Let's learn about centralities!

For this assignment we will be exploring several centralities to get an intuitive sense of what the various centrality metrics tell us about the nodes in the graph.

Before messing with data, let's think about one of the foundational centralities called eigenvector centrality. The idea is an extension of the degree centrality. Instead of defining the centrality of a node as its degree, eigenvector centrality defines the centrality of a node as the sum of eigenvector centrality of the neighbors. In other words, an important node is not just a node with many neighbors, but the one with many *important* neighbors.

Mathematically this idea can be translated into finding the eigenvector of the adjacency matrix that corresponds to the largest (and positive) eigenvalue. According to the Perron-Frobenius theorem, there is only one non-negative eigenvector and it is the leading eigenvector with the largest eigenvalue.

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$$

Here is a question: consider an undirected k-regular graph with only one connected component. That means that everyone can be reached from everyone else and every node's degree is k. What would be the eigenvector centrality vector of this graph?

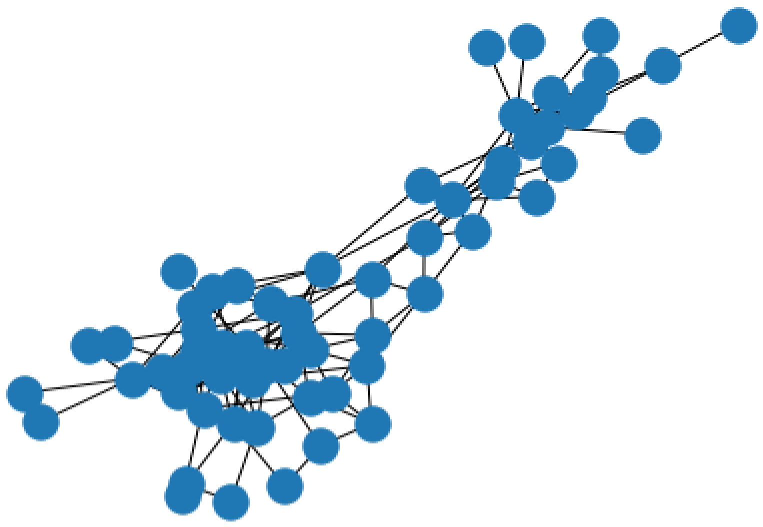
Your answer here

We will be using the Dolphin social network. (You may need to copy the link address into a new tab/window to trigger the download.) Download the graph and load it as a networkx graph.

In [1]:
```python
import networkx as nx
dolphin_social_network = nx.read_gml('dolphins.gml')
nx.info(dolphin_social_network)
```

Out[1]:
```
'Graph with 62 nodes and 159 edges'
```

In [2]:
```python
nx.draw(dolphin_social_network)
```



Centrality in Networkx

Networkx has several functions available for calculating the centralities of the nodes in the graph. There are functions for eigenvector, katz, closeness, betweenness, degree, etc. For a full list you can visit the documentation page.

These functions take a graph as an argument and return a dictionary with nodes as keys and the centrality as values. This is convenient for us because we can set these as attributes for the nodes in the graph using the `set_node_attributes` function. For example:

In [3]:
```
import networkx as nx

my_graph = nx.erdos_renyi_graph(500, 0.3)

# Get the eigenvector centralities for all the nodes
centralities = nx.eigenvector_centrality(my_graph)

# Set the attributes of the nodes to include the centralities
# The arguments are: <graph> <attribute key> <values>
# Where <values> is a dictionary with keys=nodes
nx.set_node_attributes(my_graph, centralities, "eigenvector")

# Now we can refer to the node's attributes in the graph
print(my_graph.nodes[3]["eigenvector"])
```

```
0.03964152067291196
```

We want to do this so that we can export our graph as a `gexf` file using networkx's write_gexf function. Gexf is able to contain a lot more information than other graph datatypes like pajek. It can contain information about the node attributes or edge attributes that belong to the graph and then these attributes will be recognized by Gephi for plotting.

Alternatively, if you use Cytoscape, you can export the centralities as a node property file, which is just a CSV contains the node IDs as the first column, and centralities as the other columns. Cytoscape can read CSV files through "important data tables" functionality: https://manual.cytoscape.org/en/stable/Node_and_Edge_Column_Data.html

Once the graph is saved and you open it in Gephi or cytoscape, you can use the node (or edge) attributes to control node (or edge) size and color.

You can then arrange your nodes accordingly and then save separate visualizations that only change the node color/size according to your saved attributes. You will be using this ability for the following questions.

What to submit: Turn in a PDF that contains your short responses and the visualizations for each of the following questions. **Keep the node location the same** for your graph visualizations.

Picking the right Dolphins

Answer the following questions:

In [16]:
```
#computing centrality measures for the Dolphin network and adding node attributes with va

def add_attributes_node(dolGraph, values, attribute):
  for node, val in values.items():
    dolGraph.nodes[node][attribute] = val
```

In [10]:
```
def getcentrality(dolGraph, func):
  return func(dolGraph)
```

In [11]:
```
def finishGraph(dolGraph, func, name):
  centrality = getcentrality(dolGraph, func)
  add_attributes_node(dolGraph, centrality, name)
```

In [7]:
```
finishings = {
    "degree": nx.degree_centrality,
    "eigenvector": nx.eigenvector_centrality,
    "closeness": nx.closeness_centrality,
    "harmonic": nx.harmonic_centrality,
    "betweenness": nx.betweenness_centrality,
    "katz": nx.katz_centrality
}
```

In [14]:
```
for name, function in finishings.items():
  print("Finishing graph nodes with centrality measure:", name)
  finishGraph(dolphin_social_network, function, name)
```

```
Finishing graph nodes with centrality measure: degree
Finishing graph nodes with centrality measure: eigenvector
Finishing graph nodes with centrality measure: closeness
Finishing graph nodes with centrality measure: harmonic
Finishing graph nodes with centrality measure: betweenness
Finishing graph nodes with centrality measure: katz
```

In [15]:
```
dolphin_social_network.nodes['Beak']
```

Out[15]:
```
{'betweenness': 0.01908259621374376,
 'closeness': 0.3465909090909091,
 'degree': 0.09836065573770492,
 'eigenvector': 0.1285035191108721,
 'harmonic': 25.983333333333324,
 'katz': 0.1352761473111646}
```

In [17]:
```
nx.write_gexf(dolphin_social_network, "dolphin_centrality.gexf")
```

1) Popularity contest

We want to know who the top dolphins are in the network, the real centers of attraction. Using what you learned about centrality from the readings and videos, choose an appropriate centrality measure that will tell us who those dolphins are. Justify your decision and list who the important dolphins are

Answer

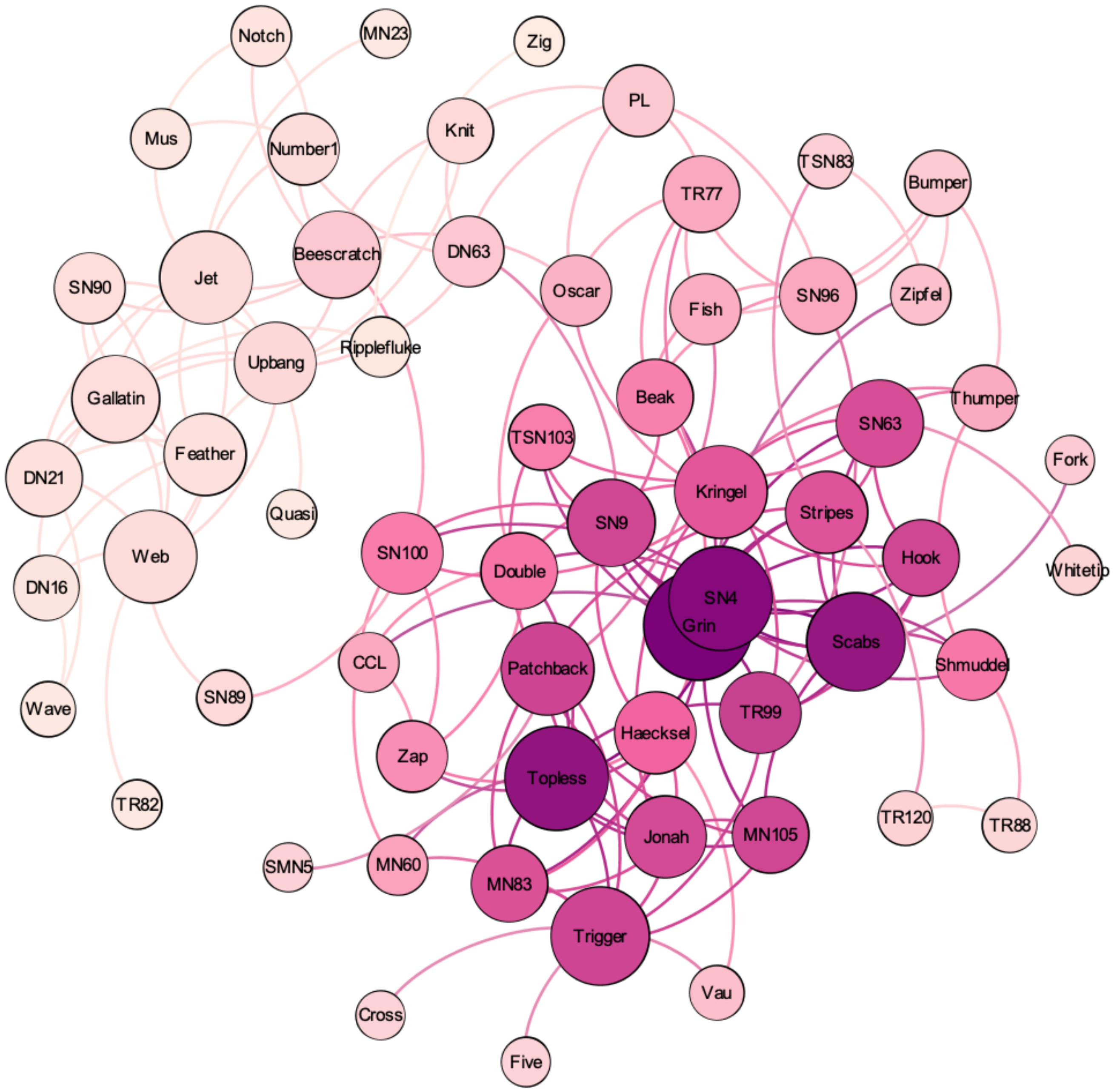
So, we want to show popularity here. I considered the centrality measures which shows the number of connections/ how strongly connected a node is. This can be achieved by centrality measures that are based on *degree*. I used eigenvector centrality here to show the top dolphins in the "entire network".

Eigenvector centrality

Top Popular Dolphins :

(highlighted in purple in the network below)

- Scabs
- SN4
- Topless
- Grin



(2) **Relay**

Dolphins like passing information around efficiently along the shortest-paths. Among their neighbors who are the most important message relayers in the network? Justify your centrality choice for finding these dolphins.

Answer

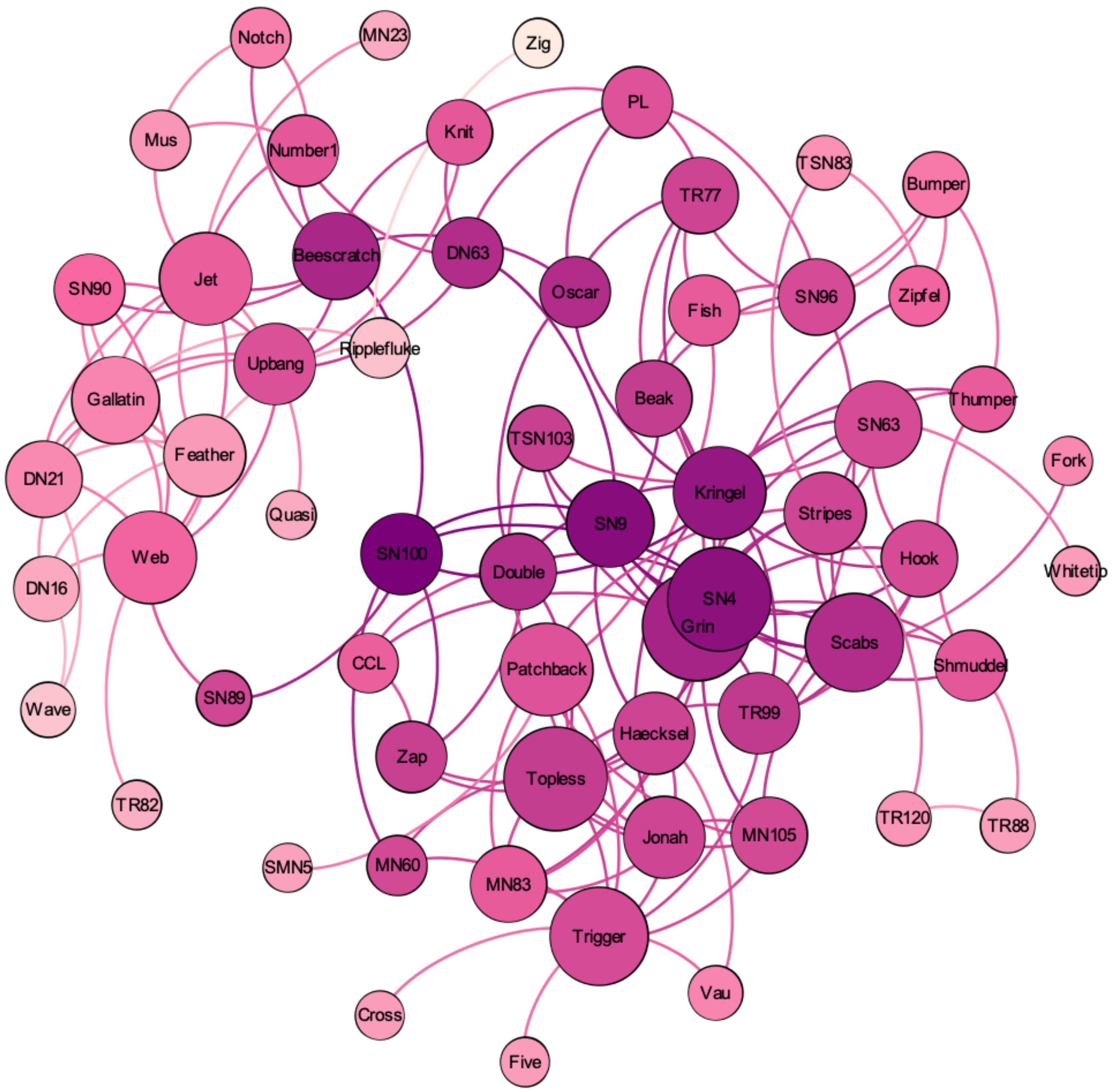
We need to find the nodes with access to the shortest paths in the network here. The nodes with most amount of shortest paths accessible to them would be most influential in relaying information most quickly in the network. I used closeness centrality measure here therefore.

Closeness centrality

Top Important Dolphins :

(highlighted in purple in the network below)

- Kringel
- SN100
- SN4
- SN9



(3)Gossip

There is a lot smack going around the pod and everyone wants to know if Flipper will be inviting them to the party next week. But gossip takes time travel. Which dolphins are in the best position for getting all the best gossip from around the pod? Justify your centrality choice for finding these dolphins.

Answer

To pass around the most amount of information requires people to have a certain kind of connections. It is not necessary to have the most amount of connections but it is about having a kind of connectivity that fills gaps and allows flow of information between various connections in their network. I therefore think betweenness centrality would be an appropriate measure to answer this question.

Betweenness centrality

Best positioned Dolphins :

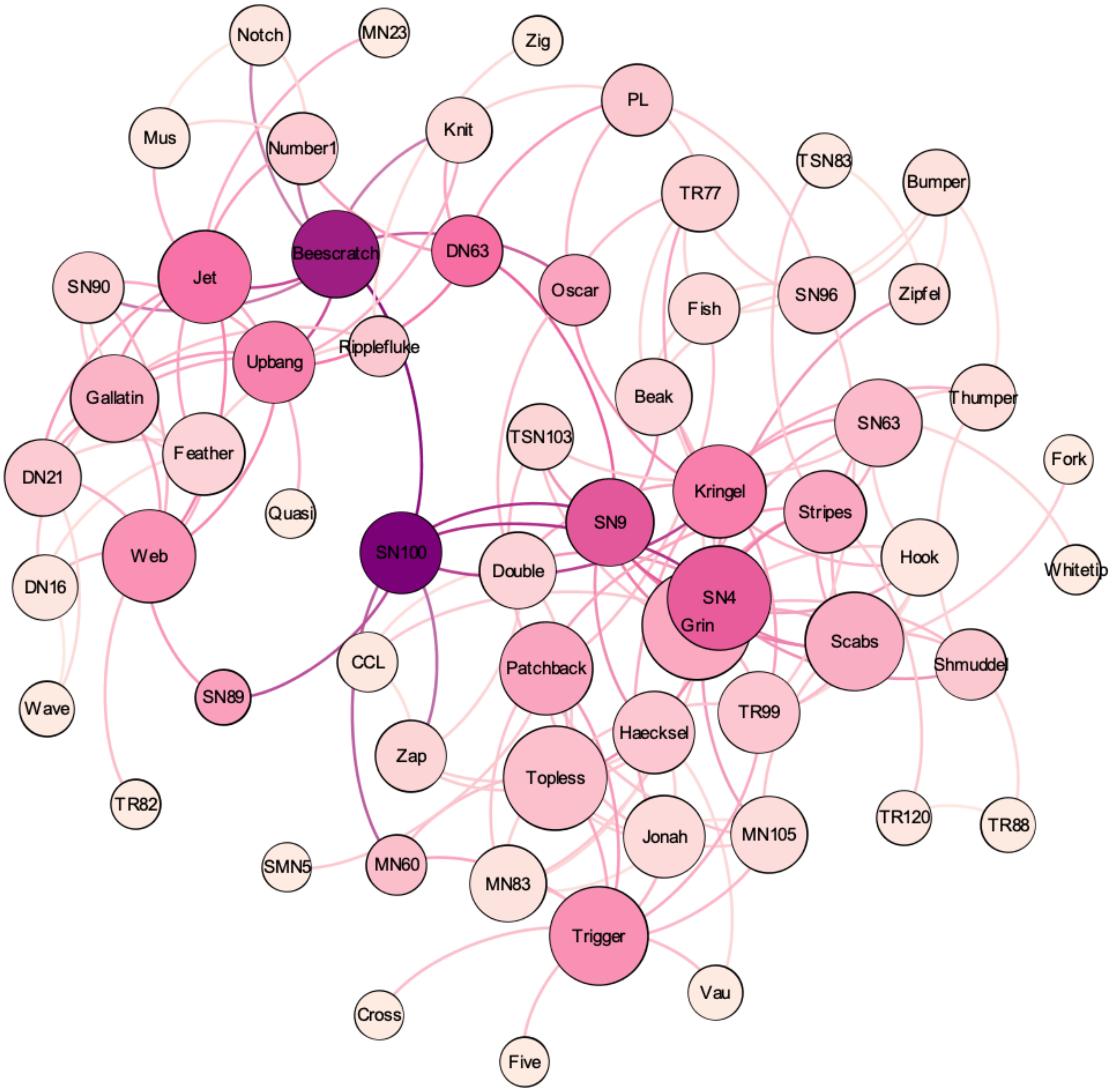
(highlighted in purple in the network below)

- Beescratch
- SN100

Slightly less important Dolphins :

(highlighted in dark pink in the network below)

- SN4
- SN9


Notch
MN23
Zig
Mus
Number1
Knit
PL
TSN83
Bumper
TR77
Beescratch
DN63
Jet
SN90
Oscar
Fish
SN96
Zipfel
Upbang
Ripplefluke
Gallatin
Beak
Thumper
SN63
Feather
TSN103
Fork
DN21
Quasi
Kringel
Stripes
SN9
SN100
Web
Hook
Whitetip
DN16
Double
SN4
Grin
Scabs
CCL
Shmuddel
Patchback
SN89
Wave
TR99
Haecksel
Zap
Topless
TR82
TR120
TR88
Jonah
MN105
SMN5
MN60
MN83
Trigger
Vau
Cross
Five