import os

import IPython

import multiprocessing

import copy

import pickle

import warnings

from datetime import datetime

from time import time

from matplotlib import font\_manager as fm, rc, rcParams

import matplotlib.pyplot as plt

import seaborn as sns

import re

import numpy as np

from numpy import array, nan, random as rnd, where as which

import pandas as pd

from pandas import DataFrame as dataframe, Series as series, isna, read\_csv

from pandas.tseries.offsets import DateOffset

from scipy.special import boxcox1p

from scipy.stats import skew

from scipy import stats

# from sklearn import preprocessing as prep

# from sklearn.impute import KNNImputer

# from sklearn.ensemble import IsolationForest

# from sklearn.model\_selection import train\_test\_split as tts, GridSearchCV as GridTuner, StratifiedKFold, KFold

# from sklearn.feature\_selection import SelectFromModel

# from sklearn.preprocessing import OneHotEncoder, StandardScaler, MinMaxScaler

# from sklearn import metrics

# from sklearn.pipeline import make\_pipeline

# from sklearn import linear\_model as lm

# from sklearn.discriminant\_analysis import QuadraticDiscriminantAnalysis as qda

# from sklearn import svm

# import lightgbm as lgb

# import xgboost as xgb

# import catboost as cat

# from sklearn import neighbors as knn

# from sklearn import ensemble

#

# import tensorflow as tf

# from tensorflow.keras.models import Model

# from tensorflow.keras import layers

# from tensorflow.keras import activations

# from tensorflow.keras import optimizers

# from tensorflow.keras import metrics as tf\_metrics

# from tensorflow.keras import callbacks as tf\_callbacks

# from tqdm.keras import TqdmCallback

# from scikeras.wrappers import KerasClassifier, KerasRegressor

# import tensorflow\_addons as tfa

# import keras\_tuner as kt

# from keras\_tuner import HyperModel

# display setting

warnings.filterwarnings(action='ignore')

rcParams['axes.unicode\_minus'] = False

pd.set\_option('display.max\_columns', 100)

pd.set\_option('display.max\_rows', 100)

pd.set\_option('display.width', 1000)

# # font setting

# font\_path = 'myfonts/NanumSquareB.ttf'

# font\_obj = fm.FontProperties(fname=font\_path, size=12).get\_name()

# rc('font', family=font\_obj)

# %reset -f

# ===== utility functions =====

# label encoding for categorical column with excepting na value

class MyLabelEncoder:

def \_\_init\_\_(self, preset={}):

# dic\_cat format -> {"col\_name": {"value": replace}}

self.dic\_cat = preset

def fit\_transform(self, data\_x, col\_names):

tmp\_x = copy.deepcopy(data\_x)

for i in col\_names:

# type check

if not ((tmp\_x[i].dtype.name == "object") or (tmp\_x[i].dtype.name == "category")):

print(F"WARNING : {i} is not object or category")

# if key is not in dic, update dic

if i not in self.dic\_cat.keys():

tmp\_dic = dict.fromkeys(sorted(set(tmp\_x[i]).difference([nan])))

label\_cnt = 0

for j in tmp\_dic.keys():

tmp\_dic[j] = label\_cnt

label\_cnt += 1

self.dic\_cat[i] = tmp\_dic

# transform value which is not in dic to nan

tmp\_x[i] = tmp\_x[i].astype("object")

conv = tmp\_x[i].replace(self.dic\_cat[i])

for conv\_idx, j in enumerate(conv):

if j not in self.dic\_cat[i].values():

conv[conv\_idx] = nan

# final return

tmp\_x[i] = conv.astype("float")

return tmp\_x

def transform(self, data\_x, col\_names):

tmp\_x = copy.deepcopy(data\_x)

for i in col\_names:

if not ((tmp\_x[i].dtype.name == "object") or (tmp\_x[i].dtype.name == "category")):

print(F"WARNING : {i} is not object or category")

# transform value which is not in dic to nan

tmp\_x[i] = tmp\_x[i].astype("object")

conv = tmp\_x[i].replace(self.dic\_cat[i])

for conv\_idx, j in enumerate(conv):

if j not in self.dic\_cat[i].values():

conv[conv\_idx] = nan

# final return

tmp\_x[i] = conv.astype("float")

return tmp\_x

def clear(self, dic\_cat={}):

self.dic\_cat = dic\_cat

class MyOneHotEncoder:

def \_\_init\_\_(self, label\_preset={}):

self.dic\_cat = {}

self.label\_preset = label\_preset

def fit\_transform(self, data\_x, col\_names):

tmp\_x = dataframe()

for i in data\_x:

if i not in col\_names:

tmp\_x = pd.concat([tmp\_x, dataframe(data\_x[i])], axis=1)

else:

if not ((data\_x[i].dtype.name == "object") or (data\_x[i].dtype.name == "category")):

print(F"WARNING : {i} is not object or category")

self.dic\_cat[i] = OneHotEncoder(sparse=False, handle\_unknown="ignore")

conv = self.dic\_cat[i].fit\_transform(dataframe(data\_x[i])).astype("int")

col\_list = []

for j in self.dic\_cat[i].categories\_[0]:

if i in self.label\_preset.keys():

for k, v in self.label\_preset[i].items():

if v == j:

col\_list.append(str(i) + "\_" + str(k))

else:

col\_list.append(str(i) + "\_" + str(j))

conv = dataframe(conv, columns=col\_list)

tmp\_x = pd.concat([tmp\_x, conv], axis=1)

return tmp\_x

def transform(self, data\_x, col\_names):

tmp\_x = dataframe()

for i in data\_x:

if not i in col\_names:

tmp\_x = pd.concat([tmp\_x, dataframe(data\_x[i])], axis=1)

else:

if not ((data\_x[i].dtype.name == "object") or (data\_x[i].dtype.name == "category")):

print(F"WARNING : {i} is not object or category")

conv = self.dic\_cat[i].transform(dataframe(data\_x[i])).astype("int")

col\_list = []

for j in self.dic\_cat[i].categories\_[0]:

if i in self.label\_preset.keys():

for k, v in self.label\_preset[i].items():

if v == j: col\_list.append(str(i) + "\_" + str(k))

else:

col\_list.append(str(i) + "\_" + str(j))

conv = dataframe(conv, columns=col\_list)

tmp\_x = pd.concat([tmp\_x, conv], axis=1)

return tmp\_x

def clear(self, dic\_cat={}, label\_preset={}):

self.dic\_cat = dic\_cat

self.label\_preset = label\_preset

class MyKNNImputer:

def \_\_init\_\_(self, k=5):

self.imputer = KNNImputer(n\_neighbors=k)

self.cat\_dic = {}

self.naidx\_dix = {}

def fit\_transform(self, x, y, cat\_vars=None):

for i in cat\_vars:

self.cat\_dic[i] = diff(list(sorted(set(x[i]))), [nan])

self.naidx\_dix[i] = list(which(array(x[i].isna()))[0])

x\_imp = dataframe(self.imputer.fit\_transform(x, y), columns=x.columns)

# if imputed categorical value are not in the range, adjust the value

for i in cat\_vars:

x\_imp[i] = x\_imp[i].apply(lambda x: int(round(x, 0)))

for j in self.naidx\_dix[i]:

if x\_imp[i][j] not in self.cat\_dic[i]:

if x\_imp[i][j] < self.cat\_dic[i][0]:

x\_imp[i][self.naidx\_dix[i]] = self.cat\_dic[i][0]

elif x\_imp[i][j] > self.cat\_dic[i][0]:

x\_imp[i][self.naidx\_dix[i]] = self.cat\_dic[i][len(self.cat\_dic[i]) - 1]

return x\_imp

def transform(self, x):

for i in self.cat\_dic.keys():

self.naidx\_dix[i] = list(which(array(x[i].isna()))[0])

x\_imp = dataframe(self.imputer.transform(x), columns=x.columns)

# if imputed categorical value are not in the range, adjust the value

for i in self.cat\_dic.keys():

x\_imp[i] = x\_imp[i].apply(lambda x: int(round(x, 0)))

for j in self.naidx\_dix[i]:

if x\_imp[i][j] not in self.cat\_dic[i]:

if x\_imp[i][j] < self.cat\_dic[i][0]:

x\_imp[i][self.naidx\_dix[i]] = self.cat\_dic[i][0]

elif x\_imp[i][j] > self.cat\_dic[i][0]:

x\_imp[i][self.naidx\_dix[i]] = self.cat\_dic[i][len(self.cat\_dic[i]) - 1]

return x\_imp

def clear(self, cat\_dic={}, naidx\_dix={}):

self.cat\_dic = cat\_dic

self.naidx\_dix = naidx\_dix

def easyIO(x=None, path=None, op="r"):

tmp = None

if op == "r":

with open(path, "rb") as f:

tmp = pickle.load(f)

return tmp

elif op == "w":

tmp = {}

print(x)

if type(x) is dict:

for k in x.keys():

if "MLP" in k:

tmp[k] = {}

for model\_comps in x[k].keys():

if model\_comps != "model":

tmp[k][model\_comps] = copy.deepcopy(x[k][model\_comps])

print(F"INFO : {k} model is removed (keras)")

else:

tmp[k] = x[k]

if input("Write [y / n]: ") == "y":

with open(path, "wb") as f:

pickle.dump(tmp, f)

print("operation success")

else:

print("operation fail")

else:

print("Unknown operation type")

def diff(first, second):

second = set(second)

return [item for item in first if item not in second]

def findIdx(data\_x, col\_names):

return [int(i) for i, j in enumerate(data\_x) if j in col\_names]

def orderElems(for\_order, using\_ref):

return [i for i in using\_ref if i in for\_order]

# concatenate by row

def ccb(df1, df2):

if type(df1) == series:

tmp\_concat = series(pd.concat([dataframe(df1), dataframe(df2)], axis=0, ignore\_index=True).iloc[:,0])

tmp\_concat.reset\_index(drop=True, inplace=True)

elif type(df1) == dataframe:

tmp\_concat = pd.concat([df1, df2], axis=0, ignore\_index=True)

tmp\_concat.reset\_index(drop=True, inplace=True)

elif type(df1) == np.ndarray:

tmp\_concat = np.concatenate([df1, df2], axis=0)

else:

print("Unknown Type: return 1st argument")

tmp\_concat = df1

return tmp\_concat

def change\_width(ax, new\_value):

for patch in ax.patches :

current\_width = patch.get\_width()

adj\_value = current\_width - new\_value

# we change the bar width

patch.set\_width(new\_value)

# we recenter the bar

patch.set\_x(patch.get\_x() + adj\_value \* .5)

def dispPerformance(result\_dic, result\_metrics):

perf\_table = dataframe(columns=result\_metrics)

for k, v in result\_dic.items():

perf\_table = pd.concat([perf\_table, v["performance"]], ignore\_index=True, axis=0)

print(perf\_table)

return perf\_table

# from sklearn import datasets

# import yfinance

import dart\_fss as dart

# # dart\_fss\_classifier Plugin 불러오기 (IFRS 도입(2011년) 전의 재무제표와의 호환성)

# import dart\_fss\_classifier

# # Attach plugin

# assert dart\_fss\_classifier.attached\_plugin() == True

from pykrx import stock

api\_key = '730e8899e231f24386cdbff47d900047cf016caf'

dart.set\_api\_key(api\_key)

corp\_list = dart.corp.CorpList()

# raw data 로드 함수

def getFinancialStatement(tickers, from\_date="20140101", end\_date="20210930", report\_type=["annual"]):

# create dictionary for dataset

# 기업들을 저장할 딕셔너리 초기화

fs\_dic = dict.fromkeys(tickers, None)

# for loop on tickers

for i in tickers:

print("Financial Statement Loading --->", i)

try:

# 각 기업의 재무데이터를 저장할 딕셔너리 초기화

fs\_dic[i] = {}

# 재무제표 로딩

corp\_obj = corp\_list.find\_by\_corp\_name(i, exactly=True)[0]. \

extract\_fs(bgn\_de=from\_date, end\_de=end\_date, report\_tp=report\_type, separator=False)

fs\_dic[i]["bs"] = corp\_obj["bs"]

fs\_dic[i]["is"] = corp\_obj["is"]

fs\_dic[i]["cis"] = corp\_obj["cis"]

fs\_dic[i]["cf"] = corp\_obj["cf"]

fs\_dic[i]["metrics"] = {}

fs\_dic[i]["stock\_code"] = {}

except:

# 만약 로딩 시 에러가 발생하면 해당 키 (해당 기업)을 딕셔너리에서 삭제

print('EXCEPTION : Error occurs, delete the key')

del fs\_dic[i]

return fs\_dic

# 재무제표 데이터의 적절한 컬럼을 추출하는 함수 (2차원 컬럼 -> 1차원 컬럼)

def clearFinancialStatement(fs):

fs\_copy = copy.deepcopy(fs)

try:

for i in list(fs.keys()):

print("===== Columns clearing", i, "=====")

# label\_ko 열에서 한글만 추출하기 위한 re 조건식 지정

expression\_hangul = re.compile('[^가-힣+]')

# Balance Sheet

# 2차원 컬럼에서 필요한 열을 추출해 저장할 리스트 생성

selected\_idx = []

selected\_column = []

# 2중 컬럼에 대한 for loop 수행

for idx, name in enumerate(fs\_copy[i]["bs"]):

# 튜플 형식으로 넘어오는 name 에 대해 검사 수행

# 만약 label\_ko (매출액, 당기순이익 등의 계정과목이 저장된 열) 이면 저장

if name[1] == "label\_ko":

selected\_idx.append(idx)

selected\_column.append(name[1])

# 일자의 경우 튜플 형식으로 넘어오므로, 튜플 확인 조건문 수행

# 만약 튜플이고, 해당 튜플의 첫번쨰 튜플 원소가 '연결재무제표' 인 경우 저장 (오타단어 포함)

if type(name[1]) == tuple and (name[1][0] == "연결재무제표" or name[1][0] == "연결재무재표"):

selected\_idx.append(idx)

selected\_column.append(name[0])

# 반복문의 통해 선택된 열 인덱스 및 열 이름으로 새로운 데이터프레임 생성

tmp\_df = fs\_copy[i]["bs"].iloc[:, selected\_idx]

tmp\_df.columns = selected\_column

# label\_ko 의 문자열에 대해 한글만 추출한 뒤 공백을 모두 제거

tmp\_df["label\_ko"] = tmp\_df["label\_ko"].apply(lambda x: expression\_hangul.sub("", x).replace(" ", ""))

# na 및 공백을 모두 0으로 대치

fs\_copy[i]["bs"] = tmp\_df.fillna(0.0)

fs\_copy[i]["bs"].replace(["", " "], 0.0, inplace=True)

# 각 컬럼의 일자를 YYYYMMDD 형식으로 통일 (fiscal year 의 마지막 일자)

tmp\_cols = ["label\_ko"]

for j in fs\_copy[i]["bs"].columns[1:]:

tmp\_cols.append(j.split("-")[-1])

fs\_copy[i]["bs"].columns = tmp\_cols

# 중복 컬럼이 있는 데이터에 대해서는 0이 제일 적은 열만 선택

dup\_cols = fs\_copy[i]["bs"].columns.value\_counts().index[fs\_copy[i]["bs"].columns.value\_counts() > 1]

if len(dup\_cols) != 0:

print("WARNING : duplicated date columns exist (column which has the least zero values is selected)")

min\_zero\_col = np.inf

drop\_cols = []

for j in dup\_cols:

dup\_col\_idx = which(fs\_copy[i]["bs"].columns == j)[0]

for z in dup\_col\_idx:

if (fs\_copy[i]["bs"].iloc[:, z] == 0).sum() <= min\_zero\_col:

min\_zero\_col = z

drop\_cols = drop\_cols + diff(dup\_col\_idx, [z])

fs\_copy[i]["bs"] = fs\_copy[i]["bs"].iloc[:, diff(list(range(len(fs\_copy[i]["bs"].columns))), drop\_cols)]

# 계정과목에 해당 하는 값이 모두 0인 경우 (label\_ko 열을 제외한 모든 열의 값이 0인 경우) 해당 계정과목을 제외시킵니다

fs\_copy[i]["bs"] = fs\_copy[i]["bs"][(fs\_copy[i]["bs"].iloc[:,1:] != 0).any(True)]

fs\_copy[i]["bs"].reset\_index(drop=True, inplace=True)

# 손익계산서가 None 이면 포괄손익계산서를 사용

# 손익계산서가 None 이 아니면 포괄손익계산서(cis) 키값에 대입 후 손익계산서(is) 키는 사용 하지 않음

if fs\_copy[i]["is"] is None:

# Consolidated Income Statement

selected\_idx\_cis = []

selected\_column\_cis = []

for idx, name in enumerate(fs\_copy[i]["cis"]):

if name[1] == "label\_ko":

selected\_idx\_cis.append(idx)

selected\_column\_cis.append(name[1])

if type(name[1]) == tuple and (name[1][0] == "연결재무제표" or name[1][0] == "연결재무재표"):

selected\_idx\_cis.append(idx)

selected\_column\_cis.append(name[0])

tmp\_df = fs\_copy[i]["cis"].iloc[:, selected\_idx\_cis]

tmp\_df.columns = selected\_column\_cis

tmp\_df["label\_ko"] = tmp\_df["label\_ko"].apply(lambda x: expression\_hangul.sub("", x).replace(" ", ""))

fs\_copy[i]["cis"] = tmp\_df.fillna(0.0)

fs\_copy[i]["cis"].replace(["", " "], 0.0, inplace=True)

else:

# Income Statement

selected\_idx\_cis = []

selected\_column\_cis = []

for idx, name in enumerate(fs\_copy[i]["is"]):

if name[1] == "label\_ko":

selected\_idx\_cis.append(idx)

selected\_column\_cis.append(name[1])

if type(name[1]) == tuple and (name[1][0] == "연결재무제표" or name[1][0] == "연결재무재표"):

selected\_idx\_cis.append(idx)

selected\_column\_cis.append(name[0])

tmp\_df = fs\_copy[i]["is"].iloc[:, selected\_idx\_cis]

tmp\_df.columns = selected\_column\_cis

tmp\_df["label\_ko"] = tmp\_df["label\_ko"].apply(lambda x: expression\_hangul.sub("", x).replace(" ", ""))

fs\_copy[i]["cis"] = tmp\_df.fillna(0.0)

fs\_copy[i]["cis"].replace(["", " "], 0.0, inplace=True)

del fs\_copy[i]["is"]

tmp\_cols = ["label\_ko"]

for j in fs\_copy[i]["cis"].columns[1:]:

tmp\_cols.append(j.split("-")[-1])

fs\_copy[i]["cis"].columns = tmp\_cols

dup\_cols = fs\_copy[i]["cis"].columns.value\_counts().index[fs\_copy[i]["cis"].columns.value\_counts() > 1]

if len(dup\_cols) != 0:

print("WARNING : duplicated date columns exist (column which has the least zero values is selected)")

min\_zero\_col = np.inf

drop\_cols = []

for j in dup\_cols:

dup\_col\_idx = which(fs\_copy[i]["cis"].columns == j)[0]

for z in dup\_col\_idx:

if (fs\_copy[i]["cis"].iloc[:, z] == 0).sum() <= min\_zero\_col:

min\_zero\_col = z

drop\_cols = drop\_cols + diff(dup\_col\_idx, [z])

fs\_copy[i]["cis"] = fs\_copy[i]["cis"].iloc[:, diff(list(range(len(fs\_copy[i]["cis"].columns))), drop\_cols)]

fs\_copy[i]["cis"] = fs\_copy[i]["cis"][(fs\_copy[i]["cis"].iloc[:, 1:] != 0).any(True)]

fs\_copy[i]["cis"].reset\_index(drop=True, inplace=True)

# Cash Flow Chart

selected\_idx\_cf = []

selected\_column\_cf = []

for idx, name in enumerate(fs\_copy[i]["cf"]):

if name[1] == "label\_ko":

selected\_idx\_cf.append(idx)

selected\_column\_cf.append(name[1])

if type(name[1]) == tuple and (name[1][0] == "연결재무제표" or name[1][0] == "연결재무재표"):

selected\_idx\_cf.append(idx)

selected\_column\_cf.append(name[0])

tmp\_df = fs\_copy[i]["cf"].iloc[:, selected\_idx\_cf]

tmp\_df.columns = selected\_column\_cf

tmp\_df["label\_ko"] = tmp\_df["label\_ko"].apply(lambda x: expression\_hangul.sub("", x).replace(" ", ""))

fs\_copy[i]["cf"] = tmp\_df.fillna(0.0)

fs\_copy[i]["cf"].replace(["", " "], 0.0, inplace=True)

tmp\_cols = ["label\_ko"]

for j in fs\_copy[i]["cf"].columns[1:]:

tmp\_cols.append(j.split("-")[-1])

fs\_copy[i]["cf"].columns = tmp\_cols

dup\_cols = fs\_copy[i]["cf"].columns.value\_counts().index[fs\_copy[i]["cf"].columns.value\_counts() > 1]

if len(dup\_cols) != 0:

print("WARNING : duplicated date columns exist (column which has the least zero values is selected)")

min\_zero\_col = np.inf

drop\_cols = []

for j in dup\_cols:

dup\_col\_idx = which(fs\_copy[i]["cf"].columns == j)[0]

for z in dup\_col\_idx:

if (fs\_copy[i]["cf"].iloc[:, z] == 0).sum() <= min\_zero\_col:

min\_zero\_col = z

drop\_cols = drop\_cols + diff(dup\_col\_idx, [z])

fs\_copy[i]["cf"] = fs\_copy[i]["cf"].iloc[:, diff(list(range(len(fs\_copy[i]["cf"].columns))), drop\_cols)]

fs\_copy[i]["cf"] = fs\_copy[i]["cf"][(fs\_copy[i]["cf"].iloc[:, 1:] != 0).any(True)]

fs\_copy[i]["cf"].reset\_index(drop=True, inplace=True)

return fs\_copy

except:

print("ERROR : return original")

return fs

# 각 기업 재무제표의 컬럼수 조정 및 일자 매칭 여부 점검 함수

def alignFinancialStatement(fs, cut\_off\_year=2019, min\_recent\_year=3, column\_unmatched\_exception="remove"):

'''

:param fs: 특정 기업의 재무데이터가 있는 딕셔너리

:param cut\_off\_year: 재무제표에서 제외시킬 최소년도 (ex. 2019 으로 설정된 경우 2019년 이상인 년도는 모두 drop)

:param min\_recent\_year: 최소로 존재해야 할 재무제표 (ex. 2019, 2018 년만 존재할 경우 해당 기업 drop)

:param column\_unmatched\_exception: bs, cis, cf 의 컬럼 일자가 맞지 않을 경우 처리 방법을 지정 (default : "remove")

:return: 정상수행 시 전처리된 컬럼을 가진 fs를 리턴, 오류발생 시 원본 fs 를 리턴

'''

fs\_copy = copy.deepcopy(fs)

try:

for k in list(fs.keys()):

print("===== Columns preprocessing", k, "=====")

# balance sheet, income statement, cash flow chart 의 컬럼 수를 비교

col\_length\_check = array([fs\_copy[k]["bs"].shape[1], fs\_copy[k]["cis"].shape[1], fs\_copy[k]["cf"].shape[1]])

column\_cutoff = col\_length\_check.min()

# 만약 컬럼 수가 다르면 수행

if col\_length\_check.var() != 0.0:

print("WARNING : Column's lengths are not same", col\_length\_check)

# 각 재무제표의 첫 열 중 가장 작은 값 및 마지막 열 중 가장 큰 값을 계산 (min\_col\_first ~ max\_col\_last 범위의 열 선택이 목적)

min\_col\_first = str(min(array([fs\_copy[k]["bs"].columns[1], fs\_copy[k]["cis"].columns[1], fs\_copy[k]["cf"].columns[1]], dtype="int")))

max\_col\_last = str(max(array([fs\_copy[k]["bs"].columns[-1], fs\_copy[k]["cis"].columns[-1], fs\_copy[k]["cf"].columns[-1]], dtype="int")))

# 강제로 가장 적은 열을 가진 재무데이터에 다른 데이터들을 맞추도록 하는 flag 설정

min\_cutoff\_flag = False

# slicing 할 열의 인덱스 위치를 저장할 딕셔너리 생성

cutoff\_index = {"bs": None, "cis": None, "cf": None}

for i in ["bs", "cis", "cf"]:

# 위에서 선정한 min\_col\_first 및 max\_col\_last 가 각각의 재무제표 열 안에 존재하는지 확인

# 만약 존재 하지 않으면 강제로 가장 작은 열을 가진 재무데이터에 다른 재무데이터들을 마지막 열부터 드랍하여 맞춤

if (min\_col\_first not in fs\_copy[k][i].columns) or (max\_col\_last not in fs\_copy[k][i].columns):

print("WARNING : an interval date doesn't exist in column sequence")

# 강제 설정되었으므로 flag를 True로 변경

min\_cutoff\_flag = True

# 모든 재무데이터에 일자가 존재해야하므로 한 경우라도 발생하면 break

break

else:

# 일자가 열에 존재하면 min\_col\_first 와 max\_col\_first 의 인덱스 위치를 찾아 cut\_off\_index 딕셔너리에 저장

cutoff\_index[i] = (which(fs\_copy[k][i].columns == min\_col\_first)[0][0], which(fs\_copy[k][i].columns == max\_col\_last)[0][0])

# min\_cutoff\_flag 가 False 이면

if min\_cutoff\_flag:

fs\_copy[k]["bs"] = fs\_copy[k]["bs"].iloc[:, 0:column\_cutoff]

fs\_copy[k]["cis"] = fs\_copy[k]["cis"].iloc[:, 0:column\_cutoff]

fs\_copy[k]["cf"] = fs\_copy[k]["cf"].iloc[:, 0:column\_cutoff]

else:

# 각 재무데이터를 min\_col\_first ~ max\_col\_last 일자로 맞춤

for i in ["bs", "cis", "cf"]:

fs\_copy[k][i] = fs\_copy[k][i].iloc[:, [0] + list(range(cutoff\_index[i][0], cutoff\_index[i][1]+1))]

length\_check\_after\_cutting = array([fs\_copy[k]["bs"].shape[1], fs\_copy[k]["cis"].shape[1], fs\_copy[k]["cf"].shape[1]])

# 만약 중간에 비어있는 일자 떄문에 재무데이터 간 컬럼 갯수가 맞지 않으면 해당 기업 drop (ex. 2019 2018 2016 과 같은 열)

if length\_check\_after\_cutting.var() != 0:

print("WARNING : lengths are not same after cutting")

del fs\_copy[k]

continue

else:

column\_cutoff = length\_check\_after\_cutting.min()

# 최종 슬라이싱 된 각 재무데이터의 열을 출력

print("After cutting :", array([fs\_copy[k]["bs"].shape[1], fs\_copy[k]["cis"].shape[1], fs\_copy[k]["cf"].shape[1]]))

# bs, cis, cf 의 컬럼의 날짜를 align 하는 부분

# 열 일자 매칭 검사에 대한 해당 기업 drop 여부 flag

col\_notmatch\_flag = False

# label\_ko을 제외한 (계정과목 관련) 데이터에 대해 for loop 수행

for i in range(1, column\_cutoff):

cols\_date = array([fs\_copy[k]["bs"].columns[i], fs\_copy[k]["cis"].columns[i], fs\_copy[k]["cf"].columns[i]]).astype("int32")

# 만약 각 재무제표들의 컬럼 일자가 맞지 않으면 경고 메시지 출력

if cols\_date.var() != 0:

print("WARNING : Cross check error on each columns' date")

print(cols\_date)

# column\_unmatched\_exception 파라미터가 "remove" 이면 해당 기업 drop

if column\_unmatched\_exception == "remove":

print("DELETE : column\_unmatched\_exception == 'remove'")

del fs\_copy[k]

col\_notmatch\_flag = True

# 한 열이라도 맞지 않으면 작업을 수행할 수 없으므로 한 번 조건 통과 시 break

break

# col\_notmath\_flag 가 False 이면 수행

if not col\_notmatch\_flag:

# cut\_off\_year 미만인 컬럼의 인덱스를 구함 (label\_ko 는 강제 포함)

seleted\_cols = list(which(pd.to\_datetime(series(fs\_copy[k]["bs"].drop("label\_ko", axis=1).columns)).dt.year < cut\_off\_year)[0] + 1)

# 각 재무제표들을 선택된 인덱스를 슬라이싱 하여 재저장 후 이름을 bs의 열 이름으로 통합

fs\_copy[k]["bs"] = fs\_copy[k]["bs"].iloc[:, ([0] + seleted\_cols)]

fs\_copy[k]["cis"] = fs\_copy[k]["cis"].iloc[:, ([0] + seleted\_cols)]

fs\_copy[k]["cf"] = fs\_copy[k]["cf"].iloc[:, ([0] + seleted\_cols)]

if k == "KG이니시스":

print(fs\_copy[k]["cis"])

# 최소 요구년도보다 적은 열만 있는 기업 드랍

if (fs\_copy[k]["bs"].shape[1] < min\_recent\_year + 1):

print("DELETE : financial data is less than", min\_recent\_year)

del fs\_copy[k]

continue

# 최소 요구년도 이상의 열 중 첫 열의 년도가 cut\_off\_year - 1 이 아니면 해당 기업 드랍

if (pd.to\_datetime(fs\_copy[k]["bs"].columns[1]).year != cut\_off\_year-1):

print("DELETE : first column is not", cut\_off\_year-1)

del fs\_copy[k]

return fs\_copy

except:

print("ERROR : return original")

return fs

# 해당 기업의 종목코드를 불러오는 함수

def getStockcode(fs):

fs\_copy = copy.deepcopy(fs)

for i in list(fs.keys()):

fs\_copy[i]["stock\_code"] = corp\_list.find\_by\_corp\_name(i, exactly=True)[0].info["stock\_code"]

return fs\_copy

sector\_list = ["comm\_services", "discretionary", "energy",

"health\_care", "industrials", "it", "materials", "staples"]

sector = {}

start\_time = time()

for i in sector\_list:

sector[i] = easyIO(None, "./kdigital\_project3/fs\_rawdata\_" + i + ".pickle", op="r")

sector[i] = clearFinancialStatement(sector[i])

sector[i] = alignFinancialStatement(sector[i], cut\_off\_year=2019, min\_recent\_year=3, column\_unmatched\_exception="remove")

sector[i] = getStockcode(sector[i])

time() - start\_time

# 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)

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

condition\_list = ["지배기업의소유주에게귀속되는당기순이익손실", "지배기업의소유주지분"]

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("Mixed EPS values are all zeros ---> pass")

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 : EPS values are all zeros ---> 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 : EPS values are all zeros ---> 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] \* 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]

# ===== diluted eps 대치 =====

# eps vector 에 0인 값이 하나라도 있으면

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

print("diluted eps replacement process")

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():

print("ERROR : EPS values are all zeros ---> 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

# 연도별 법인세

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

condition\_list = ["법인세의납부", "법인세비용수익", "법인세납부환급", "법인세납부", "법인세비용", "법인세납부액", "법인세의환급지급", "법인세환급",

"법인세환급납부", "법인세납부영업", "법인세비용", "법인세납부액환급액"]

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

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

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

print(fs\_meta["cf"][primary\_check\_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)

print(output\_vector)

# 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", "조정"]

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

condition\_list = ["자본총계"]

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

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

condition\_list = ["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"]

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

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

condition\_list = ["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", "계속영업에서창출된현금흐름",\

"영업활동으로부터창출된현금흐름","영업활동에서창출된현금흐름","영업활동으로창출된현금", "영업으로부터창출된현금" ]

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="first"):

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="first"):

condition\_list = ["배당금지급", "배당금의지급", '배당금의지급등']

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="first"):

condition\_list = ["자기주식의처분", '자기주식의취득']

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', '매출채권및대출채권',

'매출채권및기타수취채권', '매출채권및수취채권', '매출채권및계약자산', '단기매출채권', '유동매출채권및기타채권']

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

# ===== financial statement =====

# 1. 성장성 지표 (전체기간 평균 증가율)

# 매출액 평균 성장률

# 영입이익 평균 성장률

# EBITDA 평균 성장률

# EPS 평균 성장률

# 주가 평균 성장률

# 배당 평균 성장률

def getRateOfChange(df, years):

return np.mean([(df.iloc[0][i] / df.iloc[0][1 + i]) \* 100 - 100 for i in range(years - 1)]) / 100

def getGrowthIndicator(fs, years=7):

fs\_copy = copy.deepcopy(fs)

print(fs\_copy.keys())

for i in fs.keys():

print(i)

##매출액 연평균 성장률

a = getRevenue(fs\_copy[i], years=years)

a = a.to\_frame().T

# print(growth(a, years))

try:

fs\_copy[i]["metrics"]["yoy\_avg\_revenue"] = getRateOfChange(a, len(a.columns))

except:

fs\_copy[i]["metrics"]["yoy\_avg\_revenue"] = nan

##영업이익 연평균 성장률

b = getOperatingIncome(fs\_copy[i], years=years)

b = b.to\_frame().T

# print(growth(b, years))

try:

fs\_copy[i]["metrics"]["yoy\_avg\_operating\_income"] = getRateOfChange(b, len(b.columns))

except:

fs\_copy[i]["metrics"]["yoy\_avg\_operating\_income"] = nan

##EBITDA 연평균 성장률

if getAdjCFO(fs\_copy[i], years).mean() == 0:

c = getEarnings(fs\_copy[i], years) + \

getDepreciationAndAmortization(fs\_copy[i], years) - \

getInterestExpense(fs\_copy[i], years) - \

getTax(fs\_copy[i], years)

else:

c = getAdjCFO(fs\_copy[i], years) - \

getInterestExpense(fs\_copy[i], years) - \

getTax(fs\_copy[i], years)

c = c.to\_frame().T

# print(growth(b, years))

try:

fs\_copy[i]["metrics"]["yoy\_avg\_ebitda\_income"] = getRateOfChange(c, len(c.columns))

except:

fs\_copy[i]["metrics"]["yoy\_avg\_ebitda\_income"] = nan

##eps 연평균 성장률

d = getEPS(fs\_copy[i], years)

if isna(d.sum()) or d.sum() == 0:

d = getEPS(fs\_copy[i], years)

d = d.to\_frame().T

# print(growth(c, years))

try:

fs\_copy[i]["metrics"]["yoy\_avg\_eps"] = getRateOfChange(d, len(d.columns))

except:

fs\_copy[i]["metrics"]["yoy\_avg\_eps"] = nan

##배당 연평균 상승률

e = getDividend(fs\_copy[i], years)

e = e.to\_frame().T

# print(growth(c, years))

try:

fs\_copy[i]["metrics"]["yoy\_avg\_div"] = getRateOfChange(e, len(e.columns))

except:

fs\_copy[i]["metrics"]["yoy\_avg\_div"] = nan

return fs\_copy

# 2. 안정성 지표 (최근 3년 평균)

# Quick ratio = (유동자산 - 재고자산) / 유동부채

# 매출채권 회전율 = 매출액 / 매출채권

# 부채비율 = 부채 / 자기자본

# ICR(Interest Coverage Ratio) = EBITDA / 이자비용

def getStabilityIndicator(fs, years=3):

fs\_copy = copy.deepcopy(fs)

for i in list(fs\_copy.keys()):

print("===== Getting metrics on", i, "=====")

try:

fs\_copy[i]["metrics"]["quick\_ratio"] = ((getCurrentAsset(fs\_copy[i], years) - getInventory(fs\_copy[i], years)) / \

getCurrentLiab(fs\_copy[i], years)).mean()

except:

fs\_copy[i]["metrics"]["quick\_ratio"] = nan

try:

fs\_copy[i]["metrics"]["receivable\_turnover\_ratio"] = (getRevenue(fs\_copy[i], years) / getBond(fs\_copy[i], years)).mean()

except:

fs\_copy[i]["metrics"]["receivable\_turnover\_ratio"] = nan

try:

fs\_copy[i]["metrics"]["asset\_turnover\_ratio"] = (getRevenue(fs\_copy[i], years) / (getEquity(fs\_copy[i], years) + getDebt(fs\_copy[i], years))).mean()

except:

fs\_copy[i]["metrics"]["asset\_turnover\_ratio"] = nan

try:

fs\_copy[i]["metrics"]["debt\_to\_equity\_ratio"] = (getDebt(fs\_copy[i], years) / getEquity(fs\_copy[i], years)).mean()

except:

fs\_copy[i]["metrics"]["debt\_to\_equity\_ratio"] = nan

try:

if getAdjCFO(fs\_copy[i], years).mean() == 0:

ebitda = (getEarnings(fs\_copy[i], years) + \

getDepreciationAndAmortization(fs\_copy[i], years) + \

getInterestExpense(fs\_copy[i], years) + \

getTax(fs\_copy[i], years)).mean()

else:

ebitda = (getAdjCFO(fs\_copy[i], years) + \

getInterestExpense(fs\_copy[i], years) + \

getTax(fs\_copy[i], years)).mean()

fs\_copy[i]["metrics"]["icr"] = (ebitda / getInterestExpense(fs\_copy[i], years)).mean()

except:

fs\_copy[i]["metrics"]["icr"] = nan

print("===== Complete =====")

return fs\_copy

# 3. 수익성 지표 (최근 3년 평균)

# 매출액

# 영업이익

# 영업이익 마진

# EBITDA

# EBITDA 마진

# ROE = 당기순이익 / 자기자본

# CFO to Earnings = 영업현금흐름 / 당기순이익

def getProfitabilityIndicator(fs, years=3):

fs\_copy = copy.deepcopy(fs)

for i in list(fs\_copy.keys()):

print("===== Getting metrics on", i, "=====")

fs\_copy[i]["metrics"]["revenue"] = getRevenue(fs\_copy[i], years).mean()

fs\_copy[i]["metrics"]["operating\_income"] = getOperatingIncome(fs\_copy[i], years).mean()

fs\_copy[i]["metrics"]["operating\_margin"] = (getOperatingIncome(fs\_copy[i], years) / getRevenue(fs\_copy[i], years)).mean()

if getAdjCFO(fs\_copy[i], years).mean() == 0:

fs\_copy[i]["metrics"]["ebitda"] = (getEarnings(fs\_copy[i], years) + \

getDepreciationAndAmortization(fs\_copy[i], years) - \

getInterestExpense(fs\_copy[i], years) - \

getTax(fs\_copy[i], years)).mean()

fs\_copy[i]["metrics"]["ebitda\_margin"] = ((getEarnings(fs\_copy[i], years) + \

getDepreciationAndAmortization(fs\_copy[i], years) - \

getInterestExpense(fs\_copy[i], years) - \

getTax(fs\_copy[i], years)) / getRevenue(fs\_copy[i], years)).mean()

else:

fs\_copy[i]["metrics"]["ebitda"] = (getAdjCFO(fs\_copy[i], years) - \

getInterestExpense(fs\_copy[i], years) - \

getTax(fs\_copy[i], years)).mean()

fs\_copy[i]["metrics"]["ebitda\_margin"] = ((getAdjCFO(fs\_copy[i], years) - \

getInterestExpense(fs\_copy[i], years) - \

getTax(fs\_copy[i], years)) / getRevenue(fs\_copy[i], years)).mean()

try:

fs\_copy[i]["metrics"]["roe"] = (getEarnings(fs\_copy[i], years) / getEquity(fs\_copy[i], years)).mean()

except:

fs\_copy[i]["metrics"]["roe"] = nan

try:

fs\_copy[i]["metrics"]["cfo\_to\_earnings"] = (getCFO(fs\_copy[i], years) / getEarnings(fs\_copy[i], years)).mean()

except:

fs\_copy[i]["metrics"]["cfo\_to\_earnings"] = nan

print("===== Complete =====")

return fs\_copy

# 4. 가치평가 지표 (최근 1년)

# PSR = 1주당 주가 / 주당 매출액

# PER = 1주당 주가 / 주당 순이익

def getValuationIndicator(fs, years=1, stock\_dates=()):

fs\_copy = copy.deepcopy(fs)

for i in list(fs\_copy.keys()):

print("===== Getting metrics on", i, "=====")

eps = getEPS(fs\_copy[i], years, epsType="basic")

if isna(eps.sum()) or eps.sum() == 0:

eps = getEPS(fs\_copy[i], years, epsType="diluted")

stock\_price = getStockPrice(fs\_copy[i], stock\_dates)

if len(stock\_price.values) == 0:

fs\_copy[i]["metrics"]["psr"] = nan

fs\_copy[i]["metrics"]["per"] = nan

else:

fs\_copy[i]["metrics"]["psr"] = (stock\_price.values / getRevenue(fs\_copy[i], years).values).mean()

fs\_copy[i]["metrics"]["per"] = (stock\_price.values / eps.values).mean()

print("===== Complete =====")

return fs\_copy

# 5. 주주환원 지표 (최근 3년 평균)

# 주주환원율 = (배당금 + 자사주매입금) / 당기순이익

def getShareholderReturnIndicator(fs, years=3):

fs\_copy = copy.deepcopy(fs)

for i in list(fs\_copy.keys()):

print("===== Getting metrics on", i, "=====")

fs\_copy[i]["metrics"]["shareholder\_return"] = ((getDividend(fs\_copy[i], years) + getBuyback(fs\_copy[i], years)) / \

getEarnings(fs\_copy[i], years)).mean()

print("===== Complete =====")

return fs\_copy

def getBeta(portfolio, benchmark):

return (stats.linregress(portfolio.pct\_change()[1:].values, benchmark.pct\_change()[1:].values)[0])

def getJensenAlpah(fs, benchmark="KOSPI", riskfree\_rate=0.02, stock\_dates=()):

fs\_copy = copy.deepcopy(fs)

comp\_df = dataframe(columns=["portfolio", "benchmark"])

if benchmark == "KOSPI":

stock\_df = stock.get\_index\_ohlcv\_by\_date(stock\_dates[0], stock\_dates[1], "1001")

comp\_df["benchmark"] = stock\_df["종가"]

comp\_df.index = stock\_df.index

else:

pass

for i in list(fs.keys()):

comp\_df["portfolio"] = getStockPrice(fs\_copy[i], stock\_dates=(stock\_dates[0], stock\_dates[1]), op="raw")["종가"]

return\_portfolio = (comp\_df["portfolio"][-1] - comp\_df["portfolio"][0]) / comp\_df["portfolio"][0]

return\_benchmark = (comp\_df["benchmark"][-1] - comp\_df["benchmark"][0]) / comp\_df["benchmark"][0]

beta = getBeta(comp\_df["portfolio"], comp\_df["benchmark"])

fs\_copy[i]["metrics"]["jensen\_alpha"] = 1 if (return\_portfolio - (riskfree\_rate + beta \* (return\_benchmark - riskfree\_rate))) >= 0 else 0

return fs\_copy

for i in sector.keys():

sector[i] = getGrowthIndicator(sector[i])

sector[i] = getStabilityIndicator(sector[i])

sector[i] = getProfitabilityIndicator(sector[i])

sector[i] = getValuationIndicator(sector[i], 1, ("20180101", "20181231"))

sector[i] = getShareholderReturnIndicator(sector[i])

sector[i] = getJensenAlpah(sector[i], stock\_dates=("20190101", "20191231"))

def transform\_frame(sector\_dict, sector\_key, col\_name):

df = dataframe(columns=col\_name)

for k, v in sector\_dict.items():

df = pd.concat([df, dataframe([v["metrics"].values()], columns=v["metrics"].keys())], axis=0)

df["sector"] = sector\_key

df.index = sector\_dict.keys()

df.index.name = "기업명"

return df

# metric 키값 구하기 및 sector 컬럼 결합

# sector iteration

for i in sector.keys():

# corporate iteration

for j in sector[i].keys():

col\_name = list(sector[i][j]["metrics"].keys()) + ["sector"]

break

# 빈데이터 프레임 생성

final\_df = dataframe(columns=col\_name)

# sector에 있는 데이터 모두 데이터프레임화

for key, item in sector.items():

a = transform\_frame(item, key, col\_name)

final\_df = pd.concat([final\_df, a], axis=0)

print(final\_df)