Data Science

Intro to Matplotlib

10.11.22

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.

Matplotlib allows you to create reproducible figures programmatically. Let's learn how to use it! I encourage you just to explore the official Matplotlib web page: http://matplotlib.org/

Matplotlib

Some of the major pros of Maplotlib 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

Installation

You'll need to install matplotlib first with either:

```
pip install matplotlib
!pip install matplotlib
```

To import you will need to type:

```
import matplotlib.pyplot as plt
%matplotlib inline
```

You need the “%matplotlib inline” to see plots in the Jupyter notebook. 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.

Matplotlib

Basic example

Lets try to walk through a very simple example. If you want to open your jupyter notebook to try the example you may do so.

We walk through a basic example:

Let first create two arrays, one for an x variable and another for the y variable for our graph.

```
import numpy as np
x = np.linspace(0,5,11)
#creates array with 11 items from 0 to 5
y = x ** 2
#creates array with 11 items where each item is squared
```

Matplotlib

Basic example continued…

Here are some basic matplotlib commands,

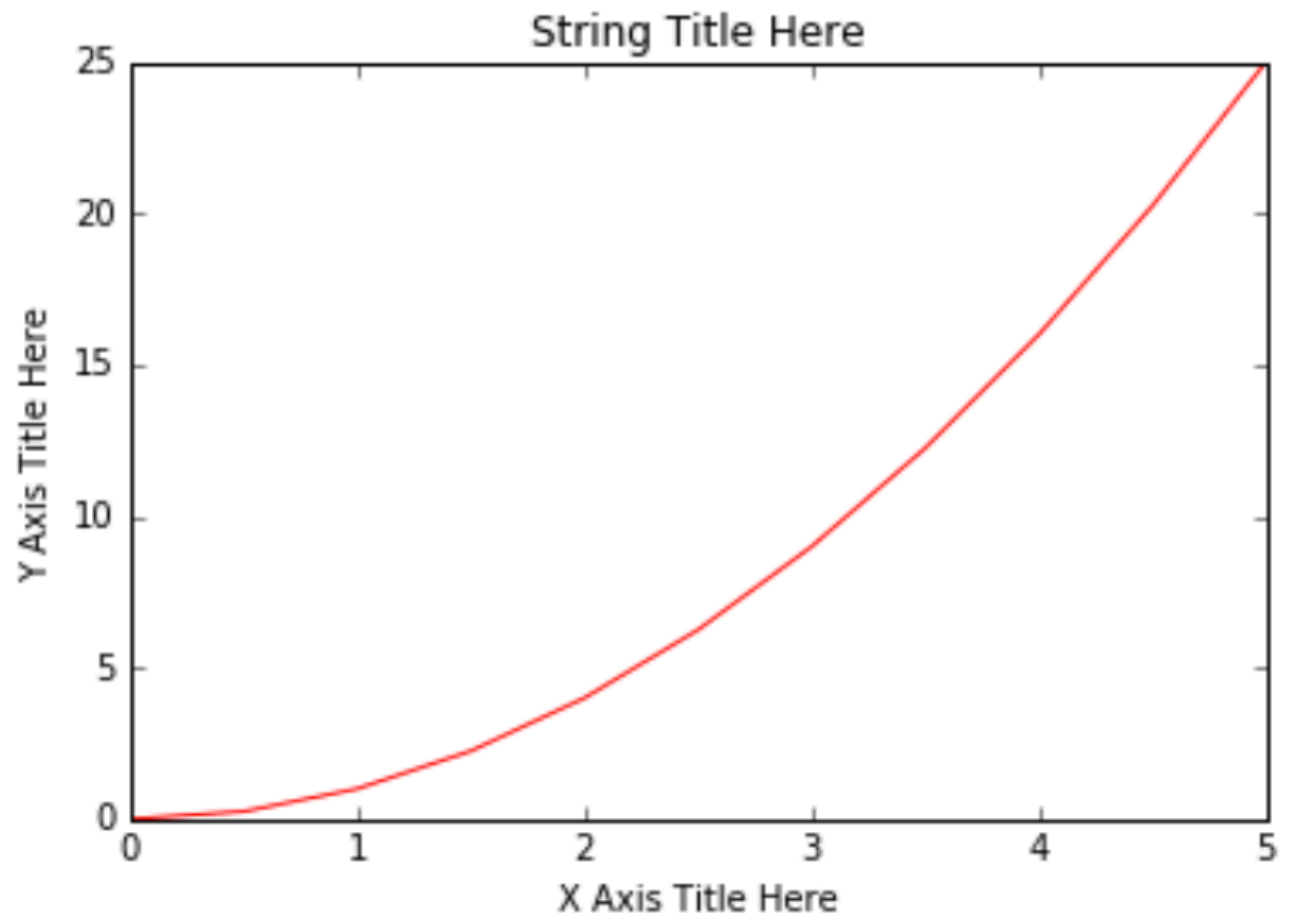
We can create a very simple line plot using the following:

```
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()
```

Matplotlib

Basic Commands

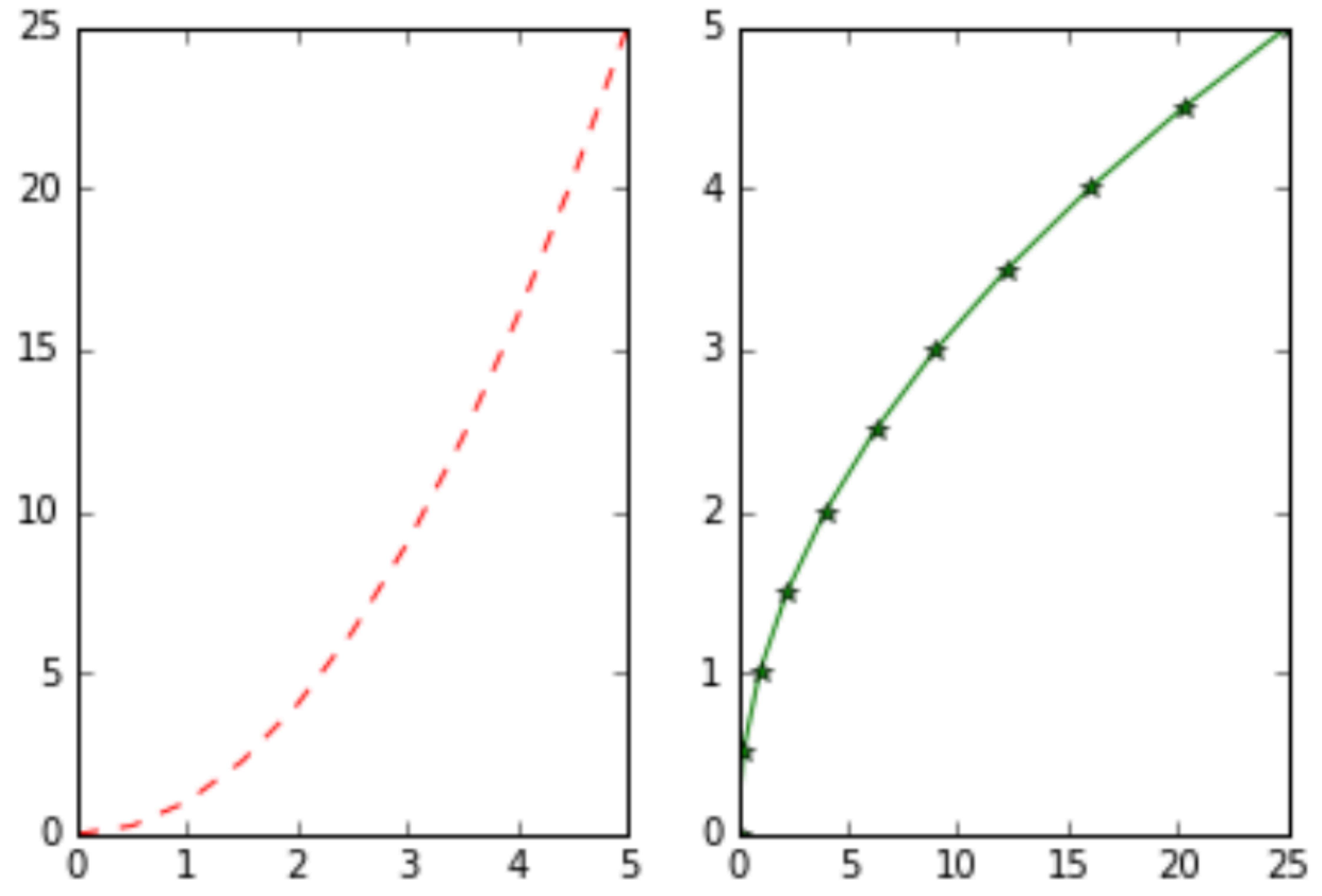
- `plt.plot(x, y, ‘r')` will plot the points (x,y)
- `plt.xlabel('X Axis Title Here’)` will plot the label for the x axis
- `plt.ylabel('Y Axis Title Here’)` will plot the label for the y axis
- `plt.title('String Title Here’)` will plot the title you want for your graph.
- `plt.show()` as we’ve mentioned before is to actually see our graph. This won’t be necessary for our notebook.



Matplotlib

Creating multiples for our Canvas

- Let’s walk through another example:
- `plt.subplot(1,2,1)` the parameters are plt.subplot(nrows, ncols, plot_number) so in this case we will have a plot with row 1 and 2 columns and this subplot is the first graph.
- plt.plot(x, y, ‘r--') we will talk more about color options later.
- plt.subplot(1,2,2) this will be a subplot of second graph in the 1 row, 2 column graph
- 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.

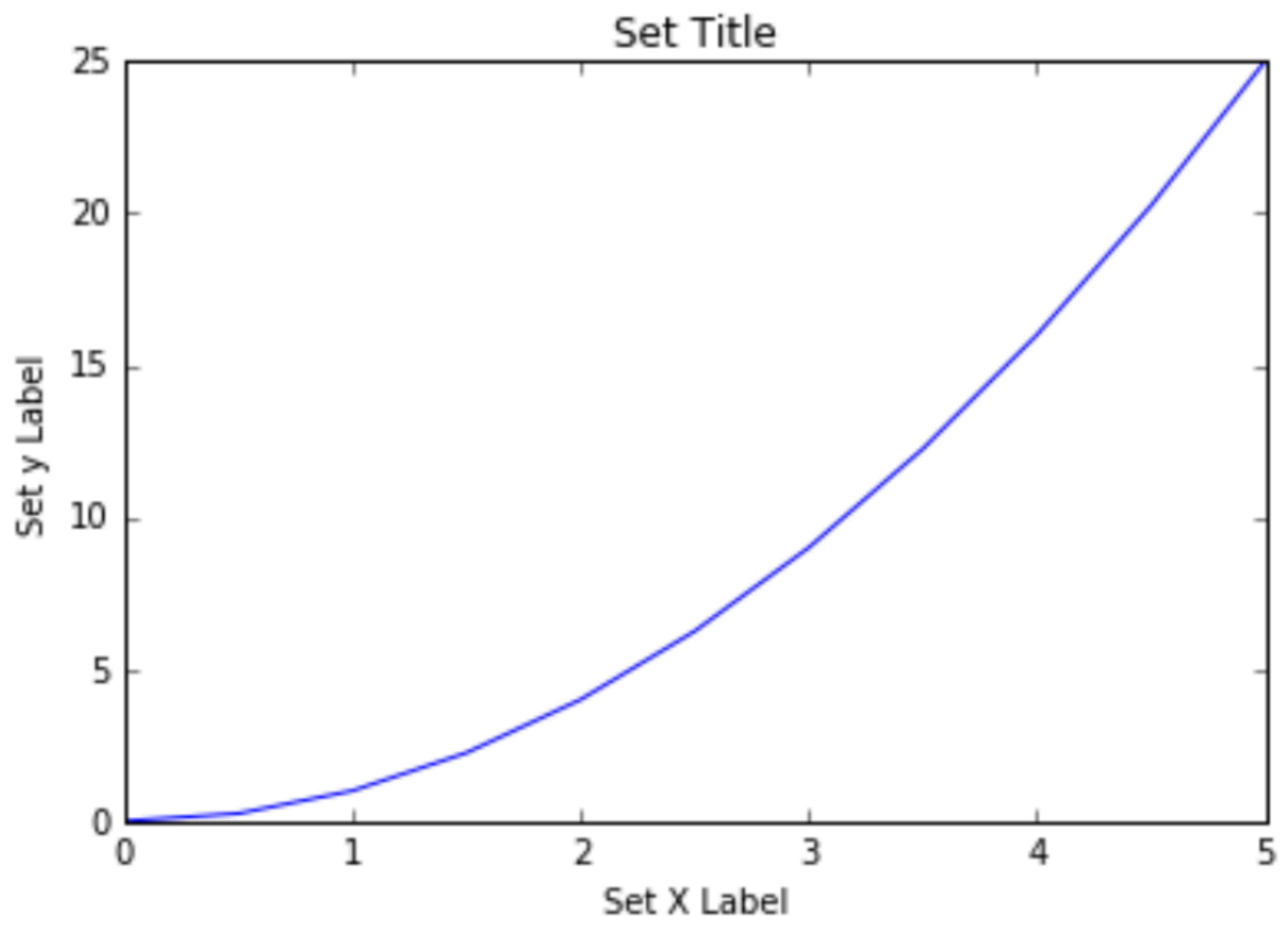
The main idea in using the more formal Object Oriented method is to create figure objects and then just call methods or attributes off of that object. This approach is nicer when dealing with a canvas that has multiple plots on it. To begin we create a figure instance.

Then we can add axes to that figure:

Matplotlib

Object oriented method

- `fig = plt.figure()` This will create a Figure i.e. an empty canvas
- `axes = fig.add_axes([0.1, 0.1, 0.8, 0.8])` Then we add set of axes to figure. The parameters in my [ ] bracket are: left, bottom, width and height (range 0 to 1)
- `axes.plot(x, y, ‘b')` This will plot on that set of axes
- `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')`



Matplotlib

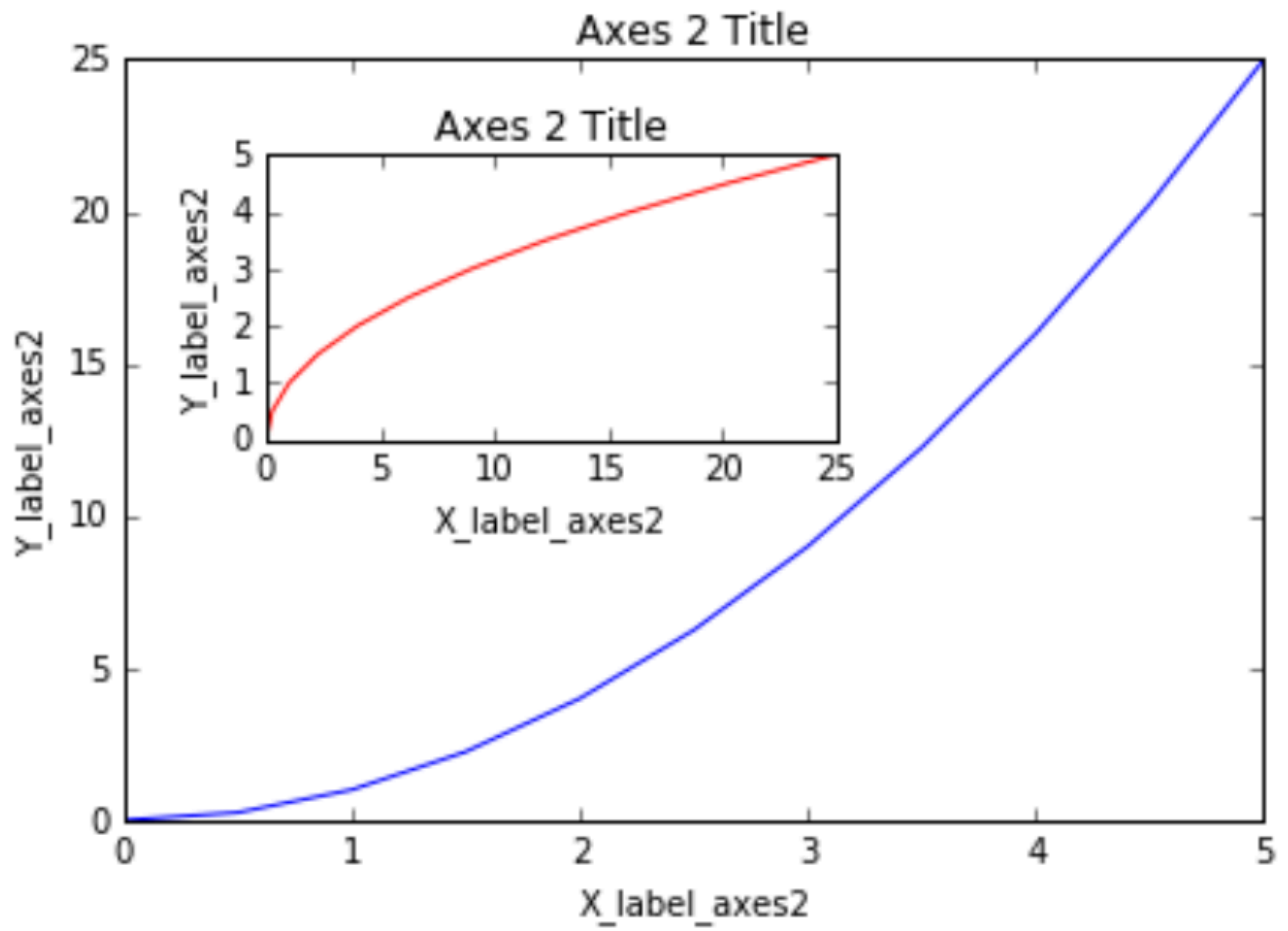
Object oriented method

Why is the object oriented method helpful?

The code can get 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.

Matplotlib

- `fig = plt.figure()`
- `axes1 = fig.add_axes([0.1, 0.1, 0.8, 0.8])`
  This will be our main axes
- `axes2 = fig.add_axes([0.2, 0.5, 0.4, 0.3])`
  Now insert axes
- `axes1.plot(x, y, ‘b')` This creates a larger Figure Axes 1
- `axes1.set_xlabel('X_label_axes2')`
- `axes1.set_ylabel('Y_label_axes2')`
- `axes1.set_title('Axes 2 Title')`
- `axes2.plot(y, x, ‘r')` We not insert figure axes 2.
- `axes2.set_xlabel('X_label_axes2')`
- `axes2.set_ylabel('Y_label_axes2')`
- `axes2.set_title('Axes 2 Title’);`



Marplotlib

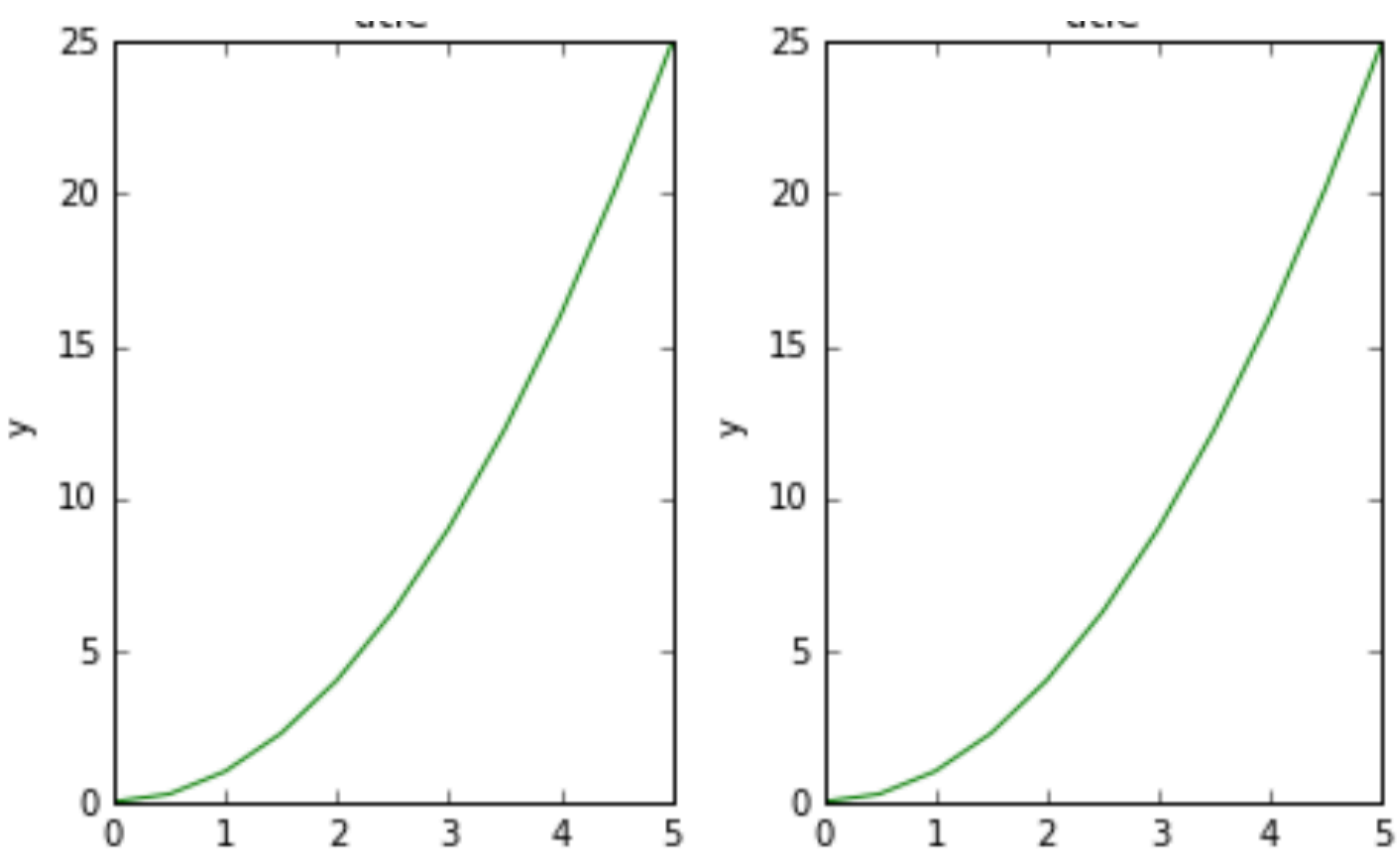
Subplots

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:

```
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()
```


y
0
5
10
15
20
25
0
1
2
3
4
5
y
0
5
10
15
20
25
0
1
2
3
4
5

Matplotlib

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.

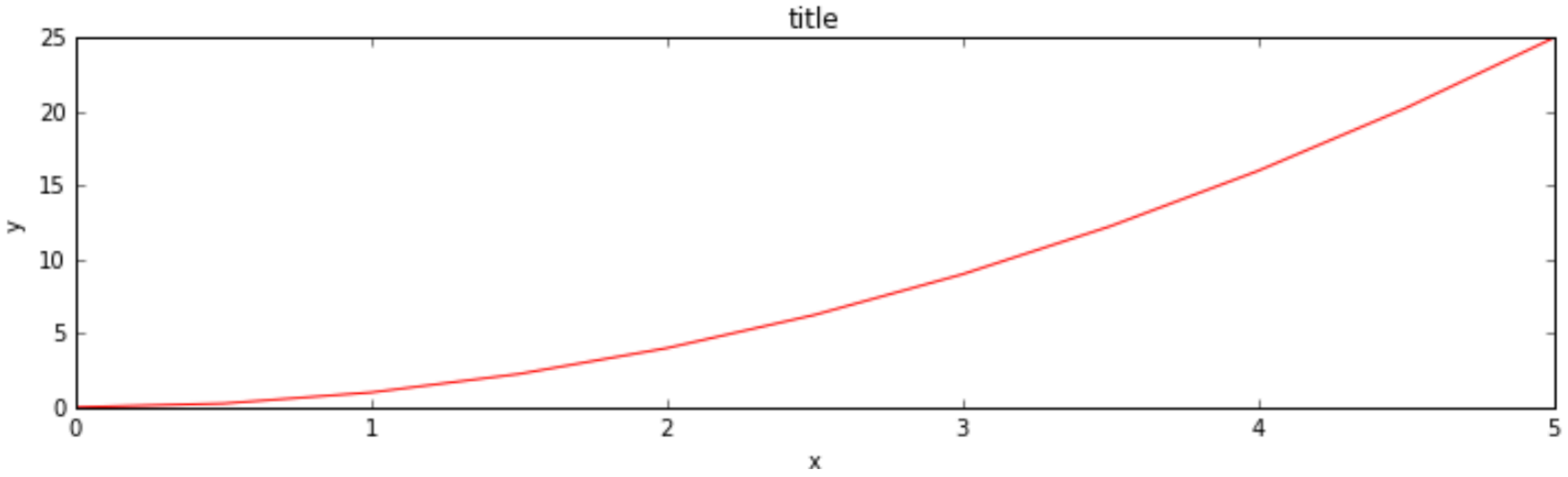
- **figsize** is a tuple of the width and height of the figure in inches
- **dpi** is the dots-per-inch (pixel per inch).

```
fig = plt.figure(figsize=(8,4), dpi=100)
```

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

```
fig, axes = plt.subplots(figsize=(12,3))

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


title
x
y
0
1
2
3
4
5
0
5
10
15
20
25

Matplotlib

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:

```
fig.savefig(“filename.png")
```

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

```
fig.savefig("filename.png", dpi=200)
```

Matplotlib

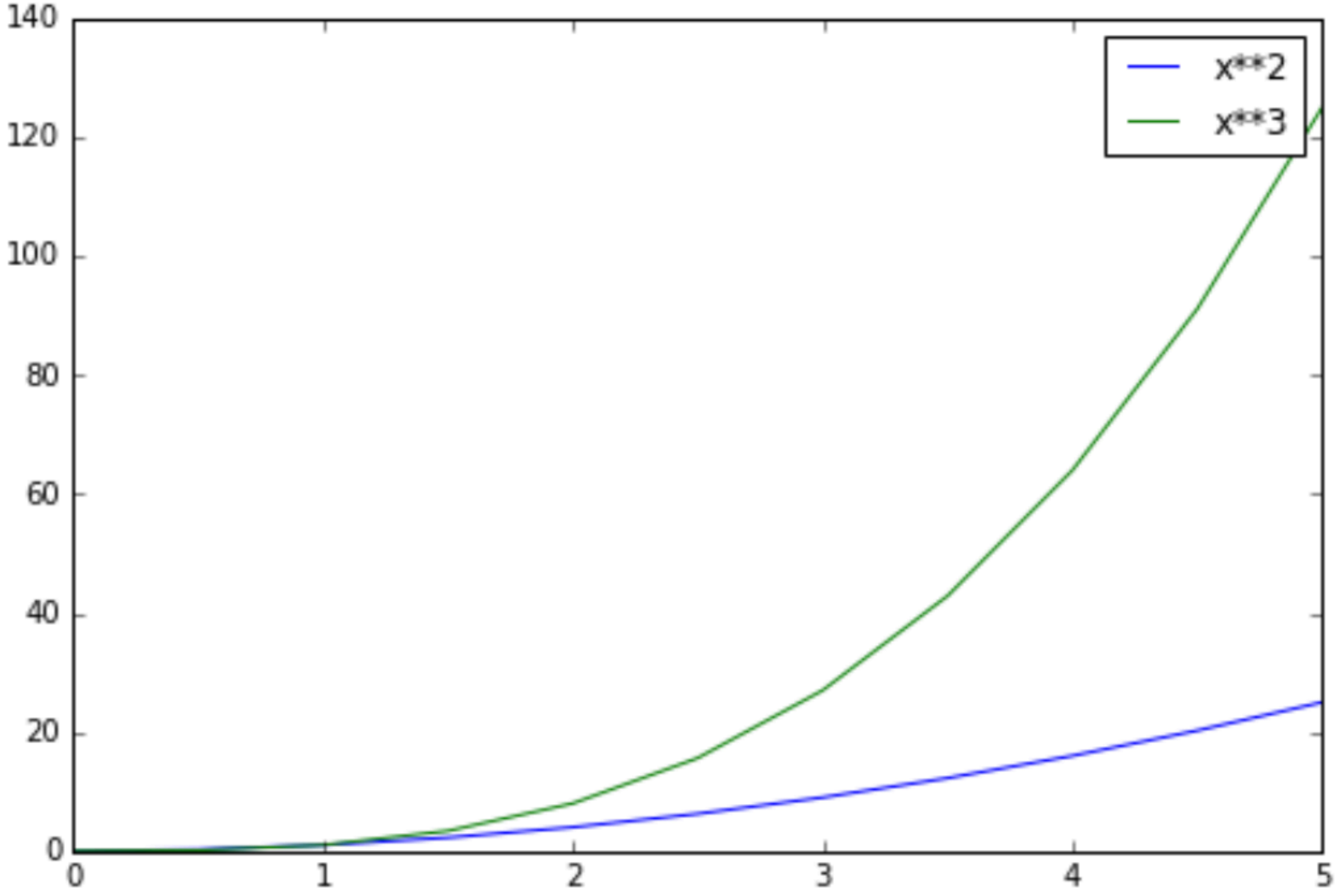
Legends

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:

```
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()
```


140
120
100
80
60
40
20
0
0
1
2
3
4
5
x**2
x**3

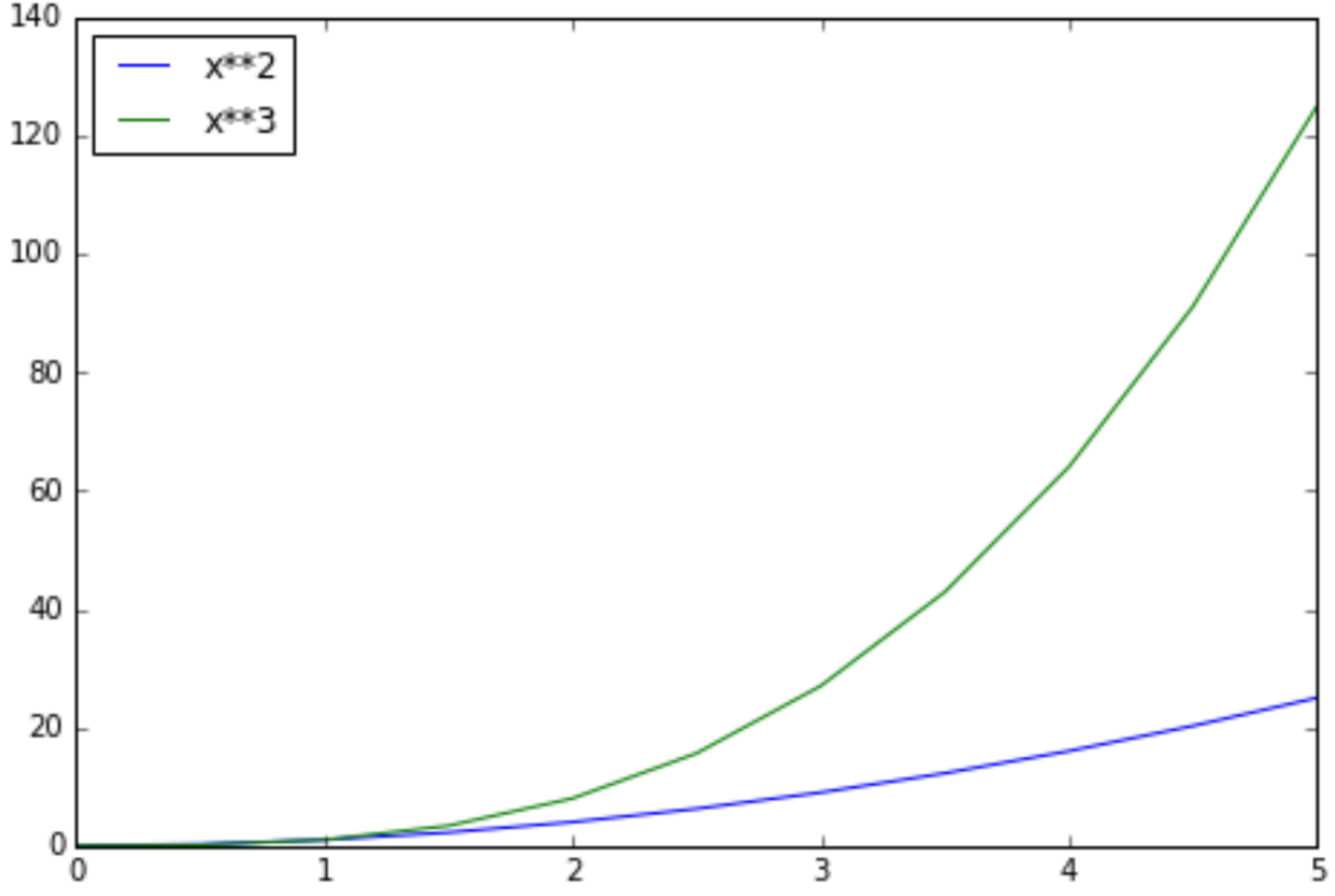
Matplotlib

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.

```
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

# Most common to choose
ax.legend(loc=0)
# let matplotlib decide the optimal location
fig
```


x**2
x**3
0
20
40
60
80
100
120
140
0
1
2
3
4
5

We will see plot styling such as color and size as well as different plots during demo…