Quality & Quantity - Supplementary Document

SUPPLEMENTARY MATERIALS

Algorithmic Thinking in the Public Interest: Navigating Technical, Legal, and Ethical Hurdles to Web Scraping in the Social Sciences

Alex Luscombe · Kevin Dick · Kevin Walby

Abstract These supplementary materials contain several examples of scrapers developed to accomplish tasks ranging from the very simple to the increasingly complex. All source code for these examples can be found at the following GitHub repository for use by the broader research community github.com/CAIJ-UW/automated-access.

Alex Luscombe
Centre for Criminology & Sociolegal Studies
University of Toronto
Toronto, Ontario, Canada M5S 3K9
ORCID: 0000-0002-3052-5401

Kevin Dick
Department of Systems and Computer Engineering
Carleton University
Ottawa, Ontario, Canada K1S 5B6
ORCID: 0000-0003-3931-523X

Kevin Walby
Department of Criminal Justice
University of Winnipeg
Winnipeg, Manitoba, Canada R3B 2E9
ORCID: 0000-0002-5107-2309

```python
""" Example 1
"""
import requests
r = requests.get('https://example-website.com/file.json')
print(r.json())
```

Supplementary Algorithm 1 Example Usage of Python's Requests Library to Download a Single JSON File.

```python
""" Example 2
"""
from ba4 import BeautifulSoup
import urllib2

url = "https://www.example-website.com"
content = urllib2.urlopen(url).read()
soup = BeautifulSoup(content)

print (soup. prettify()})

# Print all the links that appear on this page
for link in soup.find_all('a'):
print (link.get('href'))
```

Supplementary Algorithm 2 Example Usage of Python's BeautifulSoup Library to obtain all URL Links that Appear on the Target Webpage.

```python
""" Example 3
"""
from bs4 import BeautifulSoup
import urllib2

target_urls = ['www.example-website-1.com',
               'www.example-website-2.com',
               'www. example-website-3.com',
                ...
               'www.example-website-n.com']
for url in target_urls:
    content = urllib2.urlopen(url).read({}
    soup = BeautifulSoup(content)
    print (soup.prettify()}

    # Print all the links that appear on this page
    for link in soup.find_all('a'):
        print (link.get('href'))
```

Supplementary Algorithm 3 Extended Example Using BeautifulSoup to obtain all URL Links that Appear on Several Target Webpages.

```python
""" Example 4:
    Given that many government websites bury their data in drill-down menus or tables, a multi-level navigation is required and Selenium is used to automate the click-through navigation to finally arrive to the specific data of interest. These data are scraped using Python and saved as a JSON file for subsequent analysis.
    ---
    Adapted from a Tutorial by Dave Gray
    Original: https://bit.ly/30Q05Ns
    Scraped Source: http://kanview.ks.gov/PayRates/PayRates_Agency.aspx
"""
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from bs4 import BeautifulSoup
import re
import pandas as pd
from tabulate import tabulate
import os

# Specify the URL for the target URL
url = "http://kanview.ks.gov/PayRates/PayRates_Agency.aspx"

# Initialize a Firefox session
driver = webdriver.Firefox()
driver.implicitly_wait(30)
driver.get(url)

# After opening the url above, Selenium clicks the specific agency link
python_btn = driver.find_element_by_id('MainContent_uxLevel1_Agencies_uxAgencyBtn_33')
python_btn.click() # click the link that corresponds to FHSU

# Selenium provides the page to BeautifulSoup which can then be parsed
soup_level_1 = BeautifulSoup(driver.page_source, 'lxml')

# Create an empty list
data = []

# Beautiful Soup finds all Job Title links on the agency page and the loop begins
for link in soup_level_1.find_all('a', id=re.compile("^MainContent_uxLevel2_JobTitles_uxJobTitleBtn_")):
    # Selenium visits each Job Title page
    python_btn = driver.find_element_by_id('MainContent_uxLevel2_JobTitles_uxJobTitleBtn_' + str(len(data)))
    python_btn.click() #click Link

    #Selenium hands of the source of the specific job page to Beautiful Soup
    soup_level_2=BeautifulsSoup(driver.page_source, 'lxml')

    # Beautiful Soup grabs the HTML table on the page
    table = soup_level_2.find_all('table')[0]

    # Giving the HTML table to pandas to put in a dataframe object
    df = pd.read_html(str(table), header=0)

    # Store the dataframe in a list
    data.append(df[0])

    # Use Selenium to navigate back by clicking the back button
    driver.execute_script("window.history.go(-1)")

# Loop ends, data are collected, close the Selenium browser session
driver.quit()

# Combine the pandas dataframes into a snigle large dataframe
result = pd.concat([pd.DataFrame(data[i]) for i in range(len(data))], ignore_index=True)

# Converet the pandas dataframe to JSON
json_data = result.to_json(orient='records')

# Pretty print the results using tabulate (convert to an ascii table)
print(tabulate(result, headers=["Employee Name","Job Title","Overtime Pay","Total Gross Pay"],tablefmt='psql'))

# Save to file in current working directory
open("fhsu_payroll data.json","w").write(json_data)

# That's all folks!
```

Supplementary Algorithm 4 Full Example Using Multiple Scraping Libraries for Complex Website navigation and Data Extraction.

```python
"""
Description: A Selenium-based scraper to download the
PDFs listed on the DocumentCloud of Muckrock-Canada.
"""
import os
from selenium import webdriver
import requests
import argparse

parser = argparse.ArgumentParser(description='')
parser.add_argument('-o', '--output_dir', required=True,
                    help='output directory for the PDFs')
parser.add_argument('-v', '--verbose', action='store_true',
                    help='increase verbosity')
args = parser.parse_args()

# EXAMPLE: python3 selenium_documentcloud_ATIPs.py  -o ./Muckrock-
    Canada/ -v

# Constants for the scraping project
BASE_URL  = 'https://www.documentcloud.org/public/search/Group:%20
    muckrock-canada'
MATCH_URL = 'documentcloud.org/document'
DRIVER    = '/usr/local/bin/chromedriver' # Specify the location
    of your own chromedriver installation
MAX_PAGES = 478

def get_page_links(driver, url):
  """ get_page_links
      With the created driver, navigate to the next page
      and get all ahref links that match the MATCH_URL.
      ---
      Input:  <driver> driver, instantiated driver
      Output: <list> links, all page links matching the MATCH_URL
  """
  driver.get(url)
  links = []

  # Obtain a list of the links on this page
  elems = driver.find_elements_by_xpath("//a[@href]")
  for elem in elems:
    link = elem.get_attribute("href")
    if MATCH_URL in link:
      if args.verbose: print(f'adding {link}')
      links.append(link)
  return links

def main():
  """ main function """
  driver = webdriver.Chrome(DRIVER)
  all_links = []

  for i in range(1, MAX_PAGES + 1):
    page = f'{BASE_URL}/p{i}'
    if args.verbose: print(f'acquiring page: {page}')
    links = get_page_links(driver, page)
    for link in links:
      download = link.replace('html', 'pdf')
      filename = download.split('/')[-1]

      if args.verbose: print(f'Downloading: {download}')
      doc = requests.get(download)
      if args.verbose: print(f'Saving: {download}')
      open(os.path.join(args.output_dir, filename), 'wb').write(doc.content)
  driver.quit()

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

Supplementary Algorithm 5 Full Example Using the Selenium Library to Automatically Navigate a Document Repository and Extract PDFs.

```python
"""
Description: Scrapes the Springer PDFs from
the Free-Springer-Ebooks.csv file
"""
import os
from bs4 import BeautifulSoup
import requests
import argparse

parser = argparse.ArgumentParser(description='')
parser.add_argument('-i', '--input', required=False, default='Free-Springer-Ebooks.csv',
                    help='input csv containing info on free Spriinger books')
parser.add_argument('-o', '--output_dir', required=False, default='./pdfs/',
                    help='output file')
parser.add_argument('-v', '--verbose', action='store_true',
                    help='increase verbosity')
args = parser.parse_args()

# EXAMPLE: python3 springer_scraper.py -v

def get_download_link(url):
  html = requests.get(url).text
  bs = BeautifulSoup(html)
  for link in bs.find_all('a'):
        if link.has_attr('href'):
            if 'content' in link.attrs['href'] and 'pdf' in link.attrs['href']: return 'https://link.springer.com' + link.attrs['href']


def main():
  """ main function """
  if not os.path.exists(args.output_dir): os.mkdir(args.output_dir)
  for book in open(args.input, 'r').readlines():
    title  = '-'.join(book.split(',')[1].split())
    author = '-'.join(book.split(',')[2].split())
    year   = '-'.join(book.split(',')[3].split())
    url    = book.split(',')[-1].strip()
    isbn   = url.split('isbn=')[-1]
    filename = '_'.join([title, author, year]) + '.pdf'
    download = get_download_link(url)

    if args.verbose: print('Title: {}\nAuthor: {}\nYear: {}\nURL: {}\nISBN: {}\nFilename: {}\nDownload: {}\n'.format(title, author, year, url, isbn, filename, download))

    cmd = 'curl {} --output {}'.format(download, os.path.join(args.output_dir, filename))
    if args.verbose: print('Downloading {}\n'.format(filename))
    os.system(cmd)

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

Supplementary Algorithm 6 Full Example Using the BeautifulSoup Library to Automatically Parse a Collection of Document Links to then Extract the Desired PDFs.