**EXERCISE 1 APPROACH**

I leveraged GPT-4o-mini's classification capabilities by carefully structuring a prompt that strictly returns "Yes" or "No." Then I iterated over each row in the dataset, sending relevant article details to the model to determine if they focus on gut health products. In each pass, if the model responded "Yes," I captured that row to build a filtered DataFrame of relevant content. This approach ensures a standardized classification pipeline using a large language model with minimal manual oversight. Finally, I saved the results to a new Excel file for further analysis and future reference.

HOW TO RUN
On the terminal, write the prompt:
export
OPENAI_API_KEY="sk-proj-PyZUMXCF3hHmPZw7K0541uYBMY_3u_WWcnYBs_8xJjRw
Vh05n7A8PYkNHAM-KIik40h1KvZeUiT3BlbkFJ1h1MZtAeyI4VB9FsL9mbA95iaJsWRO6448
hJfliI66HWLLgGRs5kYv-lR66hjNbi3IU-T6ZTAA"

Then you can run the file with python task1.py

**EXERCISE 2 APPROACH**

First, I needed a way to retrieve web pages reliably, so I used requests for fetching HTML and implemented error handling to avoid failures from broken links or timeouts. Extracting relevant subpage links was crucial, so BeautifulSoup was used to parse and filter only valid absolute URLs.
Once links were gathered, the challenge was extracting structured data from unstructured HTML. Instead of manually defining parsing rules, OpenAI's API was leveraged to interpret and return structured JSON data. Since the API has context limits, truncation ensured only the most relevant content was sent.
I then defined a dictionary to store extracted data, using company names as unique keys. If a company already exists, missing fields are supplemented instead of overwriting existing information, maintaining completeness. Finally, the extracted data is saved into a CSV file.

To run, simply change the url_main variable to the desired one, and run in the terminal with python task2.py