Webscraping

Applications of Cloud Computing and Big Data - ECON 446

Bella Rakhlina
Lora Yovcheva
Mauricio Vargas-Estrada

Master Of Quantitative Economics
University of California - Los Angeles

In [ ]:
```
import requests
import string
import re
import toolz
import time
import pickle as pkl
import pandas as pd
import numpy as np
import os
import random

from bs4 import BeautifulSoup
from multiprocessing import Pool
from tqdm import tqdm
from rich import inspect
from pprint import pprint

from selenium import webdriver
from selenium.webdriver.firefox.options import Options
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.common.exceptions import NoSuchElementException

from stem import Signal
from stem.control import Controller

import tbselenium.common as cm
from tbselenium.utils import launch_tbb_tor_with_stem
from tbselenium.tbdriver import TorBrowserDriver

import pickle as pkl
```

Web Crawling Tables

a.)

Create a list of links for all the wikipedia pages for NYSE traded companies A-Z and 0-9.

In [ ]:
```
headers = {
    'Accept-Encoding': 'gzip, deflate, br',
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/42.0.2311.135 Safari/537.36 E
}
```

In [ ]:
```
url = lambda x: f'https://en.wikipedia.org/wiki/Companies_listed_on_the_New_York_Stock_Exchange_({x})'
```

In [ ]:
```
def get_table_info(soup):
    links = []
    names = []
    tickers = []

    # For each row in the table of companies.
    for r in soup.find_all('tr'):
        # Get the columns for the row `r`.
        row = r.find_all('td')
        # Check if the row contains data.
        if row:
            # Extract the link of the company in the row `r`.
            tmp_link = row[0].find_all('a')
            # Check if the company has a link.
            if tmp_link:
                names.append(tmp_link[0].text)
                # Adding the root to the relative path.
                links.append('https://www.wikipedia.org' + tmp_link[0].get('href'))
                tickers.append(row[1].text.strip())
    return names, tickers, links
```

In [ ]:
```
links = []
names = []
```

```
tickers = []
for letter in tqdm(list(string.ascii_uppercase) + ['0-9']):
    req = requests.get(url(letter), headers=headers, timeout=20)
    # Pass if the request is not successful or time out.
    if not req.ok:
        continue
    soup = BeautifulSoup(req.content, 'html.parser')
    tmp_n, tmp_t, tmp_l= get_table_info(soup)
    links.extend(tmp_l)
    names.extend(tmp_n)
    tickers.extend(tmp_t)
```

In [ ]:
```
companies = pd.DataFrame({'company': names, 'ticker': tickers, 'link': links})
```

In [ ]:
```
#companies.to_csv('data/companies_links.csv', index=False)
#companies = pd.read_csv('data/companies_links.csv')
companies = companies.drop_duplicates()
```

In [ ]:
```
print(companies.head())
```

```
                company ticker  \
0  A. O. Smith Corporation    AOS
1       A10 Networks, Inc.   ATEN
2          AAR Corporation    AIR
3             Aaron's Inc.    AAN
4                 ABB LTD.    ABB

                                        link
0      https://www.wikipedia.org/wiki/A._O._Smith
1     https://www.wikipedia.org/wiki/A10_Networks
2         https://www.wikipedia.org/wiki/AAR_Corp
3  https://www.wikipedia.org/wiki/Aaron%27s,_Inc.
4        https://www.wikipedia.org/wiki/ABB_Group
```

b.)

Crawl through all the URLs and make 1 DF with all the NYSE publically traded companies.

In [ ]:
```
def get_type(url):
    headers = {
        'Accept-Encoding': 'gzip, deflate, br',
        'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/42.0.2311.135 Safari/537
    }
    # Crawling the wiki page of the company. Rise and exception if the url
    # is broken.
    try:
        req = requests.get(url, headers=headers, timeout=20)
    except:
        print(f'Request failed in {url}')
        return ''
    # Pass if the request is not successful or time out.
    if not req.ok:
        return ''
    soup = BeautifulSoup(req.content, 'html.parser')
    # Extracting the type of the company from the infobox vcard. Rise an

    try:
        vcard = soup.find_all('table', {'class': 'infobox vcard'})[0]
        comp_type = vcard.find('a', {'title': 'Public company'}).text
    except:
        print(f'No company type in {url}')
        return ''
    return comp_type
```

In [ ]:
```
# Multiprocessing to crawl the websites in parallel.
with Pool(100) as p:
    comp_type = p.map(get_type, companies['link'])
```

In [ ]:
```
# Type of companies in lowercase.
companies['type'] = [x.lower() for x in comp_type]
```

In [ ]:
```
# Because the format is not uniform between wikis, extract the word 'public'
# from those companies that have that pattern in their type.
pattern = re.compile(r'.*public.*')
companies['type'] = [
    pattern.sub('public', s)
    if pattern.match(s) else s for s in companies.type
]
```

In [ ]:
```
#companies.to_csv('data/companies_links_with_type.csv', index=False)
#companies = pd.read_csv('data/companies_links_with_type.csv')
```

In [ ]:
```
# Companies that are public according to their wiki page.
print(companies.query("type=='public'"))
```

```
                          company ticker  \
0          A. O. Smith Corporation    AOS
1               A10 Networks, Inc.   ATEN
2                  AAR Corporation    AIR
3                     Aaron's Inc.    AAN
4                         ABB LTD.    ABB
...                             ...    ...
1522  Zimmer Biomet Holdings, Inc.    ZBH
1524                   Zoetis Inc.    ZTS
1525                   Zuora, Inc.    ZUO
1526        3D Systems Corporation    DDD
1527                    3M Company    MMM

                                              link    type
0         https://www.wikipedia.org/wiki/A._O._Smith  public
1        https://www.wikipedia.org/wiki/A10_Networks  public
2           https://www.wikipedia.org/wiki/AAR_Corp  public
3     https://www.wikipedia.org/wiki/Aaron%27s,_Inc.  public
4           https://www.wikipedia.org/wiki/ABB_Group  public
...                                              ...     ...
1522    https://www.wikipedia.org/wiki/Zimmer_Biomet  public
1524          https://www.wikipedia.org/wiki/Zoetis  public
1525           https://www.wikipedia.org/wiki/Zuora  public
1526       https://www.wikipedia.org/wiki/3D_Systems  public
1527               https://www.wikipedia.org/wiki/3M  public

[1015 rows x 4 columns]
```

c.)

What is the percetages of companies that contain 1 letter, 2 letters, 3 letters, 4 letters, 5 letters,... in the ticker (drop punctuation)?

In [ ]:
```python
# Cleaning the ticker column.
companies['clean_ticker'] = [
    re.match(r"[^,.!? ]+", x).group(0)
    for x in companies.ticker
]
```

In [ ]:
```python
# Calculating the length of the ticker.
companies['len_ticker'] = [len(x) for x in companies.clean_ticker]
```

In [ ]:
```python
# Counting the percentage of companies with each length of ticker.
len_ticker_pct = toolz.pipe(
    companies[['clean_ticker', 'len_ticker']],
    lambda x: x.drop_duplicates(),
    lambda x: x.groupby('len_ticker'),
    lambda x: x.count(),
    lambda x: x * 100 / x.sum()
)
```

In [ ]:
```python
for i, row in len_ticker_pct.iterrows():
    tmp = f'''
    ----------------------------------------
        Length of the ticker: {i}
        % in the total of companies: {row.clean_ticker:0.2f}
    ----------------------------------------
    '''
    print(tmp)
```

```
    ----------------------------------------
        Length of the ticker: 1
        % in the total of companies: 1.52
    ----------------------------------------


    ----------------------------------------
        Length of the ticker: 2
        % in the total of companies: 10.13
    ----------------------------------------


    ----------------------------------------
        Length of the ticker: 3
        % in the total of companies: 72.58
    ----------------------------------------


    ----------------------------------------
        Length of the ticker: 4
        % in the total of companies: 15.76
    ----------------------------------------
```

Web Scraping Using Beautiful Soup

a.)

Using Beautiful soup .findAll method you will webscrape the front page of Reddit. Get a list of all of the "timestamps"

In [ ]:
```
# User agent to simulate a browser in selenium.
options = Options()
options.set_preference("general.useragent.override", "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Geck
```

In [ ]:
```
# Initialize the browser.
driver = webdriver.Firefox(options=options)
```

In [ ]:
```
# Open the website in the browser.
driver.get("https://www.reddit.com")
time.sleep(5)

# Verifying the numbers of posts extracted from the page.
num_posts_before = 0

# posts inside the website to scroll down.
body = driver.find_element(By.TAG_NAME, 'body')

start_time = time.time()
time_limit_scroll = 600 # 10 minutes

# Scrolling down the page until the time limit is reached.
print('Scrolling down the page...')
while time.time() - start_time < time_limit_scroll:
    print(f'Time elapsed: {time.time() - start_time:.2f} seconds')
    # Extracting the articles from the page.
    posts = driver.find_elements(By.XPATH, '//article[@class="w-full m-0"]')

    # Verifying that the number of posts is changing in each iteration.
    if len(posts) == num_posts_before:
        print('Something went wrong. The number of posts is not changing. Stopping..')
        break
    else:
        num_posts_before = len(posts)

    # Simulate pressing the END key to scroll down the page.
    body.send_keys(Keys.END)

    # Wait for the page to load.
    time.sleep(6)
print('Time limit reached.')
```

In [ ]:
```
# Extracting the html for each post.
html_content = [x.get_attribute('outerHTML') for x in posts]
```

In [ ]:
```
# Save the html content to a pickle file.
with open('data/reddit_posts.pkl', 'wb') as f:
    pickle.dump(html_content, f)
```

In [ ]:
```
# Close the browser.
driver.quit()
```

In [ ]:
```
# Load the html content from the pickle file to avoid running the code again.
#with open('data/reddit_posts.pkl', 'rb') as f:
    #html_content = pickle.load(f)
```

In [ ]:
```
# Parsing each post with BeautifulSoup.
html_content = [BeautifulSoup(x, 'html.parser') for  x in html_content]
```

In [ ]:
```
# Extracting only the timestamps for each post.
timestamps = [x.find('time').get_attribute_list('datetime')[0] for x in html_content]
```

In [ ]:
```
print(f'Total number of posts extracted: {len(timestamps)}')
print('First 10 timestamps:')
pprint(timestamps[0:9])
```

```
Total number of posts extracted: 2478
First 10 timestamps:
['2024-04-19T07:10:50.758Z',
 '2024-04-19T17:25:09.146Z',
 '2024-04-19T13:10:37.611Z',
 '2024-04-19T11:20:55.005Z',
 '2024-04-19T15:02:59.261Z',
 '2024-04-19T19:17:01.403Z',
 '2024-04-19T02:00:27.276Z',
 '2024-04-19T11:38:59.637Z',
 '2024-04-19T19:45:27.032Z']
```

b.)

Using the functions findChild, descendents, etc. locate the post title, text and post time into a dataframe.

In [ ]:
```
titles = [x.find('article').get_attribute_list('aria-label')[0] for x in html_content]
```

In [ ]:
```
text_div_class = re.compile(r'.*feed-card-text-preview.*')
texts = []
for post in html_content:
    tmp = post.find_all(
        'div',
        class_ = lambda x : x and text_div_class.match(x)
```

```
    )
    if tmp:
        tmp = tmp[0].text
    else:
        tmp = np.nan
    texts.append(tmp)
```

In [ ]:
```
post_df = pd.DataFrame({
    'timestamp': timestamps,
    'title': titles,
    'text': texts
})
```

In [ ]:
```
# Printing the first 10 posts with text.
print('First 5 posts with text:')
print(post_df.query('text.notna()').head(5))
```

```
First 5 posts with text:
                  timestamp  \
2   2024-04-19T13:10:37.611Z
21  2024-04-19T15:55:01.669Z
23  2024-04-19T16:35:15.247Z
28  2024-04-19T05:30:53.159Z
34  2024-04-19T15:22:26.286Z

                                           title  \
2   My husband won't let me take more than two sho...
21  AITAH for breaking up with my bf after he alle...
23  After years of tipping 20-25% I'm DONE. I'm ti...
28  Orbital cooldowns are way too long compared to...
34  Is it just me or is the new Taylor swift album...

                                           text
2   \n\n    This is the weirdest thing my husband ...
21  \n\n    I'm not a clubbing kind of girl. My bf...
23  \n\n    I have always "over tipped" according ...
28  \n\n    Orbital Precision Strike has a 100s co...
34  \n\n    I'm not here to be a hater but I felt ...
```

RegEx

a.)

Using RegEx, get all the urls of ladder faculty profiles for UCLA Economics

In [ ]:
```
URL = "https://economics.ucla.edu/faculty/ladder"
```

In [ ]:
```
# Getting the flex table of the ladder faculty.
req_ladder = requests.get(URL, headers=headers)
```

In [ ]:
```
# Extracting the name and url for each profile in the ladder faculty.
tmp = re.compile(r'flex_column av_one_fourth.*')
profiles_info = toolz.pipe(
    req_ladder,
    lambda x: x.content,
    lambda x: BeautifulSoup(x, 'html.parser'),
    lambda x : x.find(
        'div',
        {'id': 'wpv-view-layout-974-CATTR0494cfbfb8d3e1b3152203680573333f'}
    ),
    lambda x: x.find_all(
        'div',
        class_ = lambda x : x and tmp.match(x)
    ),
    lambda x: [y.find('h3') for y in x],
    lambda x: {
        'name': [y.text for y in x],
        'url': [y.find('a')['href'] for y in x]
    },
    lambda x: pd.DataFrame(x)
)
```

b.)

Webcrawl the links from A and use RegEx to get all the emails and phone numbers of ladder faculty profiles.

In [ ]:
```
# Crawling the profiles urls.
crawled_info = []
for url in tqdm(profiles_info['url']):
    tmp = requests.get(url, headers=headers).content
    crawled_info.append(tmp)
    time.sleep(10)
```

In [ ]:
```
# Extracting the email and phone from the crawled info using regex.
regex_email = re.compile(r'[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+')
regex_phone = re.compile(r'([\+]?[(]?[0-9]{3}[)]?[-\s\.][0-9]{3}[-\s\.][0-9]{4,6})')
email = []
phone = []
```

```
for raw in crawled_info:
    tmp = BeautifulSoup(raw, 'html.parser')
    tmp_email = tmp.find(string=regex_email)
    tmp_phone = tmp.find(string=regex_phone)

    if tmp_email:
        email.append(tmp_email)
    else:
        email.append(np.nan)

    if tmp_phone:
        phone.append(tmp_phone)
    else:
        phone.append(np.nan)
```

In [ ]:
```
profiles_info['email'] = email
profiles_info['phone'] = phone
```

In [ ]:
```
# Printing those profiles with phone number.
print(profiles_info.query('phone.notna()').head(10))
```

```
                  name                                                url  \
8         Ariel Burstein  https://economics.ucla.edu/person/ariel-burstein/
12      Pablo Fajgelbaum  https://economics.ucla.edu/person/pablo-fajgel...
17    Martin B. Hackmann  https://economics.ucla.edu/person/martin-b-hac...
18          Jinyong Hahn    https://economics.ucla.edu/person/jinyong-hahn/
20        Hugo Hopenhayn  https://economics.ucla.edu/person/hugo-hopenhayn/
23  Adriana Lleras-Muney  https://economics.ucla.edu/person/adriana-ller...
25                Jay Lu          https://economics.ucla.edu/person/jay-lu/
26          Rosa Matzkin  https://economics.ucla.edu/person/rosa-liliana...
27     Maurizio Mazzocco  https://economics.ucla.edu/person/maurizio-maz...
28      Kathleen McGarry  https://economics.ucla.edu/person/kathleen-mcg...

                       email            phone
8        arielb@econ.ucla.edu   (310) 206-6732
12  pfajgelbaum@econ.ucla.edu   (310) 794-7241
17     hackmann@econ.ucla.edu   (310) 825-1011
18         hahn@econ.ucla.edu   (310) 825-1011
20        hopen@econ.ucla.edu   (310) 206-8896
23      alleras@econ.ucla.edu   (310) 825-3925
25          jay@econ.ucla.edu   (310) 825-7380
26      matzkin@econ.ucla.edu   (310) 825-7371
27     mmazzocc@econ.ucla.edu   (310) 825-6682
28           mcgarry@ucla.edu   (310) 825-1011
```

In [ ]:
```
tmp = f"""
Number of profiles: {profiles_info.shape[0]}
Number of profiles with phone number: {profiles_info.query('phone.notna()').shape[0]}
"""
print(tmp)
```

```
Number of profiles: 45
Number of profiles with phone number: 16
```

Selenium

a.)

Pick a website that has useful data to a business or economic question. Put your website you plan to scrape here:

https://docs.google.com/spreadsheets/d/1PJ2DOTCVCh51fn0ry1yB7qTyccR33_IXFpkYdd58MFs/edit?usp=sharing

You must have use website that no other group has. First come first serve.

The selected website, www.realtor.com, offers listings for properties for sale and rent, including those around the UCLA campus, which is our area of interest. We are particularly focused on gathering rental information from the following neighborhoods:

- Bel Air
- Brentwood
- Culver City
- Encino
- Mar Vista
- Mid Wilshire
- Pacific Palisades
- Palms
- Playa del Rey
- Playa Vista
- Santa Monica
- Sawtelle
- Sherman Oaks
- Studio City
- Venice
- West Los Angeles
- Westwood

To facilitate webscraping, we are integrating `Selenium` with `Tor`. The following code snippet is utilized to extract the required information from the website.

b.)

Use Selenium to scrape valuable information from your website and store in a dataframe.

> In this section it is defined the functions that will be used to scrape the website. The functions are defined in the following order:
>
> - `save_data` : Function to save data to a pickle file.
> - `random_sleep` : Function to generate a random sleep time to mimic human behavior.
> - `human_like_scroll` : Function to simulate human-like scrolling behavior more slowly.
> - `switch_tor_circuit` : Function to switch the Tor circuit.
> - `get_tor_version` : Function to get the Tor version.
> - `get_current_circuit` : Function to retrieve and print the current Tor circuit.
> - `zipcode_scrapping` : Function to scrape the website using Tor and Selenium.
>
> After defining the functions, the neighborhoods of interest are scraped using the `zipcode_scrapping` function. The results are saved to pickle files for further analysis.

In [ ]:
```python
def save_data(data, filename):
    script_directory = os.path.dirname(os.path.abspath(__file__))
    file_path = os.path.join(script_directory, filename)
    with open(file_path, 'wb') as file:
        pkl.dump(data, file)
    print(f"Data saved to {file_path}")
```

In [ ]:
```python
def random_sleep(minimum=2, maximum=5):
    """Generate a random sleep time to mimic human behavior."""
    time.sleep(random.uniform(minimum, maximum))
```

In [ ]:
```python
def human_like_scroll(driver):
    """Simulate human-like scrolling behavior more slowly."""
    total_height = driver.execute_script("return document.body.scrollHeight")
    current_scroll_position = 0
    increment = total_height / 20  # Divide the scroll into smaller steps

    while current_scroll_position <= total_height:
        # Scroll down to the next increment
        driver.execute_script(f"window.scrollTo(0, {current_scroll_position});")
        current_scroll_position += increment

        # Wait a random time between scrolls to mimic human behavior
        time.sleep(random.uniform(0.5, 3))  # Adjust timing as needed

    # Finally, scroll to the very bottom to ensure all lazy loaded items are triggered
    driver.execute_script("window.scrollTo(0, document.body.scrollHeight);")
    time.sleep(random.uniform(15, 20))  # A final pause at the bottom
```

In [ ]:
```python
def switch_tor_circuit():
    """Use Stem to switch Tor circuit."""
    with Controller.from_port(port=9251) as controller:
        controller.authenticate()
        controller.signal(Signal.NEWNYM)
```

In [ ]:
```python
def get_tor_version():
    try:
        with Controller.from_port(port=9251) as controller:
            controller.authenticate()  # Asume que no se necesita contraseña
            version = controller.get_version()
            print("Connected to Tor version:", version)
    except Exception as e:
        print(f"Error connecting to Tor control port: {e}")
```

In [ ]:
```python
def get_current_circuit():
    """Retrieve and print the current Tor circuit."""
    try:
        with Controller.from_port(port=9251) as controller:
            controller.authenticate()  # Assumes no password is needed
            for circ in controller.get_circuits():
                if circ.status == 'BUILT':
                    print("Circuit ID: {}".format(circ.id))
                    print("Circuit Path:")
                    for i, entry in enumerate(circ.path):
                        desc = controller.get_network_status(entry[0], None)
                        fingerprint, nickname = entry
                        address = desc.address if desc else 'unknown'
                        print(f"  {i+1}: {nickname} ({fingerprint}) at {address}")
                    print("\n")
                    break  # Just show the first 'BUILT' circuit
    except Exception as e:
        print(f"Error retrieving current circuit: {e}")
```

In [ ]:
```python
def zipcode_scrapping(zipcode):
    tor_path = '/home/m4wnn/tor-browser-linux-x86_64-13.0.14/tor-browser'
```

```
while True:
    try:
        tor_process = launch_tbb_tor_with_stem(tbb_path=tor_path)
        break
    except Exception as e:
        print(f"Error initializing Tor process: {e}")
        print("Retrying to initialize.")
print("Tor process initialized.")


## Setting a random user-agent using pref_dict
user_agent_list = [
    "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.212 Safari/537.36",
    "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/14.0.3 Safari/605.1.
    "Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like Gecko",
    "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:88.0) Gecko/20100101 Firefox/88.0",
    "Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:57.0) Gecko/20100101 Firefox/57.0",
    "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.3",
    "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/53
    "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/88.0.4324.150 Safari/537.36",
    "Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0",
    "Mozilla/5.0 (Windows NT 6.3; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/77.0.3865.90 Safari/537.36"
]

random_user_agent = random.choice(user_agent_list)
pref_dict = {"general.useragent.override": random_user_agent}

# Create a TorBrowserDriver instance with customized user-agent
try:
    driver = TorBrowserDriver(
        tor_path,
        tor_cfg=cm.USE_STEM,
        pref_dict=pref_dict,
        headless=True
    )
except Exception as e:
    print(f"Error creating TorBrowserDriver: {e}")
    return []

time.sleep(3)
#driver.get(f'https://www.realtor.com/realestateandhomes-search/{zipcode}')
driver.get(f'https://www.realtor.com/apartments/{zipcode}')
tmp_current_url = driver.current_url
print(tmp_current_url)

random_sleep(5, 10)



regex_property = re.compile(r'placeholder_property_\d*')
property_info = []
try:
    refresh_count = 0
    while True:
        human_like_scroll(driver)

        try:
            property_section = driver.find_element(
                By.CSS_SELECTOR,
                'section[class^="PropertiesList_propertiesContainer"]'
            )
            section_content = BeautifulSoup(
                property_section.get_attribute('innerHTML'),
                'html.parser'
            )

            properties = section_content.find_all(
                'div',
                id=lambda x: x and regex_property.match(x)
            )

            property_htmls = [str(prop) for prop in properties]
            property_info.extend(property_htmls)


        except NoSuchElementException:
            if refresh_count == 2:
                print("Refreshing limit reached. Exiting loop.")
                break
            print("Property section not found.")
            print("Changing circuit.")
            switch_tor_circuit()
            print("Refreshing Website.")
            refresh_count += 1
            driver.refresh()
            continue

        refresh_count = 0
        try:
            tmp_current_url = driver.current_url
            switch_tor_circuit()  # Switch Tor circuit before loading the page
            time.sleep(15)  # Wait for the new circuit to be established
            next_button = driver.find_element(By.LINK_TEXT, "Next")
            next_button.click()
```

```
                random_sleep(3, 6)
                tmp_new_url = driver.current_url
                if tmp_current_url == tmp_new_url:
                    print("Same page detected. Exiting loop.")
                    break
                print(tmp_new_url)
            except NoSuchElementException:
                print("Next button not found. Exiting loop.")
                break

    except Exception as e:
        print(f"Unexpected error: {e}")
    finally:
        driver.quit()
        tor_process.kill()

    return property_info
```

In [ ]:
```
neighborhoods = [
    'Bel-Air_Los-Angeles_CA',
    'Brentwood_Los-Angeles_CA',
    'Culver-City_CA',
    'Encino_Los-Angeles_CA',
    'Mar-Vista_Los-Angeles_CA',
    'Mid-Wilshire_Los-Angeles_CA',
    'Pacific-Palisades_Los-Angeles_CA',
    'Palms_Los-Angeles_CA',
    'Playa-del-Rey_Los-Angeles_CA',
    'Playa-Vista_Los-Angeles_CA',
    'Santa-Monica_CA',
    'Sawtelle_Los-Angeles_CA',
    'Sherman-Oaks_Los-Angeles_CA',
    'Studio-City_Los-Angeles_CA',
    'Venice_CA',
    'West-Los-Angeles_CA',
    'Westwood_Los-Angeles_CA'
]
```

In [ ]:
```
for n in neighborhoods:
    results = zipcode_scrapping(n)
    # Saving the results to a pickle file for further analysis.
    save_data(results, f'results/{n}_results.pkl')
```

> In this section, the information extracted from the website is joined and cleaned. The data is then saved to a CSV file for further analysis. Some properties have a range for the number of bedrooms, bathrooms, and area. In these cases, the minimum and maximum values are extracted and stored in separate columns. The same is done for the rent. Even though the previous section extracted the information, for simplicity of development, the following conde loads the information from the pickle files, this way the code can be run without the need to scrape the website again.

In [ ]:
```
FILES_PATH = os.path.join(
    'results'
)
```

In [ ]:
```
raw = []
ngh = []

re_ngh = re.compile(r'\S+(?=_results\.pkl)')

for file in os.listdir(FILES_PATH):
    tmp_file = os.path.join(FILES_PATH, file)

    with open(tmp_file, 'rb') as f:
        tmp_info = pkl.load(f)

    raw.extend(tmp_info)
    ngh.extend(re_ngh.findall(file) * len(tmp_info))
```

In [ ]:
```
raw_soup = [BeautifulSoup(x, 'html.parser') for x in raw]
```

In [ ]:
```
rent = [
    x.find('div', {'data-testid':'card-price'}).text
    for x in raw_soup
]
```

In [ ]:
```
rent_min = [
    re.findall(r'(?<=^\$)\d+,\d+(?=$|\s\-)', x)
    for x in rent
]
rent_min = [x[0] if x else '' for x in rent_min]
rent_min = [x.replace(',', '') if x != '' else '' for x in rent_min]
rent_min = [float(x) if x != '' else np.nan for x in rent_min]
```

In [ ]:
```
rent_max = [
    re.findall(r'(?<=\s\-\s\$)\d+,\d+', x)
    for x in rent
]
rent_max = [x[0] if x else '' for x in rent_max]
```

```
rent_max = [x.replace(',', '') if x != '' else '' for x in rent_max]
rent_max = [float(x) if x != '' else np.nan for x in rent_max]
```

In [ ]:
```
n_beds = [
    x.find('li', {'data-testid':'property-meta-beds'})
    for x in raw_soup
]
n_beds = [x.text if x else '' for x in n_beds]
```

In [ ]:
```
n_beds_min = [
    re.findall(r'(?<=^)(\d+|Studio)', x)
    for x in n_beds
]
n_beds_min = [x[0] if x else np.nan for x in n_beds_min]
```

In [ ]:
```
n_beds_max = [
    re.findall(r'(?<=\-\s)\d+', x)
    for x in n_beds
]
n_beds_max = [x[0] if x else np.nan for x in n_beds_max]
```

In [ ]:
```
n_baths = [
    x.find('li', {'data-testid':'property-meta-baths'})
    for x in raw_soup
]
n_baths = [x.text if x else '' for x in n_baths]
```

In [ ]:
```
n_baths_min = [
    re.findall(r'(?<=^)\d+', x)
    for x in n_baths
]
n_baths_min = [x[0] if x else np.nan for x in n_baths_min]
n_baths_min = [float(x) if x != np.nan else np.nan for x in n_baths_min]
```

In [ ]:
```
n_baths_max = [
    re.findall(r'(?<=\-\s)\d+', x)
    for x in n_baths
]
n_baths_max = [x[0] if x else np.nan for x in n_baths_max]
n_baths_max = [float(x) if x != np.nan else np.nan for x in n_baths_max]
```

In [ ]:
```
area = [
    x.find('li', {'data-testid':'property-meta-sqft'})
    for x in raw_soup
]
area = [x.text if x else '' for x in area]
area =[
    re.findall(r'(?<=sqft).+(?=\ssquare\sfeet)', x)
    if x != '' else [] for x in area

]
area = [x[0] if x else '' for x in area]
```

In [ ]:
```
area_min = [
    re.findall(r'(?<=^)(\d{3}|\d+,\d+)', x)
    for x in area
]
area_min = [x[0] if x else '' for x in area_min]
area_min = [x.replace(',', '') if x != '' else '' for x in area_min]
area_min = [float(x) if x != '' else np.nan for x in area_min]
```

In [ ]:
```
area_max = [
    re.findall(r'(?<=\-\s)(\d{3}|\d+,\d+)', x)
    for x in area
]
area_max = [x[0] if x else '' for x in area_max]
area_max = [x.replace(',', '') if x != '' else '' for x in area_max]
area_max = [float(x) if x != '' else np.nan for x in area_max]
```

In [ ]:
```
addss_1 = [
    x.find('div', {'data-testid':'card-address-1'})
    for x in raw_soup
]
addss_1 = [x.text if x else '' for x in addss_1]
```

In [ ]:
```
addss_2 = [
    x.find('div', {'data-testid':'card-address-2'})
    for x in raw_soup
]
addss_2 = [x.text if x else '' for x in addss_2]
```

In [ ]:
```
data = pd.DataFrame({
    'neighborhood': ngh,
    'rent_min': rent_min,
    'rent_max': rent_max,
    'n_beds_min': n_beds_min,
    'n_beds_max': n_beds_max,
    'n_baths_min': n_baths_min,
    'n_baths_max': n_baths_max,
    'area_min': area_min,
```

```
    'area_max': area_max,
    'address_1': addss_1,
    'address_2': addss_2
})
```

In [ ]:
```
data.loc[data.rent_max.isna(), 'rent_max'] = data.loc[data.rent_max.isna(), 'rent_min']
```

In [ ]:
```
data.loc[data.n_beds_max.isna(), 'n_beds_max'] = data.loc[data.n_beds_max.isna(), 'n_beds_min']
```

In [ ]:
```
data.loc[data.n_baths_max.isna(), 'n_baths_max'] = data.loc[data.n_baths_max.isna(), 'n_baths_min']
```

In [ ]:
```
data.loc[data.area_max.isna(), 'area_max'] = data.loc[data.area_max.isna(), 'area_min']
```

In [ ]:
```
data.to_csv(os.path.join(FILES_PATH, '..', 'neighborhoods_around_ucla.csv'), index=False)
```

In [ ]:
```
data
```

Out[ ]:

	neighborhood	rent_min	rent_max	n_beds_min	n_beds_max	n_baths_min	n_baths_max	area_min	area_max	address_1	address_2
0	Studio-City_Los-Angeles_CA	7600.0	7600.0	3	3	3.0	3.0	2150.0	2150.0	11734 Sunshine Ter	Studio City, CA 91604
1	Studio-City_Los-Angeles_CA	1789.0	3276.0	Studio	3	1.0	2.0	426.0	1567.0	Ava Studio City	10979 Bluffside Dr, Los Angeles, CA 91604
2	Studio-City_Los-Angeles_CA	1895.0	1995.0	1	1	1.0	1.0	750.0	750.0	Arch Apartments	4151 Arch Dr, Studio City, CA 91604
3	Studio-City_Los-Angeles_CA	2695.0	2695.0	2	2	2.0	2.0	1150.0	1150.0	4418 Colfax Ave Apt 3	Studio City, CA 91602
4	Studio-City_Los-Angeles_CA	1850.0	2895.0	Studio	2	1.0	1.0	470.0	710.0	Alhambra at Studio City	10937 Fruitland Dr, Los Angeles, CA 91604
...	...	...	...	...	...	...	...	...	...	...	...
3856	Playa-del-Rey_Los-Angeles_CA	6590.0	6590.0	3	3	3.0	3.0	2332.0	2332.0	7230 W 90th St	Los Angeles, CA 90045
3857	Playa-del-Rey_Los-Angeles_CA	8000.0	8000.0	4	4	2.0	2.0	1923.0	1923.0	7301 Vista Del Mar Apt B105	Playa Del Rey, CA 90293
3858	Playa-del-Rey_Los-Angeles_CA	16500.0	16500.0	2	2	2.0	2.0	NaN	NaN	6975 Trolleyway	Playa Del Rey, CA 90293
3859	Playa-del-Rey_Los-Angeles_CA	3650.0	3650.0	1	1	1.0	1.0	785.0	785.0	8650 Gulana Ave Unit 1172	Los Angeles, CA 90293
3860	Playa-del-Rey_Los-Angeles_CA	3889.0	3889.0	2	2	2.0	2.0	1039.0	1039.0	8300 Manitoba St Apt 216	Playa Del Rey, CA 90293

3861 rows × 11 columns

In [ ]:
```
tmp = f"""
Number of properties: {data.shape[0]}
Number of neighborhoods: {data.neighborhood.nunique()}
"""
print(tmp)
```

```
Number of properties: 3861
Number of neighborhoods: 17
```

c.)

Write a short paragraph about the businesses or research that would use the data you scraped. Describe it's value and what it can be used for.

We picked `Realtor.com` as the website to web scrape. We find great purpose in our task is there are many opportunities to capitalize on the information that can be accessed from the website. To begin with, it contains real estate listings with homes for sale or rents in and specific filters based on the clients' preferences, providing insights for an informed decision-making process. Some important features of interest that can be pulled from the website are:

- Location (zip code, address)
- Price range
- Type of property (house, apartment, condo, commercial)
- Number of bedrooms and bathrooms
- Amenities (pet friendly, in-unit laundry, pool, gym, parking, etc.)

Overall, the information that can be scraped is very valuable and can be applied to many scenarios, such as:

1. **UCLA - Market and Academic Research:** The information can be used to access the details of properties in the neighboring areas near UCLA. In the context of a market research the price levels and home size can be utilized to show trends and fluctuations in pricing based on proximity to the campus. The university can provide help for informed decision making to the students who do not have much experience in renting a space, and average expectation to avoid fraudulent situations. In terms of academia, some topics of interest might be seasonality in demand and prices based on the quarter/semester school year structure, demand of a certain type or size of a home and etc.
2. **Real Estate Market Analysis:** Can aid real estate agencies and investors on the properties in demand. For example, if there are many large houses in the area and a shortage of affordable apartment buildings, which are of preference near a university, how can market opportunities be identified both for renters and investors in properties, often represented by real estate agencies.
3. **Urban Planning and Development:** This is another application that differentiates more from the previous two as the main benefit is in terms of public services, development, and infrastructure projects. Based on housing density and types, regulators can make informed decisions to upgrade the conditions in the area.

It is important to note that the information in Realtor.com is detailed and valuable for the above said uses. As their source of business, the owners of the website have used various levels of protection in order to prevent scraping from bots, which made our goal much more complex.