**Australia tarrif scraping script documentation**

This documentation provides a detailed explanation of the Python script designed to scrape tariff classification data from the Australian Border Force (ABF) website - [Schedule 3 - Current tariff classification](https://www.abf.gov.au/importing-exporting-and-manufacturing/tariff-classification/current-tariff/schedule-3) (request limit: 2 requests/hr for my region)

**Script summary**:

* The script extracts data from multiple sections and chapters, processes it, and saves it into a CSV file.

**Overview:**

The script performs the following tasks:

1. Generates URLs for all sections and chapters of the tariff classification schedule.
2. Scrapes data from each URL, extracting tables and processing their content.
3. Cleans the data to ensure consistency and accuracy.
4. Combines the data from all pages into a single DataFrame.
5. Saves the combined data to a CSV file.

**Dependencies**

The script uses the following Python libraries:

* requests: To send HTTP requests and fetch webpage content.
* BeautifulSoup (from `bs4`): To parse HTML and extract data.
* pandas: To handle and manipulate the scraped data in a tabular format.
* re: To use regular expressions for cleaning and extracting specific patterns.

**Install the required libraries using:**

*!pip install* ***requests beautifulsoup4 pandas***

**Code Structure:**

The script is organized into the following functions and steps:

* **generate\_urls()**

Generates URLs for all sections and chapters of the tariff classification schedule.

* Parameters:

None.

* Returns:

A list of URLs (`list[str]`).

* Logic:
* The base URL is `https://www.abf.gov.au/importing-exporting-and-manufacturing/tariff-classification/current-tariff/schedule-3`.
* Sections and their corresponding chapters are defined in a dictionary (`sections`).
* URLs are constructed by appending the section and chapter numbers to the base URL(URL sequencing pattern as used by website)

Example Output:

[

"https://www.abf.gov.au/.../section-i/chapter-1",

"https://www.abf.gov.au/.../section-i/chapter-2",

...

]

* **scrape\_page(url)**

Scrapes data from a single webpage.

* Parameters:

-`url` (str): The URL of the webpage to scrape.

* Returns:

A list of rows (`list[list[str]]`), where each row represents a table row and contains the column values.

* Logic:

- Fetches the webpage content using `requests.get()`.

- Parses the HTML using `BeautifulSoup`.

- Locates all tables on the page using `soup.find\_all('table')`.

- Extracts data from each table row (`<tr>`) and column (`<td>`).

- Ensures each row has the same number of columns as the headers by padding with empty strings if necessary.

Example Output:

[

['1234', 'kg', 'Meat of bovine animals', '5% DCT: 4% DCS: $10', '0201.10.00', 'Some text'],

['5678', 'kg', 'Flowers', 'Free', '0601.10.00', '']

]

* **clean\_data(df)**

Cleans the combined DataFrame.

* Parameters:

`df` (pd.DataFrame): The DataFrame containing the scraped data.

* Returns:

A cleaned DataFrame (`pd.DataFrame`).

* Logic:

General Cleaning:

-Replaces multiple spaces with a single space in all columns.

**Reference Number** Cleaning:

-Extracts the numeric part (e.g., `0201.10.00`) from strings like 'View TCOs for 0201.10.00'.

-Uses regular expressions to find patterns like `\d{4}\.\d{2}\.\d{2}`.

**Goods** Column Cleaning:

-Replaces `=-` with `-`.

-Removes unnecessary special characters and normalizes spaces around dashes.

Final Cleaning:

-Removes empty rows.

-Trims leading/trailing whitespace from all string values and remove extra characters

* **Main Script**

Generate URLs:

-Calls `generate\_urls()` to create a list of URLs to scrape.

Scrape All Pages:

-Iterates through each URL and calls `scrape\_page()` to extract data.

-Combines the data from all pages into a single list (`all\_data`).

Create and Clean DataFrame:

-Converts `all\_data` into a DataFrame.

-Calls `clean\_data()` to clean the DataFrame.

Save to CSV:

-Saves the cleaned DataFrame to a CSV file (`tariff\_classification\_all\_sections.csv`).

**Script Usage:**

* Run the command **python scraper.py** in your preferred CLI (Command Line Interface) window with all dependencies installed in the environment.
* The script saves the scraped and cleaned data to `tariff\_classification\_all\_sections.csv`.
* Prints the first 5 rows of the combined table for verification.

**Error Handling:**

* Invalid URLs: If a URL is invalid or the page is unavailable, the script will skip it and continue.
* Empty Rows: Rows with no data are removed during cleaning.

**Limitations:**

* The script assumes that all tables on the webpage have the same structure. It meets requirement for this milestone, but might fail with tables with varying structure.
* Some of **ReferenceNumber** values are missing in the website
* Extra data cleaning is required after running the script for Columns such as **Goods**(replacing extra characters), **ReferenceNumber**(missing values due to random data encryption at product category rows, this missing values can easily be derived from the products ReferenceNumber under the category name.).
* 30% of the **reference Number** is missing in the website, while 12% is encrypted hence requires manual imputation.

**Improvements for upcoming milestones where possible scraping restrictions might exist:**

* Implement error handling for network issues or anti-scraping measures for restricted website(CAPTCHA, rate limits etc).
* Implement dynamic table structure script to scrap tables with different structures.
* Optimize the script for larger datasets using parallel processing instead of serial processing if necessary.