A Brief Intro to Matplotlib

Rebecca Pitts

Introducing Matplotlib

Matplotlib is *the* standard Python library for visualizing data for print or digital media.

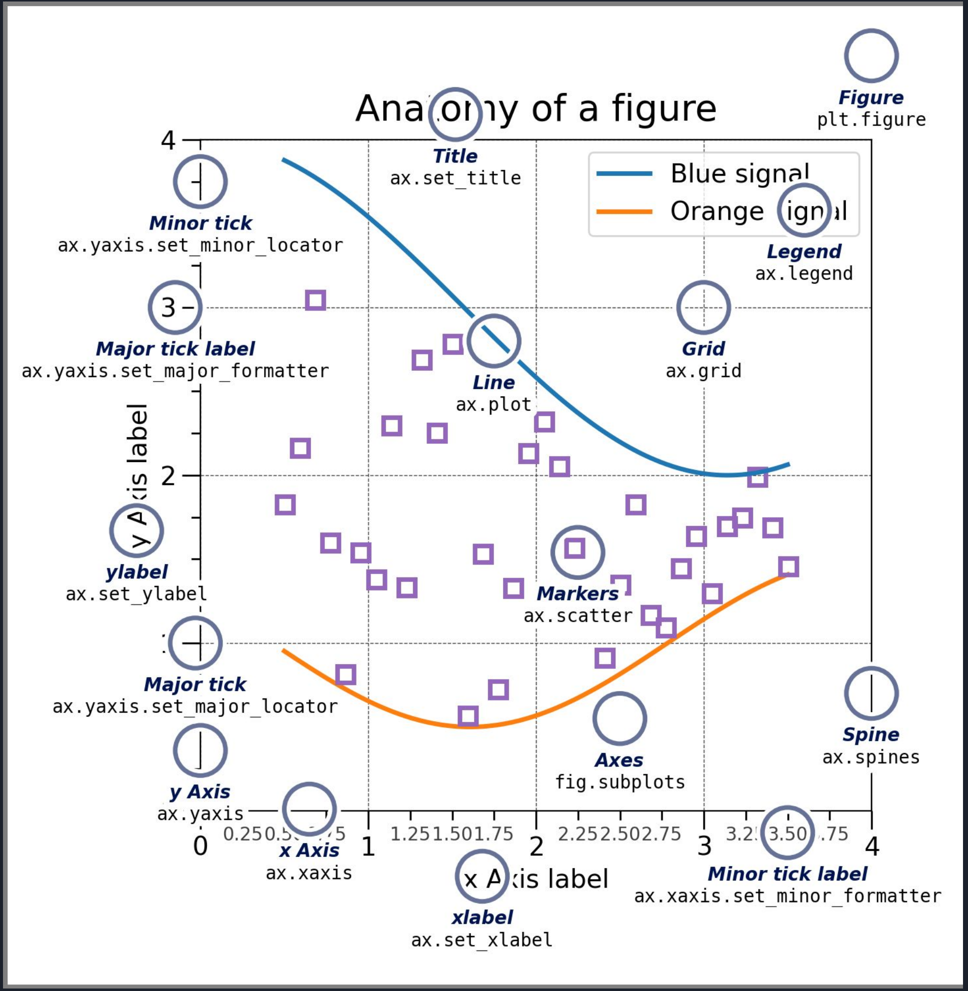
- Figures may be static, interactive, animated, and/or embedded in a Jupyter notebook
- Plots can be 2D or 3D, single or tiled as part of a larger figure
- Cartesian, polar, and many other coordinate systems supported, including map projections and sky coordinates
- Excellent documentation with many, MANY tutorials

Basic Terms

Standard terminology for elements of a plot →

One important clarification:

- **Figure (fig)** = the frame & everything in it (which could include multiple sets of axes)
- **Axis instance (ax)** = one pair (or cube) of axes, their labels, & all data enclosed



More definitions to know before starting

When you see a Python function described in documentation like:

`module.fxn_name(*args, **kwargs)`

Need to know:

- args = positional arguments; usually mandatory
- kwargs = keyword arguments; usually optional

You should also know what classes, methods, & attributes are →

Classes are templates to make Python objects, with methods & attributes

Methods associate functions with the class & allow quick evaluation for each class instance. **Syntax:** `obj.method()` or `obj.method(*args, **kwargs)`

Attributes let you automatically compute & store values that can be derived for any instance of the class. **Syntax:** `obj.attribute`

Pyplot: the Workhorse

The minimum working matplotlib code will virtually always require you to import matplotlib.pyplot (and usually also NumPy)

- Provides MATLAB-like functions to create or alter plots and their components
- In older code you may see `matplotlib.pylab`, which is now deprecated and risks overwriting some built-in functions.

Standard call: `from matplotlib import pyplot as plt`

Minimal working examples

Single data series, no typesetting (implicit application programming interface, or API):

```
from matplotlib import pyplot as plt

plt.plot( x, y, 'k-' )
plt.xlabel('X')
plt.ylabel('Y')
plt.show()
```

Format string may include single-letter color specifier & linestyle specifier (‘-’, ‘:’, ‘-.’, ‘--’). Will show more options later.

Any data you want to add to, typeset, or include as a subplot (explicit API):

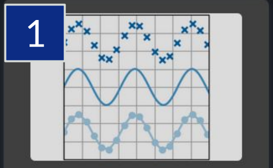
```
from matplotlib import pyplot as plt

fig, ax = plt.subplots(**kwargs)
ax.plot( x, y, 'k-' )
ax.set_xlabel('X')
ax.set_ylabel('Y')
plt.show()
```

Formatting commands for axis objects typically have `set_` in front

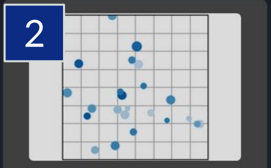
Plot types available through Matplotlib 1: Pairwise data

1. .plot(x1, y1, *fmt1, label=label1, x2, y2, fmt2, label=label2, …*)
2. .scatter(x, y, *[size, color]*)
3. .bar(x, y) or .barh(x,y)
4. .stem(x, y)
5. .fill_between(x, y1, *y2=0, color=‘tab:blue’, alpha=1*)
6. .stackplot(x, y, *baseline=0*)
7. .stairs(y, *edges=[x[0]]+x*)
8. .step(x, y, *where=‘pre’*)



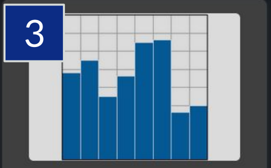
1

plot(x, y)



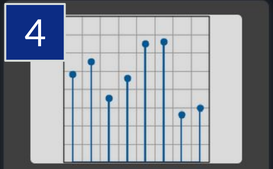
2

scatter(x, y)



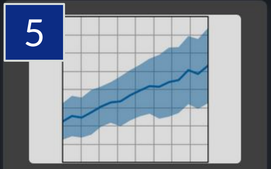
3

bar(x, height)



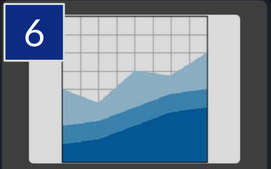
4

stem(x, y)



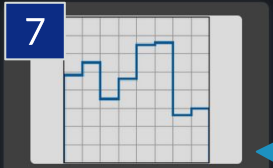
5

fill_between(x, y1, y2)



6

stackplot(x, y)



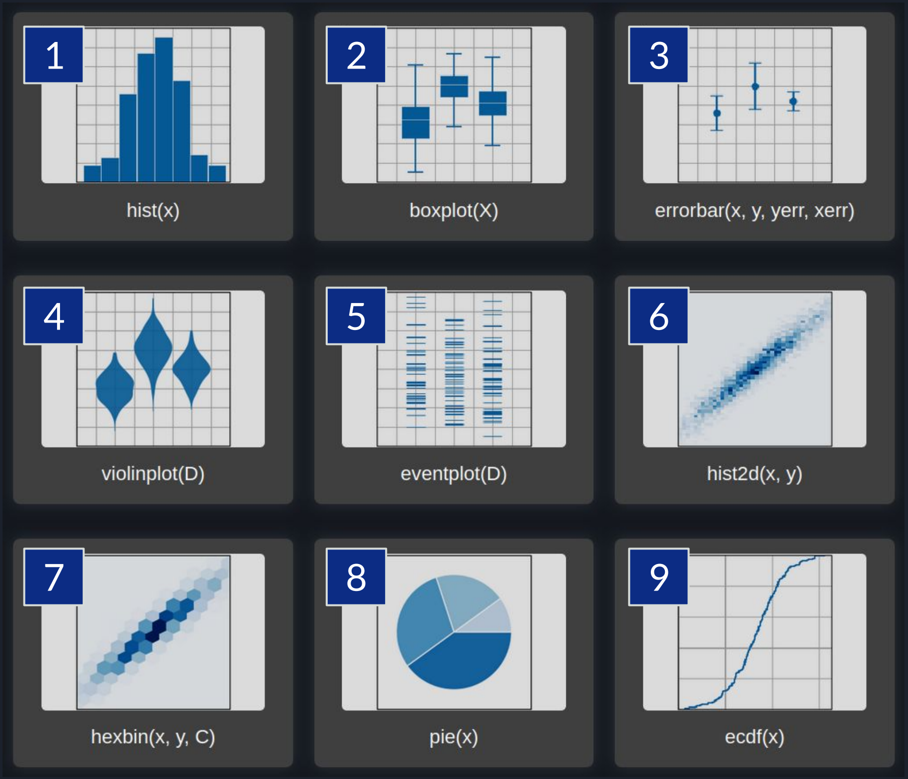
7

stairs(values)

step() is like stairs() but you can set whether each step starts (pre), is centered (mid), or ends (post) on x

Plot types available through Matplotlib 2: Statistical data

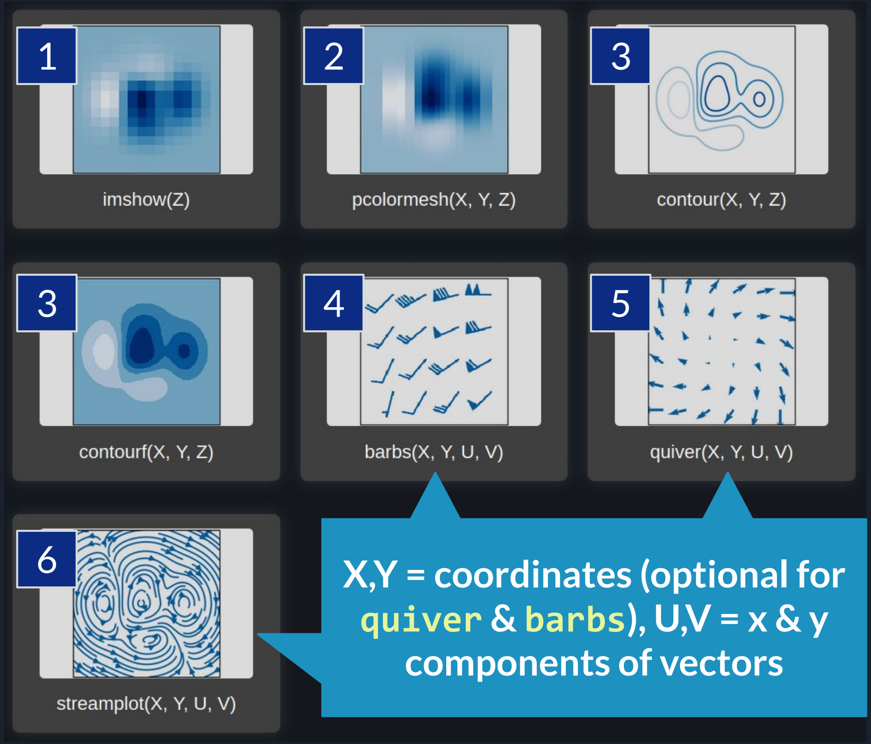
1. `.hist(x, bins=10)`
2. `.boxplot(X)` (X is array-like)
3. `.errorbar(x, y, xerr, yerr)`
4. `.violinplot(X)` (X is array-like)
5. `.eventplot(X)` (X is array-like)
6. `.hist2d(x, y, bins=100)`
7. `.hexbin(x, y, C=None)` (C is 2D)
8. `.pie(wedges)` (avoid)
9. `.ecdf(x)` (ecdf = empirical cumulative distribution function)



Plot types available through Matplotlib 3: Gridded data

These require `X,Y=np.meshgrid(x,y)`

1. `.imshow(C)` (C is either 2D, an *M*×*N*×3 stack of RGB values, or an *M*×*N*×4 stack of RGBA values)
2. `.pcolormesh(X, Y, C)` (like `imshow` but allows non-rectangular pixels)
   a. `.pcolor(X, Y, C)` (only use to mask coordinates instead of C-values)
3. `.contour[f](X, Y, Z)`
4. `.barbs([X, Y,] U, V, [C])`
5. `.quiver([X, Y,] U, V, [C])`
6. `.streamplot(X, Y, U, V)`

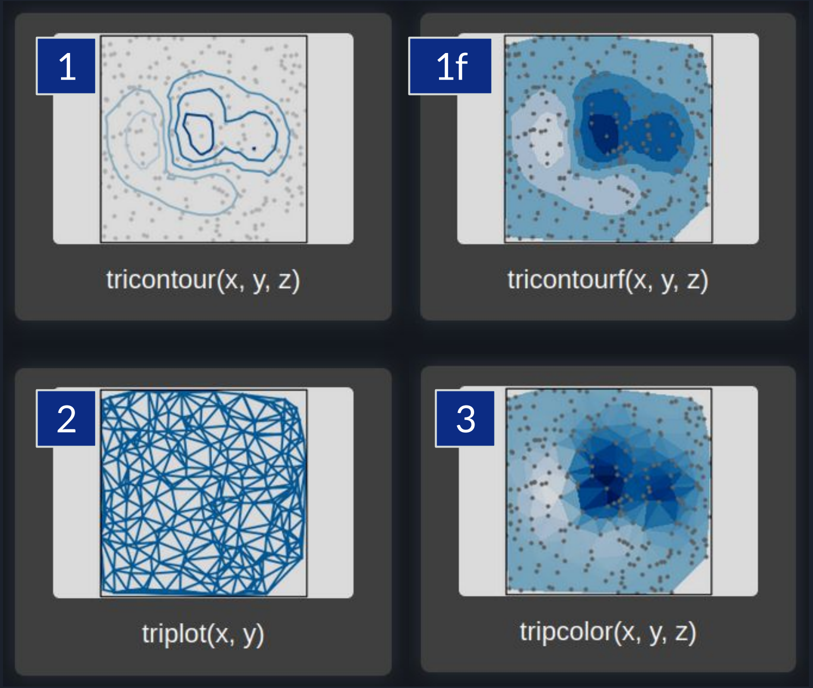


X,Y = coordinates (optional for `quiver` & `barbs`), U,V = x & y components of vectors

Plot types available through Matplotlib 4: Irregularly-gridded data

1. `.tricontour[f](Triangulation, z)` or `.tricontour[f](x, y, z)`
2. `.triplot(Triangulation)` or `.triplot(x, y)`
3. `.tripcolor(Triangulation, c)` or `.tripcolor(x, y, c)`

`mpl.tri.Triangulation(x, y, triangles=None)`: computes Delaunay triangles from x, y vertex coordinates, or takes array of 3-tuples to specify triangle sides from *indexes* of x & y in anticlockwise order.



tricontour(x, y, z)

tricontourf(x, y, z)

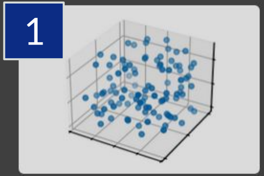
triplot(x, y)

tripcolor(x, y, z)

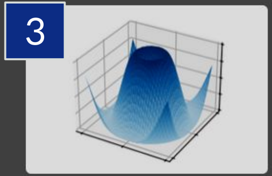
Plot types available through Matplotlib 5: 3D & volumetric data

All functions below, & others with 3D capability, must be plotted on a figure with
`fig, ax = plt.subplots(subplot_kw = {"projection": "3d"})`

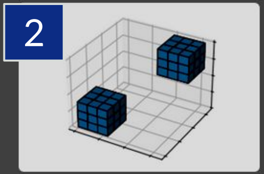
1. `.scatter(x, y, z)`*
2. `.voxels([x, y, z], filled)` (`filled` is a 3D boolean mask)
3. `.plot_surface(X, Y, Z)` (X, Y, & Z are computed with np.meshgrid())
4. `.plot_wireframe(X, Y, Z)` (X, Y, & Z are computed with np.meshgrid())
5. `.plot_trisurf(x, y, z)`



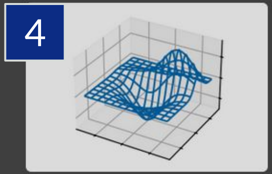
scatter(xs, ys, zs)



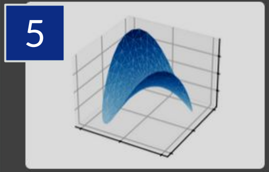
plot_surface(X, Y, Z)



voxels([x, y, z], filled)



plot_wireframe(X, Y, Z)



plot_trisurf(x, y, z)

*Many pairwise functions take a 3rd parameter: `plot`, `stem`, `errorbar`, ...

Labels, legends, & titles

Set x & y axis labels & plot title:

```
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_title('[Sub]plot Title')
```

for the explicit API, or

```
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Plot Title')
```

for the implicit API.

To title a figure with subplots (always explicit API), use `fig.suptitle()`

Most plotting functions include a `label` kwarg to pass to `ax.legend()`.

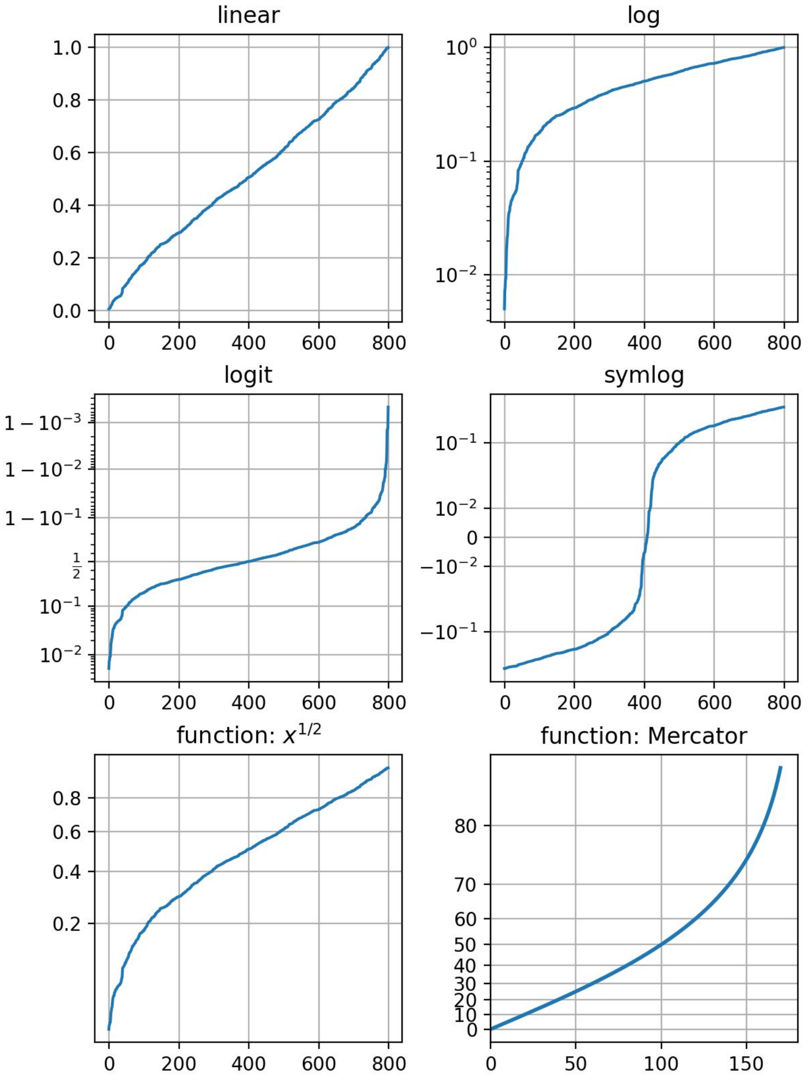
- Control legend position with `loc` (in plot area) or `bbox_to_anchor` (in or out of plot area; more exact)
- Some plotting functions (e.g. `.bar()`) take array of labels as 1st arg instead of `label` kwarg
- Can explicitly pass lines/data handles & labels for e.g. multiple sets of contours

Axis scales

Some plotting functions have `xscale` & `yscale` kwargs. For others, you can set `ax.set_xscale()/ax.set_yscale()`:

- Str-type arg can be any of the scales at right; `‘linear’` is the default
- If `‘function’`, must define both forward & reverse functions for transforming to/from linear & pass them as tuple of function names

Usually automatic tick spacing is fine. I’ll show later what to do if it’s not.



Subplots

Axes created with `plt.subplots()` are iterable if `nrows` &/or `ncols` are >1. These can be set to share x & y axes labels, & `.subplots_adjust()` can adjust or remove column/row spacing:

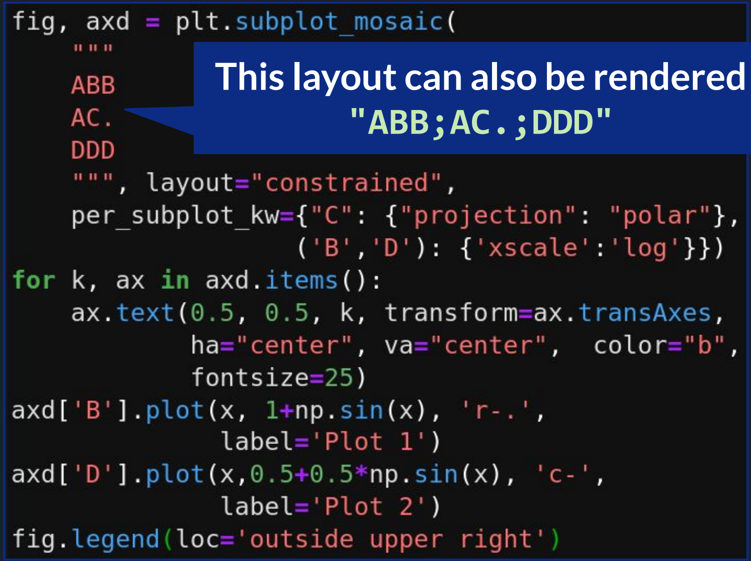
```
fig, axes = plt.subplots(nrows=3, ncols=2,\
sharex=True, sharey=True)
plt.subplots_adjust(hspace=0, wspace=0)
for i in range(3):
    for j in range(2):
        k = (i%3)*2+j
        axes[i][j]=plt.plot(x,y[k])
        if j==0:
            axes[i][j].set_ylabel('Y')
    if i==2:
        axes[i][j].set_xlabel('X')
```

`plt.subplot()` is more tedious but allows separate projections for each plot. Example:

```
fig = plt.figure(figsize=(8,4))
ax1 = plt.subplot(121)
ax1.plot(x, 3+3*np.sin(x), 'b-')
ax1.set_xlabel('x [rads]')
ax1.set_ylabel('y')
ax2 = plt.subplot(122, \
projection= 'polar')
ax2.plot(x, 3+3*np.sin(x), 'b-')
```

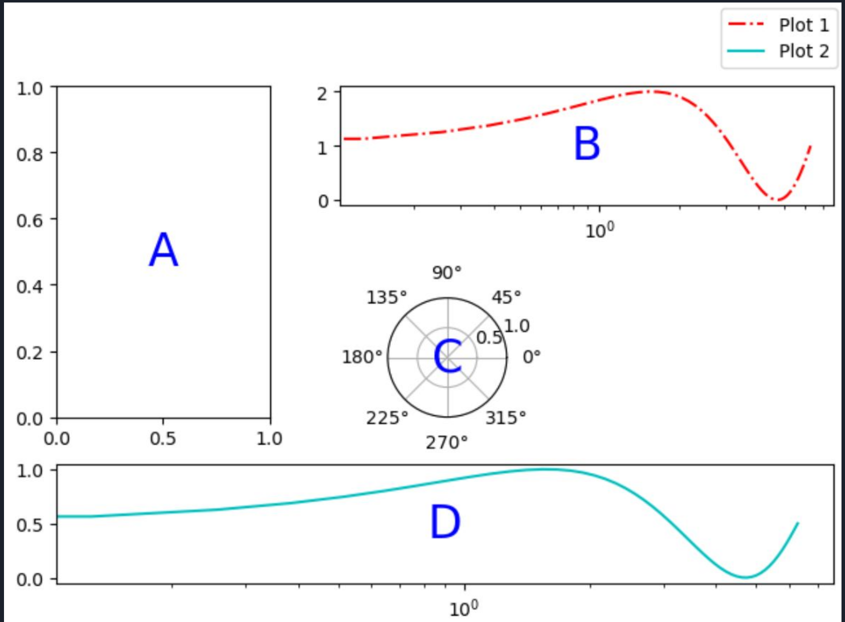
Subplot Mosaics

The API for subplot mosaics lets you lay out subplots so some plots span multiple rows or columns (`"."` denotes gaps). Example:



```
fig, axd = plt.subplot_mosaic(
    """
    ABB
    AC.
    DDD
    """, layout="constrained",
    per_subplot_kw={"C": {"projection": "polar"},
                   ('B','D'): {'xscale':'log'}})
for k, ax in axd.items():
    ax.text(0.5, 0.5, k, transform=ax.transAxes,
            ha="center", va="center",  color="b",
            fontsize=25)
axd['B'].plot(x, 1+np.sin(x), 'r-.',
              label='Plot 1')
axd['D'].plot(x,0.5+0.5*np.sin(x), 'c-',
              label='Plot 2')
fig.legend(loc='outside upper right')
```

This layout can also be rendered `"ABB;AC.;DDD"`



Placement of text, legends, etc.

2 functions for adding text to plots at arbitrary points: `.annotate()` & `.text()`

- `.text()` is base function; it only adds & formats text (e.g. `ha` & `va` set horizontal & vertical alignment)
- `.annotate()` adds kwargs to format connectors between points & text; coordinates for point & text are specified separately
- Positions for both are in data coordinates unless one includes `transform=ax.transAxes`

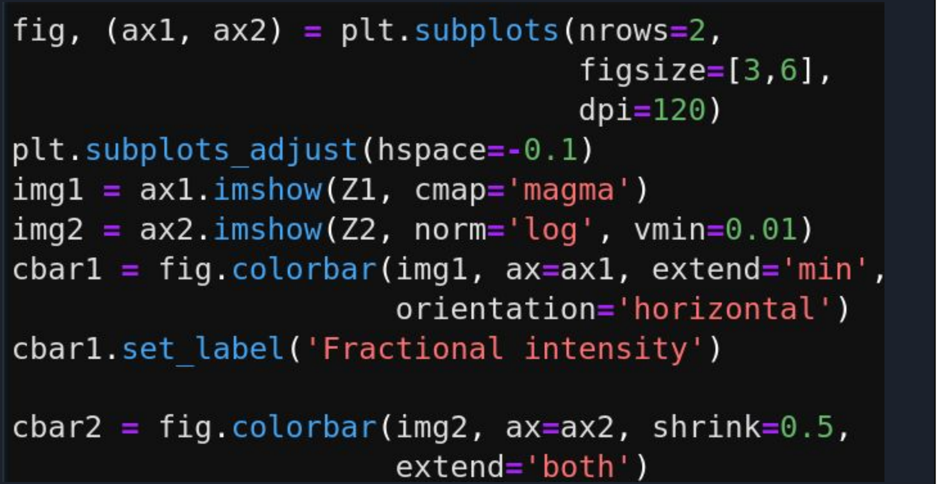
`ax.transAxes` switches from data coordinates to axes-relative coordinates where x & y axes are length-1 & (0,0) is lower left corner.

Legend placement via `bbox_to_anchor` uses unit-axes coordinates by default, & can specify any coordinates on or off plot area.

- Legend position `loc` can be set to integer or 2-word string like `'lower left'` or `'upper center'`
- Whole-figure legends can use `loc='outside <va> <ha>'`

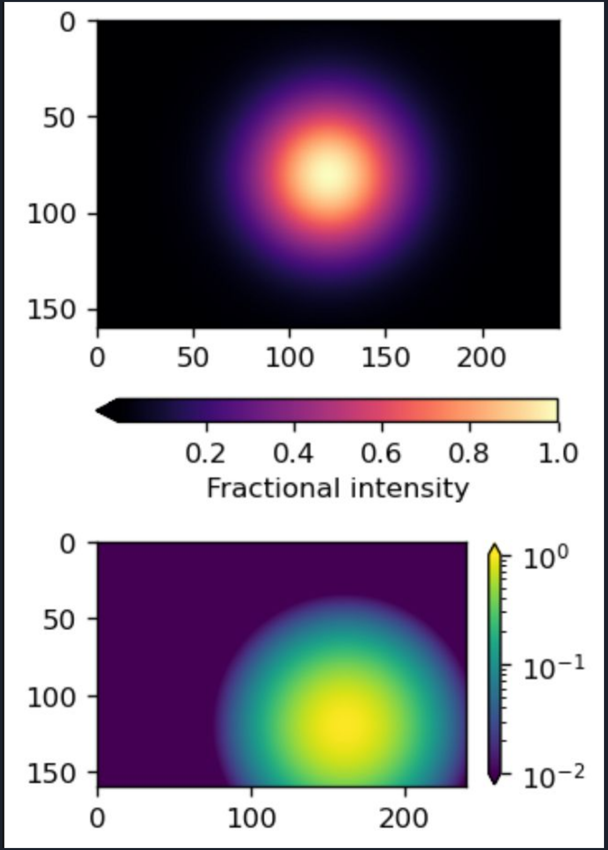
Colorbars

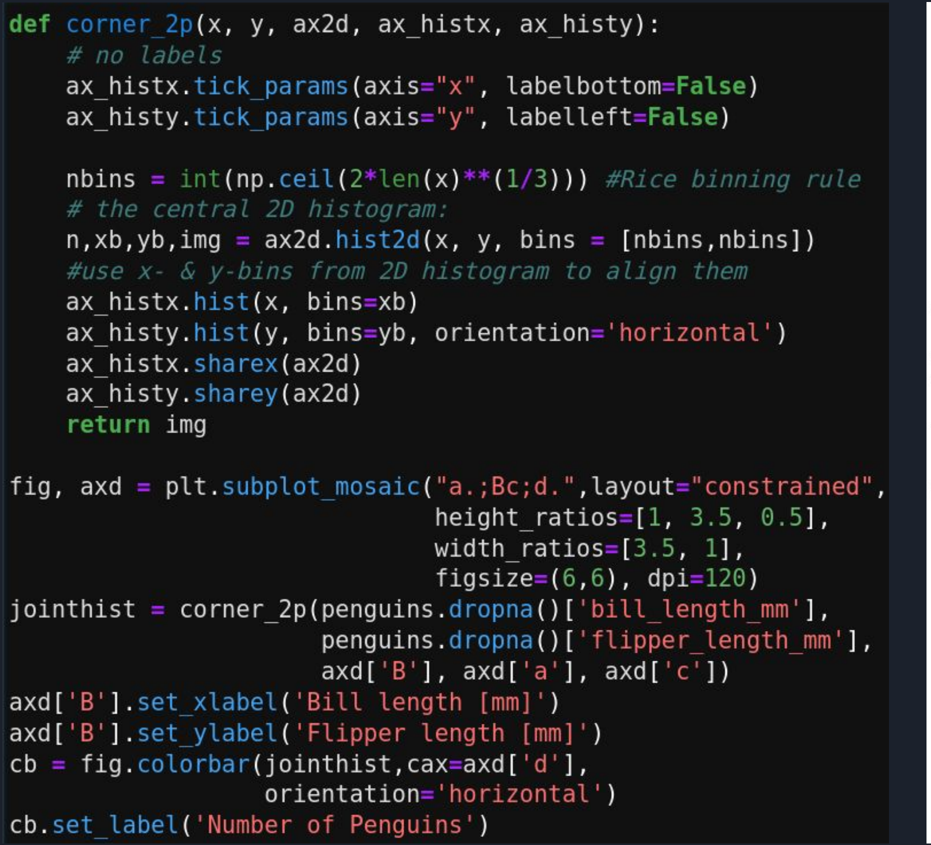
Colorbars are methods of `Figure`, not `Axes`, in the explicit API. Each axis object must be passed to each colorbar command explicitly, & the first arg must be a mappable (check docs). Example:



```
fig, (ax1, ax2) = plt.subplots(nrows=2,
                               figsize=[3,6],
                               dpi=120)
plt.subplots_adjust(hspace=-0.1)
img1 = ax1.imshow(Z1, cmap='magma')
img2 = ax2.imshow(Z2, norm='log', vmin=0.01)
cbar1 = fig.colorbar(img1, ax=ax1, extend='min',
                     orientation='horizontal')
cbar1.set_label('Fractional intensity')

cbar2 = fig.colorbar(img2, ax=ax2, shrink=0.5,
                     extend='both')
```

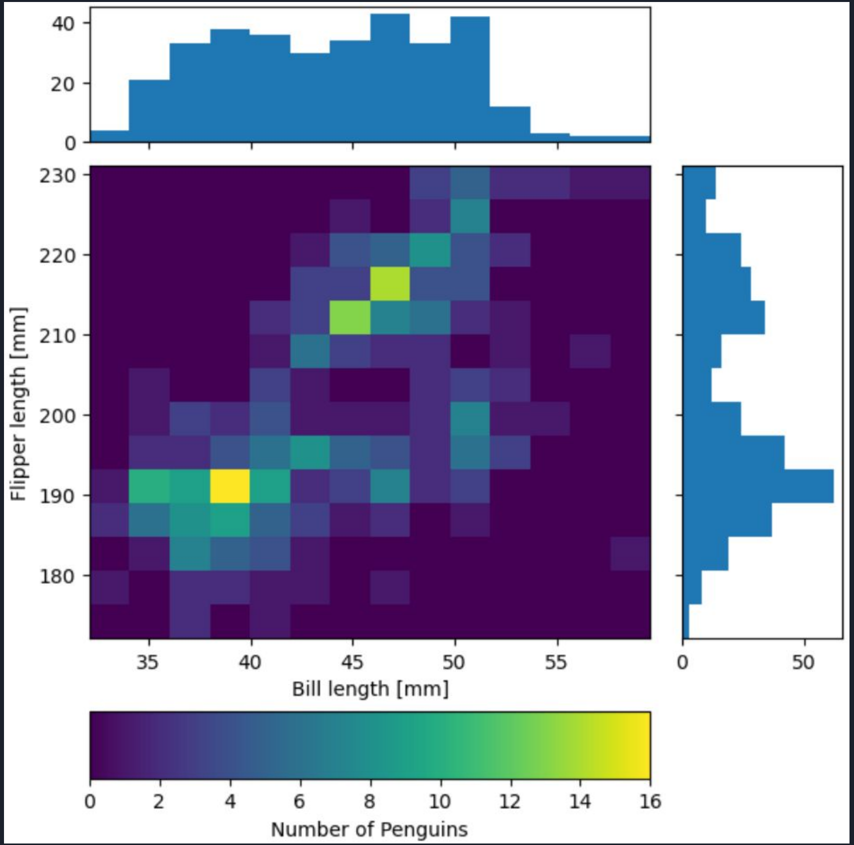




```python
def corner_2p(x, y, ax2d, ax_histx, ax_histy):
    # no labels
    ax_histx.tick_params(axis="x", labelbottom=False)
    ax_histy.tick_params(axis="y", labelleft=False)

    nbins = int(np.ceil(2*len(x)**(1/3))) #Rice binning rule
    # the central 2D histogram:
    n,xb,yb,img = ax2d.hist2d(x, y, bins = [nbins,nbins])
    #use x- & y-bins from 2D histogram to align them
    ax_histx.hist(x, bins=xb)
    ax_histy.hist(y, bins=yb, orientation='horizontal')
    ax_histx.sharex(ax2d)
    ax_histy.sharey(ax2d)
    return img

fig, axd = plt.subplot_mosaic("a.;Bc;d.",layout="constrained",
                              height_ratios=[1, 3.5, 0.5],
                              width_ratios=[3.5, 1],
                              figsize=(6,6), dpi=120)
jointhist = corner_2p(penguins.dropna()['bill_length_mm'],
                      penguins.dropna()['flipper_length_mm'],
                      axd['B'], axd['a'], axd['c'])
axd['B'].set_xlabel('Bill length [mm]')
axd['B'].set_ylabel('Flipper length [mm]')
cb = fig.colorbar(jointhist,cax=axd['d'],
                  orientation='horizontal')
cb.set_label('Number of Penguins')
```



Example: 1- & 2D histograms on a scaled subplot mosaic with a separate colorbar axis. Note that `hist2d()` returns 3 other parameters before the mappable.

Animated Plots

`import matplotlib.animation`

2 main functions:

- `FuncAnimation()`: make data for 1st frame, update data for subsequent frames
- `ArtistAnimation()`: make list (iterable) of artists (plots, shapes, etc.) to be drawn in each frame.

5 save options: