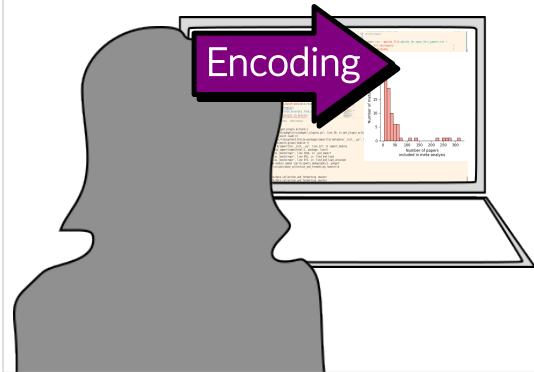




Introduction to Data Visualization

Part 2: Encoding



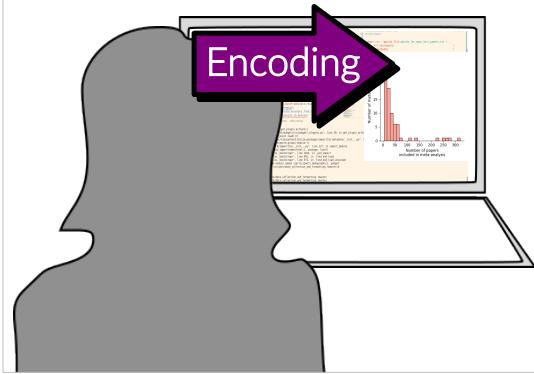
Kendra Oudyk

Welcome to introduction to data visualization, part 2: encoding, where we'll talk about programming data visualizations in Python.



Introduction to Data Visualization

Part 2: Encoding

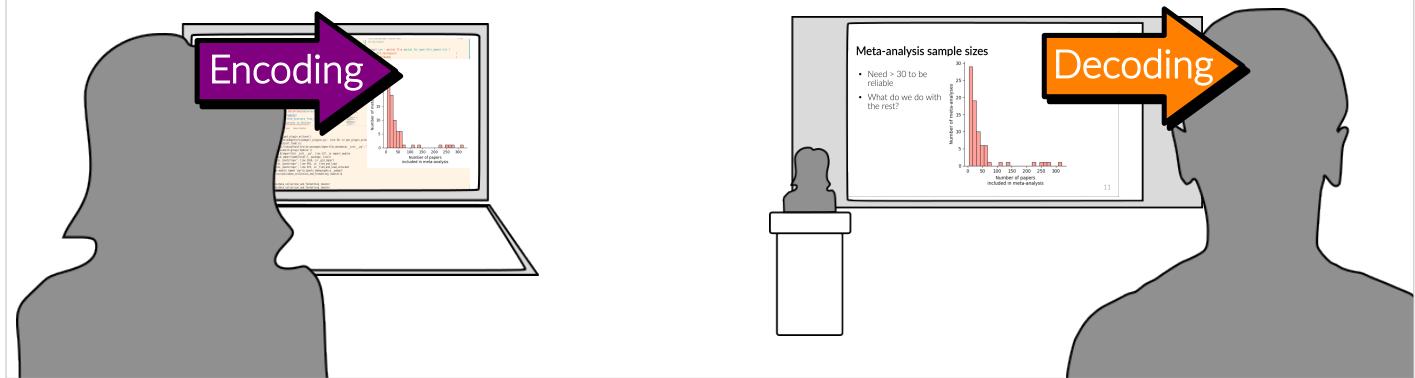


Kendra Oudyk

Welcome to introduction to data visualization, part 2: encoding, where we'll talk about programming data visualizations in Python.

Goal

Use principles of visual **encoding** and **decoding**
to **efficiently** create visualizations
that are **effective** and **reproducible**



In part 1 of this lecture, we talked about Decoding, and I'll just go over the difference again here.

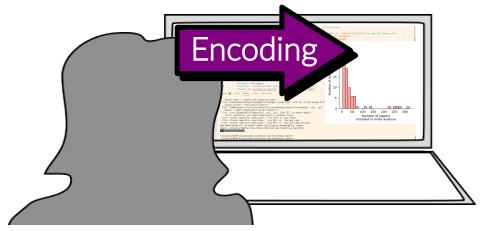
Decoding happens when someone sees your figure and understands something about your data.

Encoding happens when you write code to create the figure according to your plan.

In these lectures, I hope you will learn to
* use principles of visual encoding and decoding
To efficiently create visualizations
That are effective and reproducible.

Understanding decoding will help you make effective figures

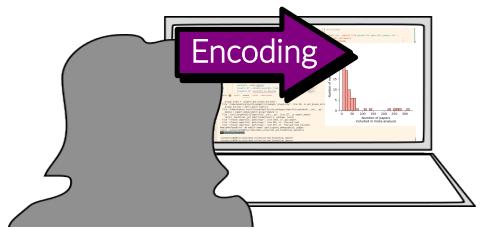
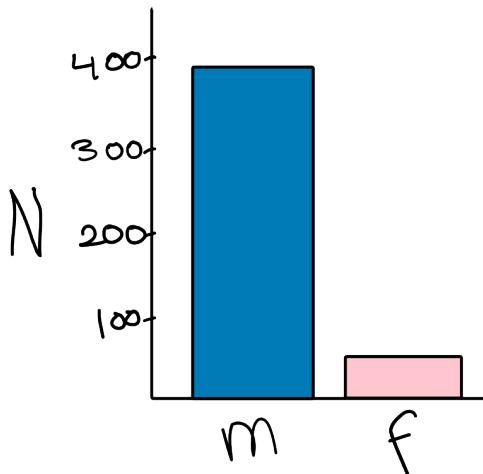
Understanding encoding will help you make figures more reproducibly and efficiently



To program a figure efficiently and reproducibly, we should understand

- Some computer graphics
 - How the computer makes figure
- Matplotlib basics
 - What you and matplotlib need to do
- Higher-level libraries
 - More complex visualizations

In order to program a figure efficiently and reproducibly, it helps to understand...



To program a figure efficiently and reproducibly, we should understand

- Some computer graphics
 - How the computer makes figure
- Matplotlib basics
 - What you and matplotlib need to do
- Higher-level libraries
 - More complex visualizations

But before getting into these topics, we're going to start by making a really simple chart in matplotlib.

If you're completely new to matplotlib, don't expect to understand the code right away, as we'll continue to unpack it as we go through different topics.

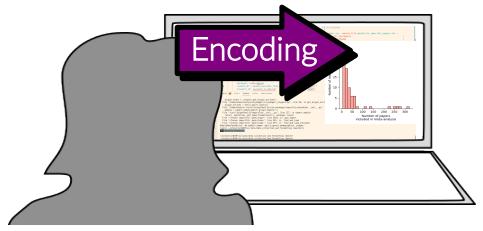
In the last lecture, we used principles of decoding to create this plan for a figure.

Now let's see how to encode this figure in python

go to Python

6

****IN PYTHON****



To program a figure efficiently and reproducibly, we should understand

- Some computer graphics
 - How the computer makes figure
- Matplotlib basics
 - What you and matplotlib need to do
- Higher-level libraries
 - More complex visualizations

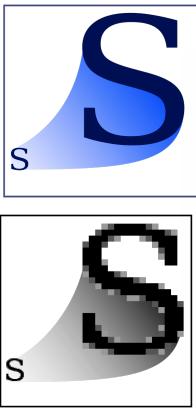
File formats

JPG | **GIF** | **PNG** | **SVG**

First, let's start with something that's probably familiar to most of us – some common file formats for images.

File formats

	JPG	GIF	PNG	SVG
Mathematically-defined shapes	VECTOR			
Pixels	RASTER	✓	✓	✓



9

<https://www.pagecloud.com/blog/web-images-png-vs-jpg-vs-gif-vs-svg>

One of the main defining features of each is whether they are constructed from mathematically-defined shapes or from pixels, that is, whether it's a vector or a raster-based image.

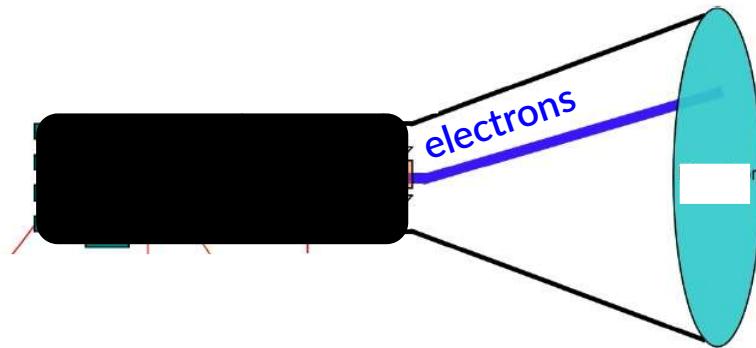
The obvious difference between these types of figures is that, when you zoom in on a raster-based image, it looks blurry.

But Vector images are just as clear no matter how zoomed in you are.

Generally, vector-based images are better if you want a good quality image in a paper, a slideshow, and a poster. It'll look just as clear.

But if the figure is very complicated, the file can become very big.

Cathode ray tube



10 https://www.tutorialspoint.com/computer_graphics/computer_graphics_quick_guide.htm

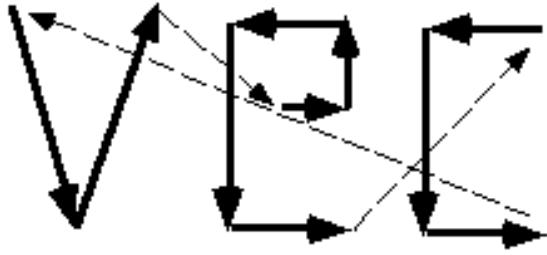
To understand why these types of files are different, we need to briefly think about how images are constructed on a screen.

This is a cathode ray tube.

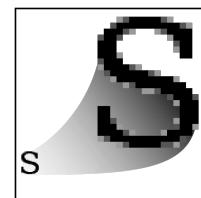
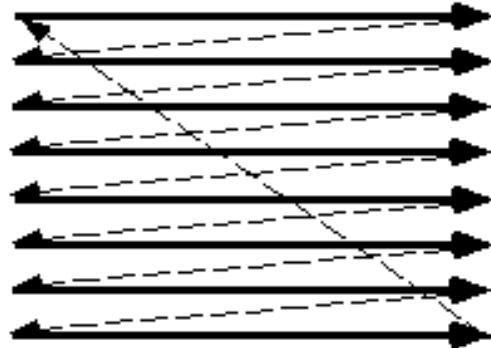
The complicated insides aren't important for our discussion.

* the main thing to take away is that there are electrode guns in the tube that shoot electrons at the screen.

Vector scan



Raster scan



11

<https://www.cs.uic.edu/~jbell/CourseNotes/ComputerGraphics/VectorVsRaster.html>

For a vector graphic, the electron beam creates the image by tracing the vector equations.

For a raster image, the beam scans across each row of pixels on the screen.

****VSC****

Save image as .png and .svg

Open both figures, zoom it

INKSCAPE

Save basic bar chart moved slightly

Do git diff on svgs

Do git diff on pngs

File formats

	JPG	GIF	PNG	SVG
VECTOR				✓
RASTER	✓	✓	✓	
TRANSPARENCY			✓	✓
ANIMATION		✓	✓	✓
LOSSY	✓			

14

<https://www.pagecloud.com/blog/web-images-png-vs-jpg-vs-gif-vs-svg>

Here are a few more differences. We won't go into why, but it's good to know this so you can choose the best format for your data visualizations

Only PNG and SVG files allow for transparency.

And GIF, PNG, and SVG can be used to create animations

The JPG is lossy. Which means that you lose information when an image is stored in this format.

Color: salient but complicated



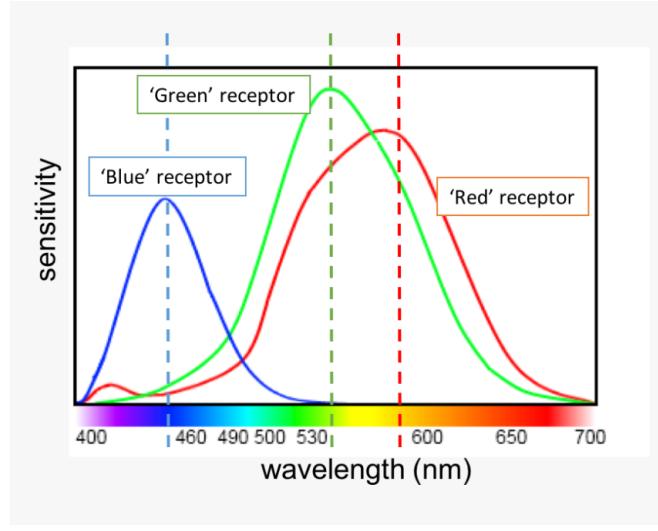
15

Now let's talk about color again.

We've already discussed how color can be useful for encoding shapes, patterns, and groupings.

To round off that discussion, let's see how computers make color.

RGB: Types of photoreceptors in the retina

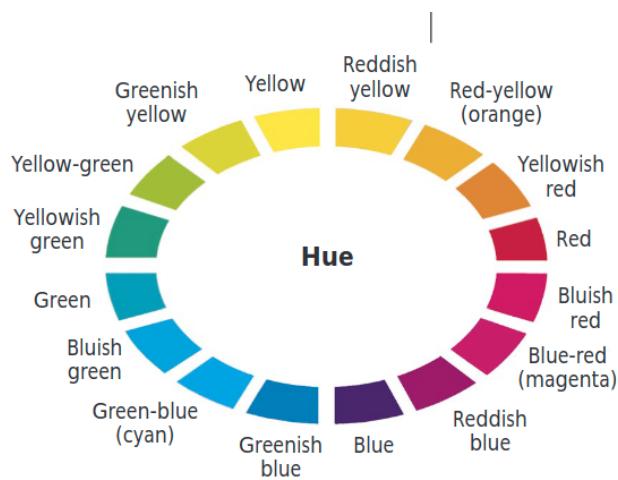


16 <https://www.simplypsychology.org/what-is-the-trichromatic-theory-of-color-vision.html>

I'm going back a bit further in the process of color perception, because I find it fascinating that both computers and our photoreceptors use red, green, and blue as the basis of color.

- * In the back of our eyes are photoreceptors that respond preferentially to light of different wavelengths: red, blue, and green light.

Hue: how we casually talk about color



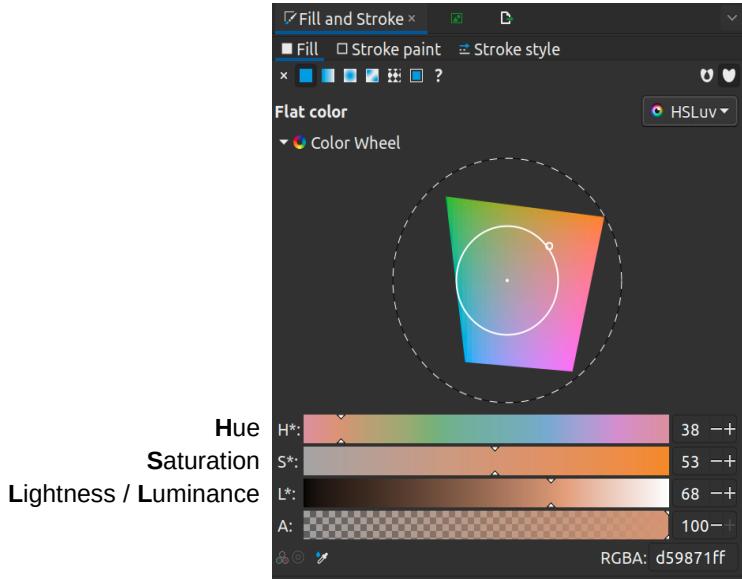
17

<https://www.enr.colostate.edu/ECE666/Handouts/WritingPapers/UsingColorEffectively.pdf>

Let's move up to cognition.

We usually talk about colors as named hues.

HSLuv: how we talk about color perception



18

But to cover the full range of colors, we need to discuss saturation and lightness as well as Hue.

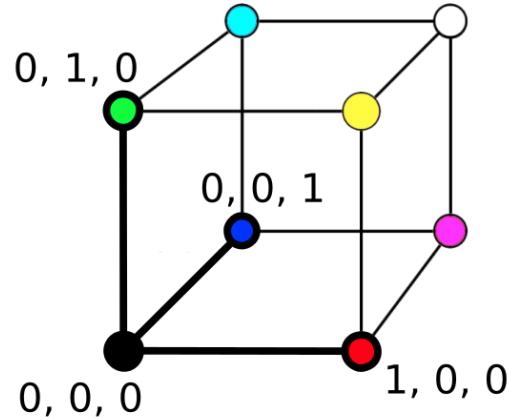
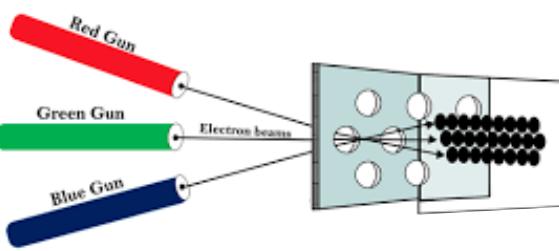
This is the HSL paletter.

In particular, here we see the HSLuv pallett, also known as $H^*S^*L^*$..

It has been corrected for our uneven perception of lightness over the hue spectrum.

Our last lecture covered this more in depth.

RGB: How a computer produces color



19 <https://www.javatpoint.com/computer-graphics-color-crt-monitors>

But a computer goes fully back to the basics, producing color through beams of electrons with frequencies in the red, green, and blue ranges of the spectrum.

* Here is the cathode ray tube again, showing how there are electron guns for the three different wavelengths.

The RGB palette can be thought of as a three-dimensional space, where each color (including its hue, saturation, and lightness) is made up of different amounts of these three frequencies.

* This can be visualized as a cube.

I find this fascinating. The three electron guns kind of match our three cone receptor types.

But the way colors are produced from mixing these three channels is very unintuitive. Let me demonstrate

Go to inkscape

****IN INKSCAPE****

Let's see how the RGB palette maps onto this 3D cube.

Try to notice how unintuitive it is, compared to the HSL palette.

*****Demo color cube*****

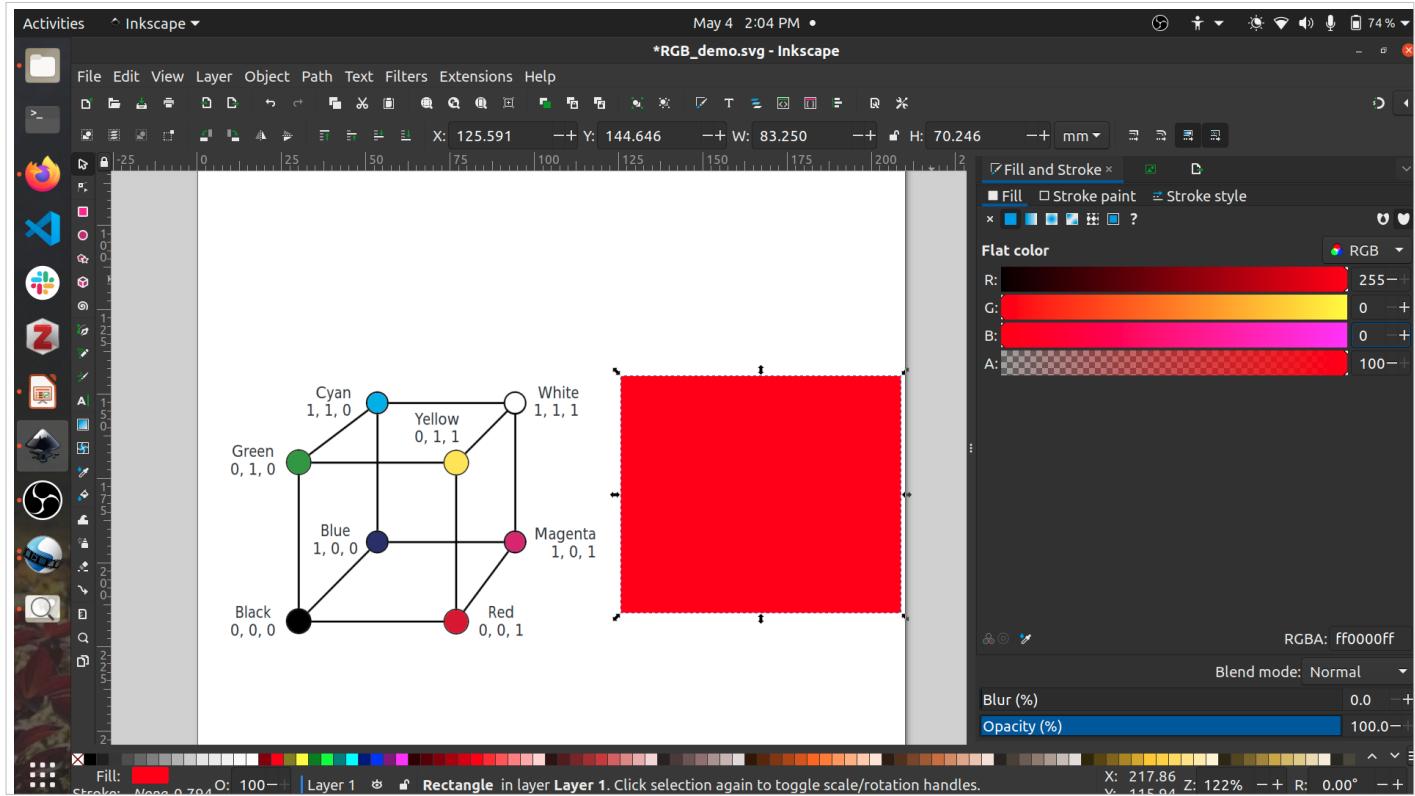
The screenshot shows a web browser displaying the matplotlib.org documentation. The URL in the address bar is <https://matplotlib.org/stable/tutorials/colors/colors.html>. The page title is "Specifying colors". On the left, there is a sidebar titled "Section Navigation" with links to various sections like "Introductory", "Intermediate", "Advanced", "Colors" (which is currently selected), "Text", "Toolkits", and "Provisional". The main content area has a heading "Color formats" and a sub-section "Format". It lists four ways to specify colors:

Format	Example
RGB or RGBA (red, green, blue, alpha) tuple of float values in a closed interval [0, 1].	<code>(0.1, 0.2, 0.5)</code> <code>(0.1, 0.2, 0.5, 0.3)</code>
Case-insensitive hex RGB or RGBA string.	<code>#0f0f0f</code> <code>#0f0f0f80</code>
Case-insensitive RGB or RGBA string equivalent hex shorthand of duplicated	<code>'abc'</code> as <code>'aabbcc'</code> <code>'fb1'</code> as <code>'ffbb11'</code>

On the right side of the page, there is a sidebar titled "On this page" with links to "Color formats", "Transparency", "Color selection", "Comparison between X11/CSS4 and xkcd colors", and "X11/CSS4 and xkcd colors". The page is set to "stable" version and has a 90% zoom level.

21

If you go to the matplotlib documentation, you can see the different ways that it accepts color specifications.



IN INKSCAPE

*If you chose a color in HSL,
You could go to the RGB palette and copy and paste
its hex code.*

Here's how the RGB scales map onto the hex code.

*The first two characters are for red, the third and fourth
for green, the fifth and 6th for blue, and the last two
are for alpha, or the transparency.*

*When I change the red scale by one, the hexadecimal
character for that scale changes by one.*

If we enter colors by name in matplotlib,
It will still encode them as RGB behind the scenes.

SVC

Ax
Containers
0 bar container
0 rectangle
_facecolor

TL;DR

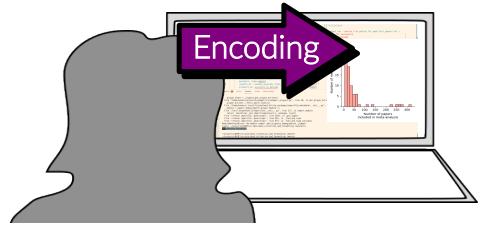
- Images have different properties/features because of **how they're made**
- The way a computer produces color (**RGB**) is **not intuitive**
 - You'll probably pick colors using named colors or an HSL tool

24

So, in summary of this very brief intro to computer graphics,

Images...

And the way...



To program a figure efficiently and reproducibly, we should understand

- Some computer graphics
 - How the computer makes figure
- Matplotlib basics
 - What you and matplotlib need to do
- Higher-level libraries
 - More complex visualizations

Now let's go over some matplotlib basics.
Matplotlib is a python library for creating visualizations.

We're going to go behind the scenes a bit to see how matplotlib works.

The important thing to remember is what you have to do.

But I hope it will be easier to remember what to do and which help to follow, if you understand a bit about what matplotlib is doing.

Activities Firefox Web Browser • May 4 2:37 PM • 80% 61% ▾

https://stackoverflow.com/questions/19125722/adding-a-matplotlib-legend

Adding a matplotlib legend

Asked 9 years, 7 months ago Modified 29 days ago Viewed 1.2m times

How can one create a legend for a line graph in Matplotlib's `PyPlot` without creating any extra variables?

469 Please consider the graphing script below:

```
if __name__ == '__main__':
    PyPlot.plot(total_lengths, sort_times_bubble, 'b-',
                total_lengths, sort_times_ins, 'r-',
                total_lengths, sort_times_merge_r, 'g+',
                total_lengths, sort_times_merge_i, 'p-' )
    PyPlot.title("Combined Statistics")
    PyPlot.xlabel("Length of list (number)")
    PyPlot.ylabel("Time taken (seconds)")
    PyPlot.show()
```

As you can see, this is a very basic use of `matplotlib`'s `PyPlot`. This ideally generates a graph like the one below:

Combined Statistics

The Overflow Blog

- ✓ Don't panic! A playbook for managing any production incident
- ✓ How to land a job in climate tech

Featured on Meta

- New blog post from our CEO Prashanth: Community is the future of AI
- We are updating our Code of Conduct and we would like your feedback
- Temporary policy: ChatGPT is banned
- Content Discovery initiative April 13 update Related questions using ...
- The [connect] tag is being burninated
- Removal of the Census badge

Linked

0 Add legends in pyplot with Pandas Dataframe data

Join Stack Overflow to find the best answer to your technical question, help others answer theirs.

Sign up with email | Sign up with Google | Sign up with GitHub | Sign up with Facebook

If you've ever used matplotlib, and ever looked up how to do something, you might have noticed that there's usually more than one way to accomplish something.

Here, someone is asking on Stack Overflow how to add a matplotlib legend.

Activities Firefox Web Browser • May 4 2:41 PM • 60% ▾

open shot inc Capability to Artist tutorial free clipart Art Artist Cra python - Ad X

https://stackoverflow.com/questions/19125722/adding-a-matplotlib-legend

80% Log in Sign up

Here's an example to help you out ...

27

```
fig = plt.figure(figsize=(10,5))
ax = fig.add_subplot(111)
ax.set_title('ADR vs Rating (CS:GO)')
ax.scatter(x=data[:,0],y=data[:,1],label='Data')
plt.plot(data[:,0], m*data[:,0] + b,color='red',label='Our Fitting Line')
ax.set_xlabel('ADR')
ax.set_ylabel('Rating')
ax.legend(loc='best')
plt.show()
```

ADR vs Rating (CS:GO)

Share Improve this answer Follow answered Dec 6, 2017 at 7:00 Akash Kandpal

Join Stack Overflow to find the best answer to your technical question, help others answer theirs.

Sign up with email Sign up with Google Sign up with GitHub Sign up with Facebook

This person suggested using `ax.legend()`

Activities Firefox Web Browser • May 4 2:42 PM • 59%

open shot inc Capability to Artist tutorial free clipart Art Artist Cra python - Ac X + ×

← → ↻ https://stackoverflow.com/questions/19125722/adding-a-matplotlib-legend 80% ☆

Log in Sign up

stack overflow About Products For Teams Search... Add a comment

Home PUBLIC Questions Tags Users Companies COLLECTIVES Explore Collectives TEAMS Stack Overflow for Teams – Start collaborating and sharing organizational knowledge. Create a free Team Why Teams?

You can access the Axes instance (`ax`) with `plt.gca()`. In this case, you can use

62 `plt.gca().legend()`

You can do this either by using the `label=` keyword in each of your `plt.plot()` calls or by assigning your labels as a tuple or list within `legend`, as in this working example:

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(-0.75, 1, 100)
y0 = np.exp(2 + 3*x - 7*x**3)
y1 = 7-4*np.sin(4*x)
plt.plot(x,y0,x,y1)
plt.gca().legend(['y0', 'y1'])
plt.show()
```

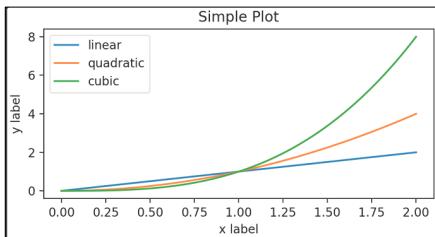
Join Stack Overflow to find the best answer to your technical question, help others answer theirs.

Sign up with email Google Sign up with GitHub Facebook Sign up with Facebook

Whereas this person suggested plt.gca().legend()

Both of these commands access the same objects behind the scenes, but they demonstrate different coding styles.

Coding styles



Explicit (object-oriented)

```
x = np.linspace(0, 2, 100) # Sample data.  
  
# Note that even in the OO-style, we use `pyplot.figure` to create the Figure.  
fig, ax = plt.subplots(figsize=(5, 2.7), layout='constrained')  
ax.plot(x, x, label='linear') # Plot some data on the axes.  
ax.plot(x, x**2, label='quadratic') # Plot more data on the axes...  
ax.plot(x, x**3, label='cubic') # ... and some more.  
ax.set_xlabel('x label') # Add an x-label to the axes.  
ax.set_ylabel('y label') # Add a y-label to the axes.  
ax.set_title("Simple Plot") # Add a title to the axes.  
ax.legend() # Add a legend.
```

Implicit

```
x = np.linspace(0, 2, 100) # Sample data.  
  
plt.figure(figsize=(5, 2.7), layout='constrained')  
plt.plot(x, x, label='linear') # Plot some data on the (implicit) axes.  
plt.plot(x, x**2, label='quadratic') # etc.  
plt.plot(x, x**3, label='cubic')  
plt.xlabel('x label')  
plt.ylabel('y label')  
plt.title("Simple Plot")  
plt.legend()
```

31

https://matplotlib.org/stable/tutorials/introductory/quick_start.html

You can use explicit or implicit coding styles in matplotlib.

The explicit style is object-oriented and is most transparent because you're directly, or explicitly, interacting with the objects.

The implicit style is more similar to matlab.

It can do the same things, in most cases, but you don't interact directly with the objects.

**Here are two pieces of code that create the same figure.

* We can see the differences if we focus on the beginnings of the line.

In the explicit style, they define an object called ax and then adjust it explicitly.

In the implicit style, they continue to call the pyplot sublibrary, which tracks the object for the axis behind the scenes.

Some objects

Figure

- The top-level container for all elements of the plot

Axes

- An **Artist** subclass attached to a **Figure**
- Incl. All elements of an individual (sub-)plot

Artist

- Abstract base class
- Everything visible is a subclass of **Artist**

Explicit (object-oriented)

```
x = np.linspace(0, 2, 100) # Sample data.  
  
# Even in the OO-style, we use `plt.subplots(figsize=(5, 2.7), layout='constrained')`  
fig, ax = plt.subplots(figsize=(5, 2.7), layout='constrained')  
ax.plot(x, x, label='linear') # Plot some data on the axes.  
ax.plot(x, x**2, label='quadratic') # Plot more data on the axes...  
ax.plot(x, x**3, label='cubic') # ... and some more.  
ax.set_xlabel('x label') # Add an x-label to the axes.  
ax.set_ylabel('y label') # Add a y-label to the axes.  
ax.set_title("Simple Plot") # Add a title to the axes.  
ax.legend() # Add a legend.
```

32

Let's take a closer look at the important objects.

*The **fig** variable is an instance of the **Figure** object. It is the top-level container for all elements of a plot.

*The **ax** variable is an instance of the **Axes** object. It is a subclass attached to a figure, and it includes all elements of an individual plot.

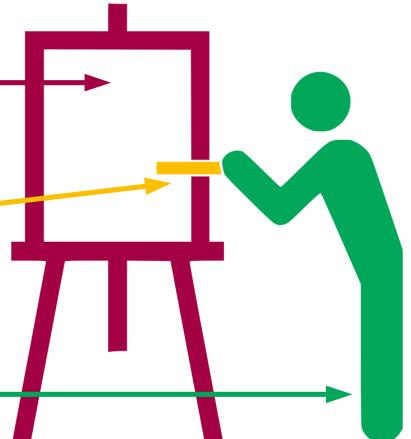
* Both the **Figure** and **Axes** objects are from the **Artist** base class.

Remember, an artist is something that uses a renderer to draw on a canvas.

3 layers of matplotlib

- “matplotlib.backend_bases.FigureCanvas” is the **area** onto which the figure is drawn
 - matplotlib.backend_bases.Renderer is the object which knows how to **draw** on the FigureCanvas
 - matplotlib.artist.Artist is the object that knows how to **use a renderer** to paint onto the canvas.”

What you'll be working with 95% of the time



33 <https://matplotlib.org/stable/tutorials/intermediate/artists.html>

<https://www.svgrepo.com/svg/307645/art-artist-craft-canvas>

Stepping back a bit, there are three main layers of the matplotlib API.

The figureCanvas is the area onto which the figure is drawn

The Renderer is the object that knows how to draw on the figure canvas

The artist is the object that knows how to use a renderer to paint on the canvas

You'll be interacting with Artist objects 95% of the time, and won't have to worry too much about what goes on behind the scenes.

But let's take a look, just so you know what's there

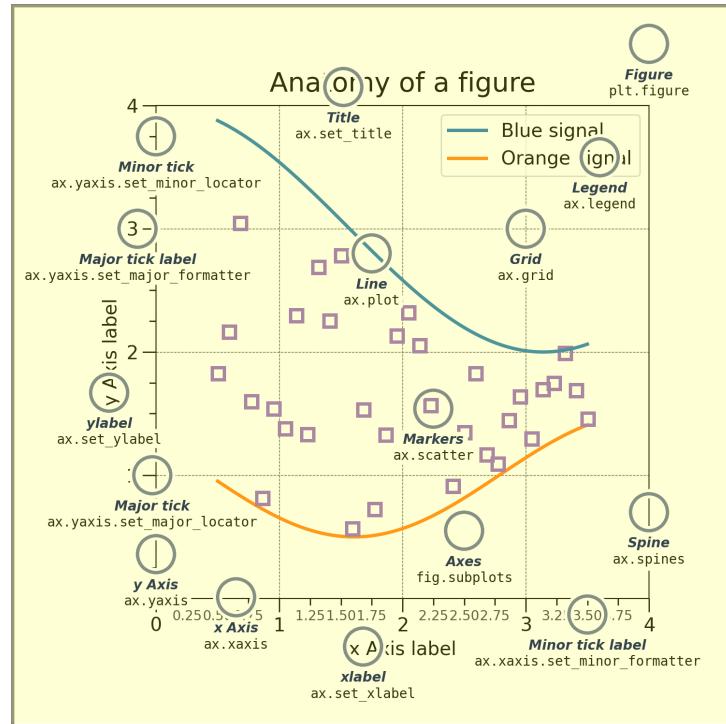
2 types of artists

1) Primatives: things to draw

- Line2D
- Rectangle
- Text
- AxesImage

2) Containers: where to draw them

- Figure
- Axes
- Axis



34

There are two main types of artists: primitives and containers.

Primitives are basic graphical objects
* like lines, rectangles, text, and images

Containers define where to draw the rectangles, etc.

The Figure includes the entire figure area

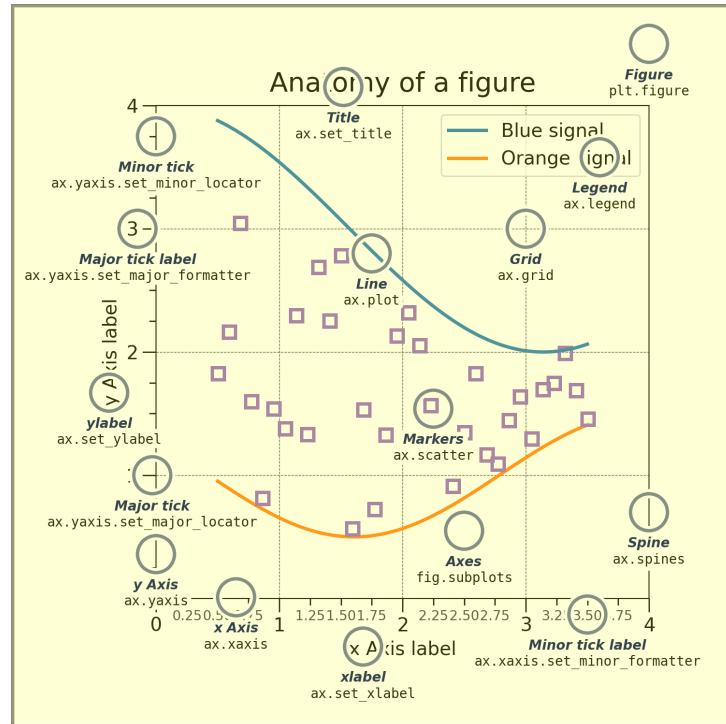
2 types of artists

1) Primatives: things to draw

- Line2D
- Rectangle
- Text
- AxesImage

2) Containers: where to draw them

- Figure
- Axes
- Axis



35

There are two main types of artists: primitives and containers.

Primitives are basic graphical objects
* like lines, rectangles, text, and images

Containers define where to draw the rectangles, etc.

The Figure includes the entire figure area

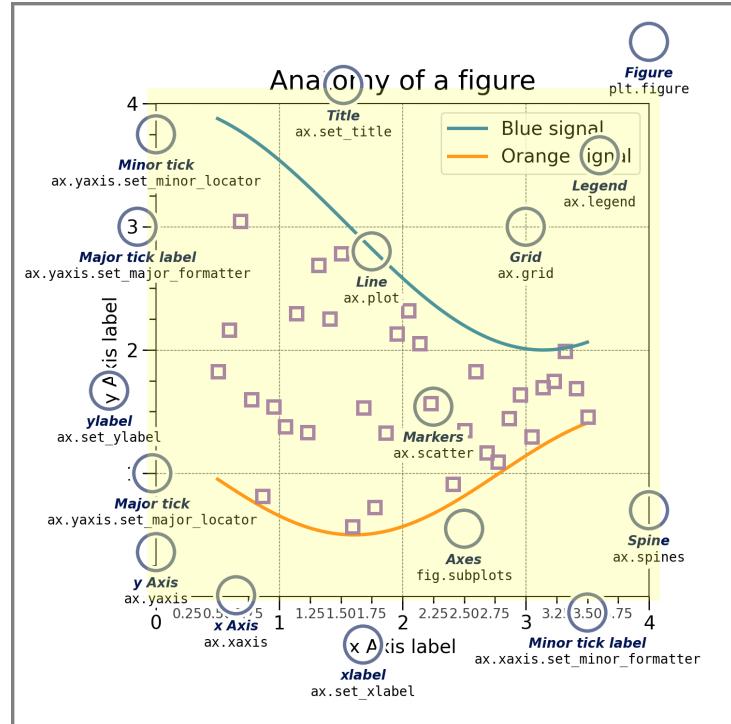
2 types of artists

1) Primatives: things to draw

- Line2D
- Rectangle
- Text
- AxesImage

2) Containers: where to draw them

- Figure
- Axes
- Axis



36

The Axes (with an es) includes all the individual axes and the main plotting area

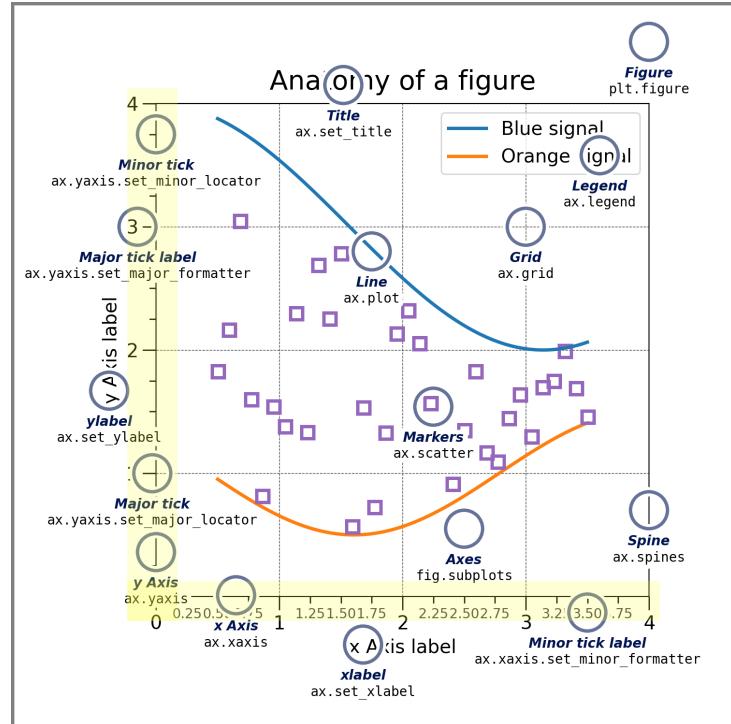
2 types of artists

1) Primatives: things to draw

- Line2D
- Rectangle
- Text
- AxesImage

2) Containers: where to draw them

- Figure
- Axes
- Axis



37

The Axis (with an is) is one individual axis, like the x axis

What actually is an Artist??

Artist class

```
class matplotlib.artist.Artist
```

Abstract base class for objects that render into a FigureCanvas.

Typically, all visible elements in a figure are subclasses of Artist.

38

So, what actually is an artist?

In the docs, Matplotlib describes it as an...

What actually is an Artist??

Artist class

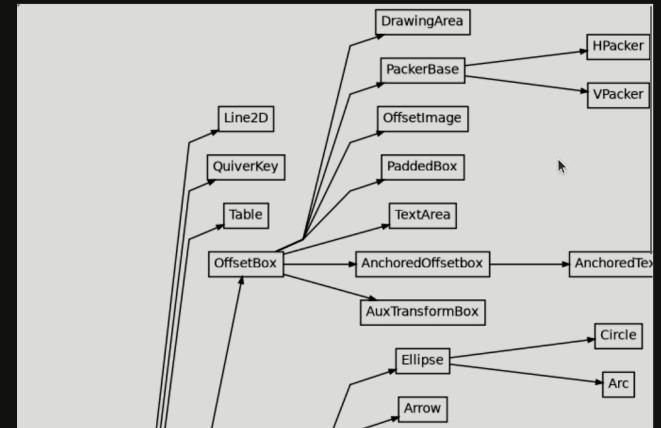
```
class matplotlib.artist.Artist
```

Abstract base class for objects that render into a FigureCanvas.

Typically, all visible elements in a figure are subclasses of Artist.

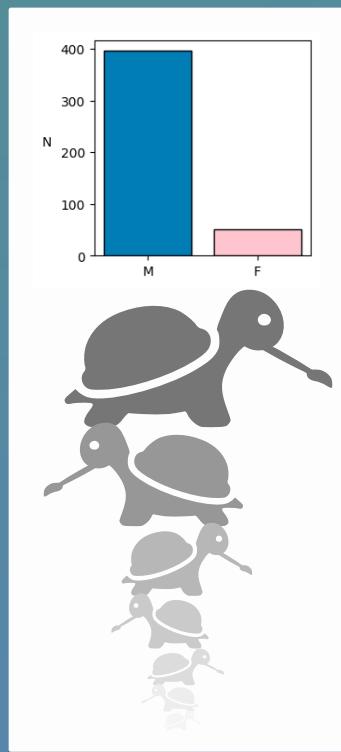
matplotlib.artist

Inheritance Diagrams



TL;DR for our purposes,
it's Artists all the way down

40



But basically, for our purposes, it's artists all the way down.

We'll usually be interacting with artists.

Axes helper methods for plot types

- `ax.plot()` # line graph
- `ax.bar()` # bar graph
- `ax.imshow()` # image

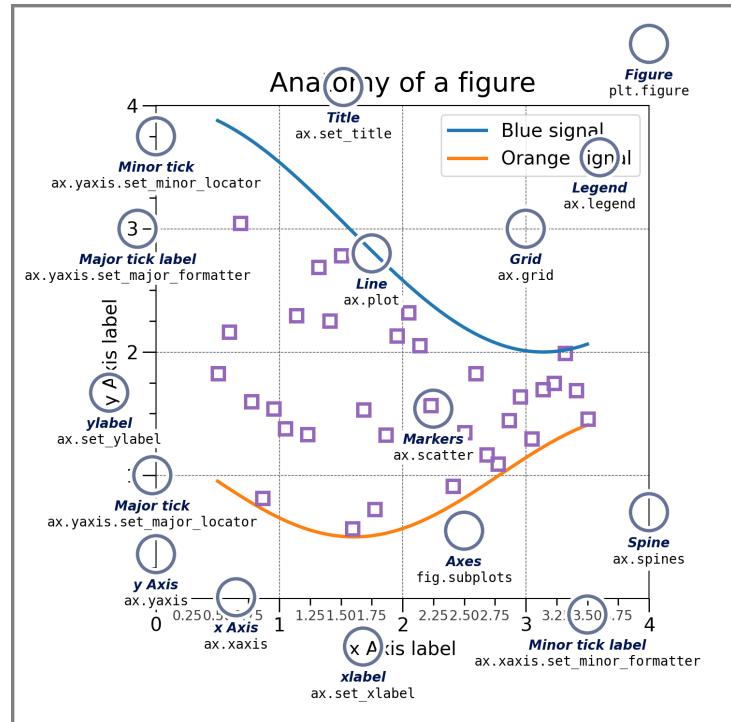
So you don't have to construct figures from shapes

41

On the Axes artis, there are helper methods for different plot types, for example, the line and bar graphs, or showing an image or heatmap.

This is very helpful – hence the name helper method - so you don't have to construct figures from basic lines and shapes.

Helper methods for customization



42

https://matplotlib.org/stable/tutorials/introductory/quick_start.html#sphx-glr-tutorials-introductory-quick-start-py

There are also helper methods for customizing charts.

Here, you can see all the anatomy of a figure, with the labels being the helper methods for editing different aspects of the figure.

Remember, this is the explicit way of coding, where you're interacting directly with the Axes artists objects

Here we see a matplotlib cheatheet, which can be found in their documentation

*I've highlighted where to find the helper methods for different basic plot types

*And the more advanced plots.

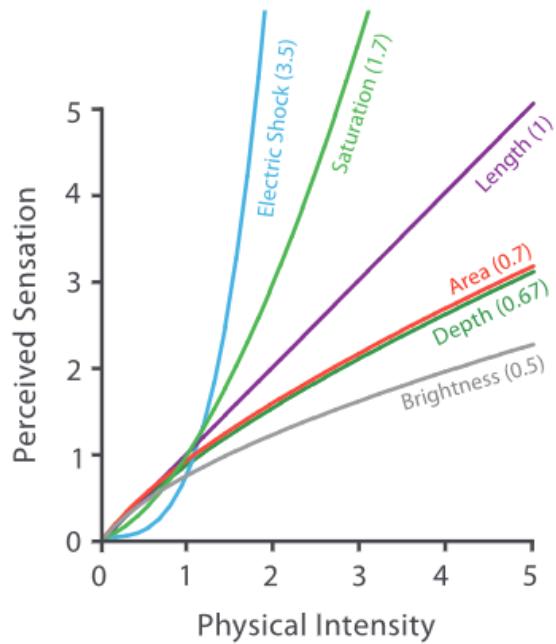
*Here we have the anatomy of a figure again, but they label parts with the names instead of the commands for changing them.

But you can find the commands here *, and they're fairly intuitive from their names.

Live coding

- Replicating this figure

Steven's Psychophysical Power Law: $S = I^N$

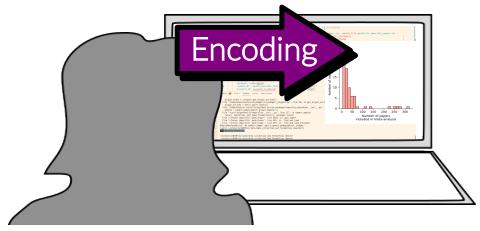


Munzner, T. (2014). Visualization analysis and design. CRC press.

44

MATPLOTLIB DEMO

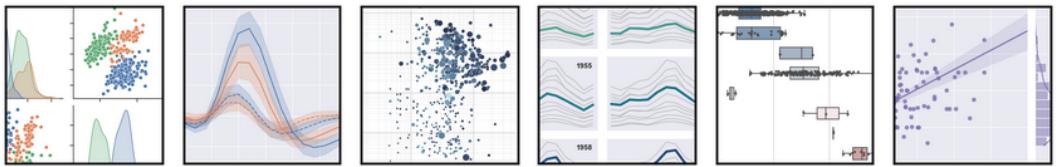
Re-create line graph from last lecture



To program a figure efficiently and reproducibly, we should understand

- Some computer graphics
 - How the computer makes figure
- Matplotlib basics
 - What you and matplotlib need to do
- Higher-level libraries
 - More complex visualizations

seaborn: statistical data visualization



Seaborn is a Python data visualization library based on [matplotlib](#). It provides a high-level interface for drawing attractive and informative statistical graphics.

Features of Seaborn

- Works well with `pandas.DataFrame()` objects
- Default colors are from **perceptually uniform** pallettes
- Can **calculate stats** as well as visualize
- More easily visualize **complex data** (e.g, multivariate)

 Search[Quickstart](#)[Examples](#)[User guide](#)[1. Introduction](#)[2. What is nilearn?](#)[3. Using nilearn for the first time](#)[4. Machine learning applications to Neuroimaging](#)[5. Decoding and MVPA: predicting from brain images](#)[6. Functional connectivity](#)

7. Plotting brain images

In this section, we detail the general tools to visualize neuroimaging volumes and surfaces with nilearn.

Nilearn comes with plotting function to display brain maps coming from Nifti-like images, in the `nilearn.plotting` module.

[Code examples](#)

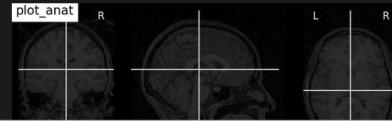
Nilearn has a whole section of the example gallery on plotting.

A small tour of the plotting functions can be found in the example [Plotting tools in nilearn](#).

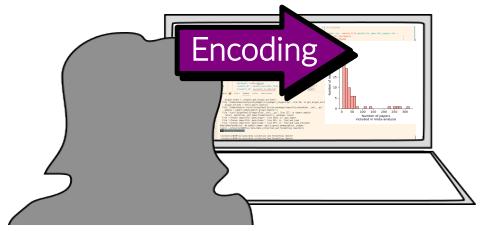
Finally, note that, as always in the nilearn documentation, clicking on a figure will take you to the code that generates it.

7.1. Different plotting functions

Nilearn has a set of plotting functions to plot brain volumes that are fine tuned to specific applications. Amongst other things, they use different heuristics to find cutting coordinates.



`plot_anat`
Plotting an anatomical image



To program a figure efficiently and reproducibly, we should understand

- **Some computer graphics**
 - How the computer makes figure
- **Matplotlib basics**
 - What you and matplotlib need to do
- **Higher-level libraries**
 - More complex visualizations