Webscraping Blog

What is Web Scraping

Web Scraping is the process of extracting data froma website. Collecting product prices off of Amazon or movie ratings off of IMDB would both be forms of web scraping. On a typical user-facing website, data is embedded in a web page in a way that is easy to interpret but hard to extract and consolidate by hand.

Web scrapers are tools that can automate the process of data collection from a browser. They serve as an interface between the web browser and the scraping instructions. The instructions can be written using most programming languages with some langauges having dedicated libraries for webscraping. Below are two examples of web scraping tools.

Selenium

Selenium WebDriver

Selenium is a webdriver designed to automate web browsing. It simulates normal user interaction with a web browser but interfaces with coding languages. This allows specific and complex instructions to be fed in and run.

Setting Up

Using Selenium with Python requires downloading the required library using pip

```
pip install selenium
```

In addition, the browser driver also needs to be installed which will depend on the browser being utilized (Chrome, Firefox, Internet Explorer). That is all the setup required to experiment with the web driver. In the case of Google Chrome, Google supports and updates the Chrome driver. The chrome web driver can be found here:

https://chromedriver.chromium.org/downloads

Experimenting with Selenium

The goal of this experiment is to open youtube in a chrome browser and search for casey neistat

In [ ]:

```
from selenium import webdriver
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.common.by import By
import time

# Open chrome webdriver
with webdriver.Chrome('/Users/USERNAME/Desktop/Programming/seleniumdrivers/chrom
    #tells driver to wait 3 seconds
    wait = WebDriverWait(browser,3)
    #checks for visibility of element on webpage
    visible = EC.visibility_of_element_located

    # opens chrome browser to youtube
    browser.get('https://youtube.com')

    #waits 3 seconds before continuing code
    time.sleep(3)

    # wait 3 seconds or until search_query is visible in html
    wait.until(visible((By.NAME, "search_query")))

    # clicks on search query
    browser.find_element_by_name('search_query').click()

    # types in casey neistat into search query
    browser.find_element_by_name('search_query').send_keys('casey neistat')

    # clicks search button
    browser.find_element_by_id('search-icon-legacy').click()

    time.sleep(3)
```

The above code will open up Youtube in a Chrome browser and search for casey neistat. This is utilizing both the installed selenium library and the chrome driver executable. Selenium will show the scraping process in real time instead of just returning an output

Beautiful Soup

Beautiful Soup is a webscraping Python library. Instead of realtime, visible navigation the browser is navigated behind the scenes using the request library.

Experimenting with Beautiful Soup

The goal of this exercize is to retrieve data from a mock bookstore website and store the book data in a dataframe.

Imports and Helper Functions

In [ ]:

```
from bs4 import BeautifulSoup
import requests
import pandas as pd

def retrieve_titles(soup_):
    '''
    retrieves the titles of books from the webpage
    '''
    marker = soup_.find('ol',class_='row')
    list_of_titles = []
    for book in marker.findAll('h3'):
        list_of_titles.append(book.find('a').attrs['title'])

    return list_of_titles

def retrieve_ratings(soup_):
    '''
    retrieves the ratings of books from the webpage
    '''
    marker = soup_.find('ol',class_='row')
    book_ratings = []
    for p in marker.findAll('p'):
        if p.attrs['class'][-1] == 'One':
            book_ratings.append(1)
        elif p.attrs['class'][-1] == 'Two':
            book_ratings.append(2)
        elif p.attrs['class'][-1] == 'Three':
            book_ratings.append(3)
        elif p.attrs['class'][-1] == 'Four':
            book_ratings.append(4)
        elif p.attrs['class'][-1] == 'Five':
            book_ratings.append(5)
    return book_ratings

def retrieve_prices(soup_):
    '''
    retrieves the prices of books from webpage
    '''
    marker = soup_.find('ol', class_='row').findAll('p',class_='price_color')
    price_list = []
    for price in marker:
        price_list.append(float(price.text[1:]))

    return price_list

def retrieve_availabilities(soup_):
    '''
    retrieves the availability of a book from the webpage
    '''


    book_statuses = soup_.findAll('i',class_='icon-ok')
    book_status_list = []

    for book_status in book_statuses:
        book_status_list.append(book_status.nextSibling.replace('\n','').strip()

    return book_status_list

def go_to_next_page(soup_):
    '''
    moves to next page of the webpage
    '''
    button = soup_.find('li', class_='next').find('a').attrs['href']
    is_next = soup_.find('li', class_ = 'next').text == 'next'

    if button[0] == 'c':
        return [button,is_next]
    else:
        return ['catalogue/' + button , is_next]
```

Retrieving Data from Website

In [ ]:

```
# Go to mock webpage and retrieve book data. Store data as a pandas dataframe
base_html = 'http://books.toscrape.com/'
next_page_html = ''
df = pd.DataFrame()
html_page = requests.get(base_html + next_page_html)
```

```
soup = BeautifulSoup(html_page.content,'html.parser')

# build initial dataframe with website homepage
df = pd.DataFrame({'title':retrieve_titles(soup),
                   'rating': retrieve_ratings(soup),
                   'price': retrieve_prices(soup),
                   'availability': retrieve_availabilities(soup)})

# loop through each consecutive page of books  and add to dataframe
for i in range(49):
    try:
        if go_to_next_page(soup)[1]:
            next_page_html = go_to_next_page(soup)[0]
            html_page = requests.get(base_html+next_page_html)
            soup = BeautifulSoup(html_page.content,'html.parser')
            df = pd.concat([df, pd.DataFrame({'title':retrieve_titles(soup),
                                               'rating': retrieve_ratings(soup),
                                               'price': retrieve_prices(soup),
                                               'availability': retrieve_availabil
    except:
        # print page where error was returned
        print(base_html+next_page_html)

print(df.shape)
df.head()
```

```
(1000, 4)
```

Out[ ]:

	title	rating	price	availability
0	A Light in the Attic	3	51.77	In stock
1	Tipping the Velvet	1	53.74	In stock
2	Soumission	1	50.10	In stock
3	Sharp Objects	4	47.82	In stock
4	Sapiens: A Brief History of Humankind	5	54.23	In stock

Best Practices

Before continuing with some addtional instructions let's first review the ethics and best practices surrounding web scraping.

When considering web scraping it's important to first identify the purpose of the data being collected and what data needs to be collected. Once these items are determined it is easier to figure out if scraping is necessary at all. Some research should be done to see if the data to be scraped can be found on a public API. If scraping is necessary, sensitive and proprietary data should not be included. It would be better to reach out to a site owner directly for access if sensitive data is necessary. Finally, care should be taken to not hurt the performance of a website by disrupting normal website traffic with too many server requests.

Web scraping is a powerful tool that if respected can save a data scientist a lot of time with data collection.