

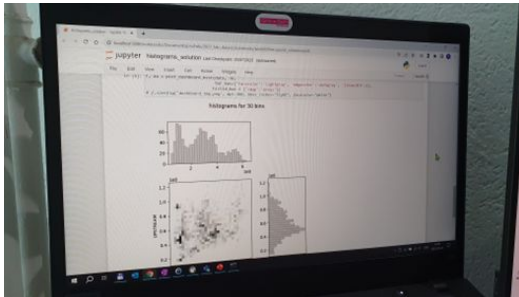
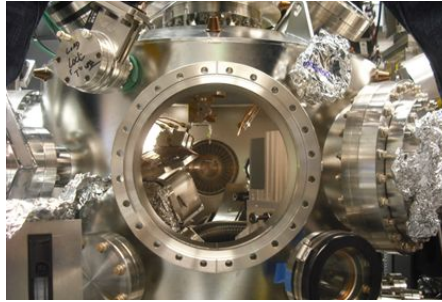
Did you know Matplotlib could do that?

Dr. Teresa Kubacka

@paniterka_ch | www.teresa-kubacka.com

Swiss Python Summit 2023, 21.09.2023

About me

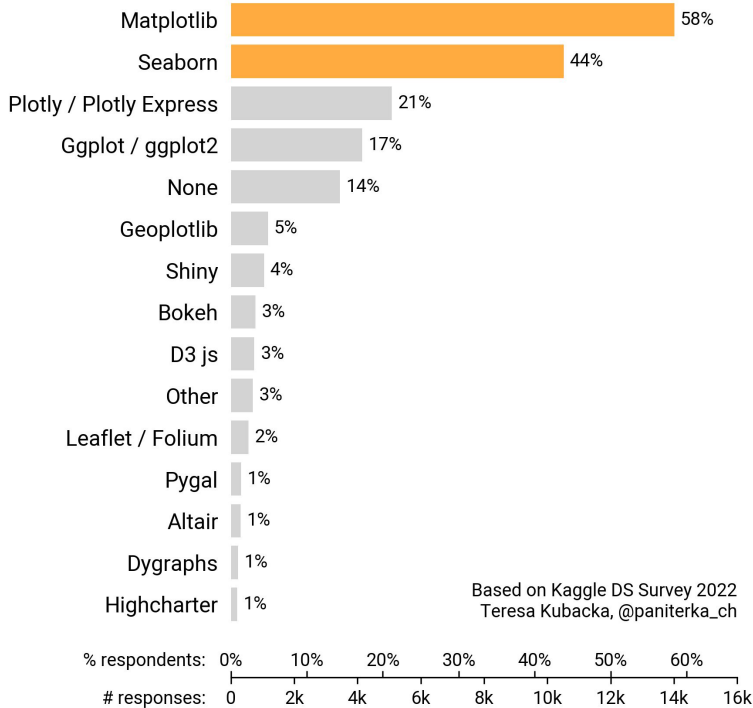


ETH zürich



HSLU Lucerne University
of Applied Sciences
and Arts

Almost 60% of data scientists use MPL on a regular basis



Posted by u/[deleted] 4 years ago

201



Am I the only one who hates matplotlib?

I've been working with python for 3-4 years. Some breaks in this timespan and a lot of different projects.

I love to do so many different projects because there are always new exciting libraries to explore. But matplotlib was since the beginning horror for me. Its the only library where I have massive problems to understand the docs and the basic concept.

Posted by u/rh1994 3 years ago



Does matplotlib ever actually make sense or do you just learn it by heart?

▲ smabie on Nov 16, 2019 | parent | context | favorite | on: Effectively Using Matplotlib (2017)

Matplotlib belongs to the worst category of software: very powerful and very awful. Nothing makes any sense and it's so profoundly unintuitive it almost feels like I'm being pranked. But, of course, use it I must.

Pandas also comes off as an unintuitive joke, but my displeasure with it has *mostly* worn off. Matplotlib however makes me feel angry pretty much everyday.

Posted by u/Live_Solution_8851 9 months ago

Matplotlib sucks

discussion

Matplotlib is the worst plotting library i have ever used:

MPL is super powerful!

animations

3D plotting

easy switch between exporting to
vector or raster formats

interactive charts

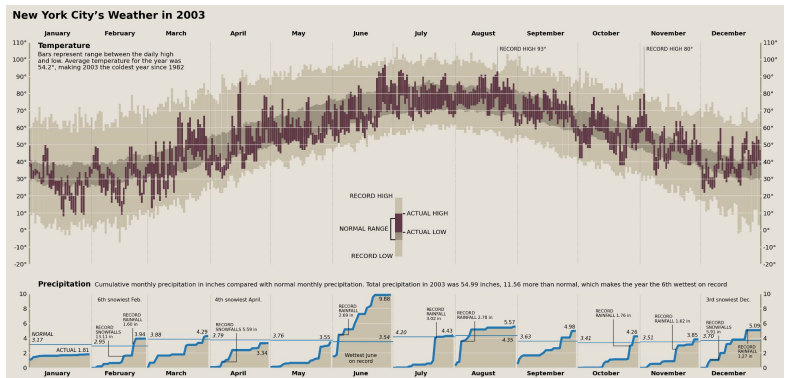
many different renderers

native Python GUIs, web-apps

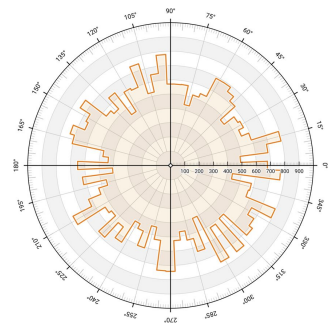
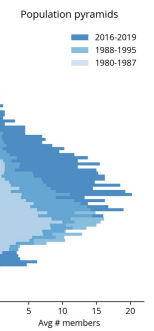
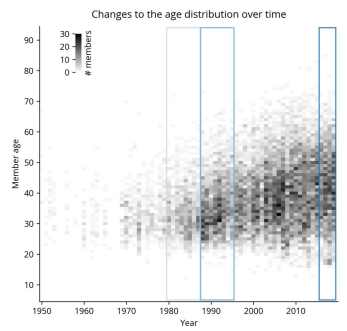
you can quite easily build your own
data-driven components from scratch

customized, beautiful charts

Dataviz for analytics

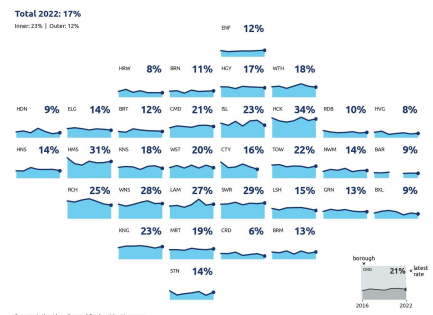


Cameron Riddell
(after E. Tufte)



LONDON CYCLING RATES

Proportion of people cycling for any purpose at least once a month, 2016-2022 (%)

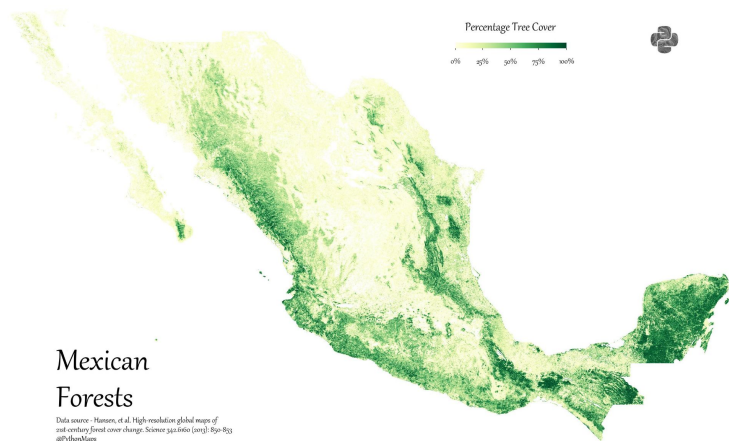


Teresa Kubacka

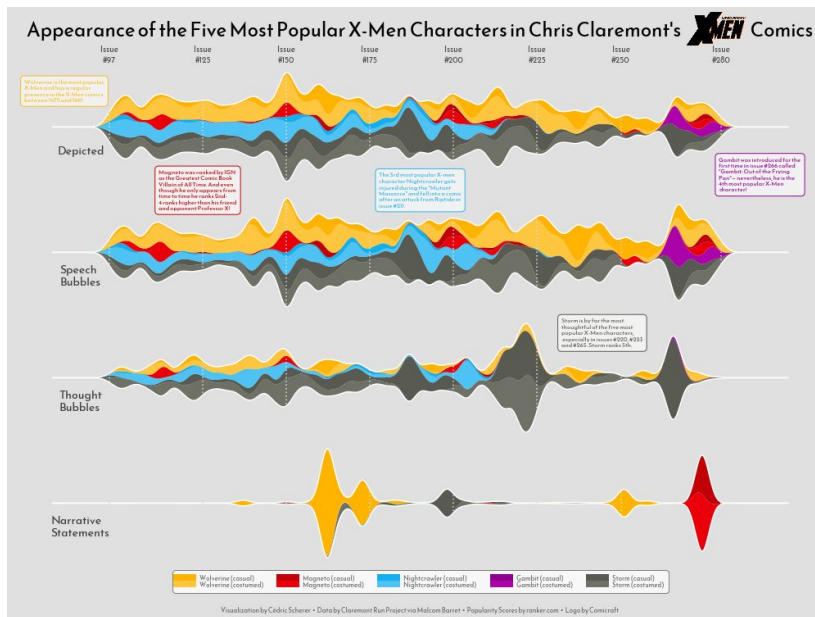
Nicolas P. Rougier

Lisa Hornung

Infographics

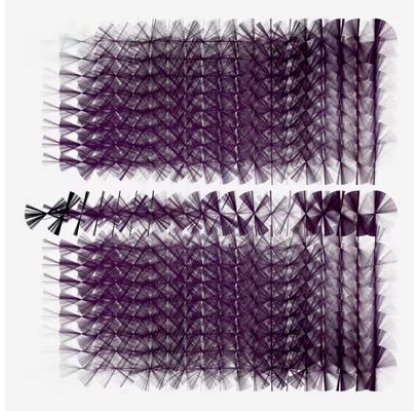


Adam Symington / PythonMaps

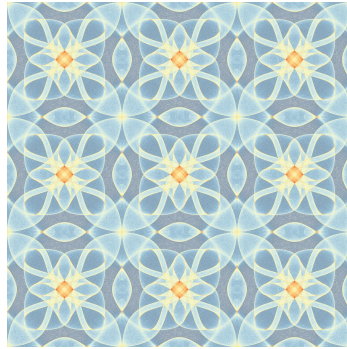


Tomi Capretto (after C. Scherer)

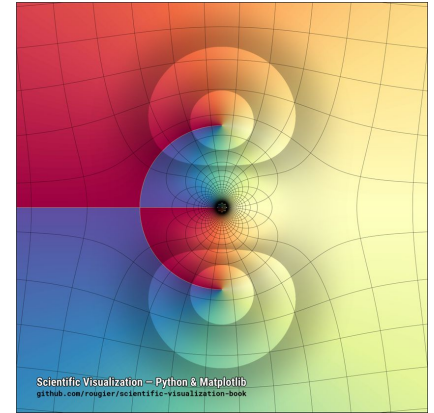
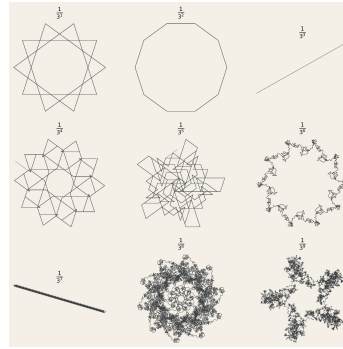
Data art



chiko.art



Simone Conradi



Nicolas P. Rougier

Plan for today

1

Understand the inner workings of MPL

2

Get familiar with some of the awesome parts of MPL

3

Your visualization is code:

- DRY: functions, properties as variables, stylesheets...
- Loops, zip, enumerate...

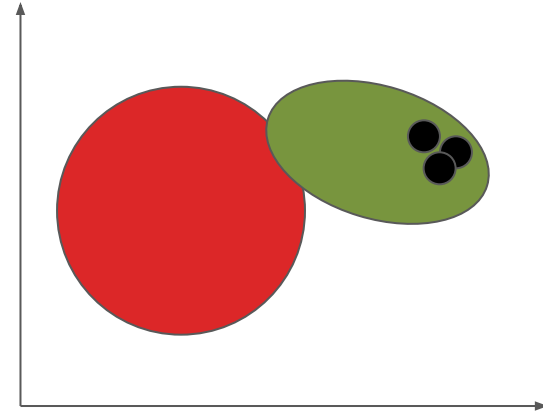
4

How to design better charts

1: Understand MPL

Dataviz and intuition

We have a natural intuition for working with visuals

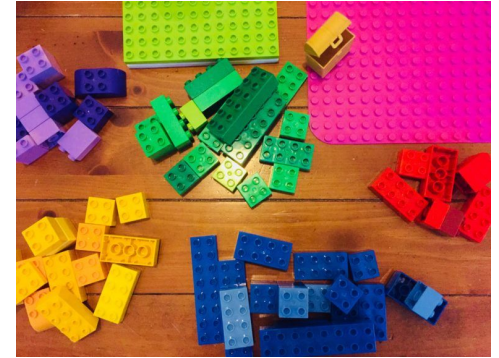


Dataviz and intuition

We have some intuition for data visualization (that we train since our early days)



<https://www.walmart.com/ip/KMTJT-Montessori-Toys-1-2-3-Year-Old-Color-Sorting-Stacking-Matching-Learning-Toy-1-3-Olds-Educational-Toy-Gift-Boy-Girl/3704572425>



<https://www.glitteronadime.com/make-a-rainbow-castle-with-lego-duplo-blocks/>

Dataviz and intuition

But we don't have that same kind of intuition for coding charts!

```
fig, ax = plt.subplots()
ax.plot(x, y_est, '-')
ax.fill_between(x,
               y_est - y_err, y_est + y_err,
               alpha=0.2)
ax.plot(x, y, 'o', color='tab:brown')
```

Matplotlib

```
base = alt.Chart(source).mark_circle(
    opacity=0.5
).transform_fold(
    fold=['A', 'B', 'C'],
    as_=['category', 'y']
).encode(
    alt.X('x:Q'),
    alt.Y('y:Q'),
    alt.Color('category:N')
)
```

Altair

```
(
    ggplot(mpg, aes(x='displ', y='hwy'))
    + geom_point()
    + geom_smooth(span=.3)
    + labs(x='displacement', y='horsepower')
)
```

Plotnine / ggplot2

```
dict_of_fig = dict({
    "data": [{"type": "bar",
              "x": [1, 2, 3],
              "y": [1, 3, 2]}],
    "layout": {"title":
               {"text": "My Figure"}}
})
fig = go.Figure(dict_of_fig)
fig.show()
```

Plotly



Dataviz and intuition

But we don't have that same kind of intuition for coding charts!

```
fig, ax = plt.subplots()
ax.plot(x, y_est, '-')
ax.fill_between(x,
               y_est - y_err, y_est + y_err,
               alpha=0.2)

(
    ggplot(mpg, aes(x='displ', y='hwy'))
    + geom_point()
    + geom_smooth(span=.3)
    + labs(x='displacement', y='horsepower')
```



Our intuition for coding is acquired and depends on what we have learnt and understood in the past.

So to find Matplotlib “intuitive” we have to understand how it works!

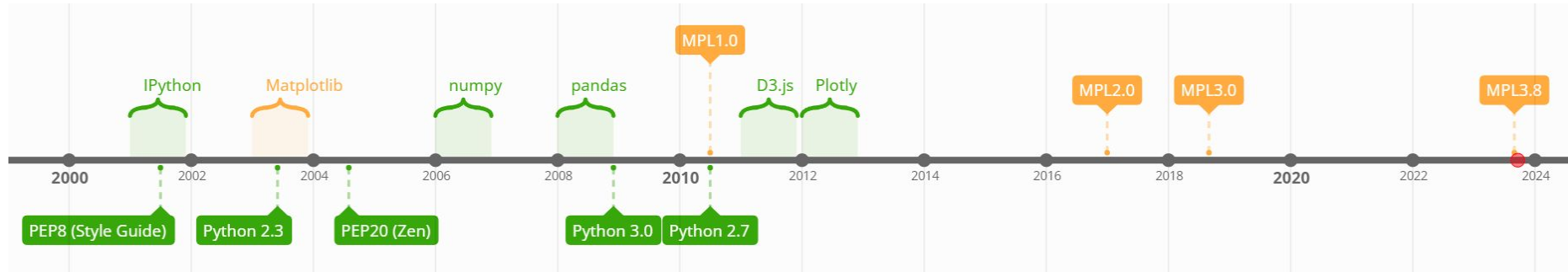
```
).transform_fold(
    fold=['A', 'B', 'C'],
    as_=['category', 'y']
).encode(
    alt.X('x:Q'),
    alt.Y('y:Q'),
    alt.Color('category:N')
)
```

Altair

```
"x": [1, 2, 3],
"y": [1, 3, 2]},
"layout": {"title":
            {"text": "My Figure"}}
    })
fig = go.Figure(dict_of_fig)
fig.show()
```

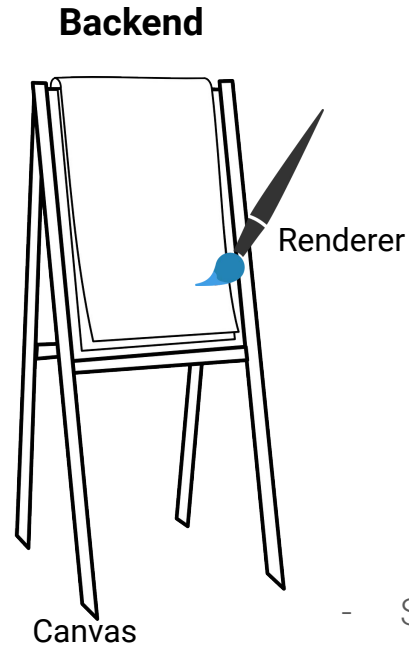
Plotly

20 years is an advantage, but also a disadvantage



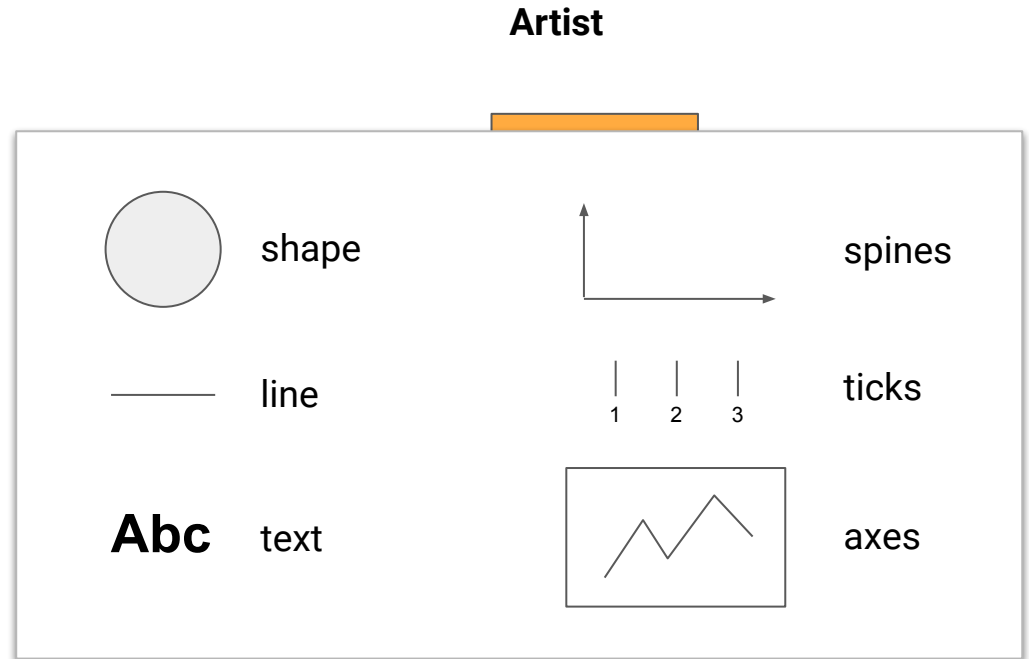
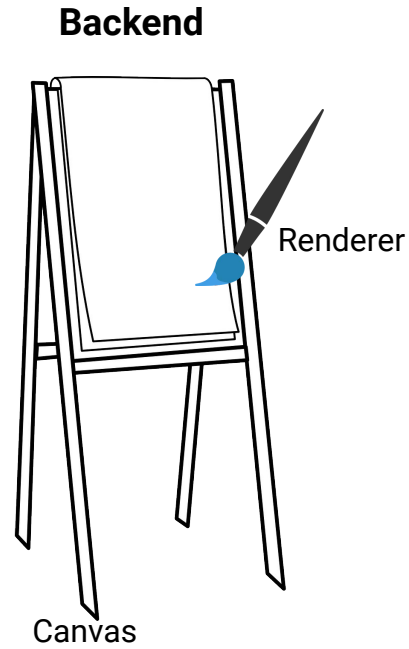
- Matplotlib has been created around 2003 and has undergone few major changes
- It is a mature dataviz library with lots of functionality
- Backward compatibility is very important
- Most of the online teaching material is plagued by legacy code which is discouraged

Main guiding idea behind MPL's architecture

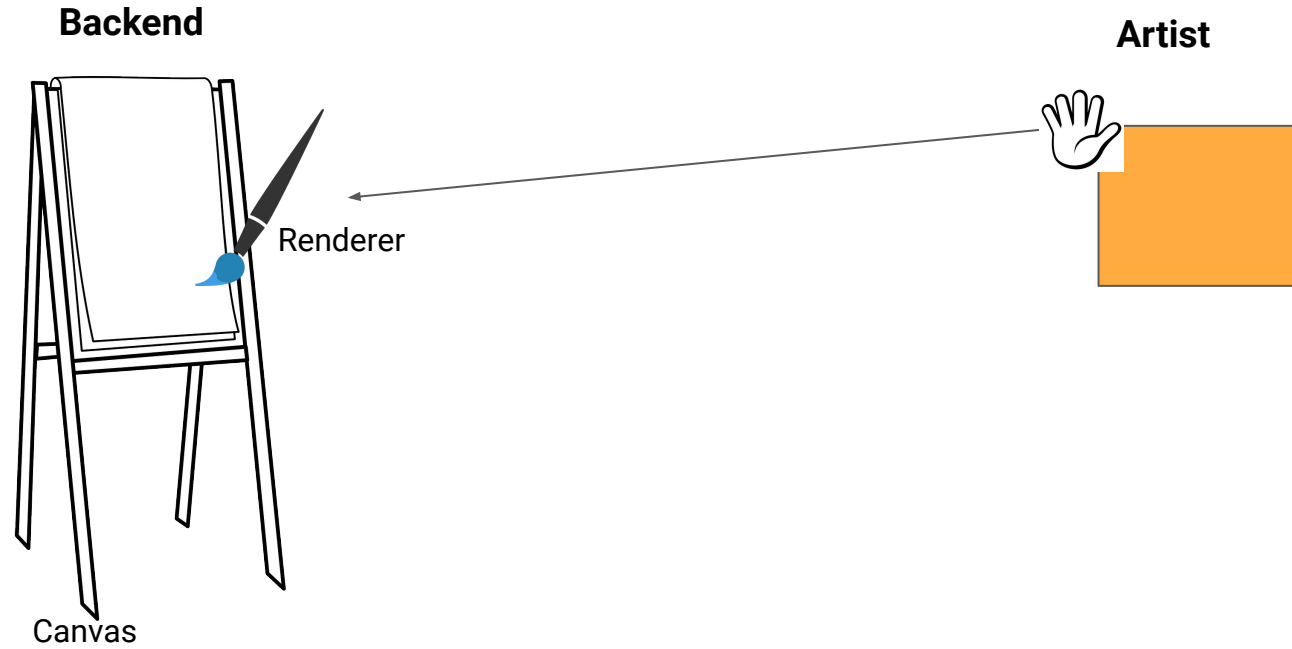


- Something that draws
- Static or interactive, vector or pixel...
- You can write your own backend

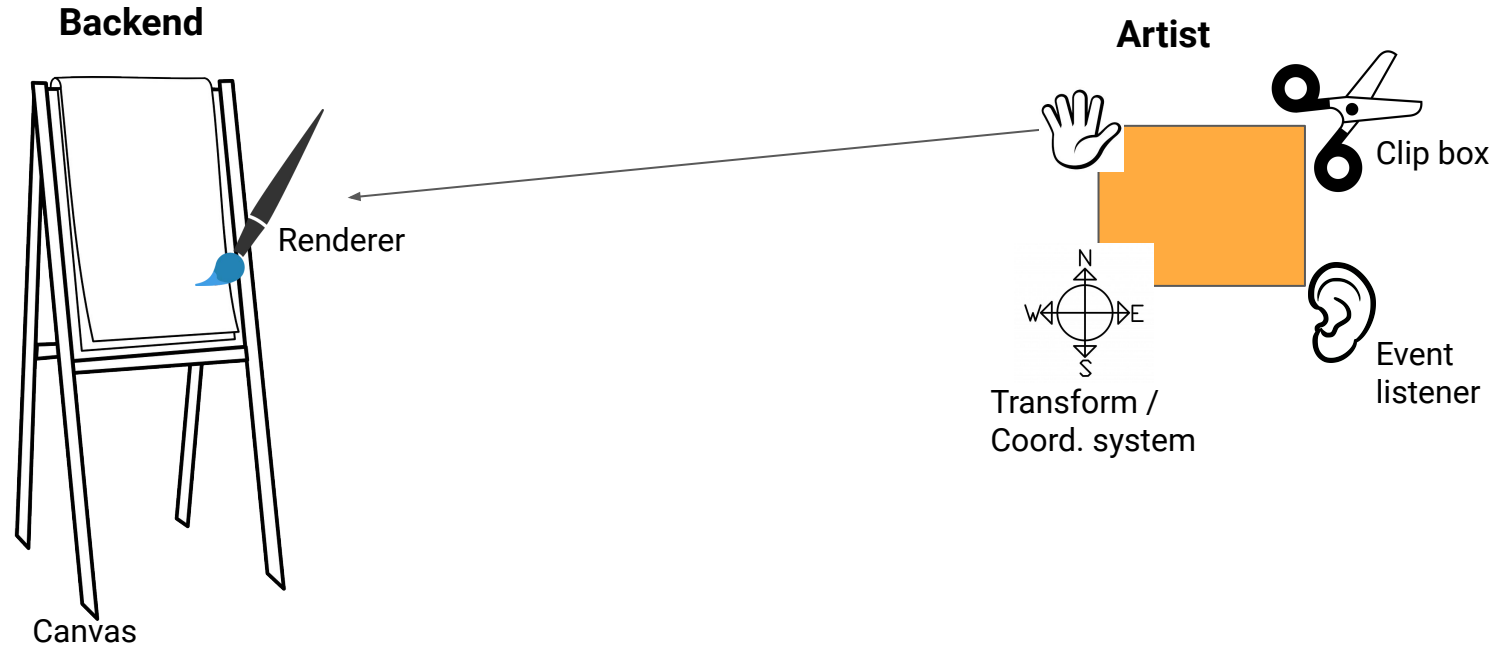
Main guiding idea behind MPL's architecture



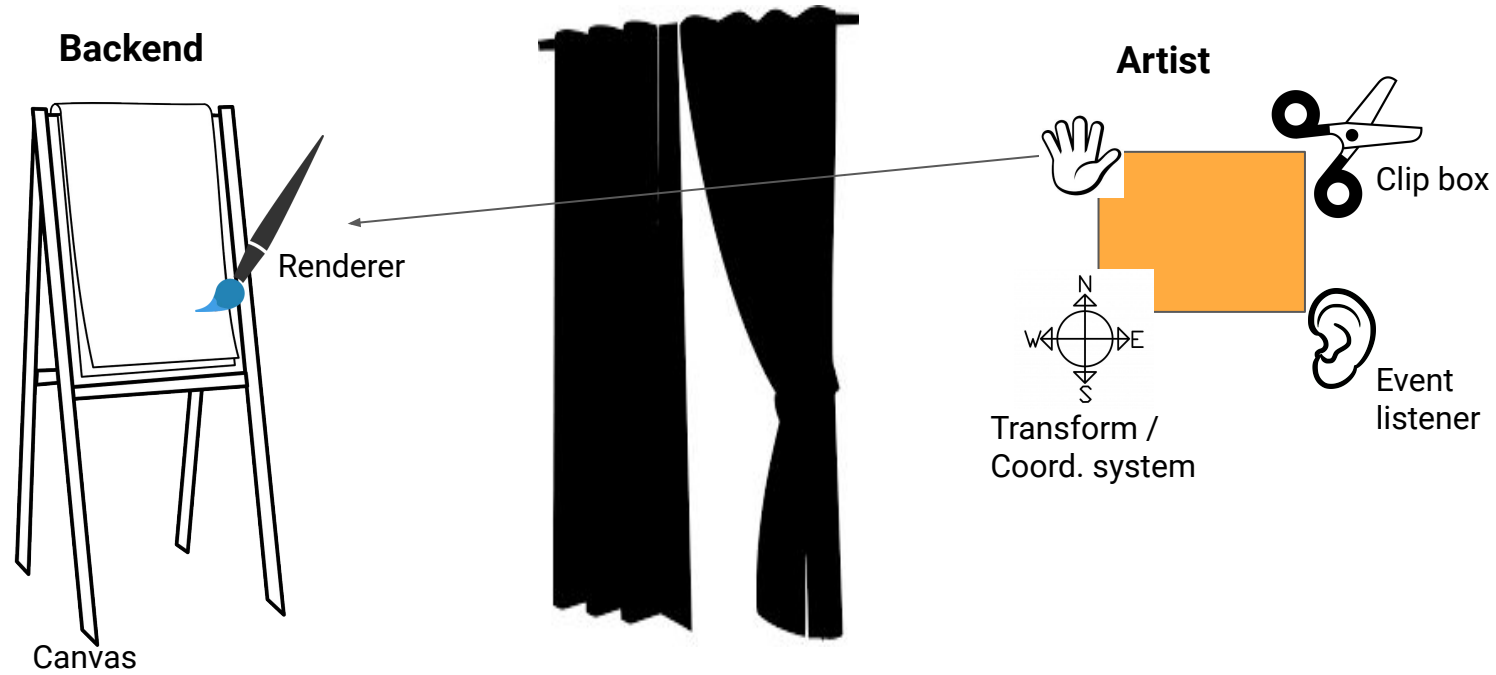
Main guiding idea behind MPL's architecture



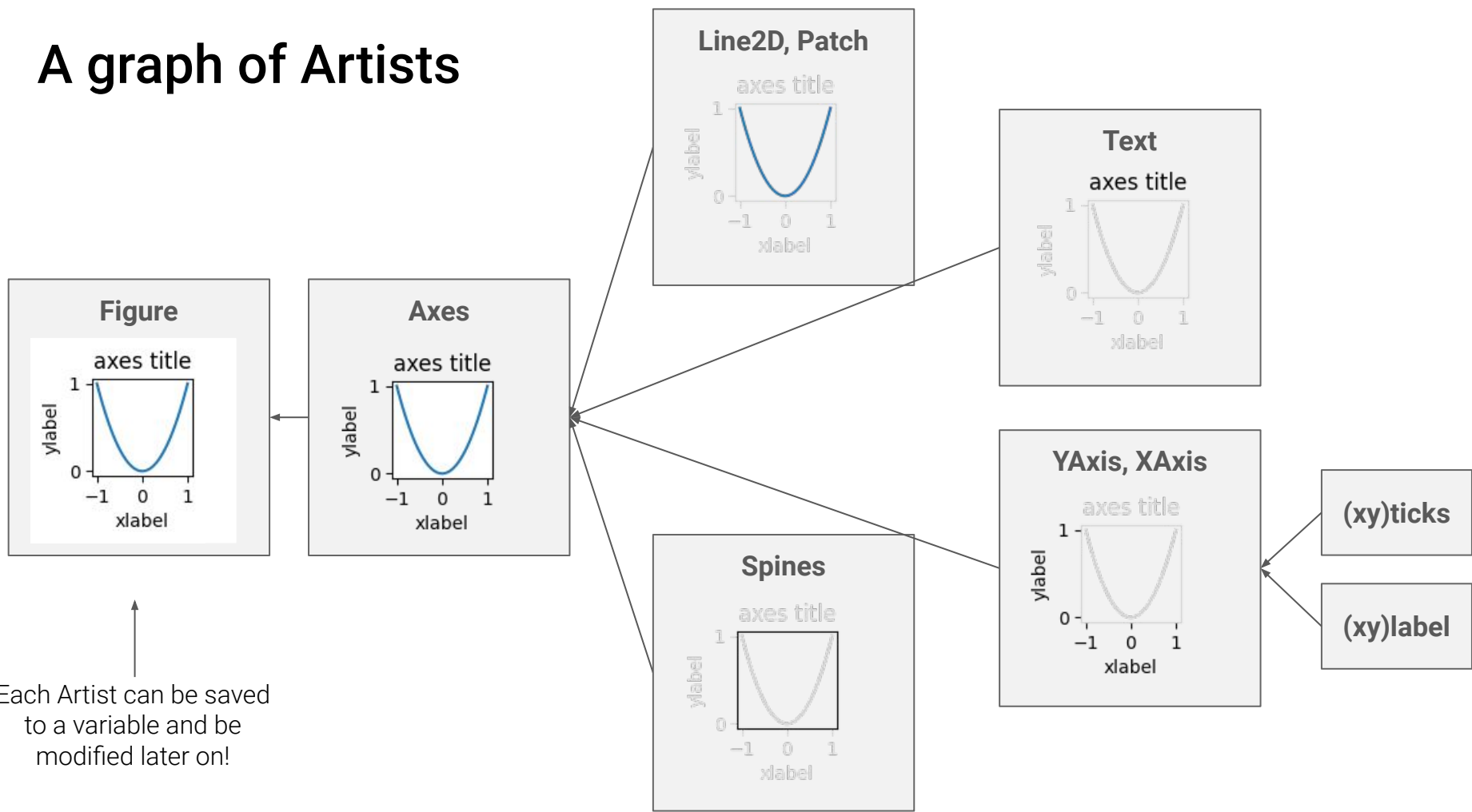
Main guiding idea behind MPL's architecture



Main guiding idea behind MPL's architecture



A graph of Artists



Each Artist can be saved to a variable and be modified later on!

MPL is composed of layers

ax.{...}

Artists

chart as a graph of
components

Backend

does the actual drawing

But what we typically do is

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

So where is all the PLT-stuff in this model?

plt.{...}

Scripting

create and modify groups of artists

ax.{...}

Artists

chart as a graph of components

Backend

does the actual drawing

For historical reasons: MATLAB-like functions

```
x = -pi:pi/10:pi;
y = tan(sin(x)) - sin(tan(x));

figure
plot(x,y,'--gs',...
      'LineWidth',2,...
      'MarkerSize',10,...
      'MarkerEdgeColor','b',...
      'MarkerFaceColor',[0.5,0.5,0.5])
title('2-D Line Plot')
xlabel('x')
ylabel('y')
```

Matlab

PLT-API (MATLAB-API) is **evil!**

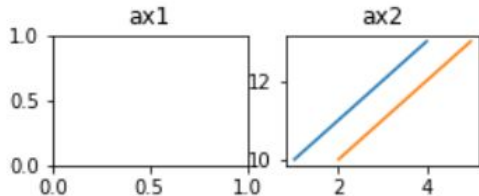
It tries to guess what is the latest active object.

```
x = np.array([1, 2, 3, 4])
y = np.array([10, 11, 12, 13])
```

```
f, axes = plt.subplots(ncols=2, figsize=(4,1.25))
axes[0].set_title('ax1')
axes[1].set_title('ax2')
```

```
for i,ax in enumerate(axes):
    plt.plot(x+i,y)
    print('plotted in {}'.format(ax.get_title()))
```

plotted in ax1
plotted in ax2

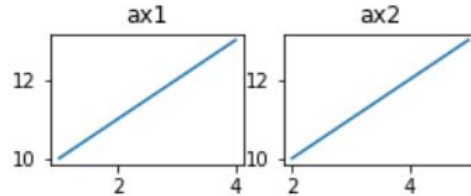


PLT / MATLAB / implicit API

```
f, axes = plt.subplots(ncols=2, figsize=(4,1.25))
axes[0].set_title('ax1')
axes[1].set_title('ax2')
```

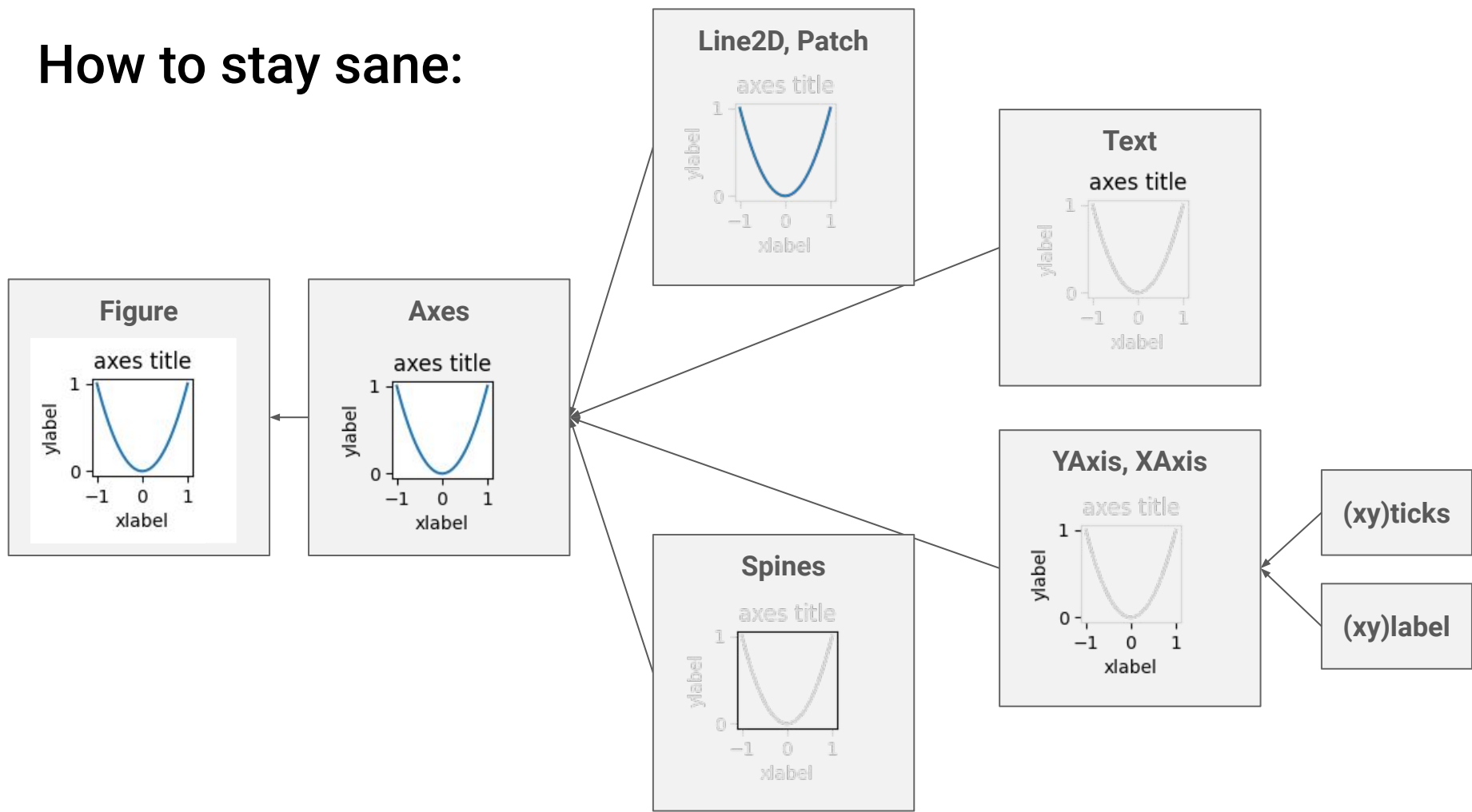
```
for i,ax in enumerate(axes):
    ax.plot(x+i,y)
    print('plotted in {}'.format(ax.get_title()))
```

plotted in ax1
plotted in ax2



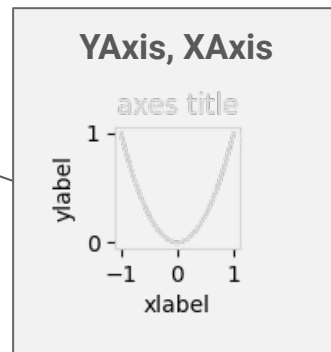
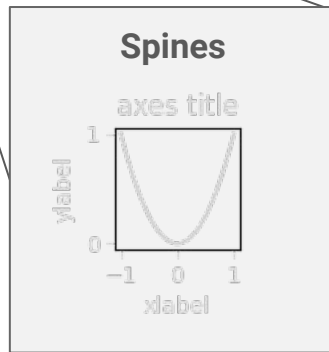
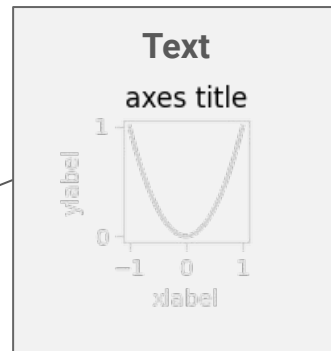
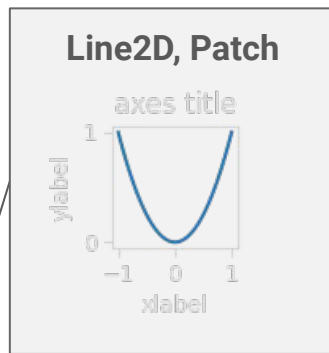
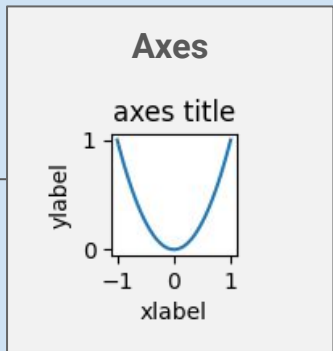
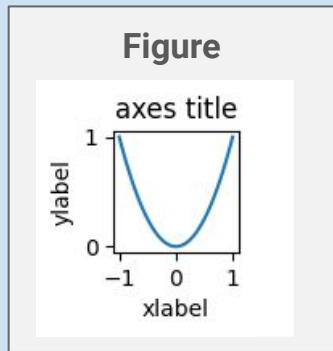
object-oriented / explicit API

How to stay sane:



How to stay sane:

Use the PLT-API only to initialize f, ax

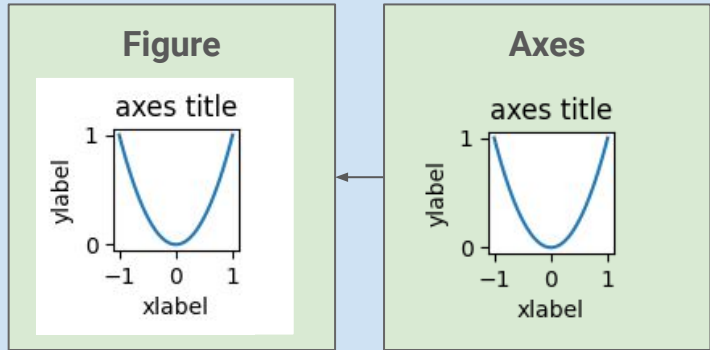


(xy)ticks

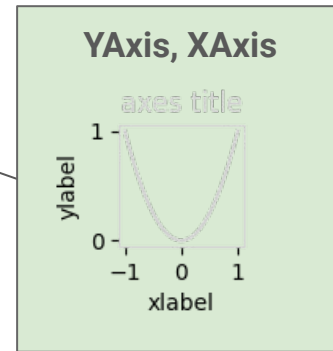
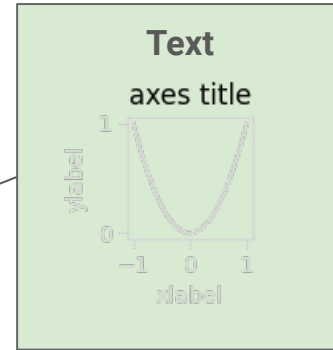
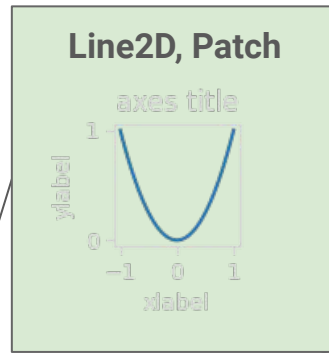
(xy)label

How to stay sane:

Use the PLT-API only to initialize `f`, `ax`

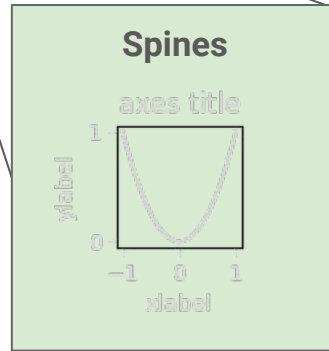


Use the OO-API to do everything else with `f`, `ax` and other objects directly



(xy)ticks

(xy)label





`ax.{...}`

Artists

chart as a graph of components

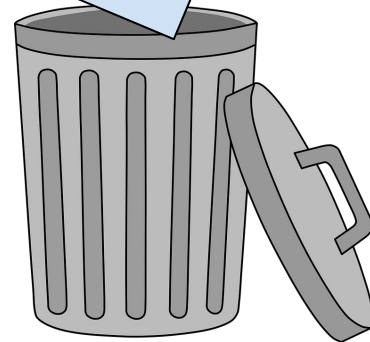
`plt.{layout}`

Backend

does the actual drawing

`plt.{...}`

Scripting
create and modify groups of artists



2: The awesome parts

matplotlib.pyplot is a tip of the iceberg

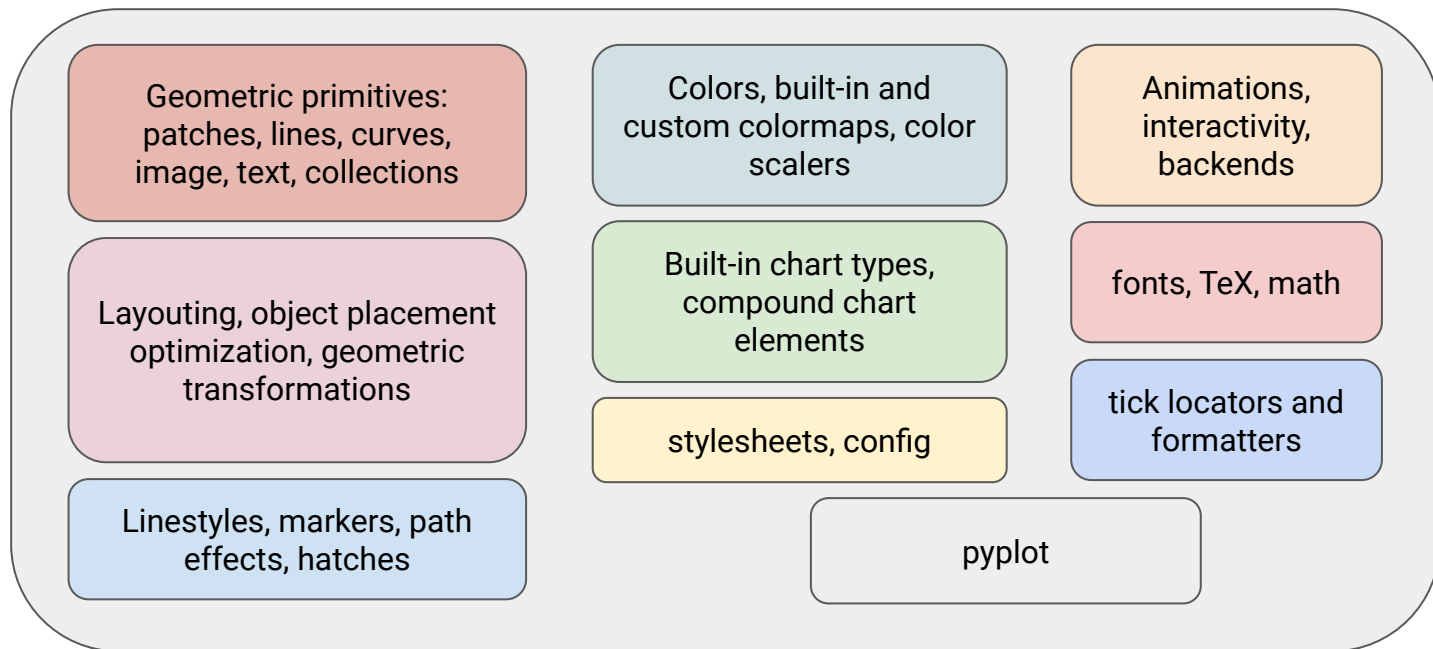
```
import matplotlib.pyplot as plt
```



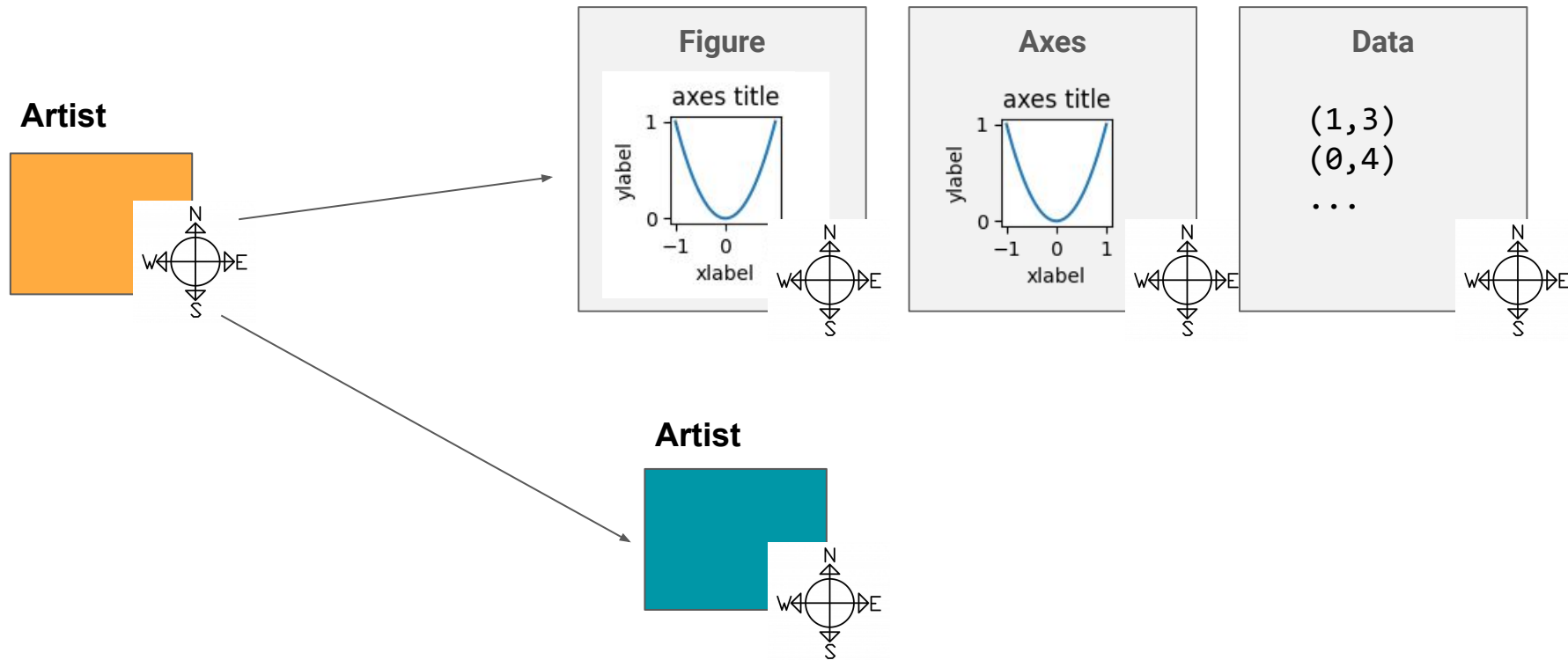
pyplot

matplotlib.pyplot is a tip of the iceberg

```
import matplotlib.pyplot as plt  
from matplotlib.{...} import {...}
```

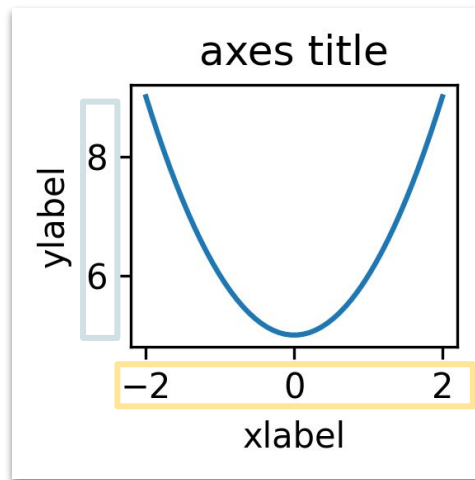


Coordinate systems



Coordinate systems

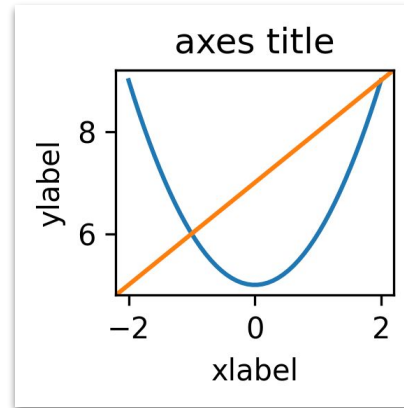
```
def example_plot(plot_kws = {}):  
    x = np.linspace(-2,2,) # -2..2  
    y = x**2+5 # 5..9  
    f, ax = plt.subplots(figsize=(2,2), layout='tight')  
    ax.plot(x, y, **plot_kws)  
    ax.set_title('axes title')  
    ax.set_xlabel('xlabel')  
    ax.set_ylabel('ylabel')  
    return f, ax  
  
f, ax = example_plot()
```



Coordinate systems

```
f, ax = example_plot()
ax.plot([0, 1], [0, 1],
        transform=ax.transAxes)
```

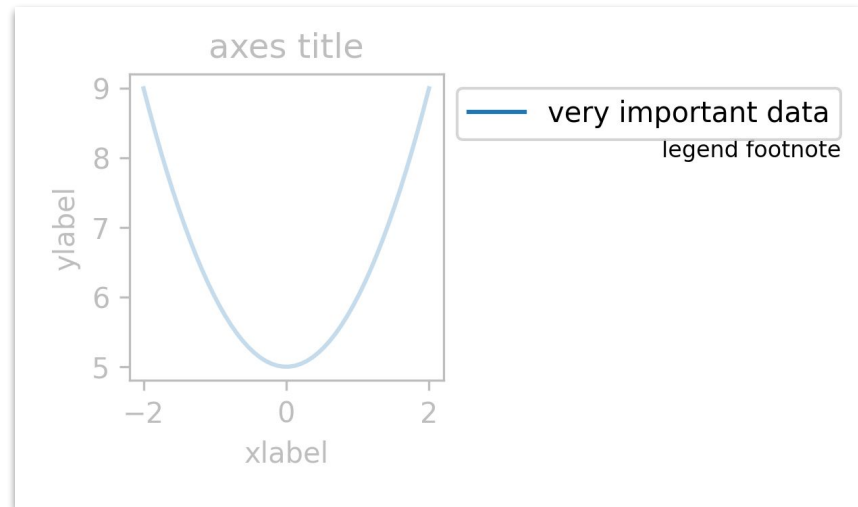
Plot an additional line from 0% to 100% of the extents of the `ax` on both `x` and `y` axis. Useful for watermarks, guiding lines etc.



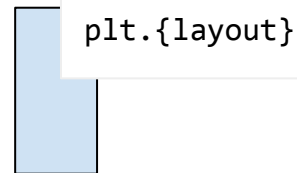
Coordinate systems

```
f, ax = example_plot_with_blur()
legend_object = ax.legend(bbox_to_anchor=[1,1])
legend_footnote = ax.annotate(
    text='legend footnote', fontsize=8,
    xy=[1,0], xycoords=legend_object,
    va='top', ha='right'
)
```

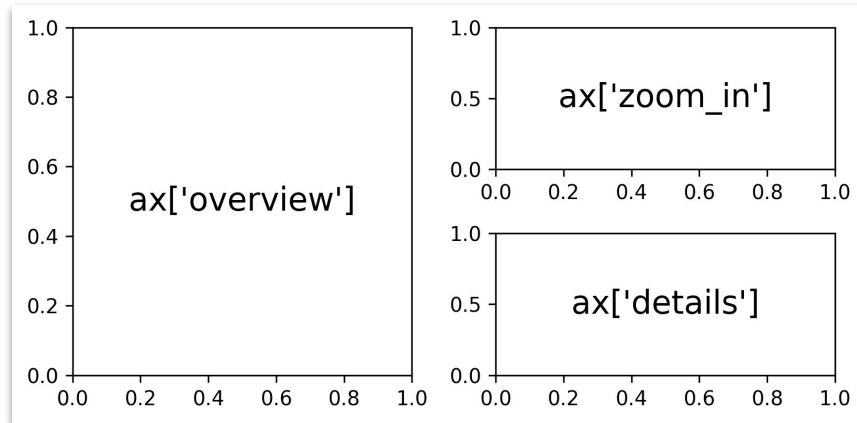
Anchor the top-right corner of the legend_footnote to the point on the chart where the legend_object has 100% of its width and 0% of its height



Clever layouting: subplot_mosaic



```
f, ax = plt.subplot_mosaic(  
    mosaic = [['overview', 'zoom_in'],  
             ['overview', 'details']],  
    figsize=(6,3), layout='tight'  
)  
  
for k in ['overview', 'zoom_in', 'details']:  
    ax[k].text(s=f"ax['{k}']",  
              x=0.5, y=0.5,  
              ha='center', va='center',  
              fontproperties={'size':16})
```



Create layouts using `arrays of keys`, which are then used to `address a particular axes` of a figure

Effortless tick positions

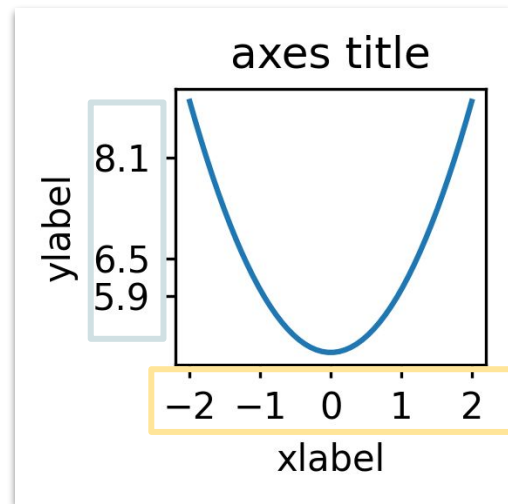
`matplotlib.ticker` contains in-built tick generators:

```
from matplotlib.ticker import MultipleLocator, FixedLocator

f, ax = example_plot()

ax.xaxis.set_major_locator(MultipleLocator(1))
ax.yaxis.set_major_locator(FixedLocator([5.9, 6.5, 8.1]))
```

Tick location generator can be attached to the each `Spine` independently.



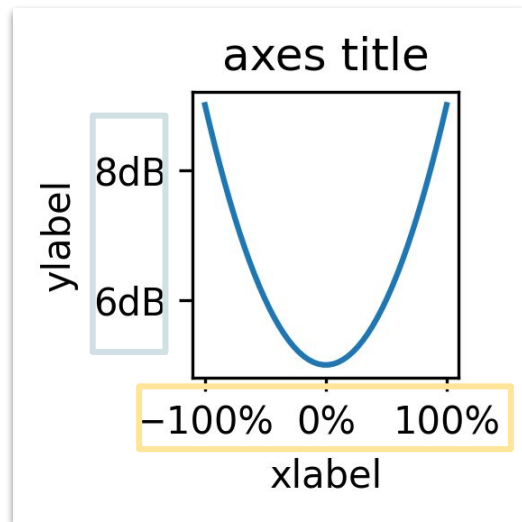
Effortless tick label formatting

```
from matplotlib.ticker import PercentFormatter, StrMethodFormatter

f, ax = example_plot()

ax.xaxis.set_major_formatter(PercentFormatter(xmax=2))
ax.yaxis.set_major_formatter(StrMethodFormatter('{x:.0f}dB'))
```

Similarly, attach a pre-built or completely custom string formatting function that should be applied for each tick label on a Spine.

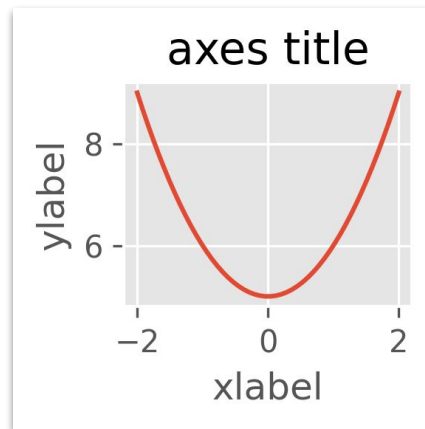


Stylesheets

```
import matplotlib as mpl
```

```
mpl.style.use('ggplot')  
example_plot()
```

```
with mpl.style.context('ggplot'):  
    example_plot()
```

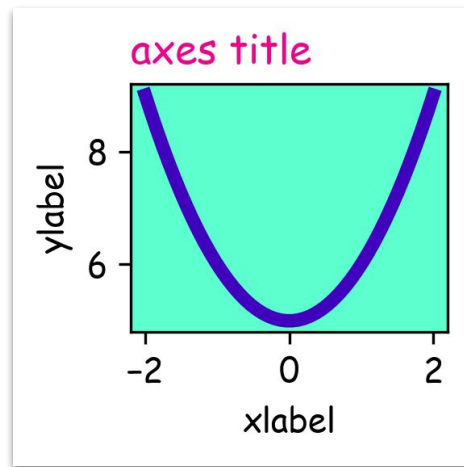


Matplotlib lets you store chart styling in stylesheets. It comes with a set of in-built stylesheets which you can **apply for the whole session** or **in the context of one figure only**.

Stylesheets

```
import matplotlib as mpl
from cycler import cycler

with mpl.rc_context({'lines.linewidth': 4,
                    'axes.prop_cycle': cycler('color', ['#4200bf']),
                    'axes.facecolor': '#5effcf',
                    'axes.titlelocation': 'left',
                    'axes.titlecolor': '#ef028c',
                    'font.family': 'Comic Sans MS'}):
    example_plot()
```



You can also define your own stylesheet as a dictionary or store it in a configuration file.

3: Viz as code

Example data

```
import seaborn as sns
data = sns.load_dataset('penguins')
data.sample(5)
```

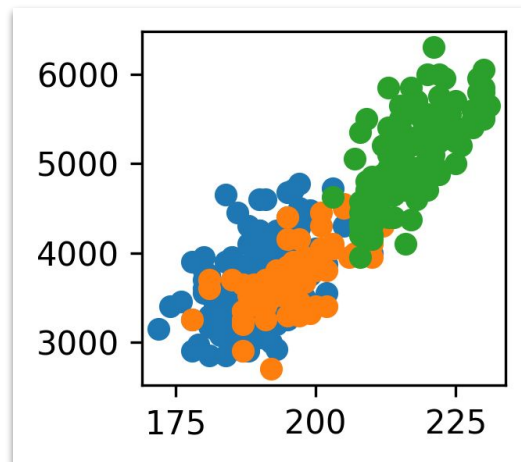
	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
171	Chinstrap	Dream	49.2	18.2	195.0	4400.0	Male
310	Gentoo	Biscoe	47.5	15.0	218.0	4950.0	Female
211	Chinstrap	Dream	45.6	19.4	194.0	3525.0	Female
15	Adelie	Torgersen	36.6	17.8	185.0	3700.0	Female
96	Adelie	Dream	38.1	18.6	190.0	3700.0	Female

Use loops

```
f, ax = plt.subplots(figsize=(2,2))  
  
for chosen in data['species'].unique():  
    tmp_data = data.query('species == @chosen')  
    ax.scatter(x=tmp_data['flipper_length_mm'],  
              y=tmp_data['body_mass_g'])
```

You can **overplot** many plot types in the same axes.
MPL will use a default cycler to change their visual properties.

If you cycle over subsets of your data, don't hardcode it!
Use loops.

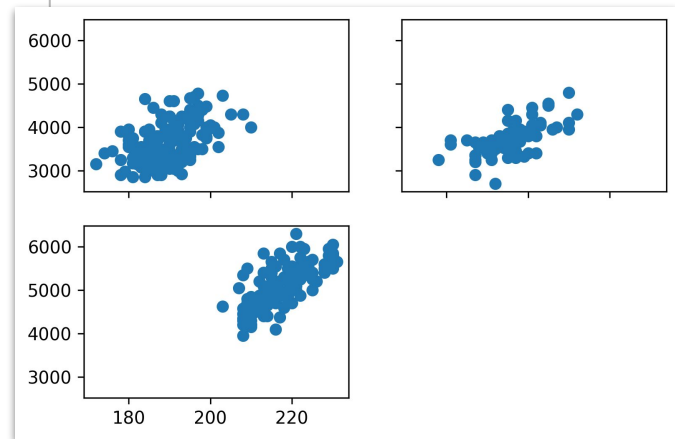


Use zip and zip_longest

```
from itertools import zip_longest

f, axes = plt.subplots(ncols=2, nrows=2,
                      sharex=True, sharey=True,
                      figsize=(6,4), squeeze=False)

for chosen, ax in zip_longest(data['species'].unique(), axes.ravel()):
    if chosen is not None:
        tmp_data = data.query('species == @chosen')
        ax.scatter(
            x=tmp_data['flipper_length_mm'],
            y=tmp_data['body_mass_g'])
    else:
        ax.remove()
```



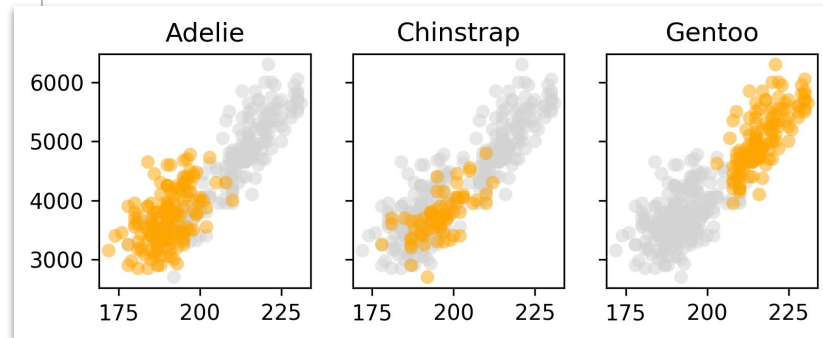
axes returned by `plt.subplots` are a numpy array. To plot a subset of data on each subplot, use loops together with `zip` and/or `zip_longest`.

Use dictionaries for styling

```
f, axes = plt.subplots(ncols=3, nrows=1,
                       sharex=True, sharey=True,
                       figsize=(6,2),)

selected = {'color': 'orange'}
background = {'color': 'lightgray'}
shared_style = {'alpha': 0.5, 's': 40, 'edgecolors': 'none'}

for chosen, ax in zip(data['species'].unique(), axes.ravel()):
    tmp_data = data.query('species == @chosen')
    bg_data = data.query('species != @chosen')
    ax.scatter(x=bg_data['flipper_length_mm'],
              y=bg_data['body_mass_g'],
              **background, **shared_style)
    ax.scatter(x=tmp_data['flipper_length_mm'],
              y=tmp_data['body_mass_g'],
              **selected, **shared_style)
    ax.set_title(chosen)
```



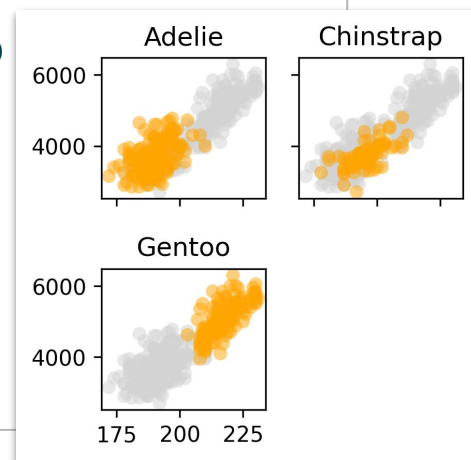
Don't hardcode the style of your charts either!
Make your code reusable and sustainable by
storing such configuration in dictionaries.

Structure your code

```
def plot_single(data, chosen, ax, selected={}, background={}, plot_kws={}):  
    tmp_data, bg_data = data.query('species == @chosen'), data.query('species != @chosen')  
    ax.scatter(x=bg_data['flipper_length_mm'], y=bg_data['body_mass_g'], **background, **plot_kws)  
    ax.scatter(x=tmp_data['flipper_length_mm'], y=tmp_data['body_mass_g'], **selected, **plot_kws)  
    ax.set_title(chosen)
```

```
def dashboard(data, selected, background, dashboard_kws={}, plot_kws={}):  
    f, axes = plt.subplots(ncols=2, nrows=2, sharex=True, sharey=True, dpi=200, **dashboard_kws)  
    for chosen, ax in zip_longest(data['species'].unique(), axes.ravel()):  
        if chosen is not None:  
            plot_single(data=data, chosen=chosen, ax=ax,  
                        selected=selected, background=background, plot_kws=plot_kws)  
        else:  
            ax.remove()  
    return f, axes
```

```
selected = {'color': 'orange'}  
background = {'color': 'lightgray'}  
shared_style = {'alpha': 0.5, 's': 40, 'edgecolors': 'none'}  
f, axes = dashboard(data, selected=selected, background=background,  
                    dashboard_kws={'figsize': (3,3), 'gridspec_kw': {'hspace': 0.5}},  
                    plot_kws=shared_style)
```



*this is not a production-ready code! Watch out for [dangerous default values](#)

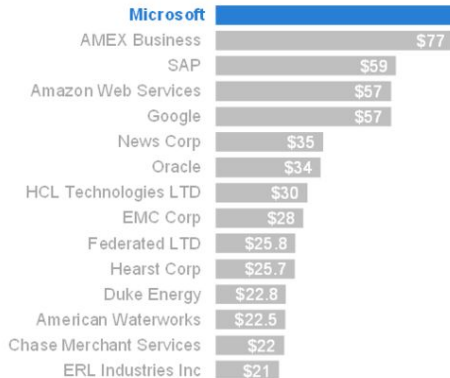
4: How to design better charts

Don't let the software decide for you

- Visualization is communication, and a bar chart != bar chart.
- Never rely on the software defaults to shape your message. Own the design of your visualization and apply it intentionally.

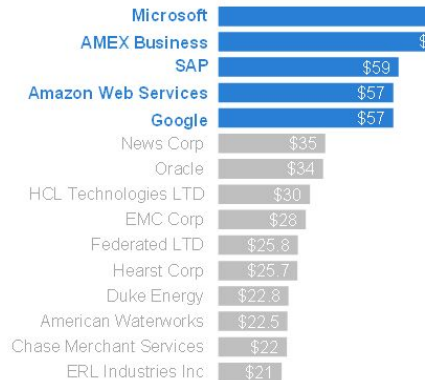
Microsoft is our largest vendor

Accounts payable by vendor (\$K)



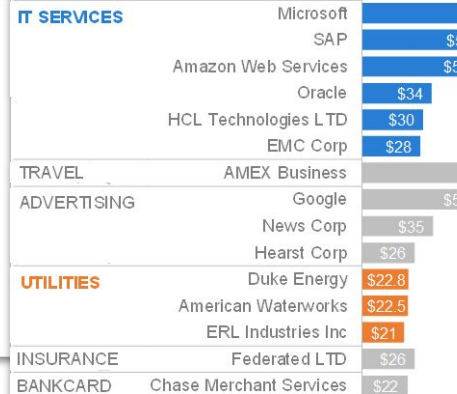
60% of spend comes from five vendors

Accounts payable by vendor (\$K)



Let's discuss implications for Q3 planning

Accounts payable by vendor (\$K)

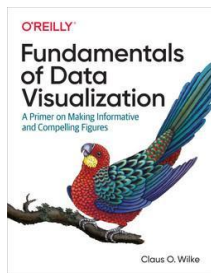


Total IT spend of \$333K is \$3K above plan. Given our needed investment in software upgrades in Q3, let's consider re-prioritizing planned spend across functions.

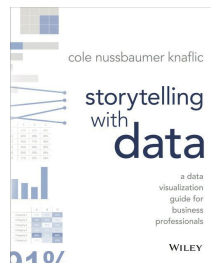
Travel & advertising expenses are in line with YTD expectations, although the distribution amongst advertising vendors needs further ROI analysis.

Utilities spend is \$5K lower than expected due to our Project Clean initiative implemented at the end of 2018. We expect continued success and this category represents an opportunity to reallocate dollars. What are our next steps?

Some resources to get better at dataviz



<https://clauswilke.com/dataviz>



<https://www.storytellingwithdata.com>

Datawrapper

Blog

<https://blog.datawrapper.de>



<https://github.com/rougier/scientific-visualization-book>

paniterka / awesome-matplotlib

<https://github.com/paniterka/awesome-matplotlib>

MPL cheatsheets and handouts

<https://matplotlib.org/cheatsheets/>



Quick start

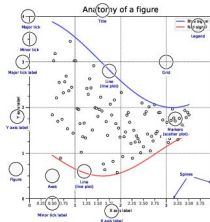
```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
```

```
X = np.linspace(0, 2*np.pi, 100)
Y = np.cos(X)
```

```
fig, ax = plt.subplots()
ax.plot(X, Y, color='green')
```

```
fig.savefig("figure.pdf")
plt.show()
```

Anatomy of a figure



Subplots layout

```
subplot(s) (rows, cols, ...)
fig, axs = plt.subplots(3, 3)
```

```
G = gridspec(rows, cols, ...)
ax = G[0, 1]
```

```
ax.inset_axes(extent)
```

```
d=make_axes_locatable(ax)
ax = d.new_horizontal(*180°)
```

Getting help

- matplotlib.org
- github.com/matplotlib/matplotlib/issues
- discourse.matplotlib.org
- stackoverflow.com/questions/tagged/matplotlib
- https://git.io/matplotlib/matplotlib
- weitz.de/matplotlib
- Matplotlib users mailing list

Basic plots

```
plot(X, Y, [fmt], ...)
% X, Y, limit, color, marker, linestyle
```

```
scatter(X, Y, ...)
X, Y, sizes, colors, marker, cmap
```

```
bar(h)(x, height, ...)
% x, height, width, bottom, align, color
```

```
imshow(Z, ...)
Z, cmap, interpolation, extent, origin
```

```
contour[f](X, Y, Z, ...)
% X, Y, Z, levels, colors, extent, origin
```

```
pcolormesh(X, Y, Z, ...)
% X, Y, Z, vmin, vmax, cmap
```

```
quiver(X, Y, U, V, ...)
% X, Y, U, V, C, units, angles
```

```
pie(x, ...)
Z, explode, labels, colors, radius
```

```
text(x, y, text, ...)
% x, y, text, va, ha, size, weight, transform
```

```
fill_between(x, ...)
% X, Y1, Y2, where, where
```

Advanced plots

```
step(X, Y, [fmt], ...)
% X, Y, limit, color, marker, where
```

```
boxplot(X, ...)
X, notch, sym, bootstrap, widths
```

```
errorbar(X, Y, xerr, yerr, ...)
% X, Y, xerr, yerr, fmt
```

```
hist(X, bins, ...)
% X, bins, bins, density, weights
```

```
violinplot(D, ...)
D, positions, widths, vert
```

```
barbs(X, Y, U, V, ...)
% X, Y, U, V, G, length, pivot, sizes
```

```
eventplot(positions, ...)
positions, orientation, lineoffsets
```

```
hexbin(X, Y, C, ...)
% X, Y, C, gridsize, bins
```

Scales

```
ax.set_xyscale(scale, ...)
% linear, log, logit
```

```
ax.set_yscale(scale, ...)
% linear, log, logit
```

```
ax.set_xscale(scale, ...)
% linear, log, logit
```

```
ax.set_yscale(scale, ...)
% linear, log, logit
```

```
ax.set_xscale(scale, ...)
% linear, log, logit
```

```
ax.set_yscale(scale, ...)
% linear, log, logit
```

```
ax.set_xscale(scale, ...)
% linear, log, logit
```

```
ax.set_yscale(scale, ...)
% linear, log, logit
```

```
ax.set_xscale(scale, ...)
% linear, log, logit
```

```
ax.set_yscale(scale, ...)
% linear, log, logit
```

Projections

```
subplot(..., projection=p)
% 'polar', '3d'
```

```
ax = plt.subplot(projection='polar')
```

```
ax = plt.subplot(projection='3d')
```

```
ax = plt.subplot(projection='polar')
```

```
ax = plt.subplot(projection='3d')
```

```
ax = plt.subplot(projection='polar')
```

```
ax = plt.subplot(projection='3d')
```

```
ax = plt.subplot(projection='polar')
```

```
ax = plt.subplot(projection='3d')
```

Tick locators

```
from matplotlib import ticker
ax.ticker_locator_locator(locator)
```

```
ax.ticker_locator_locator(locator)
```

```
ax.ticker_locator_locator(locator)
```

```
ax.ticker_locator_locator(locator)
```

```
ax.ticker_locator_locator(locator)
```

```
ax.ticker_locator_locator(locator)
```

```
ax.ticker_locator_locator(locator)
```

```
ax.ticker_locator_locator(locator)
```

```
ax.ticker_locator_locator(locator)
```

```
ax.ticker_locator_locator(locator)
```

Tick formatters

```
from matplotlib import ticker
ax.set_xaxis_major_formatter(formatter)
```

```
ax.set_xaxis_minor_formatter(formatter)
```

```
ax.set_yaxis_formatter(formatter)
```

```
ax.set_yaxis_formatter(formatter)
```

```
ax.set_yaxis_formatter(formatter)
```

```
ax.set_yaxis_formatter(formatter)
```

```
ax.set_yaxis_formatter(formatter)
```

```
ax.set_yaxis_formatter(formatter)
```

```
ax.set_yaxis_formatter(formatter)
```

Animation

```
import matplotlib.animation as mpla
```

```
T = np.linspace(0, 2*np.pi, 100)
```

```
S = np.sin(T)
```

```
line = plt.plot(T, S)
```

```
def animate(i):
```

```
    line.set_ydata(np.sin(T+i/50))
```

```
    anim = mpla.FuncAnimation(
```

```
        plt.gcf(), animate, interval=5)
```

```
plt.show()
```

Styles

```
plt.style.use('style')
```

```
plt.style.use('style')
```

```
plt.style.use('style')
```

```
plt.style.use('style')
```

```
plt.style.use('style')
```

```
plt.style.use('style')
```

```
plt.style.use('style')
```

```
plt.style.use('style')
```

```
plt.style.use('style')
```

```
plt.style.use('style')
```

```
plt.style.use('style')
```

Colors

```
ax.set_facecolor('color')
```

```
ax.set_facecolor('color')
```

```
ax.set_facecolor('color')
```

```
ax.set_facecolor('color')
```

```
ax.set_facecolor('color')
```

```
ax.set_facecolor('color')
```

```
ax.set_facecolor('color')
```

```
ax.set_facecolor('color')
```

```
ax.set_facecolor('color')
```

```
ax.set_facecolor('color')
```

Colormaps

```
plt.get_cmap(name)
```

```
plt.get_cmap(name)
```

```
plt.get_cmap(name)
```

```
plt.get_cmap(name)
```

```
plt.get_cmap(name)
```

```
plt.get_cmap(name)
```

```
plt.get_cmap(name)
```

```
plt.get_cmap(name)
```

```
plt.get_cmap(name)
```

```
plt.get_cmap(name)
```

Ornaments

```
ax.legend(...)
```

```
ax.legend(...)
```

```
ax.legend(...)
```

```
ax.legend(...)
```

```
ax.legend(...)
```

```
ax.legend(...)
```

```
ax.legend(...)
```

```
ax.legend(...)
```

```
ax.legend(...)
```

```
ax.legend(...)
```

```
ax.legend(...)
```

ax.colorbar(...)

```
mappable, ax, cax, orientation
```

```
mappable, ax, cax, orientation
```

```
mappable, ax, cax, orientation
```

```
mappable, ax, cax, orientation
```

```
mappable, ax, cax, orientation
```

```
mappable, ax, cax, orientation
```

```
mappable, ax, cax, orientation
```

```
mappable, ax, cax, orientation
```

```
mappable, ax, cax, orientation
```

```
mappable, ax, cax, orientation
```

ax.annotate(...)

```
text, xy, xytext, xycoords, textcoords, arrowprops
```

```
text, xy, xytext, xycoords, textcoords, arrowprops
```

```
text, xy, xytext, xycoords, textcoords, arrowprops
```

```
text, xy, xytext, xycoords, textcoords, arrowprops
```

```
text, xy, xytext, xycoords, textcoords, arrowprops
```

```
text, xy, xytext, xycoords, textcoords, arrowprops
```

```
text, xy, xytext, xycoords, textcoords, arrowprops
```

```
text, xy, xytext, xycoords, textcoords, arrowprops
```

```
text, xy, xytext, xycoords, textcoords, arrowprops
```

```
text, xy, xytext, xycoords, textcoords, arrowprops
```

text

```
text, xy, xytext, xycoords, textcoords, arrowprops
```

```
text, xy, xytext, xycoords, textcoords, arrowprops
```

```
text, xy, xytext, xycoords, textcoords, arrowprops
```

```
text, xy, xytext, xycoords, textcoords, arrowprops
```

```
text, xy, xytext, xycoords, textcoords, arrowprops
```

```
text, xy, xytext, xycoords, textcoords, arrowprops
```

```
text, xy, xytext, xycoords, textcoords, arrowprops
```

```
text, xy, xytext, xycoords, textcoords, arrowprops
```

```
text, xy, xytext, xycoords, textcoords, arrowprops
```

```
text, xy, xytext, xycoords, textcoords, arrowprops
```

Event handling

```
fig, ax = plt.subplots()
```

```
def on_click(event):
```

```
    print(event)
```

```
    print(event)
```

```
fig.canvas.mpl_connect(
```

```
    'button_press_event', on_click)
```

```
fig.canvas.mpl_connect(
```

```
    'button_press_event', on_click)
```

```
fig.canvas.mpl_connect(
```

```
    'button_press_event', on_click)
```

Quick reminder

```
ax.grid()
```

```
ax.set_xlim(vmin, vmax)
```

```
ax.set_ylabel(label)
```

```
ax.set_xticks(ticks, [labels])
```

```
ax.set_yticklabels(labels)
```

```
ax.set_title(title)
```

```
ax.tick_params(width=10, ...)
```

```
ax.set_axis_on/off()
```

```
fig.suptitle(title)
```

```
fig.tight_layout()
```

```
plt.gcf(); plt.gca()
```

```
mpl.rc('axes', linewidth=1, ...)
```

```
[fig, ax].patch.set_alpha(0)
```

```
text = '$\frac{e+h\sqrt{13}}{2}n$'
```

Keyboard shortcuts

```
ctrl+S Save
```

```
ctrl+W Close plot
```

```
r Reset view
```

```
F Fullscreen 0/1
```

```
f View forward
```

```
b View back
```

```
p Pan view
```

```
z Zoom to rect
```

```
x X pan/zoom
```

```
ctrl+W Close plot
```

```
r Reset view
```

```
F Fullscreen 0/1
```

```
f View forward
```

```
b View back
```

```
p Pan view
```

```
z Zoom to rect
```

```
x X pan/zoom
```

```
y Y pan/zoom
```

```
ctrl+W Close plot
```

```
r Reset view
```

```
F Fullscreen 0/1
```

```
f View forward
```

```
b View back
```

```
p Pan view
```

```
z Zoom to rect
```

```
x X pan/zoom
```

```
y Y pan/zoom
```

```
ctrl+W Close plot
```

```
r Reset view
```

```
F Fullscreen 0/1
```

```
f View forward
```

```
b View back
```

```
p Pan view
```

```
z Zoom to rect
```

```
x X pan/zoom
```

```
y Y pan/zoom
```

```
ctrl+W Close plot
```

```
r Reset view
```

```
F Fullscreen 0/1
```

```
f View forward
```

```
b View back
```

```
p Pan view
```

```
z Zoom to rect
```

```
x X pan/zoom
```

```
y Y pan/zoom
```

```
ctrl+W Close plot
```

```
r Reset view
```

```
F Fullscreen 0/1
```

```
f View forward
```

```
b View back
```

```
p Pan view
```

```
z Zoom to rect
```

```
x X pan/zoom
```

```
y Y pan/zoom
```

```
ctrl+W Close plot
```

```
r Reset view
```

```
F Fullscreen 0/1
```

```
f View forward
```

```
b View back
```

```
p Pan view
```

```
z Zoom to rect
```

```
x X pan/zoom
```

```
y Y pan/zoom
```

```
ctrl+W Close plot
```

```
r Reset view
```

```
F Fullscreen 0/1
```

```
f View forward
```

```
b View back
```

```
p Pan view
```

```
z Zoom to rect
```

```
x X pan/zoom
```

```
y Y pan/zoom
```

```
ctrl+W Close plot
```

```
r Reset view
```

```
F Fullscreen 0/1
```

```
f View forward
```

```
b View back
```

```
p Pan view
```

```
z Zoom to rect
```

```
x X pan/zoom
```

```
y Y pan/zoom
```

```
ctrl+W Close plot
```

```
r Reset view
```

```
F Fullscreen 0/1
```

```
f View forward
```

```
b View back
```

```
p Pan view
```

```
z Zoom to rect
```

```
x X pan/zoom
```

```
y Y pan/zoom
```

```
ctrl+W Close plot
```

```
r Reset view
```

```
F Fullscreen 0/1
```

```
f View forward
```

```
b View back
```

```
p Pan view
```

```
z Zoom to rect
```

```
x X pan/zoom
```

```
y Y pan/zoom
```

```
ctrl+W Close plot
```

```
r Reset view
```

```
F Fullscreen 0/1
```

```
f View forward
```

```
b View back
```

```
p Pan view
```

```
z Zoom to rect
```

```
x X pan/zoom
```

```
y Y pan/zoom
```

```
ctrl+W Close plot
```

```
r Reset view
```

Matplotlib needs you!

- Documentation is constantly being improved, and there are many new projects going on
- Sneak peek: <https://matplotlib.org/devdocs/>
- Fresh eyes are always the best for improving the docs! Let me know if you are interested to contribute



Thank you!

Dr. Teresa Kubacka

[@paniterka_ch](#)

www.teresa-kubacka.com

www.pythonviz.blog

linkedin.com/in/tkubacka