به نام خدا

گزارش توقف پروژه ی TSE

عملکرد های موفق کد:

کلاس ذخیره ی تاریخچه:

این کلاس اطلاعات تاریخچه ی مورد نیاز جهت بهینه سازی را به صورت numpy array ذخیره میکند.

بهتر است مقادیر date در غالب pandas date ذخیره شوند.(هم اکنون به صورت یک عدد در قالب yyyymmdd ذخیره میشود.)

class TickerHistory:  
  
 def \_\_init\_\_(self, date: np.ndarray,  
 o: np.ndarray, h: np.ndarray, l: np.ndarray, c: np.ndarray,  
 vol: np.ndarray, bc: np.ndarray, bv: np.ndarray, sc: np.ndarray, sv: np.ndarray):  
 self.date = date  
 self.open = o  
 self.high = h  
 self.low = l  
 self.close = c  
 self.vol = vol  
 self.buy\_vol = bv  
 self.buy\_count = bc  
 self.sell\_vol = sv  
 self.sell\_count = sc

کلاس اطلاعات یک سهم:

این کلاس در درون خود اطلاعات تاریخچه و اطلاعات آخرین روز را در بر دارد. اطلاعات آخرین روز در برخی موارد به درستی ثبت نمیشود و مربوط به یک روز قبل تر است. متغیر start یک اندیس مربوط به date در تاریخچه است که بهینه سازی از تاریخ مربوط به این اندیس تا انتهای لیست تاریخچه انجام میشود.

class TickerInfo:  
  
 def \_\_init\_\_(self, ticker, group, ppe, group\_ppe,  
 vol, op, hp, lp, cp, cpm1, lpm1, hpm1,  
 mean\_vol, vol\_std,  
 bc, bv, sc, sv,  
 history: TickerHistory):  
 self.ticker = ticker  
 self.group = group  
 self.ppe = ppe  
 self.group\_ppe = group\_ppe  
 self.vol = vol  
 self.open = op  
 self.high = hp  
 self.low = lp  
 self.close = cp  
 self.yesterday\_close = cpm1  
 self.yesterday\_low = lpm1  
 self.yesterday\_high = hpm1  
 self.mean\_vol = mean\_vol  
 self.vol\_std = vol\_std  
 self.buy\_count = bc  
 self.buy\_vol = bv  
 self.sell\_count = sc  
 self.sell\_vol = sv  
 self.history = history  
 self.start = 0

تابع ذخیره اطلاعات آرائه ای از سهم ها در فایل:

این تابع نام پوشه را میگیرد (برای مثال “t1/” و به تعداد سهم های موجود در آرائه ی ورودی از 0 تا n-1 فایل با اسم های شماره ای از جنس .csv اطلاعات سهم ها را ذخیره میکند. در خط اول ، اطلاعات کلی سهم را ذخیره میکند و سپس در خطوط بعدی اطلاعات تاریخچه را ذخیره میکند.

def write\_tickers\_to\_file(tickers: list[TickerInfo], file\_dir):  
 for i in range(len(tickers)):  
 with open(file\_dir + str(i) + '.csv', mode='x', encoding='utf-8', newline='') as file:  
 writer = csv.writer(file, delimiter=',', quotechar='|', quoting=csv.QUOTE\_MINIMAL)  
 writer.writerow([str(tickers[i].ticker), str(tickers[i].group), str(tickers[i].ppe),  
 str(tickers[i].group\_ppe), str(tickers[i].vol), str(tickers[i].open), str(tickers[i].high),  
 str(tickers[i].low), str(tickers[i].close), str(tickers[i].yesterday\_close),  
 str(tickers[i].yesterday\_low), str(tickers[i].yesterday\_high), str(tickers[i].mean\_vol),  
 str(tickers[i].vol\_std), str(tickers[i].buy\_vol), str(tickers[i].buy\_count),  
 str(tickers[i].sell\_vol), str(tickers[i].sell\_count)])  
 for j in range(len(tickers[i].history.date)):  
 writer.writerow([str(tickers[i].history.date[j]), str(tickers[i].history.open[j]),  
 str(tickers[i].history.high[j]), str(tickers[i].history.low[j]),  
 str(tickers[i].history.close[j]), str(tickers[i].history.vol[j]),  
 str(tickers[i].history.buy\_vol[j]), str(tickers[i].history.buy\_count[j]),  
 str(tickers[i].history.sell\_vol[j]), str(tickers[i].history.sell\_count[j])])

تابع خواندن اطلاعات از فایل:

این تابع در دقیقا عمل معکوس تابع قبل را انجام میدهد و اطلاعات ذخیره شده در فایل ها را به صورت آرائه ای از سهم ها خروجی میدهد.

def read\_tickers\_from\_files(count):  
 tickers = []  
 for i in range(count):  
 with open('t1/' + str(i) + '.csv', mode='r', encoding='utf-8', newline='') as file:  
 reader = csv.reader(file, delimiter=',', quotechar='|', quoting=csv.QUOTE\_MINIMAL)  
 rows = []  
 for row in reader:  
 rows.append(row)  
 first\_row: list = rows.pop(0)  
 for j in range(len(first\_row)):  
 if first\_row[j] == 'None':  
 first\_row[j] = -1  
 tic = TickerInfo(first\_row[0], first\_row[1], float(first\_row[2]), float(first\_row[3]),  
 int(float(first\_row[4])), int(float(first\_row[5])), int(float(first\_row[6])),  
 int(float(first\_row[7])), int(float(first\_row[8])), int(float(first\_row[9])),  
 int(float(first\_row[10])), int(float(first\_row[11])), float(first\_row[12]),  
 float(first\_row[13]), int(float(first\_row[15])), int(float(first\_row[14])),  
 int(float(first\_row[17])), int(float(first\_row[16])), None)  
 d = []  
 o = []  
 h = []  
 l = []  
 c = []  
 v = []  
 bv = []  
 bc = []  
 sv = []  
 sc = []  
 for row in rows:  
 d.append(int(float(row[0])))  
 o.append(int(float(row[1])))  
 h.append(int(float(row[2])))  
 l.append(int(float(row[3])))  
 c.append(int(float(row[4])))  
 v.append(int(float(row[5])))  
 bv.append(int(float(row[6])))  
 bc.append(int(float(row[7])))  
 sv.append(int(float(row[8])))  
 sc.append(int(float(row[9])))  
 tic.history = TickerHistory(np.array(d), np.array(o), np.array(h), np.array(l), np.array(c), np.array(v),  
 np.array(bc), np.array(bv), np.array(sc), np.array(sv))  
 tickers.append(tic)  
 return tickers

تابع اسم سهم ها:

این تابع اسم سهم ها را به صورت ارایه ای از رشته ها بازگردانی میکند. از این اطلاعات در تابع بعدی برای دانلود اطلاعات سهم ها استفاده میشود.

def bourse\_tickers():  
 df1 = tse2.Build\_Market\_StockList(bourse=True, farabourse=False, payeh=False, detailed\_list=True,  
 show\_progress=False, save\_excel=False, save\_csv=False)  
 tickers = df1.index.to\_list()  
 print(len(tickers))  
 return tickers

تابع دانلود اطلاعات سهم ها:

این تابع اسم سهم ها را در یک لیست به عنوان ورودی میگیرد و خروجی آن آخرین اطلاعات تا آخرین روز ثبت شده برای آن ها میباشد که به صورت کامل در قالب دیتا پک تعریف شده آرائه ای از سهم ها بازگردانی میکند. ممکن است بعضی سهم ها دانلود نشوند که به صورت اررور نمایش داده میشوند که موجب توقف برنامه نمیشود و به سراغ سهم بعدی میرود. همچنین ممکن است اطلاعات مربوط به تاریخچه ی روز آخر نیز دانلود نشود.

def download(symbols: list):  
 tick\_res = []  
 for name in symbols:  
 print(name)  
 try:  
 tick = tse.Ticker(symbol=name)  
 dates = np.array([i.year \* 10000 + i.month \* 100 + i.day for i in tick.history['date']])  
 dates2 = [int(i) for i in tick.client\_types['date']]  
 bc2 = tick.client\_types['individual\_buy\_count'].to\_numpy()  
 bv2 = tick.client\_types['individual\_buy\_vol'].to\_numpy()  
 sc2 = tick.client\_types['individual\_sell\_count'].to\_numpy()  
 sv2 = tick.client\_types['individual\_sell\_vol'].to\_numpy()  
 bc = []  
 bv = []  
 sc = []  
 sv = []  
 for i in dates:  
 if i in dates2:  
 bc.append(int(bc2[dates2.index(i)]))  
 bv.append(int(bv2[dates2.index(i)]))  
 sc.append(int(sc2[dates2.index(i)]))  
 sv.append(int(sv2[dates2.index(i)]))  
 else:  
 bc.append(-1)  
 bv.append(-1)  
 sc.append(-1)  
 sv.append(-1)  
 tick\_res.append(TickerInfo(  
 name,  
 tick.group\_name,  
 tick.p\_e\_ratio,  
 tick.group\_p\_e\_ratio,  
 tick.volume,  
 tick.open\_price,  
 tick.high\_price,  
 tick.low\_price,  
 tick.last\_price,  
 tick.history['close'].get(tick.history['close'].count() - 2),  
 tick.history['low'].get(tick.history['low'].count() - 2),  
 tick.history['high'].get(tick.history['high'].count() - 2),  
 tick.history['volume'].mean(),  
 tick.history['volume'].std(),  
 bc[-1],  
 bv[-1],  
 sc[-1],  
 sv[-1],  
 TickerHistory(  
 np.array([i.year \* 10000 + i.month \* 100 + i.day for i in tick.history['date']]),  
 tick.history['open'].to\_numpy(),  
 tick.history['high'].to\_numpy(),  
 tick.history['low'].to\_numpy(),  
 tick.history['close'].to\_numpy(),  
 tick.history['volume'].to\_numpy(),  
 np.array(bc),  
 np.array(bv),  
 np.array(sc),  
 np.array(sv)  
 )  
  
 )  
 )  
 except:  
 print("error in download", name)  
 print(len(tick\_res))  
 return tick\_res

تابع بررسی سود بر اساس داده های مشخص برای الگوریتم تعریف شده:

این تابع به درستی کار نمیکند. روش این تابع این است که به عنوان ورودی فیلتر های مربوط به الگوریتم را دریافت میکند.این فیلتر های به ترتیب به این صورت هستند: سهم توسط حقیقی ها معامله میشود یا حقوقی ها – در هنگام خرید حجم چه میزانی از واریانس بیشتر از میانگین باشد – قدرت خرید نسبت به قدرت فروش در هنگام خرید چه نسبتی داشته باشد – در هنگام خرید تیک سعودی رخ داده باشد یا خیر – در هنگام خرید گپ به چه میزان باشد(اعداد -2و-1و0و1و2) که طبق شرط های آورده شده مشخص است – در هنگام فروش چه میزان از واریانس حجم نسبت به میانگین داشته باشد – در هنگام فروش قدرت فروش چه نسبتی با قدرت خرید داشته باشد – در هنگام فروش تیک نزولی رخ داده باشد یا خیر – در هنگام فروش گپ منفی وجود داشته باشد یا خیر که طبق شرط های آورده شده مقادیر مشخص میشوند – حداکثر میزان باز بودن معامله چند روز کاری باشد – پرافیت حداکثر چه قدر باشد – حد ضرر چه میزان باشد.

اشکال این تابع به این صورت رخ میدهد: این تابع ابتدا بر اساس فیلتر ها برای هر سهم به صورت مستقل بررسی میکند که در چه هنگام در چه تاریخ تا چه تاریخ و با چه قیمت خرید و فروشی سهم قابل معامله است . تمام این حالت ها را در یک آرائه ذخیره میکند . سپس این آرائه را بر اساس تاریخ شروع معامله منظم میکند. سپس از ابتدای آرایه شروع به حرکت میکند تا زمانی که به تعداد حداکثر معاملات به صورت همزمان باز برسد و سایر معاملات را نادیده میگرد تا یکی از معاملات همزمان بسته شود و سپس مجدد اولین معامله ی قابل انجام در لیست را به عنوان معامله ی باز انتخاب میکند. اشکال زمانی رخ میدهد که سورت کردن معاملات تنها بر اساس تاریخ شروع معامله انجام نمیشود و در درجه ی دوم تاریخ پایان هم سورت میشود و در نتیجه معاملاتی که در یک روز انجام میشود ، به این ترتیب سورت میشوند که معامله ای که زود تر به اتمام میرسد در ابتدا قرار میگیرد و معامله ای که دیرتر به انجام میرسد در انتها قرار میگیرد ، این باعث میشود معاملات طولاتی کمتر انجام شوند. در روند بهینه سازی نیز همین عامل باعث خطا در بهینه سازی شد . به این صورت که فیلتر های باز شدن معامله به سمت حذف شدن پیش رفت و تعداد معاملاتی که در یک روز باز میشد بسیار زیاد میشد(گاهی تا 50 معامله به صورت همزمان در یک روز) در این صورت حداقل یک از معاملات وجود داشت که در همان روز اول 5 درصد سود کند و میزان حداکثر پرافیت هم همین مقدار و یا مقادیر کم بود و این باعث میشد که حداقل یکی از معاملات وجود داشته باشد که به سرعت به سود برسد و بسته شود و تاریخ پایان آن کوتاه باشد و به عنوان معامله ی مد نظر انتخاب شود. سایر معاملات که در مدت کوتاه به سود نرسند هم روند بهینه سازی به سمتی رفت که به راحتی و در مدت کوتاه بسته نشوند تا به عنوان معامله ی مد نظر انتخاب نشوند. مثلا حد ضرر بسیار بزرگ شد. تعداد روز های مجاز معامله بسیار زیاد شد و ... .

راه حل جایگزین پیشنهادی ، به جای در نظر گرفتن تعداد معامله ی همزمان مجاز ، تمامی معاملات همزمان به نسبت همین تعداد در اثر گذاری سود تاثیر داشته باشند برای مثال اگر 3 معامله ی همزمان در یک روز وجود دارد که اولی 5 درصد در 5 روز ، دومی -10 درصد در 2 روز و سومی 20 درصد در 10 روز میباشد ، سود روز جاری معادل 1 درصد برای اولی بعلاوه -5 درصد برای دومی بعلاوه ی 2درصد برای سومی باشد که در مجموع -2 درصد در این روز میشود.

good = []  
max\_pr = 0.0  
  
  
def check(ticks: list[TickerInfo], is\_individual: bool,  
 buy\_vol, buy\_pow, asc, buy\_gap,  
 sell\_vol, sell\_pow, des, sell\_gap,  
 days, tp, sl):  
 global max\_pr  
 global good  
 d\_type = [('start', int), ('end', int), ('profit', float)]  
 values = []  
 profit = 0.0  
 # file\_name = ''  
 # if is\_individual:  
 # file\_name = file\_name + 'I'  
 # else:  
 # file\_name = file\_name + 'C'  
 # file\_name = file\_name + 'bv' + str(buy\_vol)  
 # file\_name = file\_name + 'bp' + str(buy\_pow)  
 # file\_name = file\_name + 'bg' + str(buy\_gap)  
 # if asc:  
 # file\_name = file\_name + 'Tasc'  
 # else:  
 # file\_name = file\_name + 'Fasc'  
 # file\_name = file\_name + 'sv' + str(sell\_vol)  
 # file\_name = file\_name + 'sp' + str(sell\_pow)  
 # file\_name = file\_name + 'sg' + str(sell\_gap)  
 # if des:  
 # file\_name = file\_name + 'Tdes'  
 # else:  
 # file\_name = file\_name + 'Fdes'  
 # file\_name = file\_name + 'days' + str(days)  
 # file\_name = file\_name + 'tp' + str(tp)  
 # file\_name = file\_name + 'sl' + str(sl)  
 # with open(file\_name + '.csv', mode='w', newline='', encoding='utf-8') as file:  
 # writer = csv.writer(file, delimiter=',', quotechar='|', quoting=csv.QUOTE\_MINIMAL)  
 # writer.writerow(['is individual', 'buy volume', 'buy power', 'buy gap', 'ascending',  
 # 'sell volume', 'sell power', 'sell gap', 'descending', 'max days', 'take profit', 'stop loss'])  
 # writer.writerow(  
 # [is\_individual, buy\_vol, buy\_pow, buy\_gap, asc, sell\_vol, sell\_pow, sell\_gap, des, days, tp, sl])  
 # writer.writerow(['ticker', 'buy date', 'sell date', 'reason', 'buy', 'sell', 'profit'])  
 for tick in ticks:  
 bool\_temp1 = False  
 if is\_individual:  
 if float(tick.history.vol.sum()) / float(  
 float(tick.history.buy\_vol.sum()) + float(tick.history.sell\_vol.sum())) < 1.0:  
 bool\_temp1 = True  
 else:  
 bool\_temp1 = True  
 if bool\_temp1:  
 price = 0.0  
 date1 = None  
 d = 0  
 for i in range(tick.start, len(tick.history.date) - 1):  
 if price == 0.0:  
 if float(tick.history.vol[i] - tick.mean\_vol) / float(tick.vol\_std) >= buy\_vol:  
 bool\_temp2 = False  
 if tick.history.buy\_count[i] == -1 or tick.history.buy\_count[i] == 0:  
 pass  
 elif tick.history.sell\_count[i] == 0 or tick.history.sell\_vol[i] == 0:  
 bool\_temp2 = True  
 elif (float(tick.history.buy\_vol[i]) / float(tick.history.buy\_count[i])) / (  
 float(tick.history.sell\_vol[i]) / float(tick.history.sell\_count[i])) >= buy\_pow:  
 bool\_temp2 = True  
 if bool\_temp2:  
 temp\_asc = False  
 if asc:  
 if tick.history.low[i] < tick.history.open[i] < tick.history.close[i]:  
 temp\_asc = True  
 else:  
 temp\_asc = True  
 if temp\_asc:  
 gap = 0  
 if tick.history.open[i + 1] > tick.history.high[i]:  
 gap = 2  
 elif tick.history.open[i + 1] > tick.history.close[i]:  
 gap = 1  
 elif tick.history.open[i + 1] < tick.history.low[i]:  
 gap = -2  
 elif tick.history.open[i + 1] < tick.history.close[i]:  
 gap = -1  
 if gap >= buy\_gap:  
 d = 0  
 price = tick.history.open[i + 1]  
 date1 = tick.history.date[i + 1]  
 else:  
 d = d + 1  
 if d > days:  
 profit = ((float(tick.history.open[i] - price) / price) \* 100.0) - 1.23  
 values.append((date1, tick.history.date[i], profit))  
 # writer.writerow(  
 # [tick.ticker, date1, tick.history.date[i], 'max days', price, tick.history.open[i],  
 # profit])  
 price = 0.0  
 elif (float(price - tick.history.low[i]) / price) \* 100.0 >= sl:  
 profit = ((float(tick.history.low[i] - price) / price) \* 100.0) - 1.23  
 values.append((date1, tick.history.date[i], profit))  
 # writer.writerow(  
 # [tick.ticker, date1, tick.history.date[i], 'stop loss', price, tick.history.low[i],  
 # profit])  
 price = 0.0  
 elif (float(tick.history.high[i] - price) / price) \* 100 >= tp:  
 profit = tp - 1.23  
 values.append((date1, tick.history.date[i], profit))  
 # writer.writerow([tick.ticker, date1, tick.history.date[i], 'take profit', price,  
 # tick.history.high[i], profit])  
 price = 0.0  
 else:  
 if float(tick.history.vol[i] - tick.mean\_vol) / tick.vol\_std >= sell\_vol:  
 bool\_temp3 = False  
 if tick.history.sell\_count[i] == -1 or tick.history.sell\_count[i] == 0:  
 pass  
 elif tick.history.buy\_vol[i] == 0 or tick.history.buy\_count[i] == 0:  
 bool\_temp3 = True  
 elif (float(tick.history.sell\_vol[i]) / float(tick.history.sell\_count[i])) / (  
 float(tick.history.buy\_vol[i]) / float(tick.history.buy\_count[i])) >= sell\_pow:  
 bool\_temp3 = True  
 if bool\_temp3:  
 profit = ((float(tick.history.open[i + 1] - price) / price) \* 100.0) - 1.23  
 values.append((date1, tick.history.date[i + 1], profit))  
 # writer.writerow([tick.ticker, date1, tick.history.date[i + 1], 'sell power', price,  
 # tick.history.open[i + 1], profit])  
 price = 0.0  
 elif (tick.history.high[i] > tick.history.open[i] > tick.history.close[i]) and des:  
 profit = ((float(tick.history.open[i + 1] - price) / price) \* 100.0) - 1.23  
 values.append((date1, tick.history.date[i + 1], profit))  
 # writer.writerow([tick.ticker, date1, tick.history.date[i + 1], 'descending', price,  
 # tick.history.open[i + 1], profit])  
 price = 0.0  
 else:  
 gap = 0  
 if tick.history.open[i + 1] > tick.history.high[i]:  
 gap = 2  
 elif tick.history.open[i + 1] > tick.history.close[i]:  
 gap = 1  
 elif tick.history.open[i + 1] < tick.history.low[i]:  
 gap = -2  
 elif tick.history.open[i + 1] < tick.history.close[i]:  
 gap = -1  
 if gap <= sell\_gap:  
 profit = ((float(tick.history.open[i + 1] - price) / price) \* 100.0) - 1.23  
 values.append((date1, tick.history.date[i + 1], profit))  
 # writer.writerow([tick.ticker, date1, tick.history.date[i + 1], 'gap', price,  
 # tick.history.open[i + 1], profit])  
 price = 0.0  
 a = np.array(values, dtype=d\_type)  
 a.sort(order='start')  
 resume = True  
 split = 0  
 while resume:  
 split = split + 1  
 if split > 36:  
 break  
 resume = False  
 dates = []  
 pr = 0  
 for s, e, p in a:  
 active\_trades = 0  
 for ds, de in dates:  
 if ds <= s <= de:  
 active\_trades = active\_trades + 1  
 if active\_trades < split:  
 dates.append((s, e))  
 pr = pr + p  
 else:  
 resume = True  
 print(is\_individual, buy\_vol, buy\_pow, asc, buy\_gap, sell\_vol, sell\_pow, des, sell\_gap, days, tp, sl, split, pr/float(split))  
 if pr/float(split) >= 110:  
 print("good")  
 good.append((is\_individual, buy\_vol, buy\_pow, asc, buy\_gap, sell\_vol, sell\_pow, des, sell\_gap, days, tp, sl, split, pr/float(split)))

تابع تعیین تاریخ شروع بهینه سازی:

این تابع در تعداد کمی از سهم ها به درستی عمل نمیکندو آن هم برای سهم های هست که برای مدتی طولانی در بازه ی پایانی ، اطلاعاتی از آن ها وجود ندارد و شرط آخر هیچ گاه صادق نمیشود.

این سهم ابتدا در صورتی که اطلاعات سهم به صورت کامل ثبت نشده باشد یا دانلود نشده باشد ، تاریخچه ی قبل را نادیده میکند.

همچنین این تابع برای مدت زمان عرضه ی اولیه شدن سهم تاریخچه را نادیده میکند تا در روند بهینه سازی اثر نکند(صف فروش و صف خرید که مانع از فروش و خرید میشود در بهینه سازی گنجانده نشده است ، در صورتی که این الگوریتم اضافه شود ، نیاز به این قسمت از تابع نیست.)

def change\_tickers(tickers: list[TickerInfo]):  
 for tic in tickers:  
 date = 0  
 for i in range(len(tic.history.date)):  
 if (tic.history.open[i] == -1 or tic.history.high[i] == -1 or tic.history.low[i] == -1 or  
 tic.history.close[i] == -1 or tic.history.buy\_vol[i] == -1 or tic.history.buy\_count[i] == -1 or  
 tic.history.sell\_vol[i] == -1 or tic.history.sell\_count[i] == -1 or tic.history.vol[i] == -1):  
 if tic.history.date[i] > date:  
 date = tic.history.date[i]  
 tic.start = i + 1  
 for i in range(len(tic.history.date) - 1):  
 if tic.history.close[i + 1] < tic.history.close[i]:  
 if tic.start < i + 1:  
 tic.start = i + 1  
 break  
 for i in range(len(tic.history.date) - 1):  
 if tic.history.date[i] < 20231220 <= tic.history.date[i + 1]:  
 if i > tic.start:  
 tic.start = i  
 break

فایل اجرای بهینه سازی:

در این خطوط ابتدا ماژول ها ایمپورت میشوند:

import TSE  
import csv

در ادامه با استفاده از توابع ماژول ها لیست اطلاعات سهم ها بدست می آید:

tickers = TSE.read\_tickers\_from\_files(378)  
  
TSE.change\_tickers(tickers)  
c = 0

فایلی حامل اطلاعات مربوط به حالت های بهینه سازی از کاربر گرفته میشود و اطلاعات آن خوانده میشود و در صورت موفقیت آمیز بودن ارروری چاپ نمیشود.

with open(name, mode='r', encoding='utf-8', newline='') as file:  
 reader = csv.reader(file, delimiter=',', quotechar='|', quoting=csv.QUOTE\_MINIMAL)  
 for row in reader:  
 if row[0] == "True":  
 is\_individual\_i = True  
 elif row[0] == "False":  
 is\_individual\_i = False  
 else:  
 print("error0")  
 print(row[0])  
 time.sleep(10)  
 if row[3] == "True":  
 asc\_i = True  
 elif row[3] == "False":  
 asc\_i = False  
 else:  
 print("error3")  
 print(row[3])  
 time.sleep(10)  
 if row[7] == "True":  
 des\_i = True  
 elif row[7] == "False":  
 des\_i = False  
 else:  
 print("error7")  
 print(row[7])  
 time.sleep(10)  
 buy\_vol\_i = float(row[1])  
 buy\_pow\_i = float(row[2])  
 buy\_gap\_i = int(row[4])  
 sell\_vol\_i = float(row[5])  
 sell\_pow\_i = float(row[6])  
 sell\_gap\_i = int(row[8])  
 days\_i = int(row[9])  
 tp\_i = float(row[10])  
 sl\_i = float(row[11]

در انتها با استفاده از دستور چک که در ماژول وجود دارد میزان سود آن الگوریتم محاسبه میشود و در صورت مطلوب بودن در آرائه ای در خود ماژول ذخیره میشود:

TSE.check(tickers, is\_individual=is\_individual\_i, buy\_vol=buy\_vol\_i, buy\_pow=buy\_pow\_i, asc=asc\_i, buy\_gap=buy\_gap\_i, sell\_vol=sell\_vol\_i, sell\_pow=sell\_pow\_i, des=des\_i, sell\_gap=sell\_gap\_i, days=days\_i, tp=tp\_i, sl=sl\_i)  
c = c + 1  
print(c)  
#li.append([is\_individual\_i,buy\_vol\_i,buy\_pow\_i,asc\_i,buy\_gap\_i,sell\_vol\_i,sell\_pow\_i,des\_i,sell\_gap\_i,days\_i,tp\_i,sl\_i])

در انتها ارایه ی مطلوب به صورت یک فایل نتایج خروجی داده میشود:

with open('res\_'+name, mode='w', newline='', encoding='utf-8') as file:  
 writer = csv.writer(file, delimiter=',', quotechar='|', quoting=csv.QUOTE\_MINIMAL)  
 for i in TSE.good:  
 writer.writerow([i[0], i[1], i[2], i[3], i[4], i[5], i[6], i[7], i[8], i[9], i[10], i[11], i[12], i[13]])

الگوریتم حذف حالات اضافی:

الگوریتم زیر حالات تکراری در نتایج را حذف میکند ، حالات تکراری زمانی رخ میدهد که برای یک حالت بهینه سازی چند مورد تست مربوط به تعداد معاملات همزمان مطلوب شده باشد. در الگوریتم جدید که نیاز به نوشته شدن دارد نیازی به انجام این کار بعد از دریافت نتایج نمیباشد:

import csv

res = []

last\_row = None

name = input("name:")

with open('res'+name+'.csv', mode='r', encoding='utf-8', newline='') as file:

reader = csv.reader(file, delimiter=',', quotechar='|', quoting=csv.QUOTE\_MINIMAL)

for row in reader:

if last\_row == None:

last\_row = [row[0],row[1],row[2],row[3],row[4],row[5],row[6],row[7],row[8],row[9],row[10],row[11]]

res.append(last\_row)

else:

con = True

for i in range(12):

con = con and (row[i] == last\_row[i])

if not con:

last\_row = [row[0],row[1],row[2],row[3],row[4],row[5],row[6],row[7],row[8],row[9],row[10],row[11]]

res.append(last\_row)

with open('res2'+name+'.csv', mode='w', newline='', encoding='utf-8') as file2:

writer = csv.writer(file2, delimiter=',', quotechar='|', quoting=csv.QUOTE\_MINIMAL)

for i in res:

writer.writerow([i[0], i[1], i[2], i[3], i[4], i[5], i[6], i[7], i[8], i[9], i[10], i[11]])

الگوریتم ساخت حالت های همسایگی حالت های مطلوب برای بهینه سازی در مرحله ی بعد:

بسته به اینکه چه فایل هایی طبق اکسل تا کنون ساخته شده باشد حالات همسایگی با حذف حالات تکراری (یک همسایگی برای دو حالت مطلوب) ساخته میشود . کامل ترین حالت الگوریتم زیر است و انتخاب الگوریتم از روی اکسل قابل خواندن است:

import csv

import time

s = input("series:")

vm5 = input("v-5:")

vxx = input("vxx:")

vp5 = input("v+5:")

hm5 = input("h-5:")

hxx = input("hxx:")

hp5 = input("h+5:")

sxxxx = []

sxxm5 = []

sxxp5 = []

sm5xx = []

sp5xx = []

nsxxxx = []

def read\_file(li, name):

with open(name, mode='r', encoding='utf-8', newline='') as file:

reader = csv.reader(file, delimiter=',', quotechar='|', quoting=csv.QUOTE\_MINIMAL)

for row in reader:

if row[0] == "True" or row[0] == "TRUE":

is\_individual\_i = True

elif row[0] == "False" or row[0] == "FALSE":

is\_individual\_i = False

else:

print("error0")

print(row[0])

time.sleep(10)

if row[3] == "True" or row[3] == "TRUE":

asc\_i = True

elif row[3] == "False" or row[3] == "FALSE":

asc\_i = False

else:

print("error3")

print(row[3])

time.sleep(10)

if row[7] == "True" or row[7] == "TRUE":

des\_i = True

elif row[7] == "False" or row[7] == "FALSE":

des\_i = False

else:

print("error7")

print(row[7])

time.sleep(10)

buy\_vol\_i = float(row[1])

buy\_pow\_i = float(row[2])

buy\_gap\_i = int(row[4])

sell\_vol\_i = float(row[5])

sell\_pow\_i = float(row[6])

sell\_gap\_i = int(row[8])

days\_i = int(row[9])

tp\_i = float(row[10])

sl\_i = float(row[11])

li.append(

[is\_individual\_i, buy\_vol\_i, buy\_pow\_i, asc\_i, buy\_gap\_i, sell\_vol\_i, sell\_pow\_i, des\_i, sell\_gap\_i,

days\_i, tp\_i, sl\_i])

read\_file(sxxxx, s + vxx + hxx + ".csv")

read\_file(sm5xx, s + vm5 + hxx + ".csv")

read\_file(sp5xx, s + vp5 + hxx + ".csv")

read\_file(sxxm5, s + vxx + hm5 + ".csv")

read\_file(sxxp5, s + vxx + hp5 + ".csv")

for ri in sxxxx:

if ri not in nsxxxx:

nsxxxx.append(ri)

ri1 = [ri[0], ri[1], ri[2], not (ri[3]), ri[4], ri[5], ri[6], ri[7], ri[8], ri[9], ri[10], ri[11]]

if ri1 not in nsxxxx:

nsxxxx.append(ri1)

if ri[4] == -2:

ri2 = [ri[0], ri[1], ri[2], ri[3], -1, ri[5], ri[6], ri[7], ri[8], ri[9], ri[10], ri[11]]

if ri2 not in sxxxx:

nsxxxx.append(ri2)

elif ri[4] == 2:

ri2 = [ri[0], ri[1], ri[2], ri[3], 1, ri[5], ri[6], ri[7], ri[8], ri[9], ri[10], ri[11]]

if ri2 not in nsxxxx:

nsxxxx.append(ri2)

else:

ri2 = [ri[0], ri[1], ri[2], ri[3], ri[4] - 1, ri[5], ri[6], ri[7], ri[8], ri[9], ri[10], ri[11]]

ri3 = [ri[0], ri[1], ri[2], ri[3], ri[4] + 1, ri[5], ri[6], ri[7], ri[8], ri[9], ri[10], ri[11]]

if ri2 not in nsxxxx:

nsxxxx.append(ri2)

if ri3 not in nsxxxx:

nsxxxx.append(ri3)

ri4 = [ri[0], ri[1], ri[2], ri[3], ri[4], ri[5] - 0.5, ri[6], ri[7], ri[8], ri[9], ri[10], ri[11]]

if ri4 not in nsxxxx:

nsxxxx.append(ri4)

ri5 = [ri[0], ri[1], ri[2], ri[3], ri[4], ri[5] + 0.5, ri[6], ri[7], ri[8], ri[9], ri[10], ri[11]]

if ri5 not in nsxxxx:

nsxxxx.append(ri5)

ri6 = [ri[0], ri[1], ri[2], ri[3], ri[4], ri[5], ri[6] - 0.5, ri[7], ri[8], ri[9], ri[10], ri[11]]

if ri6 not in nsxxxx:

nsxxxx.append(ri6)

ri7 = [ri[0], ri[1], ri[2], ri[3], ri[4], ri[5], ri[6] + 0.5, ri[7], ri[8], ri[9], ri[10], ri[11]]

if ri7 not in nsxxxx:

nsxxxx.append(ri7)

ri8 = [ri[0], ri[1], ri[2], ri[3], ri[4], ri[5], ri[6], not (ri[7]), ri[8], ri[9], ri[10], ri[11]]

if ri8 not in nsxxxx:

nsxxxx.append(ri8)

if ri[8] == -2:

ri9 = [ri[0], ri[1], ri[2], ri[3], ri[4], ri[5], ri[6], ri[7], -1, ri[9], ri[10], ri[11]]

if ri9 not in nsxxxx:

nsxxxx.append(ri9)

elif ri[8] == 2:

ri9 = [ri[0], ri[1], ri[2], ri[3], ri[4], ri[5], ri[6], ri[7], 1, ri[9], ri[10], ri[11]]

if ri9 not in nsxxxx:

nsxxxx.append(ri9)

else:

ri9 = [ri[0], ri[1], ri[2], ri[3], ri[4], ri[5], ri[6], ri[7], ri[8] - 1, ri[9], ri[10], ri[11]]

ri10 = [ri[0], ri[1], ri[2], ri[3], ri[4], ri[5], ri[6], ri[7], ri[8] + 1, ri[9], ri[10], ri[11]]

if ri9 not in nsxxxx:

nsxxxx.append(ri9)

if ri10 not in nsxxxx:

nsxxxx.append(ri10)

ri11 = [ri[0], ri[1], ri[2], ri[3], ri[4], ri[5], ri[6], ri[7], ri[8], ri[9] + 3, ri[10], ri[11]]

if ri11 not in nsxxxx:

nsxxxx.append(ri11)

ri12 = [ri[0], ri[1], ri[2], ri[3], ri[4], ri[5], ri[6], ri[7], ri[8], ri[9] - 3, ri[10], ri[11]]

if ri12 not in nsxxxx:

nsxxxx.append(ri12)

ri13 = [ri[0], ri[1], ri[2], ri[3], ri[4], ri[5], ri[6], ri[7], ri[8], ri[9], ri[10] + 5.0, ri[11]]

if ri13 not in nsxxxx:

nsxxxx.append(ri13)

ri14 = [ri[0], ri[1], ri[2], ri[3], ri[4], ri[5], ri[6], ri[7], ri[8], ri[9], ri[10] - 5.0, ri[11]]

if ri14 not in nsxxxx:

nsxxxx.append(ri14)

ri15 = [ri[0], ri[1], ri[2], ri[3], ri[4], ri[5], ri[6], ri[7], ri[8], ri[9], ri[10], ri[11] + 5.0]

if ri15 not in nsxxxx:

nsxxxx.append(ri15)

ri16 = [ri[0], ri[1], ri[2], ri[3], ri[4], ri[5], ri[6], ri[7], ri[8], ri[9], ri[10], ri[11] - 5.0]

if ri16 not in nsxxxx:

nsxxxx.append(ri16)

for ri in sxxm5:

ri[2] = ri[2] + 0.5

if ri not in nsxxxx:

nsxxxx.append(ri)

for ri in sxxp5:

ri[2] = ri[2] - 0.5

if ri not in nsxxxx:

nsxxxx.append(ri)

for ri in sm5xx:

ri[1] = ri[1] + 0.5

if ri not in nsxxxx:

nsxxxx.append(ri)

for ri in sp5xx:

ri[1] = ri[1] - 0.5

if ri not in nsxxxx:

nsxxxx.append(ri)

with open("x" + s + vxx + hxx + ".csv", mode='w', newline='', encoding='utf-8') as file:

writer = csv.writer(file, delimiter=',', quotechar='|', quoting=csv.QUOTE\_MINIMAL)

for i in nsxxxx:

writer.writerow(i)