Mini Project Report

Amit Das

SI 568 Applied Data Science

April 18, 2023

Revised: May 14, 2025

AI Resume Analyzer & Summarizer

For this project, we were required to conceptualize potential real-world business problems and program AI-based solutions, using OpenAI's API (OpenAI, 2023) to experiment with those same challenges. This report will cover my general thought process and challenges, project timeline, strengths, weaknesses, included files within the repository, and any additional information regarding those files. These are generalized system requirements and sampled media inputs I used for testing. There were multiple milestones throughout the project, but I will also discuss a major challenge and project change that happened later on.

Project Overview

My initial plans for the project were to create an AI interface that could give food recommendations to assist in customer preparedness and wait times. I ran into some issues with this, so I stuck with creating a sorted chat transcript based on burgers, nuggets, and fries, with the initial question being limited to McDonald's. From this point, my project did not solve business problems as initially hoped, so I had to restructure my idea and thought process. The final project I completed and experimented with was an AI resource to assist applicants and employers in their job search and hiring process.

A potential problem within business I thought about was within the hiring process, which often allocates a significant amount of resources to find the best candidate. Similar things can be said for candidates, with the varying tasks people must do before applying. While it is general knowledge that there are many filtering systems in place today, I can understand the difficulty of the hiring process. With that being said, my project acted as an experiment to see if I could utilize OpenAI's API (OpenAI, 2023) within Python to innovate new or productive business tools within both sides of the hiring process. Granted, this is not a product, but a showcase of what I have learned. Below is a summary of the timeline of my project.

Timeline summary:

1. **March 25, 2024 - April 3, 2024:** Learn the AI documentation, retrieve the key, connect to the AI, and complete the original project for course feedback.
2. **April 17, 2024:** Updated the project to change my business problem. Created an interface to give applicants job roles that match their resumes, and offered chances to ask additional questions. Required me to research file handling limitations within the code.
3. **April 18, 2024:** Created a second interface for employers to input applicant resumes to retrieve summarized information. Cleaned up both interfaces and debugged errors.
4. **May 4th, 2025 (Revision):** Fixed an error with unnecessary nested while loops within the "ask_question" functions. Removed history / past message storage from the employer version, as it did not need to store past questions and answers.

Methodology

Script Overview

To implement the project, I created two separate Python scripts, "applicants.py" and "employers.py". The following are things both scripts have functionally in common: Both scripts use GPT-3.5 Turbo provided by OpenAI's API (OpenAI, 2023) essentially by default, but the users may modify the model used for personal testing. They can give step-by-step instructions and let the users exit via a quit function within the code. They can upload one-page PDF files through the “get_resume” function and loop under certain parameters, which it is satisfied. Whenever users send a message, they can enter a case-insensitive “exit” message, which allows them to leave, unless they manually exit. This can be done anytime, even during the file input stage. It is meant to experiment and give users full clarity and ease of access. You may find these files within the GitHub repository, while instructions on how to use them are in the “README.md” file. Some differentiating information about the two scripts is below:

- **Applicants Version (applicants.py):** This script is intended for applicants to gain feedback on their resumes by allowing them to input PDF files, receive instant and tailored job suggestions, then directly chat with the AI model (in command-line) with the previous context stored per response.
- **Employers Version (employers.py):** This script is meant for employers or hiring managers to get resume summaries for a given PDF file input, but it does not directly chat with the AI. Recently, I updated it so it doesn’t store previous message context, as its main function is to process one resume at a time. Storage in this context would be unnecessary as it is not required for the next responses, if deleted soon after.

Challenges

Some challenges were limiting the types of follow-up questions the users could freely ask the AI within the applicant code version. While the users may follow up with resume-related questions, they can also ask anything. I have attempted to limit topics but failed at doing so. It’s important to remember that each model does come with costs to call, which differ per model, so this may be something to refer to within the API documentation. Nonetheless, these are some challenges and considerations for the users I have so far for the applicant component of the project.

I highlighted above that I removed history within the employer script, as it’s unnecessary to be coded there. This means there isn’t contextual information stored, and it’s instantly deleted, if ever saved locally. The AI or API may have different data policies, which can be referred to in my README.md file. This may be a challenge for those who would like feedback, but I kept it out to experiment with two different functions. In the end, both scripts had remaining challenges with the input file types, though I have revised them.

Before my revision, the files searched for one-page PDF files. Since most resumes are one page, I filtered on this, checked if the files existed, and verified if they ended in “.PDF”. I’ve also revised the code, adding a filter to exclude blank files and a try-except block to work against empty and corrupted files. However, this may not be enough, knowing that almost any PDF file could be sent in, especially since the code does not check if it’s one page, but extracts the first page from a given file. This helps with processing, but adds some issues if the resume is not on the first page. This may be something the user may need to edit, but it might make it more difficult to use.

It might be additionally challenging considering off-topic files and the types of models used. For the applicant version, the code prints several job suggestions generated by the AI based on the file input. Since the employer code summarizes input files, the content may be processed as long as it’s readable. It’s unknown how the AI would react to these situations, as the info leading up to it has unique factors, especially with varying content. A theoretical solution I thought of was to add common resume terms, like header titles, to check the file as it’s being extracted. I considered this solution because of the computational complexities or costs associated with off-topic file uploads.

File Handling

While the previous restaurant recommendation project fell short, I stepped away from the AI chat system I had already built and focused on shifting towards the resume-based project. During the initial process, I concentrated on testing file uploads and input handling for the get_resume function rather than implementing responses. I attempted to upload various DOCX and TXT versions of the files I created to test how to read and extract different types of file formats within Python, and printed the results to cross-reference them. Although I had issues reading in DOCX files, I loaded TXT files before making changes. Later, I expanded the code to read and extract PDF files through the PyPDF Python library (Fenniak et al., 2022), ultimately filtering it down to PDFs only for commonality.

In the end, I uploaded several sample resumes to test if the code works when a one-page PDF file is uploaded and processed through either script, offering context to the AI, and receiving feedback transformed from the input files. A consideration is that I did not test against non-resume files. While my general PDF files in the previous steps worked to test file handling for different file formats, the goal was not to process them further. Thankfully, the resulting outputs confirmed some accuracy within the code for multiple samples, fulfilling the purpose of the task. However, these are not definitive results, as many factors affect the outcome of the results. There are many areas of debate around AI, such as inaccuracy, ethics, the job market, etc. There is much more to learn about these topics and the future. I hope my project can offer interesting points through my experience. As of this project date, I am satisfied with the functionality and how it can, at the minimum, retrieve feedback.

Code Functions:

1. **Get_resume:** Same for both scripts. Reads in a file via an input statement, checks through a try-except block that: the file exists, is a PDF, has readable content, and isn't corrupted. If any rules are broken or the user types in "exit", the code quits.
2. **ask_question:** Slightly different for each script. This function is a pipeline function used to connect to the AI. It houses an if statement for the user to quit the code. However, in the model code, the main differentiating factor in each code is that the employers.py does not contain the history variable, which is a list of dictionary values for previous questions and answers. The applicants.py does have this within their model function to add previous context.
3. **main:** Heavily different for each script. Holds both the get_resume and ask_question functions within a while loop. The differentiating factor here is that the applicants.py stores previous questions and answers within a list of dictionaries. The initial job suggestion question and get_resume are set up before the while loop so that it is not mixed up with the other questions. Then the feedback questions are run via the ask_question function within the while loop, appending it back to the history variable. The employers.py version does not have any code outside of the while loop, so it holds the get_resume function to input a file, then ask_question with a pre-set resume summarizing question, and loops for the next file, all until the user exits.

Overall Strengths:

1. Able to get information from AI responses and store them for context.
2. Both types of users can get immediate feedback on the entered information.
3. Applicants have the option to ask for contextual feedback about their resumes.

4. Detailed step-by-step instructions for ease of running the application.

Overall Weaknesses:

1. Unable to enter multiple file types, I picked PDF because most people use it. Initially, it was possible to able to read TXT files, but I removed the option since the code did not work for other types of files. This removes user preference, but saves code complexity. **May 2025 Update:** A thing I could have done was to allow TXT or use additional steps to check file types.
2. Employers cannot currently ask additional questions other than entering a new resume for a summary. I originally kept this for usability purposes to speed up summary times. I also wanted to experiment with different functions of the code.
3. Changed the original idea but kept functionality.

Repository Files:

1. **requirements.txt** - a list of libraries required to run
2. **README.md** - detailed information about the project, the interface, how it's run, and anything else users should know.
3. **yourkey.py** - a file where you need to enter **your OpenAI API key**.
4. **applicants.py** - the script that applicants can run to get resume feedback.
5. **employers.py** - the script that employers can run to get resume summaries.
6. **sampleoutput.pdf** - a PDF file displaying the transformed AI-generated responses and highlighting what both applicants and employers would see. Outputs were retrieved from one of the five sample resumes referenced below.

Sample Resumes (5)

I previously used 5 sample resumes by Stephen Greet from different articles on the website “BeamJobs” for code (Greet 2023a, 2023b, 2023c, 2023d, 2023e). They are not included in the repository. These samples are for the roles of ‘data scientist’, ‘business’, ‘attorney’, ‘data analyst’, and ‘dentist.’ Feel free to use your own files to test the code locally, but do be aware that AI may have sensitive data or inaccuracy risks. Use at your own risk. More information is available in the README.md.

a. sample1.pdf: 9 Data Scientist Resume Examples for 2023
b. sample2.pdf: 5 Business Resume Examples That Got the Job in 2023
c. sample3.pdf: 7 Attorney Resume Examples That Got the Job in 2023
d. sample4.pdf: 11 Data Analyst Resume Examples for 2023
e. sample5.pdf 5 Dentist Resume Examples Guaranteed to Work in 2023

Minimum Requirements:

Users can read the README.md file for detailed instructions on the dependencies and pip installs required. They can also view the “sampleoutput.pdf” to see how the interface should look in their command line.

Conclusion

While there is much more to learn, I believe this project can help utilize our knowledge on AI and how it can increase productivity for business solutions. The applicants' understanding of their skills and being able to ask questions helps them understand where they are and what they can do to improve themselves. Although most may already have prior knowledge of their skills and roles, the AI's input adds a potential for other jobs they are qualified for, with suggestions they may be unaware of. The hiring process can be time-consuming and draining as they have to look through many resumes, so a summarizer may be able to assist in those times. While there may be many other things to indulge in, like ethics, information, and future improvements, this project aimed to experiment with how helpful AI can be in solving real-world business problems. From my experience with this project, I hope it can give some insight to others or showcase my coding process.

References

Fenniak, M., Stamy, M., pubpub-zz, Thoma, M., Peveler, M., exiledkingcc, & PyPDF2 Contributors. (2022). *The PyPDF2 library*. https://pypi.org/project/PyPDF2/

Greet, S. (2023a, April 10). *9 Data Scientist Resume Examples for 2023*. BeamJobs. Retrieved April 18, 2023. https://www.beamjobs.com/resumes/data-science-resume-example-guide

Greet, S. (2023b, March 12). *5 Business Resume Examples That Got the Job in 2023*. BeamJobs. Retrieved April 18, 2023. https://www.beamjobs.com/resumes/business-resume-examples

Greet, S. (2023c, April 4). *7 Attorney Resume Examples That Got the Job in 2023*. BeamJobs. Retrieved April 18, 2023, from https://www.beamjobs.com/resumes/attorney-resume-examples

Greet, S. (2023d, April 7). *11 Data Analyst Resume Examples for 2023*. BeamJobs. Retrieved April 18, 2023, from https://www.beamjobs.com/resumes/data-analyst-resume-examples#writing-your-data-analyst-resume

Greet, S. (2023e, March 12). *5 Dentist Resume Examples Guaranteed to Work in 2023*. BeamJobs. Retrieved April 18, 2023, from https://www.beamjobs.com/resumes/dentist-resume-examples

OpenAI. (n.d.). OpenAI API. Retrieved March 2023, from https://openai.com/api/