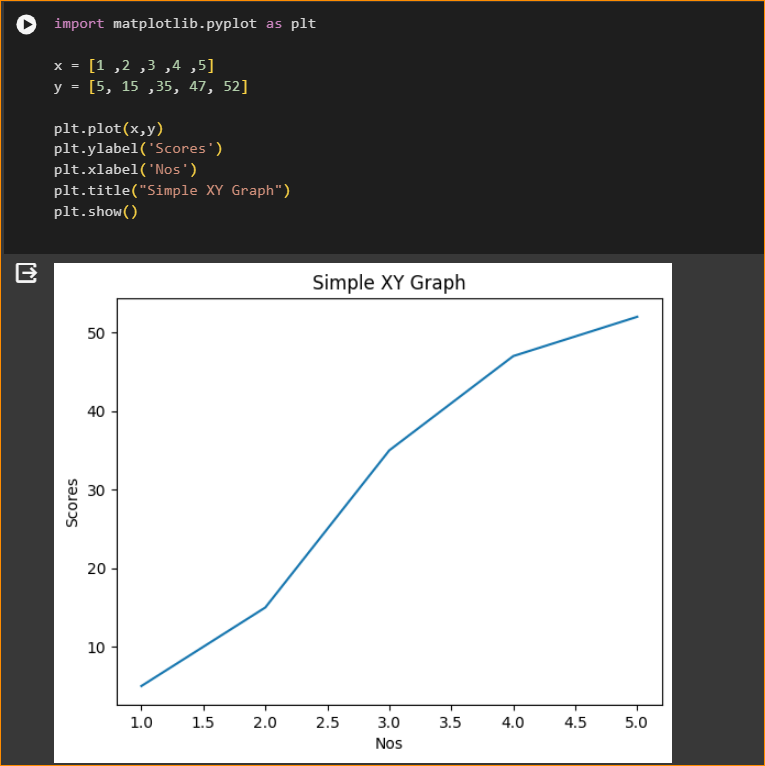
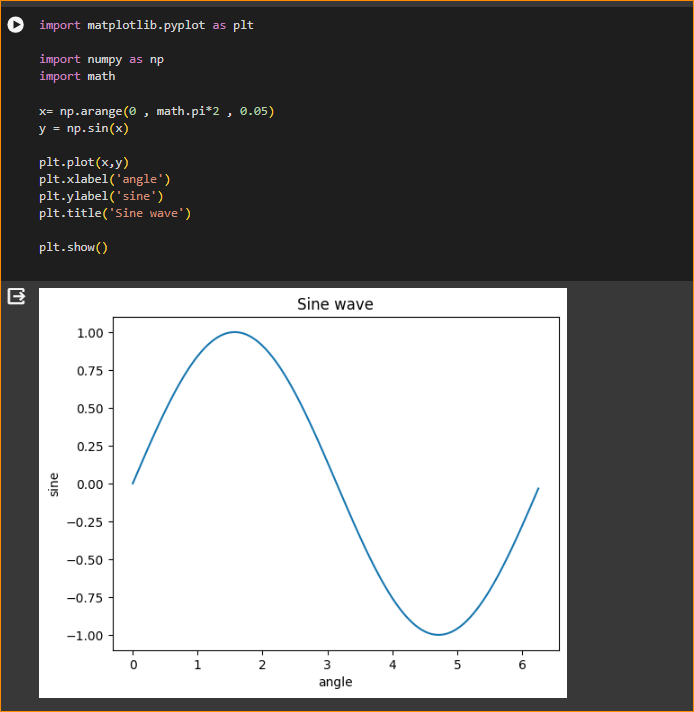
1. Using Matplotlib

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..



In here I have Add two simple arrays and made a pyploat.

1. Sin(x) plot



**import matplotlib.pyplot as plt** # this is the default declaration, we are importing from the matplotlib package, pyplot is a class and we are aliasing it as plt

**import numpy as np** # We need numpy as well

**import mat**h #needed for definition of pi

**x = np.arange(0, math.pi\*2, 0.05)** # generate an array from 0 to math.pi\*2 with

# an interval of 0.05

**y = np.sin(x)** # built in trignometry function in numpy

**plt.plot(x,y)** # an x, y plot

**plt.xlabel("angle")**

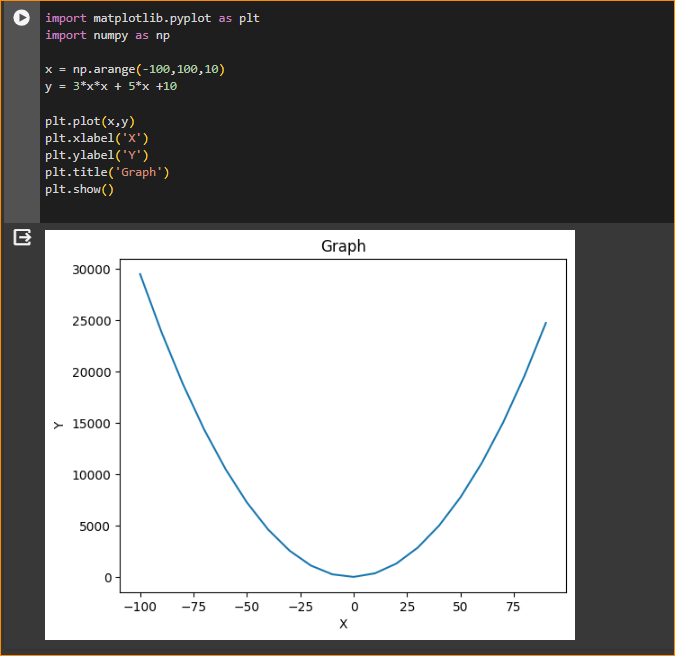
**plt.ylabel("sine")**

**plt.title('sine wave')**

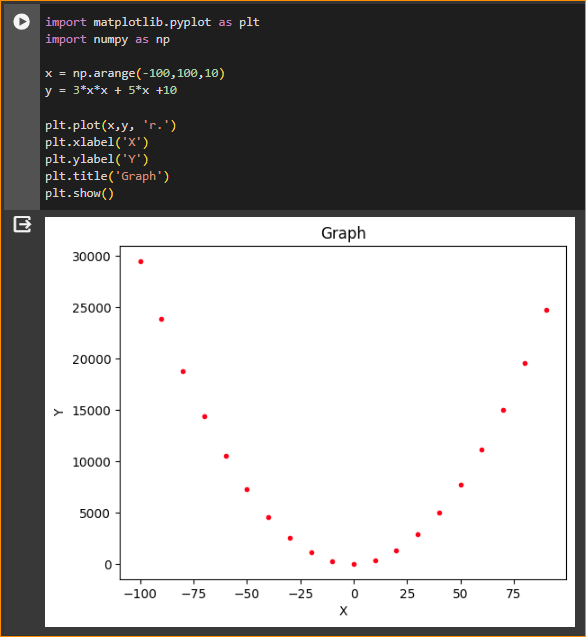
**plt.show()** # the plot is shown only when you give the show command

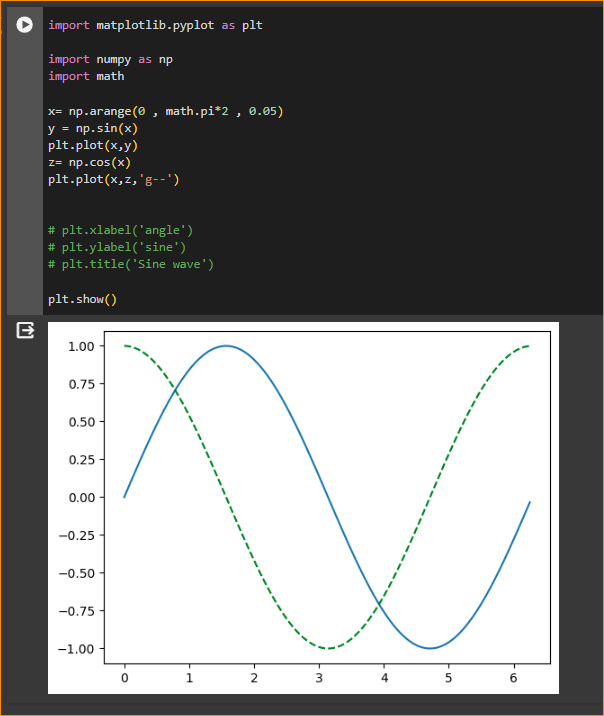
Exercise

Try to plot : 3x2+5x+10



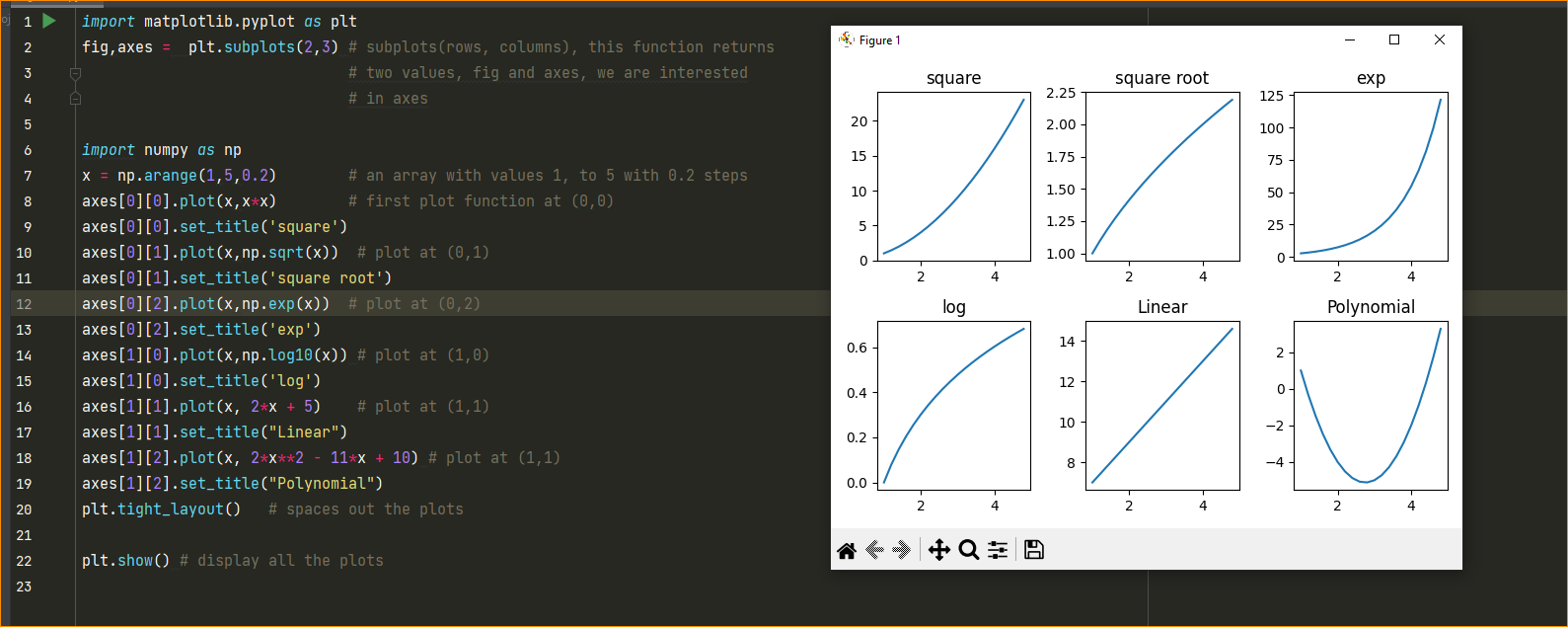
Change Graph Style-

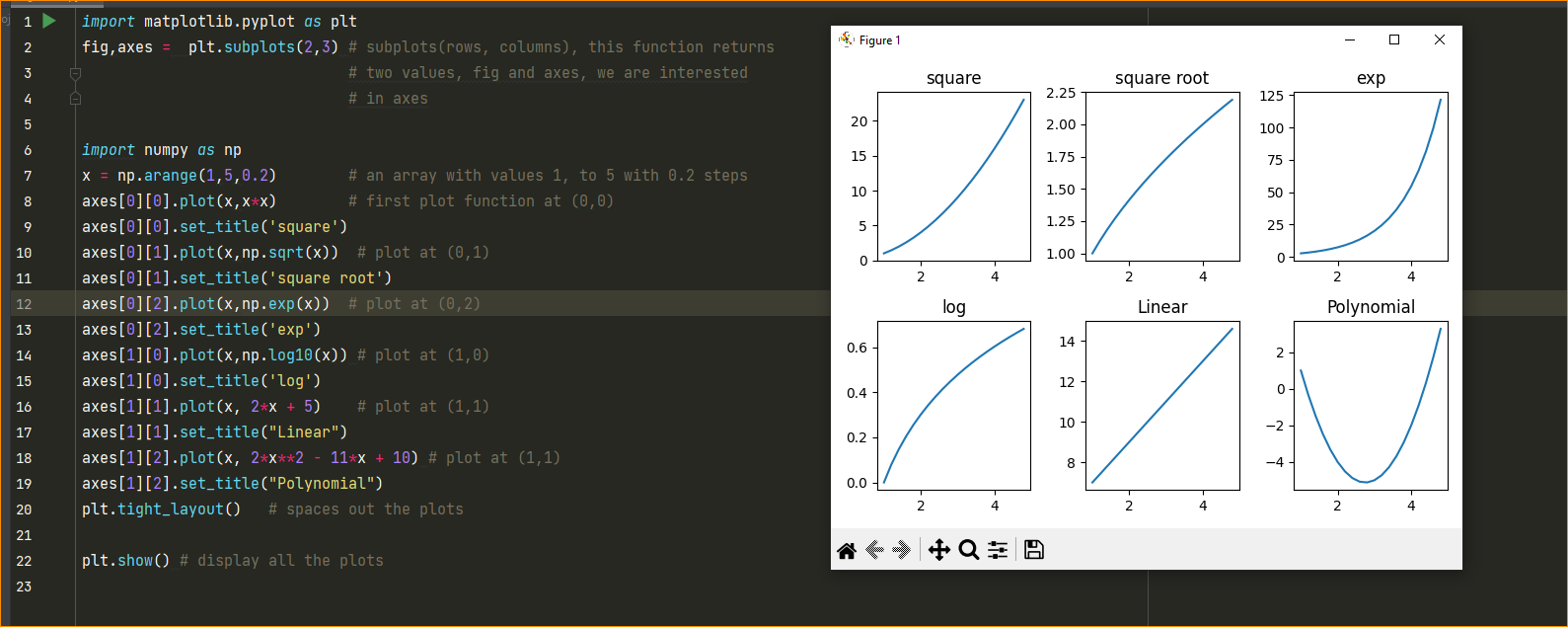


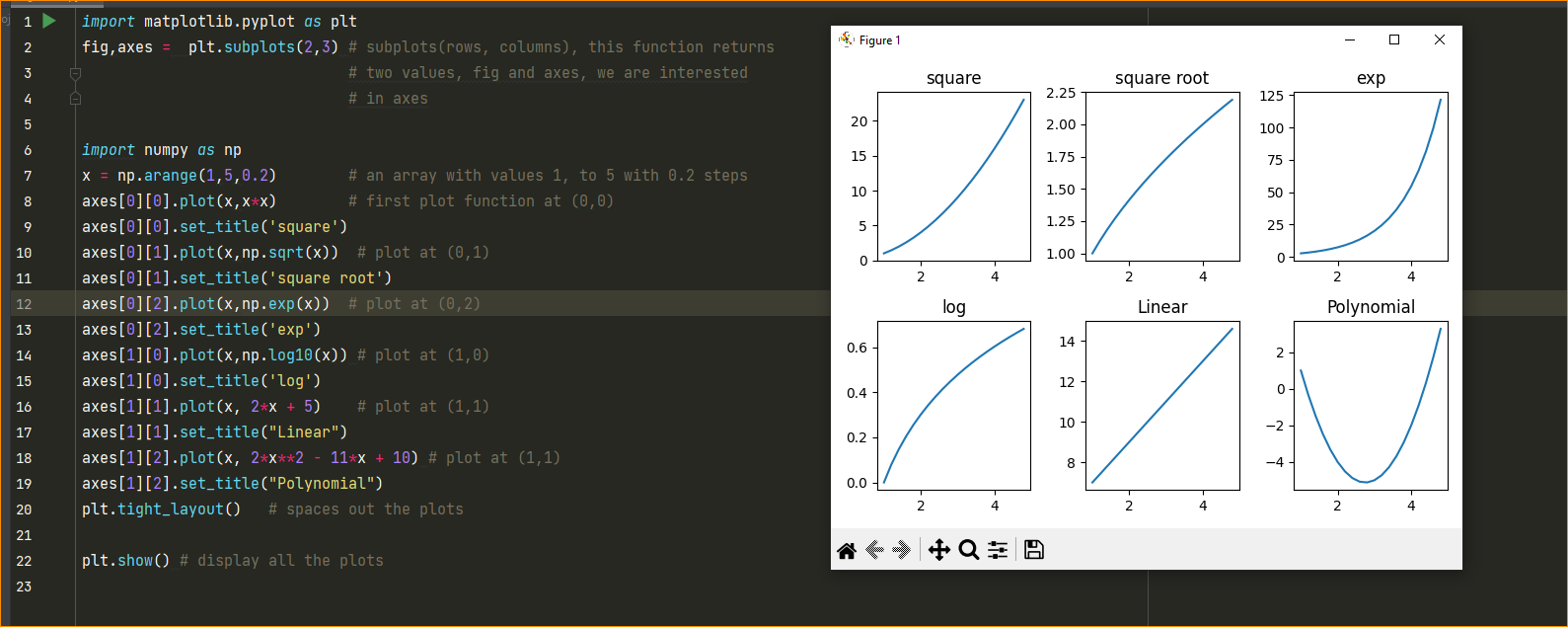


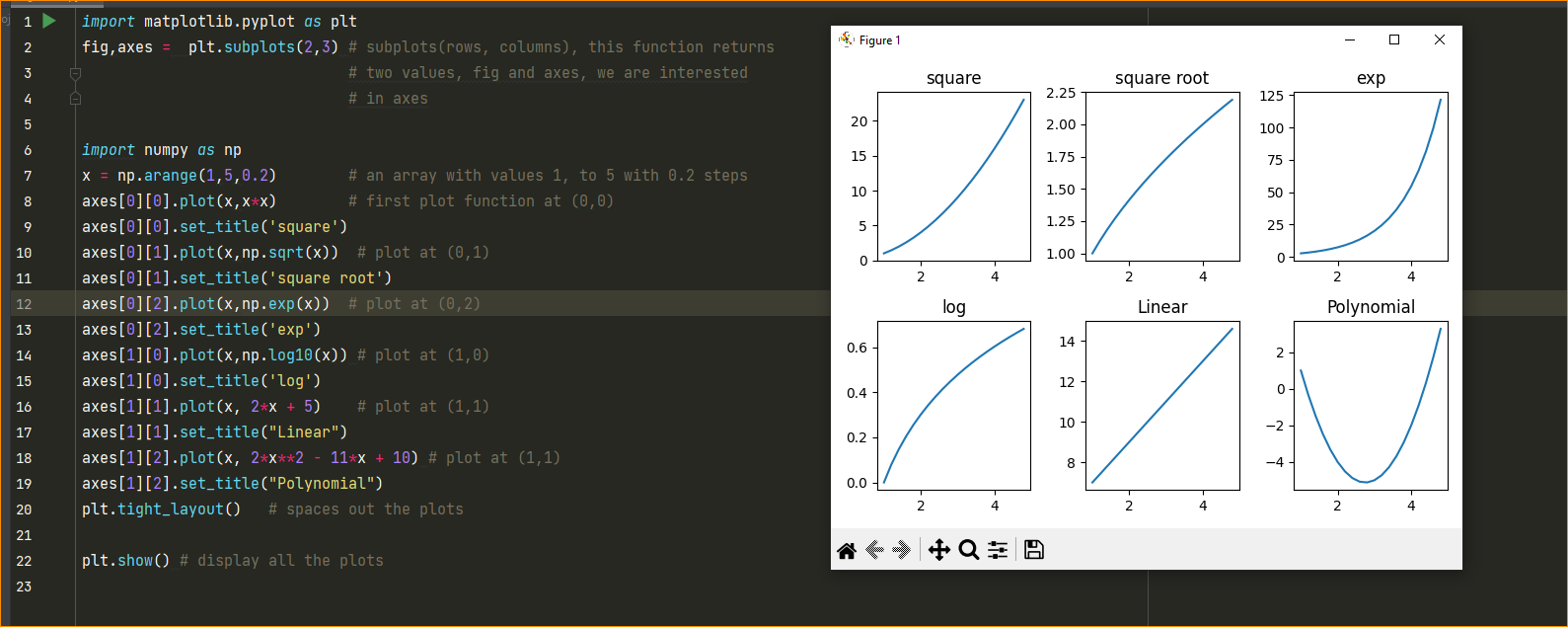
1. Sub Plots

* Matplotlib’s pyplot API has a convenience function called **subplots()** which acts as a utility wrapper and helps in creating common layouts of subplots, including the enclosing figure object, in a single call.
* Plt.subplots(nrows, ncols) The two integer arguments to this function specify the number of rows and columns of the subplot grid. The function returns a figure object and a tuple containing axes objects equal to nrows\*ncols. Each axes object is accessible by its index. Here we create a subplot of 2 rows by 2 columns and display 4 different plots in each subplot.



[0][2] 

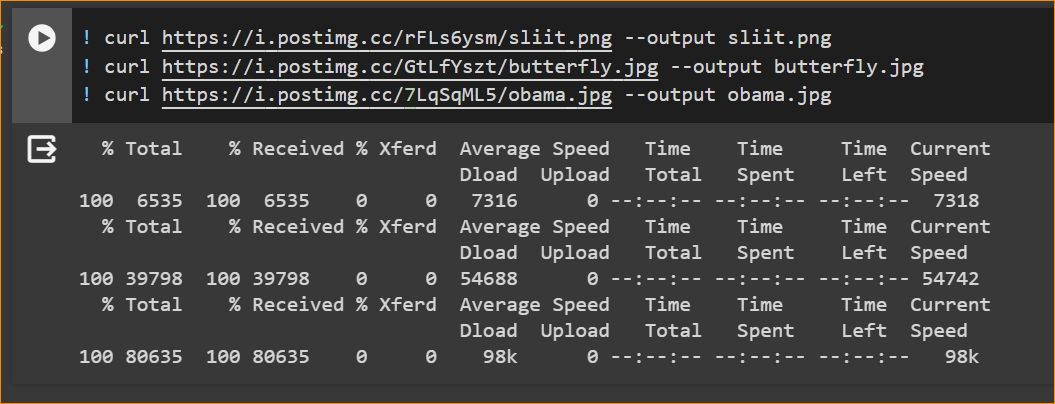
[0][1] 

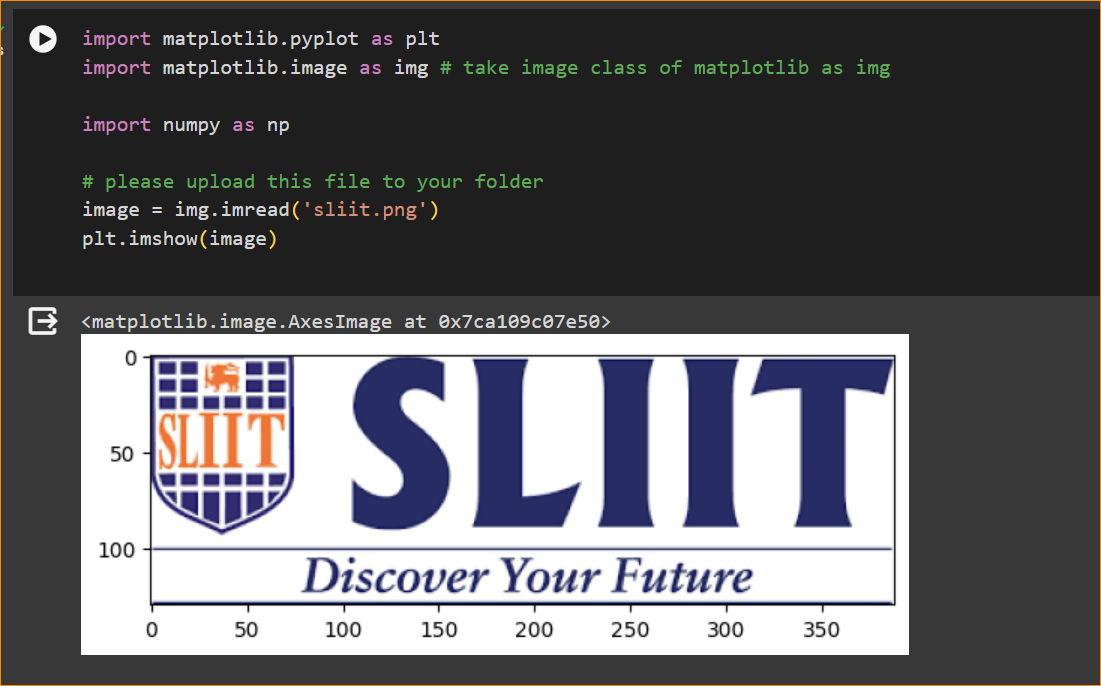
[0][0] 

1. Using Images and Image manipulation

We can use MatPlotLib with numpy to display images. We will use the linux command curl to download some images. We can run any linux command in Google CoLab by using ! mark

**Please run this to download the files used**





Matplotlib – image command

1. Plotting numpy arrays as images

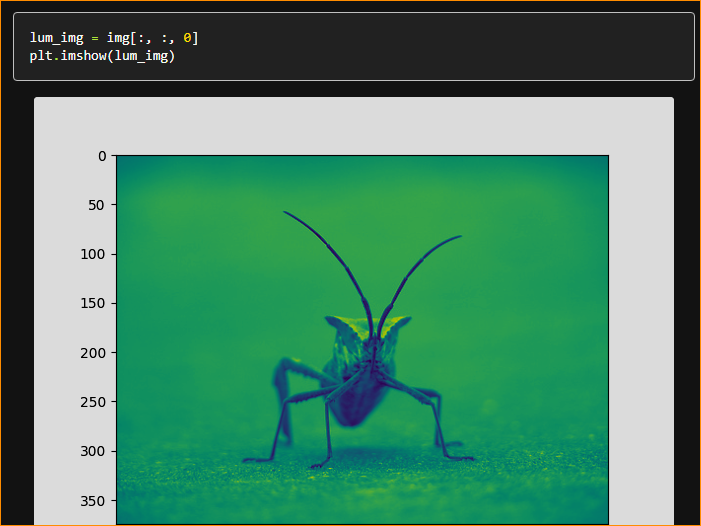
Imshow()



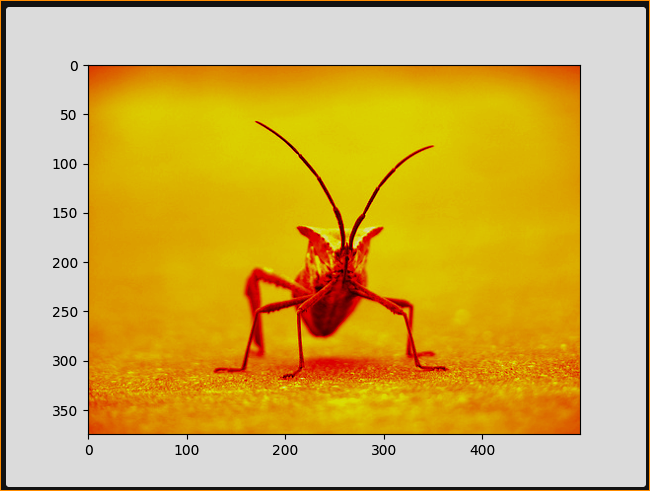
This is performed using the imshow() function. Here we'll grab the plot object. This object gives you an easy way to manipulate the plot from the prompt.

1. Applying pseudocolor schemes to image plots

* Pseudocolor can be a useful tool for enhancing contrast and visualizing your data more easily. This is especially useful when making presentations of your data using projectors - their contrast is typically quite poor.
* Pseudocolor is only relevant to single-channel, grayscale, luminosity images. We currently have an RGB image. Since R, G, and B are all similar (see for yourself above or in your data), we can just pick one channel of our data using array slicing (you can read more in the Numpy tutorial):

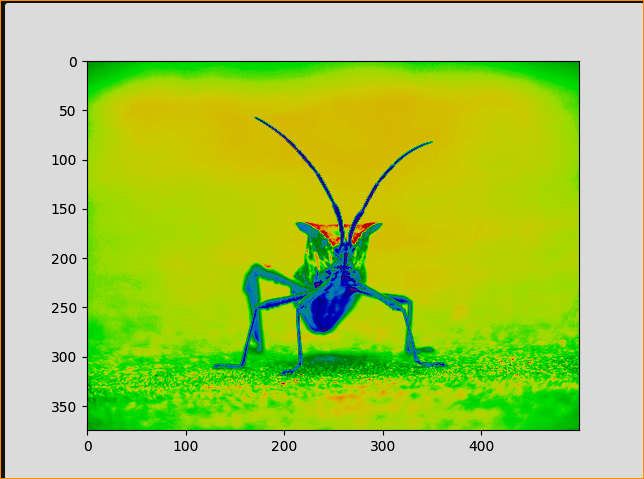


Now, with a luminosity (2D, no color) image, the default colormap (aka lookup table, LUT), is applied. The default is called viridis. There are plenty of others to choose from.



Note that you can also change colormaps on existing plot objects using the set\_cmap() method:

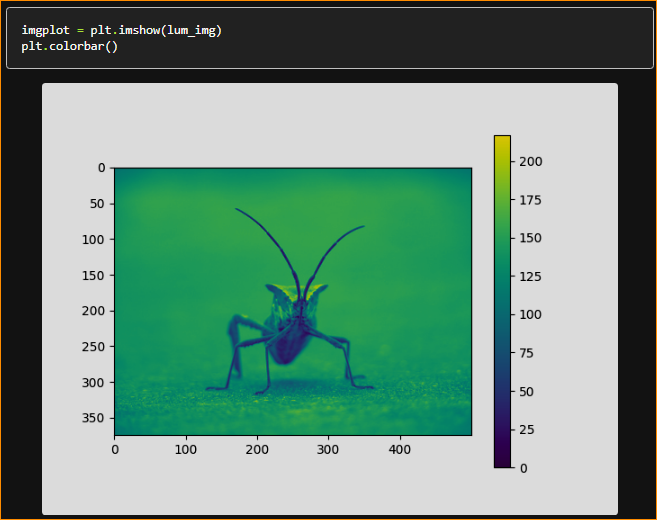




More color maps = <https://matplotlib.org/stable/users/explain/colors/colormaps.html#colormaps>

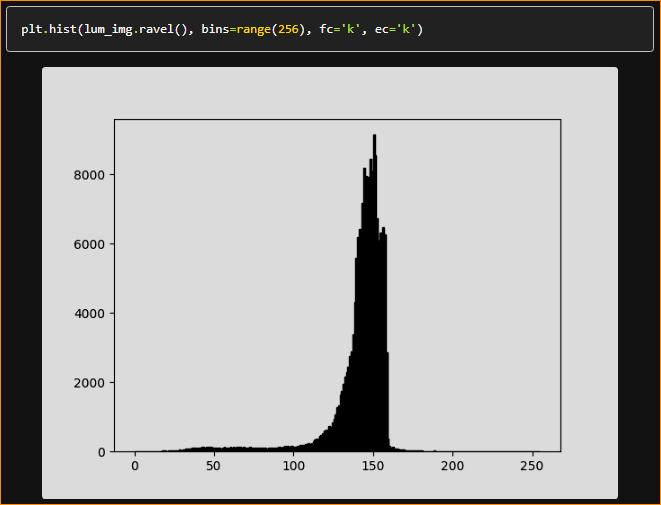
1. Color scale reference

It's helpful to have an idea of what value a color represents. We can do that by adding a color bar to your figure:



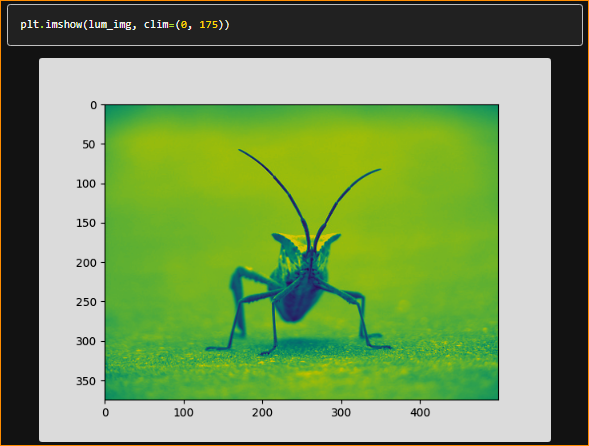
1. Examining a specific data range

Sometimes you want to enhance the contrast in your image, or expand the contrast in a particular region while sacrificing the detail in colors that don't vary much, or don't matter. A good tool to find interesting regions is the histogram. To create a histogram of our image data, we use the hist() function.

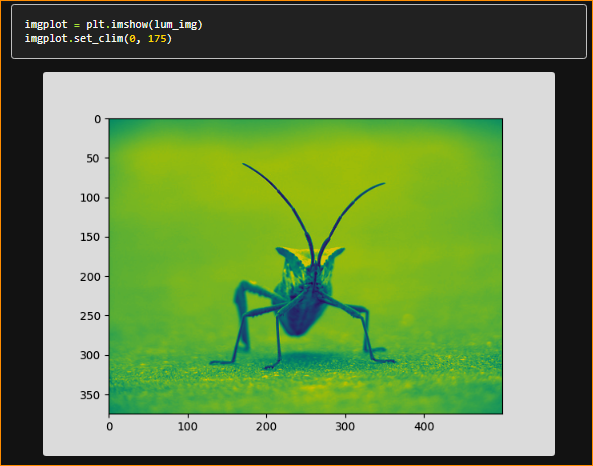


Most often, the "interesting" part of the image is around the peak, and you can get extra contrast by clipping the regions above and/or below the peak. In our histogram, it looks like there's not much useful information in the high end (not many white things in the image). Let's adjust the upper limit, so that we effectively "zoom in on" part of the histogram. We do this by setting clim, the colormap limits.

This can be done by passing a clim keyword argument in the call to imshow.



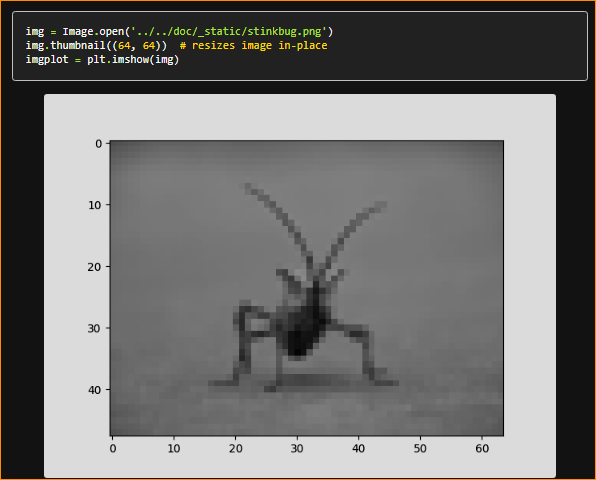
This can also be done by calling the set\_clim() method of the returned image plot object, but make sure that you do so in the same cell as your plot command when working with the Jupyter Notebook - it will not change plots from earlier cells.



1. Array Interpolation schemes

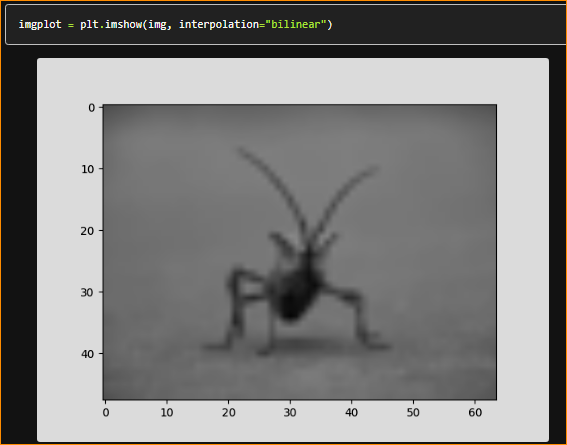
Interpolation calculates what the color or value of a pixel "should" be, according to different mathematical schemes. One common place that this happens is when you resize an image. The number of pixels change, but you want the same information. Since pixels are discrete, there's missing space. Interpolation is how you fill that space. This is why your images sometimes come out looking pixelated when you blow them up. The effect is more pronounced when the difference between the original image and the expanded image is greater. Let's take our image and shrink it. We're effectively discarding pixels, only keeping a select few. Now when we plot it, that data gets blown up to the size on your screen. The old pixels aren't there anymore, and the computer has to draw in pixels to fill that space.

We'll use the Pillow library that we used to load the image also to resize the image.

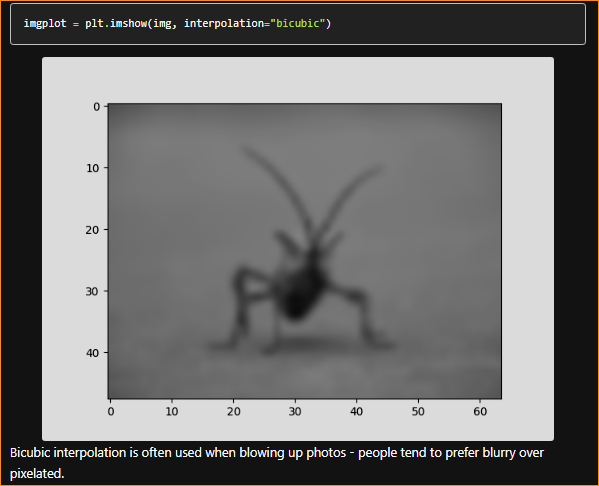


Here we use the default interpolation ("nearest"), since we did not give imshow() any interpolation argument.

Let's try some others. Here's "**bilinear**":



and **bicubic**:



1. contains of this image

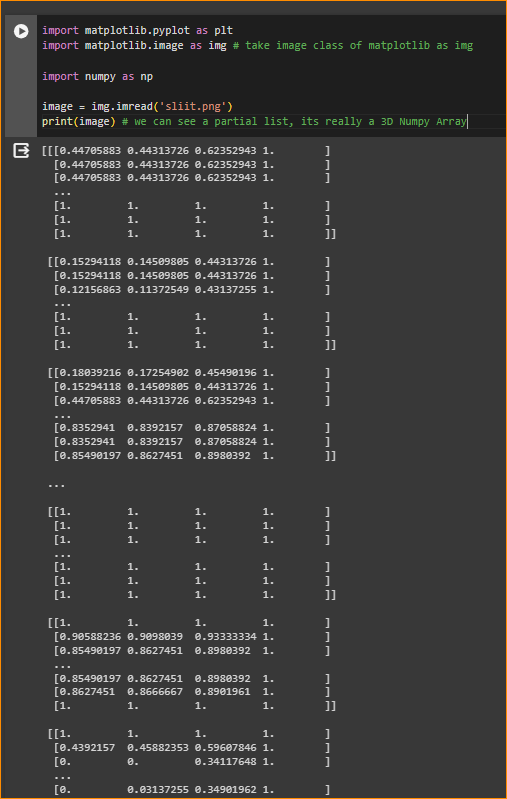
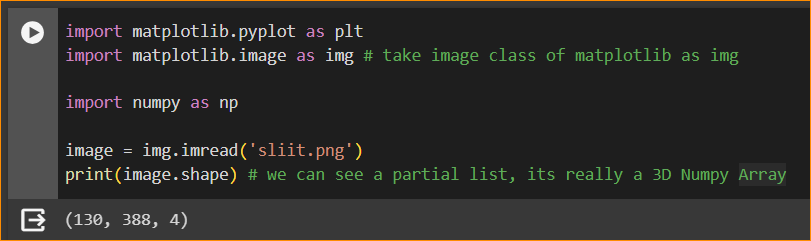


Image shape –



Obama image -



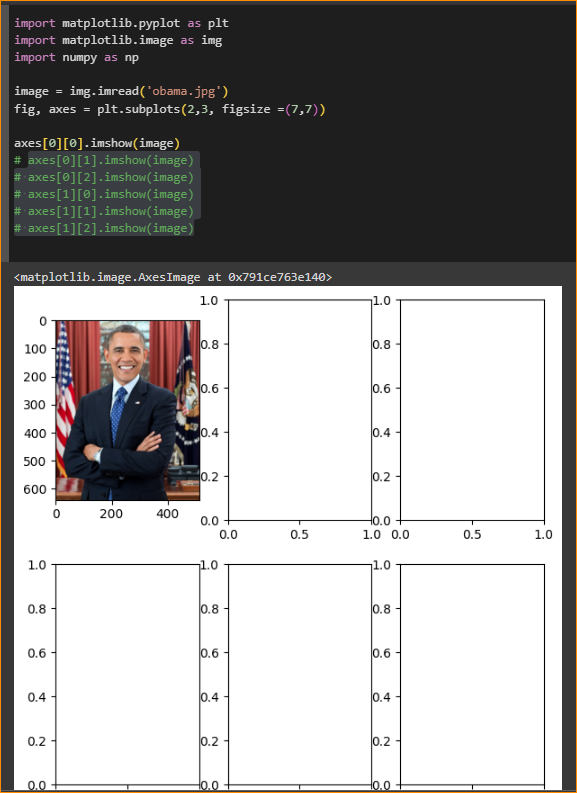
1. Seperating an image to 3 colour channels

What happen in this code line –

using the matplotlib library in Python to create a figure with multiple subplots arranged in a 2x3 grid. The plt.subplots(2, 3, figsize=(7, 7)) line of code is creating a figure with 2 rows and 3 columns of subplots, and setting the overall size of the figure to be 7x7 inches.

After running this code, you'll have a fig variable representing the entire figure, and an axes variable containing a 2x3 array of subplot axes. You can then use these axes to plot your data on the individual subplots.

fig,axes =  plt.subplots(2,3,figsize = (7,7))



**axes[0][0].imshow(image[:,:,0],cmap='Reds') # Red Channel**

* image is assumed to be a 3D array or a multidimensional array representing an image. The [:,:,0] indexing is used to select the first channel of the image (assuming it's a multi-channel image such as an RGB image). This would correspond to the red channel in typical RGB images.
* cmap='Reds' sets the colormap for the image. In this case, it's set to 'Reds', which means the lower values will be displayed with lighter red colors, and higher values with darker red colors.



1. Georgie Tech Face Database - Looking at a set of images

When you work with machine learning you work with a lot of data, this database is over 100 MB large and has seperate folders containing 15 images of each person.

Let's try to see images in one of the folders from the Georgia dataset

