GAM Assignment1 - Summary Result - Vipin Indoria

January 1, 2023

1 GAM Assignment 1

S.No.	Name	BITS ID
1.	Vipin Indoria	2021FC04245

1.0.1 A. Generate random graphs of at least 100 node connected graphs.

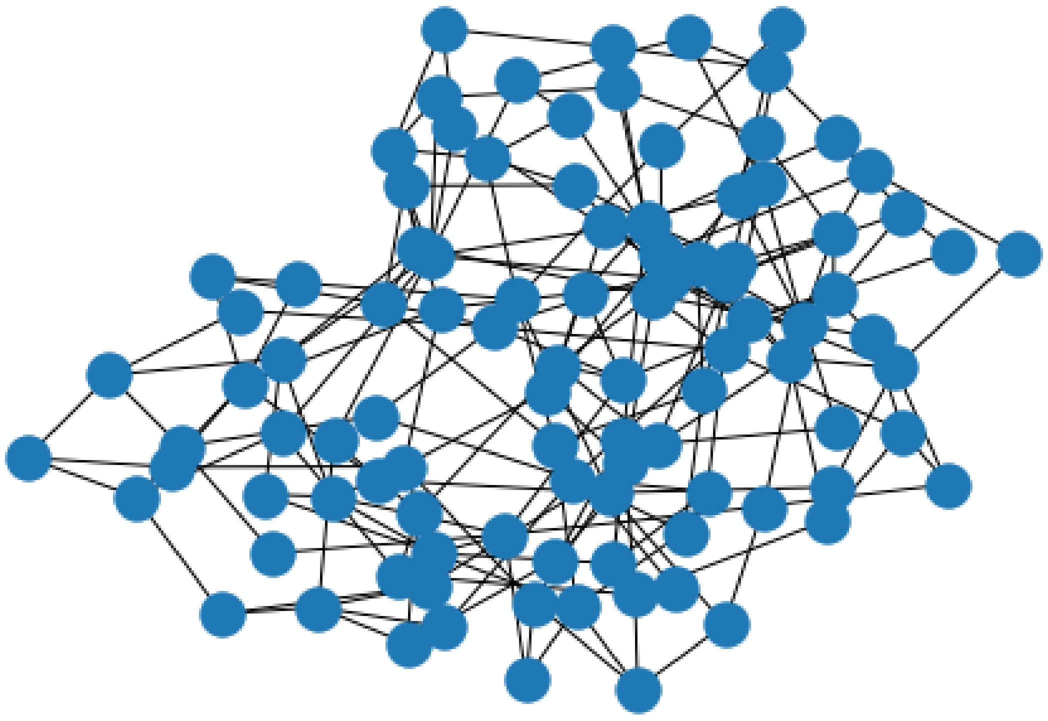
To generate a random connected graph of at least 100 nodes, we can use a graph generation algorithm such as the Watts-Strogatz model. The Watts-Strogatz model is a model for generating random graphs with small-world properties, developed by Duncan Watts and Steven Strogatz in 1998. It is a type of random graph that has both short average path lengths (like a regular lattice) and high clustering (like a random graph).

To generate a graph using the Watts-Strogatz model, we can use the following steps:

1. Start with a regular lattice of n nodes, where each node is connected to its k nearest neighbors. This forms a regular ring lattice.
2. Randomly rewire each edge with probability p. This means that for each edge, you flip a coin with probability p of rewiring the edge to a randomly chosen node.
3. Calculate the average path length and the clustering coefficient for the resulting graph. The average path length is the average number of edges between two nodes in the graph, while the clustering coefficient is a measure of the degree to which nodes in the graph tend to cluster together.

To use the Watts-Strogatz model in Python, we can use the NetworkX library. Here is an example of how to generate a Watts-Strogatz graph with 100 nodes, each connected to its 4 nearest neighbors, with a rewiring probability of 0.5:

[2]:
```
nx.draw(G)
```



1.0.2 B. Perform the network measurements

Once we have generated a random connected graph, we can perform the following network measurements:

1. Eccentricity: The eccentricity of a node is the maximum distance between that node and any other node in the graph. We can calculate the eccentricity of a node using the eccentricity() function in the networkx library.

The eccentricity function returns a dictionary where the keys are the nodes and the values are the eccentricities. We can also use the eccentricity function to compute the eccentricities of all nodes in the graph by passing it the entire graph as the first argument.

[3]:
```
# prints a dictionary with the eccentricities of all nodes
```

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

Diameter: We can also use the diameter function to find the diameter of the graph, which is the maximum eccentricity of any node in the graph.

[4]:
```
# prints the maximum eccentricity of any node in the graph
```

```
6
```

2. Radius: To find the radius of a graph, we can use the radius function in networkx. The radius of a graph is the minimum eccentricity of any node in the graph.

Here is an example of how to use the radius function to find the radius of a graph.

[5]:
```
# prints the minimum eccentricity of any node in the graph
```

```
5
```

3. Centrality: There are several centrality measures that we can use to identify the most important nodes in a graph. Here are a few examples of how to compute these centralities using networkx:

- **Degree centrality:** This measure assigns a higher score to nodes with more connections. To compute degree centrality in networkx, we can use the degree_centrality function:

[6]:
```
# prints a dictionary with the degree centralities of all nodes
```

```
{0: 0.030303030303030304, 1: 0.030303030303030304, 2: 0.030303030303030304, 3:
0.030303030303030304, 4: 0.030303030303030304, 5: 0.05050505050505051, 6:
0.030303030303030304, 7: 0.06060606060606061, 8: 0.08080808080808081, 9:
0.05050505050505051, 10: 0.07070707070707072, 11: 0.030303030303030304, 12:
0.04040404040404041, 13: 0.05050505050505051, 14: 0.04040404040404041, 15:
0.030303030303030304, 16: 0.04040404040404041, 17: 0.05050505050505051, 18:
0.04040404040404041, 19: 0.04040404040404041, 20: 0.030303030303030304, 21:
0.030303030303030304, 22: 0.05050505050505051, 23: 0.04040404040404041, 24:
0.04040404040404041, 25: 0.020202020202020204, 26: 0.05050505050505051, 27:
0.030303030303030304, 28: 0.030303030303030304, 29: 0.030303030303030304, 30:
0.09090909090909091, 31: 0.04040404040404041, 32: 0.05050505050505051, 33:
0.030303030303030304, 34: 0.05050505050505051, 35: 0.06060606060606061, 36:
0.06060606060606061, 37: 0.04040404040404041, 38: 0.030303030303030304, 39:
0.04040404040404041, 40: 0.020202020202020204, 41: 0.030303030303030304, 42:
0.05050505050505051, 43: 0.030303030303030304, 44: 0.05050505050505051, 45:
0.04040404040404041, 46: 0.020202020202020204, 47: 0.05050505050505051, 48:
0.04040404040404041, 49: 0.020202020202020204, 50: 0.030303030303030304, 51:
0.04040404040404041, 52: 0.030303030303030304, 53: 0.05050505050505051, 54:
0.05050505050505051, 55: 0.06060606060606061, 56: 0.020202020202020204, 57:
0.04040404040404041, 58: 0.05050505050505051, 59: 0.04040404040404041, 60:
0.05050505050505051, 61: 0.020202020202020204, 62: 0.030303030303030304, 63:
0.030303030303030304, 64: 0.030303030303030304, 65: 0.05050505050505051, 66:
0.020202020202020204, 67: 0.05050505050505051, 68: 0.04040404040404041, 69:
0.05050505050505051, 70: 0.04040404040404041, 71: 0.06060606060606061, 72:
```

```
0.06060606060606061, 73: 0.06060606060606061, 74: 0.05050505050505051, 75:
0.020202020202020204, 76: 0.05050505050505051, 77: 0.030303030303030304, 78:
0.030303030303030304, 79: 0.030303030303030304, 80: 0.04040404040404041, 81:
0.04040404040404041, 82: 0.04040404040404041, 83: 0.020202020202020204, 84:
0.04040404040404041, 85: 0.05050505050505051, 86: 0.030303030303030304, 87:
0.030303030303030304, 88: 0.030303030303030304, 89: 0.030303030303030304, 90:
0.04040404040404041, 91: 0.06060606060606061, 92: 0.030303030303030304, 93:
0.04040404040404041, 94: 0.05050505050505051, 95: 0.04040404040404041, 96:
0.04040404040404041, 97: 0.04040404040404041, 98: 0.030303030303030304, 99:
0.030303030303030304}
```

- **Eigenvector centrality:** This measure assigns a higher score to nodes that are connected to other high-score nodes. To compute eigenvector centrality in networkx, we can use the eigenvector_centrality function:

[7]:
```
# prints a dictionary with the eigenvector centralities of all nodes
```

```
{0: 0.08577375002568856, 1: 0.050337886052868264, 2: 0.08416740712846865, 3:
0.06343674094103154, 4: 0.07826201211434992, 5: 0.15399208640027665, 6:
0.12991269216405424, 7: 0.18263235153825202, 8: 0.24857419586782128, 9:
0.1599280460526879, 10: 0.18526424866956262, 11: 0.08076724878347696, 12:
0.08783231357027671, 13: 0.09071930303180543, 14: 0.06617117271172016, 15:
0.06184572083885784, 16: 0.05766430705370883, 17: 0.08903222961794936, 18:
0.07531763672615167, 19: 0.08902771999223213, 20: 0.07332763709073575, 21:
0.09026369706544712, 22: 0.13392818961698766, 23: 0.08469825864739032, 24:
0.09403973665654655, 25: 0.03306842722821334, 26: 0.06718075246335013, 27:
0.05137958394275498, 28: 0.07694742537215846, 29: 0.10611761580976674, 30:
0.244757810032737, 31: 0.12499463267756813, 32: 0.12850864862596365, 33:
0.08118397728241204, 34: 0.1127425590265326, 35: 0.12443203266804506, 36:
0.12337222486681901, 37: 0.06854601429049666, 38: 0.05294454164479498, 39:
0.05073634804273033, 40: 0.03195874792914591, 41: 0.03735920624285845, 42:
0.08177481179760644, 43: 0.05974493477408361, 44: 0.08737234433800643, 45:
0.08944158974074601, 46: 0.04767134348502123, 47: 0.10188853086470015, 48:
0.11285105375642458, 49: 0.07428698587884276, 50: 0.07473112298043007, 51:
0.06805528075383953, 52: 0.0743701419528417, 53: 0.10451113435238196, 54:
0.12726143961542832, 55: 0.11805505853417216, 56: 0.057575885851803385, 57:
0.0959772564625317, 58: 0.09508549626033114, 59: 0.08391808638513501, 60:
0.12229693322046091, 61: 0.03415879562131055, 62: 0.030491765797987833, 63:
0.05245244558018007, 64: 0.0663178254079795, 65: 0.087723750783657, 66:
0.04665292070473868, 67: 0.12409352278864355, 68: 0.0860577734040184, 69:
0.13968652234789064, 70: 0.1270274848585023, 71: 0.1907123797783331, 72:
0.20139528440762516, 73: 0.16426851828321506, 74: 0.1479559900979569, 75:
0.043618192870224115, 76: 0.13016332320759919, 77: 0.056117375648768425, 78:
0.10806667611883998, 79: 0.10805759660133522, 80: 0.08310399387881177, 81:
0.09230645994796725, 82: 0.07348040294770085, 83: 0.037472070379305035, 84:
0.04207071125574839, 85: 0.06984876957047723, 86: 0.041364775978117316, 87:
0.04454836625069913, 88: 0.03206612776267733, 89: 0.04322727090830694, 90:
0.05047604169726218, 91: 0.07192776412252555, 92: 0.04642766563742202, 93:
```

```
0.07978157546801752, 94: 0.07887864279113771, 95: 0.06980756602725423, 96:
0.06806835810282091, 97: 0.08731367725973489, 98: 0.07061929481704071, 99:
0.09122966338572443}
```

- **Katz centrality:** This measure assigns a score to each node based on the number and strength of its connections. To compute Katz centrality in networkx, we can use the katz_centrality function:

```
[8]: # prints a dictionary with the Katz centralities of all nodes
```

```
{0: 0.09106613734532175, 1: 0.08939267099809639, 2: 0.0903157607516702, 3:
0.08937610012174227, 4: 0.08953101141643334, 5: 0.11426886405323074, 6:
0.09574074273831702, 7: 0.12300108862712637, 8: 0.14347110851799497, 9:
0.11422827904594643, 10: 0.1303158337581639, 11: 0.09129051168479639, 12:
0.09846364538674579, 13: 0.10745920528091764, 14: 0.09698758572519987, 15:
0.09015747875332404, 16: 0.0971290765121952, 17: 0.10911941312200239, 18:
0.09906702318130364, 19: 0.09970580666761299, 20: 0.08705399325420218, 21:
0.09036076466426872, 22: 0.11064533968632334, 23: 0.09641612422209626, 24:
0.10097696791207758, 25: 0.07864214321913607, 26: 0.10202475240792691, 27:
0.08672396479298719, 28: 0.09094503984715582, 29: 0.09286738472655505, 30:
0.15072517608929953, 31: 0.10394131403200406, 32: 0.11236955835428236, 33:
0.09204518482207531, 34: 0.11097190835783798, 35: 0.12044395133061964, 36:
0.1188013317873593, 37: 0.09759023303439147, 38: 0.08745832423617621, 39:
0.09369083109314207, 40: 0.07752047035609629, 41: 0.08649255079622593, 42:
0.10773018236140584, 43: 0.08820588679329922, 44: 0.1078431354211504, 45:
0.09959611690772723, 46: 0.07822018150770946, 47: 0.10499849696048531, 48:
0.10053683972607216, 49: 0.08150670542143397, 50: 0.09001975103189715, 51:
0.09532804171044164, 52: 0.08936758631769796, 53: 0.10976830548980784, 54:
0.11087616848959705, 55: 0.11767393371184515, 56: 0.08037772946777061, 57:
0.10088859517302244, 58: 0.10648367509646346, 59: 0.09895156187444583, 60:
0.11036221850118089, 61: 0.07791451045571063, 62: 0.0836772772845107, 63:
0.08850093054052133, 64: 0.08897065074392406, 65: 0.10699162363381765, 66:
0.0780665922434195, 67: 0.1090699394415424, 68: 0.09492950337652648, 69:
0.11080521139948747, 70: 0.10225009845529615, 71: 0.12374986782551843, 72:
0.1261782747767153, 73: 0.12062715494503345, 74: 0.11270584505755794, 75:
0.07739932589519144, 76: 0.10835612447456303, 77: 0.08623499134893901, 78:
0.09215124451325951, 79: 0.09301881565949698, 80: 0.0980120830673556, 81:
0.09795315462678417, 82: 0.09818694940729536, 83: 0.07880312561633636, 84:
0.09353970721650864, 85: 0.10532489517639218, 86: 0.08832988142963383, 87:
0.08855881195490484, 88: 0.08539187677390925, 89: 0.08822661864989745, 90:
0.09533481935483118, 91: 0.11274862138905162, 92: 0.08759618190372542, 93:
0.09884320568448039, 94: 0.10511731439568034, 95: 0.09842497118517204, 96:
0.09663112046092082, 97: 0.09760338272002149, 98: 0.08849908598761048, 99:
0.09301122921751744}
```

- **PageRank:** This measure assigns a score to each node based on the importance of the nodes it is connected to. To compute PageRank in networkx, we can use the pagerank function.

```
[9]: # prints a dictionary with the PageRanks of all nodes
```

```
{0: 0.00788588952964869, 1: 0.007897844574446772, 2: 0.007957299255812945, 3:
0.0076873560532421835, 4: 0.007880451029088902, 5: 0.011623083783699358, 6:
0.007388433332230383, 7: 0.01367426332136393, 8: 0.018593437229781604, 9:
0.01167805829757292, 10: 0.016431144648889208, 11: 0.007678107682015224, 12:
0.009977988513812711, 13: 0.012132424122999159, 14: 0.010017860460599267, 15:
0.007698566160134538, 16: 0.010136915988803468, 17: 0.011966253951804447, 18:
0.00973960458130239, 19: 0.009708603057311471, 20: 0.00797660086232403, 21:
0.007892709770157035, 22: 0.01210715451403319, 23: 0.010153599672573224, 24:
0.009783261813226685, 25: 0.005763489904724689, 26: 0.012837668632277019, 27:
0.008069094231243214, 28: 0.007960843251563415, 29: 0.007700191141629308, 30:
0.020702527958493838, 31: 0.009791848618820018, 32: 0.01199939579745891, 33:
0.0075484354063512825, 34: 0.011932215431640778, 35: 0.013977570454128936, 36:
0.014365478728944657, 37: 0.01016878871944415, 38: 0.008000382926075017, 39:
0.010807113683197063, 40: 0.005949940352973623, 41: 0.008086127701613335, 42:
0.012243609142700267, 43: 0.007746048576847625, 44: 0.01213892251952612, 45:
0.009790223203424283, 46: 0.0058189331976912315, 47: 0.012816876558732903, 48:
0.010074119650814018, 49: 0.005698266401184112, 50: 0.007809407452552404, 51:
0.01017317076193288, 52: 0.007672423424928966, 53: 0.011875626066713746, 54:
0.011718043171716907, 55: 0.014385927038776827, 56: 0.005571288017374341, 57:
0.009748269298403053, 58: 0.012204508813892631, 59: 0.009870450376685214, 60:
0.011806670412984251, 61: 0.005975474697104335, 62: 0.008612091881850772, 63:
0.008033322817821069, 64: 0.008125271916652823, 65: 0.012088840582289127, 66:
0.00577924288765116, 67: 0.012096843997737907, 68: 0.010462039689778038, 69:
0.011876767660504717, 70: 0.009715377906524039, 71: 0.013748499396421424, 72:
0.013875276012348212, 73: 0.014093690484554687, 74: 0.01168169349355781, 75:
0.005874328346691119, 76: 0.012192839772846667, 77: 0.008009445382713675, 78:
0.007730521605706908, 79: 0.007704081917040035, 80: 0.010025875623400798, 81:
0.0098922767771806, 82: 0.009822154571555152, 83: 0.005834484344652274, 84:
0.010691775229050352, 85: 0.012544720612379589, 86: 0.007975255469764566, 87:
0.007921137865439212, 88: 0.00828464124584063, 89: 0.008007681508132379, 90:
0.01044205040769036, 91: 0.014809170289340019, 92: 0.007911202039966422, 93:
0.009921379972293195, 94: 0.012257368878648102, 95: 0.009881650530827863, 96:
0.01004329440176836, 97: 0.010031594500040326, 98: 0.007865449659200774, 99:
0.007670376388699658}
```

4. Exploratory Analysis: To perform exploratory analysis of the centralities calculated in step 3, we can use various visualization and statistical techniques to examine the distribution, patterns, and relationships between the centralities and other properties of the graph. Here are a few examples of what you can do:

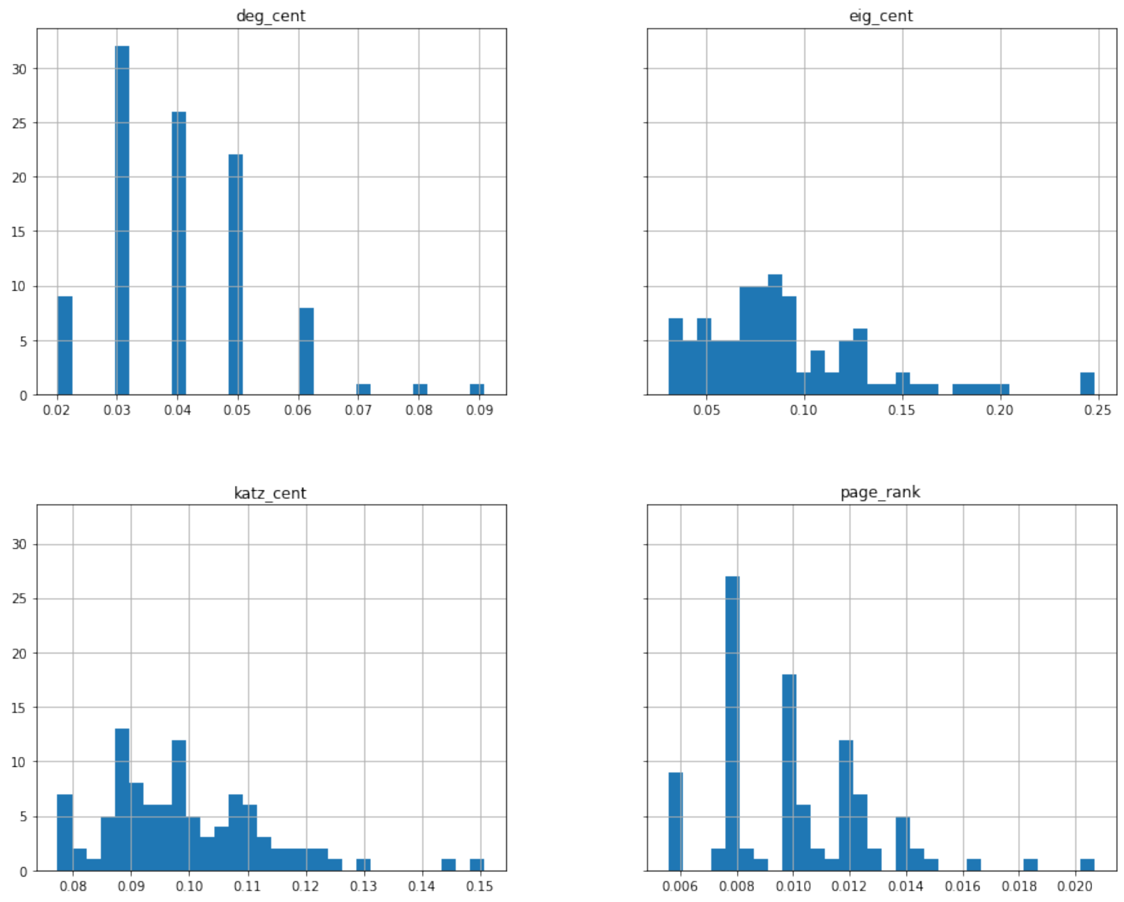
- Plot the centrality scores as a histogram or box plot to visualize their distribution and identify any outliers or patterns. We can also use a scatterplot to visualize the relationship between different centrality measures.

[14]:
```
# Create a pandas dataframe from centrality scores where dataframe index is␣
 ↪node id
```

```
[14]:    deg_cent  eig_cent  katz_cent  page_rank
      0  0.030303  0.085774   0.091066   0.007886
      1  0.030303  0.050338   0.089393   0.007898
      2  0.030303  0.084167   0.090316   0.007957
      3  0.030303  0.063437   0.089376   0.007687
      4  0.030303  0.078262   0.089531   0.007880
```

```
[27]: # show summary statistics
      # build histograms for each numeric feature
```

```
         deg_cent    eig_cent   katz_cent   page_rank
count  100.000000  100.000000  100.000000  100.000000
mean     0.040404    0.090091    0.099053    0.010000
std      0.013314    0.043620    0.013801    0.002787
min      0.020202    0.030492    0.077399    0.005571
25%      0.030303    0.061321    0.089374    0.007897
50%      0.040404    0.082439    0.097360    0.009876
75%      0.050505    0.109236    0.107758    0.011941
max      0.090909    0.248574    0.150725    0.020703
```

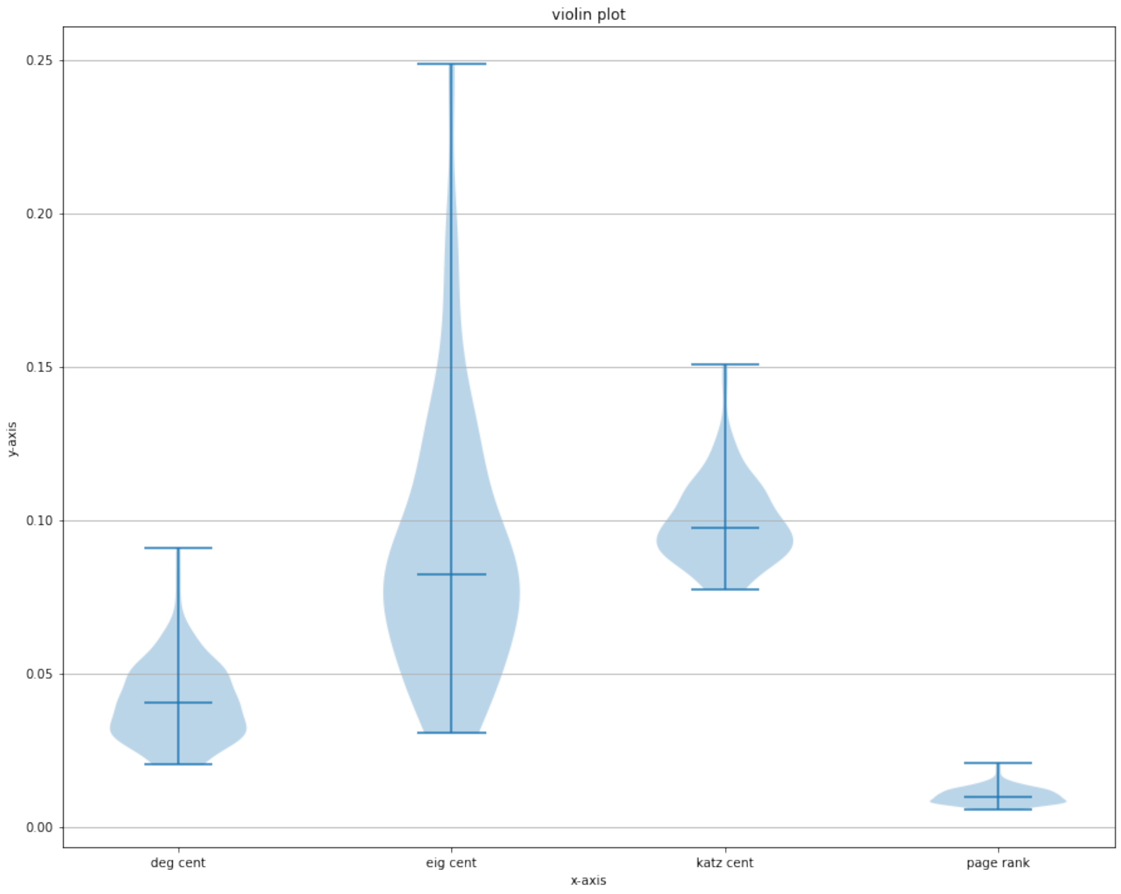


Here's a more detailed interpretation of these statistics:

- deg_cent: This variable has 100 observations, with a mean of 0.04 and a standard deviation of 0.01. The minimum value is 0.02, while the maximum value is 0.09. The 25th percentile is 0.03, the 50th percentile is 0.04, and the 75th percentile is 0.05.
- eig_cent: This variable also has 100 observations, with a mean of 0.09 and a standard deviation of 0.04. The minimum value is 0.03, while the maximum value is 0.25. The 25th percentile is 0.06, the 50th percentile is 0.08, and the 75th percentile is 0.11.
- katz_cent: This variable has 100 observations, with a mean of 0.10 and a standard deviation of 0.01. The minimum value is 0.08, while the maximum value is 0.15. The 25th percentile is 0.09, the 50th percentile is 0.10, and the 75th percentile is 0.11.
- page_rank: This variable has 100 observations, with a mean of 0.01 and a standard deviation of 0.003. The minimum value is 0.006, while the maximum value is 0.02. The 25th percentile is 0.008, the 50th percentile is 0.01, and the 75th percentile is 0.012.

Overall, these summary statistics suggest that the variables except `eigen centrality` have relatively small ranges and are relatively concentrated around their means. The `eigen centrality` variable appears to be slightly more spread out than the other variables, as indicated by its larger standard deviation.

[24]:
```
# Show distribution using violin plot
```

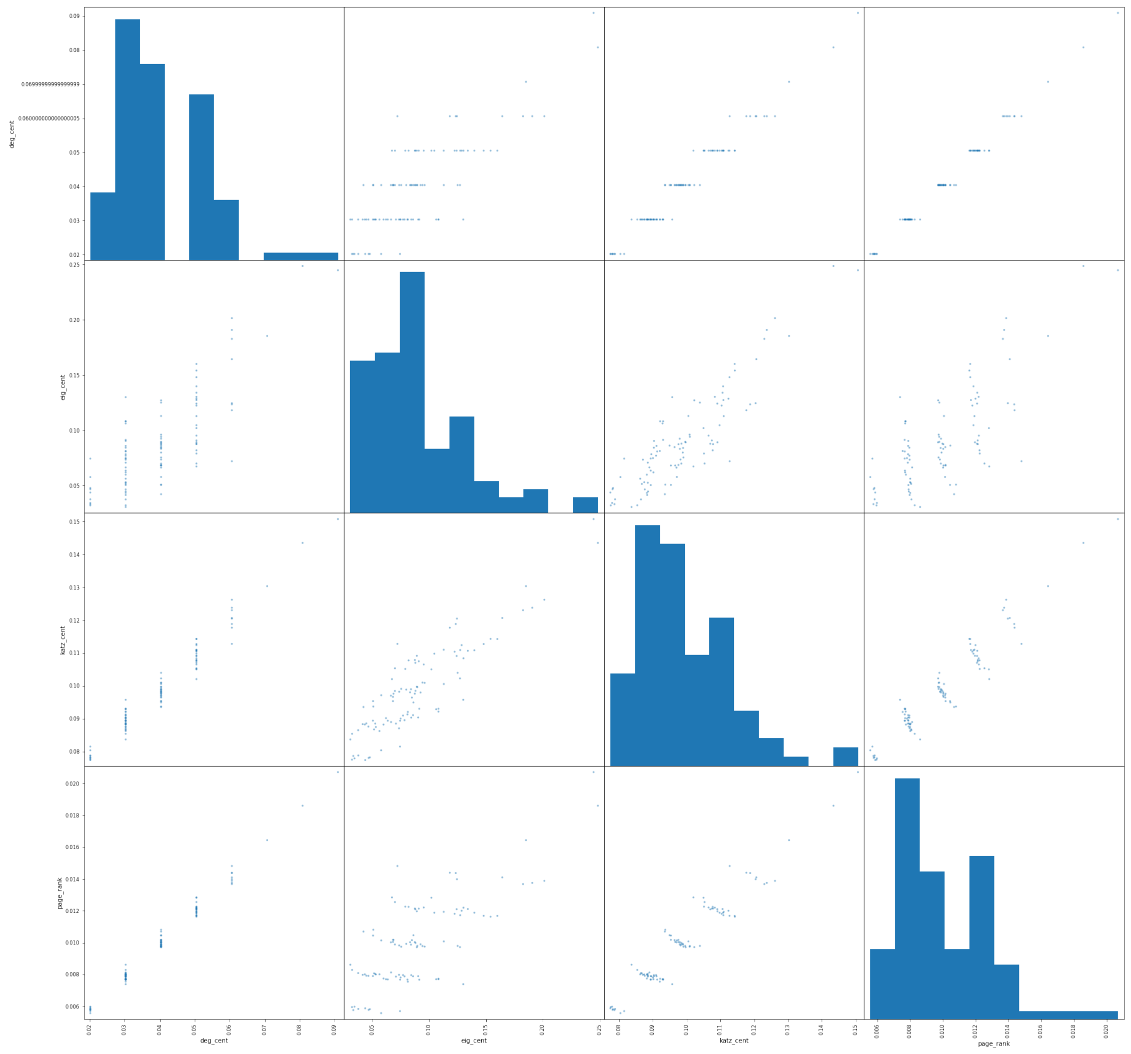


The thickness of the violin plot represents the probability density of the data at different values. From above plot we can interpret that for degree centrality, katz centrality and page rank violin plot height is less and is more thick, it means that the centrality is more concentrated and has a higher probability density at the values represented by the thicker part of the plot. This can be interpreted as the centrality being more clustered around a central value and having less spread.

Whereas for eigen centrality violin plot is taller and thinner, it indicate that this is a more uniform distribution of centrality, with fewer nodes having extremely high or low centrality.

[28]:
```
# Display correlation between centralities
```

```
           deg_cent  eig_cent  katz_cent  page_rank
deg_cent   1.000000  0.798526   0.980004   0.995206
eig_cent   0.798526  1.000000   0.889128   0.755628
katz_cent  0.980004  0.889128   1.000000   0.959108
page_rank  0.995206  0.755628   0.959108   1.000000
```



Here's a brief interpretation of the correlations in the matrix:

- `deg_cent` and `eig_cent`: These variables have a correlation of 0.80, which indicates a strong positive relationship. This means that as one variable increases, the other tends to increase as well.
- `deg_cent` and `katz_cent`: These variables have a correlation of 0.98, which indicates a very strong positive relationship. This means that as one variable increases, the other tends to increase significantly.
- `deg_cent` and `page_rank`: These variables have a correlation of 0.995, which indicates a nearly perfect positive relationship. This means that as one variable increases, the other increases almost perfectly in sync.
- `eig_cent` and `katz_cent`: These variables have a correlation of 0.89, which indicates a strong positive relationship. This means that as one variable increases, the other tends to increase as well.

- `eig_cent` and `page_rank`: These variables have a correlation of 0.76, which indicates a moderate positive relationship. This means that as one variable increases, the other tends to increase somewhat.
- `katz_cent` and `page_rank`: These variables have a correlation of 0.96, which indicates a very strong positive relationship. This means that as one variable increases, the other tends to increase significantly.

Overall, these correlations suggest that there are strong positive relationships between the variables. This means that as one variable increases, the other variables tend to increase as well. However, the strength of the relationships varies, with some being stronger than others.

5. Betweenness and Closeness Centrality: Betweenness centrality and closeness centrality are two measures of centrality that capture different aspects of a node's importance in a graph. Betweenness centrality measures the degree to which a node lies on paths between other nodes, while closeness centrality measures the distance of a node to all other nodes in the graph.

To calculate betweenness centrality and closeness centrality for a graph using networkx, we can use the betweenness_centrality and closeness_centrality functions, respectively.

[29]:
```
# prints a dictionary with the betweenness centralities of all nodes
```

```
{0: 0.020903476376574703, 1: 0.016091102145523915, 2: 0.016944059971270853, 3:
0.01456398968767553, 4: 0.00991924427298577, 5: 0.0532451909848323, 6:
0.010970164881729508, 7: 0.051113898470113675, 8: 0.10254562371136272, 9:
0.04583720123299591, 10: 0.08181605208506877, 11: 0.009935497860667929, 12:
0.019817480098865376, 13: 0.036403422227231527, 14: 0.014907881234411847, 15:
0.007451166758526069, 16: 0.025662758300358 79, 17: 0.033555399912851995, 18:
0.021649733028830116, 19: 0.022577003266551803, 20: 0.004520259282164044, 21:
0.011166000791851129, 22: 0.031044897062213066, 23: 0.013238556605903548, 24:
0.04043050904522766, 25: 0.007041763751720461, 26: 0.03929636066308917, 27:
0.015442497446208018, 28: 0.023823675694423993, 29: 0.010446381534816909, 30:
0.1315895624337182 2, 31: 0.04157662637872903, 32: 0.063383213707889, 33:
0.0035693835013562904, 34: 0.027550871675794363, 35: 0.038237266638627185, 36:
0.05965171306853435, 37: 0.0344153464437942, 38: 0.014631640605666575, 39:
0.026613084343449216, 40: 0.007516450543661425, 41: 0.01477251018067344, 42:
0.04386106403731635, 43: 0.007361463978656312, 44: 0.040296320985869515, 45:
0.025143742397916793, 46: 0.0032338284038964304, 47: 0.04292110750057564, 48:
0.022296253758838796, 49: 0.005517762660619803, 50: 0.01845806624718189, 51:
0.018247329641887466, 52: 0.007023588656241716, 53: 0.03504879474267228, 54:
0.03205172882043629, 55: 0.06078855236245466, 56: 0.007170261081825707, 57:
0.026813706575611333, 58: 0.035052696107117 86, 59: 0.01488621607669227, 60:
0.02724751942439017, 61: 0.005100801189236563, 62: 0.009081355324397992, 63:
0.016716625297330315, 64: 0.020418730700115982, 65: 0.046427118427736856, 66:
0.0021534587861118466, 67: 0.04597249792054987, 68: 0.012826106873725922, 69:
0.03235554671856466, 70: 0.02796247592165959, 71: 0.03700072402917177, 72:
0.058403016736350066, 73: 0.06262201860532103, 74: 0.03424832743014561, 75:
0.0032538700906047835, 76: 0.028696096043034813, 77: 0.009516038257534857, 78:
0.011852489913714404, 79: 0.005857489701027116, 80: 0.0228380560322428, 81:
```

```
0.024029744391525464, 82: 0.0210035629639835, 83: 0.007325686407319061, 84:
0.024146060540618356, 85: 0.043210203754421445, 86: 0.0069485829226088956, 87:
0.012047916809821571, 88: 0.01153329022840464, 89: 0.011638238076085944, 90:
0.030224818814800266, 91: 0.04397178498910101, 92: 0.012978333881240497, 93:
0.021277972468448658, 94: 0.014393249504566758, 95: 0.01762115378380066, 96:
0.007570209858397859, 97: 0.020230883434346633, 98: 0.010313287802465291, 99:
0.01929328487770046}
```

[30]:
```python
# prints a dictionary with the closeness centralities of all nodes
```

```
{0: 0.2911764705882353, 1: 0.2668463611859838, 2: 0.28530259365994237, 3:
0.2742382271468144, 4: 0.2796610169491525, 5: 0.3256578947368421, 6:
0.3065015479876161, 7: 0.32781456953642385, 8: 0.3473684210526316, 9:
0.3235294117647059, 10: 0.32781456953642385, 11: 0.2757660167130919, 12:
0.2804532577903683, 13: 0.2861271676300578, 14: 0.26975476839237056, 15:
0.26756756756756755, 16: 0.276536312849162, 17: 0.28448275862068967, 18:
0.28205128205128205, 19: 0.2911764705882353, 20: 0.2591623036649215, 21:
0.2757660167130919, 22: 0.3009118541033435, 23: 0.2661290322580645, 24:
0.29819277108433734, 25: 0.2584856396866841, 26: 0.2861271676300578, 27:
0.27049180327868855, 28: 0.28695652173913044, 29: 0.2894736842105263, 30:
0.35106382978723405, 31: 0.31629392971246006, 32: 0.3183279742765273, 33:
0.27197802197802196, 34: 0.2964071856287425, 35: 0.2920353982300885, 36:
0.3046153846153846, 37: 0.28530259365994237, 38: 0.2727272727272727, 39:
0.26902173913043476, 40: 0.2551546391752577, 41: 0.264, 42: 0.2955223880597015,
43: 0.27123287671232876, 44: 0.2955223880597015, 45: 0.29464285714285715, 46:
0.25384615384615383, 47: 0.3009118541033435, 48: 0.2877906976744186, 49:
0.26902173913043476, 50: 0.28695652173913044, 51: 0.2742382271468144, 52:
0.26902173913043476, 53: 0.29464285714285715, 54: 0.30368098159509205, 55:
0.3113207547169811, 56: 0.2647058823529412, 57: 0.2886297376093295, 58:
0.2920353982300885, 59: 0.2727272727272727, 60: 0.2911764705882353, 61:
0.24324324324324326, 62: 0.24504950495049505, 63: 0.26975476839237056, 64:
0.27887323943661974, 65: 0.2955223880597015, 66: 0.25063291139240507, 67:
0.3103448275862069, 68: 0.26975476839237056, 69: 0.30275229357798167, 70:
0.2972972972972973, 71: 0.31528662420382164, 72: 0.32781456953642385, 73:
0.31528662420382164, 74: 0.3046153846153846, 75: 0.2512690355329949, 76:
0.29376854599406527, 77: 0.2591623036649215, 78: 0.2836676217765043, 79:
0.28125, 80: 0.28695652173913044, 81: 0.29289940828402367, 82:
0.28285714285714286, 83: 0.2584856396866841, 84: 0.26756756756756755, 85:
0.2886297376093295, 86: 0.2578125, 87: 0.2619047619047619, 88:
0.24504950495049505, 89: 0.25, 90: 0.2682926829268293, 91: 0.2804532577903683,
92: 0.2456575682382134, 93: 0.27123287671232876, 94: 0.2591623036649215, 95:
0.25984251968503935, 96: 0.25190839694656486, 97: 0.28448275862068967, 98:
0.2682926829268293, 99: 0.2836676217765043}
```

The physical significance of betweenness centrality and closeness centrality depends on the context in which the graph is used. Here are a few examples of how these measures can be interpreted in different contexts:

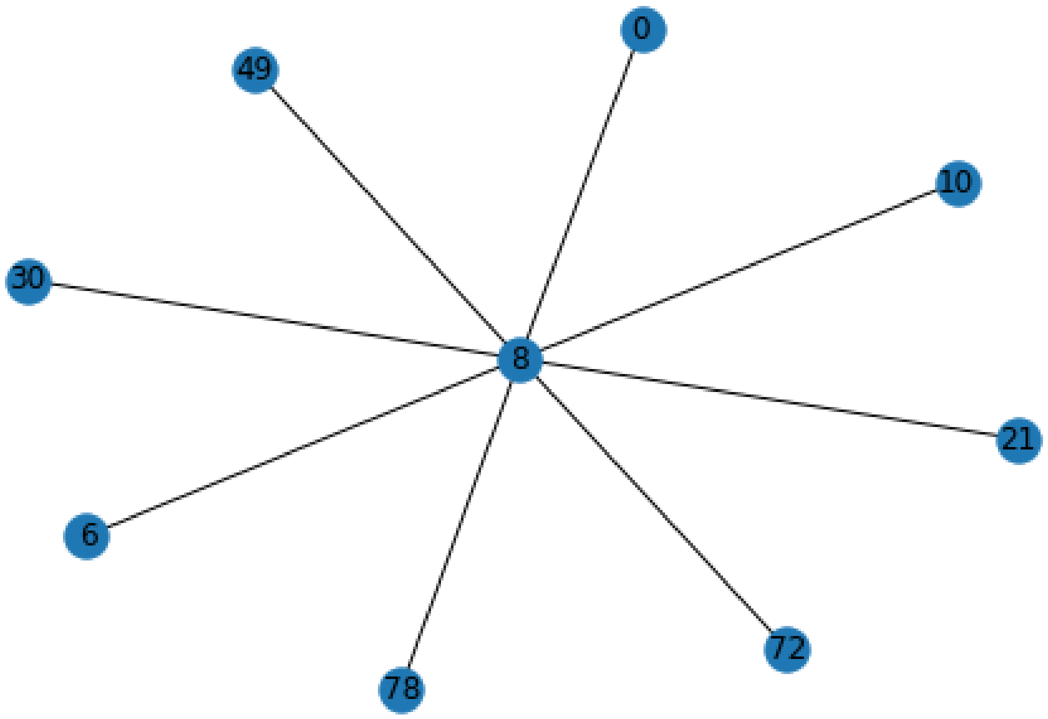
- In a social network, betweenness centrality can be interpreted as the degree to which a person serves as a “bridge” or “gatekeeper” between different groups or communities. A person with high betweenness centrality may be well-connected and able to facilitate communication and information flow between different groups.

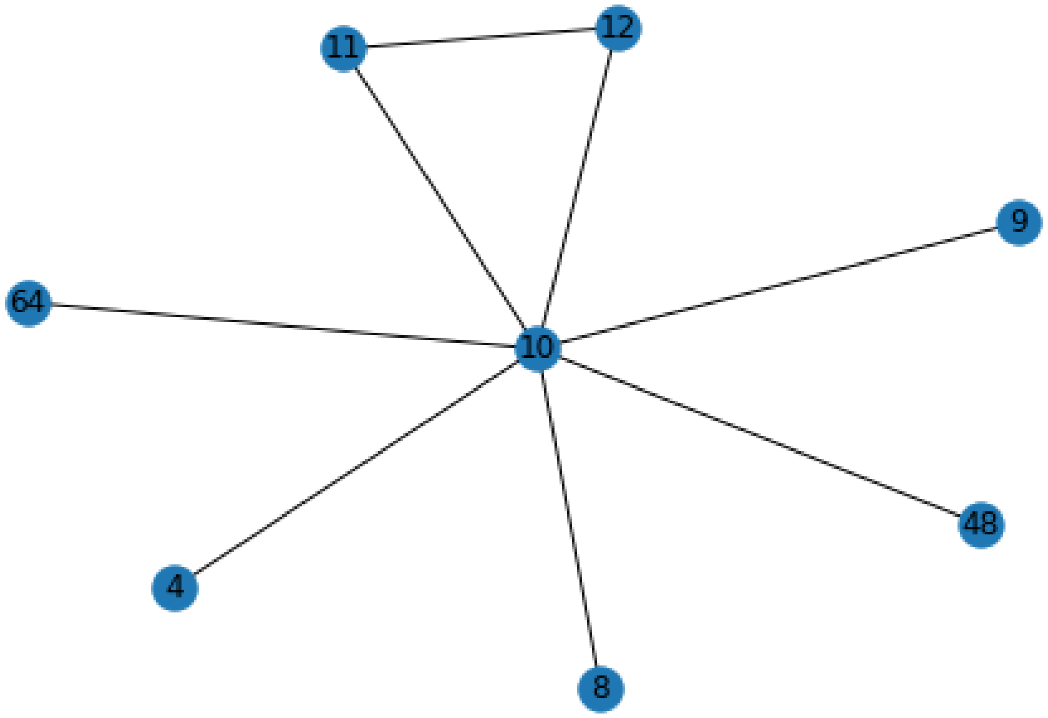
- In a transportation network, betweenness centrality can be interpreted as the importance of a node as a transit hub. A node with high betweenness centrality may be a major transportation hub that connects different parts of the network.
- In a biological network, betweenness centrality can be interpreted as the importance of a protein in the regulation of signaling pathways. A protein with high betweenness centrality may be involved in many different pathways and have a central role in the regulation of cellular processes.
- In a communication network, closeness centrality can be interpreted as the speed at which information can be transmitted to or from a node. A node with high closeness centrality may be able to quickly disseminate information to other nodes in the network.
- In a supply chain network, closeness centrality can be interpreted as the ability of a node to access resources or raw materials. A node with high closeness centrality may be able to quickly obtain the resources it needs to produce goods or services.
- In a social network, closeness centrality can be interpreted as the ability of a person to reach other people in the network. A person with high closeness centrality may be well-connected and able to easily communicate with other people in the network.

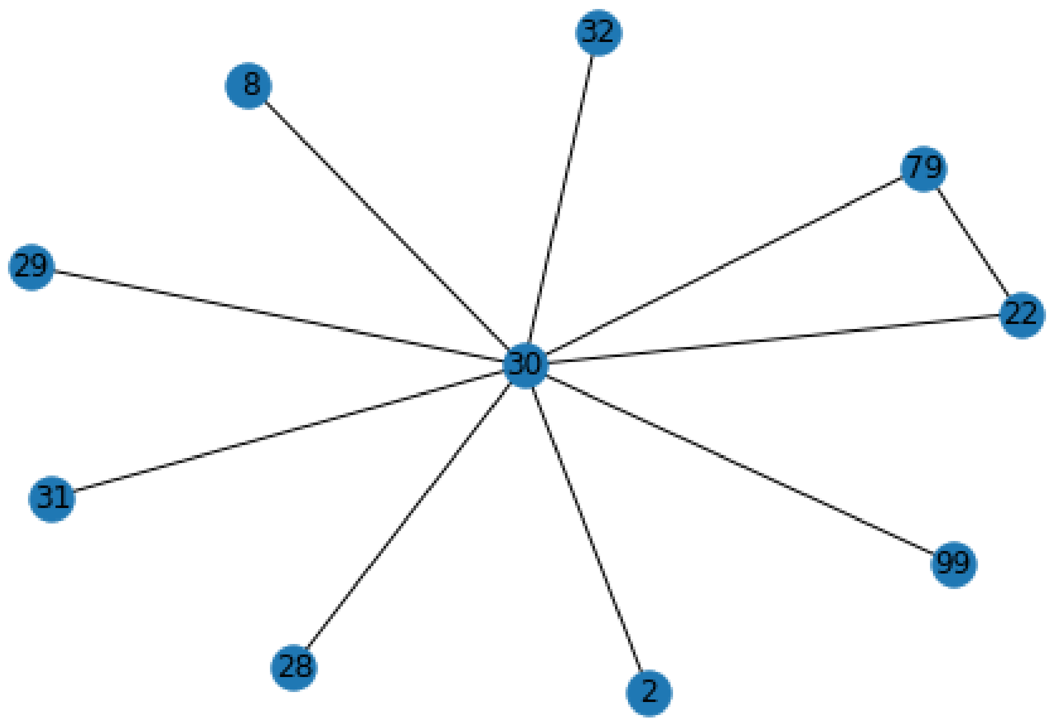
6. Ego-graph To draw ego graph we would be using 0.13 as threshold and katz centrality as centrality measure.

[31]:
```

```


0
49
10
30
8
21
6
72
78


11
12
9
64
10
48
4
8



7. Global Clustering Coefficient The global clustering coefficient of a graph is a measure of the tendency of its nodes to form triangles. A high global clustering coefficient indicates that the nodes in the graph tend to cluster together and form densely connected groups.

To calculate the global clustering coefficient of a random graph using networkx, we can use the average_clustering function.

[32]:
```
# prints the global clustering coefficient
```

```
0.0694206349206349
```

The resulting value is a float between 0 and 1, where 0 indicates that the nodes do not form triangles and 1 indicates that all possible triangles are present in the graph.

8. Local Clustering Coefficient The local clustering coefficient of a node in a graph is a measure of the tendency of its neighbors to form triangles. A high local clustering coefficient for a node indicates that it is surrounded by a dense group of neighbors that are well-connected to each other.

To calculate the local clustering coefficient of the nodes of a random graph using networkx, we can use the clustering function.

[33]:
```
# prints a dictionary with the local clustering coefficients of all nodes
```

```
{0: 0, 1: 0, 2: 0, 3: 0, 4: 0, 5: 0.1, 6: 0.3333333333333333, 7:
0.06666666666666667, 8: 0, 9: 0, 10: 0.047619047619047616, 11:
0.3333333333333333, 12: 0.16666666666666666, 13: 0, 14: 0, 15: 0, 16: 0, 17: 0,
18: 0.16666666666666666, 19: 0, 20: 0, 21: 0, 22: 0.1, 23: 0, 24: 0, 25: 0, 26:
0, 27: 0, 28: 0, 29: 0, 30: 0.027777777777777776, 31: 0, 32: 0.1, 33:
0.6666666666666666, 34: 0.3, 35: 0.13333333333333333, 36: 0.06666666666666667,
37: 0, 38: 0, 39: 0, 40: 0, 41: 0, 42: 0, 43: 0.3333333333333333, 44: 0.1, 45:
0.16666666666666666, 46: 0, 47: 0, 48: 0, 49: 0, 50: 0, 51: 0, 52: 0, 53: 0, 54:
0, 55: 0, 56: 0, 57: 0, 58: 0.1, 59: 0.16666666666666666, 60: 0.1, 61: 0, 62: 0,
63: 0, 64: 0, 65: 0.1, 66: 0, 67: 0, 68: 0, 69: 0, 70: 0.16666666666666666, 71:
0.13333333333333333, 72: 0.06666666666666667, 73: 0, 74: 0.1, 75: 0, 76: 0.1,
77: 0, 78: 0, 79: 0.3333333333333333, 80: 0, 81: 0, 82: 0.16666666666666666, 83:
0, 84: 0, 85: 0, 86: 0.3333333333333333, 87: 0.3333333333333333, 88: 0, 89: 0,
90: 0, 91: 0.06666666666666667, 92: 0.3333333333333333, 93: 0.3333333333333333,
94: 0.3, 95: 0.3333333333333333, 96: 0.16666666666666666, 97: 0, 98: 0, 99: 0}
```

The resulting dictionary maps each node to a float value between 0 and 1, where 0 indicates that the node has no triangles with its neighbors and 1 indicates that all possible triangles are present.

9. Highest Local Clustering Coefficient To identify the highest local clustering coefficient and their significance in terms of structural behavior, we can first sort the local clustering coefficients in descending order and then examine the nodes with the highest values.

[34]:
```
# Print the nodes with the highest local clustering coefficients
```

```
Nodes with the highest local clustering coefficients:
Node 33: 0.67
Node 6: 0.33
Node 11: 0.33
Node 43: 0.33
Node 79: 0.33
```

The significance of the nodes with the highest local clustering coefficients in terms of structural behavior depends on the context in which the graph is used.

- In a social network, for example, a node with a high local clustering coefficient may be a hub of a tightly-knit community or a connector between different communities.
- In a transportation network, a node with a high local clustering coefficient may be a hub that connects multiple routes or a bottleneck that slows down the flow of traffic.
- In a biological network, a node with a high local clustering coefficient may be a key regulator of signaling pathways or a hub of protein-protein interactions.