Connection Detection in Wikipedia Article Network of Chameleon

Table of Contents

Abstract

This assignment focuses on analyzing the Wikipedia Article Network related to the Chameleon topic. It employs page-to-page networks, where each article is a node, and the connections between them are edges representing relationships. Information networks are crucial for decision-making, knowledge discovery, and understanding complex data interconnections. The study compares methodologies from two other research projects on Amazon's Co-Purchasing Network and GitHub user interactions. By implementing the Louvain algorithm, the research identifies community structures and influential nodes within the Chameleon dataset. The analysis includes network graph visualization, exploration of network characteristics, and centrality measures and community detection. The results reveal a complex and decentralized structure, highlighting well-defined communities and influential nodes in the dataset. Future work is addressed encompassing additional datasets, subset visualization, link prediction, and k-core cliques analysis. In conclusion, the study enhances our understanding of the Chameleon dataset's network dynamics, influential communities, robust structures, and potential applications in broader network analysis research.

Literature Review

Wikipedia Article Networks come under the category of information networks. In an information network, nodes represent a piece of information, and edges represent the relationships between them, such as similarity, influence, collaboration, etc. Information networks are important in real life as they provide a powerful structure for analyzing and extracting meaningful information from complex and interconnected data. It can provide improved decision-making, knowledge discovery, and enhanced understanding of relationships in various systems. Wikipedia is the most popular encyclopedia website globally. Wikipedia helps with providing valuable resources for educational research, and a wealth of knowledge to users worldwide offering up-to-date information on current events and technologies. Page-to-page networks, highly valuable on platforms like Wikipedia, supply essential information about the structure, relationships, and interconnectivity of information. Overall a page-to-page network signifies the connections between different articles. Each node in the network implies a specific Wikipedia page, and an edge between two nodes signifies a link or reference from one page to another. The page-to-page network function in Wikipedia involves using techniques like connection detection within the Chameleon topic. For example, if node 1 has a "Chameleon features" article and node 2 has a " Chameleons Varieties" article, Edges connect the first node " Chameleon features" and the second node " Chameleons Varieties " because they contain relevant information. Connection detection in the Wikipedia Article Network of Chameleon helps us understand the connections between articles. By exploring links and mutual associations between pages, we learn how information is connected within the Chameleon topic. This analytical approach shows the relationships and influences between different sets of information and increases our understanding of Chameleon-related content on Wikipedia. Several researches have been conducted on network analysis techniques to investigate the analysis of information networks.

For instance, Overlapping Community Detection Versus Ground-Truth in the Amazon Co-Purchasing Network [1] In this study, nodes represent Amazon products, and edges represent their co-purchased products. It introduces a framework for fairly evaluating community detection algorithms using overlapping community structure networks. It highlights the importance of considering topological properties for the effective evaluation of community detection algorithms. The study compares overlapping community detection algorithms using various metrics and performance measures. Different algorithms show varying abilities to detect community structures accurately. Metrics like NMI, Omega, and F-score are used to evaluate algorithm performance. The study stresses the need for alternative methodologies to effectively compare community structures and the complexity of performance measures in community detection. They considered the

distribution of node degree, average clustering coefficient, and hop distance to understand the community structure properties. The analysis focused on macroscopic and microscopic levels, including average clustering coefficient, average shortest path, density, degree correlation, and community size distribution. The research paper lacks the availability of labeled ground-truth data, which blocks a deeper understanding of community structures in real-world networks.

Understanding User Interactions on GitHub: A Social Network Perspective[2]. The research covers a significant portion of GitHub users and provides insights into user behaviors and social connectivity on the platform. where node represents the GitHub developer and edge indicates the interaction among developers on the platform. The study explores from macroscopic to microscopic perspectives, focusing on graph structures, communities, and SH spanners (Social Hub). The network analysis methods used in the study include graph composition analysis, degree distribution analysis, Balance Metric Calculation, Clustering Coefficient, and Betweenness Centrality Analysis. Finally, the Community Detection Method Using the Louvain algorithm to identify and analyze communities within the graph. The clustering coefficient calculation in the study reveals how nodes tend to cluster. The Outcomes indicate that over 90% of nodes have a clustering coefficient of zero. This suggests that a larger percentage of developers on GitHub unintentionally avoid engaging in big deeds and only maintain necessary interactions with specific users. The limitation they may have faced in finding the communities is that they have a large number of datasets.

Through the comparative analysis of these two studies, our research intends to determine their methodologies and address their restrictions. We wish to enhance the analysis by implementing filtering techniques. This approach allows us to achieve a complete understanding of community structures, and identify influential communities and nodes. Our analysis aims to pre-process the dataset, visualize the graph, observe degrees, closeness, and betweenness, and perform community detection using the Louvain algorithm. In comparison to [2], our approach not only involves community detection but also implements filtering techniques to identify the most influential communities and nodes. By performing this operation we can achieve Data Understanding, Graph Visualization, Node Analysis, Community Detection, Identification of Influential Communities and nodes, Enhanced Decision-Making, and Comparative Analysis

Dataset

In our analysis, the dataset was obtained from [3], a Wikipedia Article Network dataset collected from English Wikipedia (December 2018). The dataset is based on the Chameleon Article, where the node represents the Chameleon article, and the edge represents the Chameleon-related article. It has about 2,277 nodes and 31,421 edges, and it is an undirected dataset. In the Jupyter Notebook, the required libraries were imported initially, and the panda's library was utilized to read the CSV file "musae_chameleon_edges.csv." Following the dataset reading, Python code was implemented to obtain the number of nodes and edges. Due to the larger dataset, pre-processing was carried out using random sampling to reduce the number of nodes and edges. 5% of the rows from the original dataset were selected, resulting in 1198 nodes and 1805 edges after applying this 5% random sampling.

Analysis of Dataset

Libraries Used

In the analysis, the imported libraries included NetworkX, Matplotlib, Pandas, NumPy, and Community. NetworkX was used for graph creation and for calculating network properties such as degree centrality, clustering coefficient, and community detection. The matplotlib library is responsible for displaying the output. In the analysis, Pandas was used to read the CSV file.

Visualization of the Chameleon dataset

After reading and pre-processing the dataset, the chameleon dataset contained 1198 nodes and 1805 edges. To understand the connections between nodes and edges, the graph was visualized using Python code. Visualization helps in understanding the structure of the network. With a large number of datasets, the graph would typically generate in a very clustered form, so two graphs were implemented to examine their visualization and purposes. Erdős-Rényi's random graph and Watts-Strogatz Graph were used for implementation. Figure 1 is the original network graph of the chameleon dataset as we have a large number of nodes and edges the graph was clustered. To visualize the dataset more effectively, Erdős-Rényi's random graph and the Watts-Strogatz method were used.

Figure 2 is the Visualization of Erdős-Rényi's Random Network. Erdős-Rényi model is a simple random network graph generated by a fixed number of nodes and assigning edges between pairs of nodes with a certain probability[4] G(n, p) where n and p denotes total number of nodes, and probability of edges connecting between any pairs of nodes. Figure 2 visualizes the nodes and their labels. As the node remains the same Erdős-Rényi's model could identify the labels of the node. Figure 3 is the visualization of the Watts-Strogatz graph of the chameleon dataset. Watts-Strogatz graph generates a random network with a small-world effect[5]. It will capture both regular and random nature from the network. As analysis is carried out using an information network, exploring Watts-Strogatz helps understand real-world networks. Used n, k, p parameters to generate the model, n represent the total collection of nodes, and Each node is linked with 'k ' closest neighbors. P stands for probability of rewiring each edge. Figure 3 shows how edges are connecting the nodes. Both Figures 2 and 3 are created only for visualization. The analysis has been made on the original graph of the chameleon dataset.

Figure 1

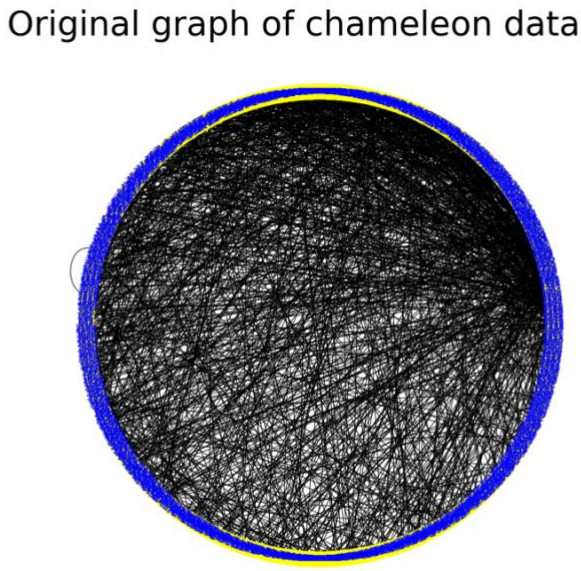


Figure 2

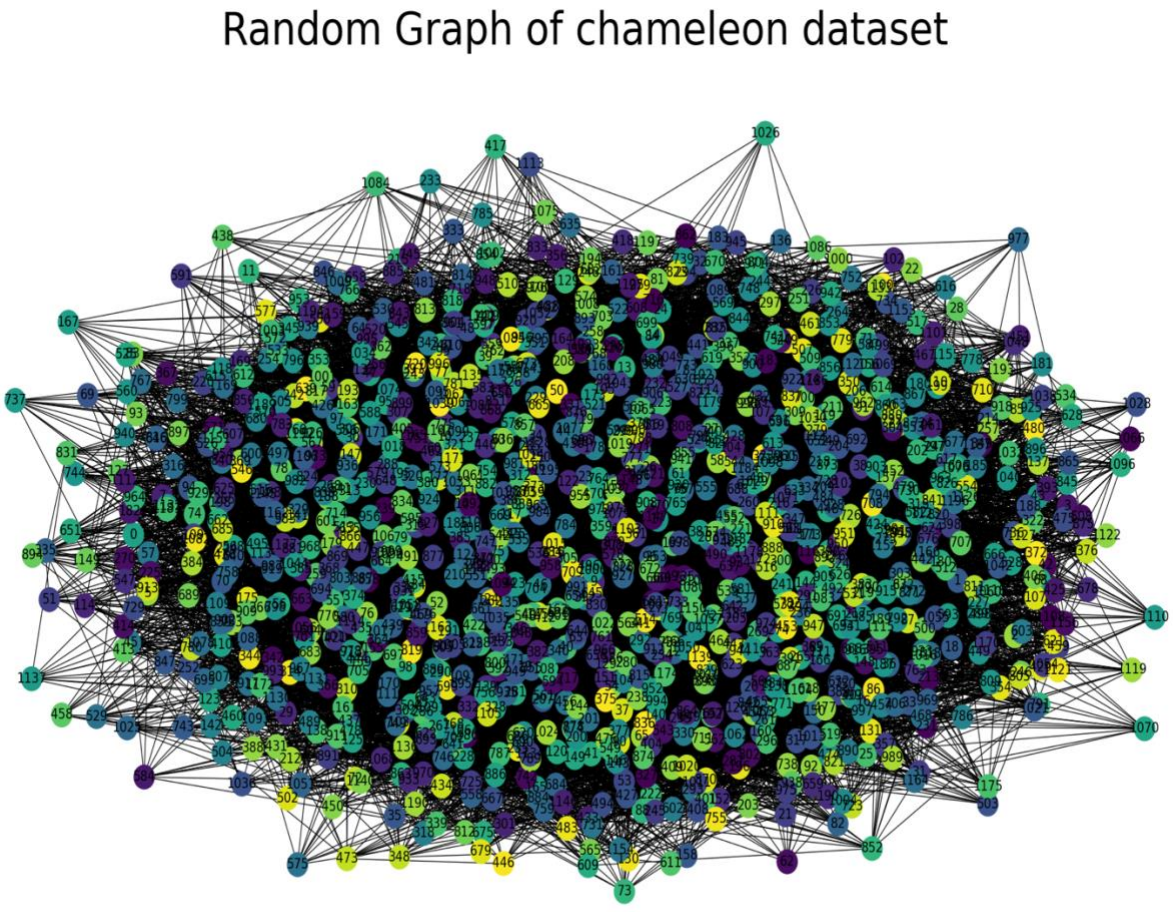
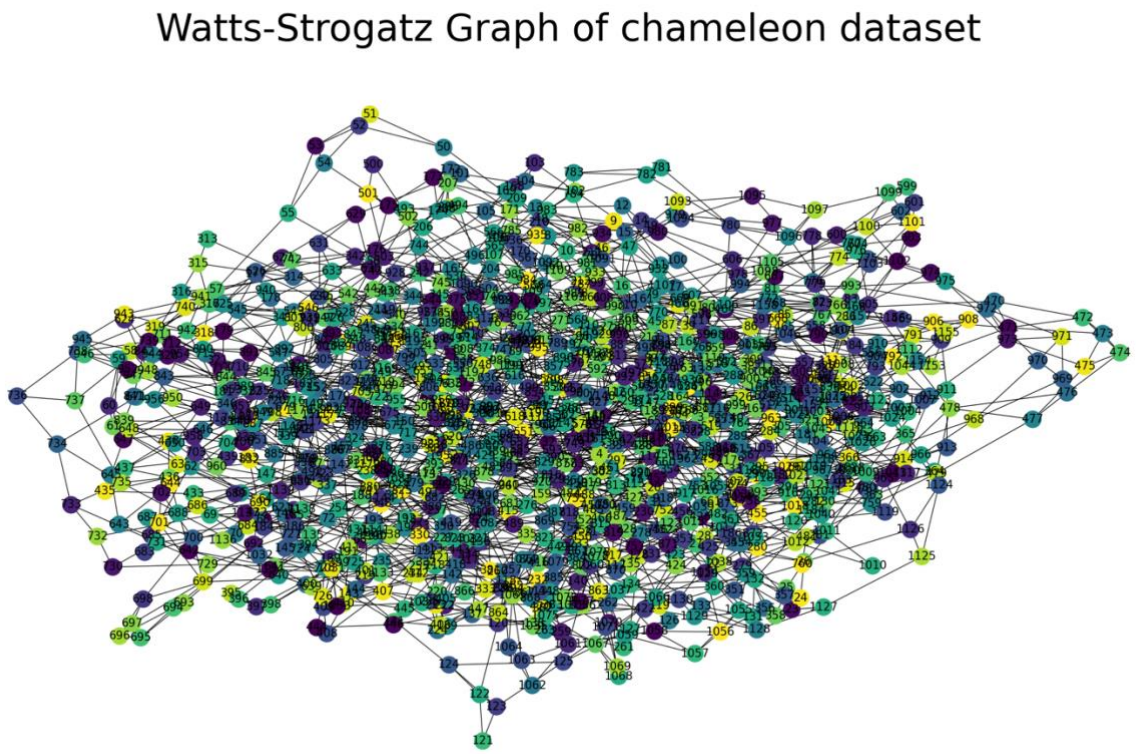


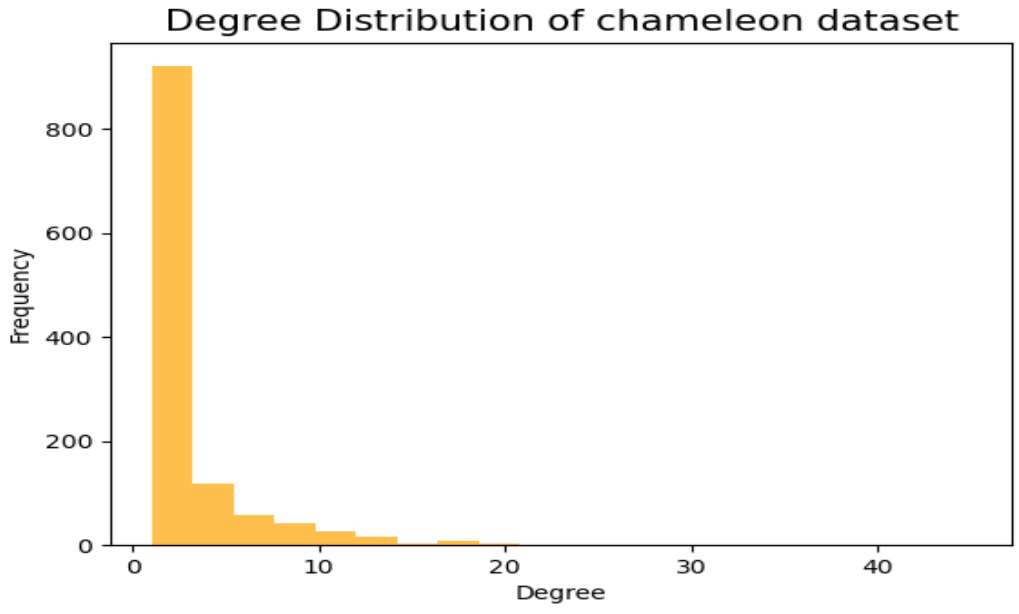
Figure 3



Mean degree

The mean degree method calculates the average connectivity in the entire network. The obtained mean degree is 2.99 this means each node has an average of 2.99 connection to other nodes in the network. For example, each Chameleon article will have an average of 2.99 mutual articles. Figure 4 is the histogram plot for the degree distribution of the chameleon dataset. The x-axis indicates the degree (number of connections a node has), and the y-axis shows the frequency (number of nodes with that degree). The degree 2.99 has the highest frequency.

Figure 4



Density

Density is defined to be the ratio of the real number of edges concerning the maximum number of edges. The below formula calculates the density of the undirected network.

$$\text{Density} = 2*M / N(N-1)$$

Here, M and N represent number of nodes and edges in undirected network. For the chameleon dataset, the obtained density value is 0.0025. This value indicates the level of connections within the chameleon dataset, with a lower density indicating a sparser network.

Degree Centrality Measures

The degree of a node in a network measures the number of edges connected to that node. Nodes with a maximum degree centrality are considered more important in the network as they have more connections. Figure 5 depicts the degree centrality of each node in the Chameleon dataset. In Figure 6, it is specified that node 1976 has the highest degree of centrality with a value of 0.0376. This indicates that node 1976 has the highest number of connections in the entire Chameleon dataset, making it a central and influential node within the network.

Closeness Centrality Measures

Closeness Centrality calculates the closeness of a node to all other nodes in a network. It calculates the average shortest length to other nodes in the network. Figure 5 shows the closeness centrality values of each node in the chameleon dataset. figure 6 points out node 2249 has the highest closeness value of 0.1623. This indicates that node 2249 is well-connected in the network.

Betweenness Centrality Measures

Betweenness Centrality calculates how important a node is for connecting to other nodes. It does this by counting how many of the shortest paths between other points pass through that specific node. Nodes with high betweenness centrality are crucial for maintaining efficient communication and connectivity in the network. figure 5 indicates the betweenness centrality values of each node. Node 2249 has a maximum value of 0.3175

Figure 5

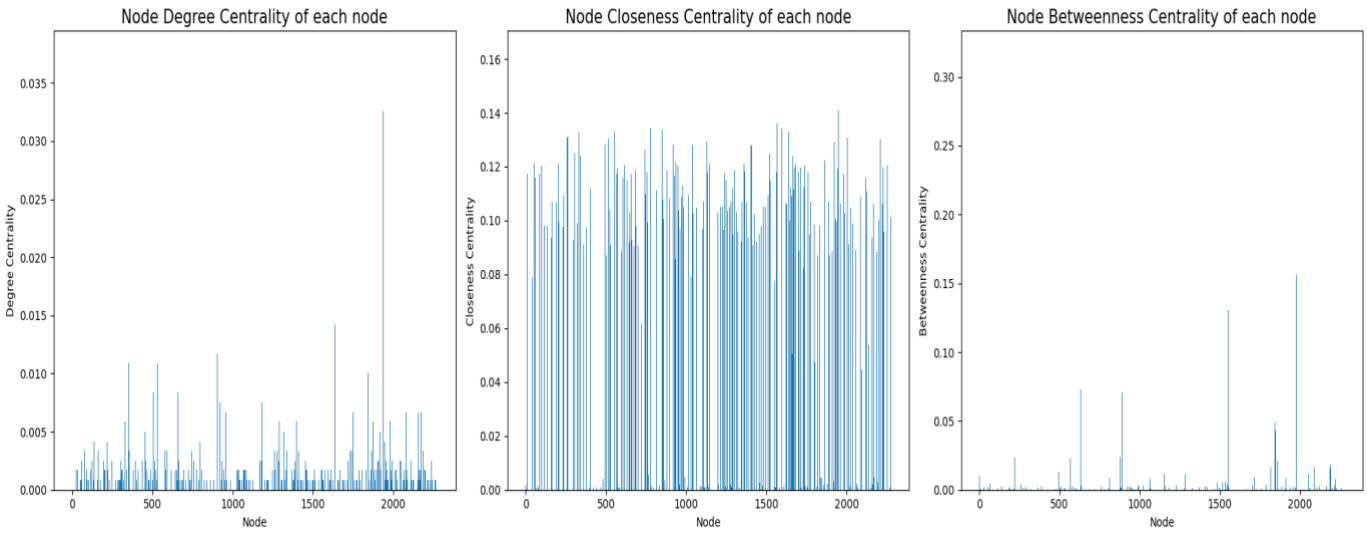
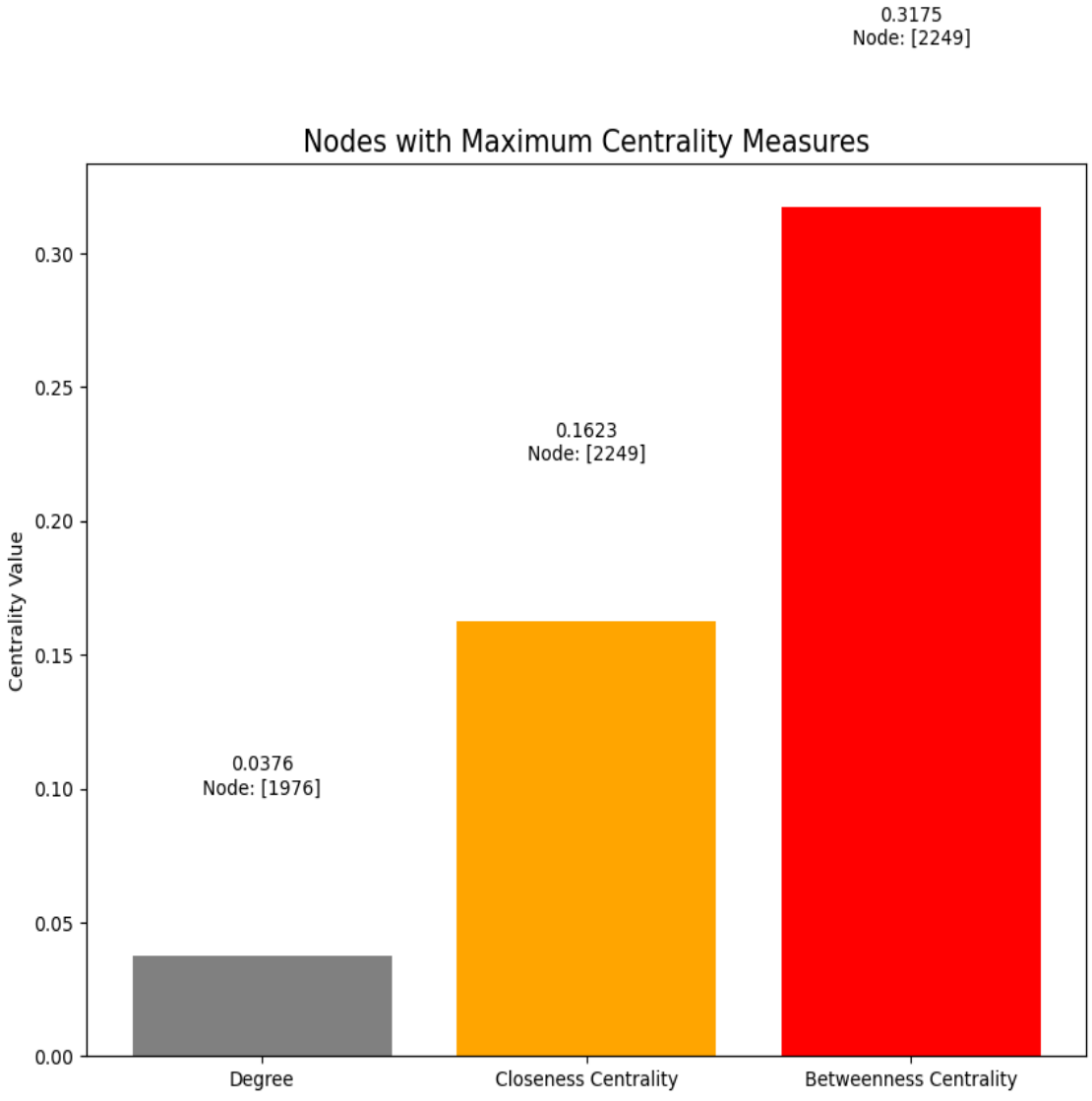


Figure 6

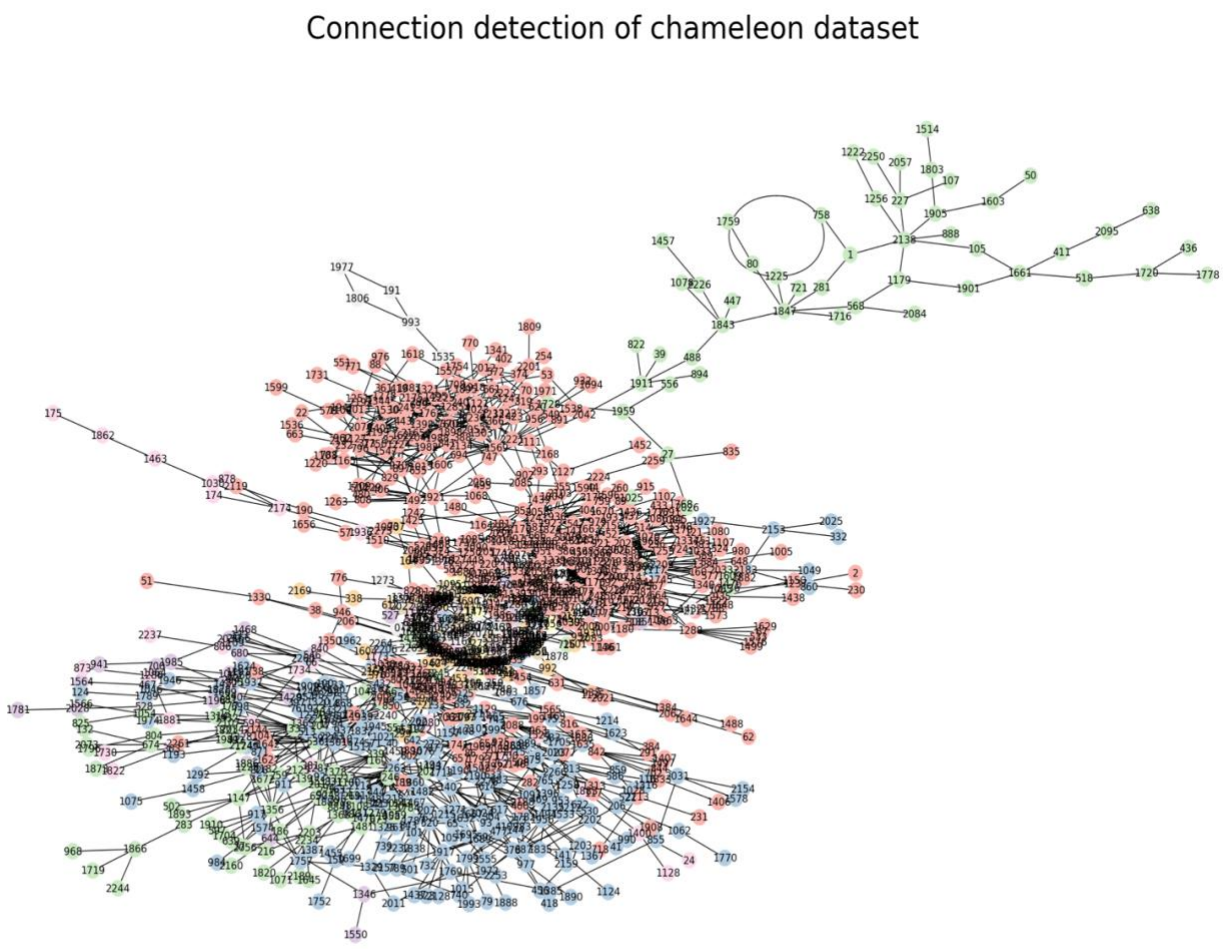


Community Detection Using Louvain Algorithm

Community detection is a method in network analysis that involves identifying groups of nodes in a network that are more connected to each other than to the rest of the network. It helps in understanding the structure within the network and enables the study of small groups separately. By applying community detection to the chameleon dataset, groups within the network were identified. Community Detection uses various algorithms like Louvain, and Girvan-Newman Algorithms, Label Propagation, etc. In this analysis, The Louvain Algorithm is used to detect the communities in the chameleon dataset. It works by iteratively taking single nodes and joining them into groups and so on. Finally, it finds the one that achieves the highest modularity value. Modularity calculates how well nodes are divided into communities. The Louvain algorithm is efficient for larger networks it can easily handle large numbers of nodes and edges and is very effective in finding high-quality communities. Because of the various advantages of the Louvain algorithm, this is method implemented for the Chameleon Network.

Figure 7 represents the graph of Connection Detection using the Louvain Algorithm. Figure 7 shows how communities are found with different colors. A total of 105 communities have been found in the chameleon dataset. From the results, community 6 has 168 nodes. The modularity of the community detection result is 0.7788, where a modularity value greater than 0.7 indicates the strongest community structure. This recommends that the Louvain algorithm has successfully identified groups of nodes that are more massively connected than the rest.

Figure 7



Filtering weak nodes in communities

After discovering 105 communities in the chameleon dataset, the goal is to enhance the community structure and find influential communities within the dataset. To accomplish this, weak nodes within the communities were filtered out, following the concept of node removal from [6]. To remove the weak nodes, a degree centrality value of 0.01 was used to filter them out in the network. After removing the weak nodes from the communities, 14 communities were found in the network. Figure 8 clearly shows the 14 communities with different colors. Lavender-colored community have the highest number of nodes in the community. This information helps find highly influential communities and nodes within the Chameleon network. The modularity of the filtered community is 0.452, indicating a moderate level of the community. The density of the filtered network is 2.4571, indicating that each node has an average of 2.4571 connections to the other nodes. The obtained density value is 0.07226, and the average clustering coefficient value is 0.1623, indicating a moderate level of clustering within the Chameleon dataset. Figure 9 shows the maximum values of degree, Closeness, and Betweenness centrality measures of nodes in the dataset after filtering the weak nodes. Node 2177 has a maximum degree value of 7, node 1567 has a maximum closeness value of 0.301, and node 1923 has a maximum betweenness value of 0.0882.

Overall, after applying the filtering technique to the original community, it was able to find the influential communities and nodes. Influential nodes often act as key communicators or spreaders of information within a chameleon network. Identifying these nodes and communities can help understand how information propagates through the network. It can achieve the robustness of the network and understand its structure, density, degree, closeness, betweenness, and nature of the network.

Figure 8

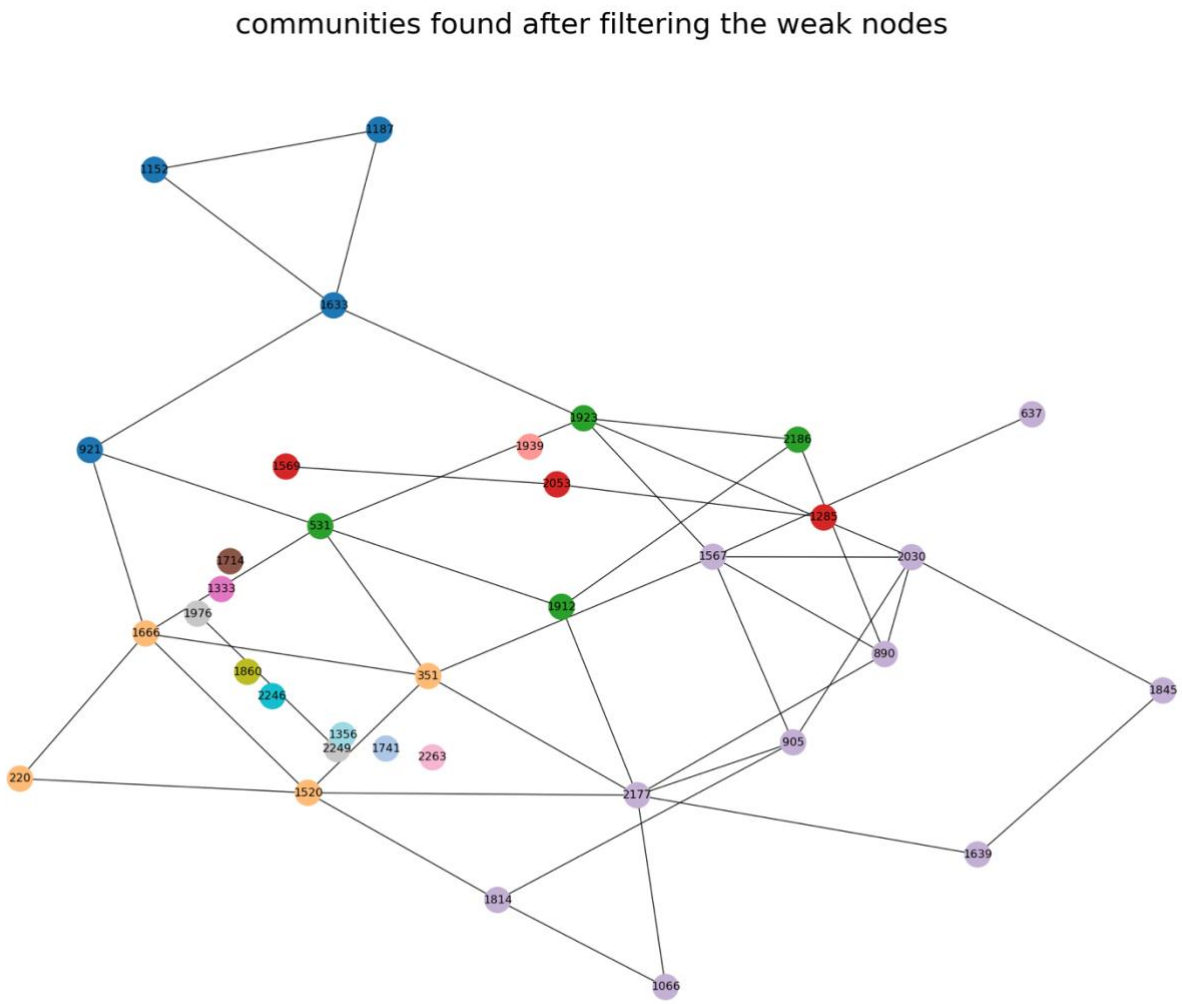
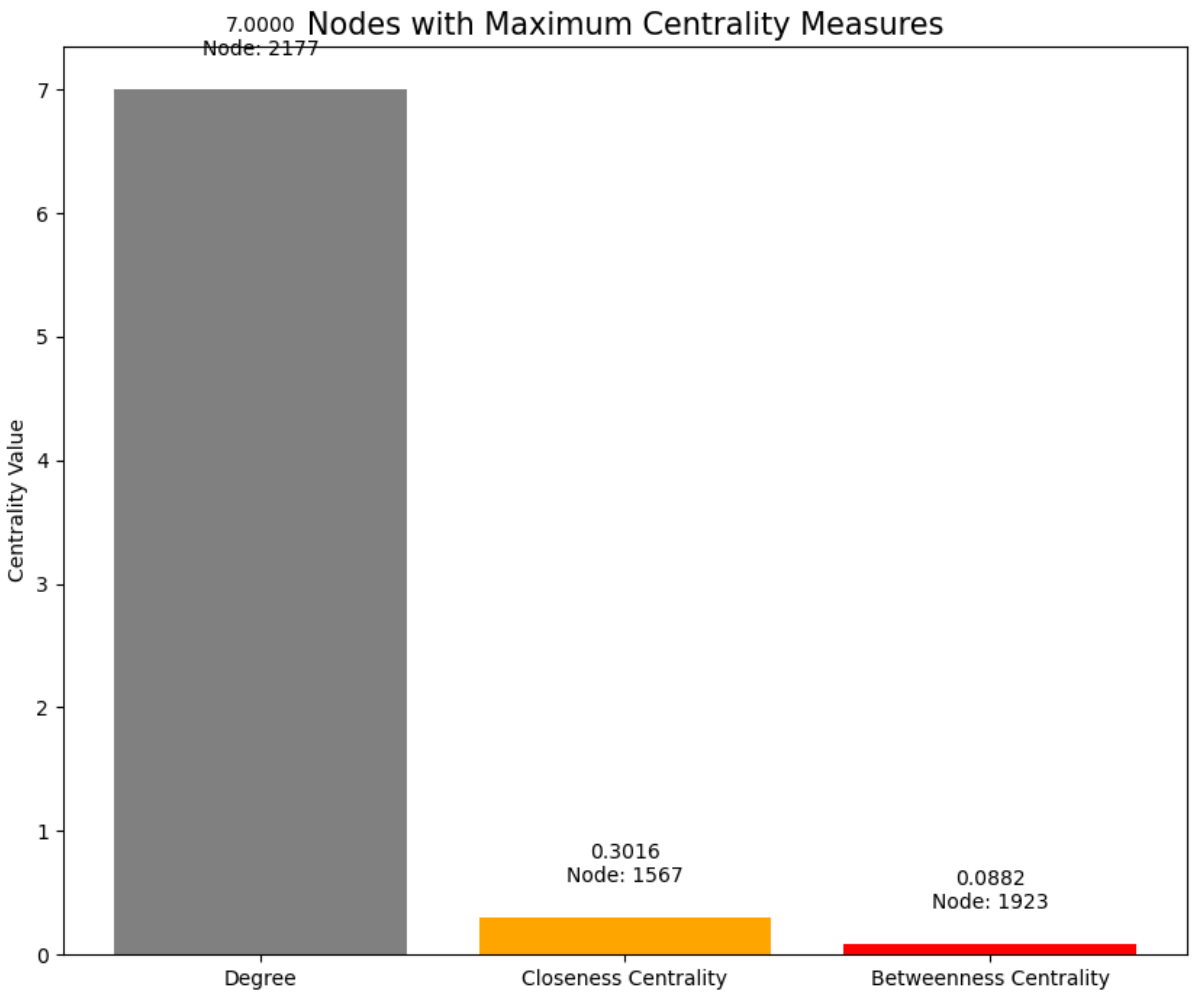


Figure 9


Nodes with Maximum Centrality Measures
7.0000
Node: 2177
0.3016
Node: 1567
0.0882
Node: 1923
Centrality Value
0
1
2
3
4
5
6
7
Degree
Closeness Centrality
Betweenness Centrality

Discussion and Interpretation of Results

Network Graph Visualization

Visualized three types of graphs to present the graph in a better way. In Figure 1 the nodes are in very cluster form because of a large number of datasets. Using Erdős-Rényi and Watts-Strogatz models helped overcome the challenges of visualizing a large network of chameleon datasets. Though Erdős-Rényi and Watts-Strogatz models have different purposes the analysis has been done on the original graph to understand the nature of the network.

Network characteristics

- The mean degree of 2.99 of the original network represents a moderate level of connectivity in the network, showing that, on average, each node is connected to approximately 2.99 other nodes. These findings are important for understanding the normal connectivity patterns in the dataset.
- The density of 0.0025 suggests most nodes are not directly connected in the network also which is common in larger datasets of information network

Centrality Measures

- The degree centrality distribution shows that Node 1976 has the maximum degree value of 0.0376, indicating Node 1976 has the most connections in the network.
- Node 2249 has the maximum Closeness centrality value compared to all other nodes in the network
- The highest betweenness centrality value node is 2249. Identifying these three measures could enable us to understand the nature of the chameleon network

Community Detection using the Louvain Algorithm

- To detect the communities within the network the best algorithm Louvain algorithm is used as it has various advantages like effectiveness, efficiency, etc

- By applying the Louvain algorithm 105 communities have found in the chameleon network. community 6 has the larger nodes also it has achieved the highest modularity value of 0.778. Table 1 indicates that the "Understanding the User Interactions on GitHub: A Social Network Perspective"[2] has achieved the modularity value of 0.467 compared to this value chameleon dataset has achieved a modularity of 0.778 which indicates Our analysis has a strong and well-defined structure.

Table 1

Application	Modularity value
GitHub [2]	0.467
Chameleon Network	0.778

Filtering weak Nodes

after filtering weak nodes with a degree centrality maximum value of 0.01, the analysis focuses on 14 influen tial communities found in the network. The density of 2.4571 suggests increased connectivity within the filte red network. The modularity value is 0.4524 suggesting a well-structured network. The Lavender color com munities have the highest number of nodes. The degree and closeness of betweenness show the quality of the network

Overall Observations

- The obtained results suggest that the Chameleon dataset shows a complex and decentralized structure , with influential nodes and well-defined communities.
- Analysing the centrality measures reveals main nodes that play crucial roles in information flow and network connectivity.
- The application of the Louvain algorithm helps in understanding the natural groupings of related artic les about a chameleon. It also plays an important role in the in-depth analysis of the Chameleon datas et

Future Work

Additional Dataset

Performing Analysis without processing the chameleon dataset can identify more communities as it has a lar ger number of nodes 2,277 and edges 31,421. Similar to the chameleon dataset, datasets of crocodiles and sq uirrels are found in [3]. Exploring the same type of application would be able to achieve different network ch aracteristics and community structures.

Subset Visualization

Subset Visualization [7] is an approach to focus on one community. For instance, Community 106 has the lar gest number of nodes. performing subset visualization on Community 106, allows us to understand more det ailed information about that particular community.

Link Prediction

Link prediction [8] is the possible task for analyzing the chameleon dataset. Link prediction, also known as link forecasting, is a crucial task in network analysis that helps to identify important links (edges) that may form between nodes. By applying this method in the chameleon dataset it will find the important mutual article(edge) in the network. The prediction is done by following approaches Similarity-based methods, Maximum likelihood methods, and Probabilistic methods.

K Core Cliques

Working on k-core cliques in the future will make our analysis worthwhile and help us understand how the network works. K-core cliques are like tightly-knit groups within the larger network, and understanding them can tell us more about how different parts of the network are strongly connected. It's like looking at smaller communities within the big picture and figuring out how they impact the entire network. This method will

provide us with more detailed information about specific groups of nodes and their importance in shaping the overall structure of the network.

By addressing these points in future work, the analysis of the chameleon dataset can be further refined, offering a deeper understanding of its structural characteristics, dynamics, and details.

Conclusion

The current analysis provides helpful information about the Chameleon dataset's network structure, characteristics, and community organization. Finding influential communities within the Chameleon network helps understand information flow and enhances the robustness of the network, this strength is important because it helps the network to handle different ways information moves around without getting easily affected. especially the application of the Louvain algorithm, which not only contributes to the understanding of the Chameleon dataset but also provides insights applicable to real-world scenarios. Understanding natural groupings and influential nodes has implications for various fields such as social network analysis, information retrieval, and content recommendation systems. the high modularity values obtained through the Louvain algorithm and community filtering signify a well-structured network. Further exploration with additional datasets and detailed subset visualizations could enhance the understanding of target communities in network dynamics and contribute to broader applications in network analysis research. Predicting links is useful in real-life situations. It helps us find important connections in the network, making it easier to recommend specific content, suggest collaborations, or understand possible links between different articles. Looking into K-core cliques in the future will help us understand the network better. It's like looking closely at groups of nodes that are strongly connected within the larger network. This can reveal more detailed information about how different parts of the network are linked, giving us a clearer picture of how the network works. The findings can be useful in real-life situations. By exploring into dynamics of complex network work, we can discover patterns and connections that go beyond just the Chameleon dataset. This opens up possibilities for making progress and improvements in the field.

References

[1] Jebabli, Malek, Cherifi, Hocine, Cherifi, Chantal, & Hamouda, Atef. (November 2015). "Overlapping Community Detection Versus Ground-Truth in the Amazon Co-Purchasing Network." A Conference Paper.

[2] Fu, Erzheng, Zhang, Jiayun, & Chen, Yang. (May 2021). "Understanding the User Interactions on GitHub: A Social Network Perspective." A Conference Paper.,

[3] Rozemberczki, B., Allen, C., & Sarkar, R. (2019). "Multi-scale Attributed Node Embedding." arXiv preprint arXiv:1909.13021. [Link to Paper](https://snap.stanford.edu/data/ the English Wikipedia) (December 2018).

[4] Newman, M. (2018). "Oxford University Press."

[5] Watts, Duncan J., & Strogatz, Steven H. (1998). "Collective dynamics of 'small-world' networks."

[6] Estrada, Ernesto. (2012). "Complex Networks: An Introduction."

[7] Hidalgo, Cesar. (2015). "Introduction to Network Science."

[8] Adamic, Lada A., & Huberman, Bernardo A. (2005). "Link prediction in social networks."

[9] NetworkX Documentation , https://networkx.org

Appendix

```
#import  libraries
import networkx as nx
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import community  as com
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = 'all'
#read the dataset took this dataset from [3]
df = pd.read_csv(r'/Users/nithyashree/Desktop/musae_chameleon_edges.csv')
 # Check the first few rows of the df
print(df.head())


# Check if the chameleon network is directed or undirected
e = df[['id1', 'id2']]
check_directed = any(e.duplicated() | e[['id1', 'id2']].duplicated())


#condition clause to print the network
if check_directed:
    print("The chameleon network is directed.")
else:
    print("The chameleon network is undirected.")
#preprocessing the dataset
#performing random sampling by selecting 5 % of rows from the original dataset
random_dfsampling = df.sample(frac=0.05, random_state=42)
```

```
# Get the no of nodes and edges after the sampling process

number_of_nsampled = pd.concat([random_dfsampling['id1'], random_dfsampling['id2']]).nunique()

number_of_esampled = len(random_dfsampling)

#display the results

print(f"Number of nodes in random sampled dataset: {number_of_nsampled}")

print(f"Number of edges in random sampled dataset: {number_of_esampled}")

#line is to convert the edge information from the df into a graph

#id1 and id2 are column names in the dataset

Graph_net = nx.from_pandas_edgelist(random_dfsampling, 'id1', 'id2')

#to execute this code I took references from labs 8 and 9

# Generating both a random graph and a Watts-Strogatz graph to understand how the graph is structured in different ways

# Number of edges for 2% connectivity in the graph

n = len(Graph_net.nodes)

numberof_e_random = 0.02 * n * (n - 1) // 2


# Used Erdős-Rényi model for visualizing the graph in a better way

# Create a random graph using gnm_random_graph

#set seed to generate a graph with the same structure every time

renyi_randomgraph = nx.gnm_random_graph(n, numberof_e_random, seed=1347)

#to execute this code I took a reference from lab 9

# Create Watts-Strogatz Graph

n = len(Graph_net.nodes)

# Each node is connected to its closest  neighbors

k = 4

# Probability of rewiring each edge

p = 0.2

#generate the graph

watts_graphnet = nx.watts_strogatz_graph(n, k, p)

# Choose a spring layout for better visualization

layout = nx.spring_layout(watts_graphnet)
```

```
# Generate a circular layout for the original graph

layout_circ = nx.circular_layout(Graph_net)

# Generate random colors for nodes

random_ncolours = np.random.rand(len(renyi_randomgraph))

random_nodeclours = np.random.rand(n)

# Draw the original graph with a circular layout

plt.figure(figsize=(25, 15))

plt.title("Original graph of chameleon dataset", fontsize= 50)

#draw the graph with node size, labels, font size, and color

nx.draw(Graph_net, pos=layout_circ, with_labels=True, node_size=500, font_size=12, font_color= "blue", node_color="yellow", edge_color='black')

plt.axis('equal')

plt.savefig("/Users/nithyashree/Desktop/agraph_chameleonh.png", dpi=100)

plt.show()


# Random Graph Visualization

plt.figure(figsize=(25, 15))

plt.title("Random Graph of chameleon of chameleon dataset", fontsize= 50)

#draw the graph with node size, labels, font size, and color

nx.draw(renyi_randomgraph, node_size=700, with_labels=True, font_size=15, node_color=random_ncolours)

#used tight layout to give spaces between graphs and avoid overlapping

plt.tight_layout()

plt.savefig("/Users/nithyashree/Desktop/aerdos_grap_chameleonh.png", dpi=100)

plt.show()


# Watts-Strogatz Graph Visualization

plt.figure(figsize=(25, 15))

plt.title("Watts-Strogatz Graph of chameleon of chameleon dataset", fontsize= 50)

#draw the graph with node size, labels, font size, and color

nx.draw(watts_graphnet, pos=layout, with_labels=True, font_size=15, node_size=500, node_color=random_nodeclours)

#used tight layout to give spaces between graphs and avoid overlapping
```

```
plt.tight_layout()
plt.savefig("/Users/nithyashree/Desktop/aawatts_graphnet_chameleon.png", dpi=100)
plt.show()
# Compute degrees for the original graph
graph_degrees = dict(nx.degree(Graph_net))
# Mean Degree for original Graph
mean_graph_degree_nx = sum(graph_degrees.values()) / len(graph_degrees)
# Display Mean Degree
print("Mean Degree of chameleon (NetworkX):", mean_graph_degree_nx)
#Histogram Plot for original Graph of chameleon Degree Distribution
#hist keyword is used for plotting
plt.hist(list(graph_degrees.values()), bins=20, color='Orange', alpha=0.7)
#title for plot
plt.title("Degree Distribution of chameleon dataset", fontsize= 15)
#  x-axis label
plt.xlabel("Degree")
# y-axis label
plt.ylabel("Frequency")
plt.savefig("/Users/nithyashree/Desktop/aameandegree.png", dpi=100)
plt.show()
# Calculate the density of the chameleon dataset
density_data = nx.density(Graph_net)
# Display the density
print(f"Density of chameleon dataset: {density_data:.4f}")
# Calculate the clustering coefficient of the chameleon dataset
clusteringcoefficient_data = nx.average_clustering(Graph_net)
#display the result
print("Average Clustering Coefficient of Chameleon dataset:", clusteringcoefficient_data)
# Calculate degree centrality for each node
d_graph = dict(nx.degree_centrality(Graph_net))
```

```
# Display the degree centrality  measures of each node in the chameleon dataset

print("The closeness centrality of each node:")

print(d_graph)

# Calculate closeness centrality for each node  in the chameleon dataset

c_graph = dict(nx.closeness_centrality(Graph_net))

# Display the closeness centrality measures of each node in the chameleon dataset

print("The closeness centrality of each node:")

print(c_graph)

 # Calculate betweenness centrality measures for each node

b_graph=dict(nx.betweenness_centrality(Graph_net))

# Display the betweenness centrality of each node in the chameleon dataset

print("The betweenness centrality of each node:")

print(b_graph)

# Save degree, closeness, and betweenness centralities  values in a data frame

chameleon_df=pd.DataFrame.from_dict([d_graph, c_graph, b_graph]).T

chameleon_df.columns = ["degree", "closeness", "betweenness"]

# Create subplots

fig, (G1, G2, G3) = plt.subplots(1, 3, figsize=(20, 6))


# Bar graph for degree centrality

G1.bar(chameleon_df.index, chameleon_df['degree'])

G1.set_title('Node Degree Centrality of each node', fontsize= 15)

G1.set_xlabel('Node')

G1.set_ylabel('Degree Centrality')


# Bar graph for closeness centrality

G2.bar(chameleon_df.index, chameleon_df['closeness'])

G2.set_title('Node Closeness Centrality of each node', fontsize= 15)

G2.set_xlabel('Node')

G2.set_ylabel('Closeness Centrality')
```

```
# Bar graph for betweenness centrality
G3.bar(chameleon_df.index, chameleon_df['betweenness'])
G3.set_title('Node Betweenness Centrality of each node', fontsize= 15)
G3.set_xlabel('Node')
G3.set_ylabel('Betweenness Centrality')


# Adjust layout
plt.tight_layout()
plt.savefig("/Users/nithyashree/Desktop/aacentrality.png", dpi=100)
# Display the plots
plt.show()
# Display the values
print("Values Degree, Closeness, and Betweenness Centralities:")
print(chameleon_df)
#save the data frame in a CSV file
chameleon_df.to_csv('/Users/nithyashree/Downloads/acentrality.csv', encoding='utf-8')
#Compute the nodes with maximum values of the degree, closeness, and betweenness centrality measures of chameleon dataset
maxi_d = max(d_graph.values())
maxi_c=max(c_graph.values())
maxi_b =max(b_graph.values())
#create a plot for maximum values
fig, g1 = plt.subplots(figsize=(10, 8))
maximum_deg_key=[items for items, value in d_graph.items()
        if value == max(d_graph.values())]
maximum_clos_key=[items for items, value in c_graph.items()
        if value == max(c_graph.values())]
maximum_betw_key=[items for items, value in b_graph.items()
        if value == max(b_graph.values())]
# Plotting the values of degree, closeness, centrality
```

```
g1.bar(['Degree', 'Closeness Centrality', 'Betweenness Centrality'],
       [maxi_d, maxi_c maxi_b], color=['grey', 'orange', 'red'])
#set the title
g1.set_title('Nodes with Maximum Centrality Measures', fontsize= 15)
g1.set_ylabel('Centrality Value')
#label for nodes on the bar
for i, (max_value, key_list) in enumerate(zip([maxi_d, maxi_c, maxi_b],
                                              [maximum_deg_key, maximum_clos_key, maximum_betw_key])):
    g1.text(i, max_value + 0.05, f'{max_value:.4f}\nNode: {key_list}\n', ha='center', va='bottom')


plt.savefig("/Users/nithyashree/Desktop/aamaximum.png", dpi=100)
#visualize the plot
plt.show()
#Display the maximum values
print("The maximum degrees  centrality value  of node is/are ", maximum_deg_key,
      ". The maximum degree is", maxi_d)
print("The maximum closeness centrality value  of node is/are ", maximum_clos_key,
      ". The maximum closeness centrality is", maxi_c)
print("The  maximum betweenness centrality value of node is/are ", maximum_betw_key,
      ". The maximum betweenness centrality is", maxi_b)
#install Louvain package
pip install python-louvain
#https://networkx.org/documentation/stable/
#for plotting the kamada_kawai_layout and  Louvain algorithm for community detection
#refered this code from this website[9]
# Apply the Louvain algorithm for community detection


partition_louvain = com.best_partition(Graph_net)


# Create a set to store community IDs
```

```
uniq_commu = set(partition_louvain.values())
# Print the number of communities found in the dataset
num_communities = len(uniq_commu)
print(f"Number of communities found: {num_communities}")
# Create a dictionary function  to store the number of nodes in each community
comm_sizes = {}
for n, cid in partition_louvain.items():
    if cid not in comm_sizes:
        comm_sizes[cid] = 1
    else:
        comm_sizes[cid] += 1
# Print the community formed and the number of nodes for each community that was formed
for cid, s in comm_sizes.items():
    print(f"Community {cid} has {s} nodes")
# Calculate modularity
modularity_louvain = com.modularity(partition_louvain, Graph_net)
print(f"Modularity of Community Detection: {modularity_louvain}")
#create graph a with kamada_kawai layout
layout_louvian = nx.kamada_kawai_layout(Graph_net)
# Use distinct color palettes for communities to identify the communities
clr_palette = plt.cm.Pastel1
#get community colors
clrs = [partition_louvain[node] for node in Graph_net.nodes()]
#assign the size for the figure
plt.figure(figsize=(25, 15))
# draw the graph
# Generate a graph for community detection
nx.draw(Graph_net, pos=layout_louvian, node_color=clrs, cmap=clr_palette, with_labels=True)
plt.title("Connection detection of chameleon dataset", fontsize= 35)
plt.savefig("/Users/nithyashree/Desktop/aaommunity.png", dpi=100)
```

```
plt.show()
# calculating degree
#by getting degree values to filter the weak nodes
degree_weak = nx.degree_centrality(Graph_net)
print(degree_weak)
#using degree values chosen 0.01 for removing the weak nodes in the community
# Print the number of communities found before filtering the weak nodes from the original communities
node_filtering = len(uniq_commu)
print(f"Number of communities found before filtering: {node_filtering}")
# Choose a maximum value for degree centrality
d_max = 0.01
# Compute degree centrality for each node
d_node = nx.degree_centrality(Graph_net)
# remove or filter the weak nodes based on degree centrality maximum value
filter_nodes = [node for node, centrality in d_node.items() if centrality >= d_max]
# Create a subgraph after removing the weak nodes
filter_sgraph = Graph_net.subgraph(filter_nodes)
# apply if the condition to check whether the subgraph is  empty or  not  before applying the Louvain algorithm
if filter_sgraph.number_of_nodes() > 0:
# Apply the Louvain algorithm for community detection on the filtered graph
    weak_filtered = com.best_partition(filter_sgraph)


    # Convert the community partition to a dictionary to perform modularity
    plouvain_dict = dict(weak_filtered)
    # Calculate modularity
    modularity_value = com.modularity(plouvain_dict, filter_sgraph)
    # Print the modularity value
    print(f"Modularity after removing weak nodes: {modularity_value}")
   # to get the number of communities formed after removing the weak nodes
    uni_community = set(weak_filtered.values())
```

```
    # Print the number of communities
    afternode_filtering = len(uni_community)
    print(f"Number of communities found after Removing weak nodes: {afternode_filtering}")
    # Choose a layout
    layout_algo = nx.kamada_kawai_layout(filter_sgraph)
 # Use distinct color palettes for communities to identify the different communities in the graph
    if 'weak_filtered' in locals():
        color_palette = plt.cm.tab20
        # get community colors
        colors = [weak_filtered[node] for node in filter_sgraph.nodes()]
    else:
        color_palette = 'red'
 # Plot the subgraph with color, node size, and labels
    plt.figure(figsize=(20, 15))
    # draw the plot
    nx.draw(
        filter_sgraph,
        layout_algo,
        node_color=colors,
        with_labels=True,
        cmap=color_palette,
        node_size=700,
        alpha=1,
    )
    plt.title("communities found after filtering  the weak nodes", fontsize= 30)
    plt.savefig("/Users/nithyashree/Desktop/aacommunityfilter.png", dpi=100)
    plt.show()
else:
    print("No communities found after filtering.")
 #calculate the degree for filtered communities
```

```
G_degrees = dict(nx.degree(filter_sgraph))

# Mean Degree for filtered communities Graph

mean_graph = sum(G_degrees.values()) / len(G_degrees)

# Display Mean Degree for filtered communities Graph of the chameleon dataset

print("Mean Degree of chameleon after filtering :", mean_graph)

# Calculate degree centrality for each node

d_center = nx.degree_centrality(filter_sgraph)

print("Degree Centrality:")

for node, centrality in d_center.items():

    print(f"Node {node}: {centrality}")

# compute closeness centrality measures for each node

c_center = nx.closeness_centrality(filter_sgraph)

print("\nCloseness Centrality:")

for node, centrality in c_center.items():

    print(f"Node {node}: {centrality}")

# Comute betweenness centrality measures for each node

b_center = nx.betweenness_centrality(filter_sgraph)

print("\nBetweenness Centrality:")

for node, centrality in b_center.items():

    print(f"Node {node}: {centrality}")

density_filtered = nx.density(filter_sgraph)

print(f"Density : {density_filtered}")

#Calculate the clustering coefficient for the filtered subgraph for communities

clustering_ccommunity = nx.average_clustering(filter_sgraph)

print(f"Average Clustering Coefficient : {clustering_ccommunity}")

# Compute degree, closeness centrality, and betweenness centrality

degrees_gh = dict(filter_sgraph.degree())

closeness_gh = nx.closeness_centrality(filter_sgraph)  # Use filter_sgraph here

betweenness_gh = nx.betweenness_centrality(filter_sgraph)  # Use filter_sgraph here
```

```
# Compute maximum values and corresponding nodes
degreekey = max(degrees_gh, key=degrees_gh.get)
closenesskey = max(closeness_gh, key=closeness_gh.get)
betweennesskey = max(betweenness_gh, key=betweenness_gh.get)

maxi_degree = degrees_gh[degreekey]
maxi_closeness = closeness_gh[closenesskey]
maxi_betweenness = betweenness_gh[betweennesskey]

# Create a plot to maximum values
fig, a1 = plt.subplots(figsize=(10, 8))

# Plotting the maximum values of degree, closeness centrality, and betweenness centrality
a1.bar(['Degree', 'Closeness Centrality', 'Betweenness Centrality'],
       [maxi_degree, maxi_closeness, maxi_betweenness], color=['grey', 'orange', 'red'])

# Set the title
a1.set_title('Nodes with Maximum Centrality Measures', fontsize= 15)
a1.set_ylabel('Centrality Value')

#label for nodes on the bar
for i, (max_value, key_list) in enumerate(zip([maxi_degree, maxi_closeness, maxi_betweenness],
                                              [degreekey, closenesskey, betweennesskey])):
    a1.text(i, max_value + 0.05, f'{max_value:.4f}\nNode: {key_list}\n', ha='center', va='bottom')
plt.savefig("/Users/nithyashree/Desktop/aacenter.png", dpi=100)
# Visualize the plot
plt.show()

# Display the maximum values
print("The node with the maximum degree is", degreekey, "with a degree of", maxi_degree)
```

```
print("The node with the maximum closeness centrality is", closenesskey,
    "with a closeness centrality of", maxi_closeness)
print("The node with the maximum betweenness centrality is", betweennesskey,
    "with a betweenness centrality of", maxi_betweenness)
```