Kevin Raj
CSCE5200 - Information Retrieval and Web Search
4/20/2023

Project Final Report: NLP Resume Screener and Chatbot

Project Name

NLP Resume Screener and Chatbot

Participants

Kevin Raj
Email: KevinRaj@my.unt.edu

Workflow

I worked by myself on this project to develop the project idea, writing the report, and developing the code with the goal of finishing the project by the final report deadline.

Google Colab Code Workbook Link

https://colab.research.google.com/drive/1kuBxk92hc2D61ZZvbQ2cECkiMOX_sYlL?usp=sharing

Project Abstract

The goal of this project is to design an automated solution that can assist the HR function with screening resumes and will also utilize a Chatbot to conduct pre-screening interviews with potential candidates for company job postings. According to an article published by ideal.com, on average it takes around 23 hours to screen resumes for one job posting. Additionally, when a job posting receives around 250 resumes – around 75% to 88% are unqualified and do not meet the qualifications required in the job posting. The resources devoted to this manual process can constrain the HR department and can ultimately result in bad hiring decisions, increased costs, poor employee performance, and higher employee turnover.

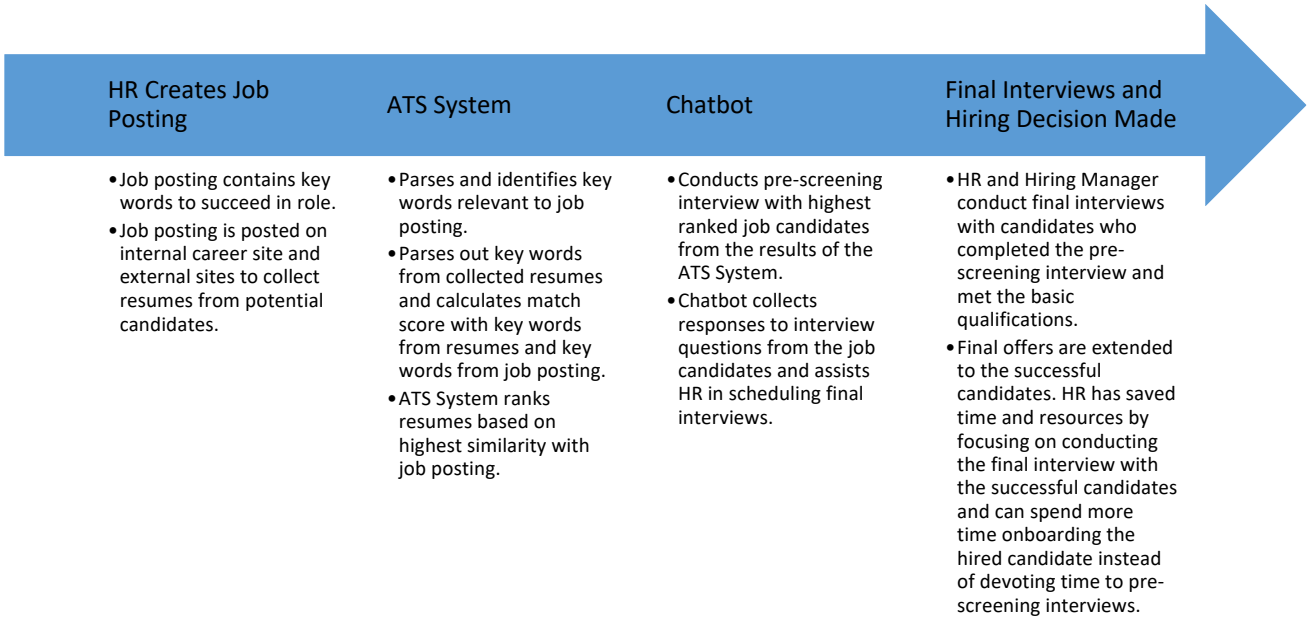
Applicant Tracking Systems (ATS) have been developed over the years to assist with screening resumes through key word matching and ranking of potential candidates. Based on the job posting, the ATS will define certain key words that are relevant to the job posting (examples: Python coding, project management, SAP implementation, etc). The ATS will then scan through the resumes and parse out key words from each resume. The ATS will determine a match score between the key words found from the resumes and the key words relevant to the job posting. The ATS will then rank the resumes based on the match score between the resume key words and job posting key words. This project will aim to develop an ATS system that can assist with parsing through resumes and ranking the best matches to a given job description.

Based on the qualified resumes selected, HR will then need to conduct pre-screening interviews with the potential job candidates. This is also a time-intensive process for the HR department and can result in delayed hiring decisions that can reflect poorly on an organization. A Chatbot can assist with this process by conducting the pre-screening interview with potential job candidates on behalf of the HR department. The Chatbot can ask basic questions such as:

- What is the earliest date you are available to interview?
- What is your ideal work environment (hybrid, on-site, remote)?
- What is your contact information (phone, email, address)?

By having the Chatbot conduct the pre-screening interviews, HR should have more time to focus on conducting the final interviews and onboard the successful candidate(s).

Workflow Idea Diagram



Data Abstraction

The dataset is from Kaggle and is called the Resume Dataset (link: https://www.kaggle.com/datasets/gauravduttakiit/resume-dataset)

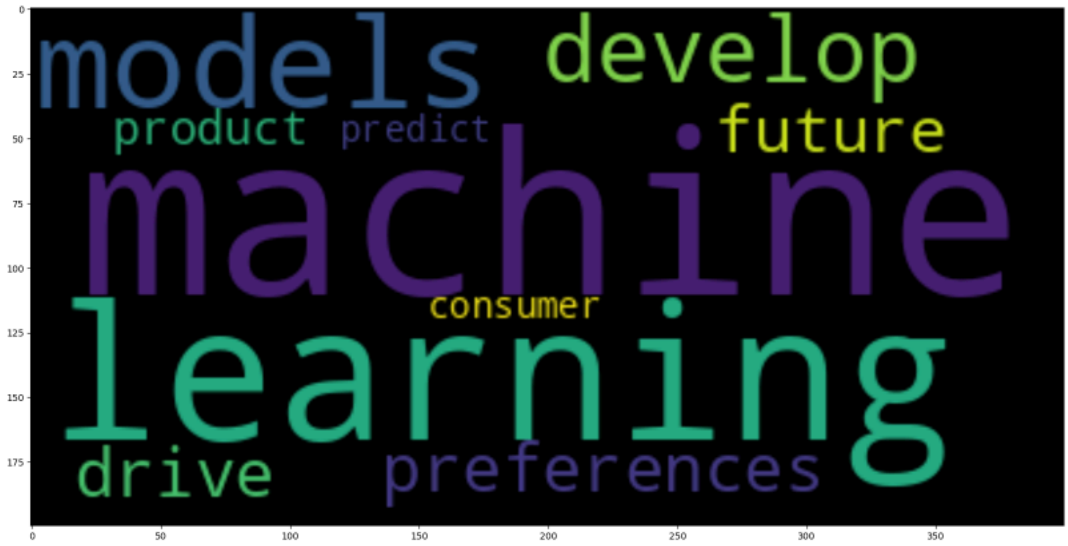
This dataset contains 962 resumes and has two columns:

- Category: this is the job category the resume is focused on.
- Resume: this is the plain text of the candidate's resume.

Since the dataset has plain text resumes, the initial intent was to parse out the relevant sections of each resume such as name, contact info, skills, education, and experience. Once the resumes are parsed, I would then be able to identify the key words and match the key words up with the key words from the job description.

Project Design

First step, I created a job requisition and identified the relevant key words within the requisition that the Applicant Tracking System (ATS) will look for in the resumes from the job candidates. After creating the job requisition, I removed numbers and punctuation using RegEx, removed stop words, and created a word cloud showing the most frequent key words in the requisition (code from tutorial: Parsing Job Profiles by Abhishek Chhibber: https://abhishekchhibber.com/2017/07/05/parsing-job-profiles-through-nlp/). The output of the word cloud showed the job description having these key words:



Also, here is the code to retrieve the top 10 key words in the job description:

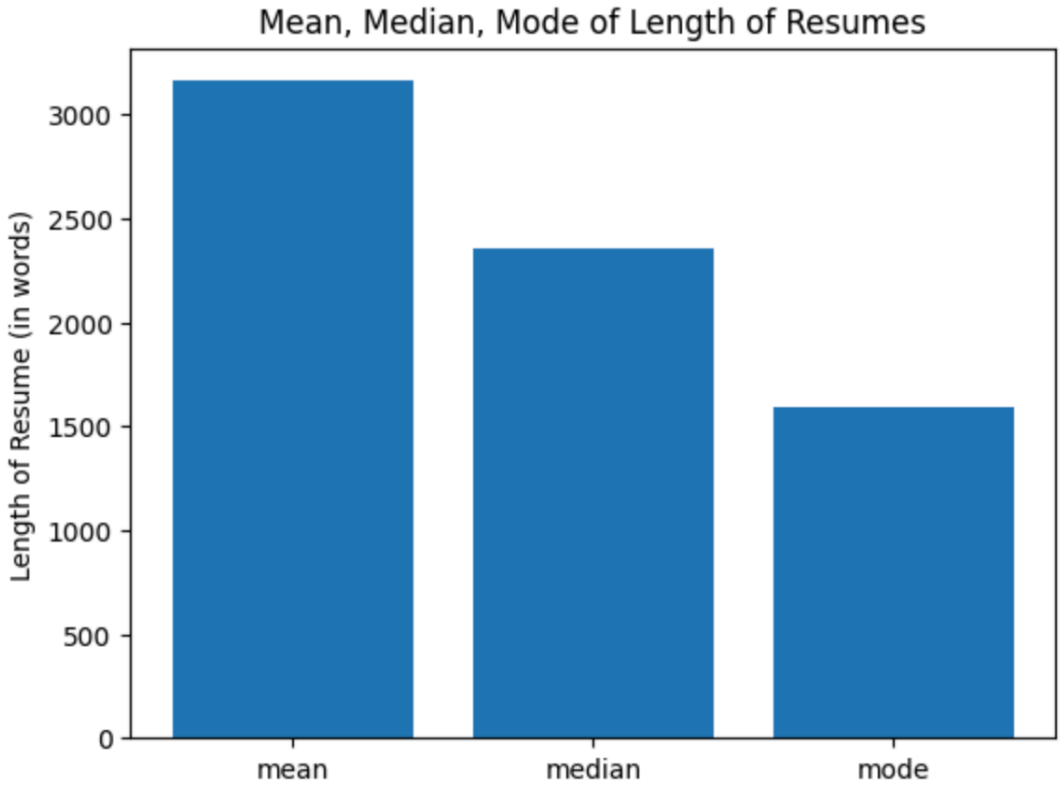
```
# Retrieve the top 10 key words in the job description

fdist_pos = nltk.FreqDist(token_list_stripped)
top_10_words = fdist_pos.most_common(10)
print(top_10_words)
```

```
[('machine', 4), ('learning', 4), ('models', 3), ('develop', 2), ('preferences', 2), ('drive', 2), ('future', 2), ('product', 2), ('pre
```

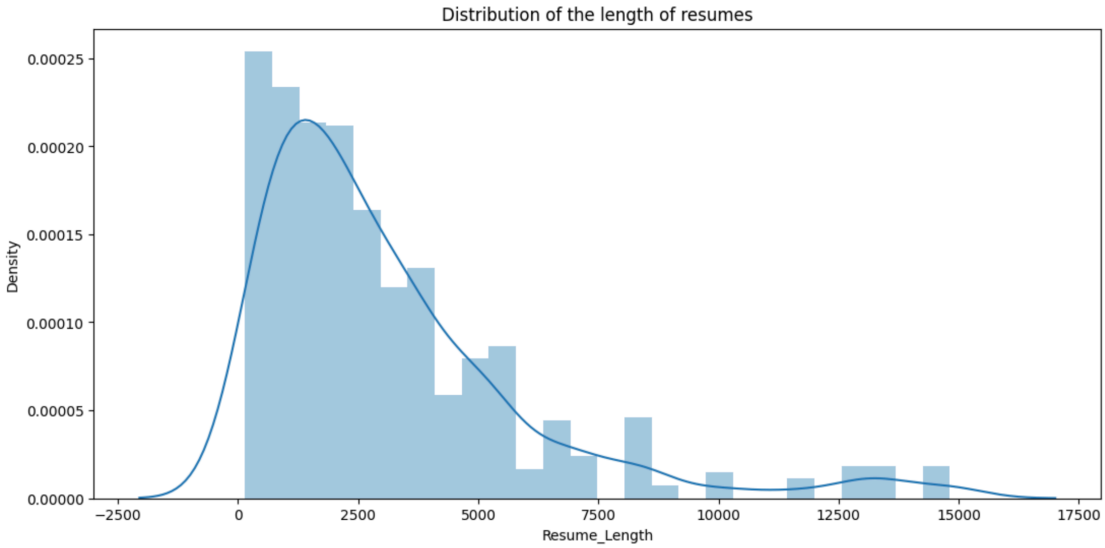
After examining the key words in the job description, I loaded the csv file of the dataset of resumes from Kaggle called the Resume Dataset (link: https://www.kaggle.com/datasets/gauravduttakiit/resume-dataset). Once the data for the resume dataset was loaded, I performed Exploratory Data Analysis (EDA) on the

dataset to better understand the data and have a better idea of key words to look for in the dataset. As part of the EDA, I looked at the mean, median, and mode of the length of the word length of the resumes and visualized the results in a bar chart using matplotlib (matplotlib documentation: https://matplotlib.org/stable/tutorials/introductory/pyplot.html#sphx-glr-tutorials-introductory-pyplot-py):

```
Mean length of resumes: 3160.364864864865
Median length of resumes: 2355.0
Mode length of resumes: 1590
```

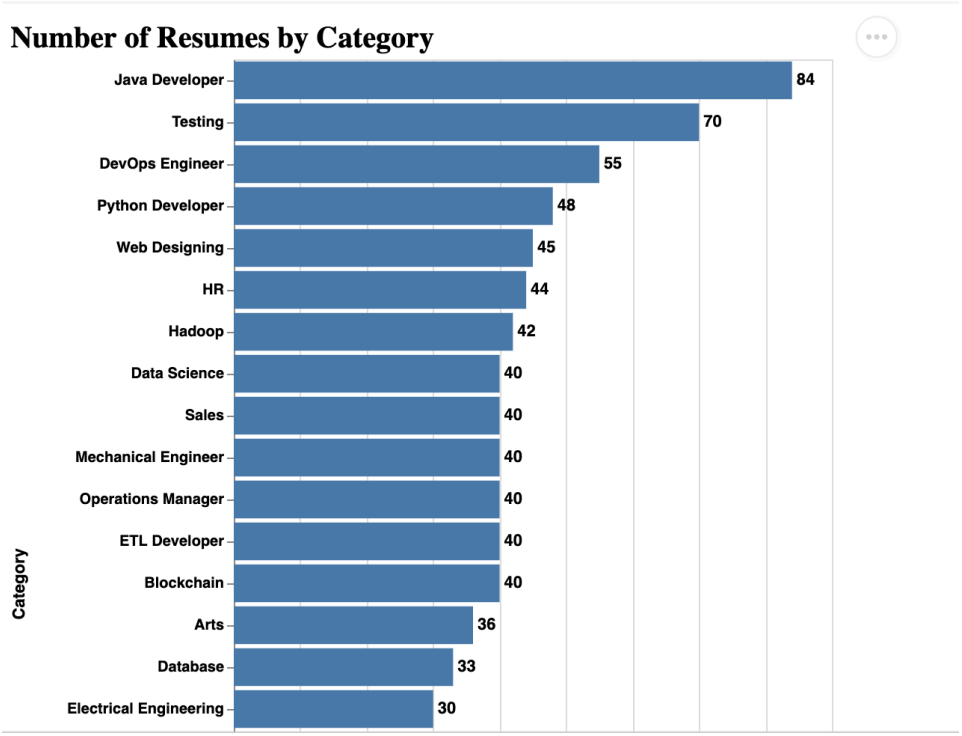


- I also used Seaborn python package to look at the distribution of the length of the resumes (tutorial code from LAKPA SHERPA code on Kaggle: https://www.kaggle.com/code/sherpalakpa18/resume-screener)



- We can see from the distribution that most of the resumes are in the 2000–2500-word length.

- After looking at the length of words distribution, I used the Altair python package to examine how many resumes were in each title category. Here is the horizontal bar chart showing the number of resumes by title category in descending order (Altair documentation: https://altair-viz.github.io/):



- From the bar chart, we can see most of the candidates in the resume dataset have a job title of java developer, testing, devops engineer, or Python developer. This is helpful for HR to understand what kind of job requisitions to hire for and the experience backgrounds of the talent pool.

- After exploring the resume dataset, the resumes needed to be preprocessed and prepped to be able to extract the relevant key words from each resume that match with the key words in the job description. Here is the code that creates new column in the resume Panda data frame called 'cleaned_up_resume (tutorial code from: https://www.analyticsvidhya.com/blog/2021/06/resume-screening-with-natural-language-processing-in-python/)

```
resume_processing_df = pd.read_csv('/content/UpdatedResumeDataSet.csv', encoding='utf-8')
def cleanResume(resumeText):
    resumeText = re.sub('http\S+\s*', ' ', resumeText)  # remove URLs
    resumeText = re.sub('#\S+', '', resumeText)  # remove hashtags
    resumeText = re.sub('@\S+', '  ', resumeText)  # remove mentions
    resumeText = re.sub('[%s]' % re.escape("""!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~"""), ' ', resumeText)  # remove punctuations
    resumeText = re.sub(r'[^\x00-\x7f]',r' ', resumeText)
    resumeText = re.sub(r'\b(\w+)\s+\1\b', r'\1', resumeText) # remove duplicate words
    resumeText = re.sub(r'\s+', ' ', resumeText)  # remove extra whitespace
    return resumeText



resume_processing_df['cleaned_up_resume'] = resume_processing_df.Resume.apply(lambda x: cleanResume(x))


resume_processing_df
```

	Category	Resume	cleaned_up_resume
0	Data Science	Skills * Programming Languages: Python (pandas...	Skills Programming Languages Python pandas num...
1	Data Science	Education Details \r\nMay 2013 to May 2017 B.E...	Education Details May 2013 to May 2017 B E UIT...
2	Data Science	Areas of Interest Deep Learning, Control Syste...	Areas of Interest Deep Learning Control System...
3	Data Science	Skills â€¢ R â€¢ Python â€¢ SAP HANA â€¢ Table...	Skills R Python SAP HANA Tableau SAP HANA SQL ...
4	Data Science	Education Details \r\n MCA YMCAUST, Faridab...	Education Details MCA YMCAUST Faridabad Haryan...

- We can observe from the output that the cleaned_up_resume column is now leaving out the unnecessary characters. From the cleaned_up_resume column, we can show the term frequencies and generate a word cloud to examine which key words are showing up in the resumes from the dataset (tutorial code from Kaggle: https://www.kaggle.com/code/aurelienkcz/resume-nlp). Here is the output and code for the word cloud showing the key words in the resume dataset:

```
# Generate Word Cloud of Frequent Words
# From Kaggle code tutorial https://www.kaggle.com/code/aurelienkcz/resume-nlp

wordcloud = WordCloud(width = 1000, height = 500).generate(",".join(str(v) for v in rec_data_tokens))

plt.imshow(wordcloud)
plt.axis("off")
plt.title('Wordcloud Data Science resume')
```

```
Text(0.5, 1.0, 'Wordcloud Data Science resume')
```

- Next, we will clean the job description and extract the key words from the job description. Based on the key words extracted from the job description, we can match the resumes from the dataset that have key words in the resumes that match up with the key words from the job description. Here is the code to find the key words in the job description and clean up unnecessary stop words, punctuation, numbers, and lower case the words for consistency in the job description (tutorial code link: https://oindrilasen.com/2021/05/build-resume-scanner-using-python-nlp/

```
#Find the key words in the job description

#From tutorial code https://oindrilasen.com/2021/05/build-resume-scanner-using-python-nlp/


def clean_job_decsription(jd):
    ''' a function to create a word cloud based on the input text parameter'''
    ## Clean the Text
    # Lower
    clean_jd = jd.lower()
    # remove punctuation
    clean_jd = re.sub(r'[^\w\s]', '', clean_jd)
    # remove trailing spaces
    clean_jd = clean_jd.strip()
    # remove numbers
    clean_jd = re.sub('[0-9]+', '', clean_jd)
    # tokenize
    clean_jd = word_tokenize(clean_jd)
    # remove stop words
    stop = stopwords.words('english')
    clean_jd = [w for w in clean_jd if not w in stop]
    return(clean_jd)

with open('/content/job_description.txt', 'r') as data:
    job_description = data.read()

clean_jd = clean_job_decsription(job_description)
print(clean_jd)
```

```
['develop', 'machine', 'learning', 'models', 'predict', 'consumer', 'travel', 'preferences', 'based', 'past', 'preferences', 'develop', 'machine', 'learning'
```

Here is the updated list of key words that we will look for in the job description:

```
relevant_skills = ['machine learning', 'models', 'predict', 'recommend', 'collaborate', 'development', 'team', 'deploy', 'production', 'troubleshoot', 'vuln

print(relevant_skills)
```

```
['machine learning', 'models', 'predict', 'recommend', 'collaborate', 'development', 'team', 'deploy', 'production', 'troubleshoot', 'vulnerabilities', 'code
```

- We will then loop through the resumes in the Kaggle dataset and find the key words in each resume in the dataset that match up with the key words from the job description. Here is the code to parse the key words from each resume in the dataset:

```
# Loop through the resumes in the Kaggle dataset and extract the key words matching up with the job description
# Tutorial code from https://medium.com/mlearning-ai/automatic-skill-extraction-from-resumes-using-spacy-710507624a1e

resume_texts = resumes
for idx, text in enumerate(resume_texts):
    doc = nlp(text)
    unique_skills = set()
    print(f"Skills in Resume {idx + 1}:")
    for ent in doc.ents:
        if ent.label_ == "SKILL":
            unique_skills.add(ent.text)
    for skill in unique_skills:
        print(skill)
    print()
```

```
Data

Skills in Resume 6:
machine learning
Data
Machine Learning
data
```

- The next step will be to calculate a ratio of the key words found in each candidate's resume with the total key words in the job description. For example, if a candidate has machine learning, models, predict, and deploy in their resume then there are 4 key terms that match out of a total of 20 key terms in the job description (score 4/20 or 20%). Here is the code to loop through the resumes and assign a count of relevant key words in the resume that match up to the job description. The code then divides by the count by the total key words in the job description and finds the top 5 resumes with the highest ratio score displaying the resume # and ratio score (tutorial code from: https://medium.com/mlearning-ai/automatic-skill-extraction-from-resumes-using-spacy-710507624a1e

```
# Loop through the resume data set and calculate the ratio of matching key words in the job description
# Tutorial code from: https://medium.com/mlearning-ai/automatic-skill-extraction-from-resumes-using-spacy-710507624a1e

resume_texts = resumes
resume_skills = []
relevant_skills = ['machine learning', 'models', 'predict', 'recommend', 'collaborate', 'development', 'team', 'deploy', 'production', 'troubleshoot', 'vul
skill_counts = []

for idx, text in enumerate(resume_texts):
    doc = nlp(text)
    skills = list(set([ent.text.lower() for ent in doc.ents if ent.label_ == "SKILL"]))
    count = len(skills)
    ratio = count / len(relevant_skills)
    skill_counts.append((idx, ratio))
    resume_skills.append(skills)


sorted_list = sorted(skill_counts, key=lambda x: x[1], reverse=True)
top_5_resumes = sorted_list[:5]
for idx, ratio in top_5_resumes:
    print(f"Resume {idx+1} has a skill count ratio of {ratio:.2f}")
```

```
Calculate the ratio of skills matching key words from job description fo
from tutorial code: https://medium.com/mlearning-ai/automatic-skill-extr


esume_texts = resumes
esume_skills = []
elevant_skills = ['machine learning', 'models', 'predict', 'recommend',
kill_counts = []

or idx, text in enumerate(resume_texts):
   doc = nlp(text)
   skills = list(set([ent.text.lower() for ent in doc.ents if ent.label_
   count = len(skills)
   ratio = count / len(relevant_skills)
   skill_counts.append((idx, ratio))
   resume_skills.append(skills)


orted_list = sorted(skill_counts, key=lambda x: x[1], reverse=True)
op_5_resumes = sorted_list[:5]
or idx, ratio in top_5_resumes:
   print(f"Resume {idx+1} has a skill count ratio of {ratio:.2f}")
```

```
Resume 4 has a skill count ratio of 0.40
Resume 14 has a skill count ratio of 0.40
Resume 24 has a skill count ratio of 0.40
Resume 34 has a skill count ratio of 0.40
Resume 512 has a skill count ratio of 0.40
```

- After finding the top 5 resumes with the highest ratio of job skills key words found in the resume with the total key words in the job description, we can create the ChatBot to collect pre-screening information from the job candidates. The questions the ChatBot will ask are:
    - What is your email address?
    - What is your phone number?
    - Would you like to work remotely, hybrid, or in-office?
    - Please enter first and last name of reference?
    - Please enter phone number for reference?
    - When is the earliest date you can interview? (Format as MM/DD/YYYY)

The responses from the candidate will then be stored in a SQL Lite database. The steps to create the ChatBot are (tutorial code from:

- https://www.twilio.com/blog/build-custom-ai-chatbot-whatsapp-python-twilio-chatgpt-api
- https://docs.sqlalchemy.org/en/20/core/selectable.html#sqlalchemy.sql.expression.Select

    - Create the SQL Lite database storing the candidate's email/phone, work preference (remote, hybrid, in-office), reference name, reference contact #, and interview date.

```
# Create a ChatBot to conduct the HR screening interview and collect the interviewer's email, phone, work location prefe

# Tutorial code from https://www.twilio.com/blog/build-custom-ai-chatbot-whatsapp-python-twilio-chatgpt-api


from sqlalchemy import create_engine, Column, Integer, String
from sqlalchemy.orm import sessionmaker
from sqlalchemy.ext.declarative import declarative_base

# Set the database connection URL
```

- Use the built-in Python function 'input' to ask the questions to the candidate and collect the responses.

```
# Create the table
Base.metadata.create_all(engine)

# Ask the interview candidate for their email, phone number, reference contact info, and earliest start date
email = input("What is your email address?")
phone = input("What is your phone number?")
work_preference = input("Would you like to work remotely, hybrid, or in-office? ")
reference_name = input("Please enter the first and last name of your reference: ")
reference_contact = input("Please enter the phone number for your reference: ")
interview_date = input("When is the earliest date you can interview? (Format as MM/DD/YYYY) ")
```

- Create the SQL Alchemy table to store the candidate's responses.

```
# Define metadata for the table
metadata = MetaData()
candidate_responses = Table('answers_candidates', metadata,
    Column('email', String),
    Column('phone', String),
    Column('work_preference', String),
    Column('reference_name', String),
    Column('reference_contact', String),
    Column('interview_date', String),
)
```

- Query the results from the database.

```
# Connect to the database and execute a SELECT statement
with engine.connect() as conn:
    select_statement = candidate_responses.select()
    rows = conn.execute(select_statement).fetchall()

# Print the results
for row in rows:
    print(row)
```

```
('dhalsim1012@yahoo.com', '972-564-6450', 'hybrid', 'Sam Goody', '972-650-9021', '04/30/2023')
```

- Now that we have collected the responses from the candidates, we can schedule final interviews. We will use the iCalendar Python library to create the calendar invite based

on the date that the candidate selected. We will set the interview time from 9-11 am CST (tutorial code from: https://learnpython.com/blog/working-with-icalendar-with-python/

- The code will import the icalendar, datetime, pathlib, os, and pytz libraries and initialize the calendar and set the time zone as Central Time.

```
# imports
import icalendar
from icalendar import Calendar, Event, vCalAddress, vText
from datetime import datetime
from pathlib import Path
import os
import pytz

# init the calendar
cal = Calendar()

# Some properties are required to be compliant
cal.add('prodid', '-//My calendar product//example.com//')
cal.add('version', '2.0')

# Get the Central Time Zone object
central_tz = pytz.timezone('US/Central')
```

- The code will also set the interview time from 9-11 AM Central Time and use the interview date the candidate selected and retrieve the date from the SQL Alchemy database. The code will add an event name and description to the calendar invite as well as an event organizer. The code will then output a calendar invite to send to the interview candidate.

```
# Set the start and end times in Central Time Zone
start_time = datetime.strptime(interview_date + ' 9:00 AM', '%m/%d/%Y %I:%M %p')
end_time = datetime.strptime(interview_date + ' 11:00 AM', '%m/%d/%Y %I:%M %p')

# Convert the times to UTC
start_time_utc = central_tz.localize(start_time).astimezone(pytz.utc)
end_time_utc = central_tz.localize(end_time).astimezone(pytz.utc)

# Create the event object
event = Event()
event.add('name', 'Interview for Data Science Position at VibranTech')
event.add('description', 'Final Interview with Hiring Manager and HR')
event.add('dtstart', start_time_utc)
event.add('dtend', end_time_utc)
```

```
# Write to disk
directory = Path.cwd() / 'MyCalendar'
try:
   directory.mkdir(parents=True, exist_ok=False)
except FileExistsError:
   print("Folder already exists")
else:
   print("Folder was created")

f = open(os.path.join(directory, 'interview.ics'), 'wb')
f.write(cal.to_ical())
f.close()

e = open('MyCalendar/interview.ics', 'rb')
ecal = icalendar.Calendar.from_ical(e.read())
for component in ecal.walk():
   print(component.name)
e.close()
```

```
Folder already exists
VCALENDAR
VEVENT
```

Project Milestones

- Developed the job requisition and used Python to parse out the relevant key words from the job description.
- Performed Exploratory Data Analysis (EDA) on the job description and resumes to examine the most frequent key words in the job description and resume dataset.
- Looped through the resume dataset to extract the key words from each resume that matched up with the key words from the job description.
- Developed the Python functions to calculate the ratio of the key words from the resumes matching with the job description by the total key words in the job description.
- Created ChatBot to conduct the screening interview collecting the candidate's phone #, candidate's email, work preference, reference name and contact info, and earliest date candidate can interview.
- Create SQL Lite database to store the candidate responses to the answers asked by the ChatBot as part of the screening interview.
- Created a calendar invite using the iCalendar Python library based on the interview date provided by the job candidate.

Result Evaluation

- To identify the top 5 resumes matching with the job description based on key word extraction, a ratio was used to calculate the key words from each resume divided by the total key words from the job description. After running the code, the results showed 5 resumes each with a 40% score. This metric was used to observe the similarity of the key words in the job description to the key words found in the resumes. By matching only relevant key words, we can identify the candidates with matching key words and schedule the pre-screening interviews. This alleviates HR resources who can focus on conducting the final interview and not spend resources collecting pre-screening responses.

```
relevant_skills = [ machine learning ,  models ,  predict ,  recommend ,  collabora
skill_counts = []

for idx, text in enumerate(resume_texts):
    doc = nlp(text)
    skills = list(set([ent.text.lower() for ent in doc.ents if ent.label_ == "SKILL
    count = len(skills)
    ratio = count / len(relevant_skills)
    skill_counts.append((idx, ratio))
    resume_skills.append(skills)


sorted_list = sorted(skill_counts, key=lambda x: x[1], reverse=True)
top_5_resumes = sorted_list[:5]
for idx, ratio in top_5_resumes:
    print(f"Resume {idx+1} has a skill count ratio of {ratio:.2f}")
```

```
Resume 4 has a skill count ratio of 0.40
Resume 14 has a skill count ratio of 0.40
Resume 24 has a skill count ratio of 0.40
Resume 34 has a skill count ratio of 0.40
Resume 512 has a skill count ratio of 0.40
```

- Once the 5 candidates were identified for the interviews, the ChatBot can be deployed to conduct the screening interview and collect the answers to the pre-screening questions. Based on the responses provided by the candidate, the ChatBot can then create a calendar invite based on the date preference of the candidate. By utilizing a ATS/ChatBot streamlined approach, HR and the hiring manager can spend more time getting to know the final candidates and gather insightful data to make a hiring decision.

```
f = open(os.path.join(directory, 'interview.ics'), 'wb')
f.write(cal.to_ical())
f.close()

e = open('MyCalendar/interview.ics', 'rb')
ecal = icalendar.Calendar.from_ical(e.read())
for component in ecal.walk():
   print(component.name)
e.close()
```

```
Folder was created
VCALENDAR
VEVENT
```

Reference Material

- Resume dataset from Kaggle
  https://www.kaggle.com/datasets/gauravduttakiit/resume-dataset

- Resume Screening Stats from ideal.com

  https://ideal.com/resume-screening/#:~:text=In%20a%20nutshell%2C%20it's%20a,interview%20%E2%80%93%20or%20to%20reject%20them.

- How to Build a Resume Recommender like ATS

  https://towardsdatascience.com/resume-screening-tool-resume-recommendation-engine-in-a-nutshell-53fcf6e6559b

- Parsing Job Profiles Through NLP

  https://abhishekchhibber.com/2017/07/05/parsing-job-profiles-through-nlp/

- Resume Screening with Natural Language Processing in Python

  https://www.analyticsvidhya.com/blog/2021/06/resume-screening-with-natural-language-processing-in-python/

- Resume Screener Kaggle Code (LAKPA SHERPA)

  https://www.kaggle.com/code/sherpalakpa18/resume-screener

- Resume NLP (Aurelien KCZ)

  https://www.kaggle.com/code/aurelienkcz/resume-nlp

- Altair Documentation

  https://altair-viz.github.io/

- Matplotlib Documentation

  https://matplotlib.org/stable/tutorials/introductory/pyplot.html#sphx-glr-tutorials-introductory-pyplot-py

- Automatic Skill Extraction from Resumes using spaCy

  https://medium.com/mlearning-ai/automatic-skill-extraction-from-resumes-using-spacy-710507624a1e

- How to Build a Customized AI Chatbot on WhatsApp with Python, Twilio, and the ChatGPT API

  https://www.twilio.com/blog/build-custom-ai-chatbot-whatsapp-python-twilio-chatgpt-api

- SQLAlchemy 2.0 Documentation

  https://docs.sqlalchemy.org/en/20/core/selectable.html#sqlalchemy.sql.expression.Select

- Working with iCalendar in Python

  https://learnpython.com/blog/working-with-icalendar-with-python/