Task 1

key considerations

1. Dynamic Content Loading (click_load_more): Use 12 in the script to avoid taking too much time.

- It repeatedly searches for and clicks buttons or links containing "load" (case-insensitive) to reveal more faculty profiles that might be hidden behind "load more" or pagination mechanisms.

2. Identifying Profile Containers (inspect_frequent_combos):

- It counts the frequency of each unique class string (combination of classes). It then takes the most frequent class combinations .
- For each of these frequent class combinations, it calculates a "hit ratio": the proportion of elements with that class combination that also contain keywords like "professor" or "lecturer" in their text content. (The container for the profile should include title information.)
- The class combination with the highest hit ratio is selected as the most likely container for individual faculty profiles.

3. LLM-Powered Rule Inference: extract the structure of the container.

- Once the main profile container class is identified, the script takes a sample HTML snippet from one of these containers.
- It constructs a prompt for the DeepSeek LLM, providing the sample HTML and asking the LLM to return a JSON object.
- This JSON object should map predefined field names (e.g., "name", "title", "email", "research interest") to:
- "selector": A concise CSS selector to find the element containing the field's value within the profile card.
- "tag_pattern": The opening and closing HTML tags (including class attributes) that usually enclose the field's value. (Though the script primarily uses the selector).

4. Data Extraction Loop:

- The script finds all elements matching the best profile container class identified earlier.
- It iterates through each of these "profile card" elements.
- For each card, and for each field defined in the LLM-generated rules (name, title, etc.): It attempts to find the specific data element using the CSS selector provided by the LLM for that field. It extracts the text content. If the text is empty, it tries a fallback by checking the data-value attribute of the parent element (adjustment for fitting more websites).

Task 2

key considerations

1. Product List Identification (find_products):

- It analyzes the page for frequently repeated CSS class combinations on major layout elements since the containers of the products are often duplicated.
- It filters out class combinations that appear at least min_products_for_list times (e.g., 10)

2. Then, for each container: Main Link Extraction from Containers

- Within each potential product container, it tries to find the primary clickable link leading to the product page.

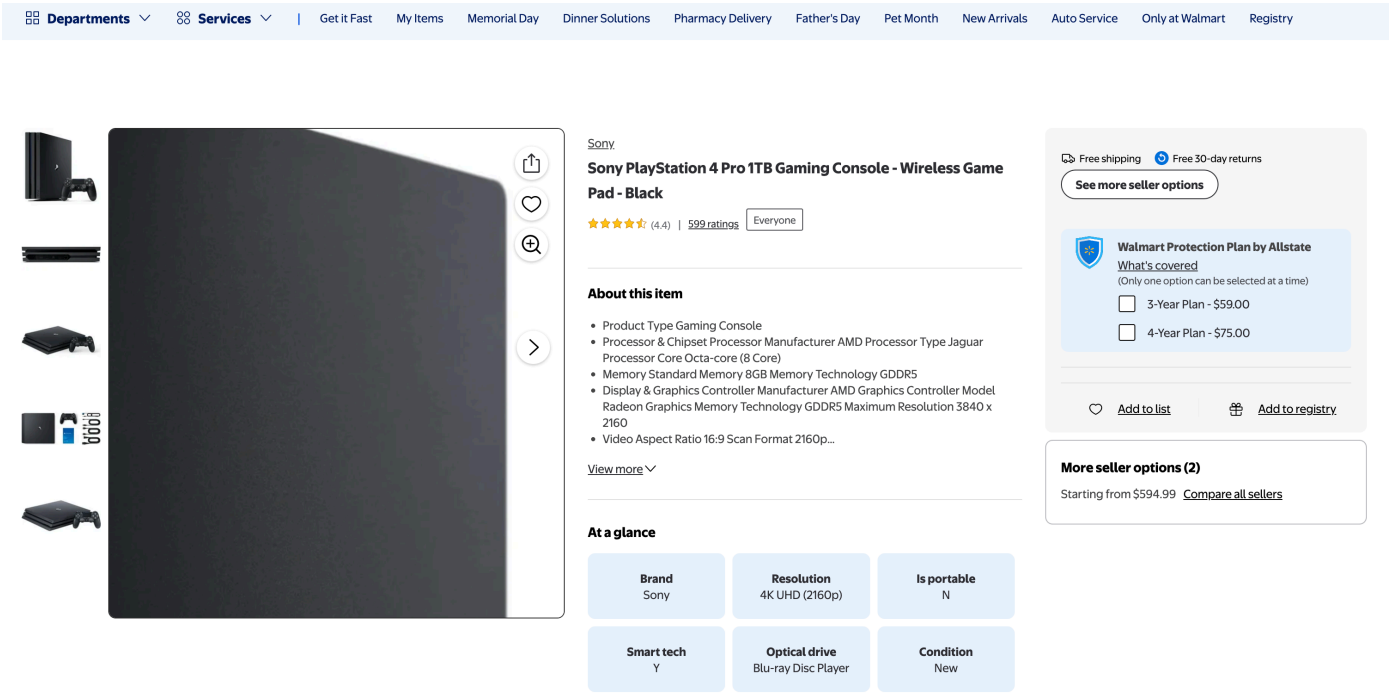
  If there is none, return. Select the second container.

- It handles lazy loading by scrolling if a link isn't immediately found after a successful previous extraction (very likely the right container). If multiple consecutive attempts fail to find a link in containers, it stops processing that class combination.

3. Within the main loop, process the corresponding container of the selector each time by its index.
4. Product Detail Extraction (extract_product_info):

- Attempts to extract the title and price using a predefined list of common CSS selectors and XPath. (Handle most cases) (fast and robust: no extra time needed for api calling)

corner cases: No price is displayed within the price area.



- If the price is not found, doesn't contain a '$' and digits, or is exactly '$0.00', it also uses extract_with_deepseek for the price.
- The extract_with_deepseek function sends the HTML content and a targeted prompt (for 'price' or 'title') to the DeepSeek API.

5. Pagination:

- After processing products on a page, it searches for a "next page" button using various common selectors.
- It attempts to click the button and verifies if the URL changes since the selector I defined might unintentionally capture a common “next” button. It also has a mechanism to try alternative next page buttons if the first one fails.

6. Error Handling

Things to optimize:

For different pages, the selector for the product container should remain consistent. Currently, it’s being re-selected each time a new page is loaded. I can simplify the logic in the future, which I believe will reduce the execution time.