CHAPTER 01
INTRODCUTION

1.1 FEEDBACK ON COMPANY

I recently completed the internship program offered by **INTERNZ LEARN**, focusing on Artificial Intelligence (AI), Python programming, and Machine Learning (ML), from the start, the team at Internz Learn provided an incredibly supportive and engaging environment. The structured learning modules, combined with hands-on projects, allowed me to quickly apply what I was learning. The course content was comprehensive, covering the essentials of Python programming, as well as foundational and advanced ML algorithms. The balance between theory and practice helped me deepen my understanding of these crucial technologies.

The mentorship was one of the highlights of this internship. My mentor not only guided me through complex concepts but also encouraged me to think critically about how to approach problem-solving. The feedback was always constructive, and there was a strong emphasis on self-paced learning, allowing me to grow at my own speed without feeling overwhelmed. I particularly appreciated the opportunity to work on real-world ML projects. The ability to contribute to actual use cases and collaborate with other interns and team members in a professional setting was invaluable. It gave me a clear sense of how AI and ML are implemented in business environments, enhancing my overall skill set.

The internship also provided excellent exposure to industry-standard tools and libraries such as TensorFlow, Keras, and scikit-learn, which I've already started applying in my personal projects. I feel much more confident in my Python programming abilities and have developed a solid foundation in machine learning that I am eager to build upon.

Overall, I am extremely grateful for this internship experience and would highly recommend it to anyone looking to break into the fields of AI and Machine Learning. The knowledge, skills, and practical experience I've gained during this time have significantly boosted my career prospects. Thank you to Internz Learn for providing such an enriching and rewarding internship and looking for next step of my career.

1.2 PROJECT DESCRIPTION

AI-based resume parsing is the process of using artificial intelligence (AI) and machine learning algorithms to scan and analyze resumes. This process helps employers quickly identify the best candidates for the job by analyzing relevant information on a resume, such as job titles, skills, qualifications, and experience. AI-based resume parsing helps employers easily identify the most qualified candidates for the job by automatically sorting resumes based on the set criteria.

HR professionals frequently use CV and resume parsing tools to automate data collection, import, and analysis. Deep learning and artificial intelligence are used by the resume parser tool to determine all required information fields from submitted profiles. Moreover, it analyses , stores, organizes, and enhances data using commonly used fitted taxonomies.

Typically, recruiters receive candidate profiles (CVs/resumes) in Word or PDF format. Resumes are typically ambiguous, may be written in multiple languages, and their formatting differs from resume to resume. As a result, when looking for a candidate, a recruiter must examine the entire pool of resumes.

On average, well, almost 90% of resumes do not match the recruiter's particular criteria. However, in order to select the most viable candidate, each resume should be reviewed, regardless of the time required. This is where the CV parsing tool comes in; recruiters can now parse CVs with pinpoint accuracy in record time and reach a hiring decision.

Furthermore, using a resume parsing tool nullifies manual data entry and provides employers with a larger pool of candidates from which to choose. With the help of a resume parser, recruiters can also pinpoint the most relevant candidates and take the necessary action.

1.2.1 Features

- A good resume parser should have the following characteristics:
- Capability to parse various CV formats.
- Maintain a comprehensive skill taxonomy for identifying candidates.
- Handles CVs and candidate profiles in multiple languages.
- Capability to extract all necessary candidate information from as many data fields as possible.
- Make a resume summary for recruiters to read in order to help them evaluate the candidate.
- Using deep learning technology, extract and better identify resume data.
- Bulk import candidate profiles (CVs/resumes) and parse multiple profiles at the same time.
- Integration with email inboxes to directly parse resumes from inboxes

1.2.2 Problem Statement

In today's competitive job market, job seekers are often required to create resumes that stand out in order to secure interviews. However, many candidates struggle with creating optimized resumes that accurately reflect their skills, experiences, and qualifications in a way that resonates with recruiters and hiring managers. Additionally, there are varying resume formats, industry-specific standards, and applicant tracking systems (ATS) that need to be considered.

Problem: Job seekers often face challenges when crafting a resume that effectively highlights their strengths, adheres to industry standards, and is optimized for ATS. Without expert guidance, candidates may submit resumes that are poorly formatted, lack critical information, or fail to align with job descriptions, reducing their chances of landing an interview.

1.3 BENEFITS

Even though most recruiters prefer to manually review resumes to ensure that candidates' applications are thoroughly screened, information may occasionally slip through the cracks. In such cases, AI-based resume screening tools can be extremely beneficial. AI-assisted resume screening can thus provide numerous benefits, including:

1. Data automation

Automated resume screening enables the company of hiring data to be automated. It assists in tracking and storing critical data such as resumes, application forms, and the source from which this data was collected, making it easier to create an applicant database. This talent pool could also be utilized in the future to help with screening and finding qualified candidates.

2. Enhanced candidate experience

Employing AI in resume screening can make the recruitment process more efficient, resulting in improved communication between recruiters and candidates. The experience of a candidate is also important because it determines whether or not the candidate wants to work for a specific organization. As a result, it is crucial to enhance the candidate experience.

3. Reducing unconscious bias

Unconscious bias is quite prevalent and can have an impact on the selection process's outcomes. However, AI in resume screening tools can provide a common base for judging different candidates and thus generate judgments based on common grounds. When there is no bias in the selection of candidates, it becomes easier to include diversity.

4. ROI (Return on Investment)

When considering purchasing a resume parsing tool, you must ensure that you will receive a substantial return on investment (ROI). If the tool has a lot of features, it is probably better for you. When you invest resources in a less expensive structure, you may discover that it does not perform satisfactorily. You might get a better ROI if you spend a little bit more.

5. Keyword Search

A resume parser that filters data through keywords can better parse CVs. Structured formats, such as Candidate Smart Cards, contain data divided into categories such as “Experience,” “Education,” “Skills,” and so on.

6. Feature for Job Applications

The process of resume parsing in recruitment can be streamlined by allowing candidates to apply for positions directly through the parsing tool. This will also improve the candidate’s experience with the organization because job applications will be much easier to complete. The resume parser can do its work much more effectively because the product requests data in a specific structure.

7. Easy comparison of applicants

It is much easier to compare resumes from different candidates when you have all of the candidate information in one place using resume parsing and analysis software. For example, you could access the experience tag and remove anyone who has no experience. The remaining candidates are qualified for the position. This is much easier than physically going through each candidate and looking for the section that reveals the candidate’s skills.

8. Systematic candidate formats

Candidates typically submit resumes in a variety of formats, making it difficult for recruiters to review some formats that are readily available to them. This could lead to the loss of qualified candidates. However, a resume parsing platform will remove these barriers, and all candidate data will be accessible in standardized formats that can be modified to suit the needs of the company.

CHAPTER 02
LITERATURE SURVEY

2.1 EXISTING SYSTEM

Before the introduction of AI-based resume simulation tools, job seekers used a variety of traditional methods to create and optimize their resumes. These methods were generally manual, time-consuming, and lacked the automated analysis and real-time feedback that modern AI tools offer. The key approaches included:

1. Traditional Resume Writing Guides: Job seekers relied on books and manuals for resume-writing tips, following standardized formats and content guidelines. These resources were valuable but lacked the personalization or automation needed for optimizing resumes for specific job roles.

2. Resume Templates: Word processing software and printed templates provided a simple structure for resumes, helping users organize their information. However, these templates didn’t suggest specific improvements or tailor resumes to job descriptions.

3. Professional Resume Writing Services: Job seekers could hire professional resume writers for customized, industry-specific advice. While these services provided tailored resumes, they were often costly and depended heavily on human expertise.

4. Job Portals with Basic Resume Builders: Platforms like Monster and Indeed offered basic resume-building tools that allowed users to create digital resumes. However, these tools were not designed for deep customization or optimization based on job descriptions or ATS compatibility.

5. Manual Keyword Matching: Job seekers would manually research job descriptions and add relevant keywords to their resumes to improve their chances with Applicant Tracking Systems (ATS). This process was tedious and lacked real-time guidance.

6. Job Coaching and Networking: Candidates often sought feedback from mentors, peers, or industry professionals through networking. While valuable, this feedback was subjective and dependent on the quality of the network.

Before AI-based resume simulation tools, the process of creating a resume was much more manual and subjective. Job seekers relied on templates, professional resume writers, personal networking, and manual strategies like keyword matching to improve their chances in the job market. These methods were effective to a degree but lacked the efficiency, personalization, and real-time analysis provided by AI-powered tools today.AI-based resume tools, with their ability to analyze job descriptions, optimize for ATS systems, and offer personalized feedback, have revolutionized the resume creation process by making it faster, more accurate, and aligned with the ever-evolving requirements of the modern job market.

2.2 PROPOSED SYSTEM

The working process behind an AI Based Screening Tool involves several key steps:

- Data collection,
- Pre-processing
- Feature extraction
- Similarity computation
- Ranking

1. Data Collection and Loading

Input: Resume Data (CSV)

In the first step, the tool loads the resume data (typically stored in a CSV file), where each resume is expected to have key information such as name, resume text, skills, experience, etc.

Function: `load_resume_data('resume.csv')

This function reads the CSV file into a Pandas DataFrame, which serves as a structured representation of the resumes.

Output: DataFrame of resumes

The data is now stored in a structured format (`resume_df`), which makes it easier to manipulate and access each resume's content for processing.

2. User Input for Job Description

Input: Job Description Text (User Input)

The user submits a job description via a web form on the homepage (`index.html`).

Form: Job Description

This form captures the textual content of the job description, which is then passed to the backend for processing. The user's job description serves as the baseline for comparing the resumes.

Output: Raw Job Description Text

This raw text input is typically a free-form paragraph or bullet points describing the role, responsibilities, and requirements of the job.

3. Text Preprocessing

Function: `preprocess_job_description(job_description)

Objective: Clean and preprocess the raw job description text. This step is crucial to reduce noise and standardize the format of text for effective analysis.

Typical Tasks Involved in Preprocessing:

- **Tokenization**: Splitting the job description into individual words or tokens.
- **Stopword Removal:** Removing common words like "the", "and", "a", etc., that do not contribute to meaningful analysis.
- **Lowercasing:** Converting all text to lowercase to avoid treating the same word as different because of capitalization (e.g., "Python" and "python").
- **Lemmatization/Stemming**: Reducing words to their base or root form (e.g., "running" becomes "run").

Output: Preprocessed job description text ready for feature extraction.

4. Feature Extraction (Text Vectorization)

Function: extract_features(resume_df['cleaned_resume'], job_desc_cleaned)

Objective: Convert the raw text (job description and resumes) into numerical feature vectors, which can be used for mathematical analysis and comparison.

Typical Tasks in Feature Extraction:

- **TF-IDF Vectorization (or similar methods like Word2Vec, Doc2Vec):**
  - TF-IDF (Term Frequency-Inverse Document Frequency)**: This method calculates the importance of each word in a document based on its frequency relative to its occurrence across the entire corpus of documents.
- **Word Embeddings:** In some advanced systems, word embeddings (e.g., using Word2Vec or BERT) might be used to capture semantic meaning and relationships between words.

Steps:

1. Resume Feature Extraction: Convert all cleaned resumes into feature vectors.

2. Job Description Feature Extraction: Similarly, convert the cleaned job description into a feature vector.

Output:

A set of vectors representing each resume (`resume_features`), and one vector for the job description (`job_desc_feature`).

5. Similarity Calculation

Function: compute_similarity(resume_features, job_desc_feature)

Objective: Quantify the similarity between each resume and the job description to rank them effectively.

Methods for Similarity Calculation

Cosine Similarity: This is a common method for measuring similarity between two vectors. The cosine of the angle between two vectors gives a measure of how similar they are.

Euclidean Distance: An alternative approach where the Euclidean distance between two vectors is computed. A smaller distance indicates greater similarity.

Dot Product: Another approach that computes the dot product between the resume vector and the job description vector.

Steps:

Compare each resume's feature vector with the job description's feature vector using one of the similarity metrics.

Output:

A list or array of similarity scores that indicates how closely each resume matches the job description.

6. Ranking Resumes

Function: rank_candidates(resume_df, similarity_scores.flatten())

Objective: Rank all resumes based on their similarity to the job description and return the top candidates.

Steps:

1. Sort the resumes by similarity score: Higher similarity scores indicate a better match to the job description.

2.Filter or Select the Top N: Depending on the application, you can select the top N resumes (e.g., top 5, top 10).

Output:

- A DataFrame or list of the top-ranked resumes, including their name and similarity score.

7. Displaying Results

Output: Top Ranked Resumes

Once the top-ranked resumes are determined, they are rendered on the `results.html` page, which is shown to the user.

Display: A table or list showing the candidate's name and their similarity score, allowing the user to see which resumes are the best match for the job description.

2.2.1 Full Workflow:

1. User Input:

- User submits a job description via a web form on the homepage.

2. Job Description Pre-processing:

- The submitted job description is cleaned and pre-processed to remove noise (e.g., stopwords, special characters, etc.).

3. Feature Extraction:

- Both resumes (from CSV) and the cleaned job description are transformed into numerical feature vectors using a text vectorization technique like TF-IDF.

4. Similarity Computation:

- Each resume's feature vector is compared to the job description's feature vector using a similarity metric (such as cosine similarity).

5.Ranking:

- Resumes are ranked based on their similarity scores to the job description, and the top candidates are selected.

6.Displaying Results:

- The top-ranked resumes are displayed in a tabular format, showing the name of the candidates and their respective similarity scores.

2.2.2 Theoretical Underpinnings:

This system relies on standard **Natural Language Processing (NLP)** and **Machine Learning (ML)** techniques for text comparison:

Text Representation: Converting text into numerical form (using TF-IDF, word embeddings, or other techniques) is fundamental in transforming human-readable text into something that an algorithm can understand and process.

Similarity Metrics: The core of the resume ranking process lies in measuring how similar a candidate's resume is to the job description. Techniques like cosine similarity allow the system to quantify this similarity mathematically.

Ranking Algorithms: Once similarity scores are computed, ranking the resumes based on these scores is a simple, but effective way to present the most relevant candidates to the user.

CHAPTER 03

CODING AND TESTING .

3.1 Hardware Requirements:

- Processor : 4th generation Intel Core i3 processor or higher
- Speed : 2.2GHz
- RAM : 4 GB
- Disk Space : 16 GB

3.2 Software Requirements Operating system

- Windows 10 and above
- Coding Language : Python 3.11.2
- Software : VScode
- Development environments : Python IDE
- Web framework: Flask

3.3 Prerequisites:

- Flask for building the web application.
- Scikit-learn for implementing machine learning techniques (e.g., TF-IDF vectorization and cosine similarity).
- Pandas for handling resume data.
- HTML templates for displaying the results.

3.4 Step-by-Step Code Implementation

1. Flask Application Setup (app.py): This is the main Flask application file that runs the web server
2. HTML Templates:
   a. **index.html** (Home Page with Form to Submit Job Description):This is the page where users will input the job description.
   b. **results.html** (Page to Display Ranked Resumes):This page displays the top-ranked resumes based on their similarity to the provided job description.
3. Modal training setup to train the modal (modal.py)
4. CSV File (resume.csv):The resumes are stored in a CSV file named resume.csv with at least two columns: name and resume.

3.5 Coding

app.py file

```
from flask import Flask, render_template, request, redirect, url_for
import pandas as pd
from model import load_resume_data, preprocess_job_description,
extract_features, compute_similarity, rank_candidates

app = Flask(__name__)

resume_df = load_resume_data('resume.csv')

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/rank_resumes', methods=['POST'])
def rank_resumes():
    if request.method == 'POST':
        job_description = request.form['job_description']

        job_desc_cleaned = preprocess_job_description(job_description)


        resume_features, job_desc_feature =
extract_features(resume_df['cleaned_resume'], job_desc_cleaned)


        similarity_scores = compute_similarity(resume_features,
job_desc_feature)


        ranked_resumes = rank_candidates(resume_df,
similarity_scores.flatten())


        top_resumes = ranked_resumes[['name',
'similarity_score']].head(5)
        return render_template('results.html',
resumes=top_resumes.to_dict(orient='records'))

if __name__ == "__main__":
    app.run(debug=True)
```

modal.py

```python
import pandas as pd

import re
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

def clean_text(text):
    text = re.sub(r'\W', ' ', str(text))
    text = re.sub(r'\s+', ' ', text)
    text = text.lower()
    return text

def load_resume_data(file_path):
    resume_df = pd.read_csv(file_path)
    resume_df['cleaned_resume'] = resume_df['resume'].apply(clean_text)
    return resume_df


def preprocess_job_description(job_desc):
    job_desc_cleaned = clean_text(job_desc)
    return job_desc_cleaned


def extract_features(resumes, job_desc, max_features=3000):
    tfidf = TfidfVectorizer(max_features=max_features)
    resume_features = tfidf.fit_transform(resumes)
    job_desc_feature = tfidf.transform([job_desc])
    return resume_features, job_desc_feature

def compute_similarity(resume_features, job_desc_feature):
    similarity_scores = cosine_similarity(resume_features, job_desc_feature)
    return similarity_scores

def rank_candidates(resume_df, similarity_scores):
    resume_df['similarity_score'] = similarity_scores
    ranked_resumes = resume_df.sort_values(by='similarity_score',
ascending=False)
    return ranked_resumes
```

index.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Resume Screening Tool</title>
</head>
<body>
    <h2>AI-based Resume Screening Tool</h2>
    <form method="POST" action="/rank_resumes">
        <label for="job_description">Enter Job Description:</label><br>
        <textarea id="job_description" name="job_description" rows="10"
cols="50"></textarea><br><br>
        <input type="submit" value="Submit">
    </form>
</body>
</html>
```

results.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Resume Screening Results</title>
</head>
<body>
    <h2>Top Ranked Resumes</h2>
    <table border="1">
        <tr>
            <th>Candidate Name</th>
            <th>Similarity Score</th>
        </tr>
        {% for resume in resumes %}
        <tr>
            <td>{{ resume['name'] }}</td>
            <td>{{ resume['similarity_score'] }}</td>
        </tr>
        {% endfor %}
    </table>
    <br>
    <a href="/">Back to Home</a>
</body>
</html>
```

3.5.1 Explanation of the Code

Flask Routes:

- The index() route renders the form where the user enters a job description.
- The rank_resumes() route handles form submission, processes the input, and ranks resumes based on their similarity to the job description.

Text Preprocessing:

- Both the job description and the resumes are preprocessed by converting all text to lowercase (you can extend this with more preprocessing techniques like stemming, lemmatization, etc.).

Feature Extraction with TF-IDF:

- We use the TfidfVectorizer from Scikit-learn to convert both the job description and resumes into TF-IDF vectors. This vectorization process helps in capturing the importance of words relative to the entire dataset, giving a numerical representation of the text.

Cosine Similarity:

- The similarity between the job description vector and each resume vector is calculated using cosine similarity. This method compares the direction of two vectors, providing a measure of similarity between the job description and each resume.

Ranking:

- Resumes are ranked based on the computed similarity scores. The resumes with the highest similarity to the job description are placed at the top.

Displaying Results:

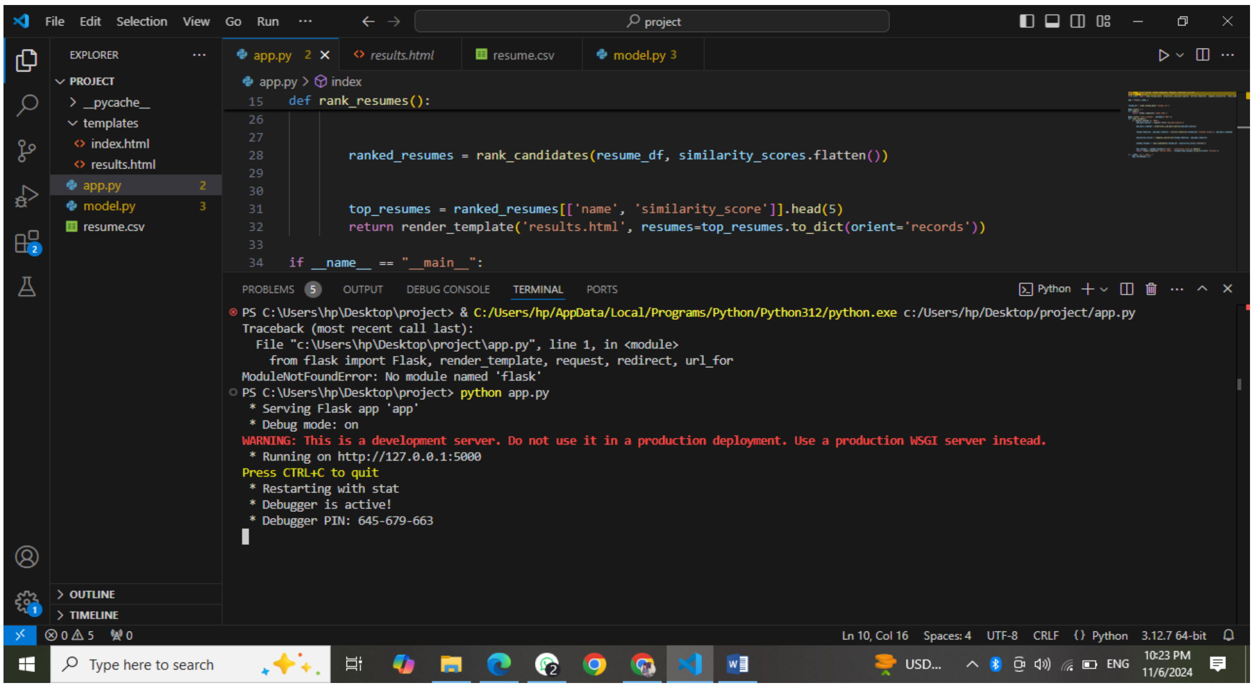
- The top 5 ranked resumes are displayed in a table on the results page, showing the candidate’s name and the similarity score.

3.5.2 Running the Flask Application

i. Save your Python script as app.py.
ii. Create the resume.csv file with your sample resumes.
iii. Save the HTML templates in a folder named templates.
iv. Run the application using the following command “**python app.py**”
v. Open your browser and go to **http://127.0.0.1:5000/** to interact with the app

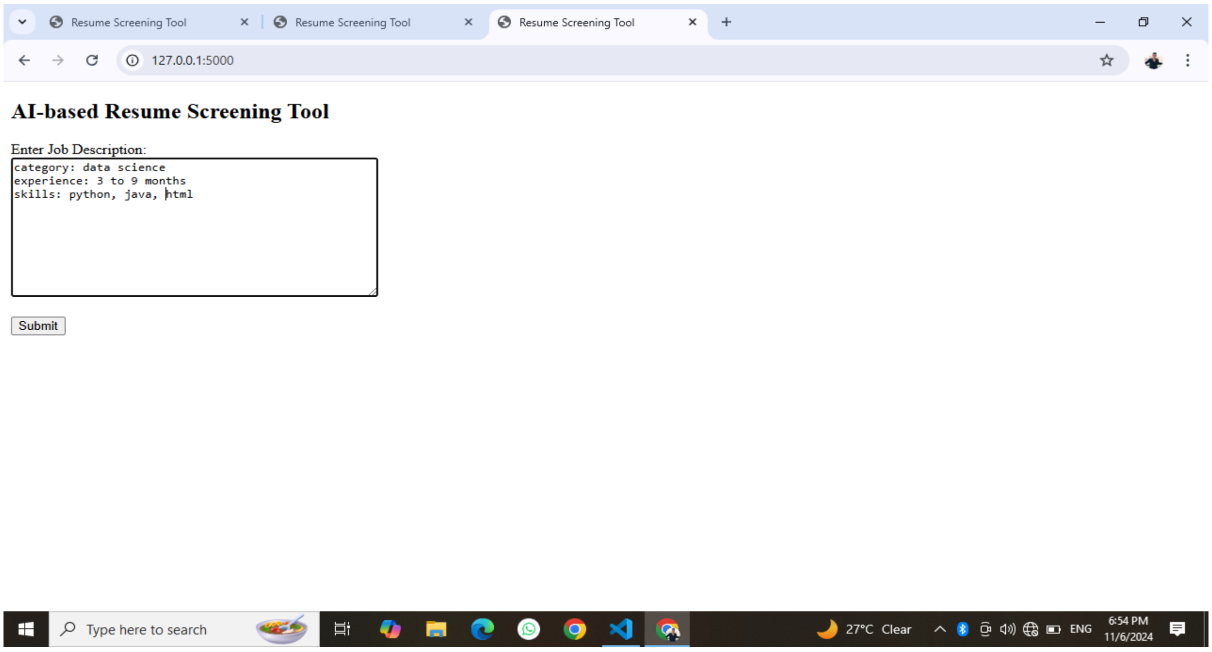
CHAPTER 04

RESULTS


```
def rank_resumes():


        ranked_resumes = rank_candidates(resume_df, similarity_scores.flatten())


        top_resumes = ranked_resumes[['name', 'similarity_score']].head(5)
        return render_template('results.html', resumes=top_resumes.to_dict(orient='records'))

if __name__ == "__main__":
```

```
PS C:\Users\hp\Desktop\project> & C:/Users/hp/AppData/Local/Programs/Python/Python312/python.exe c:/Users/hp/Desktop/project/app.py
Traceback (most recent call last):
  File "c:\Users\hp\Desktop\project\app.py", line 1, in <module>
    from flask import Flask, render_template, request, redirect, url_for
ModuleNotFoundError: No module named 'flask'
PS C:\Users\hp\Desktop\project> python app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 645-679-663
```

Fig:- screenshot of running Flask application to generate URL in terminal

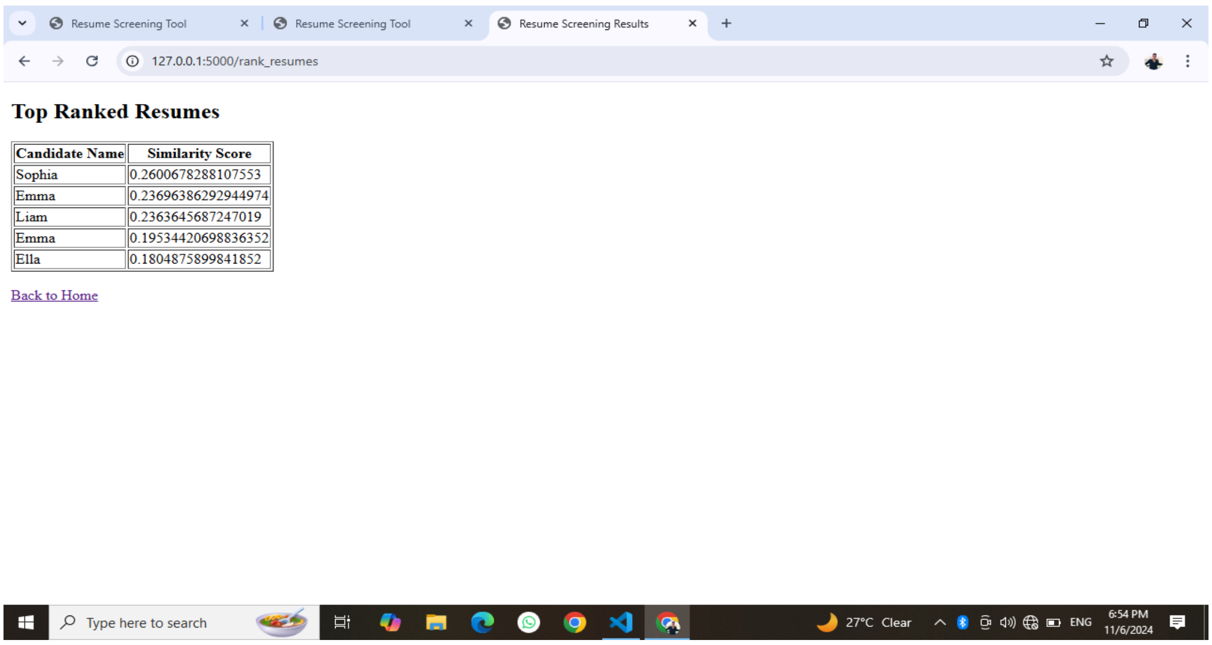

AI-based Resume Screening Tool

Enter Job Description:

```
category: data science
experience: 3 to 9 months
skills: python, java, html
```

Submit

Fig:- screenshot of index.html page



Top Ranked Resumes

Candidate Name	Similarity Score
Sophia	0.2600678288107553
Emma	0.23696386292944974
Liam	0.2363645687247019
Emma	0.19534420698836352
Ella	0.1804875899841852

Back to Home

Fig:- screenshot of results.html

CHAPTER 05

FUTURE ENHANCEMENT

Future enhancement:

1. Advanced Text Processing & NLP

Enhance resume and job description analysis with lemmatization, stemming, and NER (Named Entity Recognition) to extract skills, job titles, and locations.

Use advanced **semantic models** like **BERT** for better contextual understanding and matching beyond simple keyword comparisons.

2. Skills & Competency Matching

Extract and match specific skills between resumes and job descriptions.

Implement competency models that consider both soft (e.g., communication) and hard skills (e.g., programming) for more comprehensive candidate evaluation.

3. Improved Similarity & Matching

Use deep learning models (like BERT) for semantic similarity and job title matching.

Add natural language query capabilities for recruiters to search using conversational queries (e.g., "Find candidates with experience in Python and machine learning").

4. Resume Parsing & Enrichment

Automate resume parsing to extract structured data (e.g., contact info, skills, experience).

Provide AI-powered resume enrichment with suggestions for improving resumes based on job descriptions.

5. Personalized Recommendations & Feedback

Offer personalized feedback to candidates on how to optimize their resumes.

Suggest career paths or roles based on the candidate's skill set and experience.

6. Scalability & User Experience

Implement a cloud-based infrastructure for scaling the tool.& enhance the user interface with an **interactive dashboard** for managing resumes, rankings, and job descriptions.

CHAPTER 06

CONCLUSION

The AI-based resume ranking tool presents a promising solution for improving the recruitment process by leveraging machine learning and NLP techniques to automate the matching of resumes to job descriptions. While the current implementation is effective for basic ranking tasks, there is significant potential for enhancement through advanced NLP models, personalized feedback mechanisms, and integration with other recruitment tools.

By continuously refining the tool’s algorithms, improving data handling capabilities, and incorporating more sophisticated AI techniques, this project has the potential to greatly enhance recruitment efficiency, reduce hiring bias, and help both employers and candidates make more informed decisions.

The AI-based resume ranking tool offers an innovative solution to streamline the recruitment process by automating the matching of resumes to job descriptions using machine learning and natural language processing (NLP). While the current implementation effectively ranks resumes based on text similarity, there are opportunities for enhancement, such as integrating advanced NLP models like BERT for better semantic understanding, incorporating skill and competency matching, and providing personalized feedback to candidates. Future improvements could also focus on real-time matching, bias detection, and integration with other HR tools like Applicant Tracking Systems (ATS). With these enhancements, the tool can further improve recruitment efficiency, reduce bias, and support data-driven decision-making, making it a valuable asset for both employers and job seekers.

6.1 Limitations and Challenges

- Basic Similarity Measures
- Static Job Matching
- Resume Data Quality

CHAPTER 07

BIBLOGRAPHY

- https://techrseries.com/ai-automation/ai-based-resume-parsing-and-analytics-tools-a-complete-guide/
- https://enhancv.com/ai-resume-builder/?utm_source=google&utm_medium=cpc&utm_campaign=google_performance_max_india&gad_source=1&gclid=CjwKCAiAxKy5BhBbEiwAYiW--5LrCvvq7JPgb_CgcuGwY-m5QVWkHBzX9hqhE9XXvuJs0EfJZlsUCBoCrN8QAvD_BwE
- https://www.livecareer.co.uk/lp/lcukrsmmb02.aspx?utm_source=google&utm_medium=sem&utm_campaign=21719551189&utm_term=how%20to%20make%20up%20a%20resume&network=g&device=m&adposition=&adgroupid=168450578318&placement=&adid=713972055094&gad_source=1&gclid=CjwKCAiAxKy5BhBbEiwAYiW---Mh1DjF8GwOCmHg1KpQx05i7S-whto4FTl6VGVMGJIdRb1-ivelshoC5KsQAvD_BwE
- https://github.com/deepakpadhi986/AI-Resume-Analyzer
-