JOBS RECOMMENDER SYSTEM

Ayush Jain, Shweta Pathak

{jain.ayu, pathak.sh} @husky.neu.edu

CSYE 7245, Spring 2019, Northeastern University

***Abstract*— In today's competitive world, it is very difficult to find the right type of employment which compliments the skills that an individual possesses. Further, due to the amount of job-profiles present in the industry, it is difficult for an individual to understand which profile is most suitable for him. Similarly, companies face the challenge of selecting a candidate best suitable for a specific job from among the list of thousands of participating candidates. Thus, a need to understand the requirements of a job profile present in the company and the skills complimenting it is needed to make this job-search process easier for both the participating company as well as the candidates.**

In this project, we develop a recommender system to advocate the job-profiles present in a company based on the requirements present in their description and a candidate's skills mentioned in the resume. For this we have first scrapped the data from *GlassDoor*'s site to get the job postings. We chose few postings with titles such as 'Data Analyst', 'Data Scientist', 'Business Intelligence Analyst' and 'Data Engineer' for the states of 'Massachusetts', 'Texas' and 'California'. The technology of Selenium with Python was utilized for performing this data scrapping. The next step involved parsing a resume to get the skills that a candidate possesses. For this we parsed sample resumes using python and were able to get the name of the candidates and their respective skill-set. The final step of this project involved curation of the Recommender System in which we used *Cosine Similarity* to find the similarity between the candidate skill-set and the company's required skills for a specific job-posting.

This paper contains an introduction to the problem that is faced and the development of our jobs-recommender system as its solution.

Keywords Selenium. Scrapping. Parsing. Python. Cosine Similarity. Recommender System.

I. INTRODUCTION

In today's competitive world, it is a task for individuals to find jobs which match their skill-set. With the advancement in the technology, there has been hike in the job profiles and the posting platforms. Further, there are too many jobs available with different job-description names and it becomes confusing to a candidate as to which profile matches him perfectly. The company on the other hand receives about 1000+ resumes per day and it is impossible to go through each of them and find a candidate whose skill-sets match the company's requirements.

The Internet-based recruiting platforms become a primary recruitment channel in most companies. While such platforms decrease the recruitment time and advertisement cost, they suffer from an inappropriateness of traditional information retrieval techniques like the Boolean search methods.[1]

The main motto behind the implementation of a Recommender System for jobs is to make this process easier by suggesting the correct job-profiles to apply to the candidates. This can be suggested using variety of criterion such as *Salary*, *Company* and *Data-Skills*. This system aims to ease the process of job applications and providing efficiency to the process. This will enable the matching of a candidate's skills and personal interests better with the company's requirements for a specific job-profile. Further the confusion with respect to different job-names having similar skill-requirements will also be solved since the recommender system will recommend based on skills and not just the job-names.

The overall idea of developing this Recommender System for Jobs is to help curb the confusion that exists due to the presence of two different job-names with same requirements, enhance the application system for both the candidate and the company. Consequently, a vast number of candidates miss the opportunity of applying to certain job-profiles due to several reasons ranging from different job-names, not enough information on the required skills or different job-board postings. The Recommender System will help the candidates in this case by providing them with all the job-profiles of various companies which match their personal interests such as Salary requirements, Company name and most importantly their skill-set.

II. METHODOLOGY

We divided the project into 3 main tasks:

1. Get the Job Listings from a Job-Board
2. Get the Details from a candidate's Resume
3. Generate a Recommender System for Jobs

These tasks define the working for the generation of the Recommender System. For this project, we curated 3 python notebooks namely, 'WebScraping_GlassDoor', 'Resume_Parser' and 'Recommender System'. Each of these notebooks contain the above mentioned tasks.

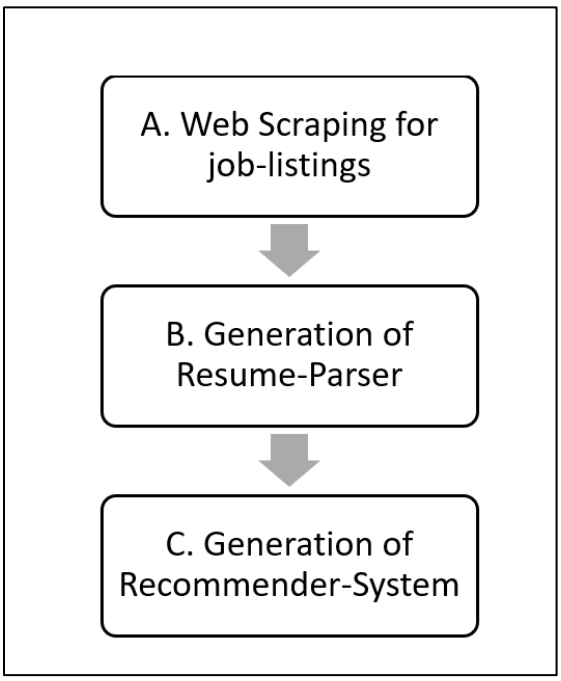

Fig 1: Work-Flow of the Project

A. WebScraping_GlassDoor

Web Scraping (also termed Screen Scraping, Web Data Extraction, Web Harvesting etc.) is a technique employed to extract large amounts of data from websites whereby the data is extracted and saved to a local file in your computer or to a database in table (spreadsheet) format.[2]

Web Scraping can be done in python using Beautiful Soup. However, this module does not work for some additions of the websites which are written using JavaScript. Because of this, we used Selenium in combination with Python. The Selenium package is used to automate web browser interaction from Python. With Selenium, programming a Python script to automate a web browser is possible. Afterwards, those pesky JavaScript links are no longer an issue.[3] Hence we imported various drivers for working with Selenium and a web-browser.

```
#importing the stop_words to get the important stop words
from stop_words import get_stop_words
import nltk
from nltk import word_tokenize
from nltk.corpus import stopwords
from collections import Counter

#importing the selenium webdriver
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.common.action_chains import ActionChains
from selenium.common.exceptions import StaleElementReferenceException
from selenium.common.exceptions import NoSuchElementException
```

Fig 2: Importing Web drivers and related content for Selenium

The first step of our project involved getting the job-profiles for various companies using some job-board. For this purpose, we used GlassDoor as our job-board and performed web-scraping on it. This helped us to acquire various job-titles, company names, company-locations, company-glassdoor-ratings, job descriptions and company glassdoor salaries. We did this to get all those criterions from three different states, namely 'MA', 'CA' and 'TX'.

```
#initialize loop variables
i=0
results=pd.DataFrame(columns=['Title','Company','Location','Description','Salary','Company_Rating','Skillset'])


for location_input in ("CA","MA","TX"):
    for job_title_input in ("Data Scientist","Data Analyst","Business Intelligence Analyst","Data Engineer"):
        #call search_jobs function
        search_jobs(driver, job_title_input, location_input)
```

Fig 3: Features Scraped from GlassDoor using Selenium & python

```
listing.click()
sleep(2)
inf=listing.find_element_by_xpath("//div[@class='header']")

inf.find_element_by_xpath("//h1[@class='jobTitle h2 strong']")
#storing the title of the job posting
title = inf.find_element_by_xpath("//h1[@class='jobTitle h2 strong']").text


sleep(2)

info= listing.find_element_by_xpath("//div[@class='compInfo']")
sleep(2)

try:
    info.find_element_by_xpath("//span[@class='compactRating lg margRtSm']")
    #storing the rating of the job posting
    rating= info.find_element_by_xpath("//span[@class='compactRating lg margRtSm']").text
    #storing the company_name of the posting
    company_name = info.find_element_by_xpath("//a[@class='plain strong empDetailsLink']").text
    #storing the company_location of the posting
    company_location=info.find_element_by_xpath("//div[@class='compInfo']//span[2]").text
```

Fig 4: Storing the scraped data from GlassDoor

The next step involves tokenizing and finding the percentage of most common skills mentioned in the scraped job description. Tokenization is "the process of breaking a stream of text up into words, phrases, symbols, or other meaningful elements called tokens.[4] In this process we also remove the stopwords to get more skills and with better frequency. Stop-words are those words in natural language which carry no own meaning and serve the purpose of connecting other words together to create grammatical sentences.[5] Examples of some Stopwords are: 'and', 'is', 'it' etc. This helps us focus on the right words which are skills in our case.

```
def tokenize_description(description):
    """take a job description and return a list of tokens excluding stop words""
    tokens = word_tokenize(description)
    stopset = set(stopwords.words('english'))
    tokens = [w.lower() for w in tokens if not w in stopset]
    text = nltk.Text(tokens)
    return list(set(text))

def find_skills_frequency(results_df):
    """count frequency of key words (as defined in dictionaries within function)
    words = []
    for description in results_df['Description']:
        words.append(tokenize_description(description))

    doc_frequency = Counter()
    [doc_frequency.update(word) for word in words]
```

Fig 5: Tokenizing the description

Figure 5 displays the method that we created for tokenization which essentially splits the data into common blocks (skills) that is used later for comparison. We found the percentage of occurrence of each skill in the total number of scraped job listings for the purpose of comparison.

	Term	NumPostingsPercentage
29	SQL	49.627329
41	Bachelor	39.006211
1	Python	36.211180
12	Excel	26.832298
36	Statistics	26.583851
0	R	25.031056
37	Mathematics	19.254658
13	Tableau	18.571429
42	Master	18.447205

Fig 6: Tokenized Skills and their percentage

Finally, we found the skill-set present in each of the scraped job descriptions and saved these results in the form of output CSV.

Title	Company	Location	Description	Salary	Company_Rating	Skillset
Senior Data Scientist	Instacart	– San Francisco, CA	Founded in 2012, Instacart is a leader in Nort...	144K–193K (Glassdoor est.)	3.7 ★	[python, sql, dashboards, leader]
DATA SCIENTIST / ANALYTIC CONSULTANT 4	Wells Fargo	– San Francisco, CA	Job Description\n\nAt Wells Fargo, we want to ...	82K–131K (Glassdoor est.)	3.5 ★	[python, hive, tensorflow, keras, machine lear...
Sr. Applications Scientist – Charged Particle ...	Multibeam	– Santa Clara, CA	Sr. Applications Scientist – Charged Particle ...	Employer Provided Salary:100K–135K	5.0 ★	[leader, data analysis]
Principal Scientist	bioMérieux	– San Diego, CA	World leader in the field of in vitro diagnost...	91K–126K (Glassdoor est.)	3.4 ★	[leader, phd, data analysis]

Fig 7: Output CSV data for Webscraping_GlassDoor

B. Resume_Parser

Our second notebook 'Resume_Parser' deals with parsing the candidate's resume to get his relevant details. Parsing is a method of extracting important and needed data from a document. We use it here for our project to generate a list of skills present in a candidate's resume.

AYUSH JAIN

1185-6, Boylston Street, Boston, MA, 02215

617-784-4883 | mailayush94@gmail.com | www.linkedin.com/in/ayush-jain | https://github.com/Ayush0910

EDUCATION

Northeastern University, Boston, MA — **Expected Aug 2019**
Master of Science in Information Systems — **(GPA 3.75)**
Courses: Data Management & Database Design, Advances in Data Science, Data Warehousing and Business Intelligence

Mumbai University, Mumbai, India — **Jul 2013-Aug 2017**
Bachelor of Engineering in Information Technology — **(GPA 3.6)**

TECHNICAL SKILLS

Database: MySQL, Oracle 11g, SQL Server, PostgreSQL, Microsoft Azure, MS Access, MongoDB
Programming languages: SQL, Python, PL/Sql, Java
BI Tools: Tableau, Qlik Sense, Power BI, Qlik View, PowerPivot, SSRS
Tools: Talend, SSIS, Informatica, Alteryx, Toad, Microsoft Office Suite, SSAS, AWS, Apache Spark
Machine Learning: Pandas, Numpy, Sklearn, Seaborn, Scikit-learn, Matplotlib, Tensorflow, Keras, H2O

PROFESSIONAL EXPERIENCE

Northeastern University, Boston, MA
Graduate Teaching Assistant – Data Warehousing and Business Intelligence — **May 2018-Present**
- Guiding students in learning Data Analysis, Business Analysis, Data Modelling, Data Integration and Data Warehousing
- Assisting graduate students with creating dashboards and visualizations in **Tableau**, **Qlik Sense** and **Power BI**
- Mentoring students to perform Data Integration using ETL tools like **Talend** and **SSIS**

Fig 8: A Part of Sample Resume

Initially we define a method to calculate the ngrams of the text present in the resume. Ngrams are basically set of words which occur together, and N can be 1, 2 or any positive integer which can be used for various machine learning algorithms.
Assume a sentence 'K', to find Ngrams in that statement we can utilize the formula given below where 'X' is the number of words present in that statement.

$$Ngrams_K = X - (N - 1)$$

We used ngrams in our project to find important words present in a candidate's resume. These wordings can be unigrams (N=1), bigrams (N=2), trigrams (N=3) or associated with any positive integer.

```
#create a method to count the ngrams of the text
def generate_ngrams(words_list, n):
    ngrams_list = []

    for num in range(0, len(words_list)):
        ngram = ' '.join(words_list[num:num + n])
        ngrams_list.append(ngram)

    return ngrams_list
```

Fig 9: Method to create Ngrams for the Resume-Parser

Next, we defined a list of skills which we are trying to find in the candidate's resume. This involves skills such as 'Python', 'R', 'Data Mining', 'Tableau' etc. After defining this list, we run the parser-loop to check if any of the skills mentioned in the above list is present in the candidate's resume. If a skill is present, it would be appended to a 'Skill-set' list which we generated.

```
#running the loop till it goes through all the paragraphs of the document
while(i<len(doc.paragraphs)):
    a = doc.paragraphs[i].text
    a= a.lower()
    words = a.split()
    for item in skill:
        for word in words:
            if item == word:
                if item not in skillset:
                    skillset.append(word)
```

Fig 10: Parser Code

We also generated a bigram parser code to be able to capture words such as 'Data-Mining', 'Machine-Learning' etc.

```
words_list = process_text(a)
# finding the bigrams in the document
bigrams = generate_ngrams(words_list, 2)
two.append(bigrams)
for sublist in two:
    for item in sublist:
        flat_list_one.append(item)
for words in flat_list_one:
    for items in skill:
        if items == words:
            if items not in skillset:
                skillset.append(words)
i=i+1
```

Fig 11: Bigram Parser Code

Finally, we extract the name and skills of a candidate using the

above methods and parser code. We generate an output csv of the same.

Name	Skills
AYUSH JAIN	[mongodb, sql, oracle, java, qlik sense, qlik ...
POOJITH SHANKAR SHETTY	[sql, oracle, java, aws, nosql, mongodb, machi...
Prashant Reddy	[python, oracle, machine learning, big data, a...
SHWETA PATHAK	[big data, data mining, sql, oracle, pl/sql, j...

Fig 12: Output of the Resume_Parser

C. Recommender System

A recommender system or recommendation system (sometimes replacing "system" with a synonym such as platform or engine) is a subclass of information filtering system that seeks to predict the "rating" or "preference" a user would give to an item.[6] Our final python notebook is the curation of the Jobs Recommender System using the earlier mentioned two output CSV files.

Title	Company	Location	Description	Salary	Company_Rating	Skillset
Senior Data Scientist	Instacart	– San Francisco, CA	Founded in 2012, Instacart is a leader in Nort...	144K–193K (Glassdoor est.)	3.7 ★	['python', 'sql', 'dashboards', 'leader']
DATA SCIENTIST / ANALYTIC CONSULTANT 4	Wells Fargo	– San Francisco, CA	Job Description\r\n\r\nAt Wells Fargo, we want...	82K–131K (Glassdoor est.)	3.5 ★	['python', 'hive', 'tensorflow', 'keras', 'mac...
Sr. Applications Scientist – Charged Particle ...	Multibeam	– Santa Clara, CA	Sr. Applications Scientist – Charged Particle ...	Employer Provided Salary: 100K–135K	5.0 ★	['leader', 'data analysis']
Principal Scientist	bioMérieux	– San Diego, CA	World leader in the field of in vitro diagnost...	91K–126K (Glassdoor est.)	3.4 ★	['leader', 'phd', 'data analysis']
Data Engineer	LeadCrunch	– San Diego, CA	Data Engineer\r\n\r\nAre you ready to be a par...	Employer Provided Salary: 125K–155K	4.1 ★	['python', 'hadoop', 'leader', 'data mining']

Fig 13: Input Data of CSV 1 (Web Scraped Data)

Name	Skills
AYUSH JAIN	['mongodb', 'sql', 'oracle', 'java', 'qlik sen...
POOJITH SHANKAR SHETTY	['sql', 'oracle', 'java', 'aws', 'nosql', 'mon...
Prashant Reddy	['python', 'oracle', 'machine learning', 'big ...
SHWETA PATHAK	['big data', 'data mining', 'sql', 'oracle', '...

Fig 14: Input Data of CSV 2 (Resume Parsed Data)

In order to find the similarity between the skills from the job-postings and that of the candidate's resume, we used 'Cosine Similarity' as it is an efficient way of finding how similar two documents are. Basically, the cosine similarity is the dot product of two vectors divided by the product of the magnitude of each vector. We divide the dot product by the magnitude because we are measuring only angle difference. On the other hand, dot product is taking the angle difference and magnitude into account. If we divide the dot product by the product of each vector's magnitude, we normalize our data and only measure the angle difference. Dot product is a better measure of similarity if we can ignore magnitude.[7]

```
#define a method to calculate the consine similarity
def counter_cosine_similarity(c1, c2):
    terms = set(c1).union(c2)
    dotprod = sum(c1.get(k, 0) * c2.get(k, 0) for k in terms)
    magA = math.sqrt(sum(c1.get(k, 0)**2 for k in terms))
    magB = math.sqrt(sum(c2.get(k, 0)**2 for k in terms))
    return dotprod / (magA * magB)
```

Fig 15: Method defined for Cosine Similarity

We then run the defined method of Cosine Similarity for each candidate's parsed data against all the job descriptions scraped from Glassdoor. This is depicted in Fig 16.

```
#find the cosine_similarity for all the rows in the dataframe
j=0
k=0
while(k<len(res)):
    if(res['Name'][k] !='N'):
        i=0
        while(i<len(comp['Skillset'])):
            counterA = Counter(comp['Skillset'][i])
            counterB = Counter(res['Skills'][k])
            sim = counter_cosine_similarity(counterA,counterB)
```

Fig 16: Cosine Similarity Code

This similarity score is then appended to the data which contains each candidate's details.

	Candidate_Name	Job_Title	Company_Name	Company_Location	Estimated_Salary	Company_Rating	Similarity_score
0	AYUSH JAIN	Senior Data Scientist	Instacart	– San Francisco, CA	144K–193K (Glassdoor est.)	3.7 ★	0.927982
1	AYUSH JAIN	DATA SCIENTIST / ANALYTIC CONSULTANT 4	Wells Fargo	– San Francisco, CA	82K–131K (Glassdoor est.)	3.5 ★	0.931434
2	AYUSH JAIN	Sr. Applications Scientist – Charged Particle ...	Multibeam	– Santa Clara, CA	Employer Provided Salary:100K–135K	5.0 ★	0.861279
3	AYUSH JAIN	Principal Scientist	bioMérieux	– San Diego, CA	91K–126K (Glassdoor est.)	3.4 ★	0.913769
4	AYUSH JAIN	Data Engineer	LeadCrunch	– San Diego, CA	Employer Provided Salary:125K–155K	4.1 ★	0.905892

Fig 17: Appended Similarity Score to the data-frame

Next, we use pattern-matching to extract the float salary values from all the job listings. There are entries where Salary is mentioned in per-hour scale. We converted it to per-year scale to maintain the consistency of data.

```
while(i<len(job)):
    st= job['Estimated_Salary'][i]
    string=''.join(st)
    #pattern if the salary is mentioned per hour
    pat=re.compile('.*Per Hour')
    matching=pat.findall(string)
    if matching:
        mat1=''.join(matching)
        mt1=mat1.replace('$','').replace('Per Hour','')
        lower1,upper1=mt1.split('-')
        low1=int(lower1)*40*52
        upp1=int(upper1)*40*52
        average1=(low1+upp1)/2000
        fin.loc[i]['Estimated_Salary']=average1


    else:
        #pattern if the salary is mentioned yearly
        pattern=re.compile('.*K-.*K')
        match=pattern.findall(string)
        mat=''.join(match)
        mt=mat.replace('$','').replace('K','').replace('Employer Provided Salary:','')
        lower, upper=mt.split('-')
        low=int(lower)
        upp=int(upper)
        average=(low+upp)/2
        fin.loc[i]['Estimated_Salary']=average
```

Fig 18: Use of Pattern-Matching

We used the same format of code for application of pattern-matching on Ratings extracted.

Estimated_Salary	Company_Rating	Similarity_Score
168.50	3.7	0.927982
106.50	3.5	0.931434
117.50	5.0	0.861279
108.50	3.4	0.913769
140.00	4.1	0.905892

Fig 19: Salary, Ratings and Similarity Score Displayed

Next, we find the minimum, maximum and mean of the Salary and Ratings score. This is done to normalize these scores.

```
#finding the mean,min and max of salary and rating
mean_salary=fin['Estimated_Salary'].mean()
mean_rating=fin['Company_Rating'].mean()
min_salary=fin['Estimated_Salary'].min()
min_rating=fin['Company_Rating'].min()
max_salary=fin['Estimated_Salary'].max()
max_rating=fin['Company_Rating'].max()
```

Fig 20: Mean, Min, Max of Salary and Ratings

Since our Similarity Score lies between 0 and 1, we needed to normalize the salaries and ratings to generate their scores between 0 and 1. This will help us in assigning the weighted score to those fields.

$$\text{Normalized Salary} = \frac{(\text{Present Value} - \min(\text{Salary}))}{(\max(\text{Salary}) - \min(\text{Salary}))}$$

$$\text{Normalized Rating} = \frac{(\text{Present Value} - \min(\text{Rating}))}{(\max(\text{Rating}) - \min(\text{Rating}))}$$

```
i=0
while(i<len(fin)):
    sal=abs(fin['Estimated_Salary'][i]-min_salary)/abs(max_salary-min_salary)
    rat=abs(fin['Company_Rating'][i]-min_rating)/abs(max_rating-min_rating)
    sal=(sal*30)/100
    rat=(rat*20)/100
    sim=fin['Similarity_Score'][i]
    sim=(sim*50)/100
    total=sal+rat+sim
```

Fig 21: Normalizing Salary and Ratings

Our final step is to provide weights to each of these criterions so that we can generate a Total Score which is used to recommend the jobs. Our total score is calculated by summing 30% of salary, 20% of ratings and 50% of the similarity score.

Total Score = 50% (Cosine Similarity Score) + 30% (Normalized Salary) + 20% (Normalized Rating)

Using this Total Score, we display the recommendation results in a descending order of the score. This means that the first displayed result is the candidate's most-recommended. We also ensured that for every candidate, the similarity score of the skills should only be displayed if it is above 0.5. This ensures that there is maximum similarity.

Candidate_Name	Job_Title	Company_Name	Estimated_Salary	Company_Rating	Total_Score
SHWETA PATHAK	Senior Data Scientist	Wealthfront	205	4.4	0.881685
SHWETA PATHAK	Technical Lead - Data Engineering	Grand Rounds	182	4.7	0.872227
SHWETA PATHAK	Principal Scientist, Computational Biology	Tesaro	224.5	3.8	0.853375
SHWETA PATHAK	Lead Data Scientist	BCG Digital Ventures	183.5	4.2	0.847308
SHWETA PATHAK	Senior Data Scientist	Intercom	203	4	0.846437
SHWETA PATHAK	Senior Software Engineer, MuleSoft Integration...	New Relic	179.5	4.5	0.843018
SHWETA PATHAK	Senior Applied Scientist - Machine Learning	Wealthfront	187	4.4	0.841617

Fig 22: Recommendation System Results for a candidate

III. RESULTS

Since we have 3 different Python notebooks for this project, we have displayed the results for each notebook.

a) Notebook-1 (WebScrapping_GlassDoor):

The result of this notebook displays the scraped data from GlassDoor's website which includes the 'Title' of job, 'Company' name, 'Location' of the company, 'Description' included in the job-listing, 'Salary', 'Company_Rating' on glassdoor and the 'Skillset' mentioned in their job-postings. This result is posted on python and extracted as a CSV.

Title	Company	Location	Description	Salary	Company_Rating	Skillset
Senior Data Scientist	Instacart	– San Francisco, CA	Founded in 2012, Instacart is a leader in Nort...	144K–193K (Glassdoor est.)	3.7 ★	[python, sql, dashboards, leader]
DATA SCIENTIST / ANALYTIC CONSULTANT 4	Wells Fargo	– San Francisco, CA	Job Description\n\nAt Wells Fargo, we want to ...	82K–131K (Glassdoor est.)	3.5 ★	[python, hive, tensorflow, keras, machine lear...
Sr. Applications Scientist – Charged Particle ...	Multibeam	– Santa Clara, CA	Sr. Applications Scientist – Charged Particle ...	Employer Provided Salary:100K–135K	5.0 ★	[leader, data analysis]
Principal Scientist	bioMérieux	– San Diego, CA	World leader in the field of in vitro diagnost...	91K–126K (Glassdoor est.)	3.4 ★	[leader, phd, data analysis]

Fig 23: Result of Notebook 1

b) Notebook-2 (Resume_Parser):

The result of this notebook displays the name of the candidate whose resume was parsed and the skills that he/she has mentioned. This result has also been displayed in python and extracted as a CSV.

Name	Skills
AYUSH JAIN	[mongodb, sql, oracle, java, qlik sense, qlik ...
POOJITH SHANKAR SHETTY	[sql, oracle, java, aws, nosql, mongodb, machi...
Prashant Reddy	[python, oracle, machine learning, big data, a...
SHWETA PATHAK	[big data, data mining, sql, oracle, pl/sql, j...

Fig 24: Result of Notebook 2

c) Notebook-3 (Recommender System):

The result of this notebooks displays the recommended job-listings for each of the candidate based on the metrics we mentioned above.

Candidate_Name	Job_Title	Company_Name	Estimated_Salary	Company_Rating	Total_Score
SHWETA PATHAK	Senior Data Scientist	Wealthfront	205	4.4	0.881685
SHWETA PATHAK	Technical Lead - Data Engineering	Grand Rounds	182	4.7	0.872227
SHWETA PATHAK	Principal Scientist, Computational Biology	Tesaro	224.5	3.8	0.853375
SHWETA PATHAK	Lead Data Scientist	BCG Digital Ventures	183.5	4.2	0.847308
SHWETA PATHAK	Senior Data Scientist	Intercom	203	4	0.846437
SHWETA PATHAK	Senior Software Engineer, MuleSoft Integration...	New Relic	179.5	4.5	0.843018
SHWETA PATHAK	Senior Applied Scientist - Machine Learning	Wealthfront	187	4.4	0.841617

Fig 25: Result of Notebook 3

IV. METRICS

We have used 3 different metrics for this project

1. **Cosine Similarity (Similarity Score):**
   Find this between the job skills requirement and the candidate skillset.

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$

2. **Normalized Salary:**

$$\text{Normalized Salary} = \frac{(\text{Present Value} - \min(\text{Salary}))}{(\max(\text{Salary}) - \min(\text{Salary}))}$$

Normalized Rating:

$$\text{Normalized Rating} = \frac{(\text{Present Value} - \min(\text{Rating}))}{(\max(\text{Rating}) - \min(\text{Rating}))}$$

3. **Total Score:**

$$\text{Total Score} = 50\%\,(\text{Cosine Similarity Score}) + 30\%\,(\text{Normalized Salary}) + 20\%\,(\text{Normalized Rating})$$

V. CONCLUSION

Our project concludes with a Recommendation System which can recommend job-postings posted on a job-board (GlassDoor) to a candidate which the candidate should focus on applying by finding the similarity between the candidate's skillset and skills in the job description. We also considered Salary and Ratings as a criterion to filter out postings for a candidate. We achieved this Recommender System with the help of Cosine Similarity metric.

We can further extend the scope and working of this Recommender System by utilizing it for more job-boards and postings. We can make use of multiple criterion.

VI. REFERENCES

[1] Shaha Alotaibi, "A Survey of Job Recommender Systems", ResearchGate [Online] https://www.researchgate.net/publication/272802616_A_survey_of_job_recommender_systems

[2] Web Scrapping, WebHarvy [Online] https://www.webharvy.com/articles/what-is-web-scraping.html

[3] WebScraping in Python using Selenium-BeautifulSoup-Pandas, Medium [Online] https://medium.freecodecamp.org/better-web-scraping-in-python-with-selenium-beautiful-soup-and-pandas-d6390592e251

[4] Jeffery Fosset Blog, "Tokenizing Raw Text in Python" [Online] http://jeffreyfossett.com/2014/04/25/tokenizing-raw-text-in-python.html

[5] Martina Pugliese, "English Stopwords and Python Libraries", GitHub [Online] https://martinapugliese.github.io/english-stopwords/

[6] Recommender System, Wikipedia [Online] https://en.wikipedia.org/wiki/Recommender_system

[7] Song Recommendation System using Cosine Similarity and Euclidian Distance, Medium [Online] https://medium.com/@mark.rethana/building-a-song-recommendation-system-using-cosine-similarity-and-euclidian-distance-748fdfc832fdW