Matplotlib

|

Assistant Professor :

Dr. Fadaeieslam </h2>

By :

Amir Shokri

Farshad Asgharzade

**Alireza Gholamnia
</h1>**

Introduction:

Matplotlib is one of the most popular Python packages used for data visualization. Matplotlib was originally written by John D. Hunter in 2003. Pyplot is a Matplotlib module which provides a MATLAB-like interface. Matplotlib is designed to be as usable as MATLAB, with the ability to use Python, and the advantage of being free and open-source. Matplotlib is open source and we can use it freely. The purpose of a plotting package is to assist you in visualizing your data as easily as possible, with all the necessary control that is, by using relatively high-level commands most of the time, and still have the ability to use the low-level commands when needed.

Installation of Matplotlib:

If you have Python and PIP already installed on a system, then installation of Matplotlib is very easy. Install it using this command:

```
pip install matplotlib
```

Matplotlib requires the following dependencies:

- NumPy
- setuptools
- cycler
- dateutil
- kiwisolver
- Pillow
- pyparsing

Anaconda distribution:

Anaconda is a free and open source distribution of the Python and R programming languages for large-scale data processing, predictive analytics, and scientific computing. The distribution makes package management and deployment simple and easy. Matplotlib and lots of other useful (data) science tools form part of the distribution. Package versions are managed by the package management system Conda. The advantage of Anaconda is that you have access to over 720 packages that can easily be installed with Anaconda's Conda, a package, dependency, and environment manager.

Checking Matplotlib Version:

Once Matplotlib is installed, import it in your applications by adding the import module statement:

```
In [74]: import matplotlib
         matplotlib.__version__
```

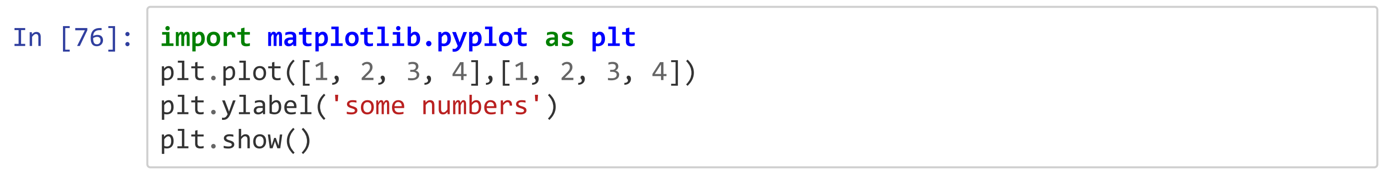
```
Out[74]: '3.1.3'
```

Most of the Matplotlib utilities lies under the pyplot submodule, and are usually imported under the plt alias:

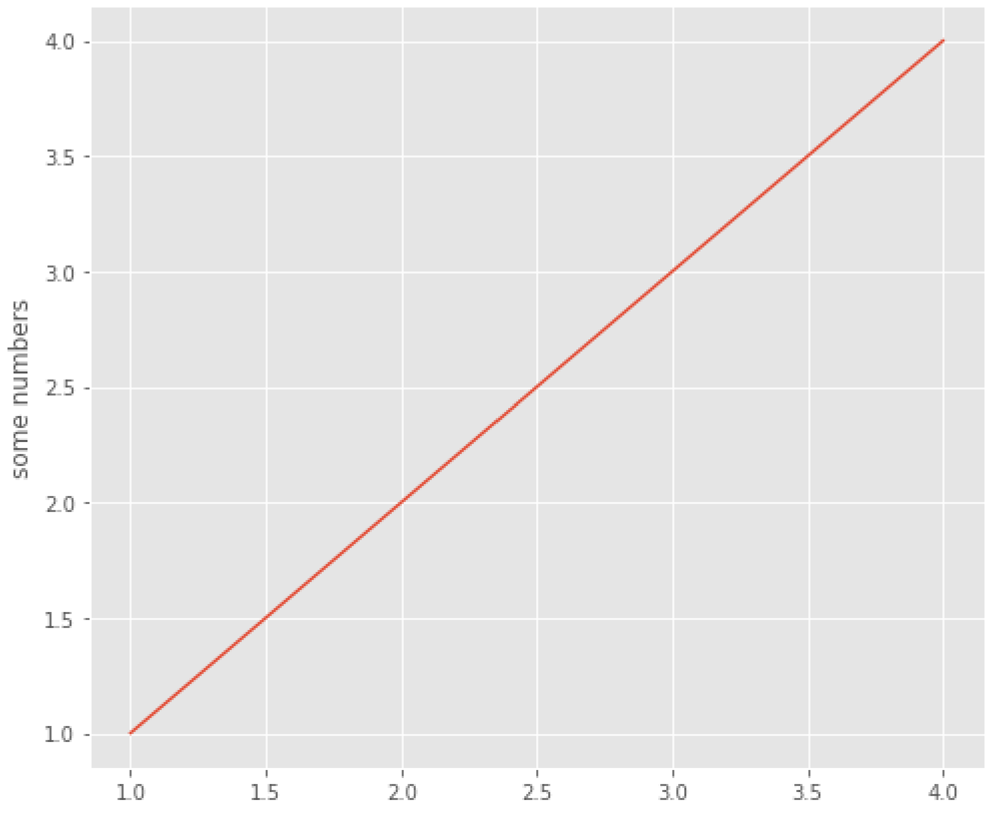
```
In [75]: import matplotlib.pyplot as plt
```

matplotlib.pyplot is a collection of functions that make matplotlib work like MATLAB. Each pyplot function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc.

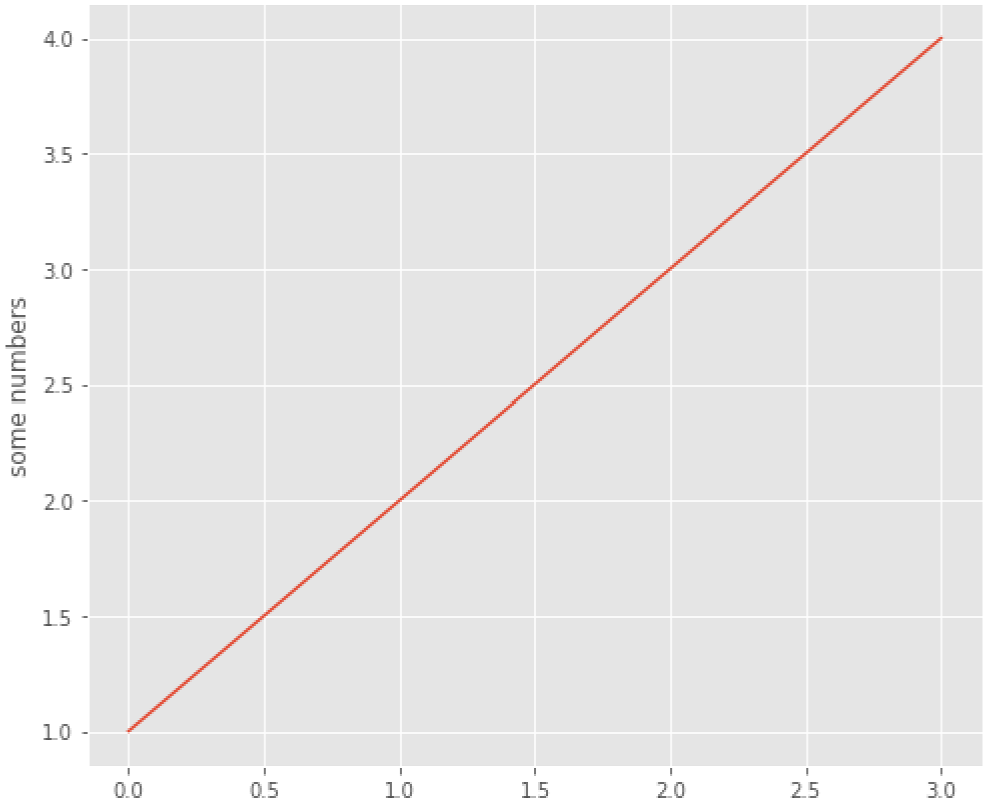
Generating visualizations with pyplot is very quick:



```
In [76]: import matplotlib.pyplot as plt
         plt.plot([1, 2, 3, 4],[1, 2, 3, 4])
         plt.ylabel('some numbers')
         plt.show()
```



```
In [77]: plt.plot([1, 2, 3, 4])
         plt.ylabel('some numbers')
         plt.show()
```

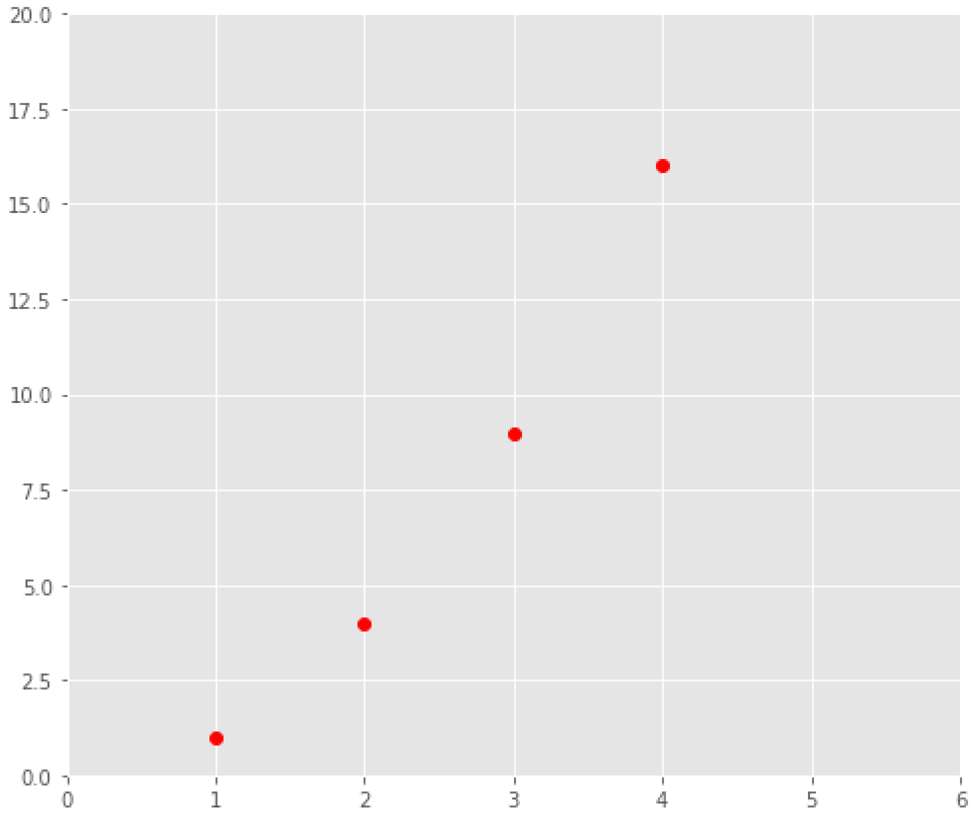


You may be wondering why the x-axis ranges from 0-3 and the y-axis from 1-4. If you provide a single list or array to plot, matplotlib assumes it is a sequence of y values, and automatically generates the x values for you. Since python ranges start with 0, the default x vector has the same length as y but starts with 0. Hence the x data are [0, 1, 2, 3].

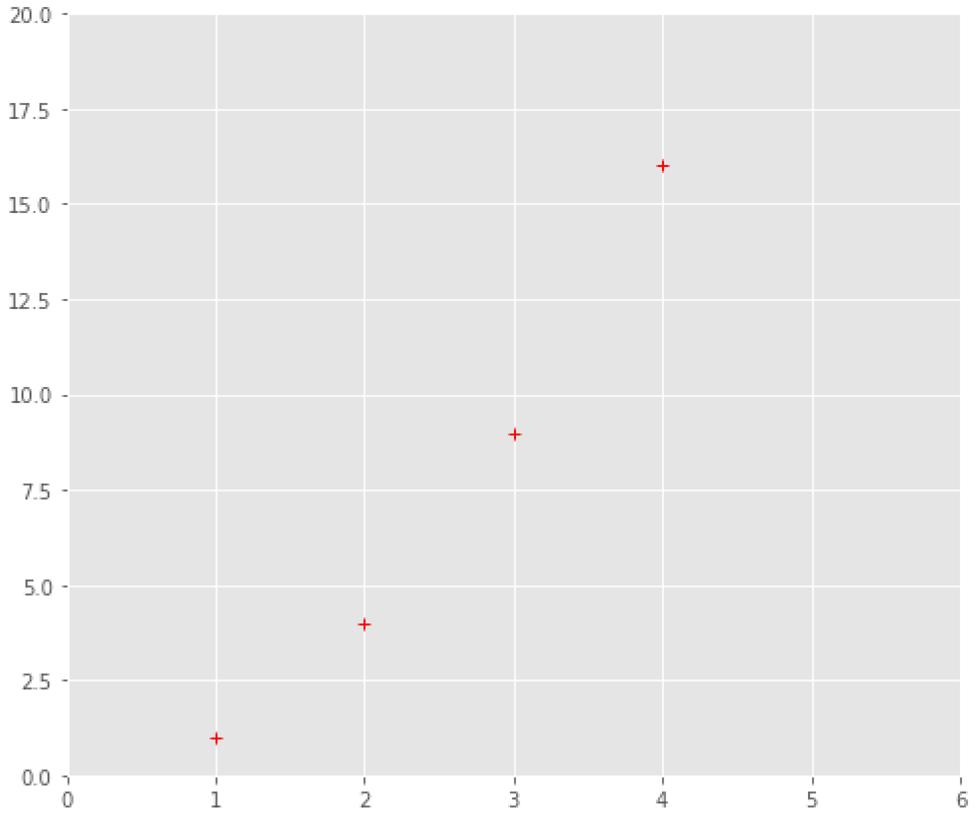
Formatting the style of your plot:

For every x, y pair of arguments, there is an optional third argument which is the format string that indicates the color and line type of the plot. The letters and symbols of the format string are from MATLAB, and you concatenate a color string with a line style string. The default format string is 'b-', which is a solid blue line. For example, to plot the above with red circles, you would issue

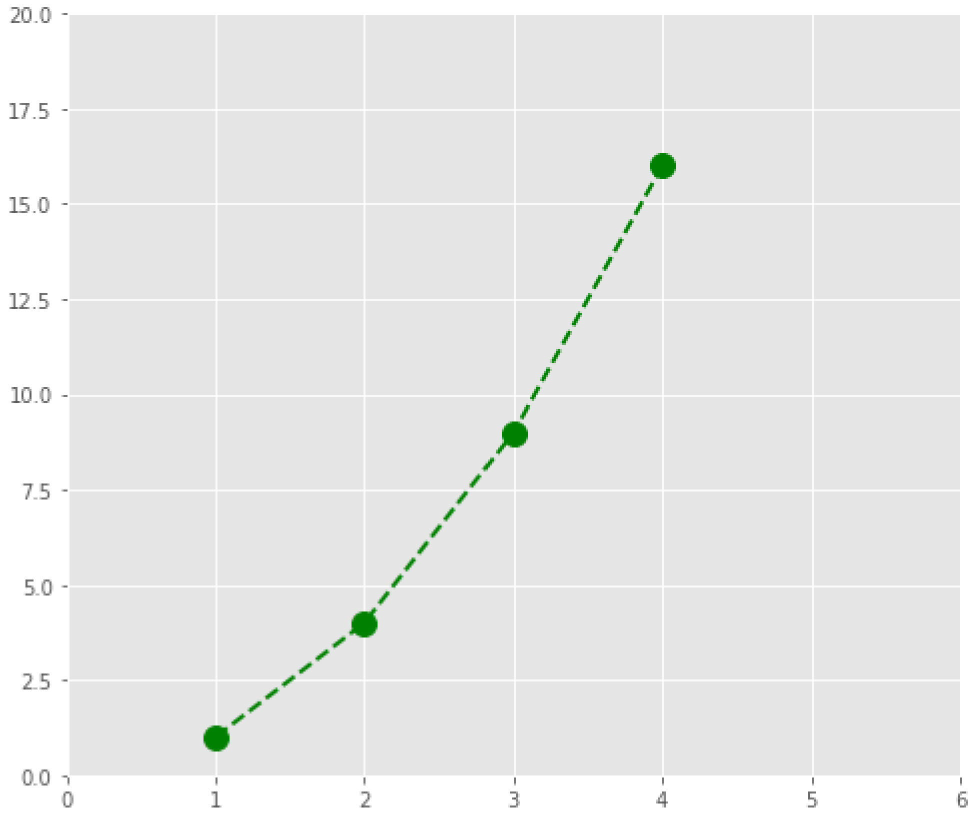
In [78]:
```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'ro')
plt.axis([0, 6, 0, 20])
plt.show()
```



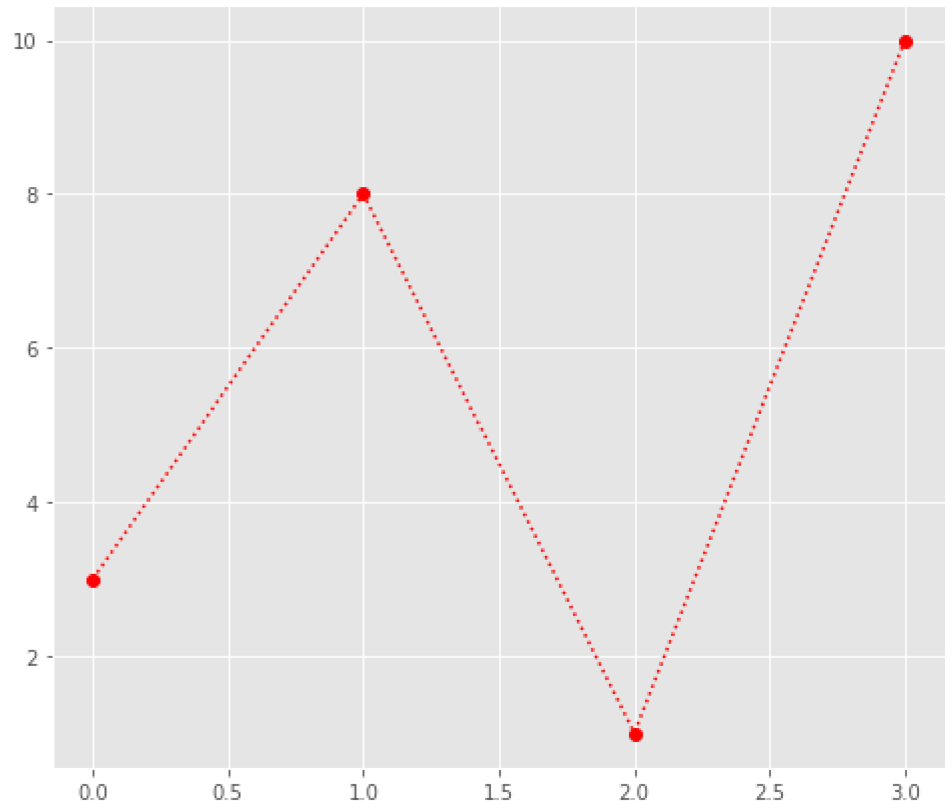
```
In [79]: plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'r+')     # ditto, but with red plusses
         plt.axis([0, 6, 0, 20])
         plt.show()
```



In [80]:
```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16],'go--', linewidth=2, markersize=12)
plt.axis([0, 6, 0, 20])
plt.show()
```

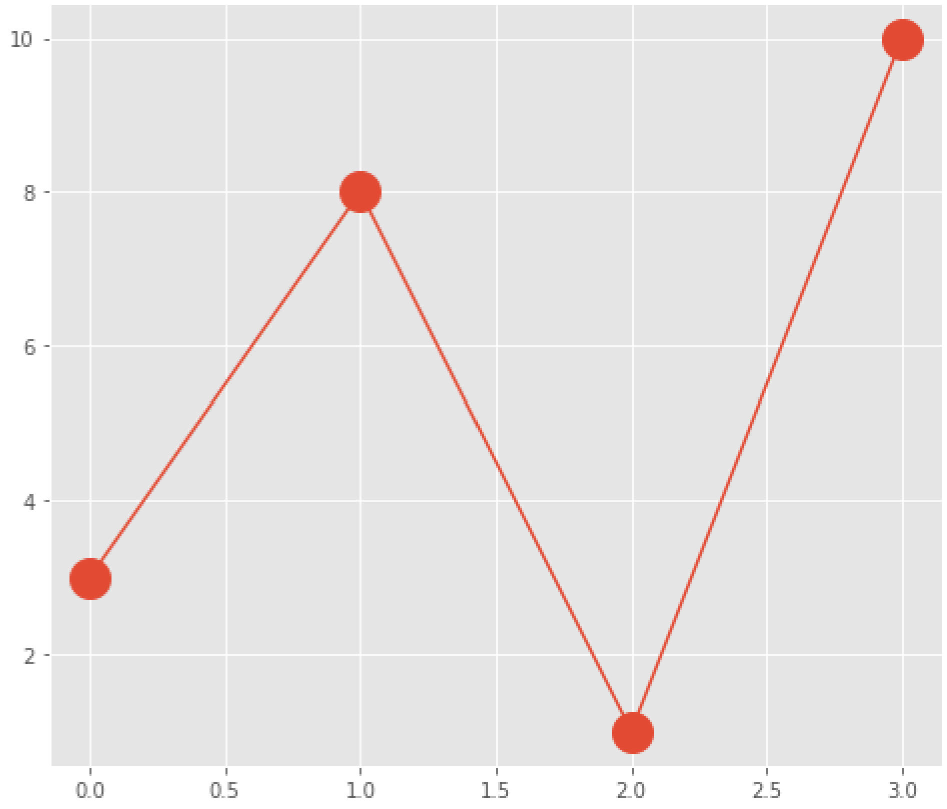


In [81]:
```python
ypoints = [3, 8, 1, 10]
plt.plot(ypoints, 'o:r')
plt.show()
```



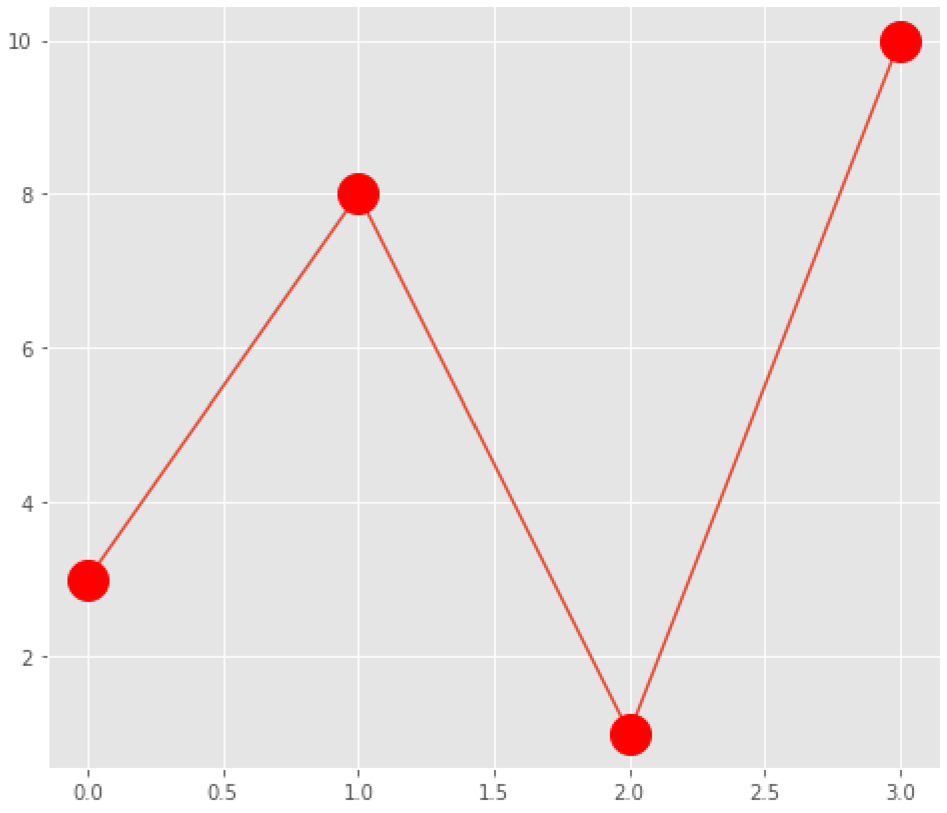
Marker Size : You can use the keyword argument markersize or the shorter version, ms to set the size of the markers

```
In [82]: ypoints = [3, 8, 1, 10]
         plt.plot(ypoints, marker = 'o', ms = 20)
         plt.show()
```



Marker Color: You can use the keyword argument markerfacecolor or the shorter mfc to set the color inside the edge of the markers:

In [83]:
```
ypoints =[3, 8, 1, 10]
plt.plot(ypoints, marker = 'o', ms = 20, mec = 'r', mfc = 'r')
plt.show()
```



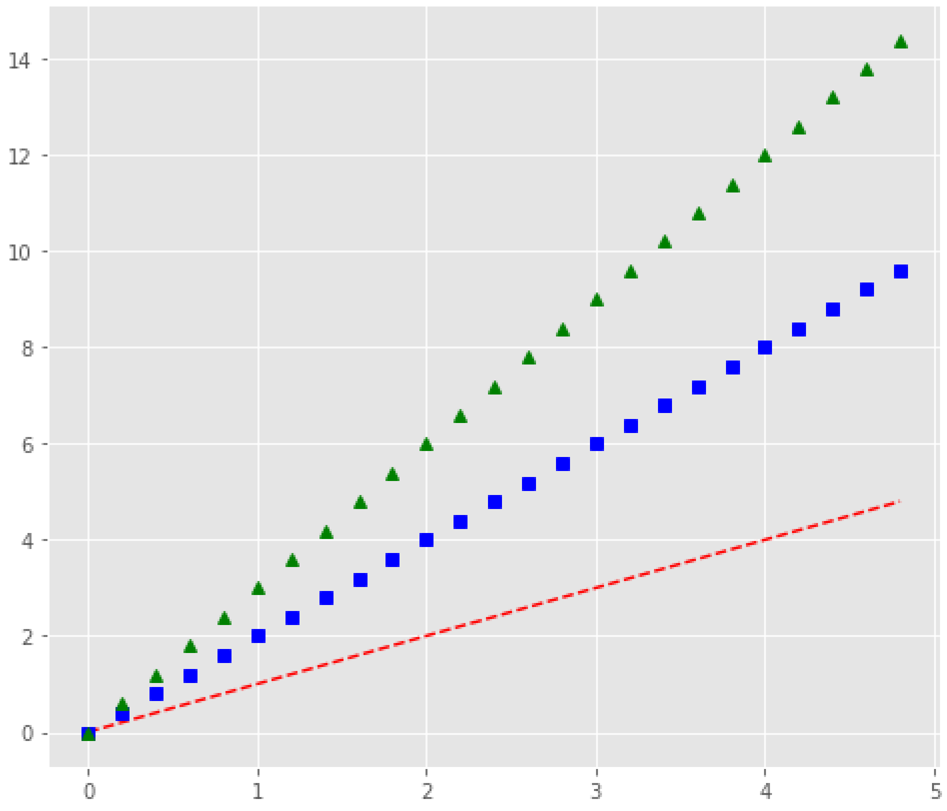
For more visit

https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot (https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot)

Use with numpy

If matplotlib were limited to working with lists, it would be fairly useless for numeric processing. Generally, you will use numpy arrays. In fact, all sequences are converted to numpy arrays internally. The example below illustrates plotting several lines with different format styles in one function call using arrays.

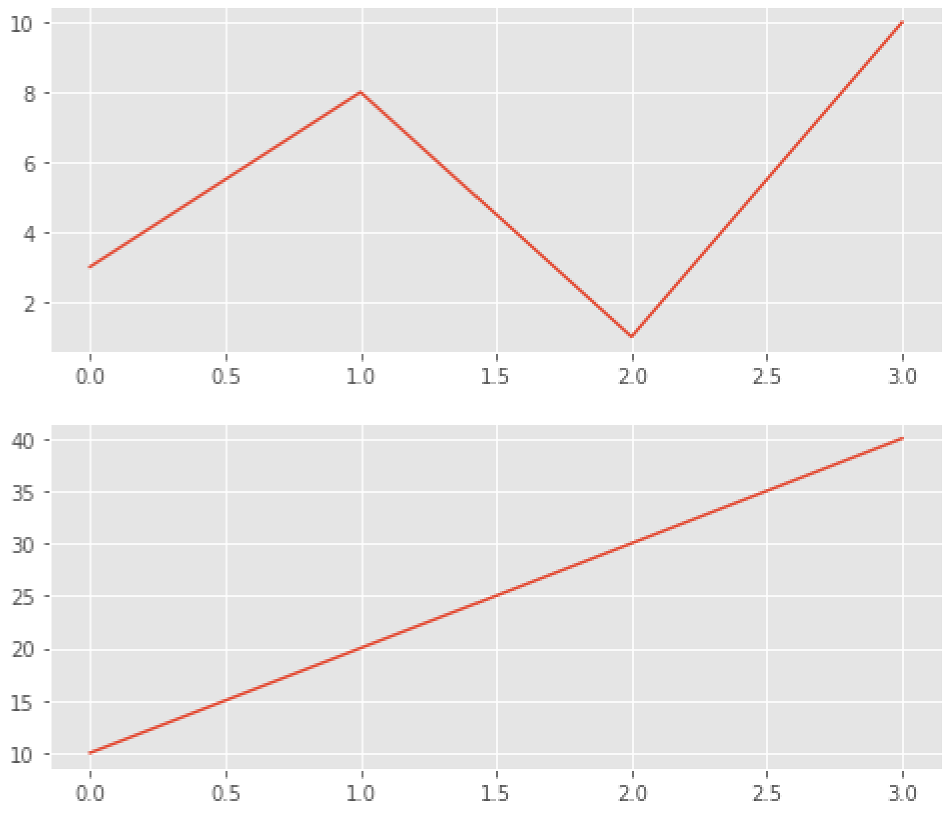
In [84]:
```python
import numpy as np
import matplotlib.pyplot as plt

# evenly sampled time at 200ms intervals
t = np.arange(0., 5., 0.2)

# red dashes, blue squares and green triangles
plt.plot(t, t, 'r--', t, t*2, 'bs', t, t*3, 'g^')
plt.show()
```



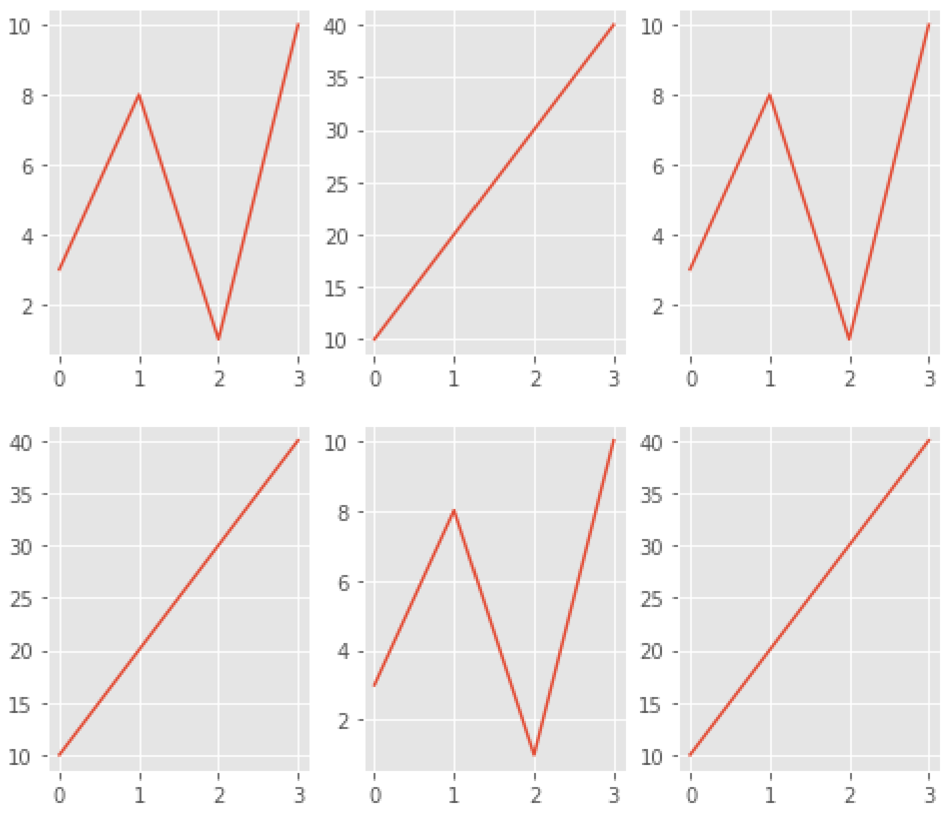
Display Multiple Plots

the subplots() function takes three arguments that describes the layout of the figure. The layout is organized in rows and columns, which are represented by the first and second argument. The third argument represents the index of the current plot.

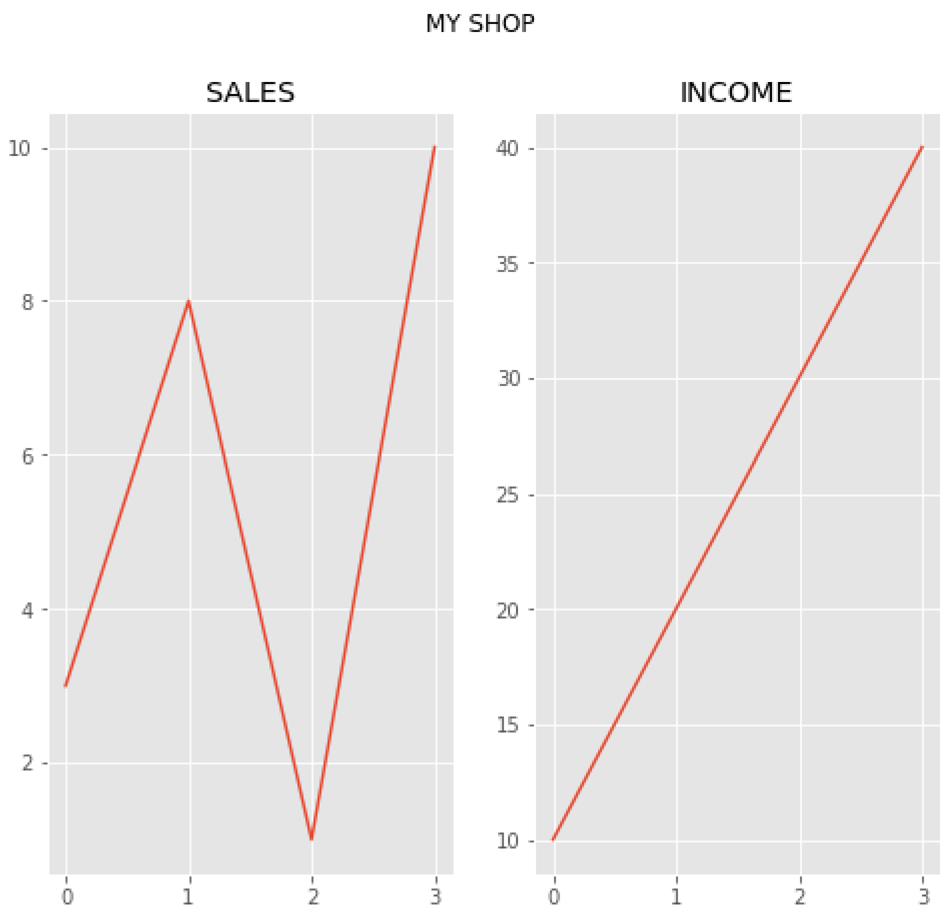
```
In [85]: x = np.array([0, 1, 2, 3])
         y = np.array([3, 8, 1, 10])
         plt.subplot(2, 1, 1)
         plt.plot(x,y)
         x = np.array([0, 1, 2, 3])
         y = np.array([10, 20, 30, 40])
         plt.subplot(2, 1, 2)
         plt.plot(x,y)
         plt.show()
```



In [86]:
```
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 3, 1)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 3, 2)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 3, 3)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 3, 4)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 3, 5)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 3, 6)
plt.plot(x,y)
plt.show()
```


0 1 2 3
2 4 6 8 10
10 15 20 25 30 35 40

In [87]:
```
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(1, 2, 1)
plt.plot(x,y)
plt.title("SALES")

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(1, 2, 2)
plt.plot(x,y)
plt.title("INCOME")

plt.suptitle("MY SHOP")
plt.show()
```

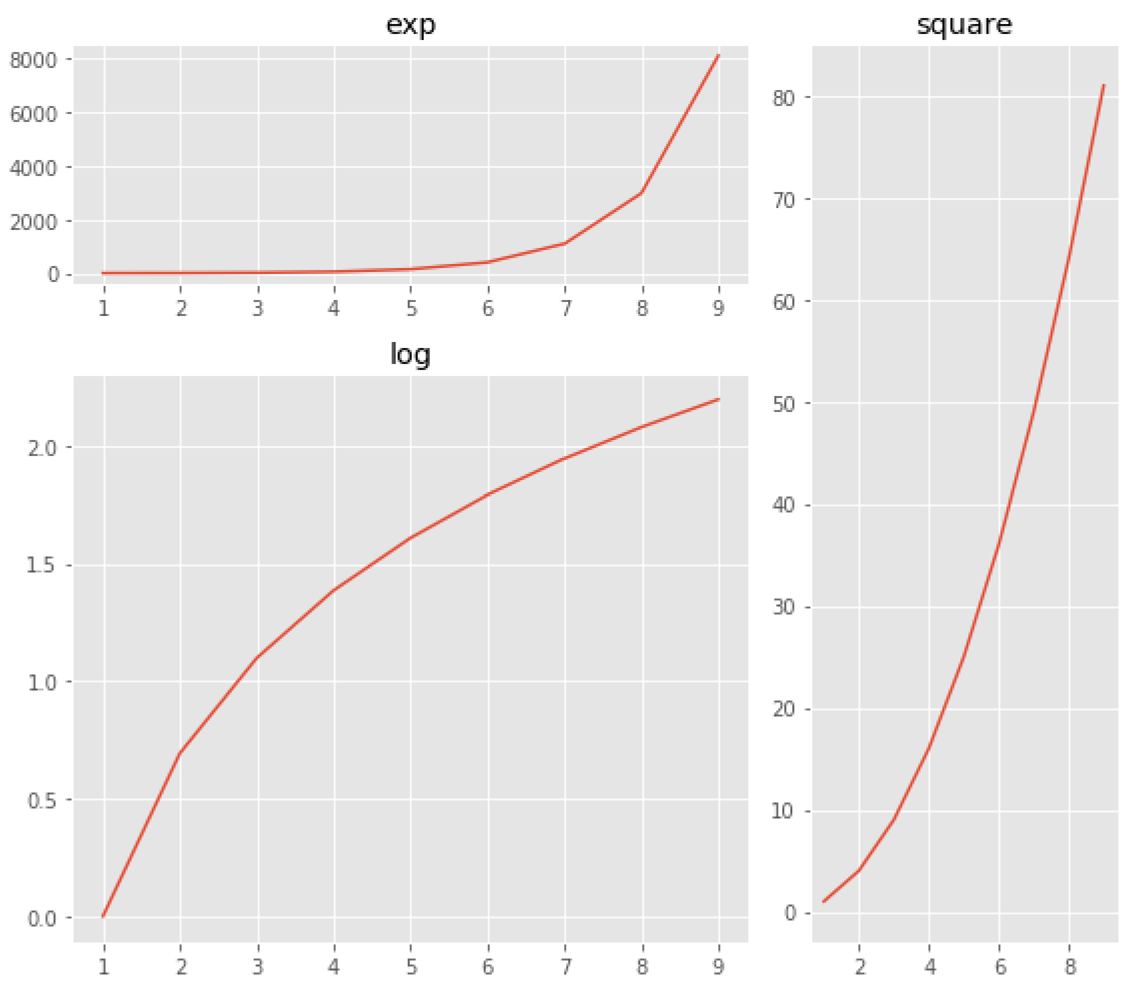


Subplot2grid() Function

This function gives more flexibility in creating an axes object at a specific location of the grid. It also allows the axes object to be spanned across multiple rows or columns.

```
Plt.subplot2grid(shape, location, rowspan, colspan)
```

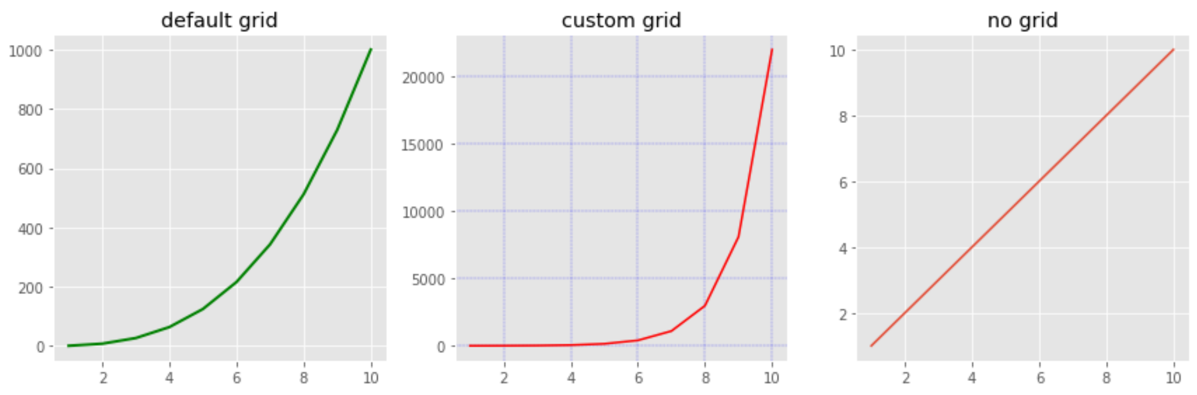
In [88]:
```python
a1 = plt.subplot2grid((3,3),(0,0),colspan = 2)
a2 = plt.subplot2grid((3,3),(0,2), rowspan = 3)
a3 = plt.subplot2grid((3,3),(1,0),rowspan = 2, colspan = 2)


x = np.arange(1,10)
a2.plot(x, x*x)
a2.set_title('square')
a1.plot(x, np.exp(x))
a1.set_title('exp')
a3.plot(x, np.log(x))
a3.set_title('log')
plt.tight_layout()
plt.show()
```



Grids

The grid() function of axes object sets visibility of grid inside the figure to on or off. You can also display major / minor (or both) ticks of the grid. Additionally color, linestyle and linewidth properties can be set in the grid() function.

```
In [89]: import matplotlib.pyplot as plt
         import numpy as np
         fig, axes = plt.subplots(1,3, figsize = (12,4))
         x = np.arange(1,11)
         axes[0].plot(x, x**3, 'g',lw=2)
         axes[0].grid(True)
         axes[0].set_title('default grid')
         axes[1].plot(x, np.exp(x), 'r')
         axes[1].grid(color='b', ls = '-.', lw = 0.25)
         axes[1].set_title('custom grid')
         axes[2].plot(x,x)
         axes[2].set_title('no grid')
         fig.tight_layout()
         plt.show()
```



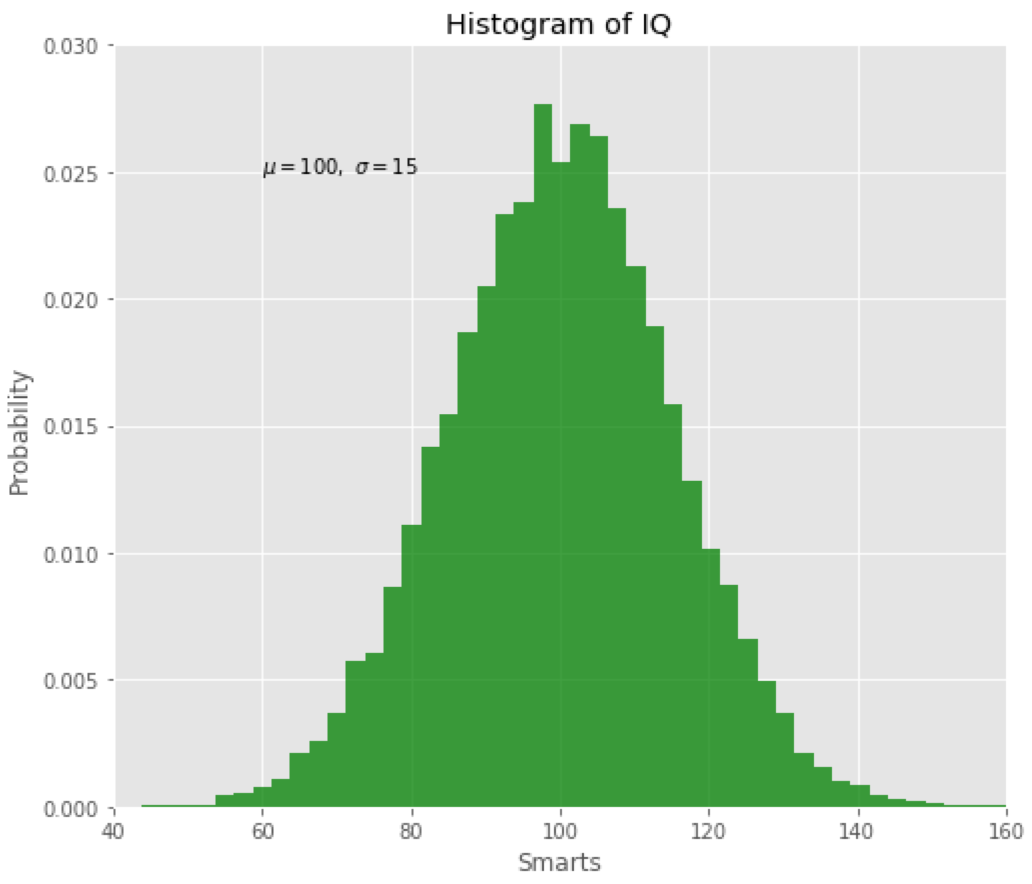
Working with text

text can be used to add text in an arbitrary location, and xlabel, ylabel and title are used to add text in the indicated locations (see Text in Matplotlib Plots for a more detailed example) Using mathematical expressions in text:

```
plt.text(60, .025, r'$\mu=100,\ \sigma=15$')
```

matplotlib accepts laTeX equation expressions in any text expression.

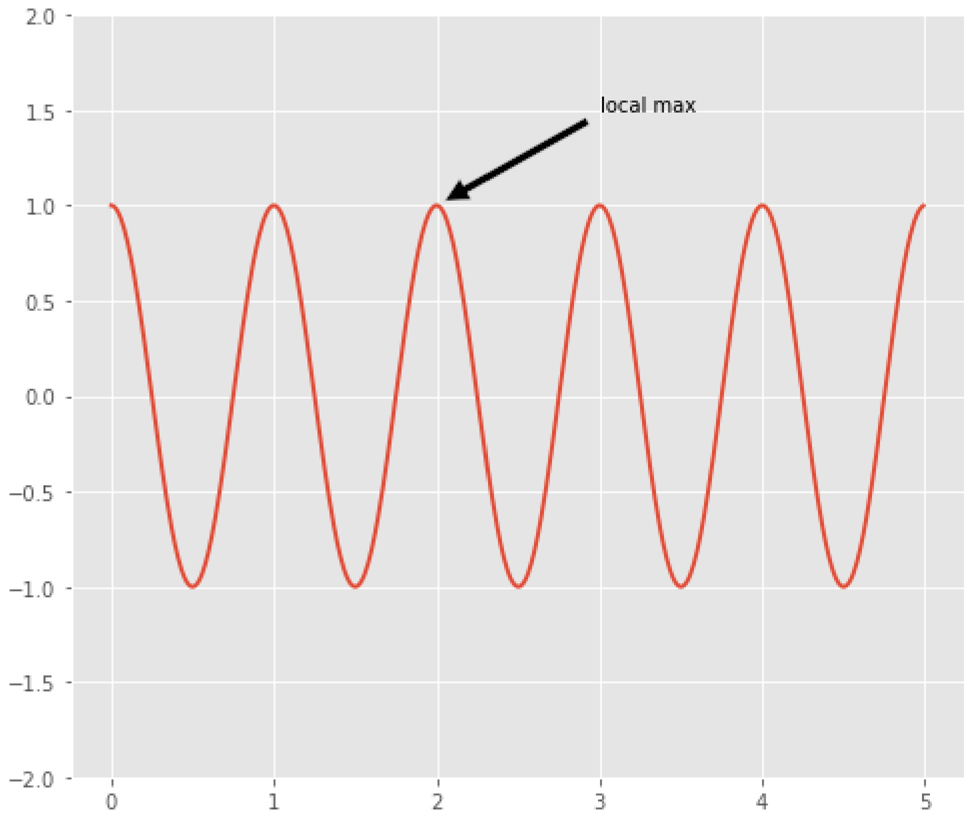
In [90]:
```
mu, sigma = 100, 15
x = mu + sigma * np.random.randn(10000)

# the histogram of the data
n, bins, patches = plt.hist(x, 50, density=1, facecolor='g', alpha=0.75)


plt.xlabel('Smarts')
plt.ylabel('Probability')
plt.title('Histogram of IQ')
plt.text(60, .025, r'$\mu=100,\ \sigma=15$')
plt.axis([40, 160, 0, 0.03])
plt.grid(True)
plt.show()
```



Annotating text

The uses of the basic text function above place text at an arbitrary position on the Axes. A common use for text is to annotate some feature of the plot, and the annotate method provides helper functionality to make annotations easy. In an annotation, there are two points to consider: the location being annotated represented by the argument xy and the location of the text xytext. Both of these arguments are (x, y) tuples.

In [91]:
```
ax = plt.subplot(111)
t = np.arange(0.0, 5.0, 0.01)
s = np.cos(2*np.pi*t)
line, = plt.plot(t, s, lw=2)

plt.annotate('local max', xy=(2, 1), xytext=(3, 1.5),
             arrowprops=dict(facecolor='black', shrink=0.05),
             )

plt.ylim(-2, 2)
plt.show()
```

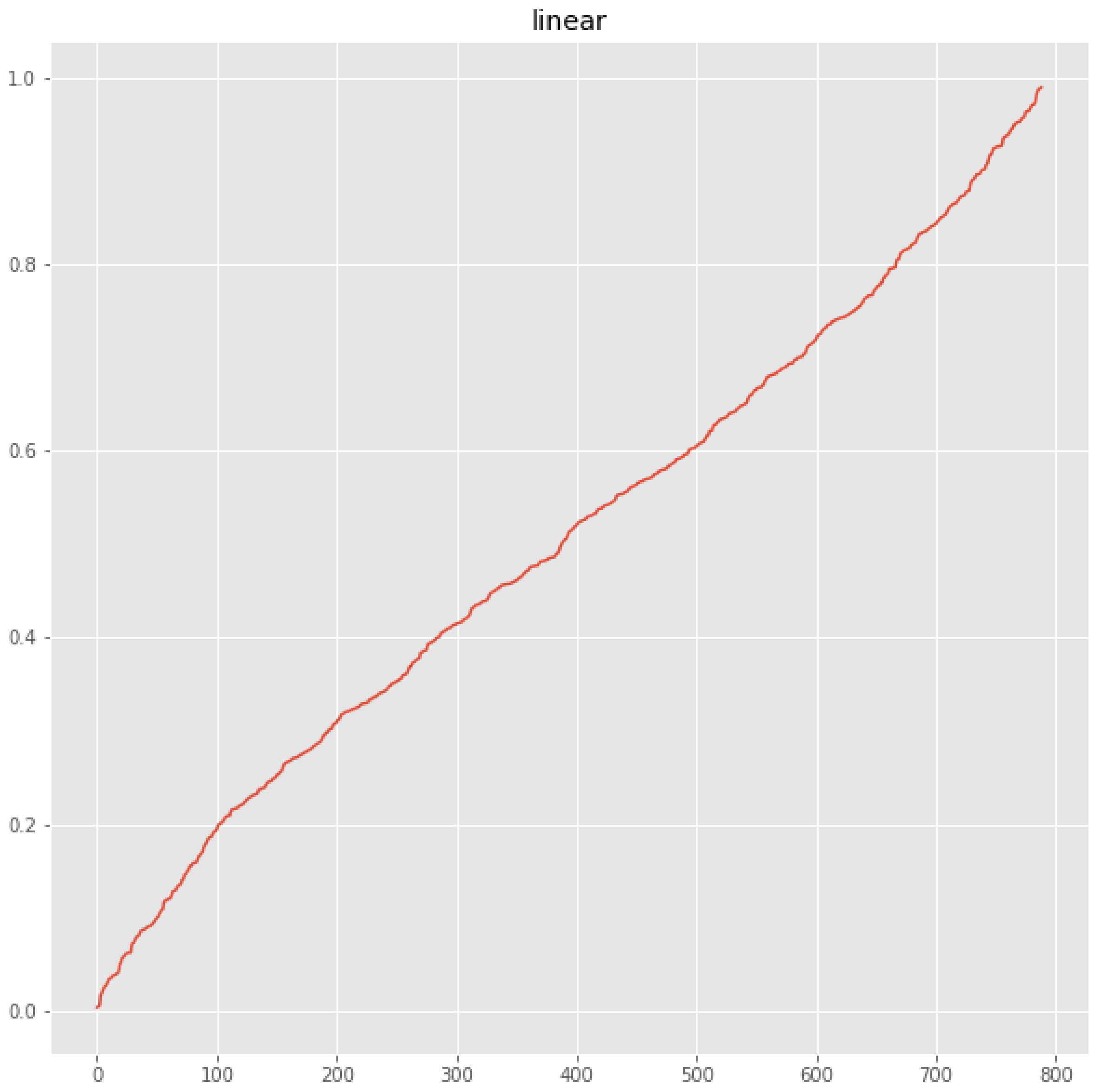


Logarithmic and other nonlinear axes

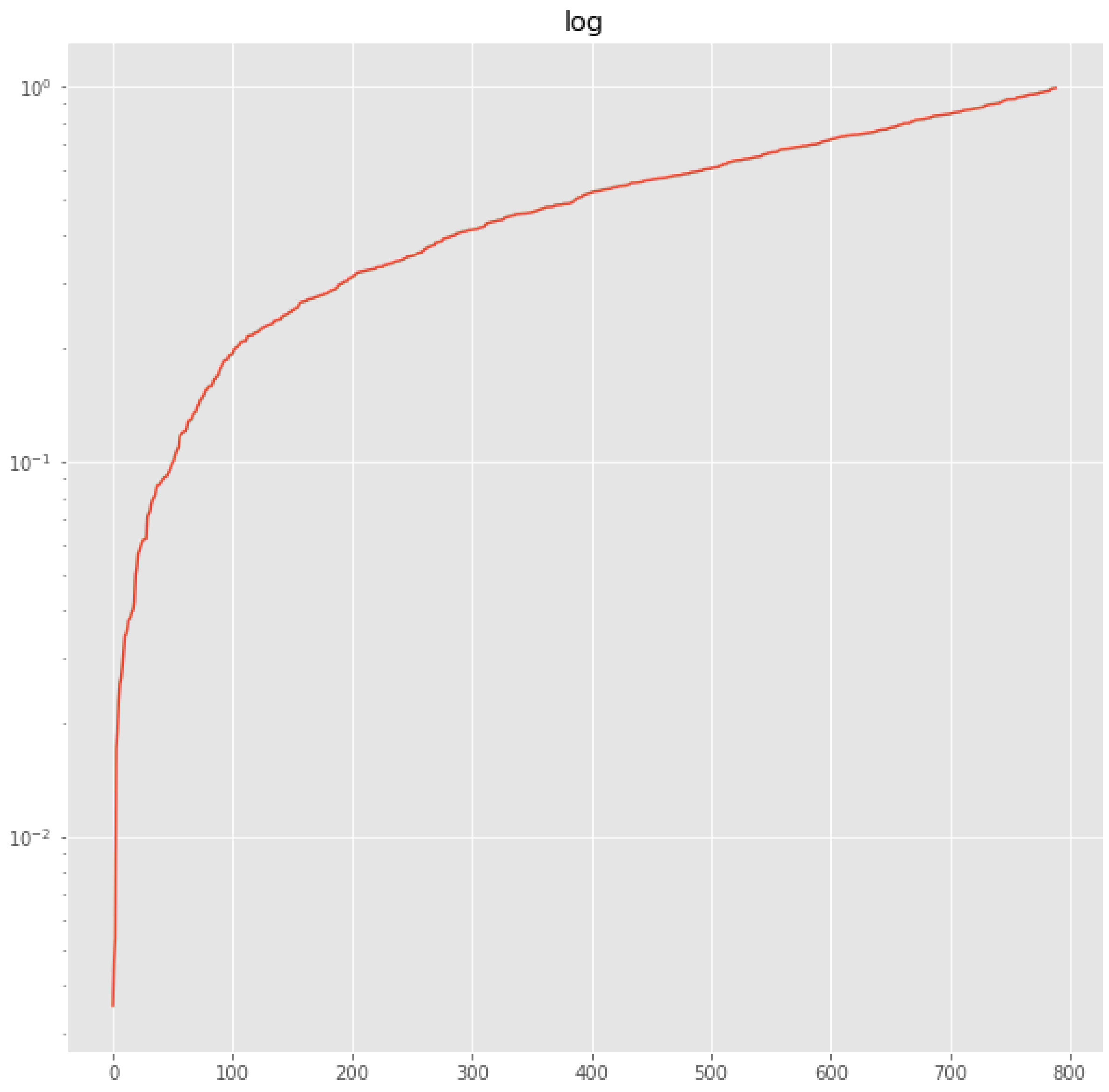
matplotlib.pyplot supports not only linear axis scales, but also logarithmic and logit scales. This is commonly used if data spans many orders of magnitude. Changing the scale of an axis is easy:

```
plt.xscale('log')
```

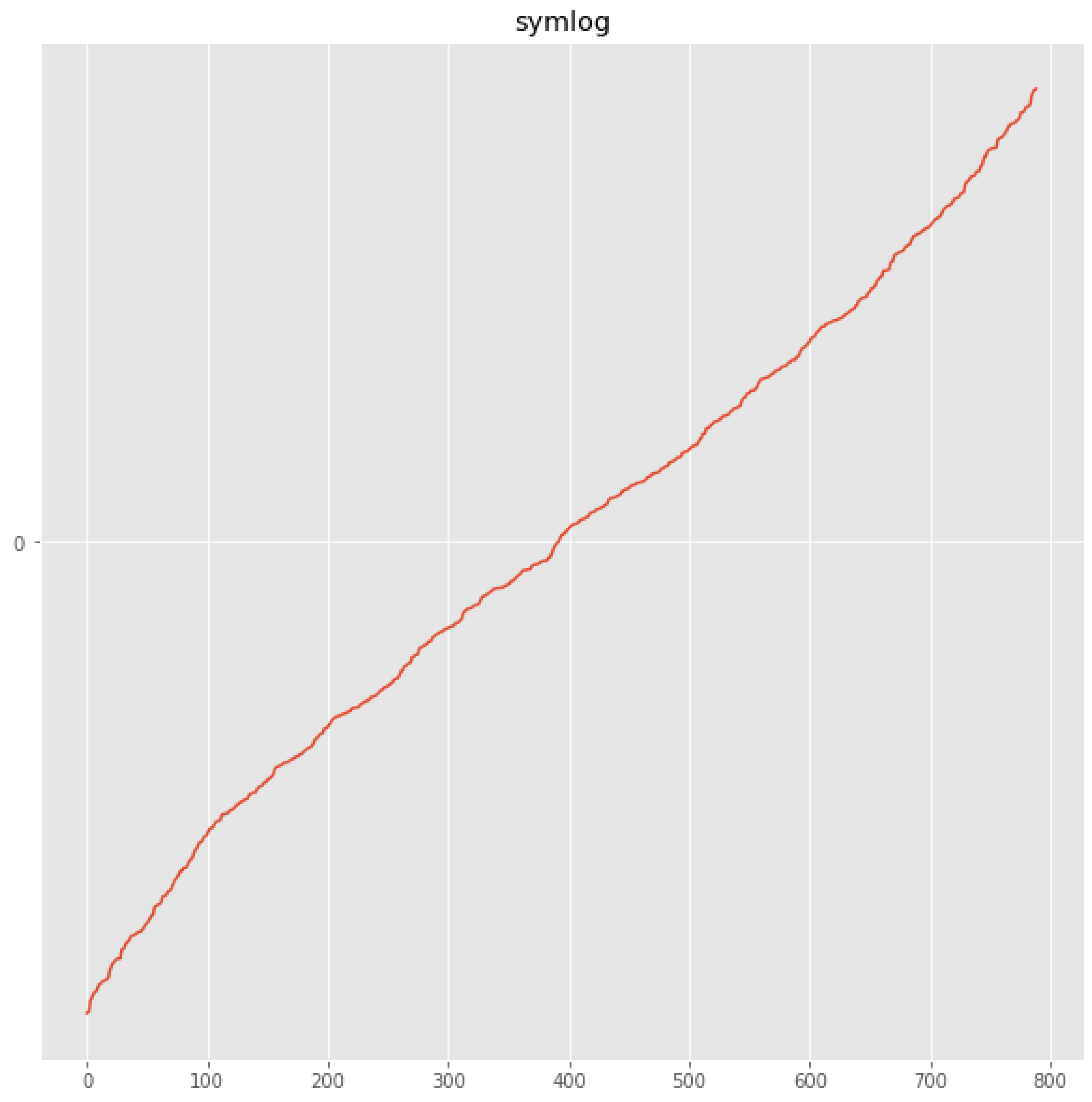
In [92]:
```python
from matplotlib.pyplot import figure

# make up some data in the open interval (0, 1)
y = np.random.normal(loc=0.5, scale=0.4, size=1000)
y = y[(y > 0) & (y < 1)]
y.sort()
x = np.arange(len(y))
figure(num=None, figsize=(10, 10))

# linear
plt.plot(x, y)
plt.yscale('linear')
plt.title('linear')
plt.grid(True)
```



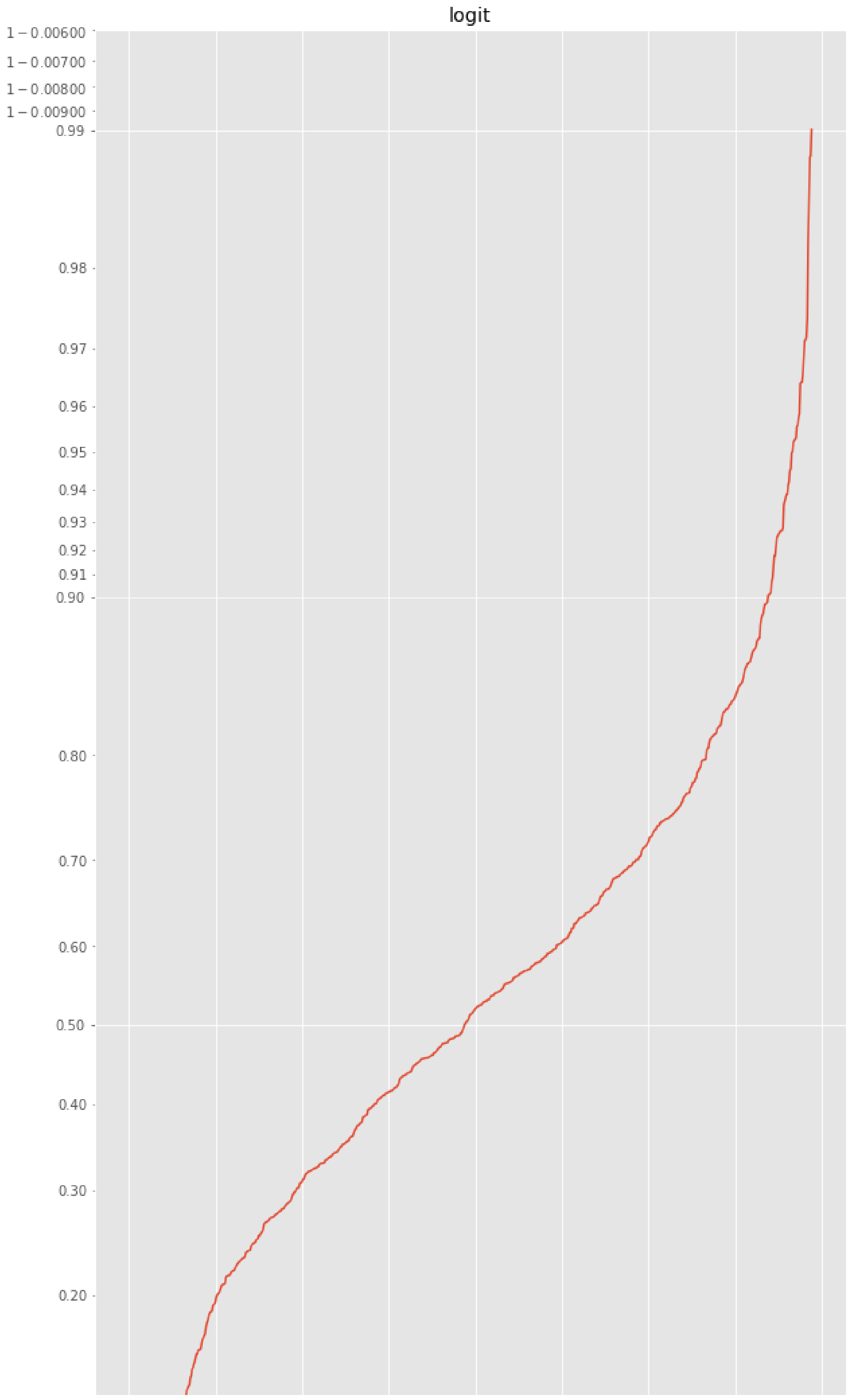
```
In [93]: # log
         figure(num=None, figsize=(10, 10))
         plt.plot(x, y)
         plt.yscale('log')
         plt.title('log')
         plt.grid(True)
         plt.show()
```

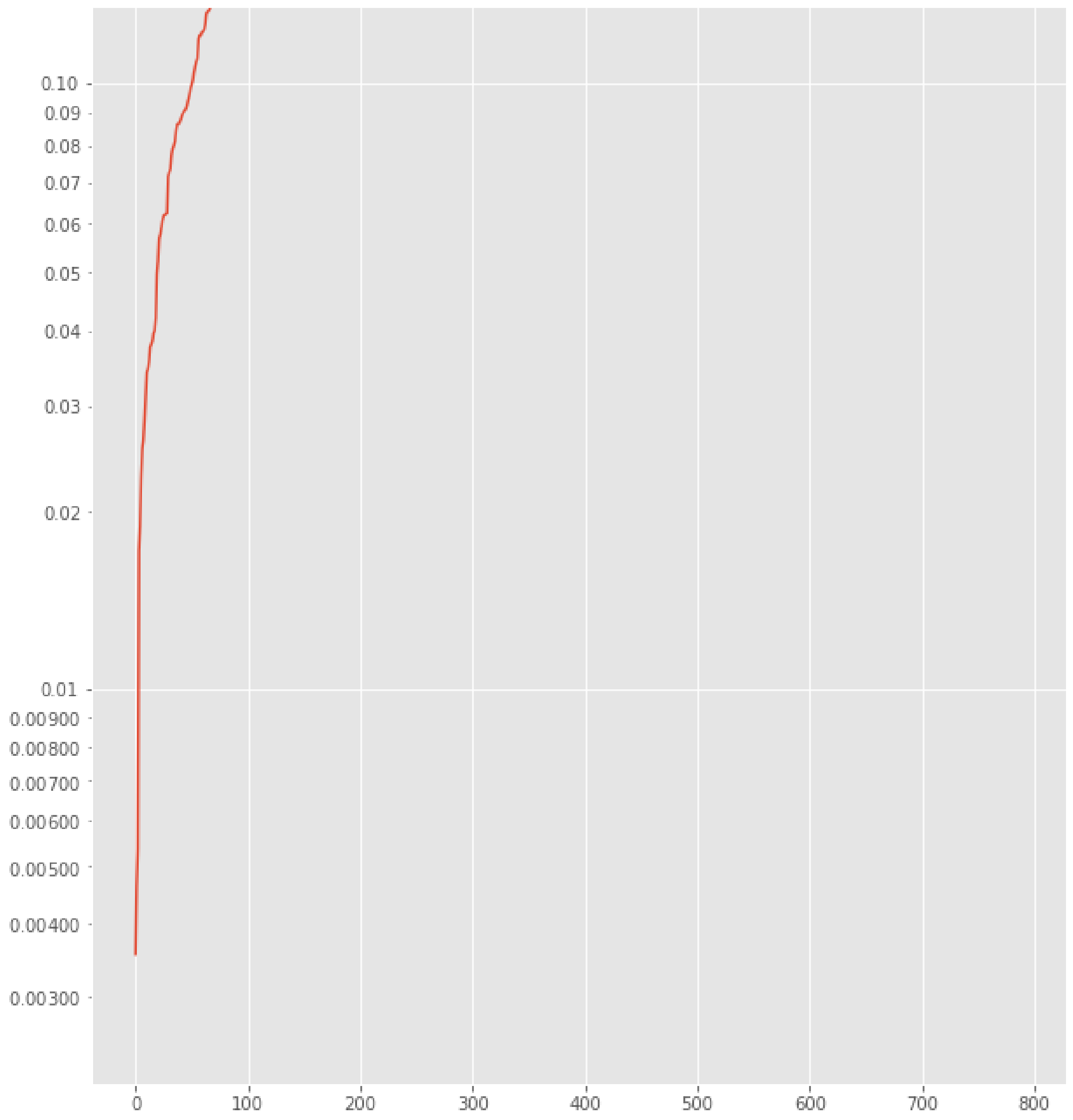


In [94]:
```
# symmetric log
figure(num=None, figsize=(10, 10))

plt.plot(x, y - y.mean())
plt.yscale('symlog', linthresh=0.01)
plt.title('symlog')
plt.grid(True)
plt.show()
```



In [95]:
```python
# logit
figure(num=None, figsize=(10, 30))
plt.plot(x, y)
plt.yscale('logit')
plt.title('logit')
plt.grid(True)
plt.show()
```


logit
1 − 0.00600
1 − 0.00700
1 − 0.00800
1 − 0.00900
0.99
0.98
0.97
0.96
0.95
0.94
0.93
0.92
0.91
0.90
0.80
0.70
0.60
0.50
0.40
0.30
0.20

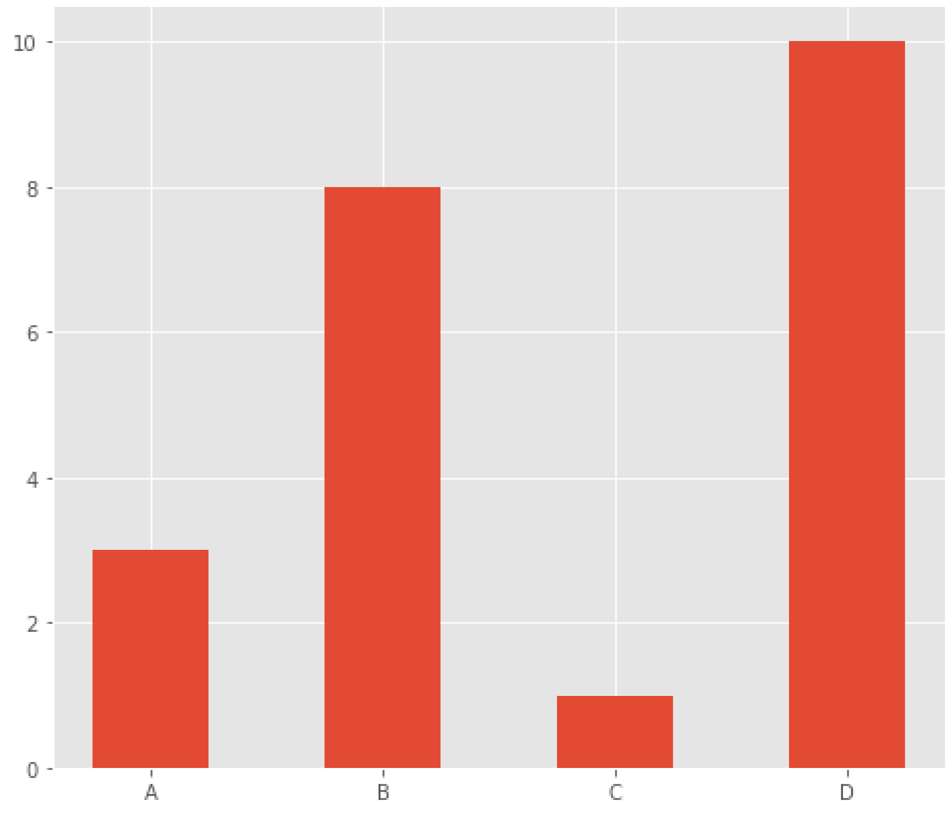

0.10
0.09
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00900
0.00800
0.00700
0.00600
0.00500
0.00400
0.00300
0
100
200
300
400
500
600
700
800

Matplotlib Bars

With Pyplot, you can use the bar() function to draw bar graphs The bar() function takes arguments that describes the layout of the bars. The categories and their values represented by the first and second argument as arrays.

```
 x = ["APPLES", "BANANAS"]
y = [400, 350]plt.bar(x, y)
plt.bar(x, y)
```

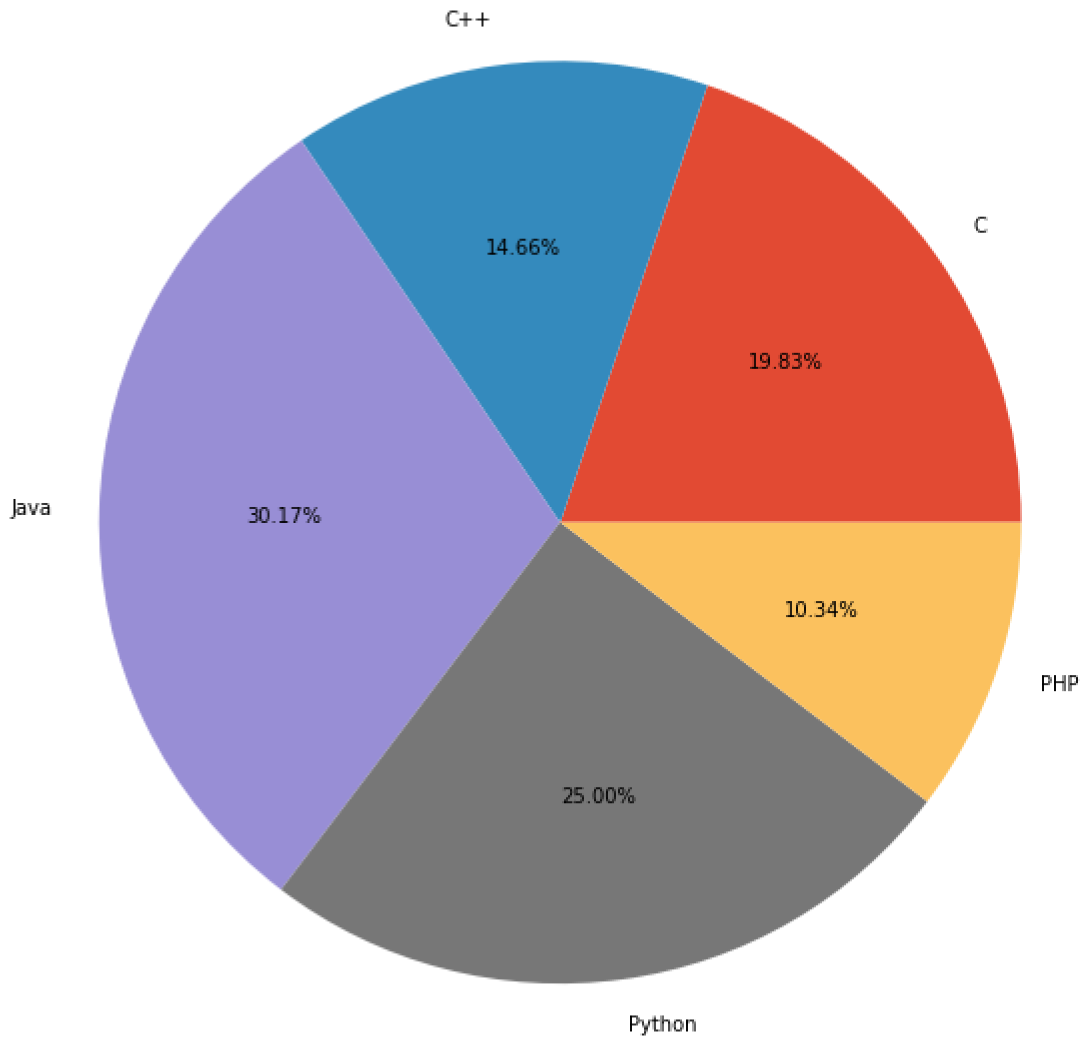
```
In [96]: x = np.array(["A", "B", "C", "D"])
         y = np.array([3, 8, 1, 10])

         plt.bar(x, y, width = 0.5)
         plt.show()
```



Matplotlib Pie Charts

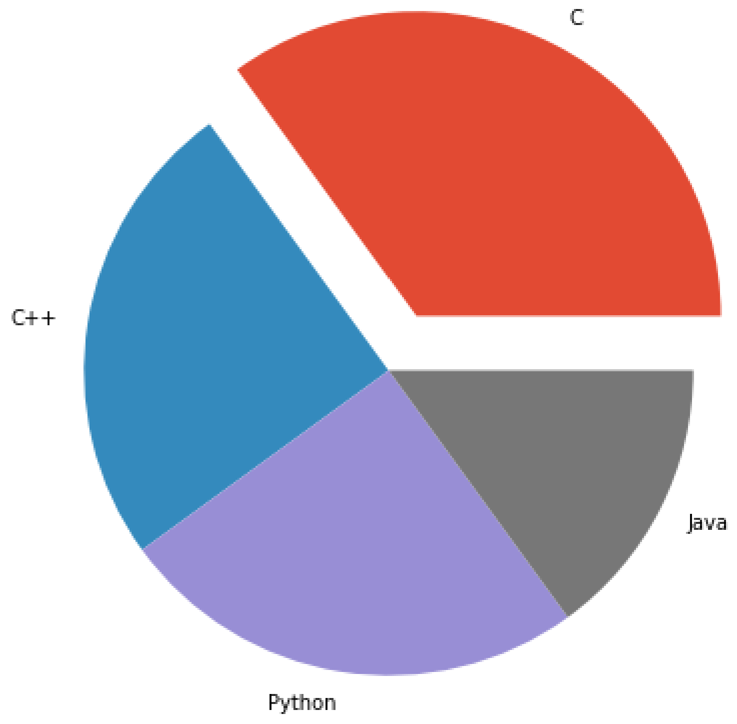
A Pie Chart can only display one series of data. Pie charts show the size of items (called wedge) in one data series, proportional to the sum of the items. The data points in a pie chart are shown as a percentage of the whole pie. Explode: Maybe you want one of the wedges to stand out? The explode parameter allows you to do that. The explode parameter, if specified, and not None, must be an array with one value for each wedge.

```
In [97]: fig = plt.figure()
         ax = fig.add_axes([0,0,1,1])
         ax.axis('equal')
         langs = ['C', 'C++', 'Java', 'Python', 'PHP']
         students = [23,17,35,29,12]
         ax.pie(students, labels = langs,autopct='%1.2f%%')
         plt.show()
```



Explode: Maybe you want one of the wedges to stand out? The explode parameter allows you to do that. The explode parameter, if specified, and not None, must be an array with one value for each wedge.

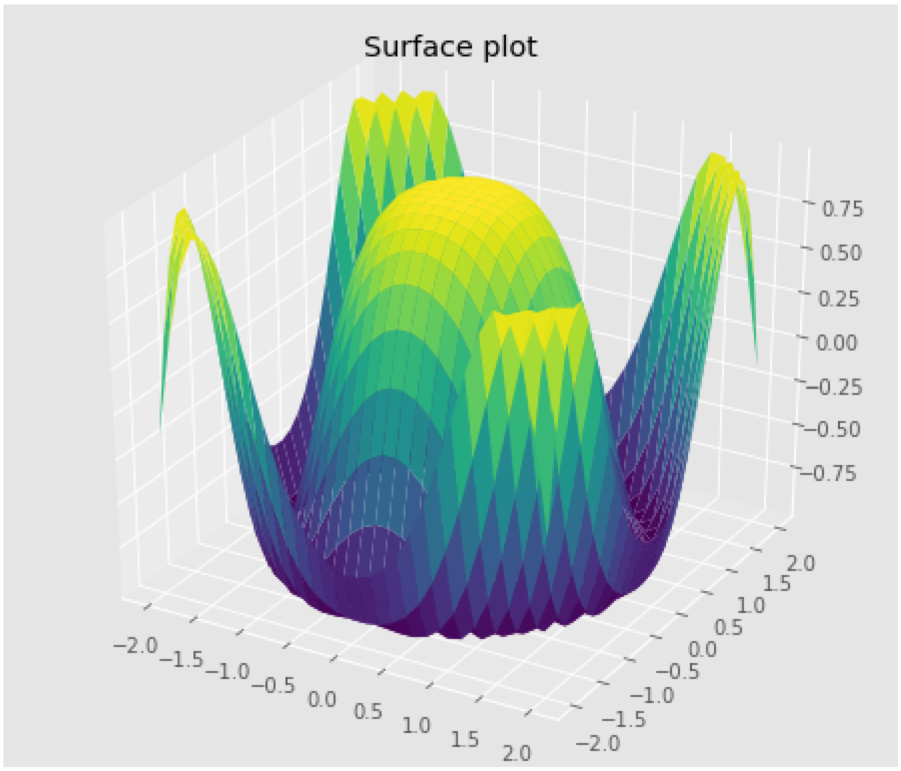
```
In [98]: y = np.array([35, 25, 25, 15])
         mylabels = ["C", "C++", "Python", "Java"]
         myexplode = [0.2, 0, 0, 0]
         plt.pie(y, explode = myexplode , labels = mylabels)
         plt.show()
```



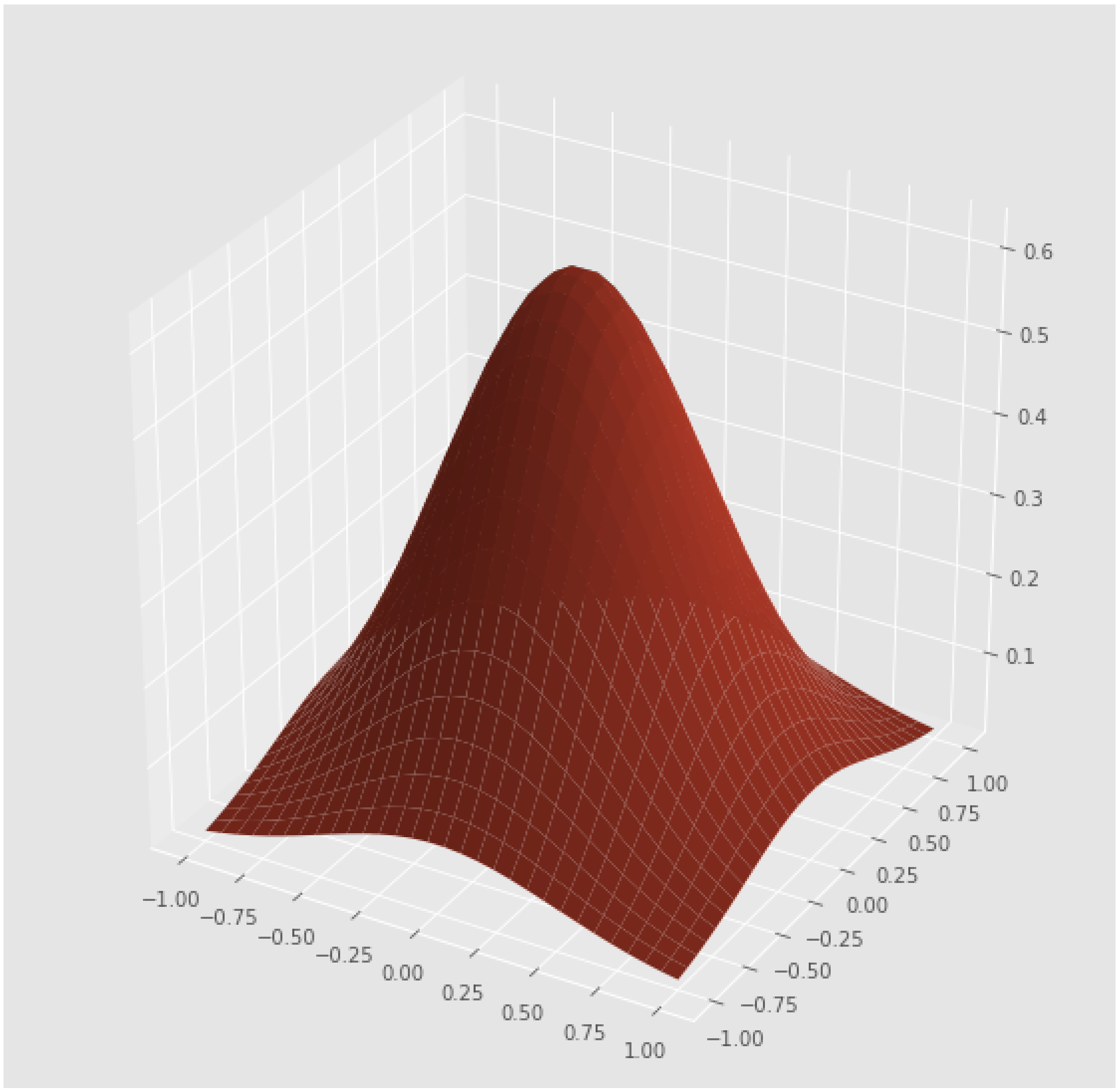
3D Surface plot

Surface plot shows a functional relationship between a designated dependent variable (Y), and two independent variables (X and Z). The plot is a companion plot to the contour plot. A surface plot is like a wireframe plot, but each face of the wireframe is a filled polygon. This can aid perception of the topology of the surface being visualized. The plot_surface() function x,y and z as arguments.

```
In [99]: from mpl_toolkits import mplot3d

         x = np.outer(np.linspace(-2, 2, 30), np.ones(30))
         y = x.copy().T
         z = np.cos(x ** 2 + y ** 2)

         fig = plt.figure()
         ax = plt.axes(projection='3d')

         ax.plot_surface(x, y, z,cmap='viridis', edgecolor='none')
         ax.set_title('Surface plot')
         plt.show()
```



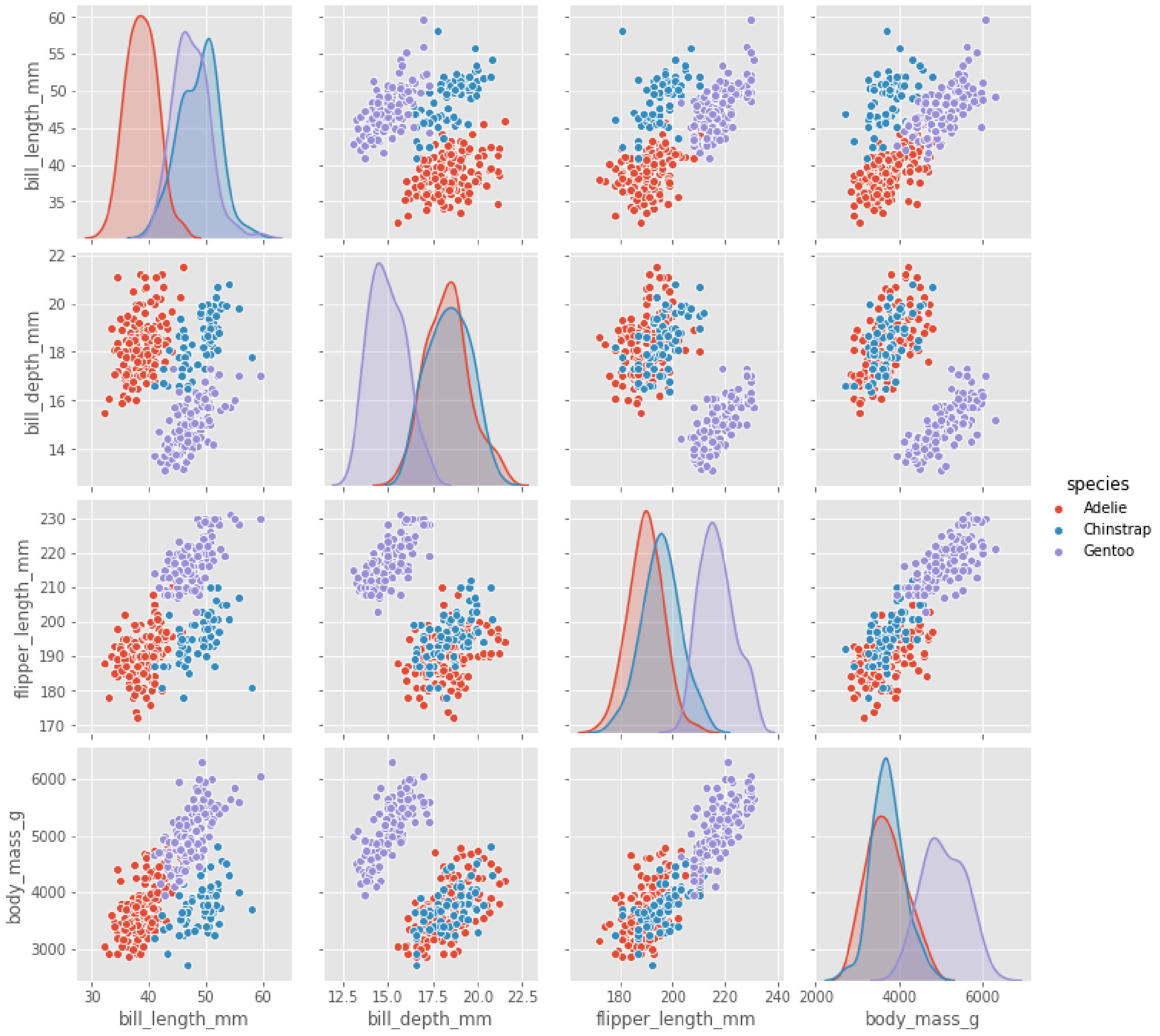
In [100]:
```
from mpl_toolkits.mplot3d import Axes3D
from scipy.stats import multivariate_normal

x, y = np.mgrid[-1.0:1.0:30j, -1.0:1.0:30j]

xy = np.column_stack([x.flat, y.flat])

mu = np.array([0.0, 0.0])

sigma = np.array([.5, .5])
covariance = np.diag(sigma**2)

z = multivariate_normal.pdf(xy, mean=mu, cov=covariance)

z = z.reshape(x.shape)

fig = plt.figure(num=None, figsize=(10, 10))

ax = fig.add_subplot(111, projection='3d')



ax.plot_surface(x,y,z)

plt.show()
```



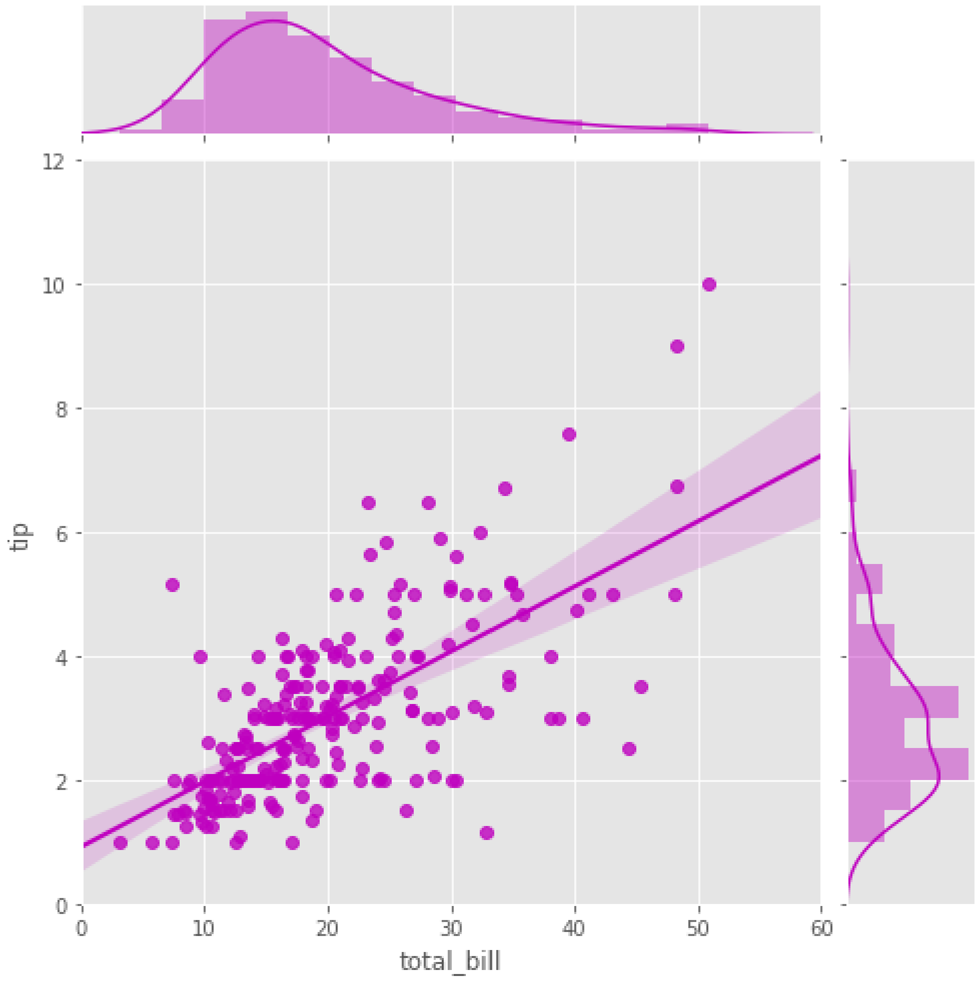
Seaborn

Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

```
In [101]: import seaborn as sns
          df = sns.load_dataset("penguins")
          sns.pairplot(df, hue="species")

Out[101]: <seaborn.axisgrid.PairGrid at 0x7fd29609e550>
```



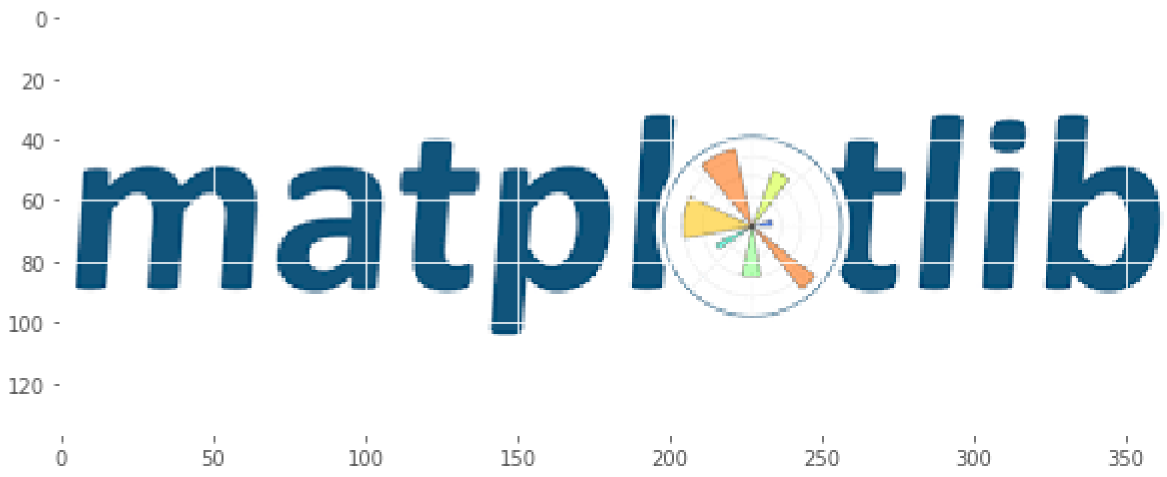
```
In [102]: tips = sns.load_dataset("tips")
          g = sns.jointplot(x="total_bill", y="tip", data=tips,
                            kind="reg", truncate=False,
                            xlim=(0, 60), ylim=(0, 12),
                            color="m", height=7)
```



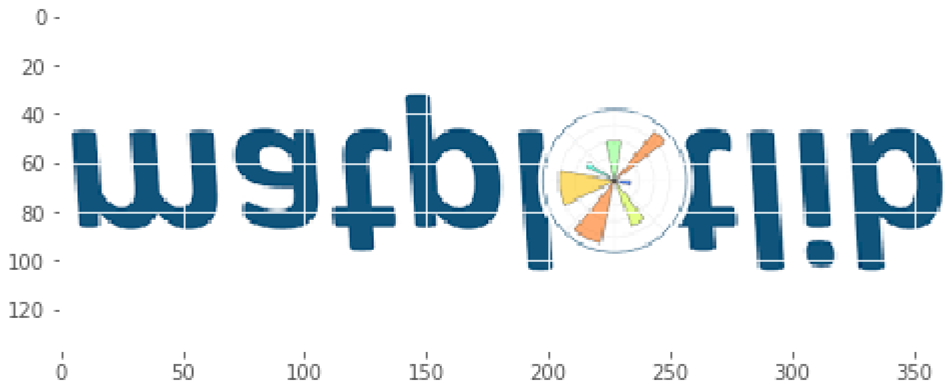
Working with Images

The image module in Matplotlib package provides functionalities required for loading, rescaling and displaying image. Loading image data is supported by the Pillow library and Matplotlib relies on the Pillow library to load image data. The imread() function is used to read image data in an ndarray object of float32 dtype.

In [103]:
```
import matplotlib.image as mpimg
img = mpimg.imread('mpl.png')
figure(num=None, figsize=(10, 10))
imgplot = plt.imshow(img)
```



In [104]:
```
plt.imsave("logo.png", img, cmap = 'gray', origin = 'lower')
newimg = mpimg.imread('logo.png')
imgplot = plt.imshow(newimg)
```

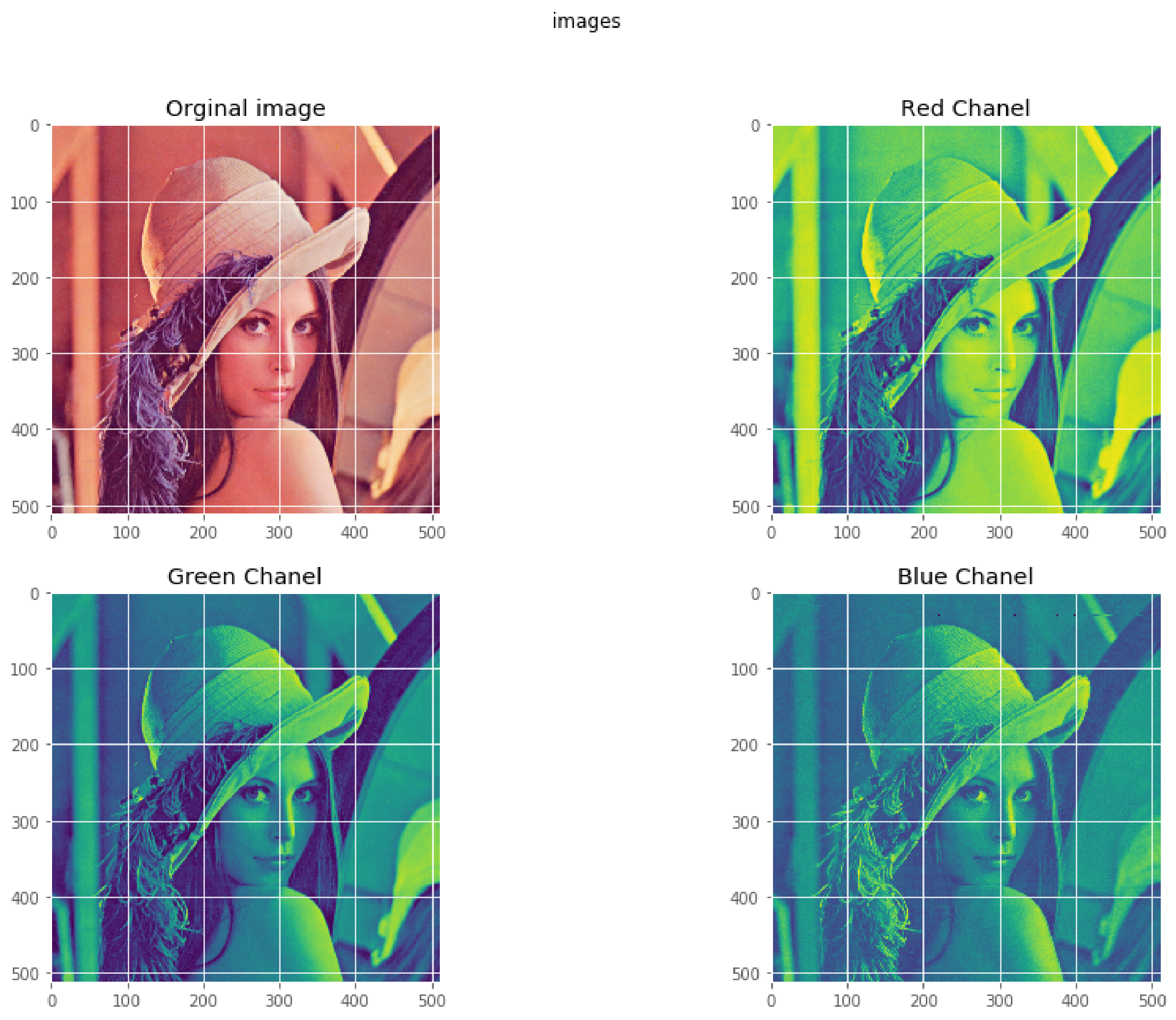


Get image size (width, height, dimension)

In [105]:
```
img_lena = mpimg.imread('lena.png')

print(img_lena.shape)
```

```
(512, 512, 3)
```

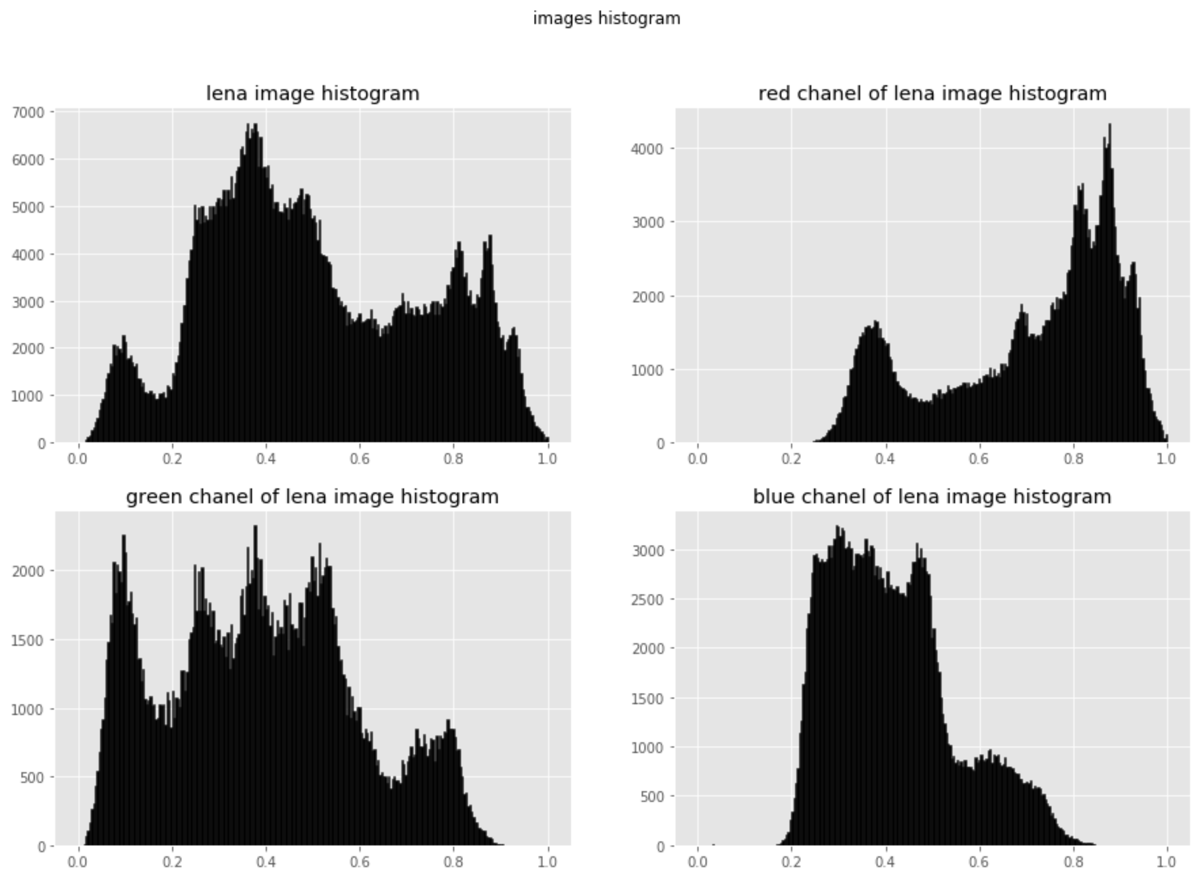
In [106]:
```
figure(num=None, figsize=(15, 10))

plt.subplot(2, 2, 1)
plt.imshow(img_lena[:,:,:])
plt.title("Orginal image")


plt.subplot(2, 2, 2)
red_img_lena = img_lena[:,:,0]
plt.imshow(red_img_lena)
plt.title("Red Chanel")



plt.subplot(2, 2, 3)
green_img_lena = img_lena[:,:,1]
plt.imshow(green_img_lena)
plt.title("Green Chanel")

plt.subplot(2, 2, 4)
blue_img_lena = img_lena[:,:,2]
plt.imshow(blue_img_lena)
plt.title("Blue Chanel")


plt.suptitle(" images")
plt.show()
```



Examining a specific data range

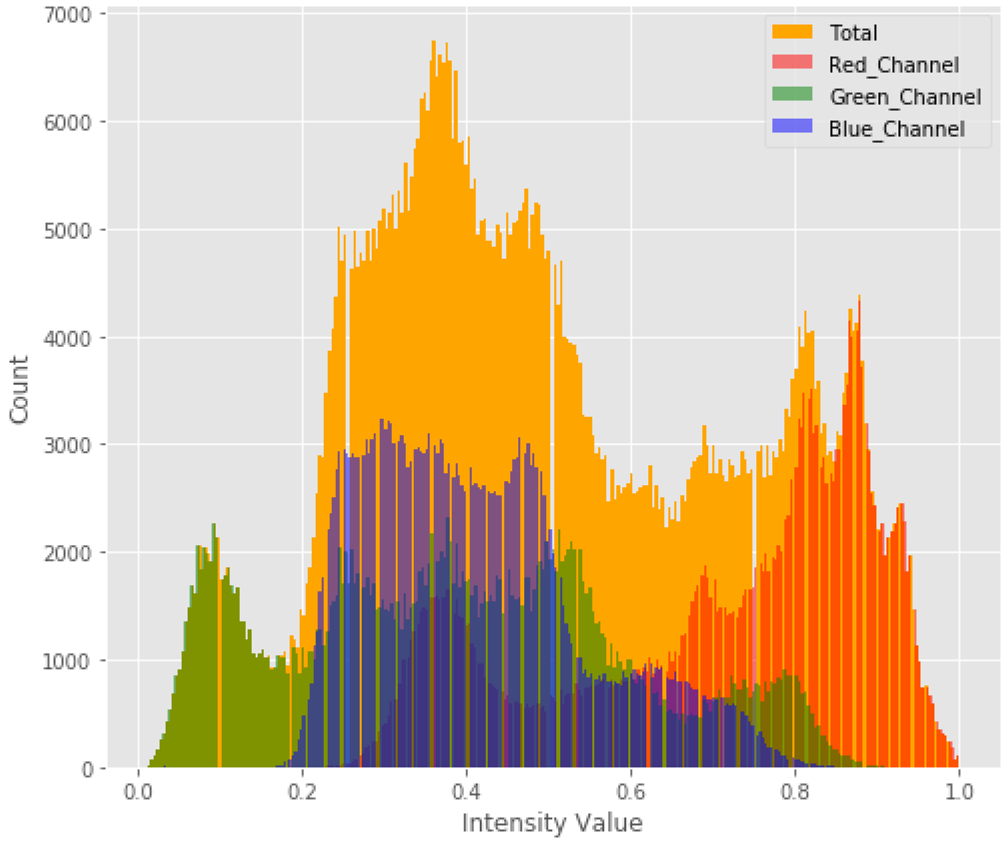
Sometimes you want to enhance the contrast in your image, or expand the contrast in a particular region while sacrificing the detail in colors that don't vary much, or don't matter. A good tool to find interesting regions is the histogram. To create a histogram of our image data, we use the hist() function.

```
In [107]: figure(num=None, figsize=(15, 10))

plt.subplot(2, 2, 1)
plt.hist(img_lena.ravel(), bins=256, range=(0.0, 1.0), fc='k', ec='k')
plt.title("lena image histogram")

plt.subplot(2, 2, 2)
plt.hist(red_img_lena.ravel(), bins=256, range=(0.0, 1.0), fc='k', ec='k')
plt.title("red chanel of lena image histogram")

plt.subplot(2, 2, 3)
plt.hist(green_img_lena.ravel(), bins=256, range=(0.0, 1.0), fc='k', ec='k')
plt.title("green chanel of lena image histogram")

plt.subplot(2, 2, 4)
plt.hist(blue_img_lena.ravel(), bins=256, range=(0.0, 1.0), fc='k', ec='k')
plt.title("blue chanel of lena image histogram")

plt.suptitle(" images histogram")
plt.show()
```



In color images, we have 3 color channels representing RGB. In Combined Color Histogram the intensity count is the sum of all three color channels.

```
In [108]: import matplotlib.pyplot as plt
          image = plt.imread('lena.png')

          _ = plt.hist(image.ravel(), bins = 256, color = 'orange', )
          _ = plt.hist(image[:, :, 0].ravel(), bins = 256, color = 'red', alpha = 0.5)
          _ = plt.hist(image[:, :, 1].ravel(), bins = 256, color = 'Green', alpha = 0.5)
          _ = plt.hist(image[:, :, 2].ravel(), bins = 256, color = 'Blue', alpha = 0.5)
          _ = plt.xlabel('Intensity Value')
          _ = plt.ylabel('Count')
          _ = plt.legend(['Total', 'Red_Channel', 'Green_Channel', 'Blue_Channel'])
          plt.show()
```

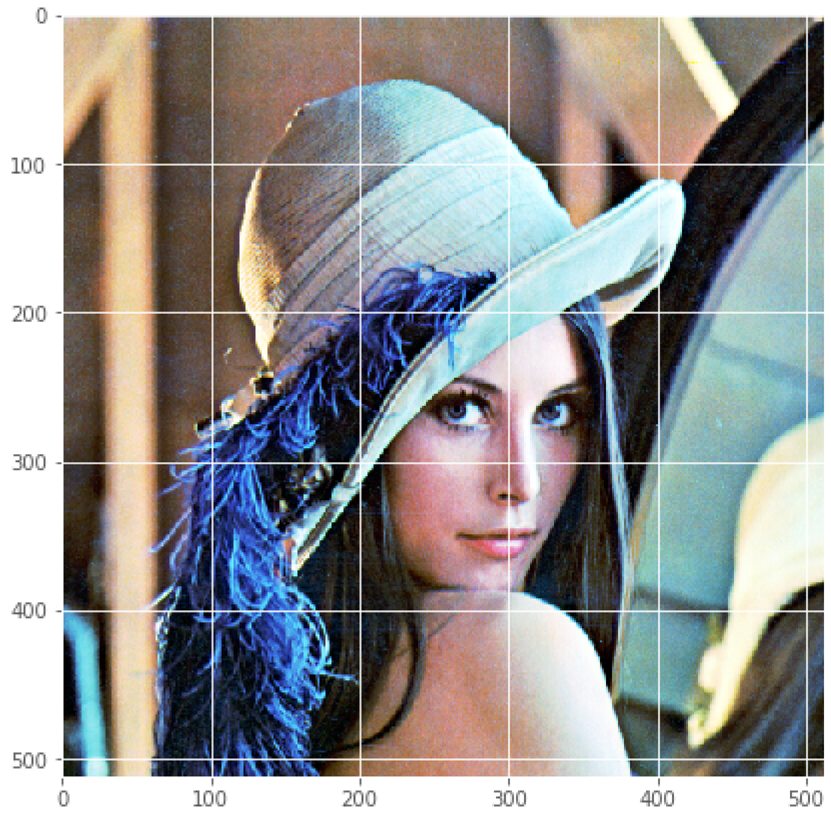


Histogram equalization

Histogram equalization is a method in image processing of contrast adjustment using the image's histogram.

```
In [109]: # Importing Image and ImageOps module from PIL package
          from PIL import Image, ImageOps
          im1 = Image.open(r"lena.png")
          im2 = ImageOps.equalize(im1, mask = None)
          plt.imshow(im2)
```

```
Out[109]: <matplotlib.image.AxesImage at 0x7fd2ac4d8910>
```

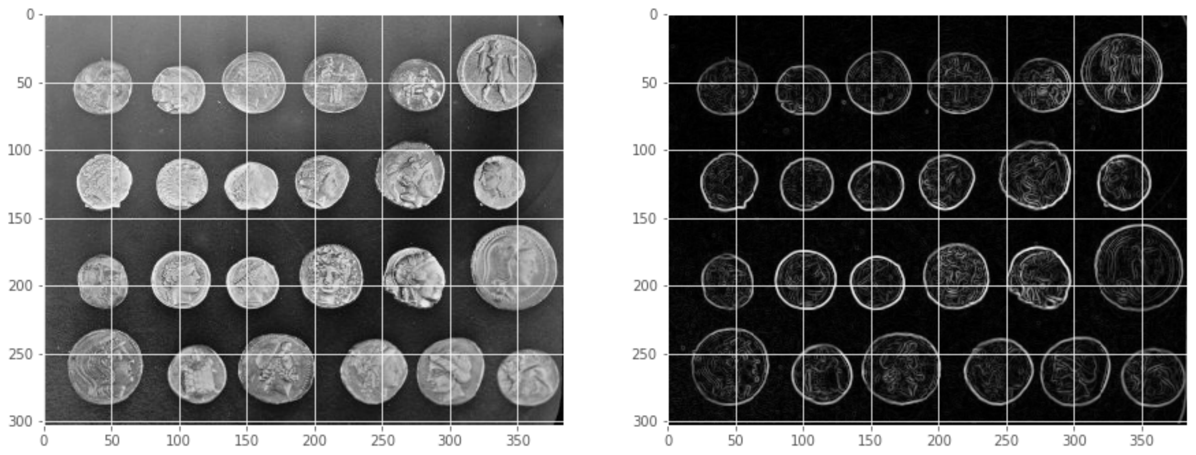


scikit-image

scikit-image is a collection of algorithms for image processing.

```
In [110]: from skimage import data,filters

          figure(num=None, figsize=(15, 10))

          img_coin = data.coins()
          edges_coin = filters.sobel(img_coin)
          plt.subplot(1, 2, 1)
          plt.imshow(img_coin, cmap='gray')
          plt.subplot(1, 2, 2)
          plt.imshow(edges_coin, cmap='gray')
```

```
Out[110]: <matplotlib.image.AxesImage at 0x7fd2b0ef0890>
```

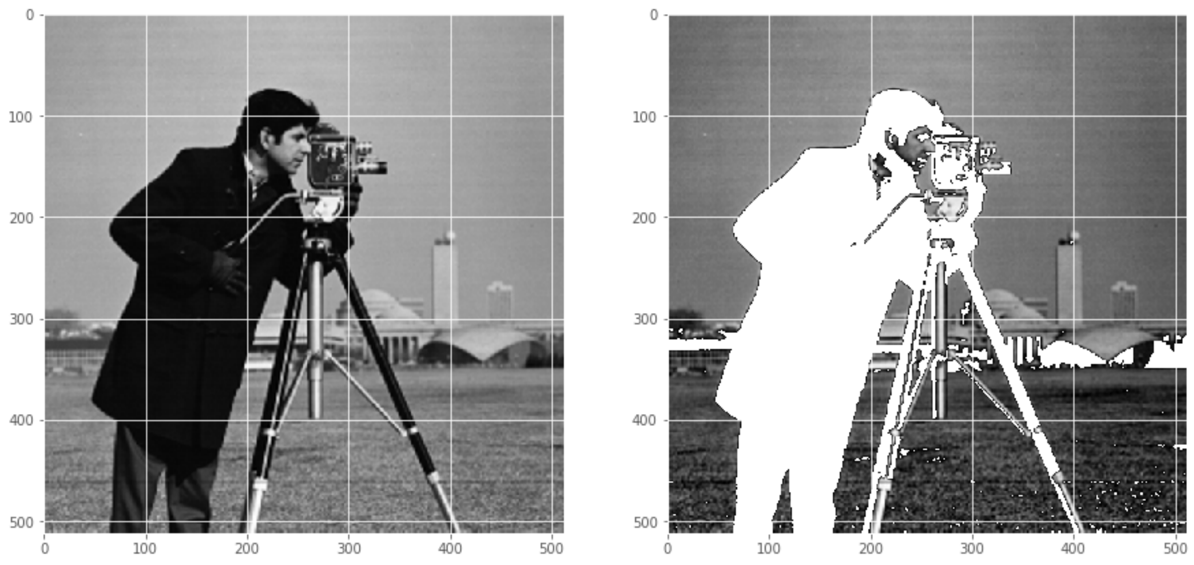


NumPy

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.

In [111]:
```
figure(num=None, figsize=(15, 10))

image_camera = data.camera()

plt.subplot(1, 2, 1)
plt.imshow(image_camera, cmap='gray')


mask_camera = image_camera < 87
image_camera[mask_camera]=255
plt.subplot(1, 2, 2)

plt.imshow(image_camera, cmap='gray')
```

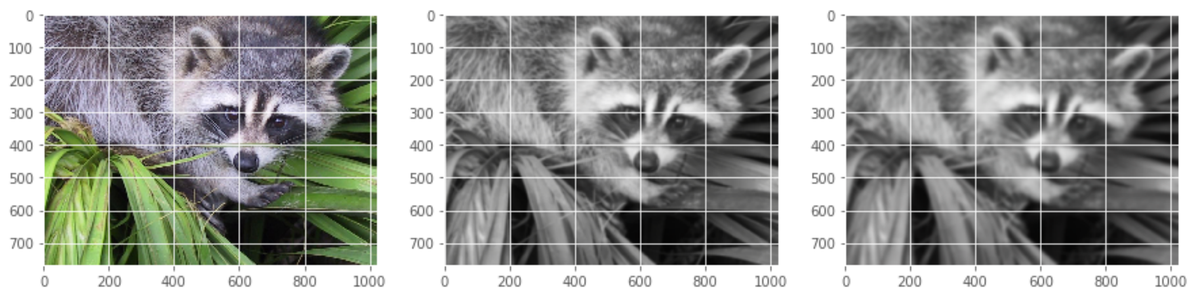
Out[111]:
```
<matplotlib.image.AxesImage at 0x7fd2b2769250>
```



Using SciPy for blurring using a Gaussian filter:

In [112]:
```python
from scipy import misc,ndimage
figure(num=None, figsize=(15, 10))

ime_face = misc.face()
blurred_face = ndimage.gaussian_filter(ime_face, sigma=3)
very_blurred = ndimage.gaussian_filter(ime_face, sigma=5)
plt.subplot(1, 3, 1)
plt.imshow(ime_face)
plt.subplot(1, 3, 2)
plt.imshow(blurred_face)
plt.subplot(1, 3, 3)
plt.imshow(very_blurred)
```

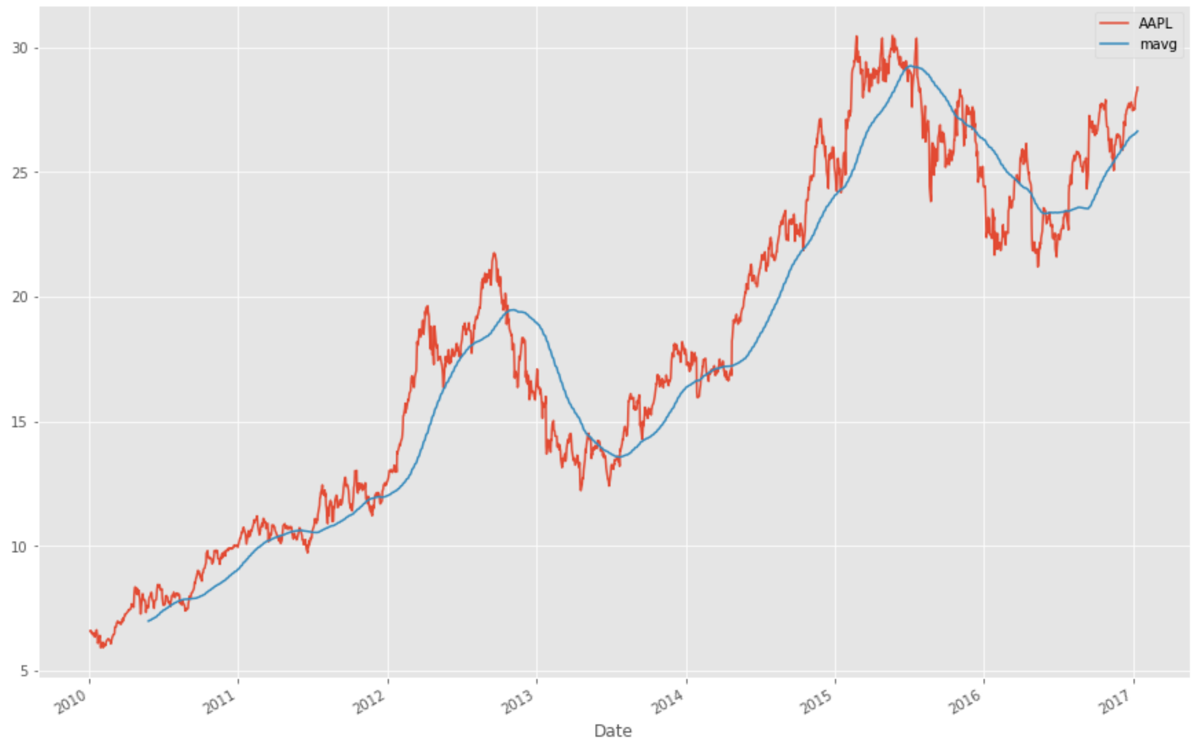
Out[112]:
```
<matplotlib.image.AxesImage at 0x7fd2aca3d110>
```



exciting small project

In [113]:
```python
import pandas as pd
import datetime
import pandas_datareader.data as web
from pandas import Series, DataFrame
figure(num=None, figsize=(15, 10))


start = datetime.datetime(2010, 1, 1)
end = datetime.datetime(2017, 1, 11)

df = web.DataReader("AAPL", 'yahoo', start, end)
close_px = df['Adj Close']
mavg = close_px.rolling(window=100).mean()

from matplotlib import style

# Adjusting the size of matplotlib
import matplotlib as mpl
mpl.rc('figure', figsize=(8, 7))
mpl.__version__

# Adjusting the style of matplotlib
style.use('ggplot')

close_px.plot(label='AAPL')
mavg.plot(label='mavg')
plt.legend()
```

Out[113]:
```
<matplotlib.legend.Legend at 0x7fd2b12c1b90>
```


AAPL
mavg
Date