# Social Data Analysis
# TP2

In this practical work, you will learn some basics on network data analysis. In Python, the 3 most popular librairies for network analysis are `networkx`, `igraph` (also available in `R`), and `graph-tool`. In this practical work, we will illustrate network analysis using the `networkx` library since it is easy-to-use, offers a broad range of functionalities, and offers good performances on relatively large networks. The documentation of the package is available at: `https://networkx.github.io/documentation/stable/`.

Note that the beginning of this tutorial is guided step by step. Its objective is not showing you all the functionalities of this package but showing the most useful instead.

The network we will use today is the *email-EU core dataset*. The network was generated using email data from a large European research institution. Information about all incoming and outgoing email between members of the research institution have been anonymized. There is an edge (u,v) in the network if person u sent person v at least one email. You can find additional information and download the dataset there: `https://snap.stanford.edu/data/email-Eu-core.html`.
**Important note.** In this practical work, you are manipulating non native libraries (`pandas`, `seaborn`, `networkx`, and `scipy`). To import these libraries, download them and install them first. Depending on your Operating System (Windows, Linux, Mac OS X, ...) the installation process differs. Generally, it can be installed using the command

```
pip3 install --user <packageName>
```

Once installed, you must import the library in your code to get all the functionalities provided by the library. To do so, you must copy and paste the following lines at the beginning of your python file.

```python
import pandas as pd
import seaborn as sns
import networkx as nx
from scipy import stats
```

## 1 Validation of the Complex Network Properties

In this section, you will achieve the following steps:

1. Load an external file and map it as a graph object
2. Plot its degree distribution
3. Check if the power-law distribution holds
4. Check if the small-world hypothesis holds
5. Check if the strong community structure holds

### 1.1 Data loading

Once you have downloaded the data, they need to be read in python. Since the graph is represented as a list of edges (i.e., an adjacency list), a particular reading function is needed. Please refer to the documentation to find the appropriate function. Note the the graph is directed. Once loaded, it is possible to access to vertices as follows.

```python
g = <the appropriate read function>

# Get the list of vertices
listVertices = g.nodes


#Get the attributes values of the first vertex
attribute_first = g.nodes[0]

#Get the value of the label attribute of the first vertex
attribute_first = g.nodes[0]["label"]

#Get the value of the attribute label for the first edge
list_label = g.edges[0]["label"]
```

## 1.2 Plotting degree distribution

Plotting functionalities are always useful to get a better understanding of the data. In this section, we will focus on plotting the degree distribution of the network we are analyzing. We will then check if the distribution follows a power-law distribution as it is the case in almost every complex networks. Since we are considering a directed network, in the following we will consider the degree as the sum of in and out degrees. For doing so, please execute these following steps.

1. Get the degree distribution using the `degree` function;
2. Transform the obtained list into a `pandas` series using the `Series` constructor (see `http://pandas.pydata.org/pandas-docs/stable/generated/pandas.Series.html`). The `pandas` library offers very convenient functionalities, e.g., discretization, sampling, or percentile calculation, in manipulating series (lists) and data frame (matrices);
3. Use the `distplot` function of the `seaborn` library to plot the degree distribution. You must specify the data, use 20 bins, add a label, and do not plot the gaussian kernel density estimate (see `http://seaborn.pydata.org/generated/seaborn.distplot.html#seaborn.distplot`).
4. Display the plot with the command `sns.plt.show()`;
5. Modify the call to the function `distplot` to check if the distribution follows a power-law distribution.

Repeat the same methodology with in-degree and out-degree separately.

## 1.3 Small-world hypothesis

In this section, we will check if the small-world hypothesis holds on this dataset. To do so, we will calculate the shortest path length between pairs of vertices and average these lengths. Of course, it is not realistic to calculate the shortest path for every pairs of vertices. We will thus consider only a subset of vertices. This subset contains 200 randomly chosen vertices. The sampling can be performed with the `sample` function of the series object. It is then needed to iterate over all possible pairs and calculate the shortest paths. Iterating over all possible pairs can be done using a nested loop whereas calculating the shortest path length can be performed by the `shortest_path_length` function associated to the graph instance.

**Instructions.** Calculate the average shortest path length of 200 randomly chosen vertices. In case the shortest path equals `math.inf` you will not consider it.

### 1.4 Strong community structure

The third characteristic of complex networks is that they exhibit a strong community structure. A typical methodology for checking if this characteristic is to calculate the average local clustering coefficient as seen in the lecture. This average local clustering coefficient is then compared to the average local clustering coefficient of a random having the same numbers of vertices and edges.
**Instructions.** Apply the above mentioned strategy to check if the strong community structure property holds on this network.

**Tips.** Here are some practical information to help you to reach your objective:

- A random graph can be generated using the `gnm_random_graph` function;
- The average local clustering coefficient can be calculated with the function `average_clustering`. Note that this function is dedicated to undirected graph. However, it is of little importance since both graph (the original one and the random one) are both considered as undirected.

## 2 On the calculation of node-level indices

We now calculate some topological measures on the vertices. Read on the documentation of the package `networkx` and calculate for each node[1]:

- Its degree centrality
- Its closeness centrality
- Its betweeness centrality
- Its eigenvector centrality

[1]Depending on the available resources on your computer, you can randomly select 200 vertices and calculate the metrics on this subset only.

- Its Page Rank
- its Hub and Authority scores

For each centrality measure, find the 10 top nodes and compare them. Are they similar?

## 3 On the estimation of the strength of links

Apply the techniques of *shortcut bridges* and *neighbourhood overlap* to find the 10 strongest and the 10 weakest links per technique.

## 4 Community detection

Apply the different community detection algorithms that are available in the library and compare the result with the ground truth. The ground truth can be downloaded on the website where you found the dataset. The quality of the clustering will be evaluated using the NMI metric. This metric has been implemented in the `scikit` library. To use it, simply write:

```python
from sklearn.metrics.cluster import normalized_mutual_info_score
    as mi

nmi = mi([1,0],[1,0])
```