

# **Estruturas de Dados para Bioinformática**

Hugo Pacheco

DCC/FCUP  
24/25

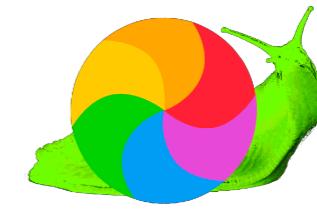
# Visualização de grafos

# Grafos

- Umas aulas atrás (*NetworkX*):
  - Representar redes de dados interligados como grafos
  - Utilização de algoritmos de grafos
- Esta aula (*NetworkX + Graphviz*):
  - Visualização de grafos
  - Mais exemplos

# Grafos

- Visualização de grafos é um problema complexo
  - Minimizar sobreposições, maximizar distâncias, etc
- Vamos explorar visualização de grafos em Python
  - Funcionalidades base do *NetworkX*
  - Integração com *Graphviz*
- Não vamos ver ferramentas mais completas de visualização de grafos
  - E.g., *Cytoscape*, *Gephi*



# NetworkX (draw)

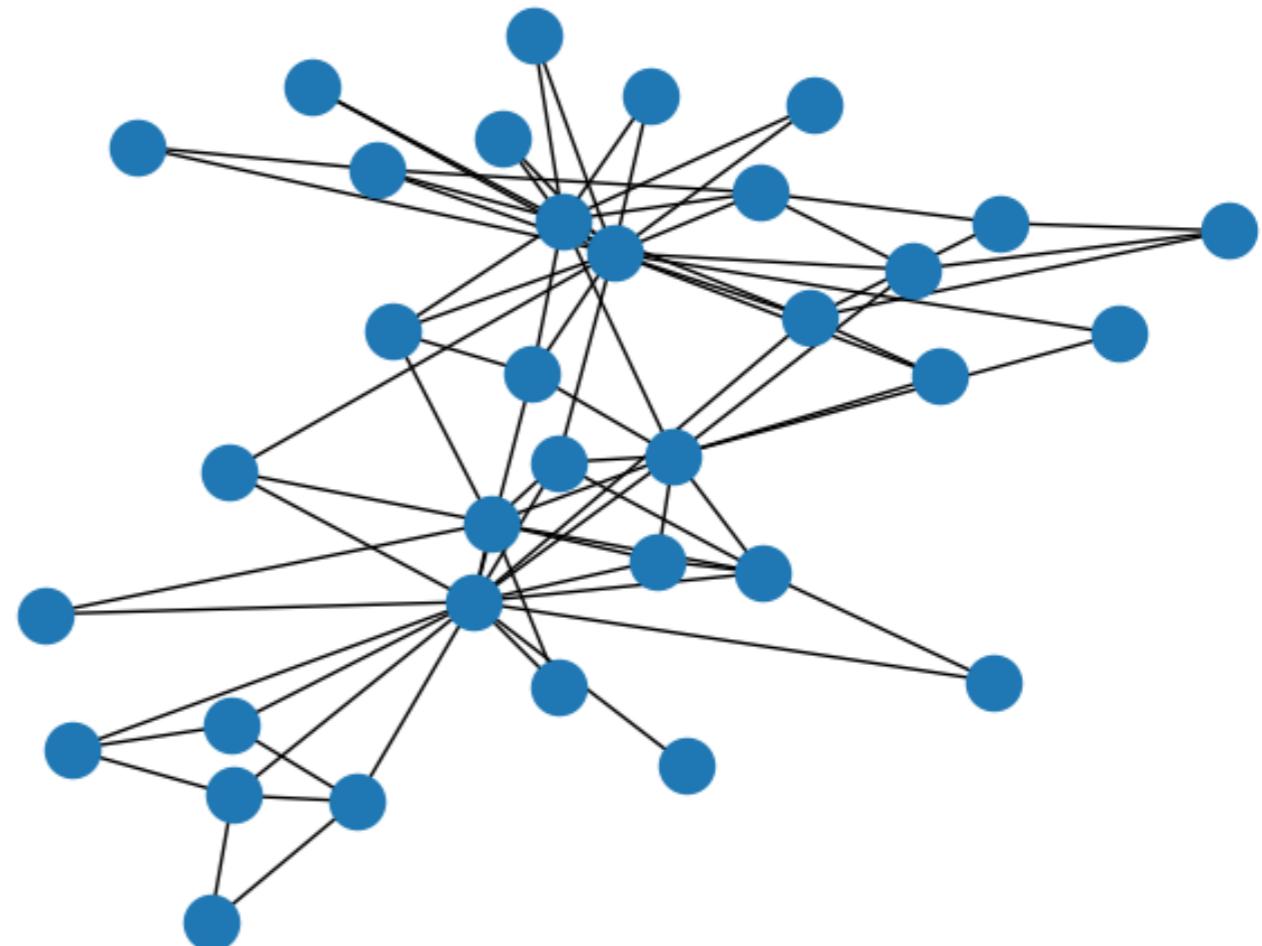
- Visualizar um grafo de exemplo



Se redesenhar o grafo, as posição mudam!

```
import networkx as nx
import matplotlib.pyplot as plt

g = nx.karate_club_graph()
print(g.nodes(data=True))
# [(0, {'club': 'Mr. Hi'}), ...]
print(g.edges(data=True))
# [(0, 1, {'weight': 4}), ...]
nx.draw(g)
plt.show()
```



# NetworkX (layout)

- Como é que o NetworkX decide como desenhar o grafo?
  - Utiliza um *layout*, e.g., um dicionário { *nodo* : *Posição2D* }
- O default é o *spring layout*, que tende a desenhar nodos mais conectados mais próximos uns dos outros

```
l = nx.spring_layout(g)
print(l)
# {0: array([-0.24019787,
-0.34623371]), ...}
l = nx.spring_layout(g)
print(l)
# {0: array([-0.07914912,
0.46013368]), ...}

nx.draw(g, pos=l)
```

## ≡ Spring system

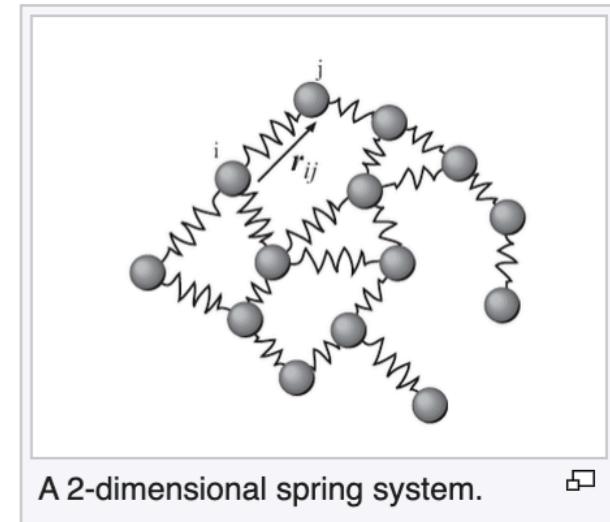
Article Talk

Add languages ▾

Tools ▾

From Wikipedia, the free encyclopedia

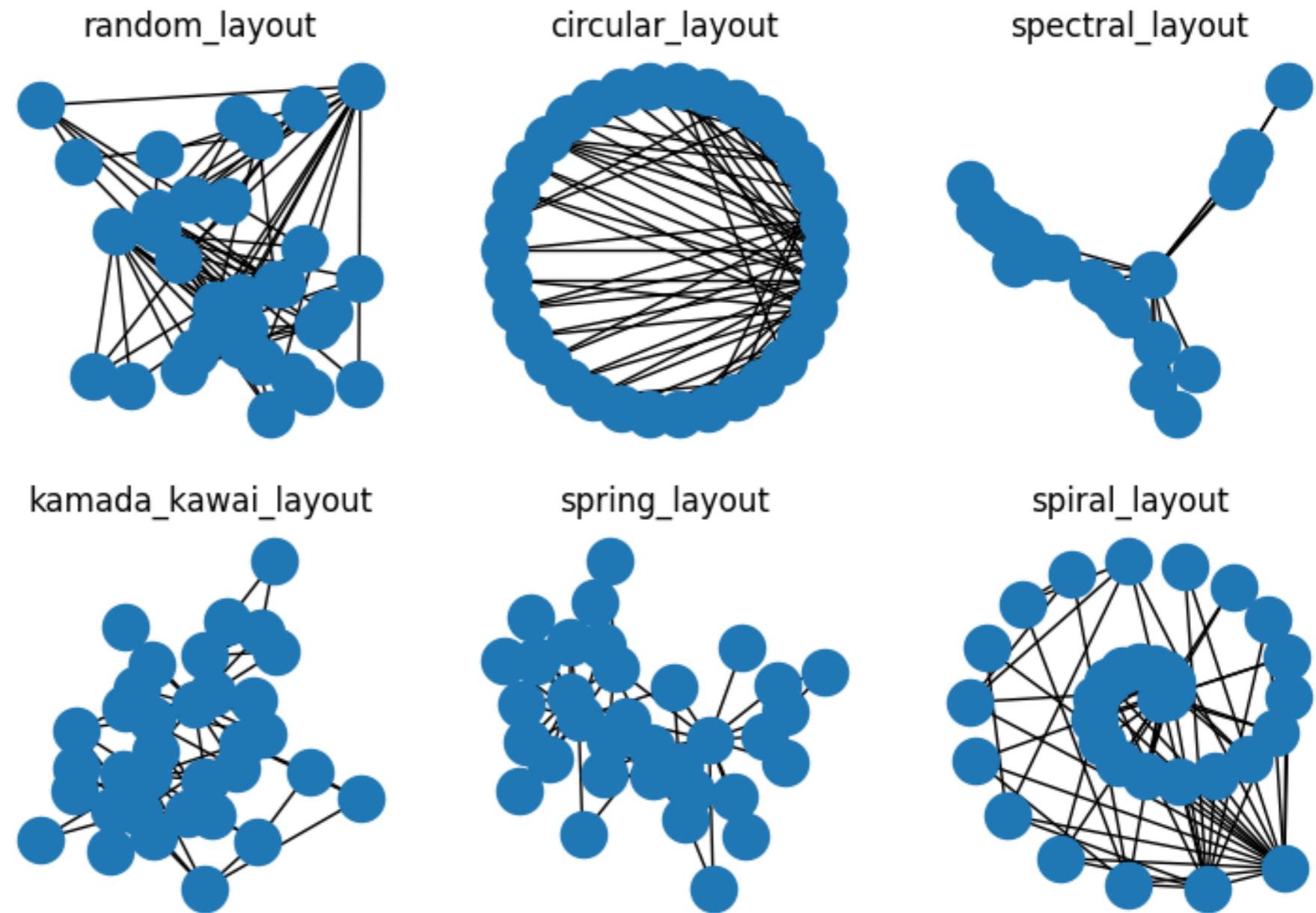
In engineering and physics, a **spring system** or **spring network** is a model of physics described as a **graph** with a position at each vertex and a **spring** of given stiffness and length along each edge. This generalizes **Hooke's law** to higher dimensions. This simple model can be used to solve the



A 2-dimensional spring system.

# NetworkX (layout)

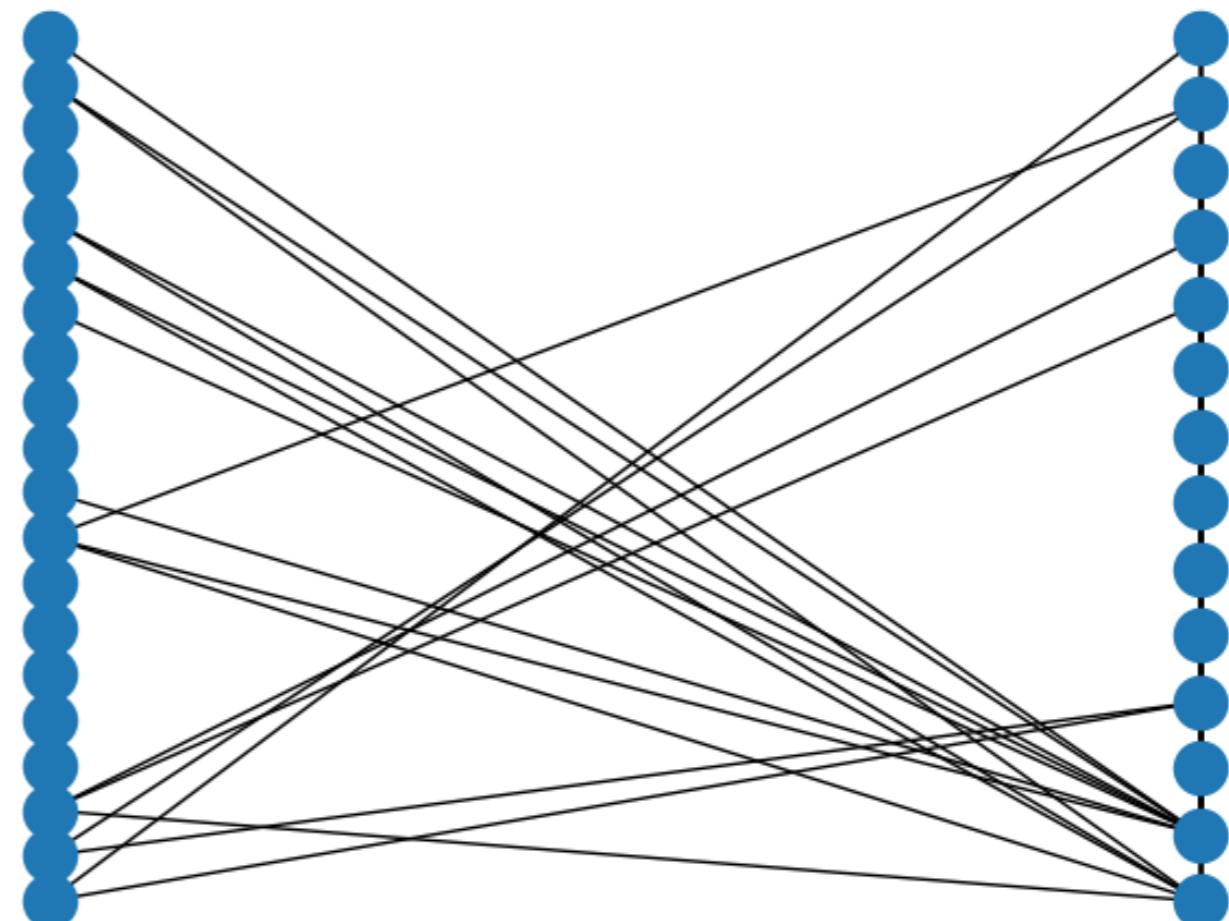
- Há vários *layouts* disponíveis (documentação)



# NetworkX (layout)

- O *bipartite\_layout* desenha o grafo de acordo com duas partições
  - Temos que passar os nodos que constituem uma delas
  - Podemos alinhar horizontalmente ou verticalmente

```
g = nx.karate_club_graph()  
  
ns = [n for n in g.nodes()  
if n < 20]  
nx.draw(g, pos=nx.bipartite  
_layout(g, ns))
```



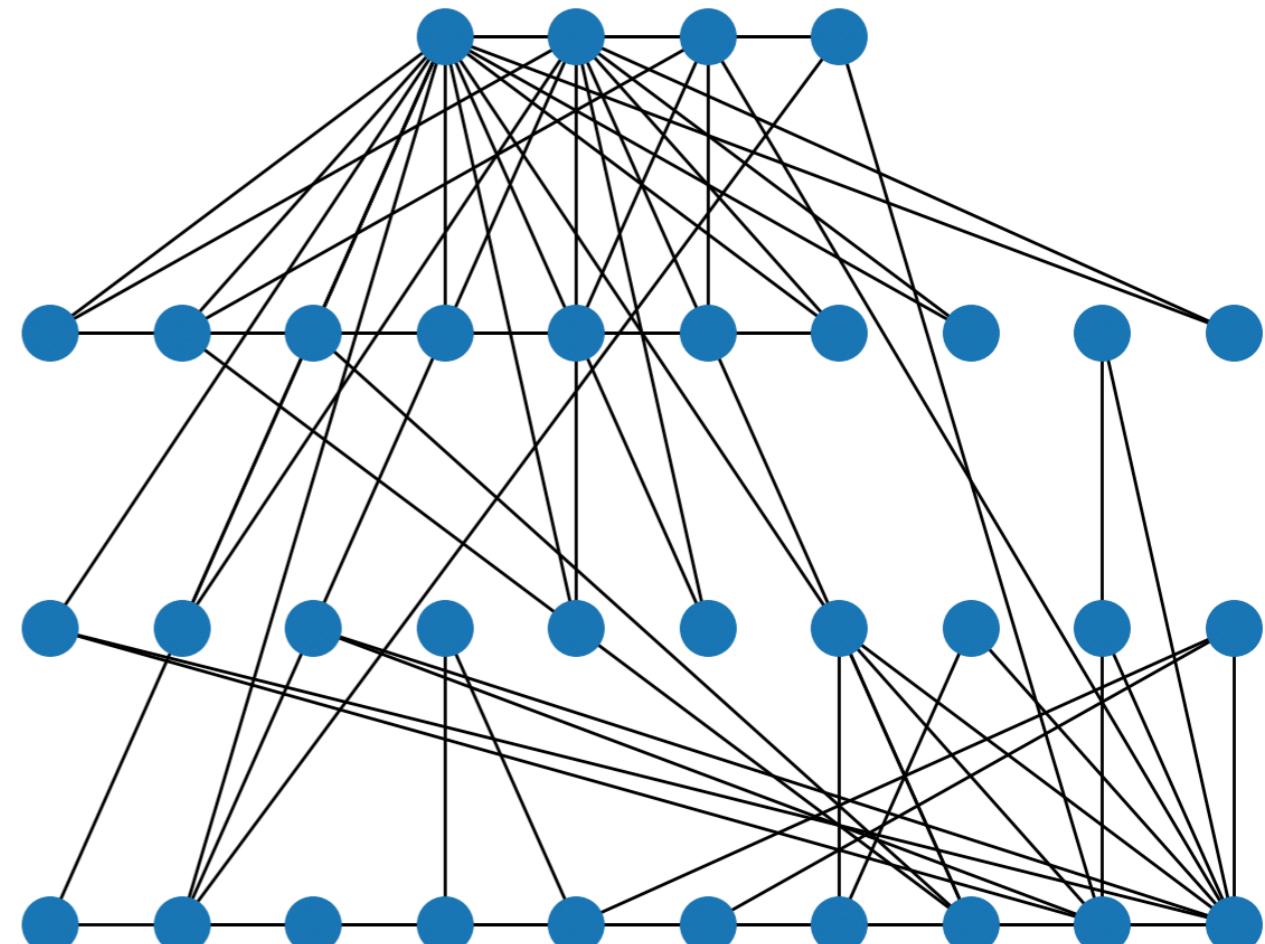
# NetworkX (layout)

- O *multipartite\_layout* desenha o grafo de acordo com várias partições
  - Um atributo define o nível de cada nodo
  - Podemos alinhar horizontalmente ou verticalmente

```
g = nx.karate_club_graph()

for n in g.nodes():
    g.nodes[n]["layer"] = n //
10

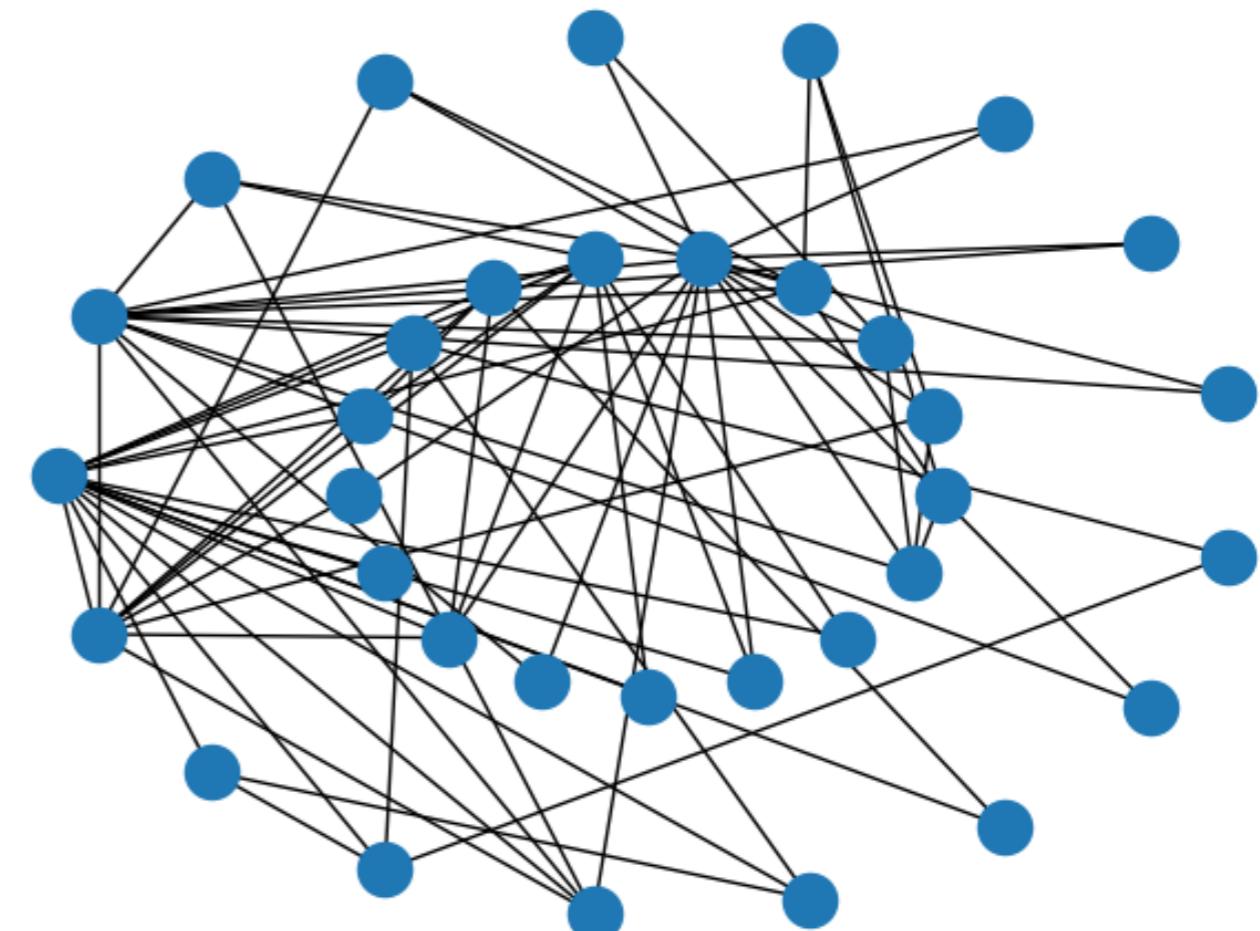
nx.draw(g, pos=nx.multipartite
         _layout(g, subset_key="layer",
         align="horizontal"))
```



# NetworkX (layout)

- O *shell\_layout* desenha o grafo por níveis
  - Temos que passar uma lista de níveis, em que um nível é uma lista de nodos

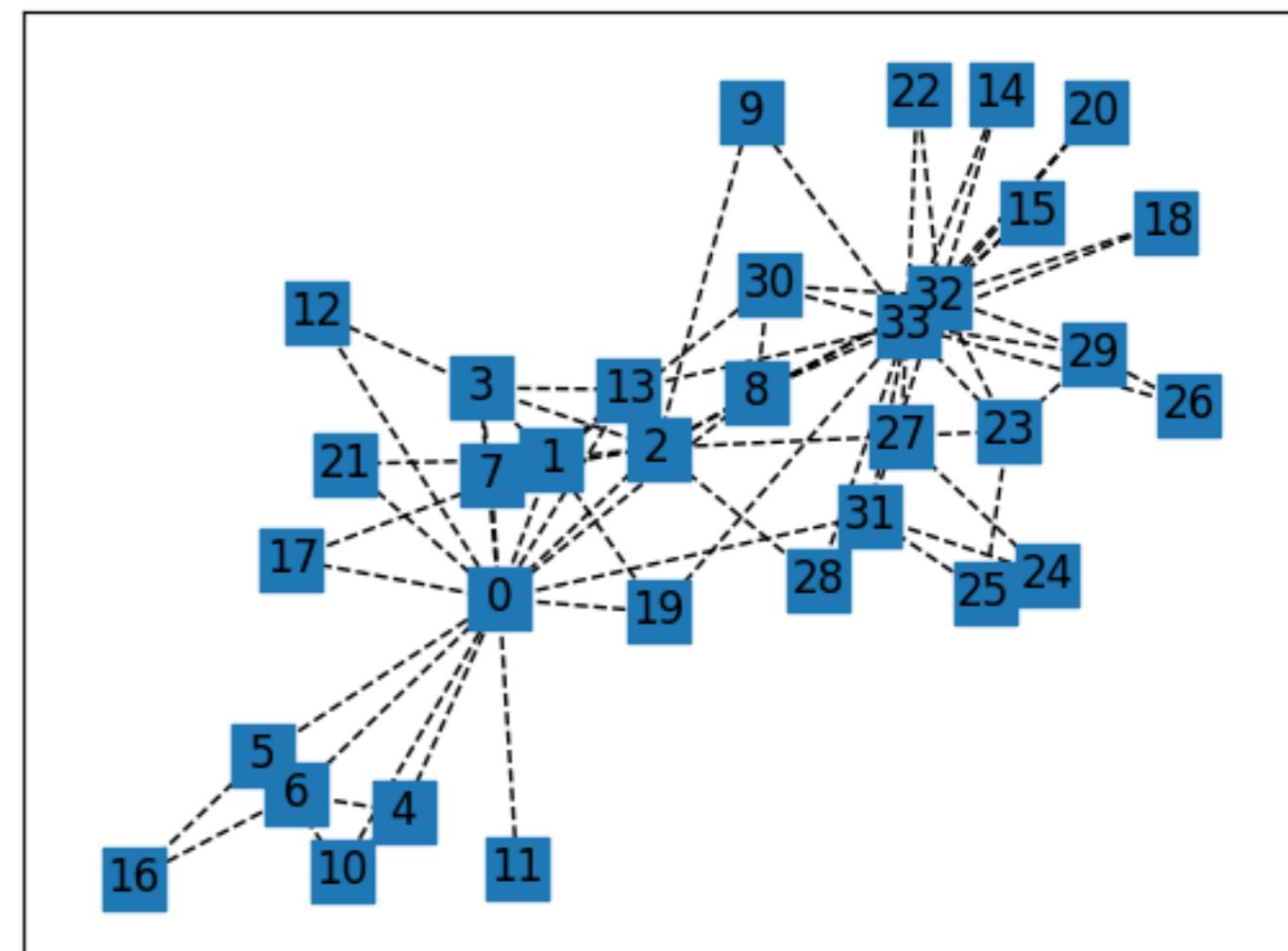
```
g = nx.karate_club_graph()  
  
ns = [[n for n in  
g.nodes() if n % i == 0]  
for i in range(1,3)]  
nx.draw(g, pos=nx.shell_layout(g, ns))
```



# NetworkX (*draw\_networkx*)

- Podemos costumizar melhor a visualização
  - e.g., formato de nodos, formato de arestas

```
g = nx.karate_club_graph()  
  
nx.draw_networkx(  
    g  
    ,node_shape='s'  
    ,style="--")  
plt.show()
```

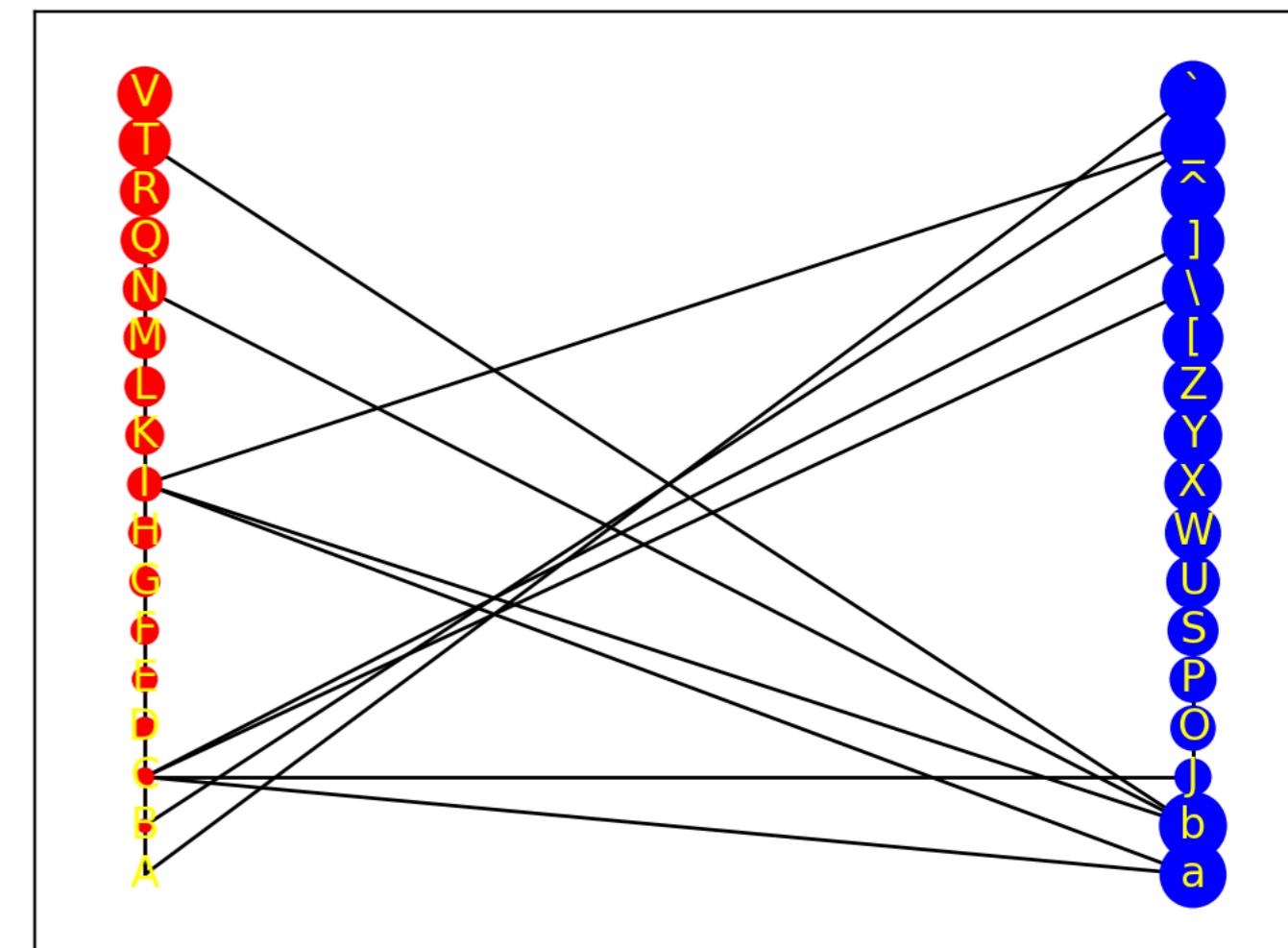


# NetworkX (`draw_networkx`)

- Podemos costumizar melhor a visualização (por nodo)
  - e.g., labels, tamanho, cor

```

g = nx.karate_club_graph()
ns = g.nodes(data=True)
color = { 'Mr. Hi': 'red',
          'Officer': 'blue' }
his = [n for n,i in ns
       if i['club']=='Mr. Hi']
colors = [ color[i['club']] for n,i in ns ]
ls = { n : chr(n+65) for n,i in ns }
szs = [10 * n for n in g.nodes()]
nx.draw_networkx(g, pos=nx.bipartite_layout(g, his), labels=ls, node_size=szs, node_color=colors, font_color='yellow')
    
```



# NetworkX (draw\_networkx)

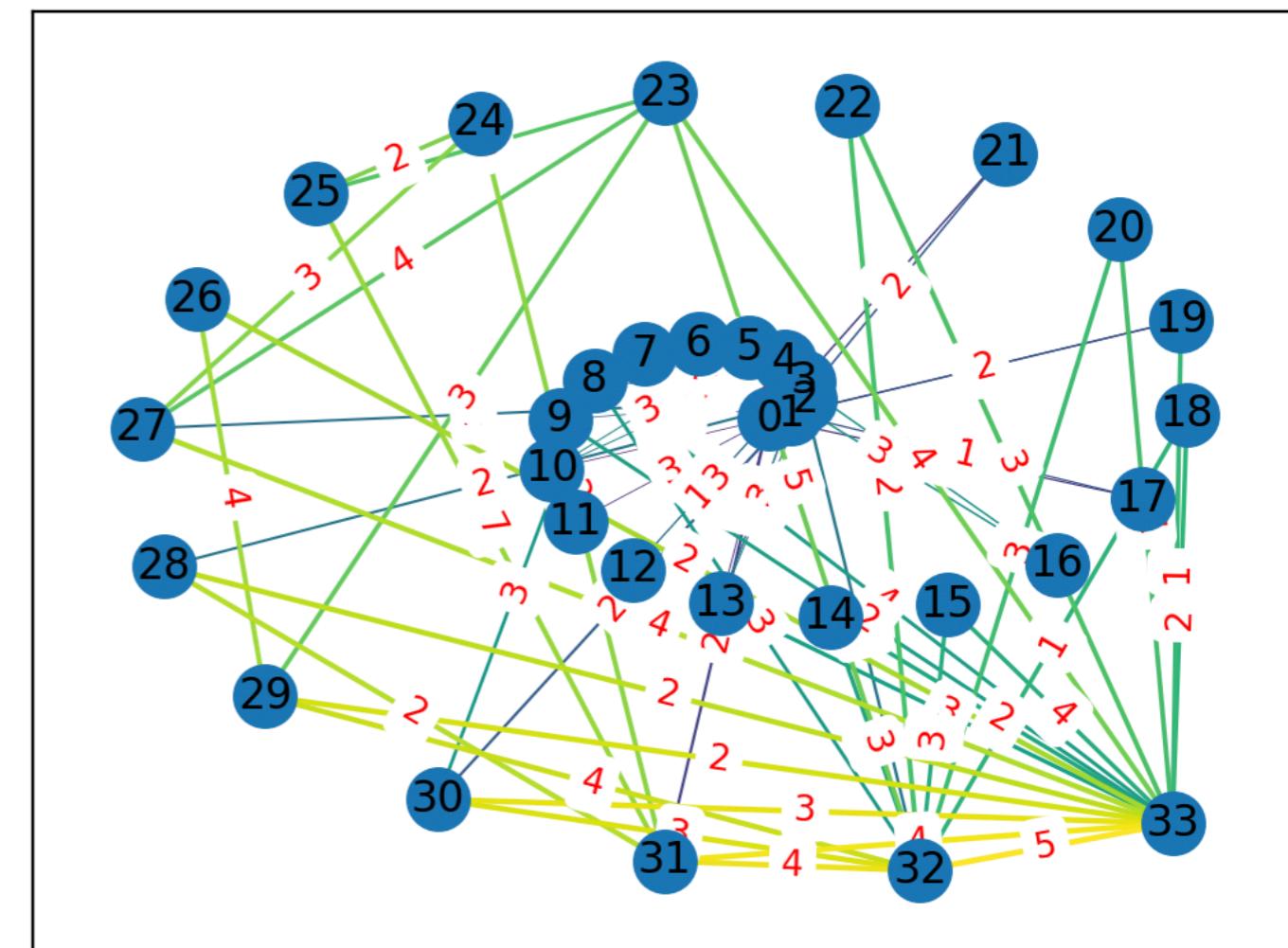
- Podemos costumizar melhor a visualização (por aresta)

- e.g., labels, grossura, cor

```

g = nx.karate_club_graph()
es = g.edges(data=True)
ws = [ (i+j)/40
      for i,j,d in es ]
cs = [ i for i
      in range(len(es)) ]
ls = { (i,j) : d['weight']
      for i,j,d in es }
ps = nx.spiral_layout(g)
nx.draw_networkx(g, pos=ps, width=ws, edge_color=cs)
nx.draw_networkx_edge_labels(g, pos=ps, edge_labels=ls, font_color='red')

```



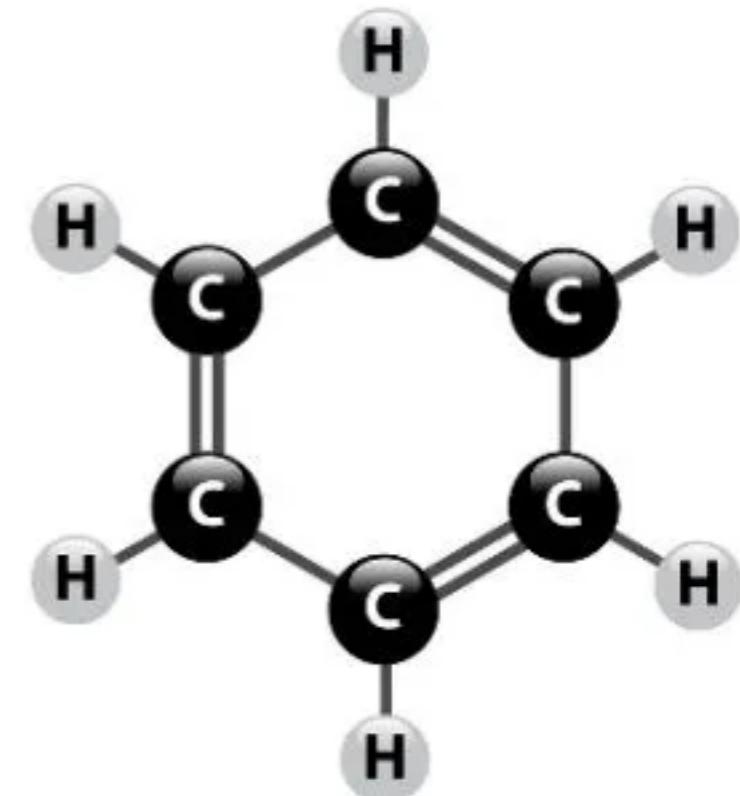
# Exemplo (Molécula)

- Desenhar uma molécula no formato MOL2 (mais detalhes), e.g., benzeno
  - nodos = átomos; arestas = ligações

```

5 @<TRIPOS>MOLECULE
6 benzene
7 12 12 0 0 0
8 SMALL
9 NO_CHARGES
10 ****
11 Any comment about the molecule goes here. No need for a pound sign here
12
13 @<TRIPOS>ATOM
14     1 C      -0.7600    1.1691   -0.0005 C.ar    1 BENZENE
15     2 C       0.6329    1.2447   -0.0012 C.ar    1 BENZENE
16     3 C      1.3947    0.0765    0.0004 C.ar    1 BENZENE
17     4 C      0.7641   -1.1677    0.0027 C.ar    1 BENZENE
18     5 C     -0.6288   -1.2432    0.0001 C.ar    1 BENZENE
19     6 C     -1.3907   -0.0751   -0.0015 C.ar    1 BENZENE
20     7 H     -1.3536    2.0792    0.0005 H      1 BENZENE
21     8 H      1.1243    2.2140   -0.0028 H      1 BENZENE
22     9 H      2.4799    0.1355   -0.0000 H      1 BENZENE
23    10 H      1.3576   -2.0778    0.0063 H      1 BENZENE
24    11 H     -1.1202   -2.2126   -0.0005 H      1 BENZENE
25    12 H     -2.4759   -0.1340   -0.0035 H      1 BENZENE
26 @<TRIPOS>BOND
27     1     1     2   ar
28     2     2     3   ar

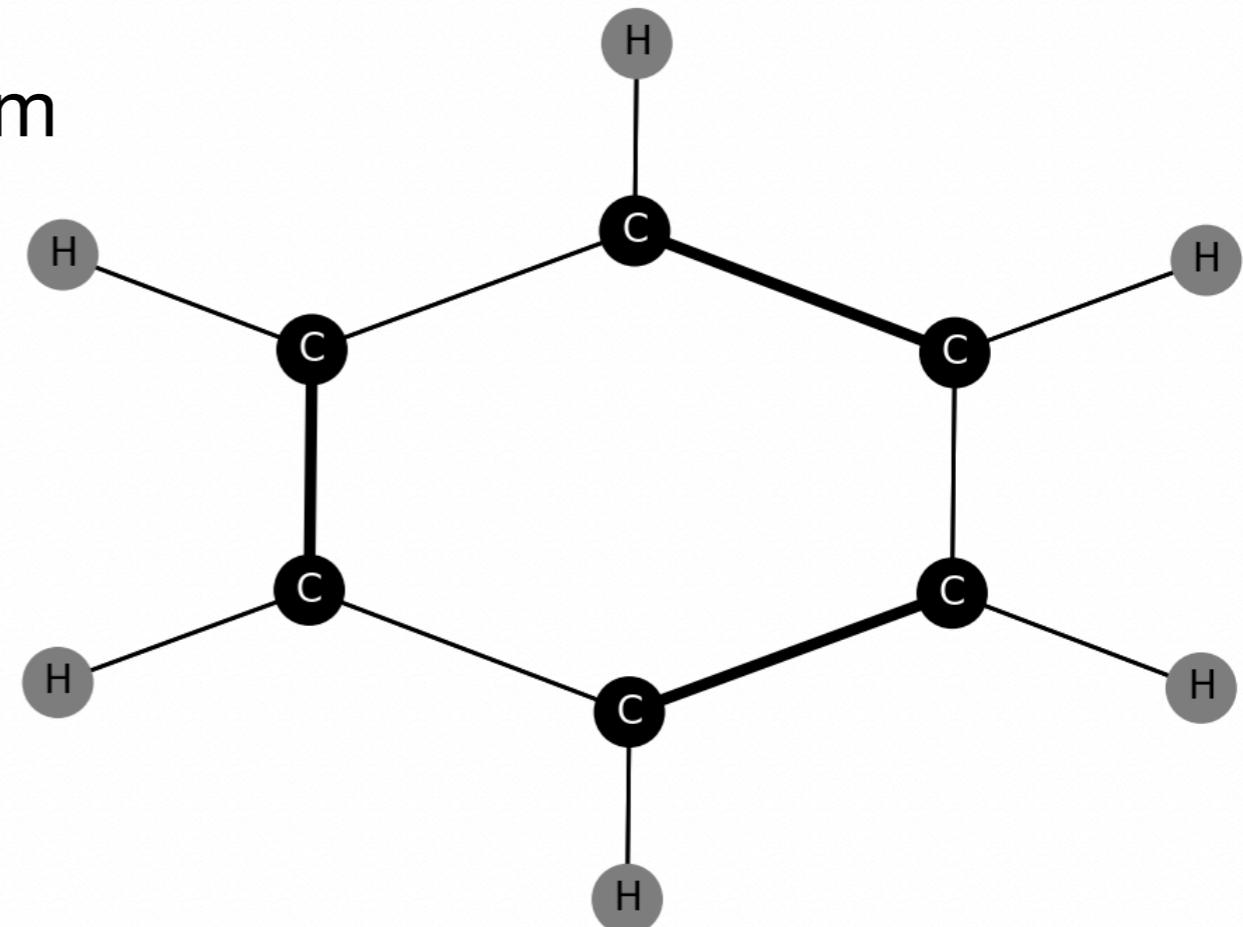
```



**Benzeno**  
C<sub>6</sub>H<sub>6</sub>

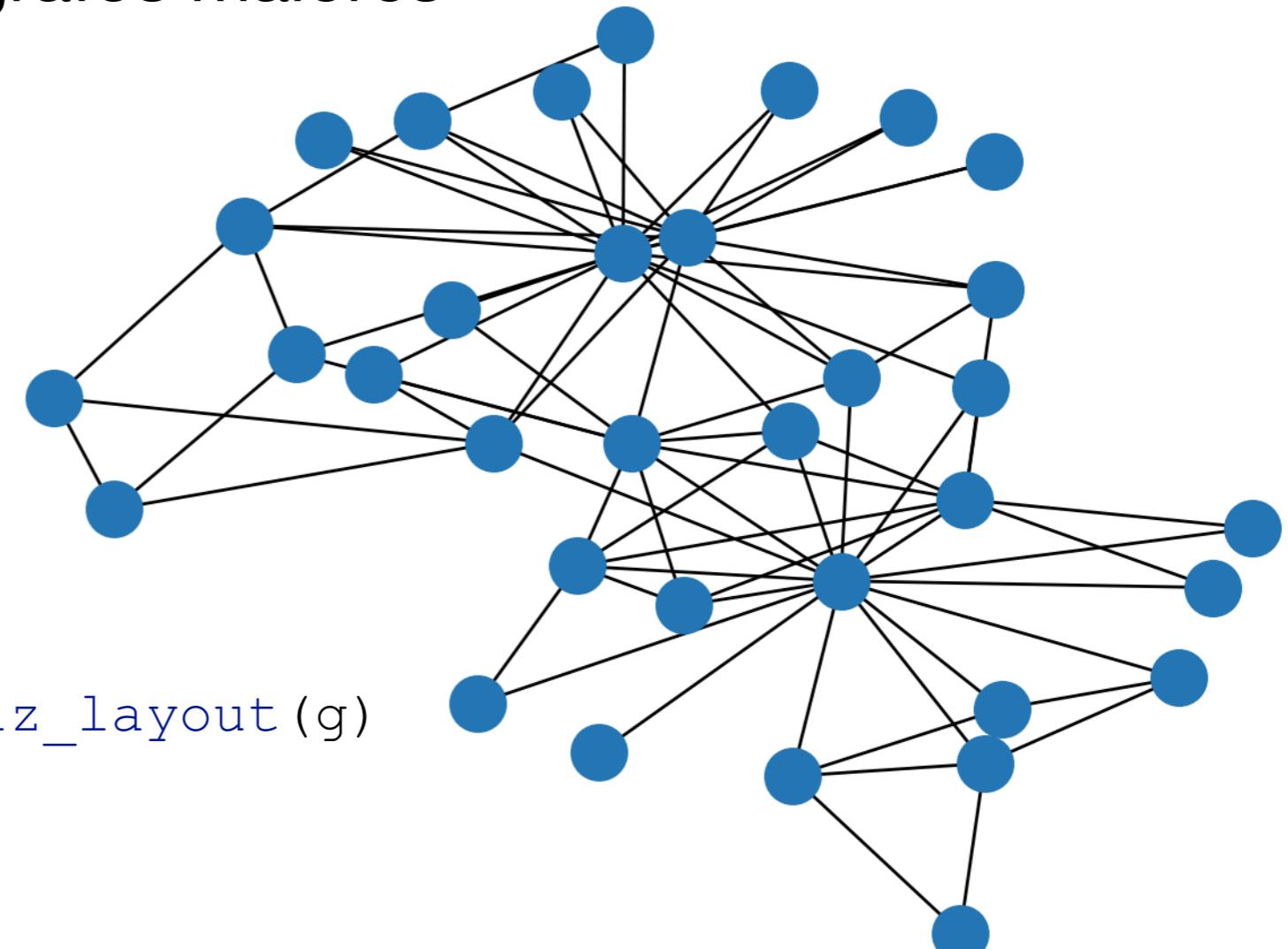
# Exemplo (Molécula)

1. Ler ficheiro MOL2 retirado [daqui](#)
2. Converter <TRIPOS>ATOM num DataFrame
3. Converter <TRIPOS>BOND num DataFrame
4. Juntar ambos num grafo NetworkX
5. Desenhar nodos e arestas com layout fixo



# NetworkX ← Graphviz

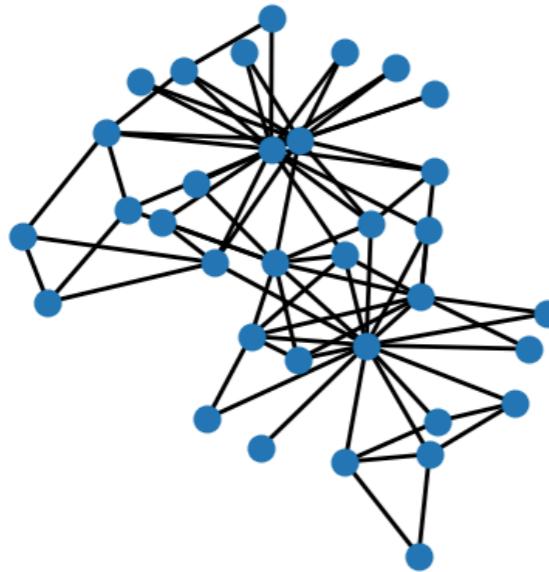
- Podemos adicionalmente utilizar layouts do Graphviz (via biblioteca *pygraphviz*)
- Particularmente útil para grafos maiores



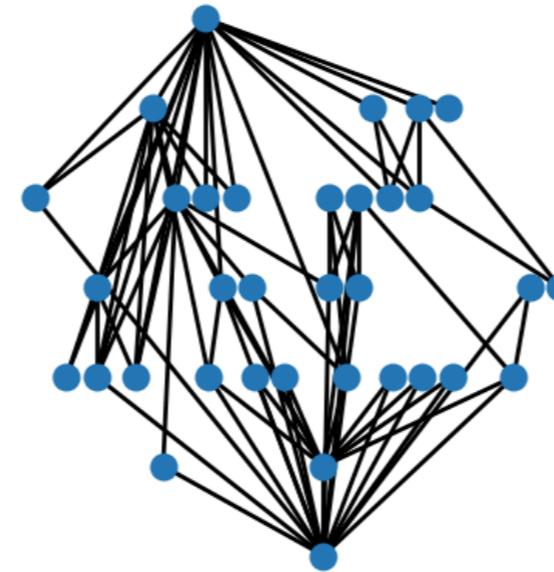
```
g = nx.karate_club_graph()  
l = nx.nx_agraph.pygraphviz_layout(g)  
nx.draw(g, pos=l)
```

# NetworkX ← Graphviz

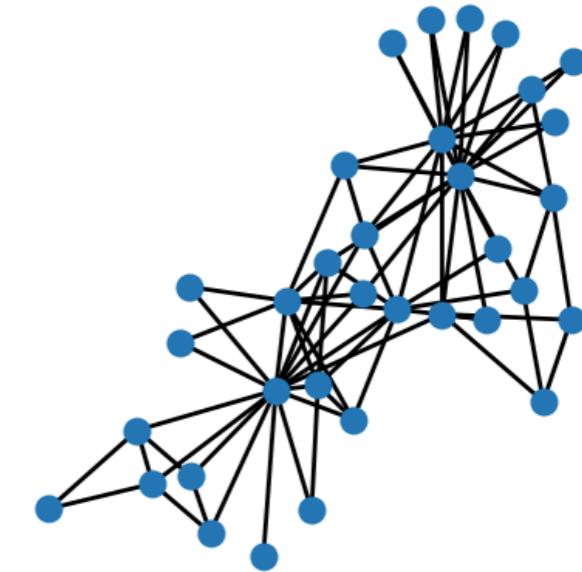
neato (default)



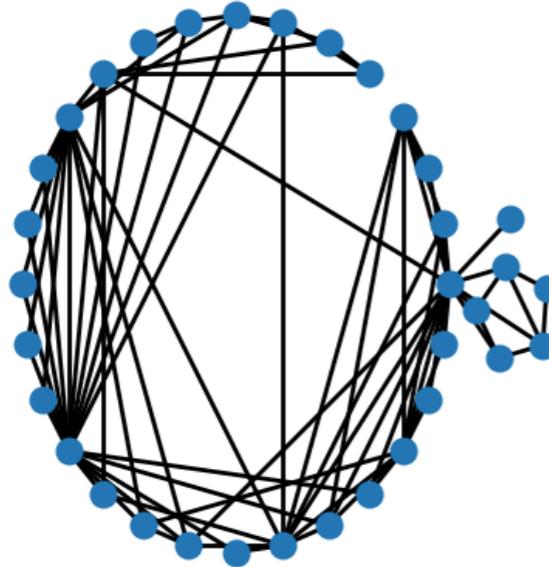
dot



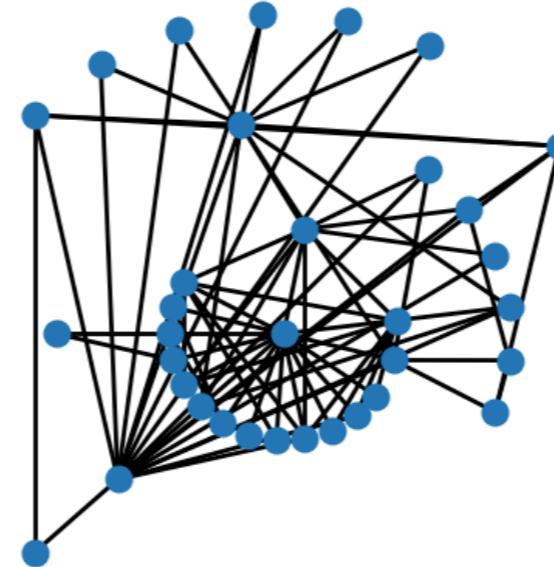
fdp



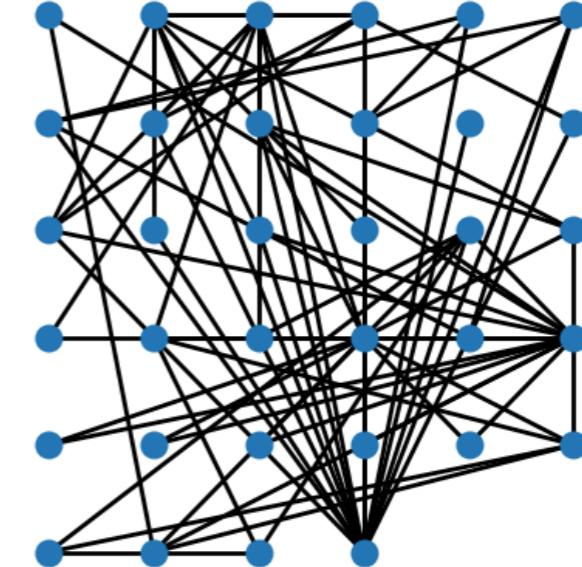
circo



twopi



osage



# NetworkX → Graphviz

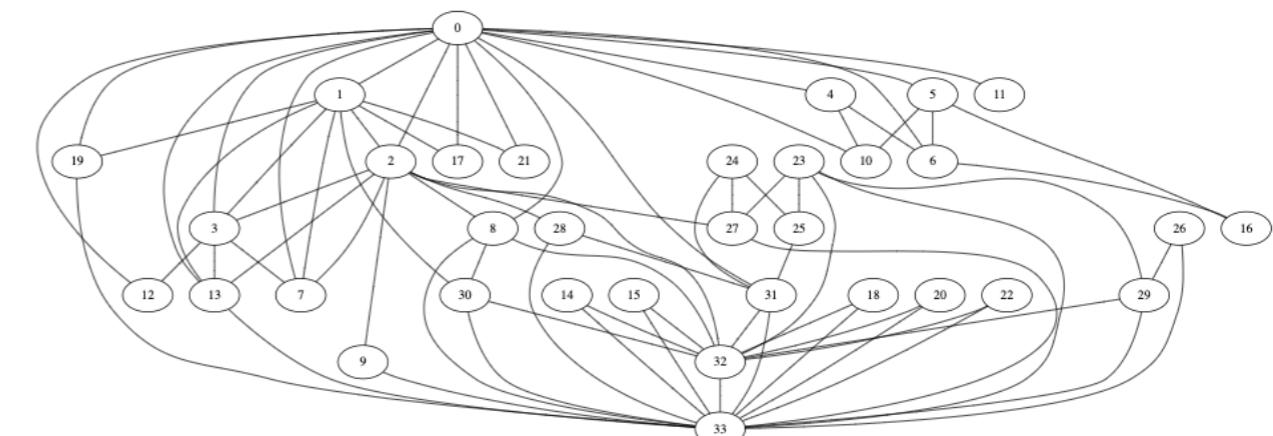
- Podemos converter um grafo no formato Graphviz, e.g., exportar para ficheiro DOT
- Particularmente útil para processamento adicional, e.g., <https://edotor.net/>

```
g = nx.karate_club_graph()
ag = nx.nx_agraph.to_agraph(ag)
ag.write('karate.dot')
```

Edotor   Load Sample ▾   Download ▾   Engine: dot ▾

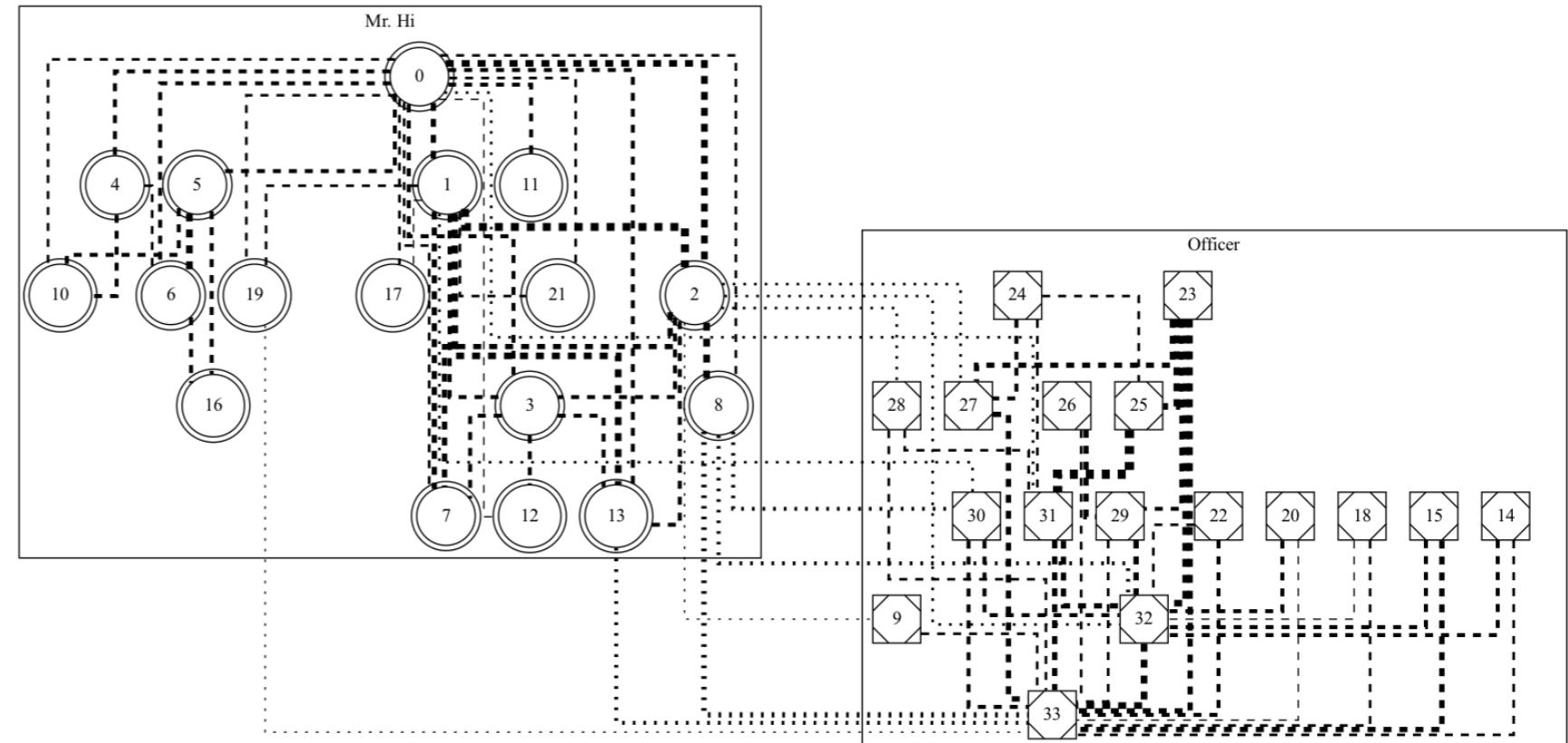
[Graphviz Documentation ↗](#)   [Issues ↗](#)   [Copy Share Link](#)

```
1 strict graph "Zachary's Karate Club" {
2     graph [name="Zachary's Karate Club"];
3     0 [club="Mr. Hi"];
4     1 [club="Mr. Hi"];
5     0 -- 1 [weight=4];
6     2 [club="Mr. Hi"];
7     0 -- 2 [weight=5];
8     3 [club="Mr. Hi"];
9     0 -- 3 [weight=3];
10    4 [club="Mr. Hi"];
11    0 -- 4 [weight=3];
12    5 [club="Mr. Hi"];
13    0 -- 5 [weight=3];
14    6 [club="Mr. Hi"];
15    0 -- 6 [weight=3];
16    7 [club="Mr. Hi"];
17    0 -- 7 [weight=2];
```



# NetworkX → Graphviz

- O Graphviz é bastante mais configurável ([documentação](#))
- Pygraphviz = interface programática
- Mais features:
  - clusters
  - formatação
  - layout
  - etc



# Visualização web (gráficos)

# Gráficos Web

- Últimas aulas: Gráficos e mapas offline
- Existem muitas bibliotecas para gráficos e mapas web
  - Mais modernas e interativas
  - Menos bem documentadas
  - Mais nuances tecnológicas (servidor/cliente, JavaScript, etc)
- Esta aula: Gráficos e mapas web em *plotly*, *folium* e *Bokeh*



Animações



Interatividade (e.g. botões)

# *Plotly* (bar)

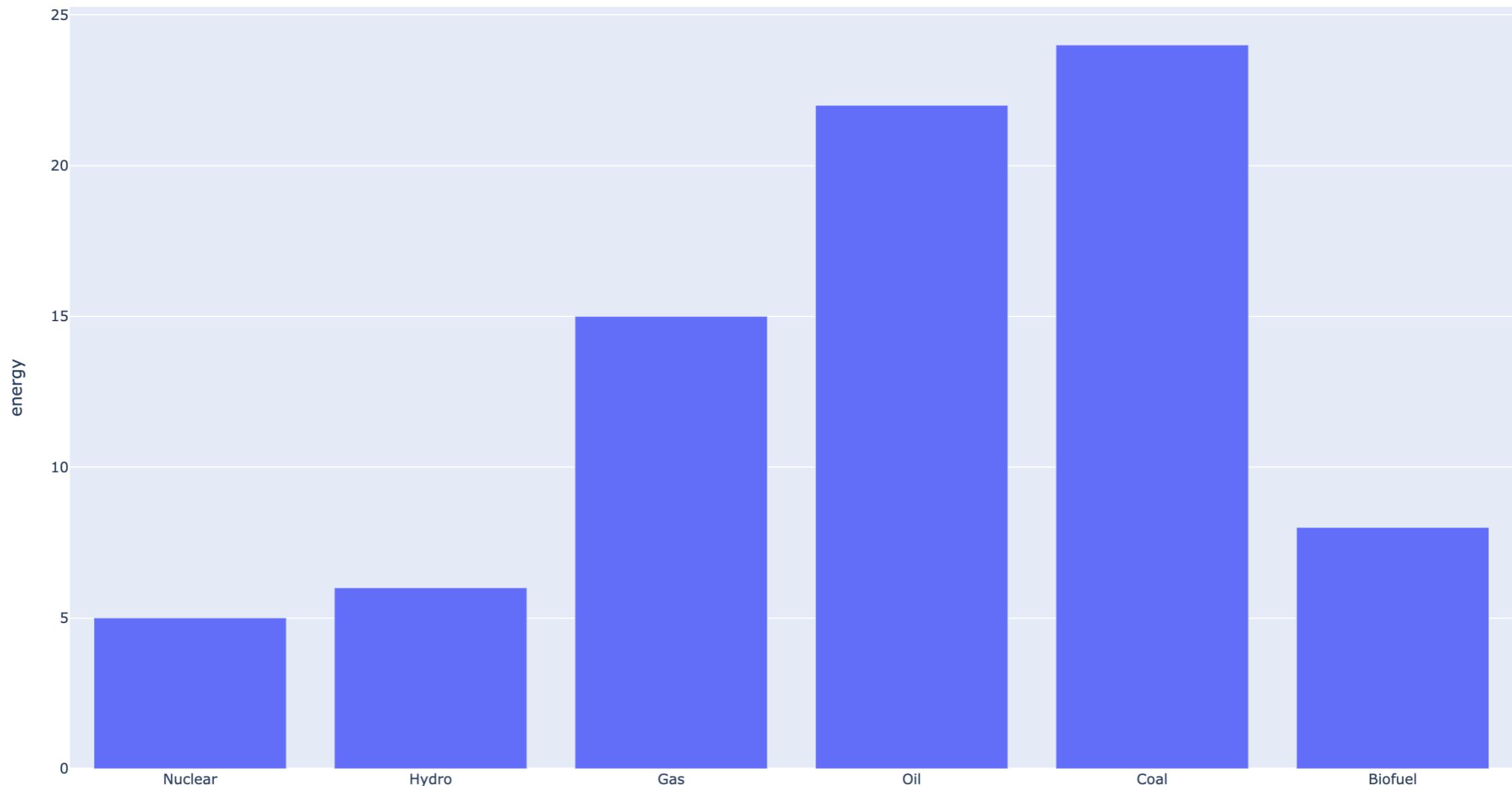
- Como desenhar um gráfico (de barras):
  1. Organizar dados num só DataFrame
    - Coluna X: eixo dos X
    - Coluna Y: eixo dos Y
  2. Uma só chamada para gerar ficheiro HTML

```
import plotly.express as px

fig = px.bar(df, x="x", y="y")
fig.write_html("file.html")
```

# Plotly (bar)

- E.g., distribuição de energia por fontes



# *Plotly (bars)*

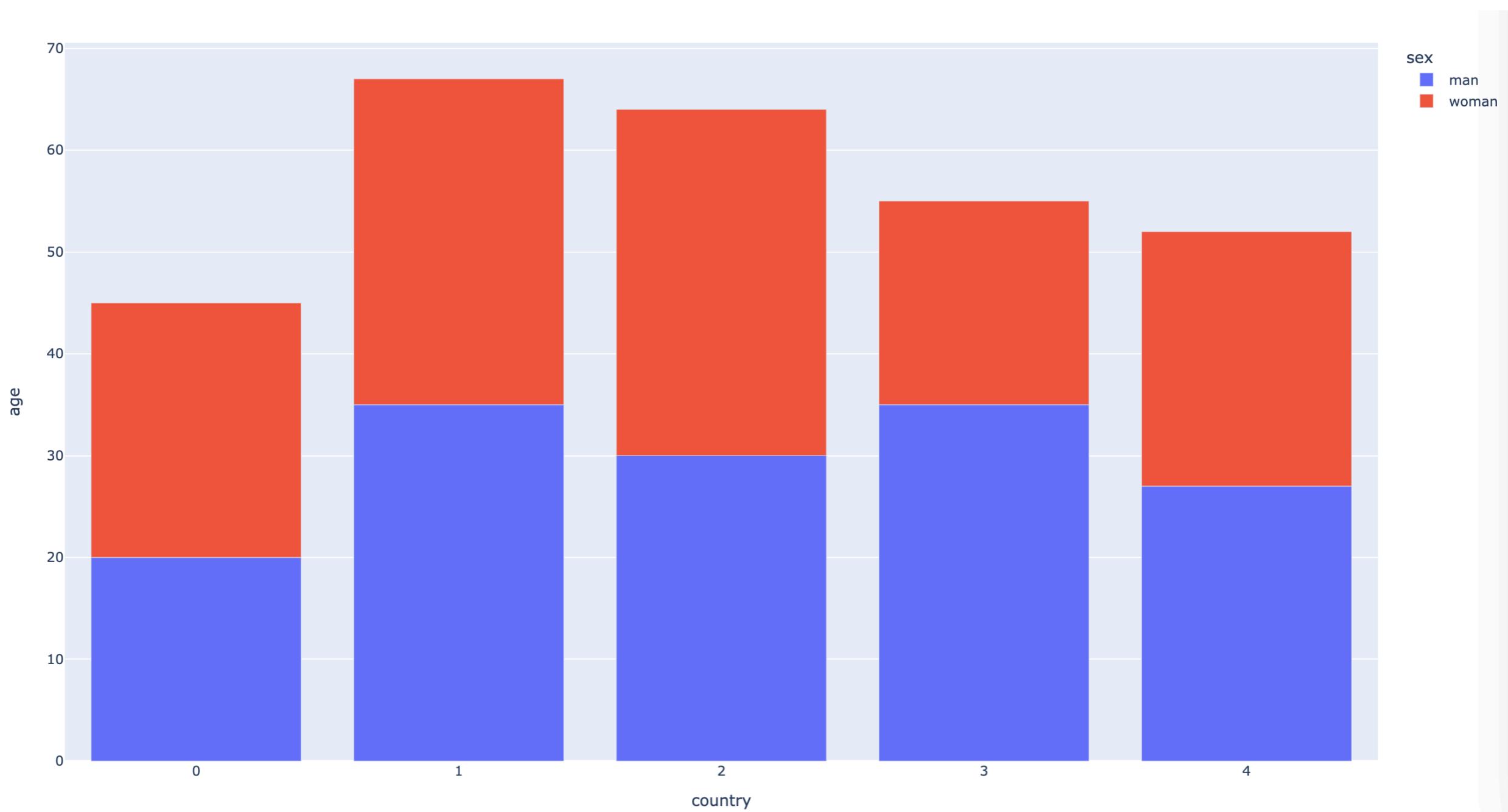
- Como desenhar um gráfico (com múltiplas barras):
  1. Organizar dados num só DataFrame
    - Coluna X: eixo dos X
    - Coluna Y: eixo dos Y
    - Coluna Color: qual a categoria/cor a que os dados (X,Y) pertencem
  2. Uma só chamada para gerar ficheiro HTML

```
import plotly.express as px

fig = px.bar(df, x="X", y="Y", color='Color')
fig.write_html("file.html")
```

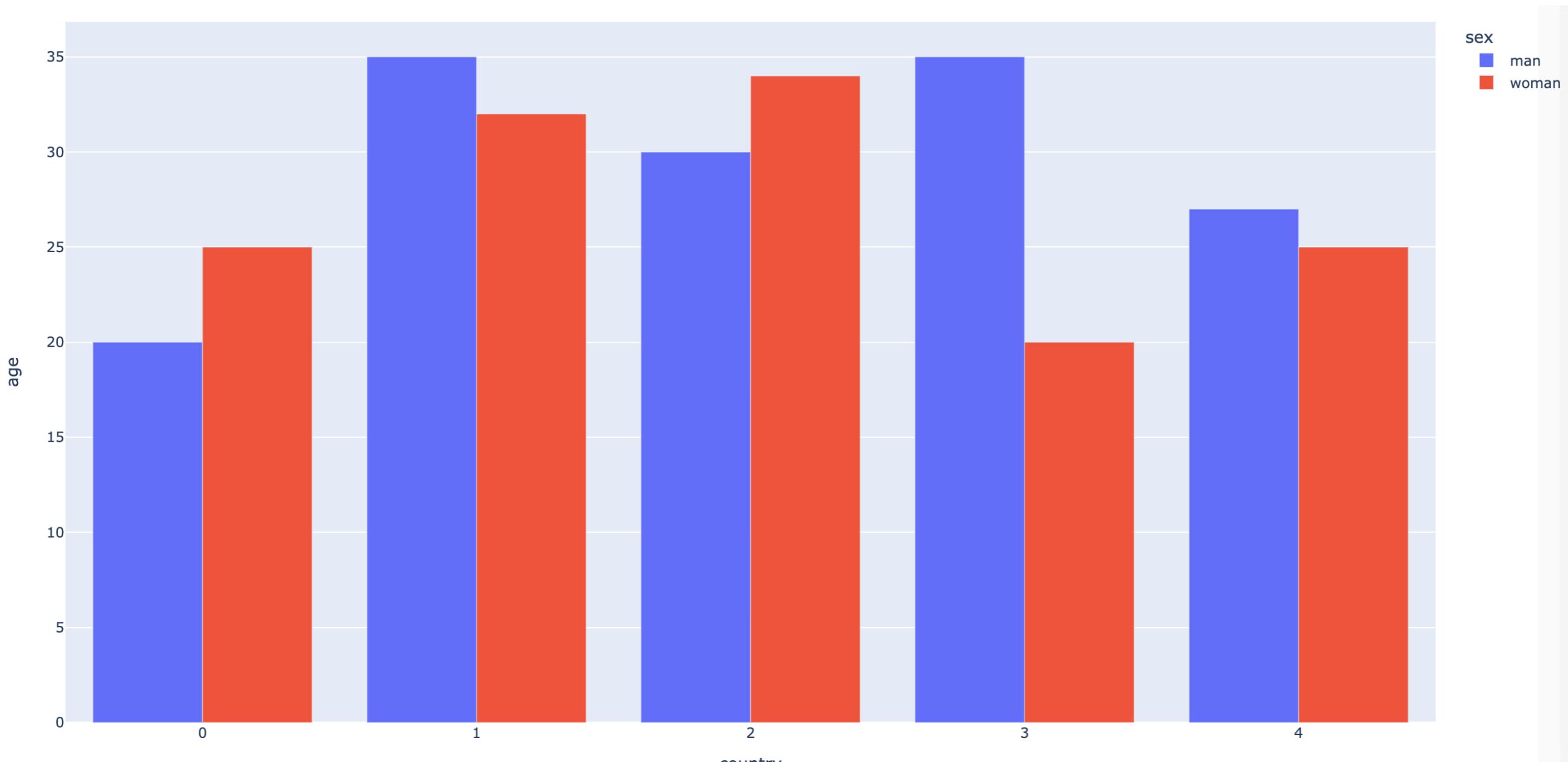
# Plotly (bar)

- E.g., demografia por sexo



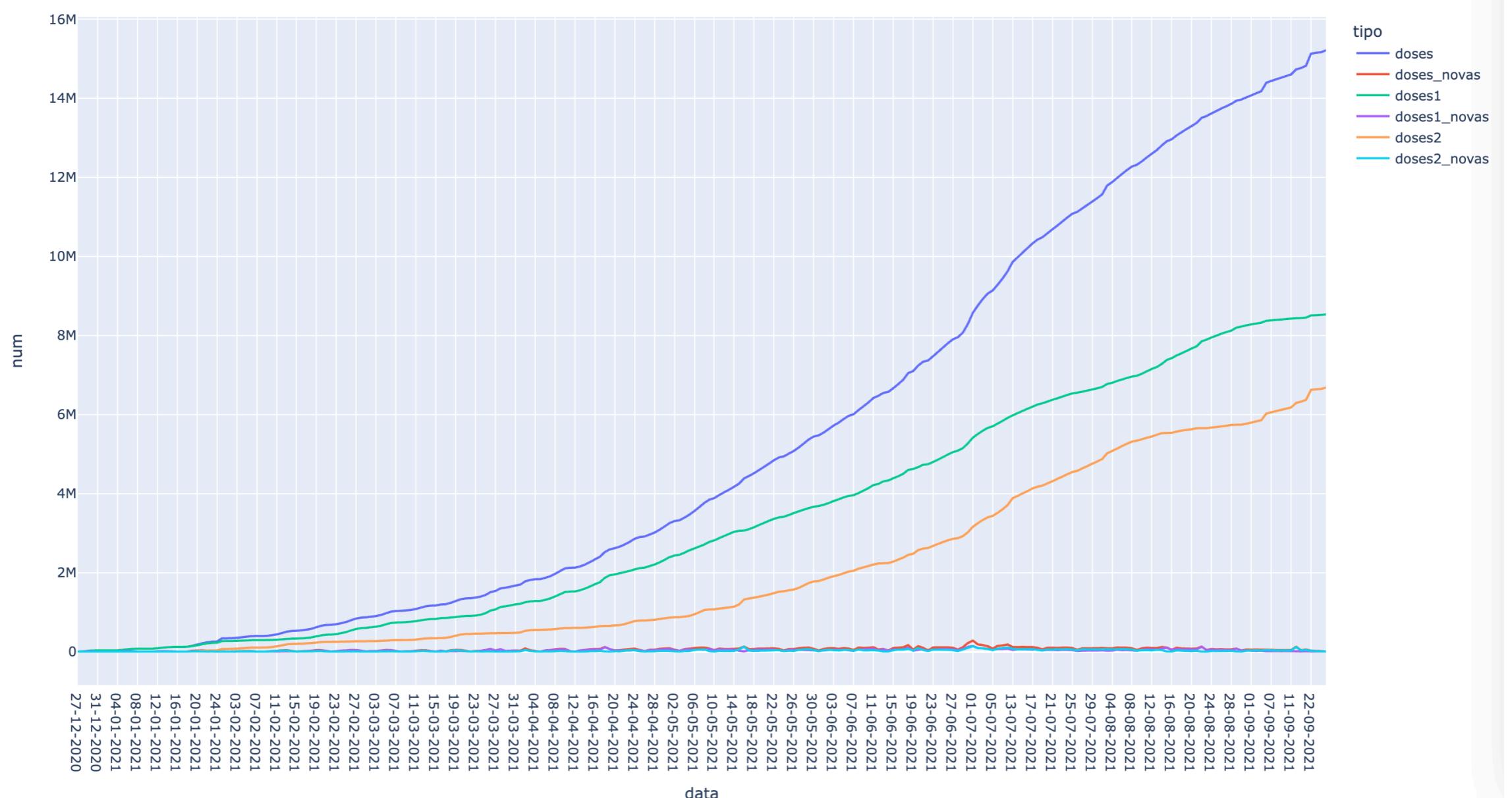
# Plotly (bar)

- E.g., demografia por sexo



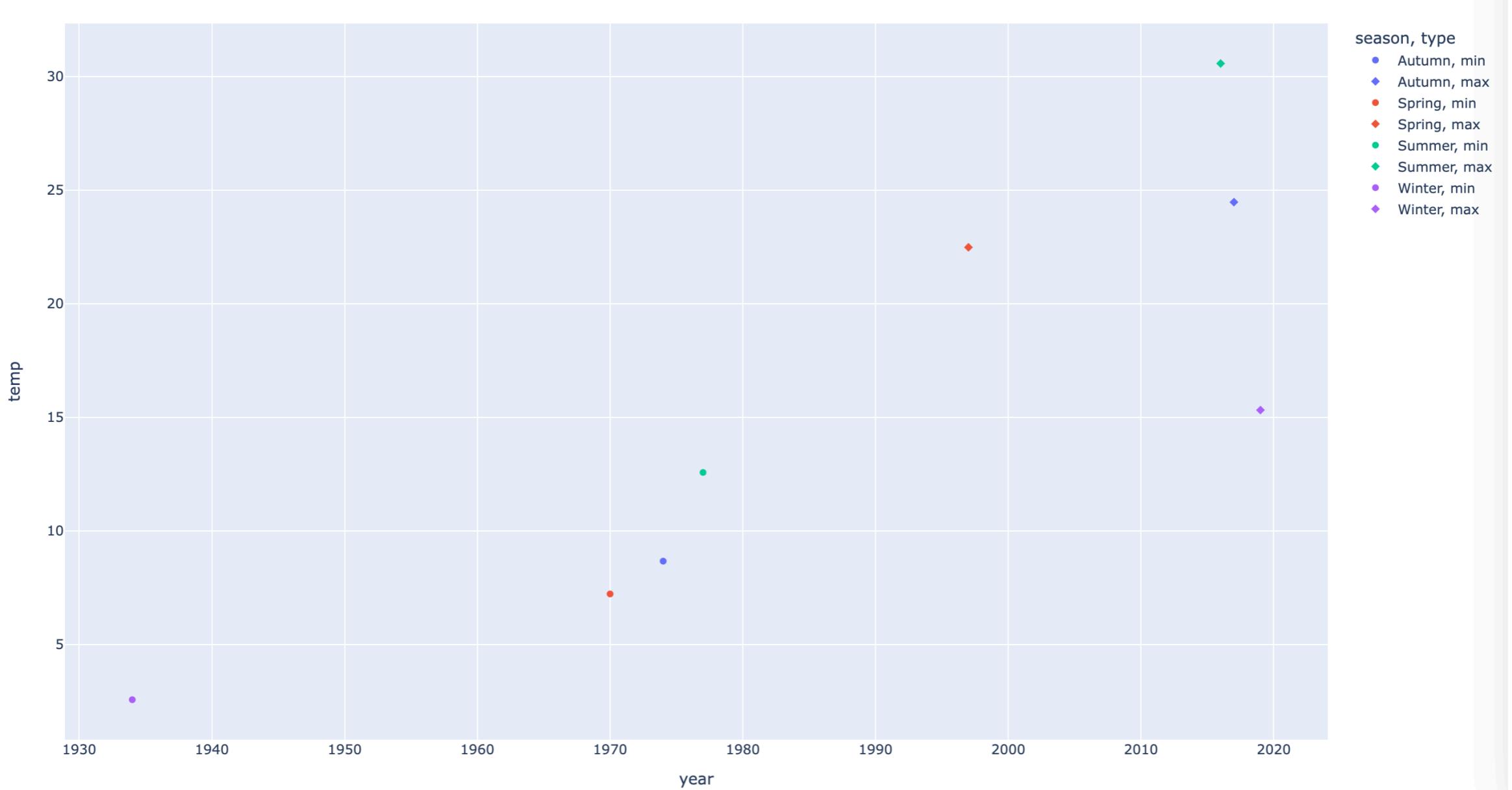
# Plotly (lines)

- E.g., dados de vacinação contra a COVID-19 publicados pela DGS e disponíveis [aqui](#)



# Plotly (scatter)

- E.g., dados climatéricos anuais de longa duração fornecidos pelo IPMA [aqui](#).



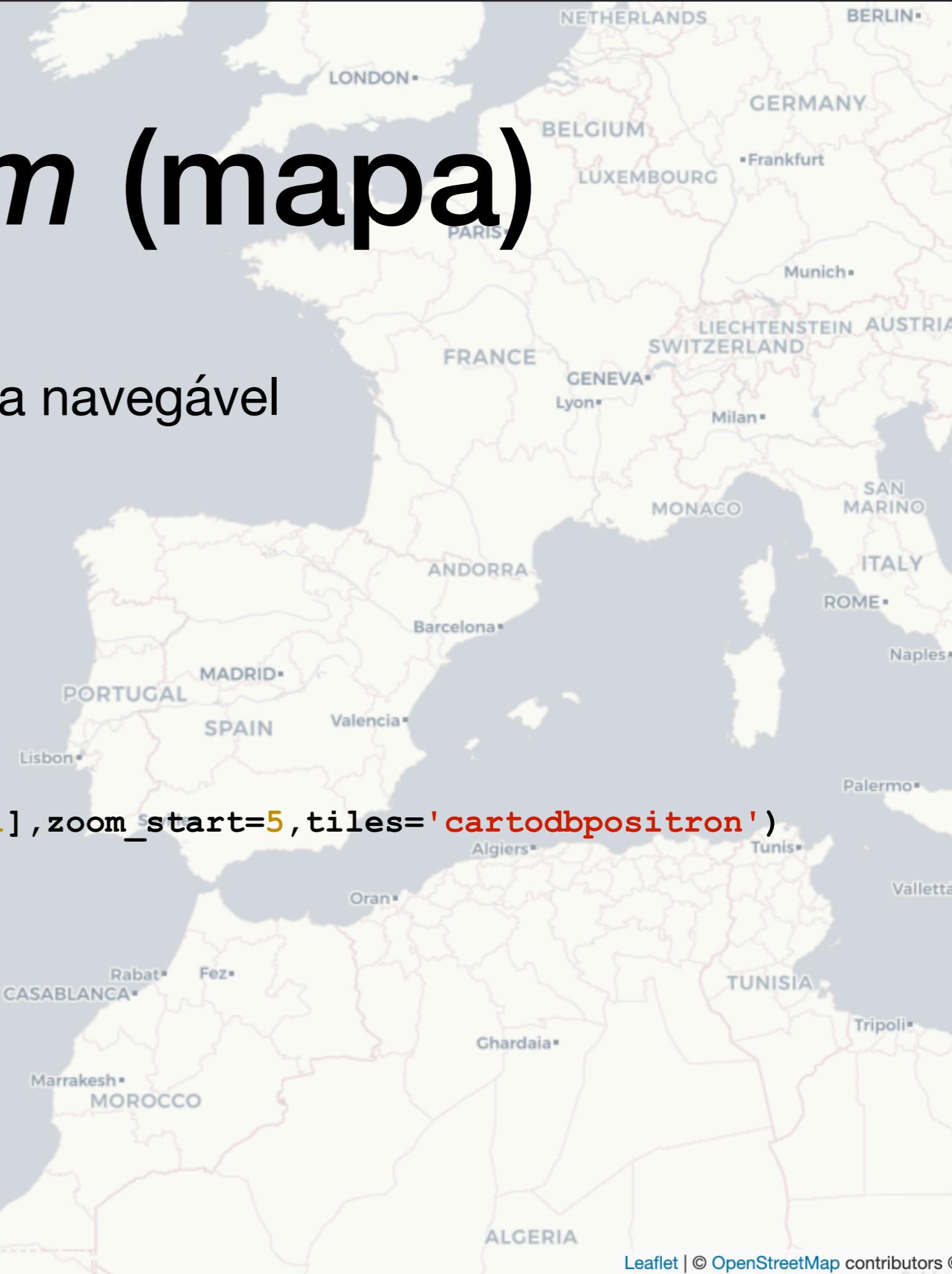
# Visualização web (mapas)

# Folium (mapa)

- Podemos criar um mapa navegável

```
import folium

# sequencia de lat,lon
mapa =
folium.Map([41.1579,-8.6291],zoom_start=5,tiles='cartodbpositron')
mapa.save("mapa.html")
```



# Folium (GeoJSON)

- Podemos criar um mapa a partir de um ficheiro GeoJSON, de forma similar ao que faz o [geojson.io](http://geojson.io)
- E.g., regiões de Portugal

```
# centrado no Porto
mapa = folium.Map([41.1579,-8.6291],
                  zoom_start=5,
                  tiles='cartodbpositron')

with open('portugal.geojson','r') as f: geojson = json.load(f)
folium.GeoJson(geojson).add_to(mapa)
mapa.save("mapa.html")
```

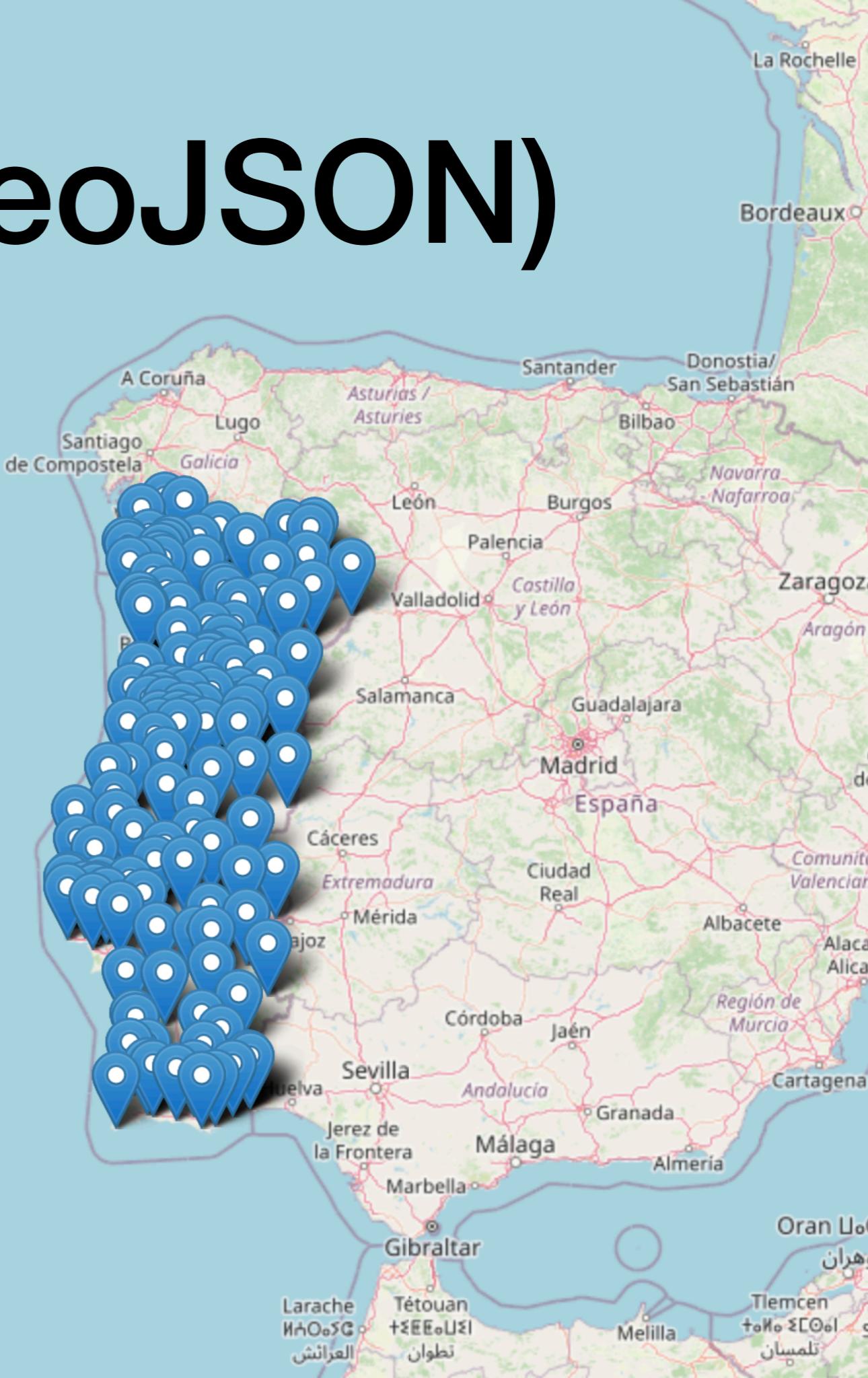


# Folium (GeoJSON)

- Podemos criar um mapa a partir de um ficheiro GeoJSON, de forma similar ao que faz o [geojson.io](#)
- E.g., estações metereológicas ([IPMA](#))

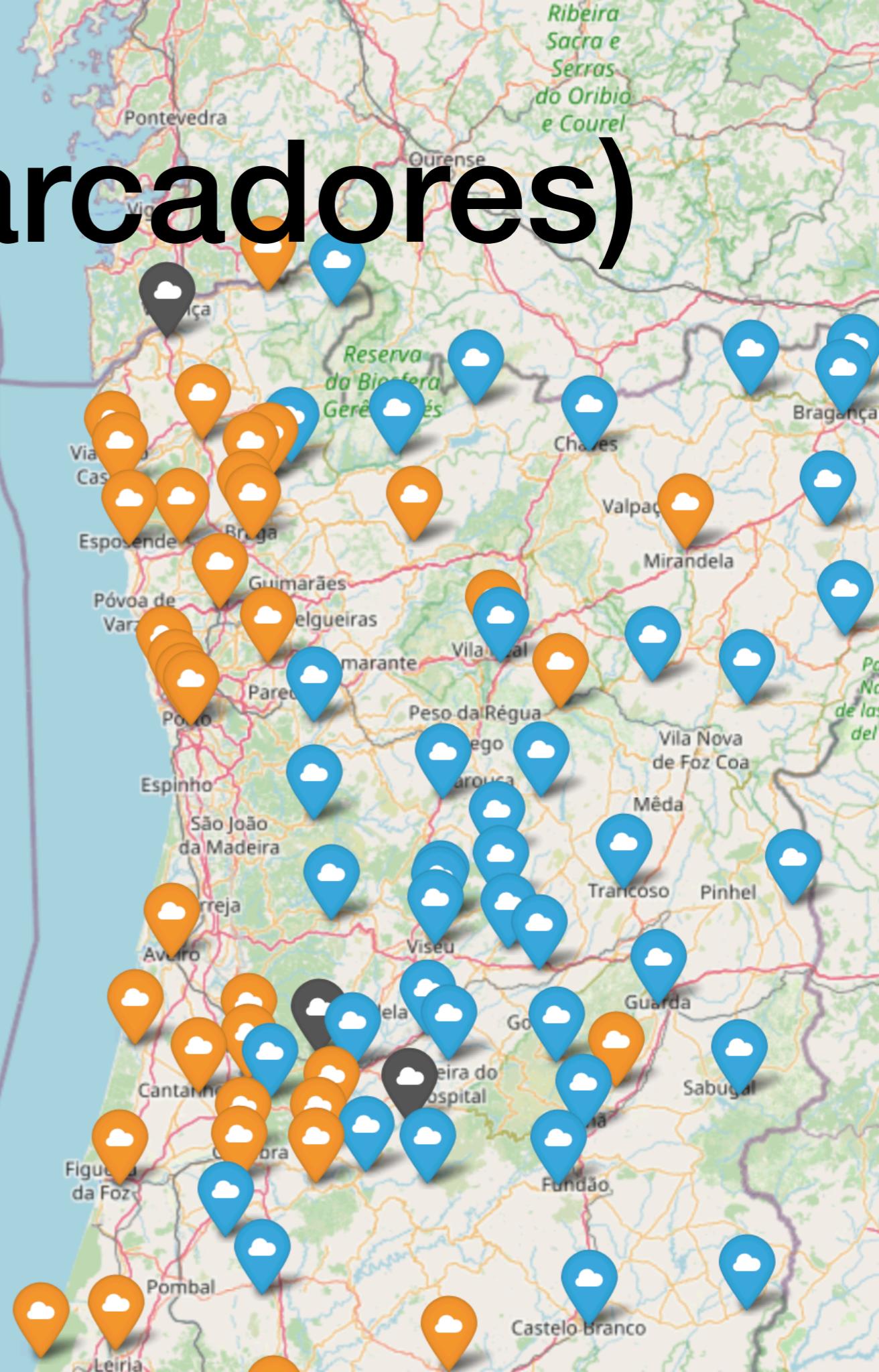
```
# centrado no Porto
mapa =
folium.Map([41.1579,-8.6291],
zoom_start=6,tiles='OpenStreetMap')

with open('obs-
surface.geojson','r') as f:
geojson = json.load(f)
folium.GeoJson(geojson).add_to(
(mapa)
mapa.save("mapa.html")
```



# Folium (marcadores)

- Podemos criar e costumizar marcadores
- E.g., estado do tempo



# Folium (marcadores)

- Carregar dados com `geopandas`
- Adicionar marcadores um a um (lista de ícones)

```
mapa = folium.Map([41.1579, -8.6291], zoom_start=8, tiles='OpenStreetMap')

gdf = gpd.read_file('obs-surface.geojson')
gdf['time'] = pd.to_datetime(gdf['time'])
latest_time = gdf['time'].max()
gdf = gdf[gdf['time'] == latest_time]

for i, row in gdf.iterrows():
    if row['temperatura'] >= 20: co='red';
    elif row['temperatura'] >= 10: co='orange';
    elif row['temperatura'] >= 0: co='blue'
    else: co='gray';
    icn = folium.Icon(color=co, icon='glyphicon-cloud')

folium.Marker(location=(row['geometry'].y, row['geometry'].x), icon=icn, tooltip=row['localEstacao'], popup=str(row)).add_to(mapa)
mapa.save("mapa.html")
```

# Folium (mapas cores)

- E.g., dados de mortes totais por COVID-19 da OMS, código de exemplo anterior
- Processar mortes totais (dados de cores)

```
codes = pd.read_csv("country-codes.csv",usecols=['ISO3166-1-Alpha-3','ISO3166-1-Alpha-2'])
codes = codes.rename(columns={'ISO3166-1-Alpha-3':'ISO_A3','ISO3166-1-Alpha-2':'ISO_A2'})
df = pd.read_csv("WHO-COVID-19-global-data.csv")
df = df.groupby(by='Country').tail(1)
df =
df.rename(columns={'Country_code':'ISO_A2','Cumulative_deaths':'Deaths'})
df = df[['ISO_A2','Deaths']]
gdf = pd.merge(codes,df,how='inner')
```

# Folium (mapas cores)

- Carregar coordenadas *geojson* e dados de cores à parte
- Código de palete de cores, e.g.. <https://colorbrewer2.org/>

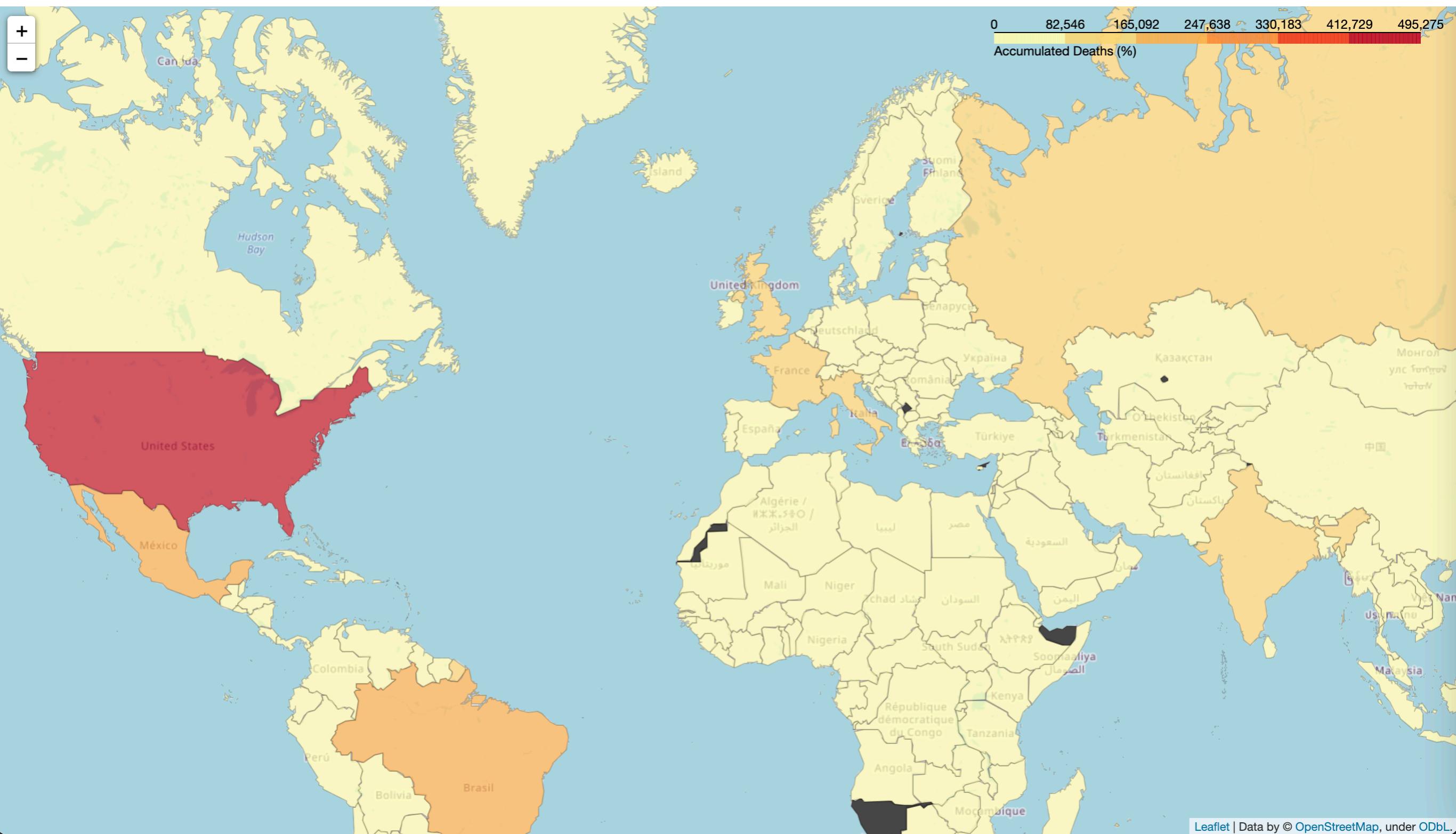
```
with open('countries.geojson', 'r') as f: geojson = json.load(f)

mapa = folium.Map([41.1579, -8.6291], zoom_start=3)

folium.Choropleth(
    geo_data=geojson,
    data=gdf,
    columns=["ISO_A3", "Deaths"],
    key_on="feature.properties.ISO_A3",
    fill_color="YlOrRd",
    fill_opacity=0.7,
    line_opacity=0.2,
    legend_name="Accumulated Deaths (%)",
).add_to(mapa)

mapa.save("mapa.html")
```

# *Folium* (mapas cores)

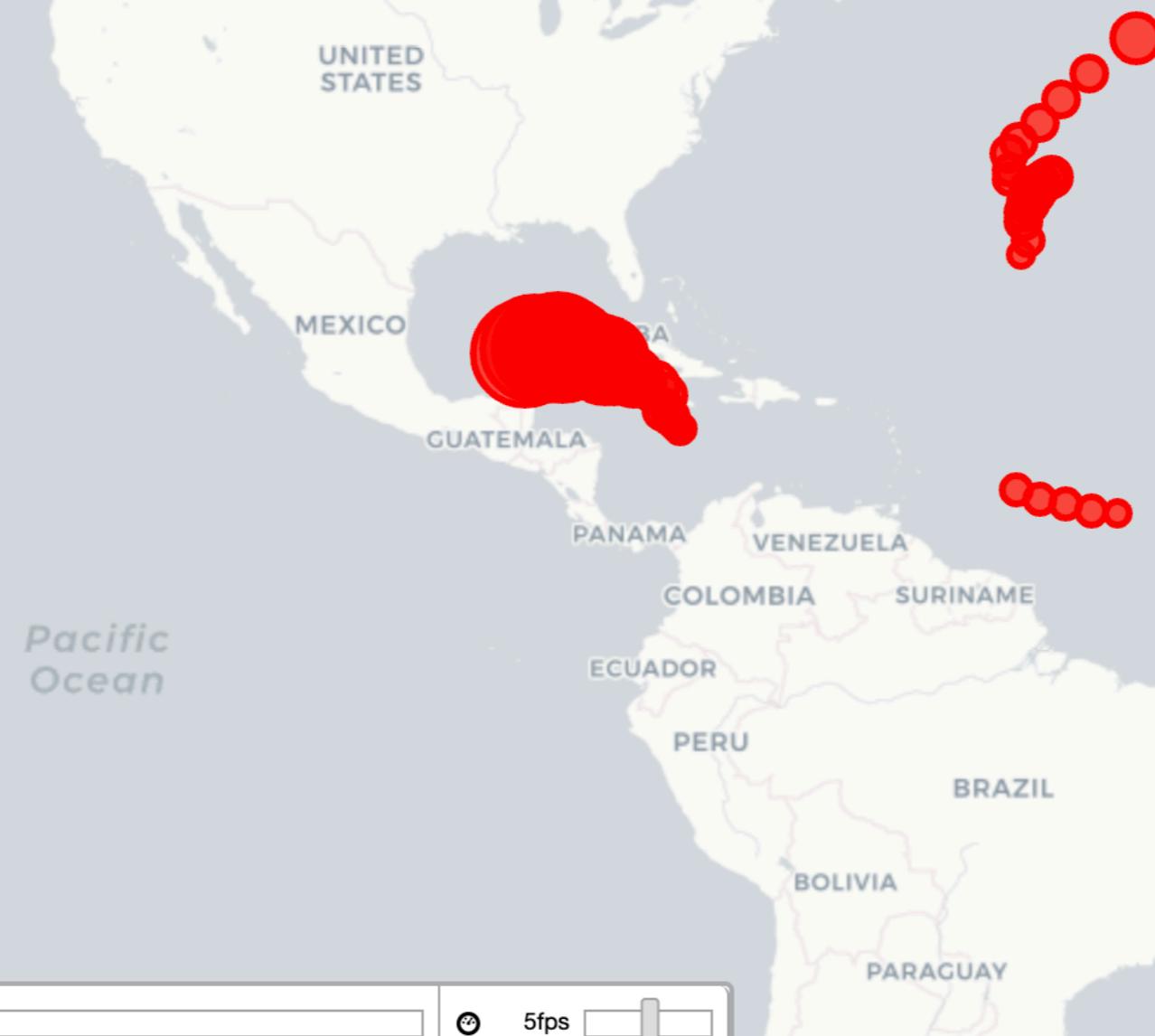


# *Folium* (animação)

- E.g., uma animação do percurso e intensidade do vento de furacões que assolaram a Costa Atlântica dos EUA
  - Dataset HURDAT2 com dados desde 1851 disponibilizado pela NOAA (United States National Oceanic and Atmospheric Administration); versão CSV aqui
1. Animar trajetória como círculos com posição de longitude/latitude e raio proporcional à intensidade do vento
  2. Cada círculo representa uma medição num instante de tempo concreto, e tem uma duração fixa na animação

# Folium (animação)

- E.g., uma animação do percurso e intensidade do vento de furações que assolararam a Costa Atlântica dos EUA



# Visualização web (grafos)

# *Plotly* (grafos)

- Podemos desenhar grafos web...



Construir grafo com o *NetworkX*

Calcular layout do grafo com o *NetworkX*



Converter arestas num *DataFrame pandas*

Desenhar arestas  $(i,j)$  como um scatter plot usando o *plotly*

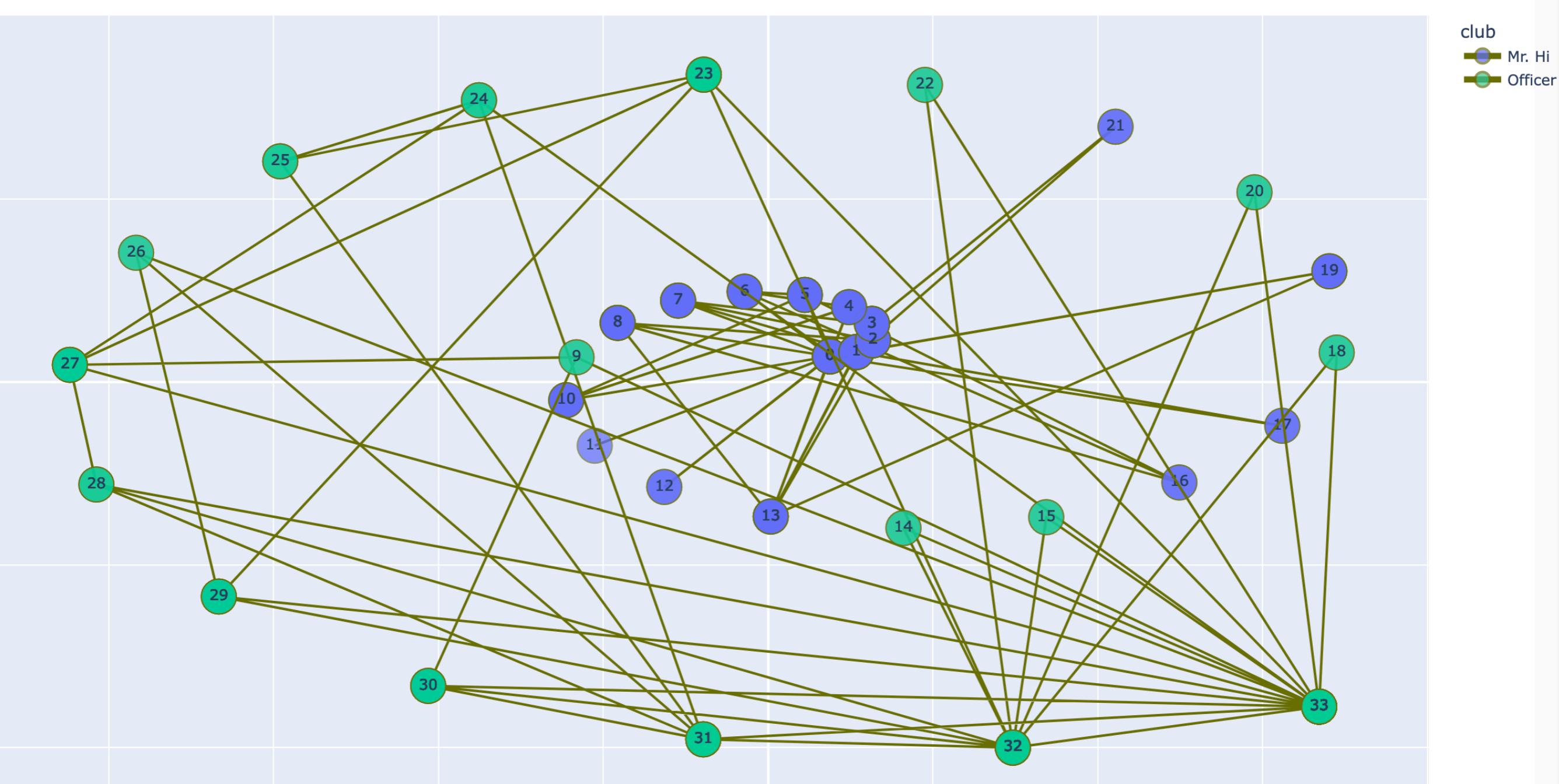
Node	X	Y
i	x_i	y_i
j	x_j	y_j
NaN	NaN	NaN



Várias complicações e limitações, e.g., labels?

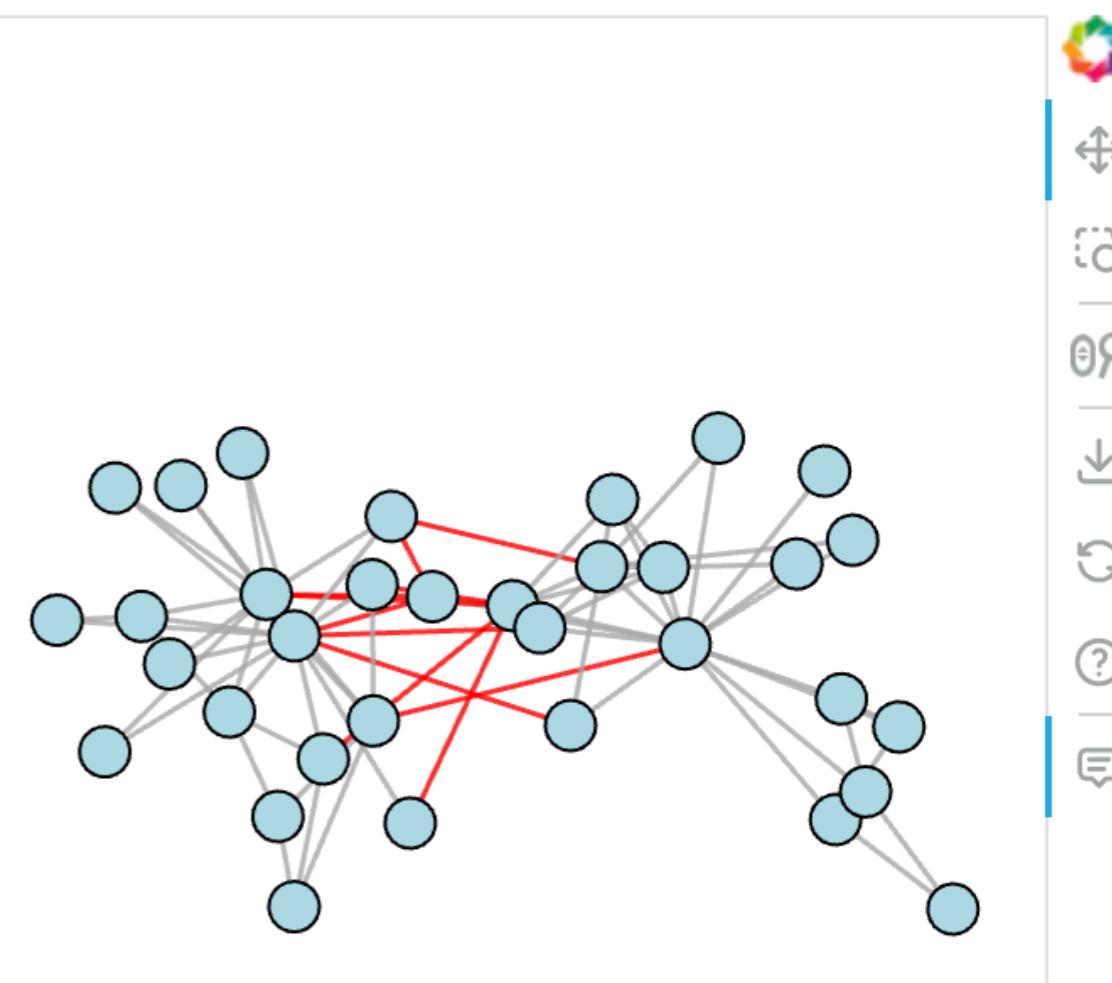
# Plotly (grafos)

- E.g., grafo da aula anterior do karate club



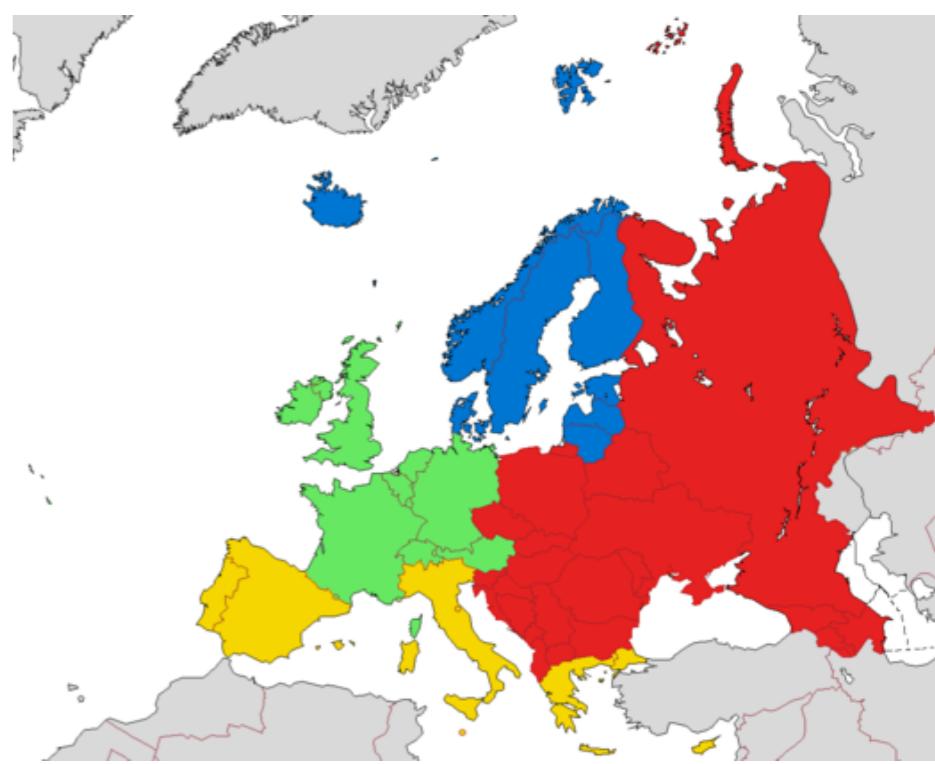
# Bokeh (grafos)

- Melhor integração com o *NetworkX*
  - Podemos desenhar diretamente um grafo
- Biblioteca mais complexa
- E.g., grafo da aula anterior do karate club



# Bokeh (chord plot)

- Um tipo concreto de visualização circular de grafos
- E.g., correlação de votos com regiões geopolíticas de países na Eurovisão
- Dados de votação retirados [daqui](#)
- Identificação de regiões retirada do [EuroVoc](#)
- Essencialmente um grafo direcionado
  - Nodos = países
  - Arestas = país P1 deu X pontos a país P2



# Bokeh (chord plot)

- E.g., correlação de votos com regiões geopolíticas de países na Eurovisão

