BCCP web scraping course

Day 1

Table of Contents

Day 1

very short intro to Python

Very short intro to Python

-

Day 1

Intro to Webscraping

Introduction to Webscraping

- Basic idea: Turn information on website to structured data
- Typical workflow:
  1. Look at website to decide best approach
     - Is an Application Programming Interface (API) available?
     - Do the HTML elements have fixed names?
     - Does the page load statically or dynamically?
  2. Download information from URL
  3. Turn information into structured data and save

Some concepts

- APIs
- HTML parsing vs text matching
- Static vs dynamic websites

APIs

- If available, a convenient way to get pre-structured data (usually JSON or XML).
- Example: OpenStreetMap (OSM) (`https://www.openstreetmap.org`)
  - When searching manually, results can be shown as XML. Automating the search on OpenStreetMap and clicking on the relevant links would therefore be a way to save this data.
  - However, OSM offers several APIs that simplify this task. One API is the Nominatim API (`https://nominatim.openstreetmap.org`).

API example: Nominatim API for OSM

- See `https://nominatim.org/release-docs/develop/api/Search/` for documentation on search syntax
- Search for 'diw berlin' and return as JSON: `https://nominatim.openstreetmap.org/search?q=diw+berlin&format=json`
- The JSON format has a similar structure as dictionaries in Python and can easily be transformed to DataFrames.

HTML parsing

- Use structure of HTML code to find needed information.
- Works best if the code is well-structured and element names are fixed.

HTML parsing example: eBay search results

- Look at results for 'star wars blu ray' on eBay: `https://www.ebay.de/sch/i.html?_nkw=star+wars+blu+ray`
- Most browsers have a feature to look at source code (e.g. in Chrome, you can right click on any website element and click on 'Inspect').
- On eBay, the HTML tags containing certain content always have the same name, this simplifies HTML parsing.
- Foe example, the tag *<div id="ResultSetItems">* contains all results. Inside this tag, the individual listings are saved in tags called *<li class="sresult">*. In Chrome, you can also look for elements using the XPATH syntax (e.g. for the individual listings: *//li[contains(@class,'sresult')]*). More information on XPATH here: `https://www.w3schools.com/xml/xpath_syntax.asp`

Text pattern matching

- If the HTML code is not well-structured or names change, text pattern matching is an alternative.
- Idea: Take text from (parts of) a page and find needed information by matching a regular expression

Example of website without clear HTML tag names: Airbnb

- Search for homes in Berlin-Mitte: `https://www.airbnb.de/s/Berlin-Mitte--Berlin/homes?query=Berlin-Mitte%2C%20Berlin`
- Say you wanted to get the number of results for this search. The element does not have a clear name. Using HTML parsing is still possible but is prone to errors. Instead, one could match on a regular expression.

Static vs dynamic websites

- On static websites, the entire content is loaded immediately. E.g. eBay: `https://www.ebay.de/sch/i.html?_nkw=star+wars+blu+ray`
- On dynamic websites, content may not load instantaneously or only after user action, making them usually more complicated to scrape. E.g. Airbnb: `https://www.airbnb.de/s/Berlin-Mitte--Berlin/homes?query=Berlin-Mitte%2C%20Berlin` (Try disabling JavaScript in your browser and reloading the page).
- Getting the complete source code from a dynamic website can be done with browser automation. The idea is to open a website in an actual browser (and interacting with it if necessary) and save the source code of the content from there.

Important Python packages

- `requests`: To load URL and recover source code (for static web pages)
- `beautifulsoup4`: To turn HTML code to navigable Python object
- `selenium`: For browser automation
- `pandas`: To create DataFrames

Day 1

APIs

Application Programming Interface

-

Twitter API

- "Conduct historical research and search from Twitter's massive archive of publicly-available Tweets posted since March 2006?"
- "Listen in real-time for Tweets of interest?"

Day 2

Table of Contents

Day 2

HTML parsing

HTML parsing

- After obtaining the HTML source code, how to obtain the information required?
- If the HTML code is well-structured and its tags have (more or less) unique names, we can navigate the HTML elements to get the information we want.
- The `beautifulsoup4` package converts the HTML code into a Python object that can be navigated using properties and functions.

Some HTML terms

- Consider `<a href="http://www.bccp-berlin.de" target="_blank">BCCP</a>`
- HTML Elements
    - The entire thing is an HTML element. Specifically, it is a link leading to the BCCP website and displayed as "BCCP".
    - HTML elements usually consist of a start tag and an end tag.
- HTML Tags
    - The start tag of the element above is `<a>` and the end tag is the corresponding `</a>`
    - Start tag can and sometimes must contain attributes.
- HTML Attributes
    - The `<a>` tag contains the attribute `href` and `target`. `href` specifies the destination to which the link should lead and `target="_blank"` specifies that the link should be opened in a new window.
    - For web scraping purposes, the attributes `class` and `id` are usually useful as these are often used to identify certain (groups of) elements.

Basic HTML documents structure

- HTML documents have a tree-like/nested structure
- Elements can contain various levels of sub-elements that in the end contain some content
- A very simple HTML document could look like this:

```
<html>
<body>
  Hi all! <br>
  Do you know
  <a href="http://www.bccp-berlin.de" target="_blank">BCCP</a>?
</body>
</html>
```

Tree structure

```
html>,html>
```

Example for today

- Let's scrape the details of all upcoming BCCP events:
  `http://www.bccp-berlin.de/events/all-events/`
- Steps:
  1. Analyze HTML structure
  2. Save information on events available on the front page
  3. Loop through individual event pages to get details
  4. Combine to DataFrame

Day 2

Text pattern matching

Day 3

Table of Contents

Day 3

Browser automation

Day 3

Own script