Matplotlib

Introduction

Matplotlib is the "grandfather" library of data visualization with Python. It was created by John Hunter. He created it to try to replicate MatLab's (another programming language) plotting capabilities in Python. So if you happen to be familiar with matlab, matplotlib will feel natural to you.

It is an excellent 2D and 3D graphics library for generating scientific figures.

Some of the 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. Let's learn how to use it! Before continuing this lecture, I encourage you just to explore the official Matplotlib web page: http://matplotlib.org/ (http://matplotlib.org/)

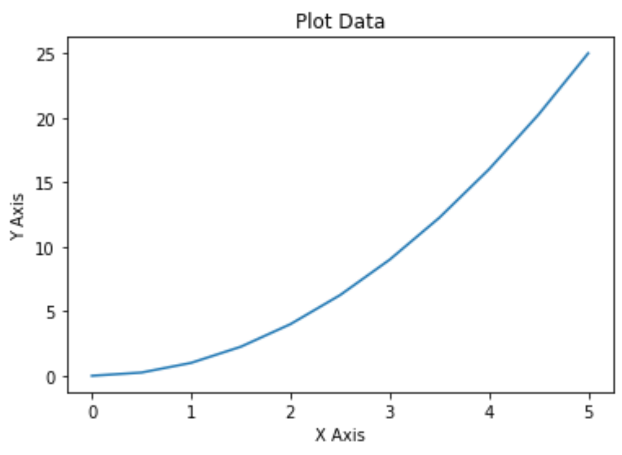
In [2]:
```
#Import the package
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

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

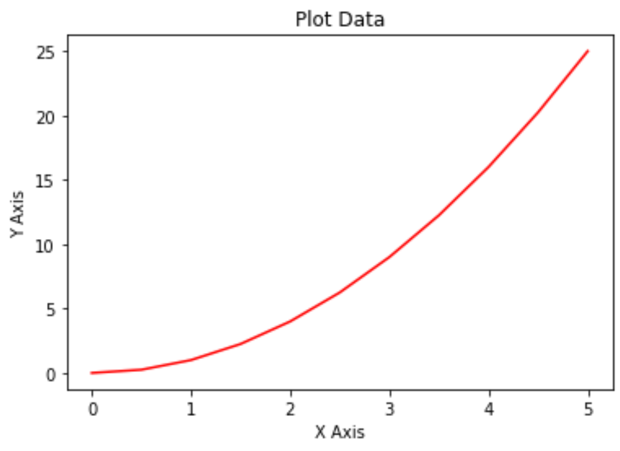
Basic Matplotlib Commands

We can create a very simple line plot using the following (I encourage you to pause and use Shift+Tab along the way to check out the document strings for the functions we are using).

In [4]:
```
plt.plot(x, y)
plt.xlabel('X Axis')
plt.ylabel('Y Axis')
plt.title('Plot Data')
plt.show()
```

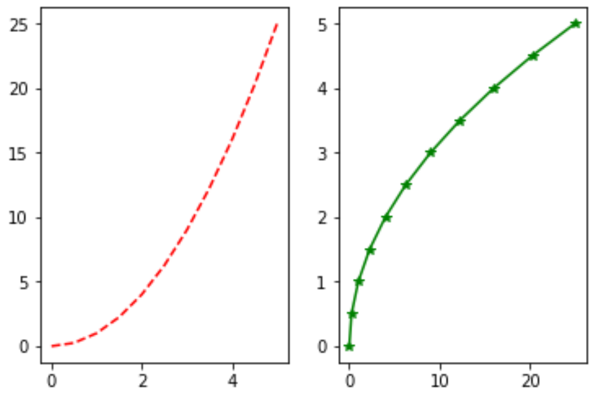


In [6]:
```
plt.plot(x, y, 'r') # 'r' is the color red
plt.xlabel('X Axis')
plt.ylabel('Y Axis')
plt.title('Plot Data')
plt.show()
```

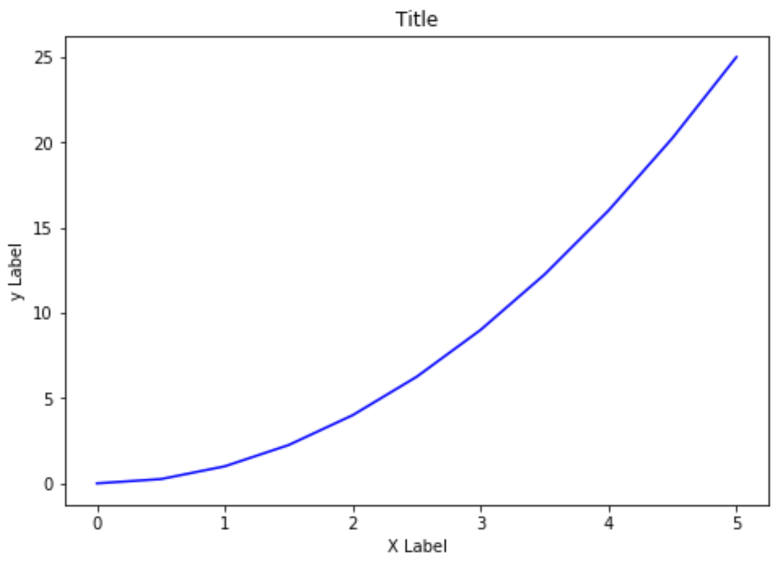


Creating Multiplots on Same Canvas

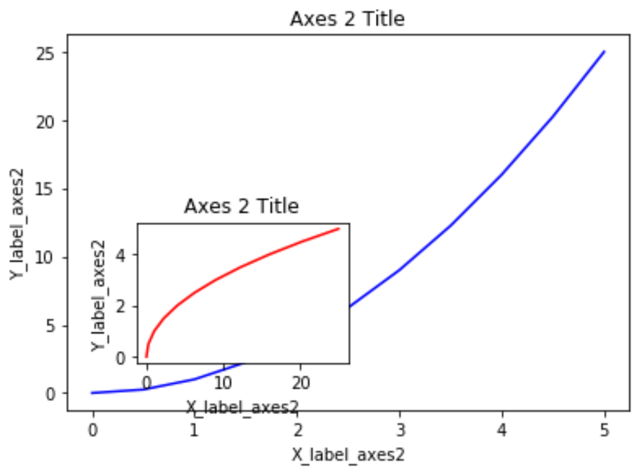
In [7]:
```
# plt.subplot(nrows, ncols, plot_number)
plt.subplot(1,2,1)
plt.plot(x, y, 'r--') # More on color options later
plt.subplot(1,2,2)
plt.plot(y, x, 'g*-');
```



```
In [8]: # Create Figure (empty canvas)
        fig = plt.figure()

        # Add set of axes to figure
        axes = fig.add_axes([0.1, 0.1, 1, 1]) # left, bottom, width, height (ran
        ge 0 to 1)
        # Plot on that set of axes
        axes.plot(x, y, 'b')
        axes.set_xlabel('X Label') # Notice the use of set_ to begin methods
        axes.set_ylabel('y Label')
        axes.set_title('Title')

Out[8]: Text(0.5,1,'Title')
```



```
In [9]: # 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, 0.2, 0.3, 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');
```

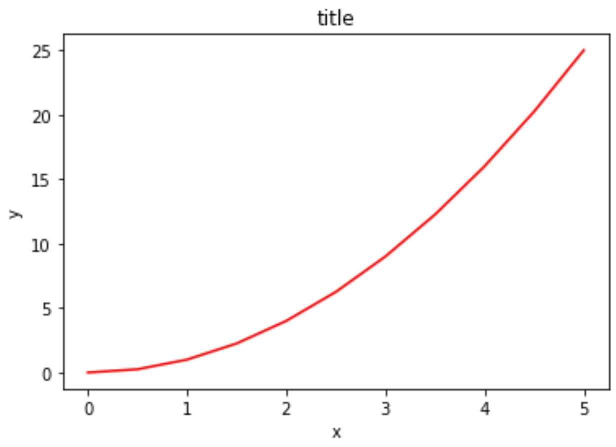


subplots()

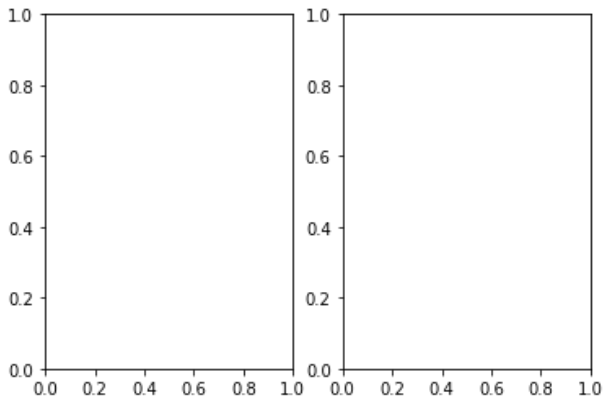
The plt.subplots() object will act as a more automatic axis manager.

Basic use cases:

In [13]:
```
# 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');
```

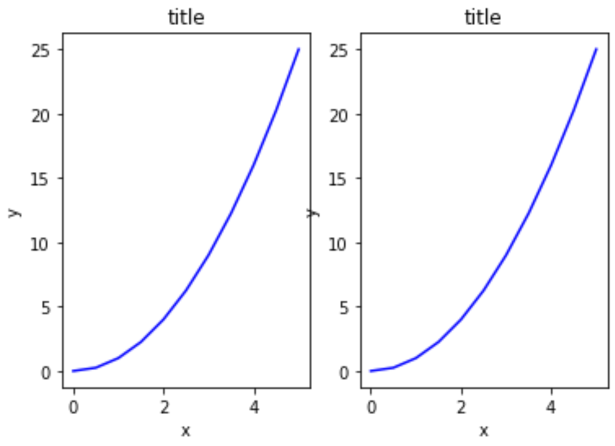


In [14]:
```
# Empty canvas of 1 by 2 subplots
fig, axes = plt.subplots(nrows=1, ncols=2)
```



In [15]:
```python
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[15]:



A common issue with matplolib is overlapping subplots or figures. We ca use **fig.tight_layout()** or **plt.tight_layout()** method, which automatically adjusts the positions of the axes on the figure canvas so that there is no overlapping content:

In [16]:
```python
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()
```

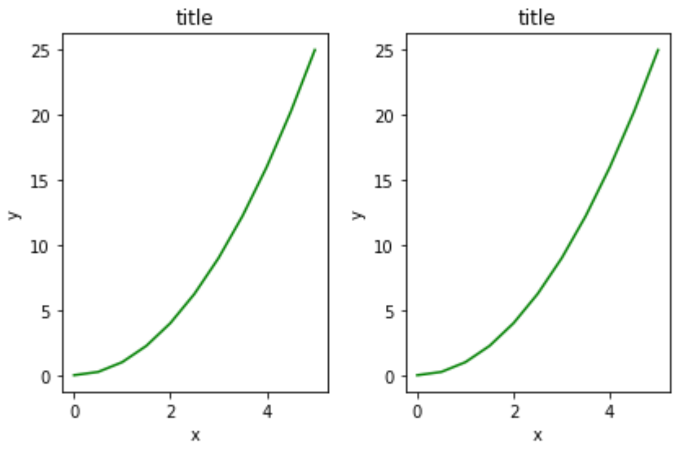


Figure size, aspect ratio and DPI

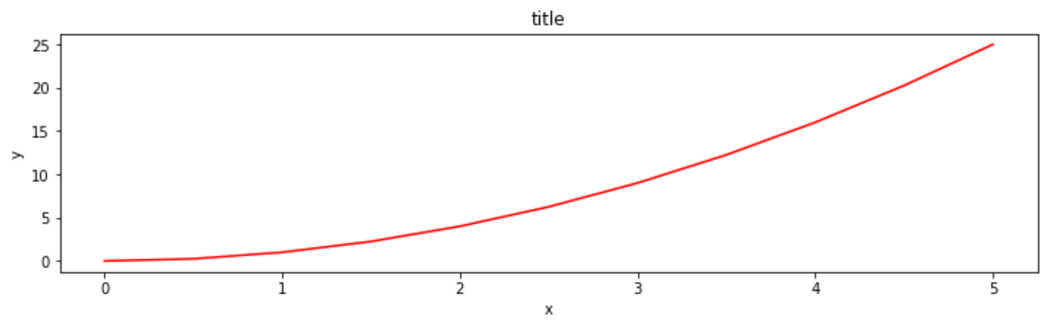
Matplotlib allows the aspect ratio, DPI and figure size to be specified when the Figure object is created. You can use the `figsize` and `dpi` keyword arguments.

- `figsize` is a tuple of the width and height of the figure in inches
- `dpi` is the dots-per-inch (pixel per inch).

For example:

In [18]:
```
fig = plt.figure(figsize=(8,4), dpi=100)
fig, axes = plt.subplots(figsize=(12,3))

axes.plot(x, y, 'r')
axes.set_xlabel('x')
axes.set_ylabel('y')
axes.set_title('title');
```

```
<matplotlib.figure.Figure at 0x7fd2b59ea860>
```



Saving figures

Matplotlib can generate high-quality output in a number formats, including PNG, JPG, EPS, SVG, PGF and PDF. To save a figure to a file we can use the `savefig` method in the `Figure` class:

In [19]:
```
fig.savefig("filename.png")
```

In [21]:
```
#Here we can also optionally specify the DPI and choose between differen
t output formats:
fig.savefig("filename.png", dpi=200)
```

Legends, Labels and titles

Now that we have covered the basics of how to create a figure canvas and add axes instances to the canvas, let's look at how decorate a figure with titles, axis labels, and legends.

Figure titles

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

Axis labels

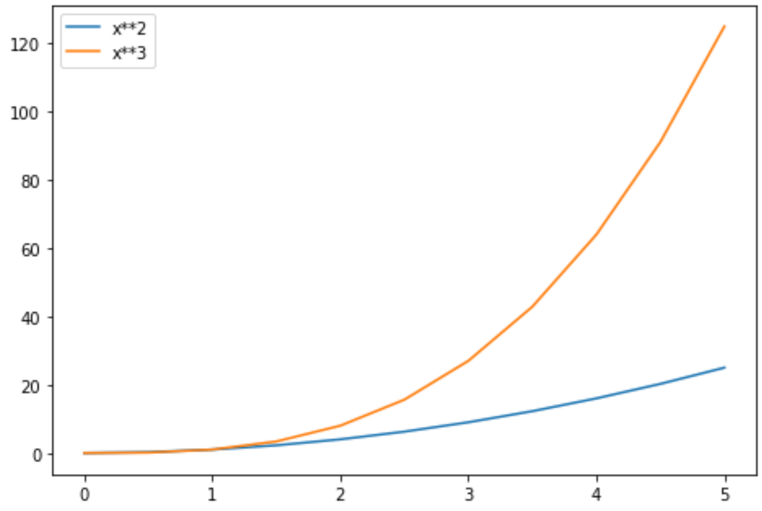
Similarly, with the methods `set_xlabel` and `set_ylabel`, we can set the labels of the X and Y axes:

Legends

You can 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:

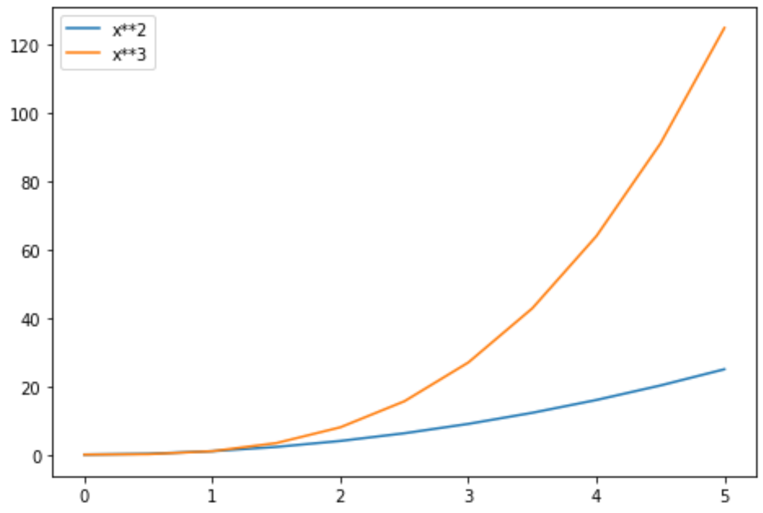
```
In [22]: fig = plt.figure()

         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[22]: <matplotlib.legend.Legend at 0x7fd2e5f9d2e8>
```



```
In [23]: # 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[23]:
```



Setting colors, linewidths, linetypes

Matplotlib gives you *a lot* of options for customizing colors, linewidths, and linetypes.

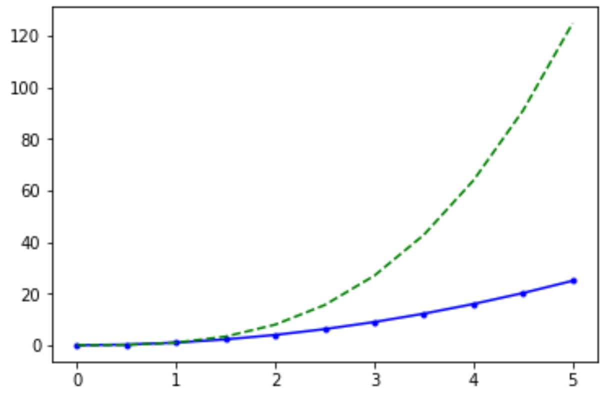
There is the basic MATLAB like syntax (which I would suggest you avoid using for more clairty sake:

Colors with MatLab like syntax

With matplotlib, we can define the colors of lines and other graphical elements in a number of ways. First of all, we can use the MATLAB-like syntax where `'b'` means blue, `'g'` means green, etc. The MATLAB API for selecting line styles are also supported: where, for example, 'b.-' means a blue line with dots:

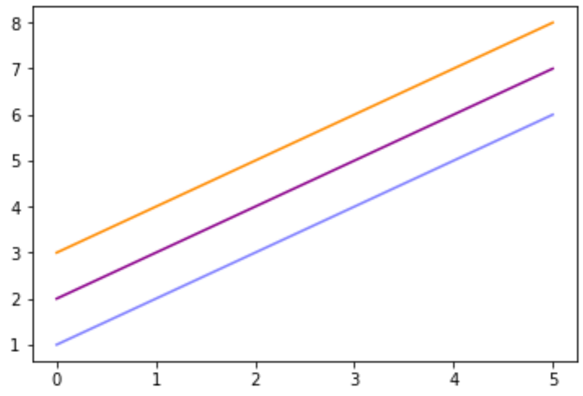
```
In [24]: # MATLAB style line color and style
         fig, ax = plt.subplots()
         ax.plot(x, x**2, 'b.-') # blue line with dots
         ax.plot(x, x**3, 'g--') # green dashed line
```

```
Out[24]: [<matplotlib.lines.Line2D at 0x7fd2b9873240>]
```



```
In [25]: 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
```

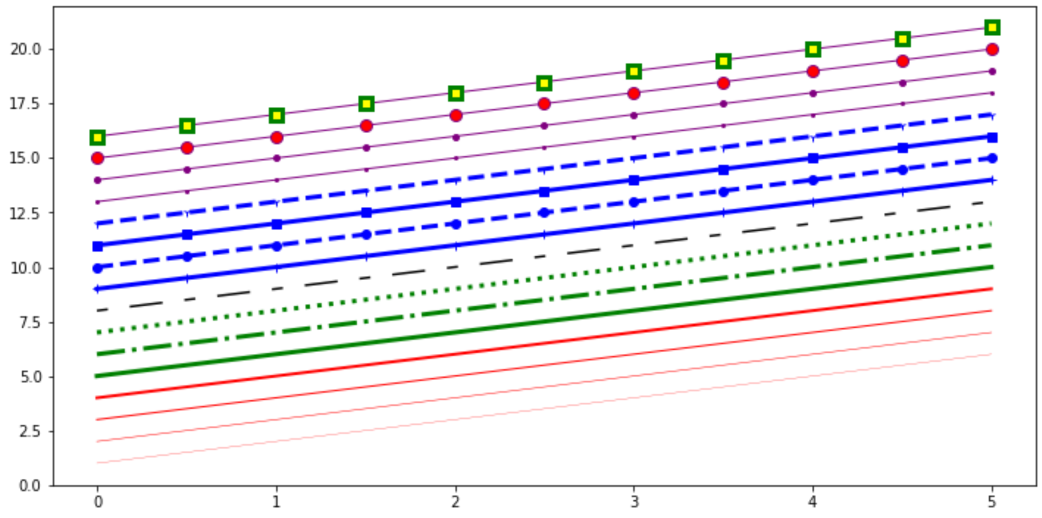
```
Out[25]: [<matplotlib.lines.Line2D at 0x7fd2e600a6a0>]
```



Line and marker styles

To change the line width, we can use the `linewidth` or `lw` keyword argument. The line style can be selected using the `linestyle` or `ls` keyword arguments:

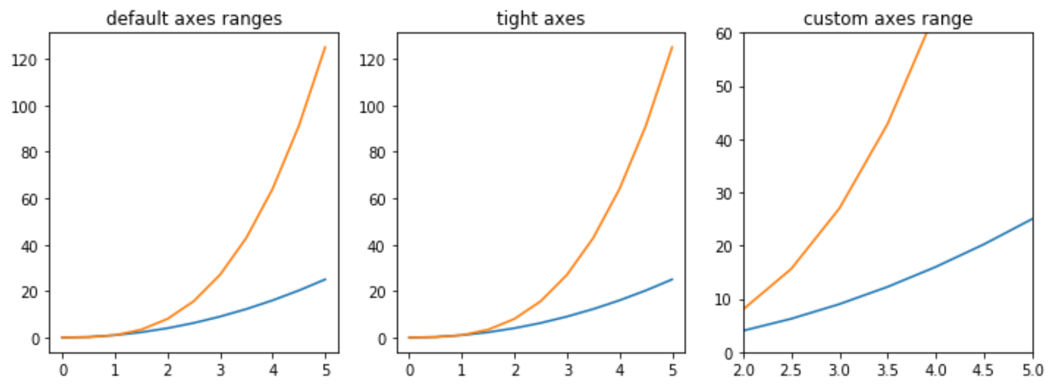
```
In [26]: fig, ax = plt.subplots(figsize=(12,6))

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)

# 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=':')

# 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="gr
een");
```



Plot range

We can 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:

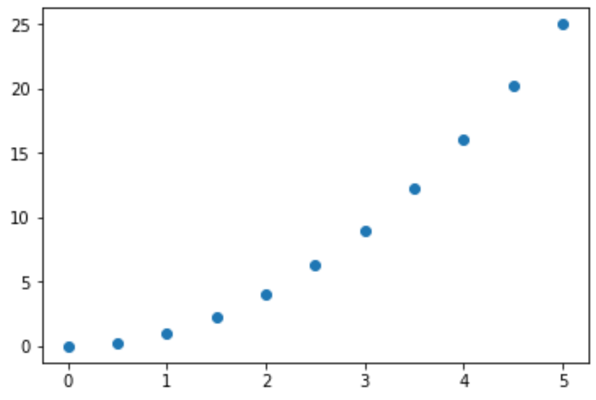
```
In [27]: fig, axes = plt.subplots(1, 3, figsize=(12, 4))

         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

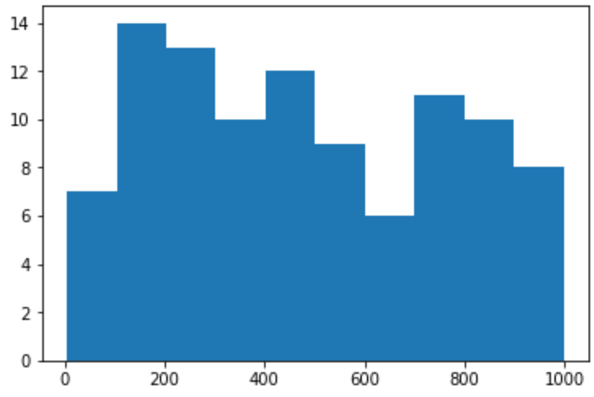
There are many specialized plots we can create, such as barplots, histograms, scatter plots, and much more. Most of these type of plots we will actually create using seaborn, a statistical plotting library for Python. But here are a few examples of these type of plots:

```
In [28]: plt.scatter(x,y)

Out[28]: <matplotlib.collections.PathCollection at 0x7fd2b51732e8>
```

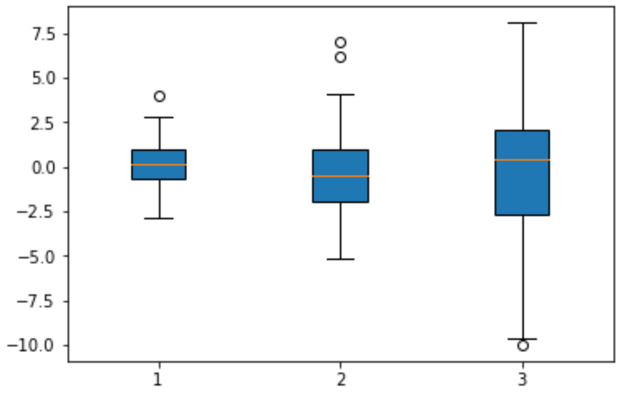


```
In [29]: from random import sample
         data = sample(range(1, 1000), 100)
         plt.hist(data)
```

```
Out[29]: (array([ 7., 14., 13., 10., 12.,  9.,  6., 11., 10.,  8.]),
          array([  5. , 104.2, 203.4, 302.6, 401.8, 501. , 600.2, 699.4, 798.6,
                 897.8, 997. ]),
          <a list of 10 Patch objects>)
```



```
In [30]: data = [np.random.normal(0, std, 100) for std in range(1, 4)]

         # rectangular box plot
         plt.boxplot(data,vert=True,patch_artist=True);
```


7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
1
2
3