函数,该函数能控制在规定时间点运行指定函数

    timerFun(sell\_Timer,sell\_stock())

**实现卖出操作函数：**

#该函数用于卖出符合要求的个股

def sell\_stock():

    #1、读取已拥有的个股代码

    df = pd.read\_csv('D:\office\Github\爬虫\自动买卖股票数据采集\data\own\_stock.csv')

    #2、创建Limit\_of\_drop列表用于存储跌停的个股

    Limit\_of\_drop = []

    #3、判断数据是否为空,不为空则遍历数据

    if len(df) != 0:

        for code in df['code']:

            # print(code)

            # g.security = f'{code}.XSHE'

            # security = g.security

            #4、调用change\_pct()函数判断其涨跌幅(返回'0'可卖出,返回'-1'为跌停,涨停返回'1')

            state = change\_pct(f'{code}.XSHE')

            if state == '0':

                #5、若满足要求卖出股票

                order\_target(code, 0)

                #6、将已经卖出的股票代码从own\_stock文件移除

                df = pd.read\_csv('D:\office\Github\爬虫\自动买卖股票数据采集\data\own\_stock.csv')

                df01 = df.drop(df[df['code']==code].index)

                df01.to\_csv('D:\office\Github\爬虫\自动买卖股票数据采集\data\own\_stock.csv',index=False)

            #7、处理跌停的个股

            elif state == '-1':

                #8、将代码添加进列表

                Limit\_of\_drop.append(code)

                #9、将跌停的股票代码从own\_stock文件移除

                df = pd.read\_csv('D:\office\Github\爬虫\自动买卖股票数据采集\data\own\_stock.csv')

                df01 = df.drop(df[df['code']==code].index)

                df01.to\_csv('D:\office\Github\爬虫\自动买卖股票数据采集\data\own\_stock.csv',index=False)

                # print("无法卖出",code)

        #10、将跌停的股票代码以CSV格式保存为Limit\_of\_drop\_stock文件

        drop = pd.DataFrame(Limit\_of\_drop)

        drop.columns=['code']

        drop.to\_csv('D:\office\Github\爬虫\自动买卖股票数据采集\data\Limit\_of\_drop\_stock.csv',index=False)

**五、9:20执行集合竞价操作**

#获取T+2日9:20的时间

    am\_tomorrow = (datetime.datetime.now() + datetime.timedelta(days = 2))

    am\_sell\_Timer = datetime.datetime(am\_tomorrow.year,am\_tomorrow.month,am\_tomorrow.day,9,20,0)

    #在9:20执行集合竞价操作

    timerFun(am\_sell\_Timer,bidding())

**实现集合竞价操作：**

#该函数用于集合竞价操作

def bidding():

    #1、读取跌停的个股代码

    df = pd.read\_csv('D:\office\Github\爬虫\自动买卖股票数据采集\data\Limit\_of\_drop\_stock.csv')

    #2、获取今天日期

    dt=context.current\_dt

    #3、得到股票昨日收盘价, 每天只需要取一次

    last\_df = history(1,'1d','close',code)

    #4、判断数据是否为空,不为空则遍历个股代码

    if len(df) != 0:

        for code in df['code']:

            #5、获取集合竞价买5档挂单数据

            d1 = get\_call\_auction(code, start\_date=dt, end\_date=dt,fields=['time', 'current', 'b1\_p', 'b1\_v'])

            money1=float(d1['b1\_p']\*d1['b1\_v'])

            d2 = get\_call\_auction(code, start\_date=dt, end\_date=dt,fields=['time', 'current', 'b2\_p', 'b2\_v'])

            money2=float(d2['b2\_p']\*d2['b2\_v'])

            d3 = get\_call\_auction(code, start\_date=dt, end\_date=dt,fields=['time', 'current', 'b3\_p', 'b3\_v'])

            money3=float(d3['b3\_p']\*d3['b3\_v'])

            d4 = get\_call\_auction(code, start\_date=dt, end\_date=dt,fields=['time', 'current', 'b4\_p', 'b4\_v'])

            money4=float(d4['b4\_p']\*d4['b4\_v'])

            d5 = get\_call\_auction(code, start\_date=dt, end\_date=dt,fields=['time', 'current', 'b5\_p', 'b5\_v'])

            money5=float(d5['b5\_p']\*d5['b5\_v'])

            #6、集合竞价有抢筹动作，有大单挂买档，至少800万金额 并且高开不能小于4%

            if money1 > g.min\_money or money2 > g.min\_money or money3>g.min\_money or money4>g.min\_money or money5>g.min\_money and d1['current']>(last\_df[code][0]\*1.04) :

                #7、若满足要求则卖出股票

                order\_target(code, 0)

                #8、将已经卖出的股票代码从Limit\_of\_drop\_stock文件移除

                df = pd.read\_csv('D:\office\Github\爬虫\自动买卖股票数据采集\data\Limit\_of\_drop\_stock.csv')

                df01 = df.drop(df[df['code']==code].index)

                df01.to\_csv('D:\office\Github\爬虫\自动买卖股票数据采集\data\Limit\_of\_drop\_stock.csv',index=False)