1. Project Overview

Objective: Automate HR processes using AI for resume screening and employee sentiment analysis.

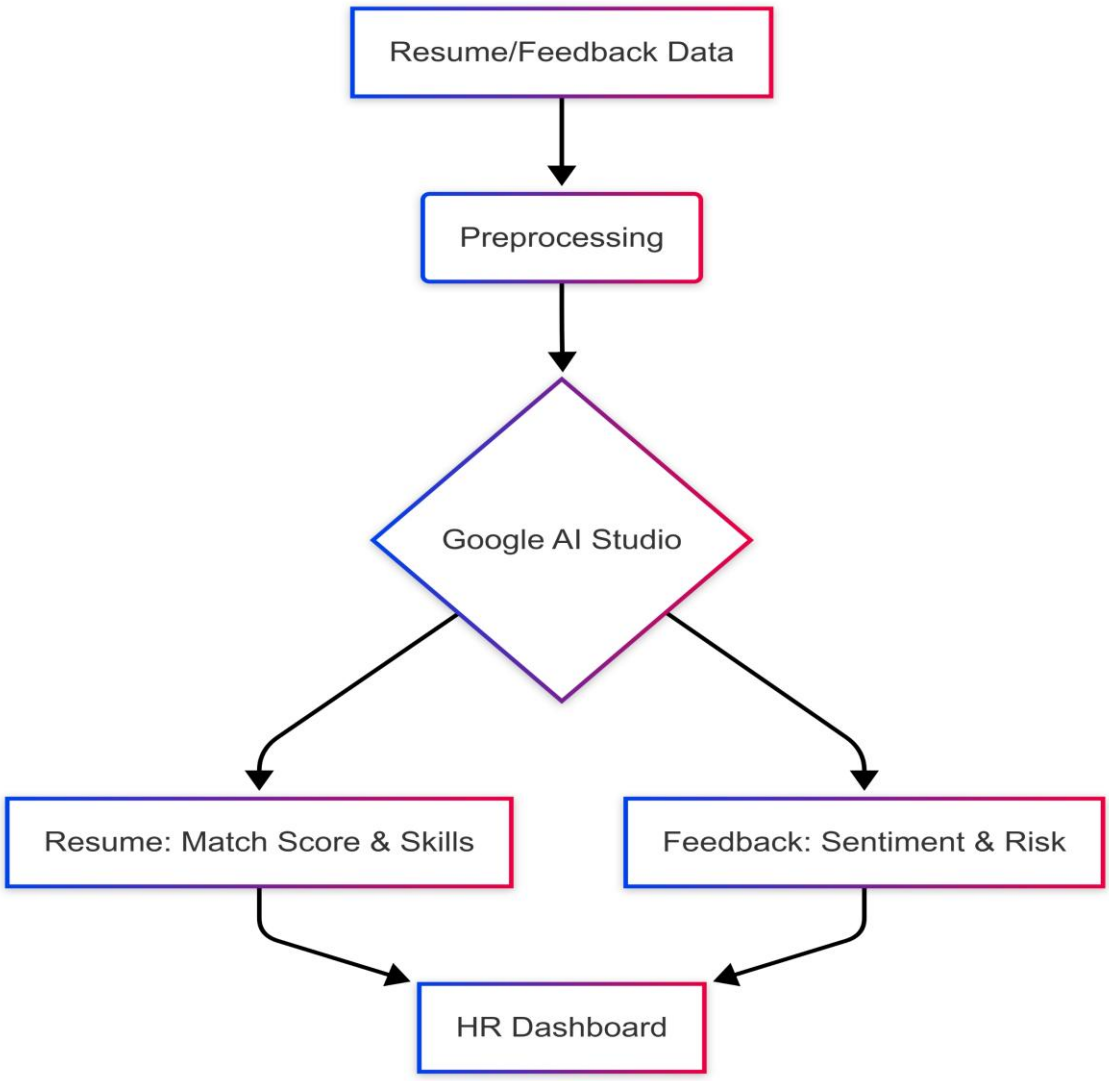
Tools:

- **Google AI Studio** (Gemini Pro) for LLM integration.
- **Python** for data processing and API development.
- **FastAPI** for local deployment (bonus).

---

2. Technical Approach

2.1 Workflow Diagram



2.2 Data Pipelines

- **Resume Screening**:
    1. **PDF Parsing**: Extract text using PyMuPDF.
    2. **Text Cleaning**: Remove irrelevant characters.
    3. **Prompt Engineering**: Structured input for LLM.
    4. **LLM Processing**: Match skills/experience with JD.
- **Sentiment Analysis**:
    1. **Text Normalization**: Lowercase, remove stopwords.
    2. **Prompt Design**: Classify sentiment and suggest strategies.
    3. **LLM Processing**: Generate JSON output.

---

3. Model Selection & Integration

- **LLM Choice**: Gemini Pro (Google AI Studio) for its 1M token context window.
- **Integration**:
    - Use Google AI Studio's API for prompt testing.
    - Simulate responses locally for rapid prototyping.

---

4. Prompt Engineering

4.1 Resume Screening Prompt

python

Copy

```
def build_resume_prompt(resume_text, job_desc):
    return f"""
    Role: Senior HR Analyst
    Task: Analyze the resume against the job description.
    Output Format: JSON with keys [matched_skills, missing_skills, match_score].

    Resume:
    {resume_text[:10000]}  # Truncate for token limits

    Job Description:
```

```
    {job_desc[:5000]}
    """
```

4.2 Sentiment Analysis Prompt

python

Copy

```
def build_sentiment_prompt(feedback):
    return f"""
    Role: Employee Engagement Expert
    Task: Analyze feedback for attrition risk and suggest improvements.
    Output Format: JSON with keys [sentiment, risk_score, strategies].

    Feedback:
    {feedback}
    """
```

---

5. Code Implementation

5.1 Resume Parser (resume_screening/parser.py**)**

python

Copy

```
import pdfplumber

def parse_pdf(file_path):
    with pdfplumber.open(file_path) as pdf:
        return " ".join([page.extract_text() for page in pdf.pages])
```

5.2 FastAPI Endpoint (api/main.py**)**

python

Copy

```
from fastapi import FastAPI
from pydantic import BaseModel

app = FastAPI()
```

```
class ResumeRequest(BaseModel):
    resume_text: str
    jd_text: str


@app.post("/screen")
async def screen_resume(request: ResumeRequest):
    # Simulated Gemini response
    return {
        "matched_skills": ["Python", "AWS"],
        "match_score": 85
    }
```

---

6. Deployment Steps

1. **Google AI Studio Setup**:
    - Create prompts and test with sample data.
    - Capture screenshots of inputs/outputs.
2. **Local API Testing**:

bash

Copy

```
uvicorn api.main:app --reload
```

    - Test via http://localhost:8000/docs.
3. **Cloud Deployment (Bonus)**:
    - Containerize with Docker.
    - Deploy to Google Cloud Run.

---

7. Technical Report Content

Sections:

1. **Introduction**: Problem statement and business impact.
2. **Methodology**: Workflow, data pipelines, model selection.
3. **Implementation**: Code snippets, prompt design.

4. **Results**: Screenshots from Google AI Studio.
5. **Challenges**: PDF parsing errors, prompt tuning.
6. **Conclusion**: Scalability and future work.