Lecture 6
Maps, graphs, time series

Data visualization · 1-DAV-105

Lecture by Broňa Brejová

More details in the notebook version

Part I: Maps

Maps

Each map is a visualization of data about location of objects.

Conventions about colors and symbols, orientation etc. allow us to quickly understand a map.

Example:
A topographic map from US

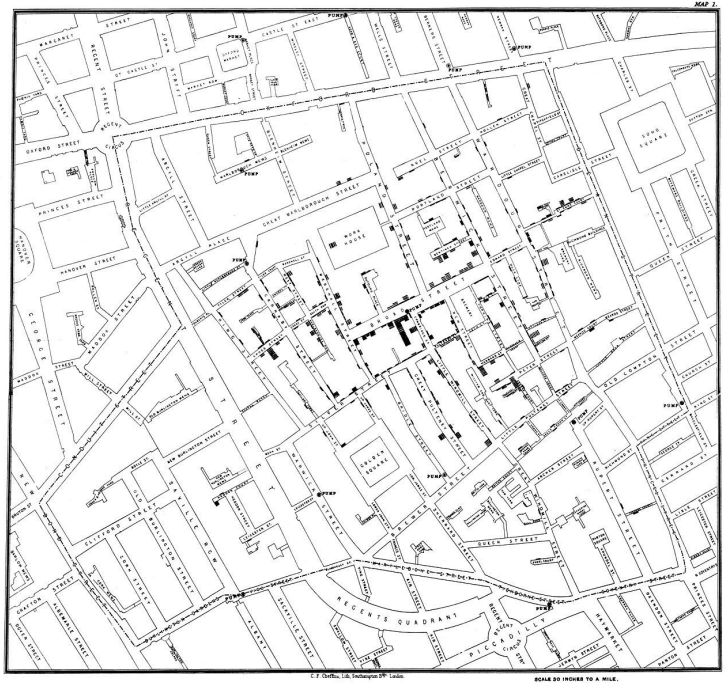
https://en.wikipedia.org/wiki/Map#/media/File:Topographic_map_example.png

Data visualization in maps

Thematic maps (tematické mapy) visualize data other than typical geographical features

Recall Snow's map of cholera cases (1854)

Additional examples: Wikipedia, GeoPlot library gallery.



https://commons.wikimedia.org/wiki/File:Snow-cholera-map-1.jpg

Map projection (kartografické zobrazenie)

A transformation to project the surface of a globe onto a plane

Each projection introduces some distortion

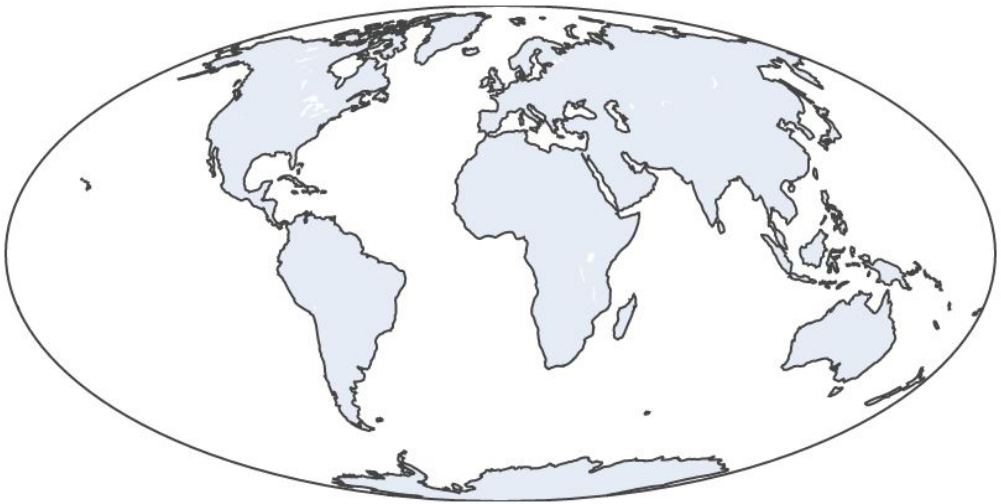
Conformal projections preserve local angles, but distort other aspects, such as lengths, areas etc.

For example, **Mercator projection** (1569) developed for navigation, but shows Greenland bigger than Africa, while in fact it is 14x smaller.

Map projection (kartografické zobrazenie)

Equal-area projections preserve areas (cannot be conformal at the same time).

These are typically good for data visualization, as they make areas comparable.



Example: **Mollweide equal-area projection** (1805)

Map projection (kartografické zobrazenie)

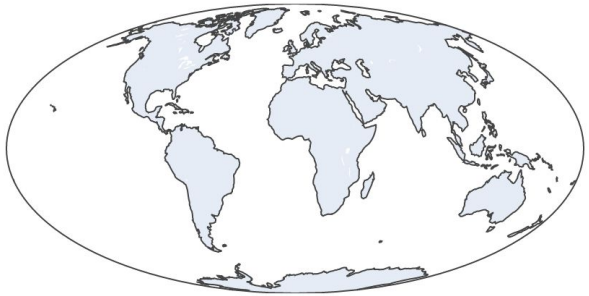
Orthographic projection is similar to a photograph of the Earth from a very distant point.

It is not an equal-area projection, but our sense of perspective may compensate.

It displays one hemisphere.

Recommended projections (Cairo, The Truthful Art)

Whole world: e.g. Mollweide equal-area projection (1805)
Continents / large countries: e.g. Lambert azimuthal equal-area projection (1772)
Countries in mid-latitudes: e.g. Albers equal-area conic projection (1805)
Polar regions: e.g. Lambert azimuthal equal-area projection (1772)



Mollweide

Lambert

Projection examples in Plotly

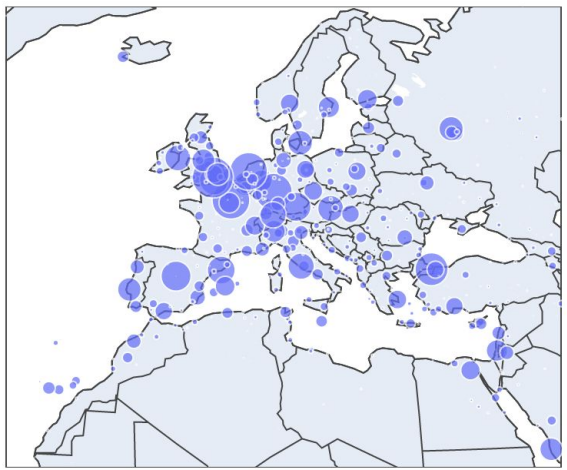
```
def show_world(projection, scope=None):
  # create a map figure with an empty scatterplot
  fig = go.Figure(go.Scattergeo())
  # set the desired projection
  fig.update_geos(projection_type=projection)
  # we can also limit the scope of the map
  if scope is not None:
    fig.update_geos(scope=scope)
  # finally, make the image smaller and with 0 margins
  fig.update_layout(height=200,
                    margin={"r":0,"t":0,"l":0,"b":0})
  # show the figure
  fig.show()
```

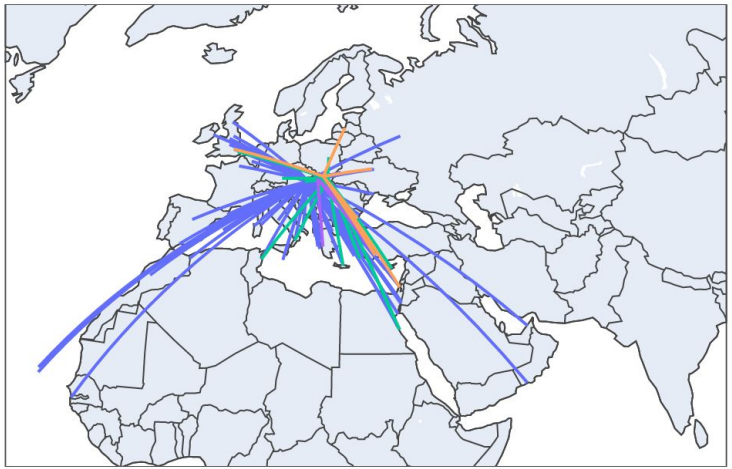
```
show_world("azimuthal equal area", "europe")
```

Adding data as points and lines to a map

Geographic coordinates of places can be projected as x and y.
Additional values can be shown using marker color / size or line color / width.

Example: airport locations in Europe and airline connections from Slovakia.





Our airport dataset

- The dataset of 2173 international airports of the world from the World Bank under the CC-BY 4.0 license, and preprocessed.
- For each airport its 3-letter code, name, country, 3-letter code of the country, the number of airplane seats per year (from unknown years) and the location.
- Stored in **GeoJSON** format.
- We parse the file using **GeoPandas** library for working with geographical data.
- It is an extension of Pandas DataFrame, with location information.

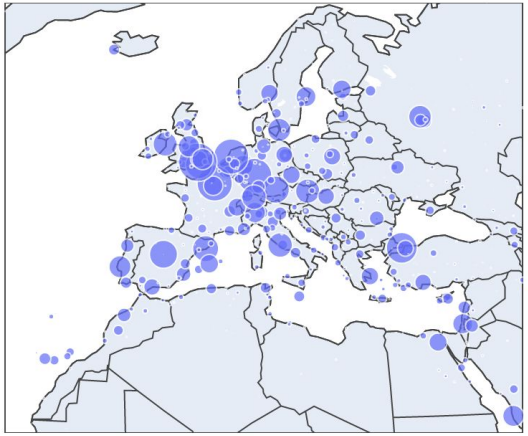
Importing airports in GeoPandas

```
import geopandas as gpd
airports = gpd.read_file("https://bbrejova.github.io/viz/data/airports.geojson")
display(airports.query('Country == "Slovakia"'))
```

	Orig	Name	TotalSeats	Country	ISO3	geometry
1489	BTS	M.R. Stefanik	1211732.116	Slovakia	SVK	POINT (17.21670 48.16670)
1490	ILZ	Zilina	3986.360	Slovakia	SVK	POINT (18.76670 49.23330)
1491	KSC	Barca	323259.132	Slovakia	SVK	POINT (21.25000 48.66670)
1492	PZY	Piestany Airport	1403.892	Slovakia	SVK	POINT (17.83330 48.63330)
1493	SLD	Sliac	11876.753	Slovakia	SVK	POINT (19.13330 48.63330)
1494	TAT	Tatry/Poprad	39612.286	Slovakia	SVK	POINT (20.24030 49.07190)

Drawing airport bubble graph in Plotly

```
fig = px.scatter_geo(
    airports,
    lat=airports.geometry.y,
    lon=airports.geometry.x,
    size="TotalSeats",
    hover_name="Name"
    )
fig.update_geos(
    projection_type="azimuthal equal area",
    lonaxis_range= [-20, 40],
    lataxis_range= [20, 70],
    showcountries = True
)
fig.update_layout(height=300, margin={"r":0,"t":0,"l":0,"b":0})
fig.show()
```



Interactive plot: zooming, panning, tooltips

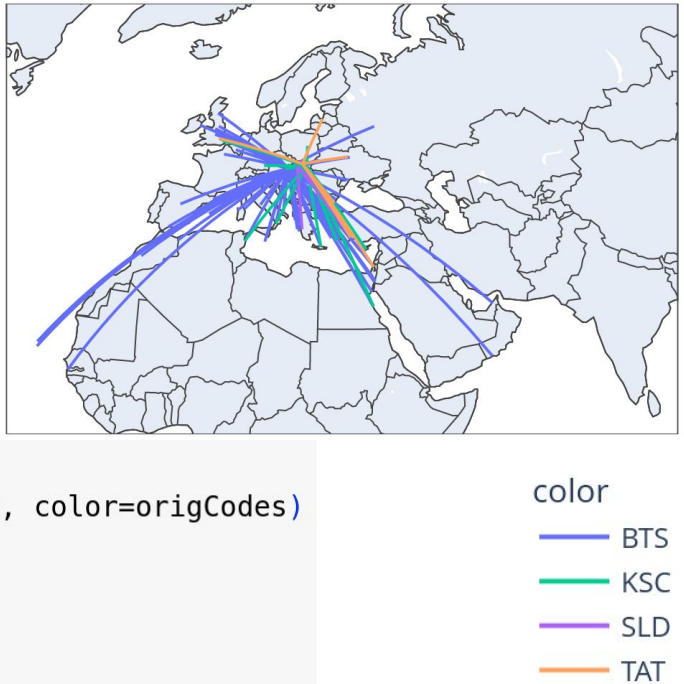
Importing airport connections in GeoPandas

```
connections = gpd.read_file("https://bbrejova.github.io/viz/data/airport_pairs_svk.geojson")
display(connections.head())
```

	OrigCode	DestCode	TotalSeats	geometry
0	BTS	ADB	7370.433	LINESTRING (17.21670 48.16670, 27.15620 38.29430)
1	BTS	AGP	15152.501	LINESTRING (17.21670 48.16670, -4.49810 36.67170)
2	BTS	AHO	14740.866	LINESTRING (17.21670 48.16670, 8.28890 40.63060)
3	BTS	AQJ	3275.748	LINESTRING (17.21670 48.16670, 35.01940 29.61250)
4	BTS	ATH	19654.488	LINESTRING (17.21670 48.16670, 23.94440 37.93640)

Drawing airline connections in Plotly

```
lats = []
lons = []
origCodes = []
destCodes = []

for index, row in connections.iterrows():
  x, y = row['geometry'].xy
  lats.extend(list(y) + [None])
  lons.extend(list(x) + [None])
  origCodes.extend([row['OrigCode']] * len(x) + [None])
  destCodes.extend([row['DestCode']] * len(x) + [None])

fig = px.line_geo(lat=lats, lon=lons, hover_name=destCodes, color=origCodes)
fig.update_geos(
  projection_type="azimuthal equal area",
  lonaxis_range= [-25, 55],
  lataxis_range= [10, 60],
  showcountries = True
)
fig.update_layout(height=300, margin={"r":0,"t":0,"l":0,"b":0})
fig.show()
```



Displaying variables that vary over space

Elevation can be measured at any point on land.
How is it visualized on geographic maps?

Displaying variables that vary over space

Elevation can be measured at any point on land.

How is it visualized on geographic maps?

https://upload.wikimedia.org/wikipedia/commons/d/d9/Slovakia_general_relief_map.svg

https://en.wikipedia.org/wiki/Map#/media/File:Topographic_map_example.png

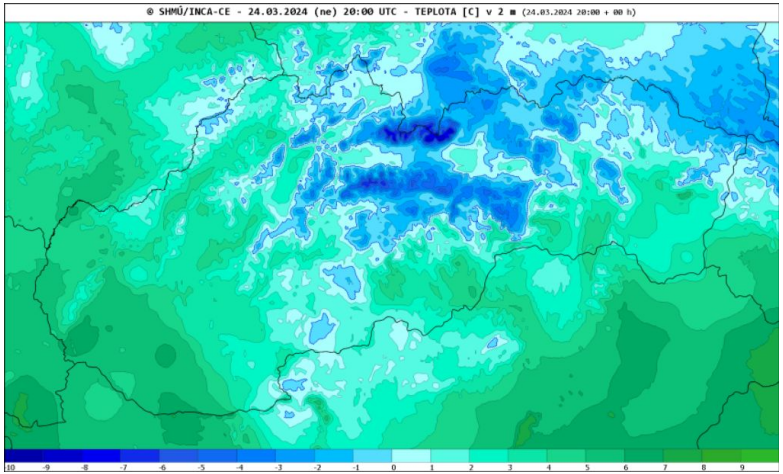
Isarithmic maps / isoline maps / heatmaps

Display a continuous variable over the map area (elevation, temperature etc.)

Value in each point can be shown by a color scale.
Some contour lines can be displayed as well.

A **contour line** (isoline, isopleth, isarithm, izočiara)
connects points of the same value.

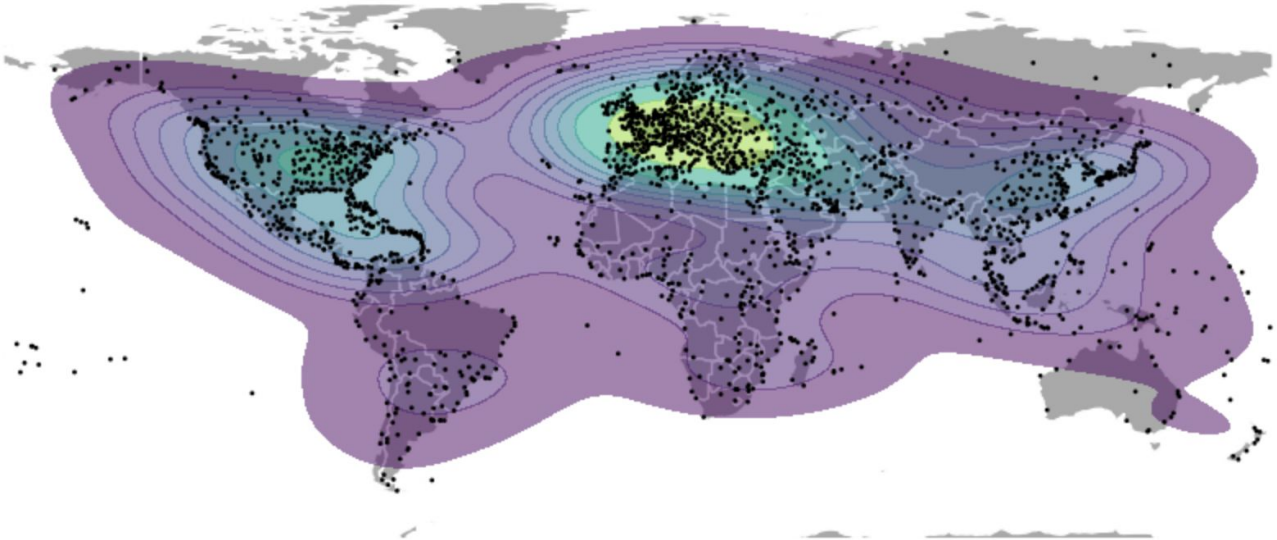
Example: short-term forecasts from
the Slovak Hydrometeorological Institute.



https://www.shmu.sk/sk/?page=1&id=meteo_inca_base

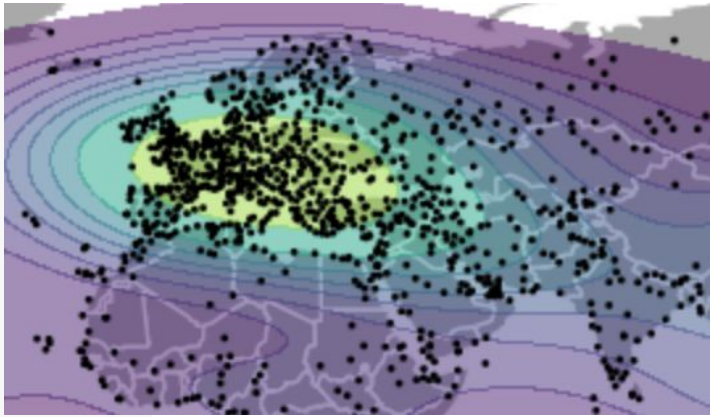
Density of airports as a isarithmic map

We use kdeplot function from the Geoplot library.
KDE stands for kernel density estimation (next lecture).



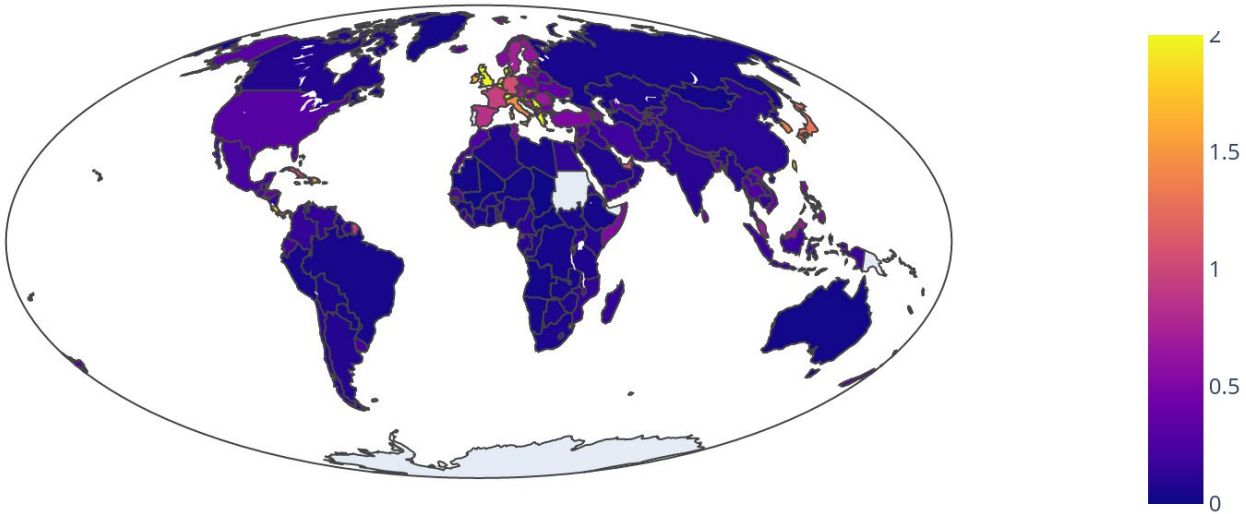
Density of airports in Geoplot library

```
import geoplot as gplt
# plot countries as a background
ax = gplt.polyplot(
    countries.explode(index_parts=True),
    edgecolor='white',
    facecolor='darkgray',
    figsize=(10, 5),
)
# plot semi-transparent isarithmic map
gplt.kdeplot(
    airports, cmap='viridis',
    fill=True, alpha=0.5, ax=ax
)
# plot points on top
gplt.pointplot(airports, s=1, color='black', ax=ax
pass
```



Choropleth maps (kartogramy)

These show numerical / categorical values for administrative regions (countries, districts, etc.) via colors applied to the whole region.
Example: The number of airports in a country per 10000 km^2



Types of variables over regions

Spatially extensive: apply to the unit as a whole. If we subdivide the region, spatially extensive variable will be often the sum of its parts.
Examples: total population, area, the number of airports in the country

Spatially intensive: may stay the same if you divide the unit, provided the unit is homogeneous without regional differences.
Examples: population density, life expectancy, GDP per person.

Spatially extensive variables are not appropriate for choropleths
(large value for a large country is visually attributed to each small subregion)

Computing airport stats per country in GeoPandas

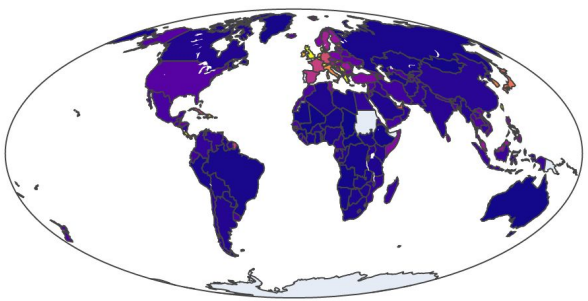
```
# compute the number of airports per country by groupby
airports_per_country = airports.groupby('ISO3').size()
# add the new column to a copy of the old table
countries2 = countries.copy(deep=True)
countries2['Airports'] = airports_per_country
# remove countries where airports or location are missing
countries2.dropna(subset=['geometry', 'Airports'], inplace=True)
# add columns with airport density and airports per million people
countries2['Airport_density'] = (countries2['Airports']
                                 / countries2['Area'] * 10000)
countries2['Airports_per_mil'] = (countries2['Airports']
                                  / countries2['Population'] * 1e6)
display(countries2.loc['SVK'])
```

Computing airport stats per country in GeoPandas

```
Type                                                 Sovereign country
Name                                                           Slovakia
Population                                                   5454073.0
Region                                             Europe & Central Asia
geometry            POLYGON ((22.558137648211755 49.08573802346714...
Area                                                      47069.779734
Airports                                                             6.0
Airport_density                                              1.274703
Airports_per_mil                                             1.100095
Name: SVK, dtype: object
```

Drawing choropleth map in Plotly

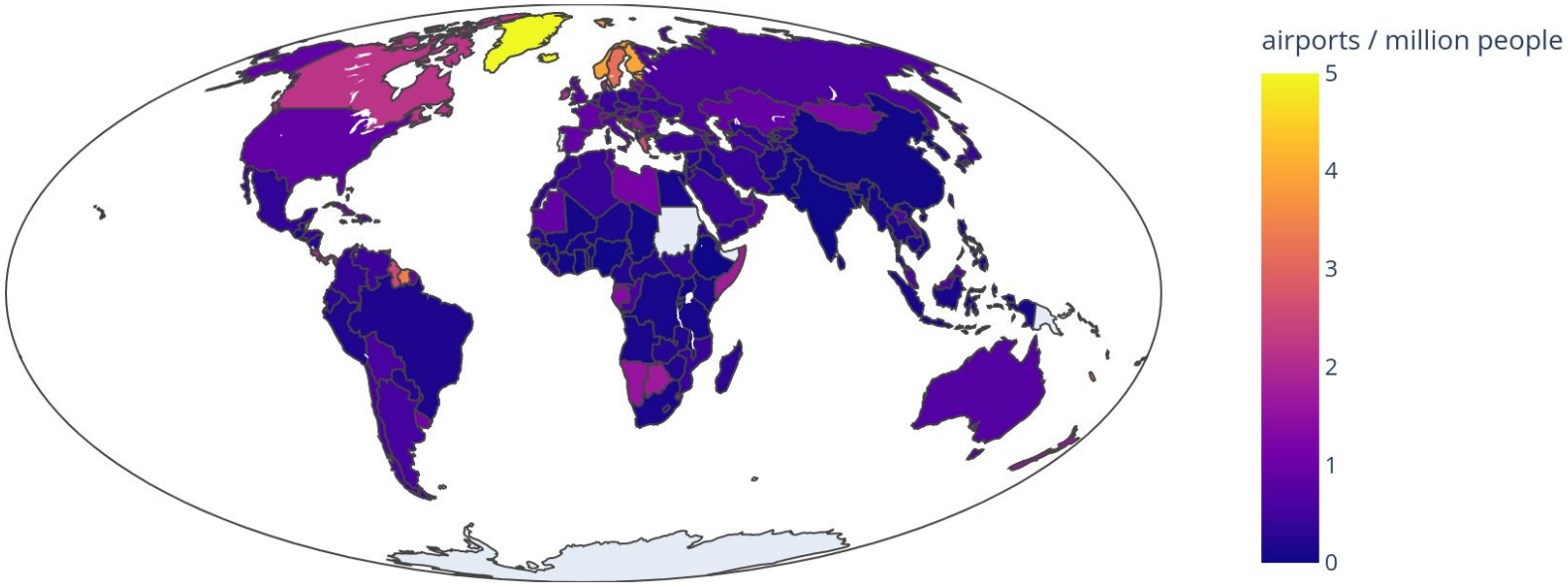
```
def draw_choropleth(data, column, range_color=None, label=None):
  fig = px.choropleth(
    data, locations=data.index, color=column,
    range_color=range_color,
    labels={column:label},
    hover_name="Name",
    projection = "mollweide"
  )
  fig.update_layout(height=300, margin={"r":0,"t":0,"l":0,"b":0})
  fig.show()

display(Markdown("**The number of airports per 10000 squared km**"))
draw_choropleth(countries2, 'Airport_density', (0, 2), 'airports / 10000 km2')
```



Another intensive variable

```
display(Markdown("**The number of airports per million inhabitans**"))
draw_choropleth(countries2, 'Airports_per_mil', (0, 5), 'airports / million people')
```

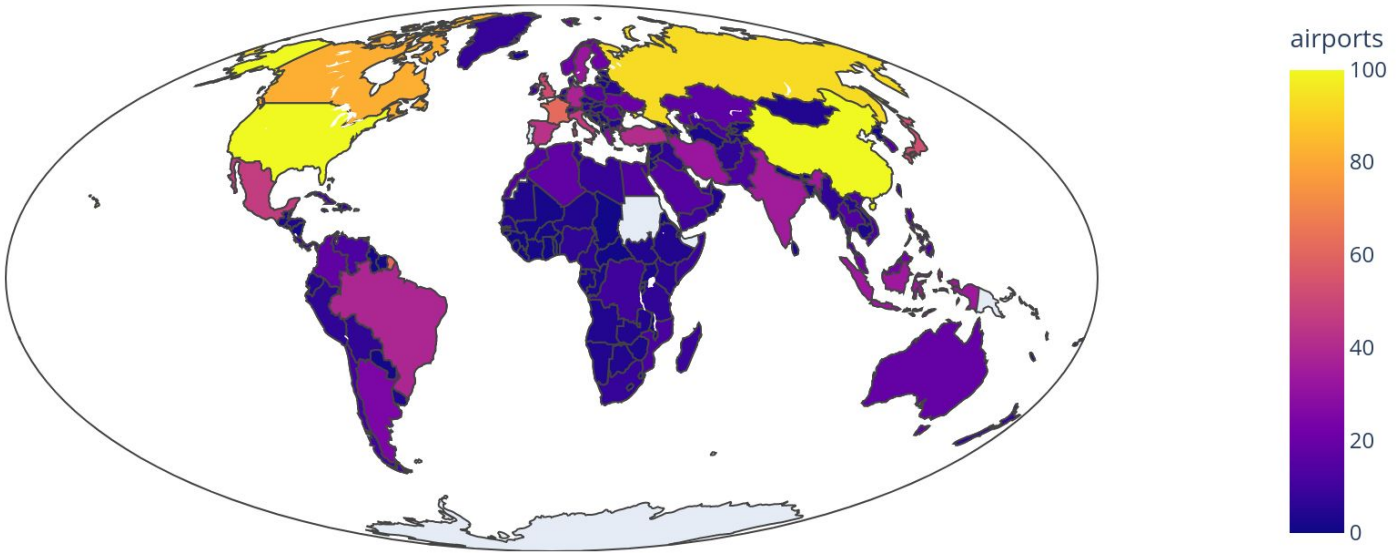
The number of airports per million inhabitans



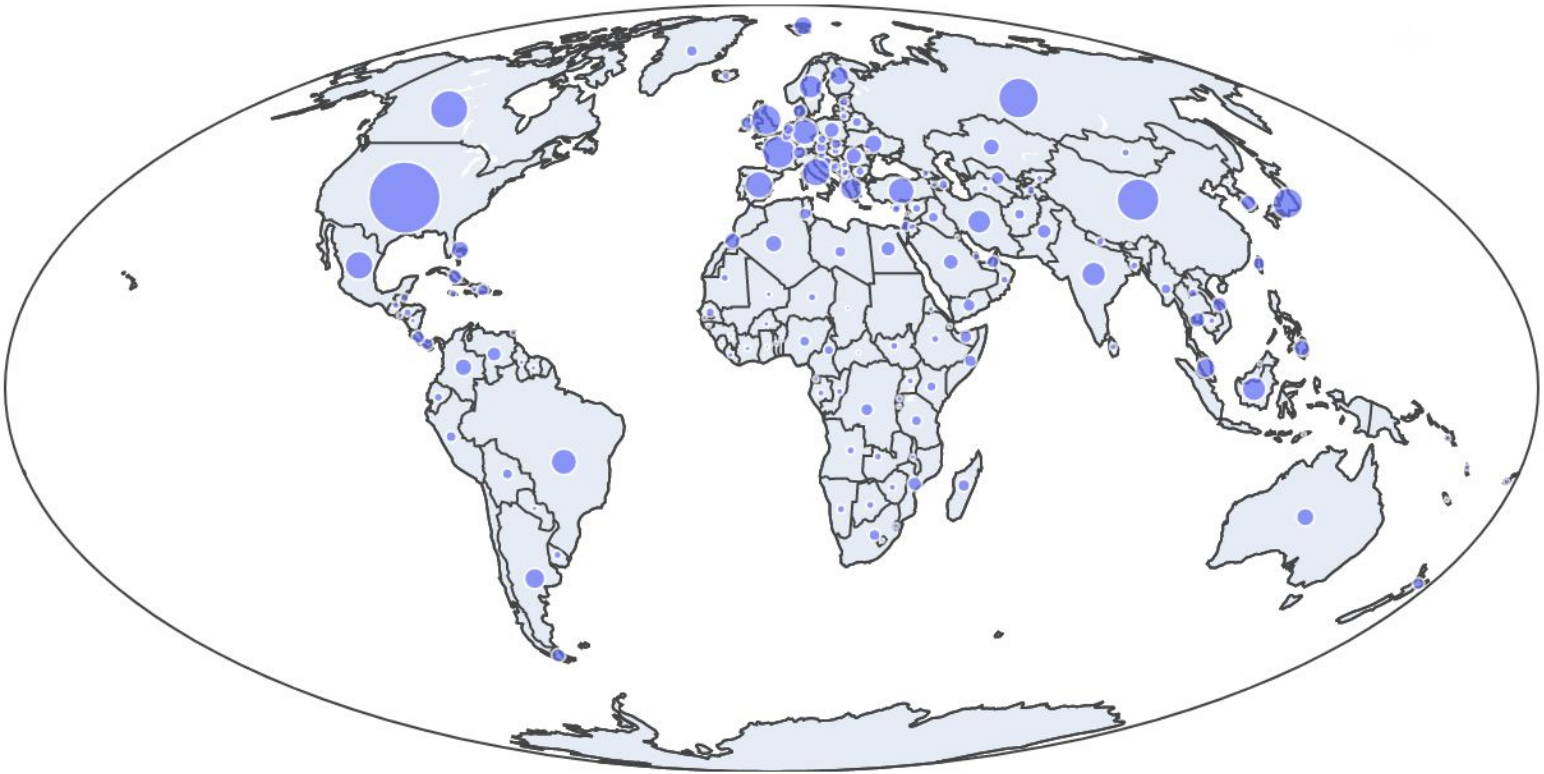
The number of airports is extensive (not good)

```
display(Markdown("**The number of airports in a country**"))
draw_choropleth(countries2, 'Airports', (0, 100), 'airports')
```

The number of airports in a country



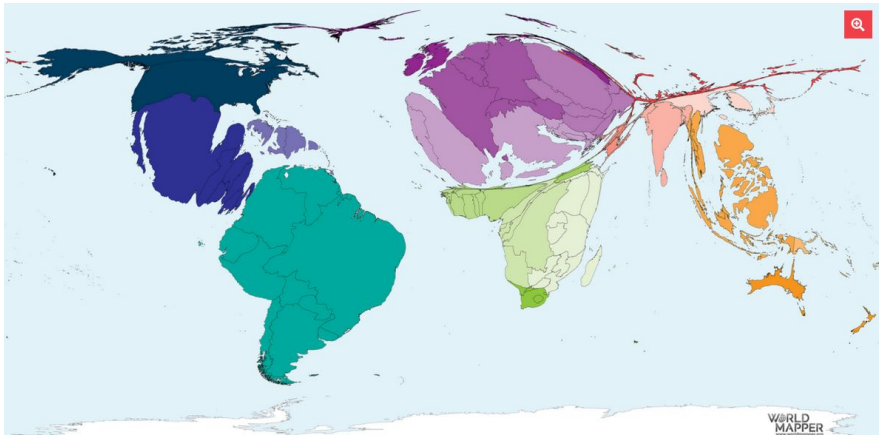
The number of airports as a bubble plot (better)



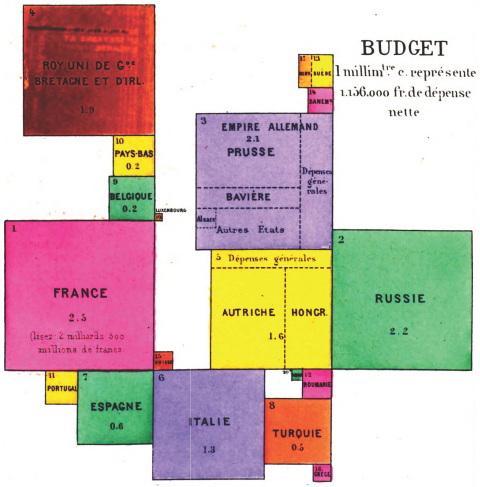
Cartogram vs kartogram

A choropleth map is called kartogram in Slovak.

In English, cartogram is a map with regions rescaled according to some variable
Examples: World's Catholic Population by World Mapper
Levasseur's cartogram of country budgets



https://worldmapper.org/maps/catholic-population-2005/



https://commons.wikimedia.org/wiki/File:Levasseur_cartogram.png

Summary of maps

- Many data sets contain geographic entities (countries, cities, coordinates)
- Displaying such data on maps shows spatial relationships
- Use appropriate equal-area projections
- Bubble plots: add points of various sizes
- Isarithmic maps / isoline maps / heatmaps: continuously varying variables
- Choropleth map: variables characterizing whole region
    - The variable should be intensive, e.g. normalized by area or population

Several useful libraries

- Geopandas for working with geographical data, extension of DataFrame
- Geoplot and Plotly for visualization

Part II: Graphs / networks

Graph / network

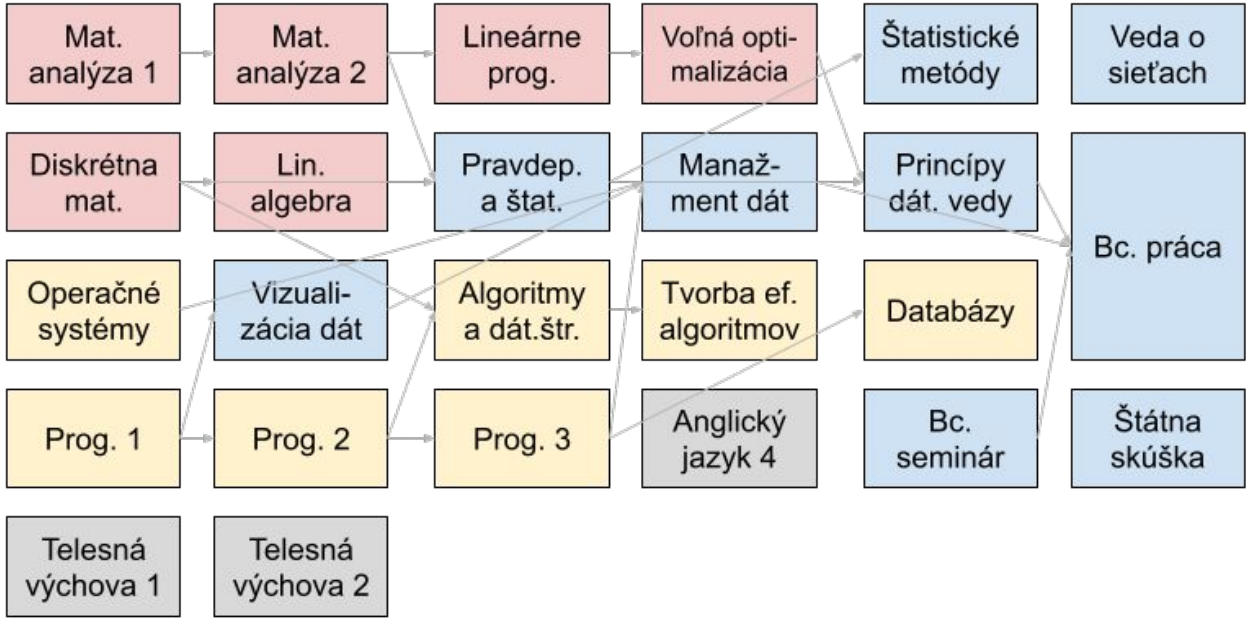
Vertices (vrcholy; also nodes, uzly) often real-world entities

Edges (hrany; also links, arcs) often relationships / connections between pairs of vertices

Many real-world examples:
places connected by roads, computers connected by network cables,
people connected by family or work relationships,
companies connected by financial transactions, texts connected by references,
tasks or courses connected by dependencies,
any objects connected based on similarity / shared features, ...

Graphs are very important in both computer science and data science.

Example: dependencies between DAV courses



Graph / network (cont.)

Edges: Directed (orientované) or undirected (for symmetric relationships)
Recall: how did you define directed / undirected edges in discrete mathematics?

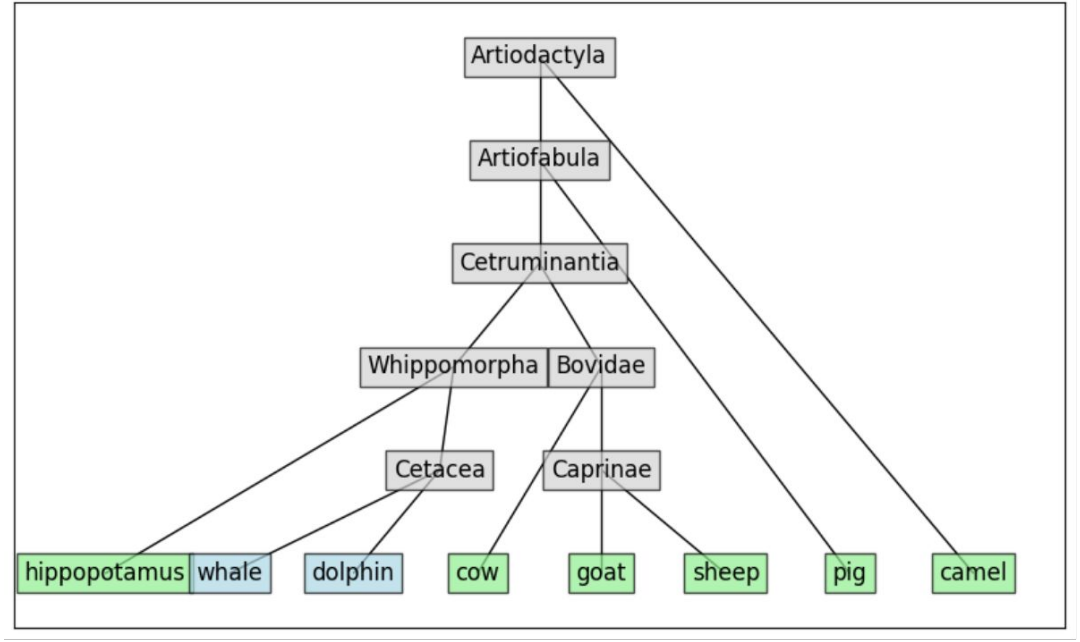
Graphs are covered in several courses: discrete mathematics, programming, design of efficient algorithms, network science.

Trees

An undirected graph is called a tree if it is connected and without cycles.

In practice we usually encounter rooted (directed) trees,
which have a single root,
all other vertices
can be reached from the root
via a unique path.

Creates parent / child
relationships between nodes



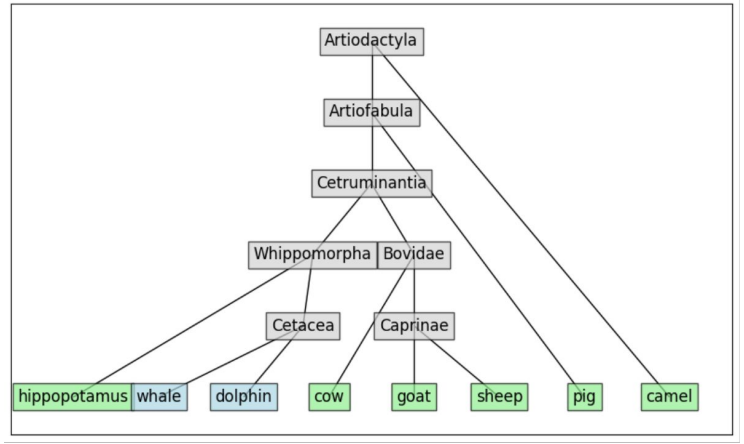
Trees and hierarchies

Trees can express hierarchies in which each entity has a single direct superior
Examples:

Company structure in which each employee (except for the head of the company) has a single supervisor (similarly army command)

Administrative divisions (country, region, district)

Species taxonomy
(animals, mammals, primates, ...)



More general hierarchies

Some hierarchies allow multiple direct superiors, for example:

- family tree where each person has two parents (and they may be distantly related),
- geometrical shapes, where a square is both a special case of a regular polygon and a special case of a rectangle and both of these are a special case of a polygon.

These hierarchies can be represented as directed acyclic graphs

- Acyclic means that by following edges we never get back to the starting node (nobody is their own ancestor).

What to study / visualize in real-life graphs?

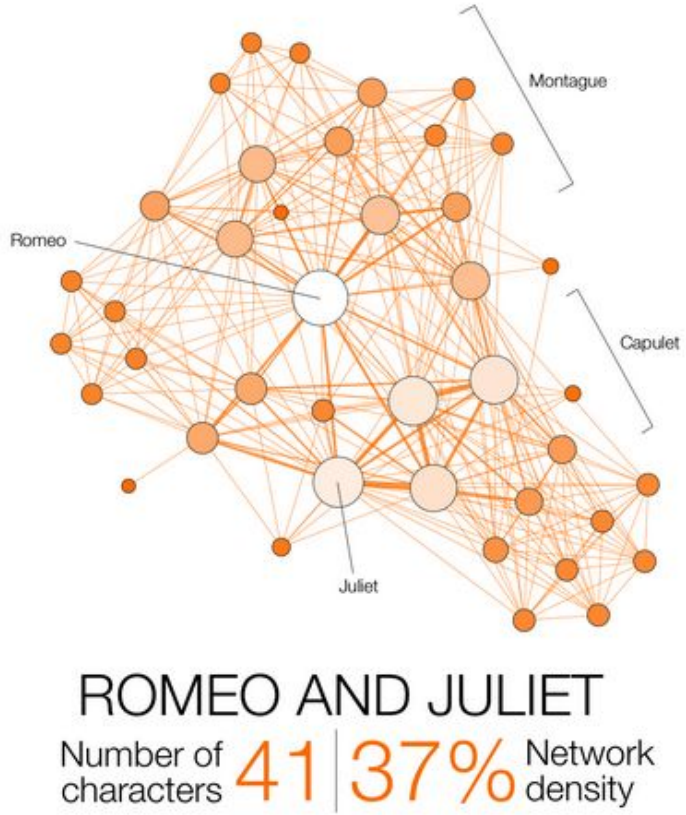
Details of connections for a particular node (requires zooming in large networks).

Overall structure of the graph: connected components, density of edges, presence of cycles, weak places (bridges and articulations), densely connected clusters

Do nodes with some property cluster together? (Are they connected by many edges?)

Example: character co-occurrence in Shakespeare

http://www.martingrandjean.ch/network-visualization-shakespeare/

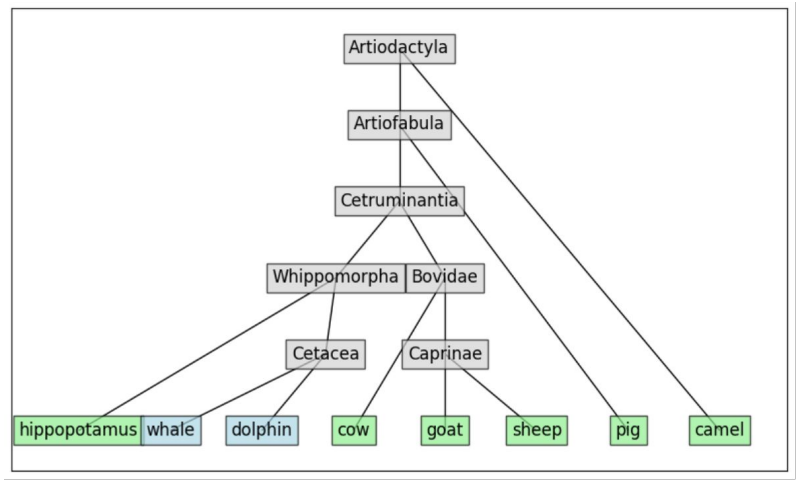


Basics of graph drawing

Vertices typically shown as markers (circles, rectangles etc.), possibly with labels, size, color, ...

Edges shown as lines connecting them, possibly of different color or width. They can be straight lines, arcs, polygonal lines or arbitrary curves.

Edge direction displayed as arrows
or all edges drawn to point in one direction,
e.g. downwards.



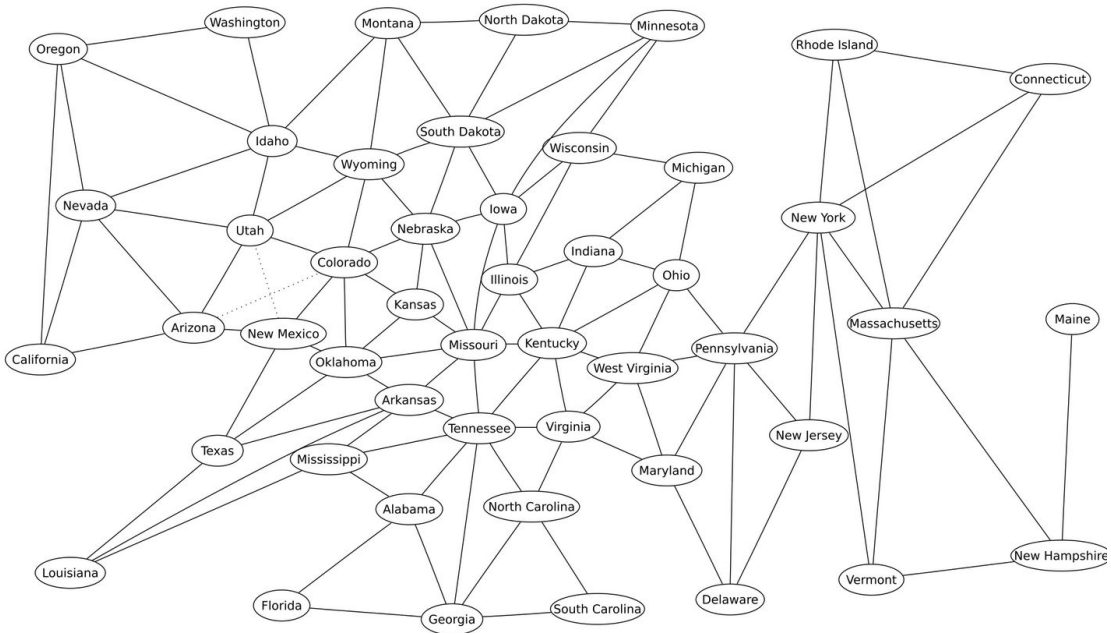
Desirable properties of a graph drawing

Nodes do not overlap.

Edges are not too long and have a simple shape without many bends.

The number of edge crossings is small.

The graph uses the space of the figure well without large empty regions.

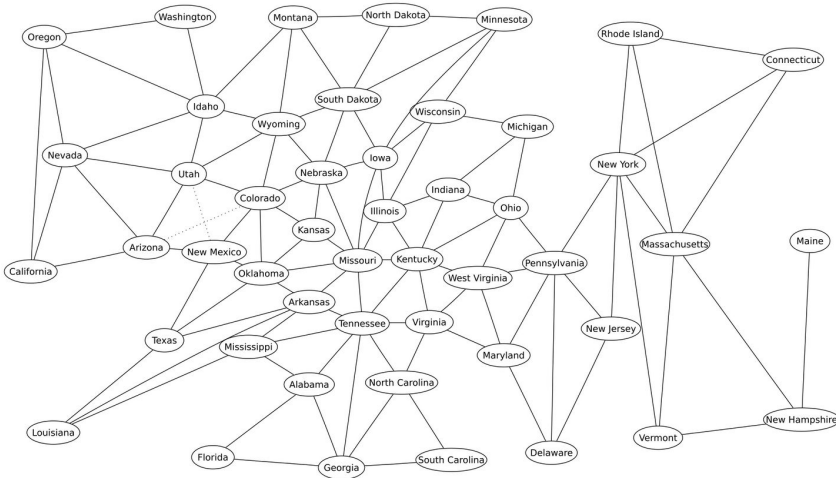


https://commons.wikimedia.org/wiki/File:UnitedStatesGraphViz.svg

Node positioning in graph drawing

Sometimes the position of nodes is given by their properties, e.g. on a map (airline connections), level of a hierarchy, timeline.

Otherwise we place nodes to optimize desirable properties, e.g. using **force-directed layout**, which assigns attractive forces (springs) between nodes connected by edges and repulsive forces between other pairs of nodes.



https://commons.wikimedia.org/wiki/File:UnitedStatesGraphViz.svg

Our hierarchy: manually created DataFrame of animals

Taxonomy of even-toed ungulates (párnokopytníky)

Level along tree (1=leaves, 6=root)

Category: land / sea / group

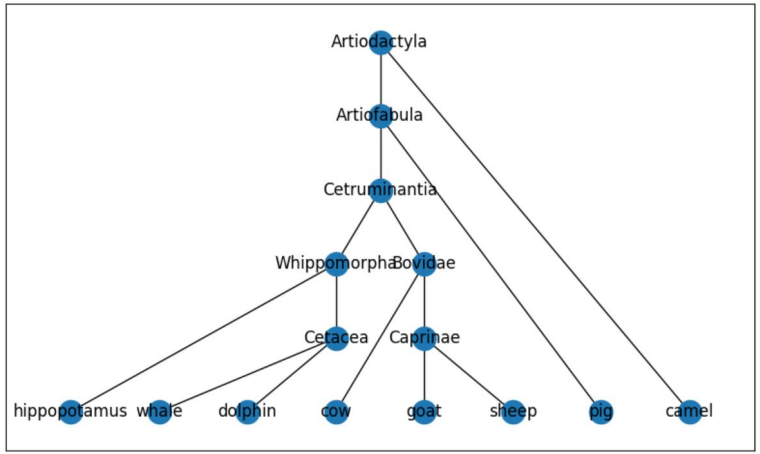
	name	parent	level	category
0	camel	Artiodactyla	1	land
1	pig	Artiofabula	1	land
2	sheep	Caprinae	1	land
3	goat	Caprinae	1	land
4	cow	Bovidae	1	land
5	dolphin	Cetacea	1	sea
6	whale	Cetacea	1	sea
7	hippopotamus	Whippomorpha	1	land
8	Caprinae	Bovidae	2	group
9	Cetacea	Whippomorpha	2	group
10	Bovidae	Cetruminantia	3	group
11	Whippomorpha	Cetruminantia	3	group
12	Cetruminantia	Artiofabula	4	group
13	Artiofabula	Artiodactyla	5	group
14	Artiodactyla	NaN	6	group

NetworkX: library for working with graphs

```
# create empty graph in NetworkX
G = nx.Graph()

# adding each table row as a node
for index, row in animals.iterrows():
  G.add_node(row['name'], level=row['level'],
             category=row['category'])

# adding an edge to each node from its parent
for index, row in animals.iterrows():
  if row['parent'] is not np.nan:
    G.add_edge(row['parent'], row['name'])
```

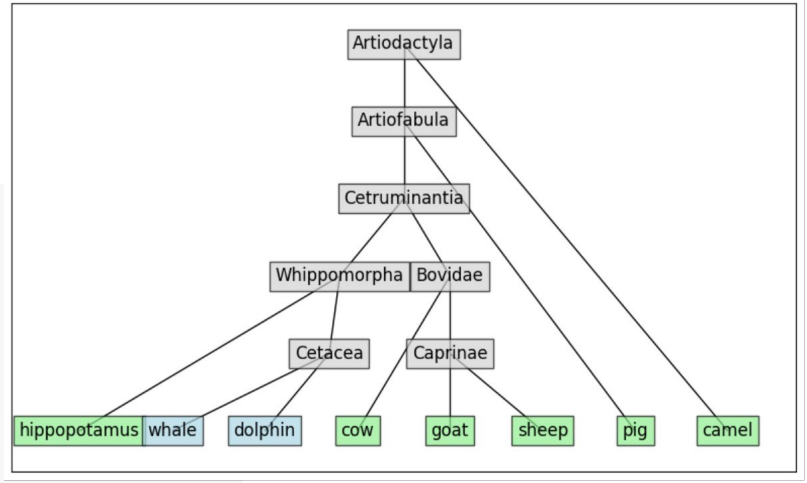
Basic graph drawing in NetworkX

```
# computing coordinates of nodes
coordinates = nx.multipartite_layout(G, subset_key="level",
                                     align='horizontal')
# drawing the graph
(figure, axes) = plt.subplots(figsize=(10, 6))
nx.draw_networkx(G, coordinates, ax=axes)
```



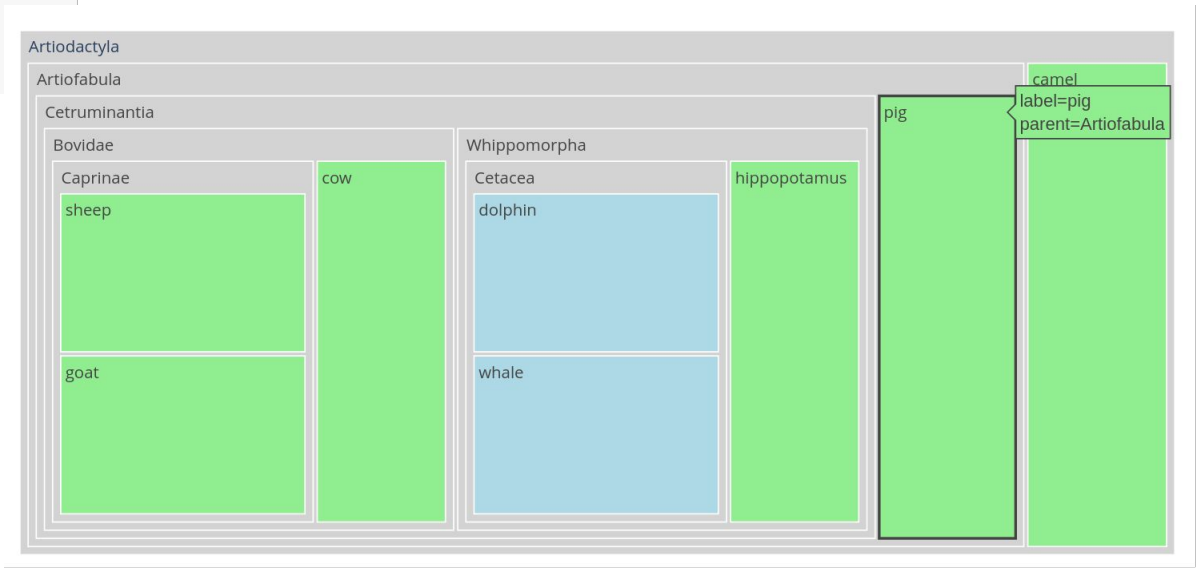
Improving the plot

```
# again compute coordinates and move some of them manually
coordinates2 = nx.multipartite_layout(G, subset_key="level",
                                      align='horizontal')
coordinates2["Whippomorpha"] += (-0.05, 0)
coordinates2["Cetacea"] += (-0.08, 0)

# plot edges only, omit nodes for now
(figure, axes) = plt.subplots(figsize=(10, 6))
nx.draw_networkx_edges(G, coordinates2, ax=axes)

# plot each category of nodes by a different color
color_dict = {'group':'lightgray', 'land':'lightgreen', 'sea':'lightblue'}
for category in color_dict:
  # create a list of nodes on the category
  category_nodes = [v for v in G.nodes if G.nodes[v]['category']==category]
  # select subgraph H of G
  H = G.subgraph(category_nodes)
  # create a dictionary of node label attributes
  label_options = {"ec": "black", "fc": color_dict[category], "alpha": 0.7}
  # draw the node labels as boxes
  nx.draw_networkx_labels(H, coordinates2, font_size=12, b
                          box=label_options, ax=axes)
```



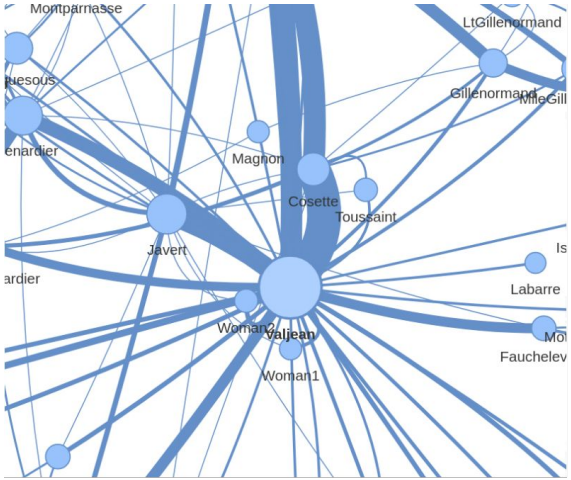
Hierarchy as a treemap in Plotly Express

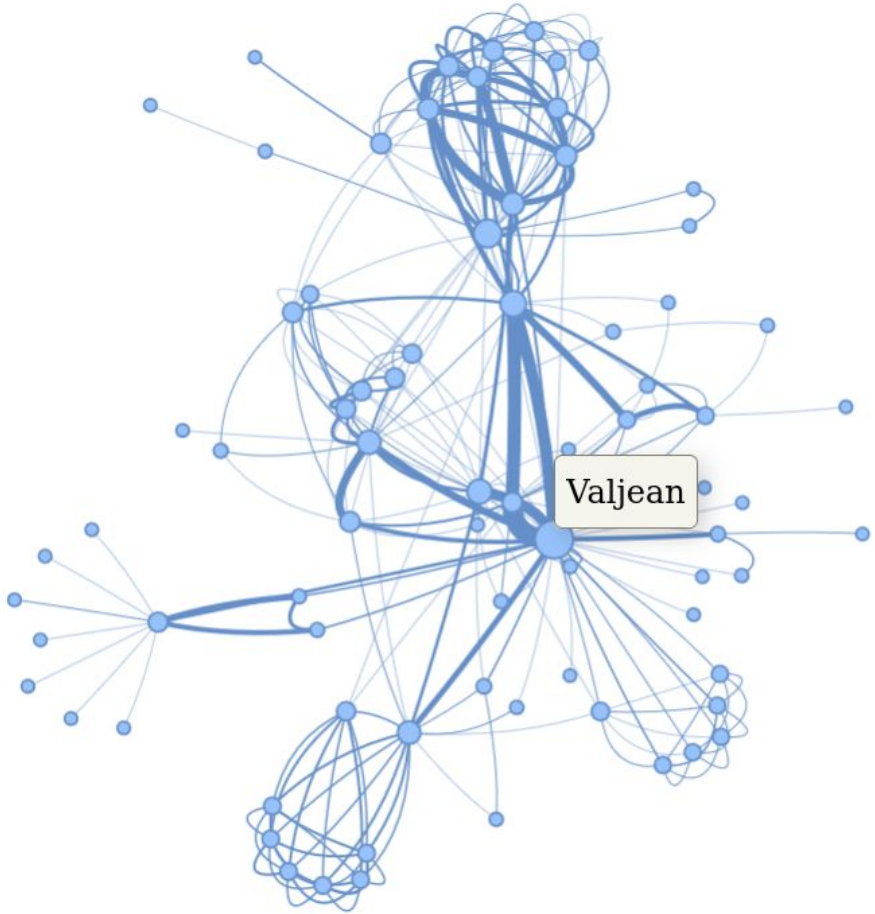
```
import plotly.express as px
fig = px.treemap(
    names=animals['name'],
    parents=animals['parent'],
    color=animals['category'],
    color_discrete_map=color_dict
)
fig.show()
```



Interactive graphs in Pyvis

- Example from NetworkX: character co-occurrence in Les Misérables by Victor Hugo.
- Edges weighted by frequency





```
# initializing an empty network, setup plot properties
pyvis_net = Network("500px", "500px", notebook=True,
                    cdn_resources='in_line')
# loading network from NetworkX
pyvis_net.from_nx(nx.les_miserables_graph())

# get a dictionary of neighbors for each node
neighbors = pyvis_net.get_adj_list()
# add additional node properties
# used as tooltip and size
for node in pyvis_net.nodes:
    node["title"] = node["id"]
    node["value"] = len(neighbors[node["id"]])

# saving the visualization in an html file
pyvis_net.show("net.html")
# displaying the html file in the notebook
from IPython.display import display, HTML
display(HTML('net.html'))
```

Summary of graphs

- Graphs important in **many applications**
- **NetworkX** library has many functions for working with graphs, several layout algorithms for visualization
- **Pyvis** allows interactive visualization
- **Plotly** can visualize trees as **treemaps**

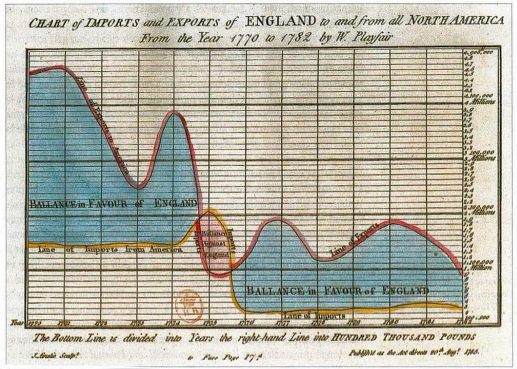
Part III: Time series

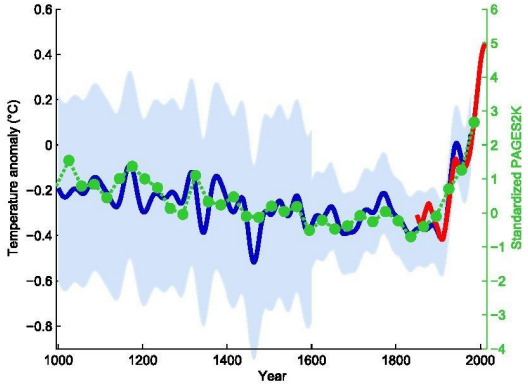
Time series (časové rady)

Sequences of measurements over time (regular or irregular time intervals).

Typically displayed as a **line graph**, with time as x-axis, time flowing from left to right (a cultural convention in western countries).

Other options: bar graphs, heat maps, box plots, ...





Recall:
Playfair 1786,
Mann, Bradley & Hughes 1999

https://commons.wikimedia.org/wiki/File:1786_Playfair_-_Chart_of_import_and_exports_of_England_to_and_from_all_North_America_from_the_year_1770_to_1782.jpg https://en.wikipedia.org/wiki/File:T_comp_61-90.pdf

Typical features of a time series

Overall trend (increasing / decreasing / flat; rate of change),

Seasonality (daily / weekly / yearly cycles),

Noise (general variability / outliers)

Two Google trend time series

Google trends compare frequency
of search terms over time and to each other.

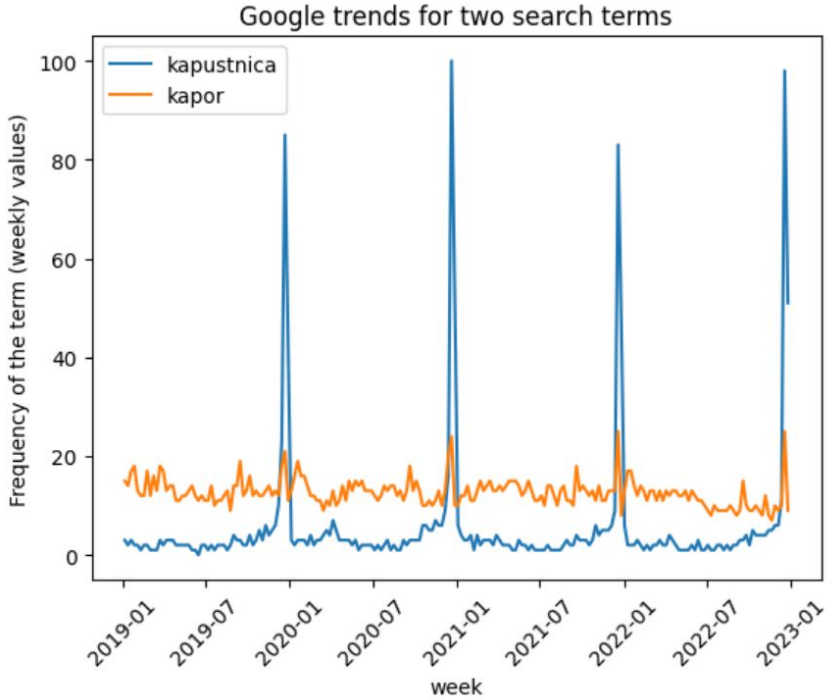
Here we use Christmas-related terms
kapustnica and kapor.

```
display(trends1.head())
```

week	kapustnica	kapor
2019-01-06	3	15
2019-01-13	2	14
2019-01-20	3	17
2019-01-27	2	18
2019-02-03	2	13

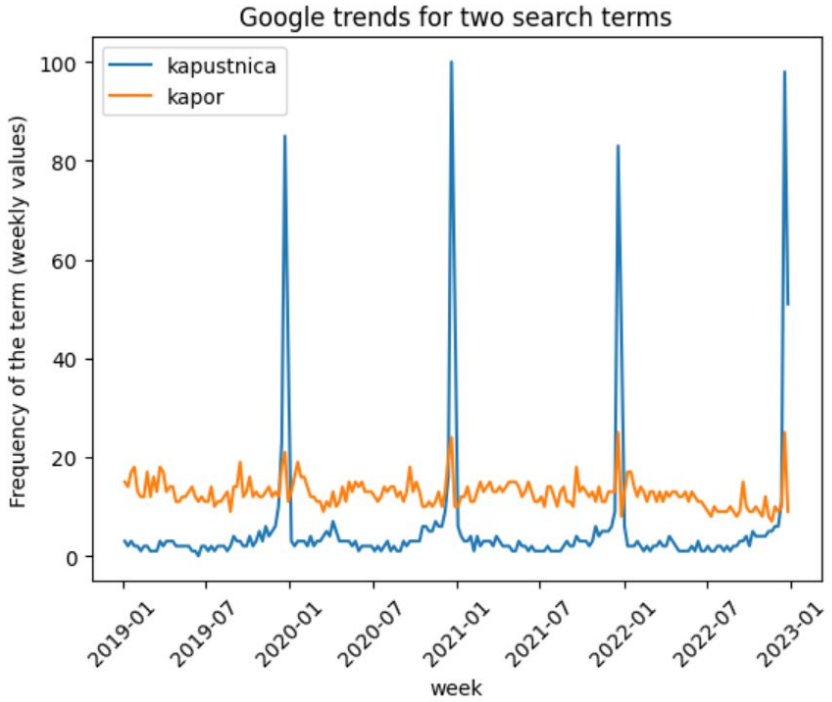
What can we see on the plot?
(trend, seasonality, noise)

Comparison of the two terms?



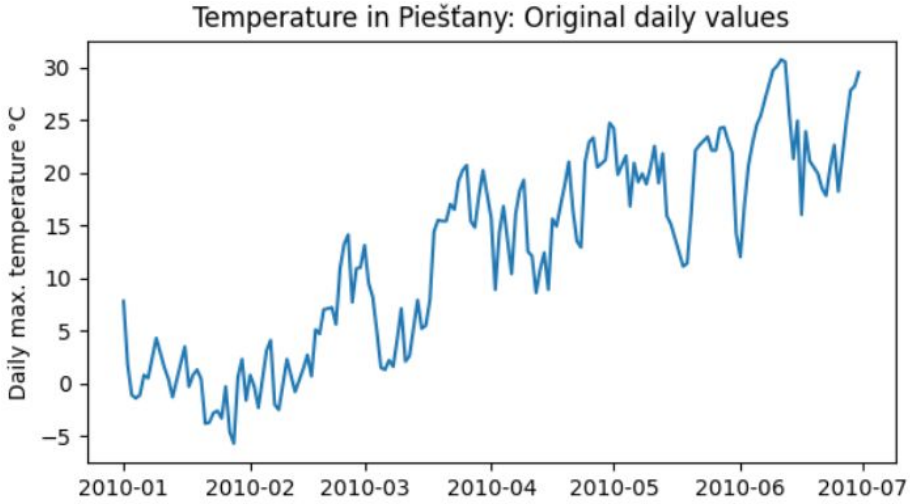

```
axes = sns.lineplot(trends1, dashes=False)
axes.set_ylabel("Frequency of the term (weekly values)")
axes.set_title("Google trends for two search terms")
# rotate tick labels
axes.tick_params(axis='x', labelrotation = 45)
```

Search terms related to kapor: zbgis kataster, zbgis mapa, zgbis, katasterportal list vlastnictva, dážďovka.



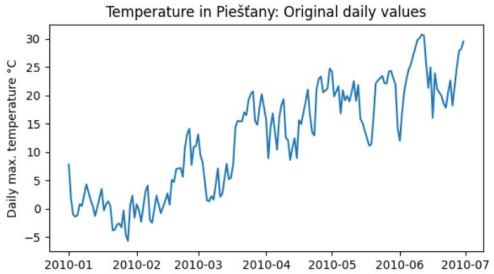
Trend: temperatures are growing in spring

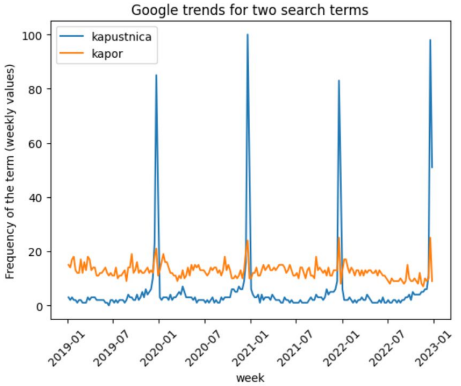
Second dataset:
Maximum daily temperature values from Piešťany January-June 2010, from US National Oceanic and Atmospheric Administration



Smoothing data (vyrovnanie, vyhladenie)

Our time series are quite noisy.





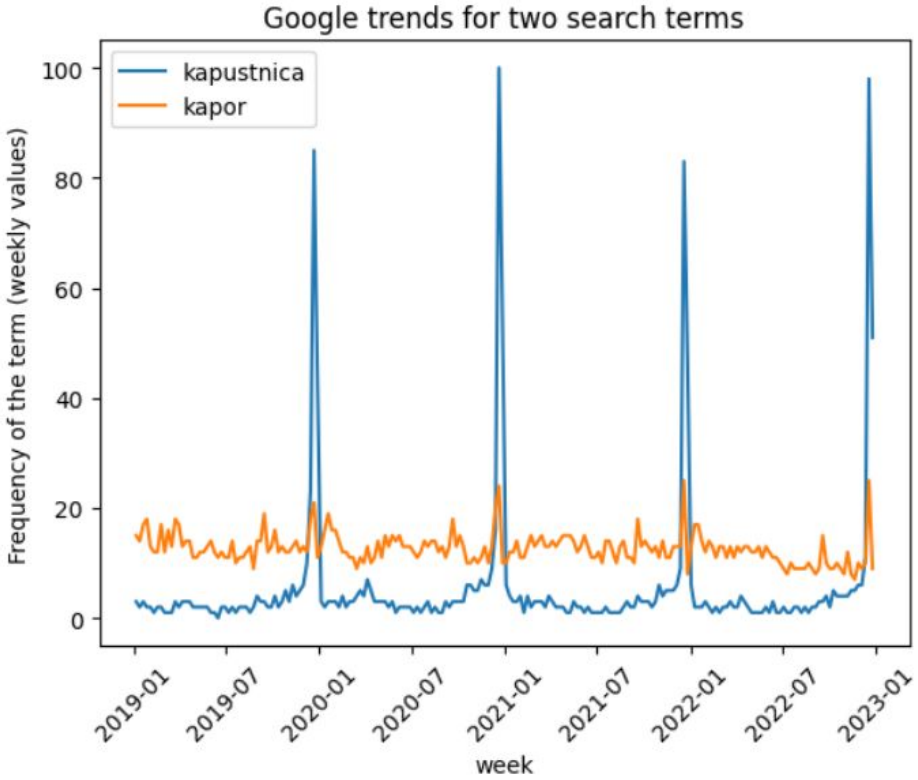
Two options for **smoothing data:**

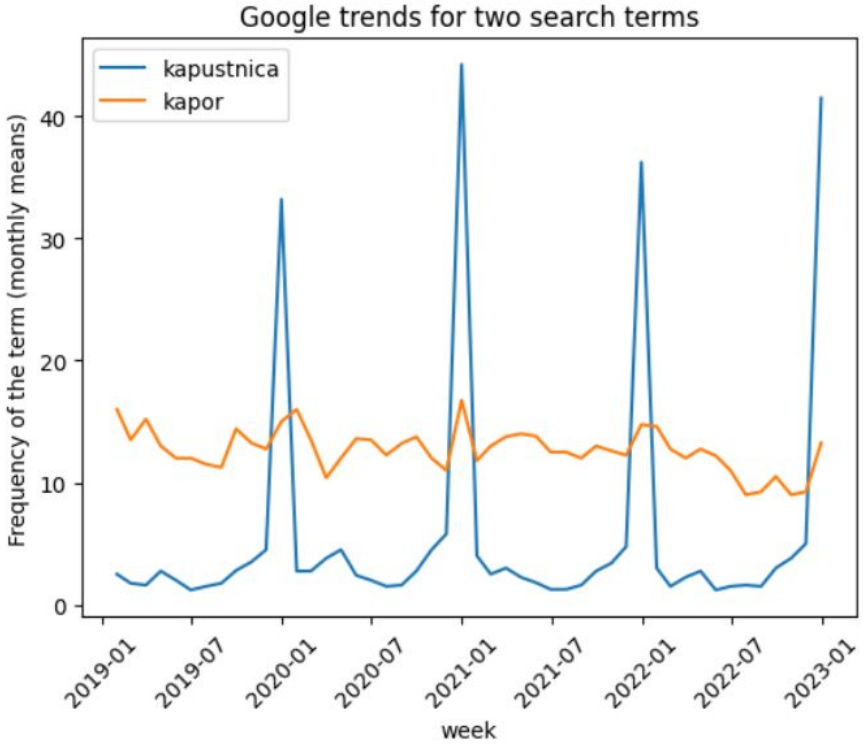
Aggregating them in longer time intervals (e.g. months).

Sliding window (kĺzavé okno): we choose a window size *w* and compute a new series, each value being mean or other summary of *w* consecutive windows in the input.

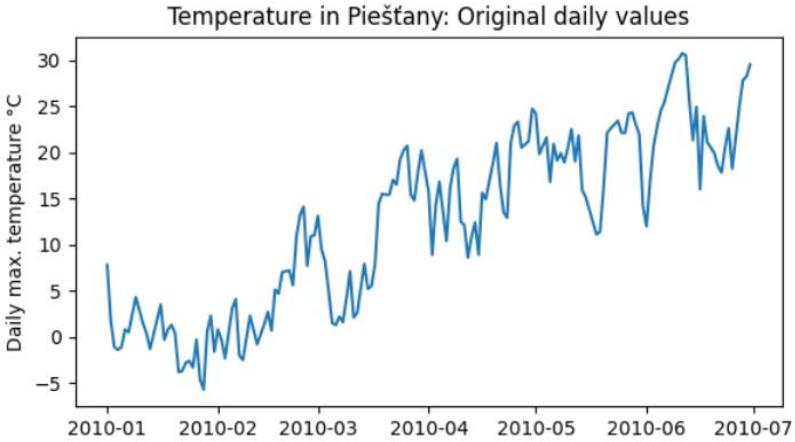
For example with values 2,6,4,2,8,2 and window size 4, we get window means 3.5, 5, 4.

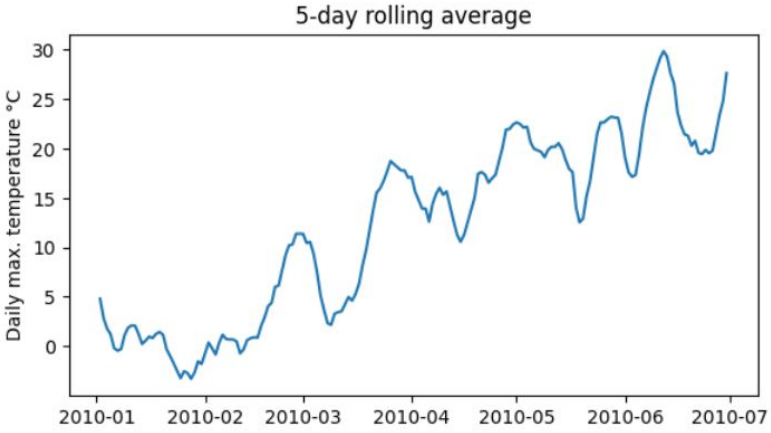
Smoothing Google trends by monthly aggregation

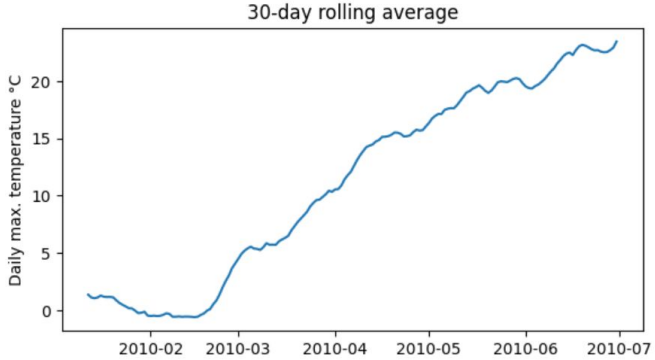




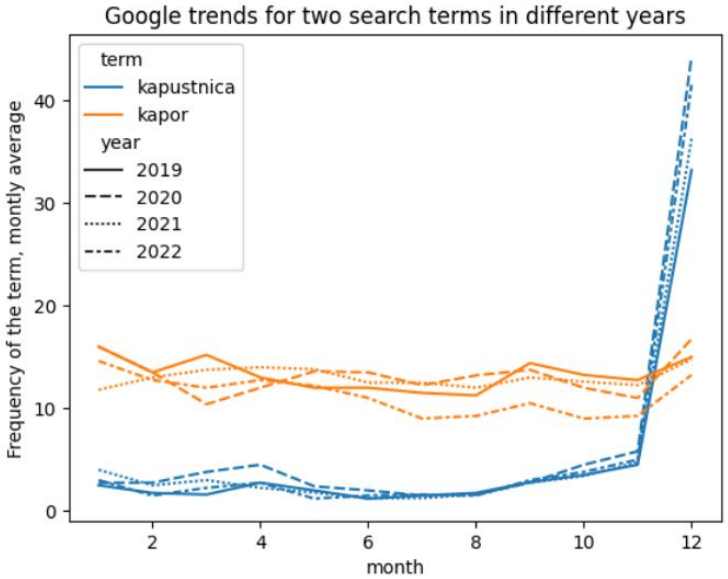
Smoothing temperatures by sliding window

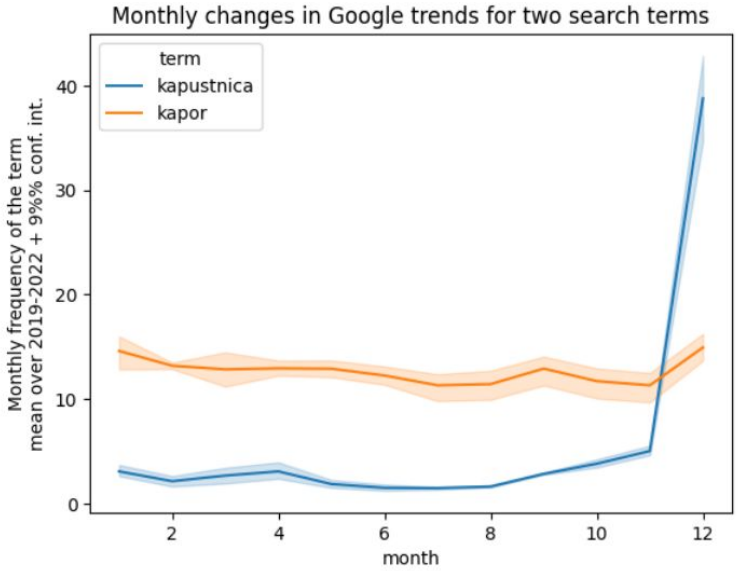






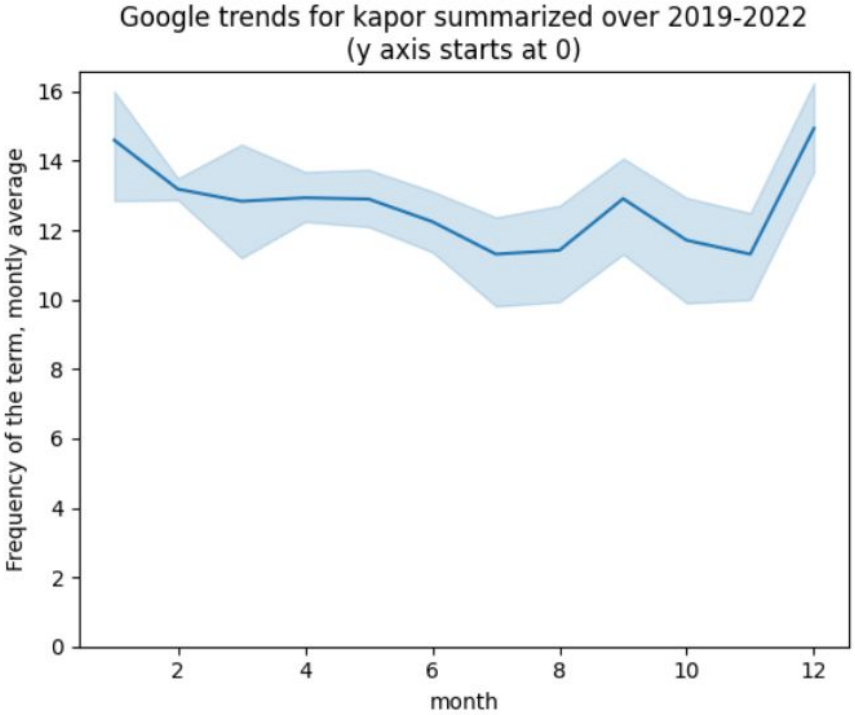
Overlapping timescales to display seasonality / showing uncertainty

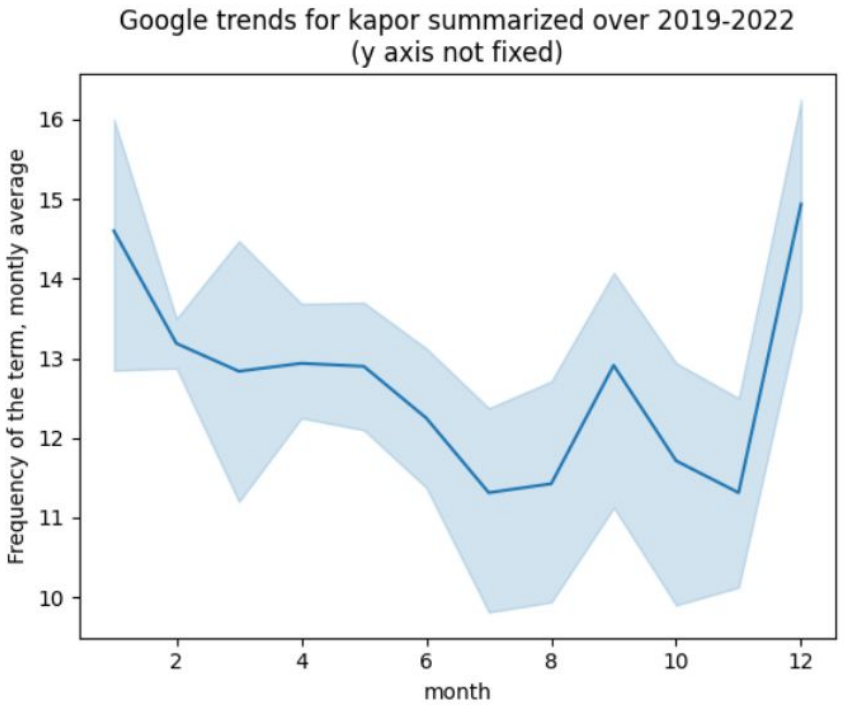




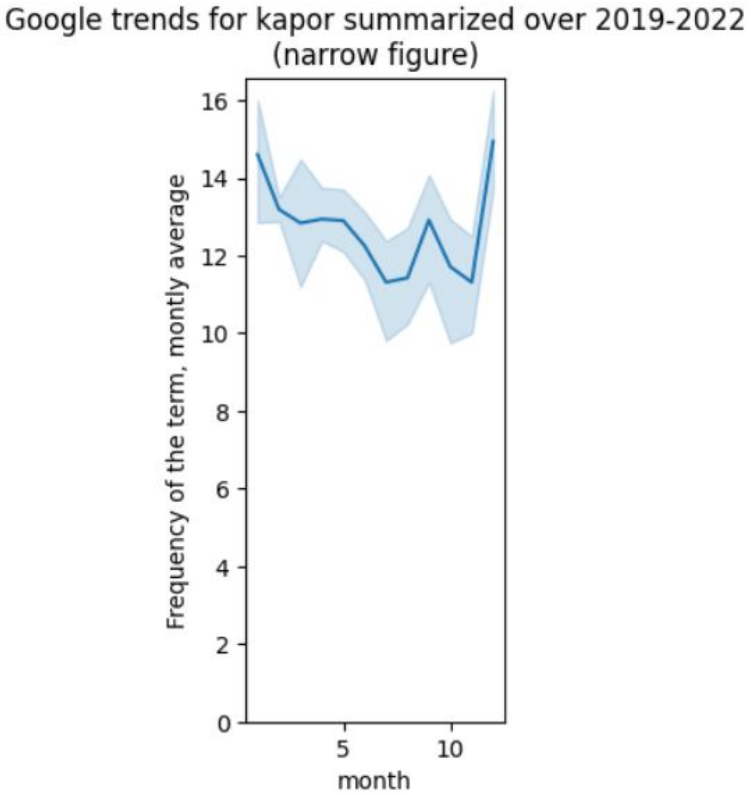
Right: multiple years summarized as the mean and its 95% confidence interval expressing uncertainty in the true value of the mean due to noise in data.

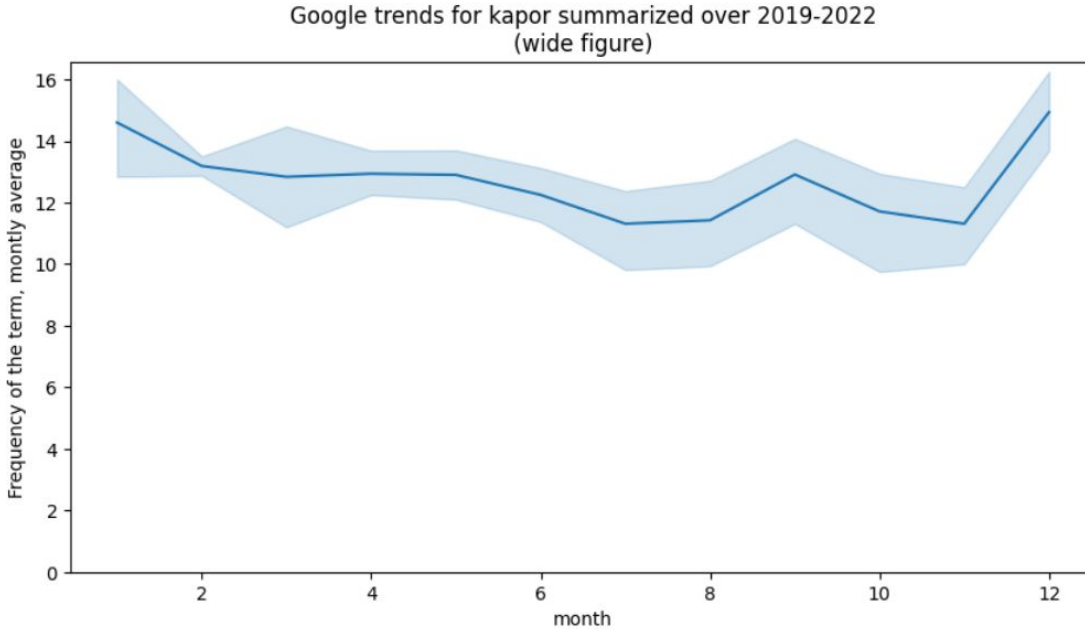
Importance of scales



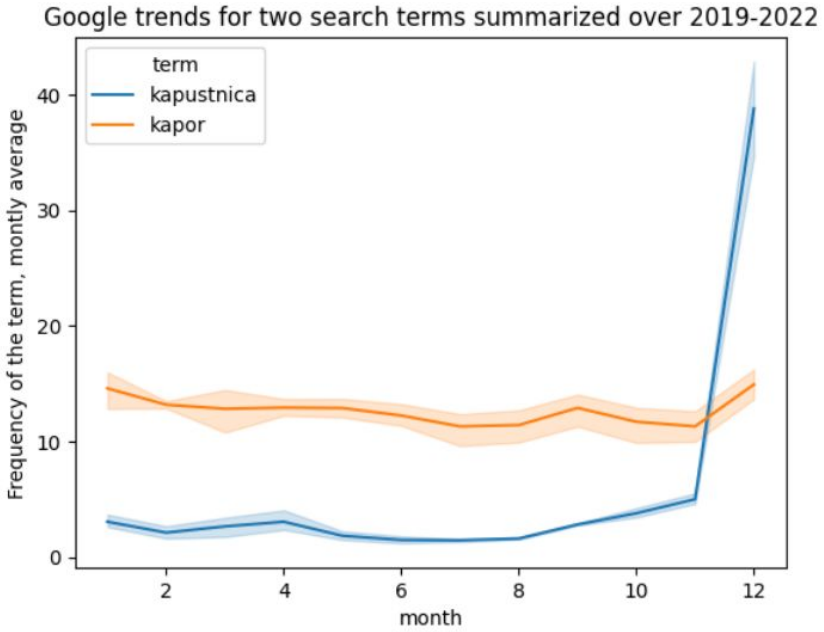


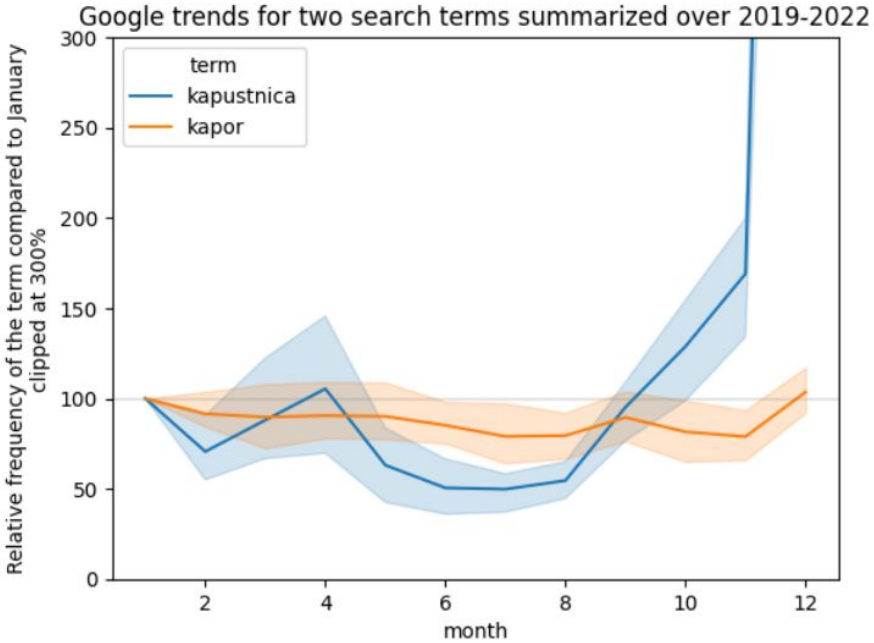
Importance of scales





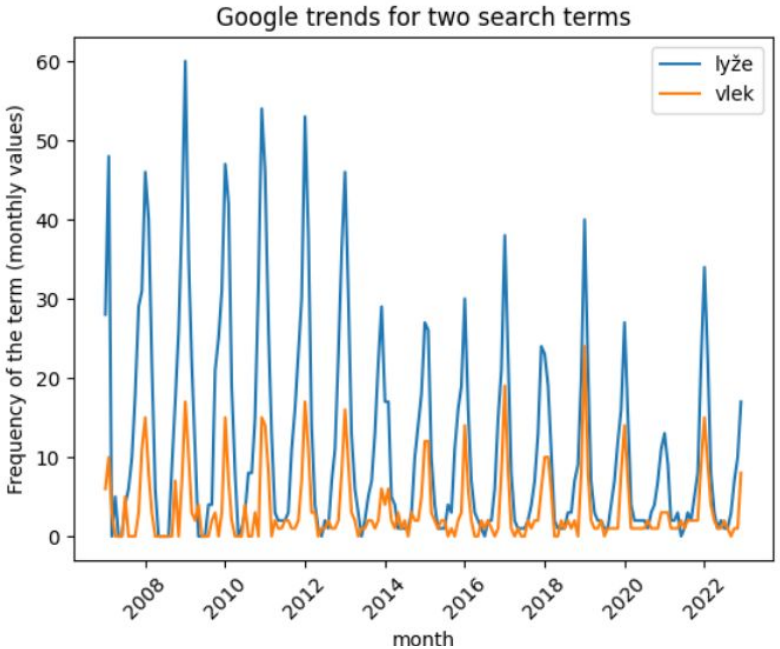
Relative scales

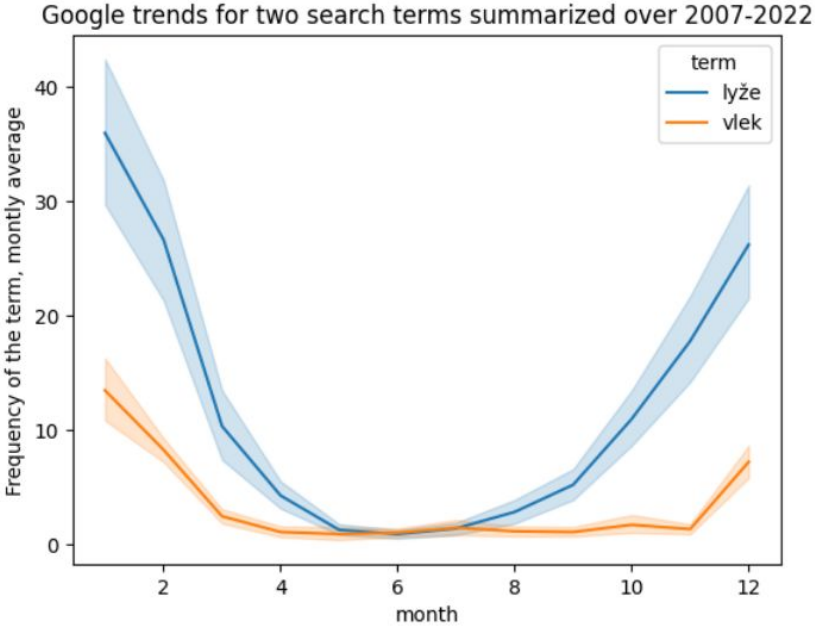




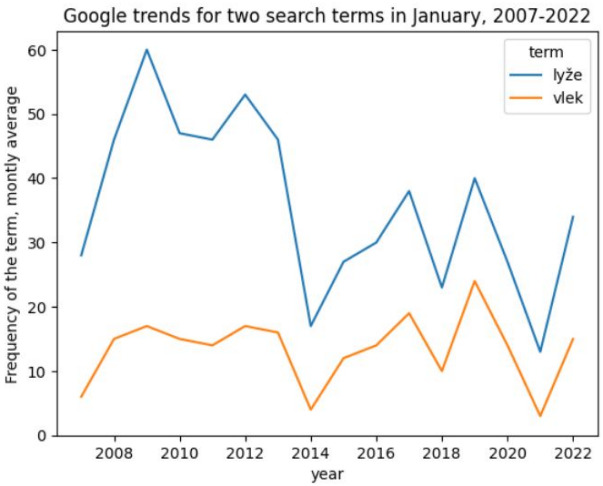
Right: monthly values relative to January
(can be used to compare trends even if overall values vary different)

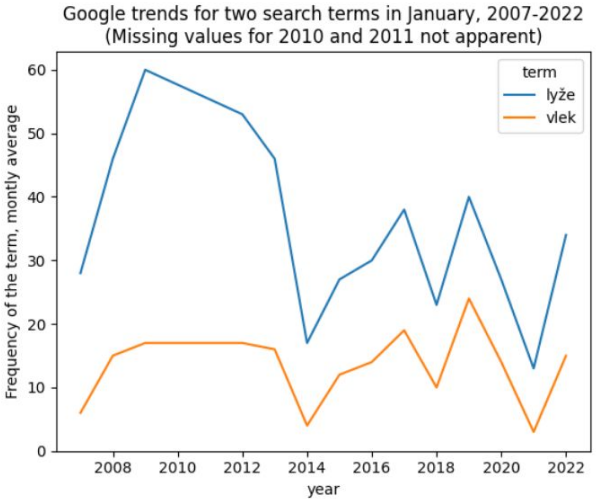
One more pair of Google trend lines

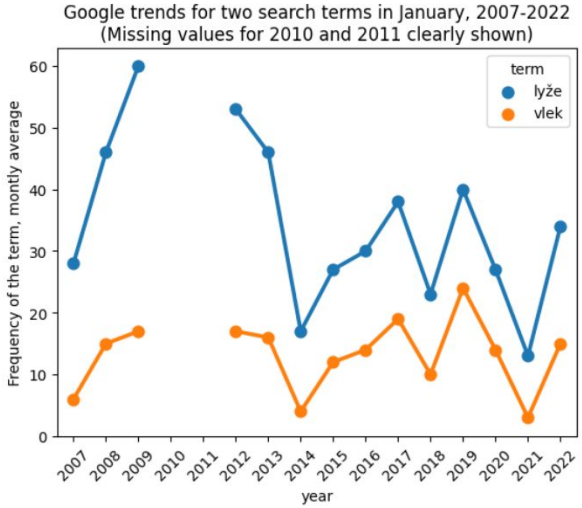




Acknowledging missing values







Middle and right: values for 2010 and 2011 missing
Middle: missing values are not visible, misleading plot
Right: missing values are easy to spot

Summary of time series

Typical goals are to observe and study:

- overall trend (increasing / decreasing / flat; rate of change),
- seasonality (daily / weekly / yearly cycles),
- noise (general variability / outliers)

Useful techniques:

- smoothing by aggregation and sliding window
- overlapping timescales
- relative scales
- showing uncertainty and missing values