Assignment 2: Webscraping

Try HackMD (https://hackmd.io?utm_source=view-page&utm_medium=logo-nav)

Assignment 2: Webscraping

Out: February 14, 2024
Due: February 28, 2024

Overview

For this assignment, you'll collect some stock data. We'll make use of investing.com (http://investing.com) to collect information on the most active stocks in the market, through web scraping. We'll supplement this with historical data about these stocks gathered through API requests.

Part 0: Set Up

You'll then be responsible for cleaning the data, creating a database from it, and analyzing stocks by querying your database.

You can click here (https://classroom.github.com/a/HtRsewJg) to get the stencil code for Homework 2. Reference this guide (https://docs.google.com/document/d/1v3IQrC_0pFxsRBXsvCEzKBDAmYjzuSJCvXhkg8ewDn0/edit#heading=h.u8r26p594iqg) for more information about Github and Github Classroom. The data is located in the `data` folder. To ensure compatibility with the autograder, you should not modify the stencil unless instructed otherwise. For this assignment, please write your solutions in the respective `.py` and `.sql` files. Failing to do so may hinder with the autograder and result in a low grade.

Part 1: Web Scraping

To get started, we're going to want to collect some data on the most active stocks in the market. Conveniently, investing.com (http://investing.com) publishes this exact data. To collect this data, you'll make use of web scraping.

For the purposes of this assignment, we've made a copy of this page to keep the data static. Note, some of the data in our static copy is intentionally modified from real stock data to ensure you've cleaned your data and handled edge cases. As such, you will scrape from this URL: https://cs1951a-s21-brown.github.io/resources/stocks_scraping_2021.html (https://cs1951a-s21-brown.github.io/resources/stocks_scraping_2021.html)

Before scraping, you'll need your code to access this webpage. You should make use of the `requests` library to make an HTTP request and collect the HTML. If you're not familiar with the `requests` library, you can read about it here (https://requests.readthedocs.io/en/latest/).

Once you have accessed the HTML and assigned it to some variable, you'll want to scrape it, collecting the following for each stock in the table and storing it in a data structure.

- company name
- stock symbol
- price
- change percentage
- volume
- HQ state

You'll use Beautiful Soup, a Python package, to scrape the HTML. This will require looking at the HTML structure of the investing.com (http://investing.com) page. You can select various HTML elements on a page by tag name, class name, and/or id. Using inspect element (https://zapier.com/blog/inspect-element-tutorial/) on your web browser, you can check what HTML tags and classes contain the relevant information.

Note: You should collect information from the 50 most active stocks in the investing.com (http://investing.com) table. This is what the investing.com (http://investing.com) HTML will contain by default.

Hint: All extracted information will be strings. You'll want to make sure the price and percent change are floats (i.e., `"24.5%"` should become the float `24.5`), and volume is an integer (i.e., `1K` should be `1000`). Lastly, lower case the HQ states (i.e. `California` should be turned into `california`). The company name and stock symbols do not need to be lowercased.

Another Hint: You will probably want to look ahead to the queries you will ultimately ask in Part 4–this will affect the type of cleaning you need to do.

A webscraping example

Consider the following simple HTML page with an unordered lists:

```
<html>
    <body>
        <h1>Welcome to My Website </h1>
        <ul>
            <li>Coffee</li>
            <li>Tea</li>
            <li>Coke</li>
        </ul>
    </body>
</html>
```

Imagine we want to get the items in the list. The ul tag indicates an unordered list. We'll then want to get each list item (list items are in li tags). Specifically, we'll want to extract the text inside each list item. To do this, we'll use the following code, where page.text is the HTML of the page.

```
soup = BeautifulSoup(page.text, 'html.parser')
items = soup.find(ul).find_all("li")
```

You'll notice that `items` is a list of three items, since there are three list items in the unordered list. You'll also see that `items[0].text` will give you the text of the first list item!

Part 2: API Requests

Rather than using web scraping to collect this data like in the previous section, we’ll now make use of an API. You'll make requests to this API using Python’s `requests` library.

Normally, we will query an externally hosted API to get data about a stock. However, these API services are often not free and almost always require you to register an account. To save you the trouble (and avoid paying money), we instead implemented a simple API server and bundled it in the stencil code, in `server.py`.

To launch the server, open a separate terminal window, activate the course Python environment, ensure that you are `cd`'d into the root of the stencil folder, and run:

```
uvicorn server:app
```

This launches a server that will respond to API queries at `localhost:8000`.

After you have launched the server, you can visit localhost:8000/docs (http://localhost:8000/docs) in your browser to read the documentation about specific APIs. Note that you can only visit the documentation page if the server is running.

We’re going to want to collect two pieces of information for each stock in investing.com (http://investing.com)'s most active stock table:

- the average closing price of each of the most active stocks over the last month
- the closing price on January 20, 2023 for each of the most active stocks

To do this, read through the API documentation and find the appropriate API calls. For the average closing price, you will want to parse through the data and average the closing price for each stock.

Working on the Department Machine

If you are working from the department machine, you would not be able to visit the documentation page from the browser on your computer, since the server is running on the department machine, not your local computer. Instead, you can visit the copy of the documentation that we hosted at https://csci1951a-spring-2024.github.io/stonks-docs/ (https://csci1951a-spring-2024.github.io/stonks-docs/). The "Try It Out" buttons on the docs do not work.

Another issue is that when you SSH into the department, the department server forwards you to one of the many available machines. However, the Mock API server you spin up is only accessible on the particular machine you are logged into. Therefore, you want the terminal running the server and the terminal running your code to be running on the same machine. To ensure this happens, you can either:

1. Use a terminal multiplexer like screen (https://tldr.inbrowser.app/pages/common/screen) or tmux (https://tldr.inbrowser.app/pages/common/tmux) to get multiple shell sessions on one SSH connection. This is more involved but very valuable to learn.
2. (Recommended) Set a flag in the SSH command to explicitly log into a particular department machine, as described by Method 3 on the department SSH guide (https://cs.brown.edu/about/system/connecting/ssh/ssh.cs.brown.edu.html).
   When you SSH into the department, you should see a line like `Forwarding to cslab3b...` . In this case, `cslab3b` is the name of the department machine you are logging into. To explicitly ask the department to log into the same machine on a separate SSH connection, run:

```
# replace cslab3b with some other machine name
ssh -t <cslogin>@ssh.cs.brown.edu host=cslab3b
```

Potential Error and Fix

If you encounter the following error when running the server

```
[Errno 48] error while attempting to bind on address ('127.0.0.1', 8000): a
```

This means that there is already a process running on port 8000. It's likely that there's another instance of the server running already. In this case, you would want to kill the other process and rerun the command. If you can't find the process and shut it down manually, you can run the following commands to do kill the process:

```
lsof -i:8000 # this outputs a list of processes using port 8000
# EXAMPLE OUTPUT:

# COMMAND   PID          USER   FD   TYPE             DEVICE SIZE/OFF NODE
# Python  36825 alexanderding   14u  IPv4 0x4b08d3701b57b055      0t0  TCP

kill 36825 # replace 36825 with the PID outputted by `lsof`
```

Part 3: Ethics Components in Web Scraping

Record your responses in the `writeup.md`

1. From the previous part of this assignment, note that you are asked to scrape data from a public website. List and explain three scenarios where scraping data may pose a risk to external and internal stakeholders. Feel free to refer to this (https://www.promptcloud.com/blog/is-data-scraping-ethical/). (3 points)

2. With investing.com (http://investing.com), the stock data you have been asked to look into is public content, but many websites we use daily collect user-generated content. Consider one of the following websites (or a website of your choice) that collects user-generated content and has interesting social/ethical implications:

   - Inspire (https://www.inspire.com/): a social network for health
   - NextDoor (https://nextdoor.com/): a social network for neighborhoods
   - TripAdvisor (https://www.tripadvisor.com/): a site for finding travel deals and reviews
   - Reddit (https://www.reddit.com/): a network of online discussion communities based on people's interests
   - Venmo (https://venmo.com/): a mobile payment service with a social networking component

   Question: Research the different uses, sections, and users of one site. Search online for any social context or controversies surrounding the sites. Based on your exploration of the site, answer the following questions (7 points):

   1. What do you imagine a user might expect about the privacy of their personal data on the site? Does the expectation of privacy vary across different sections or types of data on the site?
   2. Identify one vulnerable user group for whom data scientists scraping data should exercise more caution.
   3. Describe one project or use case where scraping data from this site could cause harm. Describe one project or use case where scraping data from this site could be beneficial.
   4. What types of data on the site should data scientists be especially cautious about collecting? If any, what types of data on the site should not be scraped by data scientists?
      a) Consider the creator of the data, specific content of the data, user expectations of privacy, and biases that shape the data.

b) Data types could be specific, like usernames and dates, or general, like posts about a certain subject or posts from certain user groups.

3. Let's think about missing data for a second. It's important to remember that the scraped data isn't all encompassing. View this artwork by Mimi Onuha: here (https://mimionuoha.com/the-library-of-missing-datasets). What kind of data may be difficult to get access to? Give one example. Why do you think that is? What is the impact of this missing or inaccessible data? (3 points)

Part 4: Databases

Record your responses in `writeup.md`

1. Read the article here (https://www.technologyreview.com/2020/12/03/1012797/fair-value-fixing-the-data-economy/)

   Question: The concept of a "data economy" comes with the perspective that data is a form of economic resource and capital. To what, if any, standards of social responsibility should we hold researchers, companies, and other entities that use data to turn a profit? What price would you put on your own personal data, if any? Justify your opinion and describe which opinions you agree or disagree with in the article. (2 points)

You now realize that to truly harness the data, you need to turn it into a database you can query. Using the provided stencil, create a database with these tables:

Companies

- `symbol` : a string of the stock symbol that is the primary key of this table
- `name` : a string of the company name
- `location` : a string of the company's HQ in lower case

Quotes

- `symbol` : a string of the stock symbol that is the primary key of this table
- `close` : the closing price of the stock on Jan 20th, 2023, a number
- `price` : the current stock price, a number
- `avg_price` : the average closing price over the last month, a number
- `volume` : the volume of this stock as a number
- `change_pct` : the percent change in the stock's price today, as a decimal

Note: we expect you to use good judgment when designing the database and set columns as primary keys and/or foreign keys as appropriate. Each table should have a primary key, and quotes should reference the company through the symbol column using a foreign key (but not the reverse). You might find the fact that a column can be both the primary key and a foreign key useful.

Working with Databases in Python

To create a connection to the database, and a cursor, we include the following lines in the stencil:

Create connection to database

```
conn = sqlite3.connect('data.db')
c = conn.cursor()
```

We also prepare the database for you by clearing out relevant tables if they already exist. This allows you to run your code multiple times and replace your old version of data.

```
c.execute('DROP TABLE IF EXISTS "companies";')
c.execute('DROP TABLE IF EXISTS "quotes";')
```

To create a database table, you'd do something like this:

```
c.execute('CREATE TABLE person(person_id int not null, name text')
conn.commit()
```

To insert a row into a table, you'd do something like this:

```
c.execute('INSERT INTO person VALUES (?, ?)', (some_variable, another_varial
```

The data is saved into the data.db file. If you want to take a look at it, one way is to use a website such as https://inloop.github.io/sqlite-viewer/ (https://inloop.github.io/sqlite-viewer/) where you can load the file and query data from the tables.

Part 5: Queries

Each SQL statement should be stored in its own file: `query1.sql`, `query2.sql`, etc.

1. Write a SQL statement to return the symbol and name of the stock with the biggest percent gain relative to its monthy average price. This should be calculated as the current price divided by the average price.
2. Write a SQL statement to return the name of the stock with the highest price of all stocks whose closing price on January 20th, 2023 is greater than their monthly average
3. Write a SQL statement to return the symbol and name of all stocks with prices above $30 and where the absolute difference between the current price and monthy average price is less than $10. Your results should be sorted by the absolute difference between current price and monthly average price, in ascending order.
4. Write a SQL statement to return each state and number of companies headquartered in that state (no need to include states not in your dataset). Order alphabetically by location (A -> Z).

Handing In

After finishing the assignment (and any assignment in the future), run `python3 zip_assignment.py` in the command line from your assignment directory, and fix any issues brought up by the script.

After the script has been run successfully, you should find the file `sql-submission-1951A.zip` in your assignment directory. Please submit this zip file on Gradescope under the respective assignment. (If you have not signed up for Gradescope already, please refer to this guide (https://docs.google.com/document/d/1X_SAAVeGNcZW9HbaM-ev8h9RrhLWtlTSYfMkE8jWVdQ/edit#heading=h.lnyy1apmgq9k)).

Credits

Made with ❤ by Jacob Meltzer and Tanvir Shahriar (2019 TAs), updated by Natalie Delworth and Nazem Aldroubi (2020 TAs); Daniela Haidar, Daniel Civita Ramirez, JP Champa and Nam Do (2021 Spring TAs), and again by Aakansha Mathur and Nam Do (2021 Summer TAs).Updated by Daniela Haidar, Benjamin Shih, James Shi, Micah Bruning, Aakansha Mathur in Spring 2022. STA component was updated by Aanchal Seth and Joanna Tasmin in Spring 2022. Updated by Livia Gimenes, Naphat Permpredanun and Filip Aleksic (Spring 2023 TAs). Updated by Kyle Lee, Alex Ding, Atif Khan, Amaris Grondin (Spring 2024 TAs).