Object Oriented Programming (Using Python)

UNIT- III

Python Libraries:

- Matplotlib cont’d.
  - Coding Styles
  - Styling Artists
  - Labelling plots
  - Axis scales and ticks
  - Color mapped data

Prof. R. MADANA MOHANA
Professor, Artificial Intelligence & Data Science
http://rmadanamohana.com/

matplotlib - Quick start guide

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

Coding Styles

The explicit and the implicit interfaces

There are essentially two ways to use Matplotlib:

- Explicitly create Figures and Axes, and call methods on them (the "object-oriented (OO) style").
- Rely on `pyplot` to implicitly create and manage the Figures and Axes, and use `pyplot` functions for plotting.

Coding Styles

The explicit and the implicit interfaces

Example of Object-oriented (OO) style:

```
>>> import matplotlib.pyplot as plt
# 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.
[<matplotlib.lines.Line2D object at 0x000001417FE6F5B0>]
```

Coding Styles

The explicit and the implicit interfaces

Example of Object-oriented (OO) style: cont'd.

```
>>> ax.plot(x, x**2, label='quadratic')  # Plot more data on the axes...
[<matplotlib.lines.Line2D object at 0x000001417FE6F880>]
>>> ax.plot(x, x**3, label='cubic')  # ... and some more.
[<matplotlib.lines.Line2D object at 0x000001417FE6FB20>]
>>> ax.set_xlabel('x label')  # Add an x-label to the axes.
Text(0.5, 0, 'x label')
>>> ax.set_ylabel('y label')  # Add a y-label to the axes.
Text(0, 0.5, 'y label')
```

Coding Styles

The explicit and the implicit interfaces

Example of Object-oriented (OO) style: cont'd.

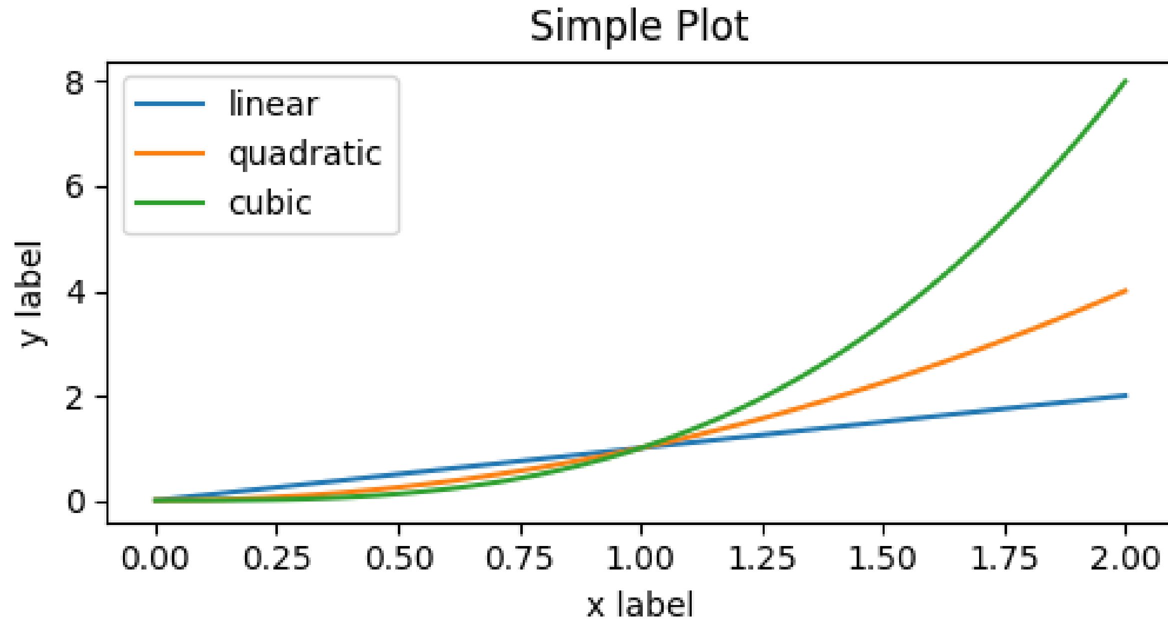
```
>>> ax.set_title("Simple Plot")  # Add a title to the axes.
Text(0.5, 1.0, 'Simple Plot')
>>> ax.legend()  # Add a legend.
<matplotlib.legend.Legend object at 0x000001417FE6FB50>
```

Coding Styles

The explicit and the implicit interfaces

Example of Object-oriented (OO) style: cont'd.

```
>>> plt.show()
```



Coding Styles

The explicit and the implicit interfaces

Example of pyplot-style:

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> x = np.linspace(0, 2, 100)  # Sample data.
>>> plt.figure(figsize=(5, 2.7), layout='constrained')
<Figure size 500x270 with 0 Axes>
>>> plt.plot(x, x, label='linear')  # Plot some data on the (implicit) axes.
[<matplotlib.lines.Line2D object at 0x00000141000BA60>]
```

Coding Styles

The explicit and the implicit interfaces

Example of pyplot-style: cont'd.

```
>>> plt.plot(x, x**2, label='quadratic')  # etc.
[<matplotlib.lines.Line2D object at 0x000001410000BD30>]
>>> plt.plot(x, x**3, label='cubic')
[<matplotlib.lines.Line2D object at 0x000001410000BFD0>]
>>> plt.xlabel('x label')
Text(0.5, 0, 'x label')
>>> plt.ylabel('y label')
Text(0, 0.5, 'y label')
```

Coding Styles

The explicit and the implicit interfaces

Example of pyplot-style: cont'd.

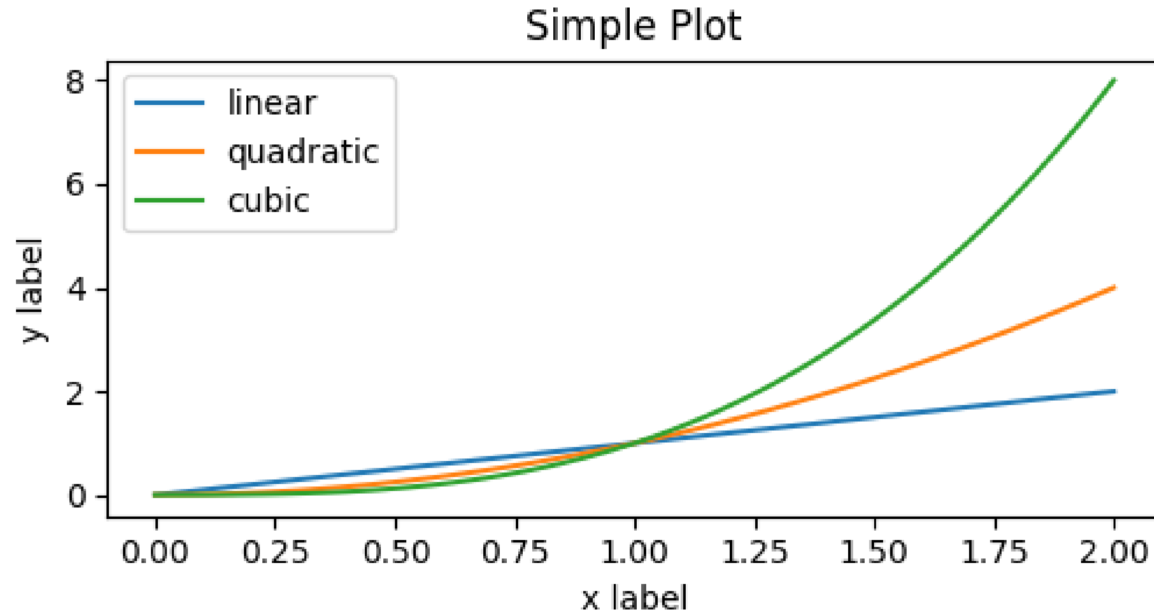
```
>>> plt.title("Simple Plot")
Text(0.5, 1.0, 'Simple Plot')
>>> plt.legend()
<matplotlib.legend.Legend object at 0x000001417FF4B190>
```

Coding Styles

The explicit and the implicit interfaces

Example of pyplot-style: cont'd.

```
>>> plt.show()
```



Coding Styles

Making a helper functions

If we need to make the same plots over and over again with different data sets, or want to easily wrap Matplotlib methods, use the recommended signature function below.

```
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>> def my_plotter(ax, data1, data2, param_dict):
    """
    A helper function to make a graph.
    """
    out = ax.plot(data1, data2, **param_dict)
    return out
```

Coding Styles

Making a helper functions

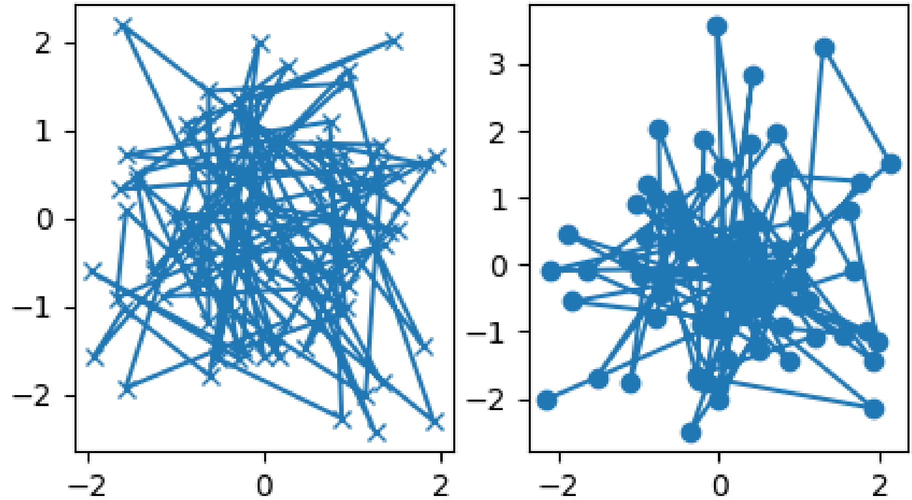
We would then use twice to populate two subplots:

```
>>> data1, data2, data3, data4 = np.random.randn(4, 100)
# make 4 random data sets
>>> fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(5,2.7))
>>> my_plotter(ax1, data1, data2, {'marker': 'x'})
[<matplotlib.lines.Line2D object at 0x000001F3D45F5B70>]
>>> my_plotter(ax2, data3, data4, {'marker': 'o'})
[<matplotlib.lines.Line2D object at 0x000001F3D45F5E70>]
```

Coding Styles

Making a helper functions

```
>>> plt.show()
```



Note that if you want to install these as a python package, or any other customizations you could use one of the many templates on the web; Matplotlib has one at `mpl-cookiecutter`

Styling Artists

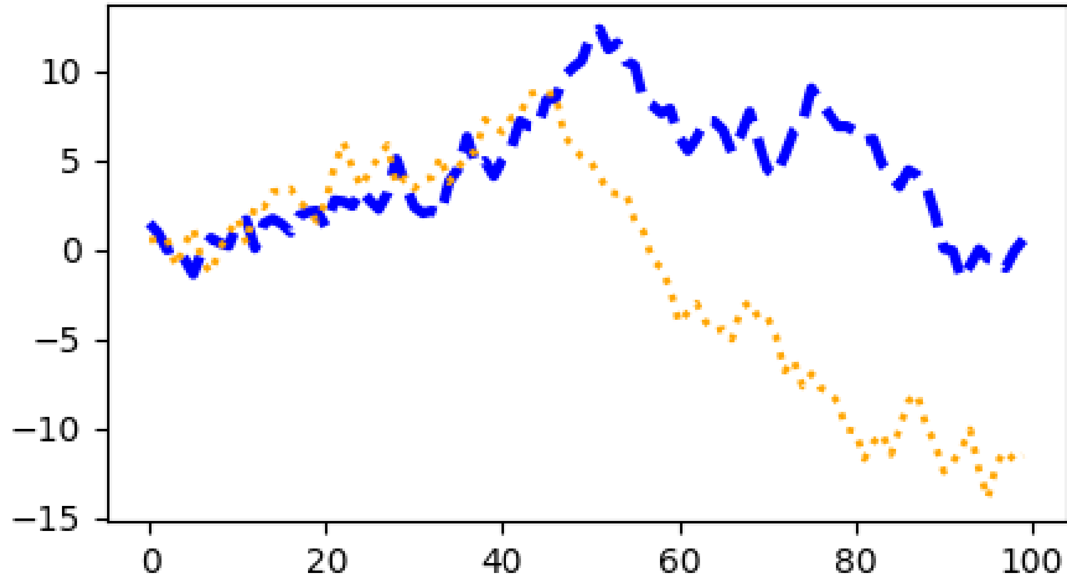
- Most plotting methods have styling options for the Artists, accessible either when a plotting method is called, or from a "setter" on the Artist.
- In the plot below we manually set the *color*, *linewidth*, and *linestyle* of the Artists created by `plot`, and we set the linestyle of the second line after the fact with `set_linestyle`.

Styling Artists

```
>>> import matplotlib.pyplot as plt
>>> data1, data2, data3, data4 = np.random.randn(4, 100)
>>> fig, ax = plt.subplots(figsize=(5, 2.7))
>>> x = np.arange(len(data1))
>>>     ax.plot(x,     np.cumsum(data1),     color='blue',
linewidth=3, linestyle='--')
[<matplotlib.lines.Line2D object at 0x000001F3D2E2F580>]
>>>  l,  =  ax.plot(x,  np.cumsum(data2),  color='orange',
linewidth=2)
>>> l.set_linestyle(':')
```

Styling Artists

```
>>> plt.show()
```



Styling Artists

Colors

- Matplotlib has a very flexible array of colors that are accepted for most Artists.
- Some Artists will take multiple colors. i.e. for a scatter plot, the edge of the markers can be different colors from the interior:

Styling Artists

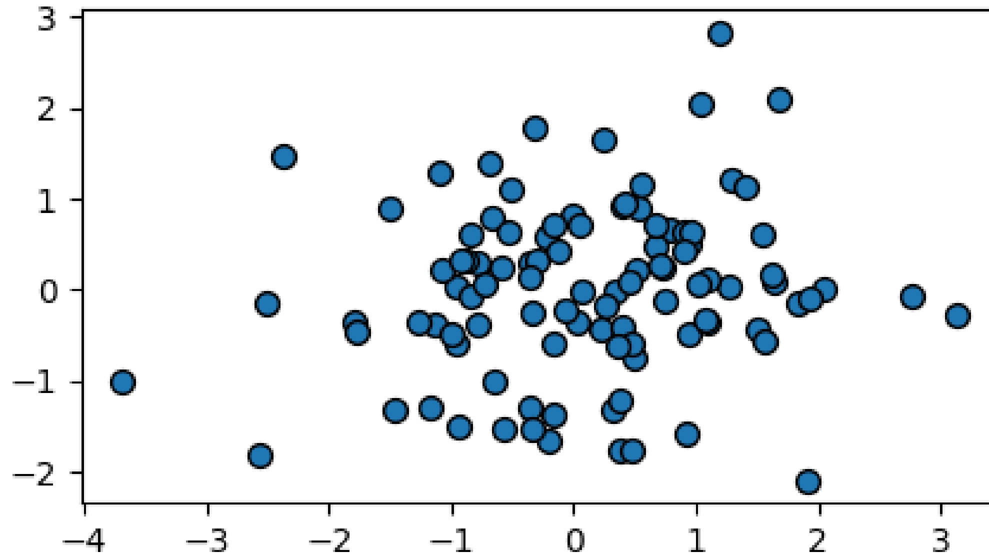
Colors

```
>>> import matplotlib.pyplot as plt
>>> data1, data2, data3, data4 = np.random.randn(4, 100)
>>> fig, ax = plt.subplots(figsize=(5, 2.7))
>>> ax.scatter(data1, data2, s=50, facecolor='C0', edgecolor='k')
<matplotlib.collections.PathCollection object at 0x000001ED8D6A4FD0>
```

Styling Artists

Colors

```
>>> plt.show()
```



Styling Artists

Linewidths, linestyles, and markersizes

- Line widths are typically in typographic points (1 pt = 1/72 inch) and available for Artists that have stroked lines.
- Similarly, stroked lines can have a linestyle.
- Marker size depends on the method being used.
- `plot` specifies markersize in points, and is generally the "diameter" or width of the marker.
- `scatter` specifies markersize as approximately proportional to the visual area of the marker.
- There is an array of markerstyles available as string codes, or users can define their own `MarkerStyle`.

Styling Artists

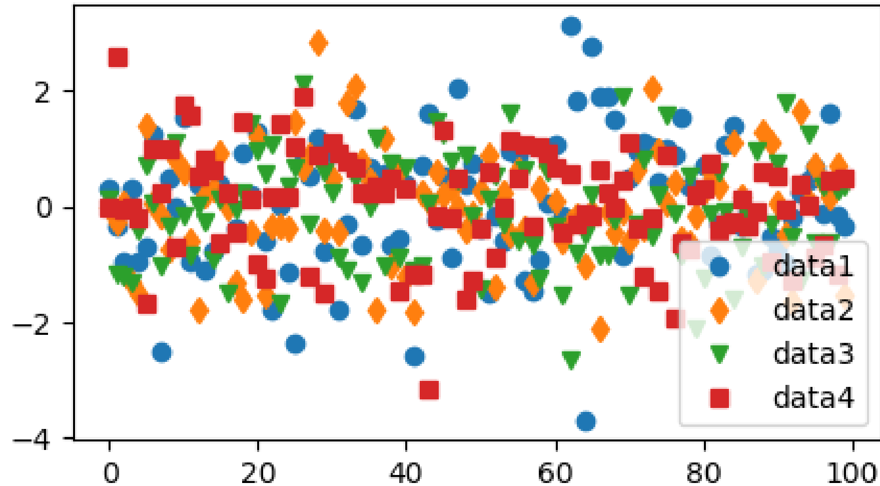
Linewidths, linestyles, and markersizes

```
>>> import matplotlib.pyplot as plt
>>> data1, data2, data3, data4 = np.random.randn(4, 100)
>>> fig, ax = plt.subplots(figsize=(5, 2.7))
>>> ax.plot(data1, 'o', label='data1')
[<matplotlib.lines.Line2D object at 0x000001ED8BEF8970>]
>>> ax.plot(data2, ‘d', label='data2')
[<matplotlib.lines.Line2D object at 0x000001ED8BEF8C40>]
>>> ax.plot(data3, ‘v', label='data3')
[<matplotlib.lines.Line2D object at 0x000001ED8BEF8DC0>]
>>> ax.plot(data4, ‘s', label='data4')
[<matplotlib.lines.Line2D object at 0x000001ED8BEF9060>]
```

Styling Artists

Linewidths, linestyles, and markersizes

```
>>> ax.legend()
<matplotlib.legend.Legend object at 0x000001ED8BEF91E0>
>>> plt.show()
```



Labelling plots

Axes labels and text

- `set_xlabel`, `set_ylabel`, and `set_title` are used to add text in the indicated locations.
- Text can also be directly added to plots using `text`:

Labelling plots

Axes labels and text

```
>>> import matplotlib.pyplot as plt
>>> mu, sigma = 115, 15
>>> x = mu + sigma * np.random.randn(10000)
>>> fig, ax = plt.subplots(figsize=(5, 2.7), layout='constrained')
>>> n, bins, patches = ax.hist(x, 50, density=True, facecolor='C0', alpha=0.75)
>>> ax.set_xlabel('Length [cm]')
Text(0.5, 0, 'Length [cm]')
>>> ax.set_ylabel('Probability')
Text(0, 0.5, 'Probability')
```

Labelling plots

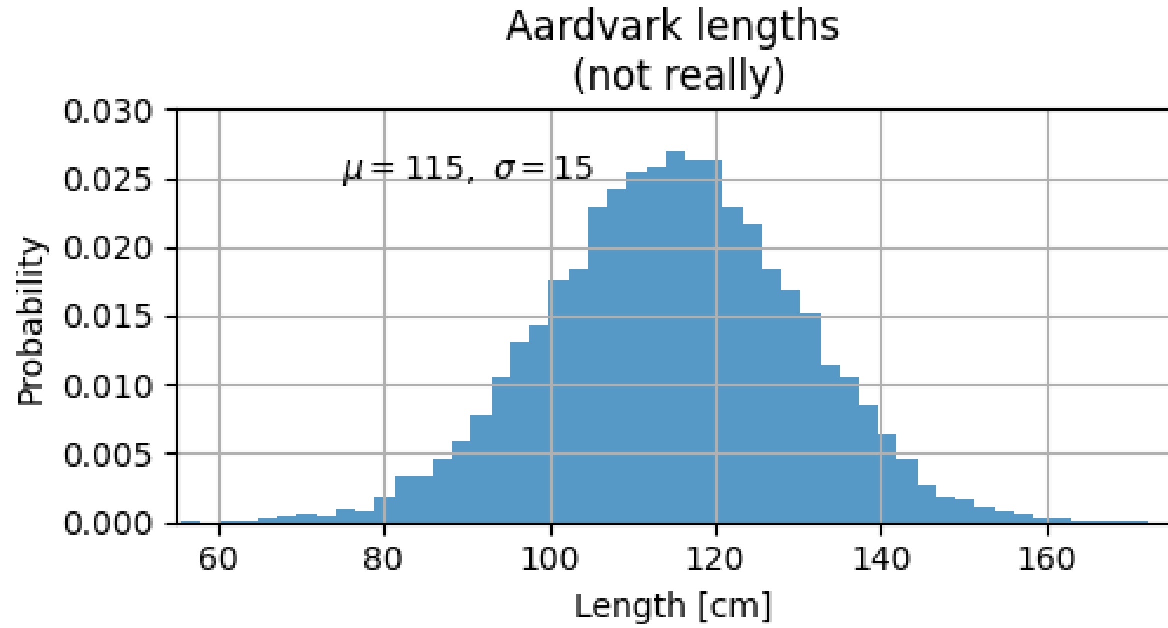
Axes labels and text

```
>>> ax.set_title('Aardvark lengths\n (not really)')
Text(0.5, 1.0, 'Aardvark lengths\n (not really)')
>>> ax.text(75, .025, r'$\mu=115,\ \sigma=15$')
Text(75, 0.025, '$\\mu=115,\\ \\sigma=15$')
>>> ax.axis([55, 175, 0, 0.03])
(55.0, 175.0, 0.0, 0.03)
>>> ax.grid(True)
```

Labelling plots

Axes labels and text

```
>>> plt.show()
```



Labelling plots

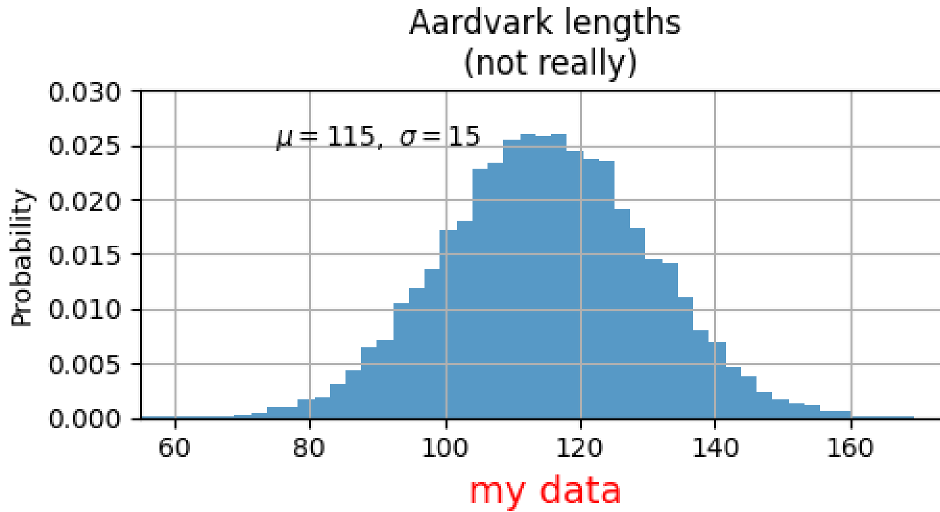
Axes labels and text

- All of the `text` functions return a `matplotlib.text.Text` instance.
- Just as with lines above, you can customize the properties by passing keyword arguments into the text functions:

Labelling plots

Axes labels and text

```
>>> ax.set_xlabel('my data', fontsize=14, color='red')
>>> plt.show()
```



Labelling plots

Annotations

- We can also annotate points on a plot, often by connecting an arrow pointing to `xy`, to a piece of text at *`xytext`*:

```
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>> fig, ax = plt.subplots(figsize=(5, 2.7))
>>> t = np.arange(0.0, 5.0, 0.01)
>>> s = np.cos(2 * np.pi * t)
>>> line, = ax.plot(t, s, lw=2)
```

Labelling plots

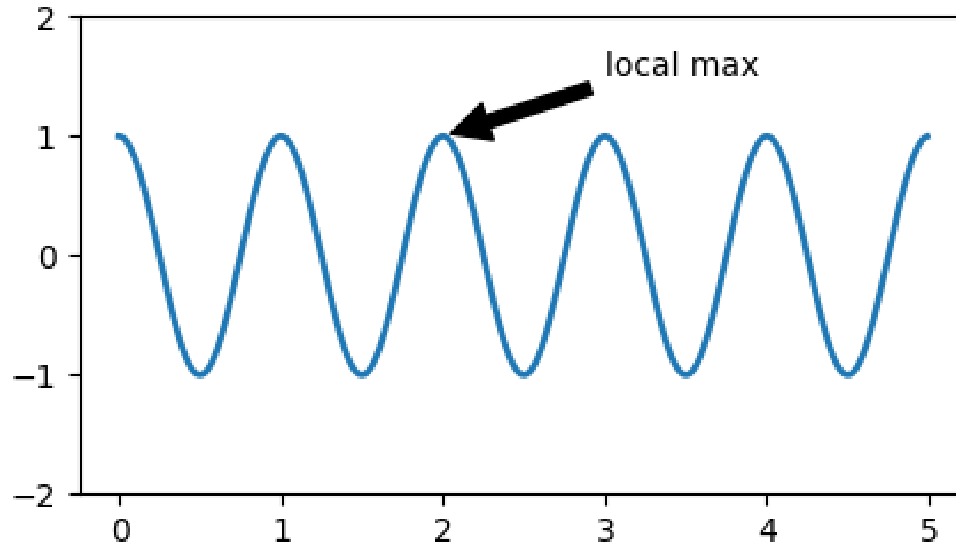
Annotations

```
>>> ax.annotate('local max', xy=(2, 1), xytext=(3, 1.5),
                arrowprops=dict(facecolor='black',
shrink=0.05))
Text(3, 1.5, 'local max')
>>> ax.set_ylim(-2, 2)
(-2.0, 2.0)
```

Labelling plots

Annotations

```
>>> plt.show()
```



Labelling plots

Legends

- Often we want to identify lines or markers with a `Axes.legend`:

```
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>> fig, ax = plt.subplots(figsize=(5, 2.7))
>>> data1, data2, data3, data4 = np.random.randn(4, 100)
>>>  ax.plot(np.arange(len(data1)), data1, label='data1')
[<matplotlib.lines.Line2D object at 0x000001ED9036CAC0>]
```

Labelling plots

Legends

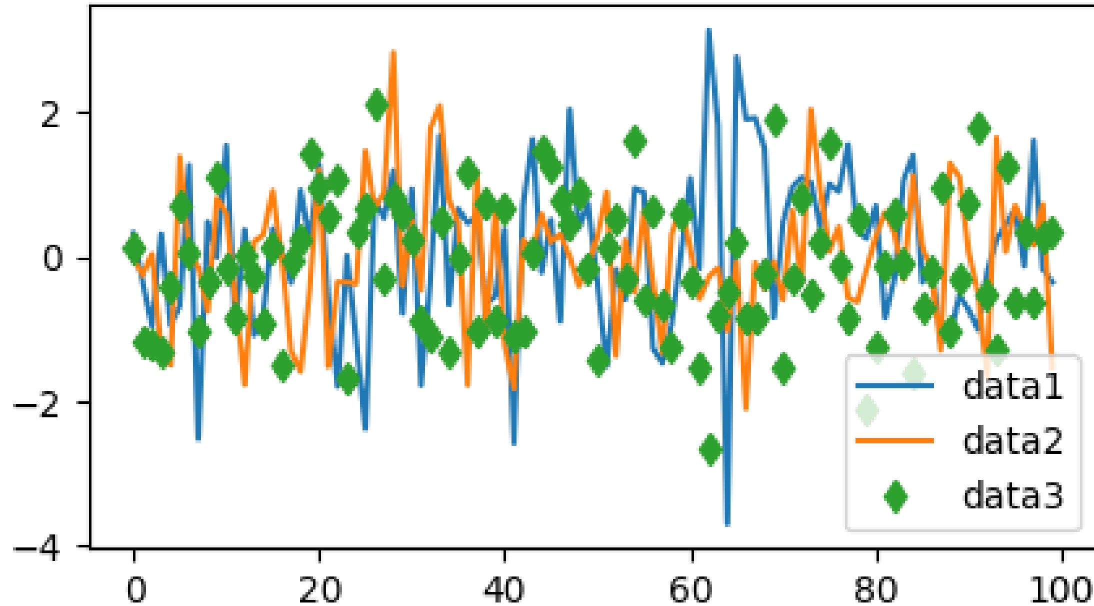
- Often we want to identify lines or markers with a `Axes.legend`:

```
>>> ax.plot(np.arange(len(data2)), data2, label='data2')
[<matplotlib.lines.Line2D object at 0x000001ED9036CD90>]
>>> ax.plot(np.arange(len(data3)), data3, 'd', label='data3')
 [<matplotlib.lines.Line2D object at 0x000001ED9036D030>]
>>> ax.legend()
<matplotlib.legend.Legend object at 0x000001ED902EC6A0>
```

Labelling plots

Legends

```
>>> plt.show()
```



Axis scales and ticks

- Each Axes has two (or three) `Axis` objects representing the x- and y-axis.
- These control the *scale* of the Axis, the tick *locators* and the tick *formatters*.
- Additional Axes can be attached to display further Axis objects.

Axis scales and ticks

Scales

- In addition to the linear scale, Matplotlib supplies non-linear scales, such as a log-scale.
- Since log-scales are used so much there are also direct methods like `loglog`, `semilogx`, and `semilogy`.
- There are a number of scales.
- Here we set the scale manually:

Axis scales and ticks

Scales

```
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>>  fig,  axs  =  plt.subplots(1,  2,  figsize=(5,  2.7), layout='constrained')
>>> data1, data2, data3, data4 = np.random.randn(4, 100)
>>> xdata = np.arange(len(data1))  # make an ordinal for this
>>> data = 10**data1
```

Axis scales and ticks

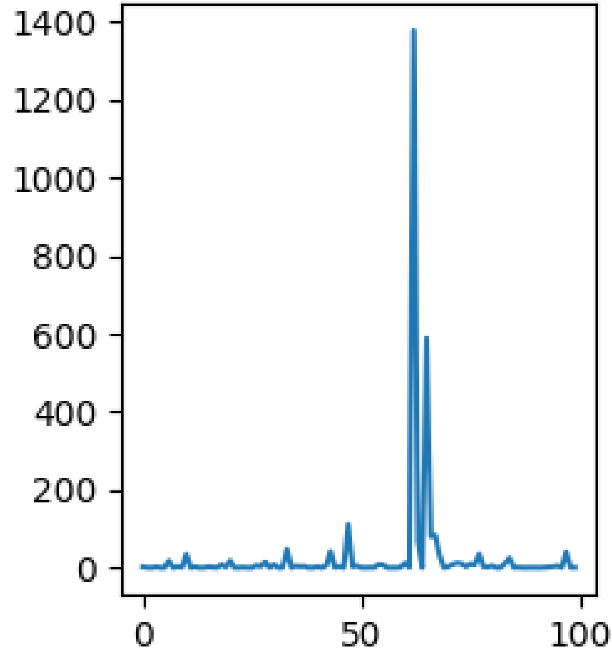
Scales

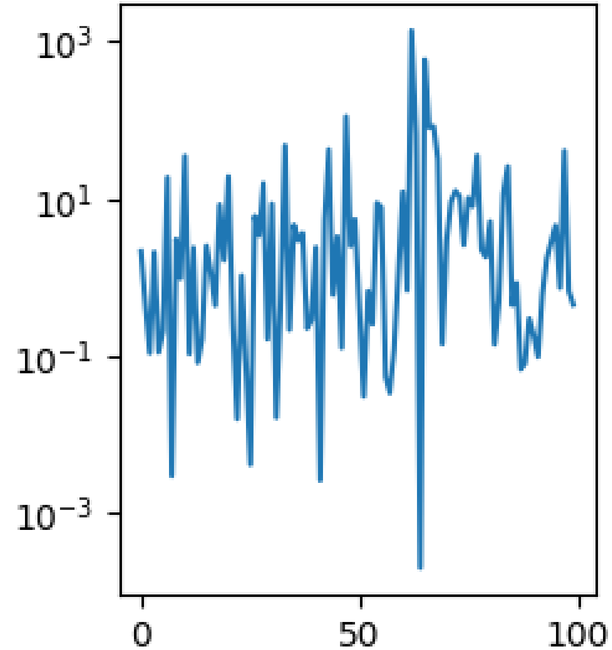
```
>>> axs[0].plot(xdata, data)
[<matplotlib.lines.Line2D object at 0x000001ED9041D1E0>]
>>> axs[1].set_yscale('log')
>>> axs[1].plot(xdata, data)
[<matplotlib.lines.Line2D object at 0x000001ED9041D4E0>]
```

Axis scales and ticks

Scales

```
>>> plt.show()
```





Axis scales and ticks

Scales

- The scale sets the mapping from data values to spacing along the Axis.
- This happens in both directions, and gets combined into a *transform*, which is the way that Matplotlib maps from data coordinates to Axes, Figure, or screen coordinates.

Axis scales and ticks

Tick locators and formatters

- Each Axis has a tick *locator* and *formatter* that choose where along the Axis objects to put tick marks.
- A simple interface to this is `set_xticks`:

Axis scales and ticks

Tick locators and formatters

```
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>> fig, axs = plt.subplots(2, 1, layout='constrained')
>>> data1, data2, data3, data4 = np.random.randn(4, 100)
>>> xdata = np.arange(len(data1))  # make an ordinal for
this
>>> axs[0].plot(xdata, data1)
[<matplotlib.lines.Line2D object at 0x000001ED902C60E0>]
```

Axis scales and ticks

Tick locators and formatters

```
>>> axs[1].plot(xdata, data1)
[<matplotlib.lines.Line2D object at 0x000001ED8C038A00>]
>>> axs[1].set_xticks(np.arange(0, 100, 30), ['zero',
'30', 'sixty', '90'])
[<matplotlib.axis.XTick object at 0x000001ED8C039FF0>,
<matplotlib.axis.XTick object at 0x000001ED8C039E40>,
<matplotlib.axis.XTick object at 0x000001ED902C4520>,
<matplotlib.axis.XTick object at 0x000001ED8BF90700>]
```

Axis scales and ticks

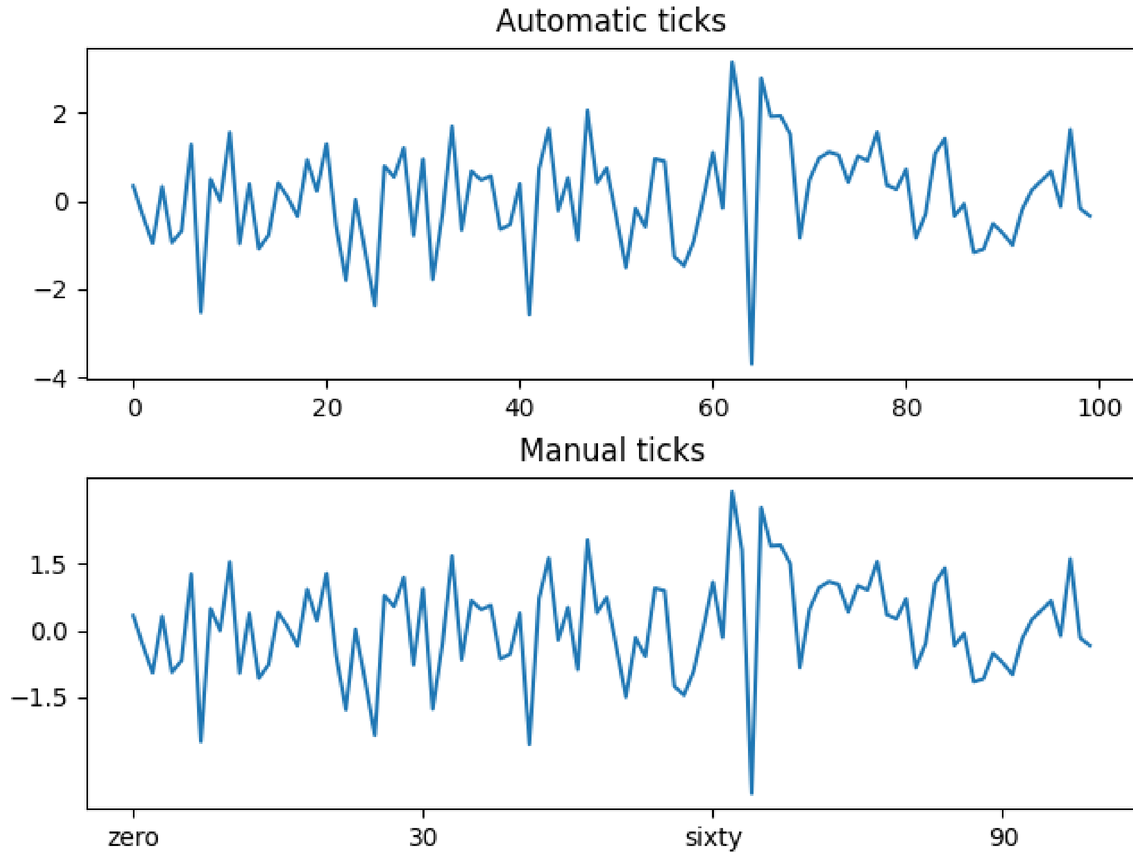
Tick locators and formatters

```
>>> axs[1].set_yticks([-1.5, 0, 1.5])  # note that we don't need to specify labels
[<matplotlib.axis.YTick object at 0x000001ED9045C640>,
<matplotlib.axis.YTick object at 0x000001ED8C038190>,
<matplotlib.axis.YTick object at 0x000001ED902C6170>]
>>> axs[1].set_title('Manual ticks')
Text(0.5, 1.0, 'Manual ticks')
```

Axis scales and ticks

Tick locators and formatters

```
>>> plt.show()
```



Axis scales and ticks

Tick locators and formatters

- Different scales can have different locators and formatters; for instance the log-scale above uses `LogLocator` and `LogFormatter`.

Axis scales and ticks

Plotting dates and strings

- Matplotlib can handle plotting arrays of dates and arrays of strings, as well as floating point numbers.
- These get special locators and formatters as appropriate.
- For dates:

Axis scales and ticks

Plotting dates and strings

```
>>> import matplotlib as mpl
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>> fig, ax = plt.subplots(figsize=(5, 2.7), layout='constrained')
>>> dates = np.arange(np.datetime64('2023-06-28'), np.datetime64('2023-07-15'),
                  np.timedelta64(1, 'h'))
```

Axis scales and ticks

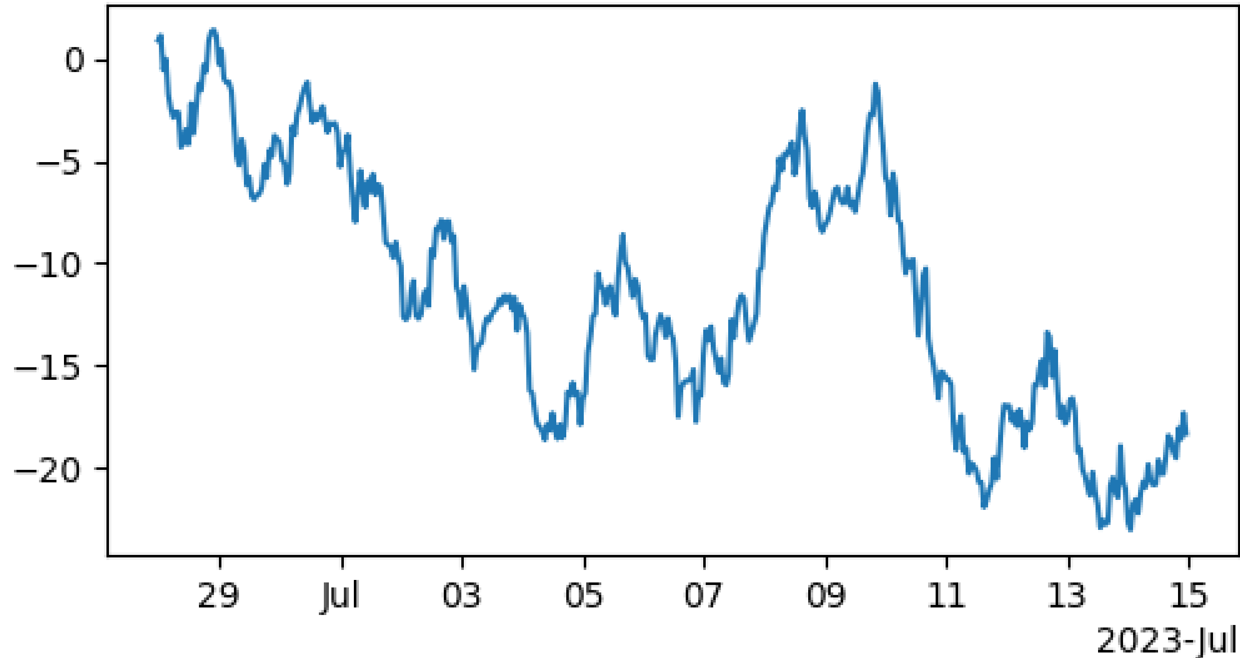
Plotting dates and strings

```
>>> data = np.cumsum(np.random.randn(len(dates)))
>>> ax.plot(dates, data)
[<matplotlib.lines.Line2D object at 0x000001ED8BF51060>]
>>> cdf =
mpl.dates.ConciseDateFormatter(ax.xaxis.get_major_locator
())
>>> ax.xaxis.set_major_formatter(cdf)
```

Axis scales and ticks

Plotting dates and strings

```
>>> plt.show()
```



Axis scales and ticks

Plotting dates and strings

- For strings: we get categorical plotting

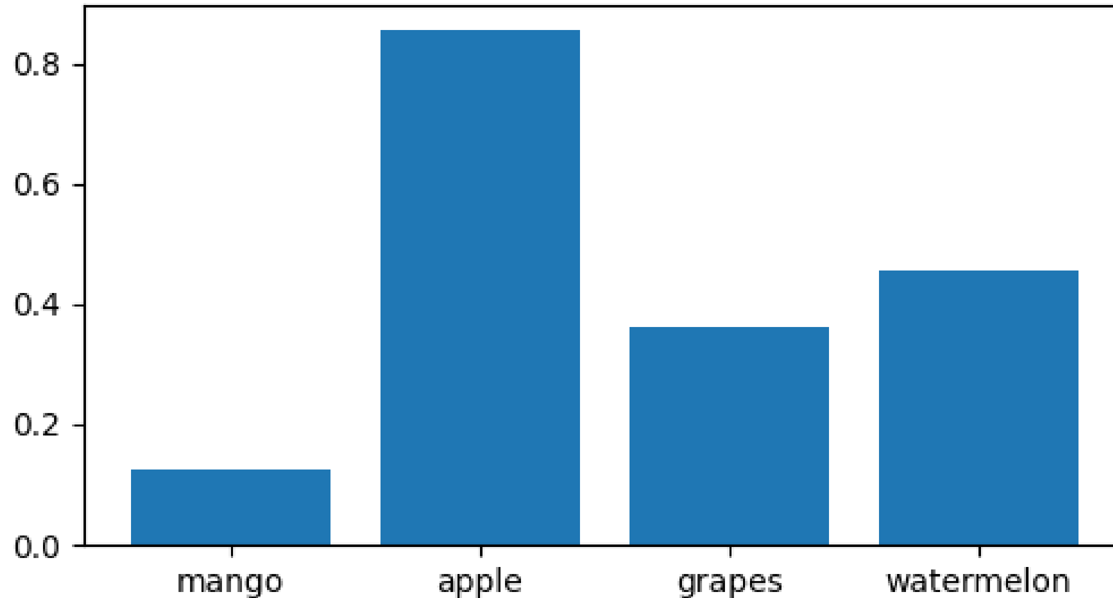
```
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>> fig, ax = plt.subplots(figsize=(5, 2.7), layout='constrained')
>>> categories = ['mango', 'apple', 'grapes', 'watermelon']
>>> ax.bar(categories, np.random.rand(len(categories)))
<BarContainer object of 4 artists>
```

Axis scales and ticks

Plotting dates and strings

- For strings: we get categorical plotting

```
>>> plt.show()
```



Axis scales and ticks

Plotting dates and strings

- One limitation about categorical plotting is that some methods of parsing text files return a list of strings, even if the strings all represent numbers or dates.
- If you pass 1000 strings, Matplotlib will think you meant 1000 categories and will add 1000 ticks to your plot.

Axis scales and ticks

Additional Axis objects

- Plotting data of different magnitude in one chart may require an additional y-axis.
- Such an Axis can be created by using `twinx` to add a new Axes with an invisible x-axis and a y-axis positioned at the right (analogously for `twiny`).
- Similarly, you can add a `secondary_xaxis` or `secondary_yaxis` having a different scale than the main Axis to represent the data in different scales or units.

Axis scales and ticks

Additional Axis objects

```
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>> fig, (ax1, ax3) = plt.subplots(1, 2, figsize=(7, 2.7), layout='constrained')
>>> l1, = ax1.plot(t, s)
>>> ax2 = ax1.twinx()
>>> l2, = ax2.plot(t, range(len(t)), 'C1')
>>> ax2.legend([l1, l2], ['Sine (left)', 'Straight (right)'])
<matplotlib.legend.Legend object at 0x000001ED905501C0>
```

Axis scales and ticks

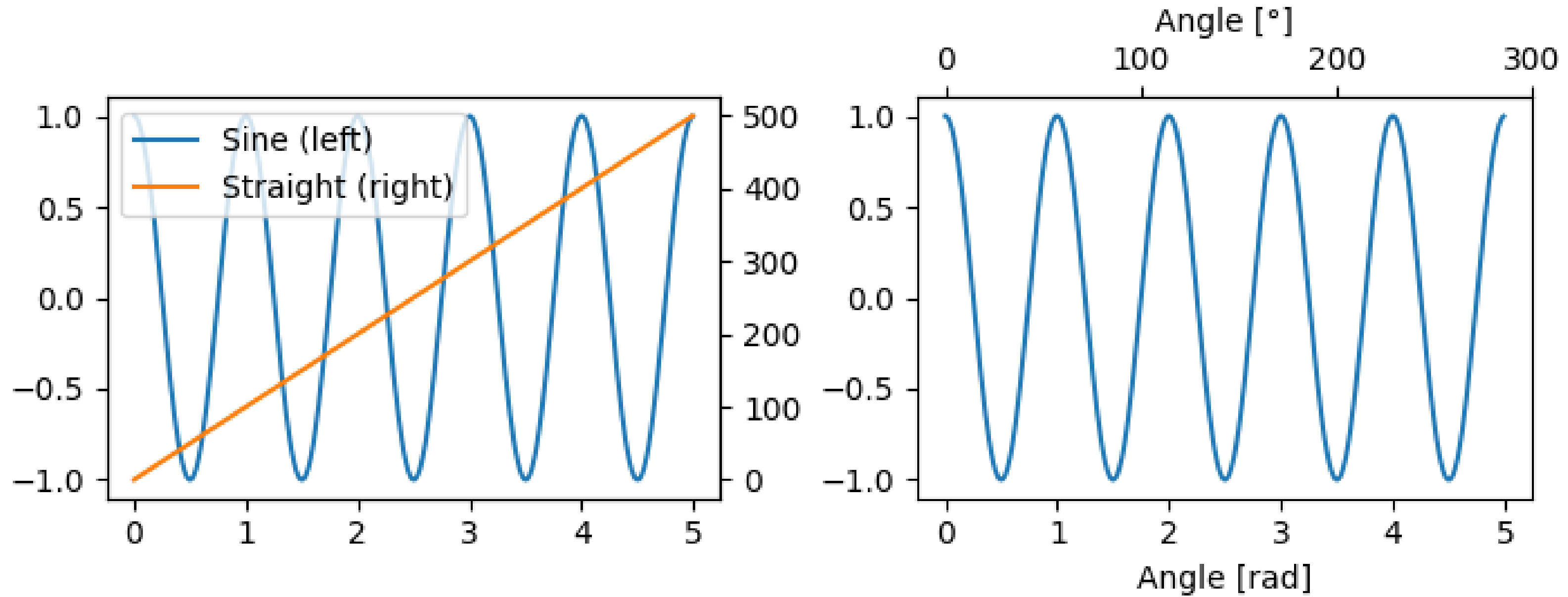
Additional Axis objects

```
>>> ax3.plot(t, s)
[<matplotlib.lines.Line2D object at 0x000001ED90581C90>]
>>> ax3.set_xlabel('Angle [rad]')
Text(0.5, 0, 'Angle [rad]')
>>> ax4 = ax3.secondary_xaxis('top', functions=(np.rad2deg, np.deg2rad))
>>> ax4.set_xlabel('Angle [°]')
Text(0.5, 0, 'Angle [°]')
```

Axis scales and ticks

Additional Axis objects

```
>>> plt.show()
```



Color mapped data

- Often we want to have a third dimension in a plot represented by a colors in a colormap.
- Matplotlib has a number of plot types that do this:

Color mapped data

```
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>> X, Y = np.meshgrid(np.linspace(-3, 3, 128), np.linspace(-3, 3, 128))
>>> Z = (1 - X/2 + X**5 + Y**3) * np.exp(-X**2 - Y**2)
>>> fig, axs = plt.subplots(2, 2, layout='constrained')
>>> pc = axs[0, 0].pcolormesh(X, Y, Z, vmin=-1, vmax=1, cmap='RdBu_r')
>>> fig.colorbar(pc, ax=axs[0, 0])
<matplotlib.colorbar.Colorbar object at 0x000001ED9372DDB0>
```

Color mapped data

```
>>> axs[0, 0].set_title('pcolormesh()')
Text(0.5, 1.0, 'pcolormesh()')
>>> co = axs[0, 1].contourf(X, Y, Z, levels=np.linspace(-1.25, 1.25, 11))
>>> fig.colorbar(co, ax=axs[0, 1])
<matplotlib.colorbar.Colorbar object at 0x000001ED9001D3C0>
>>> axs[0, 1].set_title('contourf()')
Text(0.5, 1.0, 'contourf()')
```

Color mapped data

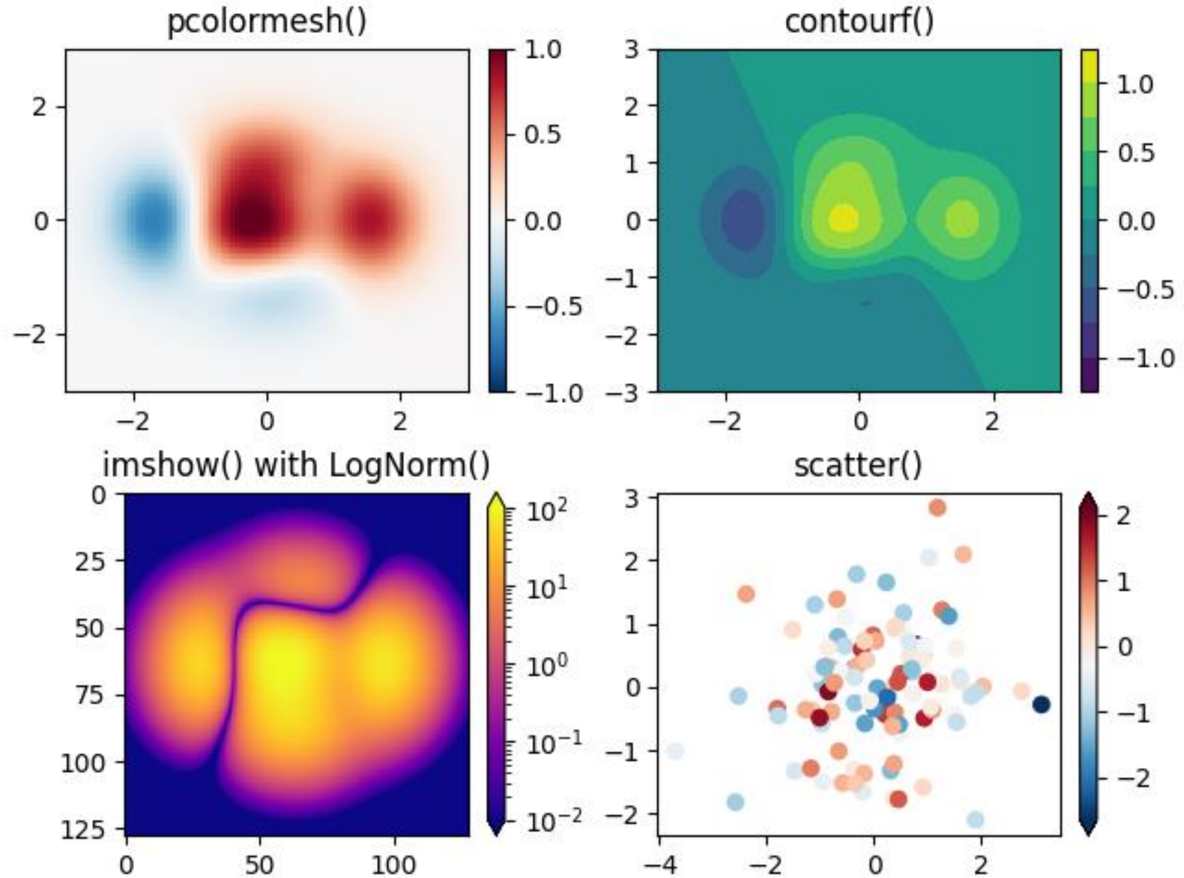
```
>>> pc = axs[1, 0].imshow(Z**2 * 100, cmap='plasma',

norm=mpl.colors.LogNorm(vmin=0.01, vmax=100))
>>> fig.colorbar(pc, ax=axs[1, 0], extend='both')
<matplotlib.colorbar.Colorbar object at
0x000001ED93696500>
>>> axs[1, 0].set_title('imshow() with LogNorm()')
Text(0.5, 1.0, 'imshow() with LogNorm()')
```

Color mapped data

```
>>> pc = axs[1, 1].scatter(data1, data2, c=data3,
cmap='RdBu_r')
>>> fig.colorbar(pc, ax=axs[1, 1], extend='both')
<matplotlib.colorbar.Colorbar object at
0x000001ED936C2530>
>>> axs[1, 1].set_title('scatter()')
Text(0.5, 1.0, 'scatter()')
```

Color mapped data

```
>>> plt.show()
```



Color mapped data

Colormaps

- These are all examples of Artists that derive from `ScalarMappable` objects.
- They all can set a linear mapping between *vmin* and *vmax* into the colormap specified by *cmap*.
- Matplotlib has many colormaps to choose from the existing, you can make your own or download as third-party packages.

Color mapped data

Normalizations

- Sometimes we want a non-linear mapping of the data to the colormap, as in the `LogNorm` example above.
- We do this by supplying the ScalarMappable with the *norm* argument instead of *vmin* and *vmax*.

Color mapped data

Colorbars

- Adding a `colorbar` gives a key to relate the color back to the underlying data.
- Colorbars are figure-level Artists, and are attached to a ScalarMappable (where they get their information about the norm and colormap) and usually steal space from a parent Axes.
- Placement of colorbars can be complex.
- You can also change the appearance of colorbars with the extend keyword to add arrows to the ends, and shrink and aspect to control the size.
- Finally, the colorbar will have default locators and formatters appropriate to the norm.
- These can be changed as for other Axis objects.