**Project Documentation**

**Project Title**

**Web Archive Data Scraper**

**Project Source**

* **GitHub Repository:** [GitHub Link For Code Files](https://github.com/huzaifasaeed123/Data-Scraping-Projects/tree/1ac3702de45ab40f21d44675b9cf61fde08c0fee/Fiver%20Projects/Archive%20Data%20Scraper#web-archive-data-scraper)
* **Google Drive:** [Click here for link that contain Complete Scraping Files of acsboe.org](https://drive.google.com/drive/folders/13xe_QySTipbudb3ko8Hr1p4hcRF6_MOR?usp=sharing)
* **Separate CSV File:** [CSV File For acsboe.org](https://drive.google.com/file/d/1N7k48vpK2r4mPrq-VJCAP6qZdDVwGuoF/view?usp=drive_link)

**Input(s)**

* **Domain Name**: Standard Python input().

**Packages**

* pandas
* dataset
* requests
* socket
* urllib.parse
* time
* random
* os
* csv

**Workflow**

1. **Ask for Domain**:
   * The user is prompted to input a domain name.
   * The domain is validated using the is\_valid\_domain function.
2. **Parse archive.org Domain List**:
   * The domain's archived data is fetched from archive.org using the fetch\_archive\_data function.
   * Only valid HTML lines (status code 200) are included.
3. **For Each Valid Row**:
   * Scrape the following data points:
     + timestamp
     + domain
     + url
     + page source (html)
   * Save data to a SQLite database using the dataset package.
4. **When All Rows Are Scraped**:
   * Select all rows from the database into a pandas DataFrame.
   * Write the DataFrame to a CSV file.

**Code Flow**

**Main Function**

The main() function drives the entire workflow:

1. **Input Domain Name**:
   * Prompts the user to enter a domain name.
   * Calls is\_valid\_domain to validate the domain.
   * If the domain is invalid, it prompts the user again.
2. **Fetch Archive Data**:
   * Calls fetch\_archive\_data to get the archived data of the domain from archive.org.
   * If no data is found, it prints an appropriate message and exits.
3. **Scrape Data**:
   * Calls scrape\_data to scrape and save the HTML data from the archive.
   * If no valid HTML data is found, it prints an appropriate message and exits.
4. **Save to Database**:
   * Calls save\_to\_database to save the scraped data into a SQLite database.
5. **Load from Database and Save to CSV**:
   * Calls load\_from\_database to load the data from the database into a pandas DataFrame.
   * Saves the DataFrame to a CSV file.

**Function Definitions**

**is\_valid\_domain(domain)**

* **Purpose**: Validates if the given domain is valid.
* **Arguments**: domain (str): The domain name to validate.
* **Returns**: bool: True if the domain is valid, False otherwise.
* **Process**:
  + Uses socket.gethostbyname to check DNS resolution.
  + Uses requests.get to check the HTTP response.

**fetch\_archive\_data(domain)**

* **Purpose**: Fetches the archive data of the domain from archive.org.
* **Arguments**: domain (str): The domain name to fetch data for.
* **Returns**: list: A list of lines containing the archive data.
* **Process**:
  + Constructs the URL for archive.org's CDX search API.
  + Sends a GET request to the URL.
  + Returns the response text split into lines if the request is successful.

**scrape\_data(lines, domain)**

* **Purpose**: Scrapes the HTML data from the archived lines.
* **Arguments**:
  + lines (list): The list of lines containing the archive data.
  + domain (str): The domain name to scrape data for.
* **Returns**: list: A list of dictionaries containing the scraped data.
* **Process**:
  + Creates a folder for the domain if it doesn't exist.
  + Iterates through each line and checks if it's valid HTML with a status code of 200.
  + Constructs the archived URL and fetches the HTML content.
  + Saves the HTML content to a file and appends the data to the list.

**save\_to\_database(data, db\_url)**

* **Purpose**: Saves the scraped data to a SQLite database.
* **Arguments**:
  + data (list): The list of dictionaries containing the scraped data.
  + db\_url (str): The URL of the SQLite database.
* **Process**:
  + Connects to the database.
  + Inserts the data into the 'web\_archive' table.

**load\_from\_database(db\_url)**

* **Purpose**: Loads the data from the SQLite database into a pandas DataFrame.
* **Arguments**: db\_url (str): The URL of the SQLite database.
* **Returns**: pandas.DataFrame: A DataFrame containing the data from the database.
* **Process**:
  + Connects to the database.
  + Loads all rows from the 'web\_archive' table into a DataFrame.

**How to Use the Project**

1. Clone the GitHub repository or download the source code from the provided link.
2. Ensure all required packages are installed.
3. Run the main() function in the script.
4. Enter a valid domain name when prompted.
5. The script will validate the domain, fetch archive data, scrape HTML content, save it to a database, and export the data to a CSV file.
6. Check the output CSV file for the scraped data.

**Request for Feedback**

If you require any changes or have any feedback, please let me know. I am happy to improve the project and meet your requirements, as I offer unlimited revisions in my package.