RAship Web-Scraping Task

Vatsl Goswami

April 2025

1 Steps to Run Python Script

The Python script involves a lot of modules as well as a text-encoding model from HuggingFace. Please follow these steps to ensure the script runs correctly on your laptop:

1. Download the zip folder and navigate to the folder directory in your terminal.
2. Run the following command to download all the required libraries: `pip install -r requirements.txt`
3. Run the python script once using `python scrape.py`. The script will first check if you have the encoder model *all-miniLM-l6-v2* downloaded locally. Let it run and download the respective model (it's only 90MB).
4. Re-run the script once your downloads are complete, and you should get the desired output displayed in your terminal.

If you prefer using python-3.0 or higher, please replace the *pip* and *python* commands with *pip*3 and *python*3.

2 Questions

a) Extract/Web Scrape all the information for each event given in the Main LCW url. (Table 1, column 3)

Ans) To do this we simply use the Requests library to search and extract HTML info from the desired links. Then, using BeautifulSoup libary, we search for the HTML-tag which has a class-attribute of $conflict\_content$. This is the tag under-which landconflicts.org stores all its desired textual information for the page.

b) Using the information extracted in part a) identify, sort and provide the following data. Explain briefly your logic of selection.

Ans) All the information was directly found in the page under different sections. Hence, I did not need to do any data manipulation to find relevant results. Instead, I simply searched the HTML-code for the specific tags that contained the required information and extracted it.

i) Summary of the event using 20 words max: I extracted the summary by simply finding the title of the conflict article. This best summarizes the page content in less than 20 words. To find this, we search for the $conflict\_title$ div-tag in the HTML code

ii) Start date of conflict: The start date of the conflict was also directly provided in the page, under the div-tag named $conflict\_stats$.

iii) Land area affected: The affected land area was also found in the $conflict\_stats$ section.

iv) Sector of the conflict event: The sector involved in the conflict was found under the $conflict\_insights-wrap$ div-tag.

v) Reasons/Cause of the conflict: The cause or reasons behind the conflict were also under the $conflict\_insights-wrap$ div-tag.

vi) Legal laws violated: Any laws that were potentially violated were listed under a div-tag with the id $legal\_data$.

vii) Demand of the affected community: The demands made by the affected community were under a div-tag with the id name $fact-sheet$.

viii) Type of Land: Information ype of land was under the div-tag $conflict\_body-content$.

ix) Total investment, if any : This was also under the div-tag $conflict\_body-content$.

c) Extract/Web scrape all the information for each event from all the secondary urls. (Table 1 col 4)

Ans) To extract information from all the secondary urls, we first search for all the <a> tags inside the primary article's Summary section. Then, applying the following logic for each link, I extracted information from them. First, I browsed through each of the secondary sources manually and found the following common points.

- All the article titles/headers were placed in the first <bdi> or <h1> tag of the page. Hence, we could simply search for the first appearance of these tags to extract the article title from each of the secondary links.
- All the article body content was placed in <p> tags. Except for Times of India, in which the article data was placed in a div-tag with class name $\_s30J$. Hence, we could use search for both of these tags to get the article body.

d) Sort, present and categorize the textual information extracted from each secondary urls in part c). You can use data headings presented in i)-ix) to present the extracted information. This should be a separate output than in part a) and b).

Ans) After observing the structural layout for each of the secondary links, I noticed that all of them follow a different HTML-layout (owing to the fact that they are all from different news sites). Therefore, there is no way for us to extract the relevant information directly, as we did in part (b).

Secondly, after reading some of the secondary links manually, I also observed that not all links contain data relevant to all the 9 headings. Instead, each of the secondary link was placed in the primary article to act as a supporting source for certain points/facts. Thus, it did not make sense to organize each of the secondary links' text into the 9 headings as the information they contain is not really relevant to all the headings. Thus, I decided to follow the following strategy to extract relevant supporting information from each of the secondary links:

- Start by extracting the textual content of each secondary article. Then, divide the article text into single-sentence chunks using a regular-expression split (that splits the text at {.?!}). We now have divided the article into small chunks that can be passed through pass a text-encoder (in this case, HuggingFace's "all-miniLM-l6-v2" model as it's free and works well for simple tasks like this).

- The text-encoder takes each sentence and returns a vector embedding that encodes the semantic meaning of the sentence. We do this for all sentences in the secondary articles and store the embeddings, along with their corresponding English-sentences, in our *embeddings_map*.

- Now, for getting the summary of each of the secondary articles, we simply present the title we extracted in part (c).

- For the other headings, we perform a semantic search to extract relevant segments. To start, we use the heading (for e.g, "Legal Laws violated") and concenate it with the answers extracted in (b) (for e.g., Provision of Panchayats Act) and pass it to our text-encoder.

- We now have a vector-embedding for our query. We can now use a $cosine-similarity$ search to find all the vectors in our *embeddings_map* that are similar to our search query. We take the top 3 most relevant sentences, and if they have a *similarity score* > 0.4, we print out that sentence along with the link that we extracted it from.

- If there are no sentences with a score of more than 0.4, then we output *"No relevant information found."*

The Result: Our script does the following:
- Extracts information from all the secondary links
- Divides them into one-sentence chunks (to find highly-relevant sentences, since long sentences contain a lot of filler words like *"a", "the"* that take away from their semantic meaning).
- Performs a similarity search to extract top-3 sentences that are relevant to each of our 9 headings.

e) Please use your logic on presenting data in part d) in relation to part b)? What is your inference on the information provided for each of the three events in Table 1? Explain briefly.

Ans) The difference in how we presented data in part(b) and (d) is that the information was directly available in part(b) and thus we could directly extract each heading's information. However, for part (d), we had to do a semantic search to find information in each of the articles that was relevant to the headings and thus arranged them according to their semantic meaning.

The information provided for each of the three events in Table 1 is sufficient for us to scrape and extract a lot of information from the main and secondary links. Using a larger table, containing 1000s of rows, we can potentially scrape all the necessary information about a large number of land-conflicts and could organize them as we did for these 3 links.