Kingdom of Saudi Arabia
Ministry of Education
Princess Nourah bint Abdulrahman University
College of Computer and Information Sciences
Graduation Project I

المملكة العربية السعودية
وزارة التعليم
جـــــامعة الأميرة نورة بنت عبد الرحمـــن
كلية علوم الحـــــاسب والمعلومات
مشروع تخرج 1

Smart Recruitment Recommendation System

KAFAA

Prepared By:

Student Name:	ID:
Sarah Fahad Alotaibi	443007420
Sema Saad Alshamrani	443007423
Yasmeen Ammar Almutairi	443007424
Deema Khaled Alshehri	443007433
Dhiyaa Abdulaziz Alsalem	443007442

Supervisor:

Dr. Manel Ayadi

A Graduation Project Report Submitted to College of Computer Sciences and Information at PNU in Partial Fulfillment of the Requirements for the Degree of Bachelor of Science in Data Science and Analytics

CCIS, PNU

Riyadh, KSA

1446

Acknowledgments:

We would like to express our deepest gratitude to everyone who supported and guided us throughout the completion of this graduation project. Their invaluable contributions and encouragement have been instrumental in its success. First and foremost, we are profoundly grateful to our supervisor, **Dr. Manel Ayadi**, for her exceptional guidance, insightful feedback, and unwavering support. Her expertise and mentorship have been pivotal in shaping this project and ensuring its success. We extend our sincere thanks to **Dr. Amel Elksibi**, whose expertise in artificial intelligence and recommendation systems provided critical insights that enriched the technical aspects of our work. Her thoughtful suggestions greatly enhanced the quality of this project. Our heartfelt appreciation also goes to **Dr. Zuhaira Zain**, for her continuous support and valuable advice throughout this journey. Her assistance has been vital in helping us navigate challenges and achieve our goals. We would also like to thank the **College of Computer and Information Sciences at Princess Nourah bint Abdulrahman University** for providing the resources and facilities needed to complete this project. Finally, we are deeply thankful to our families and friends for their unwavering encouragement and support, which have been a source of strength and motivation throughout this journey.

This project represents the culmination of not only our efforts but also the collective support and guidance of all the individuals and organizations mentioned above. We are truly grateful for their contributions, which have made this journey a fulfilling and memorable one.

Table of Contents

List of Tables

List of Figures

Table of Abbreviations:

Abbreviation	Full Form
PNU	Princess Nourah University
CCIS	College of Computer and Information Sciences
AI	Artificial Intelligence
ML	Machine Learning
NLP	Natural Language Processing
RS	Recommendation System
IR	Information Retrieval
ROC	Receiver Operating Characteristic
AUC	Area Under the ROC Curve
F1-Score	F-measure, combining Precision and Recall
SVM	Support Vector Machine
KNN	K-Nearest Neighbors
DSSM	Deep Semantic Structure Modeling
LSTM	Long Short-Term Memory
CBOW	Continuous Bag of Words
BM25	Best Matching 25
TF-IDF	Term Frequency-Inverse Document Frequency
FAISS	Facebook AI Similarity Search
NER	Named Entity Recognition
AIRM	AI Recruiting Model

BIRCH	Balanced Iterative Reducing and Clustering Hierarchies
RoBERTa	Robustly Optimized BERT Approach
EDA	Exploratory Data Analysis
Min-Max	Min-Max Normalization
All-MiniLM-L6-v2	A pre-trained Minimal Language Model for embeddings
SVD	Singular Value Decomposition
MS MARCO	Microsoft Machine Reading Comprehension Dataset
Word2Vec	Word to Vector (an embedding technique)
CSV	Comma-Separated Values
CV	Curriculum Vitae

Table 1: Abbreviation

Abstract:

This project introduces KAFAA. A Smart Recruitment Recommendation System aimed at improving the effectiveness and openness of the hiring process, in Saudi Arabia. Enhancing talent, matching challenges and minimizing hiring procedures are focuses of KAFAA as it strives to provide job seekers and employers with better access to suitable opportunities. Utilizing cutting edge technologies, like artificial intelligence (AI) machine learning (ML) and natural language processing (NLP). The main features of KAFAA involve matching job opportunities by considering the skills and preferences of candidates to ensure a fit between job seekers and employers. It also includes an AI driven assistant for career advice and practice interviews. Moreover, the system gathers data on government organizations to tackle job postings and enhance trust in the job market.

KAFAAs implementation plays a role in advancing Saudi Vision 2030 by enhancing the labor market's vitality and driving progress while lowering unemployment rates. This initiative signifies not a leap but also a significant stride towards empowering both individuals and organizations to realize their career aspirations and recruitment objectives.

Incorporating technology, like recruitment and recommendation systems into the workforce aligns with Saudi Vision 2030. Leveraging intelligence and machine learning, in natural language processing can enhance job matching. Optimize the employment sector.

Keywords: Smart Recruitment, Recommendation System, Artificial Intelligence, Natural Language Processing, Saudi Vision 2030, Job Matching.

1. Introduction

1.1. Context of study:

In recent years, the recruitment landscape in Saudi Arabia has undergone significant changes, driven by economic diversification efforts under Saudi Vision 2030. The government has been actively promoting the development of human capital, leading to an increase in job opportunities within both public and private sectors. However, challenges persist, such as limited access to job information and inefficient recruitment processes. This study addresses these gaps by providing a technologically advanced solution that aligns with national goals, ultimately benefiting job seekers and employers alike.

1.2. Problem statement & motivation:

The Saudi labor market faces a critical challenge of mismatching, where candidates struggle to find roles that truly align with their skills and experiences, while recruiters face difficulties identifying the right talent for their vacancies [1].From the candidate's perspective, this mismatch often leads to repeated applications to roles they are either overqualified or underqualified for, resulting in prolonged job searches and increased frustration.

Recruiters, on the other hand, are burdened with manually reviewing large volumes of resumes, a process that is not only time-consuming and resource-intensive but also often fails to accurately match candidates to job requirements. This inefficiency results in poor hiring decisions, leading to suboptimal workforce placement and organizational setbacks.

At its core, this mismatching problem underscores the disconnect between candidate profiles and job requirements. Tackling this issue requires a robust, technology-driven solution to enhance the accuracy of job-candidate matching, thereby streamlining the recruitment process, improving workforce alignment, and contributing to the labor market's efficiency in alignment with Vision 2030 goals.

1.3. Project Aim:

The primary goal of this project is to change the hiring process for the better by addressing some of the most common problems companies and job applicants deal with. The main aims are to lower mismatches between job requirements and applicant skills, get more job offers out, make the application process more transparent, optimize time spent on tedious recruiting work, and keep applicants engaged throughout hiring. By putting these plans into action, we hope to improve recruitment for employers and job seekers, making it more productive and engaging for everyone to participate.

1.4. Proposed solution:

The proposed system, **KAFAA**, is primarily a Smart Job Matching System designed to address the pressing challenges of the Saudi labour market. At its core, KAFAA functions as a recommendation system, ensuring accurate alignment between job descriptions and candidate profiles based on their skills, experiences, and preferences. By consolidating job postings, KAFAA reduces fragmentation and makes the job search process more efficient, ensuring candidates focus on roles tailored to their qualifications and aspirations.

The Smart Job Matching System leverages advanced technology to dynamically match candidates with opportunities that fit their profiles. This approach bridges the gap between employers and job seekers, reducing mismatches and enhancing the efficiency of the recruitment process. By providing an intuitive and accessible platform, KAFAA simplifies job discovery and creates a streamlined experience for both candidates and recruiters.

To further enhance the recruitment experience, KAFAA integrates a complementary AI-powered chatbot. This chatbot primarily supports job seekers in three ways:

1. Interview Simulation: The chatbot offers an interactive simulation of job interviews, tailored to specific job descriptions or fields of study. Candidates receive real-time feedback on their responses, enabling them to refine their skills and build confidence for actual interviews.
2. Career Guidance and Consultation: Personalized career consultations are provided by the chatbot, helping users identify potential career paths, improve their professional development, and optimize their job search strategies. Recommendations are aligned with the candidate's unique qualifications, ensuring relevant and practical advice.
3. General Assistance on Saudi Companies: The chatbot delivers detailed information about various Saudi companies, including their missions, roles, and verified contact details. This feature equips candidates with the knowledge needed to engage effectively with potential employers and fosters trust by providing accurate and reliable information.

By focusing on its main function as a Smart Job Matching System, complemented by the AI chatbot's features, KAFAA empowers job seekers, streamlines the recruitment process, and aligns seamlessly with the goals of Saudi Vision 2030.

1.5. Project plan:

The following project plan outlines our timeline for developing a comprehensive job recommendation system, spanning from August 2024 to the end of November 2024, with four main chapters covering system overview, literature review, data analysis, and model implementation.

Figure 1 : Project Plan

1.6. Structure of the Report:

Chapter 1: This chapter provides an overview of our job recommendation system, explaining its purpose, goals, and overall structure.

Chapter 2: This chapter reviews related work, summarizing existing systems and research to highlight the foundation for our approach.

Chapter 3: This chapter focuses on data collection, sources, exploratory data analysis (EDA), and preprocessing steps like cleaning and organizing the data.

Chapter 4: This chapter outlines the plan for building the recommendation model, including the techniques and approaches to be implemented.

1.7. Definitions of New Terms:

These terms are key to providing a better understanding of the project

Term	Definition
Recommendation Systems	Algorithms that suggest items (e.g., products, movies, jobs) based on user preferences and past behavior.
Artificial Intelligence	The science of creating intelligent agents that can reason, learn, and make decisions.

Labor Market	The market where labor is bought and sold, involving job seekers and employers.
Vision 2030	A strategic framework developed by the Saudi Arabian government to diversify the economy and reduce reliance on oil.
TF-IDF	A technique used in natural language processing to measure the importance of words in a document relative to a collection of documents.
Tokenization	The process of breaking down text into individual words or tokens.
Job Machine	A hypothetical machine that could automate job search and matching processes.
Job Embeddings:	Numerical representations of job descriptions that capture semantic and syntactic information.
Natural Language Processing	A field of AI that focuses on the interaction between computers and human language.
Hybrid Recommendation Systems	A combination of multiple recommendation techniques to improve accuracy and relevance.

Table 2: Definition of New Terms

Summary

In this chapter, we have covered the problem that our proposed solution will resolve. As well as the project aim and our project plan, and discussed the structure of this report, and defining the project new terms.

2. Related Work

2.1. Background Material:

2.1.1. Introduction

According to Anna Gatzioura and Miquel Sánchez [2], the objective of a recommendation system is to provide users with valuable and relevant content (items) based on their interactions and preferences.

Recommendation systems are part of AI, aimed at predicting user preferences through the suggestion of relevant options. These systems are specifically designed in such a way that they could predict user preferences and suggest items, services, or information that best meets individual needs. The significance of RSs involves a wide range of industries, where they serve to offer personalized suggestions that enhance user satisfaction and assist in informed decision-making. In a time where rapid technological development and exponential growth of information is present, Recommendation systems have developed as a means to overcome the problem of information overload. The problem nowadays is that users have too many choices to make, which results in choice overload and a lack of ability to find a product or content that actually reflects their interest. Recommendation systems cope with the problem of the unexpected emergence of overwhelming volumes of data by many methods; they represent the user with listings of options that best reflect their preferences and needs. [3]

In the development of recommendation systems different strategies have been developed that consider the unique needs and preferences of the users. They would help deal with information overload and a large variety of choices that face the user in the current digital environment. Recommendation systems rely on user-generated data, item data, and user-preference data to produce recommendations, as shown in *Figure 2*

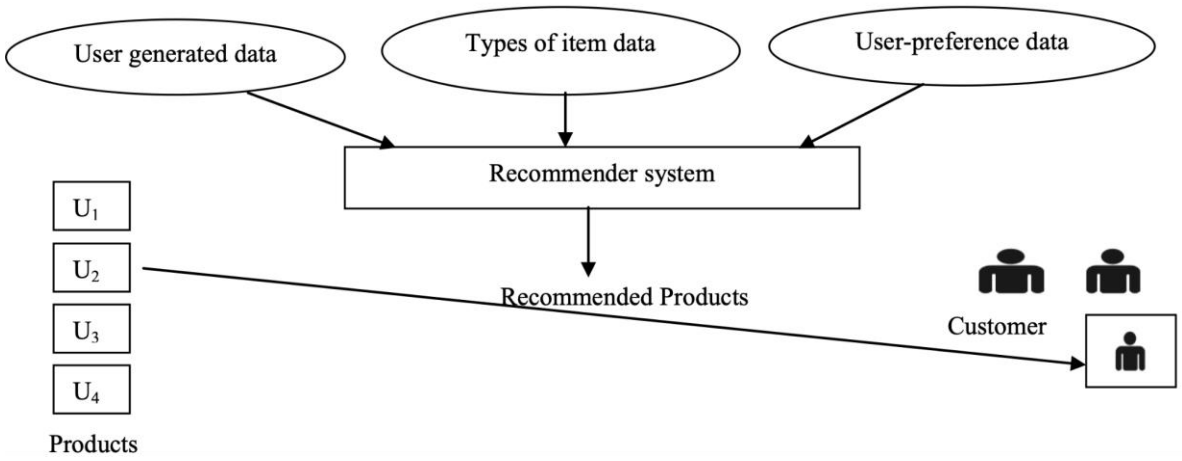

Figure 2: Framework of Recommendation Process

recommendation systems follow a structured process, as shown in Figure 1. The process involves the following key components:

- **User-generated data:** Information collected from the user's actions, such as past purchases, clicks, or reviews.
- **Item data:** Characteristics or features of the items that can be recommended, such as categories, descriptions, or attributes.
- **User-preference data:** Data that captures the user's preferences, which can be gathered from explicit feedback or inferred from interactions.

2.1.2. Types of recommendation systems

The recommendation system processes this data to generate suggestions, which are then presented to the user. This process helps users discover products or content that best match their needs and interests. [4]

There are two main types of recommendation systems: personalized and non-personalized. Personalized systems focus on individual user data to provide customized recommendations based on activities like user ratings, buying habits, and browsing behavior. This method is commonly used to enhance user engagement and satisfaction.

non-personalized recommendation systems take a more general approach. They suggest items based on their features and broader user trends rather than individual preferences. This system is used where content recommendations rely on overall popularity or feature evaluation rather than personal user data. While non-personalized systems are simpler and less intrusive, they lack the ability to directly influence individual buying behaviors or interests. Both systems are crucial in their respective contexts, with personalized systems being more intricate and impactful in driving user interaction. [5]

2.1.3. approaches of recommendation system

Over the years, with evolution in technology and user behavioral aspects classic recommendations approaches have been established. Each of these different approaches has its focus on fine-tuning recommendations based on one or another aspect, such as past interactions made by the user, preference of similar users, or contextual information like time and location. Understanding the different recommendation approaches is crucial, each of the recommendation approaches has different strengths and limitations; and many systems utilize a combination of methods to optimize recommendation accuracy.

The main classes of recommendation approaches include:

- **Content-Based Recommendation:**

The system suggests items that are like what the user already liked or interacted with. The similarity of the items would be calculated with attributes like genre, keywords, or other attributes. For example, users who enjoy reading mystery novels will get recommendations on other mystery novels.

Advantage: Content-based recommendations are useful for making early suggestions when there is limited information about user preferences. This is an approach that plays out well in many situations where items are 'well-defined' by their features.

- **Collaborative Filtering:**

  This widely used technique where the system makes recommendations based on users' tastes that are similar, generally follows two major approaches: user-based and item-based collaborative filtering. User-based collaborative filtering recommends items liked by people of similar tastes, whereas item-based collaborative filtering suggests items like those the user has rated highly.

  Advantage: include collaborative filtering effectiveness in personalization because it learns patterns from the user's behavior. It might suffer from the "cold start" problem when not enough interaction data is available for a new user or item.

- **Demographic-based recommendation:**

  The system bases recommendations on demographic groups of users based on characteristics such as age, gender, or geographical location, this approach customizes recommendations to these groups. A demographic-based system could recommend news from a certain region to users from that region.

  Advantage: Demographic-based recommendation systems are helpful for creating initial recommendations, particularly for new users. It allows recommendations to generalize on similar users without having prior data specific to everyone.

- **Knowledge-Based Recommendation:**

  Knowledge-based systems use detailed domain knowledge to find matches for the user by matching items for the most specialized needs or requirements of that user. For example, a knowledge-based system might recommend cameras based on the user’s photography experience and intended use (e.g., for travel or professional work).

  Advantage: This works when there is little user history or when preferences are very specific. Knowledge-based systems provide relevant suggestions in domains where the "utility" of items can be closely tied to specific attributes or user criteria.

- **Community-Based Recommendation:**

  Community-based or social recommendation systems recommend items based on the preference of friends or social contacts of the user. This approach focuses on social networks and leverages the idea that users often trust and share the preferences of their friends or people in their immediate social circles.

Advantage: Such recommendations can build credibility and trust, especially on social networking sites where users value recommendations made by friends. This method proves most helpful where information is shared within close social networks.

- **Hybrid Recommendation:**

  Hybrid systems combine two or more recommendation approaches to enhance the accuracy of recommendations. For example, a hybrid recommendation system could integrate collaborative filtering with content-based recommendations to overcome their limitations. For example, hybrid system combining (collaborative filtering and content-based recommendations) may initially filter recommendations according to content, such as genre, and then prioritize them based on user similarity.

  Advantages: Hybrid methods improve personalization by addressing individual shortcomings, like the cold start problem in collaborative filtering. They also provide flexibility by adapting recommendations using various data sources, resulting in increased precision and relevance. [3]

In addition to these approaches, various advanced approaches have been developed to address specific challenges in recommendation systems. By continually evolving and integrating new data and techniques, these systems aim to offer more relevant and personalized recommendations, therefore enhancing the overall user experience in a wide range of domains.

Commonly used evaluation metrics in Recommendation Systems involve comprehending and quantifying essential performance metrics such as Precision, Recall, Accuracy, ROC Curves, and F-score. Each metric offers distinct insights into the efficacy of recommendations.

2.1.4. Evaluation metrics in recommendation system

Precision:

Precision is fundamental metrics derived from Information Retrieval, displaying the Recommendation Systems aim of suggesting relevant items from a pool of resources.

Precision It measures the fraction of relevant items among all items recommended to a user. [6]

$$Precision = \frac{a}{a + b}$$

Where:

a = number of relevant items recommended

b = number of irrelevant items recommended

Recall:

Recall is fundamental metrics derived from Information Retrieval, displaying the Recommendation Systems aim of suggesting relevant items from a pool of resources.

Recall measures the fraction of relevant recommended items out of the total relevant items that should have been recommended.

$$Recall = \frac{a}{a + c}$$

Where:

a = number of relevant items recommended

c = number of relevant items not recommended

A proficient Recommender System strives to enhance both Precision and Recall, ensuring a broad coverage of relevant items while upholding their relevance. [6]

Accuracy:

Represents the percentage of correctly recommended items out of all recommendations provided. Usually, it is computed using split-validation, where a part of a user's behavior history dataset is used for model training, and the remaining part is allocated for testing purposes. [6]

$$Accuracy = \frac{Number\ of\ successful\ recommendations}{Total\ number\ of\ recommendations}$$

ROC:

The Receiver Operating Characteristic (ROC) curve evaluates the balance between Recall (True Positive Rate) and Fallout (False Positive Rate). By visualizing this, the ROC curve assists in identifying the optimal threshold for categorizing items as recommended or not recommended.

A perfect ROC curve maximizes Recall and minimizes Fallout, with the Area Under the ROC Curve (AUC) quantifying this trade-off. An AUC value nearing 1 signifies a greater likelihood of accurate item classification. [6]

F-measure:

Combines both Precision and Recall evaluating a recommendation system's performance. It uses a parameter, β(beta), to adjust the balance between Precision and Recall:

β > 1: Prioritizes Recall (finding all relevant items).

β < 1: Prioritizes Precision (only recommending highly relevant items).

$$F\beta = \frac{(1 + \beta^2) \times (Precision \times Recall)}{(\beta^2 \times Precision) + Recall}$$

β = 1: Weights Precision and Recall equally, called the F1-Score.

$$F1 - Score = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

The selection of β relies on the system's requirements. For example, if relevant items are strongly undesired (high Recall), a higher β would prioritize Recall. If the goal is to recommend only highly relevant items, a lower β would prioritize Precision. [6]

By using these metrics, a thorough comprehension of a Recommendation System's performance can be analyzed, enabling modifications to align with organizational objectives.

2.1.5. Examples of recommendation system

Recommendation systems play a major role in various industries, enhancing user experiences and operational efficiency. In streaming services, these systems provide personalized content suggestions by analyzing user preferences and viewing history, ultimately boosting engagement and retention. Social network platforms utilize them to recommend friends, groups, and content by studying user interactions and interests, promoting connectivity and user activity. In tourism services, recommendations are customized based on past trips and preferences, motivating users in discovering new experiences. E-commerce platforms enhance the shopping journey by suggesting relevant products, leading to increased sales and customer satisfaction. Healthcare services leverage recommendation systems to offer personalized health guidance and treatment options, ultimately improving outcomes and patient trust. Education services use these systems to customize learning experiences, providing courses and materials that cater to individual needs, thereby enhancing engagement and progression. Lastly, recruitment systems match job seekers with suitable positions by analyzing profiles and preferences, ultimately improving job search experience and recruitment success. These diverse applications underscore the significant impact of recommendation systems across various industries. [7]

2.1.6. Recommendation system in recruitment

We understand that Recommendation systems aim to predict user preferences and propose relevant options. In recruitment, these systems aim to align candidates' skills and experiences with available job opportunities, therefore addressing issues like mismatched talent, information gaps, and long, inefficient hiring process.

Recruitment in simple words is finding and attracting qualified candidates for job openings within an organization. The goal is to ensure the organization has the right people, with the right skills, in the right positions, to meet both current and future needs. [8]

The recruitment process is complex, involving multiple stages from job posting to final selection. Employers frequently encounter challenges in finding candidates who meet specific job requirements, while job seekers face their own challenges, such as fake job advertisements, long hiring processes, and lack of transparency

throughout the process. Additionally, many job seekers remain unaware of suitable positions due to the lack of a centralized platform, which makes it difficult for job seekers to find suitable positions, contributing to high unemployment rates. [8]

By developing a personalized job recommendation system which will customize job recommendations based on an individual's specific skills, experience, and preferences. By examining data such as resumes, search history, and profile information, the system can predict which job roles are most suitable for the person. This system will smooth the job process, enabling candidates to easily discover positions that match their qualifications and preferences. For employers, this method involves connecting with better-suited candidates and will smooth the long recruitment process. [9] This system will benefit both parties: job seekers can more easily find opportunities matching their qualifications, while employers can identify and attract the right talent faster and more efficiently.

Job recommendation systems utilize machine learning to conduct analysis of candidate data, aligning the system's forecasts with user preferences. Machine learning algorithms process data in real-time, adjusting recommendations based on changing trends and preferences. In recruitment, this implies that as the job market evolves or candidate qualifications vary, the recommendation system can learn and adapt recommendations accordingly. For instance, natural language processing (NLP), a subset of AI, enables systems to analyze resumes and job descriptions, pinpointing keywords that match candidates with suitable job positions. This approach streamlines the job search process for job seekers, reducing the time and effort needed to manually sift through job listings. [10]

Machine learning models such as logistic regression, decision trees, and deep neural networks are commonly employed to train recommendation systems on extensive datasets. This allows them to forecast outcomes based on user profiles and job data. For instance, logistic regression can assess the probability of a candidate clicking on a specific job post, while deep learning can identify intricate patterns in user behavior and job market trends. These models enhance the system's capability to provide timely and relevant recommendations, aiding candidates in discovering suitable positions and assisting employers in connecting with top-tier talent. Within job recommendation systems, these models elevate search efficiency and accuracy by presenting users with customized lists of job opportunities that align precisely with their profiles. [11]

2.1.7. Conclusion

Job recommendation systems like our project KAFAA leverage machine learning and AI to offer candidates personalized job opportunities and simplify the recruitment process for employers. By analyzing a variety of data sources and fine-tuning system performance using evaluation metrics, job recommendation systems serve as a bridge between candidates and employers, facilitating an effective and mutually beneficial hiring process that aligns with the objectives of Saudi Vision 2030. With the ongoing advancements in AI, the capabilities of

job recommendation systems to revolutionize recruitment processes are poised to expand, fostering a dynamic and adaptive job market.

2.2. Literature review:

We conducted an in-depth analysis of existing studies on jobs recommendation systems, extracting the most relevant insights into our project's objectives. Through this review, we identified key methods and techniques that contribute to building effective job recommendation models, as well as challenges that need to be addressed for optimal performance. To provide a clear framework for evaluating each study, we categorized the studies based on three primary recommendation approaches: **Collaborative Filtering**, **Content-Based Filtering**, and **Hybrid Filtering**. Each main approach is further divided into specific subcategories, allowing us to understand the nuances of each technique and assess its suitability for our project.

1. **Collaborative Filtering**
    - **User-Based Collaborative Filtering**: Studies that fall under this category, such as Recommendation System for Online Job Vacancy Using ML [12]Models and A Recommendation System for Selecting the Appropriate Undergraduate Program Using Graduate Data [13], use user behavior and preference data to suggest items based on the actions of similar users. While user-based collaborative filtering is useful for personalized recommendations, it suffers from limitations such as the cold start problem, where new users or items lack sufficient interaction data for accurate recommendations.
    - **Item-Based Collaborative Filtering**: Studies such as Job Recommendation System Using Content and Collaborative Filtering with Word2Vec [14] focus on matching items (like job postings and candidate profiles) based on similarity measures. However, item-based collaborative filtering can face challenges with data sparsity, as it requires a significant amount of data to identify reliable patterns in item similarity.
2. **Content-Based Filtering**
    - **Text-Based Content Filtering with NLP**: Studies like Enhancing Job Recommendations Using NLP and Machine Learning Techniques [15] employ advanced NLP techniques, such as Word2Vec and Named Entity Recognition (NER), to analyze textual data in job descriptions and resumes. These techniques allow for a more refined, context-based recommendation, enhancing accuracy and relevance. However, NLP-based models are computationally intensive, which may pose challenges for real-time applications.
    - **Skill and Qualification Matching**: Other studies, such as Learning-Based Matched Representation System for Job Recommendation [16] and A Machine Learning Approach

for Automation of Resume Recommendation [17], use simpler content-based filtering focused on matching specific skills or qualifications without advanced NLP. While this approach is efficient and requires less computational power, it lacks the personalization of collaborative filtering, which can lead to overspecialized recommendations.

3. **Hybrid Filtering**
    - **Simple Hybrid Models**: Studies such as Job Recommendation System Using Hybrid Filtering combine content-based and collaborative filtering techniques [18] in straightforward hybrid models, which aim to balance the strengths of each approach. These models provide a versatile solution but may still struggle with issues like cold starts and may require substantial user data to maintain recommendation quality.
    - **Advanced Hybrid Models with Deep Learning or Specialized Techniques:** More sophisticated hybrid models, such as those presented in Deep Semantic Structure Modeling (DSSM) [19] for Job Recommendation, Prediction of Recommendations for Employment Utilizing Machine Learning Procedures and Geo-Area Based Framework [20], and the AIRM for the Saudi Arabia Labor Market [21], use advanced techniques like DSSM for deep semantic understanding, geo-location filtering, and clustering for contextual matching. The AIRM model, in particular, leverages techniques like RoBERTa embeddings and FAISS indexing to address large-scale data challenges, providing high-accuracy recommendations tailored to the Saudi labor market. Although highly effective, these models are computationally demanding and require significant tuning and resources for optimal performance in large-scale applications.

This structured categorization of the studies allows us to systematically **evaluate each approach's advantages and limitations in the context of our project**. By understanding the distinctions between these methods, we can identify the most suitable candidate models and leverage the insights gathered to inform the design of our recommendation system.

This table presents a summary of key studies related to job recommendation systems, highlighting their methodologies, datasets, evaluation metrics, and limitations. The table below provides a concise overview, enabling a clearer understanding of the weaknesses of various approaches and guiding the design of our proposed solution.

Title of Study	Approaches	Algorithms	Dataset	Evaluation Metrics	Limitations
Recommendation system for online job vacancy using machine learning models [12]	Collaborative Filtering	SVM, KNN, Naïve Bayes	24,475 job postings and 200 resumes from Kaggle	Accuracy: SVM: 69%, KNN: 71%,	Metrics may not fully capture long-term recommendation relevance
A Recommendation System for Selecting the Appropriate Undergraduate Program at Higher Education Institutions Using Graduate Student Data [13]	Collaborative Filtering	Decision Tree, SVM, Random Forest	MBA student data from CMS Business School (Kaggle)	Accuracy: Decision Tree: 79%, SVM: 84%, Random Forest: 95%	Requires high-quality input features, preprocessing critical
Job Recommendation System using Content and Collaborative Filtering based Techniques [14]	Collaborative Filtering	Deep Learning with Word2Vec	WUZZUF dataset (online recruitment platform)	Precision: 78%, Recall: 63.97%, F1: 70.4%	collaborative filtering overfits, content filtering reduces diversity
Enhancing Job Recommendations Using NLP and Machine Learning Techniques [15]	Content-Based Filtering	NLP (Word2Vec, Cosine Similarity, Sentiment Analysis, Named Entity Recognition)	Stack Overflow Developer Survey and web-scraped job postings	Precision: 77%, Recall: 89%, F1: 82.6%	Requires substantial user data, struggles with unique preferences or information overload
Learning-Based Matched Representation System for Job Recommendation [16]	Content-Based Filtering	NLP (Cosine Similarity, Jaccard Coefficient)	Dataset scraped from Indeed.com	Precision: 88%, Recall: 86%, F1: 86%	Limited Data Source, focused on three Saudi cities. Content-Based Filtering, Limited to explicit skills, no collaborative filtering used.
A Machine Learning Approach for Automation of Resume Recommendation System [17]	Content-Based Filtering	Random Forest, Naïve Bayes, Logistic	Resume dataset from Kaggle	Accuracy: Random Forest: 38.99%, Naïve Bayes:	The model is restricted to CSV files, limiting flexibility for other formats. Also,

		Regression, Linear SVM		44.39%, Logistic Regression: 62.40%, Linear SVM: 78.53%	Gensim-based summarization may lose essential details, requiring adjustments to retain key information.
Job Recommendation System Using Hybrid Filtering [18]	Hybrid Filtering	TF-IDF, Cosine Similarity, SVD, Weighted Average Technique	The Jobs data was sourced from AmbitionBox for company information and profile data artificially generated user profiles.	Precision: 41.11%, Recall: 62.44%, F1: 47.68%	Cold-start problems, sparsity issues, reliance on user data
Enhanced DSSM (Deep Semantic Structure Modeling) Technique for Job Recommendation [19]	Hybrid Filtering	DSSM, LSTM, TF-IDF, BM25, Doc2Vec,	Naukri.com and CareerBuilder.com datasets	Precision: 91.9%, Recall: 96.6%	Requires large datasets, struggles with smaller datasets
Prediction of Recommendations for Employment Utilizing Machine Learning Procedures and Geo-Area-Based Recommender Framework [20]	Hybrid Filtering	Random Forest, Logistic Regression, KNN, SVM, AdaBoost	LinkedIn profile and Facebook datasets	Accuracy: LinkedIn: 99.78%, Facebook: 99.03%	Limited to LinkedIn and Facebook, restricted generalization
Implementing AIRM: A New AI Recruiting Model for the Saudi Arabia Labor Market [21]	Hybrid Filtering	BIRCH, RoBERTa, FAISS	Generated dataset	Accuracy: 84%	No specific mechanism to recognize or give credit to candidates with double majors.

Table 3: Studies

Gaps and Limitations

Upon reviewing the studies, several gaps and limitations became apparent across the different approaches, highlighting areas for improvement in recommendation systems. **Collaborative filtering** approaches, while effective for personalized recommendations, frequently encounter the cold start problem and data sparsity issues, particularly when user interaction data is limited. **Content-based filtering** approaches demonstrate strong performance in skill and qualification matching but often lack personalization and tend to produce overspecialized recommendations, which may not suit diverse job roles. Additionally, **hybrid filtering models**, though they address many limitations of individual filtering techniques, tend to be computationally demanding. Many hybrid models incorporate advanced methods like deep learning or geo-location filtering, which, while powerful, require substantial resources and careful tuning to achieve efficiency.

These gaps indicate the need for models that balance personalization, computational efficiency, adaptability across different user scenarios, and reflect preferences accurately. Addressing these limitations is essential to developing a robust recommendation system that can deliver accurate, relevant, and scalable job recommendations.

Candidates Models

After reviewing the studies and evaluating their strengths and limitations, we selected four candidate models that are well-suited to our project's objectives. These models include three state-of-the-art approaches derived from the studies and one proposed model that addresses specific limitations and enhances recommendation quality. Each candidate model aligns with one of the subcategories outlined in the introductory section, ensuring that we leverage a diverse range of techniques to build a robust recommendation system.

State of the art models:

1. **Candidate Model 1: User-Based Collaborative Filtering with Machine Learning**
    - **Justification and Features**: User-based collaborative filtering as demonstrated in the study *"Recommendation System for Online Job Vacancy Using ML Models"* [12] leverages user preferences and behavioral data to make personalized recommendations. The studies in this category demonstrated that user-based collaborative filtering provides an effective foundation for personalized job matching. However, we plan to address common issues such

as the cold start problem by integrating additional data preprocessing and smoothing techniques to ensure robust recommendations, even for new users or jobs.

2. **Candidate Model 2: Text-Based Content Filtering with NLP Techniques**

- **Source Study**: *Enhancing Job Recommendations Using NLP and Machine Learning Techniques* [15]
- **Justification and Features**: This candidate model uses NLP methods such as Word2Vec and Named Entity Recognition (NER) to capture semantic relationships within job descriptions and resumes. This approach was shown to improve relevance by extracting entities like skills and qualifications, which are critical in making job recommendations. To apply this model in our project, we will incorporate NER for entity extraction and sentiment analysis for enhanced matching, making it particularly effective in matching skills with job requirements.

3. Candidate Model 3: Semantic AI Job Matching with BIRCH and FAISS

- **Justification and Features**: This candidate model, inspired by the study *"Enhancing Job Recommendations Using NLP and Machine Learning Techniques"* [15] Specifically developed for Saudi Arabia, AIRM uses BIRCH clustering for efficient candidate grouping, RoBERTa sentence transformers for precise job-candidate matching, and FAISS indexing for rapid retrieval. These components enable high-accuracy, context-driven recommendations that align with the unique requirements of a government job portal in Saudi Arabia. AIRM delivers high matching accuracy (84%) and dramatically reduces manual sorting from days to minutes, ideal for large-scale, high-demand environments like Saudi Arabia's government job portal. The model's clustering and retrieval methods ensure scalability and speed, while its data-lake architecture streamlines integration, making it a powerful tool for efficiently managing Saudi labor market data.

4. **Proposed Model: Advanced Hybrid Filtering with NLP and machine learning**

- **Design Rationale**: Drawing on insights from studies that used advanced hybrid approaches, our proposed model integrates multiple techniques to maximize relevance and personalization. This model will incorporate NER for entity recognition, collaborative filtering for preference matching. By combining these approaches, we aim to create a flexible, personalized recommendation system that delivers precise job matches tailored to each user's unique preferences.

- **Planned Implementation**: Our proposed model is designed to address limitations seen in previous studies, such as computational intensity in NLP models and the cold start problem in collaborative filtering. We will use optimization techniques like feature engineering and dimensionality reduction to streamline processing, ensuring that the model remains efficient and effective across a variety of user scenarios.

By structuring our candidate models according to the categories established in the review, we ensure that each model contributes a unique strength to the recommendation system. This selection of models allows us to leverage proven techniques from literature while innovating to address gaps identified in the studies. Our approach creates a well-rounded foundation for a recommendation system that is accurate, personalized, and adaptable to user needs.

2.3 Summary

In chapter 2 we provided background information on the recommendation system, defined our recommendation system approaches and its related component and evaluation metrics, also we reviewed relevant literature and selected four candidate models

3. Data collection and preparation

In this chapter we focused on the data we used to build the recruitment recommendation system. We made sure that the quality of the data matches our objective. To successfully match candidate with suitable job opportunities. We work with two main datasets: one containing job postings in Saudi Arabia and the other contains the LinkedIn profiles of the candidates. These two datasets were carefully chosen to support our objective of matching job seekers' qualifications to relevant jobs. We also performed Exploratory Data Analysis to understand the data better before preprocessing and to spot any issues before preparing the data for analysis using univariate, bivariate, and multivariate analysis, we gain insights into the data that help guide our data cleaning and transformation process.

3.1. Data Sources

The data used in the recruitment recommendation system is divided into two data sets, one focusing on job postings in Saudi Arabia and the other on candidate profiles. These datasets were carefully selected to ensure that it support our problem and works better for the recruitment recommendation system in connecting job opportunities and the candidates with their qualifications.

Datasets:

- **Job Dataset:**

  The first dataset is in the LinkedIn Jobs Data in Saudi Arabia 2020, the source of the data is from Kaggle. It contains detailed information about the job postings in Saudi Arabia during 2020.

  Includes (*LinkedIn ID, position ID, Position, company, location, level, job function, industries, description*). The goal is to analyze this dataset to capture the key skills and qualifications needed in the job market, this is essential to understand the candidates in demand across various industries in Saudi Arabia. [22]

- **Candidate Dataset:**

  The dataset used in this project combines candidate's profiles from both linked LinkedIn and GitHub, we used the LinkedIn's Sales Navigator tool to extract profile data, including skills, experience, certifications, and more. By combining and integrating the data from these two sources, we were able to create a complete and useful dataset that effectively addresses the problem we are working on. The reason for combining the two datasets is that the first dataset alone contained an insufficient number of records for our project's needs. The dataset includes (*Intro, Full name, Workplace, Location, Connections, Photo, Followers, About, Experience, Number of experience, Education, Number of education, License, Number of license, Volunteering , Number of volunteering, Skills, Number of skills, Recommendations, Number of recommendations, Projects, Number of projects, Publications, Number*

of publications, Courses, Number of courses, Honours, Number of Honours, Scores, Number of scores, Languages, Number of languages, Organizations, Number of organizations, Interests, Number of interests, Activities, Number of activities label ,About -length, Experience _length). [23]

#	Feature	Type	Description
1.	linkedin_id	Categorical	A unique identifier for the job posting on LinkedIn.
2.	position_id	Numerical	A unique identifier for the position
3.	position	Categorical	The job title or role being advertised (e.g., “Head of Branding Governance Unit”).
4.	company	Categorical	The organization offering the job
5.	location	Categorical	The city and country where the job is located (e.g., “Riyadh, Saudi Arabia”).
6.	level	Categorical	The seniority level or experience required for the position (e.g., “Mid-Senior level”, “Associate”).
7.	date	Categorical	The date when the job was posted or last updated.
8.	Job_function	Categorical	The primary job functions or areas of responsibility (e.g., “Marketing”, “Management”).
9.	industries	Categorical	The industry or sector related to the job (e.g., “Banking”, “Government Administration”).
10.	description	Categorical	The job description or details, including qualifications, responsibilities, and other information about the role.

Table 4: Job Dataset

#	Feature	Type	Description
1.	Intro	Categorical	Contains structured data about the individual's introduction, such as their full name and workplace.
2.	Full Name	Categorical	The complete name of the individual (e.g., "NEHA CHANDOK").
3.	Workplace	Categorical	The organization or company where the individual is employed or affiliated with (e.g., "Software Analyst at Infosys").
4.	Location	Categorical	The geographical location of the individual, including city and country (e.g., "Wuxi, Jiangsu, China").
5.	Connections	Numerical	The number of professional connections or contacts the individual has.
6.	Photo	Categorical	Indicates whether the individual has a profile photo ("Yes" or "No").
7.	Followers	Numerical	The number of people following the individual's profile or activity.
8.	About	Categorical	A brief personal or professional summary provided by the individual.
9.	Experiences	Categorical	Structured data about the individual's professional experiences, including roles and workplaces.
10.	Number of Experiences	Numerical	The total count of professional experiences listed.
11.	Educations	Categorical	Structured data about the individual's educational background, including institutions attended and degrees obtained.
12.	Number of Educations	Numerical	The total count of educational entries listed.
13.	Licenses	Categorical	Structured data about the certifications or licenses the individual holds (e.g., "Six Sigma Yellow, Green Belt").
14.	Number of Licenses	Numerical	The total count of licenses or certifications listed for the individual.
15.	Volunteering	Categorical	Structured data about the individual's volunteering activities or roles.
16.	Number of Volunteering	Numerical	The total count of volunteering experiences listed for the individual.
17.	Skills	Categorical	A list of skills the individual has mentioned (e.g., "Management," "Microsoft Excel").
18.	Number of Skills	Numerical	The total count of skills listed by the individual.
20.	Recommendations	Categorical	Structured data about any professional recommendations the individual has received.
21.	Number of Recommendations	Numerical	The total count of professional recommendations the individual has received.
22.	Projects	Categorical	Structured data about any professional or academic projects the individual has worked on.
23.	Number of Projects	Numerical	The total count of projects listed for the individual.
24.	Publications	Categorical	Structured data about publications authored by the individual
25.	Number of Publications	Numerical	The total count of publications authored by the individual.
26.	Courses	Categorical	Structured data about the courses an individual has taken (e.g., topics, certifications)
27.	Number of Courses	Numerical	The total count of courses listed for the individual.
28.	Honors	Categorical	Structured data about the honors or awards received by the individual.
29.	Number of Honors	Numerical	The total count of honors or awards listed for the individual.
30.	Scores	Categorical	Structured data about test scores or evaluations associated with the individual (e.g., "Test: Leaving cert," "Score: 85").
31.	Number of Scores	Numerical	The total count of test scores listed for the individual.
32.	Languages	Categorical	Structured data about the languages known by the individual, possibly including proficiency levels or certifications (e.g., "Language 0: Bangla," "Language 1: English").

33.	Number of Languages	Numerical	The total count of languages listed for the individual.
34.	Organizations	Categorical	Structured data about organizations the individual is associated with (e.g., memberships or affiliations).
35.	Number of Organizations	Numerical	The total count of organizations listed for the individual.
36.	Interests	Numerical	Structured data about the individual's interests, which may include specific topics, organizations, or public figures (e.g., "Trina Solar," "Mohamed El-Erian").
37.	Number of Interests	Categorical	The total count of interests listed for the individual.
38.	Activities	Categorical	Structured data about the individual's activities, which may include events, collaborations, or engagements
39.	Number of Activities	Numerical	The total count of activities listed for the individual.
40.	Label	Numerical	Legitimate LinkedIn Profiles (LLPs)

Table 5: Candidate Dataset

3.2. Exploratory data analysis (EDA)

It's an essential step in the process of analyzing data. It allows us to understand and explore data before applying data preprocessing. After preprocessing, it ensures that the data is clean, well-structured, and ready for advanced analysis or modeling techniques. It helps us with identifying patterns and hidden patterns and understanding the structure of the data before applying any transformations or preprocessing techniques. Our objective of applying EDA is to gain insights about the dataset to guide us about how to clean the data, select the key features in the dataset, and model the data to understand the relationship between variables and features that may influence the analysis.

In our recruitment recommendation system, we applied an EDA to the dataset to understand its features and identify any issues that could affect the quality of the data. By combining insights gathered before and after preprocessing, we ensured that the dataset was prepared effectively to support the recommendation system. This helped make the process of recommending a job for candidates more organized. We applied the three types of analysis: Univariate Analysis, Bivariate Analysis, and Multivariate Analysis. Each analysis we applied helped in looking at different aspects of the dataset.

Type of analysis	**Definition**
Univariate Analysis	It’s the simplest type of analysis because it analyzes one variable at a time. We applied this type of analysis to examine the distribution of a single variable to understand its characteristics and describe the basic features such as mean and median and show the shape of the distribution if its normal, skewed left or right. But it does not consider the relationship between variables because it examines one variable at a time.

Bivariate Analysis	This type of analysis examines the relationship between two variables. Seeing if a change one variable can lead to change in another variable and reveal trend and correlations. We applied bivariate analysis to examine how variables like Number of Experience and Number of Skills are related
Multivariate Analysis	This type of analysis analyzes the relationship between three or more variables simultaneously. This will help us to uncover hidden patterns and correlations that may not be shown when examining individual pairs of variables

Table 6: Type of analysis

3.2.1. Exploratory data analysis (EDA) of data before data preprocessing:

To initiate our Exploratory Data Analysis prior to preprocessing, we utilize visualizations to thoroughly explore the dataset's structure, identify meaningful patterns, and detect any anomalies or issues that may impact subsequent steps.

- **Univariate Analysis:**

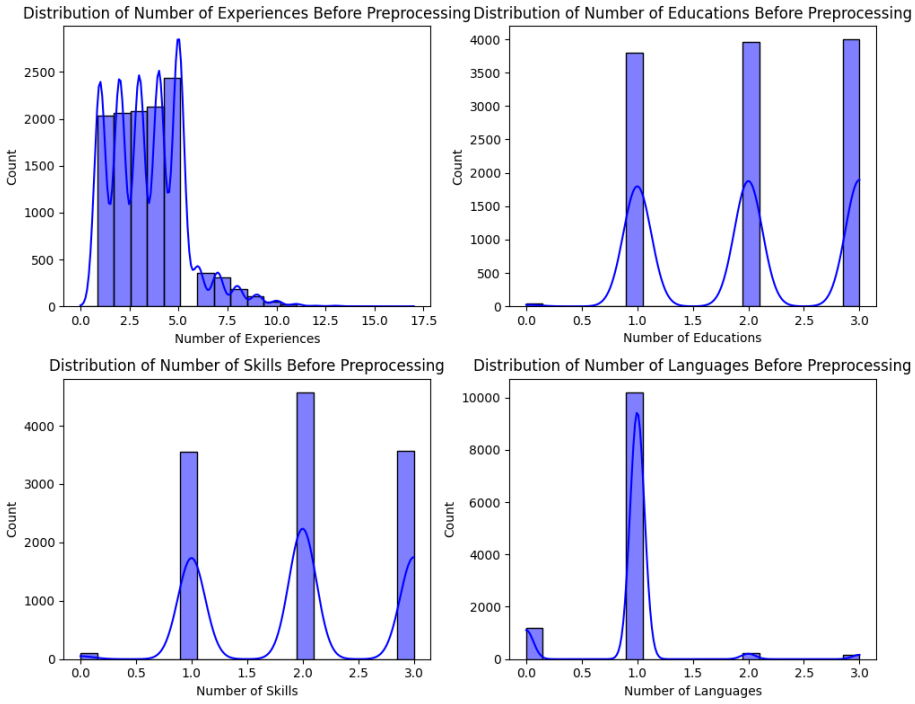

Figure 3:Plot numeric column distributions of the Candidate Data set before data preprocessing

Distribution of numerical features:

The distribution of the *Number of Experience* is skewed to the right, indicating that most candidates have less than five experiences listed in their profiles. The *Number of Education* distribution is bimodal, with most users having either 1 or 2 educational qualifications, while a minority have no formal education. The distribution of the *Number of Skills* is discreet, with peaks at 1, 2, and 3 skills, and some profiles might lack detailed skill information. Lastly, the distribution of the *Number of Languages* is highly concentrated around 1 language, indicating that most users are monolingual.

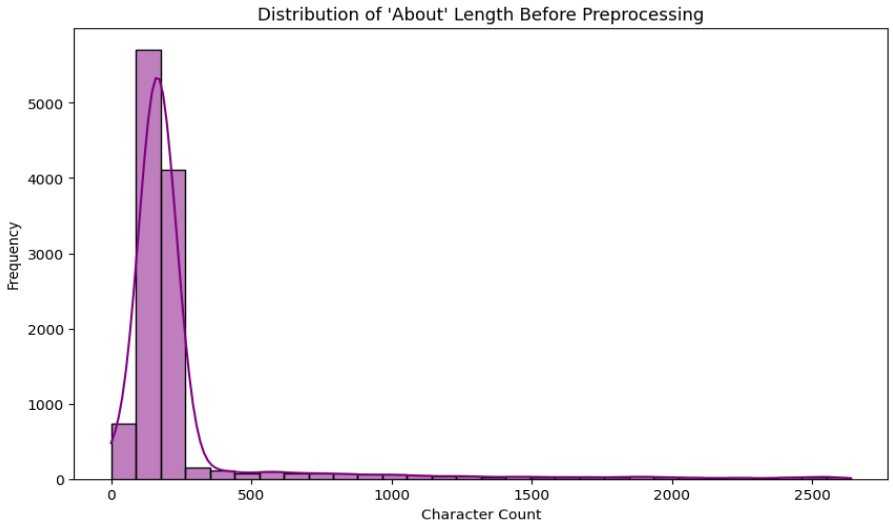

Figure 4:Plot the text length distributions for the 'About' column of the Candidate Dataset before data preprocessing.

Text Length Distributions:

The text length for 'About' is positively skewed, with many profiles having minimal or no summary. A few profiles contain long descriptions.

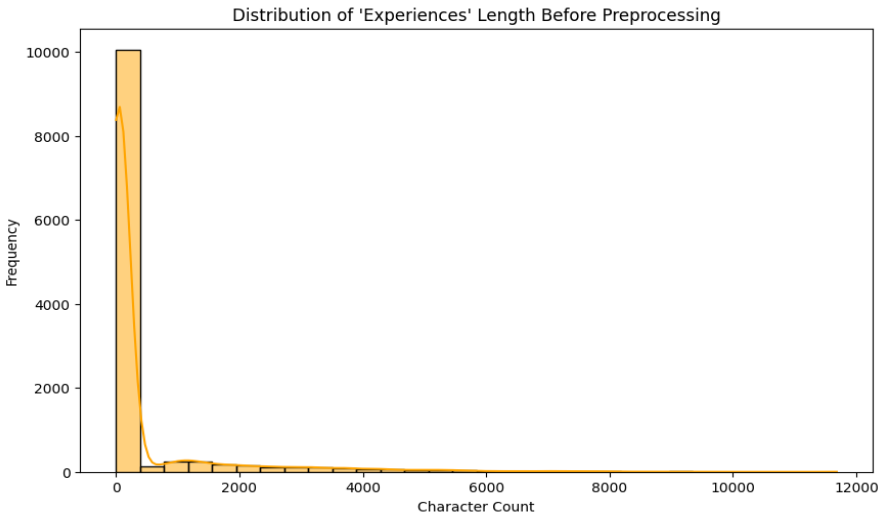

Figure 5: Plot the text length distributions for the 'Experiences' column of the Candidate Dataset before data preprocessing.

Experiences' Length Distribution:

Like 'About', 'Experiences' length shows skewness with a majority having brief descriptions.

- **Bivariate Analysis**

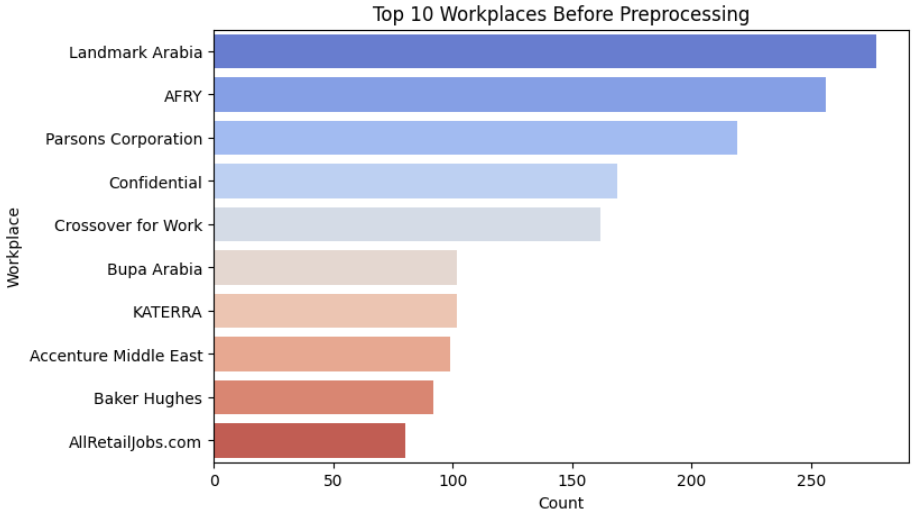

Figure 6:Top 10 workplaces in the 'Workplace' column of the Candidate Dataset before data preprocessing.

Analysis of the Top 10 Workplaces:

The graph shoes that Landmark Arabia and AFRY dominate the top workplaces, and other *workplaces* like "Confidential" may need additional data enrichment for better insights.

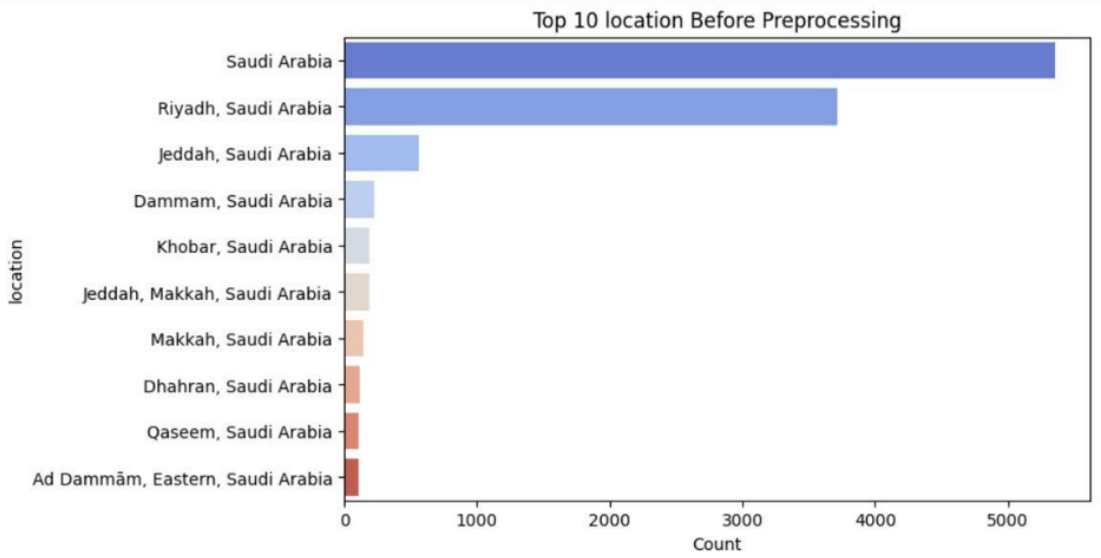

Figure 7: Top 10 location in the "location" column of the Job Dataset before data preprocessing.

Analysis of the Top 10 Location:

This graph shows the top 10 *locations* mentioned in a dataset before any data cleaning or preprocessing which the name of the city is next to the country name.

- **Multivariate Analysis:**

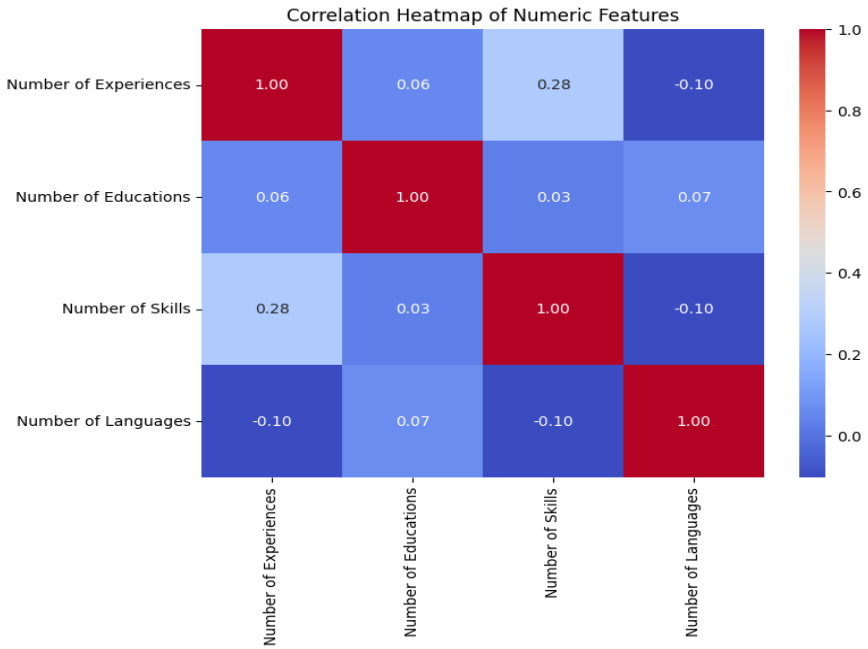

Figure 8: Correlation matrix of numeric columns of the Candidate Data set before data preprocessing.

Correlation Heatmap of numerical features:

The graph shows weak correlations are observed between numeric features, such as between *Number of Educations* and *Number of Skills*

3.2.2. Exploratory data analysis (EDA) of data after data preprocessing:

After preprocessing, we proceed with further Exploratory Data Analysis to evaluate the effectiveness of the data cleaning process, refine our insights, and ensure the dataset is well-prepared for modeling.

- **Univariate Analysis:**

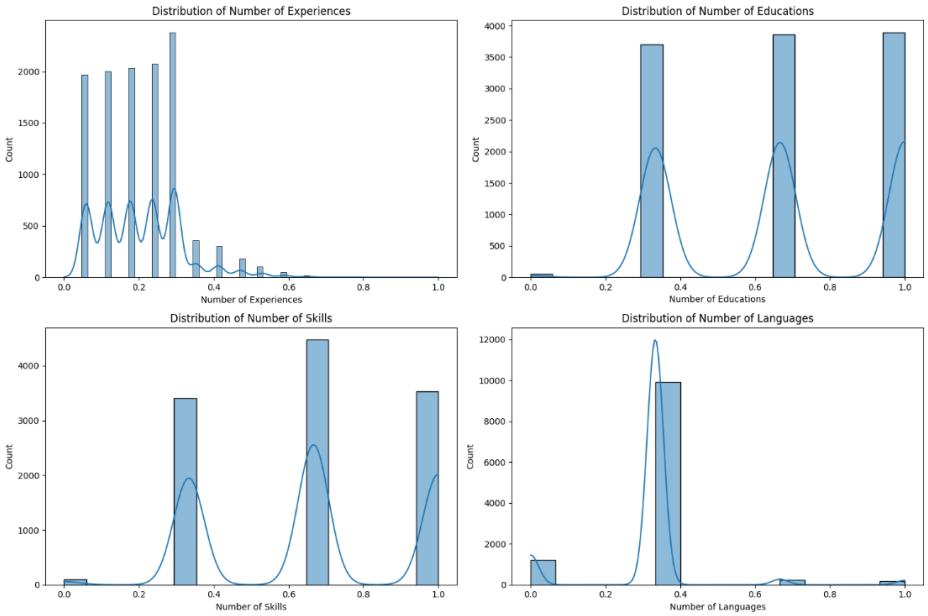

Figure 9: Plot numeric column distributions of the Candidate Data set after data preprocessing

Distributions of the numerical features:

After preprocessing, the data reveals several key patterns: the *number of experiences* shows multiple peaks between 0 and 7.5, with most individuals having 1 to 5 experiences; the *number of education* entries displays three clear peaks around 1 and 2, indicating that most people have one or two educational qualifications; the *number of skills* exhibits three main clusters at 1, 2, and 3 skills; and the *number of languages* has a dominant peak around 1, with very few people listing multiple languages. These distributions reflect the diversity and sparsity of the data, and the normalization has effectively scaled the features for fair comparison across users.

- **Bivariate Analysis**

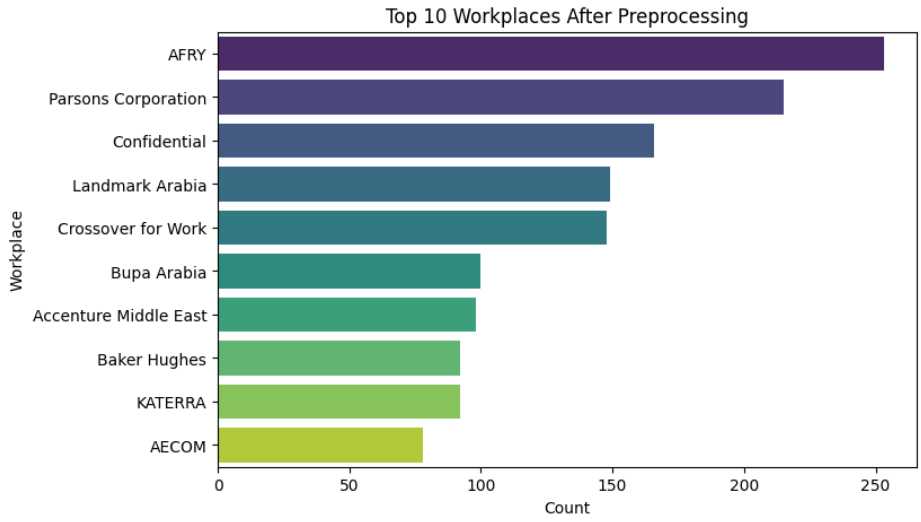

Figure 10: Top 10 workplaces in the 'Workplace' column of the Candidate Dataset after data preprocessing.

Analysis of top 10 workplaces:

The top *workplaces* have been extracted, showcasing companies such as AFRY, Parsons Corporation, and Confidential. The distribution is cleaner and does not show dominance by generic labels like "Confidential," which were likely corrected during pre-processing. Cleaning workplace names enhanced the usability of this feature for clustering or trend analysis in job recommendations.

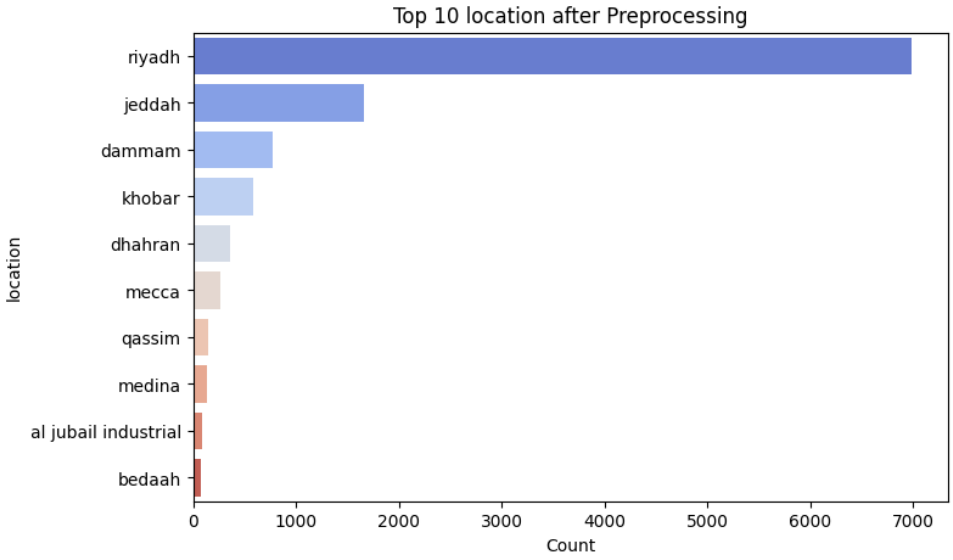

Figure 11: Top 10 location in the "location" column of the Job Dataset after data preprocessing.

Analysis of top 10 locations:

This graph shows the *location* column after applying preprocessing which the country name got deleted next to the city name to make the data in this column more understandable cause our goal only limited to the country. We can observe that the city Riyadh is the highest among other cities with a count around 7000, and the Less frequent locations include bedaah with counts below 100.

- **Multivariate Analysis:**

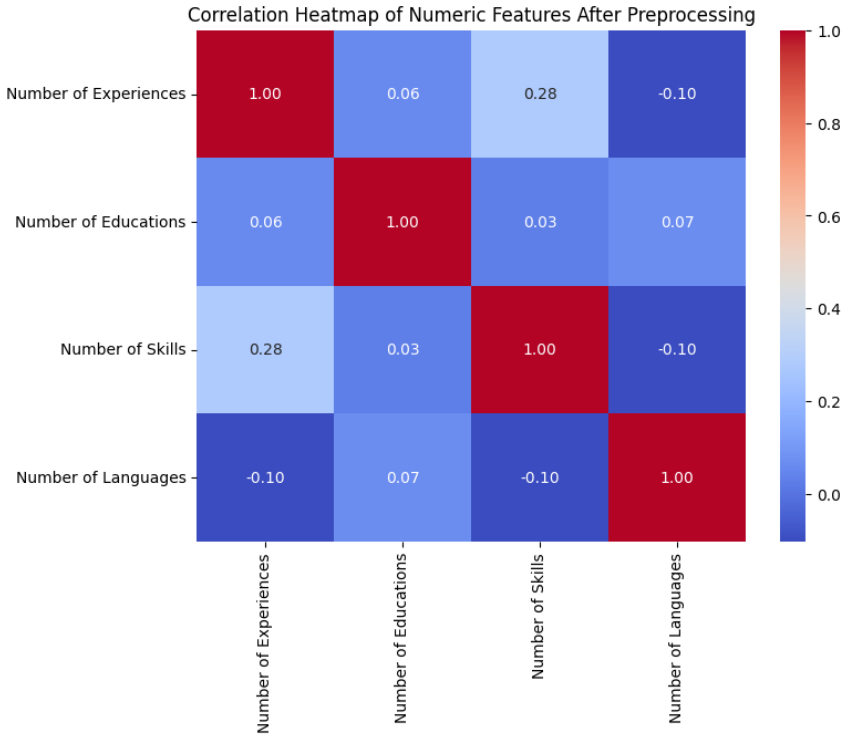

Figure 12: Correlation matrix of numeric columns of the Candidate Data set after data preprocessing.

Correlation Heatmap of numerical features:

The *number of Experiences* has a weak positive correlation with *Number of Skills* (0.28), the *number of Educations* and *number of Experiences* show almost no correlation (0.06), and the *number of Languages* has no significant correlation with any other features. The weak correlations indicate that these features independently contribute to the matching algorithm, reinforcing the need to weigh them separately during model design.

3.3. Preprocessing

Data preprocessing is the process of cleaning, organizing, and transforming raw data into a format that can be easily analyzed. It involves handling missing values, removing errors or inconsistencies, normalizing or scaling data, and converting it into a structure suitable for analysis or machine learning. This step is crucial because real-world data inconsistent, incomplete, or unstructured, and preprocessing ensures that the data is accurate, consistent, and ready to be trained for the model.

3.3.1. Candidate dataset Data Preprocessing

The steps in data preprocessing in the **candidate dataset**:

1. **Drop Unnecessary Columns:**

   Some columns are not relevant to the job matching task. These columns were dropped to avoid cluster.

2. **Dictionary cleaning:**

   To ensure that there's no empty entries in the dataset, these type values can make a lot of errors during the analysis step. The data was written in a dictionary so we used this cleaning techniques to transform messy LinkedIn data into clean, usable text. The clean_experience_text function takes work experience entries (stored as dictionaries) and converts them into readable text like "Software Engineer at Google (2 years)". The clean_dict_text function handles all other columns, turning complex dictionary structures into simple text. We apply these cleaners to six key columns (*Workplace, About, Experiences, Educations, Skills, Languages*) to make the data consistent and searchable for our job matching system.

3. **Handle Missing values:**

   Missing values is a significant issue that needs to be addressed. To identify and handle them effectively, we performed a thorough check of the dataset and obtained the following results:

   About column: 362 missing entries.

   Blank fields: Some fields appeared blank but were empty strings (''), empty lists ([]), or empty dictionaries ().

   Various missing values across different columns:

   - **Experiences**: 22 missing values.
   - **Education**: 50 missing values.
   - **Skills**: 102 missing values.
   - **Languages**: 1200 missing values.

```
final Null counts:
Workplace                 0
About                   362
Experiences              22
Number of Experiences     0
Educations               50
Number of Educations      0
Skills                  493
Number of Skills          0
Languages              1200
Number of Languages       0
```

Figure 13:Missing values in the Candidate Dataset

Approach to Handle Missing Values

After identifying the missing values, we developed a function to standardize all types of empty values (empty strings, empty dictionaries, empty lists) to proper Nan values. This gave us a clearer picture of the truly missing data. We then implemented the solution in three main phases:

Clean-up Phase:

We created a function to standardize all types of empty values (empty strings, empty dictionaries, empty lists) to Nan values. This transformation helped us accurately identify and handle missing data.

Smart Filling Phase:

Workplace: Extracted missing information from the experience column or used the default value "Open to Opportunities" if unavailable.

Experiences: Used workplace data to create a basic entry for missing experience values.

Skills: Developed a sophisticated system to infer missing skills based on job titles and experience.

Languages: Defaulted to "English" if language data was missing.

Education: Inferred the missing education data based on the experience level.

Numeric columns: Calculated missing numeric values using related text fields.

About Section Generation:

For the 362 missing entries in the **About** section, we generated summaries using available information from other columns, combining data from workplace, experience, education, skills, and languages into coherent, meaningful summaries.

By following this structured approach, we ensured that the dataset was complete and meaningful, while maintaining its integrity and professional relevance.

4. Checking for duplicate values:

Duplicates are a significant issue in the dataset that can lead to false results such as skewing results and analysis. And this step is essential as duplicate profiles can introduce bias and inefficiencies in job recommendations.

We identified and removed exactly 289 duplicate profiles by comparing identical values across the *Workplace, About, Experiences, Number of Experiences, Educations, Number of Educations, Number of Skills, Languages, and Number of Languages* columns. Importantly, even when we excluded the Skills column from duplicate detection due to its formatting issues, we still found the same 289 duplicates. This confirmed that the exclusion of the Skills column did not impact on our ability to detect duplicate.

5. **Numeric Feature Scaling:**

It is important to scale numeric data to ensure that all features are on the same scale cause the larger scale can dominate the calculations. We applied numeric feature scaling to enhance the job matching process and operate it effectively.

This approach enables our matching algorithm to fairly compare these different features, ensuring that each factor is weighted appropriately, regardless of its original scale.

3.3.2. Job Dataset Preprocessing

The steps in data preprocessing **job dataset**:

1. **Drop unnecessary columns:**

The columns *'date'* and *'linkedin_id'* should be dropped from the job dataset for the following reasons: The *'date'* column is not relevant for job matching analysis since all jobs are from 2020, making temporal information unnecessary and only adding noise to the dataset. Similarly, the *'linkedin_id'* column is an internal identifier that does not provide any meaningful insights into job requirements or skills. It is not needed for analysis and could potentially cause confusion. Removing these columns will streamline the dataset and focus on the relevant information for job matching.

2. **Dictionary cleaning**

We implemented a data cleaning function to transform list-formatted strings in our LinkedIn dataset into clean text. For example, ['Human Resources'] became "Human Resources" and ['Sales', 'Business Development'] became "Sales Business Development". This transformation removed unnecessary syntax (brackets and quotes) while preserving the content, making the data more consistent and easier to process for text analysis and matching algorithms. The cleaning was applied across all columns to ensure uniform data formatting while maintaining information integrity.

3. Handle missing values:

We have no missing values in the dataset, as we ensured this by only loading and preprocessing the data through specific steps: reading the Excel file, applying dropna() to eliminate any rows containing missing values, and selecting the first 11,511 rows using head(11511) to match the size of the another dataset.

	0
position_id	0
position	0
company	0
location	0
level	0
job_functions	0
industries	0
description	0

Figure 14: Missing values in the Job Dataset

4. Duplicate checking

There's no duplicates in the job dataset.

3.3.2 Model Preprocessing

Preprocessing is a fundamental stage in building effective job recommendation models, ensuring that raw data—whether candidate profiles or job descriptions—is clean, consistent, and enriched for subsequent analysis. This section consolidates and highlights the preprocessing strategies employed across our candidate models **User-Based Collaborative Filtering with Machine Learning**, **Text-Based Content Filtering with NLP Techniques**, **Semantic AI Job Matching with BIRCH and FAISS (SAJM-BF)**, and the **Proposed Model: Advanced Hybrid Filtering with NLP and Personalized Matching**.

While these models share core preprocessing techniques, such as text normalization, tokenization, and lemmatization, each incorporates unique adaptations to address specific design goals. This chapter details the unified preprocessing pipeline and emphasizes the model-specific customizations applied to meet the individual requirements of each approach.

Unified Preprocessing Pipeline

The preprocessing pipeline ensures that textual and numerical data are cleaned, standardized, and prepared for feature extraction. The following steps form the backbone of the preprocessing process, applied consistently across all models:

1. **Text Normalization**

   Textual data is converted to lowercase to ensure uniformity and reduce variability caused by inconsistent capitalization. Irrelevant elements, such as special characters, numbers, email addresses, and extra whitespace, are removed to reduce noise and improve readability. This step creates a standardized foundation for further analysis.

2. **Tokenization**

   Text is segmented into smaller units, such as words or tokens, to facilitate granular analysis. By breaking down sentences into individual components, tokenization enables the models to process and analyze text more effectively at the word or phrase level.

3. **Stopword Removal**

   Commonly occurring but uninformative words, such as "the," "is," and "and," are removed to retain only meaningful content. Additionally, domain-specific stopwords, like "etc" or "eg," are excluded to further refine the text and reduce irrelevant noise.

Model-Specific Adaptations in Preprocessing

While we maintained consistent core preprocessing steps like cleaning and normalization across all models, each approach required specific adaptations to address its unique requirements:

- **User-Based Collaborative Filtering**: We standardized candidate profiles and job descriptions to ensure clean and consistent data. This prepared the dataset for similarity computations, enabling robust analysis of user preferences and behaviors.
- **Text-Based Content Filtering**: For this model, we focused on removing irrelevant elements such as URLs, special characters, and domain-specific terms. These steps enhanced the model's ability to perform keyword-based matching and semantic analysis effectively.
- **Semantic AI Job Matching**: Missing values were systematically addressed to preserve dataset integrity, and numerical fields were scaled using Min-Max normalization. These enhancements optimized the data for clustering and fast retrieval, ensuring efficiency in handling complex datasets.
- **Proposed Hybrid Model**: We applied Named Entity Recognition (NER) to extract and retain critical attributes like skills, organizations, and languages. These attributes were aggregated into unified fields such as "account" and "job_posting," providing a structured representation for personalized and effective matching.

These preprocessing strategies not only enhance the quality of data but also prepare it for advanced feature extraction, similarity computations, and recommendation generation. Each model's preprocessing pipeline is carefully designed to address its unique goals, ensuring robust and accurate performance in diverse job matching scenarios.

3.4. Summary

In this chapter, we discussed the sources of the datasets, provided an explanation of the dataset features, and provided a detailed description of the preprocessing steps. Additionally, we presented the Exploratory Data Analysis (EDA) conducted both before and after preprocessing. By addressing issues such as missing values, duplicates, and inconsistencies, as well as applying text preprocessing techniques like tokenization, text normalization, stop word removal, and lemmatization, we were able to clean and organize the data, making it ready for the next step. These efforts ensure that the recommendation system will be more accurate and efficient in matching candidates with the right job opportunities.

4. Model planning

In this chapter we present the implementation of the models introduced in Chapter 2, focusing on their adaptation for a comprehensive recruitment recommendation system. This chapter is dedicated to the detailed realization of four distinct models: User-Based Collaborative Filtering with Machine Learning, Text-Based Content Filtering with NLP Techniques, Semantic AI Job Matching with BIRCH and FAISS, and the Proposed Model, Advanced Hybrid Filtering with NLP, and machine learning.

The chapter begins by the application of advanced feature engineering techniques, leveraging embeddings like TF-IDF, BERT, and Word2Vec to create robust numerical representations of text. For models utilizing clustering, methods like BIRCH are introduced to group similar candidates, optimizing computational performance.

A critical component of this chapter is the similarity computation process, where sophisticated techniques such as cosine similarity are employed to measure the alignment between candidate profiles and job postings. Finally, the chapter culminates in the recommendation generation, showcasing how each model produces actionable insights and tailored outputs that align candidates with job opportunities.

4.1. User-Based Collaborative Filtering with Machine Learning

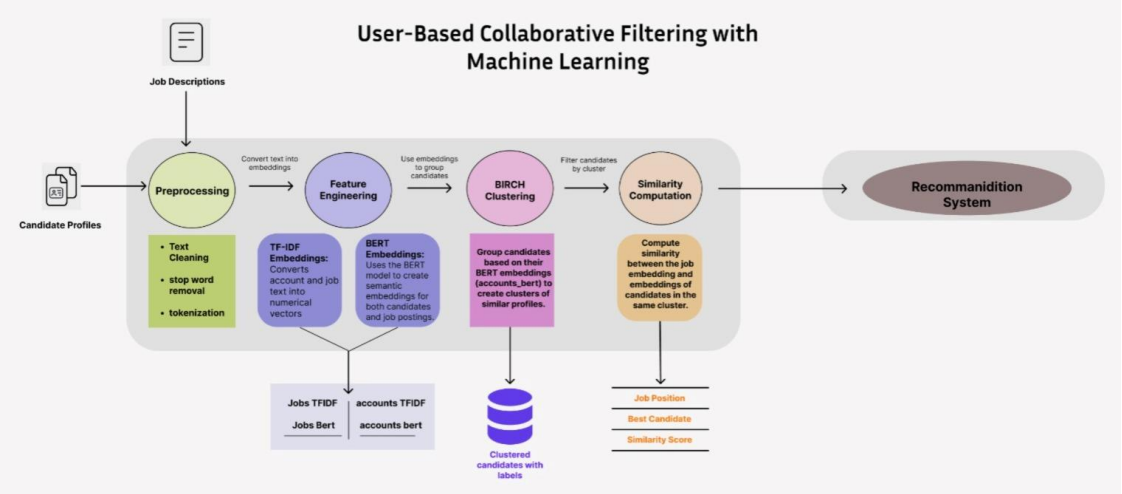

Figure 15: User-based collaborative filtering using machine learning architecture

User-Based Collaborative Filtering (UBCF) with Machine Learning, this model is particularly suitable for a job recommendation system, as it focuses on leveraging the similarities between user (candidate) profiles and job descriptions to provide tailored recommendations. The strength of UBCF lies in its ability to identify patterns in candidate profiles and job requirements to match candidates effectively with relevant job postings. To enhance the accuracy of recommendations and address challenges such as sparse data or contextual mismatches, we incorporate advanced techniques like feature engineering, clustering, and semantic embeddings.

The implementation process of this model involves four major stages: **feature engineering**, **clustering**, **similarity computation**, and **model evaluation**. Each stage plays a critical role in transforming raw input data into actionable insights that drive the recommendation system.

4.1.1. Feature Engineering

Feature engineering is a critical stage where the cleaned textual data is transformed into numerical representations that can be analyzed computationally. This step is pivotal to capturing the essence of both candidate profiles and job postings, allowing for meaningful similarity comparisons. In this project, we utilize two distinct embedding techniques to generate these numerical representations: TF-IDF vectorization and MiniLM-L6-v2 embeddings.

TF-IDF (Term Frequency-Inverse Document Frequency) is used to measure the importance of each word in a document relative to its frequency across all documents. This method captures word-level relevance and produces sparse numerical vectors for both candidate profiles and job descriptions. These vectors are efficient for quick calculations, making TF-IDF suitable for preliminary similarity assessments.

Complementing TF-IDF**, All-MiniLM-L6-v2 embeddings** (Minimal Language Model) embeddings are employed to capture the semantic context and relationships within the textual data. By leveraging the pre-trained "paraphrase-MiniLM-L6-v2" model, we generate dense embeddings for candidate profiles and job descriptions, it provides a deep contextual understanding of the text, enabling the model to comprehend nuanced relationships between candidates and jobs.

The outputs of feature engineering—TF-IDF vectors (accounts_tfidf and jobs_tfidf) and BERT embeddings (accounts_bert and jobs_bert)—serve as the foundation for clustering and similarity computations. These two embedding techniques ensure that the model considers both lexical and contextual similarities, enhancing the robustness of the recommendation system.

4.1.2. Clustering

Given the scale and complexity of the datasets, clustering is applied to optimize the recommendation process by grouping similar candidates based on their profile embeddings. For this purpose, we employ **the BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) algorithm**, which is well-suited for handling large datasets efficiently.

Using the MiniLM-L6-v2 embeddings of candidate profiles (accounts_bert), the BIRCH model identifies groups of candidates with similar skills, experiences, and qualifications. Each candidate is assigned a cluster label, which facilitates narrowing down the search space for similarity computations. By focusing on candidates within the same cluster as a job posting, the system reduces computational overhead while maintaining accuracy. This step ensures that the recommendation process is both efficient and scalable.

To further validate the system and enable predictive capabilities, a classification model is trained. The MiniLM-L6-v2 embeddings of candidate profiles (accounts_bert) serve as the input features, while the target labels are simulated based on predefined criteria (e.g., matched or unmatched). A logistic regression model is used for classification, leveraging the embeddings to predict candidate-job compatibility.

4.1.3. Similarity Computation

The similarity computation stage lies at the core of the recommendation system, where job postings are matched with candidates from the relevant clusters. For each job posting, the system retrieves the cluster label predicted by the BIRCH model and filters the candidates to include only those within the same cluster. This step significantly reduces the number of candidates for consideration for each job, improving the system's performance.

Within the identified cluster, the **cosine similarity** metric is used to calculate the similarity between the job's embedding (jobs_bert) and the embedding of candidates in that cluster (accounts_bert). Cosine similarity evaluates the alignment of vectors in a multi-dimensional space, producing a score between 0 and 1, where 1 indicates a perfect match. The candidate with the highest similarity score is identified as the best match for the job. In cases where no candidates exist in the identified cluster, the system returns "No Match Found."

The output of this stage is a ranked list of candidates for each job posting, complete with similarity scores. These ranking forms the basis for the final recommendation output.

4.1.4. Final Recommendation System

The recommendation system integrates all the components described above into a seamless workflow. Preprocessed data is transformed into embeddings during the feature engineering stage, clustered using BIRCH, and compared with job postings through cosine similarity. The output is a ranked list of candidates

for each job, providing actionable recommendations based on both lexical and semantic analyses. This output will provide actionable insights by presenting a ranked list of candidates for each job.

4.2. Text-Based Content Filtering with NLP Techniques

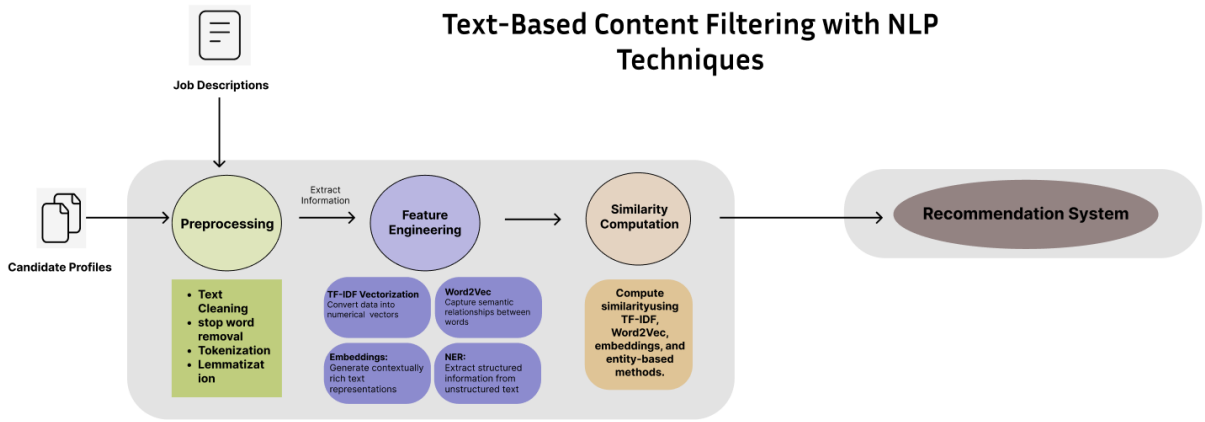

Figure 16: Text-Based Content Filtering with NLP Techniques architecture

Text-Based Content Filtering model, which uses advanced Natural Language Processing (NLP) techniques to align candidate profiles with job postings. As introduced in Chapter 2, this model focuses on extracting meaningful information from textual data to generate accurate and relevant recommendations. Unlike traditional methods, this model not only considers explicit matches, such as keywords, but also captures implicit relationships and contextual meaning, making it a powerful tool for a recruitment recommendation system.

The implementation process involves transforming raw textual data into structured, numerical representations and then using these representations to compute similarity scores between profiles and job descriptions. By combining multiple embedding techniques, including TF-IDF, Word2Vec, All-MiniLM-L6-v2 embedding, and Named entity recognition, the model ensures that both lexical relevance and semantic meaning are accounted for, providing a comprehensive approach to matching. The process is divided into three stages: **feature engineering**, **similarity computation**, and **recommendation generation**. Each stage is meticulously crafted to address the nuances of textual data and deliver actionable insights.

4.2.1. Feature Engineering

Feature engineering is the stage where the cleaned textual data is converted into numerical formats that allow computational comparison between profiles and job descriptions. This transformation is essential for the model's ability to assess alignment effectively. Four techniques are employed to capture different aspects of the data: TF-IDF, Word2Vec, All-MiniLM-L6-v2 embeddings, and Named entity recognition.

TF-IDF (Term Frequency-Inverse Document Frequency) is used to measure the importance of words in a document relative to their occurrence across the entire dataset. This method highlights terms that are unique to a particular document, making it effective for identifying specific skills or qualifications mentioned in job postings or profiles. The model uses unigrams and bigrams (single words and word pairs) to capture both individual terms and contextual relationships. To reduce noise, common and domain-specific stopwords are excluded, and the vocabulary size is limited to 1,000 terms, focusing on the most relevant information.

Word2Vec embeddings provide a complementary approach by capturing semantic relationships between words. Unlike TF-IDF, which considers word frequency, Word2Vec learns word associations based on their co-occurrence in the dataset. By using a skip-gram model, it predicts the context of a word, enabling it to understand relationships like "Python" and "Django" or "Data Science" and "Machine Learning." The model generates 300-dimensional vectors for each word, balancing detail with computational efficiency.

The **All-MiniLM-L6-v2 embeddings** come from the MiniLM (Minimal Language Model), a fast transformer model optimized for efficiency while maintaining strong performance. It captures the meaning and relationships between words by considering their context within a sentence. During processing, text is broken into smaller units called subwords, which ensures even uncommon or technical terms are represented accurately. The sequences are then padded and truncated to a uniform length for consistency and processed in batches, enabling efficient and scalable computation.

Named Entity Recognition (NER) is used to identify specific entities such as skills, certifications, locations, or industries within the text. By extracting these entities, the model gains a structured understanding of key attributes in profiles and job descriptions, which is critical for matching on a granular level. For instance, it can identify "Python" as a skill or "Riyadh" as a location, providing additional layers of alignment.

The outputs of these techniques—TF-IDF vectors, Word2Vec embeddings, All-MiniLM-L6-v2 embeddings, and Named entity recognition—form the backbone of the recommendation system. By combining these representations, the model captures both lexical matches and semantic context, ensuring a robust evaluation of profile-job alignment.

4.2.2. Similarity Computation

Similarity computation is the core of the model, where candidate profiles and job postings are compared to assess how well they align. This stage leverages the numerical representations generated during feature engineering to calculate similarity scores using multiple methods.

The first method involves **TF-IDF similarity**, where cosine similarity is computed between the TF-IDF vectors of profiles and job postings. This approach identifies explicit matches, such as specific keywords mentioned in both the profile and the job description. For example, if a job posting requires "Python programming" and a profile explicitly mentions "Python," the TF-IDF similarity score will be high.

Next, **Word2Vec similarity** is used to capture implicit relationships between terms. By calculating cosine similarity between Word2Vec embeddings, the model can identify connections that are not explicitly stated but are semantically related. For instance, it can recognize that "data analysis" is related to "business intelligence."

then, **All-MiniLM-L6-v2 embedding similarity** computes cosine similarity between the BERT embeddings of profiles and job postings. This method excels at capturing subtle and complex relationships, ensuring that the system understands not just the words but the context and intent behind them.

Finally, **entity-based similarity** compares structured attributes like skills, and locations. Using a Jaccard-like approach, the calculate_entity_similarity function measures the overlap between entity sets in profiles and job postings, accounting for missing entities. For example, it identifies matches between "machine learning" and related skills. The entity_similarity_matrix function extends this by creating similarity scores for all candidate-job pairs, enabling multi-dimensional comparisons. Combined with text-based methods, this approach ensures robust and comprehensive alignment, enhancing the recruitment process.

4.2.3. Final recommendation system

The final stage of the model involves aggregating the similarity scores from these methods into a final metric that reflects the overall alignment between a profile and a job posting to generate actionable recommendations based on the computed similarity scores. For each job posting, candidate profiles are ranked according to their aggregated similarity scores. The highest-ranked profiles represent the best matches, providing recruiters with a clear and prioritized list of potential candidates.

The system also provides transparency by including matched keywords or entities in the recommendation output. For example, a recommendation might highlight that a candidate scored 85% in terms of skills alignment and explicitly matches certifications mentioned in the job description. This transparent and comprehensive approach builds trust in the system's recommendations and ensures that recruiters can make informed decisions.

4.3. Semantic AI Job Matching with BIRCH and FAISS

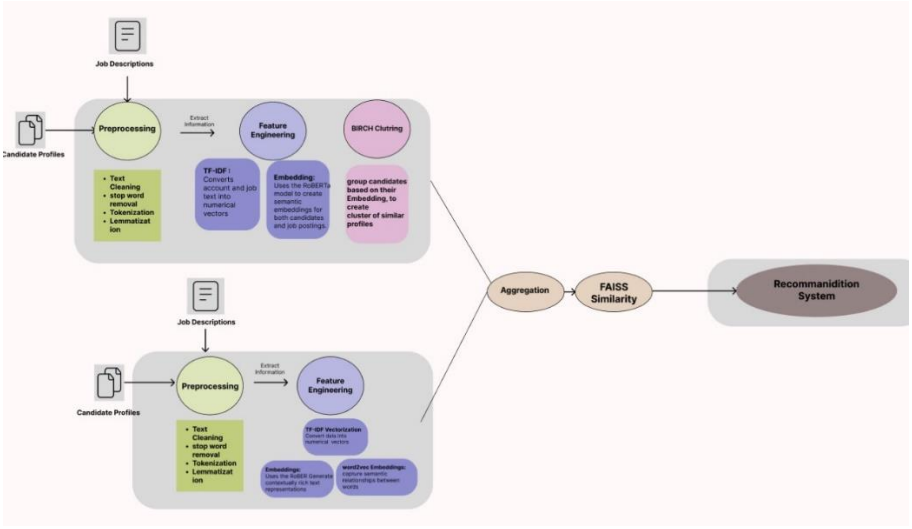

Figure 17: Semantic Al Job Matching with BIRCH and FAISS architecture

The Semantic AI Job Matching with BIRCH and FAISS or Advanced AI Recruiting Model (AIRM) it combines clustering, semantic embeddings, and efficient retrieval methods to deliver high-accuracy recommendations that align with the unique demands of Saudi Arabia's recruitment landscape. By leveraging advanced techniques such as BIRCH clustering, RoBERTa sentence transformers, and FAISS indexing, AIRM ensures rapid processing, scalability, and precise job-candidate alignment.

4.3.1. Feature Engineering:

Feature engineering is a crucial stage where pre-processed data is transformed into numerical representations for computational analysis.

- **Content-Based Filtering Feature Engineering:**

TF-IDF: Converts text into numerical vectors by weighing terms based on their frequency in a document relative to their occurrence across the entire corpus. This ensures unique and significant terms are highlighted.

Word2Vec: Learns word representations by predicting neighbouring words in a given context, enabling the capture of semantic relationships between words.

RoBERTa Embeddings: The RoBERTa transformer model creates dense semantic embeddings that capture nuanced relationships within the text. These embeddings are ideal for representing complex job and profile descriptions. Each candidate profile and job description are processed through the pre-trained RoBERTa model to create high-dimensional vector representations. These embeddings effectively encode explicit and implicit relationships, making them ideal for recruitment tasks

- **Collaborative Filtering Feature Engineering :**

TF-IDF: same process in content-based , it will Converts text into numerical vectors by weighing terms based on their frequency.

RoBERTa Embeddings: transformer model to represent candidate profiles and job descriptions as high-dimensional vectors.

Clustering Using the **BIRCH** (Balanced Iterative Reducing and Clustering using Hierarchies) algorithm, candidates are clustered into groups based on their RoBERTa embeddings. BIRCH efficiently identifies clusters of candidates with similar skills and experiences, significantly reducing computational overhead by narrowing the search space to relevant clusters.

4.3.2. Aggregation:

In this step, the results from Collaborative Filtering and Content-Based Filtering are combined into a single unified feature set. This step leverages the advantages of both methods.

4.3.3. Similarity Computation using FAISS:

Similarity computation is the core of the AIRM model, where candidate profiles and job postings are compared to evaluate their alignment. This stage utilizes the numerical embeddings generated during feature engineering and clustering to calculate similarity scores by using FAISS (Facebook AI Similarity Search) used for efficient similarity search and clustering of high-dimensional data.

Hybrid Similarity Score:

= (w1·TF-IDF Similarity+w2·BERT Similarity) \ text {Hybrid Similarity Score} Aggregate similarity scores from different feature types using weighted averages, the output is a hybrid similarity score matrix indicating candidate-job alignment.

4.3.4. Final recommendation system:

In this step involves generating actionable recommendations based on FAISS results, it will rank the candidates by their similarity scores for each job posting. This shows the workflow for this model, starting

with aggregating the Collaborative Filtering and Content-Based Filtering and then using FAISS to recommend the candidates with their similarity scores.

4.4. Advanced Hybrid Filtering with NLP and Machine learning

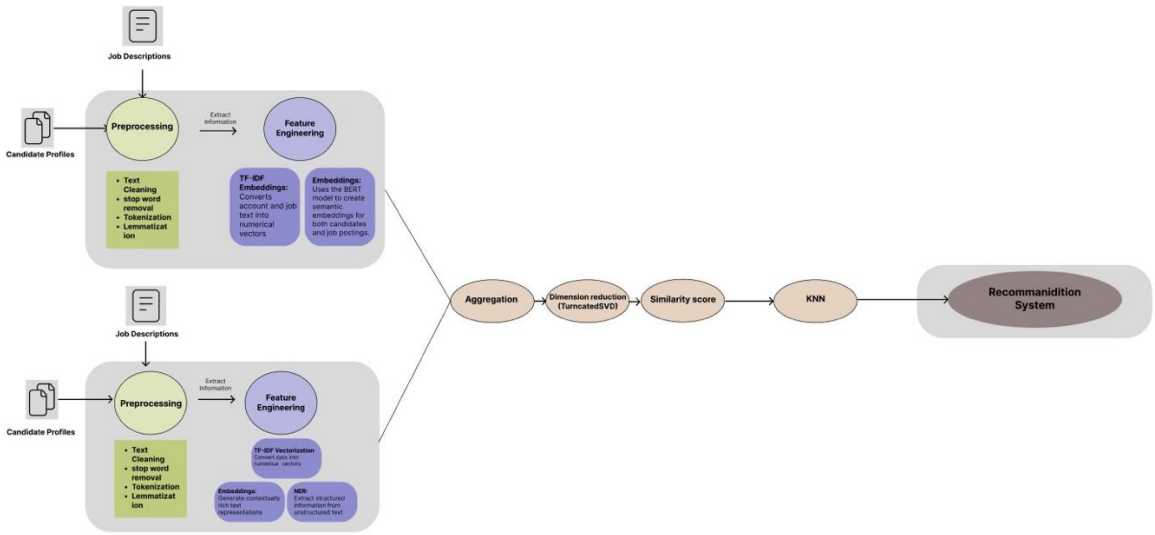

Figure 18: Advanced Hybrid Filtering with NLP and Personalized Matching architecture

The Proposed Model integrates two distinct filtering methods—Collaborative Filtering and Content-Based Filtering—to create a comprehensive job recommendation system. By aggregating features from both methods, followed by dimensionality reduction and similarity computation, the system delivers highly precise and efficient candidate-job matches. The integration of clustering ensures scalability for large datasets, while K-Nearest Neighbors (KNN) refines the recommendations within clusters.
This section describes the step-by-step implementation of the hybrid filtering model.

4.4.1. Feature Engineering

Feature engineering transforms preprocessed text into numerical representations. This section separates the Collaborative Filtering and Content-Based Filtering processes before combining their outputs.

- **Collaborative Filtering Feature Engineering**

To capture user-job relationships by encoding profiles and job postings using TF-IDF and All-MiniLM-L6-v2 embeddings.

TF-IDF Vectorization:

Converts text into sparse numerical vectors by measuring the importance of terms relative to their occurrence in the dataset. The output is a Sparse TF-IDF vectors for candidate profiles and job postings.

All-MiniLM-L6-v2 Embeddings:

Uses the all-MiniLM-L6-v2 pre-trained model to generate dense embeddings for text.

Embeddings capture semantic relationships, enabling the system to understand contextual nuances. The output Dense BERT embeddings for profiles (account_BERT) and jobs (job_BERT).

- **Content-Based Filtering Feature Engineering**

To enrich job-candidate alignment by leveraging domain-specific information and structured feature representations.

TF-IDF Vectorization:

Same process as in Collaborative Filtering but focused on the job descriptions and their explicit features. The output Sparse TF-IDF vectors for job postings.

4.4.2. Aggregation

The outputs from Collaborative Filtering and Content-Based Filtering are aggregated into a unified feature set. This step combines the strengths of both methods, ensuring that the system considers both lexical relevance and semantic understanding.

The process Merge TF-IDF vectors and the embeddings from both filtering methods into a single feature representation.

4.4.3. Dimensionality Reduction

To optimize computational efficiency, dimensionality reduction is applied to the aggregated features. The method used is **Truncated Singular Value Decomposition (SVD)** it reduces the combined feature matrix to a lower-dimensional space while retaining critical information. The outputs are compact feature representations (account_reduced, job_reduced) for candidates and job postings, respectively.

4.4.4. Similarity Computation

Similarity computation quantifies the alignment between candidate profiles and job postings.

Computing similarities between reduced candidate features and job posting features using Cosine Similarity. And Separate similarity scores are calculated for each feature type.

Hybrid Similarity Score:

= (w1 ·TF-IDF Similarity+w2 ·BERT Similarity) \ text {Hybrid Similarity Score}

Aggregate similarity scores from different feature types using weighted averages,

The output is a hybrid similarity score matrix indicating candidate-job alignment.

4.4.5. K-Nearest Neighbors (KNN):

The KNN identify the top N most similar candidates based on their hybrid similarity scores

For each job posting. It Ranks candidates by their similarity scores, ensuring precise and relevant matches.

4.4.6. Final recommendation system

The final stage involves generating actionable recommendations based on clustering and KNN results. For each job posting, candidates are ranked within their cluster based on similarity scores.This workflow explicitly separates the Collaborative Filtering and Content-Based Filtering processes, followed by aggregation, dimensionality reduction, similarity computation, and finally clustering and KNN. By combining the strengths of lexical and semantic filtering methods, the model delivers a robust and scalable job recommendation system that effectively addresses modern recruitment challenges.

Conclusion:

This section provides a comprehensive overview of the models implemented to power the KAFAA platform's recommendation system. Each model is designed to address specific challenges in job-candidate matching, using distinct methodologies and technologies to optimize performance. The **User-Based Collaborative Filtering model** leverages similarities between user profiles and job descriptions to deliver tailored recommendations, while the **Text-Based Content Filtering model** incorporates advanced Natural Language Processing (NLP) techniques to capture both explicit and contextual relationships. The **Semantic AI Job Matching model** introduces scalable clustering and efficient retrieval mechanisms through BIRCH and FAISS, making it particularly effective for large-scale recruitment platforms. Finally, the **Proposed Model** integrates hybrid filtering methods, combining TF-IDF, embeddings, and Named Entity Recognition (NER) for a holistic and personalized recommendation approach.

5. References:

[1] M. . A. ALZUBAIDI, "The Incidence and Consequences of Skill Mismatch Among Higher Education Graduates in Saudi Arabia".

[2] Gatzioura, A. and Sànchez-Marrè, M, "A case-based recommendation approach for market basket data.".

[3] Lior Rokach, Bracha Shapira and RICCI, Francesco, Recommender systems: Techniques, applications, and challenges. Recommender systems handbook, 2021.

[4] M. AAMIR and M. BHUSRY, "Recommendation system: state of the art approach," 2015.

[5] MOHANTY and Sachi Nandan, "Recommender system with machine learning and artificial intelligence: Practical tools and applications in medical, agricultural and other industries," 2020.

[6] FAYYAZ and Zeshan, "Recommendation systems: Algorithms, challenges, metrics, and business opportunities," 2020.

[7] KO and Hyeyoung, "A survey of recommendation systems: recommendation models, techniques, and application fields," 2022.

[8] I. CUNNINGHAM, and J. HYMAN, "Devolving human resource responsibilities to the line: beginning of the end or a new beginning for personnel?," 1999.

[9] D. A. Rahman and D. A. N. Ameen, "Artificial Intelligence in HR Recruitment".

[10] M. NEGNEVITSKY, Artificial Intelligence.

[11] S. Kaddoura, "Handbook of Research on AI Methods and Applications in Computer Engineering," 2023.

[12] S. Y. F. and A. WIBOWO, " RECOMMENDATION SYSTEM FOR ONLINE JOB VACANCY USING MACHINE LEARNING MODELS," 2023.

[13] . Y. ZAYED, Y. SALMAN and A. HASASNEH, " A recommendation system for selecting the appropriate undergraduate program at higher education institutions using graduate student data.," 2022.

[14] J. DHAMELIYA and N. DESAI, "Job recommendation system using content and collaborative filtering based techniques.," 2019.

[15] R. NARULA, " Enhancing Job Recommendations Using NLP and Machine Learning Techniques".

[16] ALSAIF and S. Ali, "Learning-based matched representation system for job recommendation," 2022.

[17] ROY, P. Kumar, CHOWDHARY, S. Singh, BHATIA and Rocky, "A Machine Learning approach for automation of Resume Recommendation system. Procedia Computer Science," 2020.

[18] A. MULAY, "Job recommendation system using hybrid filtering.," 2022.

[19] R. MISHRA and S. RATHI, "Enhanced DSSM (deep semantic structure modelling) technique for job recommendation.," 2022.

[20] B. PARIDA, P. KUMARPATRA and S. MOHANTY, "Prediction of recommendations for employment utilizing machine learning procedures and geo-area based recommender framework.," 2022.

[21] BELOFF and Natalia, "Implementing AIRM: A New AI Recruiting Model for the Saudi Arabia Labour Market," 2021.

[22] Majed Al Hulayel, "LinkedIn Jobs Data in Saudi Arabia 2020," kaggale, 2021.

[23] nayoobi@cougarnet.uh.edu., "LinkedIn-Dataset," GitHub, 2023.

[24] Ricci, F., Rokach, L., & Shapira, B, Techniques, applications, and challenges. Recommender systems handbook, 2021.

[25] Aamir, M., & Bhusry, M., " Recommendation system: state of the art approach.," 2015.

[26] Gatzioura, A., & Sànchez-Marrè, M., "A case-based recommendation approach for market basket data," 2014.

[27] A case-based recommendation approach for market ba, "A case-based recommendation approach for market basket data," 2014.

6. Appendices:

6.1. Appendix A: Data Sources

	linkedin_id	position_id	position	company	location	level	date	job_functions	industries	description
0	1632901070	51791	Head of Branding Governance Unit	Riyad Bank	Riyadh, Saudi Arabia	Mid-Senior level	2020-01-06	['Marketing', 'Management']	['Banking']	NaN
1	1628497696	51792	Document Controller	Confidential,Confidential	Riyadh, Saudi Arabia	Associate	2020-01-06	['']	['Government Administration']	Document Controller- (Saudi candidates only) ...
2	1666951758	51793	Employee Engagement Specialist	PepsiCo	Riyadh, Saudi Arabia	Mid-Senior level	2020-01-06	['Human Resources']	['Consumer Goods', 'Food & Beverages']	Auto req ID: 199735BRJob DescriptionPepsiCo Sa...
3	1671138260	51794	Strategic Workforce Analyst	Saudi Air Navigation Services	Jeddah, Saudi Arabia	Mid-Senior level	2020-01-06	['']	['Aviation & Aerospace']	Role PurposeTo conduct workforce analysis to s...
4	1670725370	51795	Account Manager, GPS – Riyadh	EY	Riyadh, Saudi Arabia	Not Applicable	2020-01-06	['Sales', 'Business Development']	['Accounting', 'Financial Services']	In a business where are our people are our pro...

11511 rows × 10 columns

Figure 19: First five rows of the Job Dataset

	Intro	Full Name	Workplace	Location	Connections	Photo	Followers	About	Experiences	Number of Experiences	...	Number of Scores	Languages	Number of Languages	Organizations	Number of Organizations	Interests	Number of Interests	Activities	Number of Activities	Label
0	{'Full Name': 'chenxia (polly) Pei', 'Workplac...	chenxia (polly) Pei	Jiangsu Junyao mainly offer services to cement...	Wuxi, Jiangsu, China	500	No	717	Our GMH brick has the proprietary intellectual...	{'0': {'Role': 'Sales', 'Workplace': 'Jiangsu ...	2	...	0	{}	0	{}	0	{'0': {'Interest': 'Trina Solar', 'ID': '69648...	4	{'chenxia-pei-80177594': {'Full Name': 'chenxi...	1	1
1	{'Full Name': 'NEHA CHANDOK', 'Workplace': 'So...	NEHA CHANDOK	Software Analyst	Noida, Uttar Pradesh, India	500	No	1340	NaN	{'0': {'Role': 'Software Analyst', 'Workplace'...	1	...	0	{'Language 0': 'Bangla', 'Language 1': 'Englis...	3	{}	0	{'0': {'Interest': 'Mohamed El-Erian', 'ID': '...	8	{}	0	1
2	{'Full Name': 'Mounika Mungamuri', 'Workplace'...	Mounika Mungamuri	Senior Consultant at Infosys	Hyderabad, Telangana, India	7	Yes	7	NaN	{'0': {'Role': 'Senior Consultant', 'Workplace...	2	...	0	{}	0	{}	0	{}	0	{}	0	1
3	{'Full Name': 'Katarina Djuric', 'Workplace': ...	Katarina Djuric	--	Belgrade, Serbia	0	Yes	0	NaN	{'0': {'Role': 'Instructor', 'Workplace': 'GE'...	1	...	0	{}	0	{}	0	{'0': {'Interest': 'GE', 'ID': '1015', 'Catego...	1	{}	0	1
4	{'Full Name': 'Rachel Lally', 'Workplace': '--...	Rachel Lally	--	Dublin, County Dublin, Ireland	61	Yes	61	tictok is is the futer	{'0': {'Role': 'Bartender', 'Workplace': 'O'Su...	1	...	1	{}	0	{}	0	{'0': {'Interest': 'Richard Branson', 'ID': 'r...	3	{'garyltravis': {'Full Name': 'Gary Travis', '...	6	1

5 rows × 39 columns

Figure 20: First five rows of the Candidate Dataset

6.2. Appendix B: Exploratory data analysis (EDA)

```
[ ]  # Plot distributions for numeric columns
     numeric_columns = ['Number of Experiences', 'Number of Educations', 'Number of Skills', 'Number of Languages']
     plt.figure(figsize=(10, 8))
     for i, col in enumerate(numeric_columns, 1):
         plt.subplot(2, 2, i)
         sns.histplot(dataset[col], kde=True, bins=20, color="blue")
         plt.title(f"Distribution of {col} Before Preprocessing")
         plt.xlabel(col)
     plt.tight_layout()
     plt.show()
```

```
# Analyze text lengths for columns like 'About' and 'Experiences'
dataset['About_length'] = dataset['About'].apply(lambda x: len(str(x)) if pd.notnull(x) else 0)
dataset['Experiences_length'] = dataset['Experiences'].apply(lambda x: len(str(x)) if pd.notnull(x) else 0)

# Plot text length distributions
plt.figure(figsize=(10, 6))
sns.histplot(dataset['About_length'], kde=True, color='purple', bins=30)
plt.title("Distribution of 'About' Length Before Preprocessing")
plt.xlabel("Character Count")
plt.ylabel("Frequency")
plt.show()

plt.figure(figsize=(10, 6))
sns.histplot(dataset['Experiences_length'], kde=True, color='orange', bins=30)
plt.title("Distribution of 'Experiences' Length Before Preprocessing")
plt.xlabel("Character Count")
plt.ylabel("Frequency")
plt.show()
```

```
# Top 10 workplaces
top_workplaces = dataset['Workplace'].value_counts().head(10)
plt.figure(figsize=(8, 5))
sns.barplot(x=top_workplaces.values, y=top_workplaces.index, palette="coolwarm")
plt.title("Top 10 Workplaces Before Preprocessing")
plt.xlabel("Count")
plt.ylabel("Workplace")
plt.show()
```

```
#location
# Top 10 location before preprocessing
top_location = df_job['location'].value_counts().head(10)
plt.figure(figsize=(8, 5))
sns.barplot(x=top_location.values, y=top_location.index, palette="coolwarm")
plt.title("Top 10 location Before Preprocessing")
plt.xlabel("Count")
plt.ylabel("location")
plt.show()
```

```
# Correlation matrix for numeric features
corr_matrix = dataset[['Number of Experiences', 'Number of Educations', 'Number of Skills', 'Number of Languages']].corr()
plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title("Correlation Heatmap of Numeric Features")
plt.show()
```

Figure 21:EDA before preprocessing

```python
# Distribution of Numeric Features
fig, axes = plt.subplots(2, 2, figsize=(15, 10))
axes = axes.ravel()

numeric_cols = ['Number of Experiences', 'Number of Educations',
                'Number of Skills', 'Number of Languages']

for idx, col in enumerate(numeric_cols):
    sns.histplot(data=dataset, x=col, kde=True, ax=axes[idx])
    axes[idx].set_title(f'Distribution of {col}')
    axes[idx].set_ylabel('Count')

plt.tight_layout()
plt.show()
```

```python
# Top 10 workplaces after preprocessing
top_workplaces = dataset['Workplace'].value_counts().head(10)
plt.figure(figsize=(8, 5))
sns.barplot(x=top_workplaces.values, y=top_workplaces.index, palette="viridis")
plt.title("Top 10 Workplaces After Preprocessing")
plt.xlabel("Count")
plt.ylabel("Workplace")
plt.show()
```

```python
# Top 10 location after  preprocessing
top_location = df_job['location'].value_counts().head(10)
plt.figure(figsize=(8, 5))
sns.barplot(x=top_location.values, y=top_location.index, palette="coolwarm")
plt.title("Top 10 location after Preprocessing")
plt.xlabel("Count")
plt.ylabel("location")
plt.show()
```

```python
# Correlation heatmap of numeric features
corr_matrix = dataset[['Number of Experiences', 'Number of Educations', 'Number of Skills', 'Number of Languages']].corr()
plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title("Correlation Heatmap of Numeric Features After Preprocessing")
plt.show()
```

Figure 22: EDA after preprocessing

6.3 Appendix C: Data Preprocessing

Data Preprocessing code:

```
# Drop unnecessary columns
df_job.drop(columns=['date' , 'linkedin_id'], inplace=True)
```

Figure 23: Drop unnecessary columns from the Job Dataset

```
# Clean list-like strings
def clean_dict_string(x):
    try:
        if pd.isna(x):
            return ''
        x_str = str(x)
        # If it looks like a list or dict string, try to clean it
        if (x_str.startswith('[') and x_str.endswith(']')) or (x_str.startswith('{') and x_str.endswith('}')):
            try:
                parsed = ast.literal_eval(x_str)
                if isinstance(parsed, (list, dict)):
                    return ' '.join(str(item) for item in parsed)
            except:
                pass
        # If not, just return the cleaned string
        return x_str.strip()
    except:
        return str(x)

# Clean all columns in the DataFrame
for column in df_job.columns:
    df_job[column] = df_job[column].apply(clean_dict_string)

# Remove any resulting 'nan' strings
df_job = df_job.replace('nan', '')
```

Figure 24: Dictionary cleaning for Job Dataset

```
df_job.isnull().sum()
```

Figure 25: Missing values for Job Dataset

```
df_job.duplicated().sum()
```

Figure 26: Duplicates checking for Job Dataset

```
df_job['location'].value_counts()
```

```python
# Initialize global variables
# List of Saudi Arabian cities for location matching
saudi_cities = [
    'riyadh', 'jeddah', 'dammam', 'mecca', 'medina', 'khobar', 'dhahran',
    'jubail', 'tabuk', 'qassim', 'hail', 'jizan', 'najran', 'abha',
    'al kharj', 'hofuf', 'yanbu', 'sakakah', 'qatif', 'khamis mushait',
    'arar', 'al bahah', 'bisha', 'umluj', 'rabigh', 'al ula', 'turaif'
]

# Caches to store results and avoid redundant operations
web_search_cache = {}  # Store web search results
company_locations = {}  # Store verified company locations
```

```python
def ddg(query, max_results=3):  # Perform a web search using DuckDuckGo's search engine.
    with DDGS() as ddgs:
        results = list(ddgs.text(query, max_results=max_results))
    return results

# This function works by taking a search query and the number of results to retrieve.
# Returns the top search results as a list of titles and descriptions.
# Why did we do this function: To fetch relevant information from the web about company locations
# when the dataset lacks this information. It helps in cases where no clear location is mentioned directly
# Example: Searching for "Aramco company location Saudi Arabia" will fetch the most relevant pages mentioning the company's location in Saudi Arabia.
```

```python
def standardize_city_name(city_name):
    if pd.isna(city_name):
        return None

    # Convert to lowercase and strip whitespace
    city_name = str(city_name).lower().strip()

    # Remove ", saudi arabia" and similar suffixes first
    city_name = city_name.replace(', saudi arabia', '')

    # Split by comma and take first part to remove region/country info
    city_name = city_name.split(',')[0].strip()

    # City name variations mapping
    city_variations = {
        'riyadh': 'riyadh',
        'dammam': 'dammam',
        'ad dammām': 'dammam',
        'ad dammam': 'dammam',
        'eastern province': 'dammam',
        'eastern': 'dammam',
        'dammam industrial area': 'dammam',
        'dammam industrial': 'dammam',
        'khobar': 'khobar',
        'al khobar': 'khobar',
```

```
    'al-khobar': 'khobar',
    'alkhobar': 'khobar',
    'jeddah': 'jeddah',
    'jiddah': 'jeddah',
    'dhahran': 'dhahran',
    'jubail': 'jubail',
    'al jubail': 'jubail',
    'al-jubail': 'jubail',
    'aljubail': 'jubail',
    'qassim': 'qassim',
    'al qasim': 'qassim',
    'qaseem': 'qassim',
    'buraidah': 'qassim',
    'buraydah': 'qassim',
    'makkah': 'mecca',
    'mecca': 'mecca',
    'mekka': 'mecca',
    'medina': 'medina',
    'madinah': 'medina',
    'al madinah': 'medina',
    'al-madinah': 'medina',
    'yanbu': 'yanbu',
    'tabuk': 'tabuk',
    'al kharj': 'al kharj',
    'alkharj': 'al kharj',
```

```
    'alkharj': 'al kharj',
    'hafr al batin': 'hafr al batin',
    'najran': 'najran',
    'qatif': 'qatif',
    'al qatif': 'qatif',
    'hail': 'hail',
    "ha'il": 'hail',
    'hofuf': 'hofuf',
    'al hufuf': 'hofuf',
    'al hofuf': 'hofuf',
    'jizan': 'jizan',
    'jazan': 'jizan',
    'sakakah': 'sakakah',
    'khamis mushait': 'khamis mushait',
    'asir': 'asir',
    'arar': 'arar',
    'abha': 'abha',
    'umluj': 'umluj',
    'rabigh': 'rabigh',
    'al bahah': 'al bahah',
    'al-bahah': 'al bahah',
    'wadi al dawasir': 'wadi al dawasir',
    'al ula': 'al ula',
    'turaif': 'turaif',
    'ad dawadimi': 'ad dawadimi',
```

```
        'bisha': 'bisha',
        'hinakiyah': 'hinakiyah'
    }

    # Check direct match first
    if city_name in city_variations:
        return city_variations[city_name]

    # Remove common prefixes if not found in mapping
    prefixes = ['al ', 'al-', 'ar ', 'ar-', 'ad ', 'ad-']
    for prefix in prefixes:
        if city_name.startswith(prefix):
            stripped_name = city_name[len(prefix):]
            if stripped_name in city_variations:
                return city_variations[stripped_name]

    # If no match found, return the cleaned city name
    return city_name

# This function works by converting city names to lowercase and strips whitespace.
# Uses a dictionary of city variations to map common alternates ("jeddah, saudi arabia" → "jeddah").
# Removes prefixes like "al-" or "ar-" for consistency.
# Returns None for invalid or unspecified cities.
# why did we do this function: Why: to ensures consistency in city names by resolving variations and formatting issues
```

```python
def search_company_location(company_name):
    # Search DuckDuckGo for company location in Saudi Arabia
    # Returns relevant location text
    try:
        query = f"{company_name} company location Saudi Arabia"
        max_retries = 3
        for attempt in range(max_retries):
            try:
                results = ddg(query, max_results=3) # the call to def ddg(query, max_results=3)
                if results:
                    combined_text = ' '.join([r['title'] + ' ' + r['body'] for r in results])
                    return combined_text.lower()
            except Exception as e:
                if attempt < max_retries - 1:
                    time.sleep(2)
                continue
    except Exception as e:
        print(f"Error searching for {company_name}: {str(e)}")
    return None

# This function works by finding a company's location using DuckDuckGo search results
# Constructs a query with the company name and appends "location Saudi Arabia."
# Searches the web up to 3 times (retry mechanism).
# Extracts and returns text content from search results.
```

```python
def extract_cities_from_text(text, saudi_cities):
    # Extract city names from text content
    # Uses pattern matching and name verification
    if pd.isna(text) or not isinstance(text, str):
        return None

    text = str(text).lower()
    text = standardize_city_name(text) # calling def standardize_city_name(city_name)

    # Direct city mention
    for city in saudi_cities:
        if standardize_city_name(city) in text:
            return standardize_city_name(city)

    # Location patterns
    location_patterns = [
        r'based in ([a-zA-Z\s-]+)',
        r'location:?\s*([a-zA-Z\s-]+)',
        r'located in ([a-zA-Z\s-]+)',
        r'position in ([a-zA-Z\s-]+)',
        r'work in ([a-zA-Z\s-]+)',
        r'office in ([a-zA-Z\s-]+)',
        r'headquarters in ([a-zA-Z\s-]+)',
        r'hq in ([a-zA-Z\s-]+)'
    ]
```

```
    for pattern in location_patterns:
        matches = re.findall(pattern, text)
        if matches:
            standardized_match = standardize_city_name(matches[0])
            if standardized_match:  # Add check for None
                best_match = process.extractOne(
                    str(standardized_match),  # Ensure string type
                    [str(standardize_city_name(city)) for city in saudi_cities if standardize_city_name(city)],  # Filter None values
                    score_cutoff=80
                )
                if best_match:
                    return best_match[0]

    return None

# This function works by cleaning and standardizes the input text.
# Searches for direct mentions of Saudi cities.
# Uses regular expressions to find location phrases like "based in Riyadh."
# Matches extracted text with the standardized city list using fuzzy matching.
```

```
def build_company_location_map(df):
    # Creates a mapping of companies to their most frequently mentioned locations in the dataset.
    # Based on historical data
    # Build a mapping of companies to their most common locations
    company_locations = {}

    # Group by company and get most common location
    company_loc_groups = df[df['location'].notna()].groupby('company')['location'].agg(
        lambda x: x.value_counts().index[0] if len(x.value_counts()) > 0 else None
    )

    for company, location in company_loc_groups.items():
        if location and str(location).lower() != 'saudi arabia':
            company_locations[str(company).lower()] = standardize_city_name(str(location))

    return company_locations

# Groups job data by company and counts location mentions.
# Selects the most common location for each company and stores it in a global dictionary.
```

```
def determine_location(row):
    # Location identification process:
    # 1. Direct location check --> Uses the location column
    # 2. Description analysis --> Extracts city names from the job description using extract_cities_from_text
    # 3. Company history check --> Looks up the company's location in the company_locations dictionary
    # 4. Similar company matching --> Matches company names using fuzzy string similarity ("Aramco" vs. "Saudi Aramco")
    # 5. Web search verification --> If all else fails, searches the web for the company's location.
    # 6. Default location (Riyadh) --> If no location is found, defaults to "Riyadh."
    if pd.notna(row['location']):
        loc = standardize_city_name(str(row['location']).strip().lower())
        if loc != 'saudi arabia' and loc:
            return loc, 'direct'

    company = str(row['company']).strip().lower()

    desc_location = extract_cities_from_text(str(row.get('description', '')), saudi_cities)
    if desc_location:
        return desc_location, 'description'

    if company in company_locations:
        return company_locations[company], 'company_history'
```

```
    similar_company = process.extractOne(
        company,
        list(company_locations.keys()),
        score_cutoff=85
    )
    if similar_company:
        return company_locations[similar_company[0]], 'similar_company'

    ## Web search
    if company not in web_search_cache:
        search_results = search_company_location(row['company'])
        web_search_cache[company] = search_results
    else:
        search_results = web_search_cache[company]

    if search_results:
        web_location = extract_cities_from_text(search_results, saudi_cities)
        if web_location:
            return web_location, 'web_search'

    return 'riyadh', 'default'
```

```
def analyze_location_results(df):
    # Generate location analysis report:
    # - Source distribution --> Summarizes the distribution of location sources (direct, description, web search)
    # - Common locations --> Lists the most common cities in the dataset
    # - Sample results --> Displays example rows categorized by the source of location information
    # - Search statistics
    print("\n=== Location Cleaning Analysis ===")

    print("\nLocation Source Distribution:")
    source_dist = df['location_source'].value_counts(dropna=False)  ## Include NaN values
    total = len(df)
    for source, count in source_dist.items():
        source_name = 'Unknown' if pd.isna(source) else source
        percentage = (count/total) * 100
        print(f"{source_name:<15} {count:>5} ({percentage:.1f}%)")

    print("\nTop 10 Most Common Locations:")
    location_counts = df['location'].value_counts().head(10)
    for city, count in location_counts.items():
        percentage = (count/total) * 100
        print(f"{standardize_city_name(city):<15} {count:>5} ({percentage:.1f}%)")
```

```
    print("\nSample Results by Source:")
    for source in df['location_source'].unique():
        if pd.isna(source):
            source_name = 'Unknown'
        else:
            source_name = source

        samples = df[df['location_source'] == source][['company', 'location']].head(2)
        print(f"\n{source_name.title()}:")
        for _, row in samples.iterrows():
            print(f"{row['company']}: {standardize_city_name(row['location'])}")

    web_searches = len(df[df['location_source'] == 'web_search'])
    if web_searches > 0:
        print(f"\nWeb Search Statistics:")
        print(f"Total web searches performed: {web_searches}")
        print(f"Percentage of entries requiring web search: {(web_searches/total)*100:.1f}%")
```

```python
def clean_location(df):
    # Main location cleaning function
    global company_locations

    # Build company location map
    company_locations = build_company_location_map(df)

    # Create temporary lists to store results
    new_locations = []
    new_sources = []

    # Process each row with tqdm
    for _, row in tqdm(df.iterrows(), total=len(df), desc="Cleaning locations"):
        location, source = determine_location(row)
        new_locations.append(location)
        new_sources.append(source)

    # Update DataFrame columns
    df['location'] = new_locations
    df['location_source'] = new_sources

    return df
```

```python
# usage
df_job = clean_location(df_job)
analyze_location_results(df_job)
```

Figure 27: Preprocessing 'location' column in the Job Dataset

```python
relevant_columns = [
    'About',                 # Professional summary
    'Workplace',             # Current/past workplaces
    'Experiences',           # Work experience details
    'Number of Experiences', # Quantity of experience
    'Educations',           # Educational background
    'Number of Educations',  # Level of education
    'Skills',               # Professional skills
    'Number of Skills',     # Skill breadth
    'Languages',            # Language proficiencies
    'Number of Languages'   # Language versatility
]

# Drop all other columns
dataset = dataset.drop(columns=[col for col in dataset.columns if col not in relevant_columns])
dataset
```

Figure 28: Drop unnecessary columns from the candidate Dataset

```
def clean_experience_text(text):
    try:
        # If it's a dictionary string, convert to dict first
        if isinstance(text, str) and text.startswith('{'):
            text = ast.literal_eval(text)

        if isinstance(text, dict):
            experience_parts = []
            for exp in text.values():
                if isinstance(exp, dict):
                    role = exp.get('Role', '').strip()
                    workplace = exp.get('Workplace', '').replace('Â·', '|').strip()
                    duration = exp.get('Duration', '').replace('Â·', '-').strip()

                    # Clean workplace (remove Full-time/Part-time)
                    if '|' in workplace:
                        workplace = workplace.split('|')[0].strip()

                    # Clean duration
                    duration = duration.replace('mos', 'months').replace('mo ', 'month ')

                    # Combine information meaningfully
                    parts = []
                    if role and workplace:
                        parts.append(f"{role} at {workplace}")
                    elif role:
                        parts.append(role)
                    if duration:
                        parts.append(f"({duration})")

                    if parts:
                        experience_parts.append(' '.join(parts))

            return ' • '.join(experience_parts)
        return text
    except:
        return text

def clean_dict_text(text, col_name):
    try:
        # If it's a dictionary string, convert to dict first
        if isinstance(text, str) and (text.startswith('{') or text.startswith('[')):
            text = ast.literal_eval(text)

        if isinstance(text, dict):
            if col_name == 'Experiences':
                return clean_experience_text(text)
```

```
            else:
                # For other columns, keep original flattening logic
                values = []
                for v in text.values():
                    if isinstance(v, dict):
                        values.extend(v.values())
                    else:
                        values.append(str(v))
                return ' '.join([str(v) for v in values if str(v) != ''])
        return text
    except:
        return text

# Apply cleaning to each column
columns_to_clean = ['Workplace', 'About', 'Experiences', 'Educations', 'Skills', 'Languages']
for col in columns_to_clean:
    dataset[col] = dataset[col].apply(lambda x: clean_dict_text(x, col))
```

Figure 29: Dictionary cleaning for Candidate Dataset

```
def check_and_clean_blanks(dataset):
    # List of possible blank values
    blank_values = ['', ' ', '[]', '{}', 'nan', 'NaN', 'None', 'none', 'null', 'NULL']

    # Print initial info about blank-like values
    print("Before cleaning:")
    for col in dataset.columns:
        blank_count = dataset[col].isin(blank_values).sum()
        empty_string_count = (dataset[col] == '').sum()
        empty_dict_count = (dataset[col] == '{}').sum()
        empty_list_count = (dataset[col] == '[]').sum()

        if blank_count > 0 or empty_string_count > 0 or empty_dict_count > 0 or empty_list_count > 0:
            print(f"\n{col}:")
            print(f"Blank values: {blank_count}")
            print(f"Empty strings: {empty_string_count}")
            print(f"Empty dicts: {empty_dict_count}")
            print(f"Empty lists: {empty_list_count}")

    # Replace blank-like values with NaN
    for col in dataset.columns:
        # Replace blank values
        dataset[col] = dataset[col].replace(blank_values, np.nan)
```

```
        # Check for string columns
        if dataset[col].dtype == 'object':
            # Replace strings that are only whitespace
            dataset[col] = dataset[col].apply(lambda x: np.nan if isinstance(x, str) and x.isspace() else x)

            # Replace empty dicts/lists (checking both string and actual dict/list)
            dataset[col] = dataset[col].apply(lambda x: np.nan if (isinstance(x, (dict, list)) and len(x) == 0) or
                                                                  (isinstance(x, str) and x.strip() in ['{}', '[]'])
                                                                  else x)

    # Print final null count
    print("\nfinal Null counts:")
    print(dataset.isnull().sum())

    return dataset

# Apply the cleaning
dataset = check_and_clean_blanks(dataset)
```

```
def fill_nulls(dataset):
    # Helper function to extract latest workplace from experience string
    def extract_workplace_from_exp(exp_text):
        if pd.isna(exp_text):
            return None
        try:
            first_exp = exp_text.split('•')[0].strip()
            if 'at' in first_exp:
                company = first_exp.split('at')[1]
                if '(' in company:
                    company = company.split('(')[0]
                return company.strip()
        except:
            return None
        return None

    # Fill Workplace
    def fill_workplace(row):
        if pd.isna(row['Workplace']):
            return extract_workplace_from_exp(row['Experiences'])
        return row['Workplace']
```

```
dataset['Workplace'] = dataset.apply(fill_workplace, axis=1)
dataset['Workplace'] = dataset['Workplace'].fillna('Open to Opportunities')

# Fill Experiences
def fill_experiences(row):
    if pd.isna(row['Experiences']):
        if row['Workplace'] != 'Open to Opportunities':
            return f"Professional at {row['Workplace']} (Current)"
        return 'Seeking New Opportunities'
    return row['Experiences']

dataset['Experiences'] = dataset.apply(fill_experiences, axis=1)

# Skills filling
def infer_skills_from_exp(exp_text):
    if pd.isna(exp_text):
        return 'Core Professional Skills'
```

```
skill_categories = {
    'Leadership': {
        'keywords': ['manager', 'director', 'lead', 'supervisor', 'chief', 'head'],
        'skills': [
            'Team Leadership',
            'Strategic Planning',
            'Decision Making',
            'Performance Management',
            'Team Development'
        ]
    },
    'Project Management': {
        'keywords': ['coordinator', 'manager', 'specialist', 'lead'],
        'skills': [
            'Project Planning',
            'Resource Management',
            'Timeline Management',
            'Risk Assessment',
            'Process Improvement'
        ]
    },
```

```
'Communication': {
    'keywords': ['relations', 'communications', 'engagement', 'specialist', 'coordinator'],
    'skills': [
        'Stakeholder Communication',
        'Presentation Skills',
        'Report Writing',
        'Public Speaking',
        'Professional Writing'
    ]
},
'Analysis': {
    'keywords': ['analyst', 'specialist', 'data', 'research', 'growth'],
    'skills': [
        'Data Analysis',
        'Market Research',
        'Strategic Analysis',
        'Performance Metrics',
        'Trend Analysis'
    ]
},
```

```
'Client Relations': {
    'keywords': ['account', 'sales', 'service', 'relations', 'broker'],
    'skills': [
        'Client Management',
        'Relationship Building',
        'Negotiation',
        'Customer Success',
        'Account Management'
    ]
},
'Operations': {
    'keywords': ['operations', 'coordinator', 'specialist', 'administrator'],
    'skills': [
        'Process Management',
        'Operational Efficiency',
        'Quality Control',
        'Policy Implementation',
        'Systems Administration'
    ]
},
```

```
'Business Development': {
    'keywords': ['growth', 'sales', 'marketing', 'business'],
    'skills': [
        'Strategic Partnerships',
        'Revenue Growth',
        'Business Strategy',
        'Market Expansion',
        'Opportunity Assessment'
    ]
},
'Program Management': {
    'keywords': ['program', 'project', 'initiative'],
    'skills': [
        'Program Development',
        'Stakeholder Management',
        'Budget Management',
        'Impact Assessment',
        'Program Evaluation'
    ]
},
```

```
    'Technical': {
        'keywords': ['technical', 'systems', 'platform', 'digital'],
        'skills': [
            'Technical Implementation',
            'Systems Management',
            'Digital Tools',
            'Technology Solutions',
            'Platform Management'
        ]
    },
    'Administrative': {
        'keywords': ['coordinator', 'administrative', 'support'],
        'skills': [
            'Process Coordination',
            'Documentation Management',
            'Administrative Support',
            'Office Management',
            'Workflow Optimization'
        ]
    }
}
```

```
    core_skills = [
        'Professional Communication',
        'Problem Solving',
        'Time Management',
        'Collaboration',
        'Organizational Skills'
    ]

    exp_lower = exp_text.lower()
    inferred_skills = set(core_skills)

    for category in skill_categories.values():
        if any(keyword in exp_lower for keyword in category['keywords']):
            inferred_skills.update(category['skills'][:3])

    final_skills = list(inferred_skills)
    if len(final_skills) > 10:
        final_skills = final_skills[:10]

    return ' | '.join(final_skills)
```

```
def fill_skills(row):
    skills = row['Skills']
    if isinstance(skills, str) and skills.strip() == '':
        return infer_skills_from_exp(row['Experiences'])
    if isinstance(skills, float) and pd.isna(skills):
        return infer_skills_from_exp(row['Experiences'])
    return skills

dataset['Skills'] = dataset.apply(fill_skills, axis=1)

# Languages filling
dataset['Languages'] = dataset['Languages'].fillna('English')

# Numeric columns filling
def count_experiences(exp_text):
    if pd.isna(exp_text):
        return 0
    return len([x for x in exp_text.split('•') if x.strip()])
```

```
def fill_numeric_column(row, col):
    if pd.isna(row[col]):
        if col == 'Number of Experiences':
            return count_experiences(row['Experiences'])
        elif col == 'Number of Educations':
            return 1
        elif col == 'Number of Languages':
            return len(row['Languages'].split('|')) if pd.notna(row['Languages']) else 1
    return row[col]

numeric_columns = ['Number of Experiences', 'Number of Educations', 'Number of Languages']
for col in numeric_columns:
    if col in dataset.columns:
        dataset[col] = dataset.apply(lambda row: fill_numeric_column(row, col), axis=1)

# Education filling
def infer_education_from_exp(exp_text):
    if pd.isna(exp_text):
        return 'Education details not specified'
```

```
        exp_lower = exp_text.lower()
        if 'manager' in exp_lower or 'director' in exp_lower:
            return "Bachelor's degree or equivalent experience"
        elif 'specialist' in exp_lower or 'coordinator' in exp_lower:
            return "Associate's degree or equivalent experience"
        return 'Professional training or equivalent experience'

    def fill_education(row):
        if pd.isna(row['Educations']):
            return infer_education_from_exp(row['Experiences'])
        return row['Educations']

    dataset['Educations'] = dataset.apply(fill_education, axis=1)

    print("\nRemaining null values after smart filling:")
    print(dataset.isnull().sum())

    return dataset

# Apply the filling
dataset = fill_nulls(dataset)
```

```
def create_about_summary(row):
    if pd.isna(row['About']):
        parts = []

        # Add current role/workplace
        if not pd.isna(row['Workplace']):
            parts.append(f"Professional working as {row['Workplace']}")

        # Add experience details if available
        if isinstance(row['Experiences'], dict):
            exp_count = len(row['Experiences'])
            latest_role = row['Experiences'].get('0', {}).get('Role', '')
            if latest_role:
                parts.append(f"Current/Latest role: {latest_role}")

        # Add education details
        if isinstance(row['Educations'], dict):
            latest_edu = row['Educations'].get('0', {})
            institute = latest_edu.get('Institute', '')
            degree = latest_edu.get('Degree', '')
            if institute and degree:
                parts.append(f"Educated in {degree} from {institute}")
```

```
        # Add skills
        if isinstance(row['Skills'], dict):
            skills = list(row['Skills'].values())[:3]  # Get first 3 skills
            if skills:
                parts.append(f"Skilled in {', '.join(skills)}")

        # Add languages if available
        if isinstance(row['Languages'], dict):
            langs = list(row['Languages'].values())
            if langs:
                parts.append(f"Proficient in {', '.join(langs)}")

        summary = ' '.join(parts)
        return summary if summary else "Professional seeking opportunities"
    return row['About']
```

```
# Store the indexes of originally null values before filling
null_indexes = dataset[dataset['About'].isnull()].index

# Fill null values
dataset['About'] = dataset.apply(create_about_summary, axis=1)

# After filling the nulls we can display samples of them
filled_samples = dataset.loc[null_indexes[:5]]  # Show first 5 filled examples

# Display original null rows with their new filled values
print("Sample of filled 'About' values:")
for idx, row in filled_samples.iterrows():
    print("\nIndex:", idx)
    print("Filled About:", row['About'])
    print("Workplace:", row['Workplace'])
    print("Skills:", row['Skills'])
    print("-" * 80)
```

```
# First, create a temporary copy of the Skills column with the original format
original_skills = dataset['Skills'].copy()
# we created copy because we realized that During duplicate checking, string operations change the format of our skills into tuples of characters


# Now proceed with duplicate checking without risking the Skills column format
# Check for exact duplicates using all relevant columns
exact_duplicates = dataset.duplicated(
    subset=['Workplace', 'About', 'Experiences', 'Number of Experiences',
            'Educations', 'Number of Educations', 'Number of Skills',
            'Languages', 'Number of Languages'],  # Skills column excluded
    keep='first'
).sum()

print(f"Number of exact duplicate profiles: {exact_duplicates}")
```

```
# Remove only exact duplicates
dataset = dataset.drop_duplicates(
    subset=['Workplace', 'About', 'Experiences', 'Number of Experiences',
            'Educations', 'Number of Educations', 'Number of Skills',
            'Languages', 'Number of Languages'],  # Removed 'Skills' from duplicate check
    keep='first'
)

# Verify duplicate removal
exact_duplicates = dataset.duplicated(
    subset=['Workplace', 'About', 'Experiences', 'Number of Experiences',
            'Educations', 'Number of Educations', 'Number of Skills',
            'Languages', 'Number of Languages'],  # Removed 'Skills' from duplicate check
    keep='first'
).sum()

print(f"Number of exact duplicate profiles after removal: {exact_duplicates}")
```

Figure 30: Duplicates checking for Candidate Dataset

```
from sklearn.preprocessing import MinMaxScaler

# Define columns for numeric feature scaling
numeric_columns = ['Number of Experiences', 'Number of Educations', 'Number of Skills', 'Number of Languages']

# Min-Max Scaling (scales each feature to range [0, 1])
scaler = MinMaxScaler()

# Scale the numeric columns
dataset[numeric_columns] = scaler.fit_transform(dataset[numeric_columns])

# Preview the scaled data
print("Scaled Data Preview:")
print(dataset[numeric_columns].head())
```

Figure 31: Numeric Feature Scaling of the Candidate Dataset

Models Preprocessing code:

```
# For the account feature
dataset['account'] = dataset['About'] + ' ' + \
                     dataset['Experiences'] + ' ' + \
                     dataset['Skills'] + ' ' + \
                     dataset['Languages']
```

```
# For the job posting feature
df_job['job_posting'] = df_job['position'] + ' ' + \
                        df_job['company'] + ' ' + \
                        df_job['location'] + ' ' + \
                        df_job['level'] + ' ' + \
                        df_job['job_functions'] + ' ' + \
                        df_job['industries'] + ' ' + \
                        df_job['description']
```

```
# Define the preprocessing function
def preprocess_text(text):
    # Lowercase the text
    text = text.lower()

    # Remove special characters and numbers
    text = re.sub(r'[^\w\s]', '', text)
    text = re.sub(r'\d+', '', text)

    # Tokenize the text
    tokens = word_tokenize(text)

    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    tokens = [word for word in tokens if word not in stop_words]

    # Join tokens back into a string
    return ' '.join(tokens)
```

```
# Apply preprocessing to the `account` and `job_posting` features
dataset['account_processed'] = dataset['About'] + ' ' + \
                    dataset['Experiences'] + ' ' + \
                    dataset['Skills'] + ' ' + \
                    dataset['Languages']

df_job['job_posting_processed'] = df_job['position'] + ' ' + \
                                  df_job['company'] + ' ' + \
                                  df_job['location'] + ' ' + \
                                  df_job['level'] + ' ' + \
                                  df_job['job_functions'] + ' ' + \
                                  df_job['industries'] + ' ' + \
                                  df_job['description']

# Preprocess the concatenated text columns
dataset['account_processed'] = dataset['account_processed'].apply(preprocess_text)
df_job['job_posting_processed'] = df_job['job_posting_processed'].apply(preprocess_text)
```

Named entity Recognition

```python
def extract_entities(text):
    # """Extract named entities using spaCy"""
    doc = nlp(str(text))
    entities = {ent.label_: ent.text for ent in doc.ents}
    return entities
```

```python
# Text preprocessing function
def clean_and_preprocess_text(text):
    # """Advanced text preprocessing with entity preservation"""
    if not isinstance(text, str):
        return ''

    # Extract entities before cleaning
    entities = extract_entities(text)

    # Basic cleaning
    text = text.lower()
    text = re.sub(r'\S+@\S+', '', text)
    text = re.sub(r'[^a-zA-Z\s]', ' ', text)
    text = ' '.join(text.split())

    # Advanced preprocessing
    stop_words = set(stopwords.words('english') + ['etc', 'ie', 'eg'])
    lemmatizer = WordNetLemmatizer()
    tokens = word_tokenize(text)
    tokens = [lemmatizer.lemmatize(token) for token in tokens if token not in stop_words]

    # Add back important entities
    for entity_type, entity_text in entities.items():
        if entity_type in ['ORG', 'PRODUCT', 'SKILL', 'LANGUAGE']:
            tokens.append(entity_text.lower())

    return ' '.join(tokens)

# Apply preprocessing to the datasets
# For LinkedIn accounts
tqdm.pandas()
dataset['account_processed'] = dataset['account'].progress_apply(clean_and_preprocess_text)
# for job postrings
df_job['job_posting_processed'] = df_job['job_posting'].progress_apply(clean_and_preprocess_text)
```

Figure 32: Model preprocessing