Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod

Vizualizace dat v Python

Bakalářská práce

Autor: Matěj Kolář
Studijní obor: Aplikovaná informatika

Vedoucí práce: Mgr. Jiří Haviger, Ph.D.

Hradec Králové Březen 2023

Prohlášení:

Prohlašuji, že jsem bakalářskou/diplomovou práci zpracoval/zpracovala samostatně a s použitím uvedené literatury.

V Hradci Králové dne 9.4.2023 Matěj Kolář

Poděkování:

Děkuji vedoucímu bakalářské práce Mgr. Jiřímu Havigerovi, Ph.D. za metodické vedení práce, množství cenných rad, podmětů, připomínek a čas, který mi věnoval při řešení dané problematiky.

Anotace

Bakalářská práce se zabývá popisem základních grafů a jejich implementací ve vizualizačních ekosystémech Matplotlib, Plotly a Vega.

Annotation

Title: Data visualization in Python

This bachelor's thesis deals with the description of basic graphs and their implementation in the visualization ecosystems Matplotlib, Plotly and Vega.

Obsah

Seznam obrázků

Seznam ukázek kódu

1 Úvod

Dle Tiobe [12] a PYPL [13] indexu se jazyk Python těší velké popularitě napříč odvětvími a není tak divu, že pro něj existuje spousta knihoven. Python Package Index [14] momentálně hlásí více než 418 000 balíčků, číslo, které v nejbližší době jen poroste. Pro účely této práce se budeme zajímat hlavně o knihovny, které implementují ekosystémy Matplotlib, Plotly a Vega. Tyto tři ekosystémy se zabývají, stejně jako tato práce, vizualizací dat.
Pro jeden graf z každé kategorie (dle Python graf gallery [3]) byla provedena implementace za pomocí knihoven výše uvedených ekosystémů.

Cílem práce je poskytnout čtenáři pohled na všechny tři ekosystémy a usnadnit mu tak rozhodování, který z ekosystémů nejlépe vystihuje jeho požadavky a preference.

2 Data a jejich specifikace

Vzhledem k povaze počítačů jsou všechna data nespojitá v čase a nespojitá v hodnotě. Vysokou vzorkovací frekvencí a vysokým rozlišením lze dosáhnou pouze zdánlivé spojitosti v čase a hodnotě.

2.1 Struktura

Dle struktury lze data rozdělit na data strukturovaná a nestrukturovaná.
Lidé i stroje produkují různé formy strukturovaných a nestrukturovaných dat.

Data strukturovaná

Základní charakteristikou strukturovaných dat je snadná prohledávatelnost. Strukturovaná data jsou uložena v databázi nebo v data warehouse. Nejčastějším využitím pro tento typ dat mají CRM a ERP systémy, inventární systémy, rezervační systémy, bankovní systém a podobně. Příkladem strukturovaných dat jsou telefonní čísla, časové údaje, uživatelská jména, názvy produktů…

Data nestrukturovaná

Nestrukturovaná data nemají předem definovanou podobu, a proto je jejich prohledávatelnost špatná. Jsou uložena přímo v aplikacích, NoSQL databázích, v data lake, nebo v data warehouse. Nestrukturovaná data jsou například využívána v prezentacích, nástrojích pro prohlížení a úpravu médií, datových pipelines a dalších aplikacích, které jakkoliv operují s nestrukturovanými daty. Příkladem nestrukturovaných dat jsou textové dokumenty, audio záznamy, fotografie, video záznamy nebo nijak nezpracovaná data.

2.2 Interpretace

Správná interpretace dat je důležitá část jejich zpracování. Chybná interpretace dat může vést k podivným či zavádějícím výsledkům. Příkladem může být misinterpretace časového razítka jako číselné hodnoty. Půlnoc 7. června 2001 se tak změní na číslo 991872000.
Chybná interpretace může nastat i u samotného datového typu, číslo 255 v typu uint8 lze špatně interpretovat jako -1 v int8.

3 Vizualizace

Vizualizace dat je způsob interpretace datových sad pomocí grafických obrazců, za účelem prezentace dat, zvýšení čitelnosti dat, či odhalení jinak nezjevných výsledků. Python není jediný jazyk, schopný vizualizace dat. Mezi další používané jazyky patří R, Scala nebo Matlab.

3.1 Druhy grafů

Grafy mohou být statické, animované nebo interaktivní. Každý typ má své využití, výhody a nevýhody.

Statické grafy. Hlavní výhodou je jejich jednoduchost, nízká náročnost implementace a všestrannost. Grafy je možné skladovat jako obrázky, a nevyžadují tak back-end, který by je neustále renderoval. Nevýhodou je absence možnosti interakce, která velmi usnadní čtení hodnot z grafu.

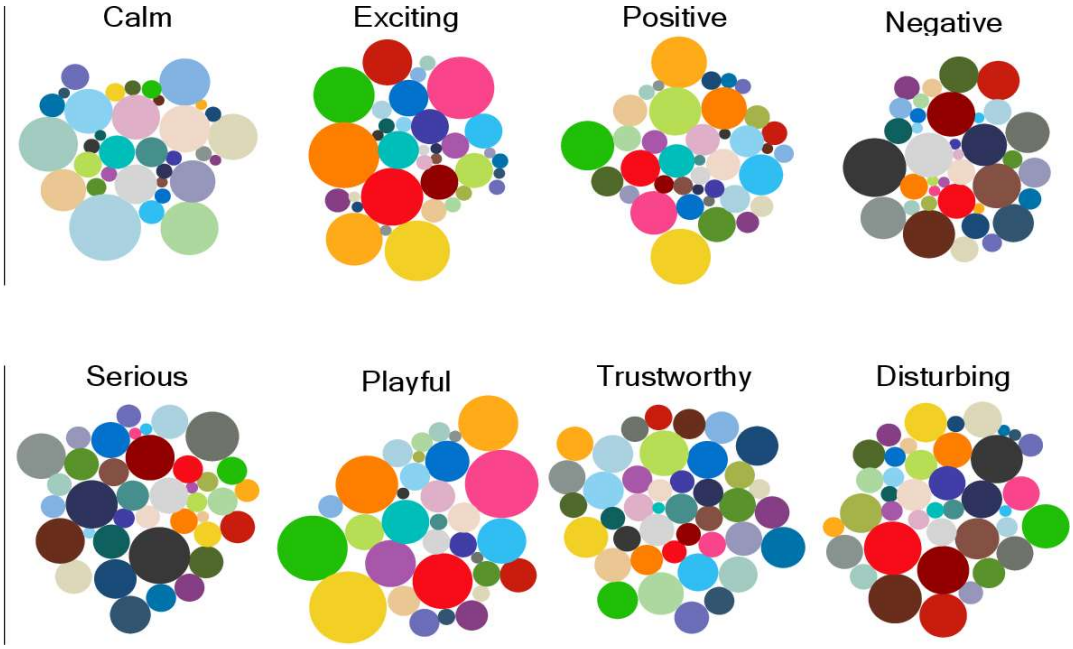
Animované grafy mohou být použity k zpestření prezentace či k zobrazení velké sady na malém prostoru. Nejvhodnější využití je pro série dat v čase. Potřebují back-end který je bude renderovat, nebo musí být uloženy do formátu, který umožňuje přehrávání animace (gif, mp4, mkv, webm).

Interaktivní grafy jsou ideální volba pro dashboardy. Umožňují precizní selekci dat a rozsahů, mohou být pravidelně aktualizovány. Avšak jsou několikanásobně náročnější na vytvoření oproti statickým grafům. Vyžadují back-end který je bude neustále renderovat a obsluhovat vstupy od uživatele.

3.2 Design

Barvy

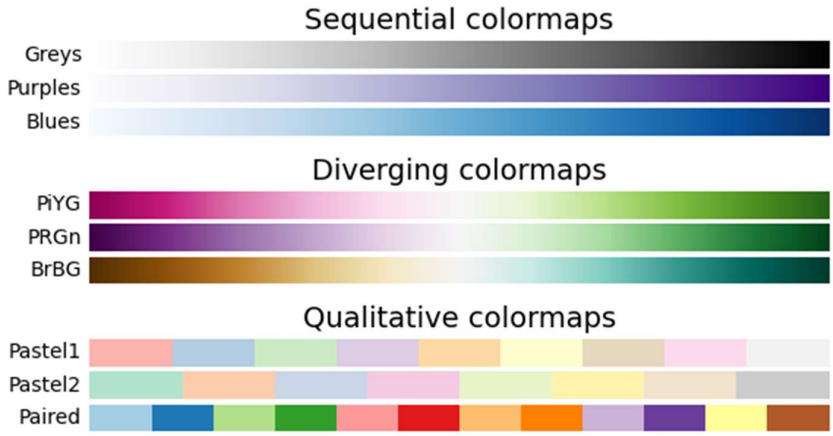
Důležitou součástí grafu je i jeho barevné schéma. Barevné schéma může ovlivnit čitelnost a přehlednost grafu nebo pocity které graf vyvolává.



Obr. 1 Pocity spojené s barvami
Zdroj: Patra Abhisekh. Affective color palettes in Visualization (2017).

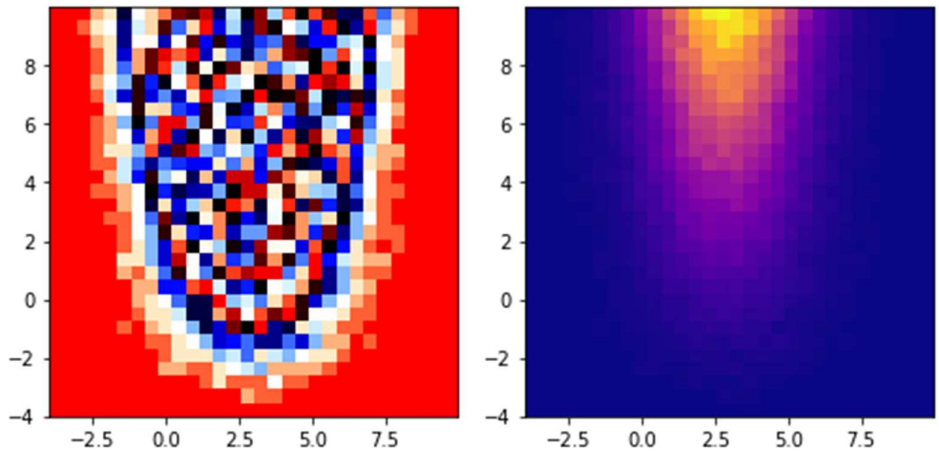
Při výběru barev je důležité vzít v potaz i cílovou skupinu. Příkladem může být graf určený osobám s částečnou barvoslepostí (Protanopie, Tritanopie atd.). V takovém případě není vhodné volit barvy, které daná skupina nerozpozná od sebe.

Všechny tři ekosystémy mají již nadefinované názvy pro některé barvy. Uživatel tak nemusí vkládat barvy pomocí hexadecimálních klíčů. Stejně tak jako jednotlivé barvy, mají ekosystémy připravené i mapy(palety) barev pro případy, kdy jsou data spojitá v hodnotě. Takovéto mapy mohou být sekvenční, divergující nebo kvalitativní. Ukázky těchto map lze vidět na následujícím obrázku (ve stejném pořadí od shora).



Obr. 2 Barevné mapy – ukázky
Zdroj: Dokumentace Matplotlib [4], upraveno na 3 ukázky z každé kategorie

Důsledkem zvolení nevhodné barevné palety trpí nejen estetická stránka grafu, ale i jeho schopnost předávat informace. Na následujícím obrázku lze vidět nevhodnou (levá strana) a vhodnou (pravá strana) selekci barev.



Obr. 3 Nevhodná a vhodná paleta
Zdroj: vlastní zpracování pomocí Matplotlib

4 Datové struktury v jazyce Python

Datové struktury lze rozdělit na dvě velké kategorie, dle toho, zda jsou Nativní či nikoliv. Všechny ukázky kódu v této kapitole jsou zpracovány autorem práce.

4.1 Nativní datové struktury

Tuple

Tuple je jednoduché pole, indexované numerickou hodnotou se začátkem v čísle nula. Typ tuple nelze jakkoliv měnit. Příklad definice tuple:

```
data = (1,0,1,2)
print(data[0]) # vypíše 1
```

Kód 1 – Tuple

List

List, je stejně jako tuple, pole indexované numerickou hodnotou, lze však jeho obsah měnit a jeho velikost je plně dynamická. Příklad definice a modifikace listu:

```
data = [1,0,1,2]
print(data[0]) # vypíše 1
data[0] = 5
print(data[0]) # vypíše 5
```

Kód 2 – List

Dictionary

Dictionary, často zkráceně dict je pokročilý způsob, jak uchovávat data. Na rozdíl od tuple a list je indexované klíčem, který je nutné specifikovat. Klíč musí být unikátní. Data v dictionary lze měnit dle libosti. Velikost není nijak omezená.
Příklad definice dictionary:

```
data = {"A": 1, "B": 0, "C": 1, "D": 2}
print(data["A"]) # vypíše 1
data["A"] = 5
print(data["A"]) # vypíše 5
```

Kód 3 – Dictionary

4.2 Externí datové struktury

Datových struktur z balíčků je podstatně více, a proto se zaměříme pouze na důležité struktury z balíčků Numpy a Pandas.

Balíček NumPy

Balíček NumPy se zabývá hlavně operacemi s čísly, avšak omezeně podporuje i text. Na pozadí využívá pro operace jazyk C. Díky tomu lze provádět identické operace rychleji než za pomocí nativních datových typů.
Balíček je ve vývoji od roku 2005. V současné je době je vyvíjený širokou komunitou jako open-source projekt.

Ndarray

Jednoduché modifikovatelné pole, indexované numerickou hodnotou. Avšak narozdíl od nativního listu není obsah dynamicky typovaný. Pro efektivní operaci je potřeba při vytváření nastavit, o jaký datový typ se jedná. Kombinací několika polí lze vytvářet matice, pro které jsou již implementované aritmetické operace.
Příklad definice a modifikace ndarray:

```
import numpy
array = numpy.array((1,0,1,2),dtype=numpy.int32)
print(array[0]) # vrátí 1
array[0] = 10
print(array[0]) # vrátí 10
```

Kód 4 – NumPy array

Balíček Pandas

Balíček Pandas slouží k zpracování a modifikaci datových sad. Pro některé operace využívá NumPy. Data je možné zpracovat i bez tohoto balíčku, avšak náročnost implementace takového zpracování drasticky stoupne.
Balíček je vyvíjený od roku 2008. Od roku 2009 jako open-source projekt. Momentálně je udržovaný a vyvíjený komunitou.

Dataframe

Pandas dataframe je jeden z nejužitečnějších datových typů určených ke zpracování dat. Lze si ho představit jako 2D tabulku s daty. Nejsnazší způsob vytvoření dataframe je jeho načtení z externího souboru (csv, txt, xls a další) pomocí funkce read.

Ukázka vytvoření dataframe:

```
import pandas
dataframe = pandas.DataFrame({"sloupec_a": [50, 10, 30],
                              "sloupec_b": [10, 80, 60]}) #vytvoření hodnot v programu
#načtení dat ze souboru
dataframe = pandas.read_csv("data.csv")
dataframe = pandas.read_xml("data.xml")
dataframe = pandas.read_excel("data.xls")
```

Kód 5 – Pandas DataFrame

Series

Datový typ určený pro skladování hodnot v čase. Series lze konvertovat na dataframe i vytvořit z dataframe.

Ukázka převodu mezi dataframe a series:

```
series = dataframe.squeeze() # dataframe >> series
dataframe = series.to_frame() # series >> dataframe
```

Kód 6 – Pandas Series a DataFrame konverze

5 Stručný popis cesty dat

Prvními kroky je samotný sběr dat a jejich případná digitalizace. Výsledkem těchto prvních kroků je soubor dat v digitální podobě, nejčastěji se jedná o formáty jako csv či xls.

Dalším krokem je načtení těchto dat do datových struktur, se kterými dokáží vizualizační balíčky pracovat. Pro načtení lze použít balíčky jako například Numpy či Pandas.

Před samotnou vizualizací je občas potřeba data vyčistit. Od případných duplikátů, prázdných řádků či jiných neduhů nám většinou pomohou již existující funkce balíčků.

Na vyčištěné sadě dat lze provádět různé operace, jako například přetypování časových údajů, zahození nepotřebných sloupců, převzorkování další.

Takto zpracovaná data jsou poté vizualizována pomocí vizualizačních balíčků.

6 Ekosystémy

6.1 Matplotlib

Jedna z nejpoužívanějších knihoven pro vizualizaci dat v jazyce Python. Celá knihovna je open source a volně dostupná na internetu. Původně byla knihovna vyvíjená Johnem Hunterem, nyní se o vývoj a údržbu stará MDT (Matplotlib Development Team). MDT je skupina uživatelů, která projekt převzala v roce 2012. Matplotlib je jako jediný z porovnávaných ekosystémů nativní pro jazyk Python.
V současné době má projekt na platformě GitHub 16 900 hvězd [16].

6.2 Plotly

Open source knihovna vyvíjená od roku 2012 společností Plotly. Jedna z hlavních výhod je interaktivita a dostupnost nejen v Pythonu. Plotly ekosystém má varianty knihoven, vyvíjené přímo společností Plotly, pro různá využití. Plotly a Plotly Express pro snadné tvoření interaktivních grafů. Plotly dash pro tvorbu dashboardů. Dash Enterprise a Chart Studio Enterprise pro komerční využití. Verze knihovny pro python Plotly.py má momentálně na GitHubu 13 000 hvězd [18].

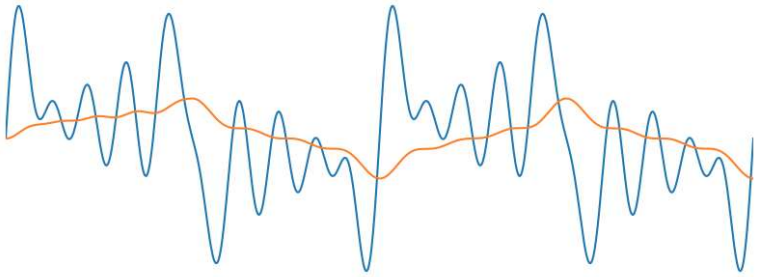
6.3 Vega

Vega visualization grammar, zkráceně pouze Vega je jeden ze základních open source nástrojů, které jsou vyvíjeny komunitou Vega project. Vega má vlastní deklarativní jazyk, který slouží k vytváření jednoduchých vizualizací. Pro propojení s Pythonem je využívána knihovna Altair. Vega samotná má na GitHubu 10 000 hvězd [17]. Knihovna Altair 8 100 hvězd [15].

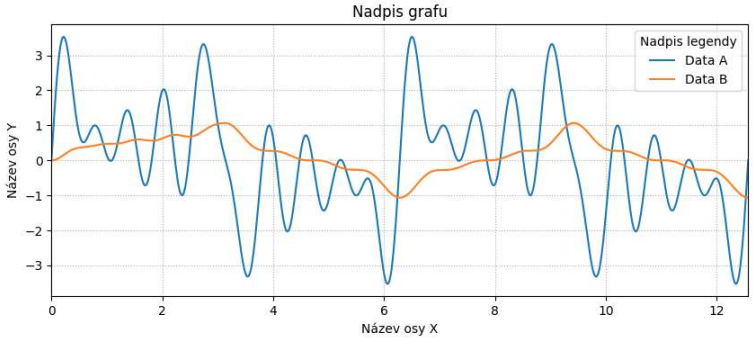
7 O grafu

7.1 Topologie grafu

Graf bez jakéhokoliv označení a kontextu je pouze shluk tvarů a čar. Proto je nutné využívat nadpisů, legend a dalších funkcí, které knihovny nabízí, k vytvoření grafu, který bude schopen předat onu klíčovou informaci.



Obr. 4 Graf bez kontextu
Zdroj: vlastní zpracování pomocí Matplotlib



Obr. 5 Graf s kontextem
Zdroj: vlastní zpracování pomocí Matplotlib

Předešlé dva grafy reprezentují identická data. Jeden však tak činní bez jakéhokoliv popisku. Již na první pohled si tak lze všimnout, že druhý graf předává mnohem více informací než graf první. Informace jako názvy os, nadpis grafu, názvy jednotlivých dat nelze z prvního grafu získat.

Nadpis grafu nás informuje, o jaké data se jedná. Názvy os a osy samotné mohou sloužit k informování čtenáře o veličinách a jejich hodnotě. Legenda je pro grafy zobrazující více řad nepostradatelná, díky ní lze lépe identifikovat jednotlivé řady dat.

7.2 Kategorie grafu

Distribuční grafy

Distribuční grafy slouží k vizualizaci výskytu hodnot. Hodnoty mohou být reprezentovány jednotlivě, nebo jako skupiny více hodnot.

Korelační grafy

Korelační grafy jsou využívány pro zobrazení a nalezení závislostí mezi hodnotami. Později mohou být využity také k predikci hodnot.

Žebříčky

Žebříčky jsou využívány pro porovnání a zobrazení pořadí hodnot.

Části celku

Tyto grafy slouží k reprezentaci poměru hodnot v jeden moment.

Vývoj v čase

Grafy z kategorie vývoj v čase slouží k vizualizaci pohybu hodnoty, či více hodnot v časovém úseku.

Mapy

Grafy určené k zobrazení hodnot v závislosti k oblasti mapy. Oblasti mohou být libovolně velké v závislosti na reprezentované informaci.

Flow

Do této kategorie patří grafy, jejichž účelem je znázornit cestu (proud – flow) nebo také síť. Jedním z takových grafů je například graf, z pohledu diskrétní matematiky. Množina propojených uzlů, může sloužit jak k reprezentaci sítě, tak cesty mezi body.

8 Metodika zpracování

8.1 Data použitá pro ukázky

Nathan Yau ve své knize „Visualize This: The Flowing Data Guide to Design, Visualization, and Statistics“ říká: „*Data jsou to, co dělá vizualizaci zajímavou. Pokud nemáte zajímavá data, skončíte u zapomenutelného grafu nebo hezkého, ale zbytečného obrázku“* [2]. Pro naše účely není zajímavost dat nijak podstatná. Proto jsou data pro ukázky 1-4 generována náhodně v moment spuštění programu. Pro zachování konzistence, je v generátoru dat nastavený seed s hodnotou „2022“.

Pro vizualizaci geografických dat (ukázka 5) bylo potřeba dvou rozdílných datových sad:

- Soubor tvarů pro vykreslení mapy – obsahuje geometrii krajů.
- Datová sada nezaměstnanosti v ČR – obsahuje samotná data.

Soubor tvarů je volně dostupný na stránce projektu Natural Earth [7]. Data o nezaměstnanosti pochází z veřejné databáze ČSÚ [11].

Pro vizualizaci sítě (ukázka 6) jsou data generována knihovnou networkx. Tato data nejsou náhodná, výsledná síť je vždy K4.

Pro dashboard (ukázka 7) je využívána datová sada „Countries CO2 Emission and more…“ od uživatele „Benjamin Vanous“ [10]. Tato sada je složena z několika menších sad a je volně přístupná na webu kaggle.com.
Jednotlivé menší sady jsou dostupné na webové stránce úřadu pro energetické informace USA (EIA): https://www.eia.gov/international/data/world

8.2 Struktura ukázek

Všechny ukázky jsou vypracované jakožto Jupyter notebooky a mají následující strukturu:

- Sekce „Sdílený kód“ - kód v této buňce slouží k přípravě dat a implementaci funkcí, které jsou použity ve všech ukázkách.
- Sekce „Matplotlib“ – v této buňce se nachází pouze kód, který je potřebný pro vykreslení grafu v ekosystému Matplotlib
- Sekce „Plotly“ – Kód, který lze nalézt v této sekci, se stará o vykreslení pomocí ekosystému Plotly
- Sekce „Vega“ – Implementace vykreslení grafu pomocí ekosystému Vega.

Všechen kód je okomentován, samotné soubory Jupyter notebooky jsou dostupné jako součást digitální verze práce nebo v sekci příloh této práce.

9 Ukázky a zhodnocení ekosystémů

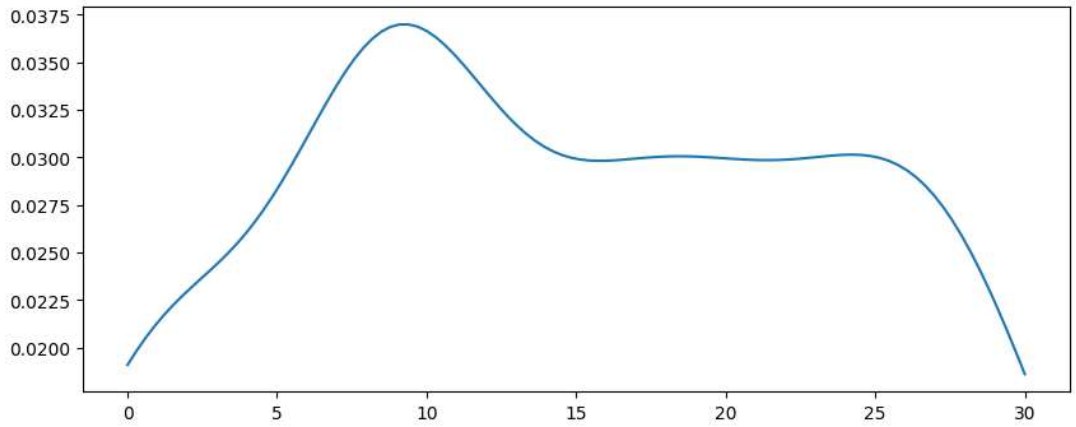
V jednotlivých částech této kapitoly lze nalézt ukázky vybraných grafů a subjektivní zhodnocení vzájemné náročnosti implementace mezi ekosystémy. Celé notebooky s ukázkami lze najít mezi přílohami.

9.1 Density chart

Density chart patří do kategorie distribučních grafů. Cílem grafu je vizualizovat rozložení hodnoty do jedné či více skupin. Rozdíl v náročnosti implementace je zanedbatelný, jednotlivé ekosystémy přistupují k vykreslení podobně.

Matplot

```
from matplotlib import pyplot as plt

plt.figure(figsize=(10, 4), dpi=100) #10 inch * 100 dpi = 1000px
plt.plot(data["x"],data["y"])
plt.show()
```



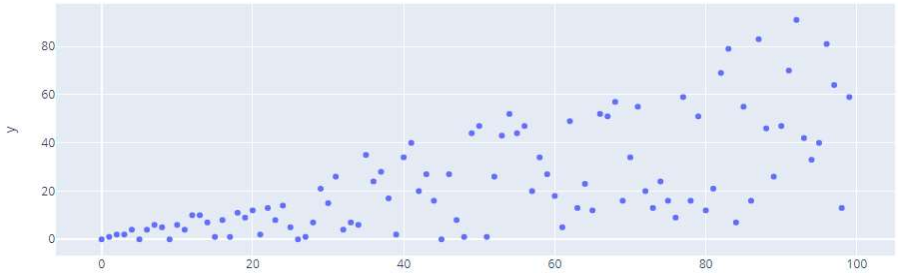
Obr. 6 Density chart a implementace
Zdroj: vlastní zpracování pomocí Matplotlib

9.2 Bodový graf

Bodový graf z kategorie korelačních grafů je používán k zobrazení vztahů mezi dvěma hodnotami, každý bod reprezentuje jednu dvojici hodnot. Rozdíl v náročnosti implementace je minimální.

Plotly

```
import plotly.express as px
import plotly.io as pio
pio.renderers.default="notebook"

fig = px.scatter(data,x="x",y="y", width=1000, height=400)
fig.show()
```



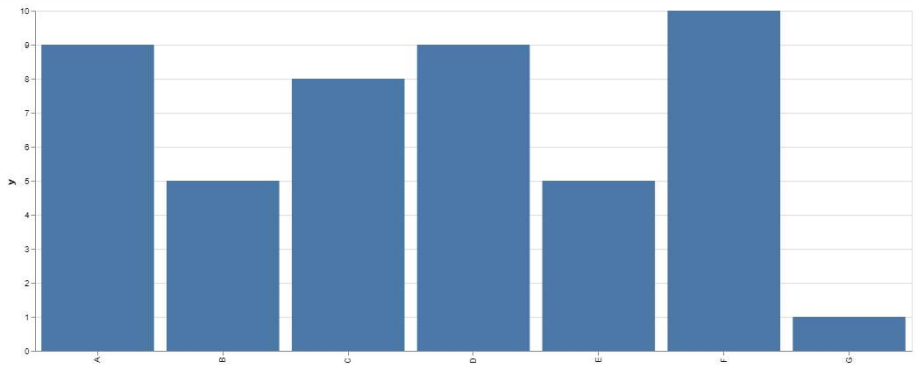
Obr. 7 Bodový graf jeho implementace
Zdroj: vlastní zpracování pomocí Plotly

9.3 Sloupcový graf

Sloupcový graf je součástí kategorie žebříčků. Slouží k zobrazení vztahu mezi numerickou a jinou hodnotou. V typickém provedení reprezentuje výška obdélníku numerickou hodnotu. Rozdíl v náročnosti minimální implementace je zanedbatelný.

Vega

```
import altair as alt

alt.Chart(data).mark_bar().encode(
        x="x",
        y="y"
).properties(width=1000, height=400)
```



Obr. 8 Sloupcový graf a jeho implementace
Zdroj: vlastní zpracování pomocí Vega

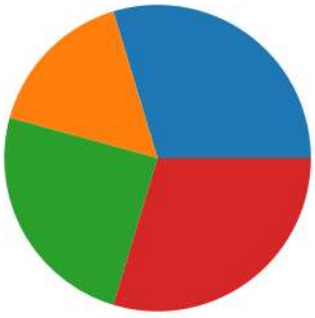
9.4 Výsečový graf

Výsečový graf je jeden z nejpoužívanějších grafů, slouží k reprezentaci části z celku. Kruhový „koláč“ je rozdělen na části(kategorie) jejichž velikost se liší podle hodnoty kterou mají reprezentovat.
Náročnost implementace v ekosystémech Matplotlib a Plotly je podobná.

Vega vyžaduje instance objektu Color, Theta, a neumí využít index datové sady. Je tedy potřeba index zkopírovat do samostatného sloupečku, aby se kategorie zobrazila správně. Zkopírování sloupečku a vytvoření instancí Color, Theta je jednoduché. Avšak při srovnání s ostatními ekosystémy je zde již rozdíl v náročnosti.

Matplot

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 4), dpi=100)
plt.pie(data["size"])
plt.show()
```



Obr. 9 Výsečový graf a implementace
Zdroj: vlastní zpracování pomocí Matplotlib

9.5 Choropleth

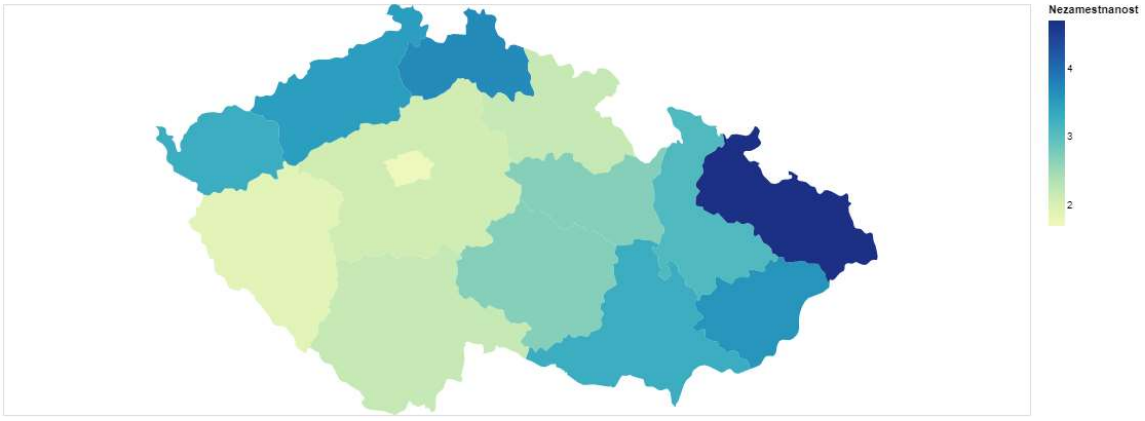
Choropleth spadá do kategorie map. Graf zbarvuje části mapy podle přiřazené hodnoty. Lze využít například k reprezentaci procentuální nezaměstnanosti v jednotlivých krajích. U tohoto grafu je již rozdíl v náročnost velmi znatelný.

Nejsnazší na implementaci se ukázala Vega. V náročnosti oproti jinému druhu grafu není téměř žádná změna. Společně s Matplotlib využívá Geopandas–GeoDataFrame. Matplotlib je v tomto případě implementován pomocí knihovny Geoplot, oproti ekosystému Vega vyžaduje více parametrů a využívá specifickou knihovnu.

Plotly je v tomto případě nejnáročnější na implementaci. Implementace vyžaduje velké množství parametrů a dalších úprav. Plotly neumí plně využít GeoDataFrame, je tedy potřeba tvary krajů exportovat jako geojson, který poté Plotly již umí využít.

Vega

```
import altair as alt

alt.Chart(data).mark_geoshape().encode(
    color="Nezamestnanost"
).project(type="mercator").properties(width=1000,height=400)
```



Obr. 10 Choropleth a implementace
Zdroj: vlastní zpracování pomocí Vega

9.6 Síť

Síťový graf zobrazuje spojení několika bodů pomocí hran. Rozdíl v náročnosti se ukázal být znatelný. Matplotlib a Vega jsou náročností velmi podobné.

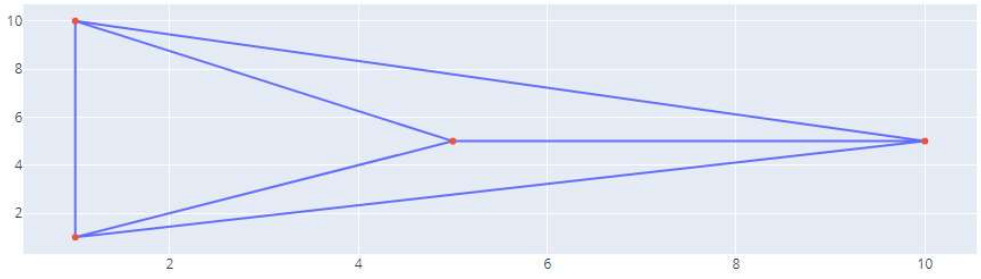
Matplotlib je v tomto případě využíván balíkem Networkx, který se stará o generaci, zpracování a vizualizaci grafů.

Vega je využívaná balíkem nx_altair, tento balík umí pracovat s datovými strukturami balíku Networkx.

Plotly nemá dle oficiální dokumentace podporu pro tento typ grafu, a je tak potřeba vygenerovat celý graf pomocí čar a bodů.

Plotly

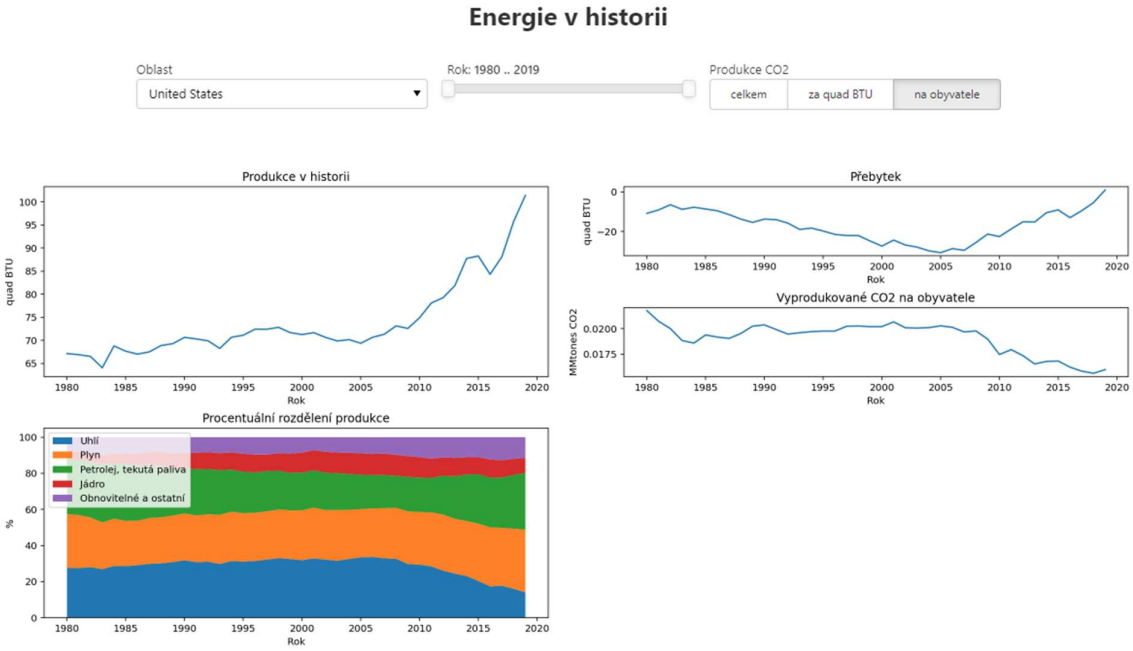
```
import plotly.graph_objects as go
import plotly.io as pio
pio.renderers.default="notebook"

nx.set_node_attributes(gr,{0:(5,5), 1:(1,1), 2:(10,5), 3:(1,10)},name="position")

edge_x = []
edge_y = []
point_x = []
point_y = []

for edge in gr.edges():
    x0, y0 = gr.nodes[edge[0]]["position"]
    x1, y1 = gr.nodes[edge[1]]["position"]
    edge_x.append(x0)
    edge_x.append(x1)
    edge_y.append(y0)
    edge_y.append(y1)

for node in gr.nodes():
    x, y = gr.nodes[node]["position"]
    point_x.append(x)
    point_y.append(y)

nodes = go.Scatter(x=point_x, y=point_y, mode="markers")
edges = go.Scatter(x=edge_x, y=edge_y, mode="lines")

fig = go.Figure(data=[edges, nodes],layout=go.Layout(width=1000, height=400, showlegend=False))
fig.show()
```



Obr. 11 Síťový graf a implementace
Zdroj: vlastní zpracování pomocí Plotly

9.7 Dashboard

Ekosystémy Vega a Matplotlib nemají na rozdíl od Plotly vlastní balíček pro tvorbu dashboardu, proto je nutné využít balíček Panel, který se stará o interaktivitu a prezentaci grafů. Z hlediska náročnosti byly mezi implementacemi značné rozdíly. Celkově nejsnazší na implementaci byl Matplotlib. Knihovna Plotly je náročnější na implementaci, ale úprava vzhledu a rozložení je příjemnější. Implementace pomocí balíčku Vega je na počet řádků krátká, avšak modifikace rozmístění a vzhledu jsou problematické a velmi omezené.



Obr. 12 Dashboard – energie v historii
Zdroj: vlastní zpracování pomocí Matplotlib

9.8 Design

Vzhled grafu a výběr barev může silně ovlivnit způsob, jakým je graf vnímán.

Nejsnazším způsobem, jak změnit vzhled vizualizace jsou styly/témata. Plotly a Matplotlib nabízejí několik předpřipravených témat. Vega bohužel žádnou obdobnou funkcionalitu nemá.

```
# Matplotlib
import matplotlib.pyplot as plt
plt.style.use("classic")

# Plotly globální
import plotly.io as pio
pio.templates.default = "plotly_white"

# Plotly agument
import plotly.express as px
px.scatter(data, template = "plotly_dark")
```

Kód 7 – Nastavení stylu pro Matplotlib a Plotly

10 Shrnutí výsledků

Jak již bylo avizováno dříve, cílem práce je poskytnou čtenáři pohled na všechny tři ekosystémy a jejich základní knihovny. Proto se v následujících řádcích nachází stručné zhodnocení každé knihovny. Avšak konečné rozhodnutí o nejlepší knihovně je na samotném čtenáři.

Všechny srovnávané ekosystémy jsou implementovány open-source knihovnami. Tyto knihovny jsou nadále rozšiřovány a využívány početnou základnou uživatel.

10.1 Matplotlib

Nejstarší z ekosystémů a zároveň nejdéle ve vývoji. V současné době udržovaný komunitou jako open-source knihovna. Využitím knihoven Seaborn a Bokeh lze snadno rozšířit vizualizační možnosti.

10.2 Plotly

V některých případech je knihovna náročnější na implementaci, než ostatní ekosystémy, ale po vizuální stránce poskytuje subjektivně nejlepší výsledky. Až na některé specifické případy, lze díky knihovnám Plotly Express, Plotly Graph Objects a Dash snadno vytvářet interaktivní dashboard vizualizace. Mezi konkrétní problémové případy patří vizualizace mapy nebo síťového grafu.

10.3 Vega

Ekosystém Vega je svým způsobem outsider, a jeho popularita je značně nižší. Své nedostatky kompenzuje náročností implementace, která je velice jednoduchá.

Pokročilé a interaktivní vizualizace jsou bohužel dosti problémové.

11 Závěry a doporučení

V této práci byly představeny, porovnány a implementovány ukázky tří ekosystémů pro vizualizaci dat v jazyce Python.

V budoucnu by bylo možné práci rozšířit o další ekosystémy a implementace grafů dle jiného řazení.

12 Zdroje a literatura

[1] VANDERPLAS, Jake. Python Data Science Handbook [online]. [cit. 10.12.2022]. Dostupné z: https://jakevdp.github.io/PythonDataScienceHandbook/

[2] YAU, Nathan. Visualize This: The Flowing Data Guide to Design, Visualization, and Statistics. vyd. Wiley Publishing inc., 2011. ISBN 978-1-118-14024-6

[3] Python graph gallery. [online]. [cit. 18.12.2022]. Dostupné z: https://www.python-graph-gallery.com/

[4] Matplotlib documentation. [online]. [cit. 5.11.2022]. Dostupné z: https://matplotlib.org/stable/index.html

[5] API reference for plotly. [online]. [cit. 8.11.2022]. Dostupné z: https://plotly.com/python-api-reference/

[6] Vega-Altair documentation. [online]. [cit. 12.11.2022]. Dostupné z: https://altair-viz.github.io/

[7] Natural Earth - Free vector and raster map data. [online]. [cit. 12.11.2022]. Dostupné z: https://www.naturalearthdata.com/

[8] API Reference for Panel. [online]. [cit. 15.12.2022]. Dostupné z: https://panel.holoviz.org/api/index.html

[9] API reference for plotly-dash. [online]. [cit. 15.12.2022]. Dostupné z: https://dash.plotly.com/reference

[10] VANOUS, Benjamin. Countries CO2 Emission and more... [online]. [cit. 15.12.2022] Dostupné z: https://www.kaggle.com/datasets/lobosi/c02-emission-by-countrys-grouth-and-population

[11] Veřejná databáze ČSÚ. Základní charakteristiky ekonomického postavení obyvatelstva ve věku 15 a více let. [online]. [cit. 11.10.2022]. Dostupné z: https://vdb.czso.cz/vdbvo2/faces/index.jsf?page=vystup-objekt&z=T&f=TABULKA&pvo=ZAM01-A&skupId=426&katalog=30853&&c=v3~8__RP2017&str=v178&kodjaz=203

[12] Tiobe index. [online]. [cit. 20.12.2022]. Dostupné z: www.tiobe.com

[13] PYPL index. [online]. [cit. 20.12.2022]. Dostupné z: https://pypl.github.io/PYPL.html

[14] PyPI. [online]. [cit. 20.12.2022]. Dostupné z: https://pypi.org/

[15] Github – Altair. [online]. [cit. 27.2.2023]. Dostupné z: https://github.com/altair-viz/altair

[16] Github – Matplotlib. [online]. [cit. 27.2.2023]. Dostupné z: https://github.com/matplotlib/matplotlib

[17] Github – Vega. [online]. [cit. 27.2.2023]. Dostupné z: https://github.com/vega/vega

[18] Github – Plotly.py. [online]. [cit. 27.2.2023]. Dostupné z: https://github.com/plotly/plotly.py

13 Přílohy

1) Struktura projektu
2) Notebook – Graf hustoty
3) Notebook – Bodový graf
4) Notebook – Sloupcový graf
5) Notebook – Výsečový graf
6) Notebook – Choropleth
7) Notebook – Síťový graf
8) Notebook – Dashboard
9) Notebook – Design, téma

Struktura projektu

- doc – Práce ve formátu docx a pdf
- src – Zdrojové kódy ukázek – Jupyter notebooky
- data – Veškerá data využitá v ukázkách

Všechny soubory jsou dostupné na adrese:
https://github.com/GlummixX/Bachelor_thesis

Graf hustoty

Matěj Kolář 2022

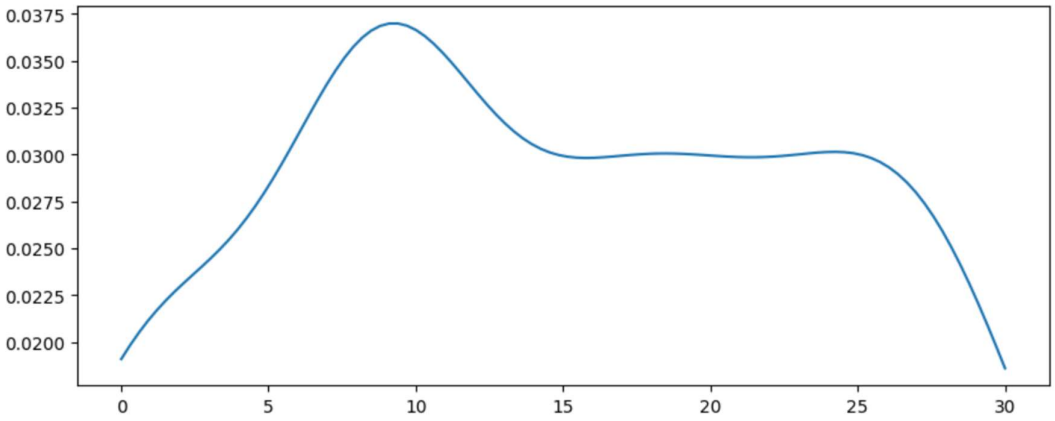
UHK - FIM

Sdílený kód

```
import numpy as np
import pandas as pd
import random
from scipy.stats import gaussian_kde

random.seed(2022) #seed

### Generace dat ###
data = []
for x in range(0,20):
    data = data + [random.randint(0,30)]*random.randint(1,10)

x = np.linspace(0, 30, 100)
data = pd.DataFrame({"x":x,"y":gaussian_kde(data)(x)})
```

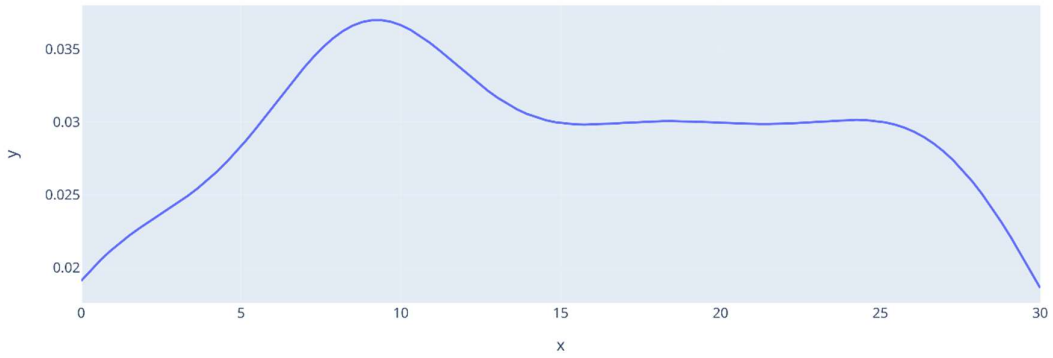
Matplotlib

```
from matplotlib import pyplot as plt

plt.figure(figsize=(10, 4), dpi=100) #10 inch * 100 dpi = 1000px
plt.plot(data["x"],data["y"])
plt.show()
```



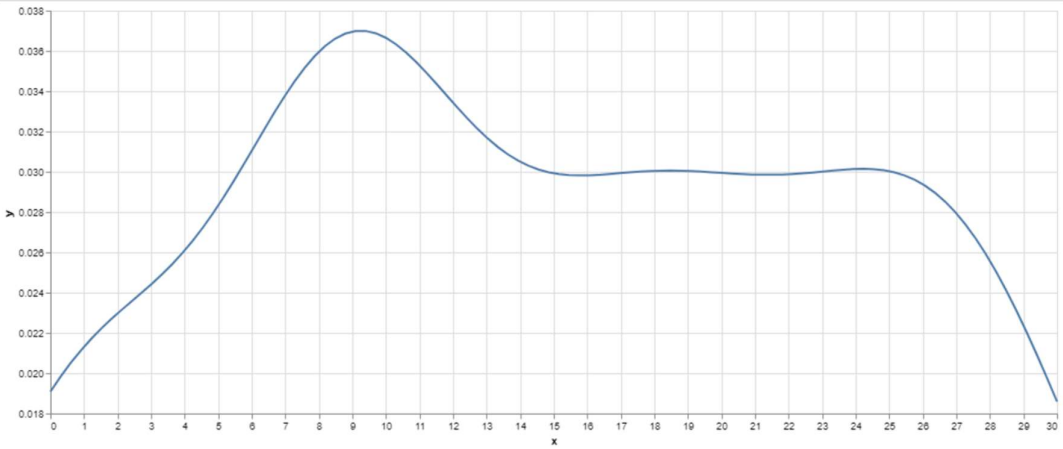
Plotly

```
import plotly.express as px
#Následující dva řádky slouží pouze pro usnadnění exportu, nejsou kritické pro běh
import plotly.io as pio
pio.renderers.default="notebook"

fig = px.line(data, x="x", y="y", width=1000, height=400)
```



Vega

```
import altair as alt

alt.Chart(data).mark_line().encode(
    alt.Y('y',scale=alt.Scale(zero=False)),
    x="x"
).properties(width=1000, height=400)
```



Bodový graf

Matěj Kolář 2022

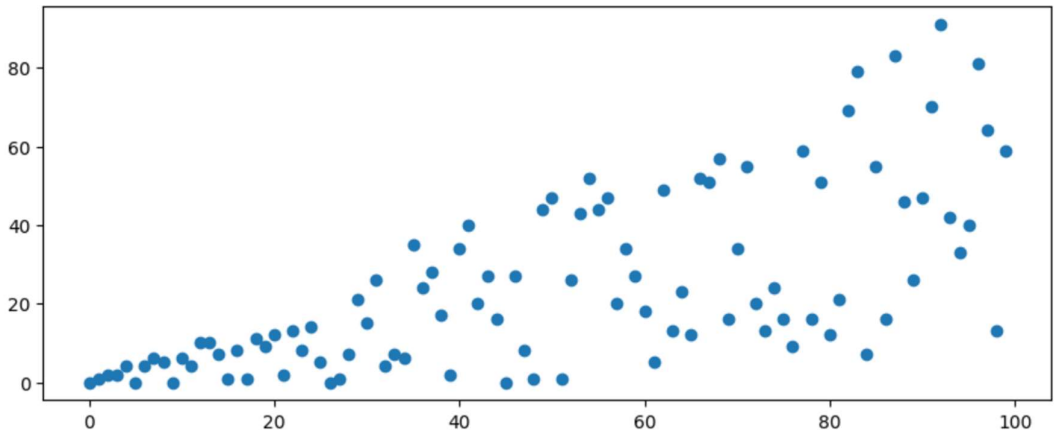
UHK - FIM

Sdílený kód

```
import random
import pandas as pd

random.seed(2022) #seed

### Generace dat ###
y = []
for x in range(0,100):
    y.append(random.randint(0,x))

data = pd.DataFrame({"x": range(0,100), "y": y})
```

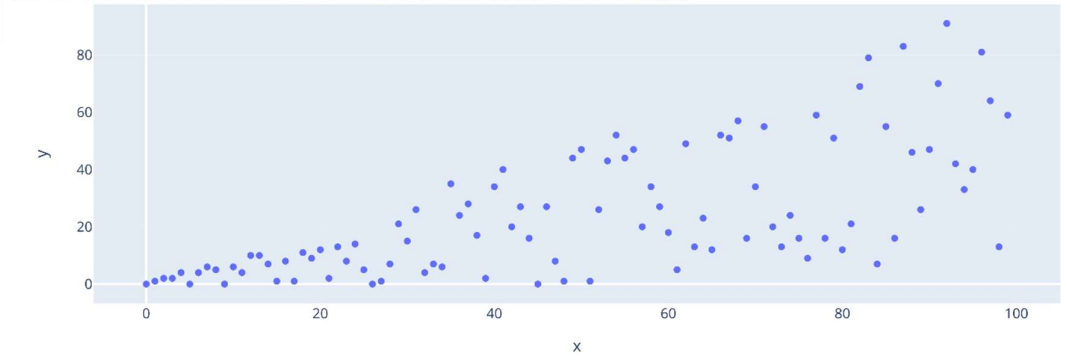
Matplotlib

```
from matplotlib import pyplot as plt

plt.figure(figsize=(10, 4), dpi=100)
plt.plot(data["x"],data["y"], linestyle='none', marker='o')
plt.show()
```



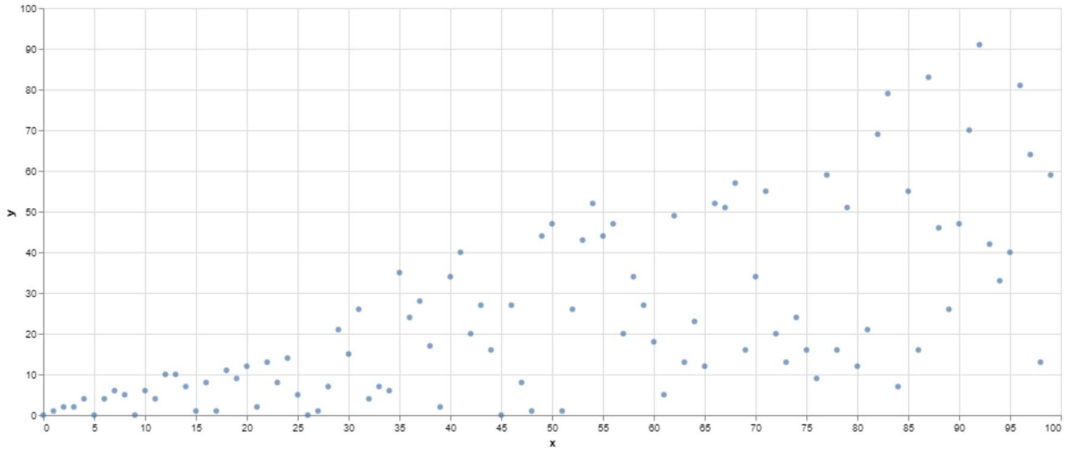
Plotly

```
import plotly.express as px
import plotly.io as pio
pio.renderers.default="notebook"

fig = px.scatter(data,x="x",y="y", width=1000, height=400)
```



Vega

```
import altair as alt

alt.Chart(data).mark_circle().encode(
    x="x",
    y="y"
).properties(width=1000, height=400)
```



Sloupcový graf

Matěj Kolář 2022

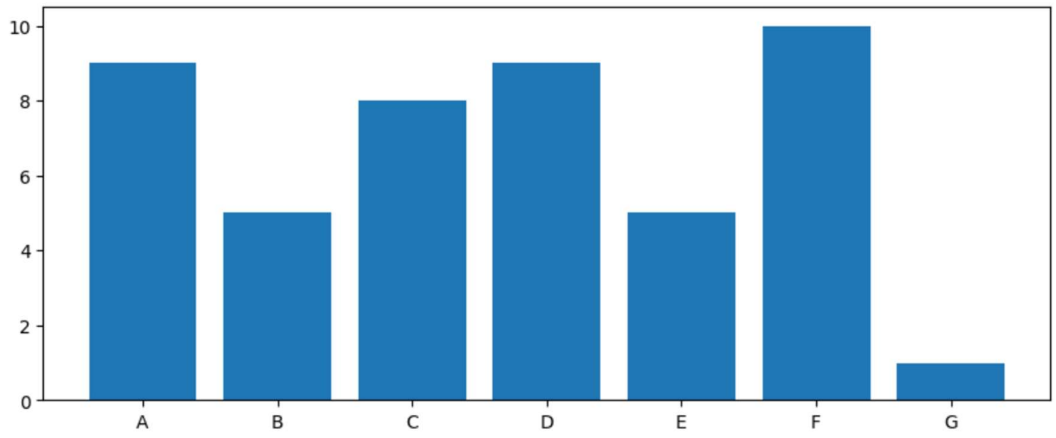
UHK - FIM

Sdílený kód

```
import pandas as pd
import random

random.seed(2022) #seed

### Generace dat ###
y = []
x = ["A", "B", "C", "D", "E", "F", "G"]
for i in x:
    y.append(random.randint(1,10))

data = pd.DataFrame({"x": x, "y": y})
```

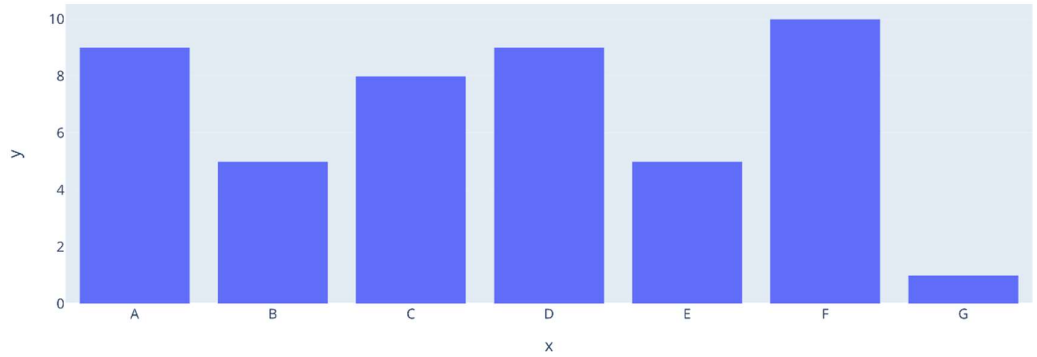
Matplotlib

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 4), dpi=100)
plt.bar(data["x"], data["y"])
plt.show()
```



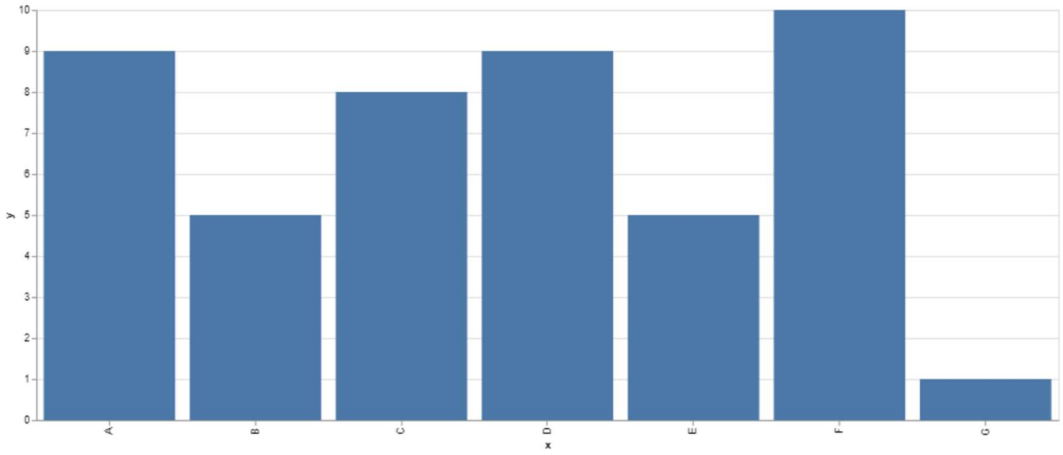
Plotly

```
import plotly.express as px
import plotly.io as pio
pio.renderers.default="notebook"

fig = px.bar(data, x="x", y="y",width=1000, height=400)
fig.show()
```



Vega

```
import altair as alt

alt.Chart(data).mark_bar().encode(
        x="x",
        y="y"
).properties(width=1000, height=400)
```



Výsečový graf

Matěj Kolář 2022

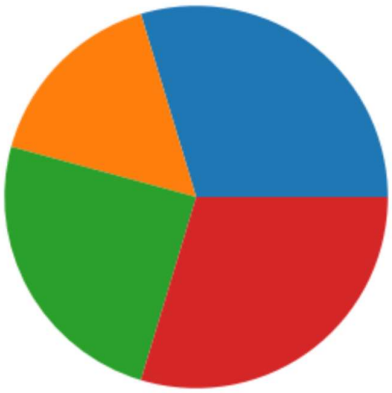
UHK - FIM

Sdílený kód

```
import pandas as pd
import random

random.seed(2022) #seed

### Generace dat ###
size = []
while sum(size) < 100:
    size.append(random.randint(1,50))

data = pd.DataFrame({"size": size})
```

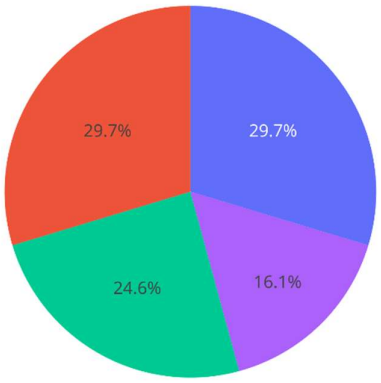
Matplotlib

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 4), dpi=100)
plt.pie(data["size"])
plt.show()
```



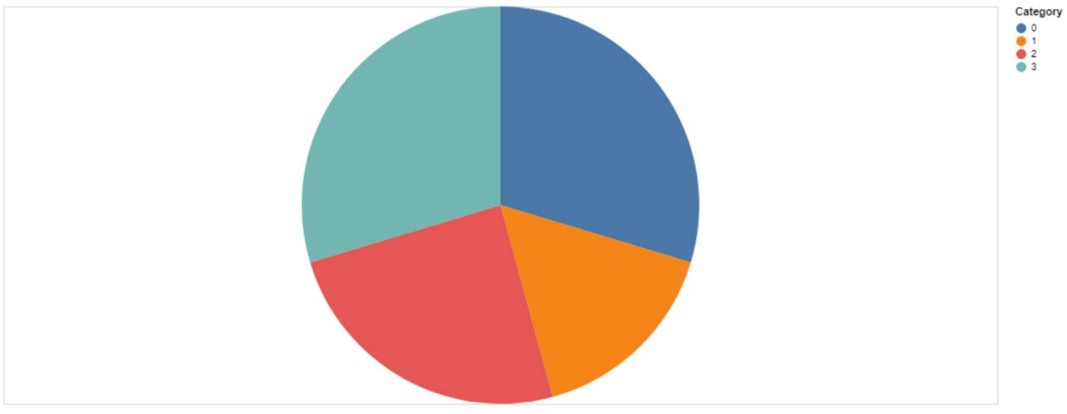
Plotly

```
import plotly.express as px
import plotly.io as pio
pio.renderers.default="notebook"

fig = px.pie(data, values="size",width=1000, height=400)
fig.show()
```



Vega

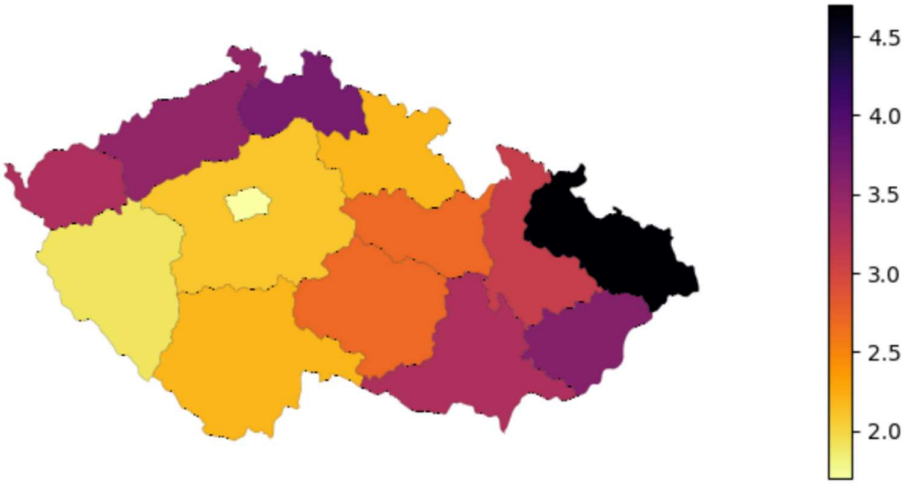
```
import altair as alt

data["Category"] = data.index #Nutno dodat kategorii dat zvlášť

alt.Chart(data).mark_arc().encode(
    theta=alt.Theta(field="size", type="quantitative"),
    color=alt.Color(field="Category", type="nominal"),
).properties(width=1000, height=400)
```



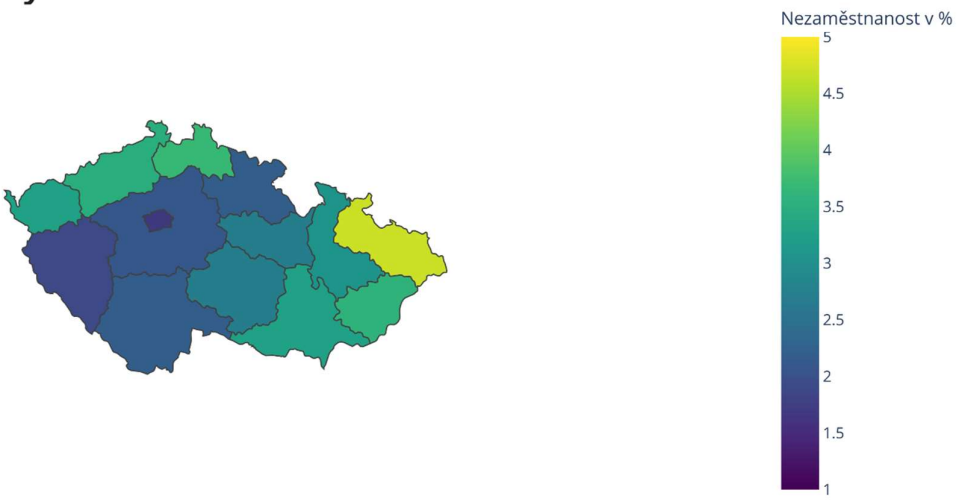
Choropleth

Celá ukázka i s kódem je příliš obsáhlá, lze jí nalézt v digitální podobě.

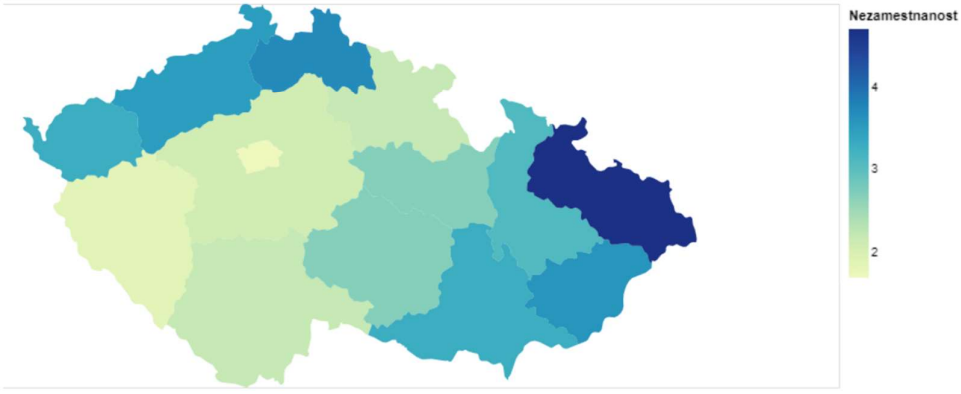
Matplotlib



Plotly



Vega



Síťový graf

Matěj Kolář 2022

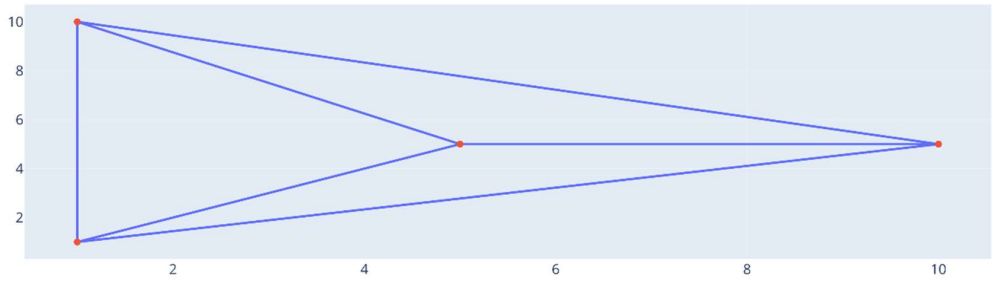
UHK - FIM

Sdílený kód

```
import networkx as nx
### Generace K4 grafu ###
gr = nx.complete_graph(4)
```

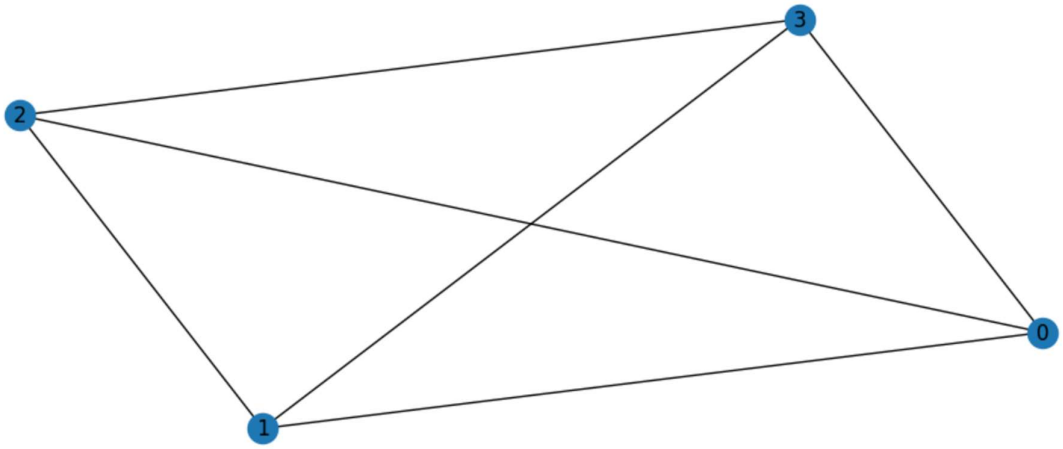
Plotly

```
import plotly.graph_objects as go
import plotly.io as pio
pio.renderers.default="notebook"

nx.set_node_attributes(gr,{0:(5,5), 1:(1,1), 2:(10,5), 3:(1,10)},name="position")

edge_x = []
edge_y = []
point_x = []
point_y = []

for edge in gr.edges():
    x0, y0 = gr.nodes[edge[0]]["position"]
    x1, y1 = gr.nodes[edge[1]]["position"]
    edge_x.append(x0)
    edge_x.append(x1)
    edge_y.append(y0)
    edge_y.append(y1)

for node in gr.nodes():
    x, y = gr.nodes[node]["position"]
    point_x.append(x)
    point_y.append(y)

nodes = go.Scatter(x=point_x, y=point_y, mode="markers")
edges = go.Scatter(x=edge_x, y=edge_y, mode="lines")

fig = go.Figure(data=[edges, nodes],
                layout=go.Layout(width=1000, height=400, showlegend=False))
fig.show()
```



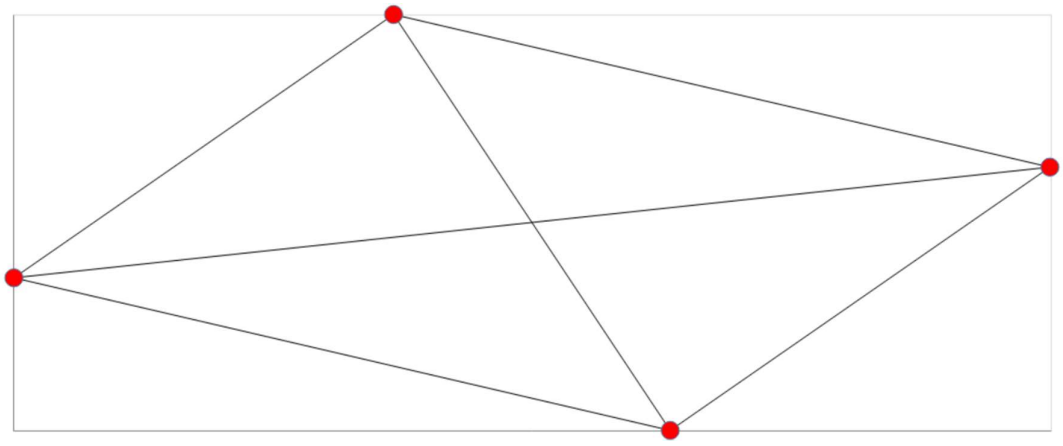
Matplotlib

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 4), dpi=100)
nx.draw(gr, with_labels=True)
plt.show()
```



Vega

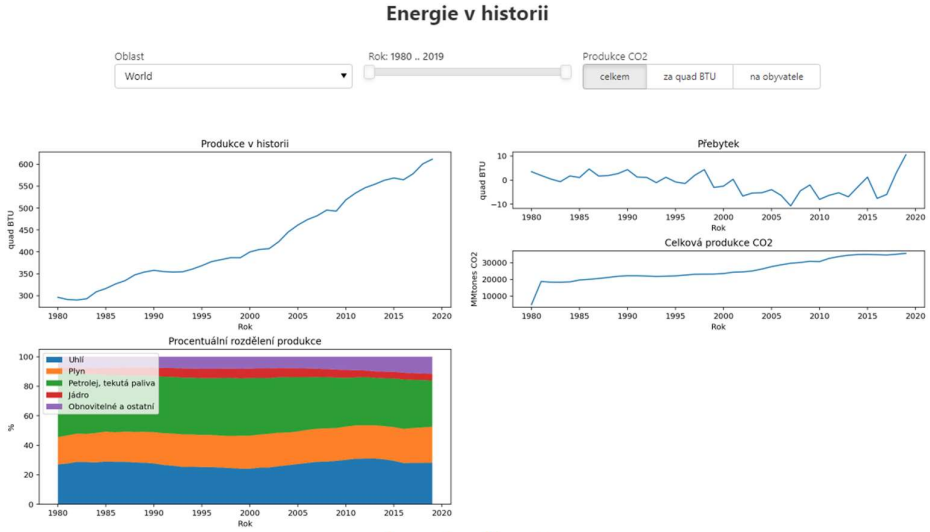
```
import nx_altair as nxa

nxa.draw_networkx(G=gr).properties(width=1000, height=400)
```

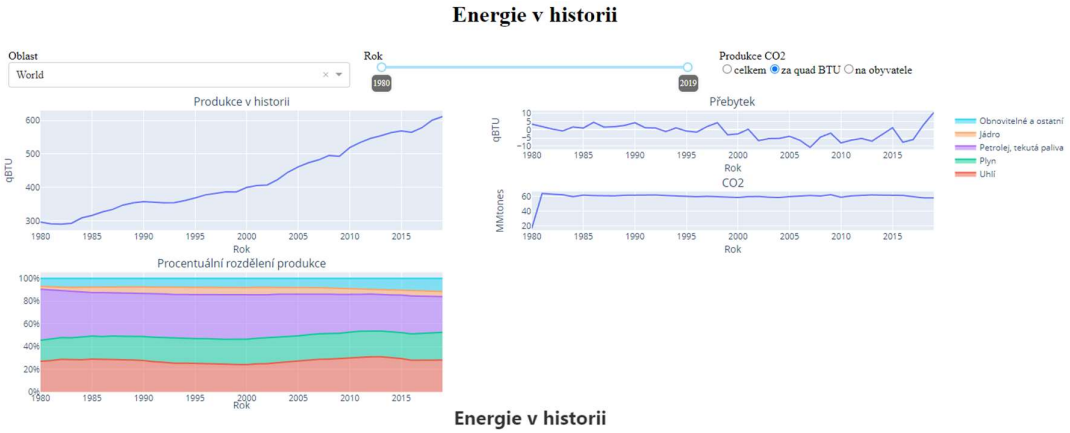


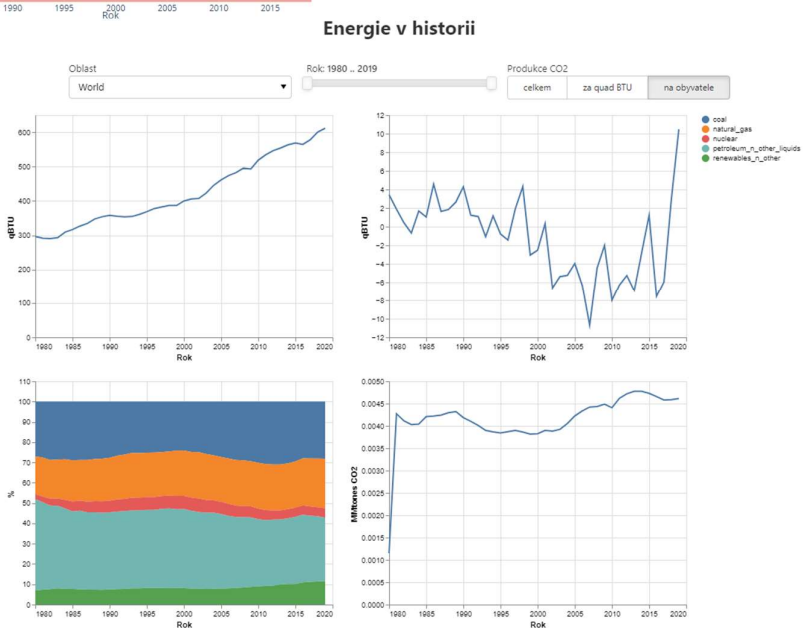
Pro nedostatek místa na stránce bylo pořadí kapitol této ukázky upraveno.

Dashboard

Celá ukázka i s kódem je příliš obsáhlá, lze jí nalézt v digitální podobě.







Design - Témata

Matěj Kolář 2023

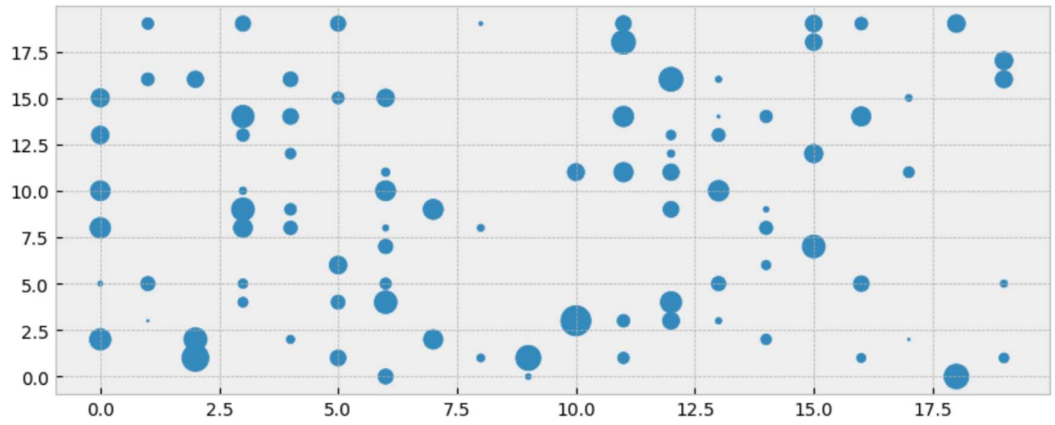
UHK - FIM

Sdílený kód

```
import numpy as np
import numpy.random as np_rand
import pandas as pd

np_rand.seed(2022) #seed

### Generace dat ###
data = {"x":[], "y":[], "s":[]}
for i in range(0,100):
    data["x"].append(np_rand.randint(0,20))
    data["y"].append(np_rand.randint(0,20))
    data["s"].append(int(abs(np_rand.normal()*100)))
data = pd.DataFrame(data).drop_duplicates(["x","y"])
```

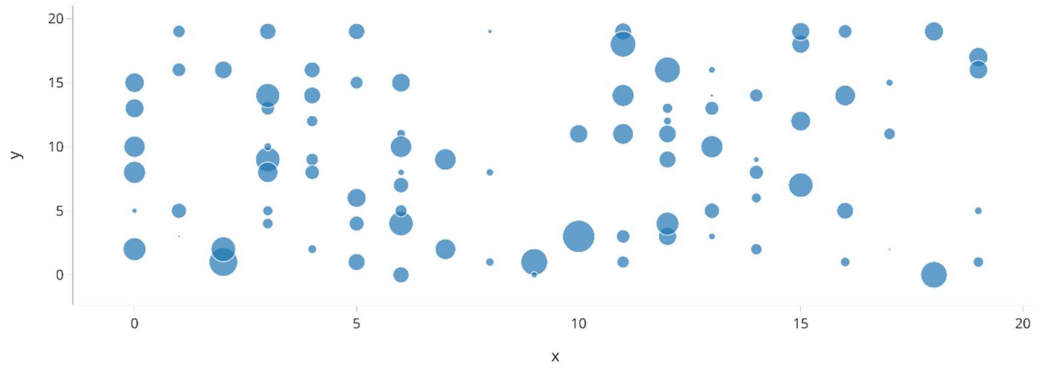
Matplotlib

```
import matplotlib.pyplot as plt

plt.style.use("bmh")
plt.figure(figsize=(10, 4), dpi=100)
plt.scatter(x=data["x"], y=data["y"], s=data["s"])
plt.show()
```



Plotly

```
import plotly.express as px
import plotly.io as pio
pio.renderers.default="notebook"

fig = px.scatter(data, x="x", y="y", size="s", width=1000, height=400, template="simple_white")
fig.show()
```



Vega

Tato funkcionalita bohužel není podporována

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání bakalářské práce

Autor: **Matěj Kolář**

Studium: I2000376

Studijní program: B1802 Aplikovaná informatika

Studijní obor: Aplikovaná informatika

Název bakalářské práce: **Vizualizace dat v Python**

Název bakalářské práce AJ: Data visualization in Python

Cíl, metody, literatura, předpoklady:

1/ popsat základní typologii grafů

2/ popsat základní vizualizační ekosystémy (matplotlib, plotly, vega)

3/ implementovat vybrané grafy z každé kategorie ve všech ekosystémech

VANDERPLAS, Jake. Python Data Science Handbook: https://jakevdp.github.io/PythonDataScienceHandbook/

YAU, Nathan: Visualize This: The FlowingData Guide to Design, Visualization, and Statistics, ISBN-13: 978-0470944882

Python graph gallery https://www.python-graph-gallery.com/

Zadávající pracoviště: Katedra informatiky a kvantitativních metod, Fakulta informatiky a managementu

Vedoucí práce: Mgr. Jiří Haviger, Ph.D.

Datum zadání závěrečné práce: 26.5.2022