1. Problem Understanding and Proposed Solution

This project addresses two critical challenges in HR processes:

1. Resume Screening:
- Automate the filtering of resumes for the 'Software Engineer' role.
- Match candidate skills, qualifications, and experience with job descriptions using LLMs.

2. Employee Sentiment Analysis:
- Analyze employee feedback (surveys, exit interviews).
- Predict attrition risk and suggest engagement strategies using sentiment classification.

The solution utilizes prompt-based large language models (LLMs) like GPT-4 or Google Gemini, deployed vi

2. Workflow Diagram of AI Pipeline

The pipeline for both tasks follows a similar structure:

1. Input Collection: Resume files (.pdf/.docx) or feedback CSV.
2. Preprocessing: Extract text using Python (pdfplumber, python-docx).
3. Prompt Engineering: Custom prompts for LLMs to generate structured output.
4. LLM Inference: Use GPT-4 or Gemini for analysis.
5. Output: JSON results for fit score or sentiment analysis.
6. (Optional) Deployment: Expose as API endpoints via Azure/Google AI Studio.

3. Prompt Engineering Details

Resume Screening Prompt:

------------------------

You are an AI recruiter. Given the job description and candidate resume, evaluate the candidate’s suitability.
Output (JSON): Skills Match, Experience Match, Qualification Match, Fit Score, Summary.

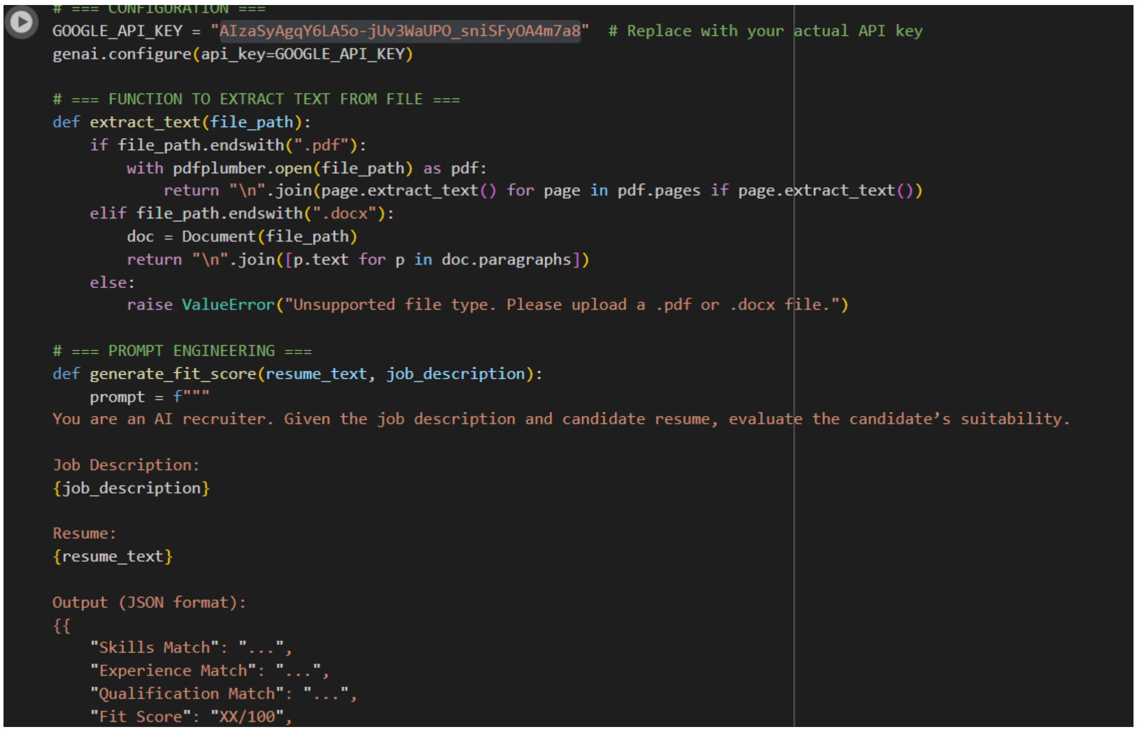
Sentiment Analysis Prompt:

-------------------------

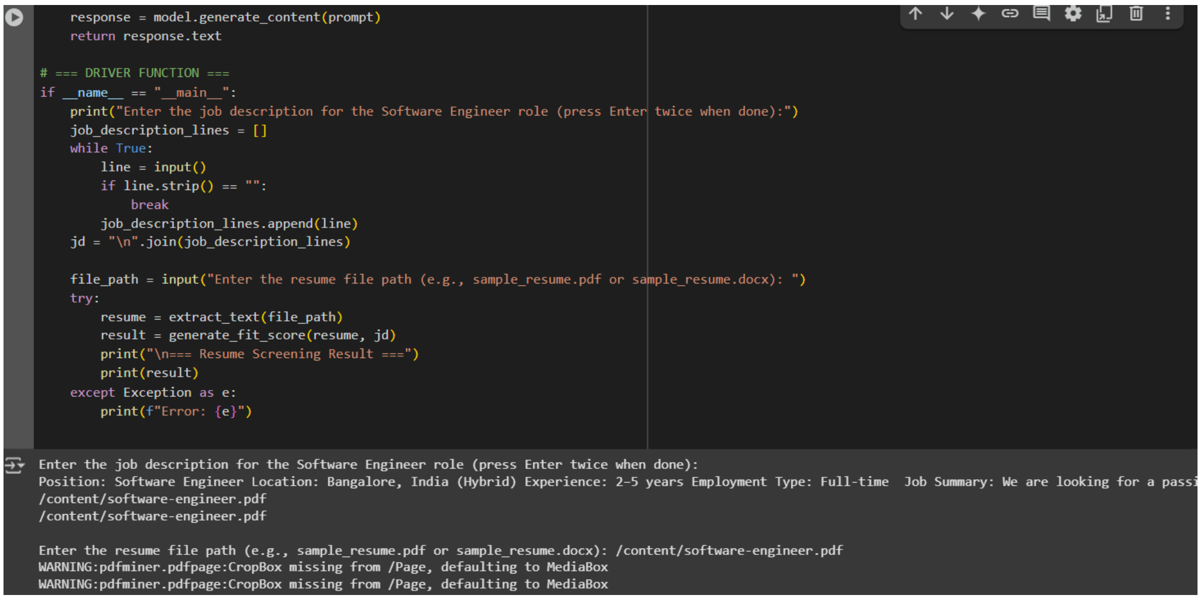
Analyze the feedback and classify the sentiment and attrition risk level.
Output (JSON): Sentiment, Risk Level, Key Concerns, Suggested Strategy.

4. Challenges Faced and Solutions

- File format compatibility: Resolved using `pdfplumber` and `python-docx` for consistent text extraction.
- Gemini output format: Handled using structured prompt engineering and careful temperature tuning.
- LLM response truncation: Minimized by limiting prompt length and using concise output formats.
- API deployment: Overcame Google AI Studio configuration issues by consulting official deployment guides

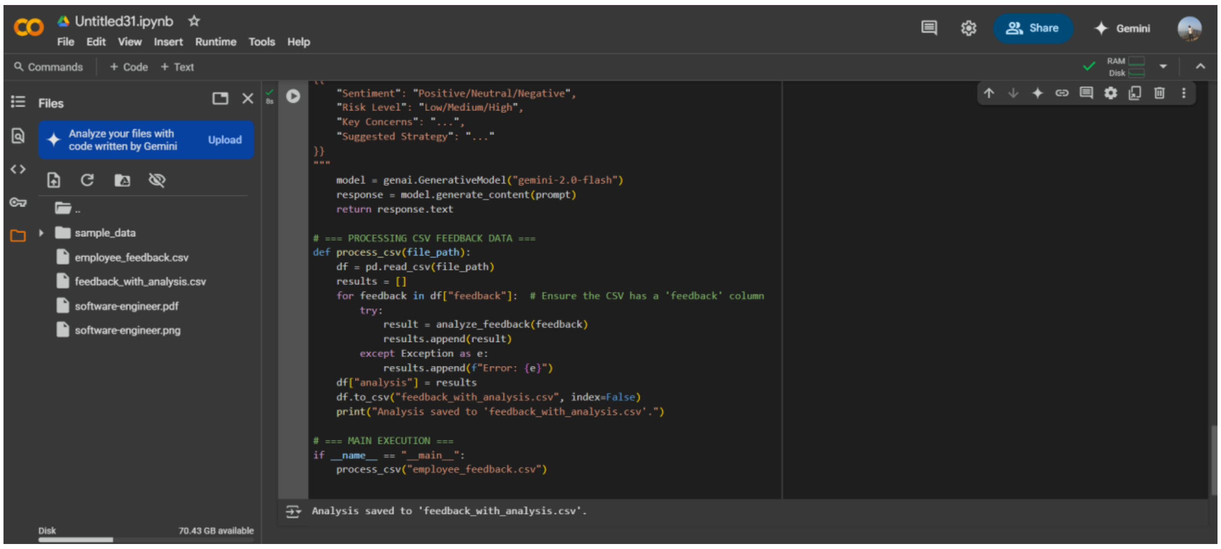


```
# === CONFIGURATION ===
GOOGLE_API_KEY = "AIzaSyAgqY6LA5o-jUv3WaUPO_sniSFyOA4m7a8"  # Replace with your actual API key
genai.configure(api_key=GOOGLE_API_KEY)

# === FUNCTION TO EXTRACT TEXT FROM FILE ===
def extract_text(file_path):
    if file_path.endswith(".pdf"):
        with pdfplumber.open(file_path) as pdf:
            return "\n".join(page.extract_text() for page in pdf.pages if page.extract_text())
    elif file_path.endswith(".docx"):
        doc = Document(file_path)
        return "\n".join([p.text for p in doc.paragraphs])
    else:
        raise ValueError("Unsupported file type. Please upload a .pdf or .docx file.")

# === PROMPT ENGINEERING ===
def generate_fit_score(resume_text, job_description):
    prompt = f"""
You are an AI recruiter. Given the job description and candidate resume, evaluate the candidate’s suitability.

Job Description:
{job_description}

Resume:
{resume_text}

Output (JSON format):
{{
    "Skills Match": "...",
    "Experience Match": "...",
    "Qualification Match": "...",
    "Fit Score": "XX/100",
```



```
    response = model.generate_content(prompt)
    return response.text

# === DRIVER FUNCTION ===
if __name__ == "__main__":
    print("Enter the job description for the Software Engineer role (press Enter twice when done):")
    job_description_lines = []
    while True:
        line = input()
        if line.strip() == "":
            break
        job_description_lines.append(line)
    jd = "\n".join(job_description_lines)

    file_path = input("Enter the resume file path (e.g., sample_resume.pdf or sample_resume.docx): ")
    try:
        resume = extract_text(file_path)
        result = generate_fit_score(resume, jd)
        print("\n=== Resume Screening Result ===")
        print(result)
    except Exception as e:
        print(f"Error: {e}")
```

```
Enter the job description for the Software Engineer role (press Enter twice when done):
Position: Software Engineer Location: Bangalore, India (Hybrid) Experience: 2–5 years Employment Type: Full-time  Job Summary: We are looking for a passi
/content/software-engineer.pdf
/content/software-engineer.pdf

Enter the resume file path (e.g., sample_resume.pdf or sample_resume.docx): /content/software-engineer.pdf
WARNING:pdfminer.pdfpage:CropBox missing from /Page, defaulting to MediaBox
WARNING:pdfminer.pdfpage:CropBox missing from /Page, defaulting to MediaBox
```



```
    "Sentiment": "Positive/Neutral/Negative",
    "Risk Level": "Low/Medium/High",
    "Key Concerns": "...",
    "Suggested Strategy": "..."
}}
"""
    model = genai.GenerativeModel("gemini-2.0-flash")
    response = model.generate_content(prompt)
    return response.text

# === PROCESSING CSV FEEDBACK DATA ===
def process_csv(file_path):
    df = pd.read_csv(file_path)
    results = []
    for feedback in df["feedback"]:  # Ensure the CSV has a 'feedback' column
        try:
            result = analyze_feedback(feedback)
            results.append(result)
        except Exception as e:
            results.append(f"Error: {e}")
    df["analysis"] = results
    df.to_csv("feedback_with_analysis.csv", index=False)
    print("Analysis saved to 'feedback_with_analysis.csv'.")

# === MAIN EXECUTION ===
if __name__ == "__main__":
    process_csv("employee_feedback.csv")
```

```
Analysis saved to 'feedback_with_analysis.csv'.
```