LEGENDS

Plot legends give meaning to a visualization, assigning labels to the various plot elements. We previously saw how to create a simple legend; here we'll take a look at customizing the placement and aesthetics of the legend in Matplotlib.

Plot legends give meaning to a visualization, assigning labels to the various plot elements. We previously saw how to create a simple legend; here we'll take a look at customizing the placement and aesthetics of the legend in Matplotlib

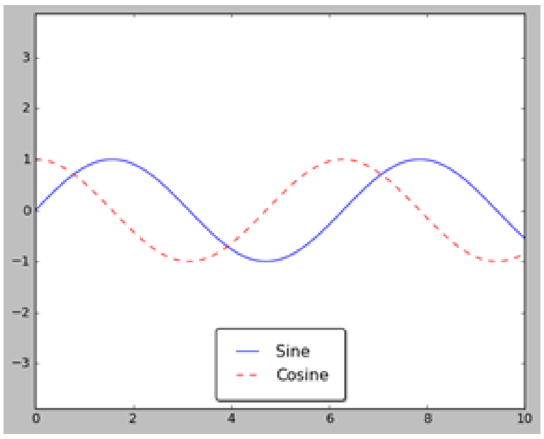
```
plt.plot(x, np.sin(x), '-b', label='Sine')
plt.plot(x, np.cos(x), '--r', label='Cosine')
plt.legend();
```

Choosing Elements for the Legend

- The legend includes all labeled elements by default. We can change which elements and labels appear in the legend by using the objects returned by plot commands.
- The plt.plot() command is able to create multiple lines at once, and returns a list of created line instances. Passing any of these to plt.legend() will tell it which to identify, along with the labels we'd like to specify

```
y = np.sin(x[:, np.newaxis] + np.pi * np.arange(0, 2, 0.5))
lines = plt.plot(x, y)
plt.legend(lines[:2],['first','second']);

# Applying label individually.
plt.plot(x, y[:, 0], label='first')
plt.plot(x, y[:, 1], label='second')
plt.plot(x, y[:, 2:])
plt.legend(framealpha=1, frameon=True);
```



Multiple legends

It is only possible to create a single legend for the entire plot. If you try to create a second legend using plt.legend() or ax.legend(), it will simply override the first one. We can work around this by creating a new legend artist from scratch, and then using the lower-level ax.add_artist() method to manually add the second artist to the plot

Example

```
import matplotlib.pyplot as plt
plt.style.use('classic')
import numpy as np
x = np.linspace(0, 10, 1000)
ax.legend(loc='lower center', frameon=True, shadow=True,borderpad=1,fancybox=True)
fig
```

COLOR

In Matplotlib, a color bar is a separate axes that can provide a key for the meaning of colors in a plot. For continuous labels based on the color of points, lines, or regions, a labeled color bar can be a great tool. The simplest colorbar can be created with the plt.colorbar() function.

Customizing Colorbars

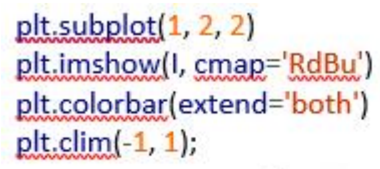
Choosing color map.

We can specify the colormap using the cmap argument to the plotting function that is creating the visualization. Broadly, we can know three different categories of colormaps:

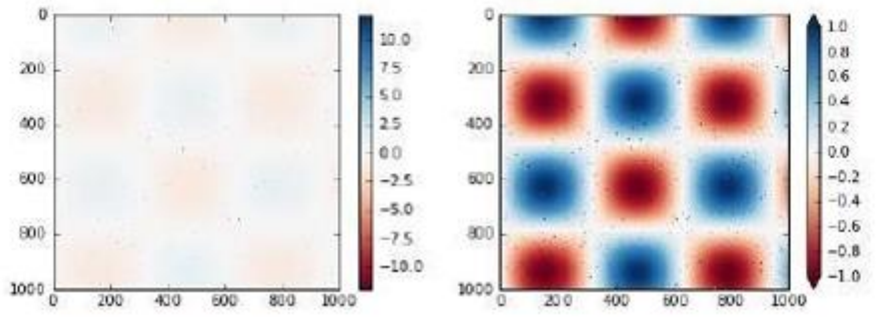
- Sequential colormaps - These consist of one continuous sequence of colors (e.g., binary or viridis).
- Divergent colormaps - These usually contain two distinct colors, which show positive and negative deviations from a mean (e.g., RdBu or PuOr).
- Qualitative colormaps - These mix colors with no particular sequence (e.g., rainbow or jet).

Color limits and extensions

- Matplotlib allows for a large range of colorbar customization. The colorbar itself is simply an instance of plt.Axes, so all of the axes and tick formatting tricks we've learned are applicable.
- We can narrow the color limits and indicate the out-of-bounds values with a triangular arrow at the top and bottom by setting the extend property.



```
plt.subplot(1, 2, 2)
plt.imshow(I, cmap='RdBu')
plt.colorbar(extend='both')
plt.clim(-1, 1);
```



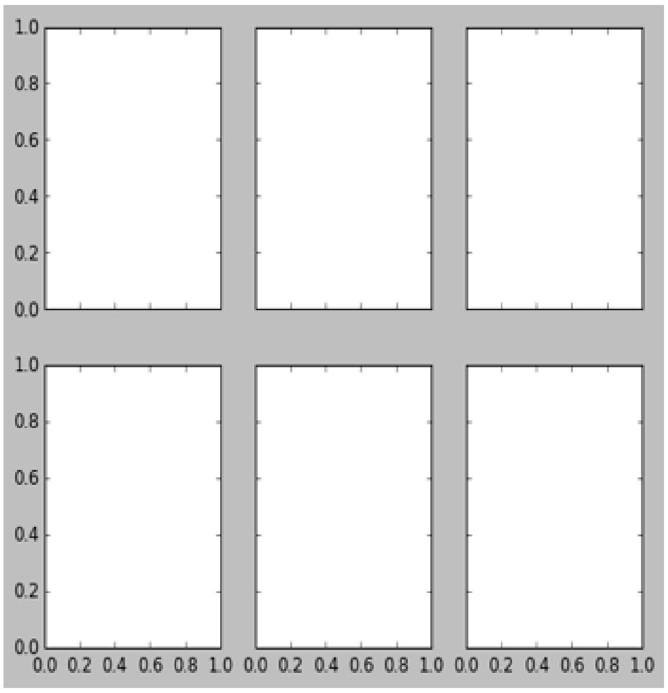
SUBPLOTS

- Matplotlib has the concept of subplots: groups of smaller axes that can exist together within a single figure.
- These subplots might be insets, grids of plots, or other more complicated layouts.
- We'll explore four routines for creating subplots in Matplotlib.
  1. plt.axes: Subplots by Hand
  2. plt.subplot: Simple Grids of Subplots
  3. plt.subplots: The Whole Grid in One Go
  4. plt.GridSpec: More Complicated Arrangements

plt.subplots: The Whole Grid in One Go

- The approach just described can become quite tedious when you're creating a large grid of subplot especially if you'd like to hide the x- and y-axis labels on the inner plots.
- For this purpose, plt.subplots() is the easier tool to use (note the s at the end of subplots).
- Rather than creating a single subplot, this function creates a full grid of subplots in a single line, returning them in a NumPy array.
- The arguments are the number of rows and number of columns, along with optional keywords sharex and sharey, which allow you to specify the relationships between different axes.
- Here we'll create a 2×3 grid of subplots, where all axes in the same row share their y-axis scale, and all axes in the same column share their x-axis scale

```
fig, ax = plt.subplots(2, 3, sharex='col', sharey='row')
```



Note that by specifying sharex and sharey, we've automatically removed inner labels on the grid to make the plot cleaner.