**Question 1:**

Random model graph:

The graph calculates the probability of connecting any two nodes based on the average degree and uses this to generate a random graph with NetworkX's G(n, p) model. In this model, 'n' is the number of nodes and 'p' is the probability of each pair of nodes being connected.

The connectivity of the graph is verified by calculating the average shortest path length between nodes.

The likelihood of connections between nodes is determined using the average degree, and a random graph is constructed accordingly using NetworkX's G(n, p) model, where 'n' denotes the total nodes and 'p' the connection probability.

Small world model:

The function measures the average path length and clustering coefficient for the small-world graph it generates.

Parameters specific to the E. coli network, such as the number of nodes, the nearest integer value for nearest neighbors, and the probability of rewiring, are established.

These parameters are used to initiate the function that creates the small-world network, after which the network's metrics are assessed.

Preferential attachment:

The function calculates the average path length and clustering coefficient for the network it generates.

Parameters specific to simulating the E. coli network are defined, including the node count and edge count, the latter of which is determined by halving the average degree.

With these parameters, the function is executed to construct the preferential attachment network, and its key metrics are then computed.

**Question 2:**

1-Cliques = {Nicholas, Layla, Cindy}

2-Cliques = {Tushar, Nicholas, Ahmed, Wusheng, Layla, Cindy , Akshat, Priyanka}

3-Cliques = {Tushar, Nicholas, Ahmed, Wusheng, Layla, Cindy, Alfredo, Komal, Akshat, Priyanka}

C:\Users\kiran> & C:/Users/kiran/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/kiran/kplexes.py

1-plex:

['Cindy', 'Nicholas', 'Layla']

2-plex:

['Tushar', 'Ahmed', 'Wusheng', 'Nicholas']

['Tushar', 'Wusheng', 'Nicholas', 'Layla']

['Cindy', 'Komal', 'Tushar', 'Nicholas']

['Alfredo', 'Tushar', 'Wusheng', 'Komal']

['Ahmed', 'Wusheng', 'Nicholas', 'Layla']

['Cindy', 'Akshat', 'Priyanka', 'Nicholas']

['Cindy', 'Priyanka', 'Wusheng', 'Layla']

3-plex:

['Tushar', 'Nicholas', 'Wusheng', 'Ahmed', 'Layla']

['Tushar', 'Nicholas', 'Wusheng', 'Layla', 'Cindy']

['Tushar', 'Nicholas', 'Wusheng', 'Cindy', 'Priyanka']

['Tushar', 'Nicholas', 'Wusheng', 'Akshat', 'Priyanka']

['Tushar', 'Nicholas', 'Komal', 'Layla', 'Cindy']

['Tushar', 'Wusheng', 'Komal', 'Layla', 'Cindy']

['Tushar', 'Wusheng', 'Komal', 'Cindy', 'Priyanka']

['Nicholas', 'Wusheng', 'Ahmed', 'Layla', 'Cindy']

['Nicholas', 'Wusheng', 'Ahmed', 'Cindy', 'Priyanka']

['Nicholas', 'Wusheng', 'Ahmed', 'Akshat', 'Priyanka']

['Nicholas', 'Wusheng', 'Layla', 'Cindy', 'Priyanka']

['Nicholas', 'Wusheng', 'Layla', 'Akshat', 'Priyanka']

['Nicholas', 'Layla', 'Cindy', 'Akshat', 'Priyanka']

['Wusheng', 'Komal', 'Layla', 'Cindy', 'Alfredo']

['Wusheng', 'Komal', 'Cindy', 'Priyanka', 'Alfredo']

4-plex:

['Tushar', 'Nicholas', 'Wusheng', 'Komal', 'Ahmed', 'Cindy']

['Tushar', 'Nicholas', 'Wusheng', 'Komal', 'Ahmed', 'Alfredo']

['Tushar', 'Nicholas', 'Wusheng', 'Komal', 'Layla', 'Cindy']

['Tushar', 'Nicholas', 'Wusheng', 'Komal', 'Layla', 'Alfredo']

['Tushar', 'Nicholas', 'Wusheng', 'Komal', 'Cindy', 'Priyanka']

['Tushar', 'Nicholas', 'Wusheng', 'Komal', 'Cindy', 'Alfredo']

['Tushar', 'Nicholas', 'Wusheng', 'Ahmed', 'Layla', 'Cindy']

['Tushar', 'Nicholas', 'Wusheng', 'Ahmed', 'Cindy', 'Priyanka']

['Tushar', 'Nicholas', 'Wusheng', 'Ahmed', 'Akshat', 'Priyanka']

['Tushar', 'Nicholas', 'Wusheng', 'Layla', 'Cindy', 'Priyanka']

['Tushar', 'Nicholas', 'Wusheng', 'Layla', 'Akshat', 'Priyanka']

['Tushar', 'Nicholas', 'Wusheng', 'Cindy', 'Akshat', 'Priyanka']

['Tushar', 'Nicholas', 'Komal', 'Cindy', 'Akshat', 'Priyanka']

['Tushar', 'Wusheng', 'Komal', 'Layla', 'Cindy', 'Priyanka']

['Tushar', 'Wusheng', 'Komal', 'Layla', 'Cindy', 'Alfredo']

['Tushar', 'Wusheng', 'Komal', 'Cindy', 'Priyanka', 'Alfredo']

['Nicholas', 'Wusheng', 'Komal', 'Ahmed', 'Cindy', 'Alfredo']

['Nicholas', 'Wusheng', 'Komal', 'Layla', 'Cindy', 'Alfredo']

['Nicholas', 'Wusheng', 'Ahmed', 'Layla', 'Cindy', 'Priyanka']

['Nicholas', 'Wusheng', 'Ahmed', 'Layla', 'Akshat', 'Priyanka']

['Nicholas', 'Wusheng', 'Ahmed', 'Cindy', 'Akshat', 'Priyanka']

['Nicholas', 'Wusheng', 'Layla', 'Cindy', 'Akshat', 'Priyanka']

['Wusheng', 'Komal', 'Layla', 'Cindy', 'Priyanka', 'Alfredo']

5-plex:

['Tushar', 'Nicholas', 'Wusheng', 'Komal', 'Ahmed', 'Layla', 'Cindy']

['Tushar', 'Nicholas', 'Wusheng', 'Komal', 'Ahmed', 'Layla', 'Alfredo']

['Tushar', 'Nicholas', 'Wusheng', 'Komal', 'Ahmed', 'Cindy', 'Priyanka']

['Tushar', 'Nicholas', 'Wusheng', 'Komal', 'Ahmed', 'Cindy', 'Alfredo']

['Tushar', 'Nicholas', 'Wusheng', 'Komal', 'Layla', 'Cindy', 'Priyanka']

['Tushar', 'Nicholas', 'Wusheng', 'Komal', 'Layla', 'Cindy', 'Alfredo']

['Tushar', 'Nicholas', 'Wusheng', 'Komal', 'Cindy', 'Akshat', 'Priyanka']

['Tushar', 'Nicholas', 'Wusheng', 'Komal', 'Cindy', 'Priyanka', 'Alfredo']

['Tushar', 'Nicholas', 'Wusheng', 'Komal', 'Akshat', 'Priyanka', 'Alfredo']

['Tushar', 'Nicholas', 'Wusheng', 'Ahmed', 'Layla', 'Cindy', 'Priyanka']

['Tushar', 'Nicholas', 'Wusheng', 'Ahmed', 'Layla', 'Akshat', 'Priyanka']

['Tushar', 'Nicholas', 'Wusheng', 'Ahmed', 'Cindy', 'Akshat', 'Priyanka']

['Tushar', 'Nicholas', 'Wusheng', 'Layla', 'Cindy', 'Akshat', 'Priyanka']

['Tushar', 'Nicholas', 'Komal', 'Layla', 'Cindy', 'Akshat', 'Priyanka']

['Tushar', 'Wusheng', 'Komal', 'Layla', 'Cindy', 'Priyanka', 'Alfredo']

['Nicholas', 'Wusheng', 'Komal', 'Ahmed', 'Layla', 'Cindy', 'Alfredo']

['Nicholas', 'Wusheng', 'Komal', 'Ahmed', 'Cindy', 'Priyanka', 'Alfredo']

['Nicholas', 'Wusheng', 'Komal', 'Layla', 'Cindy', 'Priyanka', 'Alfredo']

['Nicholas', 'Wusheng', 'Komal', 'Cindy', 'Akshat', 'Priyanka', 'Alfredo']

['Nicholas', 'Wusheng', 'Ahmed', 'Layla', 'Cindy', 'Akshat', 'Priyanka']

6-plex:

['Tushar', 'Nicholas', 'Wusheng', 'Komal', 'Ahmed', 'Layla', 'Cindy', 'Priyanka']

['Tushar', 'Nicholas', 'Wusheng', 'Komal', 'Ahmed', 'Layla', 'Cindy', 'Alfredo']

['Tushar', 'Nicholas', 'Wusheng', 'Komal', 'Ahmed', 'Cindy', 'Akshat', 'Priyanka']

['Tushar', 'Nicholas', 'Wusheng', 'Komal', 'Ahmed', 'Cindy', 'Priyanka', 'Alfredo']

['Tushar', 'Nicholas', 'Wusheng', 'Komal', 'Ahmed', 'Akshat', 'Priyanka', 'Alfredo']

['Tushar', 'Nicholas', 'Wusheng', 'Komal', 'Layla', 'Cindy', 'Akshat', 'Priyanka']

['Tushar', 'Nicholas', 'Wusheng', 'Komal', 'Layla', 'Cindy', 'Priyanka', 'Alfredo']

['Tushar', 'Nicholas', 'Wusheng', 'Komal', 'Layla', 'Akshat', 'Priyanka', 'Alfredo']

['Tushar', 'Nicholas', 'Wusheng', 'Komal', 'Cindy', 'Akshat', 'Priyanka', 'Alfredo']

['Tushar', 'Nicholas', 'Wusheng', 'Ahmed', 'Layla', 'Cindy', 'Akshat', 'Priyanka']

['Nicholas', 'Wusheng', 'Komal', 'Ahmed', 'Layla', 'Cindy', 'Priyanka', 'Alfredo']

['Nicholas', 'Wusheng', 'Komal', 'Ahmed', 'Cindy', 'Akshat', 'Priyanka', 'Alfredo']

['Nicholas', 'Wusheng', 'Komal', 'Layla', 'Cindy', 'Akshat', 'Priyanka', 'Alfredo']

7-plex:

['Tushar', 'Nicholas', 'Wusheng', 'Komal', 'Ahmed', 'Layla', 'Cindy', 'Akshat', 'Priyanka']

['Tushar', 'Nicholas', 'Wusheng', 'Komal', 'Ahmed', 'Layla', 'Cindy', 'Priyanka', 'Alfredo']

['Tushar', 'Nicholas', 'Wusheng', 'Komal', 'Ahmed', 'Layla', 'Akshat', 'Priyanka', 'Alfredo']

['Tushar', 'Nicholas', 'Wusheng', 'Komal', 'Ahmed', 'Cindy', 'Akshat', 'Priyanka', 'Alfredo']

['Tushar', 'Nicholas', 'Wusheng', 'Komal', 'Layla', 'Cindy', 'Akshat', 'Priyanka', 'Alfredo']

['Nicholas', 'Wusheng', 'Komal', 'Ahmed', 'Layla', 'Cindy', 'Akshat', 'Priyanka', 'Alfredo']

8-plex:

['Tushar', 'Nicholas', 'Wusheng', 'Komal', 'Ahmed', 'Layla', 'Cindy', 'Akshat', 'Priyanka', 'Alfredo']



**c.** Differences:

K-plexes vary in density and size, with density being the ratio of actual to potential maximum edges. A K-plex can exceed the size of K and exhibit a lower density compared to a K-clique.

K-plexes permit varying degrees of connectivity among the nodes within the subgraph, offering more structural versatility. They are effective in identifying communities or clusters that demonstrate significant internal connections, even if they are not fully connected.

K-cliques are completely connected network structures consisting of exactly K nodes, where every possible edge between nodes is included, resulting in the highest possible average density.

Across the network, K-cliques represent rigid, fully connected structures. They are commonly utilized to identify tightly knit groups of nodes, where every node is directly linked to every other node in the group.

**Question 3:**

Determining the edge betweenness

PS C:\Users\kiran> & C:/Users/kiran/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/kiran/edgeBetweenness.py

Edge Betweenness:

('Tushar', 'Nicholas') : 5.316666666666666

('Tushar', 'Wusheng') : 4.783333333333333

('Tushar', 'Komal') : 4.033333333333333

('Nicholas', 'Ahmed') : 5.033333333333333

('Nicholas', 'Layla') : 3.5333333333333337

('Nicholas', 'Cindy') : 4.783333333333333

('Nicholas', 'Akshat') : 5.666666666666666

('Wusheng', 'Ahmed') : 5.033333333333333

('Wusheng', 'Priyanka') : 7.0

('Wusheng', 'Alfredo') : 6.85

('Wusheng', 'Layla') : 4.533333333333333

('Komal', 'Cindy') : 6.116666666666666

('Komal', 'Alfredo') : 3.6500000000000004

('Layla', 'Cindy') : 3.0

('Cindy', 'Priyanka') : 4.333333333333333

('Akshat', 'Priyanka') : 4.333333333333333  
Now lets remove the highest value edge and recalculate the edge betweenness. (Priyanka, 'Wusheng')

Edge Betweenness:

('Tushar', 'Nicholas') : 6.6499999999999995

('Tushar', 'Wusheng') : 4.283333333333333

('Tushar', 'Komal') : 4.533333333333333

('Nicholas', 'Ahmed') : 6.533333333333333

('Nicholas', 'Layla') : 4.033333333333333

('Nicholas', 'Cindy') : 5.783333333333334

('Nicholas', 'Akshat') : 8.333333333333334

('Wusheng', 'Ahmed') : 4.533333333333333

('Wusheng', 'Alfredo') : 5.35

('Wusheng', 'Layla') : 6.033333333333333

('Komal', 'Cindy') : 7.783333333333332

('Komal', 'Alfredo') : 5.15

('Layla', 'Cindy') : 5.0

('Cindy', 'Priyanka') : 7.666666666666666

('Akshat', 'Priyanka') : 3.333333333333333

Lets remove Komal, Cindy edge

Edge Betweenness:

('Tushar', 'Nicholas') : 11.333333333333334

('Tushar', 'Wusheng') : 4.833333333333333

('Tushar', 'Komal') : 7.833333333333333

('Nicholas', 'Ahmed') : 6.5

('Nicholas', 'Layla') : 4.5

('Nicholas', 'Cindy') : 6.5

('Nicholas', 'Akshat') : 9.5

('Wusheng', 'Ahmed') : 4.833333333333333

('Wusheng', 'Alfredo') : 8.166666666666666

('Wusheng', 'Layla') : 8.833333333333334

('Komal', 'Alfredo') : 3.1666666666666665

('Layla', 'Cindy') : 6.0

('Cindy', 'Priyanka') : 6.5

('Akshat', 'Priyanka') : 3.5

Lets remove Wusheng, Layla edge

Edge Betweenness:

('Tushar', 'Nicholas') : 16.666666666666668

('Tushar', 'Wusheng') : 7.000000000000002

('Tushar', 'Komal') : 9.333333333333334

('Nicholas', 'Ahmed') : 10.0

('Nicholas', 'Layla') : 7.0

('Nicholas', 'Cindy') : 9.5

('Nicholas', 'Akshat') : 10.5

('Wusheng', 'Ahmed') : 7.333333333333334

('Wusheng', 'Alfredo') : 6.666666666666668

('Komal', 'Alfredo') : 4.0

('Layla', 'Cindy') : 2.0

('Cindy', 'Priyanka') : 5.5

('Akshat', 'Priyanka') : 4.5

Lets remove ('Nicholas', Tushar) edge

Edge Betweenness:

('Tushar', 'Wusheng') : 11.0

('Tushar', 'Komal') : 5.0

('Nicholas', 'Ahmed') : 25.0

('Nicholas', 'Layla') : 7.0

('Nicholas', 'Cindy') : 9.5

('Nicholas', 'Akshat') : 10.5

('Wusheng', 'Ahmed') : 24.0

('Wusheng', 'Alfredo') : 11.0

('Komal', 'Alfredo') : 5.0

('Layla', 'Cindy') : 2.0

('Cindy', 'Priyanka') : 5.5

('Akshat', 'Priyanka') : 4.5

lets remove Nicholas, Ahmed edge

Edge Betweenness:

('Tushar', 'Wusheng') : 3.5

('Tushar', 'Komal') : 2.5

('Nicholas', 'Layla') : 2.0

('Nicholas', 'Cindy') : 2.0

('Nicholas', 'Akshat') : 3.0

('Wusheng', 'Ahmed') : 4.0

('Wusheng', 'Alfredo') : 3.5

('Komal', 'Alfredo') : 2.5

('Layla', 'Cindy') : 2.0

('Cindy', 'Priyanka') : 3.0

('Akshat', 'Priyanka') : 2.0

Lets remove Wusheng, Ahmed edge

Edge Betweenness:

('Tushar', 'Wusheng') : 2.0

('Tushar', 'Komal') : 2.0

('Nicholas', 'Layla') : 2.0

('Nicholas', 'Cindy') : 2.0

('Nicholas', 'Akshat') : 3.0

('Wusheng', 'Alfredo') : 2.0

('Komal', 'Alfredo') : 2.0

('Layla', 'Cindy') : 2.0

('Cindy', 'Priyanka') : 3.0

('Akshat', 'Priyanka') : 2.0



**Question 4:**

A whiteboard with blue text and math equations

Description automatically generated

**Question 5:**

1. The minimum value of NMI is 0 and maximum value of NMI is 1.
2. Entropy is a measure of uncertainty or disorder within a dataset, used particularly in community detection to assess the ambiguity in how nodes are grouped. Mutual information measures the amount of information shared between two variables. Normalized Mutual Information (NMI) is derived by using entropy and mutual information. It standardizes mutual information by dividing it by the average entropy of the two datasets involved, allowing NMI to be compared across different datasets and cluster sizes. Essentially, NMI measures how much the uncertainty about one variable is reduced by knowing the other, making it an effective metric for evaluating the quality of clustering.

**Question 6:**

A white board with blue writing on it

Description automatically generated

A white board with blue writing on it

Description automatically generated

**Question 7:**

High precision alone isn't always sufficient because it doesn't consider all the actual positive cases in the dataset; it only tells us about the accuracy of the positive predictions we made. Precision is the proportion of true positive predictions in the total predicted positives. However, this metric does not account for the positive cases that the model failed to predict, which is where recall comes into play. Recall (or sensitivity) measures the proportion of actual positives that were correctly identified by the model.

To understand why both precision and recall are important, let's consider a real-world example in the context of a medical screening for a disease:

Example: Medical Screening for a Disease

Imagine a scenario where a new test is developed to screen for a particular disease. The goal is for the test to correctly identify all individuals who have the disease (true positives) while minimizing the number of people who don't have the disease but test positive (false positives).

High Precision Scenario:

Suppose the test is designed to have very high precision. This means that almost everyone who tests positive really does have the disease.

Precision: High (e.g., 95%)

However, if this precision is achieved by setting very stringent criteria for testing positive, the test might miss many people who actually have the disease but don't meet these strict criteria.

Recall: Low (e.g., 30%)

Outcome: Many actual cases of the disease go undetected, which could lead to a failure in controlling the disease's spread or in timely treatment for many patients.

In this case, while the precision of the test is excellent, its low recall means it's not very useful for a screening program because it misses too many cases of the disease. This could be detrimental to public health efforts.

High Recall Scenario:

On the other hand, if a test is designed to have high recall, it identifies most or all of the people who have the disease.

Recall: High (e.g., 95%)

However, if achieving this involves lowering the threshold for a positive result, the test may end up with many false positives—identifying many people as having the disease when they actually do not.

Precision: Low (e.g., 30%)

Outcome: While most cases are detected, the high number of false positives could lead to unnecessary anxiety, treatment, and healthcare costs.

Conclusion

In many real-world applications, especially in critical fields like healthcare, both high precision and high recall are important to ensure the test or model is both accurate and useful. In practice, a balance often needs to be struck between these two measures, which can be quantified using the **F-score**—a harmonic mean of precision and recall. This balance ensures that a model or test is not only accurate when it predicts a positive but also effective in identifying most of the actual positive cases.

**Question 8:**

Purity is a metric often used in clustering to evaluate the homogeneity of the clusters produced by a clustering algorithm. It calculates the proportion of the dominant class in each cluster compared to the size of the cluster. Specifically, for each cluster, the number of most frequent class members is divided by the size of the cluster, and these values are then averaged across all clusters.

However, purity can indeed be misleading or manipulated in certain scenarios, particularly when the clusters are not representative of meaningful groupings in the data. Here are two cases where purity can generate misleadingly high values:

1. Singleton Communities (Clusters of Size 1)

In the extreme case where each node in a dataset forms its own cluster (singleton clusters), purity becomes meaningless but technically perfect. Here's why:

Singleton Clusters: Each cluster contains exactly one data point.

Purity Calculation: Since each cluster contains only one point, the most frequent (and only) label in each cluster matches 100% of the cluster.

Overall Purity: The purity of each cluster is 1 (or 100%), making the average purity across all clusters also 1.

Implication: High purity in this scenario indicates perfect homogeneity but reveals nothing about the actual clustering quality or the underlying structure of the data. This scenario doesn't capture any meaningful relationships or groupings except the trivial fact that every point is identical to itself.

2. Very Large Pure Communities

Another scenario where purity can be misleading is when a clustering algorithm groups most of the data into a few very large clusters that predominantly consist of the majority class:

Large Pure Clusters: If a cluster is large and mostly contains members of the majority class, its purity will be high.

Overall Purity: The large size of these pure clusters heavily influences the overall purity metric, because these clusters dominate the average calculation.

Implication: Even if the clustering does not meaningfully separate different classes within the data, the purity measure will still indicate high values, suggesting effective clustering. This can be misleading if the goal of clustering was to discover meaningful subgroups or patterns in the data.

Conclusion

Both of these cases illustrate why purity, while a simple and often useful measure of cluster homogeneity, can fail to provide a complete or accurate assessment of clustering quality, especially when the cluster sizes are highly imbalanced or artificially manipulated. This can lead to high purity scores without meaningful insight into the data structure.

**Question 9:**

A white paper with blue writing on it

Description automatically generated

A white board with blue writing on it

Description automatically generated

A white board with blue writing on it

Description automatically generated

A white board with blue writing on it

Description automatically generated