



Lisbon School  
of Economics  
& Management  
Universidade de Lisboa



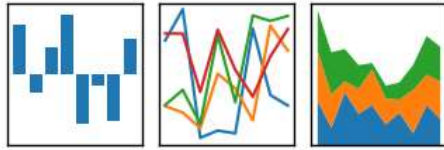
Carlos J. Costa

# NETWORKX



pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$



Python

```
# importar bibliotecas
import numpy as np
import pandas as pd
import networkx as nx
import matplotlib.pyplot as plt
```



matplotlib



# Generating a Network

```
# Generating a network  
G = nx.barabasi_albert_graph(10, 3)  
nx.draw(G, with_labels=True)
```

# Degree

```
# degree of each node link number that each node has
degrees = [deg for node, deg in nx.degree(G)]
print(degrees)
```

Result:

```
[4, 6, 1, 7, 6, 5, 4, 3, 3, 3]
```

# Degree

```
# kmin - minimum degree
kmin = np.min(degrees)

# kmax - maximum degree
kmax = np.max(degrees)

# kavg - average degree
kavg = np.mean(degrees)
```

# Shortest path

```
# shortest path between two nodes  
nx.shortest_path(G,1,2)
```

```
# shortest path between two nodes  
nx.shortest_path(G,1,2, weight=True)
```

# Clustering coefficient

```
# triangles
nx.triangles(G)

# clustering coefficient of a node
nx.clustering(G)

# clustering coefficient of all nodes
# (returns a dictionary)
nx.clustering(G)

# clustering coefficient of the network
cc = nx.clustering(G)
avg_clust = sum(cc.values()) / len(cc)
print(avg_clust)
```

# Centrality

```
# betweenness centrality of network  
nx.betweenness_centrality(G)
```

```
# closeness centrality of network  
nx.closeness_centrality(G)
```

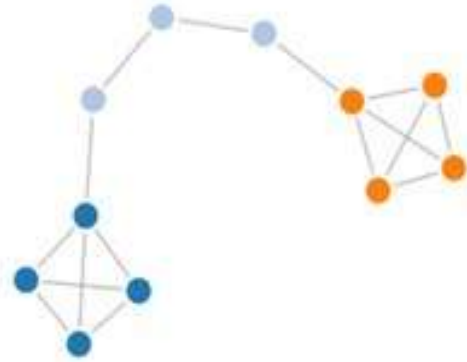
```
# eigenvector centrality of network  
nx.eigenvector_centrality(G)
```

```
# degree centrality  
nx.degree_centrality(G)
```



# Connected Components

```
# find number of connected components
nx.number_connected_components(G)
# get the nodes in the same component as *n*
nx.node_connected_component(G, 3)
# Assortativity
nx.degree_assortativity_coefficient(G)
```



# NetworkX



# Networks

```
# create a network
```

```
G = nx.Graph()
```

```
# add node to a network
```

```
G.add_node('Mary')
```

```
# adding nodes using a list
```

```
G.add_nodes_from(['Mary', 'Steven', 'Alice', 'John'])
```

# Networks

```
# remove nodes
G.remove_node('Mary')

# remove several nodes
G.remove_nodes_from(['Mary', 'Steven'])

# see all nodes
G.nodes
```

# Networks

```
# add several edges (list of tuples)
G.add_edges_from([('Mary', 'Steven') ,
                  ('Mary', 'Alice'), ('Mary', 'John'), ('Mary', 'Edward')])

# view all the edges of a network
G.edges
```

# Networks

```
# add edges
G.add_edge('Mary', 'Steven')

# remove edges
G.remove_edge('Mary', 'Alice')
```

# Networks

```
# number of nodes in a network
G.number_of_nodes()

# number of edges in a network
G.number_of_edges()

# neighbors of a node (result is a dictionary)
neighbors = G.neighbors('Alice')
print (neighbors)

# number of neighbors
G.degree('Alice')
```

# Networks

```
# save network
nx.write_edgelist(G, "partel")

# delete content of a network
G.clear()

# read the content of a network
G = nx.read_edgelist("partel")
```



# Networks

```
#draw network  
nx.draw(G)
```

```
#draw network with labels  
nx.draw(G, with_labels=True)
```

# Networks

```
# set weight to edges
G.add_edge('Mary', 'Steven', weight=500)
G.add_edge('John', 'Mary', weight=10)
G.add_edge('Mary', 'Alice', weight=200)

# get weight from edges
G['Mary']['Steven']

# change the weight of an edge
G['Mary']['Steven']['weight'] = 6
```

# Networks

```
#create a directed network  
dg = nx.DiGraph()
```

```
# not directed representation of a network  
nx.to_undirected(G)
```

```
# directed representation of a network  
nx.to_directed(G)
```

# Networks

```
# multigraphs allows to store several properties
# from the same edges
MG = nx.MultiGraph()
MG.add_weighted_edges_from([(1, 2, 3.0), (1, 2, 75), (2, 3,
5)])

# show edges without weight
MG.edges
```

# Networks

```
# show edge data including eight
MG.edges.data('weight', default=1)

# get the weight of a edge
MG[1][2]

#save network with weighted edges
nx.write_weighted_edgelist(G, "parte2")
```