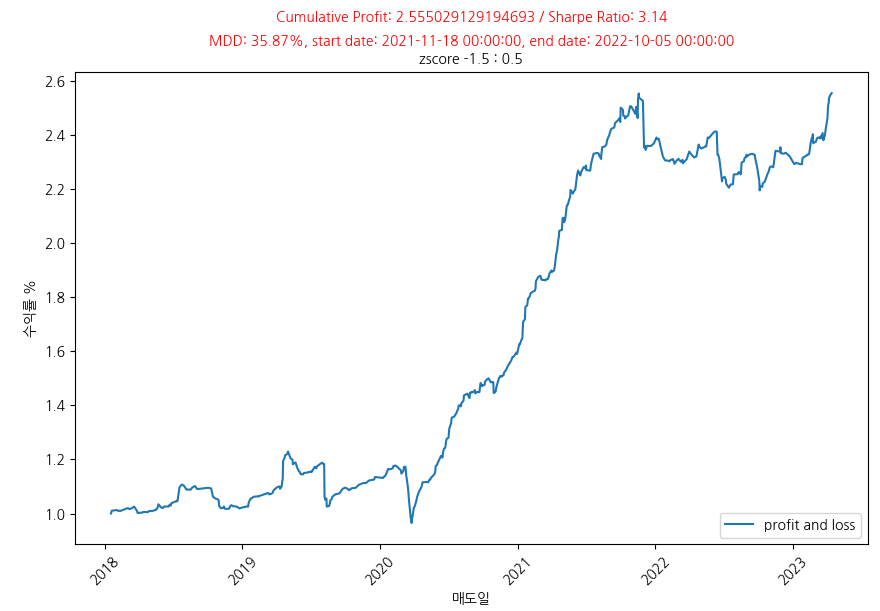
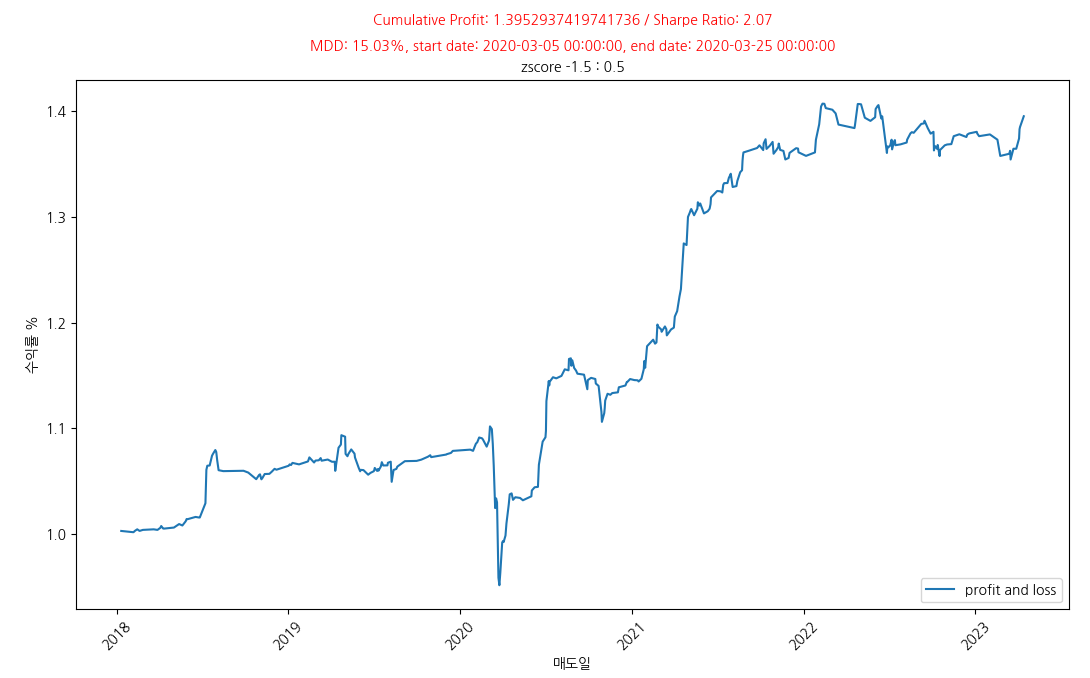


CAGR : 12.23%



CAGR : 20.64%



CAGR : 6.89%

Code

from pykrx import stock

from datetime import datetime, timedelta

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

import matplotlib.ticker as ticker

import pandas as pd

import statsmodels.api as sm

from pykrx import stock

import datetime as dt

# kospi.csv 파일 읽기

kospi\_data = pd.read\_csv("kospidata.csv", dtype={"code": str})

# sp500data.csv 파일 읽기

sp500\_data = pd.read\_csv("sp500data.csv")

# S&P500 데이터에서 필요한 컬럼만 추출

sp500\_data = sp500\_data[["Date", "Close"]]

# 한국 휴장일

khday= pd.read\_excel('kholiday.xls')

sp500\_data['Date'] = pd.to\_datetime(sp500\_data['Date'].astype(str).str[:10])

sp500\_data['Date'] = pd.to\_datetime(sp500\_data['Date'], format='%Y-%m-%d')  # Date 컬럼을 datetime으로 변환

khday['날짜'] = pd.to\_datetime(khday['날짜'])

for i, date in enumerate(sp500\_data['Date']):

    if date.weekday() == 4:  # 금요일인 경우

        sp500\_data.loc[i, 'Date'] += pd.Timedelta(days=3)

    elif date.weekday() != 4:

        sp500\_data.loc[i, 'Date'] += pd.Timedelta(days=1)

for i, date in enumerate(sp500\_data['Date']):

    if date.strftime('%Y-%m-%d') in khday['날짜'].dt.strftime('%Y-%m-%d').tolist():

        sp500\_data.loc[i, 'Date'] += pd.Timedelta(days=1)

for i, date in enumerate(sp500\_data['Date']):

    if date.weekday() == 5:  # 토요일인 경우

        sp500\_data.loc[i, 'Date'] += pd.Timedelta(days=2)

# Date 기준으로 정렬

sp500\_data = sp500\_data.sort\_values(by="Date")

#중복이 있을경우 첫번째꺼를 제거 하는 방식 // 수익이 별로면 두번째로 바꿔보자

sp500\_data = sp500\_data.drop\_duplicates(subset='Date', keep='last')

# 거래량이 0인 날짜를 찾아서 해당 종목의 데이터 삭제

for code in kospi\_data["code"].unique():

    stock\_data = kospi\_data[kospi\_data["code"] == code][["날짜", "종가", "거래량"]]

    zero\_volume\_count = (stock\_data["거래량"] == 0).sum()

    if zero\_volume\_count >= 20:

        kospi\_data = kospi\_data[kospi\_data["code"] != code]

# 코스피 종목별로 cointegration 검증

result = []

for code in kospi\_data["code"].unique():

    # 해당 종목의 데이터 추출

    stock\_data = kospi\_data[kospi\_data["code"] == code][["날짜", "종가"]]

    stock\_data["날짜"] = pd.to\_datetime(stock\_data["날짜"], format='%Y-%m-%d')

    # 데이터를 날짜 기준으로 merge

    merged\_data = pd.merge(sp500\_data, stock\_data, left\_on="Date", right\_on="날짜", how="inner")

    # cointegration 검증

    \_, pvalue, \_ = sm.tsa.coint(merged\_data["Close"], merged\_data["종가"])

    # p-value가 0.05보다 작으면 cointegration이 있다고 판단

    if pvalue < 0.03:

        code.astype(str)

        name = stock.get\_market\_ticker\_name(code)

        result.append([name, code, pvalue])

# 결과 출력

result\_df = pd.DataFrame(result, columns=["종목", "코드", "P-value"])

result\_df = result\_df.dropna() # NaN 값이 있는 행 제거

result\_df = result\_df.reset\_index(drop=True) # 인덱스 재설정

result\_df.to\_csv('kcointegration.csv',encoding='utf-8-sig')

#high cointegration 코스피, S&P 데이터

kospi=pd.read\_csv('kcointegration.csv')

Sp500=pd.read\_csv('sp500modify.csv')

Sp500['Date'] = pd.to\_datetime(Sp500['Date'], format='%Y-%m-%d')

tickers = stock.get\_market\_ticker\_list("20230225")

kospi['코드']=kospi['코드'].astype(str)

result = []

for code in kospi['코드']:

    code = str(code).zfill(6)

    kospi\_data = stock.get\_market\_ohlcv('20180102', '20230419', code)

    kospi\_data = kospi\_data.reset\_index()

    kospi\_data['종목'] = stock.get\_market\_ticker\_name(code)

    # S&P500 데이터와 합치기

    kospi\_data = pd.merge(kospi\_data, Sp500, how='inner', left\_on='날짜', right\_on='Date')

    # 등락률 계산

    kospi\_data['kospi\_rtn'] = kospi\_data['종가'].pct\_change()

    kospi\_data['sp500\_rtn'] = kospi\_data['Close'].pct\_change()

    # spread 계산

    kospi\_data['spread'] = kospi\_data['kospi\_rtn'] - kospi\_data['sp500\_rtn']

    # z-score 계산

    kospi\_data['zscore'] = (kospi\_data['spread'] - np.mean(kospi\_data['spread'])) / np.std(kospi\_data['spread'])

    result.append(kospi\_data)

# 종목별 결과 합치기

df = pd.concat(result, axis=0)

df = df[['날짜', '시가', '종가', '거래량', '종목', 'kospi\_rtn', 'sp500\_rtn', 'spread', 'zscore']]

df.to\_csv('kzscore.csv', encoding='utf-8-sig')

# zscore 데이터 불러오기

zscore = pd.read\_csv('kzscore.csv')

# signal 컬럼 추가

signal = pd.DataFrame(0, index=zscore.index, columns=['signal'])

zscore['signal'] = signal

#buy,sell signal 설정

buy\_signal = sell\_signal = 0

#저장할 딕셔너리

trades = {}

for name in zscore['종목'].unique():

    # 종목별 데이터 추출

    name\_data = zscore[zscore['종목'] == name]

    trades[name] = []

    # buy signal, sell signal 구하기

    for i in range(len(name\_data)):

        zscore\_value = name\_data.loc[name\_data.index[i], 'zscore']

        if zscore\_value <= -1.5:

            # 이미 buy\_signal이 존재하는 경우는 추가적인 buy\_signal 생략

            if buy\_signal == 0:

                name\_data.loc[name\_data.index[i], 'signal'] = 1

                buy\_signal = 1

                sell\_signal = 0 # sell signal 초기화

        elif zscore\_value >= 0.5:

            # sell\_signal 발생

            if buy\_signal == 1 and sell\_signal == 0:

                name\_data.loc[name\_data.index[i], 'signal'] = -1

                sell\_signal = 1

                buy\_signal = 0 # buy signal 초기화

        elif buy\_signal == 1 and sell\_signal == 0:

            # 스탑로스

            last\_buy\_trade = trades[name][-1] if len(trades[name]) > 0 else None

            if last\_buy\_trade is not None:

                last\_buy\_price = last\_buy\_trade['price']

                current\_price = name\_data.loc[name\_data.index[i], '종가']

                if current\_price <= last\_buy\_price \* 0.9:

                    name\_data.loc[name\_data.index[i], 'signal'] = -1

                    sell\_signal = 1

                    buy\_signal = 0

                elif current\_price>= last\_buy\_price \*0.9:

                    continue

        # buy\_signal이 존재하는 경우, sell\_signal이 나오기 전까지 signal 생략

        else:

            continue

    # 결과를 zscore 데이터프레임에 저장

    zscore[zscore['종목'] == name] = name\_data

    for i in range(1, len(name\_data)):

        # buy signal이 나온 다음날 시가에 매수

        if name\_data.loc[name\_data.index[i-1], 'signal'] == 1:

            buy\_price = name\_data.loc[name\_data.index[i], '시가']

            buy\_date = name\_data.loc[name\_data.index[i], '날짜']

            trades[name].append({'date': buy\_date, 'price': buy\_price})

        # sell signal이 나온 다음날 시가에 매도

        elif name\_data.loc[name\_data.index[i-1], 'signal'] == -1:

            # get the latest buy trade for this stock

            last\_buy\_trade = trades[name][-1] if len(trades[name]) > 0 else None

            if last\_buy\_trade is not None: # buy signal에 대한 매수가 이루어진 경우에만 매도

                sell\_price = name\_data.loc[name\_data.index[i], '시가']

                sell\_date = name\_data.loc[name\_data.index[i], '날짜']

                # calculate and store the profit for this trade

                profit = ((sell\_price - last\_buy\_trade['price']) / last\_buy\_trade['price'])

                trades[name].append({'sell\_date': sell\_date, 'sell\_price': sell\_price, 'profit' : profit})

#trades를 리스트의 리스트 데이타프레임으로 바꾸기

trades\_list = []

for name in trades:

    if trades[name]:

        for trade in trades[name]:

            trades\_list.append([name, trade.get('date', None), trade.get('price', None), trade.get('sell\_date', None), trade.get('sell\_price', None), trade.get('profit', None)])

trades\_df = pd.DataFrame(trades\_list, columns=['종목', '매수일', '매수가', '매도일', '매도가', '수익률'])

trades\_df['매수일'] = pd.to\_datetime(trades\_df['매수일'])

trades\_df['매도일'] = pd.to\_datetime(trades\_df['매도일'])

trades\_df['수익률'] = trades\_df['수익률'] \* 100

print(trades\_df)

trades\_df.to\_csv('코스피결과.csv',encoding='utf-8-sig')

result=pd.read\_csv('코스피결과.csv')

# 매도일 기준으로 정렬

result\_sorted = result.sort\_values(by='매도일')

result\_sorted = result\_sorted.dropna(subset=['매도일'])

# 중복 없는 매도일 데이터 가져오기

dates = result\_sorted['매도일'].unique().astype(str)

# str에서 날짜로 변환

dates = [datetime.strptime(date, '%Y-%m-%d') for date in dates]

# 날짜순으로 정렬

dates.sort()

# 각 날짜별 수익률 합산하기

cumulative\_profit = []

grouped = result\_sorted.groupby('매도일')

for date, group in grouped:

    profit\_sum = group['수익률'].sum()

    profit = profit\_sum / 100.0 / 15.0

    cumulative\_profit.append(cumulative\_profit[-1] \* (1 + profit) if cumulative\_profit else 1.0 + profit)

print(cumulative\_profit[-1])

#MDD

max\_profit = 0.0

drawdown = 0.0

max\_drawdown = 0.0

mdd\_start\_date = None

mdd\_end\_date = None

for date, profit in zip(dates, cumulative\_profit):

    if profit > max\_profit:

        max\_profit = profit

        drawdown = 0.0

    else:

        drawdown = max(drawdown, max\_profit - profit)

        if drawdown > max\_drawdown:

            max\_drawdown = drawdown

            mdd\_start\_date = dates[cumulative\_profit.index(max\_profit)]

            mdd\_end\_date = date

mdd\_text = f'MDD: {max\_drawdown:.2%}, start date: {mdd\_start\_date}, end date: {mdd\_end\_date}'

print(f'MDD: {max\_drawdown:.2%}, start date: {mdd\_start\_date}, end date: {mdd\_end\_date}')

cumpro\_text = f'Cumulative Profit: {cumulative\_profit[-1]}'

#Sharpe ratio

profits = pd.DataFrame({'date': dates, 'profit': cumulative\_profit})

profits['daily\_return'] = profits['profit'].pct\_change()

volatility = profits['daily\_return'].std()

#변동성

annual\_volatility = volatility \* np.sqrt(252)

annual\_return = (cumulative\_profit[-1] \*\* (252/len(dates))) - 1

risk\_free\_rate = 0.03

sharpe\_ratio = (annual\_return - risk\_free\_rate) / annual\_volatility

sharpe\_text = f'Sharpe Ratio: {sharpe\_ratio:.2f}'

print(sharpe\_text)

# 그래프 그리기

plt.rc('font', family='NanumGothic')

plt.plot(dates, cumulative\_profit, label='profit and loss')

plt.legend(loc='lower right')

plt.xlabel('매도일')

plt.ylabel('수익률 %')

plt.xticks(rotation=45)

plt.text(0.5, 1.05, mdd\_text, transform=plt.gca().transAxes, ha='center', va='bottom', color='r', fontsize=10)

plt.text(0.5,1.1,cumpro\_text + ' / '+ sharpe\_text, transform=plt.gca().transAxes, ha='center', va='bottom', color='r', fontsize=10)

plt.title('zscore -1.5 : 0.5', fontsize=10)

plt.show()