In [67]:
```python
___
# Matplotlib Overview Lecture
```
Out[67]:
```
Text(0.5, 1.0, 'String Title Here')
```

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/

Installation

You'll need to install matplotlib first with either:

```
conda install matplotlib
```

or pip install matplotlib

Importing

Import the `matplotlib.pyplot` module under the name `plt` (the tidy way):

In [68]:
```python
import matplotlib.pyplot as plt
```

You'll also need to use this line to see plots in the notebook:

In [69]:
```python
%matplotlib inline
```

That line is only for jupyter notebooks, if you are using another editor, you'll use: **plt.show()** at the end of all your plotting commands to have the figure pop up in another window.

Basic Example

Let's walk through a very simple example using two numpy arrays:

Example

Let's walk through a very simple example using two numpy arrays. You can also use lists, but most likely you'll be passing numpy arrays or pandas columns (which essentially also behave like arrays).

The data we want to plot:

In [70]:
```python
import numpy as np
x = np.linspace(0, 5, 11)
y = x ** 2
```

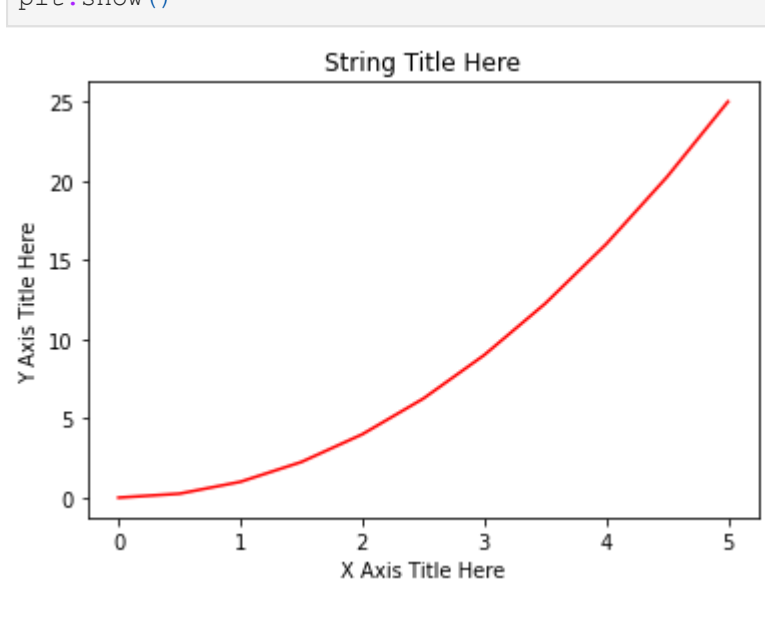
In [71]:
```python
x
```
Out[71]:
```
array([0. , 0.5, 1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5, 5. ])
```

In [72]:
```python
y
```
Out[72]:
```
array([ 0.  ,  0.25,  1.  ,  2.25,  4.  ,  6.25,  9.  , 12.25, 16.  ,
       20.25, 25.  ])
```

Basic Matplotlib Commands

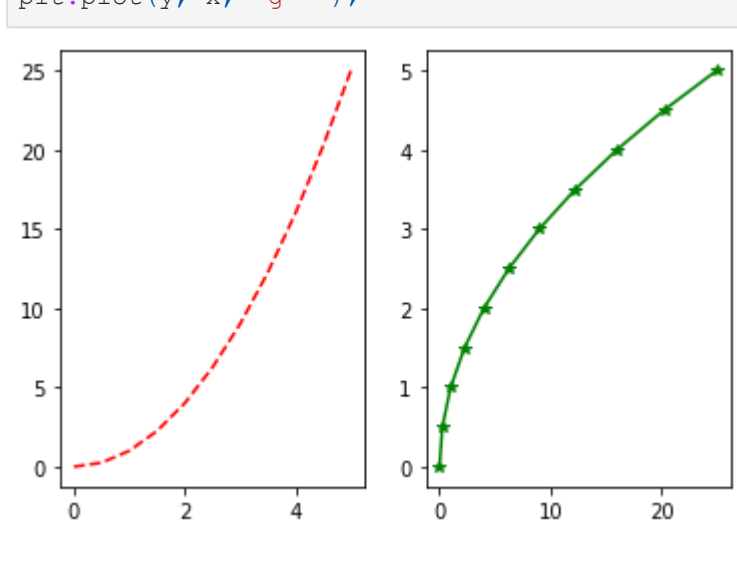
We can create a very simple line plot using the following (I encourage you to pause and use Shift+Tab along the way to check out the document strings for the functions we are using).

In [73]:
```python
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()
```



Creating Multiplots on Same Canvas

In [74]:
```python
# 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*-');
```



Matplotlib Object Oriented Method

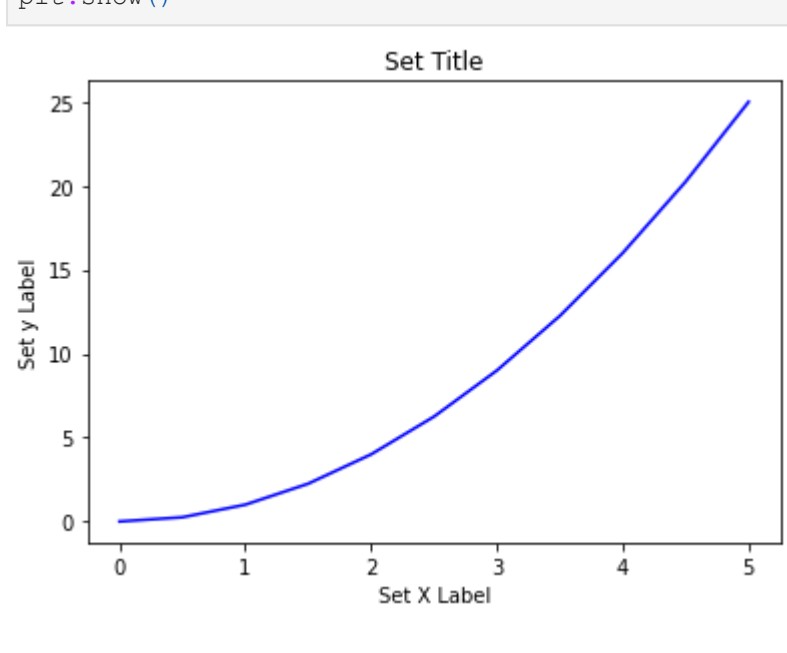
Now that we've seen the basics, let's break it all down with a more formal introduction of Matplotlib's Object Oriented API. This means we will instantiate figure objects and then call methods or attributes from that object.

Introduction to the 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. This approach is nicer when dealing with a canvas that has multiple plots on it.

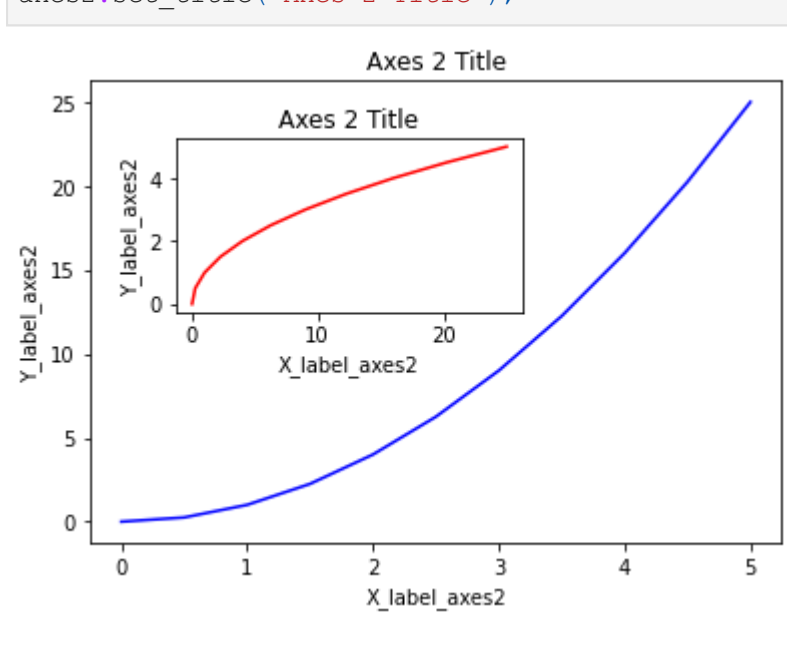
To begin we create a figure instance. Then we can add axes to that figure:

In [75]:
```python
# 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')
plt.show()
```



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

In [76]:
```python
# 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');
```

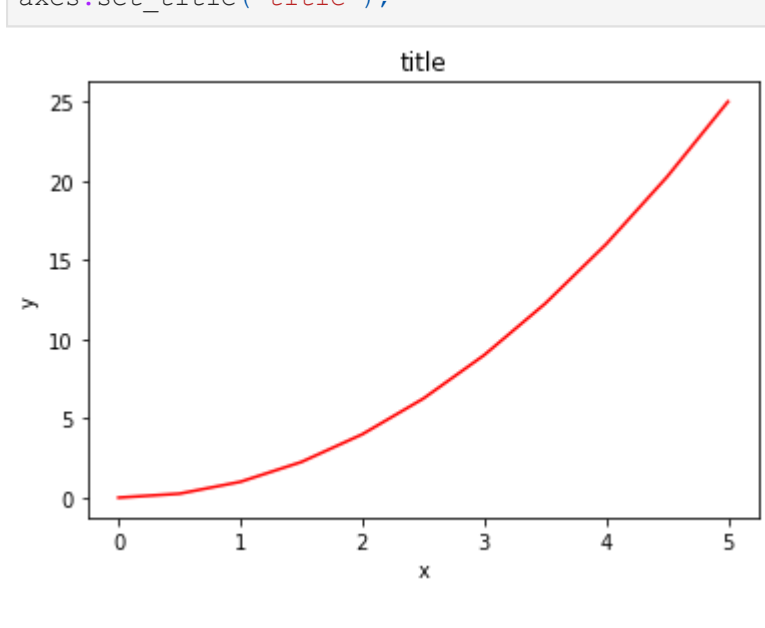


subplots()

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

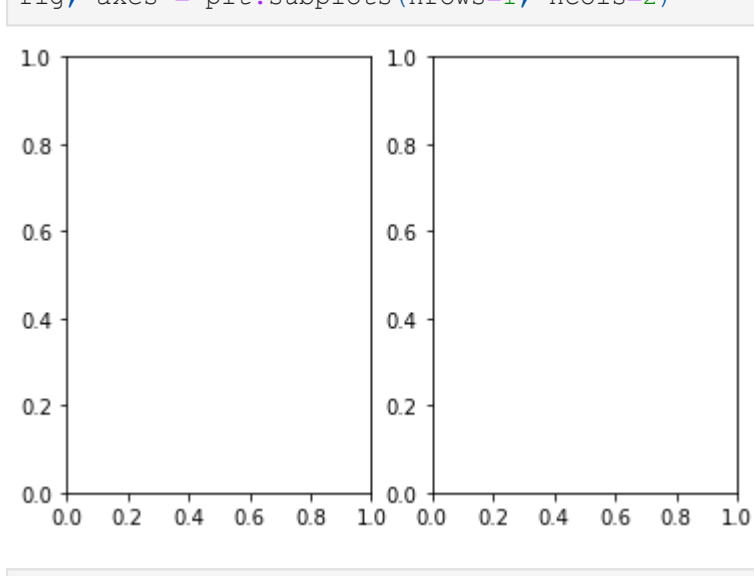
Basic use cases:

In [77]:
```python
# 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');
```



Then you can specify the number of rows and columns when creating the subplots() object:

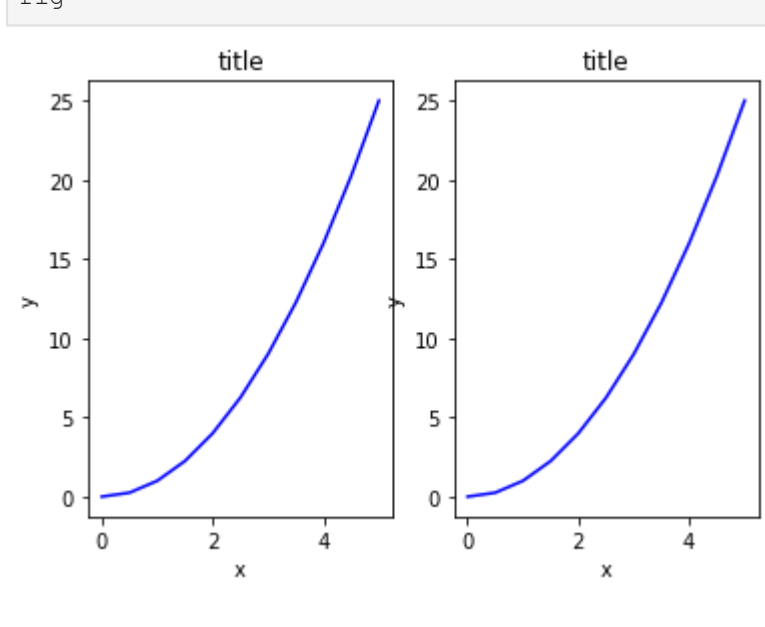
In [78]:
```python
# Empty canvas of 1 by 2 subplots
fig, axes = plt.subplots(nrows=1, ncols=2)
```



In [79]:
```python
# Axes is an array of axes to plot on
axes
```
Out[79]:
```
array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)
```

We can iterate through this array:

In [80]:
```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[80]:



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 [81]:
```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()
```

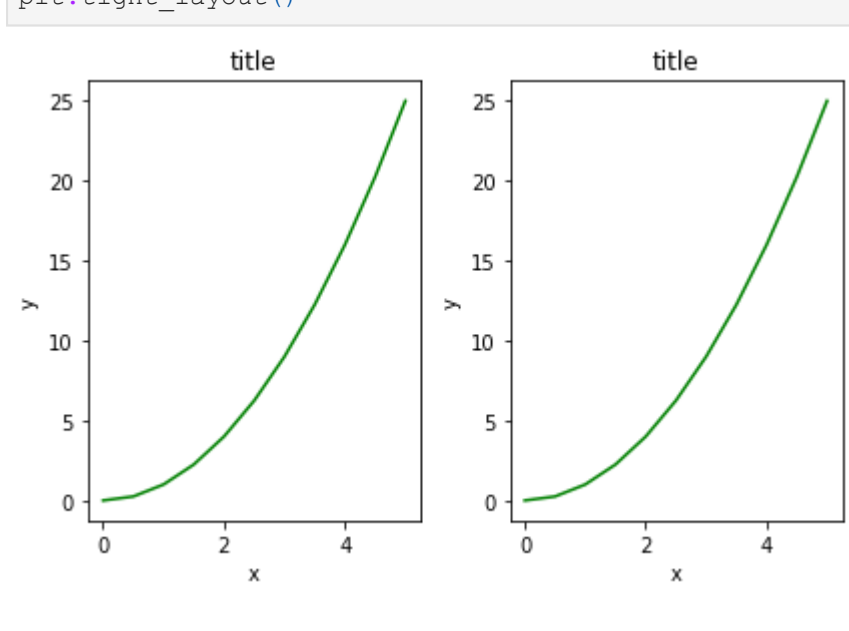


Figure size, aspect ratio and DPI

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.

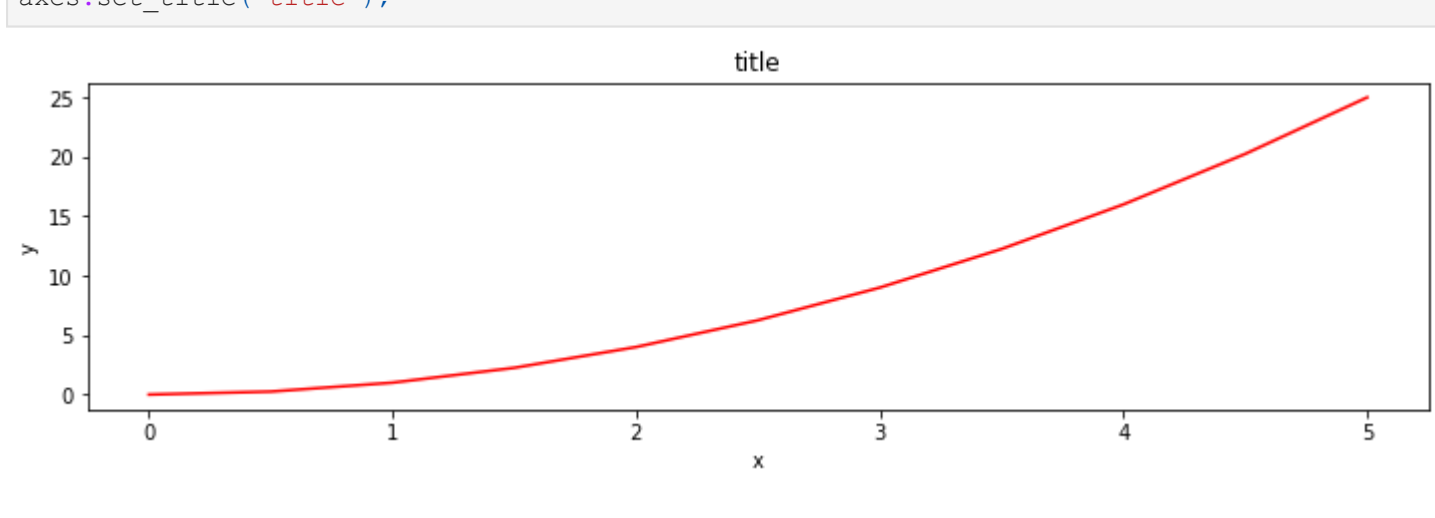
- `figsize` is a tuple of the width and height of the figure in inches
- `dpi` is the dots-per-inch (pixel per inch).

For example:

In [82]:
```python
fig = plt.figure(figsize=(8,4), dpi=100)
```
```
<Figure size 800x400 with 0 Axes>
```

The same arguments can also be passed to layout managers, such as the `subplots` function:

In [100…
```python
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

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 [84]:
```python
fig.savefig("filename.png")
```

Here we can also optionally specify the DPI and choose between different output formats:

In [102…
```python
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:

In [86]:
```python
ax.set_title("title");
```

Axis labels

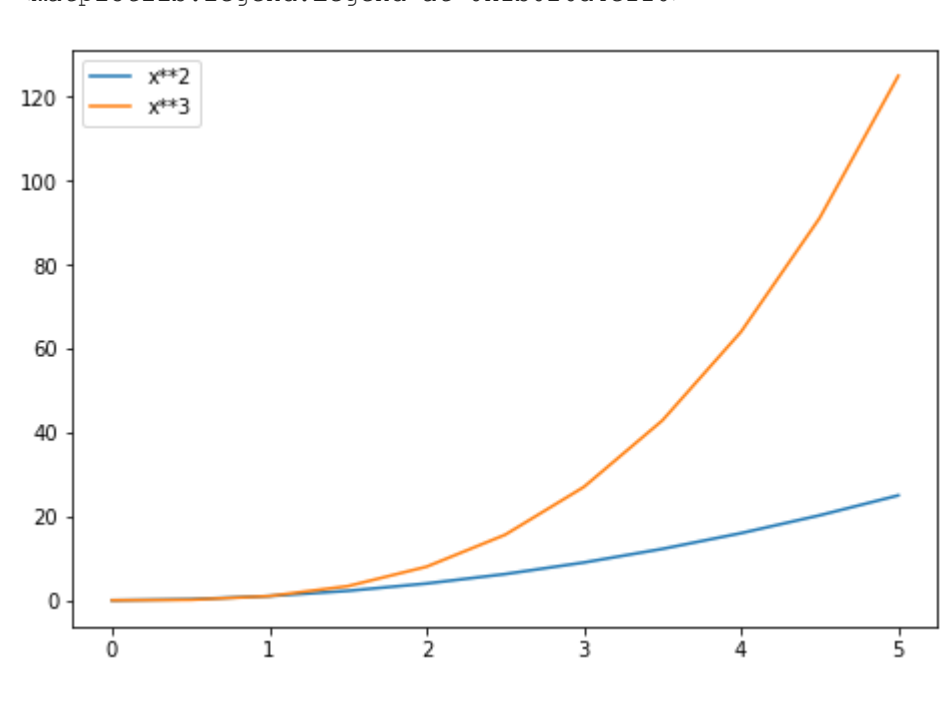
Similarly, with the methods `set_xlabel` and `set_ylabel`, we can set the labels of the X and Y axes:

In [87]:
```python
ax.set_xlabel("x")
ax.set_ylabel("y");
```

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:

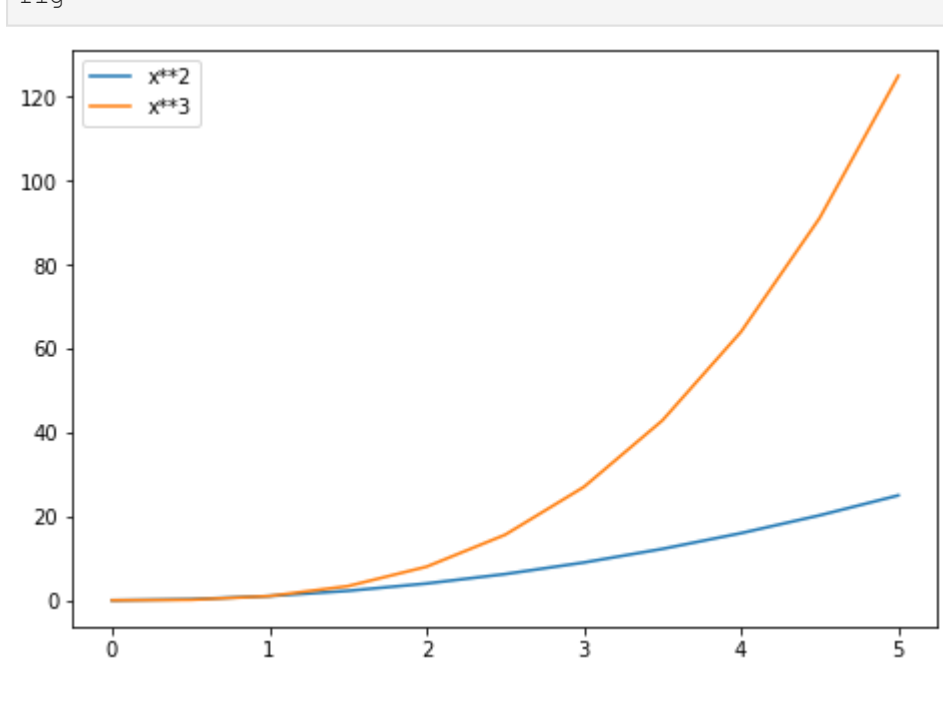
In [88]:
```python
fig = plt.figure()

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[88]:
```
<matplotlib.legend.Legend at 0x1b026d4ef10>
```



Notice how are legend overlaps some of the actual plot!

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. See the documentation page for details. Some of the most common **loc** values are:

In [89]:
```python
# 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[89]:



Setting colors, linewidths, linetypes

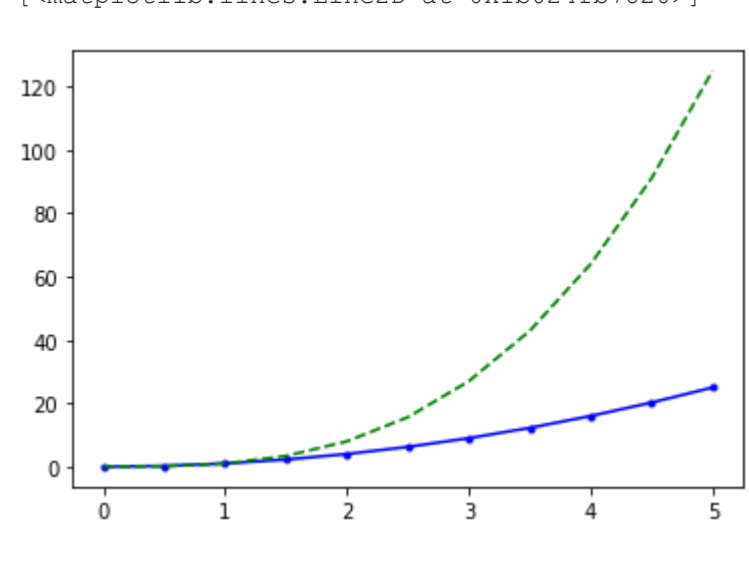
Matplotlib gives you *a lot* of options for customizing colors, linewidths, and linetypes.

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:

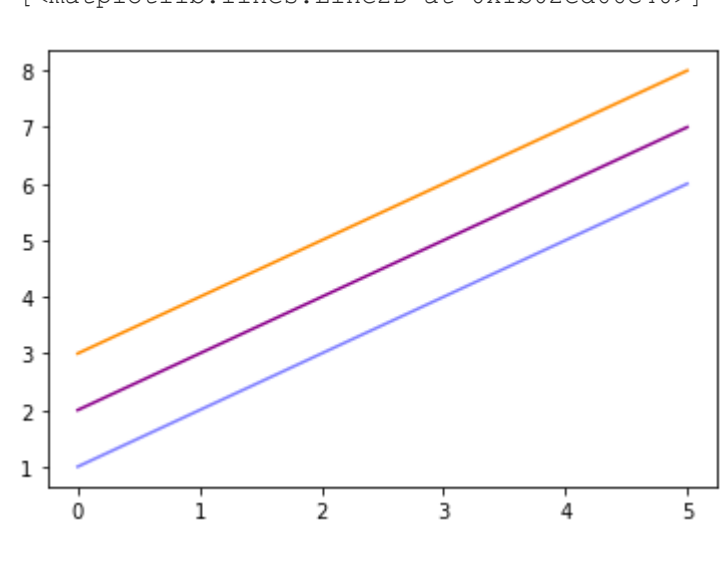
In [90]:
```python
# MATLAB style line color and style
fig, ax = plt.subplots()
ax.plot(x, x**2, 'b.-') # blue line with dots
ax.plot(x, x**3, 'g--') # green dashed line
```
Out[90]:
```
[<matplotlib.lines.Line2D at 0x1b024fb7820>]
```



Colors with the color= parameter

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.

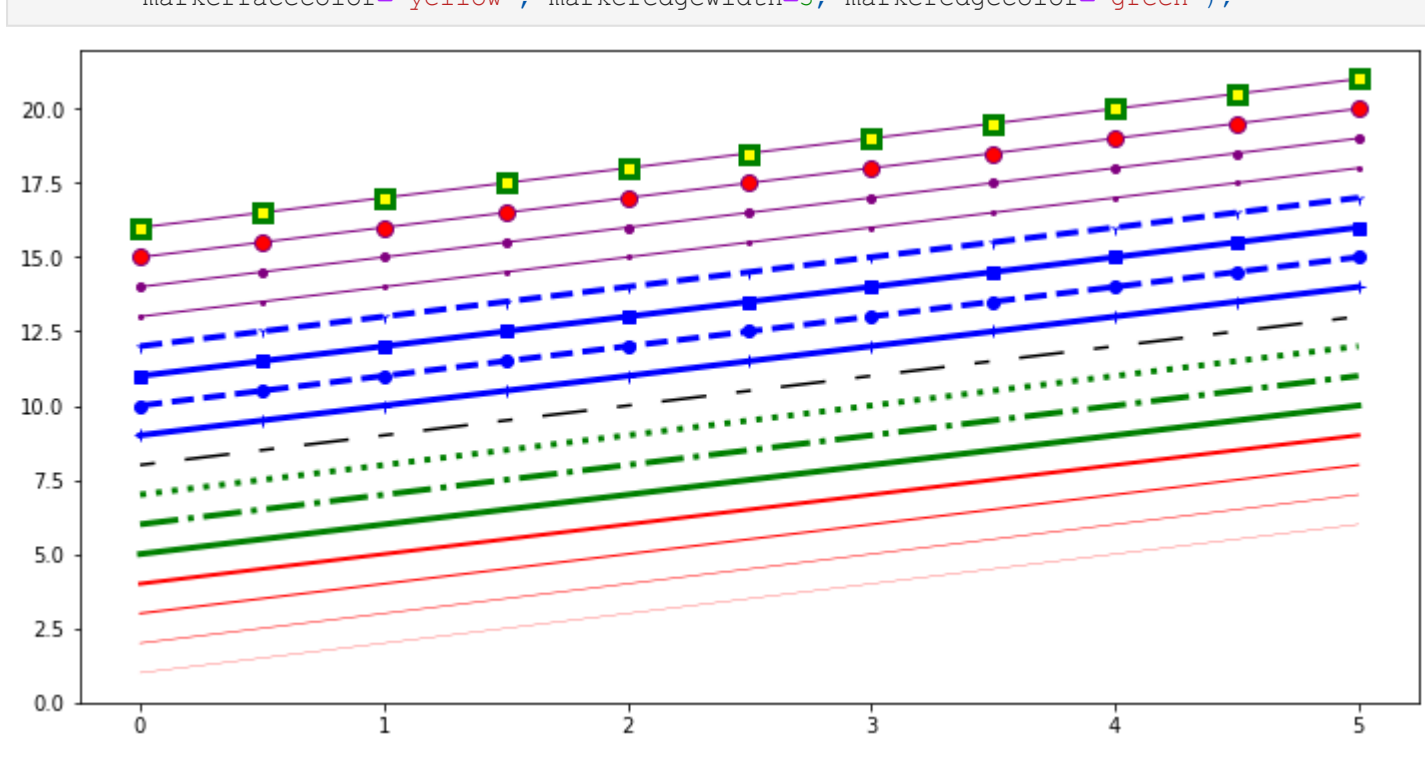
In [104…
```python
fig, ax = plt.subplots()

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[104]:
```
[<matplotlib.lines.Line2D at 0x1b02ed66c40>]
```



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 [92]:
```python
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="green");
```



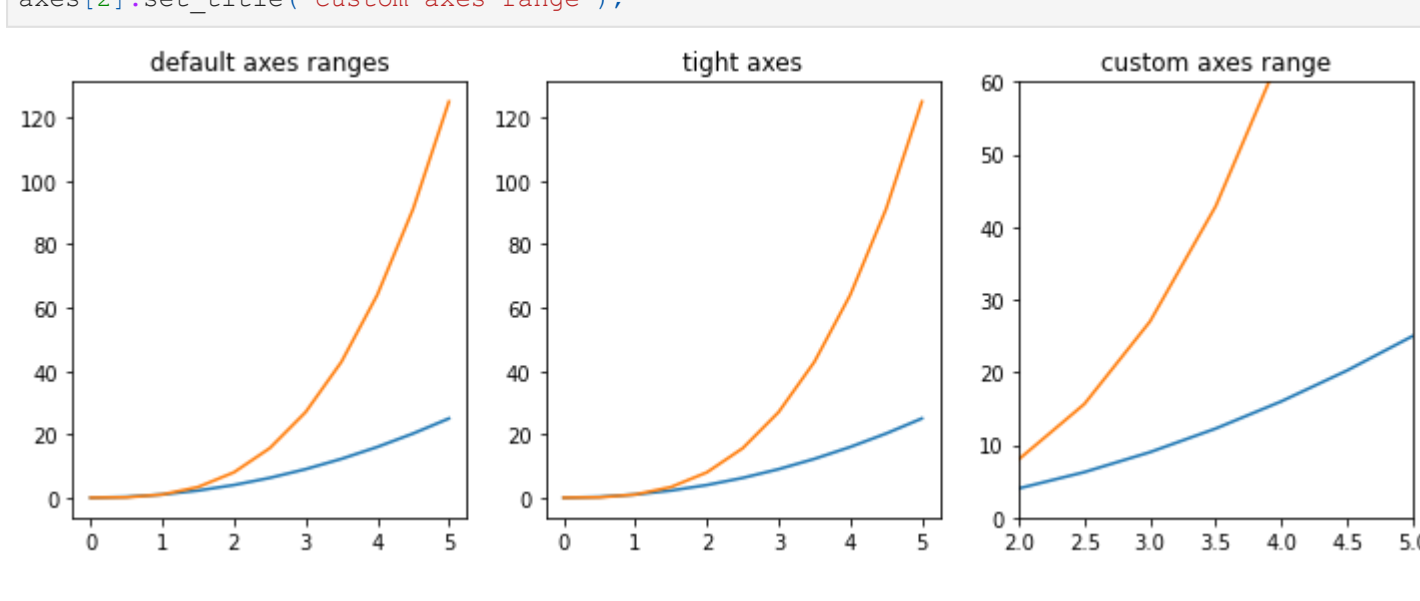
Control over axis appearance

In this section we will look at controlling axis sizing properties in a matplotlib figure.

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:

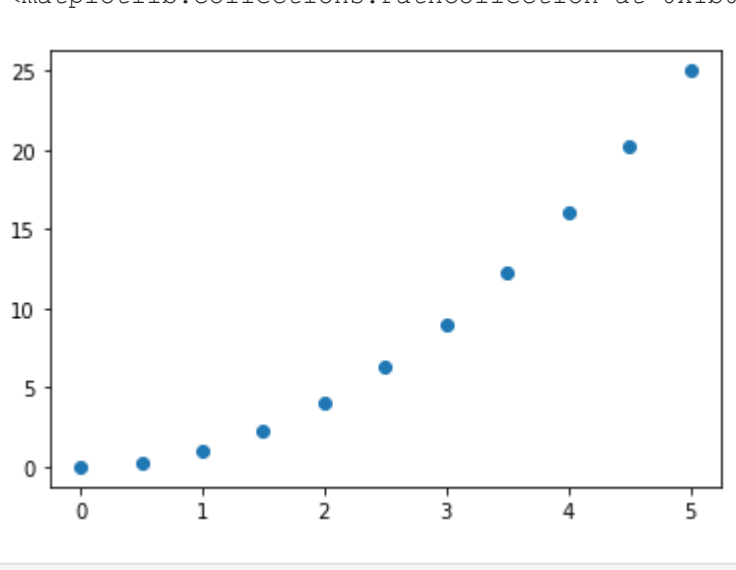
In [93]:
```python
fig, axes = plt.subplots(1, 3, figsize=(12, 4))

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");
```



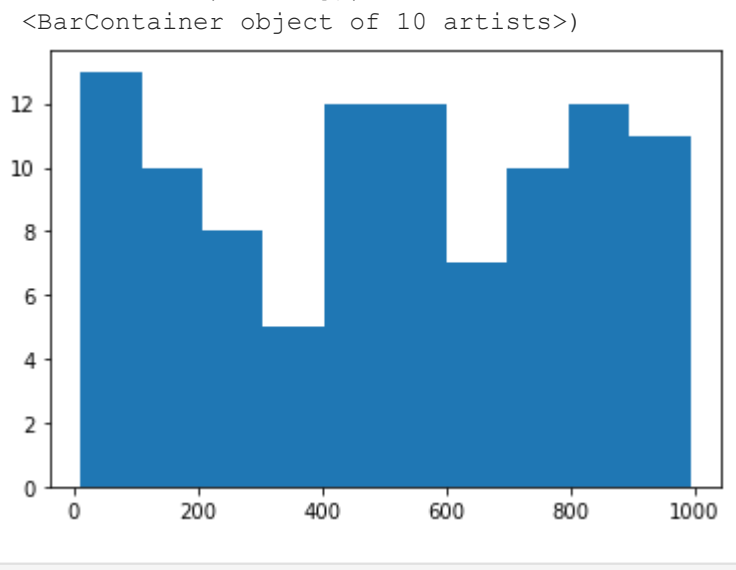
Special Plot Types

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 pandas. But here are a few examples of these type of plots:

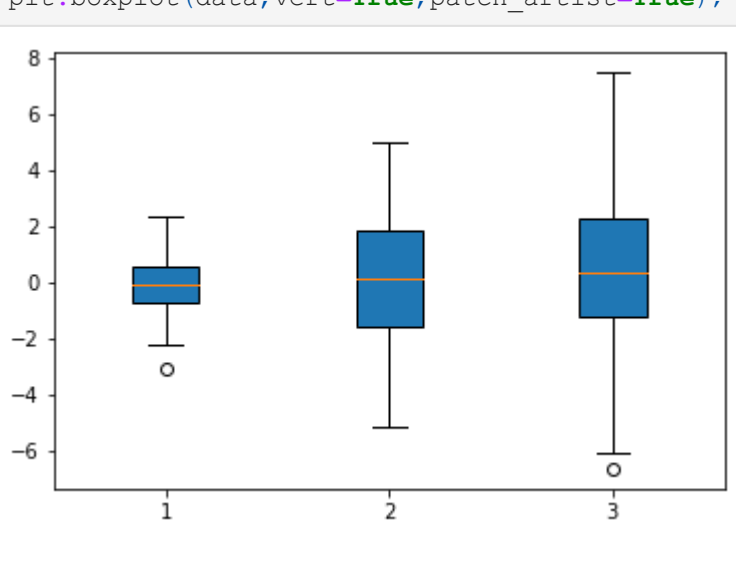
In [94]:
```python
plt.scatter(x,y)
```
Out[94]:
```
<matplotlib.collections.PathCollection at 0x1b0285b3550>
```



In [95]:
```python
from random import sample
data = sample(range(1, 1000), 100)
plt.hist(data)
```
Out[95]:
```
(array([13., 10.,  8.,  5., 12., 12.,  7., 10., 12., 11.]),
 array([ 11. , 109.1, 207.2, 305.3, 403.4, 501.5, 599.6, 697.7, 795.8,
        893.9, 992. ]),
 <BarContainer object of 10 artists>)
```



In [108…
```python
data = [np.random.normal(0, std, 100) for std in range(1, 4)]

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



Further reading

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