**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 Červenec 2023

Prohlášení:

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

V Hradci Králové dne 19.7.2023 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 zaměřuje na popis vybraných grafů a jejich implementaci ve vizualizačních ekosystémech Matplotlib, Plotly a Vega. Cílem práce je poskytnout čtenářům ucelený přehled o těchto knihovnách, jejich funkcionalitách a možnostech pro tvorbu grafů v jazyce Python.

V práci je také věnována pozornost datovým strukturám pro práci s daty v jazyce Python. Jsou prezentovány jak nativní datové struktury, tak i externí struktury, které poskytují rozšířenou funkcionalitu pro efektivnější práci s daty.

Celkově práce přináší informace o vybraných grafech a jejich implementaci ve zmíněných ekosystémech. V závěru práce se nachází porovnání ekosystémů a doporučení kdy který systém využít.

# Annotation

## Title: Data visualization in Python

This bachelor's thesis focuses on the description of selected graphs and their implementation in the visualization ecosystems Matplotlib, Plotly and Vega. The aim of the work is to provide readers with a comprehensive overview of these libraries, their functionalities and options for creating graphs in the Python language.

The work also pays attention to data structures for working with data in the Python language. Both native data structures and external structures that provide extended functionality for more efficient work with data are presented.

Overall, the work provides information about selected graphs and their implementation in the mentioned ecosystems. At the end of the work there is a comparison of ecosystems and recommendations when to use which system.

# Obsah

# Seznam obrázků



# Seznam ukázek kódu

# 1 Úvod

V současném době je vizualizace klíčová pro pochopení a interpretaci informací. Jazyk Python se výrazně prosadil a stal se jedním z nejoblíbenějších programovacích jazyků napříč odvětvími. Podle indexů Tiobe [12] a PYPL [13] patří Python k nejpopulárnějším jazykům, což je zřejmé i z obrovského množství dostupných knihoven. Python Package Index [14] momentálně eviduje více než 418 000 balíčků, číslo, které stále roste. Tato práce se zaměřuje na tři ekosystémy pro vizualizaci dat v jazyce Python, a to na Matplotlib, Plotly a Vega.

Cílem této práce je porovnat tyto ekosystémy a jejich schopnost vytvářet různé druhy grafů. Každý z těchto ekosystémů přináší své vlastní nástroje pro vizualizaci dat, a proto je důležité porovnat jejich silné stránky, omezení a schopnost přizpůsobit se potřebám uživatele.

Pro dosažení tohoto cíle byla provedena implementace jednoho grafu z každé kategorie dle Python graf gallery [3]. Během implementace byla testována funkcionalita jednotlivých knihoven a jejich schopnost vytvářet esteticky příjemné a srozumitelné grafy. Následně byly tyto implementace porovnány z hlediska jejich náročnosti.

# 2 Data a jejich specifikace

Daty jsou v informatice rozuměny záznamy, fakta, údaje nebo jakékoliv jiné informace, které jsou zachyceny ve formě, která je vhodná pro zpracování, uchovávání a přenos. Vzhledem k povaze počítačů jsou všechna data převáděna do digitální podoby, což má za následek jejich diskrétnost, tedy samotné datové body jsou nespojité jak v čase, tak i v hodnotě. Pomocí vysoké vzorkovací frekvence, dostatečného rozlišení a interpolace lze dosáhnout spojitosti.

## *2.1 Struktura*

Dle struktury lze data rozdělit na data strukturovaná a nestrukturovaná. Lidé i stroje produkují různé formy strukturovaných a nestrukturovaných dat. Vhodná struktura dat umožňuje jejich snadnější interpretaci, efektivní filtrování a selekci dat pro vizualizaci.

### Data strukturovaná

Strukturovaná data mají pevně definovanou strukturu, kde každá položka má určený formát a význam. Jsou organizována v databázích nebo data warehouse a nacházejí široké využití v oblasti CRM a ERP systémů a dalších oblastech. Příklady strukturovaných dat zahrnují telefonní čísla, časové údaje, uživatelská jména, názvy produktů a mnoho dalších. Díky své organizované povaze jsou data lehce zpracovatelná a umožňují efektivní práci s informacemi. Jejich zásadní výhoda spočívá v snadné prohledávatelnosti.

### Data nestrukturovaná

Nestrukturovaná data se vyznačují absencí předem definované podoby, což ztěžuje jejich prohledávatelnost. Tyto data jsou uložena přímo v aplikacích, NoSQL databázích, data lake nebo data warehouse. V praxi se nestrukturovaná data využívají v různých aplikacích, jako jsou prezentace, nástroje pro prohlížení a úpravu médií, datové pipelines a další. Příklady nestrukturovaných dat zahrnují souvislý nijak nestrukturovaný text, audio záznamy, fotografie, video záznamy nebo nezpracovaná surová data. Díky své specifické povaze vyžadují nestrukturovaná data speciální přístup při jejich zpracování a analyzování.

## *2.2 Interpretace hodnot*

Data jsou v digitální podobě reprezentována byty, proto je správná interpretace datových typů zásadní pro dosažení spolehlivých a relevantních výsledků při zpracování dat. Jedním příkladem chybné interpretace dat může být nesprávné rozpoznání časového razítka jako číselné hodnoty. Například, pokud

bude půlnoc 7. června 2001 interpretována jako číslo, může dojít k převodu na hodnotu 991872000, což není správné. Chyba interpretace může nastat i u samotného datového typu. Například číslo 255 v datovém typu uint8 by může být špatně interpretováno jako -1 v datovém typu int8. Je velmi důležité používat vhodné funkce pro interpretaci dat, a to jak z hlediska jejich formátu, tak i datových typů.

## 2.3 Veličiny

Pro správnou vizualizaci dat je důležité znát vlastnosti veličin, které se snažíme zobrazit. Tato kapitola se zaměřuje na tyto vlastnosti veličin a jejich případné dělení.

### Dimenze veličiny

Při vizualizaci dat je důležité zohlednit dimenzi veličiny. Vlastnost nebo veličina může být vyjádřena jednou nebo více dimenzemi, které určují její charakteristiku. Mezi typické dimenze patří časová, která zobrazuje vývoj hodnot v čase. Další dimenze mohou zahrnovat kategorické hodnoty, které dělí data do různých kategorií, nebo numerické hodnoty, které reprezentují kvantitativní informace. Správná vizualizace zohledňuje tyto dimenze a umožňuje lepší porozumění a analýzu datových sad.

### Veličiny diskrétní a spojité

Diskrétní veličiny nabývají pouze izolovaných a oddělených hodnot. Tyto hodnoty jsou spočetné a mají jasně vymezený rozsah. Spojité veličiny, na rozdíl od diskrétních, nabývají libovolné hodnoty v určitém rozsahu. Tyto hodnoty mohou být spojité, nekonečné či zahrnovat racionální i iracionální čísla.

### Veličiny kvantitativní a kvalitativní

Kvantitativní veličiny jsou měřitelné a vyjádřitelné pomocí číselných hodnot, zaměřují se na kvantitativní aspekty a poskytují konkrétní informace o jevech nebo vlastnostech. Nacházejí využití například v matematice, fyzice nebo ekonomii. Oproti tomu kvalitativní veličiny se zaměřují na popisné vlastnosti, charakteristiky a kategorie jevů, které nelze vyjádřit numericky. Tyto veličiny se často vyskytují v oblastech jako sociologie, psychologie nebo lingvistika.

### Veličiny nominální a ordinální

Nominální veličiny nemají žádné pořadí nebo hierarchii mezi kategoriemi. Zatím co ordinální veličiny zachovávají kategorický charakter, ale zároveň vyjadřují určité pořadí nebo hierarchii mezi kategoriemi.

### Veličiny intervalové a poměrové

Intervalové veličiny zachovávají pořadí kategorií a umožňují měření rozdílů mezi hodnotami. Navíc mají konstantní rozdělení mezi jednotlivými hodnotami.

Poměrové veličiny jsou nejkomplexnější formou kvantitativních veličin. Kromě všech vlastností intervalových veličin umožňují také stanovit poměr mezi hodnotami.

### Veličiny závislé a nezávislé

Nezávislé veličiny se mohou měnit nezávisle na ostatních veličinách. Zatím co závislé veličiny jsou nějakým způsobem propojené nebo jinak závislé na jiných veličinách v pozorovaném systému. Příkladem závislé veličiny je teplota, která závisí na lokaci a čase měření. Zatím co čas je veličina nezávislá.

## *2.4 Formáty souborů*

Pro pohodlný přenos a skladování dat jsou využívány různé formáty souborů.

### CSV

Comma separated values je primitivní způsob ukládání tabulkových dat do souboru. Data sloupců jsou na řádku spojeny do jednoho řetězce, takzvaný separátor je poté využit k oddělení jednotlivých sloupců. Separátor může být jakýkoliv symbol, nejčastěji se používá čárka, středník, dvojtečka nebo pipe („|“). Taková data mohou být poté bez obtíží načtena pomocí vhodné knihovny nebo SW který formát podporuje. Formát CSV může implementovat kdokoliv.

### XLS (XLSX)

XLS a jeho novější verze XLSX jsou formáty používané tabulkovým editorem Excel od společnosti Microsoft. Existují knihovny a open-source programy, které umožňují práci s těmito formáty. Nicméně je důležité zmínit, že XLSX je komplexní, a hlavně uzavřený standard, což znamená, že jeho specifikace nejsou veřejně dostupné a jeho využití může být náročnější a kompatibilita nižší.

### Parquet

Parquet je pokročilý open-source formát určený pro efektivní skladování a správu velkých datových sad. Byl vyvinut společností Apache Software Foundation. Vyznačuje se nižším využitím paměťového prostoru ve srovnání s formátem CSV, což umožňuje úsporu místa při ukládání dat. Navíc, rychlost IO operací nad daty v tomto formátu je výrazně vyšší, což umožňuje rychle a efektivněji daná data zpracovat. Díky těmto vlastnostem je Parquet často preferovaným formátem pro Big Data aplikace. Pro efektivní implementaci je potřeba využití specializovaných aplikací a knihoven.

### HDF5

HDF je široce využívaný formát pro vědecká data, vynikající svou schopností efektivně ukládat komplexní datové struktury. Podpora komprese a paralelních IO operací přispívá k rychlému a efektivnímu zpracování dat. Díky nativní podpoře metadat a verzování umožňuje snazší zálohování a organizaci datových sad. Pro využití tohoto formátu je však zapotřebí specializovaný software, je proto vhodný pouze pro rozsáhlé datové sady.

### JSON

JSON (JavaScript Object Notation) je populární formát pro ukládání a výměnu dat v různých aplikacích. Jeho jednoduchá struktura klíč-hodnota umožňuje snadnou reprezentaci dat a čitelnost. JSON je velmi flexibilní a snadno použitelný pro ukládání a přenos dat různých typů. Je vhodnější volbou pro menší datové sady, kde se vyžaduje snadná implementace mezi různými technologiemi a systémy.

### XML

Formát XML je více strukturovaný a obsáhlejší ve srovnání s formátem JSON, což znamená, že zpracování XML dat může být náročnější. Přesto XML zůstává preferovaným formátem pro aplikace, které potřebují zpracovávat a ukládat složitá hierarchická data s důrazem na detailní strukturu a sémantiku.

# 3 Vizualizace

Vizualizace dat je způsob interpretace datových sad pomocí grafických prvků a symbolů, za účelem prezentace dat, zvýšení čitelnosti dat, či odhalení jinak nezjevných výsledků. Jedná se tedy o způsob, jakým lze snadno prezentovat data a výsledky, který není určený pouze pro odbornou komunitu. Data lze vizualizovat bez nutnosti psát vlastní programy a scripty, avšak využití programovacích jazyků poskytuje uživateli větší kontrolu nad výsledným grafem. Touto prací používaný Python není jediný jazyk vhodný k vizualizaci dat. Mezi další často používané jazyky patří R, Scala nebo Matlab.

## *3.1 Druhy vizualizací*

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

### Statické vizualizace

Statické vizualizace nabízejí řadu výhod, které zahrnují jednoduchost, snadnou implementaci a všestrannost. Mohou být uloženy jako obrázky, což eliminuje potřebu nepřetržitého renderování pomocí back-endu. Nicméně, jejich nevýhodou je absence interaktivních funkcí, které usnadňují čtení a interpretaci hodnot z grafu.

### Animované vizualizace

Animované vizualizace poskytují zajímavý a atraktivní způsob prezentace dat. Mohou být využity k oživení prezentace nebo ke zobrazení velkého množství dat na omezeném prostoru. Zejména jsou vhodné pro vizualizaci časových řad, kde je možné sledovat jejich vývoj v čase. Nicméně, animované vizualizace vyžadují back-end, který je bude aktivně renderovat. Nebo musí být uloženy ve formátech, které umožňují přehrávání animací, jako jsou gif, mp4 nebo mkv. Díky animovaným grafům lze efektivněji komunikovat dynamické aspekty dat a zvýšit zapamatovatelnost vizualizace.

### Interaktivní vizualizace

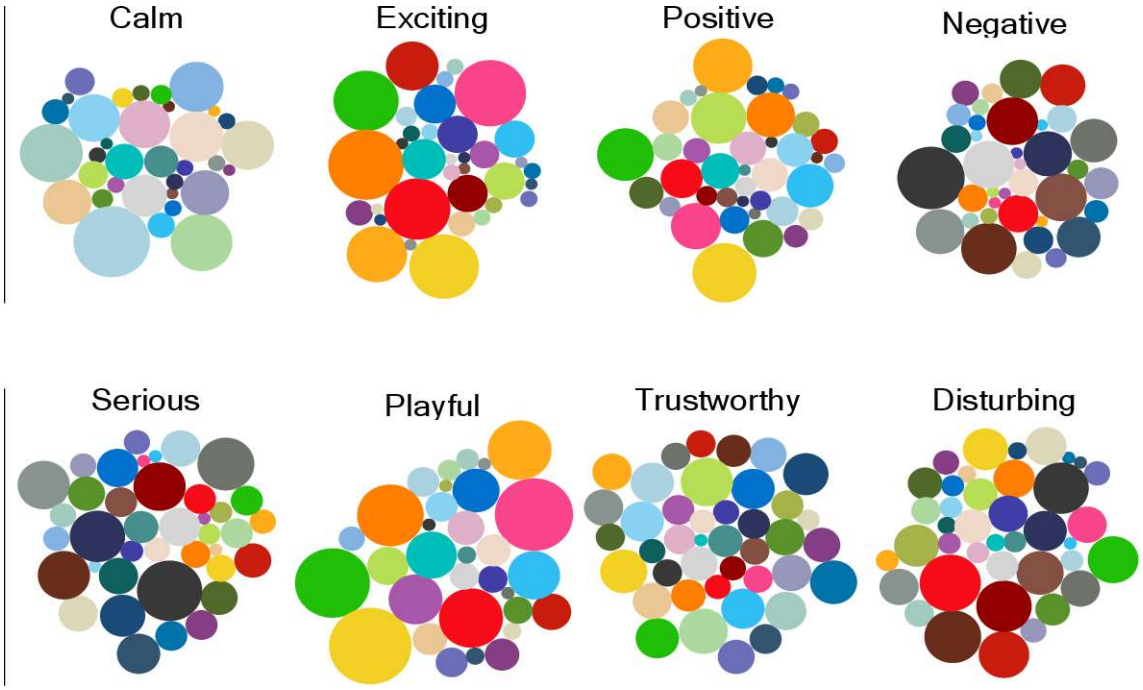
Interaktivní grafy představují ideální volbu pro tvorbu dashboardů a interaktivních vizualizací. Jejich hlavní výhodou je schopnost výběru dat, rozsahů a možnost pravidelné aktualizace. Interaktivní vizualizace jsou výrazně náročnější na implementaci než jejich statické alternativy. Stejně jako animované vizualizace, vyžadují back-end, který je bude neustále renderovat a navíc musí obsluhovat vstupy od uživatele.

## 3.2 Design

Design je klíčovým prvkem vizualizace dat, který nám umožňuje efektivně komunikovat informace. Tato kapitola se zaměřuje na různé prvky, které ovlivňují vzhled a čitelnost grafu.

### Barvy

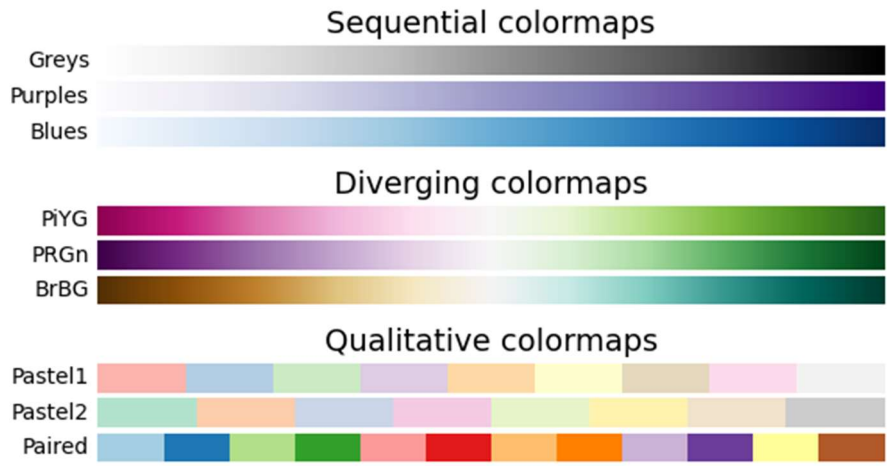
Důležitou součástí vizualizace je barevné schéma. Barevné schéma může ovlivnit čitelnost a přehlednost grafu nebo pocity které graf vyvolává.




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

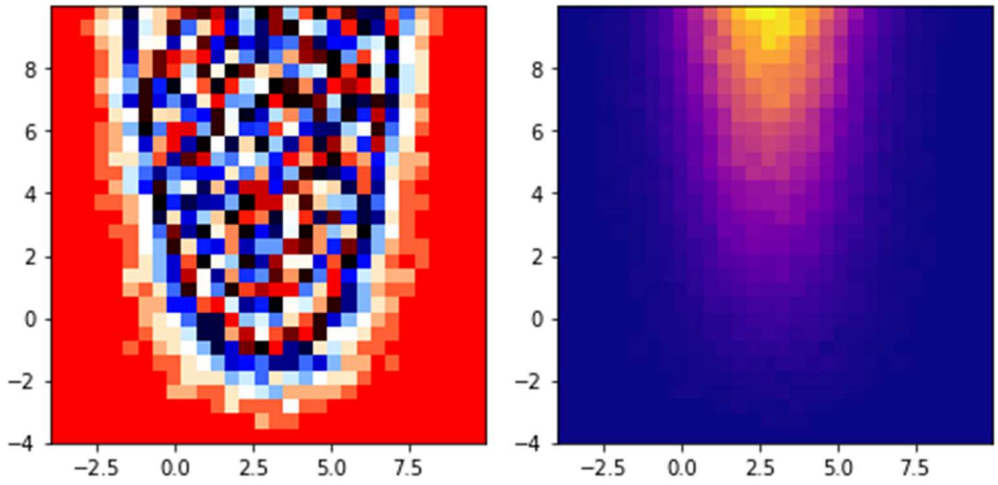
Při výběru barev je důležité vzít v potaz i cílovou skupinu. Příkladem může být vizualizace určená osobám s částečnou barvoslepostí (Protanopie, Tritanopie atd.). V takovém případě není vhodné volit barvy, které daná skupina nerozpozná od sebe.

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, rozptýlené nebo kvalitativní. Ukázky těchto map lze vidět na následujícím obrázku (ve stejném pořadí od shora).




**Obr. 2 Ukázky kategorií barevných map**
Zdroj: Dokumentace Matplotlib [4], upraveno na 3 ukázky z každé kategorie

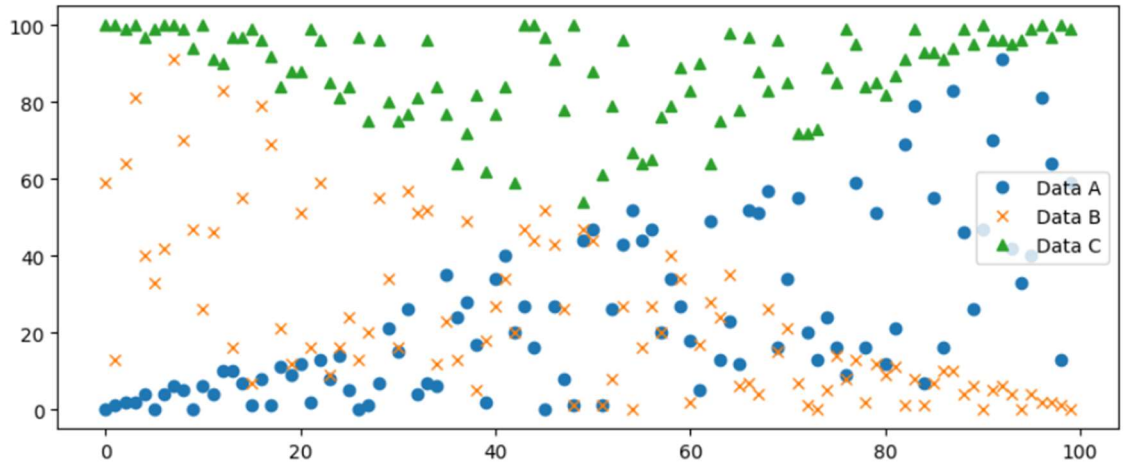
Důsledkem zvolení nevhodné barevné palety trpí nejen estetická stránka grafu, ale i jeho schopnost předávat informace. Na následujícím obrázku lze vidět nevhodnou (levá strana) a vhodnou (pravá strana) selekci barev.




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

## Markery

V oblasti vizualizace dat jsou názvem markery označovány grafické symboly různých tvarů a barev využívané k reprezentaci jednotlivých datových bodů grafu. Markery se obvykle používají v bodových grafech a dalších typech grafů, kde je potřeba vizuálně odlišit jednotlivé datové body. Tvar a barva markeru může sloužit k odlišení jednotlivých datových řad, což usnadňuje interpretaci dat.




**Obr. 4 Markery**
Zdroj: vlastní zpracování pomocí Matplotlib

## Osy grafu

Osy grafu hrají zásadní roli v designu a čitelnosti vizualizací. Jsou klíčovým prvkem, který umožňuje uživateli interpretovat hodnoty v grafu. Dobře zvolené osy poskytují referenční rámec pro měření hodnot a umožňují přesné zachycení rozložení dat. Správné umístění, vhodný výběr měřítka a přesné popisky os jsou důležité pro čitelnost vizualizace. Vhodné formátování textu osy a vhodné rozsahy hodnot zajišťují jednoduchou interpretaci grafu. Design os také zahrnuje odstranění přebytečných prvků, tím lze dosáhnout zpřehlednění rozložení a zlepšení estetiky, která přispívá k celkovému dojmu vizualizace. Stupnice osy nemusí být pouze lineární, pro některé aplikace může být vhodnější logaritmická stupnice.

## Legenda grafu

Legenda grafu má svojí roli v designu vizualizací dat. Jejím účelem je poskytnout uživatelům informace o významu barev, symbolů nebo jiných vizuálních prvků, které jsou využity v grafu. Správně navržená legenda by měla být přehledná a umístěna tak, aby nezakrývala důležité informace. Volba vhodného popisku, velikosti a formátu textu, přispívá ke snadnému porozumění a interpretaci dat.

### Mřížka grafu

Mřížka grafu je důležitým prvkem designu, výrazně ovlivňuje čitelnost prezentovaných dat. Její role spočívá v poskytování vizuální struktury a usnadňování interpretace hodnot grafu. Vhodně nastavená mřížka pomáhá vizuálně porovnávat hodnoty nebo identifikovat chybná data. Hustota a styl mřížky by měly být zvoleny tak, aby nepřekrývala samotná dat, ale zároveň poskytovala dostatečný vizuální orientační bod. Kvalitní mřížka je neintruzivní, jednoduchá a dobře čitelná. Je důležité přizpůsobit mřížku konkrétním potřebám prezentovaných dat.

### Titulky, popisky a anotace

Titulky, popisky a anotace jsou klíčovými prvky v designu a čitelnosti grafů. Titulky slouží k jasnější identifikaci obsahu grafu a předávání klíčových informací. Popisky, umístěné při jednotlivých osách, pomáhají interpretovat hodnoty a jednotky zobrazených dat. Anotace jsou používány k vyznačení specifických bodů nebo oblastí na grafu, čímž umožňují detailnější analýzu. Správné umístění, čitelná velikost a vhodný styl písma těchto prvků přispívají k lepšímu pochopení dat a estetickému provedení. Vhodně umístěné a informačně bohaté titulky, popisky a anotace usnadňují interpretaci dat.

# 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 pomocí funkce read. Tato funkce umožňuje načítat různé formáty souborů, včetně csv, txt, xls a dalších. Stačí specifikovat cestu k souboru a Pandas automaticky načte data a vytvoří dataframe. Tímto způsobem můžeme snadno začít pracovat s daty a využít bohatou funkcionalitu kterou Pandas nabízí.

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

Pandas Series je datový typ pro skladování hodnot v čase. Může být snadno konvertován na dataframe nebo použit při vytváření nového dataframe. Series představuje jednorozměrné pole s indexem, který umožňuje efektivní manipulaci s daty a jejich analýzu.

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

Proces vizualizace začíná sběrem samotných dat a jejich případným převodem do digitální podoby. Výsledkem je soubor dat v digitální podobě, mezi nejčastěji používané formáty patří csv nebo xls.

Následujícím krokem je načtení těchto dat do vhodných datových struktur, které jsou kompatibilní s vizualizačními balíčky. K tomuto účelu se často využívají balíčky jako Numpy nebo Pandas.

Před samotnou vizualizací je často nutné provést čištění dat a případné opravy formátu. Mezi běžné operace patří odstranění duplikátů, prázdných řádků a dalších nežádoucích prvků jako například chybných hodnot. Naštěstí tyto balíčky nabízejí velký výběr funkcí, které slouží k usnadnění a zrychlení těchto operací.

Na takto zpracované sadě dat lze provádět další operace, jako je přetypování časových údajů, zahození nepotřebných sloupců, převzorkování a mnoho dalších.

Takto zpracovaná data jsou poté vizualizována pomocí vizualizačních balíčků.

# 6 Ekosystémy

V kontextu pythonu se pojmem ekosystém rozumí široká sada balíčků a nástrojů, které má vývojář k dispozici. V tomto případě se jedná o skupinu balíčků, které jsou přímo závislé nebo jinak podporují jednu ze zvolených vizualizačních knihoven.

## *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].

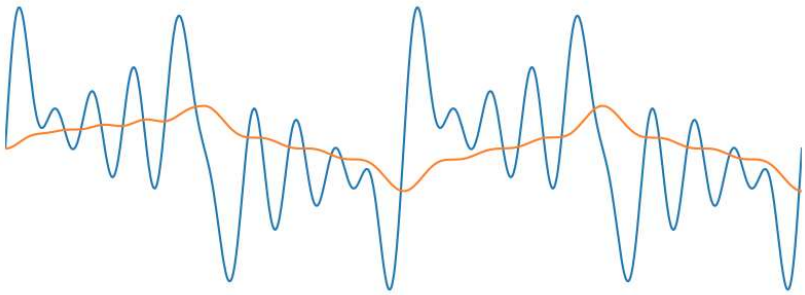
## *6.3 Vega*

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].

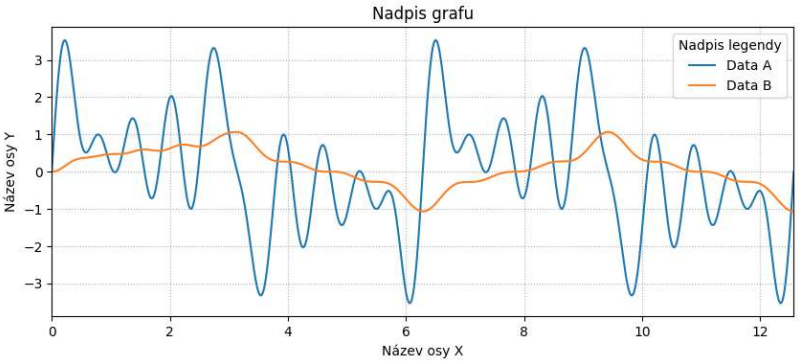
# 7 O grafu

## 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. 5 Graf bez kontextu**
Zdroj: vlastní zpracování pomocí Matplotlib




**Obr. 6 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 jsou používané k zobrazení rozložení dat. Tyto grafy poskytují informace o tom, jak jsou data rozdělena a jaké jsou jejich charakteristiky, jako je střední hodnota, rozptyl, špičatost apod.

### Korelační grafy

Korelační grafy slouží k analýze vztahu mezi proměnnými. Pomocí těchto grafů můžeme identifikovat a vyhodnotit vztah mezi dvěma nebo více proměnnými v datasetu. Bodový graf je nejběžnějším korelačním grafem, zobrazuje body v rovině a umožňuje vizuálně hodnotit vztahy. Při interpretaci korelačních grafů je důležité brát v úvahu to, že korelace nemusí znamenat příčinný vztah mezi proměnnými, ale pouze poukazuje jejich vzájemnou souvislost.

### Ranking grafy

Ranking grafy (žebříčky) jsou používané k porovnávání hodnot mezi různými kategoriemi. Tyto grafy zobrazují data ve formě sloupců, kde výška sloupce představuje hodnotu. Slouží k rychlému porovnání relativních hodnot mezi prvky a jsou široce využívány ve mnoha oblastech. Grafy mohou být vertikální nebo horizontální, a často mohou obsahovat další informace jako třeba popisky. Žebříčky jsou efektivním způsobem vizualizace dat za účelem snadného a přehledného porovnání jednotlivých kategorií datasetu.

### Části celku

Části celku slouží k zobrazení podílu jednotlivých částí na celkové hodnotě. Tyto grafy se často využívají k prezentaci struktury dat, kde každá část je reprezentována jako segment v kruhovém diagramu. Segmenty jsou velké podle svého podílu na celkové hodnotě a umožňují rychlé porovnání mezi různými kategoriemi. Tento typ grafu je užitečný pro vizuální vyjádřen í rozložení podílů, a to zejména ve statistice, marketingu a dalších oborech.

### Vývoj v čase

Grafy z kategorie vývoj v čase slouží k vizuálnímu zobrazení změn a trendů. Často používané typy grafů v této kategorii zahrnují čárové a spojnicové grafy. Tyto grafy jsou velmi užitečné při sledování, porovnávání a určování trendů, sezónních vzorců a dlouhodobých změn. Grafy reprezentující vývoj hodnot v čase jsou široce

využívány v oblastech jako ekonomie, energetika, meteorologie a mnoha dalších, kde je sledování dat v čase klíčové pro pochopení jejich dynamiky a vývoje.

### Mapy

Grafy z kategorie mapy slouží k zobrazení geografických dat a informací na geografické mapě. Tyto grafy umožňují vizualizaci prostorové distribuce různých jevů, například demografických údajů, ekonomických ukazatelů nebo rozložení přírodních zdrojů. Využití těchto grafů je časté v oblastech, jako je geografie, ekologie nebo také doprava a plánování.

### Flow

Grafy z kategorie flow jsou využívány pro zobrazení toků nebo interakcí mezi jednotlivými prvky v rámci určitého systému. Tyto grafy graficky reprezentují uzly a hrany pomocí markerů a čar. Tento druh vizualizace je často využíván v rámci diskrétní matematiky. Grafy z této kategorie jsou důležitým nástrojem při analýze a modelování různých situací, jako je doprava, logistika, sociální sítě, elektrické sítě a další. Diskrétní matematika poskytuje teoretický základ pro práci s grafy a umožňuje provádět rozsáhlé analýzy jejich vlastností.

# 8 Metodika zpracování

## 8.1 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.

## 8.2 Náročnost implementace

Pro účely objektivního porovnání náročnosti implementace grafů byl využit počet řádků potřebný k implementaci. Pro zachování konzistence byl kód byl automaticky zformátován dle PEP 8. Mezi řádky kódu nejsou započítávány komentáře a prázdné řádky. Některé ukázky potřebují kód pro běh v Jupyter notebooku, tento kód není potřeba pro běh mimo Jupyter a není proto započítáván. Tento ukazatel náročnosti je vhodný pro delší ukázky. Výsledná Tabulka obsahuje i krátké ukázky pro zachování konzistence.

Dalším alternativním ukazatelem náročnosti je počet znaků. Tento ukazatel je však méně vhodný. Hlavním problémem je, že počet znaků nemusí přesně odrážet náročnost implementace. To je způsobeno použitím zkratek (Pandas jako pd, Numpy jako np atd.) nebo proměnnou délkou názvů proměnných, které nemají přímý vliv na složitost.

## 8.3 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 načtena a předzpracována knihovnou networkx. Datová sada Enzymes8 je volně dostupná ke stažení na webu Network Repository [19].

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

## 8.4 Popularita v komunitě

Popularitu ekosystémů v komunitě je těžké přesně určit. Lze však vycházet z několika zdrojů informací pro vytvoření přesnější představy.

**Stack Overflow**

Na platformě Stack Overflow je možné pomocí dotazu *„[ekosystem] created:2022 isaccepted:yes*“ získat počet zodpovězených otázek za rok 2022 pro konkrétní ekosystém. Což poskytuje cenný náhled do popularity a používání tohoto ekosystému v komunitě.

### Python package index

Na zjištění popularity jednotlivých knihoven může sloužit také počet stažení. Data o historii stažení knihoven z PyPi jsou ukládána a volně přístupná pomocí Google BigQuery (bigquery-public-data.pypi.file_downloads). Za pomocí SQL dotazů je možné jednoduše získat informace o počtu stažení jednotlivých balíčků. Je však třeba poznamenat, že počet stažení neodráží přesný počet skutečných uživatelů. Výsledná čísla mohou být zkreslena situacemi, kdy uživatel má více zařízení nebo kdy si balíček stáhl, ale již ho nepoužívá. Přesto tato data poskytují užitečný ukazatel popularity knihoven a jejich šíření v komunitě.

### Github stars

Pro sledování vývoje popularity balíčků na platformě GitHub lze využít nástroj na stránce https://seladb.github.io/ který vytváří graf historie počtu hvězd pro každý balíček. Tento projekt také dále počítá průměrný denní přírůstek uživatel, kteří projektu dali hvězdu, od dne založení balíčku. Toto lze také využít jako další ukazatel popularity balíčku.

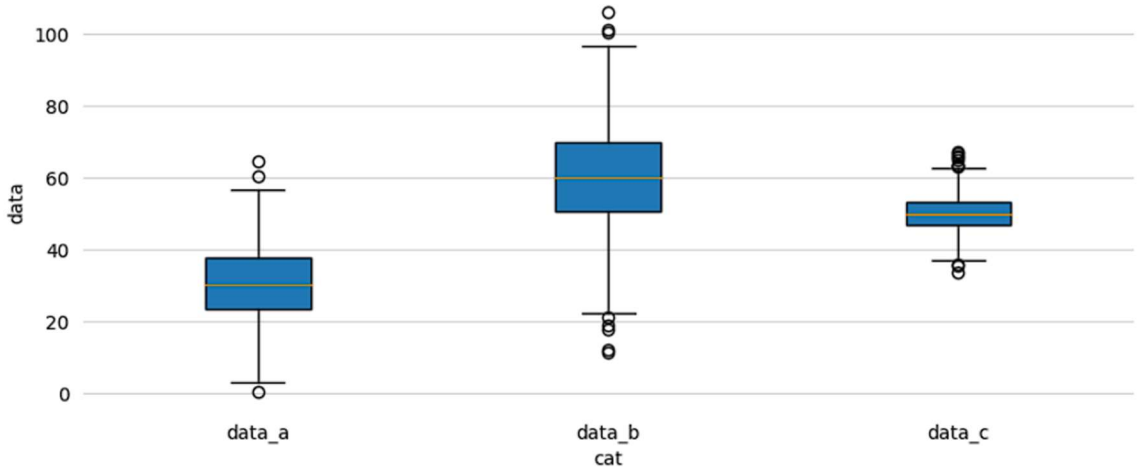
# 9 Ukázky ekosystémů

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 Boxplot*

Boxplot neboli krabicový graf slouží k vizualizaci distribuce dat. Je tvořen obdélníkem, jehož výška je určena pomocí rozpětí mezi prvním a třetím kvartilem. Vodorovná čára uvnitř označuje medián. Vnější markery reprezentují extrémní hodnoty, které jsou mimo rozpětí. Boxplot poskytuje rychlý přehled o rozptylu a přítomnosti odlehlých hodnot v datech. Je užitečný při porovnávání distribucí mezi různými skupinami a odhalování anomálií. Boxplot je důležitý nástroj ve statistice a je snadno implementovatelný ve všech porovnávaných knihovnách.

```
from matplotlib import pyplot as plt

plt.figure(figsize=(10, 4), dpi=100) #10 inch * 100 dpi = 1000px
plt.boxplot(data, labels=data.columns, patch_artist=True)
plt.xlabel("cat")
plt.ylabel("data")
plt.grid(axis="y", alpha=0.65)
plt.tick_params(bottom=False, left=False)
plt.gca().spines[:].set_visible(False)
plt.show()
```

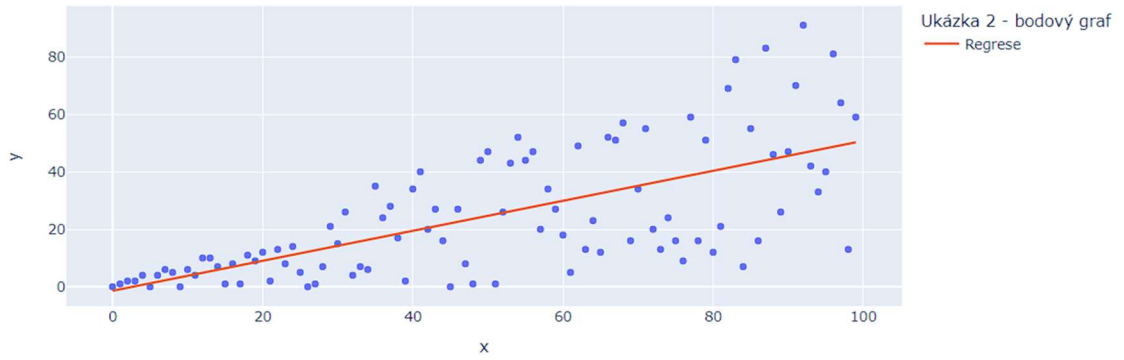



**Obr. 7 Boxplot a implementace**
Zdroj: vlastní zpracování pomocí Matplotlib

## 9.2 Bodový graf

Bodový graf, který spadá do kategorie korelačních grafů, slouží k vizuálnímu zobrazení vztahů mezi dvěma hodnotami. Každý bod na grafu reprezentuje jednu dvojici hodnot. Tento typ grafu umožňuje porovnání hodnot a identifikaci jejich vzájemného vztahu. Bodový graf se často používá při analýze statistických dat, studiu korelace mezi dvěma proměnnými a identifikaci případných odchylek. Implementace bodového grafu je snadná. Přidání regresivní přímky je náročnější a vyžaduje využití Numpy ale i tak je stále velmi jednoduché.

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

fig = px.scatter(data, x="x", y="y", width=1000, height=400)
fig.add_trace(go.Scatter(x=data["x"], y=line_y, mode="lines", name="Regrese"))
fig.update_layout(legend=dict(title="Ukázka 2 - bodový graf"))
fig.show()
```

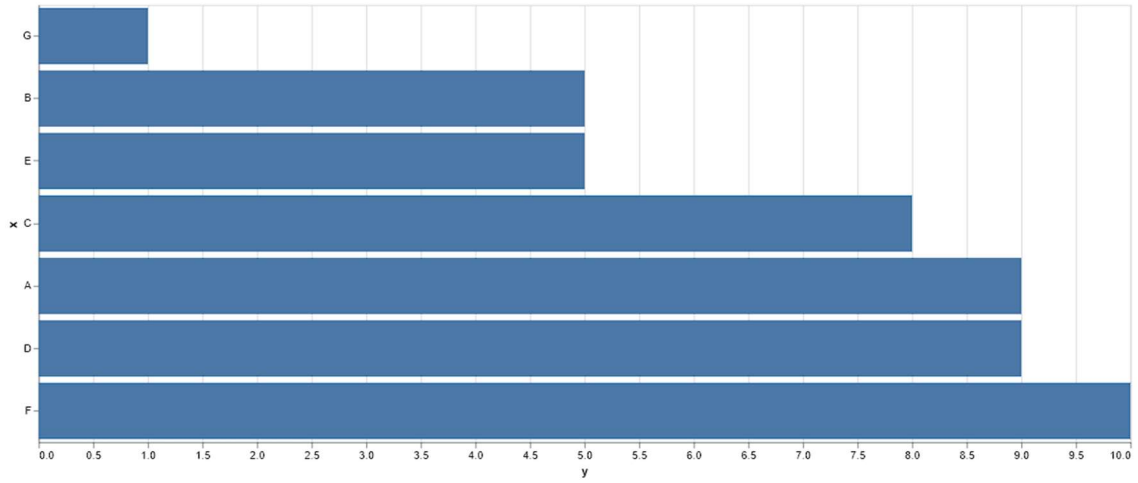



**Obr. 8 Bodový graf a jeho implementace**
Zdroj: vlastní zpracování pomocí Plotly

## 9.3 Sloupcový graf

Sloupcový graf patří do kategorie žebříčků, slouží k vizuálnímu znázornění vztahu mezi numerickou hodnotou a jiným parametrem. V tradiční podobě představuje výška sloupce numerickou hodnotu. Tento typ grafu je často využíván pro porovnávání hodnot mezi různými kategoriemi nebo prvky. Sloupcové grafy mohou mít různé varianty, jako jsou jednoduché sloupce nebo vrstvené sloupce. Pro některé datové sady je vhodnější mít sloupce horizontální. Implementace sloupcového grafu je ve všech ekosystémech jednoduchá.

```
import altair as alt

#Altair vykresluje opačně než ostatní
data = data.sort_values(by="y", ascending=True)
alt.Chart(data).mark_bar().encode(
    x=alt.X("y:Q"),
    y=alt.Y("x:O",sort=None)
).properties(width=1000, height=400)
```




**Obr. 9 Sloupcový graf a jeho implementace**
Zdroj: vlastní zpracování pomocí Vega

## 9.4 Výsečový graf

Výsečový graf, také známý jako koláčový graf, je jedním z nejčastěji používaných grafů, který slouží k vizualizaci části z celku. Kruh je rozdělen na části, které reprezentují jednotlivé kategorie. Velikost částí se liší podle hodnot, které mají reprezentovat. Tento typ grafu je často využíván k přehlednému zobrazení procentuálního rozložení nebo podílu jednotlivých částí v celku. Implementace výsečového grafu v ekosystémech Matplotlib a Plotly je podobná. Oproti tomu Vega neumí využít index datové sady. Je tedy potřeba index zkopírovat do samostatného sloupečku, aby se kategorie zobrazila správně.

Matplot

```
import matplotlib.pyplot as plt

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

**Obr. 10 Výsečový graf a implementace**
Zdroj: Vlastní zpracování

## 9.5 Choropleth

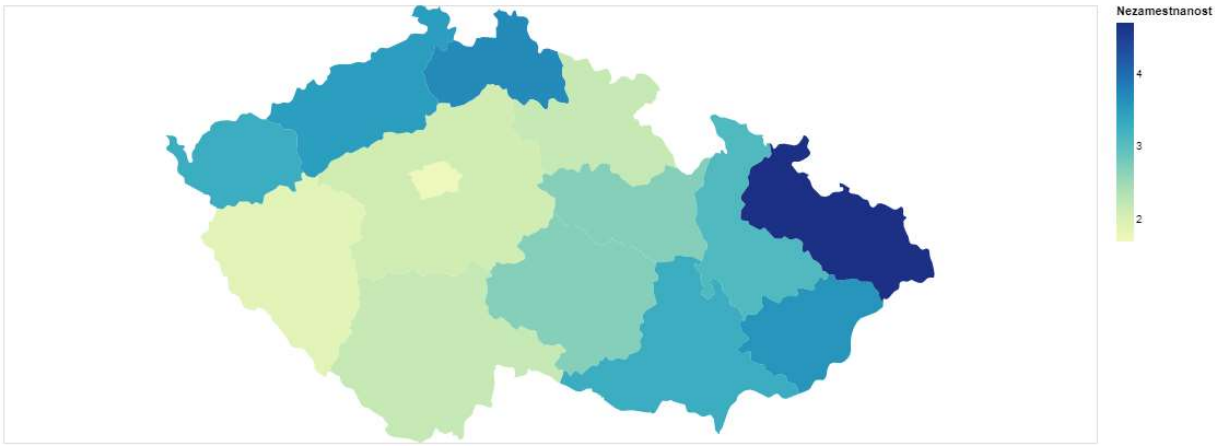
Choropleth patří do kategorie map. Jeho hlavním účelem je prezentovat data na mapě, toho dosahuje zbarvením částí mapy v souladu s přiřazenými hodnotami. Tento typ grafu je často využíván pro reprezentaci statistických údajů nebo ukazatelů, jako je například procentuální nezaměstnanost. Jedná se o efektivní způsob vizualizace prostorových dat, který umožňuje snadné porovnání hodnot v různých oblastech.

Nejsnazší na implementaci se ukázal Altair. Implementace je oproti ostatním ekosystémům velmi snadná. Stejně jako Matplotlib podporuje využití Geopandas. Matplotlib je vykreslován pomocí knihovny Geoplot, oproti ekosystému Vega vyžaduje více parametrů a využívá přídavnou knihovnu.

Plotly je v tomto případě nejnáročnější na implementaci. Vyžaduje velké množství parametrů a dalších úprav pro dosažení identického výsledku. Bohužel neumí plně využít GeoDataFrame, je tedy potřeba tvary krajů exportovat jako geojson, který poté Plotly již vykreslit dokáže.

Vega

```
import altair as alt

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

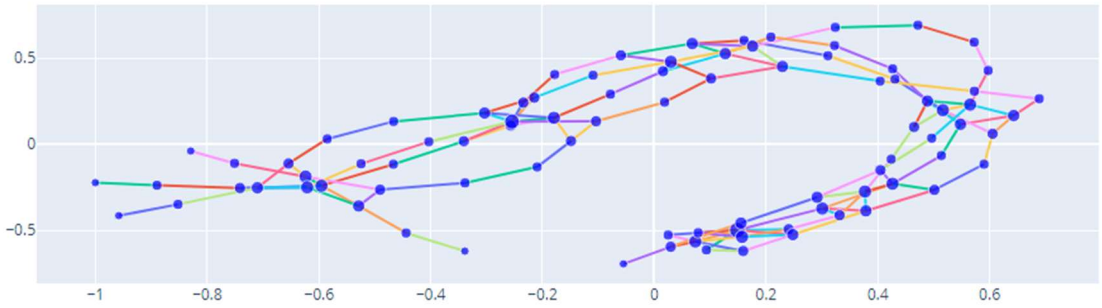



**Obr. 11 Choropleth a implementace**
Zdroj: vlastní zpracování pomocí Vega

## 9.6 Síťový graf

Síťový graf je efektivní nástroj pro vizualizaci komplexních sítí a vztahů mezi jednotlivými prvky. Při porovnání náročnosti implementace síťových grafů je zjevný rozdíl. Balík Networkx, který slouží k načtení sítě má vlastní vykreslovací funkce založené na Matplotlib. Což Matplotlibu poskytuje výhodu oproti ostatním.

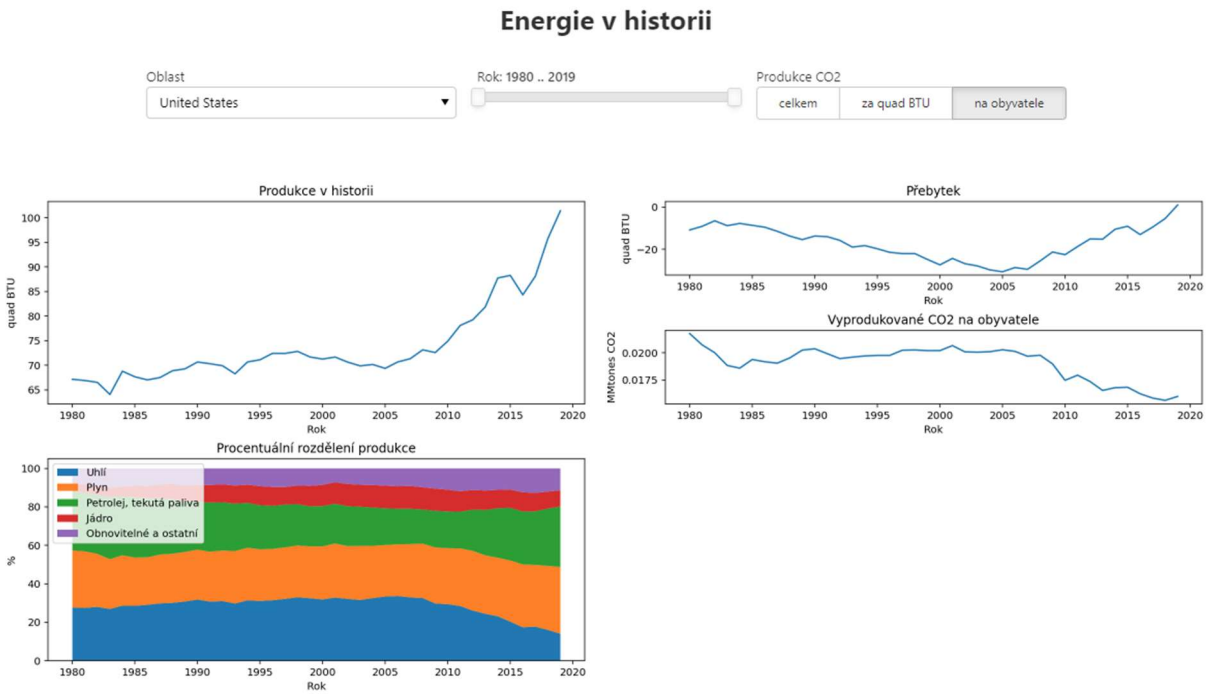
```
import plotly.graph_objects as go
import plotly.io as pio
pio.renderers.default = "notebook"

fig = go.Figure()
for edge in gr.edges:
    x0, y0 = pos[edge[0]]
    x1, y1 = pos[edge[1]]
    fig.add_trace(go.Scatter(x=[x0, x1], y=[y0, y1], mode="lines"))
x, y = zip(*pos.values())
fig.add_trace(go.Scatter(x=x, y=y, mode="markers",
              marker=dict(size=node_sizes_plotly, color="blue")))
fig.update_layout(width=1000, height=400, showlegend=False,)
fig.show()
```




**Obr. 12 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. 13 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. Témata obvykle mění všechny vizuální aspekty grafu. Vega bohužel žádnou takovou funkcionalitu témat 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 zhodnocení každé knihovny. Konečné porovnání ekosystémů je uvedeno na konci této kapitoly.

## *10.1 Matplotlib*

Nejstarší z porovnávaných ekosystémů a zároveň nejdéle ve vývoji. V současné době udržovaný komunitou jako open-source knihovna. Jednou z hlavních výhod je početná a aktivní komunita uživatelů. Díky tomu existuje velké množství kompatibilních knihoven, které rozšiřují vizualizační možnosti. Příkladem jsou knihovny Seaborn a Bokeh, které přidávají interaktivitu a mění celkový vzhled výsledných grafů. Matplotlib je osvědčeným a vysoce rozšířeným nástrojem pro vizualizaci dat.

## *10.2 Plotly*

Plotly je v některých případech složitější na implementaci. Ve srovnání s ostatními ekosystémy ale poskytuje subjektivně nejlepší vizuální výsledky. Silnou stránkou Plotly je schopnost vytvářet interaktivní vizualizace a snadno s nimi pracovat díky skupině balíčků Plotly Express, Plotly Graph Objects a Dash.

Plotly Express poskytuje jednoduchý způsob tvorby grafů s minimálním množstvím kódu. Na druhou stranu, Plotly Graph Objects umožňuje vytvářet pokročilejší grafy. A balíček Dash umožňuje vytvářet interaktivní dashboardy. Je důležité poznamenat, že v některých specifických případech, může být vizualizace pomocí Plotly problémová. Nicméně, i tak platí, že Plotly je silným nástrojem pro tvorbu interaktivních vizualizací.

## *10.3 Vega*

Ekosystém Vega je svým způsobem outsider, a jeho současná popularita je značně nižší. Pro implementaci v jazyce Python je používána knihovna Altair, jejíž popularita v poslední době prudce stoupá. Své početné nedostatky kompenzuje náročností implementace, která je velice jednoduchá. Pokročilé vizualizace jsou bohužel dosti problémové.

## *10.4 Porovnání ekosystémů*

### Náročnost implementace

| | Matplotlib | Plotly | Vega |
|---|---|---|---|
| 1 - Boxplot | 9 | **4** | 6 |
| 2 - Bodový graf | **6** | **6** | 11 |
| 3 - Sloupcový graf | 4 | **3** | 5 |
| 4 - Výsečový graf | 4 | **3** | 6 |
| 5 - Choropleth | 15 | 17 | **4** |
| 6 - Síťový graf | **5** | 11 | 15 |
| 7 - Dashboard | 71 | 91 | **43** |

**Tabulka 1 – Náročnost implementace ukázek na řádky**
Zdroj: vlastní zpracování

Náročnost implementace je těžké odvodit pouze z počtu řádků, tato tabulka tedy slouží spíše k dokreslení představy než jako primární zdroj pro porovnání. U primitivních ukázek 1,3,4 se obtížnost implementace pohybuje na stejné úrovni. Rozdíly se začnou projevovat až u ukázek náročnějších. Kde je náročnost Plotly značně vyšší. Množství řádků, které vyžaduje Altair je malé, ale po započítání komplikací spojených s neohebností je celková náročnost balíku Matplotlib menší. Konečné pořadí s ohledem na subjektivní názor je následující. Matplotlib na prvním místě, poté Altair (Vega) a na konec Plotly.

### Popularita ekosystémů

Rozdíl v růstu hodnot mezi roky 2021 a 2022

| | Matplotlib | Plotly.py | Altair (Vega) |
|---|---|---|---|
| Počet stažení | 51 720 739 | -1 975 611 | **64 147 965** |
| Počet hvězd | **51** | -230 | -100 |
| Počet dotazů | -1107 | **-23** | -25 |

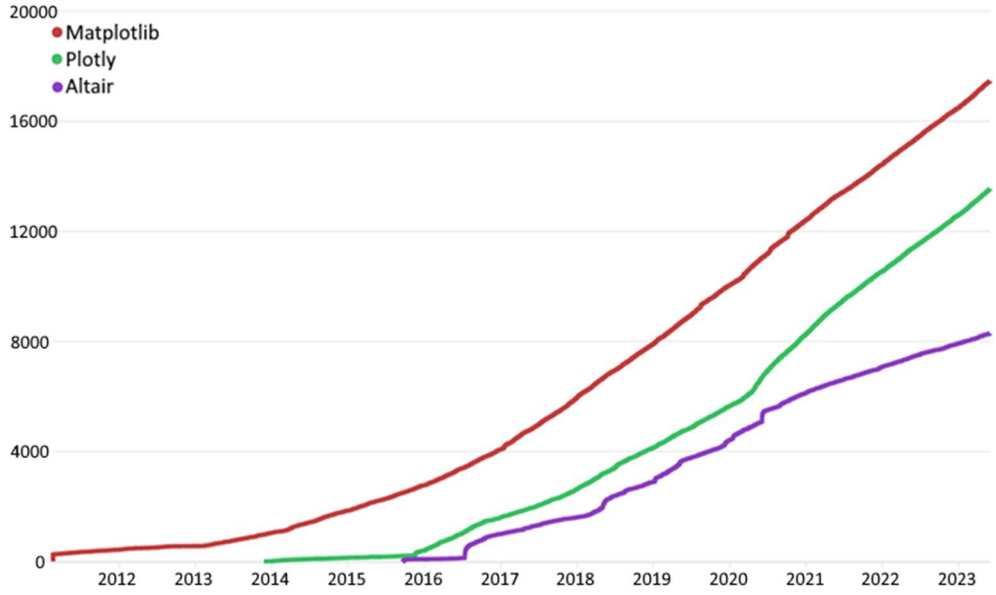
**Tabulka 2 – Rozdíl mezi roky 2021 a 2022**
Zdroj: vlastní zpracování

Průměrný denní nárůst počtu hvězd od dne založení Git repositáře. K 30. 5. 2023.

| | Matplotlib | Plotly.py | Altair (Vega) |
|---|---|---|---|
| Nárůst hvězd | 3,895 | **3,899** | 2,949 |

**Tabulka 3 – Průměrný nárůst hvězd za den**
Zdroj: https://seladb.github.io/

Balíček Altair zaznamenal nárůst o více než 64 milionů v počtu stažení za rok 2022 oproti roku 2021, čímž překonal i Matplotlib s 51 miliony. Plotly oproti tomu zaznamenal úpadek ve všech třech kategoriích ale i přesto dopadl nejlépe v počtu zodpovězených dotazů. Matplotlib byl jediný balíček, který se dočkal růstu ročního příbytku hvězd. Nejrychleji rostoucím balíčkem dle počtu hvězd je momentálně Plotly.




**Obr. 14 Historie počtu hvězd balíčků**
Zdroj: https://seladb.github.io/ upraveno pro lepší čitelnost

## Licence

Všechny srovnávané ekosystémy jsou open-source, stejně tak jako knihovny, které je implementují. Uživatel tak může v případě potřeby modifikovat zdrojový kód nebo vytvořit svůj vlastní balíček, postavený na jednom z ekosystémů.

| | Matplotlib | Plotly | Vega |
|---|---|---|---|
| Licence | Vlastní, založena na PSF | MIT License | BSD 3 – Clause |

**Tabulka 4 – Licence ekosystémů**
Zdroj: Jednotlivé ekosystémy [4][18][6]

## Konečné srovnání

Všechny tři ekosystémy lze doporučit. Práce s ekosystémem Matplotlib byla velmi usnadněna, jeho jednoduchostí a množstvím dostupných návodů. Ekosystém Plotly se ukázal jako výborný pro tvorbu interaktivních grafů a dashboardů, ale implementace je náročnější. Altair velmi dobře zvládá základní vizualizace, ale tvorba jakékoliv komplexní vizualizace s více prvky je frustrující a vyžaduje různé pomocné kousky kódu. Následující tabulka obsahuje subjektivní hodnocení aspektů a vhodnosti jednotlivých ekosystémů pro konkrétní využití. Hodnocení je udělováno na škále od 1 do 3, přičemž hodnota 1 odpovídá maximu.

| | Matplotlib | Plotly | Vega |
|---|---|---|---|
| Vhodnost pro využití k tvorbě | | | |
| Jednoduchá statická vizualizace | 1 | 1 | 1 |
| Komplexní statické vizualizace | 2 | 1 | 3 |
| Jednoduchá interaktivní vizualizace | 2 | 1 | 2 |
| Dashboard | 2 | 2 | 3 |
| Aspekty | | | |
| Dokumentace | 1 | 1 | 2 |
| Přívětivost k uživateli | 1 | 2 | 2 |
| Komunita | 1 | 2 | 2 |

**Tabulka 5 – Konečné srovnání ekosystémů**
Zdroj: Vlastní zpracování

# 11 Závěr

Tato práce byla zaměřená na představení, porovnání a implementaci tří vybraných ekosystémů pro vizualizaci dat v jazyce Python. Práce poskytuje náhled na tyto ekosystémy a jejich schopnosti v oblasti vizualizace dat. Vzhledem k široké škále funkcí a specifických implementací není možné prohlásit jeden z ekosystémů za celkově nejlepší. Je však možné doporučit jednotlivé ekosystémy v závislosti na plánovaném využití.

Plotly je výborný ekosystém pro tvorbu interaktivních grafů. V ekosystému Matplotlib je možné vytvořit jakoukoliv vizualizaci, hlavně díky velkému množství balíčků, ačkoliv ne vždy se bude jednat o snadné a elegantní řešení. Vega je velice zajímavý ekosystém, který by díky své jednoduchosti implementace mohl jednoho dne dosáhnou stejného statusu jako Matplotlib, momentálně se však dá doporučit jen a pouze pro jednoduché vizualizace.

V budoucnu je možné rozšířit tuto práci o další ekosystémy, které by mohly nabídnout jiné zajímavé vizualizace a přístupy. Výběr implementovaných grafů dle jiného řazení by také přinesl nový pohled na jednotlivé ekosystémy a jejich schopnosti.

# 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

[19] ROSSI, Ryan, A., AHMED, Nesreen, K. The Network Data Repository with Interactive Graph Analytics and Visualization, 2015. [online]. [cit. 1.6.2023]. Dostupné z: http://networkrepository.com

# 13 Přílohy

1) Struktura projektu
2) Notebook – Boxplot
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

# Boxplot

Matěj Kolář 2022

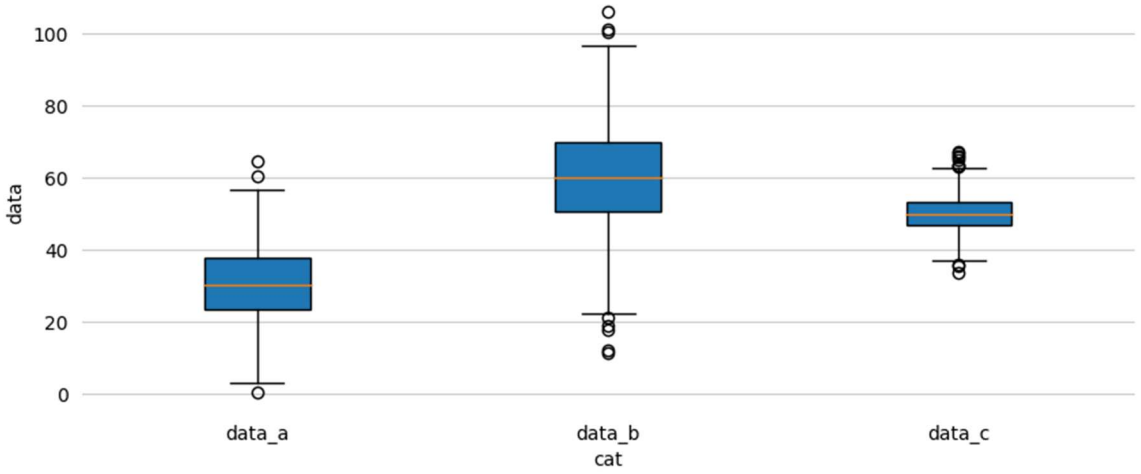
UHK - FIM

## Sdílený kód

```
from numpy import random
import pandas as pd

random.seed(2022) #seed

### Generace dat ###
data_a = []
data_b = []
data_c = []
category = ["a","b","c"]*1000
for x in range(0,1000):
    data_a.append(random.normal(30,10))
    data_b.append(random.normal(60,15))
    data_c.append(random.normal(50,5))

data = pd.DataFrame({"data_a":data_a,"data_b":data_b,"data_c":data_c})
```

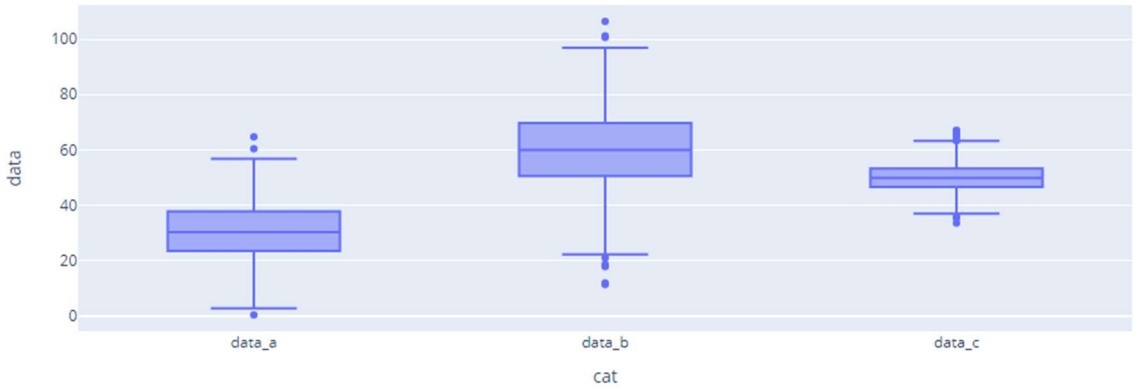
## Matplotlib

```
from matplotlib import pyplot as plt

plt.figure(figsize=(10, 4), dpi=100) #10 inch * 100 dpi = 1000px
plt.boxplot(data, labels=data.columns, patch_artist=True)
plt.xlabel("cat")
plt.ylabel("data")
plt.grid(axis="y", alpha=0.65)
plt.tick_params(bottom=False, left=False)
plt.gca().spines[:].set_visible(False)
plt.show()
```



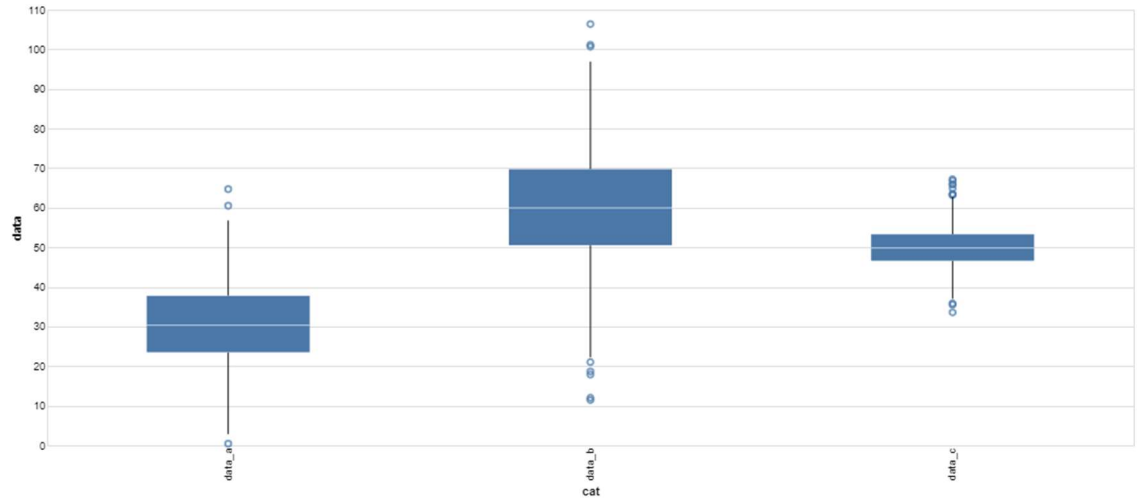
# Plotly

```
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"

labels = {"value": "data", "variable": "cat"}
fig = px.box(data, labels=labels, width=1000, height=400)
fig.show()
```




# Vega

```
import altair as alt

# konverze do formátu pro altair
data_v = data.melt(var_name="cat", value_name="data")

alt.Chart(data_v).mark_boxplot(size=150).encode(
    x="cat:O",
    y="data:Q"
).properties(width=1000, height=400).configure_axis(domain=False, ticks=False)
```



# Bodový graf

Matěj Kolář 2022

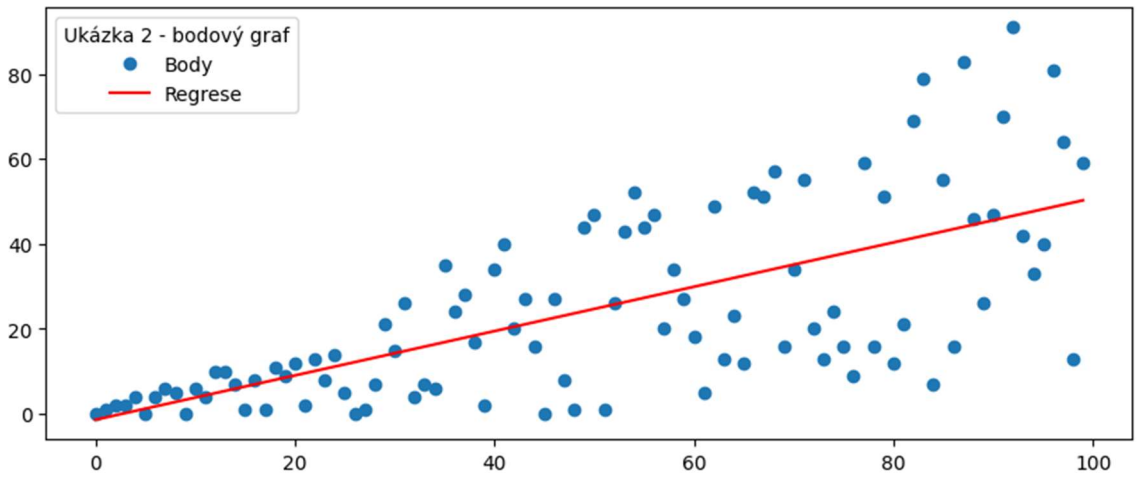
UHK - FIM

## Sdílený kód

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

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})

# generace křivky lineární regrese
coefficients = np.polyfit(data["x"], data["y"], 1)
line_y = np.polyval(coefficients, data["x"])
```

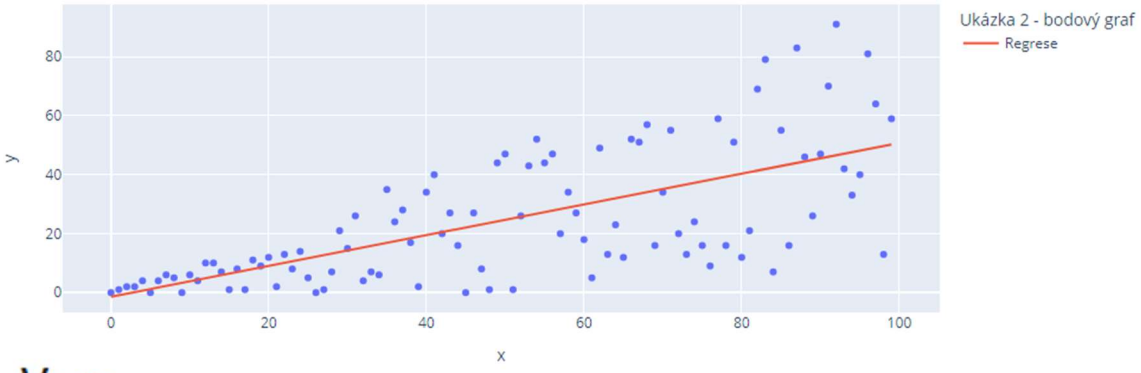
## Matplotlib

```
from matplotlib import pyplot as plt

ax = plt.figure(figsize=(10, 4), dpi=100).add_subplot()
ax.plot(data["x"], data["y"], linestyle="none", marker="o")
ax.plot(data["x"], line_y, color="red")
ax.legend(["Body", "Regrese"], title="Ukázka 2 - bodový graf")
plt.show()
```



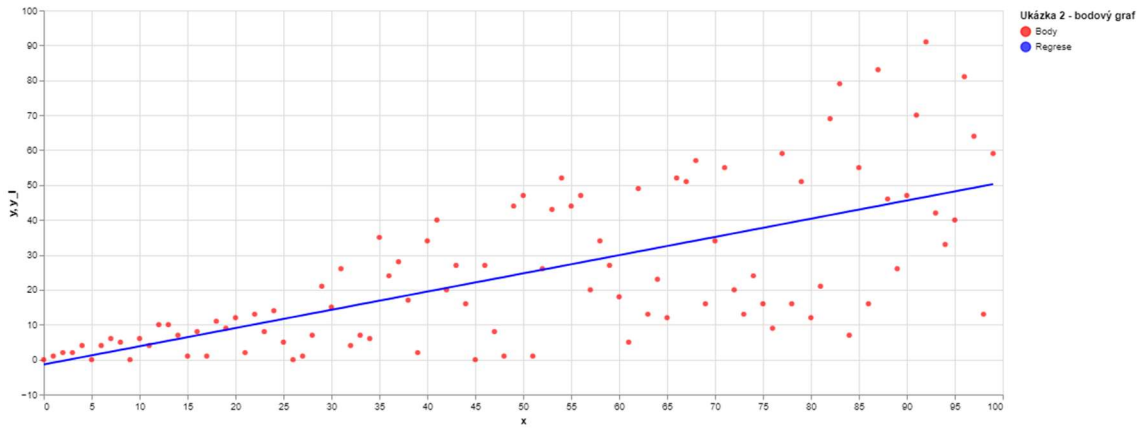
# Plotly

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

fig = px.scatter(data, x="x", y="y", width=1000, height=400)
fig.add_trace(go.Scatter(x=data["x"], y=line_y, mode="lines", name="Regrese"))
fig.update_layout(legend=dict(title="Ukázka 2 - bodový graf"))
fig.show()
```




# Vega

```
import altair as alt

# neohebnost Vegy, nelze nastavit konkrétní barvu nebo název bez tohoto "hacku"
color_scale = alt.Scale(range=["#FF0000", "#0000FF"])
data["name"] = "Body"  # Nutno vložit název jako sloupec
data["line"] = "Regrese"
data["y_l"] = line_y
legend = alt.Legend(title="Ukázka 2 - bodový graf")
scatter = alt.Chart(data).mark_circle().encode(
    x="x", y="y", color=alt.Color("name", legend=legend, scale=color_scale))
line = alt.Chart(data).mark_line().encode(
    x="x", y="y_l", color=alt.Color("line", legend=legend, scale=color_scale))
(scatter+line).properties(width=1000, height=400)
```



# Sloupcový graf

Matěj Kolář 2022

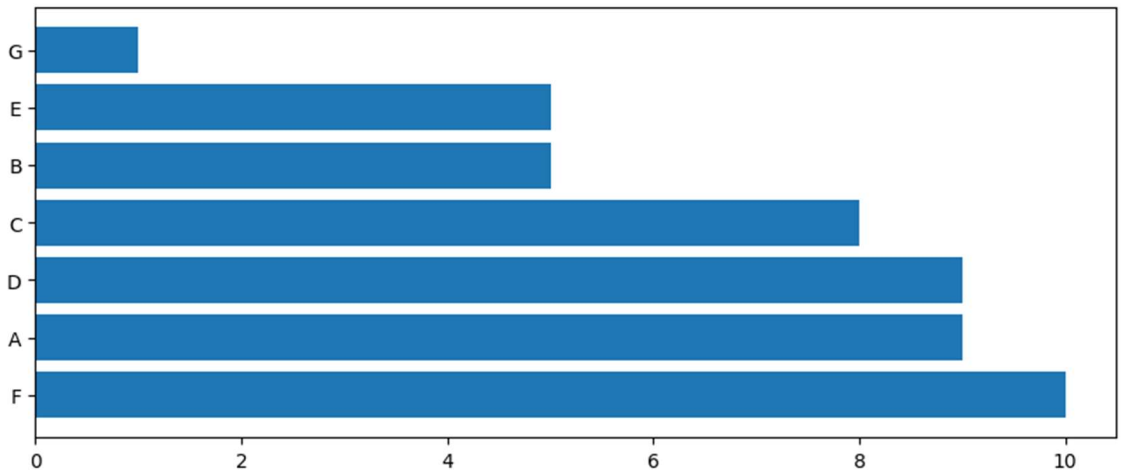
UHK - FIM

## 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})
data = data.sort_values(by="y", ascending=False)
```

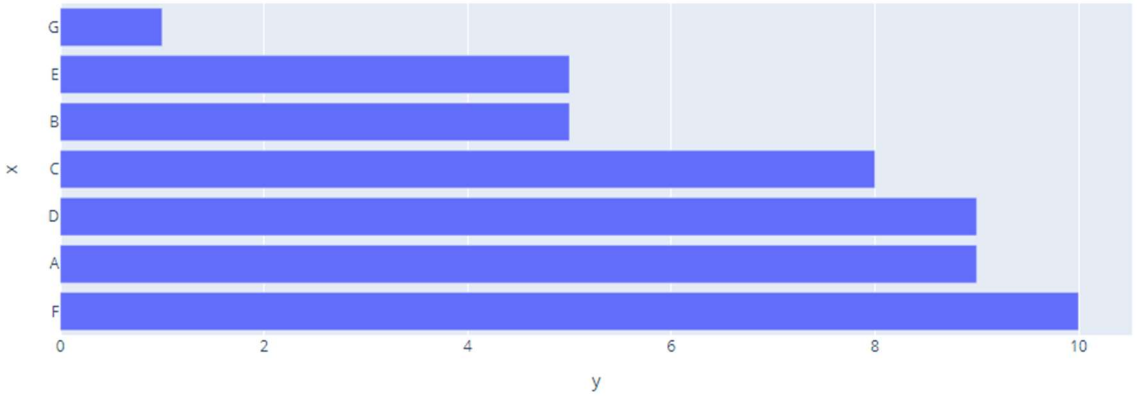
## Matplotlib

```
import matplotlib.pyplot as plt

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



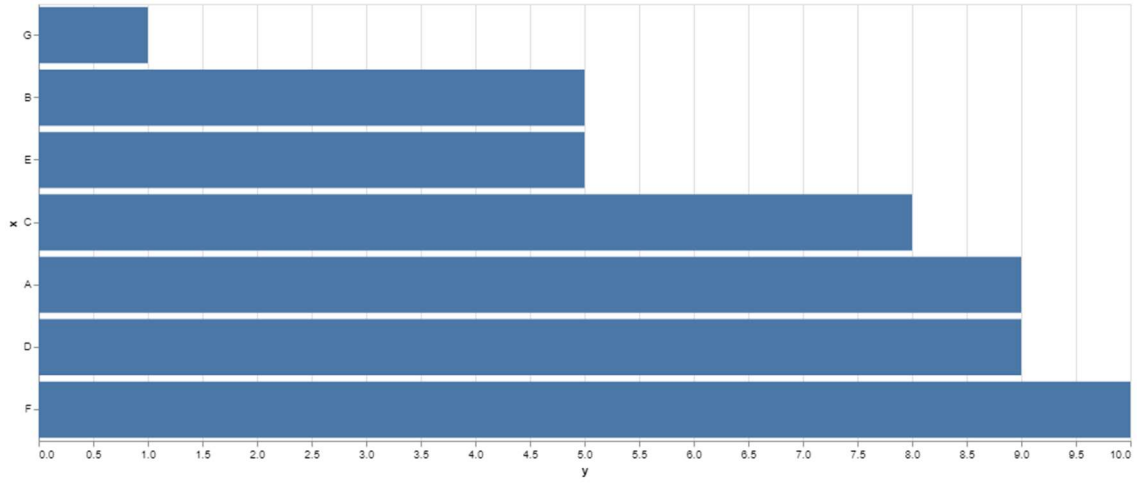
## Plotly

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

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




## Vega

```
import altair as alt

#Altair vykresluje opačně než ostatní
data = data.sort_values(by="y", ascending=True)
alt.Chart(data).mark_bar().encode(
    x=alt.X("y:Q"),
    y=alt.Y("x:O",sort=None)
).properties(width=1000, height=400)
```



# Výsečový graf

Matěj Kolář 2022

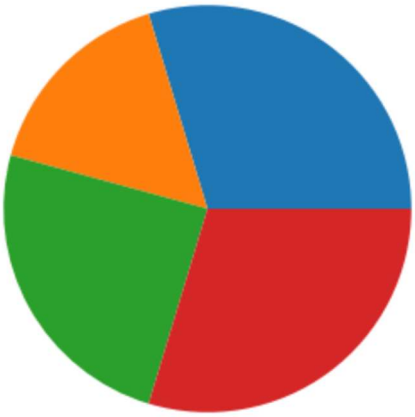
UHK - FIM

## 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})
```

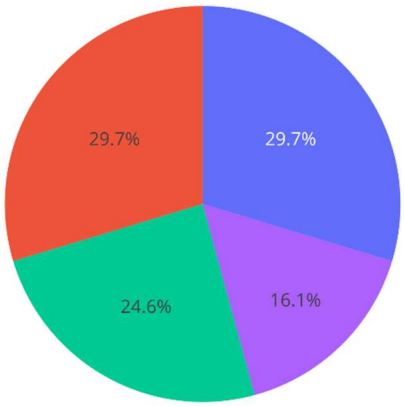
## Matplotlib

```
import matplotlib.pyplot as plt

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



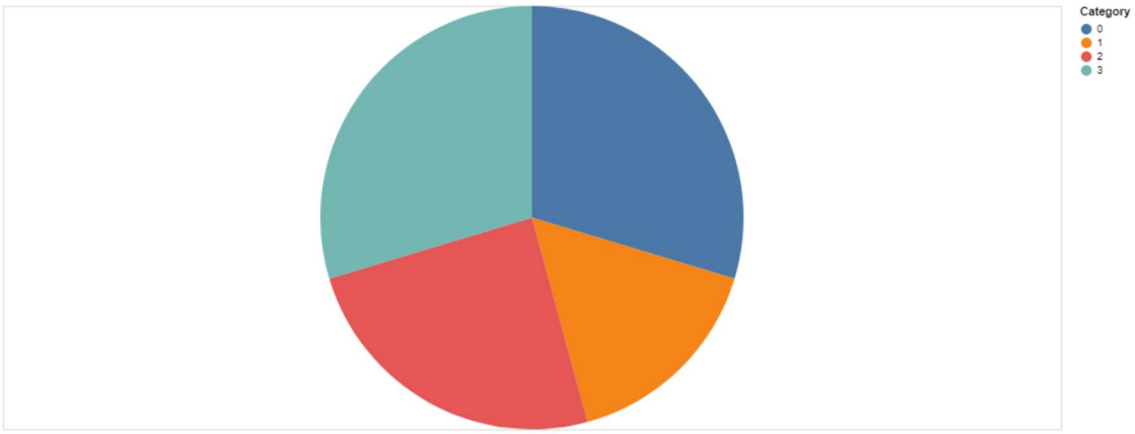
## Plotly

```
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

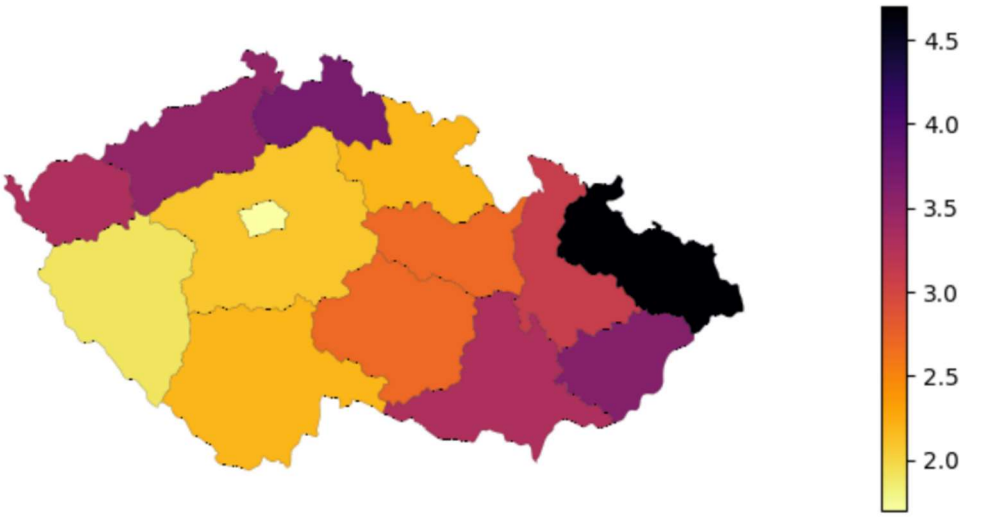
```
import altair as alt

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)
```



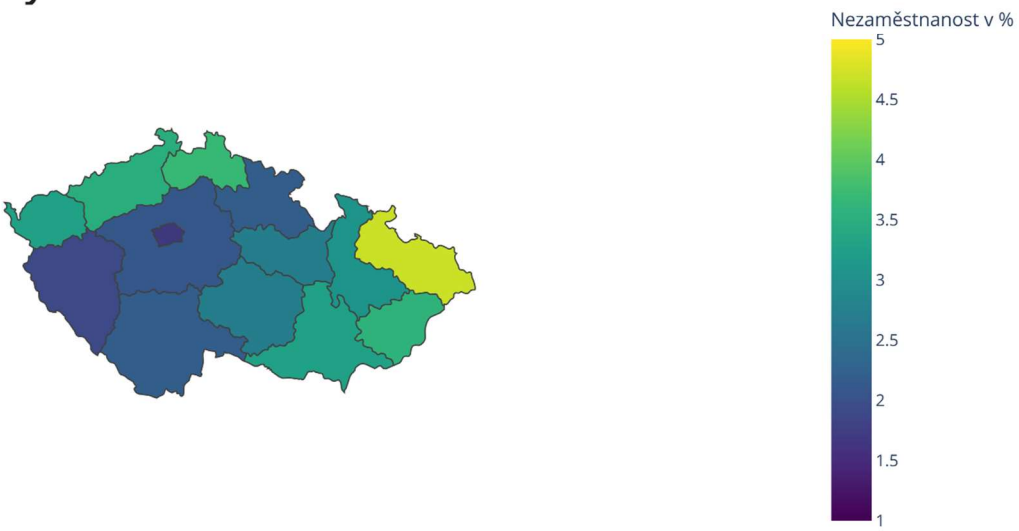
## Choropleth

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

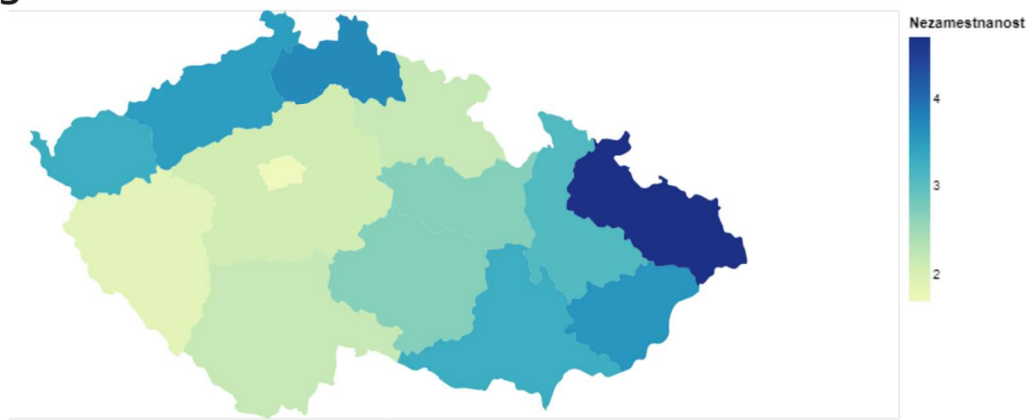
### Matplotlib




### Plotly




### Vega



# Síťový graf

Matěj Kolář 2022

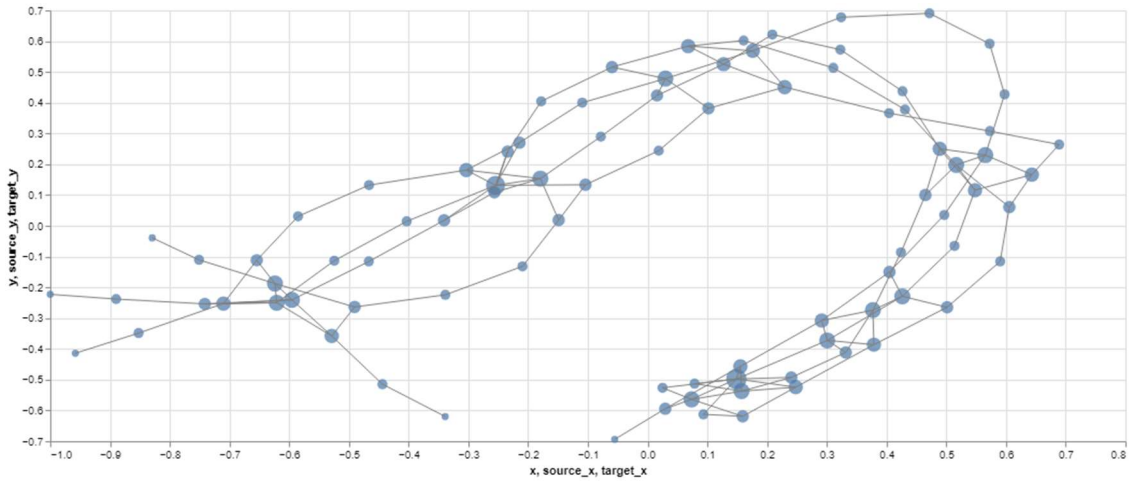
UHK - FIM

## Sdílený kód

```
import networkx as nx
import math

# načtení grafu
# zdroj: https://networkrepository.com/ENZYMES8.php
gr = nx.read_edgelist("../data/ENZYMES8.edges")
pos = nx.fruchterman_reingold_layout(gr)
# určení velikosti vrcholu podle počtu hran, rozdílné velikosti pro každý ekosystém
node_sizes_matplot_vega = [10*dict(gr.degree)[node] for node in gr.nodes]
node_sizes_plotly = [4+3*math.sqrt(dict(gr.degree)[node]) for node in gr.nodes]
```

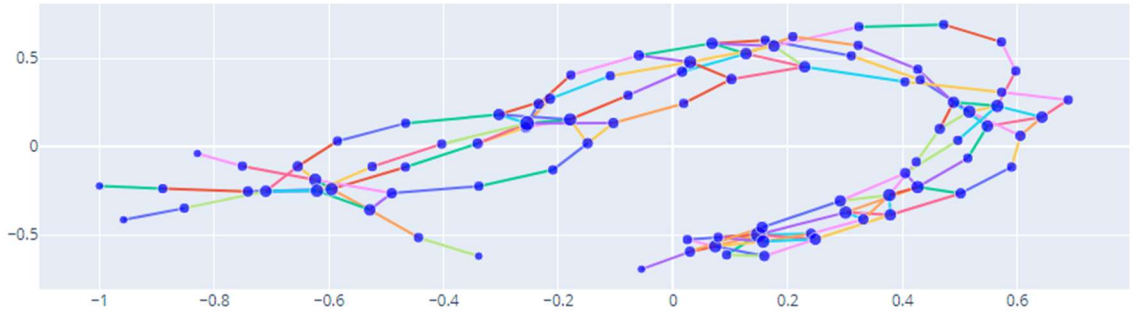
## Vega

```
import altair as alt
import pandas as pd

# zpracování dat do dataframes pro vegu
nodes = pd.DataFrame(pos.keys(), columns=["node"])
nodes["x"], nodes["y"] = zip(*pos.values())
nodes["size"] = node_sizes_matplot_vega
edges = pd.DataFrame(gr.edges, columns=["source", "target"])
edges = edges.merge(nodes, left_on="source", right_on="node", how="left").rename(
    columns={"x": "source_x", "y": "source_y"}).drop(columns=["node"])
edges = edges.merge(nodes, left_on="target", right_on="node", how="left").rename(
    columns={"x": "target_x", "y": "target_y"}).drop(columns=["node"])

# vykreslení grafu
edge_chart = alt.Chart(edges).mark_rule().encode(
    x="source_x", y="source_y", x2="target_x", y2="target_y", color=alt.value("gray"))
node_chart = alt.Chart(nodes).mark_circle().encode(x="x", y="y", size="size")
(node_chart+edge_chart).properties(width=1000,
                                   height=400).configure_legend(disable=True)
```



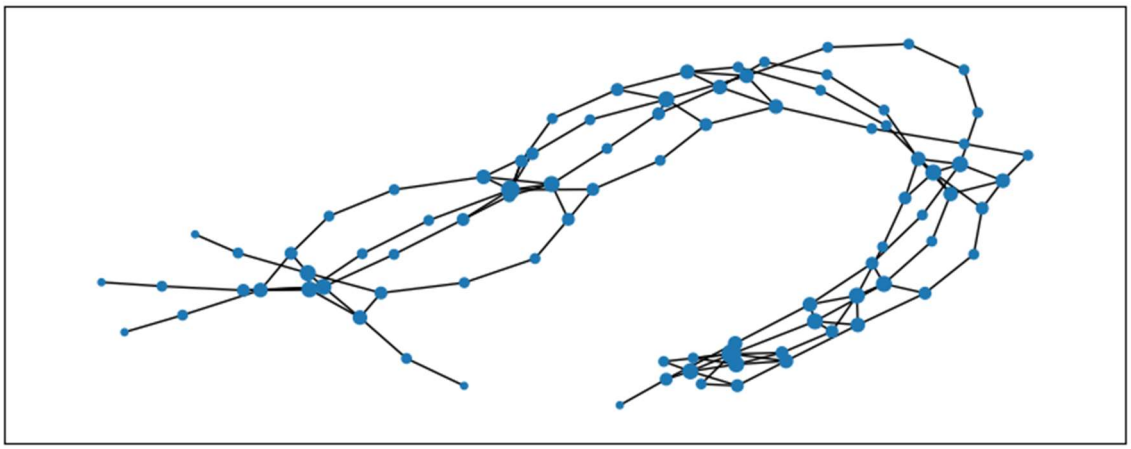
## Plotly

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

fig = go.Figure()
for edge in gr.edges:
    x0, y0 = pos[edge[0]]
    x1, y1 = pos[edge[1]]
    fig.add_trace(go.Scatter(x=[x0, x1], y=[y0, y1], mode="lines"))
x, y = zip(*pos.values())
fig.add_trace(go.Scatter(x=x, y=y, mode="markers",
              marker=dict(size=node_sizes_plotly, color="blue")))
fig.update_layout(width=1000, height=400, showlegend=False,)
fig.show()
```



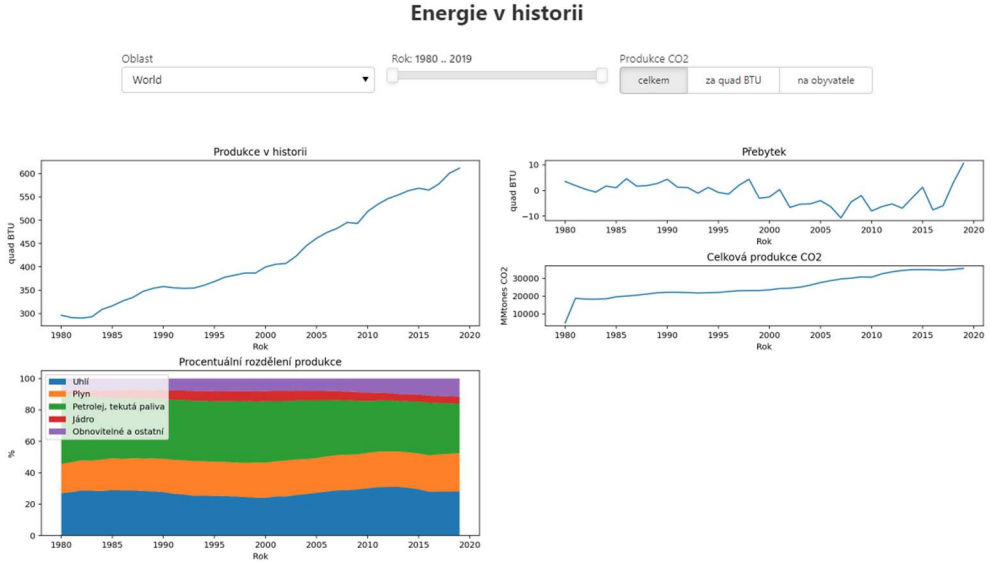

## Matplotlib

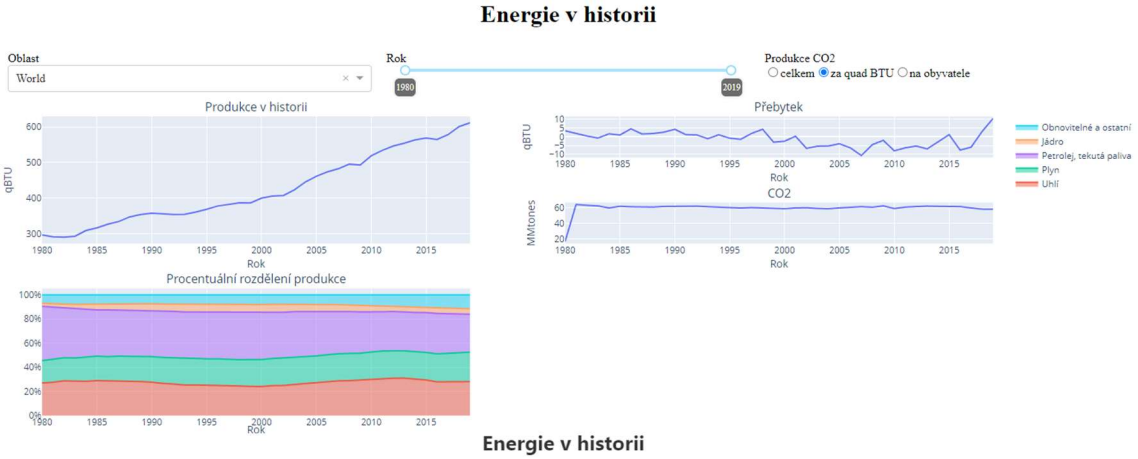
```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 4), dpi=100)
nx.draw_networkx_nodes(gr, pos, node_size=node_sizes_matplot_vega)
nx.draw_networkx_edges(gr, pos)
plt.show()
```

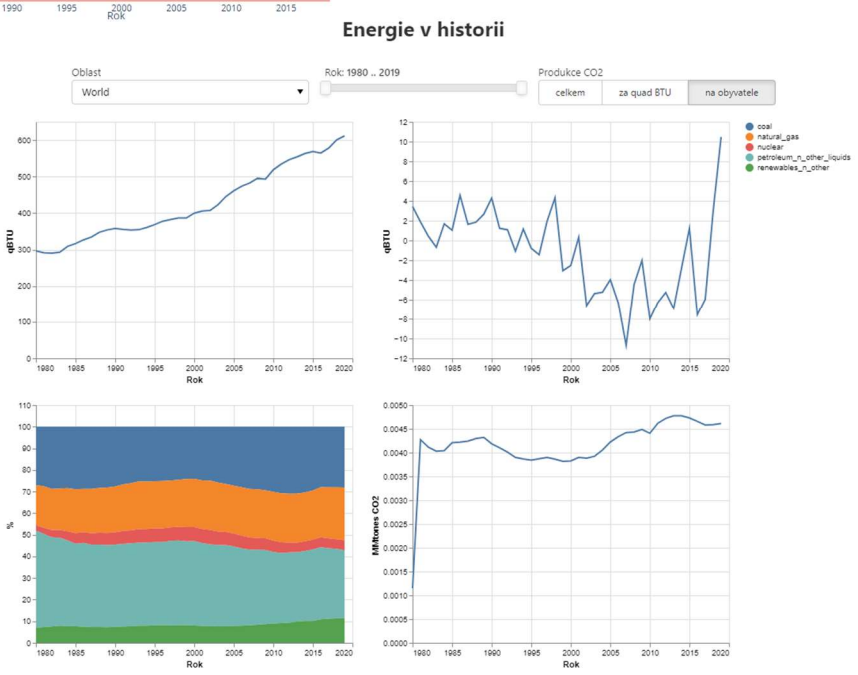


# Dashboard

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



# Design - Témata

Matěj Kolář 2023

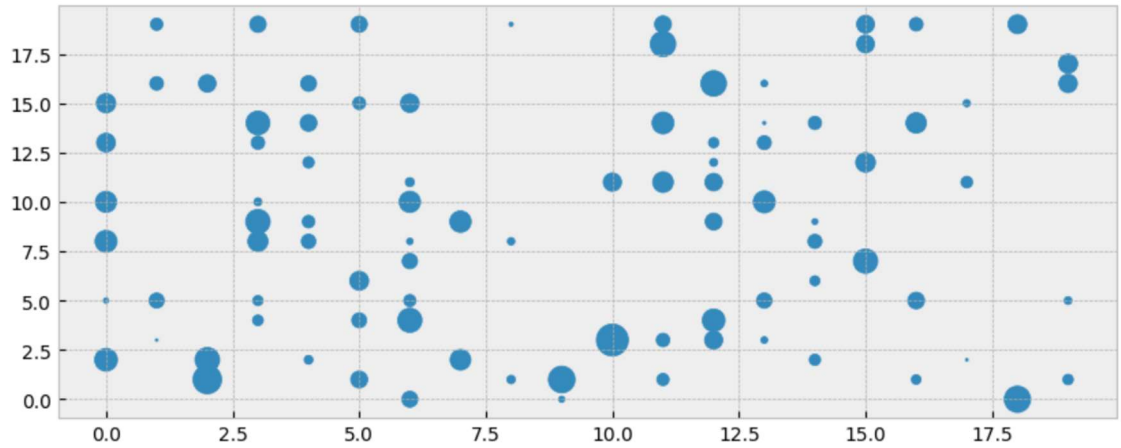
UHK - FIM

## 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"])
```

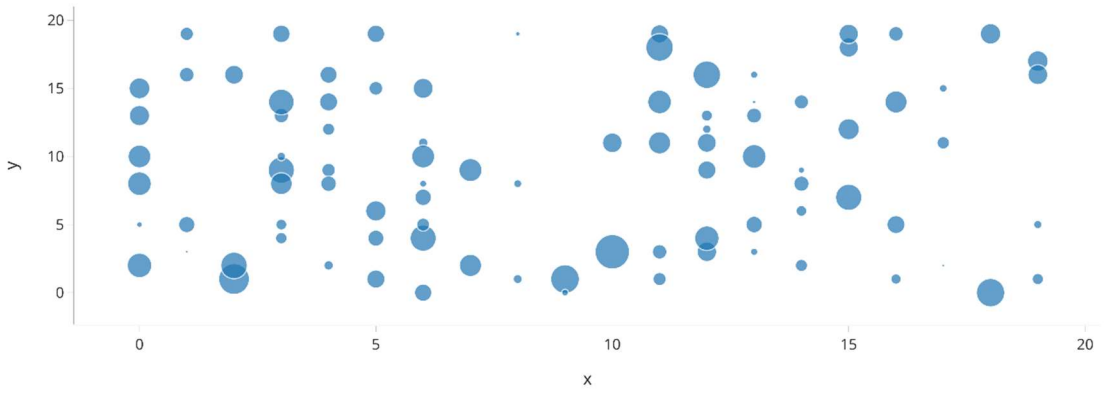
## Matplotlib

```
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


# 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 |