Final Home Assignment:

Maor Nave – ID.313603391, Tel – 0543121144

Date: 17/08/2023

**Full Code: IEEE Crawler programming, Data Processing and Analysis**

**Main.py File:**

from ActivationFunctions import \*  
import ActivationFunctions as af  
  
  
# Main function  
def main():  
 # Load general config for the assignment  
 af.data\_config = yaml\_loader('Input/config.yml')  
 max\_retries = af.data\_config['max\_retries']  
 print('Session initialized')  
  
 # Calls a function that crawls to login the website:  
 if af.data\_config['toggel\_for\_login\_crawler']:  
 call\_function\_with\_retry(crawl\_login, max\_retries)  
 else:  
 pass  
  
 # Call to function that gets all articles from the specified https  
 if af.data\_config['toggel\_for\_juornal\_crawler']:  
 call\_function\_with\_retry(crawl\_journal, max\_retries)  
 else:  
 pass  
  
 # Create a full list of total links from relevant issues and years on the journal  
 rev\_articles\_links\_names\_list = af.os.listdir(af.data\_config['articels\_list\_by\_year\_folder\_name'])  
 rev\_articles\_links\_paths\_list = [af.os.path.join(af.data\_config['articels\_list\_by\_year\_folder\_name'], article\_name)  
 for article\_name in rev\_articles\_links\_names\_list]  
 full\_links\_articles\_list = full\_articles\_links\_list\_creator(rev\_articles\_links\_paths\_list)  
  
 # Call to function that gets all articles from the specified https  
 # For each article - we save the full article data dict to a list  
 if af.data\_config['toggel\_for\_extract\_article\_info']:  
 call\_function\_with\_retry(extract\_article\_info, max\_retries, full\_links\_articles\_list)  
 else:  
 pass  
  
 # Get all saved data from folder and extract one output file.  
 rev\_articles\_data\_names\_list = af.os.listdir(af.data\_config['articels\_data\_by\_year\_folder\_name'])  
 rev\_articles\_data\_paths\_list = [  
 af.os.path.join(af.data\_config['articels\_data\_by\_year\_folder\_name'], article\_data\_name) for article\_data\_name in  
 rev\_articles\_data\_names\_list]  
 af.build\_full\_data\_df(rev\_articles\_data\_paths\_list)  
  
 af.driver.quit() # Close the browser when done  
 print("Driver closed")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

**ActivationFunctions.py File:**

import pandas as pd  
from Main import \*  
import pandas as pd  
from selenium import webdriver  
from selenium.webdriver.chrome.service import Service  
from tqdm import tqdm  
import yaml  
import os  
import pickle  
import urllib.request  
import re  
from PIL import Image  
import keyboard  
from selenium.webdriver.common.keys import Keys  
from selenium.webdriver.support.ui import WebDriverWait  
from selenium.webdriver.support import expected\_conditions as EC  
import numpy as np  
import time  
from genderize import Genderize  
from datetime import datetime  
from selenium.webdriver.common.by import By  
global data\_config  
  
# General Core functions  
  
# Function that calls relevant functions, retries with sleep for max retries  
def call\_function\_with\_retry(function, max\_retries, article\_path=False):  
 retries = 0  
 while retries < max\_retries:  
 try:  
 driver.window\_handles  
 except:  
 if retries > 0:  
 driver.close()  
 driver.quit()  
 time.sleep(0.5)  
 call\_driver()  
 if len(driver.window\_handles) != 0:  
 driver.switch\_to.window(driver.window\_handles[0])  
 if article\_path == False:  
 res = function()  
 else:  
 res = function(article\_path)  
  
 if res:  
 break  
 else:  
 print(f"Retrying with function: {function}...")  
 retries += 1  
  
# Initialize the Selenium Chrome driver  
def call\_driver():  
 global driver  
 global wait  
 global genderize  
 chrome\_service = Service(af.data\_config['chrome\_service\_path'])  
 driver = webdriver.Chrome(service=chrome\_service)  
 driver.maximize\_window()  
 wait = WebDriverWait(driver, 45)  
 genderize = Genderize()  
  
# Press a keyboard key to prevent screen lock  
def press\_key():  
 keyboard.press('up')  
 time.sleep(0.5)  
 keyboard.release('up')  
 time.sleep(5)  
  
# Check if a folder exists, and create if not  
def folder\_exists(folder\_name):  
 if os.path.isdir(folder\_name) == False:  
 os.mkdir(folder\_name)  
 print(f'Created a directory: {folder\_name}')  
 else:  
 print(f'Directory {folder\_name} already exists')  
 pass  
  
# Load config file  
def yaml\_loader(path):  
 with open(path, "r") as yaml\_file:  
 data = yaml.safe\_load(yaml\_file)  
 yaml\_file.close()  
 return data  
  
# Dump data into a YAML file  
def yaml\_dumper(path, data):  
 with open(path, "w") as yaml\_file:  
 yaml.dump(data, yaml\_file)  
 yaml\_file.close()  
  
# Load data from a pickled file  
def pickle\_loader(path):  
 with open(path, 'rb') as file:  
 loaded\_object = pickle.load(file)  
 file.close()  
 return loaded\_object  
  
# Dump data into a pickled file  
def pickel\_dumper(path, data):  
 with open(path, 'wb') as file:  
 pickle.dump(data, file)  
 file.close()  
  
# Find element by method and path, then click  
def find\_element\_click(method, path):  
 element = wait.until(EC.presence\_of\_element\_located((method, path)))  
 element.click()  
  
# Find element by XPath and clear its content  
def find\_element\_xpath\_clear(path):  
 element = wait.until(EC.presence\_of\_element\_located((By.XPATH, path)))  
 element.clear()  
 return element  
  
def full\_articles\_links\_list\_creator(rev\_articles\_links\_paths\_list):  
 *"""  
 Create a list of full article links from paths to individual article lists.  
  
 Args:  
 rev\_articles\_links\_paths\_list (list): List of paths to article lists.  
  
 Returns:  
 list: List of unique article links.  
 """* full\_articels\_links\_list = []  
 for articel\_list\_path in rev\_articles\_links\_paths\_list:  
 articel\_data\_list = yaml\_loader(articel\_list\_path)  
 for article\_link in articel\_data\_list:  
 if article\_link not in full\_articels\_links\_list:  
 full\_articels\_links\_list.append(article\_link)  
 else:  
 continue  
  
 return full\_articels\_links\_list  
  
  
def full\_names\_checker(rev\_authors\_list):  
 *"""  
 Check if the first and last author names in the list are different.  
  
 Args:  
 rev\_authors\_list (list): List of author elements.  
  
 Returns:  
 bool: True if first and last names are different, False otherwise.  
 """* result = True  
 first\_full\_name = rev\_authors\_list[0].text.split('\n')[0]  
 last\_full\_name = rev\_authors\_list[-1].text.split('\n')[0]  
 if first\_full\_name == last\_full\_name:  
 result = False  
  
 return result  
  
  
def text\_cleaner(text):  
 *"""  
 Clean the input text by removing specified characters and extra spaces.  
  
 Args:  
 text (str): Input text to be cleaned.  
  
 Returns:  
 str: Cleaned text.  
 """* chars\_to\_remove = ['$', '^', '\*', '/', '@', '#', '\_', '\\', '{', '}', '"']  
 for char in chars\_to\_remove:  
 text = text.replace(char, '')  
 sentence = text.replace('\n', ' ')  
 sentence = re.sub(r'‘|’', "'", sentence)  
 sentence = re.sub("'", "", sentence)  
 sentence = re.sub(r'\s+', ' ', sentence)  
 return sentence  
  
  
def articels\_list\_checker(year):  
 *"""  
 Check if an articles list file exists for the given year.  
  
 Args:  
 year (str): Year to check for.  
  
 Returns:  
 bool: True if an articles list file exists for the given year, False otherwise.  
 """* year\_text = year.text  
 all\_files = os.listdir(data\_config['articels\_list\_by\_year\_folder\_name'])  
 prefix = "articles\_list\_up\_to\_"  
 matching\_files = [file for file in all\_files if file.startswith(prefix)]  
 year\_file\_exist = False  
 for file\_year\_name in matching\_files:  
 if year\_text in file\_year\_name:  
 year\_file\_exist = True  
 return year\_file\_exist  
  
 return year\_file\_exist  
  
  
def jpg\_writer(image\_url, output\_path):  
 *"""  
 Write an image from a URL to a specified output path, resizing it to a standard size.  
  
 Args:  
 image\_url (str): URL of the image to download.  
 output\_path (str): Path to save the downloaded and resized image.  
 """* target\_width = 800  
 target\_height = 600  
 urllib.request.urlretrieve(image\_url, output\_path)  
 image = Image.open(output\_path)  
 if image.mode != 'RGB':  
 image = image.convert('RGB')  
 resized\_image = image.resize((target\_width, target\_height))  
 resized\_image.save(output\_path, 'JPEG')  
 print('Image saved successfully.')  
  
# Semi-Core functions  
def fill\_full\_article\_data\_dict(input\_df, general\_dict, authors\_dict, figures\_dict, tables\_dict):  
 *"""  
 Create a dictionary containing all the information for a full article record.  
  
 Args:  
 input\_df (DataFrame): DataFrame containing article data.  
 general\_dict (dict): Dictionary containing general article information.  
 authors\_dict (dict): Dictionary containing author information.  
 figures\_dict (dict): Dictionary containing figure information.  
 tables\_dict (dict): Dictionary containing table information.  
  
 Returns:  
 dict: Dictionary containing complete article data.  
 """* full\_data\_article\_dict = dict.fromkeys(input\_df.columns)  
 for i in range(1, 11):  
 full\_data\_article\_dict['Figure ' + str(i) + ' Link'] = figures\_dict['figures\_data']['figure\_' + str(i)]['figure\_link']  
 full\_data\_article\_dict['Figure ' + str(i) + ' caption'] = figures\_dict['figures\_data']['figure\_' + str(i)]['figure\_text']  
 full\_data\_article\_dict['Table ' + str(i) + ' caption'] = tables\_dict['tables\_data']['table' + str(i)]['table\_text']  
  
 full\_data\_article\_dict['Affiliation of the first author'] = authors\_dict['first\_author']['affiliation']  
 full\_data\_article\_dict['Affiliation of the last author'] = authors\_dict['last\_author']['affiliation']  
 full\_data\_article\_dict['First author gender probability'] = authors\_dict['first\_author']['gender\_probability']  
 full\_data\_article\_dict['Gender of the first author'] = authors\_dict['first\_author']['gender']  
 full\_data\_article\_dict['Gender of the last author'] = authors\_dict['last\_author']['gender']  
 full\_data\_article\_dict['Last author gender probability'] = authors\_dict['last\_author']['gender\_probability']  
 full\_data\_article\_dict['Name of the first author'] = authors\_dict['first\_author']['name']  
 full\_data\_article\_dict['Name of the last author'] = authors\_dict['last\_author']['name']  
 full\_data\_article\_dict['Number of Figures'] = figures\_dict['num\_figuers']  
 full\_data\_article\_dict['Number of Tables'] = tables\_dict['num\_tables']  
 full\_data\_article\_dict['Number of authors'] = authors\_dict['num\_authors']  
 full\_data\_article\_dict['Paper title'] = general\_dict['article\_name']  
 full\_data\_article\_dict['Publication Date'] = general\_dict['publication\_date']  
 full\_data\_article\_dict['paper DOI'] = general\_dict['doi']  
  
 return full\_data\_article\_dict  
  
  
def get\_gender(name):  
 *"""  
 Get gender information for a given name.  
  
 Args:  
 name (str): Name to determine gender for.  
  
 Returns:  
 str, float: Gender and probability of the determined gender.  
 """* names\_list = name.split(' ')  
 rev\_api\_names\_list = []  
 rev\_dict\_names\_list = []  
 if 'names\_dict.pickel' in os.listdir(data\_config['names\_dict\_path'].split('\\')[0]):  
 names\_dict = pickle\_loader(data\_config['names\_dict\_path'])  
 else:  
 pickel\_dumper(data\_config['names\_dict\_path'], dict())  
 names\_dict = pickle\_loader(data\_config['names\_dict\_path'])  
 res\_dict = dict.fromkeys(['gender', 'probability', 'count'])  
  
 for name in names\_list:  
 if name in names\_dict.keys():  
 rev\_dict\_names\_list.append(name)  
 else:  
 rev\_api\_names\_list.append(name)  
 if len(rev\_api\_names\_list) != 0:  
 try:  
 gender\_predictions\_list = genderize.get(rev\_api\_names\_list)  
 except:  
 print('Problem with API access. Will try again in 24 Hours.')  
 time.sleep(88200)  
 print('Back after 24 hours.')  
 press\_key()  
 gender\_predictions\_list = genderize.get(rev\_api\_names\_list)  
 else:  
 gender\_predictions\_list = []  
  
 for name\_key in rev\_dict\_names\_list:  
 if res\_dict['probability'] == 0 or res\_dict['probability'] is None:  
 res\_dict['gender'] = names\_dict[name\_key]['gender']  
 res\_dict['probability'] = names\_dict[name\_key]['probability']  
 res\_dict['count'] = names\_dict[name\_key]['count']  
 elif names\_dict[name\_key]['probability'] > res\_dict['probability'] and names\_dict[name\_key]['count'] > res\_dict['count']:  
 res\_dict['gender'] = names\_dict[name\_key]['gender']  
 res\_dict['probability'] = names\_dict[name\_key]['probability']  
 res\_dict['count'] = names\_dict[name\_key]['count']  
  
 for prediction in gender\_predictions\_list:  
 names\_dict[prediction['name']] = dict.fromkeys(['gender', 'probability', 'count'])  
 names\_dict[prediction['name']]['gender'] = prediction['gender']  
 names\_dict[prediction['name']]['probability'] = prediction['probability']  
 names\_dict[prediction['name']]['count'] = prediction['count']  
 pickel\_dumper(data\_config['names\_dict\_path'], names\_dict)  
  
 if res\_dict['probability'] == 0 or res\_dict['probability'] is None:  
 res\_dict['gender'] = prediction['gender']  
 res\_dict['probability'] = prediction['probability']  
 res\_dict['count'] = prediction['count']  
 elif prediction['probability'] >= res\_dict['probability'] and prediction['count'] > res\_dict['count']:  
 res\_dict['gender'] = prediction['gender']  
 res\_dict['probability'] = prediction['probability']  
 res\_dict['count'] = prediction['count']  
  
 if res\_dict['gender'] is None:  
 res\_dict['gender'] = 'Not Available'  
 res\_dict['probability'] = 0  
 gender = res\_dict['gender']  
 probability = res\_dict['probability']  
 return gender, probability  
  
  
  
def none\_figuers\_dict\_extractor():  
 *"""  
 Creates a dictionary with placeholder information for figures.  
  
 Returns:  
 dict: A dictionary containing placeholder information for figures.  
 """* figuers\_dict = dict.fromkeys(['num\_figuers', 'figures\_data'])  
 figuers\_dict['num\_figuers'] = 0  
 figuers\_dict['figures\_data'] = dict()  
 for i in range(1, 11):  
 figure\_key = 'figure\_' + str(i)  
 figuers\_dict['figures\_data'].update(  
 {figure\_key: {'figure\_text': 'Not Available', 'figure\_link': 'Not Available'}})  
 return figuers\_dict  
  
  
def figuers\_data\_extractor(figures\_bar\_div, data\_year, paper\_name):  
 *"""  
 Extract figure data from the figures bar div.  
  
 Args:  
 figures\_bar\_div (WebElement): WebElement containing the figures bar div.  
 data\_year (str): Year of the article data.  
 paper\_name (str): Name of the article.  
  
 Returns:  
 dict: Dictionary containing figure data.  
 """* if figures\_bar\_div is None:  
 return none\_figuers\_dict\_extractor()  
  
 folder\_exists(data\_config['article\_figuers\_main\_folder\_name'])  
 folder\_exists(os.path.join(data\_config['article\_figuers\_main\_folder\_name'], data\_year))  
 rev\_paper\_name\_list = [char for char in paper\_name if char.isalpha() or char == ' ']  
 rev\_paper\_name = ''.join(rev\_paper\_name\_list).replace(' ', '\_')  
 if len(rev\_paper\_name) > 80:  
 rev\_paper\_name = rev\_paper\_name[:80]  
 folder\_exists(os.path.join(data\_config['article\_figuers\_main\_folder\_name'], data\_year, rev\_paper\_name))  
 figuers\_dict = dict.fromkeys(['num\_figuers', 'figures\_data'])  
 figuers\_dict['figures\_data'] = dict()  
 for i in range(1, 11):  
 figure\_key = 'figure\_' + str(i)  
 figuers\_dict['figures\_data'].update(  
 {figure\_key: {'figure\_text': 'Not Available', 'figure\_link': 'Not Available'}})  
  
 wait.until(EC.presence\_of\_element\_located((By.TAG\_NAME, 'i')))  
 figures\_click\_sign = figures\_bar\_div.find\_element(By.TAG\_NAME, 'i')  
 figures\_click\_sign.click()  
 time.sleep(2)  
  
 figuers\_data\_div\_data\_result = False  
 while not figuers\_data\_div\_data\_result:  
 figuers\_data\_div = figures\_bar\_div.find\_elements(By.CLASS\_NAME, 'stats-figures-carousel-container')  
 if len(figuers\_data\_div) == 0:  
 figuers\_data\_div\_data\_result = False  
 sign\_in\_problem\_div\_list = figures\_bar\_div.find\_elements(By.CLASS\_NAME, 'stats-figures-signInToView')  
 if data\_config['figuers\_problem\_check\_string'] in figures\_bar\_div.text:  
 print('no figuers')  
 os.remove(os.path.join(data\_config['article\_figuers\_main\_folder\_name'], data\_year, rev\_paper\_name))  
 return none\_figuers\_dict\_extractor()  
 if len(sign\_in\_problem\_div\_list) == 1:  
 print('Sign in div continue')  
 sign\_in\_problem\_div\_list[0].click()  
 find\_element\_click(By.CLASS\_NAME,  
 'stats-Doc\_Details\_sign\_in\_seamlessaccess\_access\_through\_institution\_name\_btn')  
 print('Break on sign in problem')  
 elif len(figuers\_data\_div) != 1:  
 print('Figures problem - length of full figures div')  
 else:  
 figuers\_data\_div\_data\_result = True  
  
 figuers\_full\_data\_div\_list = figuers\_data\_div[0].find\_elements(By.CLASS\_NAME, 'hide-mobile')  
 rev\_figuers\_data\_div\_list = [element for element in figuers\_full\_data\_div\_list if  
 element.find\_element(By.CLASS\_NAME, 'figure-name').text != '']  
 figuers\_dict['num\_figuers'] = len(rev\_figuers\_data\_div\_list)  
 figuers\_data\_div\_list = rev\_figuers\_data\_div\_list[:10]  
  
 for ind, figure\_data\_div in enumerate(figuers\_data\_div\_list):  
 key\_num = str(ind + 1)  
 key = 'figure\_' + key\_num  
 figuers\_dict['figures\_data'][key]['figure\_link'] = os.path.join(data\_config['article\_figuers\_main\_folder\_name'],  
 data\_year, rev\_paper\_name, key + '.jpg')  
 figuers\_elements\_data\_load = False  
 while not figuers\_elements\_data\_load:  
 try:  
 figuers\_dict['figures\_data'][key]['figure\_text'] = text\_cleaner(  
 figure\_data\_div.find\_element(By.TAG\_NAME, 'p').text)  
 figure\_web\_link = figure\_data\_div.find\_element(By.TAG\_NAME, 'img').get\_attribute('src')  
 figuers\_elements\_data\_load = True  
 except:  
 figuers\_elements\_data\_load = False  
  
 jpg\_writer(figure\_web\_link, figuers\_dict['figures\_data'][key]['figure\_link'])  
  
 time.sleep(0.5)  
 figures\_click\_sign.click()  
 return figuers\_dict  
  
  
def tabels\_data\_extractor(full\_text\_section\_div):  
 *"""  
 Extract table data from the full text section div.  
  
 Args:  
 full\_text\_section\_div (WebElement): WebElement containing the full text section div.  
  
 Returns:  
 dict: Dictionary containing table data.  
 """* tables\_data\_dict = dict.fromkeys(['num\_tables', 'tables\_data'])  
 tables\_data\_dict['tables\_data'] = dict()  
  
 for i in range(1, 1000000):  
 table\_id\_name = 'table' + str(i)  
 try:  
 tables\_data\_dict['tables\_data'].update({table\_id\_name: {'table\_text': None}})  
 tables\_data\_dict['tables\_data'][table\_id\_name]['table\_text'] = text\_cleaner(full\_text\_section\_div.find\_element(By.ID, table\_id\_name).text)  
 except:  
 del tables\_data\_dict['tables\_data'][table\_id\_name]  
 tables\_data\_dict['num\_tables'] = i - 1  
 for j in range(i, 11):  
 new\_table\_id\_name = 'table' + str(j)  
 tables\_data\_dict['tables\_data'].update({new\_table\_id\_name: {'table\_text': 'Not Available'}})  
 break  
 time.sleep(1)  
 return tables\_data\_dict  
  
  
def authors\_data\_extractor(authors\_bar\_div):  
 *"""  
 Extract author data from the authors bar div.  
  
 Args:  
 authors\_bar\_div (WebElement): WebElement containing the authors bar div.  
  
 Returns:  
 dict: Dictionary containing author data.  
 """* if authors\_bar\_div is None:  
 authors\_dict = dict.fromkeys(['first\_author', 'last\_author', 'num\_authors'])  
 authors\_dict['num\_authors'] = 'Not Available'  
 for author\_key in ['first\_author', 'last\_author']:  
 authors\_dict[author\_key] = {  
 "name": 'Not Available',  
 "gender": 'Not Available',  
 "gender\_probability": 'Not Available',  
 "affiliation": 'Not Available'  
 }  
 return authors\_dict  
  
 authors\_dict = dict.fromkeys(['first\_author', 'last\_author', 'num\_authors'])  
 wait.until(EC.presence\_of\_element\_located((By.TAG\_NAME, 'i')))  
 authors\_click\_sign = authors\_bar\_div.find\_element(By.TAG\_NAME, 'i')  
 authors\_click\_sign.click()  
 wait.until(EC.presence\_of\_element\_located((By.ID, 'authors')))  
  
 authors\_div\_list = authors\_bar\_div.find\_elements(By.ID, 'authors')  
 authors\_div = authors\_div\_list[1]  
 author\_data\_list = authors\_div.find\_elements(By.CLASS\_NAME, 'col-14-24')  
 if len(author\_data\_list) == 0:  
 author\_data\_list = authors\_div.find\_elements(By.CLASS\_NAME, 'col-24-24')  
 authors\_dict['num\_authors'] = len(author\_data\_list)  
 rev\_authors\_list = [author\_data\_list[0], author\_data\_list[-1]]  
 check\_name = full\_names\_checker(rev\_authors\_list)  
  
 for ind, author in enumerate(rev\_authors\_list):  
 author\_data\_list = author.text.split('\n')  
 if len(author\_data\_list) != 1:  
 author\_name = author\_data\_list[0]  
 author\_affi = author\_data\_list[1].lower()  
 else:  
 author\_name = author\_data\_list[0]  
 author\_affi = 'Not Available'  
  
 if ind != 0:  
 if check\_name:  
 author\_gender, author\_probability = get\_gender(author\_name)  
 else:  
 pass  
 else:  
 author\_gender, author\_probability = get\_gender(author\_name)  
  
 if ind == 0:  
 author\_key = 'first\_author'  
 else:  
 author\_key = 'last\_author'  
  
 authors\_dict[author\_key] = {  
 "name": author\_name,  
 "gender": author\_gender,  
 "gender\_probability": author\_probability,  
 "affiliation": text\_cleaner(author\_affi)  
 }  
 time.sleep(0.5)  
 authors\_click\_sign.click()  
 return authors\_dict  
  
  
def publication\_doi\_div\_exctractor():  
 *"""  
 Extract DOI and publication date data from the document page.  
  
 Returns:  
 tuple: Tuple containing the DOI data div and publication data div.  
 """* data\_found = False  
 while not data\_found:  
 try:  
 all\_data\_divs = driver.find\_elements(By.TAG\_NAME, 'div')  
 doi\_data\_div = [element for element in all\_data\_divs if  
 element.get\_attribute('class') == 'u-pb-1 stats-document-abstract-doi']  
 publication\_data\_div = [element for element in all\_data\_divs if  
 element.get\_attribute('class') == 'u-pb-1 doc-abstract-pubdate']  
 data\_found = True  
 except:  
 data\_found = False  
 return doi\_data\_div, publication\_data\_div  
  
  
def cover\_articles\_deletor(article\_link, full\_articles\_links\_list, general\_article\_data\_dict):  
 *"""  
 Delete cover articles from the list of full articles links.  
  
 Args:  
 article\_link (str): Link of the article to be deleted.  
 full\_articles\_links\_list (list): List of full articles links.  
 general\_article\_data\_dict (dict): Dictionary containing general article data.  
  
 Returns:  
 None  
 """* print('exception\_cover')  
 if 'cover\_list.yml' in os.listdir(data\_config['cover\_list\_path'].split('/')[0]):  
 is\_cover\_problem\_list = yaml\_loader(data\_config['cover\_list\_path'])  
 else:  
 yaml\_dumper(data\_config['cover\_list\_path'], [])  
 is\_cover\_problem\_list = yaml\_loader(data\_config['cover\_list\_path'])  
  
 if article\_link in is\_cover\_problem\_list:  
 full\_articles\_links\_list = full\_articles\_links\_list.tolist()  
 full\_articles\_links\_list.remove(article\_link)  
 yaml\_dumper(data\_config['full\_articels\_links\_list\_no\_cover\_path'], full\_articles\_links\_list)  
 else:  
 is\_cover\_problem\_list.append(article\_link)  
 yaml\_dumper(data\_config['cover\_list\_path'], np.array(is\_cover\_problem\_list).tolist())  
 print(f"deleted {general\_article\_data\_dict['article\_name'].lower()} on link : {article\_link}")  
 full\_articles\_links\_list = full\_articles\_links\_list.tolist()  
 full\_articles\_links\_list.remove(article\_link)  
 yaml\_dumper(data\_config['full\_articels\_links\_list\_no\_cover\_path'], full\_articles\_links\_list)  
  
  
# Full-Core functions:  
# Function to concat\_all\_yaml\_lists\_to\_One\_df  
def build\_full\_data\_df(rev\_articles\_data\_paths\_list):  
 *"""  
 Build a full data DataFrame from the list of article data paths.  
  
 Args:  
 rev\_articles\_data\_paths\_list (list): List of article data paths.  
  
 Returns:  
 None  
 """* full\_df = pd.DataFrame()  
 bad\_paths = []  
 for data\_path in tqdm(rev\_articles\_data\_paths\_list):  
 data\_list = yaml\_loader(data\_path)  
 for data\_dict in data\_list:  
 for key, value in data\_dict.items():  
 if type(value) == str:  
 if 'link' in key.lower():  
 if os.path.exists(value):  
 continue  
 else:  
 if value != 'Not Available':  
 bad\_paths.append(value)  
 continue  
 else:  
 continue  
 df = pd.DataFrame(data\_list)  
 full\_df = pd.concat([full\_df, df])  
  
 full\_df\_name = 'output.xlsx'  
 full\_df.to\_excel(os.path.join(data\_config['full\_output\_folder'], full\_df\_name), index=False)  
 print(bad\_paths)  
  
  
def extract\_article\_info(article\_path):  
 *"""  
 Extracts information from an article and saves it to the appropriate data structures.  
  
 Args:  
 article\_path (str): Path to the article.  
  
 Returns:  
 list: A list containing success status and action (if needed).  
 """* try:  
 cover\_suspected = False  
 general\_article\_data\_dict = dict.fromkeys(['article\_name', 'doi', 'publication\_date'])  
 folder\_exists(data\_config['articels\_data\_by\_year\_folder\_name'])  
 dict\_list = os.listdir(data\_config['articels\_data\_by\_year\_folder\_name'])  
 articels\_start\_index = len(dict\_list) \* 10 + 1  
 if 'full\_articels\_links\_list\_no\_cover.yml' in os.listdir(  
 data\_config['full\_articels\_links\_list\_no\_cover\_path'].split('/')[0]):  
 full\_articles\_links\_list = np.sort(yaml\_loader(data\_config['full\_articels\_links\_list\_no\_cover\_path']))  
 articles\_links\_list = full\_articles\_links\_list[articels\_start\_index:]  
 else:  
 if articels\_start\_index == 1:  
 articles\_links\_list = np.sort(article\_path)[articels\_start\_index - 1:]  
 else:  
 articles\_links\_list = np.sort(article\_path)[articels\_start\_index:]  
 press\_key()  
 full\_article\_data\_dict\_list = []  
 input\_df = pd.read\_excel(data\_config['input\_df\_path'])  
 for ind, article\_link in enumerate(articles\_links\_list):  
 ind += articels\_start\_index  
 driver.get(article\_link)  
 try:  
 find\_element\_click(By.XPATH, data\_config['url\_xpath\_dict']['cookies'])  
 except:  
 pass  
 driver.execute\_script("window.scrollTo(0, document.body.scrollHeight);")  
 # time.sleep(0.5)  
 driver.execute\_script("window.scrollTo(0, 0);")  
 general\_article\_data\_dict['article\_name'] = text\_cleaner(wait.until(  
 EC.presence\_of\_element\_located((By.XPATH, data\_config['url\_xpath\_dict']['article\_title']))).text)  
 time.sleep(0.5)  
 doi\_data\_div, publication\_data\_div = publication\_doi\_div\_exctractor()  
 general\_article\_data\_dict['doi'] = doi\_data\_div[0].find\_element(By.TAG\_NAME, 'a').get\_attribute('href')  
  
 publication\_date\_str = publication\_data\_div[0].text.split(":")[1].lstrip()  
 general\_article\_data\_dict['publication\_date'] = datetime.strptime(publication\_date\_str, "%d %B %Y")  
 data\_year = general\_article\_data\_dict['publication\_date'].strftime("%Y")  
  
 driver.execute\_script("window.scrollTo(0, document.body.scrollHeight\*0.975);")  
 article\_data\_div = wait.until(  
 EC.presence\_of\_element\_located((By.XPATH, data\_config['url\_xpath\_dict']['article\_data\_div'])))  
 article\_data\_div\_list = article\_data\_div.find\_elements(By.TAG\_NAME, 'div')  
 full\_relevent\_article\_data\_divs = [article\_data\_innerdiv for article\_data\_innerdiv in article\_data\_div\_list  
 if article\_data\_innerdiv.get\_attribute('class') == "accordion-item"]  
 relevent\_article\_data\_divs = [element\_div for element\_div in full\_relevent\_article\_data\_divs if  
 element\_div.text.lower() == 'authors' or element\_div.text.lower() == 'figures']  
 if len(relevent\_article\_data\_divs) == 1:  
 data\_div\_name = relevent\_article\_data\_divs[0].text.lower()  
 if data\_div\_name == 'figures':  
 figures\_bar\_div = relevent\_article\_data\_divs[0]  
 authors\_bar\_div = None  
 elif data\_div\_name == 'authors':  
 authors\_bar\_div = relevent\_article\_data\_divs[0]  
 figures\_bar\_div = None  
 elif len(relevent\_article\_data\_divs) == 0:  
 cover\_suspected = True  
 wait.until(EC.presence\_of\_element\_located((By.ID, 'article')))  
 wait.until(EC.presence\_of\_element\_located((By.ID, 'full-text-section')))  
 cover\_suspected = False  
 else:  
 if relevent\_article\_data\_divs[0].text.lower() == 'authors':  
 authors\_bar\_div = relevent\_article\_data\_divs[0]  
 figures\_bar\_div = relevent\_article\_data\_divs[1]  
 else:  
 authors\_bar\_div = relevent\_article\_data\_divs[1]  
 figures\_bar\_div = relevent\_article\_data\_divs[0]  
  
 authors\_data\_dict = authors\_data\_extractor(authors\_bar\_div)  
 figures\_data\_dict = figuers\_data\_extractor(figures\_bar\_div, data\_year,  
 general\_article\_data\_dict['article\_name'])  
 full\_text\_section\_div = wait.until(EC.presence\_of\_element\_located((By.ID, 'full-text-section')))  
 tables\_data\_dict = tabels\_data\_extractor(full\_text\_section\_div)  
 full\_article\_data\_dict = fill\_full\_article\_data\_dict(input\_df, general\_article\_data\_dict, authors\_data\_dict,  
 figures\_data\_dict, tables\_data\_dict)  
 full\_article\_data\_dict\_list.append(full\_article\_data\_dict)  
  
 if ind % 10 == 0 and ind != 0:  
 file\_name = 'output' + '\_' + str(ind + 1) + '.yml'  
 yaml\_dumper(os.path.join(data\_config['articels\_data\_by\_year\_folder\_name'], file\_name),  
 full\_article\_data\_dict\_list)  
 full\_article\_data\_dict\_list = []  
  
 print("extract\_article\_info Function executed successfully!")  
 return True  
  
 except Exception as e:  
 if cover\_suspected:  
 cover\_articles\_deletor(article\_link, full\_articles\_links\_list, general\_article\_data\_dict)  
 elif not cover\_suspected:  
 print('exception on not suspected cover papers')  
 print(f"extract\_article\_info Function failed: {e}")  
 return False  
  
  
def crawl\_login():  
 *"""  
 Performs the login process to the journal website using institutional credentials.  
  
 Returns:  
 bool: True if login is successful, False otherwise.  
 """* try:  
 activation\_stay\_key = data\_config['activation\_stay\_key']  
 press\_key()  
 driver.get(data\_config['journal\_url'])  
 # click on browser  
 find\_element\_click(By.XPATH, data\_config['url\_xpath\_dict']['sign\_in\_button'])  
 find\_element\_click(By.XPATH, data\_config['url\_xpath\_dict']['institution\_button'])  
 # calls function that clear the input box element inside the browser  
 input\_element = find\_element\_xpath\_clear(data\_config['url\_xpath\_dict']['institution\_input\_element'])  
 # Type a string into the input element --> same institution for everyone  
 input\_element.send\_keys(data\_config['institution\_name'])  
 press\_key()  
 # sign in via institution  
 find\_element\_click(By.XPATH, data\_config['url\_xpath\_dict']['institution\_selection\_first\_item'])  
 find\_element\_click(By.XPATH, data\_config['url\_xpath\_dict']['students\_button'])  
 # puts email add and passwd info  
 email\_input = find\_element\_xpath\_clear(data\_config['url\_xpath\_dict']['institution\_email\_input'])  
 email\_input.send\_keys(data\_config['login\_email\_add'])  
 email\_input.send\_keys(Keys.ENTER)  
 press\_key()  
 email\_passwd\_input = find\_element\_xpath\_clear(data\_config['url\_xpath\_dict']['institution\_email\_passwd\_input'])  
 email\_passwd\_input.send\_keys(data\_config['login\_passwd'])  
 email\_passwd\_input.send\_keys(Keys.ENTER)  
  
 # waits to identification from microsoft authenticator  
 if activation\_stay\_key == False:  
 find\_element\_click(By.XPATH, data\_config['url\_xpath\_dict']['sign\_in\_activation'])  
 else:  
 find\_element\_click(By.XPATH, data\_config['url\_xpath\_dict']['sign\_in\_activation\_1'])  
  
 print("crawl\_login Function executed successfully!")  
 return True  
 except Exception as e:  
 print(f"crawl\_login Function failed: {e}")  
 return False  
  
  
def crawl\_journal():  
 *"""  
 Crawls through journal pages, extracting article information and saving article links.  
  
 Returns:  
 bool: True if crawling is successful, False otherwise.  
 """* try:  
 articles\_list = []  
 folder\_exists(data\_config['articels\_list\_by\_year\_folder\_name'])  
 #gets to the relevent browser link  
 driver.get(data\_config['journal\_url'])  
 # work by decades  
 find\_element\_click(By.XPATH, data\_config['url\_xpath\_dict']['all\_issues\_botton'])  
 decades\_elements\_div = wait.until(EC.presence\_of\_element\_located((By.XPATH, data\_config['url\_xpath\_dict']['decades\_div'])))  
 decades\_li\_ele = decades\_elements\_div.find\_elements(By.TAG\_NAME, "li")  
 for decedes\_ind, decades\_ele in tqdm(enumerate(decades\_li\_ele[:2])):  
 press\_key()  
 #for a case that there is no cookies pop up  
 try:  
 find\_element\_click(By.XPATH, data\_config['url\_xpath\_dict']['cookies'])  
 except:  
 pass  
 decades\_ele.click()  
 years\_elements\_div = driver.find\_element(By.XPATH, data\_config['url\_xpath\_dict']['years\_div'])  
 years\_li\_ele = years\_elements\_div.find\_elements(By.TAG\_NAME, "li")  
 if decedes\_ind == 1:  
 years\_li\_ele = years\_li\_ele[:5]  
 # work on issues by years  
 for year in years\_li\_ele:  
 press\_key()  
 # for a case of that data already outputted for year  
 if articels\_list\_checker(year):  
 continue  
 # select relevat year  
 year\_string = year.text  
 print(year\_string)  
 year.click()  
 # gets the isuses volume div  
 volume\_elements\_div = wait.until(EC.presence\_of\_element\_located((By.XPATH, data\_config['url\_xpath\_dict']['volume\_div'])))  
 volume\_li\_ele = volume\_elements\_div.find\_elements(By.TAG\_NAME, "div")  
 for issue\_div in volume\_li\_ele:  
 press\_key()  
 # gets element issue from every div  
 try:  
 issue\_ele = issue\_div.find\_element(By.TAG\_NAME, "a")  
 except:  
 continue  
  
 href\_issue\_link = issue\_ele.get\_attribute("href")  
 driver.execute\_script("window.open()")  
 wait.until(EC.presence\_of\_element\_located((By.TAG\_NAME, "body")))  
 driver.switch\_to.window(driver.window\_handles[1])  
 driver.get(href\_issue\_link)  
 # open all papers on same window  
 find\_element\_click(By.XPATH,data\_config['url\_xpath\_dict']['items\_per\_page'])  
 items\_ele = wait.until(  
 EC.presence\_of\_element\_located((By.XPATH, data\_config['url\_xpath\_dict']['items\_per\_page\_list'])))  
 relevent\_number\_items = items\_ele.find\_elements(By.TAG\_NAME, "button")  
 #clicking on the last element for open all issue papers on the same page  
 relevent\_number\_items[-1].click()  
 # opens papers div on window  
 papers\_div = wait.until(EC.presence\_of\_element\_located((By.XPATH, data\_config['url\_xpath\_dict']['papers\_div'])))  
 papers\_divs\_list = papers\_div.find\_elements(By.TAG\_NAME, "div")  
 relevent\_papers\_divs = [paperdiv for paperdiv in papers\_divs\_list if paperdiv.get\_attribute("class")=="List-results-items"]  
 press\_key()  
 for paper\_div in tqdm(relevent\_papers\_divs):  
 # by searching with a tag we are clear from Cover papers which are located in  
 # span tags on List-results-items class divs on the results page  
 # check for cover papers -- only a tags  
 paper\_title = paper\_div.find\_element(By.TAG\_NAME, "h2")  
 # for cover articles case  
 try:  
 title\_link = paper\_title.find\_element(By.TAG\_NAME, "a").get\_attribute("href")  
 except:  
 continue  
 articles\_list.append(title\_link)  
  
 #close the session with the new window that opened  
 driver.close()  
 #switch to original window  
 driver.switch\_to.window(driver.window\_handles[0])  
 wait.until(EC.presence\_of\_element\_located((By.TAG\_NAME, "body")))  
  
 articles\_list\_name = 'articles\_list\_up\_to\_'+year\_string+'.yml'  
 yaml\_dumper(os.path.join(data\_config['articels\_list\_by\_year\_folder\_name'], articles\_list\_name), articles\_list)  
  
 print("crawl\_journal Function executed successfully!")  
 return True  
  
 except Exception as e:  
 print(f"crawl\_journal Function failed: {e}")  
 return False