MATH3090U: Network Science
Take Home Exam

Syed Naqvi
100590852

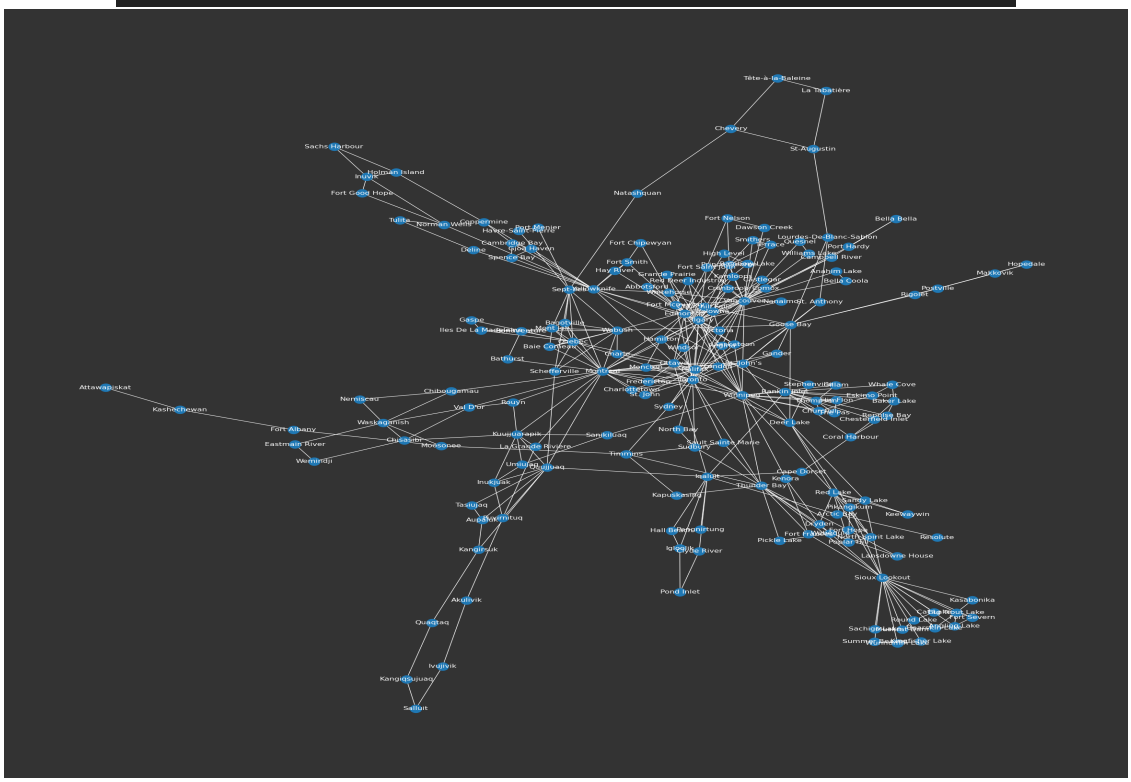
December 15, 2023

1.

(a) Importing data and creating Canadian Airport Network using networkx:

```
G = nx.read_edgelist("Canada_airport_network",
                     delimiter=",",
                     create_using=nx.Graph,
                     data=False)

pos = nx.spring_layout(G)
fig, ax = plt.subplots()
nx.draw(G, with_labels=True,
           edge_color='white',
           font_color='white')
fig.set_facecolor('#333333')
fig.set_size_inches((30,30))
plt.show()
```

[27] ✓ 0.7s

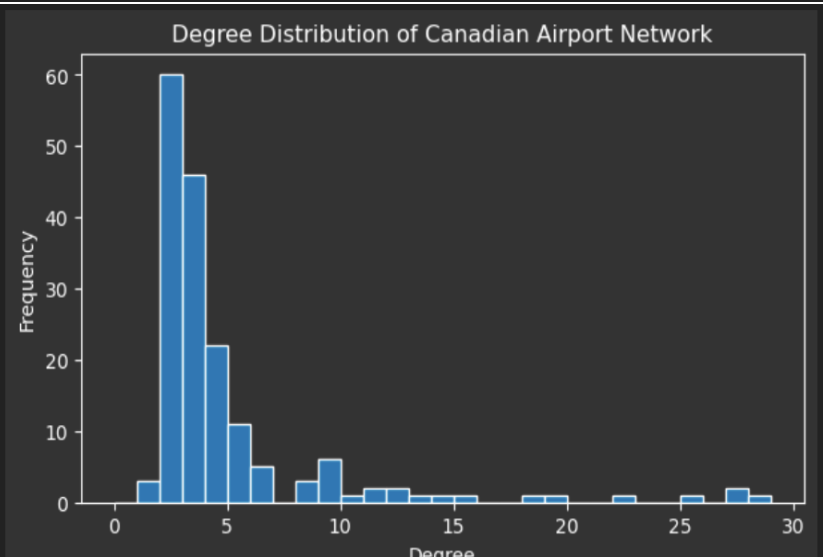


(b) Degree distribution histogram and top ten airports(nodes) in terms of degree:

```python
# CREATION OF HISTOGRAM
d = [degree for node,degree in G.degree()]
fig, ax = plt.subplots()
fig.set_facecolor('#333333')
fig.set_size_inches(7,4)
ax.set_facecolor('#333333')
ax.set_ylabel("Frequency", color='white')
ax.set_xlabel("Degree", color='white')
ax.set_title("Degree Distribution of Canadian Airport Network",
             color='white')
ax.set_xticks(range(0,31,5))
ax.tick_params(colors='white')
plt.setp(ax.spines.values(), color='white')
ax.hist(d, bins=range(0,max(d)+2), edgecolor='white')
#TEN HIGHEST DEGREE NODES
sorted_node_degrees = sorted(G.degree(),
                             key=lambda item: item[1],
                             reverse=True)
for i in range(10):
    print(sorted_node_degrees[i])
plt.show()
```

[69] ✓ 0.1s

```
('Vancouver', 28)
('Toronto', 27)
('Calgary', 27)
('Montreal', 25)
('Edmonton', 22)
('Winnipeg', 19)
('Sioux Lookout', 18)
('Halifax', 15)
('Ottawa', 14)
('Yellowknife', 13)
```



Sioux lookout Airport being on this list was slighlty surprising as it has 18 connections to other airports yet does not reside in a major Canadian city like many of the other nodes on the list. After some research it turns out that Sioux Lookout is sometimes referred to as the "Gateway to the North" and serves as a hub for providing essential services and connections to many remote first nations communities in Northwestern Ontario.

(c) Nodes with top ten closeness centralities:

```
#TOP TEN CLOSENESS CENTRALITIES
closeness_centralities = list(nx.closeness_centrality(G).items())
sorted_closeness_centralities = sorted(closeness_centralities,
                                       key=lambda item: item[1],
                                       reverse=True)
top_ten = sorted_closeness_centralities[:10]
for i in range(10):
    print(top_ten[i])
```

[69] ✓ 0.0s

```
('Toronto', 0.40669856459330145)
('Montreal', 0.4)
('Winnipeg', 0.3890160183066362)
('Calgary', 0.3881278538812785)
('Ottawa', 0.38724373576309795)
('Edmonton', 0.37527593818984545)
('Vancouver', 0.37199124726477023)
('Halifax', 0.35490605427974947)
("St. John's", 0.3420523138832998)
('Quebec', 0.3386454183266932)
```

(d) Nodes with top ten betweenness centralities:

```
#TOP TEN BETWEENNESS CENTRALITIES
betweenness_centralities = list(nx.betweenness_centrality(G).items())
sorted_betweenness_centralities = sorted(betweenness_centralities,
                                         key=lambda item: item[1],
                                         reverse=True)
top_ten = sorted_betweenness_centralities[:10]
for i in range(10):
    print(top_ten[i])
```

[70] ✓ 0.0s

```
('Montreal', 0.29082665887369474)
('Winnipeg', 0.233389710780838)
('Toronto', 0.20220852349025703)
('Thunder Bay', 0.1541636772698015)
('Vancouver', 0.1476909979116367)
('Sioux Lookout', 0.13894203835380306)
('Yellowknife', 0.13396347623936597)
('Ottawa', 0.12343459094905944)
('Calgary', 0.11439352651468462)
('Iqaluit', 0.1026056055766285)
```

(e) The airports with the highest degree centrality are those that have the most direct connections to other airports. Usually this indicates that the airport is in a large metropolitan area but can also indicate a central hub that serves as a connection point for many surrounding airports such as with Sioux Lookout.

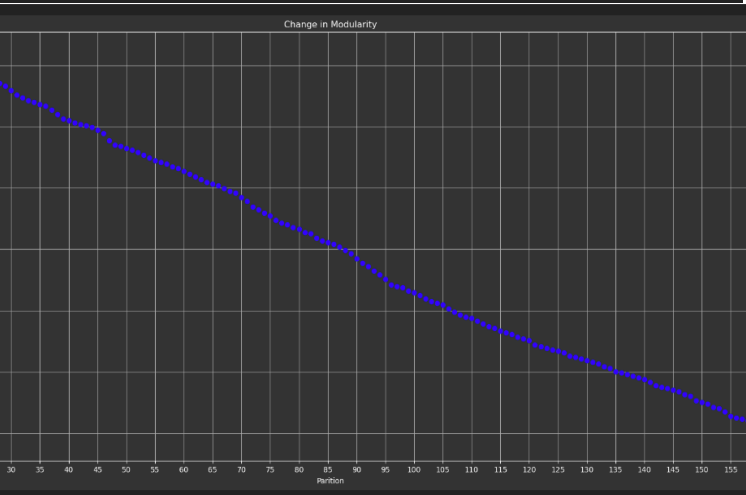
The closeness centrality is a better indication of a "main airport". An airport with high closeness centrality is one from which the rest of the country is accessible via minimum intermediary stops. Unsurprisingly, the top ten nodes with highest closeness centralities are all captial cities or highly populated cities. This is to be expected since these are locations most people would be traveling to or from and so would be closeset to all other airports.

A high betweenness centrality indicates an airport that sits between different clusters of airports. Such airports tend to be on many shortest flight routes between airports in opposing parts of the network. We can see this reflected in the fact that most airports on the top ten betweenness centrality list are in geographically central parts of Canada, thus lying on the shortest flight routes between east and west cost. A few others, althout not geographically central, serve as local bottlenecks separating smaller clusters from the rest of the network such as Sioux Lookout and Vancouver.

(f) Using Girvan Newman Algorithm to determine a parition with highest modularity:

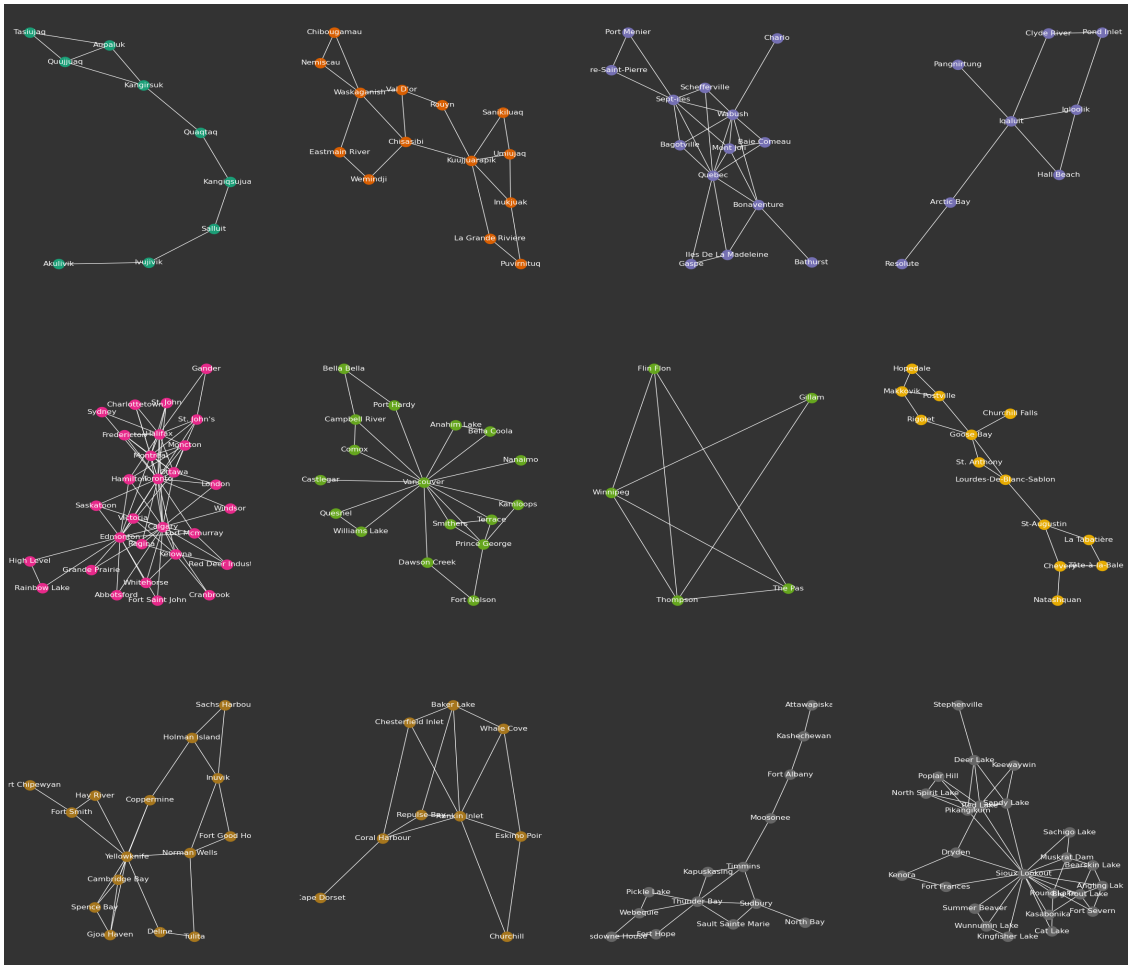
```python
communities_generator = nx.community.girvan_newman(G)
x=[]
mod=[]
partitions=[]
counter=0

for c in communities_generator:
    counter+=1
    x.append(counter)
    mod.append(nx.community.modularity(G,c))
    partitions.append(c)

fig, ax = plt.subplots()
fig.set_facecolor('#333333')
fig.set_size_inches(25,10)
ax.set_facecolor('#333333')
ax.set_ylabel("Modularity", color='white')
ax.set_xlabel("Parition", color='white')
ax.set_xticks(range(0,171,5))
ax.set_title("Change in Modularity", color='white')
ax.tick_params(colors='white')
plt.setp(ax.spines.values(), color='white')
ax.plot(x,mod,'bo')
ax.margins(x=0.02)
ax.grid()
plt.show()
print(f"Maximum modularity = {max(mod)}",
      f"\nPartition with Maximum Modularity = {mod.index(max(mod))}")
```

[36] ✓ 4.5s




```
Maximum modularity = 0.6224924274630673
Partition with Maximum Modularity = 10
```

(g) I will be dicussing the community structure from part (f) with the visualization presented in this part.



One notable observation of this network is the strong separation by region and by the strongest hubs. Overall, the communities seem to be split by region likely due to the fact that geographically closer airports have more direct flights between each other than with other airports in the network. This also makes sense when we consider that the Girvan-Newman approach uses betweenness centrality to remove edges for each partition and so connection between regions would be removed leaving the different regions as distinct communities.

It was also notable that the most central hub airports such as Toronto, Montreal, Ottawa, Calgary and their surrounding airports also formed one large community. This is probably because of the strong connections between each of these airports leading to multiple paths from one of these central airports to their peripheral/regional connections resulting in the entirety of this cluster becoming one community.

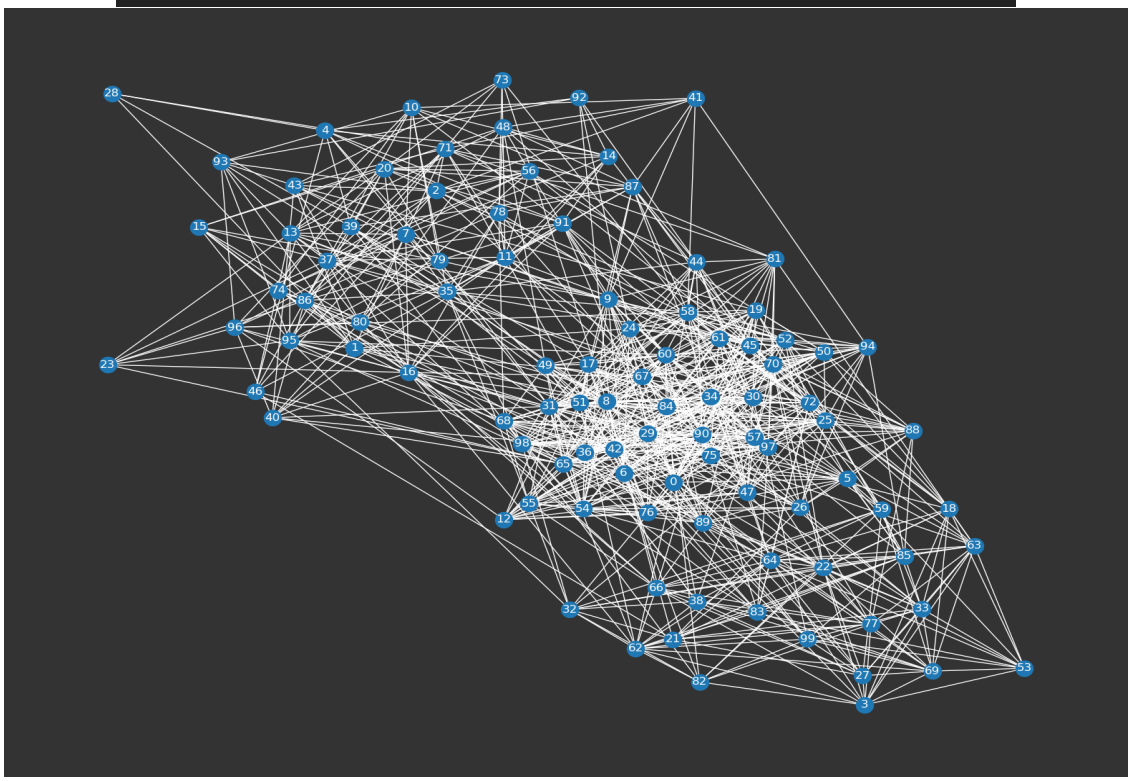
I think this is a strong visualization as we can clearly see all the communities from the parition with the largest modularity score. To create this network, I first stored the list of sets of nodes constituting the strongest partition, lets call this P_{best}. I then iterated through the original graph and removed all edges between nodes that did not belong to the same community as determined by P_{best}. After the graph had been separated into components where each component was a community, i then generated 12 subplots, and used nx.draw() to draw each component as a separate graph on its own plot out of the 12 subplots. I also colored each plot with a different color from the other plots for more readability.

2.

(a) Importing data and creating Social Network using networkx:

```
import pandas as pd

nodesdf = pd.read_csv('socialnetwork_attributes.csv')
edgesdf = pd.read_csv('socialnetwork_edges.csv')

A = nx.Graph()
for index, row in nodesdf.iterrows():
    A.add_node(row['node_id'], **row[1:].to_dict())

for index, row in edgesdf.iterrows():
    A.add_edge(row[0],row[1])

pos = nx.spring_layout(A)
fig, ax = plt.subplots()
nx.draw(A, with_labels=True,
            edge_color='white',
            font_color='white')
fig.set_facecolor('#333333')
fig.set_size_inches((20,15))
plt.show()
```

[141] ✓ 0.4s



(b) Attributes and possible options:

```
unique_ages = set([A.nodes[i]['age'] for i in A.nodes()])
unique_int = set([A.nodes[i]['weekly internet usage'] for i in A.nodes()])
unique_stream = set([A.nodes[i]['weekly streaming'] for i in A.nodes()])

print("Attribute Names: age, weekly internet usage, weekly streaming\n")
print("Possible Values for Attributes:")
print("age = ", unique_ages)
print("weekly internet usage = ", unique_int)
print("weekly streaming = ", unique_stream)
```

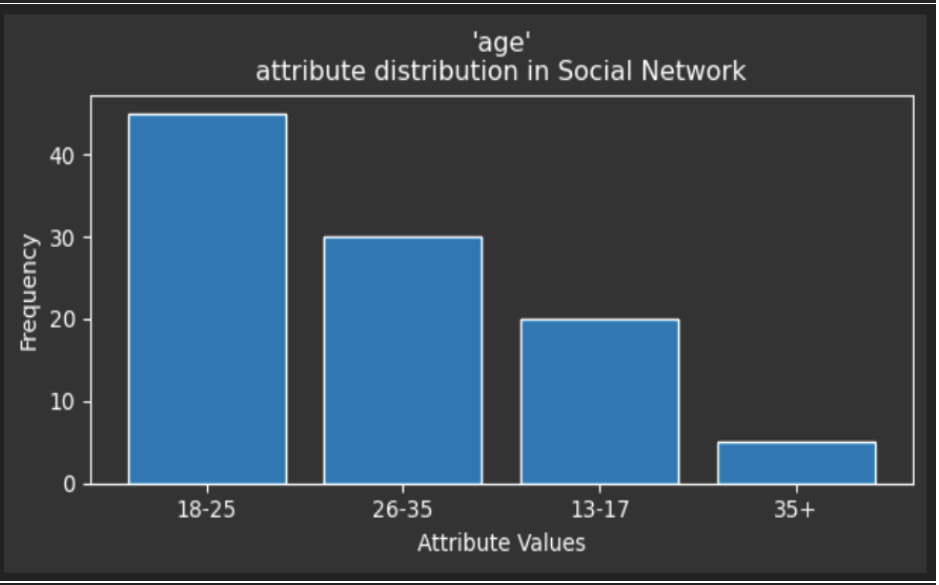
[147] ✓ 0.0s

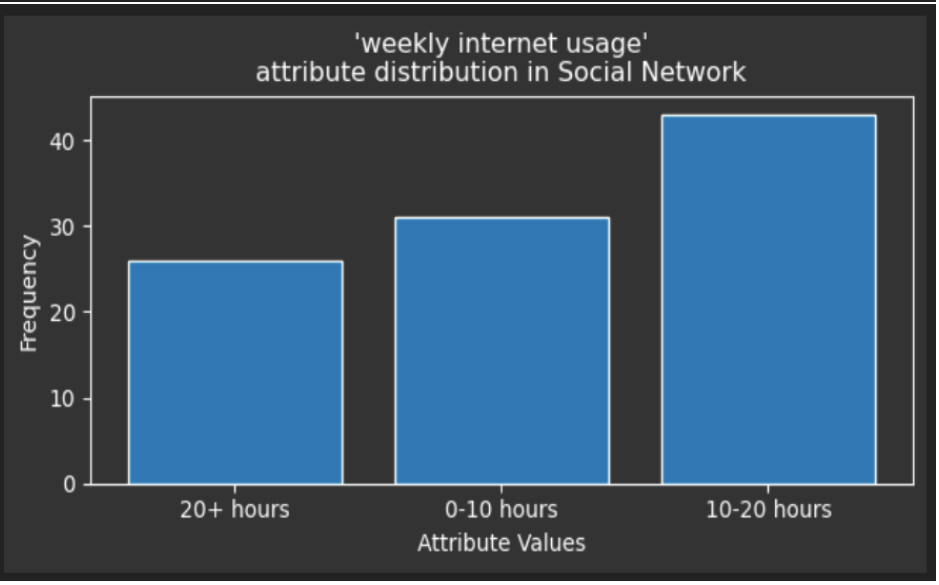
```
Attribute Names: age, weekly internet usage, weekly streaming

Possible Values for Attributes:
age =  {'13-17', '18-25', '35+', '26-35'}
weekly internet usage =  {'20+ hours', '0-10 hours', '10-20 hours'}
weekly streaming =  {'medium', 'low', 'high'}
```

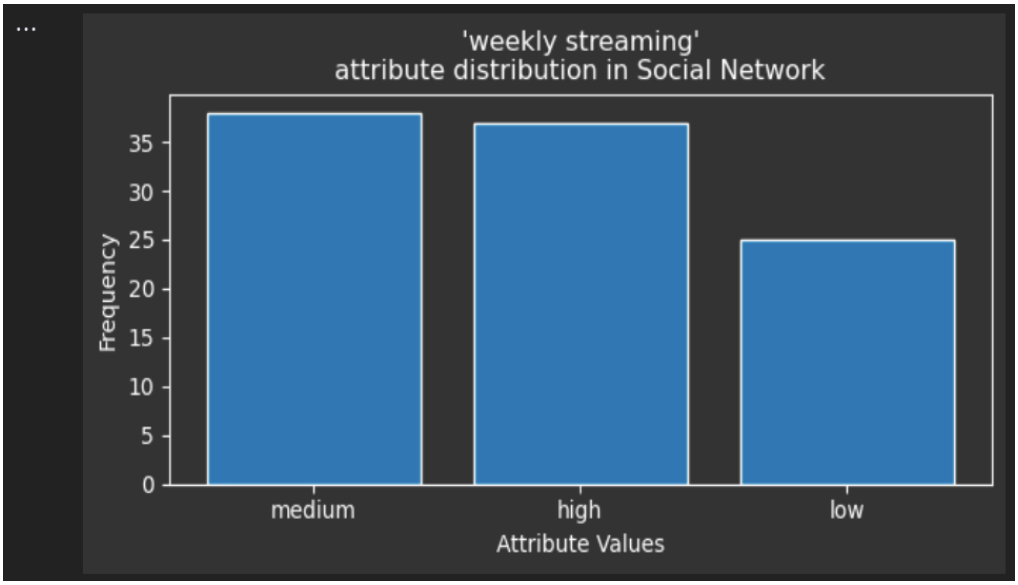
(c) Attribute value distributions:

```python
from collections import Counter

for atts in A.nodes[0].keys():
    hist = Counter([A.nodes[i][atts] for i in A.nodes()])
    fig,ax = plt.subplots()
    ax.bar(hist.keys(), hist.values(), edgecolor='white')
    ax.set_ylabel("Frequency", color='white')
    ax.set_xlabel("Attribute Values", color='white')
    ax.set_title(f"\'{atts}\'\nattribute distribution in Social Network",
        color='white')
    ax.set_facecolor('#333333')
    ax.tick_params(colors='white')
    plt.setp(ax.spines.values(), color='white')
    fig.set_size_inches(7,3)
    fig.set_facecolor('#333333')
    plt.show()
```

[169] ✓ 0.3s







(d) Modularites based on attribute value partitioning:

```
def partition_nodes(A, attribute, unique_values):
    partition = []
    for unique_value in unique_values:
        s = [n for n in A.nodes() if A.nodes[n][attribute] == unique_value]
        partition.append(set(s))
    return partition

age_partition = partition_nodes(A, 'age', unique_ages)
int_partition = partition_nodes(A, 'weekly internet usage', unique_int)
stream_partition = partition_nodes(A, 'weekly streaming', unique_stream)

print("Modularities based on attribute value partitioning:\n")
print("\'age\' = ", nx.community.modularity(A,age_partition))
print("\'weekly internet usage\' = ", nx.community.modularity(A,int_partition))
print("\'weekly streaming\' = ", nx.community.modularity(A,stream_partition))
```

[185] ✓ 0.0s

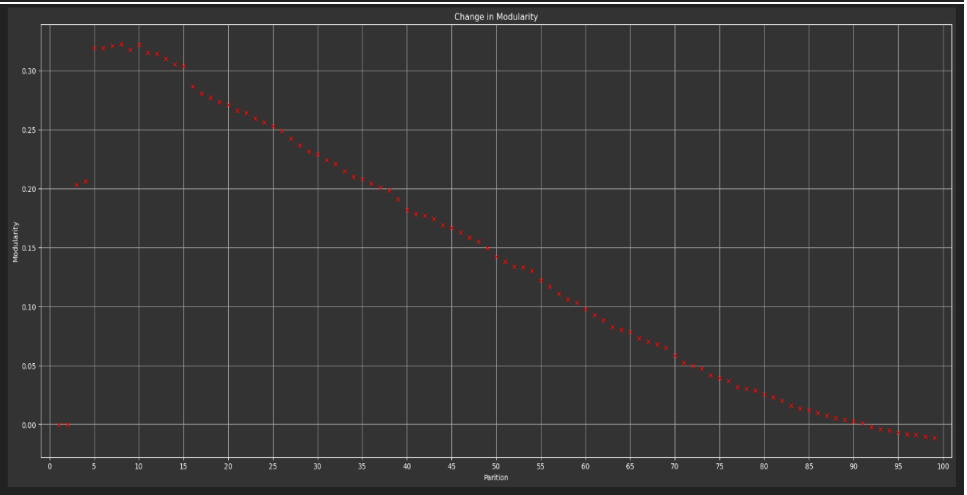
```
Modularities based on attribute value partitioning:

'age' =  0.33024624725923424
'weekly internet usage' =  0.03853600944510034
'weekly streaming' =  0.11829650868611909
```

(e) The findings from part d suggest that people in similar age groups are more likely to be friends. If we were to consider a graphical representation of partitioning based on the 'age' attribute, we would find that the strongest communities are formed amongst nodes in the same age bracket.

(f)

```python
communities_generator = nx.community.girvan_newman(A)
x=[]
mod=[]
partitions=[]
counter=0

for c in communities_generator:
    counter+=1
    x.append(counter)
    mod.append(nx.community.modularity(A,c))
    partitions.append(c)

fig, ax = plt.subplots()
fig.set_facecolor('#333333')
fig.set_size_inches(25,10)
ax.set_facecolor('#333333')
ax.set_ylabel("Modularity", color='white')
ax.set_xlabel("Parition", color='white')
ax.set_xticks(range(0,101,5))
ax.set_title("Change in Modularity", color='white')
ax.tick_params(colors='white')
plt.setp(ax.spines.values(), color='white')
ax.plot(x,mod,'bo')
ax.margins(x=0.02)
ax.grid()
plt.show()
print(f"Maximum modularity = {max(mod)}",
      f"\nPartition with Maximum Modularity = {mod.index(max(mod))}")
```

[187] ✓ 12.1s




```
Maximum modularity = 0.32255776690841625
Partition with Maximum Modularity = 7
```

We observe that using the Girvan-Newman method in order to find the best partition of a graph based on modularity may not always capture community structures perfectly. Here our best partitioning achieves a maximum modularity of approx 0.323 which although very close to the partitioning based on age (0.330246) is not as high.

(g)

```python
def strong_communities(G, partition):
    str_coms = []
    for c in partition:
        add = True
        for n in c:
            indeg = 0
            outdeg = 0
            for adj in G.neighbors(n):
                if find_set(partition, adj) == find_set(partition, n):
                    indeg+=1
                else:
                    outdeg+=1
            if outdeg >= indeg:
                add = False
        if add:
            str_coms.append(c)
    return str_coms

def weak_communities(G, partition):
    weak_coms = []
    for c in partition:
        insum = 0
        outsum = 0
        for n in c:
            for adj in G.neighbors(n):
                if find_set(partition, adj) == find_set(partition, n):
                    insum+=1
                else:
                    outsum+=1
        if insum > outsum:
                weak_coms.append(c)
    return weak_coms

print("Weak Communities:\n", weak_communities(A,stream_partition))
print()
print("Strong Communities:\n", strong_communities(A,stream_partition))
```

[191] ✓ 0.0s

```
Weak Communities:
 [{3, 16, 18, 21, 22, 27, 32, 38, 52, 53, 59, 62, 63, 64, 66, 69, 72, 77, 80, 81, 83, 84, 85, 88, 99}]

Strong Communities:
 []
```

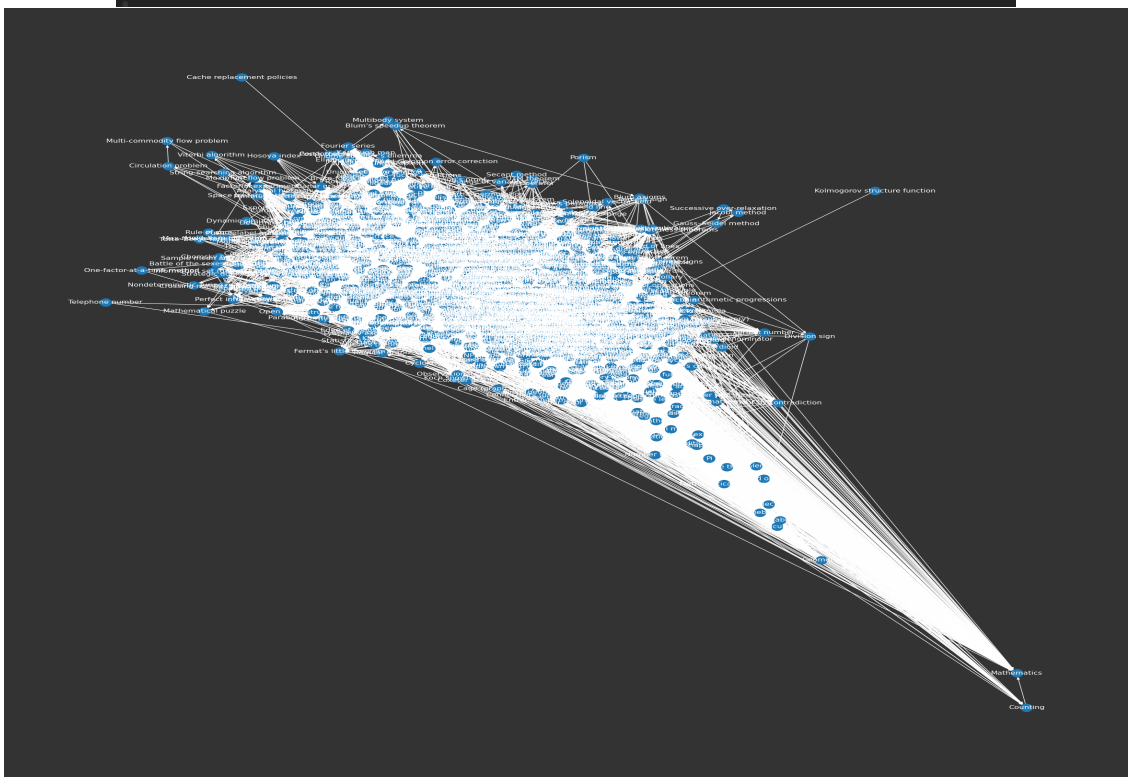
Only one of the subgraphs induced by the streaming usage attribute partition forms a weak community and none form a strong community.

3.

(a) Importing data and creating Wiki Page Network using networkx:

```
W = nx.read_edgelist("Wiki_math",
                     delimiter="\t",
                     create_using=nx.DiGraph,
                     data=False)
pos = nx.spring_layout(W)
fig, ax = plt.subplots()
nx.draw(W, with_labels=True,
           edge_color='white',
           font_color='white')
fig.set_facecolor('#333333')
fig.set_size_inches((30,30))
plt.show()
```

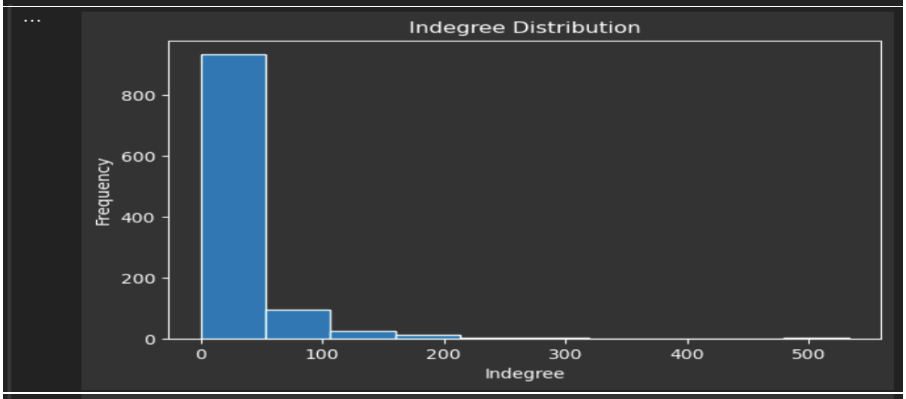
[206] ✓ 1m 34.1s

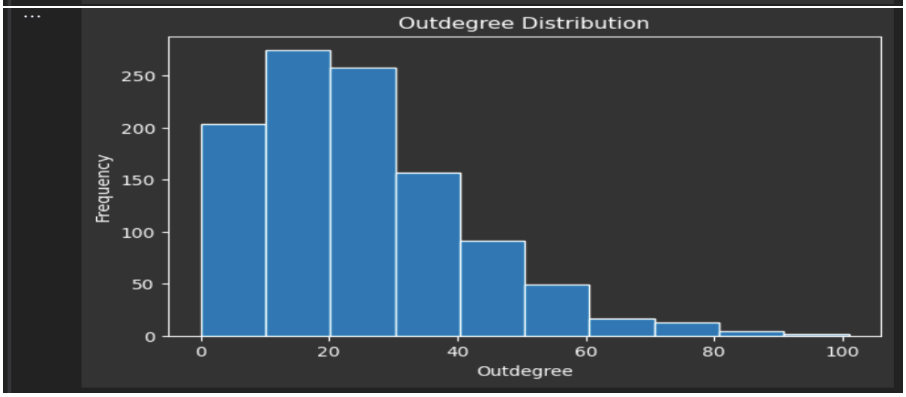


(b) Indegree and Outdegree distributions:

```python
# Get the in-degrees and out-degrees
in_degrees = [deg for node, deg in W.in_degree()]
out_degrees = [deg for node, deg in W.out_degree()]

for x in [("Indegree",in_degrees),("Outdegree",out_degrees)]:
    fig,ax = plt.subplots()
    ax.hist(x[1], edgecolor='white')
    ax.set_ylabel("Frequency", color='white')
    ax.set_xlabel(x[0], color='white')
    ax.set_title(f"{x[0]} Distribution",
        color='white')
    ax.set_facecolor('#333333')
    ax.tick_params(colors='white')
    plt.setp(ax.spines.values(), color='white')
    fig.set_size_inches(7,4)
    fig.set_facecolor('#333333')
    plt.show()
```

[217] ✓ 0.2s





i) Only the Indgree distribution follows the power-law.

ii) I believe these distributions do make sense. The majority of pages would be linked to by very few other pages while a small number of nodes representing the fundamental concepts in mathematics such as algebra, calculus, geometry, etc... would be linked to by almost everything else. This would create a power law for the indegree distribution.

As for the outdegree, we could expect that the majority of topics in math are related to a moderate amount of other topics, the majority of which being in the same core branch. Very few topics would be related to a very large number of other topics and so the right skewed graph for the number of pages a given page links does seem reasonable.

(c) Converting the network to an undirected graph:

```
# Convert the directed graph W to an undirected graph U
U = W.to_undirected()
# Now U is an undirected version of W
```

[218] ✓ 13.5s

(d) Top ten betweenness centralities:

```
#TOP TEN BETWEENNESS CENTRALITIES
bc = list(nx.betweenness_centrality(U).items())
sorted_bc = sorted(bc, key=lambda item: item[1], reverse=True)
top_ten = sorted_bc[:10]
for i in range(10):
    print(top_ten[i])
```

[220] ✓ 7.8s

```
('Mathematics', 0.14548605412675597)
('Function (mathematics)', 0.03574646072960819)
('Real number', 0.03318836559516409)
('Algorithm', 0.029452691330308346)
('Graph theory', 0.02602351259021627)
('Integer', 0.01894963376857839)
('Statistics', 0.014505511948628643)
('Geometry', 0.014149613729872383)
('Natural number', 0.01350134183089047)
('Complex number', 0.013226495300095413)
```

The core branches of mathematics are represented by nodes with the highest betweenness centralities. This is logical since every topic in math is categorized under one or more of these branches. Consequently, the branches serve as links relating seperate topics to each other. This justifies their high betweenness centralities.

(e)

```
import community as community_louvain

# Compute the best partition using the Louvain method
partition = community_louvain.best_partition(U)

# Convert the partition to a list of communities
communities = {}
for node, community in partition.items():
    communities.setdefault(community, set()).add(node)

communities = list(communities.values())

modularity = nx.community.modularity(U,communities)

print("modularity = ", modularity)
# print(communities[2])
```

[233] ✓ 0.4s

```
modularity =  0.4514611545209169
```

Using the louvain algorithm we find a partition with maximum modularity of approx 0.451 which indicates good community structure.

(f) Looking at the third community, we see that the main theme seems to be calculus and differential equations.

(g) Converting from a directed to undirected graph would be approriate depending on the type of analysis being performed. For example, in the context of the math pages graph, all topics eventually point back to their containing branch and so converting the edges to undirected is an effective way to detect communitites. This is because in both the the directed and undirected case, and edge retains its main purpose of depicting relation between topics.