Web Scraper

Gurpreet Singh

March 6, 2025

1 Introduction

The web scraper, an integral part of the project, as it allows to fetch article data for the implementation of the models. The web scraper was initially designed to scrape publicly accessible data from various websites in order to minimise bias and increase credibility of data. However, along with other changes, the scraper was made to be specific to one website, the BBC news.

BBC news, a leading news agency in the UK. (**Add more context here**).

2 Design

The design of the scraper has been modified several times over the course of the project to ensure that the most optimal and efficient design is chosen to strictly adhere to time guidelines.

2.1 Initial Design

2.1.1 Methodology

Many methodologies were explored such as traditional and hybrid models. It was decided that the hybrid models would be best hence the spiral model was chosen as it allowed continuous improvement until the sought for product was made.

add figure showing visual representation of model and description using references

2.1.2 Structure

The use of different python scripts was firstly discussed to ensure that each component would work well independently and therefore not become an unsolvable issue where at the last step of the implementation all components would be imported into one file and that would be the executable.

The design revolved around each script being able to do only one component hence the python scripts were first designed as expressed in table 1.

Script Name	Description
scraper	loading the API or bs4.
fetching	accessing different data from the website.
dataFormat	import and export of data to and from the python data types.
main	executing all scripts in order to run the web scraper.

Table 1: Python scripts

2.1.3 Python Modules

It was important to understand that to gain access to data, a secure path was needed to the website, in light of this there were two potential pathways that were explored:

1. API;
2. Selenium;

3. BeautifulSoup4(bs4).

All three of the options had their advantages and limitation, and predominantly all deemed as not viable options. API's were either specific to the different projects worked in, outdated or simply missing components. Whereas, selenium and bs4, due to lack of prior knowledge, became difficult to comprehend, establish a successfully connection and essentially scrape the website.

To transfer data to files the csv module was initially selected as the most expertise were with that file type, however this was deemed inefficient as the data was not stored to be easily accessible.

Request module was also needed in coherence to the above outlined options 1-3. This module was used to send requests which allowed to gain access to specific data from the HTML.

2.2 Current design model

The current design model was completed after careful consideration of deadlines, limitations and through trial and error of software.

2.2.1 Methodology

The methodology remained the same, the spiral method. The hybrid model allowed constant critique of implemented code whilst allowing the betterment of the scraper. Each process would be put through the process individually to ensure that before the scraper work as a whole they work individually. The process are than collected and executed together and the method is implemented on the code again.

2.2.2 Structure

The processes were not to be implemented the same way as a decision was made before implementing to use the OOP structure to ensure that a scraper object can be called and used as easily on future projects as well. The differentiation is that for this to be successfully be operational, all components were needed to be further broken down so that each component had subroutines which could be called individually. The subroutines which are listed in Table 2 are carefully outlined so that the implementation would be completed with great regards to time efficiency.

Subroutine name	Function
init	set self variables
fetchdata	load and manipulate data from json files
loadHtml	get HTML of page
getarticledata	execution of all subroutines to get article data
getlinks	getting all article links from results page
main	executing all scripts in order to run the web scraper.

Table 2: WebScraper class subroutines

3 Implementation

The implementation was carried out over several months and completed to satisfy basic requirements of the scraper. An OOP model was created to ensure smooth modification at later stages. The scraper collects data into two json files, an article links file and and article data file. (**add figures and make metadata**)

The webscraper class is the only class in the script defined on line 16, in Appendix A.1. The webscraper class has one parameter, an initial url needs to be inputted for it to work. This is done on lines 194, 208, 228, 246 in appendix A.1.

A Appendix- Web Scraper code

```
###
#WebScraper
#OOP designed software to scrape data off given website and collect publicly available d
#Gurpreet singh
#21131818
###

from requests_html import HTMLSession
from time import sleep
from datetime import datetime
from dateutil import parser
import json
import os
#import all required modules

class Webscraper():
    def __init__(self, url) -> None:
        #initial constructor creation
        self.session = HTMLSession()
        self.base_url = "https://www.bbc.co.uk/"
        self.url = url

    def load_html (self):
        #load the html for the webpage
        self.reader = self.session.get(self.url)
        self.reader.html.render(sleep=1, scrolldown=0)

    def close(self):
        #used to complete session and free all resources
        self.session.close()

    def fetch_aldata(self, div_finder):
        #this selects the link and title from the div by finding correct data in the htm
        divs = self.reader.html.find(div_finder)
        data = []
        for div in divs:
            # Goes through specific objects to fetch data
            link_element = div.find("a", first=True)
            if link_element:
                link = link_element.attrs.get('href', '')
                title = link_element.text
                data.append({'title':title, 'link': link})
        return data

    def fetch_adata(self):

        article_data = []
        msg = "No"
        article = ''
        title = (self.reader.html.find('title', first=True) or self.reader.html.find('h1

        try:
            paragraphs = self.reader.html.find('article')
            if paragraphs:
                for p in paragraphs:
                    article += p.text.strip()
            else:
                article = msg
        except Exception as e:
            print(f"An error occurred: {e} article")

        date = None
        try:
            time_elements = self.reader.html.find('time')
            if time_elements:

                date_published = time_elements[0].text.strip() or time_elements[0].dateti
                print (date_published)
                try:
                    date_published = parser.parse(date_published, default=datetime(dateti
                    date_published = date_published.strftime('%d-%m-%Y')
                except:
                    date_published = msg


            else:
                date_published = msg
        except Exception as e:
            print(f"An error occurred: {e} time")



        try:
            author = self.reader.html.find('.ssrcss-68pt20-Text-TextContributorName', fi
            author = author.text

            if not author:
                author = msg
        except Exception as e:
            print(f"An error occurred: {e} author")

        article_data.append({'title': title, 'article': article, 'Publishdate': date_pub

        return article_data

    def pagination(self):
        #get links for all successive pages of the results
        self.reader = self.session.get(self.url)
        npg_links = []
        for link in self.reader.html:
            if link.search("page="):
                print("yes")
                #sleep(0.5)
                link = str(link)
                link = link.split("'")[1]
                npg_links.append(link)
                sleep(1)

            else:
                break
        return npg_links

    def link_format(self):
        #check url is correct format
        search_ext = self.url
        if search_ext and not search_ext.startswith("https://"):
                self.url = self.base_url + search_ext
        else:
            self.url = search_ext
            return self.url
        pass

    def file_checker(self, found, filename):

        path = os.path.dirname(os.path.realpath(__file__))
        destination = ""
        if found == False:

            for root, dirs, files in os.walk(path):
                for file in files:
                    if file.endswith('.json') and file.startswith(filename):
                        print ('yes')
                        destination =  (root+'/'+str(file))
                        print (destination)
                        found = True

        return found, destination

    def datatype_conversion(self, found, data, destination):

        if found  == False:

            json_obj = json.dumps(data, indent=4)
            with open(destination, "w") as crfile:
                crfile.write(json_obj)
                found = True
                return found, 'File created and hyperlinks stored successfully'

        elif found == True:
                with open(destination, 'r+')  as file:
                    json_obj = json.load(file)
                for item in data:
                    json_obj.append(item)

                print(json_obj)

                with  open(destination, 'w') as file:
                    json.dump(json_obj, file,
                                indent=4,
                                separators=(',',': '))

                return found, 'Data stored successfully'



    def fetch_adata_links(self,filename):
        with open(filename, 'r+') as file:
            link_data = json.load(file)
            return link_data

    def div_select(self):
        #div selector for specific data needed from html
        if self.url  and not self.url.endswith("NEWS_PS") or self.url.find("page="):
            div_finder = ".ssrcss-tq7xfh-PromoContent > *"

        else:
            div_finder= "h1.ssrcss-1j5vay3-Heading.e1hq9lx0"
            #should be for individual article page - find specific div that correllates
            #might need more thn one check page

        return div_finder

def get_article_link_data ():
    #runs all the function and evokes the object
    has_run = False
    div_finder = ".ssrcss-tq7xfh-PromoContent > *"
    found  = False
    filename = 'article_links'



    while has_run  == False:
        url = "search?q=s%26p+500&seqId=e1005640-2774-11ef-b757-6398eaf17df6&d=NEWS_PS"
        scraper = Webscraper(url)
        scraper.link_format()
        npg_links = scraper.pagination()
        has_run = True
        found, destination = scraper.file_checker(found, filename)
        print("—")
        print(found)
        scraper.close()


    try:

        for link in npg_links:
            url = link
            scraper = Webscraper(url)
            print(scraper.url)
            div_finder = scraper.div_select()
            scraper.load_html()
            data = scraper.fetch_aldata(div_finder)
            found, message = scraper.datatype_conversion(found, data, destination)
            print (message)
            for item in data:
                print(item)
            sleep(1)
            print("__________________________________")
            scraper.close()
            print(found)

    except Exception as e:
        print(f"An error occurred: {e}")
```

```python

def get_article_data():
    url = "search?q=s%26p+500&seqId=e1005640-2774-11ef-b757-6398eaf17df6&d=NEWS_PS"
    scraper = Webscraper(url)
    al_file = 'article_links.json'
    a_file = 'article_data.json'
    url = scraper.fetch_adata_links(al_file)
    found = False
    found, filename = scraper.file_checker(found, al_file)
    link_data = scraper.fetch_adata_links(filename)
    found = False
    found, destination = scraper.file_checker(found, a_file)
    if found == False:
        destination = a_file


    try:

        for link in link_data:
            url = (link['link'])
            print(url)
            scraper = Webscraper(url)
            scraper.load_html()
            data = scraper.fetch_adata()
            found, message = scraper.datatype_conversion(found, data, destination)
            print(message)
            print('----------------------------')

            scraper.close()


    except Exception as e:
        print(f"An error occurred: {e}")

def main():
    get_article_data()
    #get_article_link_data()




if __name__ == "__main__":
    main()
```