Shubham Joshi 437962

**WEB SCRAPING PROJECT**

**Introduction:**

In the realm of web data extraction and analysis, this report delves into a project centered around web scraping, an essential technique for gathering information from online sources. The objective of this project is to create three distinct web scrapers employing different tools: Beautiful Soup, Scrapy, and Selenium.

These scrapers will collectively extract data from the renowned TED Talks website (https://www.ted.com/talks), a platform dedicated to the dissemination of innovative ideas and fostering a global community. The extracted data will encompass crucial details such as speaker names, video URLs, talk titles, and posting dates. Through this initiative, we aim to showcase the capabilities of these scraping methodologies, provide a brief analysis of the amassed data, and evaluate their comparative performance. This report unveils the technical intricacies of each scraper, the nature of the gathered output, and rudimentary insights that highlight the potential utility of the collected data.

**Description of the Project:**

These methodologies facilitate automated data retrieval from web pages by utilizing diverse approaches such as HTML parsing, XPath traversal, and browser automation. Our objective is to build an extensive dataset of TED Talks available on the platform through meticulous configuration and execution of these scraping techniques. The subsequent sections of this report will delve into the technical nuances of each scraper, followed by a concise examination of the obtained output data and its basic potential for analysis.

Python's Beautiful Soup library offers an uncomplicated yet potent technique for web scraping. Its user-friendly API enables parsing and extraction of information from HTML and XML documents. In contrast, Scrapy, a robust and highly adaptable web scraping framework, presents a more structured and scalable strategy for extracting data. Finally, Selenium, a tool for browser automation, empowers us to scrape dynamic websites heavily reliant on JavaScript for content rendering.

Following the collection of desired data using these scraping techniques, we will proceed to perform a straightforward analysis of the amassed information. This analysis will unveil insights into the dispersion of talks across various sectors. Through a comparison of the performance of the three scraping methods, we can assess their efficiency in terms of speed, implementation ease, and dependability. Such a relative analysis will aid in determining the most suitable approach for forthcoming web scraping projects, taking into account specific requisites and challenges.

**Description of scraper mechanics**

Beautiful Soup

The provided code demonstrates the use of Beautiful Soup and XML parsing in tandem to extract specific elements from the HTML content of web pages. This process allows the script to retrieve targeted information, which is then collected, organized, and stored for further analysis. Let's break down the mechanism in detail:

1. **Importing Required Libraries:**

The code begins by importing essential libraries: Beautiful Soup for HTML parsing , urllib for making HTTP requests , lxml for XML parsing, time for controlling delays , and pandas for data manipulation and analysis.

2. **Page Limit and Data Storage Initialization:**

The script uses a Boolean parameter, page\_limit, to determine whether the scraping process should be limited to a specific number of pages. I also initializes empty lists (video\_titles, speaker, date, video\_links) to store the scraped data.

3. **Iterating through Pages:**

A loop iterates through a range of pages, from 1 to the specified number. For each page, the script constructs the URL for the TED Talks page using the page number.

4. **Fetching and Parsing HTML:**

Using the constructed URL, the script makes an HTTP GET request to fetch the HTML content of the page. The fetched HTML content is then parsed using Beautiful Soup with the html.parser parser, creating an object that represents the parsed HTML.

5. **Extracting Speaker Names:**

The code uses Beautiful Soup to find all `h4` elements with the class `'h12 talk-link\_\_speaker'`. It iterates through these elements and extracts the text content, which represents the names of the speakers. The extracted speaker names are appended to the speaker list.

6. **Extracting Talk Titles and Video Links:**

The script locates each `div` element with the class `col` within the parsed HTML. For each of these elements, it further searches for an `h4` element with the class `'h9'`. This element contains the talk title. The code also finds an anchor (`<a>`) element with the class `'ga-link'` and extracts the `href` attribute, which represents the video link. The extracted talk titles and video links are appended to the `video\_titles` and `video\_links` lists, respectively.

7. **Extracting Posting Dates:**

The script uses Beautiful Soup to locate all `span` elements with the class `'meta\_val'` that contain the posting dates. The text content of these elements, representing the posting dates, is extracted and appended to the `date` list.

8. **Constructing a Data Dictionary and DataFrame:**

Once all relevant data is extracted, the script constructs a dictionary named `data\_dictionary`. This dictionary is structured with keys such as `'Title'`, `'Speaker'`, `'Date Posted'`, and `'Link'`, each corresponding to the collected data lists. The `data\_dictionary` is then used to create a pandas DataFrame, with data categories becoming columns and individual data points becoming rows.

9. **Data Organization and CSV Output:**

The DataFrame is transposed to ensure that data categories become rows and data points become columns. The rows are sorted alphabetically based on the talk titles. Finally, the processed DataFrame is saved as a CSV file named `'data\_bsoup.csv'`.

In conclusion, the provided code demonstrates a systematic approach to web scraping using Beautiful Soup and lxml. It highlights the capability of these tools to navigate and extract specific elements from HTML content, which are then structured and stored for further analysis and exploration.

**Selenium**

The provided code demonstrates the use of the Selenium library to perform web scraping of TED Talks data. Selenium is a powerful tool that allows automated interaction with web pages, making it particularly useful for extracting data from websites that require dynamic content loading or user interactions. Here's a detailed breakdown of how the Selenium scraper works:

**1. Importing Required Libraries:**

The code begins by importing the necessary libraries: webdriver from the Selenium library, `By` to locate elements using various strategies, time for controlling delays, and pandas for data manipulation and analysis.

**2. Configuring WebDriver and Headless Mode:**

The script sets up the WebDriver for Firefox by providing the path to the geckodriver executable and configuring options. The `headless` option is set to False, indicating that the browser window will be visible during scraping. You can set it to `True` to run the browser in headless mode.

3. **Initializing WebDriver and Loading URL:**

The WebDriver is initialized using the configured options and geckodriver service. The URL of the TED Talks page is specified, and the WebDriver navigates to this URL using the get() method.

4. **Setting Page Limit and Data Storage:**

A Boolean parameter is used to determine whether the scraping process should be limited to a specific number of pages. Lists (video\_titles, speaker, date, video\_links) are initialized to store the scraped data.

5. **Looping Through Pages and Scraping Data:**

The script enters a loop that iterates through a range of pages, from 2 to the specified max\_pages. Inside the loop, a delay of 1 second is introduced to allow the page to load completely.

6**. Extracting Video Titles, Speaker Names, Dates, and Links:**

For each page, Selenium locates and extracts the elements containing video titles, speaker names, posting dates, and video links. It uses XPath expressions to identify the specific elements. Extracted data is appended to the respective lists.

**7. Navigating to Next Page:**

After extracting data from the current page, Selenium navigates to the next page using the driver.get() method, with the URL constructed based on the page number.

**8. Creating a Data Dictionary and DataFrame:**

Once data has been extracted from all pages, a dictionary named data\_dictionary is constructed. This dictionary contains keys like Name, Speaker, Date, and Links, corresponding to the collected data lists. The dictionary is used to create a pandas DataFrame.

**9. Sorting and Saving Data:**

The DataFrame is sorted alphabetically based on video names using the sort\_values() method. The processed DataFrame is then saved to a CSV file named data\_selenium.csv.

**10. Cleaning Up:**

After scraping is complete, the WebDriver instance is closed using the quit() method, effectively closing the browser window.

In summary, the Selenium scraper automates the interaction with TED Talks pages, extracting relevant information from dynamically-loaded content. This approach is particularly useful for websites with JavaScript-driven content, as it emulates user interaction to retrieve data. The collected data is stored in a structured format for further analysis and exploration.

**Scrapy**

The code showcases the use of the Scrapy framework to perform web scraping of TED Talks data. Scrapy is a versatile and powerful Python framework designed for web crawling and scraping tasks, providing a structured and efficient approach. Here's a detailed breakdown of how the Scrapy scraper operates:

**1. Importing Required Libraries:**

The script begins by importing essential libraries, including scrapy for the Scrapy framework, pandas for data manipulation, and CrawlerProcess to manage and run the Scrapy spider.

**2. Setting Page Limit Flag:**

A Boolean parameter, page\_limit, is used to determine whether the scraping process should be limited to a specific number of pages.

**3. Defining the Spider Class:**

The core of the code is the definition of the Scrapy Spider class named `CareerSpider`. This class inherits from the `scrapy.Spider` base class. It contains attributes such as the name of the Spider and the allowed\_domains that restrict the Spider to crawl URLs within those domains.

**4. Generating Start URLs:**

The start\_urls attribute is populated with a list of URLs to start the scraping process. These URLs are generated based on the specified number of pages using a list comprehension.

**5. Parsing Function:**

The parse method is the heart of the Spider. It receives the response from the URLs visited and extracts the required data using XPath expressions. Video titles, speaker names, posting dates, and video links are extracted from the response using the response.xpath method. Extracted data is then stored in respective lists.

**6. Creating and Manipulating Data:**

After extracting data from the response, a dictionary named data\_dictionary is created. This dictionary organizes the scraped data with keys such as Title, Speaker, Date Posted, and Link. A pandas DataFrame is constructed from this dictionary, and the DataFrame is sorted in ascending order based on the video titles.

**7. Saving Data to CSV:**

The sorted DataFrame is saved to a CSV file named data\_scrapy.csv. The mode parameter is set to append the data to the existing file instead of overwriting it.

**8. Running the Spider:**

A CrawlerProcess instance is created to manage and run the Scrapy spider. The crawl method is used to start the CareerSpider spider defined earlier. The process.start() method is called to initiate the scraping process.

The Scrapy spider is initiated by running the command `scrapy crawl ted.py` in the Anaconda prompt. This starts the crawling process, and the Spider visits the specified URLs, extracts data, and saves it to the CSV file.

In summary, the Scrapy scraper leverages the Scrapy framework's structured approach to perform web scraping. It automates the process of navigating through pages, extracting data using XPath, and saving the organized data to a CSV file. This approach provides modularity, scalability, and efficient handling of web scraping tasks.

**Technical description of the output**

The output consists of a tabular format with four columns: Title, Speaker, Date Posted, and URL. Each row represents a different Ted talk video and provides information about the Video.

This tabular output serves as a structured representation of the scraped data obtained from the website. It allows for easy comprehension and analysis of the information, enabling further exploration and manipulation of the data as needed.

**Data Analysis**

A graph showing a line

Description automatically generated

The above line chart represents Year over Year released videos by Ted.

* The data shows that total scrapped records are 3600.
* We could see that the Ted started releasing the ted talk videos from 2016.
* In 2018 Ted released maximum number of videos which is equal to 648 videos.
* From its peak in 2018 , it is in a downtrend and is been releasing lesser and lesser videos.