Definitions & basic recap

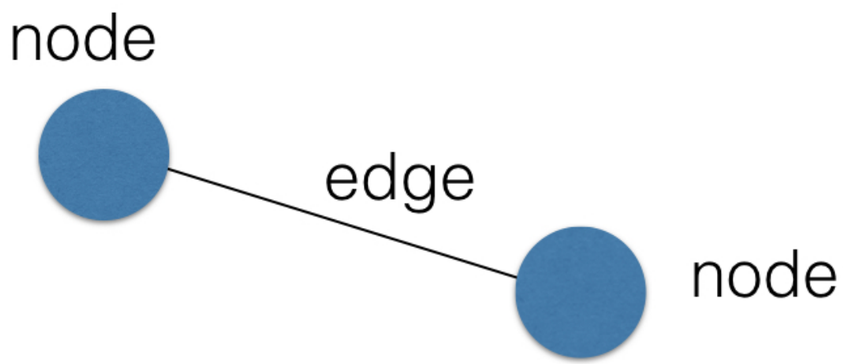
NETWORK ANALYSIS IN PYTHON (PART 2)

Eric Ma

Data Carpentry instructor and author of nxviz package

Network/Graph

- Network = Graph = (nodes, edges)
- Directed or Undirected
  - Facebook: Undirected
  - Twitter: Directed
- `networkx` : API for analysis of graphs



Basic NetworkX API

```
import networkx as nx

G
```

```
<networkx.classes.graph.Graph at 0x10b192da0>
```

```
G.nodes()
```

```
['customer1', 'customer3', 'customer2']
```

Basic NetworkX API

```
len(G.nodes())
```

```
3
```

```
len(G.edges())
```

```
2
```

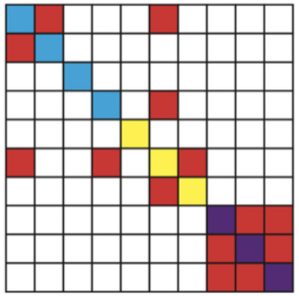
```
type(G)
```

```
networkx.classes.graph.Graph
```

Network visualization

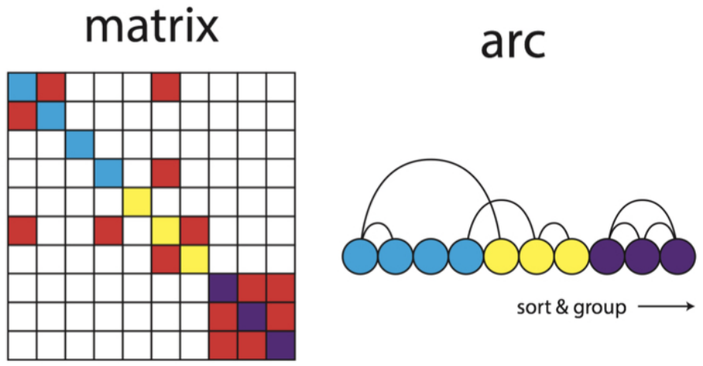
- nxviz: API for creating beautiful and rational graph viz
- Prioritize placement of nodes

matrix



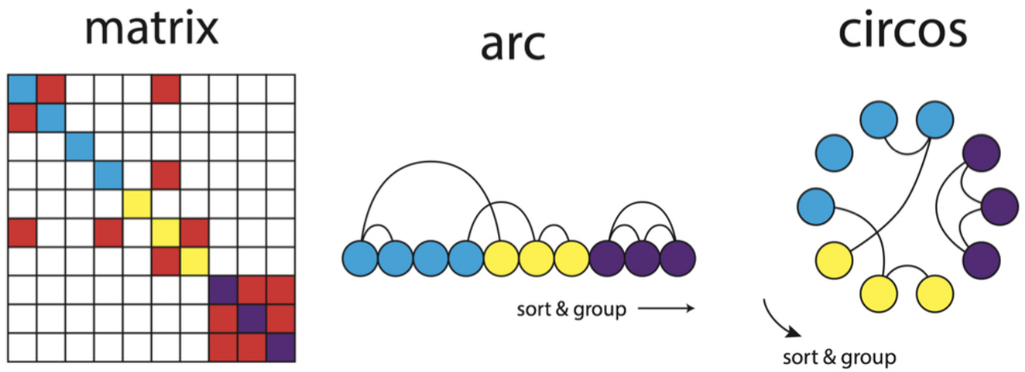
Network visualization

- nxviz: API for creating beautiful and rational graph viz
- Prioritize placement of nodes



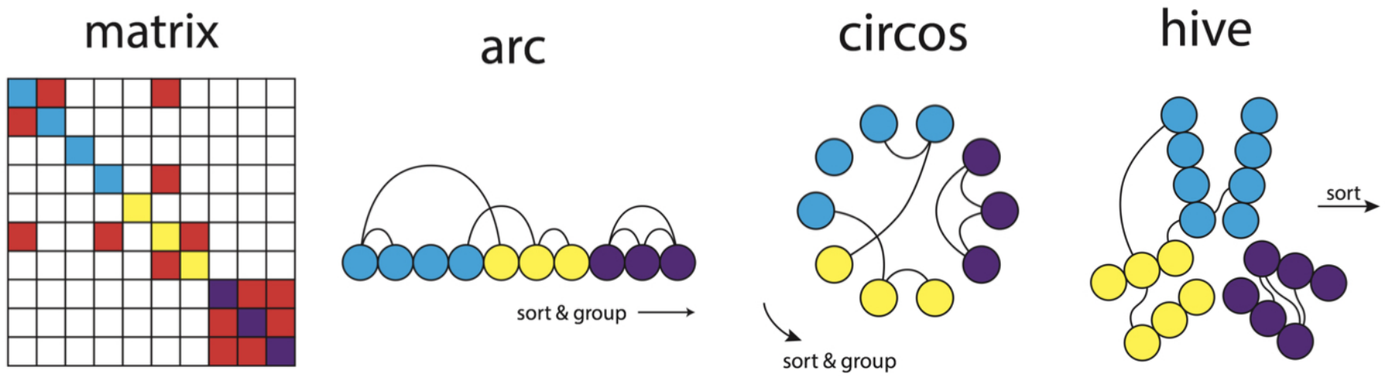
Network visualization

- nxviz: API for creating beautiful and rational graph viz
- Prioritize placement of nodes



Network visualization

- nxviz: API for creating beautiful and rational graph viz
- Prioritize placement of nodes



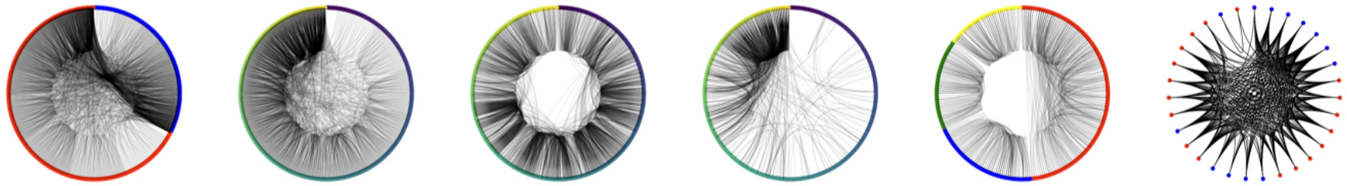
Basic nxviz API

```
import nxviz as nv
import matplotlib.pyplot as plt

c = nv.CircosPlot(G)

c.draw()

plt.show()
```



Let's practice!

NETWORK ANALYSIS IN PYTHON (PART 2)

Bipartite graphs

NETWORK ANALYSIS IN PYTHON (PART 2)

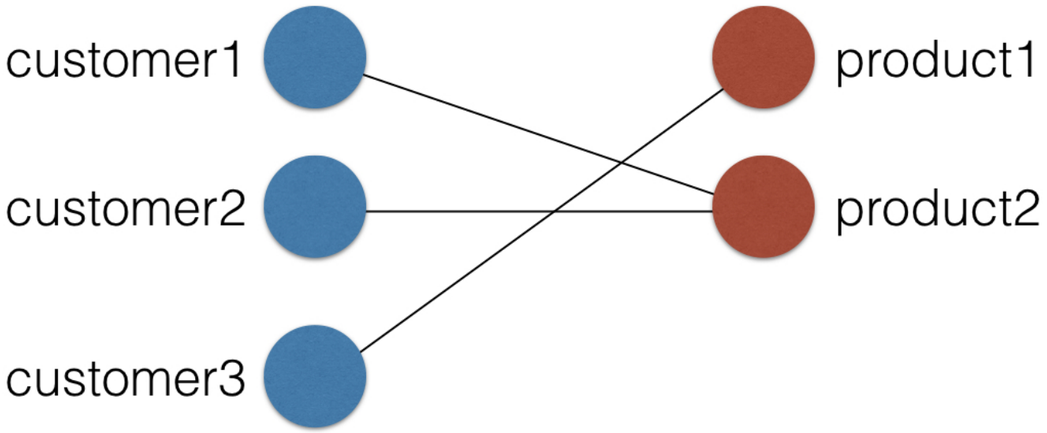
Eric Ma

Data Carpentry instructor and author of nxviz package

Bipartite graphs

- A graph that is partitioned into two sets
- Nodes are only connected to nodes in other partitions
- Contrast: “unipartite”

Bipartite graphs: Example



Bipartite graphs in NetworkX

```
import networkx as nx
G = nx.Graph()

numbers = range(3)
G.add_nodes_from(numbers, bipartite='customers')

letters = ['a', 'b']
G.add_nodes_from(letters, bipartite='products')
```

Bipartite graphs in NetworkX

```
G.nodes(data=True)
```

```
[(0, {'bipartite': 'customers'}),
 (1, {'bipartite': 'customers'}),
 (2, {'bipartite': 'customers'}),
 ('b', {'bipartite': 'products'}),
 ('a', {'bipartite': 'products'})]
```

Degree centrality

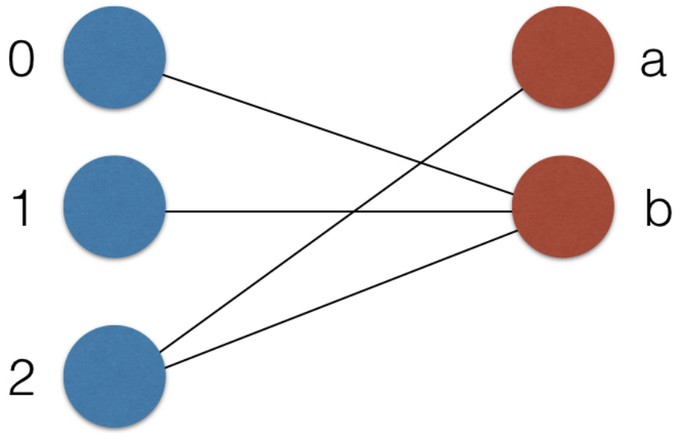
- Definition:

$$\frac{\text{number of neighbors}}{\text{number of possible neighbors}}$$

- Number of possible neighbors depends on graph type

Bipartite centrality metrics

- Denominator: number of nodes in opposite partition, rather than all other nodes



Filtering graphs

```
cust_nodes = [n for n, d in G.nodes(data=True) if
                  d['bipartite'] == 'customers']
```

```
cust_nodes
```

```
[(0, {'bipartite': 'customers'}),
 (1, {'bipartite': 'customers'}),
 (2, {'bipartite': 'customers'})]
```

```
nx.bipartite.degree_centrality(G, cust_nodes)
```

```
{0: 0.5,
 1: 0.5,
 2: 1.0,
 'a': 0.333,
 'b': 1.0}
```

Let's practice!

NETWORK ANALYSIS IN PYTHON (PART 2)

Bipartite graphs and recommendation systems

NETWORK ANALYSIS IN PYTHON (PART 2)

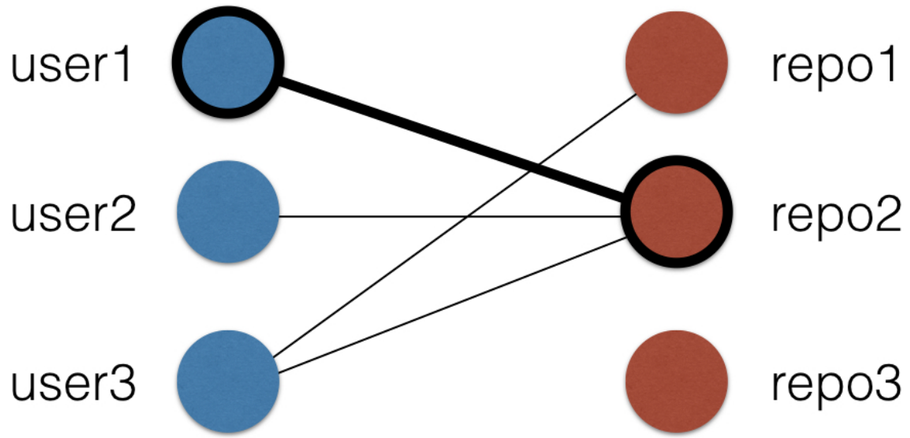
Eric Ma

Data Carpentry instructor and author of nxviz package

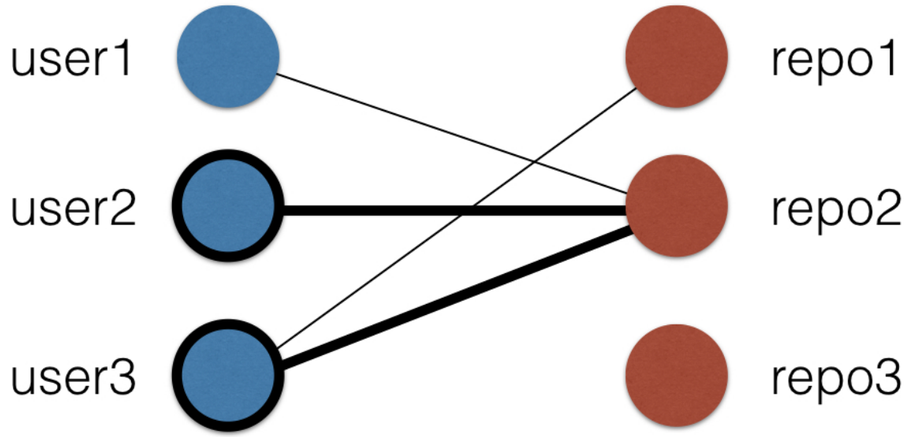
Recommendation systems

- Previously: Recommended users to connect with one another
- Graph: "unipartite" (or users-only) version
- Now: "bipartite" or (repo-users) version
- Recommending repositories for users to work on

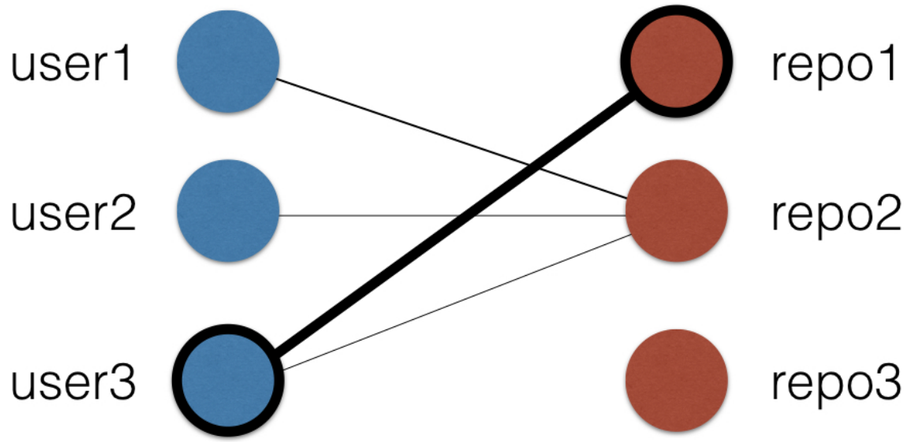
Recommendation systems



Recommendation systems



Recommendation systems



Code: Node sets

```
G.nodes(data=True)
```

```
[('repo3', {'bipartite': 'repositories'}),
 ('repo1', {'bipartite': 'repositories'}),
 ('user1', {'bipartite': 'users'}),
 ('user2', {'bipartite': 'users'}),
 ('repo2', {'bipartite': 'repositories'}),
 ('user3', {'bipartite': 'users'})]
```

```
G.edges()
```

```
[('repo1', 'user3'),
 ('user1', 'repo2'),
 ('user2', 'repo2'),
 ('repo2', ‘user3')]
```

Code: Node sets

```
user1_nbrs = G.neighbors('user1')

user1_nbrs
```

```
['repo2']
```

```
user3_nbrs = G.neighbors('user3')
user3_nbrs
```

```
['repo2', 'repo1']
```

Code: Node sets

```
set(user1_nbrs).intersection(user3_nbrs)
```

```
{'repo2'}
```

```
set(user3_nbrs).difference(user1_nbrs)
```

```
{'repo1'}
```

Let's practice!

NETWORK ANALYSIS IN PYTHON (PART 2)