APPROACH

Data Crawling

1. The scraping starts at the *popular* page under *opinions* section of *debate.org* website. The URL of this page is defined in the ***start_requests()*** method of the *QuotesSpider* class. This method yields a request to scrape the specified "start URL".

2. The scrape request generated a response and calls the ***parse()*** method with the generated response passed as an input parameter.

3. In the ***parse()*** method, we scrape the URLs and Debate ID of the top 5 most popular debates. A request is then made to scrape each of these debates iteratively one after another.

4. The scrape request calls the ***parse_debate()*** method with the generated response and the debateID of that debate passed as input parameters.

5. In the ***parse_debate()*** method, the first page of the particular debate is scraped. The *topic* and *category* are scraped and stored. The *title* and *body* of the pro and con arguments are scraped separately from the first page using scrapy selectors and stored in corresponding lists.

6. In order to scrape the arguments, present in next pages of the debate, a POST request has to be made with appropriate headers and body. The header and body are generated by observing the behaviour from the front end, upon clicking the "Load More Arguments" button and observing the payload data being passed from the "Network" tool of the browser.

7. An ***inline*** scrape POST request is generated for each page iteratively. The response is obtained in JSON format. The HTML text is extracted from the JSON data.

8. The extracted text is split into a list with two elements based on a separator ***"{ddo.split}"*** – first element of the list being the PRO arguments and the second element being the CON arguments in that particular page. The idea is that in the extracted text, all the "yes" arguments occur before the *"{ddo.split}"* and all the "no" arguments occur after it.

9. The two strings are converted into an HTML object in order to be able to use Scrapy selectors on them. The pro and con arguments are scraped separately using selectors and appended to the already existing pro and con arguments list.

10. The scraping is carried out for each page, until data is not found in a page, at which point the loop terminated.

11. The topic, category and the list containing the title and body of the pro and con arguments are returned as a JSON object for each debate and stored in ***data.json*** file.

Preliminary Statistics

1. The json data that is generated by data crawling is first opened into a *TextIOWrapper* which is then fed to ***json.load()*** to convert it into a list called *json_data*.

Task 1: *A histogram of argument lengths*

a. The length of arguments is calculated by the **number of tokens**.
b. A function ***Argument_Length()*** is called with the above list to calculate the length of all arguments as the number of tokens irrespective of topics or categories. This function iterates over all the arguments one by one.
c. Next, another method called ***tokenization()*** is called with each sentence to calculate the number of tokens by a white space and returns the number of tokens of that sentence back to ***Argument_Length()***
d. Finally, a list *token_size_lst* is returned which contains the length of each sentence present in json file after performing above steps iteratively.

Task 2: *Bar graph representing pro/con arguments per category*

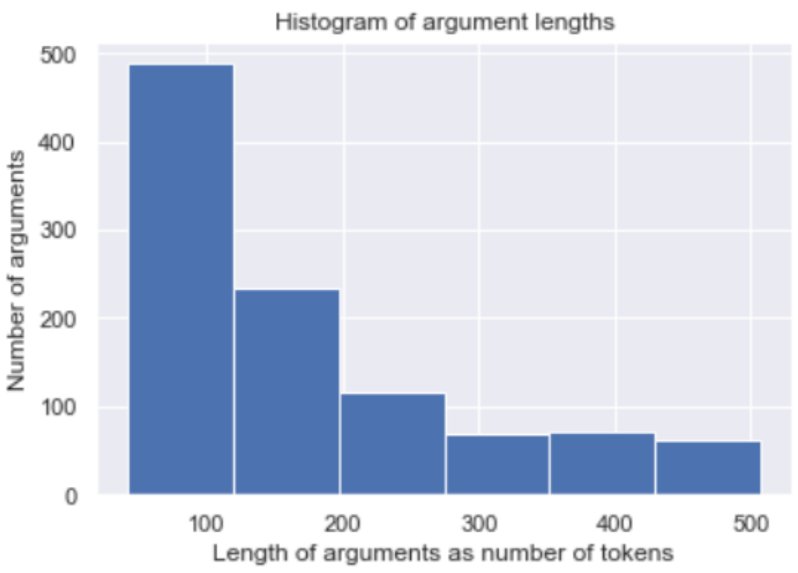
a. A grouped bar chart is generated to distinguish between number of pro arguments and con arguments per category.
b. A function ***Category_Title()*** is called with the list *json_data* to count the number of pro and con arguments.
c. Before adding a category to a new list, a condition to check for identical category is employed. Therefore, only distinct categories that are present in *json_data* are added to this list.
d. Pro and con arguments are added to the separate lists by iterating over distinct categories. Later, the number of pro and con arguments of the same category that is already present in the list are added accordingly in their respective lists.
e. Finally, three different lists that contain categories, pro and con arguments separated based on categories are returned.

Task 3: *Bar graph representing pro/con arguments per title*

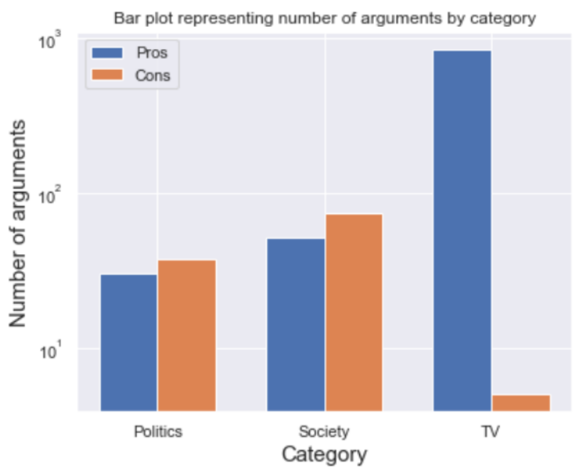
a. A grouped bar chart is generated to distinguish between number of pro arguments and con arguments per title.
b. A function ***Category_Title()*** is called with the list *json_data* to count the number of pro and con arguments.
c. The title of each topic, pro and con arguments of the same topic are added to three separate lists iteratively by performing dictionary operations over *json_data.*
d. Finally, three different lists that contain titles, pro and con arguments separated based on titles are returned.

Visualizations:

1. A histogram of argument lengths is plotted using *matplot.pyplot* using ***hist()*** function.
   Representation of axes:
   - **x-axis**: Length of arguments as number of tokens
   - **y-axis**: Number of arguments

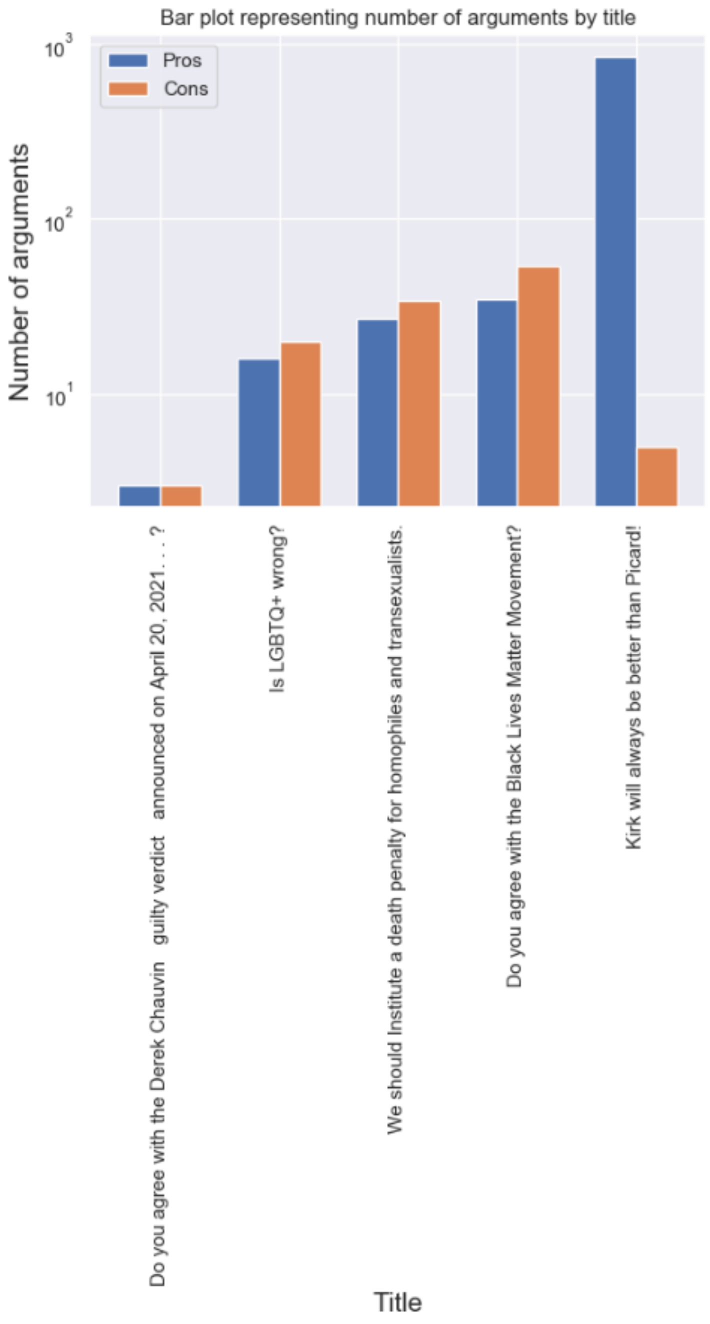


2. Grouped bar graphs that represent number of arguments per category and title are plotted using *matplotlib.pyplot* using ***bar()*** function. As one of the titles has a large number of pro arguments, the y-axis scale has been made ***logarithmic*** for better view and understanding. A legend for each of the graphs is created to represent pro arguments (blue) and con arguments (orange).
   Representation of axes (per category):
   - **x-axis**: Category
   - **y-axis**: Number of arguments



Representation of axes (per title):

- **x-axis**: Title
- **y-axis**: Number of arguments



EXECUTION INSTRUCTIONS

Before proceeding with the execution, it has to be ensured that the necessary packages and libraries are present in the system. From the scope of this assignment, the following packages and libraries need to be installed for successful execution of the assignment:
scrapy, urllib, json, scrapy-inline-requests, numpy, matplotlib, seaborn, mplcursors

In case any of the above-mentioned packages are missing, it can be installed using the command **pip install <package-name>.**

Steps for execution:

1. Unzip the file: Assignment1-HardlyHuman.zip.

2. The unzipped folder will have the following contents inside:

    a. **Code:** This folder contains the Scrapy project called *data_acquisition_debates* and a Jupiter notebook file called *Preliminary_Statistics.ipynb*

    b. **Documentation.pdf:** PDF file describing the approach and execution instructions.

3. Navigate to the Scrapy project folder *data_acquisition_debates*. Inside the project, in the ***settings.py*** file, the *DOWNLOAD_DELAY* parameter is set to 5, *CONCURRENT_REQUEST*S parameters are set to 16 and *FEED_EXPORT_ENCODING* parameter is set to 'utf-8'. The file should be retained as is.

4. Inside the scrapy project, navigate to the directory called ***spiders.*** The folder contains ***spider_popular_debates.py*** which is the code for the crawler called ***debate_crawler***. Run the crawler using the following command on command line/terminal from inside the project folder:

    ```
    scrapy crawl debate_crawler -o "data.json"
    ```

    Running this command generates a json filed called *data.json* inside the scrapy project folder.

5. Running the ***Preliminary_Statistics.ipynb*** in Jupiter notebook will generate the preliminary statistics from the *data.json* file created in Step 3. **The *data.json* file should be placed in the same folder as the notebook file**. Another option is to modify the *"path"* variable in the code accordingly.

Group: Hardly humans
Siddhanth Janadri - 6882502
Sneha Hiremath - 6881479
Sanjay Gupta - 6882964
Sai Nikhil Menon - 6882524