1.1_crawl_movie_list

April 4, 2024

0.1 1.1_crawl_movie_list (181 lines in total)

Extract detailed information about movies listed on a web page, including their names, genres, ratings, and other relevant details, and then save this information into a CSV file. The script performs the following main functions:

1. **crawl_page**: Extracts and returns a list of movie URLs from a BeautifulSoup object, which parses the HTML content of a web page. This function targets 'div' elements with the class 'bd doulist-subject' to find the relevant URLs.

2. **crawl_movie**: Extracts details such as the movie's name, year, genres, ratings, number of people who rated, series status, and comment link from a BeautifulSoup object and appends this information to a provided list of movie details.

3. **request_and_soup**: Sends an HTTP GET request to a specified URL using optional headers, proxies, and cookies, then parses and returns the content as a BeautifulSoup object. It includes error handling such as timeouts and employs a random delay between requests to avoid being banned by the server for too many requests.

4. **get_next_page_link**: Extracts and returns the URL of the next page from the parsed HTML, enabling pagination through a list of movies.

5. **save_to_csv**: Saves the extracted movie data, which is structured as a list of dictionaries, into a CSV file with specified fieldnames.

The script appears to be a comprehensive solution for web scraping, designed to navigate through paginated lists of movies, extract detailed information from each page, and compile the data into an easily accessible format. It uses common Python libraries such as `requests` for web requests, `BeautifulSoup` from `bs4` for parsing HTML, and Python's built-in `csv` module for writing the data to a CSV file. Additionally, it employs random delays and proxies to minimize the risk of being blocked by the server for scraping activities.

[ ]:
```python
# Required libraries
import requests
from bs4 import BeautifulSoup
import queue
import csv
import util
import time
import random
```

```
[ ]: def crawl_page(soup):
         """
         Extract and return a list of URLs from the specified BeautifulSoup object.

         :input
             soup: BeautifulSoup object containing the HTML of a webpage.
         :return:
             List of URL strings extracted from the 'div' elements with class
             'bd doulist-subject'.
         """
         urls = []
         # Extract URLs from each specified 'div' element within the page
         for block in soup.find_all('div',  class_ = 'bd doulist-subject'):
             url = block.find('a', href = True)
             urls.append(url['href'])

         return urls

     def crawl_movie(url, soup, movies):
         """
         Extracts details of a movie from a BeautifulSoup object and appends it to
             the provided list.

         :input
             url: String of the base URL of the movie page.
             soup: BeautifulSoup object containing the HTML of the movie page.
             movies: List of dictionaries, where each dictionary contains
                 details of a movie.
         :return: The updated list of movie dictionaries with the new movie
             details added.
         """
         genre = []
         # Extract movie name, year, genres, ratings, and other details
         name_element = soup.find('span', attrs = {'property': 'v:itemreviewed'})
         name = name_element.get_text() if name_element else "N/A"

         year_element = soup.find('span', class_ = 'year')
         year = year_element.get_text() if year_element else "N/A"

         for tag in soup.find_all('span', attrs = {'property': 'v:genre'}):
             genre.append(tag.get_text())

         rating_element = soup.find('strong', class_='rating_num')
         rating = rating_element.get_text() if rating_element else "N/A"

         rate_people_element = soup.find('span', attrs={'property': 'v:votes'})
```

```
    rate_people = rate_people_element.get_text() if rate_people_element else "N/
↪A"

    comment_link_element = soup.find('a', href=True, class_='rating_people')
    comment_link = url + comment_link_element['href'] if comment_link_element␣
↪else "N/A"

    # Check if the movie is part of a series
    series_tag = soup.find('span', class_='pl',
                           string=lambda text: text and " " in text)
    if_series = 1 if series_tag else 0

    # Add extracted details to the list
    movies.append({'movie_name': name, 'year': year[1:5], 'genre': genre,
    'rating': rating, 'rate_num': rate_people, 'if_series': if_series,
    'comment_link': comment_link})

    return movies

def request_and_soup(url, headers, proxy_pool, cookies):
    """
    Send a request to the specified URL and return a BeautifulSoup object of
    the content.

    :input
        url: String representing the URL to fetch.
        headers: Dictionary of HTTP headers to send with the request.
        proxy_pool: List of proxy addresses to choose from.
        cookies: Dictionary of cookies to send with the request.
    :return:
        BeautifulSoup object parsed from the content of the response.
    """
    # Make the request using the specified proxies and headers
    response = requests.get(url, headers = headers,timeout=30,stream=True,
                            proxies=random.choice(proxy_pool),cookies=cookies)
    # Log the status for debugging
    print(url, "Status", response.status_code)
    # Random delay to avoid being banned by the server
    time.sleep(random.randint(1,5))

    # Return the parsed HTML of the page
    return BeautifulSoup(response.content, 'html.parser')

def get_next_page_link(soup):
    """
    Extract and return the link to the next page from the BeautifulSoup object.
    :input
        soup: BeautifulSoup object containing the HTML of a webpage.
    :return:
        String containing the URL of the next page or None if there is no
        next page link.
    """
    # Find the link to the next page, if it exists
    next_page_tag = soup.find('link', attrs={'rel': 'next'})
    if next_page_tag and 'href' in next_page_tag.attrs:
        return next_page_tag['href']

    return None

def save_to_csv(data, filename, fieldnames):
    """
    Save a list of dictionaries to a CSV file with specified fieldnames.

    :input
        data: List of dictionaries, each representing a row to be written to␣
 ↪the CSV.
        filename: String representing the name of the file to save the data to.
        fieldnames: List of strings representing the fieldnames for the CSV.
    :return:
        None.
    """
    # Open the file and write data
    with open(filename, mode='w', newline='', encoding='utf-8') as file:
        writer = csv.DictWriter(file, fieldnames=fieldnames)
        writer.writeheader()
        for element in data:
            writer.writerow(element)
    # Log the save operation
    print(f"Data saved to {filename}")
```

```
[ ]: # Main execution starts here
     # Initial setup: URLs, domain restrictions, headers, proxies, and cookies
     starting_url = ("https://www.douban.com/doulist/111904815/?start=0"
                     "&sort=time&playable=0&sub_type=")
     limiting_domain = "movie.douban.com"
     headers = {"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'}
     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'}]
```

```
cookie = 'viewed="2154713"; bid=zUvd30hDt8Q;␣
 ↪gr_user_id=9a4d2f8b-30aa-4787-9693-23d0170f15d3;␣
 ↪__gads=ID=bcb2ae53b49d2098-22de3892b9d600ac:T=1664375231:RT=1664375231:
 ↪S=ALNI_MYPz49Jz3I9IUfytEldfpeMD-esqw; ll="118254"; dbcl2="247613379:
 ↪nSzs9qgkAZ4"; push_noty_num=0; push_doumail_num=0; __utmv=30149280.24761;␣
 ↪_vwo_uuid_v2=D35CB1279D5898733AA7D9BEE02EF984A|fe2974e096a3c9086033df436020d9ab;
 ↪ __yadk_uid=4Qgm1oPG3VvgkLUUtxL9RyjZli9KEK3S; ck=jtTn; ap_v=0,6.0;␣
 ↪__utmc=30149280; __utmz=30149280.1664496568.4.4.
 ↪utmcsr=google|utmccn=(organic)|utmcmd=organic|utmctr=(not provided);␣
 ↪__utmc=223695111; __utmz=223695111.1664496568.3.3.
 ↪utmcsr=google|utmccn=(organic)|utmcmd=organic|utmctr=(not provided);␣
 ↪__gpi=UID=000009fe6062d11e:T=1664375231:RT=1664496575:
 ↪S=ALNI_MYCw-G5X6haik3yJ-WRYu3xD3g_3g; __utma=30149280.1572677335.1664375233.
 ↪1664496568.1664498496.5; __utmb=30149280.0.10.1664498496; __utma=223695111.
 ↪186291436.1664455437.1664496568.1664498496.4; __utmb=223695111.0.10.
 ↪1664498496; _pk_ref.100001.4cf6=["","",1664498496,"https://www.google.com/"];
 ↪ _pk_ses.100001.4cf6=*; _pk_id.100001.4cf6=4e77779d18308ecb.1664455437.4.
 ↪1664498523.1664496615.'
cookies = {'cookie': cookie}

# Queue for URLs to be crawled and set for visited URLs
url_queue = queue.Queue()
url_queue.put(starting_url)
visited_urls = set()
movies = []

# Main crawling loop
while not url_queue.empty() and len(visited_urls)<40:
    current_url = url_queue.get()
    if current_url not in visited_urls:
        visited_urls.add(current_url)
        soup = request_and_soup(current_url, headers, proxy_pool, cookies)
        movie_urls = crawl_page(soup)
        for movie_url in movie_urls:
            if util.is_url_ok_to_follow(movie_url, limiting_domain):
                movie_soup = request_and_soup(movie_url, headers, proxy_pool,␣
 ↪cookies)
                movies = crawl_movie(movie_url, movie_soup, movies)
        next_page_url = get_next_page_link(soup)
        if next_page_url and next_page_url not in visited_urls:
            url_queue.put(next_page_url)
        # Log the next URL to be crawled for debugging
        print(f"Next URL: {next_page_url}")

# Save the results to a CSV file
save_to_csv(movies, 'movies.csv', fieldnames = ['movie_name', 'year',
    'genre', 'rating', 'rate_num', 'if_series', 'comment_link'])
```

```
https://www.douban.com/doulist/111904815/?start=0&sort=time&playable=0&sub_type=
Status 200
https://movie.douban.com/subject/35168544/ Status 200
https://movie.douban.com/subject/35682539/ Status 200
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[4], line 29
     27 for movie_url in movie_urls:
     28     if util.is_url_ok_to_follow(movie_url, limiting_domain):
---> 29         movie_soup = request_and_soup(movie_url, headers, proxy_pool,␣
  ↪cookies)
     30         movies = crawl_movie(movie_url, movie_soup, movies)
     31 next_page_url = get_next_page_link(soup)

Cell In[3], line 83, in request_and_soup(url, headers, proxy_pool, cookies)
     81 print(url, "Status", response.status_code)
     82 # Random delay to avoid being banned by the server
---> 83 time.sleep(random.randint(1,5))
     85 # Return the parsed HTML of the page
     86 return BeautifulSoup(response.content, 'html.parser')

KeyboardInterrupt:
```