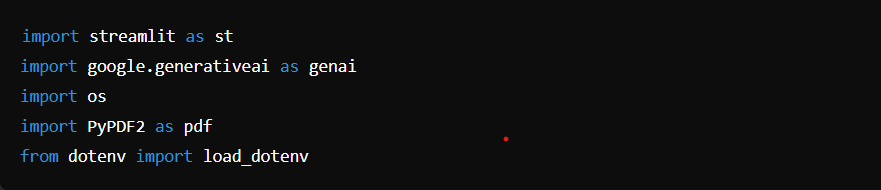
Ats-Optimized Resume Analyzer Using Gemini Model

**Step 1: Importing Required Libraries**

****

* **Streamlit (st)**: Used to create the web interface of the application, handling inputs, file uploads, buttons, etc.
* **Google Generative AI (genai)**: This library connects to Google Gemini Pro, used here for generating responses by analyzing resumes and job descriptions.
* **OS (os)**: Used to access environment variables, such as API keys, in a secure way.
* **PyPDF2 (pdf)**: This library is used to read and extract text from uploaded PDF resume files.
* **dotenv (load\_dotenv)**: Used to load environment variables (e.g., API keys) from a .env file.

**Step 2: Loading Environment Variables**

****

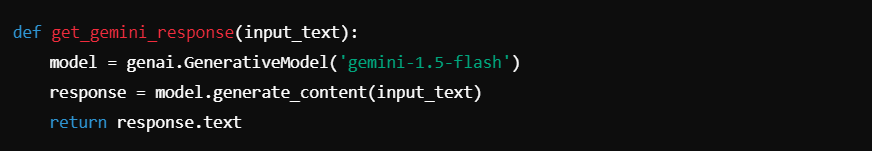
* **load\_dotenv()**: This function loads all the key-value pairs from a .env file into the environment. Here, it's used to retrieve the GOOGLE\_API\_KEY securely from the environment.

**Step 3:** **Configuring Google Generative AI**

****

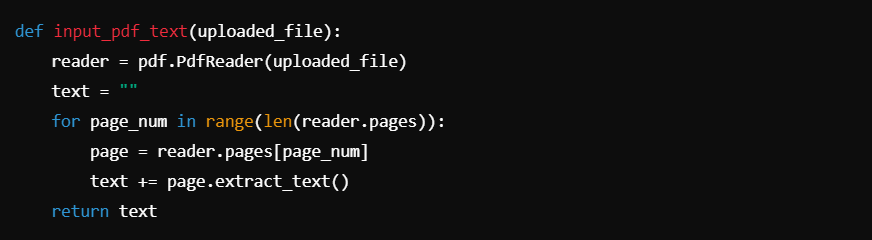
This step sets up the Google Generative AI (Gemini) API by configuring it with the API key, which is fetched from the environment using os.getenv("GOOGLE\_API\_KEY").

**Step 4: Gemini Pro Response Function**

****

* **get\_gemini\_response(input\_text)**:
* This function generates a response using the Gemini Pro AI model.
* **GenerativeModel('gemini-1.5-flash')**: Specifies the model version (gemini-1.5-flash) for the response generation.
* **generate\_content(input\_text)**: The AI model generates content based on the input prompt, which could be resume details and job descriptions.
* The function returns the response in text format.

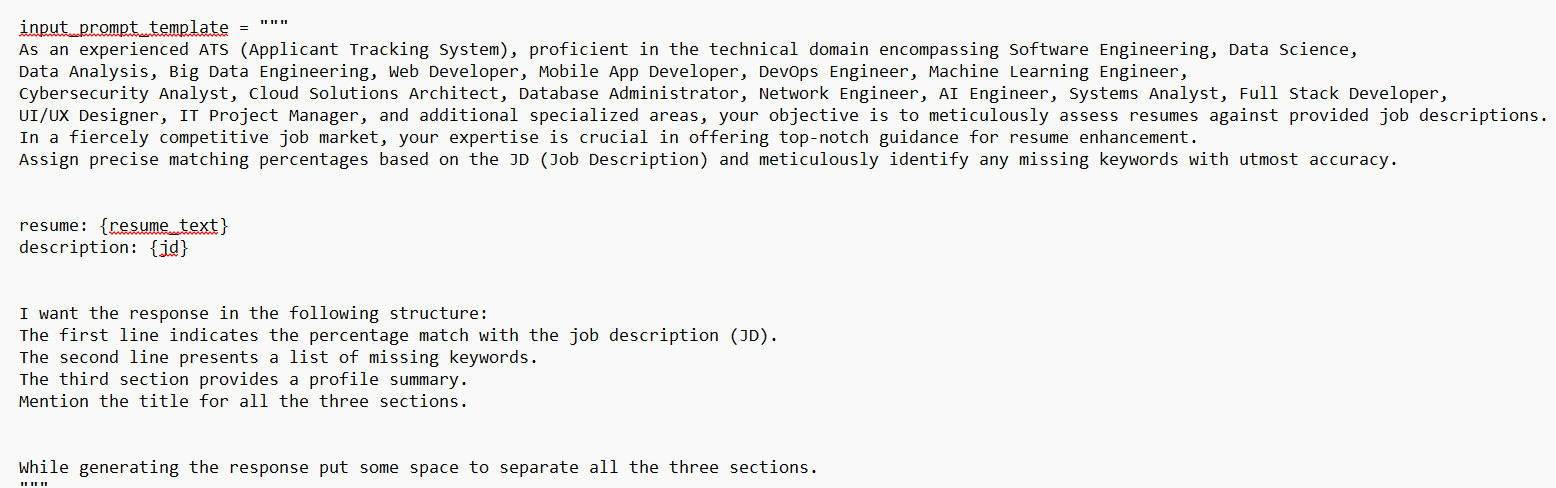
**Step 5:** **Extracting Text from Uploaded PDF**



**input\_pdf\_text(uploaded\_file)**:

* This function extracts all text from the uploaded PDF file.
* It initializes a PdfReader object to read the uploaded PDF and iterates through all pages using a loop.
* **extract\_text()**: Extracts text from each page and appends it to the text variable.
* The function returns the concatenated text extracted from the entire PDF file.

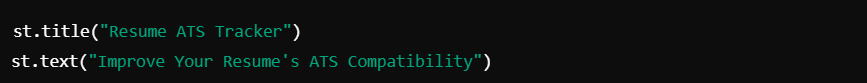
**Step 6:** **Prompt Template for Generative AI**



**input\_prompt\_template**: This is the structured prompt given to the AI for resume analysis. It guides the model on:

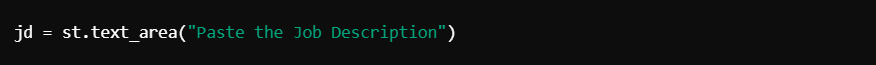
* The technical domains to focus on.
* How to assess the resume against the job description.
* The desired response format, which includes:
  + **Percentage match** with the job description.
  + **List of missing keywords**.
  + **Profile summary**.

**Step 7:** **Streamlit App Interface**



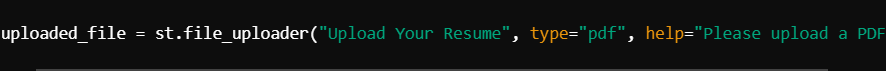
**Title and Description**: These lines set up the basic heading and description for the Streamlit app.

**Step 8:** **Job Description Input Field**

****

**st.text\_area()**: A text box for the user to input the job description, which will later be compared with the uploaded resume.

**Step 9:** **PDF Upload Field**

****

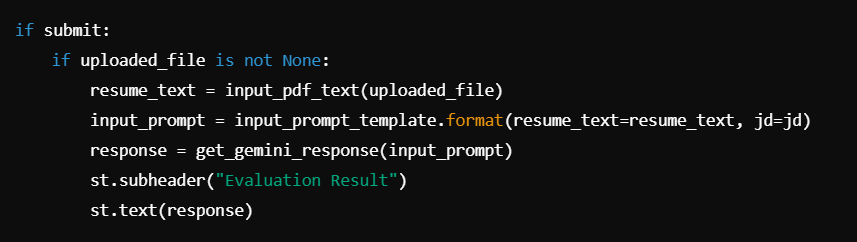
**st.file\_uploader()**: A file upload widget where users can upload their resume in PDF format.

**Step 10:** **Submit Button**

****

**st.button("Submit")**: A button that triggers the logic for resume analysis once clicked.

**Step 11:** **Handling Submission**

****

**Submit Logic**:

* When the submit button is clicked, it first checks if a resume (uploaded\_file) has been uploaded.
* If so, it extracts the resume's text using the input\_pdf\_text function.
* It then prepares a complete prompt (input\_prompt) using the extracted resume text and the provided job description (jd).
* **get\_gemini\_response()**: The prompt is sent to the Gemini AI model, and the response is returned.
* Finally, the result is displayed on the Streamlit app using **st.subheader()** and **st.text()**.