University of Colorado **Boulder**

Data Mining (CSCI 5502-872)

PROJECT REPORT

On

"Resume Parsing and Analysis"

Authors

Jayan Agarwal | *Tuan Nguyen* **|** *Shivani Madan*
(GROUP 20)

I. 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.

II. 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.

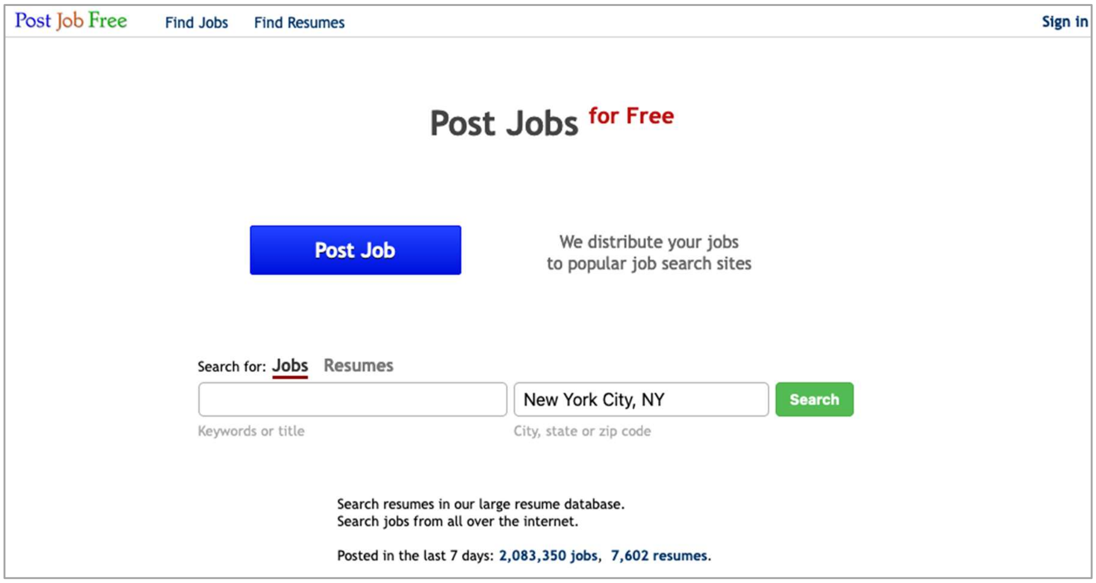
   Several data sources were considered, each with its own characteristics:

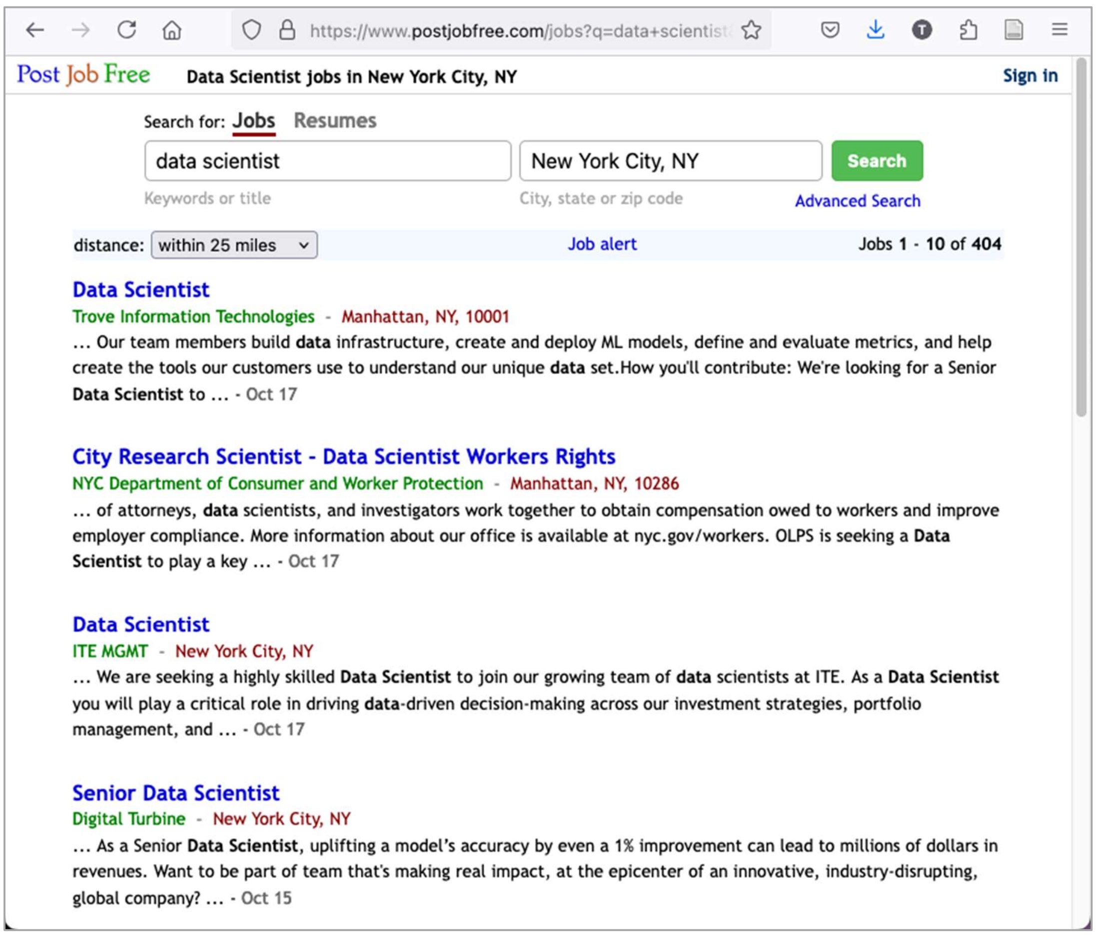
   - LinkedIn:
     - Advantages: A frequently updated source with a large volume of user-generated data, closely related to current job roles and market demand.
     - Disadvantages: Complex structure, can only scrape public profiles.
   - LiveCareer:
     - Advantages: Rich data source, CVs are structured and can be scraped in sections.
     - Disadvantages: Complex website structure, data is updated via JavaScript, requiring more advanced scraping methods like Selenium.
   - PostFreeJobs:
     - 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.

2. Building scraping scripts.
   a. Understand the website's structure.

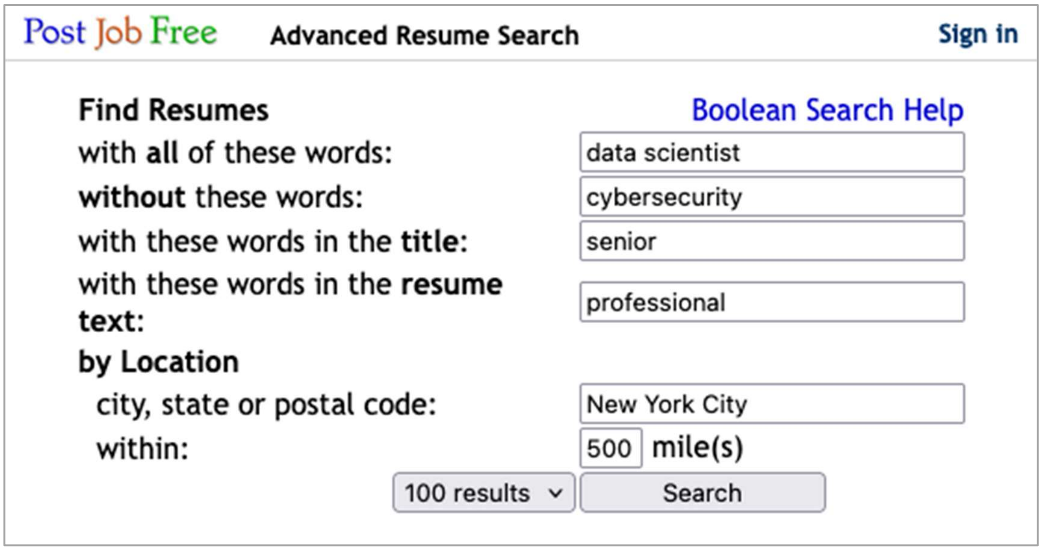
   PostJobFree has a simple structure and allows users to search based on job titles or resume titles. There are several search attributes available to refine results, including location, distance, title keywords, content keywords, and the number of results displayed per page.





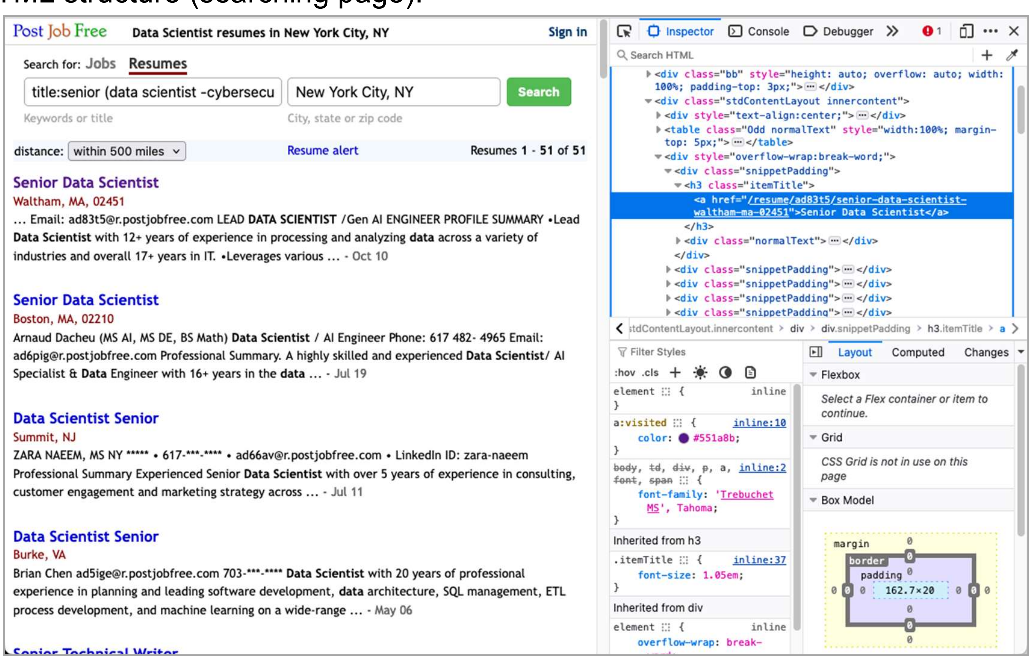
The search results are displayed with the following content: Title, Location, Date Posted, and Content. We can optimize to display up to 100 resume per page.

b. Understand the url and HTML elements.



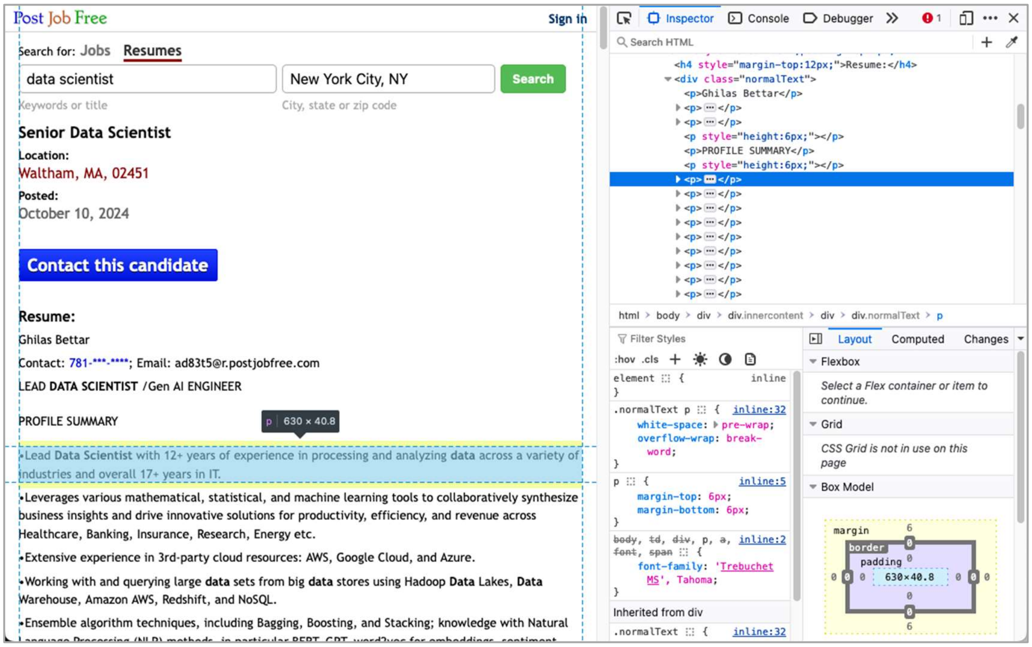
- Sample of URL: https://www.postjobfree.com/resumes?q=data+scientist&n=cybersecurity&t=senior&d=professional&l=New+York+City&radius=500&r=100

- 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: filter by location
    - &radius=500: filter by distance
    - &r=100: number of results per page
- HTML structure (searching page):



- <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):



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

c. 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
- Github Notebook

3. 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)

III. 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):

Name	Date modified	Type	Size
Earlier this week			
network_engineer	10/15/2024 5:59 PM	File folder	
test_automation_engineer	10/15/2024 9:02 AM	File folder	
technical_consultant	10/15/2024 9:01 AM	File folder	
ux	10/15/2024 9:01 AM	File folder	
technical_program_manager	10/15/2024 9:01 AM	File folder	
technical_writer	10/15/2024 9:01 AM	File folder	
site_reliability_engineer_(sre)	10/15/2024 9:01 AM	File folder	
software_engineer	10/15/2024 9:01 AM	File folder	
solutions_architect	10/15/2024 9:00 AM	File folder	
systems_administrator	10/15/2024 9:00 AM	File folder	

1. We have 49 similar folders which represent job role

Name	Date modified	Type	Size
Earlier this week			
ad85qs_data-engineer-senior-irving-tx-75060.txt	10/15/2024 8:51 AM	Text Document	30 KB
ad5wk6_data-engineer-lake-edison-nj.txt	10/15/2024 8:52 AM	Text Document	30 KB
ad8pd2_data-engineer-senior-plano-tx.txt	10/15/2024 8:51 AM	Text Document	30 KB
ad673k_data-center-mechanical-gaithersburg-md.txt	10/15/2024 8:51 AM	Text Document	30 KB
ad5z9n_data-analyst-engineer-burlington-ma.txt	10/15/2024 8:52 AM	Text Document	30 KB
ad7k1a_data-engineer-software-san-leandro-ca.txt	10/15/2024 8:51 AM	Text Document	30 KB
ad8ma4_data-engineer-quality-frederick-md.txt	10/15/2024 8:51 AM	Text Document	30 KB
ad7yd9_data-engineer-machine-lewisville-tx.txt	10/15/2024 8:51 AM	Text Document	30 KB

2. This represents the txt format data inside each folder

```
SENIOR DATA ENGINEER

Name: Pallavi Priya K +1-469-***-**** Email: ad85qs@r.postjobfree.com

PROFESSIONAL SUMMARY:

●Senior Data Engineer with 10 years' experience in designing, implementing, and optimizing end-to-end data engineering workflows to support data-driven decision-making and business growth using GCP Cloud Platform and BigQuery in the Banking, Retail, Healthcare, and E-commerce domains.
●Worked with Flask framework for designing REST API's using Python language Flask Framework.
●Experience in layers of Hadoop Framework - Storage (HDFS), Analysis (Pig and Hive), Engineering (Jobs and Workflows), extending the functionality by writing custom UDFs.
●Developed spark applications in python (PySpark) on distributed environment to load huge number of CSV files with different schema in to Hive ORC tables.
●Developed Python application for Google Analytics aggregation and reporting and used Django configuration to manage URLs and application parameters.
●Created several types of data visualizations using Python and Tableau, created modules for spark streaming in data into Data Lake using Spark. Conducted statistical analysis on healthcare data using python and various tools.
```

3. This is the txt format in the file

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	Resume File Name	Resume Folder (Job Tit	Word Count	Special Charact	Continued Co	Resume	Name	Contact	Profession	Skills	Education	Experience	Certificatic	Accomplis	Achieveme	Hobbies	Languages	Linkedin	Github
2	ac01ih_turbine-con	algorithm_engineer	1139	185	0	* * * * * C a r l	0	0	0	0	0	5	0	0	0	0	0	0	0
3	ace5so_semicondu	algorithm_engineer	934	277	0	Feng Ma	0	0	1	2	3	4	0	0	0	0	0	0	0
4	acec8w_physics-da	algorithm_engineer	472	112	0	Henry	0	0	0	3	1	1	0	0	0	0	1	1	0
5	ad80wr_computer-v	algorithm_engineer	334	84	0	EXPERIENCE	0	1	0	0	1	1	0	0	0	0	0	0	0
6	ac1j2o_selenium-js	backend_developer	301	136	0	Sheena Shah	0	0	0	1	1	0	0	0	0	0	0	0	1
7	ac1vo4_software-er	backend_developer	336	95	0	Bruce	0	0	0	2	1	3	0	0	0	0	1	0	0
8	ac2whl_app-ios-pro	backend_developer	708	276	0	Luca	0	0	0	1	0	0	0	0	0	0	2	0	0
9	ac33we_ruby-azure	backend_developer	888	306	0	Matt Lankford	0	1	0	1	0	2	0	0	0	0	0	0	0
10	ac82jg_azure-mvc-c	backend_developer	2705	751	0	Balaji	0	0	0	3	0	4	1	0	0	0	2	2	1
11	accros_developer-w	backend_developer	947	334	0	Aleksandr	1	0	0	6	1	4	0	0	0	0	2	0	0

4. 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.

```python
print("This is raw text: ", df['content'][0][:100])
print("\nThis is cleaned text: ", df['cleaned_text'][0][:100])
print("\nThis is tokens: ", df['tokens'][0][:100])
```

```
✓ 0.0s                                                                    Python
```

```
This is raw text:  Devi Y
DevOps/Cloud Engineer
E-mail: ad9bik@r.postjobfree.com Phone no: 815-***-****

PROFESSIONAL S

This is cleaned text:  Devi Y DevOps Cloud Engineer E mail ad9bik Phone no 815 PROFESSIONAL SUMMARY Over 7 years of IT expe

This is tokens:  ['Devi', 'Y', 'DevOps', 'Cloud', 'Engineer', 'E', 'mail', 'ad9bik', 'Phone', '815', 'PROFESSIONAL', 'SUMMARY', 'Over', '
```

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 cycles of their native app, mobile-web, web-tier stack, databases and API services to endpoints including bare-m'

Cleaned text:

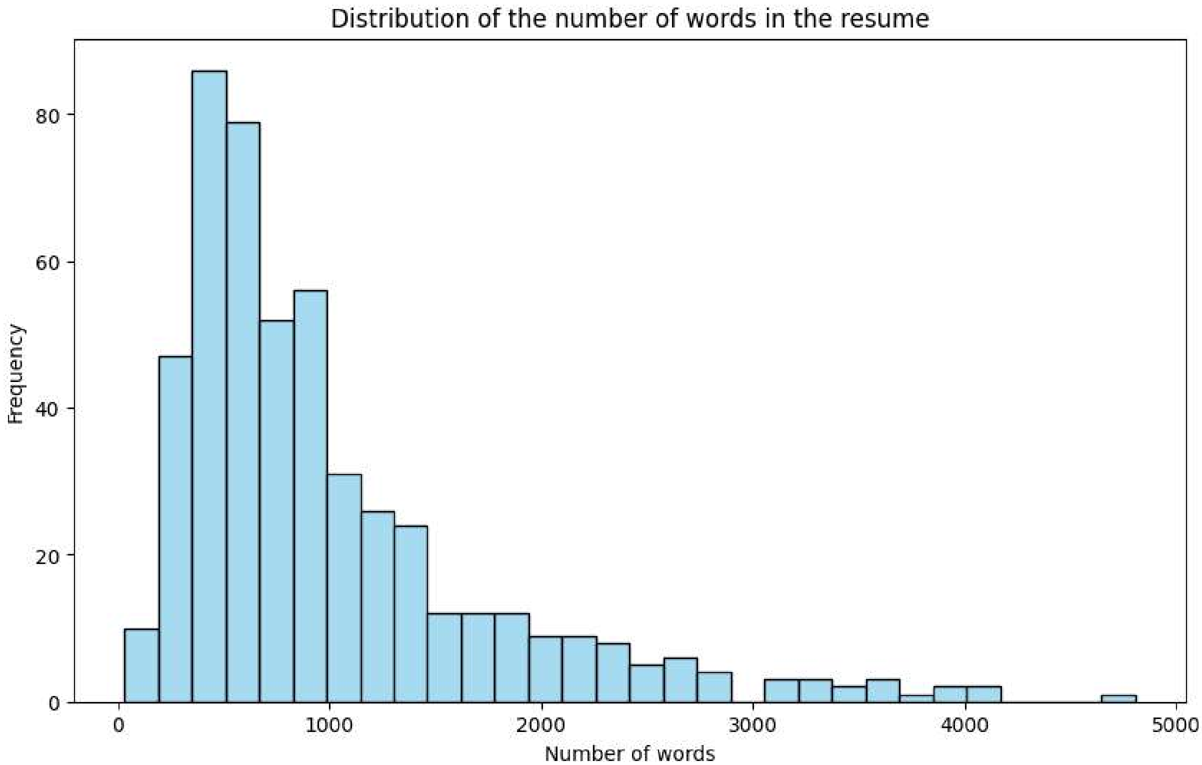
'Devi Y DevOps Cloud Engineer E mail ad9bik Phone no 815 PROFESSIONAL SUMMARY Over 7 years of IT experience in the design implementation and support of automated Continuous Integration CI and Continuous Delivery CD DevOps tooling leveraging open source Linux tool chains to support software development 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']

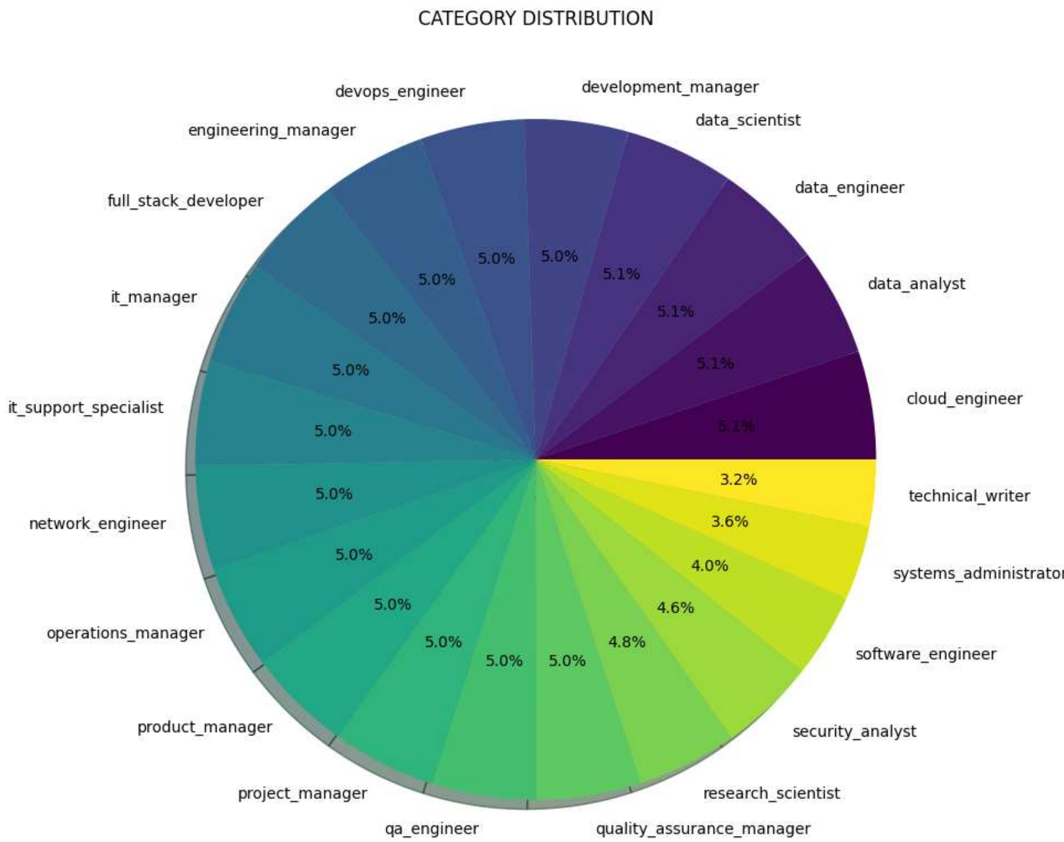
IV. Visualization

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.

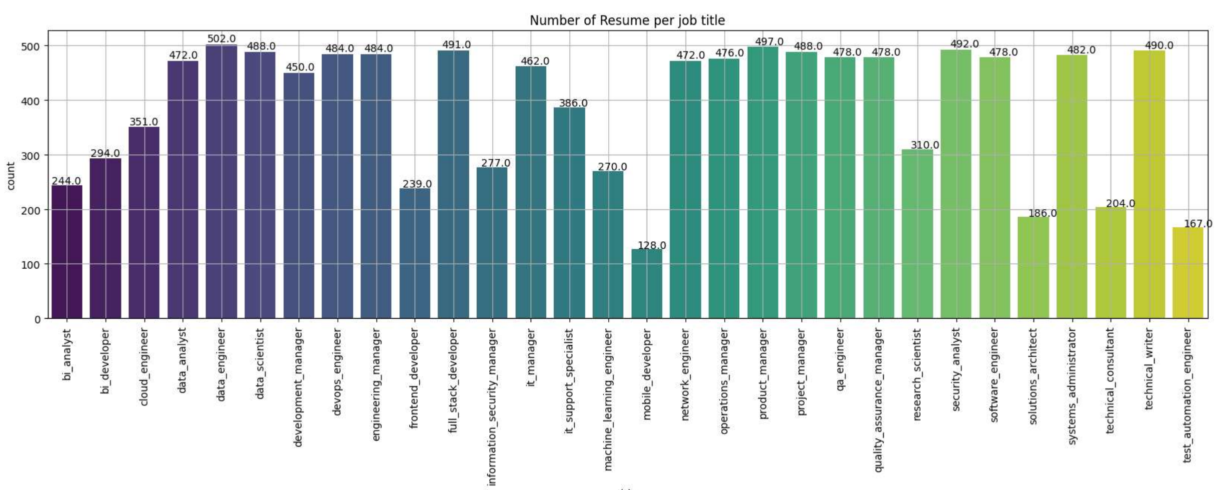


It can be observed that the distribution is left-skewed, with most resumes falling within the range of 500 to 600 words. The maximum length of a resume is nearly 5,000 words.

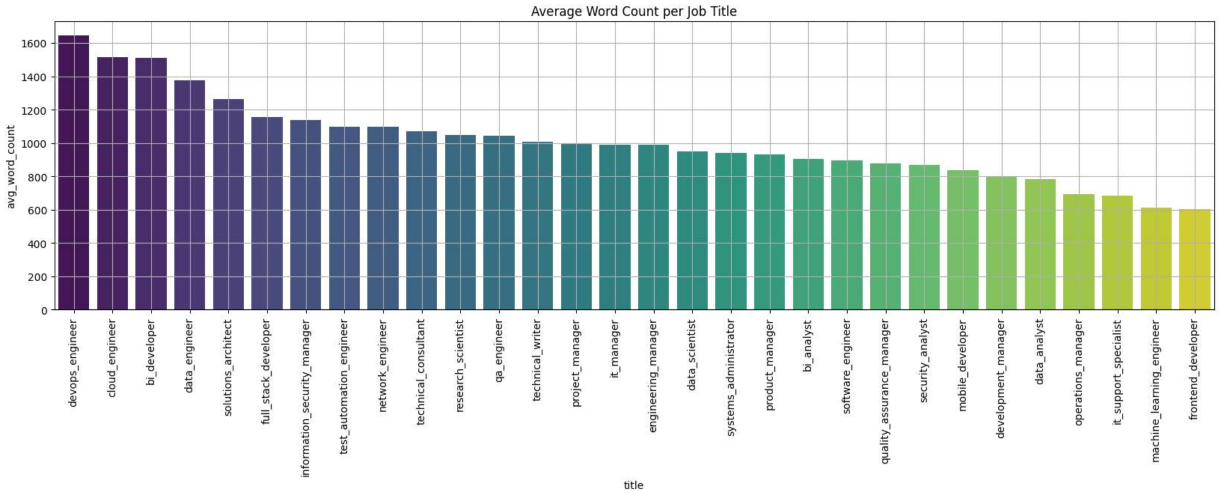
The classification based on keywords and the desired occupations of candidates was also analyzed. The pie chart below illustrates the proportion of the most commonly sought jobs present in the dataset.



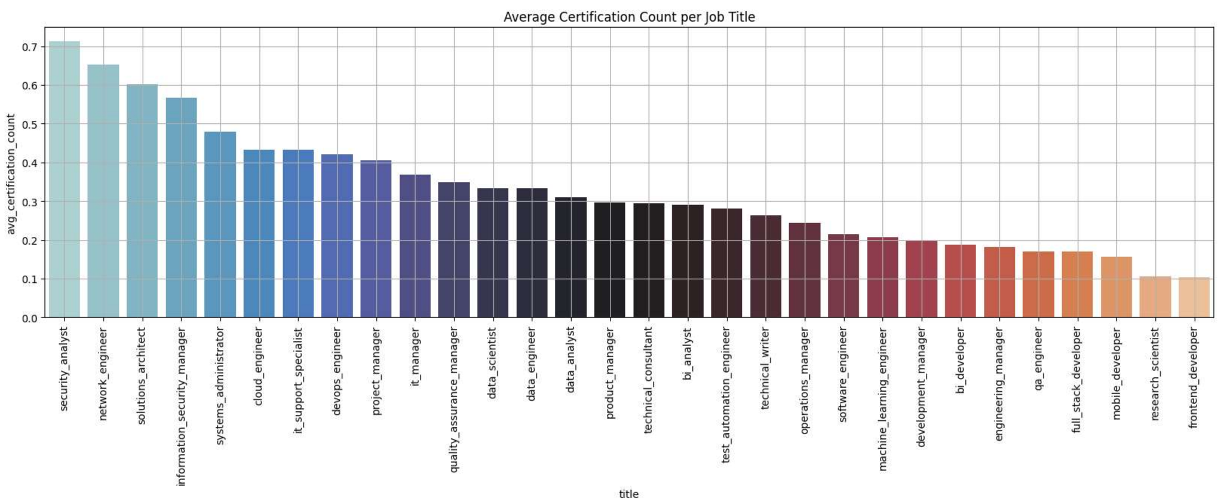
Number of Resume per job title



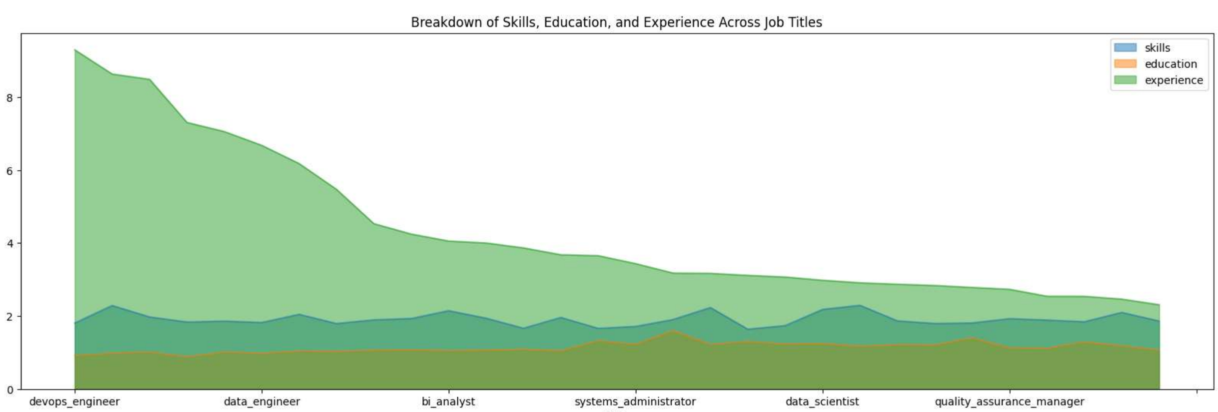
Average Word Count per Job Title



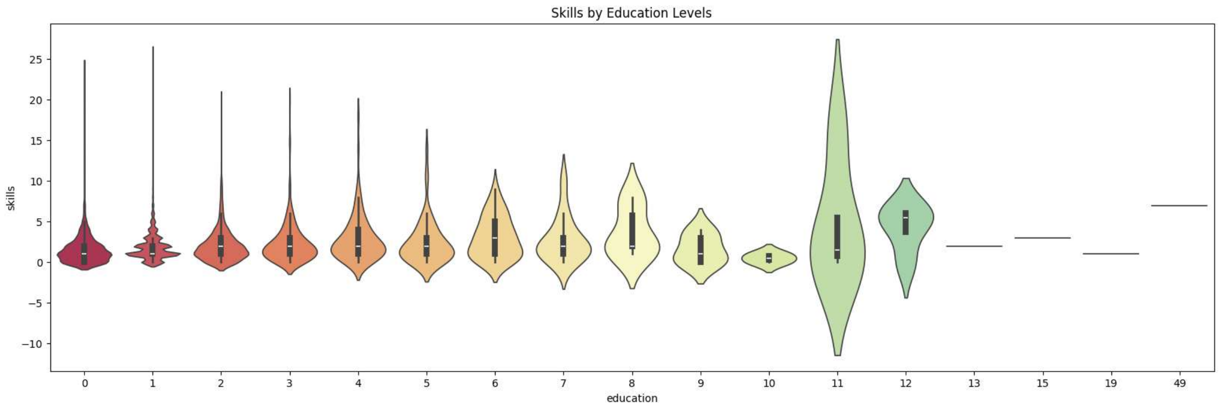
Average Certification Count per Job Title



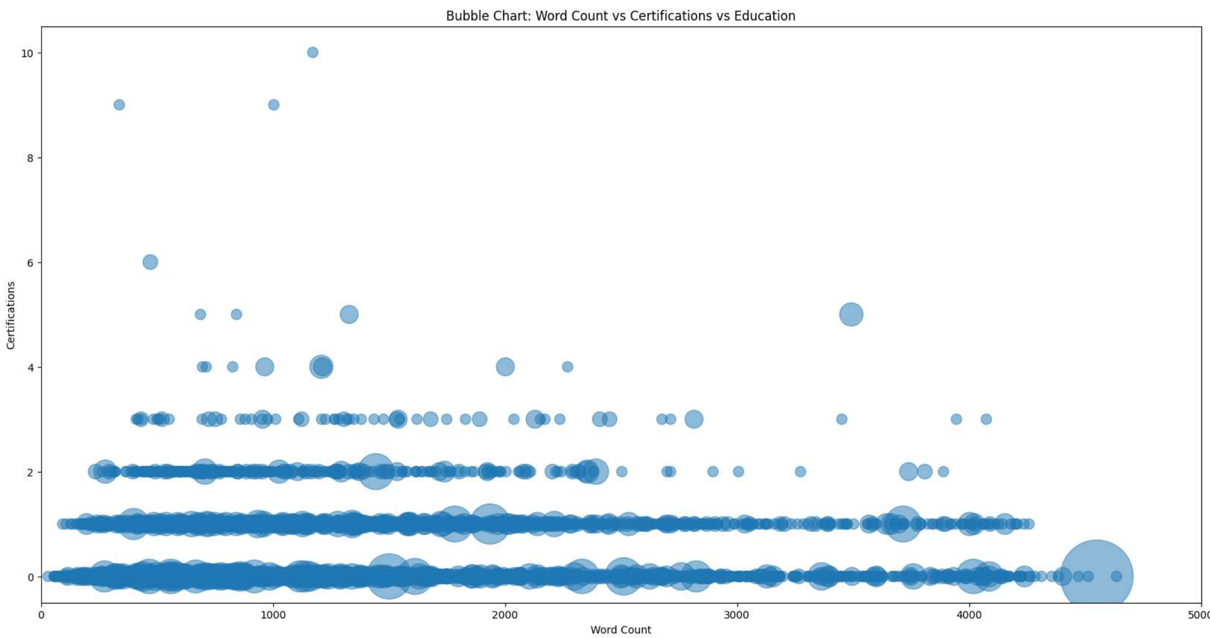
Breakdown of Skills, Education, and Experience Across Job Titles



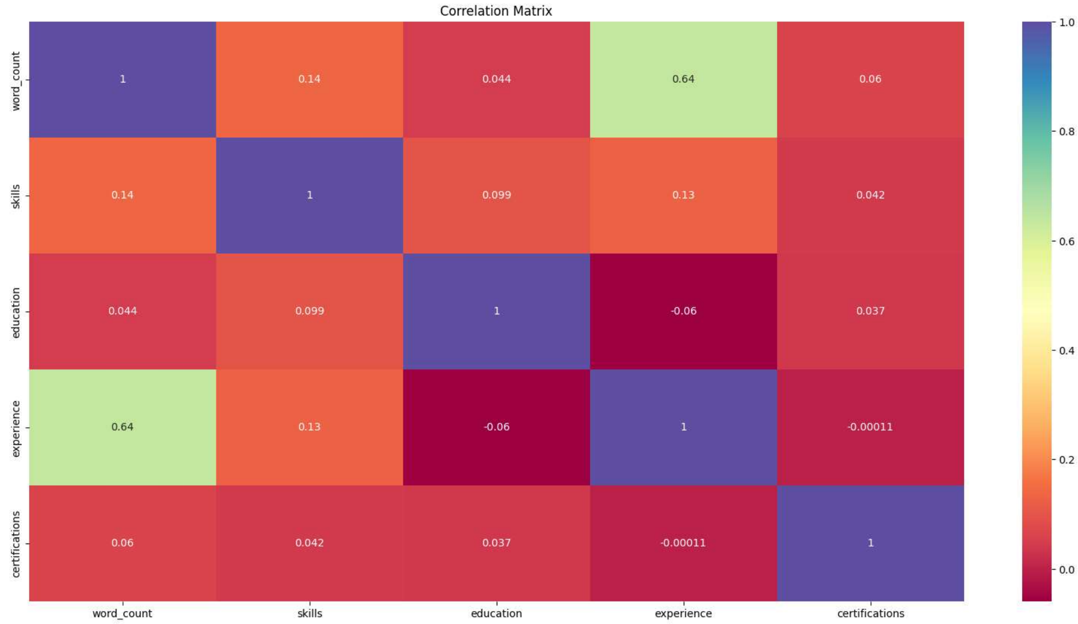
Skills by Education Levels



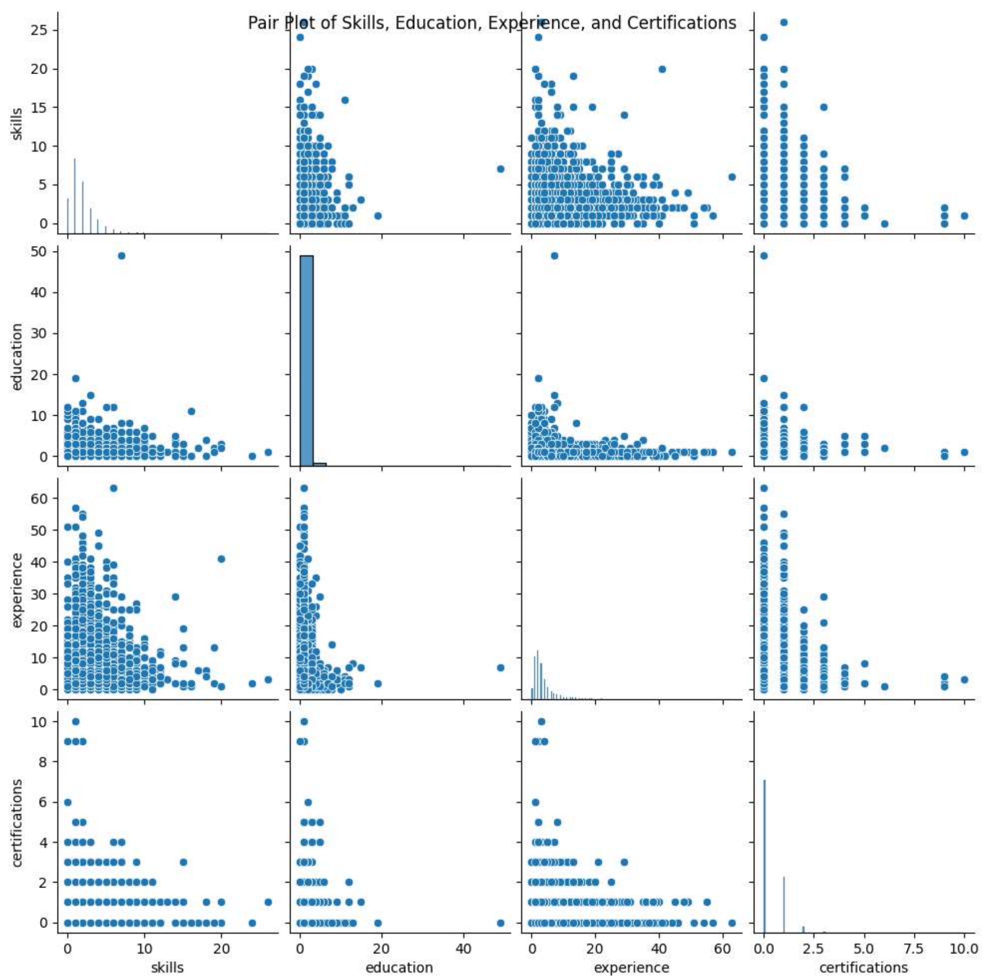
Bubble Chart: Word Count vs Experience vs Education



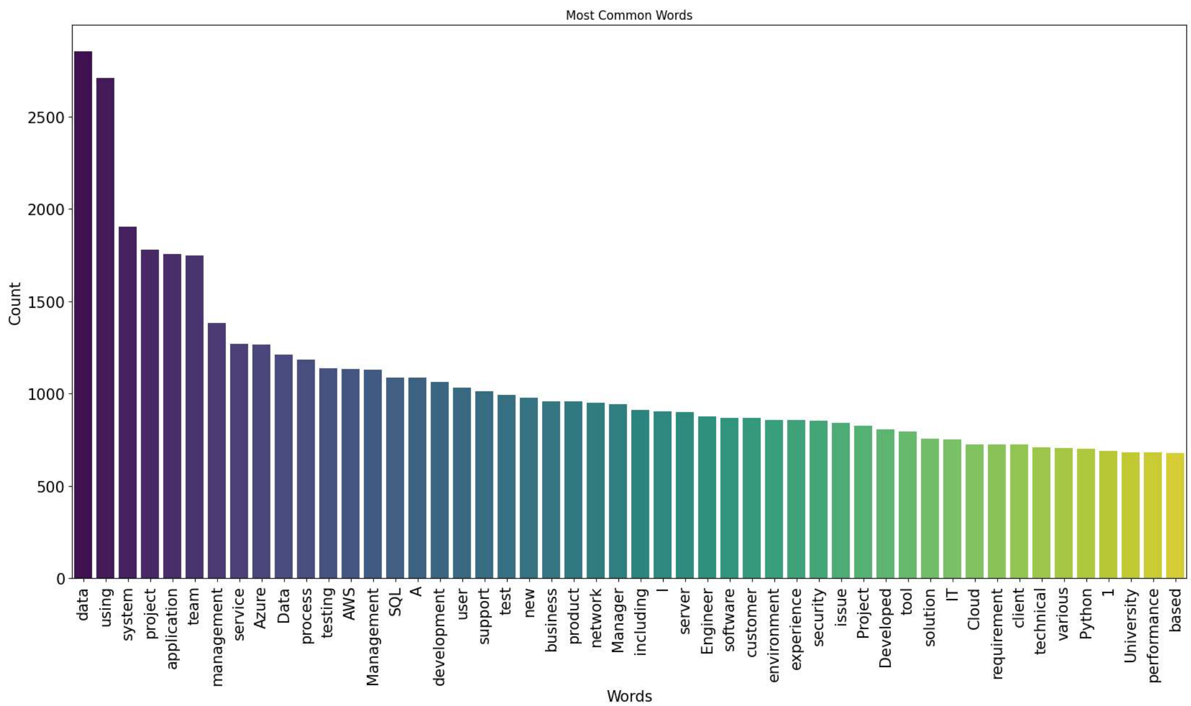
Correlation Matrix between Numeric Variables: Word Count, Skills, Education, Experience and Certifications



Pair Plot of Skills, Education, Experience, and Certifications



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.



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