Notino Scraper

Alessia Michela Di Campi

June 1, 2024

Abstract

In this project, information about toothpastes was requested from the Notino site, including product names, brands, prices, URLs, and images. Python, Beautiful Soup, and Selenium were used for web scraping and data extraction. The data found were saved in a CSV.

1. Project Structure

The project is presented as follows (in blue the new ones):

- Folder - Scrapers
  - Folder - abstract
    - File - abstract_scraper.py
  - Folder - notino
    - File - get_counties.py
    - File - scraper.py
    - File - transformation.py
- File - info.json
- File - notino_raw.csv
- File - notino_transformed.csv
- File - scraping.log
- File - small.json

1.1. abstract_scraper.py

The `AbstractScraper` class is designed for web scraping tasks. The class offers a framework for creating web scrapers by managing HTTP requests and logging activities. Below is a detailed explanation of the functionality and methods provided in the script.

Class: `AbstractScraper`

Attributes

- `retailer`: The name of the retailer being scraped.
- `country`: The country from which the scraping is being performed.
- `logger`: A logger instance for logging messages.

Constructor

```
def __init__(self, retailer, country):
    self.retailer = retailer
    self.country = country
    self.logger = logging.getLogger(__name__)
```

Methods

`get(url, headers=None)`

- Sends a GET request to the specified URL.
- Optionally accepts HTTP headers.
- Uses the `requests.Session` to maintain a session.
- Returns the response object.

```
def get(url, headers=None):
    session = requests.Session()
    if headers:
        session.headers.update(headers)
    response = session.get(url)
    return response
```

`send_post_request(self, url, headers=None, cookies=None, json=None)`

- Sends a POST request to the specified URL.
- Accepts optional HTTP headers, cookies, and JSON payload.
- Raises an exception for any HTTP error responses (4xx or 5xx status codes).
- Logs an error message if the request fails.
- Returns the response object if successful, otherwise returns `None`.

```
def send_post_request(self, url, headers=None, cookies=None, json=None):
    try:
        response = requests.post(url, headers=headers,
        cookies=cookies, json=json)
        response.raise_for_status()
        # Raise an exception for 4xx or 5xx status codes
        return response
    except requests.RequestException as e:
        self.logger.error(f"POST request to {url} failed: {e}")
        return None
```

1.1.1. Logging

The logging functionality in this class helps keep track of the scraping process, recording both successful operations and errors. This is useful for debugging and maintaining the scraper.

`setup_logger()`

- Configures the logging settings.
- Logs messages to a file named `scraping.log`.
- Specifies the log format and sets the logging level to INFO.

```
def setup_logger():
    logging.basicConfig(
        filename='scraping.log',
        format='%(asctime)s - %(levelname)s - %(message)s',
        level=logging.INFO
    )
```

`log_info(message)`

- Logs an informational message using the logging module.

```
def log_info(message):
    logging.info(message)
```

`log_error(message)`

- Logs an error message using the logging module.

```
def log_error(message):
    logging.error(message)
```

1.2. get_counties.py

This subsection provides an overview of a Python script designed to scrape information about Notino dealers from the Notino website. The script extracts URLs and language abbreviations for each dealer and saves the information to a JSON file. This data will be useful for subsequent scraping tasks, such as gathering information on toothpastes or other products available from each dealer.

Script Overview

- The script sends a GET request to the Notino website.
- It parses the HTML response using BeautifulSoup.
- It extracts information about each dealer, including the URL and language abbreviation.
- The extracted information is saved to a JSON file named `info.json`.

Dependencies

The script requires the following Python libraries:

- `requests` for sending HTTP requests.
- `BeautifulSoup` from the `bs4` package for parsing HTML.
- `json` for handling JSON operations.

Explanation of the code

Sending a GET Request

```
url = 'https://www.notino.com/'
response = requests.get(url)
soup = BeautifulSoup(response.text, 'html.parser')
```

The script starts by sending a GET request to the Notino website and parsing the HTML response using BeautifulSoup.

Extracting Dealer Information

```
info_list = []

flags_div = soup.find_all('div', class_='flags')

for div in flags_div:
    flag_links = div.find_all('a')
    for link in flag_links:
        url = link.get('href')
        hreflang = link.get('hreflang')
        flag_name = link.find('span', class_='flag__name').text
        info_list.append({"url": url, "abbreviation": hreflang})
```

The script searches for all `div` elements with the class `flags`. It then iterates over each `div` and its `a` tags to extract the URL, language abbreviation, and flag name. This information is stored in a list of dictionaries.

Saving to JSON File

```
with open('info.json', 'w') as f:
    json.dump(info_list, f, indent=4)
```

Finally, the extracted information is saved to a JSON file named `info.json` in a human-readable format with indentation.

1.3. scraper.py

This subsection provides an overview of a Python script designed to scrape product information from the Notino website for various countries. The script uses Selenium for browser automation and BeautifulSoup for HTML parsing. It also utilizes the Google Translator API for translating keywords.

Script Overview

The script performs the following tasks:

- Sends requests to the Notino website for various countries.
- Parses the HTML response using BeautifulSoup.
- Extracts product information including product name, brand, description, price, currency, URL, image, promotion, and discount code.
- Handles pagination to scrape all products.
- Saves the scraped data into a CSV file.

Dependencies

The script requires the following Python libraries:

- `pandas` for data manipulation and storage.
- `requests` for sending HTTP requests.
- `BeautifulSoup` from the `bs4` package for parsing HTML.
- `selenium` for browser automation.
- `deep_translator` for translating keywords.
- `sys`, `time`, `datetime`, and `json` for various system operations.

Explanation of the code

Class Definition

```
class NotinoScraper(AbstractScraper):

    def __init__(self, retailer, country):
        super().__init__(retailer, country)
        self.base_url = f"{country}"
        self.session = requests.Session()
        self.session.headers.update({
            "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
            AppleWebKit/537.36 (KHTML, like Gecko)
            Chrome/125.0.0.0 Safari/537.36",
            "Accept-Language": "it-IT,it;q=0.9,en-US;q=0.8,en;q=0.7",
            "Referer": "https://www.google.com/"
        })
```
This initializes the `NotinoScraper` class, setting up the base URL and HTTP session headers.

Selenium Scraper

```
    def scrape_with_selenium(self, transl):
      [...]
    return products_df
```
This method initializes the Selenium WebDriver, navigates to the target URL, accepts cookies, scrapes product data, handles pagination, and returns a DataFrame containing the scraped data.

Translation Function

```
def all_countries(countries):
    translated= {}
    to_translate = 'dentifrici'
    for country in countries:
        translation = (GoogleTranslator(source='auto',
        target=country["abbreviation"]).translate(to_translate))
        translation = translation.replace(" ", "-")
        translated[country["url"]] = translation
    return translated
```
This function translates the word 'dentifrici' to various languages specified in the input JSON file and returns a dictionary mapping URLs to their respective translations.

Main Function

```
def main(retailer: str) -> pd.DataFrame:
    [..]
    return products_df

if __name__ == "__main__":
    df_raw = main(retailer="notino")
    df_raw.to_csv("notino_raw.csv", index=False)
    AbstractScraper.log_info("Saved on CSV")
```
This is the main function that sets up logging, loads the country data, performs translations, and initializes the scraping process for each country. The results are saved to a CSV file.

1.4. transformation.py

This subsection provides an overview of a Python script designed to transform raw product data scraped from the Notino website. The script filters out invalid data and calculates the discounted price for valid entries.

Script Overview

The script performs the following tasks:

- Initializes the transformation class with country and retailer information.
- Filters out rows with invalid price data.
- Converts price and promotional discount data into numerical format.
- Calculates the price after applying the promotional discount.
- Saves the transformed data into a CSV file.

Dependencies

The script requires the following Python libraries:

- `pandas` for data manipulation and storage.
- `datetime` for handling date and time.

Explanation of the code

Class Definition

```
class NotinoTransformation:
    def __init__(self, country: str, retailer: str):
        self.country = country
        self.retailer = retailer
```

This initializes the `NotinoTransformation` class with the country and retailer information.

Data Transformation Method

```
    def transform_data(self, raw_df: pd.DataFrame) -> pd.DataFrame:
        [...]
    return transformed_df
```

This method performs the data transformation:

- Filters out rows where the 'Price' column has the value "not_available".
- Converts the 'Price' column from string to float after replacing commas with dots.
- Converts the 'Promo' column from string to float after removing '%' and '-' characters.
- Calculates the 'Price_After_Discount' column as the price after applying the promotional discount.

Main Function

```
def main(raw_df: pd.DataFrame, country: str, retailer: str):
    transformation = NotinoTransformation(country=country, retailer=retailer)
    transformed_df = transformation.transform_data(raw_df)
    return transformed_df

if __name__ == "__main__":
    raw_df = pd.read_csv("notino_raw.csv")
    transformed_df = main(raw_df, country="it", retailer="notino")
    transformed_df.to_csv("notino_transformed.csv", index=False)
```

This is the main function that:

- Reads the raw data from a CSV file.
- Initializes the `NotinoTransformation` class with the specified country and retailer.
- Applies the transformation method to the raw data.
- Saves the transformed data to a new CSV file.

1.5. info.json

Json resulting from the get_countries script

1.6. notino_raw.csv

CSV in raw state after scraping

1.7. notino_transformed.csv

CSV transformed after applying the required changes and implemented in transformation.py

1.8. scraping.log

Log file initialised and written in the scraping phase

1.9. small.json

Same info.json file only with one country to test the correct functioning of all functions

2. CSV structure

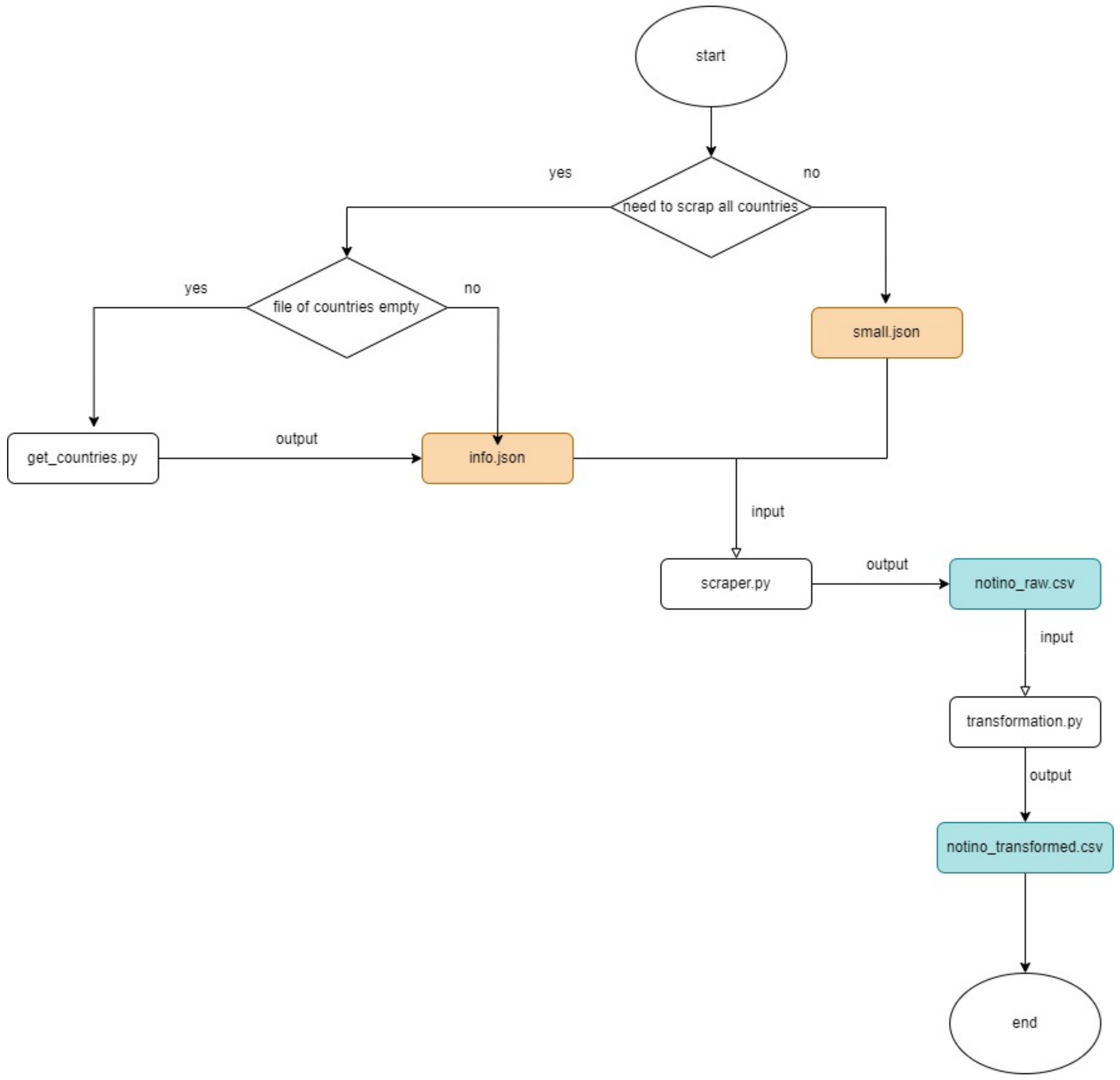
2.1. Raw CSV

Data	**Description**
Product Name	Name of the product
Brand	Brand
Description	Description of the product
Price	Price
Currency	Currency
URL	Product website URL
Image	Product image
Country	Country
Promo	Discount value
Code	Code to use at checkout to get the discount
Scraped At	Date and time of data collection

2.2. Transformed CSV

Column Name	**Description**
Product Name	The name of the product.
Brand	The brand of the product.
Description	A brief description of the product.
Price	The price of the product before any discounts, formatted as a float.
Currency	The currency in which the price is listed.
URL	The URL of the product page on the Notino website.
Image	The URL of the product image.
Country	The country from which the data was scraped.
Promo	The promotional discount percentage, formatted as a float.
Code	The discount code, if any.
Scraped At	The timestamp when the data was scraped.
Price After Discount	The price of the product after applying the promotional discount, calculated as a float.

3. Program flow



Explanation of the flow If we need to run the scraper on all the Notino countries to find the required product information, we should check if the country JSON is empty. If it is empty, we need to run get_countries to fill it. If the countries file is already populated, we have the option to run the scraper on all countries or just a specific selection by using small.json (which may contain only one country) instead of info.json (which contains all countries). It is possible to implement the choice of countries to visit directly in the info.json file using code (this could be a future improvement). Once the countries have been selected, we can run scraper.py, which will generate the file notino_raw.csv. This file will then be appropriately transformed using transformation.py to produce another CSV called notino_transformed.csv.

Clarification on searching for the required product. I have successfully developed a translator that accurately translates the word "toothpaste" into multiple languages. However, there are some instances where the word is entered in different forms across languages, leading to issues with certain pages not working and not being scraped.

These issues are meticulously logged, detailing the site (i.e. country) and the translation for which the 'url/word' access failed to produce results.

As a future enhancement, I propose implementing a solution to search for the word, even if in the "wrong" form, in the website's search bar if direct URL access fails, and then capture the first result. Manual tests have demonstrated that this approach yields positive outcomes.