# example of using sec api:

#from: https://github.com/AdamGetbags/secAPI/blob/main/secFilingScraper.py

# -\*- coding: utf-8 -\*-

"""

SEC Filing Scraper

@author: AdamGetbags

"""

# import modules

import requests

import pandas as pd

# create request header

headers = {'User-Agent': "email@address.com"}

# get all companies data

companyTickers = requests.get(

"https://www.sec.gov/files/company\_tickers.json",

headers=headers

)

# review response / keys

print(companyTickers.json().keys())

# format response to dictionary and get first key/value

firstEntry = companyTickers.json()['0']

# parse CIK // without leading zeros

directCik = companyTickers.json()['0']['cik\_str']

# dictionary to dataframe

companyData = pd.DataFrame.from\_dict(companyTickers.json(),

orient='index')

# add leading zeros to CIK

companyData['cik\_str'] = companyData['cik\_str'].astype(

str).str.zfill(10)

# review data

print(companyData[:1])

cik = companyData[0:1].cik\_str[0]

# get company specific filing metadata

filingMetadata = requests.get(

f'https://data.sec.gov/submissions/CIK{cik}.json',

headers=headers

)

# review json

print(filingMetadata.json().keys())

filingMetadata.json()['filings']

filingMetadata.json()['filings'].keys()

filingMetadata.json()['filings']['recent']

filingMetadata.json()['filings']['recent'].keys()

# dictionary to dataframe

allForms = pd.DataFrame.from\_dict(

filingMetadata.json()['filings']['recent']

)

# review columns

allForms.columns

allForms[['accessionNumber', 'reportDate', 'form']].head(50)

# 10-Q metadata

allForms.iloc[11]

# get company facts data

companyFacts = requests.get(

f'https://data.sec.gov/api/xbrl/companyfacts/CIK{cik}.json',

headers=headers

)

#review data

companyFacts.json().keys()

companyFacts.json()['facts']

companyFacts.json()['facts'].keys()

# filing metadata

companyFacts.json()['facts']['dei'][

'EntityCommonStockSharesOutstanding']

companyFacts.json()['facts']['dei'][

'EntityCommonStockSharesOutstanding'].keys()

companyFacts.json()['facts']['dei'][

'EntityCommonStockSharesOutstanding']['units']

companyFacts.json()['facts']['dei'][

'EntityCommonStockSharesOutstanding']['units']['shares']

companyFacts.json()['facts']['dei'][

'EntityCommonStockSharesOutstanding']['units']['shares'][0]

# concept data // financial statement line items

companyFacts.json()['facts']['us-gaap']

companyFacts.json()['facts']['us-gaap'].keys()

# different amounts of data available per concept

companyFacts.json()['facts']['us-gaap']['AccountsPayable']

companyFacts.json()['facts']['us-gaap']['Revenues']

companyFacts.json()['facts']['us-gaap']['Assets']

# get company concept data

companyConcept = requests.get(

(

f'https://data.sec.gov/api/xbrl/companyconcept/CIK{cik}'

f'/us-gaap/Assets.json'

),

headers=headers

)

# review data

companyConcept.json().keys()

companyConcept.json()['units']

companyConcept.json()['units'].keys()

companyConcept.json()['units']['USD']

companyConcept.json()['units']['USD'][0]

# parse assets from single filing

companyConcept.json()['units']['USD'][0]['val']

# get all filings data

assetsData = pd.DataFrame.from\_dict((

companyConcept.json()['units']['USD']))

# review data

assetsData.columns

assetsData.form

# get assets from 10Q forms and reset index

assets10Q = assetsData[assetsData.form == '10-Q']

assets10Q = assets10Q.reset\_index(drop=True)

# plot

assets10Q.plot(x='end', y='val')

# end example

#open xbrl

"""

Convert JSON files from Edgar into CSV.

Using the companyfacts.zip (all XBRL accounting data in SEC filings),

parse to find the accounting terms we want.

"""

import os

import json

import datetime

class AccountingParser() :

\_MAPPED\_SUFFIX = "\_map\_from"

def \_\_init\_\_(self, workingdir : str ) :

self.\_workingdir = workingdir

# import field mappings, assume in JSON file in same dir as code

jsonfile = os.path.join( os.path.dirname(\_\_file\_\_), 'concept\_handling.json')

with open(jsonfile, "r") as f:

self.\_concept\_handling = json.load(f)

self.\_output\_params = list()

for m in self.\_concept\_handling.keys():

self.\_output\_params.append( m )

self.\_output\_params.append( f"{m}{AccountingParser.\_MAPPED\_SUFFIX}")

self.\_filing\_id\_params = [ 'FY\_year'

, 'FY\_quarter'

, 'form'

, 'FY\_date'

, 'CY\_date'

, 'CY\_filing\_date'

]

def parse\_file( self, filename : str ) -> dict :

"""

Parse Edgar JSON file w/ accounting data

Parameters:

filename : the JSON file to read

Returns :

dict with CIK and other corp parameters

and SEC filings in ['filings']

"""

filename = os.path.join( self.\_workingdir, filename )

with open(filename, "r") as f:

data = json.load(f)

entries = data['facts'].get('us-gaap')

if( entries == None ) :

entries = data['facts'].get('ifrs-full')

if( entries == None ) :

print( '[ERROR] CIK: {cik} does not have US-GAAP or IFRS-FULL data.')

return None

# STEP 1: Transform JSON to dict w/ filings

filings = dict()

cik = data['cik']

entityName = data['entityName']

# Add from facts.dei

entry = data['facts'].get('dei')

if( entry != None ) :

entry = data['facts']['dei'].get('EntityPublicFloat')

if( entry != None ) :

entries['EntityPublicFloat'] = entry

for accounting\_parameter in entries:

entry = entries.get(accounting\_parameter)

# TBD: Check for non-USD currency

# Also, some units are 'shares'; we ignore those

if( entry['units'].get('USD') == None ) :

continue

units = entry['units']['USD']

for item in units :

form = item['form']

# restrict to 10-X forms --> 10-Q, 10-K, 10-K/A

if( form[0:2] != '10' ) :

continue

fy\_year = item['fy']

# Remove bad data (there are some year=0 and other nonsense)

if( (fy\_year == None) or (fy\_year == '') ) :

continue

else :

fy\_year = int(fy\_year)

if( (fy\_year < 2009) or (fy\_year > 2025) ) :

continue

fy\_quarter = item['fp']

if( (fy\_quarter == 'FY') or (fy\_quarter == 'CY')

or (fy\_quarter == None) or (fy\_quarter == '' ) ) :

fy\_quarter = 0

else :

fy\_quarter = int(fy\_quarter[1:2])

# Assume all entries have same filing date

filing\_date = item['filed']

# Calendar Year framing may not exist, if so impute

cy\_frame = item.get('frame')

if( (cy\_frame == None) or (cy\_frame == '') ) :

# impute from 'end' (end of period)

# back up date a bit (just in case reporting date goes a little past end-of-quarter)

d = datetime.datetime.strptime(item['end'], '%Y-%m-%d') - datetime.timedelta(days=8)

cy\_frame = f"{d.year}Q{(d.month - 1) // 3 + 1}"

else : # format is CYyyyyQqI

cy\_frame = cy\_frame[2:8]

if( (len(cy\_frame)==5) or (len(cy\_frame)==4) ) :

# frame is missing Q; try to fix

cy\_year = int(cy\_frame[0:5])

d = datetime.datetime.strptime(item['end'], '%Y-%m-%d') - datetime.timedelta(days=8)

if( cy\_year == d.year ):

cy\_frame = f"{d.year}Q{(d.month - 1) // 3 + 1}"

else :

if( item.get('start') != None ) :

d = datetime.datetime.strptime(item['start'], '%Y-%m-%d')

if( cy\_year == d.year ) :

cy\_frame = f"{d.year}Q4"

if( len(cy\_frame) < 6 ) :

print (f"[Warning] CIK:{cik} Form:{form} Year:{fy\_year} Quarter:{fy\_quarter} Tag:{accounting\_parameter} has CY\_frame mismatch to end-of-period.")

index = (fy\_year,fy\_quarter,form)

value = item['val']

filing = filings.get(index)

if( filing == None ) :

# Create new filing record

new\_q = fy\_quarter

if( new\_q == 0 ) :

new\_q = 4 # Put yearly results in Q4

filings[index] = {

'FY\_year' : fy\_year

, 'FY\_quarter' : fy\_quarter

, 'form' : form

, 'FY\_date' : f"{fy\_year}Q{new\_q}"

, 'CY\_date' : cy\_frame

, 'CY\_filing\_date' : filing\_date

}

filing = filings[index]

# Add the parameter to the filing record

filing[accounting\_parameter] = value

# end for

# STEP 2:

# Go through form by form and calculate the accounting concepts we want

VALUE\_NaN = "NaN"

VALUE\_NotAvailable = "NA"

# Mapping between accounting concepts and fields

# Code will try to go down the list for each concept until it finds one

concept\_handling = self.\_concept\_handling

for ndx in filings.keys() :

row = filings[ndx]

for entry\_name in concept\_handling.keys() :

row[entry\_name] = None

for field\_name in concept\_handling[entry\_name]['map'] :

if( row.get(field\_name) != None ) :

row[entry\_name] = row[field\_name]

row[f"{entry\_name}{AccountingParser.\_MAPPED\_SUFFIX}"] = field\_name

break

if( row[entry\_name] == None ) :

row[entry\_name] = concept\_handling[entry\_name]['on\_missing']

row[f"{entry\_name}{AccountingParser.\_MAPPED\_SUFFIX}"] = "Not\_Found"

if( row[entry\_name] == VALUE\_NaN ) :

print (f"[Warning] CIK:{cik} Form:{row['form']} Year:{row['FY\_year']} Quarter:{row['FY\_quarter']} is missing {entry\_name}")

# Public float

entry\_name = '\_EntityPublicFloat'

if( row.get('EntityPublicFloat') != None ) :

row[entry\_name] = row['EntityPublicFloat']

elif( row['form'].startswith("10-Q") ) :

# 10-Q's don't have this value

row[entry\_name] = VALUE\_NotAvailable

else :

row[entry\_name] = VALUE\_NaN

print (f"[Warning] CIK:{cik} Form:{row['form']} Year:{row['FY\_year']} is missing EntityPublicFloat")

# end loop on accounting concepts

# STEP 3: Impute certain quarterly data

for ndx in filings.keys() :

row = filings[ndx]

# We modify the 10-K rows (these are the FY 4Q)

if( not row['form'].startswith('10-K') ) :

continue

fy\_year = row[ 'FY\_year' ]

for series\_name in [ '\_Profits' ] :

year\_value = row[ series\_name ]

if( year\_value == VALUE\_NaN ) :

continue

# Keep old values as \_annual

row[ series\_name + '\_annual' ] = row[series\_name]

# impute quarterly value on this FY 10-K

# by subtracting the FY 10-Q values from the year's

for q in range(1,4) :

q\_row = filings.get((fy\_year, q, '10-Q'))

if( q\_row == None ) :

# try another form

q\_row = filings.get((fy\_year, q, '10-Q/A'))

if( (q\_row == None) or ( q\_row[series\_name] == VALUE\_NaN ) ) :

print( f"[Warning] Missing 10-Q form for CIK:{cik} FY\_year:{fy\_year} for quarter: {q}. Skipping year's {series\_name}")

year\_value = VALUE\_NaN

break

else :

year\_value -= q\_row[series\_name]

# Overwrite w/ imputed value

row[series\_name] = year\_value

# end loop on imputation

all = dict()

all['CIK'] = cik

all['entityName'] = entityName

all['filings'] = filings

return all

# end parse\_file

def \_csv\_header(self) -> str :

"""

CSV header fields

"""

column\_names = self.\_filing\_id\_params + self.\_output\_params

s = '"CIK","entityName"'

for param in column\_names:

s += f',"{param}"'

return s

def \_csv\_filing(self, cik : int, entityName : str, filing\_fields : dict ) -> str :

"""

Takes output of parse\_file() and returns CSV row

"""

cols = self.\_filing\_id\_params + self.\_output\_params

s = f'{cik},"{entityName}"'

for c in cols :

param = filing\_fields[c]

if( type(param) == str ) :

s += f',"{param}"'

else :

s += f',{param}'

return s

def parse\_files\_to\_csv( self, ciks : list, outputfile : str ) -> None :

"""

Read files of CIKs in list, call .parse\_file() on each one, and produce CSV

Parameters:

ciks : list of CIKS to read. Filename in format 'CIK{cik:010}.json'

outputfile : path (folder and filename) to write CSV to

"""

progress\_update\_at = max(1, int(len(ciks)/50))

cik\_progress = 0

column\_names = self.\_filing\_id\_params + self.\_output\_params

with open(outputfile, "w") as f:

print(self.\_csv\_header(), file=f) #header

num\_rows = 0

for cik in ciks :

x = self.parse\_file( f"CIK{cik:010}.json" )

for ndx in x['filings'].keys() :

filing\_fields = x['filings'][ndx]

s = self.\_csv\_filing( cik, x['entityName'], filing\_fields)

print(s, file=f)

num\_rows += 1

cik\_progress += 1

# Show progress

if( cik\_progress % progress\_update\_at == 0 ) :

print( f"Parsed {cik\_progress} out of {len(ciks)}")

print( f"Finished parsing. Total filings: {num\_rows} ")

def get\_CIK\_list( self, min\_size : int = 0 ) -> list :

"""

Read <working\_dir>, find all CIK files, extract CIKs for big-enough files

Parameters:

min\_size : minimum size (in KB) of files

"""

min\_size = min\_size \* 1024

# Get list of all files only in the given directory

func = lambda x : os.path.isfile(os.path.join(self.\_workingdir, x))

files\_list = filter(func, os.listdir(self.\_workingdir))

cik\_list = list()

for f in files\_list :

sz = os.stat(os.path.join(self.\_workingdir, f)).st\_size

if( sz > min\_size ) :

cik = int(f[3:13])

cik\_list.append(cik)

return cik\_list

# end AccountingParser

"""

Downloader of SEC Edgar data

"""

import os

import requests

import json

import pandas

import time

class Downloader( ) :

def get\_companyfacts( self, cik : int, workingdir : str ) -> str :

"""

Downloads a single companyfacts JSON file to the working dir

Parameters:

cik : CIK id for company

workingdir : path to folder to which do download the file

Returns:

name of JSON file downloaded

"""

filename = f"CIK{cik:010}.json"

url = f"https://data.sec.gov/api/xbrl/companyfacts/{filename}"

content = self.\_load\_url(url)

if( not os.path.exists(workingdir) ) :

os.makedirs(workingdir, exist\_ok=True)

outfile = os.path.join(workingdir, filename)

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

f.write(content)

return filename

def get\_urls( self, ciks, forms, from\_date, to\_date ) -> pandas.DataFrame :

"""

Returns DataFrame with URLs to Edgar data based on query criteria

Parameters:

ciks : list of CIK numbers of firms

forms : list of SEC forms

from\_date : start of period in 'yyyy-mm-dd' format

to\_date : end of period in 'yyyy-mm-dd' format

Returns :

pandas.DataFrame with columns CIK, filing\_date, form, URL

"""

sec\_data = dict()

for cik in ciks :

cik = int(cik)

url = f"https://data.sec.gov/submissions/CIK{cik:010}.json"

data = json.loads(self.\_load\_url(url))

# Convert JSON to dataframe for easier use

recents = pandas.DataFrame(data['filings']['recent'])

recents['reportDate'] = pandas.to\_datetime(recents['reportDate'])

recents['filingDate'] = pandas.to\_datetime(recents['filingDate'])

for form in forms :

df = recents[(recents['form'] == form) &

(recents['filingDate'] >= from\_date) &

(recents['filingDate'] <= to\_date ) ]

for i in range(len(df)) :

row = df.iloc[i]

accessionNumber = row['accessionNumber'].replace("-", "")

filename = f"{row['accessionNumber']}.txt"

url = f"https://www.sec.gov/Archives/edgar/data/{cik:010}/{accessionNumber}/{filename}"

sec\_data[ (cik, form, row['filingDate']) ] = { 'URL': url, 'filename' : filename }

# Convert dict to DataFrame

df = pandas.DataFrame.from\_dict( sec\_data, orient='index' )

xf = pandas.DataFrame.from\_records( df.index, columns=['CIK', 'form', 'filing\_date'])

df.reset\_index(inplace=True)

df = df.join(xf)

return df[['CIK', 'form', 'filing\_date', 'URL', 'filename']]

# end get\_urls()

def get\_files( self, ciks, forms, from\_date, to\_date, workingdir : str ) -> None :

"""

Downloads and saves Edgar data to working dir.

Parameters:

ciks : list of CIK numbers of firms

forms : list of SEC forms to download

from\_date : start of period to download in 'yyyy-mm-dd' format

to\_date : end of period to download in 'yyyy-mm-dd' format

workingdir : folder to download files to

"""

os.makedirs( workingdir, exist\_ok=True)

num\_files = 0

df = self.get\_urls( ciks, forms, from\_date, to\_date )

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

row = df.iloc[ndx]

filename = row['filename']

url = row['URL']

cik = row['CIK']

form = row['form']

content = self.\_load\_url(url)

with open(os.path.join(workingdir, filename), "wb") as f:

f.write(content)

num\_files += 1

print(f"Downloaded CIK:{cik} Form:{form} to {filename}")

print( f"Download request finished. {num\_files} files fetched.")

# end get\_files()

def \_load\_url(self, url : str) :

"""

Helper function to download URLs from EDGAR

"""

# Wait a bit as per SEC Edgar rate use requirements

time.sleep(0.11)

response = requests.get(url,headers={"User-Agent": "Mozilla/5.0"})

if response.status\_code != 200:

raise Exception(f"Failed to fetch data from URL: {url}")

return response.content

# end \_load\_url()

# end Downloader()

{

"\_Liabilities" :

{

"map" : ["Liabilities", "LiabilitiesAndStockholdersEquity", "LiabilitiesCurrent", "LiabilitiesNoncurrent", "OtherLiabilities", "OtherLiabilitiesCurrent", "OtherLiabilitiesNoncurrent"]

, "on\_missing" : "NaN"

}

, "\_Assets" :

{

"map" : ["Assets", "AssetsCurrent", "AssetsNoncurrent", "OtherAssetsCurrent", "OtherAssetsNoncurrent", "OtherAssets", "IntangibleAssetsGrossExcludingGoodwill", "IntangibleAssetsNetExcludingGoodwill"]

, "on\_missing" : "NaN"

}

, "\_Profits" :

{

"map" : ["GrossProfit", "ProfitLoss", "OperatingIncomeLoss", "NetIncomeLoss", "OperatingIncomeLoss", "ComprehensiveIncomeNetOfTax", "NetIncomeLossIncludingPortionAttributableToNonredeemableNoncontrollingInterest"]

, "on\_missing" : "NaN"

}

, "\_Revenues" :

{

"map" : ["RevenueFromContractWithCustomerExcludingAssessedTax", "Revenues", "SalesRevenueNet", "SalesRevenueServicesGross", "SalesRevenueGoodsNet", "SalesRevenueServicesNet", "OtherSalesRevenueNet", "RevenueFromContractWithCustomerIncludingAssessedTax"]

, "on\_missing" : "NaN"

}

, "\_Costs" :

{

"map" : ["CostOfGoodsSold", "CostOfRevenue", "CostsAndExpenses", "CostOfGoodsAndServicesSold", "CostOfServices"]

, "on\_missing" : "NaN"

}

, "\_Dividends" :

{

"map" : ["DividendsCommonStock", "DividendsCommonStockCash", "Dividends", "DividendsCash", "CashDividends", "PaymentsOfDividends", "PaymentsOfDividendsCommonStock"]

, "on\_missing" : 0

}

}

"""

Main access point for OpenXBRL api

"""

from openxbrl import AccountingParser, Downloader

import os

class OpenXBRL() :

def \_\_init\_\_(self, base\_dir : str ) :

"""

Initialization requires pointing to a base directory.

This folder will hold filings data.

"""

self.\_base\_dir = base\_dir

def get\_filing\_accounting( self, cik : int, fiscal\_year : int, fiscal\_quarter : int ) -> dict :

"""

Return dict with accounting fields parsed from a corporate filing.

Parameters:

cik : CIK identifier of the company

fiscal\_year : Fiscal year of the filing

fiscal\_quarter : Fiscal quarter of the filing.

NOTE: Use fiscal\_quarter = 0 to get the annual (10-K) filing

"""

workingdir = os.path.join(self.\_base\_dir, 'companyfacts/')

downloader = Downloader()

parser = AccountingParser( workingdir )

filename = downloader.get\_companyfacts(cik=cik, workingdir=workingdir)

result = parser.parse\_file( filename )

filings = result['filings']

# Look for filing. Rem index=(fy\_year,fy\_quarter,form)

if( fiscal\_quarter == 0 ) :

form = '10-K'

else :

form = '10-Q'

filing = filings.get((fiscal\_year, fiscal\_quarter, form))

if( filing == None ) :

form += '/A' # look for amended

filing = filings.get((fiscal\_year, fiscal\_quarter, form))

if( filing != None ) :

filing['CIK'] = cik

filing['entityName'] = result['entityName']

return filing

# end open xbrl project

#delta api edgar scrapper

from datetime import datetime, timedelta

import pandas as pd

import json

from .connector import APIconnector

def fillTo10D(cell):

while len(cell) != 10:

cell = '0' + cell

return cell

def generate\_CIK\_TICKER(filename = 'ticker-SEC.csv'):

jsonSEClist = APIconnector('https://www.sec.gov/files/company\_tickers.json').get\_request()

recentFilings = pd.DataFrame.from\_dict(jsonSEClist.json()).T

recentFilings['cik10D'] = recentFilings['cik\_str'].astype(str).apply(lambda x: fillTo10D(x))

#recentFilings.to\_csv(filename, index=False)

return recentFilings

def rename\_cols\_in\_df(df):

renameDic = {}

for column in df.columns:

newName = column.strip().replace(' ', '\_').replace('-', '\_')

renameDic[column] = newName

df.rename(columns=renameDic, inplace=True)

return df

def monthWindow(df):

daysDiff = df['diffDate']

months = 0

while daysDiff >= 30:

daysDiff -=30

months +=1

return months

def unpactUnitsJson(index,jsonDataframe):

key = list(jsonDataframe['units'].iloc[index].keys())[0]

valuesInTable = pd.DataFrame(jsonDataframe['units'].iloc[index][key])

if 'start' in valuesInTable.columns:

valuesInTable['startFormat'] = pd.to\_datetime(valuesInTable['start'])

if 'end' in valuesInTable.columns:

valuesInTable['endFormat'] = pd.to\_datetime(valuesInTable['end'])

if 'end' in valuesInTable.columns and 'start' in valuesInTable.columns:

valuesInTable['diffDate'] = (valuesInTable['endFormat'] - valuesInTable['startFormat']).dt.days

valuesInTable['monthWindow'] = valuesInTable.apply(monthWindow, axis=1)

#valuesInTable['time'] = valuesInTable['endFormat'].dt.year.astype(str).str[2:] + valuesInTable['endFormat'].dt.month.map("{:02}".format)

return valuesInTable

def unpackSECjson(cik):

scr = APIconnector(f'https://data.sec.gov/api/xbrl/companyfacts/{cik}.json')

jsonRequest = scr.get\_request().json()

r = jsonRequest['facts']['us-gaap']

r = json.dumps(r)

jsonDataframe = pd.read\_json(r).T

jsonDataframe.reset\_index(inplace=True, names='finType')

valuesDF = pd.DataFrame()

for index, row in jsonDataframe.iterrows():

unit = unpactUnitsJson(index, jsonDataframe)

unit['finType'] = jsonDataframe.iloc[index]['finType']

valuesDF = pd.concat([unit,valuesDF])

mergedDF = jsonDataframe.merge(valuesDF, on='finType')

mergedDF = mergedDF[['finType', 'val', 'accn', 'fy', #'label', 'description'

'fp', 'form', 'filed', 'frame', 'endFormat', 'time', 'startFormat', 'monthWindow']]

return mergedDF

def get\_spy500\_formWiki():

table = pd.read\_html('https://en.wikipedia.org/wiki/List\_of\_S%26P\_500\_companies')[0]

table['CIK'] = table['CIK'].astype(str).apply(lambda x: fillTo10D(x))

table = rename\_cols\_in\_df(table)

return table

def get\_rangeOfDates(yearOffset):

now = datetime.now()

dateto = int(now.timestamp())

dt = timedelta(days=366\*yearOffset)

datefrom = int((now - dt).timestamp())

return dateto, datefrom

def get\_StockPrices(ticker, startSelect = None, endSelect = None, interval = '1d'):

if startSelect:

end = int(datetime.strptime(endSelect,"%Y-%m-%d").timestamp())

start = int(datetime.strptime(startSelect,"%Y-%m-%d").timestamp())

else:

end, start = get\_rangeOfDates(24)

stockPrice = APIconnector(f'https://query2.finance.yahoo.com/v8/finance/chart/{ticker}?period1={start}&period2={end}&interval={interval}')

adjClose = stockPrice.get\_request().json()['chart']['result'][0]['indicators']['adjclose'][0]['adjclose']

timestap = stockPrice.get\_request().json()['chart']['result'][0]['timestamp']

priceDF = pd.DataFrame({'adjClose':adjClose, 'time':timestap})

priceDF['date'] = pd.to\_datetime(priceDF['time'].apply(lambda x: datetime.fromtimestamp(x))).dt.strftime('%Y-%m-%d')

priceDF['ticker'] = ticker

priceDF.drop('time', inplace=True, axis=1)

priceDF["Quarter"] = "Q" + pd.to\_datetime(priceDF['date']).dt.quarter.astype(str) + "-" \

+ pd.to\_datetime(priceDF['date']).dt.year.astype(str)

groupByQ = priceDF[["Quarter", "adjClose"]].groupby("Quarter").mean().reset\_index().rename(columns={'adjClose': 'quarter\_meanADJclose'})

mergeMean = pd.merge(priceDF, groupByQ, on = 'Quarter')

mergeMean["Month"] = "Month" + pd.to\_datetime(mergeMean['date']).dt.month.astype(str) + "-" \

+ pd.to\_datetime(mergeMean['date']).dt.year.astype(str)

groupByQ = mergeMean[["Month", "adjClose"]].groupby("Month").mean().reset\_index().rename(columns={'adjClose': 'month\_meanADJclose'})

mergeMean = pd.merge(mergeMean, groupByQ, on = 'Month')

mergeMean = mergeMean.apply(lambda x: addDateKey(x, 'date', 'date'), axis=1)

mergeMean.drop(['Quarter', 'date', 'Month'], axis=1, inplace=True)

return mergeMean

def cleaned\_companyfacts(jsonDataframe):

valuesDF = pd.DataFrame()

for index, row in jsonDataframe.iterrows():

unit = unpactUnitsJson(index, jsonDataframe)

unit['finType'] = jsonDataframe.iloc[index]['finType']

valuesDF = pd.concat([unit,valuesDF])

mergedDF = jsonDataframe.merge(valuesDF, on='finType')

mergedDF.drop(['units'], axis = 1, inplace=True)

#mergedDF.fillna("null", inplace = True)

return mergedDF

def get\_companyfacts(cik, ticker):

if "CIK" not in cik:

cik = "CIK" + cik

baseUrl = f'https://data.sec.gov/api/xbrl/companyfacts/{cik}.json'

scr = APIconnector(baseUrl)

jsonRequest = scr.get\_request().json()

r = jsonRequest['facts']['us-gaap']

r = json.dumps(r)

jsonDataframe = pd.read\_json(r).T

jsonDataframe.reset\_index(inplace=True, names='finType')

mergedDF = cleaned\_companyfacts(jsonDataframe)

mergedDF = mergedDF.apply(lambda x: addDateKey(x, 'end', 'start', ticker), axis=1)

mergedDF['ticker'] = ticker

desired\_order = [

'finType', 'label', 'description', 'end', 'val', 'accn', 'fy', 'fp',

'form', 'filed', 'frame', 'endFormat', 'start', 'startFormat',

'diffDate', 'monthWindow', 'yearMonthDay', 'ticker']

mergedDF = mergedDF[desired\_order]

mergedDF.drop(["fy", "endFormat", "startFormat", "diffDate", 'frame'], inplace=True,axis=1)

return mergedDF

def get\_CIK\_by\_Ticker(ticker, filename = 'ticker-SEC.csv', fill0 = True):

recentFilings = pd.read\_csv(filename)

selectedTicker = recentFilings[recentFilings['ticker'] == ticker]

if not fill0:

return str(selectedTicker.cik\_str.values[0])

return fillTo10D(str(selectedTicker.cik\_str.values[0]))

def addDateKey(row, col1, col2, ticker = None):

pd.options.mode.copy\_on\_write = True

try:

row['yearMonthDay'] = pd.to\_datetime(row[col1]).strftime("%Y%m%d")

except ValueError:

row['yearMonthDay'] = pd.to\_datetime(row[col2]).strftime("%Y%m%d")

except:

row['yearMonthDay'] = ''

if ticker:

row['ticker'] = ticker

return row

def get\_SEC\_filings(cik, ticker):

pd.options.mode.copy\_on\_write = True

if "CIK" not in cik:

cik = "CIK" + cik

clearCik = cik.replace('CIK', '')

reqURL = F'https://data.sec.gov/submissions/CIK{clearCik}.json'

scr = APIconnector(reqURL)

scr.URL = reqURL

res = scr.get\_request()

JSONresponse = res.json()

#https://data.sec.gov/api/xbrl/companyfacts/CIK0000320193.json

filings = pd.DataFrame.from\_dict(JSONresponse['filings']['recent'])

filings['accessionNumberCLEAN'] = filings['accessionNumber'].apply(lambda x: x.replace('-', ''))

filings['fileURL'] = 'https://www.sec.gov/Archives/edgar/data/' + clearCik + "/" + \

filings['accessionNumberCLEAN'] + "/"+ filings['primaryDocument']

filings.drop('accessionNumberCLEAN', inplace = True, axis = 1)

filings = filings.apply(lambda x: addDateKey(x, 'reportDate', 'filingDate', ticker), axis=1)

#filings['ticker'] = ticker

desired\_order = [

'accessionNumber', 'filingDate', 'reportDate', 'acceptanceDateTime',

'act', 'form', 'fileNumber', 'filmNumber', 'items', 'size',

'isXBRL', 'isInlineXBRL', 'primaryDocument', 'primaryDocDescription',

'fileURL', 'yearMonthDay', 'ticker']

filings = filings[desired\_order]

return filings

if \_\_name\_\_=="main":

pass

# end delta api edgar scrapper

#quants

import pandas as pd

import requests

headers = {"User-Agent": "ian.ye.fu@gmail.com"}

data\_folder\_download = './datasets/download/'

data\_folder\_generate = './datasets/generate/'

def cik\_matching\_ticker(ticker, headers=headers):

ticker = ticker.upper().replace(".", "-")

ticker\_json = requests.get(

"https://www.sec.gov/files/company\_tickers.json", headers=headers

).json()

for company in ticker\_json.values():

if company["ticker"] == ticker:

cik = str(company["cik\_str"]).zfill(10)

return cik

raise ValueError(f"Ticker {ticker} not found in SEC database")

def get\_fiscal\_YE(sp500\_cik\_list):

"""

Get\_fiscal\_year\_end for S&P companies by passing the cik list.

"""

fiscal\_YE = []

for cik in sp500\_cik\_list:

url = f"https://data.sec.gov/submissions/CIK{cik}.json"

try:

# Make the API request

company\_json = requests.get(url, headers=headers).json()

# Check if 'fiscalYearEnd' exists in the JSON response

fiscal\_year\_end = company\_json.get('fiscalYearEnd', 'N/A')

# Append the result

fiscal\_YE.append(fiscal\_year\_end)

except Exception as e:

# In case of error, append 'N/A' or handle accordingly

print(f"Error fetching data for CIK {cik}: {e}")

fiscal\_YE.append('N/A')

return fiscal\_YE

def get\_facts(ticker, sp500\_df, headers):

"""

Load company\_facts data from SEC API

"""

try:

cik = f'CIK{sp500\_df.loc[ticker, "CIK"]}'

url = f"https://data.sec.gov/api/xbrl/companyfacts/{cik}.json"

response = requests.get(url, headers=headers)

# Check if the response status code is 200 (success)

if response.status\_code == 200:

company\_facts = response.json()

return company\_facts

else:

print(f"Error: {response.status\_code} for ticker {ticker}. Skipping.")

return None

except requests.exceptions.RequestException as e:

print(f"An error occurred: {e}. Skipping ticker {ticker}.")

return None

def facts\_DF(ticker, sp500\_df, headers=headers):

"""

Convert the company\_facts dict to dataframes

"""

facts = get\_facts(ticker, sp500\_df, headers) # get company facts data

# Check if facts is None before processing

if facts is None:

print(f"No data available for {ticker}. Skipping...")

return pd.DataFrame()

# Process us\_gaap data

us\_gaap\_data = facts.get("facts", {}).get("us-gaap", {})

df\_data = []

for fact, details in us\_gaap\_data.items():

for unit in details.get("units", {}).keys():

for item in details["units"][unit]:

row = item.copy() # keep the original data intact

row["fact"] = fact

row["label"] = details["label"]

df\_data.append(row)

# Create a DataFrame from the collected data

df = pd.DataFrame(df\_data)

df["end"] = pd.to\_datetime(df["end"])

df["start"] = pd.to\_datetime(df["start"])

df = df.drop\_duplicates(subset=["fact", "end", "val"])

df.set\_index("end", inplace=True)

labels\_dict = {fact: details["label"] for fact, details in us\_gaap\_data.items()}

return df

def download\_financial\_data\_from\_SEC(sp500\_tickers, sp500\_df, data\_category, headers = headers):

"""

download all the financial metrics in the data\_category from SEC for the sp500 companies

"""

sp500\_financial\_data = {} # outer dict

for ticker in sp500\_tickers:

df = facts\_DF(ticker, sp500\_df, headers) # Correct data extraction for the ticker

financial\_data= {} # inner dict

# Loop over all categories in the data\_category list

for category in data\_category:

x = df.query('fact == @category')

x = x[(x['val']!= 0) & (x['val'].notna())]

# remove the duplicated rows based on 'end' index and keep the last record

cleaned\_data = x[~x.index.duplicated(keep='last')].sort\_index(ascending = True)

# slice data from only 2013 onwards

financial\_data[category] = cleaned\_data.loc['2013':]

# Assign the financial data for each ticker

sp500\_financial\_data[ticker] = financial\_data

return sp500\_financial\_data

def convert\_annual\_to\_quarter(sp500\_financial\_data, updated\_data\_category):

"""

This function updates financial data by subtracting the annual rows by

the sum of the previous three quarters for a given list of tickers and data categories

"""

for ticker in sp500\_financial\_data.keys():

for category in sp500\_financial\_data[ticker].keys():

new\_df = sp500\_financial\_data[ticker][category].reset\_index().copy()

if category == "EarningsPerShareDiluted" or category == 'CommonStockDividendsPerShareDeclared' or category == 'NetIncomeLoss':

# identify the annual rows

index\_list = new\_df[(new\_df['end'] - new\_df['start']).dt.days > 130].index.tolist()

# subtract the annual rows by the sum of the previous three quarters.

for i in index\_list:

new\_df.loc[i,'val'] = new\_df.loc[i,'val'] - new\_df.loc[i-3: i-1, 'val'].sum()

sp500\_financial\_data[ticker][category] = new\_df.set\_index('end')

return sp500\_financial\_data

def weighting(metric\_ranking\_df, monthly\_availability\_df):

"""

Calculate the weighting among selected stocks to construct the portfolio for each month

Select the top 20% and bottom 20% stocks based on ranking and assign equal weights between the stocks.

"""

# filter the PB ranking df based on the monthly availability of the stocks.

filtered\_metric\_ranking\_df = metric\_ranking\_df\*monthly\_availability\_df

# Calculating the top and bottom 20% of the companies for each month:

top\_20\_threshold = filtered\_metric\_ranking\_df.quantile(0.8, axis=1)

bottom\_20\_threshold = filtered\_metric\_ranking\_df.quantile(0.2, axis=1)

# Create masks for top 20% and bottom 20%

top\_20\_mask = filtered\_metric\_ranking\_df.ge(top\_20\_threshold, axis=0)

bottom\_20\_mask = filtered\_metric\_ranking\_df.le(bottom\_20\_threshold, axis=0)

# Filter dataframes for top 20% and bottom 20%

top\_20\_df = filtered\_metric\_ranking\_df[top\_20\_mask]

bottom\_20\_df = filtered\_metric\_ranking\_df[bottom\_20\_mask]

# Calculate equal weight for each stock in top 20% and bottom 20%

top\_20\_weights = 1 / top\_20\_df.count(axis=1)

bottom\_20\_weights = 1 / bottom\_20\_df.count(axis=1)

# Apply the weights across the dataframe.

top\_20\_df\_weights = top\_20\_df.map(lambda x: 1 if pd.notna(x) else np.NaN).multiply(top\_20\_weights, axis=0)

bottom\_20\_df\_weights = bottom\_20\_df.map(lambda x: 1 if pd.notna(x) else np.NaN).multiply(bottom\_20\_weights, axis=0)

return top\_20\_df\_weights, bottom\_20\_df\_weights

def cal\_cum\_rets(top\_20\_df\_weights, bottom\_20\_df\_weights, monthly\_returns\_df):

value\_20130331 = 100

# Multiply the monthly returns by the equal weights and aggregate monthly returns.

top\_20\_weighted\_monthly\_returns = ((top\_20\_df\_weights.shift(1) \* monthly\_returns\_df).sum(axis = 1)).iloc[1:]

bottom\_20\_weighted\_monthly\_returns = ((bottom\_20\_df\_weights.shift(1) \* monthly\_returns\_df).sum(axis = 1)).iloc[1:]

long\_short\_monthly\_returns = bottom\_20\_weighted\_monthly\_returns - top\_20\_weighted\_monthly\_returns

top\_20\_weighted\_cum\_returns = value\_20130331\*(1 + top\_20\_weighted\_monthly\_returns).cumprod() - 1

bottom\_20\_weighted\_cum\_returns = value\_20130331\*(1 + bottom\_20\_weighted\_monthly\_returns).cumprod() - 1

long\_short\_cum\_returns = value\_20130331\*(1 + long\_short\_monthly\_returns).cumprod() - 1

return top\_20\_weighted\_cum\_returns, bottom\_20\_weighted\_cum\_returns, long\_short\_cum\_returns

def cal\_index\_cum\_rets(index\_name, data\_folder\_download):

value\_20130331 = 100

index\_price = pd.read\_csv(data\_folder\_download + index\_name + '.csv', index\_col = "Date", parse\_dates = True)

index\_price = index\_price.resample('ME').last()

index\_monthly\_returns = index\_price / index\_price.shift(1) - 1

index\_monthly\_returns = index\_monthly\_returns.iloc[3: ]

index\_cum\_returns = value\_20130331\*(1 + index\_monthly\_returns).cumprod() - 1

return index\_cum\_returns

def calc\_rolling\_12mon\_return\_n\_vol(monthly\_return):

"""

calculate the rolling 12month returns and volatilities of a given monthly\_return series

"""

rolling\_12mon\_return = (1 + monthly\_return).rolling(window = 12).apply(lambda x: x.prod()) - 1

rolling\_12mon\_std = rolling\_12mon\_return.rolling(window = 12).std()\*np.sqrt(12)

return rolling\_12mon\_return, rolling\_12mon\_std

def calc\_index\_monthly\_returns(ticker, folder):

index\_price = pd.read\_csv(folder + ticker + '.csv', index\_col = "Date", parse\_dates = True)

index\_price = index\_price.resample('ME').last()

index\_monthly\_returns = index\_price / index\_price.shift(1) - 1

return index\_monthly\_returns

def get\_10Y\_bond\_yields(folder, file, syear, smonth, sday, eyear, emonth, eday):

"""

Get the US 10Y government bond yield

"""

# read the file into a dataframe

df = pd.read\_csv(data\_folder\_download + file + '.csv', index\_col = 'DATE', parse\_dates = True).rename(columns = {'DGS10': '10YR'})

df['10YR'] = pd.to\_numeric(df['10YR'], errors = 'coerce')

df['10YR'] = df['10YR']/100

# convert datetime

start = f"{syear}-{smonth}-{sday}"

start = pd.to\_datetime(start)

end = f"{eyear}-{emonth}-{eday}"

end = pd.to\_datetime(end)

# select the timespan

df = df.loc[start:end]

return df

def sharpe\_ratio(r, riskfree\_rate, periods\_per\_year):

"""

Compute the annualized sharpe ratio of a set of returns

"""

length = r.shape[0]

riskfree\_rate = pd.Series(riskfree\_rate, index = r.index)

rf\_per\_period = (1+riskfree\_rate)\*\*(1/periods\_per\_year)-1

rf\_per\_period = rf\_per\_period.iloc[3:]

excess\_ret = r - rf\_per\_period

annualized\_ret = (excess\_ret + 1).prod() \*\* (periods\_per\_year/length) - 1

annualized\_vol = excess\_ret.std()\*(periods\_per\_year\*\*0.5)

return annualized\_ret/annualized\_vol

def drawdown(return\_series: pd.Series):

"""

Takes a time series of asset returns.

returns a DataFrame with columns for

the wealth index,

the previous peaks, and

the percentage drawdown

"""

wealth\_index = 1000\*(1+return\_series).cumprod()

previous\_peaks = wealth\_index.cummax()

drawdowns = (wealth\_index - previous\_peaks)/previous\_peaks

return pd.DataFrame({"Wealth": wealth\_index,

"Previous Peak": previous\_peaks,

"Drawdown": drawdowns})

def skewness(r):

"""

Alternative to scipy.stats.skew()

Computes the skewness of the supplied Series or DataFrame

Returns a float or a Series

"""

demeaned\_r = r - r.mean()

# use the population standard deviation, so set dof=0

sigma\_r = r.std(ddof=0)

exp = (demeaned\_r\*\*3).mean()

return exp/sigma\_r\*\*3

def kurtosis(r):

"""

Alternative to scipy.stats.kurtosis()

Computes the kurtosis of the supplied Series or DataFrame

Returns a float or a Series

"""

demeaned\_r = r - r.mean()

# use the population standard deviation, so set dof=0

sigma\_r = r.std(ddof=0)

exp = (demeaned\_r\*\*4).mean()

return exp/sigma\_r\*\*4

def var\_historic(r, level):

if isinstance(r, pd.DataFrame):

return r.aggregate(lambda x: abs(np.percentile(x, level)), axis = 0)

elif isinstance(r, pd.Series):

return abs(np.percentile(r, level)) # note here I only want the scaler without index for further comparison or masking.

else:

raise TypeError("Expected r to be Series or DataFrame")

from scipy.stats import norm

def var\_gaussian(r, level=5, modified=False):

"""

Returns the Parametric Gauusian VaR of a Series or DataFrame

If "modified" is True, then the modified VaR is returned,

using the Cornish-Fisher modification

"""

# compute the Z score assuming it was Gaussian

z = norm.ppf(level/100)

if modified:

# modify the Z score based on observed skewness and kurtosis

s = skewness(r)

k = kurtosis(r)

z = (z +

(z\*\*2 - 1)\*s/6 +

(z\*\*3 -3\*z)\*(k-3)/24 -

(2\*z\*\*3 - 5\*z)\*(s\*\*2)/36

)

return -(r.mean() + z\*r.std(ddof=0))

def cvar\_historic(r, level=5):

"""

Computes the Conditional VaR of Series or DataFrame

"""

if isinstance(r, pd.Series):

var\_value = var\_historic(r, level=level)

is\_beyond = r<= -var\_value

return -r[is\_beyond].mean()

elif isinstance(r, pd.DataFrame):

return r.aggregate(cvar\_historic, level = level)

else:

raise TypeError("Expected r to be a Series or DataFrame")

#investor helper

import requests

import json

from dotenv import load\_dotenv

import os

import pandas as pd

from datetime import datetime

from functions.data\_wrangle import historical\_10Q\_dw, historical\_10Q\_merge

def get\_ticker\_cik\_mapping(ticker\_list):

try:

with open('functions/data/cik\_mapping.json', 'r') as f:

cik\_mapping = json.load(f)

f.close()

except:

print("Error in loading cik\_mapping.json, check directory")

return {"ticker": None}

result = {"ticker": "CIK" + cik\_mapping[ticker][0] for ticker in ticker\_list if ticker in cik\_mapping.keys()}

if result == {}:

return {"ticker": None}

return result

def load\_env(api: str = None, cik = None) -> dict:

try:

load\_dotenv('functions/.env')

user\_agent : str = os.getenv('USER\_AGENT')

headers = {

'User-Agent': user\_agent

}

url = f"{api}{cik}.json"

print("Load .env variables -- SUCCESS")

return {"url": url, "headers": headers}

except:

print("Error in loading environment variables, check .env file")

return {"url": None, "headers": None}

def get\_company\_info\_by\_CIK(cik) -> dict:

env\_var = load\_env("https://data.sec.gov/submissions/", cik = cik)

if env\_var["url"] is None or env\_var["headers"] is None:

return {"client\_info": None, "contact\_info": None}

attempt = 0

max\_attempt = 1

while attempt < max\_attempt:

try:

res = requests.get(env\_var["url"], headers=env\_var["headers"]).json()

client\_info = {key: res[key] for key in [

"cik", "name", "sic", "sicDescription", "ownerOrg"

]}

for key in ["tickers", "exchanges"]:

client\_info[key] = ", ".join(list(set(res[key])))

contact\_info = {

"mailing\_address" : res['addresses']['mailing'],

"phone" : res['phone']

}

with open('functions/data/countrycode\_mapping.json', 'r') as f:

countrycode = json.load(f)

f.close()

contact\_info["mailing\_address"]["Country\_Region"] = countrycode[contact\_info["mailing\_address"]["stateOrCountry"]]

return {"client\_info": client\_info, "contact\_info": contact\_info}

except:

attempt += 1

if attempt == max\_attempt:

print("Error in SEC API call, check API validity")

return {"client\_info": None, "contact\_info": None}

def get\_company\_info\_by\_ticker(ticker) -> dict:

cik\_mapping = get\_ticker\_cik\_mapping([ticker])

if cik\_mapping["ticker"] is None:

return {"client\_info": None, "contact\_info": None}

return get\_company\_info\_by\_CIK(cik\_mapping["ticker"])

def get\_report\_by\_CIK(cik, reportType=["10-Q"]) -> dict:

env\_var = load\_env(api = "https://data.sec.gov/api/xbrl/companyfacts/", cik = cik)

if env\_var["url"] is None or env\_var["headers"] is None:

print("Error in loading environment variables, check .env file")

return {}

try:

res = requests.get(env\_var["url"], headers=env\_var["headers"]).json()

data = res['facts']['us-gaap']

metadata = {

key: {

"label": data[key]["label"],

"description": data[key]["description"]

}

for key in data.keys()

}

result = {"metadata": metadata, "data": {}, "fileDate": {}}

for key in metadata.keys():

for k in data[key]['units'].keys():

record\_list = [{'val': item['val'], 'fileDate': item['filed'], 'fyfp': str(item['fy']) + item['fp'][1], 'endDate': item['end']} for item in data[key]['units'][k] if item['form'] in reportType]

record\_list, file\_date\_list = historical\_10Q\_dw(pd.DataFrame(record\_list))

if len(record\_list) < 4:

continue

result["data"][key] = record\_list

result["fileDate"][key] = file\_date\_list

print(f"{reportType} report loaded -- SUCCESS")

result['data'], result['time'] = historical\_10Q\_merge(result['data'], result['fileDate'])

result['metadata']['fileDate'] = {"label": "Filed Date", "description": "The last release date of the filing."}

print(f"{reportType} report loaded -- SUCCESS")

return result

except:

print("Error in SEC API call, check API validity")

return {}

def get\_report\_by\_ticker(ticker, reportType="10-Q") -> dict:

cik\_mapping = get\_ticker\_cik\_mapping([ticker])

if cik\_mapping["ticker"] is None:

return {}

return get\_report\_by\_CIK(cik\_mapping["ticker"], reportType=reportType)

import pandas as pd

import json

import os

def historical\_10Q\_dw(df):

if df.empty:

return {}, {}

df[['fileDate', 'endDate']] = df[['fileDate', 'endDate']].apply(pd.to\_datetime)

df = df.sort\_values(by=['fileDate', 'endDate'])

df = df.drop\_duplicates(subset='fileDate', keep='last')

df.index = df['fyfp']

return df['val'].to\_dict(), df['fileDate'].to\_dict()

def historical\_10Q\_merge(data, fileDate):

col\_name = data.keys()

data = (pd.concat([pd.Series(data[key]) for key in data.keys()], axis=1))

data = data.fillna('--')

data.columns = col\_name

data = data.sort\_index()

idx = data.index.astype(str)

data.index = idx.str[:-1] + ' Q' + idx.str[-1]

fileDate = pd.concat([pd.Series(fileDate[key]) for key in fileDate.keys()]).sort\_values().drop\_duplicates(keep='last').astype(str).to\_frame(name='fileDate')

fileDateIdx = fileDate.index.astype(str)

fileDate.index = fileDateIdx.str[:-1] + ' Q' + fileDateIdx.str[-1]

data = pd.merge(data, fileDate, left\_index=True, right\_index=True, how='inner')

return data.to\_dict(), list(data.index)

def read\_json(filepath):

if os.path.isfile(filepath):

with open(filepath, 'r') as f:

data = json.load(f)

f.close()

return data

return [{}]

def get\_sector\_options\_results(filter\_sector = "--", filepath = "functions/data/sector\_mapping.json"):

if not os.path.isfile(filepath):

return {"opt" : [], "res" : [], "res\_display" : "none", "ini\_alert" : "inline-block"}

try:

filter\_sector = filter\_sector[0:3]

data = pd.read\_json(filepath).drop\_duplicates()

data['sic\_'] = data['sic'].str[:-1]

d\_opt = data.groupby('sic\_')['sicDescription'].apply(lambda x: ' | '.join(x.unique())).drop(['000',''])

d\_opt = pd.DataFrame([d\_opt.index,d\_opt]).T

d\_opt.columns = ['sic','sicDescription']

return {

"opt" : (d\_opt['sic'] + '0 : ' + d\_opt['sicDescription']).to\_list(),

"res" : data[data['sic\_'] == filter\_sector][["tickers", "name", "cik", 'sic', 'sicDescription']].drop\_duplicates().sort\_values('sic').values.tolist(),

"res\_display" : "inline-block",

"ini\_alert" : "none"

}

except:

print("Error in loading sectors.")

return []

from functions.tenQ\_report import get\_ticker\_cik\_mapping, get\_company\_info\_by\_CIK, get\_company\_info\_by\_ticker, get\_report\_by\_CIK, get\_report\_by\_ticker

from functions.cik\_mapping import get\_mapping

from functions.data\_wrangle import read\_json, get\_sector\_options\_results

from functions.data\_load import sample\_data\_load

def initialize\_api():

print("Initializing data...")

get\_mapping()

print("Data initialization -- SUCCESS")

def search\_cik\_api(ticker):

cik\_mapping = get\_ticker\_cik\_mapping([ticker])

if cik\_mapping["ticker"] is None:

return {str(ticker):"The above ticker is invalid."}

return cik\_mapping

def search\_company\_api(company, search\_method):

if search\_method == "CIK number":

company\_info = get\_company\_info\_by\_CIK(company)

else:

company\_info = get\_company\_info\_by\_ticker(company)

return company\_info

def historical\_10Q\_api(company, search\_method):

if search\_method == "CIK number":

historical\_10Q = get\_report\_by\_CIK(company)

else:

historical\_10Q = get\_report\_by\_ticker(company)

return historical\_10Q

def search\_sector\_api(filter\_sector = "--"):

return get\_sector\_options\_results(filter\_sector=filter\_sector)

def sample\_data\_api():

return sample\_data\_load()

# gap keys

{

"0": {

"0": "AcceleratedShareRepurchaseProgramAdjustment",

"1": "AcceleratedShareRepurchasesFinalPricePaidPerShare",

"2": "AcceleratedShareRepurchasesSettlementPaymentOrReceipt",

"3": "AccountsPayableCurrent",

"4": "AccountsReceivableNetCurrent",

"5": "AccruedIncomeTaxesNoncurrent",

"6": "AccruedIncomeTaxesPayable",

"7": "AccruedLiabilitiesCurrent",

"8": "AccruedProfessionalFeesCurrentAndNoncurrent",

"9": "AccruedRentCurrent",

"10": "AccruedRentNoncurrent",

"11": "AccruedRoyaltiesCurrent",

"12": "AccumulatedDepreciationDepletionAndAmortizationPropertyPlantAndEquipment",

"13": "AccumulatedOtherComprehensiveIncomeLossCumulativeChangesInNetGainLossFromCashFlowHedgesEffectNetOfTax",

"14": "AccumulatedOtherComprehensiveIncomeLossNetOfTax",

"15": "AdditionalPaidInCapital",

"16": "AdditionalPaidInCapitalCommonStock",

"17": "AdjustmentsNoncashItemsToReconcileNetIncomeLossToCashProvidedByUsedInOperatingActivitiesOther",

"18": "AdjustmentsRelatedToTaxWithholdingForShareBasedCompensation",

"19": "AdjustmentsToAdditionalPaidInCapitalConvertibleDebtWithConversionFeature",

"20": "AdjustmentsToAdditionalPaidInCapitalEquityComponentOfConvertibleDebtSubsequentAdjustments",

"21": "AdjustmentsToAdditionalPaidInCapitalSharebasedCompensationRequisiteServicePeriodRecognitionValue",

"22": "AdjustmentsToAdditionalPaidInCapitalTaxEffectFromShareBasedCompensation",

"23": "AdvertisingExpense",

"24": "AllocatedShareBasedCompensationExpense",

"25": "AllowanceForDoubtfulAccountsReceivableCurrent",

"26": "AmortizationOfAcquiredIntangibleAssets",

"27": "AmortizationOfDebtDiscountPremium",

"28": "AmortizationOfFinancingCosts",

"29": "AmortizationOfFinancingCostsAndDiscounts",

"30": "AmortizationOfIntangibleAssets",

"31": "AntidilutiveSecuritiesExcludedFromComputationOfEarningsPerShareAmount",

"32": "AssetRetirementObligation",

"33": "AssetRetirementObligationsNoncurrent",

"34": "Assets",

"35": "AssetsCurrent",

"36": "AssetsHeldForSaleCurrent",

"37": "AvailableForSaleDebtSecuritiesAccumulatedGrossUnrealizedGainBeforeTax",

"38": "AvailableForSaleDebtSecuritiesAccumulatedGrossUnrealizedLossBeforeTax",

"39": "AvailableForSaleDebtSecuritiesAmortizedCostBasis",

"40": "AvailableForSaleSecurities",

"41": "AvailableForSaleSecuritiesAccumulatedGrossUnrealizedGainBeforeTax",

"42": "AvailableForSaleSecuritiesAmortizedCost",

"43": "AvailableForSaleSecuritiesChangeInNetUnrealizedHoldingGainLossNetOfTax",

"44": "AvailableForSaleSecuritiesContinuousUnrealizedLossPosition12MonthsOrLongerAccumulatedLoss",

"45": "AvailableForSaleSecuritiesContinuousUnrealizedLossPosition12MonthsOrLongerAggregateLosses",

"46": "AvailableForSaleSecuritiesContinuousUnrealizedLossPositionAccumulatedLoss",

"47": "AvailableForSaleSecuritiesContinuousUnrealizedLossPositionAggregateLosses",

"48": "AvailableForSaleSecuritiesContinuousUnrealizedLossPositionFairValue",

"49": "AvailableForSaleSecuritiesContinuousUnrealizedLossPositionLessThan12MonthsAccumulatedLoss",

"50": "AvailableForSaleSecuritiesContinuousUnrealizedLossPositionLessThan12MonthsAggregateLosses",

"51": "AvailableForSaleSecuritiesContinuousUnrealizedLossPositionLessThanTwelveMonthsFairValue",

"52": "AvailableForSaleSecuritiesContinuousUnrealizedLossPositionTwelveMonthsOrLongerFairValue",

"53": "AvailableForSaleSecuritiesDebtMaturitiesAfterOneThroughFiveYearsAmortizedCost",

"54": "AvailableForSaleSecuritiesDebtMaturitiesAfterOneThroughFiveYearsFairValue",

"55": "AvailableForSaleSecuritiesDebtMaturitiesAmortizedCost",

"56": "AvailableForSaleSecuritiesDebtMaturitiesFairValue",

"57": "AvailableForSaleSecuritiesDebtMaturitiesWithinOneYearAmortizedCost",

"58": "AvailableForSaleSecuritiesDebtMaturitiesWithinOneYearFairValue",

"59": "AvailableForSaleSecuritiesDebtMaturitiesWithoutSingleMaturityDateAmortizedCost",

"60": "AvailableForSaleSecuritiesDebtMaturitiesWithoutSingleMaturityDateFairValue",

"61": "AvailableForSaleSecuritiesDebtSecurities",

"62": "AvailableForSaleSecuritiesFairValueDisclosure",

"63": "AvailableForSaleSecuritiesGrossRealizedGainLossNet",

"64": "AvailableForSaleSecuritiesGrossUnrealizedGains",

"65": "AvailableForSaleSecuritiesGrossUnrealizedLoss",

"66": "AvailableForSaleSecuritiesGrossUnrealizedLosses1",

"67": "AvailableForSaleSecuritiesInUnrealizedLossPositionsQualitativeDisclosureNumberOfPositions",

"68": "AvailableforsaleSecuritiesContinuousUnrealizedLossPosition12MonthsOrLongerAggregateLosses1",

"69": "AvailableforsaleSecuritiesContinuousUnrealizedLossPosition12MonthsOrLongerAggregateLosses2",

"70": "AvailableforsaleSecuritiesContinuousUnrealizedLossPositionAggregateLosses1",

"71": "AvailableforsaleSecuritiesContinuousUnrealizedLossPositionAggregateLosses2",

"72": "AvailableforsaleSecuritiesContinuousUnrealizedLossPositionLessThan12MonthsAggregateLosses1",

"73": "AvailableforsaleSecuritiesContinuousUnrealizedLossPositionLessThan12MonthsAggregateLosses2",

"74": "AvailableforsaleSecuritiesGrossUnrealizedGain",

"75": "AvailableforsaleSecuritiesGrossUnrealizedGainLoss1",

"76": "AvailableforsaleSecuritiesInUnrealizedLossPositionsQualitativeDisclosureNumberOfPositions1",

"77": "BusinessAcquisitionContingentConsiderationPotentialCashPayment",

"78": "BusinessAcquisitionCostOfAcquiredEntityCashPaid",

"79": "BusinessAcquisitionCostOfAcquiredEntityTransactionCosts",

"80": "BusinessAcquisitionProFormaEarningsPerShareBasic",

"81": "BusinessAcquisitionProFormaEarningsPerShareDiluted",

"82": "BusinessAcquisitionProFormaNetIncomeLoss",

"83": "BusinessAcquisitionProFormaRevenue",

"84": "BusinessAcquisitionPurchasePriceAllocationAmortizableIntangibleAssets",

"85": "BusinessAcquisitionPurchasePriceAllocationAssetsAcquired",

"86": "BusinessAcquisitionPurchasePriceAllocationAssetsAcquiredLiabilitiesAssumedNet",

"87": "BusinessAcquisitionPurchasePriceAllocationCurrentAssetsInventory",

"88": "BusinessAcquisitionPurchasePriceAllocationCurrentAssetsPrepaidExpenseAndOtherAssets",

"89": "BusinessAcquisitionPurchasePriceAllocationCurrentAssetsReceivables",

"90": "BusinessAcquisitionPurchasePriceAllocationCurrentLiabilitiesAccountsPayable",

"91": "BusinessAcquisitionPurchasePriceAllocationCurrentLiabilitiesAccruedLiabilities",

"92": "BusinessAcquisitionPurchasePriceAllocationGoodwillAmount",

"93": "BusinessAcquisitionPurchasePriceAllocationNoncurrentAssets",

"94": "BusinessAcquisitionPurchasePriceAllocationNotesPayableAndLongTermDebt",

"95": "BusinessAcquisitionPurchasePriceAllocationOtherNoncurrentAssets",

"96": "BusinessAcquisitionPurchasePriceAllocationPropertyPlantAndEquipment",

"97": "BusinessAcquisitionsProFormaNetIncomeLoss",

"98": "BusinessAcquisitionsProFormaRevenue",

"99": "BusinessCombinationAcquisitionRelatedCosts",

"100": "CapitalLeaseObligationsCurrent",

"101": "CapitalLeaseObligationsNoncurrent",

"102": "CapitalLeasesFutureMinimumPaymentsDue",

"103": "CapitalLeasesFutureMinimumPaymentsDueCurrent",

"104": "CapitalLeasesFutureMinimumPaymentsDueInFiveYears",

"105": "CapitalLeasesFutureMinimumPaymentsDueInFourYears",

"106": "CapitalLeasesFutureMinimumPaymentsDueInThreeYears",

"107": "CapitalLeasesFutureMinimumPaymentsDueInTwoYears",

"108": "CapitalLeasesFutureMinimumPaymentsDueThereafter",

"109": "CapitalLeasesFutureMinimumPaymentsPresentValueOfNetMinimumPayments",

"110": "CashAndCashEquivalentsAtCarryingValue",

"111": "CashAndCashEquivalentsPeriodIncreaseDecrease",

"112": "CashCashEquivalentsRestrictedCashAndRestrictedCashEquivalents",

"113": "CashCashEquivalentsRestrictedCashAndRestrictedCashEquivalentsPeriodIncreaseDecreaseIncludingExchangeRateEffect",

"114": "CashEquivalentsAtCarryingValue",

"115": "ClassOfWarrantOrRightNumberOfSecuritiesCalledByEachWarrantOrRight",

"116": "ClassOfWarrantOrRightNumberOfSecuritiesCalledByWarrantsOrRights",

"117": "ClassOfWarrantOrRightOutstanding",

"118": "CommercialPaper",

"119": "CommitmentsAndContingencies",

"120": "CommonStockCapitalSharesReservedForFutureIssuance",

"121": "CommonStockDividendsPerShareCashPaid",

"122": "CommonStockDividendsPerShareDeclared",

"123": "CommonStockParOrStatedValuePerShare",

"124": "CommonStockSharesAuthorized",

"125": "CommonStockSharesIssued",

"126": "CommonStockSharesOutstanding",

"127": "CommonStockValue",

"128": "ComprehensiveIncomeNetOfTax",

"129": "ComprehensiveIncomeNetOfTaxIncludingPortionAttributableToNoncontrollingInterest",

"130": "ContractWithCustomerLiability",

"131": "ContractWithCustomerLiabilityCurrent",

"132": "ContractWithCustomerLiabilityIncreaseDecreaseForContractAcquiredInBusinessCombination",

"133": "ContractWithCustomerLiabilityNoncurrent",

"134": "ContractWithCustomerLiabilityRevenueRecognized",

"135": "ConvertibleDebt",

"136": "ConvertibleDebtCurrent",

"137": "ConvertibleDebtFairValueDisclosures",

"138": "ConvertibleDebtNoncurrent",

"139": "CostMethodInvestments",

"140": "CostOfGoodsAndServicesSold",

"141": "CostOfGoodsSoldAmortization",

"142": "CostOfRevenue",

"143": "CumulativeEffectOfNewAccountingPrincipleInPeriodOfAdoption",

"144": "CurrentFederalTaxExpenseBenefit",

"145": "CurrentForeignTaxExpenseBenefit",

"146": "CurrentIncomeTaxExpenseBenefit",

"147": "CurrentStateAndLocalTaxExpenseBenefit",

"148": "CustomerAdvancesAndDeposits",

"149": "DebtConversionConvertedInstrumentSharesIssued1",

"150": "DebtCurrent",

"151": "DebtInstrumentConvertibleCarryingAmountOfTheEquityComponent",

"152": "DebtInstrumentConvertibleConversionPrice1",

"153": "DebtInstrumentConvertibleConversionRatio",

"154": "DebtInstrumentConvertibleConversionRatio1",

"155": "DebtInstrumentConvertibleEffectiveInterestRate",

"156": "DebtInstrumentConvertibleIfConvertedValueInExcessOfPrincipal",

"157": "DebtInstrumentConvertibleInterestExpense",

"158": "DebtInstrumentConvertibleNumberOfEquityInstruments",

"159": "DebtInstrumentConvertibleThresholdPercentageOfStockPriceTrigger",

"160": "DebtInstrumentFaceAmount",

"161": "DebtInstrumentInterestRateEffectivePercentage",

"162": "DebtInstrumentInterestRateStatedPercentage",

"163": "DebtInstrumentUnamortizedDiscount",

"164": "DebtInstrumentUnamortizedDiscountPremiumAndDebtIssuanceCostsNet",

"165": "DebtSecuritiesAvailableForSaleContinuousUnrealizedLossPosition12MonthsOrLonger",

"166": "DebtSecuritiesAvailableForSaleContinuousUnrealizedLossPosition12MonthsOrLongerAccumulatedLoss",

"167": "DebtSecuritiesAvailableForSaleContinuousUnrealizedLossPositionLessThan12Months",

"168": "DebtSecuritiesAvailableForSaleContinuousUnrealizedLossPositionLessThan12MonthsAccumulatedLoss",

"169": "DebtSecuritiesAvailableForSaleUnrealizedLossPosition",

"170": "DebtSecuritiesAvailableForSaleUnrealizedLossPositionAccumulatedLoss",

"171": "DeferredCompensationLiabilityClassifiedNoncurrent",

"172": "DeferredFederalIncomeTaxExpenseBenefit",

"173": "DeferredForeignIncomeTaxExpenseBenefit",

"174": "DeferredIncomeTaxAssetsNet",

"175": "DeferredIncomeTaxExpenseBenefit",

"176": "DeferredIncomeTaxLiabilities",

"177": "DeferredIncomeTaxLiabilitiesNet",

"178": "DeferredRentCredit",

"179": "DeferredRevenue",

"180": "DeferredRevenueAdditions",

"181": "DeferredRevenueCurrent",

"182": "DeferredRevenueNoncurrent",

"183": "DeferredRevenueRevenueRecognized",

"184": "DeferredRevenueRevenueRecognized1",

"185": "DeferredStateAndLocalIncomeTaxExpenseBenefit",

"186": "DeferredTaxAssetsGross",

"187": "DeferredTaxAssetsGrossNoncurrent",

"188": "DeferredTaxAssetsLiabilitiesNet",

"189": "DeferredTaxAssetsLiabilitiesNetNoncurrent",

"190": "DeferredTaxAssetsNet",

"191": "DeferredTaxAssetsNetCurrent",

"192": "DeferredTaxAssetsNetNoncurrent",

"193": "DeferredTaxAssetsOperatingLossCarryforwards",

"194": "DeferredTaxAssetsOperatingLossCarryforwardsDomestic",

"195": "DeferredTaxAssetsOperatingLossCarryforwardsForeign",

"196": "DeferredTaxAssetsOperatingLossCarryforwardsStateAndLocal",

"197": "DeferredTaxAssetsTaxCreditCarryforwards",

"198": "DeferredTaxAssetsTaxCreditCarryforwardsOther",

"199": "DeferredTaxAssetsTaxCreditCarryforwardsResearch",

"200": "DeferredTaxAssetsTaxDeferredExpenseCompensationAndBenefitsShareBasedCompensationCost",

"201": "DeferredTaxAssetsTaxDeferredExpenseReservesAndAccruals",

"202": "DeferredTaxAssetsValuationAllowance",

"203": "DeferredTaxExpenseFromStockOptionsExercised",

"204": "DeferredTaxLiabilities",

"205": "DeferredTaxLiabilitiesNoncurrent",

"206": "DeferredTaxLiabilitiesUndistributedForeignEarnings",

"207": "DefinedContributionPlanCostRecognized",

"208": "DefinedContributionPlanMaximumAnnualContributionsPerEmployeePercent",

"209": "DepositsAssets",

"210": "DepositsAssetsNoncurrent",

"211": "Depreciation",

"212": "DepreciationAndAmortization",

"213": "DepreciationDepletionAndAmortization",

"214": "DerivativeLiabilities",

"215": "DerivativeLiabilitiesNoncurrent",

"216": "Dividends",

"217": "DividendsCash",

"218": "DividendsCommonStockCash",

"219": "EarningsPerShareBasic",

"220": "EarningsPerShareDiluted",

"221": "EffectiveIncomeTaxRateContinuingOperations",

"222": "EffectiveIncomeTaxRateReconciliationAtFederalStatutoryIncomeTaxRate",

"223": "EmployeeRelatedLiabilitiesCurrent",

"224": "EmployeeServiceShareBasedCompensationAllocationOfRecognizedPeriodCostsCapitalizedAmount",

"225": "EmployeeServiceShareBasedCompensationNonvestedAwardsTotalCompensationCostNotYetRecognized",

"226": "EmployeeServiceShareBasedCompensationNonvestedAwardsTotalCompensationCostNotYetRecognizedPeriodForRecognition",

"227": "EmployeeServiceShareBasedCompensationUnrecognizedCompensationCostsOnNonvestedAwards",

"228": "EntityWideDisclosureOnGeographicAreasLongLivedAssetsInIndividualForeignCountriesAmount",

"229": "EquitySecuritiesFvNi",

"230": "EquitySecuritiesFvNiUnrealizedGain",

"231": "EquitySecuritiesFvNiUnrealizedLoss",

"232": "ExcessTaxBenefitFromShareBasedCompensationFinancingActivities",

"233": "ExcessTaxBenefitFromShareBasedCompensationOperatingActivities",

"234": "ExtinguishmentOfDebtAmount",

"235": "FairValueMeasurementWithUnobservableInputsReconciliationRecurringBasisAssetTransfersNet",

"236": "FairValueMeasurementWithUnobservableInputsReconciliationRecurringBasisAssetValue",

"237": "FiniteLivedIntangibleAssetsAccumulatedAmortization",

"238": "FiniteLivedIntangibleAssetsAmortizationExpense",

"239": "FiniteLivedIntangibleAssetsAmortizationExpenseAfterYearFive",

"240": "FiniteLivedIntangibleAssetsAmortizationExpenseNextTwelveMonths",

"241": "FiniteLivedIntangibleAssetsAmortizationExpenseRemainderOfFiscalYear",

"242": "FiniteLivedIntangibleAssetsAmortizationExpenseYearFive",

"243": "FiniteLivedIntangibleAssetsAmortizationExpenseYearFour",

"244": "FiniteLivedIntangibleAssetsAmortizationExpenseYearThree",

"245": "FiniteLivedIntangibleAssetsAmortizationExpenseYearTwo",

"246": "FiniteLivedIntangibleAssetsGross",

"247": "FiniteLivedIntangibleAssetsNet",

"248": "FiniteLivedIntangibleAssetsPeriodIncreaseDecrease",

"249": "FiniteLivedIntangibleAssetsUsefulLife",

"250": "ForeignCurrencyTransactionGainLossRealized",

"251": "FutureAmortizationExpenseAfterYearFive",

"252": "FutureAmortizationExpenseRemainderOfFiscalYear",

"253": "FutureAmortizationExpenseYearFive",

"254": "FutureAmortizationExpenseYearFour",

"255": "FutureAmortizationExpenseYearOne",

"256": "FutureAmortizationExpenseYearThree",

"257": "FutureAmortizationExpenseYearTwo",

"258": "GainLossOnForeignCurrencyCashFlowHedgeIneffectiveness",

"259": "GainLossOnInvestments",

"260": "GainsLossesOnExtinguishmentOfDebt",

"261": "Goodwill",

"262": "GoodwillAcquiredDuringPeriod",

"263": "GoodwillImpairmentLoss",

"264": "GoodwillPeriodIncreaseDecrease",

"265": "GoodwillTranslationAndPurchaseAccountingAdjustments",

"266": "GrossProfit",

"267": "ImpairmentOfInvestments",

"268": "IncomeLossFromContinuingOperationsBeforeIncomeTaxesDomestic",

"269": "IncomeLossFromContinuingOperationsBeforeIncomeTaxesExtraordinaryItemsNoncontrollingInterest",

"270": "IncomeLossFromContinuingOperationsBeforeIncomeTaxesForeign",

"271": "IncomeLossFromContinuingOperationsBeforeIncomeTaxesMinorityInterestAndIncomeLossFromEquityMethodInvestments",

"272": "IncomeTaxExpenseBenefit",

"273": "IncomeTaxExpenseBenefitContinuingOperationsAdjustmentOfDeferredTaxAssetLiability",

"274": "IncomeTaxesPaidNet",

"275": "IncomeTaxReconciliationForeignIncomeTaxRateDifferential",

"276": "IncomeTaxReconciliationIncomeTaxExpenseBenefitAtFederalStatutoryIncomeTaxRate",

"277": "IncomeTaxReconciliationNondeductibleExpenseRestructuringCharges",

"278": "IncomeTaxReconciliationNondeductibleExpenseShareBasedCompensationCost",

"279": "IncomeTaxReconciliationOtherAdjustments",

"280": "IncomeTaxReconciliationStateAndLocalIncomeTaxes",

"281": "IncomeTaxReconciliationTaxCreditsResearch",

"282": "IncreaseDecreaseInAccountsPayable",

"283": "IncreaseDecreaseInAccountsReceivable",

"284": "IncreaseDecreaseInAccruedLiabilities",

"285": "IncreaseDecreaseInAccruedLiabilitiesAndOtherOperatingLiabilities",

"286": "IncreaseDecreaseInContractWithCustomerLiability",

"287": "IncreaseDecreaseInInventories",

"288": "IncreaseDecreaseInOtherNoncurrentLiabilities",

"289": "IncreaseDecreaseInOtherOperatingAssets",

"290": "IncreaseDecreaseInPrepaidDeferredExpenseAndOtherAssets",

"291": "IncrementalCommonSharesAttributableToCallOptionsAndWarrants",

"292": "IncrementalCommonSharesAttributableToConversionOfDebtSecurities",

"293": "InsuranceRecoveries",

"294": "IntangibleAssetsNetExcludingGoodwill",

"295": "InterestExpense",

"296": "InterestExpenseDebt",

"297": "InterestExpenseDebtExcludingAmortization",

"298": "InterestPaidNet",

"299": "InterestPayableCurrent",

"300": "InterestRateDerivativeLiabilitiesAtFairValue",

"301": "InventoryFinishedGoods",

"302": "InventoryFinishedGoodsNetOfReserves",

"303": "InventoryNet",

"304": "InventoryRawMaterials",

"305": "InventoryRawMaterialsNetOfReserves",

"306": "InventoryWorkInProcess",

"307": "InventoryWorkInProcessNetOfReserves",

"308": "InvestmentIncomeInterest",

"309": "LeaseAndRentalExpense",

"310": "LesseeOperatingLeaseLiabilityPaymentsDue",

"311": "LesseeOperatingLeaseLiabilityPaymentsDueAfterYearFive",

"312": "LesseeOperatingLeaseLiabilityPaymentsDueNextTwelveMonths",

"313": "LesseeOperatingLeaseLiabilityPaymentsDueYearFive",

"314": "LesseeOperatingLeaseLiabilityPaymentsDueYearFour",

"315": "LesseeOperatingLeaseLiabilityPaymentsDueYearThree",

"316": "LesseeOperatingLeaseLiabilityPaymentsDueYearTwo",

"317": "LesseeOperatingLeaseLiabilityPaymentsRemainderOfFiscalYear",

"318": "LesseeOperatingLeaseLiabilityUndiscountedExcessAmount",

"319": "Liabilities",

"320": "LiabilitiesAndStockholdersEquity",

"321": "LiabilitiesAssumed",

"322": "LiabilitiesAssumed1",

"323": "LiabilitiesCurrent",

"324": "LineOfCreditFacilityCurrentBorrowingCapacity",

"325": "LitigationSettlementExpense",

"326": "LitigationSettlementGross",

"327": "LongTermDebt",

"328": "LongTermDebtCurrent",

"329": "LongTermDebtNoncurrent",

"330": "MarketableSecurities",

"331": "MarketableSecuritiesCurrent",

"332": "MarketableSecuritiesRealizedGainLossExcludingOtherThanTemporaryImpairments",

"333": "MarketableSecuritiesRealizedGainLossOtherThanTemporaryImpairmentsAmount",

"334": "MaterialsSuppliesAndOther",

"335": "MoneyMarketFundsAtCarryingValue",

"336": "NetCashProvidedByUsedInFinancingActivities",

"337": "NetCashProvidedByUsedInInvestingActivities",

"338": "NetCashProvidedByUsedInOperatingActivities",

"339": "NetIncomeLoss",

"340": "NoncashOrPartNoncashAcquisitionIntangibleAssetsAcquired",

"341": "NoncurrentAssets",

"342": "NonoperatingIncomeExpense",

"343": "NumberOfForeignCurrencyDerivativesHeld",

"344": "NumberOfReportableSegments",

"345": "OperatingExpenses",

"346": "OperatingIncomeLoss",

"347": "OperatingLeaseCost",

"348": "OperatingLeaseLiability",

"349": "OperatingLeaseLiabilityCurrent",

"350": "OperatingLeaseLiabilityNoncurrent",

"351": "OperatingLeasePayments",

"352": "OperatingLeaseRightOfUseAsset",

"353": "OperatingLeasesFutureMinimumPaymentsDue",

"354": "OperatingLeasesFutureMinimumPaymentsDueCurrent",

"355": "OperatingLeasesFutureMinimumPaymentsDueInFiveYears",

"356": "OperatingLeasesFutureMinimumPaymentsDueInFourYears",

"357": "OperatingLeasesFutureMinimumPaymentsDueInThreeYears",

"358": "OperatingLeasesFutureMinimumPaymentsDueInTwoYears",

"359": "OperatingLeasesFutureMinimumPaymentsDueThereafter",

"360": "OperatingLeaseWeightedAverageDiscountRatePercent",

"361": "OperatingLossCarryforwards",

"362": "OptionIndexedToIssuersEquityStrikePrice1",

"363": "OtherAccruedLiabilitiesCurrent",

"364": "OtherAssets",

"365": "OtherAssetsCurrent",

"366": "OtherAssetsMiscellaneousNoncurrent",

"367": "OtherAssetsNoncurrent",

"368": "OtherCommitment",

"369": "OtherComprehensiveIncomeAvailableforsaleSecuritiesAdjustmentNetOfTaxPortionAttributableToParent",

"370": "OtherComprehensiveIncomeDerivativesQualifyingAsHedgesNetOfTaxPortionAttributableToParent",

"371": "OtherComprehensiveIncomeLossAvailableForSaleSecuritiesAdjustmentBeforeTax",

"372": "OtherComprehensiveIncomeLossCashFlowHedgeGainLossAfterReclassificationAndTax",

"373": "OtherComprehensiveIncomeLossCashFlowHedgeGainLossBeforeReclassificationAfterTax",

"374": "OtherComprehensiveIncomeLossCashFlowHedgeGainLossReclassificationAfterTax",

"375": "OtherComprehensiveIncomeLossDerivativesQualifyingAsHedgesNetOfTax",

"376": "OtherComprehensiveIncomeLossNetOfTax",

"377": "OtherComprehensiveIncomeLossNetOfTaxPeriodIncreaseDecrease",

"378": "OtherComprehensiveIncomeLossNetOfTaxPortionAttributableToParent",

"379": "OtherComprehensiveIncomeLossReclassificationAdjustmentForSaleOfSecuritiesIncludedInNetIncomeNetOfTax",

"380": "OtherComprehensiveIncomeLossReclassificationAdjustmentForSaleOfSecuritiesIncludedInNetIncomeTax",

"381": "OtherComprehensiveIncomeLossReclassificationAdjustmentFromAOCIForSaleOfSecuritiesNetOfTax",

"382": "OtherComprehensiveIncomeLossReclassificationAdjustmentFromAOCIForSaleOfSecuritiesTax",

"383": "OtherComprehensiveIncomeLossReclassificationAdjustmentFromAOCIOnDerivativesNetOfTax",

"384": "OtherComprehensiveIncomeLossReclassificationAdjustmentFromAOCIPensionAndOtherPostretirementBenefitPlansForNetGainLossNetOfTax",

"385": "OtherComprehensiveIncomeLossTax",

"386": "OtherComprehensiveIncomeReclassificationAdjustmentForSaleOfSecuritiesIncludedInNetIncomeBeforeTax",

"387": "OtherComprehensiveIncomeReclassificationAdjustmentForSaleOfSecuritiesIncludedInNetIncomeNetOfTax",

"388": "OtherComprehensiveIncomeReclassificationAdjustmentForSaleOfSecuritiesIncludedInNetIncomeTax",

"389": "OtherComprehensiveIncomeReclassificationOfDefinedBenefitPlansNetGainLossRecognizedInNetPeriodicBenefitCostNetOfTax",

"390": "OtherComprehensiveIncomeUnrealizedGainLossOnDerivativesArisingDuringPeriodNetOfTax",

"391": "OtherComprehensiveIncomeUnrealizedGainLossOnDerivativesArisingDuringPeriodTax",

"392": "OtherComprehensiveIncomeUnrealizedHoldingGainLossOnSecuritiesArisingDuringPeriodBeforeTax",

"393": "OtherComprehensiveIncomeUnrealizedHoldingGainLossOnSecuritiesArisingDuringPeriodNetOfTax",

"394": "OtherComprehensiveIncomeUnrealizedHoldingGainLossOnSecuritiesArisingDuringPeriodTax",

"395": "OtherLiabilitiesNoncurrent",

"396": "OtherLongTermInvestments",

"397": "OtherNoncashIncomeExpense",

"398": "OtherNoncurrentLiabilities",

"399": "OtherNonoperatingIncomeExpense",

"400": "OtherNonrecurringExpense",

"401": "OtherNonrecurringIncomeExpense",

"402": "OtherPrepaidExpenseCurrent",

"403": "OtherSundryLiabilitiesNoncurrent",

"404": "OtherThanTemporaryImpairmentLossDebtSecuritiesAvailableForSaleRecognizedInEarnings",

"405": "PaymentsForHedgeFinancingActivities",

"406": "PaymentsForLegalSettlements",

"407": "PaymentsForProceedsFromOtherInvestingActivities",

"408": "PaymentsForRepurchaseOfCommonStock",

"409": "PaymentsForRestructuring",

"410": "PaymentsOfDebtIssuanceCosts",

"411": "PaymentsOfDividends",

"412": "PaymentsRelatedToTaxWithholdingForShareBasedCompensation",

"413": "PaymentsToAcquireAvailableForSaleSecuritiesDebt",

"414": "PaymentsToAcquireBusinessesNetOfCashAcquired",

"415": "PaymentsToAcquireMarketableSecurities",

"416": "PaymentsToAcquireOtherInvestments",

"417": "PaymentsToAcquireProductiveAssets",

"418": "PaymentsToAcquirePropertyPlantAndEquipment",

"419": "PreferredStockParOrStatedValuePerShare",

"420": "PreferredStockSharesAuthorized",

"421": "PreferredStockSharesIssued",

"422": "PreferredStockSharesOutstanding",

"423": "PreferredStockValueOutstanding",

"424": "PrepaidExpenseAndOtherAssetsCurrent",

"425": "PrepaidExpenseCurrent",

"426": "PrepaidInsurance",

"427": "PrepaidRent",

"428": "PrepaidTaxes",

"429": "ProceedsFromConvertibleDebt",

"430": "ProceedsFromDebtNetOfIssuanceCosts",

"431": "ProceedsFromIncomeTaxRefunds",

"432": "ProceedsFromIssuanceOfCommonStock",

"433": "ProceedsFromIssuanceOfDebt",

"434": "ProceedsFromIssuanceOfLongTermDebt",

"435": "ProceedsFromIssuanceOfWarrants",

"436": "ProceedsFromMaturitiesPrepaymentsAndCallsOfAvailableForSaleSecurities",

"437": "ProceedsFromPaymentsForOtherFinancingActivities",

"438": "ProceedsFromSaleAndMaturityOfMarketableSecurities",

"439": "ProceedsFromSaleOfAvailableForSaleSecurities",

"440": "ProceedsFromSaleOfAvailableForSaleSecuritiesDebt",

"441": "ProceedsFromStockOptionsExercised",

"442": "ProductWarrantyAccrual",

"443": "ProductWarrantyAccrualClassifiedCurrent",

"444": "ProductWarrantyAccrualPayments",

"445": "ProductWarrantyAccrualWarrantiesIssued",

"446": "ProductWarrantyExpense",

"447": "PropertyPlantAndEquipmentGross",

"448": "PropertyPlantAndEquipmentNet",

"449": "PropertyPlantAndEquipmentUsefulLifeMaximum",

"450": "PropertyPlantAndEquipmentUsefulLifeMinimum",

"451": "PurchaseObligation",

"452": "PurchaseObligationDueInFourthYear",

"453": "PurchaseObligationDueInNextTwelveMonths",

"454": "PurchaseObligationDueInSecondYear",

"455": "PurchaseObligationDueInThirdYear",

"456": "PurchaseObligationFutureMinimumPaymentsRemainderOfFiscalYear",

"457": "RealizedInvestmentGainsLosses",

"458": "ReclassificationFromAccumulatedOtherComprehensiveIncomeCurrentPeriodNetOfTax",

"459": "RepaymentsOfConvertibleDebt",

"460": "RepaymentsOfDebt",

"461": "RepaymentsOfLongTermCapitalLeaseObligations",

"462": "ResearchAndDevelopmentExpense",

"463": "ResearchAndDevelopmentInProcess",

"464": "RestructuringCharges",

"465": "RestructuringReserve",

"466": "RestructuringReserveSettledWithoutCash1",

"467": "RestructuringReserveSettledWithoutCash2",

"468": "RetainedEarningsAccumulatedDeficit",

"469": "RevenueFromContractWithCustomerExcludingAssessedTax",

"470": "RevenueRemainingPerformanceObligation",

"471": "RevenueRemainingPerformanceObligationPercentage",

"472": "Revenues",

"473": "RightOfUseAssetObtainedInExchangeForOperatingLeaseLiability",

"474": "SellingGeneralAndAdministrativeExpense",

"475": "SettlementLiabilitiesCurrent",

"476": "SeveranceCosts",

"477": "SeveranceCosts1",

"478": "ShareBasedCompensation",

"479": "ShareBasedCompensationArrangementByShareBasedPaymentAwardEquityInstrumentsOtherThanOptionsForfeitedInPeriod",

"480": "ShareBasedCompensationArrangementByShareBasedPaymentAwardEquityInstrumentsOtherThanOptionsForfeitedInPeriodWeightedAverageGrantDateFairValue",

"481": "ShareBasedCompensationArrangementByShareBasedPaymentAwardEquityInstrumentsOtherThanOptionsForfeituresWeightedAverageGrantDateFairValue",

"482": "ShareBasedCompensationArrangementByShareBasedPaymentAwardEquityInstrumentsOtherThanOptionsGrantsInPeriod",

"483": "ShareBasedCompensationArrangementByShareBasedPaymentAwardEquityInstrumentsOtherThanOptionsGrantsInPeriodWeightedAverageGrantDateFairValue",

"484": "ShareBasedCompensationArrangementByShareBasedPaymentAwardEquityInstrumentsOtherThanOptionsNonvestedNumber",

"485": "ShareBasedCompensationArrangementByShareBasedPaymentAwardEquityInstrumentsOtherThanOptionsNonvestedWeightedAverageGrantDateFairValue",

"486": "ShareBasedCompensationArrangementByShareBasedPaymentAwardEquityInstrumentsOtherThanOptionsVestedInPeriod",

"487": "ShareBasedCompensationArrangementByShareBasedPaymentAwardEquityInstrumentsOtherThanOptionsVestedInPeriodWeightedAverageGrantDateFairValue",

"488": "ShareBasedCompensationArrangementByShareBasedPaymentAwardMaximumEmployeeSubscriptionRate",

"489": "ShareBasedCompensationArrangementByShareBasedPaymentAwardNonOptionEquityInstrumentsOther",

"490": "ShareBasedCompensationArrangementByShareBasedPaymentAwardNonOptionEquityInstrumentsOutstandingNumber",

"491": "ShareBasedCompensationArrangementByShareBasedPaymentAwardNumberOfSharesAvailableForGrant",

"492": "ShareBasedCompensationArrangementByShareBasedPaymentAwardOptionsExercisableIntrinsicValue",

"493": "ShareBasedCompensationArrangementByShareBasedPaymentAwardOptionsExercisableNumber",

"494": "ShareBasedCompensationArrangementByShareBasedPaymentAwardOptionsExercisableWeightedAverageExercisePrice",

"495": "ShareBasedCompensationArrangementByShareBasedPaymentAwardOptionsExercisableWeightedAverageRemainingContractualTerm",

"496": "ShareBasedCompensationArrangementByShareBasedPaymentAwardOptionsExercisesInPeriod",

"497": "ShareBasedCompensationArrangementByShareBasedPaymentAwardOptionsExercisesInPeriodTotalIntrinsicValue",

"498": "ShareBasedCompensationArrangementByShareBasedPaymentAwardOptionsExercisesInPeriodWeightedAverageExercisePrice",

"499": "ShareBasedCompensationArrangementByShareBasedPaymentAwardOptionsForfeituresAndExpirationsInPeriodWeightedAverageExercisePrice",

"500": "ShareBasedCompensationArrangementByShareBasedPaymentAwardOptionsForfeituresInPeriod",

"501": "ShareBasedCompensationArrangementByShareBasedPaymentAwardOptionsForfeituresInPeriodWeightedAverageExercisePrice",

"502": "ShareBasedCompensationArrangementByShareBasedPaymentAwardOptionsGrantsInPeriod",

"503": "ShareBasedCompensationArrangementByShareBasedPaymentAwardOptionsGrantsInPeriodGross",

"504": "ShareBasedCompensationArrangementByShareBasedPaymentAwardOptionsGrantsInPeriodWeightedAverageExercisePrice",

"505": "ShareBasedCompensationArrangementByShareBasedPaymentAwardOptionsGrantsInPeriodWeightedAverageGrantDateFairValue",

"506": "ShareBasedCompensationArrangementByShareBasedPaymentAwardOptionsOtherIncreasesDecreasesInPeriod",

"507": "ShareBasedCompensationArrangementByShareBasedPaymentAwardOptionsOutstandingIntrinsicValue",

"508": "ShareBasedCompensationArrangementByShareBasedPaymentAwardOptionsOutstandingNumber",

"509": "ShareBasedCompensationArrangementByShareBasedPaymentAwardOptionsOutstandingWeightedAverageExercisePrice",

"510": "ShareBasedCompensationArrangementByShareBasedPaymentAwardOptionsOutstandingWeightedAverageRemainingContractualTerm",

"511": "ShareBasedCompensationArrangementByShareBasedPaymentAwardOptionsVestedAndExpectedToVestExercisableNumber",

"512": "ShareBasedCompensationArrangementByShareBasedPaymentAwardOptionsVestedAndExpectedToVestOutstandingAggregateIntrinsicValue",

"513": "ShareBasedCompensationArrangementByShareBasedPaymentAwardOptionsVestedAndExpectedToVestOutstandingNumber",

"514": "ShareBasedCompensationArrangementByShareBasedPaymentAwardOptionsVestedAndExpectedToVestOutstandingWeightedAverageExercisePrice",

"515": "ShareBasedCompensationArrangementByShareBasedPaymentAwardOptionsVestedAndExpectedToVestOutstandingWeightedAverageRemainingContractualTerm",

"516": "ShareBasedCompensationArrangementByShareBasedPaymentAwardOptionsVestedInPeriodFairValue",

"517": "ShareBasedCompensationArrangementByShareBasedPaymentAwardPerShareWeightedAveragePriceOfSharesPurchased",

"518": "ShareBasedCompensationArrangementsByShareBasedPaymentAwardOptionsExercisesInPeriodWeightedAverageExercisePrice",

"519": "ShareBasedCompensationArrangementsByShareBasedPaymentAwardOptionsForfeituresInPeriodWeightedAverageExercisePrice",

"520": "ShareBasedCompensationArrangementsByShareBasedPaymentAwardOptionsGrantsInPeriodWeightedAverageExercisePrice",

"521": "SharebasedCompensationArrangementBySharebasedPaymentAwardAwardVestingRightsPercentage",

"522": "SharebasedCompensationArrangementBySharebasedPaymentAwardOptionsExercisableIntrinsicValue1",

"523": "SharebasedCompensationArrangementBySharebasedPaymentAwardOptionsOutstandingWeightedAverageRemainingContractualTerm1",

"524": "SharebasedCompensationArrangementBySharebasedPaymentAwardOptionsVestedInPeriodFairValue1",

"525": "SharebasedCompensationArrangementBySharebasedPaymentAwardPurchasePriceOfCommonStockPercent",

"526": "SharebasedCompensationSharesAuthorizedUnderStockOptionPlansExercisePriceRangeExercisableOptionsWeightedAverageRemainingContractualTerm1",

"527": "SharePrice",

"528": "SharesPaidForTaxWithholdingForShareBasedCompensation",

"529": "StockGrantedDuringPeriodValueSharebasedCompensationGross",

"530": "StockholdersEquity",

"531": "StockholdersEquityNoteStockSplitConversionRatio1",

"532": "StockIssuedDuringPeriodSharesEmployeeStockPurchasePlans",

"533": "StockIssuedDuringPeriodSharesNewIssues",

"534": "StockIssuedDuringPeriodSharesStockOptionsExercised",

"535": "StockIssuedDuringPeriodValueEmployeeStockOwnershipPlan",

"536": "StockIssuedDuringPeriodValueShareBasedCompensation",

"537": "StockOptionPlanExpense",

"538": "StockRepurchasedAndRetiredDuringPeriodShares",

"539": "StockRepurchasedAndRetiredDuringPeriodValue",

"540": "StockRepurchasedDuringPeriodShares",

"541": "StockRepurchasedDuringPeriodValue",

"542": "StockRepurchaseProgramAuthorizedAmount",

"543": "StockRepurchaseProgramAuthorizedAmount1",

"544": "StockRepurchaseProgramRemainingAuthorizedRepurchaseAmount",

"545": "StockRepurchaseProgramRemainingAuthorizedRepurchaseAmount1",

"546": "TaxBenefitFromStockOptionsExercised",

"547": "TaxBenefitFromStockOptionsExercised1",

"548": "TaxCutsAndJobsActOf2017IncomeTaxExpenseBenefit",

"549": "TaxesPayableCurrent",

"550": "TemporaryEquityValueExcludingAdditionalPaidInCapital",

"551": "TreasuryStockAcquiredAverageCostPerShare",

"552": "TreasuryStockShares",

"553": "TreasuryStockSharesAcquired",

"554": "TreasuryStockValue",

"555": "TreasuryStockValueAcquiredCostMethod",

"556": "UndistributedEarningsOfForeignSubsidiaries",

"557": "UnrealizedGainLossOnCashFlowHedgingInstruments",

"558": "UnrealizedGainLossOnDerivatives",

"559": "UnrealizedGainLossOnSecurities",

"560": "UnrecognizedTaxBenefits",

"561": "UnrecognizedTaxBenefitsDecreasesResultingFromPriorPeriodTaxPositions",

"562": "UnrecognizedTaxBenefitsDecreasesResultingFromSettlementsWithTaxingAuthorities",

"563": "UnrecognizedTaxBenefitsIncomeTaxPenaltiesAndInterestAccrued",

"564": "UnrecognizedTaxBenefitsIncreasesResultingFromCurrentPeriodTaxPositions",

"565": "UnrecognizedTaxBenefitsIncreasesResultingFromPriorPeriodTaxPositions",

"566": "UnrecognizedTaxBenefitsReductionsResultingFromLapseOfApplicableStatuteOfLimitations",

"567": "UnrecognizedTaxBenefitsThatWouldImpactEffectiveTaxRate",

"568": "ValuationAllowancesAndReservesDeductions",

"569": "WeightedAverageNumberDilutedSharesOutstandingAdjustment",

"570": "WeightedAverageNumberOfDilutedSharesOutstanding",

"571": "WeightedAverageNumberOfSharesOutstandingBasic",

"572": "CapitalExpendituresIncurredButNotYetPaid",

"573": "DeferredTaxAssetsOther",

"574": "TreasuryStockRetiredCostMethodAmount",

"575": "TreasuryStockSharesRetired",

"576": "FinitelivedIntangibleAssetsAcquired1",

"577": "InventoryWriteDown",

"578": "ProductWarrantyAccrualPeriodIncreaseDecrease",

"579": "ProductWarrantyAccrualPreexistingIncreaseDecrease",

"580": "DeferredTaxLiabilitiesGoodwillAndIntangibleAssetsIntangibleAssets",

"581": "EffectiveIncomeTaxRateReconciliationForeignIncomeTaxRateDifferential",

"582": "EffectiveIncomeTaxRateReconciliationNondeductibleExpenseShareBasedCompensationCost",

"583": "EffectiveIncomeTaxRateReconciliationOtherAdjustments",

"584": "EffectiveIncomeTaxRateReconciliationStateAndLocalIncomeTaxes",

"585": "EffectiveIncomeTaxRateReconciliationTaxCreditsResearch",

"586": "PurchaseObligationDueAfterFifthYear",

"587": "PurchaseObligationDueInFifthYear",

"588": "UnrecognizedTaxBenefitsIncomeTaxPenaltiesAndInterestExpense",

"589": "RestrictedCashCurrent",

"590": "IncomeTaxExaminationIncreaseDecreaseInLiabilityFromPriorYear",

"591": "EffectiveIncomeTaxRateReconciliationFdiiAmount",

"592": "EffectiveIncomeTaxRateReconciliationFdiiPercent",

"593": "EquitySecuritiesWithoutReadilyDeterminableFairValueAmount",

"594": "EquitySecuritiesWithoutReadilyDeterminableFairValueUpwardPriceAdjustmentAnnualAmount",

"595": "EquitySecuritiesWithoutReadilyDeterminableFairValueUpwardPriceAdjustmentCumulativeAmount",

"596": "FinanceLeaseRightOfUseAssetAccumulatedAmortization",

"597": "ProceedsFromStockPlans",

"598": "BusinessCombinationConsiderationTransferred1",

"599": "EquitySecuritiesFVNINoncurrent",

"600": "InterestExpenseNonoperating",

"601": "PaymentsToAcquireEquitySecuritiesFvNi",

"602": "ProceedsFromSaleOfEquitySecuritiesFvNi",

"603": "UnbilledContractsReceivable",

"604": "IncomeTaxesReceivableNoncurrent"

}

}

#44 project

import json

from functools import cache

import requests

import pandas as pd

import plotly.express as px

import logging

def get\_all\_cik():

headers = {

'User-Agent': "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/42.0.2311.135 Safari/537.36 Edge/12.246",

'Accept': 'application/json'

}

url = "https://www.sec.gov/files/company\_tickers.json"

response = requests.get(url, headers=headers)

if "json" in response.headers.get('Content-Type'):

logging.info(f"Content Type is: --------------------------------------- {response.headers.get('Content-Type')}")

response = response.json()

df = pd.DataFrame.from\_dict(response).T

df.rename(columns={"cik\_str": "cik", "title": "NAME"}, inplace=True)

# formatting CIK number

df["cik"] = df.apply(lambda x: "CIK" + (10 - len(str(x['cik']))) \* '0' + str(x['cik']), axis=1)

return df

else:

logging.error(f"Content Type not json, but is: --------------------------------------- {response.headers.get('Content-Type')}")

html\_text\_response = response.text

logging.error(html\_text\_response)

df = pd.DataFrame.from\_dict(json.loads(html\_text\_response)).T

df.rename(columns={"cik\_str": "cik", "title": "NAME"}, inplace=True)

# formatting CIK number

df["cik"] = df.apply(lambda x: "CIK" + (10 - len(str(x['cik']))) \* '0' + str(x['cik']), axis=1)

return df

def create\_spark\_line(data, \_height: int = 100, \_width: int = 250):

df = None

if not isinstance(data, pd.DataFrame):

df = pd.DataFrame(data)

df = data

fig = px.area(df, height=\_height, width=\_width)

# hide and lock down axes

fig.update\_xaxes(visible=False, fixedrange=True)

fig.update\_yaxes(visible=False, fixedrange=True)

# remove facet/subplot labels

fig.update\_layout(annotations=[], overwrite=True)

# strip down the rest of the plot

fig.update\_layout(

showlegend=False,

plot\_bgcolor="white",

margin=dict(t=10, l=10, b=10, r=10))

fig.update\_traces(line\_color="#32CD32")

return fig.show()

def get\_company\_logo\_url(name):

url = 'https://s3-symbol-logo.tradingview.com/'

name = str.lower(name)

word\_to\_remove = [".com", "the ", " (the)", 'company', 'group']

for item in word\_to\_remove:

if item in name:

name = name.replace(item, "")

suffix = name.split()

suffix\_list = ["corp.", "corporation", "inc", 'incorporated', 'inc.', '(the)'

"limited", "ltd", 'plc', "laboratories", "communications", 'the', "company", ".com", " company", "new", 'motor', 'ag']

if suffix[-1] in suffix\_list:

name = name.replace(suffix[-1], '-big.svg')

name = name.replace(' ', '-')

name = name.replace('and', '-')

name = name.replace('&', '-')

name = name.replace("'", '')

result = f"{url}{name}"

# print(result)

return result

else:

name = name + '--big.svg'

name = name.replace(' ', '-')

name = name.replace('and', '-')

name = name.replace("'", '')

name = name.replace('&', '-')

result = f"{url}{name}"

# check if the logo exists:

status\_code = requests.get(url=result).status\_code

if status\_code == 200:

print(result)

return result

else:

return "https://placehold.co/600x400?text=Logo"

def request\_company\_filing(cik: str) -> json:

# Get a copy of the default headers that requests would use

#headers = requests.utils.default\_headers() # type: ignore

# headers.update({'User-Agent': 'My User Agent 1.0', }) # type: ignore

headers = {

'User-Agent': 'My User Agent 1.0',

'accept': 'application/json'

}

url = f"https://data.sec.gov/api/xbrl/companyfacts/{cik}.json"

response = requests.get(url, headers=headers)

return response.json()

if \_\_name\_\_ == "\_\_main\_\_":

#print(get\_all\_cik())

response = request\_company\_filing("CIK0000320193")

accounting\_norm\_list = [x for x in [\*response["facts"].keys()] if x not in ["srt", "invest", "dei"]]

print(accounting\_norm\_list)

import logging

import requests

import pandas as pd

import plotly.express as px

import plotly.graph\_objects as go

from plotly import plot

from functools import reduce

from FortyFour.Finance.utils import get\_all\_cik, request\_company\_filing

class Company:

# Initial initialization method

def \_\_init\_\_(self, ticker, cik=None):

self.cik = cik

if cik is None:

self.ticker = str.upper(ticker)

df = get\_all\_cik()

df.set\_index('ticker', inplace=True)

df = df[df.index == self.ticker]

self.cik = df["cik"].values[0]

self.response = request\_company\_filing(self.cik) # If facts is not available in the database (mongodb), then fetch from sec url

# for example us-gaap or ifrs etc...

accounting\_norm\_list = [x for x in [\*self.response["facts"].keys()] if x not in ["srt", "invest", "dei"]]

logging.info(f"Accounting Norms for {ticker}: {accounting\_norm\_list}")

self.GAAP\_NORM = accounting\_norm\_list[-1]

company\_name = self.response['entityName']

self.company\_name = company\_name

self.gaap\_List = self.response['facts'][self.GAAP\_NORM].keys()

def compounding\_annual\_growth\_rate(self, df, nb\_years, inline\_graph=False):

# Check if the data is a dataframe:

is\_dataframe = isinstance(df, pd.DataFrame)

if not isinstance(df, pd.DataFrame):

df = pd.DataFrame(df)

if len(df.index) >= (nb\_years + 1):

ending\_value = df.iloc[-1][0]

beginning\_value = df.iloc[-nb\_years - 1][0]

try:

result = round(

((ending\_value / beginning\_value) \*\* (1 / nb\_years) - 1), 3) \* 100

except Exception as e:

logging.warning('The function compounding\_annual\_growth\_rate() encounter an exception: ', e)

result = 0

fig = go.Figure(go.Indicator(

domain={'x': [0, 1], 'y': [0, 1]},

value=result,

mode="gauge+number+delta",

title={

'text': f"{self.company\_name}<br><sup> {nb\_years}-Years GAGR</sup>"},

delta={'reference': 10},

gauge={'axis': {'range': [-20, 20]},

'steps': [

{'range': [-20, 0], 'color': "pink"},

{'range': [0, 5], 'color': "gray"},

{'range': [5, 10], 'color': "lightcyan"},

{'range': [10, +20], 'color': "lightgreen"}

],

'threshold': {'line': {'color': "red", 'width': 4}, 'thickness': 0.75, 'value': 10}}))

if inline\_graph: # usefully in jupyter notebook

fig.show()

return fig

else:

fig = go.Figure(go.Indicator(

domain={'x': [0, 1], 'y': [0, 1]},

value=0,

mode="gauge+number+delta",

title={

'text': f"{self.company\_name}<br><sup>No Data for {nb\_years}-Years GAGR</sup>"},

delta={'reference': 0},

gauge={'axis': {'range': [0, 20]},

'steps': [

],

}))

return fig

def Financials(self, selected\_gaap: [], form\_type=None):

"""Select a list of gaap and the function will return a dataframe ot the selected gaaps"""

df\_list\_to\_merge = []

for gaap in selected\_gaap:

df = pd.DataFrame()

try: # try gaap norm

gaap\_unit = list(self.response['facts'][self.GAAP\_NORM][gaap]['units'].keys())[-1] # sometimes it can be multiple currencies so I select the last

df = pd.DataFrame.from\_records(self.response['facts'][self.GAAP\_NORM][gaap]["units"][gaap\_unit]).dropna()

except Exception as e: # try dei

# sometimes it can be multiple currencies so I select the last

gaap\_unit = list(self.response['facts']["dei"][gaap]['units'].keys())[-1]

df = pd.DataFrame.from\_records(self.response['facts']["dei"][gaap]["units"][gaap\_unit])

logging.error(f"An exception occurred while retrieving {gaap}")

df.rename(columns={'val': gaap, 'end': 'Date'}, errors="ignore", inplace=True)

# We want to drop the column only if it exists by using errors='ignore'

df.drop(['accn', 'fy', 'fp', 'form', 'filed', 'start'], errors='ignore', axis=1, inplace=True)

# if "frame column exists":

if "frame" in df.columns:

df = df.dropna(axis=0)

match form\_type:

case "10-Q":

df = df[df["frame"].str.contains('Q')]

df.drop(['frame'], axis=1, inplace=True)

df\_list\_to\_merge.append(df)

case "10-K":

df = df[~df["frame"].str.contains('Q')]

df.drop(["frame"], axis=1, inplace=True)

df\_list\_to\_merge.append(df)

case "20-F":

df = df[~df["frame"].str.contains('Q')]

df.drop(["frame"], axis=1, inplace=True)

df\_list\_to\_merge.append(df)

case "20-F/A":

df = df[~df["frame"].str.contains('Q')]

df.drop(["frame"], axis=1, inplace=True)

df\_list\_to\_merge.append(df)

case \_:

df.drop(["frame"], axis=1, inplace=True)

df\_list\_to\_merge.append(df)

pass

# df.set\_index('Date', inplace=True)

else:

pass

final\_df = reduce(lambda left, right: pd.merge(left, right, on=['Date'], how='outer'), df\_list\_to\_merge)

final\_df['Date'] = pd.to\_datetime(final\_df['Date'])

final\_df.set\_index('Date', inplace=True)

final\_df.sort\_values(by="Date", inplace=True)

return final\_df

def CommonStockSharesOutstanding(self, form\_type="10-K"):

# don't show de result of gaaplist on screen, just use the result

gaaplist = self.gaap\_List

synonyms = ['WeightedAverageNumberOfDilutedSharesOutstanding',

'CommonStockSharesIssued',

'EntityCommonStockSharesOutstanding',

'NumberOfSharesOutstanding',

'NumberOfSharesIssuedAndFullyPaid',

'WeightedAverageShares',

'CommonStockSharesOutstanding',

'NumberOfSharesIssuedAndFullyPaid']

list = [item for item in synonyms if item in gaaplist]

df = self.Financials(list, form\_type=form\_type)

df['NumberOfSharesOutstanding'] = reduce(lambda x, y: x.combine\_first(y), [df[col] for col in list])

return df

def MarketCap(self, form\_type="10-Q"):

df = self.Financials(["EntityPublicFloat"], form\_type="10-Q")

return df

def Equity(self, form\_type="10-K"):

gaaplist = self.gaap\_List

synonyms = ['StockholdersEquity', 'Equity', 'EquityAttributableToOwnersOfParent', 'StockholdersEquityIncludingPortionAttributableToNoncontrollingInterest']

list = [item for item in synonyms if item in gaaplist]

df = self.Financials(list, form\_type=form\_type)

df['Equity'] = reduce(lambda x, y: x.combine\_first(y), [df[col] for col in list])

return df

def AssetsLiabilitiesEquity(self, form\_type="10-Q", show\_graph=False):

df = pd.DataFrame()

df['Assets'] = self.Financials(["Assets"], form\_type=form\_type)["Assets"]

df["Equity"] = self.Equity(form\_type=form\_type)["Equity"]

df["Liabilities"] = df['Assets'] - df["Equity"]

df['LiabilitiesToAssetsRatio'] = df["Liabilities"] / df['Assets']

fig = px.area(df, x=df.index, y=['LiabilitiesToAssetsRatio'],

title=f"{self.company\_name}<br><sup>Liabilities To Assets Ratio</sup>",

line\_shape="spline",

template="seaborn",

)

fig.update\_xaxes(showline=True, linewidth=1.5, linecolor='black')

fig.update\_yaxes(showline=True, linewidth=1.5, linecolor='black')

fig.update\_layout(margin=dict(l=0, r=0),

# width=1200,

# height=500,

yaxis\_title=None, xaxis\_title=None)

fig.update\_layout(legend=dict(

orientation="h",

yanchor="bottom",

y=1.02,

xanchor="right",

x=1,

))

fig.update\_traces(line\_color="darksalmon")

# fig.update\_layout({'plot\_bgcolor': 'rgba(0,0,0,0)','paper\_bgcolor': 'rgba(0,0,0,0)'})

if show\_graph == True:

fig.show()

return df, fig

def Capex(self, form\_type="10-K"):

list = []

# don't show de result of gaaplist on screen, just use the result

gaaplist = self.gaap\_List

for item in ['PaymentsToAcquirePropertyPlantAndEquipment',

"PaymentsToAcquireOtherPropertyPlantAndEquipment",

'PurchaseOfPropertyPlantAndEquipmentClassifiedAsInvestingActivities',

"PaymentsToAcquireRealEstateHeldForInvestment",

"PaymentsToDevelopRealEstateAssets",

"PaymentsForCapitalImprovements",

'PaymentsToAcquireAndDevelopRealEstate',

"PaymentsToAcquireRealEstate",

"PaymentsToAcquireCommercialRealEstate",

"PaymentsToAcquireProductiveAssets",

"PurchaseOfPropertyPlantAndEquipmentIntangibleAssetsOtherThanGoodwillInvestmentPropertyAndOtherNoncurrentAssets",

"PurchaseOfPropertyPlantAndEquipmentAndIntangibleAssets",

'PurchasesOfPropertyAndEquipmentAndIntangibleAssets',

"PurchasesOfPropertyAndEquipmentAndIntangibleAssets"]:

if item in gaaplist:

list.append(item)

df = self.Financials(list, form\_type=form\_type)

# This line remove duplicates horizontally across columns

df = df.apply(lambda x: pd.Series(x.unique()), axis=1)

df['CAPEX'] = df.sum(axis=1)

return df

def PropertyPlantAndEquipmentGross(self, form\_type="10-Q"):

# don't show de result of gaaplist on screen, just use the result

gaaplist = self.gaap\_List

synonyms = ["PropertyPlantAndEquipmentGross", 'PropertyPlantAndEquipment', "RealEstateInvestmentPropertyAtCost", "GrossInvestmentInRealEstateAssets"]

list = [item for item in synonyms if item in gaaplist]

df = self.Financials(list, form\_type=form\_type)

df['PropertyPlantAndEquipmentGross'] = reduce(lambda x, y: x.combine\_first(y), [df[col] for col in list])

return df

def PropertyPlantAndEquipmentNet(self, form\_type="10-Q"):

# don't show de result of gaaplist on screen, just use the result

gaaplist = self.gaap\_List

synonyms = ["PropertyPlantAndEquipmentNet", 'RealEstateInvestmentPropertyNet', 'NetInvestmentInRealEstateAssets', 'PropertyPlantAndEquipment']

list = [item for item in synonyms if item in gaaplist]

df = self.Financials(list, form\_type=form\_type)

df['PropertyPlantAndEquipmentNet'] = reduce(lambda x, y: x.combine\_first(y), [df[col] for col in list])

return df

def PropertyPlantAndEquipmentGrossAndNet(self, form\_type="10-K"):

list = []

gaaplist = self.gaap\_List

for item in ["PropertyPlantAndEquipmentGross", 'PropertyPlantAndEquipment', "PropertyPlantAndEquipmentNet",

"RealEstateInvestmentPropertyAtCost", "RealEstateInvestmentPropertyNet"]:

if item in gaaplist:

list.append(item)

df = self.Financials(list, form\_type=form\_type)

df['Ratio'] = df[list[-1]] / df[list[0]]

df['Ratio'] = df['Ratio'].round(2)

df.ffill(inplace=True)

return df

def Revenues(self, form\_type='10-K', show\_graph=False):

gaaplist = self.gaap\_List

synonyms = ['RevenueFromContractWithCustomerExcludingAssessedTax', 'Revenues',

'SalesRevenueNet', 'Revenue', 'RevenueFromSaleOfGoods', 'RevenueFromContractsWithCustomers',

'NoninterestIncome']

list = [item for item in synonyms if item in gaaplist]

df = self.Financials(list, form\_type=form\_type)

df['Revenues'] = reduce(lambda x, y: x.combine\_first(y), [df[col] for col in list])

fig = px.bar(df, x=df.index, y=['Revenues'],

title=f"{self.company\_name}<br><sup>Revenues</sup>",

# line\_shape="spline",

text\_auto=True,

template="seaborn"

)

fig.update\_traces(textangle=0, textposition="outside", cliponaxis=False)

fig.update\_xaxes(showline=True, linewidth=1.5, linecolor='black')

fig.update\_yaxes(showline=True, linewidth=1.5, linecolor='black')

fig.update\_layout(margin=dict(l=0, r=0),

# width=1200,

height=500,

yaxis\_title=None, xaxis\_title=None)

fig.update\_layout(legend=dict(

orientation="h",

yanchor="bottom",

y=1.02,

xanchor="right",

x=1,

))

# fig.update\_layout({'plot\_bgcolor': 'rgba(0,0,0,0)','paper\_bgcolor': 'rgba(0,0,0,0)'})

if show\_graph == True:

fig.show()

return df['Revenues'], fig

def NetIncomeLoss(self, form\_type='10-K'):

# don't show de result of gaaplist on screen, just use the result

gaaplist = self.gaap\_List

synonyms = ["NetIncomeLoss", 'ProfitLossAttributableToOwnersOfParent', "NetIncomeLossAvailableToCommonStockholdersBasic"]

list = [item for item in synonyms if item in gaaplist]

df = self.Financials(list, form\_type=form\_type)

df['NetIncome'] = reduce(lambda x, y: x.combine\_first(y), [df[col] for col in list])

return df

def OperatingIncomeLoss(self, form\_type='10-K'): # should be same as EBIT

gaaplist = self.gaap\_List

synonyms = ["OperatingIncomeLoss", "IncomeLossFromOperationsBeforeIncomeTaxExpenseBenefit",

"IncomeLossFromContinuingOperationsBeforeIncomeTaxesExtraordinaryItemsNoncontrollingInterest",

"IncomeLossFromContinuingOperationsBeforeIncomeTaxesExtraordinaryItemsNoncontrollingInterest",

"IncomeLossFromContinuingOperationsBeforeIncomeTaxesMinorityInterestAndIncomeLossFromEquityMethodInvestments"]

list = [item for item in synonyms if item in gaaplist]

df = self.Financials(list, form\_type=form\_type)

df['OperatingIncome'] = reduce(lambda x, y: x.combine\_first(y), [df[col] for col in list])

return df

def ProfitMargin(self, form\_type='10-K', show\_graph=False):

df = pd.DataFrame()

df["Revenues"], fig = self.Revenues(form\_type=form\_type, show\_graph=False)

df["NetIncome"] = self.NetIncomeLoss(form\_type=form\_type)["NetIncome"]

df["ProfitMargin"] = df["NetIncome"] / df["Revenues"]

fig = px.area(df, x=df.index, y=['ProfitMargin'],

title=f"{self.company\_name}<br><sup>Net Profit Margin</sup>",

line\_shape="vh",

template="seaborn"

)

fig.update\_xaxes(showline=True, linewidth=1.5, linecolor='black')

fig.update\_yaxes(showline=True, linewidth=1.5, linecolor='black')

fig.update\_layout(margin=dict(l=0, r=0),

# width=1200,

height=500,

yaxis\_title=None, xaxis\_title=None)

fig.update\_layout(legend=dict(

orientation="h",

yanchor="bottom",

y=1.02,

xanchor="right",

x=1,

))

# fig.update\_layout({'plot\_bgcolor': 'rgba(0,0,0,0)','paper\_bgcolor': 'rgba(0,0,0,0)'})

# ------------------------------------------------------------------ ADD THE 44SCIENTIFICS LOGO

fig.add\_layout\_image(

dict(

source="/Users/cheikhcamara/Documents/GitHub/3SIGMA/44 Scientifics logo.jpeg",

xref="paper", yref="paper",

x=1, y=1.05,

sizex=0.2, sizey=0.2,

xanchor="right", yanchor="bottom",

)

)

if show\_graph:

fig.show()

return df['ProfitMargin'], fig

def PriceToEarningsRatio(self, form\_type="10-K"):

df = pd.DataFrame()

df["NetIncome"] = self.NetIncomeLoss(form\_type=form\_type)["NetIncome"]

df["Equity"] = self.Equity(form\_type=form\_type)["Equity"]

# df = df.resample('Y').agg({'NetIncome': 'sum', 'Equity': 'last'}).head()

df["PE"] = df["Equity"] / df["NetIncome"]

return df

def EPS(self, form\_type="10-K", show\_graph=False):

gaaplist = self.gaap\_List

synonyms = ["EarningsPerShareDiluted"]

list = [item for item in synonyms if item in gaaplist]

df = pd.DataFrame()

if len(list) >= 1:

df = self.Financials(list, form\_type=form\_type)

# This line removes duplicates horizontally across columns

df = df.apply(lambda x: pd.Series(x.unique()), axis=1)

df['EarningsPerShare'] = df.sum(axis=1)

else:

df = self.NetIncomeLoss(form\_type=form\_type)

df["SharesOutstanding"] = self.CommonStockSharesOutstanding(form\_type=form\_type)

df["EarningsPerShare"] = df["NetIncome"] / df["SharesOutstanding"]

df["EarningsPerShare"].fillna(method='ffill', inplace=True)

df["EarningsPerShare"] = df["EarningsPerShare"].round(2)

fig = px.area(df, x=df.index, y=['EarningsPerShare'],

title=f"{self.company\_name}<br><sup>Diluted Earning Per Share (EPS)</sup>",

line\_shape="vh",

template="seaborn"

)

if df['EarningsPerShare'].min() <= 0:

fig.add\_shape( # add a horizontal "target" line

type="line", line\_color="salmon", line\_width=2, opacity=1, line\_dash="dot",

x0=0, x1=1, xref="paper", y0=0.4, y1=0.4, yref="y")

fig.update\_xaxes(showline=True, linewidth=1.5, linecolor='black')

fig.update\_yaxes(showline=True, linewidth=1.5, linecolor='black')

fig.update\_layout(margin=dict(l=0, r=0),

# width=1200,

height=500,

yaxis\_title=None, xaxis\_title=None)

fig.update\_layout(legend=dict(

orientation="h",

yanchor="bottom",

y=1.02,

xanchor="right",

x=1,

))

# fig.update\_layout({'plot\_bgcolor': 'rgba(0,0,0,0)','paper\_bgcolor': 'rgba(0,0,0,0)'})

# ------------------------------------------------------------------ ADD THE 44SCIENTIFICS LOGO

fig.add\_layout\_image(

dict(

source="/Users/cheikhcamara/Documents/GitHub/3SIGMA/44 Scientifics logo.jpeg",

xref="paper", yref="paper",

x=1, y=1.05,

sizex=0.2, sizey=0.2,

xanchor="right", yanchor="bottom",

)

)

if show\_graph == True:

fig.show()

return df['EarningsPerShare'], fig

def DividendPerShare(self, form\_type="10-K", show\_graph=False):

# don't show de result of gaaplist on screen, just use the result

gaaplist = self.gaap\_List

synonyms = ["CommonStockDividendsPerShareCashPaid", "CommonStockDividendsPerShareDeclared", "DividendsRecognisedAsDistributionsToOwnersPerShare"]

mlist = [item for item in synonyms if item in gaaplist]

df = self.Financials(mlist, form\_type=form\_type)

df['DividendPerShare'] = reduce(lambda x, y: x.combine\_first(y), [df[col] for col in mlist])

return df

def PaymentsOfDividends(self, form\_type="10-K", show\_graph=False):

# don't show de result of gaaplist on screen, just use the result

gaaplist = self.gaap\_List

synonyms = ["PaymentsOfDividendsCommonStock", "PaymentsOfDividends", 'PaymentsOfOrdinaryDividends',

'DividendsPaidToEquityHoldersOfParentClassifiedAsFinancingActivities',

'DividendsPaidClassifiedAsFinancingActivities']

list = [item for item in synonyms if item in gaaplist]

df = self.Financials(list, form\_type=form\_type)

df['PaymentsOfDividends'] = reduce(lambda x, y: x.combine\_first(y), [df[col] for col in list])

return df

def DividendPayoutRatio(self, form\_type="10-K", show\_graph=False):

df = self.DividendPerShare(form\_type=form\_type, show\_graph=False)

eps\_df, fig = self.EPS(form\_type=form\_type)

eps = eps\_df.to\_frame()

df['EarningsPerShare'] = eps['EarningsPerShare']

df['DividendPayoutRatio'] = df['DividendPerShare'] / df['EarningsPerShare']

template = "seaborn"

fig = px.area(df, x=df.index, y=['DividendPayoutRatio'],

title=f"{self.company\_name}<br><sup>Dividend Payout Ratio</sup>",

line\_shape="hv", # "spline",

# width=1200,

template=template

)

fig.update\_xaxes(showline=True, linewidth=1.5, linecolor='black')

fig.update\_yaxes(showline=True, linewidth=1.5, linecolor='black')

fig.update\_layout(margin=dict(l=0, r=0),

yaxis\_title=None, xaxis\_title=None)

fig.update\_layout(legend=dict(

orientation="h",

yanchor="bottom",

y=1.02, x=1,

xanchor="right",

))

# ------------------------------------------------------------------ ADD THE 44SCIENTIFICS LOGO

fig.add\_layout\_image(

dict(

source="/Users/cheikhcamara/Documents/GitHub/3SIGMA/44 Scientifics logo.jpeg",

xref="paper", yref="paper",

x=0, y=1.05,

sizex=0.2, sizey=0.2,

xanchor="left", yanchor="bottom",

)

)

if show\_graph == True:

fig.show()

return df[["EarningsPerShare", "DividendPerShare", "DividendPayoutRatio"]]

def DividendYield(self, form\_type="10-K", show\_graph=False):

# don't show de result of gaaplist on screen, just use the result

gaaplist = self.gaap\_List

synonyms = [

"ShareBasedCompensationArrangementByShareBasedPaymentAwardFairValueAssumptionsExpectedDividendRate"]

list = [item for item in synonyms if item in gaaplist]

df = self.Financials(list, form\_type=form\_type)

df['DividendYield'] = reduce(lambda x, y: x.combine\_first(y), [df[col] for col in list])

return df

# def NextDividendDate(self):

# compani = reader.get\_quote\_yahoo(self.ticker)

# next\_dividend\_date = datetime.fromtimestamp(

# compani["dividendDate"][0] if "dividendDate" in compani.columns else 0)

# print(next\_dividend\_date)

# day = next\_dividend\_date.day

# month = next\_dividend\_date.month

# year = next\_dividend\_date.year

#

# return day, month, year

def PropertyPlantEQuipmentRatio(self, form\_type="10-Q", show\_graph=False):

# create an Empty DataFrame object

df = pd.DataFrame()

df["PropertyPlantAndEquipmentGross"] = self.PropertyPlantAndEquipmentGross(form\_type=form\_type)["PropertyPlantAndEquipmentsGross"]

df["PropertyPlantAndEquipmentNet"] = self.PropertyPlantAndEquipmentNet(form\_type=form\_type)["PropertyPlantAndEquipmentsNet"]

df['Ratio'] = df["PropertyPlantAndEquipmentNet"] / df["PropertyPlantAndEquipmentGross"]

template = "seaborn"

fig = px.line(df, x=df.index, y=['Ratio'],

title=f"{self.company\_name}<br><sup>Current State of the company's fixed assets</sup>",

line\_shape="spline",

# width=1200,

template=template

)

if df['Ratio'].min() <= 0.4:

fig.add\_shape( # add a horizontal "target" line

type="line", line\_color="salmon", line\_width=2, opacity=1, line\_dash="dot",

x0=0, x1=1, xref="paper", y0=0.4, y1=0.4, yref="y",

)

fig.update\_xaxes(showline=True, linewidth=1.5, linecolor='black')

fig.update\_yaxes(showline=True, linewidth=1.5, linecolor='black')

fig.update\_layout(margin=dict(l=0, r=0), yaxis\_title=None, xaxis\_title=None)

fig.update\_layout(legend=dict(

orientation="h",

yanchor="bottom",

y=1.02,

xanchor="right",

x=1,

))

if show\_graph == True:

fig.show()

return df, fig

def CashFromOperatingActivities(self, form\_type="10-K"):

# don't show de result of gaaplist on screen, just use the result

gaaplist = self.gaap\_List

synonyms = ['NetCashProvidedByUsedInOperatingActivitiesContinuingOperations',

'CashFlowsFromUsedInOperatingActivities', 'NetCashProvidedByUsedInOperatingActivities']

list = [item for item in synonyms if item in gaaplist]

df = self.Financials(list, form\_type=form\_type)

df['CashFromOperatingActivities'] = reduce(lambda x, y: x.combine\_first(y), [df[col] for col in list])

return df

def CostOfGoodsAndServicesSold(self, form\_type="10-K"):

gaaplist = self.gaap\_List

synonyms = ['CostOfGoodsAndServicesSold', 'CostOfRevenue']

list = [item for item in synonyms if item in gaaplist]

df = self.Financials(list, form\_type=form\_type)

df['CostOfGoodsAndServicesSold'] = reduce(lambda x, y: x.combine\_first(y), [df[col] for col in list])

return df

def CashCashEquivalents(self, form\_type="10-K", show\_graph=False):

# don't show de result of gaaplist on screen, just use the result

gaaplist = self.gaap\_List

synonyms = ['CashCashEquivalentsRestrictedCashAndRestrictedCashEquivalents',

'CashCashEquivalentsRestrictedCashAndRestrictedCashEquivalentsIncludingDisposalGroupAndDiscontinuedOperations',

'CashAndCashEquivalents']

list = [item for item in synonyms if item in gaaplist]

df = self.Financials(list, form\_type=form\_type)

df['CashCashEquivalents'] = reduce(lambda x, y: x.combine\_first(y), [df[col] for col in list])

template = "seaborn"

fig = px.bar(df, x=df.index, y=['CashCashEquivalents'],

title=f"{self.company\_name}<br><sup>Cash & Cash Equivalents</sup>",

# line\_shape="spline",

# width=1200,

template=template

)

fig.update\_xaxes(showline=True, linewidth=1.5, linecolor='black')

fig.update\_yaxes(showline=True, linewidth=1.5, linecolor='black')

fig.update\_layout(margin=dict(l=0, r=0),

yaxis\_title=None, xaxis\_title=None)

fig.update\_layout(legend=dict(

orientation="h",

yanchor="bottom",

y=1.02,

xanchor="right",

x=1,

))

if show\_graph == True:

fig.show()

return df, fig

def MarketableSecurities(self, form\_type="10-K", show\_graph=False):

# don't show de result of gaaplist on screen, just use the result

gaaplist = self.gaap\_List

synonyms = ['MarketableSecuritiesCurrent', 'ShortTermInvestments']

list = [item for item in synonyms if item in gaaplist]

df = self.Financials(list, form\_type=form\_type)

df['MarketableSecurities'] = reduce(lambda x, y: x.combine\_first(y), [df[col] for col in list])

df1, figu = self.CashCashEquivalents(form\_type=form\_type, show\_graph=False)

df['Cash'] = df1['CashCashEquivalents']

df["Cash and MarketableSecurities"] = df['Cash'] + df['MarketableAssets']

result = pd.concat([df1, df])

template = "seaborn"

fig = px.bar(result, x=result.index, y=['MarketableAssets', 'CashCashEquivalents'],

title=f"{self.company\_name}<br><sup>Marketable Securities and Cash</sup>",

# width=1200,

# text\_auto='.2f',

text\_auto=True,

# text="nation",

template=template

)

fig.update\_traces(textangle=0,

textposition="outside", cliponaxis=False)

fig.update\_xaxes(showline=True, linewidth=1.5, linecolor='black')

fig.update\_yaxes(showline=True, linewidth=1.5, linecolor='black')

fig.update\_layout(margin=dict(l=0, r=0),

yaxis\_title=None, xaxis\_title=None)

fig.update\_layout(legend=dict(

orientation="h",

yanchor="bottom",

y=1.02,

xanchor="right",

x=1,

))

if show\_graph == True:

fig.show()

return result, fig

def CashFlowFromFinancingActivities(self, form\_type='10-K'):

# don't show de result of gaaplist on screen, just use the result

gaaplist = self.gaap\_List

synonyms = ['NetCashProvidedByUsedInFinancingActivitiesContinuingOperations',

'CashFlowsFromUsedInFinancingActivities', 'NetCashProvidedByUsedInFinancingActivities']

list = [item for item in synonyms if item in gaaplist]

df = self.Financials(list, form\_type=form\_type)

df['CashFlowFromFinancingActivities'] = reduce(lambda x, y: x.combine\_first(y), [df[col] for col in list])

return df

def CashFlowFromInvestingActivities(self, form\_type='10-K'):

# don't show de result of gaaplist on screen, just use the result

gaaplist = self.gaap\_List

synonyms = ['NetCashProvidedByUsedInInvestingActivitiesContinuingOperations',

'CashFlowsFromUsedInInvestingActivities', 'NetCashProvidedByUsedInInvestingActivities']

list = [item for item in synonyms if item in gaaplist]

df = self.Financials(list, form\_type=form\_type)

df['CashFlowFromInvestingActivities'] = reduce(lambda x, y: x.combine\_first(y), [df[col] for col in list])

return df

def OperatingCash\_VS\_Capex(self, form\_type="10-K", show\_graph=False):

df = self.CashFromOperatingActivities(form\_type=form\_type)

capex = self.Capex(form\_type=form\_type)

df['CAPEX'] = capex['CAPEX']

# ===================================================GRAPH

fig = go.Figure()

fig.add\_trace(go.Line(x=df.index, y=df["CashFromOperatingActivities"],

name="Cash From Operating Activities", yaxis='y',

line\_shape="hv",

)

)

fig.add\_trace(go.Line(x=df.index, y=df["CAPEX"],

name="Capex", yaxis="y2",

line\_shape="hv",

)

)

# Create axis objects

fig.update\_layout(

autosize=False,

width=1200,

# height=500,

yaxis=dict(

title="Cash From Operating Activities",

titlefont=dict(color="#1f77b4"),

tickfont=dict(color="#1f77b4")),

# create 2nd y axis

yaxis2=dict(title="Capital Expenditure (CAPEX)", overlaying="y", side="right", position=1))

fig.update\_xaxes(showline=True, linewidth=1.5, linecolor='black')

fig.update\_yaxes(showline=True, linewidth=1.5, linecolor='black')

fig.update\_layout(

template="seaborn",

title=f"{self.company\_name}<br><sup>Capex vs. Cash From Operating Activities</sup>",

legend=dict(

orientation="h",

yanchor="bottom",

y=1.02,

xanchor="right",

x=1

))

if show\_graph == True:

fig.show()

return df, fig

def IncomeTaxRate(self, form\_type="10-K"):

gaaplist = self.gaap\_List

synonyms = ['EffectiveIncomeTaxRateContinuingOperations']

list = [item for item in synonyms if item in gaaplist]

df = self.Financials(list, form\_type=form\_type)

df['IncomeTaxRate'] = reduce(lambda x, y: x.combine\_first(y), [df[col] for col in list])

return df

def DepreciationAndAmortization(self, form\_type="10-K"):

# don't show de result of gaaplist on screen, just use the result

gaaplist = self.gaap\_List

synonyms = ['DepreciationAndAmortization', 'DepreciationAndAmortizationExcludingNuclearFuel',

"DepreciationDepletionAndAmortization"]

list = [item for item in synonyms if item in gaaplist]

df = self.Financials(list, form\_type=form\_type)

df['DepreciationAndAmortization'] = reduce(lambda x, y: x.combine\_first(y), [df[col] for col in list])

return df

def DebtInterestRate(self, form\_type="10-K"):

list = []

# don't show de result of gaaplist on screen, just use the result

gaaplist = self.gaap\_List

synonyms = ["DebtWeightedAverageInterestRate", "LongtermDebtWeightedAverageInterestRate",

'ShortTermDebtWeightedAverageInterestRate']

list = [item for item in synonyms if item in gaaplist]

df = self.Financials(list, form\_type=form\_type)

df['DebtInterestRate'] = reduce(lambda x, y: x.combine\_first(y), [df[col] for col in list])

return df

def InterestExpense(self, form\_type="10-K"):

list = []

# don't show de result of gaaplist on screen, just use the result

gaaplist = self.gaap\_List

synonyms = ["InterestExpense", "InterestExpenseNetOfHedgeIneffectiveness"]

list = [item for item in synonyms if item in gaaplist]

df = self.Financials(list, form\_type=form\_type)

df['InterestExpense'] = reduce(lambda x, y: x.combine\_first(y), [df[col] for col in list])

return df

def IncomeTaxes(self, form\_type='10-K'):

list = []

# don't show de result of gaaplist on screen, just use the result

gaaplist = self.gaap\_List

# IncomeTaxExpenseBenefit

synonyms = ['IncomeTaxExpenseBenefit',

# "IncomeTaxesPaidNet"

]

list = [item for item in synonyms if item in gaaplist]

df = self.Financials(list, form\_type=form\_type)

df['IncomeTaxes'] = reduce(lambda x, y: x.combine\_first(y), [df[col] for col in list])

return df

def CapitalEmployed(self, form\_type="10-Q"):

gaaplist = self.gaap\_List

synonyms = ['Assets', 'LiabilitiesCurrent']

list = [item for item in synonyms if item in gaaplist]

df = self.Financials(list, form\_type=form\_type)

df["Capital Employed"] = df[df.columns[0]] - df[df.columns[-1]]

return df

def DebtRepayment(self, form\_type="10-K"):

# don't show de result of gaaplist on screen, just use the result

gaaplist = self.gaap\_List

synonyms = ["RepaymentsOfLongTermDebt"]

list = [item for item in synonyms if item in gaaplist]

df = self.Financials(list, form\_type=form\_type)

df['DebtRepayment'] = reduce(lambda x, y: x.combine\_first(y), [df[col] for col in list])

return df

def LongTermDebt(self, form\_type="10-K", show\_graph=False):

# don't show de result of gaaplist on screen, just use the result

gaaplist = self.gaap\_List

synonyms = ['NoncurrentFinancialLiabilities', 'LongtermBorrowings', 'LongTermDebtNoncurrent', 'LongTermDebt',

"LongTermDebtAndCapitalLeaseObligations", "OtherLiabilitiesNoncurrent"]

# ["DebtInstrumentCarryingAmount"]

list = [item for item in synonyms if item in gaaplist]

df = self.Financials(list, form\_type=form\_type)

df['LongTermDebt'] = reduce(lambda x, y: x.combine\_first(y), [df[col] for col in list])

fig = px.area(df, x=df.index, y=['TotalLongTermDebt'],

title=f"{self.company\_name}<br><sup>Long-Term Debt</sup>",

line\_shape="hv",

# width=1200,

template="seaborn"

)

fig.update\_xaxes(showline=True, linewidth=1.5, linecolor='black')

fig.update\_yaxes(showline=True, linewidth=1.5, linecolor='black')

fig.update\_layout(margin=dict(l=0, r=0),

yaxis\_title=None, xaxis\_title=None)

fig.update\_layout(legend=dict(

orientation="h",

yanchor="bottom",

y=1.02,

xanchor="right",

x=1,

))

if show\_graph == True:

fig.show()

return df, fig

def ShortTermDebt(self, form\_type="10-K", show\_graph=False):

gaaplist = self.gaap\_List

synonyms = ["TradeAndOtherCurrentPayables", "AccountsPayableCurrent", "EmployeeRelatedLiabilitiesCurrent",

"CurrentTaxLiabilitiesCurrent", "AccruedIncomeTaxesCurrent", "OtherShorttermProvisions",

"LiabilitiesIncludedInDisposalGroupsClassifiedAsHeldForSale",

"ContractWithCustomerLiabilityCurrent", "AccountsPayableAndAccruedLiabilitiesCurrent",

"AccruedLiabilitiesCurrent", "AccruedRebatesReturnsAndPromotions", "OtherLiabilitiesCurrent"]

list = [item for item in synonyms if item in gaaplist]

df = self.Financials(list, form\_type=form\_type)

df = df.apply(lambda x: pd.Series(x.unique()), axis=1)

df["total\_to\_subtract"] = df.sum(axis=1)

df2 = self.Financials(["LiabilitiesCurrent"], form\_type=form\_type)

df2["ShortTermDebt"] = df2["LiabilitiesCurrent"] - df["total\_to\_subtract"]

return df2

def TotalDebt(self, form\_type="10-Q"):

TotalDebt = self.ShortTermDebt(form\_type=form\_type, show\_graph=False)

TotalDebt["LongTermDebt"], fig = self.LongTermDebt(

form\_type=form\_type, show\_graph=False)

TotalDebt["TotalDebt"] = TotalDebt["LongTermDebt"] + TotalDebt["ShortTermDebt"]

return TotalDebt[["ShortTermDebt", "LongTermDebt", "TotalDebt"]]

def ROE(self, form\_type="10-Q"):

match form\_type:

case "10-Q":

df = self.AssetsLiabilitiesEquity(form\_type=form\_type)

df['NetIncome'] = self.NetIncomeLoss(form\_type=form\_type)['NetIncome']

df['ROE'] = df['NetIncome'] / df['Equity']

return df

case "10-K":

df = self.AssetsLiabilitiesEquity(form\_type="10-Q")

df['NetIncome'] = self.NetIncomeLoss(form\_type=form\_type)['NetIncome']

df['ROE'] = df['NetIncome'] / df['Equity']

df.dropna(axis=0, inplace=True)

return df

case \_:

df = self.AssetsLiabilitiesEquity(form\_type="10-Q")

df['NetIncome'] = self.NetIncomeLoss(form\_type=form\_type)['NetIncome']

df['ROE'] = df['NetIncome'] / df['Equity']

df.dropna(axis=0, inplace=True)

return df

def ROCE(self, form\_type='10-Q', show\_graph=False):

ebit = self.OperatingIncomeLoss(form\_type="10-K")

capitalEmployed = self.CapitalEmployed(form\_type="10-Q")

df = pd.concat([capitalEmployed["Capital Employed"], ebit["OperatingIncome"]], axis=1)

df.dropna(axis=0, inplace=True)

df['ROCE'] = df['OperatingIncome'] / df['Capital Employed']

template = "seaborn"

fig = px.area(df, x=df.index, y=['ROCE'],

title=f"{self.company\_name}<br><sup>Return on Capital Employed (ROCE)</sup>",

line\_shape="vh",

template=template

)

fig.update\_xaxes(showline=True, linewidth=1.5, linecolor='black')

fig.update\_yaxes(showline=True, linewidth=1.5, linecolor='black')

fig.update\_layout(margin=dict(l=0, r=0), yaxis\_title=None, xaxis\_title=None)

fig.update\_layout(legend=dict(

orientation="h",

yanchor="bottom",

y=1.02,

xanchor="right",

x=1,

))

if show\_graph == True:

fig.show()

return df, fig

def EBITDA(self, form\_type="10-Q", show\_graph=False):

df = self.OperatingIncomeLoss(form\_type=form\_type)

df2 = self.DepreciationAndAmortization(form\_type=form\_type)

df["Amortization"] = df2["AmortizationAndDepreciation"]

df["EBITDA"] = df["OperatingIncomeLoss"] + df["Amortization"]

# PLOTLY FIGURE

template = "seaborn"

fig = px.bar(df, x=df.index, y=['EBITDA'],

title=f"{self.company\_name}<br><sup>EBITDA</sup>",

# line\_shape="spline",

template=template,

# width=1200

)

fig.update\_xaxes(showline=True, linewidth=1.5, linecolor='black')

fig.update\_yaxes(showline=True, linewidth=1.5, linecolor='black')

fig.update\_layout(margin=dict(l=0, r=0),

yaxis\_title=None, xaxis\_title=None)

fig.update\_layout(legend=dict(

orientation="h",

yanchor="bottom",

y=1.02,

xanchor="right",

x=1,

))

if show\_graph == True:

fig.show()

figplot = plot(fig, output\_type="div")

return df, figplot

def EBIT(self, form\_type="10-K"):

net\_income = self.NetIncomeLoss(form\_type=form\_type)

interest\_expense = self.InterestExpense(form\_type=form\_type)

income\_tax = self.IncomeTaxes(form\_type=form\_type)

df = pd.concat([net\_income["NetIncome"], interest\_expense["InterestExpense"], income\_tax["IncomeTaxes"]], axis=1)

df["EBIT"] = df.sum(axis=1)

return df

def WorkingCapital(self, form\_type="10-Q", show\_graph=False):

list = []

gaaplist = self.gaap\_List

for item in ["AssetsCurrent", 'LiabilitiesCurrent']:

if item in gaaplist:

list.append(item)

df = self.Financials(list, form\_type=form\_type)

# if form\_type == "10-K":

# df = df.resample('BY').last()

df["Working Capital"] = df["AssetsCurrent"] - df["LiabilitiesCurrent"]

# PLOTLY FIGURE

template = "seaborn"

fig = px.line(df, x=df.index, y=['Working Capital'],

title=f"{self.company\_name}<br><sup>Working Capital - Quarterely</sup>",

line\_shape="spline",

template=template,

width=1200

)

fig.update\_xaxes(showline=True, linewidth=1.5, linecolor='black')

fig.update\_yaxes(showline=True, linewidth=1.5, linecolor='black')

fig.update\_layout(margin=dict(l=0, r=0),

yaxis\_title=None, xaxis\_title=None)

fig.update\_layout(legend=dict(

orientation="h",

yanchor="bottom",

y=1.02,

xanchor="right",

x=1,

))

if show\_graph == True:

fig.show()

figplot = plot(fig, output\_type="div")

return df, figplot

def FreeCashFlow(self, form\_type='10-K', show\_graph=False):

df = self.CashFromOperatingActivities(form\_type=form\_type)

df1 = self.Capex(form\_type=form\_type)

df["CAPEX"] = df1['CAPEX']

df["FreeCashFlow"] = df["CashFromOperatingActivities"] - df["CAPEX"]

df['NumberOfSharesOutstanding'] = self.CommonStockSharesOutstanding(form\_type=form\_type)['NumberOfSharesOutstanding']

df['NumberOfSharesOutstanding'].ffill(inplace=True)

df["FreeCashFlowPerShare"] = df["FreeCashFlow"] / df['NumberOfSharesOutstanding']

return df

def EntrepriseValue(self, form\_type='10-K', show\_graph=False):

list = []

gaaplist = self.gaap\_List

for item in ["DepreciationDepletionAndAmortization", 'InterestExpense', 'NetIncomeLoss',

'IncomeTaxExpenseBenefit', 'InterestIncomeExpenseNet']:

if item in gaaplist:

list.append(item)

df = self.Financials(list, form\_type=form\_type)

df['EBITDA'] = df.sum(axis=1)

return df

#sep api - adam getbags

# -\*- coding: utf-8 -\*-

"""

SEC Filing Scraper

@author: AdamGetbags

"""

# import modules

import requests

import pandas as pd

# create request header

headers = {'User-Agent': "email@address.com"}

# get all companies data

companyTickers = requests.get(

"https://www.sec.gov/files/company\_tickers.json",

headers=headers

)

# review response / keys

print(companyTickers.json().keys())

# format response to dictionary and get first key/value

firstEntry = companyTickers.json()['0']

# parse CIK // without leading zeros

directCik = companyTickers.json()['0']['cik\_str']

# dictionary to dataframe

companyData = pd.DataFrame.from\_dict(companyTickers.json(),

orient='index')

# add leading zeros to CIK

companyData['cik\_str'] = companyData['cik\_str'].astype(

str).str.zfill(10)

# review data

print(companyData[:1])

cik = companyData[0:1].cik\_str[0]

# get company specific filing metadata

filingMetadata = requests.get(

f'https://data.sec.gov/submissions/CIK{cik}.json',

headers=headers

)

# review json

print(filingMetadata.json().keys())

filingMetadata.json()['filings']

filingMetadata.json()['filings'].keys()

filingMetadata.json()['filings']['recent']

filingMetadata.json()['filings']['recent'].keys()

# dictionary to dataframe

allForms = pd.DataFrame.from\_dict(

filingMetadata.json()['filings']['recent']

)

# review columns

allForms.columns

allForms[['accessionNumber', 'reportDate', 'form']].head(50)

# 10-Q metadata

allForms.iloc[11]

# get company facts data

companyFacts = requests.get(

f'https://data.sec.gov/api/xbrl/companyfacts/CIK{cik}.json',

headers=headers

)

#review data

companyFacts.json().keys()

companyFacts.json()['facts']

companyFacts.json()['facts'].keys()

# filing metadata

companyFacts.json()['facts']['dei'][

'EntityCommonStockSharesOutstanding']

companyFacts.json()['facts']['dei'][

'EntityCommonStockSharesOutstanding'].keys()

companyFacts.json()['facts']['dei'][

'EntityCommonStockSharesOutstanding']['units']

companyFacts.json()['facts']['dei'][

'EntityCommonStockSharesOutstanding']['units']['shares']

companyFacts.json()['facts']['dei'][

'EntityCommonStockSharesOutstanding']['units']['shares'][0]

# concept data // financial statement line items

companyFacts.json()['facts']['us-gaap']

companyFacts.json()['facts']['us-gaap'].keys()

# different amounts of data available per concept

companyFacts.json()['facts']['us-gaap']['AccountsPayable']

companyFacts.json()['facts']['us-gaap']['Revenues']

companyFacts.json()['facts']['us-gaap']['Assets']

# get company concept data

companyConcept = requests.get(

(

f'https://data.sec.gov/api/xbrl/companyconcept/CIK{cik}'

f'/us-gaap/Assets.json'

),

headers=headers

)

# review data

companyConcept.json().keys()

companyConcept.json()['units']

companyConcept.json()['units'].keys()

companyConcept.json()['units']['USD']

companyConcept.json()['units']['USD'][0]

# parse assets from single filing

companyConcept.json()['units']['USD'][0]['val']

# get all filings data

assetsData = pd.DataFrame.from\_dict((

companyConcept.json()['units']['USD']))

# review data

assetsData.columns

assetsData.form

# get assets from 10Q forms and reset index

assets10Q = assetsData[assetsData.form == '10-Q']

assets10Q = assets10Q.reset\_index(drop=True)

# plot

assets10Q.plot(x='end', y='val')

### invest programs

#!/usr/bin/python3

# Programs called by invest.cgi

#

# Copyright (C) 2019 - 2024 Tom Stevelt

#

# This program is free software: you can redistribute it and/or modify

# it under the terms of the GNU Affero General Public License as

# published by the Free Software Foundation, either version 3 of the

# License, or (at your option) any later version.

#

# This program is distributed in the hope that it will be useful,

# but WITHOUT ANY WARRANTY; without even the implied warranty of

# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

# GNU Affero General Public License for more details.

#

# You should have received a copy of the GNU Affero General Public License

# along with this program. If not, see <https://www.gnu.org/licenses/>.

import sys

import os

from datetime import datetime

import requests

import pandas as pd

import numpy as np

#import scipy as stats

from scipy.stats import linregress

Debug = 0

# 1 = update

# 2 = chart

# 3 = csv

# 4 = 10 year projected

Format = 1

if len(sys.argv) < 6:

print ( "USAGE: FCF.py TICKER PRICE CIK FRAME UNIXTIME" )

if Debug == 0:

quit()

ticker = 'BA'

price = 195.00

cik = '0000012927'

ticker = 'IBM'

price = 158.00

cik = '0000051143'

ticker = 'KVUE'

price = 21.5

cik = '0001944048'

frame = 'CY1970Q'

systime = 1699636971

else:

ticker = sys.argv[1:]

ticker = ticker[0]

price = sys.argv[2:]

price = float(price[0])

cik = sys.argv[3:]

cik = cik[0]

frame = sys.argv[4:]

frame = frame[0]

systime = sys.argv[5:]

systime = systime[0]

if Debug:

print ( '-- ', ticker, price, cik, frame, systime)

# create request header

# get company facts data

headers = {'User-Agent': "tstevelt@silverhammersoftware.com"}

try:

companyFacts = requests.get( f'https://data.sec.gov/api/xbrl/companyfacts/CIK{cik}.json', headers=headers )

except:

print ( f'-- {ticker} {cik} no companyFacts' )

quit()

if companyFacts.status\_code != 200:

print ( f'-- Stock {ticker} {cik} bad URL, status\_code', companyFacts.status\_code )

print ( f'-- https://data.sec.gov/api/xbrl/companyfacts/CIK{cik}.json' )

quit()

##

## Start up functions

##

def FormatOne ( FieldList ):

for Field in FieldList:

try:

ndx = len(companyFacts.json()['facts']['dei'][f'{Field}']['units']['shares']) - 1

Shares = companyFacts.json()['facts']['dei'][f'{Field}']['units']['shares'][ndx]['val']

FiledDate = companyFacts.json()['facts']['dei'][f'{Field}']['units']['shares'][ndx]['filed']

try:

Frame = companyFacts.json()['facts']['dei'][f'{Field}']['units']['shares'][ndx]['frame']

if Debug:

print ( f'-- Format One: Found shares at {Field}' )

except:

Frame = "none"

return Shares, Frame, FiledDate

except:

continue

return 0, "none", "none"

def FormatTwo ( FieldList ):

for Field in FieldList:

try:

ndx = len(companyFacts.json()['facts']['dei'][f'{Field}']['units']['USD']) - 1

Shares = companyFacts.json()['facts']['dei'][f'{Field}']['units']['USD'][ndx]['val']

FiledDate = companyFacts.json()['facts']['dei'][f'{Field}']['units']['USD'][ndx]['filed']

try:

Frame = companyFacts.json()['facts']['dei'][f'{Field}']['units']['USD'][ndx]['frame']

if Debug:

print ( f'-- Format Two: Found shares at {Field}' )

except:

Frame = "none"

return Shares, Frame, FiledDate

except:

continue

return 0, "none", "none"

def FormatThree ( FieldList, SubField ):

for Field in FieldList:

try:

ndx = len(companyFacts.json()['facts']['us-gaap'][f'{Field}']['units'][f'{SubField}']) - 1

Shares = companyFacts.json()['facts']['us-gaap'][f'{Field}']['units'][f'{SubField}'][ndx]['val']

FiledDate = companyFacts.json()['facts']['us-gaap'][f'{Field}']['units'][f'{SubField}'][ndx]['filed']

try:

Frame = companyFacts.json()['facts']['us-gaap'][f'{Field}']['units'][f'{SubField}'][ndx]['frame']

if Debug:

print ( f'-- Format Three: Found shares at {Field}' )

except:

Frame = "none"

return Shares, Frame, FiledDate

except:

continue

return 0, "none", "none"

def days\_between(d1, d2):

d1 = datetime.strptime(d1, "%Y-%m-%d")

d2 = datetime.strptime(d2, "%Y-%m-%d")

return abs((d2 - d1).days)

##

## Start up -- get filed date and shares

##

FieldList = { 'EntityCommonStockSharesOutstanding', 'EntityPublicFloat', 'CommonStockSharesOutstanding', 'CommonStockSharesIssued', 'NumberOfSharesOutstanding', 'WeightedAverageNumberOfSharesOutstandingBasic' }

Shares, Frame, FiledDate = FormatOne ( FieldList )

if Frame == 'none':

Shares, Frame, FiledDate = FormatTwo ( FieldList )

if Frame == 'none':

Shares, Frame, FiledDate = FormatThree ( FieldList, 'shares' )

if Frame == 'none':

print ( f'-- {ticker} unrecognizable json file' )

print ( f'-- https://data.sec.gov/api/xbrl/companyfacts/CIK{cik}.json' )

quit()

CurrentDate = datetime.today().strftime('%Y-%m-%d')

try:

Days = days\_between ( FiledDate, CurrentDate )

except:

print ( f'-- {ticker} days\_between failed' )

print ( f'-- https://data.sec.gov/api/xbrl/companyfacts/CIK{cik}.json' )

quit()

if Days < 100 and frame == Frame[0:8]:

print ( f'-- No new filing for {ticker}' )

quit ()

if frame > Frame[0:8]:

if Debug:

print ( f'-- Wacky {ticker} {frame} > {Frame}' )

#quit ()

Frame = frame

if Debug:

print ( f'-- {ticker} Shares {Shares}, {Frame}, {FiledDate}' )

if Days > 100:

print ( f'-- {ticker} processing old filed date {FiledDate}' )

##

## Functions for getting other data fields

##

def GetDataList ( Fields ):

for Field in Fields:

try:

data = companyFacts.json()['facts']['us-gaap'][f'{Field}']['units']['USD']

df = pd.DataFrame.from\_dict(data)

df = df.dropna()

if Debug:

print ( f'-- Get Data List: found {Field} in USD' )

return( df )

except:

try:

data = companyFacts.json()['facts']['us-gaap'][f'{Field}']['units']['USD/shares']

df = pd.DataFrame.from\_dict(data)

df = df.dropna()

if Debug:

print ( f'-- Get Data List: found {Field} in USD/shares' )

return( df )

except:

continue

return pd.DataFrame()

def GetLatesetFiscalYear(df):

Number = 0

if df.empty:

return Number

Length = len(df)

ndx = Length - 1

while ndx >= 0:

if 'fp' in df.columns and df.iloc[ndx]['fp'] == 'FY':

Number = int(df.iloc[ndx]['val'])

return Number

ndx -= 1

return Number

def MakeArray ( dfNetCash, dfCapEx ):

Numbers = []

if dfNetCash.empty:

return Numbers

lenNetCash = len(dfNetCash)

ndx = 0

xo = -1

if dfCapEx.empty:

lenCapEx = 0

else:

lenCapEx = len(dfCapEx)

while ndx < lenNetCash:

Numbers.append(dfNetCash.iloc[ndx]['val'])

xo = xo + 1

ThisDate = (dfNetCash.iloc[ndx]['end'])

if not dfCapEx.empty:

for xc in range(lenCapEx):

if ThisDate == dfCapEx.iloc[xc]['end']:

Numbers[xo] = Numbers[xo] - dfCapEx.iloc[xc]['val']

break

ndx = ndx + 1

return Numbers

##

## Get data fields

##

##

## Free Cash Flow is commonly calculated as:

##

## FCF=NetCashfromOperatingActivities−CapitalExpendituresFCF=NetCashfromOperatingActivities−CapitalExpenditures

##

## Here are some XBRL tags that you might use:

##

## Net Cash Provided by Operating Activities:

## XBRL Tag: us-gaap:NetCashProvidedByUsedInOperatingActivities

##

## Capital Expenditures (Capital Expenditures are typically part of Investing Activities):

## XBRL Tag: us-gaap:PaymentsToAcquirePropertyPlantAndEquipmen/

##

FieldList = { 'NetCashProvidedByUsedInOperatingActivities' }

dfNetCash = GetDataList ( FieldList )

if Debug:

print ( f'{ticker} NetCashProvidedByUsedInOperatingActivities' )

print ( dfNetCash.to\_string() )

if not dfNetCash.empty:

dfNetCash = dfNetCash.drop(dfNetCash[dfNetCash.fp != 'FY'].index)

else:

print ( f'-- {ticker} missing NetCashProvidedByUsedInOperatingActivities' )

quit()

NetCash = GetLatesetFiscalYear(dfNetCash)

FieldList = { 'PaymentsToAcquirePropertyPlantAndEquipment' }

dfCapEx = GetDataList ( FieldList )

if not dfCapEx.empty:

dfCapEx = dfCapEx.drop(dfCapEx[dfCapEx.fp != 'FY'].index)

if Debug:

print ( f'{ticker} PaymentsToAcquirePropertyPlantAndEquipment' )

print ( dfCapEx.to\_string() )

CapEx = GetLatesetFiscalYear(dfCapEx)

FreeCashFlow = NetCash - CapEx;

if Format == 2:

print ( f'-- {ticker} NetCash {NetCash} - CapEx {CapEx} = FreeCashFlow {FreeCashFlow}')

FCF\_Array = MakeArray ( dfNetCash, dfCapEx )

if len(FCF\_Array) == 0:

quit()

arr = np.array(FCF\_Array)

xl = len(arr)

slope, intercept, r, p, std\_err = 0.0, 0.0, 0.0, 0.0, 0.0

tangent = 0.0

MEAN = 0.0

CV = 0

if xl > 6:

try:

SD = np.std(FCF\_Array)

MEAN = np.mean(FCF\_Array)

if MEAN != 0:

CV = SD / MEAN

years = []

value = 1

for i in range(xl):

years.append(value)

value = value + 1

slope, intercept, r, p, std\_err = linregress(years, arr)

if intercept != 0:

tangent = slope / intercept

except:

slope, intercept, r, p, std\_err = 0.0, 0.0, 0.0, 0.0, 0.0

if Format == 1 and FreeCashFlow != 0:

tangent = tangent \* 100.0

if pd.isna(tangent) or np.isinf(tangent):

tangent = 0.0;

if pd.isna(CV) or np.isinf(CV):

CV = 0.0;

print ( f"update fundamental set Ffreecash = {FreeCashFlow}, Ffcfgrow = {tangent:.2f}, Ffcfcv = {CV:.2f}, Fupdated = {systime} where Fticker = '{ticker}';" );

if Format == 2:

print ( f'-- {ticker} StdDev {SD:.0f} Mean {MEAN:.0f} CV {CV:.3f}' )

print ( f'-- {ticker} slope {slope}, intercept {intercept}, r {r}, p {p}, std\_err {std\_err}' )

x = 0

for i in range(xl):

x = years[i]

y = intercept + slope \* x

print ( arr[i], ',', y )

for i in range(10):

y = intercept + slope \* (x + i + 1)

print ( ' ', ',', y )

if Format == 3 and xl > 8:

print ( f'{ticker}, {xl}, {CV:.3f}, {slope:.0f}, {intercept:.0f}, {tangent:.3f}, {FreeCashFlow}' )

if Format == 4 and xl > 0:

x = xl

y0 = intercept + slope \* (x)

x = x + 1

y1 = intercept + slope \* (x)

x = x + 1

y2 = intercept + slope \* (x)

x = x + 1

y3 = intercept + slope \* (x)

x = x + 1

y4 = intercept + slope \* (x)

x = x + 1

y5 = intercept + slope \* (x)

x = x + 1

y6 = intercept + slope \* (x)

x = x + 1

y7 = intercept + slope \* (x)

x = x + 1

y8 = intercept + slope \* (x)

x = x + 1

y9 = intercept + slope \* (x)

print ( f'{ticker}, {y0:.0f}, {y1:.0f}, {y2:.0f}, {y3:.0f}, {y4:.0f}, {y5:.0f}, {y6:.0f}, {y7:.0f}, {y8:.0f}, {y9:.0f}' )

#!/usr/bin/python3

# Programs called by invest.cgi

#

# Copyright (C) 2019 - 2024 Tom Stevelt

#

# This program is free software: you can redistribute it and/or modify

# it under the terms of the GNU Affero General Public License as

# published by the Free Software Foundation, either version 3 of the

# License, or (at your option) any later version.

#

# This program is distributed in the hope that it will be useful,

# but WITHOUT ANY WARRANTY; without even the implied warranty of

# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

# GNU Affero General Public License for more details.

#

# You should have received a copy of the GNU Affero General Public License

# along with this program. If not, see <https://www.gnu.org/licenses/>.

import sys

import os

from datetime import datetime

import requests

import pandas as pd

import numpy as np

if len(sys.argv) < 6:

print ( "USAGE: FCF.py TICKER PRICE CIK FRAME UNIXTIME" )

#fixit

quit()

ticker = 'BA'

price = 195.00

cik = '0000012927'

ticker = 'IBM'

price = 158.00

cik = '0000051143'

frame = 'CY1970Q'

systime = 1699636971

else:

ticker = sys.argv[1:]

ticker = ticker[0]

price = sys.argv[2:]

price = float(price[0])

cik = sys.argv[3:]

cik = cik[0]

frame = sys.argv[4:]

frame = frame[0]

systime = sys.argv[5:]

systime = systime[0]

#fixit

Debug = 0

if Debug:

print ( '-- ', ticker, price, cik, frame, systime)

# create request header

# get company facts data

headers = {'User-Agent': "tstevelt@silverhammersoftware.com"}

try:

companyFacts = requests.get( f'https://data.sec.gov/api/xbrl/companyfacts/CIK{cik}.json', headers=headers )

except:

print ( f'-- {ticker} {cik} no companyFacts' )

quit()

if companyFacts.status\_code != 200:

print ( f'-- Stock {ticker} {cik} bad URL, status\_code', companyFacts.status\_code )

print ( f'-- https://data.sec.gov/api/xbrl/companyfacts/CIK{cik}.json' )

quit()

##

## Start up functions

##

def FormatOne ( FieldList ):

for Field in FieldList:

try:

ndx = len(companyFacts.json()['facts']['dei'][f'{Field}']['units']['shares']) - 1

Shares = companyFacts.json()['facts']['dei'][f'{Field}']['units']['shares'][ndx]['val']

FiledDate = companyFacts.json()['facts']['dei'][f'{Field}']['units']['shares'][ndx]['filed']

try:

Frame = companyFacts.json()['facts']['dei'][f'{Field}']['units']['shares'][ndx]['frame']

if Debug:

print ( f'-- Format One: Found shares at {Field}' )

except:

Frame = "none"

return Shares, Frame, FiledDate

except:

continue

return 0, "none", "none"

def FormatTwo ( FieldList ):

for Field in FieldList:

try:

ndx = len(companyFacts.json()['facts']['dei'][f'{Field}']['units']['USD']) - 1

Shares = companyFacts.json()['facts']['dei'][f'{Field}']['units']['USD'][ndx]['val']

FiledDate = companyFacts.json()['facts']['dei'][f'{Field}']['units']['USD'][ndx]['filed']

try:

Frame = companyFacts.json()['facts']['dei'][f'{Field}']['units']['USD'][ndx]['frame']

if Debug:

print ( f'-- Format Two: Found shares at {Field}' )

except:

Frame = "none"

return Shares, Frame, FiledDate

except:

continue

return 0, "none", "none"

def FormatThree ( FieldList, SubField ):

for Field in FieldList:

try:

ndx = len(companyFacts.json()['facts']['us-gaap'][f'{Field}']['units'][f'{SubField}']) - 1

Shares = companyFacts.json()['facts']['us-gaap'][f'{Field}']['units'][f'{SubField}'][ndx]['val']

FiledDate = companyFacts.json()['facts']['us-gaap'][f'{Field}']['units'][f'{SubField}'][ndx]['filed']

try:

Frame = companyFacts.json()['facts']['us-gaap'][f'{Field}']['units'][f'{SubField}'][ndx]['frame']

if Debug:

print ( f'-- Format Three: Found shares at {Field}' )

except:

Frame = "none"

return Shares, Frame, FiledDate

except:

continue

return 0, "none", "none"

def days\_between(d1, d2):

d1 = datetime.strptime(d1, "%Y-%m-%d")

d2 = datetime.strptime(d2, "%Y-%m-%d")

return abs((d2 - d1).days)

##

## Start up -- get filed date and shares

##

FieldList = { 'EntityCommonStockSharesOutstanding', 'EntityPublicFloat', 'CommonStockSharesOutstanding', 'CommonStockSharesIssued', 'NumberOfSharesOutstanding', 'WeightedAverageNumberOfSharesOutstandingBasic' }

Shares, Frame, FiledDate = FormatOne ( FieldList )

if Frame == 'none':

Shares, Frame, FiledDate = FormatTwo ( FieldList )

if Frame == 'none':

Shares, Frame, FiledDate = FormatThree ( FieldList, 'shares' )

if Frame == 'none':

print ( f'-- {ticker} unrecognizable json file' )

print ( f'-- https://data.sec.gov/api/xbrl/companyfacts/CIK{cik}.json' )

quit()

CurrentDate = datetime.today().strftime('%Y-%m-%d')

try:

Days = days\_between ( FiledDate, CurrentDate )

except:

print ( f'-- {ticker} days\_between failed' )

print ( f'-- https://data.sec.gov/api/xbrl/companyfacts/CIK{cik}.json' )

quit()

if Days < 100 and frame == Frame[0:8]:

print ( f'-- No new filing for {ticker}' )

quit ()

if frame > Frame[0:8]:

if Debug:

print ( f'-- Wacky {ticker} {frame} > {Frame}' )

#quit ()

Frame = frame

if Debug:

print ( f'-- {ticker} Shares {Shares}, {Frame}, {FiledDate}' )

if Days > 100:

print ( f'-- {ticker} processing old filed date {FiledDate}' )

##

## Functions for getting other data fields

##

def PrintLastTen ( df ):

Length = len(df)

if Length >= 15:

print(df.iloc[Length-15:Length].to\_string())

else:

print(df.to\_string())

def GetDataList ( Fields ):

for Field in Fields:

try:

data = companyFacts.json()['facts']['us-gaap'][f'{Field}']['units']['USD']

df = pd.DataFrame.from\_dict(data)

df = df.dropna()

if Debug:

print ( f'-- Get Data List: found {Field} in USD' )

return( df )

except:

try:

data = companyFacts.json()['facts']['us-gaap'][f'{Field}']['units']['USD/shares']

df = pd.DataFrame.from\_dict(data)

df = df.dropna()

if Debug:

print ( f'-- Get Data List: found {Field} in USD/shares' )

return( df )

except:

continue

return pd.DataFrame()

def GetLatesetFiscalYear(df):

Number = 0

if df.empty:

return Number

Length = len(df)

ndx = Length - 1

while ndx >= 0:

if 'fp' in df.columns and df.iloc[ndx]['fp'] == 'FY':

Number = int(df.iloc[ndx]['val'])

return Number

ndx -= 1

return Number

##

## Get data fields

##

##

## Free Cash Flow is commonly calculated as:

##

## FCF=NetCashfromOperatingActivities−CapitalExpendituresFCF=NetCashfromOperatingActivities−CapitalExpenditures

##

## Here are some XBRL tags that you might use:

##

## Net Cash Provided by Operating Activities:

## XBRL Tag: us-gaap:NetCashProvidedByUsedInOperatingActivities

##

## Capital Expenditures (Capital Expenditures are typically part of Investing Activities):

## XBRL Tag: us-gaap:PaymentsToAcquirePropertyPlantAndEquipmen/

##

FieldList = { 'NetCashProvidedByUsedInOperatingActivities' }

df = GetDataList ( FieldList )

if Debug:

print ( f'{ticker} NetCashProvidedByUsedInOperatingActivities' )

PrintLastTen ( df )

NetCash = GetLatesetFiscalYear(df)

FieldList = { 'PaymentsToAcquirePropertyPlantAndEquipment' }

df = GetDataList ( FieldList )

if Debug:

print ( f'{ticker} PaymentsToAcquirePropertyPlantAndEquipment' )

PrintLastTen ( df )

CapEx = GetLatesetFiscalYear(df)

FreeCashFlow = NetCash - CapEx;

print ( f'-- {ticker} NetCash {NetCash} - CapEx {CapEx} = FreeCashFlow {FreeCashFlow}')

if FreeCashFlow != 0:

print ( f"update fundamental set Ffreecash = {FreeCashFlow}, Fupdated = {systime} where Fticker = '{ticker}';" );

#!/usr/bin/python3

# Programs called by invest.cgi

#

# Copyright (C) 2019 - 2024 Tom Stevelt

#

# This program is free software: you can redistribute it and/or modify

# it under the terms of the GNU Affero General Public License as

# published by the Free Software Foundation, either version 3 of the

# License, or (at your option) any later version.

#

# This program is distributed in the hope that it will be useful,

# but WITHOUT ANY WARRANTY; without even the implied warranty of

# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

# GNU Affero General Public License for more details.

#

# You should have received a copy of the GNU Affero General Public License

# along with this program. If not, see <https://www.gnu.org/licenses/>.

import sys

import os

from datetime import datetime

import requests

import pandas as pd

Debug = 0

if len(sys.argv) < 6:

print ( "USAGE: getfundSEC.py TICKER PRICE CIK FRAME UNIXTIME" )

if not Debug:

quit()

ticker = 'IBM'

price = 158.00

cik = '0000051143'

ticker = 'SKW'

price = 94.0

cik = '0000093556'

ticker = 'BA'

price = 195.00

cik = '0000012927'

frame = 'CY1970Q'

systime = 1699636971

else:

ticker = sys.argv[1:]

ticker = ticker[0]

price = sys.argv[2:]

price = float(price[0])

cik = sys.argv[3:]

cik = cik[0]

frame = sys.argv[4:]

frame = frame[0]

systime = sys.argv[5:]

systime = systime[0]

if Debug:

print ( '-- ', ticker, price, cik, frame, systime)

# create request header

# get company facts data

headers = {'User-Agent': "tstevelt@silverhammersoftware.com"}

try:

companyFacts = requests.get( f'https://data.sec.gov/api/xbrl/companyfacts/CIK{cik}.json', headers=headers )

except:

print ( f'-- {ticker} {cik} no companyFacts' )

quit()

if companyFacts.status\_code != 200:

print ( f'-- Stock {ticker} {cik} bad URL, status\_code', companyFacts.status\_code )

print ( f'-- https://data.sec.gov/api/xbrl/companyfacts/CIK{cik}.json' )

quit()

##

## Start up functions

##

def FormatOne ( FieldList ):

for Field in FieldList:

try:

ndx = len(companyFacts.json()['facts']['dei'][f'{Field}']['units']['shares']) - 1

Shares = companyFacts.json()['facts']['dei'][f'{Field}']['units']['shares'][ndx]['val']

FiledDate = companyFacts.json()['facts']['dei'][f'{Field}']['units']['shares'][ndx]['filed']

try:

Frame = companyFacts.json()['facts']['dei'][f'{Field}']['units']['shares'][ndx]['frame']

if Debug:

print ( f'-- FormatOne: Found shares at {Field}' )

except:

Frame = "none"

return Shares, Frame, FiledDate

except:

continue

return 0, "none", "none"

def FormatTwo ( FieldList ):

for Field in FieldList:

try:

ndx = len(companyFacts.json()['facts']['dei'][f'{Field}']['units']['USD']) - 1

Shares = companyFacts.json()['facts']['dei'][f'{Field}']['units']['USD'][ndx]['val']

FiledDate = companyFacts.json()['facts']['dei'][f'{Field}']['units']['USD'][ndx]['filed']

try:

Frame = companyFacts.json()['facts']['dei'][f'{Field}']['units']['USD'][ndx]['frame']

if Debug:

print ( f'-- FormatTwo: Found shares at {Field}' )

except:

Frame = "none"

return Shares, Frame, FiledDate

except:

continue

return 0, "none", "none"

def FormatThree ( FieldList, SubField ):

for Field in FieldList:

try:

ndx = len(companyFacts.json()['facts']['us-gaap'][f'{Field}']['units'][f'{SubField}']) - 1

Shares = companyFacts.json()['facts']['us-gaap'][f'{Field}']['units'][f'{SubField}'][ndx]['val']

FiledDate = companyFacts.json()['facts']['us-gaap'][f'{Field}']['units'][f'{SubField}'][ndx]['filed']

try:

Frame = companyFacts.json()['facts']['us-gaap'][f'{Field}']['units'][f'{SubField}'][ndx]['frame']

if Debug:

print ( f'-- FormatThree: Found shares at {Field}' )

except:

Frame = "none"

return Shares, Frame, FiledDate

except:

continue

return 0, "none", "none"

def days\_between(d1, d2):

d1 = datetime.strptime(d1, "%Y-%m-%d")

d2 = datetime.strptime(d2, "%Y-%m-%d")

return abs((d2 - d1).days)

##

## Start up -- get filed date and shares

##

FieldList = { 'EntityCommonStockSharesOutstanding', 'EntityPublicFloat', 'CommonStockSharesOutstanding', 'CommonStockSharesIssued', 'NumberOfSharesOutstanding', 'WeightedAverageNumberOfSharesOutstandingBasic' }

Shares, Frame, FiledDate = FormatOne ( FieldList )

if Frame == 'none':

Shares, Frame, FiledDate = FormatTwo ( FieldList )

if Frame == 'none':

Shares, Frame, FiledDate = FormatThree ( FieldList, 'shares' )

if Frame == 'none':

print ( f'-- {ticker} unrecognizable json file' )

print ( f'-- https://data.sec.gov/api/xbrl/companyfacts/CIK{cik}.json' )

quit()

CurrentDate = datetime.today().strftime('%Y-%m-%d')

try:

Days = days\_between ( FiledDate, CurrentDate )

except:

print ( f'-- {ticker} days\_between failed' )

print ( f'-- https://data.sec.gov/api/xbrl/companyfacts/CIK{cik}.json' )

quit()

if Days < 100 and frame == Frame[0:8]:

print ( f'-- No new filing for {ticker}' )

quit ()

if frame > Frame[0:8]:

if Debug:

print ( f'-- Wacky {ticker} {frame} > {Frame}' )

#quit ()

Frame = frame

if Debug:

print ( f'-- {ticker} Shares {Shares}, {Frame}, {FiledDate}' )

if Days > 100:

print ( f'-- {ticker} processing old filed date {FiledDate}' )

##

## Functions for getting other data fields

##

def PrintLastTen ( df ):

Length = len(df)

if Length >= 10:

print(df.iloc[Length-10:Length])

else:

print(df)

SubFields = { 'USD', 'USD/shares', 'shares' }

def GetData ( Field ):

for SubField in SubFields:

try:

data = companyFacts.json()['facts']['us-gaap'][f'{Field}']['units'][f'{SubField}']

df = pd.DataFrame.from\_dict(data)

df = df.dropna()

return( df )

except:

continue

return pd.DataFrame()

def GetDataList ( Fields ):

for Field in Fields:

try:

data = companyFacts.json()['facts']['us-gaap'][f'{Field}']['units']['USD']

df = pd.DataFrame.from\_dict(data)

df = df.dropna()

if Debug:

print ( f'-- GetDataList: found {Field} in USD' )

return( df )

except:

try:

data = companyFacts.json()['facts']['us-gaap'][f'{Field}']['units']['USD/shares']

df = pd.DataFrame.from\_dict(data)

df = df.dropna()

if Debug:

print ( f'-- GetDataList: found {Field} in USD/shares' )

return( df )

except:

continue

return pd.DataFrame()

def GetValueTTM ( df ):

if df.empty:

return 0

try:

Length = len(df)

if Length < 7:

return 0

except:

return 0

ndx = Length - 1

Value = 0

if df.iloc[ndx]['fp'] == 'FY':

Value = df.iloc[ndx]['val']

elif df.iloc[ndx]['fp'] == 'Q1':

for x in range(ndx-1,Length):

Value += df.iloc[x]['val']

Value -= df.iloc[ndx-4]['val']

elif df.iloc[ndx]['fp'] == 'Q2':

for x in range(ndx-2,Length):

Value += df.iloc[x]['val']

Value -= df.iloc[ndx-4]['val']

Value -= df.iloc[ndx-5]['val']

elif df.iloc[ndx]['fp'] == 'Q3':

for x in range(ndx-3,Length):

Value += df.iloc[x]['val']

Value -= df.iloc[ndx-4]['val']

Value -= df.iloc[ndx-5]['val']

Value -= df.iloc[ndx-6]['val']

return Value

def GetValueLastFourQtr ( df ):

if df.empty:

return 0

try:

Length = len(df)

if Length < 10:

return 0

except:

return 0

Value = 0

Count = 0

ndx = Length - 1

try:

while Count < 4 and ndx >= 0:

if df.iloc[ndx]['fp'] == 'Q1' or df.iloc[ndx]['fp'] == 'Q2' or df.iloc[ndx]['fp'] == 'Q3' or df.iloc[ndx]['fp'] == 'Q4':

Value += df.iloc[ndx]['val']

Count += 1

ndx -= 1

except:

print ( f'GetValueLastFourQtr failed on {ticker}, length {Length}' );

return Value

def GetValueLast ( df ):

if df.empty:

return 0

try:

Length = len(df)

except:

return 0

Value = df.iloc[Length-1]['val']

return Value

def GetQuarters(df):

Numbers = []

if df.empty:

return Numbers

Length = len(df)

if Length < 4:

return Numbers

ndx = Length - 1

while ndx >=4:

if df.iloc[ndx]['fp'] == 'FY':

QtrValue = df.iloc[ndx]['val'] - (df.iloc[ndx-1]['val'] + df.iloc[ndx-2]['val'] + df.iloc[ndx-3]['val'])

Numbers.append(int(QtrValue))

else:

Numbers.append(df.iloc[ndx]['val'])

ndx = ndx - 1

return Numbers

def CalcYoYPercent (Numbers):

Length = len(Numbers)

if Length < 5:

return 0.0

if Numbers[4] <= 0 or Numbers[0] <= 0:

return (0.0)

Answer = 100.0 \* (Numbers[0] - Numbers[4]) / Numbers[4]

return Answer

def GetLatesetFiscalYear(df):

Number = 0

if df.empty:

return Number

Length = len(df)

ndx = Length - 1

while ndx >= 0:

if 'fp' in df.columns and df.iloc[ndx]['fp'] == 'FY':

Number = int(df.iloc[ndx]['val'])

return Number

ndx -= 1

return Number

##

## Get data fields

##

if Shares < 1000:

df = GetData('CommonStockSharesIssued')

if Debug > 1:

print ( df )

Shares = GetValueLast ( df )

df = GetDataList ( {'EarningsPerShareDiluted','EarningsPerShareBasic'} )

if Debug > 1:

PrintLastTen ( df )

EPS = GetValueTTM ( df )

MyArray = GetQuarters ( df )

EarnGrowPct = CalcYoYPercent(MyArray)

if Debug:

print ( f'MyArray {MyArray}')

print ( f'EarnGrowPct {EarnGrowPct}')

FieldList = { 'Revenues', 'RevenuesNetOfInterestExpense', 'RevenueFromContractWithCustomerExcludingAssessedTax' }

df = GetDataList ( FieldList )

if Debug > 1:

PrintLastTen ( df )

Revenue = GetValueTTM ( df )

if Revenue <= 0:

Revenue = GetValueLastFourQtr ( df )

MyArray = GetQuarters ( df )

RevenueGrowPct = CalcYoYPercent(MyArray)

if Debug:

print ( f'MyArray {MyArray}')

print ( f'RevenueGrowPct {RevenueGrowPct}')

df = GetData ( 'Assets' )

Assets = GetValueLast ( df )

df = GetDataList({'StockholdersEquity','StockholdersEquityIncludingPortionAttributableToNoncontrollingInterest'})

StockholdersEquity = GetValueLast(df)

df = GetData('LiabilitiesAndStockholdersEquity')

LiabilitiesAndStockholdersEquity = GetValueLast(df)

df = GetData('LongTermDebt')

LongTermDebt = GetValueLast(df)

# df = GetData('IncomeLossFromContinuingOperations')

#df = GetData('OperatingIncomeLoss')

#Earnings = GetValueLast ( df )

df = GetData('InterestExpense')

Interest = GetValueTTM ( df )

FieldList = { 'IncomeTaxesPaid', 'IncomeTaxesPaidNet' }

df = GetDataList(FieldList)

Taxes = GetValueTTM ( df )

FieldList = { 'Depreciation', 'DepreciationAndAmortization', 'NetIncomeLossFromContinuingOperationsAvailableToCommonShareholdersBasic' }

df = GetDataList(FieldList)

Depreciation = GetValueTTM ( df )

df = GetData ( 'AmortizationOfFinancingCosts' )

AmortFinance = GetValueTTM ( df )

df = GetData ( 'AmortizationOfIntangibleAssets' )

AmortIntangible = GetValueTTM ( df )

df = GetData('MinorityInterest')

MinorityInterest = GetValueLast ( df )

df = GetData('LongTermDebtNoncurrent')

LongTermDebtNoncurrent = GetValueLast ( df )

df = GetData('OperatingLeaseLiabilityCurrent')

OperatingLeaseLiabilityCurrent = GetValueLast ( df )

df = GetData('AccountsPayableCurrent')

AccountsPayableCurrent = GetValueLast ( df )

df = GetData('CashAndCashEquivalentsAtCarryingValue')

CashEquiv = GetValueLast ( df )

FieldList = { 'NetIncomeLoss', 'NetIncomeLossAvailableToCommonStockholdersDiluted' }

df = GetDataList ( FieldList )

NetIncome = GetValueTTM ( df )

df = GetData ( 'AssetsCurrent' )

AssetsCurrent = GetValueLast ( df )

df = GetData ( 'TreasuryStockShares' )

TreasuryStockShares = GetValueLast ( df )

FieldList = { 'Liabilities', 'LiabilitiesCurrent' }

df = GetDataList ( FieldList )

Liabilities = GetValueLast ( df )

FieldList = { 'NetCashProvidedByUsedInOperatingActivities' }

df = GetDataList ( FieldList )

NetCash = GetLatesetFiscalYear(df)

FieldList = { 'PaymentsToAcquirePropertyPlantAndEquipment' }

df = GetDataList ( FieldList )

CapEx = GetLatesetFiscalYear(df)

if Debug:

print ( '-- After extract all data' )

print ( f'-- Shares {Shares}' )

print ( f'-- EPS {EPS}' )

print ( f'-- Assets {Assets}' )

print ( f'-- StockholdersEquity {StockholdersEquity}' )

print ( f'-- LiabilitiesAndStockholdersEquity {LiabilitiesAndStockholdersEquity}' )

print ( f'-- LongTermDebt {LongTermDebt}' )

print ( f'-- LongTermDebtNoncurrent {LongTermDebtNoncurrent}' )

print ( f'-- Interest {Interest}' )

print ( f'-- Taxes {Taxes}' )

print ( f'-- Depreciation {Depreciation}' )

print ( f'-- AmortFinance {AmortFinance}' )

print ( f'-- AmortIntangible {AmortIntangible}' )

print ( f'-- MinorityInterest {MinorityInterest}' )

print ( f'-- OperatingLeaseLiabilityCurrent {OperatingLeaseLiabilityCurrent}' )

print ( f'-- AccountsPayableCurrent {AccountsPayableCurrent}' )

print ( f'-- CashEquiv {CashEquiv}' )

print ( f'-- NetIncome {NetIncome}' )

print ( f'-- Revenue {Revenue}' )

print ( f'-- AssetsCurrent {AssetsCurrent}' )

print ( f'-- TreasuryStockShares {TreasuryStockShares}' )

print ( f'-- Liabilities {Liabilities}\n\n' )

print ( f'-- NetCash {NetCash}' )

print ( f'-- CapEx {CapEx}' )

##

## Calculations

##

if Debug:

print ( '-- Calculations' )

if StockholdersEquity == 0:

StockholdersEquity = LiabilitiesAndStockholdersEquity - Liabilities

Fdebteq = 0.0

if StockholdersEquity > 0:

Fdebteq = LongTermDebt / StockholdersEquity

if Debug:

print ( f'-- Fdebteq {Fdebteq}' )

Earnings = Shares \* EPS

if Debug:

print ( f'-- Earnings {Earnings}' )

EBIT = Earnings + Interest + Taxes

EBITDA = EBIT + Depreciation + AmortFinance + AmortIntangible

if Debug:

print ( f'-- EBIT {EBIT}' )

print ( f'-- Depreciation {Depreciation}' )

print ( f'-- AmortFinance {AmortFinance}' )

print ( f'-- AmortIntangible {AmortIntangible}' )

print ( f'-- EBITDA {EBITDA}' )

MarketCap = Shares \* price;

EnterpriseValue = MarketCap + MinorityInterest + LongTermDebtNoncurrent + OperatingLeaseLiabilityCurrent + AccountsPayableCurrent-CashEquiv

if Debug:

print ( f'-- ticker {MarketCap} + {MinorityInterest} + {LongTermDebtNoncurrent} + {OperatingLeaseLiabilityCurrent} + {AccountsPayableCurrent} + {CashEquiv} = {EnterpriseValue}' )

if EBITDA > 0:

StockValue = EnterpriseValue / EBITDA

else:

StockValue = 0.0

# this is normal for ETF

if Assets > 0:

ReturnAssets = 100.0 \* NetIncome / Assets

else:

ReturnAssets = 0.0

print ( f'-- {ticker} No Assets!!!' )

if Revenue > 0:

GrossMargin = 100.0 \* NetIncome / Revenue

else:

GrossMargin = 0.0

print ( f'-- {ticker} No Revenue!!!' )

"""

Quick Ratio from https://www.investopedia.com/terms/q/quickratio.asp

"""

TenPercent = Shares \* 0.1

if TreasuryStockShares > TenPercent:

MarketableSecurities = price \* TenPercent

if Debug:

print ( f'-- price {price} 10% Shares {TenPercent} MarketableSecurities {MarketableSecurities}' )

else:

MarketableSecurities = price \* TreasuryStockShares

if Debug:

print ( f'-- price {price} TreasuryStockShares {TreasuryStockShares} MarketableSecurities {MarketableSecurities}' )

if Debug:

print ( f'-- MarketableSecurities', MarketableSecurities)

if Liabilities > 0:

QuickRatio = ( AssetsCurrent + CashEquiv + MarketableSecurities) / (Liabilities )

else:

QuickRatio = 0.0

print ( f'-- {ticker} No Liabilities!!!' )

if LongTermDebtNoncurrent == 0 and LongTermDebt > 0:

LongTermDebtNoncurrent = LongTermDebt

FreeCashFlow = NetCash - CapEx;

print ( f"update fundamental set Fdebteq = {Fdebteq:.2f}, Ftotasst = {Assets}, FreturnA = {ReturnAssets:.2f}, Fmargin = {GrossMargin:.2f}, Fequity = {StockholdersEquity:.0f}, Febitda = {EBITDA}, Fminority = {MinorityInterest}, FdebtLT = {LongTermDebt}, FdebtNC = {LongTermDebtNoncurrent}, Fliab = {Liabilities}, Fpayables = {AccountsPayableCurrent}, Fcurasst = {AssetsCurrent}, Fcash = {CashEquiv}, Ftreasury = {TreasuryStockShares:.0f}, Frevenuegrow = {RevenueGrowPct:.2f}, Fearngrow = {EarnGrowPct:.2f}, Fframe = '{Frame[0:8]}', Ffreecash = {FreeCashFlow}, Fupdated = {systime} where Fticker = '{ticker}';" )

### data scrpaing

import requests

import pandas as pd

import matplotlib.pyplot as plt

from datetime import datetime, timedelta

# Define the CIK for Microsoft (0000789019)

cik = '0000789019'

# Set headers for the request

headers = {'User-Agent': "joshjothom05@gmail.com"}

# Get filing metadata for Microsoft

filingMetadata = requests.get(

f'https://data.sec.gov/submissions/CIK{cik}.json',

headers=headers

)

# Convert filing metadata to DataFrame

allForms = pd.DataFrame.from\_dict(filingMetadata.json()['filings']['recent'])

# Filter for only 10-K forms in the last 5 years

allForms['reportDate'] = pd.to\_datetime(allForms['reportDate'], errors='coerce')

five\_years\_ago = datetime.now() - timedelta(days=5\*365)

tenK\_filings = allForms[(allForms['form'] == '10-K') & (allForms['reportDate'] > five\_years\_ago)]

# Get company facts data

companyFacts = requests.get(

f'https://data.sec.gov/api/xbrl/companyfacts/CIK{cik}.json',

headers=headers

)

# Retrieve 'us-gaap: Assets' data

companyConcept = requests.get(

f'https://data.sec.gov/api/xbrl/companyconcept/CIK{cik}/us-gaap/Assets.json',

headers=headers

)

# Convert assets data to DataFrame

assetsData = pd.DataFrame.from\_dict(companyConcept.json()['units']['USD'])

# Filter only 10-K forms and dates in the past 5 years

assetsData['end'] = pd.to\_datetime(assetsData['end'], errors='coerce')

assets10K = assetsData[(assetsData['form'] == '10-K') & (assetsData['end'] > five\_years\_ago)]

assets10K = assets10K.reset\_index(drop=True)

# Plot assets over time for 10-K filings

assets10K.plot(x='end', y='val', title='Microsoft Annual Assets (10-K Filings in Last 5 Years)', marker='o')

plt.xlabel('Year')

plt.ylabel('Assets (in USD)')

plt.grid(True)

plt.show()

### data scraping

import pandas as pd

import requests

import os

def get\_income\_statement(ticker, limit, key, period):

URL = f'https://financialmodelingprep.com/api/v3/income-statement/{ticker}?period={period}&limit={limit}&apikey={key}'

try:

r = requests.get(URL)

r.raise\_for\_status() # Check if the request was successful

print(f"Received Income Statement for {ticker}")

income\_statement = pd.DataFrame.from\_dict(r.json())

return income\_statement

except requests.exceptions.HTTPError as e:

print(f"HTTPError for {ticker}: {str(e)}")

print(f"Response: {r.text}")

except Exception as e:

print(f"Unexpected ERROR while requesting Income Statement for {ticker}: {str(e)}")

# (Other functions remain the same...)

if \_\_name\_\_ == "\_\_main\_\_":

# Reading API key and ticker list

try:

# Check if the key.txt file exists

if not os.path.exists('key.txt'):

raise FileNotFoundError("API key file 'key.txt' not found.")

# Load the API key

key = pd.read\_csv('key.txt', header=None)[0][0]

# Check if the tickers.txt file exists

if not os.path.exists('tickers.txt'):

raise FileNotFoundError("Ticker list file 'tickers.txt' not found.")

# Load the tickers from tickers.txt

tickers = pd.read\_csv('tickers.txt', header=None)[0]

if tickers.empty:

raise ValueError("The tickers list is empty. Please provide valid tickers in 'tickers.txt'.")

print(f"API Key: {key}")

print(f"Tickers: {tickers}")

except FileNotFoundError as e:

print(f"File not found: {str(e)}")

except pd.errors.EmptyDataError:

print("Error: 'tickers.txt' or 'key.txt' is empty.")

except Exception as e:

print(f"Error reading files: {str(e)}")

# Check if 'tickers' was successfully defined

if 'tickers' in locals():

# Process each ticker

for ticker in tickers:

print(f"\nProcessing ticker: {ticker}")

IS = get\_income\_statement(ticker=ticker, limit=6, key=key, period='annual')

# ... (Other API calls and Excel writing steps remain the same) ...

else:

print("Error: 'tickers' is not defined. Please check the 'tickers.txt' file.")

# -\*- coding: utf-8 -\*-

"""

SEC Filing Scraper

@author: AdamGetbags

"""

# import modules

import requests

import pandas as pd

import matplotlib.pyplot as plt

# create request header

headers = {'User-Agent': "email@address.com"}

# get all companies data from SEC

company\_tickers\_url = "https://www.sec.gov/files/company\_tickers.json"

company\_tickers\_response = requests.get(company\_tickers\_url, headers=headers)

# format response to dictionary and get first key/value

company\_tickers\_dict = company\_tickers\_response.json()

first\_entry = company\_tickers\_dict['0']

# parse CIK without leading zeros

direct\_cik = first\_entry['cik\_str']

# dictionary to dataframe

company\_data = pd.DataFrame.from\_dict(company\_tickers\_dict, orient='index')

# add leading zeros to CIK

company\_data['cik\_str'] = company\_data['cik\_str'].astype(str).str.zfill(10)

# review first entry

print(company\_data.head(1))

# Get the first company's CIK

cik = company\_data.iloc[0]['cik\_str']

# get company specific filing metadata from SEC's EDGAR API

filing\_metadata\_url = f'https://data.sec.gov/submissions/CIK{cik}.json'

filing\_metadata\_response = requests.get(filing\_metadata\_url, headers=headers)

# review keys

filing\_metadata\_json = filing\_metadata\_response.json()

print(filing\_metadata\_json.keys())

# parse filings

filings = filing\_metadata\_json['filings']['recent']

# dictionary to dataframe

all\_forms = pd.DataFrame.from\_dict(filings)

# review columns and extract specific filing metadata

print(all\_forms[['accessionNumber', 'reportDate', 'form']].head(50))

# Example: get metadata for 10-Q filings

form\_10q\_metadata = all\_forms[all\_forms['form'] == '10-Q']

print(form\_10q\_metadata.head())

# get company facts data (e.g., stock shares outstanding)

company\_facts\_url = f'https://data.sec.gov/api/xbrl/companyfacts/CIK{cik}.json'

company\_facts\_response = requests.get(company\_facts\_url, headers=headers)

# parse facts

company\_facts\_json = company\_facts\_response.json()

print(company\_facts\_json['facts'].keys())

# example: get stock shares outstanding

stock\_shares\_outstanding = company\_facts\_json['facts']['dei']['EntityCommonStockSharesOutstanding']['units']['shares'][0]

print(stock\_shares\_outstanding)

# get company concept data (e.g., Assets from filings)

company\_concept\_url = f'https://data.sec.gov/api/xbrl/companyconcept/CIK{cik}/us-gaap/Assets.json'

company\_concept\_response = requests.get(company\_concept\_url, headers=headers)

# parse concept data

company\_concept\_json = company\_concept\_response.json()

assets\_data = pd.DataFrame.from\_dict(company\_concept\_json['units']['USD'])

# filter assets data for 10-Q forms and reset index

assets\_10k = assets\_data[assets\_data['form'] == '10-K'].reset\_index(drop=True)

# plot assets over time for 10-Q filings

assets\_10k.plot(x='end', y='val', kind='line', title='Assets (10-K) Over Time')

plt.xlabel('Filing Date')

plt.ylabel('Assets (USD)')

plt.show()

"""

This is a script that will scrape financial information using the

FinancialModelingPrep API.

Website: https://financialmodelingprep.com/developer/

Free plan with 250 requests per day.

Includes:

- Income Statements

- Balance Sheets

- Cash Flow Statements

- Financial Ratios

- Key Metrics

- Daily Prices

- Enterprise Value

Parameter specification:

- ticker: Company stock name (e.g., AAPL for Apple)

- limit: Number of records you'd like (e.g., for annual, 6 will give 6 years)

- key: API key generated from the Financial Modeling Prep account

- period: 'annual' or 'quarter'

"""

import Financial\_Data\_Scraping as fds

import pandas as pd

if \_\_name\_\_ == "\_\_main\_\_":

"""Running the scraper to obtain financial data."""

# Load API key from a file

key = pd.read\_csv('key.txt', header=None)[0][0]

# Load list of tickers from a file

tickers = pd.read\_csv('tickers.txt', header=None)[0]

# Loop through each ticker to scrape data

for ticker in tickers:

print(f"Starting scraping for: {ticker}")

# Scrape financial data using the custom module 'Financial\_Data\_Scraping'

IS = fds.get\_income\_statement(ticker=ticker, limit=6, key=key, period='annual')

BS = fds.get\_balance\_sheet(ticker=ticker, limit=6, key=key, period='annual')

CF = fds.get\_cash\_flow\_statement(ticker=ticker, limit=6, key=key, period='annual')

FR = fds.get\_financial\_ratios(ticker=ticker, limit=6, key=key, period='annual')

KM = fds.get\_key\_metrics(ticker=ticker, limit=6, key=key, period='annual')

P = fds.get\_daily\_prices(ticker=ticker, timeseries=5 \* 261, key=key) # 5 years of daily prices

EV = fds.get\_enterprise\_value(ticker=ticker, rate=5 \* 261, key=key, period='annual') # 5 years of enterprise value

# Creating an Excel writer object to save data to an Excel file

writer = pd.ExcelWriter(f'{ticker}.xlsx', engine='xlsxwriter')

# Write data to separate sheets within the Excel file

IS.to\_excel(writer, sheet\_name='Income Statement')

BS.to\_excel(writer, sheet\_name='Balance Sheet Statement')

CF.to\_excel(writer, sheet\_name='Cash Flow Statement')

FR.to\_excel(writer, sheet\_name='Financial Ratios')

KM.to\_excel(writer, sheet\_name='Key Metrics')

P.to\_excel(writer, sheet\_name='Daily Prices')

EV.to\_excel(writer, sheet\_name='Enterprise Value')

# Save the Excel file

writer.save()

print(f'Finished scraping for: {ticker}')

print("All tickers processed.")

import pandas as pd

import numpy as np

# Load existing dataset

file\_path = r'synthetic\_dataset.csv'

try:

df = pd.read\_csv(file\_path)

print("Original data loaded successfully!")

except FileNotFoundError as e:

print(f"Error: {e}")

exit()

# Function to generate synthetic data

def generate\_synthetic\_data(df, n\_samples=500):

synthetic\_data = []

for \_ in range(n\_samples):

# Randomly select values based on existing data distributions

timestamp = pd.Timestamp.now() # You can modify this to create realistic timestamps

name = f"Synthetic\_{np.random.randint(1000)}" # Generate a synthetic name

gender = np.random.choice(df['Gender'].dropna().unique()) # Sample gender

section = np.random.choice(df['Section'].dropna().unique()) # Sample section

age = np.random.randint(17, 25) # Assuming age range from 17 to 25

collision\_types = np.random.choice(

['Body to Ground', 'Ball to body Impact', 'Head to Body collisions', 'None'],

size=np.random.randint(1, 4), replace=False

)

collision\_types\_str = ';'.join(collision\_types)

injuries = np.random.choice(

['Knee Injury', 'Head Injury', 'Ligament Tear', 'Dislocation of joint', 'None'],

size=np.random.randint(1, 3), replace=False

)

injuries\_str = ';'.join(injuries)

symptoms = np.random.choice(

['Swelling', 'Pain', 'Bruising', 'Weakness', 'None'],

size=np.random.randint(1, 4), replace=False

)

symptoms\_str = ';'.join(symptoms)

knee\_injury\_overtime = np.random.choice(['Yes', 'No'])

knee\_injury\_instant = np.random.choice(['Yes', 'No'])

synthetic\_data.append([

timestamp, name, gender, section, age,

collision\_types\_str, injuries\_str,

symptoms\_str, knee\_injury\_overtime,

knee\_injury\_instant

])

# Create a DataFrame from synthetic data

columns = [

"Timestamp", "Name", "Gender", "Section", "Age",

"Have you experienced these collision during any sports activity?",

"What kind of injuries have you experienced during game?",

"Symptoms experienced by player after injury?",

"Did you experience knee injury overtime?",

"Did you experience knee injury at one instant of the game?"

]

synthetic\_df = pd.DataFrame(synthetic\_data, columns=columns)

return synthetic\_df

# Generate synthetic data

synthetic\_df = generate\_synthetic\_data(df)

# Combine original and synthetic datasets if needed

combined\_df = pd.concat([df, synthetic\_df], ignore\_index=True)

# Save the combined dataset to an Excel file

output\_file\_path = r'C:\Users\User\Desktop\project 1\synthetic\_dataset.xlsx'

combined\_df.to\_excel(output\_file\_path, index=False)

print(f"Combined dataset saved successfully to {output\_file\_path}")

import requests

import pandas as pd

import matplotlib.pyplot as plt

from datetime import datetime, timedelta

# Define the CIK for Microsoft (0000789019)

cik = '0000789019'

# Set headers for the request

headers = {'User-Agent': "joshjothom05@gmail.com"}

# Get filing metadata for Microsoft

filingMetadata = requests.get(

f'https://data.sec.gov/submissions/CIK{cik}.json',

headers=headers

)

# Convert filing metadata to DataFrame

allForms = pd.DataFrame.from\_dict(filingMetadata.json()['filings']['recent'])

# Filter for only 10-K forms in the last 5 years

allForms['reportDate'] = pd.to\_datetime(allForms['reportDate'], errors='coerce')

five\_years\_ago = datetime.now() - timedelta(days=5\*365)

tenK\_filings = allForms[(allForms['form'] == '10-K') & (allForms['reportDate'] > five\_years\_ago)]

# Get company facts data

companyFacts = requests.get(

f'https://data.sec.gov/api/xbrl/companyfacts/CIK{cik}.json',

headers=headers

)

# Retrieve 'us-gaap: Assets' data

companyConcept = requests.get(

f'https://data.sec.gov/api/xbrl/companyconcept/CIK{cik}/us-gaap/Assets.json',

headers=headers

)

# Convert assets data to DataFrame

assetsData = pd.DataFrame.from\_dict(companyConcept.json()['units']['USD'])

# Filter only 10-K forms and dates in the past 5 years

assetsData['end'] = pd.to\_datetime(assetsData['end'], errors='coerce')

assets10K = assetsData[(assetsData['form'] == '10-K') & (assetsData['end'] > five\_years\_ago)]

assets10K = assets10K.reset\_index(drop=True)

# Plot assets over time for 10-K filings

assets10K.plot(x='end', y='val', title='Microsoft Annual Assets (10-K Filings in Last 5 Years)', marker='o')

plt.xlabel('Year')

plt.ylabel('Assets (in USD)')

plt.grid(True)

plt.show()

### edgar video

import pandas as pd

import requests

import edgar\_functions as edgar

from headers import headers

import os

def get\_facts(ticker, headers=headers):

cik = edgar.cik\_matching\_ticker(ticker)

url = f"https://data.sec.gov/api/xbrl/companyfacts/CIK{cik}.json"

company\_facts = requests.get(url, headers=headers).json()

return company\_facts

facts = get\_facts("WSM")

facts["facts"]["us-gaap"]["AccountsPayableCurrent"]['units']['USD']

def facts\_DF(ticker, headers=headers):

facts = get\_facts(ticker, headers)

us\_gaap\_data = facts["facts"]["us-gaap"]

df\_data = []

for fact, details in us\_gaap\_data.items():

for unit in details["units"]:

for item in details["units"][unit]:

row = item.copy()

row["fact"] = fact

df\_data.append(row)

df = pd.DataFrame(df\_data)

df["end"] = pd.to\_datetime(df["end"])

df["start"] = pd.to\_datetime(df["start"])

df = df.drop\_duplicates(subset=["fact", "end", "val"])

df.set\_index("end", inplace=True)

labels\_dict = {fact: details["label"] for fact, details in us\_gaap\_data.items()}

return df, labels\_dict

facts, label\_dict = facts\_DF("WSM")

def annual\_facts(ticker, headers=headers):

accession\_nums = edgar.get\_filtered\_filings(

ticker, ten\_k=True, just\_accession\_numbers=True

)

df, label\_dict = facts\_DF(ticker, headers)

ten\_k = df[df["accn"].isin(accession\_nums)]

ten\_k = ten\_k[ten\_k.index.isin(accession\_nums.index)]

pivot = ten\_k.pivot\_table(values="val", columns="fact", index="end")

pivot.rename(columns=label\_dict, inplace=True)

return pivot.T

def quarterly\_facts(ticker, headers=headers):

accession\_nums = get\_filtered\_filings(

ticker, ten\_k=False, just\_accession\_numbers=True

)

df, label\_dict = facts\_DF(ticker, headers)

ten\_q = df[df["accn"].isin(accession\_nums)]

ten\_q = ten\_q[ten\_q.index.isin(accession\_nums.index)].reset\_index(drop=False)

ten\_q = ten\_q.drop\_duplicates(subset=["fact", "end"], keep="last")

pivot = ten\_q.pivot\_table(values="val", columns="fact", index="end")

pivot.rename(columns=label\_dict, inplace=True)

return pivot.T

quarterly = quarterly\_facts('WSM')

def save\_dataframe\_to\_csv(dataframe, folder\_name, ticker, statement\_name, frequency):

directory\_path = os.path.join(folder\_name, ticker)

os.makedirs(directory\_path, exist\_ok=True)

file\_path = os.path.join(directory\_path, f"{statement\_name}\_{frequency}.csv")

dataframe.to\_csv(file\_path)

return None

import os

import pandas as pd

import requests

from headers import headers # change to your own headers file or add variable in code

def cik\_matching\_ticker(ticker, headers=headers):

ticker = ticker.upper().replace(".", "-")

ticker\_json = requests.get(

"https://www.sec.gov/files/company\_tickers.json", headers=headers

).json()

for company in ticker\_json.values():

if company["ticker"] == ticker:

cik = str(company["cik\_str"]).zfill(10)

return cik

raise ValueError(f"Ticker {ticker} not found in SEC database")

def get\_submission\_data\_for\_ticker(ticker, headers=headers, only\_filings\_df=False):

"""

Get the data in json form for a given ticker. For example: 'cik', 'entityType', 'sic', 'sicDescription', 'insiderTransactionForOwnerExists', 'insiderTransactionForIssuerExists', 'name', 'tickers', 'exchanges', 'ein', 'description', 'website', 'investorWebsite', 'category', 'fiscalYearEnd', 'stateOfIncorporation', 'stateOfIncorporationDescription', 'addresses', 'phone', 'flags', 'formerNames', 'filings'

Args:

ticker (str): The ticker symbol of the company.

Returns:

json: The submissions for the company.

"""

cik = cik\_matching\_ticker(ticker)

headers = headers

url = f"https://data.sec.gov/submissions/CIK{cik}.json"

company\_json = requests.get(url, headers=headers).json()

if only\_filings\_df:

return pd.DataFrame(company\_json["filings"]["recent"])

else:

return company\_json

def get\_filtered\_filings(

ticker, ten\_k=True, just\_accession\_numbers=False, headers=headers

):

company\_filings\_df = get\_submission\_data\_for\_ticker(

ticker, only\_filings\_df=True, headers=headers

)

if ten\_k:

df = company\_filings\_df[company\_filings\_df["form"] == "10-K"]

else:

df = company\_filings\_df[company\_filings\_df["form"] == "10-Q"]

if just\_accession\_numbers:

df = df.set\_index("reportDate")

accession\_df = df["accessionNumber"]

return accession\_df

else:

return df

def get\_facts(ticker, headers=headers):

cik = cik\_matching\_ticker(ticker)

url = f"https://data.sec.gov/api/xbrl/companyfacts/CIK{cik}.json"

company\_facts = requests.get(url, headers=headers).json()

return company\_facts

def facts\_DF(ticker, headers=headers):

facts = get\_facts(ticker, headers)

us\_gaap\_data = facts["facts"]["us-gaap"]

df\_data = []

for fact, details in us\_gaap\_data.items():

for unit in details["units"]:

for item in details["units"][unit]:

row = item.copy()

row["fact"] = fact

df\_data.append(row)

df = pd.DataFrame(df\_data)

df["end"] = pd.to\_datetime(df["end"])

df["start"] = pd.to\_datetime(df["start"])

df = df.drop\_duplicates(subset=["fact", "end", "val"])

df.set\_index("end", inplace=True)

labels\_dict = {fact: details["label"] for fact, details in us\_gaap\_data.items()}

return df, labels\_dict

def annual\_facts(ticker, headers=headers):

accession\_nums = get\_filtered\_filings(

ticker, ten\_k=True, just\_accession\_numbers=True

)

df, label\_dict = facts\_DF(ticker, headers)

ten\_k = df[df["accn"].isin(accession\_nums)]

ten\_k = ten\_k[ten\_k.index.isin(accession\_nums.index)]

pivot = ten\_k.pivot\_table(values="val", columns="fact", index="end")

pivot.rename(columns=label\_dict, inplace=True)

return pivot.T

def quarterly\_facts(ticker, headers=headers):

accession\_nums = get\_filtered\_filings(

ticker, ten\_k=False, just\_accession\_numbers=True

)

df, label\_dict = facts\_DF(ticker, headers)

ten\_q = df[df["accn"].isin(accession\_nums)]

ten\_q = ten\_q[ten\_q.index.isin(accession\_nums.index)].reset\_index(drop=False)

ten\_q = ten\_q.drop\_duplicates(subset=["fact", "end"], keep="last")

pivot = ten\_q.pivot\_table(values="val", columns="fact", index="end")

pivot.rename(columns=label\_dict, inplace=True)

return pivot.T

def save\_dataframe\_to\_csv(dataframe, folder\_name, ticker, statement\_name, frequency):

directory\_path = os.path.join(folder\_name, ticker)

os.makedirs(directory\_path, exist\_ok=True)

file\_path = os.path.join(directory\_path, f"{statement\_name}\_{frequency}.csv")

dataframe.to\_csv(file\_path)

return None

import pandas as pd

import requests

from headers import headers # change to your own headers file or add variable in code

def cik\_matching\_ticker(ticker, headers=headers):

ticker = ticker.upper().replace(".", "-")

ticker\_json = requests.get(

"https://www.sec.gov/files/company\_tickers.json", headers=headers

).json()

for company in ticker\_json.values():

if company["ticker"] == ticker:

cik = str(company["cik\_str"]).zfill(10)

return cik

raise ValueError(f"Ticker {ticker} not found in SEC database")

def get\_submission\_data\_for\_ticker(ticker, headers=headers, only\_filings\_df=False):

"""

Get the data in json form for a given ticker. For example: 'cik', 'entityType', 'sic', 'sicDescription', 'insiderTransactionForOwnerExists', 'insiderTransactionForIssuerExists', 'name', 'tickers', 'exchanges', 'ein', 'description', 'website', 'investorWebsite', 'category', 'fiscalYearEnd', 'stateOfIncorporation', 'stateOfIncorporationDescription', 'addresses', 'phone', 'flags', 'formerNames', 'filings'

Args:

ticker (str): The ticker symbol of the company.

Returns:

json: The submissions for the company.

Raises:

ValueError: If ticker is not a string.

"""

cik = cik\_matching\_ticker(ticker)

headers = headers

url = f"https://data.sec.gov/submissions/CIK{cik}.json"

company\_json = requests.get(url, headers=headers).json()

if only\_filings\_df:

return pd.DataFrame(company\_json["filings"]["recent"])

else:

return company\_json

def get\_filtered\_filings(

ticker, ten\_k=True, just\_accession\_numbers=False, headers=headers

):

company\_filings\_df = get\_submission\_data\_for\_ticker(

ticker, only\_filings\_df=True, headers=headers

)

if ten\_k:

df = company\_filings\_df[company\_filings\_df["form"] == "10-K"]

else:

df = company\_filings\_df[company\_filings\_df["form"] == "10-Q"]

if just\_accession\_numbers:

df = df.set\_index("reportDate")

accession\_df = df["accessionNumber"]

return accession\_df

else:

return df

import os

import pandas as pd

import numpy as np # make sure to add

import requests

from bs4 import BeautifulSoup

import logging # make sure to add

import calendar # make sure to add

from headers import headers # change to your own headers file or add variable in code

pd.options.display.float\_format = (

lambda x: "{:,.0f}".format(x) if int(x) == x else "{:,.2f}".format(x)

)

statement\_keys\_map = {

"balance\_sheet": [

"balance sheet",

"balance sheets",

"statement of financial position",

"consolidated balance sheets",

"consolidated balance sheet",

"consolidated financial position",

"consolidated balance sheets - southern",

"consolidated statements of financial position",

"consolidated statement of financial position",

"consolidated statements of financial condition",

"combined and consolidated balance sheet",

"condensed consolidated balance sheets",

"consolidated balance sheets, as of december 31",

"dow consolidated balance sheets",

"consolidated balance sheets (unaudited)",

],

"income\_statement": [

"income statement",

"income statements",

"statement of earnings (loss)",

"statements of consolidated income",

"consolidated statements of operations",

"consolidated statement of operations",

"consolidated statements of earnings",

"consolidated statement of earnings",

"consolidated statements of income",

"consolidated statement of income",

"consolidated income statements",

"consolidated income statement",

"condensed consolidated statements of earnings",

"consolidated results of operations",

"consolidated statements of income (loss)",

"consolidated statements of income - southern",

"consolidated statements of operations and comprehensive income",

"consolidated statements of comprehensive income",

],

"cash\_flow\_statement": [

"cash flows statement",

"cash flows statements",

"statement of cash flows",

"statements of consolidated cash flows",

"consolidated statements of cash flows",

"consolidated statement of cash flows",

"consolidated statement of cash flow",

"consolidated cash flows statements",

"consolidated cash flow statements",

"condensed consolidated statements of cash flows",

"consolidated statements of cash flows (unaudited)",

"consolidated statements of cash flows - southern",

],

}

def cik\_matching\_ticker(ticker, headers=headers):

ticker = ticker.upper().replace(".", "-")

ticker\_json = requests.get(

"https://www.sec.gov/files/company\_tickers.json", headers=headers

).json()

for company in ticker\_json.values():

if company["ticker"] == ticker:

cik = str(company["cik\_str"]).zfill(10)

return cik

raise ValueError(f"Ticker {ticker} not found in SEC database")

def get\_submission\_data\_for\_ticker(ticker, headers=headers, only\_filings\_df=False):

"""

Get the data in json form for a given ticker. For example: 'cik', 'entityType', 'sic', 'sicDescription', 'insiderTransactionForOwnerExists', 'insiderTransactionForIssuerExists', 'name', 'tickers', 'exchanges', 'ein', 'description', 'website', 'investorWebsite', 'category', 'fiscalYearEnd', 'stateOfIncorporation', 'stateOfIncorporationDescription', 'addresses', 'phone', 'flags', 'formerNames', 'filings'

Args:

ticker (str): The ticker symbol of the company.

Returns:

json: The submissions for the company.

"""

cik = cik\_matching\_ticker(ticker)

headers = headers

url = f"https://data.sec.gov/submissions/CIK{cik}.json"

company\_json = requests.get(url, headers=headers).json()

if only\_filings\_df:

return pd.DataFrame(company\_json["filings"]["recent"])

else:

return company\_json

def get\_filtered\_filings(

ticker, ten\_k=True, just\_accession\_numbers=False, headers=None

):

"""

Retrieves either 10-K or 10-Q filings for a given ticker and optionally returns just accession numbers.

Args:

ticker (str): Stock ticker symbol.

ten\_k (bool): If True, fetches 10-K filings; otherwise, fetches 10-Q filings.

just\_accession\_numbers (bool): If True, returns only accession numbers; otherwise, returns full data.

headers (dict): Headers for HTTP request.

Returns:

DataFrame or Series: DataFrame of filings or Series of accession numbers.

"""

# Fetch submission data for the given ticker

company\_filings\_df = get\_submission\_data\_for\_ticker(

ticker, only\_filings\_df=True, headers=headers

)

# Filter for 10-K or 10-Q forms

df = company\_filings\_df[company\_filings\_df["form"] == ("10-K" if ten\_k else "10-Q")]

# Return accession numbers if specified

if just\_accession\_numbers:

df = df.set\_index("reportDate")

accession\_df = df["accessionNumber"]

return accession\_df

else:

return df

def get\_facts(ticker, headers=None):

"""

Retrieves company facts for a given ticker.

Args:

ticker (str): Stock ticker symbol.

headers (dict): Headers for HTTP request.

Returns:

dict: Company facts in JSON format.

"""

# Get CIK number matching the ticker

cik = cik\_matching\_ticker(ticker)

# Construct URL for company facts

url = f"https://data.sec.gov/api/xbrl/companyfacts/CIK{cik}.json"

# Fetch and return company facts

company\_facts = requests.get(url, headers=headers).json()

return company\_facts

def facts\_DF(ticker, headers=None):

"""

Converts company facts into a DataFrame.

Args:

ticker (str): Stock ticker symbol.

headers (dict): Headers for HTTP request.

Returns:

tuple: DataFrame of facts and a dictionary of labels.

"""

# Retrieve facts data

facts = get\_facts(ticker, headers)

us\_gaap\_data = facts["facts"]["us-gaap"]

df\_data = []

# Process each fact and its details

for fact, details in us\_gaap\_data.items():

for unit in details["units"]:

for item in details["units"][unit]:

row = item.copy()

row["fact"] = fact

df\_data.append(row)

df = pd.DataFrame(df\_data)

# Convert 'end' and 'start' to datetime

df["end"] = pd.to\_datetime(df["end"])

df["start"] = pd.to\_datetime(df["start"])

# Drop duplicates and set index

df = df.drop\_duplicates(subset=["fact", "end", "val"])

df.set\_index("end", inplace=True)

# Create a dictionary of labels for facts

labels\_dict = {fact: details["label"] for fact, details in us\_gaap\_data.items()}

return df, labels\_dict

def annual\_facts(ticker, headers=None):

"""

Fetches and processes annual (10-K) financial facts for a given ticker.

Args:

ticker (str): Stock ticker symbol.

headers (dict): Headers for HTTP request.

Returns:

DataFrame: Transposed pivot table of annual financial facts.

"""

# Get accession numbers for 10-K filings

accession\_nums = get\_filtered\_filings(

ticker, ten\_k=True, just\_accession\_numbers=True, headers=headers

)

# Extract and process facts data

df, label\_dict = facts\_DF(ticker, headers)

# Filter data for 10-K filings

ten\_k = df[df["accn"].isin(accession\_nums)]

ten\_k = ten\_k[ten\_k.index.isin(accession\_nums.index)]

# Pivot and format the data

pivot = ten\_k.pivot\_table(values="val", columns="fact", index="end")

pivot.rename(columns=label\_dict, inplace=True)

return pivot.T

def quarterly\_facts(ticker, headers=None):

"""

Fetches and processes quarterly (10-Q) financial facts for a given ticker.

Args:

ticker (str): Stock ticker symbol.

headers (dict): Headers for HTTP request.

Returns:

DataFrame: Transposed pivot table of quarterly financial facts.

"""

# Get accession numbers for 10-Q filings

accession\_nums = get\_filtered\_filings(

ticker, ten\_k=False, just\_accession\_numbers=True, headers=headers

)

# Extract and process facts data

df, label\_dict = facts\_DF(ticker, headers)

# Filter data for 10-Q filings

ten\_q = df[df["accn"].isin(accession\_nums)]

ten\_q = ten\_q[ten\_q.index.isin(accession\_nums.index)].reset\_index(drop=False)

# Remove duplicate entries

ten\_q = ten\_q.drop\_duplicates(subset=["fact", "end"], keep="last")

# Pivot and format the data

pivot = ten\_q.pivot\_table(values="val", columns="fact", index="end")

pivot.rename(columns=label\_dict, inplace=True)

return pivot.T

def save\_dataframe\_to\_csv(dataframe, folder\_name, ticker, statement\_name, frequency):

"""

Saves a given DataFrame to a CSV file in a specified directory.

Args:

dataframe (DataFrame): The DataFrame to be saved.

folder\_name (str): The folder name where the CSV file will be saved.

ticker (str): Stock ticker symbol.

statement\_name (str): Name of the financial statement.

frequency (str): Frequency of the financial data (e.g., annual, quarterly).

Returns:

None

"""

# Create directory path

directory\_path = os.path.join(folder\_name, ticker)

os.makedirs(directory\_path, exist\_ok=True)

# Construct file path and save DataFrame

file\_path = os.path.join(directory\_path, f"{statement\_name}\_{frequency}.csv")

dataframe.to\_csv(file\_path)

def \_get\_file\_name(report):

"""

Extracts the file name from an XML report tag.

Args:

report (Tag): BeautifulSoup tag representing the report.

Returns:

str: File name extracted from the tag.

"""

html\_file\_name\_tag = report.find("HtmlFileName")

xml\_file\_name\_tag = report.find("XmlFileName")

# Return the appropriate file name

if html\_file\_name\_tag:

return html\_file\_name\_tag.text

elif xml\_file\_name\_tag:

return xml\_file\_name\_tag.text

else:

return ""

def \_is\_statement\_file(short\_name\_tag, long\_name\_tag, file\_name):

"""

Determines if a given file is a financial statement file.

Args:

short\_name\_tag (Tag): BeautifulSoup tag for the short name.

long\_name\_tag (Tag): BeautifulSoup tag for the long name.

file\_name (str): Name of the file.

Returns:

bool: True if it's a statement file, False otherwise.

"""

return (

short\_name\_tag is not None

and long\_name\_tag is not None

and file\_name # Ensure file\_name is not an empty string

and "Statement" in long\_name\_tag.text

)

def get\_statement\_file\_names\_in\_filing\_summary(ticker, accession\_number, headers=None):

"""

Retrieves file names of financial statements from a filing summary.

Args:

ticker (str): Stock ticker symbol.

accession\_number (str): SEC filing accession number.

headers (dict): Headers for HTTP request.

Returns:

dict: Dictionary mapping statement types to their file names.

"""

try:

# Set up request session and get filing summary

session = requests.Session()

cik = cik\_matching\_ticker(ticker)

base\_link = f"https://www.sec.gov/Archives/edgar/data/{cik}/{accession\_number}"

filing\_summary\_link = f"{base\_link}/FilingSummary.xml"

filing\_summary\_response = session.get(

filing\_summary\_link, headers=headers

).content.decode("utf-8")

# Parse the filing summary

filing\_summary\_soup = BeautifulSoup(filing\_summary\_response, "lxml-xml")

statement\_file\_names\_dict = {}

# Extract file names for statements

for report in filing\_summary\_soup.find\_all("Report"):

file\_name = \_get\_file\_name(report)

short\_name, long\_name = report.find("ShortName"), report.find("LongName")

if \_is\_statement\_file(short\_name, long\_name, file\_name):

statement\_file\_names\_dict[short\_name.text.lower()] = file\_name

return statement\_file\_names\_dict

except requests.RequestException as e:

print(f"An error occurred: {e}")

return {}

def get\_statement\_soup(

ticker, accession\_number, statement\_name, headers, statement\_keys\_map

):

"""

Retrieves the BeautifulSoup object for a specific financial statement.

Args:

ticker (str): Stock ticker symbol.

accession\_number (str): SEC filing accession number.

statement\_name (str): has to be 'balance\_sheet', 'income\_statement', 'cash\_flow\_statement'

headers (dict): Headers for HTTP request.

statement\_keys\_map (dict): Mapping of statement names to keys.

Returns:

BeautifulSoup: Parsed HTML/XML content of the financial statement.

Raises:

ValueError: If the statement file name is not found or if there is an error fetching the statement.

"""

session = requests.Session()

cik = cik\_matching\_ticker(ticker)

base\_link = f"https://www.sec.gov/Archives/edgar/data/{cik}/{accession\_number}"

# Get statement file names

statement\_file\_name\_dict = get\_statement\_file\_names\_in\_filing\_summary(

ticker, accession\_number, headers

)

statement\_link = None

# Find the specific statement link

for possible\_key in statement\_keys\_map.get(statement\_name.lower(), []):

file\_name = statement\_file\_name\_dict.get(possible\_key.lower())

if file\_name:

statement\_link = f"{base\_link}/{file\_name}"

break

if not statement\_link:

raise ValueError(f"Could not find statement file name for {statement\_name}")

# Fetch the statement

try:

statement\_response = session.get(statement\_link, headers=headers)

statement\_response.raise\_for\_status() # Check for a successful request

# Parse and return the content

if statement\_link.endswith(".xml"):

return BeautifulSoup(

statement\_response.content, "lxml-xml", from\_encoding="utf-8"

)

else:

return BeautifulSoup(statement\_response.content, "lxml")

except requests.RequestException as e:

raise ValueError(f"Error fetching the statement: {e}")

def extract\_columns\_values\_and\_dates\_from\_statement(soup):

"""

Extracts columns, values, and dates from an HTML soup object representing a financial statement.

Args:

soup (BeautifulSoup): The BeautifulSoup object of the HTML document.

Returns:

tuple: Tuple containing columns, values\_set, and date\_time\_index.

"""

columns = []

values\_set = []

date\_time\_index = get\_datetime\_index\_dates\_from\_statement(soup)

for table in soup.find\_all("table"):

unit\_multiplier = 1

special\_case = False

# Check table headers for unit multipliers and special cases

table\_header = table.find("th")

if table\_header:

header\_text = table\_header.get\_text()

# Determine unit multiplier based on header text

if "in Thousands" in header\_text:

unit\_multiplier = 1

elif "in Millions" in header\_text:

unit\_multiplier = 1000

# Check for special case scenario

if "unless otherwise specified" in header\_text:

special\_case = True

# Process each row of the table

for row in table.select("tr"):

onclick\_elements = row.select("td.pl a, td.pl.custom a")

if not onclick\_elements:

continue

# Extract column title from 'onclick' attribute

onclick\_attr = onclick\_elements[0]["onclick"]

column\_title = onclick\_attr.split("defref\_")[-1].split("',")[0]

columns.append(column\_title)

# Initialize values array with NaNs

values = [np.NaN] \* len(date\_time\_index)

# Process each cell in the row

for i, cell in enumerate(row.select("td.text, td.nump, td.num")):

if "text" in cell.get("class"):

continue

# Clean and parse cell value

value = keep\_numbers\_and\_decimals\_only\_in\_string(

cell.text.replace("$", "")

.replace(",", "")

.replace("(", "")

.replace(")", "")

.strip()

)

if value:

value = float(value)

# Adjust value based on special case and cell class

if special\_case:

value /= 1000

else:

if "nump" in cell.get("class"):

values[i] = value \* unit\_multiplier

else:

values[i] = -value \* unit\_multiplier

values\_set.append(values)

return columns, values\_set, date\_time\_index

def get\_datetime\_index\_dates\_from\_statement(soup: BeautifulSoup) -> pd.DatetimeIndex:

"""

Extracts datetime index dates from the HTML soup object of a financial statement.

Args:

soup (BeautifulSoup): The BeautifulSoup object of the HTML document.

Returns:

pd.DatetimeIndex: A Pandas DatetimeIndex object containing the extracted dates.

"""

table\_headers = soup.find\_all("th", {"class": "th"})

dates = [str(th.div.string) for th in table\_headers if th.div and th.div.string]

dates = [standardize\_date(date).replace(".", "") for date in dates]

index\_dates = pd.to\_datetime(dates)

return index\_dates

def standardize\_date(date: str) -> str:

"""

Standardizes date strings by replacing abbreviations with full month names.

Args:

date (str): The date string to be standardized.

Returns:

str: The standardized date string.

"""

for abbr, full in zip(calendar.month\_abbr[1:], calendar.month\_name[1:]):

date = date.replace(abbr, full)

return date

def keep\_numbers\_and\_decimals\_only\_in\_string(mixed\_string: str):

"""

Filters a string to keep only numbers and decimal points.

Args:

mixed\_string (str): The string containing mixed characters.

Returns:

str: String containing only numbers and decimal points.

"""

num = "1234567890."

allowed = list(filter(lambda x: x in num, mixed\_string))

return "".join(allowed)

def create\_dataframe\_of\_statement\_values\_columns\_dates(

values\_set, columns, index\_dates

) -> pd.DataFrame:

"""

Creates a DataFrame from statement values, columns, and index dates.

Args:

values\_set (list): List of values for each column.

columns (list): List of column names.

index\_dates (pd.DatetimeIndex): DatetimeIndex for the DataFrame index.

Returns:

pd.DataFrame: DataFrame constructed from the given data.

"""

transposed\_values\_set = list(zip(\*values\_set))

df = pd.DataFrame(transposed\_values\_set, columns=columns, index=index\_dates)

return df

def process\_one\_statement(ticker, accession\_number, statement\_name):

"""

Processes a single financial statement identified by ticker, accession number, and statement name.

Args:

ticker (str): The stock ticker.

accession\_number (str): The SEC accession number.

statement\_name (str): Name of the financial statement.

Returns:

pd.DataFrame or None: DataFrame of the processed statement or None if an error occurs.

"""

try:

# Fetch the statement HTML soup

soup = get\_statement\_soup(

ticker,

accession\_number,

statement\_name,

headers=headers,

statement\_keys\_map=statement\_keys\_map,

)

except Exception as e:

logging.error(

f"Failed to get statement soup: {e} for accession number: {accession\_number}"

)

return None

if soup:

try:

# Extract data and create DataFrame

columns, values, dates = extract\_columns\_values\_and\_dates\_from\_statement(

soup

)

df = create\_dataframe\_of\_statement\_values\_columns\_dates(

values, columns, dates

)

if not df.empty:

# Remove duplicate columns

df = df.T.drop\_duplicates()

else:

logging.warning(

f"Empty DataFrame for accession number: {accession\_number}"

)

return None

return df

except Exception as e:

logging.error(f"Error processing statement: {e}")

return None

def get\_label\_dictionary(ticker, headers):

facts = get\_facts(ticker, headers)

us\_gaap\_data = facts["facts"]["us-gaap"]

labels\_dict = {fact: details["label"] for fact, details in us\_gaap\_data.items()}

return labels\_dict

def rename\_statement(statement, label\_dictionary):

# Extract the part after the first "\_" and then map it using the label dictionary

statement.index = statement.index.map(

lambda x: label\_dictionary.get(x.split("\_", 1)[-1], x)

)

return statement

from edgar\_functions import \*

from headers import headers

ticker = "WSM"

accn = get\_filtered\_filings(

ticker, ten\_k=True, just\_accession\_numbers=False, headers=headers

)

acc\_num = accn['accessionNumber'].iloc[2].replace("-", "")

soup = get\_statement\_soup(

ticker,

acc\_num,

"balance\_sheet",

headers=headers,

statement\_keys\_map=statement\_keys\_map,

)

accn

extract\_columns\_values\_and\_dates\_from\_statement(soup)

process\_one\_statement(ticker, acc\_num, "cash\_flow\_statement")

import os

import pandas as pd

import numpy as np # make sure to add

import requests

from bs4 import BeautifulSoup

import logging # make sure to add

import calendar # make sure to add

from headers import headers # change to your own headers file or add variable in code

pd.options.display.float\_format = (

lambda x: "{:,.0f}".format(x) if int(x) == x else "{:,.2f}".format(x)

)

statement\_keys\_map = {

"balance\_sheet": [

"balance sheet",

"balance sheets",

"statement of financial position",

"consolidated balance sheets",

"consolidated balance sheet",

"consolidated financial position",

"consolidated balance sheets - southern",

"consolidated statements of financial position",

"consolidated statement of financial position",

"consolidated statements of financial condition",

"combined and consolidated balance sheet",

"condensed consolidated balance sheets",

"consolidated balance sheets, as of december 31",

"dow consolidated balance sheets",

"consolidated balance sheets (unaudited)",

],

"income\_statement": [

"income statement",

"income statements",

"statement of earnings (loss)",

"statements of consolidated income",

"consolidated statements of operations",

"consolidated statement of operations",

"consolidated statements of earnings",

"consolidated statement of earnings",

"consolidated statements of income",

"consolidated statement of income",

"consolidated income statements",

"consolidated income statement",

"condensed consolidated statements of earnings",

"consolidated results of operations",

"consolidated statements of income (loss)",

"consolidated statements of income - southern",

"consolidated statements of operations and comprehensive income",

"consolidated statements of comprehensive income",

],

"cash\_flow\_statement": [

"cash flows statement",

"cash flows statements",

"statement of cash flows",

"statements of consolidated cash flows",

"consolidated statements of cash flows",

"consolidated statement of cash flows",

"consolidated statement of cash flow",

"consolidated cash flows statements",

"consolidated cash flow statements",

"condensed consolidated statements of cash flows",

"consolidated statements of cash flows (unaudited)",

"consolidated statements of cash flows - southern",

],

}

def cik\_matching\_ticker(ticker, headers=headers):

ticker = ticker.upper().replace(".", "-")

ticker\_json = requests.get(

"https://www.sec.gov/files/company\_tickers.json", headers=headers

).json()

for company in ticker\_json.values():

if company["ticker"] == ticker:

cik = str(company["cik\_str"]).zfill(10)

return cik

raise ValueError(f"Ticker {ticker} not found in SEC database")

def get\_submission\_data\_for\_ticker(ticker, headers=headers, only\_filings\_df=False):

"""

Get the data in json form for a given ticker. For example: 'cik', 'entityType', 'sic', 'sicDescription', 'insiderTransactionForOwnerExists', 'insiderTransactionForIssuerExists', 'name', 'tickers', 'exchanges', 'ein', 'description', 'website', 'investorWebsite', 'category', 'fiscalYearEnd', 'stateOfIncorporation', 'stateOfIncorporationDescription', 'addresses', 'phone', 'flags', 'formerNames', 'filings'

Args:

ticker (str): The ticker symbol of the company.

Returns:

json: The submissions for the company.

"""

cik = cik\_matching\_ticker(ticker)

headers = headers

url = f"https://data.sec.gov/submissions/CIK{cik}.json"

company\_json = requests.get(url, headers=headers).json()

if only\_filings\_df:

return pd.DataFrame(company\_json["filings"]["recent"])

else:

return company\_json

def get\_filtered\_filings(

ticker, ten\_k=True, just\_accession\_numbers=False, headers=None

):

"""

Retrieves either 10-K or 10-Q filings for a given ticker and optionally returns just accession numbers.

Args:

ticker (str): Stock ticker symbol.

ten\_k (bool): If True, fetches 10-K filings; otherwise, fetches 10-Q filings.

just\_accession\_numbers (bool): If True, returns only accession numbers; otherwise, returns full data.

headers (dict): Headers for HTTP request.

Returns:

DataFrame or Series: DataFrame of filings or Series of accession numbers.

"""

# Fetch submission data for the given ticker

company\_filings\_df = get\_submission\_data\_for\_ticker(

ticker, only\_filings\_df=True, headers=headers

)

# Filter for 10-K or 10-Q forms

df = company\_filings\_df[company\_filings\_df["form"] == ("10-K" if ten\_k else "10-Q")]

# Return accession numbers if specified

if just\_accession\_numbers:

df = df.set\_index("reportDate")

accession\_df = df["accessionNumber"]

return accession\_df

else:

return df

def get\_facts(ticker, headers=None):

"""

Retrieves company facts for a given ticker.

Args:

ticker (str): Stock ticker symbol.

headers (dict): Headers for HTTP request.

Returns:

dict: Company facts in JSON format.

"""

# Get CIK number matching the ticker

cik = cik\_matching\_ticker(ticker)

# Construct URL for company facts

url = f"https://data.sec.gov/api/xbrl/companyfacts/CIK{cik}.json"

# Fetch and return company facts

company\_facts = requests.get(url, headers=headers).json()

return company\_facts

def facts\_DF(ticker, headers=None):

"""

Converts company facts into a DataFrame.

Args:

ticker (str): Stock ticker symbol.

headers (dict): Headers for HTTP request.

Returns:

tuple: DataFrame of facts and a dictionary of labels.

"""

# Retrieve facts data

facts = get\_facts(ticker, headers)

us\_gaap\_data = facts["facts"]["us-gaap"]

df\_data = []

# Process each fact and its details

for fact, details in us\_gaap\_data.items():

for unit in details["units"]:

for item in details["units"][unit]:

row = item.copy()

row["fact"] = fact

df\_data.append(row)

df = pd.DataFrame(df\_data)

# Convert 'end' and 'start' to datetime

df["end"] = pd.to\_datetime(df["end"])

df["start"] = pd.to\_datetime(df["start"])

# Drop duplicates and set index

df = df.drop\_duplicates(subset=["fact", "end", "val"])

df.set\_index("end", inplace=True)

# Create a dictionary of labels for facts

labels\_dict = {fact: details["label"] for fact, details in us\_gaap\_data.items()}

return df, labels\_dict

def annual\_facts(ticker, headers=None):

"""

Fetches and processes annual (10-K) financial facts for a given ticker.

Args:

ticker (str): Stock ticker symbol.

headers (dict): Headers for HTTP request.

Returns:

DataFrame: Transposed pivot table of annual financial facts.

"""

# Get accession numbers for 10-K filings

accession\_nums = get\_filtered\_filings(

ticker, ten\_k=True, just\_accession\_numbers=True, headers=headers

)

# Extract and process facts data

df, label\_dict = facts\_DF(ticker, headers)

# Filter data for 10-K filings

ten\_k = df[df["accn"].isin(accession\_nums)]

ten\_k = ten\_k[ten\_k.index.isin(accession\_nums.index)]

# Pivot and format the data

pivot = ten\_k.pivot\_table(values="val", columns="fact", index="end")

pivot.rename(columns=label\_dict, inplace=True)

return pivot.T

def quarterly\_facts(ticker, headers=None):

"""

Fetches and processes quarterly (10-Q) financial facts for a given ticker.

Args:

ticker (str): Stock ticker symbol.

headers (dict): Headers for HTTP request.

Returns:

DataFrame: Transposed pivot table of quarterly financial facts.

"""

# Get accession numbers for 10-Q filings

accession\_nums = get\_filtered\_filings(

ticker, ten\_k=False, just\_accession\_numbers=True, headers=headers

)

# Extract and process facts data

df, label\_dict = facts\_DF(ticker, headers)

# Filter data for 10-Q filings

ten\_q = df[df["accn"].isin(accession\_nums)]

ten\_q = ten\_q[ten\_q.index.isin(accession\_nums.index)].reset\_index(drop=False)

# Remove duplicate entries

ten\_q = ten\_q.drop\_duplicates(subset=["fact", "end"], keep="last")

# Pivot and format the data

pivot = ten\_q.pivot\_table(values="val", columns="fact", index="end")

pivot.rename(columns=label\_dict, inplace=True)

return pivot.T

def save\_dataframe\_to\_csv(dataframe, folder\_name, ticker, statement\_name, frequency):

"""

Saves a given DataFrame to a CSV file in a specified directory.

Args:

dataframe (DataFrame): The DataFrame to be saved.

folder\_name (str): The folder name where the CSV file will be saved.

ticker (str): Stock ticker symbol.

statement\_name (str): Name of the financial statement.

frequency (str): Frequency of the financial data (e.g., annual, quarterly).

Returns:

None

"""

# Create directory path

directory\_path = os.path.join(folder\_name, ticker)

os.makedirs(directory\_path, exist\_ok=True)

# Construct file path and save DataFrame

file\_path = os.path.join(directory\_path, f"{statement\_name}\_{frequency}.csv")

dataframe.to\_csv(file\_path)

def \_get\_file\_name(report):

"""

Extracts the file name from an XML report tag.

Args:

report (Tag): BeautifulSoup tag representing the report.

Returns:

str: File name extracted from the tag.

"""

html\_file\_name\_tag = report.find("HtmlFileName")

xml\_file\_name\_tag = report.find("XmlFileName")

# Return the appropriate file name

if html\_file\_name\_tag:

return html\_file\_name\_tag.text

elif xml\_file\_name\_tag:

return xml\_file\_name\_tag.text

else:

return ""

def \_is\_statement\_file(short\_name\_tag, long\_name\_tag, file\_name):

"""

Determines if a given file is a financial statement file.

Args:

short\_name\_tag (Tag): BeautifulSoup tag for the short name.

long\_name\_tag (Tag): BeautifulSoup tag for the long name.

file\_name (str): Name of the file.

Returns:

bool: True if it's a statement file, False otherwise.

"""

return (

short\_name\_tag is not None

and long\_name\_tag is not None

and file\_name # Ensure file\_name is not an empty string

and "Statement" in long\_name\_tag.text

)

def get\_statement\_file\_names\_in\_filing\_summary(ticker, accession\_number, headers=None):

"""

Retrieves file names of financial statements from a filing summary.

Args:

ticker (str): Stock ticker symbol.

accession\_number (str): SEC filing accession number.

headers (dict): Headers for HTTP request.

Returns:

dict: Dictionary mapping statement types to their file names.

"""

try:

# Set up request session and get filing summary

session = requests.Session()

cik = cik\_matching\_ticker(ticker)

base\_link = f"https://www.sec.gov/Archives/edgar/data/{cik}/{accession\_number}"

filing\_summary\_link = f"{base\_link}/FilingSummary.xml"

filing\_summary\_response = session.get(

filing\_summary\_link, headers=headers

).content.decode("utf-8")

# Parse the filing summary

filing\_summary\_soup = BeautifulSoup(filing\_summary\_response, "lxml-xml")

statement\_file\_names\_dict = {}

# Extract file names for statements

for report in filing\_summary\_soup.find\_all("Report"):

file\_name = \_get\_file\_name(report)

short\_name, long\_name = report.find("ShortName"), report.find("LongName")

if \_is\_statement\_file(short\_name, long\_name, file\_name):

statement\_file\_names\_dict[short\_name.text.lower()] = file\_name

return statement\_file\_names\_dict

except requests.RequestException as e:

print(f"An error occurred: {e}")

return {}

def get\_statement\_soup(

ticker, accession\_number, statement\_name, headers, statement\_keys\_map

):

"""

Retrieves the BeautifulSoup object for a specific financial statement.

Args:

ticker (str): Stock ticker symbol.

accession\_number (str): SEC filing accession number.

statement\_name (str): has to be 'balance\_sheet', 'income\_statement', 'cash\_flow\_statement'

headers (dict): Headers for HTTP request.

statement\_keys\_map (dict): Mapping of statement names to keys.

Returns:

BeautifulSoup: Parsed HTML/XML content of the financial statement.

Raises:

ValueError: If the statement file name is not found or if there is an error fetching the statement.

"""

session = requests.Session()

cik = cik\_matching\_ticker(ticker)

base\_link = f"https://www.sec.gov/Archives/edgar/data/{cik}/{accession\_number}"

# Get statement file names

statement\_file\_name\_dict = get\_statement\_file\_names\_in\_filing\_summary(

ticker, accession\_number, headers

)

statement\_link = None

# Find the specific statement link

for possible\_key in statement\_keys\_map.get(statement\_name.lower(), []):

file\_name = statement\_file\_name\_dict.get(possible\_key.lower())

if file\_name:

statement\_link = f"{base\_link}/{file\_name}"

break

if not statement\_link:

raise ValueError(f"Could not find statement file name for {statement\_name}")

# Fetch the statement

try:

statement\_response = session.get(statement\_link, headers=headers)

statement\_response.raise\_for\_status() # Check for a successful request

# Parse and return the content

if statement\_link.endswith(".xml"):

return BeautifulSoup(

statement\_response.content, "lxml-xml", from\_encoding="utf-8"

)

else:

return BeautifulSoup(statement\_response.content, "lxml")

except requests.RequestException as e:

raise ValueError(f"Error fetching the statement: {e}")

def extract\_columns\_values\_and\_dates\_from\_statement(soup):

"""

Extracts columns, values, and dates from an HTML soup object representing a financial statement.

Args:

soup (BeautifulSoup): The BeautifulSoup object of the HTML document.

Returns:

tuple: Tuple containing columns, values\_set, and date\_time\_index.

"""

columns = []

values\_set = []

date\_time\_index = get\_datetime\_index\_dates\_from\_statement(soup)

for table in soup.find\_all("table"):

unit\_multiplier = 1

special\_case = False

# Check table headers for unit multipliers and special cases

table\_header = table.find("th")

if table\_header:

header\_text = table\_header.get\_text()

# Determine unit multiplier based on header text

if "in Thousands" in header\_text:

unit\_multiplier = 1

elif "in Millions" in header\_text:

unit\_multiplier = 1000

# Check for special case scenario

if "unless otherwise specified" in header\_text:

special\_case = True

# Process each row of the table

for row in table.select("tr"):

onclick\_elements = row.select("td.pl a, td.pl.custom a")

if not onclick\_elements:

continue

# Extract column title from 'onclick' attribute

onclick\_attr = onclick\_elements[0]["onclick"]

column\_title = onclick\_attr.split("defref\_")[-1].split("',")[0]

columns.append(column\_title)

# Initialize values array with NaNs

values = [np.NaN] \* len(date\_time\_index)

# Process each cell in the row

for i, cell in enumerate(row.select("td.text, td.nump, td.num")):

if "text" in cell.get("class"):

continue

# Clean and parse cell value

value = keep\_numbers\_and\_decimals\_only\_in\_string(

cell.text.replace("$", "")

.replace(",", "")

.replace("(", "")

.replace(")", "")

.strip()

)

if value:

value = float(value)

# Adjust value based on special case and cell class

if special\_case:

value /= 1000

else:

if "nump" in cell.get("class"):

values[i] = value \* unit\_multiplier

else:

values[i] = -value \* unit\_multiplier

values\_set.append(values)

return columns, values\_set, date\_time\_index

def get\_datetime\_index\_dates\_from\_statement(soup: BeautifulSoup) -> pd.DatetimeIndex:

"""

Extracts datetime index dates from the HTML soup object of a financial statement.

Args:

soup (BeautifulSoup): The BeautifulSoup object of the HTML document.

Returns:

pd.DatetimeIndex: A Pandas DatetimeIndex object containing the extracted dates.

"""

table\_headers = soup.find\_all("th", {"class": "th"})

dates = [str(th.div.string) for th in table\_headers if th.div and th.div.string]

dates = [standardize\_date(date).replace(".", "") for date in dates]

index\_dates = pd.to\_datetime(dates)

return index\_dates

def standardize\_date(date: str) -> str:

"""

Standardizes date strings by replacing abbreviations with full month names.

Args:

date (str): The date string to be standardized.

Returns:

str: The standardized date string.

"""

for abbr, full in zip(calendar.month\_abbr[1:], calendar.month\_name[1:]):

date = date.replace(abbr, full)

return date

def keep\_numbers\_and\_decimals\_only\_in\_string(mixed\_string: str):

"""

Filters a string to keep only numbers and decimal points.

Args:

mixed\_string (str): The string containing mixed characters.

Returns:

str: String containing only numbers and decimal points.

"""

num = "1234567890."

allowed = list(filter(lambda x: x in num, mixed\_string))

return "".join(allowed)

def create\_dataframe\_of\_statement\_values\_columns\_dates(

values\_set, columns, index\_dates

) -> pd.DataFrame:

"""

Creates a DataFrame from statement values, columns, and index dates.

Args:

values\_set (list): List of values for each column.

columns (list): List of column names.

index\_dates (pd.DatetimeIndex): DatetimeIndex for the DataFrame index.

Returns:

pd.DataFrame: DataFrame constructed from the given data.

"""

transposed\_values\_set = list(zip(\*values\_set))

df = pd.DataFrame(transposed\_values\_set, columns=columns, index=index\_dates)

return df

def process\_one\_statement(ticker, accession\_number, statement\_name):

"""

Processes a single financial statement identified by ticker, accession number, and statement name.

Args:

ticker (str): The stock ticker.

accession\_number (str): The SEC accession number.

statement\_name (str): Name of the financial statement.

Returns:

pd.DataFrame or None: DataFrame of the processed statement or None if an error occurs.

"""

try:

# Fetch the statement HTML soup

soup = get\_statement\_soup(

ticker,

accession\_number,

statement\_name,

headers=headers,

statement\_keys\_map=statement\_keys\_map,

)

except Exception as e:

logging.error(

f"Failed to get statement soup: {e} for accession number: {accession\_number}"

)

return None

if soup:

try:

# Extract data and create DataFrame

columns, values, dates = extract\_columns\_values\_and\_dates\_from\_statement(

soup

)

df = create\_dataframe\_of\_statement\_values\_columns\_dates(

values, columns, dates

)

if not df.empty:

# Remove duplicate columns

df = df.T.drop\_duplicates()

else:

logging.warning(

f"Empty DataFrame for accession number: {accession\_number}"

)

return None

return df

except Exception as e:

logging.error(f"Error processing statement: {e}")

return None

### equity research

import requests

import json

from sec\_cik\_mapper import StockMapper

global symbol

def add\_zeros\_to\_cik():

global symbol

while True:

symbol = input("Enter a ticker symbol: ")

mapper = StockMapper()

num = mapper.ticker\_to\_cik[f'{symbol.upper()}']

if num.isdigit() and 0 < len(num) <= 10:

return str(num).zfill(10)

else:

print("Invalid input. Please enter a numeric CIK number with 1-10 digits.")

cik = add\_zeros\_to\_cik()

sec\_url = f"https://data.sec.gov/api/xbrl/companyfacts/CIK{cik}.json"

print(sec\_url)

headers = {

'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.3',

'Accept': 'application/json',

'Accept-Language': 'en-US,en;q=0.5',

'Accept-Encoding': 'gzip, deflate',

'Connection': 'keep-alive'

}

response = requests.get(sec\_url, headers=headers)

if response.status\_code == 200:

data = response.json()

with open(f'./SEC\_company\_facts/{symbol.upper()}.json', 'w') as f:

json.dump(data, f, indent=4)

print(f"Response saved as {symbol.upper()}.json")

else:

print("Failed to retrieve data")

https://data.sec.gov/api/xbrl/companyfacts/CIK0000921114.json

Response saved as response.json

### equity reserach

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from scipy.stats import norm

import requests

def get\_world\_bank\_data(indicator, country, start\_year, end\_year):

"""

Get data from the World Bank API.

Parameters:

indicator (str): The indicator code (e.g. 'SP.POP.TOTL' for total population)

country (str): The country code (e.g. 'USA' for United States)

start\_year (int): The start year for the data

end\_year (int): The end year for the data

Returns:

pandas.DataFrame: The data from the World Bank API

"""

url = f'http://api.worldbank.org/v2/country/{country}/indicator/{indicator}?format=json&per\_page=1000&date={start\_year}:{end\_year}'

response = requests.get(url)

data = response.json()[1]

df = pd.DataFrame(data)

return df

def calculate\_healthcare\_costs(resistance\_rate, population, avg\_treatment\_cost):

"""

Calculate the cumulative annual healthcare costs for treating drug-resistant bacterial infections.

Parameters:

resistance\_rate (float): The rate of antimicrobial resistance (e.g. 0.2 for 20%)

population (int): The population of the country/region

avg\_treatment\_cost (float): The average cost of treating a drug-resistant bacterial infection

Returns:

float: The cumulative annual healthcare costs

"""

return resistance\_rate \* population \* avg\_treatment\_cost

def estimate\_market\_size(year, growth\_rate, initial\_market\_size):

"""

Estimate the market size for novel antibacterial therapies.

Parameters:

year (int): The year for which to estimate the market size

growth\_rate (float): The annual growth rate of the market (e.g. 0.1 for 10%)

initial\_market\_size (float): The initial market size

Returns:

float: The estimated market size for the given year

"""

return initial\_market\_size \* (1 + growth\_rate) \*\* (year - 2023)

def job\_creation\_estimate(market\_size, job\_creation\_rate):

"""

Estimate the number of jobs created in the biotechnology and pharmaceutical industries.

Parameters:

market\_size (float): The estimated market size for novel antibacterial therapies

job\_creation\_rate (float): The rate of job creation per dollar of market size (e.g. 0.01 for 1%)

Returns:

int: The estimated number of jobs created

"""

return int(market\_size \* job\_creation\_rate)

def plot\_healthcare\_costs(years, healthcare\_costs):

"""

Plot the cumulative annual healthcare costs over time.

Parameters:

years (list): The years for which to plot the data

healthcare\_costs (list): The cumulative annual healthcare costs for each year

"""

plt.plot(years, healthcare\_costs)

plt.xlabel('Year')

plt.ylabel('Cumulative Annual Healthcare Costs (billions USD)')

plt.title('Cumulative Annual Healthcare Costs for Treating Drug-Resistant Bacterial Infections')

plt.show()

def main():

# Get population data from the World Bank API

population\_df = get\_world\_bank\_data('SP.POP.TOTL', 'USA', 2010, 2030)

population = population\_df['value'].iloc[-1]

# Example usage:

resistance\_rate = 0.2 # 20% antimicrobial resistance rate

avg\_treatment\_cost = 10000 # average cost of treating a drug-resistant bacterial infection

healthcare\_costs = []

years = list(range(2023, 2031))

for year in years:

healthcare\_cost = calculate\_healthcare\_costs(resistance\_rate, population, avg\_treatment\_cost)

healthcare\_costs.append(healthcare\_cost)

print(f"Cumulative annual healthcare costs: ${healthcare\_costs[-1]:.2f} billion")

# Plot the cumulative annual healthcare costs

plot\_healthcare\_costs(years, healthcare\_costs)

initial\_market\_size = 1e9 # initial market size for novel antibacterial therapies (2023)

growth\_rate = 0.1 # 10% annual growth rate

market\_size = estimate\_market\_size(2030, growth\_rate, initial\_market\_size)

print(f"Estimated market size for novel antibacterial therapies in 2030: ${market\_size:.2f} billion")

job\_creation\_rate = 0.01 # 1% job creation rate per dollar of market size

jobs\_created = job\_creation\_estimate(market\_size, job\_creation\_rate)

print(f"Estimated number of jobs created in biotechnology and pharmaceutical industries: {jobs\_created:,}")

if \_\_name\_\_ == "\_\_main\_\_":

main()

### finpy

from finpy.edgar.company import company

from finpy.utils.components import sp500

from finpy.utils.components import russel3000

from finpy.utils.components import custom

import argparse

import pandas as pd

import os

import sqlite3

if \_\_name\_\_ == "\_\_main\_\_":

parser = argparse.ArgumentParser(description='Name and Email.')

parser.add\_argument('-dir', default="app", help="app directory")

parser.add\_argument('-tick', help='ticker file list')

parser.add\_argument('-sp500', action="store\_true", default=False, help="include all tickers in s&p 500")

parser.add\_argument('-russel3000', action="store\_true", default=False, help="include all tickers in russel 3000")

parser.add\_argument('-header', default='{% extends "base.html" %}\n{% block content %}\n', help="header: html code before the table")

parser.add\_argument('-footer', default="{% endblock %}", help="footer: html code before the table")

args = parser.parse\_args()

tickers = custom(args.tick)

error\_tickers = []

r = []

e = []

if args.sp500:

tickers += sp500();

if args.russel3000:

missing\_file = os.path.join(args.dir, "missing.txt");

r3k\_missing = custom(missing\_file)

r3k = russel3000(format="pandas")

r3k = r3k.drop(columns=['Notional Value', 'Asset Class', 'Location','Exchange','Currency','FX Rate','Market Currency','Accrual Date', 'Market Value', 'Shares'])

r3k = r3k[~r3k['Ticker'].isin(r3k\_missing)]

tickers += list(r3k['Ticker'])

r3k.insert(2, 'CJK', "")

tickers = list(set(tickers))

company\_ticker\_json\_db = os.path.join(os.path.join(os.environ['FINPYDATA'], "edgar", "files", "company\_tickers.db"))

conn = sqlite3.connect(company\_ticker\_json\_db)

# print(tickers, len(tickers))

for i in tickers:

try:

r.append(company(i, conn))

except:

error\_tickers.append(i)

print("Error {}".format(i))

conn.close()

data = []

for i in r:

latest\_str = "<a href={}>{}</a>".format(i.latest\_inline\_xbrl, i.latest\_form)

cik\_str = "<a href=https://data.sec.gov/api/xbrl/companyfacts/CIK{}.json>{}</a>".format(i.cik, i.cik)

tick\_str = "<a href={}\_edgar.html>{}</a>".format(i.ticker, i.ticker)

name\_str = "<a href=https://www.sec.gov/edgar/browse/?CIK={}&owner=exclude>{}</a>".format(i.cik, i.name)

r3k.loc[r3k.Ticker == i.ticker, 'CJK'] = cik\_str

tick\_file = os.path.join(args.dir, "{}.html".format(i.ticker))

r3k.loc[r3k.Ticker == i.ticker, 'Name'] = name\_str

if os.path.isfile(tick\_file):

r3k.loc[r3k.Ticker == i.ticker, 'Ticker'] = tick\_str

df2 = {'Ranking' : i.ranking, 'Ticker' : tick\_str, 'name' : name\_str, 'cik' : cik\_str, 'sic' : i.sic, 'sicDesciption' : i.sicDescription, 'latest': latest\_str}

data.append(df2)

df = pd.DataFrame(data)

sics = set(df['sic'])

for sic in sics:

sic\_file\_name = os.path.join(args.dir, "sic\_{}.html".format(sic))

with open(sic\_file\_name, 'w') as sic\_f:

sic\_f.write(args.header)

sic\_f.write(df.loc[df['sic'] == sic].sort\_values(by=['Ranking']).to\_html(escape=False,table\_id="EdgarMain",index=False))

sic\_f.write(args.footer)

sic\_str = "<a href=sic\_{}.html>{}</a>".format(sic, sic)

df.loc[df['sic'] == sic, 'sic'] = sic\_str

file\_name = os.path.join(args.dir, 'edgar.html')

with open(file\_name , 'w') as f:

f.write(args.header)

f.write(df.sort\_values(by=['Ranking']).to\_html(escape=False,table\_id="EdgarMain",index=False))

f.write(args.footer)

if args.russel3000:

russel3000\_file = os.path.join(args.dir, "russel3000.html");

with open(russel3000\_file , 'w') as f:

f.write(args.header)

f.write(r3k.to\_html(escape=False,table\_id="Russel 3000",index=False))

f.write(args.footer)

from finpy.edgar.download import download

from finpy.edgar.download import async\_download\_url

from finpy.utils.components import sp500

from finpy.utils.components import russel3000

from finpy.utils.components import custom

from aiolimiter import AsyncLimiter

import asyncio

import time

from datetime import date

import argparse

import os

import json

import sqlite3

from contextlib import closing

import logging

logger = logging.getLogger(\_\_name\_\_)

async def async\_get\_company\_ticker\_json(hdr, tickers, limiter, semaphore):

url\_str = 'https://www.sec.gov/files/company\_tickers.json'

if not os.path.isdir(os.path.join(os.environ['FINPYDATA'], "edgar", "files")):

os.makedirs(os.path.join(os.environ['FINPYDATA'], "edgar", "files"));

company\_ticker\_json\_file = os.path.join(os.path.join(os.environ['FINPYDATA'], "edgar", "files", "company\_tickers.json"))

company\_ticker\_json\_db = os.path.join(os.path.join(os.environ['FINPYDATA'], "edgar", "files", "company\_tickers.db"))

content = await async\_download\_url(url\_str, hdr, limiter, semaphore)

print(company\_ticker\_json\_file)

with open(company\_ticker\_json\_file, 'w') as file:

file.write(content)

company\_tickers\_json = json.loads(content)

try:

with closing(sqlite3.connect(company\_ticker\_json\_db)) as conn:

with closing(conn.cursor()) as cursor:

cursor.execute('''CREATE TABLE COMPANY (ranking INT NOT NULL,

cik TEXT NOT NULL,

ticker TEXT PRIMARY KEY NOT NULL,

name TEXT NOT NULL,

sic INT NULL,

sicDescription TEXT NULL,

latest\_filing\_date TEXT NULL,

latest\_report\_date TEXT NULL,

latest\_primaryDocument TEXT NULL,

latest\_accessionNumber TEXT NULL,

latest\_form TEXT NULL

);''')

for i in company\_tickers\_json:

if company\_tickers\_json[i]["ticker"] in set(tickers):

cursor.execute("INSERT INTO COMPANY (ranking, cik, ticker, name) VALUES (?, ?, ?, ?)", \

(i, str(company\_tickers\_json[i]["cik\_str"]).zfill(10), company\_tickers\_json[i]["ticker"], company\_tickers\_json[i]["title"]))

conn.commit()

except:

with closing(sqlite3.connect(company\_ticker\_json\_db)) as conn:

with closing(conn.cursor()) as cursor:

for i in company\_tickers\_json:

cursor.execute("""

INSERT OR IGNORE INTO COMPANY (ranking, cik, ticker, name) VALUES (?, ?, ?, ?)

""", \

(i, str(company\_tickers\_json[i]["cik\_str"]).zfill(10), company\_tickers\_json[i]["ticker"], company\_tickers\_json[i]["title"]))

conn.commit()

async def main(name, email, nodownload, missing\_tick, tickers):

slot = 0.25

limiter = AsyncLimiter(1, slot)

tasks = []

semaphore = asyncio.Semaphore(value=10)

hdr = {'User-Agent' : name + email}

await async\_get\_company\_ticker\_json(hdr, tickers, limiter, semaphore)

company\_ticker\_json\_db = os.path.join(os.path.join(os.environ['FINPYDATA'], "edgar", "files", "company\_tickers.db"))

r = {}

num = 0

with open(missing\_tick, 'w') as file:

for ticker in tickers:

with closing(sqlite3.connect(company\_ticker\_json\_db)) as conn:

with closing(conn.cursor()) as cursor:

row = cursor.execute("SELECT \* FROM COMPANY WHERE ticker = '{}'".format(ticker)).fetchone()

ticker\_info = {}

try:

ticker\_info['ranking'] = row[0]

ticker\_info['cik'] = row[1]

ticker\_info['ticker'] = row[2]

ticker\_info['name'] = row[3]

ticker\_info['sic'] = row[4]

ticker\_info['sicDescription'] = row[5]

ticker\_info['latest\_filing\_date'] = date.fromisoformat(row[6]) if isinstance(row[6], str) else row[6]

ticker\_info['latest\_report\_date'] = date.fromisoformat(row[7]) if isinstance(row[7], str) else row[7]

ticker\_info['latest\_primaryDocument'] = row[8]

ticker\_info['latest\_accessionNumber'] = row[9]

ticker\_info['latest\_form'] = row[10]

tasks.append(download.async\_create(ticker\_info, name, email, nodownload, True, limiter, semaphore, r))

except:

file.write("{}\n".format(ticker))

print("Error", ticker)

await asyncio.wait(tasks)

# for i in r:

# print(i)

return r

if \_\_name\_\_ == "\_\_main\_\_":

parser = argparse.ArgumentParser(description='Name and Email.')

parser.add\_argument('-name', help='Name')

parser.add\_argument('-email', help='E-Mail address')

parser.add\_argument('-tick', help='ticker file list')

parser.add\_argument('-sp500', action="store\_true", default=False, help="include all tickers in s&p 500")

parser.add\_argument('-russel3000', action="store\_true", default=False, help="include all tickers in russel 3000")

parser.add\_argument('-nodownload', action="store\_true", default=False, help="only update the database from the exisiting json files")

parser.add\_argument('-dir', default="app", help="app directory")

args = parser.parse\_args()

tickers = custom(args.tick)

if args.sp500:

sp500 = sp500()

tickers += sp500

if args.russel3000:

r3k = russel3000(format="list")

tickers += r3k

tickers = list(set(tickers))

missing\_tick = os.path.join(args.dir, "missing.txt")

s = time.perf\_counter()

asyncio.run(main(args.name, args.email, args.nodownload, missing\_tick, tickers)) # Activate this line if the code is to be executed in VS Code

# , etc. Otherwise deactivate it.

# r = await main() # Activate this line if the code is to be executed in Jupyter

# Notebook! Otherwise deactivate it.

elapsed = time.perf\_counter() - s

print(f"Execution time: {elapsed:0.2f} seconds.")

from finpy.edgar.company import company

from finpy.utils.components import sp500

from finpy.utils.components import russel3000

from finpy.utils.components import custom

import argparse

import pandas as pd

import os

import sqlite3

if \_\_name\_\_ == "\_\_main\_\_":

parser = argparse.ArgumentParser(description='Name and Email.')

parser.add\_argument('-dir', default="app", help="app directory")

parser.add\_argument('-tick', help='ticker file list')

parser.add\_argument('-sp500', action="store\_true", default=False, help="include all tickers in s&p 500")

parser.add\_argument('-russel3000', action="store\_true", default=False, help="include all tickers in russel 3000")

parser.add\_argument('-header', default='{% extends "base.html" %}\n{% block content %}\n', help="header: html code before the table")

parser.add\_argument('-footer', default="{% endblock %}", help="footer: html code before the table")

args = parser.parse\_args()

tickers = custom(args.tick)

error\_tickers = []

r = []

e = []

if args.sp500:

tickers += sp500();

if args.russel3000:

missing\_file = os.path.join(args.dir, "missing.txt");

r3k\_missing = custom(missing\_file)

r3k = russel3000(format="pandas")

r3k = r3k.drop(columns=['Notional Value', 'Asset Class', 'Location','Exchange','Currency','FX Rate','Market Currency','Accrual Date', 'Market Value', 'Shares'])

r3k = r3k[~r3k['Ticker'].isin(r3k\_missing)]

tickers += list(r3k['Ticker'])

r3k.insert(2, 'CJK', "")

tickers = list(set(tickers))

company\_ticker\_json\_db = os.path.join(os.path.join(os.environ['FINPYDATA'], "edgar", "files", "company\_tickers.db"))

conn = sqlite3.connect(company\_ticker\_json\_db)

# print(tickers, len(tickers))

for i in tickers:

try:

r.append(company(i, conn))

except:

error\_tickers.append(i)

print("Error {}".format(i))

conn.close()

data = []

for i in r:

latest\_str = "<a href={}>{}</a>".format(i.latest\_inline\_xbrl, i.latest\_form)

cik\_str = "<a href=https://data.sec.gov/api/xbrl/companyfacts/CIK{}.json>{}</a>".format(i.cik, i.cik)

tick\_str = "<a href={}\_edgar.html>{}</a>".format(i.ticker, i.ticker)

name\_str = "<a href=https://www.sec.gov/edgar/browse/?CIK={}&owner=exclude>{}</a>".format(i.cik, i.name)

r3k.loc[r3k.Ticker == i.ticker, 'CJK'] = cik\_str

tick\_file = os.path.join(args.dir, "{}.html".format(i.ticker))

r3k.loc[r3k.Ticker == i.ticker, 'Name'] = name\_str

if os.path.isfile(tick\_file):

r3k.loc[r3k.Ticker == i.ticker, 'Ticker'] = tick\_str

df2 = {'Ranking' : i.ranking, 'Ticker' : tick\_str, 'name' : name\_str, 'cik' : cik\_str, 'sic' : i.sic, 'sicDesciption' : i.sicDescription, 'latest': latest\_str}

data.append(df2)

df = pd.DataFrame(data)

sics = set(df['sic'])

for sic in sics:

sic\_file\_name = os.path.join(args.dir, "sic\_{}.html".format(sic))

with open(sic\_file\_name, 'w') as sic\_f:

sic\_f.write(args.header)

sic\_f.write(df.loc[df['sic'] == sic].sort\_values(by=['Ranking']).to\_html(escape=False,table\_id="EdgarMain",index=False))

sic\_f.write(args.footer)

sic\_str = "<a href=sic\_{}.html>{}</a>".format(sic, sic)

df.loc[df['sic'] == sic, 'sic'] = sic\_str

file\_name = os.path.join(args.dir, 'edgar.html')

with open(file\_name , 'w') as f:

f.write(args.header)

f.write(df.sort\_values(by=['Ranking']).to\_html(escape=False,table\_id="EdgarMain",index=False))

f.write(args.footer)

if args.russel3000:

russel3000\_file = os.path.join(args.dir, "russel3000.html");

with open(russel3000\_file , 'w') as f:

f.write(args.header)

f.write(r3k.to\_html(escape=False,table\_id="Russel 3000",index=False))

f.write(args.footer)

import sys

import csv

import matplotlib

matplotlib.use('Agg') # fix for matplotlib under multiprocessing

import matplotlib.pyplot as plt

import matplotlib.dates as mdates

import datetime as dt

import sets

import dateutil

from finpy.utils import get\_tickdata

from finpy.equity import Equity

from finpy.portfolio import Portfolio

from finpy.order import Order

import finpy.fpdateutil as du

if \_\_name\_\_ == '\_\_main\_\_':

"""

python marketsim.py 1000000 orders.csv values.csv

Where the number represents starting cash and orders.csv is a file of orders organized like this:

2008-12-3, AAPL, BUY, 130

2008-12-8, AAPL, SELL, 130

2008-12-5, IBM, BUY, 50

values.csv

2008-12-3, 1000000

2008-12-4, 1000010

2008-12-5, 1000250

"""

cash = sys.argv[1]

order\_file = sys.argv[2]

value\_file = sys.argv[3]

order\_list = []

dt\_timeofday = dt.timedelta(hours=16)

with open(order\_file, 'rU') as csvfile:

order\_reader = csv.reader(csvfile, delimiter=',', skipinitialspace=True)

for row in order\_reader:

date = dateutil.parser.parse(row[0] + "-16")

if len(row) == 4:

o = Order(action=row[2], date=date, tick=row[1], shares=row[3])

else:

o = Order(action=row[2], date=date, tick=row[1], shares=row[3], price=row[4])

order\_list.append(o)

# order\_list needs to be sorted. Otherwise the algorithm won't work.

date\_list = [x.date for x in order\_list]

date\_list.sort()

dt\_start = date\_list[0]

dt\_end = date\_list[-1]

tick\_set = sets.Set([x.tick for x in order\_list])

ls\_symbols = ['$GSPC']

while(tick\_set):

ls\_symbols.append(tick\_set.pop())

ldt\_timestamps = du.getNYSEdays(dt\_start, dt\_end, dt\_timeofday)

all\_stocks = get\_tickdata(ls\_symbols=ls\_symbols, ldt\_timestamps=ldt\_timestamps)

pf = Portfolio(equities=all\_stocks, cash=cash, dates=ldt\_timestamps, order\_list=order\_list)

pf.sim()

equity\_col = ['buy', 'sell', 'close']

pf.csvwriter(csv\_file=value\_file, d=',', cash=False)

print("The final value of the portfolio using the sample file is -- ", pf.total[-1])

print("Details of the Performance of the portfolio :")

print("Data Range :", ldt\_timestamps[0], "to", ldt\_timestamps[-1])

print("Sharpe Ratio of Fund :", pf.sharpe\_ratio())

print("Sortino Ratio of Fund :", pf.sortino())

print("Sharpe Ratio of $GSPC :", pf.equities['$GSPC'].sharpe\_ratio())

print("Total Return of Fund :", pf.return\_ratio())

print(" Total Return of $GSPC :", pf.equities['$GSPC'].return\_ratio())

print("Standard Deviation of Fund :", pf.std())

print(" Standard Deviation of $GSPC :", pf.equities['$GSPC'].std())

print("Average Daily Return of Fund :", pf.avg\_daily\_return())

print("Average Daily Return of $GSPC :", pf.equities['$GSPC'].avg\_daily\_return())

print("Information Ratio of Fund:", pf.info\_ratio(pf.equities['$GSPC']))

fig = plt.figure()

ax = fig.add\_subplot(111)

ax.plot(ldt\_timestamps, pf.equities['$GSPC'].normalized())

ax.plot(ldt\_timestamps, pf.total/pf.total[0])

legend = ['$GSPC', "Portfolio"]

ax.legend(legend, loc=2)

fig.autofmt\_xdate()

pdf\_file = order\_file + '.pdf'

fig.savefig(pdf\_file, format='pdf')

beta, alpha = pf.beta\_alpha(pf.equities['$GSPC'])

print("Beta of the fund is ", beta, ". Alpha of the fund is ", alpha)

import pandas as pd

import re

import os

import json

from dyplot.bar import Bar

from bs4 import BeautifulSoup

import xml.etree.ElementTree as ET

from contextlib import closing

import sqlite3

import logging

logger = logging.getLogger(\_\_name\_\_)

class company():

def \_\_init\_\_(self, ticker, conn, debug = True):

self.ticker = ticker

self.url = ""

self.debug = debug

with closing(conn.cursor()) as cursor:

row = cursor.execute("SELECT \* FROM COMPANY WHERE ticker = '{}'".format(self.ticker)).fetchone()

self.ranking = row[0]

self.cik = row[1]

self.name = row[3]

self.sic = row[4]

self.sicDescription = row[5]

self.latest\_filing\_date = row[6]

self.latest\_report\_date = row[7]

self.latest\_primaryDocument = row[8]

self.latest\_accessionNumber = row[9]

self.latest\_form = row[10]

self.latest\_filing\_url = "https://www.sec.gov/Archives/edgar/data/{}/{}/{}.htm".format(self.cik, self.latest\_accessionNumber.replace('-', ''), self.latest\_primaryDocument)

self.latest\_inline\_xbrl = "https://www.sec.gov/ix?doc=/Archives/edgar/data/{}/{}/{}".format(self.cik, self.latest\_accessionNumber.replace('-', ''), self.latest\_primaryDocument)

self.fact\_json\_file = os.path.join(os.path.join(os.environ['FINPYDATA'], "edgar", "api", "xbrl", "companyfacts",'{}.json'.format(self.ticker)))

self.edgar\_html = "<a href={}\_edgar.html>{}</a>".format(self.ticker, self.ticker)

try:

with open(self.fact\_json\_file, 'r') as file:

self.fact\_json = json.load(file)

except:

print("Error loading {} json file".format(ticker))

def get\_cik(self):

return self.cik

def get\_ticker(self):

return self.ticker

def get\_concept\_json(self, concept, accounting='us-gaap'):

try:

concept\_json = self.fact\_json['facts'][accounting][concept]

except:

raise ValueError("concept {} does not exists for {}".format(concept, self.ticker))

return concept\_json

def get\_concept\_quaterly\_df(self, concept, accounting='us-gaap', units='USD'):

concept\_json = self.get\_concept\_json(concept, accounting)

df = pd.DataFrame.from\_records(concept\_json['units'][units])

df = df[df['frame'].notna()]

df['start'] = pd.to\_datetime(df.loc[:]['start'])

df['end'] = pd.to\_datetime(df.loc[:]['end'])

qf = df[df['frame'].str.contains('Q')]

qf.loc[:]['frame'] = qf['frame'].str[2:]

qf = qf.set\_index(pd.PeriodIndex(qf['frame'], freq='Q'))

qf = qf.reindex(pd.PeriodIndex(pd.date\_range( qf.iloc[0]['start'], qf.iloc[-1]['end'],freq = 'Q')))

nan\_indexes = qf[qf['start'].isna()].index.tolist()

row\_above\_nan = [x-1 for x in nan\_indexes]

row\_below\_nan = [x+1 for x in nan\_indexes]

qf.loc[qf['start'].isna(),'start'] = list(qf.loc[row\_above\_nan, 'end'] + pd.Timedelta(1, 'd'))

qf.loc[qf['end'].isna(),'end'] = list(qf.loc[row\_below\_nan, 'start'] - pd.Timedelta(1, 'd'))

val\_nan\_row = qf.loc[qf['val'].isna()].index.tolist()

row3\_above\_val\_nan = [x - 3 for x in val\_nan\_row]

start = list(qf.loc[row3\_above\_val\_nan,'start'])

end = list(qf.loc[val\_nan\_row,'end'])

year\_period = pd.DataFrame({'start': start, 'end': end})

merged = df.merge(year\_period, how='outer', indicator=True)

merged[merged['\_merge'] =='both'].val

qf.loc[val\_nan\_row, 'val'] = 0

qf.loc[val\_nan\_row, 'val'] = list(merged[merged['\_merge'] =='both'].val) - qf['val'].rolling(4).sum()[val\_nan\_row]

qf.loc[val\_nan\_row, 'accn'] = list(merged[merged['\_merge'] =='both'].accn)

qf.loc[val\_nan\_row, 'filed'] = list(merged[merged['\_merge'] =='both'].filed)

qf = qf.rename(columns={'val': concept})

return qf

def get\_concept\_yearly\_df(self, concept, accounting='us-gaap', units='USD'):

concept\_json = self.get\_concept\_json(concept, accounting)

try:

df = pd.DataFrame.from\_records(concept\_json['units'][units])

except:

raise ValueError("unit does not exists in concept {} for {}".format(concept, self.ticker))

try:

df = df[df['frame'].notna()]

except:

raise ValueError("frame does not exists in concept {} for {}".format(concept, self.ticker))

df['start'] = pd.to\_datetime(df.loc[:]['start'])

df['end'] = pd.to\_datetime(df.loc[:]['end'])

df['timedelta'] = df['end'] - df['start']

one\_year = pd.Timedelta(days=300)

df = df[(df['timedelta'] >= one\_year)].sort\_values(by='start')

df.loc[:]['frame'] = df['frame'].str[2:]

df = df.set\_index(pd.PeriodIndex(df['frame'], freq='Y'))

df = df.rename(columns={'val': concept})

return df

def plot\_concept\_quaterly(self, concept, accounting='us-gaap', type = "Bar"):

qf = self.get\_concept\_quaterly\_df(concept, accounting)

g = Bar(height=qf[concept], label=concept)

g.set\_xticklabels(list(qf[concept].index.strftime("%YQ%q")), "categories")

g.option["axis"]["x"]["tick"]["rotate"] = 90

return(g.savefig(html\_file="c3\_bar.html", width="800px", height="800px"))

def plot\_concept\_yearly(self, concept, accounting='us-gaap', type = "Bar"):

qf = self.get\_concept\_yearly\_df(concept, accounting)

if self.debug:

print(qf)

g = Bar(height=qf[concept], label=concept)

g.set\_xticklabels(list(qf[concept].index.strftime("%Y")), "categories")

g.option["axis"]["x"]["tick"]["rotate"] = 90

return(g.savefig(html\_file="c3\_bar.html", width="800px", height="800px"))

def get\_concepts(self, concept, duplicated\_list=[], accounting='us-gaap', drop\_columns=["accn", "fy", "fp", "frame", "timedelta"]):

"""

The argument of concept should be in the following example format.

concept = [{"name" : "NetIncomeLoss", "units" : 'USD'},

{"name" : "ProfitLoss", "units" : 'USD'},

{"name" : "RevenueFromContractWithCustomerExcludingAssessedTax", "units" : 'USD'},

{"name" : "Revenues", "units" : 'USD'}

]

duplicated\_list = [

"Net Income": ["NetIncomeLoss", "ProfitLoss"],

"Revenues" : ["RevenueFromContractWithCustomerExcludingAssessedTax", "Revenues"]

]

"""

concepts = {}

for i in concept:

try:

df = self.get\_concept\_yearly\_df(i['name'], accounting, i['units'])

except ValueError as e:

logger.error(e.args)

continue

df = df.drop(columns=drop\_columns)

concepts[i['name']] = df

if duplicated\_list:

concepts = self.remove\_duplicated\_concepts(concepts, duplicated\_list)

return concepts

def remove\_duplicated\_concepts(self, concepts, duplicated\_list=[]):

"""

concepts should be from the function of get\_concepts.

It is a dictionary of dataframes. The keys of the dictionary are the concepts of edgar. The dataframes has the facts

of these concepts.

The following is an example of duplicated lists.

duplicated\_list = [

["NetIncomeLoss", "ProfitLoss"],

["RevenueFromContractWithCustomerExcludingAssessedTax", "Revenues"]

]

For example, it checks the latest date of NetIncomeLoss and ProfitIncomeLoss from the duplicated list.

It removes the keys with an earlier date and renames the key.

"""

for l in duplicated\_list:

dl = []

for i in l:

if (i in concepts) and (not concepts[i].empty):

dl.append(i)

if not dl:

logger.error("All items in {} do not exist in {}.".format(l, self.ticker))

if len(dl) > 1:

p = dl[0]

for i in range(1, len(dl)):

if ((concepts[dl[i]]['end'][-1] > concepts[p]['end'][-1]) or ((concepts[dl[i]]['end'][-1] == concepts[p]['end'][-1]) and (concepts[dl[i]]['end'][0] < concepts[p]['end'][0]))):

del concepts[p]

p = i

else:

del concepts[dl[i]]

return concepts

def get\_latest\_filing(self, cik\_json, forms = ["10-Q", "10-K"]):

filings\_recent = zip(cik\_json['filings']['recent']['form'],

cik\_json['filings']['recent']['accessionNumber'],

cik\_json['filings']['recent']['filingDate'],

cik\_json['filings']['recent']['reportDate'],

cik\_json['filings']['recent']['primaryDocument']

)

for i in filings\_recent:

if i[0] in forms:

return(i)

def find\_form\_accessionNumbers(self, form):

"""

All the sec forms:

13F-HR

8-K

10-K

10-Q

"""

form\_accessionNumbers = []

accessionNumber = zip(self.cik\_json['filings']['recent']['form'],

self.cik\_json['filings']['recent']['accessionNumber'],

self.cik\_json['filings']['recent']['filingDate'])

for i in accessionNumber:

if i[0] == form:

form\_accessionNumbers.insert(0, (i[1], i[2]))

return form\_accessionNumbers

def get\_all\_forms(self, form):

form\_accessionNumbers = self.find\_form\_accessionNumbers(form)

for a in form\_accessionNumbers:

aN = a[0].replace('-', '')

if not os.path.isdir(os.path.join(self.concept\_dir, aN)):

print("form accession number " + a[0] + "of " + self.cik + " is never processed. fetching it...")

os.makedirs(os.path.join(self.concept\_dir, aN))

if form == "13F-HR":

f13\_html = self.edgar\_root + self.edgar\_data + self.cik + "/" + aN + "/" + a[0] + "-index.html"

if self.debug:

print(f13\_html)

f13\_req = requests.get(f13\_html, headers = self.hdr)

soup = BeautifulSoup(f13\_req.text, 'html.parser')

match\_re = "/xslForm13F\_X01/" + "([\-\d]\*|form13fInfoTable).xml"

pattern = re.compile(match\_re)

xml\_path = ""

for link in soup.find\_all('a'):

href = link.get('href')

if pattern.search(href):

xml\_path = href

html\_file\_name = link.contents[0]

break

if xml\_path == "":

continue

f13\_xml = self.edgar\_root + xml\_path

if self.debug:

print(f13\_xml)

f13\_xml\_req = requests.get(f13\_xml, headers = self.hdr)

f13\_final\_html = os.path.join(self.concept\_dir, aN, html\_file\_name)

with open(f13\_final\_html , 'w') as file:

file.write(f13\_xml\_req.text)

with open(os.path.join(self.concept\_dir, aN, "13F-HR"), 'w') as file:

pass

else:

print("form accession number %s of %s has been processed.", a[0], self.cik)

import asyncio

import aiohttp

from aiolimiter import AsyncLimiter

import time

from datetime import date

import datetime

import pandas as pd

import re

import os

import sqlite3

import json

from contextlib import closing

async def async\_download\_url(url, hdr, limiter, semaphore):

s = time.perf\_counter()

async with aiohttp.ClientSession(headers=hdr) as session:

await semaphore.acquire()

async with limiter:

async with session.get(url) as resp:

content = await resp.text()

semaphore.release()

return content

class download():

def \_\_init\_\_(self, ticker\_info, name, email, debug = True):

self.ranking = ticker\_info['ranking']

self.cik = ticker\_info['cik']

self.ticker = ticker\_info['ticker']

self.name = ticker\_info['name']

self.sic = ticker\_info['sic']

self.sicDescription = ticker\_info['sicDescription']

self.latest\_filing\_date = ticker\_info['latest\_filing\_date']

self.latest\_report\_date = ticker\_info['latest\_report\_date']

self.latest\_primaryDocument = ticker\_info['latest\_primaryDocument']

self.latest\_accessionNumber = ticker\_info['latest\_accessionNumber']

self.latest\_form = ticker\_info['latest\_form']

self.company\_ticker\_json\_file = os.path.join(os.path.join(os.environ['FINPYDATA'], "edgar", "files", "company\_tickers.json"))

self.company\_ticker\_json\_db = os.path.join(os.path.join(os.environ['FINPYDATA'], "edgar", "files", "company\_tickers.db"))

self.hdr = {'User-Agent' : name + email}

self.url = ""

self.debug = debug

@classmethod

async def async\_create(cls, ticker\_info, name, email, nodownload, debug, limiter, semaphore, r):

self = cls(ticker\_info, name, email, debug)

print(self.ticker, self.latest\_filing\_date)

if (self.latest\_filing\_date == None) or (date.today() > (self.latest\_filing\_date + datetime.timedelta(days=90))):

await self.async\_get\_cik\_json(nodownload, limiter, semaphore)

try:

await self.async\_get\_fact\_json(limiter, semaphore)

except:

print("Error getting {} fact json: ".format(self.ticker))

with closing(sqlite3.connect(self.company\_ticker\_json\_db)) as conn:

with closing(conn.cursor()) as cursor:

cursor.execute("""

UPDATE COMPANY SET

sic = ?,

sicDescription = ?,

latest\_filing\_date = ?,

latest\_report\_date = ?,

latest\_primaryDocument = ?,

latest\_accessionNumber = ?,

latest\_form = ? where ticker = ?""", \

(self.sic, self.sicDescription, self.latest\_filing\_date, self.latest\_report\_date, self.latest\_primaryDocument, self.latest\_accessionNumber, self.latest\_form , self.ticker))

conn.commit()

r[self.ticker] = self

return self

async def async\_get\_cik\_json(self, nodownload, limiter, semaphore):

self.edgar\_root = "https://www.sec.gov/"

self.edgar\_data = "Archives/edgar/data/"

url\_str = 'https://data.sec.gov/submissions/CIK{}.json'.format(self.cik)

if not os.path.isdir(os.path.join(os.environ['FINPYDATA'], "edgar", "submissions")):

os.makedirs(os.path.join(os.environ['FINPYDATA'], "edgar", "submissions"))

self.cik\_json\_file = os.path.join(os.environ['FINPYDATA'], "edgar", "submissions", self.ticker + '.json')

if not os.path.exists(self.cik\_json\_file) or (date.fromtimestamp(os.path.getmtime(self.cik\_json\_file)) != date.today()) or not nodownload:

print("Get cjk json of {}. URL:{}".format(self.ticker, url\_str))

content = await async\_download\_url(url\_str, self.hdr, limiter, semaphore)

with open(self.cik\_json\_file, 'w') as file:

file.write(content)

cik\_json = json.loads(content)

else:

with open(self.cik\_json\_file, 'r') as file:

try:

cik\_json = json.load(file)

except:

print("Error loading json file ", self.cik\_json\_file)

exit()

self.sic = cik\_json["sic"]

self.sicDescription = cik\_json["sicDescription"]

filings\_recent = zip(cik\_json['filings']['recent']['form'],

cik\_json['filings']['recent']['accessionNumber'],

cik\_json['filings']['recent']['filingDate'],

cik\_json['filings']['recent']['reportDate'],

cik\_json['filings']['recent']['primaryDocument']

)

fin\_forms = {"8-K", "10-Q", "10-K", "20-K", "20-F", "40-F"}

for i in filings\_recent:

if i[0] in fin\_forms:

self.latest\_form = i[0]

self.latest\_accessionNumber = i[1]

self.latest\_filing\_date = date.fromisoformat(i[2])

self.latest\_report\_date = date.fromisoformat(i[3])

self.latest\_primaryDocument = i[4]

print(self.ticker, "the latest filing and report date of 10-Q or 10-K", self.latest\_filing\_date, self.latest\_report\_date)

break

def get\_all\_forms(self, form):

form\_accessionNumbers = self.find\_form\_accessionNumbers(form)

for a in form\_accessionNumbers:

aN = a[0].replace('-', '')

if not os.path.isdir(os.path.join(self.concept\_dir, aN)):

print("form accession number " + a[0] + "of " + self.cik + " is never processed. fetching it...")

os.makedirs(os.path.join(self.concept\_dir, aN))

if form == "13F-HR":

f13\_html = self.edgar\_root + self.edgar\_data + self.cik + "/" + aN + "/" + a[0] + "-index.html"

if self.debug:

print(f13\_html)

f13\_req = requests.get(f13\_html, headers = self.hdr)

soup = BeautifulSoup(f13\_req.text, 'html.parser')

match\_re = "/xslForm13F\_X01/" + "([\-\d]\*|form13fInfoTable).xml"

pattern = re.compile(match\_re)

xml\_path = ""

for link in soup.find\_all('a'):

href = link.get('href')

async def async\_get\_fact\_json(self, limiter, semaphore):

url\_str = 'https://data.sec.gov/api/xbrl/companyfacts/CIK{}.json'.format(self.cik)

if not os.path.isdir(os.path.join(os.environ['FINPYDATA'], "edgar", "api", "xbrl", "companyfacts")):

os.makedirs(os.path.join(os.environ['FINPYDATA'], "edgar", "api", "xbrl", "companyfacts"));

self.fact\_json\_file = os.path.join(os.path.join(os.environ['FINPYDATA'], "edgar", "api", "xbrl", "companyfacts",'{}.json'.format(self.ticker)))

if self.debug:

print(self.fact\_json\_file)

if os.path.isfile(self.fact\_json\_file):

print("TICKER latest\_filing\_date, fact\_json\_file\_time")

print(self.ticker, self.latest\_filing\_date, date.fromtimestamp(os.path.getmtime(self.fact\_json\_file)))

try:

if not os.path.isfile(self.fact\_json\_file) or self.latest\_filing\_date > date.fromtimestamp(os.path.getmtime(self.fact\_json\_file)):

if self.debug:

print(url\_str)

content = await async\_download\_url(url\_str, self.hdr, limiter, semaphore)

with open(self.fact\_json\_file, 'w') as file:

file.write(content)

except:

print("Error: {}, latest\_filing\_date: {}".format(self.ticker, self.latest\_filing\_date))

"""

(c) 2013 Tsung-Han Yang

This source code is released under the Apache license.

Created on April 1, 2013

"""

import datetime as dt

import pandas as pd

import numpy as np

import finpy.utils.fpdateutil as du

import finpy.data.dataaccess as da

import finpy.utils.utils as ut

from .fincommon import FinCommon

from finpy.edgar.company import company

import os

import sqlite3

def get\_tickdata(ls\_symbols, ldt\_timestamps, csv\_col = [], fill=True, df=pd.DataFrame, actions=True, concepts=[]):

"""

To get all price data of all tickers in ls\_symbols within the list of ldt\_timestamps

:param ls\_symbols: A list with all tickers

:param ldt\_timestamps: A list with all trading days within the time frame.

:param fill: Whether to fill invalid data. Default is True.

"""

c\_dataobj = da.DataAccess("Yahoo", cachestalltime=0)

if csv\_col:

ls\_keys = csv\_col

else:

ls\_keys = ['open', 'high', 'low', 'actual\_close', 'close', 'volume']

ldf\_data = c\_dataobj.get\_data(ldt\_timestamps, ls\_symbols, ls\_keys, actions)

d\_data = dict(list(zip(ls\_symbols, ldf\_data)))

if fill == True:

for s\_key in ls\_symbols:

d\_data[s\_key] = d\_data[s\_key].fillna(method = 'ffill')

d\_data[s\_key] = d\_data[s\_key].fillna(method = 'bfill')

d\_data[s\_key] = d\_data[s\_key].fillna(1.0)

stocks = dict()

for s in ls\_symbols:

stocks[s] = df(index=ldt\_timestamps, data=d\_data[s])

stocks[s]['shares'] = np.nan

stocks[s].loc[ldt\_timestamps[0],'shares'] = 0

if len(concepts) != 0:

company\_ticker\_json\_db = os.path.join(os.path.join(os.environ['FINPYDATA'], "edgar", "files", "company\_tickers.db"))

try:

conn = sqlite3.connect(company\_ticker\_json\_db)

except:

print("Please run downlaod\_edgar.py to create company db")

for s in ls\_symbols:

c = company("NVDA", conn)

facts = c.get\_concepts(concepts)

return stocks

"""

(c) 2013 Tsung-Han Yang

This source code is released under the Apache license.

blacksburg98@yahoo.com

Created on April 1, 2013

"""

import numpy as np

from finpy.utils import utils as ut

class FinCommon():

"""

This class has some common functions used by both Equity and Portfolio.

This is an abstract class.

"""

def avg\_daily\_return(self):

"""

Average of the daily\_return list

:return np.average(self.daily\_return()):

"""

return np.average(self.daily\_return())

def beta\_alpha(self, benchmark):

"""

benchmark is an Equity representing the market.

It can be S&P 500, Russel 2000, or your choice of market indicator.

This function uses polyfit in numpy to find the closest linear equation.

:return beta:

:return alpha:

"""

beta, alpha = np.polyfit(benchmark.normalized(), self.normalized(), 1)

return beta, alpha

def beta(self, benchmark):

"""

benchmark is an Equity representing the market.

This function uses cov in numpy to calculate beta.

"""

benchmark\_close = benchmark.normalized()

C = np.cov(benchmark\_close, self.normalized())/np.var(benchmark\_close)

beta = C[0][1]/C[0][0]

return beta

def sharpe\_ratio(self, rf\_tick="$TNX"):

"""

Return the Original Sharpe Ratio.

https://en.wikipedia.org/wiki/Sharpe\_ratio

rf\_tick is Ten-Year treasury rate ticker at Yahoo.

"""

return self.mean\_excess\_return(rf\_tick)/self.excess\_risk(rf\_tick)

def info\_ratio(self, benchmark, rf\_tick="$TNX"):

"""

Information Ratio

https://en.wikipedia.org/wiki/Information\_ratio

Information Ratio is defined as active return divided by active risk,

where active return is the difference between the return of the security

and the return of a selected benchmark index, and active risk is the

standard deviation of the active return.

"""

return self.mean\_active\_return(benchmark)/self.active\_risk(benchmark)

def appraisal\_ratio(self, benchmark, rf\_tick="$TNX"):

"""

Appraisal Ratio

https://en.wikipedia.org/wiki/Appraisal\_ratio

Appraisal Ratio is defined as residual return divided by residual risk,

where residual return is the difference between the return of the security

and the return of a selected benchmark index, and residual risk is the

standard deviation of the residual return.

"""

return self.mean\_residual\_return(benchmark, rf\_tick)/self.residual\_risk(benchmark, rf\_tick)

def excess\_return(self, rf\_tick="$TNX"):

"""

An active return is the difference between the benchmark and the actual return.

"""

return self.daily\_return() - ut.riskfree\_return(self.ldt\_timestamps(), rf\_tick="$TNX")

def mean\_excess\_return(self, rf\_tick="$TNX"):

return np.mean(self.excess\_return(rf\_tick))

def excess\_risk(self, rf\_tick="$TNX"):

"""

$FVX is another option. Five-Year treasury rate.

An excess risk is the standard deviation of the excess return.

"""

return np.std(self.excess\_return(rf\_tick))

def active\_return(self, benchmark):

"""

An active return is the difference between the benchmark and the actual return.

"""

return self.daily\_return() - benchmark.daily\_return()

def mean\_active\_return(self, benchmark):

return np.mean(self.active\_return(benchmark))

def active\_risk(self, benchmark):

"""

An active risk is the standard deviation of the active return.

"""

return np.std(self.active\_return(benchmark))

def residual\_return(self, benchmark, rf\_tick="$TNX"):

"""

A residual return is the excess return minus beta times the benchmark excess return.

"""

beta = self.beta(benchmark)

return self.excess\_return(rf\_tick="$TNX") - beta \* benchmark.excess\_return(rf\_tick="$TNX")

def residual\_risk(self, benchmark, rf\_tick="$TNX"):

"""

Residual Risk is the standard deviation of the residual return.

"""

return np.std(self.residual\_return(benchmark, rf\_tick))

def mean\_residual\_return(self, benchmark, rf\_tick="$TNX"):

return np.mean(self.residual\_return(benchmark, rf\_tick))

class Transaction():

def \_\_init\_\_(self, buy\_date, buy\_price, sell\_date=None, sell\_price=None):

self.buy\_date = buy\_date

self.sell\_date = sell\_date

self.buy\_price = buy\_price

self.sell\_price = sell\_price

"""

(c) 2013 Tsung-Han Yang

This source code is released under the Apache license.

blacksburg98@yahoo.com

Created on April 1, 2013

"""

import datetime as dt

import pandas as pd

import numpy as np

import random

import csv

from .order import Order

from .fincommon import FinCommon

import finpy.utils.fpdateutil as du

from finpy.utils import utils as ut

from finpy.financial.equity import get\_tickdata

class Portfolio():

"""

Portfolio has three items.

equities is a panda Panel of equity data.

Reference by ticker. self.equities['AAPL']

cash is a pandas series with daily cash balance.

total is the daily balance.

order\_list is a list of Order

"""

def \_\_init\_\_(self, equities, cash, dates, order\_list=None):

self.equities = pd.concat(equities, names=["tick", "date"])

self.equities.sort\_index(inplace=True)

# self.equities = self.equities.reorder\_levels(order=["date", "tick"])

"""

:var equities: is a Panel of equities.

"""

if order\_list == None:

self.order = pd.DataFrame(columns=['tick', 'date', 'action', 'shares', 'price'])

self.order = self.order.set\_index(["tick","date"])

else:

ol = order\_list

ol.sort(key=lambda x: x.date)

self.order = pd.DataFrame.from\_records([s.to\_dict() for s in ol])

self.order = self.order.set\_index(["tick","date"])

xi = self.order[self.order["price"].isnull()].index

self.order.loc[xi, "price"] = self.equities.loc[xi, "close"]

self.cash = pd.Series(index=dates)

self.cash[0] = cash

self.total = pd.Series(index=dates)

self.total[0] = self.dailysum(dates[0])

self.dates = dates

def dailysum(self, date):

" Calculate the total balance of the date."

equities\_total = np.nansum(self.equities.xs(key=date, level=1)['shares'] \* self.equities.xs(key=date, level=1)['close'])

total = equities\_total + self.cash[date]

return total

def buy(self, shares, tick, price, date, update\_ol=False):

"""

Portfolio Buy

Calculate total, shares and cash upto the date.

Before we buy, we need to update share numbers. "

"""

self.cal\_total(date)

last\_valid = self.equities.loc[(tick,slice(None)),'shares'].last\_valid\_index()[1]

self.equities.loc[(tick, slice(last\_valid, date)), 'shares'] = self.equities.loc[(tick, last\_valid), 'shares']

self.equities.loc[(tick, date), 'shares'] += shares

self.cash[date] -= price\*shares

self.total[date] = self.dailysum(date)

if update\_ol:

self.order = self.order.append(pd.DataFrame({"action": "buy", "shares" : shares, "price": self.equities.loc[(tick, date), 'close']}, [(tick, date)]))

def sell(self, shares, tick, price, date, update\_ol=False):

"""

Portfolio sell

Calculate shares and cash upto the date.

"""

self.cal\_total(date)

last\_valid = self.equities.loc[(tick,slice(None)),'shares'].last\_valid\_index()[1]

self.equities.loc[(tick, slice(last\_valid, date)), 'shares'] = self.equities.loc[(tick, last\_valid), 'shares']

self.equities.loc[(tick, date), 'shares'] -= shares

self.cash[date] += price\*shares

self.total[date] = self.dailysum(date)

if update\_ol:

self.order = self.order.append(pd.DataFrame({"action": "sell", "shares" : shares, "price": self.equities.loc[(tick, date), 'close']}, [(tick, date)]))

def fillna\_cash(self, date):

" fillna on cash up to date "

update\_start = self.cash.last\_valid\_index()

update\_end = date

self.cash[update\_start:update\_end] = self.cash[update\_start]

return update\_start, update\_end

def fillna(self, date):

"""

fillna cash and all equities.

return update\_start and update\_end.

"""

update\_start, update\_end = self.fillna\_cash(date)

for tick in self.equities.index.unique(0).tolist():

self.equities.loc[(tick, slice(update\_start, update\_end)),'shares'] = self.equities.loc[(tick, update\_start), 'shares']

return update\_start, update\_end

def cal\_total(self, date=None):

"""

Calculate total up to "date".

"""

if date == None:

equities\_sum = pd.Series(index=self.ldt\_timestamps())

each\_total = self.equities.loc[(slice(None),slice(None)),'close'] \* self.equities.loc[(slice(None),slice(None)),'shares']

equities\_sum = each\_total.groupby(level=1).sum()

self.total = self.cash + equities\_sum

else:

start, end = self.fillna(date)

equities\_total\_df = self.equities.loc[(slice(None),slice(start,end)),'shares'] \* self.equities.loc[(slice(None),slice(start,end)),'close']

equities\_total = equities\_total\_df.groupby(level=1).sum()

self.total[start:end ] = equities\_total + self.cash[start:end]

def put\_orders(self):

"""

Put the order list to the DataFrame.

Update shares, cash columns of each Equity

"""

for o in self.order:

if o.action.lower() == "buy":

self.buy(date=o.date, shares=np.float(o.shares), price=np.float(o.price), tick=o.tick)

elif o.action.lower() == "sell":

self.sell(shares=np.float(o.shares), tick=o.tick, price=np.float(o.price), date=o.date)

def sim(self, ldt\_timestamps=None):

"""

Go through each day and calculate total and cash.

"""

self.put\_orders()

if ldt\_timestamps == None:

ldt\_timestamps = self.ldt\_timestamps()

dt\_end = ldt\_timestamps[-1]

self.cal\_total()

def csvwriter(self, equity\_col=None, csv\_file="pf.csv", total=True, cash=True, d=','):

"""

Write the content of the Portfolio to a csv file.

If total is True, the total is printed to the csv file.

If cash is True, the cash is printed to the csv file.

equity\_col specify which columns to print for an equity.

The specified columns of each equity will be printed.

"""

lines = []

l = []

l.append("Date")

if total:

l.append("Total")

if cash:

l.append("Cash")

if equity\_col != None:

for e in self.equities:

for col in equity\_col:

label = e + col

l.append(label)

lines.append(l)

for i in self.ldt\_timestamps():

l = []

l.append(i.strftime("%Y-%m-%d"))

if total:

l.append(round(self.total[i], 2))

if cash:

l.append(round(self.cash[i], 2))

if equity\_col != None:

for e in self.equities.index.droplevel(1).drop\_duplicates():

for col in equity\_col:

l.append(round(self.equities.loc[(e, i), col], 2))

lines.append(l)

with open(csv\_file, 'w') as fp:

cw = csv.writer(fp, lineterminator='\n', delimiter=d)

for line in lines:

cw.writerow(line)

def write\_order\_csv(self, csv\_file="pf\_order.csv", d=','):

self.order.reorder\_levels(["date", "tick"]).to\_csv(path\_or\_buf = csv\_file, sep = d, header = False, columns = ["action", "shares"])

def daily\_return(self,tick=None):

"""

Return the return rate of each day, a list.

:param tick: The ticker of the equity.

:type string:

"""

if tick == None:

total = self.total

else:

total = self.equities.loc[(tick,slice(None)),'close'].droplevel(0)

daily\_rtn = total/total.shift(1)-1

daily\_rtn[0] = 0

return np.array(daily\_rtn)

def avg\_daily\_return(self, tick=None):

" Average of the daily\_return list "

return np.average(self.daily\_return(tick))

def std(self, tick=None):

" Standard Deviation of the daily\_return "

return np.std(self.daily\_return(tick))

def normalized(self, tick=None):

start = self.ldt\_timestamps()[0]

if tick == None:

return self.total/self.total[0]

else:

return (self.equities.loc[(tick, slice(None)), 'close']/self.equities.loc[(tick, start), 'close']).droplevel(0)

def normalized\_price(self, tick):

self.equities.loc[(tick, slice(None)),'open'] = self.equities.loc[(tick, slice(None)),'open'] \* self.equities.loc[(tick, slice(None)),'close']/self.equities.loc[(tick, slice(None)),'actual\_close']

self.equities.loc[(tick, slice(None)),'high'] = self.equities.loc[(tick, slice(None)),'high'] \* self.equities.loc[(tick, slice(None)),'close']/self.equities.loc[(tick, slice(None)),'actual\_close']

self.equities.loc[(tick, slice(None)),'low'] = self.equities.loc[(tick, slice(None)),'low'] \* self.equities.loc[(tick, slice(None)),'close']/self.equities.loc[(tick, slice(None)),'actual\_close']

def sortino(self, k=252, tick=None):

"""

Return Sortino Ratio.

You can overwirte the coefficient with k.

The default is 252.

"""

daily\_rtn = self.daily\_return(tick)

negative\_daily\_rtn = daily\_rtn[daily\_rtn < 0]

sortino\_dev = np.std( negative\_daily\_rtn)

sortino = (self.avg\_daily\_return(tick) / sortino\_dev) \* np.sqrt(k)

return sortino

def return\_ratio(self, tick=None):

" Return the return ratio of the period "

if tick == None:

return self.total[-1]/self.total[0]

else:

return self.equities.loc[(tick, self.ldt\_timestamps()[-1]), 'close']/self.equities.loc[(tick, self.ldt\_timestamps()[0]), 'close']

def moving\_average(self, window=20, tick=None):

"""

Return an array of moving average. Window specified how many days in

a window.

"""

if tick == None:

ma = pd.stats.moments.rolling\_mean(self.total, window=window)

else:

ma = self.equities[tick].stats.moments.rolling\_mean(window=window)

ma[0:window] = ma[window]

return ma

def drawdown(self, window=10):

"""

Find the peak within the retrospective window.

Drawdown is the difference between the peak and the current value.

"""

ldt\_timestamps = self.ldt\_timestamps()

pre\_timestamps = ut.pre\_timestamps(ldt\_timestamps, window)

# ldf\_data has the data prior to our current interest.

# This is used to calculate moving average for the first window.

merged\_data = self.total[pd.Index(pre\_timestamps[0]), ldt\_timestamps[-1]]

total\_timestamps = merged\_data.index

dd = pd.Series(index=ldt\_timestamps)

j = 0

for i in range(len(pre\_timestamps), len(total\_timestamps)):

win\_start = total\_timestamps[i - window]

win\_end = total\_timestamps[i]

ts\_value = merged\_data[win\_start:win\_end]

current = merged\_data[win\_end]

peak = np.amax(ts\_value)

dd[j] = (peak-current)/peak

j += 1

return dd

def random\_choose\_tick(self, exclude=[]):

"""

Randomly return a ticker in the portfolio.

The items in exclude list are not in the select pool.

"""

ex\_set = set(exclude)

pf\_set = set([x for x in self.equities])

sel\_ls = [s for s in pf\_set - ex\_set]

return random.choice(sel\_ls)

def equities\_long(self, date):

"""

Return the list of long equities on the date.

"Long equities" means the number of shares of the equity is greater than 0.

"""

return [x for x in self.equities if self.equities[x].shares[date] > 0]

def ldt\_timestamps(self):

"""

Return an array of datetime objects.

"""

ldt\_index = self.total.index

dt\_start = ldt\_index[0]

dt\_end = ldt\_index[-1]

dt\_timeofday = dt.timedelta(hours=16)

ldt\_timestamps = du.getNYSEdays(dt\_start, dt\_end, dt\_timeofday)

return ldt\_timestamps

def excess\_return(self, rf\_tick="$TNX", tick=None):

"""

An excess return is the difference between an asset's return and the riskless rate.

"""

return self.daily\_return(tick=tick) - ut.riskfree\_return(self.ldt\_timestamps(), rf\_tick=rf\_tick)

def mean\_excess\_return(self, rf\_tick="$TNX", tick=None):

return np.mean(self.excess\_return(rf\_tick=rf\_tick, tick=tick))

def residual\_return(self, benchmark, rf\_tick="$TNX", tick=None):

"""

A residual return is the excess return minus beta times the benchmark excess return.

"""

beta = self.beta(benchmark, tick)

return self.excess\_return(rf\_tick=rf\_tick, tick=tick) - beta \* self.excess\_return(rf\_tick=rf\_tick, tick=benchmark)

def mean\_residual\_return(self, benchmark, rf\_tick="$TNX", tick=None):

return np.mean(self.residual\_return(benchmark=benchmark, rf\_tick=rf\_tick, tick=tick))

def residual\_risk(self, benchmark, rf\_tick="$TNX", tick=None):

"""

Residual Risk is the standard deviation of the residual return.

"""

return np.std(self.residual\_return(benchmark=benchmark, rf\_tick=rf\_tick, tick=tick))

def active\_return(self, benchmark, tick=None):

"""

An active return is the difference between the benchmark and the actual return.

"""

return self.daily\_return(tick=tick) - self.daily\_return(tick=benchmark)

def mean\_active\_return(self, benchmark, tick=None):

return np.mean(self.active\_return(benchmark, tick))

def beta\_alpha(self, benchmark):

"""

benchmark is an Equity representing the market.

It can be S&P 500, Russel 2000, or your choice of market indicator.

This function uses polyfit in numpy to find the closest linear equation.

"""

beta, alpha = np.polyfit(self.daily\_return(tick=benchmark), self.daily\_return(), 1)

return beta, alpha

def beta(self, benchmark, tick=None):

"""

benchmark is an Equity representing the market.

This function uses cov in numpy to calculate beta.

"""

benchmark\_return = self.daily\_return(tick=benchmark)

C = np.cov(benchmark\_return, self.daily\_return(tick=tick))/np.var(benchmark\_return)

beta = C[0][1]/C[0][0]

return beta

def excess\_risk(self, rf\_tick="$TNX", tick=None):

"""

$FVX is another option. Five-Year treasury rate.

An excess risk is the standard deviation of the excess return.

"""

return np.std(self.excess\_return(rf\_tick=rf\_tick, tick=tick))

def active\_risk(self, benchmark, tick=None):

"""

An active risk is the standard deviation of the active return.

"""

return np.std(self.active\_return(benchmark, tick))

def info\_ratio(self, benchmark, rf\_tick="$TNX", tick=None):

"""

Information Ratio

https://en.wikipedia.org/wiki/Information\_ratio

Information Ratio is defined as active return divided by active risk,

where active return is the difference between the return of the security

and the return of a selected benchmark index, and active risk is the

standard deviation of the active return.

"""

return self.mean\_active\_return(benchmark=benchmark, tick=tick)/self.active\_risk(benchmark=benchmark, tick=tick)

def appraisal\_ratio(self, benchmark, rf\_tick="$TNX", tick=None):

"""

Appraisal Ratio

https://en.wikipedia.org/wiki/Appraisal\_ratio

Appraisal Ratio is defined as residual return divided by residual risk,

where residual return is the difference between the return of the security

and the return of a selected benchmark index, and residual risk is the

standard deviation of the residual return.

"""

return self.mean\_residual\_return(benchmark, rf\_tick, tick)/self.residual\_risk(benchmark, rf\_tick, tick)

def sharpe\_ratio(self, rf\_tick="$TNX", tick=None):

"""

Return the Original Sharpe Ratio.

https://en.wikipedia.org/wiki/Sharpe\_ratio

rf\_tick is Ten-Year treasury rate ticker at Yahoo.

"""

return self.mean\_excess\_return(rf\_tick=rf\_tick, tick=tick)/self.excess\_risk(rf\_tick=rf\_tick, tick=tick)

def up\_ratio(self, date, tick, days=10):

"""

Return the ratio of the past up days.

This function only applies to equities.

"""

ldt\_index = self.ldt\_timestamps()

last = date

first = date-days

up = 0.0

dn = 0.0

for i in range(first, last+1):

if self.equities.loc[(tick, ldt\_index[i]), 'close'] < self.equities.loc[(tick, ldt\_index[i-1]), 'close']:

dn += 1

else:

up += 1

ratio = up / (dn + up)

return ratio

def dn\_ratio(self, date,tick , days=10):

"""

Return the ratio of the past down days.

This function only applies to equities.

"""

ratio = 1.0 - self.up\_ratio(date=date, tick=tick, days=days)

return ratio

def rolling\_normalized\_stdev(self, tick, window=50):

"""

Return the rolling standard deviation of normalized price.

This function only applies to equities.

"""

ldt\_timestamps = self.ldt\_timestamps()

pre\_timestamps = ut.pre\_timestamps(ldt\_timestamps, window)

# ldf\_data has the data prior to our current interest.

# This is used to calculate moving average for the first window.

ldf\_data = get\_tickdata([tick], pre\_timestamps)

pre\_data = pd.concat(ldf\_data, names=["tick", "date"])

merged\_data = pd.concat([pre\_data.loc[(tick, slice(None)), 'close'], self.equities.loc[(tick,slice(None)),'close']])

all\_timestamps = pre\_timestamps.append(ldt\_timestamps)

merged\_daily\_rtn = (self.equities.loc[(tick,slice(None)),'close']/self.equities.loc[(tick,slice(None)),'close'].shift(1)-1)

merged\_daily\_rtn[0] = 0

sigma = merged\_daily\_rtn.rolling(window).std()

return sigma.droplevel(0)[self.ldt\_timestamps()]

def max\_rise(self, tick, date, window=20):

"""

Find the maximum change percentage between the current date and the bottom of the retrospective window.

:param tick: ticker

:type tick: string

:param date: date to calculate max\_rise

:type date: datetime

:param window: The days of window to calculate max\_rise.

:type window: int

"""

ldt\_timestamps = self.ldt\_timestamps()

pre\_timestamps = ut.pre\_timestamps(ldt\_timestamps, window)

first = pre\_timestamps[0]

# ldf\_data has the data prior to our current interest.

# This is used to calculate moving average for the first window.

try:

self.equities.loc[(tick, first), 'close']

merged\_data = self.equties.loc[(tick, slice(None)), 'close']

except:

ldf\_data = get\_tickdata([tick], pre\_timestamps)

pre\_data = pd.concat(ldf\_data, names=["tick", "date"])

merged\_data = pd.concat([pre\_data.loc[(tick, slice(None)), 'close'], self.equities.loc[(tick,slice(None)),'close']])

if(isinstance(date , int)):

int\_date = ldt\_timestamps[date]

else:

int\_date = date

merged\_data = merged\_data.droplevel(0)

c = merged\_data.index.get\_loc(int\_date)

m = merged\_data[c-window:c].min()

r = (merged\_data[c]-m)/merged\_data[c]

return r

def max\_fall(self, tick, date, window=20):

"""

Find the change percentage between the top and the bottom of the retrospective window.

:param tick: ticker

:type tick: string

:param date: date to calculate max\_rise

:type date: datetime

:param window: The days of window to calculate max\_rise.

:type window: int

"""

ldt\_timestamps = self.ldt\_timestamps()

pre\_timestamps = ut.pre\_timestamps(ldt\_timestamps, window)

first = pre\_timestamps[0]

# ldf\_data has the data prior to our current interest.

# This is used to calculate moving average for the first window.

try:

self.equities.loc[(tick, first), 'close']

merged\_data = self.equties.loc[(tick, slice(None)), 'close']

except:

ldf\_data = get\_tickdata([tick], pre\_timestamps)

pre\_data = pd.concat(ldf\_data, names=["tick", "date"])

merged\_data = pd.concat([pre\_data.loc[(tick, slice(None)), 'close'], self.equities.loc[(tick,slice(None)),'close']])

if(isinstance(date , int)):

int\_date = ldt\_timestamps[date]

else:

int\_date = date

merged\_data = merged\_data.droplevel(0)

c = merged\_data.index.get\_loc(int\_date)

mx = merged\_data[c-window:c].max()

mn = merged\_data[c-window:c].min()

r = (mx-mn)/merged\_data[c]

return r

def moving\_average(self, tick, window=20):

"""

Return an array of moving average. Window specified how many days in

a window.

:param tick: ticker

:type tick: string

:param window: The days of window to calculate moving average.

:type window: int

"""

mi = self.bollinger\_band(tick=tick, window=window, mi\_only=True)

return mi

def bollinger\_band(self, tick, window=20, k=2, mi\_only=False):

"""

Return four arrays for Bollinger Band. The upper band at k times an N-period

standard deviation above the moving average. The lower band at k times an N-period

below the moving average.

:param tick: ticker

:type tick: string

:param window: The days of window to calculate Bollinger Band.

:type window: int

:param k: k \*

:return bo: bo['mi'] is the moving average. bo['lo'] is the lower band.

bo['hi'] is the upper band. bo['ba'] is a seris of the position of the current

price relative to the bollinger band.

:type bo: A dictionary of series.

"""

ldt\_timestamps = self.ldt\_timestamps()

pre\_timestamps = ut.pre\_timestamps(ldt\_timestamps, window)

# ldf\_data has the data prior to our current interest.

# This is used to calculate moving average for the first window.

ldf\_data = get\_tickdata([tick], pre\_timestamps)

pre\_data = pd.concat(ldf\_data, names=["tick", "date"])

merged\_data = pd.concat([pre\_data.loc[(tick, slice(None)), 'close'], self.equities.loc[(tick,slice(None)),'close']]).droplevel(0)

bo = dict()

bo['mi'] = merged\_data.rolling(window).mean()[ldt\_timestamps]

if mi\_only:

return bo['mi']

else:

sigma = merged\_data.rolling(window).std()

bo['hi'] = bo['mi'] + k \* sigma[ldt\_timestamps]

bo['lo'] = bo['mi'] - k \* sigma[ldt\_timestamps]

bo['ba'] = (merged\_data[ldt\_timestamps] - bo['mi']) / (k \* sigma[ldt\_timestamps])

return bo

def RSI(self, tick):

"""

Relative Strength Index

http://stockcharts.com/school/doku.php?id=chart\_school:technical\_indicators:relative\_strength\_index\_rsi

This function uses roughly 250 prior points to calculate RS.

:param tick: The ticker to calculate RSI

:type tick: string

:return rsi[ldt\_timestamps]: RSI series

"""

ldt\_timestamps = self.ldt\_timestamps()

pre\_timestamps = ut.pre\_timestamps(ldt\_timestamps, 250)

ldf\_data = get\_tickdata([tick], pre\_timestamps)

merged\_data = pd.concat([ldf\_data[tick]['close'], self.equities[tick]['close']])

delta = merged\_data.diff()

gain = pd.Series(delta[delta > 0], index=delta.index).fillna(0)

loss = pd.Series(delta[delta < 0], index=delta.index).fillna(0).abs()

avg\_gain = pd.Series(index=delta.index)

avg\_loss = pd.Series(index=delta.index)

rsi = pd.Series(index=delta.index)

avg\_gain[14] = gain[1:15].mean()

avg\_loss[14] = loss[1:15].mean()

for i in range(15, len(delta.index)):

avg\_gain[i] = (avg\_gain[i-1]\*13+gain[i])/14

avg\_loss[i] = (avg\_loss[i-1]\*13+loss[i])/14

if avg\_loss[i] == 0:

rsi[i] = 100

else:

rs = avg\_gain[i]/avg\_loss[i]

rsi[i] = 100 - 100/(1+rs)

return(rsi[ldt\_timestamps])

import pandas as pd

def sp500():

table=pd.read\_html('https://en.wikipedia.org/wiki/List\_of\_S%26P\_500\_companies')

df = table[0]

return list(df["Symbol"].str.replace('.', '-', regex=False))

def russel3000(format = 'list'):

table = pd.read\_csv('https://www.ishares.com/us/products/239714/ishares-russell-3000-etf/1467271812596.ajax?fileType=csv&fileName=IWV\_holdings&dataType=fund', header=9, index\_col=False)

table = table[table.Ticker.str.match('^[A-Z]\*$')]

table = table[~table["Exchange"].str.contains("NO MARKET")]

table["Ticker"] = table["Ticker"].str.replace('BRKB', 'BRK-B', regex=False)

table["Ticker"] = table["Ticker"].str.replace('^BFB$', 'BF-B', regex=True)

table["Ticker"] = table["Ticker"].str.replace('^BFA$', 'BF-A', regex=True)

if format == 'list':

return list(table["Ticker"])

else:

table.insert(0, 'Ranking', table.index)

return table

def custom(file):

f = open(file, "r")

l = f.read().splitlines()

return l

'''

(c) 2011, 2012 Georgia Tech Research Corporation

This source code is released under the New BSD license. Please see

http://wiki.quantsoftware.org/index.php?title=QSTK\_License

for license details.

Created on Jan 16, 2013

@author: Sourabh Bajaj

@contact: sourabhbajaj90@gmail.com

@summary: EventProfiler

'''

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import finpy.dataaccess as da

import finpy.fpdateutil as du

def eventprofiler(df\_events, all\_stocks, i\_lookback=20, i\_lookforward=20,

s\_filename='study', b\_market\_neutral=True, b\_errorbars=True,

s\_market\_sym='SPY', out\_pict=False):

''' Event Profiler for an event matix'''

df\_tmpclose = {}

df\_tmprets = {}

for x in all\_stocks:

df\_tmpclose[x] = all\_stocks[x]['close'].copy()

df\_tmprets[x] = pd.Series(all\_stocks[x].daily\_return(), index=df\_tmpclose[x].index)

df\_close = pd.DataFrame(df\_tmpclose)

df\_rets = pd.DataFrame(df\_tmprets)

if b\_market\_neutral == True:

df\_rets = df\_rets - df\_rets[s\_market\_sym]

del df\_rets[s\_market\_sym]

del df\_events[s\_market\_sym]

df\_close = df\_close.reindex(columns=df\_events.columns)

# Removing the starting and the end events

df\_events.values[0:i\_lookback, :] = np.nan

df\_events.values[-i\_lookforward:, :] = np.nan

# Number of events

i\_no\_events = np.nansum(df\_events.values)

# i\_no\_events = 0

# for i in df\_events:

# for j in df\_events[i]:

# if j == 1:

# i\_no\_events += 1

na\_event\_rets = "False"

# Looking for the events and pushing them to a matrix

for i, s\_sym in enumerate(df\_events.columns):

for j, dt\_date in enumerate(df\_events.index):

if df\_events[s\_sym][dt\_date] == 1:

na\_ret = df\_rets[s\_sym][j - i\_lookback:j + 1 + i\_lookforward]

if type(na\_event\_rets) == type(""):

na\_event\_rets = na\_ret

else:

na\_event\_rets = np.vstack((na\_event\_rets, na\_ret))

# Computing daily rets and retuns

na\_event\_rets = np.cumprod(na\_event\_rets + 1, axis=1)

na\_event\_rets = (na\_event\_rets.T / na\_event\_rets[:, i\_lookback]).T

# Study Params

na\_mean = np.mean(na\_event\_rets, axis=0)

na\_std = np.std(na\_event\_rets, axis=0)

li\_time = list(range(-i\_lookback, i\_lookforward + 1))

# Plotting the chart

if out\_pict:

plt.clf()

plt.axhline(y=1.0, xmin=-i\_lookback, xmax=i\_lookforward, color='k')

if b\_errorbars == True:

plt.errorbar(li\_time[i\_lookback:], na\_mean[i\_lookback:],

yerr=na\_std[i\_lookback:], ecolor='#AAAAFF',

alpha=0.1)

plt.plot(li\_time, na\_mean, linewidth=3, label='mean', color='b')

plt.xlim(-i\_lookback - 1, i\_lookforward + 1)

if b\_market\_neutral == True:

plt.title('Market Relative mean return of ' +\

str(i\_no\_events) + ' events')

else:

plt.title('Mean return of ' + str(i\_no\_events) + ' events')

plt.xlabel('Days')

plt.ylabel('Cumulative Returns')

plt.savefig(s\_filename, format='pdf')

return i\_no\_events

'''

(c) 2011, 2012 Georgia Tech Research Corporation

This source code is released under the New BSD license. Please see

http://wiki.quantsoftware.org/index.php?title=QSTK\_License

for license details.

Created on Jan 1, 2011

@author:Drew Bratcher

@contact: dbratcher@gatech.edu

@summary: Contains tutorial for backtester and report.

'''

import datetime as dt

from datetime import timedelta

import time as t

import numpy as np

import os

import pandas as pd

def \_cache\_dates():

''' Caches dates '''

try:

# filename = os.environ['QS'] + "/qstkutil/NYSE\_dates.txt"

filename = os.path.join(os.path.dirname(\_\_file\_\_), 'NYSE\_dates.txt')

except KeyError:

print("Please be sure you have NYSE\_dates.txt in the finpy/utils directory")

with open(filename) as f:

datestxt = [x.strip('\n') for x in f]

dates = []

for i in datestxt:

dates.append(dt.datetime.strptime(i, "%m/%d/%Y"))

return pd.Series(index=dates, data=dates)

GTS\_DATES = \_cache\_dates()

def getMonthNames():

return(['JAN','FEB','MAR','APR','MAY','JUN','JUL','AUG','SEP','OCT','NOV','DEC'])

def getYears(funds):

years=[]

for date in funds.index:

if(not(date.year in years)):

years.append(date.year)

return(years)

def getMonths(funds,year):

months=[]

for date in funds.index:

if((date.year==year) and not(date.month in months)):

months.append(date.month)

return(months)

def getDays(funds,year,month):

days=[]

for date in funds.index:

if((date.year==year) and (date.month==month)):

days.append(date)

return(days)

def getDaysBetween(ts\_start, ts\_end):

days=[]

for i in range(0,(ts\_end-ts\_start).days):

days.append(ts\_start+timedelta(days=1)\*i)

return(days)

def getFirstDay(funds,year,month):

for date in funds.index:

if((date.year==year) and (date.month==month)):

return(date)

return('ERROR')

def getLastDay(funds,year,month):

return\_date = 'ERROR'

for date in funds.index:

if((date.year==year) and (date.month==month)):

return\_date = date

return(return\_date)

def getNextOptionClose(day, trade\_days, offset=0):

#get third friday in month of day

#get first of month

year\_off=0

if day.month+offset > 12:

year\_off = 1

offset = offset - 12

first = dt.datetime(day.year+year\_off, day.month+offset, 1, hour=16)

#get weekday

day\_num = first.weekday()

#get first friday (friday - weekday) add 7 if less than 1

dif = 5 - day\_num

if dif < 1:

dif = dif+7

#move to third friday

dif = dif + 14

friday = first+dt.timedelta(days=(dif-1))

#if friday is a holiday, options expire then

if friday in trade\_days:

month\_close = first + dt.timedelta(days=dif)

else:

month\_close = friday

#if day is past the day after that

if month\_close < day:

return\_date = getNextOptionClose(day, trade\_days, offset=1)

else:

return\_date = month\_close

return(return\_date)

def getLastOptionClose(day, trade\_days):

start = day

while getNextOptionClose(day, trade\_days)>=start:

day= day - dt.timedelta(days=1)

return(getNextOptionClose(day, trade\_days))

def getNYSEoffset(mark, offset):

''' Returns NYSE date offset by number of days '''

mark = mark.replace(hour=0, minute=0, second=0, microsecond=0)

i = GTS\_DATES.index.searchsorted(mark, side='right')

# If there is no exact match, take first date in past

if GTS\_DATES[i] != mark:

i -= 1

ret = GTS\_DATES[i + offset]

ret = ret.replace(hour=16)

return ret

def getNYSEdays(startday = dt.datetime(1964,7,5), endday = dt.datetime(2020,12,31),

timeofday = dt.timedelta(0)):

"""

@summary: Create a list of timestamps between startday and endday (inclusive)

that correspond to the days there was trading at the NYSE. This function

depends on a separately created a file that lists all days since July 4,

1962 that the NYSE has been open, going forward to 2020 (based

on the holidays that NYSE recognizes).

@param startday: First timestamp to consider (inclusive)

@param endday: Last day to consider (inclusive)

@return list: of timestamps between startday and endday on which NYSE traded

@rtype datetime

"""

dates = GTS\_DATES[startday:endday]

return(dates)

def getNextNNYSEdays(startday, days, timeofday):

"""

@summary: Create a list of timestamps from startday that is days days long

that correspond to the days there was trading at NYSE. This function

depends on the file used in getNYSEdays and assumes the dates within are

in order.

@param startday: First timestamp to consider (inclusive)

@param days: Number of timestamps to return

@return list: List of timestamps starting at startday on which NYSE traded

@rtype datetime

"""

try:

# filename = os.environ['QS'] + "/qstkutil/NYSE\_dates.txt"

filename = os.path.join(os.path.dirname(\_\_file\_\_), 'NYSE\_dates.txt')

except KeyError:

print("Please be sure to set the value for QS in config.sh or\n")

print("in local.sh and then \'source local.sh\'.\n")

datestxt = np.loadtxt(filename,dtype=str)

dates=[]

for i in datestxt:

if(len(dates)<days):

if((dt.datetime.strptime(i,"%m/%d/%Y")+timeofday)>=startday):

dates.append(dt.datetime.strptime(i,"%m/%d/%Y")+timeofday)

return(dates)

def getPrevNNYSEday(startday, timeofday):

"""

@summary: This function returns the last valid trading day before the start

day, or returns the start day if it is a valid trading day. This function

depends on the file used in getNYSEdays and assumes the dates within are

in order.

@param startday: First timestamp to consider (inclusive)

@param days: Number of timestamps to return

@return list: List of timestamps starting at startday on which NYSE traded

@rtype datetime

"""

try:

# filename = os.environ['QS'] + "/qstkutil/NYSE\_dates.txt"

filename = os.path.join(os.path.dirname(\_\_file\_\_), 'NYSE\_dates.txt')

except KeyError:

print("Please be sure to set the value for QS in config.sh or\n")

print("in local.sh and then \'source local.sh\'.\n")

datestxt = np.loadtxt(filename,dtype=str)

#''' Set return to first day '''

dtReturn = dt.datetime.strptime( datestxt[0],"%m/%d/%Y")+timeofday

#''' Loop through all but first '''

for i in datestxt[1:]:

dtNext = dt.datetime.strptime(i,"%m/%d/%Y")

#''' If we are > startday, then use previous valid day '''

if( dtNext > startday ):

break

dtReturn = dtNext + timeofday

return(dtReturn)

def ymd2epoch(year, month, day):

"""

@summary: Convert YMD info into a unix epoch value.

@param year: The year

@param month: The month

@param day: The day

@return epoch: number of seconds since epoch

"""

return(t.mktime(dt.date(year,month,day).timetuple()))

def epoch2date(ts):

"""

@summary Convert seconds since epoch into date

@param ts: Seconds since epoch

@return thedate: A date object

"""

tm = t.gmtime(ts)

return(dt.date(tm.tm\_year,tm.tm\_mon,tm.tm\_mday))

def \_trade\_dates(dt\_start, dt\_end, s\_period):

'''

@summary: Generate dates on which we need to trade

@param c\_strat: Strategy config class

@param dt\_start: Start date

@param dt\_end: End date

'''

ldt\_timestamps = getNYSEdays(dt\_start,

dt\_end, dt.timedelta(hours=16) )

# Use pandas reindex method instead

# Note, dates are index as well as values, we select based on index

# but return values since it is a numpy array of datetimes instead of

# pandas specific.

ts\_dates = pd.Series(index=ldt\_timestamps, data=ldt\_timestamps)

# These are the dates we want

if s\_period[:2] == 'BW':

# special case for biweekly

dr\_range = pd.DateRange(dt\_start, dt\_end,

timeRule=s\_period[1:])

dr\_range = np.asarray(dr\_range)

li\_even = np.array(list(range(len(dr\_range))))

dr\_range = dr\_range[li\_even[li\_even % 2 == 0]]

else:

dr\_range = pd.DateRange(dt\_start, dt\_end,

timeRule=s\_period)

dr\_range = np.asarray(dr\_range)

# Warning, we MUST copy the date range, if we modify it it will be returned

# in it's modified form the next time we use it.

dr\_range = np.copy(dr\_range)

dr\_range += pd.DateOffset(hours=16)

ts\_dates = ts\_dates.reindex( dr\_range, method='bfill' )

ldt\_dates = ts\_dates[ts\_dates.notnull()].values

#Make unique

sdt\_unique = set()

ldt\_dates = [x for x in ldt\_dates

if x not in sdt\_unique and not sdt\_unique.add(x)]

return ldt\_dates

"""

(c) 2013 Tsung-Han Yang

This source code is released under the Apache license.

blacksburg98@yahoo.com

Created on November 24, 2014

"""

import datetime as dt

from . import fpdateutil as du

import numpy as np

import pandas as pd

import finpy.data.dataaccess as da

def riskfree\_return(ldt\_timestamps, rf\_tick="$TNX"):

"""

Default is $TNX. Ten-year treasury rate

$FVX is another option. Five-Year treasury rate.

"""

c\_dataobj = da.DataAccess('Yahoo', cachestalltime=0)

ls\_keys = ['open', 'high', 'low', 'close', 'volume', 'actual\_close']

ldf\_data = c\_dataobj.get\_data(ldt\_timestamps, [rf\_tick], ls\_keys)

rf = (ldf\_data[0]['close']/100)/365

return rf

def pre\_timestamps(ldt\_timestamps, window):

"""

Return an list of timestamps.

Start roughly from ldt\_timestamps[0] - window.

End at ldt\_timestamps[0] - 1

"""

dt\_timeofday = dt.timedelta(hours=16)

days\_delta = dt.timedelta(days=(np.ceil(window\*7/5)+20))

dt\_start = ldt\_timestamps[0] - days\_delta

dt\_end = ldt\_timestamps[0] - dt.timedelta(days=1)

pre\_timestamps = du.getNYSEdays(dt\_start, dt\_end, dt\_timeofday)

return pre\_timestamps

def get\_max\_draw\_down(ts\_vals):

"""

@summary Returns the max draw down of the returns.

@param ts\_vals: 1d numpy array or fund list

@return Max draw down

"""

MDD = 0

DD = 0

peak = -99999

for value in ts\_vals:

if (value > peak):

peak = value

else:

DD = (peak - value) / peak

if (DD > MDD):

MDD = DD

return MDD

### intelligetn investor

import { httpsGET } from './httpsGET';

import { CompanyData } from '@/src/types/CompanyData';

const COMPANY\_FACTS\_URL = 'https://data.sec.gov/api/xbrl/companyfacts/CIK';

export async function fetchCompanyData(cik: string, email: string) {

const url = `${COMPANY\_FACTS\_URL}${"0".repeat(10 - cik.toString().length)}${cik}.json`;

const data = await httpsGET(url, 'data.sec.gov', email);

const parsedData: CompanyData = JSON.parse(data) ;

return parsedData;

}

import { App, normalizePath, TFile, Notice } from 'obsidian';

import { IntelligentInvestorSettings } from '@/src/settings/IntelligentInvestorSettings';

import { useState, useEffect } from 'react';

import { CompanyData, Share } from '@/src/types/CompanyData';

import getUnits from '@/src/components/getUnits';

// import { Line } from 'react-chartjs-2';

// import { Chart as ChartJS, CategoryScale, LinearScale, PointElement, LineElement, Title, Tooltip, Legend } from 'chart.js';

// ChartJS.register(CategoryScale, LinearScale, PointElement, LineElement, Title, Tooltip, Legend);

interface CompanyDataDisplayProps {

app: App;

settings: IntelligentInvestorSettings;

companyData: CompanyData;

}

export default function CompanyDataDisplay({ app, settings, companyData }: CompanyDataDisplayProps) {

const [formType, setFormType] = useState("10-Q");

const [companyMetric, setCompanyMetric] = useState("Assets");

const [years, setYears] = useState<number[]>([]);

const [startYear, setStartYear] = useState<number>(0);

const [endYear, setEndYear] = useState<number>(0);

const [displayData, setDisplayData] = useState<Share[]>([]);

const fiscalQuarterOrder = { Q1: 1, Q2: 2, Q3: 3, Q4: 4, FY: 5 } as any;

useEffect(() => {

const units = getUnits(companyData, companyMetric);

const years = Array.from(

new Set(

units

.map((report: any) => report.fy)

.sort((a: number, b: number) => a - b)

)

) as number[];

setYears(years);

if (startYear === 0) {

setStartYear(years[0]);

}

if (endYear === 0) {

setEndYear(years[years.length - 1]);

}

const data = units

.filter((datum: any) => {

return datum.form === formType && datum.fy >= startYear && datum.fy <= endYear

})

.slice()

.sort((a: any, b: any) => {

if (a.fy !== b.fy) {

return a.fy - b.fy;

}

if (fiscalQuarterOrder[a.fp] !== fiscalQuarterOrder[b.fp]) {

return fiscalQuarterOrder[a.fp] - fiscalQuarterOrder[b.fp];

}

const endDateComparison = new Date(a.end).getTime() - new Date(b.end).getTime();

if (endDateComparison !== 0) {

return endDateComparison;

}

return a.frame ? 1 : -1;

});

console.log("Displaying Data: ", data);

setDisplayData(data);

}, [formType, companyMetric, startYear, endYear]);

const exportCompanyData = async () => {

console.log("Exporting company data");

const csvContent = displayData.map((datum) => {

const row = [datum.fp, datum.fy, datum.filed, datum.end, datum.frame, datum.val];

return row.join(',');

}).join('\n');

const exportDirectory = settings.exportDirectory;

const filePath = normalizePath(`${exportDirectory}/${companyData.entityName} ${companyMetric}.csv`);

const existingFile = app.vault.getAbstractFileByPath(filePath);

if (existingFile instanceof TFile) {

await app.vault.modify(existingFile, csvContent);

} else {

await app.vault.create(filePath, csvContent);

}

new Notice(`Company data exported to ${filePath}`);

}

const handleSelectFormType = (e: React.ChangeEvent<HTMLSelectElement>) => {

setFormType(e.target.value);

}

const handleSelectMetric = (e: React.ChangeEvent<HTMLSelectElement>) => {

setCompanyMetric(e.target.value);

}

const handleSelectStartYear = (e: React.ChangeEvent<HTMLSelectElement>) => {

setStartYear(Number(e.target.value));

}

const handleSelectEndYear = (e: React.ChangeEvent<HTMLSelectElement>) => {

setEndYear(Number(e.target.value));

}

return (

<div>

<div className="company-data-control">

<label className="control-label">Company Metric</label>

<select

className="select-metric-control"

onChange={handleSelectMetric}

>

{Object.keys(companyData.facts["us-gaap"]).map((metric, index) => (

<option key={index} value={metric}>{metric}</option>

))}

</select>

</div>

<div className="company-data-controls">

<div className="company-data-control">

<label className="control-label">Select Form Type</label>

<select

className="select-control"

onChange={handleSelectFormType}

>

<option value="10-Q">10-Q</option>

<option value="10-K">10-K</option>

</select>

</div>

<div className="company-data-control">

<label className="control-label">

Start Year

</label>

<select

className="select-control"

value={startYear}

onChange={handleSelectStartYear}

>

{years.map((year, index) => (

<option key={index} value={year}>{year}</option>

))}

</select>

</div>

<div className="company-data-control">

<label className="control-label">

End Year

</label>

<select

className="select-control"

value={endYear}

onChange={handleSelectEndYear}

>

{years.map((year, index) => (

<option key={index} value={year}>{year}</option>

))}

</select>

</div>

<button className="export-button" onClick={exportCompanyData}>

Export

</button>

</div>

<div className="company-data-container">

<table className="company-data-table">

<thead>

<tr>

<th>FP</th>

<th>FY</th>

<th>Filed</th>

<th>End</th>

<th>Frame</th>

<th>Value</th>

</tr>

</thead>

<tbody>

{displayData.map((data: any, index: number) => (

<tr key={index} className={index % 2 === 0 ? "company-data-row-light" : "company-data-row-dark"}>

<td className="company-data-cell">{data.fp}</td>

<td className="company-data-cell">{data.fy}</td>

<td className="company-data-cell">{data.filed}</td>

<td className="company-data-cell">{data.end}</td>

<td className="company-data-cell">{data.frame}</td>

<td className="company-data-cell">{data.val}</td>

</tr>

))}

</tbody>

</table>

</div>

</div>

);

}

import {

CompanyData,

UsGaap,

DefinedContributionPlanEmployerMatchingContributionPercentUnits,

CommonStockParOrStatedValuePerShareUnits,

NumberOfOperatingSegmentsUnits,

EntityCommonStockSharesOutstandingUnits,

EntityPublicFloatUnits

} from '@/src/types/CompanyData';

const DefinedContributionPlanEmployerMatchingContributionPercent = [

"EffectiveIncomeTaxRateReconciliationAtFederalStatutoryIncomeTaxRate",

"RevenueRemainingPerformanceObligationPercentage",

"FinanceLeaseWeightedAverageDiscountRatePercent",

"LineOfCreditFacilityUnusedCapacityCommitmentFeePercentage",

"OperatingLeaseWeightedAverageDiscountRatePercent",

"ShareBasedCompensationArrangementByShareBasedPaymentAwardFairValueAssumptionsExpectedDividendRate",

"DefinedContributionPlanEmployerMatchingContributionPercent",

"RestructuringAndRelatedCostNumberOfPositionsEliminatedPeriodPercent"

]

const CommonStockParOrStatedValuePerShare = [

"ShareBasedCompensationArrangementByShareBasedPaymentAwardOptionsForfeituresAndExpirationsInPeriodWeightedAverageExercisePrice",

"ShareBasedCompensationArrangementByShareBasedPaymentAwardOptionsExercisableWeightedAverageExercisePrice",

"PreferredStockParOrStatedValuePerShare",

"EarningsPerShareBasicAndDiluted",

"ShareBasedCompensationArrangementsByShareBasedPaymentAwardOptionsExercisesInPeriodWeightedAverageExercisePrice",

"ShareBasedCompensationArrangementByShareBasedPaymentAwardOptionsOutstandingWeightedAverageExercisePrice","TemporaryEquityParOrStatedValuePerShare",

"ShareBasedCompensationArrangementByShareBasedPaymentAwardOptionsGrantsInPeriodWeightedAverageGrantDateFairValue",

"EarningsPerShareBasic",

"EarningsPerShareDiluted"

]

const NumberOf = [

"NumberOfReportingUnits",

"NumberOfOperatingSegments"

]

const EntityCommonStockSharesOutstanding = [

"AntidilutiveSecuritiesExcludedFromComputationOfEarningsPerShareAmount",

"CommonStockSharesAuthorized",

"CommonStockSharesIssued",

"CommonStockSharesOutstanding",

"ConversionOfStockSharesConverted1",

"PreferredStockSharesAuthorized",

"PreferredStockSharesIssued",

"ConversionOfStockSharesIssued1",

"PreferredStockSharesOutstanding",

"ShareBasedCompensationArrangementByShareBasedPaymentAwardOptionsExercisableNumber",

"ShareBasedCompensationArrangementByShareBasedPaymentAwardOptionsForfeituresAndExpirationsInPeriod",

"ShareBasedCompensationArrangementByShareBasedPaymentAwardOptionsOutstandingNumber",

"StockIssuedDuringPeriodSharesStockOptionsExercised",

"TemporaryEquitySharesAuthorized",

"TemporaryEquitySharesIssued",

"TemporaryEquitySharesOutstanding",

"WeightedAverageNumberOfShareOutstandingBasicAndDiluted",

"WeightedAverageNumberOfDilutedSharesOutstanding",

"ShareBasedCompensationArrangementByShareBasedPaymentAwardOptionsGrantsInPeriodGross",

"WeightedAverageNumberOfSharesOutstandingBasic"

]

export default function getUnits(companyData: CompanyData, metric: string): any {

if (DefinedContributionPlanEmployerMatchingContributionPercent.includes(metric)) {

const units = companyData.facts["us-gaap"][metric as keyof UsGaap].units as DefinedContributionPlanEmployerMatchingContributionPercentUnits;

return units.pure;

} else if (CommonStockParOrStatedValuePerShare.includes(metric)) {

const units = companyData.facts["us-gaap"][metric as keyof UsGaap].units as CommonStockParOrStatedValuePerShareUnits;

return units["USD/shares"];

} else if (NumberOf.includes(metric)) {

const units = companyData.facts["us-gaap"][metric as keyof UsGaap].units as NumberOfOperatingSegmentsUnits;

return units.Segment;

} else if (EntityCommonStockSharesOutstanding.includes(metric)) {

const units = companyData.facts["us-gaap"][metric as keyof UsGaap].units as EntityCommonStockSharesOutstandingUnits;

return units.shares;

} else {

const units = companyData.facts["us-gaap"][metric as keyof UsGaap].units as EntityPublicFloatUnits;

return units.USD;

}

}

### sec api

import os

import requests

import pickle

import pandas as pd

# ------------------------------------------------------------

def download\_company\_facts(cik: str):

url = f'https://data.sec.gov/api/xbrl/companyfacts/CIK{cik}.json'

headers = { 'User-Agent': os.environ.get('SEC\_GOV\_USER\_AGENT') }

response = requests.get(url, headers=headers)

response.raise\_for\_status()

return response

# ------------------------------------------------------------

# symbol = 'hood'

# cik = '0001783879'

def download\_and\_save\_company\_facts(symbol: str, cik: str):

response = download\_company\_facts(cik)

file = os.path.join('data', symbol, 'company\_facts\_response.pkl')

os.makedirs(os.path.dirname(file), exist\_ok=True)

with open(file, 'wb') as f:

pickle.dump(response.json(), f)

df\_all = pd.DataFrame()

taxonomies = response.json()['facts']

for taxonomy in taxonomies:

facts = list(response.json()['facts'][taxonomy].keys())

facts.sort()

for fact in facts:

units = response.json()['facts'][taxonomy][fact]['units'].keys()

for unit in units:

print(f'{taxonomy:<10}: {fact:<50}: {unit:<10}')

tmp = pd.DataFrame(response.json()['facts'][taxonomy][fact]['units'][unit])

tmp['taxonomy'] = taxonomy

tmp['fact'] = fact

tmp['unit'] = unit

df\_all = pd.concat([df\_all, tmp])

df\_all = df\_all[['filed', 'fy', 'fp', 'start', 'end', 'frame', 'form', 'taxonomy', 'fact', 'unit', 'accn', 'val']]

file = os.path.join('data', symbol, 'df\_all\_facts.pkl')

os.makedirs(os.path.dirname(file), exist\_ok=True)

df\_all.to\_pickle(file)

#### company valuations

import datetime

import time

import requests

import pandas as pd

from bs4 import BeautifulSoup

from dateutil.relativedelta import relativedelta

from pymongo.errors import DocumentTooLarge

import mongodb

AAPL\_CIK = "0000320193"

BABA\_CIK = "0001577552"

ATKR\_CIK = "0001666138"

META\_CIK = "0001326801"

\_8K\_URL = "https://www.sec.gov/Archives/edgar/data/320193/000114036123023909/ny20007635x4\_8k.htm"

def make\_edgar\_request(url):

"""

Make a request to EDGAR (Electronic Data Gathering, Analysis and Retrieval)

:param url:

:return: response

"""

headers = {

"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/113.0.0.0 Safari/537.36",

"Accept-Encoding": "gzip, deflate, br",

}

return requests.get(url, headers=headers)

def download\_cik\_ticker\_map():

"""

Get a mapping of cik (Central Index Key, id of company on edgar) and ticker on the exchange.

It saves this mapping in mongodb.

"""

CIK\_TICKER\_URL = "https://www.sec.gov/files/company\_tickers\_exchange.json"

response = make\_edgar\_request(CIK\_TICKER\_URL)

r = response.json()

r["\_id"] = "cik\_ticker"

mongodb.upsert\_document("cik\_ticker", r)

def get\_df\_cik\_ticker\_map():

"""

Create DataFrame from cik ticker document on mongodb.

:return: DataFrame

"""

try:

cik\_ticker = mongodb.get\_collection\_documents("cik\_ticker").next()

except StopIteration:

print("cik ticker document not found")

return

df = pd.DataFrame(cik\_ticker["data"], columns=cik\_ticker["fields"])

# add leading 0s to cik (always 10 digits)

df["cik"] = df.apply(lambda x: add\_trailing\_to\_cik(x["cik"]), axis=1)

return df

def company\_from\_cik(cik):

"""

Get company info from cik

:param cik: company id on EDGAR

:return: DataFrame row with company information (name, ticker, exchange)

"""

df = get\_df\_cik\_ticker\_map()

try:

return df[df["cik"] == cik].iloc[0]

except IndexError:

return None

def cik\_from\_ticker(ticker):

"""

Get company cik from ticker

:param ticker: company ticker

:return: cik (company id on EDGAR)

"""

df = get\_df\_cik\_ticker\_map()

try:

cik = df[df["ticker"] == ticker]["cik"].iloc[0]

except:

cik = -1

return cik

def download\_all\_cik\_submissions(cik):

"""

Get list of submissions for a single company.

Upsert this list on mongodb (each download contains all the submissions).

:param cik: cik of the company

:return:

"""

url = f"https://data.sec.gov/submissions/CIK{cik}.json"

response = make\_edgar\_request(url)

r = response.json()

r["\_id"] = cik

mongodb.upsert\_document("submissions", r)

def download\_submissions\_documents(cik, forms\_to\_download=("10-Q", "10-K", "8-K"), years=5):

"""

Download all documents for submissions forms 'forms\_to\_download' for the past 'max\_history' years.

Insert them on mongodb.

:param cik: company cik

:param forms\_to\_download: a tuple containing the form types to download

:param years: the max number of years to download

:return:

"""

try:

submissions = mongodb.get\_document("submissions", cik)

except StopIteration:

print(f"submissions file not found in mongodb for {cik}")

return

cik\_no\_trailing = submissions["cik"]

filings = submissions["filings"]["recent"]

for i in range(len(filings["filingDate"])):

filing\_date = filings['filingDate'][i]

difference\_in\_years = relativedelta(datetime.date.today(),

datetime.datetime.strptime(filing\_date, "%Y-%m-%d")).years

# as the document are ordered cronologically when we reach the max history we can return

if difference\_in\_years > years:

return

form\_type = filings['form'][i]

if form\_type not in forms\_to\_download:

continue

accession\_no\_symbols = filings["accessionNumber"][i].replace("-","")

primary\_document = filings["primaryDocument"][i]

url = f"https://www.sec.gov/Archives/edgar/data/{cik\_no\_trailing}/{accession\_no\_symbols}/{primary\_document}"

# if we already have the document, we don't download it again

if mongodb.check\_document\_exists("documents", url):

continue

print(f"{filing\_date} ({form\_type}): {url}")

download\_document(url, cik, form\_type, filing\_date)

# insert a quick sleep to avoid reaching edgar rate limit

time.sleep(0.2)

def download\_document(url, cik, form\_type, filing\_date, updated\_at=None):

"""

Download and insert submission document

:param url:

:param cik:

:param form\_type:

:param filing\_date:

:return:

"""

response = make\_edgar\_request(url)

r = response.text

doc = {"html": r, "cik": cik, "form\_type": form\_type, "filing\_date": filing\_date, "updated\_at": updated\_at, "\_id": url}

try:

mongodb.insert\_document("documents", doc)

except DocumentTooLarge:

# DocumenTooLarge is raised by mongodb when uploading files larger than 16MB

# To avoid this it is better to save this kind of files in a separate storate like S3 and retriving them when needed.

# Another options could be using mongofiles: https://www.mongodb.com/docs/database-tools/mongofiles/#mongodb-binary-bin.mongofiles

# for management of large files saved in mongo db.

print("Document too Large (over 16MB)", url)

def download\_financial\_data(cik):

"""

Download financial data for a company.

Upsert document on mongodb (each requests returns the entire history)

:param cik:

:return:

"""

url = f"https://data.sec.gov/api/xbrl/companyfacts/CIK{cik}.json"

response = make\_edgar\_request(url)

try:

r = response.json()

r["\_id"] = cik

r["url"] = url

mongodb.upsert\_document("financial\_data", r)

# ETFs, funds, trusts do not have financial information

except:

print(f"ERROR {cik} - {response} - {url}")

print(company\_from\_cik(cik))

def get\_filing\_from\_index(url):

"""

Get the document url from the filing index page.

This is a filing index page:

https://www.sec.gov/Archives/edgar/data/320193/000114036123023909/0001140361-23-023909-index.htm

The document url we want is the first url in the Document Format Files table.

:param url: filing index page url

:return:

"""

index\_page = make\_edgar\_request(url)

soup = BeautifulSoup(index\_page.text, "html.parser")

table = soup.find("table", {"class": "tableFile", "summary": "Document Format Files"})

return table.find("a")["href"]

def add\_trailing\_to\_cik(cik\_no\_trailing):

return "{:010d}".format(cik\_no\_trailing)

def get\_size\_in\_bytes(size\_string):

size = int(size\_string.split()[0])

unit = size\_string.split()[1].upper()

if unit == "MB":

return size \* 1024 \* 1024

elif unit == "KB":

return size \* 1024

else:

raise ValueError("Invalid size unit. Must be either MB or KB.")

def get\_latest\_filings(form\_type, start\_date):

"""

Get new filings (for all companies) since 'start\_date' (yyyy-mm-dd).

Insert new submission documents on mongodb.

Insert new financial data on mongodb.

Used to update submissions documents and financial data in our db.

:param form\_type: form that we want to request (you can pass multiple forms delimited by commas 10-K,10-Q,...

:param start\_date: date from where we want to retrieve new submissions

:return:

"""

start\_idx = 0

entries\_per\_request = 100

done = False

cik\_df = get\_df\_cik\_ticker\_map()

ciks = list(cik\_df["cik"].unique())

while not done:

url = f"https://www.sec.gov/cgi-bin/browse-edgar?action=getcurrent&type={form\_type}&datea={start\_date}&" \

f"start={start\_idx}&count={entries\_per\_request}&output=atom"

print(f"{url}")

response = make\_edgar\_request(url)

soup = BeautifulSoup(response.text, 'xml')

entries = soup.findAll("entry")

# If the response contains less entry than what we requested it means we are done

if len(entries) < entries\_per\_request:

done = True

for entry in entries:

index\_url = entry.find("link")["href"]

entry\_form\_type = entry.find("category")["term"]

entry\_updated\_at = entry.find("updated").text.split("T")[0]

entry\_summary = entry.find("summary").text.replace("<b>",";").replace("</b>","").replace("\n", "")

filed\_date = entry\_summary.split(';')[1].split(":")[1].strip()

size = get\_size\_in\_bytes(entry\_summary.split(';')[3].split(":")[1].strip())

start\_cik = index\_url.find('data/') + 5

end\_cik = index\_url.find('/', start\_cik)

cik = add\_trailing\_to\_cik(int(index\_url[start\_cik: end\_cik]))

if cik not in ciks:

print(f"{cik} not present in cik map - skip")

continue

if size > 16 \* 1024 \* 1024:

print(f"SKIP {cik} because of size {size}")

continue

url = get\_filing\_from\_index(index\_url)

url = f"https://www.sec.gov/{url.replace('/ix?doc=/','')}"

# if we already have the document on mongodb we can skip

if mongodb.check\_document\_exists("documents", url):

continue

download\_document(url, cik, entry\_form\_type, filed\_date, entry\_updated\_at)

# if entry\_form\_type in ["10-Q", "10-Q/A" "10-K", "10-K/A"]:

# download\_financial\_data(cik)

start\_idx += entries\_per\_request

if \_\_name\_\_ == '\_\_main\_\_':

apple\_tiker = "AAPL"

cik = cik\_from\_ticker(apple\_tiker)

download\_all\_cik\_submissions(cik)

# get\_latest\_filings("10-K", "2023-01-01")

# download\_cik\_ticker\_map()

# download\_all\_cik\_submissions("0001326801")

# download\_submissions\_documents("0001326801")

import requests

import urllib3

from bs4 import BeautifulSoup

currency\_country = {

"AUD": "Australia",

"BRL": "Brazil",

"CAD": "Canada",

"CHF": "Switzerland",

"CLP": "Chile",

"CNY": "China",

"COP": "Colombia",

"CZK": "Czech Republic",

"DKK": "Denmark",

"EGP": "Egypt",

"EUR": "Germany",

"GBP": "United Kingdom",

"HKD": "Hong Kong",

"HUF": "Hungary",

"IDR": "Indonesia",

"ILS": "Israel",

"INR": "India",

"ISK": "Iceland",

"JPY": "Japan",

"KRW": "South Korea",

"KZT": "Kazakhstan",

"MXN": "Mexico",

"MYR": "Malaysia",

"NGN": "Nigeria",

"NOK": "Norway",

"NZD": "New Zealand",

"PHP": "Philippines",

"PLN": "Poland",

"QAR": "Qatar",

"RUB": "Russia",

"SGD": "Singapore",

"THB": "Thailand",

"TRY": "Turkey",

"TWD": "Taiwan",

"USD": "United States",

"ZAR": "South Africa",

}

country\_url = {

"Australia": "https://www.investing.com/rates-bonds/australia-10-year-bond-yield",

"Austria": "https://www.investing.com/rates-bonds/austria-10-year-bond-yield",

"Belgium": "https://www.investing.com/rates-bonds/belguim-10-year-bond-yield",

"Brazil": "https://www.investing.com/rates-bonds/brazil-10-year-bond-yield",

"Canada": "https://www.investing.com/rates-bonds/canada-10-year-bond-yield",

"Chile": "https://www.investing.com/rates-bonds/chile-10-year-bond-yield",

"China": "https://www.investing.com/rates-bonds/china-10-year-bond-yield",

"Colombia": "https://www.investing.com/rates-bonds/colombia-10-year-bond-yield",

"Cyprus": "https://www.investing.com/rates-bonds/cyprus-10-year",

"Czech Republic": "https://www.investing.com/rates-bonds/czech-republic-10-year-bond-yield",

"Denmark": "https://www.investing.com/rates-bonds/denmark-10-year-bond-yield",

"Egypt": "https://www.investing.com/rates-bonds/egypt-10-year-bond-yield",

"Finland": "https://www.investing.com/rates-bonds/finland-10-year-bond-yield",

"France": "https://www.investing.com/rates-bonds/france-10-year-bond-yield",

"Germany": "https://www.investing.com/rates-bonds/germany-10-year-bond-yield",

"Greece": "https://www.investing.com/rates-bonds/greece-10-year-bond-yield",

"Hong Kong": "https://www.investing.com/rates-bonds/hong-kong-10-year-bond-yield",

"Hungary": "https://www.investing.com/rates-bonds/hungary-10-year-bond-yield",

"Iceland": "https://www.investing.com/rates-bonds/iceland-10-year-bond-yield",

"India": "https://www.investing.com/rates-bonds/india-10-year-bond-yield",

"Indonesia": "https://www.investing.com/rates-bonds/indonesia-10-year-bond-yield",

"Ireland": "https://www.investing.com/rates-bonds/ireland-10-year-bond-yield",

"Israel": "https://www.investing.com/rates-bonds/israel-10-year-bond-yield",

"Italy": "https://www.investing.com/rates-bonds/italy-10-year-bond-yield",

"Japan": "https://www.investing.com/rates-bonds/japan-10-year-bond-yield",

"Kazakhstan": "https://www.investing.com/rates-bonds/kazakhstan-10-year",

"Malaysia": "https://www.investing.com/rates-bonds/malaysia-10-year-bond-yield",

"Malta": "https://www.investing.com/rates-bonds/malta-10-year",

"Mauritius": "https://www.investing.com/rates-bonds/mauritius-10-year",

"Mexico": "https://www.investing.com/rates-bonds/mexico-10-year",

"Netherlands": "https://www.investing.com/rates-bonds/netherlands-10-year-bond-yield",

"New Zealand": "https://www.investing.com/rates-bonds/new-zealand-10-years-bond-yield",

"Nigeria": "https://www.investing.com/rates-bonds/nigeria-10-year",

"Norway": "https://www.investing.com/rates-bonds/norway-10-year-bond-yield",

"Philippines": "https://www.investing.com/rates-bonds/philippines-10-year-bond-yield",

"Poland": "https://www.investing.com/rates-bonds/poland-10-year-bond-yield",

"Portugal": "https://www.investing.com/rates-bonds/portugal-10-year-bond-yield",

"Qatar": "https://www.investing.com/rates-bonds/qatar-10-year-bond-yield",

"Russia": "https://www.investing.com/rates-bonds/russia-10-year-bond-yield",

"Singapore": "https://www.investing.com/rates-bonds/singapore-10-year-bond-yield",

"South Africa": "https://www.investing.com/rates-bonds/south-africa-10-year-bond-yield",

"South Korea": "https://www.investing.com/rates-bonds/south-korea-10-year-bond-yield",

"Spain": "https://www.investing.com/rates-bonds/spain-10-year-bond-yield",

"Switzerland": "https://www.investing.com/rates-bonds/switzerland-10-year-bond-yield",

"Taiwan": "https://www.investing.com/rates-bonds/taiwan-10-year-bond-yield",

"Thailand": "https://www.investing.com/rates-bonds/thailand-10-year-bond-yield",

"Turkey": "https://www.investing.com/rates-bonds/turkey-10-year-bond-yield",

"United Kingdom": "https://www.investing.com/rates-bonds/uk-10-year-bond-yield",

"United States": "https://www.investing.com/rates-bonds/u.s.-10-year-bond-yield",

"Vietnam": "https://www.investing.com/rates-bonds/vietnam-10-year-bond-yield"

}

def get\_10y\_bond\_yield(currency):

urllib3.disable\_warnings(urllib3.exceptions.InsecureRequestWarning)

if currency not in currency\_country:

return None, None

url = country\_url[currency\_country[currency]]

headers = {

'accept': 'text/plain, \*/\*; q=0.01',

'accept-encoding': 'gzip, deflate, utf-8',

'accept-language': 'en,it-IT;q=0.9,it;q=0.8,en-US;q=0.7',

'cache-control': 'no-cache',

'origin': 'https://www.investing.com',

'pragma': 'no-cache',

'sec-ch-ua': '".Not/A)Brand";v="99", "Google Chrome";v="103", "Chromium";v="103"',

'sec-ch-ua-mobile': '?0',

'sec-ch-ua-platform': '"Windows"',

'sec-fetch-dest': 'empty',

'sec-fetch-mode': 'cors',

'sec-fetch-site': 'same-origin',

'user-agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/103.0.0.0 Safari/537.36',

'upgrade-insecure-requests': '1',

}

bondyield = None

retries = 0

max\_retries = 3

while bondyield is None:

response = request\_with\_retries(url, headers=headers)

# with open("response.html", "w", encoding="utf-8") as f:

# f.write(response.text)

soup = BeautifulSoup(response.text, 'html.parser')

span = soup.select\_one('dd[data-test="prevClose"]')

try:

bondyield = round(float(span.text) / 100, 5)

except:

print("ERROR in getting riskfree", url)

retries += 1

if retries >= max\_retries:

break

return bondyield, currency\_country[currency]

def request\_with\_retries(url, headers=None):

urllib3.disable\_warnings(urllib3.exceptions.InsecureRequestWarning)

resp = None

max\_retry = 5

retry = 0

while resp is None and retry < max\_retry:

try:

if headers is not None:

resp = requests.get(url, verify=False, headers=headers)

else:

resp = requests.get(url, verify=False)

except:

print(f"{url} conn err - retry")

retry += 1

return resp

from configparser import ConfigParser

from pymongo import MongoClient

import os

DB\_NAME = 'company\_eval'

def get\_mongodb\_client():

"""

Get mongodb client

:return: mongodb client

"""

# Get credentials

parser = ConfigParser()

\_ = parser.read(os.path.join("credentials.cfg"))

username = parser.get("mongo\_db", "username")

password = parser.get("mongo\_db", "password")

# Set connection string

LOCAL\_CONNECTION = "mongodb://localhost:27017"

ATLAS\_CONNECTION = f"mongodb+srv://{username}:{password}@cluster0.3dxfmjo.mongodb.net/?" \

f"retryWrites=true&w=majority"

ATLAS\_OLD\_CONNECTION = f"mongodb://{username}:{password}@cluster0.3dxfmjo.mongodb.net:27017/?" \

f"retryWrites=true&w=majority&tls=true"

# print(ATLAS\_CONNECTION)

connection\_string = LOCAL\_CONNECTION

# Create a connection using MongoClient

client = MongoClient(connection\_string)

return client

def get\_collection(collection\_name):

db = get\_mongodb\_client()[DB\_NAME]

return db[collection\_name]

def get\_file\_size(file\_name):

file\_stats = os.stat(file\_name)

print(f'File Size in Bytes is {file\_stats.st\_size}')

return file\_stats.st\_size

def get\_dict\_size(data):

import sys

print("The size of the dictionary is {} bytes".format(sys.getsizeof(data)))

return sys.getsizeof(data)

def upsert\_document(collection\_name, data):

collection = get\_collection(collection\_name)

collection.replace\_one({"\_id":data["\_id"]}, data, upsert=True)

def insert\_document(collection\_name, data):

collection = get\_collection(collection\_name)

collection.insert\_one(data)

def get\_document(collection\_name, document\_id):

collection = get\_collection(collection\_name)

return collection.find({"\_id": document\_id}).next()

def check\_document\_exists(collection\_name, document\_id):

collection = get\_collection(collection\_name)

return collection.count\_documents({"\_id": document\_id}, limit=1) > 0

def get\_collection\_documents(collection\_name):

collection = get\_collection(collection\_name)

return collection.find({})

import time

import traceback

from datetime import datetime

import pandas as pd

import pymongo

from bs4 import BeautifulSoup

import mongodb

from edgar\_utils import company\_from\_cik, AAPL\_CIK, download\_all\_cik\_submissions, download\_submissions\_documents

from openai\_interface import summarize\_section

from postgresql import get\_df\_from\_table, country\_to\_region, area\_to\_repr\_country

def restructure\_parsed\_10k(doc):

"""

Look for and select only the sections specified in result dictionary.

:param doc: mongo document from "documents" collection

:return: a dictionary containing the parsed document sections titles and their text.

"""

result = {

"business": {"text":"", "links":[]}, # important

"risk": {"text":"", "links":[]}, # important

"unresolved": {"text":"", "links":[]},

"property": {"text":"", "links":[]},

# "MD&A": {"text":"", "links":[]}, # important

"legal": {"text":"", "links":[]},

"foreign": {"text":"", "links":[]},

# "notes": {"text":"", "links":[]},

"other": {"text":"", "links":[]}

}

for s in doc["sections"]:

found = None

if ("business" in s.lower() or "overview" in s.lower() or "company" in s.lower() or "general" in s.lower() or "outlook" in s.lower())\

and not "combination" in s.lower():

found = "business"

elif "propert" in s.lower() and not "plant" in s.lower() and not "business" in s.lower():

found = "property"

elif "foreign" in s.lower() and "jurisdiction" in s.lower():

found = "foreign"

elif "legal" in s.lower() and "proceeding" in s.lower():

found = "legal"

# elif "management" in s.lower() and "discussion" in s.lower():

# found = "MD&A"

# elif "supplementa" in s.lower() or ("note" in s.lower() and "statement" not in s.lower()):

# found = "notes"

elif "information" in s.lower() and "other" in s.lower():

found = "other"

elif "unresolved" in s.lower():

found = "unresolved"

elif "risk" in s.lower():

found = "risk"

if found is not None:

result[found]["text"] += doc["sections"][s]["text"]

result[found]["links"].append({

"title": s,

"link": doc["sections"][s]["link"] if "link" in doc["sections"][s] else None

})

return result

def restructure\_parsed\_10q(doc):

result = {

"risk": {"text":"", "links":[]}, # important

"MD&A": {"text":"", "links":[]}, # important

"legal": {"text":"", "links":[]},

"other": {"text":"", "links":[]},

"equity": {"text":"", "links":[]},

"defaults": {"text":"", "links":[]},

}

for s in doc["sections"]:

found = None

if "legal" in s.lower() and "proceeding" in s.lower():

found = "legal"

elif "management" in s.lower() and "discussion" in s.lower():

found = "MD&A"

elif "information" in s.lower() and "other" in s.lower():

found = "other"

elif "risk" in s.lower():

found = "risk"

elif "sales" in s.lower() and "equity" in s.lower():

found = "equity"

elif "default" in s.lower():

found = "defaults"

if found is not None:

result[found]["text"] += doc["sections"][s]["text"]

result[found]["links"].append({

"title": s,

"link": doc["sections"][s]["link"] if "link" in doc["sections"][s] else None

})

return result

def restructure\_parsed\_8k(doc):

result = {}

for s in doc["sections"]:

if "financial statements and exhibits" in s.lower():

continue

result[s] = doc["sections"][s]

return result

def sections\_summary(doc, verbose=False):

"""

Summarize all sections of a document using openAI API.

Upsert summary on mongodb (overwrite previous one, in case we make changes to openai\_interface)

This method is configured to use gpt-3.5-turbo. At the moment this model has two different version,

a version with 4k token and a version with 16k tokens. That are used with based on the length of a sections.

:param doc: a parsed\_document from mongodb

:param verbose: passed to langchain verbose

:return:

"""

company = company\_from\_cik(doc["cik"])

result = {"\_id": doc["\_id"],

"name": company["name"],

"ticker": company["ticker"],

"form\_type": doc["form\_type"],

"filing\_date": doc["filing\_date"]}

total\_cost = 0

total\_start\_time = time.time()

if "10-K" in doc["form\_type"]:

new\_doc = restructure\_parsed\_10k(doc)

elif doc["form\_type"] == "10-Q":

new\_doc = restructure\_parsed\_10q(doc)

elif doc["form\_type"] == "8-K":

new\_doc = restructure\_parsed\_8k(doc)

else:

print(f"form\_type {doc['form\_type']} is not yet implemented")

return

for section\_title, section in new\_doc.items():

section\_links = section["links"] if "links" in section else None

section\_text = section["text"]

start\_time = time.time()

if len(section\_text) < 250:

continue

if section\_title in ["business", "risk", "MD&A"]:

chain\_type = "refine"

if len(section\_text) > 25000:

model = "gpt-3.5-turbo-16k"

else:

model = "gpt-3.5-turbo"

else:

if len(section\_text) < 25000:

chain\_type = "refine"

model = "gpt-3.5-turbo"

elif len(section\_text) < 50000:

chain\_type = "map\_reduce"

model = "gpt-3.5-turbo"

else:

chain\_type = "map\_reduce"

model = "gpt-3.5-turbo-16k"

original\_len = len(section\_text)

# get summary from openAI model

print(f"{section\_title} original\_len: {original\_len} use {model} w/ chain {chain\_type}")

summary, cost = summarize\_section(section\_text, model, chain\_type, verbose)

result[section\_title] = {"summary":summary, "links": section\_links}

summary\_len = len(''.join(summary))

reduction = 100 - round(summary\_len / original\_len \* 100, 2)

total\_cost += cost

duration = round(time.time() - start\_time, 1)

print(f"{section\_title} original\_len: {original\_len} summary\_len: {summary\_len} reduction: {reduction}% "

f"cost: {cost}$ duration:{duration}s used {model} w/ chain {chain\_type}")

mongodb.upsert\_document("items\_summary", result)

total\_duration = round(time.time() - total\_start\_time, 1)

print(f"\nTotal Cost: {total\_cost}$, Total duration: {total\_duration}s")

def extract\_segments(doc):

"""

Extract segments information (industry, geographical) from document

:param url: url of the document, used as id on mongodb

:return: list of dictionaries {"date": date, "segment":{"axis":"member", ...}, "value": number, "measure": "measure/metric"}

"""

# doc = mongodb.get\_document("documents", url)

page = doc["html"]

soap = BeautifulSoup(page, features="html.parser")

ix\_resources = soap.find("ix:resources")

if ix\_resources is None:

return

contexts = ix\_resources.findAll("xbrli:context")

axis = [

"srt:ProductOrServiceAxis",

"us-gaap:StatementBusinessSegmentsAxis",

"srt:ConsolidationItemsAxis",

"srt:StatementGeographicalAxis",

]

result = []

for c in contexts:

context\_id = c["id"]

s = c.find("xbrli:segment")

if s is not None:

members = s.find\_all("xbrldi:explicitmember")

if len(members) == 0:

continue

include = True

for m in members:

if m["dimension"] not in axis:

include = False

break

if not include:

continue

try:

period = c.find("xbrli:enddate").text

except:

period = c.find("xbrli:instant").text

period = datetime.strptime(period, "%Y-%m-%d").date()

element = soap.find("ix:nonfraction", attrs={"contextref": context\_id})

if element is None or "name" not in element.attrs:

continue

try:

value = float(element.text.replace(",",""))

except:

continue

segment = {}

for m in members:

segment[m["dimension"]] = m.text

result.append({

"date": period,

"segment": segment,

"value": value,

"measure": element["name"]

})

return result

def map\_geographic\_area(string):

if "other" in string and ("region" in string or "countr" in string or "continent" in string):

return "Global"

elif "foreign" in string:

return "Global"

elif "europe" in string:

return "Western Europe"

elif "asia" in string:

return "Asia"

elif "emea" in string:

return "EMEA" # 70% western europe, 15% middle east, 15% africa

elif "apac" in string:

return "APAC" # 90% asia, 10% australia

elif "lacc" in string:

return "LACC" # 50% central & south america, 40% canada, 10% caribbean

elif "centralandsouthamerica" in string or "southamerica" in string or "americas" in string:

return "Central and South America"

elif "africa" in string:

return "Africa"

elif "middleeast" in string:

return "Middle East"

elif "northamerica" in string:

return "North America"

def geography\_distribution(segments, ticker):

df = pd.DataFrame(segments)

if df.empty:

return df

df["segment"] = df["segment"].astype(str)

# filter by geography segments

df = df[(df["segment"].str.contains('srt:StatementGeographicalAxis'))&

~(df["segment"].str.contains("srt:ProductOrServiceAxis"))&

~(df["segment"].str.contains("us-gaap:StatementBusinessSegmentsAxis"))]

# print(df.to\_markdown())

# filter by measure

measures = list(df["measure"].unique())

selected\_measure = None

for m in measures:

if "revenue" in m.lower() and ticker in m.lower():

selected\_measure = m

break

if selected\_measure is None:

for m in [

"Revenues",

"RevenueFromContractWithCustomerExcludingAssessedTax",

"RevenueFromContractWithCustomerIncludingAssessedTax",

"SalesRevenueNet",

"OperatingIncomeLoss",

"IncomeLossFromContinuingOperationsBeforeInterestExpenseInterestIncomeIncomeTaxesExtraordinaryItemsNoncontrollingInterestsNet",

"IncomeLossFromContinuingOperationsBeforeIncomeTaxesMinorityInterestAndIncomeLossFromEquityMethodInvestments",

"IncomeLossFromContinuingOperationsBeforeIncomeTaxesExtraordinaryItemsNoncontrollingInterest",

"IncomeLossFromContinuingOperationsBeforeIncomeTaxesForeign",

"IncomeLossFromContinuingOperationsBeforeIncomeTaxesDomestic",

"NetIncomeLoss",

"NetIncomeLossAvailableToCommonStockholdersBasic",

"NetIncomeLossAvailableToCommonStockholdersDiluted",

"ComprehensiveIncomeNetOfTax",

"IncomeLossFromContinuingOperations",

"ProfitLoss",

"IncomeLossFromContinuingOperationsIncludingPortionAttributableToNoncontrollingInterest",

"IncomeLossFromSubsidiariesNetOfTax"

]:

if f"us-gaap:{m}" in measures:

selected\_measure = f"us-gaap:{m}"

break

df = df[df["measure"] == selected\_measure]

df = df[df.groupby(["segment","measure"])['date'].transform('max') == df['date']]\

.drop(["date","measure"], axis=1)

# print(df.to\_markdown())

# if only 'srt:StatementGeographicalAxis'

df["segment"] = df["segment"].apply(lambda x:

x[x.find("'srt:StatementGeographicalAxis':")+len("'srt:StatementGeographicalAxis':"):]

.split("}")[0].split(",")[0].split(":")[1].split("'")[0])

# MAP SEGMENTS

# 1st try and match countries

country\_stats = get\_df\_from\_table("damodaran\_country\_stats", most\_recent=True)[["country","alpha\_2\_code"]]

df = pd.merge(df, country\_stats, left\_on="segment", right\_on="alpha\_2\_code", how="left").drop("alpha\_2\_code", axis=1)

# 2st try and map regions

df["area"] = df["segment"].apply(lambda x: map\_geographic\_area(x.lower()))

# manage the rest

df = df[~(df["country"].isna())|~(df["area"].isna())]

df["value"] /= df["value"].sum()

df["country\_area"] = df["country"].fillna(df["area"])

aggregate\_areas\_df = pd.DataFrame([

{"country\_area": "EMEA", "part\_area": "Western Europe", "area\_percent": 0.7},

{"country\_area": "EMEA", "part\_area": "Middle East", "area\_percent": 0.15},

{"country\_area": "EMEA", "part\_area": "Africa", "area\_percent": 0.15},

{"country\_area": "APAC", "part\_area": "Asia", "area\_percent": 0.9},

{"country\_area": "APAC", "part\_area": "Australia & New Zealand", "area\_percent": 0.1},

{"country\_area": "LACC", "part\_area": "Central and South America", "area\_percent": 0.5},

{"country\_area": "LACC", "part\_area": "Canada", "area\_percent": 0.4},

{"country\_area": "LACC", "part\_area": "Caribbean", "area\_percent": 0.1},

])

# print(df.to\_markdown())

df = pd.merge(df, aggregate\_areas\_df, how="left", left\_on="country\_area", right\_on="country\_area")

df["part\_area"] = df["part\_area"].fillna(df["country\_area"])

df["area\_percent"] = df["area\_percent"].fillna(1)

df = df.drop("country\_area", axis=1)

df = df.rename(columns={"part\_area":"country\_area"})

df["value"] = df["value"] \* df["area\_percent"]

df["region"] = df["country\_area"].apply(lambda x: country\_to\_region[x] if x in country\_to\_region else "Global")

df["country\_representative"] = df["country\_area"].apply(lambda x: area\_to\_repr\_country[x] if x in area\_to\_repr\_country else None)

df["country"] = df["country"].fillna(df["country\_representative"])

# print(df.to\_markdown())

return df.drop(["segment","country\_representative","area", "area\_percent"], axis=1)

def try\_geo\_segments():

# url = "https://www.sec.gov/Archives/edgar/data/1666138/000166613822000128/atkr-20220930.htm" # ATKR 10-k

# url = "https://www.sec.gov/Archives/edgar/data/320193/000032019322000108/aapl-20220924.htm" # AAPL 10-K

# url = "https://www.sec.gov/Archives/edgar/data/1800/000162828023004026/abt-20221231.htm" # ABT 10-K

# url = "https://www.sec.gov/Archives/edgar/data/2098/000156459023003422/acu-10k\_20221231.htm" # ACU 10-K

# url = "https://www.sec.gov/Archives/edgar/data/4447/000162828023005059/hes-20221231.htm" # HES 10-K

docs = mongodb.get\_collection\_documents("documents")

for doc in docs:

if doc["form\_type"] != "10-K":

continue

print(doc["\_id"])

ticker = doc["\_id"].split("/")[-1].split("-")[0]

segments = extract\_segments(doc)

geography\_distribution(segments, ticker)

# segments = extract\_segments(url)

# geography\_distribution(segments, "hes")

def get\_last\_document(cik, form\_type):

download\_all\_cik\_submissions(cik)

download\_submissions\_documents(cik, forms\_to\_download=("10-K", "10-Q", "8-K",), years=1)

collection = mongodb.get\_collection("documents")

docs = collection.find({"cik": cik, "form\_type": form\_type})

last\_doc = None

last\_date = None

for doc in docs:

filing\_date = datetime.strptime(doc["filing\_date"], "%Y-%m-%d")

if last\_date is None or filing\_date > last\_date:

last\_date = filing\_date

last\_doc = doc

return last\_doc

def get\_recent\_docs(cik, filing\_date):

collection = mongodb.get\_collection("documents")

docs = collection.find({"cik": cik, "filing\_date": {"$gte":filing\_date}})

# sort by date asc

docs = docs.sort("filing\_date", pymongo.ASCENDING)

return docs

if \_\_name\_\_ == '\_\_main\_\_':

# doc = get\_last\_document(AAPL\_CIK, "10-K")

# print(doc["\_id"])

try\_geo\_segments()

import datetime

import traceback

import pandas as pd

import mongodb

from edgar\_utils import company\_from\_cik, cik\_from\_ticker, download\_financial\_data

from postgresql import get\_df\_from\_table, get\_generic\_info

from qualitative\_analysis import get\_last\_document, extract\_segments, geography\_distribution, get\_recent\_docs, \

sections\_summary

from utils import parse\_document, find\_auditor

from valuation\_helper import convert\_currencies, get\_target\_info, get\_normalized\_info, get\_dividends\_info, \

get\_final\_info, calculate\_liquidation\_value, dividends\_valuation, fcff\_valuation, get\_status, summary\_valuation, \

r\_and\_d\_amortization, get\_growth\_ttm, capitalize\_rd, debtize\_op\_leases, get\_roe\_roc, get\_spread\_from\_dscr, \

company\_complexity, company\_share\_diluition, get\_company\_type, currency\_bond\_yield, get\_industry\_data

from yahoo\_finance import get\_current\_price\_from\_yahoo

EARNINGS\_TTM = "EARNINGS\_TTM"

EARNINGS\_NORM = "EARNINGS\_NORM"

GROWTH\_FIXED = "GROWTH\_FIXED"

GROWTH\_TTM = "GROWTH\_TTM"

GROWTH\_NORM = "GROWTH\_NORM"

STATUS\_OK = "OK"

STATUS\_NI = "NI"

STATUS\_KO = "KO"

def build\_financial\_df(doc, measure, unit="USD", tax="us-gaap"):

"""

Build a DataFrame from a company financial document (containing all history and all measures).

DataFrame is built on a specific subsection of the document, identified with taxonomy, measure, unit

:param doc: company financial document

:param measure: measure we are interest in

:param unit: unit of measure (usually is a single one for each measure)

:param tax: taxonomy

:return: DataFrame

"""

try:

data = doc["facts"][tax][measure]["units"][unit]

except:

return None

df = pd.DataFrame(data)

df["val"] = pd.to\_numeric(df["val"])

# for income statement or cashflow statement measures we have a start and end date to represent a period.

# for example Revenues, we need to know the period (start-end) in which they have been generated.

# for balance sheet measures we have only end date as they are snapshot in time.

# for example Cash, we just know the amount in a certain date, there is no start-date period concept.

try:

if "start" in df.columns:

df["start"] = pd.to\_datetime(df["start"])

df["end"] = pd.to\_datetime(df["end"])

df["filed"] = pd.to\_datetime(df["filed"])

except:

return None

# print(measure, unit, tax)

# print(df)

try:

df = df[~df.frame.isna()]

except:

df = df[0:0]

return df

def get\_ttm\_from\_df(df):

"""

Compute TTM (trailing twelve months) value from a DataFrame containing quarterly and annual values.

:param df: DataFrame containing quarterly and annual values

:return: ttm value, year of last annual value in DataFrame

"""

# create a copy as we are going to edit and filter it

ttm\_df = df.copy()

# Get last annual value

try:

# Keep only annual and quarterly periods

ttm\_df["period"] = (ttm\_df["end"] - ttm\_df["start"]).dt.days

ttm\_df = ttm\_df[~(ttm\_df.frame.str.contains("Q")) | ((ttm\_df.frame.str.contains("Q")) & (ttm\_df.period < 100))]

last\_yearly\_row = ttm\_df[ttm\_df.period > 100].iloc[-1]

except:

return None, None

# Get quarterly values AFTER the annual value

post\_quarterly\_rows = ttm\_df[ttm\_df.index > last\_yearly\_row.name]

# Get corresponding quarterly values BEFORE the annual value

pre\_frames = list(post\_quarterly\_rows.frame)

pre\_frames = [x[:2] + str(int(x[2:6]) - 1) + x[6:] for x in pre\_frames]

pre\_quarterly\_rows = ttm\_df[ttm\_df.frame.isin(pre\_frames)]

# TTM = annual value + quarterly values after - corresponding quarterly values before

ttm = last\_yearly\_row.val + post\_quarterly\_rows.val.sum() - pre\_quarterly\_rows.val.sum()

return ttm, last\_yearly\_row.end

def get\_most\_recent\_value\_from\_df(df):

"""

Get most recent value and date in DataFrame (last row)

:param df: DataFrame containing quarterly and annual values

:return: most recent value and date in DataFrame

"""

return {"date":df.iloc[-1]["end"], "value":df.iloc[-1]["val"]}

def get\_last\_annual\_report\_date\_and\_fy(df):

if df is None:

return None, None

year\_df = df[~df.frame.str.contains("Q")]

dates = list((year\_df.frame.str.replace("CY", "")).astype(int))

last\_annual\_report\_date = year\_df.iloc[-1].end if len(year\_df) > 0 else None

last\_annual\_report\_fy = dates[-1] if len(dates) > 0 else None

return last\_annual\_report\_date, last\_annual\_report\_fy

def get\_quarter\_of\_annual\_report(df, last\_annual\_report\_date, last\_annual\_report\_fy):

if df is None:

return None, None

last\_annual\_report\_row = df[df.end == last\_annual\_report\_date]

if last\_annual\_report\_row.empty:

return None, None

# frame is a string CYXXXXQXI, we want the X between Q and I

try:

quarter\_of\_annual\_report = last\_annual\_report\_row.iloc[0]["frame"][7]

except:

print(last\_annual\_report\_row)

return None, None

year\_bs = int(last\_annual\_report\_row.frame.iloc[0][2:6])

years\_diff = year\_bs - last\_annual\_report\_fy

return quarter\_of\_annual\_report, years\_diff

def get\_yearly\_values\_from\_df(df, instant=False, quarter\_of\_annual\_report=None, years\_diff=0):

"""

Get yearly data from DataFrame

:param df: DataFrame containing quarterly and annual values

:param instant: bool that indicates if the measure is instantaneous (snapshot), if True it means the measure is a

balance sheet measure, otherwise it's an income statement/cashflow statement measure (period instead of snapshot)

:param quarter\_of\_annual\_report: in case of instant measure, we also need the quarter when the annual report is

released

:param years\_diff: used when the fiscal year ends in a different solar year

:return: dict {

"dates": [date1, date2, ..., dateN],

"values": [val1, val2, ..., valN],

"last\_annual\_report\_date": date

}

"""

# create a copy as we are going to edit and filter it

year\_df = df.copy()

# income statement / cashflow statement

if not instant:

# get only annual frames

year\_df = year\_df[~year\_df.frame.str.contains("Q")]

dates = list((year\_df.frame.str.replace("CY", "")).astype(int))

return {"dates": dates,

"values": list(year\_df.val)}

# balance sheet

else:

# keep only only rows with quarters of annual reports

year\_df = year\_df[year\_df.frame.str.contains(f"Q{quarter\_of\_annual\_report}I")]

year\_df["frame"] = year\_df.frame.str.replace("CY", "").str.replace(f"Q{quarter\_of\_annual\_report}I","").astype(int) - years\_diff

return {"dates": list(year\_df.frame),

"values": list(year\_df.val)}

def get\_values\_from\_measures(doc, measures, get\_ttm=True, get\_most\_recent=True, get\_yearly=True, instant=False,

quarter\_of\_annual\_report=None, years\_diff=0, debug=False, unit="USD", tax="us-gaap"):

"""

Retrieve requested financial values from company financial document (containing all history and all measures).

Measures are interpreted in a hierarchical way, meaning that if we have a value for 2020 for the first measure,

and a value for 2020 for the second measure, we are going to keep the first.

This is done in order to account for different possible measures that represent the same metric but could be present

in a company but not in another. The hierarchy is useful in case a company has more than one measure and we need

to choose which one to keep

:param doc: company financial document

:param measures: measures we are interest in (in order of "importance")

:param get\_ttm: bool, whether to compute ttm value

:param get\_most\_recent: bool, whether to compute most recent value

:param get\_yearly: bool, whether to compute yearly values

:param instant: bool, indicates if the measures are instantaneous (snapshot, balance sheet) or not (period,

income statement / cashflow statement)

:param quarter\_of\_annual\_report: quarter of annual report, used for instant measures

:param years\_diff: difference between fiscal year and solar year, used for instant measures

:param debug: bool, print debug statements

:param unit: unit for build\_financial\_df

:param tax: taxonomy for build\_financial\_df

:return: most recent value, ttm value, yearly values (0 or empty if not requested)

"""

ttm = 0

ttm\_year = None

most\_recent = 0

most\_recent\_date = None

yearly = {"dates": [], "values": []}

for m in measures:

# Build the DataFrame

df = build\_financial\_df(doc, m, unit, tax)

# The df is None if the company does not have the measure m in its financial data

if df is None or df.empty:

continue

if get\_ttm:

# Get TTM

ttm\_value\_tmp, ttm\_year\_tmp = get\_ttm\_from\_df(df)

if ttm\_value\_tmp is not None:

# We override ttm if we have a more recent value

if ttm\_year is None or ttm\_year\_tmp > ttm\_year:

ttm = ttm\_value\_tmp

ttm\_year = ttm\_year\_tmp

if debug:

print(m, ttm\_year\_tmp, ttm\_value\_tmp)

if get\_most\_recent:

# Get most recent value

most\_recent\_tmp = get\_most\_recent\_value\_from\_df(df)

if most\_recent\_tmp["value"] is not None:

# We override most\_recent\_value if we have a more recent value

if most\_recent\_date is None or most\_recent\_tmp["date"] > most\_recent\_date:

most\_recent\_date = most\_recent\_tmp["date"]

most\_recent = most\_recent\_tmp["value"]

if debug:

print(m, most\_recent\_tmp["date"], most\_recent\_tmp["value"])

if get\_yearly:

# Get yearly values

yearly\_tmp = get\_yearly\_values\_from\_df(df, instant, quarter\_of\_annual\_report, years\_diff)

if yearly\_tmp is not None:

# for each date

for i, d in enumerate(yearly\_tmp["dates"]):

# if we don't have it already (hierarchical), we add the values

if d not in yearly["dates"]:

yearly["dates"].append(d)

yearly["values"].append(yearly\_tmp["values"][i])

if debug:

print(m, yearly\_tmp)

# sort dates and values from the least recent to the most recent

sort = sorted(zip(yearly["dates"], yearly["values"]))

yearly["dates"] = [x for x, \_ in sort]

yearly["values"] = [x for \_, x in sort]

if debug:

print("ttm", ttm)

print("most recent", most\_recent)

print("yearly", yearly)

return {"date":most\_recent\_date, "value":most\_recent}, {"date":ttm\_year, "value":ttm}, yearly

def merge\_subsets\_yearly(superset, subsets, must\_include=None):

"""

Sum multiple measures into a single one. Superset is the measure that should already represent the sum. Subsets are

its components.

This method is used when we don't have the aggregated measure or we don't have it for all the years where we have

the disaggregated measures.

For example we have Total Assets for 2020,2021,2022 + Current Assets and Non-Current Assets from 2019 to 2022.

In this case we can build Total Assets also for 2019.

:param superset: aggregated measure

:param subsets: disaggregated measures to be summed

:param must\_include: we can pass a tuple in order to consider a summed value iff we all the measures in subsets

with indexes included in 'must\_include' have values. In the example above for example we can say to add the 2019

value for Total Assets iff we have Current Assets for 2019. If we have only Non-Current assets we will not add the

2019 value for Total Assets.

:return:

"""

to\_add = {"dates":[],"values":[]}

# if no must\_include

if must\_include is None:

# for each subset

for s in subsets:

# for each date in the subset

for i, d in enumerate(s["dates"]):

# if that date is not in superset

if d not in superset["dates"]:

# if it's the first subset with that date, append the value

if d not in to\_add["dates"]:

to\_add["dates"].append(d)

to\_add["values"].append(s["values"][i])

# else add the value to the existing one

else:

idx = to\_add["dates"].index(d)

to\_add["values"][idx] += s["values"][i]

# if must\_include

else:

if not isinstance(must\_include, tuple):

raise Exception("must\_include must be a tuple")

# get dates for the first must\_include (all others in must\_include must have the same dates for the date

# to be included)

tmp\_dates = subsets[must\_include[0]]["dates"]

remove\_dates = []

# for each date

for d in tmp\_dates:

# for each index in must\_include

for m in must\_include:

# if the subset does not have the date we remove it

s = subsets[m]

if d not in s["dates"]:

remove\_dates.append(d)

# keep only the dates where we have values for every must\_include subset

must\_include\_dates = [x for x in tmp\_dates if x not in remove\_dates and x not in superset["dates"]]

# if none return

if len(must\_include\_dates) == 0:

return

# set to 0 the values for each date

for m in must\_include\_dates:

to\_add["dates"].append(m)

to\_add["values"].append(0)

# for each subset, add the value for the dates

for s in subsets:

for i, d in enumerate(s["dates"]):

if d in to\_add["dates"]:

idx = to\_add["dates"].index(d)

to\_add["values"][idx] += s["values"][i]

for i, d in enumerate(to\_add["dates"]):

superset["dates"].append(d)

superset["values"].append(to\_add["values"][i])

# sort date and values in superset

sort = sorted(zip(superset["dates"], superset["values"]))

superset["dates"] = [x for x, \_ in sort]

superset["values"] = [x for \_, x in sort]

def merge\_subsets\_most\_recent(superset, subsets):

"""

Sum multiple most recent (or ttm) measures into a single one. Superset is the measure that should already represent the sum. Subsets are

its components.

This method is used when we don't have the aggregated measure or we don't have it for all the years where we have

the disaggregated measures.

For example we have Total Assets for 2020,2021,2022 + Current Assets and Non-Current Assets from 2019 to 2022.

In this case we can build Total Assets also for 2019.

:param superset: aggregated measure

:param subsets: disaggregated measures to be summed

:return:

"""

replace = False

for s in subsets:

if superset["date"] is None or (s["date"] is not None and s["date"] > superset["date"]):

replace = True

break

if replace:

dates = [x["date"] for x in subsets if x["date"] is not None]

# we are here if neither the superset nor the subsets have any value

if len(dates) == 0:

return

d = max(dates)

superset["date"] = d

superset["value"] = 0

for s in subsets:

if s["date"] == d:

superset["value"] += s["value"]

def extract\_shares(doc, quarter\_of\_annual\_report, years\_diff):

"""

Extract number of shares from company financial document

:param doc: company financial document

:return: number of common shares outstanding (most recent and annual)

"""

df = build\_financial\_df(doc, "EntityCommonStockSharesOutstanding", unit="shares", tax="dei")

debug = False

if debug:

print(df.to\_markdown())

try:

most\_recent\_shares = get\_most\_recent\_value\_from\_df(df)

except:

most\_recent\_shares = {"date":None, "value":0}

measures = ["CommonStockSharesOutstanding"]

mr\_common\_shares, \_, yearly\_common\_shares = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report,

years\_diff=years\_diff, get\_ttm=False,

get\_most\_recent=True, debug=debug, unit="shares")

measures = ["WeightedAverageNumberOfSharesOutstandingBasic"]

mr\_average\_shares, \_, yearly\_average\_shares = get\_values\_from\_measures(

doc, measures, instant=False, quarter\_of\_annual\_report=quarter\_of\_annual\_report,

years\_diff=years\_diff, get\_ttm=False,

get\_most\_recent=True, debug=debug, unit="shares")

merge\_subsets\_most\_recent(most\_recent\_shares, [mr\_common\_shares])

merge\_subsets\_most\_recent(most\_recent\_shares, [mr\_average\_shares])

try:

yearly\_shares = get\_yearly\_values\_from\_df(df, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report,

years\_diff=years\_diff)

merge\_subsets\_yearly(yearly\_common\_shares, [yearly\_average\_shares])

merge\_subsets\_yearly(yearly\_shares, [yearly\_common\_shares])

except:

merge\_subsets\_yearly(yearly\_common\_shares, [yearly\_average\_shares])

yearly\_shares = yearly\_common\_shares

# in some filings the company report shares with a wrong unit of measure (million shares instead of thousand shares)

try:

max\_num\_shares = max(yearly\_shares["values"])

except:

raise NoSharesException()

yearly\_shares["values"] = [x \* 1000 if x / max\_num\_shares < 0.01 else x for x in yearly\_shares["values"]]

if most\_recent\_shares["value"] / max\_num\_shares < 0.01:

most\_recent\_shares["value"] \*= 1000

return {

"mr\_shares": most\_recent\_shares,

"shares": yearly\_shares,

}

class NoSharesException(Exception):

pass

def extract\_income\_statement(doc):

"""

Extract income statement measures from company financial document.

Measures include:

- revenue

- R&D

- net income

- interest expense

- gross profit

- depreciation and amortization

- EBIT

:param doc: company financial document

:return: dict with ttm and yearly measures

"""

measures = [

"Revenues",

"RevenueFromContractWithCustomerExcludingAssessedTax",

"RevenueFromContractWithCustomerIncludingAssessedTax",

"SalesRevenueNet"

]

\_, ttm\_revenue, yearly\_revenue = get\_values\_from\_measures(doc, measures, get\_most\_recent=False, debug=False)

last\_annual\_report\_date = None

last\_annual\_report\_fy = None

for m in measures:

df = build\_financial\_df(doc, m)

if df is not None and not df.empty and "frame" in df.columns:

annual\_rd, annual\_fy = get\_last\_annual\_report\_date\_and\_fy(df)

if last\_annual\_report\_date is None or (annual\_rd is not None and annual\_rd > last\_annual\_report\_date):

last\_annual\_report\_date = annual\_rd

last\_annual\_report\_fy = annual\_fy

#### R and D ####

measures = ["ResearchAndDevelopmentExpense"]

\_, \_, yearly\_rd = get\_values\_from\_measures(

doc, measures, get\_ttm=False, get\_most\_recent=False, debug=False)

measures = ["ResearchAndDevelopmentExpenseExcludingAcquiredInProcessCost"]

\_, \_, yearly\_rd\_not\_inprocess = get\_values\_from\_measures(

doc, measures, get\_ttm=False, get\_most\_recent=False, debug=False)

measures = ["ResearchAndDevelopmentInProcess"]

\_, \_, yearly\_rd\_inprocess = get\_values\_from\_measures(

doc, measures, get\_ttm=False, get\_most\_recent=False, debug=False)

merge\_subsets\_yearly(yearly\_rd, [yearly\_rd\_not\_inprocess, yearly\_rd\_inprocess])

#### Net Income ####

measures = [

"NetIncomeLoss",

"NetIncomeLossAvailableToCommonStockholdersBasic",

"NetIncomeLossAvailableToCommonStockholdersDiluted",

"ComprehensiveIncomeNetOfTax",

"IncomeLossFromContinuingOperations",

# including minority interest

"ProfitLoss",

"IncomeLossFromContinuingOperationsIncludingPortionAttributableToNoncontrollingInterest",

"IncomeLossFromSubsidiariesNetOfTax"

]

\_, ttm\_net\_income, yearly\_net\_income = get\_values\_from\_measures(doc, measures, get\_most\_recent=False, debug=False)

#### Interest Expenses ####

measures = [

"InterestExpense",

"InterestAndDebtExpense",

"InterestPaid",

"InterestPaidNet",

"InterestCostsIncurred"]

\_, ttm\_interest\_expenses, \_ = get\_values\_from\_measures(doc, measures, get\_most\_recent=False, get\_yearly=False,

debug=False)

# Probably we don't need yearly interest expenses

# measures = ["InterestExpenseDebt",

# "InterestExpenseDebtExcludingAmortization"]

# \_, ttm\_ie\_debt, yearly\_ie\_debt = get\_values\_from\_measures(doc, measures, get\_most\_recent=False,

# debug=False)

#

# measures = ["InterestExpenseLongTermDebt"]

# \_, ttm\_ie\_debt\_lt, yearly\_ie\_debt\_lt = get\_values\_from\_measures(doc, measures, get\_most\_recent=False,

# debug=False)

#

# measures = ["InterestExpenseShortTermBorrowings"]

# \_, ttm\_ie\_debt\_st, yearly\_ie\_debt\_st = get\_values\_from\_measures(doc, measures, get\_most\_recent=False,

# debug=False)

# merge\_subsets(yearly\_ie\_debt, [yearly\_ie\_debt\_lt, yearly\_ie\_debt\_st])

#

#

# measures = ["InterestExpenseBorrowings"]

# \_, ttm\_ie\_borrowings, yearly\_ie\_borrowings = get\_values\_from\_measures(doc, measures, get\_most\_recent=False,

# debug=False)

# measures = ["InterestExpenseDeposits"]

# \_, ttm\_ie\_deposits, yearly\_ie\_deposits = get\_values\_from\_measures(doc, measures, get\_most\_recent=False,

# debug=False)

# measures = ["InterestExpenseOther"]

# \_, ttm\_ie\_others, yearly\_ie\_others = get\_values\_from\_measures(doc, measures, get\_most\_recent=False,

# debug=False)

# measures = ["InterestExpenseRelatedParty"]

# \_, ttm\_ie\_related, yearly\_ie\_related = get\_values\_from\_measures(doc, measures, get\_most\_recent=False,

# debug=False)

#

# merge\_subsets(yearly\_ie\_borrowings, [yearly\_ie\_debt, yearly\_ie\_deposits, yearly\_ie\_others, yearly\_ie\_related])

# merge\_subsets(yearly\_interest\_expenses, [yearly\_ie\_borrowings])

#### Gross Profit ####

if ttm\_interest\_expenses == 0:

measures = ["InterestExpenseBorrowings"]

\_, ttm\_ie\_borrowings, \_ = get\_values\_from\_measures(doc, measures, get\_most\_recent=False, get\_yearly=False,

debug=False)

if ttm\_ie\_borrowings["value"] == 0:

measures = ["InterestExpenseDebt",

"InterestExpenseDebtExcludingAmortization"]

\_, ttm\_ie\_debt, \_ = get\_values\_from\_measures(doc, measures, get\_most\_recent=False, get\_yearly=False,

debug=False)

if ttm\_ie\_debt["value"] == 0:

measures = ["InterestExpenseLongTermDebt"]

\_, ttm\_ie\_debt\_lt, \_ = get\_values\_from\_measures(doc, measures, get\_most\_recent=False, get\_yearly=False,

debug=False)

measures = ["InterestExpenseShortTermBorrowings"]

\_, ttm\_ie\_debt\_st, \_ = get\_values\_from\_measures(doc, measures, get\_most\_recent=False, get\_yearly=False,

debug=False)

merge\_subsets\_most\_recent(ttm\_ie\_debt, [ttm\_ie\_debt\_lt, ttm\_ie\_debt\_st])

measures = ["InterestExpenseDeposits"]

\_, ttm\_ie\_deposits, \_ = get\_values\_from\_measures(doc, measures, get\_most\_recent=False, get\_yearly=False,

debug=False)

measures = ["InterestExpenseOther"]

\_, ttm\_ie\_others, \_ = get\_values\_from\_measures(doc, measures, get\_most\_recent=False, get\_yearly=False,

debug=False)

measures = ["InterestExpenseRelatedParty"]

\_, ttm\_ie\_related, \_ = get\_values\_from\_measures(doc, measures, get\_most\_recent=False, get\_yearly=False,

debug=False)

ttm\_ie\_borrowings = merge\_subsets\_most\_recent(ttm\_ie\_borrowings,

[ttm\_ie\_debt, ttm\_ie\_deposits, ttm\_ie\_others, ttm\_ie\_related])

ttm\_interest\_expenses = ttm\_ie\_borrowings

measures = ["Gross Profit"]

\_, ttm\_gross\_profit, yearly\_gross\_profit = get\_values\_from\_measures(doc, measures, get\_most\_recent=False,

debug=False)

#### Depreciation ####

measures = [

"DepreciationDepletionAndAmortization",

"DepreciationAmortizationAndAccretionNet"]

\_, \_, yearly\_depreciation\_amortization = get\_values\_from\_measures(doc, measures, get\_most\_recent=False, get\_ttm=False,

debug=False)

measures = ["Depreciation"]

\_, \_, yearly\_depreciation = get\_values\_from\_measures(doc, measures, get\_most\_recent=False, get\_ttm=False,

debug=False)

measures = ["AmortizationOfFinancingCostsAndDiscounts"]

\_, \_, yearly\_amortization\_fincost\_disc = get\_values\_from\_measures(doc, measures, get\_most\_recent=False, get\_ttm=False,

debug=False)

measures = ["AmortizationOfDebtDiscountPremium"]

\_, \_, yearly\_amortization\_disc = get\_values\_from\_measures(doc, measures, get\_most\_recent=False, get\_ttm=False,

debug=False)

measures = ["AmortizationOfFinancingCosts"]

\_, \_, yearly\_amortization\_fincost = get\_values\_from\_measures(doc, measures, get\_most\_recent=False, get\_ttm=False,

debug=False)

merge\_subsets\_yearly(yearly\_amortization\_fincost\_disc, [yearly\_amortization\_disc, yearly\_amortization\_fincost])

measures = ["AmortizationOfDeferredCharges"]

\_, \_, yearly\_amortization\_charges = get\_values\_from\_measures(doc, measures, get\_most\_recent=False, get\_ttm=False,

debug=False)

measures = ["AmortizationOfDeferredSalesCommissions"]

\_, \_, yearly\_amortization\_comm = get\_values\_from\_measures(doc, measures, get\_most\_recent=False, get\_ttm=False,

debug=False)

measures = ["AmortizationOfIntangibleAssets"]

\_, \_, yearly\_amortization\_intan = get\_values\_from\_measures(doc, measures, get\_most\_recent=False, get\_ttm=False,

debug=False)

yearly\_amortization = {"dates":[], "values":[]}

merge\_subsets\_yearly(yearly\_amortization, [yearly\_amortization\_fincost\_disc, yearly\_amortization\_charges,

yearly\_amortization\_comm, yearly\_amortization\_intan])

merge\_subsets\_yearly(yearly\_depreciation\_amortization, [yearly\_depreciation, yearly\_amortization])

#### EBIT ####

measures = ["OperatingIncomeLoss",

"IncomeLossFromContinuingOperationsBeforeInterestExpenseInterestIncomeIncomeTaxesExtraordinaryItemsNoncontrollingInterestsNet",

"IncomeLossFromContinuingOperationsBeforeIncomeTaxesMinorityInterestAndIncomeLossFromEquityMethodInvestments",

"IncomeLossFromContinuingOperationsBeforeIncomeTaxesExtraordinaryItemsNoncontrollingInterest",

"IncomeLossFromContinuingOperationsBeforeIncomeTaxesForeign",

"IncomeLossFromContinuingOperationsBeforeIncomeTaxesDomestic",

]

\_, ttm\_ebit, yearly\_ebit = get\_values\_from\_measures(doc, measures, get\_most\_recent=False,

debug=False)

return {

"ttm\_revenue": ttm\_revenue,

"ttm\_gross\_profit": ttm\_gross\_profit,

"ttm\_ebit": ttm\_ebit,

"ttm\_net\_income": ttm\_net\_income,

"ttm\_interest\_expenses": ttm\_interest\_expenses,

"revenue": yearly\_revenue,

"gross\_profit": yearly\_gross\_profit,

"rd": yearly\_rd,

"ebit": yearly\_ebit,

"depreciation": yearly\_depreciation\_amortization,

"net\_income": yearly\_net\_income,

"last\_annual\_report\_date": last\_annual\_report\_date,

"last\_annual\_report\_fy": last\_annual\_report\_fy

}

def extract\_balance\_sheet\_current\_assets(doc, quarter\_of\_annual\_report, years\_diff):

"""

Extract balance sheet measures (Current Assets) from company financial document.

Measures include:

- cash

- inventory

- other assets

- receivables

- securities

:param doc: company financial document

:param quarter\_of\_annual\_report: quarter of annual report

:param years\_diff: difference between fiscal year and solar year

:return: dict with most recent and yearly measures

"""

# No need for aggregate values

# #### ASSETS ####

# measures = ["Assets"]

# most\_recent\_assets, \_, yearly\_assets = get\_values\_from\_measures(

# doc, measures, instant=True, last\_annual\_report\_date=last\_annual\_report\_date, get\_ttm=False, debug=False)

#

# #### Current Assets ####

# measures = ["AssetsCurrent"]

# most\_recent\_current\_assets, \_, yearly\_current\_assets = get\_values\_from\_measures(

# doc, measures, instant=True, last\_annual\_report\_date=last\_annual\_report\_date, get\_ttm=False, debug=False)

#### Inventory ####

#### Cash ####

measures = ["CashCashEquivalentsRestrictedCashAndRestrictedCashEquivalents"]

most\_recent\_cash\_and\_restricted, \_, yearly\_cash\_and\_restricted = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = ["CashAndCashEquivalentsAtCarryingValue", "Cash"]

most\_recent\_cash, \_, yearly\_cash = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = [

"RestrictedCashAndCashEquivalentsAtCarryingValue",

"RestrictedCashAndCashEquivalents",

"RestrictedCash",

"RestrictedCashAndInvestmentsCurrent",

"RestrictedCashCurrent"

]

most\_recent\_restrictedcash, \_, yearly\_restrictedcash = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

merge\_subsets\_yearly(yearly\_cash\_and\_restricted, [yearly\_cash, yearly\_restrictedcash], must\_include=(0,))

if most\_recent\_cash\_and\_restricted["date"] is None \

or (most\_recent\_cash["date"] is not None and most\_recent\_cash["date"] > most\_recent\_cash\_and\_restricted["date"]):

most\_recent\_cash\_and\_restricted["date"] = most\_recent\_cash["date"]

most\_recent\_cash\_and\_restricted["value"] = most\_recent\_cash["value"]

if most\_recent\_restrictedcash["date"] == most\_recent\_cash["date"]:

most\_recent\_cash\_and\_restricted["value"] += most\_recent\_restrictedcash["value"]

#### Inventory ####

measures = [

"InventoryNet",

"InventoryGross",

"FIFOInventoryAmount",

"InventoryLIFOReserve",

"LIFOInventoryAmount",

]

most\_recent\_inventory, \_, yearly\_inventory = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = [

"RetailRelatedInventory",

"RetailRelatedInventoryMerchandise"

]

most\_recent\_inventory\_retail, \_, yearly\_inventory\_retail = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = [

"EnergyRelatedInventory"

]

most\_recent\_inventory\_energy, \_, yearly\_inventory\_energy = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = [

"PublicUtilitiesInventory"

]

most\_recent\_inventory\_utilities, \_, yearly\_inventory\_utilities = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = [

"InventoryRealEstate"

]

most\_recent\_inventory\_re, \_, yearly\_inventory\_re = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = [

"AirlineRelatedInventory"

]

most\_recent\_inventory\_airline, \_, yearly\_inventory\_airline = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

merge\_subsets\_most\_recent(most\_recent\_inventory,

[most\_recent\_inventory\_retail, most\_recent\_inventory\_airline,

most\_recent\_inventory\_energy, most\_recent\_inventory\_re, most\_recent\_inventory\_utilities])

merge\_subsets\_yearly(yearly\_inventory, [yearly\_inventory\_retail, yearly\_inventory\_airline, yearly\_inventory\_energy,

yearly\_inventory\_re, yearly\_inventory\_utilities])

#### Other Assets ####

measures = [

"OtherAssetsCurrent",

"OtherAssetsMiscellaneousCurrent",

"PrepaidExpenseAndOtherAssetsCurrent",

"OtherAssetsFairValueDisclosure",

"OtherAssetsMiscellaneous",

"PrepaidExpenseAndOtherAssets"

]

most\_recent\_other\_current\_assets, \_, yearly\_other\_current\_assets = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = ["PrepaidExpenseCurrent"]

most\_recent\_prepaid\_exp, \_, yearly\_prepaid\_exp = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = ["PrepaidInsurance"]

most\_recent\_prepaid\_ins, \_, yearly\_prepaid\_ins = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = ["PrepaidTaxes",

"IncomeTaxesReceivable",

"IncomeTaxReceivable"]

most\_recent\_prepaid\_tax, \_, yearly\_prepaid\_tax = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, get\_yearly=False, debug=False)

merge\_subsets\_yearly(yearly\_other\_current\_assets, [yearly\_prepaid\_exp, yearly\_prepaid\_ins, yearly\_prepaid\_tax])

merge\_subsets\_most\_recent(most\_recent\_other\_current\_assets,

[most\_recent\_prepaid\_exp, most\_recent\_prepaid\_ins, most\_recent\_prepaid\_tax])

#### Receivables ####

measures = [

"AccountsAndOtherReceivablesNetCurrent",

"AccountsNotesAndLoansReceivableNetCurrent",

"ReceivablesNetCurrent",

"NontradeReceivablesCurrent"]

most\_recent\_receivables, \_, yearly\_receivables = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = ["AccountsReceivableNetCurrent",

"AccountsReceivableNet",

"AccountsReceivableGrossCurrent",

"AccountsReceivableGross"]

most\_recent\_ar, \_, yearly\_ar = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = ["LoansAndLeasesReceivableNetReportedAmount",

"LoansAndLeasesReceivableNetOfDeferredIncome",

"LoansReceivableFairValueDisclosure",

"LoansAndLeasesReceivableGrossCarryingAmount"]

most\_recent\_loans\_rec, \_, yearly\_loans\_rec = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = ["NotesReceivableNet",

"NotesReceivableFairValueDisclosure",

"NotesReceivableGross"]

most\_recent\_notes\_rec, \_, yearly\_notes\_rec = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

merge\_subsets\_yearly(yearly\_receivables, [yearly\_ar, yearly\_loans\_rec, yearly\_notes\_rec])

merge\_subsets\_most\_recent(most\_recent\_receivables,

[most\_recent\_ar, most\_recent\_loans\_rec, most\_recent\_notes\_rec])

#### Securities ####

measures = [

"MarketableSecurities"

"AvailableForSaleSecurities"]

most\_recent\_securities, \_, yearly\_securities = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = ["AvailableForSaleSecuritiesDebtSecurities"]

most\_recent\_debtsecurities, \_, yearly\_debtsecurities = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = ["AvailableForSaleSecuritiesEquitySecurities"]

most\_recent\_equitysecurities, \_, yearly\_equitysecurities = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

merge\_subsets\_yearly(yearly\_securities, [yearly\_debtsecurities, yearly\_equitysecurities])

merge\_subsets\_most\_recent(most\_recent\_securities,

[most\_recent\_debtsecurities, most\_recent\_equitysecurities])

measures = ["DerivativeAssets",

"DerivativeAssetsCurrent"]

most\_recent\_derivatives, \_, yearly\_derivatives = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = ["HeldToMaturitySecurities",

"HeldToMaturitySecuritiesFairValue",

"HeldToMaturitySecuritiesCurrent",

]

most\_recent\_held\_securities, \_, yearly\_held\_securities = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = ["AvailableForSaleSecuritiesNoncurrent",

"AvailableForSaleSecuritiesDebtSecuritiesNoncurrent",

]

most\_recent\_non\_curr\_sec, \_, yearly\_non\_curr\_sec = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = ["MarketableSecuritiesCurrent",

"AvailableForSaleSecuritiesDebtSecuritiesCurrent"]

most\_recent\_marksecurities\_cur, \_, yearly\_marksecurities\_cur = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = ["ShortTermInvestments"]

most\_recent\_st\_inv, \_, yearly\_st\_inv = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = ["MoneyMarketFundsAtCarryingValue"]

most\_recent\_mm, \_, yearly\_mm = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

merge\_subsets\_yearly(yearly\_securities, [yearly\_derivatives, yearly\_held\_securities, yearly\_non\_curr\_sec,

yearly\_marksecurities\_cur, yearly\_st\_inv, yearly\_mm])

merge\_subsets\_most\_recent(most\_recent\_securities,

[most\_recent\_derivatives, most\_recent\_held\_securities, most\_recent\_non\_curr\_sec,

most\_recent\_marksecurities\_cur, most\_recent\_st\_inv, most\_recent\_mm])

# merge\_subsets(yearly\_current\_assets, [yearly\_cash\_and\_restricted, yearly\_inventory, yearly\_other\_current\_assets,

# yearly\_receivables, yearly\_securities])

return {

"mr\_cash": most\_recent\_cash\_and\_restricted,

"cash": yearly\_cash\_and\_restricted,

"mr\_inventory": most\_recent\_inventory,

"inventory": yearly\_inventory,

"mr\_other\_assets": most\_recent\_other\_current\_assets,

"other\_assets": yearly\_other\_current\_assets,

"mr\_receivables": most\_recent\_receivables,

"receivables": yearly\_receivables,

"mr\_securities": most\_recent\_securities,

"securities": yearly\_securities

}

def extract\_balance\_sheet\_noncurrent\_assets(doc, quarter\_of\_annual\_report, years\_diff):

"""

Extract balance sheet measures (Non-Current Assets) from company financial document.

Measures include:

- equity investments

- other financial assets

- PP&E

- investment property

- tax benefits

:param doc: company financial document

:param quarter\_of\_annual\_report: quarter of annual report

:param years\_diff: difference between fiscal year and solar year

:return: dict with most recent measures

"""

# ##### Non current assets ####

# measures = ["AssetsNoncurrent",

# "NoncurrentAssets"]

# most\_recent\_non\_curr\_asset, \_, yearly\_non\_curr\_asset = get\_values\_from\_measures(

# doc, measures, instant=True, last\_annual\_report\_date=last\_annual\_report\_date, get\_ttm=False, debug=False)

#### Equity investments ####

#### Equity Investments ####

measures = [

"EquityMethodInvestmentAggregateCost",

"EquityMethodInvestments",

"InvestmentOwnedAtCost",

"Investments",

"InvestmentsInAffiliatesSubsidiariesAssociatesAndJointVentures",

]

most\_recent\_equity\_investments, \_, \_ = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, get\_yearly=False,

debug=False)

measures = [

"EquityMethodInvestmentsFairValueDisclosure",

"InvestmentOwnedAtFairValue",

"InvestmentsFairValueDisclosure",

]

most\_recent\_equity\_inv\_fv, \_, \_ = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, get\_yearly=False, debug=False)

measures = ["EquitySecuritiesWithoutReadilyDeterminableFairValueAmount", ]

most\_recent\_equity\_inv\_notfv, \_, \_ = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, get\_yearly=False, debug=False)

# merge\_subsets\_yearly(yearly\_equity\_investments, [yearly\_equity\_inv\_fv, yearly\_equity\_inv\_notfv])

merge\_subsets\_most\_recent(most\_recent\_equity\_investments,

[most\_recent\_equity\_inv\_fv, most\_recent\_equity\_inv\_notfv])

measures = ["MarketableSecuritiesNoncurrent"]

most\_recent\_securities\_non\_curr, \_, \_ = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, get\_yearly=False, debug=False)

# yearly\_equity\_investments\_and\_securities = {"dates": [], "values": []}

# merge\_subsets\_yearly(yearly\_equity\_investments\_and\_securities, [yearly\_equity\_investments, yearly\_securities\_non\_curr])

merge\_subsets\_most\_recent(most\_recent\_equity\_investments, [most\_recent\_securities\_non\_curr])

#### Other financial assets ####

measures = [

"PrepaidExpenseNoncurrent",

"PrepaidExpenseOtherNoncurrent",

]

most\_recent\_prepaid\_non\_curr, \_, \_ = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, get\_yearly=False, debug=False)

measures = [

"RestrictedCashAndCashEquivalentsNoncurrent",

"RestrictedCashAndInvestmentsNoncurrent",

"RestrictedCashNoncurrent"

]

most\_recent\_cash\_non\_curr, \_, \_ = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, get\_yearly=False, debug=False)

measures = ["DerivativeAssetsNoncurrent", ]

most\_recent\_derivatives\_non\_curr, \_, \_ = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, get\_yearly=False, debug=False)

measures = ["EscrowDeposit"]

most\_recent\_escrow, \_, \_ = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, get\_yearly=False, debug=False)

# yearly\_other\_financial\_assets = {"dates": [], "values": []}

# merge\_subsets\_yearly(yearly\_other\_financial\_assets, [yearly\_prepaid\_non\_curr, yearly\_cash\_non\_curr,

# yearly\_derivatives\_non\_curr, yearly\_escrow])

most\_recent\_other\_financial\_assets = {"date":None, "value":0}

merge\_subsets\_most\_recent(most\_recent\_other\_financial\_assets,

[most\_recent\_prepaid\_non\_curr, most\_recent\_cash\_non\_curr,

most\_recent\_derivatives\_non\_curr, most\_recent\_escrow])

#### PP&E ####

measures = [

"PropertyPlantAndEquipmentNet",

"PropertyPlantAndEquipmentAndFinanceLeaseRightOfUseAssetAfterAccumulatedDepreciationAndAmortization"

]

most\_recent\_ppe, \_, \_ = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, get\_yearly=False, debug=False)

#### Investment property ####

measures = [

"RealEstateInvestments",

"RealEstateInvestmentPropertyNet",

"RealEstateInvestmentPropertyAtCost",

"RealEstateHeldforsale"

]

most\_recent\_property, \_, \_ = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, get\_yearly=False, debug=False)

measures = ["InvestmentBuildingAndBuildingImprovements"]

most\_recent\_buildings, \_, \_ = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, get\_yearly=False, debug=False)

measures = [

"LandAndLandImprovements",

"Land",

]

most\_recent\_land, \_, \_ = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, get\_yearly=False, debug=False)

# merge\_subsets\_yearly(yearly\_property, [yearly\_buildings, yearly\_land])

merge\_subsets\_most\_recent(most\_recent\_property,

[most\_recent\_buildings, most\_recent\_land])

# merge\_subsets(yearly\_non\_curr\_asset, [yearly\_property, yearly\_ppe, yearly\_other\_financial\_assets,

# yearly\_equity\_investments\_and\_securities])

# merge\_subsets(yearly\_assets, [yearly\_current\_assets, yearly\_non\_curr\_asset])

#### Tax Benefits ####

measures = [

"UnrecognizedTaxBenefits",

"UnrecognizedTaxBenefitsThatWouldImpactEffectiveTaxRate",

"IncomeTaxesReceivableNoncurrent",

]

most\_recent\_tax\_benefit, \_, \_ = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, get\_yearly=False, debug=False)

return {

"mr\_equity\_investments": most\_recent\_equity\_investments,

"mr\_other\_financial\_assets": most\_recent\_other\_financial\_assets,

"mr\_ppe": most\_recent\_ppe,

"mr\_investment\_property": most\_recent\_property,

"mr\_tax\_benefits": most\_recent\_tax\_benefit

}

def extract\_balance\_sheet\_debt(doc, quarter\_of\_annual\_report, years\_diff):

"""

Extract balance sheet DEBT measures (Current + Non-Current) from company financial document.

:param doc: company financial document

:param quarter\_of\_annual\_report: quarter of annual report

:param years\_diff: difference between fiscal year and solar year

:return: dict with most recent and yearly measures

"""

# DEBT LONG + SHORT

measures = ["DebtLongtermAndShorttermCombinedAmount"]

most\_recent\_debt, \_, yearly\_debt = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = ["MortgageLoansOnRealEstate"]

most\_recent\_mortgage, \_, yearly\_mortgage = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = ["OtherBorrowings"]

most\_recent\_other\_borr, \_, yearly\_other\_borr = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

# DEBT SHORT

measures = [

"ShortTermBorrowings",

"ShorttermDebtAverageOutstandingAmount",

"ShorttermDebtFairValue",

]

most\_recent\_debt\_st, \_, yearly\_debt\_st = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = [

"CommercialPaper",

"CommercialPaperAtCarryingValue",

]

most\_recent\_cp, \_, yearly\_cp = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = ["BankOverdrafts"]

most\_recent\_overdraft, \_, yearly\_overdraft = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = ["ShortTermBankLoansAndNotesPayable"]

most\_recent\_loans\_st, \_, yearly\_loans\_st = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = ["BridgeLoan"]

most\_recent\_bridge, \_, yearly\_bridge = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

# DEBT LONG

measures = [

"LongTermDebtAndCapitalLeaseObligationsIncludingCurrentMaturities",

"LongTermDebt",

"LongTermDebtFairValue",

"DebtInstrumentFaceAmount",

"DebtInstrumentCarryingAmount",

]

most\_recent\_debt\_lt, \_, yearly\_debt\_lt = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = [

"ConvertibleDebt",

"ConvertibleNotesPayable",

]

most\_recent\_convertible, \_, yearly\_convertible = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = [

"LineOfCredit",

"LineOfCreditFacilityFairValueOfAmountOutstanding",

]

most\_recent\_revolver, \_, yearly\_revolver = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = ["LoansPayable"]

most\_recent\_loans\_pay, \_, yearly\_loans\_pay = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = ["SecuredDebt"]

most\_recent\_debt\_sec, \_, yearly\_debt\_sec = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = ["NotesPayable",

"SeniorNotes"]

most\_recent\_debt\_notes, \_, yearly\_debt\_notes = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = ["UnsecuredDebt"]

most\_recent\_debt\_unsec, \_, yearly\_debt\_unsec = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

# DEBT LONG - CURRENT

measures = [

"LongTermDebtAndCapitalLeaseObligationsCurrent",

"LongTermDebtCurrent",

]

most\_recent\_debt\_lt\_cur, \_, yearly\_debt\_lt\_cur = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = [

"ConvertibleDebtCurrent",

"ConvertibleNotesPayableCurrent",

]

most\_recent\_convertible\_cur, \_, yearly\_convertible\_cur = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = ["LinesOfCreditCurrent"]

most\_recent\_revolver\_cur, \_, yearly\_revolver\_cur = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = ["NotesPayableCurrent",

"SeniorNotesCurrent"]

most\_recent\_notes\_cur, \_, yearly\_notes\_cur = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = ["SecuredDebtCurrent"]

most\_recent\_debt\_sec\_cur, \_, yearly\_debt\_sec\_cur = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = ["UnsecuredDebtCurrent"]

most\_recent\_debt\_unsec\_cur, \_, yearly\_debt\_unsec\_cur = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

# DEBT LONG - NON CURRENT

measures = [

"LongTermDebtAndCapitalLeaseObligations",

"LongTermDebtNoncurrent",

]

most\_recent\_debt\_lt\_noncur, \_, yearly\_debt\_lt\_noncur = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = [

"ConvertibleDebtNoncurrent",

"ConvertibleLongTermNotesPayable",

]

most\_recent\_convertible\_noncur, \_, yearly\_convertible\_noncur = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = ["LongTermLineOfCredit"]

most\_recent\_revolver\_noncur, \_, yearly\_revolver\_noncur = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = ["LongTermNotesPayable",

"SeniorLongTermNotes"]

most\_recent\_notes\_noncur, \_, yearly\_notes\_noncur = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = ["SecuredLongTermDebt"]

most\_recent\_debt\_sec\_noncur, \_, yearly\_debt\_sec\_noncur = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = ["UnsecuredLongTermDebt"]

most\_recent\_debt\_unsec\_noncur, \_, yearly\_debt\_unsec\_noncur = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

merge\_subsets\_yearly(yearly\_debt\_st, [yearly\_cp, yearly\_overdraft, yearly\_bridge, yearly\_loans\_st])

merge\_subsets\_yearly(yearly\_debt\_lt, [yearly\_convertible, yearly\_revolver, yearly\_loans\_pay, yearly\_debt\_sec,

yearly\_debt\_notes, yearly\_debt\_unsec])

merge\_subsets\_yearly(yearly\_debt\_lt\_cur, [yearly\_convertible\_cur, yearly\_revolver\_cur, yearly\_notes\_cur,

yearly\_debt\_sec\_cur, yearly\_debt\_unsec\_cur])

merge\_subsets\_yearly(yearly\_debt\_lt\_noncur, [yearly\_convertible\_noncur, yearly\_revolver\_noncur, yearly\_notes\_noncur,

yearly\_debt\_sec\_noncur, yearly\_debt\_unsec\_noncur])

merge\_subsets\_yearly(yearly\_debt\_lt, [yearly\_debt\_lt\_cur, yearly\_debt\_lt\_noncur])

merge\_subsets\_yearly(yearly\_debt, [yearly\_debt\_lt, yearly\_debt\_st])

merge\_subsets\_yearly(yearly\_debt, [yearly\_debt, yearly\_mortgage, yearly\_other\_borr], (0,))

merge\_subsets\_most\_recent(most\_recent\_debt\_st, [most\_recent\_cp, most\_recent\_overdraft, most\_recent\_bridge, most\_recent\_loans\_st])

merge\_subsets\_most\_recent(most\_recent\_debt\_lt, [most\_recent\_convertible, most\_recent\_revolver, most\_recent\_loans\_pay, most\_recent\_debt\_sec,

most\_recent\_debt\_notes, most\_recent\_debt\_unsec])

merge\_subsets\_most\_recent(most\_recent\_debt\_lt\_cur, [most\_recent\_convertible\_cur, most\_recent\_revolver\_cur, most\_recent\_notes\_cur,

most\_recent\_debt\_sec\_cur, most\_recent\_debt\_unsec\_cur])

merge\_subsets\_most\_recent(most\_recent\_debt\_lt\_noncur, [most\_recent\_convertible\_noncur, most\_recent\_revolver\_noncur, most\_recent\_notes\_noncur,

most\_recent\_debt\_sec\_noncur, most\_recent\_debt\_unsec\_noncur])

merge\_subsets\_most\_recent(most\_recent\_debt\_lt, [most\_recent\_debt\_lt\_cur, most\_recent\_debt\_lt\_noncur])

merge\_subsets\_most\_recent(most\_recent\_debt, [most\_recent\_debt\_lt, most\_recent\_debt\_st])

for m in [most\_recent\_mortgage, most\_recent\_other\_borr]:

if m["date"] == most\_recent\_debt["date"]:

most\_recent\_debt["value"] += m["value"]

return {

"mr\_debt": most\_recent\_debt,

"debt": yearly\_debt

}

def extract\_balance\_sheet\_liabilities(doc, quarter\_of\_annual\_report, years\_diff, most\_recent\_debt):

"""

Extract balance sheet measures (Total Liabilities) from company financial document.

:param doc: company financial document

:param quarter\_of\_annual\_report: quarter of annual report

:param years\_diff: difference between fiscal year and solar year

:return: dict with most recent measures

"""

measures = [

"Liabilities",

"LiabilitiesFairValueDisclosure",

"LiabilitiesAssumed1",

]

most\_recent\_liabilities, \_, \_ = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, get\_yearly=False, debug=False)

measures = ["DerivativeLiabilities"]

most\_recent\_derivatives\_liability, \_, yearly\_derivatives\_liability = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = [

"AccountsPayableAndAccruedLiabilitiesCurrentAndNoncurrent",

"AccountsPayableCurrentAndNoncurrent",

]

most\_recent\_ap, \_, yearly\_ap = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = ["DueToRelatedPartiesCurrentAndNoncurrent"]

most\_recent\_due\_related\_parties, \_, yearly\_due\_related\_parties = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = ["DueToAffiliateCurrentAndNoncurrent"]

most\_recent\_due\_affiliates, \_, yearly\_due\_affiliates = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

# yearly\_liabilities\_ex\_debt = {"dates": [], "values": []}

# merge\_subsets\_yearly(yearly\_liabilities\_ex\_debt, [yearly\_derivatives\_liability, yearly\_ap, yearly\_due\_related\_parties,

# yearly\_due\_affiliates])

# Current

measures = ["LiabilitiesCurrent"]

most\_recent\_liabilities\_cur, \_, \_ = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, get\_yearly=False, debug=False)

measures = [

"AccountsPayableAndOtherAccruedLiabilitiesCurrent",

"AccountsPayableAndAccruedLiabilitiesCurrent",

]

most\_recent\_ap\_complete\_cur, \_, yearly\_ap\_complete\_cur = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = ["AccountsPayableCurrent"]

most\_recent\_ap\_cur, \_, yearly\_ap\_cur = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = ["AccountsPayableOtherCurrent"]

most\_recent\_apother\_cur, \_, yearly\_apother\_cur = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = ["AccountsPayableRelatedPartiesCurrent"]

most\_recent\_ap\_rel\_cur, \_, yearly\_ap\_rel\_cur = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = ["AccountsPayableTradeCurrent"]

most\_recent\_ap\_trade\_cur, \_, yearly\_ap\_trade\_cur = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

merge\_subsets\_yearly(yearly\_ap\_complete\_cur, [yearly\_ap\_cur, yearly\_apother\_cur, yearly\_ap\_rel\_cur,

yearly\_ap\_trade\_cur])

measures = ["DueToAffiliateCurrent"]

most\_recent\_due\_affiliates\_cur, \_, yearly\_due\_affiliates\_cur = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = ["DueToRelatedPartiesCurrent"]

most\_recent\_due\_related\_cur, \_, yearly\_due\_related\_cur = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = ["DerivativeLiabilitiesCurrent"]

most\_recent\_derivatives\_liability\_cur, \_, yearly\_derivatives\_liability\_cur = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

# merge\_subsets\_yearly(yearly\_liabilities\_cur, [yearly\_ap\_complete\_cur, yearly\_due\_affiliates\_cur, yearly\_due\_related\_cur,

# yearly\_derivatives\_liability\_cur])

# Non - Current

measures = ["LiabilitiesNoncurrent"]

most\_recent\_liabilities\_noncur, \_, \_ = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, get\_yearly=False, debug=False)

measures = ["DerivativeLiabilitiesNoncurrent"]

most\_recent\_derivatives\_liability\_noncur, \_, \_ = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, get\_yearly=False, debug=False)

measures = ["DueToAffiliateNoncurrent"]

most\_recent\_due\_affiliates\_noncur, \_, \_ = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, get\_yearly=False, debug=False)

measures = ["DueToRelatedPartiesNoncurrent"]

most\_recent\_due\_related\_noncur, \_, \_ = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, get\_yearly=False, debug=False)

# merge\_subsets\_yearly(yearly\_liabilities\_noncur, [yearly\_derivatives\_liability\_noncur, yearly\_due\_affiliates\_noncur,

# yearly\_due\_related\_noncur])

# merge\_subsets\_yearly(yearly\_liabilities\_ex\_debt, [yearly\_liabilities\_cur, yearly\_liabilities\_noncur])

# merge\_subsets\_yearly(yearly\_liabilities, [yearly\_liabilities\_ex\_debt, yearly\_debt])

merge\_subsets\_most\_recent(most\_recent\_ap\_complete\_cur, [most\_recent\_ap\_cur, most\_recent\_apother\_cur,

most\_recent\_ap\_rel\_cur, most\_recent\_ap\_trade\_cur])

merge\_subsets\_most\_recent(most\_recent\_ap, [most\_recent\_ap\_complete\_cur])

merge\_subsets\_most\_recent(most\_recent\_derivatives\_liability, [most\_recent\_derivatives\_liability\_cur,

most\_recent\_derivatives\_liability\_noncur])

merge\_subsets\_most\_recent(most\_recent\_due\_affiliates, [most\_recent\_due\_affiliates\_cur,

most\_recent\_due\_affiliates\_noncur])

merge\_subsets\_most\_recent(most\_recent\_due\_related\_parties, [most\_recent\_due\_related\_cur,

most\_recent\_due\_related\_noncur])

merge\_subsets\_most\_recent(most\_recent\_liabilities, [most\_recent\_liabilities\_cur, most\_recent\_liabilities\_noncur])

combo\_liabilities = {"date":None, "value":0}

merge\_subsets\_most\_recent(combo\_liabilities, [most\_recent\_ap, most\_recent\_derivatives\_liability,

most\_recent\_due\_affiliates, most\_recent\_due\_related\_parties,

most\_recent\_debt])

if most\_recent\_liabilities["date"] is None:

most\_recent\_liabilities = combo\_liabilities

elif combo\_liabilities["date"] is not None:

if combo\_liabilities["date"] > most\_recent\_liabilities["date"]:

most\_recent\_liabilities = combo\_liabilities

elif combo\_liabilities["date"] == most\_recent\_liabilities["date"] and combo\_liabilities["value"] > most\_recent\_liabilities["value"]:

most\_recent\_liabilities = combo\_liabilities

# for working capital

merge\_subsets\_yearly(yearly\_ap\_complete\_cur, [yearly\_ap])

merge\_subsets\_yearly(yearly\_due\_affiliates\_cur, [yearly\_due\_affiliates])

merge\_subsets\_yearly(yearly\_due\_related\_cur, [yearly\_due\_related\_parties])

return {

"mr\_liabilities": most\_recent\_liabilities,

"account\_payable": yearly\_ap\_complete\_cur,

"due\_to\_affiliates": yearly\_due\_affiliates\_cur,

"due\_to\_related\_parties": yearly\_due\_related\_cur

}

def extract\_balance\_sheet\_equity(doc, last\_annual\_report\_date, last\_annual\_report\_fy):

"""

Extract balance sheet EQUITY measures from company financial document.

:param doc: company financial document

:param last\_annual\_report\_date: date of last annual report

:return: dict with most recent and yearly measures

"""

# measures = ["LiabilitiesAndStockholdersEquity"]

# most\_recent\_liabilities\_and\_equity, \_, yearly\_liabilities\_and\_equity = get\_values\_from\_measures(

# doc, measures, instant=True, last\_annual\_report\_date=last\_annual\_report\_date, get\_ttm=False, debug=False)

df = build\_financial\_df(doc, "StockholdersEquityIncludingPortionAttributableToNoncontrollingInterest")

quarter\_of\_annual\_report, years\_diff = get\_quarter\_of\_annual\_report(df, last\_annual\_report\_date, last\_annual\_report\_fy)

if quarter\_of\_annual\_report is None:

df = build\_financial\_df(doc, "StockholdersEquity")

quarter\_of\_annual\_report, years\_diff = get\_quarter\_of\_annual\_report(df, last\_annual\_report\_date,

last\_annual\_report\_fy)

measures = ["StockholdersEquityIncludingPortionAttributableToNoncontrollingInterest"]

most\_recent\_equity, \_, yearly\_equity = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = ["StockholdersEquity"]

most\_recent\_equity\_no\_mi, \_, yearly\_equity\_no\_mi = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

measures = ["MinorityInterest"]

most\_recent\_minority\_interest, \_, yearly\_minority\_interest = get\_values\_from\_measures(

doc, measures, instant=True, quarter\_of\_annual\_report=quarter\_of\_annual\_report, years\_diff=years\_diff,

get\_ttm=False, debug=False)

merge\_subsets\_yearly(yearly\_equity, [yearly\_equity\_no\_mi, yearly\_minority\_interest], (0,))

if most\_recent\_equity["date"] is None or \

(most\_recent\_equity\_no\_mi["date"] is not None and most\_recent\_equity\_no\_mi["date"] > most\_recent\_equity["date"]):

merge\_subsets\_most\_recent(most\_recent\_equity, [most\_recent\_equity\_no\_mi, most\_recent\_minority\_interest])

return {

"mr\_equity": most\_recent\_equity,

"equity": yearly\_equity,

"mr\_minority\_interest": most\_recent\_minority\_interest,

"quarter\_of\_annual\_report": quarter\_of\_annual\_report,

"years\_diff": years\_diff

}

def extract\_cashflow\_statement(doc):

"""

Extract cashflow statement measures from company financial document.

Measures include:

- dividends

- CAPEX

- net income

- interest expense

- gross profit

- depreciation and amortization

- EBIT

:param doc: company financial document

:return: dict with ttm and yearly measures

"""

# DIVIDENDS

measures = [

"Dividends",

"DividendsCash",

"PaymentsOfDividends",

"PaymentsOfOrdinaryDividends"

]

\_, ttm\_dividends, yearly\_dividends = get\_values\_from\_measures(doc, measures, get\_most\_recent=False,

debug=False)

measures = [

"DividendsCommonStock",

"DividendsCommonStockCash",

"PaymentsOfDividendsCommonStock"

]

\_, ttm\_dividends\_cs, yearly\_dividends\_cs = get\_values\_from\_measures(doc, measures, get\_most\_recent=False,

debug=False)

measures = [

"DividendsPreferredStock",

"DividendsPreferredStockCash",

"PaymentsOfDividendsPreferredStockAndPreferenceStock"

]

\_, ttm\_dividends\_ps, yearly\_dividends\_ps = get\_values\_from\_measures(doc, measures, get\_most\_recent=False,

debug=False)

merge\_subsets\_yearly(yearly\_dividends, [yearly\_dividends\_cs, yearly\_dividends\_ps])

merge\_subsets\_most\_recent(ttm\_dividends, [ttm\_dividends\_cs, ttm\_dividends\_ps])

# CAPEX

# Acquisition

measures = ["BusinessAcquisitionCostOfAcquiredEntityTransactionCosts"]

\_, \_, yearly\_acquisition\_costs = get\_values\_from\_measures(doc, measures, get\_most\_recent=False, get\_ttm=False,

debug=False)

measures = [

"PaymentsForPreviousAcquisition",

"PaymentsForProceedsFromPreviousAcquisition",

]

\_, \_, yearly\_acquisition\_adj = get\_values\_from\_measures(doc, measures, get\_most\_recent=False, get\_ttm=False,

debug=False)

measures = [

"PaymentsToAcquireBusinessesNetOfCashAcquired",

"PaymentsToAcquireBusinessesGross",

]

\_, \_, yearly\_acquisition = get\_values\_from\_measures(doc, measures, get\_most\_recent=False, get\_ttm=False,

debug=False)

measures = ["PaymentsToAcquireBusinessTwoNetOfCashAcquired"]

\_, \_, yearly\_acquisition2 = get\_values\_from\_measures(doc, measures, get\_most\_recent=False, get\_ttm=False,

debug=False)

measures = ["PaymentsToAcquireInterestInSubsidiariesAndAffiliates"]

\_, \_, yearly\_sub\_aff = get\_values\_from\_measures(doc, measures, get\_most\_recent=False, get\_ttm=False,

debug=False)

measures = ["PaymentsToAcquireAdditionalInterestInSubsidiaries"]

\_, \_, yearly\_sub = get\_values\_from\_measures(doc, measures, get\_most\_recent=False, get\_ttm=False,

debug=False)

measures = ["PaymentsToAcquireBusinessesAndInterestInAffiliates"]

\_, \_, yearly\_aff = get\_values\_from\_measures(doc, measures, get\_most\_recent=False, get\_ttm=False,

debug=False)

merge\_subsets\_yearly(yearly\_sub\_aff, [yearly\_sub, yearly\_aff])

measures = ["PaymentsToAcquireInterestInJointVenture"]

\_, \_, yearly\_jv = get\_values\_from\_measures(doc, measures, get\_most\_recent=False, get\_ttm=False,

debug=False)

# PP&E

measures = ["CapitalExpendituresIncurredButNotYetPaid"]

\_, \_, yearly\_capex\_not\_paid = get\_values\_from\_measures(doc, measures, get\_most\_recent=False, get\_ttm=False,

debug=False)

measures = ["PaymentsForCapitalImprovements"]

\_, \_, yearly\_capex\_imp = get\_values\_from\_measures(doc, measures, get\_most\_recent=False, get\_ttm=False,

debug=False)

measures = ["PaymentsToAcquireOtherProductiveAssets"]

\_, \_, yearly\_productive\_assets\_other = get\_values\_from\_measures(doc, measures, get\_most\_recent=False, get\_ttm=False,

debug=False)

measures = ["PaymentsToAcquireOtherPropertyPlantAndEquipment"]

\_, \_, yearly\_ppe\_other = get\_values\_from\_measures(doc, measures, get\_most\_recent=False, get\_ttm=False,

debug=False)

measures = ["PaymentsToAcquireProductiveAssets"]

\_, \_, yearly\_productive\_assets = get\_values\_from\_measures(doc, measures, get\_most\_recent=False, get\_ttm=False,

debug=False)

measures = ["PaymentsToAcquirePropertyPlantAndEquipment"]

\_, \_, yearly\_ppe = get\_values\_from\_measures(doc, measures, get\_most\_recent=False, get\_ttm=False,

debug=False)

measures = ["PaymentsForSoftware"]

\_, \_, yearly\_sw = get\_values\_from\_measures(doc, measures, get\_most\_recent=False, get\_ttm=False,

debug=False)

measures = ["PaymentsToDevelopSoftware"]

\_, \_, yearly\_sw\_dev = get\_values\_from\_measures(doc, measures, get\_most\_recent=False, get\_ttm=False,

debug=False)

# Intangibles

measures = ["PaymentsToAcquireIntangibleAssets"]

\_, \_, yearly\_intangibles = get\_values\_from\_measures(doc, measures, get\_most\_recent=False, get\_ttm=False,

debug=False)

yearly\_capex = {"dates": [], "values": []}

merge\_subsets\_yearly(yearly\_capex, [yearly\_acquisition\_costs, yearly\_acquisition\_adj, yearly\_acquisition,

yearly\_acquisition2, yearly\_sub\_aff, yearly\_jv, yearly\_capex\_not\_paid,

yearly\_capex\_imp, yearly\_productive\_assets\_other, yearly\_ppe\_other,

yearly\_productive\_assets,

yearly\_ppe, yearly\_sw, yearly\_sw\_dev, yearly\_intangibles])

return {

"ttm\_dividends": ttm\_dividends,

"dividends": yearly\_dividends,

"capex": yearly\_capex

}

def extract\_operating\_leases(doc):

"""

Extract operating leases measures from company financial document.

:param doc: company financial document

:return: dict with most recent measures

"""

# Last year expenses

measures = [

"OperatingLeasePayments",

"OperatingLeaseCost",

"OperatingLeaseExpense",

]

\_, mr\_op\_leases\_expense, \_ = get\_values\_from\_measures(doc, measures, get\_ttm=True, get\_most\_recent=False, get\_yearly=False, debug=False)

# Next year expenses

measures = [

"LesseeOperatingLeaseLiabilityPaymentsDueNextTwelveMonths",

"OperatingLeasesFutureMinimumPaymentsDueCurrent",

"LesseeOperatingLeaseLiabilityPaymentsDueNextRollingTwelveMonths",

]

mr\_op\_leases\_next\_year, \_, \_ = get\_values\_from\_measures(doc, measures, get\_ttm=False, get\_yearly=False, debug=False)

# Next 2year expenses

measures = [

"LesseeOperatingLeaseLiabilityPaymentsDueYearTwo",

"OperatingLeasesFutureMinimumPaymentsDueInTwoYears",

"LesseeOperatingLeaseLiabilityPaymentsDueInRollingYearTwo",

]

mr\_op\_leases\_next\_2year, \_, \_ = get\_values\_from\_measures(doc, measures, get\_ttm=False, get\_yearly=False, debug=False)

# Next 3year expenses

measures = [

"LesseeOperatingLeaseLiabilityPaymentsDueYearThree",

"OperatingLeasesFutureMinimumPaymentsDueInThreeYears",

"LesseeOperatingLeaseLiabilityPaymentsDueInRollingYearThree",

]

mr\_op\_leases\_next\_3year, \_, \_ = get\_values\_from\_measures(doc, measures, get\_ttm=False, get\_yearly=False, debug=False)

# Next 4year expenses

measures = [

"LesseeOperatingLeaseLiabilityPaymentsDueYearFour",

"OperatingLeasesFutureMinimumPaymentsDueInFourYears",

"LesseeOperatingLeaseLiabilityPaymentsDueInRollingYearFour",

]

mr\_op\_leases\_next\_4year, \_, \_ = get\_values\_from\_measures(doc, measures, get\_ttm=False, get\_yearly=False, debug=False)

# Next 5year expenses

measures = [

"LesseeOperatingLeaseLiabilityPaymentsDueYearFive",

"OperatingLeasesFutureMinimumPaymentsDueInFiveYears",

"LesseeOperatingLeaseLiabilityPaymentsDueInRollingYearFive",

]

mr\_op\_leases\_next\_5year, \_, \_ = get\_values\_from\_measures(doc, measures, get\_ttm=False, get\_yearly=False, debug=False)

# After 5year expenses

measures = [

"LesseeOperatingLeaseLiabilityPaymentsDueAfterYearFive",

"OperatingLeasesFutureMinimumPaymentsDueThereafter",

"LesseeOperatingLeaseLiabilityPaymentsDueAfterRollingYearFive",

]

mr\_op\_leases\_after\_5year, \_, \_ = get\_values\_from\_measures(doc, measures, get\_ttm=False, get\_yearly=False, debug=False)

return {

"mr\_op\_leases\_expense": mr\_op\_leases\_expense,

"mr\_op\_leases\_next\_year": mr\_op\_leases\_next\_year,

"mr\_op\_leases\_next\_2year": mr\_op\_leases\_next\_2year,

"mr\_op\_leases\_next\_3year": mr\_op\_leases\_next\_3year,

"mr\_op\_leases\_next\_4year": mr\_op\_leases\_next\_4year,

"mr\_op\_leases\_next\_5year": mr\_op\_leases\_next\_5year,

"mr\_op\_leases\_after\_5year": mr\_op\_leases\_after\_5year

}

def extract\_options(doc):

"""

Extract options measures from company financial document.

:param doc: company financial document

:return: dict with most recent measures

"""

# Last year expenses

measures = [

"EmployeeServiceShareBasedCompensationNonvestedAwardsTotalCompensationCostNotYetRecognized",

]

mr\_sbc, \_, \_ = get\_values\_from\_measures(doc, measures, get\_ttm=False, get\_yearly=False, debug=False)

measures = [

# options

"EmployeeServiceShareBasedCompensationNonvestedAwardsTotalCompensationCostNotYetRecognizedStockOptions",

"ShareBasedCompensationArrangementByShareBasedPaymentAwardOptionsOutstandingIntrinsicValue",

"ShareBasedCompensationArrangementByShareBasedPaymentAwardOptionsVestedAndExpectedToVestOutstandingAggregateIntrinsicValue",

"ShareBasedCompensationArrangementByShareBasedPaymentAwardOptionsVestedAndExpectedToVestExercisableAggregateIntrinsicValue",

"SharebasedCompensationArrangementBySharebasedPaymentAwardOptionsExercisableIntrinsicValue1",

]

mr\_sbc\_options, \_, \_ = get\_values\_from\_measures(doc, measures, get\_ttm=False, get\_yearly=False, debug=False)

measures = [

# non-options

"EmployeeServiceShareBasedCompensationNonvestedAwardsTotalCompensationCostNotYetRecognizedShareBasedAwardsOtherThanOptions",

"SharebasedCompensationArrangementBySharebasedPaymentAwardEquityInstrumentsOtherThanOptionsAggregateIntrinsicValueOutstanding",

"SharebasedCompensationArrangementBySharebasedPaymentAwardEquityInstrumentsOtherThanOptionsAggregateIntrinsicValueNonvested",

]

mr\_sbc\_non\_options, \_, \_ = get\_values\_from\_measures(doc, measures, get\_ttm=False, get\_yearly=False, debug=False)

merge\_subsets\_most\_recent(mr\_sbc, [mr\_sbc\_options, mr\_sbc\_non\_options])

return {

"mr\_sbc": mr\_sbc,

}

def extract\_company\_financial\_information(cik):

"""

Extract financial data required for valuation from company financial document

:param cik: company cik

:return: dict with income statement and balance sheet metrics

"""

try:

doc = mongodb.get\_document("financial\_data", cik)

except:

download\_financial\_data(cik)

doc = mongodb.get\_document("financial\_data", cik)

income\_statement\_measures = extract\_income\_statement(doc)

last\_annual\_report\_date = income\_statement\_measures["last\_annual\_report\_date"]

last\_annual\_report\_fy = income\_statement\_measures["last\_annual\_report\_fy"]

equity = extract\_balance\_sheet\_equity(doc, last\_annual\_report\_date, last\_annual\_report\_fy)

quarter\_of\_annual\_report = equity["quarter\_of\_annual\_report"]

years\_diff = equity["years\_diff"]

shares = extract\_shares(doc, quarter\_of\_annual\_report, years\_diff)

current\_assets = extract\_balance\_sheet\_current\_assets(doc, quarter\_of\_annual\_report, years\_diff)

non\_current\_assets = extract\_balance\_sheet\_noncurrent\_assets(doc, quarter\_of\_annual\_report, years\_diff)

debt = extract\_balance\_sheet\_debt(doc, quarter\_of\_annual\_report, years\_diff)

liabilities = extract\_balance\_sheet\_liabilities(doc, quarter\_of\_annual\_report, years\_diff, debt["mr\_debt"])

cashflow\_statement\_measures = extract\_cashflow\_statement(doc)

leases = extract\_operating\_leases(doc)

options = extract\_options(doc)

return {

\*\*shares,

\*\*income\_statement\_measures,

\*\*current\_assets,

\*\*non\_current\_assets,

\*\*debt,

\*\*liabilities,

\*\*equity,

\*\*cashflow\_statement\_measures,

\*\*leases,

\*\*options

}

def get\_selected\_years(data, key, start, end):

"""

Get the values corresponding to selected years from a dictionary {"key": {"dates":[],"values":[]}}

:param data: dictionary {"key": {"dates":[],"values":[]}}

:param key: the key of the dictionary that we want to extract the selected years

:param start: initial year

:param end: final year

:return: list of values corresponding to selected years (or 0 if year not found)

"""

r = []

for y in range(start, end + 1, 1):

try:

idx = data[key]["dates"].index(y)

r.append(data[key]["values"][idx] / 1000)

except ValueError:

r.append(0)

return r

def null\_valuation(price\_per\_share=0):

fcff\_value = div\_value = liquidation\_per\_share = -1

fcff\_delta = div\_delta = liquidation\_delta = 10

status = STATUS\_KO

return price\_per\_share, fcff\_value, div\_value, fcff\_delta, div\_delta, liquidation\_per\_share, liquidation\_delta, status

def valuation(cik, years=5, recession\_probability = 0.5, qualitative=False, debug=False):

"""

Compute valuation for company. Valuation is done following principles teached by Prof. Damodaran in his Valuation

Course (FCFF Valuation and Dividends Valuation).

We build 4 different scenarios for both FCFF and Dividends Valuation:

1. Earnings TTM & Historical Growth

2. Earnings Normalized & Historical Growth

3. Earnings TTM & Growth TTM

4. Earnings Normalized & Growth Normalized

Each scenario is also run with a recession hypothesis.

We compute a median value for FCFF, Recession FCFF, Dividends, Recession Dividends and then compute 2

Expected Values based on the recession\_probability.

These 2 values are then used to compute the final valuation (value/share) skewing the result towards the lowest

value (to be conservative).

:param cik: company cik

:param years: how many financial years to consider in the valuation

:param debug:

:return: price\_per\_share (current price/share), fcff\_value (FCFF EV), div\_value (Dividends EV),

fcff\_delta premium(discount) on shares, div\_delta premium(discount) on shares, status

(OK if company is underpriced, NI if company is correctly priced, KO is company is overpriced)

"""

# Check if we have financial data

# Check if we have submissions (at least the last 10k)

try:

download\_financial\_data(cik)

data = extract\_company\_financial\_information(cik)

except NoSharesException:

print(cik, "no shares")

return null\_valuation()

except StopIteration:

print(cik, "no financial data")

return null\_valuation()

if debug:

print(data)

print()

# Retrieve company revenues

try:

final\_year = data["revenue"]["dates"][-1]

initial\_year = final\_year - years + 1

except:

print(cik, "no revenue")

return null\_valuation()

# Calculate ERP

erp = get\_df\_from\_table("damodaran\_erp")

erp = erp[erp["date"] == erp["date"].max()]["value"].iloc[0]

# Retrieve company info

company\_info = company\_from\_cik(cik)

ticker = company\_info["ticker"]

# Get current price per share from yahoo

price\_per\_share = get\_current\_price\_from\_yahoo(ticker)

if price\_per\_share is None:

print(ticker, "delisted")

return null\_valuation()

# Get generic info

try:

company\_name, country, industry, region = get\_generic\_info(ticker)

except IndexError:

print(ticker, "not found in db")

return null\_valuation()

# Retrieve currency and financial currency from postgreSQL DB

yahoo\_equity\_ticker = get\_df\_from\_table("yahoo\_equity\_tickers", f"where symbol = '{ticker}'", most\_recent=True).iloc[0]

db\_curr = yahoo\_equity\_ticker["currency"]

db\_financial\_curr = yahoo\_equity\_ticker["financial\_currency"]

# Retrieve bond\_spread from postgreSQL DB

damodaran\_bond\_spread = get\_df\_from\_table("damodaran\_bond\_spread", most\_recent=True)

damodaran\_bond\_spread["greater\_than"] = pd.to\_numeric(damodaran\_bond\_spread["greater\_than"])

damodaran\_bond\_spread["less\_than"] = pd.to\_numeric(damodaran\_bond\_spread["less\_than"])

# Make sure to retrieve last annual report (10-K on SEC)

doc = get\_last\_document(cik, "10-K")

# Extract business segments and geographic distributions

if doc is not None:

segments = extract\_segments(doc)

geo\_segments\_df = geography\_distribution(segments, ticker)

else:

geo\_segments\_df = None

# Retrieve country statistics from postgreSQL DB

country\_stats = get\_df\_from\_table("damodaran\_country\_stats", most\_recent=True)

# Compute tax rate, country default spread anc country risk premium based on company country

tax\_rate = 0

country\_default\_spread = 0

country\_risk\_premium = 0

if geo\_segments\_df is None or geo\_segments\_df.empty:

try:

filter\_df = country\_stats[country\_stats["country"] == country.replace(" ", "")].iloc[0]

except:

filter\_df = country\_stats[country\_stats["country"] == "Global"].iloc[0]

tax\_rate = float(filter\_df["tax\_rate"])

country\_default\_spread = float(filter\_df["adjusted\_default\_spread"])

country\_risk\_premium = float(filter\_df["country\_risk\_premium"])

else:

for \_, row in geo\_segments\_df.iterrows():

percent = row["value"]

search\_key = row["country\_area"]

try:

filter\_df = country\_stats[country\_stats["country"] == search\_key.replace(" ", "")].iloc[0]

except:

filter\_df = country\_stats[country\_stats["country"] == "Global"].iloc[0]

t = float(filter\_df["tax\_rate"])

cds = float(filter\_df["adjusted\_default\_spread"])

crp = float(filter\_df["country\_risk\_premium"])

tax\_rate += t \* percent

country\_default\_spread += cds \* percent

country\_risk\_premium += crp \* percent

# Compute final ERP adding country risk premium

final\_erp = float(erp) + country\_risk\_premium

# Select alpha\_3\_code from company country

try:

alpha\_3\_code = country\_stats[country\_stats["country"] == country.replace(" ", "")].iloc[0]["alpha\_3\_code"]

except:

alpha\_3\_code = None

# Retrieve the riskfree rate based on the company financial currency and the country statistics

riskfree = currency\_bond\_yield(db\_financial\_curr, alpha\_3\_code, country\_stats)

# Check if the riskfree rate exists

if riskfree == -1:

print(ticker, "no riskfree")

return null\_valuation(price\_per\_share)

if debug:

print("===== GENERAL INFORMATION =====\n")

print("ticker", ticker)

print("cik", cik)

print("company\_name", company\_name)

print("country", country)

print("region", region)

print("industry", industry)

print("financial currency", db\_financial\_curr)

print("riskfree", riskfree)

print("erp", erp)

print("\n\n")

# Retrieve shares number

mr\_shares = data["mr\_shares"]["value"] / 1000

shares = get\_selected\_years(data, "shares", initial\_year, final\_year)

# Convert Currency

fx\_rate = None

if db\_curr is None or db\_curr.strip() == "":

return null\_valuation(price\_per\_share)

if db\_financial\_curr is None or db\_financial\_curr.strip() == "":

return null\_valuation(price\_per\_share)

# they are different

if db\_curr != db\_financial\_curr:

fx\_rate = convert\_currencies(db\_curr, db\_financial\_curr)

fx\_rate\_financial\_USD = 1

if db\_financial\_curr != "USD":

fx\_rate\_financial\_USD = convert\_currencies("USD", db\_financial\_curr)

# Retrieve financial data

ttm\_revenue = data["ttm\_revenue"]["value"] / 1000

ttm\_ebit = data["ttm\_ebit"]["value"] / 1000

ttm\_net\_income = data["ttm\_net\_income"]["value"] / 1000

ttm\_dividends = data["ttm\_dividends"]["value"] / 1000

ttm\_interest\_expense = data["ttm\_interest\_expenses"]["value"] / 1000

mr\_cash = data["mr\_cash"]["value"] / 1000

mr\_securities = data["mr\_securities"]["value"] / 1000

mr\_debt = data["mr\_debt"]["value"] / 1000

mr\_equity = data["mr\_equity"]["value"] / 1000

ebit = get\_selected\_years(data, "ebit", initial\_year, final\_year)

net\_income = get\_selected\_years(data, "net\_income", initial\_year, final\_year)

dividends = get\_selected\_years(data, "dividends", initial\_year, final\_year)

capex = get\_selected\_years(data, "capex", initial\_year, final\_year)

depreciation = get\_selected\_years(data, "depreciation", initial\_year, final\_year)

equity\_bv = get\_selected\_years(data, "equity", initial\_year, final\_year)

cash = get\_selected\_years(data, "cash", initial\_year, final\_year)

securities = get\_selected\_years(data, "securities", initial\_year, final\_year)

debt\_bv = get\_selected\_years(data, "debt", initial\_year, final\_year)

revenue = get\_selected\_years(data, "revenue", initial\_year-1, final\_year)

# Compute revenue growth

revenue\_growth = []

revenue\_delta = []

for i in range(len(revenue) - 1):

if revenue[i] < 0:

print("negative revenue")

return null\_valuation(price\_per\_share)

revenue\_delta.append(revenue[i + 1] - revenue[i])

try:

revenue\_growth.append(revenue[i + 1] / revenue[i] - 1)

except:

revenue\_growth.append(0)

# Drop 1st element we don't need

revenue = revenue[1:]

revenue\_growth = revenue\_growth[1:]

# Retrieve R&D

try:

r\_and\_d\_amortization\_years = r\_and\_d\_amortization[industry]

except:

print(f"\n#######\nCould not find industry: {industry} mapping. "

f"Check r\_and\_d\_amortization dictionary.\n#######\n")

r\_and\_d\_amortization\_years = 5

r\_and\_d = get\_selected\_years(data, "rd", final\_year - r\_and\_d\_amortization\_years, final\_year)

while len(r\_and\_d) < years:

r\_and\_d.insert(0, 0)

ebit\_r\_and\_d\_adj, tax\_benefit, r\_and\_d\_unamortized, r\_and\_d\_amortization\_cy = \

capitalize\_rd(r\_and\_d, r\_and\_d\_amortization\_years, tax\_rate, years)

# Compute R&D-adjusted values

ttm\_ebit\_adj = ttm\_ebit + ebit\_r\_and\_d\_adj[-1]

ebit\_adj = [sum(x) for x in zip(ebit, ebit\_r\_and\_d\_adj)]

ttm\_net\_income\_adj = ttm\_net\_income + ebit\_r\_and\_d\_adj[-1]

net\_income\_adj = [sum(x) for x in zip(net\_income, ebit\_r\_and\_d\_adj)]

mr\_equity\_adj = mr\_equity + r\_and\_d\_unamortized[-1]

equity\_bv\_adj = [sum(x) for x in zip(equity\_bv, r\_and\_d\_unamortized)]

capex\_adj = [sum(x) for x in zip(capex, r\_and\_d[-years:])]

depreciation\_adj = [sum(x) for x in zip(depreciation, r\_and\_d\_amortization\_cy)]

ebit\_after\_tax = [sum(x) for x in zip([x \* (1 - tax\_rate) for x in ebit\_adj], tax\_benefit)]

ttm\_eps\_adj = ttm\_net\_income\_adj / mr\_shares

# Retrieve Operating Leases

leases = [

data["mr\_op\_leases\_expense"]["value"] / 1000,

data["mr\_op\_leases\_next\_year"]["value"] / 1000,

data["mr\_op\_leases\_next\_2year"]["value"] / 1000,

data["mr\_op\_leases\_next\_3year"]["value"] / 1000,

data["mr\_op\_leases\_next\_4year"]["value"] / 1000,

data["mr\_op\_leases\_next\_5year"]["value"] / 1000,

data["mr\_op\_leases\_after\_5year"]["value"] / 1000,

]

last\_year\_leases = max([i for i, x in enumerate(leases) if x != 0], default=-1)

if last\_year\_leases != -1:

ebit\_op\_adj, int\_exp\_op\_adj, debt\_adj, tax\_benefit\_op, company\_default\_spread = \

debtize\_op\_leases(ttm\_interest\_expense, ttm\_ebit\_adj, damodaran\_bond\_spread, riskfree, country\_default\_spread,

leases, last\_year\_leases, tax\_rate, revenue\_growth)

# Compute OperatingLeases-adjusted values

ttm\_ebit\_adj += ebit\_op\_adj[-1]

ttm\_interest\_expense\_adj = ttm\_interest\_expense + int\_exp\_op\_adj

mr\_debt\_adj = mr\_debt + debt\_adj[-1]

ebit\_adj = [sum(x) for x in zip(ebit\_adj, ebit\_op\_adj)]

debt\_bv\_adj = [sum(x) for x in zip(debt\_bv, debt\_adj)]

ebit\_after\_tax = [sum(x) for x in zip(ebit\_after\_tax, tax\_benefit\_op)]

ttm\_ebit\_after\_tax = ttm\_ebit\_adj \* (1 - tax\_rate) + tax\_benefit[-1] + tax\_benefit\_op[-1]

# no leases

else:

ttm\_interest\_expense\_adj = ttm\_interest\_expense

mr\_debt\_adj = mr\_debt

debt\_bv\_adj = debt\_bv

company\_default\_spread = get\_spread\_from\_dscr(12.5, damodaran\_bond\_spread)

ttm\_ebit\_after\_tax = ttm\_ebit\_adj \* (1 - tax\_rate) + tax\_benefit[-1]

# Compute cost of debt

cost\_of\_debt = riskfree + country\_default\_spread + company\_default\_spread

# Compute cash and securities

mr\_cash\_and\_securities = mr\_cash + mr\_securities

cash\_and\_securities = [sum(x) for x in zip(cash, securities)]

# Consider EPS/dividends as with most recent number of shares (to account for splits and buybacks)

eps = [x / mr\_shares for x in net\_income]

eps\_adj = [x/mr\_shares for x in net\_income\_adj]

dividends = [x/mr\_shares for x in dividends]

# Compute Working Capital as inventory + receivables + other assets - payables - due to affiliates - due to related

wc = {}

for i in ["inventory", "receivables", "other\_assets", "account\_payable", "due\_to\_affiliates", "due\_to\_related\_parties"]:

val = get\_selected\_years(data, i, initial\_year-1, final\_year)

wc[i] = val

df = pd.DataFrame(wc)

df["wc"] = df["inventory"] + df["receivables"] + df["other\_assets"] - df["account\_payable"] \

- df["due\_to\_affiliates"] - df["due\_to\_related\_parties"]

df["delta\_wc"] = df["wc"].diff(1)

df = df.dropna()

working\_capital = df["wc"].to\_list()

delta\_wc = df["delta\_wc"].to\_list()

# Compute reinvestments

reinvestment = []

for i in range(len(capex)):

reinvestment.append(capex\_adj[i] + delta\_wc[i] - depreciation\_adj[i])

# Compute equity market value (market cap)

equity\_mkt = mr\_shares \* price\_per\_share

if fx\_rate is not None:

equity\_mkt /= fx\_rate

# Compute debt market value

debt\_mkt = ttm\_interest\_expense\_adj \* (1 - (1 + cost\_of\_debt) \*\* -6) / cost\_of\_debt + mr\_debt\_adj / (

1 + cost\_of\_debt) \*\* 6

# Get company industry data

target\_sales\_capital, industry\_payout, pbv, unlevered\_beta, target\_operating\_margin, target\_debt\_equity = \

get\_industry\_data(industry, region, geo\_segments\_df, revenue, ebit\_adj, revenue\_delta, reinvestment,

equity\_mkt, debt\_mkt, equity\_bv\_adj, debt\_bv\_adj, mr\_equity\_adj, mr\_debt\_adj)

# Retrieve minority interest

mr\_original\_min\_interest = data["mr\_minority\_interest"]["value"] / 1000

mr\_minority\_interest = mr\_original\_min\_interest \* pbv

# Retrieve tax benefits and Share Based Compensation

mr\_tax\_benefits = data["mr\_tax\_benefits"]["value"] / 1000

mr\_sbc = data["mr\_sbc"]["value"] / 1000

if debug:

print("===== Last Available Data =====\n")

print("Outstanding Shares", mr\_shares)

print("Price/Share (price currency)", price\_per\_share)

print("FX Rate:", 1 if fx\_rate is None else fx\_rate)

print("FX Rate USD:", fx\_rate\_financial\_USD)

print("ttm\_revenue", ttm\_revenue)

print("ttm\_ebit", ttm\_ebit, "=>", ttm\_ebit\_adj)

print("ttm\_net\_income", ttm\_net\_income, "=>", ttm\_net\_income\_adj)

print("ttm\_dividends", ttm\_dividends)

# print("ttm\_eps", ttm\_eps, "=>", ttm\_eps\_adj)

print("ttm\_interest\_expense", ttm\_interest\_expense, "=>", ttm\_interest\_expense\_adj)

print("tax\_credit", mr\_tax\_benefits)

# print("minority\_interest", mr\_original\_min\_interest, "=>", mr\_minority\_interest)

# print("cash&securities", mr\_cash\_and\_securities)

# print("BV of debt", mr\_debt, "=>", mr\_debt\_adj)

# print("BV of equity", mr\_equity, "=>", mr\_equity\_adj)

print("\n\n")

print("===== Historical Data =====\n")

print("initial\_year", initial\_year)

print("revenue", revenue)

print("revenue\_delta", revenue\_delta)

print("ebit", ebit, "=>", ebit\_adj)

# print("ebit\_after\_tax\_adj", ebit\_after\_tax)

print("net\_income", net\_income, "=>", net\_income\_adj)

# print("eps", eps, "=>", eps\_adj)

print("dividends", dividends)

print("working\_capital", working\_capital)

print("delta\_WC", delta\_wc)

print("capex", capex, "=>", capex\_adj)

print("depreciation", depreciation, "=>", depreciation\_adj)

print("shares\_outstanding", shares)

print("equity\_bv", equity\_bv, "=>", equity\_bv\_adj)

print("cash&securities", cash\_and\_securities)

print("debt\_bv", debt\_bv, "=>", debt\_bv\_adj)

print("\n\n")

print("===== R&D =====")

print("r\_and\_d", r\_and\_d)

print("amortization\_years", r\_and\_d\_amortization\_years)

print("\n===== Operating Leases =====")

print("leases", leases)

print("\n===== Segments =====\n")

if geo\_segments\_df is None:

print("10-K not found. Check annual report on company website.")

else:

print(geo\_segments\_df.to\_markdown())

print("\n===== Options =====")

print("mr\_sbc", mr\_sbc)

print("\n\n")

# Compute get\_growth\_ttm

roc\_last, reinvestment\_last, growth\_last, roe\_last, reinvestment\_eps\_last, growth\_eps\_last = \

get\_growth\_ttm(ttm\_ebit\_after\_tax, ttm\_net\_income\_adj, mr\_equity\_adj, mr\_debt\_adj, mr\_cash\_and\_securities,

reinvestment, ttm\_dividends, industry\_payout)

# Get ROE and ROC

roe, roc = get\_roe\_roc(equity\_bv\_adj, debt\_bv\_adj, cash\_and\_securities, ebit\_after\_tax, net\_income\_adj)

# Compute Target info

cagr, target\_levered\_beta, target\_cost\_of\_equity, target\_cost\_of\_debt, target\_cost\_of\_capital = \

get\_target\_info(revenue, ttm\_revenue, country\_default\_spread, tax\_rate, final\_erp, riskfree,

unlevered\_beta, damodaran\_bond\_spread, company\_default\_spread, target\_debt\_equity)

# Normalize info

revenue\_5y, ebit\_5y, operating\_margin\_5y, sales\_capital\_5y, roc\_5y, reinvestment\_5y, growth\_5y, \

net\_income\_5y, roe\_5y, reinvestment\_eps\_5y, growth\_eps\_5y = \

get\_normalized\_info(revenue, ebit\_adj, revenue\_delta, reinvestment, target\_sales\_capital,

ebit\_after\_tax, industry\_payout, cagr, net\_income\_adj, roe, dividends, eps\_adj, roc)

# Get dividends info

eps\_5y, payout\_5y = get\_dividends\_info(eps\_adj, dividends)

# Get final info

survival\_prob, debt\_equity, \

levered\_beta, cost\_of\_equity, equity\_weight, debt\_weight, cost\_of\_capital = \

get\_final\_info(riskfree, cost\_of\_debt, equity\_mkt, debt\_mkt, unlevered\_beta,

tax\_rate, final\_erp, company\_default\_spread)

# Select final data

mr\_receivables = data["mr\_receivables"]["value"] / 1000

mr\_inventory = data["mr\_inventory"]["value"] / 1000

mr\_other\_current\_assets = data["mr\_other\_assets"]["value"] / 1000

mr\_ppe = data["mr\_ppe"]["value"] / 1000

mr\_property = data["mr\_investment\_property"]["value"] / 1000

mr\_equity\_investments = data["mr\_equity\_investments"]["value"] / 1000

mr\_total\_liabilities = data["mr\_liabilities"]["value"] / 1000

# Compute liquidation value

try:

liquidation\_value = calculate\_liquidation\_value(mr\_cash, mr\_receivables, mr\_inventory, mr\_securities,

mr\_other\_current\_assets, mr\_property,

mr\_ppe, mr\_equity\_investments, mr\_total\_liabilities, equity\_mkt,

mr\_debt, mr\_equity, mr\_original\_min\_interest,

mr\_minority\_interest, debug=debug)

except:

print(traceback.format\_exc())

liquidation\_value = 0

# Compute liquidation per share

liquidation\_per\_share = liquidation\_value / mr\_shares

if debug:

print("===== Growth =====\n")

print("cagr", round(cagr,4))

print("riskfree", round(riskfree,4))

print("\n\n")

print("===== Model Helper Calculation =====\n")

print("roc\_last", round(roc\_last,4))

print("reinvestment\_last", round(reinvestment\_last,4))

print("growth\_last", round(growth\_last,4))

print("ROC history", roc)

print("roc\_5y", round(roc\_5y,4))

print("Reinvestment history", reinvestment)

print("reinvestment\_5y", round(reinvestment\_5y,4))

print("growth\_5y", round(growth\_5y,4))

print("revenue\_5y", revenue\_5y)

print("ebit\_5y", ebit\_5y)

print("roe\_last", round(roe\_last,4))

print("reinvestment\_eps\_last", round(reinvestment\_eps\_last,4))

print("growth\_eps\_last", round(growth\_eps\_last,4))

print("sales\_capital\_5y", round(sales\_capital\_5y,4))

print("roe\_5y", round(roe\_5y,4))

print("reinvestment\_eps\_5y", round(reinvestment\_eps\_5y,4))

print("growth\_eps\_5y", round(growth\_eps\_5y,4))

print("eps\_5y", round(eps\_5y,4))

print("payout\_5y", round(payout\_5y,4))

print("industry\_payout", round(industry\_payout,4))

print("target\_sales\_capital", round(target\_sales\_capital,4))

print("\n\n")

print("===== Recap Info =====\n")

print("country\_default\_spread", round(country\_default\_spread,4))

print("country\_risk\_premium", round(country\_risk\_premium,4))

print("riskfree", round(riskfree,4))

print("final\_erp", round(final\_erp,4))

print("unlevered\_beta", round(unlevered\_beta,4))

print("tax\_rate", round(tax\_rate,4))

print("levered\_beta", round(levered\_beta,4))

print("cost\_of\_equity", round(cost\_of\_equity,4))

print("cost\_of\_debt", round(cost\_of\_debt,4))

print("equity\_weight", round(equity\_weight,4))

print("debt\_weight", round(debt\_weight,4))

print("cost\_of\_capital", round(cost\_of\_capital,4))

print("equity\_mkt", round(equity\_mkt,2))

print("debt\_mkt", round(debt\_mkt,2))

print("debt\_equity", round(debt\_equity,4))

print("equity\_bv\_adj", round(mr\_equity\_adj,2))

print("debt\_bv\_adj", round(mr\_debt\_adj,2))

print("ebit\_adj", round(ttm\_ebit\_adj,2))

print("company\_default\_spread", round(company\_default\_spread,4))

print("survival\_prob", round(survival\_prob,4))

print("liquidation value", round(liquidation\_value, 2))

print("\n\n")

print("===== Other Model inputs =====\n")

print("operating\_margin\_5y", round(operating\_margin\_5y,4))

print("target\_operating\_margin", round(target\_operating\_margin,4))

print("target\_debt\_equity", round(target\_debt\_equity,4))

print("target\_levered\_beta", round(target\_levered\_beta,4))

print("target\_cost\_of\_equity", round(target\_cost\_of\_equity,4))

print("target\_cost\_of\_debt", round(target\_cost\_of\_debt,4))

print("target\_cost\_of\_capital", round(target\_cost\_of\_capital,4))

print("\n\n")

# Perform valuations

dict\_values\_for\_bi = {}

stock\_value\_div\_ttm\_fixed = dividends\_valuation(EARNINGS\_TTM, GROWTH\_FIXED, cagr, growth\_eps\_5y, growth\_5y,

riskfree, industry\_payout, cost\_of\_equity,

target\_cost\_of\_equity, growth\_eps\_last, eps\_5y, payout\_5y, ttm\_eps\_adj,

reinvestment\_eps\_last, fx\_rate, survival\_prob, liquidation\_per\_share, debug=debug, dict\_values\_for\_bi=dict\_values\_for\_bi)

stock\_value\_div\_norm\_fixed = dividends\_valuation(EARNINGS\_NORM, GROWTH\_FIXED, cagr, growth\_eps\_5y, growth\_5y,

riskfree, industry\_payout, cost\_of\_equity,

target\_cost\_of\_equity, growth\_eps\_last, eps\_5y, payout\_5y, ttm\_eps\_adj,

reinvestment\_eps\_last, fx\_rate, survival\_prob, liquidation\_per\_share, debug=debug, dict\_values\_for\_bi=dict\_values\_for\_bi)

stock\_value\_div\_ttm\_ttm = dividends\_valuation(EARNINGS\_TTM, GROWTH\_TTM, cagr, growth\_eps\_5y, growth\_5y, riskfree,

industry\_payout, cost\_of\_equity, target\_cost\_of\_equity,

growth\_eps\_last, eps\_5y, payout\_5y, ttm\_eps\_adj,

reinvestment\_eps\_last, fx\_rate, survival\_prob, liquidation\_per\_share, debug=debug, dict\_values\_for\_bi=dict\_values\_for\_bi)

stock\_value\_div\_norm\_norm = dividends\_valuation(EARNINGS\_NORM, GROWTH\_NORM, cagr, growth\_eps\_5y, growth\_5y, riskfree,

industry\_payout, cost\_of\_equity,

target\_cost\_of\_equity, growth\_eps\_last, eps\_5y, payout\_5y, ttm\_eps\_adj,

reinvestment\_eps\_last, fx\_rate, survival\_prob, liquidation\_per\_share, debug=debug, dict\_values\_for\_bi=dict\_values\_for\_bi)

stock\_value\_div\_ttm\_fixed\_recession = dividends\_valuation(EARNINGS\_TTM, GROWTH\_FIXED, cagr, growth\_eps\_5y, growth\_5y,

riskfree, industry\_payout, cost\_of\_equity,

target\_cost\_of\_equity, growth\_eps\_last, eps\_5y, payout\_5y, ttm\_eps\_adj,

reinvestment\_eps\_last, fx\_rate, survival\_prob, liquidation\_per\_share, debug=debug, recession=True, dict\_values\_for\_bi=dict\_values\_for\_bi)

stock\_value\_div\_norm\_fixed\_recession = dividends\_valuation(EARNINGS\_NORM, GROWTH\_FIXED, cagr, growth\_eps\_5y, growth\_5y,

riskfree, industry\_payout, cost\_of\_equity,

target\_cost\_of\_equity, growth\_eps\_last, eps\_5y, payout\_5y, ttm\_eps\_adj,

reinvestment\_eps\_last, fx\_rate, survival\_prob, liquidation\_per\_share, debug=debug, recession=True, dict\_values\_for\_bi=dict\_values\_for\_bi)

stock\_value\_div\_ttm\_ttm\_recession = dividends\_valuation(EARNINGS\_TTM, GROWTH\_TTM, cagr, growth\_eps\_5y, growth\_5y, riskfree,

industry\_payout, cost\_of\_equity, target\_cost\_of\_equity,

growth\_eps\_last, eps\_5y, payout\_5y, ttm\_eps\_adj,

reinvestment\_eps\_last, fx\_rate, survival\_prob, liquidation\_per\_share, debug=debug, recession=True, dict\_values\_for\_bi=dict\_values\_for\_bi)

stock\_value\_div\_norm\_norm\_recession = dividends\_valuation(EARNINGS\_NORM, GROWTH\_NORM, cagr, growth\_eps\_5y, growth\_5y, riskfree,

industry\_payout, cost\_of\_equity,

target\_cost\_of\_equity, growth\_eps\_last, eps\_5y, payout\_5y, ttm\_eps\_adj,

reinvestment\_eps\_last, fx\_rate, survival\_prob, liquidation\_per\_share, debug=debug, recession=True, dict\_values\_for\_bi=dict\_values\_for\_bi)

stock\_value\_fcff\_ttm\_fixed = fcff\_valuation(EARNINGS\_TTM, GROWTH\_FIXED, cagr, riskfree, ttm\_revenue, ttm\_ebit\_adj,

target\_operating\_margin, mr\_tax\_benefits, tax\_rate, sales\_capital\_5y, target\_sales\_capital,

debt\_equity, target\_debt\_equity, unlevered\_beta, final\_erp, cost\_of\_debt,

target\_cost\_of\_debt, mr\_cash, mr\_securities, debt\_mkt, mr\_minority\_interest, survival\_prob, mr\_shares,

liquidation\_value, growth\_last, growth\_5y, revenue\_5y, ebit\_5y, fx\_rate, mr\_property, mr\_sbc, debug=debug, dict\_values\_for\_bi=dict\_values\_for\_bi)

stock\_value\_fcff\_norm\_fixed = fcff\_valuation(EARNINGS\_NORM, GROWTH\_FIXED, cagr, riskfree, ttm\_revenue, ttm\_ebit\_adj,

target\_operating\_margin, mr\_tax\_benefits, tax\_rate, sales\_capital\_5y, target\_sales\_capital,

debt\_equity, target\_debt\_equity, unlevered\_beta, final\_erp, cost\_of\_debt,

target\_cost\_of\_debt, mr\_cash, mr\_securities, debt\_mkt, mr\_minority\_interest, survival\_prob, mr\_shares,

liquidation\_value, growth\_last, growth\_5y, revenue\_5y, ebit\_5y, fx\_rate, mr\_property, mr\_sbc, debug=debug, dict\_values\_for\_bi=dict\_values\_for\_bi)

stock\_value\_fcff\_ttm\_ttm = fcff\_valuation(EARNINGS\_TTM, GROWTH\_TTM, cagr, riskfree, ttm\_revenue, ttm\_ebit\_adj,

target\_operating\_margin, mr\_tax\_benefits, tax\_rate, sales\_capital\_5y, target\_sales\_capital,

debt\_equity, target\_debt\_equity, unlevered\_beta, final\_erp, cost\_of\_debt,

target\_cost\_of\_debt, mr\_cash, mr\_securities, debt\_mkt, mr\_minority\_interest, survival\_prob, mr\_shares,

liquidation\_value, growth\_last, growth\_5y, revenue\_5y, ebit\_5y, fx\_rate, mr\_property, mr\_sbc, debug=debug, dict\_values\_for\_bi=dict\_values\_for\_bi)

stock\_value\_fcff\_norm\_norm = fcff\_valuation(EARNINGS\_NORM, GROWTH\_NORM, cagr, riskfree, ttm\_revenue, ttm\_ebit\_adj,

target\_operating\_margin, mr\_tax\_benefits, tax\_rate, sales\_capital\_5y, target\_sales\_capital,

debt\_equity, target\_debt\_equity, unlevered\_beta, final\_erp, cost\_of\_debt,

target\_cost\_of\_debt, mr\_cash, mr\_securities, debt\_mkt, mr\_minority\_interest, survival\_prob, mr\_shares,

liquidation\_value, growth\_last, growth\_5y, revenue\_5y, ebit\_5y, fx\_rate, mr\_property, mr\_sbc, debug=debug, dict\_values\_for\_bi=dict\_values\_for\_bi)

stock\_value\_fcff\_ttm\_fixed\_recession = fcff\_valuation(EARNINGS\_TTM, GROWTH\_FIXED, cagr, riskfree, ttm\_revenue, ttm\_ebit\_adj,

target\_operating\_margin, mr\_tax\_benefits, tax\_rate, sales\_capital\_5y, target\_sales\_capital,

debt\_equity, target\_debt\_equity, unlevered\_beta, final\_erp, cost\_of\_debt,

target\_cost\_of\_debt, mr\_cash, mr\_securities, debt\_mkt, mr\_minority\_interest, survival\_prob, mr\_shares,

liquidation\_value, growth\_last, growth\_5y, revenue\_5y, ebit\_5y, fx\_rate, mr\_property, mr\_sbc, debug=debug, recession=True, dict\_values\_for\_bi=dict\_values\_for\_bi)

stock\_value\_fcff\_norm\_fixed\_recession = fcff\_valuation(EARNINGS\_NORM, GROWTH\_FIXED, cagr, riskfree, ttm\_revenue, ttm\_ebit\_adj,

target\_operating\_margin, mr\_tax\_benefits, tax\_rate, sales\_capital\_5y, target\_sales\_capital,

debt\_equity, target\_debt\_equity, unlevered\_beta, final\_erp, cost\_of\_debt,

target\_cost\_of\_debt, mr\_cash, mr\_securities, debt\_mkt, mr\_minority\_interest, survival\_prob, mr\_shares,

liquidation\_value, growth\_last, growth\_5y, revenue\_5y, ebit\_5y, fx\_rate, mr\_property, mr\_sbc, debug=debug, recession=True, dict\_values\_for\_bi=dict\_values\_for\_bi)

stock\_value\_fcff\_ttm\_ttm\_recession = fcff\_valuation(EARNINGS\_TTM, GROWTH\_TTM, cagr, riskfree, ttm\_revenue, ttm\_ebit\_adj,

target\_operating\_margin, mr\_tax\_benefits, tax\_rate, sales\_capital\_5y, target\_sales\_capital,

debt\_equity, target\_debt\_equity, unlevered\_beta, final\_erp, cost\_of\_debt,

target\_cost\_of\_debt, mr\_cash, mr\_securities, debt\_mkt, mr\_minority\_interest, survival\_prob, mr\_shares,

liquidation\_value, growth\_last, growth\_5y, revenue\_5y, ebit\_5y, fx\_rate, mr\_property, mr\_sbc, debug=debug, recession=True, dict\_values\_for\_bi=dict\_values\_for\_bi)

stock\_value\_fcff\_norm\_norm\_recession = fcff\_valuation(EARNINGS\_NORM, GROWTH\_NORM, cagr, riskfree, ttm\_revenue, ttm\_ebit\_adj,

target\_operating\_margin, mr\_tax\_benefits, tax\_rate, sales\_capital\_5y, target\_sales\_capital,

debt\_equity, target\_debt\_equity, unlevered\_beta, final\_erp, cost\_of\_debt,

target\_cost\_of\_debt, mr\_cash, mr\_securities, debt\_mkt, mr\_minority\_interest, survival\_prob, mr\_shares,

liquidation\_value, growth\_last, growth\_5y, revenue\_5y, ebit\_5y, fx\_rate, mr\_property, mr\_sbc, debug=debug, recession=True, dict\_values\_for\_bi=dict\_values\_for\_bi)

# Aggregate valuation

fcff\_values\_list = [stock\_value\_fcff\_ttm\_fixed, stock\_value\_fcff\_norm\_fixed, stock\_value\_fcff\_ttm\_ttm,

stock\_value\_fcff\_norm\_norm]

fcff\_recession\_values\_list = [stock\_value\_fcff\_ttm\_fixed\_recession, stock\_value\_fcff\_norm\_fixed\_recession,

stock\_value\_fcff\_ttm\_ttm\_recession, stock\_value\_fcff\_norm\_norm\_recession]

div\_values\_list = [stock\_value\_div\_ttm\_fixed, stock\_value\_div\_norm\_fixed, stock\_value\_div\_ttm\_ttm,

stock\_value\_div\_norm\_norm]

div\_recession\_values\_list = [stock\_value\_div\_ttm\_fixed\_recession, stock\_value\_div\_norm\_fixed\_recession,

stock\_value\_div\_ttm\_ttm\_recession, stock\_value\_div\_norm\_norm\_recession]

# Summarize valuations

fcff\_value = summary\_valuation(fcff\_values\_list)

fcff\_recession\_value = summary\_valuation(fcff\_recession\_values\_list)

ev\_fcff = fcff\_value \* (1 - recession\_probability) + fcff\_recession\_value \* recession\_probability

div\_value = summary\_valuation(div\_values\_list)

div\_recession\_value = summary\_valuation(div\_recession\_values\_list)

ev\_dividends = div\_value \* (1 - recession\_probability) + div\_recession\_value \* recession\_probability

if fx\_rate is not None:

fcff\_value \*= fx\_rate

div\_value \*= fx\_rate

liquidation\_per\_share \*= fx\_rate

# Compute valuation delta

fcff\_delta = price\_per\_share / ev\_fcff - 1 if fcff\_value > 0 else 10

div\_delta = price\_per\_share / ev\_dividends - 1 if div\_value > 0 else 10

liquidation\_delta = price\_per\_share / liquidation\_per\_share - 1 if liquidation\_per\_share > 0 else 10

# Compute company size

market\_cap\_USD = equity\_mkt \* fx\_rate\_financial\_USD

if market\_cap\_USD < 50 \* 10 \*\* 3:

company\_size = "Nano"

elif market\_cap\_USD < 300 \* 10 \*\* 3:

company\_size = "Micro"

elif market\_cap\_USD < 2 \* 10 \*\* 6:

company\_size = "Small"

elif market\_cap\_USD < 10 \* 10 \*\* 6:

company\_size = "Medium"

elif market\_cap\_USD < 200 \* 10 \*\* 6:

company\_size = "Large"

else:

company\_size = "Mega"

# Compute company complexity

complexity = company\_complexity(doc, industry, company\_size)

# Compute company share diluition

dilution = company\_share\_diluition(shares)

# Retrieve inventory and account receivables

inventory = get\_selected\_years(data, "inventory", initial\_year-1, final\_year)

receivables = get\_selected\_years(data, "receivables", initial\_year-1, final\_year)

company\_type = get\_company\_type(revenue\_growth, mr\_debt\_adj, equity\_mkt, liquidation\_value, operating\_margin\_5y, industry)

# Retrieve auditor

auditor = find\_auditor(doc)

if debug:

print("===== Risk Assessment =====\n")

print("MKT CAP USD: ", market\_cap\_USD)

print("company\_size", company\_size)

print("company complexity", complexity)

print("share dilution", round(dilution, 4))

print("revenue", revenue)

print("inventory", inventory)

print("receivables", receivables)

print("company\_type", company\_type)

print("Auditor", auditor)

print()

# Compute status

status = get\_status(fcff\_delta, div\_delta, liquidation\_delta, country, region, company\_size, company\_type, dilution, complexity,

revenue, receivables, inventory, debug)

if debug:

print("FCFF values")

print([round(x, 2) for x in fcff\_values\_list])

print("\nFCFF values w/ Recession")

print([round(x, 2) for x in fcff\_recession\_values\_list])

print("\n\nDiv values")

print([round(x, 2) for x in div\_values\_list])

print("\nDiv values w/ Recession")

print([round(x, 2) for x in div\_recession\_values\_list])

print("\n\n\n")

print("Price per Share", price\_per\_share)

print("FCFF Result", ev\_fcff)

print("FCFF Deviation", fcff\_delta)

print("Dividends Result", ev\_dividends)

print("Dividends Deviation", div\_delta)

print("Status", status)

# Add qualitative analysis

if qualitative and doc is not None:

l = []

recent\_docs = get\_recent\_docs(cik, doc["filing\_date"])

for d in recent\_docs:

print("##############")

print(d["form\_type"], d["filing\_date"], d["\_id"])

print("##############\n")

if not mongodb.check\_document\_exists("parsed\_documents", d["\_id"]):

parse\_document(d)

parsed\_doc = mongodb.get\_document("parsed\_documents", d["\_id"])

if not mongodb.check\_document\_exists("items\_summary", d["\_id"]):

sections\_summary(parsed\_doc)

summary\_doc = mongodb.get\_document("items\_summary", d["\_id"])

for k, v in summary\_doc.items():

if isinstance(v, dict):

print(f"=== {k} ===")

for info in v["summary"]:

print(info)

if v["links"] is None:

v["links"] = [{"title":"", "link":""}]

for link in v["links"]:

l.append({

"ticker": ticker,

"created\_at": datetime.datetime.now().date(),

"form\_type": d["form\_type"],

"filing\_date": d["filing\_date"],

"url": d["\_id"],

"section": k,

"information": info,

"section\_link": link["link"],

"section\_link\_title": link["title"]

})

print()

print("\n")

summary\_df = pd.DataFrame(l)

else:

summary\_df = None

#### DFs for BI ####

company\_info\_df = pd.DataFrame([

{

"ticker": ticker,

"company\_name": company\_name,

"country": country,

"region": region,

"industry": industry,

"financial\_currency": db\_financial\_curr,

"quote\_currency": db\_curr,

"price\_per\_share": price\_per\_share,

"number\_of\_shares": mr\_shares \* 1000,

"market\_cap": equity\_mkt \* 1000,

"options\_value": mr\_sbc \* 1000,

"equity": mr\_equity \* 1000,

"equity\_adj": mr\_equity\_adj \* 1000,

"cash": mr\_cash \* 1000,

"cash\_and\_securities": mr\_cash\_and\_securities \* 1000,

"debt": mr\_debt \* 1000,

"debt\_adj": mr\_debt\_adj \* 1000,

"minority\_interest": mr\_original\_min\_interest \* 1000,

"minority\_interest\_adj": mr\_minority\_interest \* 1000,

"survival\_rate": survival\_prob,

"liquidation\_value": liquidation\_value \* 1000,

"created\_at": datetime.datetime.now().date()

}

])

financial\_data = []

for i in range(years):

financial\_data.append({

"ticker": ticker,

"created\_at": datetime.datetime.now().date(),

"scenario": "actual",

"year": initial\_year + i,

"revenue": revenue[i] \* 1000,

"ebit": ebit[i] \* 1000,

"ebit\_adj": ebit\_adj[i] \* 1000,

"ebit\_after\_tax": ebit\_after\_tax[i] \* 1000,

"reinvestment": reinvestment[i] \* 1000,

"fcff": (ebit\_after\_tax[i] - reinvestment[i]) \* 1000,

"eps": eps[i],

"eps\_adj": eps\_adj[i],

"dividends\_per\_share": dividends[i],

"dividends": dividends[i] \* mr\_shares \* 1000

})

for scenario in dict\_values\_for\_bi:

scenario\_values = dict\_values\_for\_bi[scenario]

for i in range(11):

financial\_data.append({

"ticker": ticker,

"created\_at": datetime.datetime.now().date(),

"scenario": scenario,

"year": final\_year + i + 1,

"revenue": scenario\_values["revenue"][i] \* 1000,

"ebit": scenario\_values["ebit"][i] \* 1000,

"ebit\_after\_tax": scenario\_values["ebit\_after\_tax"][i] \* 1000,

"reinvestment": scenario\_values["reinvestment"][i] \* 1000,

"fcff": scenario\_values["FCFF"][i] \* 1000,

"cost\_of\_capital": scenario\_values["cost\_of\_capital"][i],

"pv\_of\_fcff": scenario\_values["pv\_of\_FCFF"][i] \* 1000,

"eps": scenario\_values["eps"][i],

"dividends\_per\_share": scenario\_values["dividends"][i],

"dividends": scenario\_values["dividends"][i] \* mr\_shares \* 1000,

"cost\_of\_equity": scenario\_values["cost\_of\_equity"][i],

"pv\_of\_dividends\_per\_share": scenario\_values["pv\_of\_dividends"][i],

})

financial\_data\_df = pd.DataFrame(financial\_data)

if geo\_segments\_df is None or geo\_segments\_df.empty:

geo\_segments\_df = pd.DataFrame([

{"value": 1,

"country": country,

"country\_area": country,

"region": region

}

])

return price\_per\_share, fcff\_value, div\_value, fcff\_delta, div\_delta, liquidation\_per\_share, liquidation\_delta, \

status, company\_info\_df, financial\_data\_df, geo\_segments\_df, summary\_df

if \_\_name\_\_ == '\_\_main\_\_':

cik = cik\_from\_ticker("BLDR")

if cik != -1:

valuation(cik, debug=True, years=6)

from typing import Any, List

from langchain.text\_splitter import RecursiveCharacterTextSplitter

from langchain.document\_loaders.unstructured import UnstructuredBaseLoader

class UnstructuredStringLoader(UnstructuredBaseLoader):

"""

Uses unstructured to load a string

Source of the string, for metadata purposes, can be passed in by the caller

"""

def \_\_init\_\_(

self, content: str, source: str = None, mode: str = "single",

\*\*unstructured\_kwargs: Any

):

self.content = content

self.source = source

super().\_\_init\_\_(mode=mode, \*\*unstructured\_kwargs)

def \_get\_elements(self) -> List:

from unstructured.partition.text import partition\_text

return partition\_text(text=self.content, \*\*self.unstructured\_kwargs)

def \_get\_metadata(self) -> dict:

return {"source": self.source} if self.source else {}

def split\_text\_in\_chunks(text, chunk\_size=20000):

text\_splitter = RecursiveCharacterTextSplitter(chunk\_size=chunk\_size, chunk\_overlap=100)

chunks = text\_splitter.split\_text(text)

return chunks

def split\_doc\_in\_chunks(doc, chunk\_size=20000):

text\_splitter = RecursiveCharacterTextSplitter(chunk\_size=chunk\_size, chunk\_overlap=100)

chunks = text\_splitter.split\_documents(doc)

return chunks

def doc\_summary(docs):

print(f'You have {len(docs)} document(s)')

num\_words = sum([len(doc.page\_content.split(' ')) for doc in docs])

print(f'You have roughly {num\_words} words in your docs')

print()

print(f'Preview: \n{docs[0].page\_content.split(". ")[0]}')

# if \_\_name\_\_ == '\_\_main\_\_':

# import mongodb

# from edgar\_utils import company\_from\_cik

#

# parser = ConfigParser()

# \_ = parser.read(os.path.join("credentials.cfg"))

# model = "gpt-3.5-turbo"

# llm = ChatOpenAI(model\_name=model, openai\_api\_key=parser.get("open\_ai", "api\_key"))

#

# url = 'https://www.sec.gov/Archives/edgar/data/8818/000000881823000002/avy-20221231.htm'

# doc = mongodb.get\_document("documents", url)

# parsed\_doc = mongodb.get\_document("parsed\_documents", url)

# company = company\_from\_cik(doc["cik"])

#

# result = {"\_id": doc["\_id"],

# "name": company["name"],

# "ticker": company["ticker"],

# "form\_type": doc["form\_type"],

# "filing\_date": doc["filing\_date"]}

#

# for section\_title, section\_text in parsed\_doc.items():

#

# # if no section to summarize, skip

# if section\_title == "\_id" or len(section\_text) == 0:

# continue

# # chunks = split\_in\_chunks(section\_text)

#

# string\_loader = UnstructuredStringLoader(section\_text)

# doc = string\_loader.load()

# doc\_summary(doc)

# docs = split\_doc\_in\_chunks(doc)

# doc\_summary(docs)

#

# chain = load\_summarize\_chain(llm, chain\_type="refine", verbose=True)

# res = chain.run(docs)

# print(res)

import copy

import json

import os

import re

import time

import traceback

from datetime import datetime

import Levenshtein as Levenshtein

from bs4 import BeautifulSoup, NavigableString

from unidecode import unidecode

import mongodb

from edgar\_utils import company\_from\_cik, AAPL\_CIK, download\_submissions\_documents, download\_all\_cik\_submissions

import string

from openai\_interface import summarize\_section

list\_10k\_items = [

"business",

"risk factors",

"unresolved staff comments",

"properties",

"legal proceedings",

"mine safety disclosures",

"market for registrant’s common equity, related stockholder matters and issuer purchases of equity securities",

"reserved",

"management’s discussion and analysis of financial condition and results of operations",

"quantitative and qualitative disclosures about market risk",

"financial statements and supplementary data",

"changes in and disagreements with accountants on accounting and financial disclosure",

"controls and procedures",

"other information",

"disclosure regarding foreign jurisdictions that prevent inspection",

"directors, executive officers, and corporate governance",

"executive compensation",

"security ownership of certain beneficial owners and management and related stockholder matters",

"certain relationships and related transactions, and director independence",

"principal accountant fees and services",

"exhibits and financial statement schedules",

]

default\_10k\_sections = {

1: {'item': 'item 1', 'title': ['business']},

2: {'item': 'item 1a', 'title': ['risk factor']},

3: {'item': 'item 1b', 'title': ['unresolved staff']},

4: {'item': 'item 2', 'title': ['propert']},

5: {'item': 'item 3', 'title': ['legal proceeding']},

6: {'item': 'item 4', 'title': ['mine safety disclosure', 'submission of matters to a vote of security holders']},

7: {'item': 'item 5', 'title': ["market for registrant's common equity, related stockholder matters and issuer purchases of equity securities"]},

8: {'item': 'item 6', 'title': ['reserved', 'selected financial data']},

9: {'item': 'item 7', 'title': ["management's discussion and analysis of financial condition and results of operations"]},

10: {'item': 'item 7a', 'title': ['quantitative and qualitative disclosures about market risk']},

11: {'item': 'item 8', 'title': ['financial statements and supplementary data']},

12: {'item': 'item 9', 'title': ['changes in and disagreements with accountants on accounting and financial disclosure']},

13: {'item': 'item 9a', 'title': ['controls and procedures']},

14: {'item': 'item 9b', 'title': ['other information']},

15: {'item': 'item 9c', 'title': ['Disclosure Regarding Foreign Jurisdictions that Prevent Inspections']},

16: {'item': 'item 10', 'title': ['directors, executive officers and corporate governance','directors and executive officers of the registrant']},

17: {'item': 'item 11', 'title': ['executive compensation']},

18: {'item': 'item 12', 'title': ['security ownership of certain beneficial owners and management and related stockholder matters']},

19: {'item': 'item 13', 'title': ['certain relationships and related transactions']},

20: {'item': 'item 14', 'title': ['principal accountant fees and services']},

21: {'item': 'item 15', 'title': ['exhibits, financial statement schedules', 'exhibits and financial statement schedules']},

}

list\_10q\_items = [

"financial statement",

"risk factor",

"legal proceeding",

"mine safety disclosure",

"management’s discussion and analysis of financial condition and results of operations",

"quantitative and qualitative disclosures about market risk",

"controls and procedures",

"other information",

"unregistered sales of equity securities and use of proceeds",

"defaults upon senior securities",

"exhibits"

]

default\_10q\_sections = {

1: {'item': 'item 1', 'title': ['financial statement']},

2: {'item': 'item 2', 'title': ["management's discussion and analysis of financial condition and results of operations"]},

3: {'item': 'item 3', 'title': ['quantitative and qualitative disclosures about market risk']},

4: {'item': 'item 4', 'title': ['controls and procedures']},

5: {'item': 'item 1', 'title': ['legal proceeding']},

6: {'item': 'item 1a', 'title': ['risk factor']},

7: {'item': 'item 2', 'title': ["unregistered sales of equity securities and use of proceeds"]},

8: {'item': 'item 3', 'title': ["defaults upon senior securities"]},

9: {'item': 'item 4', 'title': ["mine safety disclosure"]},

10: {'item': 'item 5', 'title': ["other information"]},

11: {'item': 'item 6', 'title': ["exhibits"]},

}

default\_8k\_sections = {

1: {'item': 'item 1.01', 'title': ["entry into a material definitive agreement"]},

2: {'item': 'item 1.02', 'title': ["termination of a material definitive agreement"]},

3: {'item': 'item 1.03', 'title': ["bankruptcy or receivership"]},

4: {'item': 'item 1.04', 'title': ["mine safety"]},

5: {'item': 'item 2.01', 'title': ["completion of acquisition or disposition of asset"]},

6: {'item': 'item 2.02', 'title': ['results of operations and financial condition']},

7: {'item': 'item 2.03', 'title': ["creation of a direct financial obligation"]},

8: {'item': 'item 2.04', 'title': ["triggering events that accelerate or increase a direct financial obligation"]},

9: {'item': 'item 2.05', 'title': ["costs associated with exit or disposal activities"]},

10: {'item': 'item 2.06', 'title': ["material impairments"]},

11: {'item': 'item 3.01', 'title': ["notice of delisting or failure to satisfy a continued listing"]},

12: {'item': 'item 3.02', 'title': ["unregistered sales of equity securities"]},

13: {'item': 'item 3.03', 'title': ["material modification to rights of security holders"]},

14: {'item': 'item 4.01', 'title': ["changes in registrant's certifying accountant"]},

15: {'item': 'item 4.02', 'title': ["non-reliance on previously issued financial statements"]},

16: {'item': 'item 5.01', 'title': ["changes in control of registrant"]},

17: {'item': 'item 5.02', 'title': ['departure of directors or certain officers']},

18: {'item': 'item 5.03', 'title': ['amendments to articles of incorporation or bylaws']},

19: {'item': 'item 5.04', 'title': ["temporary suspension of trading under registrant"]},

20: {'item': 'item 5.05', 'title': ["amendment to registrant's code of ethics"]},

21: {'item': 'item 5.06', 'title': ["change in shell company status"]},

22: {'item': 'item 5.07', 'title': ['submission of matters to a vote of security holders']},

23: {'item': 'item 5.08', 'title': ["shareholder director nominations"]},

24: {'item': 'item 6.01', 'title': ["abs informational and computational material"]},

25: {'item': 'item 6.02', 'title': ['change of servicer or trustee']},

26: {'item': 'item 6.03', 'title': ['change in credit enhancement or other external support']},

27: {'item': 'item 6.04', 'title': ["failure to make a required distribution"]},

28: {'item': 'item 6.05', 'title': ["securities act updating disclosure"]},

29: {'item': 'item 7.01', 'title': ["regulation fd disclosure"]},

30: {'item': 'item 8.01', 'title': ['other events']},

31: {'item': 'item 9.01', 'title': ["financial statements and exhibits"]},

}

def string\_similarity\_percentage(string1, string2):

"""

Compute the leveshtein distance between the two strings and return the percentage simialrity.

:param string1:

:param string2:

:return: a float representing the percentage of similarity

"""

distance = Levenshtein.distance(string1.replace(" ", ""), string2.replace(" ", ""))

max\_length = max(len(string1), len(string2))

similarity\_percentage = (1 - (distance / max\_length)) \* 100

return similarity\_percentage

def clean\_section\_title(title):

"""

Clean the title string removing special words and punctuation that makes harder to recognize.

:param title: a string

:return: a cleaned string, lowercase

"""

# lower case

title = title.lower()

# remove special html characters

title = unidecode(title)

# remove item

title = title.replace("item ", "")

# remove '1.' etc

for idx in range(20, 0, -1):

for let in ['', 'a', 'b', 'c']:

title = title.replace(f"{idx}{let}.", "")

for idx in range(10, 0, -1):

title = title.replace(f"f-{idx}", "")

# remove parentesis and strip

title = re.sub(r'\([^)]\*\)', '', title).strip(string.punctuation + string.whitespace)

return title

def is\_title\_valid(text):

"""

Check if title is valid, meaning;

it does not starts with key words like: item, part, signature, page or is digit and has less than 2 chars

:param text: a string representing the title

:return: True if all conditions are satisfied else False

"""

valid = not (

text.startswith("item") or

text.startswith("part") or

text.startswith("signature") or

text.startswith("page") or

text.isdigit() or

len(text) <= 2)

return valid

def parse\_segments():

done\_ciks = []

docs = mongodb.get\_collection\_documents("documents")

for doc in docs:

if "aapl" not in doc["\_id"]:

continue

if doc["form\_type"] != "10-K":

continue

cik = doc["\_id"].split("data/")[1].split("/")[0]

if cik in done\_ciks:

continue

else:

done\_ciks.append(cik)

print(f"######## {doc['\_id']} ##########\n")

page = doc["html"]

soap = BeautifulSoup(page, features="html.parser")

ix\_resources = soap.find("ix:resources")

contexts = ix\_resources.findAll("xbrli:context")

axis = [

"srt:ProductOrServiceAxis",

"us-gaap:StatementBusinessSegmentsAxis",

"srt:ConsolidationItemsAxis",

"srt:StatementGeographicalAxis",

]

for c in contexts:

context\_id = c["id"]

s = c.find("xbrli:segment")

if s is not None:

members = s.find\_all("xbrldi:explicitmember")

if len(members) == 0:

continue

include = True

for m in members:

if m["dimension"] not in axis:

include = False

break

if not include:

continue

try:

period = c.find("xbrli:enddate").text

except:

period = c.find("xbrli:instant").text

period = datetime.strptime(period, "%Y-%m-%d").date()

# dimension = "+".join([x["dimension"] for x in members])

# value = "+".join([x.text for x in members])

# if dimension not in result\_dict:

# result\_dict[dimension] = {}

#

# if value not in result\_dict[dimension] or period > result\_dict[dimension][value]["period"]:

# result\_dict[dimension][value] = {"period":period,"id":context\_id}

element = soap.find("ix:nonfraction", attrs={"contextref": context\_id})

if element is None:

continue

segment = {}

for m in members:

segment[m["dimension"]] = m.text

print(f"{period} - {segment} => {element.text} ({element['name']})")

return

def find\_possible\_axis():

axis = []

docs = mongodb.get\_collection\_documents("documents")

for doc in docs:

page = doc["html"]

soap = BeautifulSoup(page, features="html.parser")

ix\_resources = soap.find("ix:resources")

if ix\_resources is None:

continue

contexts = ix\_resources.findAll("xbrli:context")

for c in contexts:

s = c.find("xbrli:segment")

if s is not None:

try:

ax = [x["dimension"] for x in s.children]

for a in ax:

if a not in axis:

print(a)

axis.append(a)

except:

pass

def identify\_table\_of\_contents(soup, list\_items):

"""

Given a soup object and a list of item, this method look for a table of contents.

:param soup: soup object of the document

:param list\_items: an array of strings related to sections titles.

:return: the table of contents PageElement object or None if not found.

"""

if list\_items is None:

return None

max\_table = 0

chosen\_table = None

tables = soup.body.findAll("table")

for t in tables:

count = 0

for s in list\_items:

r = t.find(string=re.compile(f'{s}', re.IGNORECASE))

if r is not None:

count += 1

if count > max\_table:

chosen\_table = t

max\_table = count

if max\_table > 3:

return chosen\_table

return None

def get\_sections\_using\_hrefs(soup, table\_of\_contents):

"""

Scan the table\_of\_contents and identify all hrefs, if present.

The method create a dictionary of sections by finding tag elements referenced inside soup with the specific hrefs.

:param soup: soup object of the document.

:param table\_of\_contents:

:return: a dictionary with the following structure:

{1:

{

'start\_el': tag element where the section starts,

'idx': an integer index of start element inside soup, used for ordering

'title': a string representing the section title,

'title\_candidates': a list of title candidates. If there is a single candidate that becomes the title

'end\_el': tag element where the section ends,

'text': the text of the section

},

...

}

Section are ordered based on chid['idx'] value

:param soup:

:return: section dictionary

"""

all\_elements = soup.find\_all()

hrefs = {}

sections = {}

for tr in table\_of\_contents.findAll("tr"):

try:

aa = tr.find\_all("a")

tr\_hrefs = [a['href'][1:] for a in aa]

except Exception as e:

continue

for el in tr.children:

text = el.text

text = clean\_section\_title(text)

if is\_title\_valid(text):

for tr\_href in tr\_hrefs:

if tr\_href not in hrefs:

h\_tag = soup.find(id=tr\_href)

if h\_tag is None:

h\_tag = soup.find(attrs={"name": tr\_href})

if h\_tag:

hrefs[tr\_href] = {

'start\_el': h\_tag,

"link": tr\_href,

'idx': all\_elements.index(h\_tag),

'title': None,

'title\_candidates': {text}}

else:

hrefs[tr\_href]['title\_candidates'].add(text)

else:

continue

for h in hrefs:

hrefs[h]['title\_candidates'] = list(hrefs[h]['title\_candidates'])

if len(hrefs[h]['title\_candidates']) == 1:

hrefs[h]['title'] = hrefs[h]['title\_candidates'][0]

else:

hrefs[h]['title'] = "+++".join(hrefs[h]['title\_candidates'])

temp\_s = sorted(hrefs.items(), key=lambda x: x[1]["idx"])

for i, s in enumerate(temp\_s):

sections[i + 1] = s[1]

if i > 0:

sections[i]["end\_el"] = sections[i + 1]["start\_el"]

sections = get\_sections\_text\_with\_hrefs(soup, sections)

return sections

def select\_best\_match(string\_to\_match, matches, start\_index):

"""

Identifies the best match, in terms of similarity distance between a string\_to\_match and a list of matches.

start\_index is used to avoid cases where the string\_to\_match is matched with the first occurence in matches.

:param string\_to\_match: a string

:param matches: a list of regular expresion matches

:param start\_index: a integer representing the index to start from

:return: a regualr expression match with highest simialrity

"""

match = None

if start\_index == 0:

del matches[0]

if len(matches) == 1:

match = matches[0]

if matches[0].start() > start\_index:

match = matches[0]

elif len(matches) > 1:

max\_similarity = -1

for i, m in enumerate(matches):

if m.start() > start\_index:

sim = string\_similarity\_percentage(string\_to\_match, m.group().lower().replace("\n", " "))

if sim > max\_similarity:

max\_similarity = sim

match = m

return match

def get\_sections\_using\_strings(soup, table\_of\_contents, default\_sections):

"""

Scan the table\_of\_contents and identify possible section text using strings that match default\_sections.

Retrieve sections strings in soup.body.text.

:param soup: the soup object

:param table\_of\_contents: a PageElement from soup that represent the table of contents

:param default\_sections: a dictionary that contains prefilled data about default sections that could be found in the document

:return: a dictionary with the following structure, representing the sections:

{1:

{

'start\_index': the start index of the section inside soup.body.text

'end\_index': the start index of the section inside soup.body.text,

'title': a string representing the section title,

'end\_el': tag element where the section ends

},

...

}

Section are ordered based on chid['idx'] value

"""

# Clean soup.body.text removing consecutive \n and spaces

body\_text = unidecode(soup.body.get\_text(separator=" "))

body\_text = re.sub('\n', ' ', body\_text)

body\_text = re.sub(' +', ' ', body\_text)

# If there is a table\_of\_contents look for items strings a check for their validity

sections = {}

if table\_of\_contents:

num\_section = 1

for tr in table\_of\_contents.findAll("tr"):

section = {}

for el in tr.children:

text = el.text

item = unidecode(text.lower()).replace("\n", " ").strip(string.punctuation + string.whitespace)

# print(text)

# input("NEXt")

# remove special html characters

item = item

if 'item' in item:

section["item"] = item

text = clean\_section\_title(text)

if 'item' in section and is\_title\_valid(text):

section['title'] = text

sections[num\_section] = section

num\_section += 1

# Different behaviour if there is a table\_of\_contents and sections is already populated.

if len(sections) == 0:

# Here we didn't find any usable table\_of\_contents sections, then we use a prefilled default\_sections dictionary

sections = copy.deepcopy(default\_sections)

start\_index = 1

else:

# Here we skip first occurrence in text since it also present in table\_of\_contents

start\_index = 0

# Loop through all sections to identify a possible item and title for a section.

# If multiple values are found we select best match based on string similarity.

for si in sections:

s = sections[si]

if 'item' in s:

match = None

if isinstance(s['title'], list):

for t in s['title']:

matches = list(re.finditer(fr"{s['item']}. \*{t}", body\_text, re.IGNORECASE + re.DOTALL))

if matches:

match = select\_best\_match(f"{s['item']} {t}", matches, start\_index)

break

else:

matches = list(re.finditer(fr"{s['item']}. \*{s['title']}", body\_text, re.IGNORECASE + re.DOTALL))

if matches:

match = select\_best\_match(f"{s['item']} {s['title']}", matches, start\_index)

if match is None:

matches = list(re.finditer(fr"{s['item']}", body\_text, re.IGNORECASE + re.DOTALL))

if matches:

match = select\_best\_match(f"{s['item']}", matches, start\_index)

if match:

s['title'] = match.group()

s["start\_index"] = match.start()

start\_index = match.start()

else:

s['remove'] = True

sections\_temp = {}

for si in sections:

if "remove" not in sections[si]:

sections\_temp[si] = sections[si]

# Eventually we populate each section in the dictionary with its text taken from body\_text

temp\_s = sorted(sections\_temp.items(), key=lambda x: x[1]["start\_index"])

sections = {}

last\_section = 0

for i, s in enumerate(temp\_s):

sections[i + 1] = s[1]

if i > 0:

sections[i]["end\_index"] = sections[i + 1]["start\_index"]

sections[i]["text"] = body\_text[sections[i]["start\_index"]:sections[i]["end\_index"]]

last\_section = i + 1

if last\_section > 0:

sections[last\_section]["end\_index"] = -1

sections[last\_section]["text"] = body\_text[sections[last\_section]["start\_index"]:sections[last\_section]["end\_index"]]

return sections

def get\_sections\_text\_with\_hrefs(soup, sections):

"""

This method try to retrieve text from soup object related to a document and its sections

:param soup: a soup object

:param sections: a dictionary containing data about sections

:return:

"""

next\_section = 1

current\_section = None

text = ""

last\_was\_new\_line = False

for el in soup.body.descendants:

if next\_section in sections and el == sections[next\_section]['start\_el']:

if current\_section is not None:

sections[current\_section]["text"] = text

text = ""

last\_was\_new\_line = False

current\_section = next\_section

next\_section += 1

if current\_section is not None and isinstance(el, NavigableString):

if last\_was\_new\_line and el.text == "\n":

continue

elif el.text == "\n":

last\_was\_new\_line = True

else:

last\_was\_new\_line = False

found\_text = unidecode(el.get\_text(separator=" "))

if sections[current\_section]['title'] is None:

if found\_text in sections[current\_section]['title\_candidates']:

print(f"{bcolors.OKCYAN}"

f'new title for {current\_section}: {found\_text} in {sections[current\_section]["title\_candidates"]}'

f"{bcolors.ENDC}")

sections[current\_section]['title'] = found\_text

if len(text) > 0 and text[-1] != " " and len(found\_text) > 0 and found\_text[0] != " ":

text += "\n"

text += found\_text.replace('\n', ' ')

if current\_section is not None:

sections[current\_section]["text"] = text

return sections

def parse\_document(doc):

"""

Take a document, SEC filing, parse the content and retrieve the sections.

Save the result on mongoDB under parsed\_documents collection.

:param doc: document from "documents" collection of mongoDB

:return:

"""

url = doc["\_id"]

form\_type = doc["form\_type"]

filing\_date = doc["filing\_date"]

sections = {}

cik = doc["cik"]

html = doc["html"]

# Supported form type are 10-K, 10-K/A, 10-Q, 10-Q/A, 8-K

if form\_type in ["10-K", "10-K/A"]:

include\_forms = ["10-K", "10-K/A"]

list\_items = list\_10k\_items

default\_sections = default\_10k\_sections

elif form\_type == "10-Q":

include\_forms = ["10-Q"]

list\_items = list\_10q\_items

default\_sections = default\_10q\_sections

elif form\_type == "8-K":

include\_forms = ["8-K"]

list\_items = None

default\_sections = default\_8k\_sections

else:

print(f"return because form\_type {form\_type} is not valid")

return

if form\_type not in include\_forms:

print(f"return because form\_type != {form\_type}")

return

company\_info = company\_from\_cik(cik)

# no cik in cik\_map

if company\_info is None:

print("return because company info None")

return

print(f"form type: \t\t{form\_type}")

print(company\_info)

soup = BeautifulSoup(html, features="html.parser")

if soup.body is None:

print("return because soup.body None")

return

table\_of\_contents = identify\_table\_of\_contents(soup, list\_items)

if table\_of\_contents:

sections = get\_sections\_using\_hrefs(soup, table\_of\_contents)

if len(sections) == 0:

sections = get\_sections\_using\_strings(soup, table\_of\_contents, default\_sections)

result = {"\_id": url, "cik": cik, "form\_type":form\_type, "filing\_date": filing\_date, "sections":{}}

for s in sections:

section = sections[s]

if 'text' in section:

text = section['text']

text = re.sub('\n', ' ', text)

text = re.sub(' +', ' ', text)

result["sections"][section["title"]] = {"text":text, "link":section["link"] if "link" in section else None}

try:

mongodb.upsert\_document("parsed\_documents", result)

except:

traceback.print\_exc()

print(result.keys())

print(result["sections"].keys())

def find\_auditor(doc):

try:

soup = BeautifulSoup(doc["html"], features="html.parser")

# auditor\_start\_string = 'Report of Independent Registered Public Accounting Firm'.lower()

# auditor\_string = ""

body = unidecode(soup.body.get\_text(separator=" "))

body = re.sub('\n', ' ', body)

body = re.sub(' +', ' ', body)

start\_sig = 0

while start\_sig != -1:

start\_sig = body.find('s/', start\_sig+1)

auditor\_candidate = body[start\_sig: start\_sig+200]

# print(auditor\_candidate)

if 'auditor since' in auditor\_candidate.lower():

pattern = r"s/.+auditor since.\*?\d{4}"

try:

match = re.findall(pattern, auditor\_candidate)[0]

return match.replace("s/", "").strip()

except:

pass

except Exception as e:

print(e)

print("NO AUDITOR FOUND")

return ""

# print(auditor\_string)

# if auditor\_start\_string in body.lower():

# start\_sig = 0

# while start\_sig != -1:

# start\_sig = body.lower().find(auditor\_start\_string, start\_sig)

# if start\_sig != -1:

# start\_sig = body.find('s/', start\_sig)

# end\_sig = body.find('.', start\_sig)

# auditor\_candidate = body[start\_sig: end\_sig]

# if 'auditor since' in auditor\_candidate.lower():

# auditor\_string += body[start\_sig: end\_sig] + "\n"

# if auditor\_string == "":

# auditor\_start\_string = 'auditor'

# if auditor\_start\_string in body.lower():

# start\_sig = 0

# while start\_sig != -1:

# start\_sig = body.lower().find(auditor\_start\_string, start\_sig + len(auditor\_start\_string))

# if start\_sig != -1:

# auditor\_string += body[start\_sig - 100: start\_sig + 100] + "\n"

# print(auditor\_string)

# return auditor\_string

class bcolors:

HEADER = '\033[95m'

OKBLUE = '\033[94m'

OKCYAN = '\033[96m'

OKGREEN = '\033[92m'

WARNING = '\033[93m'

FAIL = '\033[91m'

ENDC = '\033[0m'

BOLD = '\033[1m'

UNDERLINE = '\033[4m'

def test():

test\_docs = {

"docs\_with\_table\_of\_contents\_and\_hrefs": [

"https://www.sec.gov/Archives/edgar/data/12040/000117494723000017/form10k-29127\_bdl.htm",

],

"docs\_with\_table\_of\_contents\_no\_hrefs": [

"https://www.sec.gov/Archives/edgar/data/10329/000143774923001642/bset20230109\_10k.htm",

],

"docs\_without\_table\_of\_contents": [

"https://www.sec.gov/Archives/edgar/data/315374/000155837023000097/hurc-20221031x10k.htm",

"https://www.sec.gov/Archives/edgar/data/97476/000009747623000007/txn-20221231.htm",

"https://www.sec.gov/Archives/edgar/data/315213/000031521323000016/rhi-20221231.htm" # item from 10 to 14 are missing in filing

]

}

import re

import sys

import time

from datetime import datetime

from statistics import median

import requests

from bs4 import BeautifulSoup

from dateutil.relativedelta import relativedelta

from forex\_python.converter import CurrencyRates, RatesNotAvailableError

from unidecode import unidecode

from urllib3.exceptions import ProtocolError

import numpy as np

import mongodb

from investing\_com import get\_10y\_bond\_yield

from postgresql import get\_df\_from\_table

from yahoo\_finance import get\_current\_price\_from\_yahoo

import math

import pandas as pd

r\_and\_d\_amortization = {

'Advertising': 2,

'Aerospace/Defense': 10,

'Air Transport': 10,

'Apparel': 3,

'Auto & Truck': 10,

'Auto Parts': 5,

'Bank (Money Center)': 2,

'Banks (Regional)': 2,

'Beverage (Alcoholic)': 3,

'Beverage (Soft)': 3,

"Broadcasting": 10,

"Brokerage & Investment Banking": 3,

'Building Materials': 5,

'Construction Supplies': 5,

"Business & Consumer Services": 5,

'Cable TV': 10,

'Chemical (Basic)': 10,

'Chemical (Diversified)': 10,

'Chemical (Specialty)': 10,

'Coal & Related Energy': 5,

'Computer & Peripherals': 5,

'Computer Services': 3,

'Diversified': 5,

"Drugs (Biotechnology)": 10,

"Drugs (Pharmaceutical)": 10,

'Education': 3,

'Electrical Equipment': 10,

'Electronics (Consumer & Office)': 5,

'Electronics (General)': 5,

"Engineering/Construction": 10,

'Entertainment': 3,

'Environmental & Waste Services': 5,

"Farming/Agriculture": 10,

'Financial Svcs. (Non-bank & Insurance)': 2,

'Food Processing': 3,

'Food Wholesalers': 3,

'Furn/Home Furnishings': 3,

"Green & Renewable Energy": 10,

"Healthcare Products": 5,

"Healthcare Support Services": 3,

'Heathcare Information and Technology': 3,

'Homebuilding': 5,

"Hospitals/Healthcare Facilities": 10,

'Hotel/Gaming': 3,

'Household Products': 3,

"Information Services": 3,

'Insurance (General)': 3,

'Insurance (Life)': 3,

'Insurance (Prop/Cas.)': 3,

'Investments & Asset Management': 3,

'Machinery': 10,

'Metals & Mining': 5,

'Office Equipment & Services': 5,

"Oil/Gas (Integrated)": 10,

"Oil/Gas (Production and Exploration)": 10,

"Oil/Gas Distribution": 10,

"Oilfield Svcs/Equip.": 5,

'Packaging & Container': 5,

'Paper/Forest Products': 10,

"Power": 10,

"Precious Metals": 5,

'Petroleum (Integrated)': 5,

'Petroleum (Producing)': 5,

'Precision Instrument': 5,

'Publishing & Newspapers': 3,

'R.E.I.T.': 3,

"Real Estate (Development)": 5,

"Real Estate (General/Diversified)": 5,

"Real Estate (Operations & Services)": 5,

'Recreation': 5,

'Reinsurance': 3,

'Restaurant/Dining': 2,

'Retail (Special Lines)': 2,

'Retail (Building Supply)': 2,

'Retail (General)': 2,

"Retail (Automotive)": 2,

"Retail (Distributors)": 2,

"Retail (Grocery and Food)": 2,

"Retail (Online)": 2,

"Rubber& Tires": 5,

'Semiconductor': 5,

'Semiconductor Equip': 5,

'Shipbuilding & Marine': 10,

'Shoe': 3,

"Software (Entertainment)": 3,

"Software (Internet)": 3,

"Software (System & Application)": 3,

'Steel': 5,

"Telecom (Wireless)": 5,

'Telecom. Equipment': 10,

'Telecom. Services': 5,

'Tobacco': 5,

'Toiletries/Cosmetics': 3,

'Transportation': 5,

'Transportation (Railroads)': 5,

'Trucking': 5,

'Utility (General)': 10,

'Utility (Water)': 10

}

industry\_complexity = {

'Advertising': 1,

'Apparel': 1,

'Auto & Truck': 1,

'Auto Parts': 1,

'Beverage (Alcoholic)': 1,

'Beverage (Soft)': 1,

'Building Materials': 1,

'Construction Supplies': 1,

'Food Wholesalers': 1,

'Furn/Home Furnishings': 1,

'Household Products': 1,

'Retail (Building Supply)': 1,

'Retail (General)': 1,

"Retail (Automotive)": 1,

"Retail (Distributors)": 1,

"Retail (Grocery and Food)": 1,

'Tobacco': 1,

'Toiletries/Cosmetics': 1,

'Restaurant/Dining': 1,

"Rubber& Tires": 1,

'Shoe': 1,

"Business & Consumer Services": 2,

'Cable TV': 2,

'Air Transport': 2,

'Coal & Related Energy': 2,

'Computer & Peripherals': 2,

'Computers/Peripherals': 2,

'Education': 2,

'Electrical Equipment': 2,

'Entertainment': 2,

'Homebuilding': 2,

"Hospitals/Healthcare Facilities": 2,

'Hotel/Gaming': 2,

'Food Processing': 2,

'Office Equipment & Services': 2,

'Packaging & Container': 2,

'Paper/Forest Products': 2,

"Real Estate (Development)": 2,

"Real Estate (General/Diversified)": 2,

"Real Estate (Operations & Services)": 2,

'Recreation': 2,

'Retail (Special Lines)': 2,

"Retail (Online)": 2,

'Publishing & Newspapers': 2,

'Steel': 2,

'Electronics (Consumer & Office)': 2,

'Electronics (General)': 2,

'R.E.I.T.': 2,

'Transportation': 2,

'Transportation (Railroads)': 2,

'Trucking': 2,

"Broadcasting": 3,

'Metals & Mining': 3,

"Precious Metals": 3,

'Aerospace/Defense': 3,

'Chemical (Basic)': 3,

'Chemical (Diversified)': 3,

'Computer Services': 3,

"Engineering/Construction": 3,

"Farming/Agriculture": 3,

'Heathcare Information and Technology': 3,

"Information Services": 3,

'Insurance (General)': 3,

'Insurance (Life)': 3,

'Insurance (Prop/Cas.)': 3,

'Investments & Asset Management': 3,

'Machinery': 3,

'Utility (General)': 3,

'Utility (Water)': 3,

"Software (Entertainment)": 3,

"Software (Internet)": 3,

"Software (System & Application)": 3,

'Reinsurance': 3,

'Semiconductor': 3,

"Power": 3,

'Telecom. Services': 3,

'Shipbuilding & Marine': 3,

"Telecom (Wireless)": 3,

'Telecom. Equipment': 3,

"Healthcare Products": 3,

"Healthcare Support Services": 3,

'Bank (Money Center)': 4,

'Banks (Regional)': 4,

"Brokerage & Investment Banking": 4,

'Financial Svcs. (Non-bank & Insurance)': 4,

'Environmental & Waste Services': 4,

"Green & Renewable Energy": 4,

"Oil/Gas (Integrated)": 4,

"Oil/Gas (Production and Exploration)": 4,

"Oil/Gas Distribution": 4,

"Oilfield Svcs/Equip.": 4,

'Petroleum (Integrated)': 4,

'Petroleum (Producing)': 4,

'Precision Instrument': 4,

'Semiconductor Equip': 4,

'Chemical (Specialty)': 5,

'Diversified': 5,

"Drugs (Biotechnology)": 5,

"Drugs (Pharmaceutical)": 5,

}

industry\_cyclical = {

'Advertising': True,

'Apparel': False,

'Auto & Truck': True,

'Auto Parts': True,

'Beverage (Alcoholic)': False,

'Beverage (Soft)': False,

'Building Materials': True,

'Construction Supplies': True,

'Food Wholesalers': False,

'Furn/Home Furnishings': True,

'Household Products': False,

'Retail (Building Supply)': True,

'Retail (General)': False,

"Retail (Automotive)": True,

"Retail (Distributors)": True,

"Retail (Grocery and Food)": False,

'Tobacco': False,

'Toiletries/Cosmetics': False,

'Restaurant/Dining': True,

"Rubber& Tires": True,

'Shoe': False,

"Business & Consumer Services": True,

'Cable TV': False,

'Air Transport': True,

'Coal & Related Energy': True,

'Computer & Peripherals': True,

'Computers/Peripherals': True,

'Education': False,

'Electrical Equipment': True,

'Entertainment': True,

'Homebuilding': True,

"Hospitals/Healthcare Facilities": False,

'Hotel/Gaming': True,

'Food Processing': False,

'Office Equipment & Services': True,

'Packaging & Container': True,

'Paper/Forest Products': True,

"Real Estate (Development)": True,

"Real Estate (General/Diversified)": True,

"Real Estate (Operations & Services)": True,

'Recreation': True,

'Retail (Special Lines)': True,

"Retail (Online)": True,

'Publishing & Newspapers': False,

'Steel': True,

'Electronics (Consumer & Office)': True,

'Electronics (General)': True,

'R.E.I.T.': True,

'Transportation': True,

'Transportation (Railroads)': True,

'Trucking': True,

"Broadcasting": False,

'Metals & Mining': True,

"Precious Metals": True,

'Aerospace/Defense': False,

'Chemical (Basic)': True,

'Chemical (Diversified)': True,

'Computer Services': True,

"Engineering/Construction": True,

"Farming/Agriculture": False,

'Heathcare Information and Technology': False,

"Information Services": True,

'Insurance (General)': False,

'Insurance (Life)': False,

'Insurance (Prop/Cas.)': False,

'Investments & Asset Management': True,

'Machinery': True,

'Utility (General)': False,

'Utility (Water)': False,

"Software (Entertainment)": True,

"Software (Internet)": True,

"Software (System & Application)": True,

'Reinsurance': True,

'Semiconductor': True,

"Power": False,

'Telecom. Services': False,

'Bank (Money Center)': False,

'Banks (Regional)': False,

"Brokerage & Investment Banking": True,

'Financial Svcs. (Non-bank & Insurance)': True,

'Environmental & Waste Services': False,

"Green & Renewable Energy": False,

"Healthcare Products": False,

"Healthcare Support Services": False,

"Oil/Gas (Integrated)": True,

"Oil/Gas (Production and Exploration)": True,

"Oil/Gas Distribution": True,

"Oilfield Svcs/Equip.": True,

'Petroleum (Integrated)': True,

'Petroleum (Producing)': True,

'Precision Instrument': True,

'Semiconductor Equip': True,

'Shipbuilding & Marine': True,

"Telecom (Wireless)": False,

'Telecom. Equipment': False,

'Chemical (Specialty)': False,

'Diversified': False,

"Drugs (Biotechnology)": False,

"Drugs (Pharmaceutical)": False,

}

def company\_complexity(doc, industry, company\_size):

base = industry\_complexity[industry]

# no 10-K

if doc is None:

length\_modifier = 1

else:

soup = BeautifulSoup(doc["html"], 'html.parser')

body\_text = unidecode(soup.body.get\_text(separator=" "))

body\_text = re.sub('\n', ' ', body\_text)

body\_text = re.sub(' +', ' ', body\_text)

length = len(body\_text)

if length > 1.4 \* 10 \*\* 6:

length\_modifier = 4

elif length > 1.1 \* 10 \*\* 6:

length\_modifier = 3

elif length > 8 \* 10 \*\* 5:

length\_modifier = 2

elif length > 5 \* 10 \*\* 5:

length\_modifier = 1

else:

length\_modifier = 0

if company\_size == "Large" or company\_size == "Mega":

company\_size\_modifier = 1

else:

company\_size\_modifier = 0

return min(5, max(1, base + length\_modifier + company\_size\_modifier))

def company\_share\_diluition(shares):

first\_idx = 0

for i, s in enumerate(shares):

if s > 0:

first\_idx = i

break

l = shares[first\_idx:]

years = len(l) - 1

return (l[-1] / l[0]) \*\* (1 / years) - 1

def get\_company\_type(revenue\_growth, mr\_debt\_adj, equity\_mkt, liquidation\_value, operating\_margin\_5y, industry):

fast\_grower = False

stalward = False

slow\_grower = False

declining = False

turn\_around = False

asset\_play = False

cyclical = industry\_cyclical[industry]

avg\_revenue\_growth = np.mean(revenue\_growth)

if avg\_revenue\_growth > 0.15:

fast\_grower = True

elif avg\_revenue\_growth > 0.07:

stalward = True

elif avg\_revenue\_growth > -0.02:

slow\_grower = True

else:

declining = True

# high debt + liquidation > debt + low margins

if mr\_debt\_adj > equity\_mkt \* 2 and liquidation\_value > mr\_debt\_adj and operating\_margin\_5y < 0.05:

turn\_around = True

if liquidation\_value > equity\_mkt:

asset\_play = True

return {

"fast\_grower": fast\_grower,

"stalward": stalward,

"slow\_grower": slow\_grower,

"declining": declining,

"turn\_around": turn\_around,

"asset\_play": asset\_play,

"cyclical": cyclical

}

def convert\_currencies(db\_curr, db\_financial\_curr, currency=None, financial\_currency=None):

if (currency is not None and financial\_currency is not None):

financial\_currency = financial\_currency.replace("Currency in ", "")

else:

currency = db\_curr

financial\_currency = db\_financial\_curr

if (financial\_currency == "GBP" and ("GBp" in currency or "0.01" in currency)) \

or (financial\_currency == "ZAR" and ("ZAC" in currency or "ZAc" in currency or "0.01" in currency)) \

or (financial\_currency == "ILS" and ("ILA" in currency or "0.01" in currency)):

fx\_rate = 100

else:

c = CurrencyRates()

rates = None

num\_retry = 0

max\_retry = 5

while (rates is None and num\_retry < max\_retry):

if num\_retry > 0:

print("# retry = " + str(num\_retry) + ", get forex rates")

if num\_retry > 0:

time.sleep(0.5 \* num\_retry)

try:

rates = c.get\_rates(financial\_currency)

except requests.exceptions.ConnectionError:

pass

except ProtocolError:

pass

except RatesNotAvailableError:

break

num\_retry += 1

multiplier = 1

if "GBp" in currency:

currency = "GBP"

multiplier = 100

if "ZAc" in currency or "ZAC" in currency:

currency = "ZAR"

multiplier = 100

if "ILA" in currency:

currency = "ILS"

multiplier = 100

if rates is not None and currency in rates:

fx\_rate = rates[currency]

else:

if financial\_currency == "USD":

fx\_rate = get\_current\_price\_from\_yahoo(f"{currency}=X")

else:

fx\_rate = get\_current\_price\_from\_yahoo(f"{financial\_currency}{currency}=X")

# print("FX Yahoo", fx\_rate)

# raise Exception("mine")

fx\_rate \*= multiplier

# if debug:

# print("CONVERT ", financial\_currency, "=>", currency, ": x", fx\_rate)

return fx\_rate

def capitalize\_rd(r\_and\_d, r\_and\_d\_amortization\_years, tax\_rate, years):

# print("DEBUG R&D")

# print(r\_and\_d)

# print(r\_and\_d\_amortization\_years)

# print(tax\_rate)

# print(years)

# last element does not amortize this year

r\_and\_d = r\_and\_d[-r\_and\_d\_amortization\_years-1:]

while len(r\_and\_d) < years:

r\_and\_d.insert(0, 0)

r\_and\_d\_amortization\_cy = [sum(i \* 1 / r\_and\_d\_amortization\_years for i in r\_and\_d[:-1])]

# first element is fully amortized after this year

r\_and\_d\_unamortized = [sum(i[0] \* i[1] for i in

zip(r\_and\_d[-r\_and\_d\_amortization\_years:],

np.linspace(1 / r\_and\_d\_amortization\_years, 1, r\_and\_d\_amortization\_years)))]

ebit\_r\_and\_d\_adj = [r\_and\_d[-1] - r\_and\_d\_amortization\_cy[0]]

tax\_benefit = [ebit\_r\_and\_d\_adj[0] \* tax\_rate]

# print("r\_and\_d after inserting 0", r\_and\_d)

r\_and\_d\_growth = []

for i in range(len(r\_and\_d) - 1):

try:

r\_and\_d\_growth.append(r\_and\_d[i + 1] / r\_and\_d[i] - 1)

except:

r\_and\_d\_growth.append(0)

# first element is the growth between most recent year and the year before that

r\_and\_d\_growth.reverse()

# print(r\_and\_d)

# print(r\_and\_d\_growth)

for g in r\_and\_d\_growth:

if g == -1:

tax\_benefit.append(0)

ebit\_r\_and\_d\_adj.append(0)

r\_and\_d\_unamortized.append(0)

r\_and\_d\_amortization\_cy.append(0)

else:

tax\_benefit.append(tax\_benefit[-1] / (1 + g))

ebit\_r\_and\_d\_adj.append(ebit\_r\_and\_d\_adj[-1] / (1 + g))

r\_and\_d\_unamortized.append(r\_and\_d\_unamortized[-1] / (1 + g))

r\_and\_d\_amortization\_cy.append(r\_and\_d\_amortization\_cy[-1] / (1 + g))

# reverse order

for l in [ebit\_r\_and\_d\_adj, tax\_benefit, r\_and\_d\_unamortized, r\_and\_d\_amortization\_cy]:

l.reverse()

# print("r\_and\_d", r\_and\_d)

# print("r\_and\_d\_amortization\_years", r\_and\_d\_amortization\_years)

# print("r\_and\_d\_growth", r\_and\_d\_growth)

# print("r\_and\_d\_amortization\_cy", r\_and\_d\_amortization\_cy)

# print("r\_and\_d\_unamortized", r\_and\_d\_unamortized)

# print("ebit\_r\_and\_d\_adj", ebit\_r\_and\_d\_adj)

# print("tax\_benefit", tax\_benefit)

return ebit\_r\_and\_d\_adj, tax\_benefit, r\_and\_d\_unamortized, r\_and\_d\_amortization\_cy

def get\_spread\_from\_dscr(interest\_coverage\_ratio, damodaran\_bond\_spread):

spread = damodaran\_bond\_spread[(interest\_coverage\_ratio >= damodaran\_bond\_spread["greater\_than"]) &

(interest\_coverage\_ratio < damodaran\_bond\_spread["less\_than"])].iloc[0]

return float(spread["spread"])

def debtize\_op\_leases(ttm\_interest\_expense, ttm\_ebit\_adj, damodaran\_bond\_spread, riskfree, country\_default\_spread,

leases, last\_year\_leases, tax\_rate, revenue\_growth):

int\_exp\_op\_adj = 0

ttm\_ebit\_op\_adj = 0

debt\_adj = [0]

interest\_coverage\_ratio = 12.5

company\_default\_spread = -1

visited\_icr = []

done = False

# CYCLE

while not done:

helper\_interest\_expense\_adj = ttm\_interest\_expense + int\_exp\_op\_adj

helper\_ebit\_adj = ttm\_ebit\_adj + ttm\_ebit\_op\_adj

try:

if helper\_interest\_expense\_adj > 0:

interest\_coverage\_ratio = min(99999, helper\_ebit\_adj / helper\_interest\_expense\_adj)

except:

interest\_coverage\_ratio = 12.5

spread = get\_spread\_from\_dscr(interest\_coverage\_ratio, damodaran\_bond\_spread)

if spread == company\_default\_spread:

done = True

else:

if interest\_coverage\_ratio in visited\_icr:

done = True

visited\_icr.append(interest\_coverage\_ratio)

company\_default\_spread = spread

cost\_of\_debt = riskfree + country\_default\_spread + company\_default\_spread

pv\_leases = []

for i in range(1, len(leases)):

pv\_leases.append(leases[i] / (1 + cost\_of\_debt) \*\* i)

debt\_adj = sum(pv\_leases)

if last\_year\_leases > 0:

op\_leases\_depreciation = (debt\_adj + leases[0]) / (last\_year\_leases + 1)

else:

op\_leases\_depreciation = 0

# update helper\_ebit

ttm\_ebit\_op\_adj = leases[0] - op\_leases\_depreciation

# update helper\_interest

int\_exp\_op\_adj = leases[0] \* (1 - 1 / (1 + cost\_of\_debt))

# print(interest\_coverage\_ratio, spread, company\_default\_spread, cost\_of\_debt)

ebit\_op\_adj = [ttm\_ebit\_op\_adj]

tax\_benefit\_op = [ebit\_op\_adj[0] \* tax\_rate]

debt\_adj = [debt\_adj]

# first element is the growth between most year and the year before that

revenue\_growth.reverse()

for g in revenue\_growth:

tax\_benefit\_op.append(tax\_benefit\_op[-1] / (1 + g))

ebit\_op\_adj.append(ebit\_op\_adj[-1] / (1 + g))

debt\_adj.append(debt\_adj[-1] / (1 + g))

# restore revenue\_growth

revenue\_growth.reverse()

# reverse order

for l in [tax\_benefit\_op, ebit\_op\_adj, debt\_adj]:

l.reverse()

# print("leases", leases)

# print("cost of debt", cost\_of\_debt)

# print("pv\_leases", pv\_leases)

# print("ttm\_ebit\_op\_adj", ttm\_ebit\_op\_adj)

# print("tax\_benefit\_op", tax\_benefit\_op)

# print("debt\_adj", debt\_adj)

# print("years dep", last\_year\_leases)

# print("depreciation", op\_leases\_depreciation)

# print("helper\_ebit\_adj", helper\_ebit\_adj)

# print("helper\_interest\_expense\_adj", helper\_interest\_expense\_adj)

return ebit\_op\_adj, int\_exp\_op\_adj, debt\_adj, tax\_benefit\_op, company\_default\_spread

def get\_growth\_ttm(ttm\_ebit\_after\_tax, ttm\_net\_income\_adj, mr\_equity\_adj, mr\_debt\_adj, mr\_cash\_and\_securities,

reinvestment, ttm\_dividends, industry\_payout):

if (mr\_debt\_adj + mr\_equity\_adj - mr\_cash\_and\_securities) > 0:

# print("ROC LAST")

# print(ttm\_ebit\_after\_tax)

# print(mr\_debt\_adj)

# print(mr\_equity\_adj)

# print(mr\_cash\_and\_securities)

roc\_last = ttm\_ebit\_after\_tax / (mr\_debt\_adj + mr\_equity\_adj - mr\_cash\_and\_securities)

else:

roc\_last = 0

if ttm\_ebit\_after\_tax > 0:

reinvestment\_last = reinvestment[-1] / ttm\_ebit\_after\_tax

else:

reinvestment\_last = 0

if reinvestment\_last < 0:

reinvestment\_last = 1 - industry\_payout

growth\_last = roc\_last \* reinvestment\_last

if mr\_equity\_adj > 0:

roe\_last = ttm\_net\_income\_adj / mr\_equity\_adj

else:

roe\_last = 0

if ttm\_net\_income\_adj > 0:

reinvestment\_eps\_last = 1 - ttm\_dividends / ttm\_net\_income\_adj

else:

reinvestment\_eps\_last = 0

growth\_eps\_last = roe\_last \* reinvestment\_eps\_last

return roc\_last, reinvestment\_last, growth\_last, roe\_last, reinvestment\_eps\_last, growth\_eps\_last

def get\_roe\_roc(equity\_bv\_adj, debt\_bv\_adj, cash\_and\_securities, ebit\_after\_tax, net\_income\_adj):

roc = []

roe = []

avg\_equity = sum(equity\_bv\_adj) / len(equity\_bv\_adj)

for i in range(len(equity\_bv\_adj)):

invested\_capital = debt\_bv\_adj[i] + equity\_bv\_adj[i] - cash\_and\_securities[i]

if invested\_capital <= 0:

roc.append(0)

else:

try:

roc.append(ebit\_after\_tax[i] / invested\_capital)

except:

roc.append(0)

if equity\_bv\_adj[i] > 0:

eq = equity\_bv\_adj[i]

else:

eq = avg\_equity

try:

roe.append(net\_income\_adj[i] / eq)

except:

roe.append(0)

return roe, roc

def get\_target\_info(revenue, ttm\_revenue, country\_default\_spread, tax\_rate, final\_erp, riskfree,

unlevered\_beta, damodaran\_bond\_spread, company\_default\_spread, target\_debt\_equity):

cagr = None

if abs(ttm\_revenue/revenue[-1] - 1) > 0.0001:

ttm = True

else:

ttm = False

if ttm:

rev\_list = revenue + [ttm\_revenue]

else:

rev\_list = revenue

first\_index = -1

if rev\_list[0] > 0:

first\_index = 0

first\_revenue = rev\_list[0]

elif rev\_list[1] > 0:

first\_index = 1

first\_revenue = rev\_list[1]

elif rev\_list[2] > 0:

first\_index = 2

first\_revenue = rev\_list[2]

else:

cagr = 0

print("error CAGR 0 - no revenue first 3 years")

if first\_index >= 0:

for i in rev\_list[first\_index:]:

if i <= 0:

cagr = 0

if cagr is None:

years\_diff = -1

for i in rev\_list:

if i > 0:

years\_diff += 1

# Simple CAGR

simple\_cagr = (rev\_list[-1] / first\_revenue) \*\* (1/(years\_diff)) - 1

capped\_simple\_cagr = max(min(simple\_cagr,0.3),-0.2)

# CAGR from start

cagr\_from\_start\_list = []

for i in range(1, years\_diff+1):

cagr\_from\_start\_list.append((rev\_list[first\_index+i] / first\_revenue) \*\* (1/i) - 1)

abs\_cagr\_from\_start = [abs(x) for x in cagr\_from\_start\_list]

cagr\_from\_start\_sorted = [x for \_, x in sorted(zip(abs\_cagr\_from\_start, cagr\_from\_start\_list), reverse=True)]

value\_sum, weight\_sum = (0,0)

for idx, value in enumerate(cagr\_from\_start\_sorted):

weight = 2\*\*idx

weight\_sum += weight

value\_sum += value \* weight

cagr\_from\_start = value\_sum / weight\_sum

capped\_cagr\_from\_start = max(min(cagr\_from\_start,0.3),-0.2)

# CAGR from end

cagr\_from\_end\_list = []

for i in range(years\_diff):

cagr\_from\_end\_list.append((rev\_list[-1] / rev\_list[first\_index+i]) \*\* (1 / (years\_diff-i)) - 1)

abs\_cagr\_from\_end = [abs(x) for x in cagr\_from\_end\_list]

cagr\_from\_end\_sorted = [x for \_, x in sorted(zip(abs\_cagr\_from\_end, cagr\_from\_end\_list), reverse=True)]

value\_sum, weight\_sum = (0, 0)

for idx, value in enumerate(cagr\_from\_end\_sorted):

weight = 2 \*\* idx

weight\_sum += weight

value\_sum += value \* weight

cagr\_from\_end = value\_sum / weight\_sum

capped\_cagr\_from\_end = max(min(cagr\_from\_end, 0.3), -0.2)

# print("rev\_list", rev\_list)

# print("first\_revenue", first\_revenue)

# print("simple\_cagr", simple\_cagr)

# print("capped\_simple\_cagr", capped\_simple\_cagr)

# print("cagr\_from\_start\_list", cagr\_from\_start\_list)

# print("cagr\_from\_start", cagr\_from\_start)

# print("capped\_cagr\_from\_start", capped\_cagr\_from\_start)

# print("cagr\_from\_end\_list", cagr\_from\_end\_list)

# print("cagr\_from\_end", cagr\_from\_end)

# print("capped\_cagr\_from\_end", capped\_cagr\_from\_end)

cagr\_3\_values = [capped\_simple\_cagr, capped\_cagr\_from\_start, capped\_cagr\_from\_end]

cagr\_3\_values.sort(reverse=True)

value\_sum, weight\_sum = (0, 0)

for idx, value in enumerate(cagr\_3\_values):

weight = 2 \*\* idx

weight\_sum += weight

value\_sum += value \* weight

cagr = value\_sum / weight\_sum

spread\_list = list(damodaran\_bond\_spread["spread"].unique())

spread\_list = [float(x) for x in spread\_list]

spread\_list.sort()

debt\_improvement\_offset = 2

idx = spread\_list.index(company\_default\_spread)

idx -= debt\_improvement\_offset

if idx < 0:

idx = 0

target\_company\_default\_spread = float(spread\_list[idx])

target\_levered\_beta = unlevered\_beta \* (1+ (1-tax\_rate) \* target\_debt\_equity)

target\_cost\_of\_equity = riskfree + final\_erp \* target\_levered\_beta

target\_cost\_of\_debt = riskfree + country\_default\_spread + target\_company\_default\_spread

target\_cost\_of\_capital = target\_cost\_of\_debt \* (1-tax\_rate) \* target\_debt\_equity / (target\_debt\_equity + 1) + \

target\_cost\_of\_equity \* 1 / (1 + target\_debt\_equity)

return cagr, target\_levered\_beta, target\_cost\_of\_equity, target\_cost\_of\_debt, target\_cost\_of\_capital

def currency\_bond\_yield(currency, alpha\_3\_code, country\_stats):

currency\_10y\_bond, mother\_country = get\_10y\_bond\_yield(currency)

if currency\_10y\_bond is not None:

filter\_df = country\_stats[country\_stats["country"] == mother\_country.replace(" ", "")].iloc[0]

country\_default\_spread = float(filter\_df["adjusted\_default\_spread"])

#10y yield currency - default risk mother currency

riskfree = currency\_10y\_bond - country\_default\_spread

else:

if alpha\_3\_code is None:

return -1

us\_10y\_bond, \_ = get\_10y\_bond\_yield("USD")

filter\_df = country\_stats[country\_stats["country"] == "UnitedStates"].iloc[0]

us\_cds = float(filter\_df["adjusted\_default\_spread"])

riskfree\_us = us\_10y\_bond - us\_cds

current\_year\_date = datetime.now().date().replace(day=1) - relativedelta(months=2)

last\_year\_date = current\_year\_date - relativedelta(years=1)

cpi\_data = get\_df\_from\_table("oecd\_financial", f"where location IN ('{alpha\_3\_code}','USA') "

f"and indicator='CPI' "

f"and date in ('{last\_year\_date.strftime('%Y-%m-%d')}',"

f"'{current\_year\_date.strftime('%Y-%m-%d')}')")

inflation\_us = cpi\_data[cpi\_data["location"] == "USA"]

inflation\_us = inflation\_us[inflation\_us["date"] == current\_year\_date]["value"].iloc[0] / \

inflation\_us[inflation\_us["date"] == last\_year\_date]["value"].iloc[0] - 1

inflation\_country = cpi\_data[cpi\_data["location"] == alpha\_3\_code]

if inflation\_country.empty:

inflation\_country = inflation\_us

else:

inflation\_country = inflation\_country[inflation\_country["date"] == current\_year\_date]["value"].iloc[0] / \

inflation\_country[inflation\_country["date"] == last\_year\_date]["value"].iloc[0] - 1

riskfree = riskfree\_us \* float(inflation\_country) / float(inflation\_us)

print("10y bond yield not found - inflation\_country", inflation\_country, "inflation\_us", inflation\_us)

return riskfree

def get\_normalized\_info(revenue, ebit\_adj, revenue\_delta, reinvestment, target\_sales\_capital,

ebit\_after\_tax, industry\_payout, cagr, net\_income\_adj, roe, dividends, eps\_adj, roc):

weights = [2\*\*x for x in range(len(revenue))]

sum\_weights = sum(weights)

revenue\_5y = sum(i[0] \* i[1] for i in zip(revenue, weights)) / sum\_weights

ebit\_5y = sum(i[0] \* i[1] for i in zip(ebit\_adj, weights)) / sum\_weights

try:

operating\_margin\_5y = ebit\_5y / revenue\_5y

except:

operating\_margin\_5y = 0

# print("## SALES CAPITAL ##")

# print(revenue\_delta)

# print(sum(revenue\_delta))

# print(reinvestment)

# print(sum(reinvestment))

# print(sum(revenue\_delta) / sum(reinvestment))

# print("target", target\_sales\_capital)

try:

sales\_capital\_5y = sum(revenue\_delta) / sum(reinvestment)

if sales\_capital\_5y <= 0:

sales\_capital\_5y = target\_sales\_capital

except:

sales\_capital\_5y = target\_sales\_capital

roc\_5y = sum(i[0] \* i[1] for i in zip(roc, weights)) / sum\_weights

try:

reinvestment\_5y = sum(reinvestment) / sum(ebit\_after\_tax)

if reinvestment\_5y <= 0:

reinvestment\_5y = 1 - industry\_payout

except:

reinvestment\_5y = 1 - industry\_payout

try:

growth\_5y = roc\_5y \* reinvestment\_5y

except:

growth\_5y = cagr

net\_income\_5y = sum(i[0] \* i[1] for i in zip(net\_income\_adj, weights)) / sum\_weights

roe\_5y = sum(i[0] \* i[1] for i in zip(roe, weights)) / sum\_weights

try:

reinvestment\_eps\_5y = 1 - sum(dividends) / sum(eps\_adj)

except:

reinvestment\_eps\_5y = reinvestment\_5y

growth\_eps\_5y = roe\_5y \* reinvestment\_eps\_5y

return revenue\_5y, ebit\_5y, operating\_margin\_5y, sales\_capital\_5y, roc\_5y, reinvestment\_5y, growth\_5y, \

net\_income\_5y, roe\_5y, reinvestment\_eps\_5y, growth\_eps\_5y

def get\_dividends\_info(eps\_adj, dividends):

weights = [2 \*\* x for x in range(len(eps\_adj))]

sum\_weights = sum(weights)

eps\_5y = sum(i[0] \* i[1] for i in zip(eps\_adj, weights)) / sum\_weights

try:

payout\_5y = sum(dividends) / sum(eps\_adj)

except:

payout\_5y = 0

if payout\_5y < 0:

payout\_5y = 0

return eps\_5y, payout\_5y

def get\_final\_info(riskfree, cost\_of\_debt, equity\_mkt, debt\_mkt, unlevered\_beta,

tax\_rate, final\_erp, company\_default\_spread):

survival\_prob = (1 - company\_default\_spread) \*\* 10

try:

debt\_equity = debt\_mkt / equity\_mkt

except:

debt\_equity = 0.5

levered\_beta = unlevered\_beta \* (1 + (1 - tax\_rate) \* debt\_equity)

cost\_of\_equity = riskfree + levered\_beta \* final\_erp

try:

equity\_weight = equity\_mkt / (equity\_mkt + debt\_mkt)

except:

equity\_weight = 0.5

debt\_weight = 1 - equity\_weight

cost\_of\_capital = cost\_of\_equity \* equity\_weight + cost\_of\_debt \* (1 - tax\_rate) \* debt\_weight

return survival\_prob, debt\_equity, \

levered\_beta, cost\_of\_equity, equity\_weight, debt\_weight, cost\_of\_capital

def calculate\_liquidation\_value(cash, receivables, inventory, securities, other\_current\_assets, mr\_property, ppe,

equity\_investments, total\_liabilities, equity\_mkt, mr\_debt, mr\_equity, mr\_original\_min\_interest,

minority\_interest, debug=True):

percent\_minority\_interest = minority\_interest / equity\_mkt

# market\_liquidation = equity\_mkt + debt\_mkt - mr\_debt

# if market\_liquidation < 0:

# market\_liquidation = 0

damodaran\_liquidation = cash + securities + mr\_property + (other\_current\_assets + inventory + receivables + ppe) \* 0.75 + \

equity\_investments \* 0.5 - total\_liabilities

if damodaran\_liquidation < 0:

damodaran\_liquidation = 0

net\_net\_wc\_liquidation = cash + receivables + inventory + securities + mr\_property + other\_current\_assets - total\_liabilities

if net\_net\_wc\_liquidation < 0:

net\_net\_wc\_liquidation = 0

\_sorted = sorted([damodaran\_liquidation, net\_net\_wc\_liquidation], reverse=True)

value\_sum, weight\_sum = (0, 0)

for idx, value in enumerate(\_sorted):

weight = 2 \*\* idx

weight\_sum += weight

value\_sum += value \* weight

liquidation\_value = value\_sum / weight\_sum \* (1-percent\_minority\_interest)

if debug:

print("===== Liquidation Value =====\n")

print("cash", cash)

print("securities", securities)

print("receivables", receivables)

print("inventory", inventory)

print("other\_current\_assets\_ms", other\_current\_assets)

print("property", mr\_property)

print("ppe", ppe)

print("equity\_investments", equity\_investments)

print()

print("total\_liabilities", total\_liabilities)

# print("percent\_minority\_interest", percent\_minority\_interest)

# print("equity\_mkt", equity\_mkt)

# print("debt\_mkt", debt\_mkt)

print("debt\_bv", mr\_debt)

print("equity\_bv", mr\_equity)

print("minority\_interest", mr\_original\_min\_interest, "=>", minority\_interest)

# print("market\_liquidation", market\_liquidation)

print("damodaran\_liquidation", damodaran\_liquidation)

print("net\_net\_wc\_liquidation", net\_net\_wc\_liquidation)

print("liquidation\_value", liquidation\_value)

print("\n\n")

return liquidation\_value

def get\_industry\_parameter(df, industry, region, parameter, debug=True):

region\_waterfall = {

"Europe": "US",

"US": "Global",

"Japan": "Global",

"China": "emerg",

"India": "emerg",

"emerg": "Global",

"Rest": "US"

}

value = None

try:

series = df[df["region"] == region].iloc[0]

value = series[parameter]

except:

pass

# print(industry, region, parameter, value)

if value is None or math.isnan(value):

if region in region\_waterfall:

if debug:

print("value not found for ", industry, region, parameter)

print("searching now in region ", region\_waterfall[region])

return get\_industry\_parameter(df, industry, region\_waterfall[region], parameter)

else:

print("\*\*\* ERROR DAMODARAN\_INDUSTRY\_DATA: ", industry, region, parameter)

if parameter == "sales\_capital":

value = 1

elif parameter == "cash\_return":

value = 0

elif parameter == "pbv":

value = 1

elif parameter == "unlevered\_beta":

value = 1

elif parameter == "opmargin\_adjusted":

value = 0.05

elif parameter == "debt\_equity":

value = 1

else:

value = 0

print("using default value ", value)

return value

else:

return float(value)

def get\_industry\_data(industry, region, geo\_segments\_df, revenue, ebit\_adj, revenue\_delta, reinvestment, equity\_mkt, debt\_mkt,

equity\_bv\_adj, debt\_bv\_adj, mr\_equity\_adj, mr\_debt\_adj, min\_std=0.1, max\_std=1):

# TAKE 1/3 value from last year

# 2/3 value from this year

columns = ["industry\_name","region","sales\_capital","cash\_return","unlevered\_beta","opmargin\_adjusted","debt\_equity","pbv"]

df\_last\_year = get\_df\_from\_table("damodaran\_industry\_data", f"where industry\_name = '{industry}' and "

f"created\_at = (SELECT MAX(created\_at) "

f"FROM damodaran\_industry\_data "

f"WHERE created\_at < date\_trunc('year',now()))")[columns]

df = get\_df\_from\_table("damodaran\_industry\_data", f"where industry\_name = '{industry}'", most\_recent=True)[columns]

# print(df\_last\_year)

# print(df)

df = pd.merge(df, df\_last\_year, left\_on=["industry\_name", 'region'],

right\_on=["industry\_name", 'region'], how="left")

for x in [i for i in columns if i not in ["industry\_name","region"]]:

# if there is no last year value take the most recent

df[x+"\_y"] = df[x+"\_y"].fillna(df[x+"\_x"])

# TAKE 1/3 value from last year

# 2/3 value from this year

df[x] = df[x+"\_y"] \* 1/3 + df[x+"\_x"] \* 2/3

industry\_sales\_capital = 0

industry\_payout = 0

pbv = 0

unlevered\_beta = 0

industry\_operating\_margin = 0

industry\_debt\_equity = 0

debug=True

if geo\_segments\_df is None or geo\_segments\_df.empty:

industry\_sales\_capital = get\_industry\_parameter(df, industry, region, "sales\_capital", debug=debug)

industry\_payout = min(1, get\_industry\_parameter(df, industry, region, "cash\_return", debug=debug))

pbv = get\_industry\_parameter(df, industry, region, "pbv", debug=debug)

unlevered\_beta = get\_industry\_parameter(df, industry, region, "unlevered\_beta", debug=debug)

industry\_operating\_margin = get\_industry\_parameter(df, industry, region, "opmargin\_adjusted", debug=debug)

industry\_debt\_equity = get\_industry\_parameter(df, industry, region, "debt\_equity", debug=debug)

else:

for \_, row in geo\_segments\_df.iterrows():

percent = row["value"]

r = row["region"]

tsc = get\_industry\_parameter(df, industry, r, "sales\_capital", debug=debug)

ip = min(1, get\_industry\_parameter(df, industry, r, "cash\_return", debug=debug))

p = get\_industry\_parameter(df, industry, r, "pbv", debug=debug)

ub = get\_industry\_parameter(df, industry, r, "unlevered\_beta", debug=debug)

tom = get\_industry\_parameter(df, industry, r, "opmargin\_adjusted", debug=debug)

tde = get\_industry\_parameter(df, industry, r, "debt\_equity", debug=debug)

industry\_sales\_capital += tsc \* percent

industry\_payout = min(1, industry\_payout + ip \* percent)

pbv += p \* percent

unlevered\_beta += ub \* percent

industry\_operating\_margin += tom \* percent

industry\_debt\_equity += tde \* percent

operating\_margin = []

debt\_equity = []

sales\_capital = []

try:

sales\_capital\_5y = sum(revenue\_delta) / sum(reinvestment)

if sales\_capital\_5y <= 0:

sales\_capital\_5y = industry\_sales\_capital

except:

sales\_capital\_5y = industry\_sales\_capital

for i in range(len(revenue)):

if revenue[i] > 0:

operating\_margin.append(ebit\_adj[i] / revenue[i])

else:

operating\_margin.append(0)

num = (debt\_bv\_adj[i] \* (debt\_mkt/mr\_debt\_adj)) if mr\_debt\_adj > 0 else debt\_bv\_adj[i]

den = (equity\_bv\_adj[i] \* (equity\_mkt/mr\_equity\_adj)) if mr\_equity\_adj > 0 else equity\_bv\_adj[i]

if den > 0 and num / den > 0:

debt\_equity.append(num/den)

else:

debt\_equity.append(0)

try:

if revenue\_delta[i] / reinvestment[i] > 0:

sales\_capital.append(revenue\_delta[i] / reinvestment[i])

else:

sales\_capital.append(sales\_capital\_5y)

except:

sales\_capital.append(sales\_capital\_5y)

weights = [x+1 for x in range(len(revenue))]

sum\_weights = sum(weights)

# print(debt\_bv\_adj)

# print(debt\_mkt)

# print(mr\_debt\_adj)

# print(equity\_bv\_adj)

# print(equity\_mkt)

# print(mr\_equity\_adj)

# print(debt\_equity)

om\_company = sum(i[0] \* i[1] for i in zip(operating\_margin, weights)) / sum\_weights

de\_company = sum(i[0] \* i[1] for i in zip(debt\_equity, weights)) / sum\_weights

sc\_company = sum(i[0] \* i[1] for i in zip(sales\_capital, weights)) / sum\_weights

std\_om\_company = np.std(operating\_margin)

std\_de\_company = np.std(debt\_equity)

std\_sc\_company = np.std(sales\_capital)

if om\_company != 0:

om\_industry\_weight = max(0, min(1, ((std\_om\_company / om\_company) - min\_std) / (max\_std - min\_std)))

else:

om\_industry\_weight = 1

if de\_company != 0:

de\_industry\_weight = max(0, min(1, ((std\_de\_company / de\_company) - min\_std) / (max\_std - min\_std)))

else:

de\_industry\_weight = 1

if sc\_company != 0:

sc\_industry\_weight = max(0, min(1, ((std\_sc\_company / sc\_company) - min\_std) / (max\_std - min\_std)))

else:

sc\_industry\_weight = 1

target\_sales\_capital = sc\_industry\_weight \* industry\_sales\_capital + (1 - sc\_industry\_weight) \* sc\_company

target\_debt\_equity = de\_industry\_weight \* industry\_debt\_equity + (1 - de\_industry\_weight) \* de\_company

target\_operating\_margin = om\_industry\_weight \* industry\_operating\_margin + (1 - om\_industry\_weight) \* om\_company

# print("DEBUG TARGETS")

# print(sales\_capital)

# print(operating\_margin)

# print(debt\_equity)

# print("sc\_company",sc\_company,"industry\_sales\_capital",industry\_sales\_capital,"std\_sc\_company",std\_sc\_company,"sc\_industry\_weight",sc\_industry\_weight,"target\_sales\_capital",target\_sales\_capital)

# print("om\_company",om\_company,"industry\_operating\_margin",industry\_operating\_margin,"std\_om\_company",std\_om\_company,"om\_industry\_weight",om\_industry\_weight,"target\_operating\_margin",target\_operating\_margin)

# print("de\_company",de\_company,"industry\_debt\_equity",industry\_debt\_equity,"std\_de\_company",std\_de\_company,"de\_industry\_weight",de\_industry\_weight,"target\_debt\_equity",target\_debt\_equity)

return target\_sales\_capital, industry\_payout, pbv, unlevered\_beta, target\_operating\_margin, target\_debt\_equity

EARNINGS\_TTM = "EARNINGS\_TTM"

EARNINGS\_NORM = "EARNINGS\_NORM"

GROWTH\_FIXED = "GROWTH\_FIXED"

GROWTH\_TTM = "GROWTH\_TTM"

GROWTH\_NORM = "GROWTH\_NORM"

def dividends\_valuation(earnings\_type, growth\_type, cagr, growth\_eps\_5y, growth\_5y, riskfree,

industry\_payout, cost\_of\_equity, target\_cost\_of\_equity,

growth\_eps\_last, eps\_5y, payout\_5y, ttm\_eps\_adj, reinvestment\_eps\_last, fx\_rate, survival\_prob,

liquidation\_per\_share, debug=True, recession=False, dict\_values\_for\_bi=None):

final\_growth = riskfree

if growth\_5y != 0:

final\_growth = riskfree \* growth\_eps\_5y / growth\_5y

if final\_growth <= 0:

final\_growth = riskfree

else:

final\_growth = riskfree \* max(min(growth\_eps\_5y / growth\_5y, 2), 0.5)

if growth\_type == GROWTH\_FIXED:

if growth\_5y == 0:

initial\_growth = cagr

else:

initial\_growth = cagr \* growth\_eps\_5y / growth\_5y

if initial\_growth <= 0:

initial\_growth = cagr

else:

initial\_growth = cagr \* max(min(growth\_eps\_5y / growth\_5y, 2),0.5)

elif growth\_type == GROWTH\_TTM:

initial\_growth = growth\_eps\_last

else:

initial\_growth = growth\_eps\_5y

growth\_history = np.linspace(initial\_growth, final\_growth, 11)

if recession:

growth\_history[3:6] = 0

if earnings\_type == EARNINGS\_TTM:

initial\_eps = ttm\_eps\_adj

else:

initial\_eps = eps\_5y

eps\_history = []

for i in range(len(growth\_history)):

if initial\_eps < 0 and i < 6:

if i == 0:

eps\_history.append(initial\_eps + abs(initial\_eps) / 5)

else:

eps\_history.append(eps\_history[i-1] + abs(initial\_eps) / 5)

else:

if i == 0:

eps\_history.append(initial\_eps \* (1+growth\_history[i]))

else:

eps\_history.append(eps\_history[i-1] \* (1+growth\_history[i]))

if earnings\_type == EARNINGS\_TTM:

try:

initial\_payout = 1 - reinvestment\_eps\_last

except:

initial\_payout = payout\_5y

else:

initial\_payout = payout\_5y

final\_payout = industry\_payout

payout\_history = np.linspace(initial\_payout, final\_payout, 12)

payout\_history = payout\_history[1:]

if recession:

payout\_history[3:6] = 0

dps\_history = []

for i in range(len(eps\_history)):

dps\_history.append(eps\_history[i] \* payout\_history[i])

initial\_coe = cost\_of\_equity

final\_coe = target\_cost\_of\_equity

cost\_of\_equity\_history = np.linspace(initial\_coe, final\_coe, 12)

cost\_of\_equity\_history = cost\_of\_equity\_history[1:]

cumulative\_coe = []

for i in range(len(cost\_of\_equity\_history)):

if i == 0:

cumulative\_coe.append(1+cost\_of\_equity\_history[i])

else:

cumulative\_coe.append(cumulative\_coe[i-1] \* (1+cost\_of\_equity\_history[i]))

present\_value = []

for i in range(len(cumulative\_coe)-1):

present\_value.append(dps\_history[i] / cumulative\_coe[i])

terminal\_value = eps\_history[-1] \* payout\_history[-1] / \

(cost\_of\_equity\_history[-1] - growth\_history[-1])

terminal\_pv = terminal\_value / cumulative\_coe[-1]

stock\_value\_price\_curr = sum(present\_value) + terminal\_pv

if fx\_rate is not None:

stock\_value = stock\_value\_price\_curr \* fx\_rate

else:

stock\_value = stock\_value\_price\_curr

stock\_value = stock\_value \* survival\_prob + liquidation\_per\_share \* (1 - survival\_prob)

if dict\_values\_for\_bi is not None:

dict\_values\_for\_bi[f"{earnings\_type} + {growth\_type} + recession:{recession}"] = {

"eps": eps\_history,

"dividends": dps\_history,

"cost\_of\_equity": cost\_of\_equity\_history,

"pv\_of\_dividends": present\_value + [terminal\_pv]

}

if debug:

for i in [growth\_history, eps\_history, payout\_history, dps\_history, cost\_of\_equity\_history, cumulative\_coe, present\_value]:

for idx, j in enumerate(i):

i[idx] = round(j,4)

print(f"===== Dividends Valuation - {earnings\_type} + {growth\_type} + recession:{recession} =====\n")

print("expected\_growth", growth\_history)

print("earnings\_per\_share", eps\_history)

print("payout\_ratio", payout\_history)

print("dividends\_per\_share", dps\_history)

print("cost\_of\_equity", cost\_of\_equity\_history)

print("cumulative\_cost\_equity", cumulative\_coe)

print("present\_value",present\_value)

print("terminal\_value",round(terminal\_value,2))

print("PV of terminal\_value", round(terminal\_pv,2))

print("stock value (price curr)", round(stock\_value\_price\_curr,2))

print("stock value (fin curr)", round(stock\_value, 2))

print("\n\n")

return stock\_value

def fcff\_valuation(earnings\_type, growth\_type, cagr, riskfree, ttm\_revenue, ttm\_ebit\_adj, target\_operating\_margin, tax\_benefits,

tax\_rate, sales\_capital\_5y, target\_sales\_capital, debt\_equity, target\_debt\_equity, unlevered\_beta,

final\_erp, cost\_of\_debt, target\_cost\_of\_debt, mr\_cash, mr\_securities, debt\_mkt, minority\_interest, survival\_prob,

share\_issued, ko\_proceeds, growth\_last, growth\_5y, revenue\_5y, ebit\_5y, fx\_rate, mr\_property, mr\_sbc, debug=True,

recession=False, dict\_values\_for\_bi=None):

# earnings ttm + growth fixed

if growth\_type == GROWTH\_FIXED:

initial\_growth = cagr

elif growth\_type == GROWTH\_TTM:

initial\_growth = growth\_last

else:

initial\_growth = growth\_5y

final\_growth = riskfree

growth\_history = np.linspace(initial\_growth, final\_growth, 11)

if recession:

growth\_history[3] = -0.1

growth\_history[4] = -0.2

growth\_history[5] = 0.4

if earnings\_type == EARNINGS\_TTM:

initial\_revenue = ttm\_revenue

else:

initial\_revenue = revenue\_5y

revenue\_history = []

for i in range(len(growth\_history)):

if i == 0:

revenue\_history.append(initial\_revenue \* (1+growth\_history[i]))

else:

revenue\_history.append(revenue\_history[i-1] \* (1+growth\_history[i]))

if earnings\_type == EARNINGS\_TTM:

if ttm\_revenue == 0:

initial\_margin = 0

else:

initial\_margin = ttm\_ebit\_adj / ttm\_revenue

else:

if revenue\_5y == 0:

initial\_margin = 0

else:

initial\_margin = ebit\_5y / revenue\_5y

final\_margin = target\_operating\_margin

margin\_history = np.linspace(initial\_margin, final\_margin, 12)[1:]

if initial\_margin < 0:

check\_margin = initial\_margin / 5

if check\_margin \* 4 > margin\_history[0]:

for i in range(len(margin\_history)):

if i < 5:

margin\_history[i] = initial\_margin - check\_margin \* (i+1)

else:

margin\_history[i] = final\_margin / 6 \* (i-4)

if recession:

margin\_history[3] \*= 0.5

margin\_history[4] \*= 0.25

margin\_history[5] \*= 0.5

ebit\_history = []

for i in range(len(revenue\_history)):

ebit\_history.append(revenue\_history[i] \* margin\_history[i])

residual\_tax\_benefits = tax\_benefits

tax\_history = []

ebit\_after\_tax\_history = []

for i in range(len(ebit\_history)):

e = ebit\_history[i]

if e < 0:

tax\_history.append(0)

else:

tax\_history.append(max(0, (e \* tax\_rate) - residual\_tax\_benefits))

residual\_tax\_benefits = max(0, residual\_tax\_benefits - (e \* tax\_rate))

ebit\_after\_tax\_history.append(ebit\_history[i] - tax\_history[i])

initial\_sales\_capital = sales\_capital\_5y

final\_sales\_capital = target\_sales\_capital

sales\_capital\_history = np.linspace(initial\_sales\_capital, final\_sales\_capital, 12)[1:]

reinvestment\_history = []

fcff\_history = []

for i in range(len(revenue\_history)):

if i == 0:

delta\_revenue = revenue\_history[i] - initial\_revenue

else:

delta\_revenue = revenue\_history[i] - revenue\_history[i-1]

reinvestment\_history.append(delta\_revenue / sales\_capital\_history[i])

if recession and i in [3,4]:

reinvestment\_history[i] = reinvestment\_history[i-1]

fcff\_history.append(ebit\_after\_tax\_history[i] - reinvestment\_history[i])

# initial\_debt\_ratio = debt\_weight

# final\_debt\_ratio = target\_debt\_equity / (1+target\_debt\_equity)

# debt\_ratio\_history = np.linspace(initial\_debt\_ratio, final\_debt\_ratio, 12)[1:]

debt\_equity\_history = np.linspace(debt\_equity, target\_debt\_equity, 12)[1:]

initial\_cost\_of\_debt = cost\_of\_debt

final\_cost\_of\_debt = target\_cost\_of\_debt

cost\_of\_debt\_history = np.linspace(initial\_cost\_of\_debt, final\_cost\_of\_debt, 12)[1:]

debt\_ratio\_history = []

beta\_history = []

cost\_of\_equity\_history = []

cost\_of\_capital\_history = []

cumulative\_wacc\_history = []

present\_value\_history = []

for i in range(len(debt\_equity\_history)):

debt\_ratio\_history.append(debt\_equity\_history[i] / (debt\_equity\_history[i] + 1))

# debt\_equity\_history.append(debt\_ratio\_history[i] / (1-debt\_ratio\_history[i]))

beta\_history.append(unlevered\_beta \* (1+(1-tax\_rate)\*debt\_equity\_history[i]))

cost\_of\_equity\_history.append(riskfree + final\_erp \* beta\_history[i])

cost\_of\_capital\_history.append(cost\_of\_equity\_history[i] \* (1-debt\_ratio\_history[i])

+ cost\_of\_debt\_history[i] \* (1-tax\_rate) \* debt\_ratio\_history[i])

if i == 0:

cumulative\_wacc\_history.append(1 + cost\_of\_capital\_history[i])

else:

cumulative\_wacc\_history.append(cumulative\_wacc\_history[i-1] \* (1 + cost\_of\_capital\_history[i]))

present\_value\_history.append(fcff\_history[i] / cumulative\_wacc\_history[i])

terminal\_value = fcff\_history[-1] / (cost\_of\_capital\_history[-1] - growth\_history[-1])

terminal\_value\_pv = terminal\_value / cumulative\_wacc\_history[-1]

firm\_value = sum(present\_value\_history[:-1]) + terminal\_value\_pv + mr\_cash + mr\_securities + mr\_property

equity\_value = firm\_value - debt\_mkt - minority\_interest - mr\_sbc

try:

stock\_value\_price\_curr = (equity\_value \* survival\_prob + ko\_proceeds \* (1-survival\_prob)) / share\_issued

except:

stock\_value\_price\_curr = 0

if fx\_rate is not None:

stock\_value = stock\_value\_price\_curr \* fx\_rate

else:

stock\_value = stock\_value\_price\_curr

if dict\_values\_for\_bi is not None:

dict\_values\_for\_bi[f"{earnings\_type} + {growth\_type} + recession:{recession}"].update({

"revenue": revenue\_history,

"ebit": ebit\_history,

"ebit\_after\_tax": ebit\_after\_tax\_history,

"reinvestment": reinvestment\_history,

"FCFF": fcff\_history,

"cost\_of\_capital": cost\_of\_capital\_history,

"pv\_of\_FCFF": present\_value\_history[:-1] + [terminal\_value\_pv]

})

if debug:

# for i in [growth\_history, revenue\_history, margin\_history, ebit\_history, tax\_history, ebit\_after\_tax\_history,

# sales\_capital\_history, reinvestment\_history, fcff\_history, debt\_ratio\_history, cost\_of\_debt\_history,

# cost\_of\_equity\_history, cost\_of\_capital\_history, cumulative\_wacc\_history, present\_value\_history]:

# for idx, j in enumerate(i):

# i[idx] = round(j,4)

print(f"===== FCFF Valuation - {earnings\_type} + {growth\_type} recession:{recession} =====\n")

print("expected\_growth", growth\_history)

print("revenue", revenue\_history)

print("margin", margin\_history)

print("ebit", ebit\_history)

print("tax\_history", tax\_history)

print("ebit\_after\_tax", ebit\_after\_tax\_history)

print("sales\_capital", sales\_capital\_history)

print("reinvestment", reinvestment\_history)

print("FCFF", fcff\_history)

print("debt ratio", debt\_ratio\_history)

print("debt2equity", debt\_equity\_history)

print("beta", beta\_history)

print("cost\_of\_equity", cost\_of\_equity\_history)

print("cost\_of\_debt", cost\_of\_debt\_history)

print("cost\_of\_capital", cost\_of\_capital\_history)

print("cumulative WACC", cumulative\_wacc\_history)

print("present value", present\_value\_history)

print("terminal value", round(terminal\_value,2))

print("PV of FCFF during growth", sum(present\_value\_history[:-1]))

print("PV of terminal value", round(terminal\_value\_pv,2))

print("Value of operating assets", sum(present\_value\_history[:-1])+terminal\_value\_pv)

print("Value of cash and property", mr\_cash + mr\_securities + mr\_property)

print("firm value", round(firm\_value,2))

print("debt outstanding", round(debt\_mkt,2))

print("equity value", round(equity\_value,2))

print("stock value (price curr)", round(stock\_value\_price\_curr,2))

print("stock value (fin curr)", round(stock\_value, 2))

print("\n\n")

return stock\_value

def summary\_valuation(valuations):

sorted = valuations.copy()

sorted = [0 if math.isnan(x) else x for x in sorted]

sorted.sort()

count\_negative = 0

for val in sorted:

if val < 0:

count\_negative += 1

if count\_negative > 1:

result = 0

elif count\_negative > 0:

result = sorted[1]

else:

second\_highest = sorted[2]

third\_highest = sorted[1]

if third\_highest == 0 or second\_highest / third\_highest > 10:

result = 0

else:

max\_val = max(sorted)

min\_val = min(sorted)

if min\_val == 0 or max\_val / min\_val > 3:

result = sorted[1]

else:

result = median(sorted)

return result

def get\_status(fcff\_delta, div\_delta, liquidation\_delta, country, region, company\_size, company\_type, dilution, complexity,

revenue, receivables=None, inventory=None, debug=False):

STATUS\_OK = "OK"

STATUS\_NI = "NI"

STATUS\_KO = "KO"

max\_base = 0.2

if liquidation\_delta < -0.5:

t = -max\_base

elif liquidation\_delta < 0:

t = liquidation\_delta / 0.5 \* max\_base

elif liquidation\_delta >= 10:

t = max\_base

else:

t = np.log10(liquidation\_delta + 1) \* max\_base

if debug:

print("base threshold", t, "(liquidation delta:", liquidation\_delta, ")")

if country == "United States":

t += 0

elif region in ["US","EU","Japan","Rest"]:

t += 0.05

else:

t += 0.1

if debug:

print("country/region threshold", t, "(country:", country, ", region:", region, ")")

if company\_size == "Mega":

t += 0

elif company\_size == "Large":

t += 0.05

elif company\_size == "Medium":

t += 0.12

elif company\_size == "Small":

t += 0.18

elif company\_size == "Micro":

t += 0.25

else:

t += 0.3

if debug:

print("company size threshold", t, "(company size:", company\_size, ")")

elif complexity == 2:

t += 0.03

elif complexity == 3:

t += 0.06

elif complexity == 4:

t += 0.1

elif complexity == 5:

t += 0.2

if debug:

print("company complexity threshold", t, "(complexity:", complexity, ")")

if dilution > 0.1:

t += 0.05

elif dilution > 0.02:

t += 0.02

if debug:

print("dilution threshold", t, "(dilution:", dilution, ")")

first\_idx = 0

for i, r in enumerate(revenue):

if r > 0:

first\_idx = i

break

ratio, score = calculate\_divergence(revenue[first\_idx], inventory[first\_idx], revenue[-1], inventory[-1])

if score is None:

if ratio < 10:

t += 0.02

elif ratio < 5:

t += 0.05

else:

if score > 0.8 or score < -0.4:

t += 0.05

elif score > 0.4 or score < -0.2:

t += 0.02

if debug:

print("inventory divergence threshold", t, "(inventory:", inventory, ", revenue:", revenue, ")")

ratio, score = calculate\_divergence(revenue[first\_idx], receivables[first\_idx], revenue[-1], receivables[-1])

if score is None:

if ratio < 10:

t += 0.02

elif ratio < 5:

t += 0.05

else:

if score > 0.8 or score < -0.4:

t += 0.05

elif score > 0.4 or score < -0.2:

t += 0.02

if debug:

print("receivables divergence threshold", t, "(receivables:", receivables, ", revenue:", revenue, ")")

if company\_type["declining"]:

t += 0.05

if company\_type["turn\_around"]:

t += 0.05

if company\_type["cyclical"]:

t += 0.05

if debug:

print("company type threshold", t, "(company\_type:", company\_type, ")")

if fcff\_delta < -t:

if div\_delta < -t:

status = STATUS\_OK

else:

status = STATUS\_NI

elif fcff\_delta < 0:

if div\_delta < 0:

status = STATUS\_NI

else:

status = STATUS\_KO

else:

if div\_delta < -t:

status = STATUS\_NI

else:

status = STATUS\_KO

if debug:

print("status", status)

print()

return status

import math

def calculate\_divergence(initial\_a, initial\_b, final\_a, final\_b):

try:

ratio = max(initial\_a, final\_a) / max(initial\_b, final\_b)

except:

return 0, 0

try:

growth\_a = final\_a / initial\_a

growth\_b = final\_b / initial\_b

growth\_ratio = growth\_b / growth\_a - 1

except:

return ratio, None

# print(initial\_a, "=>", final\_a, "|", initial\_b, "=>", final\_b, "ratio:", ratio, "growth\_ratio:", growth\_ratio, "result:", growth\_ratio/ratio)

return ratio, growth\_ratio / ratio

# A growing

# < -0.5 = risky

# > 0.4 = risky

# > 2 = bad

# A declining

# < -2 = bad

# < 0.4 risky

if \_\_name\_\_ == '\_\_main\_\_':

initial\_a = 100

final\_a = [0]

initial\_b = [100]

final\_b = [

[50, 0]

]

for f\_a in final\_a:

for idx, i\_b in enumerate(initial\_b):

for f\_b in final\_b[idx]:

divergence = calculate\_divergence(initial\_a, i\_b, f\_a, f\_b)

print()

print("=====")

import time

from datetime import datetime

from ssl import SSLError

import yfinance as yf

import requests

from bs4 import BeautifulSoup

from dateutil.relativedelta import relativedelta

from requests import ReadTimeout

from requests.exceptions import ChunkedEncodingError

from urllib3.exceptions import MaxRetryError, ReadTimeoutError

import traceback

packet\_stream\_proxy = "http://easymap\_buyer:34Qgo0O03zOhrx8h@proxy.packetstream.io:31112"

bright\_data\_proxy = 'http://brd-customer-hl\_f8b1a708-zone-finance:u7iz73qdf9wv@brd.superproxy.io:22225'

proxy = bright\_data\_proxy

def request\_yahoo\_url(url):

hed = {'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,\*/\*;q=0.8',

'Accept-Encoding': 'gzip, deflate, br', 'Accept-Language': 'it-IT,it;q=0.9,en-US;q=0.8,en;q=0.7',

'Upgrade-Insecure-Requests': '1',

'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_14\_1) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/12.0.1 Safari/605.1.15',

'Cache-Control': 'PUBLIC'}

cookies = {

"EuConsent": "CPUNe08PUNe08AOACBITB-CoAP\_AAH\_AACiQIJNe\_X\_\_bX9n-\_59\_\_t0eY1f9\_r3v-QzjhfNt-8F2L\_W\_L0H\_2E7NB36pq4KuR4ku3bBIQFtHMnUTUmxaolVrzHsak2MpyNKJ7LkmnsZe2dYGHtPn9lD-YKZ7\_7\_\_\_f73z\_\_\_9\_-39z3\_9f\_\_\_d9\_-\_\_-vjfV\_993\_\_\_\_\_\_\_\_9nd\_\_\_\_BBIAkw1LyALsSxwJNo0qhRAjCsJCoBQAUUAwtEVgAwOCnZWAT6ghYAITUBGBECDEFGDAIAAAIAkIiAkALBAIgCIBAACAFCAhAARMAgsALAwCAAUA0LEAKAAQJCDI4KjlMCAiRaKCWysQSgr2NMIAyywAoFEZFQgIlCCBYGQkLBzHAEgJYAYaADAAEEEhEAGAAIIJCoAMAAQQSA",

"OTH": "v=1&d=eyJraWQiOiIwMTY0MGY5MDNhMjRlMWMxZjA5N2ViZGEyZDA5YjE5NmM5ZGUzZWQ5IiwiYWxnIjoiUlMyNTYifQ.eyJjdSI6eyJndWlkIjoiWVZURENIQVJDVFFUSVM3WDVBN0g0NzZYVDQiLCJwZXJzaXN0ZW50Ijp0cnVlLCJzaWQiOiJNZm1Bc291aHZTbzIifX0.Qz8bX4q6yUmgNqoxVogtnln1kNlA5oc9hhMFm\_baVHvl2\_gnK5almd6r-u\_Wx4W9c9uhi2g9dvovheQr6DXlkGlG7Bw7OJubPeSGqy4asxOWAO4VNpUppmdK9kVuwOQIbnpg5skXXuGykmWRnUrtZH4resNBrOJhXgfUehIROpQ",

"GUC": "AQAABgFiBrFi6kIhUQUB",

"maex": "%7B%22v2%22%3A%7B%22106c4e0d%22%3A%7B%22lst%22%3A1644604741%2C%22ic%22%3A56%7D%7D%7D",

"UIDR": "1599641610",

"cmp": "v=22&t=1644604104&j=1",

"PRF": "t%3DAAPL%252BABTG.MI%252BFB%252B%255ESOX%252BMSFT%252BSLV%252BKO%252BGOOG%252BVT%252BREET%252BBNDW%252BFCT.MI%252BISP.MI%252BCRES.MI%252BTRASTOR.AT",

"A1S": "d=AQABBJDkvF8CEEnmdkmy3hsZxUP4oHXu3MoFEgAABgGxBmLqYudVb2UB9iMAAAcIf7Z8XzRwxloID4-gDUXX3Q7JnS7c59zqFwkBBwoBMA&S=AQAAAujaKZu-E9Ike-e7u6WnYmk&j=WORLD",

"B": "5lhjg6hfnpdjv&b=4&d=Jw.N6YdtYFmKelHVCZg9&s=cc&i=j6ANRdfdDsmdLtzn3OoX",

}

proxies = {

"http": proxy,

"https": proxy,

}

response = None

num\_retry = 0

max\_retry = 5

#return requests.get(url, headers=hed, proxies=proxies)

while ((response is None or response.status\_code == 403) and num\_retry < max\_retry):

if num\_retry > 0:

print("# retry = "+str(num\_retry)+", response = "+str(response)+", url = "+url)

if num\_retry > 0:

time.sleep(0.5 \* num\_retry)

try:

response = requests.get(url, headers=hed, proxies=proxies, timeout=20)

except requests.exceptions.SSLError:

pass

except requests.exceptions.ConnectionError:

pass

except ReadTimeout:

pass

except ChunkedEncodingError:

pass

except MaxRetryError:

pass

except ReadTimeoutError:

pass

except:

print(traceback.format\_exc())

num\_retry += 1

return response

def get\_premarket\_price\_yahoo(ticker):

url = f"https://finance.yahoo.com/quote/{ticker}"

response = request\_yahoo\_url(url)

# network\_size = len(response.content)

if response is not None:

if response.status\_code == 200:

soup = BeautifulSoup(response.text, 'lxml')

quote\_header = soup.find("div", id="quote-header-info")

if quote\_header is None:

return None#, network\_size

premarket = quote\_header.select\_one('fin-streamer[data-field="preMarketPrice"]')

if premarket is not None:

return float(premarket.text.replace(",",""))#, network\_size

regular = quote\_header.select\_one('fin-streamer[data-field="regularMarketPrice"]')

if regular is not None:

return float(regular.text.replace(",",""))#, network\_size

return None#, network\_size

def get\_current\_price\_from\_yahoo(ticker, created\_at=None):

if created\_at is None:

price = get\_premarket\_price\_yahoo(ticker)

if price is not None:

return price

else:

created\_at = datetime.now().date()

t = yf.Ticker(ticker)

todays\_data = None

max\_retry = 5

retry = 0

while todays\_data is None and retry < max\_retry:

try:

todays\_data = t.history(start=created\_at - relativedelta(days=5), end=created\_at,

interval="1m")

except:

print(f"{ticker} conn err - retry")

retry += 1

try:

return todays\_data['Close'][-1]

except IndexError:

return None

except TypeError:

return None

#### fundamentalytics

import requests

import json

import os.path

import pandas as pd

header = {'User-Agent': 'Fundamentalytics'}

def download\_tickers(filename:str='all\_tickers.json'):

url = 'https://www.sec.gov/files/company\_tickers.json'

response = requests.get(url, headers=header)

with open(filename, 'w') as f:

json.dump(response.json(), f)

def load\_tickers(filename:str='all\_tickers.json'):

if not os.path.isfile(filename):

download\_tickers()

with open(filename, 'r') as f:

all\_tickers = json.load(f)

return all\_tickers

def get\_all\_companies():

raw = load\_tickers()

all\_companies = []

for \_, data in raw.items():

all\_companies.append(str(data['ticker'] + ' ' + data['title'] + ' ' + str(data['cik\_str'])))

return all\_companies

def get\_company\_facts(cik: str):

if len(cik) < 10:

cik = (10 - len(cik)) \* '0' + cik

url = f'https://data.sec.gov/api/xbrl/companyfacts/CIK{cik}.json'

response = requests.get(url, headers=header).json()

result = {

'info': {

'cik': response['cik'],

'entityName': response['entityName']

},

'facts': {

}

}

facts = response['facts']['dei'] | response['facts']['us-gaap']

for k, v in facts.items():

if k in ['WeightedAverageNumberOfSharesOutstandingBasic', 'WeightedAverageNumberOfDilutedSharesOutstanding',

'Revenues', 'RevenueFromContractWithCustomerExcludingAssessedTax', 'OperatingIncomeLoss',

'IncomeLossFromContinuingOperationsBeforeIncomeTaxesExtraordinaryItemsNoncontrollingInterest',

'IncomeTaxExpenseBenefit', 'NetIncomeLoss', 'CostOfGoodsAndServicesSold', 'CashAndCashEquivalentsAtCarryingValue',

'CashCashEquivalentsAndShortTermInvestments', 'ResearchAndDevelopmentExpense', 'InventoryNet', 'AssetsCurrent',

'StockholdersEquity', 'Assets', 'LiabilitiesCurrent', 'EarningsPerShareBasic','EarningsPerShareDiluted', 'Liabilities']:

df = pd.DataFrame(v['units'][list(v['units'].keys())[0]])

temp\_dict = {

'label': v['label'],

'description': v['description'],

'unit': list(v['units'].keys())[0],

'data': df

}

result['facts'][k] = temp\_dict

del response

return result

### bulk data

Bulk Data Transfer

The SEC offers a bulk data download option for all of the data in both the Frame and Company Facts APIs. This data is updated nightly and can be accessed at http://www.sec.gov/Archives/edgar/daily-index/xbrl/companyfacts.zip. While all available data is provided (~14.5k files), only those companies in the S&P 500 are needed for this project. Due to the number of files that need to be selected and copied, it's necessary to create a script to automatically select the correct files.

The sp500\_ciks.csv data was sourced from the "S&P 500 component stocks" table on the List of S&P 500 companies Wikipedia page.

import shutil

import pandas as pd

df = pd.read\_csv('data/sp500\_ciks.csv', dtype=str)

df.head()

Symbol Security SEC filings GICS Sector GICS Sub-Industry Headquarters Location Date first added CIK Founded

0 MMM 3M reports Industrials Industrial Conglomerates Saint Paul, Minnesota 1976-08-09 0000066740 1902

1 ABT Abbott Laboratories reports Health Care Health Care Equipment North Chicago, Illinois 1964-03-31 0000001800 1888

2 ABBV AbbVie reports Health Care Pharmaceuticals North Chicago, Illinois 2012-12-31 0001551152 2013 (1888)

3 ABMD Abiomed reports Health Care Health Care Equipment Danvers, Massachusetts 2018-05-31 0000815094 1981

4 ACN Accenture reports Information Technology IT Consulting & Other Services Dublin, Ireland 2011-07-06 0001467373 1989

src = 'E:/Downloads/sec\_bulk\_data'

dst = 'data/sec\_bulk\_data/'

for cik in df.CIK:

try:

shutil.copy(src + f'/CIK{cik}.json', dst)

except FileNotFoundError as e:

print('File not found for CIK:', cik, f'({df[df.CIK == cik].Security.values})')

File not found for CIK: 0001132979 (['First Republic Bank'])

I manually confirmed that the CIK for First Republic Bank is correct as shown above. However, the SEC's API shows that the data is missing when trying to access it through the website (https://data.sec.gov/api/xbrl/companyfacts/CIK0001132979.json). Therefore, it makes sense that the data would also be missing from the bulk data download. 499 / 500 companies were successfully found and copied.