Home Project #8: Product Catalog Crawler

Target Websites (Select one or more e-commerce sites)

- Amazon product listings
- eBay search results
- Etsy product pages
- Best Buy electronics
- Local e-commerce sites

Required Tasks

1. **Set up Selenium WebDriver with rotating user agents and appropriate headers.**

   Configure your WebDriver to appear as different browsers by rotating user agent strings. This helps avoid detection as a bot. Include headers like Accept, Accept-Language, and Accept-Encoding to make requests look more natural. Set up ChromeOptions or FirefoxOptions to disable images and CSS loading for faster scraping.

2. **Create a function to navigate through product listing pages with pagination.**

   Build a function that can automatically move through multiple pages of product listings. Handle different pagination styles: numbered pages, "Next" buttons, infinite scroll, or "Load More" buttons. Your function should detect when you've reached the last page and stop automatically.

3. **Implement both BeautifulSoup and Selenium for different content types.**

   Use BeautifulSoup for static HTML content that loads immediately, and Selenium for dynamic content that requires JavaScript execution. Learn to identify which elements are static versus dynamic by checking if they appear in the initial page source versus being loaded later through AJAX calls.

4. **Extract at least 8 product attributes from each item:**

   - **Product name and description:** Get the main title and any available product details or specifications
   - **Current price and original price:** Handle different price formats, discount indicators, and currency symbols
   - **Product image URL:** Extract the main product image link, handling lazy-loaded images
   - **Customer rating:** Convert star ratings to numerical values (e.g., 4.5/5 stars)
   - **Number of reviews:** Extract review counts, handling formats like "1,234 reviews" or "(567)"
   - **Availability status:** Detect if item is in stock, out of stock, or limited availability
   - **Seller information:** Get seller name, whether it's sold by the site or third party
   - **Product category:** Extract category breadcrumbs or tags associated with the product

5. **Handle dynamic content loading with explicit waits.**

   Use WebDriverWait with expected_conditions to wait for specific elements to appear before trying to extract data. Common waits include: presence_of_element_located, element_to_be_clickable, and visibility_of_element_located. Set reasonable timeout values (10-30 seconds) and handle TimeoutExceptions gracefully.

6. **Implement comprehensive error handling for missing elements and network issues.**

   Wrap your extraction code in try-except blocks to handle missing product information (some products might not have ratings or descriptions). Handle common exceptions like NoSuchElementException, TimeoutException, and WebDriverException. When an element is missing, store "N/A" or None rather than crashing.

7. **Add retry logic with exponential backoff for failed requests.**

   When a page fails to load or an element can't be found, retry the operation with increasing delays (1 second, 2 seconds, 4 seconds, etc.). Limit the number of retries (usually 3-5 attempts) and log each retry attempt. This handles temporary network issues or server overload.

8. **Navigate through at least 5 pages of product listings and collect minimum 50 products.**

   Your scraper should automatically move through multiple pages and accumulate products from all pages. Keep track of total products collected and stop when you reach your target number. Handle cases where there are fewer than 5 pages available.

9. **Clean and normalize extracted data.**

   Process the raw extracted text to make it consistent: remove extra whitespace, standardize price formats (convert "$1,234.56" to 1234.56), convert ratings to decimal numbers, clean up product descriptions by removing HTML tags or excessive formatting.

10. **Store data in both CSV and JSON formats with proper encoding.**

    Save your data in two formats: CSV for easy viewing in spreadsheet applications, and JSON for programmatic access. Use UTF-8 encoding to handle international characters. Include column headers in CSV and proper structure in JSON with meaningful field names.

11. **Add configurable rate limiting delays between requests.**

    Implement delays between page loads and data extraction to avoid overwhelming the server. Use random delays between 2-5 seconds to make the scraping pattern less predictable. Make these delays configurable so they can be adjusted for different websites.

12. **Create a simple data analysis summary.**

    After collecting your data, generate basic statistics: average price per category, highest and lowest priced items, most common sellers, average ratings, and price distribution. Display this analysis in a readable format, either printed to console or saved to a text file.

13. **Implement logging to track scraping progress and errors.**

    Use Python's logging module to record important events: when scraping starts/ends, successful page loads, extraction errors, retry attempts, and final statistics. Set up different log levels (INFO for progress, ERROR for problems) and save logs to a file for later review.