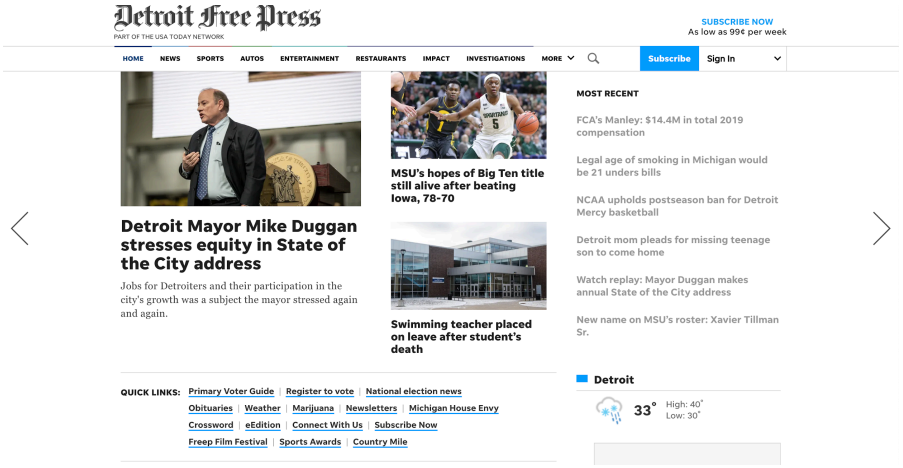
Project 2: Web Scraping: 200 Points

Due March 19

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 Detroit Free Press, 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.

In order to expose you to using multiple data cleaning methods at once, you will need to combine BeautifulSoup with Regex and write output to a .csv file to complete this assignment.

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 code. In order to write good test cases, you will need to open the websites, explore, and get a sense for what your data should actually look like.

The code

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

```
def get_headlines_from_search_results(filename):
    """
    Write a function that creates a BeautifulSoup object on the passed filename. Parse
    through the object and return a list of headlines for each search result in the
    following format:

    ['Headline 1', 'Headline 2'...]
    """

def get_most_recent():
    """
    Write a function that creates a BeautifulSoup object after retrieving content from
    "https://freep.com". Parse through the object and return a list of URLs for each
    of the articles in the "MOST RECENT" section using the following format:

    ['https://www.freep.com/story/news/local/michigan/2020/02/26/michigan-foia-slow-
    costly-whitmer-transparency/4786653002/', ...]
    """

def get_article_summary(article_url)
    """
    Write a function that creates a BeautifulSoup object that extracts information
    from an article, given the relative URL of the article. parse through
    the BeautifulSoup object, and capture the article title, author, and date. Make
    sure to strip() any whitespace from the date. If the timestamp contains
    "Published" and "Updated" information, you should only capture the "Published"
    part.
```

```
    This function should return a list of tuples in the following format:

    [('Some headline', 'Some Author', 'Published 12:00 a.m. ET Jan 01, 2020')...]

    HINT: Using BeautifulSoup's find() method may help you here.
    You can easily capture CSS selectors with your browser's inspector window.
    """

def summarize_corrections(filepath):
    """
    Write a function to get the section tags and dates of the Corrections &
    Clarifications article in "corrections.htm". You need to use regex to accomplish
    this. This function should create a BeautifulSoup object from a filepath and
    return a list of (tag, date) tuples.

    For example, if the article contains: "Sports: Jan 1, a basketball story was
    Published", you should append ("Sports", "Jan 1") 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 summarize_corrections()), writes the data to a csv file, and saves it
    to the passed filename.

    The first row of the csv should contain "Tag" and
    "Date" respectively. 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:

    Tag,Date
    Some Tag, Jan. 1
    Another Tag, Feb. 2
    Yet another Tag, Mar. 3


    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_headlines_from_search_results(filepath):`	30
`def get_most_recent():`	30
`def get_article_summary(article_url):`	30
`def summarize_corrections(filepath):`	30
`def write_csv(data, filename):`	30
`TestCases (10 points for each)`	50
Total	**200**
`def summarize_corrections_expanded():`	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 format specified in the docstrings 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: Named Entity Recognition - 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 content. 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 can be separated by spaces or hyphens (-)
- The first word contains at least 3 letters

Write a new function *summarize_corrections_expanded()* that takes a single *filepath* parameter. It should operate the same as your *summarize_corrections()* function, but it should add 2 additional items to each tuple: a list of the named entities, and the URL associated with each correction. Your final tuples should follow this format:

```
('Some headline', 'Some Author', 'Published 12:00 a.m. ET Jan 01, 2020',
'https://www.freep.com/story/…', ['Ann Arbor', 'Jane Doe', 'Foo Foo-Bar'...])
```

You will receive 10 points of extra credit for correctly identifying and outputting the named entities, and 5 points of extra credit for correctly identifying and outputting the URLs. We won't give partial credit, so 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 52 named entities.