

Vizualizace dat

Vizualizace dat v prostředí Python pomocí knihovny Matplotlib

Zdeněk VAŠÍČEK

Fakulta informačních technologií, Vysoké učení technické v Brně
Brno, Czech Republic
vasicek@fit.vutbr.cz



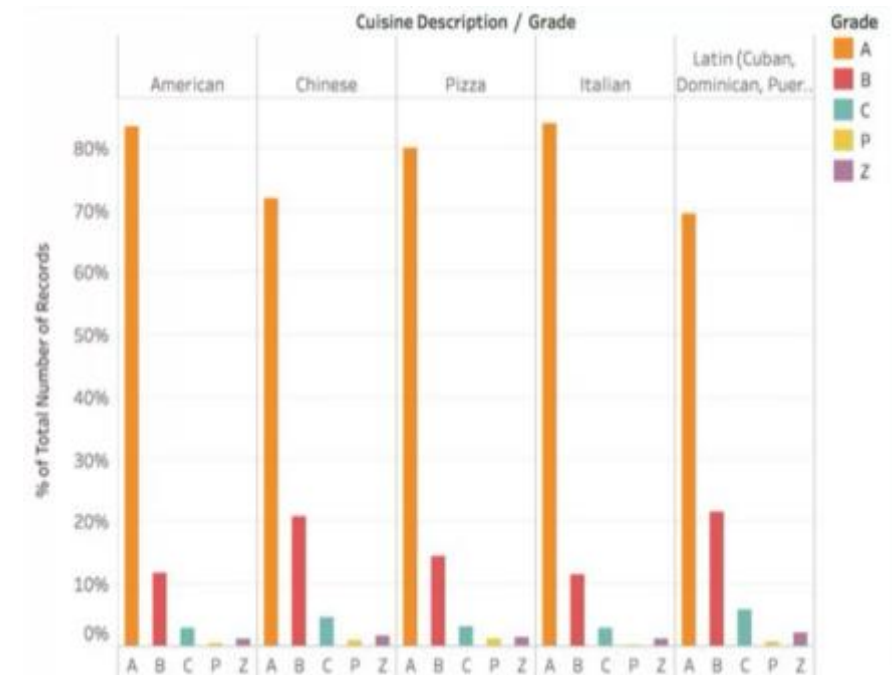
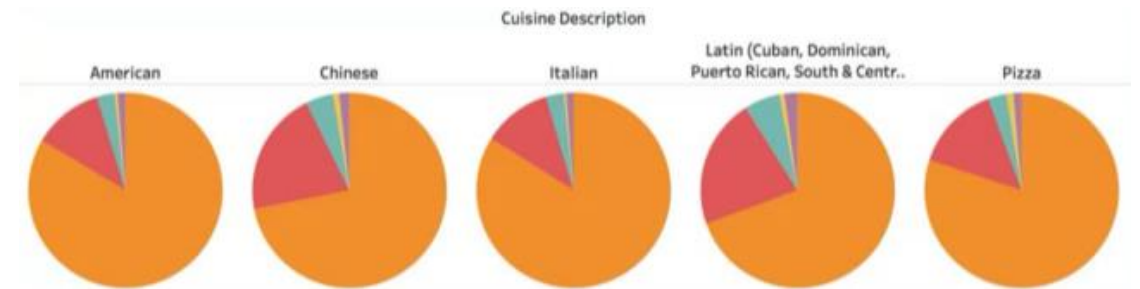
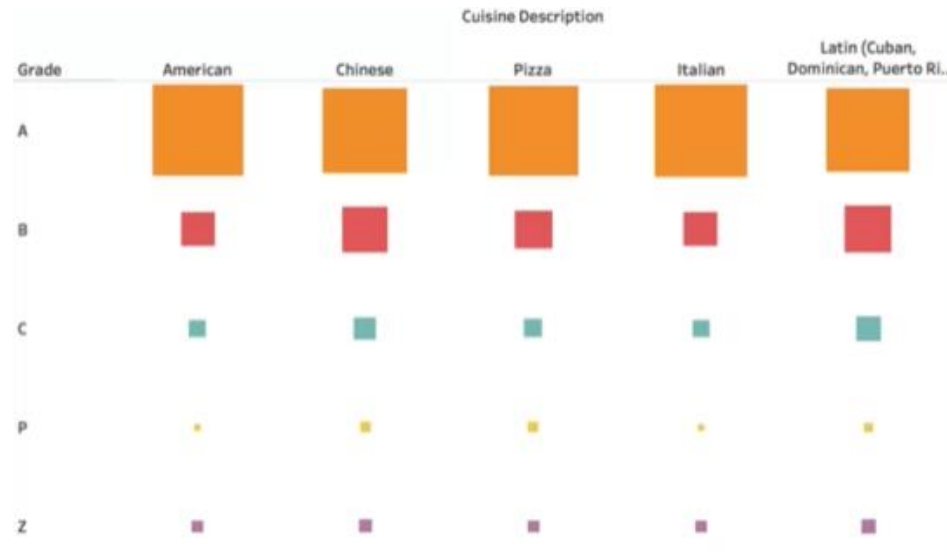
Obsah

- **Vizualizace dat**
 - proces vizualizace
 - principy visuálního vnímání, tvarová psychologie, estetika
- **Základní pravidla tvorby grafů**
 - co dělá graf grafem
 - graf jako prostředek manipulace
 - volba typu grafu
- **Tvorba grafů v prostředí Python pomocí Matplotlib**
 - generování výstupu
 - organizace grafu
 - konfigurace os
 - typy grafů
 - stylování

Vizualizace jako proces transformace dat

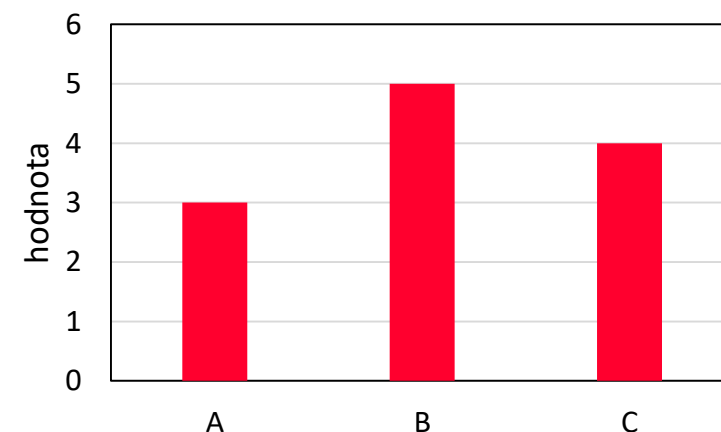
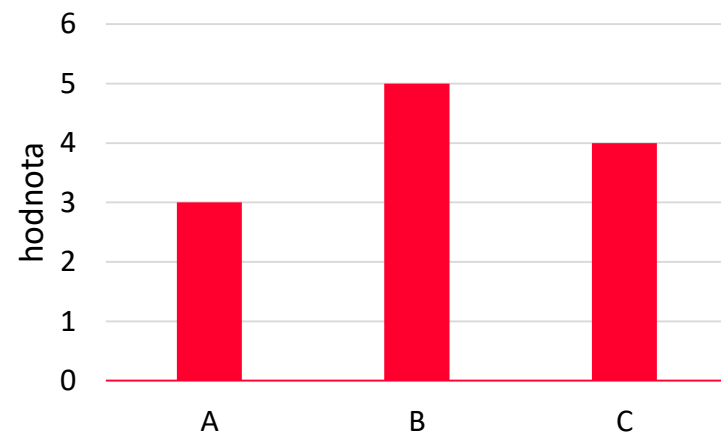
- Vizualizace je abstrahovaná forma informace v určité schematické podobě, která je nejen srozumitelná a matematicky korektní, ale také esteticky akceptovatelná
 - lidský mozek je schopen zpracovávat pouze omezené množství číselných údajů
 - 90% informace z prostředí vnímáme skrze zrak, 50% neuronů se účastní zpracování vizuální informace
 - správná forma prezentace umožní lépe porozumět vztahům v datech
- Vizualizace dat je iterativní proces
 - kvalitní a úspěšná vizualizace nezačíná vytvořením grafu
 - je nutné rozlišit mezi tvorbou grafu v průběhu **procesu porozumění** a tvorbou grafu jako **finálního produktu**
 - kroky: pořízení dat (acquire), zpracování dat (parse), čištění (filter), analýza (mine), volba formy vizualizace (represent), optimalizace (refine), finální produkt (interact)
- Vizualizace se mívá účinkem, není-li komunikována pro cílové publikum srozumitelně
 - podobně jako dobré literární dílo musí i dobrá vizualizace sdělit myšlenku jasně, přesně a efektivně
 - vizualizace by měla být promyšlená a správně organizovaná (viz [Gestalt Design Principles](#))

Forma vizualizace



Co dělá graf dobrým nebo špatným

- **Neatraktivní graf (ugly)**
 - graf, který je z estetického hlediska problémový ale jinak jasný a informativní
- **Matoucí graf (bad)**
 - graf, který má problémy týkající se chápání; může být nejasný, matoucí, komplikovaný nebo klamavý
- **Chybný graf (wrong)**
 - graf, který má problémy týkající se matematického pojetí; z objektivního hlediska nekorektní
- **Ideální graf**
 - takový graf, který je z estetického i matematického hlediska korektní a nepotřebuje další komentář k pochopení jeho sdělení.



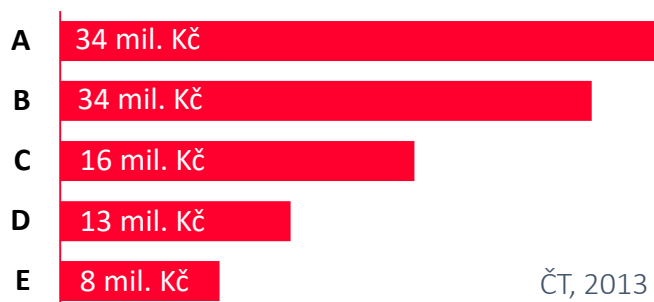
Chybné grafy

■ Reálné příklady

– Kombinace neslučitelných údajů

– Zatajené měřítko

PENÍZE ZA VOLBY



ČT, 2013

NA ČEM JSOU LIDÉ NEJVÍCE ZÁVISLÍ

V ROCE 2013 BRALO:



týdeník Dotyk, 2015

Problém interpretace, volba cílového publika

- Důležité je vždy pamatovat na to, pro jakou cílovou skupinu je graf určen a jaké je jeho sdělení.
- Příklad grafu, který otevírá mnoho otázek:
 - Teplota je nefyzikální veličina, která se mění skokově?
 - „Změna režimu v roce 1989 zapříčinila skokovou změnu průměrné teploty?“
 - Sdělení pochopí v tomto případě jen znalý problematiky, viz WMO, Resolution 4.1(4)/2 (Cg-17)



Český hydrometeorologický ústav

1 h · 🌐

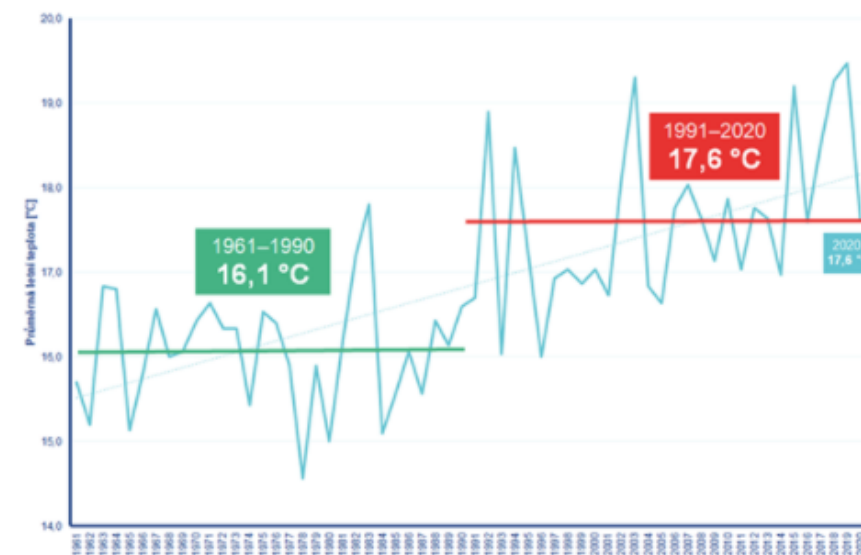
? "Léto je/bylo průměrné" - tuto větu jsme v minulých týdnech slyšeli často. A skutečnost? ✅ Bylo!

📊 Na grafu vidíte průměrnou letní teplotu v jednotlivých letech v období 1961–2020. Zeleně je zvýrazněn průměr letních teplot za první a červeně za druhé třicetiletí. 📌 Léto se u nás tedy v průměru oteplilo o 1,5 °C.

➡ Zajímavé je, že průměrná letní teplota v roce 2020 (17,6 °C) přesně odpovídá průměru posledního třicetiletí (1991–2020), léto tedy bylo teplotně průměrné, neboli normální.

📌 Ještě je zajímavé zmínit, že ve srovnání s průměrem 1961–1990 bychom letošní léto hodnotili jako teplotně nadnormální až silně nadnormální.

Průměrná letní teplota

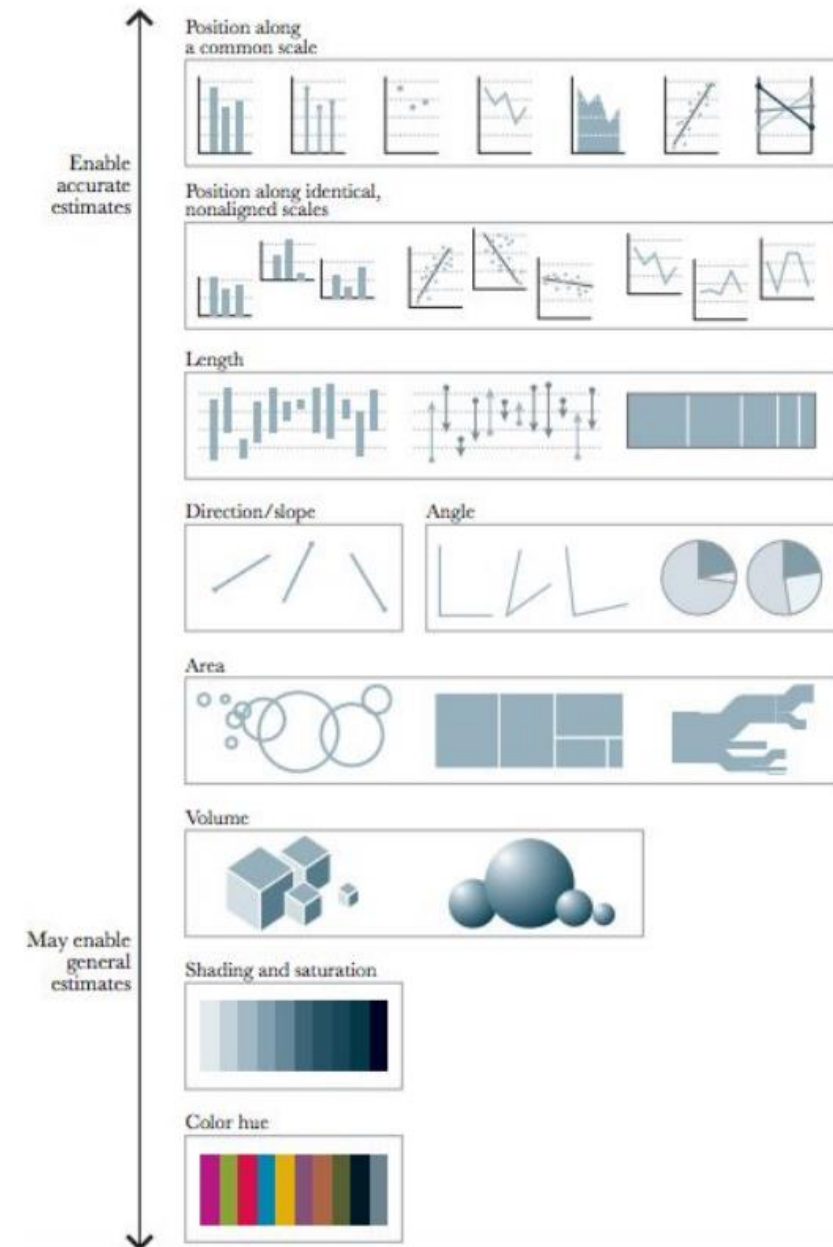


Zavádějící a zkreslující grafy

- Kromě uvedených typů existují ještě tzv. zavádějící grafy ([misleading graphs](#))
 - mohou být z matematického hlediska korektní, ale po vizuální stránce dochází ke zkreslení reality (eskalováno v případě 3D grafů)
- Typické prohřešky:
 - oříznutí osy X, oříznutí osy Y
 - neexistující nebo nevhodné měřítko osy (rozsah neodpovídající datům, nepřiznaná logaritmická osa, neuniformní intervaly, ...)
 - vynechání některých datových bodů
 - zneužití perspektivy 3D grafů
 - volba nevhodného typu grafu (např. koláčový graf pro nesouvisející hodnoty)
- Příklady z praxe:
 - <https://demagog.cz/diskuze/politici-a-jejich-grafy-vyrokdne>
 - <https://demagog.cz/diskuze/politici-a-stale-ty-grafy>
 - https://www.idnes.cz/technet/veda/manipulace-grafy-statistika.A151023_164547_veda_pka
 - <https://venngage.com/blog/misleading-graphs/>

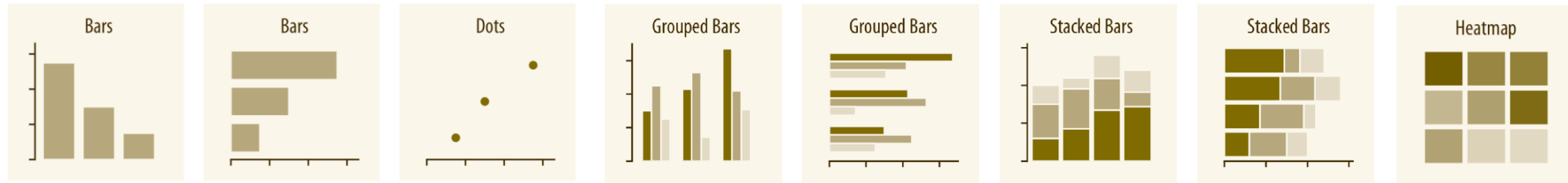
Vizualizace dat

- Typický postup procesu vizualizace dat pokud již máme zpracovaná a analyzovaná data
 - definovat cíl vizualizace a cílové publikum
 - zvolit vhodnou formu vizualizace v závislosti na požadovaném cíli
 - navrhnout vhodné grafické zpracování vizualizace (potřeba kreativity s ohledem na splnění prvního bodu)

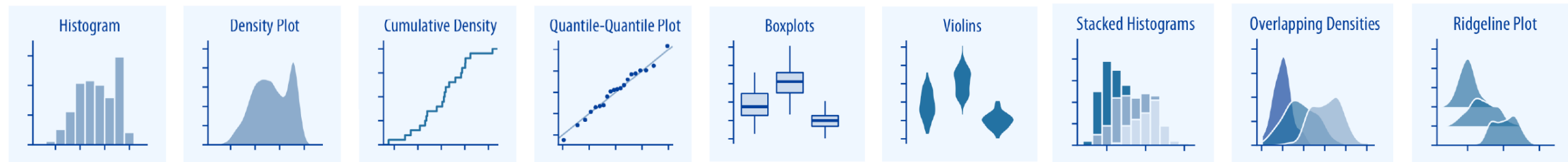


Volba typu grafu

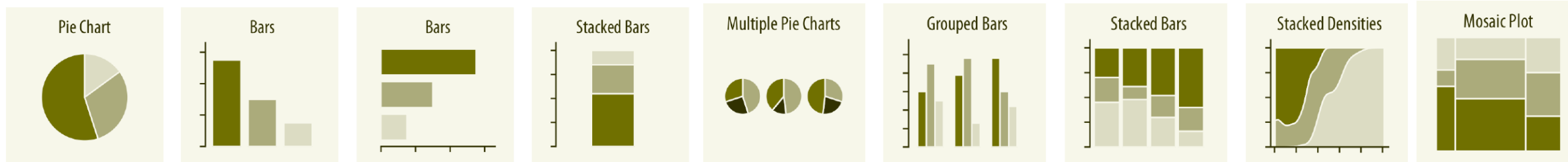
■ Množství



■ Rozložení

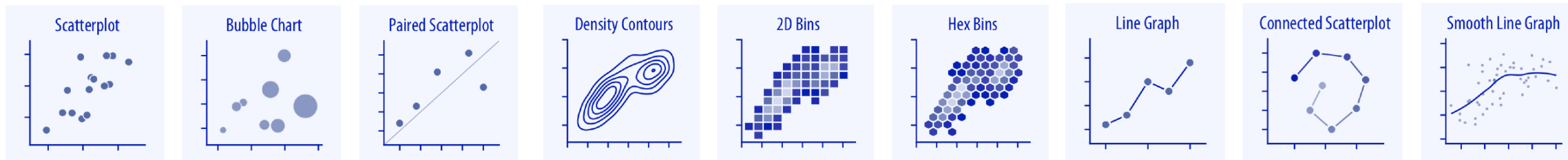


■ Vzájemné proporce, vyjádření součásti celku



Volba typu grafu

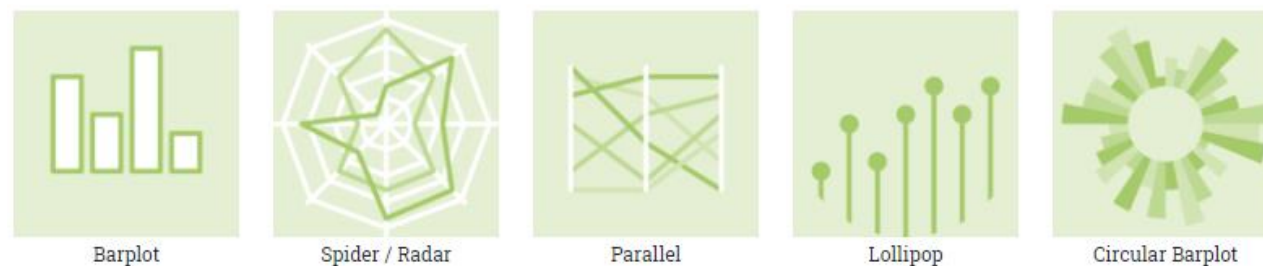
■ Vztah mezi prvky, korelace



■ Vyjádření nejistoty



■ Ranking



Barva a barevná paleta

■ Volba palety

- v závislosti na charakteru dat volíme typ palety
- maximálně 10 barev abychom byli schopni odlišit

■ Monochromatická sekvenční paleta (sequential)

- zobrazení numerických hodnot, které lze uspořádat od nejmenší po největší (typicky nejtmaší odstín).



■ Divergentní paleta (diverging)

- vhodné pro reprezentaci numerických hodnot, která lze kategorizovat a znázornit tak odchylku od průměru/mediánu/nuly, apod.
- tmavší odstín znamená větší odchylku v určitém směru



■ Kvalitativní paleta (qualitative)

- reprezentace dat, která lze kategorizovat a odlišit tak příslušnost k určité kategorii
- mohou být různého typu (viz Accent / Paired v [brewer](#))

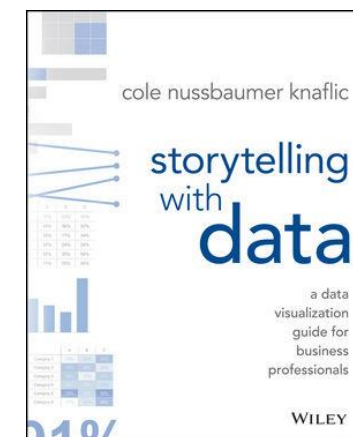
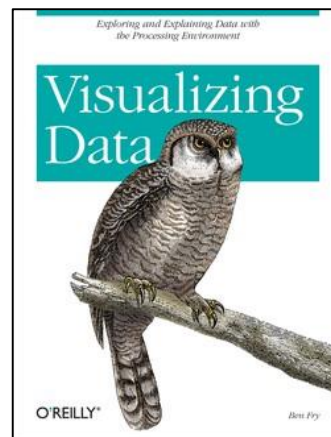
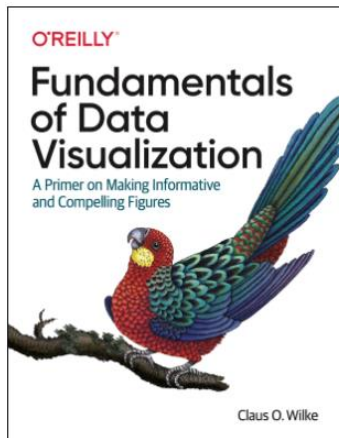


■ Palety viz:

- <https://jiffyclub.github.io/palettable/>
- https://seaborn.pydata.org/tutorial/color_palettes.html

Doporučená literatura

- Claus O. Wike: *Fundamentals of Data Visualization: A Primer on Making Informative and Compelling Figures*
 - viz clauswilke.com/dataviz
- Ben Fry: *Visualizing Data*
 - viz oreilly.com
- C.N. Knaflic: *Storytelling with Data: A Data Visualization Guide for Business Professionals*



Vizualizace dat v prostředí Python

■ Základní možnosti vizualizace dat

- **Matplotlib** (komplexní knihovna pro vizualizaci dat, komunitní projekt s 70k+ LoC s 15+ letou historií)
- **Pandas** (knihovna pro datovou analýzu, metoda `plot()` součást datových řad, 12+ let, viz [dokumentace](#))
- **Seaborn** (balík nástrojů pro statistické grafy pracující nad sloupci v pandas, viz [dokumentace](#))

■ Pandas a seaborn

- Pandas i seaborn jsou knihovny implementované nad Matplotlib a nabízí vysoko úrovněvé API, které umožňuje snadno získat vizuálně atraktivní a komplexní (zejména seaborn) grafy
- Pandas zvládne vizualizovat jen data, která se vejdou do paměti

■ Typický postup

- využít pandas + seaborn v prvotní fázi datové analýzy (viz např. [pairplot](#), [FacetGrid](#), v seaborn)
- využít seaborn + matplotlib pro požadovanou vizualizaci zpracovaných dat

IZV

Zpracování a vizualizace dat v prostředí Python

Matplotlib



Matplotlib (MPL)

■ Matplotlib je knihovna pro generování 2D a 3D grafů vědecké kvality

- tvorba mnoha typů grafů pomocí několika málo řádků kódu
- rozhraní kompatibilní s MATLAB™ dovoluje snadný přechod
- generuje kvalitní výstup do nejběžnějších rastrových (PNG, JPG) i vektorových (SVG, PS, PDF) formátů
- integruje interaktivní GUI pro základní inspekci dat
- využívá NumPy pro zvýšení efektivity některých operací
- nativní podpora v Jupyter Notebook, IPython

■ Architektura knihovny

- [Backend Layer](#) — nejkomplexnější vrstva mající na starost interakci s toolkity (Agg, Cairo, Gtk, Qt, Wx, ...) pro renderování grafického výstupu do souboru i formou interaktivního GUI. Základní prvky: FigureCanvas, Renderer, Event
- [Artist Layer](#) — vrstva obsahující všechny prvky, které se zobrazí na grafické plátno. Dovoluje plnou kontrolu nad grafickým výstupem. **Figure** je tzv. top-level container všech vizuálních prvků grafu.
- [Scripting Layer](#) (Frontend interface) – rozhraní, které typicky používá uživatel. Nabízí dva druhy rozhraní *procedurální stavové* rozhraní (**pyplot** API a dnes již nedoporučované **pylab** API) a *objektově orientované* rozhraní

Rozhraní knihovny Matplotlib

? Jak vytvořit instanci Figure bez pyplot?

■ Procedurální rozhraní

- stavové rozhraní ve stylu MATLAB™
- stav zachován mezi jednotlivými voláními funkcí modulu **matplotlib.pyplot**
- každá funkce způsobí nějakou změnu v grafu

■ Objektově orientované rozhraní

- využívá OO návrhu knihovny Matplotlib
- dovoluje vyšší míru kontroly nad výstupem
- prvním krokem je vytvoření instance **Figure** (přímo nebo pomocí procedurálního rozhraní)

```
import matplotlib.pyplot as plt
```

```
plt.figure(figsize=(6,4))  
plt.subplot(2,1,1)  
plt.plot(x1, y1)  
plt.plot(x2, y2)  
plt.subplot(2,1,2)  
plt.plot(x3, y3)  
plt.savefig('plot.png')
```

```
fig = plt.figure(figsize=(6,4))  
ax = fig.add_subplot(2,1,1)  
ax.plot(x1, y1)  
ax.plot(x2, y2)  
ax = fig.add_subplot(2,1,2)  
ax.plot(x3, y3)  
fig.savefig('plot.png')
```

```
plt.show()
```

```
plt.close(fig)
```

? Jaký je účel close()?

```
class matplotlib.figure.Figure(figsize=None, dpi=None, facecolor=None, edgecolor=None,  
                               linewidth=0.0, frameon=None, subplotpars=None, tight_layout=None, constrained_layout=None)
```



Rozhraní knihovny Matplotlib



- Běžná praxe a jeden z neduhů knihovny: svévolné míchání obou přístupů
 - naprosto nevhodný přístup, může vést na těžce laditelné chyby a problémy v přenositelnosti

```
import matplotlib.pyplot as plt
```

```
plt.figure(figsize=(6,4))  
ax = plt.subplot(2,1,1)  
ax.plot(x1, y1)  
plt.plot(x2, y2)  
ax = plt.subplot(2,1,2)  
plt.plot(x3, y3)  
plt.gca().set_xlabel('t')  
plt.savefig('plot.png')
```

```
plt.show()
```

Generování výstupu

■ Uložení grafu do souboru

- pomocí volání funkce `plt.savefig` (viz [doc](#)) nebo metody `Figure.savefig` (viz [doc](#)), nebo

```
...savefig(fname, dpi=None, facecolor='w', edgecolor='w',  
            orientation='portrait', papertype=None, format=None,  
            transparent=False, bbox_inches=None, pad_inches=0.1, frameon=None, metadata=None)
```

- metody `savefig` backendu PdfPages (např. `backend_pdf.PdfPages` viz [ukázka](#))

■ Parametr `fname` může být

- název souboru vč. přípony; podporované formáty viz `Figure.canvas.get_supported_filetypes()`

```
fig.savefig('plot.png')
```

- path-like nebo file-like object

```
plt.savefig(open('plot.svg', 'wb'), format='svgz', transparent=True)
```

■ Zobrazení grafu pomocí interaktivního GUI

```
plt.show()
```

- Pozn: v rámci Notebooku dochází k zobrazení implicitně, pozor na vynucený rámeček kolem Figure.



Nekonzistence v generování výstupu (savefig vs. show v VSC Notebooku)

■ Notebook vs Python skript

- v Notebooku dochází k zobrazení grafu implicitně, nikoliv při volání plt.show

■ V Notebooku je při zobrazení

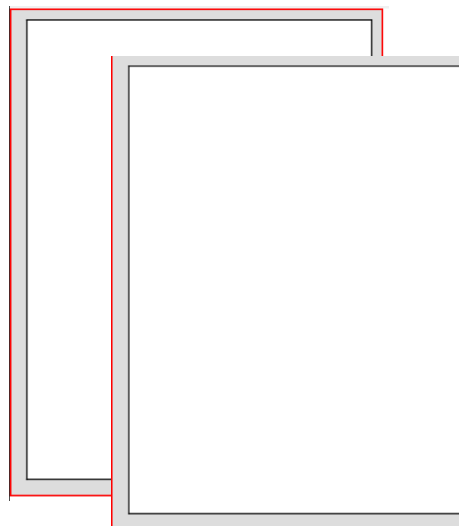
- vždy přidán okraj (margin u Figure) třebaže kresba zabírá celou plochu
- obraz oříznut na nejmenší plochu bez ohledu na rozložení

■ Příklad:

- výstup savefig



- výstup Notebook



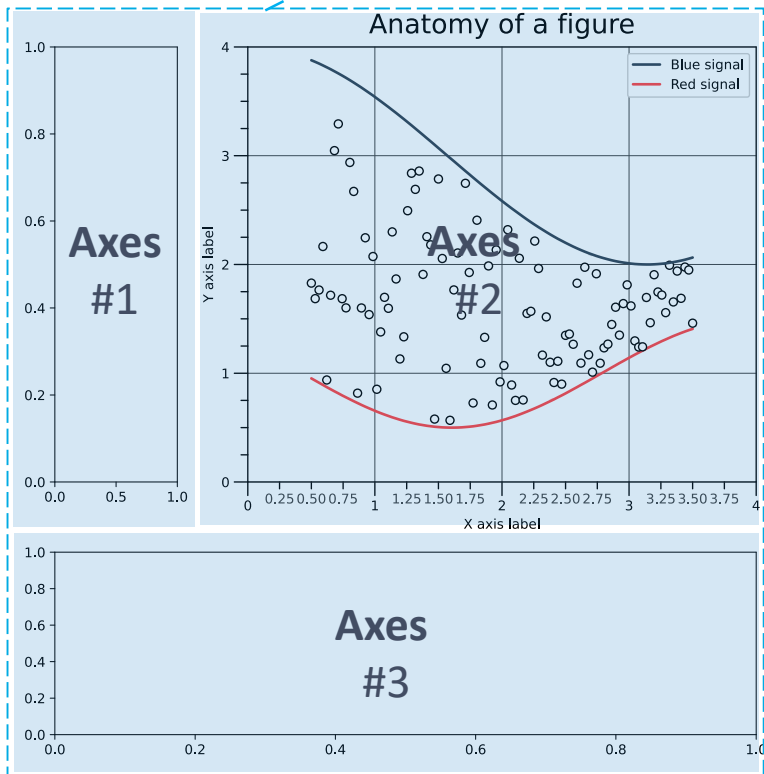
```
fig = plt.figure(figsize=(6,4),  
                 frameon=True, linewidth=2,  
                 facecolor='#dddddd',  
                 edgecolor='red')  
  
ax = fig.add_axes((0,0,0.5,1),  
                 frameon=True, facecolor='w')  
  
ax.xaxis.set_visible(False)  
ax.yaxis.set_visible(False)
```

Anatomie Matplotlib grafu

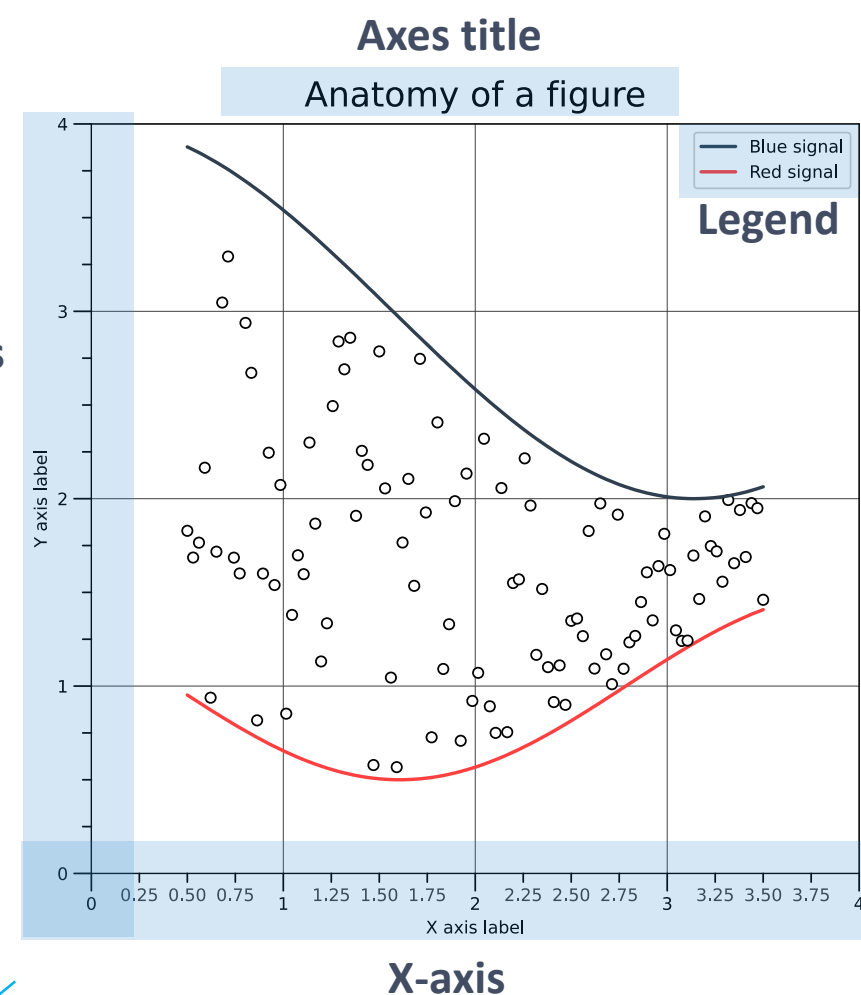
■ Reprezentace

- třída Figure – graf
- třída Axes – oblast do které lze kreslit (canvas)

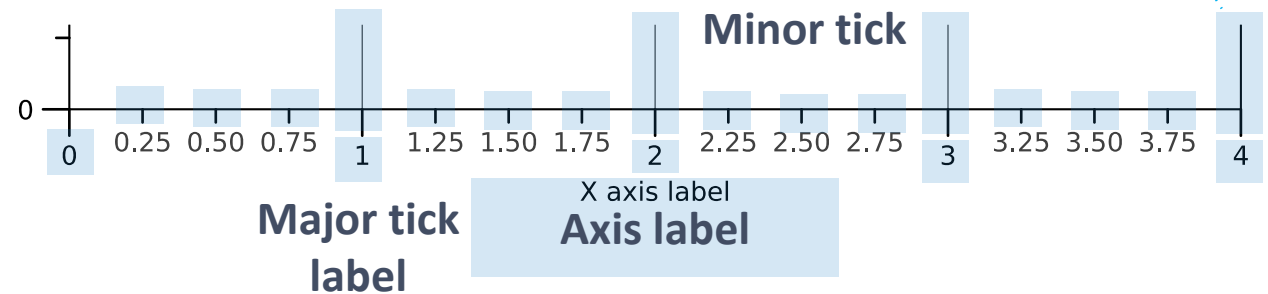
Figure



Y-axis



Major tick

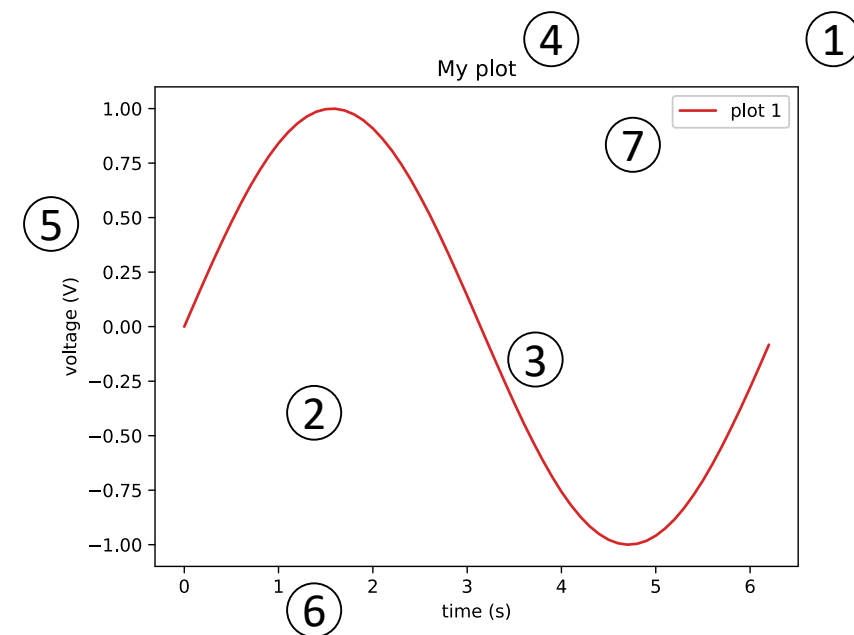


Přístup k prvkům Artist vrstvy z procedurálního rozhraní

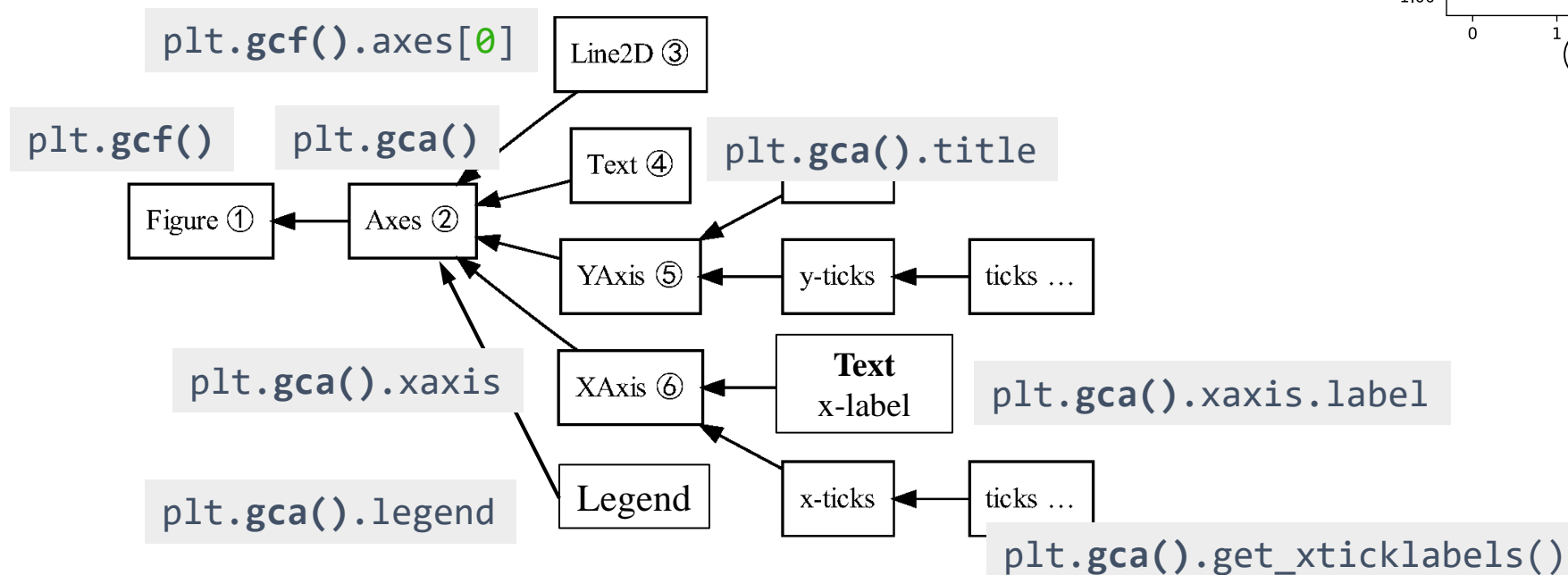
```
import matplotlib.pyplot as plt
import numpy as np
```

■ Příklad

```
x = np.arange(0, 2*np.pi, 0.1)
plt.plot(x, np.sin(x), 'C3', label='plot 1')
plt.xlabel('time (s)')
plt.ylabel('voltage (V)')
plt.title('My plot')
plt.legend()
plt.show()
```



■ Hierarchie instancí v Artist Layer



Organizace grafu (axes organization)

- Do grafu lze umísťovat podgrafy (axes) na fixní pozici pomocí
 - vícenásobného volání metody `Figure.add_axes` (viz [doc](#))

```
Figure.add_axes(rect, projection=None, polar=False, **kwargs)
```

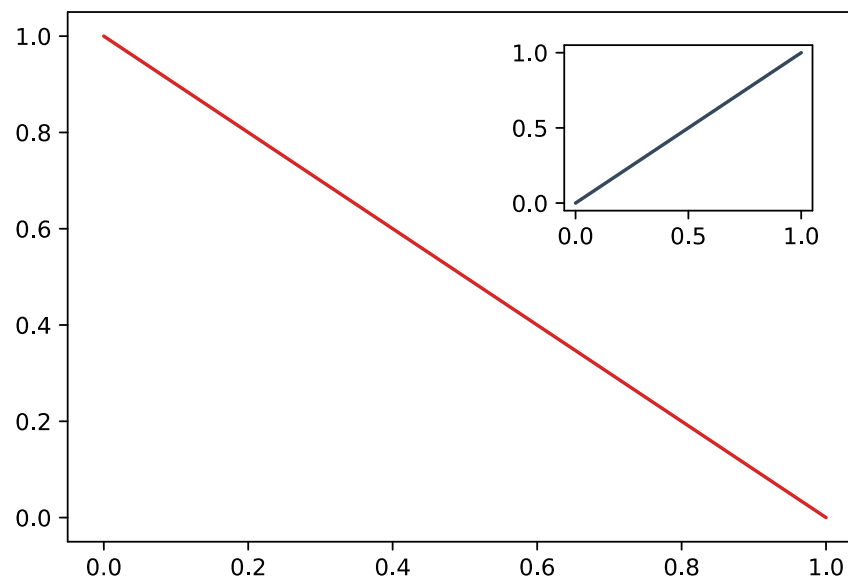
- Parametr *rect* je čtveřice (x ,y, w, h)
 - souřadnice levého spodního rohu + šířka a výška
 - hodnoty jsou v relativních souřadnicích
- Příklad:

```
fig = plt.figure(figsize=(6,4))

ax1 = fig.add_axes((0.1, 0.1, 0.8, 0.8))
ax2 = fig.add_axes((0.6, 0.6, 0.25, 0.25))

ax1.plot([0,1],[1,0],color='C3')
ax2.plot([0,1],[0,1],color='#35495e')

plt.show()
```



Organizace grafu (axes organization)

■ Podgrafy (axes) lze umísťovat automaticky do mřížky pomocí

- volání funkce `plt.subplots` (viz [doc](#)) nebo metody `Figure.subplots` (viz [doc](#))

```
matplotlib.pyplot.subplots(nrows=1, ncols=1, *, sharex=False, sharey=False,  
                           squeeze=True, subplot_kw=None, gridspec_kw=None, **fig_kw)
```

```
Figure.subplots(self, nrows=1, ncols=1, *, sharex=False, sharey=False,  
               squeeze=True, subplot_kw=None, gridspec_kw=None)
```

- vícenásobného volání funkce `plt.subplot` (viz [doc](#)) nebo metody `Figure.add_subplot` (viz [doc](#))

```
plt.subplot(self, *args, **kwargs)
```

```
Figure.add_subplot(self, *args, **kwargs)
```

■ V případě druhé varianty lze umístění podgrafu určit pomocí

- trojice (`nrows`, `ncols`, `index`)

```
Figure.add_subplot(self, nrows, ncols, index, **kwargs)
```

- pozice (`integer`)

```
Figure.add_subplot(self, pos, **kwargs) # pos = nrows*100 + ncols*10 + index
```

- objektu `SubplotSpec` (viz [GridSpec](#)), který dovoluje nepravidelnou mřížku / velikosti podgrafů

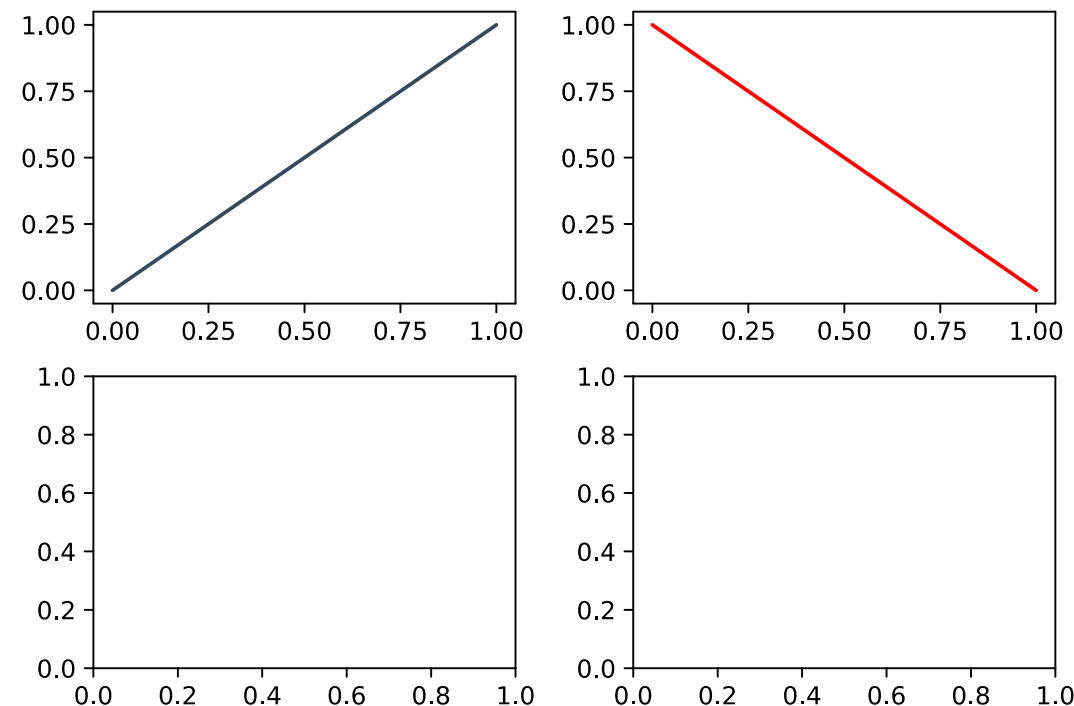
Organizace grafu (axes organization)

■ Příklad:

- umístění 4 stejně velkých podgrafů pomocí volání funkce `plt.subplots`

```
fig, axes = plt.subplots(ncols=2, nrows=2,  
                        constrained_layout=True,  
                        figsize=(6,4)  
                        )  
(ax1,ax2),(ax3,ax4) = axes  
  
ax1.plot([0,1],[0,1],color='#35495e')  
ax2.plot([0,1],[1,0],color='C3')  
  
plt.show()
```

- *constrained layout* zajistí automatické umístění všech prvků grafu tak, aby nedošlo k jejich překryvu a byla maximálně využita dostupná plocha; další varianta je *tight_layout*



Organizace grafu pomocí GridSpec mřížky

■ Příklad:

- umístění 4 stejně velkých podgrafů s využitím objektu `GridSpec`

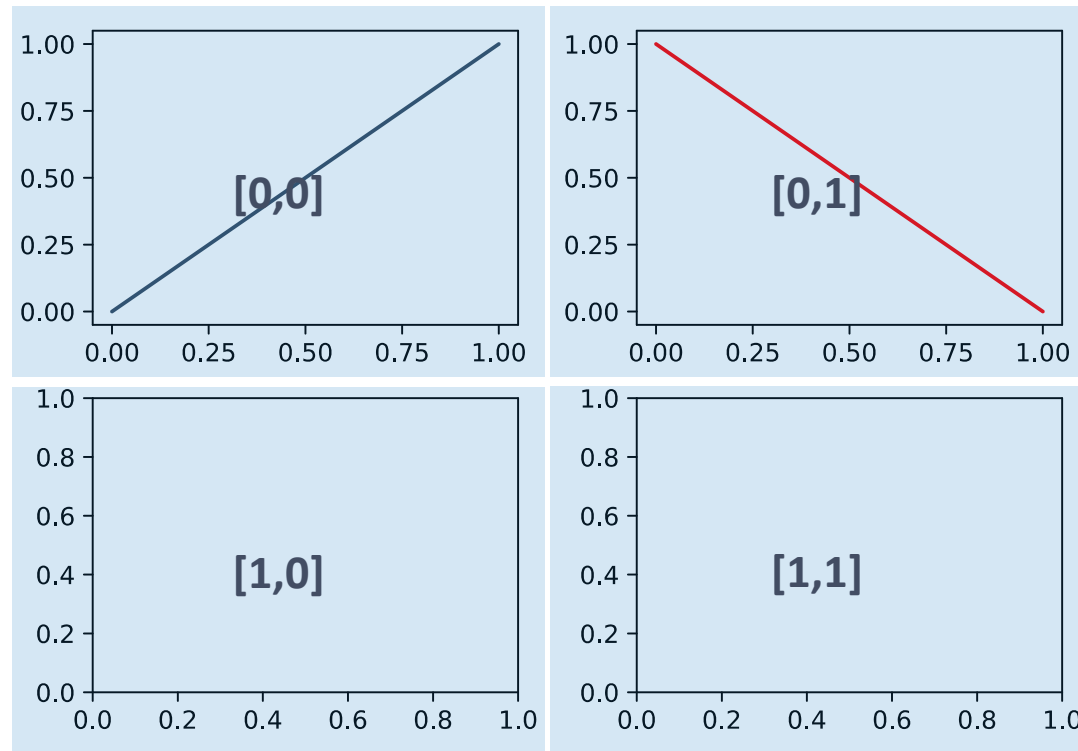
```
fig = plt.figure(constrained_layout=True,  
                 figsize=(6,4))
```

```
ax1, ax2, ax3, ax4 = (  
    fig.add_gridspec(ncols=2, nrows=2)  
    .subplots()  
)
```

```
ax1.plot([0,1],[0,1],color='#35495e')
```

```
ax2.plot([0,1],[1,0],color='C3')
```

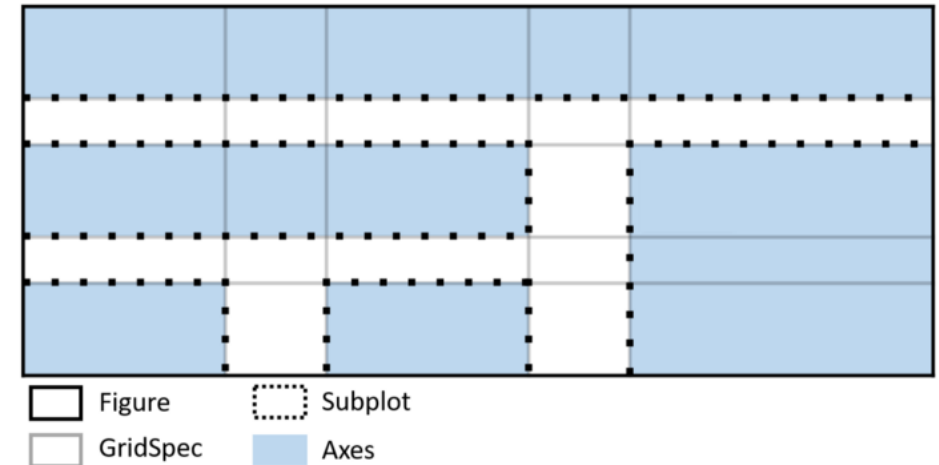
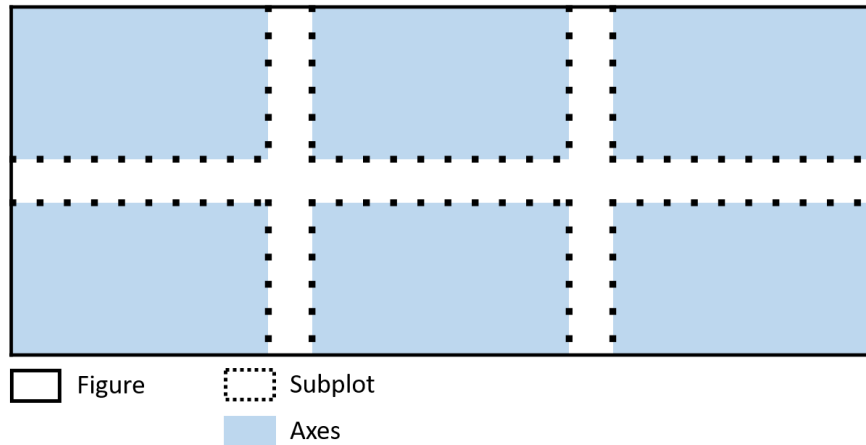
```
plt.show()
```



```
class matplotlib.gridspec.GridSpec(nrows, ncols, figure=None,  
                                   left=None, bottom=None, right=None, top=None,  
                                   wspace=None, hspace=None,  
                                   width_ratios=None, height_ratios=None)
```



Subplots vs. GridSpec

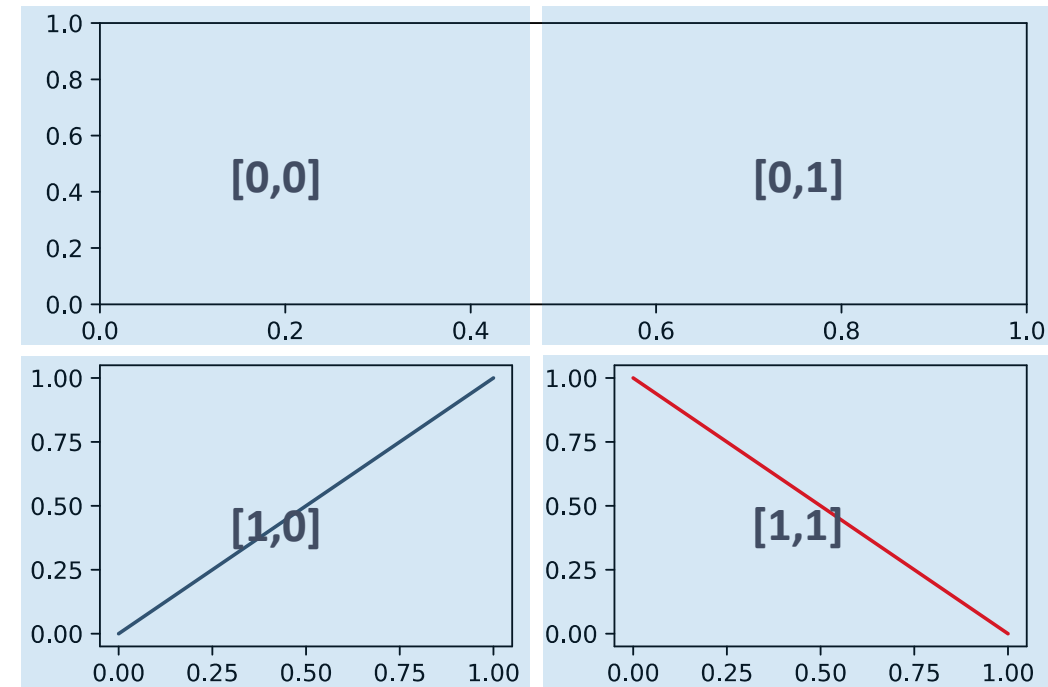


```
class matplotlib.gridspec.GridSpec(nrows, ncols, figure=None,  
                                   left=None, bottom=None, right=None, top=None,  
                                   wspace=None, hspace=None,  
                                   width_ratios=None, height_ratios=None)
```

Organizace grafu pomocí GridSpec mřížky

- Objekt **Gridspec** (viz [doc](#)) nabízí flexibilní způsob organizace podgrafů
 - dovoluje uživateli určit rozměry podgrafů a jejich umístění v rámci mřížky pomocí slice syntaxe NumPy
 - lze hierarchicky zanořovat (viz [ukázka](#)), buňky mřížky nemusí být stejné velikosti (viz parametry ratios)
- Příklad:
 - umístění 3 podgrafů různých velikostí

```
fig = plt.figure(constrained_layout=True,  
                 figsize=(6,4))  
  
spec = gridspec.GridSpec(ncols=2, nrows=2,  
                         figure=fig)  
  
ax1 = fig.add_subplot(spec[0, :])  
ax3 = fig.add_subplot(spec[1, 0])  
ax4 = fig.add_subplot(spec[1, 1])  
  
ax3.plot([0,1],[0,1],color='#35495e')  
ax4.plot([0,1],[1,0],color='C3')  
  
plt.show()
```



Konfigurace osy X a Y (xaxis and yaxis)

- V případě os máme možnost přizpůsobit (viz [API](#))
 - viditelnost os
 - rozsah hodnot a směr
 - měřítko (tj. transformaci)
 - obsah, umístění a styl titulku (label)
 - umístění a styl os (spines)
 - četnost a styl hlavní a vedlejší značky (ticker marker)
 - četnost, styl a obsah popisku hlavní a vedlejší značky (ticker label)
 - sdílení os – viz vedlejší osa (twin axis)

Základní konfigurace os

■ viditelnost

```
ax.set_axis_off()
```

```
ax.xaxis.set_visible(False)
```

■ rozsah hodnot a směr

- určuje nejmenší a největší viditelnou hodnotu

```
ax.set_xlim(left=None, right=None, emit=True, auto=False, *, xmin=None, xmax=None)
```

- směr růstu hodnot je možné změnit voláním

```
ax.invert_xaxis()
```

■ měřítko (scale)

- definuje rozmístění hodnot v rámci osy

```
ax.xscale(value, **scaleOpts)
```

- základní varianty implementované v rámci modulu **matplotlib.scale** (viz [doc](#)) jsou LinearScale, LogScale, SymmetricalLogScale, LogitScale (pro data mezi 0 a 1) a odpovídají value "linear", "log", "symlog", "logit",
- nový typ měřítka je možné přidat pomocí metody **matplotlib.scale.register_scale()**

■ obsah, umístění a styl titulku (label)

```
ax.set_xlabel(self, xlabel, fontdict=None, labelpad=None, *, loc=None, **TextProps)
```



Spines

Pro každou ze čtyř Spine čar lze individuálně přizpůsobit (viz [API](#))

■ viditelnost

```
ax.spines["top"].set_visible(False)
ax.spines["right"].set_visible(False)
```

■ rozsah

```
ax.spines["left"].set_bounds(-0.3, 1.1)
```

■ umístění

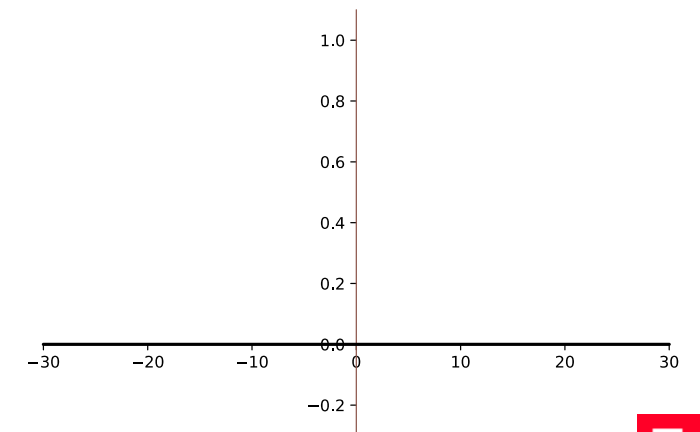
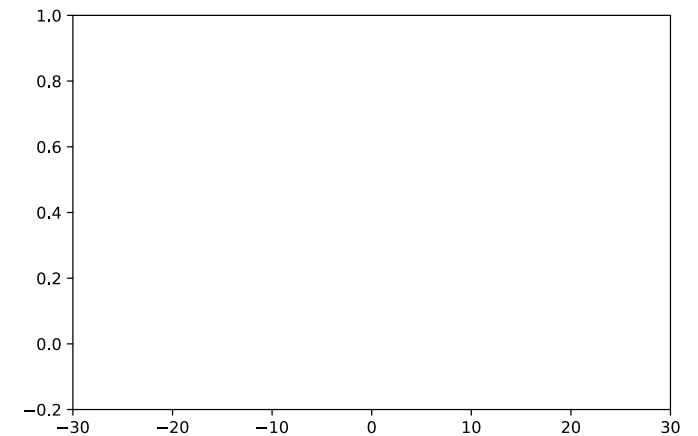
- definice pomocí dvojice (position type, amount), přičemž *position type* může být 'outward', 'axes' nebo 'data', výchozí ('outward', 0)
- zkratky: 'center' -> ('axes', 0.5), 'zero' -> ('data', 0.0)

```
ax.spines["bottom"].set_position('zero')
ax.spines["left"].set_position(("data", 0))
```

■ styl

- voláním `set_color`, `set_linestyle`, `set_linewidth`, `set_capstyle`

```
ax.spines["bottom"].set_linewidth(2)
ax.spines["left"].set_color("C5")
```



Hlavní a vedlejší značky (major and minor tickers)

- Styl značek lze konfigurovat pomocí metody `tick_params` (viz [doc](#))
 - selektivní nastavení parametrů týkajících se hlavních a vedlejších značek (a mřížky)

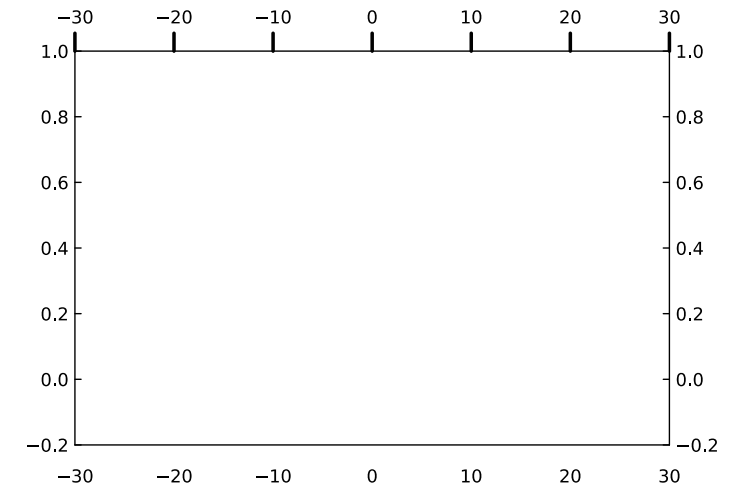
```
ax.tick_params(which='major', axis='both', **kwargs)
```

- konkrétní osy

```
ax.xaxis.set_tick_params(which='major', **kwargs)
```

U každého prvku lze přizpůsobit jeho

- viditelnost
 - parametry `bottom`, `top`, `left`, `right` a `labelbottom`, `labeltop`, `labelleft`, `labelright`
- styl značky
 - parametry pro nastavení stylu značky `direction`, `length`, `width`, `color`, `pad`
- styl popisku
 - parametry pro nastavení stylu popisku `labelsize`, `labelcolor`, `labelrotation`



```
ax.tick_params(axis='x', direction='out', width=2,
               length=10, bottom=False, top=True,
               labelbottom=True, labeltop=True)
ax.tick_params(axis='y', direction='in', left=True,
               right=True, labelleft=True, labelright=True)
```


Hlavní a vedlejší značky (major and minor tickers)

- Četnost značek na hlavní a vedlejší ose určuje tzv. Locator (viz [doc](#)), popisky přiřazené jednotlivým značkám určuje tzv. Formatter (viz [doc](#))

```
ax.xaxis.set_minor_locator(loc)  
ax.xaxis.set_minor_formatter(fmt)
```

```
ax.yaxis.set_major_locator(loc)  
ax.yaxis.set_major_formatter(fmt)
```

- Předdefinované třídy v matplotlib.ticker
 - **AutoLocator** – automatické generování hlavních značek (výchozí),
AutoMinorLocator – automatické generování vedlejších značek,
NullLocator – bez značek (výchozí pro minor),
MaxNLocator – automatické vygenerování maximálně N značek (základ AutoLocator),
LinearLocator – rovnoměrně rozmístěný pevně daný počet značek,
MultipleLocator – značky na násobcích báze,
LogLocator – značky rovnoměrně či nerovnoměrně rozmístěné na logaritmické ose,
FixedLocator – značky na pevně daných pozicích,
další **IndexLocator**, **SymmetricalLogLocator**, **LogitLocator**
 - **NullFormatter** – vrací prázdný řetězec, **FixedFormatter** – seznam řetězců definovaný pro jednotlivé značky,
FuncFormatter – formátování pomocí funkce, **FormatStrFormatter** – formátování pomocí sprintf,
StrMethodFormatter – formátování pomocí format, **PercentFormatter** – přidá procenta,
LogFormatter, **LogFormatterExponent**, **LogFormatterMathtext** – pro log. osu, a další

Hlavní a vedlejší značky (major and minor tickers)

```
import matplotlib.ticker as t
```

■ Příklady

- změna hlavních značek

```
fmt=t.FixedLocator([-20,-5,0,5,20])
ax.xaxis.set_major_locator(fmt)
```

- přidání vedlejších značek

```
ax.xaxis.set_minor_locator(t.AutoMinorLocator())
```

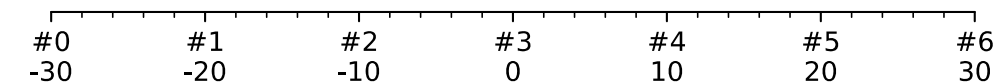
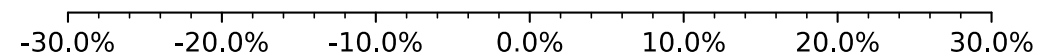
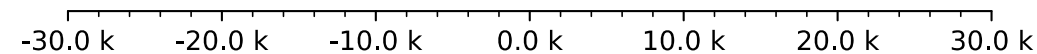
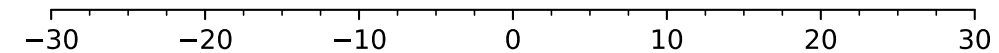
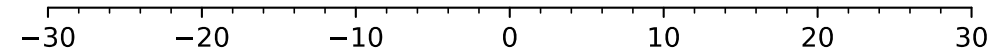
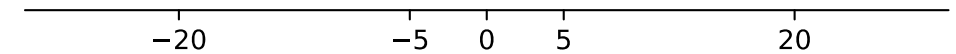
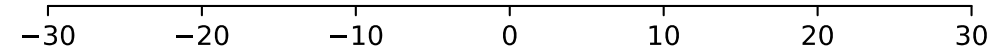
```
ax.xaxis.set_minor_locator(t.MultipleLocator(2.5))
```

- změna formátování popisku

```
fmt=t.FormatStrFormatter("%.1f k")
ax.xaxis.set_major_formatter(fmt)
```

```
fmt=t.PercentFormatter(xmax=100, decimals=1)
ax.xaxis.set_major_formatter(fmt)
```

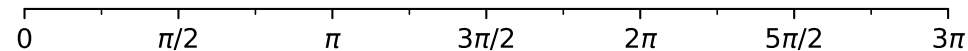
```
def myfmt(x, pos):
    return f'#{pos}\n{x:.0f}',
fmt=t.FuncFormatter(myfmt)
ax.xaxis.set_major_formatter(fmt)
```



Hlavní a vedlejší značky (major and minor tickers)

■ Příklad

- formátování vyžívající matematických výrazů (Mathtext)



```
def format_func(value, tick_number):  
    N = int(np.round(2 * value / np.pi))  
    if N >= 0 and N < 3:  
        return {0: "0", 1: r"$\pi/2$", 2: r"$\pi$"}[N]  
    elif N % 2 > 0:  
        return f"${N}\pi/2$"  
    else:  
        return f"${N // 2}\pi$"  
  
ax.set_xlim(0, 3*np.pi)  
ax.xaxis.set_major_locator(plt.MultipleLocator(np.pi / 2))  
ax.xaxis.set_minor_locator(plt.MultipleLocator(np.pi / 4))  
ax.xaxis.set_major_formatter(plt.FuncFormatter(format_func))
```

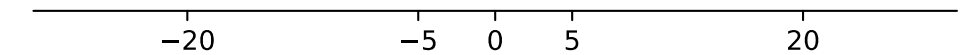
Hlavní a vedlejší značky (major and minor tickers)

- K variantě založené na Locator a Formatter existuje alternativna pro volbu konkrétních pozic značek a nastavení jejich popisku

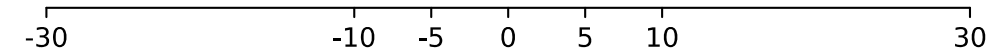
- Příklady

- volba konkrétních značek

```
ax.set_xticks([-20,-5,0,5,20])
```

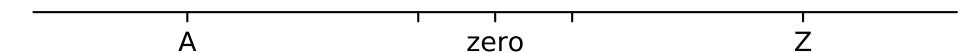


```
ax.xaxis.set_ticks([-30,-10,-5,0,5,10,30])
```

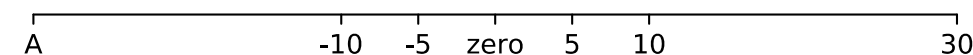


- nastavení popisku k existujícím značkám

```
ax.set_xticks([-20,-5,0,5,20])  
ax.set_xticklabels(['A',None,'zero',None,'Z','x'])
```



```
ax.xaxis.set_ticks([-30,-10,-5,0,5,10,30])  
ax.xaxis.set_ticklabels(['A',-10,-5,'zero',5,10,30])
```



Mřížka (grid)

■ Mřížku lze konfigurovat pomocí metody grid (viz [doc](#))

- selektivní nastavení vlastností 2D čar hlavní a vedlejší mřížky

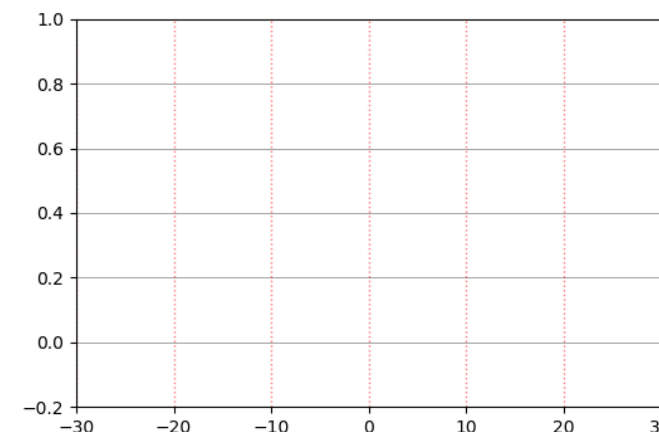
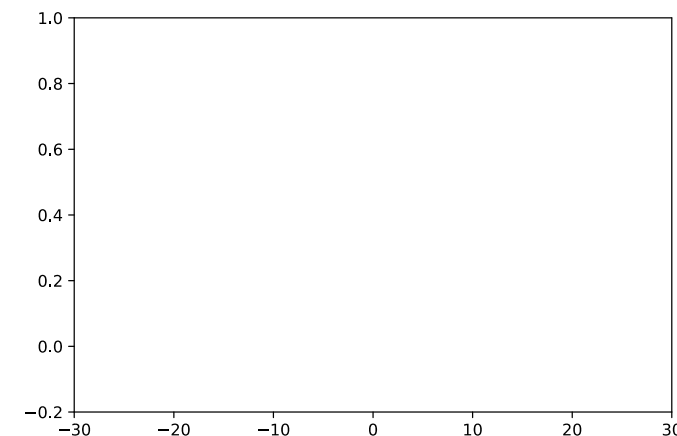
```
ax.grid(b=None, which='major', axis='both', **Line2dprops)
```

■ Parametry

- parametr **b** určuje viditelnost
- parametr **which** určuje konfiguruje-li hlavní / vedlejší mřížku
- parametr **axis** určuje konfiguruje-li horizontální / vertikální čáry
- **keywords** parametry viz vlastnosti 2D čáry [Line2Dprops](#)

■ Příklad

```
ax.grid(axis="x", color="red", linewidth=1, linestyle=":")  
ax.grid(axis="y", color="black", alpha=.5, linewidth=.5)
```



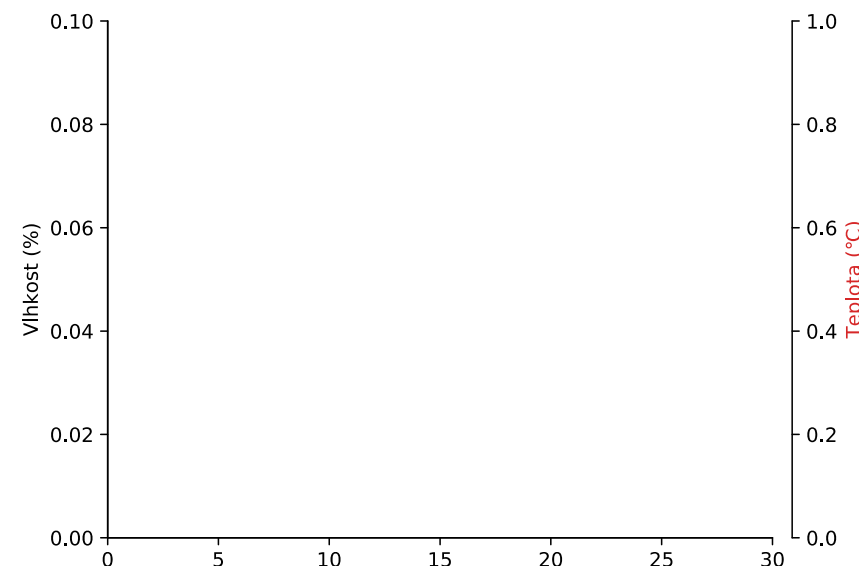
Vedlejší osa

- Metoda **ax.twinx()** a **ax.twiny()** umožňuje vytvořit nový podgraf (axes), který sdílí osu X, respektive osu Y, což umožňuje vytvoření vedlejší osy Y
 - pozor, duplikuje se celý objekt axes včetně všech prvků
- Příklad

```
fig = plt.figure(figsize=(6,4), constrained_layout=True)
ax1 = fig.add_subplot()
ax2 = ax1.twinx()

ax1.spines["top"].set_visible(False)
ax1.spines["right"].set_visible(False)
ax2.spines["top"].set_visible(False)
ax2.spines["right"].set_position(('outward',10))

ax1.set_xlim(0,30)
ax1.set_ylim(0,0.1)
ax2.set_ylim(0,1)
ax1.set_ylabel('Vlhkost (%)')
ax2.set_ylabel('Teplota (\u2103)', color='C3')
```



- Více vedlejších os viz [tutorial](#)

Legenda (legend)

■ Legendu lze konfigurovat pomocí metody legend (viz [doc](#) a [tutorial](#))

- prvky legendy jsou určeny automaticky z vykreslených dat

```
ax.legend()
```

- textová část legendy je uvedena explicitně

```
ax.legend(['label 1'])
```

- textová i grafická část legendy je uvedena explicitně

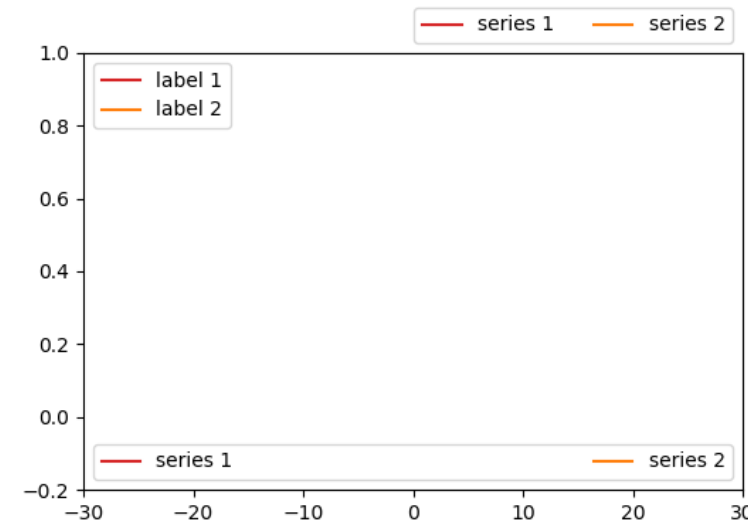
```
ax.legend((artistobj1, artistobj2, artistobj3), ('label1', 'label2', 'label3'))
```

■ Parametry

- umístění `loc`, kotva `bbox_to_anchor`, poč. sloupců `ncol` a mnoho dalších

■ Příklad

```
l= ax.legend(['label 1', 'label 2'], loc='upper left')
ax.add_artist(l)
l = ax.legend(['series 1', 'series 2'], loc='lower right',
              bbox_to_anchor=(1.0, 1.0), ncol=2)
ax.add_artist(l)
ax.legend(['series 1', 'series 2'], loc='lower left',
          bbox_to_anchor=(0, 0, 1, .1), ncol=2, mode="expand")
```



Podporované typy 2D grafů

■ Základní grafy

- Čárový graf (Line plot)
- Schodový graf (Step plot)
- Bodový graf (Scatter Plot)
- Vertikální a horizontální sloupcový graf (Bar chart) a Histogram (Histogram, 2D Hex bin)
- Tyčkový graf (Stem plots)
- Koláčový graf (Pie)

■ Statistické sumarizační grafy

- Krabicový graf (Box plot)
- Violin plot

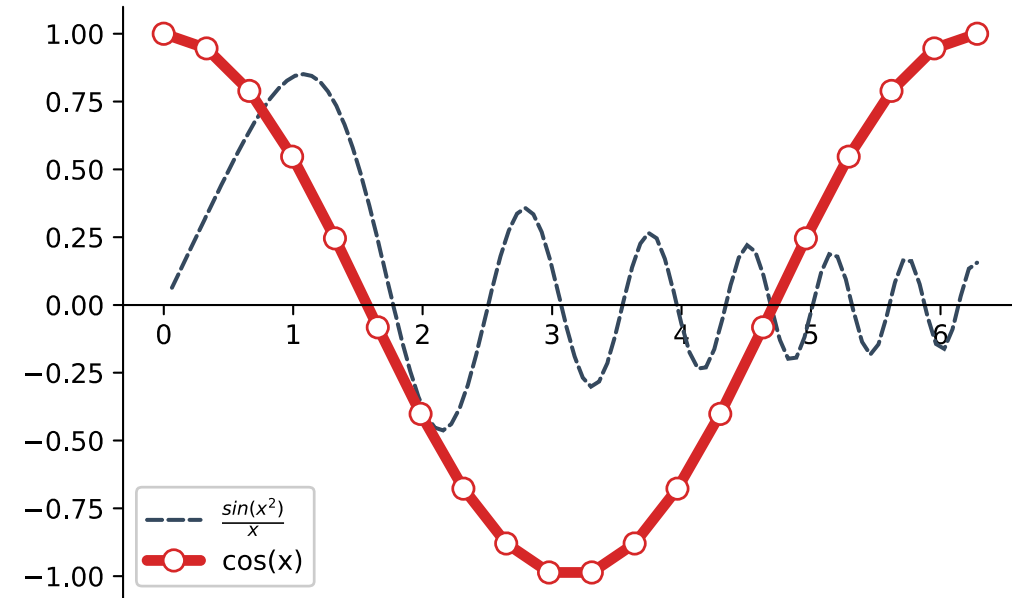
■ Další

- Vrstevnice (Contour plots)
- Vektorové pole (Stream plots)
- Grafy s tabulkou obsahující data (table)
- Kompletní přehled viz [doc](#)

Čárový graf (Line Plot)

■ Příklad (plot viz [doc](#))

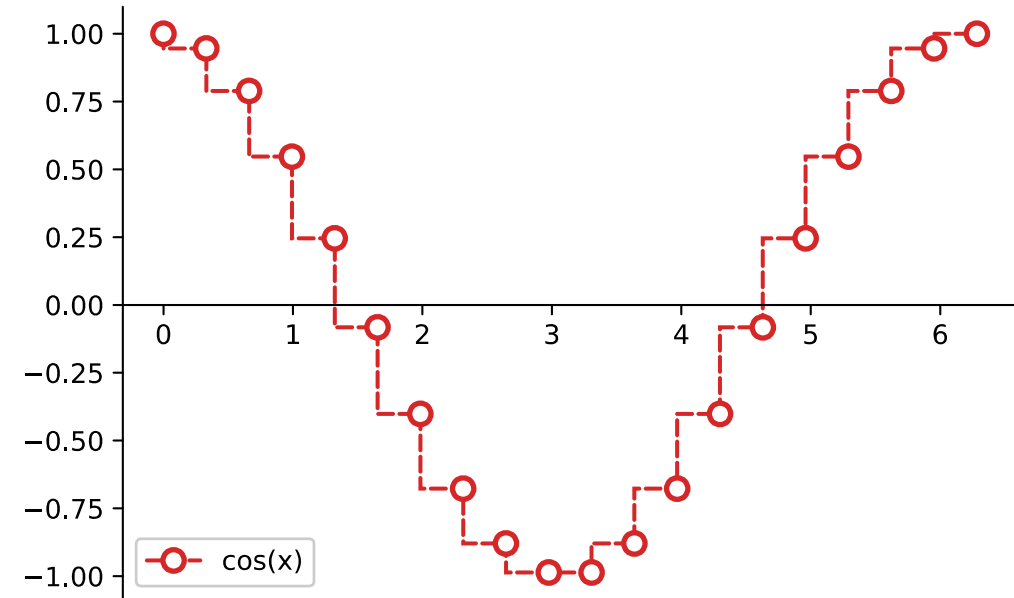
```
x1,x2=np.linspace(0,2*np.pi,100),  
        np.linspace(0,2*np.pi,20)  
y1,y2=np.sin(x1**2)/x1,np.cos(x2)  
  
fig = plt.figure(figsize=(6,4))  
ax = fig.add_subplot()  
  
ax.plot(x1, y1, c='#35495e', ls='--',  
        label=r'$\frac{\sin(x^2)}{x}$')  
  
ax.plot(x2, y2, 'C3o-',  
        lw=4, ms=8, mfc='w', label='cos(x)')  
  
ax.spines['top'].set_visible(False)  
ax.spines['right'].set_visible(False)  
ax.spines['bottom'].set_position('zero')  
ax.legend()  
  
plt.show()
```



Schodový graf (Step plot)

■ Příklad

```
x1,x2=np.linspace(0,2*np.pi,100),  
      np.linspace(0,2*np.pi,20)  
y1,y2=np.sin(x1**2)/x1,np.cos(x2)  
  
fig = plt.figure(figsize=(6,4))  
ax = fig.add_subplot()  
  
ax.step(x2, y2, 'C3o-', ls='--',  
       ms=8, mew=2, mfc='w',  
       where='pre', #pre/post/mid  
       label='cos(x)')  
  
ax.spines['top'].set_visible(False)  
ax.spines['right'].set_visible(False)  
ax.spines['bottom'].set_position('zero')  
ax.legend()  
plt.show()
```



Pozn. **ax.plot** má kwararg `drawstyle / ds`: {'default', 'steps', 'steps-pre', 'steps-mid', 'steps-post'},
výchozí hodnota je 'default'

Tyčkový graf (Stem plots)

■ Příklad

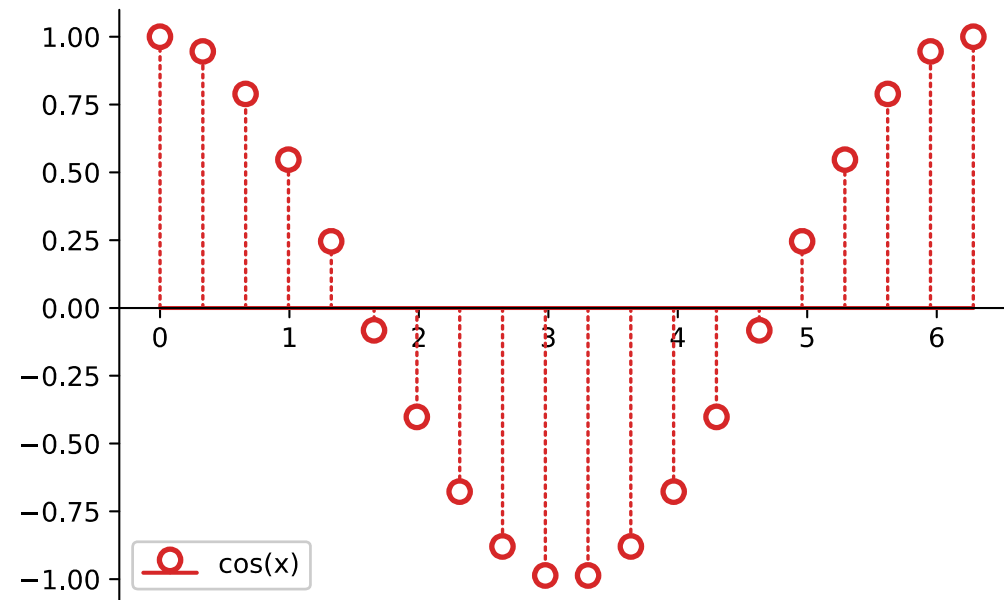
```
x1,x2=np.linspace(0,2*np.pi,100),  
        np.linspace(0,2*np.pi,20)  
y1,y2=np.sin(x1**2)/x1,np.cos(x2)
```

```
fig = plt.figure(figsize=(6,4))  
ax = fig.add_subplot()
```

```
markerline, stemline, baseline, =  
ax.stem(x2, y2,  
        linefmt='C3:',  
        markerfmt='oC3',  
        label='cos(x)')
```

```
plt.setp(stemline, linewidth = 1.25)  
plt.setp(markerline, mew=2 , ms = 8, mfc='w')
```

```
ax.spines['top'].set_visible(False)  
ax.spines['right'].set_visible(False)  
ax.spines['bottom'].set_position('zero')  
ax.legend()  
plt.show()
```



Sloupcový graf (Bar charts)

■ Příklad

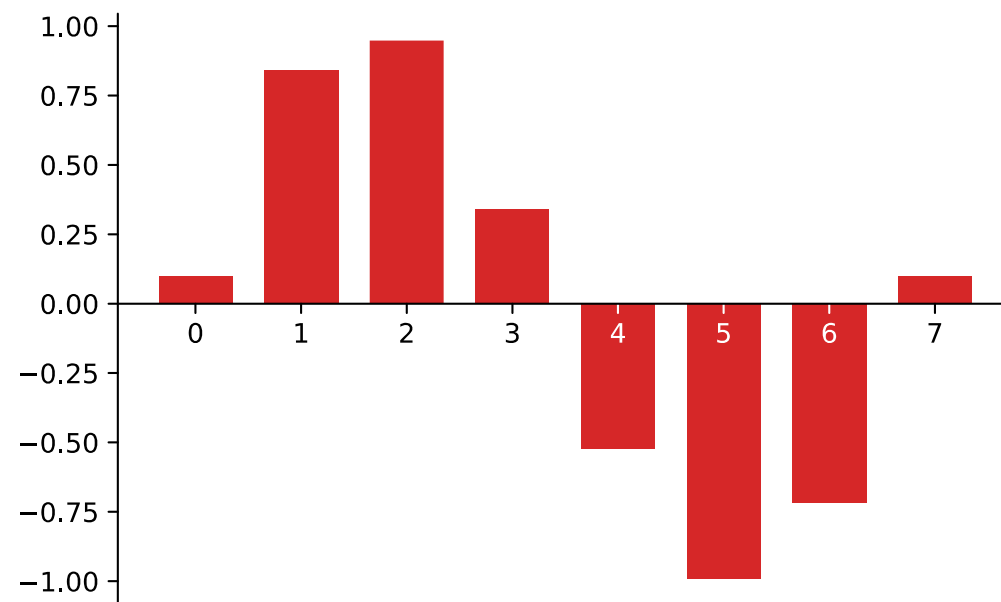
```
x1=np.linspace(0,7,8)
y1=np.sin(np.linspace(0,2*np.pi,8))+0.1

fig = plt.figure(figsize=(6,4))
ax = fig.add_subplot()

ax.bar(x1, y1,
       width=0.7, bottom=0, align='center',
       color='C3')

ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
ax.spines['bottom'].set_position('zero')
ax.margins(0.05)

plt.setp(ax.get_xticklabels()[5:8],
         color="white")
plt.setp(ax.get_xticklines()[10:15],
         markeredgecolor="white")
plt.show()
```



Stacked Bar chart

■ Příklad

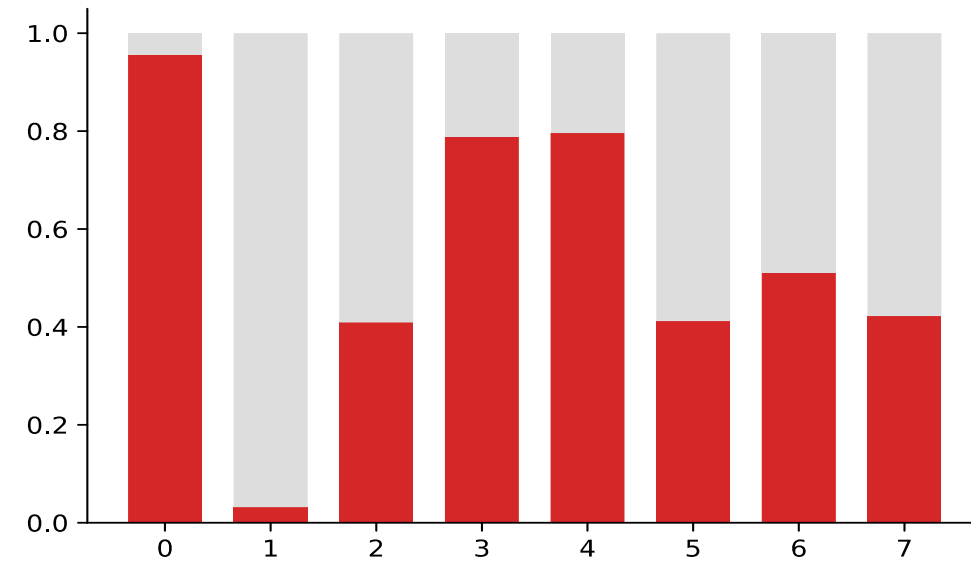
```
x1=range(0,8)
y1=np.random.uniform(0,1, 8)
y2=(1-y1)

fig = plt.figure(figsize=(6,4))
ax = fig.add_subplot()

ax.bar(x1, y1, width=0.7, align='center',
       bottom=0, color='C3')
ax.bar(x1, y2, width=0.7, align='center',
       bottom=y1, color='#dddddd')

ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
ax.spines['bottom'].set_position('zero')

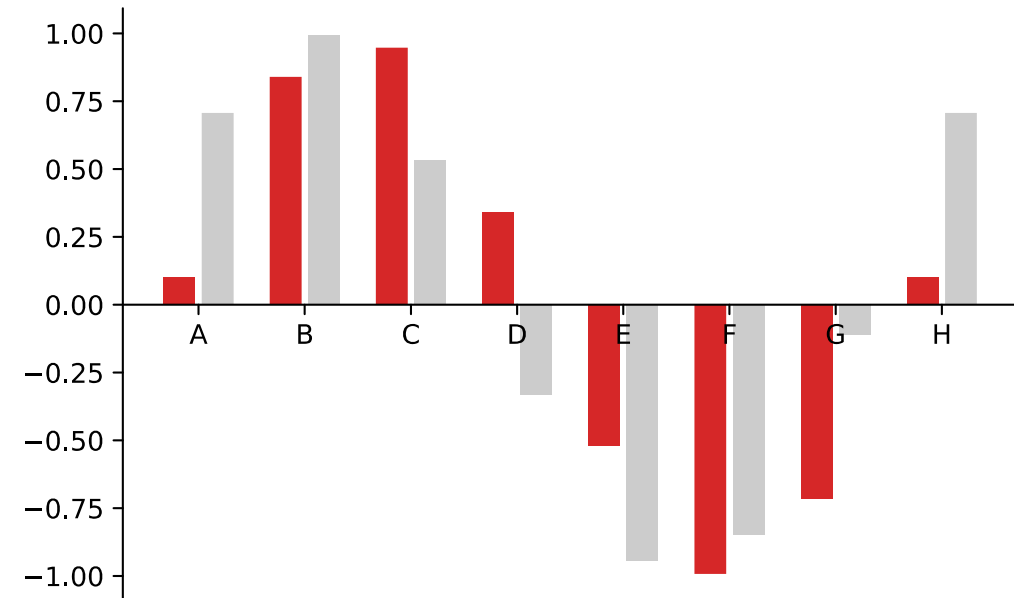
plt.show()
```



Sloupcový graf (Bar charts)

■ Příklad

```
x1,x=np.linspace(0,7,8),  
      np.linspace(0,2*np.pi,8)  
y1,y2=np.sin(x+0.1),np.sin(x+np.pi/4)  
  
fig = plt.figure(figsize=(6,4))  
ax = fig.add_subplot()  
  
ax.bar(x1-0.18, y1, width=0.3, bottom=0,  
       align='center', color='C3')  
ax.bar(x1+0.18, y2, width=0.3, bottom=0,  
       align='center', color='#cccccc')  
  
ax.spines['top'].set_visible(False)  
ax.spines['right'].set_visible(False)  
ax.spines['bottom'].set_position('zero')  
ax.margins(0.05)  
ax.set_xticklabels(' ABCDEFGH')  
  
plt.show()
```



Bodový graf (Scatter plot)

■ Příklad

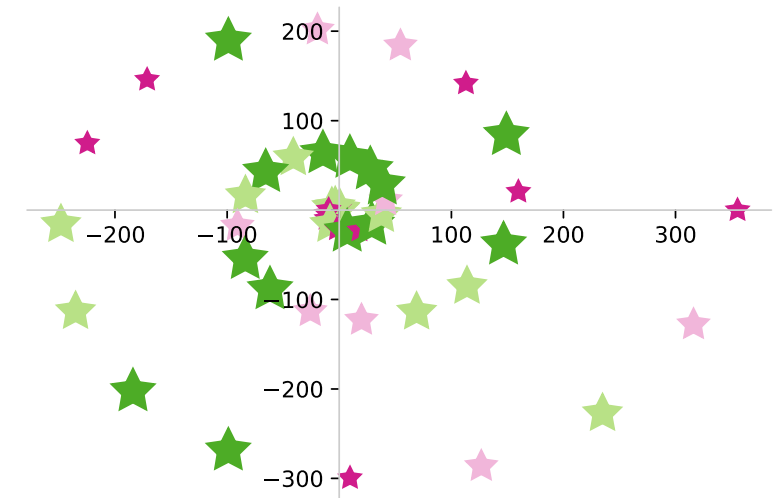
```
theta = np.radians(np.linspace(0,360*3, 50))  
r = theta**2  
x, y = r*np.cos(theta), r*np.sin(theta)  
sz = np.random.randint(1, 5, 50)
```

```
pal = palettable.colorbrewer.diverging.PiYG_4  
ax = plt.figure(figsize=(6,4)).add_subplot()
```

```
ax.scatter(x, y, marker=(5, 1),  
          c=sz, cmap=pal.mpl_colormap,  
          s=sz*100)
```

```
ax.spines['top'].set_visible(False)  
ax.spines['right'].set_visible(False)  
for spine in ['bottom', 'left']:  
    ax.spines[spine].set_position('zero')  
    ax.spines[spine].set_color('0.8')  
ax.xaxis.get_majorticklabels()[3].set_visible(False)  
ax.yaxis.get_majorticklabels()[4].set_visible(False)
```

```
plt.show()
```



Barvy v Matplotlib

- Barvu lze specifikovat několika způsoby (viz [matplotlib.colors](https://matplotlib.org/3.1.1/using-matplotlib.html#color-lookup)):
 - RGB / RGBA n-tice desetinných hodnot v rozsahu `[0,1]` , např. `(0.1, 0.2, 0.5)`
 - hexadecimální RGB / RGBA řetězec, např. `#FF8000` `#FF8000FF`
 - řetězec reprezentující hodnotu v rozsahu `[0,1]` pro úroveň šedé, např. `'0.5'`
 - jednoznaková konstanta `'b', 'g', 'r', 'c', 'm', 'y', 'k', 'w'`
 - řetězec dle [X11/CSS4](#), např. `'Coral'` , `'lawngreen'`
 - řetězec dle [xkcd color survey](#) s prefixem `'xkcd:'` , např. `'xkcd:sky blue'`
 - řetězec z palety Tableau T10 (výchozí paleta pro MPL) `'tab:blue', 'tab:orange', 'tab:green', 'tab:red', 'tab:purple', 'tab:brown', 'tab:pink', 'tab:gray', 'tab:olive', 'tab:cyan'`
 - řetězec tvořený znakem 'C' následovaným jednou číslicí, která určuje index do výchozí palety (`matplotlib.rcParams['axes.prop_cycle']`). Pokud je index mimo rozsah, je použita černá.

Typy čar v Matplotlib (line style)


```
ax.plot(..., linestyle=ls,
         linewidth=lw)
```


■ Typ čáry lze specifikovat následovně (viz [doc](#)):


- jedna z variant symbolických zkratk `'', '-', ':', '--', '-.'` odpovídající následujícím stylům: bez čáry, čárkovaná, čerchovaná, tečkovaná
- jeden z řetězců `'solid', 'dotted', 'dashed', 'dashdot'` korespondující s předchozími styly
- pomocí tzv. dash tuple `(offset, (on_off_sequence))`, která umožňuje detailní kontrolu nad délkou a četností jednotlivých segmentů


■ Příklady dash tuple

- solid odpovídá `(0, ())`, dotted odpovídá `(0, (1, 1))`, definice `(0, (3, 10, 1, 15))` znamená čára 3pt, mezera 10pt, čára 1pt, mezera 15pt, offset 0

'solid' 
solid

'dotted' 
dotted

'dashed' 
dashed


'dashdot' 
dashdot


'-' 
solid


':' 
dotted


'--' 
dashed

'-.' 
dashdot

`(0, (1, 10))` 
custom #1

`(0, (1, 1))` 
custom #2

`(0, (5, 10))` 
custom #3









`(5, (5, 10))` 
custom #4

Značky v Matplotlib (markers)

```
ax.plot(..., marker=markerstyle,
        markeredgewidth=mec,
        markersize=msz,
        markeredgewidth=mew,
        markerfacecolor=mfc)
```

■ Typ značky lze specifikovat pomocí (viz [doc](#)):

- jedné z variant symbolických zkratk `' ', '!', 'o', 'd', 'v', '^', '<', '>', '+', 'x', '*', ...`
- řetězce začínajícího a končícího znakem `$` definující text vysázený pomocí `MathText`, např. `"α"`
- seznamu dvojic určující pozice vrcholů cesty, přičemž střed je umístěn v (0,0), např.
`[(-1, -1), (1, -1), (1, 1), (-1, -1)]`
- dvojice (*numsides*, N) nebo trojice (*numsides*, N, *angle*) pro pravidelný a) polygon s *numsides* stranami (N=0), b) hvězdu s *numsides* cípy (N=1) nebo c) hvězdici s *numsides* paprsky (N=2) otočený o úhel *angle*
- instance třídy [Path](#)

<code>'o'</code> circle		<code>'\$\alpha\$'</code> mathtext	α	<code>[(-1, -1), (1, -1), (1, 1), (-1, -1)]</code> verts	
<code>'>'</code> triangle		<code>'\$\oplus\$'</code> mathtext	\oplus	<code>(5, 0, 10)</code> polygon	
<code>'^'</code> triangle		<code>'\$f\$'</code> mathtext	f	<code>(5, 1)</code> star-like	
<code>'*'</code> star		<code>'\$\u2103\$'</code> unicode	$^{\circ}\text{C}$	<code>Path(...)</code> path	

Aplikace stylu

■ Styl jednotlivých prvků grafu lze definovat

- lokálně pomocí keyword argumentů metod pro manipulaci s grafem

```
ax.plot(x1, y1, c='#35495e', ls='--', label=r'$\frac{\sin(x^2)}{x}$')  
ax.plot(x2, y2, 'C3o-', lw=4, ms=8, mfc='w', label='cos(x)')
```

- globálně pomocí tzv. [runtime configuration](#), která obsahuje výchozí nastavení pro jednotlivé prvky grafu včetně možnosti definovat tzv. [property cycler](#)

```
plt.rc('lines', markersize=8, markerfacecolor='w')
```

- globálně pomocí *stylesheetu* (viz `plt.style.available` a [galerie](#) stylů), které definují výchozí nastavení pro jednotlivé prvky grafu stejně jako runtime configuration.
 - Možnost kombinování více stylů
 - Možnost definice vlastních stylů dostupných ze souboru nebo URL
 - Podpora context manageru pro dočasnou změnu stylu.

```
with plt.style.context('seaborn'):  
    ax = plt.figure(figsize=(6,4)).add_subplot()  
    ax.plot(x1, y1, label=r'$\frac{\sin(x^2)}{x}$')  
    ax.plot(x2, y2, label='cos(x)')
```

? Jak nakonfigurovat `plt.rc`, abychom dosáhli výsledku z první ukázky?



Co jsme vynechali

■ Anotace a práce s textem

- vkládání textu do grafu, anotování prvků grafu
- https://matplotlib.org/api/as_gen/matplotlib.pyplot.annotate.html
- <https://matplotlib.org/tutorials/text/annotations.html>

■ Pokročilejší typy 2D grafů

- grafy v jiném souřadném systému (polární grafy), vrstevnicové grafy, ...

■ 3D grafy

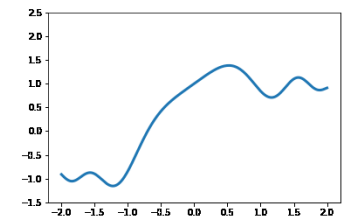
- viz atribut *projection* objektu Axes

■ Animované grafy

- animace v GUI; animovaný GIF, MP4, HTML
- https://matplotlib.org/api/animation_api.html

■ Příklady viz

- <https://matplotlib.org/gallery/index.html>



SVC classification

