Project 2: Web Scraping (200 Points)

Introduction

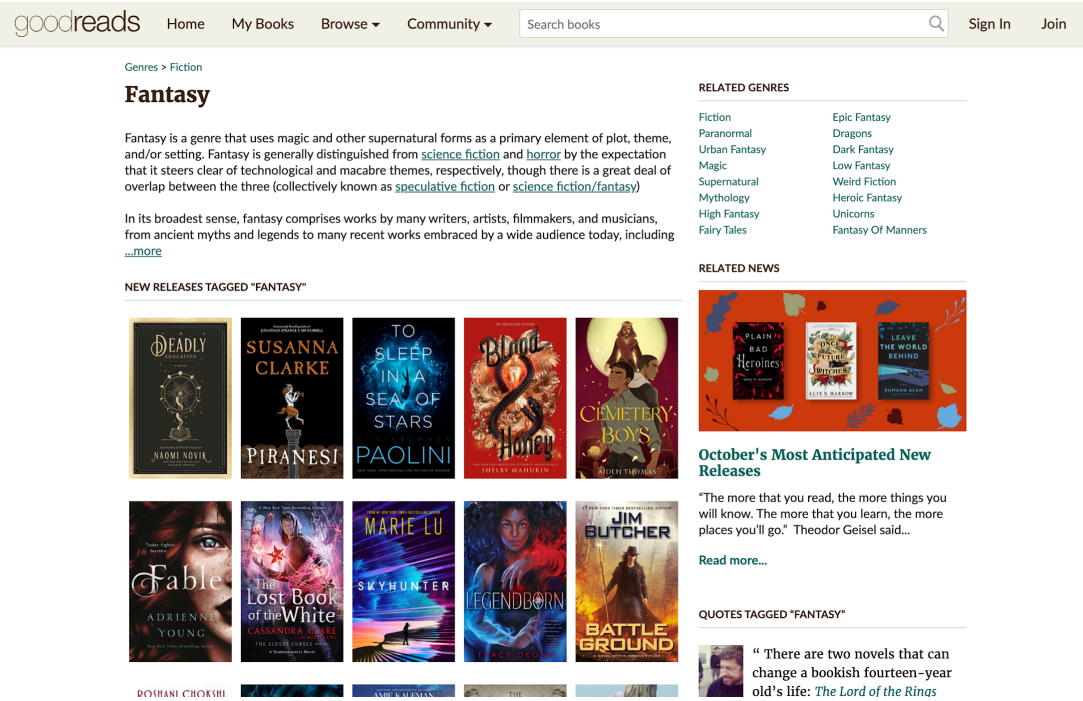
Often in the world of data science, there isn't a neat prepackaged dataset for the problem that we're interested in solving.

When this occurs, we're often forced to compile and process a dataset from scratch so that we can do data analysis and answer the questions that interest us.

One powerful way to do that is through web scraping. Web scraping is the process of taking messy data from the web, processing it, cleaning it, and turning it into something useful for analysis.

The ability to do web scraping well is a powerful tool since a large majority of data science work is often cleaning up messy data.

In this project…



You will be scraping data taken from Goodreads.com, cleaning it, and extracting information from it. You will need to use the BeautifulSoup library to parse through the HTML documents.

We have provided two static documents for you to use, but you will need to scrape some live content as well.

After you've implemented all of the required functions, you will need to write test cases for each one. We have provided guidance for what to test for in the comments, but it will be up to you to implement the logic in the code. In order to write good test cases, you will need to open the websites, explore, and get a sense of what your data should actually look like.

If you choose to do the extra credit part, you will be exposed to using multiple data cleaning methods at once. For that, you need to combine BeautifulSoup with Regex and write the output to a .csv file.

The code

You will need to write several functions and their test cases. Start from the starter code provided, which looks like the following:

```
def get_titles_from_search_results(filename):
    """
    Write a function that creates a BeautifulSoup object on "search_results.htm". Parse
    through the object and return a list of tuples containing book titles and authors
    (as printed on the Goodreads website) in the format given below. Make sure to
    strip() any newlines from the book titles and author names.

    [('Book title 1', 'Author 1'), ('Book title 2', 'Author 2')...]
    """



def get_search_links():
    """
    Write a function that creates a BeautifulSoup object after retrieving content from
    "https://www.goodreads.com/search?q=fantasy&qid=NwUsLiA2Nc". Parse through the
    object and return a list of URLs for each of the first ten books in the search
    using the following format:


    ['https://www.goodreads.com/book/show/84136.Fantasy_Lover?from_search=true&fro
    m_srp=true&qid=NwUsLiA2Nc&rank=1', ...]

    Notice that you should ONLY add URLs that start with
    "https://www.goodreads.com/book/show/" to your list, and be sure to append the full
    path to the URL so that the url is in the format
    "https://www.goodreads.com/book/show/kdkd".
```

```
    """



def get_book_summary(book_url):
    """
    Write a function that creates a BeautifulSoup object that extracts book
    information from a book's webpage, given the URL of the book. Parse through
    the BeautifulSoup object, and capture the book title, book author, and number of
    pages. This function should return a tuple in the following format:

    ('Some book title', 'the book's author', number of pages)

    HINT: Using BeautifulSoup's find() method may help you here.
    You can easily capture CSS selectors with your browser's inspector window.
    Make sure to strip() any newlines from the book title and number of pages.
    """

def summarize_best_books(filepath):
    """
    Write a function to get a list of categories, book title and URLs from the "BEST
    BOOKS OF 2020" page in "best_books_2020.htm". This function should create a
    BeautifulSoup object from a filepath and return a list of (category, book title,
    URL) tuples.

    For example, if the best book in category "Fiction" is "The Testaments (The
    Handmaid's Tale, #2)", with URL
    https://www.goodreads.com/choiceawards/best-fiction-books-2020, then you should
    append("Fiction", "The Testaments (The Handmaid's Tale, #2)",
    "https://www.goodreads.com/choiceawards/best-fiction-books-2020")
    to your list of tuples.
    """

def write_csv(data, filename):
    """
    Write a function that takes in a list of tuples (called data, i.e. the
    one that is returned by get_titles_from_search_results()), writes the data to a
    csv file, and saves it to the passed filename.

    The first row of the csv should contain "Book title", and "Author Name",
    respectively as column headers. For each tuple in data, write a new
    row to the csv, placing each element of the tuple in the correct column.
```

```
    When you are done your CSV file should look like this:

    Book title,Author Name
    Book1,Author1
    Book2,Author2
    Book3,Author3


    This function should not return anything.
    """

class TestCases(unittest.TestCase):

For each function you wrote above you should write a non-trivial test case
to make sure that your function works properly.

We have described the test cases that you should write in the comment for the test
functions. It is up to you to correctly implement this logic using the assert
statements in the unittest library.
```

Grading

Function	*points*
`def get_titles_from_search_results(filepath)`	30
`def get_search_links()`	30
`def get_book_summary(book_url)`	30
`def summarize_best_books(filepath)`	40
`def write_csv(data, filename)`	20
`TestCases (10 points for each)`	50
Total	**200**
`def extra_credit():`	15 pts extra credit

We will be checking to make sure that you've implemented each function correctly. You will need to make sure that you are returning data in the specified format to get full credit. You will also need to make sure that you are calling the other functions when directed to do so.

We have provided descriptions of what you should be testing for in order to make sure that you are on track. You will need to implement the actual code for these tests.

Tips

Work on one function at a time. Choose the one that you think is the easiest, and work on it until you can get all the tests related to that function to pass. This is a great strategy since ***the solution to some functions can be used to quickly complete other functions.***

Extra Credit: 15 points

Sometimes when processing text data, it is useful to extract a list of people, places, and things that a document is about. This allows us to quickly tag documents by their content and can allow for faster search and retrieval, as well as providing a brief summary of the document's contents. In the field of Natural Language Processing, this task is called Named Entity Recognition (NER).

These days, most NER is done using Artificial Intelligence. But, we can create a simple entity recognizer using Regex! Since English conveniently capitalizes proper nouns, we can use this to construct a regex pattern to easily grab many named entities from text.

For the purposes of this assignment, we will define a named entity as follows:

- Named entities contain 2 or more capitalized words, with no lowercase words in-between them
- The words must be separated by spaces
- The first word must contain at least 3 letters

Write a new function *extra_credit()* that takes a single *filepath* parameter. It should create a BeautifulSoup object from the filepath, given that *filepath* corresponds to the webpage for a book on Goodreads.com. Extract the description** of the book from the BeautifulSoup object and find all the named entities (using the criteria given above) within the book description. This function should return a list of all named entities present in the book description for the given *filepath*. Your list should be in the following format:

[‘Named Entity_1’, ‘Named Entity_2, …..]

You have to get all of the named entities (and not any extras) in order to receive the points. If you implement this correctly, you should find 9 named entities in the file “extra_credit.htm”

**For example, in the screenshot of a book shown below, the description refers to all the text that is present in the highlighted region.