# 임포트 및 세팅

import dart\_fss as dart

import re

김영준 API key = '730e8899e231f24386cdbff47d900047cf016caf'

조준연 API key = ‘5969fe90a2b3773ede6935cfa3301015931f7545’

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

corp\_list = dart.get\_corp\_list()

# raw data 로드 함수

# 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

# pickle 파일 로드 및 전처리

# sector\_list = ["energy", "materials", "industrials", "discretionary",

# "staples", "health\_care", "it", "comm\_services"]

sector\_list = ["it"]

sector = {}

start\_time = time()

for i in sector\_list:

sector[i] = easyIO(None, "./kdigital\_project3/fs\_rawdata\_" + "it" + ".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

# 10/10 20:28 수정

# 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)])

def getGrowthIndicator(fs, years=5):

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

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

##eps 연평균 성장률

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

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

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

c = c.to\_frame().T

# print(growth(c, years))

try:

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

except:

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

# ##주가 연평균 상승률

# from\_date = pd.to\_datetime(str(pd.to\_datetime(fs\_copy[i]["bs"].columns[-1]).year) + "0101")

# to\_date = pd.to\_datetime(str(pd.to\_datetime(fs\_copy[i]["bs"].columns[1]).year) + "0101")

# d = getStockPrice(fs\_copy[i], years=(from\_date, to\_date))

# d = d.to\_frame().T

# # print(growth(c, years))

# try:

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

# except:

# fs\_copy[i]["metrics"]["yoy\_avg\_stock\_price"] = 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

# 10/07 19:21 수정

# 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

# 10/07 19:21 수정

# 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

# 10/6 수정

# 10/07 19:21 수정

# 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

# train dataset 만들기

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)

# train dataset 생성

print(final\_df)