AI RESUME SCREENING SYSTEM

(Using Python, AI & Flask)

1️⃣ INTRODUCTION

In today's recruitment process, companies receive hundreds or thousands of resumes for a single job opening. Manually checking each resume is time-consuming and inefficient. To solve this problem, many companies use **Applicant Tracking Systems (ATS)** that automatically filter resumes based on job requirements.

This project **"AI Resume Screening System"** is designed to simulate such an ATS using **Artificial Intelligence and Natural Language Processing (NLP)** techniques.

The system automatically analyzes a candidate's resume and compares it with a given job description, then provides:

- Skill match percentage
- AI similarity score
- Selection decision (Selected / Not Selected)

2️⃣ OBJECTIVE OF THE PROJECT

The main objectives of this project are:

- To automate resume screening using AI
- To reduce manual effort in resume evaluation
- To compare resume content with job description
- To calculate skill matching accuracy
- To provide a user-friendly web interface using Flask

---

3 SOLUTION APPROACH

This project solves the problem using:

- Text Cleaning
- Skill Extraction
- Skill Matching
- Score Calculation
- Final Decision Logic

This project is useful for:

- HR teams
- Recruitment agencies
- Job portals
- Educational projects
- AI & Python learners

The system supports:

- Resume upload in **PDF format**
- Job description input
- AI-based text comparison
- Web-based output display

---

4 TECHNOLOGIES USED

◇ Programming Language

- **Python** – Core programming language

◇ Framework

- **Flask** – Web framework for backend and UI interaction

◇ Libraries

- **PyPDF2** – Extract text from resume PDF
- **Scikit-learn** – TF-IDF & cosine similarity

- **OS & Re** – File handling and text cleaning

◇ Frontend

- **HTML** – Structure
- **CSS** – Styling and layout

---

5 SYSTEM ARCHITECTURE

Workflow:

1. User uploads resume PDF
2. User enters job description
3. Resume text is extracted
4. Text is cleaned and normalized
5. Skills are matched
6. AI similarity score is calculated
7. Result is displayed on web page

---

6 MODULE DESCRIPTION

Module 1: Resume Upload

- User uploads resume in PDF format
- File is saved temporarily
- PyPDF2 extracts text from resume

Module 2: Job Description Input

- Job description is entered via web form
- Converted into plain text
- Used for comparison

Module 3: Text Preprocessing

- Converts text to lowercase
- Removes symbols and unwanted characters
- Makes text machine-readable

Module 4: Skill Matching

- Predefined list of skills (Python, SQL, HTML, CSS, etc.)
- Checks how many skills appear in resume
- Calculates skill match percentage

Module 5: AI Similarity Calculation

- Measures similarity between resume and job description
- Used TF-IDF for measuring
- Cosine Similarity

Module 6: Decision Engine

- Combines skill match & AI score
- Generates final result:
    - ✅ Selected
    - ❌ Not Selected

7️ ALGORITHM USED

◇ TF-IDF (Term Frequency – Inverse Document Frequency)

- Measures importance of words
- Ignores common words
- Highlights relevant skills

◇ Cosine Similarity

- Measures similarity between two texts
- Returns a score between 0 and 1
- Converted into percentage

8️⃣ FLASK IMPLEMENTATION

Flask is used to:

- Create a web server
- Handle file uploads
- Process resume & job description
- Display output dynamically

Flask Flow:

- `app.py` → main backend
- `/` route → homepage
- `/upload` route → processing
- HTML templates → UI
- CSS → styling

```
import re


# ================================
# STEP 1: READ FILE
# ================================
def read_file(file_name):
    with open(file_name, "r") as file:
        return file.read()
```

```
# ================================
# STEP 2: TEXT CLEANING
# ================================
def clean_text(text):
    text = text.lower()
    text = re.sub(r'[^a-z ]', '', text)
    return text
```

```python
# ================================
# STEP 3: MASTER SKILL DATABASE
# ================================
MASTER_SKILLS = [
    "python", "java", "c", "c++", "sql",
    "html", "css", "javascript",
    "flask", "django",
    "react", "node",
    "machine learning", "data science",
    "git", "github", "docker"
]
```

```python
# ================================
# STEP 4: EXTRACT REQUIRED SKILLS
# ================================
def extract_job_skills(job_text, master_skills):
    job_skills = []
    for skill in master_skills:
        if skill in job_text:
            job_skills.append(skill)
    return job_skills
```

```python
# ==============================
# STEP 5: MATCH RESUME SKILLS
# ==============================
def match_skills(resume_text, job_skills):
    matched = []
    missing = []

    for skill in job_skills:
        if skill in resume_text:
            matched.append(skill)
        else:
            missing.append(skill)

    return matched, missing
```

```python
# ==============================
# STEP 6: FINAL DECISION
# ==============================
def final_decision(matched, required):
    score = len(matched) / len(required) * 100

    if score == 100:
        return score, "✅ SELECTED (100% Skill Match)"
    else:
        return score, "❌ REJECTED"
```

```
# ==============================
# MAIN PROGRAM FLOW
# ==============================
job_text = clean_text(read_file("job_description.txt"))
resume_text = clean_text(read_file("resume.txt"))

job_skills = extract_job_skills(job_text, MASTER_SKILLS)

if not job_skills:
    print("⚠️ No skills found in Job Description")
    exit()
```

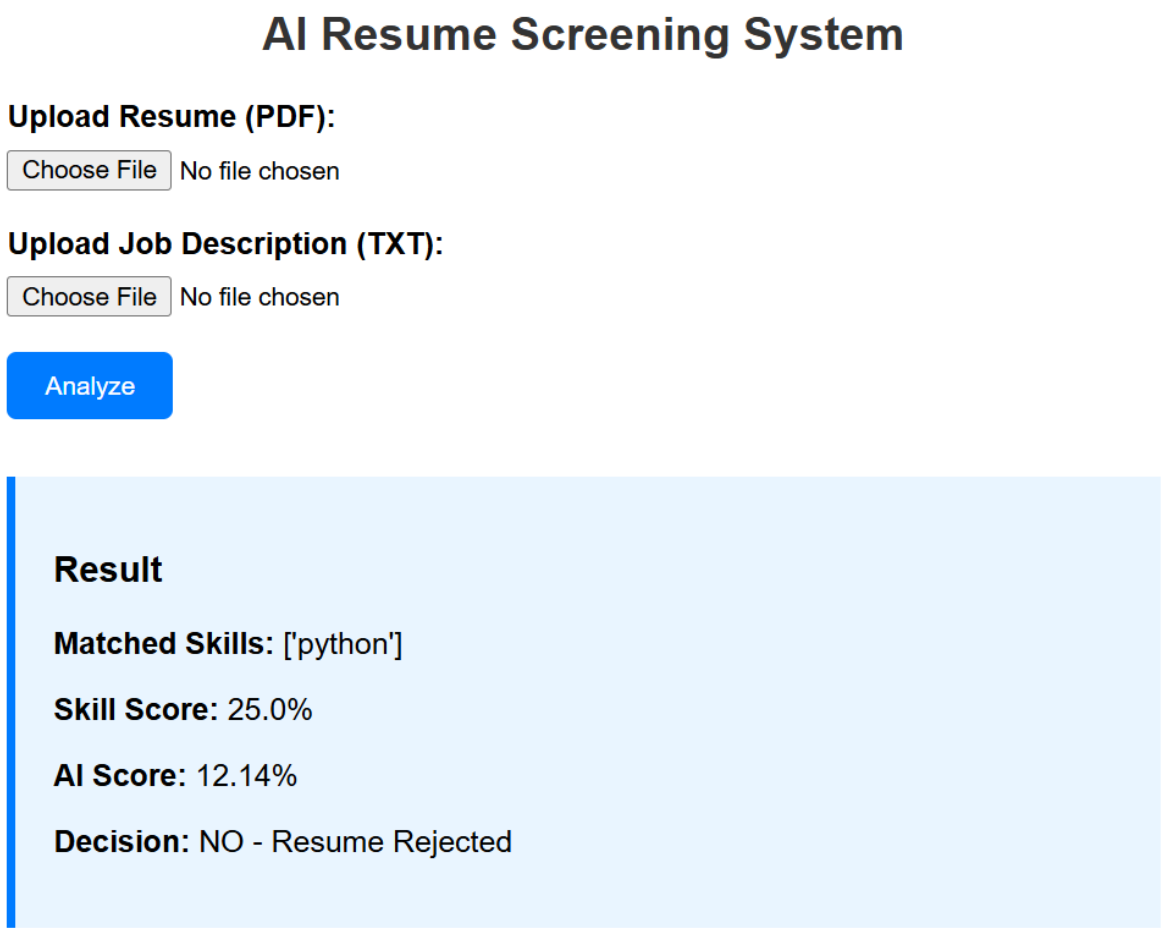
```
matched, missing = match_skills(resume_text, job_skills)
score, decision = final_decision(matched, job_skills)

print("\n📌 Job Required Skills:", job_skills)
print("✅ Matched Skills:", matched)
print("❌ Missing Skills:", missing)
print(f"\n🎯 Final Decision: {decision}")
print(f"📊 Match Score: {score:.2f}%")
```

---

9 OUTPUT SCREEN

The final output displays:

- Uploaded Resume Name
- Matched Skills
- Skill Match Percentage
- AI Similarity Score
- Final Selection Status



🔟 RESULTS AND ANALYSIS

- Accurate matching based on skills
- AI score varies based on resume content
- Demonstrates real-world ATS behavior
- Shows how AI assists HR decisions

ADVANTAGES

- Saves time
- Reduces human bias
- Fast processing

- User-friendly interface
- Scalable for multiple resumes

LIMITATIONS

- Depends on predefined skills
- Basic NLP (can be improved)
- Accuracy varies with resume format

FUTURE ENHANCEMENTS

- Use Machine Learning models
- Add resume ranking system
- Support multiple job roles
- Deploy on cloud
- Improve UI using React
- Use advanced NLP (BERT)

CONCLUSION

The **AI Resume Screening System** successfully demonstrates how Artificial Intelligence and Python can be used to automate resume evaluation. This project provides a strong foundation in AI, NLP, Flask, and web development, making it highly suitable for academic and professional use.