Extract Information from Websites (use any 100 websites)
Report on Web Scraping Project

Objective:

Develop a solution to extract the following information from a list of websites:

- Robots.txt URL
- Sitemap URL
- Contact email
- Contact Address
- Contact number with country code
- Language by HTML
- CMS/MVC
- Category of website

Approach:

1. Environment Setup:

- Installed required Python libraries in Google Colab for web scraping (‘requests’, ‘BeautifulSoup’) and database operations (psycopg2.connect’).

- Set up the PostgreSQL database ‘project’ and the table ‘companies_info’.

2. Database Configuration:

- Created a PostgreSQL table ‘companies_info’ with columns for each type of data to be stored.

- Configure the database connection parameters.

3. Web Scraping Functions:

- Fetching HTML Content: Developed a function to fetch HTML content using ‘requests’.

- Extracting Information: Created functions to extract specific information from HTML using ‘BeautifulSoup’:

- Robots.txt URL: Manage crawling traffic if you think your server will be overwhelmed by requests from Google's crawler, or to avoid crawling unimportant or similar pages on your site.

- Sitemap URL: A sitemap tells search engines which pages and files you think are important in your site, and also provides valuable information about these files.

- Tech Stack: Identified common CMS platforms, and MVC frameworks.

- Website Language: Extracted the ‘lang’ attribute from the HTML tag.

- Category of website: Classified the website into predefined categories based on keywords in the text.

- Contact email: Extracted the email id from the HTML tag.

- Contact Address: Extracted the contact address from the HTML tag.

- Contact number with country code: Extracted the contact number of the companies with country code from the HTML tag.

4. Execution Loop:

- Created a list of URLs from various categories to be scrapped.

- Iterated through the list, scraping and storing data for each website.

- Introduced a delay between requests to avoid overwhelming the servers and to comply with polite scraping practices.

5. Data Storage:

- Developed a function to insert the scraped data into the PostgreSQL database.

- Managed database connections and ensured proper closing of connections after operations.

6. Execution Loop:

- Created a list of URLs from various categories to be scrapped.

- Iterated through the list, scraping and storing data for each website.

- Introduced a delay between requests to avoid unnecessary the servers and to comply with polite scraping practices.

Challenges Encountered:

1. Connection Issues:

- Encountered issues with remote servers closing connections or blocking requests. This was partially mitigated by adding delays between requests.

2. Diverse Website Structures:

- Different websites have varying structures, making it challenging to develop a one-size-fits-all extraction approach. This was addressed by creating flexible and robust extraction functions that could handle various HTML structures.

3. Data Quality and Completeness:

- Ensuring the completeness and quality of the extracted data was challenging. Implemented thorough checks and fallback mechanisms to handle missing or incomplete data.

Code Explanation:

1. Database Connection:

- Established a connection to the PostgreSQL database using psycopg2.connector'.

- Defined the cursor for executing SQL queries.

2. URL List:

- Provided a list of URLs from various categories such as travel, news, e-commerce, technology, education, entertainment, health, finance, blogs, and social media.

3. Extraction Functions:

- ‘extract_tech_stack(soup)’: Identifies tech stack components from the scripts and meta tags.

- ‘extract_website_language(soup)’: Extracts the language attribute from the HTML tag.

- ‘extract_category_of_website(soup)’: Determines the category of the website based on keywords.

4. Scraping Function:

- ‘scrape_website(url)’: Fetches HTML content and applies the extraction functions to gather all required information.

- Handles potential errors gracefully by checking if content exists before accessing it.

5. Database Insertion:

- ‘save_to_database(data)’: Inserts the scraped data into the PostgreSQL database.

6. Main Execution Loop:

- Iterates through the URL list, scrapes each website, and saves the data into the database.

- Adds a delay between requests to avoid overwhelming servers.

Conclusion:

A web scraping solution was successfully deployed for the project in order to extract and save data from a variety of websites. The solution addressed a number of problems with inconsistent website structures, missing data, and connection problems. The final dataset offers a thorough compilation of data appropriate for additional use and research.

With further development, the scraping system could be made more effective and able to handle a larger variety of websites and data kinds.

Project Links:

Google Colab Link:

https://colab.research.google.com/drive/1gjnqh4vgOXjqeOf_vNcvH5GHS8Ag18z1?usp=sharing

Github Link:

https://github.com/chandan523/web-scraping-project