# 투자지표 계산 및 Feature 생성 (“label\_ko” 는 띄어쓰기가 없는 한글만 존재)

# 투자지표를 계산하기 위한 서브지표 계산 함수들

# 섹터 dictionary에 대해 최근 n년간의 지표를 구함

# 10/06 수정

# 10/07 전체 코드 수정 (18:47)

# 10/11 전체 코드 수정 (14:22)

# ===== financial metric extraction =====

# 연도별 매출액

def getRevenue(fs\_meta, years=3, multidriverException="sum"):

# 서비스업은 주로 매출액이 아닌 영업수익 계정과목으로 매출실적을 나타냄

condition\_list = ["영업수익", "영업수익매출액"]

primary\_check\_list = fs\_meta["cis"]["label\_ko"].apply(lambda x: (x in condition\_list) and ("매출원가" not in x))

# 영업수익 또는 영업수익매출액 키워드를 찾고, 없으면 제조업과 관련된 계정과목으로 매출실적을 찾음

if primary\_check\_list.sum() == 0:

condition\_list = ["매출", "매출액", "수익매출액", '수익']

primary\_check\_list = fs\_meta["cis"]["label\_ko"].apply(lambda x: (x in condition\_list or "매출액" in x or "수익매출액" in x) and ("매출원가" not in x))

# 계정과목을 탐색하여 vector를 생성하고, 존재하지 않을 경우 0인 벡터를 생성합니다

if primary\_check\_list.sum() > 0:

if primary\_check\_list.sum() > 1: print("WARNING : Multi drivers are detected")

# 계정과목이 하나 이상 탐지된 경우 제일 첫 번째 열을 사용합니다

if multidriverException == "first":

output\_vector = fs\_meta["cis"][primary\_check\_list].iloc[0, 1:(1 + years)]

# 계정과목이 여러 개 탐지된 경우 모두 합산한 값을 사용합니다

else:

output\_vector = fs\_meta["cis"][primary\_check\_list].iloc[:, 1:(1 + years)].sum(axis=0)

# eps vector 값이 모두 0일 경우 리턴값을 설정합니다

if (output\_vector == 0).all():

print("INFO : All values are zero ---> return nan")

return series([nan] \* len(fs\_meta["cis"].iloc[0, 1:(1 + years)].index), fs\_meta["cis"].iloc[0, 1:(1 + years)].index)

# 아니면 eps vector 를 리턴합니다

else:

return output\_vector

# driver가 찾아지지 않은 경우, 리턴값을 설정합니다

else:

print("ERROR : Any driver is not found ---> return nan")

return series([nan] \* len(fs\_meta["cis"].iloc[0, 1:(1 + years)].index), fs\_meta["cis"].iloc[0, 1:(1 + years)].index)

# 연도별 영업이익

def getOperatingIncome(fs\_meta, years=3, multidriverException="first"):

condition\_list = ["영업이익", "영업이익손실", "영업손익", "영업손실"]

primary\_check\_list = fs\_meta["cis"]["label\_ko"].apply(lambda x: x in condition\_list)

# 계정과목을 탐색하여 vector를 생성하고, 존재하지 않을 경우 0인 벡터를 생성합니다

if primary\_check\_list.sum() > 0:

if primary\_check\_list.sum() > 1: print("WARNING : Multi drivers are detected")

# 계정과목이 하나 이상 탐지된 경우 제일 첫 번째 열을 사용합니다

if multidriverException == "first":

output\_vector = fs\_meta["cis"][primary\_check\_list].iloc[0, 1:(1 + years)]

# 계정과목이 여러 개 탐지된 경우 모두 합산한 값을 사용합니다

else:

output\_vector = fs\_meta["cis"][primary\_check\_list].iloc[:, 1:(1 + years)].sum(axis=0)

# eps vector 값이 모두 0일 경우 리턴값을 설정합니다

if (output\_vector == 0).all():

print("INFO : All values are zero ---> return nan")

return series([nan] \* len(fs\_meta["cis"].iloc[0, 1:(1 + years)].index),

fs\_meta["cis"].iloc[0, 1:(1 + years)].index)

# 아니면 eps vector 를 리턴합니다

else:

return output\_vector

# driver가 찾아지지 않은 경우, 리턴값을 설정합니다

else:

print("ERROR : Any driver is not found ---> return nan")

return series([nan] \* len(fs\_meta["cis"].iloc[0, 1:(1 + years)].index),

fs\_meta["cis"].iloc[0, 1:(1 + years)].index)

# 연도별 당기순이익

def getEarnings(fs\_meta, years=3, multidriverException="first"):

condition\_list = ["당기순이익", "당기순이익손실", "당기순손실", "당기순손익", "연결당기순손익", "연결당기순이익손실"]

primary\_check\_list = fs\_meta["cis"]["label\_ko"].apply(lambda x: x in condition\_list)

# 계정과목을 탐색하여 vector를 생성하고, 존재하지 않을 경우 0인 벡터를 생성합니다

if primary\_check\_list.sum() > 0:

if primary\_check\_list.sum() > 1: print("WARNING : Multi drivers are detected")

# 계정과목이 하나 이상 탐지된 경우 제일 첫 번째 열을 사용합니다

if multidriverException == "first":

output\_vector = fs\_meta["cis"][primary\_check\_list].iloc[0, 1:(1 + years)]

# 계정과목이 여러 개 탐지된 경우 모두 합산한 값을 사용합니다

else:

output\_vector = fs\_meta["cis"][primary\_check\_list].iloc[:, 1:(1 + years)].sum(axis=0)

# eps vector 값이 모두 0일 경우 리턴값을 설정합니다

if (output\_vector == 0).all():

print("INFO : All values are zero ---> return nan")

return series([nan] \* len(fs\_meta["cis"].iloc[0, 1:(1 + years)].index),

fs\_meta["cis"].iloc[0, 1:(1 + years)].index)

# 아니면 eps vector 를 리턴합니다

else:

return output\_vector

# driver가 찾아지지 않은 경우, 리턴값을 설정합니다

else:

print("ERROR : Any driver is not found ---> return nan")

return series([nan] \* len(fs\_meta["cis"].iloc[0, 1:(1 + years)].index),

fs\_meta["cis"].iloc[0, 1:(1 + years)].index)

# 연도별 EPS

def getEPS(fs\_meta, years=3, multidriverException="first", epsType="either", mixed\_eps\_fisrt=True):

'''

:param fs\_meta:

:param years:

:param multidriverException:

:param epsType:

:param mixed\_eps\_fisrt:

:return:

additional : eps 벡터 중 0 값이 있을경우, 자동적으로 계속 및 중단 eps 의 합으로 대체하는 연산을 수행합니다

'''

mixed\_eps = ["기본및희석주당보통주순이익손실", "기본및희석주당이익", "기본및희석주당순이익",

"기본및희석주당이익손실", "기본및희석주당순손실", "기본및희석주당순이익손실"]

mixed\_continuing\_eps = ["계속영업기본및희석주당이익손실", "계속영업기본및희석주당순이익"]

mixed\_discontinued\_eps = ["중단영업기본및희석주당이익손실", "중단영업기본및희석주당순이익"]

basic\_eps = ["기본주당이익", "기본주당순이익", "기본주당이익손실", "기본주당순이익손실", "기본주당손익",

"기본보통주당순이익", "기본보통주당순이익", "보통주기본주당이익", "기본주당손실", "보통주기본주당순이익"]

basic\_continuing\_eps = ["계속영업기본주당이익손실", "계속영업기본주당순손익", "계속영업기본주당손실"]

basic\_discontinued\_eps = ["중단영업기본주당이익손실", "중단영업기본주당순손익", "중단영업기본주당손실"]

diluted\_eps = ["희석주당이익", "희석주당순이익", "희석주당이익손실", "희석주당순이익손실", "희석주당손익",

"희석보통주당순이익", "희석보통주당순이익", "보통주희석주당이익", "희석주당손실", "보통주희석주당순이익"]

diluted\_continuing\_eps = ["계속영업희석주당이익손실", "계속영업희석주당순손익", "계속영업희석주당이익"]

diluted\_discontinued\_eps = ["중단영업희석주당이익손실", "중단영업희석주당순손익", "중단영업희석주당손실"]

# mixed\_eps\_first == True 이면, 기본 및 희석 eps 를 탐색합니다

if mixed\_eps\_fisrt:

mixed\_label = fs\_meta["cis"]["label\_ko"].apply(lambda x: x in mixed\_eps)

# 기본 및 희석 eps 를 탐색하여 vector를 생성하고, 존재하지 않을 경우 0인 벡터를 생성합니다

if mixed\_label.sum() > 0:

# 계정과목이 하나 이상 탐지된 경우 제일 첫 번째 열을 사용합니다

if multidriverException == "first":

mixed\_eps\_vector = fs\_meta["cis"][mixed\_label].iloc[0, 1:(1 + years)]

# 계정과목이 여러 개 탐지된 경우 모두 합산한 값을 사용합니다

else:

mixed\_eps\_vector = fs\_meta["cis"][mixed\_label].iloc[:, 1:(1 + years)].sum(axis=0)

else:

mixed\_eps\_vector = series([0] \* len(fs\_meta["cis"].iloc[0, 1:(1 + years)].index),

fs\_meta["cis"].iloc[0, 1:(1 + years)].index)

# 기본 eps vector 에 0인 값이 존재하면,

# 계속영업 및 중단영업 합으로 대체하는 process를 수행합니다

if (mixed\_eps\_vector == 0).sum() > 0:

continuing\_label = fs\_meta["cis"]["label\_ko"].apply(lambda x: x in mixed\_continuing\_eps)

discontinued\_label = fs\_meta["cis"]["label\_ko"].apply(lambda x: x in mixed\_discontinued\_eps)

# 계속영업 기본 eps 를 탐색하여 vector를 생성하고, 존재하지 않을 경우 0인 벡터를 생성합니다

if continuing\_label.sum() > 0:

continuing\_vector = fs\_meta["cis"][continuing\_label].iloc[0, 1:(1 + years)]

else:

continuing\_vector = series([0] \* len(fs\_meta["cis"].iloc[0, 1:(1 + years)].index),

fs\_meta["cis"].iloc[0, 1:(1 + years)].index)

# 중단영업 기본 eps 를 탐색하여 vector를 생성하고, 존재하지 않을 경우 0인 벡터를 생성합니다

if discontinued\_label.sum() > 0:

discontinued\_vector = fs\_meta["cis"][discontinued\_label].iloc[0, 1:(1 + years)]

else:

discontinued\_vector = series([0] \* len(fs\_meta["cis"].iloc[0, 1:(1 + years)].index),

fs\_meta["cis"].iloc[0, 1:(1 + years)].index)

# 기본 eps vector에 0인 값을 계속영업 eps + 중단영업 eps 값으로 대체합니다

con\_discon\_sum = continuing\_vector + discontinued\_vector

for idx, value in enumerate(mixed\_eps\_vector):

if value == 0:

mixed\_eps\_vector[idx] = con\_discon\_sum[idx]

# eps vector 값이 모두 0이면 다음 process를 진행합니다

if (mixed\_eps\_vector == 0).all():

print("ERROR : Any driver is not found ---> return nan")

pass

# 아니면 eps vector 를 리턴합니다

else:

return mixed\_eps\_vector

else:

pass

# 기본 eps 탐색

if epsType == "basic":

basic\_label = fs\_meta["cis"]["label\_ko"].apply(lambda x: x in basic\_eps)

# 기본 eps 를 탐색하여 vector를 생성하고, 존재하지 않을 경우 0인 벡터를 생성합니다

if basic\_label.sum() > 0:

# 계정과목이 하나 이상 탐지된 경우 제일 첫 번째 열을 사용합니다

if multidriverException == "first":

basic\_eps\_vector = fs\_meta["cis"][basic\_label].iloc[0, 1:(1 + years)]

# 계정과목이 여러 개 탐지된 경우 모두 합산한 값을 사용합니다

else:

basic\_eps\_vector = fs\_meta["cis"][basic\_label].iloc[:, 1:(1 + years)].sum(axis=0)

else:

basic\_eps\_vector = series([0] \* len(fs\_meta["cis"].iloc[0, 1:(1 + years)].index),

fs\_meta["cis"].iloc[0, 1:(1 + years)].index)

# 기본 eps vector 에 0인 값이 존재하면,

# 계속영업 및 중단영업 합으로 대체하는 process를 수행합니다

if (basic\_eps\_vector == 0).sum() > 0:

continuing\_label = fs\_meta["cis"]["label\_ko"].apply(lambda x: x in basic\_continuing\_eps)

discontinued\_label = fs\_meta["cis"]["label\_ko"].apply(lambda x: x in basic\_discontinued\_eps)

# 계속영업 기본 eps 를 탐색하여 vector를 생성하고, 존재하지 않을 경우 0인 벡터를 생성합니다

if continuing\_label.sum() > 0:

continuing\_vector = fs\_meta["cis"][continuing\_label].iloc[0, 1:(1 + years)]

else:

continuing\_vector = series([0] \* len(fs\_meta["cis"].iloc[0, 1:(1 + years)].index),

fs\_meta["cis"].iloc[0, 1:(1 + years)].index)

# 중단영업 기본 eps 를 탐색하여 vector를 생성하고, 존재하지 않을 경우 0인 벡터를 생성합니다

if discontinued\_label.sum() > 0:

discontinued\_vector = fs\_meta["cis"][discontinued\_label].iloc[0, 1:(1 + years)]

else:

discontinued\_vector = series([0] \* len(fs\_meta["cis"].iloc[0, 1:(1 + years)].index),

fs\_meta["cis"].iloc[0, 1:(1 + years)].index)

# 기본 eps vector에 0인 값을 계속영업 eps + 중단영업 eps 값으로 대체합니다

con\_discon\_sum = continuing\_vector + discontinued\_vector

for idx, value in enumerate(basic\_eps\_vector):

if value == 0:

basic\_eps\_vector[idx] = con\_discon\_sum[idx]

# eps vector 값이 모두 0이면 다음 nan을 리턴합니다

if (basic\_eps\_vector == 0).all():

print("ERROR : Any driver is not found ---> return nan")

return series([nan] \* len(fs\_meta["cis"].iloc[0, 1:(1 + years)].index),

fs\_meta["cis"].iloc[0, 1:(1 + years)].index)

# 아니면 eps vector 를 리턴합니다

else:

return basic\_eps\_vector

# 희석 eps 탐색

elif epsType == "diluted":

diluted\_label = fs\_meta["cis"]["label\_ko"].apply(lambda x: x in diluted\_eps)

# 희석 eps 를 탐색하여 vector를 생성하고, 존재하지 않을 경우 0인 벡터를 생성합니다

if diluted\_label.sum() > 0:

# 계정과목이 하나 이상 탐지된 경우 제일 첫번째 열만 사용합니다

if multidriverException == "first":

diluted\_eps\_vector = fs\_meta["cis"][diluted\_label].iloc[0, 1:(1 + years)]

# 계정과목이 여러 개 탐지된 경우 모두 합산한 값을 사용합니다

else:

diluted\_eps\_vector = fs\_meta["cis"][diluted\_label].iloc[:, 1:(1 + years)].sum(axis=0)

else:

diluted\_eps\_vector = series([0] \* len(fs\_meta["cis"].iloc[0, 1:(1 + years)].index),

fs\_meta["cis"].iloc[0, 1:(1 + years)].index)

# 희석 eps vector 에 0인 값이 존재하면,

# 계속영업 및 중단영업 합으로 대체하는 process를 수행합니다

if (diluted\_eps\_vector == 0).sum() > 0:

continuing\_label = fs\_meta["cis"]["label\_ko"].apply(lambda x: x in diluted\_continuing\_eps)

discontinued\_label = fs\_meta["cis"]["label\_ko"].apply(lambda x: x in diluted\_discontinued\_eps)

# 계속영업 희석 eps 를 탐색하여 vector를 생성하고, 존재하지 않을 경우 0인 벡터를 생성합니다

if continuing\_label.sum() > 0:

continuing\_vector = fs\_meta["cis"][continuing\_label].iloc[0, 1:(1 + years)]

else:

continuing\_vector = series([0] \* len(fs\_meta["cis"].iloc[0, 1:(1 + years)].index),

fs\_meta["cis"].iloc[0, 1:(1 + years)].index)

# 중단영업 희석 eps 를 탐색하여 vector를 생성하고, 존재하지 않을 경우 0인 벡터를 생성합니다

if discontinued\_label.sum() > 0:

discontinued\_vector = fs\_meta["cis"][discontinued\_label].iloc[0, 1:(1 + years)]

else:

discontinued\_vector = series([0] \* len(fs\_meta["cis"].iloc[0, 1:(1 + years)].index),

fs\_meta["cis"].iloc[0, 1:(1 + years)].index)

# 희석 eps vector에 0인 값을 계속영업 eps + 중단영업 eps 값으로 대체합니다

con\_discon\_sum = continuing\_vector + discontinued\_vector

for idx, value in enumerate(diluted\_eps\_vector):

if value == 0:

diluted\_eps\_vector[idx] = con\_discon\_sum[idx]

# eps vector 값이 모두 0이면 다음 nan을 리턴합니다

if (diluted\_eps\_vector == 0).all():

print("ERROR : Any driver is not found ---> return nan")

return series([nan] \* len(fs\_meta["cis"].iloc[0, 1:(1 + years)].index),

fs\_meta["cis"].iloc[0, 1:(1 + years)].index)

# 아니면 eps vector 를 리턴합니다

else:

return diluted\_eps\_vector

# 기본 eps 및 희석 eps 를 둘 다 탐색하여, 기본 eps 가 0 이면 희석 eps 값으로 대체

else:

# ===== basic eps 계산 =====

basic\_label = fs\_meta["cis"]["label\_ko"].apply(lambda x: x in basic\_eps)

# 기본 eps 를 탐색하여 vector를 생성하고, 존재하지 않을 경우 0인 벡터를 생성합니다

if basic\_label.sum() > 0:

# 계정과목이 하나 이상 탐지된 경우 제일 첫 번째 열을 사용합니다

if multidriverException == "first":

basic\_eps\_vector = fs\_meta["cis"][basic\_label].iloc[0, 1:(1 + years)]

# 계정과목이 여러 개 탐지된 경우 모두 합산한 값을 사용합니다

else:

basic\_eps\_vector = fs\_meta["cis"][basic\_label].iloc[:, 1:(1 + years)].sum(axis=0)

else:

basic\_eps\_vector = series([0] \* years, fs\_meta["cis"].iloc[0, 1:(1 + years)].index)

# 기본 eps vector 에 0인 값이 존재하면,

# 계속영업 및 중단영업 합으로 대체하는 process를 수행합니다

if (basic\_eps\_vector == 0).sum() > 0:

continuing\_label = fs\_meta["cis"]["label\_ko"].apply(lambda x: x in basic\_continuing\_eps)

discontinued\_label = fs\_meta["cis"]["label\_ko"].apply(lambda x: x in basic\_discontinued\_eps)

# 계속영업 기본 eps 를 탐색하여 vector를 생성하고, 존재하지 않을 경우 0인 벡터를 생성합니다

if continuing\_label.sum() > 0:

continuing\_vector = fs\_meta["cis"][continuing\_label].iloc[0, 1:(1 + years)]

else:

continuing\_vector = series([0] \* len(fs\_meta["cis"].iloc[0, 1:(1 + years)].index),

fs\_meta["cis"].iloc[0, 1:(1 + years)].index)

# 중단영업 기본 eps 를 탐색하여 vector를 생성하고, 존재하지 않을 경우 0인 벡터를 생성합니다

if discontinued\_label.sum() > 0:

discontinued\_vector = fs\_meta["cis"][discontinued\_label].iloc[0, 1:(1 + years)]

else:

discontinued\_vector = series([0] \* len(fs\_meta["cis"].iloc[0, 1:(1 + years)].index),

fs\_meta["cis"].iloc[0, 1:(1 + years)].index)

# 기본 eps vector에 0인 값을 계속영업 eps + 중단영업 eps 값으로 대체합니다

con\_discon\_sum = continuing\_vector + discontinued\_vector

for idx, value in enumerate(basic\_eps\_vector):

if value == 0:

basic\_eps\_vector[idx] = con\_discon\_sum[idx]

# ===== diluted eps 대치 =====

# eps vector 에 0인 값이 하나라도 있으면

if (basic\_eps\_vector == 0).sum() > 0:

diluted\_label = fs\_meta["cis"]["label\_ko"].apply(lambda x: x in diluted\_eps)

# 희석 eps 를 탐색하여 vector를 생성하고, 존재하지 않을 경우 0인 벡터를 생성합니다

if diluted\_label.sum() > 0:

# 계정과목이 하나 이상 탐지된 경우 제일 첫번째 열만 사용합니다

if multidriverException == "first":

diluted\_eps\_vector = fs\_meta["cis"][diluted\_label].iloc[0, 1:(1 + years)]

# 계정과목이 여러 개 탐지된 경우 모두 합산한 값을 사용합니다

else:

diluted\_eps\_vector = fs\_meta["cis"][diluted\_label].iloc[:, 1:(1 + years)].sum(axis=0)

else:

diluted\_eps\_vector = series([0] \* len(fs\_meta["cis"].iloc[0, 1:(1 + years)].index),

fs\_meta["cis"].iloc[0, 1:(1 + years)].index)

# 희석 eps vector 에 0인 값이 존재하면,

# 계속영업 및 중단영업 합으로 대체하는 process를 수행합니다

if (diluted\_eps\_vector == 0).sum() > 0:

continuing\_label = fs\_meta["cis"]["label\_ko"].apply(lambda x: x in diluted\_continuing\_eps)

discontinued\_label = fs\_meta["cis"]["label\_ko"].apply(lambda x: x in diluted\_discontinued\_eps)

# 계속영업 희석 eps 를 탐색하여 vector를 생성하고, 존재하지 않을 경우 0인 벡터를 생성합니다

if continuing\_label.sum() > 0:

continuing\_vector = fs\_meta["cis"][continuing\_label].iloc[0, 1:(1 + years)]

else:

continuing\_vector = series([0] \* len(fs\_meta["cis"].iloc[0, 1:(1 + years)].index),

fs\_meta["cis"].iloc[0, 1:(1 + years)].index)

# 중단영업 희석 eps 를 탐색하여 vector를 생성하고, 존재하지 않을 경우 0인 벡터를 생성합니다

if discontinued\_label.sum() > 0:

discontinued\_vector = fs\_meta["cis"][discontinued\_label].iloc[0, 1:(1 + years)]

else:

discontinued\_vector = series([0] \* len(fs\_meta["cis"].iloc[0, 1:(1 + years)].index),

fs\_meta["cis"].iloc[0, 1:(1 + years)].index)

# 희석 eps vector에 0인 값을 계속영업 eps + 중단영업 eps 값으로 대체합니다

con\_discon\_sum = continuing\_vector + discontinued\_vector

for idx, value in enumerate(diluted\_eps\_vector):

if value == 0:

diluted\_eps\_vector[idx] = con\_discon\_sum[idx]

# 기본 eps vector에 0인 값을 희석 eps vector의 값으로 대체합니다

for idx, value in enumerate(basic\_eps\_vector):

if value == 0:

basic\_eps\_vector[idx] = diluted\_eps\_vector[idx]

# eps vector 값이 모두 0이면 다음 nan을 리턴합니다

if (basic\_eps\_vector == 0).all():

return series([nan] \* len(fs\_meta["cis"].iloc[0, 1:(1 + years)].index),

fs\_meta["cis"].iloc[0, 1:(1 + years)].index)

# 아니면 eps vector 를 리턴합니다

else:

return basic\_eps\_vector

# 연도별 법인세

def getTax(fs\_meta, years=3, multidriverException="sum"):

condition\_list = ["법인세의납부", "법인세비용수익", "법인세납부환급", "법인세납부", "법인세비용"]

primary\_check\_list = fs\_meta["cf"]["label\_ko"].apply(lambda x: x in condition\_list)

# 계정과목을 탐색하여 vector를 생성하고, 존재하지 않을 경우 0인 벡터를 생성합니다

if primary\_check\_list.sum() > 0:

if primary\_check\_list.sum() > 1: print("WARNING : Multi drivers are detected")

# 계정과목이 하나 이상 탐지된 경우 제일 첫 번째 열을 사용합니다

if multidriverException == "first":

output\_vector = fs\_meta["cf"][primary\_check\_list].iloc[0, 1:(1 + years)]

# 계정과목이 여러 개 탐지된 경우 모두 합산한 값을 사용합니다

else:

output\_vector = fs\_meta["cf"][primary\_check\_list].iloc[:, 1:(1 + years)].sum(axis=0)

# eps vector 값이 모두 0일 경우 리턴값을 설정합니다

if (output\_vector == 0).all():

print("INFO : All values are zero ---> return 0")

return series([0] \* len(fs\_meta["cis"].iloc[0, 1:(1 + years)].index), fs\_meta["cf"].iloc[0, 1:(1 + years)].index)

# 아니면 eps vector 를 리턴합니다

else:

return output\_vector

# driver가 찾아지지 않은 경우, 리턴값을 설정합니다

else:

print("ERROR : Any driver is not found ---> return 0")

return series([0] \* len(fs\_meta["cis"].iloc[0, 1:(1 + years)].index), fs\_meta["cf"].iloc[0, 1:(1 + years)].index)

# 연도별 이자비용

def getInterestExpense(fs\_meta, years=3, multidriverException="sum"):

condition\_list = ["이자비용", "이자의지급", "이자지급", "이자지급영업"]

primary\_check\_list = fs\_meta["cf"]["label\_ko"].apply(lambda x: x in condition\_list)

# 계정과목을 탐색하여 vector를 생성하고, 존재하지 않을 경우 0인 벡터를 생성합니다

if primary\_check\_list.sum() > 0:

if primary\_check\_list.sum() > 1: print("WARNING : Multi drivers are detected")

# 계정과목이 하나 이상 탐지된 경우 제일 첫 번째 열을 사용합니다

if multidriverException == "first":

output\_vector = fs\_meta["cf"][primary\_check\_list].iloc[0, 1:(1 + years)]

# 계정과목이 여러 개 탐지된 경우 모두 합산한 값을 사용합니다

else:

output\_vector = fs\_meta["cf"][primary\_check\_list].iloc[:, 1:(1 + years)].sum(axis=0)

# eps vector 값이 모두 0일 경우 리턴값을 설정합니다

if (output\_vector == 0).all():

print("INFO : All values are zero ---> return 0")

return series([0] \* len(fs\_meta["cis"].iloc[0, 1:(1 + years)].index), fs\_meta["cf"].iloc[0, 1:(1 + years)].index)

# 아니면 eps vector 를 리턴합니다

else:

return output\_vector

# driver가 찾아지지 않은 경우, 리턴값을 설정합니다

else:

print("ERROR : Any driver is not found ---> return 0")

return series([0] \* len(fs\_meta["cis"].iloc[0, 1:(1 + years)].index), fs\_meta["cf"].iloc[0, 1:(1 + years)].index)

# 연도별 감가상각비

def getDepreciationAndAmortization(fs\_meta, years=3, multidriverException="sum"):

condition\_list = ["감가상각비", "자산상각비", "대손상각비", "당기순이익조정을위한가감"]

primary\_check\_list = fs\_meta["cf"]["label\_ko"].apply(lambda x: x in condition\_list)

# 계정과목을 탐색하여 vector를 생성하고, 존재하지 않을 경우 0인 벡터를 생성합니다

if primary\_check\_list.sum() > 0:

if primary\_check\_list.sum() > 1: print("WARNING : Multi drivers are detected")

# 계정과목이 하나 이상 탐지된 경우 제일 첫 번째 열을 사용합니다

if multidriverException == "first":

output\_vector = fs\_meta["cf"][primary\_check\_list].iloc[0, 1:(1 + years)]

# 계정과목이 여러 개 탐지된 경우 모두 합산한 값을 사용합니다

else:

output\_vector = fs\_meta["cf"][primary\_check\_list].iloc[:, 1:(1 + years)].sum(axis=0)

# eps vector 값이 모두 0일 경우 리턴값을 설정합니다

if (output\_vector == 0).all():

print("INFO : All values are zero ---> return 0")

return series([0] \* len(fs\_meta["cis"].iloc[0, 1:(1 + years)].index), fs\_meta["cf"].iloc[0, 1:(1 + years)].index)

# 아니면 eps vector 를 리턴합니다

else:

return output\_vector

# driver가 찾아지지 않은 경우, 리턴값을 설정합니다

else:

print("ERROR : Any driver is not found ---> return 0")

return series([0] \* len(fs\_meta["cis"].iloc[0, 1:(1 + years)].index), fs\_meta["cf"].iloc[0, 1:(1 + years)].index)

# 연도별 자기자본

def getEquity(fs\_meta, years=3, multidriverException="nonzero"):

condition\_list = ["자본총계", "자본"]

primary\_check\_list = fs\_meta["bs"]["label\_ko"].apply(lambda x: x in condition\_list)

# 계정과목을 탐색하여 vector를 생성하고, 존재하지 않을 경우 0인 벡터를 생성합니다

if primary\_check\_list.sum() > 0:

if primary\_check\_list.sum() > 1: print("WARNING : Multi drivers are detected")

# 계정과목이 하나 이상 탐지된 경우 제일 첫 번째 열을 사용합니다

if multidriverException == "first":

output\_vector = fs\_meta["bs"][primary\_check\_list].iloc[0, 1:(1 + years)]

# 계정과목이 여러 개 탐지된 경우 모두 합산한 값을 사용합니다

else:

output\_vector = fs\_meta["bs"][primary\_check\_list].iloc[:, 1:(1 + years)].sum(axis=0)

# eps vector 값이 모두 0일 경우 리턴값을 설정합니다

if (output\_vector == 0).all():

print("INFO : All values are zero ---> return nan")

return series([nan] \* len(fs\_meta["cis"].iloc[0, 1:(1 + years)].index), fs\_meta["bs"].iloc[0, 1:(1 + years)].index)

# 아니면 eps vector 를 리턴합니다

else:

return output\_vector

# driver가 찾아지지 않은 경우, 리턴값을 설정합니다

else:

print("ERROR : Any driver is not found ---> return nan")

return series([nan] \* len(fs\_meta["cis"].iloc[0, 1:(1 + years)].index), fs\_meta["bs"].iloc[0, 1:(1 + years)].index)

# 연도별 부채

def getDebt(fs\_meta, years=3, multidriverException="first"):

condition\_list = ["부채총계", "부채"]

primary\_check\_list = fs\_meta["bs"]["label\_ko"].apply(lambda x: x in condition\_list)

# 계정과목을 탐색하여 vector를 생성하고, 존재하지 않을 경우 0인 벡터를 생성합니다

if primary\_check\_list.sum() > 0:

if primary\_check\_list.sum() > 1: print("WARNING : Multi drivers are detected")

# 계정과목이 하나 이상 탐지된 경우 제일 첫 번째 열을 사용합니다

if multidriverException == "first":

output\_vector = fs\_meta["bs"][primary\_check\_list].iloc[0, 1:(1 + years)]

# 계정과목이 여러 개 탐지된 경우 모두 합산한 값을 사용합니다

else:

output\_vector = fs\_meta["bs"][primary\_check\_list].iloc[:, 1:(1 + years)].sum(axis=0)

# eps vector 값이 모두 0일 경우 리턴값을 설정합니다

if (output\_vector == 0).all():

print("INFO : All values are zero ---> return nan")

return series([nan] \* len(fs\_meta["cis"].iloc[0, 1:(1 + years)].index), fs\_meta["bs"].iloc[0, 1:(1 + years)].index)

# 아니면 eps vector 를 리턴합니다

else:

return output\_vector

# driver가 찾아지지 않은 경우, 리턴값을 설정합니다

else:

print("ERROR : Any driver is not found ---> return nan")

return series([nan] \* len(fs\_meta["cis"].iloc[0, 1:(1 + years)].index), fs\_meta["bs"].iloc[0, 1:(1 + years)].index)

# 연도별 영업활동 현금흐름 (조정항목이 계산되지 않은 현금흐름)

def getCFO(fs\_meta, years=3, multidriverException="first"):

condition\_list = ["영업활동현금흐름", "영업활동으로인한현금흐름"]

primary\_check\_list = fs\_meta["cf"]["label\_ko"].apply(lambda x: x in condition\_list)

# 계정과목을 탐색하여 vector를 생성하고, 존재하지 않을 경우 0인 벡터를 생성합니다

if primary\_check\_list.sum() > 0:

if primary\_check\_list.sum() > 1: print("WARNING : Multi drivers are detected")

# 계정과목이 하나 이상 탐지된 경우 제일 첫 번째 열을 사용합니다

if multidriverException == "first":

output\_vector = fs\_meta["cf"][primary\_check\_list].iloc[0, 1:(1 + years)]

# 계정과목이 여러 개 탐지된 경우 모두 합산한 값을 사용합니다

else:

output\_vector = fs\_meta["cf"][primary\_check\_list].iloc[:, 1:(1 + years)].sum(axis=0)

# eps vector 값이 모두 0일 경우 리턴값을 설정합니다

if (output\_vector == 0).all():

print("INFO : All values are zero ---> return nan")

return series([nan] \* len(fs\_meta["cis"].iloc[0, 1:(1 + years)].index), fs\_meta["cf"].iloc[0, 1:(1 + years)].index)

# 아니면 eps vector 를 리턴합니다

else:

return output\_vector

# driver가 찾아지지 않은 경우, 리턴값을 설정합니다

else:

print("ERROR : Any driver is not found ---> return nan")

return series([nan] \* len(fs\_meta["cis"].iloc[0, 1:(1 + years)].index), fs\_meta["cf"].iloc[0, 1:(1 + years)].index)

# 연도별 영업에서 창출된 현금흐름 (조정항목이 미리 계산된 현금흐름)

def getAdjCFO(fs\_meta, years=3, multidriverException="first"):

condition\_list = ["영업에서창출된현금흐름", "영업으로부터창출된현금흐름", "영업에서창출된현금"]

primary\_check\_list = fs\_meta["cf"]["label\_ko"].apply(lambda x: x in condition\_list)

# 계정과목을 탐색하여 vector를 생성하고, 존재하지 않을 경우 0인 벡터를 생성합니다

if primary\_check\_list.sum() > 0:

if primary\_check\_list.sum() > 1: print("WARNING : Multi drivers are detected")

# 계정과목이 하나 이상 탐지된 경우 제일 첫 번째 열을 사용합니다

if multidriverException == "first":

output\_vector = fs\_meta["cf"][primary\_check\_list].iloc[0, 1:(1 + years)]

# 계정과목이 여러 개 탐지된 경우 모두 합산한 값을 사용합니다

else:

output\_vector = fs\_meta["cf"][primary\_check\_list].iloc[:, 1:(1 + years)].sum(axis=0)

# eps vector 값이 모두 0일 경우 리턴값을 설정합니다

if (output\_vector == 0).all():

print("INFO : All values are zero ---> return nan")

return series([nan] \* len(fs\_meta["cis"].iloc[0, 1:(1 + years)].index), fs\_meta["cf"].iloc[0, 1:(1 + years)].index)

# 아니면 eps vector 를 리턴합니다

else:

return output\_vector

# driver가 찾아지지 않은 경우, 리턴값을 설정합니다

else:

print("ERROR : Any driver is not found ---> return nan")

return series([nan] \* len(fs\_meta["cis"].iloc[0, 1:(1 + years)].index), fs\_meta["cf"].iloc[0, 1:(1 + years)].index)

# 연도별 재고자산

def getInventory(fs\_meta, years=3, multidriverException="sum"):

condition\_list = ['재고자산']

primary\_check\_list = fs\_meta["bs"]["label\_ko"].apply(lambda x: x in condition\_list)

# 계정과목을 탐색하여 vector를 생성하고, 존재하지 않을 경우 0인 벡터를 생성합니다

if primary\_check\_list.sum() > 0:

if primary\_check\_list.sum() > 1: print("WARNING : Multi drivers are detected")

# 계정과목이 하나 이상 탐지된 경우 제일 첫 번째 열을 사용합니다

if multidriverException == "first":

output\_vector = fs\_meta["bs"][primary\_check\_list].iloc[0, 1:(1 + years)]

# 계정과목이 여러 개 탐지된 경우 모두 합산한 값을 사용합니다

else:

output\_vector = fs\_meta["bs"][primary\_check\_list].iloc[:, 1:(1 + years)].sum(axis=0)

# eps vector 값이 모두 0일 경우 리턴값을 설정합니다

if (output\_vector == 0).all():

print("INFO : All values are zero ---> return 0")

return series([0] \* len(fs\_meta["cis"].iloc[0, 1:(1 + years)].index), fs\_meta["bs"].iloc[0, 1:(1 + years)].index)

# 아니면 eps vector 를 리턴합니다

else:

return output\_vector

# driver가 찾아지지 않은 경우, 리턴값을 설정합니다

else:

print("ERROR : Any driver is not found ---> return 0")

return series([0] \* len(fs\_meta["cis"].iloc[0, 1:(1 + years)].index), fs\_meta["bs"].iloc[0, 1:(1 + years)].index)

# 연도별 유동자산

def getCurrentAsset(fs\_meta, years=3, multidriverException="first"):

condition\_list = ["유동자산"]

primary\_check\_list = fs\_meta["bs"]["label\_ko"].apply(lambda x: x in condition\_list)

# 계정과목을 탐색하여 vector를 생성하고, 존재하지 않을 경우 0인 벡터를 생성합니다

if primary\_check\_list.sum() > 0:

if primary\_check\_list.sum() > 1: print("WARNING : Multi drivers are detected")

# 계정과목이 하나 이상 탐지된 경우 제일 첫 번째 열을 사용합니다

if multidriverException == "first":

output\_vector = fs\_meta["bs"][primary\_check\_list].iloc[0, 1:(1 + years)]

# 계정과목이 여러 개 탐지된 경우 모두 합산한 값을 사용합니다

else:

output\_vector = fs\_meta["bs"][primary\_check\_list].iloc[:, 1:(1 + years)].sum(axis=0)

# eps vector 값이 모두 0일 경우 리턴값을 설정합니다

if (output\_vector == 0).all():

print("INFO : All values are zero ---> return nan")

return series([nan] \* len(fs\_meta["cis"].iloc[0, 1:(1 + years)].index), fs\_meta["bs"].iloc[0, 1:(1 + years)].index)

# 아니면 eps vector 를 리턴합니다

else:

return output\_vector

# driver가 찾아지지 않은 경우, 리턴값을 설정합니다

else:

print("ERROR : Any driver is not found ---> return 0")

return series([nan] \* len(fs\_meta["cis"].iloc[0, 1:(1 + years)].index), fs\_meta["bs"].iloc[0, 1:(1 + years)].index)

# 연도별 유동부채

def getCurrentLiab(fs\_meta, years=3, multidriverException="first"):

condition\_list = ["유동부채"]

primary\_check\_list = fs\_meta["bs"]["label\_ko"].apply(lambda x: x in condition\_list)

# 계정과목을 탐색하여 vector를 생성하고, 존재하지 않을 경우 0인 벡터를 생성합니다

if primary\_check\_list.sum() > 0:

if primary\_check\_list.sum() > 1: print("WARNING : Multi drivers are detected")

# 계정과목이 하나 이상 탐지된 경우 제일 첫 번째 열을 사용합니다

if multidriverException == "first":

output\_vector = fs\_meta["bs"][primary\_check\_list].iloc[0, 1:(1 + years)]

# 계정과목이 여러 개 탐지된 경우 모두 합산한 값을 사용합니다

else:

output\_vector = fs\_meta["bs"][primary\_check\_list].iloc[:, 1:(1 + years)].sum(axis=0)

# eps vector 값이 모두 0일 경우 리턴값을 설정합니다

if (output\_vector == 0).all():

print("INFO : All values are zero ---> return nan")

return series([nan] \* len(fs\_meta["cis"].iloc[0, 1:(1 + years)].index), fs\_meta["bs"].iloc[0, 1:(1 + years)].index)

# 아니면 eps vector 를 리턴합니다

else:

return output\_vector

# driver가 찾아지지 않은 경우, 리턴값을 설정합니다

else:

print("ERROR : Any driver is not found ---> return nan")

return series([nan] \* len(fs\_meta["cis"].iloc[0, 1:(1 + years)].index), fs\_meta["bs"].iloc[0, 1:(1 + years)].index)

# 연도별 배당금

def getDividend(fs\_meta, years=3, multidriverException="sum"):

condition\_list = ["배당금지급", "배당금의지급"]

primary\_check\_list = fs\_meta["cf"]["label\_ko"].apply(lambda x: x in condition\_list)

# 계정과목을 탐색하여 vector를 생성하고, 존재하지 않을 경우 0인 벡터를 생성합니다

if primary\_check\_list.sum() > 0:

if primary\_check\_list.sum() > 1: print("WARNING : Multi drivers are detected")

# 계정과목이 하나 이상 탐지된 경우 제일 첫 번째 열을 사용합니다

if multidriverException == "first":

output\_vector = fs\_meta["cf"][primary\_check\_list].iloc[0, 1:(1 + years)]

# 계정과목이 여러 개 탐지된 경우 모두 합산한 값을 사용합니다

else:

output\_vector = fs\_meta["cf"][primary\_check\_list].iloc[:, 1:(1 + years)].sum(axis=0)

# eps vector 값이 모두 0일 경우 리턴값을 설정합니다

if (output\_vector == 0).all():

print("INFO : All values are zero ---> return 0")

return series([0] \* len(fs\_meta["cis"].iloc[0, 1:(1 + years)].index), fs\_meta["cf"].iloc[0, 1:(1 + years)].index)

# 아니면 eps vector 를 리턴합니다

else:

return output\_vector

# driver가 찾아지지 않은 경우, 리턴값을 설정합니다

else:

print("ERROR : Any driver is not found ---> return 0")

return series([0] \* len(fs\_meta["cis"].iloc[0, 1:(1 + years)].index), fs\_meta["cf"].iloc[0, 1:(1 + years)].index)

# 연도별 자사주매입금

def getBuyback(fs\_meta, years=3, multidriverException="sum"):

condition\_list = ["자기주식의처분"]

primary\_check\_list = fs\_meta["cf"]["label\_ko"].apply(lambda x: x in condition\_list)

# 계정과목을 탐색하여 vector를 생성하고, 존재하지 않을 경우 0인 벡터를 생성합니다

if primary\_check\_list.sum() > 0:

if primary\_check\_list.sum() > 1: print("WARNING : Multi drivers are detected")

# 계정과목이 하나 이상 탐지된 경우 제일 첫 번째 열을 사용합니다

if multidriverException == "first":

output\_vector = fs\_meta["cf"][primary\_check\_list].iloc[0, 1:(1 + years)]

# 계정과목이 여러 개 탐지된 경우 모두 합산한 값을 사용합니다

else:

output\_vector = fs\_meta["cf"][primary\_check\_list].iloc[:, 1:(1 + years)].sum(axis=0)

# eps vector 값이 모두 0일 경우 리턴값을 설정합니다

if (output\_vector == 0).all():

print("INFO : All values are zero ---> return 0")

return series([0] \* len(fs\_meta["cis"].iloc[0, 1:(1 + years)].index), fs\_meta["cf"].iloc[0, 1:(1 + years)].index)

# 아니면 eps vector 를 리턴합니다

else:

return output\_vector

# driver가 찾아지지 않은 경우, 리턴값을 설정합니다

else:

print("ERROR : Any driver is not found ---> return 0")

return series([0] \* len(fs\_meta["cis"].iloc[0, 1:(1 + years)].index), fs\_meta["cf"].iloc[0, 1:(1 + years)].index)

# 연도별 매출채권 (not used)

def getBond(fs\_meta, years=3, multidriverException="sum"):

condition\_list = ["매출채권및기타채권", "매출채권", '매출채권및기타유동채권', '매출채권및기타채권',

'매출채권및기타수취채권', '매출채권및수취채권', '매출채권및계약자산']

primary\_check\_list = fs\_meta["bs"]["label\_ko"].apply(lambda x: x in condition\_list)

# 계정과목을 탐색하여 vector를 생성하고, 존재하지 않을 경우 0인 벡터를 생성합니다

if primary\_check\_list.sum() > 0:

if primary\_check\_list.sum() > 1: print("WARNING : Multi drivers are detected")

# 계정과목이 하나 이상 탐지된 경우 제일 첫 번째 열을 사용합니다

if multidriverException == "first":

output\_vector = fs\_meta["bs"][primary\_check\_list].iloc[0, 1:(1 + years)]

# 계정과목이 여러 개 탐지된 경우 모두 합산한 값을 사용합니다

else:

output\_vector = fs\_meta["bs"][primary\_check\_list].iloc[:, 1:(1 + years)].sum(axis=0)

# eps vector 값이 모두 0일 경우 리턴값을 설정합니다

if (output\_vector == 0).all():

print("INFO : All values are zero ---> return 0")

return series([0] \* len(fs\_meta["cis"].iloc[0, 1:(1 + years)].index), fs\_meta["bs"].iloc[0, 1:(1 + years)].index)

# 아니면 eps vector 를 리턴합니다

else:

return output\_vector

# driver가 찾아지지 않은 경우, 리턴값을 설정합니다

else:

print("ERROR : Any driver is not found ---> return 0")

return series([0] \* len(fs\_meta["cis"].iloc[0, 1:(1 + years)].index), fs\_meta["bs"].iloc[0, 1:(1 + years)].index)

# 연도별 현금성자산 (not used)

def getCash(fs\_meta, years=3, multidriverException="sum"):

condition\_list = ['현금및현금성자산', '현금']

primary\_check\_list = fs\_meta["bs"]["label\_ko"].apply(lambda x: x in condition\_list)

# 계정과목을 탐색하여 vector를 생성하고, 존재하지 않을 경우 0인 벡터를 생성합니다

if primary\_check\_list.sum() > 0:

if primary\_check\_list.sum() > 1: print("WARNING : Multi drivers are detected")

# 계정과목이 하나 이상 탐지된 경우 제일 첫 번째 열을 사용합니다

if multidriverException == "first":

output\_vector = fs\_meta["bs"][primary\_check\_list].iloc[0, 1:(1 + years)]

# 계정과목이 여러 개 탐지된 경우 모두 합산한 값을 사용합니다

else:

output\_vector = fs\_meta["bs"][primary\_check\_list].iloc[:, 1:(1 + years)].sum(axis=0)

# eps vector 값이 모두 0일 경우 리턴값을 설정합니다

if (output\_vector == 0).all():

print("INFO : All values are zero ---> return 0")

return series([0] \* len(fs\_meta["cis"].iloc[0, 1:(1 + years)].index), fs\_meta["bs"].iloc[0, 1:(1 + years)].index)

# 아니면 eps vector 를 리턴합니다

else:

return output\_vector

# driver가 찾아지지 않은 경우, 리턴값을 설정합니다

else:

print("ERROR : Any driver is not found ---> return 0")

return series([0] \* len(fs\_meta["cis"].iloc[0, 1:(1 + years)].index), fs\_meta["bs"].iloc[0, 1:(1 + years)].index)

# 연도별 일평균 주가 (종가기준)

def getStockPrice(fs\_meta, stock\_dates=(), item\_name="종가", op="mean"):

if type(stock\_dates) in (tuple, list):

from\_date = stock\_dates[0]

to\_date = stock\_dates[1]

else:

from\_date = (pd.to\_datetime(datetime.now()) - DateOffset(days=stock\_dates)).strftime('%Y%m%d')

to\_date = pd.to\_datetime(datetime.now()).strftime('%Y%m%d')

if op == "mean":

ohlc = stock.get\_market\_ohlcv\_by\_date(fromdate=from\_date, todate=to\_date, ticker=fs\_meta["stock\_code"]).resample("1Y").mean().transpose()

ohlc.columns = ohlc.columns.to\_series().apply(lambda x: x.strftime("%Y%m%d"))

ohlc.reset\_index(inplace=True)

ohlc.columns = ["label\_ko"] + list(ohlc.columns[1:])

ohlc.columns.name = None

ohlc = ohlc[ohlc["label\_ko"] == item\_name].iloc[0, 1:]

ohlc.sort\_index(ascending=False, inplace=True)

else:

ohlc = stock.get\_market\_ohlcv\_by\_date(fromdate=from\_date, todate=to\_date, ticker=fs\_meta["stock\_code"])

return ohlc