Downloading libraries

If you're not using Anaconda, you might have to install pip (if not already installed) for downloading libraries. Use the following command for the same: python -m pip install -U pip setuptools

For downloading libraries, use pip install library_name. E.g., for downloading numpy, you can use: pip install numpy.

These commands can be used in a terminal. They can also be used within the notebook by using a '!' before the command: !pip install numpy

In [1]:
```
!pip install numpy
```

```
Requirement already satisfied: numpy in /Users/mugdhab/anaconda3/lib/py
thon3.7/site-packages (1.16.3)
WARNING: You are using pip version 19.2.3, however version 20.0.1 is av
ailable.
You should consider upgrading via the 'pip install --upgrade pip' comma
nd.
```

In [2]:
```
print ("Hello")
```

```
Hello
```

Importing libraries

In [3]:
```
import numpy as np

import matplotlib.pyplot as plt
%matplotlib inline

from IPython.display import Image
```

To make plots using Matplotlib, you must first enable IPython's matplotlib mode. To do this, run the %matplotlib magic command to enable plotting in the current Notebook. This magic takes an optional argument that specifies which Matplotlib backend should be used. Most of the time, in the Notebook, you will want to use the inline backend, which will embed plots inside the Notebook:

Introduction to matplotlib

Refer https://matplotlib.org/tutorials/index.html (https://matplotlib.org/tutorials/index.html) for more information.

A Figure object is the outermost container for a matplotlib graphic, which can contain multiple Axes objects. One source of confusion is the name: an Axes actually translates into what we think of as an individual plot or graph (rather than the plural of “axis,” as we might expect).

You can think of the Figure object as a box-like container holding one or more Axes (actual plots). Below the Axes in the hierarchy are smaller objects such as tick marks, individual lines, legends, and text boxes. Almost every “element” of a chart is its own manipulable Python object, all the way down to the ticks and labels:

Plotting using plt.plot

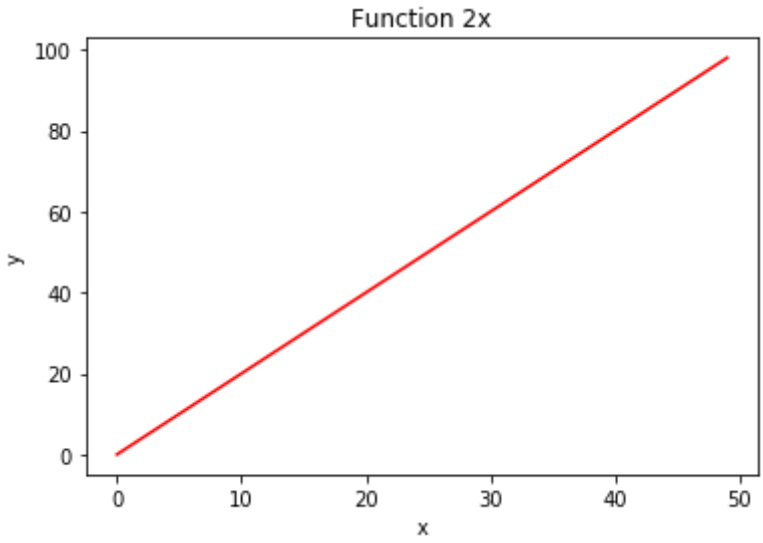
In [4]:
```
# Defining dummy data

x = np.arange(50)
y = 2*x

print(x)
print(y)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 2
3
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 4
7
 48 49]
[ 0  2  4  6  8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 4
6
 48 50 52 54 56 58 60 62 64 66 68 70 72 74 76 78 80 82 84 86 88 90 92 9
4
 96 98]
```

Coding tip

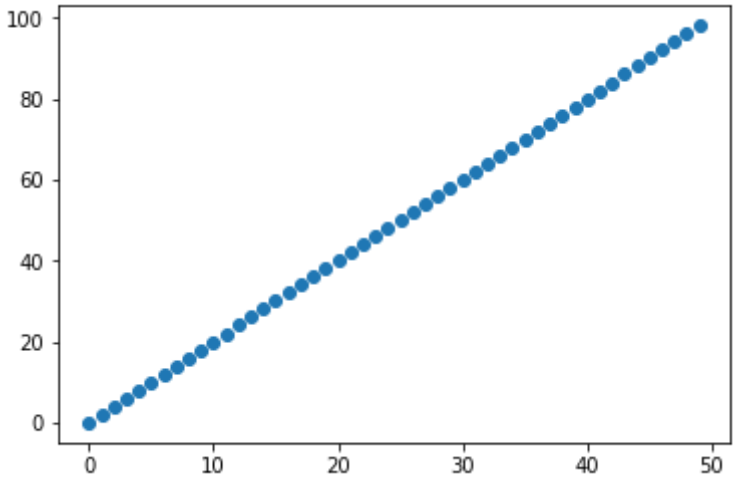
- When need to debug or read code, try printing at every line so its easier to understand.

In [5]:
```
#1a

plt.plot(x, y, color='red')
plt.title('Function 2x')
plt.xlabel('x')
plt.ylabel('y')

plt.show()
```



In [6]:
```
plt.scatter(x, y)
```

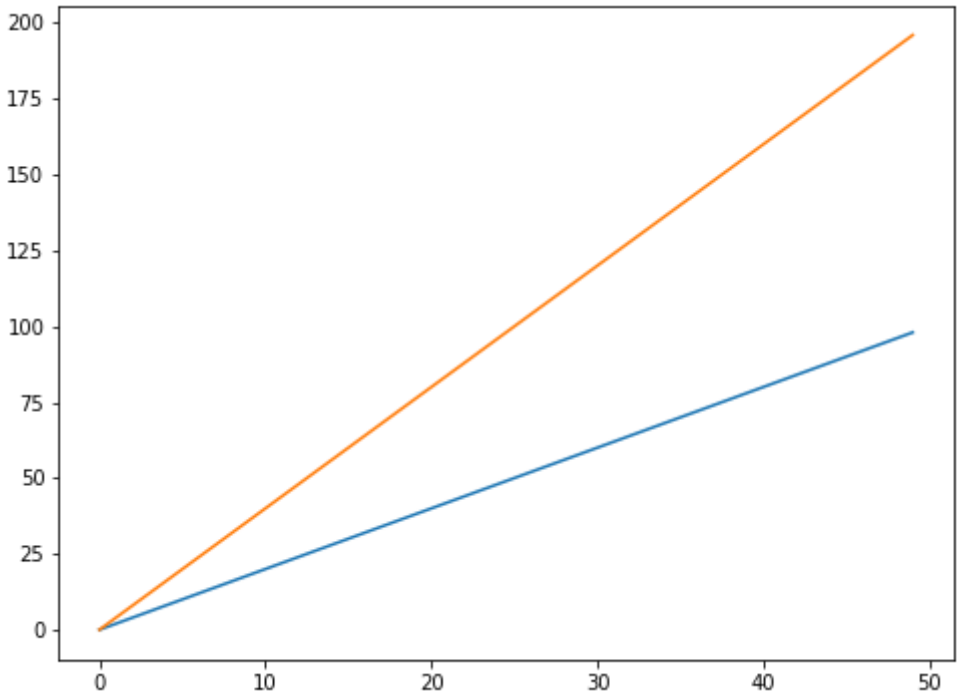
Out[6]:
```
<matplotlib.collections.PathCollection at 0x118437a58>
```



In order to change the size of the plot, you can call plt.figure(figsize=(x,y)).

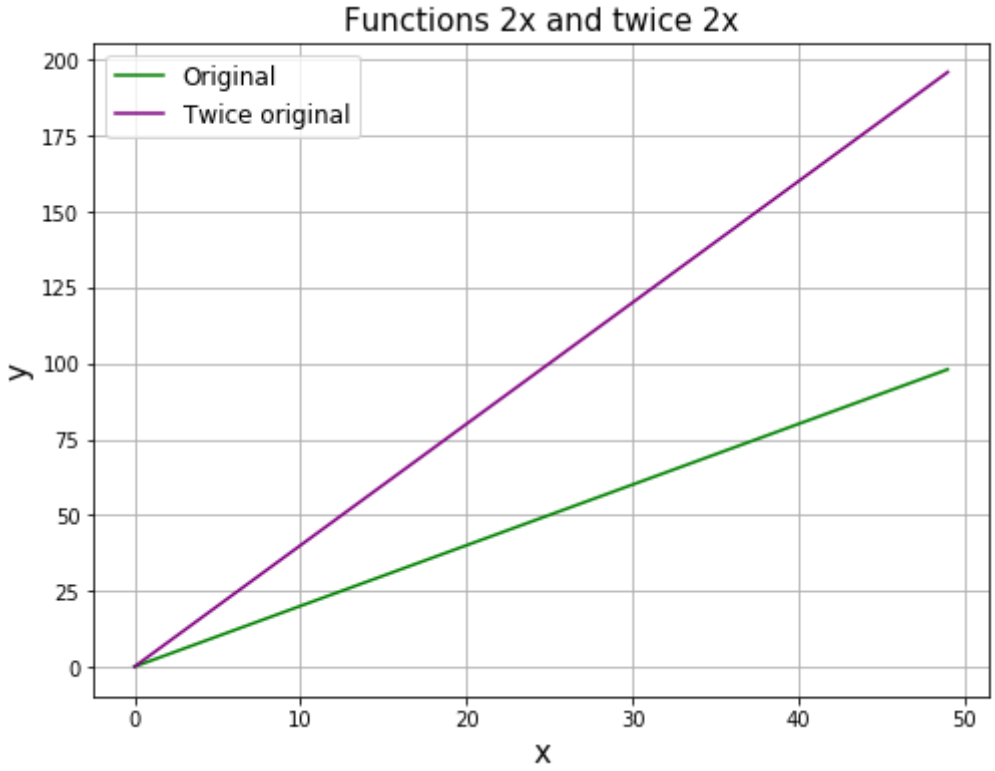
x is the desired width and y is the desired height.

In [10]:
```
plt.figure(figsize=(8,6))
plt.plot(x, y)

# If you want multiple lines on the same plot, you can call two plt.plot
functions with the respective data.
plt.plot(x, y+y)
plt.show()
```



A good plot should be self-explanatory. Add title, x-axis and y-axis labels, legend, etc.

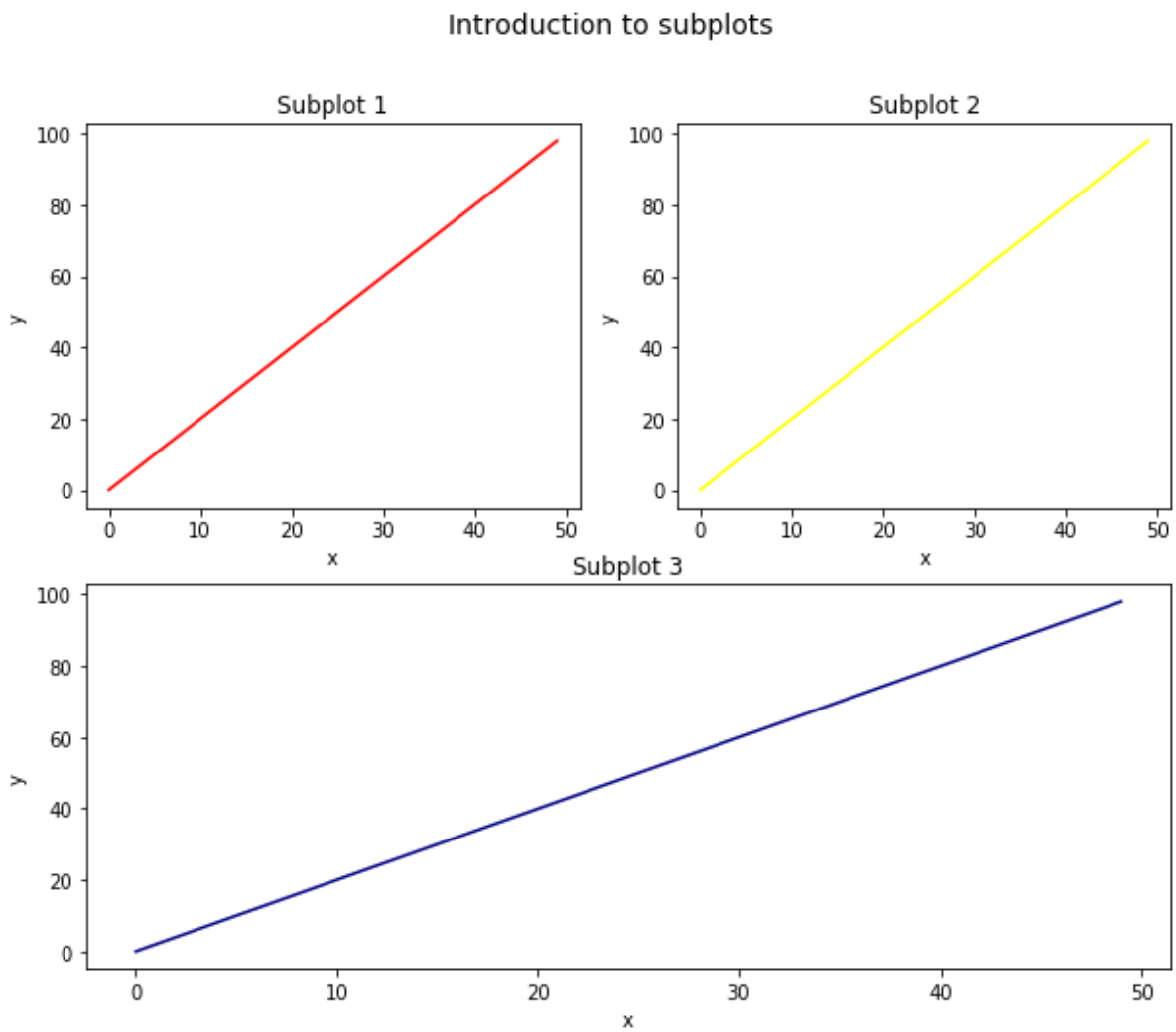
In [13]:
```
plt.figure(figsize=(8,6))
plt.plot(x, y, color = 'green', label = 'Original')
plt.plot(x, y+y, color = 'purple', label = 'Twice original')
plt.title('Functions 2x and twice 2x', fontsize = 15)
plt.xlabel('x', fontsize = 15)
plt.ylabel('y', fontsize = 15)
plt.legend(fontsize = 12)
plt.grid(True)
plt.show()
```



Creating subplots

In [14]:
```
fig = plt.figure(figsize=(10, 8))
fig.suptitle("Introduction to subplots", fontsize=14)

ax1 = fig.add_subplot(221)
plt.plot(x, y, color='red')
ax1.set_title("Subplot 1")
ax1.set_xlabel("x")
ax1.set_ylabel("y")

ax2 = fig.add_subplot(222)
plt.plot(x, y, color='yellow')
plt.title("Subplot 2")
plt.xlabel("x")
plt.ylabel("y")

ax3 = fig.add_subplot(212)
plt.plot(x, y, color='navy')
plt.title("Subplot 3")
plt.xlabel("x")
plt.ylabel("y")
```

Out[14]:
```
Text(0, 0.5, 'y')
```

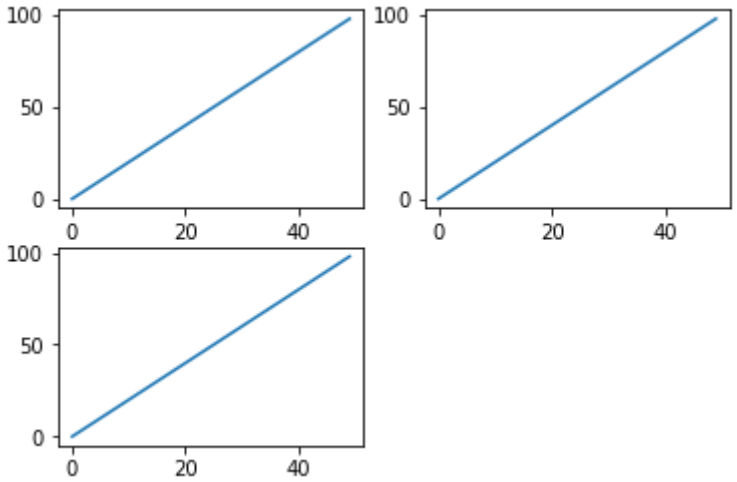


The way that this works is with 3 numbers, which are: height, width, plot number.

So, a 221 means 2 tall, 2 wide, plot number 1. 222 is 2 tall, 2 wide, and plot number 2. Finally, 212 is a 2 tall, 1 wide, plot number 1.

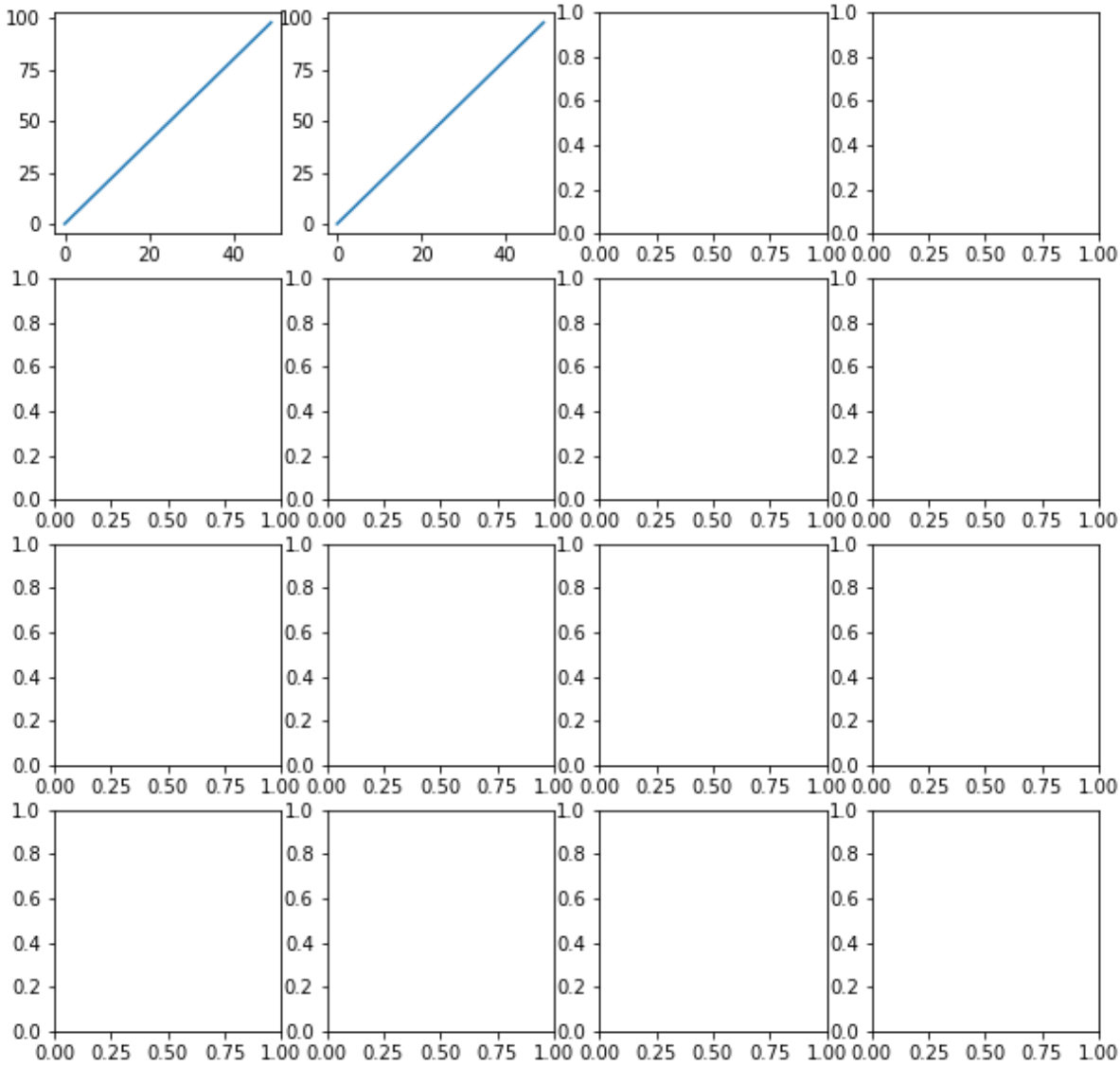
In [15]:
```
# equivalent but more general
ax1=plt.subplot(2,2,1)
plt.plot(x, y)
ax1=plt.subplot(2, 2, 2)
plt.plot(x, y)
ax1=plt.subplot(2, 2, 3)
plt.plot(x, y)
```

Out[15]:
```
[<matplotlib.lines.Line2D at 0x118dfc240>]
```



In [16]:
```
fig, axes = plt.subplots(nrows=4, ncols=4)
fig.set_figheight(10)
fig.set_figwidth(10)
#axes[0,0].set(title='Upper Left')
#axes[0,1].set(title='Upper Right')
#axes[1,0].set(title='Lower Left')
#axes[1,1].set(title='Lower Right')
axes[0,0].plot(x, y)
axes[0,1].plot(x, y)
```

Out[16]:
```
[<matplotlib.lines.Line2D at 0x10d496f98>]
```



Saving a plot

You could save a plot as a figure by using:

In [28]:
```
plt.savefig('dis1.png')
```

```
<matplotlib.figure.Figure at 0x110bb6358>
```

Writing for loops

Cumulative functions, summation functions

What can i interate over?

- We can make for loops run through numerical lists
  - range(10) means i goes over 0 to 9
  - range(2,10) means i goes over 2 to 9
  - [2,4,6,8] means i goes over these values
- We can make for loops run through string lists
  - ['dog','cat', 'monkey] mean i will assumes value dog in first cycle, cat in second and monkey in last

In [21]:
```
current_value = 0
list_of_values = [] # defines a list
for i in range(10):
    current_value = current_value + i    # can be written as current_val
ue += i
    list_of_values.append(current_value)

list_of_values
```

Out[21]:
```
[0, 1, 3, 6, 10, 15, 21, 28, 36, 45]
```

Writing functions

In [35]:
```
def example_function(a, b, x):
    y = a*x + b
    return y

example_x = np.arange(10)
example_y = example_function(5, 2, example_x)
print(example_y)
```

```
[ 2  7 12 17 22 27 32 37 42 47]
```

What is normal distribution?

A normal distribution, sometimes called the bell curve, is a distribution that occurs naturally in many situations. For example, the bell curve is seen in tests like the SAT and GRE. The bulk of students will score the average (C), while smaller numbers of students will score a B or D. An even smaller percentage of students score an F or an A. This creates a distribution that resembles a bell (hence the nickname). The bell curve is symmetrical. Half of the data will fall to the left of the mean; half will fall to the right.

The standard deviation controls the spread of the distribution. A smaller standard deviation indicates that the data is tightly clustered around the mean; the normal distribution will be taller. A larger standard deviation indicates that the data is spread out around the mean; the normal distribution will be flatter and wider.

Source: https://www.statisticshowto.datasciencecentral.com/probability-and-statistics/normal-distributions/ (https://www.statisticshowto.datasciencecentral.com/probability-and-statistics/normal-distributions/)

Generating random numbers

In [26]:
```
# For a random number between 0 and 1
np.random.random()
```

Out[26]:
```
0.6800560423941239
```

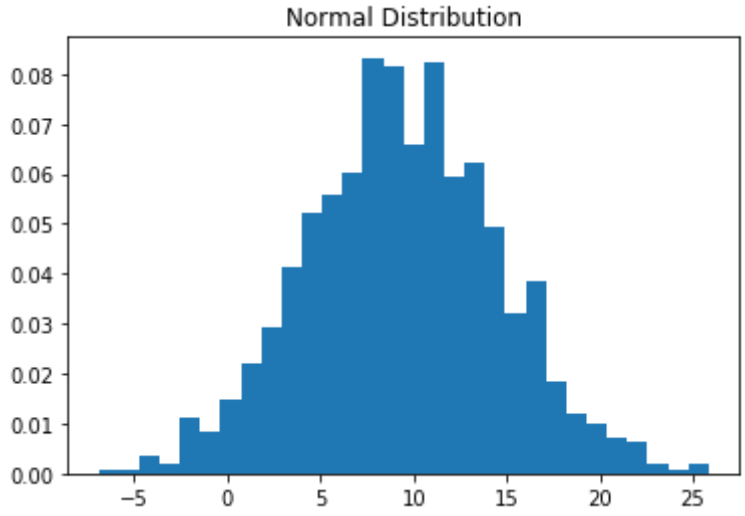
In [28]:
```
# For a random integer between 0 and specified integer
np.random.randint(100)
```

Out[28]:
```
25
```

In [47]:
```
# Method1: For a random sample drawn from a standard normal distributio
 n;

mu = 10 # mean
sigma = 5 #standard deviation
sample = np.random.normal(mu, sigma, 2)
sample
```

Out[47]:
```
array([4.87978325, 4.89282116])
```

In [52]:
```
sample_plot = np.random.normal(mu, sigma, 1000)
plt.hist(sample_plot, 30, normed=True) # bin it into n = 30 bins
plt.title("Normal Distribution")
plt.show()
```



In [ ]:
```
```