Resume Analysis System for Skill-Based Candidate Management

Group number: 8

Aarushi Garg: 225890388

Ishita Singh: 225890344

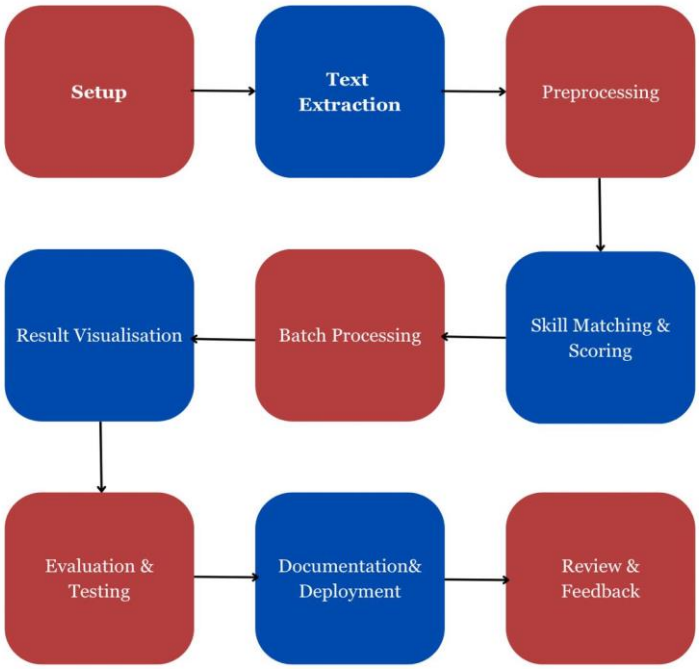
B L Sunayana: 225890048

Objective:

The primary objective of this project is to design and implement an automated system for extracting, analysing, and scoring resumes based on their alignment with a predefined set of required skills. By leveraging Natural Language Processing techniques and machine learning tools, the system aims to efficiently process resumes in both PDF and Word formats, tokenize the content, and evaluate each candidate's proficiency in key skills such as Python, machine learning, data analysis, and other relevant technical competencies.

The system generates a quantitative score for each resume, reflecting the percentage of required skills matched, which allows recruiters to quickly identify the most qualified candidates. Additionally, the project aims to provide clear and insightful visualizations, such as a pie chart distribution of scores, to assist hiring managers in understanding the overall skill levels of applicants. This automated analysis enhances the resume screening process, reducing manual effort and enabling more data-driven recruitment decisions.

Workflow Diagram:



Step-by-Step Work Plan:

1) Define Project Scope and Objectives:

- Clearly outline the project's objectives, including automating resume analysis and skill-based scoring.
- Identify the required skills to evaluate (e.g., Python, machine learning, data analysis).

2) Set Up the Development Environment:

- Install necessary libraries and tools such as NLTK, PyPDF2, Scikit-learn, Pandas and Matplotlib.
- Ensure compatibility with both PDF and Word file formats for resume input.

3) Data Extraction from Resumes:

- Implement a function to extract text from PDF files using PyPDF2 and from Word documents using python-docx.
- Develop a method to handle different resume formats and ensure accurate text extraction.

4) Data Preprocessing:

- Preprocess extracted resume text by tokenizing it into words, converting text to lowercase, and removing irrelevant characters or stopwords.
- Use NLTK for tokenization and cleaning the resume data.

5) Skill Matching and Scoring:

- Create a predefined list of required skills for matching (e.g., Python, TensorFlow, NumPy).
- Implement a CountVectorizer to compare the resume content against the required skills.
- Develop a scoring algorithm to calculate a percentage score based on how many of the required skills are present in the resume.

6) Batch Resume Analysis:

- Write a script to iterate over multiple resumes in a directory.
- For each resume, extract the text, score the skills, and store the results (filename and score) in a structured format like a Pandas DataFrame.

7) Data Visualization:

- Create a pie chart to visualize the distribution of resume scores using Matplotlib.
- Define score ranges (e.g., 80-100%, 60-80%, etc.) and represent how many resumes fall into each range, making it easier for recruiters to evaluate candidates at a glance.

8) Testing:

- Test the system with a variety of resumes to ensure that the text extraction and skill matching processes are working as expected.

Expected outcome:

Each resume will be scored based on how well it matches a predefined set of required skills, with a percentage score reflecting the number of skills a candidate possesses. This will allow recruiters to prioritize high-scoring candidates for further review.

The project will produce visual representations, such as pie charts, that display the distribution of resume scores across different ranges (e.g., 80-100%, 60-80%, etc.). This will provide hiring managers with a clear, at-a-glance view of the overall skill levels of candidates.

Code Snippets:

```
import os
import pandas as pd
import numpy as np
import nltk
from sklearn.feature_extraction.text import CountVectorizer
import matplotlib.pyplot as plt
from PyPDF2 import PdfReader
from docx import Document
import tkinter as tk
from tkinter import simpledialog

✓ 6.5s
```

```python
# Function to extract text from resumes (both PDF and DOCX)
def extract_text(file_path):
    if file_path.endswith('.pdf'):
        # Extract text from a PDF file
        text = ''
        with pdfplumber.open(file_path) as pdf:
            for page in pdf.pages:
                page_text = page.extract_text()
                if page_text:
                    text += page_text
        return text
    elif file_path.endswith('.docx'):
        # Extract text from a DOCX file
        doc = docx.Document(file_path)
        return '\n'.join([para.text for para in doc.paragraphs])
    else:
        return None
```

```python
# Function to score the resume based on required skills
def score_resume(resume_text, required_skills):
    resume_text = resume_text.lower()
    tokens = word_tokenize(resume_text)
    found_skills = [skill for skill in required_skills if skill in tokens]
    score = len(found_skills) / len(required_skills) * 100  # Calculate score as percentage
    return score, found_skills
```

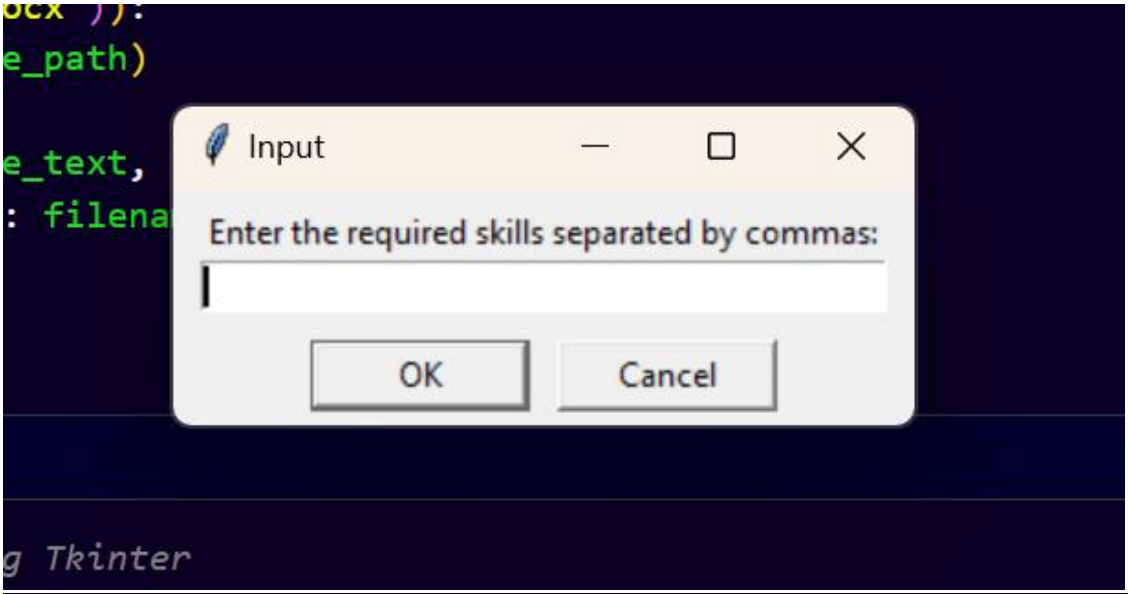
✓ 0.0s

```python
# Function to analyze all resumes in a directory
def analyze_resumes(resume_dir, required_skills):
    results = []
    for filename in os.listdir(resume_dir):
        file_path = os.path.join(resume_dir, filename)
        if file_path.endswith(('.pdf', '.docx')):
            resume_text = extract_text(file_path)
            if resume_text:
                score = score_resume(resume_text, required_skills)
                results.append({'Filename': filename, 'Score': score})
    return pd.DataFrame(results)
```

✓ 0.0s

```python
# Function to prompt for skills input using Tkinter
def get_required_skills():
    root = tk.Tk()
    root.withdraw()  # Hide the main window
    skills = simpledialog.askstring("Input", "Enter the required skills separated by commas:")
    return [skill.strip() for skill in skills.split(',')] if skills else []
```

✓ 0.0s




```python
# Specify the directory containing resumes
resume_directory = r'C:\Users\sunay\OneDrive\Documents\INFORMATION-TECHNOLOGY\INFORMATION-TECHNOLOGY'

# Get required skills from user
required_skills = get_required_skills()
```

```python
# Analyze resumes
df_results = analyze_resumes(resume_directory, required_skills)

# Display the results sorted by score
df_sorted = df_results.sort_values(by='Score', ascending=False)
df_sorted
```

```python
# Define score ranges and labels
score_ranges = {
    '80-100%': df_sorted[df_sorted['Score'] >= 80].shape[0],
    '60-80%': df_sorted[(df_sorted['Score'] >= 60) & (df_sorted['Score'] < 80)].shape[0],
    '40-60%': df_sorted[(df_sorted['Score'] >= 40) & (df_sorted['Score'] < 60)].shape[0],
    '<40%': df_sorted[df_sorted['Score'] < 40].shape[0]
}
```

```python
# Create pie chart
labels = score_ranges.keys()
sizes = score_ranges.values()

# Define a gradient color palette
colors = ['#ff9999', '#66b3ff', '#99ff99', '#ffcc99']

# Explode the first slice
explode = (0.1, 0, 0, 0)

# Create the pie chart
plt.figure(figsize=(10, 8))
wedges, texts, autotexts = plt.pie(sizes, explode=explode, labels=labels, colors=colors,
                                   autopct='%1.1f%%', shadow=True, startangle=140,
                                   textprops=dict(color="w", fontsize=14))

# Customize the font size and color of the percentage text
for text in autotexts:
    text.set_color("black")
    text.set_fontsize(16)

# Add a title with a larger font size
plt.title('Resume Analysis: Distribution of Candidates by Score Ranges', fontsize=20, fontweight='bold')

# Create a legend
plt.legend(wedges, labels, title="Score Ranges", loc="center left", bbox_to_anchor=(1, 0, 0.5, 1), fontsize=12)

plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.tight_layout()  # Adjust layout to make room for the legend
plt.show()
```