# Homework 8: Structured Data

Due April 5, 11:59 pm

Worth 15 points

Instructions on writing and submitting your homework can be found on the course webpage at `http://pages.stat.wisc.edu/~kdlevin/teaching/Spring2024/STAT606/hw_instructions.html`. *Failure to follow these instructions will result in lost points.* Please direct any questions the instructor.

## 1 Warmup: Parsing HTML (4 points)

Let's get started using `BeautifulSoup` with a couple of simple exercises. Both of the following subproblems ask you to retrieve the HTML from a URL given by an argument `s` and determine some simple information about that HTML. In both functions, you should raise an appropriate error in the event that `s` is not a string, and you should raise an `HTTPError` with an appropriate error message in the event that the success code that results from trying to access the URL is not code 200. You may rely on `requests` and/or `urllib` to raise an error for you in the event that `s` is a string but does not encode a valid URL.

1. Write a function `get_page_title` that takes a string `s` as its only argument and returns a string. Your function should try to treat `s` as a URL, and return the string stored in the title tag of the HTML page stored at that URL. In the unlikely event that the URL has more than one title tag, your function should return the text stored in the first one. If no such title exists, your function should return `None`. A good way to test this function is to simply check that your function returns the string matching the title in your browser– the page title will always be displayed in the tab in which you have the page open.

2. Write a function `count_links` that takes a string `s` encoding a URL as its only argument and returns an integer corresponding to the number of hyperlinks on the webpage stored at that URL. The homework instructions page linked to above is an easy page to test your code on– there are only three links.

## 2 Retrieving Data from the Web (7 points)

In this problem, we'll scrape data from Wikipedia using `BeautifulSoup`. Documentation for BeauitfulSoup can be found at `https://www.crummy.com/software/BeautifulSoup/bs4/doc/`. As mentioned in lecture, there is another package, called `requests`, which is becoming quite popular, which you are welcome to use for this problem instead, if

you wish. Documentation for the `requests` package can be found at `http://docs.python-requests.org/en/master/`.

Suppose you are trying to choose a city to vacation in. A major factor in your decision is weather. Conveniently, lots of weather information is present in the Wikipedia articles for most world cities. Your job in this problem is to use `BeautifulSoup` to retrieve weather information from Wikipedia articles. We should note that in practice, such information is more easily obtained from, for example, the National Oceanic and Atmospheric Administration (NOAA) in the case of American cities, and from analogous organizations in other countries.

1. Look at a few Wikipedia pages corresponding to cities. For example:

   - `https://en.wikipedia.org/wiki/Madison,_Wisconsin`
   - `https://en.wikipedia.org/wiki/Buenos_Aires`
   - `https://en.wikipedia.org/wiki/Harbin`

   Note that most city pages include a table titled something like "Climate data for [Cityname] (normals YYYY-YYYY, extremes YYYY-YYYY)" Find a Wikipedia page for a city that includes such a table (such as one of the three above). In your jupyter notebook, open the URL and read the HTML using either `urllib` or `requests`, and parse it with `BeautifulSoup` using the standard parser, `html.parser`. Have a look at the parsed HTML and find the climate data table, which will have the tag `table` and will contain a child tag `th` containing a string similar to

   ```
   Climate data for [Cityname] (normals YYYY-YYYY, extremes YYYY-YYYY).
   ```

   Find the node in the `BeautifulSoup` object corresponding to this table. What is the structure of this node of the tree (e.g., how many children does the table have, what are their tags, etc.)? You may want to learn a bit about the structure of HTML tables by looking at the resources available on these websites:

   - `https://developer.mozilla.org/en-US/docs/Web/HTML/Element/table`
   - `https://www.w3schools.com/html/html_tables.asp`
   - `https://www.w3.org/TR/html401/struct/tables.html`

2. Write a function `retrieve_climate_table` that takes as its only argument a string representing a URL, and returns the `BeautifulSoup` tag object corresponding to the climate data table (if it exists in the page) and returns `None` if no such table exists on the page. You should check that the URL is retrieved successfully, and raise an error if `urllib2` fails to successfully read the website. You may notice that some city pages include more than one climate data table or several nested tables (see, for example, `https://en.wikipedia.org/wiki/Los_Angeles`). In this case, your function may arbitrarily choose one of the tables to return as a `BeautifulSoup` object. **Note:** a good way to check for edge cases is to test your script on the Wikipedia pages for a few of your favorite cities. The pages for Los Angeles and Boston will give good examples of edge cases that you should be able to handle. Of course, there is no expectation that your code handle every possible edge case. That would be impossible. Your code should, however, be reasonably robust to small differences among some of the city pages you find when you start testing your code. **Hint:** make use of the `get_text()` method supported by `BeautifulSoup` objects.

3. As you look at some of the climate data tables, you may notice that different cities' tables contain different information. For example, not all cities include snowfall data. Write a function `list_climate_table_row_names` that takes as its only argument a Wikipedia URL and returns a list of the row names of the climate data table, or returns `None` if no such table exists. The list returned by your function should, ideally, consist solely of Python strings (either Unicode or ASCII), and should not include any `BeautifulSoup` objects or HTML, and the strings should not have any trailing whitespace (**Hint:** see the `BeautifulSoup` method `get_text()`). The list returned by your script should *not* include an entry corresponding to the `Climate data for...` row in the table. **Second hint:** you are looking for HTML table header (`th`) objects. The HTML attribute `scope` is your friend here, because in the context of an HTML table it tells you when a `th` tag is the header of a row or a column.

4. The next natural step would be to write a function that takes a URL and a row name and retrieves the data from that row of the climate data table (if the table exists and has that row name). Doing this would require some complicated string wrangling to get right, so I'll spare you the trouble. Instead, please **briefly** describe either in pseudo code or in plain English how you would accomplish this, using the two functions you wrote above and the tools available to you in the `BeautifulSoup` package. **Note:** just to be clear, you **do not** have to write any code for this last step. Of course, if you want a challenge, you are welcome to try writing this code, but it is not required for this assignment.

## 3 JSON (4 points)

In this problem, you'll get a bit of practice working with JSON objects.

1. Download `yelp_simplified.json` from `http://pages.stat.wisc.edu/~kdlevin/teaching/Spring2024/STAT606/yelp_simplified.json`. This file contains a string representation of a JSON object generated by the Yelp API when I asked to retrieve all restaurants matching the search term `'coffee'` within 200 meters of the UW-Madison statistics department. I've removed some of the attributes for the sake of simplicity. We'll discuss how to interact with APIs like the Yelp API and others later in the course. Load the JSON string into a Python `json` object called `yelp_json`. Please include `yelp_simplified.json` in your submission. **Hint:** you can read a JSON object directly from the file by creating a file handle `f` and writing `json.load(f)`. Don't forget to close the file after you're done reading from it!

2. The `'businesses'` attribute of the JSON attribute has as its value an array whose elements are themselves JSON objects, each of which represents one of the three establishments within 200 meters of the Medical Sciences Center matching my search for `'coffee'`. Extract this array (you can simply treat it as a Python list!) and save it in a variable called `search_results`.

3. Pick out one of the JSON elements from the list `search_results` and examine its attributes. Extract the attributes and save them in a Python list `resto_attrs_sorted`, with the attributes sorted in non-decreasing order. **Hint:** the attributes of a JSON object in the Python `json` module really are just the keys of a dictionary, so all you need to do is extract the keys of a dictionary, save them in a list, and sort that list.

4. A few of us in the statistics department have decided to follow our passion for baking and open the George E. P. Box Bakery in the basement of MSC. Create a Python dictionary `gepbox_json` representing our bakery's JSON object. It should have the same structure as the other restaurants in the JSON search results. In particular:

   - The name of our bakery on Yelp will be `'G. E. P. Box Bakery'`.
   - Since our bakery is brand new, let us set its `'review_count'` to 0.
   - The `'categories'` attribute should be an array containing a single JSON object `{'alias': 'cafes', 'title': 'Cafes'}`
   - Set the `'rating'` attribute to have value 5.0.
   - The coordinates of the MSC are latitude 43.074281 and longitude -89.407391.
   - We aim to provide affordable croissants to statisticians of all income levels. The `'price'` attribute should be a single dollar sign.
   - The phone number will be the same as the department office: `+16082622598`.
   - The other restaurants in the search results have a `'distance'` attribute, which is the distance from MSC. Since our bakery is in the basement of MSC, you can set the `'distance'` attribute to have value 0.0.

5. Use the Python `json` module to create a string representation of our `gepbox_json` object, and save it in a file `gepbox.json`. Please include this file in your submission. **Hint:** to make sure that you have created your JSON file correctly, try reading it back into Python using the same pattern you used above to read in `yelp_simplified.json`.