Build an AI Resume Screener with Python & Llama 3

Part 1: The Setup

Before writing code, we need to set up our environment. We are keeping this private and free by running the AI locally on your machine.

1. **Install Ollama:** First, we need an engine to run our AI model. Ollama is the easiest way to run open-source LLMs locally. Download it from ollama.com.
2. **Pull the Model:** Once installed, open your terminal/command prompt and run:

```
ollama pull llama3
```

3. **Install Python Libraries:** Next, we need Python to interact with Ollama and to read PDF files. Run this in your terminal:

```
pip install ollama pymupdf
```

Part 2: Building the Screener

We'll break the process into three digestible parts: **The Reader**, **The Brain**, and **The Execution**.

Step 1: The Reader

LLMs can't see PDF files directly; they need raw text. We will use `PyMuPDF` (imported as `fitz`) to strip the text layer from the document.

Data ingestion is often the messiest part of Data Science. If you feed garbage text (encoding errors, weird formatting) into the AI, you will get garbage results. This function iterates through every page and stitches the text together.

```
import fitz

def extract_text_from_pdf(pdf_path):
    doc = fitz.open(pdf_path)
    text = ""
    for page in doc:
        text += page.get_text()
    return text
```

Step 2: The Brain

This is the key part of our AI Resume Screener. We aren't just sending text; we are "prompt engineering." We will give the AI a persona (Senior Technical Recruiter) and specific constraints (JSON format).

Here, we tell the AI that it has 20 years of experience, which changes its behavior to be more critical and nuanced. We also explicitly ask for a valid JSON format only, which is crucial for using the data programmatically later.

```
import ollama

def screen_resume(resume_text, job_description):
    prompt = f"""
    You are a Senior Technical Recruiter with 20 years of experience.
    Your goal is to objectively evaluate a candidate based on the Job Description and their Resum
```

```
    JOB DESCRIPTION:
    {job_description}

    CANDIDATE RESUME:
    {resume_text}

    TASK:
    Analyze the resume against the JD. Look for specific keywords, experience levels, and nuance.
    Be strict but fair. "React" matches "React.js".

    OUTPUT FORMAT:
    Provide the response in valid JSON format only. Do not add any conversational text. Use this
    {{
        "candidate_name": "extracted name",
        "match_score": "0-100",
        "key_strengths": ["list of 3 key strengths"],
        "missing_critical_skills": ["list of missing skills"],
        "recommendation": "Interview" or "Reject",
        "reasoning": "A 2-sentence summary of why"
    }}
    """

    response = ollama.chat(model='llama3', messages=[
        {'role': 'user', 'content': prompt},
    ])

    return response['message']['content']
```

Step 3: The Execution

Now, we will define our standard (the Job Description), load our input (the Resume), and handle the output.

```
import json

# 1. Define the Job Description (The Standard)
job_description = """
We are looking for a Junior Data Scientist.
Must have:
- Experience with SQL
- Python (Pandas, NumPy, Scikit-Learn)
- Basic understanding of Machine Learning algorithms
- Good communication skills

Nice to have:
- Experience with AWS or Cloud deployment
- Knowledge of NLP
"""

# 2. Load the Resume (The Input)
# NOTE: Replace the path below with your actual PDF path
pdf_path = "/path/to/your/resume.pdf"

try:
    resume_text = extract_text_from_pdf(pdf_path)
    print(f"Resume loaded. Length: {len(resume_text)} characters.")
except Exception as e:
    print(f"Error loading resume: {e}")
    exit()

# 3. The Screening (The Processing)
print("AI is analyzing the candidate... (this may take a few seconds on local hardware)")
result_json_string = screen_resume(resume_text, job_description)

# 4. Parse and Display Results
try:
    # Sometimes LLMs wrap JSON in ```json blocks. We clean that up.
    clean_json = result_json_string.replace("```json", "").replace("```", "").strip()
    result_data = json.loads(clean_json)
```

```
    print("\n--- SCREENING REPORT ---")
    print(f"Candidate: {result_data.get('candidate_name')}")
    print(f"Score: {result_data.get('match_score')}/100")
    print(f"Decision: {result_data.get('recommendation')}")
    print(f"Reasoning: {result_data.get('reasoning')}")
    print(f"Missing Skills: {', '.join(result_data.get('missing_critical_skills', []))}")

except json.JSONDecodeError:
    print("Failed to parse JSON. Raw output:")
    print(result_json_string)
```

Example Output

Here is an example of what the output might look like when running the script:

```
Resume loaded. Length: 2802 characters.
AI is analyzing the candidate... (this may take a few seconds on local hardware)

--- SCREENING REPORT ---
Candidate: Aman Kharwal
Score: 92/100
Decision: INTERVIEW
Reasoning: Aman has demonstrated strong technical skills in machine learning and Python, making h
Missing Skills: SQL
```

Note: If your resume lists "Linear Regression" and "Random Forests," but the JD asks for "Machine Learning algorithms," a traditional keyword search might fail. Llama 3 understands that those are machine learning algorithms.

Closing Thoughts

This project teaches three critical GenAI skills:

1. **Context Window Management:** Understanding how much text (Resume + JD) fits into the prompt.
2. **Structured Output:** Forcing a creative writer (the LLM) to act like a database (JSON).
3. **Local Inference:** Running AI without sending private data to the cloud (crucial for GDPR and privacy in HR).