munity-detection-using-networkx-2

December 18, 2023

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import networkx as nx
```

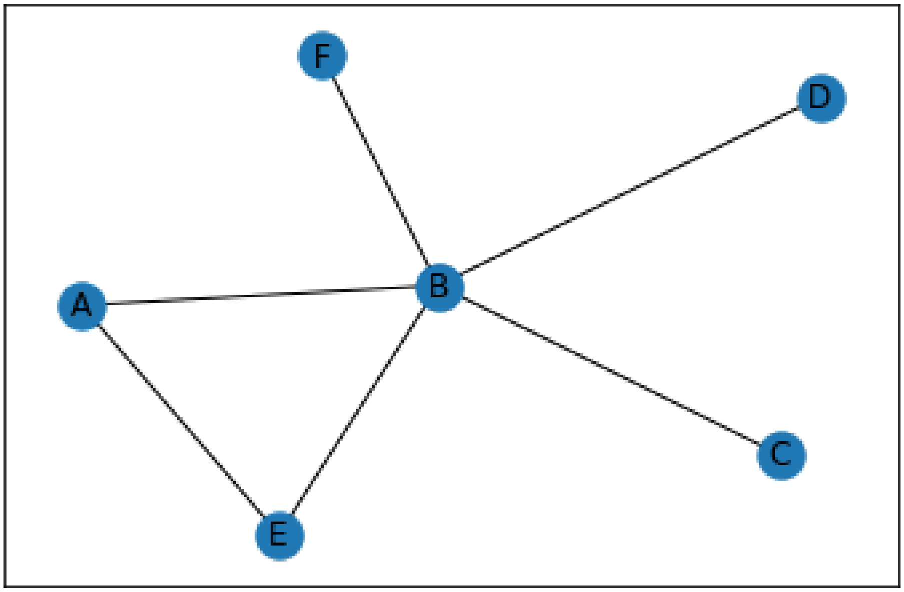
[2]:
```
G = nx.Graph()
G.add_edge('A','B',weight=13,relation='friend')
G.add_edge('B','C',weight=9,relation='family')
G.add_edge('B','D',weight=7,relation='friend')
G.add_edge('E','B',weight=10,relation='friend')
G.add_edge('E','A',weight=1,relation='enemy')
G.add_edge('F','B',weight=13,relation='family')
G.edges(data=True)
```

[2]:
```
EdgeDataView([('A', 'B', {'weight': 13, 'relation': 'friend'}), ('A', 'E',
{'weight': 1, 'relation': 'enemy'}), ('B', 'C', {'weight': 9, 'relation':
'family'}), ('B', 'D', {'weight': 7, 'relation': 'friend'}), ('B', 'E',
{'weight': 10, 'relation': 'friend'}), ('B', 'F', {'weight': 13, 'relation':
'family'})])
```

[3]:
```
G.add_node('A',role='Trader')
G.add_node('B',role='Analyst')
G.add_node('C',role='Manager')
G.nodes(data=True)
```

[3]:
```
NodeDataView({'A': {'role': 'Trader'}, 'B': {'role': 'Analyst'}, 'C': {'role':
'Manager'}, 'D': {}, 'E': {}, 'F': {}})
```

[4]:
```
nx.draw_networkx(G, with_labels=True)
```

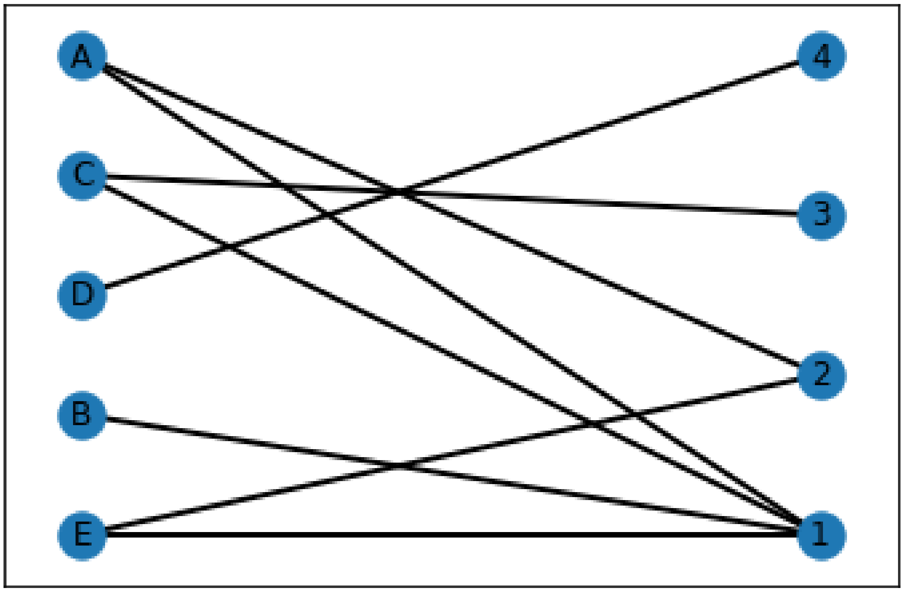


[5]:
```
from networkx.algorithms import bipartite
B = nx.Graph()
B.add_nodes_from(['A','B','C','D','E'],bipartite=0)
B.add_nodes_from([1,2,3,4],bipartite=1)
B.
 ↪add_edges_from([('A',1),('B',1),('C',1),('C',3),('D',4),('E',1),('A',2),('E',2)])
bipartite.is_bipartite(B)
```

[5]: True

[6]:
```
edges = B.edges()
nx.draw_networkx(
    B,
    pos = nx.drawing.layout.bipartite_layout(B, ['A','B','C','D','E']),
    width = 2)
print(edges)
```

```
[('A', 1), ('A', 2), ('B', 1), ('C', 1), ('C', 3), ('D', 4), ('E', 1), ('E', 2)]
```



DOLPHINS NETWORK

This is a directed social network of bottlenose dolphins. The nodes are the bottlenose dolphins (genus Tursiops) of a bottlenose dolphin community living off Doubtful Sound, a fjord in New Zealand (spelled fiord in New Zealand). An edge indicates a frequent association. The dolphins were observed between 1994 and 2001. The network that has been used is Dolphin network [19]. Dolphin network contains 62 vertices and 159 links. After applying MFLM in Dolphin network, we receive 0.518 as modularity score and 5 communities.

Network Graph Generated using Gephi

[7]:
```
import pandas as pd

read_file = pd.read_excel ("dolphin.xlsx")

read_file.to_csv ("dolphin.csv", index = None, header=True)

df = pd.DataFrame(pd.read_csv("dolphin.csv"))

df
```

[7]:

	Node1	Node2
0	0	35
1	0	2
2	0	6
3	0	42
4	0	7
..	…	…

```
154      52      56
155      53      56
156      53      61
157      55      60
158      56      61

[159 rows x 2 columns]
```

[8]:
```python
G = nx.Graph()
for index, row in df.iterrows():
    G.add_edge(row['Node1'], row['Node2'])
```

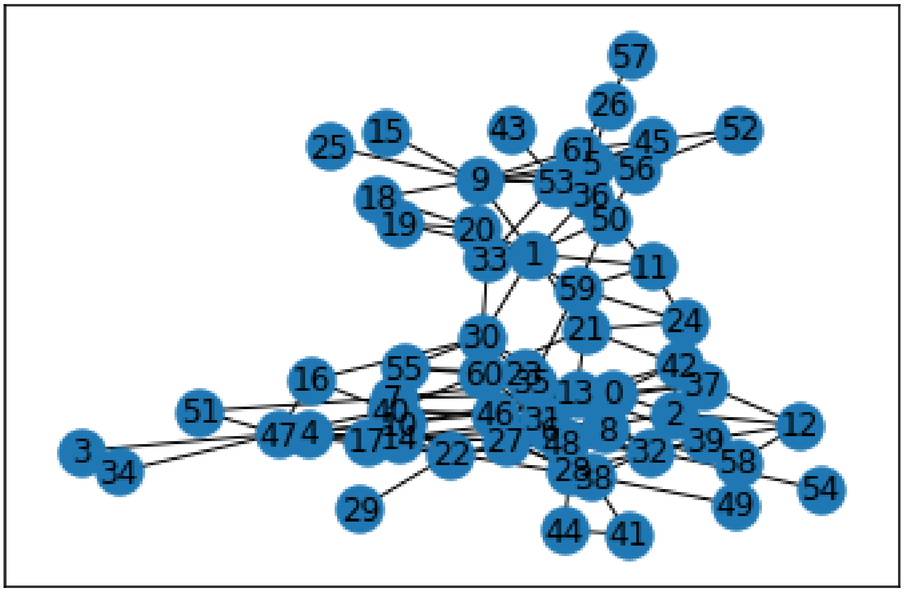
[9]:
```python
print(G.nodes)
```

```
[0, 35, 2, 6, 42, 7, 37, 1, 30, 36, 9, 11, 50, 19, 20, 21, 12, 22, 3, 47, 4, 27,
5, 45, 56, 61, 26, 53, 23, 28, 31, 32, 38, 8, 46, 48, 17, 40, 10, 51, 55, 13,
15, 18, 25, 14, 59, 24, 39, 58, 60, 16, 29, 57, 44, 33, 54, 34, 41, 49, 43, 52]
```

[10]:
```python
print(G.edges)
```

```
[(0, 35), (0, 2), (0, 6), (0, 42), (0, 7), (0, 37), (35, 6), (35, 7), (35, 27),
(35, 30), (35, 31), (35, 59), (35, 48), (2, 12), (2, 37), (2, 22), (2, 42), (6,
23), (6, 27), (6, 28), (6, 31), (6, 32), (6, 38), (6, 8), (6, 46), (6, 48), (6,
17), (42, 13), (42, 21), (42, 24), (42, 37), (7, 40), (7, 10), (7, 51), (7, 17),
(7, 55), (37, 12), (37, 24), (37, 46), (1, 30), (1, 36), (1, 9), (1, 11), (1,
50), (1, 19), (1, 20), (1, 21), (30, 13), (30, 16), (30, 31), (30, 33), (30,
55), (36, 5), (36, 61), (36, 53), (36, 50), (9, 5), (9, 56), (9, 61), (9, 15),
(9, 53), (9, 18), (9, 20), (9, 25), (11, 59), (11, 50), (11, 24), (50, 5), (50,
56), (50, 59), (50, 53), (19, 18), (19, 20), (20, 18), (20, 59), (21, 13), (21,
60), (21, 24), (12, 39), (12, 58), (22, 10), (22, 14), (22, 17), (22, 29), (22,
38), (22, 40), (22, 47), (22, 48), (3, 47), (47, 10), (47, 14), (47, 16), (47,
17), (47, 34), (47, 40), (47, 46), (47, 51), (4, 27), (27, 8), (27, 14), (27,
28), (27, 31), (27, 32), (27, 38), (27, 46), (5, 45), (5, 56), (5, 61), (5, 26),
(5, 53), (45, 52), (45, 61), (45, 53), (56, 52), (56, 53), (56, 61), (61, 26),
(61, 53), (26, 57), (53, 33), (53, 43), (23, 60), (23, 55), (28, 39), (28, 31),
(28, 44), (31, 8), (31, 14), (31, 60), (31, 38), (31, 40), (31, 58), (32, 8),
(32, 13), (32, 38), (32, 39), (32, 48), (32, 54), (38, 41), (38, 49), (8, 46),
(8, 13), (46, 13), (46, 40), (17, 10), (17, 40), (40, 10), (40, 14), (40, 16),
(40, 60), (40, 55), (10, 13), (10, 14), (55, 60), (13, 60), (13, 39), (59, 24),
(58, 49), (44, 41)]
```

[11]:
```python
nx.draw_networkx(G, with_labels=True)
```

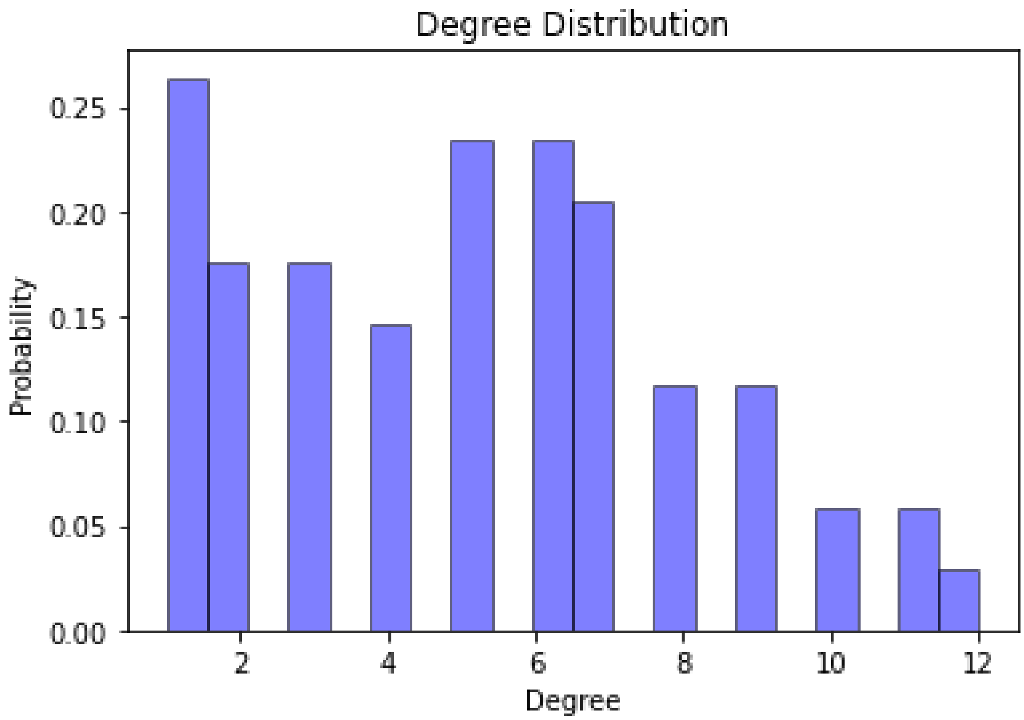


0.1 Network Type Prediction in Dolphin Newtork

[12]:
```
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np
from collections import Counter
from scipy.stats import norm, powerlaw, expon
from scipy.optimize import curve_fit

# Load your dataset as a NetworkX graph (replace this with your specific dataset)
G = nx.read_edgelist("dolphin.txt")

# Compute the degree of each node
degrees = dict(G.degree())
degree_values = list(degrees.values())

# Calculate the degree histogram
hist, bin_edges = np.histogram(degree_values, bins=20, density=True)

# Plot the degree distribution
plt.hist(degree_values, bins=20, density=True, alpha=0.5, color='b', edgecolor='black')
plt.title("Degree Distribution")
plt.xlabel("Degree")
plt.ylabel("Probability")
```

```python
# Define functions for fitting different distribution types
def power_law(x, a, b):
    return a * (x**b)

def exponential(x, scale):
    return scale * np.exp(-scale * x)

# Fit the degree distribution to different candidate distributions
params_powerlaw, _ = curve_fit(power_law, bin_edges[:-1], hist)
params_exponential, _ = curve_fit(exponential, bin_edges[:-1], hist)

# Determine the type of distribution based on the parameters
alpha_powerlaw = params_powerlaw[1]
scale_exponential = 1 / params_exponential[0]

if alpha_powerlaw > 2.0:
    print("The degree distribution appears to be closer to a power-law␣
 ↪(scale-free) distribution.")
elif scale_exponential < 2.0:
    print("The degree distribution appears to be closer to an exponential␣
 ↪distribution.")
else:
    print("The degree distribution does not seem to be a power-law or␣
 ↪exponential distribution.")

plt.show()
```

```
The degree distribution does not seem to be a power-law or exponential
distribution.
```



[13]:
```
def network_describe(G):
    num_nodes = G.number_of_nodes()
    num_edges = G.number_of_edges()
    degree_sequence = [degree for (node, degree) in G.degree()]
    avg_degree = sum(degree_sequence) / num_nodes
    degree_centrality = nx.degree_centrality(G)
    betweenness_centrality = nx.betweenness_centrality(G)
    eigenvector_centrality = nx.eigenvector_centrality(G)
    avg_clustering_coefficient = nx.average_clustering(G)
    degree_centrality = nx.degree_centrality(G)
    metric_values = list(degree_centrality.values())
    variance = np.var(metric_values)
    list_of_betw_tuples = list(betweenness_centrality.items())
    list_of_eigen_tuples = list(eigenvector_centrality.items())
    list_of_degree_tuples = list(degree_centrality.items())
    print("NETWORK STATISTICS :")
    print("Network Info : ",nx.info(G))
    print("Nodes : ",num_nodes)
    print("Edges : ",num_edges)
    print("Degree Centality : ")
    print(list_of_degree_tuples)
    print("Betweenness Centality : ")
    print(list_of_betw_tuples)
```

```python
    print("Eigenvector Centality : ")
    print(list_of_eigen_tuples)
    print("Average Clustering Coefficient : ")
    print(avg_clustering_coefficient)
    print("Variance : ")
    print(variance)
```

[14]:
```python
network_describe(G)
```

```
NETWORK STATISTICS :
Network Info :  Graph with 62 nodes and 159 edges
Nodes :  62
Edges :  159
Degree Centality :
[('0', 0.09836065573770492), ('40', 0.13114754098360656), ('10',
0.0819672131147541), ('14', 0.19672131147540983), ('47', 0.09836065573770492),
('15', 0.11475409836065574), ('42', 0.09836065573770492), ('1',
0.13114754098360656), ('36', 0.11475409836065574), ('41', 0.0819672131147541),
('17', 0.14754098360655737), ('19', 0.06557377049180328), ('54',
0.11475409836065574), ('26', 0.04918032786885246), ('27', 0.0819672131147541),
('28', 0.0819672131147541), ('2', 0.06557377049180328), ('44',
0.06557377049180328), ('61', 0.04918032786885246), ('3', 0.04918032786885246),
('8', 0.09836065573770492), ('59', 0.0819672131147541), ('4',
0.01639344262295082), ('51', 0.1639344262295082), ('5', 0.06557377049180328),
('56', 0.03278688524590164), ('9', 0.11475409836065574), ('13',
0.13114754098360656), ('57', 0.14754098360655737), ('6', 0.09836065573770492),
('7', 0.0819672131147541), ('30', 0.0819672131147541), ('37',
0.18032786885245902), ('45', 0.18032786885245902), ('20', 0.14754098360655737),
('32', 0.04918032786885246), ('29', 0.14754098360655737), ('11',
0.01639344262295082), ('12', 0.01639344262295082), ('33', 0.1639344262295082),
('34', 0.0819672131147541), ('38', 0.13114754098360656), ('43',
0.11475409836065574), ('16', 0.09836065573770492), ('50', 0.11475409836065574),
('52', 0.06557377049180328), ('24', 0.09836065573770492), ('18',
0.11475409836065574), ('55', 0.03278688524590164), ('22', 0.01639344262295082),
('25', 0.04918032786885246), ('31', 0.01639344262295082), ('21',
0.09836065573770492), ('23', 0.04918032786885246), ('35', 0.01639344262295082),
('60', 0.01639344262295082), ('49', 0.03278688524590164), ('39',
0.03278688524590164), ('58', 0.01639344262295082), ('46', 0.03278688524590164),
('53', 0.03278688524590164), ('48', 0.01639344262295082)]
Betweenness Centality :
[('0', 0.01908259621374376), ('40', 0.1431495183426175), ('10',
0.01609202091169304), ('14', 0.06197200484885411), ('47', 0.023201476119508912),
('15', 0.033292220982233216), ('42', 0.02915795194483718), ('1',
0.21332443553281097), ('36', 0.24823719602893804), ('41', 0.02325160067018617),
('17', 0.11430016291546972), ('19', 0.013314394166853186), ('54',
0.09912164676351941), ('26', 0.00436247723132969), ('27', 0.029236860493157973),
('28', 0.06675695466395656), ('2', 0.00907281243346817), ('44',
```

```
0.012037805849281259), ('61', 0.014194982613015399), ('3',
0.0023737965131407756), ('8', 0.022365737598409235), ('59',
0.02033277469038459), ('4', 0.0), ('51', 0.08467725475022556), ('5',
0.004380300179480506), ('56', 0.0001366120218579235), ('9',
0.02089438036159347), ('13', 0.05284632843869151), ('57', 0.08420468343495603),
('6', 0.029372536747686688), ('7', 0.11823861926938345), ('30',
0.033050460771772254), ('37', 0.13856978865859435), ('45',
0.040670440596470195), ('20', 0.10264573972090967), ('32', 0.03278688524590164),
('29', 0.06552928249649562), ('11', 0.0), ('12', 0.0), ('33',
0.0571664401172598), ('34', 0.032694759702956426), ('38', 0.045352238835845396),
('43', 0.06283060442896508), ('16', 0.0033047098620869113), ('50',
0.03341103492742837), ('52', 0.01923434489008259), ('24', 0.007383043489600867),
('18', 0.014854899716954898), ('55', 0.0008774695250105086), ('22', 0.0), ('25',
0.0016441148408361526), ('31', 0.0), ('21', 0.012700653930162124), ('23',
0.04218278563875617), ('35', 0.0), ('60', 0.0), ('49', 0.0009289617486338798),
('39', 0.07051677853993399), ('58', 0.0), ('46', 0.0030084669428931724), ('53',
0.0011930783242258653), ('48', 0.0)]
Eigenvector Centality :
[('0', 0.12850351911087213), ('40', 0.20787263130203706), ('10',
0.07525346435103512), ('14', 0.3157810764804676), ('47', 0.08037195763313358),
('15', 0.16417491138339663), ('42', 0.08095068341031325), ('1',
0.042091441435724804), ('36', 0.13276550630017522), ('41',
0.015262103099078309), ('17', 0.01753496833868234), ('19',
0.020682545955984934), ('54', 0.023022376520473394), ('26',
0.008949192287688038), ('27', 0.016326687320731025), ('28',
0.06822697202643722), ('2', 0.039757133098010516), ('44', 0.07780212964010047),
('61', 0.05199117201156754), ('3', 0.07933447607445519), ('8',
0.1431022167512096), ('59', 0.1118186569975593), ('4', 0.029287057142628192),
('51', 0.21068020892089806), ('5', 0.006572752164517532), ('56',
0.0026123141901086854), ('9', 0.012220169764468426), ('13',
0.015030415487480686), ('57', 0.017401973681970617), ('6',
0.012211980286547347), ('7', 0.04290802005630266), ('30', 0.04075065098782636),
('37', 0.30056092847049554), ('45', 0.28500310473240426), ('20',
0.18447787335323973), ('32', 0.003864343758966286), ('29', 0.21176109464474216),
('11', 0.029287057142628192), ('12', 0.03907586116163528), ('33',
0.2810970171605869), ('34', 0.1388272511035687), ('38', 0.19661653892836548),
('43', 0.19033796034550665), ('16', 0.20799316926734232), ('50',
0.21769051331081496), ('52', 0.1295635910153738), ('24', 0.19321180983300903),
('18', 0.20249300124247616), ('55', 0.05210934393071285), ('22',
0.0024382375398253426), ('25', 0.005952326426510383), ('31',
0.0024382375398253426), ('21', 0.20734961722376982), ('23',
0.08736202260403907), ('35', 0.029437314009677567), ('60',
0.0005374384753918982), ('49', 0.02342956700211971), ('39',
0.02087597074555255), ('58', 0.02733206208697855), ('46', 0.029716244849600233),
('53', 0.03368665459890722), ('48', 0.0024198473930088903)]
Average Clustering Coefficient :
0.2589582460550202
Variance :
```

```
0.0023101999199637803
```

Modularity Score

Modularity is a measure of the structure of networks or graphs which measures the strength of division of a network into modules (also called groups, clusters or communities). Networks with high modularity have dense connections between the nodes within modules but sparse connections between nodes in different modules. Modularity is often used in optimization methods for detecting community structure in networks. Biological networks, including animal brains, exhibit a high degree of modularity.

[15]:
```
def modularity_score(Gr, i, c):
    E=1
    m=Gr.number_of_edges()
    ki = Gr.degree(i)
    kc = Gr.degree(c)
    p = (ki*kc)/((2*m)**2)
    q = E/(2*m)
    return q-p
```

[16]:
```
def calculate_community_mod(G1,n,a):
   x=[]
   for i in a:
       x.append(list(i))

   comm_mod = 0
   for k in range(0,n):
       H=G1.subgraph(x[0])
       for i in x[0]: #
           #neighbours
           i_neighbours = list(H.adj[i])
           for j in i_neighbours:
                comm_mod = comm_mod + modularity_score(H, i, j)

   return comm_mod
```

[17]:
```
import networkx as nx

def girvan_newman(G, num_communities):
    communities = [list(G.nodes())]

    while len(communities) < num_communities:
        edge_betweenness = nx.edge_betweenness_centrality(G)

        max_edge_betweenness = max(edge_betweenness.values())
        edges_to_remove = [edge for edge, centrality in edge_betweenness.
↪items() if centrality == max_edge_betweenness]
        G.remove_edges_from(edges_to_remove)
```

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

        if len(new_communities) > len(communities):
            communities = new_communities

    return communities
```

[18]:
```
import matplotlib.pyplot as plt
def draw_communities(G, communities):
    pos = nx.spring_layout(G)
    color_map = ['blue', 'green', 'red', 'orange', 'purple', 'pink', 'yellow',␣
↪'crimson', 'saddlebrown', 'magenta']

    for i, community in enumerate(communities):
        nx.draw_networkx_nodes(G, pos, nodelist=list(community),␣
↪node_color=color_map[i], node_size=500)

    nx.draw_networkx_edges(G, pos, alpha=0.5)
    plt.show()
```

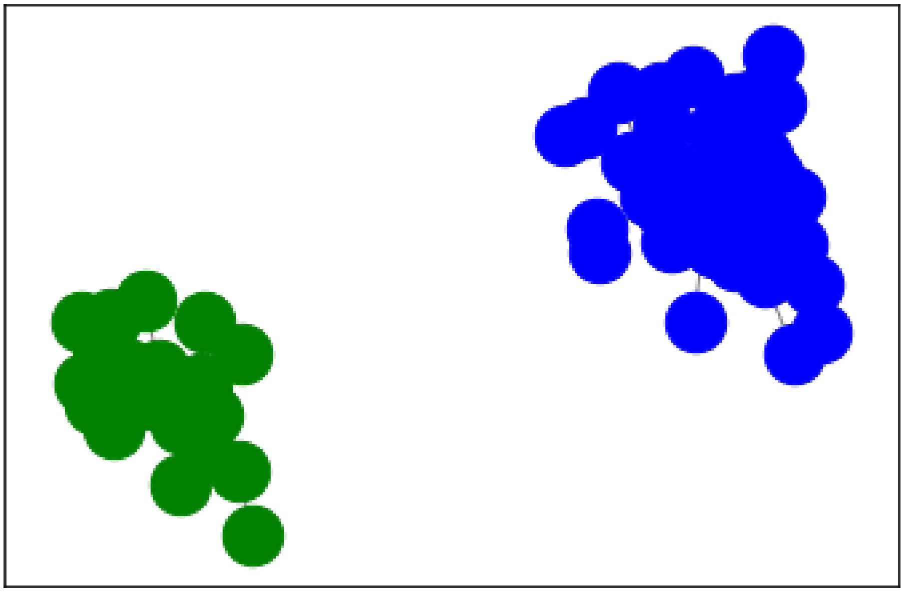
COMMUNITY DETECTION IN FOOTBALL NETWORK

Network Graph Generated using Gephi

[19]:
```
a = girvan_newman(G, 2)
print(a)
```

```
[{'37', '11', '2', '34', '24', '8', '47', '52', '45', '0', '59', '18', '49',
'12', '16', '21', '50', '38', '55', '14', '30', '43', '23', '29', '44', '42',
'28', '46', '36', '3', '40', '15', '51', '58', '33', '35', '20', '10', '61',
'53', '4'}, {'54', '56', '26', '9', '48', '25', '22', '1', '57', '17', '7',
'39', '41', '6', '5', '32', '60', '27', '13', '19', '31'}]
```

[20]:
```
draw_communities(G,a)
```



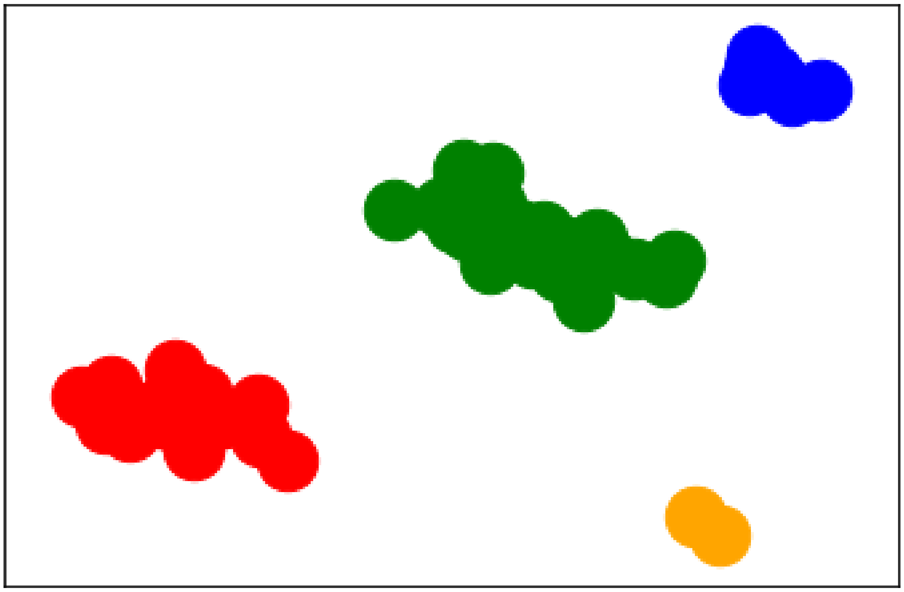
[21]:
```
mod_G_2 = calculate_community_mod(G,2,a)
print(mod_G_2)
```

```
1.5522262435821903
```

[22]:
```
a = girvan_newman(G, 4)
print(a)
```

```
[{'42', '28', '2', '10', '47', '30', '0'}, {'37', '11', '34', '24', '8', '52',
'45', '59', '18', '49', '12', '16', '21', '38', '50', '55', '14', '43', '23',
'29', '44', '46', '36', '3', '40', '15', '51', '58', '33', '35', '20', '4'},
{'54', '56', '26', '9', '48', '25', '22', '1', '57', '17', '7', '39', '41', '6',
'5', '32', '60', '27', '13', '19', '31'}, {'61', '53'}]
```

[23]:
```
draw_communities(G,a)
```



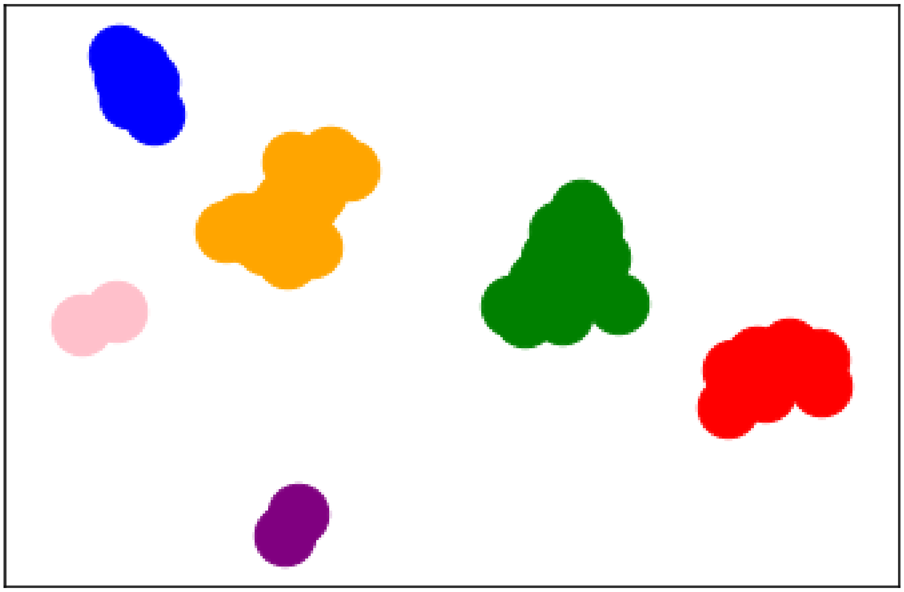
[24]:
```python
mod_G_4 = calculate_community_mod(G,4,a)
print(mod_G_4)
```

```
1.6249999999999993
```

[25]:
```python
a = girvan_newman(G, 6)
print(a)
```

```
[{'42', '28', '2', '10', '47', '30', '0'}, {'37', '34', '8', '52', '59', '49',
'12', '16', '38', '50', '14', '43', '44', '46', '36', '3', '40', '58', '33',
'20'}, {'11', '18', '35', '24', '21', '55', '4', '23', '15', '29', '45', '51'},
{'54', '56', '26', '9', '48', '25', '22', '1', '57', '17', '7', '39', '41', '6',
'5', '27', '13', '19', '31'}, {'61', '53'}, {'32', '60'}]
```

[26]:
```python
draw_communities(G,a)
```



[27]:
```
mod_G_6 = calculate_community_mod(G,6,a)
print(mod_G_6)
```

```
2.4374999999999987
```

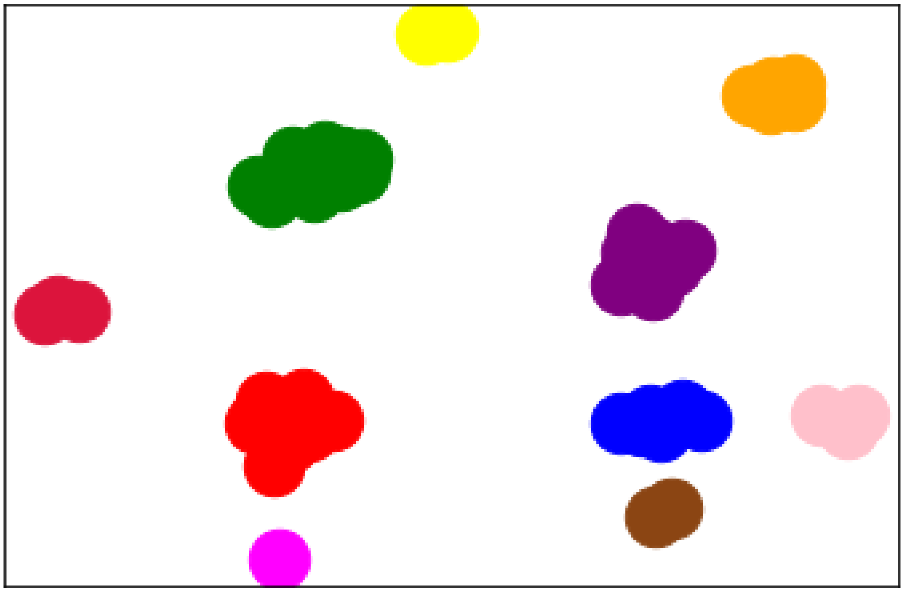
[28]:
```
a = girvan_newman(G, 10)
print(a)
```

```
[{'42', '28', '2', '10', '47', '30', '0'}, {'37', '33', '49', '46', '20', '34',
'16', '36', '43', '44', '38', '50', '14', '40', '52'}, {'11', '18', '35', '24',
'21', '55', '4', '23', '15', '29', '45', '51'}, {'25', '7', '26', '1', '19',
'27'}, {'54', '57', '39', '13', '41', '6', '56', '5', '9', '48'}, {'31', '17',
'22'}, {'61', '53'}, {'8', '59', '3'}, {'32', '60'}, {'12'}, {'58'}]
```

[29]:
```
draw_communities(G,a)
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_26224/1699081581.py in <module>
----> 1 draw_communities(G,a)

~\AppData\Local\Temp/ipykernel_26224/2270384348.py in draw_communities(G,␣
↪communities)
      5
      6     for i, community in enumerate(communities):
----> 7         nx.draw_networkx_nodes(G, pos, nodelist=list(community),␣
↪node_color=color_map[i], node_size=500)
```

```
    nx.draw_networkx_edges(G, pos, alpha=0.5)

IndexError: list index out of range
```



[30]:
```python
mod_G_10 = calculate_community_mod(G,10,a)
print(mod_G_10)
```

```
4.062499999999998
```

[31]:
```python
from networkx.algorithms import community
from operator import itemgetter
communities = community.greedy_modularity_communities(G)
```

[32]:
```python
betweenness_dict = nx.betweenness_centrality(G)
eigenvector_dict = nx.eigenvector_centrality(G)

nx.set_node_attributes(G, betweenness_dict, 'betweenness')
nx.set_node_attributes(G, eigenvector_dict, 'eigenvector')
```

[33]:
```python
modularity_dict = {}
for i,c in enumerate(communities):
    for name in c:
        modularity_dict[name]=i

nx.set_node_attributes(G, modularity_dict, 'modularity')
class0 = [n for n in G.nodes() if G.nodes[n]['modularity'] == 0]
```

```
class0_eigenvector = {n:G.nodes[n]['eigenvector'] for n in class0}

class0_sorted_by_eigenvector = sorted(class0_eigenvector.items(),
 ↪key=itemgetter(1), reverse=True)

for i,c in enumerate(communities):
    if len(c) > 2:
        print('Class '+str(i)+':', list(c))
        print('\n')
```

```
Class 0: ['37', '49', '33', '46', '20', '34', '16', '36', '40', '44', '50',
'38', '14', '43', '52']


Class 1: ['11', '18', '35', '24', '21', '55', '4', '23', '15', '29', '45', '51']


Class 2: ['54', '57', '39', '13', '41', '56', '6', '5', '9', '48']


Class 3: ['42', '10', '28', '47', '2', '0', '30']


Class 4: ['25', '7', '19', '1', '26', '27']


Class 5: ['59', '8', '3']


Class 6: ['31', '17', '22']
```

FOOTBALL NETWORK

The file football.gml contains the network of American football games between Division IA colleges during regular season Fall 2000, as compiled by M. Girvan and M. Newman. The nodes have values that indicate to which conferences they belong.

Network Graph Generated using Gephi

[34]:
```
import pandas as pd

read_file = pd.read_excel ("football.xlsx")

read_file.to_csv ("football.csv", index = None, header=True)

df = pd.DataFrame(pd.read_csv("football.csv"))
```

```
df
```

[34]:

```
     Node1  Node2
0        0      1
1        0     44
2        0     49
3        0     77
4        0     42
..     …      …
608    103    114
609    104    108
610    106    108
611    106    107
612    112    113

[613 rows x 2 columns]
```

[35]:

```
G = nx.Graph()
for index, row in df.iterrows():
    G.add_edge(row['Node1'], row['Node2'])
```

[36]:

```
print(G.nodes)
```

```
[0, 1, 44, 49, 77, 42, 7, 104, 23, 51, 31, 105, 108, 46, 6, 8, 55, 12, 103, 68,
33, 35, 4, 2, 60, 113, 10, 5, 85, 87, 13, 95, 98, 72, 3, 27, 71, 20, 22, 41, 48,
57, 76, 96, 114, 26, 37, 39, 62, 66, 69, 92, 109, 38, 50, 63, 83, 79, 101, 18,
74, 94, 9, 67, 107, 88, 102, 11, 82, 29, 30, 80, 90, 91, 15, 75, 112, 61, 32,
36, 81, 14, 56, 59, 64, 86, 97, 93, 16, 54, 78, 106, 111, 17, 24, 28, 70, 89,
110, 19, 43, 45, 47, 53, 21, 25, 99, 34, 84, 65, 52, 73, 40, 58, 100]
```

[37]:

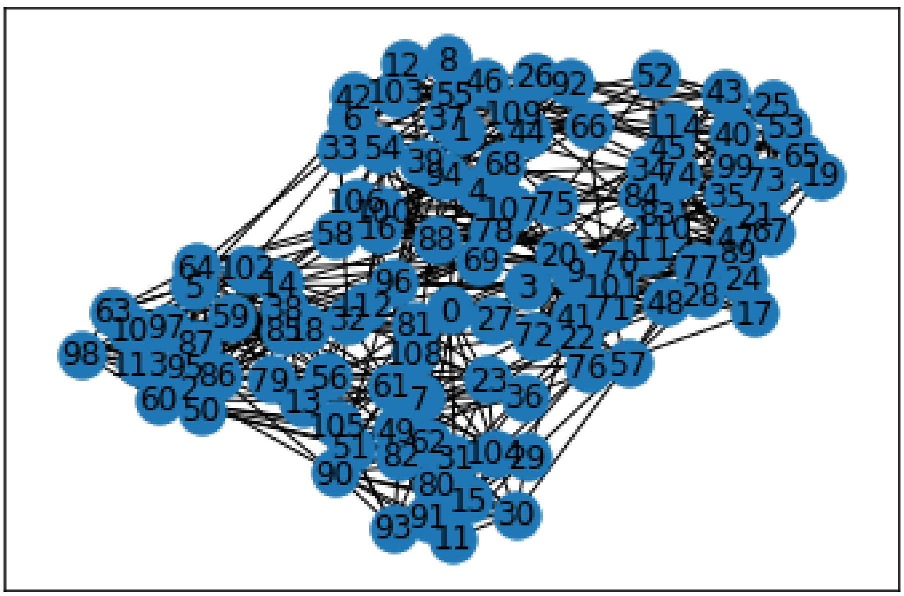
```
print(G.edges)
```

```
[(0, 1), (0, 44), (0, 49), (0, 77), (0, 42), (0, 7), (0, 104), (0, 23), (0, 51),
(0, 31), (0, 105), (0, 108), (1, 42), (1, 46), (1, 6), (1, 8), (1, 55), (1, 12),
(1, 103), (1, 68), (1, 33), (1, 35), (1, 4), (44, 4), (44, 26), (44, 37), (44,
39), (44, 43), (44, 54), (44, 92), (44, 109), (44, 66), (44, 107), (49, 7), (49,
11), (49, 23), (49, 31), (49, 36), (49, 60), (49, 104), (49, 51), (49, 108),
(49, 81), (77, 17), (77, 24), (77, 28), (77, 35), (77, 67), (77, 75), (77, 111),
(77, 78), (77, 83), (77, 101), (42, 6), (42, 8), (42, 12), (42, 26), (42, 33),
(42, 46), (42, 55), (42, 103), (7, 104), (7, 13), (7, 23), (7, 31), (7, 51), (7,
79), (7, 85), (7, 101), (7, 108), (7, 18), (104, 11), (104, 23), (104, 30),
(104, 31), (104, 51), (104, 76), (104, 93), (104, 108), (23, 79), (23, 47), (23,
95), (23, 24), (23, 51), (23, 31), (23, 108), (51, 31), (51, 50), (51, 93), (51,
79), (51, 108), (31, 15), (31, 30), (31, 91), (31, 105), (31, 108), (105, 13),
(105, 32), (105, 36), (105, 60), (105, 61), (105, 81), (105, 93), (108, 94),
(108, 96), (108, 106), (46, 6), (46, 8), (46, 12), (46, 33), (46, 45), (46, 55),
(46, 94), (46, 103), (46, 110), (6, 5), (6, 33), (6, 55), (6, 83), (6, 103), (6,
```

8), (6, 12), (8, 33), (8, 55), (8, 74), (8, 94), (8, 103), (8, 12), (55, 12), (55, 20), (55, 33), (55, 54), (55, 103), (55, 74), (12, 33), (12, 39), (12, 75), (12, 92), (12, 103), (103, 33), (103, 66), (103, 102), (103, 114), (68, 16), (68, 52), (68, 54), (68, 58), (68, 67), (68, 88), (68, 100), (68, 106), (68, 107), (33, 9), (33, 32), (33, 64), (35, 24), (35, 34), (35, 111), (35, 83), (35, 89), (35, 67), (35, 110), (35, 74), (35, 75), (4, 26), (4, 37), (4, 39), (4, 62), (4, 66), (4, 69), (4, 92), (4, 109), (2, 60), (2, 113), (2, 10), (2, 5), (2, 85), (2, 87), (2, 13), (2, 95), (2, 98), (2, 72), (60, 10), (60, 13), (60, 38), (60, 112), (60, 113), (60, 95), (60, 63), (60, 98), (60, 87), (113, 5), (113, 10), (113, 50), (113, 58), (113, 63), (113, 90), (113, 95), (113, 98), (113, 112), (10, 5), (10, 50), (10, 85), (10, 88), (10, 95), (10, 98), (10, 102), (5, 3), (5, 38), (5, 50), (5, 63), (5, 85), (5, 87), (85, 27), (85, 38), (85, 50), (85, 63), (85, 84), (85, 95), (85, 87), (87, 27), (87, 38), (87, 50), (87, 63), (87, 86), (87, 96), (87, 98), (13, 38), (13, 112), (13, 61), (13, 32), (13, 36), (13, 81), (95, 38), (95, 50), (95, 96), (95, 97), (95, 98), (98, 32), (98, 38), (98, 59), (98, 63), (98, 97), (72, 9), (72, 20), (72, 22), (72, 27), (72, 48), (72, 57), (72, 70), (72, 71), (72, 76), (72, 84), (3, 27), (3, 71), (3, 20), (3, 22), (3, 41), (3, 48), (3, 57), (3, 76), (3, 96), (3, 114), (27, 9), (27, 20), (27, 21), (27, 22), (27, 76), (27, 38), (27, 71), (27, 57), (71, 9), (71, 41), (71, 48), (71, 57), (71, 66), (71, 69), (71, 76), (71, 82), (71, 99), (20, 9), (20, 14), (20, 19), (20, 41), (20, 76), (20, 48), (20, 22), (22, 9), (22, 18), (22, 21), (22, 41), (22, 80), (22, 48), (22, 107), (22, 57), (41, 9), (41, 29), (41, 40), (41, 76), (41, 48), (41, 59), (41, 57), (48, 9), (48, 47), (48, 57), (48, 96), (48, 65), (57, 30), (57, 56), (57, 76), (57, 73), (76, 9), (76, 15), (76, 75), (96, 37), (96, 39), (96, 82), (96, 90), (96, 94), (96, 109), (114, 21), (114, 25), (114, 40), (114, 43), (114, 65), (114, 73), (114, 84), (114, 99), (26, 25), (26, 45), (26, 54), (26, 92), (26, 109), (26, 66), (26, 37), (26, 39), (37, 52), (37, 92), (37, 94), (37, 109), (37, 66), (37, 106), (37, 39), (39, 54), (39, 92), (39, 61), (39, 66), (39, 109), (62, 11), (62, 15), (62, 29), (62, 30), (62, 50), (62, 61), (62, 93), (62, 80), (62, 90), (62, 91), (66, 40), (66, 65), (66, 92), (66, 109), (69, 18), (69, 24), (69, 38), (69, 45), (69, 112), (69, 102), (69, 70), (69, 75), (92, 40), (92, 53), (92, 94), (92, 109), (109, 43), (109, 94), (38, 34), (38, 50), (38, 63), (50, 82), (50, 63), (63, 16), (63, 64), (83, 17), (83, 28), (83, 53), (83, 67), (83, 74), (83, 81), (83, 89), (83, 110), (79, 14), (79, 18), (79, 56), (79, 59), (79, 64), (79, 80), (79, 97), (79, 86), (79, 102), (101, 17), (101, 24), (101, 28), (101, 32), (101, 67), (101, 74), (101, 100), (101, 111), (101, 110), (18, 14), (18, 56), (18, 59), (18, 64), (18, 86), (18, 97), (18, 102), (74, 24), (74, 28), (74, 67), (74, 73), (74, 89), (74, 110), (94, 14), (94, 99), (94, 100), (94, 106), (9, 67), (9, 107), (67, 111), (67, 89), (67, 110), (107, 16), (107, 58), (107, 73), (107, 78), (107, 88), (107, 100), (107, 106), (88, 16), (88, 28), (88, 54), (88, 58), (88, 78), (88, 89), (88, 100), (102, 14), (102, 56), (102, 59), (102, 64), (102, 81), (102, 86), (102, 97), (11, 82), (11, 29), (11, 30), (11, 80), (11, 90), (11, 91), (11, 15), (82, 15), (82, 29), (82, 30), (82, 80), (82, 93), (82, 86), (82, 90), (82, 91), (29, 15), (29, 28), (29, 93), (29, 80), (29, 90), (29, 56), (29, 30), (30, 93), (30, 80), (30, 90), (30, 91), (80, 15), (80, 61), (80, 93), (80, 91), (90, 15), (90, 86), (90, 93), (90, 91), (91, 15), (91, 36), (91, 61), (91, 93), (15, 93), (15, 56), (75, 16), (75, 52), (75, 70), (75, 112),

```
(112, 16), (112, 70), (112, 100), (61, 32), (61, 36), (61, 81), (32, 14), (32,
78), (32, 81), (32, 36), (36, 17), (36, 81), (36, 47), (81, 97), (81, 106), (81,
110), (14, 56), (14, 59), (14, 64), (14, 86), (14, 97), (56, 86), (56, 59), (56,
97), (56, 64), (59, 58), (59, 86), (59, 97), (59, 64), (64, 58), (64, 86), (64,
97), (64, 100), (86, 97), (16, 54), (16, 78), (16, 106), (16, 111), (54, 78),
(54, 58), (54, 100), (54, 106), (78, 58), (78, 70), (78, 106), (78, 89), (78,
100), (106, 58), (106, 100), (111, 17), (111, 24), (111, 28), (111, 58), (111,
89), (17, 24), (17, 28), (17, 70), (17, 89), (17, 110), (24, 19), (24, 28), (24,
110), (28, 45), (28, 89), (70, 45), (70, 89), (89, 110), (19, 43), (19, 45),
(19, 47), (19, 53), (19, 21), (19, 25), (19, 99), (19, 34), (43, 25), (43, 34),
(43, 40), (43, 84), (43, 52), (43, 65), (43, 73), (45, 25), (45, 53), (47, 21),
(47, 25), (47, 34), (47, 84), (47, 53), (47, 99), (47, 65), (53, 21), (53, 34),
(53, 40), (53, 52), (53, 99), (53, 73), (21, 84), (21, 99), (21, 65), (21, 34),
(25, 84), (25, 52), (25, 65), (25, 73), (25, 40), (99, 34), (99, 40), (34, 52),
(34, 73), (84, 40), (84, 65), (84, 73), (65, 40), (65, 73), (73, 40), (58, 100)]
```

[38]:
```
nx.draw_networkx(G, with_labels=True)
```

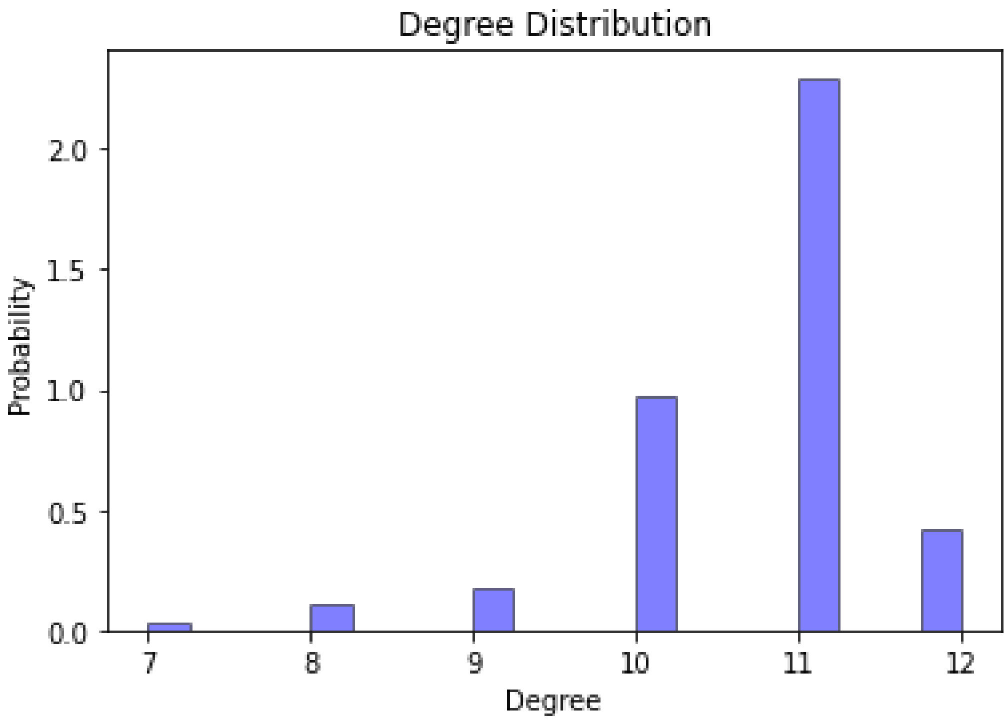


0.2 Network Type Prediction in Football Newtork

[39]:
```
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np
from collections import Counter
from scipy.stats import norm, powerlaw, expon
from scipy.optimize import curve_fit
```

```python
# Load your dataset as a NetworkX graph (replace this with your specific␣
↪dataset)
G = nx.read_edgelist("football.txt")

# Compute the degree of each node
degrees = dict(G.degree())
degree_values = list(degrees.values())

# Calculate the degree histogram
hist, bin_edges = np.histogram(degree_values, bins=20, density=True)

# Plot the degree distribution
plt.hist(degree_values, bins=20, density=True, alpha=0.5, color='b',␣
↪edgecolor='black')
plt.title("Degree Distribution")
plt.xlabel("Degree")
plt.ylabel("Probability")

# Define functions for fitting different distribution types
def power_law(x, a, b):
    return a * (x**b)

def exponential(x, scale):
    return scale * np.exp(-scale * x)

# Fit the degree distribution to different candidate distributions
params_powerlaw, _ = curve_fit(power_law, bin_edges[:-1], hist)
params_exponential, _ = curve_fit(exponential, bin_edges[:-1], hist)

# Determine the type of distribution based on the parameters
alpha_powerlaw = params_powerlaw[1]
scale_exponential = 1 / params_exponential[0]

if alpha_powerlaw > 2.0:
    print("The degree distribution appears to be closer to a power-law␣
↪(scale-free) distribution.")
elif scale_exponential < 2.0:
    print("The degree distribution appears to be closer to an exponential␣
↪distribution.")
else:
    print("The degree distribution does not seem to be a power-law or␣
↪exponential distribution.")

plt.show()
```

```
The degree distribution appears to be closer to a power-law (scale-free)
distribution.
```



[40]:
```
network_describe(G)
```

```
NETWORK STATISTICS :
Network Info :  Graph with 115 nodes and 613 edges
Nodes :  115
Edges :  613
Degree Centality :
[('0', 0.10526315789473684), ('1', 0.10526315789473684), ('35',
0.09649122807017543), ('4', 0.09649122807017543), ('65', 0.09649122807017543),
('33', 0.08771929824561403), ('104', 0.10526315789473684), ('9',
0.09649122807017543), ('16', 0.09649122807017543), ('41', 0.08771929824561403),
('23', 0.09649122807017543), ('90', 0.07894736842105263), ('93',
0.08771929824561403), ('37', 0.09649122807017543), ('103', 0.08771929824561403),
('105', 0.08771929824561403), ('45', 0.09649122807017543), ('109',
0.09649122807017543), ('89', 0.09649122807017543), ('57', 0.08771929824561403),
('25', 0.09649122807017543), ('27', 0.09649122807017543), ('101',
0.08771929824561403), ('2', 0.10526315789473684), ('64', 0.09649122807017543),
('3', 0.10526315789473684), ('100', 0.09649122807017543), ('6',
0.10526315789473684), ('72', 0.09649122807017543), ('74', 0.09649122807017543),
('13', 0.09649122807017543), ('14', 0.08771929824561403), ('47',
0.09649122807017543), ('15', 0.10526315789473684), ('60', 0.09649122807017543),
('106', 0.09649122807017543), ('5', 0.10526315789473684), ('102',
0.08771929824561403), ('81', 0.09649122807017543), ('40', 0.09649122807017543),
('11', 0.08771929824561403), ('52', 0.08771929824561403), ('84',
```

```
0.09649122807017543), ('26', 0.08771929824561403), ('58', 0.08771929824561403),
('108', 0.08771929824561403), ('28', 0.07894736842105263), ('69',
0.09649122807017543), ('97', 0.07017543859649122), ('98', 0.09649122807017543),
('10', 0.08771929824561403), ('107', 0.08771929824561403), ('32',
0.09649122807017543), ('39', 0.09649122807017543), ('7', 0.10526315789473684),
('85', 0.07894736842105263), ('55', 0.09649122807017543), ('68',
0.09649122807017543), ('8', 0.09649122807017543), ('73', 0.09649122807017543),
('77', 0.09649122807017543), ('78', 0.09649122807017543), ('111',
0.09649122807017543), ('82', 0.09649122807017543), ('21', 0.09649122807017543),
('22', 0.09649122807017543), ('51', 0.09649122807017543), ('50',
0.07894736842105263), ('24', 0.08771929824561403), ('12', 0.08771929824561403),
('34', 0.09649122807017543), ('36', 0.07017543859649122), ('38',
0.09649122807017543), ('43', 0.09649122807017543), ('17', 0.09649122807017543),
('18', 0.09649122807017543), ('110', 0.09649122807017543), ('99',
0.08771929824561403), ('71', 0.08771929824561403), ('54', 0.08771929824561403),
('92', 0.09649122807017543), ('114', 0.09649122807017543), ('67',
0.10526315789473684), ('96', 0.08771929824561403), ('113', 0.08771929824561403),
('20', 0.09649122807017543), ('87', 0.09649122807017543), ('62',
0.09649122807017543), ('95', 0.08771929824561403), ('42', 0.06140350877192982),
('19', 0.09649122807017543), ('61', 0.09649122807017543), ('31',
0.09649122807017543), ('44', 0.09649122807017543), ('79', 0.09649122807017543),
('94', 0.08771929824561403), ('29', 0.09649122807017543), ('30',
0.09649122807017543), ('70', 0.09649122807017543), ('75', 0.08771929824561403),
('76', 0.09649122807017543), ('46', 0.09649122807017543), ('66',
0.09649122807017543), ('53', 0.10526315789473684), ('56', 0.08771929824561403),
('63', 0.07894736842105263), ('80', 0.09649122807017543), ('91',
0.09649122807017543), ('49', 0.09649122807017543), ('59', 0.07017543859649122),
('112', 0.08771929824561403), ('48', 0.09649122807017543), ('86',
0.09649122807017543), ('83', 0.09649122807017543), ('88', 0.10526315789473684)]
Betweenness Centality :
[('0', 0.03248994918389481), ('1', 0.017621112680095283), ('35',
0.015475551922823372), ('4', 0.01066386944940908), ('65', 0.010095341327721365),
('33', 0.007354298482007652), ('104', 0.018819414499005812), ('9',
0.01183115085985047), ('16', 0.022213447427560137), ('41',
0.006438559060854568), ('23', 0.0075893356814986605), ('90',
0.008619870213633486), ('93', 0.013952929320212781), ('37',
0.009130695528918065), ('103', 0.013536271209453075), ('105',
0.004371887628309955), ('45', 0.011443177305595334), ('109',
0.00895708821340555), ('89', 0.017157286536913383), ('57',
0.010379976482942584), ('25', 0.01778327242869339), ('27',
0.010660261265228408), ('101', 0.01828733781038952), ('2', 0.01312249705343107),
('64', 0.020365117971128537), ('3', 0.023070098856845616), ('100',
0.011781046708853002), ('6', 0.01968128436895882), ('72', 0.016295178668676143),
('74', 0.010169665340926906), ('13', 0.017291363429954677), ('14',
0.007074447857815539), ('47', 0.015102330786005163), ('15',
0.020895265847707078), ('60', 0.0152026490823968l), ('106',
0.02049815993287542), ('5', 0.010660982082613546), ('102',
0.015927353848291886), ('81', 0.011509807147482301), ('40',
```

```
0.01398268199291763), ('11', 0.011400348520391542), ('52',
0.009851352698010492), ('84', 0.00743251292651829), ('26',
0.017003369986679725), ('58', 0.02882282342077093), ('108',
0.0030021221558908604), ('28', 0.01817205827359323), ('69',
0.02413941977644928), ('97', 0.0103852848761511), ('98', 0.0141402954218983),
('10', 0.006907991485590175), ('107', 0.010360655316940269), ('32',
0.014876854560096126), ('39', 0.010703577458606346), ('7',
0.014563035979091633), ('85', 0.008033939555272976), ('55',
0.011734000737586283), ('68', 0.010756799902972795), ('8',
0.005008648931669307), ('73', 0.008117245702971869), ('77',
0.011900534557201971), ('78', 0.005522846002138627), ('111',
0.0075158855048337025), ('82', 0.033532956726629475), ('21',
0.022126807176414874), ('22', 0.006732525804006759), ('51',
0.01490455333589998), ('50', 0.014372027035004977), ('24', 0.02070022463158683),
('12', 0.010144133783987274), ('34', 0.008025924976628357), ('36',
0.013206024770572687), ('38', 0.025186801970868828), ('43',
0.013595444913298569), ('17', 0.016601996603909323), ('18',
0.008492946116024448), ('110', 0.014370414434605228), ('99',
0.009062853759648142), ('71', 0.01256671016070767), ('54',
0.003905984219124952), ('92', 0.02383635493910066), ('114',
0.010246338156874259), ('67', 0.011810287619392327), ('96',
0.006940379384725653), ('113', 0.0064981105526952), ('20',
0.023046321730608006), ('87', 0.014199915092060613), ('62',
0.014197578069783773), ('95', 0.006927429565443744), ('42',
0.007789504694591859), ('19', 0.009759312735771707), ('61',
0.020755436736059032), ('31', 0.008417562670077419), ('44',
0.014372451528538928), ('79', 0.008171049457356317), ('94',
0.00531688593399675), ('29', 0.009028716823683688), ('30',
0.015086641056354318), ('70', 0.015189872298630964), ('75',
0.01168051872820568), ('76', 0.005946668030335614), ('46',
0.010352652686330774), ('66', 0.010700267807535182), ('53',
0.016560890657173696), ('56', 0.009539783889116069), ('63',
0.01828620821279898), ('80', 0.029161053264223722), ('91',
0.016539482936742626), ('49', 0.00978477343348033), ('59',
0.010865586431809465), ('112', 0.011581581298031529), ('48',
0.017789451176510235), ('86', 0.012377687742137097), ('83',
0.006636641787814403), ('88', 0.02007948941202635)]
Eigenvector Centality :
[('0', 0.1065032293978512), ('1', 0.09638542911769848), ('35',
0.08845985842435759), ('4', 0.10119022701509163), ('65', 0.08141805949050634),
('33', 0.08336777335484148), ('104', 0.11370347792666846), ('9',
0.10653182761728126), ('16', 0.1037909894144992), ('41', 0.09913706039114369),
('23', 0.104481677338296), ('90', 0.07559667897188732), ('93',
0.09410449094935798), ('37', 0.08603421757032985), ('103', 0.08174545556712047),
('105', 0.08192080144470823), ('45', 0.09054271208658636), ('109',
0.08727001085955553), ('89', 0.0914716566711243), ('57', 0.07561031288087337),
('25', 0.09337865264768187), ('27', 0.07951282904057869), ('101',
0.0817152211954476), ('2', 0.11626189945612073), ('64', 0.10645041383760033),
```

```
('3', 0.10624954486445581), ('100', 0.1065014852926181), ('6',
0.1128557163575337), ('72', 0.10099290025784835), ('74', 0.10436714109314042),
('13', 0.10599493440730602), ('14', 0.07663083900658144), ('47',
0.11005250319786304), ('15', 0.11630287637595103), ('60', 0.10194484906088372),
('106', 0.1050279388059104), ('5', 0.10227246540146043), ('102',
0.09032380860564543), ('81', 0.10332061037276577), ('40', 0.1029648816875081),
('11', 0.07986799659559274), ('52', 0.09286230069136649), ('84',
0.10214329652983056), ('26', 0.07091526556639006), ('58', 0.0791455859125681),
('108', 0.10098968033185535), ('28', 0.07011049909509273), ('69',
0.08927304875736317), ('97', 0.058903268579883455), ('98', 0.09621104212189124),
('10', 0.0902578822418965), ('107', 0.09159546710160624), ('32',
0.10920929047838855), ('39', 0.10438478125077645), ('7', 0.12072375014454019),
('85', 0.06817342660723884), ('55', 0.0887723780481328), ('68',
0.11018711699727457), ('8', 0.10781145990002308), ('73', 0.11990229488965841),
('77', 0.11006654691110677), ('78', 0.10430487148702827), ('111',
0.11119849038833507), ('82', 0.10091534627322078), ('21', 0.10996876923522055),
('22', 0.11032339044451624), ('51', 0.10424838995592449), ('50',
0.07434226829121743), ('24', 0.08031930443204505), ('12', 0.07111456568033121),
('34', 0.07681728596169585), ('36', 0.056144254828027394), ('38',
0.08048860896825337), ('43', 0.07457057269224159), ('17', 0.0817893195580222),
('18', 0.07512079745044924), ('110', 0.1149430157363661), ('99',
0.07539511848651986), ('71', 0.076861731791182), ('54', 0.07603169311031659),
('92', 0.08992131428999357), ('114', 0.11769509918671897), ('67',
0.12768299669980845), ('96', 0.07389062373636697), ('113', 0.07045154411385118),
('20', 0.08121510679767424), ('87', 0.08381956695503574), ('62',
0.08252612692980255), ('95', 0.07483965754574211), ('42', 0.04831159242912886),
('19', 0.08348493962139777), ('61', 0.08262817511578947), ('31',
0.08255409289615033), ('44', 0.08599035904154792), ('79', 0.08638687604467117),
('94', 0.08072798109900771), ('29', 0.08576436291400143), ('30',
0.08755028811547799), ('70', 0.0805530683771957), ('75', 0.07768370051308644),
('76', 0.077289492968593), ('46', 0.11956811875565132), ('66',
0.08241407173251931), ('53', 0.1225766153957418), ('56', 0.07555260824037373),
('63', 0.06201949118988243), ('80', 0.09041480488903647), ('91',
0.087836818016329), ('49', 0.117245996201321), ('59', 0.05457208623294783),
('112', 0.07301466471520812), ('48', 0.09250838953437406), ('86',
0.08790696917638745), ('83', 0.11647136158522092), ('88', 0.12128875621075214)]
Average Clustering Coefficient :
0.40321601104209814
Variance :
6.0067922600584965e-05
```

COMMUNITY DETECTION IN FOOTBALL NETWORK

Network Graph Generated using Gephi

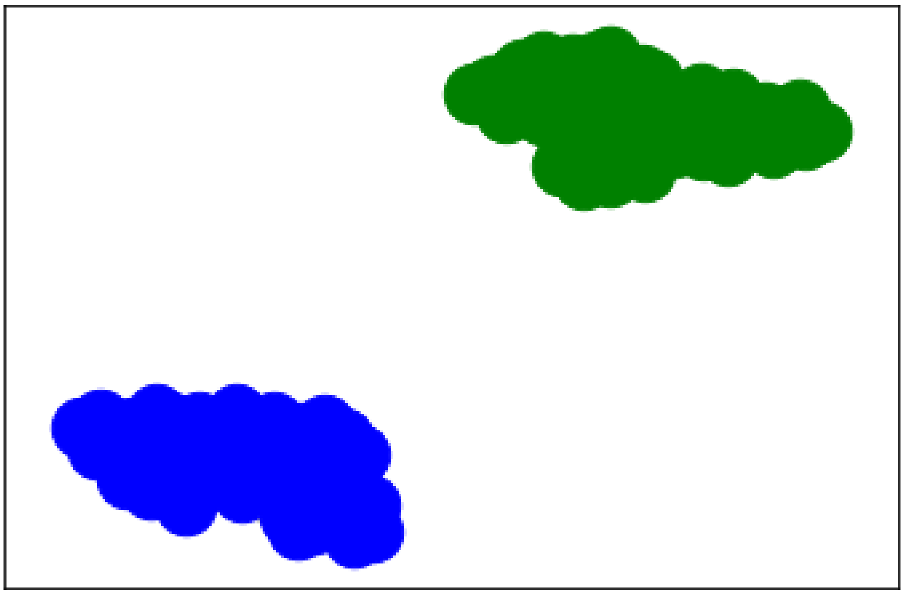
[41]:
```
a = girvan_newman(G, 2)
print(a)
```

```
[{'98', '11', '2', '24', '100', '8', '83', '84', '67', '74', '47', '72', '9',
```

```
'73', '0', '52', '114', '102', '22', '49', '16', '82', '21', '50', '77', '23',
'111', '28', '104', '110', '90', '7', '41', '69', '46', '5', '78', '88', '68',
'39', '3', '6', '64', '32', '60', '40', '15', '51', '107', '81', '13', '108',
'10', '106', '4', '53', '93'}, {'37', '54', '103', '56', '34', '89', '26', '91',
'113', '63', '86', '105', '65', '45', '48', '99', '25', '85', '87', '59', '18',
'66', '12', '1', '38', '79', '62', '55', '30', '109', '14', '43', '44', '29',
'71', '42', '75', '57', '17', '94', '92', '36', '76', '70', '112', '80', '27',
'58', '97', '96', '33', '35', '20', '61', '19', '101', '31', '95'}]
```

[42]:

```
draw_communities(G,a)
```



[43]:

```
mod_G_2 = calculate_community_mod(G,2,a)
print(mod_G_2)
```

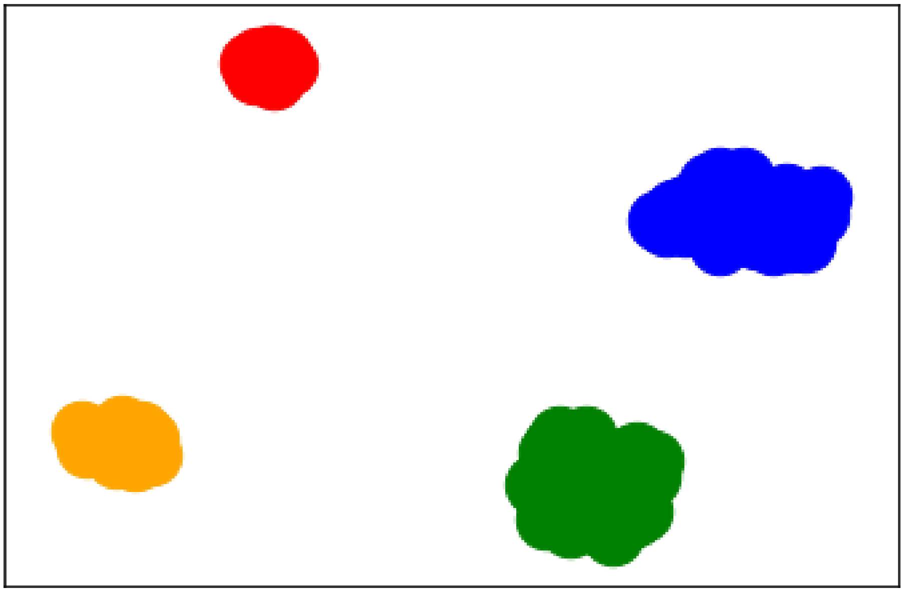
```
1.6448035815951454
```

[44]:

```
a = girvan_newman(G, 4)
print(a)
```

```
[{'98', '11', '24', '8', '83', '84', '67', '74', '72', '9', '73', '0', '52',
'114', '102', '22', '49', '16', '82', '21', '50', '77', '23', '111', '28',
'104', '110', '90', '7', '41', '69', '46', '5', '78', '88', '68', '3', '40',
'51', '107', '81', '108', '10', '4', '53', '93'}, {'37', '103', '56', '89',
'91', '113', '63', '86', '105', '65', '45', '48', '87', '25', '59', '66', '1',
'79', '62', '55', '30', '109', '44', '29', '75', '57', '17', '94', '92', '76',
'70', '112', '80', '27', '58', '97', '96', '33', '35', '20', '19', '101', '95'},
{'39', '13', '6', '2', '100', '32', '64', '47', '60', '106', '15'}, {'42', '54',
'18', '12', '34', '26', '36', '38', '61', '14', '43', '31', '85', '71', '99'}]
```

[45]:
```
draw_communities(G,a)
```



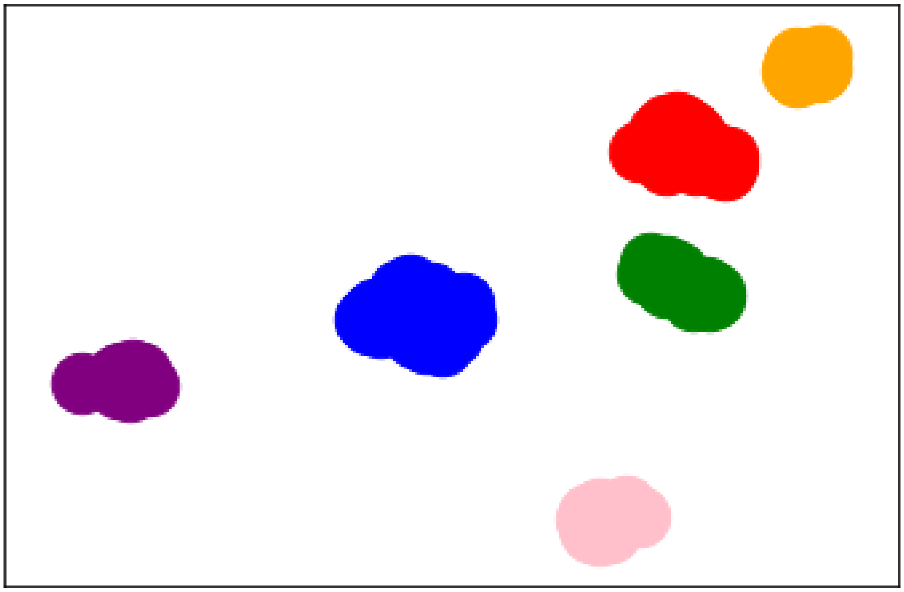
[46]:
```
mod_G_4 = calculate_community_mod(G,4,a)
print(mod_G_4)
```

```
3.1349213029206444
```

[47]:
```
a = girvan_newman(G, 6)
print(a)
```

```
[{'11', '24', '8', '83', '67', '9', '73', '0', '114', '22', '49', '16', '21',
'50', '77', '23', '111', '28', '104', '110', '90', '7', '41', '69', '46', '88',
'78', '68', '51', '108', '4', '53', '93'}, {'37', '25', '33', '80', '103', '35',
'94', '89', '1', '29', '79', '19', '101', '55', '109', '105', '45', '30'},
{'56', '113', '91', '63', '86', '65', '48', '87', '59', '66', '62', '44', '75',
'57', '17', '92', '76', '70', '112', '27', '58', '97', '96', '20', '95'}, {'39',
'13', '6', '2', '100', '32', '64', '47', '60', '106', '15'}, {'98', '107', '81',
'102', '5', '3', '82', '84', '74', '10', '72', '40', '52'}, {'42', '54', '18',
'12', '34', '26', '36', '38', '61', '14', '43', '31', '85', '71', '99'}]
```

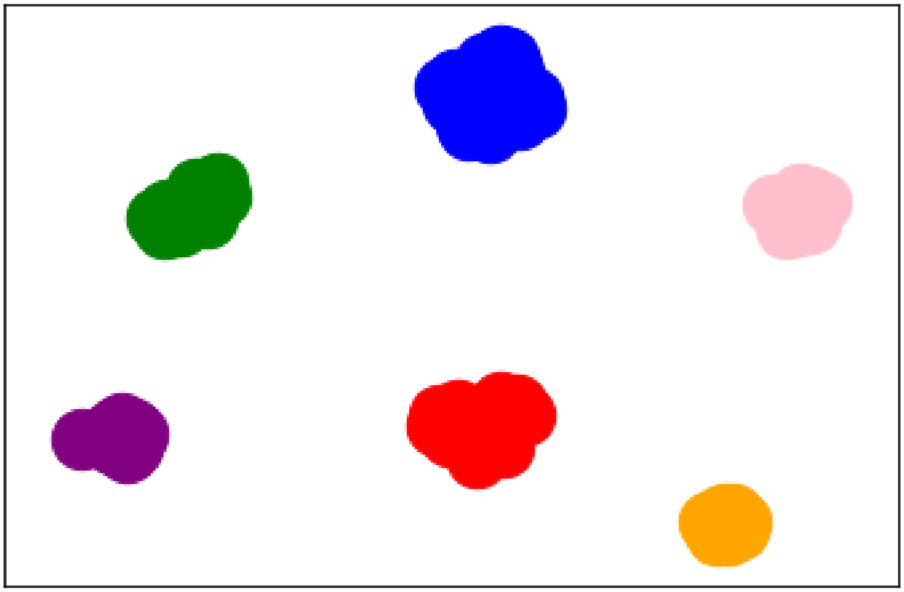
[48]:
```
draw_communities(G,a)
```



```
[49]: a = girvan_newman(G, 6)
      print(a)
```

```
[{'11', '24', '8', '83', '67', '9', '73', '0', '114', '22', '49', '16', '21',
'50', '77', '23', '111', '28', '104', '110', '90', '7', '41', '69', '46', '88',
'78', '68', '51', '108', '4', '53', '93'}, {'37', '25', '33', '80', '103', '35',
'94', '89', '1', '29', '79', '19', '101', '55', '109', '105', '45', '30'},
{'56', '113', '91', '63', '86', '65', '48', '87', '59', '66', '62', '44', '75',
'57', '17', '92', '76', '70', '112', '27', '58', '97', '96', '20', '95'}, {'39',
'13', '6', '2', '100', '32', '64', '47', '60', '106', '15'}, {'98', '107', '81',
'102', '5', '3', '82', '84', '74', '10', '72', '40', '52'}, {'42', '54', '18',
'12', '34', '26', '36', '38', '61', '14', '43', '31', '85', '71', '99'}]
```

```
[50]: draw_communities(G,a)
```



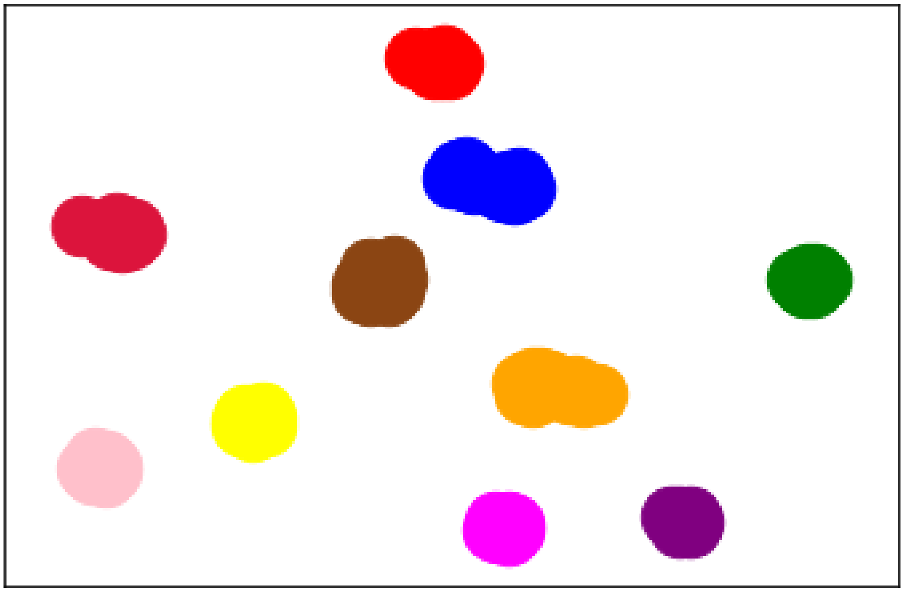
[51]:
```
mod_G_6 = calculate_community_mod(G,6,a)
print(mod_G_6)
```

```
4.293072330973211
```

[52]:
```
a = girvan_newman(G, 10)
print(a)
```

```
[{'104', '22', '7', '111', '41', '16', '8', '78', '68', '108', '21', '77', '4',
'9', '23', '0', '93', '51'}, {'37', '25', '33', '103', '89', '1', '109', '105',
'45'}, {'35', '80', '94', '79', '19', '101', '30', '55', '29'}, {'97', '58',
'96', '17', '59', '20', '56', '113', '62', '63', '76', '27', '70', '95', '65',
'87'}, {'28', '11', '90', '69', '24', '50'}, {'75', '57', '66', '92', '91',
'112', '44', '86', '48'}, {'39', '13', '6', '2', '100', '32', '64', '47', '60',
'106', '15'}, {'98', '107', '81', '102', '5', '3', '82', '84', '74', '10', '72',
'40', '52'}, {'42', '54', '18', '12', '34', '26', '36', '38', '61', '14', '43',
'31', '85', '71', '99'}, {'114', '110', '49', '46', '88', '83', '67', '53',
'73'}]
```

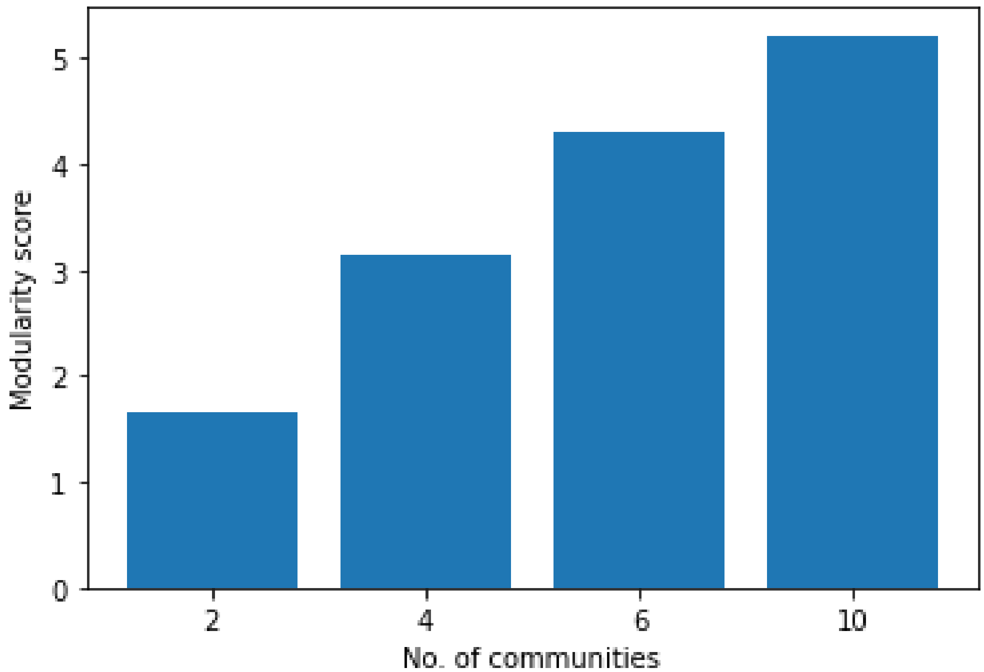
[53]:
```
draw_communities(G,a)
```



[54]:
```
mod_G_10 = calculate_community_mod(G,10,a)
print(mod_G_10)
```

```
5.215111111111103
```

[55]:
```
import matplotlib.pyplot as plt
import numpy as np

x = np.array(["2", "4", "6", "10"])
y = np.array([mod_G_2, mod_G_4, mod_G_6, mod_G_10])

plt.xlabel("No. of communities")
plt.ylabel("Modularity score")

plt.bar(x,y)
plt.show()
```



KARATE CLUB NETWORK

Zachary's karate club is a social network of a university karate club, described in the paper "An Information Flow Model for Conflict and Fission in Small Groups" by Wayne W. Zachary. The network became a popular example of community structure in networks after its use by Michelle Girvan and Mark Newman in 2002.[1]

Network Graph Generated using Gephi

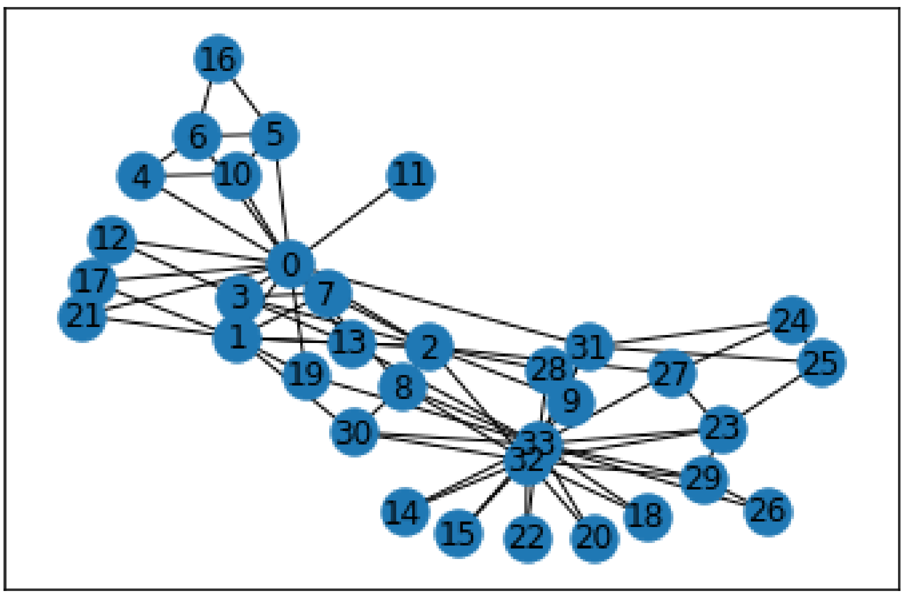
[56]:
```
import pandas as pd

read_file = pd.read_excel ("Karate.xlsx")

read_file.to_csv ("Karate.csv", index = None, header=True)

df = pd.DataFrame(pd.read_csv("Karate.csv"))

df
```

[56]:

	Node1	Node2
0	0	1
1	0	2
2	0	3
3	0	4
4	0	5
..	…	…

```
73      30      33
74      30      32
75      31      32
76      31      33
77      32      33

[78 rows x 2 columns]
```

[57]:
```
G = nx.Graph()
for index, row in df.iterrows():
    G.add_edge(row['Node1'], row['Node2'])

nx.draw_networkx(G, with_labels=True)
```

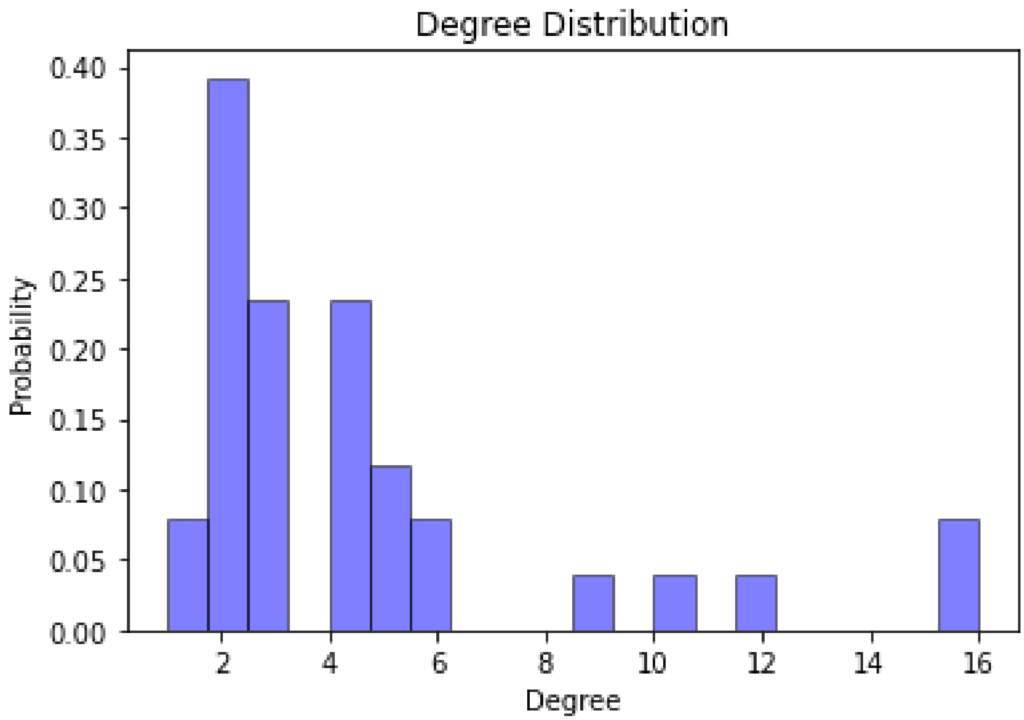


0.3 Network Type Prediction in Karate Club Newtork

[58]:
```
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np
from collections import Counter
from scipy.stats import norm, powerlaw, expon
from scipy.optimize import curve_fit

# Load your dataset as a NetworkX graph (replace this with your specific dataset)
G = nx.read_edgelist("karate.txt")
```

```python
# Compute the degree of each node
degrees = dict(G.degree())
degree_values = list(degrees.values())

# Calculate the degree histogram
hist, bin_edges = np.histogram(degree_values, bins=20, density=True)

# Plot the degree distribution
plt.hist(degree_values, bins=20, density=True, alpha=0.5, color='b',␣
 ↪edgecolor='black')
plt.title("Degree Distribution")
plt.xlabel("Degree")
plt.ylabel("Probability")

# Define functions for fitting different distribution types
def power_law(x, a, b):
    return a * (x**b)

def exponential(x, scale):
    return scale * np.exp(-scale * x)

# Fit the degree distribution to different candidate distributions
params_powerlaw, _ = curve_fit(power_law, bin_edges[:-1], hist)
params_exponential, _ = curve_fit(exponential, bin_edges[:-1], hist)

# Determine the type of distribution based on the parameters
alpha_powerlaw = params_powerlaw[1]
scale_exponential = 1 / params_exponential[0]

if alpha_powerlaw > 2.0:
    print("The degree distribution appears to be closer to a power-law␣
 ↪(scale-free) distribution.")
elif scale_exponential < 2.0:
    print("The degree distribution appears to be closer to an exponential␣
 ↪distribution.")
else:
    print("The degree distribution does not seem to be a power-law or␣
 ↪exponential distribution.")

plt.show()
```

```
The degree distribution does not seem to be a power-law or exponential
distribution.
```



[59]:
```
network_describe(G)
```

```
NETWORK STATISTICS :
Network Info :  Graph with 34 nodes and 77 edges
Nodes :  34
Edges :  77
Degree Centality :
[('0', 0.48484848484848486), ('9', 0.15151515151515152), ('14',
0.15151515151515152), ('15', 0.06060606060606061), ('16', 0.06060606060606061),
('19', 0.06060606060606061), ('20', 0.09090909090909091), ('21',
0.06060606060606061), ('23', 0.06060606060606061), ('24', 0.15151515151515152),
('27', 0.06060606060606061), ('28', 0.12121212121212122), ('29',
0.09090909090909091), ('30', 0.12121212121212122), ('31', 0.12121212121212122),
('32', 0.18181818181818182), ('33', 0.36363636363636365), ('2',
0.2727272727272727), ('1', 0.48484848484848486), ('3', 0.30303030303030304),
('4', 0.18181818181818182), ('5', 0.09090909090909091), ('6',
0.12121212121212122), ('7', 0.12121212121212122), ('8', 0.12121212121212122),
('10', 0.030303030303030304), ('11', 0.09090909090909091), ('12',
0.030303030303030304), ('13', 0.06060606060606061), ('17', 0.06060606060606061),
('18', 0.06060606060606061), ('22', 0.06060606060606061), ('26',
0.09090909090909091), ('25', 0.09090909090909091)]
Betweenness Centality :
[('0', 0.2768743987493986), ('9', 0.057399660524660556), ('14',
0.04674376549376549), ('15', 0.0), ('16', 0.0), ('19', 0.0), ('20',
```

```
0.032513898138898135), ('21', 0.0), ('23', 0.0), ('24', 0.0172979797979798),
('27', 0.0), ('28', 0.025365144115144116), ('29', 0.0035637973137973137), ('30',
0.003291847041847042), ('31', 0.014431401931401931), ('32',
0.13766372516372513), ('33', 0.16064352314352315), ('2', 0.05557324619824619),
('1', 0.43960692085692077), ('3', 0.1741200928700929), ('4',
0.011928696303696303), ('5', 0.0006313131313131313), ('6', 0.02998737373737374),
('7', 0.02998737373737374), ('8', 0.0), ('10', 0.0), ('11',
0.0006313131313131313), ('12', 0.0), ('13', 0.0), ('17', 0.0), ('18', 0.0),
('22', 0.0), ('26', 0.0038404882154882162), ('25', 0.0025252525252525255)]
Eigenvector Centality :
[('0', 0.36028869718137285), ('9', 0.2285824966102697), ('14',
0.22946890344377588), ('15', 0.10020390488798826), ('16', 0.10020390488798826),
('19', 0.10020390488798826), ('20', 0.14964911843723078), ('21',
0.10020390488798826), ('23', 0.10020390488798826), ('24', 0.1490309278297953),
('27', 0.07404311479213908), ('28', 0.1323105473918868), ('29',
0.13012948421188536), ('30', 0.1336439665192374), ('31', 0.17534431969032657),
('32', 0.19206726854315842), ('33', 0.3081600666425366), ('2',
0.2726760840775042), ('1', 0.36533721426632837), ('3', 0.31572719522124854),
('4', 0.21674854874719207), ('5', 0.0790098758057299), ('6',
0.08272774744850384), ('7', 0.08272774744850385), ('8', 0.17545978398577347),
('10', 0.047328600171325085), ('11', 0.07900987580572988), ('12',
0.05476507094052167), ('13', 0.08725625577574166), ('17', 0.024802115794733043),
('18', 0.09563999897922838), ('22', 0.09563999897922838), ('26',
0.05976459234511723), ('25', 0.057584792190228266)]
Average Clustering Coefficient :
0.5710710041592393
Variance :
0.01292255680428062
```

LES MISERABLES NETWORK

Network Graph Generated using Gephi

[60]:
```python
import pandas as pd

read_file = pd.read_excel ("lesmis.xlsx")

read_file.to_csv ("lesmis.csv", index = None, header=True)

df = pd.DataFrame(pd.read_csv("lesmis.csv"))

df
```

[60]:

	Node1	Node2
0	0	1
1	0	2
2	0	3
3	0	4

```
4         0      5
..      …     …
249      69     75
250      70     71
251      70     75
252      71     75
253      73     74

[254 rows x 2 columns]
```

[61]:
```python
G = nx.Graph()
for index, row in df.iterrows():
    G.add_edge(row['Node1'], row['Node2'])
```

[62]:
```python
print(G.edges)
```

```
[(0, 1), (0, 2), (0, 3), (0, 4), (0, 5), (0, 6), (0, 7), (0, 8), (0, 9), (0,
11), (2, 3), (2, 11), (3, 11), (11, 10), (11, 12), (11, 13), (11, 14), (11, 15),
(11, 23), (11, 24), (11, 25), (11, 26), (11, 27), (11, 28), (11, 29), (11, 31),
(11, 32), (11, 33), (11, 34), (11, 35), (11, 36), (11, 37), (11, 38), (11, 43),
(11, 44), (11, 48), (11, 49), (11, 51), (11, 55), (11, 58), (11, 64), (11, 68),
(11, 69), (11, 70), (11, 71), (11, 72), (12, 23), (23, 16), (23, 17), (23, 18),
(23, 19), (23, 20), (23, 21), (23, 22), (23, 24), (23, 25), (23, 27), (23, 29),
(23, 30), (23, 31), (24, 25), (24, 26), (24, 27), (24, 41), (24, 42), (24, 50),
(24, 68), (24, 69), (24, 70), (25, 26), (25, 27), (25, 39), (25, 40), (25, 41),
(25, 42), (25, 48), (25, 55), (25, 68), (25, 69), (25, 70), (25, 71), (25, 75),
(26, 16), (26, 27), (26, 43), (26, 49), (26, 51), (26, 54), (26, 55), (26, 72),
(27, 28), (27, 29), (27, 31), (27, 33), (27, 43), (27, 48), (27, 58), (27, 68),
(27, 69), (27, 70), (27, 71), (27, 72), (28, 44), (28, 45), (29, 34), (29, 35),
(29, 36), (29, 37), (29, 38), (31, 30), (34, 35), (34, 36), (34, 37), (34, 38),
(35, 36), (35, 37), (35, 38), (36, 37), (36, 38), (37, 38), (48, 47), (48, 55),
(48, 57), (48, 58), (48, 59), (48, 60), (48, 61), (48, 62), (48, 63), (48, 64),
(48, 65), (48, 66), (48, 68), (48, 69), (48, 71), (48, 73), (48, 74), (48, 75),
(48, 76), (49, 50), (49, 51), (49, 54), (49, 55), (49, 56), (51, 52), (51, 53),
(51, 54), (51, 55), (55, 16), (55, 39), (55, 41), (55, 54), (55, 56), (55, 57),
(55, 58), (55, 59), (55, 61), (55, 62), (55, 63), (55, 64), (55, 65), (58, 57),
(58, 59), (58, 60), (58, 61), (58, 62), (58, 63), (58, 64), (58, 65), (58, 66),
(58, 70), (58, 76), (64, 57), (64, 59), (64, 60), (64, 61), (64, 62), (64, 63),
(64, 65), (64, 66), (64, 76), (68, 41), (68, 69), (68, 70), (68, 71), (68, 75),
(69, 41), (69, 70), (69, 71), (69, 75), (70, 41), (70, 71), (70, 75), (71, 41),
(71, 75), (16, 17), (16, 18), (16, 19), (16, 20), (16, 21), (16, 22), (17, 18),
(17, 19), (17, 20), (17, 21), (17, 22), (18, 19), (18, 20), (18, 21), (18, 22),
(19, 20), (19, 21), (19, 22), (20, 21), (20, 22), (21, 22), (41, 42), (41, 57),
(41, 62), (41, 75), (39, 52), (57, 59), (57, 61), (57, 62), (57, 63), (57, 65),
(57, 67), (62, 59), (62, 60), (62, 61), (62, 63), (62, 65), (62, 66), (62, 76),
(46, 47), (59, 60), (59, 61), (59, 63), (59, 65), (59, 66), (60, 61), (60, 63),
(60, 65), (60, 66), (61, 63), (61, 65), (61, 66), (63, 65), (63, 66), (63, 76),
```

```
(65, 66), (65, 76), (66, 76), (73, 74)]
```

PARTITIONED LES MISERABLES NETWORK

Network Graph Generated using Gephi

[63]:
```
network_describe(G)
```

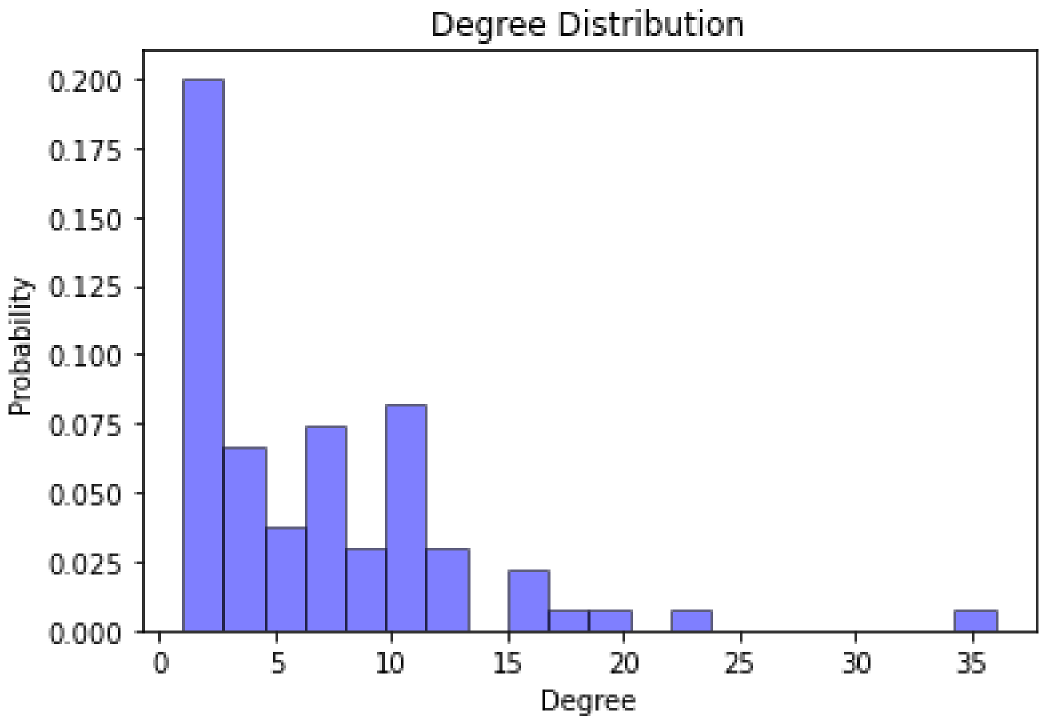
```
NETWORK STATISTICS :
Network Info :  Graph with 77 nodes and 254 edges
Nodes :  77
Edges :  254
Degree Centality :
[(0, 0.13157894736842105), (1, 0.013157894736842105), (2, 0.039473684210526314),
(3, 0.039473684210526314), (4, 0.013157894736842105), (5, 0.013157894736842105),
(6, 0.013157894736842105), (7, 0.013157894736842105), (8, 0.013157894736842105),
(9, 0.013157894736842105), (11, 0.47368421052631576), (10,
0.013157894736842105), (12, 0.02631578947368421), (13, 0.013157894736842105),
(14, 0.013157894736842105), (15, 0.013157894736842105), (23,
0.19736842105263158), (24, 0.14473684210526316), (25, 0.21052631578947367), (26,
0.14473684210526316), (27, 0.22368421052631576), (28, 0.05263157894736842), (29,
0.10526315789473684), (31, 0.05263157894736842), (32, 0.013157894736842105),
(33, 0.02631578947368421), (34, 0.07894736842105263), (35, 0.07894736842105263),
(36, 0.07894736842105263), (37, 0.07894736842105263), (38, 0.07894736842105263),
(43, 0.039473684210526314), (44, 0.02631578947368421), (48, 0.2894736842105263),
(49, 0.09210526315789473), (51, 0.09210526315789473), (55, 0.25), (58,
0.19736842105263158), (64, 0.17105263157894735), (68, 0.13157894736842105), (69,
0.13157894736842105), (70, 0.13157894736842105), (71, 0.11842105263157894), (72,
0.039473684210526314), (16, 0.11842105263157894), (17, 0.09210526315789473),
(18, 0.09210526315789473), (19, 0.09210526315789473), (20, 0.09210526315789473),
(21, 0.09210526315789473), (22, 0.09210526315789473), (30, 0.02631578947368421),
(41, 0.14473684210526316), (42, 0.039473684210526314), (50,
0.02631578947368421), (39, 0.039473684210526314), (40, 0.013157894736842105),
(75, 0.09210526315789473), (54, 0.05263157894736842), (45,
0.013157894736842105), (52, 0.02631578947368421), (57, 0.14473684210526316),
(62, 0.17105263157894735), (46, 0.013157894736842105), (47,
0.02631578947368421), (59, 0.14473684210526316), (60, 0.11842105263157894), (61,
0.14473684210526316), (63, 0.15789473684210525), (65, 0.15789473684210525), (66,
0.13157894736842105), (73, 0.02631578947368421), (74, 0.02631578947368421), (76,
0.09210526315789473), (56, 0.02631578947368421), (53, 0.013157894736842105),
(67, 0.013157894736842105)]
Betweenness Centality :
[(0, 0.17684210526315788), (1, 0.0), (2, 0.0), (3, 0.0), (4, 0.0), (5, 0.0), (6,
0.0), (7, 0.0), (8, 0.0), (9, 0.0), (11, 0.5699890527836186), (10, 0.0), (12,
0.0), (13, 0.0), (14, 0.0), (15, 0.0), (23, 0.12964454098819425), (24,
0.02900241873046176), (25, 0.07490122123424227), (26, 0.023796253454148184),
(27, 0.05433155966478437), (28, 0.026491228070175437), (29,
0.008040935672514618), (31, 0.008640295033483887), (32, 0.0), (33, 0.0), (34,
0.0), (35, 0.0), (36, 0.0), (37, 0.0), (38, 0.0), (43, 0.0), (44, 0.0), (48,
```

0.1651125024258477), (49, 0.020210621583197756), (51, 0.047598927875243655), (55, 0.132032488621946), (58, 0.0425533568221771), (64, 0.030753650179957816), (68, 0.004960383978389518), (69, 0.004960383978389518), (70, 0.0048618041955992095), (71, 0.0038738298738298736), (72, 0.0), (16, 0.04062934817733579), (17, 0.0), (18, 0.0), (19, 0.0), (20, 0.0), (21, 0.0), (22, 0.0), (30, 0.0), (41, 0.011487550654163002), (42, 0.0), (50, 0.00021720969089390142), (39, 0.006925438596491229), (40, 0.0), (75, 0.00043859649122807013), (54, 0.0), (45, 0.0), (52, 0.0003508771929824561), (57, 0.027661236424394314), (62, 0.005267029881988332), (46, 0.0), (47, 0.02631578947368421), (59, 0.0012501455659350393), (60, 0.0), (61, 0.0012501455659350393), (63, 0.0021854883087570063), (65, 0.0021854883087570063), (66, 0.00015037593984962405), (73, 0.0), (74, 0.0), (76, 0.0), (56, 0.0), (53, 0.0), (67, 0.0)]
Eigenvector Centality :
[(0, 0.028134336026755365), (1, 0.0023434559950116173), (2, 0.026872999836996418), (3, 0.026872999836996418), (4, 0.0023434559950116173), (5, 0.0023434559950116173), (6, 0.0023434559950116173), (7, 0.0023434559950116173), (8, 0.0023434559950116173), (9, 0.0023434559950116173), (11, 0.26761817598853926), (10, 0.022291152877501864), (12, 0.029767714740768846), (13, 0.022291152877501864), (14, 0.022291152877501864), (15, 0.022291152877501864), (23, 0.08975922949834111), (24, 0.12228242172143362), (25, 0.1878077051550094), (26, 0.11103702398134034), (27, 0.184225163210257), (28, 0.04004860673568995), (29, 0.06227506194694945), (31, 0.046055011005835665), (32, 0.022291152877501864), (33, 0.03763613953656061), (34, 0.04120828302958653), (35, 0.04120828302958653), (36, 0.04120828302958653), (37, 0.04120828302958653), (38, 0.04120828302958653), (43, 0.046884942434314236), (44, 0.025627007259539696), (48, 0.31783893977497674), (49, 0.06539729702430212), (51, 0.063169051443388), (55, 0.2591111453417876), (58, 0.26717863282356663), (64, 0.24213078637474134), (68, 0.14543155406624994), (69, 0.14543155406624994), (70, 0.14153627306562788), (71, 0.13602919446668402), (72, 0.046884942434314236), (16, 0.04814615586401206), (17, 0.019685736709537924), (18, 0.019685736709537924), (19, 0.019685736709537924), (20, 0.019685736709537924), (21, 0.019685736709537924), (22, 0.019685736709537924), (30, 0.011312731565893446), (41, 0.14193827361489467), (42, 0.03765154186582524), (50, 0.015632758907497154), (39, 0.03792696672003668), (40, 0.01564337736411263), (75, 0.1012869150267418), (54, 0.04153983995608888), (45, 0.0033358543820378303), (52, 0.008420710184175471), (57, 0.19502891203664752), (62, 0.232467197170214), (46, 0.0022204916389402625), (47, 0.026658767697788018), (59, 0.21073457488115607), (60, 0.17581635449396724), (61, 0.21073457488115607), (63, 0.22155360926119957), (65, 0.22155360926119957), (66, 0.1866353888740108), (73, 0.02887925933672828), (74, 0.02887925933672828), (76, 0.14071116072806059), (56, 0.027029413866137036), (53, 0.005261623192198204), (67, 0.01624446552014877)]
Average Clustering Coefficient :
0.5731367499320135
Variance :
0.006233855003506406

[64]:
```
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np
from collections import Counter
from scipy.stats import norm, powerlaw, expon
from scipy.optimize import curve_fit

# Load your dataset as a NetworkX graph (replace this with your specific␣
↪dataset)
G = nx.read_edgelist("lesmis (1).txt")

# Compute the degree of each node
degrees = dict(G.degree())
degree_values = list(degrees.values())

# Calculate the degree histogram
hist, bin_edges = np.histogram(degree_values, bins=20, density=True)

# Plot the degree distribution
plt.hist(degree_values, bins=20, density=True, alpha=0.5, color='b',␣
↪edgecolor='black')
plt.title("Degree Distribution")
plt.xlabel("Degree")
plt.ylabel("Probability")

# Define functions for fitting different distribution types
def power_law(x, a, b):
    return a * (x**b)

def exponential(x, scale):
    return scale * np.exp(-scale * x)

# Fit the degree distribution to different candidate distributions
params_powerlaw, _ = curve_fit(power_law, bin_edges[:-1], hist)
params_exponential, _ = curve_fit(exponential, bin_edges[:-1], hist)

# Determine the type of distribution based on the parameters
alpha_powerlaw = params_powerlaw[1]
scale_exponential = 1 / params_exponential[0]

if alpha_powerlaw > 2.0:
    print("The degree distribution appears to be closer to a power-law␣
↪(scale-free) distribution.")
elif scale_exponential < 2.0:
    print("The degree distribution appears to be closer to an exponential␣
↪distribution.")
else:
```

```
    print("The degree distribution does not seem to be a power-law or␣
↪exponential distribution.")

plt.show()
```

```
The degree distribution does not seem to be a power-law or exponential
distribution.
```




```
[ ]:
```