from flask import Flask, request, jsonify

from dateutil.relativedelta import relativedelta

import calendar

from datetime import datetime

import os

import regex as re

import waitress

import PyPDF2

from docx import Document

from docx.opc.constants import RELATIONSHIP\_TYPE as RT

from docx.document import Document as \_Document

from docx.oxml.text.paragraph import CT\_P

from docx.oxml.table import CT\_Tbl

from docx.table import \_Cell, Table

from docx.text.paragraph import Paragraph

from docx2python import docx2python

from docx2python.iterators import iter\_paragraphs

import pdfplumber

import openpyxl as op

import spacy

from find\_job\_titles import Finder

from bs4 import BeautifulSoup as bs

from operator import itemgetter

from flair.data import Sentence

from flair.models import SequenceTagger

from nltk.corpus import wordnet as wn

## Changes done in this version - Sped up version to run in 1-2 seconds on average

application = Flask(\_\_name\_\_)

@application.route('/', methods=['GET'])

def test():

return jsonify({"result": "Hello world"})

glove = spacy.load('en\_core\_web\_lg')

tagger = SequenceTagger.load("flair/ner-english-fast")

finder = Finder()

@application.route('/getParsedData', methods=['POST'])

def getParsedData():

First\_Name=[]

Last\_Name = []

email\_ids = []

phone\_nos = []

summary\_1=[]

education\_1= []

links = []

linkedIn = []

gitHub = []

all\_positions = []

all\_companies = []

all\_dates = []

all\_univ = []

all\_degrees = []

all\_majors = []

file = request.files['file']

file\_name = file.filename.split('.')[0] + datetime.now().strftime("%Y%m%d%H%M%S") + '.' + file.filename.split('.')[-1]

file.save('uploads/' + file\_name)

if file\_name.endswith('.docx'):

links.append(find\_url\_docx('uploads/' + file\_name))

document,name = open\_docx\_file('uploads/' + file\_name,glove)

if file\_name.endswith('.pdf'):

links.append(find\_url\_pdf('uploads/' + file\_name))

document,name = open\_pdf('uploads/' + file\_name,glove)

if file\_name.endswith('.doc'):

document,name = open\_doc\_file('uploads/' + file\_name,glove)

links.append(re.findall('www.+[.]com',' '.join(document)))

if name != [] :

First\_Name.append(" ".join(re.findall("[a-zA-Z]+", name[0])))

if len(name)>1:

Last\_Name.append(" ".join(re.findall("[a-zA-Z]+", name[-1])))

else:

First\_Name.append('')

Last\_Name.append('')

email = get\_email(document)

if len(email) > 0:

email\_ids.append(email[0])

elif len(get\_email(links[-1]))>0:

email\_ids.append(get\_email(links[-1])[0])

else:

email\_ids.append('')

phone\_no = get\_phone\_no(document)

if len(phone\_no) > 0:

phone\_nos.append(phone\_no[0])

else:

phone\_nos.append('')

summ = get\_summary(document)

if len(summ) > 1:

summary\_1.append(' '.join(summ))

else:

summary\_1.append('')

education = get\_education(document)

if len(education) > 1:

education\_1.append(education)

else:

education\_1.append('')

experiences = get\_experience(document)

univ = get\_university(education)

#print ('Education ', get\_education(document))

for school in univ:

all\_univ.append(re.sub(

'(jan(uary)?|feb(ruary)?|mar(ch)?|apr(il)?|may|jun(e)?|jul(y)?|aug(ust)?|sep(tember)?|oct(ober)?|nov(ember)?|dec(ember)?)(\s|\S)?(\d{2,4})(.\*(jan(uary)?|feb(ruary)?|mar(ch)?|apr(il)?|may|jun(e)?|jul(y)?|aug(ust)?|sep(tember)?|oct(ober)?|nov(ember)?|dec(ember)?)(\s|\S)(\d{2,4}))?', '', school).title())

if links != []:

linkedIn.append(getLinkedIn(links))

gitHub.append(getGithub(links))

else:

linkedIn.append('')

gitHub.append('')

if len(experiences)>0 and len(experiences[0])>2:

for experience in experiences:

if 'PROJECT' not in experience[0].upper() or len(experiences)==1:

position,company,date = extractDataJob(experience[1:],finder,tagger)

if 'INTERN' in experience[0].upper() and not any(pos!='' for pos in position):

position = ['Intern']\*len(company)

all\_positions = all\_positions + position

all\_companies = all\_companies + company

all\_dates = all\_dates + date

for ind,person in enumerate(education\_1):

temp,\_,\_,\_,majors = getEduInfo(person)

all\_degrees = temp

all\_majors = majors

extracted\_data = {

'First\_Name': First\_Name,

'Last\_Name': Last\_Name,

'Email': email\_ids,

'LinkedIn URL': linkedIn,

'GitHub URL': gitHub,

'Contact\_Number': phone\_nos,

'Summary': summary\_1,

}

for ind,item in enumerate(all\_univ):

extracted\_data[f"University\_{ind+1}"] = item

for ind,item in enumerate(all\_degrees):

extracted\_data[f"University {ind+1} Degree"] = item

for ind,item in enumerate(all\_majors):

extracted\_data[f"University {ind+1} Major"] = item

for ind,item in enumerate(all\_positions):

extracted\_data[f"Work Experince {ind+1} Position"] = item

for ind,item in enumerate(all\_companies):

extracted\_data[f"Work Experince {ind+1} Company Name"] = item

for ind,item in enumerate(all\_dates):

extracted\_data[f"Work Experince {ind+1} Dates worked"] = item

for file in os.listdir('uploads'):

os.remove('uploads/' + file)

extracted\_data\_json = jsonify(extracted\_data)

return extracted\_data\_json

def remove\_non\_ascii(string):

return string.encode('ascii', errors='ignore').decode()

def open\_docx\_file(file\_name,nlp):

# A function that iterates through each block in a docx, checks whether it's a pargraph or a table

# , extracting the text in the form {header:cell} for tables. The text is then processed and returned.

# The name is then extracted from the text.

labels = ['TECHNOLOGY','DESIGNATION','ROLE','CLIENT','ORGANIZATION','PROJECT','TITLE','CUSTOMER',

'CLIENT DOMAIN','STAKEHOLDERS','ENVIRONMENT','SKILLS','APPLICATIONS','DOMAIN','INDUSTRY','SERVICE',

'DESCRIPTION','RESPONSIBILITES','SIZE','PERIOD','DURATION','TECHNOLOGY','TOOLS','ACHIEVEMENTS',

'PROGRAMMING LANGUAGES','DATABASES','SKILLS','PROTOCOLS','MULESOFT','PROJECTS','TEAM SIZE','FRAMEWORKS',

'LOCATION','ROLE/ACTIVITIES','NAME','ROLES','PROJECT NAME','SCORE','QUALIFICATION','SCHOOL/COLLEGE',

'SCHOOL/COLLEGE YEAR','INSTITUTION','DEGREE /CERTIFICATE','SUBJECT']

def iter\_block\_items(parent):

if isinstance(parent, \_Document):

parent\_elm = parent.element.body

elif isinstance(parent, \_Cell):

parent\_elm = parent.\_tc

for child in parent\_elm.iterchildren():

if isinstance(child, CT\_P):

yield Paragraph(child, parent)

elif isinstance(child, CT\_Tbl):

yield Table(child, parent)

document = Document(file\_name)

text = []

for block in iter\_block\_items(document):

if isinstance(block, Paragraph):

text.append(block.text)

else:

horizontal = False

short = True

table = True

for i, row in enumerate(block.rows[1:]):

if len(row.cells) <= 1:

table = False

break

start\_ind = 0

while start\_ind<len(block.rows):

cleaned = [\*set([cell.text for cell in block.rows[start\_ind].cells if cell.text.strip()!='' and len(cell.text.strip())>2])]

if len(cleaned)>1:

break

else:

start\_ind = start\_ind + 1

for i, row in enumerate(block.rows[start\_ind+1:]):

cleaned = [\*set([cell.text for cell in row.cells if cell.text.strip()!='' and len(cell.text.strip())>2])]

if len(cleaned) > 1 and len(cleaned[0])>0 and any(label == cleaned[0].strip().upper() for label in labels):

horizontal = True

break

elif len(cleaned)>0 and len(re.split('\n|\s',cleaned[0])) >3:

short = False

temp\_text = [cell.text.strip().upper() for cell in block.rows[start\_ind].cells[1:]] if len(block.rows)>start\_ind and len(block.rows[start\_ind].cells)>1 else []

if not horizontal and short and not any(label in temp\_text for label in labels):

horizontal = True

if horizontal and table:

for i, row in enumerate(block.rows):

temp\_text = [cell.text for cell in row.cells]

cleaned\_text = []

[cleaned\_text.append(x) for x in temp\_text if x not in cleaned\_text and x.strip()!='']

if len(cleaned\_text)>1:

row\_data = [str(cleaned\_text[0]) + ':' + str(item) for item in cleaned\_text[1:] if len(item.strip())>2]

text = text + row\_data

elif len(cleaned\_text)>0:

text.append(cleaned\_text[0])

elif table:

keys = None

for i, row in enumerate(block.rows):

temp\_text = [cell.text for cell in row.cells]

cleaned\_text = []

[cleaned\_text.append(x) for x in temp\_text if x not in cleaned\_text and x.strip()!='']

if keys == None and len(cleaned\_text)>1:

keys = list(cleaned\_text)

continue

if len(cleaned\_text)>1:

row\_data = [str(key) + ':' + str(item) for (key,item) in list(zip(keys, list(temp\_text))) if len(item.strip())>2]

text = text + row\_data

elif len(cleaned\_text)>0:

text.append(cleaned\_text[0])

else:

for i, row in enumerate(block.rows):

temp\_text = [cell.text for cell in row.cells]

if len(temp\_text) >0:

text = text + temp\_text[0].split('\n')

text = [remove\_non\_ascii(line.replace('\t', ' ')) for line in text if line != '' and len(remove\_non\_ascii(line).strip())>1]

doc = docx2python(file\_name)

header\_text = [i for i in list(iter\_paragraphs(doc.header)) if i.replace('\t','').strip()!='' and not '.png' in i and not '.jpg' in i and not '.gif' in i and not '.jpeg' in i and not '.Tiff' in i]

if header\_text != []:

text = header\_text + text

if len(text)>0:

doc2 = nlp(text[0].title())

names = [ent.text for ent in doc2.ents if ent.label\_ == 'PERSON' or ent.label\_ == 'GPE']

if names == [] and 'name:' in text[0].lower():

names = [text[0][text[0].find(':')+1:]]

else:

names = []

return (text,names[0].split() if len(names)>0 else [])

def get\_top\_coordinates(page):

#A function to determine where the header of a pdf stop by checking for letters in strips that 2% of the pdf height

start = .05

while start<.3:

top = page.crop((0, start \* page.height,

page.width,(start+.001) \* page.height))

if top.extract\_text().strip() == '' or len(top.extract\_text().strip())<2:

break

start = start+.01

start = round(start,2)

if start == .3:

return 120

else:

return int(start\*page.height)

def open\_pdf(file,nlp):

#A function to scrape the text from a pdf, first checking if a pdf contains tables and if it does, it processes the text using pdfplumber such that

# the format becomes {header:cell}. If it does not contain tables, it just parses the pdf using pdfplumber. The name is then extracted from the text.

tables = False

labels = ['TECHNOLOGY','DESIGNATION','ROLE','CLIENT','ORGANIZATION','PROJECT','TITLE','CUSTOMER','CLIENT DOMAIN','STAKEHOLDERS','ENVIRONMENT','SKILLS','APPLICATIONS','DOMAIN','INDUSTRY','SERVICE','DESCRIPTION','RESPONSIBILITES','SIZE','PERIOD','DURATION','TECHNOLOGY','TOOLS','ACHIEVEMENTS','PROGRAMMING LANGUAGES','DATABASES','SKILLS','PROTOCOLS','MULESOFT','PROJECTS','TEAM SIZE','FRAMEWORKS','LOCATION','ROLE/ACTIVITIES','NAME','ROLES','PROJECT NAME','SCORE','QUALIFICATION','SCHOOL/COLLEGE','SCHOOL/COLLEGE YEAR','INSTITUTION','DEGREE /CERTIFICATE','SUBJECT']

with pdfplumber.open(file) as pdf:

for page in pdf.pages:

tables = page.find\_tables()

if tables != []:

tables = True

break

def check\_bboxes(word, table\_bbox):

"""

Check whether word is inside a table bbox.

"""

l = word['x0'], word['top'], word['x1'], word['bottom']

r = table\_bbox

return l[0] > r[0] and l[1] > r[1] and l[2] < r[2] and l[3] < r[3]

if tables:

temp = []

with pdfplumber.open(file) as pdf:

for page in pdf.pages:

tables = page.find\_tables()

table\_bboxes = [i.bbox for i in tables]

tables = [{'table': i.extract(), 'top': i.bbox[1]} for i in tables]

non\_table\_words = [word for word in page.extract\_words() if not any(

[check\_bboxes(word, table\_bbox) for table\_bbox in table\_bboxes])]

lines = []

for cluster in pdfplumber.utils.cluster\_objects(

non\_table\_words + tables, itemgetter('top'), tolerance=5):

if 'text' in cluster[0]:

temp.append(' '.join([i['text'] for i in cluster if 'text' in i]))

elif 'table' in cluster[0]:

table = True

horizontal = False

for i, row in enumerate(cluster[0]['table']):

if len(row) <= 1:

table = False

break

horizontal = next((any(label == x[0].strip().upper() for label in labels) for x in cluster[0]['table'][1:] if len(x)>0 and x!=None and x[0]!=None and any(label == x[0].strip().upper() for label in labels)), False)

row\_data = [x.strip().upper() for x in cluster[0]['table'][0][1:] if x!=None]

if not horizontal and not any(len(x[0].split()) >2 for x in cluster[0]['table'] if len(x)>0 and x!=None and x[0]!=None) and not any(label in row\_data for label in labels):

horizontal = True

if horizontal and table:

for i in range(0,len(cluster[0]['table'])):

if len(cluster[0]['table'][i])>1:

row\_data = [str(cluster[0]['table'][i][0]) + ':' + str(x) for x in cluster[0]['table'][i][1:] if x != None and len(x.strip())>2]

temp = temp + row\_data

elif table:

keys = cluster[0]['table'][0]

for i in range(1,len(cluster[0]['table'])):

row\_data = [str(key) + ':' + str(item) for (key,item) in list(zip(keys, list(cluster[0]['table'][i]))) if item != None and len(item.strip())>2]

temp = temp + row\_data

else:

for i, row in enumerate(cluster[0]['table']):

if len(row) >0 and row[0].strip()!='':

temp = temp + row[0].split('\n')

result = []

for line in temp:

line = line.strip()

if line != '' and len(remove\_non\_ascii(line).strip())>1:

result.append(remove\_non\_ascii(line))

if len(result)>0:

doc2 = nlp(result[0].title())

names = [ent.text for ent in doc2.ents if ent.label\_ == 'PERSON']

else:

names = []

return (result,names[0].split() if len(result)>0 and len(names)>0 else [])

else:

output\_string = ""

with pdfplumber.open(file) as pdf:

for ind,page in enumerate(pdf.pages):

start = .2

top\_cord = get\_top\_coordinates(page)

top = page.crop((0, 0, page.width,top\_cord))

while start<.5:

left = page.crop((start \* page.width, top\_cord,

(start+.020) \* page.width,page.height))

if left.extract\_text().strip() == '' or len(left.extract\_text().strip())<2:

break

start = start+.01

start = round(start,2)

if start == .5:

start = .8

while start>.5:

right = page.crop(((start-.020) \* page.width,

top\_cord, start \* page.width,page.height))

if right.extract\_text().strip() == '' or len(right.extract\_text().strip())<2:

break

start = start-.01

start = round(start,2)

if start == .5:

output\_string = output\_string + \

page.extract\_text(x\_tolerance=1) + "\n"

if ind == 0:

stripped\_doc = [i for i in page.extract\_text(x\_tolerance=1).split('\n') if i!=None and i.strip()!='' and len(i.strip())>1]

doc2 = nlp(stripped\_doc[0].title()) if len(stripped\_doc)>0 else []

names = [ent.text for ent in doc2.ents if ent.label\_ == 'PERSON' or ent.label\_ == 'GPE'] if doc2!=[] else []

else:

left = page.crop((0, top\_cord,

start \* page.width+.01,page.height))

right = page.crop((start \* page.width-.01,

top\_cord,page.width,page.height))

output\_string = output\_string + \

top.extract\_text(x\_tolerance=1)+ '\n' +\

left.extract\_text(x\_tolerance=1) + '\n' +\

right.extract\_text(x\_tolerance=1)+'\n'

if ind == 0:

for doc in [top.extract\_text(x\_tolerance=1).split('\n'), left.extract\_text(x\_tolerance=1).split('\n'), right.extract\_text(x\_tolerance=1).split('\n')]:

stripped\_doc = [i for i in doc if i!=None and i.strip()!='' and len(i.strip())>1]

doc2 = nlp(stripped\_doc[0].title()) if len(stripped\_doc)>0 else []

names = [ent.text for ent in doc2.ents if ent.label\_ == 'PERSON' or ent.label\_ == 'GPE'] if doc2!=[] else []

if names != []:

break

else:

left = page.crop((0, top\_cord,

start \* page.width+.01,page.height))

right = page.crop((start \* page.width-.01,

top\_cord,page.width,page.height))

output\_string = output\_string +\

top.extract\_text(x\_tolerance=1) + '\n'+\

left.extract\_text(x\_tolerance=1) + '\n' +\

right.extract\_text(x\_tolerance=1)+'\n'

if ind == 0:

for doc in [left.extract\_text(x\_tolerance=1).split('\n'),right.extract\_text(x\_tolerance=1).split('\n')]:

stripped\_doc = [i for i in doc if i!=None and i.strip()!='' and len(i.strip())>1]

doc2 = nlp(stripped\_doc[0].title()) if len(stripped\_doc)>0 else []

names = [ent.text for ent in doc2.ents if ent.label\_ == 'PERSON' or ent.label\_ == 'GPE'] if doc2!=[] else []

if names != []:

break

result = []

for line in output\_string.split('\n'):

line = line.strip()

if line != '' and len(remove\_non\_ascii(line).strip())>1:

result.append(remove\_non\_ascii(line))

return (result,names[0].split() if len(result)>0 and len(names)>0 else [])

def open\_doc\_file(file\_name,nlp):

# A function to open and extract the text and name from a doc file.

soup = bs(open(file\_name, encoding="ISO-8859-1").read())

[s.extract() for s in soup(['style', 'script'])]

tmpText = soup.get\_text()

text = [remove\_non\_ascii(line) for line in tmpText.split('\n') if line.strip() != '' and len(remove\_non\_ascii(line).strip())>1]

if len(text)>0:

doc2 = nlp(text[0].title())

names = [ent.text for ent in doc2.ents if ent.label\_ == 'PERSON' or ent.label\_ == 'GPE']

else:

names = []

return (text,names[0].split() if len(names)>0 else [])

def get\_email(document):

# A function to extract a email from the text of a resume using regex

emails = []

pattern = re.compile(r'[a-z0-9\.\-+\_]+@[a-z0-9\.\-+\_]+\.[a-z]+')

for line in document:

matches = pattern.findall(line)

for mat in matches:

if len(mat) > 0:

emails.append(mat)

return (emails)

def get\_phone\_no(document):

# A function to extract a phone number from the text of a resume using regex

mob\_num\_regex = r'(\d{3}[-\.\s]??\d{3}[-\.\s]??\d{4}|\(\d{3}\)[-\.\s]\*\d{3}[-\.\s]??\d{4}|\d{5}[-\.\s]??\d{4})'

pattern = re.compile(mob\_num\_regex)

matches = []

for line in document:

match = pattern.findall(line)

for mat in match:

if len(mat) > 9:

matches.append(mat)

return (matches)

stopwords = ["SUMMARY",'PROFILE', 'EDUCATION','EXPERIENCE', 'EMPLOYMENT',

'CERTIFICATES','AWARDS', 'CERTIFICATIONS','ACTIVITIES',

'CURRICULAR',"PROJECTS","SKILLS","CERTIFICATE" ,

"MISCELLANEOUS","COMPETENCIES", "RESEARCH EXPERIENCE",

"SKILL SET", "LEADERSHIP & ACTIVITES","PROJECT EXPERIENCE",

"ACHIEVEMENTS","POSITION OF RESPONSIBILITY",'AWARDS','INTERESTS',

'EMPLOYER I','PROJECT EXPERIENCE','PERSONAL DETAILS',

'ACADEMIC PERFORMANCE','ACADEMIC PROJECT','INTERNSHIP']

def get\_summary(document):

# A function to find the summary section of a resume and scrape the text from the section.

summ = []

sum\_summary\_flag = False

sum\_summary = ''

exact\_matches = ['SUMMARY','PROFESSIONAL SUMMARY','EXPERIENCE SUMMARY','PROFILE']

contains = ['OBJECTIVE']

for line in document:

if line[0].isupper() and (any(line.upper().strip() == word for word in exact\_matches) or any(word in line.upper() for word in contains)) and len(line.strip().split(' '))<4:

if ':' in line:

summ.append(line[line.find(':')+1:])

sum\_summary\_flag = True

elif sum\_summary\_flag:

if (line.strip()[0].isupper() and any(word in line.upper() for word in stopwords)) or line == '':

break

sum\_summary += line

summ.append(line)

return(summ)

def get\_education(document):

# A function to find the education section of a resume and scrape the text from the section.

education = []

edu\_summary\_flag = False

edu\_summary = ''

contains = ['EDUCATION','ACADEMICS','QUALIFICATION']

for line in document:

if any(word in line.upper() for word in contains) and len(line.strip().split(' '))<6:

edu\_summary\_flag = True

if line.find(':')!=-1 and line.strip() != line[:line.find(':')+1].strip():

education.append(line)

elif edu\_summary\_flag:

if (line.strip()[0].isupper() and any(word in line.upper() for word in stopwords)) and len(line.strip().split(' '))<6 or line == '':

break

edu\_summary += line

education.append(line)

return(education)

def get\_university(document):

# A function to find the universites in the text of a resume

univ\_terms = []

univ\_terms = ['college','university','institute','academy','school','vishwavidyalaya','polytechnic']

uni = []

for line in document:

line = line[line.find(':')+1:].strip() if line.find(':')!=-1 else line.strip()

if any(word in line.lower() for word in univ\_terms) and len(line)>2 :

#logger.warning(word)

if line not in uni:

univ = next((x.strip().lower() for x in re.split(',|[|]|[-]|[(]|[)]|\sfrom\s|\sin\s',line) if any(word in x.lower() for word in univ\_terms)), None)

spaces = univ.find(' ')

if spaces!=-1:

uni.append(univ[:spaces])

else:

loc2 = univ.rfind(' ')

if loc2 != -1 and univ[loc2+1:] in univ[:loc2]:

uni.append(univ[:loc2])

else:

uni.append(univ)

return (uni)

def get\_experience(document):

# A function to find the experience section of a resume and scrape the text from the section.

experience = []

exp\_summary\_flag = False

exp\_summary = ''

exact\_matches = ['INTERNSHIP','EXPERIENCE','WORK EXPERIENCE']

contains = ['RELEVANT EXPERIENCE','PROJECT UNDERTAKEN','WORK HISTORY','PROJECTS','PROJECT 1','PROJECT EXPERIENCE',

'DOMAIN EXPERIENCE','EMPLOYMENT DETAILS','EMPLOYEER','EMPLOYER','PROJECT/EXPERIENCE SUMMARY',

'WORK EXPERIENCE SUMMARY','JOB EXPERIENCE','EMPLOYMENT','PROFESSIONAL EXPERIENCE','RELATED EXPERIENCE',

'INTERNSHIPS', 'POSITION OF RESPONSIBILITY','WORK EXPERIENCE','PROJECT NAME','PROJECT DETAILS']

for line in document:

if line[0].isupper() and (any(line.upper().strip() == word for word in exact\_matches) or any(word in line.upper() for word in contains) ) and (len(line.strip().split(' '))<6 or (line.find(':')!=-1 and len(line[:line.find(':')].strip().split(' '))<6)):

exp\_summary\_flag = True

experience.append([line])

elif exp\_summary\_flag:

if line.strip()[0].isupper() and any(word.title() in line or word in line for word in stopwords) and len(line.strip().split(' '))<6 and (line.find(':')==-1 or (any(word in line[:line.find(':')].upper() for word in stopwords) and line.strip() == line[:line.find(':')+1].strip())):

exp\_summary\_flag = False

else:

exp\_summary += line

experience[-1].append(line)

return(experience)

def find\_url\_pdf(file):

# A function to find the links located in a pdf

file = open(file, 'rb')

readPDF = PyPDF2.PdfFileReader(file)

links = []

key = '/Annots'

uri = '/URI'

ank = '/A'

for page in range(readPDF.numPages):

pageSliced = readPDF.getPage(page)

pageObject = pageSliced.getObject()

if key in pageObject.keys():

ann = pageObject[key]

for a in ann:

u = a.getObject()

if uri in u[ank].keys():

links.append(u[ank][uri])

file.close()

return links

def find\_url\_docx(file):

# A function to find the links located in a docx

links = []

document = Document(file)

rels = document.part.rels

for rel in rels:

if rels[rel].reltype == RT.HYPERLINK:

links.append(rels[rel].\_target)

return links

def getLinkedIn(links):

# A function to find the linkedIn link from a list of links

linkedIns = []

for personLinks in links:

temp = list(filter(lambda x: re.search(

'linkedin.com', x), personLinks))

if temp != []:

linkedIns.append(temp[0])

else:

linkedIns.append('')

return linkedIns

def getGithub(links):

# A function to find the gitHub link from a list of links

gits = []

for personLinks in links:

temp = list(filter(lambda x: re.search('github.com', x), personLinks))

if temp != []:

gits.append(temp[0])

else:

gits.append('')

return gits

def parse\_date(x, fmts=("%b %Y", "%B %Y","%b%Y","%b %y","%B %y","%b%y","%Y")):

# A function to parse a string containing a date, trying various formats to see which one matches

for fmt in fmts:

try:

return datetime.strptime(x, fmt)

except ValueError:

pass

def convert\_string\_to\_datetime(date):

# A function to get the datetime representations of strings using regex

if 'Sept' in date and 'September' not in date:

date = date.replace('Sept','Sep')

months = "|".join(calendar.month\_abbr[1:] + calendar.month\_name[1:])

pattern = fr"(?i)((?:{months})? \*\d{{2,4}}) \*(?:-|–|\s)? \*(ongoing|current|present|till date|(?:{months})? \*\d{{2,4}})"

matches = re.findall(pattern, str(date))

if matches:

for start, end in matches:

if end.lower() == "present" or end.lower() == "current" or end.lower() == "ongoing" or end.lower() == "till date":

today = datetime.today()

end = f"{calendar.month\_abbr[today.month]} {today.year}"

return (parse\_date(end), parse\_date(start))

# print(f"{start}-{end} ({duration.years} years, {duration.months} months)")

else:

return None

def get\_exp(texts):

# A function to get the number of months of experience using the difference in datetime elements

total\_experience = 0

for text in texts:

dates = convert\_string\_to\_datetime(text)

if dates!=None:

duration = relativedelta(dates[0], dates[1])

total\_experience = total\_experience + duration.years\*12 + duration.months

else:

total\_experience = total\_experience +1

return(total\_experience)

def check\_overlap\_dates(date1,date2):

# A function to check for overlap in datetime elements

date1 = convert\_string\_to\_datetime(date1)

date2 = convert\_string\_to\_datetime(date2)

if date1 != None and date2 != None and date1[0] !=None and date1[1]!=None and date2[0]!=None and date2[1]!=None and date1[1]>=date2[1] and date1[0]<=date2[0]:

return True

else:

return False

def extractDataJob(jobExperience,finder,company\_tagger):

# A function to extract the job titles, companies, and dates of an individual's work experiences

# using ner fast and job\_description\_finder to locate potential matches in lines. Experience sections with dates

# are processed seperately with companies and job titles being above or 1 below the dates and the rest description.

# Specific checks are implemented using data from previous resumes.

dates = []

companies = []

positions = []

partions = []

partion\_lines = []

companyWords = ['PRIVATE','LIMITED','LTD', 'CO.', 'CORP','COMPANY', 'UNIVERSITY','PVT','INSTITUTE','SCHOOL''.COM','INC','LLC','IT.','HOLDING','INTERNATIONAL']

additionalJobWords = ['INTERN','ENGINEER','LEAD','ARCHITECT','MANAGER','DEVELOPER','ANALYST','CONSULTANT','BUSINESS INTELLIGENCE','DIRECTOR','ROLE','SENIOR','TRAINEE','SR.']

job\_labels = ['DESIGNATION','ROLE']

company\_labels = ['CLIENT','ORGANIZATION','PROJECT','TITLE','CUSTOMER','CLIENT DOMAIN','STAKEHOLDERS']

skills\_labels = ['ENVIRONMENT','SKILLS','APPLICATIONS','DOMAIN','INDUSTRY','SERVICE','DESCRIPTION','RESPONSIBILITES','SIZE','PERIOD','DURATION','TECHNOLOGY','TOOLS','ACHIEVEMENTS']

tech\_labels = ['technologies','environment','applications','skills']

responsiblity\_labels = ['responsibilites','accountabilities','description','tasks']

current = 0

format\_length = -1

group\_of\_positions = ''

overlap = []

for index,line in enumerate(jobExperience):

date = re.search(r"(((jan(uary)?|feb(ruary)?|mar(ch)?|apr(il)?|may|jun(e)?|jul(y)?|aug(ust)?|sep(tember)?|oct(ober)?|nov(ember)?|dec(ember)?)(`)?(\s|-)?\d{2,4}|(\d{2}/)?(\d{2}/\d{2,4}))|\d{1,2}-\w+-\d{2,4})((\s)?(–|-|\s|till|to)(\s)?(((jan(uary)?|feb(ruary)?|mar(ch)?|apr(il)?|may|jun(e)?|jul(y)?|aug(ust)?|sep(tember)?|oct(ober)?|nov(ember)?|dec(ember)?)(`)?(\s|-)?\d{2,4}|(\d{2}/)?(\d{2}/\d{2,4}))|\d{1,2}-\w+-\d{2,4}|present|ongoing|current|till\sdate))?", line.lower())

if date is not None and 'from' not in line and 'during' not in line:

if date.start() == 0:

dates.append(line[:date.end()])

partion\_lines.append(line[date.end()+1:])

partions.append(index)

elif date.end()>(len(line)/2):

dates.append(line[date.start():])

partion\_lines.append(line[:date.start()])

partions.append(index)

for index,date1 in enumerate(dates):

for date2 in dates:

if date1 !=date2 and check\_overlap\_dates(date1,date2):

overlap.append(index)

break

if len(partions) != 0 and (len(jobExperience)/len(partions))>30:

partions = []

if partions != []:

format\_length = partions[0]

position = ''

company = ''

for index,line in enumerate(jobExperience):

label = False

skills\_label = False

company\_label = False

job\_label = False

#########UPDATE CURRENT PARTION#########

if current< len(partions)-1 and index == partions[current+1]-format\_length:

if position == '' and company !='' and overlap!=[]:

group\_of\_positions = company

elif group\_of\_positions != '':

positions.append(position)

companies.append(group\_of\_positions)

else:

positions.append(position)

companies.append(company)

position = ''

company = ''

current = current+1

if current not in overlap:

group\_of\_positions = ''

################Find Job/Company and

if index<partions[current]+3 and index != partions[current]:

try:

job\_check = finder.findall(line)[0].match

except:

job\_check = None

company\_check = None

sentence = Sentence(line)

company\_tagger.predict(sentence)

for entity,labelled in zip(sentence.get\_spans('ner'),sentence.get\_labels('ner')):

if labelled.value == 'ORG' and len(entity.text)>3:

company\_check = entity.text

if company\_check == None and any(word in line.upper() for word in companyWords):

sentence = Sentence(line.upper())

company\_tagger.predict(sentence)

for entity,labelled in zip(sentence.get\_spans('ner'),sentence.get\_labels('ner')):

if labelled.value == 'ORG' and len(entity.text)>3:

company\_check = entity.text

job\_found = bool((job\_check!=None and re.findall('([^a-zA-Z]|^){}([^a-zA-Z]|$)'.format(job\_check.lower()),line.lower()) != []) or any(re.findall('([^a-zA-Z]|^){}([^a-zA-Z]|$)'.format(word.lower()),line.lower()) != [] for word in additionalJobWords))

company\_found = bool((company\_check!=None or any(re.findall('([^a-zA-Z]|^){}([^a-zA-Z]|$)'.format(word.lower()),line.lower()) != [] for word in companyWords)) and (len(line.split())<7 or index < partions[current]))#any(word in line.upper() for word in companyWords)

if ':' in line and line.strip() != line[:line.find(':')+1].strip() and len(line[:line.find(':')].strip())>2:

label = True

job\_label = bool(any(word in line[:line.find(':')].upper() for word in job\_labels))

job\_found = bool(job\_found or job\_label)

company\_label = bool(any(word in line[:line.find(':')].upper() for word in company\_labels))

company\_found = bool(company\_found or company\_label)

skills\_label = bool(any(word in line[:line.find(':')].upper() for word in skills\_labels))

elif index == partions[current]:

try:

job\_check = finder.findall(partion\_lines[current])[0].match

except:

job\_check = None

company\_check = None

sentence = Sentence(partion\_lines[current])

company\_tagger.predict(sentence)

for entity,labelled in zip(sentence.get\_spans('ner'),sentence.get\_labels('ner')):

if labelled.value == 'ORG' and len(entity.text)>3:

company\_check = entity.text

if company\_check == None and any(word in partion\_lines[current].upper() for word in companyWords):

sentence = Sentence(partion\_lines[current].upper())

company\_tagger.predict(sentence)

for entity,labelled in zip(sentence.get\_spans('ner'),sentence.get\_labels('ner')):

if labelled.value == 'ORG' and len(entity.text)>3:

company\_check = entity.text

job\_found = bool((job\_check!=None and re.findall('([^a-zA-Z]|^){}([^a-zA-Z]|$)'.format(job\_check.lower()),line.lower()) != [] ) or any(re.findall('([^a-zA-Z]|^){}([^a-zA-Z]|$)'.format(word.lower()),line.lower()) != [] for word in additionalJobWords))

company\_found = bool(company\_check!=None or any(re.findall('([^a-zA-Z]|^){}([^a-zA-Z]|$)'.format(word.lower()),line.lower()) != [] for word in companyWords))#any(word in partion\_lines[current].upper() for word in companyWords)

if ':' in partion\_lines[current] and partion\_lines[current].strip() != partion\_lines[current][:partion\_lines[current].find(':')+1].strip() and len(partion\_lines[current][:partion\_lines[current].find(':')].strip())>2:

label = True

job\_label = bool(any(word in partion\_lines[current][:partion\_lines[current].find(':')].upper() for word in job\_labels))

job\_found = bool(job\_found or job\_label)

company\_label = bool(any(word in partion\_lines[current][:partion\_lines[current].find(':')].upper() for word in company\_labels))

company\_found = bool(company\_found or company\_label)

skills\_label = bool(any(word in partion\_lines[current][:partion\_lines[current].find(':')].upper() for word in skills\_labels))

partion\_lines[current] = partion\_lines[current][partion\_lines[current].find(':')+1:].strip() if partion\_lines[current].find(':')!=-1 else partion\_lines[current].strip()

line = line[line.find(':')+1:].strip() if line.find(':')!=-1 else line.strip()

if index != partions[current] and not ('ROLE'in line.upper() and 'RESPONSIBILITIES'in line.upper()):

if index < partions[current] and ' at ' in line.lower() and not skills\_label and (job\_check!=None and job\_check in line) and (company\_check != None and company\_check in line):

position = line[:line.lower().find(' at ')]

company = line[line.lower().find(' at ')+4:]

elif index < partions[current] and ' for ' in line.lower() and not skills\_label and (job\_check!=None and job\_check in line) and (company\_check != None and company\_check in line):

position = line[:line.lower().find(' for ')]

company = line[line.lower().find(' for ')+5:]

elif index < partions[current] and ' in ' in line.lower() and not skills\_label and (job\_check!=None and job\_check in line) and (company\_check != None and company\_check in line):

position = line[:line.lower().find(' in ')]

company = line[line.lower().find(' in ')+4:]

elif index < partions[current] and job\_check!=None and company\_check!=None and len(line.strip().split(' '))<10:

position = job\_check

company = company\_check

elif index < partions[current] and ((job\_found and not skills\_label) or job\_label) and not company\_label:

position = line

elif index < partions[current] and ((company\_found and not skills\_label) or company\_label) and not job\_label:

company = line

elif ((len(re.split("[(]|[|]|[-]",line.strip())[0].split(' '))<7 and index == partions[current]+1) or (len(line.strip().split(' '))<4 and index == partions[current]+2)) and company\_found and not job\_found and not skills\_label and (line.count(',')>1 or not any(wn.synsets(word)[0].pos()=='v' for word in line.split(' ') if len(wn.synsets(word))>0)):

company = line

elif ((len(re.split("[(]|[|]|[,]",line.strip())[0].split(' '))<8 and index == partions[current]+1) or (len(line.strip().split(' '))<4 and index == partions[current]+2)) and job\_found and not skills\_label and not any(wn.synsets(word)[0].pos()=='v' for word in re.split("[(]|[|][,]",line.strip())[0] if len(wn.synsets(word))>0):

position = line

elif len(line) > 1 and (line[0] != '(' or line[-1] != ')') and partion\_lines[current].strip()!='' and not ('ROLE'in line.upper() and 'RESPONSIBILITIES'in line.upper()):

if company\_label:

company = partion\_lines[current].strip()

elif job\_label:

position = partion\_lines[current].strip()

else:

check = re.split(",|;|-|[|]",partion\_lines[current].strip())

if len(check)>0 and (' at ' in check[0]) and (job\_check!=None and job\_check in check[0]) and (company\_check != None and company\_check in check[0]):

position = check[0][:check[0].find(' at ')]

company = check[0][check[0].find(' at ')+4:]

elif len(check)>0 and (' for ' in check[0]) and (job\_check!=None and job\_check in check[0]) and (company\_check != None and company\_check in check[0]):

position = check[0][:check[0].find(' for ')]

company = check[0][check[0].find(' for ')+5:]

elif len(check)>0 and (' in ' in check[0]) and (job\_check!=None and job\_check in check[0]) and (company\_check != None and company\_check in check[0]):

position = check[0][:check[0].find(' in ')]

company = check[0][check[0].find(' in ')+4:]

elif len(check)>1:

# if (company\_check != None and company\_check in check[0]):#any(word in check[0].upper() for word in companyWords)

# position = check[1].strip()

# company = check[0].strip()

# else:

# position = check[0].strip()

# company = check[1].strip()

for item in check:

if (job\_check != None and job\_check in item) or any(word in item.upper() for word in additionalJobWords):

position = item.strip()

elif (company\_check != None and company\_check in item):

company = item.strip()

elif len(check) >0 and (company\_check != None and company\_check in check[0]) and not (job\_check != None and job\_check in check[0]):#any(word in check[0].upper() for word in companyWords)

company = check[0]

elif len(check) >0 and (job\_check != None and job\_check in check[0]) or any(word in check[0].upper() for word in additionalJobWords):

position = check[0]

positions.append(position)

if group\_of\_positions == '':

companies.append(company)

else:

companies.append(group\_of\_positions)

return (positions,companies,dates)

else:

##########NON DATE VERSION##

position = ''

company = ''

company\_added = False

tech\_flag = False

respon\_flag = False

for index,line in enumerate(jobExperience):

label = False

skills\_label = False

company\_label = False

job\_label = False

try:

job\_check = finder.findall(line)[0].match

except:

job\_check = None

if any(word in line.lower() for word in tech\_labels) and (len(line.strip().split())<4 or (':' in line and any(word in line.lower()[:line.find(':')] for word in tech\_labels))):

tech\_flag = True

continue

elif tech\_flag and (len(line.strip().split())<6 or line[-1] == ',' or (index>0 and jobExperience[index-1][-1] ==',') or line.count(',')>1) and not (job\_check or (':' in line and any(word in line.upper()[:line.find(':')] for word in job\_labels+company\_labels)) or any(word in line.upper() for word in additionalJobWords) or any(word in line.upper() for word in companyWords)):

continue

elif tech\_flag:

tech\_flag = False

sentence = Sentence(line)

company\_tagger.predict(sentence)

company\_check = None

for entity,labelled in zip(sentence.get\_spans('ner'),sentence.get\_labels('ner')):

if labelled.value == 'ORG' and len(entity.text)>3:

company\_check = entity.text

if any(word in line.lower() for word in responsiblity\_labels) and (len(line.strip().split())<4 or (':' in line and any(word in line.lower()[:line.find(':')] for word in responsiblity\_labels))):

respon\_flag = True

continue

elif respon\_flag and (any(wn.synsets(word)[0].pos()=='v' for word in line.split(' ') if len(wn.synsets(word))>0) or len(line.strip().split())>10 or line[-1] == '.' or ((index>0 and (any(wn.synsets(word)[0].pos()=='v' for word in jobExperience[index-1].strip()[-1].split(' ') if len(wn.synsets(word))>0) or len(jobExperience[index-1].strip().split())>10 or jobExperience[index-1].strip()[-1] =='.')) and (index<len(jobExperience)-1 and (any(wn.synsets(word)[0].pos()=='v' for word in jobExperience[index+1].strip()[-1].split(' ') if len(wn.synsets(word))>0) or len(jobExperience[index+1].strip().split())>10 or jobExperience[index+1].strip()[-1] =='.')))) and not (job\_check or (':' in line and any(word in line.upper()[:line.find(':')] for word in job\_labels+company\_labels))):

continue

elif respon\_flag:

respon\_flag = False

job\_found = bool((job\_check!=None and re.findall('([^a-zA-Z]|^){}([^a-zA-Z]|$)'.format(job\_check.lower()),line.lower()) != []) or 'INTERN' in line.upper() or 'SR.' in line.upper())

company\_found = bool(company\_check!=None and len(line.split())<6 )#any(word in line.upper() for word in companyWords)

if ':' in line and line.strip() != line[:line.find(':')+1].strip():

label = True

job\_label = bool(any(word in line[:line.find(':')].upper() for word in job\_labels))

company\_label = bool(any(word in line[:line.find(':')].upper() for word in company\_labels))

job\_found = bool(job\_found or job\_label)

company\_found = bool(company\_found or company\_label)

skills\_label = bool(any(word in line[:line.find(':')].upper() for word in skills\_labels))

line = line[line.find(':')+1:].strip() if line.find(':')!=-1 else line.strip()

if len(line.strip())<2:

continue

check = re.split(",|;|-|[|]",line.strip())

if len(line)>0 and (line[0] != '(' or line[-1] != ')') and len(line.strip().split(' '))<7 and not skills\_label and not ('ROLE'in line.upper() and 'RESPONSIBILITIES'in line.upper()):#and not any(word in line.upper() for word in description\_words)

if ' at ' in line:

position = line[:line.find(' at ')]

company = line[line.find(' at ')+4:]

elif not any(wn.synsets(word)[0].pos()=='v' for word in line.split(' ') if len(wn.synsets(word))>0) or company\_label or job\_label:

if len(check)>1 and not label:

for item in check:

if (any(word in item.upper() for word in additionalJobWords) or (job\_check != None and job\_check in item)) or job\_label:#any(word in item.upper() for word in companyWords)

position = item

elif ((company\_check != None and company\_check in item) or company\_label) and not company\_added:#(any(word in item.upper() for word in companyWords) and not any(word in item.upper() for word in additionalJobWords))

company = item #company\_check

elif job\_found and (len(re.split("[(]|[|]",line.strip())[0].split(' '))<7 or job\_label):

position = line

elif company\_found and (len(line.strip().split(' '))<7 or company\_label) and not job\_found and not company\_added:

company = line

if position != '':

positions.append(position)

position = ''

if company != '':

companies.append(company)

company = ''

company\_added = True

else:

company\_added = False

return (positions,companies,dates)

def getEduInfo(education):

# A function to extract the degrees, dates, courses, accolades, and majors from an education section in a resume. It does so using rules from previous resumes

# and regex checks to find where items are located.

degrees = []

dates = []

courses = []

accolades = []

majors = []

keywords = ["BACHELOR", "MASTER", "DEGREE", "MS ", "M.S.","B.TECH", "B. TECH","B.E","BACHELORS","M.TECH","M. TECH",'BACHERLORS','DIPLOMA','BTECH','M.C.A','MS:','PHD','B.SC','MBA ','MA -']

bachelor\_keywords = ["BACHELOR","B.TECH","B. TECH","B.E","BACHELORS",'BACHERLORS','BTECH','B.C.A','B.SC']

masters\_keywords = ["MASTER", "MS ", "M.S.","M.TECH","M. TECH",'M.C.A','MBA','MS:','MA -']

PHD\_keywords = ['DOCTOR','PH.D','PHD','POSTGRADUATE']

univ\_terms = ['college','university','institute','academy','school','vishwavidyalaya']

for index,line in enumerate(education):

date = re.search(r"((Jan(uary)?|Feb(ruary)?|Mar(ch)?|Apr(il)?|May|Jun(e)?|Jul(y)?|Aug(ust)?|Sep(tember)?|Oct(ober)?|Nov(ember)?|Dec(ember)?)(`)?(\s)?\d{2,4}((\s)?(–|-|\s)(\s)?((Jan(uary)?|Feb(ruary)?|Mar(ch)?|Apr(il)?|May|Jun(e)?|Jul(y)?|Aug(ust)?|Sep(tember)?|Oct(ober)?|Nov(ember)?|Dec(ember)?)(`)?(\s)?\d{2,4}|Present|Ongoing))?|\d{2}/\d{4}((\s)?(–|-|\s)(\s)?(\d{2}/\d{4}|Present|Ongoing))?|\d{4}((\s)?(–|-|\s)(\s)?(\d{4}|Present|Ongoing))?)", line)

course = re.search(r"COURSE(S|WORK)?(:)?(\s)?", line.upper())

if date is not None:

dates.append(line[date.start():date.end()])

if course is not None:

courses.append(line[course.end():].strip())

if any(word in line.upper() for word in keywords):

accolades.append([])

gpa = max(line.upper().find("GPA"),line.upper().find("CPI"))

minlen = len(line)

if gpa!=-1:

minlen = min([gpa,minlen])

if date is not None and any(word in line[:date.start()].upper() for word in keywords):

minlen = min([minlen,date.start()])

elif date is not None and not any(word in line[:date.start()].upper() for word in keywords):

line = line[date.end()+1:]

line = line[:minlen]

check = re.split(";|[|]",line)#|,

if len(check)>1:

for i in check:

if any(word in i.upper() for word in keywords):

degree = i.strip(', ()')

if ' in ' in degree or ',' in degree:

split\_edu = re.split("\sin\s|,|:|\swith\s|\sfrom\s|[(]|[)]",degree)

else:

split\_edu = re.split("\sin\s|\sof\s|,|:|\swith\s|\sfrom\s|[(]|[)]",degree)

ind = [index for index in range(len(split\_edu)) if any(word in ' ' + split\_edu[index].upper() + ' ' for word in keywords)]

majors.append(split\_edu[ind[0]+1] if len(ind)!= 0 and len(split\_edu)>1 and ind[0]+1<len(split\_edu) and not any(word in line.lower() for word in univ\_terms) else '')

if any(word in degree.upper() for word in bachelor\_keywords):

degrees.append('Bachelors')

elif any(word in degree.upper() for word in masters\_keywords):

degrees.append('Masters')

elif any(word in degree.upper() for word in PHD\_keywords):

degrees.append('Ph.D')

else:

degree = check[0].strip(', ()')

if len(re.split("\sin\s|\sof\s|,|:|\swith\s|\sfrom\s|[(]|[)]",degree))==1 and index<len(education)-1:

start = 0

end = len(degree)

if '(' in degree:

start = degree.find('(')

if ')' in degree:

end = degree.find(')')

if 'from' in degree and degree.find('from')>end:

end = degree.find('from')

if start!=0 or end!=len(degree):

majors.append(degree[start:end] if not any(word in degree[start:end].lower() for word in univ\_terms) else '')

else:

majors.append(education[index+1] if not any(word in education[index+1].lower() for word in univ\_terms) else '')

else:

if ' in ' in degree or ',' in degree:

split\_edu = re.split("\sin\s|,|:|\swith\s|\sfrom\s|[(]|[)]",degree)

else:

split\_edu = re.split("\sin\s|\sof\s|,|:|\swith\s|\sfrom\s|[(]|[)]",degree)

ind = [index for index in range(len(split\_edu)) if any(word in ' ' + split\_edu[index].upper() + ' ' for word in keywords)]

majors.append(split\_edu[ind[0]+1] if len(ind)!= 0 and len(split\_edu)>1 and ind[0]+1<len(split\_edu) and not any(word in split\_edu[ind[0]+1].lower() for word in univ\_terms) else '')

if any(word in degree.upper() for word in bachelor\_keywords):

degrees.append('Bachelors')

elif any(word in degree.upper() for word in masters\_keywords):

degrees.append('Masters')

elif any(word in degree.upper() for word in PHD\_keywords):

degrees.append('Ph.D')

elif course is None and not any(word in line.lower() for word in univ\_terms) and len(degrees)>0:

# special = re.search(r"[^a-zA-Z0-9\.]", line)

if not bool(re.match("[^0-9a-zA-Z\.'()]+",line)) and line.upper().find(",") == -1 and line.upper().find("MAJOR") == -1 and date is None and max(line.upper().find("GPA"),line.upper().find("CPI"),line.upper().find("PERCENTAGE")) ==-1:

accolades[-1].append(line)

elif len(line.split(' '))>3 and line.upper().find('CLUB') == -1 and line.upper().find('LEADERSHIP') == -1 and date is None:

if len(courses)>0:

courses[-1] = courses[-1] + ' ' + line

else:

courses.append(line)

return degrees,dates,courses,accolades,majors

if \_\_name\_\_ == '\_\_main\_\_':

# application.run(debug=True)

from waitress import serve

serve(application, host="0.0.0.0", port=9090)

"""data = {'First\_Name': First\_Name, 'Last\_Name': Last\_Name, 'Email': email\_ids, "LinkedIn URL": linkedIn[0], "GitHub URL": gitHub[0], 'Contact\_Number': phone\_nos, 'Summary': summary\_1,

'University\_1': univ\_1, 'University\_2': univ\_2, 'University 1 Degree': degree1, 'University 2 Degree': degree2, 'University 1 GPA': gpa\_1, 'University 2 GPA': gpa\_2,

"University 1 Courses": courses1, "University 2 Courses": courses2, "University 1 Major": major1, "University 2 Major": major2, "University 1 Accolades": accolades1,

"University 2 Accolades": accolades2, "University 1 Dates": eduDates1, "University 2 Dates": eduDates2, "University Ranking 1": univ\_1\_Ranking, "University Ranking 2": univ\_2\_Ranking,

'Work Experince 1 Position': positions1, 'Work Experince 1 Company Name': companies1, 'Work Experince 1 Dates worked': dates1, 'Work Experince 1 Descriptions': descriptions1,

'Work Experince 2 Position': positions2, 'Work Experince 2 Company Name': companies2, 'Work Experince 2 Dates worked': dates2, 'Work Experince 2 Descriptions': descriptions2,

'Total work experience': experience\_total, 'Skills': skills\_1, 'Type of skill': skill\_descriptions, 'Skill Tools': skills\_2, "Jobs associated with Skills": jobs\_with\_skill,

'Certificates': certificates\_1, 'Activities': activities\_1}

wb = op.Workbook()

ws = wb.active

ws.title = 'extracted\_data'

for ind, title in enumerate(list(data.keys())):

ws.cell(row=1, column=ind+1).value = title

for ind, col in enumerate(list(data.values())):

for ind2, item in enumerate(col):

ws.cell(row=ind2+2, column=ind+1).value = str(item)

wb.save('extracted\_data.xlsx')"""