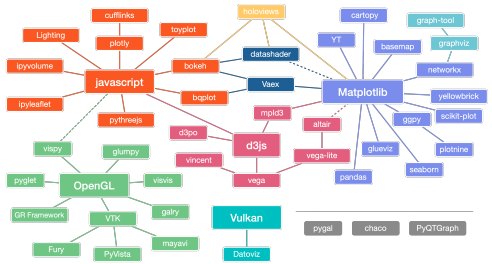
Data Science II
- Python Visualization Landscape -

Prof. Dr. Eduard Kromer
Summer Semester 2024
University of Applied Sciences Landshut

Python Visualization Landscape



How to choose?

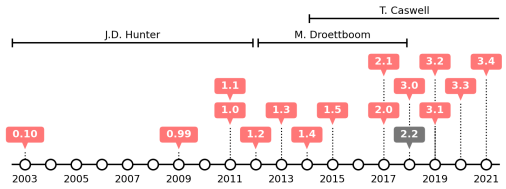
- desktop or web rendering (interactive visualization)?
- publication quality?
- data size?
- active community, good documentation, tutorials?

➡ visit https://pyviz.org and checkout the tutorials section

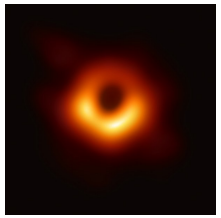
matplotlib

- the de facto standard for Python scientific visualization -

Matplotlib Timeline

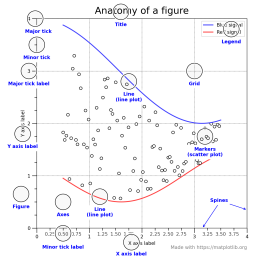


Matplotlib



- versatile and powerful library that allows you to design high quality figures
- it allows you to modify anything within a figure
- the timeline on the previous slide was created with matplotlib

Anatomy of a Figure



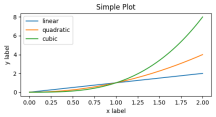
- a matplotlib figure is composed of a hierarchy of elements that form the actual figure
- Figure: the figure itself
  - you can specify size, background color and title
- Axes: the actual area where your data will be rendered; surrounded by spines (left, right, top, bottom)
- Axis: the decorated spines are called axis; x-axis (horizontal) and y-axis (vertical)
- Spines: the lines connecting the axis tick marks noting the boundaries of the data area; may be visible or invisible
- Artist: everything on the figure; when the figure is rendered, all of the artists are drawn to the canvas

Coding Styles

```
x = np.linspace(0, 2, 100)  # Sample data.


fig, ax = plt.subplots(figsize=(5, 2.7), layout='constrained')
ax.plot(x, x, label='linear')  # Plot some data on the axes.
ax.plot(x, x**2, label='quadratic')  # Plot more data on the axes...
ax.plot(x, x**3, label='cubic')  # ... and some more.
ax.set_xlabel('x label')  # Add an x-label to the axes.
ax.set_ylabel('y label')  # Add a y-label to the axes.
ax.set_title("Simple Plot")  # Add a title to the axes.
ax.legend();  # Add a legend.
```

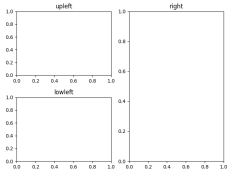
Object Oriented

```
x = np.linspace(0, 2, 100)  # Sample data.

plt.figure(figsize=(5, 2.7), layout='constrained')
plt.plot(x, x, label='linear')  # Plot some data on the (implicit) axes.
plt.plot(x, x**2, label='quadratic')  # etc.
plt.plot(x, x**3, label='cubic')
plt.xlabel('x label')
plt.ylabel('y label')
plt.title("Simple Plot")
plt.legend();
```



Pyplot style

Multiple Axes



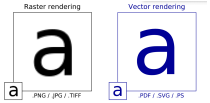
- a figure can have more than one Axes on it
- if you want your axes to be on a regular grid system, then use `plt.subplots(…)` to create a figure and add the axes to it automatically

```
fig, axes = plt.subplots(nrows=2, ncols=2)
```

```
fig, axd = plt.subplot_mosaic([['upleft', 'right'],
                               ['lowleft', 'right']], layout='constrained')
axd['upleft'].set_title('upleft')
axd['lowleft'].set_title('lowleft')
axd['right'].set_title('right');
```

Backends

Renderer	Type	Filetype
Agg	raster	Portable Network Graphic (PNG)
PS	vector	Postscript (PS)
PDF	vector	Portable Document Format (PDF)
SVG	vector	Scalable Vector Graphics (SVG)
Cairo	raster / vector	PNG / PDF / SVG




```
import matplotlib
matplotlib.use("renderer")
```

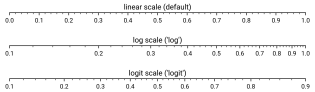
Saving Figures

```
x = np.linspace(0, 2, 100)


fig, ax = plt.subplots(figsize=(5, 2.7))
ax.plot(x, x, label='linear')
ax.plot(x, x**2, label='quadratic')
ax.plot(x, x**3, label='cubic')
ax.set_xlabel('x label')
ax.set_ylabel('y label')
ax.set_title("Simple Plot")
ax.legend()
fig.savefig("my_figure.png")
```

Available file formats (depending on installed backends):

- `eps`
- `jpeg, jpg`
- `pdf`
- `png`
- `ps`
- `svg`
- `tif, tiff`
- …

Scales



Define your own scale

```
def forward(x):
  return x**(1/2)

def inverse(x):
  return x**2

ax.set_xscale('function', functions=(forward, inverse))
```

- scales provide a mapping mechanism between the data and their representation
- matplotlib offers 4 different scales:
  - linear, log, symlog and logit
- each scale can be applied to x-axis only, y-axis only or to both
- log scales are used for values that are strictly positive
- the logit scale is used for values in (0,1) and uses a logarithmic scale on the border and quasi-linear scale in the middle (around 0.5)

Typography

- typography for visualizations means typeface, scripts, unicode, hinting, shaping, kerning, weight, slant …
- it is important to have a basic understanding of typography
  - read at least *Typography in ten minutes* by M. Butterick
- the matplotlib fontstack is defined using four different typeface families: sans, serif, monospace and cursive
- the default font stack is based on the DejaVu fonts

Fonts

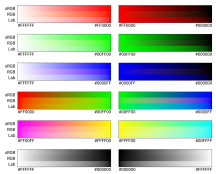
```
from matplotlib.font_manager import findfont, FontProperties
for family in ['serif', 'sans', 'monospace', 'cursive']:
  font = findfont(FontProperties(family=family))
  print(family, ':' , os.path.basename(font))
```

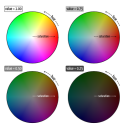
```
serif ; DejaVuSerif.ttf
sans ; DejaVuSans.ttf
monospace ; DejaVuSansMono.ttf
cursive ; Apple Chancery.ttf
```

Serif	Sans	Monospace	*Cursive*
Serif	Sans	Monospace	*Cursive*
Serif	Sans	Monospace	*Cursive*

- you can design your own font stack by choosing a set of alternative font families
- the default can be changed by modifying the rcParams or the stylesheets

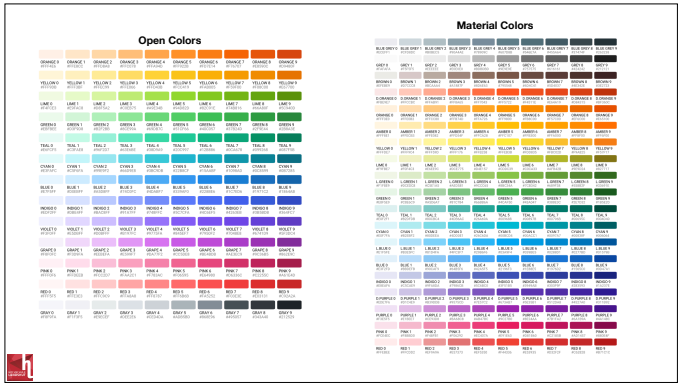
Colors





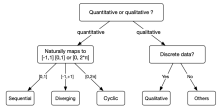
HSV Color wheel

- to represent color on a computer we use a color model (how do we represent color) and a color space (what colors can be represented)
- color models: RGB, HSV, CMYK, …
- color spaces: sRGB, Adobe RGB, …
  - sRGB is the standard color space; mix different amounts of red, green and blue light to obtain a color of your choice
- when you specify a color in matplotlib, it is encoded in the sRGB color model and space

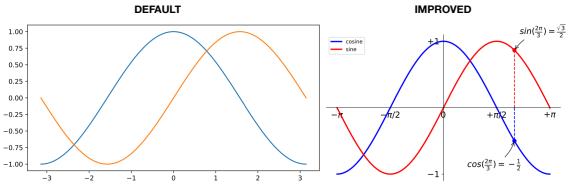


Color Maps

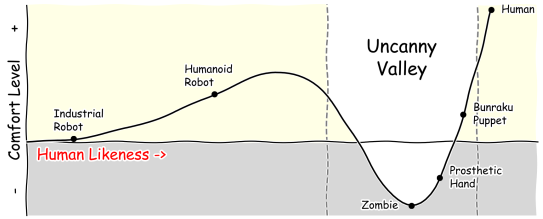
- colormapping corresponds to the mapping of values to colors
- it is important to use the right colormap that corresponds to your data
- How to choose the right colormap?



Don't Trust the Defaults

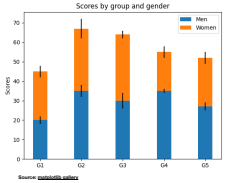


Message Trumps Beauty



Examples

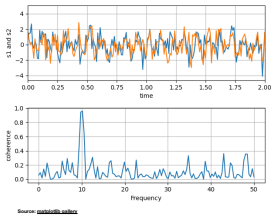
Stacked Bar Chart




```python
import matplotlib.pyplot as plt


labels = ['G1', 'G2', 'G3', 'G4', 'G5']
men_means = [20, 35, 30, 35, 27]
women_means = [25, 32, 34, 20, 25]
men_std = [2, 3, 4, 1, 2]
women_std = [3, 5, 2, 3, 3]
width = 0.35 # the width of the bars: can also be len(x) sequence

fig, ax = plt.subplots()

ax.bar(labels, men_means, width, yerr=men_std, label='Men')
ax.bar(labels, women_means, width, yerr=women_std, bottom=men_means,
       label='Women')

ax.set_ylabel('Scores')
ax.set_title('Scores by group and gender')
ax.legend()

plt.show()
```

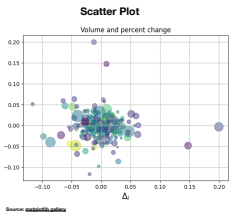
Examples

Multiple Line Plots




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

# Fixing random state for reproducibility
np.random.seed(19680801)

dt = 0.01
t = np.arange(0, 30, dt)
nse1 = np.random.randn(len(t))         # white noise 1
nse2 = np.random.randn(len(t))         # white noise 2

# Two signals with a coherent part at 10Hz and a random part
s1 = np.sin(2 * np.pi * 10 * t) + nse1
s2 = np.sin(2 * np.pi * 10 * t) + nse2

fig, axs = plt.subplots(2, 1)
axs[0].plot(t, s1, t, s2)
axs[0].set_xlim(0, 2)
axs[0].set_xlabel('time')
axs[0].set_ylabel('s1 and s2')
axs[0].grid(True)

cxy, f = axs[1].cohere(s1, s2, 256, 1. / dt)
axs[1].set_ylabel('coherence')

fig.tight_layout()
plt.show()
```

Examples



Source: matplotlib gallery

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


price_data = (cbook.get_sample_data('goog.npz', np_load=True)['price_data']
                   .view(np.recarray))
price_data = price_data[-250:]  # get the most recent 250 trading days

deltal = np.diff(price_data.adj_close) / price_data.adj_close[:-1]

# Marker size in units of points^2
volume = (15 * price_data.volume[:-2] / price_data.volume[0])**2
close = 0.003 * price_data.close[:-2] / 0.003 * price_data.open[:-2]

fig, ax = plt.subplots()
ax.scatter(deltal[:-1], deltal[1:], c=close, s=volume, alpha=0.5)

ax.set_xlabel(r'$\Delta_i$', fontsize=15)
ax.set_ylabel(r'$\Delta_{i+1}$', fontsize=15)
ax.set_title('Volume and percent change')

ax.grid(True)
fig.tight_layout()

plt.show()
```

Your Turn

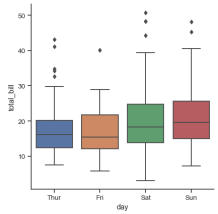
- read and work through the following matplotlib tutorials:
  - basic usage
  - pyplot
  - the lifecycle of a plot
- try to recreate the right plot on slide 16 (sin and cosine functions) without the formula annotations

seaborn

seaborn

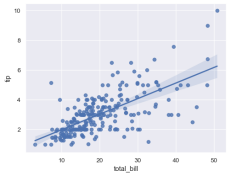
- if you are a little overwhelmed by the customizability of matplotlib, the seaborn library might be a good choice to start your data visualization journey
- it is based on matplotlib and provides a high-level interface for drawing attractive and informative statistical graphics
- installation: `conda install seaborn`

seaborn examples



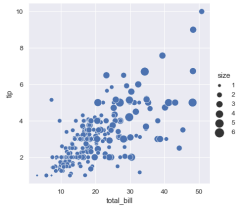

```
import seaborn as sns
sns.set_theme(style="ticks", color_codes=True)

tips = sns.load_dataset("tips")
sns.catplot(x="day", y="total_bill", kind="box", data=tips)
```


seaborn examples




```
import seaborn as sns
sns.set_theme(style="ticks", color_codes=True)

tips = sns.load_dataset("tips")
sns.regplot(x="total_bill", y="tip", data=tips);
```

seaborn examples




```
import seaborn as sns
sns.set_theme(style="ticks", color_codes=True)

tips = sns.load_dataset("tips")
sns.relplot(x="total_bill", y="tip", size="size", sizes=(15, 200), data=tips);
```

References

- Slide 2,5: N.P. Rougier, Scientific Visualization - Python & Matplotlib
- Slide 6: Wikipedia: https://en.wikipedia.org/wiki/Black_hole
- Slide 7: Matplotlib Tutorial: https://matplotlib.org/stable/tutorials/introductory/usage.html#sphx-glr-tutorials-introductory-usage-py
- Slide 9,10,12,14-19: N.P. Rougier, Scientific Visualization - Python & Matplotlib
- Slide 20: Matplotlib Gallery: https://matplotlib.org/devdocs/gallery/lines_bars_and_markers/bar_stacked.html#sphx-glr-gallery-lines-bars-and-markers-bar-stacked-py
- Slide 21: Matplotlib Gallery: https://matplotlib.org/devdocs/gallery/lines_bars_and_markers/cohere.html#sphx-glr-gallery-lines-bars-and-markers-cohere-py
- Slide 22: Matplotlib Gallery: https://matplotlib.org/devdocs/gallery/lines_bars_and_markers/scatter_demo2.html#sphx-glr-gallery-lines-bars-and-markers-scatter-demo2-py