Programming and Data Science for Biology (PDSB)

Session 11
Spring 2018

Today’s topics

1. **Review of last assignment (pymc3)**
2. **Plotting guide**
3. **Plotting standard: matplotlib**
4. **Plotting for the web: toyplot / bokeh**
5. **Servers, apps,**

Upcoming topics relate to projects

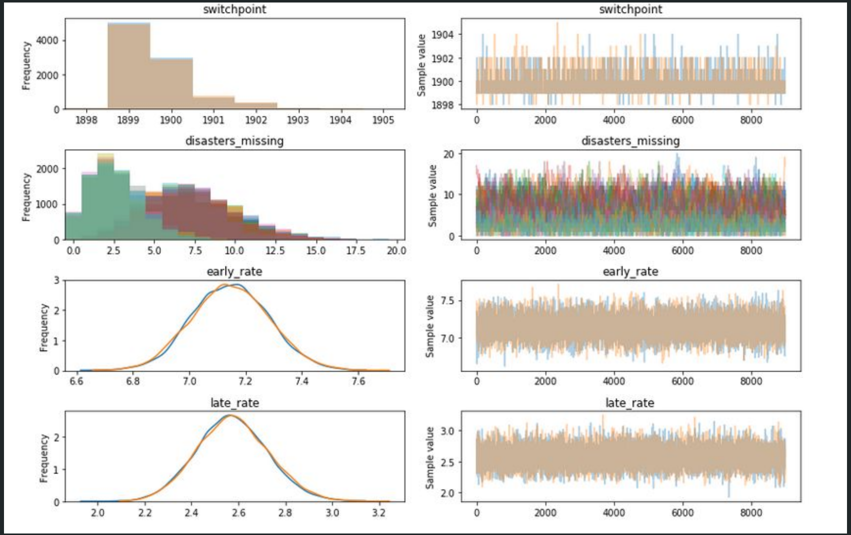
1. **Today:** **Plotting methods: matplotlib, toyplot, folium, bokeh**
2. **Next week:** **Genomic analysis tools & example presentation.**
3. **Two weeks:** **Genomics and beginning Presentations**

Last week’s assignment:
Applying Bayesian estimation to a simulated data set

Assignment

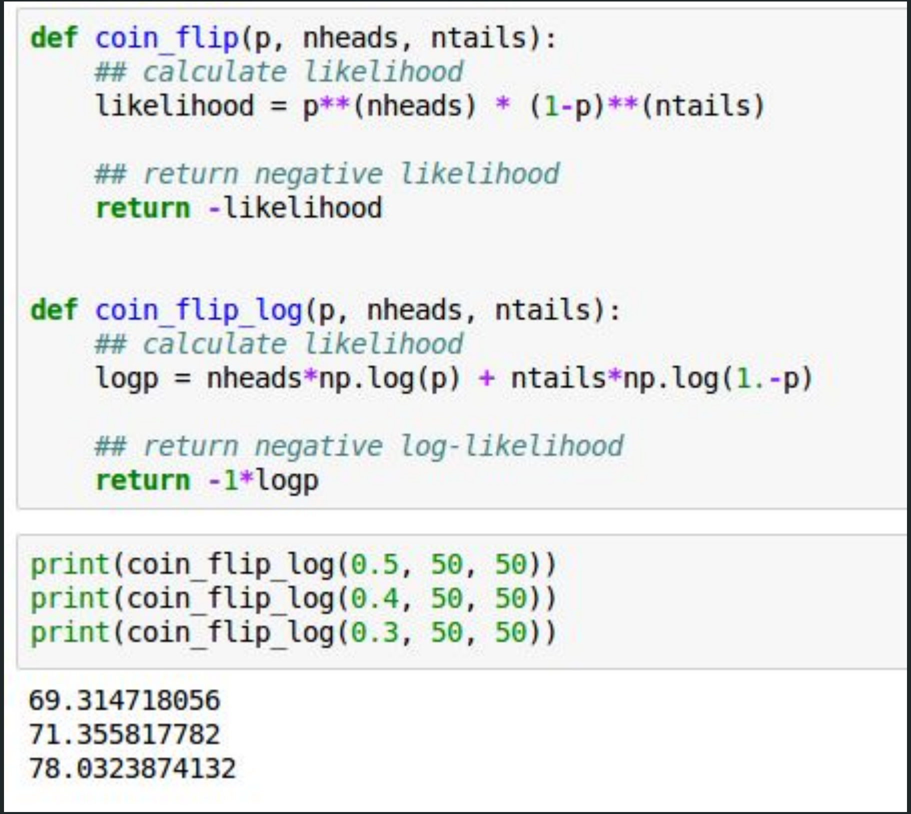
Your assignment was to apply a model from pymc3 for Bayesian posterior parameter estimation.

- You should have generated data using random sampling functions
- You should defined a model that uses similar assumptions about the distribution of parameters
- You should have fit the model and discussed the goodness of fit.



Probability and Likelihood

Likelihood:
A likelihood function is a function of the parameters of a statistical model given data. In other words, it **returns** a value that depends on how well the data fit the model with the given parameters. It is typically used to infer the best set of parameters for a model.



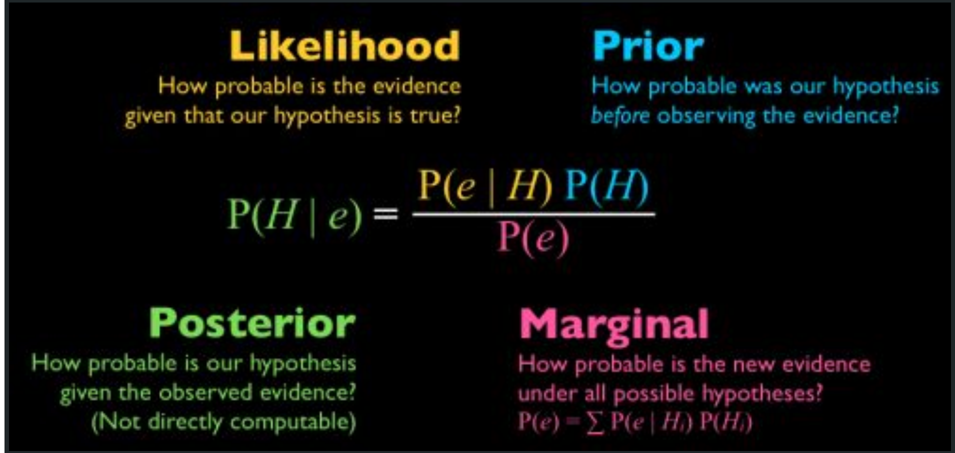
```python
def coin_flip(p, nheads, ntails):
    ## calculate likelihood
    likelihood = p**(nheads) * (1-p)**(ntails)

    ## return negative likelihood
    return -likelihood


def coin_flip_log(p, nheads, ntails):
    ## calculate likelihood
    logp = nheads*np.log(p) + ntails*np.log(1.-p)

    ## return negative log-likelihood
    return -1*logp
```

```python
print(coin_flip_log(0.5, 50, 50))
print(coin_flip_log(0.4, 50, 50))
print(coin_flip_log(0.3, 50, 50))
```

```
69.314718056
71.355817782
78.0323874132
```

The philosophy of Bayesian Inference

- We define priors on our beliefs
- We define a likelihood function that is maximized when data fits our model.
- We sample data using MCMC (a simulation optimization method) to search over all possible hypotheses (*marginalize).*
- We calculate the posterior (updated beliefs) as likelihood x priors.

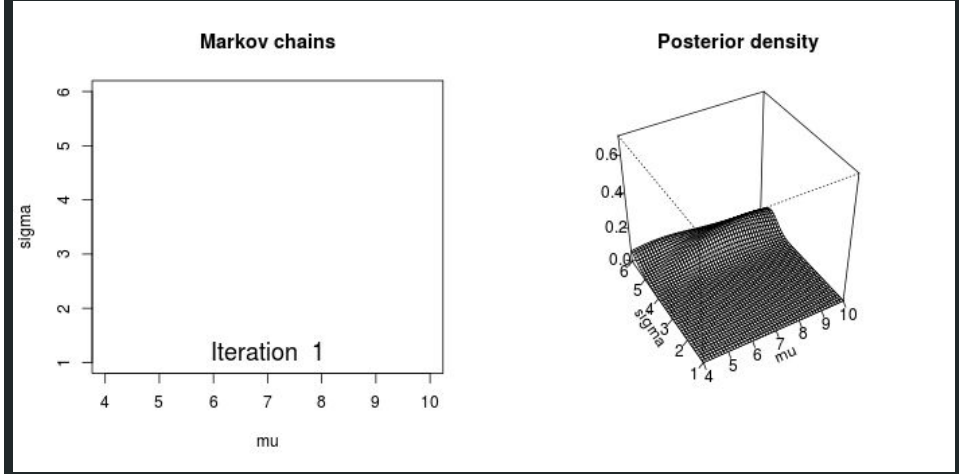


Likelihood
How probable is the evidence given that our hypothesis is true?
Prior
How probable was our hypothesis *before* observing the evidence?
$P(H \mid e) = \frac{P(e \mid H)\, P(H)}{P(e)}$
Posterior
How probable is our hypothesis given the observed evidence? (Not directly computable)
Marginal
How probable is the new evidence under all possible hypotheses?
$P(e) = \sum P(e \mid H_i)\, P(H_i)$

The philosophy of Bayesian Inference

Markov chain Monte Carlo:

Marginalize over probabilities using *importance sampling.* The amount of time spent in any given part of parameter space is equal to its probability, under certain assumptions.

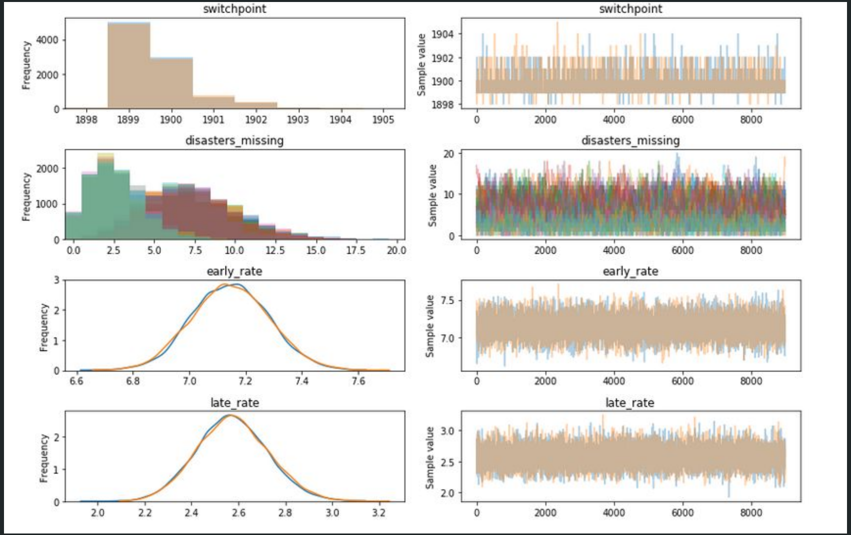


Using simulated data

Learning statistics and statistical methods is hard, takes a long time, and the details are often over our head. The best way to **learn** and **apply** statistical methods is to:

1. Look for applications where others have analyzed similar types of data, or answered similar types of questions, and try to understand their implementations.

2. Simulate data to look like your results under a known generative model, to **validate** that the method is working how you think it is.



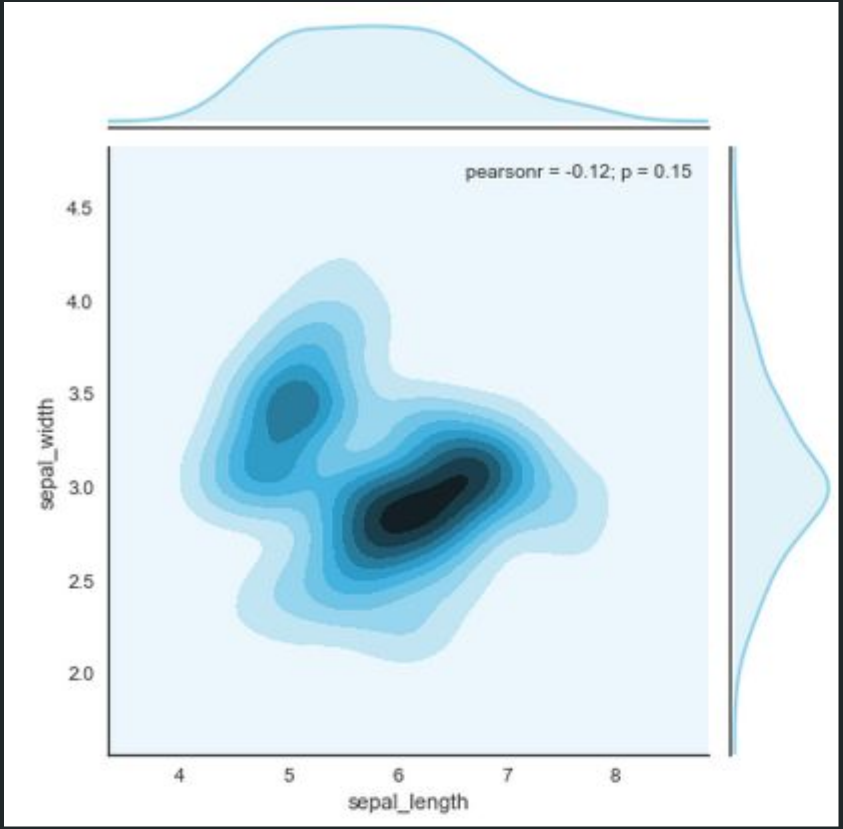
Plotting tools: why, how, and which?

Why to graph?

Plots/graphs are one of the most important and effective tools for exploring data, and for communicating science.

A picture is worth a thousand words.

And a picture can be worth a thousand analyses.

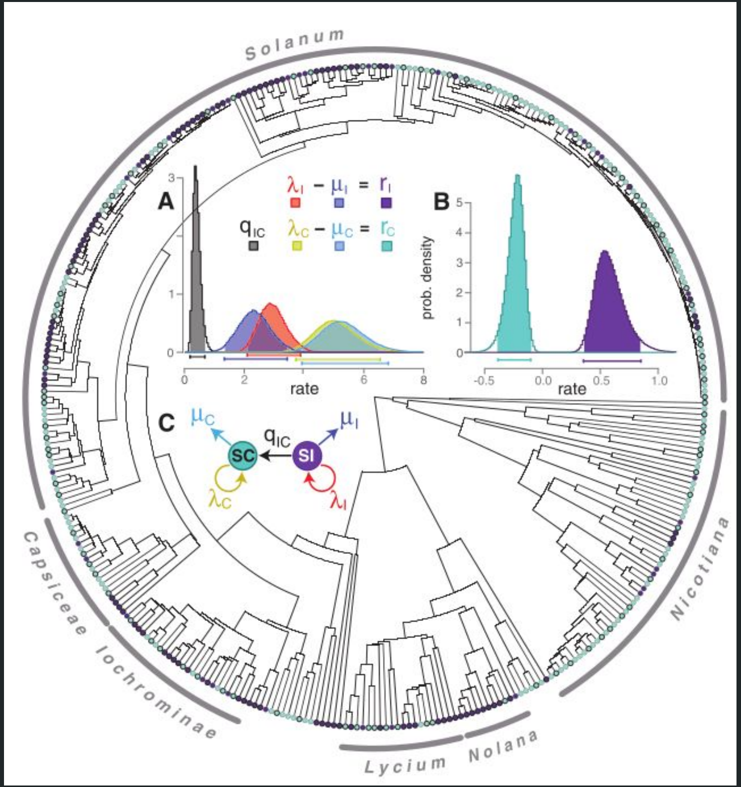


Bokeh docs

Graph types

There are three main reasons for plotting:

1. Exploratory analyses: display raw data, look for outliers, striking patterns.
2. Summary plots: examine model fit, distributions, simulations, means.
3. Final results: a communication tool. A clear presentation that is instantly interpretable.



Igic et al. 2010

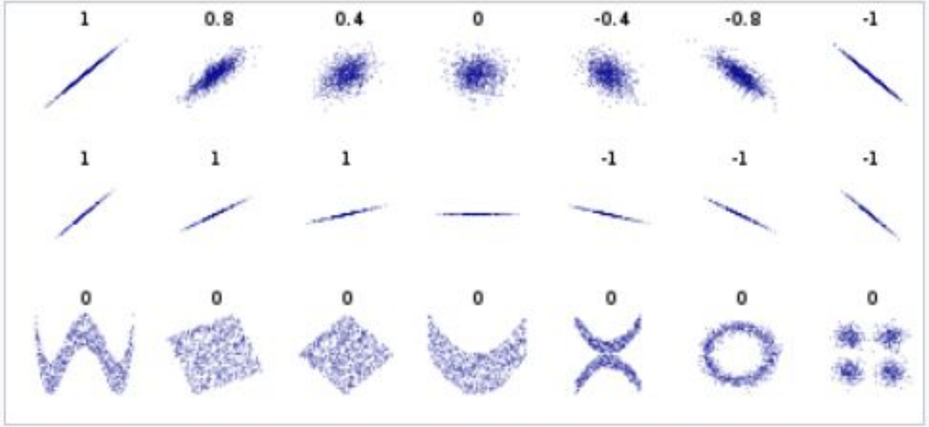
Graph types

There are three main reasons for plotting:

1. **Exploratory analyses**: display raw data, look for outliers, striking patterns.

Bivariate scatterplots are a fundamental tool for exploring the relationship among variables.

Without viewing the data, a measure of correlation alone **can be limiting,** or even deceiving.



Several sets of (*x*, *y*) points, with the Pearson correlation coefficient of *x* and *y* for each set. Note that the correlation reflects the noisiness and direction of a linear relationship (top row), but not the slope of that relationship (middle), nor many aspects of nonlinear relationships (bottom). N.B.: the figure in the center has a slope of 0 but in that case the correlation coefficient is undefined because the variance of *Y* is zero.

Wikipedia

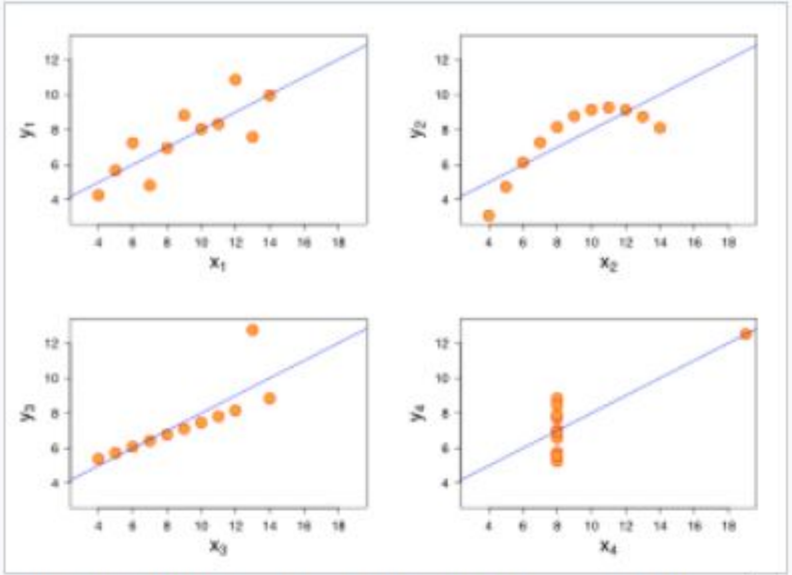
Graph types

There are three main reasons for plotting:

1. **Exploratory analyses**: display raw data, look for outliers, striking patterns.

Bivariate scatterplots are a fundamental tool for exploring the relationship among variables.

Without viewing the data, a measure of correlation alone can be limiting, **or even deceiving.**



Four sets of data with the same correlation of 0.816

Wikipedia

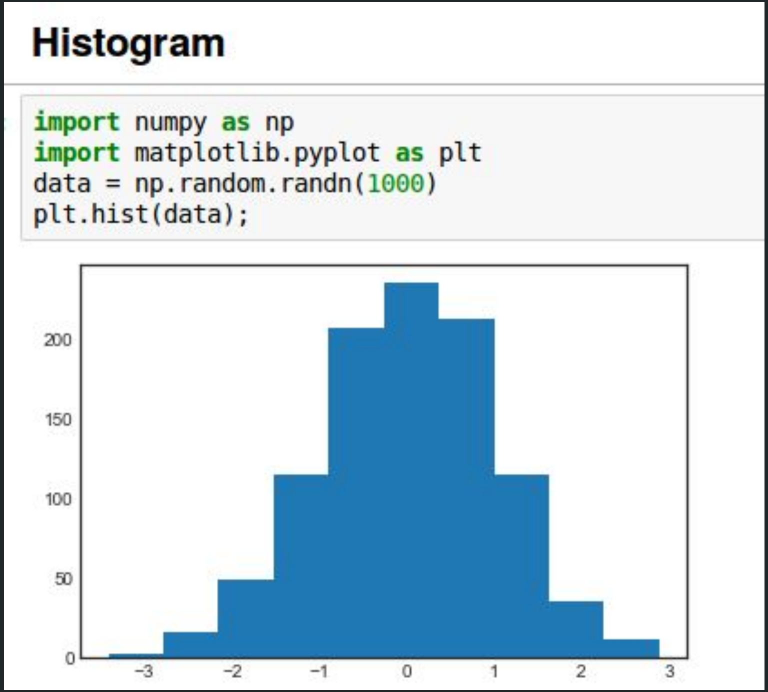
Graph types

There are three main reasons for plotting:

1. **Exploratory analyses**: display raw data, look for outliers, striking patterns.

Univariate histograms are a fundamental tool for exploring the distribution of a variable

Is the distribution centered, is it skewed, should you transform the data (e.g., log)?



Histogram

```
import numpy as np
import matplotlib.pyplot as plt
data = np.random.randn(1000)
plt.hist(data);
```

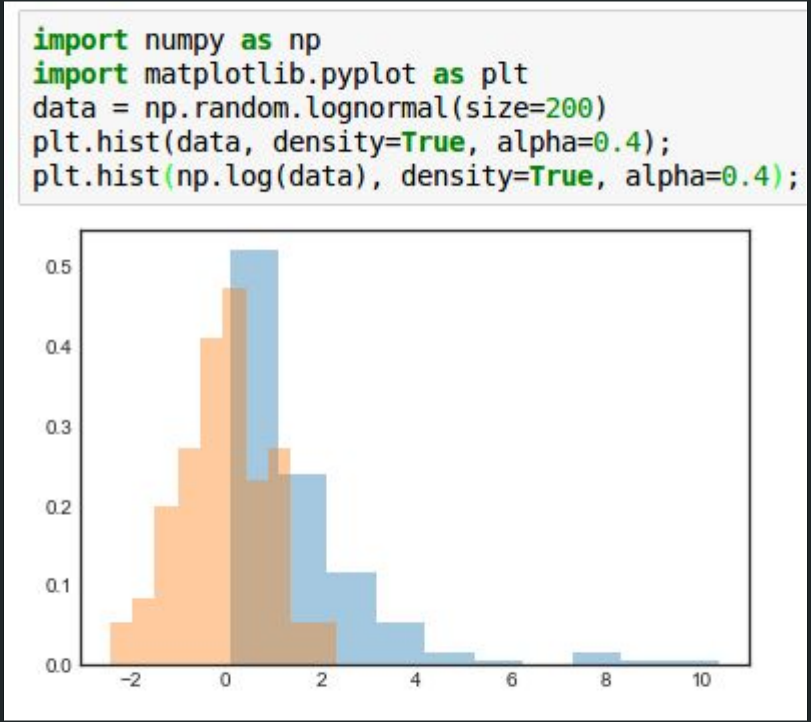
Graph types

There are three main reasons for plotting:

1. **Exploratory analyses**: display raw data, look for outliers, striking patterns.

Univariate histograms are a fundamental tool for exploring the distribution of a variable

Is the distribution centered, is it skewed, should you transform the data (e.g., log)?



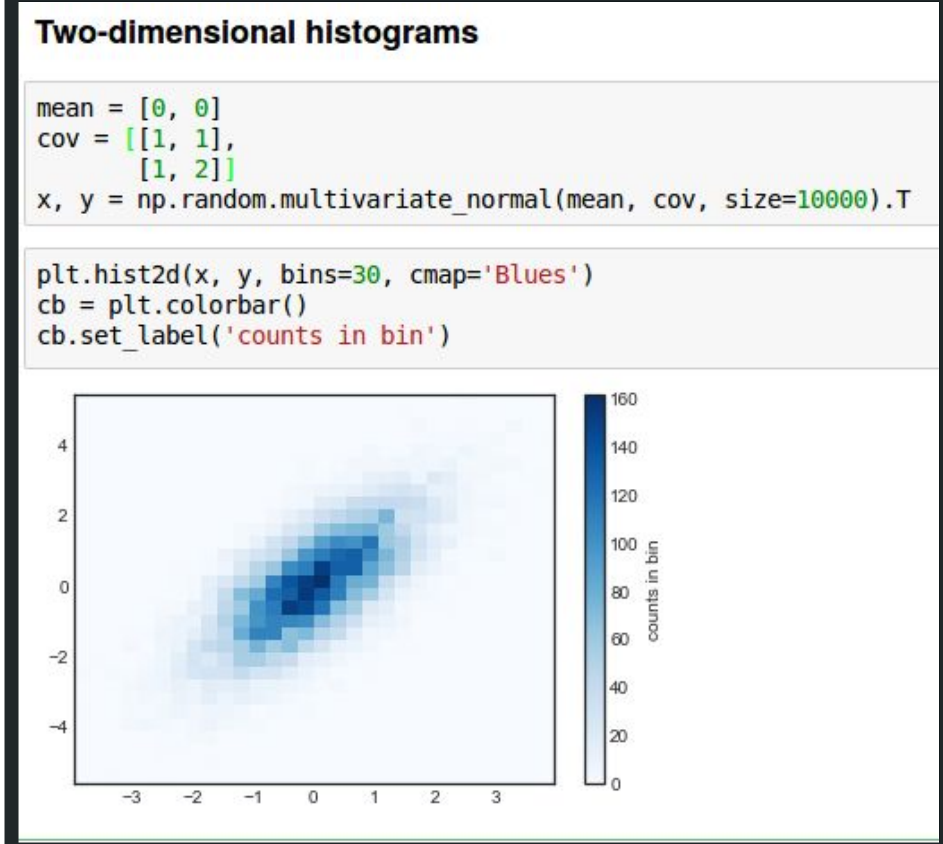
```python
import numpy as np
import matplotlib.pyplot as plt
data = np.random.lognormal(size=200)
plt.hist(data, density=True, alpha=0.4);
plt.hist(np.log(data), density=True, alpha=0.4);
```

Graph types

There are three main reasons for plotting:

1. **Exploratory analyses**: display raw data, look for outliers, striking patterns.

Pairplots: Two-dimensional histograms, and or scatterplots



Two-dimensional histograms

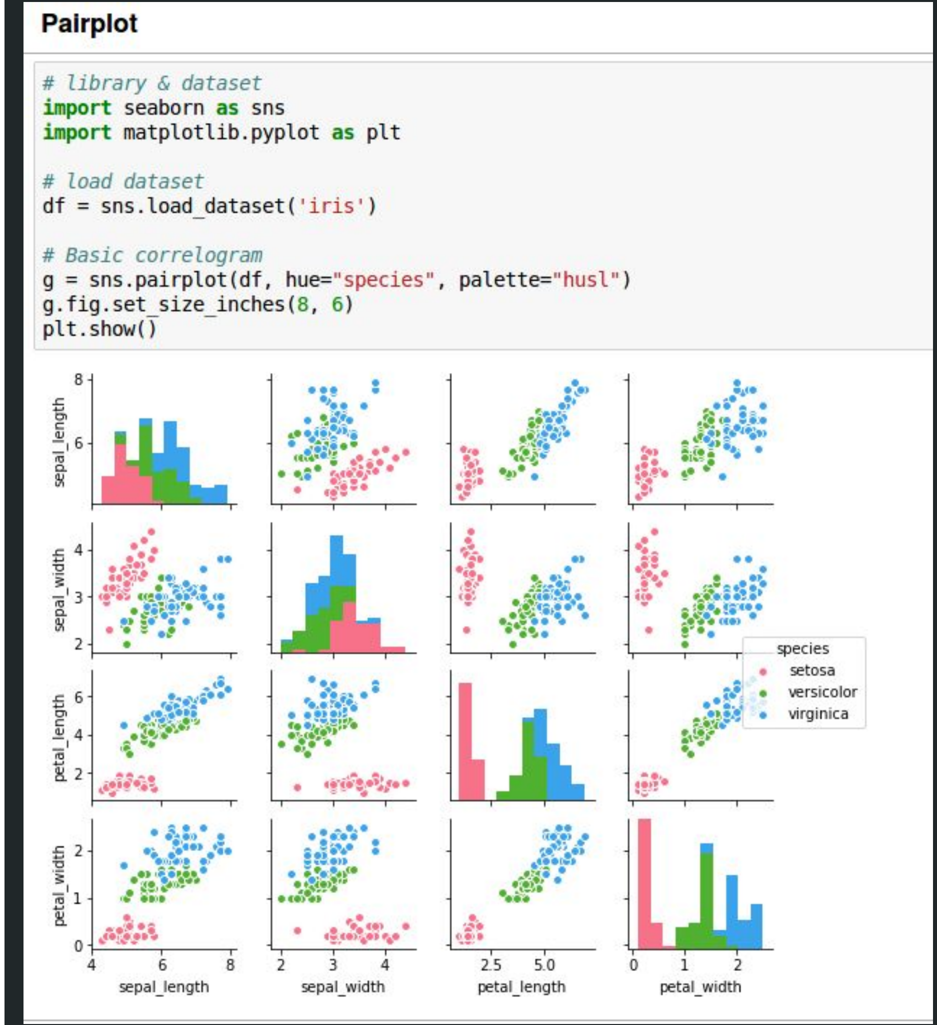
```
mean = [0, 0]
cov = [[1, 1],
       [1, 2]]
x, y = np.random.multivariate_normal(mean, cov, size=10000).T
```

```
plt.hist2d(x, y, bins=30, cmap='Blues')
cb = plt.colorbar()
cb.set_label('counts in bin')
```

Pair plot exploratory tool

Quickly view relationships among multiple variables using both scatterplots and histograms.

The matplotlib + seaborn functions for this are very convenient to use.



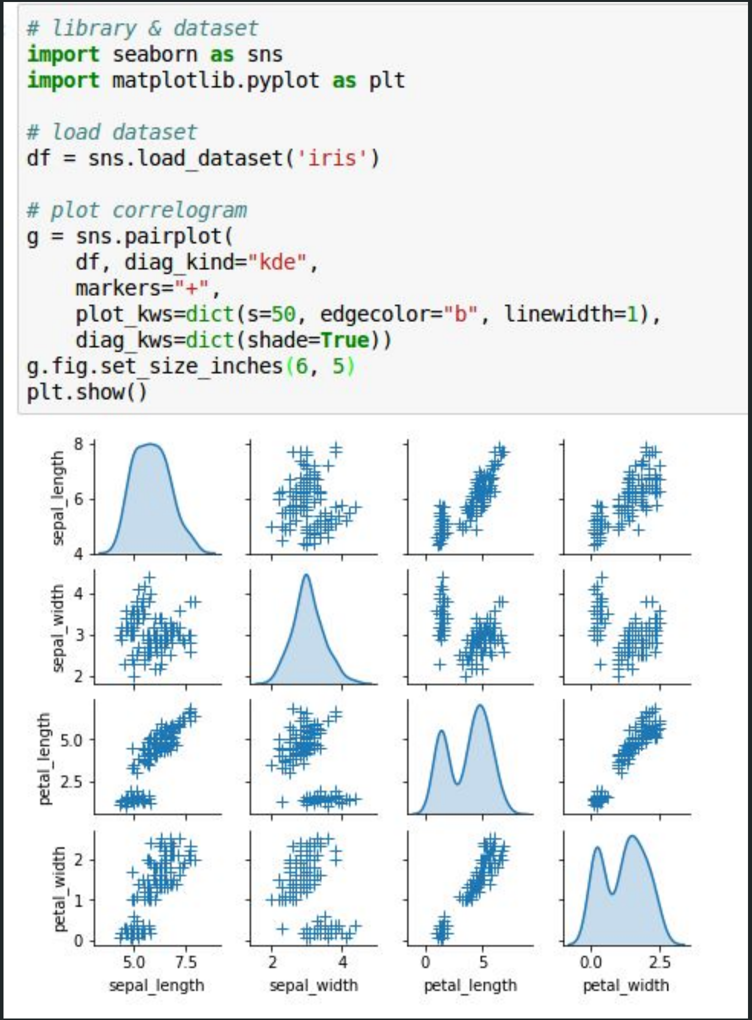
Pairplot

```python
# library & dataset
import seaborn as sns
import matplotlib.pyplot as plt

# load dataset
df = sns.load_dataset('iris')

# Basic correlogram
g = sns.pairplot(df, hue="species", palette="husl")
g.fig.set_size_inches(8, 6)
plt.show()
```

Pair plot exploratory tool

Quickly view relationships among multiple variables using both scatterplots and histograms.

The matplotlib + seaborn functions for this are very convenient to use.



```
# library & dataset
import seaborn as sns
import matplotlib.pyplot as plt

# load dataset
df = sns.load_dataset('iris')

# plot correlogram
g = sns.pairplot(
    df, diag_kind="kde",
    markers="+",
    plot_kws=dict(s=50, edgecolor="b", linewidth=1),
    diag_kws=dict(shade=True))
g.fig.set_size_inches(6, 5)
plt.show()
```

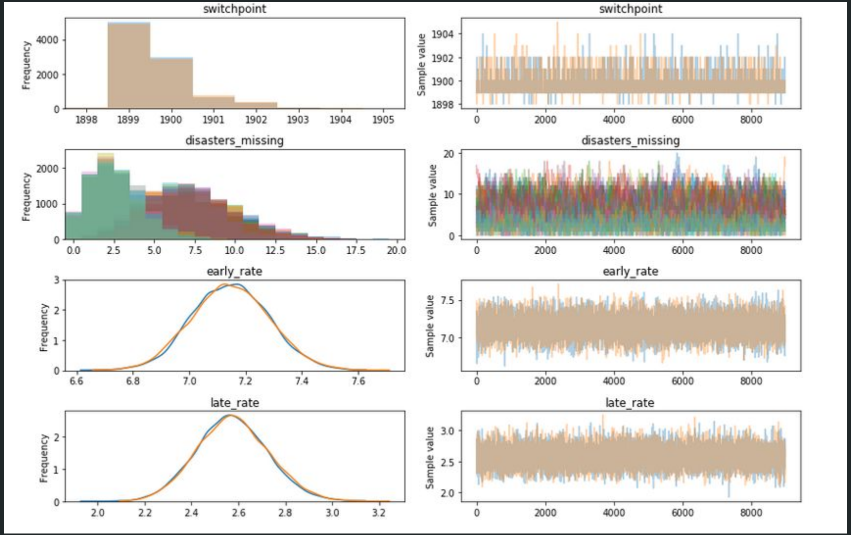
Graph types

There are three main reasons for plotting:

1. Exploratory analyses: display raw data, look for outliers, striking patterns.

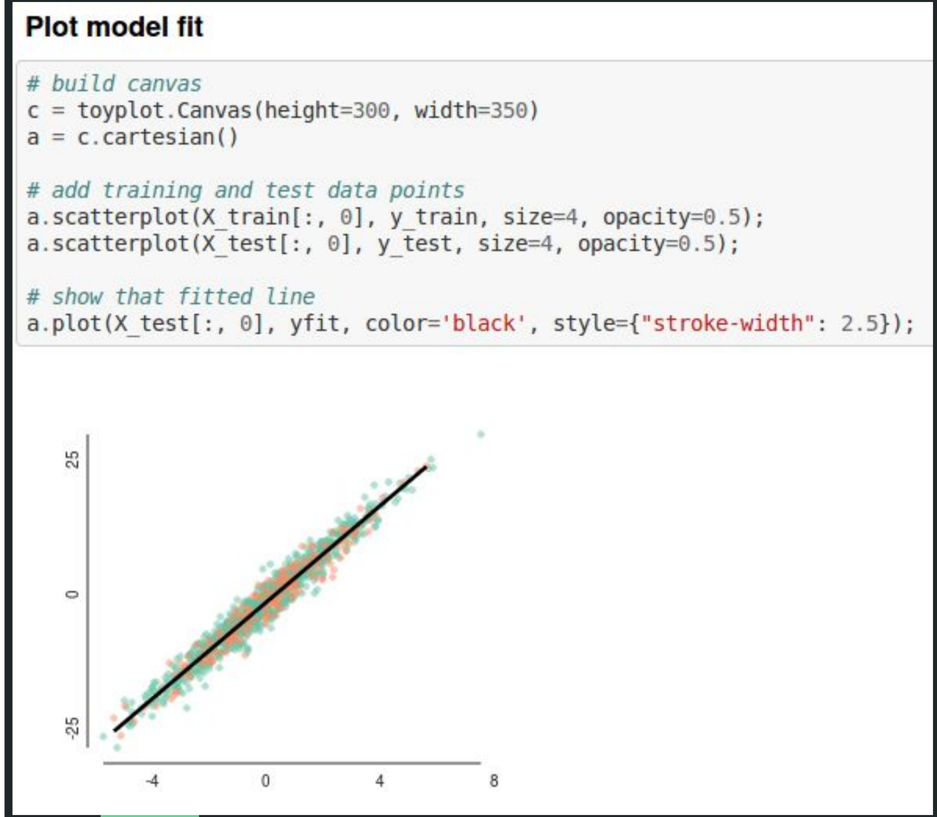
2. **Summary plots**: examine model fit, distributions, simulations, means.

3. Final results: a communication tool. A clear presentation that is instantly interpretable.



Summary plots

These will often involve plotting a model fit on top of your raw data. You want to see if your model fits your visual expectation. If your model infers a negative slope, but there is very clearly a positive slope in the distribution of points (at least to your eye) then you should really consider debugging your code further to double check it.



Plot model fit

```
# build canvas
c = toyplot.Canvas(height=300, width=350)
a = c.cartesian()

# add training and test data points
a.scatterplot(X_train[:, 0], y_train, size=4, opacity=0.5);
a.scatterplot(X_test[:, 0], y_test, size=4, opacity=0.5);

# show that fitted line
a.plot(X_test[:, 0], yfit, color='black', style={"stroke-width": 2.5});
```

Graph types

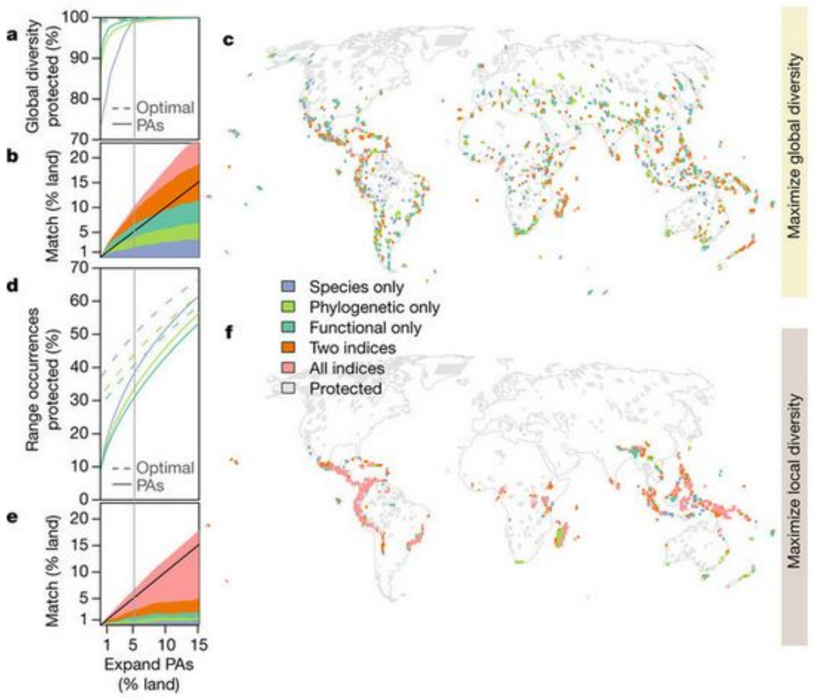
There are three main reasons for plotting:

1. Exploratory analyses: display raw data, look for outliers, striking patterns.

2. Summary plots: examine model fit, distributions, simulations, means.

3. **Final results:** a communication tool. A clear presentation that is instantly interpretable.

Figure 3: Priorities for expanding protected areas to conserve species, phylogenetic and functional trees of life for birds and mammals.



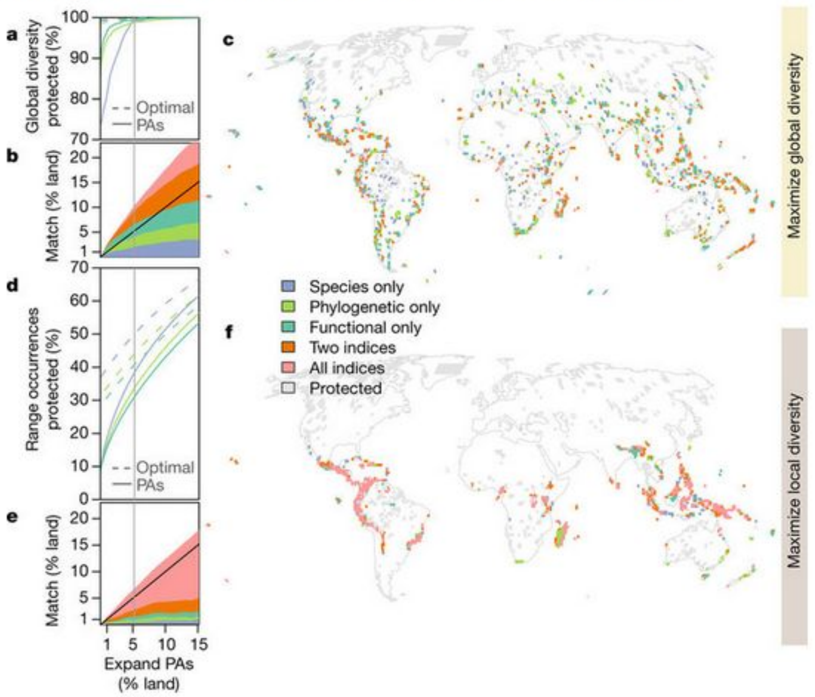
Igic et al. 2017

Final figures and editing

Plotting often has to do with much more than just representing data. It can be about making a striking image that catches attention. It is advertising.

At the same time, care should be taken that figures you create are not deceiving. Never fudge your data, and if you manually edit any part of a plot make it clear that that part is not a representation of actual data..

Figure 3: Priorities for expanding protected areas to conserve species, phylogenetic and functional trees of life for birds and mammals.

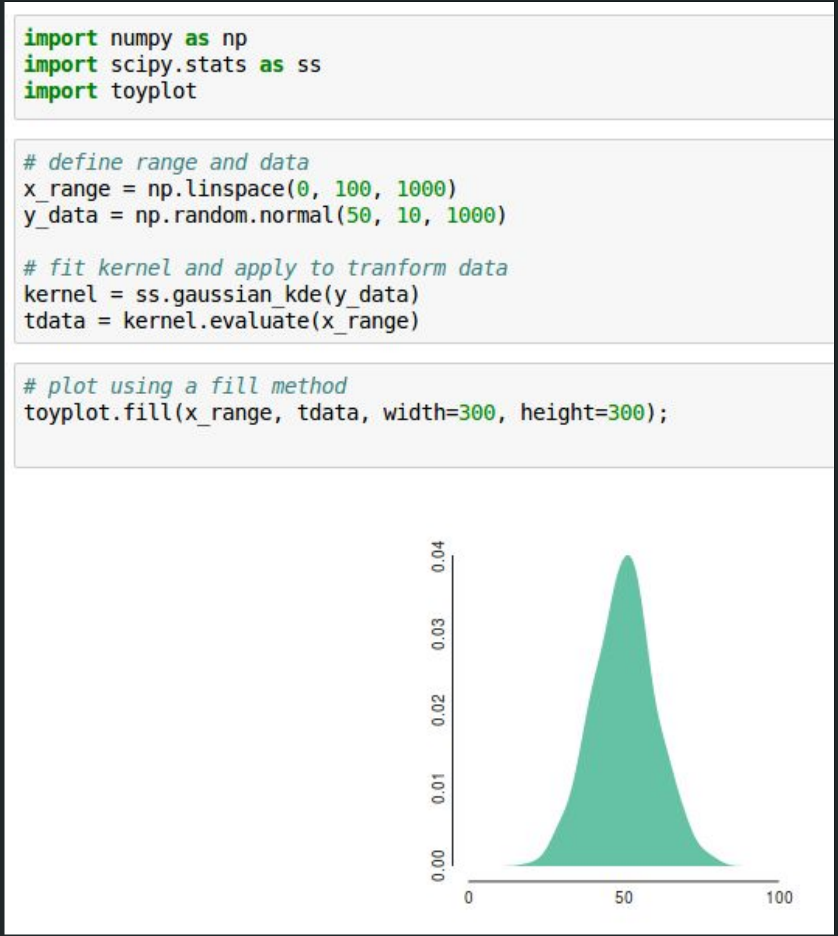


Bar plots and kernel density plots (smoothing)

Representing distributions of data can be tricky. Barplots are nice but they discretize data by binning, which hides some information.

A smoothed relationship across a distribution can be estimated using kernel smoothing. This effectively interpolates values between unsampled points by assuming a Gaussian (normal) distribution of points.

Scipy has kernel density estimation functions, and these are nicely integrated into seaborn.



```
import numpy as np
import scipy.stats as ss
import toyplot
```

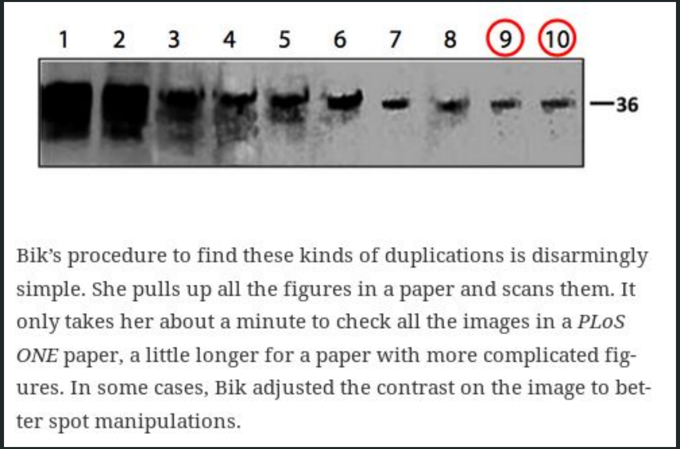
```
# define range and data
x_range = np.linspace(0, 100, 1000)
y_data = np.random.normal(50, 10, 1000)

# fit kernel and apply to tranform data
kernel = ss.gaussian_kde(y_data)
tdata = kernel.evaluate(x_range)
```

```
# plot using a fill method
toyplot.fill(x_range, tdata, width=300, height=300);
```

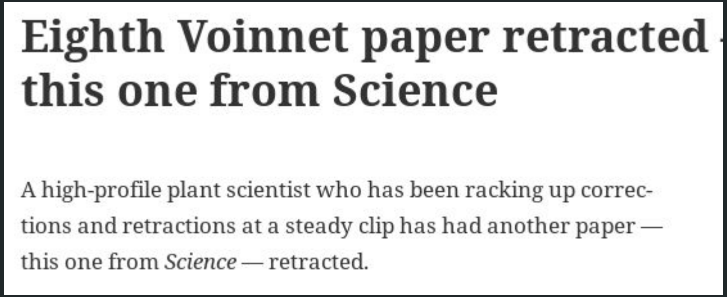
Manually fudging figures: It's fraudulent

Gel images are a famous example. It's ok to edit the contrast or brightness of an image to improve visibility, but labeling a sample as something different from what it is is fraud.

In gel images many examples of fraud have been uncovered because people splice together samples from multiple different lanes into a single image, but they copy an exact duplicate of a successful experiment and label it as the result for some other experiment that likely wasn't successful.



Bik's procedure to find these kinds of duplications is disarmingly simple. She pulls up all the figures in a paper and scans them. It only takes her about a minute to check all the images in a *PLoS ONE* paper, a little longer for a paper with more complicated figures. In some cases, Bik adjusted the contrast on the image to better spot manipulations.



Eighth Voinnet paper retracted this one from Science

A high-profile plant scientist who has been racking up corrections and retractions at a steady clip has had another paper — this one from *Science* — retracted.

retractionwatch.com

Python plotting tools:

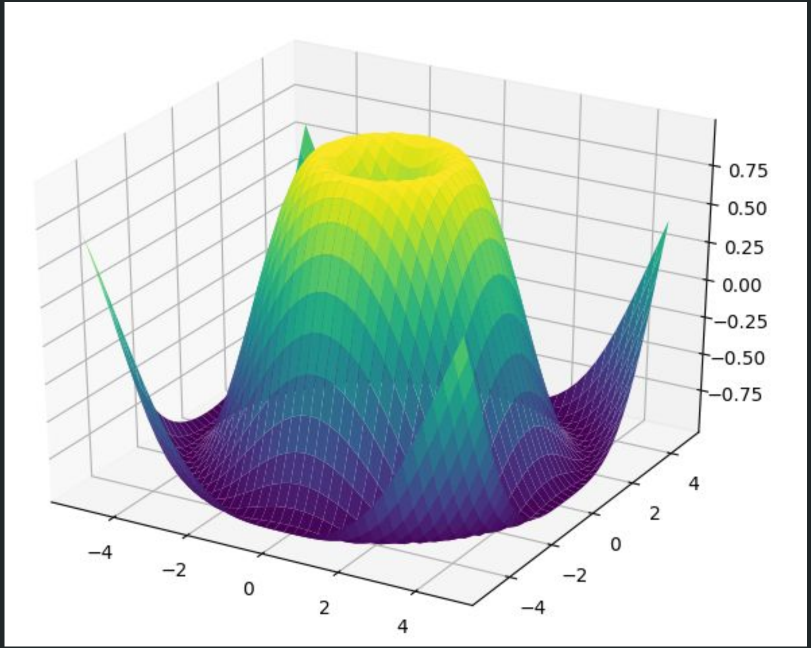
Matplotlib

Matplotlib was originally developed for Python as a clone of the most popular plotting tool at the time, which is from the language Matlab, which is a Python like language but is not free open-source.

Big update last year to matplotlib 2.0.

This is the largest and most developed plotting library in Python, but because it is old, the style and syntax can feel a bit aged.

Still, it is fast, powerful, and rich in functions.



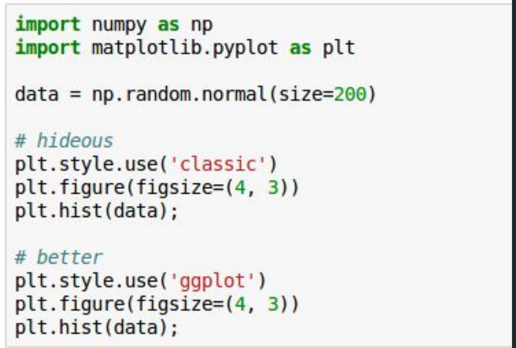
https://matplotlib.org/gallery/index.html

Matplotlib

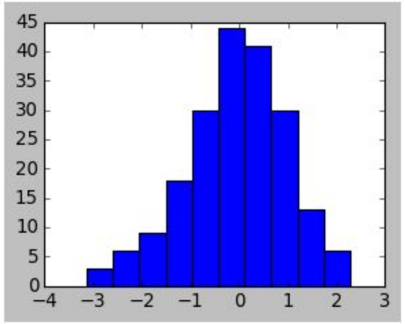
Built-in styles.

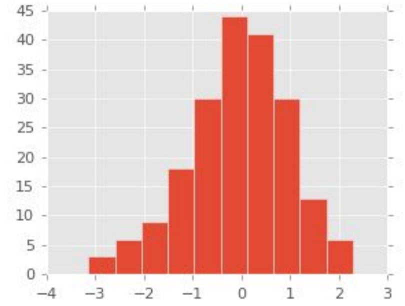
The defaults of matplotlib are ugly. This is one of my biggest complaints with it too.

However, you can easily set it to use preset styles, or design your own.



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

data = np.random.normal(size=200)

# hideous
plt.style.use('classic')
plt.figure(figsize=(4, 3))
plt.hist(data);

# better
plt.style.use('ggplot')
plt.figure(figsize=(4, 3))
plt.hist(data);
```

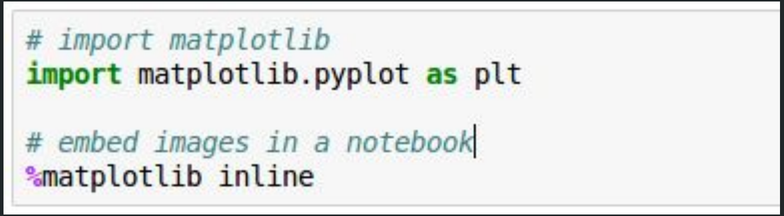




Matplotlib

Importing is a bit weird, but that's just the way it is. There are a few additional tools that you need to import for some fancier plots, like 3-d images.

The "inline" call only needs to be made once, and on some systems doesn't need to be made at all.



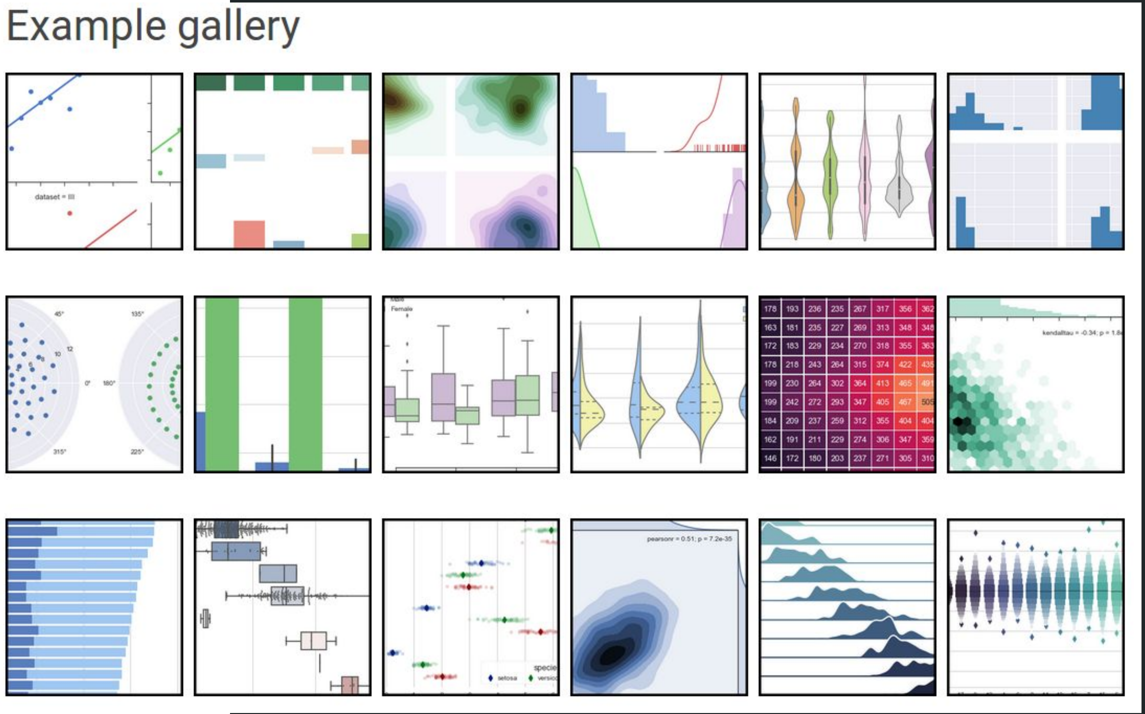
```
# import matplotlib
import matplotlib.pyplot as plt

# embed images in a notebook
%matplotlib inline
```

Seaborn

Seaborn is based on matplotlib

It adds additional styling on top of Matplotlib plots, and also has a number of useful functions for plotting statistical distributions

It is especially useful for kernel density plots.

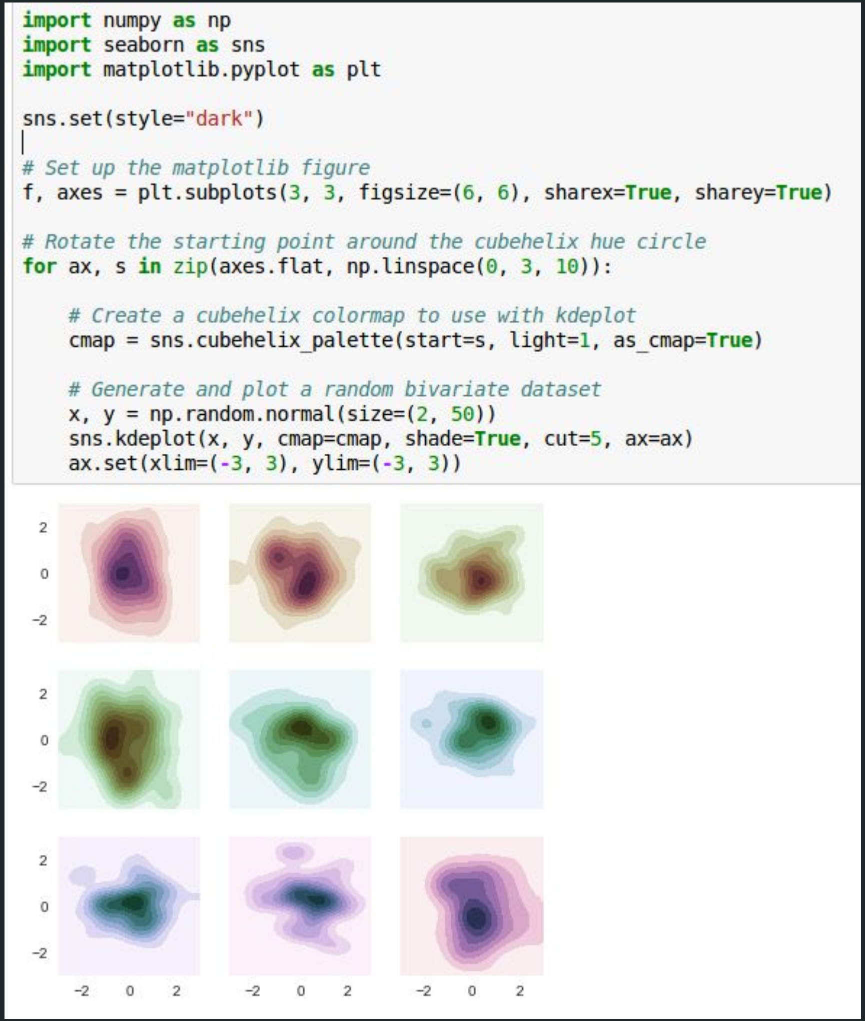


Seaborn

Seaborn is based on matplotlib

Here the example is using `sns.kdeplot( )` to plot bivariate kernel density plots.

The seaborn plots are still organized in a layout pattern using matplotlib as the base plotting functions.



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

sns.set(style="dark")

# Set up the matplotlib figure
f, axes = plt.subplots(3, 3, figsize=(6, 6), sharex=True, sharey=True)

# Rotate the starting point around the cubehelix hue circle
for ax, s in zip(axes.flat, np.linspace(0, 3, 10)):

    # Create a cubehelix colormap to use with kdeplot
    cmap = sns.cubehelix_palette(start=s, light=1, as_cmap=True)

    # Generate and plot a random bivariate dataset
    x, y = np.random.normal(size=(2, 50))
    sns.kdeplot(x, y, cmap=cmap, shade=True, cut=5, ax=ax)
    ax.set(xlim=(-3, 3), ylim=(-3, 3))
```

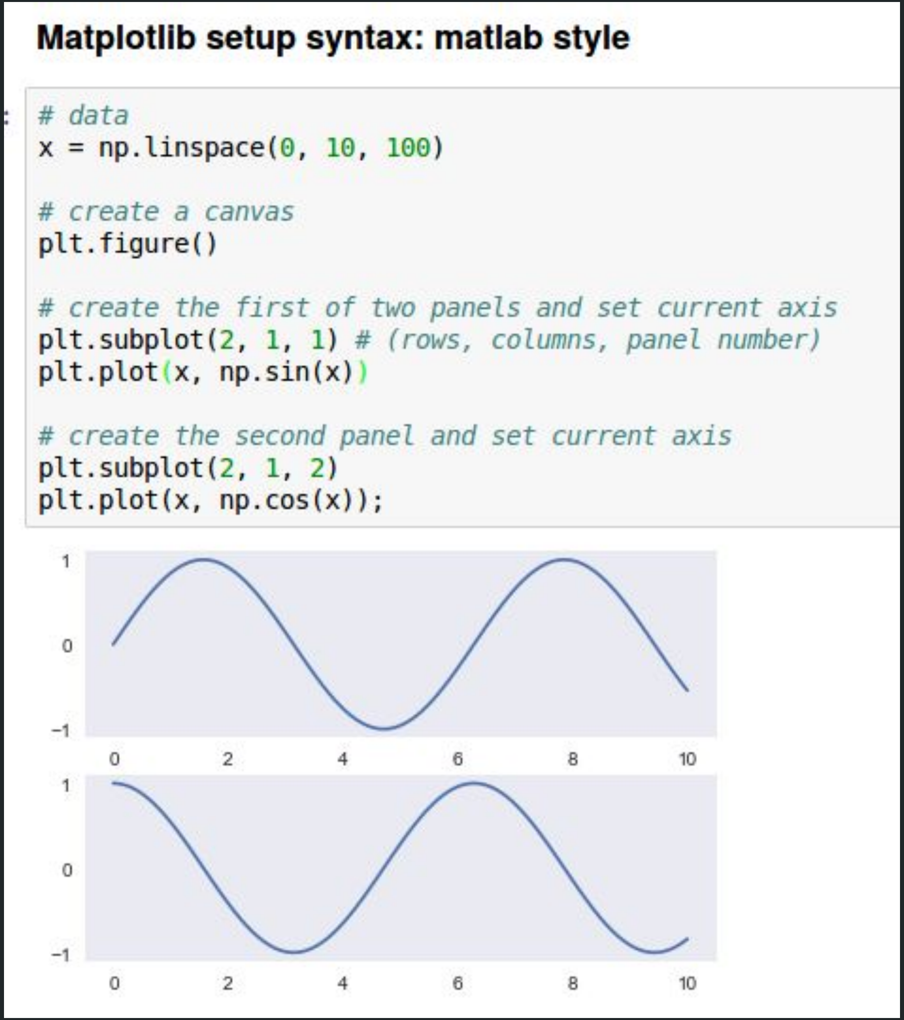
Matplotlib syntax

As described in your reading, there are two syntax types that can be used to create a matplotlib figure.

1. MATLAB style (old style).

All calls come from plt, you only work on one plot or subplot at a time.

Here it is sometimes useful to call plt.gcf( ) which will "get current figure".


Matplotlib setup syntax: matlab style

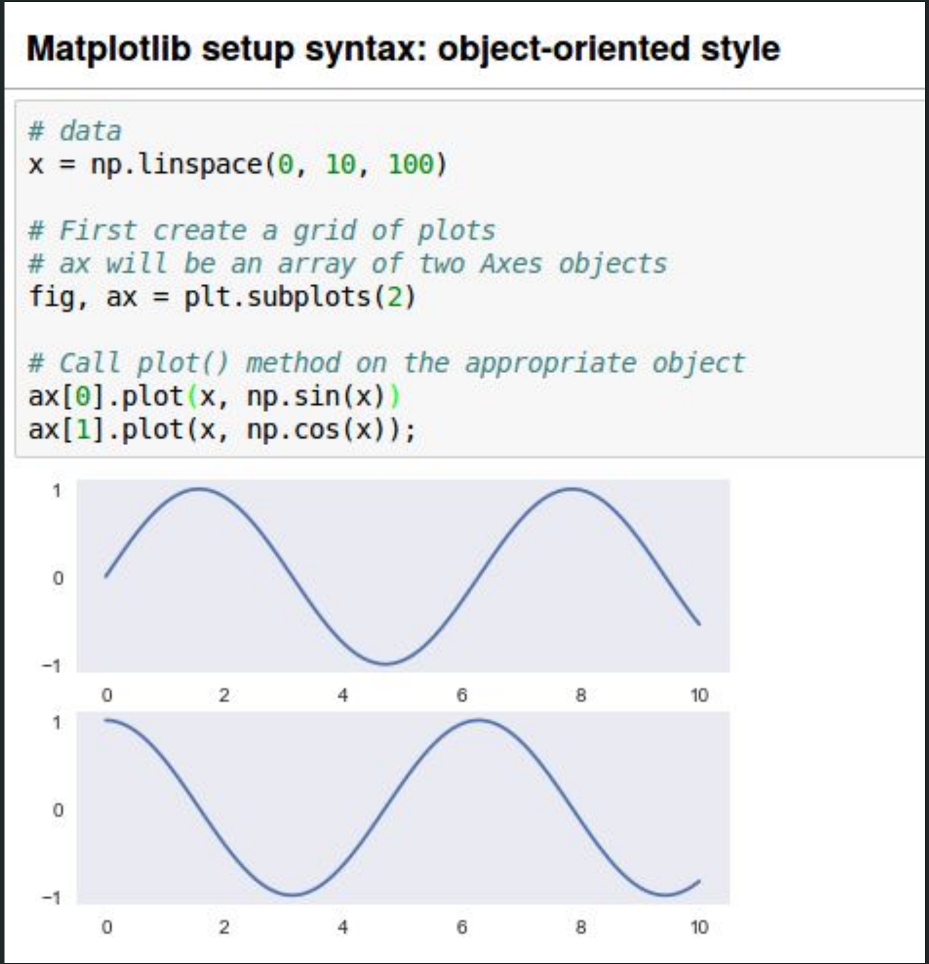
```
# data
x = np.linspace(0, 10, 100)

# create a canvas
plt.figure()

# create the first of two panels and set current axis
plt.subplot(2, 1, 1) # (rows, columns, panel number)
plt.plot(x, np.sin(x))

# create the second panel and set current axis
plt.subplot(2, 1, 2)
plt.plot(x, np.cos(x));
```

Matplotlib syntax

As described in your reading, there are two syntax types that can be used to create a matplotlib figure.

1. MATLAB style (old style).
2. Python style (object oriented)

Each subplot is its own object, you can modify them as much as you want in whatever order you want.



Matplotlib setup syntax: object-oriented style

```
# data
x = np.linspace(0, 10, 100)

# First create a grid of plots
# ax will be an array of two Axes objects
fig, ax = plt.subplots(2)

# Call plot() method on the appropriate object
ax[0].plot(x, np.sin(x))
ax[1].plot(x, np.cos(x));
```

Matplotlib syntax

figure is a class object that holds axes and dictates the size of the canvas.

axes is a class object that sets a scale to the canvas and on which data will be plotted. You cannot see a plot until it has axes.

The simpler functions like plt.plot(x) implicitly create a figure and axes when this is called.

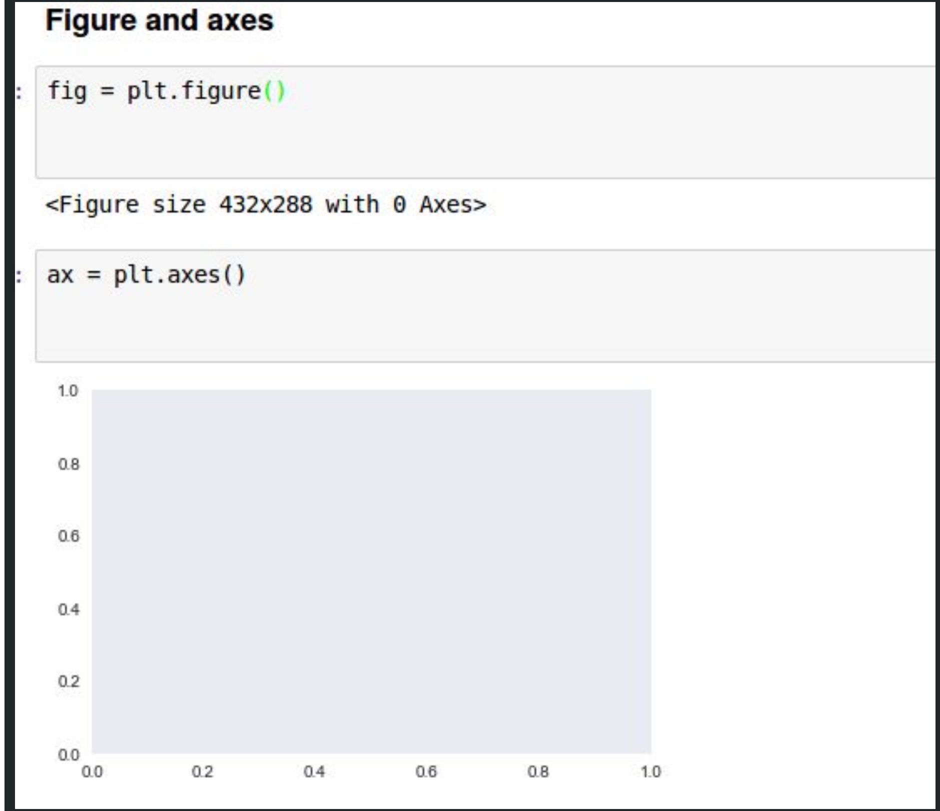


Figure and axes

```
fig = plt.figure()
```

```
<Figure size 432x288 with 0 Axes>
```

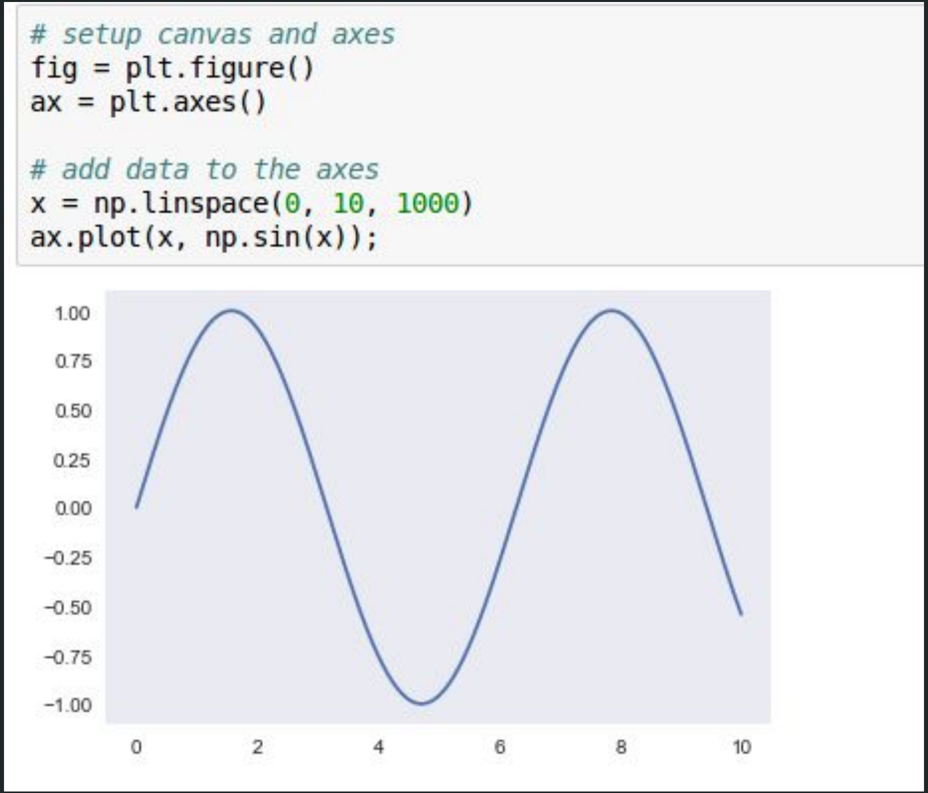
```
ax = plt.axes()
```

Matplotlib syntax

figure is a class object that holds axes and dictates the size of the canvas.

axes is a class object that sets a scale to the canvas and on which data will be plotted. You cannot see a plot until it has axes.

The simpler functions like plt.plot(x) implicitly create a figure and axes when this is called.



```
# setup canvas and axes
fig = plt.figure()
ax = plt.axes()

# add data to the axes
x = np.linspace(0, 10, 1000)
ax.plot(x, np.sin(x));
```

Matplotlib syntax

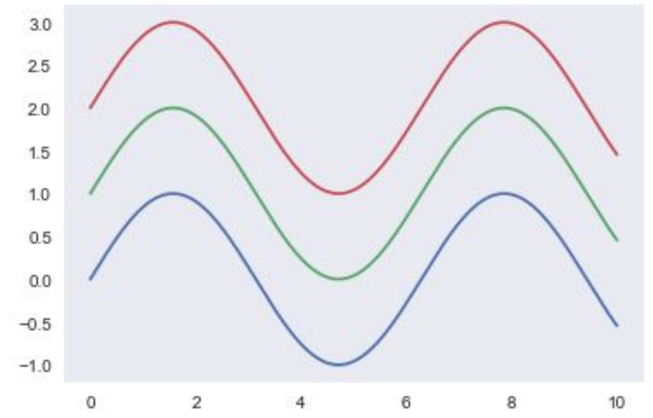
figure is a class object that holds axes and dictates the size of the canvas.

axes is a class object that sets a scale to the canvas and on which data will be plotted. You cannot see a plot until it has axes.

The simpler functions like plt.plot(x) implicitly create a figure and axes when this is called.

Sequential calls to the same axes object will add plots on top of the same coordinate grid. It will cycle through the default colormap for each subsequent plot.

```
# setup canvas and axes
fig = plt.figure()
ax = plt.axes()

# add data to the axes
x = np.linspace(0, 10, 1000)
ax.plot(x, np.sin(x));
ax.plot(x, np.sin(x) + 1);
ax.plot(x, np.sin(x) + 2);
```

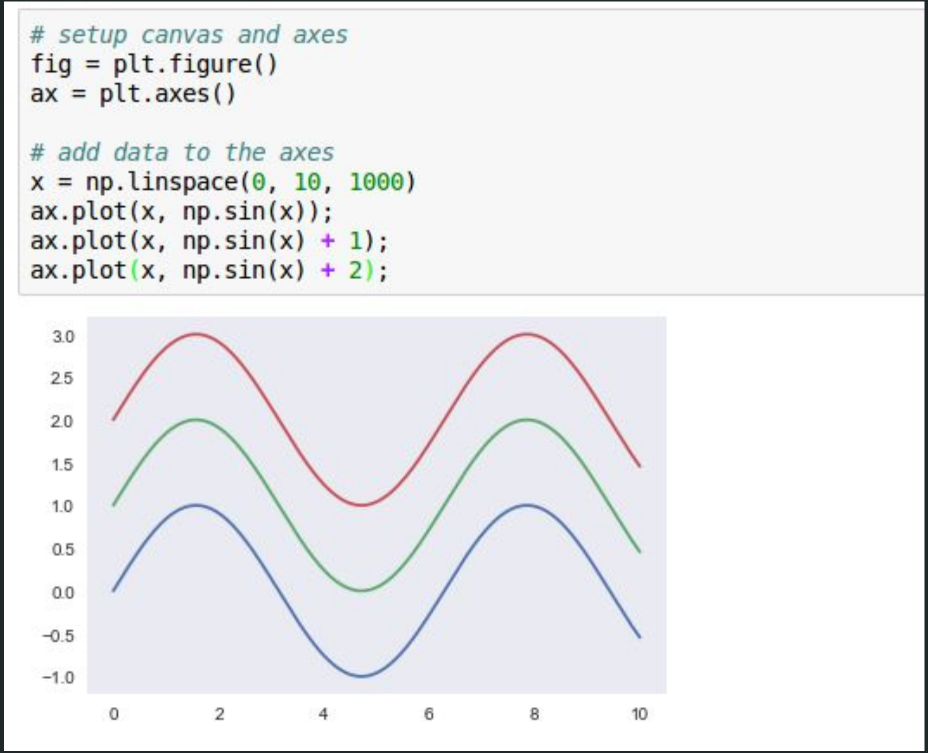


Matplotlib syntax

Manually styling plots. There are soooo many options. How to find them?

Example: How do you control spacing of tick marks on the x-axis? Ask google.

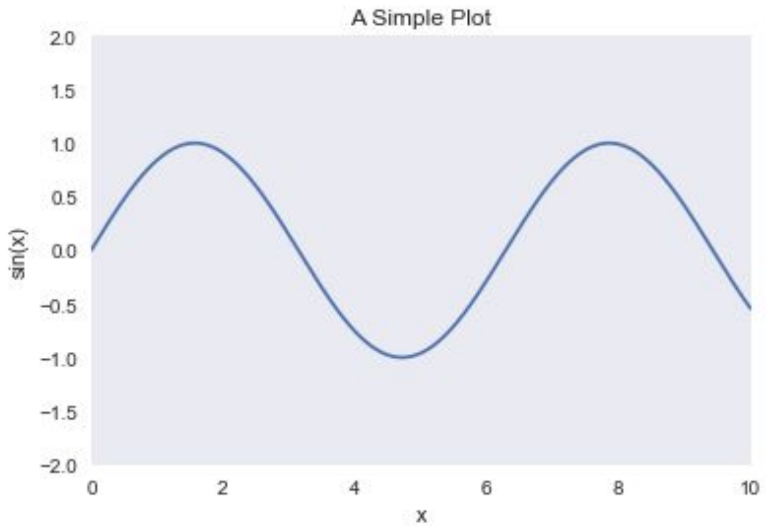
Also explore options use tab-completion of figure and axes objects.

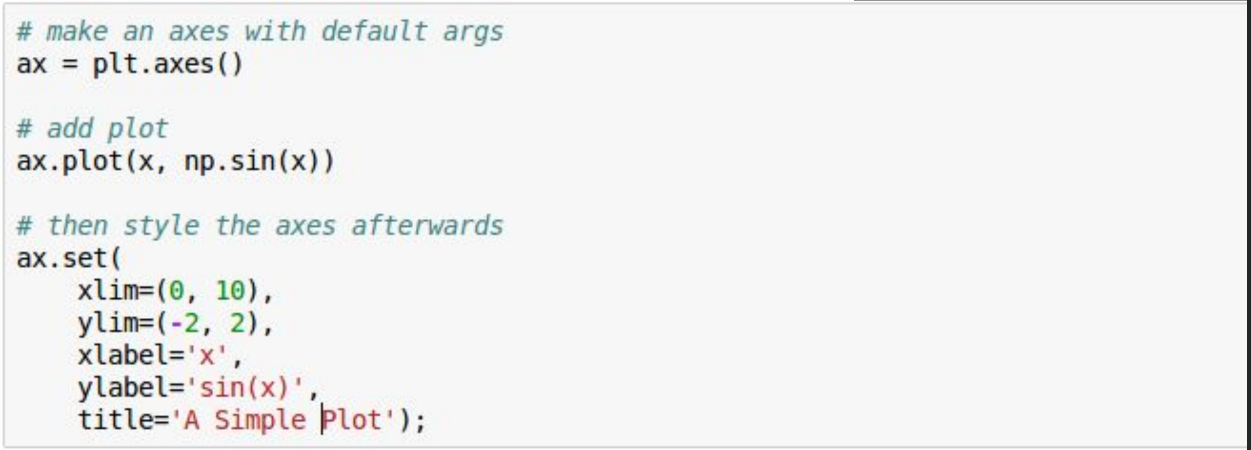


```
# setup canvas and axes
fig = plt.figure()
ax = plt.axes()

# add data to the axes
x = np.linspace(0, 10, 1000)
ax.plot(x, np.sin(x));
ax.plot(x, np.sin(x) + 1);
ax.plot(x, np.sin(x) + 2);
```

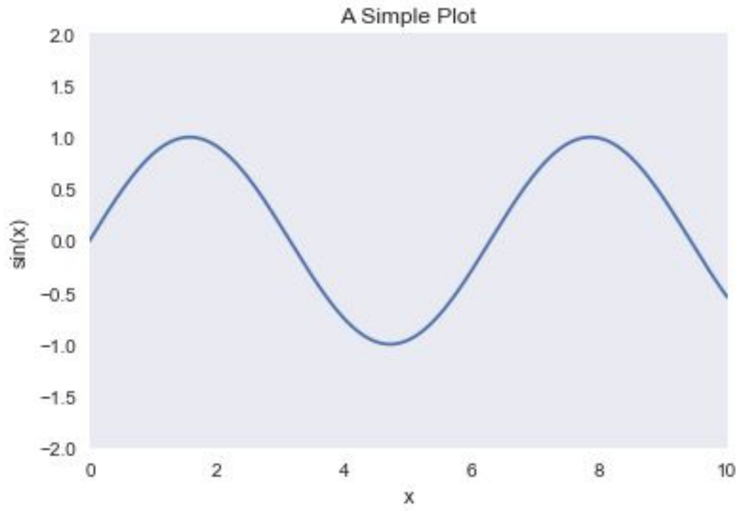
Matplotlib syntax

```
# initialize the axes with all arguments you wish them you have
ax = plt.axes(xlim=(0, 10), ylim=(-2, 2), xlabel='x', ylabel='sin(x)', title='A Simple Plot')
ax.plot(x, np.sin(x));
```





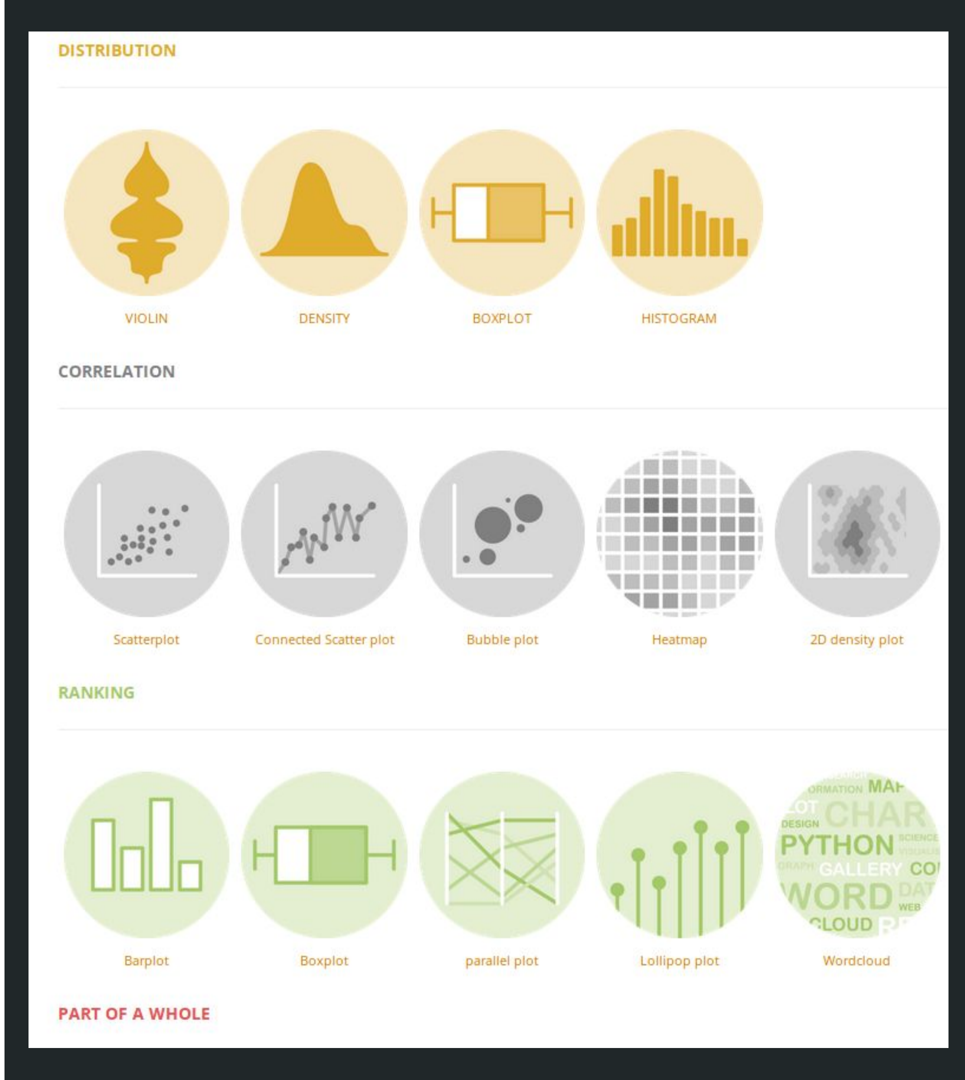
```
# make an axes with default args
ax = plt.axes()

# add plot
ax.plot(x, np.sin(x))

# then style the axes afterwards
ax.set(
    xlim=(0, 10),
    ylim=(-2, 2),
    xlabel='x',
    ylabel='sin(x)',
    title='A Simple Plot');
```



Numerous examples

https://python-graph-gallery.com/

https://matplotlib.org/gallery.html

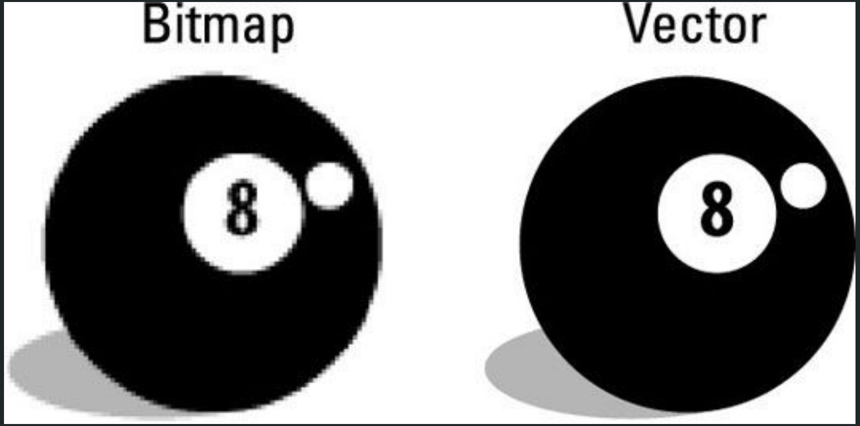


Saving figures (formats)

Bitmap: (e.g., PNG) The image is a grid of points and each point is mapped to a color in a three-dimensional matrix (r, g, b) or four-dims (r, g, b, a).

Pros: This is a compact method of storing images, and is how digital photos are stored.

Cons: pixelated. Lines are not crisp. File size can become very large if you increase DPI.

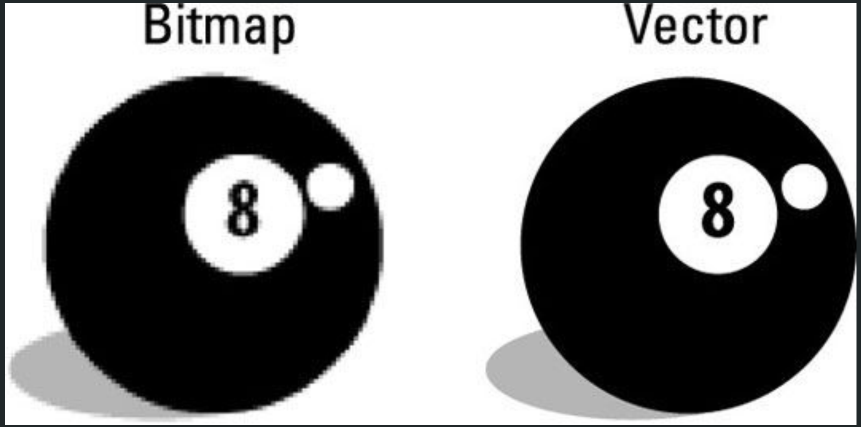


Saving figures (formats)

Vector: (e.g., SVG) The image is a set of code written in an XML-based drawing language that describes a set of shapes and their 2-dimensional relationship. Can include animation and even embedded bitmaps.

Pros: infinitely scalable and does not become pixelated. Very easy to edit the image after it is created using Illustrator or Inkscape.

Cons: If your image needs to contain a photo then it will still be stored as a bitmap. For plots with many points/objects the filesize can become quite large.



Toyplot and other web-based plotting libraries

SVG is the default format for data.

HTML can be wrapped around SVG so that it displays in a browser (or in the output of a jupyter notebook.

JavaScript functions can be added to the HTML document (toyplot) or provided separate (bokeh) to add interactivity to the plot by giving instructions to your browser.

Toyplot and other web-based plotting libraries

Toyplot, bokeh, altair, and several other Python plotting libraries aim to provide a clean and easy Pythonic interface to create rich interactive web-based plots.

This avoids the need for learning additional languages like javascript and CSS, although knowledge of those certainly makes these libraries easier to use.

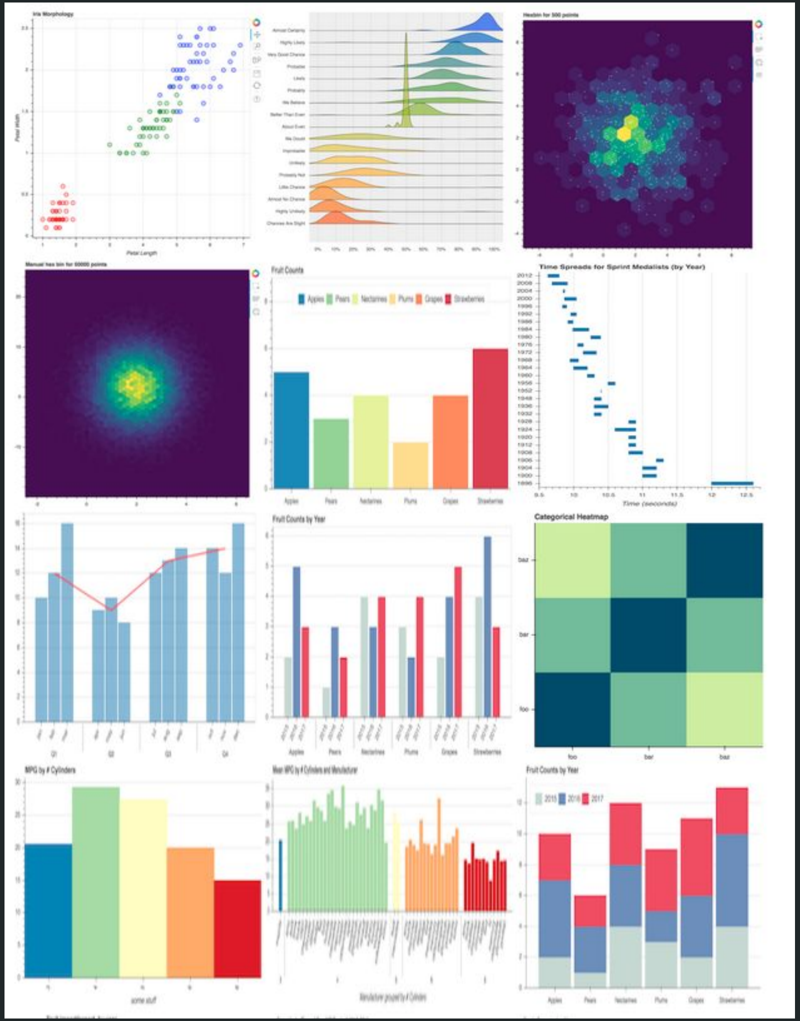
Bokeh

Bokeh: The most complicated, and the most javascript-like.

You can develop apps that run on a server, and which can be embedded in a website. These can handle super large datasets and react very quickly.

It is very powerful, but very complicated to learn. *In my opinion, it often unnecessarily complicated.*

https://demo.bokehplots.com/apps/surface3d



Bokeh

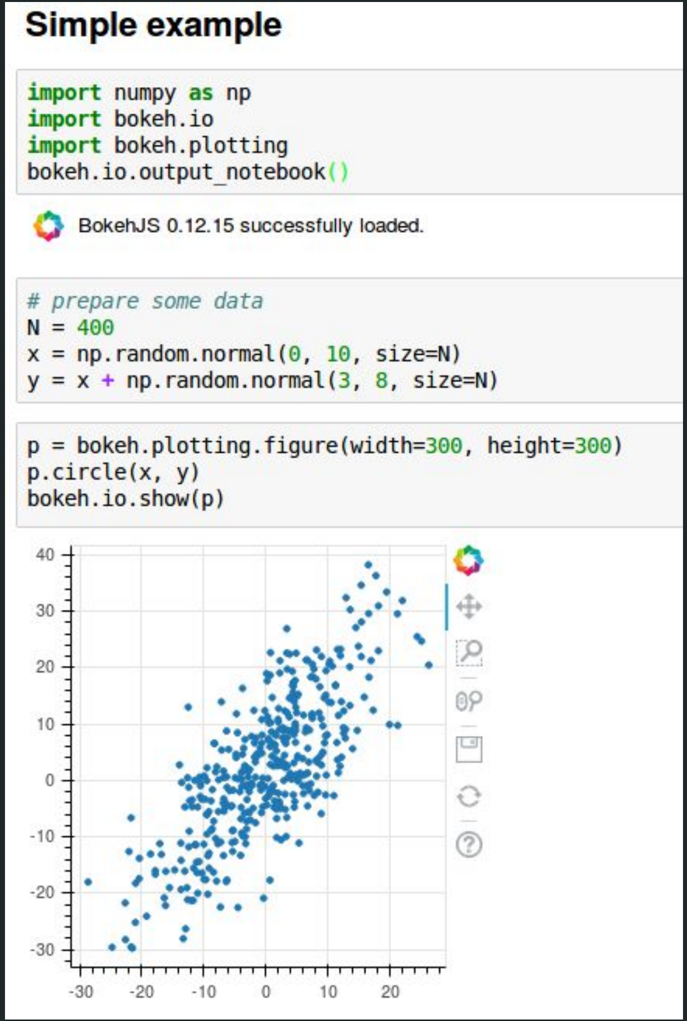
The syntax is mostly similar to matplotlib, and other libraries we'll look at, in that it is object oriented.

It requires many separate imports. And requires that you call .show()

Can be used without setting up a server, to embed "static plots" with limited interactivity.

Works well with jupyter.

notebooks/nb-11.1-bokeh-scatterplot.ipynb
notebooks/nb-11.2-bokeh-layout.ipynb
notebooks/nb-11.3-bokeh-hist.ipynb



Simple example

```
import numpy as np
import bokeh.io
import bokeh.plotting
bokeh.io.output_notebook()
```

BokehJS 0.12.15 successfully loaded.

```
# prepare some data
N = 400
x = np.random.normal(0, 10, size=N)
y = x + np.random.normal(3, 8, size=N)
```

```
p = bokeh.plotting.figure(width=300, height=300)
p.circle(x, y)
bokeh.io.show(p)
```

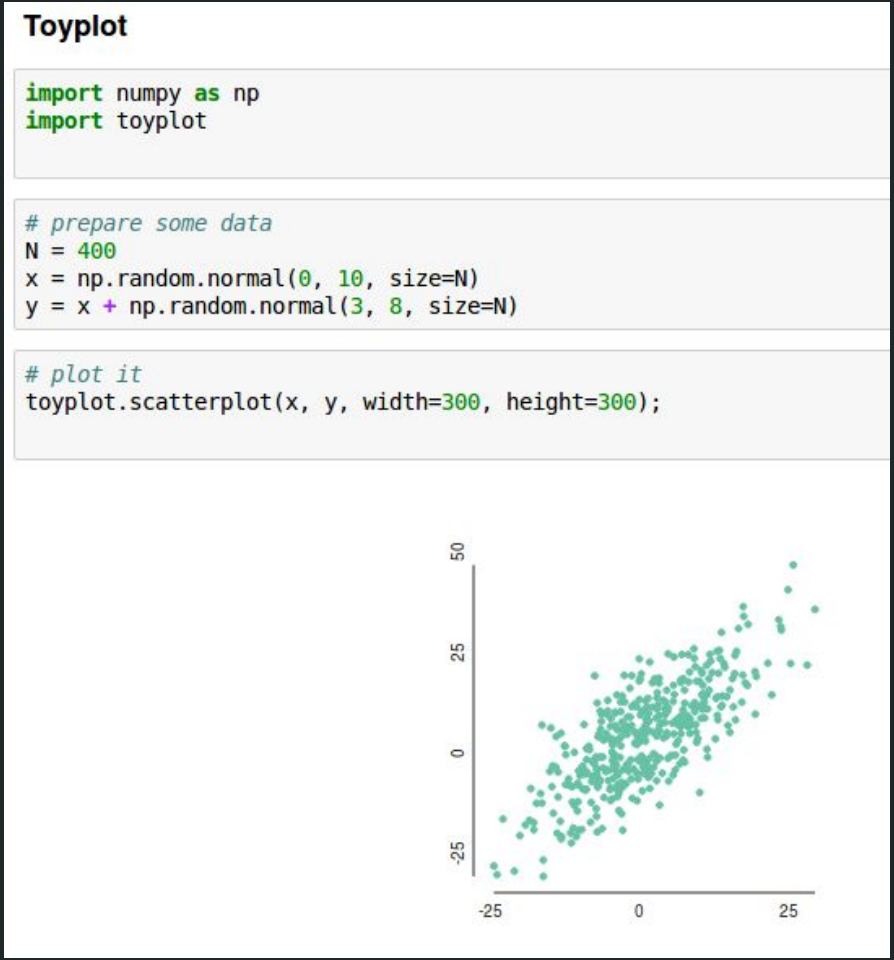
Toyplot

Very minimalist, and pretty new.

Creates interactive plots as a single file/object: SVG+HTML+JS

This means that the interactive plots are highly portable. Send interactive plots through email, add them to your website, and if you're lucky, get an editor to put it in the online version of your paper.

Works well with jupyter.



Toyplot

```
import numpy as np
import toyplot
```

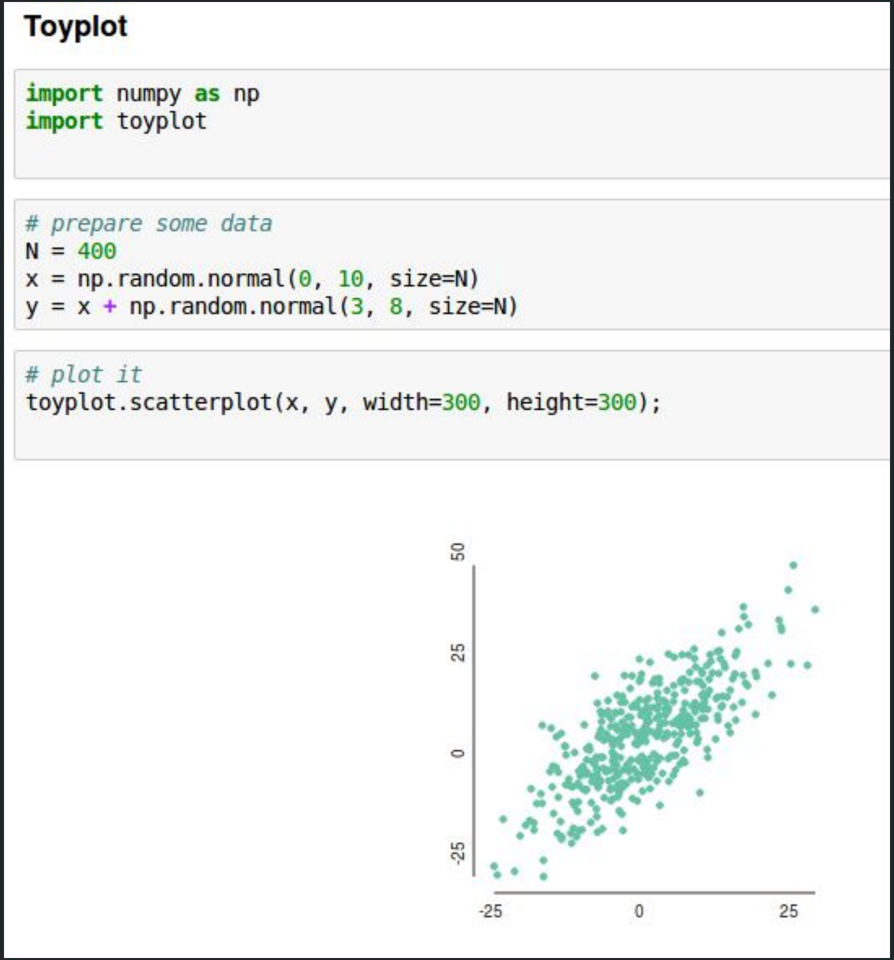
```
# prepare some data
N = 400
x = np.random.normal(0, 10, size=N)
y = x + np.random.normal(3, 8, size=N)
```

```
# plot it
toyplot.scatterplot(x, y, width=300, height=300);
```

Toyplot

Pros: Easy to use, good documentation (not 1 million scattered examples like matplotlib). Good default styles and colors. Portability.

Cons: Large plots become burdensome due to the all-in-one format. But these can easily be reduced by saving them as SVG or PNG. For embedding large files in jupyter use PNG, otherwise default of HTML is best. Lacks some fancier features in matplotlib like 3d plots.



Toyplot

```
import numpy as np
import toyplot
```

```
# prepare some data
N = 400
x = np.random.normal(0, 10, size=N)
y = x + np.random.normal(3, 8, size=N)
```

```
# plot it
toyplot.scatterplot(x, y, width=300, height=300);
```

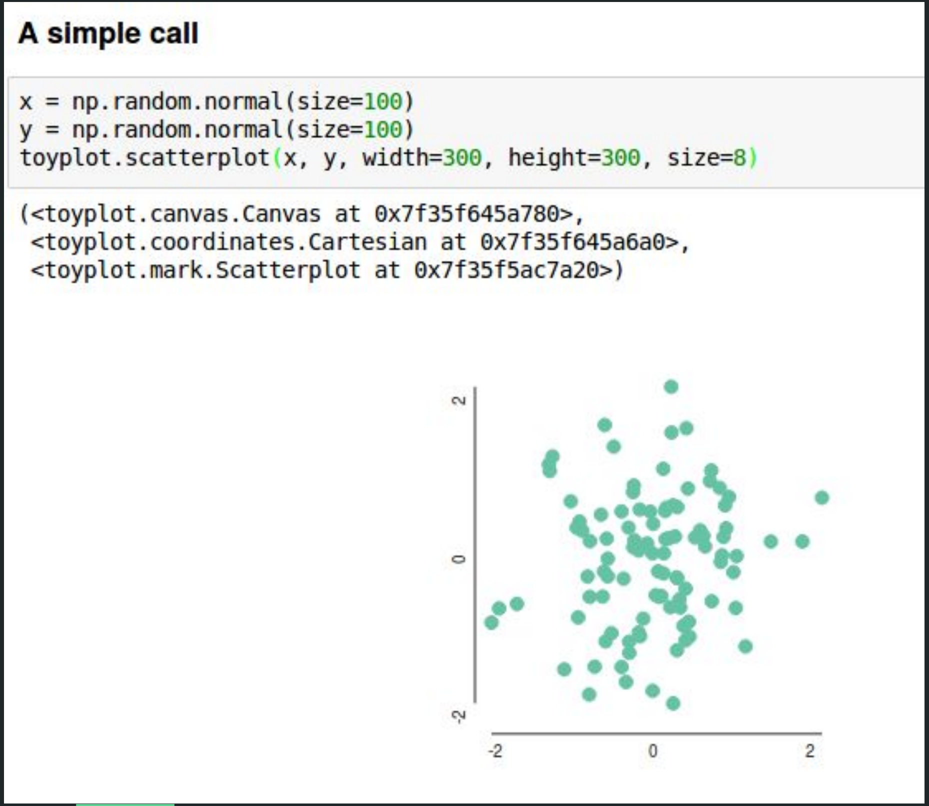
Toyplot

Syntax is like the object oriented version of matplotlib.

Class objects:

Canvas: the plotting area.
Axes: the scaled cartesian grid.
Mark: points, lines, or shapes drawn on axes.

A simple call from `toyplot.<function>` will create all three objects and return them as a tuple.



A simple call

```
x = np.random.normal(size=100)
y = np.random.normal(size=100)
toyplot.scatterplot(x, y, width=300, height=300, size=8)
```

```
(<toyplot.canvas.Canvas at 0x7f35f645a780>,
 <toyplot.coordinates.Cartesian at 0x7f35f645a6a0>,
 <toyplot.mark.Scatterplot at 0x7f35f5ac7a20>)
```

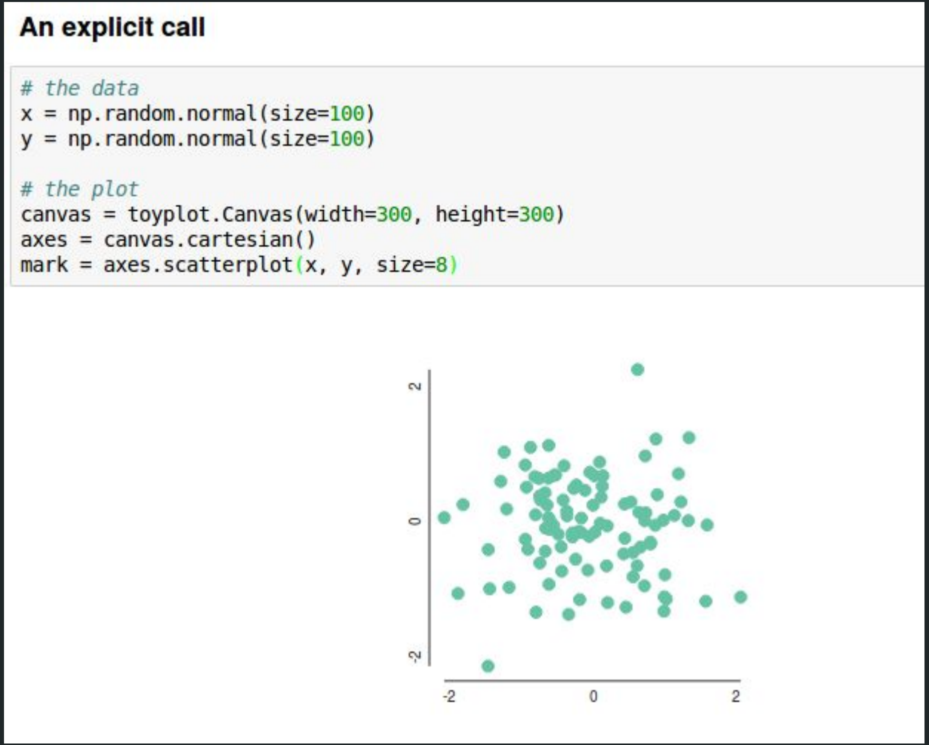
Toyplot

Syntax is like the object oriented version of matplotlib.

With explicit calls to all three objects you can see that each is a function of another.

Multiple axes can be created on the same canvas.

Multiple marks can be added to the same axes.



An explicit call

```
# the data
x = np.random.normal(size=100)
y = np.random.normal(size=100)

# the plot
canvas = toyplot.Canvas(width=300, height=300)
axes = canvas.cartesian()
mark = axes.scatterplot(x, y, size=8)
```

Toyplot

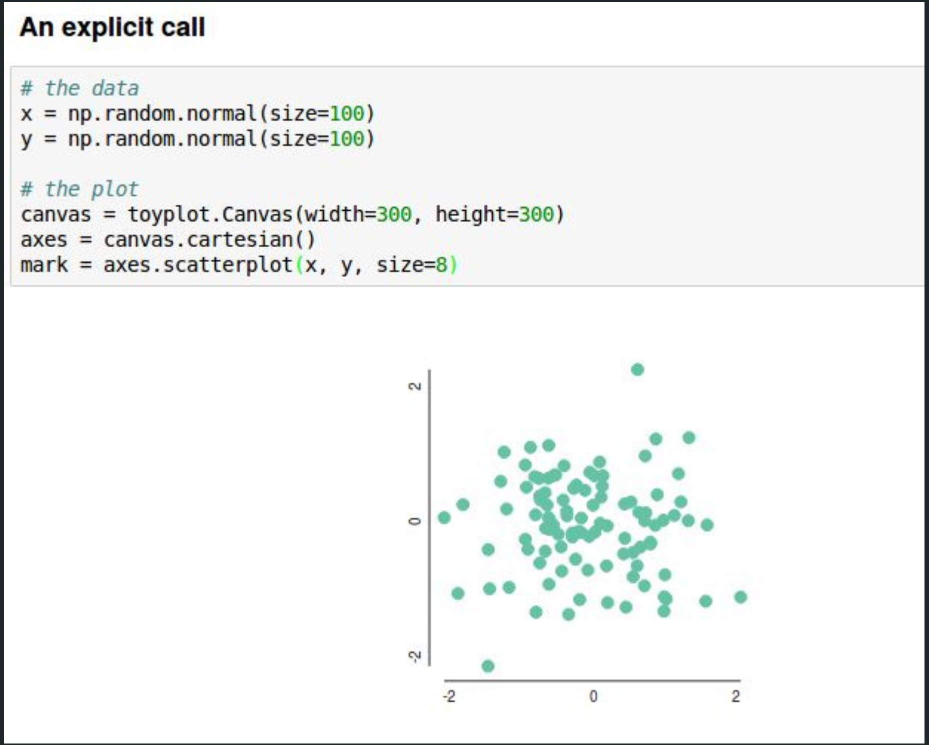
Syntax is like the object oriented version of matplotlib.

With explicit calls to all three objects you can see that each is a function of another.

Multiple axes can be created on the same canvas.

Multiple marks can be added to the same axes.

Notebooks/nb-11.4-toyplot.ipynb



An explicit call

```
# the data
x = np.random.normal(size=100)
y = np.random.normal(size=100)

# the plot
canvas = toyplot.Canvas(width=300, height=300)
axes = canvas.cartesian()
mark = axes.scatterplot(x, y, size=8)
```

Plotting philosophy and tips:

Showing all data vs binning/summarizing

The first plot is explicit and clear about how the data support the model fit (line).

The second plot is true, but is less clear about the actual data underlying the relationship. It suggests there is much less variance in the data.

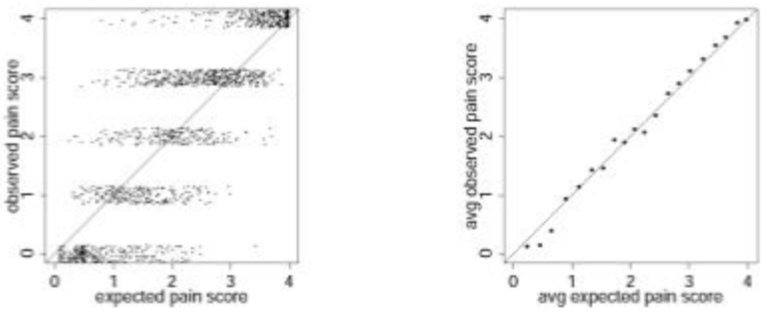


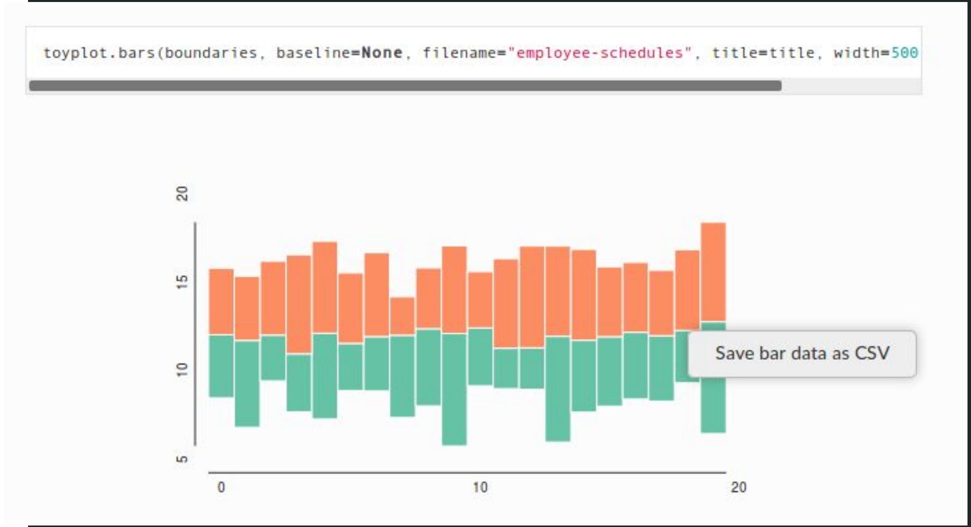
Figure B.6 *(a) Observed versus expected pain relief scores (0 = no pain relief, ..., 5 = complete pain relief) for data from the analysis of Sheiner, Beal, and Dunne (1997). Observed pain relief scores are jittered. (b) Average observed versus averaged expected pain relief scores, with data divided into 20 equally sized bins defined by ranges of expected pain relief scores.*

Showing all data vs binning/summarizing

Toyplot takes the philosophical approach of believing data should generally be as visible as possible, and explicit (you can click or hover on it for more details).

In fact, toyplot has features that can allow you to right-click on a barplot and download the raw data table.

Other libraries, like seaborn for example, provide lots of functions for summarizing data (binning, kernel smoothing, violin plots).



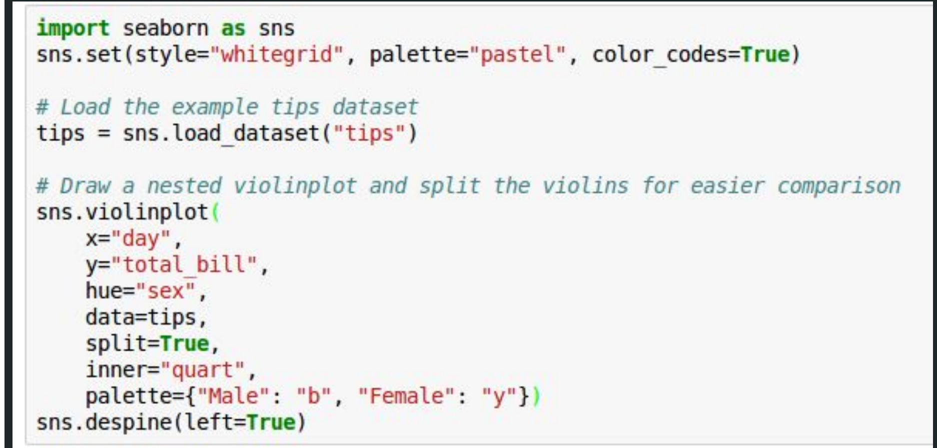
```
toyplot.bars(boundaries, baseline=None, filename="employee-schedules", title=title, width=500
```

Polished - professional quality figures

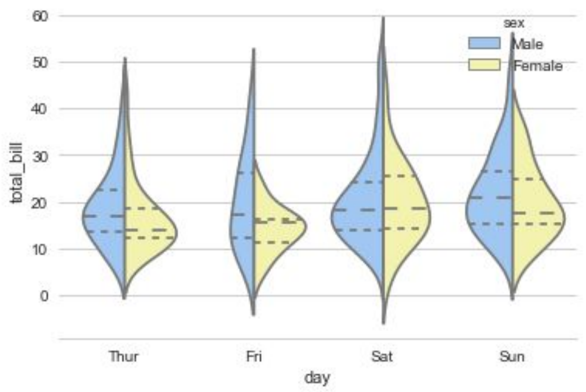
Do not always rely on the default styling. And do not overuse styles that are common.

We can all recognize figures that are from Excel. But default R and matplotlib styling is pretty generic as well.

If it looks like you spent little time on your figures then it might be assumed that you spent little time on your analyses. And thus you lose credibility.



```python
import seaborn as sns
sns.set(style="whitegrid", palette="pastel", color_codes=True)

# Load the example tips dataset
tips = sns.load_dataset("tips")

# Draw a nested violinplot and split the violins for easier comparison
sns.violinplot(
    x="day",
    y="total_bill",
    hue="sex",
    data=tips,
    split=True,
    inner="quart",
    palette={"Male": "b", "Female": "y"})
sns.despine(left=True)
```

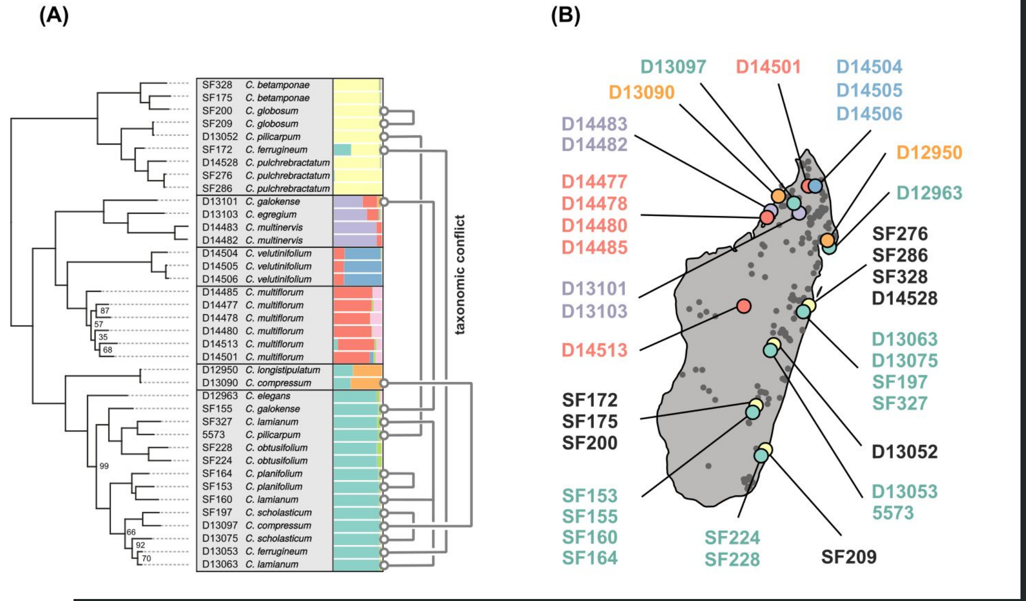


Polished - professional quality figures

Make composite figures. One figure that conveys a lot of information is much more interesting than a single barplot.

Use vector graphics and a vector graphics editor to create your final figures. Examples are Illustrator (expensive) and Inkscape (free).

Add images of your study organism or site, get permissions for images that are not yours.



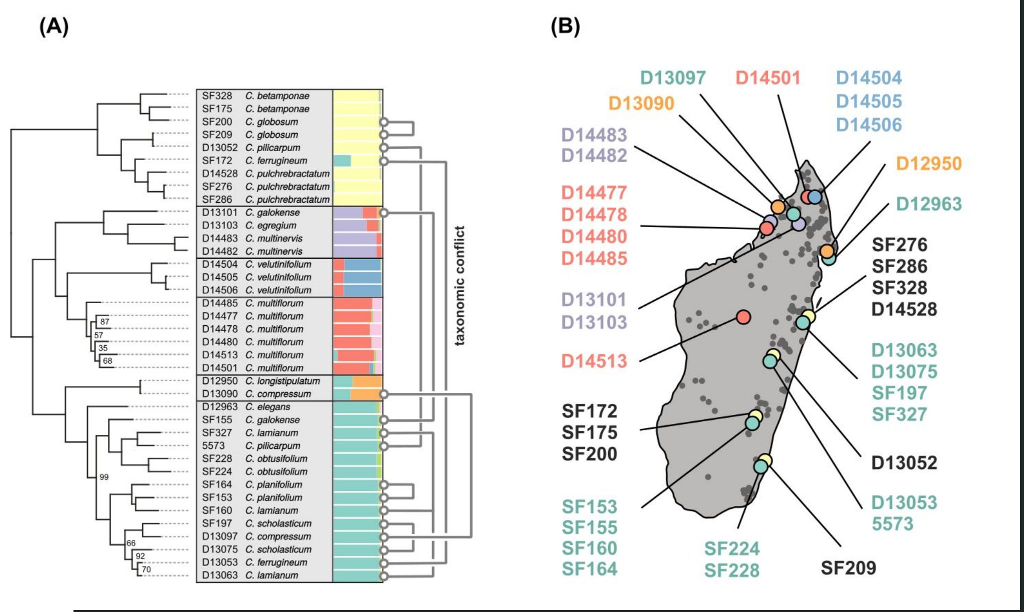
Polished - professional quality figures

For example, in this figure the phylogenetic tree was estimated and the plotted and saved as an SVG file.

The structure barplot was also estimated and plotted as an SVG file.

Both toyplot figures were then imported into Inkscape where I added annotations and the Madagascar map with labeled text added by hand.

The final figure could be saved as an SVG, PDF, and TIFF.



Polished - professional quality figures

Make composite figures. One figure that conveys a lot of information is much more interesting than a single barplot.

Use vector graphics and a vector graphics editor to create your final figures. Examples are Illustrator (expensive) and Inkscape (free).

Add images of your study organism or site, get permissions for images that are not yours.

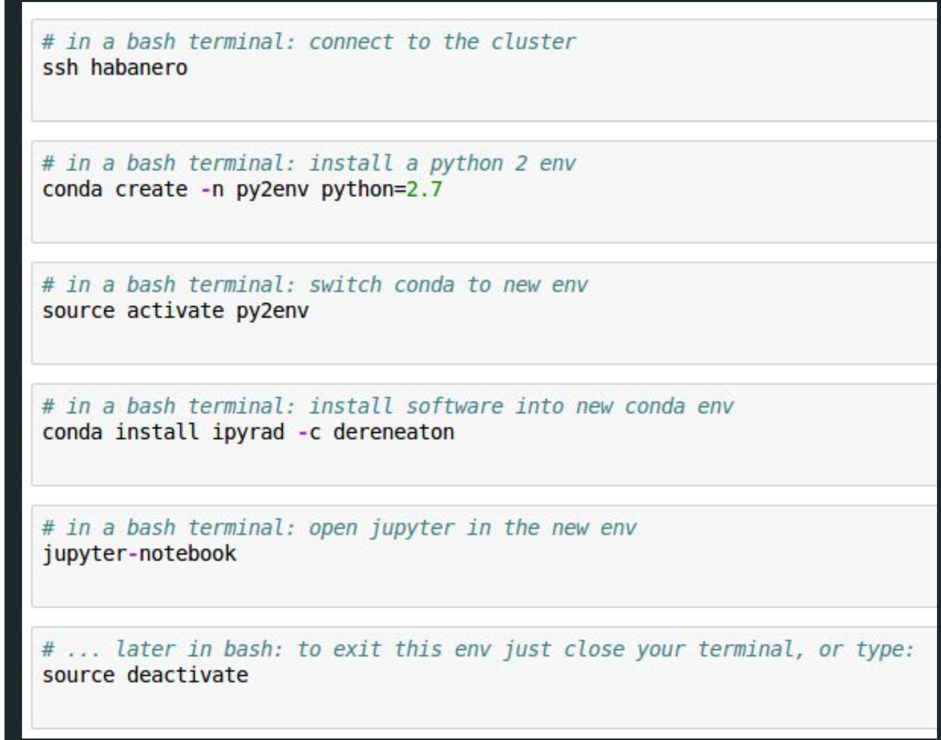
Something different: conda envs

Conda environments

Conda environments allow you to keep different versions of your software separate.

This can be useful if you just want to test out someone else's code that has various dependencies but you do not want to keep those libraries long term. Install into a separate env and delete it when you're done.

Also useful for keeping multiple versions of software. Example, some code only works on Python2 or only on Python3.



```
# in a bash terminal: connect to the cluster
ssh habanero
```

```
# in a bash terminal: install a python 2 env
conda create -n py2env python=2.7
```

```
# in a bash terminal: switch conda to new env
source activate py2env
```

```
# in a bash terminal: install software into new conda env
conda install ipyrad -c dereneaton
```

```
# in a bash terminal: open jupyter in the new env
jupyter-notebook
```

```
# ... later in bash: to exit this env just close your terminal, or type:
source deactivate
```

Assignments notebook is in assignments/
Assigned reading in syllabus:

Assignment: Link to Session 11 repo
Readings: See syllabus
Collaborate: Work together in this gitter chatroom