def getBreakthroughPoint(df, col1, col2, patient\_days, fill\_method="fb"):

'''

:param df: dataframe (including col1, col2)

:param col1: obj

:param col2: obj moving average

:param patient\_days: patient days detected as breakthrough point

:return: signal series

'''

sigPrice = []

flag = -1 # A flag for the trend upward/downward

for i in range(0, len(df)):

if df[col1][i] > df[col2][i] and flag != 1:

tmp = df['Close'][i:(i + patient\_days + 1)]

if len(tmp) == 1:

sigPrice.append("buy")

flag = 1

else:

if (tmp.iloc[1:] > tmp.iloc[0]).all():

sigPrice.append("buy")

flag = 1

else:

sigPrice.append(nan)

elif df[col1][i] < df[col2][i] and flag != 0:

tmp = df['Close'][i:(i + patient\_days + 1)]

if len(tmp) == 1:

sigPrice.append("sell")

flag = 0

else:

if (tmp.iloc[1:] < tmp.iloc[0]).all():

sigPrice.append("sell")

flag = 0

else:

sigPrice.append(nan)

else:

sigPrice.append(nan)

sigPrice = series(sigPrice)

for idx, value in enumerate(sigPrice):

if not isna(value):

if value == "buy":

sigPrice.iloc[1:idx] = "sell"

else:

sigPrice.iloc[1:idx] = "buy"

break

# if fill\_method == "bf":

#

# elif fill\_method == ""

sigPrice.ffill(inplace=True)

return sigPrice

def stochastic(df, n=14, m=5, t=5):

#데이터 프레임으로 받아오기 때문에 불필요

#n 일중 최저가

ndays\_high = df['High'].rolling(window=n, min\_periods=n).max()

ndays\_low = df['Low'].rolling(window=n, min\_periods=n).min()

fast\_k = ((df['Close'] - ndays\_low) / (ndays\_high - ndays\_low) \* 100)

slow\_k = fast\_k.ewm(span=m, min\_periods=m).mean()

slow\_d = slow\_k.ewm(span=t, min\_periods=t).mean()

df = df.assign(fast\_k=fast\_k, fast\_d=slow\_k, slow\_k=slow\_k, slow\_d=slow\_d)

return df

# ===== raw data loading =====

# 한 종목코드에 대한 주가 정보를 로드

# 임의 선별

# 삼성전자

# NAVER

# 카카오

# 랜덤 선별

# rnd.seed(48)

# stock\_list.iloc[rnd.randint(len(stock\_list))]

# 금호석유

# 티움바이오

# 테크윙

# 제테마

# 주성엔지니어링

# 고바이오랩

# 고영

# Get Stock List

path = 'projects/dacon\_stockprediction/open\_week4/'

list\_name = 'Stock\_List.csv'

sample\_name = 'sample\_submission\_week4.csv'

# raw features (5개)

# 주가, 거래량, 기관순매수, 외인순매수, 뉴스 기사(embedding)

# derived features (14개)

# 주가이평, 거래량이평, 기관순매수이평, 외인순매수이평, 뉴스 기사에 대한 긍부정점수, 요일, sin변환(5일), cos변환(5일)

# 산식 보조 지표

# 1. 주가 관련 지표 : Stochastic(20), RSI(20), 볼린저밴드(20)

# 2. 거래량 관련 지표 : OBV, VR(20)

# 3. 혼합지표 : MFI(주가 + 거래량)

# 종목 코드 로드

stock\_list = read\_csv(os.path.join(path, list\_name))

stock\_list['종목코드'] = stock\_list['종목코드'].apply(lambda x: str(x).zfill(6))

# Get Data & Modeling

# 분석할 date 변수 지정

start\_date = '20201201'

end\_date = '20211001'

start\_weekday = pd.to\_datetime(start\_date).weekday()

max\_weeknum = pd.to\_datetime(end\_date).strftime('%V')

business\_days = pd.DataFrame(pd.date\_range(start\_date, end\_date, freq='B'), columns=['Date'])

# 선택 종목

stock\_list.set\_index("종목명", inplace=True)

selected\_codes = ["삼성전자", "NAVER", "카카오", "금호석유", "티움바이오", "테크윙", "제테마", "주성엔지니어링", "고바이오랩", "고영"]

stock\_list = stock\_list.loc[selected\_codes]["종목코드"]

# # 모든 종목

# stock\_list.set\_index("종목명", inplace=True)

# selected\_codes = stock\_list.index.tolist()

# stock\_list = stock\_list.loc[selected\_codes]["종목코드"]

stock\_dic = dict.fromkeys(selected\_codes)

error\_list = []

corr\_list = []

anova\_weekday = 0

anova\_weeknum = 0

timeunit\_gap = 1

metric\_days = 14

# ==== selected feature =====

selected\_features = ["date", "close", "fast\_d", "obv", "fore\_mv20", "inst\_mv20", "kospi", "trading\_amount\_mv20"]

# selected\_features = []

for stock\_name, stock\_code in stock\_list.items():

try:

print("=====", stock\_name, "=====")

# 종목 주가 데이터 로드

stock\_dic[stock\_name] = dict.fromkeys(["df", "target\_list"])

stock\_df = stock.get\_market\_ohlcv\_by\_date(start\_date, end\_date, stock\_code).reset\_index()

sleep(1)

investor\_df = stock.get\_market\_trading\_volume\_by\_date(start\_date, end\_date, stock\_code)[["기관합계", "외국인합계"]].reset\_index()

sleep(1)

kospi\_df = stock.get\_index\_ohlcv\_by\_date(start\_date, end\_date, "1001")[["종가"]].reset\_index()

sleep(1)

stock\_df.columns = ["Date", "Open", "High", "Low", "Close", "Volume"]

investor\_df.columns = ["Date", "inst", "fore"]

kospi\_df.columns = ["Date", "kospi"]

# 영업일과 주가 정보를 outer 조인

train\_x = pd.merge(business\_days, stock\_df, how='left', on="Date")

train\_x = pd.merge(train\_x, investor\_df, how='left', on="Date")

train\_x = pd.merge(train\_x, kospi\_df, how='left', on="Date")

# 종가데이터에 생긴 na 값을 선형보간 및 정수로 반올림

train\_x.iloc[:,1:] = train\_x.iloc[:,1:].ffill(axis=0).round(0)

# ===== feature engineering =====

# 요일 및 주차 파생변수 추가

train\_x['weekday'] = train\_x["Date"].apply(lambda x: x.weekday())

train\_x['weeknum'] = train\_x["Date"].apply(lambda x: week\_of\_month(x))

cat\_vars = ["weekday", "weeknum"]

# 거래대금 파생변수 추가

train\_x['trading\_amount'] = train\_x["Close"] \* train\_x["Volume"]

# 월별 주기성 특징을 잡기 위한 sin 및 cos 변환 파생변수 추가

day\_to\_sec = 24 \* 60 \* 60

month\_to\_sec = 20 \* day\_to\_sec

timestamp\_s = train\_x["Date"].apply(datetime.timestamp)

timestamp\_freq = round((timestamp\_s / month\_to\_sec).diff(20)[20], 1)

train\_x['dayofmonth\_freq\_sin'] = np.sin((timestamp\_s / month\_to\_sec) \* ((2 \* np.pi) / timestamp\_freq))

train\_x['dayofmonth\_freq\_cos'] = np.cos((timestamp\_s / month\_to\_sec) \* ((2 \* np.pi) / timestamp\_freq))

# OBV 파생변수 추가

# 매수 신호: obv > obv\_ema

# 매도 신호: obv < obv\_ema

obv = [0]

for i in range(1, len(train\_x.Close)):

if train\_x.Close[i] > train\_x.Close[i - 1]:

obv.append(obv[-1] + train\_x.Volume[i])

elif train\_x.Close[i] < train\_x.Close[i - 1]:

obv.append(obv[-1] - train\_x.Volume[i])

else:

obv.append(obv[-1])

train\_x['obv'] = obv

train\_x['obv'][0] = nan

train\_x['obv\_ema'] = train\_x['obv'].ewm(com=metric\_days, min\_periods=metric\_days).mean()

# Stochastic 파생변수 추가

# fast\_d = moving average on fast\_k

train\_x[["fast\_k", "fast\_d"]] = stochastic(train\_x, n=metric\_days)[["fast\_k", "fast\_d"]]

# MFI 파생변수 추가

# MFI = 100 - (100 / 1 + MFR)

# MFR = 14일간의 양의 MF / 14일간의 음의 MF

# MF = 거래량 \* (당일고가 + 당일저가 + 당일종가) / 3

# MF 컬럼 만들기

train\_x["mf"] = train\_x["Volume"] \* ((train\_x["High"]+train\_x["Low"]+train\_x["Close"]) / 3)

# 양의 MF와 음의 MF 표기 컬럼 만들기

p\_n = []

for i in range(len(train\_x['mf'])):

if i == 0 :

p\_n.append(nan)

else:

if train\_x['mf'][i] >= train\_x['mf'][i-1]:

p\_n.append('p')

else:

p\_n.append('n')

train\_x['p\_n'] = p\_n

# 14일간 양의 MF/ 14일간 음의 MF 계산하여 컬럼 만들기

mfr = []

for i in range(len(train\_x['mf'])):

if i < metric\_days-1:

mfr.append(nan)

else:

train\_x\_=train\_x.iloc[(i-metric\_days+1):i]

a = sum(train\_x\_['mf'][train\_x['p\_n']=='p']) / sum(train\_x\_['mf'][train\_x['p\_n'] == 'n'])

mfr.append(a)

train\_x['mfr'] = mfr

# 최종 MFI 컬럼 만들기

train\_x['mfi'] = 100 - (100/(1+train\_x['mfr']))

train\_x["mfi\_signal"] = train\_x['mfi'].apply(lambda x: "buy" if x > 50 else "sell")

# 이동평균 추가

train\_x["close\_mv5"] = train\_x["Close"].rolling(5, min\_periods=5).mean()

train\_x["close\_mv10"] = train\_x["Close"].rolling(10, min\_periods=10).mean()

train\_x["close\_mv20"] = train\_x["Close"].rolling(20, min\_periods=20).mean()

train\_x["volume\_mv5"] = train\_x["Volume"].rolling(5, min\_periods=5).mean()

train\_x["volume\_mv10"] = train\_x["Volume"].rolling(10, min\_periods=10).mean()

train\_x["volume\_mv20"] = train\_x["Volume"].rolling(20, min\_periods=20).mean()

train\_x["trading\_amount\_mv5"] = train\_x["trading\_amount"].rolling(5, min\_periods=5).mean()

train\_x["trading\_amount\_mv10"] = train\_x["trading\_amount"].rolling(10, min\_periods=10).mean()

train\_x["trading\_amount\_mv20"] = train\_x["trading\_amount"].rolling(20, min\_periods=20).mean()

train\_x["inst\_mv5"] = train\_x["inst"].rolling(5, min\_periods=5).mean()

train\_x["inst\_mv10"] = train\_x["inst"].rolling(10, min\_periods=10).mean()

train\_x["inst\_mv20"] = train\_x["inst"].rolling(20, min\_periods=20).mean()

train\_x["fore\_mv5"] = train\_x["fore"].rolling(5, min\_periods=5).mean()

train\_x["fore\_mv10"] = train\_x["fore"].rolling(10, min\_periods=10).mean()

train\_x["fore\_mv20"] = train\_x["fore"].rolling(20, min\_periods=20).mean()

train\_x["kospi\_mv5"] = train\_x["kospi"].rolling(5, min\_periods=5).mean()

train\_x["kospi\_mv10"] = train\_x["kospi"].rolling(10, min\_periods=10).mean()

train\_x["kospi\_mv20"] = train\_x["kospi"].rolling(20, min\_periods=20).mean()

# 지표계산을 위해 쓰인 컬럼 drop

train\_x.drop(["mf", "p\_n", "mfr", "Open", "High", "Low"], inplace=True, axis=1)

# 2021/1/4 이후 일자만 선택

train\_x = train\_x[train\_x["Date"] >= datetime(2021, 1, 4)]

train\_x = train\_x.dropna()

train\_x.reset\_index(drop=True, inplace=True)

# create target list

target\_list = []

target\_list.append(train\_x["Close"])

target\_list.append(train\_x["Close"].shift(-1))

target\_list.append(train\_x["Close"].shift(-2))

target\_list.append(train\_x["Close"].shift(-3))

target\_list.append(train\_x["Close"].shift(-4))

target\_list.append(train\_x["Close"].shift(-5))

for idx, value in enumerate(target\_list):

value.name = "target\_shift" + str(idx)

# 컬럼이름 소문자 변환 및 정렬

train\_x.columns = train\_x.columns.str.lower()

train\_x = pd.concat([train\_x[["date"]], train\_x.iloc[:,1:].sort\_index(axis=1)], axis=1)

# <visualization>

# 시각화용 데이터프레임 생성

train\_bi = pd.concat([target\_list[timeunit\_gap], train\_x], axis=1)[:-timeunit\_gap]

# 평균 상관관계를 측정하기 위해 연산

corr\_obj = train\_bi.corr().round(3)

corr\_rows = corr\_obj.index.tolist()

corr\_cols = corr\_obj.columns.tolist()

corr\_list.append(corr\_obj.to\_numpy().round(3)[...,np.newaxis])

# # 상관관계 시각화

# fig, ax = plt.subplots(figsize=(12, 6))

# sns.heatmap(corr\_obj, cmap="YlGnBu", linewidths=0.2, annot=True)

# plt.xticks(rotation=45)

# fig.subplots\_adjust(left=0.15, bottom=0.2)

# plt.title('Correlation Visualization on ' + stock\_name, fontsize=15, fontweight="bold", pad=15)

# plt.savefig("projects/dacon\_stockprediction/graphs/" + stock\_name + ".png", dpi=300)

# plt.close()

# feature 와 target 간 시각화

# # ===== scatter plot on numerical feature =====

# for i in train\_x.columns:

# if i == "date" or i in cat\_vars:

# pass

# else:

# fig, ax = plt.subplots(figsize=(12, 6))

# graph = sns.regplot(x=train\_bi[i], y=train\_bi["target\_shift" + str(timeunit\_gap)], color="green",

# scatter\_kws={'s': 15}, line\_kws={"color": "orange"})

# graph.set\_title(i + " on " + stock\_name, fontsize=15, fontweight="bold", pad=15)

# plt.show()

# createFolder('projects/dacon\_stockprediction/graphs/' + stock\_name)

# plt.savefig('projects/dacon\_stockprediction/graphs/' + stock\_name + "/" + i +".png", dpi=300)

# plt.close()

# # feature 분포 시각화

# # ===== hist plot on numerical feature =====

# for i in train\_x.columns:

# if i == "date" or i in cat\_vars:

# pass

# else:

# plt.figure(figsize=(12, 6))

# graph = sns.histplot(x=train\_bi[i], bins=50, color="orange")

# graph.set\_title("Distribution on " + stock\_name + " (skewness : " + str(train\_bi[i].skew().round(3)) + ")", fontsize=15, fontweight="bold", pad=15)

# graph.set\_xlabel(graph.get\_xlabel(), fontsize=12, fontweight="bold", labelpad=15)

# graph.set\_ylabel(graph.get\_ylabel(), fontsize=12, fontweight="bold", labelpad=15)

# plt.show()

# createFolder('projects/dacon\_stockprediction/graphs/' + stock\_name)

# plt.savefig('projects/dacon\_stockprediction/graphs/' + stock\_name + "/dist\_" + i +".png", dpi=300)

# plt.close()

# <feature scaling>

# close, fast\_d, kospi, trading\_amount\_mv20 -> 로그 변환

train\_x[["close", "fast\_d", "kospi", "trading\_amount\_mv20"]] = train\_x[["close", "fast\_d", "kospi", "trading\_amount\_mv20"]].apply(np.log1p)

# scaling 후 재 시각화

# # ===== hist plot on numerical feature =====

# for i in ["close", "fast\_d", "kospi", "trading\_amount\_mv20"]:

# plt.figure(figsize=(12, 6))

# graph = sns.histplot(x=train\_x[i], bins=50, color="orange")

# graph.set\_title("After log scaling distribution on " + stock\_name + " (skewness : " + str(train\_x[i].skew().round(3)) + ")", fontsize=15,

# fontweight="bold", pad=15)

# graph.set\_xlabel(graph.get\_xlabel(), fontsize=12, fontweight="bold", labelpad=15)

# graph.set\_ylabel(graph.get\_ylabel(), fontsize=12, fontweight="bold", labelpad=15)

# plt.show()

# createFolder('projects/dacon\_stockprediction/graphs/' + stock\_name)

# plt.savefig('projects/dacon\_stockprediction/graphs/' + stock\_name + "/dist\_logTrans\_" + i + ".png", dpi=300)

# plt.close()

# # categorical 변수에 대한 분산분석 (target 과의 상관관계 파악)

# # 귀무가설(H0) : 두 변수는 상관관계가 없다

# # 대립가설(H1) : 두 변수는 상관관계가 있다

# cat\_list = train\_bi.groupby("weekday")["target\_shift" + str(timeunit\_gap)].apply(list)

# anova\_weekday += 1 / len(stock\_list) if f\_oneway(\*cat\_list)[1] <= 0.05 else 0

# cat\_list = train\_bi.groupby("weeknum")["target\_shift" + str(timeunit\_gap)].apply(list)

# anova\_weeknum += 1 / len(stock\_list) if f\_oneway(\*cat\_list)[1] <= 0.05 else 0

# # export csv for BI tool

# corr\_obj.to\_csv("projects/dacon\_stockprediction/bi\_dataset/bi\_corr\_" + stock\_name + ".csv", encoding="euc-kr", index\_label=True, header=False)

# train\_bi.to\_csv("projects/dacon\_stockprediction/bi\_dataset/bi\_data\_" + stock\_name + ".csv", encoding="euc-kr", index=False)

# <feature selection>

if len(selected\_features) != 0:

train\_x = train\_x[selected\_features]

b

stock\_dic[stock\_name]["df"] = train\_x.copy()

stock\_dic[stock\_name]["target\_list"] = target\_list.copy()

except:

print("ERROR :", stock\_name)

error\_list.append((stock\_name, stock\_code))

del train\_x