PROJECT REPORT

CS 579 – Online Social Network Analysis

Spring 2024

Project 1 – Social Media Data Analysis

Illinois Institute of Technology

By

Akshat Behera – A20516439

Rushikesh Kadam – A20517258

INDEX

Introduction

Our project investigates social media data analysis with a focus on Reddit. We detail our approach, starting with data collection. Initially, we intended to use Twitter but encountered API and other issues discussed later, prompting us to switch to Reddit for its data reliability.

We visually analyze Reddit data, using histograms and bar plots to understand network structures. We then examine network measures like degree centrality and betweenness centrality to determine node importance and influence.

Furthermore, we explore additional metrics such as PageRank, clustering coefficient, and degree distribution to gain insights into Reddit's network dynamics.

In summary, our report provides insights into Reddit's social network structure and community interactions, contributing to the broader understanding of online social networks and suggesting avenues for future research.

1. Transition from Twitter to Reddit: Overcoming Data Collection Challenges

Roadblocks Encountered:

During the initial phase of our project, we had planned to utilize the X's Twitter API (f.k.a. "Twitter") for data collection due to its extensive user base and rich data offerings. However, we encountered several significant roadblocks that impeded our progress and ultimately led us to switch to Reddit for data collection.

Limited Developer Access: One major challenge stemmed from the limited accessibility provided by Twitter's API, particularly in its free tier. We found ourselves constrained by inadequate functionalities and restricted access to essential data attributes. Moreover, the limitations on API call frequency impeded our ability to gather real-time or comprehensive data.

Rapid API Changes: Additionally, Twitter's API underwent frequent updates and revisions, notably with the introduction of API V2. These rapid changes presented compatibility issues, making it challenging to adapt to the evolving API structure. Consequently, ensuring compatibility and functionality became increasingly burdensome.

Furthermore, to overcome the limitations of the free tier, Twitter suggested upgrading to the Basic plan, which incurs a significant monthly cost of $100. However, there was uncertainty regarding whether the Basic plan would adequately address our data collection needs. Committing to such a huge amount without assurance of compatibility or success posed a significant risk to our project.

Decision to Switch to Reddit:

After careful consideration of the challenges posed by Twitter's API and the limitations of the free tier, we made the decision to transition our data collection efforts to Reddit. Several factors influenced this decision:

Rich Data Availability: Reddit offers a vast and diverse repository of user-generated content, discussions, and interactions across various topics and communities. The platform provides extensive access to data attributes, including user profiles, submissions, comments, and engagement metrics, which align closely with our project objectives.

API Flexibility and Documentation: Reddit's API provides robust documentation and a more flexible framework for data retrieval and analysis. The endpoints and parameters offered by Reddit's API are well-documented and allow for more granular control over data collection processes.

Cost-Effectiveness: Unlike Twitter's Basic plan, Reddit's API access is available at no cost, making it a more cost-effective option for our project. Without the financial burden associated with subscription fees, we can allocate resources more efficiently and focus on refining our data collection and analysis methodologies.

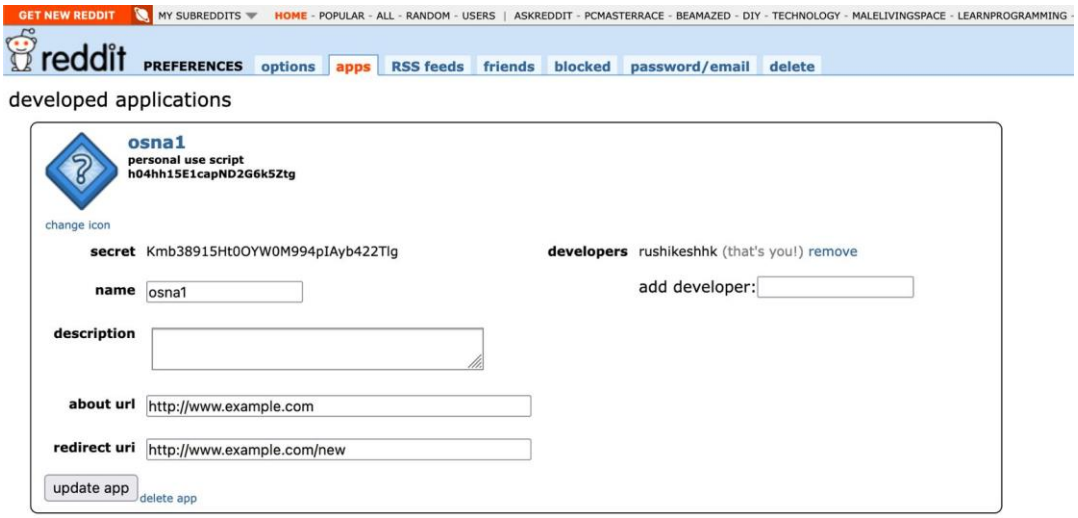
In summary, the decision to switch from Twitter to Reddit for data collection was driven by the significant roadblocks encountered with Twitter's API, including limited developer access, rapid API changes, and the high monetary commitment required for full access. Reddit emerged as a viable alternative due to its rich data availability, flexible API framework, and cost-effectiveness. This transition represents a decision aimed at overcoming challenges and advancing our project goals effectively.

2. Data Collection

The network constructed from Reddit data, where users interact with each other and submissions (i.e., Posts) through comments and submissions (i.e., Posts), is a friendship network, and it represents a unimodal network since it consists of only one type of node (users) connected by edges representing interactions between users.

Creating Reddit Developers Account:

- First, we have login or sign up at Reddit.com, with required credentials (i.e., username and password). In this case, we have used our personal account.
- Then, we visit the https://www.reddit.com/prefs/apps, from which we can "apps", we gain the API credentials. As shown below:



- From this, we obtain API credentials such as personal keys and secret key. Which enables us to call the API and access and collect data from the source for the purposes of our project.

Data Collection Methodology:

For this project, we utilized the Reddit API through the PRAW (Python Reddit API Wrapper) library to collect data from the Reddit social media platform. The data collection process involved the following steps:

1. Import Packages and Initializing Reddit API Client: We imported the necessary packages such as praw, network, Json, pandas, matplotlib, re and NumPy. Then, we initialized the Reddit API client by providing the necessary authentication credentials such as client ID, client secret, and user agent.

Package Imports

In [186]:
```
import praw
import pandas as pd
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np
import json
import re
```

Initializing Reddit API Client with Credentials from JSON File

In [187]:
```
# This function will load credentials from a JSON file
def load_credentials(path):
    with open(path, 'r') as file:
        return json.load(file)

#THis function will initialize the Reddit API client
def initialize_reddit(credentials):
    #get credentials by using get()
    client_id = credentials.get('client_id')
    client_secret = credentials.get('client_secret')
    user_agent = credentials.get('user_agent')

    return praw.Reddit(client_id=client_id,
                       client_secret=client_secret,
                       user_agent=user_agent)

file_name = 'credentials.json'
credentials = load_credentials(file_name)#load credentials
reddit = initialize_reddit(credentials)#initialize credentials

print("Reddit API Intialized Successfully")
```

```
Reddit API Intialized Successfully
```

2. Extracting Data from a Subreddit: We defined a function `extract_data` to extract data from a specified subreddit. The function retrieves the top submissions (i.e., Posts) from the subreddit and iterates through each submission to extract the author's username and up to 120 commenters' usernames per submission (i.e., Post). We limited the number of comments per submission to control the data volume and processing time.

Global Variable Declaration

In [188]:
```
#defining variables to store nodes, edges and related data
raw_data = []
nodes = set()
edges = []
```

User Input: Set Limit and Subreddit Name

In [189]:
```
subreddit_name1 = input("Enter the subreddit name from which you want to extract the data: ")
```

```
Enter the subreddit name from which you want to extract the data: dataengineering
```

In [190]:
```
limit1 = int(input("Enter the limit to extract number of Submissions: "))
```

```
Enter the limit to extract number of Submissions: 6
```

Data Extraction

In [191]:
```
import praw
# this function will extract the data from Reddit API
def extract_data(subreddit_name, limit):
    subreddit = reddit.subreddit(subreddit_name) #Accessing the subreddit with the given name
    submissions = subreddit.top(limit=limit) # Selecting top submissions with the given limit

    for submission in submissions:
        if submission.author:
            author = submission.author.name #setting the name of an author to the attribute
        else:
            author = "Unknown"# if there is no name of an author then setting it as unknown
        nodes.add(author)

        comment_count = 0  #counting comments for limiting the extraction

        #this loop will add name of the commenter in nodes who has commented on a post posted by an author
        #Also this loop will append the list of edges with the pair of author and commenter
        for comment in submission.comments.list():
            if(comment_count < 120):
                if hasattr(comment, 'author') and comment.author:
                    commenter = comment.author.name #getting name of the author who has commented
                    nodes.add(commenter)
                    edges.append((author, commenter))
                    comment_count = comment_count + 1 #increasing the comment count by one after getting data for comment

    #this loop will append edges in raw_data
    for edge in edges:
        raw_data.append({'source': edge[0], 'target': edge[1]})

    return nodes, edges, raw_data


#calling Extract_data() to get edges, nodes and raw data with required limit and subreddit name
nodes, edges, raw_data = extract_data(subreddit_name1, limit1)

print("Successfully extracted data from Reddit.")
```

```
Successfully extracted data from Reddit.
```

3. Cleaning Data: After extracting the data, we performed data cleaning to remove any noise, inconsistencies, or irrelevant information. The cleaning process involved removing self-loops, duplicate edges, edges with missing or invalid values, and filtering out usernames based on various criteria such as length, presence of special characters, and non-ASCII characters.

Data Cleaning

In [193]:
```
#removing any self loops

filtered_edges = []

for edge in edges:
    # Check if the source node is not equal to the target node
    if edge[0] != edge[1]:
        # If the edge does not form a self-loop, append it to the filtered_edges list
        filtered_edges.append(edge)
edges = filtered_edges
```

In [194]:
```
#removing duplicate edges
unique_edges = []
for edge in edges:
    sorted_edge = tuple(sorted(edge))

    if sorted_edge not in unique_edges:
        unique_edges.append(sorted_edge)
```

In [195]:
```
#removing edges with missing or None values
clean_edges = []
for edge in unique_edges:
    if None not in edge and "" not in edge:
        clean_edges.append(edge)
```

In [196]:
```
#filtering out irrelevant or noisy data (removing edges with usernames containing special characters)
filtered_clean_edges = []

for edge in clean_edges:
    if all(c.isalnum() or c in ['_', '-'] for c in edge):
        filtered_clean_edges.append(edge)

# Update the 'clean_edges' list with the filtered clean edges
clean_edges = filtered_clean_edges
```

In [197]:
```
#removing edges with usernames that are too short or too long(Here we are considering legth between 3 and 20)

filtered_clean_edges = []

for edge in clean_edges:
    if 3 <= len(edge[0]) <= 20 and 3 <= len(edge[1]) <= 20:
        filtered_clean_edges.append(edge)

clean_edges = filtered_clean_edges
```

In [198]:
```
#removing edges with usernames containing non-ASCII characters

filtered_clean_edges = []

for edge in clean_edges:
    if all(ord(c) < 128 for c in edge[0]) and all(ord(c) < 128 for c in edge[1]):
        filtered_clean_edges.append(edge)

clean_edges = filtered_clean_edges
```

In [199]:
```
#removing edges with usernames containing consecutive underscores or hyphens

filtered_clean_edges = []

for edge in clean_edges:
    if not re.search(r'[-_]{2,}', edge[0]) and not re.search(r'[-_]{2,}', edge[1]):
        filtered_clean_edges.append(edge)

clean_edges = filtered_clean_edges
```

In [200]:
```
#storing all cleaned edges in clean data

clean_data = [{'source': edge[0], 'target': edge[1]} for edge in clean_edges]

print("Data cleaning completed successfully!!!")
```

```
Data cleaning completed successfully!!!
```

4. Saving Data to CSV Files: We saved both raw, cleaned as well as the nodes.csv and edges.csv data to CSV files for further analysis. The raw data CSV contains information about the source and target nodes, while the clean data CSV contains sanitized data ready for analysis. Nodes.csv contains unique node identifiers and usernames, while edges.csv includes source-target pairs representing connections.

Exporting extracted data to CSV File

In [201]:
```
#this function will save extracted data to CSV files according to the type of data
def save_data_to_csv(data, file_name):

    #if the given data is in list it will convert that list into dataframe
    if isinstance(data, list):
        data = pd.DataFrame(data)

    data.to_csv(file_name, index=False)#saving in CSV file
    print("Successfully saved data Dto CSV:", file_name)

#saving raw_data and clean_edges to CSV files
save_data_to_csv(raw_data, 'raw_data.csv')
save_data_to_csv(clean_edges, 'clean_data.csv')


#saving nodes data to CSV file
# generating list of unique ids for nodes
node_ids = range(len(nodes))
# Creating dataframe with id and node for storing nodes data
nodes_df = pd.DataFrame({'ID': node_ids, 'node': list(nodes)})
#saving data to csv
save_data_to_csv(nodes_df, 'nodes.csv')

#saving edges data to CSV file
#creating dataframe with 'source' and 'target' columns for edges
edges_df = pd.DataFrame({'source': [edge[0] for edge in edges], 'target': [edge[1] for edge in edges]})
#saving edges data to CSV file
save_data_to_csv(edges_df, 'edges.csv')
```

```
Successfully saved data Dto CSV: raw_data.csv
Successfully saved data Dto CSV: clean_data.csv
Successfully saved data Dto CSV: nodes.csv
Successfully saved data Dto CSV: edges.csv
```

nodes.csv

	A	B
1	ID	node
2	0	ThatGrayZ
3	1	MadDevloper
4	2	DozenAlarmedGoats
5	3	vmonsale
6	4	uncomfortablepanda
7	5	7818
8	6	PhotographsWithFilm
9	7	puripy
10	8	ProgrammersAreSexy
11	9	SailorGirl29
12	10	Georgehwp
13	11	7unom
14	12	freaking_scared
15	13	East_Pattern_7420
16	14	XtremeGoose
17	15	BlueSea9357
18	16	DirtzMaGertz
19	17	Awkward-Block-5005
20	18	turalfirst
21	19	BoringWozniak
22	20	pescennius
23	21	ZirePhiinix
24	22	Crowsby
25	23	Imaginary-Hawk-8407
26	24	Nick_AxeusConsulting
27	25	LelouchYagami_

edges.csv

	A	B
1	source	target
2	OldPartici	kentmaxwell
3	OldPartici	PhotographsWithFilm
4	OldPartici	zazzersmel
5	OldPartici	MadDevloper
6	OldPartici	chrisgarzon19
7	OldPartici	daguito81
8	OldPartici	BoringWozniak
9	OldPartici	creamyhorror
10	OldPartici	FloggingTheHorses
11	OldPartici	FatLeeAdama2
12	OldPartici	mTiCP
13	OldPartici	ChewbaccaFuzball
14	OldPartici	GreenWoodDragon
15	OldPartici	Live-Key8030
16	OldPartici	elus
17	OldPartici	Puzzleheaded-Sun3107
18	OldPartici	NumerousIndependent2
19	OldPartici	dfwtjms
20	OldPartici	marmenia
21	OldPartici	Cpt_keaSar
22	OldPartici	jayrob211
23	OldPartici	mohamed_af1
24	OldPartici	Evening_Chemist_2367
25	OldPartici	Picasso1067

Challenges Faced:

- **API Rate Limiting**: Reddit's API has rate limits for requests, which can slow down the data collection process, especially when dealing with a large amount of data. We managed this challenge by implementing appropriate throttling mechanisms and limiting the number of API requests per unit of time.
- **Data Noise and Inconsistencies**: Reddit data can contain noise, such as deleted users or spam accounts. We addressed this challenge by implementing robust data cleaning procedures to filter out irrelevant or invalid data points.
- **Data Privacy Concerns**: While collecting data from Reddit, we ensured compliance with Reddit's API usage policies and respected user privacy by not collecting any personally identifiable information.

Impact on Data Collection:

The issues faced during data collection affected both the quality and quantity of the data gathered. Implementing rigorous data cleaning methods helped minimize the effects of noise and inconsistencies on the dataset's reliability.

However, restrictions imposed by API rate limits slowed down the data collection process and potentially reduced the dataset size. Nevertheless, despite these challenges, the collected data remains valuable for understanding the patterns and behaviors within Reddit's online social networks.

User Privacy Policy and Data Usage Policy [14]:

As per Reddit's user privacy policy and data usage policy: Below are few citations from their documentation.

User Privacy Policy: Reddit collects user data to provide and improve its services, personalize user experiences, and comply with legal obligations. Reddit may collect various types of data, including user-generated content, device information, and cookies. Reddit takes user privacy seriously and implements measures to protect user data from unauthorized access or disclosure.

Data Usage Policy: Reddit uses collected data for various purposes, such as providing and improving services, conducting research and analysis, and personalizing content and advertisements. Reddit may share user data with third-party service providers, affiliates, or legal authorities as required by law or to protect its rights and interests.

3. Data Visualization

The software used for graph analysis is NetworkX, a Python library known for its extensive functionality in analyzing and visualizing complex networks. Chosen for its Python integration, comprehensive features, active development community, and flexibility, NetworkX offers tools for various network measures and graph visualization.

Visualization of Network Graph

In [202]:
```
#this function will visualize the network graph by using cleaned nodes and edges data
def visualize_network_graph(nodes, edges):
    G = nx.Graph()

    #adding nodes and edged to the graph
    G.add_nodes_from(nodes)
    G.add_edges_from(edges)

    plt.figure(figsize=(16, 14))

    #drawing the graph
    nx.draw(G, with_labels=True, node_color='pink', node_size=500, edge_color='black', linewidths=1, font_size=8)

    plt.title('Reddit Users Network Graph')
    plt.show()
    print("Reddit Users Network graph visualized Successfully!")

# Visualize the network graph
visualize_network_graph(nodes, edges)
```

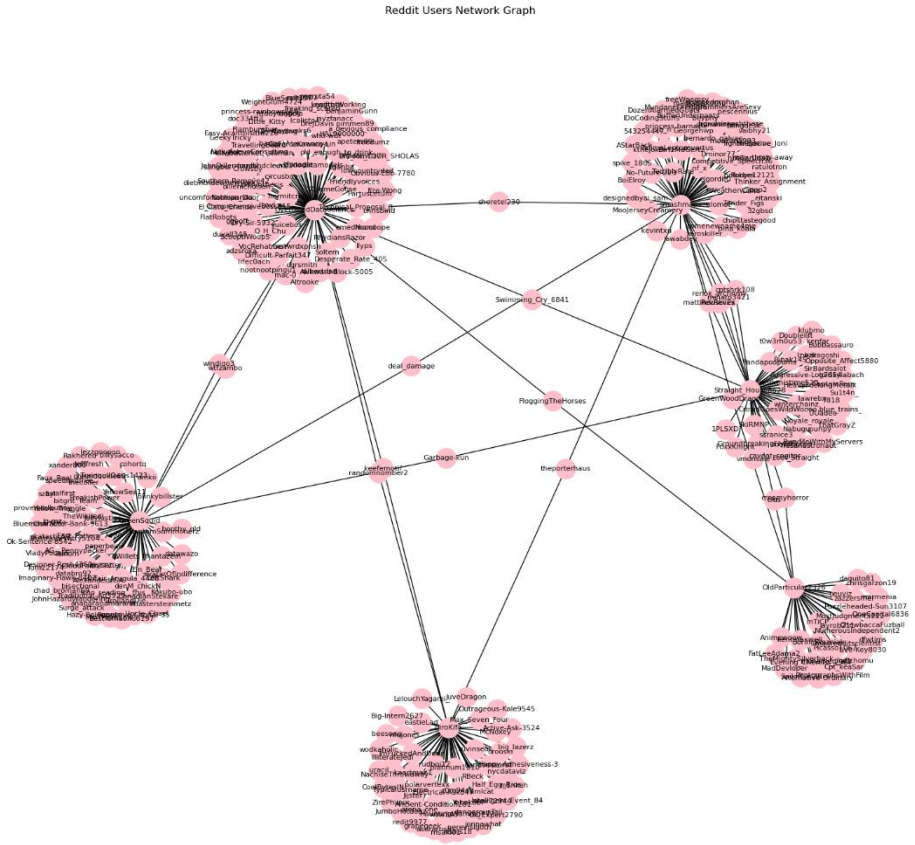


Fig. Network Graph

In addition, Gephi, a user-friendly network visualization tool, was used alongside NetworkX. nodes.csv and edges.csv were imported into Gephi for advanced visual exploration of the network's structure. Gephi's intuitive interface and visualization options complemented NetworkX's analysis, enhancing insights into the social network's dynamics.

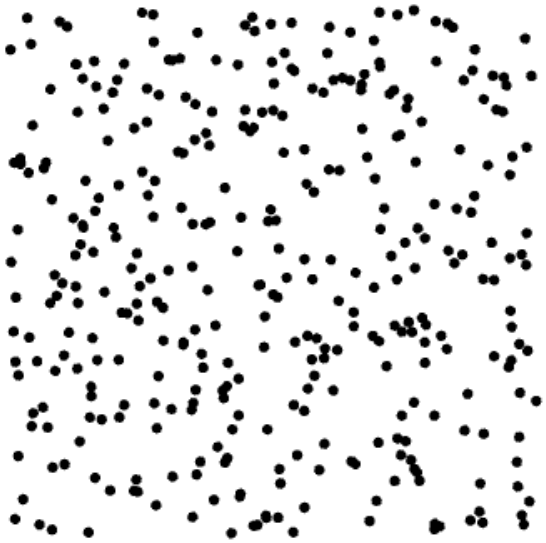


Fig. Nodes

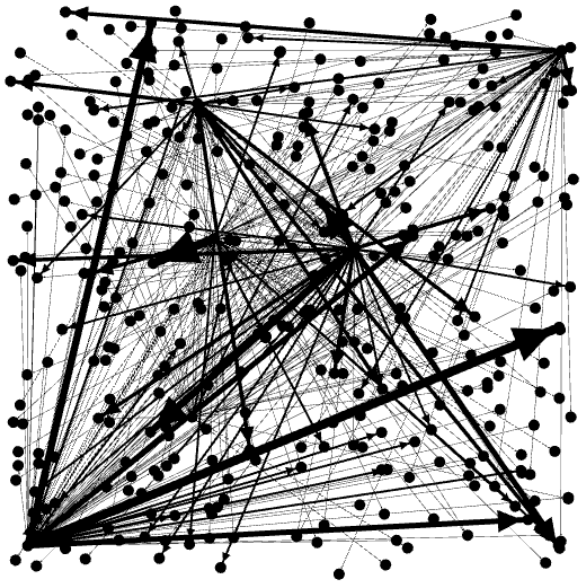


Fig. Connected Nodes with Edges

The data input format consists of CSV files for nodes and edges. Nodes.csv contains unique node identifiers and usernames, while edges.csv includes source-target pairs representing connections. This format allows for easy data manipulation using pandas and seamless integration with Gephi and NetworkX.

4. Network Measures

For this section, we examined the degree distribution, providing a glimpse into node connectivity and importance. Subsequently, we explored the clustering coefficient to understand structures within Reddit. Additionally, we investigated betweenness centrality and other measures, and plotted histograms for each of the measures.

We will see these measures in detail and give analysis comment for each of the measure in the following section:

1. Degree Centrality

In [204]:
```
#calculating degree centrality
degree_centrality = nx.degree_centrality(G)

#converting degree centrality to DataFrame for better readability
df = pd.DataFrame(degree_centrality.items(), columns=['Node', 'Degree Centrality'])

#calculating Degree Centrality Mean
mean_degree_centrality = df['Degree Centrality'].mean()


#printing results
print("Degree Centrality calculated:")
print(df)
print("Mean Degree Centrality:", mean_degree_centrality)

print("Degree Centrality calculated successfully!!!")
```

```
Degree Centrality calculated:
                  Node  Degree Centrality
0             ThatGrayZ           0.002849
1           MadDevloper           0.002849
2     DozenAlarmedGoats           0.002849
3              vmonsale           0.002849
4    uncomfortablepanda           0.002849
..                   ...                ...
347           apeters89           0.002849
348       chad_broman69           0.002849
349             Q_H_Chu           0.002849
350        polarvertexx           0.002849
351          hermitcrab           0.002849

[352 rows x 2 columns]
Mean Degree Centrality: 0.005892255892255898
Degree Centrality calculated successfully!!!
```

Average Degree Centrality: Nodes, on average, have a low level of connections, with an average degree centrality of about 0.0059.

2. Betweenness Centrality

In [205]:
```
#calculating betweenness centrality
betweenness_centrality = nx.betweenness_centrality(G)

#converting betweenness centrality to DataFrame for better readability
df = pd.DataFrame(betweenness_centrality.items(), columns=['Node', 'Betweenness Centrality'])

#calculating Betweenness Centrality Mean
mean_betweenness_centrality = df['Betweenness Centrality'].mean()

#printing results
print("Betweenness Centrality calculated:")
print(df)
print("Mean Betweenness Centrality:", mean_betweenness_centrality)

print("Betweenness Centrality calculated successfully!!!")
```

```
Betweenness Centrality calculated:
                    Node  Betweenness Centrality
0             ThatGrayZ                     0.0
1           MadDevloper                     0.0
2     DozenAlarmedGoats                     0.0
3              vmonsale                     0.0
4    uncomfortablepanda                     0.0
..                  ...                     ...
347          apeters89                     0.0
348       chad_broman69                     0.0
349            Q_H_Chu                     0.0
350        polarvertexx                     0.0
351          hermitcrab                     0.0

[352 rows x 2 columns]
Mean Betweenness Centrality: 0.00819985569985569
Betweenness Centrality calculated successfully!!!
```

Average Betweenness Centrality: Nodes play a moderate role in connecting different parts of the network, with an average betweenness centrality of approximately 0.0082.

3. Closeness Centrality

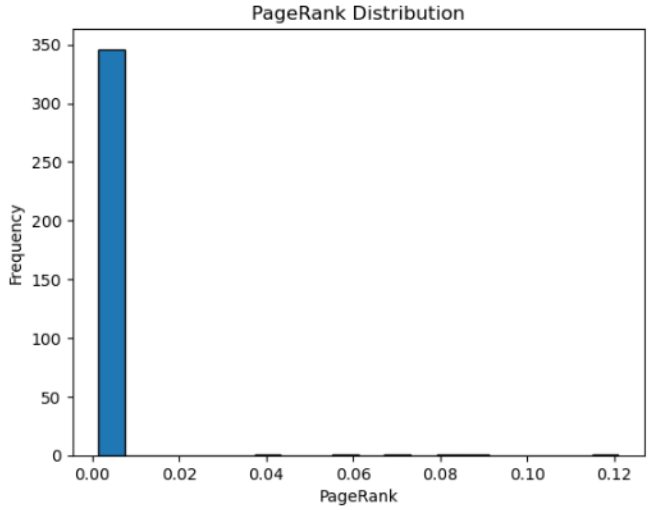
In [206]:
```
#calculating closeness centrality
closeness_centrality = nx.closeness_centrality(G)

#converting closeness centrality to DataFrame for better readability
df = pd.DataFrame(closeness_centrality.items(), columns=['Node', 'Closeness Centrality'])

#calculating Closeness Centrality Mean
mean_closeness_centrality = df['Closeness Centrality'].mean()

#printing results
print("Closeness Centrality calculated:")
print(df)
print("Mean Closeness Centrality:", mean_closeness_centrality)

print("Closeness Centrality calculated successfully!!!")
```

```
Closeness Centrality calculated:
                    Node  Closeness Centrality
0             ThatGrayZ              0.248936
1           MadDevloper              0.223282
2     DozenAlarmedGoats              0.276378
3              vmonsale              0.248936
4    uncomfortablepanda              0.290563
..                  ...                   ...
347          apeters89              0.290563
348       chad_broman69              0.246489
349            Q_H_Chu              0.290563
350        polarvertexx              0.224138
351          hermitcrab              0.290563

[352 rows x 2 columns]
Mean Closeness Centrality: 0.2615897138932656
Closeness Centrality calculated successfully!!!
```

Average Closeness Centrality: Nodes are relatively well-connected within the network, with an average closeness centrality of around 0.2616.

4. PageRank

```
In [207]: pagerank = nx.pagerank(G)

          #plotting histogram
          plt.hist(list(pagerank.values()), bins=20, edgecolor='black')

          #setting title and labels
          plt.title('PageRank Distribution')
          plt.xlabel('PageRank')
          plt.ylabel('Frequency')

          #displaying the plot
          plt.show()

          pagerank_values = list(pagerank.values())
          avg_pagerank = sum(pagerank_values) / len(pagerank_values)

          print("Average PageRank:", avg_pagerank)

          print("PageRank distribution plotted successfully!")
```

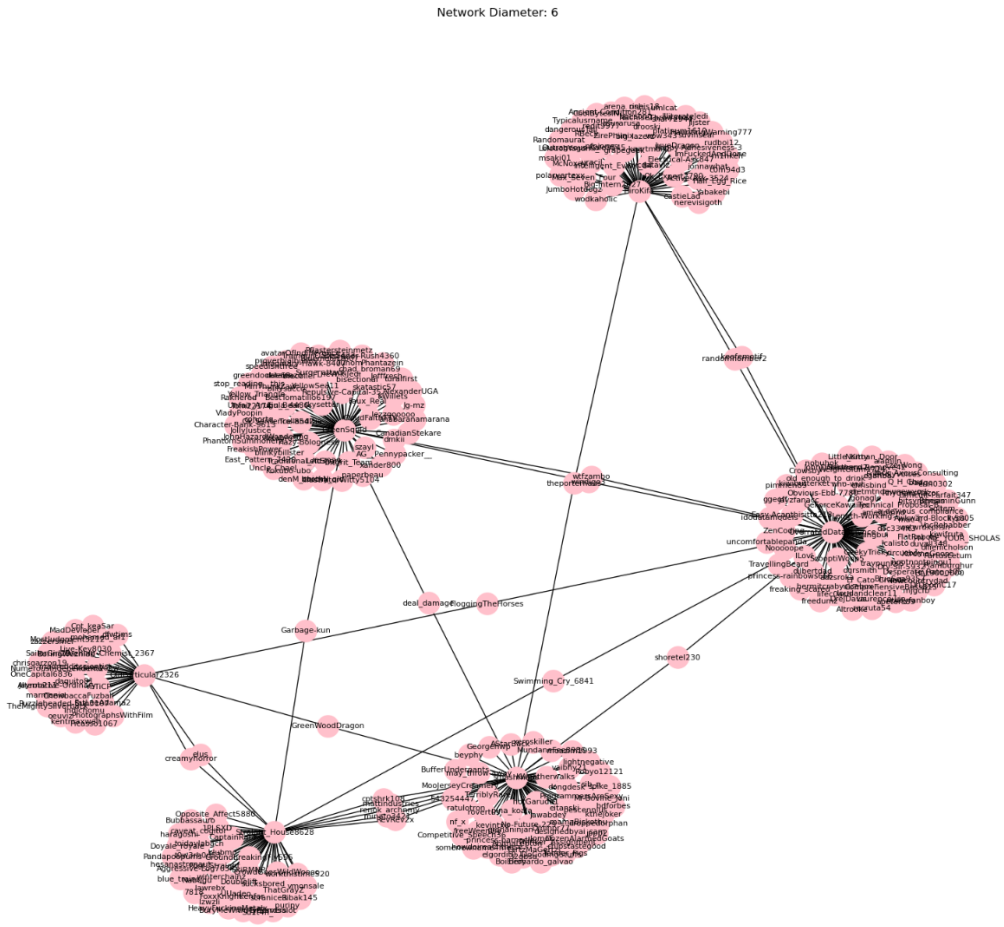



```
Average PageRank: 0.002840909090909106
PageRank distribution plotted successfully!
```

Average PageRank: Nodes have relatively low importance based on incoming links, with an average PageRank score of about 0.0028.

5. Network Diameter

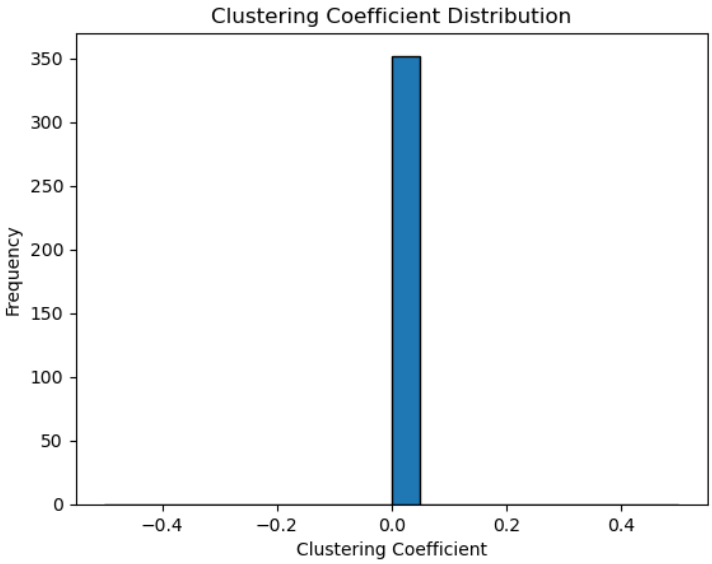
```
In [214]:

          #calculating the network diameter
          diameter = nx.diameter(G)

          # Plot the network
          pos = nx.spring_layout(G)  # positions for all nodes

          plt.figure(figsize=(16, 14)) #setting fig size for the plot

          nx.draw(G, pos, with_labels=True, node_color='pink', node_size=500, edge_color='black', linewidths=1, font_size=8)
          plt.title(f'Network Diameter: {diameter}') #setting title

          #displaying results
          plt.show()
```



Network Diameter: The network diameter is 6, indicating a moderate level of interconnectedness.

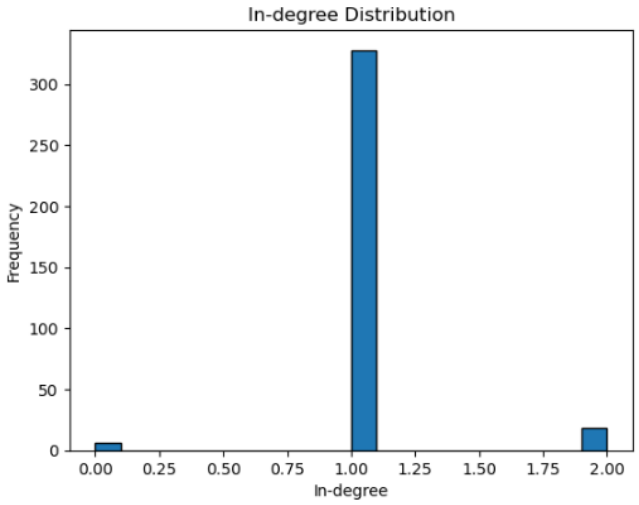
6. Clustering Coefficient

```
In [219]: clustering = nx.clustering(G)
          clustering_values = list(clustering.values())

          #calculating average clustering coefficient
          avg_clustering = sum(clustering_values) / len(clustering_values)

          #plotting histogram
          plt.hist(list(clustering.values()), bins=20, edgecolor='black')

          #setting title and labels
          plt.title('Clustering Coefficient Distribution')
          plt.xlabel('Clustering Coefficient')
          plt.ylabel('Frequency')

          #displaying results
          plt.show()

          #print("Average Clustering Coefficient:", avg_clustering)

          print("Clustering coefficient distribution plotted successfully!")
```

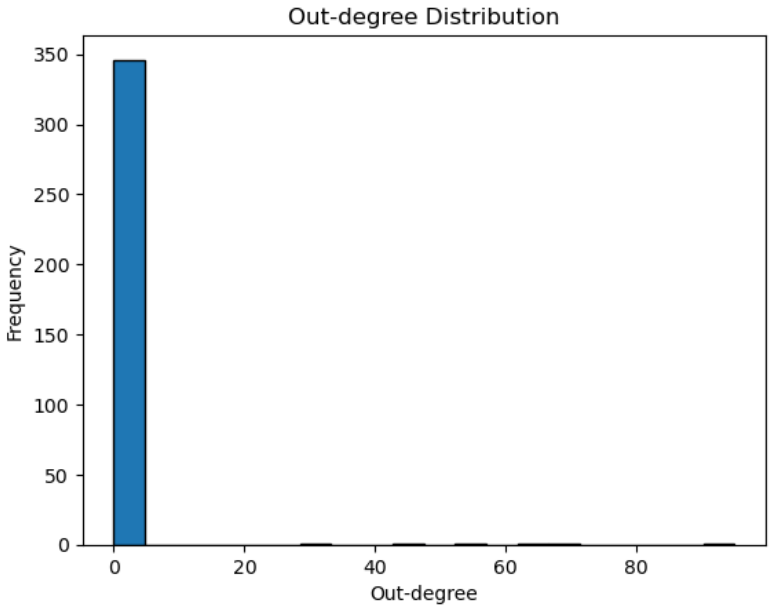


7. In-degree and Out-degree Distribution

In [210]:
```
#calculating In-degree
indegree = dict(G.in_degree())

#plotting histogram
plt.hist(list(indegree.values()), bins=20, edgecolor='black')

#setting title and labels
plt.title('In-degree Distribution')
plt.xlabel('In-degree')
plt.ylabel('Frequency')

#displaying the plot
plt.show()
print("In-degree distribution plotted successfully!")

#calculating and printing the average in-degree
avg_indegree = sum(indegree.values()) / len(indegree)
print("Average In-degree:", avg_indegree)
```



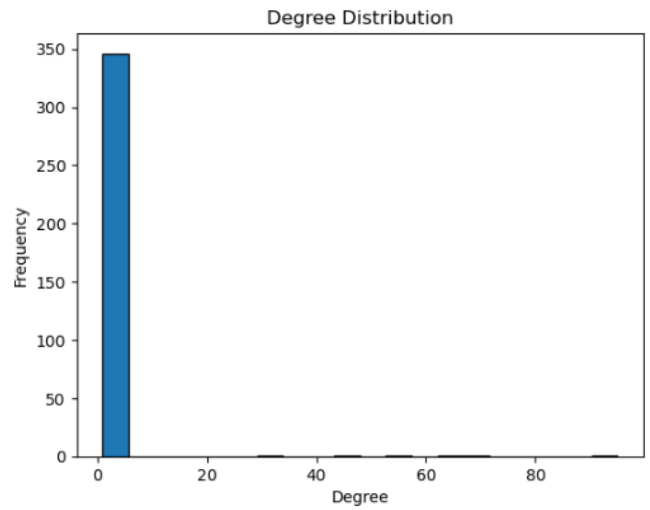
```
In [211]: #calculating Out-degree Distribution
outdegree = dict(G.out_degree())

#plotting histogram
plt.hist(list(outdegree.values()), bins=20, edgecolor='black')

#setting title and labels
plt.title('Out-degree Distribution')
plt.xlabel('Out-degree')
plt.ylabel('Frequency')

#displaying results
plt.show()
print("Out-degree distribution plotted successfully!")

#calculating and printing the average out-degree
avg_outdegree = sum(outdegree.values()) / len(outdegree)
print("Average Out-degree:", avg_outdegree)
```



Average In-degree and Out-degree Distribution: Both in-degree and out-degree distributions show a balanced distribution of connections, with an average value of approximately 1.03409.

8. Degree Distribution

```
In [213]: #calculating Degree Distribution
degree_sequence = [d for n, d in G.degree()]

#calculating mean degree distribution
mean_degree = np.mean(degree_sequence)

#plotting the histogram
plt.hist(degree_sequence, bins=20, edgecolor='black')

#setting title and labels
plt.title('Degree Distribution')
plt.xlabel('Degree')
plt.ylabel('Frequency')

#displaying the results
plt.show()
print("Mean Degree Distribution:", mean_degree)
print("Degree distribution plotted successfully!")
```




```
Mean Degree Distribution: 2.0681818181818183
Degree distribution plotted successfully!
```

Average Degree Distribution: The average degree distribution is 2.0909, suggesting a sparse and decentralized network structure.

Extra Measures:

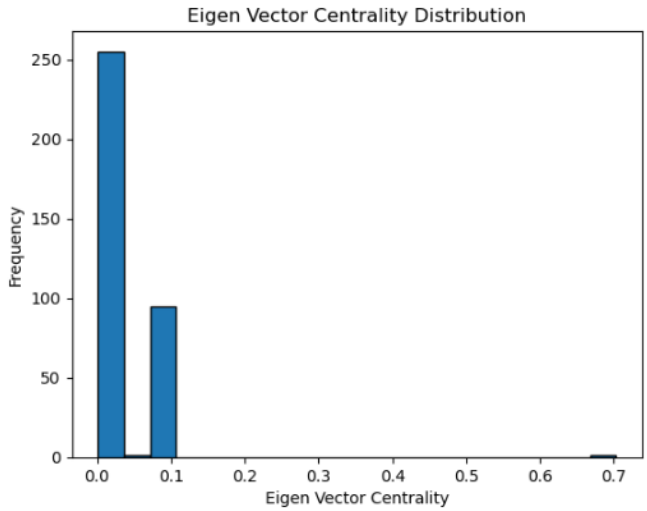
1. Graph Density: The graph density is about 0.0059, indicating a low level of connectivity in the network.

In [217]:
```
#calculating and displaying Graph Density

density = nx.density(G)

print("Graph Density:", density)
```

```
Graph Density: 0.005892255892255892
```

2. Eigen Vector Centrality

In [215]:
```
#increasing the maximum number of iterations and adjusting the tolerance level for more accurate calculations
eigen_centrality = nx.eigenvector_centrality(G, max_iter=1000, tol=1e-6)

#plotting histogram
plt.hist(list(eigen_centrality.values()), bins=20, edgecolor='black')

#setting title and labels
plt.title('Eigen Vector Centrality Distribution')
plt.xlabel('Eigen Vector Centrality')
plt.ylabel('Frequency')

#displaying results
plt.show()
print("Eigen vector centrality distribution plotted successfully!")


# Calculate and print the average eigen centrality
avg_eigen = sum(eigen_centrality.values()) / len(eigen_centrality)
print("Avg. Eigen Centrality Value:", avg_eigen)
```

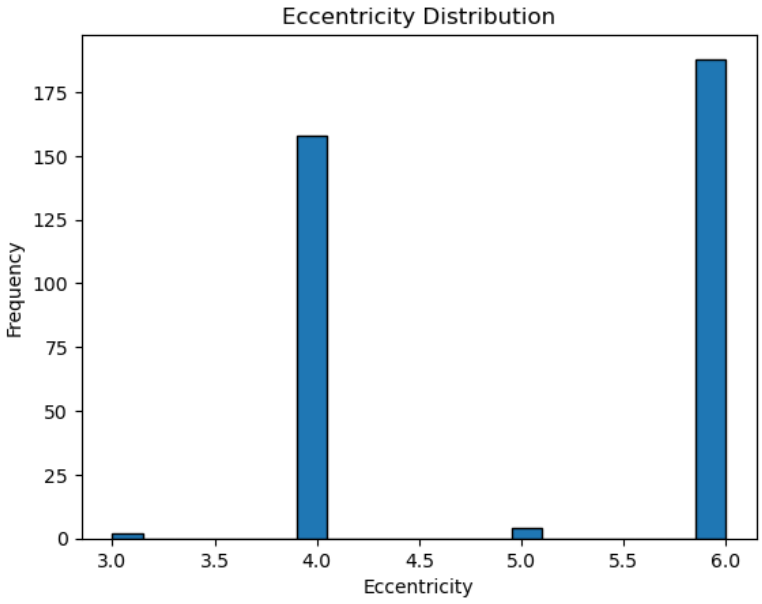



```
Eigen vector centrality distribution plotted successfully!
Avg. Eigen Centrality Value: 0.024519867249187956
```

Average Eigen Centrality Value: Nodes have moderate importance based on connections to other important nodes, with an average eigen centrality value of around 0.0245.

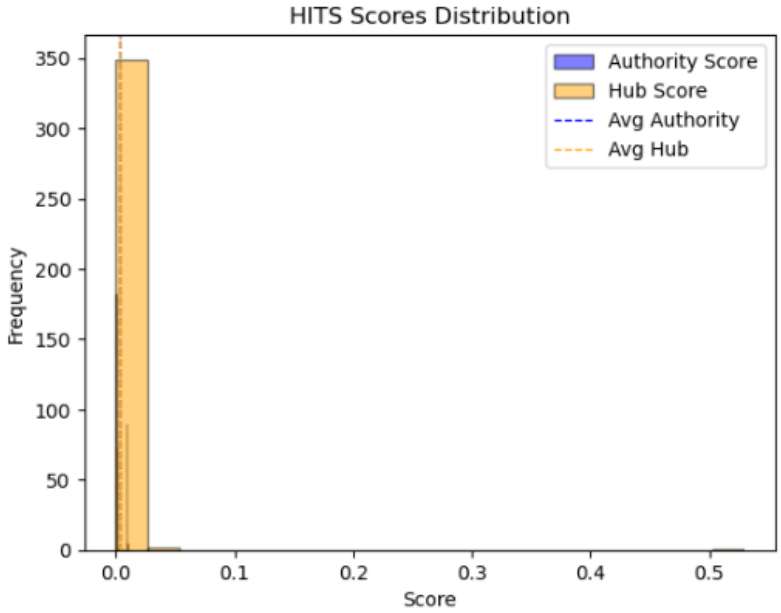
3. Eccentricity Distribution

```
In [216]: #calculating Eccentricity Distribution
          eccentricity = nx.eccentricity(G)

          #plotting histogram for Eccentricity Distribution
          plt.hist(list(eccentricity.values()), bins=20, edgecolor='black')

          #setting title and labels
          plt.title('Eccentricity Distribution')
          plt.xlabel('Eccentricity')
          plt.ylabel('Frequency')

          #displaying results
          plt.show()
          print("Eccentricity distribution plotted successfully!")
```



4. HITS Score

```
In [218]: #calculating HITS scores for the graph
hits = nx.hits(G)

#extracting authority and hub scores from the HITS scores
authority_scores = list(hits[0].values())
hub_scores = list(hits[1].values())

#calculating average authority and hub scores
avg_authority = sum(authority_scores) / len(authority_scores)
avg_hub = sum(hub_scores) / len(hub_scores)

#adding dashed lines representing average authority and hub scores
plt.hist(authority_scores, bins=20, color='blue', alpha=0.5, edgecolor='black', label='Authority Score')
plt.hist(hub_scores, bins=20, color='orange', alpha=0.5, edgecolor='black', label='Hub Score')

plt.axvline(x=avg_authority, color='blue', linestyle='dashed', linewidth=1, label='Avg Authority')
plt.axvline(x=avg_hub, color='orange', linestyle='dashed', linewidth=1, label='Avg Hub')

#setting title and labels for the plot
plt.title('HITS Scores Distribution')
plt.xlabel('Score')
plt.ylabel('Frequency')
plt.legend()

#displaying the plot
plt.show()

print("Average Authority Score:", avg_authority)
print("Average Hub Score:", avg_hub)
print("HITS distribution plotted.")
```




```
Average Authority Score: 0.0028409090909090958
Average Hub Score: 0.0028409090909090958
HITS distribution plotted.
```

Average Authority and Hub Scores: Nodes have low authority and hub scores, with both scores averaging around 0.0028.

In summary, the network displays characteristics of a moderately connected and decentralized structure, with nodes playing moderate roles in connecting different parts of the network and having relatively low importance based on various centrality measures.

5. Discussion of Results and Insights

Observations from data visualization and network analysis provide valuable insights into our network:

Network Structure: The network exhibits decentralization, characterized by nodes having a balanced number of connections. Although not densely connected overall, nodes maintain relative proximity.

Node Importance: Individual nodes demonstrate moderate importance in facilitating connections across the network, rather than high individual significance.

Network Connectivity: Despite not being highly connected, nodes display reasonable interconnectivity.

Questions for Further Investigation:

Community Structure: Is there evidence of distinct communities within the network? If so, how do they interact?

Temporal Dynamics: What are the patterns and trends in the network's evolution over time?

Node Attributes: Do specific node types (e.g., users, content) exert more influence on the network? Are there discernible characteristics that distinguish them?

Proposed Investigation Steps:

Community Detection: Utilize algorithms to uncover potential community structures and analyze their interactions.

Temporal Analysis: Collect data across various time frames to discern evolutionary trends and recurring patterns.

Attribute Analysis: Conduct a detailed examination of node attributes to gain insights into their roles and impact on the network.

6. Conclusion

In our project, we used the praw library to extract data from Reddit, yielding a dataset comprising 352 nodes. The extracted data was then stored in a .csv file for further analysis. Visualizing the friendship network, we utilized Gephi to gain insights into the connectivity patterns and community structures within the Reddit network.

Additionally, we computed network measures including Degree Distribution, PageRank Distribution, and Clustering Coefficients, among others. Histograms were plotted for each measure, providing a visual representation of their respective distributions.

By accomplishing these tasks, we successfully fulfilled the three main objectives of our project and obtained the desired results, contributing to a deeper understanding of Reddit's social media network analysis.

REFERENCES

1. **PRAW** (Python Reddit API Wrapper):

   Package URL: [PRAW GitHub Repository]

   a. PRAW provides a convenient way to access Reddit's API in Python.
2. **Pandas**: Package URL: [Pandas Documentation]

   a. McKinney, W. (2010). Data Structures for Statistical Computing in Python. In Proceedings of the 9th Python in Science Conference (pp. 56–61).
3. **NetworkX**: Package URL: [NetworkX Documentation]

   Stable Documentation URL: (https://networkx.org/documentation/stable/)
4. DiscoverSdk. (n.d.). Compare Gephi vs NetworkX. Retrieved from http://www.discoversdk.com/compare/gephi-vs-networkx.
5. **Matplotlib**:

   Package URL: [Matplotlib Documentation]

   Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. Computing in Science & Engineering, 9(3), 90–95.
6. **NumPy**: Package URL: [NumPy Documentation]
7. **Regular Expressions** (re module): Python Standard Library: [re — Regular expression operations]
8. **Reddit API**: Official Reddit API Documentation: [Reddit API Documentation] The Reddit API provides access to Reddit's vast collection of data including posts, comments, and user information.
9. **Python Standard Library**: Python Software Foundation. (n.d.).

   a. Python Standard Library Documentation
10. **Gephi.** (n.d.). Gephi - The Open Graph Viz Platform. Retrieved from https://gephi.org/
11. **Reddit. (n.d.).** Privacy Policy.
12. Hagberg, A. A., Schult, D. A., & Swart, P. J. (2008). Exploring network structure, dynamics, and function using NetworkX. In Proceedings of the 7th Python in Science Conference (pp. 11–15).
13. **Reddit. (n.d.).** Developer Terms.
14. **Reddit. (n.d.).** Data Usage Policy.