

Data Mining (CSCI 5502-872)

PROJECT REPORT

On

***“Resume Parsing and Analysis”***

*Authors*

*Jayan Agarwal**| Tuan Nguyen* ***|*** *Shivani Madan*

**(GROUP 20)**

1. **Abstract**

The primary goal of this project is to evaluate CVs and assist candidates in refining their resumes to better prepare for entering the job market. Based on personal experience, the research team recognizes that independently researching the labor market, identifying required skills, and adequately preparing can be a long and labor-intensive process. This project aims to leverage data to develop models for suggestions, classification, and predictions, supporting candidates in understanding market demands, assessing their own CVs, and receiving recommendations for self-improvement to meet labor market requirements.

This report will cover data collection through scraping, text processing, data cleaning, and data visualization.

1. **Data Collection**

To prepare the data for this project, several steps need to be completed, including:

* Evaluating and selecting data sources.
* Building scraping scripts.
* Storing the data.

Let us discuss the above in detail below:

1. Data Sources

Given the high demand for CV refinement and improvement, there are many platforms available that help users create CVs. However, these platforms mostly offer sample resumes. To gather data for the project, our team focused on searching for and scraping real user-generated CV data. Job-related websites were the primary targets for data extraction.

Several data sources were considered, each with its own characteristics:

* [LinkedIn](http://www.linkedin.com):
  + Advantages: A frequently updated source with a large volume of user-generated data, closely related to current job roles and market demand.
  + Disadvantages: Complex structure, can only scrape public profiles.
* [LiveCareer](http://www.livecareer.com):
  + Advantages: Rich data source, CVs are structured and can be scraped in sections.
  + Disadvantages: Complex website structure, data is updated via JavaScript, requiring more advanced scraping methods like Selenium.
* [PostFreeJobs](http://www.postjobfree.com):
  + Advantages: Simple web structure, easily scraped using HTML content through BeautifulSoup, large volume of user data, and up-to-date resumes.
  + Disadvantages: Resumes are not divided into sections, only full content can be retrieved.

With these pros and cons, we decide to select PostFreeJobs resource for our project. The scripts for scraping can be shown in the next section.

1. Building scraping scripts.
2. Understand the website’s structure.

PostJobFree has a simple structure and allows users to search based on job titles or resume titles. There are several search attributes available to refine results, including location, distance, title keywords, content keywords, and the number of results displayed per page.

A screenshot of a search box

Description automatically generated

A search engine window with text

Description automatically generated

The search results are displayed with the following content: Title, Location, Date Posted, and Content. We can optimize to display up to 100 resume per page.

1. Understand the url and HTML elements.

A screenshot of a computer

Description automatically generated

* Sample of URL:

https://www.postjobfree.com/resumes?q=data+scientist&n=cybersecurity&t=senior&d=professional&l=New+York+City&radius=500&r=100

* URL structure:
  + https://www.postjobfree.com/resumes: main site
  + ?q=data+scientist: main keyword
  + &n=cybersecurity: without words
  + &t=senior: search in title
  + &d=professional: search in resume content
  + &l=New+York+City: fitler by location
  + &radius=500: filter by distance
  + &r=100: number of results per page
* HTML structure (searching page):

A screenshot of a computer

Description automatically generated

* + <h3 class="itemTitle">: Resume title contain <a href> tag with the link to resume’s content page.
  + <span class="colorLocation">: Location.
  + <span class="colorDate">: Uploaded date.
  + <div class="normalText">: Excerpt from resume content.
* HTML structure (resume page):

A screenshot of a computer

Description automatically generated

* + <div class="normalText"><p>: The resume contents are located in several p tag inside class “normalText”.

1. Write the code.

import requests

from bs4 import BeautifulSoup

import pandas as pd

import time, os

class PJF():

def \_\_init\_\_(self):

self.search\_url = []

self.cv\_urls = []

self.keywords = ''

def load\_cv\_data(self, path:str):

# Check if the file exists

try:

self.cv\_data = pd.read\_csv(path, sep=';')

except FileNotFoundError:

self.cv\_data = pd.DataFrame(columns=['Keyword', 'ID', 'Link', 'Title', 'Location', 'PostedDate'])

def search(self, allwords:str='', nowords:str='', title:str='', textwords:str='', location:str='', radius:int=500, r:int=100):

for p in range(1, 6):

\_url = "https://www.postjobfree.com/resumes?q="+allwords.replace(' ', '+')+"&n="+nowords.replace(' ', '+')+"&t="+title.replace(' ', '+')+"&d="+textwords.replace(' ', '+')+"&l="+location.replace(' ', '+')+"&radius="+str(radius)+"&r="+str(r)+"&p="+str(p)

self.search\_url.append(\_url)

for url in self.search\_url:

try:

self.get\_resume\_links(url)

except Exception as e:

print("Error in scraping", url)

time.sleep(0.5)

print("Total number of resume:", len(self.cv\_urls))

def scraping(self):

# Initialize the dataframe or load the existing one

self.load\_cv\_data('cv\_data.csv')

for i, link in enumerate(self.cv\_urls):

print(f"{self.keywords}: Scraping link {i+1} / {len(self.cv\_urls)}")

if len(self.cv\_data) > 0:

unique\_links = self.cv\_data['Link'].unique()

# Loop through the links

if link in unique\_links:

print(f"Link {i+1} already scraped")

continue

else:

self.get\_resume\_content(link)

else:

self.get\_resume\_content(link)

def search\_and\_scraping(self, allwords:str='', nowords:str='', title:str='', textwords:str='', location:str='', radius:int=500, r:int=100):

self.keywords = title

self.search(allwords, nowords, title, textwords, location, radius, r)

self.scraping()

def get\_resume\_links(self, search\_url:str):

page = requests.get(search\_url)

soup = BeautifulSoup(page.content, 'html.parser')

link\_elements = soup.find\_all('h3', class\_='itemTitle')

for link in link\_elements:

\_combined\_link = "https://www.postjobfree.com" + link.a['href']

if \_combined\_link not in self.cv\_urls:

self.cv\_urls.append(\_combined\_link)

else:

print("Duplicate link found", link.a['href'])

def save\_to\_new\_txt(self, text:str, filename:str):

# Define the directory path

title = self.keywords.replace(' ', '\_').lower()

directory = f'.../postjobfree/{title}'

# Ensure the directory exists

if not os.path.exists(directory):

os.makedirs(directory)

# Create the file and write text

file\_path = os.path.join(directory, f"{filename}.txt")

try:

with open(file\_path, 'w') as f:

f.write(text)

print(f"File saved successfully at {file\_path}")

except Exception as e:

print(f"Error saving the file: {e}")

def get\_resume\_content(self, resume\_link:str):

try:

page = requests.get(resume\_link)

soup = BeautifulSoup(page.content, 'html.parser')

\_title = soup.find('h1').text

\_location = soup.find('a', class\_='colorLocation').text

\_posted\_data = soup.find('span', class\_='colorDate').text

\_text = [element.text for element in soup.find\_all('div', class\_='normalText')]

\_text = '\n'.join(\_text)

\_text = \_text.replace(';', ',')

# Save \_text to a text file

self.save\_to\_new\_txt(\_text, resume\_link[35:].replace("/", "\_"))

# Fix: wrap scalar values in lists to create a single-row DataFrame

\_df = pd.DataFrame({

'Keyword': [self.keywords],

'ID': [resume\_link[35:].replace('/', '\_')],

'Link': [resume\_link],

'Title': [\_title],

'Location': [\_location],

'PostedDate': [\_posted\_data],

})

# Concatenate with the existing cv\_data

self.cv\_data = pd.concat([self.cv\_data, \_df], ignore\_index=True)

# Save the dataframe to a csv file

self.cv\_data.to\_csv('cv\_data.csv', index=False, sep=';')

print(f"Scraped link:", resume\_link[27:])

except Exception as e:

print(e)

print(f"Error in scraping {resume\_link}")

time.sleep(0.2)

ResumeCraw = PJF()

keywords = ['Software Engineer', 'Frontend Developer', 'Backend Developer', 'Full Stack Developer', 'Mobile Developer', 'DevOps Engineer', 'Embedded Systems Engineer', 'Data Scientist', 'Data Engineer', 'Data Analyst', 'Machine Learning Engineer', 'AI Research Scientist', 'Product Manager', 'Technical Program Manager', 'UX/UI Designer', 'Interaction Designer', 'Cloud Engineer', 'Site Reliability Engineer (SRE)', 'Infrastructure Engineer', 'Network Engineer', 'Cybersecurity Engineer', 'Security Analyst', 'Penetration Tester', 'Information Security Manager', 'QA Engineer', 'Test Automation Engineer', 'Performance Engineer', 'Research Scientist', 'Algorithm Engineer', 'IT Support Specialist', 'Systems Administrator']

for i, keyword in enumerate(keywords):

print(f"Scraping keyword {i+1} / {len(keywords)}")

ResumeCraw.search\_and\_scraping(title=keyword)

* Link to scraping code: [Deepnote Project](https://deepnote.com/workspace/autunds-36923398-a2ef-4978-aeef-9b471dda4f96/project/5502-Data-Mining-Final-Project-85305fdb-eead-4cdc-ada4-f24147896889/notebook/Scraping-b33c2543d8ae49febe6c79b57c100830)
* [Github Notebook](https://github.com/jayanagarwal/Data-Mining-Project)

1. Storing the data.

From scraping process, we collected 12,474 resumes with several titles in IT industry, it covers from junior, senior to manager level.

All data are saved as text files and a data tabular format (\*.csv, \*.parquet)

Link to data: ([Github](https://github.com/jayanagarwal/Data-Mining-Project))

1. **Data preparation and cleaning**

Currently the data we have is in raw structure, as it is a collection of txt files which include alphabets, digits and special characters in it. As for this project, the process of cleaning requires usage of NLP (Natural Language Processing) based modelling which will be part of the next Milestone. So currently we have collected all the txt files that we had into a csv file (explained below using images):

A screenshot of a computer

Description automatically generated

1. We have 49 similar folders which represent job role

A screenshot of a computer

Description automatically generated

1. This represents the txt format data inside each folder

A screen shot of a computer

Description automatically generated

1. This is the txt format in the file

A screenshot of a computer

Description automatically generated

1. We have compiled all the resumes in a single file and processed some pattern search on it

As we could gain some insights using the regex pattern search on the txt data to get a brief understanding of the data. We have also performed cleanup of data using this.

We have performed the following cleanup:

1. We could see some files had ‘0’ word count, hence were empty so we removed them.
2. We identified that a few files had different language (4 files) we removed them as well as we are currently focusing only on the English language.
3. We have got a brief idea on the special characters count (the files which have 50% word count are mostly corrupted data and few files do exist) so we have been checking on them. Not removed yet as we will first test the model on them and then remove if required.
4. We have identified a few resume begin with the word continued (which might have been a user error) so we removed that word from it.
5. There was a common text in some resume (“Candidate has been called”) this might have been a part of the Resume feedback. But as this doesn’t give us any knowledge we have removed them from our data.
6. We expect to gain more insights after training NLP Model on it.

We have started with the next steps on creating an NLP Model, below are some of the things we could implement and identify.

**Next Steps of Cleaning (Milestone 3 Model based)**

The text in the raw resume contains special characters, spelling errors, or links. Therefore, to prepare for text data extraction, cleaning and processing the text data is extremely important. Below are some code snippets for processing and cleaning the data.

# Function to clean the text

def clean\_text(text):

# remove URLs

text = re.sub('http\S+\s\*', ' ', text)

# remove RT and cc

text = re.sub('RT|cc', ' ', text)

# remove hashtags

text = re.sub('#\S+', '', text)

# remove mentions

text = re.sub('@\S+', ' ', text)

# remove punctuations

text = re.sub('[%s]' % re.escape("""!"#$%&'()\*+,-./:;<=>?@[\]^\_`{|}~"""), ' ', text)

text = re.sub(r'[^\x00-\x7f]',r' ', text)

# remove extra whitespace

text = re.sub('\s+', ' ', text)

return text

# Function to tokenize the text

def tokenize\_text(text):

# Tokenize the text into words

tokens = nltk.word\_tokenize(text)

return tokens

# Function to remove stopwords

def remove\_stopwords(tokens):

stop\_words = set(stopwords.words('english')) # Use English stopwords

filtered\_tokens = [word for word in tokens if word not in stop\_words]

return filtered\_tokens

def lemmatize\_tokens(tokens):

# Function to perform lemmatization

lemmatizer = WordNetLemmatizer()

lemmatized\_tokens = [lemmatizer.lemmatize(token) for token in tokens]

return lemmatized\_tokens

def preprocess\_resume(text):

# Step 1: Clean the text

text = clean\_text(text)

# Step 2: Tokenize the text

tokens = tokenize\_text(text)

# Step 3: Remove stopwords

tokens = remove\_stopwords(tokens)

# Step 4: Lemmatize tokens

tokens = lemmatize\_tokens(tokens)

return tokens

To work with NLP model, text should be tokenized to tokens. This is the process of breaking down text into smaller units called tokens. These tokens can be words, phrases, or symbols, depending on the specific task.

Tokenization helps in understanding and analyzing the text by converting it into a structured format that algorithms can easily process.

A screenshot of a computer program

Description automatically generated

***Raw text:***

'Devi Y\nDevOps/Cloud Engineer\nE-mail: ad9bik@r.postjobfree.com Phone no: 815-\*\*\*-\*\*\*\*\n\nPROFESSIONAL SUMMARY:\nOver 7 years of IT experience in the design, implementation, and support of automated Continuous Integration (CI) and Continuous Delivery (CD) DevOps tooling, leveraging open-source Linux tool chains to support software development teams with the testing/packaging [/build/release](https://file+.vscode-resource.vscode-cdn.net/build/release) cycles of their native app, mobile-web, web-tier stack, databases and API services to endpoints including bare-m'

***Cleaned text:***

'Devi Y DevOps Cloud Engineer E mail ad9bik Phone no 815 PROFESSIONAL SUMMARY Over 7 years of IT experience in the design implementation and support of automated Continuous Integration CI and Continuous Delivery CD DevOps tooling leveraging open source Linux tool chains to support software development teams with the testing packaging build release cycles of their native app mobile web web tier stack databases and API services to endpoints including bare metal local virtualized as well as cloud ba'

***Tokens:***

['Devi', 'Y', 'DevOps', 'Cloud', 'Engineer', 'E', 'mail', 'ad9bik', 'Phone', '815', 'PROFESSIONAL', 'SUMMARY', 'Over', '7', 'year', 'IT', 'experience', 'design', 'implementation', 'support', 'automated', 'Continuous', 'Integration', 'CI', 'Continuous', 'Delivery', 'CD', 'DevOps', 'tooling', 'leveraging', 'open', 'source', 'Linux', 'tool', 'chain', 'support', 'software', 'development', 'team', 'testing', 'packaging', 'build', 'release', 'cycle', 'native', 'app', 'mobile', 'web', 'web', 'tier', 'stack', 'database', 'API', 'service', 'endpoint', 'including', 'bare', 'metal', 'local', 'virtualized', 'well', 'cloud']

1. **Visualization**

After cleaning the data, the team created several visualizations. The distribution of resume lengths is represented below by counting the number of words in each resume.

A graph of numbers and a number of words

Description automatically generated

It can be observed that the distribution is left-skewed, with most resumes falling within the range of 500 to 600 words. The maximum length of a resume is nearly 5,000 words.

The classification based on keywords and the desired occupations of candidates was also analyzed. The pie chart below illustrates the proportion of the most commonly sought jobs present in the dataset.

A colorful circle with different colored numbers

Description automatically generated with medium confidence

Number of Resume per job title

A graph of a number of resumes

Description automatically generated

Average Word Count per Job Title A graph of average world count

Description automatically generated with medium confidence

Average Certification Count per Job Title

A graph of different colored bars

Description automatically generated

Breakdown of Skills, Education, and Experience Across Job TitlesA graph of green and blue hills

Description automatically generated with medium confidence

Skills by Education Levels

A graph showing different colored shapes

Description automatically generated

Bubble Chart: Word Count vs Experience vs Education

A graph of blue dots

Description automatically generated

Correlation Matrix between Numeric Variables: Word Count, Skills, Education, Experience and Certifications A screenshot of a computer screen

Description automatically generated

Pair Plot of Skills, Education, Experience, and Certifications

A collage of blue and white graphs

Description automatically generated

The data cleaning process in the previous step enabled a more effective analysis of word frequency. The most commonly occurring words in the resumes are listed in the chart below. It is evident that the keyword "data" is predominant, indicating that all candidates are interested in or have experience working with data.

A graph of different colored bars

Description automatically generated with medium confidence

Additionally, using the WordCloud library, the representation of keywords provides a more visually striking way to highlight the terms of interest.

A close-up of words

Description automatically generated