Glassdoor Scrapper Documentation

Files:

Scrapper_glassdoor.py

This is the file which scrapes the data from the glassdoor job postings and sends email to the user.

properties.json

This is the json file which contains the server name, username, password, database name. These are required to connect to the database.

Variables:

Data - to store the loaded json file for database connection.

final_skills – to store the skills ids and the corresponding skills.

Mapping_dict – to store the mapping between the resume id and the corresponding skill id.

Email_dict – to store the resume id and the corresponding email id of the users.

Final_dict – to store the job description links scrapped from glassdoor.

Total – to store the user id of the user and the job links which matches with skills in the resume of the user.

Code segments:

1. **Db_connect(properties)**

This function loads the data from the properties.json file and connects to the database. It returns the connection object.

```
# ---------------Database Connector Script ----------------------------------
def db_connect(properties):
    import mysql.connector
    from mysql.connector import Error
    import json
    data = json.load(properties)
    server_name = data['server_name']
    user_name = data['user_name']
    password = data['password']
    db_name = data['db_name']
    connection = mysql.connector.connect(host=server_name,
                                         database=db_name,
                                         user=user_name,
                                         password=password)

    return connection
#                   fetch total skills database
```

2. **get_total_skills(connection)**

This function executes the query to fetch the skill id and the corresponding skills from the skill_master table. The result is stored in the dictionary name final_skills.

```
def get_total_skills(connection):
    query = "select skill_id,skill_title from skill_master"
    cursor = connection.cursor()
    cursor.execute(query)
    table = cursor.fetchall()
    final_skills = {}
    for row in table:
        final_skills[row[0]]=row[1]
    return final_skills
```

3. **get_resume_id_skills(connection)**

This function executes the query to fetch the resume_id of the user and the corresponding skill id from the resume_skills table. The result is stored in the dictionary name mapping_dict.

```
def get_resume_id_skills(connection):
    query1 = "select r.resume_id,r.skill_id from resume_skills r where is_active='1'"
    cursor = connection.cursor()
    cursor.execute(query1)
    records = cursor.fetchall()
    mapping_dict = {}
    for row in records:
        if row[0] in mapping_dict:
            mapping_dict[row[0]].append(row[1])
        else:
            mapping_dict[row[0]] = [row[1]]
    return mapping_dict
```

4. **get_email_id_users(connection)**

This function executes the query to fetch the resume id and the email id of the user from user_resume and user_master tables based on a join. The result is stored in the dictionary name email_dict.

```
def get_email_id_users(connection):
    query2 = "select r.resume_id,u.user_email from user_resume r join user_master u on r.user_id=u.user_id"
    cursor = connection.cursor()
    cursor.execute(query2)
    details = cursor.fetchall()
    email_dict = {}
    for row in details:
        if row[0] in email_dict:
            email_dict[row[0]].append(row[1])
        else:
            email_dict[row[0]] = [row[1]]
    return email_dict
```

5. **get_job_description(keyword,num_jobs,verbose)**

This function is used to scrape the glassdoor website, the function takes the keyword, i.e the title of the job which the user wants to apply and the number of jobs openings which the user wants to see.

First all the jobs postings urls are collected and stored in a list named job_urls.

```
final_dict = {}
threshold = 1
def get_job_description(keyword,num_jobs,verbose):
     options = Options()
     options.add_argument("--window-size-1920,1200")
     options.add_argument('--headless')
     options.add_argument('--no-sandbox')
     options.add_argument('--disable-dev-shm-usage')
     driver  =  webdriver.Chrome (options=options,executable_path=ChromeDriverManager().install())
     url = "https://www.glassdoor.com/Job/jobs.htm?suggestCount=0&suggestChosen=false&clickSource=searchBtn&typedK
     driver.get(url)
     job_urls = []
     c=0
     job_buttons = driver.find_elements_by_xpath('.//a[@class = "jobLink job-search-key-1rd3saf eigr9kq1"]')  #jl
     time.sleep(2)
     print(len(job_buttons))
     for text in job_buttons:
         if text.get_attribute('href'):                      ### get all the job postings URL's
            job_urls.append(text.get_attribute('href'))
            c=c+1
            if(c>=num_jobs):
               break;
```

Then the job_urls list is iterated to get the job description and each job description is stored in list named final_dict

```
# ========== Iterate through each url and get the job description ================================
    for i in job_urls:
            time.sleep(5)
            jobs = []
            driver.get(i)
            button = driver.find_element_by_xpath('//*[@id="JobDescriptionContainer"]/div[2]')
            button.click()
            job_description = driver.find_element_by_xpath('//*[@id="JobDescriptionContainer"]/div[1]').text
            jobs.append(job_description)
            final_dict[i] = job_description
    return final_dict
```

6. **ke.get_user_id_to_list_of_job_ids(mapping_dict,final_dict,connection,final_skills,threshold)**

This function is taken from a different python file named keyword_extraction_modules.

This function returns user id with their corresponding job openings links that matches with the skills in the resume of the user.

This file uses four different functions:

a. **get_list_of_matched_skills(description, total_skills)**

This function takes the description and matches with skills from the skills table. It returns the list of matches skills

```
def get_list_of_matched_skills(description, total_skills):
    list_of_skills_matched = []
    description = description.upper()
    desc_list = description.split(" ")
    for skill in total_skills:
        if (total_skills[skill].upper() in desc_list) or ((total_skills[skill].upper() + ".") in desc_list) or (tota
            list_of_skills_matched.append(skill)
    #print(list_of_skills_matched)
    return list_of_skills_matched
```

b. **get_dict_with_list_of_skills_from_description(links_description_dict, total_skills)**

This function creates a dictionary of job description and skills using the get_list_of_matched_skills function.

```
def get_dict_with_list_of_skills_from_description(links_description_dict, total_skills):
    job_desc_link_and_skills_dict = dict()
    for link in links_description_dict:
        description = links_description_dict[link]
        skills_matched_in_curr_description = get_list_of_matched_skills(description, total_skills)
        job_desc_link_and_skills_dict[link] = skills_matched_in_curr_description
    return job_desc_link_and_skills_dict
```

c. **match_both_lists(list_of_skills_in_resume, list_of_skills_in_description, threshold, total_skill_count)**

This function returns true if the % of skills in the resume matches skills in the job description is greater than the threshold.

```
def match_both_lists(list_of_skills_in_resume, list_of_skills_in_description, threshold, total_skill_count):
    count = 0
    for skill in list_of_skills_in_description:
        if skill in list_of_skills_in_resume:
            count += 1
    if ((count / total_skill_count) * 100 >= threshold): return True
    return False
```

d. **get_user_id_to_list_of_job_ids(resume_skills_dict, links_description_dict, db_connection, total_skills, threshold)**

This function returns the user ids and the corresponding job links the matches with the skills in the resume.

```
def get_user_id_to_list_of_job_ids(resume_skills_dict, links_description_dict, db_connection, total_skills, threshol
    result_dict = dict()
    job_desc_link_and_skills_dict = get_dict_with_list_of_skills_from_description(links_description_dict, total_skil
    for curr_resume in resume_skills_dict:
        list_of_skills_in_resume = resume_skills_dict[curr_resume]
        for link in job_desc_link_and_skills_dict:
            list_of_skills_in_description = job_desc_link_and_skills_dict[link]
            if match_both_lists(list_of_skills_in_resume, list_of_skills_in_description, threshold, len(total_skills
                if curr_resume in result_dict:
                    curr_list = result_dict[curr_resume]
                else:
                    curr_list = []
                curr_list.append(link)
                result_dict[curr_resume] = curr_list
    return result_dict
```

Sending email to the user

7. The result from ke.get_user_id_to_list_of_job_ids(mapping_dict,final_dict,connection,final_skills,threshold) and get_email_id_users are used to send email to the user using smtplib library

```
port = 587
smtp_server = "smtp.gmail.com"
login = "srijas.alerts@gmail.com"
password = "SRIJASGMAILPWD"
sender = "srijas.alerts@gmail.com"
for key in total:
    if key in email_dict:
        receiver = ''.join(email_dict[key])
        print(receiver)
        msg = MIMEMultipart()
        msg['From'] = sender
        msg['To'] = receiver
        msg['Subject'] = 'JOB Listing'
        body = """Hi \n PFA the attached list of jobs that match your resume \n """
        temp_str = ""
        list_curr_links = total[key]
        counter=1
        for link in list_curr_links:
            temp_str+= (str(counter) + link + '\n')
            counter+=1
        body+=temp_str
        msg.attach(MIMEText(body, 'plain'))
        text = msg.as_string()
```

```
    try:
        server = smtplib.SMTP(smtp_server, port)
        server.connect(smtp_server,port)
        server.ehlo()
        server.starttls()
        server.ehlo()
        server.login(login, password)
        server.sendmail(sender, receiver, text)
        server.quit()
        print('Sent')
    except (gaierror, ConnectionRefusedError):
        print('Failed to connect to the server. Bad connection settings?')
    except smtplib.SMTPServerDisconnected as e:
        print('Failed to connect to the server. Wrong user/password?')
        print(str(e))
    except smtplib.SMTPException as e:
        print('SMTP error occurred: ' + str(e))
```