# Get Stock List

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

list\_name = 'Stock\_List.csv'

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

# 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))

stock\_list

# Get Data & Modeling

# 분석할 date 변수 지정

start\_date = '20210531'

end\_date = '20210910'

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'])

print(f'WEEKDAY of "start\_date" : {start\_weekday}')

print(f'NUM of WEEKS to "end\_date" : {max\_weeknum}')

print(f'HOW MANY "Business\_days" : {business\_days.shape}', )

print(business\_days.head(20))

# 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)

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

# 한 종목코드에 대한 주가 정보를 로드

stock\_code = stock\_list.loc[0, '종목코드']

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

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

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

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

# 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")

# 종가데이터에 생긴 na 값을 선형보간 및 정수로 반올림

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

print(train\_x.isna().sum())

# ===== 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))

# 주기성 신호로 변환한 파생변수 추가

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

weekday\_to\_sec = day\_to\_sec \* 5

year\_to\_sec = (365.2425) \* day\_to\_sec

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

timestamp\_freq = round((timestamp\_s / weekday\_to\_sec).diff(5)[5],1)

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

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

# sin transformation visualization

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

sns.set\_palette([sns.color\_palette("hls",10)[6]])

graph = sns.lineplot(x=range(len(train\_x["weekday"])), y=train\_x['weekday\_freq\_sin'])

graph.set\_title("Sine transformation on weekday", fontsize=15, fontweight="bold", pad=15)

plt.xticks(range(2, len(train\_x["weekday"]), 5), ["수요일"] \* len(train\_x["weekday"][2::5]))

for i in range(2, len(train\_x["weekday"]), 5):

plt.axvline(x=i, color="orange", linestyle="--")

# cos transformation visualization

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

sns.set\_palette([sns.color\_palette("hls",10)[6]])

graph = sns.lineplot(x=range(len(train\_x["weekday"])), y=train\_x['weekday\_freq\_cos'])

graph.set\_title("Cosine transformation on weekday", fontsize=15, fontweight="bold", pad=15)

plt.xticks(range(3, len(train\_x["weekday"]), 5), ["목요일"] \* len(train\_x["weekday"][3::5]))

for i in range(3, len(train\_x["weekday"]), 5):

plt.axvline(x=i, color="orange", linestyle="--")

# setting metrics days

metric\_days = 14

# obv

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()

# 매수/매도 타이밍 신호 찾는 함수

# 매수 신호: obv > obv\_ema

# 매도 신호: obv < obv\_ema

def getBreakthroughPoint(df, col1, col2, patient\_days):

'''

: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)

print(sigPrice)

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

sigPrice.ffill(inplace=True)

return sigPrice

train\_x['obv\_signal'] = getBreakthroughPoint(train\_x, 'obv', 'obv\_ema', 2)

train\_x

#OBV와 OBV\_EMA 시각화

plt.figure(figsize=(12,8))

plt.plot(train\_x['obv'], label='obv', color='orange')

plt.plot(train\_x['obv\_ema'], label='obv\_ema', color='purple')

plt.xticks(rotation=45)

#매수/매도 신호 시각화

plt.figure(figsize=(12,8))

plt.scatter(train\_x.index[train\_x['obv\_signal']=="buy"], train\_x["obv"][train\_x['obv\_signal']=="buy"], color = 'green',

label = 'Buy Signal', marker = '^', alpha = 1)

plt.scatter(train\_x.index[train\_x['obv\_signal']=="sell"], train\_x["obv"][train\_x['obv\_signal']=="sell"], color = 'red',

label = 'Sell Signal', marker = 'v', alpha = 1)

plt.plot(train\_x['obv'], label = 'OBV', alpha = 0.35)

plt.plot(train\_x['obv\_ema'], label = 'OBV moving average', alpha = 0.35)

plt.xticks(rotation=45)

plt.title('The Stock Buy / Sell Signals')

plt.xlabel('Date', fontsize = 18)

plt.ylabel('Close Price USD ($)', fontsize=18)

plt.legend(loc='upper right')

plt.show()

### stochastic 계산식

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

#데이터 프레임으로 받아오기 때문에 불필요

#df = pd.DataFrame(df)

#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

# 호출 방법

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

train\_x['stochastic\_signal'] = getBreakthroughPoint(train\_x, 'fast\_k', 'fast\_d', 2)

train\_x

#MFI 지표 구하기

#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")

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

train\_x.head(20)

train\_x["mv5"] = train\_x["Close"].rolling(5, min\_periods=5).mean()

train\_x["mv20"] = train\_x["Close"].rolling(20, min\_periods=20).mean()