CS-579 Online Social Network Analysis

Project-1 Report

On

Social Media Data Analysis

By

Name of Student	CWID
1. Manavkumar Patel	A20543097
2. Priyanka Basani	A20513566

Under The Supervision
of
Prof. Dr. Cindy Hood

College Of Computing
Illinois Institute of Technology
Chicago, Illinois.

February 2024

CONTENTS

SECTION I PROJECT DETAILS

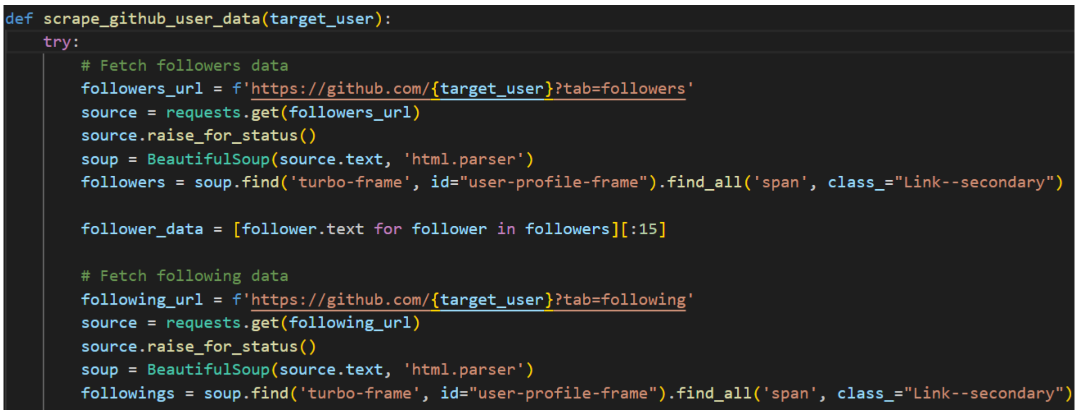
1.1 Project Statement

The project's goal is to gather data from social media platforms, analyze it, and provide some analysis on it. We have selected GitHub as our social networking platform for our project. With the use of web scraping, we have gathered the data from GitHub to build a social network. We used a buddy network with between 100 and 500 nodes to create this social network. We then used Gephi to visualize the network as a graph and plotted network metrics including degree distribution, betweenness centrality, closeness centrality distribution, PageRank, and clustering coefficient.
https://github.com/manav347/Osna-project

SECTION II DATA COLLECTION

2.1 Data Source

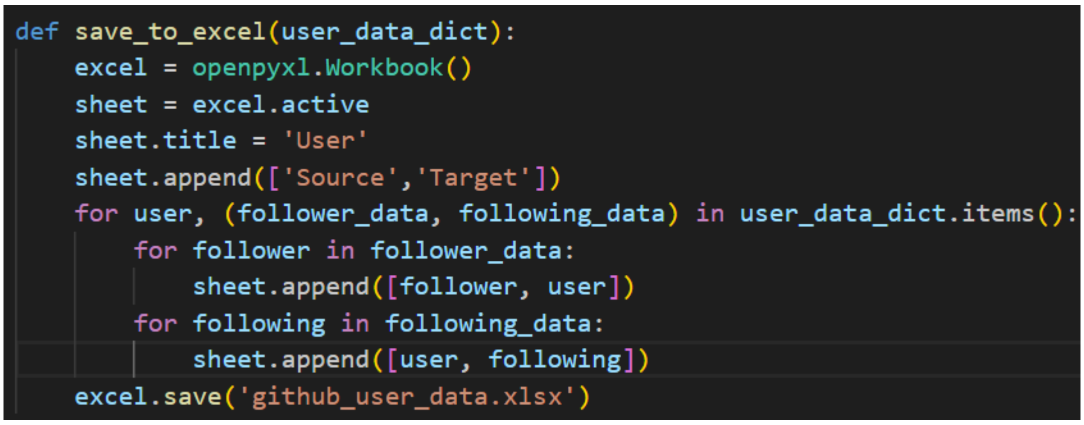
The data for this project is sourced from GitHub user profiles using web scraping techniques. Web scraping involves programmatically extracting data from web pages, bypassing the need for an API. Specifically, we utilize the BeautifulSoup library for parsing HTML and extracting pertinent details from GitHub user profiles. Additionally, we leverage the requests module to send HTTP requests to the GitHub website, enabling retrieval of HTML content from targeted pages. Subsequently, we employ BeautifulSoup once more to parse this HTML content and extract key information, including data related to followers and users being followed by a specified GitHub user.



```python
def scrape_github_user_data(target_user):
    try:
        # Fetch followers data
        followers_url = f'https://github.com/{target_user}?tab=followers'
        source = requests.get(followers_url)
        source.raise_for_status()
        soup = BeautifulSoup(source.text, 'html.parser')
        followers = soup.find('turbo-frame', id="user-profile-frame").find_all('span', class_="Link--secondary")

        follower_data = [follower.text for follower in followers][:15]

        # Fetch following data
        following_url = f'https://github.com/{target_user}?tab=following'
        source = requests.get(following_url)
        source.raise_for_status()
        soup = BeautifulSoup(source.text, 'html.parser')
        followings = soup.find('turbo-frame', id="user-profile-frame").find_all('span', class_="Link--secondary")
```

Figure 1: GitHub User Data Scraper: Extracting Followers and Followings Dynamically

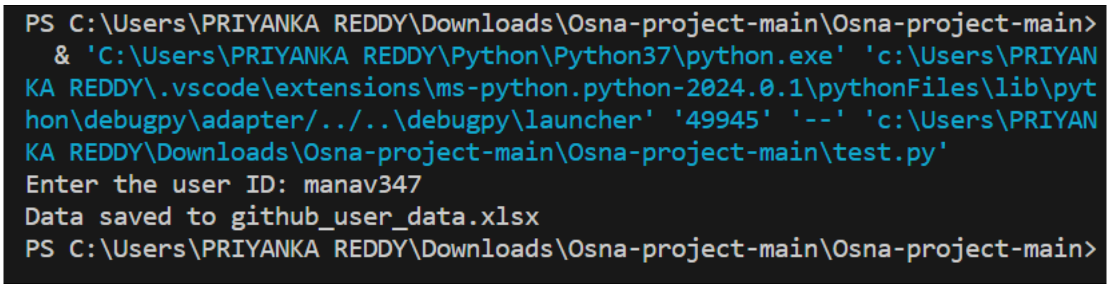
The save_to_excel function creates a new Excel workbook and adds scraped user data into it. It organizes the data with headers for "Source" and "Target" columns, then iterates through the user data dictionary, appending rows for each user's followers and followings. Finally, it saves the workbook as "github_user_data.xlsx".



```python
def save_to_excel(user_data_dict):
    excel = openpyxl.Workbook()
    sheet = excel.active
    sheet.title = 'User'
    sheet.append(['Source','Target'])
    for user, (follower_data, following_data) in user_data_dict.items():
        for follower in follower_data:
            sheet.append([follower, user])
        for following in following_data:
            sheet.append([user, following])
    excel.save('github_user_data.xlsx')
```

Figure 2: Excel Data Organizer: Saving GitHub User Network Relationships

"The user ID 'manav347' was manually selected by the user for analysis, allowing us to focus on a specific user's data. Subsequently, the system statistically fetched the top 15 followers associated with this user ID. Manually selecting user IDs allows for easier tracking and control over the data being used. It's a common practice when working with small-scale or experimental datasets. However, it's essential to remember that in real-world applications, user IDs are typically assigned dynamically or through some automated process, such as user registration. This automated process ensures that the most influential or active followers are included in our dataset without any manual selection bias."



```
PS C:\Users\PRIYANKA REDDY\Downloads\Osna-project-main\Osna-project-main>
  & 'C:\Users\PRIYANKA REDDY\Python\Python37\python.exe' 'c:\Users\PRIYAN
KA REDDY\.vscode\extensions\ms-python.python-2024.0.1\pythonFiles\lib\pyt
hon\debugpy\adapter/../..\debugpy\launcher' '49945' '--' 'c:\Users\PRIYAN
KA REDDY\Downloads\Osna-project-main\Osna-project-main\test.py'
Enter the user ID: manav347
Data saved to github_user_data.xlsx
PS C:\Users\PRIYANKA REDDY\Downloads\Osna-project-main\Osna-project-main>
```

Figure 3: GitHub User Data Extraction and Storage Automation Successful

2.2 Data Storing

Data storing refers to the process of saving information retrieved from GitHub user profiles into an Excel file. This allows for convenient storage and further analysis of the data. Data storing is implemented to automate the organization and storage of follower and following data obtained through web scraping.

Data Retrieval and Excel File Creation:

The script initiates data retrieval by utilizing web scraping techniques to gather information about the followers and followings of a specified GitHub user ID. Upon successfully retrieving this data, the script proceeds to create a new Excel workbook. This workbook serves as a structured container for storing the extracted information in an organized manner. To accomplish this, the script utilizes the openpyxl library, a powerful tool for working with

Excel files in Python. It leverages the capabilities of openpyxl to create a new workbook instance.

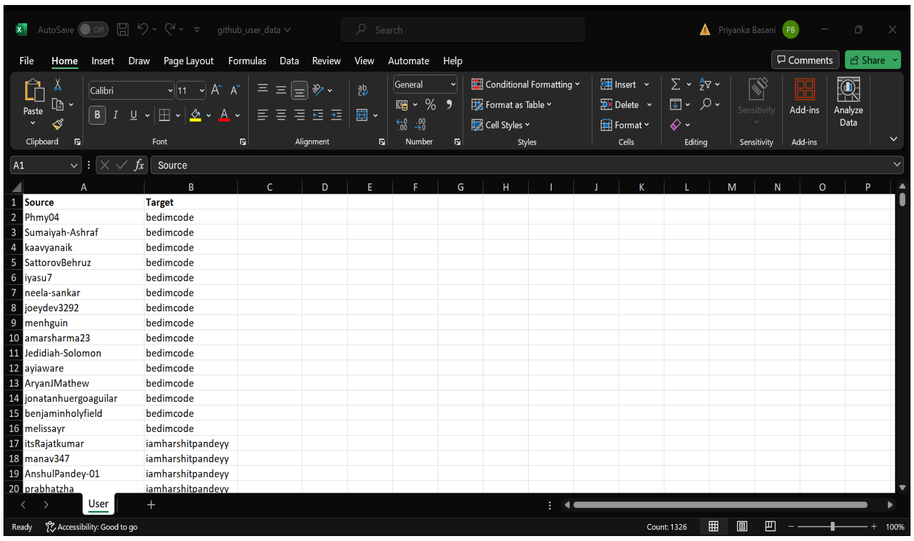
Data Storage Process:

Once the Excel workbook is created, the script allocates a new worksheet within the workbook, assigning it the title 'github_user_data'. This worksheet will serve as the primary storage space for the retrieved follower and following data. As the script retrieves data about followers and followings, it systematically appends this information to the 'github_user_data' worksheet. Each piece of data is carefully organized into rows, with each row representing a distinct relationship between a source user (follower) and a target user (following). By directly appending the data to the worksheet as it is retrieved, the script ensures that the information remains organized and readily accessible for subsequent analysis.

Saving to Excel:

Once all the relevant follower and following data has been appended to the 'github_user_data' worksheet, the script finalizes the data storage process by saving the populated Excel workbook to a file named 'github_user_data.xlsx'.

This file serves as a comprehensive repository of organized data pertaining to the follower and following relationships within the GitHub user network. It provides a structured overview of the connections between users, facilitating further analysis and exploration.



	A	B
1	Source	Target
2	Phmy04	bedimcode
3	Sumaiyah-Ashraf	bedimcode
4	kaavyanaik	bedimcode
5	SattorovBehruz	bedimcode
6	iyasu7	bedimcode
7	neela-sankar	bedimcode
8	joeydev3292	bedimcode
9	menhguin	bedimcode
10	amarsharma23	bedimcode
11	Jedidiah-Solomon	bedimcode
12	ayiaware	bedimcode
13	AryanJMathew	bedimcode
14	jonatanhuergoaguilar	bedimcode
15	benjaminholyfield	bedimcode
16	melissayr	bedimcode
17	itsRajatkumar	iamharshitpandeyy
18	manav347	iamharshitpandeyy
19	AnshulPandey-01	iamharshitpandeyy
20	prabhatzha	iamharshitpandeyy

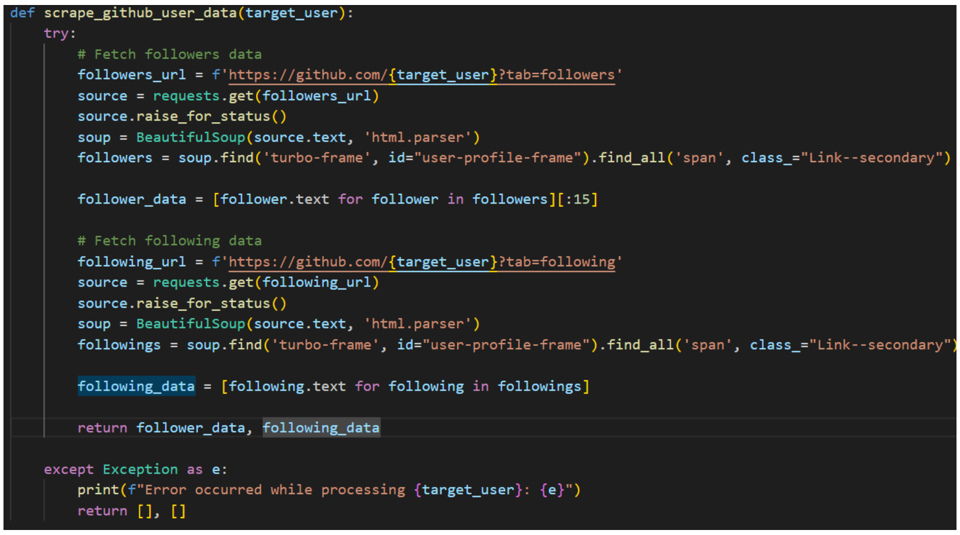
Figure 4: GitHub Users Data Excel sh

2.3 Data Processing

The extracted GitHub user data was obtained in a clean and ready-to-use format, requiring minimal preprocessing. Due to the nature of the data extraction process and the inherent structure of GitHub user profiles, no significant cleaning, transformation, or integration steps were necessary. The data was collected directly from the source in a standardized format, eliminating the need for additional preprocessing efforts. As a result, the extracted data is immediately suitable for analysis, saving time and resources typically associated with preprocessing tasks.

Data Integrity

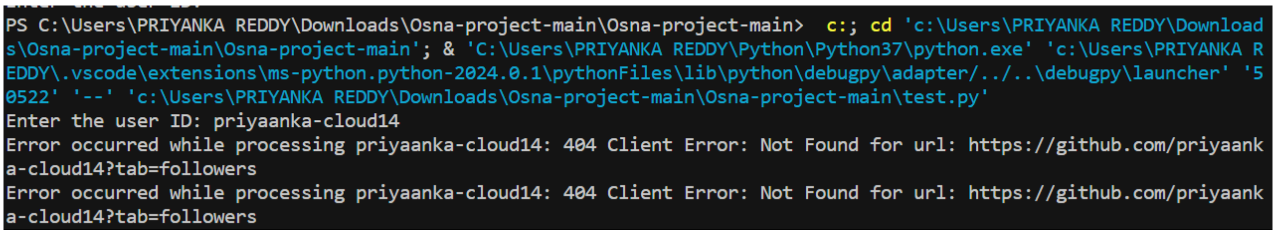
We have Ensured the integrity of the extracted data by handling exceptions during the scraping process. If an error occurs while fetching data for a particular user, the code continues execution and prints an error message without interrupting the process.



```python
def scrape_github_user_data(target_user):
    try:
        # Fetch followers data
        followers_url = f'https://github.com/{target_user}?tab=followers'
        source = requests.get(followers_url)
        source.raise_for_status()
        soup = BeautifulSoup(source.text, 'html.parser')
        followers = soup.find('turbo-frame', id="user-profile-frame").find_all('span', class_="Link--secondary")

        follower_data = [follower.text for follower in followers][:15]

        # Fetch following data
        following_url = f'https://github.com/{target_user}?tab=following'
        source = requests.get(following_url)
        source.raise_for_status()
        soup = BeautifulSoup(source.text, 'html.parser')
        followings = soup.find('turbo-frame', id="user-profile-frame").find_all('span', class_="Link--secondary")

        following_data = [following.text for following in followings]

        return follower_data, following_data

    except Exception as e:
        print(f"Error occurred while processing {target_user}: {e}")
        return [], []
```

Figure 5: Error Handling

The try-except block in the code safeguards the scraping process from errors like network issues or parsing failures. If an error occurs, the code gracefully handles it by printing an error message and proceeds with the execution. This approach ensures that even if errors occur for certain users, the scraping process continues uninterrupted for others, contributing to the overall robustness and integrity of the data extraction process.



```
PS C:\Users\PRIYANKA REDDY\Downloads\Osna-project-main\Osna-project-main>  c:; cd 'c:\Users\PRIYANKA REDDY\Download
s\Osna-project-main\Osna-project-main'; & 'C:\Users\PRIYANKA REDDY\Python\Python37\python.exe' 'c:\Users\PRIYANKA R
EDDY\.vscode\extensions\ms-python.python-2024.0.1\pythonFiles\lib\python\debugpy\adapter/../..\debugpy\launcher' '5
0522' '--' 'c:\Users\PRIYANKA REDDY\Downloads\Osna-project-main\Osna-project-main\test.py'
Enter the user ID: priyaanka-cloud14
Error occurred while processing priyaanka-cloud14: 404 Client Error: Not Found for url: https://github.com/priyaank
a-cloud14?tab=followers
Error occurred while processing priyaanka-cloud14: 404 Client Error: Not Found for url: https://github.com/priyaank
a-cloud14?tab=followers
```

Figure 6: Example for error handling

SECTION III DATA VISUALIZATION

3.1 Creation of Node

To generate nodes, we've gathered the username of every follower, which constitutes the nodes created earlier. Additionally, we've included the followers of followers in the data variable. Our limit for nodes is set at a maximum of 500.

3.2 Creation of Edges

After the nodes are generated, the subsequent stage involves establishing edges between all nodes among my followers and the followers of followers. The notion of "Source" and "Target" edges is employed. We've created edges among all nodes generated in the preceding step. The output displays all pairs of nodes that share an edge.

3.3 Visualization Using Different Graphs

❖ ***Fruchterman reingold layout***

Once the above nodes and edges among all the nodes have been established, we have experimented with different types of graphs available on the Gephi software.

- Within Gephi, we selected the Fruchterman-Reingold layout algorithm from the available layout options. This algorithm was chosen for its effectiveness in creating visually appealing graph layouts while effectively representing the relationships between nodes.
- Prior to applying the Fruchterman-Reingold layout, we prepared our GitHub data by extracting information about users and their connections. This data included the source and target nodes representing relationships between GitHub users.
- The Fruchterman-Reingold layout algorithm was executed within Gephi to compute the positions of nodes based on the specified parameters. Gephi iteratively adjusted the positions of nodes until a stable layout was achieved, minimizing the total energy of the system.

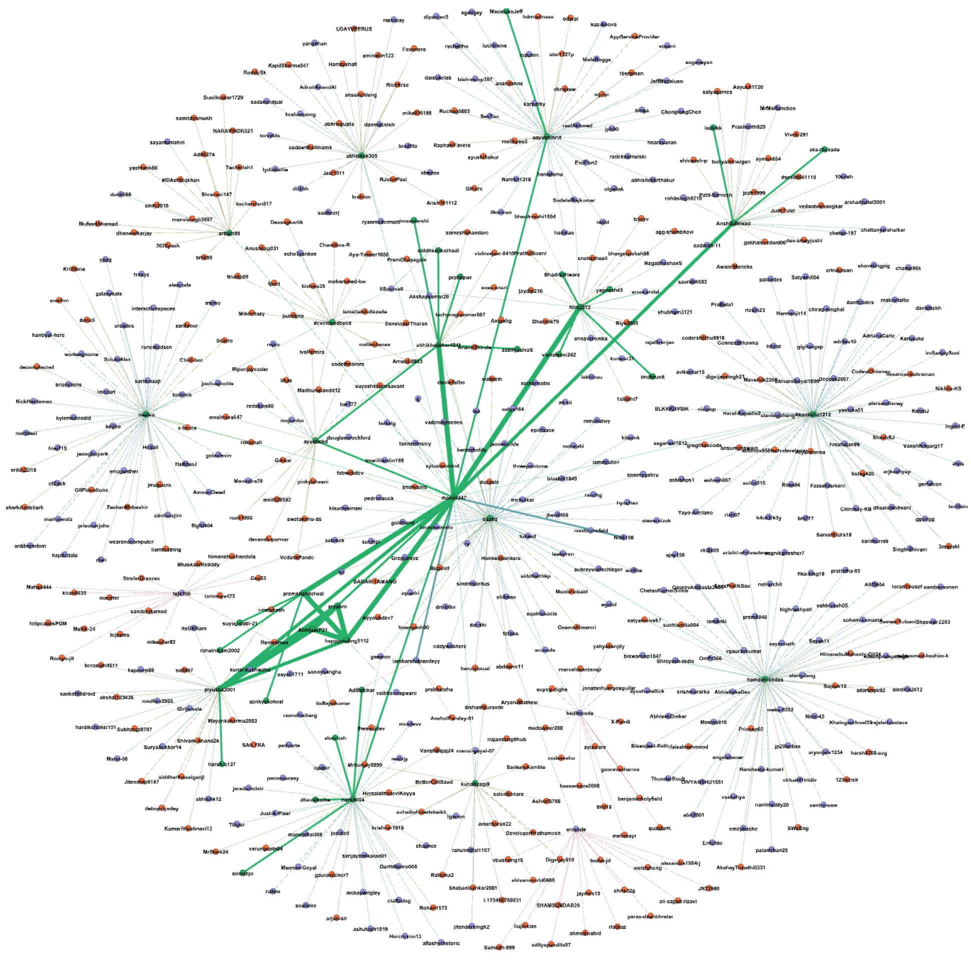


Figure 7: Fruchterman reingold layout

Insights and Interpretations from the Graph Visualization:

- Detection of Community: Through the application of the Fruchterman-Reingold layout in Gephi to visualise the graph, we were able to find discrete communities or clusters within the GitHub network. These clusters show user groups that are highly connected to one another, which may point to joint initiatives or common interests.

- Influential Users: We were able to recognize nodes with high connection levels thanks to the visualization, which pointed to individuals who are important members of the GitHub community. These well-known users might serve as centers or links between other users, encouraging interaction and cooperation.

- Structural Patterns: The graph's arrangement showed the presence of cliques, bridge nodes, and centralized or decentralized structures. Gaining an understanding of these structural patterns helped to explain the dynamics and structure of the GitHub network.

- Anomalies or Outliers: We may also be able to spot solitary nodes or outliers in the graph, which might be users with unusual roles or habits. Further examination of these anomalies may reveal intriguing trends or irregularities among the GitHub community.

We were able to effectively visualize and analyze the GitHub project data using the Fruchterman-Reingold layout algorithm in Gephi, enabling us to uncover valuable insights and inform decision-making processes within our project.

❖ ***Yifan Hu Layout***

The next graph we experimented with the Gephi was Yifan Hu layout. By leveraging the Yifan Hu layout algorithm in Gephi, we gained valuable insights into the structure, dynamics, and community organization of the GitHub network. The visualization facilitated the exploration and interpretation of complex network relationships, supporting our analysis and decision-making processes.

Insights from Yifan Hu Layout Visualization

Cluster Identification: GitHub users were efficiently classified into visually distinguishable clusters by the Yifan Hu layout method, showing cohesive groupings with strong relationships. These clusters most likely indicate user communities working together on related projects or with comparable interests. The identification and examination of community structures inside the GitHub network was made easier by the distinct cluster visualization.

Community Structure: By looking at the arrangement, we were able to identify the GitHub network's general structure, which included the presence of areas that were closely connected to one another and the boundaries that separated other communities. Gaining an understanding of the community structure helped to understand how users engage with one another and organize themselves inside the GitHub ecosystem. This data is useful for recognizing important communities, comprehending their traits, and evaluating their influence on the network.

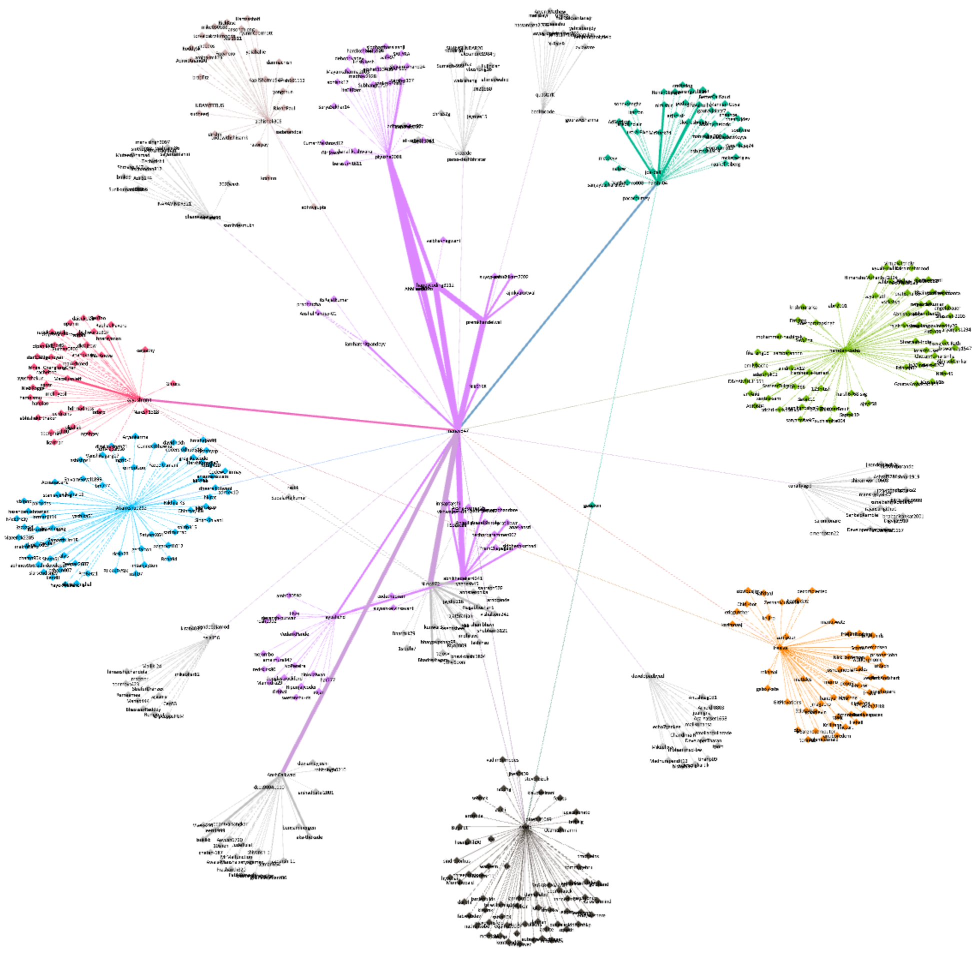


Figure 8: Yifan Hu Layout

Yifan Hu layout visualization in Gephi provided comprehensive insights into the GitHub network's structure, community dynamics, influential nodes, and anomalies. This information is invaluable for decision-making, analysis, and strategic planning within the GitHub project, helping to optimize collaboration, foster innovation, and enhance community engagement.

Choice of Gephi over Other Software Tools

Visualization Capabilities: Gephi offers advanced visualization features and a user-friendly interface specifically designed for graph analysis and visualization. Its interactive visualization tools, customizable layouts, and wide range of visualization options make it a preferred choice for exploring complex network data.

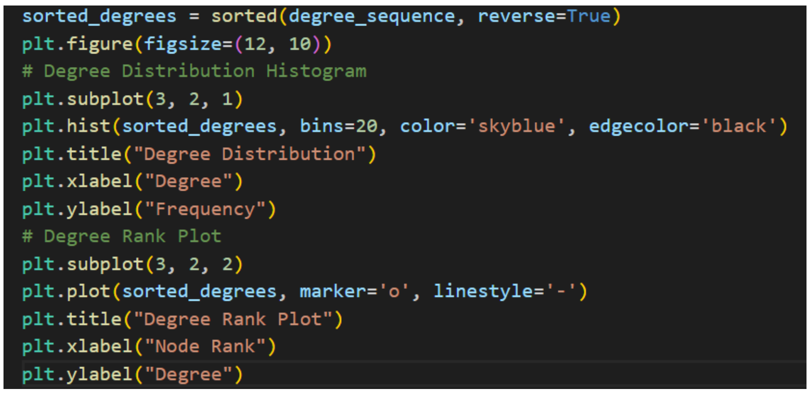
Community Support: Gephi has a large and active user community, providing access to a wealth of resources, tutorials, and plugins. This community support enhances the usability and effectiveness of the tool for graph visualization and analysis tasks.

Compatibility: Gephi supports various data formats and import options, allowing seamless integration with external data sources and facilitating data analysis workflows. Its compatibility with common graph file formats makes it easy to import and visualize data generated from different sources.

SECTION IV NETWORK MEASURES

Network measures are quantitative metrics used to analyze and characterize the structure, dynamics, and properties of complex networks, such as social networks, biological networks, and communication networks. These measures provide insights into various aspects of network behavior, including connectivity patterns, centrality of nodes, robustness, and information flow.
In NetworkX, we used degree() method to get the degree of each node in the graph. Then collected these degrees and create a histogram to visualize the distribution.

Degree Distribution:



```
sorted_degrees = sorted(degree_sequence, reverse=True)
plt.figure(figsize=(12, 10))
# Degree Distribution Histogram
plt.subplot(3, 2, 1)
plt.hist(sorted_degrees, bins=20, color='skyblue', edgecolor='black')
plt.title("Degree Distribution")
plt.xlabel("Degree")
plt.ylabel("Frequency")
# Degree Rank Plot
plt.subplot(3, 2, 2)
plt.plot(sorted_degrees, marker='o', linestyle='-')
plt.title("Degree Rank Plot")
plt.xlabel("Node Rank")
plt.ylabel("Degree")
```

- The degree of a node in a network is the number of connections it has to other nodes. The degree distribution plot shows how many nodes have a particular degree.
- The "Degree Rank Plot" is a scatter plot that ranks nodes by their degree. It appears to show a few nodes with very high degrees (possibly 'hubs' in a network) and many nodes with low degrees. This is characteristic of scale-free networks, where the principle of "the rich get richer" often applies.
- The "Degree Distribution" histogram indicates that a vast majority of nodes have a low degree (close to 0), with very few nodes having higher degrees. This distribution suggests that the network is likely to be inhomogeneous, with most nodes having only a few connections and a small number of nodes having many connections, possibly indicating a network with a few highly connected hubs.

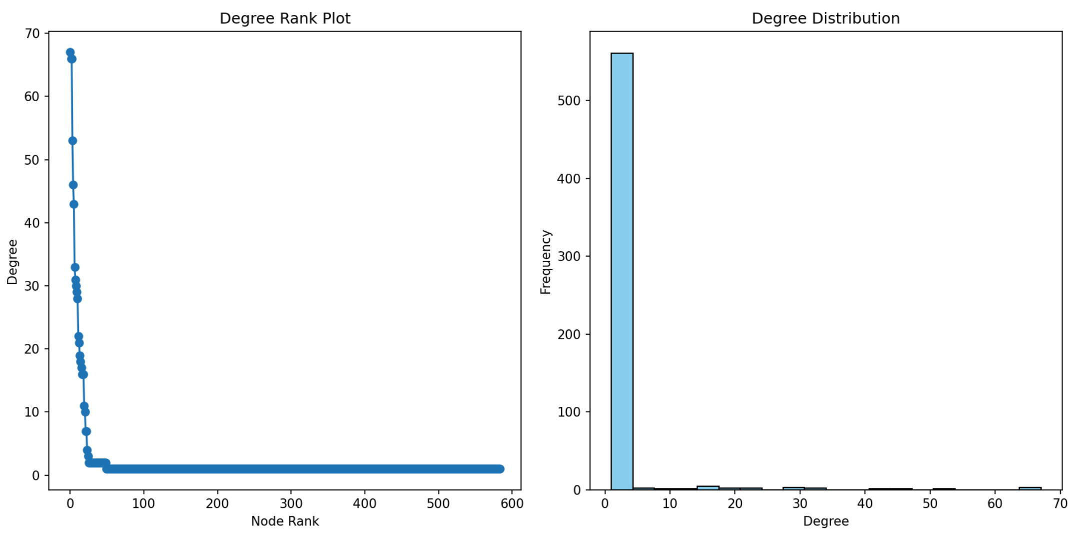
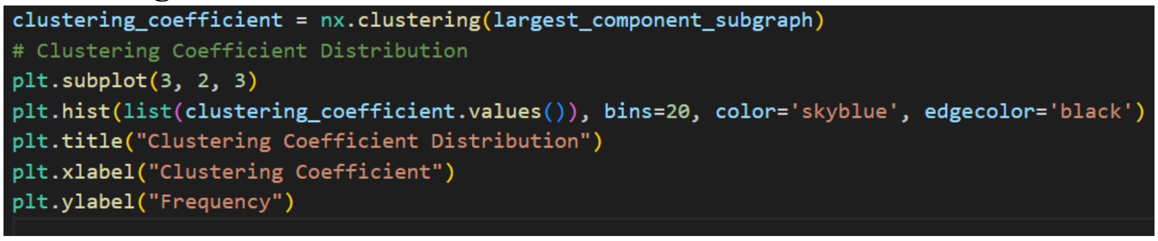


Figure 9: Degree Distribution Plot

Clustering Coefficient:



```
clustering_coefficient = nx.clustering(largest_component_subgraph)
# Clustering Coefficient Distribution
plt.subplot(3, 2, 3)
plt.hist(list(clustering_coefficient.values()), bins=20, color='skyblue', edgecolor='black')
plt.title("Clustering Coefficient Distribution")
plt.xlabel("Clustering Coefficient")
plt.ylabel("Frequency")
```

- The clustering coefficient measures the likelihood that two neighbors of a node are also neighbors of each other. It is a measure of the degree to which nodes in a network tend to cluster together.
- The histogram indicates that the majority of nodes have a clustering coefficient of 0, suggesting that for most nodes, their neighbors are not connected to each other. A low clustering coefficient for most nodes might imply a lack of tightly-knit communities within the network or that the network is not locally dense.

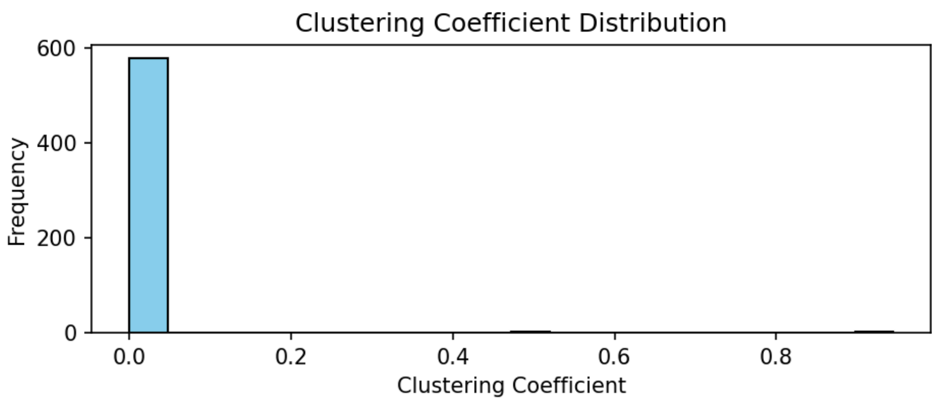
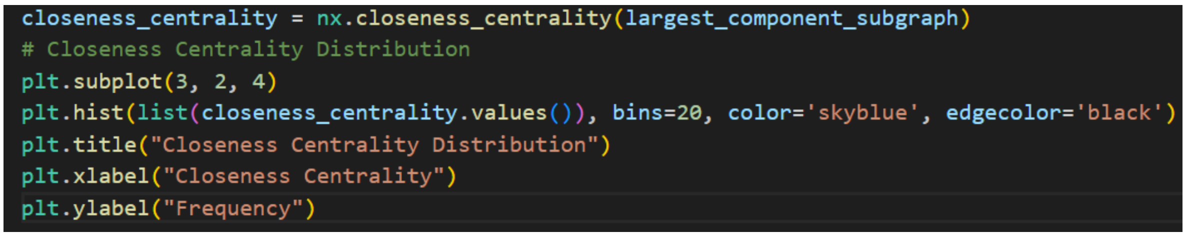


Figure 10: Clustering Coefficient Plot

Closeness Centrality Distribution:



```python
closeness_centrality = nx.closeness_centrality(largest_component_subgraph)
# Closeness Centrality Distribution
plt.subplot(3, 2, 4)
plt.hist(list(closeness_centrality.values()), bins=20, color='skyblue', edgecolor='black')
plt.title("Closeness Centrality Distribution")
plt.xlabel("Closeness Centrality")
plt.ylabel("Frequency")
```

- Closeness centrality measures how close a node is to all other nodes in the network. A higher closeness centrality indicates that a node can reach other nodes more quickly within the network.
- The histogram shows a distribution with a peak at a low closeness centrality value, suggesting that most nodes are not very close to other nodes, which could indicate that the network has a sparse connectivity or that it is large in size with nodes spread out. The smaller peaks at higher values of closeness centrality could indicate the presence of a few nodes that are more central within the network structure.

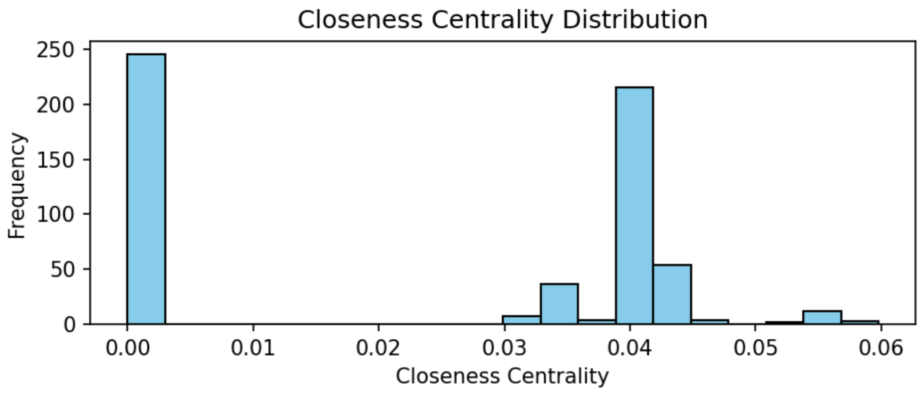
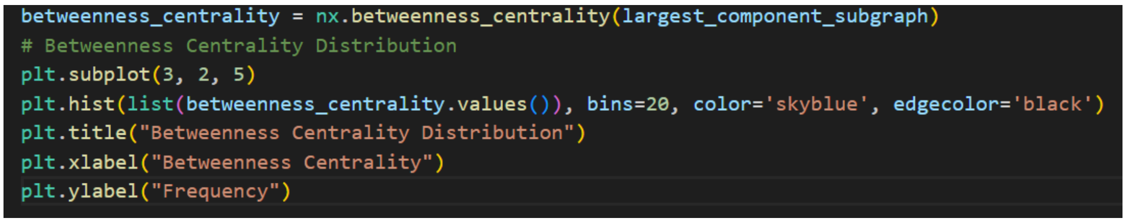


Figure 11: Closeness Centrality Distribution plot

Betweenness Centrality Distribution:



```python
betweenness_centrality = nx.betweenness_centrality(largest_component_subgraph)
# Betweenness Centrality Distribution
plt.subplot(3, 2, 5)
plt.hist(list(betweenness_centrality.values()), bins=20, color='skyblue', edgecolor='black')
plt.title("Betweenness Centrality Distribution")
plt.xlabel("Betweenness Centrality")
plt.ylabel("Frequency")
```

- Betweenness centrality measures the extent to which a node lies on paths between other nodes. Nodes with high betweenness centrality control the flow of information since many shortest paths pass through them.
- The histogram suggests that most nodes have a very low betweenness centrality, indicating that they do not often act as bridges between other nodes. This could mean that the network does not have many choke points for information flow and that the network might have multiple pathways linking

nodes.

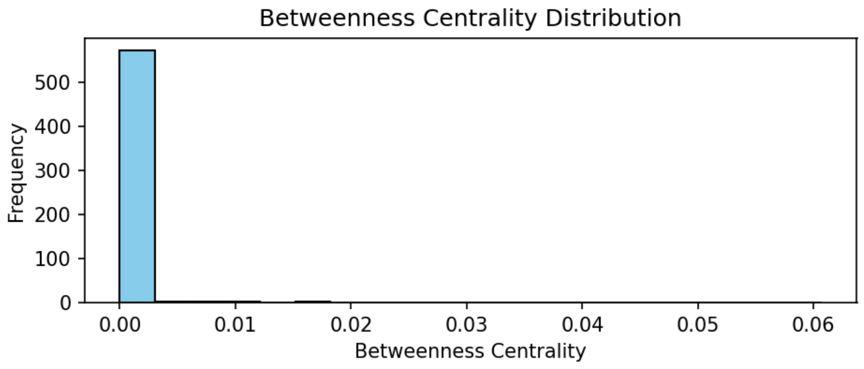


Figure 12: Betweenness Centrality Distribution plot

Regarding connections that are strong or weak

The presence of both strong and weak connections in a network is a phenomenon often described by the strength of weak ties theory. In many networks, especially social ones, weak ties play a crucial role in bridging different communities and facilitating the flow of information to parts of the network that might otherwise be disconnected. They can be particularly important for spreading novel information or for connecting individuals to new job opportunities.

However, a predominance of weak ties might indicate a vulnerability in the network's structure, as weak ties are less reliable and may not contribute to strong community structures. If a network relies heavily on weak ties, it could be prone to fragmentation if those weak ties are severed.

So, whether the prevalence of weak ties is a problem depends on the context of the network. For robustness and the spread of information, weak ties can be beneficial, but for strong community structures and resilience, a higher proportion of strong ties might be preferable.

The reasons to choose these measures over others:

- Network Size and Complexity: Measures like diameter can become less meaningful in large and complex networks where local properties (like clustering) might be more informative
- Computational Efficiency: Calculating the diameter of a large network or applying the PageRank algorithm can be computationally intensive. In contrast, the chosen measures can be computed relatively easily for each node.
- Interpretability: The chosen measures are often easier for stakeholders to understand and interpret, particularly in social networks where concepts like 'closeness' and 'betweenness' can be related to real-world scenarios.

- Relevance to the Research Question: If the research is not focused on the global properties of the network (like diameter) or the influence of nodes based on their link structure (like PageRank), then these measures might be more relevant.

  Each network measure provides unique information, and the choice largely depends on the focus of the analysis. In certain cases, global measures like diameter or sophisticated ranking algorithms like PageRank might be omitted because they do not align with the specific objectives of the network analysis.

SECTION V “OVERCOMING OBSTACLES"

Change in Platform Selection

Challenge:
Initially, We planned to scrape data from YouTube for analysis. However, upon further investigation and evaluation of available platforms, it was determined that GitHub would better serve the project's objectives.
Reasoning:
YouTube primarily hosts video content and focuses on user-generated videos, comments, and engagement metrics, which may not align with the project's specific requirements.
GitHub, on the other hand, offers a wealth of structured data related to software development, including repositories, user profiles, contributions, and collaboration networks. This data is well-suited for analysis and insights generation, aligning more closely with the project's goals.

Handling Errors

Challenge:
Scraping may encounter various errors, such as network errors or HTTP status codes indicating issues.
Solution:
Implement robust error handling to gracefully handle exceptions and retries. Log errors for debugging and monitoring purposes.
Website Changes

Challenge:
Websites frequently undergo changes in layout, structure, or HTML elements, causing scraping scripts to break.
Solution:
Regularly monitor the target website for changes. Implement robust error handling to gracefully handle unexpected changes. Update scraping scripts accordingly to adapt to the new structure.

SECTION VI FURTHER QUESTIONS AND ANALYSIS

Further Questions:

- Are there specific patterns or trends in the frequency and nature of website changes?
- How do website changes correlate with updates or events within the organization?
- Are there any underlying factors driving these website modifications?
- What impact do these changes have on user experience or functionality?
- Are there any differences in the frequency of changes across different sections or pages of the website?

Next Steps:

- Conduct a deeper analysis of the timing and frequency of website changes.
- Investigate potential reasons behind these modifications, such as product launches, policy updates, or technological advancements.
- Assess the user feedback or response to recent website changes.
- Explore strategies to streamline the website change management process and minimize disruptions.
- Collaborate with web development teams to enhance communication and coordination regarding website updates.

SECTION VII TEAM’S CONTRIBUTION

Tasks	**Name**
Data Sourcing and Authentication	Priyanka
Data Storing and Processing	Manav
Network Measures Calculation	Priyanka
Graph Visualization	Priyanka, Manav
Observations and Research	Manav
Project Report Writing	Priyanka, Manav

SECTION VIII DATA USAGE AND POLICY

- Unlike accessing data through the GitHub API, web scraping GitHub user profiles typically does not require authentication. Publicly accessible user profiles can be scraped without the need for authentication credentials.However, if the script encounters rate limiting or access restrictions due to excessive scraping, it may be necessary to implement measures such as rate limiting, delays between requests, or rotating IP addresses to avoid detection and potential blocking by GitHub.
- **Data Usage Policy:** While web scraping publicly available data from GitHub is generally permissible, it's important to adhere to GitHub's terms of service and acceptable use policy. GitHub's terms of service prohibit abusive behavior, including excessive scraping that may impact the performance or availability of their services.Developers should be mindful of GitHub's robots.txt file, which may specify rules and guidelines for web crawlers and scrapers.
- **Data Privacy and Security:** When scraping data from GitHub user profiles, developers should be cautious not to inadvertently collect or expose sensitive information.GitHub users may have different privacy settings configured for their profiles, so it's essential to respect these settings and only collect information that is publicly available.Additionally, developers should implement measures to ensure the security of collected data, such as securely storing and handling any personally identifiable information.

https://docs.github.com/en/site-policy/acceptable-use-policies/github-acceptable-use-policies
https://docs.github.com/en/site-policy/privacy-policies/github-general-privacy-statemen

SECTION IX   REFERENCE

1) https://docs.python.org/3/library/

2) https://www.crummy.com/software/BeautifulSoup/bs4/doc/

3) https://stackoverflow.com/questions/49722640/how-to-create-edges-for-nodes

4) https://ieeexplore.ieee.org/document/8821809

5) https://www.geeksforgeeks.org/betweenness-centrality-centrality-measure/

6) https://www.youtube.com/watch?v=0EekpQBEP_8

7) https://gephi.org/plugins/#/

8) https://www.youtube.com/watch?v=OYrzMhT0Kag&t=60s

9) https://www.youtube.com/watch?v=YM_37z_uURM

10) https://docs.gephi.org/Developer_Documentation/localization