EASC2410 Lecture 6

Python Basics: 1-D plots using Matplotlib

Dr. Binzheng Zhang
Department of Earth Sciences

Review of Lecture 5

In Lecture 5, we learned:

- **Basics about NumPy Arrays (1-D)**
- **Creating 1-D NumPy Arrays**
- **Using NumPy Array functions**
- **Generating line plots using Matplotlib.pyplot**

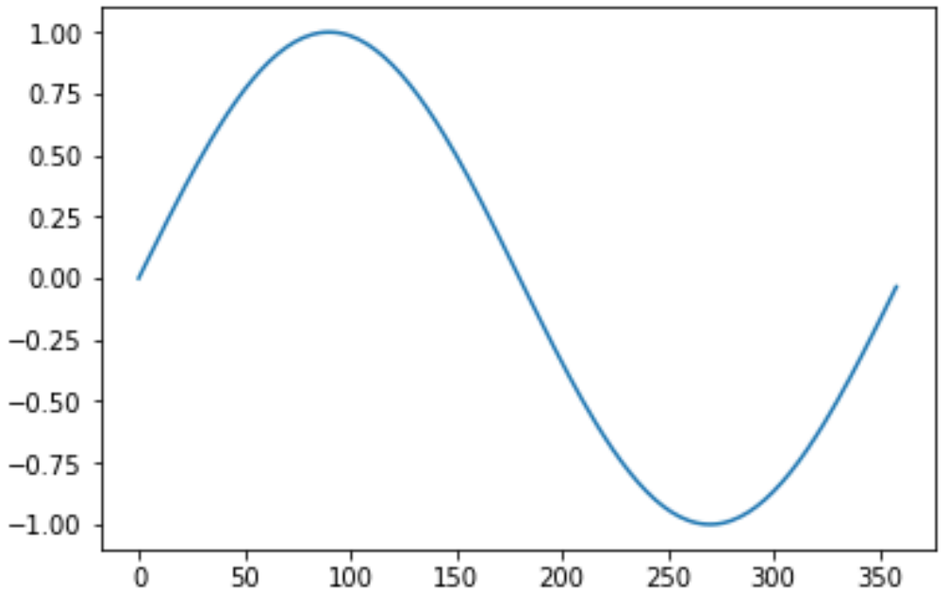
In Lecture 6, you will learn:

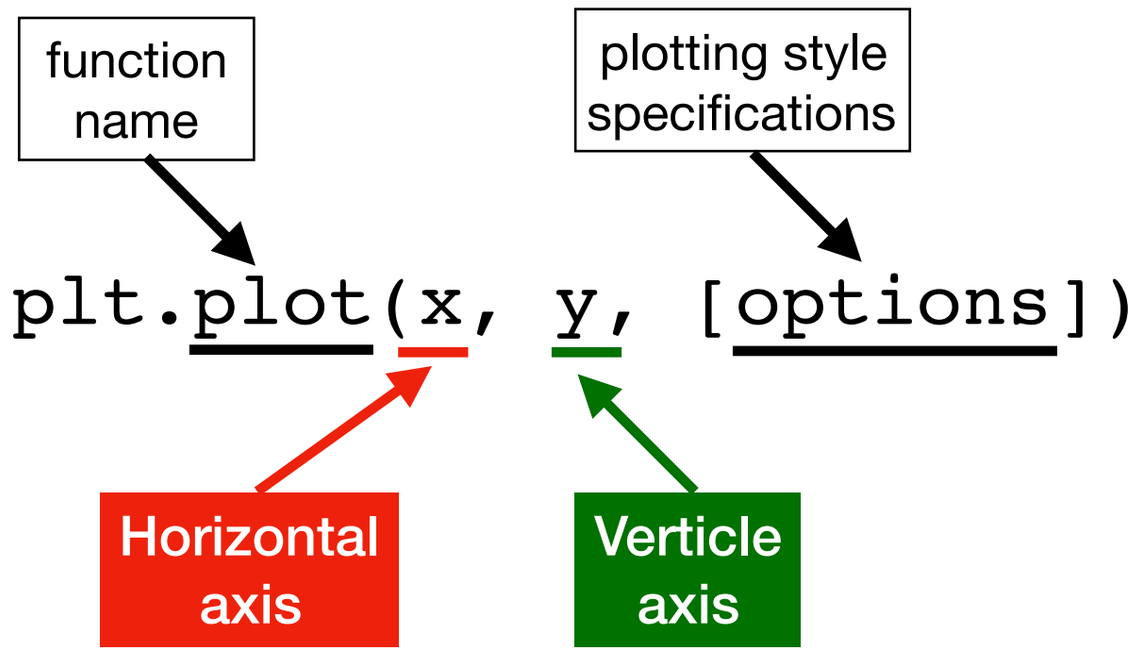
- **Style and Control on line plots**
- **More 1-D plots with Matplotlib**

Recall: Visualize 1-D NumPy Arrays using Matplotlib

The easiest way to do this is using the package **matplotlib** which has many plotting functions, among them a whole module called **pyplot**. We **import** the **matplotlib.pyplot** module as **plt**.

```
In [91]: import matplotlib.pyplot as plt

         plt.plot(thetas_in_degrees,sines); # plot the sines with the angles
         plt.show() # if you don't have this .show() function, Python will not bother showing the plot!
```





Import **pyplot** from **matplotlib**

```
import matplotlib.pyplot as plt # import the pyplot sub-module from the matplotlib module
```

Use the **plot()** function from **plt** to make a line plot

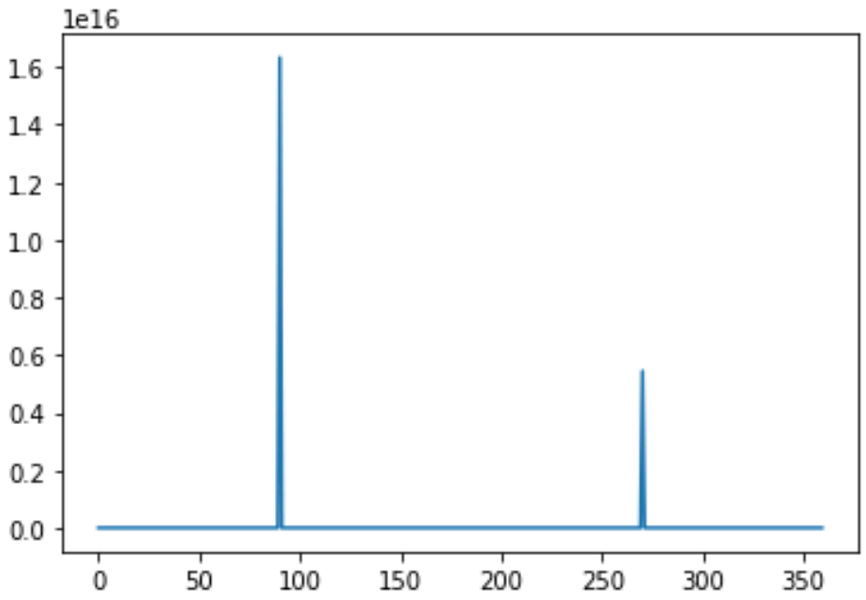
```
plt.plot(thetas_in_degrees,sines); # plot the sines with the angles
```

Show the plot you just made using the **show()** function from **plt**

```
plt.show() # if you don't have this .show() function, Python will not bother showing the plot!
```

Recall: when you tried to make a plot for $\tan\theta$

In [5]:
```
import numpy as np # import the numpy module ad name it as np
import matplotlib.pyplot as plt # import the pyplot sub-module from the matplotlib module

theta = np.arange(0,360,1) # here we used two numpy functions within one line of code: arange() and deg2rad()
plt.plot(theta,np.tan(np.deg2rad(theta)))       # use the plot function from plt to make the 1-D line plot
plt.show()                                  # show results, make sure you type this everything when a plot is done
```



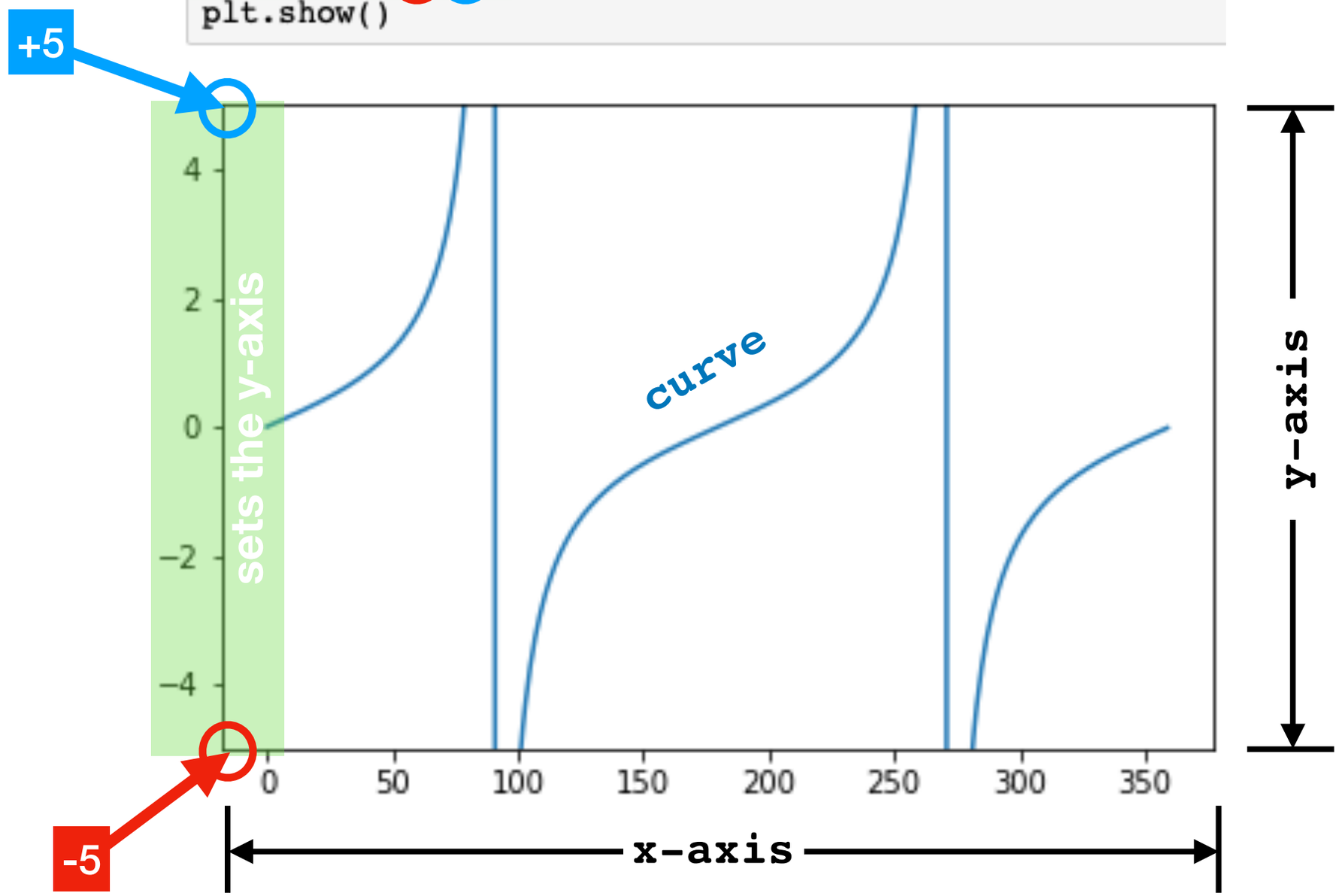
This plot doesn't look like a typical curve of tan( ). What's the problem here?

Because $\tan\frac{\pi}{2} = \infty$, while theta approaches pi/2, tan(theta) becomes a huge number (1e16) here, so the plot was dominated by that "spike" due to the limits of the y axis.

How to fix it? Specify the limits of the y-axis using the ylim() function

remember that plt.plot() is the function gives you the curve, the plt.ylim( ) function help you set up the maximum and minimum values for your vertical axis (y-axis), so you can get a more reasonable curve of tan(theta)

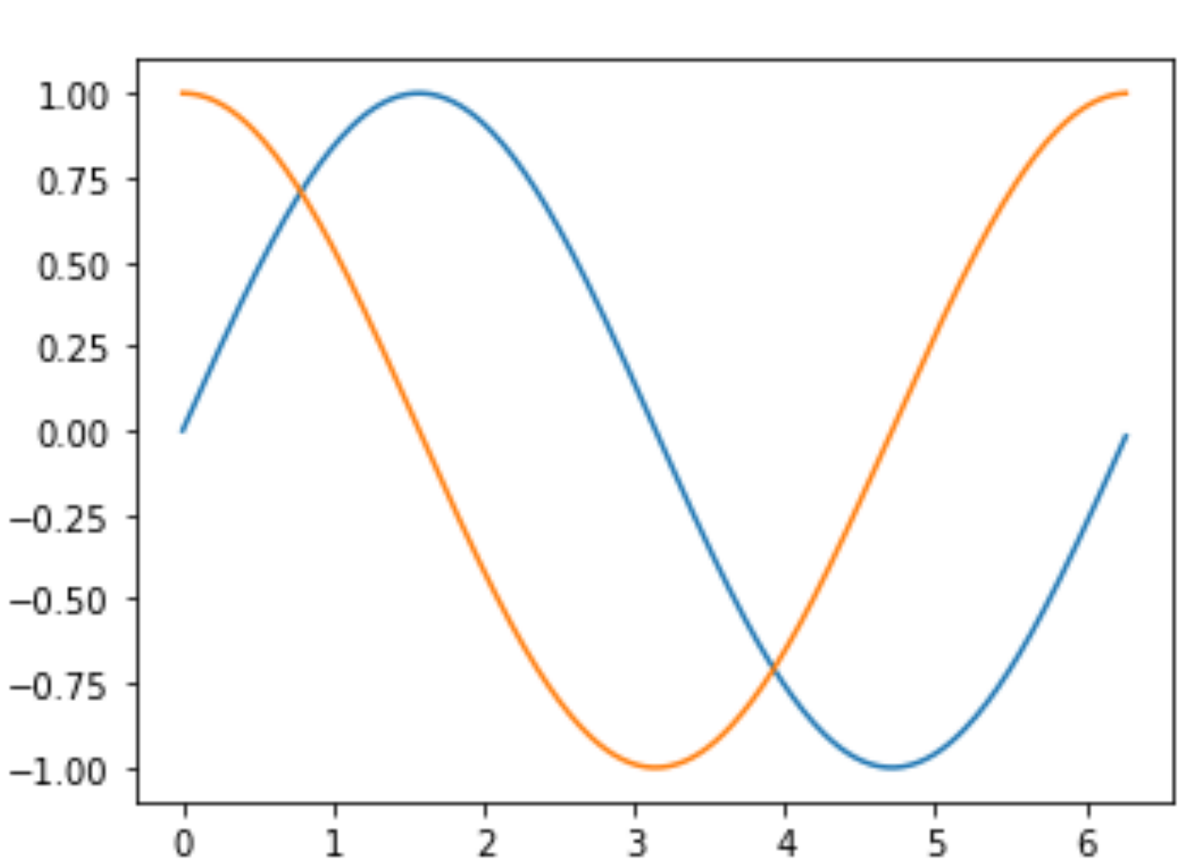
```
In [114]: plt.plot(theta,np.tan(np.deg2rad(theta)))

          plt.ylim([-5,5]) # set the y limits from -5 to 5
          plt.show()
```

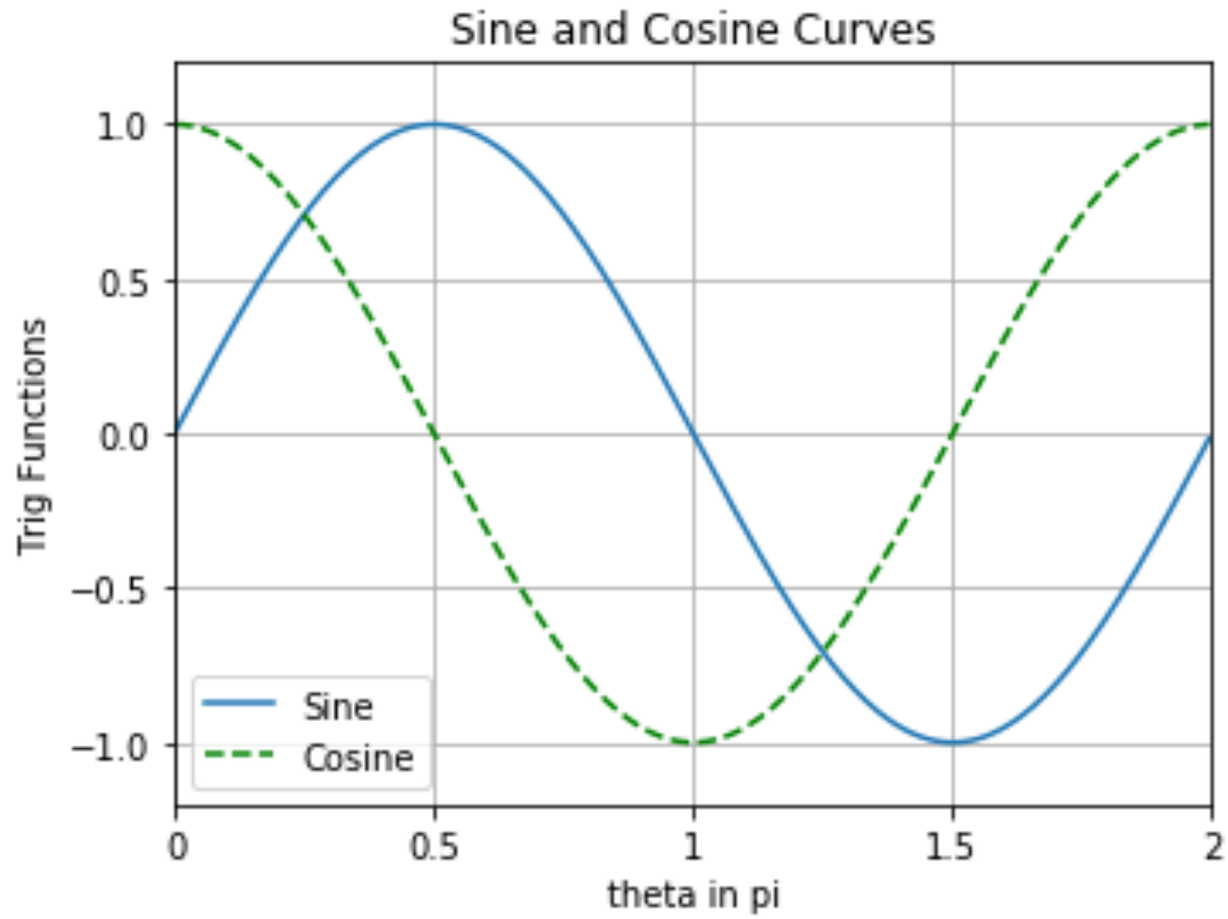



```
plt.ylim([y_min,y_max])
```

Figure Styling in Matplotlib

Let’s take a look at two plots of trigonometry functions to start with:





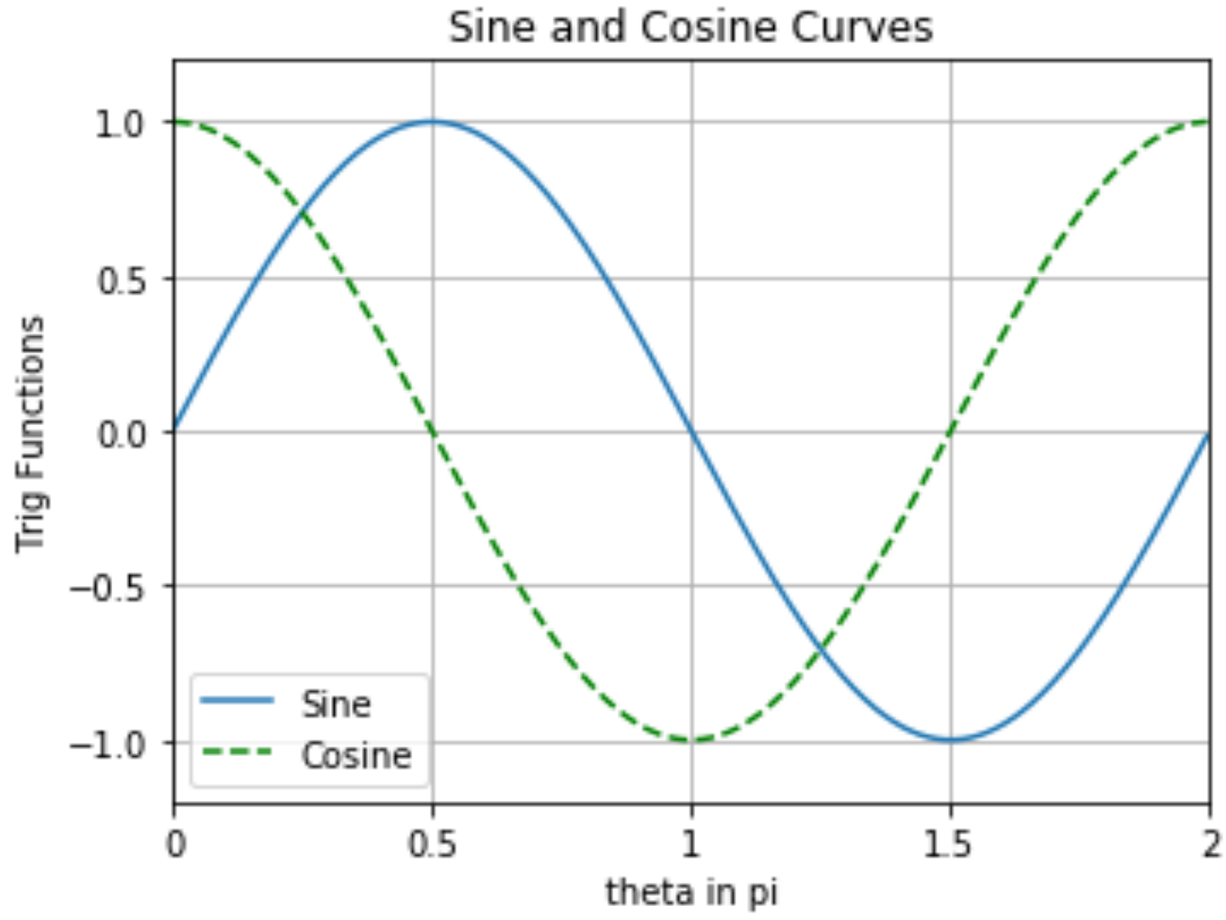
Which one is a better example of visualisation?

Figure Styling in Matplotlib

Of course everyone likes the right one!

Tips on figure styling

- select reasonable ranges for your axis
- label your axis
- give your figure a title
- used colours for different datasets
- put on a legend
- pay attention to the font size
- fine tuning of the ticks



```
plt.plot(theta, np.sin(theta), color = 'b', linestyle='-', label='Sine')
```

plot the sine curve

```
plt.plot(theta, np.cos(theta), linestyle='--', color='g', label='Cosine')
```

plot the cosine curve

```
plt.xlim([0,theta.max()])
plt.ylim([-1.2,1.2])
```

set the limits for x and y-axis

```
plt.xlabel('theta in pi')
plt.ylabel('Trig Functions')
```

label the axis

```
plt.title('Sine and Cosine Curves')
plt.legend()
```

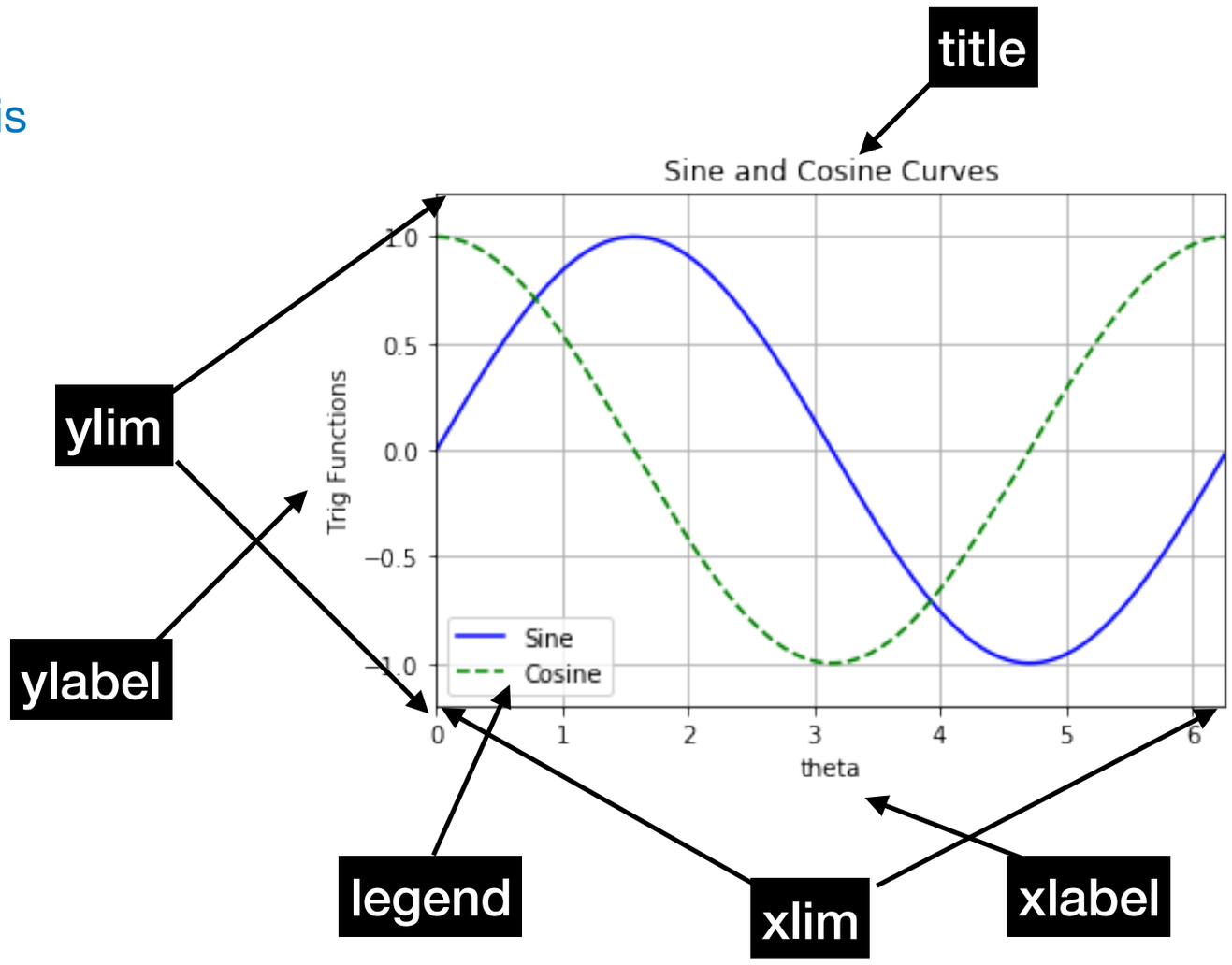
add title and legend

```
plt.grid()
```

Show the grid in both x and y axis

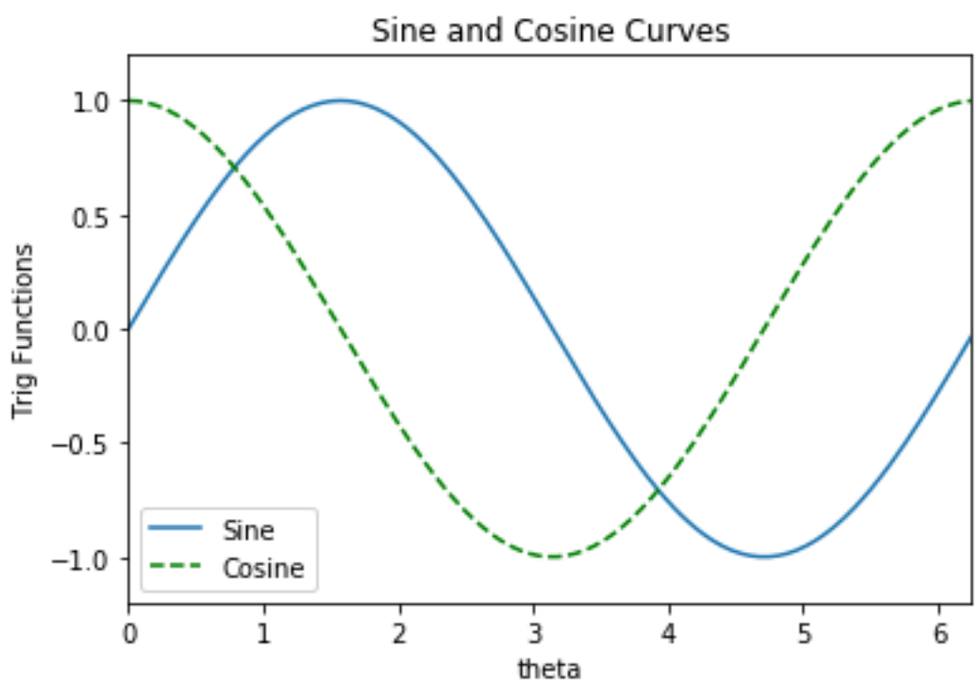
Tips on figure styling

- select reasonable ranges for your axis
- label your axis
- give your figure a title
- used colours for different datasets
- put on a legend
- pay attention to the font size
- fine tuning of the ticks



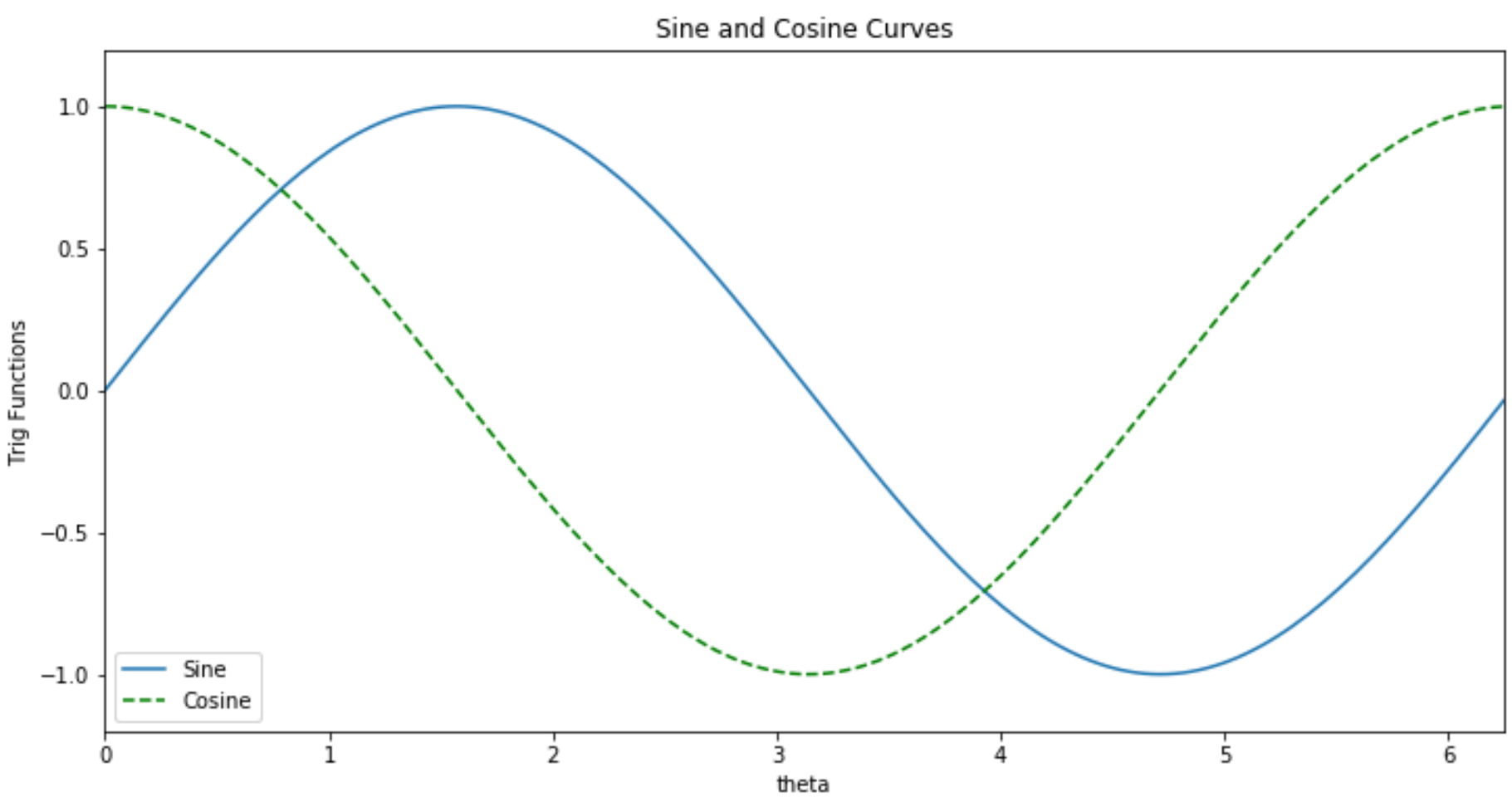
Put it all together

In [2]:
```
import numpy as np # import the numpy module
import matplotlib.pyplot as plt # import the pyplot sub-module from matplotlib

theta = np.deg2rad( np.arange(0,360,2) ) # generate a numpy array for theta

plt.plot(theta, np.sin(theta), linestyle='-', label='Sine') # plot sin(theta)
plt.plot(theta, np.cos(theta), linestyle='--', color='g', label='Cosine') # plot cos(theta)
plt.xlim([0,theta.max()]) # set the limits for x axis
plt.ylim([-1.2,1.2])      # set the limits for y axis
plt.xlabel('theta')       # label the x axis
plt.ylabel('Trig Functions')  # label the y axis
plt.title('Sine and Cosine Curves')  # title for the figure
plt.legend()  # show legend

plt.show() # show plot
```



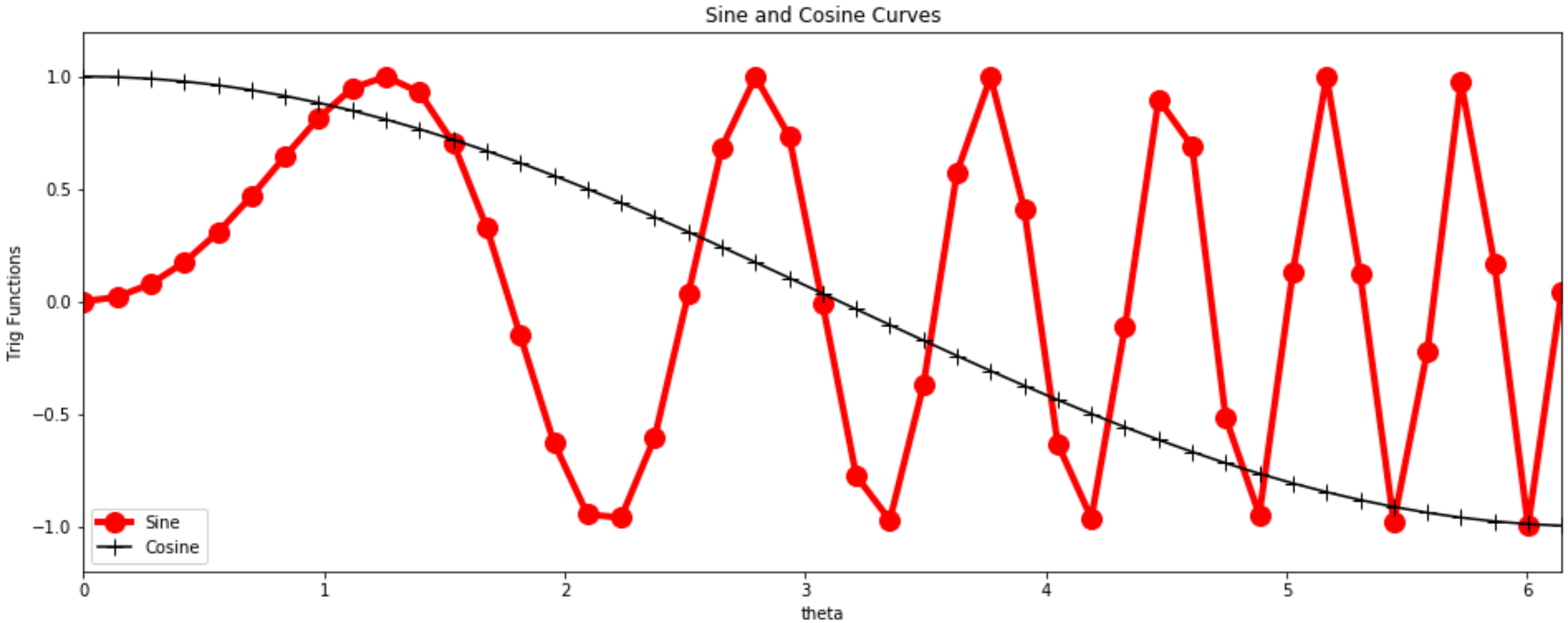
You can also change the size of your figure

In [4]:
```
import numpy as np # import the numpy module
import matplotlib.pyplot as plt # import the pyplot sub-module from matplotlib

theta = np.deg2rad( np.arange(0,360,2) ) # generate a numpy array for theta

plt.figure(figsize=(12,6))

plt.plot(theta, np.sin(theta), linestyle='-', label='Sine') # plot sin(theta)
plt.plot(theta, np.cos(theta), linestyle='--', color='g', label='Cosine') # plot cos(theta)
plt.xlim([0,theta.max()]) # set the limits for x axis
plt.ylim([-1.2,1.2])      # set the limits for y axis
plt.xlabel('theta')       # label the x axis
plt.ylabel('Trig Functions')  # label the y axis
plt.title('Sine and Cosine Curves')  # title for the figure
plt.legend()  # show legend

plt.show() # show plot
```



You can also use markers for your data points

In [12]:
```
plt.figure(1,figsize=(16,6))

plt.plot(theta, np.sin(theta**2), color='r', linestyle='-', marker='o', label='Sine', linewidth = 4, markersize = 12)
plt.plot(theta, np.cos(theta/2), 'k-+', label='Cosine', markersize = 10)
plt.xlim([0,theta.max()])
plt.ylim([-1.2,1.2])
plt.xlabel('theta')
plt.ylabel('Trig Functions')
plt.title('Sine and Cosine Curves')
plt.legend()
plt.show()
```



So here are some useful properties we used in the plt.plot() function to control your plots

Property	Values
linewidth	float value in points
linestyle	['-' | '--' | '-.' | ':' | 'steps' | ...]
marker	['+' | ',' | '.' | 'o' | 'd' | '*' | '1']
color	['b', 'g', 'r', 'k', 'y', 'm', 'c'] or any matplotlib colorm
markersize	float
label	any string

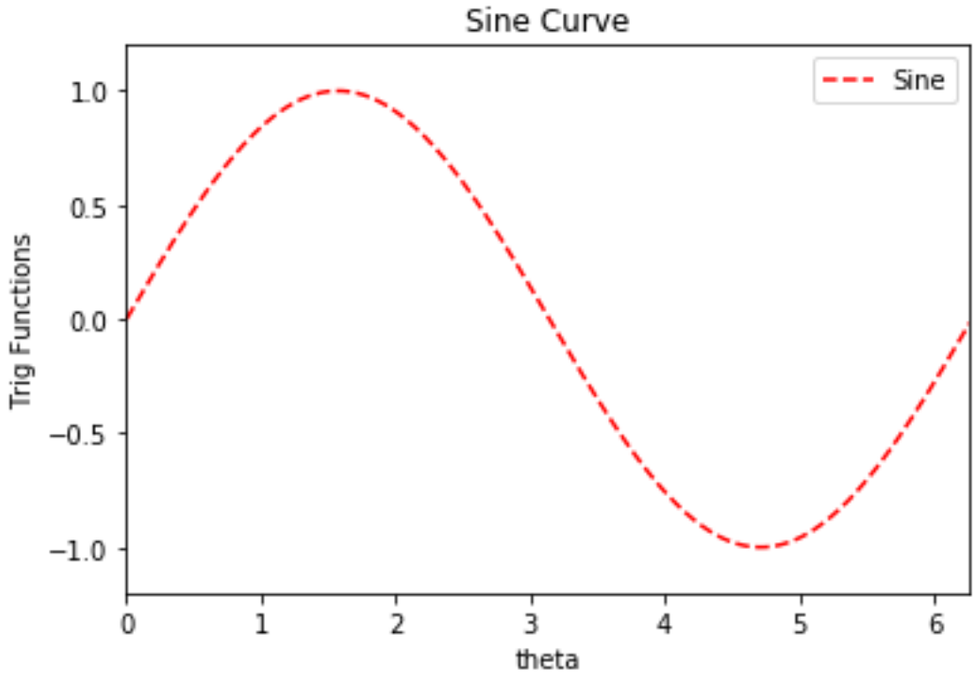
here are some useful functions in plt to control your plots:

Function	Purpose	Examples
figure([num, figsize, dpi, facecolor, ...])	make a new figure	plt.figure(1, figsize = (8,4)
grid([b, which, axis])	Configure the grid lines	plt.grid()
show(*args*, *kw)	show the plot	plt.show()
subplot(*args*, *kwargs)	Add a subplot to the current figure	plt.subplot(2,2,1)
text(x, y, s[, fontdict, withdash])	Add text to the figure	plt.text(1,2,'my curve')
title(label[, fontdict, loc, pad])	Set a title for the axes	plt.title('My plot')
xlabel(xlabel[, fontdict, labelpad])	Set the label for the x-axis	plt.xlabel('x axis')
xlim(*args*, *kwargs)	Get or set the x limits of the current axes	plt.xlim([-1, 1])
xscale(*args)	set the x sclae	plt.scale('linear')
xticks([ticks, labels])	Get or set the current tick locations and labels of the x-axis	xticks(np.arange(5), ('Tom', 'Dick', 'Harry', 'Sally', 'Sue'))

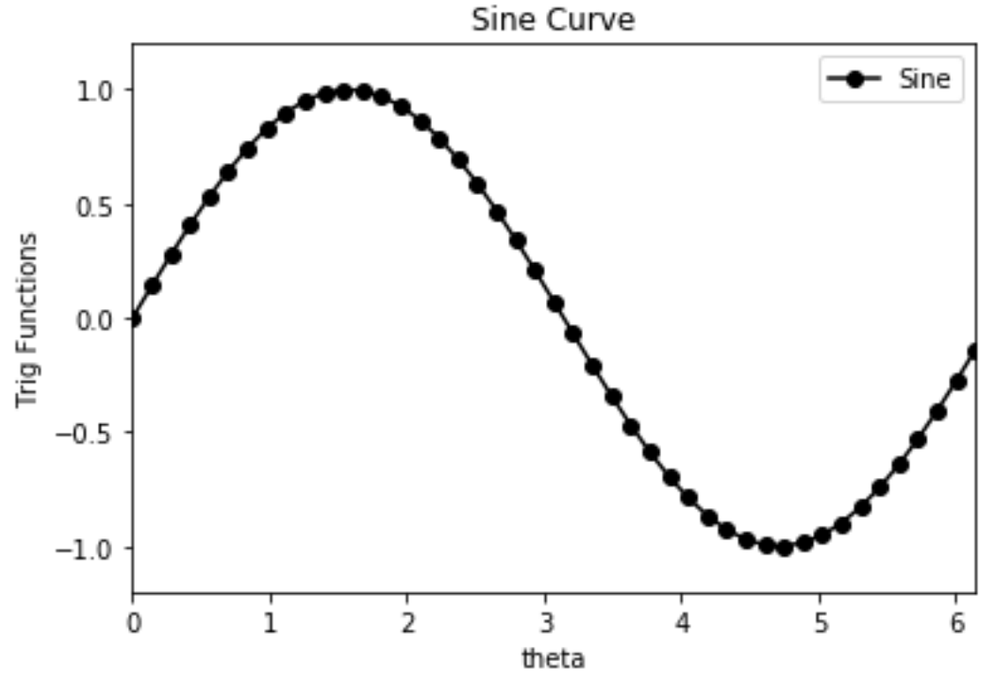
More information about controlling line properties can be found here: https://matplotlib.org/tutorials/introductory/pyplot.html#controlling-line-properties

A couple of examples

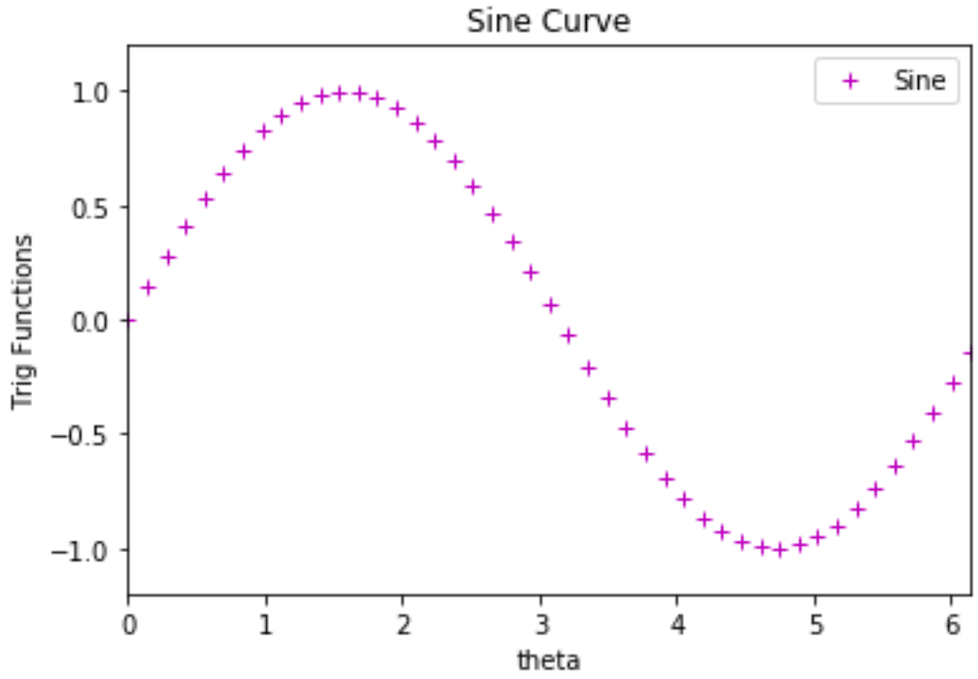
```
plt.plot(theta, np.sin(theta), color = 'r', linestyle='--', label='Sine')
```



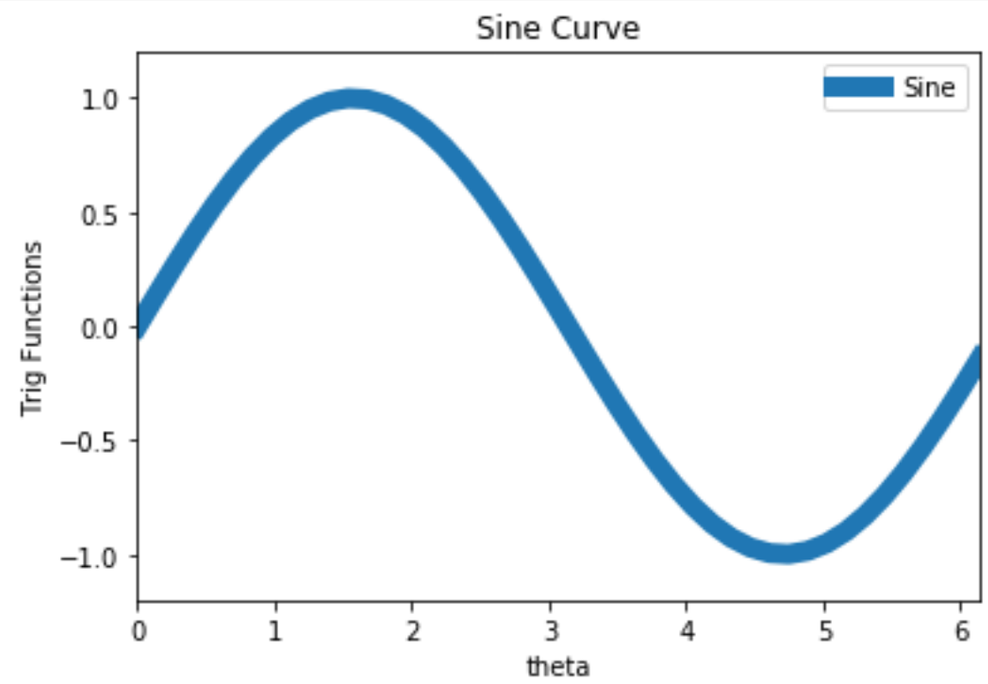

```
plt.plot(theta, np.sin(theta), color = 'k', linestyle='-', marker = 'o',label=
```




```
plt.plot(theta, np.sin(theta), color = 'm', linestyle='none', marker = '+', label='Sine')
```



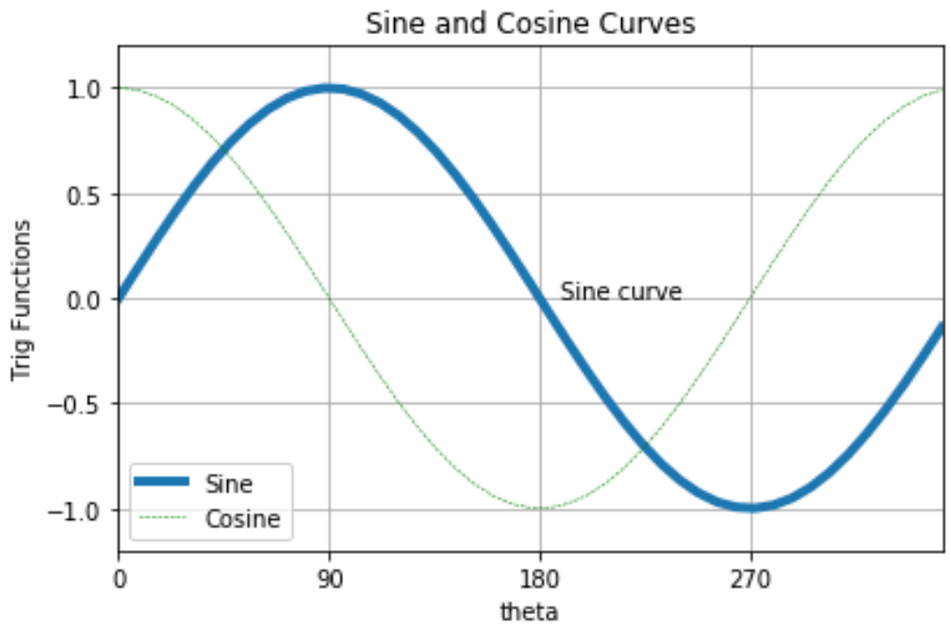

```
plt.plot(theta, np.sin(theta), '-', linewidth = 8, label='Sine')
```

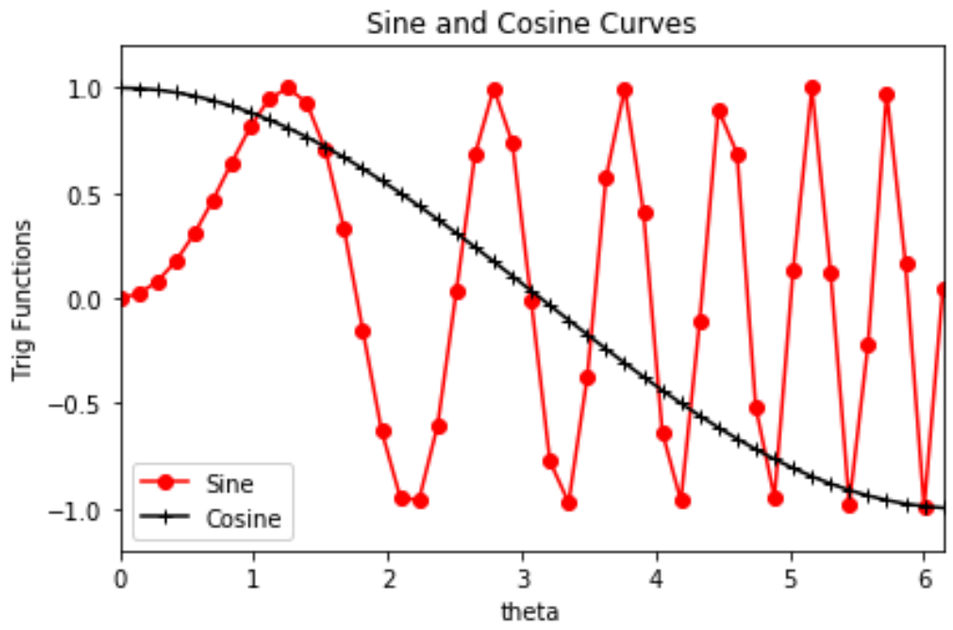



```
plt.plot(theta, np.sin(theta), 'm+', label='Sine')
```

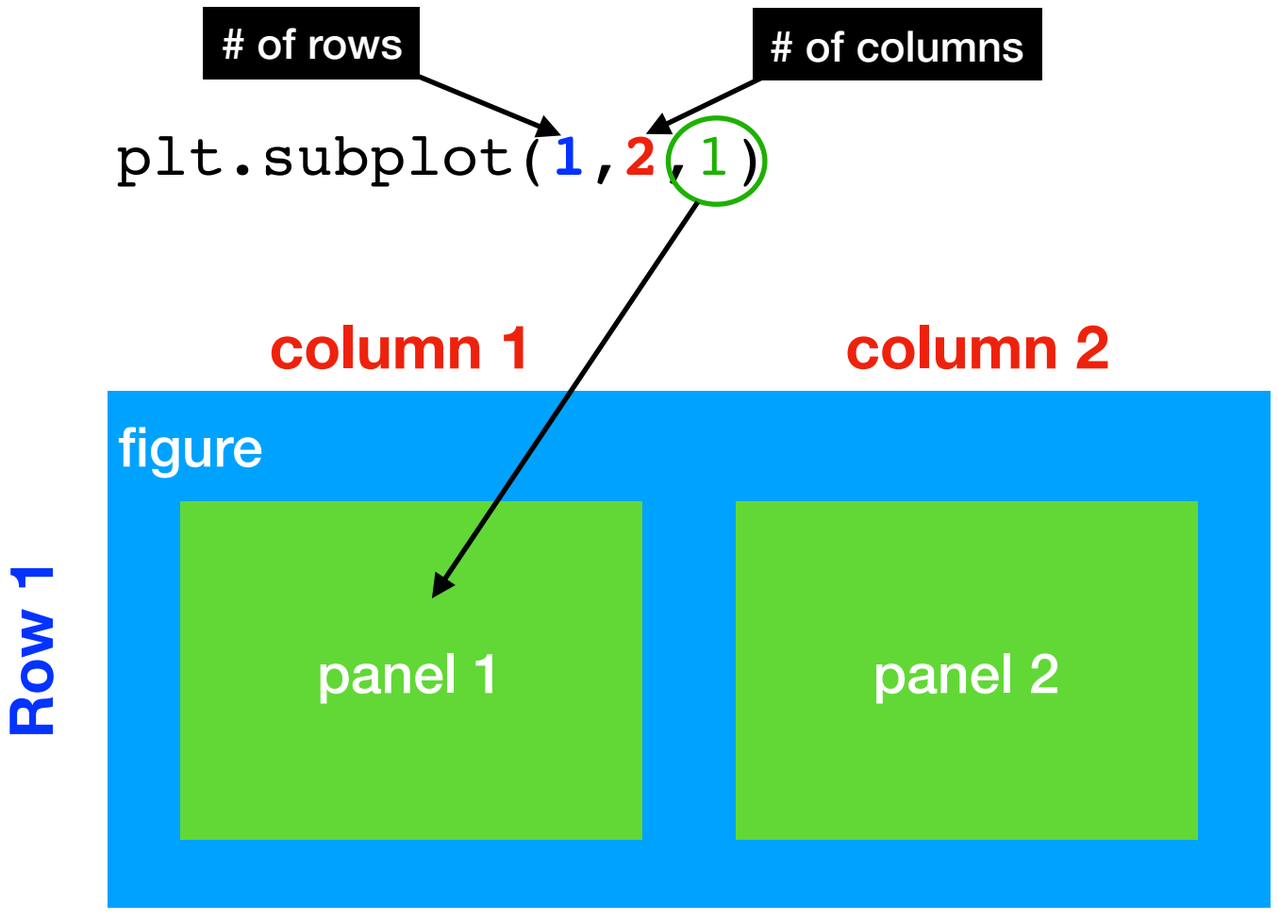
Putting more panels in one plot: `subplot( )`

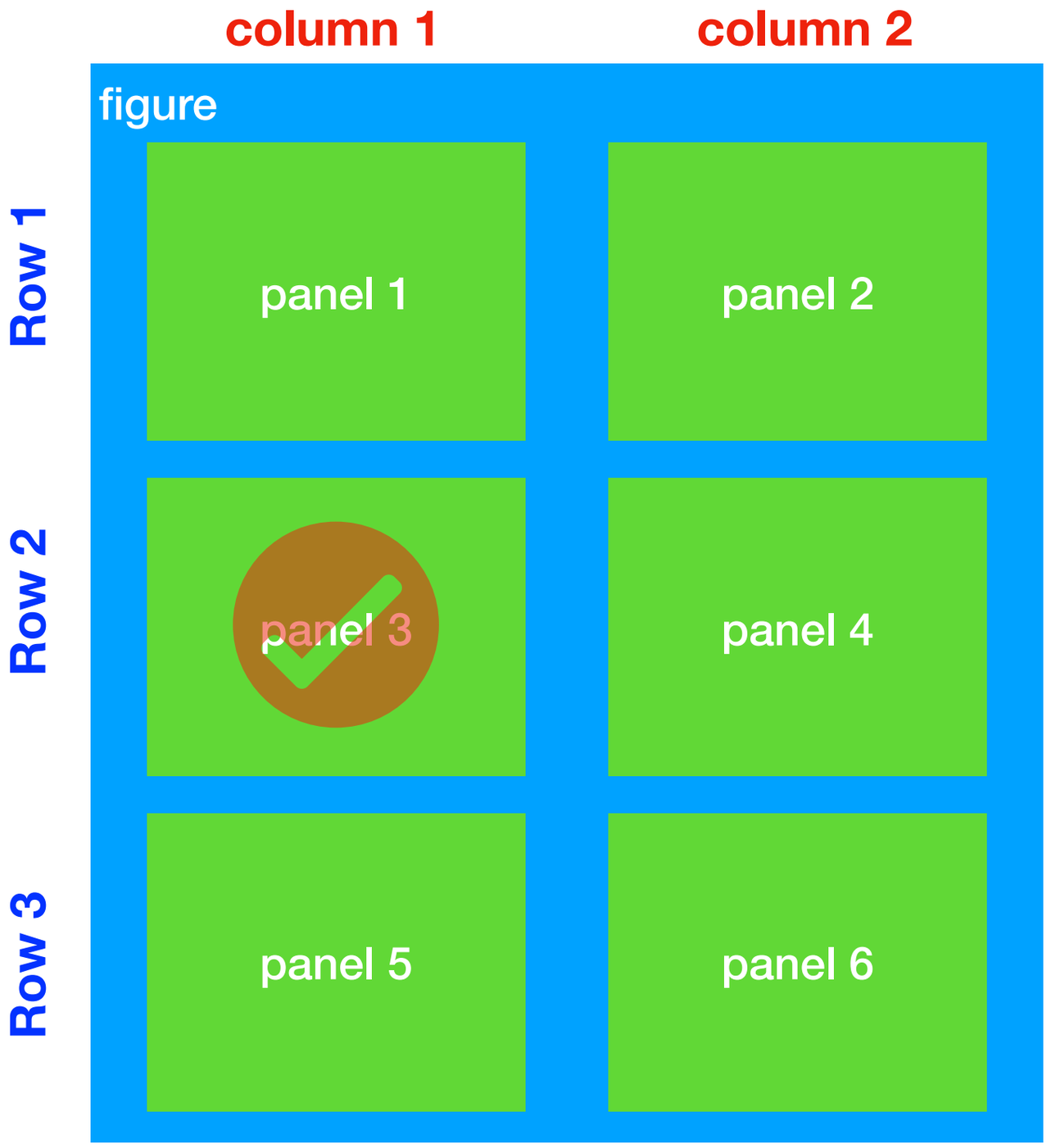
In [14]:
```
plt.figure(figsize=(14, 4)) # start a figure with size 14 (horizontal) and 4 (verticle)

plt.subplot(1, 2, 1) # the first panel
plt.plot(theta, np.sin(theta), '-', label='Sine', linewidth = 4) # plot sine and cosine
plt.plot(theta, np.cos(theta), 'g--', label='Cosine', linewidth = 0.5)
plt.xlim([0,theta.max()]) # set the limit for the x axis
plt.ylim([-1.2,1.2]) # set the limit for the y axis
plt.xlabel('theta') # label the x-axis
plt.ylabel('Trig Functions') # label the y axis
plt.xticks(np.arange(0,np.pi*2,np.pi/2),('0','90','180','270')) # set the ticks of the x axis
plt.text(3.3,0,'Sine curve') # annotate text at x=3.3, y=0.0
plt.title('Sine and Cosine Curves') # set a title
plt.legend() # show legend
plt.grid() # show grid

plt.subplot(1, 2, 2) # the first panel second
plt.plot(theta, np.sin(theta**2), 'r-o', label='Sine') # make another two plots
plt.plot(theta, np.cos(theta/2), 'k-+', label='Cosine')
plt.xlim([0,theta.max()])  # set the limit for the x axis
plt.ylim([-1.2,1.2])  # set the limit for the y axis
plt.xlabel('theta') # label the x-axis
plt.ylabel('Trig Functions') # label the y axis
plt.title('Sine and Cosine Curves') # set a title
plt.legend() # show legend

plt.show() # show the plots
```





How do the subplot indices work?





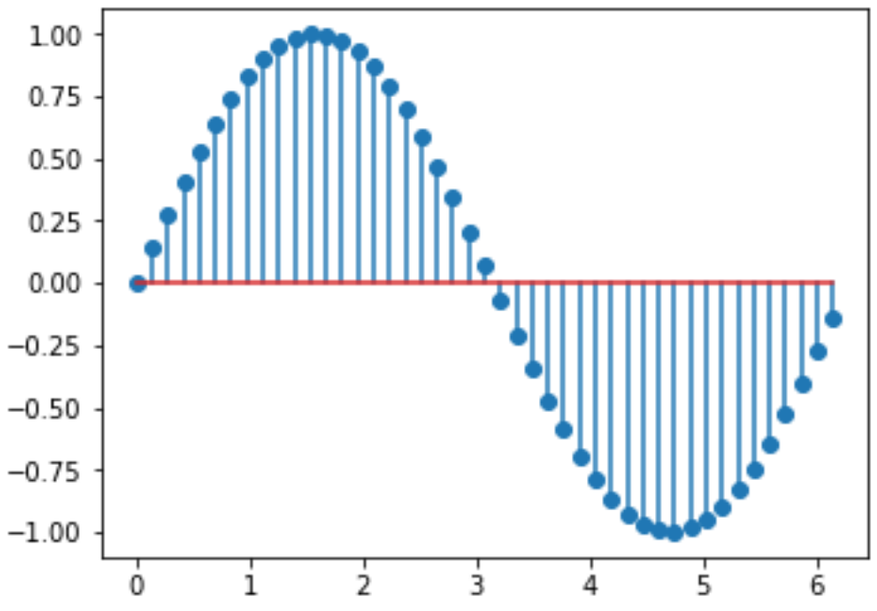
What's the syntax for subplots?

```
plt.subplot(rows,columns,panel #1)
      [make your plot here]

plt.subplot(rows,columns,panel #2)
      [make your plot here]

                  ……

plt.show()
```

More choices on 1-D plots: stem, step and fill plots

a stem plot:

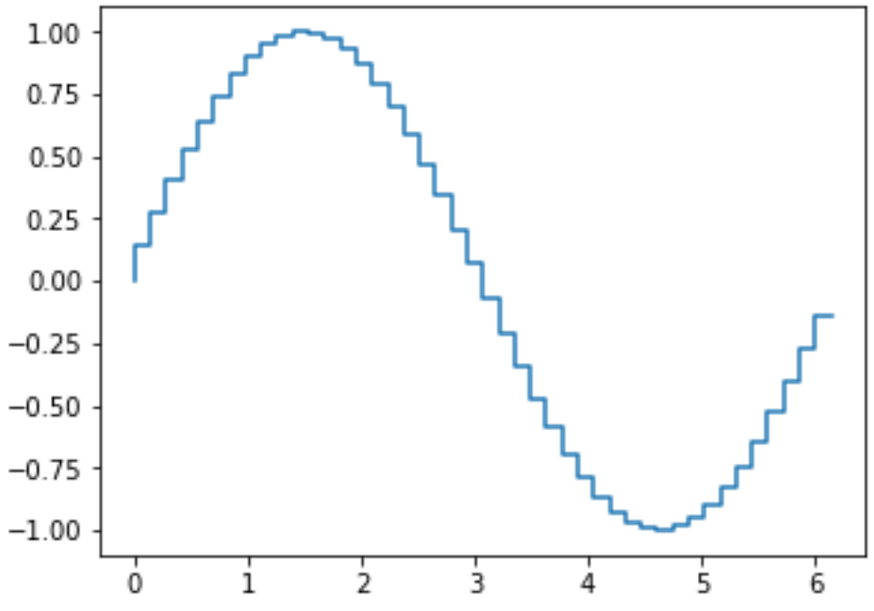
syntax: `plt.stem(theta,np.sin(theta))`



plt.stem([x],y,[options])

a step plot:

syntax: `plt.step(theta,np.sin(theta))`

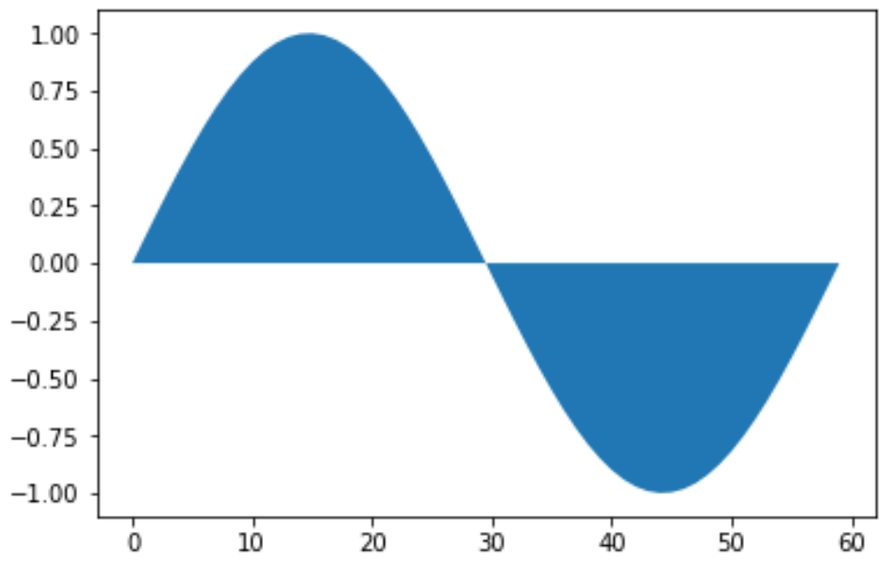


plt.step([x],y,[options])

a filled plot:

syntax: `plt.fill(theta,np.sin(theta))`

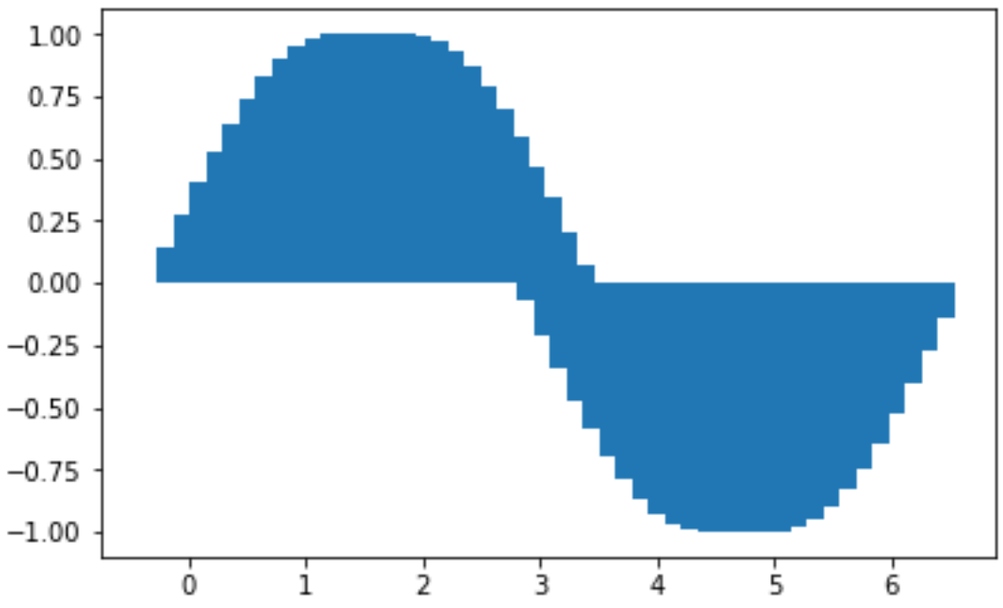
plt.fill([x],y,[options])



More choices on 1-D plots: bar plots

a bar plot:

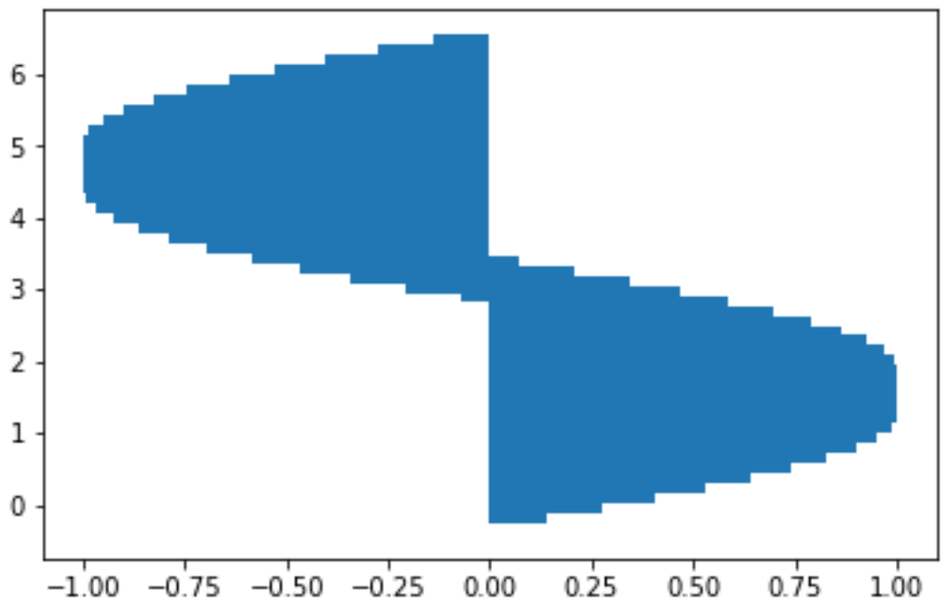
syntax: `plt.bar(theta,np.sin(theta))`



plt.bar([x],y,[options])

a horizontal bar plot:

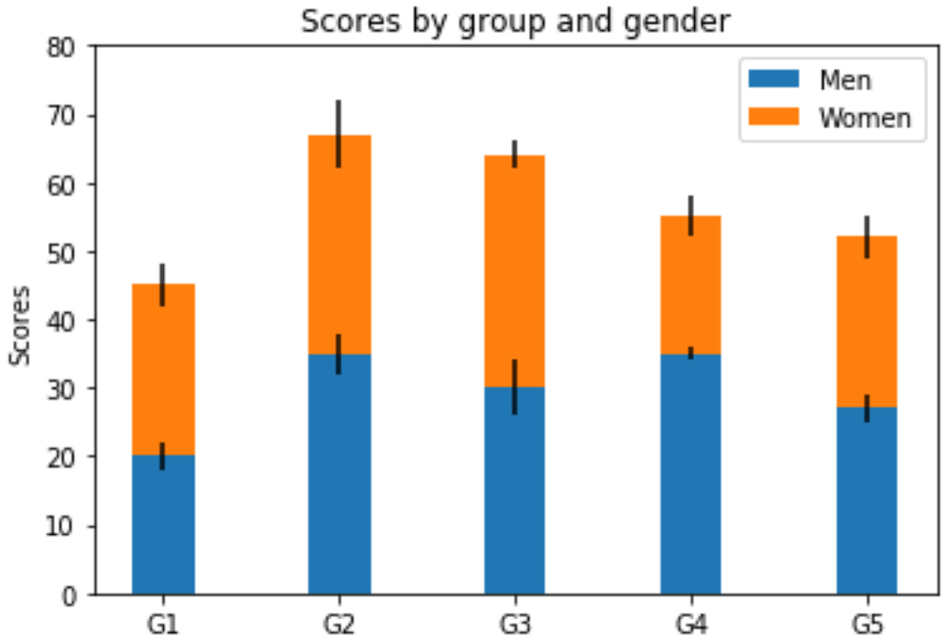
syntax: `plt.barh(theta,np.sin(theta))`



plt.hbar([x],y,[options])

More choices on 1-D plots: bar plots

Bar plots can be much more information-rich, see this example:



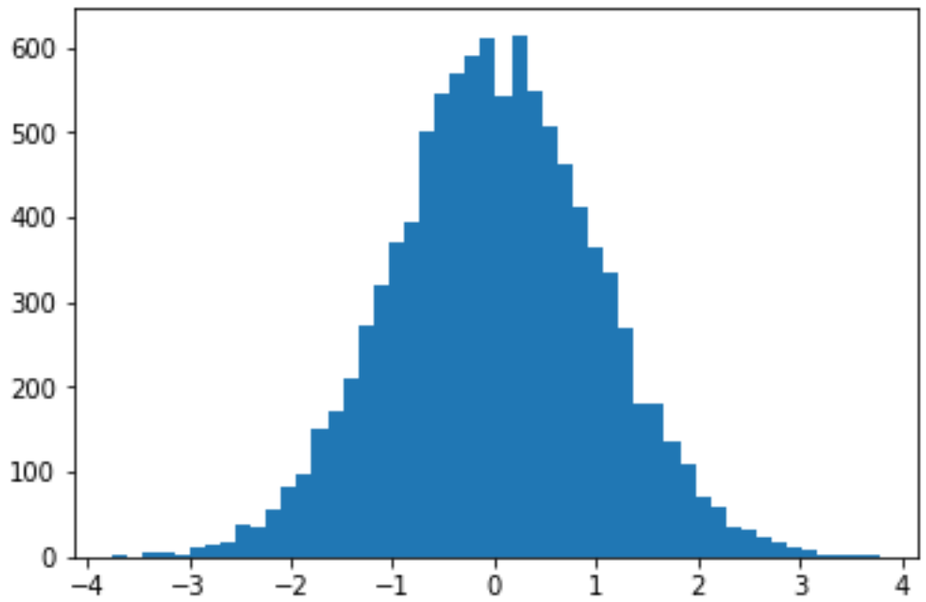
- This bar plot shows two data sets
- Different colors are used to the bars
- Error-bar of the data is also shown

In [15]:
```
import numpy as np
import matplotlib.pyplot as plt


N = 5
menMeans = (20, 35, 30, 35, 27)
womenMeans = (25, 32, 34, 20, 25)
menStd = (2, 3, 4, 1, 2)
womenStd = (3, 5, 2, 3, 3)
ind = np.arange(N)    # the x locations for the groups
width = 0.35       # the width of the bars: can also be len(x) sequence

p1 = plt.bar(ind, menMeans, width, yerr=menStd)
p2 = plt.bar(ind, womenMeans, width,
             bottom=menMeans, yerr=womenStd)

plt.ylabel('Scores')
plt.title('Scores by group and gender')
plt.xticks(ind, ('G1', 'G2', 'G3', 'G4', 'G5'))
plt.yticks(np.arange(0, 81, 10))
plt.legend((p1[0], p2[0]), ('Men', 'Women'))

plt.show()
```

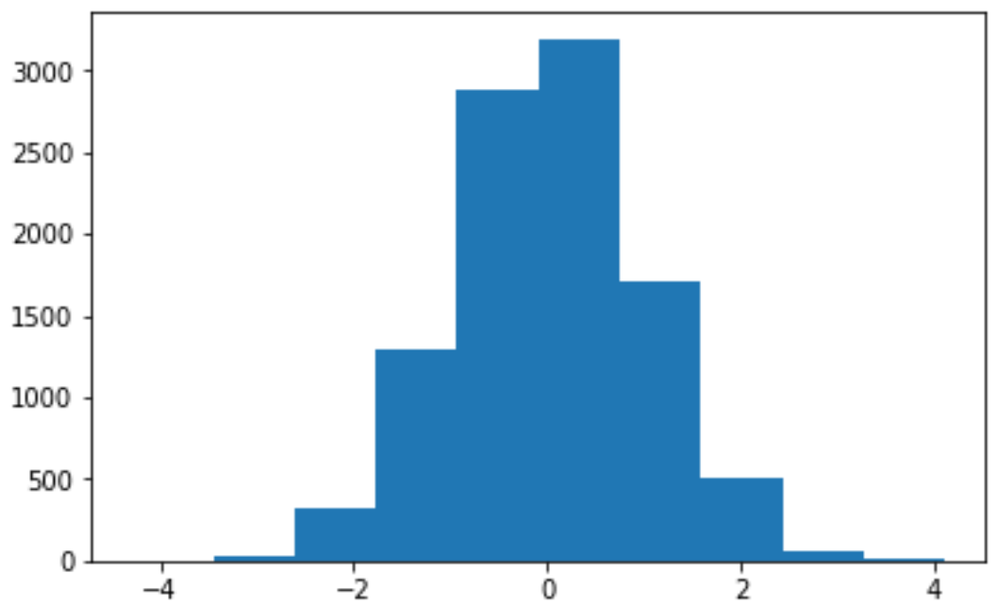
More choices on 1-D plots: histgrams

a histgram: **plt.hist(x,[options])**

syntax: `plt.hist(x, bins=50)`



`plt.hist(x, bins=10)`



- **A histogram is an accurate representation of the distribution of numerical data. It is an estimate of the probability distribution of a continuous variable and was first introduced by Karl Pearson.**
- **It differs from a bar graph, in the sense that a bar graph relates two variables, but a histogram relates only one**

Definition from Wikipedia

What is a histgram?

StatQuest: Histograms, clearly explained!!!

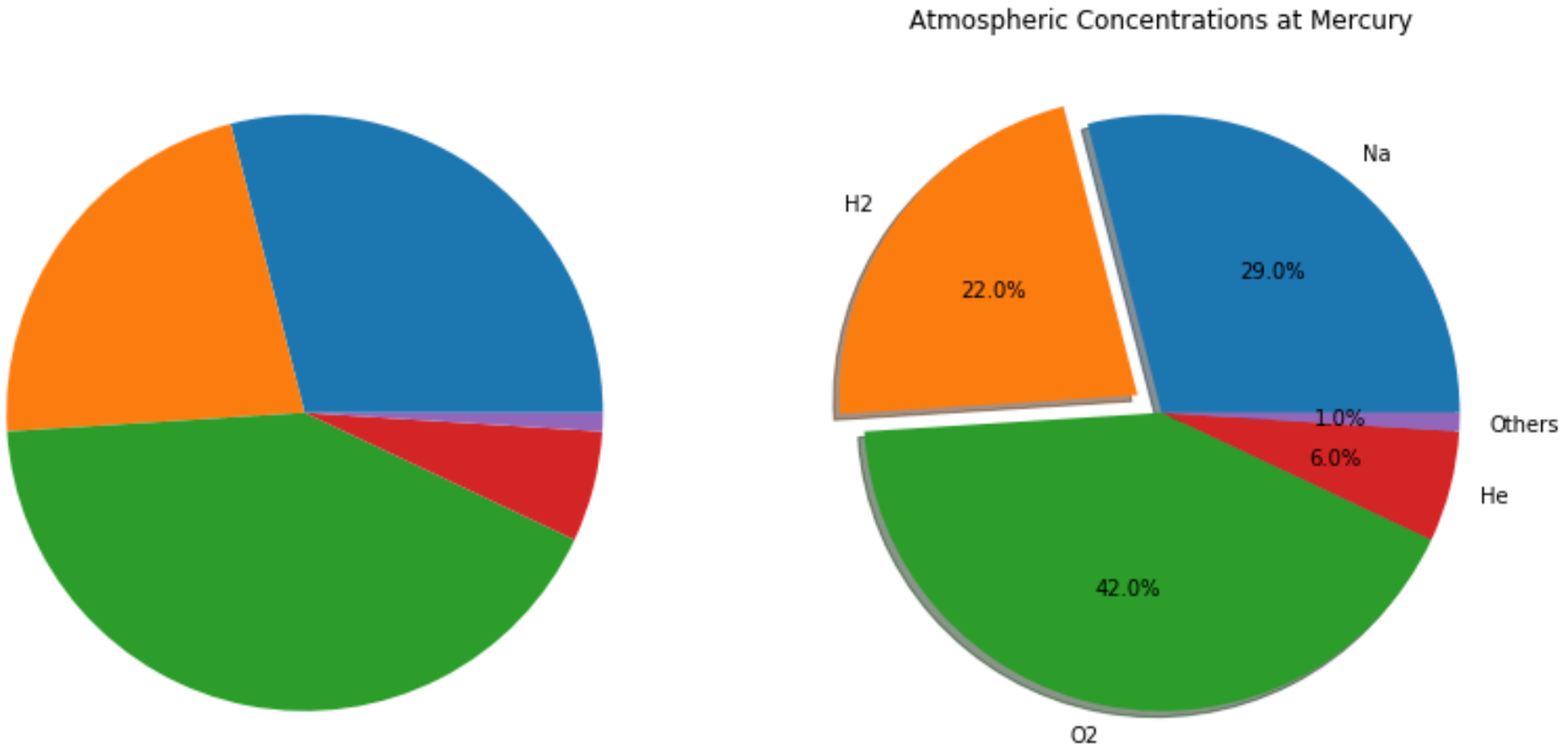
source: https://youtu.be/qBigTkBLU6g

More choices on 1-D plots: pie plots

a pie plot: **plt.pie(x,[options])**

```
# Pie chart, where the slices will be ordered and plotted counter-clockwise:
labels  = ['Na', 'H2', 'O2', 'He','Others']
sizes   = [0.29, 0.22, 0.42, 0.06, 0.01]
explode = (0, 0.1, 0, 0, 0)  # only "explode" the 2nd slice (i.e. 'Hogs')

plt.figure(3, figsize=(14, 7)) # generate a figure

plt.subplot(1, 2, 1) # the left panel
plt.pie(sizes) # generate a simple pie plot
```

```
plt.subplot(1, 2, 2) # the right panel
plt.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%', # generate a better pie plot with more options
        shadow=True, startangle=0)
plt.title('Atmospheric Concentrations at Mercury') # adding a title
```



More choices on 1-D plots: scatter (bubble) plots

a scatter plot: **plt.scatter(x,y,[options])**

```
planets = ['Mercury','Venus','Earth','Mars','Jupiter','Saturn','Uranus','Neptune']
colors = ['b','r','g','k','y','m','c','b']
distance = np.array([0.39,  0.72,  1.  ,  1.52,  5.2 ,  9.58, 19.2 , 30.05]) # normalized to Sun-Earth distance
radius = np.array([0.38,  0.95,  1.  ,  0.53, 11.21,  9.45,  4.01,  3.88]) # normalized to Earth radius
num = np.arange(0,8,1); # index for the planets, from
```

List

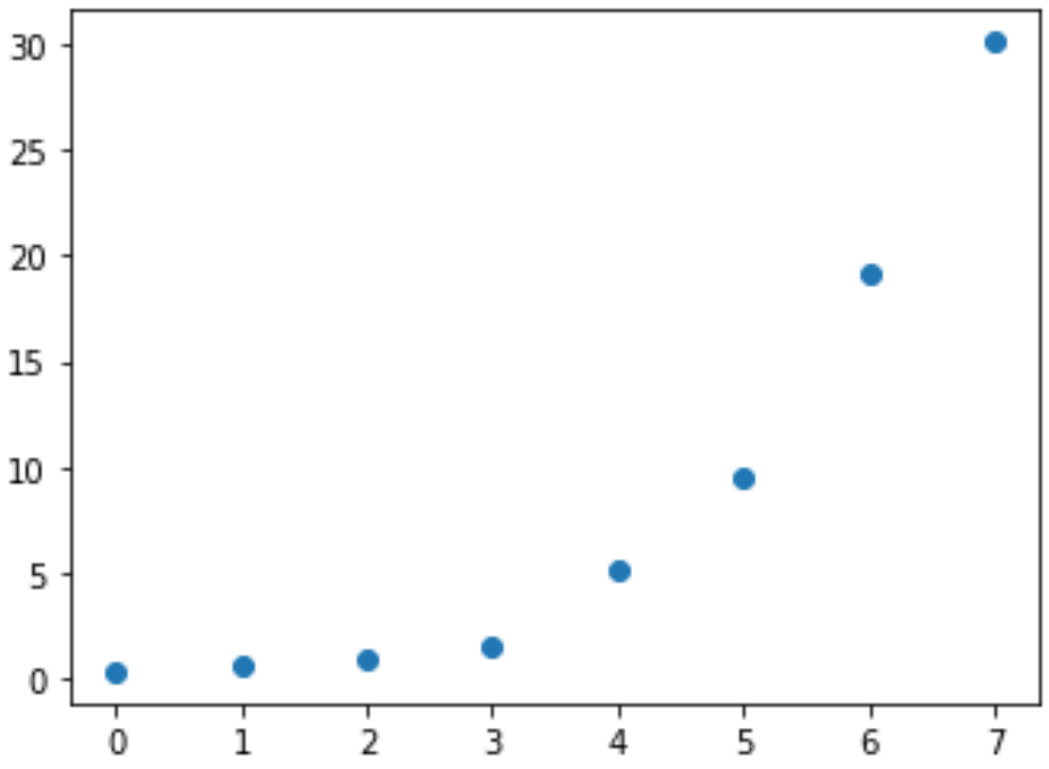
List

Horizontal axis

Vertical axis

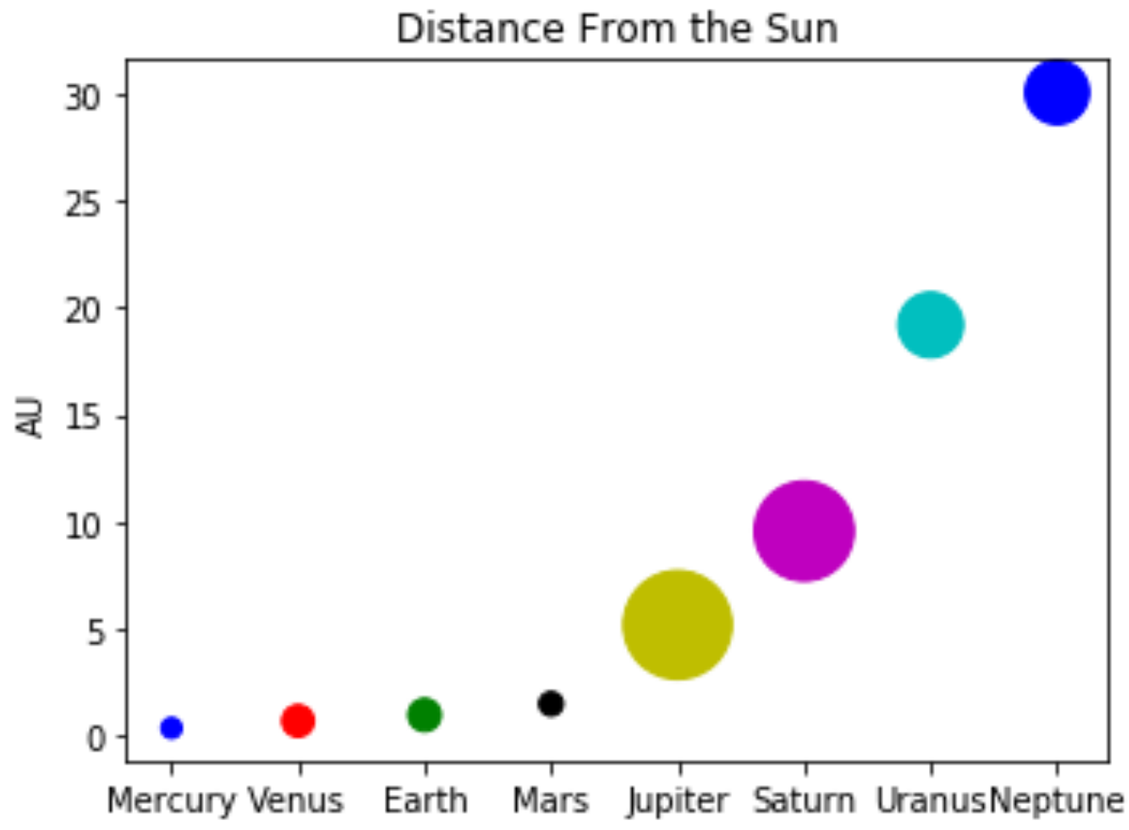
Scale (size) of bubbles

```
plt.scatter(num,distance)
```




```
plt.scatter(num,distance,radius*100,color=colors)
plt.ylabel('AU')
plt.title('Distance From the Sun')
plt.xticks(num,planets)
```

color of bubbles



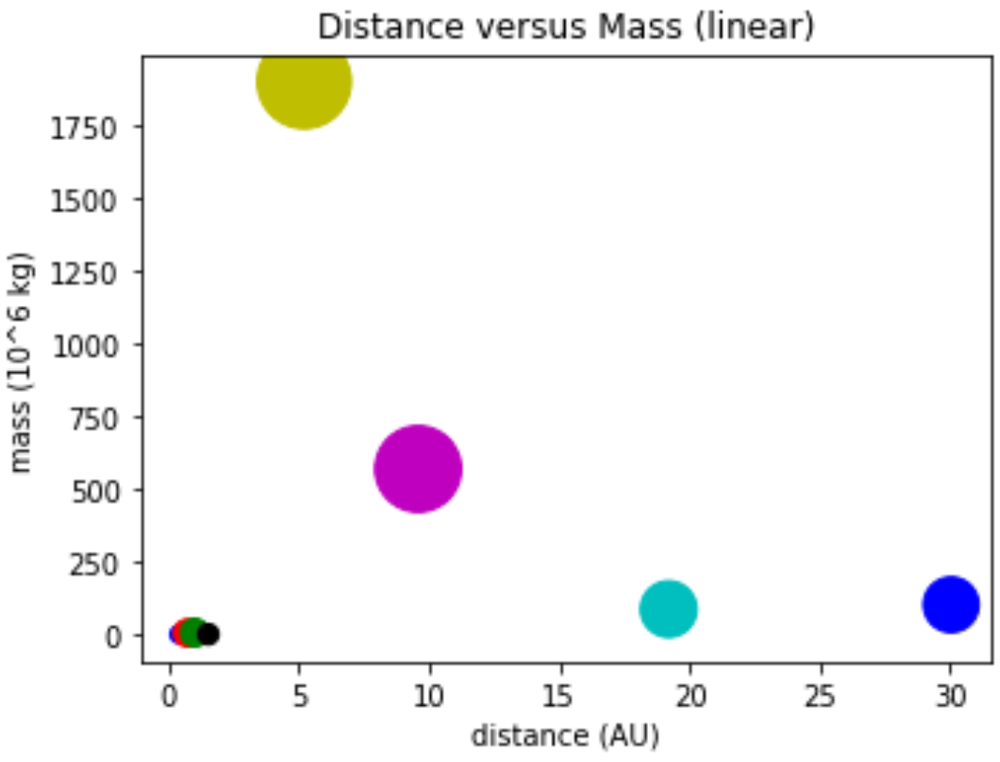
Scales of plot axis: linear versus logarithm

Let's try plot the mass of solar planets as a function of their distance from the sun:

```
planets = ['Mercury','Venus','Earth','Mars','Jupiter','Saturn','Uranus','Neptune']
colors = ['b','r','g','k','y','m','c','b']
distance = np.array([0.39,  0.72,  1.  ,  1.52,  5.2 ,  9.58, 19.2 , 30.05]) # normalized to Sun-Earth distance
radius = np.array([0.38,  0.95,  1.  ,  0.53, 11.21,  9.45,  4.01,  3.88]) # normalized to Earth radius
mass = np.array([0.330,4.87,5.97,0.642,1898,568,86.8,102])

plt.figure(5, figsize=(12,4))
```

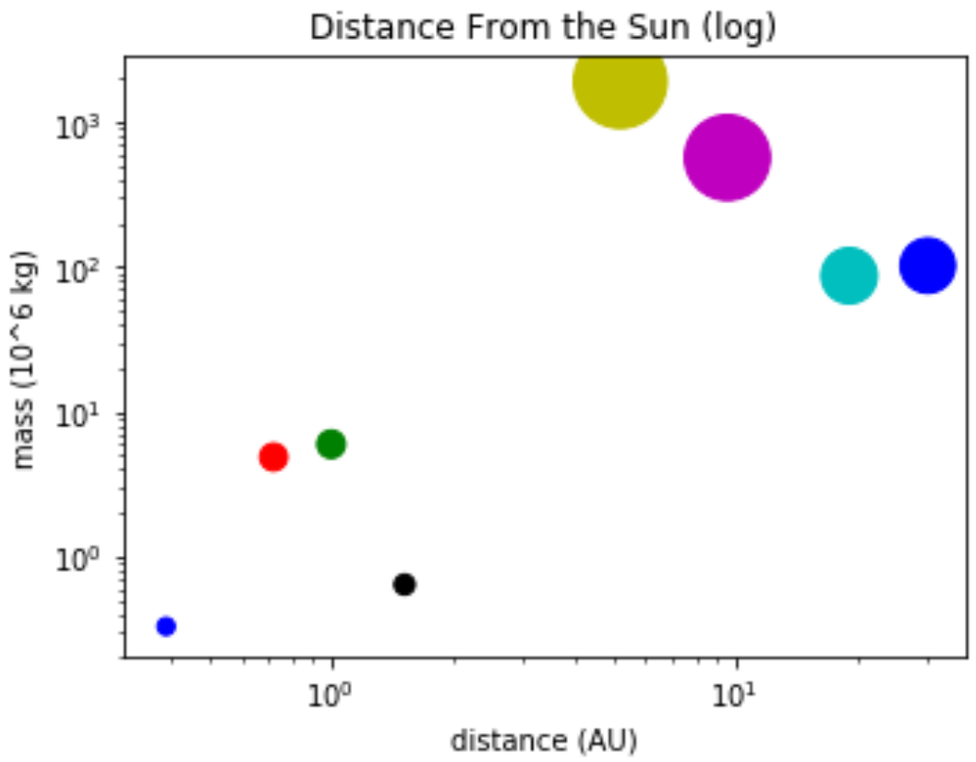
First try this one (linear scale):

```
plt.subplot(1,2,1)
plt.scatter(distance,mass,radius*100,color=colors) #
                                                   #
plt.xlabel('distance (AU)') # xlabel
plt.ylabel('mass (10^6 kg)') # ylabel
plt.title('Distance versus Mass (linear)') # title
```



Now try this one (log scale):

```
plt.scatter(distance,mass,radius*100,color=colors)
plt.xlabel('distance (AU)')
plt.ylabel('mass (10^6 kg)')
plt.title('Distance From the Sun (log)')
plt.xscale('log')
plt.yscale('log')
```



Now let's make some plots!