Esther Kim

HCI 584 – Review of version 1

My project aims to solve a common challenge in conducting systematic reviews: the large amount of time and manual labor required to screen article titles and abstracts. Researchers often spend hours going through hundreds of articles just to identify which ones are relevant to their study. To address this, I’m building a Python-based web app that automates the early stages of article screening. The app parses article information exported from databases and uses ChatGPT to rate each article’s relevance based on a predefined research theme. Specifically, this theme (for the purpose of demonstration) focuses on intervention studies that measure physical activity and use mobile apps or wearable technology to either measure activity or deliver interventions aimed at increasing it. Once articles are rated, the results are saved into a structured CSV file for easier review and analysis, and in later stages, the app will allow visualization of scores and filtering options.

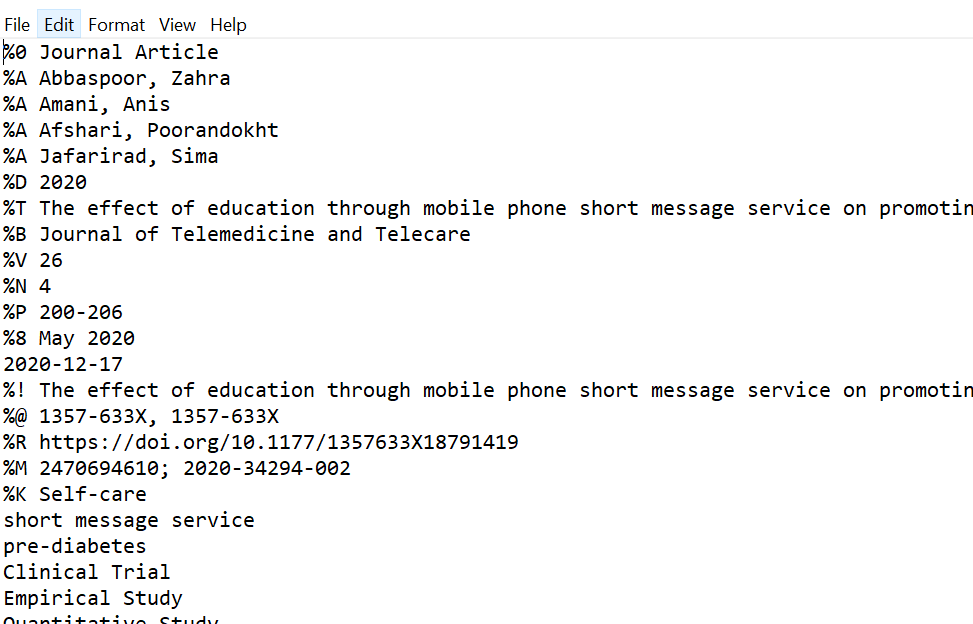
So far, I have successfully implemented several core parts of the project. The current version of the app can upload a .txt file exported from a bibliography platforms like Clarivate Endnote and parse it to extract key fields such as author names, article title, abstract, publication year, and DOI. I used regular expressions to target specific field codes in the exported file (e.g., %A for author, %T for title, %X for abstract), which solved the parsing issues I faced when initially trying to treat the file as a standard CSV. Once the data is structured, the app connects to the ChatGPT API using a secure keys.py file stored at the project root. It then loops through each article’s abstract, applies a clearly defined scoring rubric, and generates a relevance score between 1 and 10. A new column with these scores is added to the dataset and exported as a parsed\_articles\_scored.csv file. I tested this in PowerShell, and after solving some key import issues, the script now runs successfully and outputs the expected file with scores.

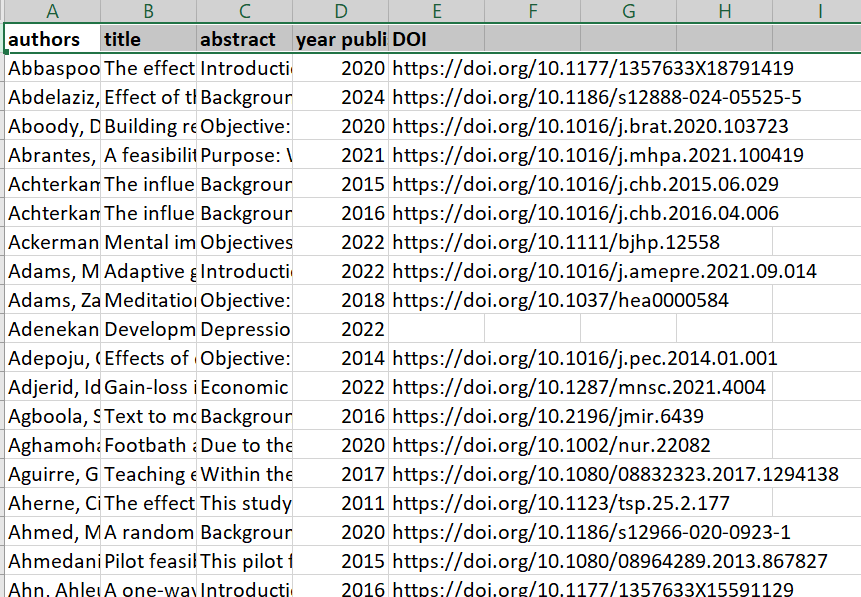
Throughout the implementation process, I encountered a few technical issues. The first major issue involved the API key import. I initially received a ModuleNotFoundError for keys, which was resolved by moving the keys.py file to the project’s root directory and appending the project root to sys.path in the script. I also ran into a NameError because the API key wasn’t wrapped in quotation marks, which Python interpreted as a variable rather than a string. Another issue was a syntax error related to the placement of from \_\_future\_\_ import annotations. Once I moved that line to the very top of the file, the script executed correctly. Parsing the .txt file was also trickier than expected; treating it like a CSV didn’t work because the number of fields varied. Using regex to extract fields based on ProQuest’s structured tags solved this problem cleanly. Finally, I had to adjust the column names to ensure consistency between the parsed data and the scoring script.

There are still a few unresolved issues and planned improvements. Currently, the app runs entirely in the command line, and the scoring process can be a bit slow since each abstract is scored sequentially through API calls. Some abstracts may also be missing or formatted in ways that cause scoring to fail. My plan is to build a simple Tkinter-based GUI that allows users to upload files and trigger the scoring process more intuitively. I will also add error handling and progress messages so users can monitor the status of their uploads and scoring. In addition, I plan to implement filtering and visualization features using matplotlib so users can see the distribution of scores and select the most relevant articles quickly.

Looking ahead, my milestones for the next few weeks are clearly defined. First, I will set up the GUI with basic file upload and a button to trigger scoring. Second, I’ll integrate the scoring process into the GUI so users can run the full pipeline without using the terminal. Third, I will add visualization tools such as bar charts or histograms to display the distribution of relevance scores. Fourth, I will build export and filtering features to allow users to easily download lists of top-scoring articles. Finally, I will work on polishing the interface, improving error handling, and ensuring that the app can handle different file formats and edge cases gracefully.

Reflecting on my progress so far, I’m satisfied with how much I’ve accomplished. Although the setup and import errors took longer to solve than expected, the core functionality of parsing and scoring works smoothly now. I feel confident that I can complete the project within the remaining time. On the other hand, handling file parsing required more careful thought than I initially assumed. One of my biggest light bulb moments was realizing how adding sys.path.append() fixed my entire import problem. Another was learning that regex provides a much more flexible solution for parsing structured text files than trying to force-fit them into a CSV format. Overall, I feel I have a solid foundation to build on and a clear plan for turning this prototype into a user-friendly app over the coming weeks. At the same time, I’m unexperienced so I would be expecting to build the app and learn as I go.

TXT file that is exported from the Clarivate Endnote website:  


CSV file that app created to parse out the articles: 

CSV file that the app created with the relevancy score:

