EVERYTHING ABOUT MATPLOTLIB

TAMAL DAS

1. Matplotlib – Introduction

Matplotlib is one of the most popular Python packages used for data visualization. It is a cross-platform library for making 2D plots from data in arrays. Matplotlib is written in Python and makes use of NumPy, the numerical mathematics extension of Python. It provides an object-oriented API that helps in embedding plots in applications using Python GUI toolkits such as PyQt, WxPythonotTkinter. It can be used in Python and IPython shells, Jupyter notebook and web application servers also.

Matplotlib has a procedural interface named the Pylab, which is designed to resemble MATLAB, a proprietary programming language developed by MathWorks. Matplotlib along with NumPy can be considered as the open source equivalent of MATLAB.

Matplotlib was originally written by John D. Hunter in 2003. The current stable version is 2.2.0 released in January 2018.

2. Matplotlib – Environment Setup

Matplotlib and its dependency packages are available in the form of wheel packages on the standard Python package repositories and can be installed on Windows, Linux as well as MacOS systems using the pip package manager.

```
pip3 install matplotlib
```

Incase Python 2.7 or 3.4 versions are not installed for all users, the Microsoft Visual C++ 2008 (64 bit or 32 bit forPython 2.7) or Microsoft Visual C++ 2010 (64 bit or 32 bit for Python 3.4) redistributable packages need to be installed.

If you are using Python 2.7 on a Mac, execute the following command:

```
xcode-select –install
```

Upon execution of the above command, the subprocess32 - a dependency, may be compiled.

On extremely old versions of Linux and Python 2.7, you may need to install the master version of subprocess32.

Matplotlib requires a large number of dependencies:

- Python (>= 2.7 or >= 3.4)
- NumPy
- setuptools
- dateutil
- pyparsing
- libpng
- pytz
- FreeType
- cycler
- six

Optionally, you can also install a number of packages to enable better user interface toolkits.

- tk
- PyQt4
- PyQt5
- wxpython
- pycairo
- Tornado

For better support of animation output format and image file formats, LaTeX, etc., you can install the following:

- _mpeg/avconv
- ImageMagick
- Pillow (>=2.0)
- LaTeX and GhostScript (for rendering text with LaTeX).

3. Matplotlib – Jupyter Notebook

Jupyter is a loose acronym meaning Julia, Python, and R. These programming languages were the first target languages of the Jupyter application, but nowadays, the notebook technology also supports many other languages.

In 2001, Fernando Pérez started developing Ipython. **IPython** is a command shell for interactive computing in multiple programming languages, originally developed for the Python.

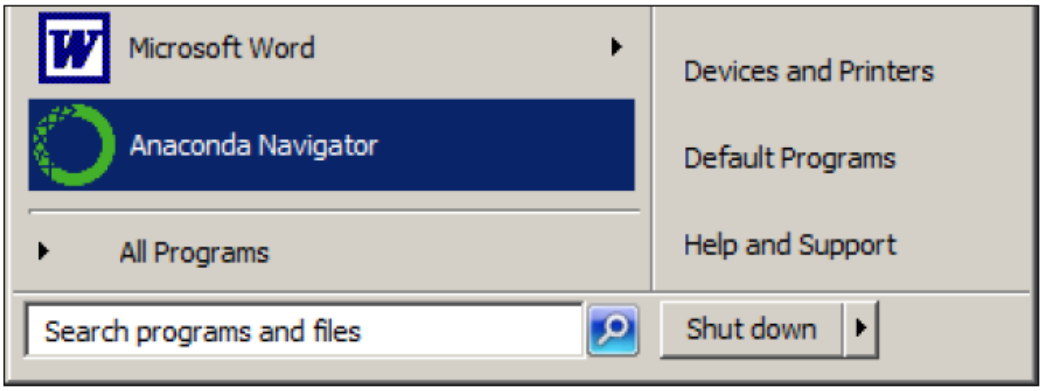
In 2001, Fernando Pérez started developing Ipython. **IPython** is a command shell for interactive computing in multiple programming languages, originally developed for the Python.

Consider the following features provided by IPython:

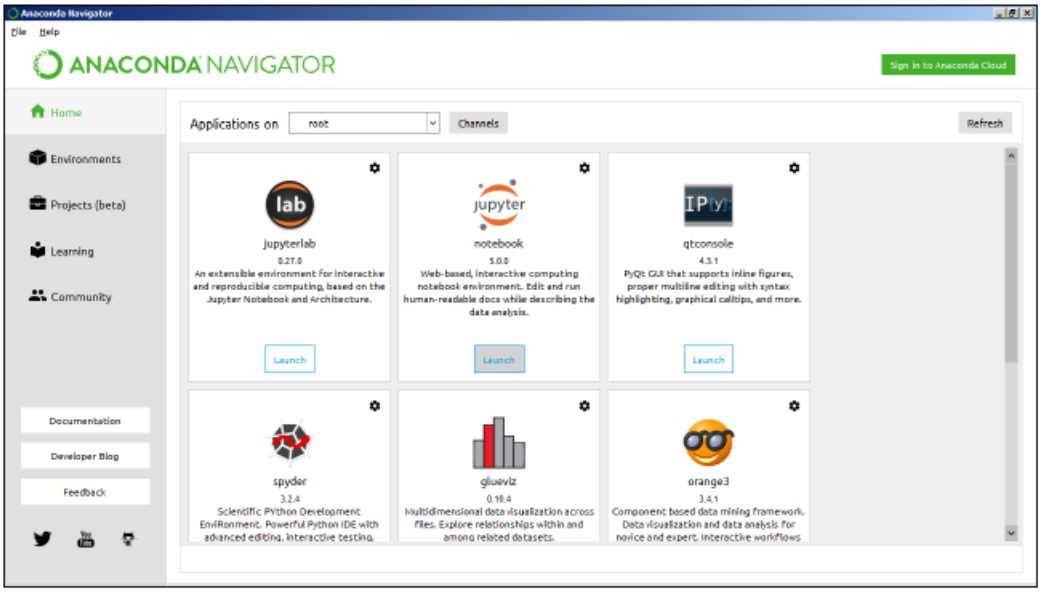
- Interactive shells (terminal and Qt-based).
- A browser-based notebook with support for code, text, mathematical expressions, inline plots and other media.
- Support for interactive data visualization and use of GUI toolkits.
- Flexible, embeddable interpreters to load into one's own projects.

In 2014, Fernando Pérez announced a spin-off project from IPython called Project Jupyter. IPython will continue to exist as a Python shell and a kernel for Jupyter, while the notebook and other language-agnostic parts of IPython will move under the Jupyter name. Jupyter added support for Julia, R, Haskell and Ruby.

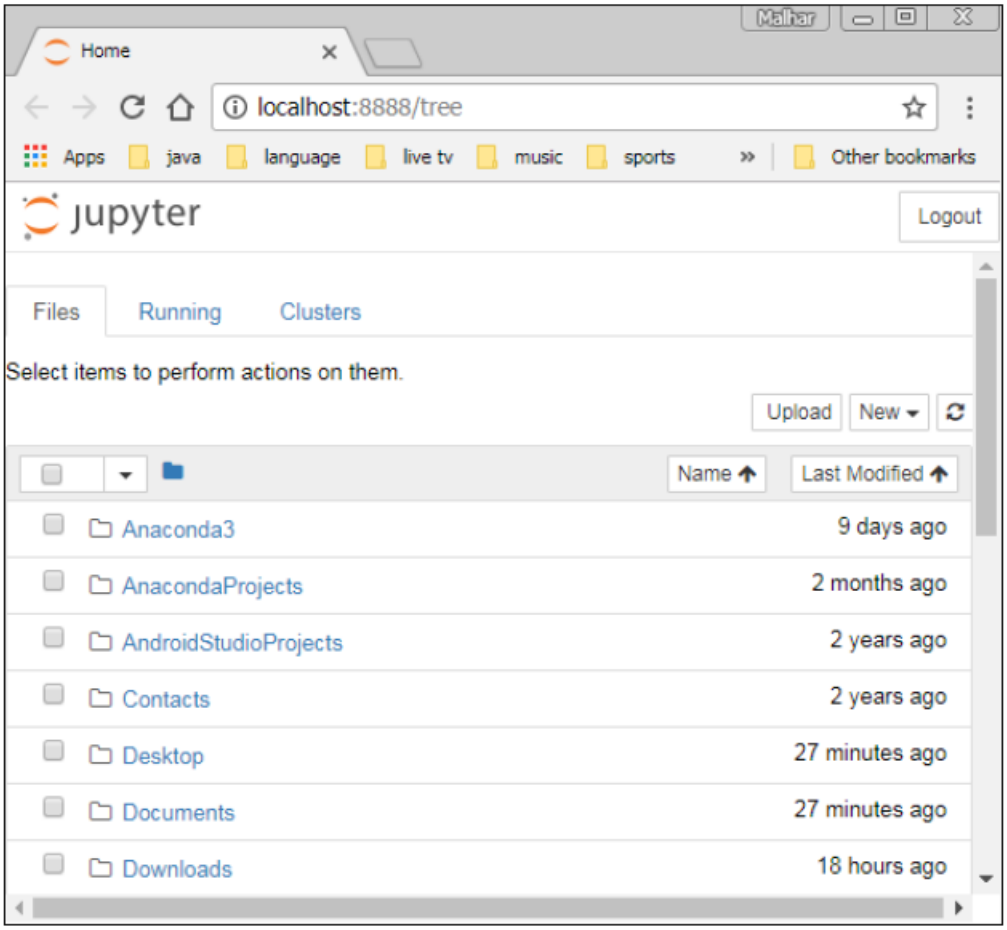
To start the Jupyter notebook, open Anaconda navigator (a desktop graphical user interface included in Anaconda that allows you to launch applications and easily manage Conda packages, environments and channels without the need to use command line commands).



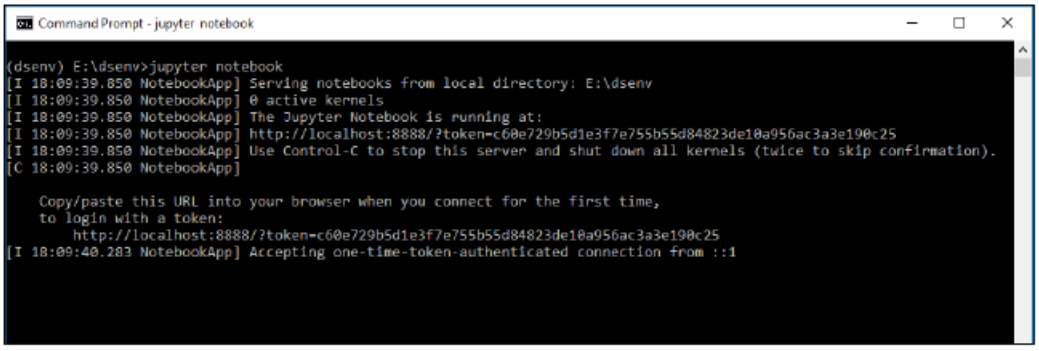
Navigator displays the installed components in the distribution.



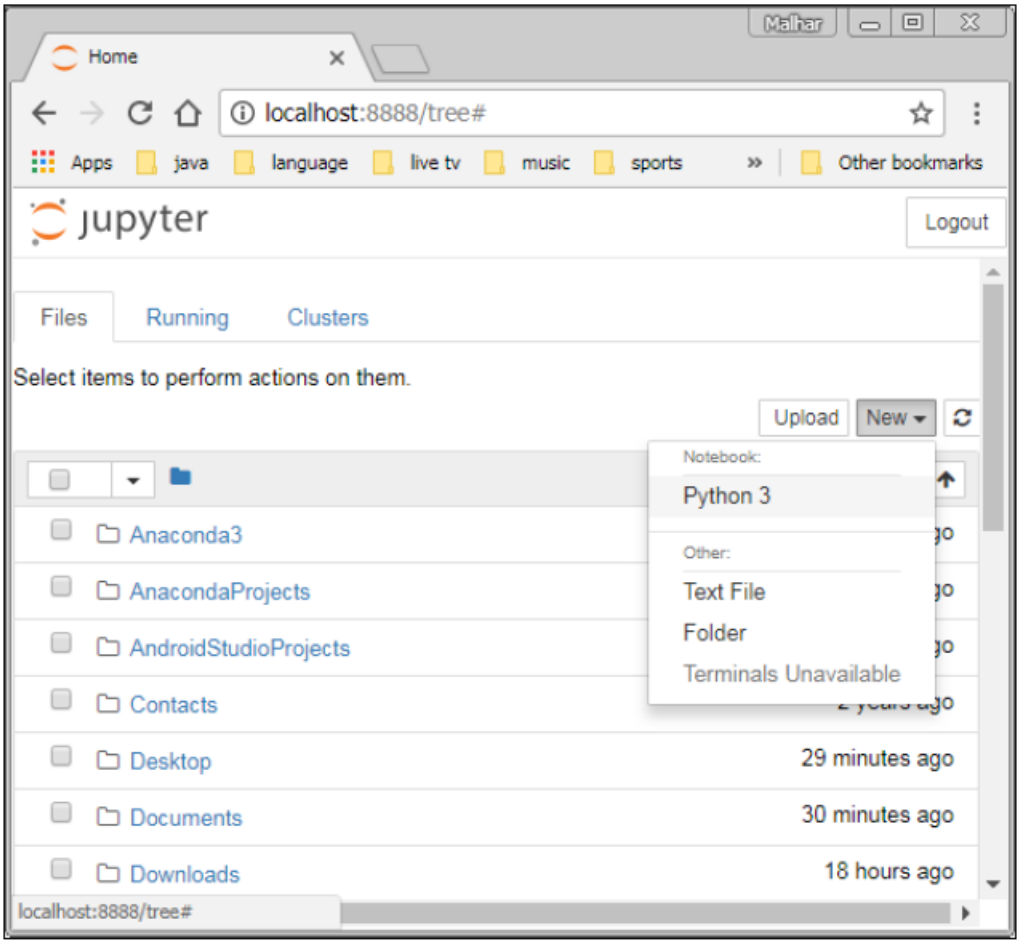
Launch Jupyter Notebook from the Navigator:



You will see the application opening in the web browser on the following address: http://localhost:8888.

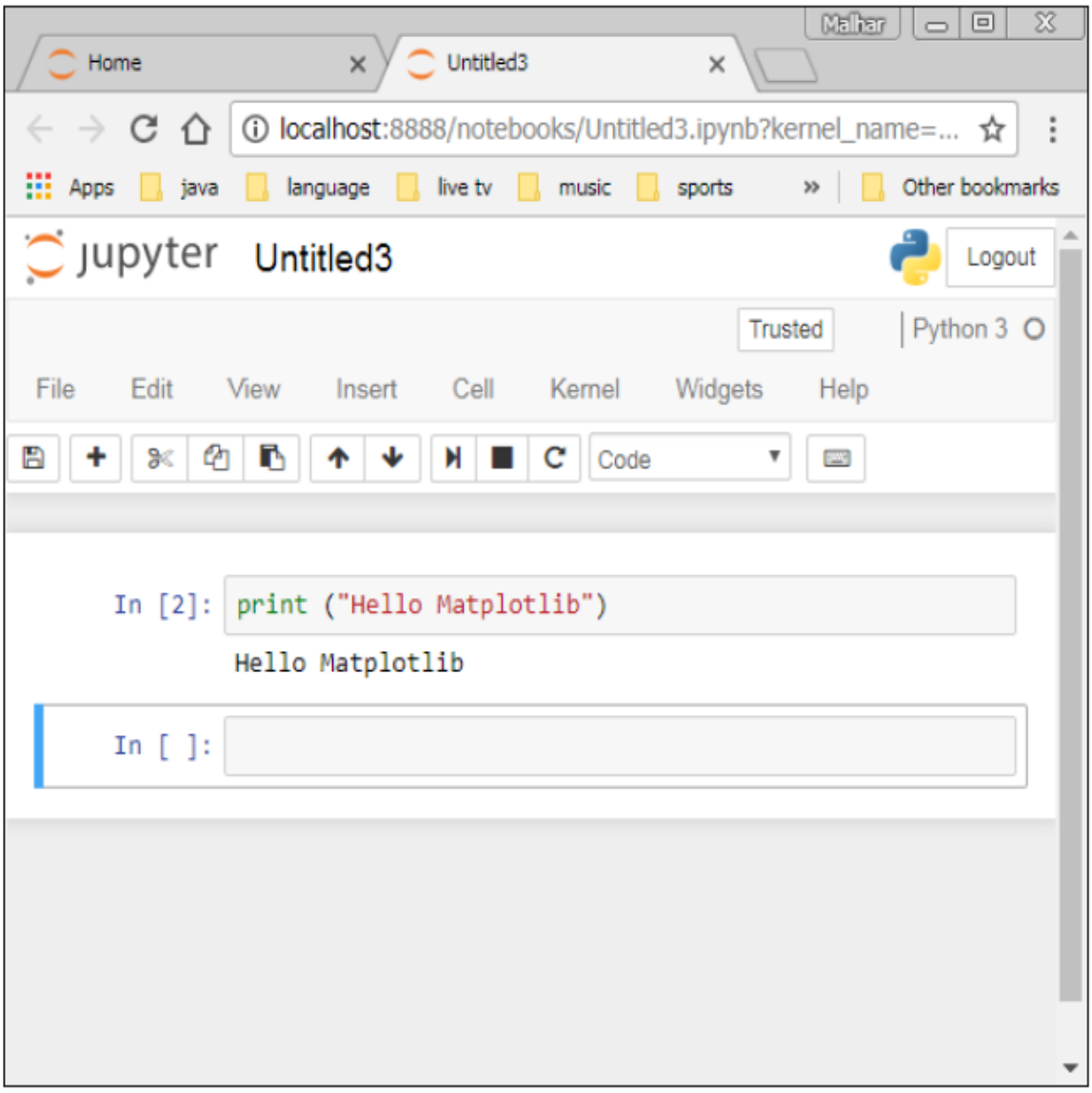


You probably want to start by making a new notebook. You can easily do this by clicking on the "New button" in the "Files tab". You see that you have the option to make a regular text file, a folder, and a terminal. Lastly, you will also see the option to make a Python 3 notebook.



4. Matplotlib – Pyplot API

A new untitled notebook with the **.ipynb** extension (stands for the IPython notebook) is displayed in the new tab of the browser.



matplotlib.pyplot is a collection of command style functions that make Matplotlib work like MATLAB. Each Pyplot function makes some change to a figure. For example, a function creates a figure, a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc.

Types of Plots

Function	**Description**

Bar	Make a bar plot.
Barh	Make a horizontal bar plot.
Boxplot	Make a box and whisker plot.
Hist	Plot a histogram.
hist2d	Make a 2D histogram plot.
Pie	Plot a pie chart.
Plot	Plot lines and/or markers to the Axes.
Polar	Make a polar plot.
Scatter	Make a scatter plot of x vs y.
Stackplot	Draws a stacked area plot.
Stem	Create a stem plot.
Step	Make a step plot.
Quiver	Plot a 2-D field of arrows.

Image Functions

Function	Description
Imread	Read an image from a file into an array.
Imsave	Save an array as in image file.
Imshow	Display an image on the axes.

Axis Functions

Function	Description
Axes	Add axes to the figure.
Text	Add text to the axes.
Title	Set a title of the current axes.
Xlabel	Set the x axis label of the current axis.
Xlim	Get or set the x limits of the current axes.
Xscale	Set the scaling of the x-axis.
Xticks	Get or set the x-limits of the current tick locations and labels.
Ylabel	Set the y axis label of the current axis.
Ylim	Get or set the y-limits of the current axes.
Yscale	Set the scaling of the y-axis.
Yticks	Get or set the y-limits of the current tick locations and labels.

Figure Functions

Function	**Description**
Figtext	Add text to figure.
Figure	Creates a new figure.
Show	Display a figure.
Savefig	Save the current figure.
Close	Close a figure window.

5. Matplotlib – Simple Plot

In this chapter, we will learn how to create a simple plot with Matplotlib.

We shall now display a simple line plot of angle in radians vs. its sine value in Matplotlib. To begin with, the Pyplot module from Matplotlib package is imported, with an alias plt as a matter of convention.

```
import matplotlib.pyplot as plt
```

Next we need an array of numbers to plot. Various array functions are defined in the NumPy library which is imported with the np alias.

```
import numpy as np
```

We now obtain the ndarray object of angles between 0 and 2n using the arange() function from the NumPy library.

```
x=np.arange(0, math.pi*2, 0.05)
```

The ndarray object serves as values on x axis of the graph. The corresponding sine values of angles in x to be displayed on y axis are obtained by the following statement:

```
y=np.sin(x)
```

The values from two arrays are plotted using the plot() function.

```
plt.plot(x,y)
```

You can set the plot title, and labels for x and y axes.

```
plt.xlabel("angle")
plt.ylabel("sine")
plt.title('sine wave')
```

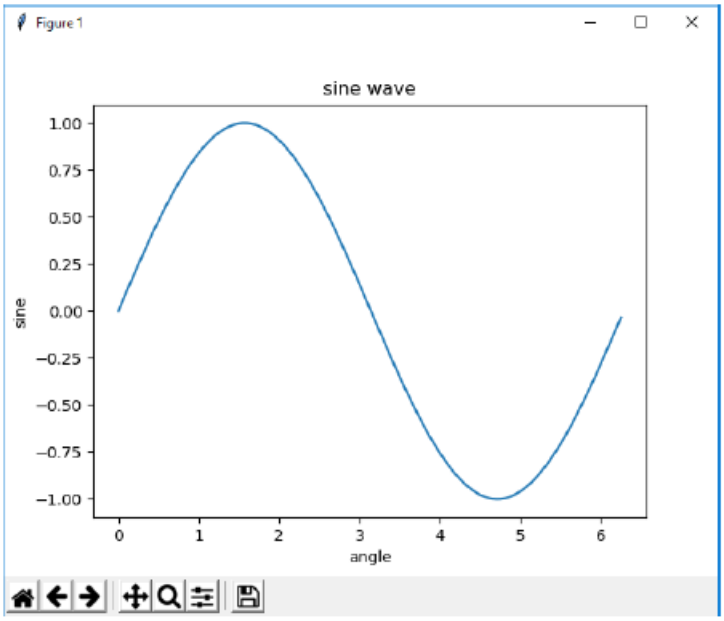
The Plot viewer window is invoked by the show() function:

```
plt.show()
```

The complete program is as follows:

```
from matplotlib import pyplot as plt
import numpy as np
import math #needed for definition of pi
x=np.arange(0, math.pi*2, 0.05)
y=np.sin(x)
plt.plot(x,y)
plt.xlabel("angle")
plt.ylabel("sine")
plt.title('sine wave')
plt.show()
```

When the above line of code is executed, the following graph is displayed:



Now, use the Jupyter notebook with Matplotlib.

Launch the Jupyter notebook from Anaconda navigator or command line as described earlier. In the input cell, enter import statements for Pyplot and NumPy:

```
from matplotlib import pyplot as plt

import numpy as np
```

To display plot outputs inside the notebook itself (and not in the separate viewer), enter the following magic statement:

```
%matplotlib inline
```

Obtain x as the ndarray object containing angles in radians between 0 to 2π, and y as sine value of each angle:

```
import math
x=np.arange(0, math.pi*2, 0.05)
y=np.sin(x)
```

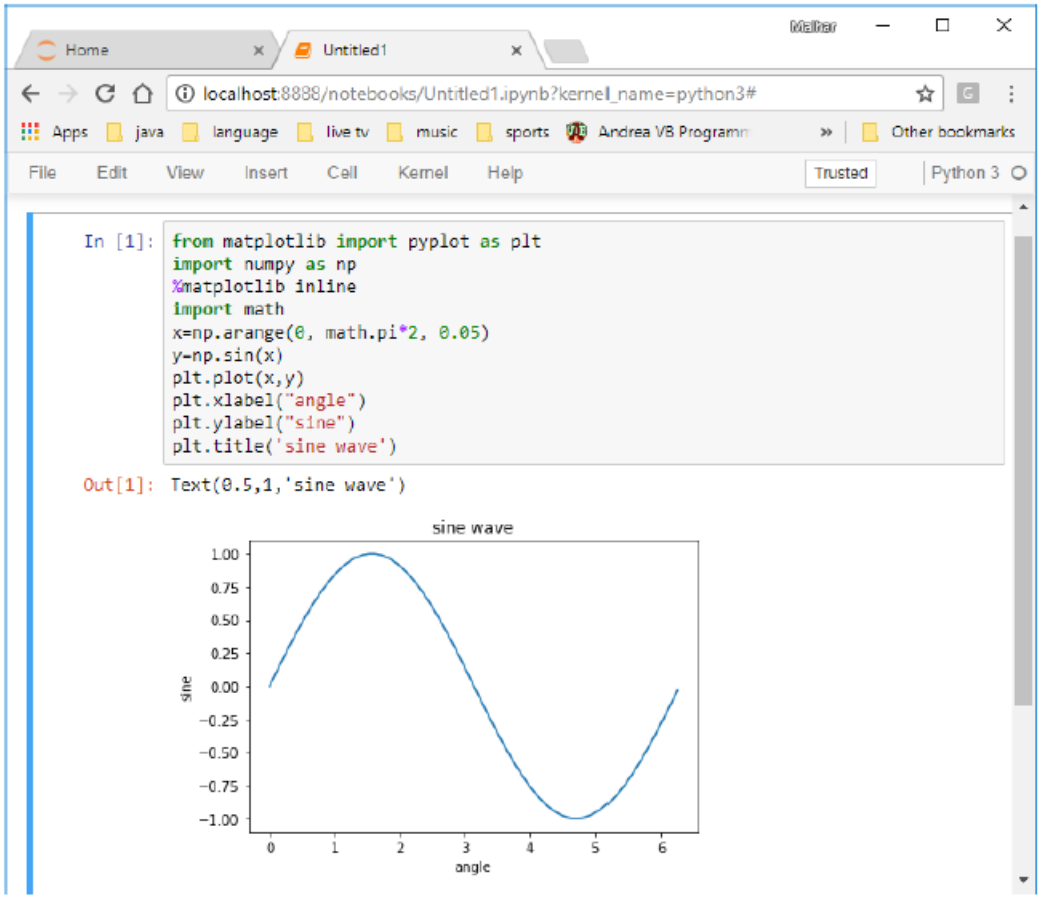
Set labels for x and y axes as well as the plot title:

```
plt.xlabel("angle")
plt.ylabel("sine")
plt.title('sine wave')
```

Finally execute the plot() function to generate the sine wave display in the notebook (no need to run the show() function):

```
plt.plot(x,y)
```

After the execution of the final line of code, the following output is displayed:



6. Matplotlib – PyLab module

PyLab is a procedural interface to the Matplotlib object-oriented plotting library. Matplotlib is the whole package; matplotlib.pyplot is a module in Matplotlib; and PyLab is a module that gets installed alongside Matplotlib.

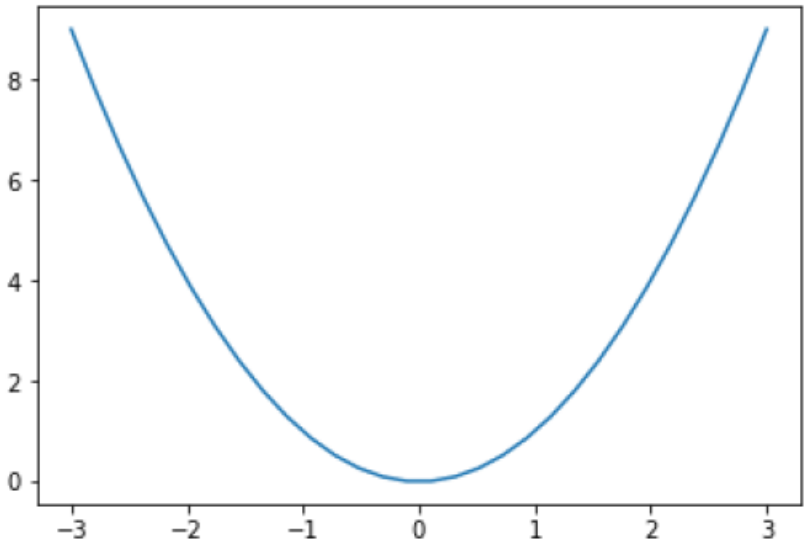
PyLab is a convenience module that bulk imports matplotlib.pyplot (for plotting) and NumPy (for Mathematics and working with arrays) in a single name space. Although many examples use PyLab, it is no longer recommended.

Basic Plotting

Plotting curves is done with the plot command. It takes a pair of same-length arrays (or sequences):

```
from numpy import *
from pylab import *
x = linspace(-3, 3, 30)
y = x**2
plot(x, y)
show()
```

The above line of code generates the following output:



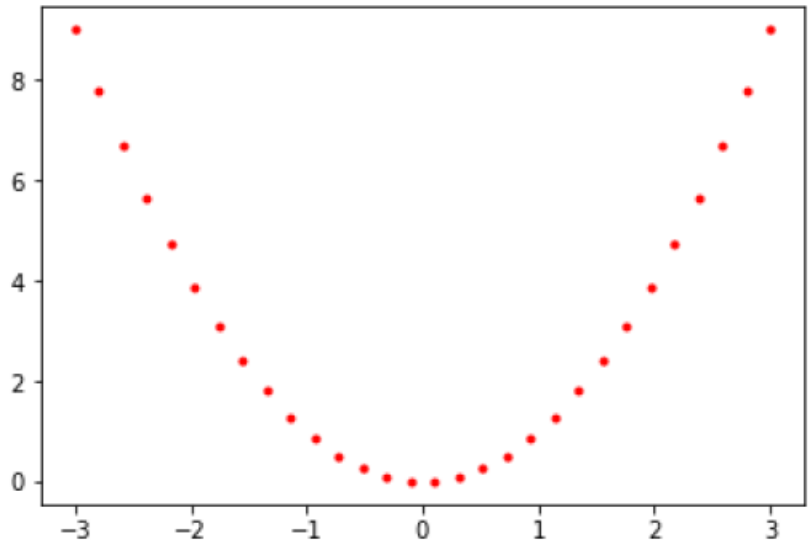
To plot symbols rather than lines, provide an additional string argument.

symbols	- , –, -., , . , , , o , ^ , v , < , > , s , + , x , D , d , 1 , 2 , 3 , 4 , h , H , p , | , _
colors	b, g, r, c, m, y, k, w

Now, consider executing the following code:

```
from pylab import *
x = linspace(-3, 3, 30)
y = x**2
plot(x, y, 'r.')
show()
```

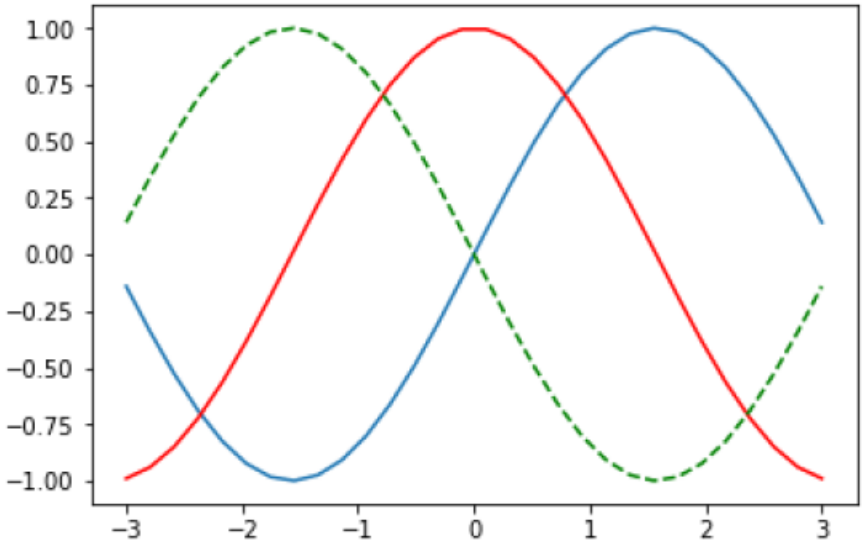
It plots the red dots as shown below:



Plots can be overlaid. Just use the multiple plot commands. Use clf() to clear the plot.

```
from pylab import *
plot(x, sin(x))
plot(x, cos(x), 'r-')
plot(x, -sin(x), 'g--')
show()
```

The above line of code generates the following output:



7. Matplotlib – Object-oriented Interface

While it is easy to quickly generate plots with the **matplotlib.pyplot** module, the use of object-oriented approach is recommended as it gives more control and customization of your plots. Most of the functions are also available in the **matplotlib.axes.Axes** class.

The main idea behind 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 helps better in dealing with a canvas that has multiple plots on it.

In object-oriented interface, Pyplot is used only for a few functions such as figure creation, and the user explicitly creates and keeps track of the figure and axes objects. At this level, the user uses Pyplot to create figures, and through those figures, one or more axes objects can be created. These axes objects are then used for most plotting actions.

To begin with, we create a figure instance which provides an empty canvas.

```
fig = plt.figure()
```

Now add axes to figure. The **add_axes()** method requires a list object of 4 elements corresponding to left, bottom, width and height of the figure. Each number must be between 0 and 1:

```
ax=fig.add_axes([0,0,1,1])
```

Set labels for x and y axis as well as title:

```
ax.set_title("sine wave")
ax.set_xlabel('angle')
ax.set_ylabel('sine')
```

Invoke the plot() method of the axes object.

```
ax.plot(x,y)
```

If you are using Jupyter notebook, the %matplotlib inline directive has to be issued; the otherwistshow() function of pyplot module displays the plot.

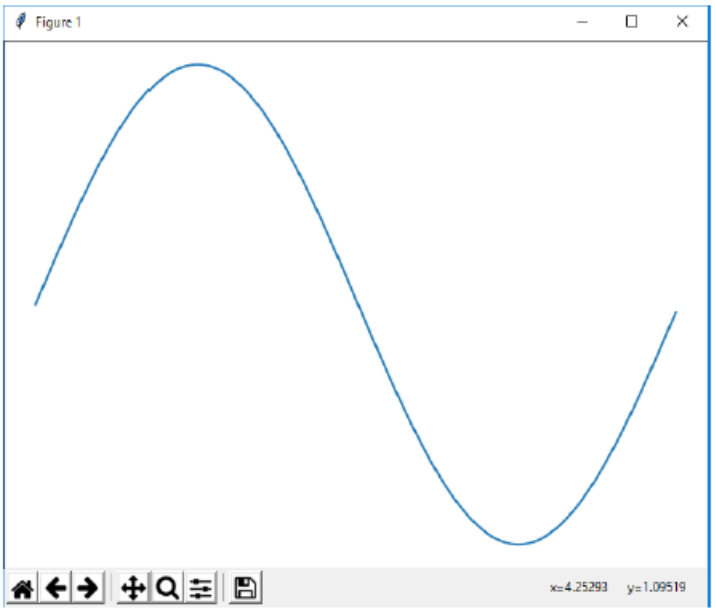
Consider executing the following code:

```
from matplotlib import pyplot as plt
import numpy as np
import math
x=np.arange(0, math.pi*2, 0.05)
```

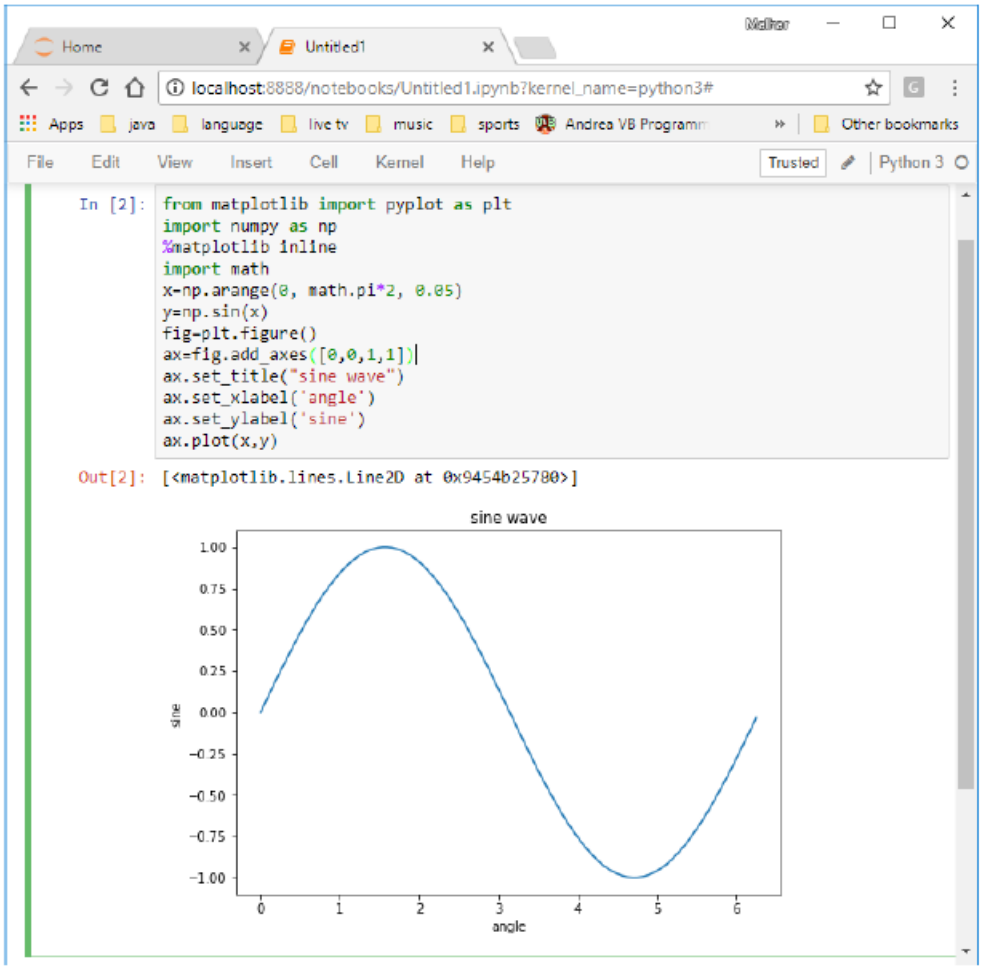
```
y=np.sin(x)
fig=plt.figure()
ax=fig.add_axes([0,0,1,1])
ax.plot(x,y)
ax.set_title("sine wave")
ax.set_xlabel('angle')
ax.set_ylabel('sine')
plt.show()
```

Output

The above line of code generates the following output:



The same code when run in Jupyter notebook shows the output as shown below:



8. Matplotlib – Figure Class

The **matplotlib.figure** module contains the Figure class. It is a top-level container for all plot elements. The Figure object is instantiated by calling the **figure()** function from the pyplot module:

```
fig=plt.figure()
```

The following table shows the additional parameters:

Figsize	(width,height) tuple in inches
Dpi	Dots per inches
Facecolor	Figure patch facecolor
Edgecolor	Figure patch edge color
Linewidth	Edge line width

9. Matplotlib – Axes Class

Axes object is the region of the image with the data space. A given figure can contain many Axes, but a given Axes object can only be in one Figure. The Axes contains two (or three in the case of 3D) Axis objects. The Axes class and its member functions are the primary entry point to working with the OO interface.

Axes object is added to figure by calling the add_axes() method. It returns the axes object and adds an axes at position rect [left, bottom, width, height] where all quantities are in fractions of figure width and height.

Parameter

Following is the parameter for the Axes class:

- rect: A 4-length sequence of [left, bottom, width, height] quantities.

```
ax=fig.add_axes([0,0,1,1])
```

The following member functions of axes class add different elements to plot:

Legend

The **legend**() method of axes class adds a legend to the plot figure. It takes three parameters:

```
ax.legend(handles, labels, loc)
```

Where labels is a sequence of strings and handles a sequence of Line2D or Patch instances. loc can be a string or an integer specifying the legend location.

Location string	Location code
Best	0
upper right	1
upper left	2
lower left	3
lower right	4
Right	5
center left	6
Center right	7
lower center	8
upper center	9
Center	10

axes.plot()

This is the basic method of axes class that plots values of one array versus another as lines or markers. The plot() method can have an optional format string argument to specify color, style and size of line and marker.

Color codes

Character	Color
'b'	Blue
'g'	Green
'r'	Red
'c'	Cyan
'm'	Magenta
'y'	Yellow
'k'	Black
'w'	White

Marker codes

Character	Description
'.'	Point marker
'o'	Circle marker
'x'	X marker
'D'	Diamond marker
'H'	Hexagon marker
's'	Square marker
'+'	Plus marker

Line styles

Character	Description
'-'	Solid line
'--'	Dashed line
'-.'	Dash-dot line
':'	Dotted line

Following example shows the advertisement expenses and sales figures of TV and smartphone in the form of line plots. Line representing TV is a solid line with yellow colour and square markers whereas smartphone line is a dashed line with green colour and circle marker.

```
import matplotlib.pyplot as plt
y = [1, 4, 9, 16, 25,36,49, 64]
x1 = [1, 16, 30, 42,55, 68, 77,88]
x2 = [1,6,12,18,28, 40, 52, 65]
fig=plt.figure()
ax=fig.add_axes([0,0,1,1])
l1=ax.plot(x1,y,'ys-') # solid line with yellow colour and square marker
l2=ax.plot(x2,y,'go--') # dash line with green colour and circle marker
ax.legend(labels=('tv', 'Smartphone'), loc='lower right') # legend placed at
lower right
ax.set_title("Advertisement effect on sales")
ax.set_xlabel('medium')
ax.set_ylabel('sales')
plt.show()
```

When the above line of code is executed, it produces the following plot: