jupyter scraping_dataset Last Checkpoint: 28 seconds ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

[1]:
```python
import requests # use it for send request to page url that i want scrapping data from
from bs4 import BeautifulSoup  # use it in web scraping process 'parsing HTML content'
import pandas as pd
import time # add delays between requests to avoid blocke
import random  # use it to add delays between requests "to not blocked from site"
from fake_useragent import UserAgent  # used to to generate random user agents for each request .. same user agent get 503 can send req only one
import re  # used for regular expressions -  some data shoud extracted from specif expressions
```

[2]:
```python
# get the soup object "to parse" from the URL
def get_soup(url, headers):
    for _ in range(5):  # Retry up to 5 times if the request fails
        try:
            response = requests.get(url, headers=headers)  # sedn GET HTTP request to the URL
            response.raise_for_status()  # raise an HTTP Error for bad responses 400 and 500 "error has occurred during the process"
            return BeautifulSoup(response.content, 'html.parser')  # parse HTML content to use it and extract data
        except requests.RequestException as e:
            # print(f"Failed to retrieve page {url}, error: {e}")  # error message
            time.sleep(random.randint(1, 5))  # Wwit for a random time between 1 to 5 seconds before retrying
    return None  # Return None if the request fails after 5 retries
```

content of page that send get HTTP request to get it

parsed HTML Content

[3]:
```python
def parse_product(product, page_number,category):
    def safe_find(element, search_dict, text=False):
        try:
            found = element.find(**search_dict)  # search for the element using provided criteria "clas , attribute"
            return found.text.strip() if (found and text) else found  # text if text=True, else return element itself
        except AttributeError:
            return None  # if element not found

    # extract product name ( product to search in ,  criteria to search based on , test =True)
    name = safe_find(product, {'name': 'span', 'attrs': {'class': 'a-size-medium a-color-base a-text-normal'}}, text=True)
    if not name:
        name = safe_find(product, {'name': 'div', 'attrs': {'data-cy': 'title-recipe'}}, text=True)

    # extract product price
    price = safe_find(product, {'name': 'span', 'attrs': {'class': 'a-offscreen'}}, text=True)
    if not price:
        price = safe_find(product, {'name': 'div', 'attrs': {'class': 'a-row a-size-base a-color-secondary'}}, text=True)

    # extract src of product image
    image_element = safe_find(product, {'name': 'img', 'attrs': {'class': 's-image'}})
    image = image_element['src'] if image_element else None

    # extract product rating
    rating_text = safe_find(product, {'name': 'span', 'attrs': {'class': 'a-icon-alt'}}, text=True)
    rating = rating_text.split()[0] if rating_text else None

    # extract rating count
    try:
        rating_count_element = product.find("div", {"class": "s-csa-instrumentation-wrapper"}).find("span", {"aria-label": True})
        rating_count_text = rating_count_element.text.strip() if rating_count_element else None
        rating_count = re.sub(r'[^0-9]', '', rating_count_text) if rating_count_text else None
    except AttributeError:
        rating_count = None

    # extract delivery information
    try:
        delivery_element = product.find("div", {"data-cy": "delivery-recipe"})
        delivery = delivery_element.find("span", {"aria-label": True}).text.strip() if delivery_element else None
    except AttributeError:
        delivery = safe_find(product, {'name': 'span', 'attrs': {'aria-label': True}}, text=True)

    # is the product is a "Best Seller"
    best_seller_element = safe_find(product, {'name': 'span', 'attrs': {'class': 'a-badge-text'}}, text=True)
    is_best_seller = 1 if best_seller_element and "Best Seller" in best_seller_element else 0

    # check if the product is an "Overall Pick" ..  Products highlighted as 'Overall Pick' are: Rated 4+ stars , Purchased often ,Returned infrequently
    overall_pick_element = safe_find(product, {'name': 'span', 'attrs': {'class': 'a-badge-text', 'data-a-badge-color': 'sx-cloud'}}, text=True)
    is_overall_pick = 1 if overall_pick_element and "Overall Pick" in overall_pick_element else 0

    # Return product details
    return {
        'page': page_number,
        'name': name,
        'category': category,
        'image': image,
        'price': price,
        'rating': rating,
        'rating_count': rating_count,
        'delivery': delivery,
        'is_best_seller': is_best_seller,
        'is_overall_pick': is_overall_pick
    }
```

conditions / keys to from where read this data

[4]:
```python
#scrape each page "url"
def scrape_page(url, headers, page_number, category):
    soup = get_soup(url, headers)  # html to parse it and extract data
    if not soup:
        return []
```

return parsed html to exratct data from

```
    # extract product divs that contain the required data-component-type attribute
    product_divs = soup.find_all('div', {"data-component-type": "s-search-result"})
    # Print the product divs to inspect their structure
    #for index, product_div in enumerate(product_divs):
    #    print(f"Product div {index}:\n", product_div.prettify(), "\n *************************\n") #get what this piece of code does

    # parse each product in the product divs
    products = [parse_product(product, page_number, category) for product in product_divs]

    return [product for product in products if product]  # filter None values
```

condition: div tag should contain this key=value to return

for each div "contain all data for one product"
parse it to get previous att's

```
[5]: ua = UserAgent()  #  UserAgent for random headers - one for each req

     headers = {
         "accept-language": "en-US,en;q=0.9",  # accept-language header
         "accept-encoding": "gzip, deflate, br",  #accept-encoding header
         "User-Agent": ua.random,  # random User-Agent for each request
         "accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7",
         "Connection": "close"  # close connection after each request to avoid block
     }

     categories = ['electronics', 'toys', 'mens', 'womens', 'foods', 'clothes', 'printers', 'flowers', 'accessories']
     all_products = []

     # for each category
     for category in categories:
         page_number = 1  # reset page number for each category

         # 10 pages for each category
         while page_number < 10:  # cant do this to be 404 becouse if page number not found display last avaliable so no condition to stop loop and huge data
             headers['User-Agent'] = ua.random  # new random User-Agent for each requ
             url = f'https://www.amazon.com/s?k={category}&page={page_number}&_encoding=UTF8&content-id=amzn1.sym.ce070039-db53-47a0-8017-250744e811c9&pd_rd_r=e
             products = scrape_page(url, headers, page_number, category)  # get product data

             if not products:
                 break

             all_products.extend(products)  # add the products to the list
             page_number += 1  # next page "iteration"

             # delay to avoid block by site
             time.sleep(random.randint(1, 5))

     # convert all products to a DataFrame
     df = pd.DataFrame(all_products)
     df.index += 1  # index from 1
     df.to_csv('amazon_products_final.csv', index_label='index')  # Save the DataFrame to a csv file
```

dynamic URL , based on category and page number

```
[6]: data = pd.read_csv('amazon_products_final.csv')
     data.info() #info about scrapped data

     <class 'pandas.core.frame.DataFrame'>
     RangeIndex: 2015 entries, 0 to 2014
     Data columns (total 11 columns):
      #   Column           Non-Null Count  Dtype
     ---  ------           --------------  -----
      0   index            2015 non-null   int64
      1   page             2015 non-null   int64
      2   name             2015 non-null   object
      3   category         2015 non-null   object
      4   image            2015 non-null   object
      5   price            1950 non-null   object
      6   rating           1849 non-null   float64
      7   rating_count     1782 non-null   float64
      8   delivery         1782 non-null   object
      9   is_best_seller   2015 non-null   int64
      10  is_overall_pick  2015 non-null   int64
     dtypes: float64(2), int64(4), object(5)
     memory usage: 173.3+ KB

[ ]:
```