Assignment 2: Sage Journal Article Scraping

Objective
Scrape the following details from the Journal of Marketing (SAGE) current issue page:

- Title
- Author names
- First published date
- DOI
- Abstract

Approach:

1. **Approach 1: Requests + BeautifulSoup**
    - Tried using Python requests to fetch static HTML.
    - Result: Did not work, since the page loads article content dynamically via JavaScript.

    Conclusion: Fast and lightweight, but only works for static pages without dynamic content.

2. **Approach 2: Selenium + BeautifulSoup**
    - Used Selenium to control Chrome browser and render JavaScript content.
    - Accepted cookies automatically.
    - Waited for full JavaScript execution to ensure all articles and details are loaded.
    - Parsed page source using BeautifulSoup.
    - Extracted title, authors, first published date, DOI, and abstract for each article.

    Result: Successfully scraped all articles and saved as CSV.

Reason for choosing Selenium

- The SAGE journal website uses JavaScript to dynamically load article content, which is not accessible in plain HTML fetched via requests.
- Selenium can simulate real user behavior including scrolling and clicking (e.g., accepting cookies).
- Ensures complete and accurate scraping of dynamically generated elements.
- Provides flexibility to handle pop-ups or future layout changes.
- Although slightly slower, Selenium guarantees that all necessary data is fully loaded and correct.

Tech Stack

- **Python 3.x**
  Used as the main programming language for scripting and data processing.

- **Selenium**
  Used for web automation to control Chrome browser, handle JavaScript-rendered content, accept cookies, and fully load dynamic web pages**.**

- **BeautifulSoup (bs4)**
  Used for parsing and navigating the final rendered HTML to extract article details (title, authors, abstract, publication date, DOI).

- **pandas**
  Used for organizing extracted data into a structured tabular format and saving it as a CSV file.

- **ChromeDriver**
  Required by Selenium to automate Google Chrome**.**

- **Requests** ***(explored but not used in final solution)***
  Initially tested as a lightweight alternative to directly fetch HTML content, but was not used in the final approach due to JavaScript rendering requirements**.**

Final Outcome

- **CSV file with complete article metadata created successfully.**
- **Selenium-based approach chosen and recommended for this assignment to handle dynamic web content reliably**.

Code & Output

- **Provided final working Python code.**
- **Output CSV**: **assignment2_articles_final.csv**