22mcb1002

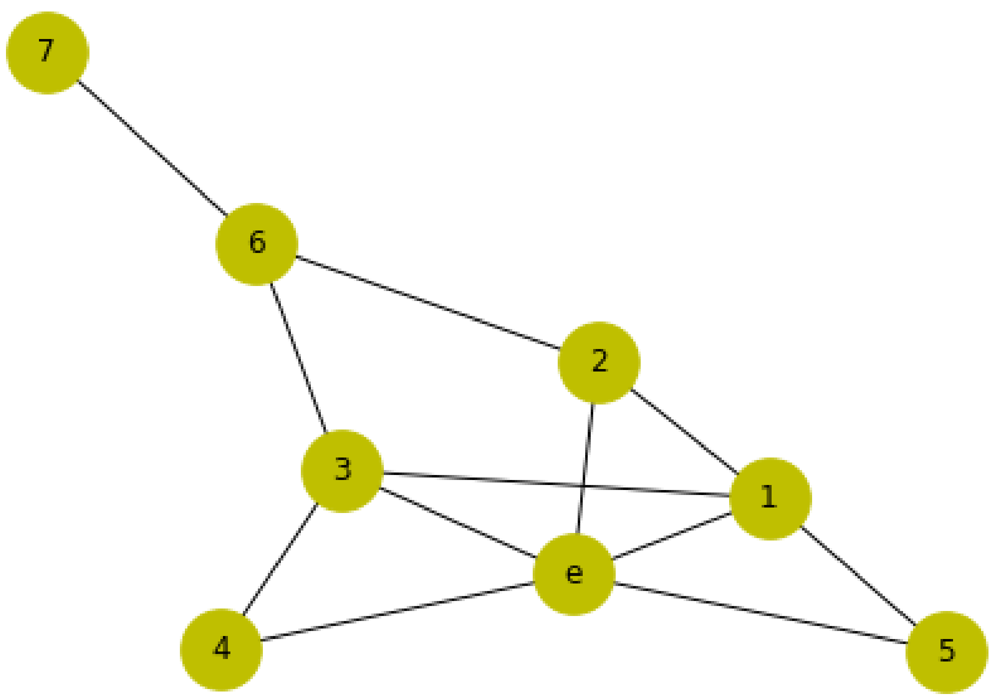
March 3, 2023

[1]:
```python
import networkx as nx
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

[81]:
```python
G = nx.Graph()
G.add_nodes_from([1, 2, 3, 4, 5, 6, 7, 'e'])
G.add_edges_from([('e',1), ('e',2), ('e',3), ('e',4), ('e',5), (1,2), (1,3),
                  (1,5), (2,6), (3,4), (3,6), (6,7)])
```

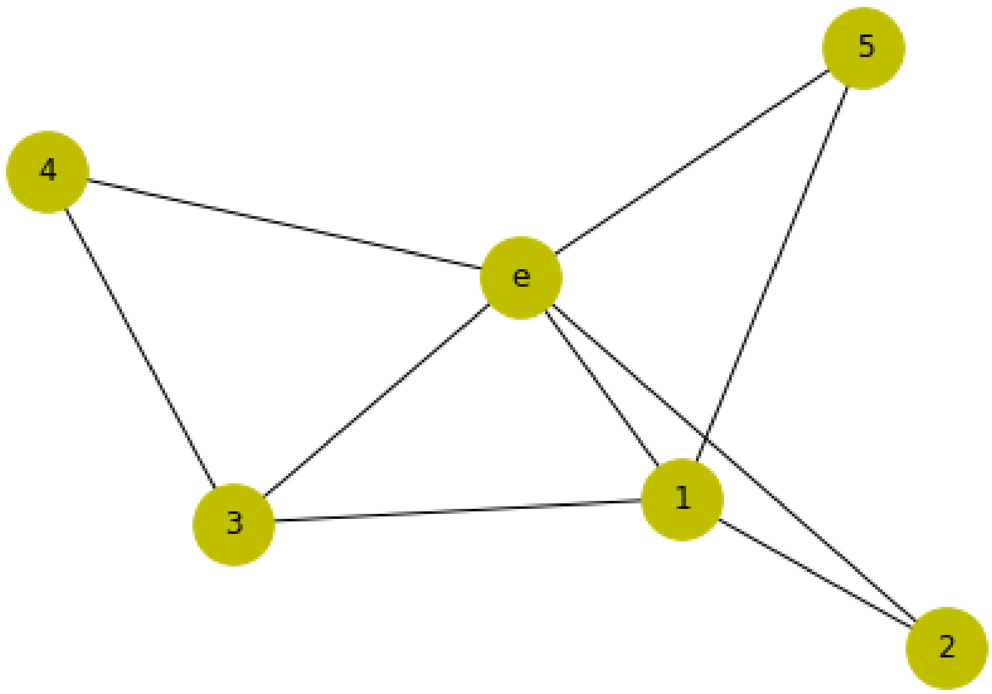
1. Graph Visualisation

[83]:
```python
nx.draw(G, with_labels = True, node_color = 'y', node_size = 1000)
plt.show()
```



1.1 Ego network from given network

[ ]:
```
EG = nx.ego_graph(G, 'e', radius = 1, center = True)
nx.draw(EG, with_labels = True, node_color = 'y', node_size = 1000)
plt.show()
```



2. **Sociomatrix**

[ ]:
```
socmat = nx.to_pandas_adjacency(G)
print(socmat)
```

```
     1    2    3    4    5    6    7    e
1  0.0  1.0  1.0  0.0  1.0  0.0  0.0  1.0
2  1.0  0.0  0.0  0.0  0.0  1.0  0.0  1.0
3  1.0  0.0  0.0  1.0  0.0  1.0  0.0  1.0
4  0.0  0.0  1.0  0.0  0.0  0.0  0.0  1.0
5  1.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0
6  0.0  1.0  1.0  0.0  0.0  0.0  1.0  0.0
7  0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0
e  1.0  1.0  1.0  1.0  1.0  0.0  0.0  0.0
```

3. Boys and girls compostion

```
[ ]: nodeCharacteristics = {1: 'boy', 2: 'boy', 3: 'girl', 4: 'girl', 5: 'girl',
                            6: 'boy', 7: 'boy'}
     nx.set_node_attributes(G, nodeCharacteristics, name = 'gender')

     boys = sum(1 for gender in nx.get_node_attributes(G, 'gender').values() if␣
      ↪gender == 'boy')
     girls = len(G) - boys -1
     boysPercent = (boys / (len(G)-1)) * 100
     girlsPercent = (girls / (len(G)-1)) * 100

     print(f"Boys composition: {boysPercent: .2f}%")
     print(f"Girls composition: {girlsPercent: .2f}%")
```

```
Boys composition:  57.14%
Girls composition:  42.86%
```

4. Ego network analysis of E

4.a Density/Clustering coefficient

```
[ ]: density = nx.density(EG)
     print("Density: ", density)

     cluster = nx.clustering(G, 'e')
     print("Clustering coefficient: ", cluster)
```

```
Density:  0.6
Clustering coefficient:  0.4
```

4.b Effective size

```
[ ]: Esize = nx.effective_size(G)
     print("Effective size: ")
     for node, size in Esize.items():
       if node == 'e':
         print(f"Node {node}: {size:.2f}")
```

```
Effective size:
Node e: 3.40
```

4.c Reciprocity

```
[ ]: reciprocity = nx.reciprocity(G)
     print(f"Reciprocity: {reciprocity:.2f}")
```

```
Reciprocity: 0.00
```

4.d Alters

```
[ ]: alters = list(G.neighbors('e'))
     print(f"Alters of node e: {alters}")
```

```
Alters of node e: [1, 2, 3, 4, 5]
```

4.e Connection between alters

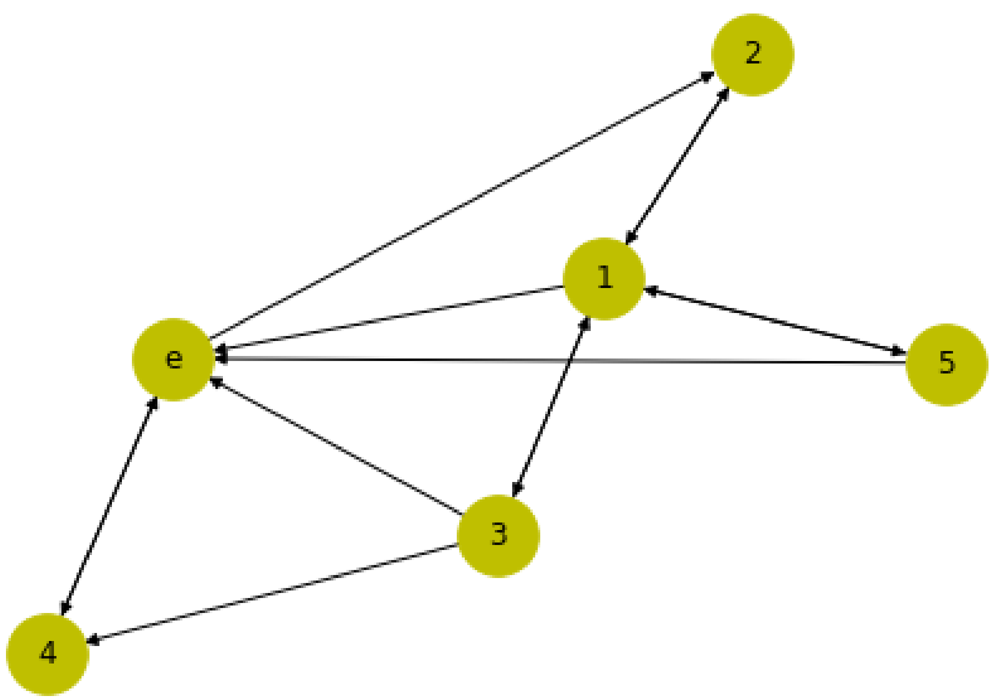
[ ]:
```python
alters_connect = G.subgraph(alters)

alters_density = nx.density(alters_connect)
alters_clustering = nx.average_clustering(alters_connect)

print(f"Density of alters subgraph: {alters_density:.2f}")
print(f"Clustering coefficient of alters subgraph: {alters_clustering:.2f}")
```

```
Density of alters subgraph: 0.40
Clustering coefficient of alters subgraph: 0.00
```

5. Neighbours of node E

[45]:
```python
G_new = nx.DiGraph()
G_new.add_nodes_from([(1, {'gender': 'Male'}), (2, {'gender': 'Female'}), (3,␣
 ↪{'gender': 'Male'}),
                     (4, {'gender': 'Male'}), ('e', {'gender': 'Female'}), (5,␣
 ↪{'gender': 'Female'})])
G_new.add_edges_from([('e',2), ('e',4), (1,2), (2,1), (1,5), (5,1), (3,4), (1,2)
                         ,(1,3), (3,1), (1,'e'), (3,'e'), (5,'e'),(4,'e')])
```

[38]:
```python
nx.draw(G_new, with_labels = True, node_color = 'y', node_size = 1000)
plt.show()
```



5.a EI Homophily

[56]:
```
count = 0
for node in G_new.nodes.data():
    if node[1]['gender'] == 'Female':
        count += 1
ei_homo = (count-1)/G_new.degree('e')
print("EI homophily: ", ei_homo)
```

```
EI homophily:  0.3333333333333333
```

5.b Heterogeneity Index

[61]:
```
total = G_new.number_of_nodes()
E_node = total-1
male_cnt = 0
for node in G_new.nodes.data():
    if node[1]['gender'] == 'Male':
        male_cnt += 1
female_cnt = E_node - male_cnt
p_male = male_cnt / E_node
p_female = female_cnt / E_node
Hetgen = 1 -(p_male**2)*(p_female**2)
print("Blau's Heterogeneity Index: ", Hetgen)
```

```
Blau's Heterogeneity Index:  0.9424
```

6. Different groups using hop distance

[66]:
```
# Deterministic graph
DG = nx.Graph()

DG.add_nodes_from([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
DG.add_edges_from([(1, 2), (1, 8), (1, 3), (2, 3), (3, 4), (4, 5), (5, 6), (5,␣
 ↪7),
                          (5,9),(6, 7), (7, 8), (8, 9), (9, 10)])

for node in DG.nodes():
    ego = nx.ego_graph(DG, node)

    # Support clique
    support_clique = list(nx.neighbors(ego, node))

    # Sympathy group
    sympathy_group = list(set(nx.single_source_shortest_path_length(ego, node,
                     cutoff=2)) - set(support_clique) - set([node]))
```

```python
    # Affinity group
    affinity_group = list(set(nx.single_source_shortest_path_length(ego, node,
                    cutoff=3)) - set(support_clique) - set(sympathy_group) -
 ↪set([node]))

    # Active network
    active_network = list(set(G.nodes()) - set(support_clique) -
                    set(sympathy_group) - set(affinity_group) - set([node]))

    print(f"Node {node}:")
    print(f"Support clique: {support_clique}")
    print(f"Sympathy group: {sympathy_group}")
    print(f"Affinity group: {affinity_group}")
    print(f"Active network: {active_network}")
```

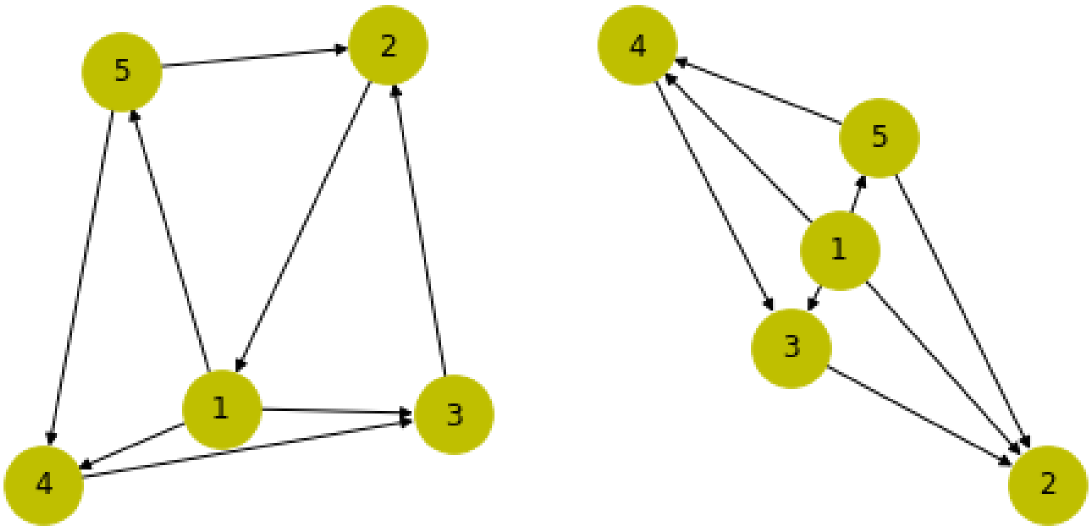
```
Node 1:
Support clique: [8, 2, 3]
Sympathy group: []
Affinity group: []
Active network: [4, 5, 6, 7, 9, 10]
Node 2:
Support clique: [1, 3]
Sympathy group: []
Affinity group: []
Active network: [4, 5, 6, 7, 8, 9, 10]
Node 3:
Support clique: [1, 2, 4]
Sympathy group: []
Affinity group: []
Active network: [5, 6, 7, 8, 9, 10]
Node 4:
Support clique: [3, 5]
Sympathy group: []
Affinity group: []
Active network: [1, 2, 6, 7, 8, 9, 10]
Node 5:
Support clique: [4, 6, 7, 9]
Sympathy group: []
Affinity group: []
Active network: [1, 2, 3, 8, 10]
Node 6:
Support clique: [5, 7]
Sympathy group: []
Affinity group: []
Active network: [1, 2, 3, 4, 8, 9, 10]
Node 7:
```

```
Support clique: [8, 5, 6]
Sympathy group: []
Affinity group: []
Active network: [1, 2, 3, 4, 9, 10]
Node 8:
Support clique: [1, 7, 9]
Sympathy group: []
Affinity group: []
Active network: [2, 3, 4, 5, 6, 10]
Node 9:
Support clique: [8, 5, 10]
Sympathy group: []
Affinity group: []
Active network: [1, 2, 3, 4, 6, 7]
Node 10:
Support clique: [9]
Sympathy group: []
Affinity group: []
Active network: [1, 2, 3, 4, 5, 6, 7, 8]
```

7. Strongly connected graph

[69]:
```
G1 = nx.DiGraph()
G1.add_nodes_from([1, 2, 3, 4, 5])
G1.add_edges_from([(1,5), (2,1), (3,2), (4,3), (5,4), (1,3), (1,4), (5,2)])

G2 = nx.DiGraph()
G2.add_nodes_from([1, 2, 3, 4, 5])
G2.add_edges_from([(1,2), (1,5), (3,2), (4,3), (5,4), (1,3), (1,4), (5,2)])
```

[75]:
```
fig, (ax1, ax2) = plt.subplots(ncols = 2, figsize = (8, 4))
nx.draw(G1, with_labels = True, node_color = 'y', node_size = 1000, ax = ax1)
nx.draw(G2, with_labels = True, node_color = 'y', node_size = 1000, ax = ax2)
plt.show()
```



[77]:
```
if nx.is_strongly_connected(G1):
  print("G1 is Strongly connected graph\n")
else:
  print("G1 is not a strongly connected graph")

if nx.is_strongly_connected(G2):
  print("G2 is Strongly connected graph\n")
else:
  print("G2 is not a strongly connected graph")
```

```
G1 is Strongly connected graph

G2 is not a strongly connected graph
```