Job Seeker System Project

Table of Contents

Project Overview:

The primary goal of this project is to develop a Job Seeker Recommendation System that helps individuals find the most relevant job opportunities based on their resumes, skills, and current market trends. Additionally, the system will assist users by identifying skills gaps and recommending courses to enhance their qualifications, making them better suited for their desired job roles.

Objectives:

- **Job Recommendation**: The system recommends relevant job opportunities to job seekers based on the information in their resumes, ensuring a match with their current skills and experience.
- **Skills Gap Analysis and Course Recommendation**: It analyzes the job seeker's existing skills, identifies missing skills, and recommends relevant courses to help enhance those skills, making the seeker more competitive for job opportunities.
- **Market Trend Analysis**: The system provides insights into the current market trends for the intended job positions, helping the job seeker understand the demand, salary range, and skills required in their desired field.

Tool and Libraries used:

Tool: Python

Libraries: Pandas, Streamlit, NumPy, Plotly Express, Scikit-learn, Sentence-Transformers, SpaCy, FuzzyWuzzy, PyPDF2, Selenium, WebDriver Manager, TQDM.

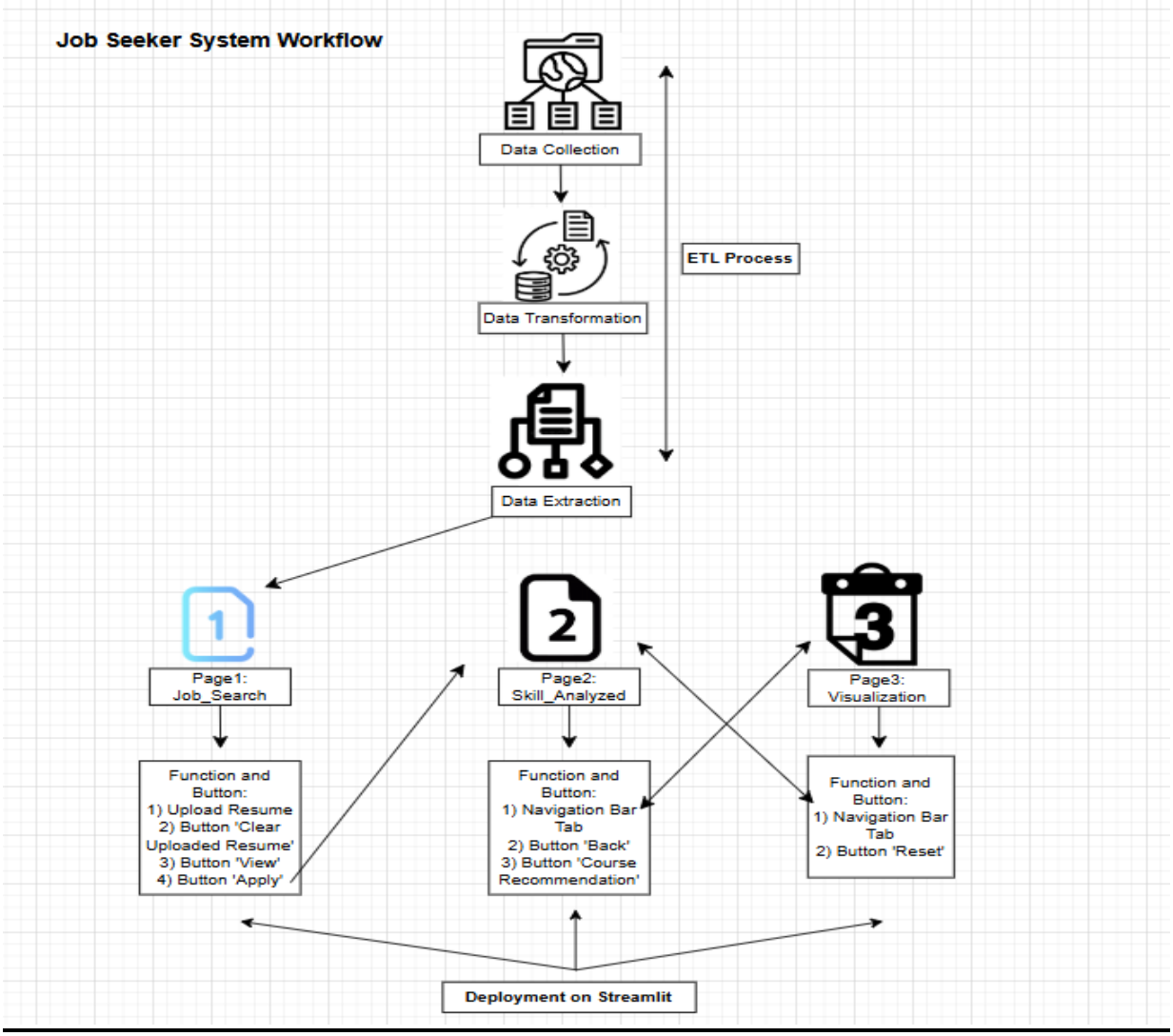
Project Structure:

Job Seeker System

```
|
├── __pycache__/
|
├── Dataset/
|   ├── Coursera/
|   |   ├── Coursera_Completed_Data.csv
|   |   ├── Coursera_after_embeddings.pkl
|   |   ├── Coursera_combined_data.csv
|   |   ├── coursera_AI_Engineer.csv
|   |   ├── coursera_Business_Intelligence_Analyst.csv
|   |   ├── coursera_Machine_Learning_Engineer.csv
|   |   ├── coursera_Software_Developer.csv
|   |   ├── coursera_Business_Analyst.csv
|   |   ├── coursera_Data_Engineer.csv
|   |   ├── coursera_Data_Analyst.csv
|   |   └── coursera_Data_Scientist.csv
|   |
|   ├── Jobstreet/
|   |   ├── Reference.csv
|   |   ├── Jobstreet_after_embeddings.pkl
|   |   ├── Jobstreet_combined_data2.csv
|   |   ├── Updated_Jobstreet_data.csv
|   |   ├── Data_Operation_sample.csv
|   |   ├── Data_Scientist_sample.csv
|   |   ├── Data_Analyst_sample.csv
|   |   ├── Data_Engineer_sample.csv
|   |   ├── Business_Analyst_sample.csv
|   |   ├── Software_Developer_sample.csv
|   |   ├── Machine_Learning_Engineer_sample.csv
|   |   ├── Business_Intelligence_Analyst_sample.csv
|   |   ├── AI_Engineer_sample.csv
```

```
│   │   └── Jobstreet_combined_data.csv
│   │
│   └── Merge Data.py
│
├── Helper/
│   ├── __pycache__/
│   ├── Function.py
│   ├── Scrap_Coursera.py
│   └── Scrap_Jobstreetdata.py
│
├── Pages/
│   ├── Job_Search.py
│   ├── Skill_Analyzed.py
│   └── Visualization.py
│
├── Coursera.ipynb
├── Jobstreet_process.py
├── Jobstreet.ipynb
├── Main.py
├── Navigation.py
└── requirements.txt
```

Workflow:



ETL Process

Data Collection:

This part outlines the process for collecting job listings data from the **JobStreet** and **Coursera** website.

Jobstreet Data Collection:

We gathered data for different job roles by using a web scraping method with Selenium WebDriver. The goal of this data collection is to pull together comprehensive job details, such as Job Title, Company Name, Location, Sector, Job Type, Salary, Job Responsibilities, and the URL Link.

The data was collected using a Python-based script that employs Selenium WebDriver to automate browser interactions with the JobStreet job listing pages. The following steps outline the data collection process:

1. Website URL: The data was collected from the JobStreet website, specifically from the page listing job opportunities for various roles related to data science, machine learning, software development, business analysis, and artificial intelligence.
2. Technology Stack:
    - Selenium WebDriver: A browser automation tool used to interact with the web pages and extract job data.
    - WebDriver Manager: Ensures the correct version of the ChromeDriver is used for Selenium.
    - Pandas: Used to store the collected job data in a structured format and export it to a CSV file.
    - Time Module: Implemented pauses to ensure proper loading of the page content before data extraction.
    - Unicode Normalization: Used to clean and normalize the extracted text.
3. Data Extraction Process:
    - The script starts by visiting the specified URL for the job listings.
    - Job URLs are collected from multiple pages using the XPath of the job links.
    - The script iterates through the collected job URLs and extracts detailed information for each job post, such as job title, company name, job location, job type, salary, job sector, and responsibilities.
    - The extracted text is cleaned and normalized using the “unicodedata” module to ensure consistency and proper encoding.
    - The process is repeated for multiple job roles to collect a comprehensive dataset.
4. Sample Size: The following job roles were targeted, and a specific number of job samples was collected for each role:

- Data Scientist (300samples)
- Data Analyst (300samples)

- Data Engineer (300samples)
- Data Operation(300samples)
- Business Analyst (300samples)
- Software Developer (300samples)
- Machine Learning Engineer (300samples)
- Business Intelligence Analyst (150samples)
- AI Engineer (150samples)

5. Data Fields Collected: The following information was extracted for each job posting:

    - Job Title: The title or position name for the job.
    - Company Name: The name of the company advertising the job.
    - Location: The geographical location of the job.
    - Sector: The industry or sector to which the job belongs (e.g., Information Technology, Finance).
    - Job Type: The type of employment (e.g., full-time, part-time, contract).
    - Salary: The salary information provided for the job (if available).
    - Job Responsibilities: A description of the job responsibilities and requirements.
    - URL Link: The URL link to the specific job posting.

Column Name	Type	Description
Job Title	Categorical/Text	The position advertised
Company Name	Categorical/Text	Name of the employer
Location	Categorical/Text	City/State of the job
Sector	Categorical/Text	Industry or category (e.g., IT, Finance)
Job Type	Categorical	Employment type (Full time, Contract, etc.)
Salary	Semi-Structured/Text	Textual range (in RM); can be parsed for numeric values
Job Responsibilities	Unstructured/Text	Full job description (rich in NLP content)
URL Link	URL/Text	Link to the job posting

6. Data Storage: The collected job data was stored in a structured format as a CSV file for further analysis and processing. Each entry in the dataset corresponds to a job posting, and the columns represent the various attributes mentioned above.

Coursera Data collection:

We gathered data for different learning course by using a web scraping method with Selenium WebDriver. The goal of this data collection is to pull together comprehensive course details, such as Course Name, Provider, Skills Gained, Rating & Reviews, Level & Duration, Course Image, Provider Image and Course Link.

The data was collected from the Coursera platform using a Python-based web scraping script. The following sections describe the steps involved in the data collection process:

1. Website URL:

   This URL returns search results for courses related to e.g.: "AI Engineer". The web scraping process iterates through the search results and course details pages to collect relevant course information.

2. Technology Stack:
   - Selenium WebDriver: Used to automate browser interactions and extract course data.
   - WebDriver Manager: Ensures the correct version of ChromeDriver is used for Selenium.
   - Pandas: Used to store the collected data in a structured format and export it to CSV.
   - Time: Pauses between interactions to ensure proper loading of page content.
3. Data Extraction Process:
   - Navigate to the Coursera Search Page: The script opens the URL and waits for the page to load.
   - Scroll and Load Dynamic Content: Since Coursera loads content dynamically, the script scrolls through the page to load additional courses.
   - Scrape Course Details: For each course found on the page and details were extracted.
   - Stop Criteria: The script stops when the required number of courses (e.g.: 100 per role) is collected or when the scroll attempts reach the maximum limit.
4. Sample Size: The following courses were targeted, and a specific number of course samples was collected for each role:

- Data Scientist (100samples)

- Data Analyst (100samples)
- Data Engineer (100samples)
- Business Analyst (100samples)
- Software Developer (100samples)
- Machine Learning Engineer (100samples)
- Business Intelligence Analyst (100samples)
- AI Engineer (100samples)

5. Data Field Collected: For each course, the following information was extracted:

    - Course Name: The title of the course.
    - Provider: The organization offering the course (e.g., Coursera, a university).
    - Skills Gained: A list of skills that learners will acquire by completing the course (if provided).
    - Rating & Reviews: The average rating of the course and the number of reviews.
    - Level & Duration: The level of the course (e.g., Beginner, Intermediate) and its duration (e.g., 4 weeks).
    - Course Image: The URL of the image representing the course.
    - Provider Image: The URL of the provider's logo or image.
    - Course Link: The direct URL linking to the specific course page for further details.

Column Name	Type	Description
Course Name	Text	Title of the course (e.g., *Generative AI: Prompt Engineering Basics*)
Provider	Categorical	Organisation/institution offering the course (e.g., *IBM*, *Whizlabs*)
Skills Gained	Text/List-like	Skills taught in the course (e.g., *ChatGPT, Generative AI*)
Rating & Reviews	Text/Semi-structured	Rating or feedback summary (e.g., *4.8 stars*, or full review text)
Level & Duration	Text	Course difficulty and estimated time (e.g., *Beginner, 1-4 Weeks*)
Course Image	URL	Link to the course's banner image
Provider Image	URL	Link to the image/logo of the provider
Course Link	URL	Link to the course on Coursera

6. Data Storage: The collected course data was stored in a structured format as a CSV file for further analysis and processing. Each entry in the dataset corresponds to a course posting, and the columns represent the various attributes mentioned above.

Data Transformation:

After Data Collection, we will scrap the data with multiple csv file, then its will be combined into two main files, which is call "Coursera_combined_data.csv" and "Jobstreet_combined_data.csv".

Coursera Dataset:

- Drop the duplicated data.
- Feature Engineering: Create new features **'Rating Score'** and **'Embeddings skills'**.
  - **'Rating Score':** Extract the numeric part from 'Rating & Reviews' column
  - **'Embeddings skills':** Imports the **SentenceTransformer** class from the sentence-transformers library, which is used to convert text into numerical vector representations **(embeddings)**, loads a pre-trained model called **'all-MiniLM-L6-v2'**, It is a lightweight, efficient model that produces 384-dimensional embeddings, commonly used for semantic textual similarity. Apply this one on 'Skills Gained' column.

Jobstreet Dataset:

- Drop the duplicated data.
- Feature Engineering and Data Preprocessing: Create new features 'Position', 'State', 'Field', 'Level', 'Cleaned_Responsibilities' and 'Embeddings cleaned responsibilities'.
  - **'Position'**: Loads the **distilbart-mnli-12-3 model** for **zero-shot classification** from Hugging Face. This model predicts which of the given candidate labels best matches an input sentence, even without fine-tuning on the labels. Uses the zero-shot classifier to determine the most relevant position. Appends the top prediction to the positions list. f an error occurs, defaults to "Others". Candidate labels = ["Data Scientist", "Data Analyst", "Data Engineer", "Data Operation", "Business Analyst", "Software

Developer", "Machine Learning Engineer", "Business Intelligence Analyst", "AI Engineer", "Others"]

- **‘Salary’**: Apply the mean value of the range. Remove the data that contain string (e.g.: '$7000.0 - $10000.0 p.m. + 9000') and apply **KNN Imputation** with n=3 to impute the missing value. Create a Subset of Data for Imputation, which specifies the columns to be used for the imputation process. This includes column ‘Salary’, categorical columns (Job Type, Position, State, Field), and Level (which represents the job level). The Level column in salary_imputation is mapped to numeric values using the level_mapping dictionary. For example, "Intern" becomes 0, "Junior" becomes 1, and so on. A OneHotEncoder instance from sklearn is created to perform one-hot encoding on categorical columns (Job Type, Position, State, and Field).
- **‘State’**: Extract the state from the column ‘Lacotion’ and remove any space ' ' in the data.
- **‘Field’:** Extract the field from the column ‘Sector’.
- **‘Level’**: Use the exact match from the column "Job Title". The job level categories as Intern, Junior, Senio, Manager, Director, Vice President, Lead and Head, default by Junior.
- **‘Cleaned Responsibilities’**: Convert the format of column ‘Job Responsibilities’ to lowercase, remove special chars/numbers and extra spaces.
- **‘Embeddings cleaned responsibilities’**: Imports the **SentenceTransformer** class from the sentence-transformers library, which is used to convert text into numerical vector representations **(embeddings)**, loads a pre-trained model called **'all-MiniLM-L6-v2'**, It is a lightweight, efficient model that produces 384-dimensional embeddings, commonly used for semantic textual similarity. Apply this one on ‘Cleaned Responsibilities’ column.

Data Loading:

After data transformation, the transformed data is serialized and saved using python’s pickle module format. The data can be reloaded for future processing or modelling task without reapplying transformation steps. The files were stored in **‘Jobstreet_after_embeddings.pkl’** and **‘Coursera_after_embeddings.pkl’**.

Deployment on Streamlit

Page1: 'Job_Search.py'

Resume Upload Section (Left Column)

Functionality:

- Users are prompted to upload their resume in PDF format.
- File uploads are restricted to a maximum size of 10MB.
- The uploaded file is encoded in Base64 and stored in the Streamlit session state.
- A preview of the uploaded PDF is displayed in an embedded viewer.
- A "Clear Uploaded Resume" button is available to reset the upload and related session states.

Matching Area (Right Column)

Process:

- When a resume is uploaded:
    1. The system decodes the Base64 PDF and extracts text from each page.
    2. It uses a pre-loaded embedding model to generate vector representations (embeddings) of the resume.
    3. These embeddings are averaged into a single vector representing the full resume.
    4. The system loads pre-embedded job postings and compares them to the resume using **cosine similarity**.
    5. A matching score is computed for each job post and scaled to percentage format.
    6. The top 10 most relevant job matches are displayed.

Job Posts Display

Each job post card includes:

- Job Title

- Company Name
- Salary
- Matching Score

Buttons under each job:

- **View**: Displays full job details in a scrollable container.
- **Apply**: Triggers the resume analysis and skill extraction process, then navigates to the “Skill Analyzed” page.

Resume & Job Skill Analysis (On Apply)

- When a user clicks "Apply":
    - The resume is re-read and processed to extract relevant skills.
    - Skills are also extracted from the job responsibilities.
    - Both sets of extracted skills are stored in the session state.
    - The system navigates to the next page for deeper skill comparison.
    - The selected job position is saved in st.session_state as selected_position so it can be accessed and used later in other parts of the app.

Skill Extraction Function

Function: extract_skills_from_job_responsibility(job_responsibility_text)

This function processes the textual content of a job post's "Cleaned_Responsibilities" field to identify and extract relevant skills.

Steps:

- Load NLP Model: Loads the medium-sized English spaCy model (en_core_web_md) for natural language processing.
- Define Known Skills: A comprehensive list of predefined technical and business-related skills is hardcoded.
- Match Skills: Uses spaCy's PhraseMatcher to find exact phrases in the job text that match any known skill.

- Lemmatize Tokens: Normalizes words to their base form to improve accuracy and enable better fuzzy matching.
- Extract Noun Chunks: Captures grouped keywords or phrases that represent potential skills.
- Filter Irrelevant Terms: Removes common but non-informative terms like "team", "experience", "skills", etc.
- Fuzzy Matching: Uses fuzz.partial_ratio (from the fuzzywuzzy library) to identify near-matches of known skills with the lemmatized tokens.
- Output: Returns a combined, deduplicated list of relevant skills extracted from the job description.

Function: extract_skills_from_resume(resume_text)

This function is used to extract skills from a user-uploaded resume in plain text format.

Steps:

- Preprocessing: Converts the entire resume text to lowercase for consistency.
- Load NLP Model: Also uses en_core_web_md for tokenization, lemmatization, and parsing.
- Match Known Skills: Applies the same predefined list of skills using the spaCy PhraseMatcher.
- Lemmatize Tokens & Noun Phrases: Just like in the job responsibility function, lemmatizes words and extracts noun phrases for broader context.
- Irrelevant Phrase Filtering: Eliminates general words that do not represent unique skills or technical knowledge.
- Fuzzy Matching: Enhances detection of skills even if they're written slightly differently (e.g., "PyTorch" vs "Pytorch").
- Output: Returns a merged, unique list of extracted and matched skills.

Page2: ‘Skills_Analyzed.py’

This page helps users analyze the gap between their current skills (extracted from a resume) and the skills required for a target job. Based on the identified skill gap, it recommends relevant training programs and courses, primarily from Coursera.

Prerequisites

- Resume must be uploaded and analyzed before using this page.
- Job must be selected to extract target skills.
- The application depends on pre-compute embeddings (from Coursera).

Navigation

This page is part of a multi-page Streamlit application:

- User can return to the ‘Job_Search.py’ page by clicking the **‘Back’** button.
- There is a navigation bar tab show at top left side, which can navigate between page ‘Skill_Anlyzed.py’ and ‘Visualization.py’.

Skill Matching and Visualization Process

- The session_state variables extracted_skills_from_job and extracted_skills_df are used to extract and compare skill sets between the job posting and the user’s resume.
- The system attempts to retrieve the top 6 most similar skills through exact matches between the two sets of skills.
- If fewer than 6 exact matches are found, TF-IDF vectorization is applied to the remaining targeted skills and remaining current skills.
- Cosine similarity is then calculated between each targeted and current skill to find the most relevant matches, filling the top 6 list.
- Next, the system detects the Top 5 missing skills by checking which targeted skills are not present in the current skill set. These missing skills are stored in remaining_missing.
- From remaining_missing, the top 5 skills are selected based on their cosine similarity with the current skill set (from current_clean), identifying the most important skill gaps.
- Finally, the results are displayed:
    - A horizontal bar chart (using Plotly) visualizes the matching scores of the top 6 matched skills.

- A list of the top 5 missing skills is shown alongside the chart, highlighting areas for improvement.

Course Recommendation (on Button Click)

- If "Course Recommendation" button is clicked:
    - Load the preprocessed Coursera dataset ‘Cousera_after_embeddings.pkl’ column name of ‘Embeddings skills’.
    - Convert the top missing skills into sentence embeddings.
    - Compare these embeddings with those from the course dataset using cosine similarity.
    - Rank courses by similarity score and cache the top 30 results.

Course Display

- Courses are displayed in a grid layout (3 columns) with visual cards.
- Each card includes:
    - Course image, provider image, name, skills, rating, and level/duration.
    - An "Enroll" button that links to the course.

Pagination

- Courses are shown 9 at a time.
- Users can navigate between pages using ‘Previous/Next’ buttons.
- Pagination state is maintained using st.session_state.

Page3: ‘Visualization.py’

This page provides visual insights into the job market for a selected position using pie charts and bar charts based on job data filtered by the user-selected job role.

Navigation

This page is part of a multi-page Streamlit application:

- There is a navigation bar tab show at top left side, which can navigate between page ‘Skill_Anlyzed.py’ and ‘Visualization.py’.

Reset Button

- A "Reset" button is displayed at the top.
- When clicked, it:
    - Clears all session state data.
    - Redirects the user back to the ‘Job Search.py’ page.
    - Triggers a rerun to reload the interface.

Data Filtering

- Before generating any visualizations, the system filters the ‘Jobstreet_after_embeddings.pkl’ jobstreet dataset using the selected job position stored in session state:
- Only entries in the dataset where the 'Position' column matches position_selected are retained for further analysis.

Visualization Generation

Using the filtered dataset, the page generates a total of six charts:

Pie Charts (3)

- These charts illustrate the percentage distribution (dominance) of job entries by:
    1. Job Type – e.g., Full-time vs Part-time
    2. State – e.g., Kuala Lumpur vs Selangor
    3. Level – e.g., Entry Level, Mid Level, Senior

These help users understand how job opportunities are distributed across different categories for the selected role.

Bar Charts (3)

- These charts display the average salary associated with:
    1. Field
    2. State
    3. Level

Each bar chart shows the average salary by category, helping users evaluate which fields, locations, or seniority levels offer better pay for the selected position.

Methodologies:

Pre-computed Embeddings: Instead of re-calculating the embeddings every time you need them, you pre-compute them once and store them.

Embeddings: It is numerical vector representations of text. They capture the semantic meaning of words, phrases, or sentences in a high-dimensional space, so similar meanings are placed closer together.

- plan to do repeated similarity comparisons.
- speed up real-time queries in applications.

Model Name: all-MiniLM-L6-v2

This is a pre-trained sentence embedding model developed by SentenceTransformers. It converts sentences or short texts into dense vector representations (embeddings) that capture their semantic meaning.

Key Features:

Fast & Lightweight: Ideal for large-scale and real-time applications (e.g., web apps, APIs).

High Accuracy: Despite being small, it provides strong semantic understanding for English.

Ready-to-use: Plug-and-play model with SentenceTransformer – no fine-tuning required.

Efficient for Search: Works well with vector similarity methods (e.g., cosine similarity) for search and retrieval tasks.

Zero-shot classification: It is an advanced Natural Language Processing (NLP) technique that allows a model to classify text into categories without being explicitly trained on those categories.

Key Characteristics:

- No training needed on specific labels: The model can classify data into user-defined labels it has never seen during training.
- Flexible: You provide the text and a list of possible labels at runtime.
- Semantic matching: It relies on understanding the meaning of both the input text and the labels.

Model Name: **valhalla/distilbart-mnli-12-3**

This model is a DistilBART architecture fine-tuned on the Multi-Genre Natural Language Inference (MNLI) dataset.

Key Features:

- Architecture: A distilled version of the **BART model** – lighter, faster, but still powerful.
- Purpose: Designed for natural language inference (NLI) tasks, which means determining the relationship (entailment, contradiction, neutral) between two sentences.
- Use in Zero-Shot:
    - The model checks whether the input text entails the label description.
    - If it does, the label is considered a match.

MNLI provides a training framework where the model learns to infer if one sentence logically follows another. This inference ability is reused in zero-shot classification, where the input is tested against each candidate label.

KNN Imputation: It is a technique used to fill in (or impute) missing values in a dataset by using the K-Nearest Neighbors (KNN) algorithm. It works by identifying the most similar (or nearest) data points (rows) to the missing value and then imputing the missing value with an average (or weighted average) of those neighboring data points.

How KNN Imputation Works:

1. Find the Nearest Neighbors:
    - The algorithm identifies the K nearest neighbors (i.e., the most similar rows) for the row with the missing value.
    - "Similarity" is usually measured based on the **distance** between the rows. The most common distance metric used is **Euclidean distance** (though others can be used).
2. Impute Missing Value:
    - Once the nearest neighbors are identified, the missing value is imputed by averaging the values of the neighbors (for numerical columns) or by selecting the most frequent value (for categorical columns).
    - For example, if the missing value is for the "Salary" column, the average salary of the nearest K rows will be used to fill in the missing salary.
3. Repeat for All Missing Values:
    - This process is repeated for all missing values in the dataset.

Types of KNN Imputation:

- • Numerical Features: For numerical features, the mean or median of the K nearest neighbors' values is used to fill in the missing value.
- • Categorical Features: For categorical features (e.g., Job Position), the mode (the most frequent category) of the K nearest neighbors' values is used to fill the missing value.

Pickle format Features:

- Preserves all columns and data types: Keeps your embeddings (NumPy arrays/lists), strings, numbers, etc.

- Fast read/write: Much faster than CSV or SQLite — ideal for loading on Streamlit app startup.
- No conversion needed: You don't need to stringify or parse JSON for embeddings.
- Perfect for cosine similarity: Embeddings stay as proper vectors (lists or arrays) — easy for comparison.
- Simple and Python-native: Seamless integration with Pandas, NumPy, Scikit-learn, etc.
- Supports lazy loading or caching: Works well with @st.cache_data in Streamlit to boost performance.

Comparison: **.pkl** vs **.csv** vs **SQLite**

Format	Supports Embeddings	Fast Load	Fast Similarity	Easy Use in Streamlit
.pkl	☑ Yes (native)	☑ Fast	☑ Yes	☑ Best
.csv	✗ No (needs stringify)	✗ Slow	✗ Needs parsing	✗ Not ideal
SQLite	⚠ Only as string	⚠ Medium	✗ No vector ops	⚠ Needs extra work

Why **.pkl** is Faster than **.csv**

Reason	Explanation
No parsing overhead	.csv needs to parse everything as text, then convert to numbers, lists, etc.
Keeps original Python objects	.pkl stores objects exactly as they were in memory (e.g., list, float, array)
Binary format	Much faster I/O than .csv which is plain text
No need to decode/clean strings	No conversion from string to list for embeddings
Smaller in RAM after load	Uses more compact structures in memory (e.g., NumPy arrays)

Real Example: 1000 rows with Embeddings:

Format	Load Time	Memory Use	Embedding Ready?
.csv	~3-5 sec	High (parsing overhead)	No (must parse string to list)
.pkl	~0.3-1 sec	Lower	Yes (list/ array already)

Workflow Overview: Precompute Embeddings + Cosine Similarity

1. **Precompute embeddings**: Convert a set of reference data (e.g., documents, sentences) into embeddings and save them.
2. **Query embedding**: When a new input (query) arrives, convert it into an embedding using the same model.
3. **Cosine similarity**: Compare the query embedding to each precomputed embedding using cosine similarity.

Formula for Cosine Similarity:

$$\text{cosine similarity} = \frac{A \cdot B}{\|A\|\|B\|}$$

Where:

- A and B are the embeddings (vectors) for "Job Responsibility" and the uploaded resume, respectively.
- $\|A\|$ and $\|B\|$ are the magnitudes of the vectors.

TF-IDF stands for **Term Frequency-Inverse Document Frequency**. It is a statistical technique used to represent text data (such as skills or documents) as numerical vectors based on how important a word is within a collection of text.

In the context of this application, **TF-IDF vectorization** is used to measure the similarity between the targeted skills (from the job post) and the current skills (from the resume), especially when exact matches are not found.

How It Works:

- **Term Frequency (TF):** Measures how frequently a word (or skill) appears in a list. Words that appear more often are considered more significant in that list.
- **Inverse Document Frequency (IDF):** Measures how unique or rare a word is across all skill lists. Words that appear in many lists are considered less important.

Why It's Used Here:

- When exact matches between job and resume skills are fewer than 6, the system applies TF-IDF to convert both the **remaining targeted** and **remaining current** skills into numerical vectors.
- Then, using **cosine similarity**, the system compares these vectors to identify **which skills are most alike**, even if they aren't identical in text.

This method allows the system to detect **semantically similar** skills like "data visualization" and "data presentation," which might not match word-for-word but still convey the same capability.

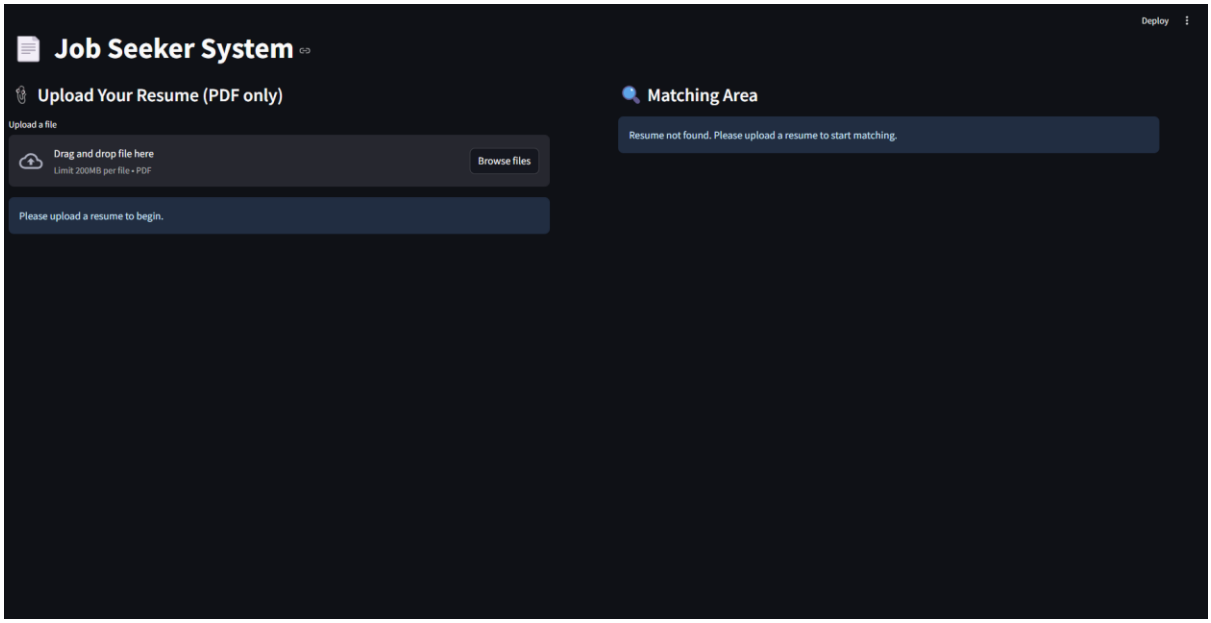
Comparison of TF-IDF vs Embeddings

Feature	TF-IDF	Embeddings
Concept	Based on word frequency statistics	Based on deep learning / neural network models
Type	Sparse vector (high-dimensional)	Dense vector (low-dimensional)
Captures	Exact word presence and importance	Semantic meaning and context
Context awareness	Not context-aware (e.g., 'bank' is the same in all uses)	Context-aware (understands 'bank' as river vs. finance)
Similarity measure	Cosine similarity on word counts	Cosine similarity on learned meanings
Performance	Fast, simple to compute	Slower but more powerful
Interpretability	Easy to understand (based on word counts)	Harder to interpret (black-box model)
Use case in your app	Used to match similar skills based on term importance	Used to match missing skills with course descriptions for course recommendation

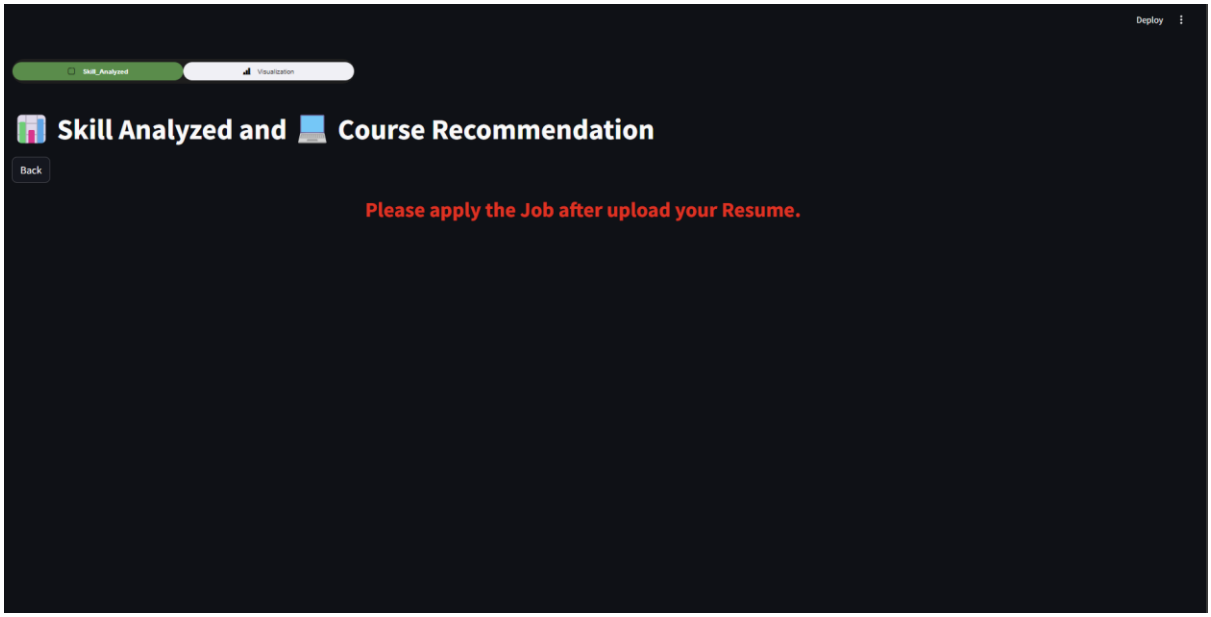
- Use **TF-IDF** for quick and interpretable text comparisons when exact word usage matters.
- Use **embeddings** when **understanding the meaning** behind the text is more important — perfect for **natural language similarity**, **recommendation systems**, and **contextual search**.

User Interface

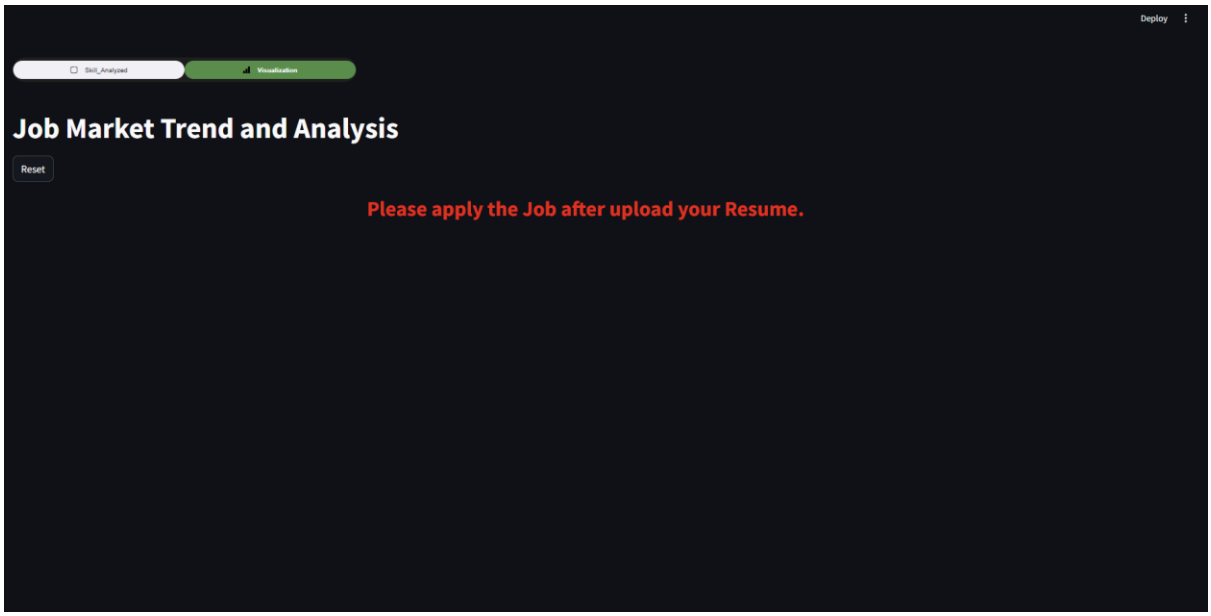
Page1: ‘Job_Search.py’



Page2: ‘Skills_Analyzed.py’

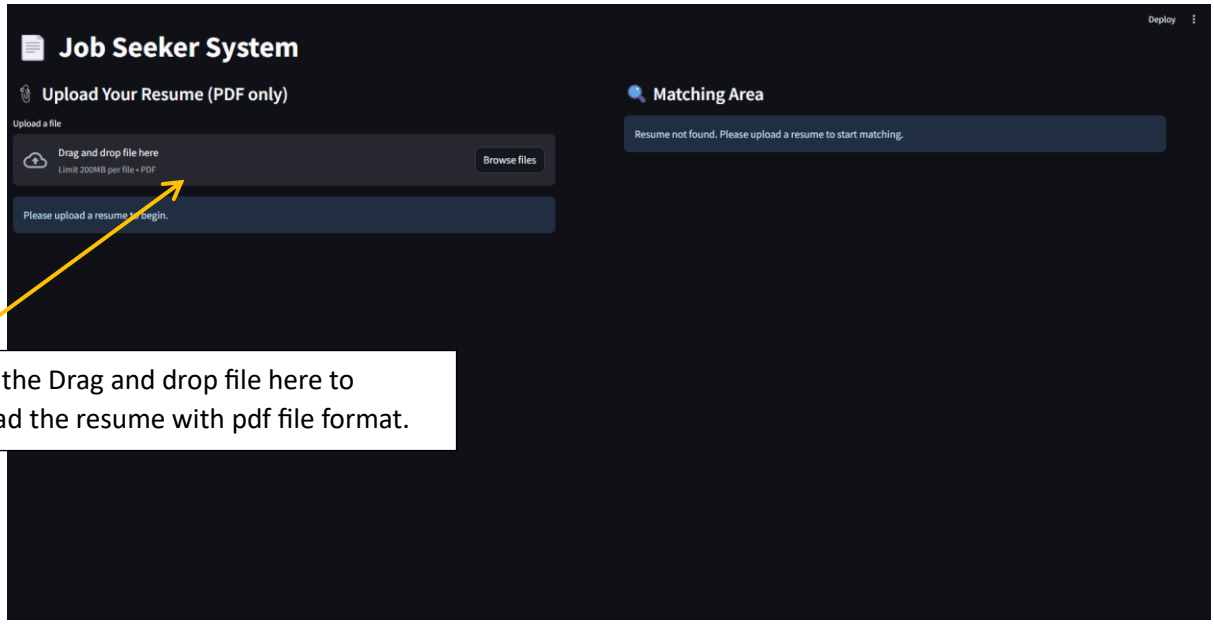


Page3: ‘Visualization.py’



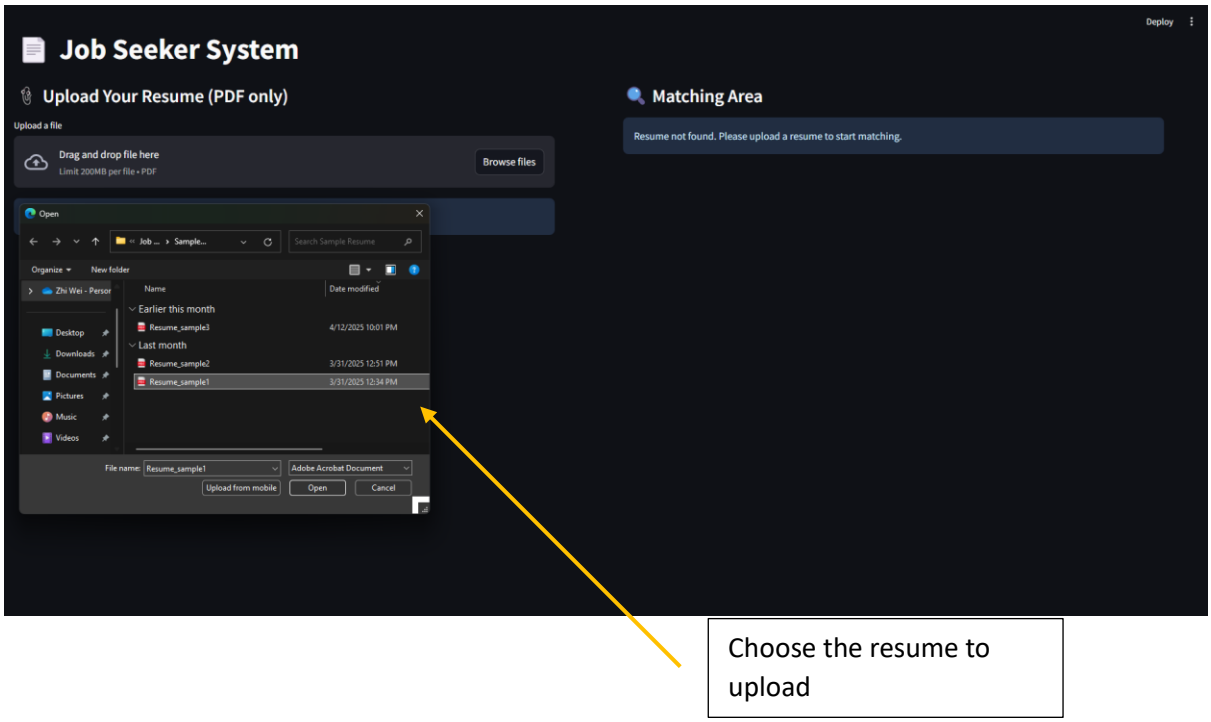
User Manual:

Step1:

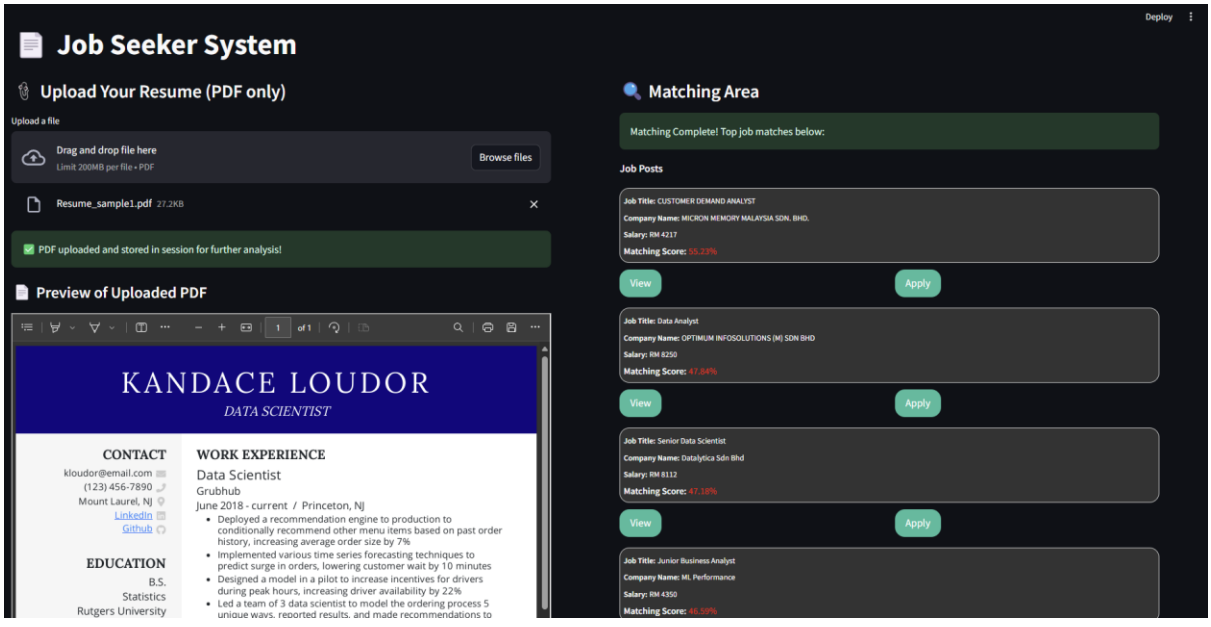


Click the Drag and drop file here to upload the resume with pdf file format.

Step2:

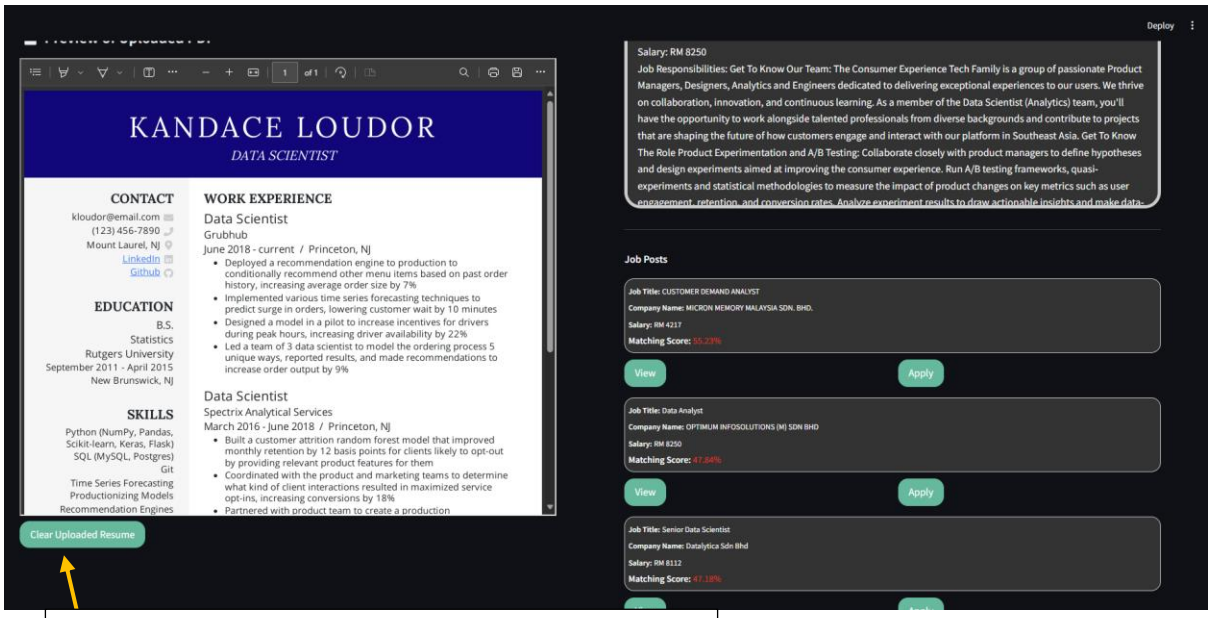


Step3:



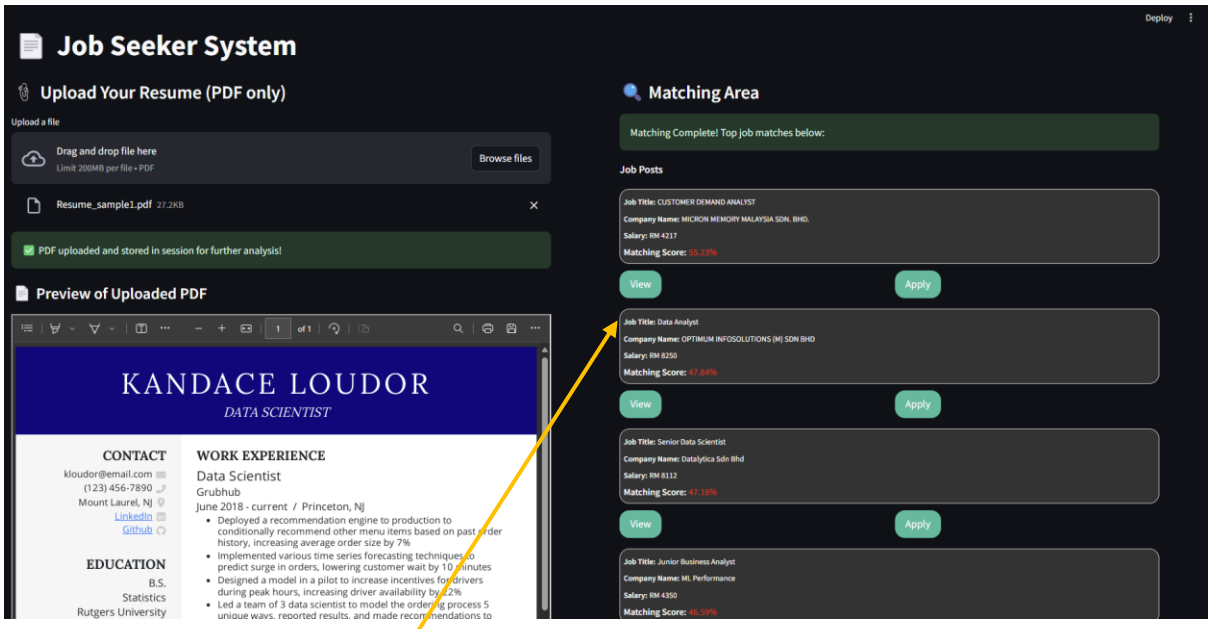
After uploaded the resume, we can see that there got a preview of uploaded resume with pdf format and the right side show the top10 matching job.

Step4:



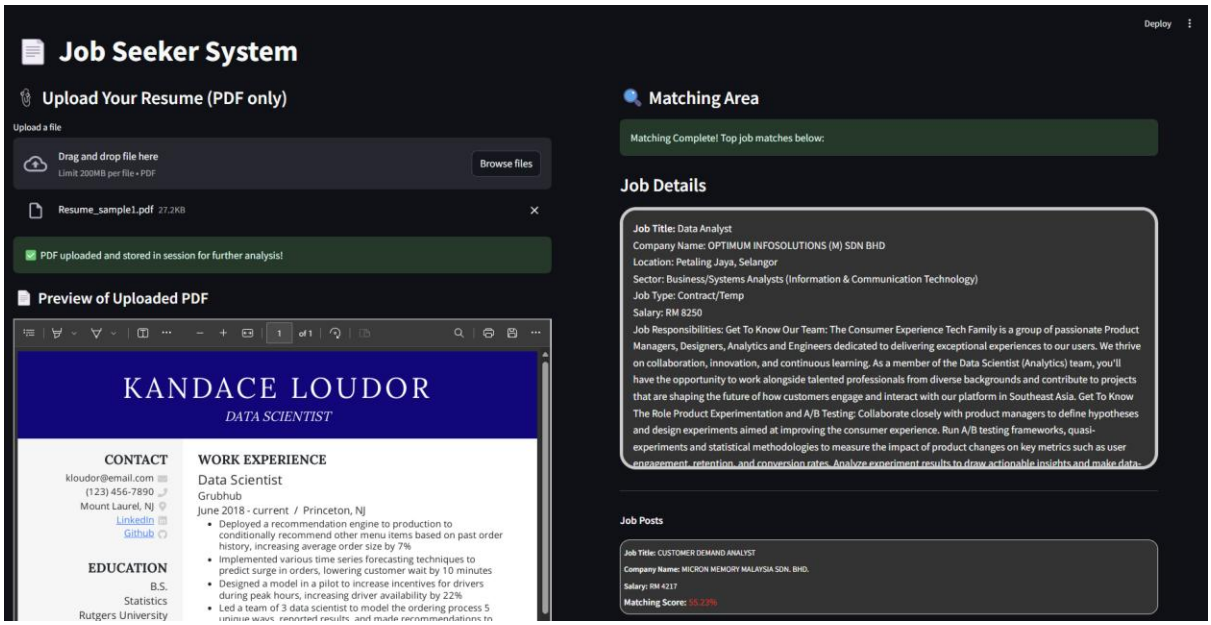
A button “Clear Uploaded Resume”, it will clear the uploaded resume and back to the Step1. (Optional)

Step5:



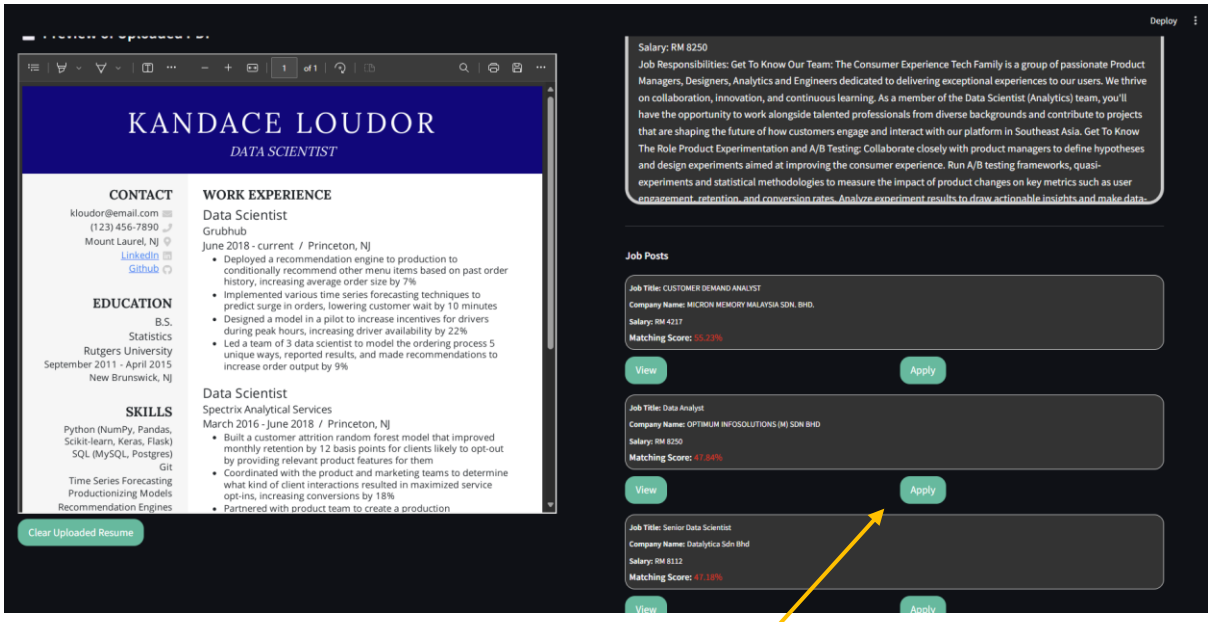
Click any job card “View” button.

Step6:



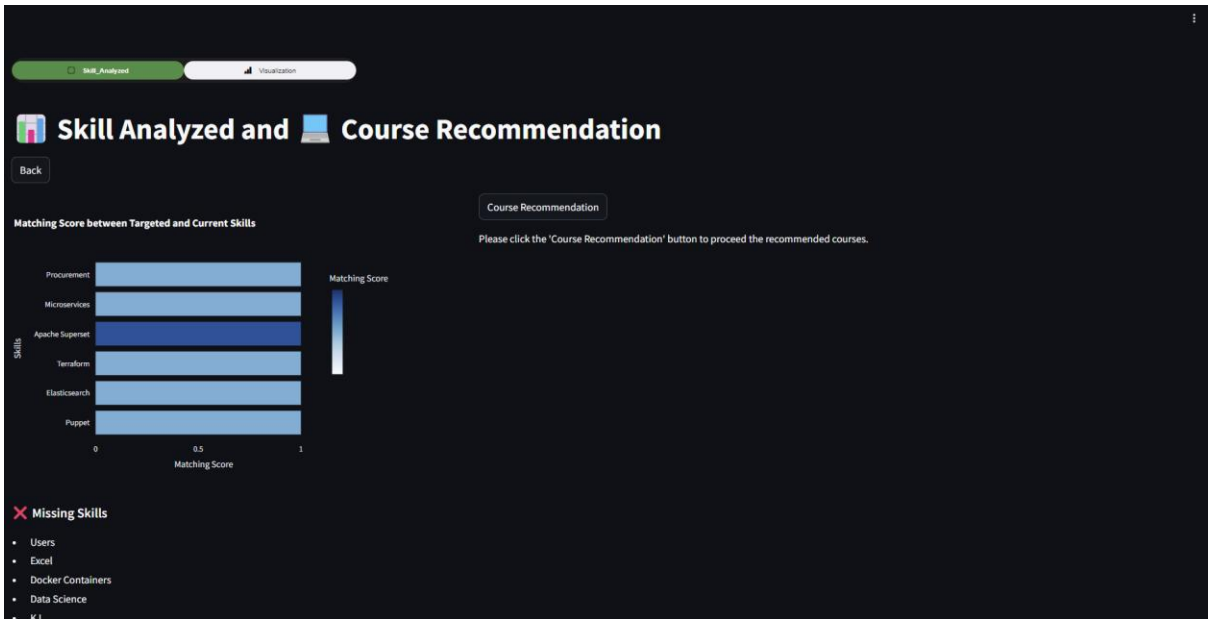
After click the “View” button, we can see that the Job Details side will be shown, which show about the Job Title, Company Name, Location, Sector, Job Type, Salary and Job Responsibilities.

Step7:



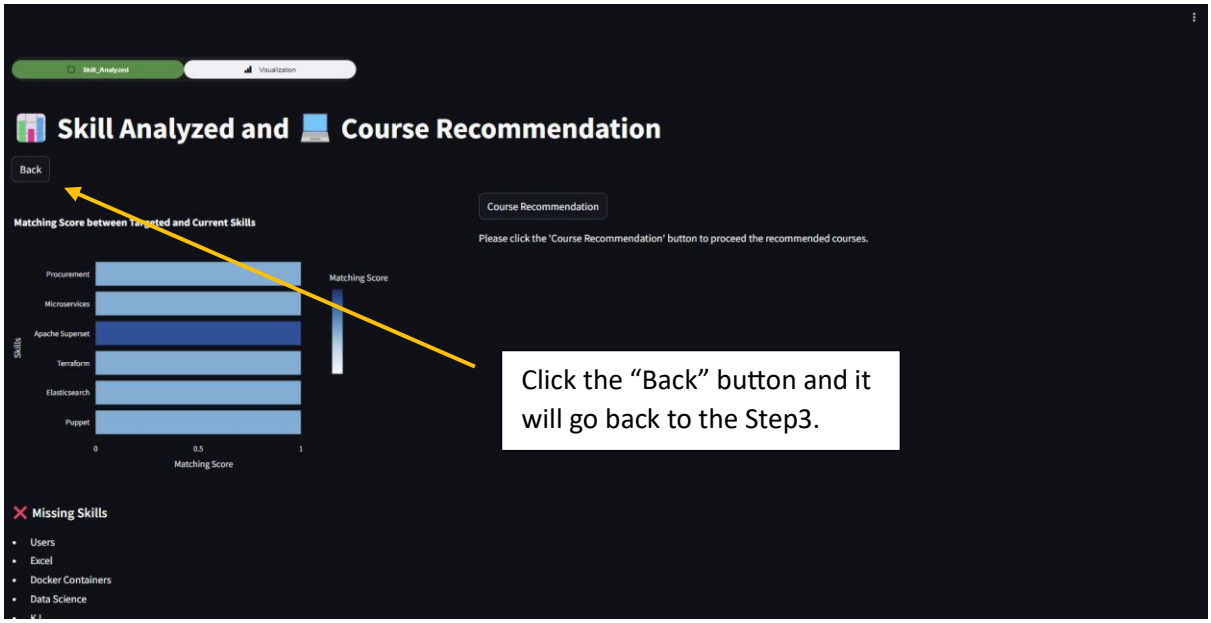
Click “Apply” button to access more function.

Step8:

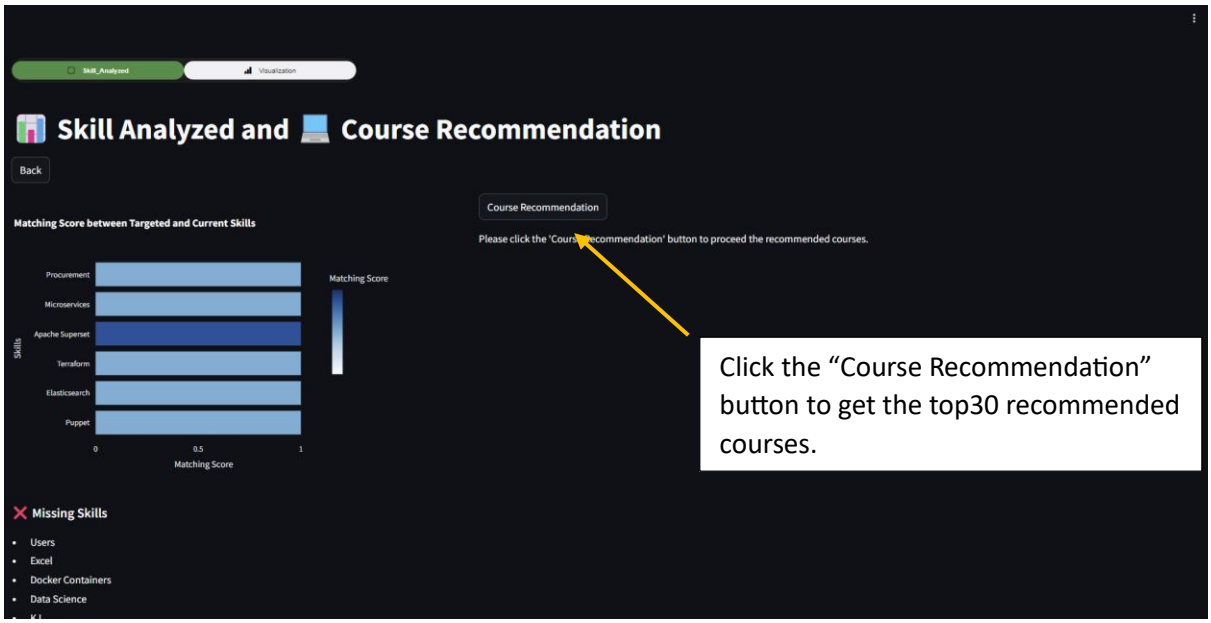


After click the “Apply” button, then we will navigate to the second page ‘Skills_Analyzed.py’, we can see about the bar chart for the skill matching with the targeted skills and current skills and the Top5 Missing Skills is shown at the bottom.

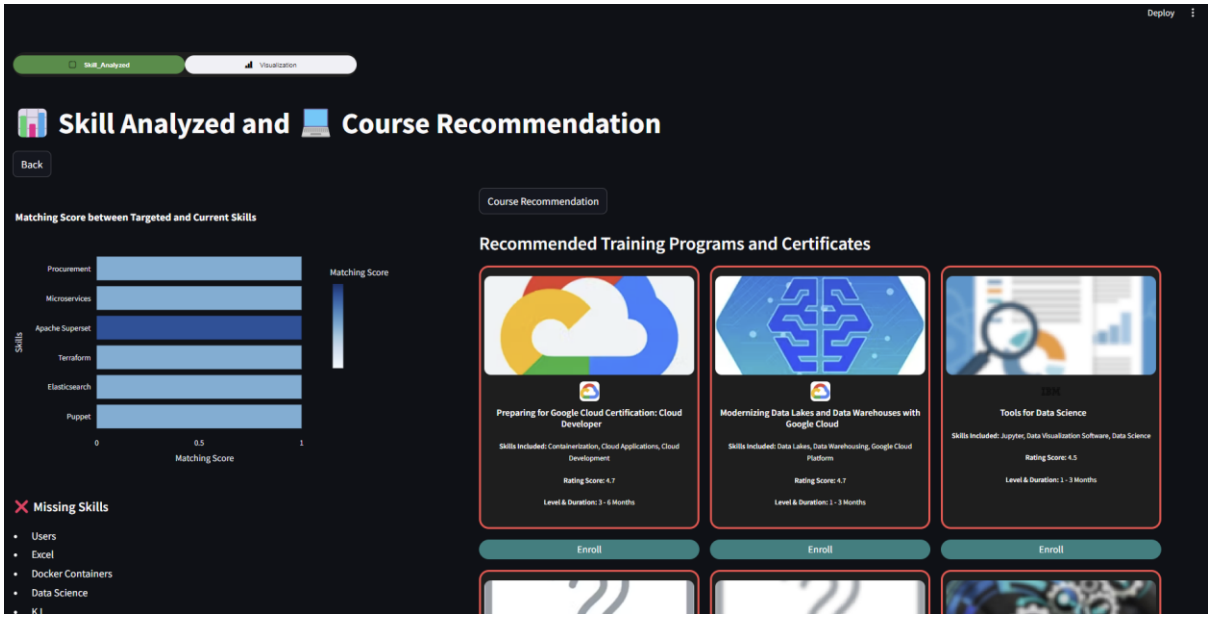
Step9:



Step10:

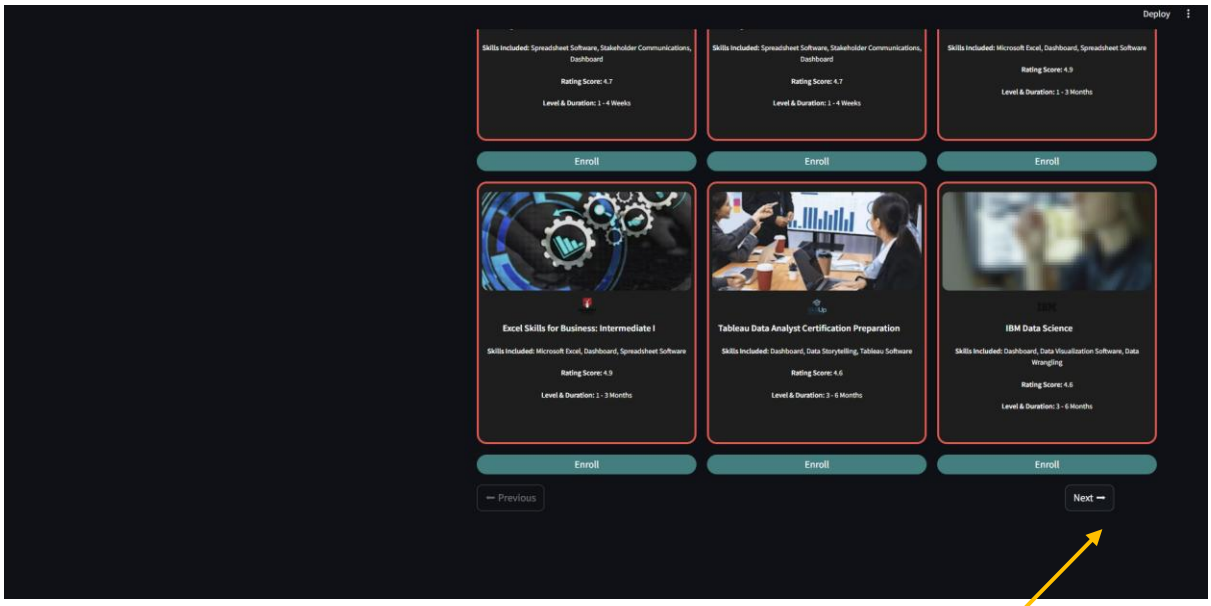


Step11:



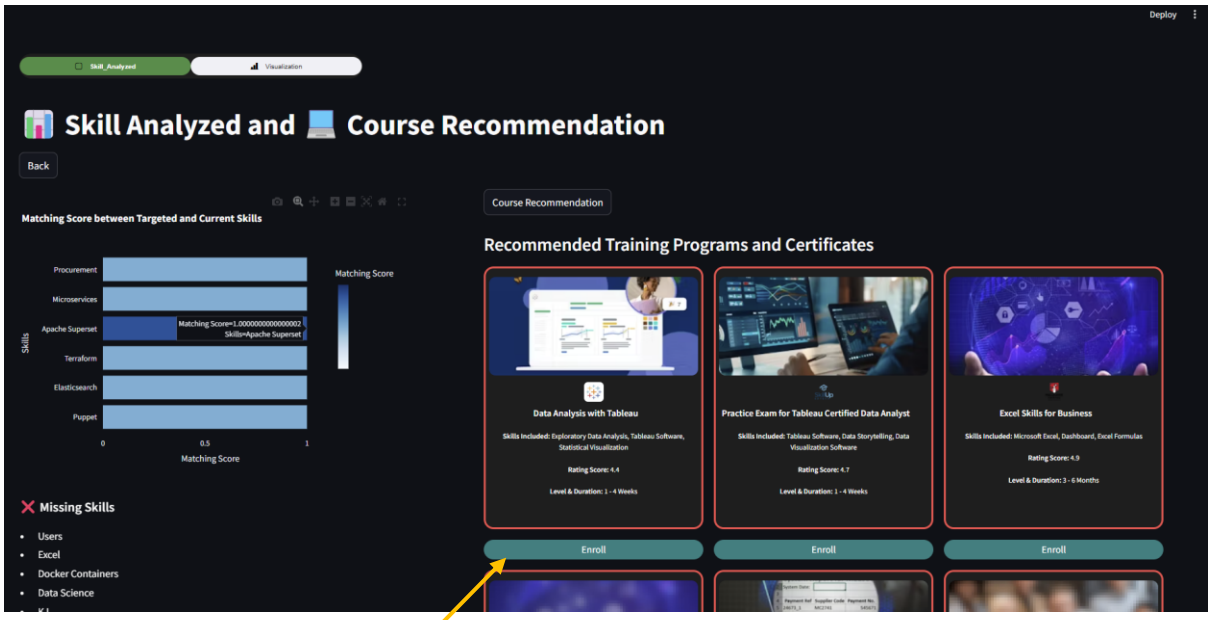
After click the button “Course Recommendation”, it will show the top30 recommended courses.

Step12:



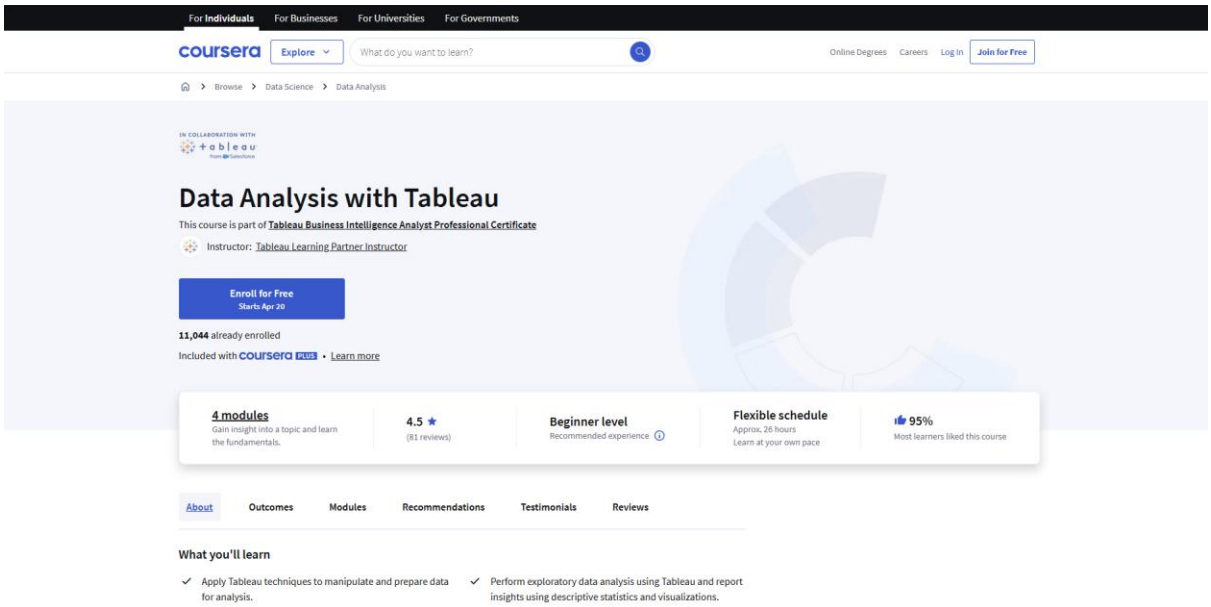
Scroll down and click the button “Next” to have more courses and max with 30 courses.

Step13:



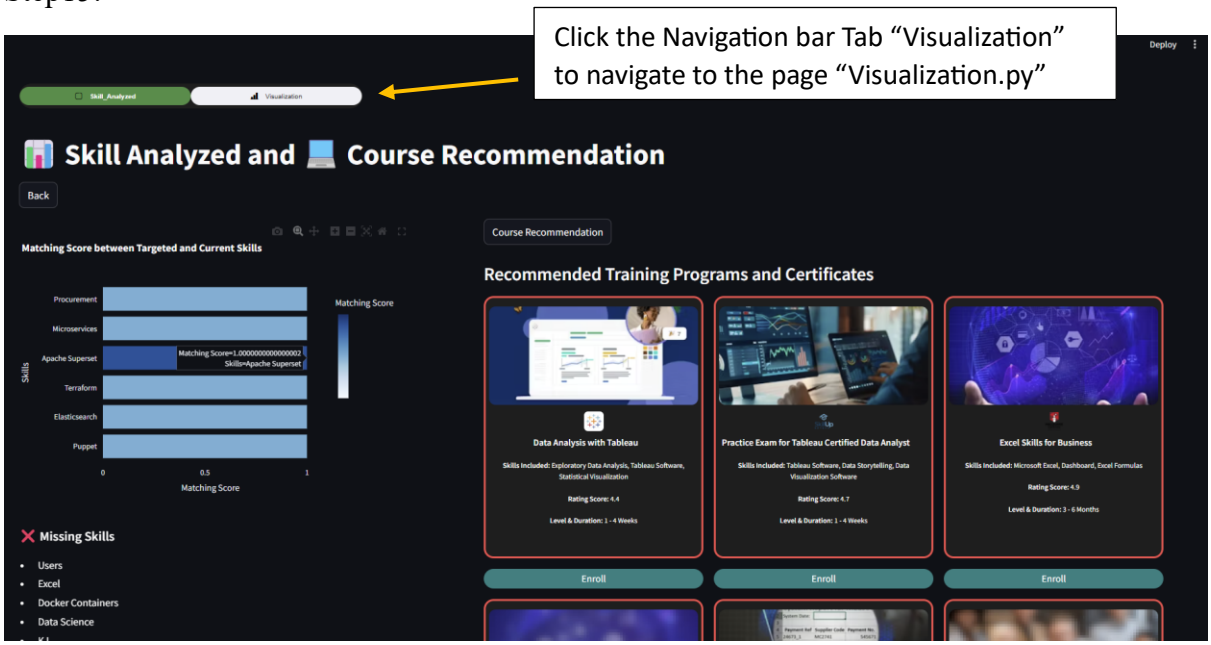
Click the button “Enroll” for the course that is interesting.

Step14:



After click the button “Enroll”, it will access to the specified course’s URL Link and go to the relevant website.

Step15:



Step16:

After click the Tab “Visualization.py”, it will navigate to the page “Visualization.py”. This page show 3 pie charts and 3 Bar Charts for the information of job market trend and analysis.

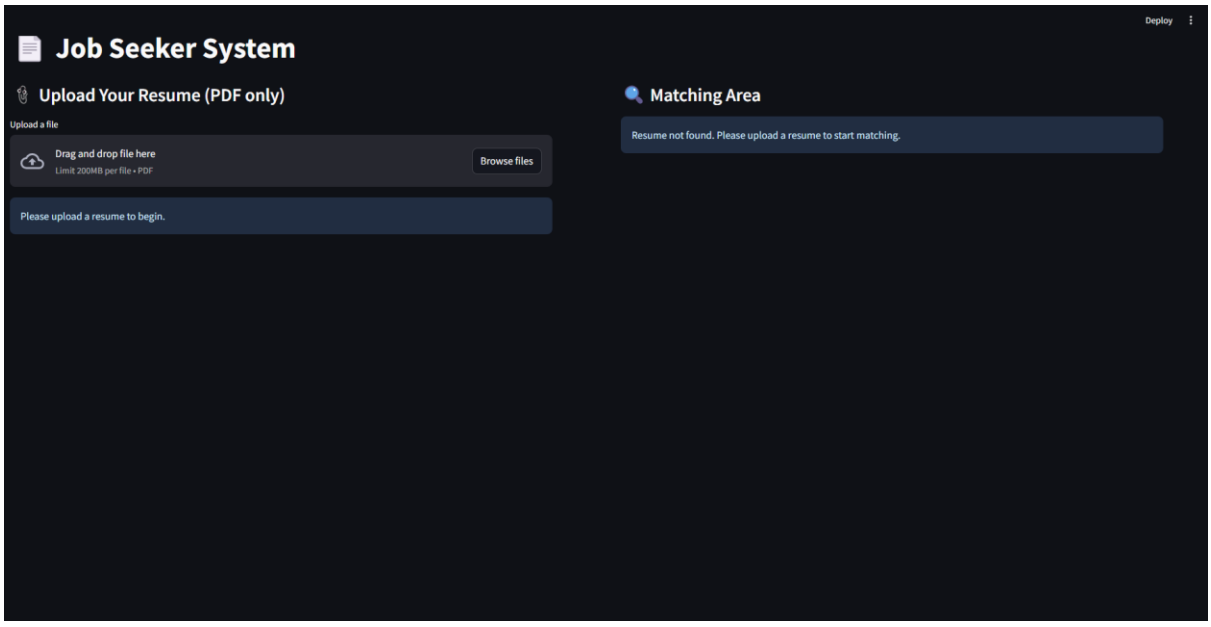
Step17:

Click the Navigation bar Tab “Skill_Analyzed” to navigate to the page “Skill_Analyzed.py” and back to Step11.

Step18:

Click “Reset” button and it will clear all and navigate to page “Job_Search.py” and back to

Step19:



After click the button “Reset”, it will back to the initial page and clean all of the selected functions.