Q7 readme

a. Please calculate and plot the clustering coefficient and degree distribution of the network.

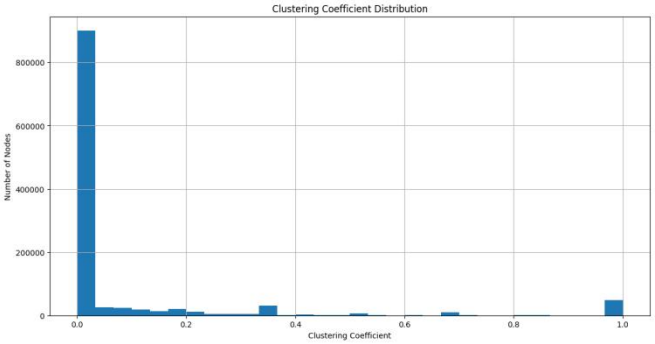
The code calculates the clustering coefficient for each node in the network G using the nx.clustering(G) function and stores the results in the variable clustering_coefficients. It also calculates the degree for each node in the network G using the G.degree(n) function and stores the results in the degrees list. And the averages are 0.0808 and 5.265.

```
# Clustering coefficient
clustering_coefficients = nx.clustering(G)
# Degree distribution
degrees = [G.degree(n) for n in G.nodes()]
```

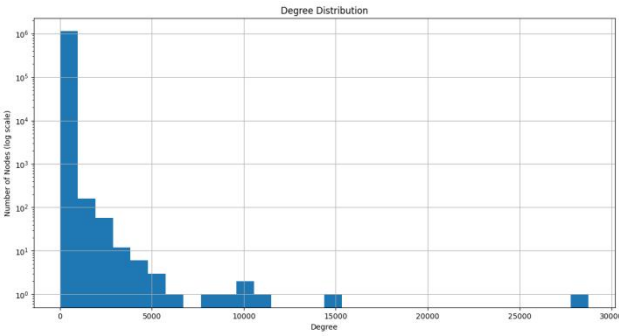
```
average_clustering_coefficient = sum(clustering_coefficients.values()) / len(G.nodes())
average_degree = sum(degrees) / len(G.nodes())

print("average_clustering_coefficient:", average_clustering_coefficient)
print("average_degree:", average_degree)
```

```
average_clustering_coefficient: 0.08080227776187864
average_degree: 5.265045951590022
```

Additionally, there are two histograms. The first histogram represents the clustering coefficient distribution, where the x-axis represents the range of clustering coefficient values and the y-axis represents the number of nodes with corresponding clustering coefficients. The second histogram represents the degree distribution, where the x-axis represents the range of degree values, and the y-axis represents the number of nodes with corresponding degrees (using a logarithmic scale for easier visualization).



the clustering coefficient



the degree distribution

b. Identify the most influential nodes in a network and analyze them.

Please use centrality metrics such as degree centrality and then visualize.

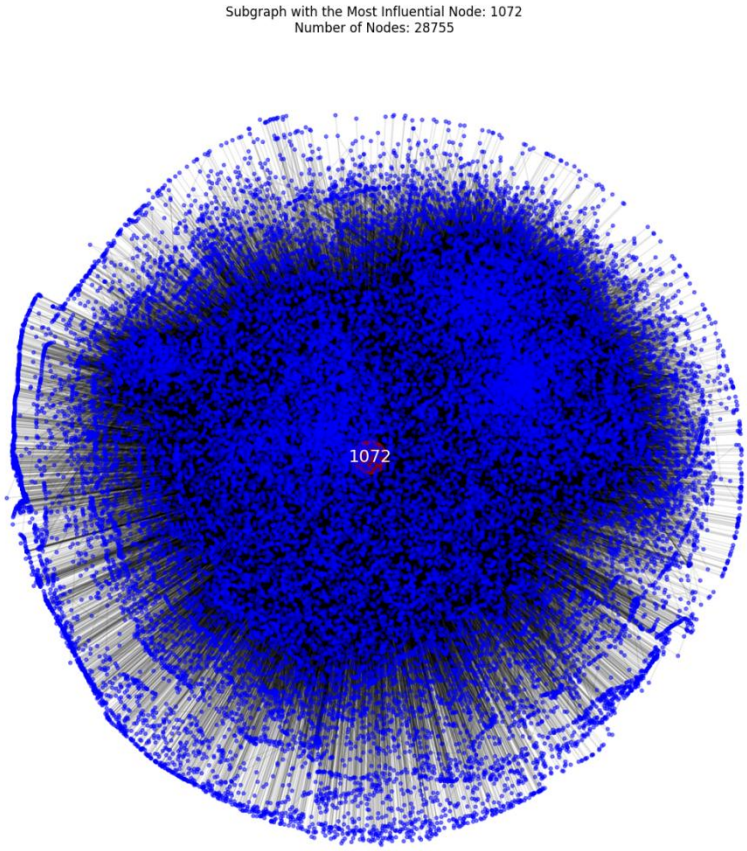
I have calculated the degree centrality for each node with nx.degree_centrality(G) function and stores the results in the degree_centrality variable. Then, sort the nodes based on their degree centrality values in descending order. The influential_nodes variable will contain a list of nodes sorted by their degree centrality, with the most influential nodes appearing at the beginning of the list.

```
degree_centrality = nx.degree_centrality(G)
# Sort nodes by degree centrality
influential_nodes = sorted(degree_centrality.items(), key=lambda item: item[1], reverse=True)
```

```
influential_nodes[:5]
```

```
[(1072, 0.025336398537654344),
 (363, 0.012900821137573806),
 (35661, 0.009940179171707541),
 (106, 0.009217641549085418),
 (482709, 0.008601722282972167)]
```

Due to the large size of the generated network, I have chosen to display a subgraph consisting of the most influential node and its neighbors. In the graph, node 1072 is identified as the most influential node. The subgraph, which includes the most influential node itself and its neighboring nodes, comprises a total of 28,755 nodes.



c. Identify Isolated Nodes in the Network

```
print("The number of unique users: ", num_unique_users)
print("The maximum ID: ", max_id)
print(f"Number of isolated users: {max_id-num_unique_users}")
```

```
The number of unique users:  1134890
The maximum ID:  1157827
Number of isolated users: 22937
```

d. Recognize Connected Components in the Network

1. `connected_components = list(nx.connected_components(G))`

```
for i, component in enumerate(connected_components):
    num_nodes = len(component)
    print(f"Connected Component {i+1}: Number of Nodes = {num_nodes}")
```

```
Connected Component 1: Number of Nodes = 1134890
```

e. Compute Average Shortest Path Length of the Network.

Due to the infeasibility of using the function nx.average_shortest_path_length(G) for a large network, we can randomly select 1000 nodes to estimate the average shortest path length of the network.

```
#Using approximation method
import random
def approximate_average_shortest_path_length(G, sample_size=1000):
    if sample_size > len(G):
        sample_size = len(G)
    random_nodes = random.choices(list(G.nodes()), k=sample_size)
    lengths = []
    for node in random_nodes:
        path_lengths = nx.single_source_shortest_path_length(G, node)
        lengths.extend(path_lengths.values())
    return sum(lengths) / len(lengths)

approx_avg_path_length = approximate_average_shortest_path_length(G)
approx_avg_path_length
#output:5.256447320885725
```

The average shortest path length of the network is about 5.3.

f. Calculate the Diameter of the Network

Similar to the function nx.diameter(G), which might not be feasible for a large network, we randomly selected 1000 nodes. Taking into account the possibility of disconnected nodes, we used G.subgraph(nx.node_connected_component(G, n)) to obtain the connected subgraph starting from each randomly selected node. This ensured that only the connected portion was considered when calculating the eccentricity.

```
def approximate_diameter(G, sample_size=1000):
    if sample_size > len(G):
        sample_size = len(G)
    random_nodes = random.choices(list(G.nodes()), k=sample_size)
    eccentricities = (nx.eccentricity(G.subgraph(nx.node_connected_component(G, n)), n) for n in r
andom_nodes)
    return max(eccentricities)

approx_diameter = approximate_diameter(G)
approx_diameter
#output:20
```

The diameter of the network is about 20.

g. Detect Community Structures in the Network.

Please employ community detection algorithms (e.g., Louvain algorithm) to find community structures within the network. Please analyze the detection results, including factors such as the number of communities and statistics within the community. Assess the impact of relevant parameters(if any). Try your best to visualize the detection results.

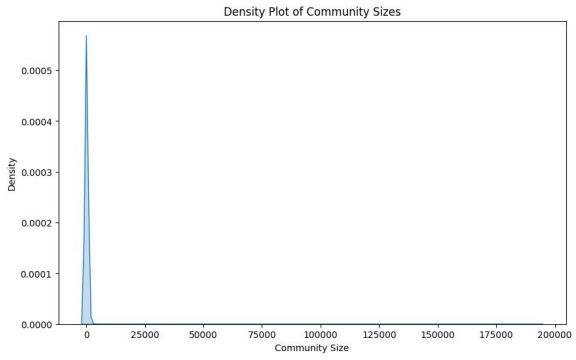
Using the Louvain algorithm, the community structure in the network was identified. The number of detected communities is 5769, and the modularity of the partition is approximately 0.7217. Additionally, we can determine the number of nodes in each community and calculate the density. A density plot can be generated to visualize the distribution of community sizes and densities.

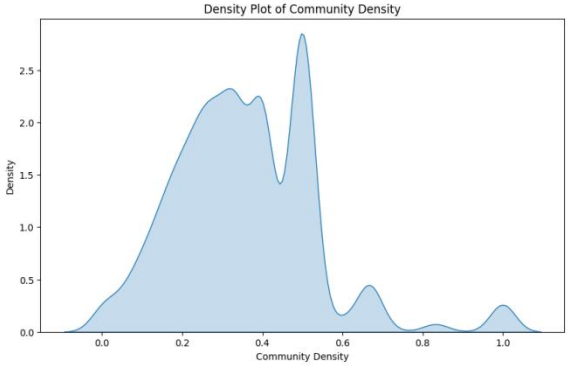
```
import networkx as nx
from community import community_louvain

num_communities = len(set(partition.values()))
community_sizes = [list(partition.values()).count(i) for i in range(num_communities)]
print(f"Number of communities detected: {num_communities}")
print("Sizes of communities:", community_sizes)
```

```
modularity = community_louvain.modularity(partition, G)
print(f"Modularity of the partition: {modularity}")

community_density = []
for community_id in range(num_communities):
    nodes_in_community = [node for node, c_id in node_community_dict.items() if c_id == communit
y_id]
    subgraph = G.subgraph(nodes_in_community)
    density = nx.density(subgraph)
    community_density.append(density)
print("Density of communities:", community_density)
```

```
Number of communities detected: 5769
Sizes of communities: [192813, 59802, 14403, 3244, 23895, 132482, 8108, 37204, 97210, 99058, 11893, 12832, 10674, 4376, 9053, 21233, 8064, 1436, 1678, 17147, 3093, 38180, 49610, 1994, 3209,
Modularity of the partition: 0.7217321605946327
```

```
Density of communities: [2.4138908558066548e-05, 7.300167317866367e-05, 0.0002154248076837284l, 0.0007149991654224046, 9.91685517361301e-05, 5.667143110341645 4e-05, 0.0002958700688600768 7,
```

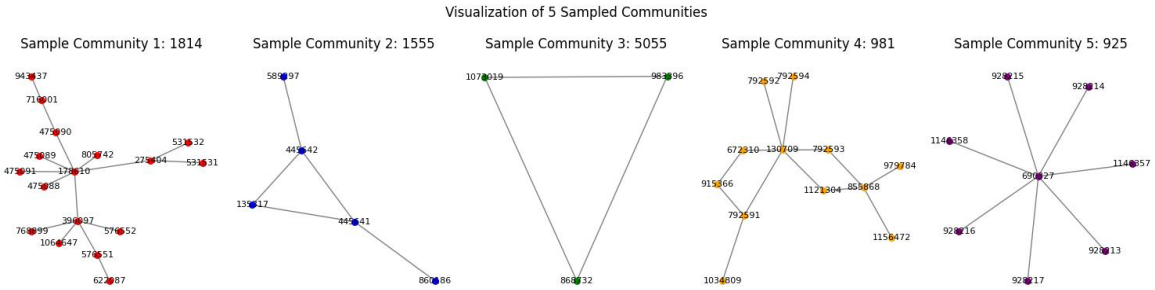




Density Plot of Community Sizes

Density Plot of Community Density

Randomly select five communities for visualization:



References：

[1] 基于 Modularity 的社区发现：https://www.jianshu.com/p/36525bee7aac

[2] 社区检测：https://zhuanlan.zhihu.com/p/41105026, https://www.jianshu.com/p/4ebe42dfa8ec

[3] python-louvain：https://python-louvain.readthedocs.io/_/downloads/en/latest/pdf/

[4] Newman M E J .Modularity and community structure in networks[J].Proceedings of the National Academy of Sciences of the United States of America, 2006, 103(23):8577-8582.DOI:10.1073/pnas.0601602103.