```
import networkx as nx
import matplotlib.pyplot as plt
import math

def drawGraph(g):
    pos = nx.spring_layout(g)
    plt.figure(figsize=(10,10))

    plt.axis('off')
    plt.show()

#Read in Data from file
DolphinGraph = nx.readwrite.gml.read_gml("dolphins.gml")


#(1) Output Degree Stats
def DegreeOutput(g):
    #Init
    for node in g:
        minDegree = g.degree[node]
        maxDegree = g.degree[node]
        degreeSum = 0
        break


    for node in g:
        if(g.degree[node] < minDegree):
            minDegree = g.degree[node]
        if(g.degree[node] > maxDegree):
            maxDegree = g.degree[node]

        degreeSum += g.degree[node]

    print("=============== #1 ===============")
    print("Max Degree: ", maxDegree)
    print("Min Degree: ", minDegree)
    print("Average Degree: ", degreeSum/len(g.nodes))



def CharacteristicPathLength(g):
    print("\n=============== #2 ===============")
    print("Average Shortest Path Length: ", nx.average_shortest_path_length(g))

def Diameter(g):
```

```
    print("\n=============== #3 ===============")
    print("Diameter: ", nx.diameter(g))

def CenterAndPeripheral(g):
    print("\n=============== #4 ===============")
    print("Center Nodes: ", nx.center(g))
    print("Peripheral Nodes: ", nx.periphery(g))


#I think you can calculate it if you take the entries in the apsp matrix (minus t
he ones along the diagonal), sort them, and take the average of the top 90%. Try
that on the example we did in class.
def EffectiveEccentricity(g):
    print("\n=============== #5 ===============")
    totalDolphins = len(list(g.nodes))
    dolphinsToKeep = math.floor(totalDolphins * 0.9)

    #Calculate every shortest path for every dolphin
    spaths = dict(nx.all_pairs_shortest_path_length(g))

    paths = list()
    ctr = 0

    for node1 in spaths:
        for node2 in spaths[node1]:
            if(node1 != node2):
                paths.append(spaths[node1][node2])

    #Sort in descending order
    paths.sort(reverse=True)

    for i in range(len(paths)):
        if(paths[i] == 1):
            ctr = ctr+1

    #Keep top 90% highest values
    paths = paths[:math.floor(len(paths)*.9)]

    #Average of this list (consisting of top 90% of shortest path values)
    print("Effective Eccenctricity: ", sum(paths)/len(paths))


def Density(g):
    print("\n=============== #6 ===============")
    print("Density: ", nx.density(g))
```

```
def ClusterCoeff(g):
    '''A graph is small-
worldif C is significantly higher than it would be for a random graph constructed
 on the same vertex set with approximately the same diameter'''
    c = nx.clustering(g)

    # compute overall coefficient
    sum = 0
    for x in c:
        sum = sum + c[x]

    print("\n=============== #7 ===============")
    print("Cluster Coefficient: ", sum/g.number_of_nodes())


def Transitivity(g):
    print("\n=============== #8 ===============")
    print("Transitivity: ", nx.transitivity(g))


def BetweennessCentrality(g):
    BC = nx.betweenness_centrality(g, normalized=True)
    node_color = [5000 * g.degree(v) for v in g]
    node_size = [v * 10000 for v in BC.values()]

    plt.figure(figsize=(20,20))
    nx.draw_networkx(g, with_labels=True, node_color=node_color, node_size=node_s
ize, font_size=9, font_color="black")
    plt.axis('off')
    plt.show()

def EigenvectorCentrality(g):
    EC = nx.eigenvector_centrality(g)
    node_color = [5000 * g.degree(v) for v in g]
    node_size = [v * 6000 for v in EC.values()]

    plt.figure(figsize=(20,20))
    nx.draw_networkx(g, with_labels=True, node_color=node_color, node_size=node_s
ize, font_size=9, font_color="black")
    plt.axis('off')
    plt.show()
```

```
def ClosenessCentrality(g):
    CC = nx.closeness_centrality(g)
    node_color = [5000 * g.degree(v) for v in g]
    node_size = [v * 3500 for v in CC.values()]

    plt.figure(figsize=(20,20))
    nx.draw_networkx(g, with_labels=True, node_color=node_color, node_size=node_s
ize, font_size=9, font_color="black")
    plt.axis('off')
    plt.show()


#1-8 drivers
DegreeOutput(DolphinGraph)
CharacteristicPathLength(DolphinGraph)
Diameter(DolphinGraph)
CenterAndPeripheral(DolphinGraph)
EffectiveEccentricity(DolphinGraph)
Density(DolphinGraph)
ClusterCoeff(DolphinGraph)
Transitivity(DolphinGraph)

#9 Drivers
BetweennessCentrality(DolphinGraph)
EigenvectorCentrality(DolphinGraph)
ClosenessCentrality(DolphinGraph)

#drawGraph(DolphinGraph)
```

Value Output

Does this network exhibit a small-world effect?

This network exhibits a small-world effect because it has a relatively small diameter compared to the total number of nodes in the graph (diameter is 8 and total # of nodes is 62, 0.125.)

The effective eccentricity is also relatively small compared to the total number of nodes, effective eccentricity is ~3.6

The characteristic path length is relatively small as well, as the average shortest path from any dolphin to any other dolphin was ~3.4.

All of these factors indicate that this graph does exhibit a small-world effect.

Summary of 3 visualizations:

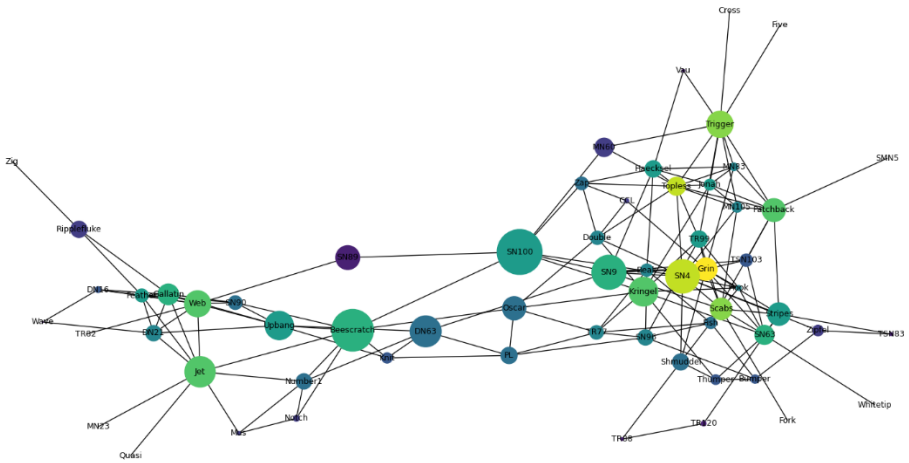
BetweenNess Centrality - This graph shows which dolphins have the highest number of shortest paths that run through them. You can see that dolphins SN100, Beescratch, and Jet have some of the highest

values for this metric. This means that these are the most "bridgey" dolphins for how information is communicated between groups.

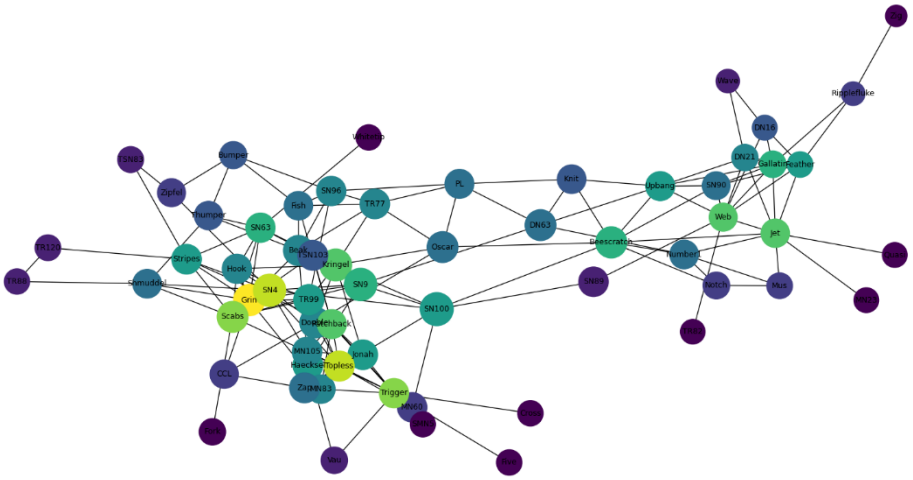
Eigen Centrality - This metric shows the dolphins that are the most "prestegious", meaning that their connections also have a large number of connections. You can see that Grin, Topless, SN4, and Scabs have the highest scores amongst all the dolphins. These dolphins would be the best ones to give information to if you wanted information to spread the quickest to all other dolphins.

Closeness Centrality - Each dolphin has roughly the same score for their closeness centrality. This means that the graph can be considered well connected. If you give information to any dolphin, it will not be long before every dolphin knows the information as well. This metric is in agreement with the fact that the diameter of this graph is low, relative to the number of dolphins total.

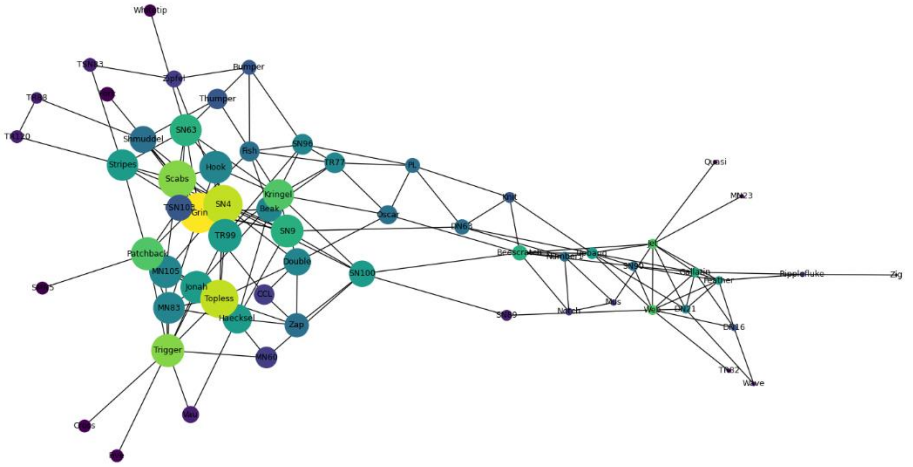
I do not see any conflicting information amongst these three visualizations, and I would conclude that this network is well connected and cohesive. Intelligent animals are typically highly social, and this graph showing a well connected network of dolphins was the expected outcome.



Betweenness Centrality (Above)



Closeness Centrality (Above)



Eigenvector Centrality (Above)