



JOB RECOMMENDATION SYSTEM

CS6003 – BIG DATA ANALYTICS

TEAM NO. 8

Alaa Adnan C – 2021103505

Amritha R – 2021103506

Arulmurugan S – 2021103510

Giritharan K – 2021103526

PROBLEM BRIEFING:

JOB RECOMMENDATION SYSTEM USING CONTENT BASED FILTERING

Overview:

The job recommendation system utilizes content-based filtering to provide personalized job suggestions based on user skills and experience. It integrates with PostgreSQL for data storage and Flask for web interface development. Users can securely register and login to access tailored job recommendations. The system aims to streamline the job search process by offering relevant suggestions, enhancing user experience, and improving job matching accuracy.

Key questions to address:

1. **Problem Statement:** What challenges does the job recommendation system aim to solve? How does it address the need for personalized and relevant job recommendations?
2. **User Requirements:** What user inputs are required for generating personalized job recommendations? How does the system capture user skills, experience, and job preferences?
3. **Algorithmic Approach:** What algorithms and methodologies are employed to generate job recommendations? How does the system calculate similarity scores and tailor recommendations based on user profiles?
4. **Feature Description:** What are the main features of the job recommendation system? How does the system generate personalized job recommendations and offer functionality to users?
5. **Data Visualization:** What visualization techniques are used to display job details, company information, and industry trends?
6. **Technological Stack:** What technologies are utilized in the implementation of the system? How is the frontend developed using Flask, HTML, and CSS? Additionally, how are database interactions managed using PostgreSQL?
7. **User Interaction:** What is the user interaction flow within the system? How do users input their skills, experience, and job preferences? How are personalized job recommendations presented to users, and how can they provide feedback?
8. **Conclusion:** What are the key takeaways from the development and implementation of the job recommendation system? How does it contribute to assisting job seekers in finding relevant job opportunities and advancing their careers?

Challenges:

- **Data Quality:** Ensuring the quality and reliability of the job dataset, including accurate job descriptions and skill requirements.
- **Algorithm Accuracy:** Fine-tuning the content-based filtering algorithm to improve the accuracy of job recommendations and minimize false positives.
- **Scalability:** Addressing scalability concerns, especially as the user base grows and the dataset expands, to maintain system performance.
- **User Engagement:** Encouraging user engagement and interaction with the system to gather sufficient data for accurate recommendations and retain active users.
- **Integration Complexity:** Managing the integration of various technologies, including Flask for the backend, PostgreSQL for data storage, and scikit-learn for machine learning tasks, to ensure smooth system operation.

Scope:

The job recommendation system aims to provide personalized job suggestions based on user skills, experience, and job preferences. It employs content-based filtering techniques to match user input with relevant job postings from a dataset. The system includes features such as user authentication, database integration, and a user-friendly web interface. Its scope encompasses improving job matching accuracy, enhancing user experience, and streamlining the job search process.

INDIVIDUAL CONTRIBUTIONS:

1) Amritha R (Frontend, Backend Integration, Visualization):

- Developed the frontend of the job recommendation system, and ensured that it is user-friendly and intuitive.
- Integrated the frontend with the backend logic, including user input processing and recommendation generation.
- Implemented visualization components to present job recommendations and system performance metrics.
- Prepared comprehensive documentation covering system architecture, methodology, algorithms used, and user guides.

2) Arulmurugan S (TF-IDF and Cosine Similarity Calculation, Validation):

- Implemented TF-IDF vectorization for text data and cosine similarity calculation between user input and job descriptions.
- Collaborated with Alaa Adnan to ensure accurate calculation of combined scores for job recommendations.
- Validated the recommendation system's performance using appropriate evaluation metrics and testing methodologies.
- Worked with Amritha in documenting the algorithmic details and validation process for the recommendation system.

3) Giritharan K (Data Loading, Extraction, Preprocessing, Frontend):

- Loaded the job dataset into the project environment and ensure its accessibility to the team.
- Implemented data extraction procedures to retrieve relevant information from the dataset.
- Performed data preprocessing tasks such as cleaning, standardization, and feature engineering to prepare the dataset for analysis.
- Collaborated with Amritha to integrate the preprocessed data into the frontend for user interaction.

4) Alaa Adnan C (TF-IDF, Cosine Similarity and Combined Score Calculation):

- Collaborated with Arulmurugan to implement TF-IDF vectorization and cosine similarity calculation.
- Develop algorithms to calculate combined scores for job recommendations based on similarity metrics and other factors.
- Work closely with Amritha to visualize recommendation results and system performance metrics for user feedback.
- Assist in documenting the combined score calculation methodology and its contribution to recommendation accuracy.

1. INTRODUCTION

The job recommendation system is an innovative solution designed to simplify and enhance the job search experience for users. Leveraging advanced machine learning techniques and database integration, the system analyses user-provided information such as skills, job titles, and experience to generate personalized job recommendations. By utilizing content-based filtering algorithms, the system matches user input with relevant job postings from a comprehensive dataset, offering tailored suggestions to meet the unique preferences and requirements of each user. With features including

secure user authentication, seamless database storage, and an intuitive web interface, the system aims to optimize job matching accuracy, improve user engagement, and streamline the job search process. Through its user-centric design and robust functionality, the job recommendation system endeavours to empower users in their quest for meaningful employment opportunities.

Key features and objectives:

- **Personalized Recommendations:** The system provides personalized job suggestions based on individual user profiles and preferences, ensuring that each user receives relevant and meaningful recommendations.
- **Content-Based Filtering:** By analysing job descriptions, key skills, and user input, the system employs content-based filtering techniques to match users with job postings that closely align with their qualifications and interests.
- **User Authentication and Database Integration:** Secure user authentication and database integration allow users to create profiles, store preferences, and receive personalized recommendations seamlessly.
- **User-Friendly Interface:** The system features a user-friendly web interface that simplifies the job search process, making it easy for users to input their skills, job titles, and experience and receive personalized recommendations.

2. ARCHITECTURE/Framework

The job recommendation system follows a client-server architecture, with the backend implemented using the Flask web framework and the frontend designed using HTML/CSS. The system integrates with a PostgreSQL database for storing user information and job data.

Components:

- **Client-Side:** The frontend interface provides users with a user-friendly web interface for interacting with the system. It includes input forms for users to provide their skills, job titles, and experience, as well as displays personalized job recommendations.
- **Server-Side:** The Flask web framework serves as the backend of the system, handling user requests, processing data, and generating job recommendations. It includes routes for user authentication, job recommendation generation, and database interactions.

- **Database:** PostgreSQL is utilized as the relational database management system for storing user profiles, job postings, and recommendation data. It stores user information such as username, password, skills, experience, and job recommendations.

Workflow:

- **User Input:** Users provide their skills, job titles, and experience through the frontend interface.
- **Backend Processing:** The Flask backend receives user input, processes it using content-based filtering algorithms, and retrieves relevant job postings from the PostgreSQL database.
- **Recommendation Generation:** The system generates personalized job recommendations based on user input and similarity scores calculated using TF-IDF vectorization and cosine similarity.
- **Database Interaction:** User information, including profiles and recommendations, is stored and retrieved from the PostgreSQL database as needed.

Framework:

- **Flask:** Lightweight and flexible web framework for Python, providing tools and libraries for building web applications.
- **PostgreSQL:** Open-source relational database management system known for its reliability, scalability, and performance.
- **HTML/CSS:** Frontend languages used for designing the user interface and enhancing user experience.

3. METHODOLOGY

3.1 Data Preparation and Feature Engineering:

- **Data Collection:** Job-related data is collected from Kaggle.
- **Data Cleaning and Standardization:** The dataset undergoes cleaning processes to remove duplicates, missing values, and inconsistencies. Standardization ensures uniformity across different data sources.

- **Feature Extraction:** Relevant features such as job titles, descriptions, required skills, experience levels, and industry categories are extracted from the dataset. These features provide essential information for matching candidates with suitable job listings.
- **Feature Transformation:** Textual features are transformed into numerical representations using techniques like TF-IDF (Term Frequency-Inverse Document Frequency) or word embeddings. This transformation enables quantitative analysis and comparison of job listings and candidate profiles.

3.2 Algorithm Development and Evaluation:

- **Similarity Computation:** Similarity scores are computed between candidate profiles and job listings based on their textual features, such as job titles, descriptions, and required skills. Techniques like cosine similarity are commonly used for this purpose.
- **Algorithm Design:** Algorithms are devised to match candidates with suitable positions by considering factors such as experience levels, skill requirements, and job preferences. Content-based filtering approaches are often employed to generate personalized recommendations.
- **Personalized Recommendation Generation:** The system generates personalized job recommendations by combining textual similarity scores with experience matching scores and other relevant features. This ensures that the recommendations align closely with the candidate's qualifications and preferences.
- **Validation and Testing:** The system's performance is validated through user testing and comparison with benchmark datasets or existing job recommendation systems. User feedback and insights are collected to further refine and optimize the recommendation algorithms.

4 STEPS INVOLVED

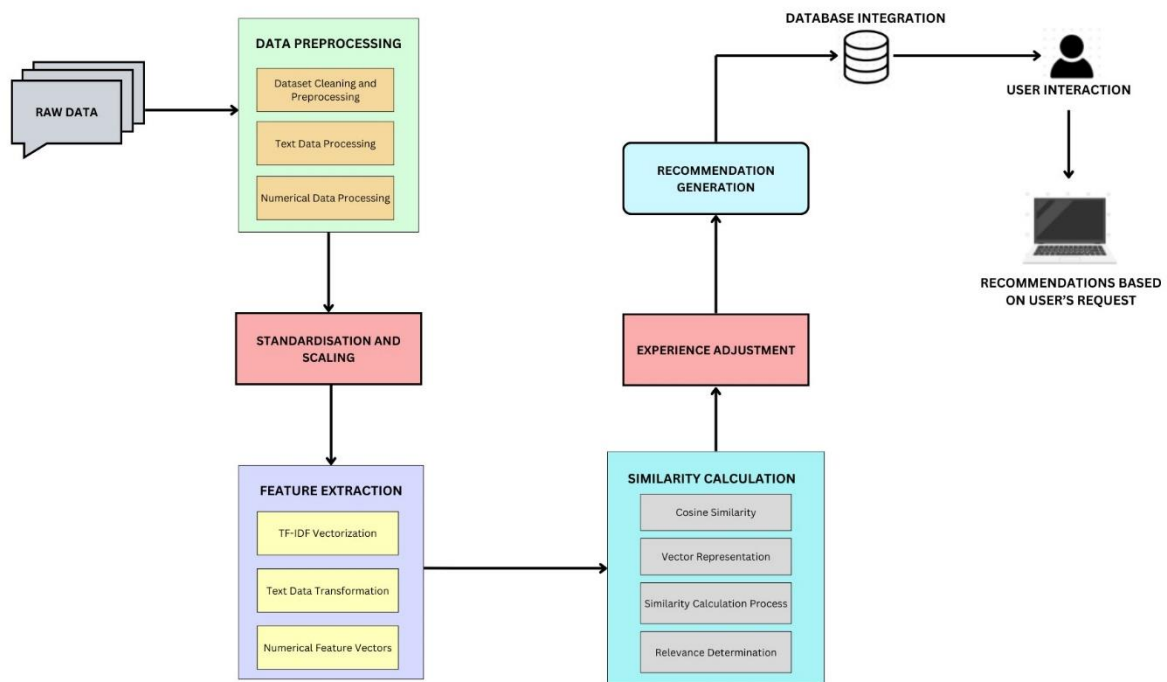


Figure 1: Architecture Diagram

4.1 Data Preprocessing:

▪ *Dataset Cleaning and Preparation:*

- The job dataset undergoes thorough cleaning and preparation to ensure its quality and suitability for analysis.
- This involves identifying and handling missing values, duplicates, inconsistencies, and other data anomalies.

▪ *Text Data Processing:*

- Text data, such as job titles, descriptions, and key skills, undergoes preprocessing to transform it into a format suitable for analysis.
- The following techniques are commonly applied:

Tokenization: Text is divided into smaller units called tokens, which can be individual words or phrases.

Normalization: Text is normalized to achieve consistency and uniformity. This may involve converting all text to lowercase, removing accents, or expanding contractions.

Stopwords Removal: Commonly occurring words with little semantic value (e.g., "the," "and," "is") are removed to reduce noise and improve analysis accuracy.

Punctuation Removal: Punctuation marks such as commas, periods, and exclamation marks are removed to focus on meaningful content.

▪ *Numerical Data Processing:*

- Numerical data, such as job salaries and years of experience, may require preprocessing to handle outliers, scale the data, or fill missing values.
- Techniques like normalization, standardization, or imputation are applied to ensure numerical data quality and consistency.

4.2 User Input:

▪ *User Interaction Interface:*

- Users interact with the job recommendation system through a user-friendly web interface.
- The interface is designed to be intuitive and accessible, ensuring a seamless user experience.

▪ *Input Fields:*

- Users provide various inputs, including their skills, job titles, and years of experience.
- These inputs are crucial for the system to generate personalized job recommendations tailored to the user's qualifications and preferences.

▪ *Skills:*

- Users enter their skills or competencies relevant to their desired job roles.
- Skills may include programming languages, technical skills, soft skills, and domain-specific knowledge.

▪ *Job Titles:*

- Users specify the job titles or positions they are interested in applying for.
- This helps the system understand the user's career aspirations and target job recommendations accordingly.

▪ *Years of Experience:*

- Users indicate their years of experience in their respective fields.
- Experience level is a crucial factor in determining job suitability and relevance of recommendations.

4.3 Feature Extraction:

- *TF-IDF Vectorization:*

- TF-IDF (Term Frequency-Inverse Document Frequency) vectorization is a technique used to convert text data into numerical feature vectors.
- It represents the importance of a term in a document relative to a corpus of documents.
- TF-IDF considers both the frequency of a term in a document (TF) and the rarity of the term in the entire corpus (IDF).

- *Text Data Transformation:*

- Job descriptions, skills, and job titles provided by users are transformed into numerical representations using TF-IDF vectorization.
- Each term in the text data is assigned a numerical value based on its TF-IDF score, representing its importance in the context of the document and the entire dataset.

- *Numerical Feature Vectors:*

- After TF-IDF vectorization, text data is represented as numerical feature vectors with each dimension corresponding to a unique term in the vocabulary.
- The values in the vectors indicate the importance or weight of each term in the respective documents.

4.4 Similarity Calculation:

- *Cosine Similarity:*

- Cosine similarity is a metric used to measure the similarity between two vectors in a multidimensional space.
- It calculates the cosine of the angle between the vectors, indicating how closely aligned they are in direction.
- Cosine similarity ranges from -1 to 1, where:
 - 1 indicates perfect similarity (parallel vectors),
 - 0 indicates no similarity (orthogonal vectors),
 - 1 indicates perfect dissimilarity (antiparallel vectors).
- Application in Job Recommendation System:

Cosine similarity is utilized to calculate the similarity between user input vectors (representing user preferences) and job description vectors (representing job postings).

By measuring the cosine similarity between these vectors, the system can determine the relevance and suitability of job postings to the user's preferences and qualifications.

- *Vector Representation:*

- User input vectors and job description vectors are represented as numerical feature vectors obtained through techniques like TF-IDF vectorization.
- Each dimension of the vectors corresponds to a unique term in the vocabulary, and the values represent the importance or weight of the terms in the respective documents.

- *Similarity Calculation Process:*

- To calculate cosine similarity, the dot product of the user input vector and the job description vector is computed.
- The dot product is divided by the product of the Euclidean norms of the two vectors to normalize the similarity measure.
- The resulting cosine similarity score indicates the degree of similarity between the user input and the job description, with higher scores indicating greater relevance.

- *Relevance Determination:*

- High cosine similarity scores suggest strong alignment between the user's preferences and the job requirements, indicating high relevance.
- Job postings with higher cosine similarity scores are considered more suitable matches for the user, and they are prioritized in the recommendation list.

4.5 Experience Adjustment:

- User-provided experience ranges are adjusted to handle smooth similarity calculations.
- Adjusted experience scores are incorporated into the similarity calculation to enhance relevance.

4.6 Recommendation Generation:

In the Job Recommendation System, personalized job recommendations are generated based on similarity scores computed between user profiles and job postings. These recommendations are tailored to match the user's skills, job title, and experience level, providing relevant opportunities for job seekers. The recommendation generation process involves the following steps:

▪ *Computing Similarity Scores:*

The system calculates similarity scores between the user's profile and each job posting in the dataset. Two main factors contribute to these similarity scores:

- **Skills Similarity:** Measures how closely the skills listed in the job posting match those provided by the user. This is computed using techniques such as TF-IDF (Term Frequency-Inverse Document Frequency) and cosine similarity.
- **Title Similarity:** Determines the similarity between the job title of each posting and the user's specified job title. Similarity scores are computed using techniques like TF-IDF and cosine similarity.

▪ *Adjusting Experience Levels:*

- Before generating recommendations, the system adjusts the user's provided experience range to ensure smooth similarity calculations.
- This adjustment ensures that all experience ranges are uniformly structured and facilitates more accurate comparison with job posting requirements.

▪ *Generating Recommendations:*

- Based on the computed similarity scores and adjusted experience levels, the system generates personalized job recommendations.
- The recommendations are selected from the pool of job postings in the dataset and are ranked based on their combined similarity scores.

▪ *Top-Ranking Recommendations:*

- The system selects the top-ranking job postings with the highest combined similarity scores as recommendations for the user.
- These recommendations are considered the most relevant and suitable matches for the user's profile based on their skills, job title, and experience level.

4.7 Database Integration:

In the Job Recommendation System, database integration plays a crucial role in storing user profiles and generated job recommendations. PostgreSQL is used as the relational database management system for efficiently managing and retrieving data. The database integration involves the following aspects:

▪ *Storing User Profiles:*

- User profiles contain essential information such as username, password, skills, and experience. These profiles are stored in the PostgreSQL database to ensure data persistence and enable user authentication and profile management functionalities.
- Database Table: userinfo
username (VARCHAR): Unique identifier for each user.
password (VARCHAR): Securely hashed password for user authentication.
skills (TEXT): Skills listed by the user, which are used for recommendation generation.
experience (INTEGER): Years of experience in the relevant field.
job_title(VARCHAR) : Title of the Job in the relevant field.
Other Fields: Additional information such as name, email, and designation can also be stored as per requirements.

▪ *Storing Job Recommendations:*

- Generated job recommendations are stored in the database for easy retrieval and access. Storing recommendations allows users to view previously recommended job postings and track their job search progress.
- Database Table: recommendations
id (SERIAL): Unique identifier for each recommendation.
user_id (INTEGER): Foreign key referencing the userinfo table, representing the user to whom the recommendation belongs.
job_id (INTEGER): Identifier for the recommended job posting.
company_id (INTEGER): Identifier for the company associated with the recommended job posting.
Other Fields: Additional information such as recommendation timestamp, recommendation score, etc., can be stored based on system requirements.

4.8 User Interaction:

In the Job Recommendation System, users interact with the system through a web interface, which serves as the primary means of input and output. The web interface provides a user-friendly platform for users to input their preferences, view personalized job recommendations, and interact with various features of the system. The user interaction process involves the following aspects:

▪ *Input Gathering:*

Users provide input to the system through the web interface, typically in the form of:

- **User Registration:** New users register for an account by providing essential information such as username, password, name, email, skills, experience, and designation.
- **Login:** Registered users log in to their accounts using their username and password to access personalized features and recommendations.
- **Profile Management:** Users can update their profile information, including skills, experience, and designation, to ensure that recommendations remain relevant over time.

▪ *Recommendation Generation:*

Once users provide their input, the system processes this information to generate personalized job recommendations. The recommendation generation process involves analysing the user's profile, computing similarity scores with job postings, adjusting experience levels, and selecting top-ranking recommendations based on relevance.

▪ *Output Display:*

The system presents personalized job recommendations to users through the web interface. Users can view recommended job postings along with relevant details such as job title, company, domain, and description. The recommendations are displayed in a user-friendly format, allowing users to explore and evaluate potential opportunities easily.

5. ALGORITHM

In the Job Recommendation System, content-based filtering, specifically using cosine similarity and TF-IDF (Term Frequency-Inverse Document Frequency), is employed to generate personalized job recommendations for users based on their skills, job titles, and experience. Let's delve into each component:

1. Cosine Similarity:

- **Definition:** Cosine similarity is a measure of similarity between two vectors in a multi-dimensional space, calculated as the cosine of the angle between them.
- **Usage in the Project:** In the context of the Job Recommendation System:

- Skills Similarity: Cosine similarity is calculated between the user's skills vector and the skills vectors of job postings. This measures how closely the user's skills match the required skills for each job.
- Title Similarity: Similarly, cosine similarity is computed between the user's job title vector and the title vectors of job postings. This evaluates the similarity between the user's job title preference and the titles of available job postings.

2. TF-IDF (Term Frequency-Inverse Document Frequency):

- Definition: TF-IDF is a numerical statistic that reflects the importance of a term (word) in a document relative to a collection of documents. It consists of two components:
 - Term Frequency (TF): Measures the frequency of a term in a document.
 - Inverse Document Frequency (IDF): Measures the rarity of a term across all documents in the dataset.
- Usage in the Project: In the context of the Job Recommendation System:
 - TF-IDF Vectorization: Text data (skills and job titles) from both user profiles and job postings is transformed into numerical vectors using TF-IDF vectorization.
 - Similarity Calculation: Cosine similarity is then calculated between the TF-IDF vectors of user profiles and job postings to determine the similarity between them.

6. TECH STACK

1. Python:

Python serves as the primary programming language for implementing various components of the Job Recommendation System. It is a versatile language known for its simplicity, readability, and extensive libraries, making it well-suited for data preprocessing, analysis, and algorithm implementation.

Key Features and Use Cases:

- Data Preprocessing: Python is used to clean and preprocess the job dataset, including text data normalization, tokenization, and feature engineering.

- **Algorithm Implementation:** Python is employed to implement content-based filtering algorithms, such as cosine similarity and TF-IDF, for generating personalized job recommendations.
- **Anomaly Detection:** Python libraries and algorithms are utilized for anomaly detection tasks, such as identifying outliers or irregularities in the dataset.

2. HTML:

HTML (Hypertext Markup Language) is a standard markup language used for creating the structure and content of web pages in the Job Recommendation System. It provides a framework for organizing and presenting information on the web.

Key Features and Use Cases:

- **User Interface Design:** HTML is used to design and structure the web interface of the Job Recommendation System, including layout, navigation menus, forms, and interactive elements.
- **Content Presentation:** HTML markup is utilized to present job recommendations, user profiles, and other relevant information to users in a structured and visually appealing manner.

3. Flask:

Flask is a lightweight and flexible web framework for Python, used for building web applications in the Job Recommendation System. It provides tools and libraries for handling HTTP requests, routing, and rendering dynamic web pages.

Key Features and Use Cases:

- **Backend Development:** Flask is employed to develop the backend logic and functionality of the Job Recommendation System, including user authentication, recommendation generation, and database interactions.
- **Integration with HTML:** Flask seamlessly integrates with HTML templates to render dynamic web pages based on user interactions and system responses.
- **Database Integration:** Flask facilitates interactions with the PostgreSQL database, allowing for efficient storage and retrieval of user profiles, job recommendations, and other data.

7. TOOLS USED

1. Development Environment:

- Visual Studio Code

2. Data Visualization:

- Matplotlib
- Seaborn
- Wordcloud
- Networkx

3. Data Manipulation and Analysis:

- Pandas
- numpy
- cosine_similarity
- TfidfVectorizer
- psycopg2
- generate_password_hash
- check_password_hash

8 DATASET USED:

The job recommendation system utilizes a dataset sourced from Kaggle, titled "job_info.csv," to generate personalized job recommendations for users. This dataset contains comprehensive information about job postings, including various attributes that enable the system to match user preferences with relevant opportunities. Here's an overview of the dataset:

1. Dataset Name: job_info.csv
2. Dataset Size: Over 9500 rows

3. Attributes:

- Job ID: Unique identifier for each job posting.
- Company ID: Identifier for the company offering the job.
- Job Salary: Salary offered for the job position, if provided.
- Required Experience: Experience level required for the job, typically specified as a range (e.g., 2-5 years).
- Key Skills: Skills required for the job, listed as keywords or phrases.
- Role Category: Category or type of role associated with the job (e.g., software development, data analysis).
- Functional Area: Functional area or department within the organization where the job is located.
- Industry: Industry sector to which the job pertains (e.g., IT, healthcare, finance).
- Job Title: Title or designation of the job posting, indicating the specific role or position.

9 SCREENSHOTS:

9.1 Code Snippets

```
2  <html lang="en">
23 <body>
33   <main class="container">
34     <hgroup>
35       <h1>Job Recommendations</h1>
36       <h3>Here are the recommended jobs based on your job title, experience and skills.</h3>
37     </hgroup><br>
38     <div class="grid">
39       <section>
40         {% if recommendations %}
41         {% if recommendations[0].get('message', None) %}
42         <h5>{{ recommendations[0]['message'] }}</h5>
43         {% else %}
44         <div class="row">
45           {% for job in recommendations %}
46             <div class="col-md-6 mb-4">
47               <div class="candidate-card">
48                 <div class="card-ids">
49                   <p>Company ID: {{ job['company id'] }}</p>
50                   <p>Job ID: {{ job['job id'] }}</p>
51                 </div>
52                 <h4>{{ job['Company'] }}</h4>
53                 <p class="card-text">Job Title: {{ job['Job Title'] }}</p>
54                 <p class="card-text">Salary: {{ job['Job Salary'] }}</p>
55                 <p class="card-text">Experience Required: {{ job['Job Experience'] }}</p>
56                 <p class="card-text">Skills Required: {{ job['Key Skills'] }}</p>
57                 <p class="card-text">Industry: {{ job['Industry'] }}</p>
58                 <p class="card-text">Company URL: {{ job['Domain'] }}</p>
59               </div>
60             </div>
61           {% endfor %}
62         </div>
63         {% endif %}
64       {% endif %}
65     </section>
66   </div>
67 </main>
68 </body>
69 </html>
```

Figure 2: HTML Code for the recommendations page

```

def clean_experience(experience):
    numbers = re.findall("\d+", experience)
    return [int(numbers[0]), int(numbers[-1])] if numbers else [0, 0]

data = pd.read_csv("jobs_info.csv")
data["Experience Range"] = data["Job Experience"].apply(clean_experience)

skills_vectorizer = TfidfVectorizer(ngram_range=(1, 2))
title_vectorizer = TfidfVectorizer(ngram_range=(1, 2))

tfidf_skills = skills_vectorizer.fit_transform(data["Key Skills"])
tfidf_titles = title_vectorizer.fit_transform(data["Job Title"])

def experience_similarity(candidate_exp, job_exp_range):
    # Adjust the similarity calculation for experience to smoothly handle ranges
    if candidate_exp < job_exp_range[0]:
        return max(0, 1 - (job_exp_range[0] - candidate_exp) / job_exp_range[0])
    elif candidate_exp > job_exp_range[1]:
        return max(0, 1 - (candidate_exp - job_exp_range[1]) / candidate_exp)
    else:
        return 1

def recommend_jobs(query_skills, query_title, query_experience):
    query_skills_vec = skills_vectorizer.transform([query_skills])
    query_title_vec = title_vectorizer.transform([query_title])

    skills_similarity = cosine_similarity(query_skills_vec, tfidf_skills).flatten()
    title_similarity = cosine_similarity(query_title_vec, tfidf_titles).flatten()

    # Normalize similarities
    skills_similarity = (skills_similarity - skills_similarity.min()) / (skills_similarity.max() - skills_similarity.min() + 1e-5)
    title_similarity = (title_similarity - title_similarity.min()) / (title_similarity.max() - title_similarity.min() + 1e-5)

    combined_similarity = (skills_similarity + title_similarity) / 2

    # Apply experience similarity and adjust combined score
    experience_scores = np.array([experience_similarity(query_experience, x) for x in data["Experience Range"]])
    combined_score = combined_similarity * experience_scores

    indices = np.argsort(-combined_score)[:10]
    if len(indices) == 0 or combined_score[indices[0]] == 0: # Check if there are no recommendations
        return []

    results = data.iloc[indices]

```

Figure 3: Functions used to find recommendations for user

```

@app.route('/recommend_jobs')
def recommend_jobs_route():
    username = session.get('username')
    if not username:
        return redirect(url_for('login'))

    skills = request.args.get('skills')
    designation = request.args.get('designation')
    experience = request.args.get('experience')
    experience = int(experience) if experience and experience.isdigit() else 0

    recommendations = recommend_jobs(skills, designation, experience)

    conn = create_db_connection()
    if conn and recommendations:
        try:
            cur = conn.cursor()
            cur.execute("SELECT id FROM userinfo WHERE username = %s", (username,))
            user_id_row = cur.fetchone()
            user_id = user_id_row[0] if user_id_row else None

            for job in recommendations:
                company_id = job.get('company id')
                job_id = int(job.get('job id')) # Ensure job_id is an integer
                if company_id:
                    try:
                        cur.execute("SELECT company, domain FROM companies WHERE company_id = %s", (company_id,))
                        company_info = cur.fetchone()
                        if company_info:
                            job['Company'] = company_info[0]
                            job['Domain'] = company_info[1]
                    except Exception as e:
                        print(f"Database query failed due to: {e}")

                if user_id and company_id:
                    cur.execute("""
                        SELECT rec_id FROM recommendations
                        WHERE user_id = %s AND job_id = %s AND company_id = %s;
                    """, (user_id, job_id, company_id))
                    if not cur.fetchone():
                        try:
                            cur.execute("""
                                INSERT INTO recommendations (user_id, job_id, company_id)
                                VALUES (%s, %s, %s);
                            """, (user_id, job_id, company_id))
                        except Exception as e:
                            print(f"Failed to insert recommendation due to: {e}")
                            conn.rollback()

                    conn.commit()
                except Exception as e:
                    print(f"Error processing recommendations: {e}")
                    conn.rollback()
            finally:
                cur.close()
                conn.close()

        if not recommendations:
            recommendations = [{"message": "No recommendations found!"}]

    return render_template('recommendations.html', recommendations=recommendations, username=username)

```

Figure 4: Code to retrieve recommendations from database

9.2 Implementation Results

1. Dataset:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
job id	company	Job Salary	Job Experi	Key Skills	Role Categ	Functional	Industry	Job Title						
10001	10001	Not Discl	2 - 5 yrs	pre sales	Retail Sale	Sales , Ret	IT-Softwar	Sales Executive/Officer						
10002	10015	2,00,000 -	0 - 5 yrs	Technical	Admin/M:	IT Softwar	IT-Softwar	Technical Support Engineer						
10003	10018	Not Discl	2 - 5 yrs	manual te	Programmr	IT Softwar	IT-Softwar	Testing Engineer						
10004	10009	7,00,000 -	5 - 7 yrs	adobe ex	Programmr	IT Softwar	IT-Softwar	System Analyst						
10005	10037	Not Discl	9 - 14 yrs	TFS Azur	Programmr	IT Softwar	IT-Softwar	Technical Architect						
10006	10029	Not Discl	2 - 7 yrs	Bde	Institution	Sales , Ret	IT-Softwar	Sales Executive/Officer						
10007	10033	Not Discl	1 - 5 yrs	technical	Voice	ITES , BPO	IT-Softwar	Associate/Senior Associate -(Technical)						
10008	10036	Not Discl	1 - 6 yrs	website	Admin/M:	IT Softwar	IT-Softwar	Webmaster						
10009	10002	Not Discl	3 - 7 yrs	Report Ge	Accounts	Accounts ,	IT-Softwar	Accounts Manager						
10010	10046	Not Discl	0 - 3 yrs	C# MS D	Programmr	IT Softwar	IT-Softwar	Software Developer						
10011	10017	2,50,000 -	0 - 4 yrs	lead gene	Retail Sale	Sales , Ret	IT-Softwar	Sales/Business Development Manager						
10012	10005	Not Discl	2 - 5 yrs	contract r	Retail Sale	Sales , Ret	IT-Softwar	Sales/Business Development Manager						
10013	10014	Not Discl	6 - 8 yrs	Java EE J	Programmr	IT Softwar	IT-Softwar	Software Developer						
10014	10035	Not Discl	4 - 9 yrs	Conceptu	Programmr	IT Softwar	IT-Softwar	Software Developer						
10015	10035	Not Discl	4 - 9 yrs	C# Java	Programmr	IT Softwar	IT-Softwar	Software Developer						
10016	10025	Not Discl	5 - 10 yrs	qmail wc	Admin/M:	IT Softwar	IT-Softwar	System Administrator						
10017	10048	Not Discl	8 - 10 yrs	Java JPA	Project M:	IT Softwar	IT-Softwar	Project Manager-IT/Software						
10018	10002	Not Discl	6 - 8 yrs	Java java	Programmr	IT Softwar	IT-Softwar	Software Developer						
10019	10048	Not Discl	2 - 7 yrs	GIS ISO 2	Programmr	IT Softwar	IT-Softwar	Project Lead						
10020	10022	Not Discl	0 - 3 yrs	Outbound	Voice	ITES , BPO	IT-Softwar	Associate/Senior Associate -(NonTechnical)						

Figure 5: few rows of jobs_info.csv

2. Dashboard:

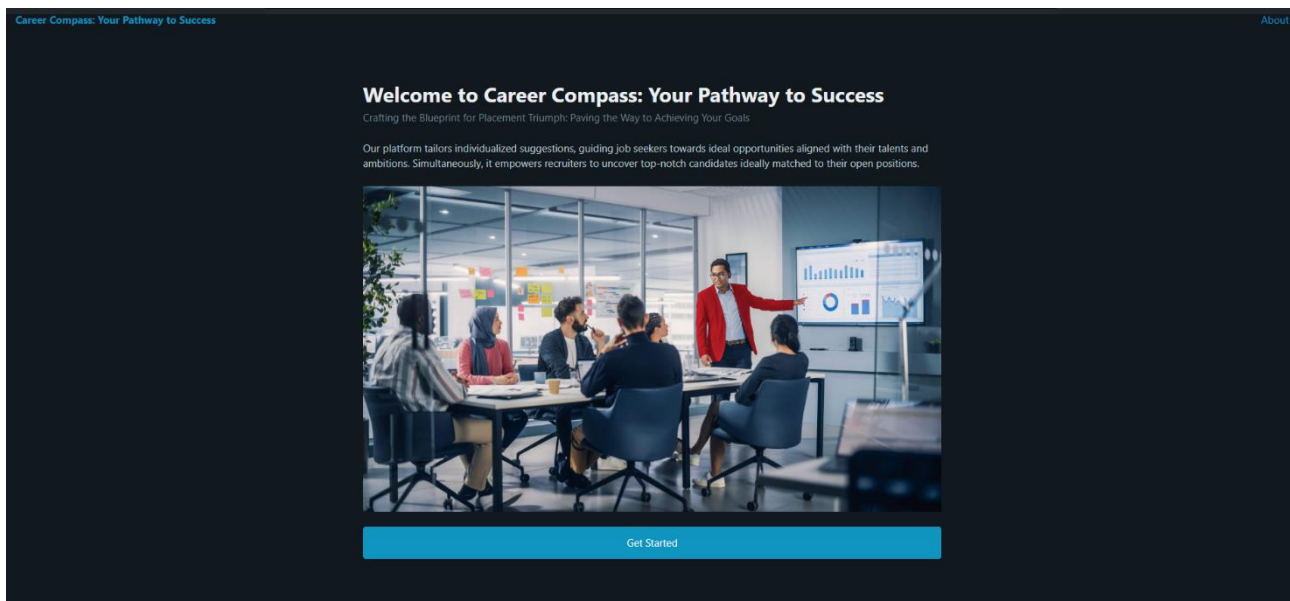
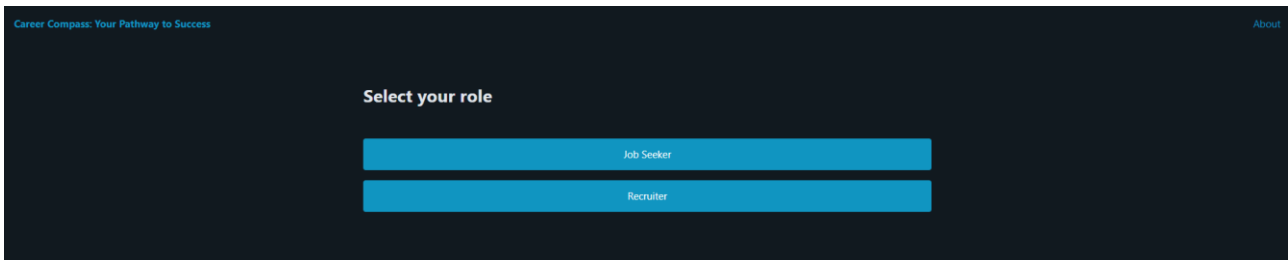


Figure 6: Dashboard of our website

3. Role Choice:

A dark-themed web page with a header "Career Compass: Your Pathway to Success" on the left and "About" on the right. The main heading is "Select your role". Below it are two blue buttons: "Job Seeker" and "Recruiter".

Career Compass: Your Pathway to Success About

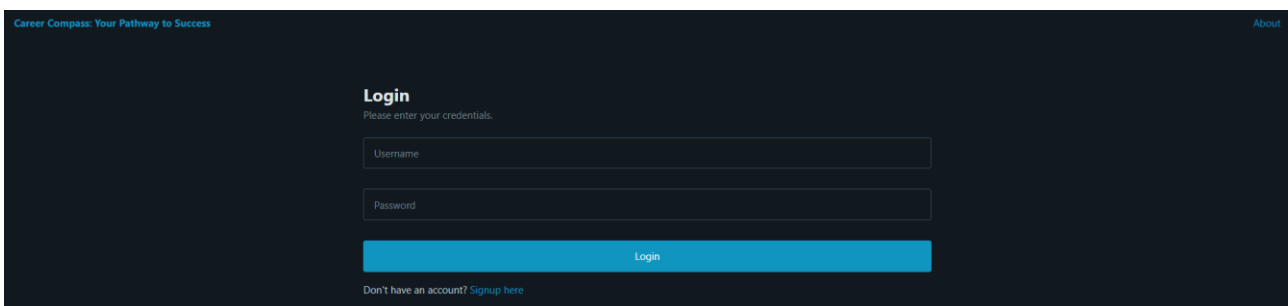
Select your role

Job Seeker

Recruiter

Figure 7: Role choice page of our website

4. Job Seeker's Login Page:

A dark-themed web page with a header "Career Compass: Your Pathway to Success" on the left and "About" on the right. The main heading is "Login". Below it is the text "Please enter your credentials." followed by two input fields: "Username" and "Password". Below these is a blue "Login" button. At the bottom, there is a link: "Don't have an account? Signup here".

Career Compass: Your Pathway to Success About

Login

Please enter your credentials.

Username

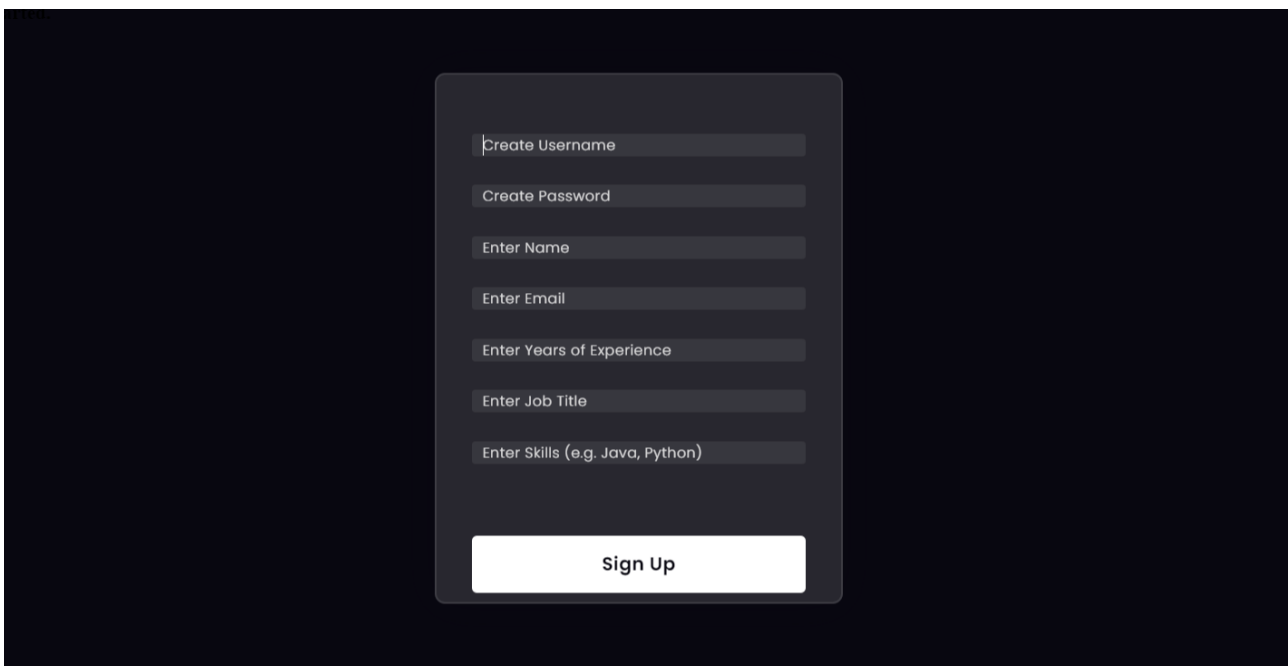
Password

Login

Don't have an account? [Signup here](#)

Figure 8: Login page for job seekers

5. Job Seeker's Registration Page:

A dark-themed web page with a central registration form. The form has a dark background and contains the following fields: "Create Username", "Create Password", "Enter Name", "Enter Email", "Enter Years of Experience", "Enter Job Title", and "Enter Skills (e.g. Java, Python)". Below these fields is a white "Sign Up" button.

Create Username

Create Password

Enter Name

Enter Email

Enter Years of Experience

Enter Job Title

Enter Skills (e.g. Java, Python)

Sign Up

Figure 9: Registration page for job seekers

6. Job Seeker's Profile Page:

Hello, Amritha R!

Username	Amritha2503
Email	amritharaja25@gmail.com
Years of experience	2
Job title	Front end developer
Skills	Java, Flutter, React, HTML/CSS

[Recommend jobs for me](#)

Figure 10: Profile page of user

7. Job Recommendation:

Job Recommendations
Here are the recommended jobs based on your job title, experience and skills.

Unisys	Company ID: 10046
Job Title: Software Developer	Job ID: 16626
Salary: Not Disclosed by Recruiter	
Experience Required: 2 - 4 yrs	
Skills Required: Interpersonal skills HTML CSS Javascript	
Industry: IT-Software, Software Services	
Company URL: unisys.com	
Sify Technologies Limited	Company ID: 10048
Job Title: Web Designer	Job ID: 13258
Salary: 50,000 - 2,75,000 PA.	
Experience Required: 1 - 6 yrs	
Skills Required: CMS HTML CSS	
Industry: IT-Software, Software Services	
Company URL: sifycorp.com	

Figure 11: Job recommendations are displayed based on user's input

9.3 Visualization

Top 10 jobs in the dataset:

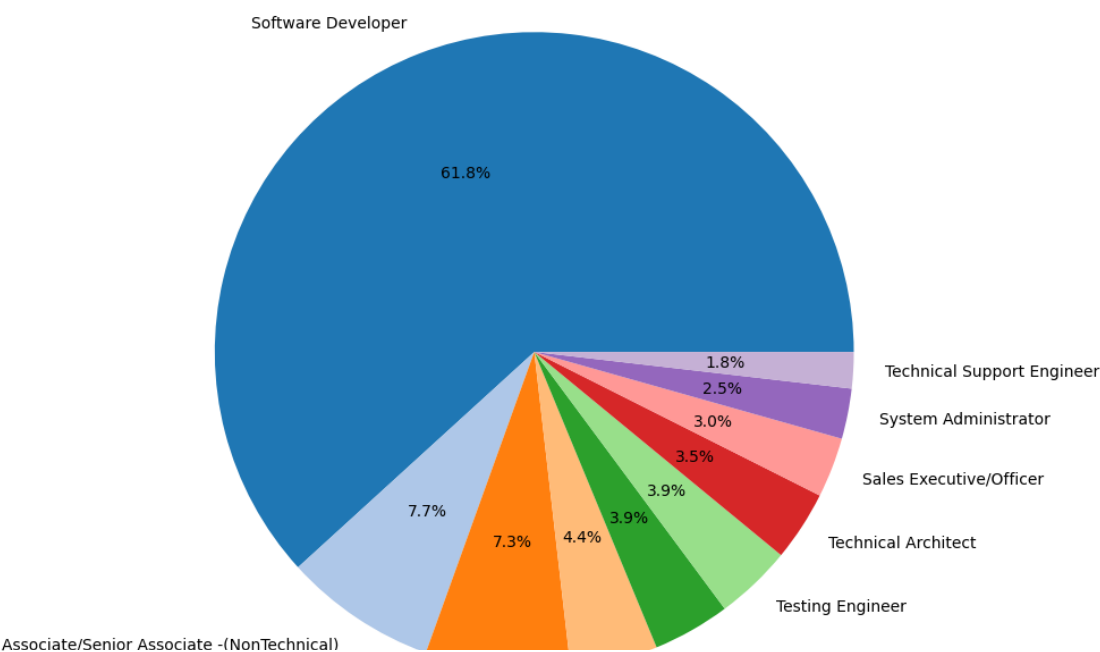


Figure 12: Pie chart to depict the top 10 jobs in the dataset

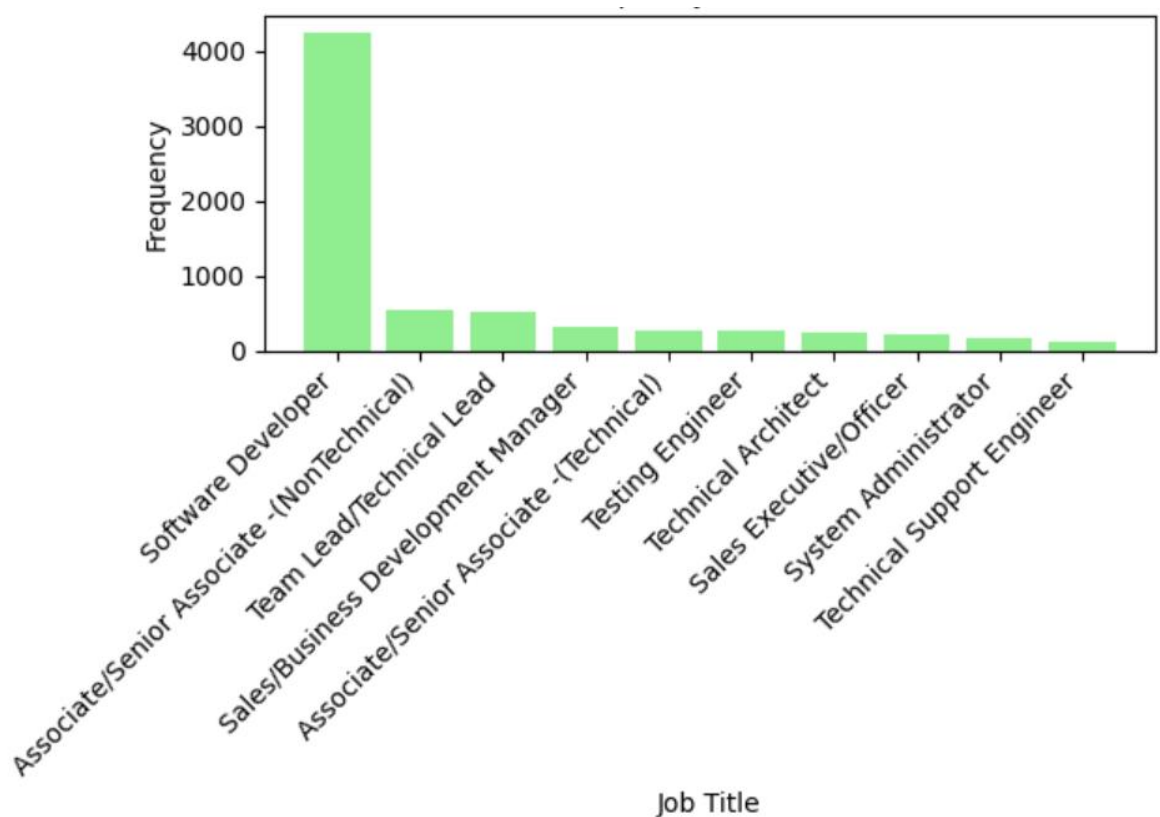


Figure 13: Bar graph to depict the top 10 jobs in the dataset

9.4 Validation

```
import pandas as pd
import re
from collections import Counter
import math

def clean_experience(experience):
    numbers = re.findall("\d+", experience)
    return [int(numbers[0]), int(numbers[-1])] if numbers else [0, 0]

def compute_tfidf(corpus):
    # Compute TF
    tf = []
    for document in corpus:
        words = document.split("|")
        word_count = Counter(words)
        total_words = len(words)
        tf.append({word: count / total_words for word, count in word_count.items()})

    # Compute IDF
    idf = {}
    for document in corpus:
        words = set(document.split("|"))
        for word in words:
            idf[word] = idf.get(word, 0) + 1

    num_documents = len(corpus)
    for word, val in idf.items():
        idf[word] = math.log(num_documents / (val + 1))
```

Figure 14: Python code snippet to validate output

```
tfidf = []
for doc_tf in tf:
    doc_tfidf = {}
    for word, tf_val in doc_tf.items():
        doc_tfidf[word] = tf_val * idf[word]
    tfidf.append(doc_tfidf)

return tfidf

data = pd.read_csv("test.csv")
data["Experience Range"] = data["Job Experience"].apply(clean_experience)

skills_corpus = data["Key Skills"].tolist()
titles_corpus = data["Job Title"].tolist()

skills_tfidf = compute_tfidf(skills_corpus)
titles_tfidf = compute_tfidf(titles_corpus)

print("Skills TF-IDF:")
for i, doc_tfidf in enumerate(skills_tfidf):
    for word, tfidf_val in doc_tfidf.items():
        print(f"({i}, {word}): {tfidf_val}")

print("\nJob Title TF-IDF:")
for i, doc_tfidf in enumerate(titles_tfidf):
    for word, tfidf_val in doc_tfidf.items():
        print(f"({i}, {word}): {tfidf_val}")
```

Figure 15: Python code snippet to validate output

```

(0, 8)      0.1986572046724401
(0, 10)     0.1986572046724401
(0, 201)    0.10001068240976345
(0, 63)     0.16980494649732986
(0, 112)    0.1986572046724401
(0, 103)    0.16980494649732986
(0, 177)    0.16980494649732986
(0, 126)    0.07115842423465317
(0, 130)    0.1986572046724401
(0, 109)    0.1986572046724401
(0, 67)     0.1986572046724401
(0, 58)     0.1986572046724401
(0, 27)     0.1986572046724401
(0, 69)     0.1986572046724401
(0, 59)     0.1986572046724401
(0, 131)    0.1986572046724401
(0, 7)      0.16980494649732986
(0, 9)      0.10001068240976345
(0, 200)    0.09162942719088123
(0, 62)     0.16980494649732986
(0, 111)    0.1493339435411018
(0, 101)    0.1493339435411018
(0, 169)    0.07115842423465317
(0, 125)    0.07115842423465317
(0, 129)    0.1986572046724401

```

Figure 16: Expected Output

```

(0, 8)      0.1986572046724401
(0, 10)     0.1986572046724401
(0, 201)    0.10001068240976345
(0, 63)     0.16980494649732986
(0, 112)    0.1986572046724401
(0, 103)    0.16980494649732986
(0, 177)    0.16980494649732986
(0, 126)    0.07115842423465317
(0, 130)    0.1986572046724401
(0, 109)    0.1986572046724401
(0, 67)     0.1986572046724401
(0, 58)     0.1986572046724401
(0, 27)     0.1986572046724401
(0, 69)     0.1986572046724401
(0, 59)     0.1986572046724401
(0, 131)    0.1986572046724401
(0, 7)      0.16980494649732986
(0, 9)      0.10001068240976345
(0, 200)    0.09162942719088123
(0, 62)     0.16980494649732986
(0, 111)    0.1493339435411018
(0, 101)    0.1493339435411018
(0, 169)    0.07115842423465317
(0, 125)    0.07115842423465317
(0, 129)    0.1986572046724401

```

Figure 17: Actual Output

It is clearly visible that the actual output matches the expected output. This means that the output received is **VALID!**

10 REFERENCES:

1. "Recommendation Systems: Principles, Methods, and Evaluation" <https://www.cambridge.org/core/books/recommendationsystems/9F2AABBB65B23C9EC0E99EACD7932C55>
2. FoDRA — A new content-based job recommendation algorithm for job seeking and recruiting <https://ieeexplore.ieee.org/document/7388018>
3. "Recommendation Systems Handbook: A Complete Guide for Research Scientists & Practitioners" <https://www.springer.com/gp/book/9783030043203>
4. "Content-Based Recommendation Systems" https://link.springer.com/chapter/10.1007/978-3-540-72079-9_2
5. Recommendation system using content filtering: A case study for college campus placement <https://ieeexplore.ieee.org/document/8389579>