Scrapping StackOverflow questions with BeautifulSoup, Scrapy and Selenium

Palina Tkachova 456044
Dominik Zabinski 306068

About the project

What?
Scrapping 1500 StackOverflow questions and related attributes such as question title, link, votes, views, answers, and full question. Questions are scrapped from 100 pages (15 each) to satisfy course requirement.

Why?
To perform simple data analysis on which tags attract more votes, answers, and views within web-scrapping topic. In other words, which tags user should put asking a question about web-scrapping.

How?
We created 3 scrappers:

1. Beautiful Soup

1. Setup: The scrapper uses Python's requests and BeautifulSoup libraries to fetch and parse HTML from StackOverflow's web scraping tagged questions.
2. Scraping: It iterates over multiple pages (via changing an URL), scraping all the info about questions and for each question on the page, extracting the title, link, tags, votes, views, and number of answers.
3. Throttling: After scraping each page, the script pauses for one second to respect server resources.
4. Continuation: The script fetches 100 pages by iteration. It contains manual check to scrap only 15 questions per page as in the view (we parametrized the script to get higher/lower number of questions per request).
5. Data Storage: The scraped data is stored in a list of dictionaries and then written into a panda's DataFrame.

2. Scrapy

1. Setup: The code imports the necessary libraries and sets up the Spider class for the Scrapy spider.
2. Configuration: The spider is named "stackoverflow_spider" and is configured with the start URL and custom settings such as user agent, download delay, and HTTP cache.

3. Page Counter and Limit: The code initializes a page counter and sets a boolean parameter to limit the number of pages to be scraped.
4. Parsing: The parse method is defined to handle the response from each page. It uses CSS selectors to extract the question elements from the HTML response.
5. Question Extraction: Inside the parse method, the code extracts the question title, link, tags, votes, answers, and views from the question elements using CSS selectors.
6. Yielding the Data: The extracted question information is yielded as a dictionary containing the relevant fields.
7. Pagination: The code increments the page counter and checks if the limit_pages parameter is set to True. If the page count reaches 100, the crawling process stops.
8. Next Page Request: The code extracts the URL for the next page and sends a new request using scrapy.Request with the callback set to the parse method for further processing.
9. Throttling: The scraping process includes a download delay of 0.2 seconds to respect server resources.
10. Data Storage: The scraped question information is typically processed in the yield statements, where it can be further processed, stored, or passed to other pipelines for storage or analysis.

3. Selenium

1. Selenium Setup: Imports necessary libraries and configures the ChromeDriver with options and preferences.
2. Scrape_website function:
    a. Coordinates the scraping process, taking a URL and a boolean parameter to limit pages.
    b. Determines the number of pages and CPU cores.
    c. Creates a list of page numbers.
    d. Uses multiprocessing.Pool to distribute scraping task across processes.
3. Scrape function:
    a. Scrapes a single page, taking a page number as input.
    b. Constructs the URL, initializes the Chrome driver, and navigates to the URL.
    c. Handles cookie acceptance if present.
    d. Extracts question information using CSS selectors.
    e. Stores data in a dictionary format.
    f. Quits the driver after a short delay.
4. Writing to CSV:
    a. Writes scraped question information to a CSV file.
5. Multiprocessing:
    a. Utilizes the multiprocessing module for parallel scraping.
    b. Determines the number of CPU cores.
    c. Uses the Pool object's map function to distribute tasks across processes.
    d. Collects and returns the results.

Output:

Above scrappers are designed to produce similar output which is a list of dictionaries transformed into CSV files. Attributes are:

- Title: string containing title of a question
- Link: string containing link to the page of the question
- Tags: string containing list of tags of this question
- Votes: string containing number of votes for this question
- Views: string containing number of views of this question
- Answers: string containing number of answers for this question

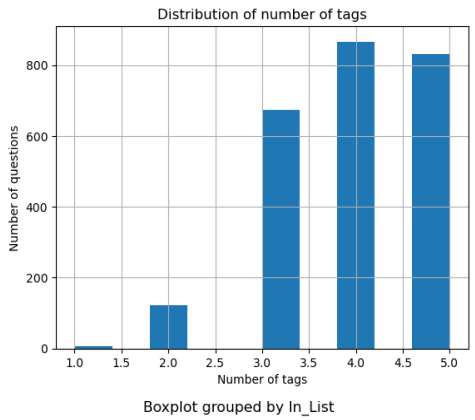
Attributes are to be converted to a suitable for data analysis type.
Base URL: https://stackoverflow.com/
URL: https://stackoverflow.com/questions/tagged/web-scraping?tab=Votes

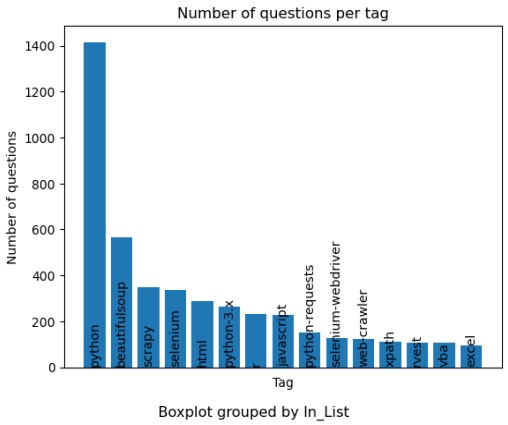
Data analysis

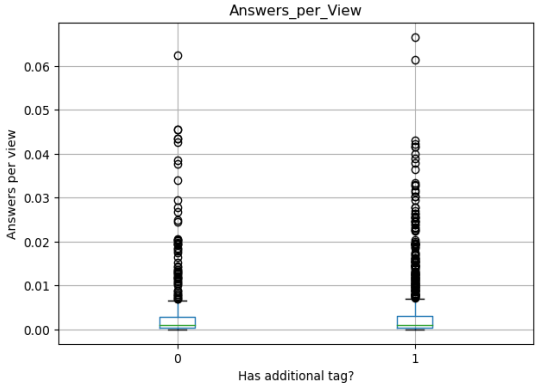
Based on the data extracted we can tell that

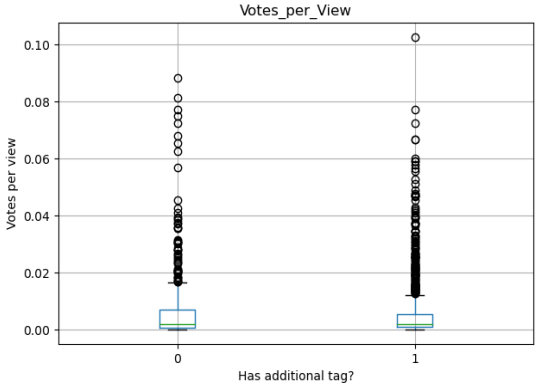
- people usually used 4 tags, Python is far more popular than R (and Excel)
- amongst free approaches, most popular is BeautfiulSoup (or at least most people have questions about it)
- when user adds extra tag, indicating which approach is being used, it corresponds to higher number of views, answers per view and vote per view.

Conclusion: when posting a question, it is beneficial to add extra tags, since it gets more views and in result might lead to higher number of better answers.









Division of work

Palina Tkachova	Dominik Zabinski
1. Full Scrapy scrapper development 2. Full Selenium scrapper development	1. Full Beautiful Soup scrapper development 2. Full Data analysis