Visualization

IOE 373 Lecture 21

Topics

- Intro to Matplotlib
- Matplotlib Commands
- Special Plot Types

Intro to Matplotlib

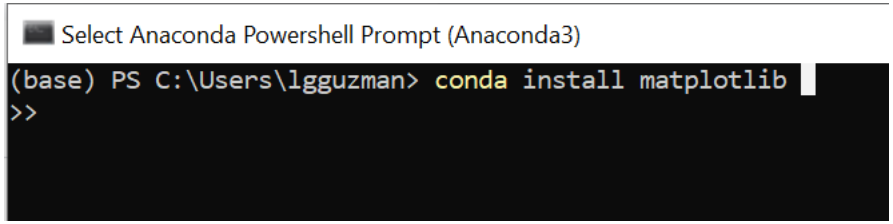
- Matplotlib is the "grandfather" library of data visualization with Python.
- Created by John Hunter. He created it to try to replicate MatLab's plotting capabilities in Python.
- If you happen to be familiar with matlab, matplotlib will feel natural to you

Matplotlib

- It is an excellent 2D and 3D graphics library for generating scientific figures.
- Major Pros of Matplotlib are:
  - Generally easy to get started for simple plots
  - Support for custom labels and texts
  - Great control of every element in a figure
  - High-quality output in many formats
  - Very customizable in general
- Matplotlib allows you to create reproducible figures programmatically,
- Explore the official Matplotlib web page for code and other ideas: http://matplotlib.org/

Installation

- As with Pandas, if you are working off of your personal computer and haven't used Matplotlib before, make sure you install it first:
  - conda install matplotlib



  - As with pandas, once you install and before you start running scripts, you'll need to import a library/module. Let's use the matplotlib.pyplot module:

```
In [1]: import matplotlib.pyplot as plt
```

Matplotlib basic commands

- Let's walk through a very simple example using two numpy arrays.
  - most likely you'll be passing numpy arrays or pandas columns (which essentially also behave like arrays) for most plots.

```
In [3]: import numpy as np
   ...: x = np.linspace(0, 5, 11)
   ...: y = x ** 2
```

Returns evenly spaced samples, calculated over the interval [start, stop]. (similar to np.arrange, but you specify the number of samples): numpy.linspace(start, stop, num)
In this example, generates a sequence of 11 values evenly spaced between 0 and 5

Basic Commands

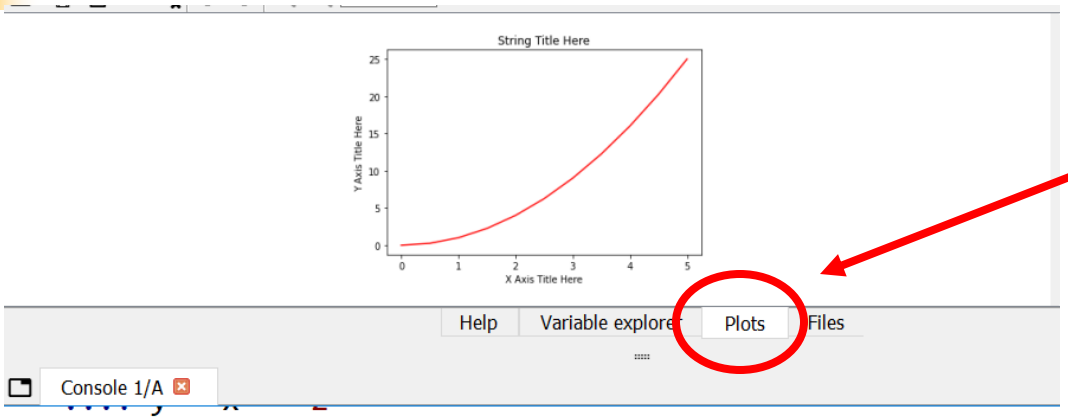
- We can create a very simple line plot using the following plt.plot:

```
In [4]: x
Out[4]: array([0. , 0.5, 1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5, 5. ])

In [5]: y
Out[5]:
array([ 0.  ,  0.25,  1.  ,  2.25,  4.  ,  6.25,  9.  , 12.25, 16.  ,
       20.25, 25.  ])

In [6]: plt.plot(x, y, 'r') # 'r' is the color red
   ...: plt.xlabel('X Axis Title Here')
   ...: plt.ylabel('Y Axis Title Here')
   ...: plt.title('String Title Here')
   ...: plt.show()
```

Basic Commands




```
In [4]: x
Out[4]: array([0. , 0.5, 1. , 1.5, 2. , 2.5, 3. , 3.5, 4. ,

In [5]: y
Out[5]:
array([ 0.  ,  0.25,  1.  ,  2.25,  4.  ,  6.25,  9.  , 12.
       20.25, 25.  ])

In [6]: plt.plot(x, y, 'r') # 'r' is the color red
   ...: plt.xlabel('X Axis Title Here')
   ...: plt.ylabel('Y Axis Title Here')
   ...: plt.title('String Title Here')
   ...: plt.show()
```

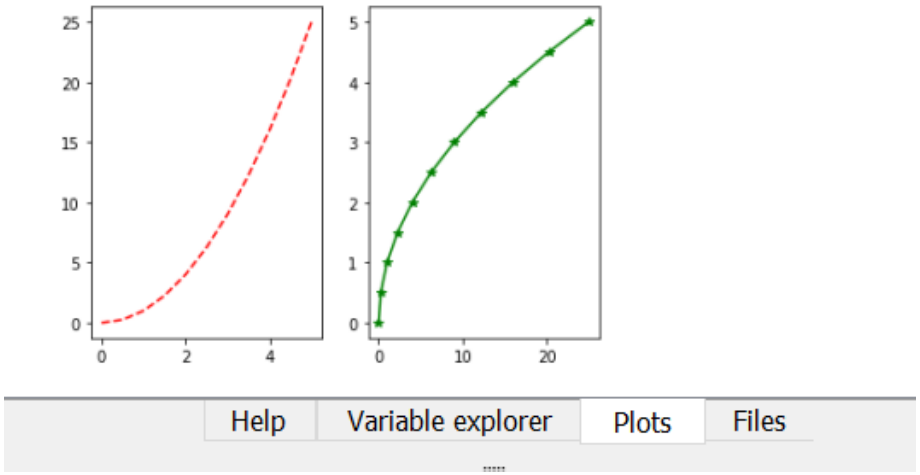
The plot will be shown in the Plots Pane on Spyder…

- you can show the plots in the console if you're using Jupyter Notebooks using the command: %matplotlib inline

Multiplots

- Creating Multiplots on Same Canvas:

```
In [9]: # plt.subplot(nrows, ncols, plot_number)
   ...: plt.subplot(1,2,1)
   ...: plt.plot(x, y, 'r--')
   ...: plt.subplot(1,2,2)
   ...: plt.plot(y, x, 'g*-');
```



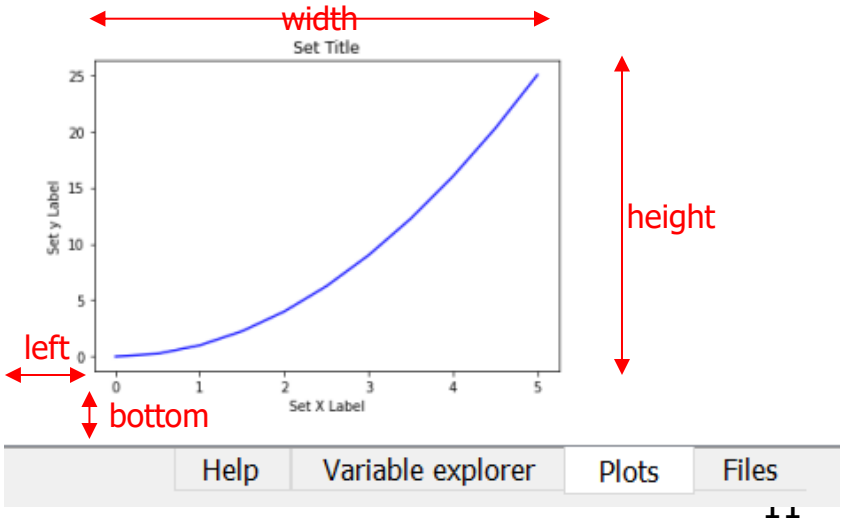
Matplotlib - Object Oriented Method

- Now that we've seen the basics, let's break it all down with a more formal introduction of Matplotlib's Object Oriented Method
- The main idea in using the more formal Object Oriented method is to create figure objects and then just call methods or attributes off of that object (similar to what we learned with VBA).
- This approach is nicer when dealing with a canvas that has multiple plots on it.

Object Oriented Method

- Let's start with a sample plot:

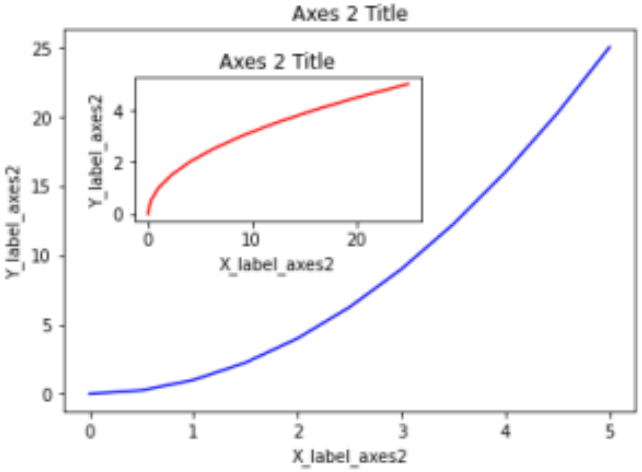
```
In [14]: # Create Figure (empty canvas)
    ...: fig = plt.figure()
    ...:
    ...: # Add set of axes to figure
    ...: axes = fig.add_axes([0.1, 0.1, 0.8, 0.8]) # left, bottom,
width, height (range 0 to 1)
    ...:
    ...: # Plot on that set of axes
    ...: axes.plot(x, y, 'b')
    ...: axes.set_xlabel('Set X Label') # Notice the use of set_ to
begin methods
    ...: axes.set_ylabel('Set y Label')
    ...: axes.set_title('Set Title')
Out[14]: Text(0.5, 1.0, 'Set Title')
```



Object Oriented Method

- Code is a little more complicated, but now we have full control of where the plot axes are placed
  - easily add more than one axis to the figure:

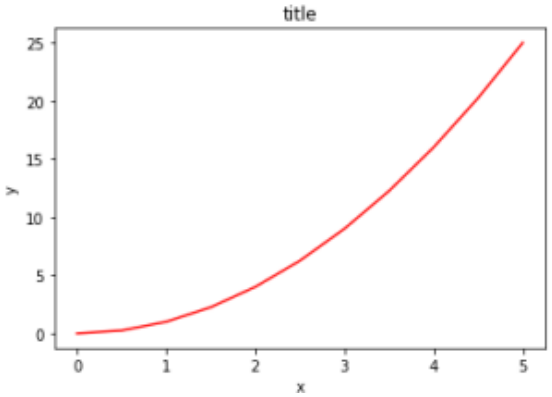
```
In [15]: # Creates blank canvas
    ...: fig = plt.figure()
    ...:
    ...: axes1 = fig.add_axes([0.1, 0.1, 0.8, 0.8]) # main axes
    ...: axes2 = fig.add_axes([0.2, 0.5, 0.4, 0.3]) # inset axes
    ...:
    ...: # Larger Figure Axes 1
    ...: axes1.plot(x, y, 'b')
    ...: axes1.set_xlabel('X_label_axes2')
    ...: axes1.set_ylabel('Y_label_axes2')
    ...: axes1.set_title('Axes 2 Title')
    ...:
    ...: # Insert Figure Axes 2
    ...: axes2.plot(y, x, 'r')
    ...: axes2.set_xlabel('X_label_axes2')
    ...: axes2.set_ylabel('Y_label_axes2')
    ...: axes2.set_title('Axes 2 Title');
```



Subplots

- The plt.subplots() object will act as a more automatic axis manager.
- Basic use cases:

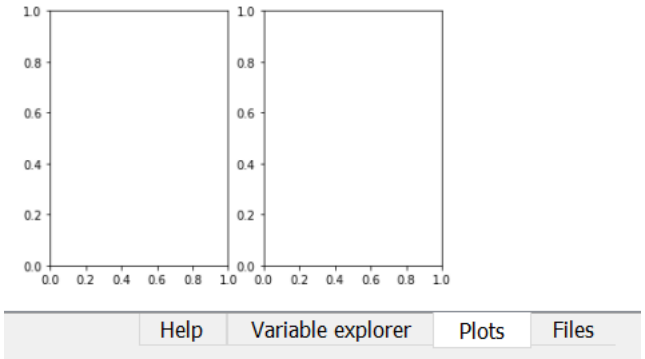
```
In [16]: # Use similar to plt.figure() except use tuple unpacking to
grab fig and axes
    ...: fig, axes = plt.subplots()
    ...:
    ...: # Now use the axes object to add stuff to plot
    ...: axes.plot(x, y, 'r')
    ...: axes.set_xlabel('x')
    ...: axes.set_ylabel('y')
    ...: axes.set_title('title');
```



Subplots

- Then, we can add the number of rows and columns when creating the subplots() object:

```
In [17]: # Empty canvas of 1 by 2 subplots
    ...: fig, axes = plt.subplots(nrows=1, ncols=2)
```



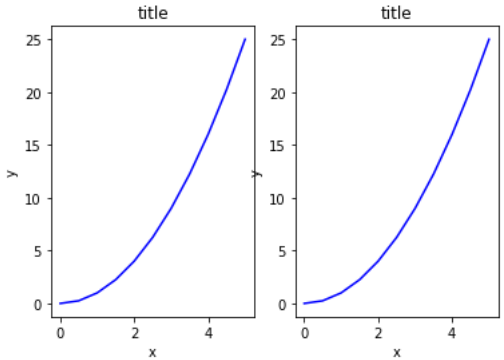

```
In [18]: # Axes is an array of axes to plot on
    ...: axes
Out[18]: array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)
```

Subplots

- We can iterate through this array:

```
In [19]: for ax in axes:
    ...:     ax.plot(x, y, 'b')
    ...:     ax.set_xlabel('x')
    ...:     ax.set_ylabel('y')
    ...:     ax.set_title('title')
    ...:
    ...: # Display the figure object
    ...: fig
```

Out[19]:



Subplots

- A common issue with matplolib is overlapping subplots or figures.
  - use **fig.tight_layout()** or **plt.tight_layout()** to adjust the positions of the axes on the figure canvas so that there is no overlapping content:

```
In [20]: fig, axes = plt.subplots(nrows=1, ncols=2)
    ...:
    ...: for ax in axes:
    ...:     ax.plot(x, y, 'g')
    ...:     ax.set_xlabel('x')
    ...:     ax.set_ylabel('y')
    ...:     ax.set_title('title')
    ...:
    ...: fig
    ...: plt.tight_layout()
```

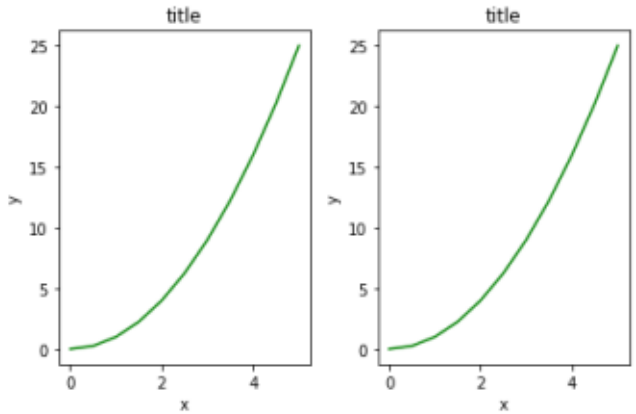


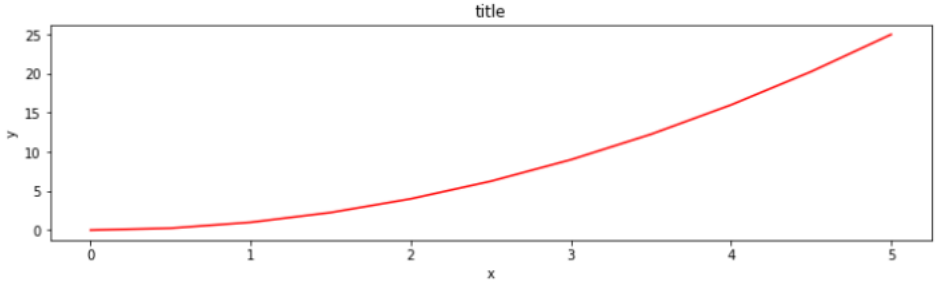
Figure size, aspect ratio and DPI

- Matplotlib allows the aspect ratio, DPI and figure size to be specified when the Figure object is created.
  - use the figsize and dpi keyword arguments:
    - figsize is a tuple of the width and height of the figure in inches
    - dpi is the dots-per-inch (pixel per inch).

```
In [21]: fig = plt.figure(figsize=(8,4), dpi=100)
<Figure size 800x400 with 0 Axes>
```

  - Same arguments can be passed to the subplots function:

```
In [22]: fig, axes = plt.subplots(figsize=(12,3))
    ...:
    ...: axes.plot(x, y, 'r')
    ...: axes.set_xlabel('x')
    ...: axes.set_ylabel('y')
    ...: axes.set_title('title');
```



Saving Figures

- To save a figure to a file we can use the savefig method in the Figure class:

```
In [23]: fig.savefig("filename.png")
```

- Optionally specify the DPI and choose between different output formats:

```
In [24]: fig.savefig("filename.png", dpi=200)
```

Legends, labels and titles

- Figure titles can be added to each axis instance in a figure. To set the title, use the set_title method in the axes instance:

```
In [25]: ax.set_title("title");
```

- Axis labels can be added with set_xlabel and set_ylabel:

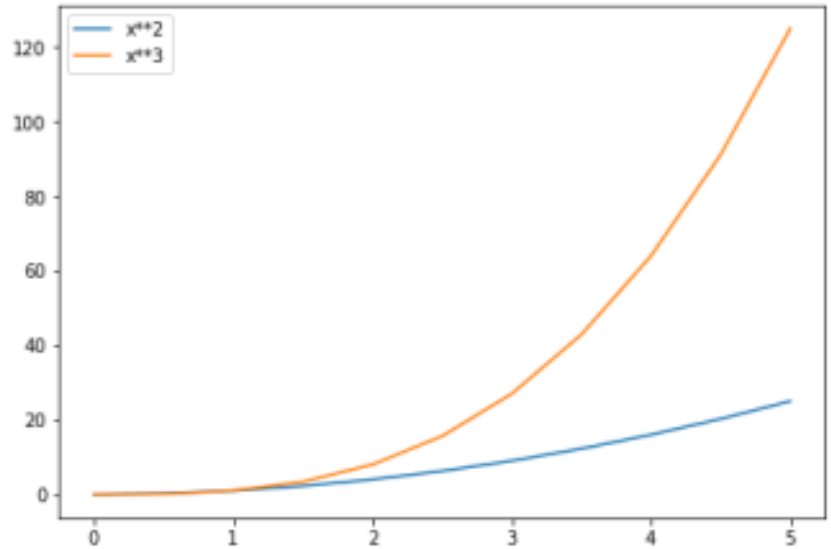
```
In [26]: ax.set_xlabel("x")
    ...: ax.set_ylabel("y");
```

- For legends use the label="label text" keyword argument when plots or other objects are added to the figure, and then using the legend method without arguments to add the legend to the figure:

Legends

```
In [26]: ax.set_xlabel("x")
    ...: ax.set_ylabel("y");

In [27]: fig = plt.figure()
    ...:
    ...: ax = fig.add_axes([0,0,1,1])
    ...:
    ...: ax.plot(x, x**2, label="x**2")
    ...: ax.plot(x, x**3, label="x**3")
    ...: ax.legend()
Out[27]: <matplotlib.legend.Legend at 0x25bf27cf430>
```

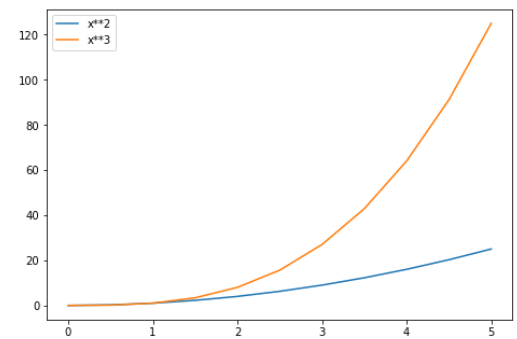


Legends

- The legend function takes an optional keyword argument loc that can be used to specify where in the figure the legend is to be drawn. The allowed values of loc are numerical codes for the various places the legend can be drawn. Some of the most common loc values are:

```
In [28]: # Lots of options....
    ...:
    ...: ax.legend(loc=1) # upper right corner
    ...: ax.legend(loc=2) # upper left corner
    ...: ax.legend(loc=3) # lower left corner
    ...: ax.legend(loc=4) # lower right corner
    ...:
    ...: # .. many more options are available
    ...:
    ...: # Most common to choose
    ...: ax.legend(loc=0) # let matplotlib decide the optimal location
    ...: fig
```

Out[28]:

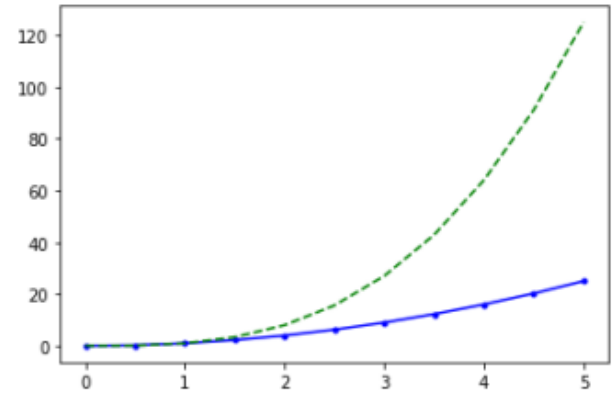


https://matplotlib.org/tutorials/intermediate/legend_guide.html

Setting colors, linewidths, linetypes

- Define the colors of lines and other graphical elements in a number of ways.
  - use the MATLAB-like syntax where 'b' means blue, 'g' means green, etc.
  - for selecting line styles use symbols after the color, for example, 'b.-' means a blue line with dots

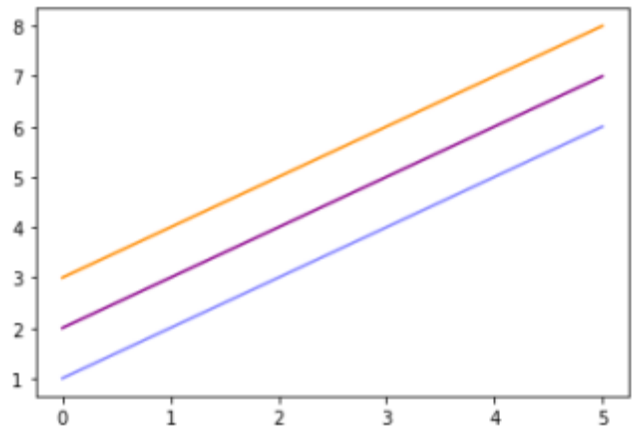
```
In [32]: # MATLAB style line color and style
    ...: fig, ax = plt.subplots()
    ...: ax.plot(x, x**2, 'b.-') # blue line with dots
    ...: ax.plot(x, x**3, 'g--') # green dashed line
Out[32]: [<matplotlib.lines.Line2D at 0x25bf4ab2070>]
```



Colors

- We can also define colors by their names or RGB hex codes and optionally provide an alpha value using the color and alpha keyword arguments. Alpha indicates opacity.

```
In [33]: fig, ax = plt.subplots()
    ...:
    ...: ax.plot(x, x+1, color="blue", alpha=0.5) # half-transparant
    ...: ax.plot(x, x+2, color="#8B008B")        # RGB hex code
    ...: ax.plot(x, x+3, color="#FF8C00")        # RGB hex code
Out[33]: [<matplotlib.lines.Line2D at 0x25bf4b0d8e0>]
```



Lines and markers

- To change the line width, use linewidth or lw keyword argument.

```
In [34]: fig, ax = plt.subplots(figsize=(12,6))
    ...:
    ...: ax.plot(x, x+1, color="red", linewidth=0.25)
    ...: ax.plot(x, x+2, color="red", linewidth=0.50)
    ...: ax.plot(x, x+3, color="red", linewidth=1.00)
    ...: ax.plot(x, x+4, color="red", linewidth=2.00)
Out[34]: [<matplotlib.lines.Line2D at 0x25bf4b6fbe0>]
```

- Line style can be selected using the linestyle or ls keyword arguments

```
In [35]: # possible linestype options '-', '–', '-.', ':', 'steps'
    ...: ax.plot(x, x+5, color="green", lw=3, linestyle='-')
    ...: ax.plot(x, x+6, color="green", lw=3, ls='-.')
    ...: ax.plot(x, x+7, color="green", lw=3, ls=':')
Out[35]: [<matplotlib.lines.Line2D at 0x25bf4d23070>]
```

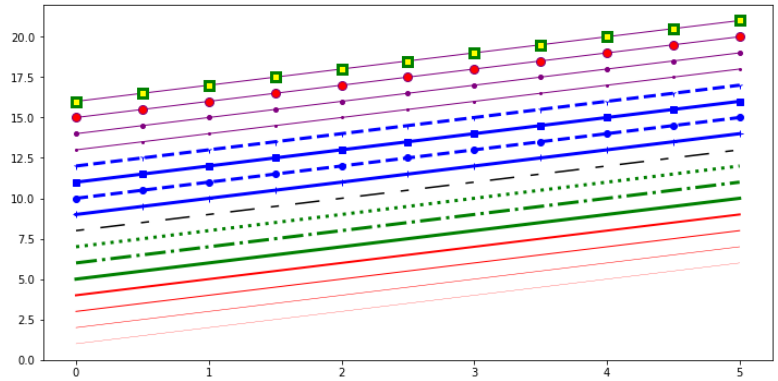
Lines and markers

```
# custom dash
line, = ax.plot(x, x+8, color="black", lw=1.50)
line.set_dashes([5, 10, 15, 10]) # format: line length, space length, ...

# possible marker symbols: marker = '+', 'o', '*', 's', ',', '.', '1', '2', '3', '4', ...
ax.plot(x, x+ 9, color="blue", lw=3, ls='-', marker='+')
ax.plot(x, x+10, color="blue", lw=3, ls='--', marker='o')
ax.plot(x, x+11, color="blue", lw=3, ls='-', marker='s')
ax.plot(x, x+12, color="blue", lw=3, ls='--', marker='1')

# marker size and color
ax.plot(x, x+13, color="purple", lw=1, ls='-', marker='o', markersize=2)
ax.plot(x, x+14, color="purple", lw=1, ls='-', marker='o', markersize=4)
ax.plot(x, x+15, color="purple", lw=1, ls='-', marker='o', markersize=8, markerfacecolor="red")
ax.plot(x, x+16, color="purple", lw=1, ls='-', marker='s', markersize=8,
        markerfacecolor="yellow", markeredgewidth=3, markeredgecolor="green");
```

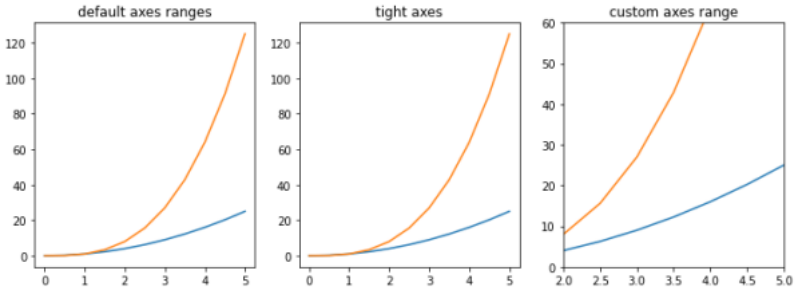
```
In [37]: fig
Out[37]:
```



Plot Ranges

- Configure the ranges of the axes using the set_ylim and set_xlim methods in the axis object,
  - or axis('tight') for automatically getting "tightly fitted" axes ranges:

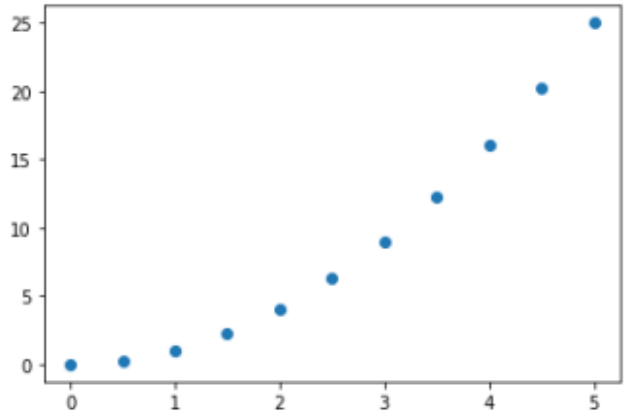
```
In [38]: fig, axes = plt.subplots(1, 3, figsize=(12, 4))
    ...:
    ...: axes[0].plot(x, x**2, x, x**3)
    ...: axes[0].set_title("default axes ranges")
    ...:
    ...: axes[1].plot(x, x**2, x, x**3)
    ...: axes[1].axis('tight')
    ...: axes[1].set_title("tight axes")
    ...:
    ...: axes[2].plot(x, x**2, x, x**3)
    ...: axes[2].set_ylim([0, 60])
    ...: axes[2].set_xlim([2, 5])
    ...: axes[2].set_title("custom axes range");
```



Special Plot Types

- There are many specialized plots we can create, such as barplots, histograms, scatter plots:
  - Scatter Plot

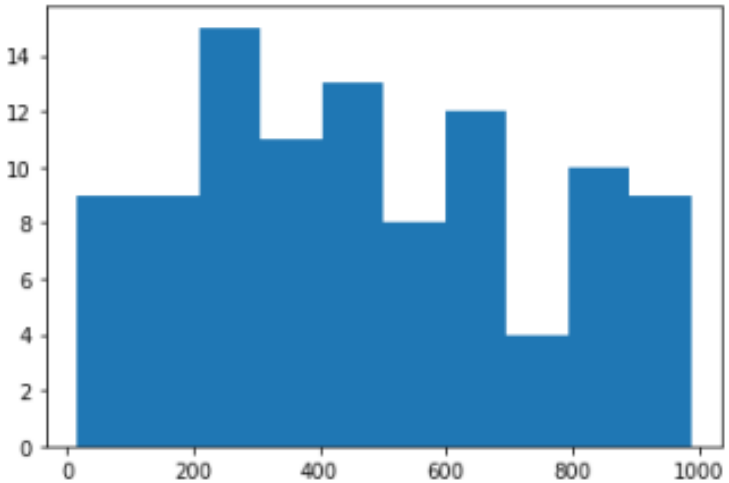
```
In [29]: plt.scatter(x,y)
Out[29]: <matplotlib.collections.PathCollection at 0x25bf412ad90>
```



Special Plot Types

- Histogram:

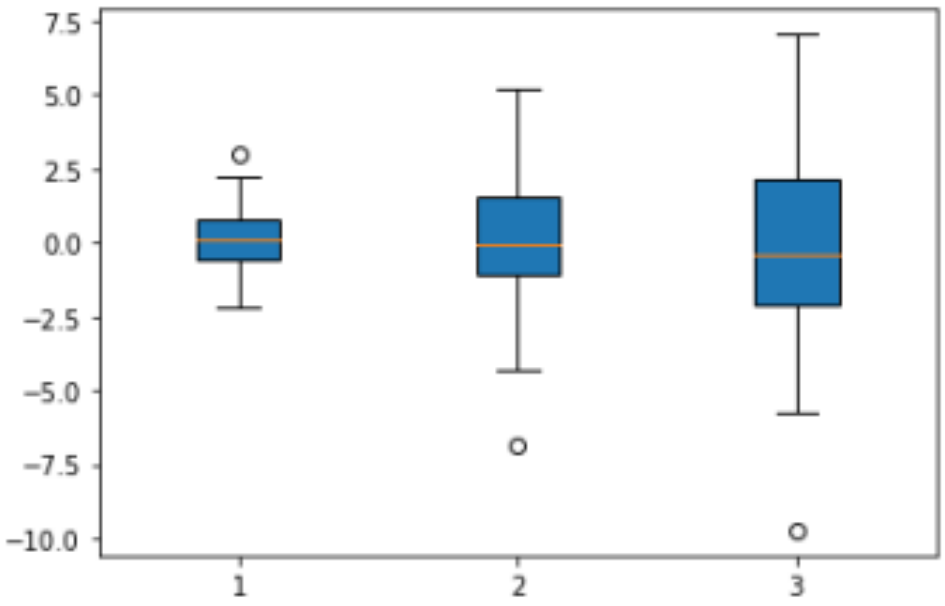
```
In [30]: from random import sample
    ...: data = sample(range(1, 1000), 100)
    ...: plt.hist(data)
Out[30]:
(array([ 9.,  9., 15., 11., 13.,  8., 12.,  4., 10.,  9.]),
 array([ 16. , 113.1, 210.2, 307.3, 404.4, 501.5, 598.6, 695.7, 792.8,
        889.9, 987. ]),
 <BarContainer object of 10 artists>)
```



Special Plot Types

- Boxplot:

```
In [31]: data = [np.random.normal(0, std, 100) for std in range(1, 4)]
    ...:
    ...: # rectangular box plot
    ...: plt.boxplot(data,vert=True,patch_artist=True);
```



Additional Matplotlib Info

- http://www.matplotlib.org - The project web page for matplotlib.
- https://github.com/matplotlib/matplotlib - The source code for matplotlib.
- https://matplotlib.org/stable/ - A large gallery showcasing various types of plots matplotlib can create. Highly recommended!
- https://seaborn.pydata.org/examples/index.html - Seaborn plot gallery.
- http://www.loria.fr/~rougier/teaching/matplotlib - A good matplotlib tutorial.
- http://scipy-lectures.github.io/matplotlib/matplotlib.html - Another good matplotlib reference.