Problem 1

```
import networkx as nx
from networkx.algorithms.community import greedy_modularity_communities
import matplotlib.pyplot as plt
from operator import itemgetter
```

1. Create a Directed graph.

```
# Create a directed graph
graph= nx.read_edgelist("connections.txt", create_using=nx.DiGraph(directed=True),nodetype = int)
```

2. Show which nodes are bridges.***

```
graphnodir = nx.read_edgelist("connections.txt", create_using=nx.Graph(),nodetype = int)
list(nx.bridges(graphnodir))
```

```
      (107, 1119),
      (107, 1145),
      (107, 1206),
      (107, 1386),
      (107, 1466),
      (107, 1560),
      (107, 1581),
      (107, 1834),
      (348, 358),
      (348, 447),
      (348, 550),
      (414, 585),
      (414, 602),
      (414, 607),
      (414, 608),
      (414, 613),
      (414, 624),
      (414, 638),
      (414, 668),
      (414, 674),
      (1684, 2842),
      (1684, 3031),
      (1684, 3071),
      (1684, 3183),
      (1684, 3230),
      (1912, 2079),
      (1912, 2195),
      (1912, 2269),
      (1912, 2457),
      (1912, 2470),
      (1912, 2569),
      (1912, 2596),
      (3437, 3451),
      (3437, 3453),
      (3437, 3570),
      (3437, 3650),
      (3437, 3709),
      (698, 883),
      (698, 891),
      (698, 892)]
```

3. Show the density of the graph. Comment about your findings

```
# Calculate density
graph_density = nx.density(graph)

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

```
Graph Density: 0.0054099817517196435
```

The measure of density is between 0 and 1. The score of density 0.0054 indicates that the graph is very light. This means that the number of the edges that we have is far less than the total number of all possible edges. The nodes are not strongly intercorrelated. There might be groups that are strongly connected to each other however there might be nearly no connection between them.

4. Show which nodes have the highest and lowest number of connections.

```
# Calculate in-degree and out-degree for each node
in_degree = dict(graph.in_degree())
out_degree = dict(graph.out_degree())

# Calculate overall degree (sum of in-degree and out-degree) for each node
overall_degree = {node: in_degree[node] + out_degree[node] for node in graph.nodes()}

# Get nodes with the highest and lowest overall degree
max_overall_degree_node = max(overall_degree, key=overall_degree.get)
min_overall_degree_node = min(overall_degree, key=overall_degree.get)

print("Node with the Highest Overall Degree:", max_overall_degree_node)
print("Node with the Lowest Overall Degree:", min_overall_degree_node)
```

```
Node with the Highest Overall Degree: 107
Node with the Lowest Overall Degree: 11
```

5. Show which nodes have the highest incoming and outgoing connections.

```
# Get nodes with the highest in-degree and out-degree
max_in_degree_node = max(in_degree, key=in_degree.get)
max_out_degree_node = max(out_degree, key=out_degree.get)

print("Node with the Highest In-Degree:", max_in_degree_node)
print("Node with the Highest Out-Degree:", max_out_degree_node)
```

```
Node with the Highest In-Degree: 1888
Node with the Highest Out-Degree: 107
```

6. Show which nodes have the highest closeness, betweenness, and eigenvector | Interpret your findings

```
# Calculate Closeness Centrality
closeness_centrality = nx.closeness_centrality(graph)

# Calculate Betweenness Centrality
betweenness_centrality = nx.betweenness_centrality(graph)

# Calculate Eigenvector Centrality
eigenvector_centrality = nx.eigenvector_centrality(graph,max_iter = 500)

# Identify nodes with the highest centrality scores for each metric
max_closeness_node = max(closeness_centrality, key=closeness_centrality.get)
max_betweenness_node = max(betweenness_centrality, key=betweenness_centrality.get)
max_eigenvector_node = max(eigenvector_centrality, key=eigenvector_centrality.get)

# Print or use these nodes for targeting
print(f"Node with highest closeness centrality: {max_closeness_node}")
print(f"Node with highest betweenness centrality: {max_betweenness_node}")
print(f"Node with highest eigenvector centrality: {max_eigenvector_node}")
```

```
Node with highest closeness centrality: 2642
Node with highest betweenness centrality: 1684
Node with highest eigenvector centrality: 2655
```

- Closeness centrality measures how close a node is to all other nodes in the network. Node 2642 has the highest closeness centrality. This suggests this node is on average closer to all other nodes in the network in terms of the shortest paths.
- Betweenness centrality quantifies the extent to which a node lies on the shortest paths between other nodes in the network. It is calculated as the fraction of shortest paths that pass through the node. Node 1684 having the highest betweenness centrality indicates that it acts as a critical bridge in connecting different parts of the network. Information or influence passing through the network often goes through node 1684.
- Eigenvector centrality measures the influence of a node in a network, considering both the direct and indirect connections. It is based on the concept that connections to high-scoring nodes contribute more to a node's own score. Node 2655 has the highest eigenvector centrality which means that it is well connected to other nodes that are themselves well connected. It is a node with a high level of influence in the network.

7. Implement a community detection algorithm on the directed graph and show how many communities were created.

```
# Apply the greedy modularity algorithm for community detection
communities = list(greedy_modularity_communities(graph))
# count the communities
num_communities = len(communities)
print("Number of communities:",num_communities)
```

```
Number of communities: 11
```

8. Show the largest and the smallest community. | Interpret your findings

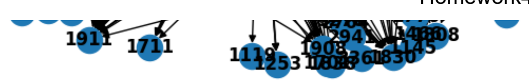
```
# Sort communities by size
sorted_communities = sorted(communities, key=len)

# Print the largest and smallest communities
largest_community = sorted_communities[-1]
smallest_community = sorted_communities[0]
print("Largest Community:", largest_community)
print("Smallest Community:", smallest_community)
print(f'Largest community: {len(largest_community)}')
print(f'Smallest community: {len(smallest_community)}')
```

```
Largest Community: frozenset({107, 353, 363, 366, 389, 428, 475, 483, 484, 517, 526, 538, 566, 580, 596, 601, 606, 629, 2678, 637, 641,
Smallest Community: frozenset({1105, 1890, 1027, 1208, 1002, 1627})
Largest community: 1001
Smallest community: 6
```

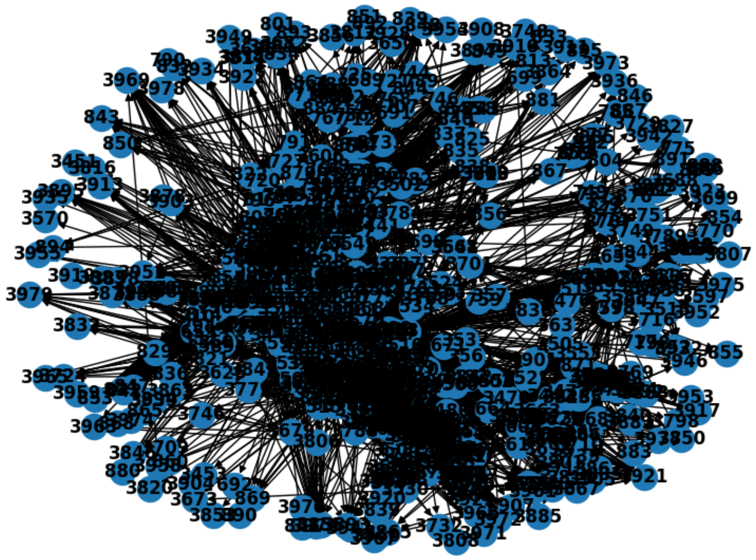
- The largest community is a significant and densely connected subgroup within our network. The nodes within this community share strong connections, suggesting potential common interests, interactions, or relationships.
- The smallest community is a small subgroup within the network. The limited size may imply more isolated or specialized connections among nodes in this community.
- In a marketing context, understanding the characteristics and interests of the largest community can help tailor marketing strategies to appeal to a broader audience.
- The insights from the smallest community may guide targeted marketing efforts to a more specific audience.

9. Select the largest three communities and draw them. After doing so, remove the top 3 nodes with the highest Degree Centrality, Closeness Centrality, Betweenness Centrality, and Eigenvector. Each should be in a separate plot/draw.

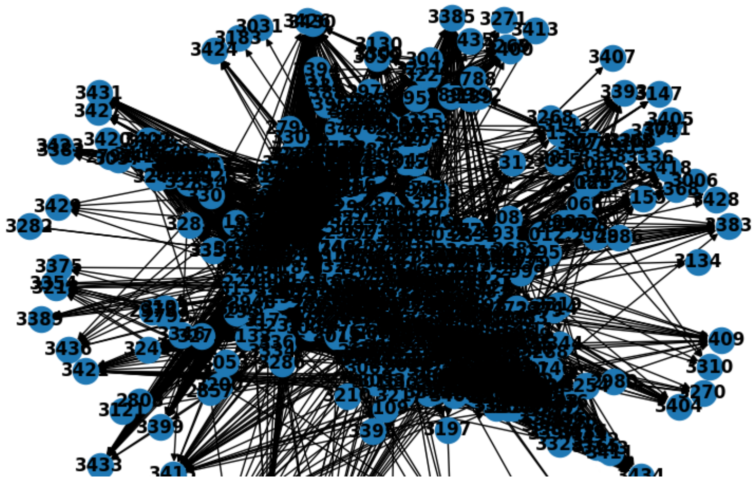
```
# Sort communities by size in descending order
sorted_communities_ = sorted(communities, key=len, reverse=True)
# Select the largest three communities
largest_three_communities = sorted_communities_[:3]


# Draw the largest three communities
for i, community in enumerate(largest_three_communities):
    subgraph = graph.subgraph(community)
    plt.figure(figsize=(8, 6))
    pos = nx.spring_layout(subgraph)
    nx.draw(subgraph, pos, with_labels=True, font_weight='bold', arrowsize=10)
    plt.title(f"Largest Community {i+1}")
    plt.show()
```



Largest Community 2



Largest Community 3



```
# Separating the largest communities
largest_community = sorted_communities[-1]
second_largest = sorted_communities[-2]
third_largest = sorted_communities[-3]


# Create subgraphs to be able to do centrality calculations
subgraph_1 = graph.subgraph(largest_community)
subgraph_2 = graph.subgraph(second_largest)
subgraph_3 = graph.subgraph(third_largest)
```

```
# Centralities for each criteria
degree_centrality_1 = nx.degree_centrality(subgraph_1)
closeness_centrality_1 = nx.closeness_centrality(subgraph_1)
betweenness_centrality_1 = nx.betweenness_centrality(subgraph_1)
eigenvector_centrality_1 = nx.eigenvector_centrality(subgraph_1,max_iter = 10000)


degree_centrality_2 = nx.degree_centrality(subgraph_2)
closeness_centrality_2 = nx.closeness_centrality(subgraph_2)
betweenness_centrality_2 = nx.betweenness_centrality(subgraph_2)
eigenvector_centrality_2 = nx.eigenvector_centrality(subgraph_2,max_iter = 10000)


degree_centrality_3 = nx.degree_centrality(subgraph_3)
closeness_centrality_3 = nx.closeness_centrality(subgraph_3)
betweenness_centrality_3 = nx.betweenness_centrality(subgraph_3)
eigenvector_centrality_3 = nx.eigenvector_centrality(subgraph_3,max_iter = 10000)


# Separating the top 3 nodes of each community of each criteria
top_nodes_degree_1 = sorted(degree_centrality_1, key=degree_centrality_1.get, reverse=True)[:3]
top_nodes_closeness_1 = sorted(closeness_centrality_1, key=closeness_centrality_1.get, reverse=True)[:3]
top_nodes_betweenness_1 = sorted(betweenness_centrality_1, key=betweenness_centrality_1.get, reverse=True)[:3]
top_nodes_eigenvector_1 = sorted(eigenvector_centrality_1, key=eigenvector_centrality_1.get, reverse=True)[:3]


top_nodes_degree_2 = sorted(degree_centrality_2, key=degree_centrality_2.get, reverse=True)[:3]
top_nodes_closeness_2 = sorted(closeness_centrality_2, key=closeness_centrality_2.get, reverse=True)[:3]
top_nodes_betweenness_2 = sorted(betweenness_centrality_2, key=betweenness_centrality_2.get, reverse=True)[:3]
top_nodes_eigenvector_2 = sorted(eigenvector_centrality_2, key=eigenvector_centrality_2.get, reverse=True)[:3]


top_nodes_degree_3 = sorted(degree_centrality_3, key=degree_centrality_3.get, reverse=True)[:3]
top_nodes_closeness_3 = sorted(closeness_centrality_3, key=closeness_centrality_3.get, reverse=True)[:3]
top_nodes_betweenness_3 = sorted(betweenness_centrality_3, key=betweenness_centrality_3.get, reverse=True)[:3]
top_nodes_eigenvector_3 = sorted(eigenvector_centrality_3, key=eigenvector_centrality_3.get, reverse=True)[:3]
```

Removing the top nodes and plotting

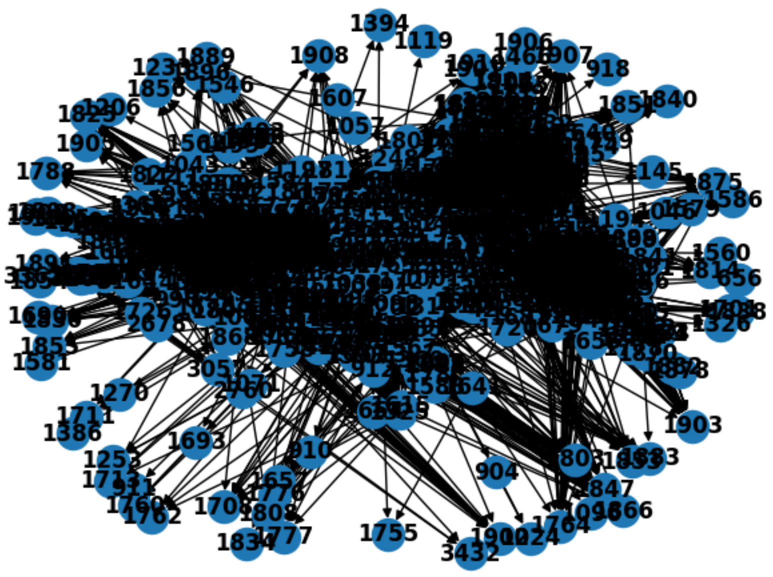
12 Plots in total

```
G_removed = subgraph_1.copy()
G_removed.remove_nodes_from(degree_centrality_1)
plt.figure()
nx.draw(subgraph_1, with_labels=True, font_weight='bold', arrowsize=10)
plt.title("Community 1 with Highest 3 Degrees Removed")
plt.show()
```

Community 1 with Highest 3 Degrees Removed

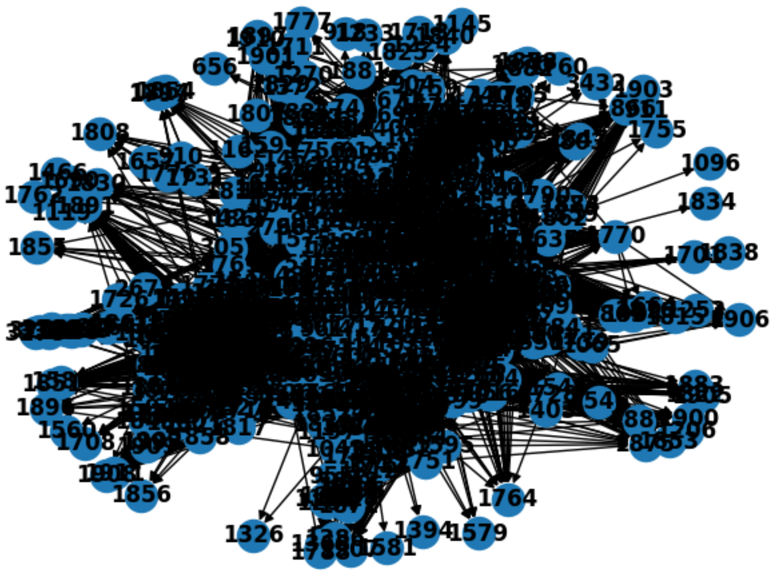
```
G_removed = subgraph_1.copy()
G_removed.remove_nodes_from(closeness_centrality_1)
plt.figure()
nx.draw(subgraph_1, with_labels=True, font_weight='bold', arrowsize=10)
plt.title("Community 1 with highest 3 Closeness Removed")
plt.show()
```

Community 1 with highest 3 Closeness Removed



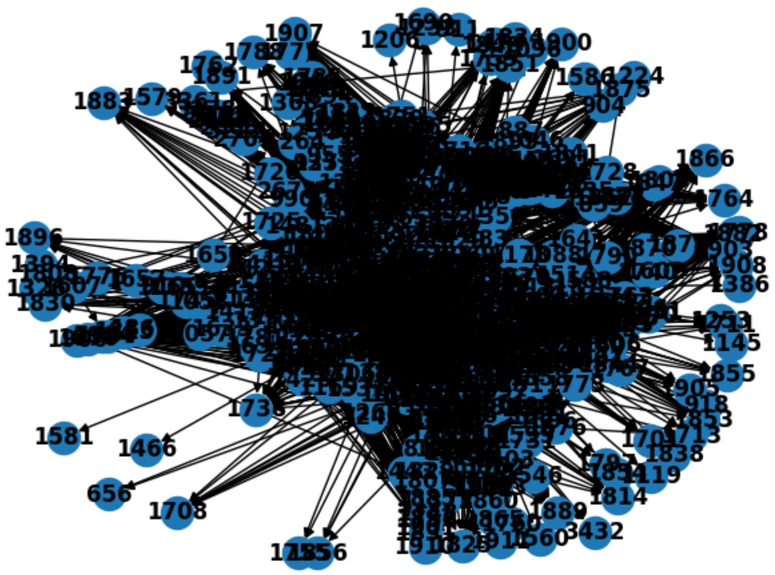
```
G_removed = subgraph_1.copy()
G_removed.remove_nodes_from(betweenness_centrality_1)
plt.figure()
nx.draw(subgraph_1, with_labels=True, font_weight='bold', arrowsize=10)
plt.title("Community 1 with highest 3 Betweenness Removed")
plt.show()
```

Community 1 with highest 3 Betweenness Removed



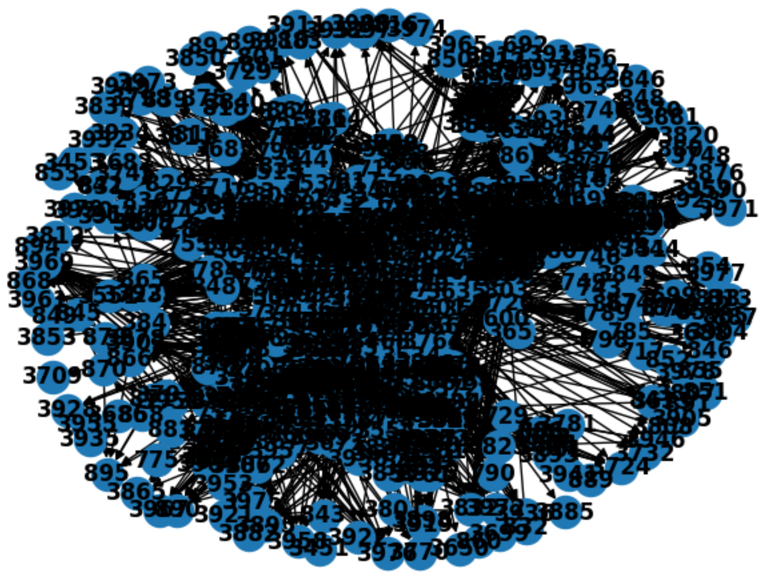
```
G_removed = subgraph_1.copy()
G_removed.remove_nodes_from(eigenvector_centrality_1)
plt.figure()
nx.draw(subgraph_1, with_labels=True, font_weight='bold', arrowsize=10)
plt.title("Community 1 with highest 3 Eigenvector Values Removed")
plt.show()
```

Community 1 with highest 3 Eigenvector Values Removed



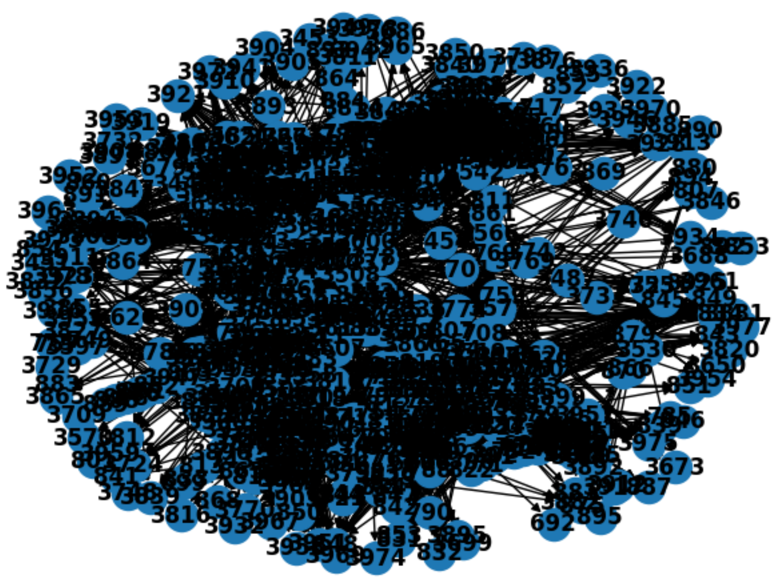
```
G_removed = subgraph_2.copy()
G_removed.remove_nodes_from(degree_centrality_2)
plt.figure()
nx.draw(subgraph_2, with_labels=True, font_weight='bold', arrowsize=10)
plt.title("Community 2 with highest 3 Degrees Removed")
plt.show()
```

Community 2 with highest 3 Degrees Removed



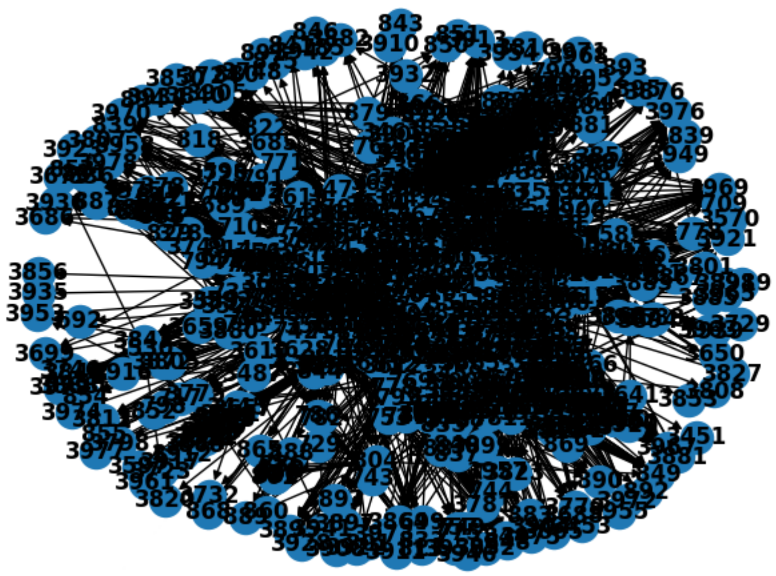
```
G_removed = subgraph_2.copy()
G_removed.remove_nodes_from(closeness_centrality_2)
plt.figure()
nx.draw(subgraph_2, with_labels=True, font_weight='bold', arrowsize=10)
plt.title("Community 2 with highest 3 Closeness Removed")
plt.show()
```

Community 2 with highest 3 Closeness Removed

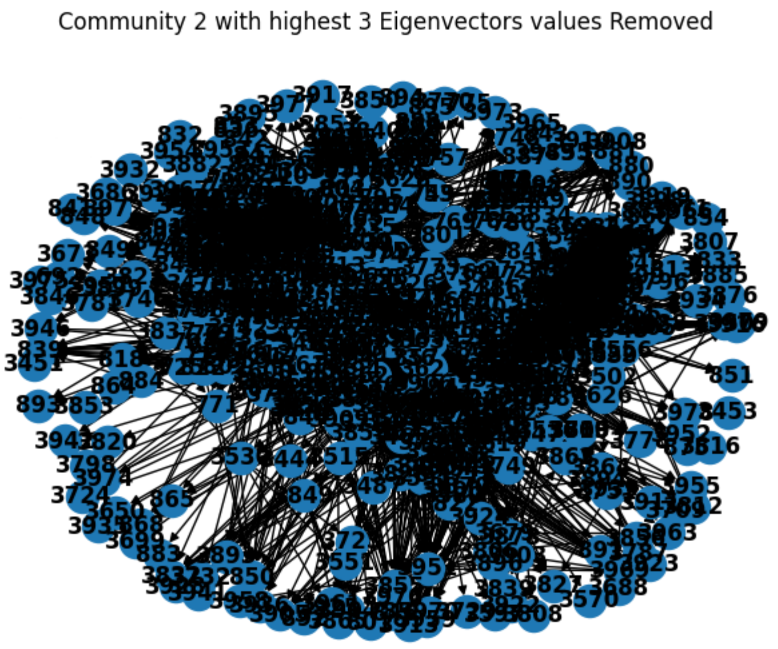


```
G_removed = subgraph_2.copy()
G_removed.remove_nodes_from(betweenness_centrality_2)
plt.figure()
nx.draw(subgraph_2, with_labels=True, font_weight='bold', arrowsize=10)
plt.title("Community 2 with highest 3 Betweenness Removed")
plt.show()
```

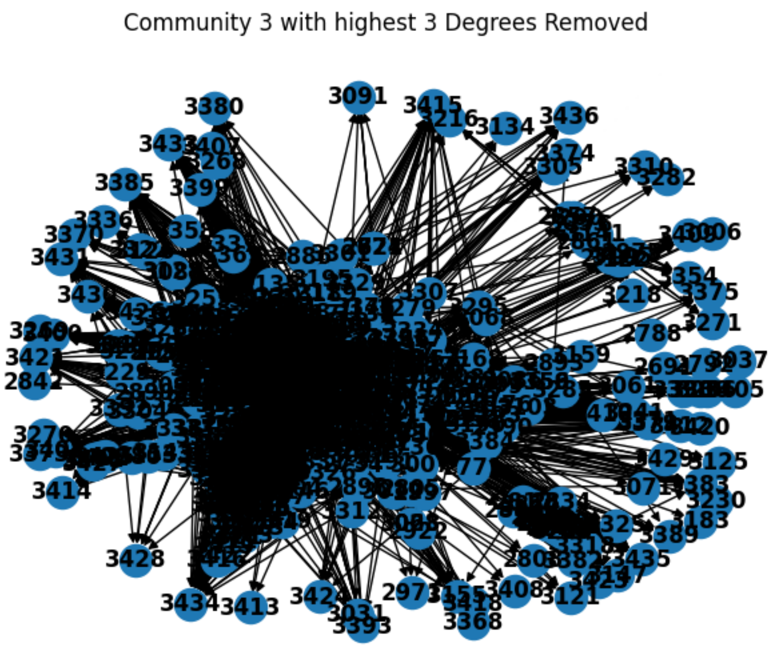
Community 2 with highest 3 Betweenness Removed



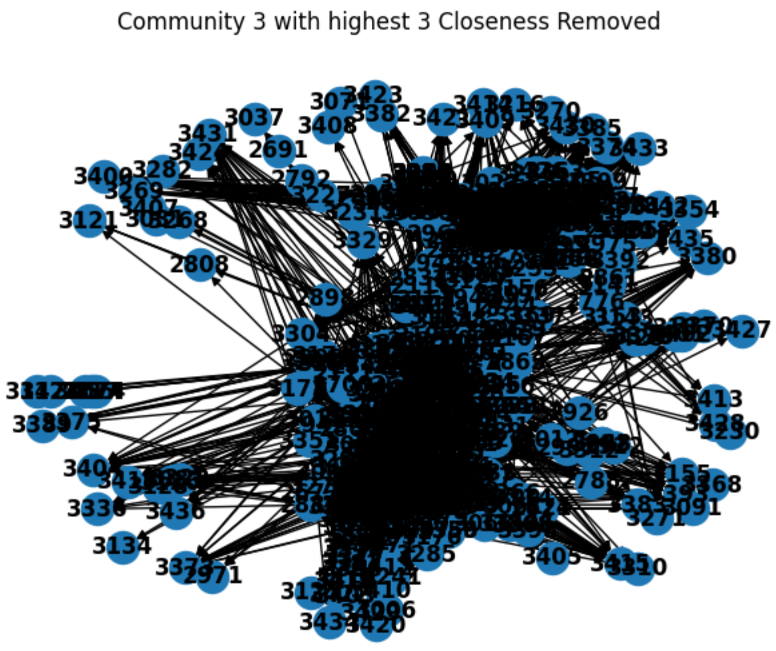
```
G_removed = subgraph_2.copy()
G_removed.remove_nodes_from(eigenvector_centrality_2)
plt.figure()
nx.draw(subgraph_2, with_labels=True, font_weight='bold', arrowsize=10)
plt.title("Community 2 with highest 3 Eigenvectors values Removed")
plt.show()
```



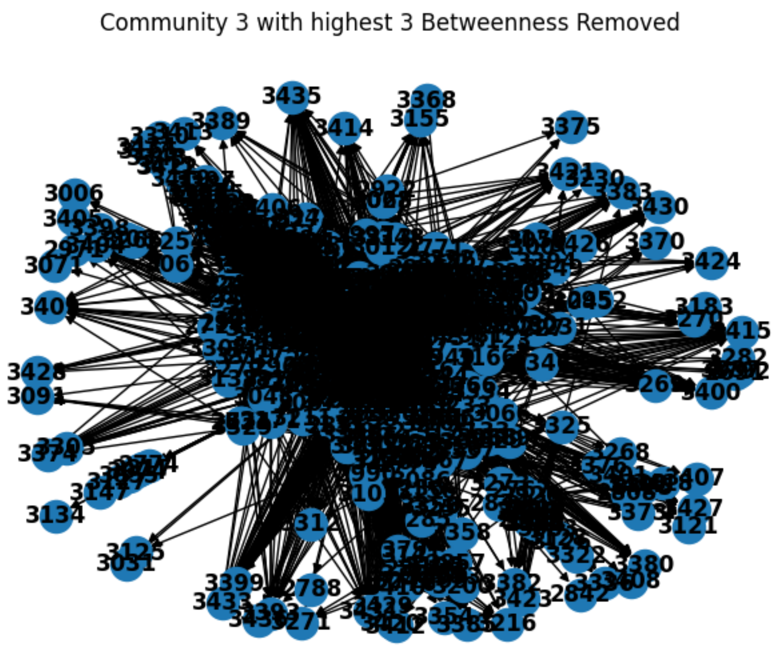
```
G_removed = subgraph_3.copy()
G_removed.remove_nodes_from(degree_centrality_3)
plt.figure()
nx.draw(subgraph_3, with_labels=True, font_weight='bold', arrowsize=10)
plt.title("Community 3 with highest 3 Degrees Removed")
plt.show()
```



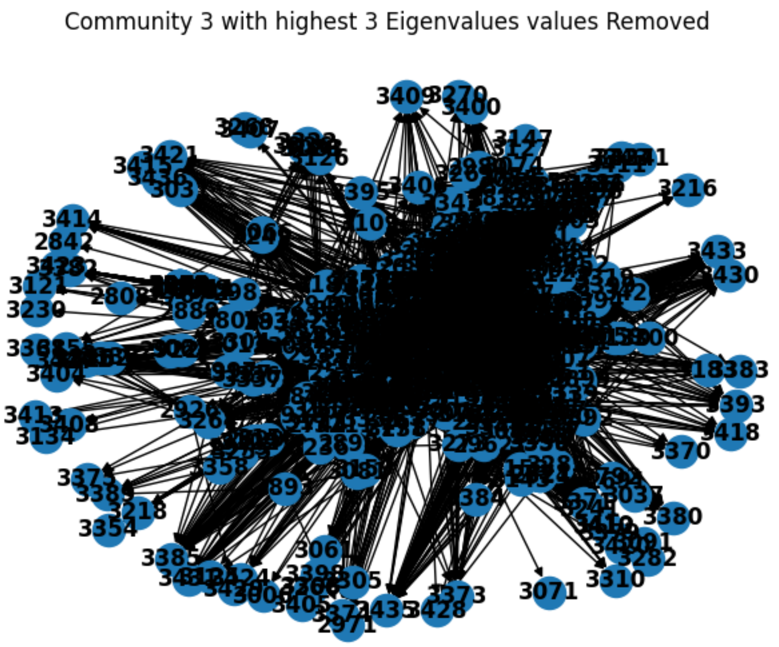
```
G_removed = subgraph_3.copy()
G_removed.remove_nodes_from(closeness_centrality_3)
plt.figure()
nx.draw(subgraph_3, with_labels=True, font_weight='bold', arrowsize=10)
plt.title("Community 3 with highest 3 Closeness Removed")
plt.show()
```



```
G_removed = subgraph_3.copy()
G_removed.remove_nodes_from(betweenness_centrality_3)
plt.figure()
nx.draw(subgraph_3, with_labels=True, font_weight='bold', arrowsize=10)
plt.title("Community 3 with highest 3 Betweenness Removed")
plt.show()
```



```
G_removed = subgraph_3.copy()
G_removed.remove_nodes_from(eigenvector_centrality_3)
plt.figure()
nx.draw(subgraph_3, with_labels=True, font_weight='bold', arrowsize=10)
plt.title("Community 3 with highest 3 Eigenvalues values Removed")
plt.show()
```



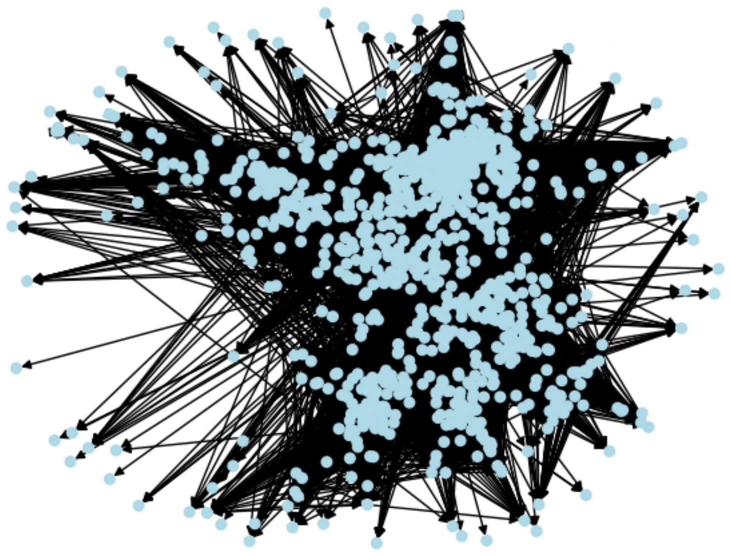
10. Draw the influencers of those top 3 communities.

Identifying the influencers

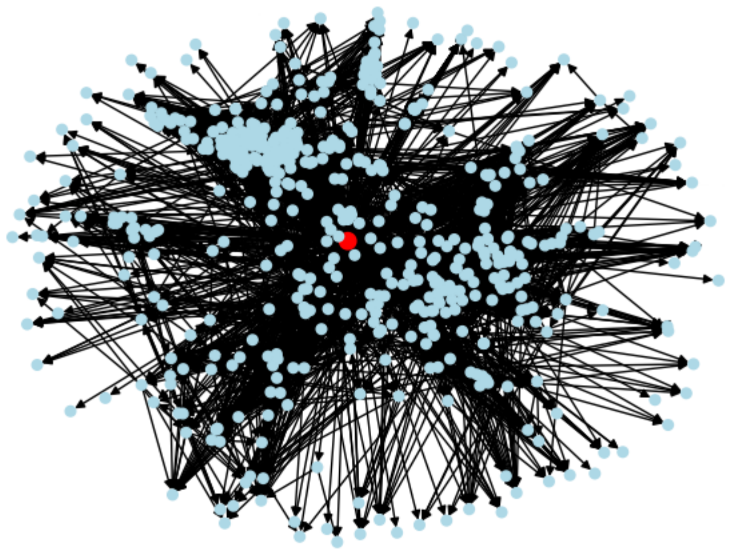
Putting the influencers in to the graphs and plotting

```
influencers = []
for community in largest_three_communities:
    subgraph = graph.subgraph(community)
    centrality = nx.degree_centrality(subgraph)
    influencer = max(centrality, key=centrality.get)
    influencers.append(influencer)


influencer_subgraphs = []
for influencer in influencers:
    influencer_subgraph = graph.subgraph(nx.ego_graph(graph, influencer, radius=1))
    influencer_subgraphs.append(influencer_subgraph)


node_sizes = []
for subgraph in influencer_subgraphs:
    centrality = nx.degree_centrality(subgraph)
    sizes = [80 if node == max(centrality, key=centrality.get) else 30 for node in subgraph.nodes()]
    node_sizes.append(sizes)


for i, subgraph in enumerate(influencer_subgraphs):
    plt.figure(i+1)
    node_colors = ['lightblue' if node != influencers[i] else 'red' for node in subgraph.nodes()]
    nx.draw(subgraph, with_labels=False, node_color=node_colors, node_size=node_sizes[i])
    plt.title(f"Top Community {i+1}")
    plt.show()
```

Top Community 1



Top Community 2



Top Community 3