ASSIGNMENT

GRAB-FOOD SCRAPER

Objective:

The objective of this assignment is to develop a web scraper to extract specific information from an app/website. This assignment will test your ability to understand the app/web scraping concepts, make use of scraping libraries, and counter challenges such as blocking, authentication, etc., in the data extraction process.

Problem Statement: Develop the scraper to scrape Grab Food Delivery.

Approach:

1. Scraping Individual Restaurant Details: Utilize BeautifulSoup library in Python to scrape details of individual restaurants from their respective URLs.

2. Iterative Pagination: Implement a function to navigate through multiple pages of the website, extracting links to each restaurant's page.

3. Data Structuring: Structure the scraped data into a format such as dictionaries or lists for easy manipulation and storage.

Problem 1:

Challenge: Difficulty in extracting specific elements from restaurant pages due to varying HTML structures or dynamic content loading.

Solution/Approach:

- Inspect the HTML structure of the restaurant pages and identify unique identifiers (e.g., class names, tags) for the desired information.
- Implement robust error handling mechanisms to handle cases where the desired elements are not found or the page structure changes.
- Utilize CSS selectors or XPath expressions for more precise element selection, if necessary.

Problem 2:

Challenge: Efficiently handling pagination to scrape data from multiple pages without overloading the server or missing any restaurant listings.

Solution/Approach:

- Determine the pagination mechanism used by the website (e.g., page numbers, "next" buttons).
- Implement a loop or recursive function to iterate through each page, extracting restaurant links.
- Incorporate rate-limiting or delay mechanisms between requests to prevent overwhelming the server with too many requests in a short period.
- Monitor for any anti-scraping measures employed by the website (e.g., CAPTCHA) and adjust the scraping strategy accordingly.

By addressing these problems with the outlined solutions, the web scraping process can be made more robust and reliable, allowing for the successful extraction of restaurant data from the target website.

Stats of data should at least cover the total count as per the selected location, as well as not null and null stats of above mentioned mandatory fields.

To gather statistics about the scraped restaurant data, including total count, counts of non-null and null values for mandatory fields like name, address, and rating, you can follow these steps:

Scraping and Data Collection: Implement the scraping code to collect restaurant data, including name, address, and rating, from the website.

Data Structuring: Store the scraped data in a suitable data structure, such as a list of dictionaries, where each dictionary represents a restaurant with its details.

Statistics Calculation: Calculate the desired statistics from the collected data.

Quality Control (QC) is crucial for ensuring the accuracy and reliability of extracted data from the GrabFood website. Here are some QC measures that can be performed:

1. Data Validation: Check if the extracted restaurant names, cuisines, ratings, and delivery times match the corresponding information displayed on the website. This involves comparing a sample of extracted data against the website to identify any discrepancies or inaccuracies.

2. Error Handling: Implement error handling mechanisms to deal with potential issues such as connection errors, timeouts, or unexpected HTML structures. This ensures robustness and reliability in data extraction.

3. Duplicate Detection: Check for duplicate entries in the extracted data to prevent redundancy. This can be done by comparing restaurant names or other unique identifiers.

4. Consistency Check: Ensure consistency in data formatting and structure. For example, verify that all ratings are represented in the same format (e.g., out of 5 stars) and that delivery times are consistently formatted (e.g., in minutes).

5. Data Integrity: Validate that the extracted data maintains its integrity throughout the extraction process. This involves verifying that all required fields are present and correctly populated.

6. Cross-Verification: Cross-verify the extracted data with other reliable sources, if available, to confirm its accuracy and completeness.

7. Performance Testing: Assess the performance of the data extraction process by measuring factors such as extraction speed, resource utilization, and scalability. This helps identify any bottlenecks or performance issues that need to be addressed.

By implementing these QC measures, we can ensure that the extracted data is accurate, reliable, and consistent, meeting the requirements of downstream analysis and applications.

Compose a brief report documenting the following:

- **Your overall approach and methodology.**
- **Challenges faced during the scraping process.**
- **Any improvements or optimizations you could envision.**

Your overall approach and methodology:

When approaching web scraping and data extraction from a website like GrabFood, the first step is to analyze the structure of the website and identify the HTML elements containing the desired data. This involves inspecting the webpage using browser developer tools to locate relevant tags, classes, and attributes. Once the target elements are identified, libraries like BeautifulSoup in Python can be employed to parse the HTML content and extract the required data efficiently.

In the case of GrabFood, the primary objective is to gather information about restaurants, including names, cuisines, ratings, and delivery times. To achieve this, we can iterate through the list of restaurants on the main page and extract the necessary details for each restaurant. Since web scraping can be time-consuming, especially for large datasets, employing multithreading or asynchronous processing techniques can significantly enhance performance by enabling concurrent scraping of multiple restaurant listings.

Additionally, it's crucial to handle potential issues gracefully, such as network errors, missing data, or changes in the website's structure. Incorporating error handling mechanisms and regular monitoring of the scraping process can help maintain robustness and reliability. Finally, ensuring compliance with the website's terms of service and legal requirements is essential to conduct ethical and responsible web scraping practices.

Challenges faced during the scraping process

During the scraping process for the GrabFood restaurant dataset, several challenges may arise. One common challenge is the dynamic nature of web pages, where elements like restaurant listings and their associated data can change frequently. This necessitates regular monitoring and adjustment of the scraping code to accommodate such changes.

Additionally, handling anti-scraping mechanisms implemented by the website, such as rate limiting, CAPTCHAs, or IP blocking, poses another challenge. These measures aim to deter automated scraping activities and may require the implementation of strategies like rotating proxies or employing headless browsers to bypass detection.

Furthermore, inconsistencies in HTML structure across different pages or unexpected variations in data formatting can complicate the scraping process, requiring robust error handling and data normalization techniques to ensure accurate extraction of information. Overall, overcoming these challenges demands a combination of technical expertise, adaptability, and perseverance.

Any improvements or optimizations you could envision

To optimize web scraping from the GrabFood website, implementing a more consistent and predictable HTML structure across pages could enhance scraping efficiency. Additionally, providing a dedicated API for accessing restaurant data in a structured format would facilitate easier and more reliable data extraction. Ensuring robust error handling mechanisms and optimizing server responses to prevent HTTP errors like 405 Method Not Allowed would further streamline the scraping process.

Steps to execute the provided code locally:

1. Install Python: If you haven't already, download and install Python from the [official Python website] (https://www.python.org/downloads/). Make sure to select the option to add Python to your system's PATH during installation.

2. Set Up a Virtual Environment (Optional): It's a good practice to work within a virtual environment to manage dependencies. Open a terminal or command prompt and navigate to your project directory.

Activate the virtual environment:

- On Windows:

  venv\Scripts\activate

- On macOS and Linux:

  source venv/bin/activate

3. Install Required Libraries: With your virtual environment activated, install the required libraries (requests and BeautifulSoup) using pip:

```
pip install requests beautifulsoup4
```

4. Modify the Code: Replace the `base_url` variable with the URL of the website you want to scrape. Adjust other parameters such as `num_pages` as needed.

5. Run the Script: Save the provided code in a Python script file (e.g., restaurant_scraper.py). Then, run the script from your terminal or command prompt:

```
python restaurant_scraper.py
```

6. Review Output: The script will execute, scraping restaurant data from the specified website and calculating statistics. It will print the total count as well as the counts of non-null and null values for the mandatory fields (name, address, and rating).

7. Troubleshooting: If you encounter any errors, carefully review the error message and check your code for any mistakes. Common issues include incorrect URLs, missing HTML elements, or network connectivity problems. You may need to adjust the code accordingly.

8. Optional: If you're working with a large amount of data or scraping frequently, consider implementing error handling, logging, and optimization techniques to improve the reliability and efficiency of your scraping process.

By following these steps, we should be able to execute the provided code locally and scrape restaurant data from the target website while calculating relevant statistics.

Code with Multithreading:

```
import requests

from bs4 import BeautifulSoup

import threading

import csv

import pandas as pd


# Define the user-agent header

HEADERS = {'User-Agent': 'Mozilla/5.0 (iPad; CPU OS 12_2 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Mobile/15E148'}

# Define the base URL of the website

BASE_URL = "https://food.grab.com"


# Function to scrape individual restaurant information

def scrape_restaurant_info(url):

    # Print a debug message indicating which restaurant info is being scraped

    print(f"Scraping restaurant info for: {url}")

    # Send a GET request to the restaurant URL

    r = requests.get(BASE_URL + url, headers=HEADERS)

    # Parse the HTML content of the response

    soup = BeautifulSoup(r.content, 'html.parser')

    # Extract relevant information about the restaurant
```

```
    out = True
    try:
        aa = soup.find("h1", {"class": ["name___1Ls94"]}).text
        bb = soup.find("h3", {"class": ["cuisine___3sorn",
"infoRow___3TzCZ"]}).text
        cc = soup.find("div", {"class": ["rating___1ZywF"]}).find('div').text
        dd = soup.find("div", {"class": ["distance___3UWcK"]}).find('div').text
    except:
        out = False
        pass
    ee=
soup.findAll("div",{"class":["category___3C8lX"],"id":["Promotions_SGCAT2
02310300201490150 78"]})
    zz= soup.findAll("div",{"class":["promoName___2qJQm"]})
    promotions = []
    for ee_ in ee:
        ss = ee_.findAll("div",{"class":"menuItemWrapper___1xIAB"})
        for ss_ in ss:
            a,b,c = None , None , None
            kk =ss_.find("div",{"class":"ant-row menuItemInfo___PyfMY"})
            try:
                a = kk.find("p",{"class":["itemNameTitle___1sFBq"]}).text

                b = kk.find("p",{"class":["itemDescription___2cIzt"]}).text

                c = kk.find("h6",{"class":["discountedPrice___3MBVA"]}).text
            except:
                pass
```

```
            print(a,b,c)
            promotions.append([a,b,c])
def get_lat_long(address):
                base_url = "https://food.grab.com"
                params = {
                    'q': address,
                    'format': 'json',
                }
                response = requests.get(base_url, params=params)
                if response.status_code == 200:
                    data = response.json()
                    if data:
                        latitude = data[0]['lat']
                        longitude = data[0]['lon']
                        return latitude, longitude
                    else:
                        return None, None
                else:
                    print("Failed to fetch coordinates:", response.status_code)
                    return None, None

            # Get latitude and longitude for the restaurant's address
            latitude, longitude = get_lat_long(dd)

        try:
            # Print the extracted information
            print('=' * 100)
```

```
        print(f"Name: {aa}")
        print(f"Cuisine: {bb}")
        print(f"Rating: {cc}")
        print(f"Delivery Time: {dd}")
        z_res = None
        if len(zz)>=2:
            z_res = zz[1].text
            print(f"Delivery Fee: ",zz[1].text)
    except:
        out = False
        pass
    if out is True:
        with open("out.csv", 'a') as csvfile:
            # creating a csv writer object
            csvwriter = csv.writer(csvfile)
            # writing the fields
            csvwriter.writerow([aa,bb,cc,dd ,z_res, promotions])

# Function to scrape restaurants from a list of URLs
threads = []
def scrape_restaurants(urls):
    print(urls)
    if True:
        # Iterate over each restaurant element
        for url in urls:
            # Find the anchor tag containing the restaurant URL
            #c = restaurant.find('a', href=True)
```

```
        # If the anchor tag is found
        if True:
            # Create a new thread to scrape restaurant info
            thread = threading.Thread(target=scrape_restaurant_info, args=(url,))
            # Add the thread to the list
            threads.append(thread)
            # Start the thread
            thread.start()
            # Print a debug message indicating the thread is started
            print(f"Thread started for: {url}")


# Main function to orchestrate the scraping process
def main():
    # URL of the main page containing restaurant links
    url = "/sg/en/restaurants"
    # Send a GET request to the main page
    r = requests.get(BASE_URL + url, headers=HEADERS)
    # Parse the HTML content of the response
    soup = BeautifulSoup(r.content, 'html.parser')
    # Find all elements containing restaurant information
    b = soup.find_all(class_=["ant-col-24 RestaurantListCol___1FZ8V", "ant-col-md-12", "ant-col-lg-6"])
    # Scrape restaurant information from the list of URLs
    scrape_restaurants([restaurant.find('a', href=True)["href"] for restaurant in b if restaurant.find('a', href=True)])
# Entry point of the script
if __name__ == "__main__":
    main()
```

Explanation of Code:

1. Importing Libraries:

- requests: Used for making HTTP requests to the website.
- BeautifulSoup: A Python library for parsing HTML and XML documents.

2. Setting Up Variables:

- HEADERS: A dictionary containing the user-agent header to mimic a web browser.
- BASE_URL: The base URL of the website to be scraped.

3. Function Definitions:

- scrape_restaurant_info(url): Scrapes information of an individual restaurant from its URL.
- scrape_restaurants(urls): Scrapes restaurants from a list of URLs using multithreading.
- main (): Main function to orchestrate the scraping process.

4. Scraping Individual Restaurant Information (`scrape_restaurant_info(url)`):

- Sends a GET request to the restaurant URL.
- Parses the HTML content of the response using BeautifulSoup.
- Extracts relevant information such as restaurant name, cuisine, rating, delivery time, and promotions.
- Writes the extracted information to a CSV file named "out.csv".

5. Scraping Restaurants from a List of URLs (scrape_restaurants(urls)):

- Creates a new thread for each restaurant URL to scrape restaurant information concurrently.
- Starts each thread to perform scraping asynchronously.

6. Main Function:

- Sends a GET request to the main page containing restaurant links.
- Parses the HTML content of the response using BeautifulSoup.
- Finds all elements containing restaurant information.
- Extracts restaurant URLs from the elements and initiates scraping of each restaurant using multithreading.

7. Entry Point of the Script:

- Checks if the script is being run directly (not imported as a module).
- Calls the main () function to start the scraping process.

Converting CSV into ndjson:

```
import csv
import json

csvfile = open('out.csv', 'r')
jsonfile = open('out.ndjson', 'w')

fieldnames = ("Name","Cuisine","Rating","Delivery Time")
reader = csv.DictReader( csvfile, fieldnames)
for row in reader:
    json.dump(row, jsonfile)
    jsonfile.write('\n')
```

Explanation of Code:

1. Imports:

- csv: This module provides functionality for reading and writing CSV files.
- json: This module provides functions for encoding and decoding JSON data.

2. File Handling:

- csvfile = open('out.csv', 'r'): Opens the CSV file named "out.csv" in read mode ('r').
- jsonfile = open('out.ndjson', 'w'): Opens a new file named "out.ndjson" in write mode ('w') to store JSON data.

3. Defining Fieldnames:fieldnames = ("Name","Cuisine","Rating","Delivery Time"): Defines the field names expected in each row of the CSV file. These will be used as keys in the JSON objects.

4. Reading CSV File and Writing to JSON File:

- reader = csv.DictReader(csvfile, fieldnames)`: Creates a `DictReader` object to read the CSV file. Each row is converted into a dictionary using the provided `fieldnames`.
- for row in reader: Iterates over each row in the CSV file.

5. Closing Files: The files are not explicitly closed in the provided code snippet. It's good practice to close the files after you've finished working with them. You can do this by adding `csvfile.close()` and `jsonfile.close()` after the loop, or by using a context manager (`with` statement) to automatically close the files after the block of code finishes execution.

Link:

https://drive.google.com/file/d/1o8hQI5OpKlY0Ctnd6XlujAjiUbX5OTu6/view?usp=sharing

This is a video link and it is all about the implementation of code

Latitude and Longitude:

```
==========
Name: Sanook Kitchen - Funan Mall
Cuisine: Thai
Rating: 4.3
Delivery Time: 45 mins  •  0.9 km
Name: Guzman Y Gomez - Funan
Cuisine: Mexican,Kids Friendly,Healthy,Group Friendly
Rating: 4.2
Delivery Time: 45 mins  •  0.9 km
==========
Name: Tandoori Zaika - 70 Boat Quay
Cuisine: Indian,Salad,Seafood,Vegetarian Friendly
Rating: 4.3
Delivery Time: 65 mins  •  1.2 km
==========
Name: Sukiya - Funan
Cuisine: Japanese,Kids Friendly,Drinks & Beverages,Seafood,Halal
Rating: 4.3
Delivery Time: 35 mins  •  0.5 km
==========
```

Output:

```
/sg/en/restaurant/dapur-penyet-north-bridge-road-delivery/4-C6EFRUMTJU2JBA?
==========
Dapur Penyet - North Bridge Road
Indonesian,Halal,Seafood
4.2
30 mins  •  0.5 km
/sg/en/restaurant/wok-hey-funan-delivery/4-C2VKJ3U2C2JZTN?
==========
WOK HEY - Funan
Bento,Chinese,Local & Malaysian,Halal,Group Friendly
4.4
45 mins  •  0.8 km
/sg/en/restaurant/koi-th%C3%A9-funan-delivery/4-C4NJGEDEE2X1TN?
==========
KOI Thé - Funan
Bubble Tea,Coffee & Tea,Drinks & Beverages
4.7
35 mins  •  0.5 km
/sg/en/restaurant/stuff-d-funan-delivery/4-C2VKLAC2MBT2NJ?
==========
Stuff'd - Funan
Healthy,Salad,Mexican,Halal,Group Friendly
4.3
35 mins  •  0.5 km
/sg/en/restaurant/liho-tea-funan-delivery/4-CZCBCGNXAY5ACA?
==========
LiHO TEA - Funan
Coffee & Tea,Drinks & Beverages,Bubble Tea,Group Friendly
4.5
40 mins  •  0.9 km
/sg/en/restaurant/sanook-kitchen-funan-mall-delivery/4-C2E3GCCFEEWXAN?
```

```
/sg/en/restaurant/sanook-kitchen-funan-mall-delivery/4-C2E3GCCFEEWXAN?
==========
Sanook Kitchen - Funan Mall
Thai
4.3
40 mins  •  0.9 km
/sg/en/restaurant/espresso-doc-chan-brother-68-delivery/4-C3LFA2NHGJ2BAN?
==========
espresso.doc - Chan Brother @ 68
Pizza,Sandwiches,Breakfast,Coffee,Juice
4.4
35 mins  •  1.1 km
/sg/en/restaurant/subway-funan-delivery/4-CZDKG25AVYVAT6?
==========
Subway - Funan
Sandwiches,Halal,Fast Food,Group Friendly
4.4
35 mins  •  0.5 km
/sg/en/restaurant/paradise-dynasty-funan-delivery/4-CYUTT6CDRUXXL6?
==========
Paradise Dynasty - Funan
Chinese,Noodles,Rice
4.6
40 mins  •  0.9 km
/sg/en/restaurant/briyani-hut-boat-quay-delivery/4-C32KFE3EFB31T2?
==========
Briyani Hut - Boat Quay
Indian,Fried Chicken,Local & Malaysian
4.3
35 mins  •  1.1 km
/sg/en/restaurant/guzman-y-gomez-funan-delivery/4-CZEUR7LAUCDEC2?
==========
Guzman Y Gomez - Funan
```