1.3_scrape_comments

April 4, 2024

0.1 1.3_scrape_comments (119 lines in total)

Automates the extraction of user comments and related data from movie comment sections on Douban, a Chinese social networking service which provides user review and recommendation services for movies, among other things. Here's a breakdown of its components and workflow:

1. **Library Imports**: The script uses Selenium for browser automation to navigate pages, Lxml for parsing HTML content, Pandas for data manipulation and CSV file operations, Requests for making HTTP requests, and standard libraries such as time, re, random, and os for various utility functions.
2. **get_data_and_write_csv Function**: This is the main function that reads a filtered list of movies from a CSV file, iterates through their comment links, and scrapes comments from multiple pages and rating levels (high, medium, low). It saves the comments into individual CSV files for each movie.
    - **Data Extraction**: The script navigates to each page of comments for the movie and extracts user comments, ratings, and timestamps using XPath queries. It also makes additional requests to user profile pages to extract detailed information such as the user's city, IP location, and registration date.
    - **Error Handling and Delays**: The script includes try-except blocks to handle errors and continues the scraping process even if some pages or data fail to load. It also uses random sleep intervals to mimic human browsing patterns and reduce the risk of being blocked by the server.
    - **CSV File Writing**: The extracted data is compiled into Pandas DataFrames and appended to a CSV file corresponding to each movie. The script checks if the file exists and removes it before starting a new scrape to ensure data is not duplicated.
3. **Selenium WebDriver**: The script initializes a Selenium WebDriver for each URL to handle dynamic content loaded by JavaScript, which might not be accessible via static HTML requests. However, there seems to be a mixed usage of Selenium and Requests, which might be redundant or intended for different scenarios (Selenium might be a leftover from template code as it's not used effectively in the provided context).
    - Reference: https://github.com/Hualalala741/douban-review-analysis
4. **Configuration Variables**: The script defines a proxy pool, request headers, and cookies for making HTTP requests. These are crucial for accessing Douban, which might have regional restrictions or require user authentication via cookies.

5. **Execution**: The script ends with a call to the `get_data_and_write_csv` function, passing in the name of the CSV file containing the filtered movie list, headers, cookies, and proxy information. This initiates the scraping process.

Overall, the script is structured to handle large-scale data extraction from a dynamic website, dealing with pagination, different user rating levels, and extracting detailed user comment data. It is designed to be robust against common web scraping challenges such as request rate limits and IP bans. However, users should be aware of Douban's terms of service and legal considerations regarding web scraping and data privacy.

[ ]:
```python
# Import necessary libraries
from selenium import webdriver
from lxml import etree
import requests
import pandas as pd
import time
import re
import random
import os
```

[ ]:
```python
def get_data_and_write_csv(filtered_movie_list_csv, header, cookie, proxy_pool):
    """
    Fetch data from movie comment links and write it into individual CSV files.

    :input
        filtered_movie_list_csv: CSV file containing filtered movie list.
        header: Headers used for making HTTP requests.
        cookie: Cookies used for making HTTP requests.
        proxy_pool: List of proxies used for making HTTP requests.
    """
    # Load filtered movie list from CSV
    filtered_movies = pd.read_csv(filtered_movie_list_csv)
    # Create a dictionary mapping movie names to their comment links
    movie_links = filtered_movies.set_index('movie_name')['comment_link'].
↪to_dict()

    # Iterate through each movie and its comment link
    for moviename, link in movie_links.items():
        file_name = moviename + '.csv' # Define file name for saving comments
        # Remove existing file to avoid appending to old data
        if os.path.exists(file_name):
            os.remove(file_name)

        # Iterate through comment types and pages
        for k in ['h', 'm', 'l']: # High, medium, and low ratings
            for i in range(3):
                try:
                    # Construct URL for the comment page
```

```
                    url = link + '?
↪percent_type={}&start={}&limit=200&status=P&sort=new_score'.format(k, i *␣
↪200)
                    # Initialize WebDriver and navigate to the URL
                    driver= webdriver.Chrome()
                    driver.get(url)
                    # Perform HTTP request for the URL
                    rq = requests.get(url, headers=header, timeout=30,␣
↪stream=True,
                                      proxies=random.choice(proxy_pool),␣
↪cookies=cookie)
                    print('Get type {} page{}'.format(k, i + 1), rq.status_code)
                    # Parse the page content
                    html_1 = etree.HTML(rq.content)
                    # Extract required information using XPaths
                    name = html_1.xpath('//*[@class="comment-info"]/a/text()')
                    u_links = html_1.xpath('//*[@class="comment-info"]/a/@href')
                    star_time = html_1.xpath('//*[@class="comment-info"]/span/
↪@title')
                    comment = html_1.xpath('//*[@class="comment"]/p/span/
↪text()')

                    # Prepare data for CSV
                    data = []
                    for j in range(len(u_links)):
                        try:
                            # Fetch additional user information from their␣
↪profile
                            u_link = u_links[j]
                            rqqq = requests.get(u_link, headers=header,␣
↪timeout=30, stream=True,
                                                proxies=random.
↪choice(proxy_pool), cookies=cookie)
                            html = etree.HTML(rqqq.content)
                            # Extract user location and registration information
                            state = html.xpath('//*[@class="basic-info"]/
↪text()')
                            if not state:
                                continue
                            city = html.xpath('//*[@class="user-info"]/a/
↪text()')
                            ip_loc = html.xpath('//*[@class="ip-location"]/
↪text()')
                            # Compile all extracted information into a␣
↪dictionary
                            item = {'city': city[0] if city else None,
```

```
                                    'ip': ip_loc[0][5:] if ip_loc else None,
                                    'reg_date': re.sub(' ', '',
                                                       html.xpath('//
↪*[@class="user-info"]/div/text()')[1]).strip(),
                                    'name': name[j],
                                    'star': star_time[2 * j],
                                    'v_time': star_time[2 * j + 1],
                                    'comment': comment[j].replace('\n', ' '),
                                    'level': k,
                                    'movie': moviename}
                            data.append(item)
                        except Exception as e:
                            print(f"Error fetching user page data for␣
↪{u_links[j]}: {e}")
                        # Random short delay to prevent being blocked
                        time.sleep(random.uniform(0.01, 0.1))
                    # Append the data of the current page to the CSV file
                    df = pd.DataFrame(data)
                    # Write header only if it's the first page
                    df.to_csv(file_name, mode='a', index=False, header=not i,␣
↪encoding="utf_8_sig")
                except Exception as e:
                    print(f"Error on page {i+1}: {e}")
                    # Ensure the WebDriver is closed after each iteration
                finally:
                    driver.quit()
```

[ ]:
```
# Define proxy pool, request headers, and cookies for HTTP requests
proxy_pool = [
    {'http': 'http://123.169.118.8:9999'},
    {'http': 'http://175.43.154.137:9999'},
    {'http': 'http://117.91.165.126:9999'},
    {'http': 'http://113.124.86.125:9999'}
]
header = {
    'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/
↪537.36 (KHTML, like Gecko) Chrome/121.0.0.0 Safari/537.36'
    }
cookie = {
```

```
    'cookie':'bid=95t-fKTsR5g; dbcl2="164819550:k8VTBoBPKLY"; ck=Eu4t;␣
 ↪__utma=30149280.1010685396.1708218433.1708218433.1708218433.1;␣
 ↪__utmb=30149280.0.10.1708218433; __utmc=30149280; __utmz=30149280.1708218433.
 ↪1.1.utmcsr=accounts.douban.com|utmccn=(referral)|utmcmd=referral|utmcct=/;␣
 ↪__utma=223695111.1998001554.1708218433.1708218433.1708218433.1;␣
 ↪__utmb=223695111.0.10.1708218433; __utmc=223695111; __utmz=223695111.
 ↪1708218433.1.1.utmcsr=accounts.douban.
 ↪com|utmccn=(referral)|utmcmd=referral|utmcct=/; _pk_ref.100001.
 ↪4cf6=%5B%22%22%2C%22%2C1708218433%2C%22https%3A%2F%2Faccounts.douban.
 ↪com%2F%22%5D; _pk_id.100001.4cf6=28b3f561e3882fc7.1708218433.; _pk_ses.
 ↪100001.4cf6=1; __gads=ID=aca2b1e4a96a0b7b:T=1708218433:RT=1708218433:
 ↪S=ALNI_MbBSoEUuut8_TAEvU9VouUMDSby7w; __gpi=UID=00000dc1fb3c222a:
 ↪T=1708218433:RT=1708218433:S=ALNI_MaUURRnLRs9BIkSwuHRUJN7aSjzIQ;␣
 ↪__eoi=ID=420df50a1411b3bf:T=1708218433:RT=1708218433:
 ↪S=AA-Afjb7NjqE5nNCCkQH_Ft3EOAK;␣
 ↪FCNEC=%5B%5B%22AKsRol8IeQEu3-h76sqZ_LITKNVBh0XogL6kb9XrQBPgwa9oFozSZIUPPzbRl2BeDn9j0M1lqfkA
    }

# Call the function to start scraping and writing data
get_data_and_write_csv('filtered_movielist.csv', header, cookie, proxy_pool)
```