# User Guide for Python Script: data\_collection\_scraper

## 1. Introduction

The data\_collection\_scraper Python script is designed for web scraping tasks. It extracts data from specified web sources, processes it, and stores the output in a structured format. This script is ideal for users needing to collect specific data from online sources and save it for further analysis.

## 2. Requirements

Before running the script, ensure you have the following:

- Python (version 3.6 or higher)  
- The following Python libraries:  
 - requests  
 - BeautifulSoup  
 - pandas  
 - os  
 - csv  
 - json (if applicable)

You can install missing libraries using pip:

pip install requests beautifulsoup4 pandas

## 3. Setting Up the Environment

1. Ensure all necessary libraries are installed.  
2. Open the script and configure any specific settings or parameters as required (such as the URLs to scrape or headers for requests).  
3. If the script requires authentication or API keys, make sure these are set securely within the script.

## 4. Directory Structure and Script Location

The script can be saved in any directory from which you intend to run it. If the script involves saving scraped data to files, ensure that the necessary folders exist or are created by the script. This will ensure that all output data is saved correctly.

## 5. Running the Script

1. \*\*Open Terminal or Command Prompt\*\*:

Navigate to the directory where the script is saved using:  
cd path/to/script-directory

2. \*\*Run the Script\*\*:

Execute the script by typing:  
python script\_name.py

## 6. Script Workflow and Explanation

This section provides a high-level overview of each part of the script:  
  
- Setting Up Requests and Headers:  
 The script initializes the requests session and sets headers, cookies, or authentication if required.  
  
- Web Scraping with BeautifulSoup:  
 The script makes HTTP requests to fetch web page content and uses BeautifulSoup to parse and extract specific data.  
  
- Data Processing:  
 Extracted data is cleaned and organized into structured formats, often a DataFrame for easy manipulation.  
  
- CSV Output:  
 The structured data is saved in CSV format (or JSON if required) for easy access and analysis.

## 7. Sample Output

The output CSV or JSON file will typically have the following structure:  
  
For CSV:  
| Column1 | Column2 | Column3 |  
|------------------|-------------------|----------------------|  
| Data1 | Data2 | Data3 |  
| ... | ... | ... |  
  
For JSON:  
{  
 'data': [  
 {  
 'Column1': 'Data1',  
 'Column2': 'Data2',  
 ...  
 }  
 ]  
}

## 8. Troubleshooting

- Library Import Errors: Ensure all required libraries are installed.  
- Connection Errors: Check your internet connection and the target website's accessibility.  
- Data Parsing Errors: Review the structure of the web pages being scraped to ensure compatibility with the script.

## 9. Checking Script Dependencies

To verify installed libraries, you can check their versions by running the following commands:

import requests  
import bs4  
import pandas  
print(requests.\_\_version\_\_, bs4.\_\_version\_\_, pandas.\_\_version\_\_)

## 10. Modifying the Script for Customization

If you want to scrape different websites or change the output format, you can modify the URL list, parsing logic in BeautifulSoup, or the output formatting sections in the script. These modifications allow the script to adapt to new sources or output requirements.

## 11. Appendix

Python Libraries Used:  
- requests: For making HTTP requests to fetch web page content.  
- BeautifulSoup (bs4): For parsing HTML and extracting specific data.  
- pandas: For data manipulation and saving CSV files.  
  
Important Notes:  
- Some websites may block or limit automated scraping. Ensure compliance with the website's terms of service.