Web Scraping and Github API

Patrick Lynch

11/02/2023

The objective of the first midterm was to obtain information about Github users from a webpage, use the Github API to ascertain more information about these users, and then use Python to visualize the received data. By using libraries like 'Pandas,' 'Beautiful Soup,' and 'Requests' I was able to successfully scrape user information on both CharcoalPaper.com and the GitHub API. I took this information and turned it into data visualizations using the 'Matplotlib' library.

Part one of the project required us to scrape the provided URL for some user information. I began by importing the 'Pandas' (pd), 'Requests', and 'Beautiful Soup' (bs4) libraries and making a GET request to the target URL. The program then used bs4 to parse the content and search for the 'Github Data' heading using 'soup.find()'. Once the heading was located, the program searched for all sibling 'div' elements. These contain user data, which were stored in the 'main_data_rows' variable. From here, an empty list was created to store the extracted data; the website was scraped for user information using the 'div' elements. I added this to the 'data' list created before. Still, only after it passed validation checks, including having a non-negative follower count, a valid membership start data, the correct spelling of login ID, and the data row should contain the expected number of data points. After the data had been extracted into a list, 'Pandas' was used to convert that list into a data frame named 'df.' This data frame has columns for 'Login ID," "Repo Count," "Follower Count," and "Member Since." After the data populated the data frame, duplicate entries and invalid or missing login IDs were removed. Finally, the data was saved into a CSV file called “github_users” to be used for parts two and three.

Part two of the project involved using the GitHub API to obtain more user information. I began by importing the time module so that a delay could be added to the requests to the GitHub API. I then created a Github personal access token so that I could send more requests per hour to the API. This was put into a TXT file and was not committed to the GitHub repository. Instead, a .gitignore file was used. Following the implementation of the personal access token, two new lists were created, one for extracted user data and one for failed login IDs. These were called ‘data’ and ‘failed_logins.’The program then begins making API requests, where a failure is indicated by a response status code of 200. These failures are logged in the failed_logins list, whereas all successful logins are logged in data. A one-second delay was added to avoid making too many requests to the GitHub Server. Once all of the requests have been made, the program then goes back and removes the failed login IDs from the “github_users” file initially created. The new information was saved into a file called “github_user_details.”

Part three of the project involved taking the information we collected from parts one and two and creating scatterplots with it. To do this, I made a separate file and imported the ‘Matplotlib’ and ‘Pandas’ libraries. This new program takes information from the CSV files we created before and puts the information into a Pandas data frame. Columns of data were then extracted to be used in the plots being created. The following sequence of code set parameters for the scatterplot. This includes labeling the axis, titling the graph, setting the graph size, and telling the diagram which data is being used. The plt.show() command will display the chart, and the plt.savefig() command saves it to the directory. Summary statistics for both CSV files can be viewed in the Python console.

Some interesting information from the summary statistics of the data includes an average repository count of 133.75 and a standard deviation of 482.29. Additionally, the average follower count was around 383.01, with a standard deviation of 1170.34. The average following per user was 3448.80, with a standard deviation of 18836.43. Finally, the average starred count was 24.94 with a standard deviation of 10.02.

First, I plotted the repository count against the follower count. I expected a positive relationship between these two variables—however, most of the data clusters near the bottom left of the graph. There were also many outliers, mainly where people had many followers with a relatively average number of repositories. I then plotted the follower count versus the starred count. Again, I expected a positive relationship between these two variables. However, after observing the graph, there is little to no relationship here. The bottom 25% of the starred count is still the maximum of 30.

I want to take this last paragraph to address the use of ChatGPT in the coding of this project. ChatGPT was used to help understand and utilize the 'Matplotlib' library, as well as use it. This wrote lines 19 and 20 in Scatterplot.py. Additionally, lines 63 and 64 in Midterm1.py were also written by ChatGPT to assist in removing negative follower counts and invalid membership dates.