# matplotlib

September 5, 2017

INTRODUCTION TO MATPLOTLIB
Data Analysis Club

## 0.1 matplotlib.pyplot

### 0.1.1 matplolib:

Matplotlib is a python library used to 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.

Matplotlib Library has functions each for a specific purpose. To access or to use these functions in your python code, just include these lines in beginning of your code **import matplotlib.pyplot as plt** Note: *Throughout your code u can write matplotlib.pyplot as plt*

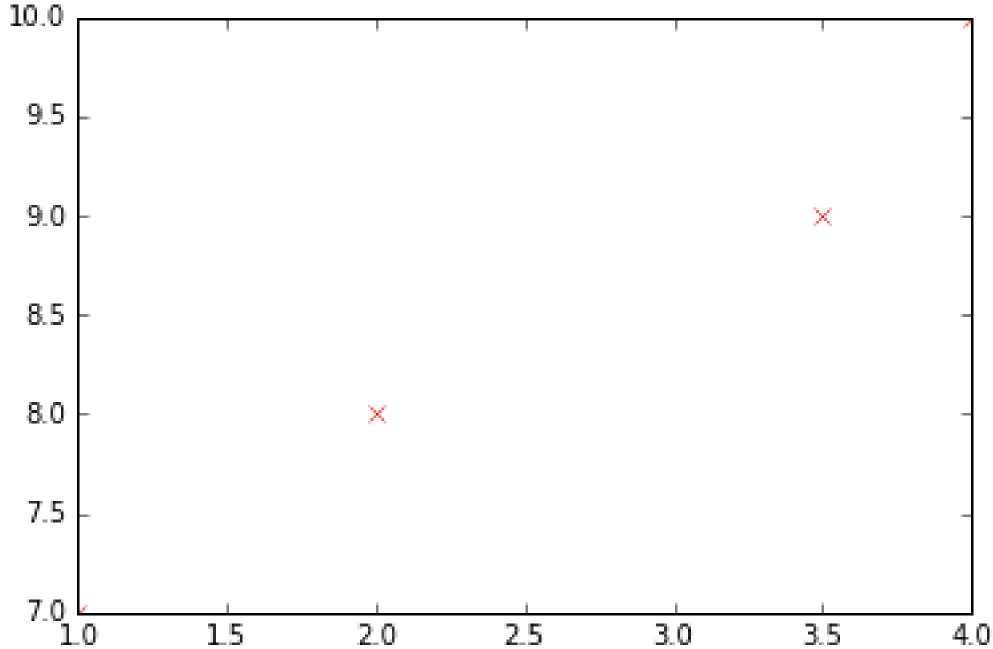
- Documentation

### 0.1.2 Let us walk through some codes, We are going to use following functions:

- **plt.plot(xAxisData,yAxisData, arg)**
  - *arg1* is the data you want to plot
  - *arg2* is type of graph you want to plot, e.g. circles, triangles, solid line, squares.
  - *ro* stands for red circle *bo* stand for blue cicle so you got that *o* in *ro* or *bo* stands for circle
  - *bs* for blue square
  - *g^* for green triangle
  - *b--* for blue dash line
  - default value is solid line
- **plt.ylabel("label for y")**
- **plt.xlabel("label for x")**
- **plt.show()** for showing plot
- Read plt.style

```
In [1]: import matplotlib.pyplot as plt
```

```
In [2]: import matplotlib.pyplot as plt
        # from matplotlib import style
        # plt.style.use('ggplot')
        plt.plot([1,2,3.5,4],[7,8,9,10],'rx')
        # plt.ylabel("some numbers")
        plt.show()
```



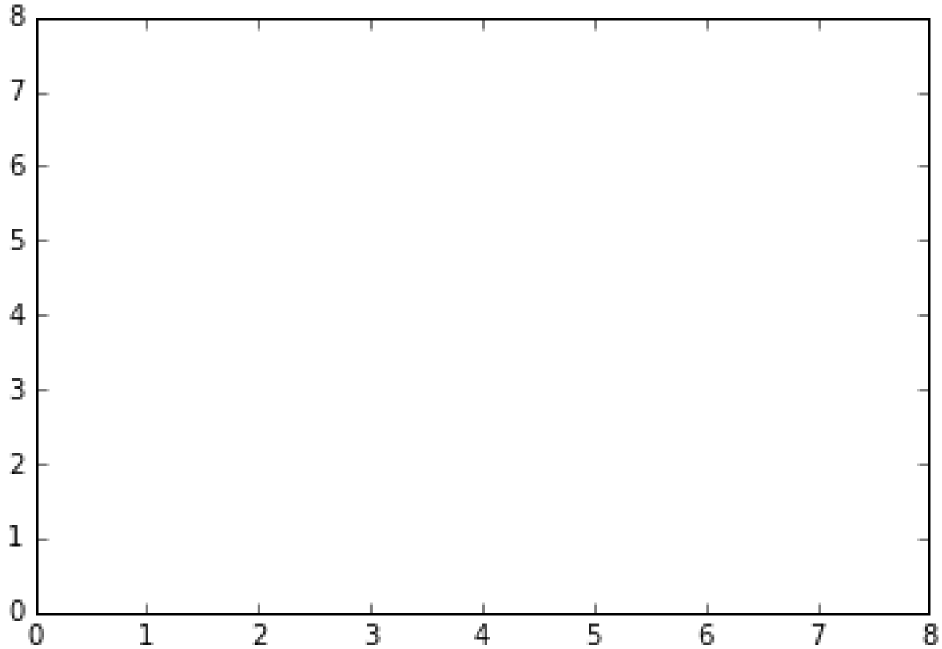

```
In [3]: %matplotlib inline
```

---

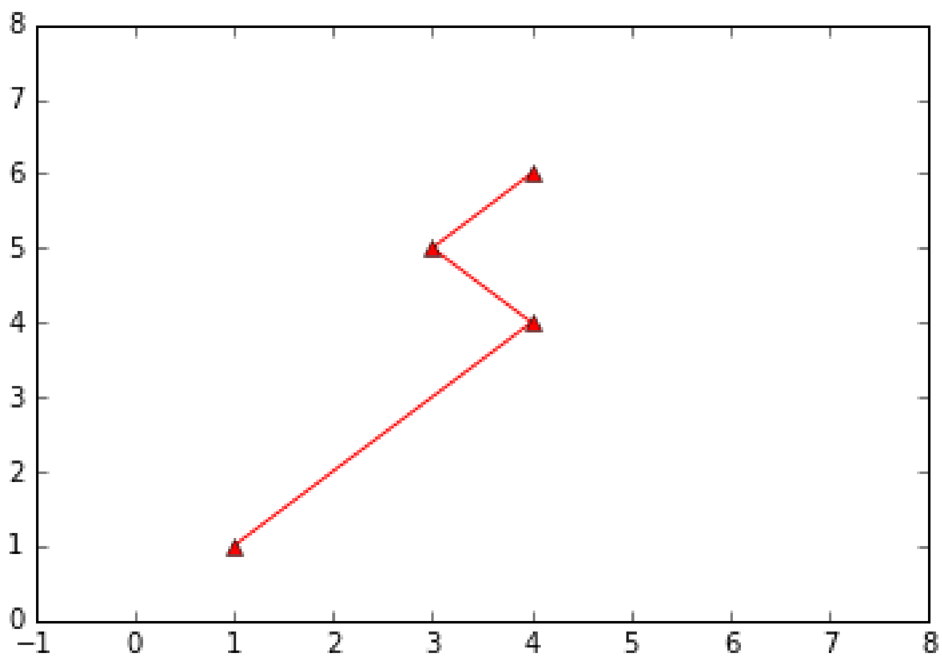
Now let we want to change the range of x and y axis, we will use **plt.axis** command for this purpose. ### plt.axis([xmin, xmax, ymin, ymax])

```
In [4]: plt.axis([0,8,0,8])
```

```
Out[4]: [0, 8, 0, 8]
```




In [5]:
```
plt.plot([1,4,3,4], [1,4,5,6],'r-',marker='^')    #Same operation can also be achieved wi
#try to find out what marker does, once delete word marker='^' in above statement and se
plt.axis([-1,8,0,8])
plt.show()
```

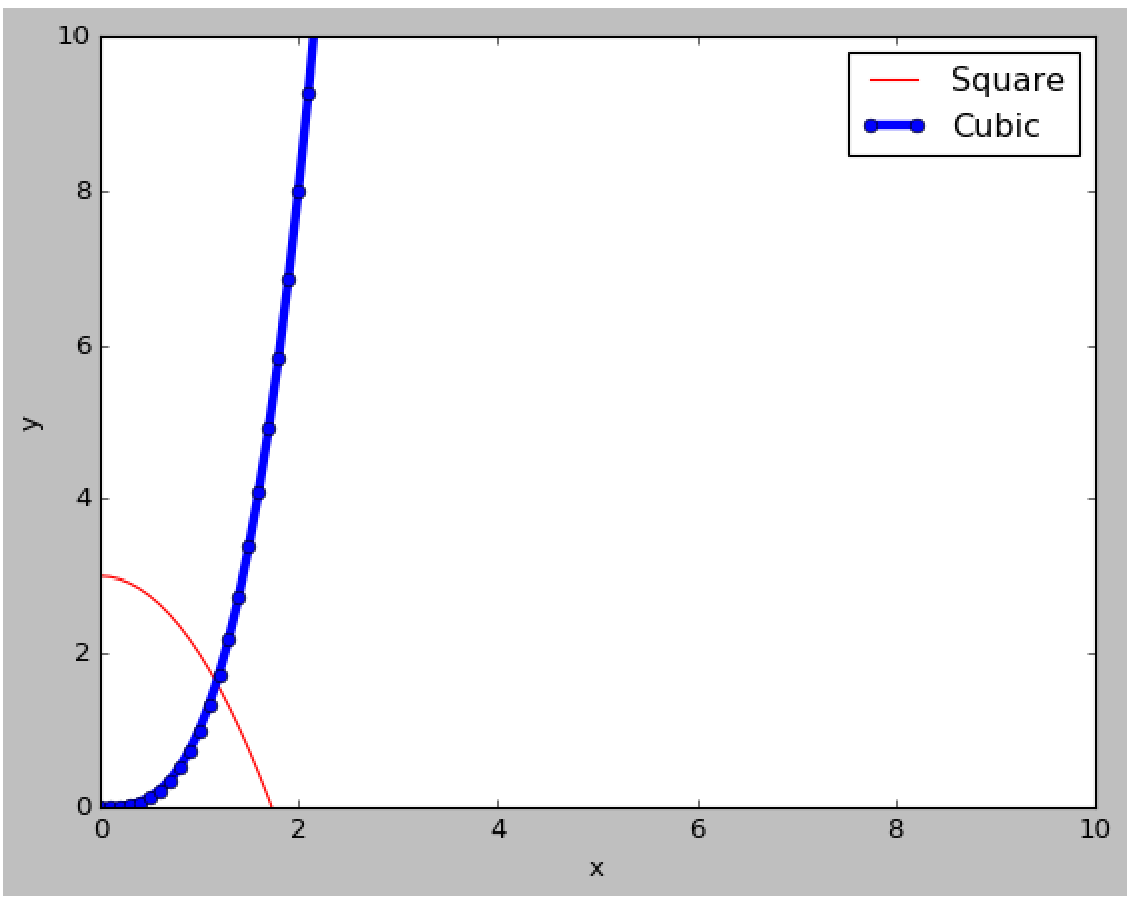


## 0.2 Let us plot graphs of square and cubic function

- line, =plt.plot() then - line is a Line2D instance returned by plot.

```
In [6]: import numpy as np
        import matplotlib.pyplot as plt
        from matplotlib import style
        plt.style.use('classic')
        arr= np.arange(0, 20, .1)
        line1, =plt.plot(arr, (-arr**2)+3,'r',label="Square",ms=4)
        line2, =plt.plot(arr, arr**3, 'b-',marker='o')
        print(type(line2))
        plt.axis([0,10,0,10])
        plt.xlabel('x')
        plt.ylabel('y')
        plt.setp(line2, label="Cubic", linewidth=4.0 ) #You can set label using plt.setp.
        # plt.legend(handles=[line1,line2])
        plt.legend()
        # plt.setp(line2)
        plt.show()
```

```
<class 'matplotlib.lines.Line2D'>
```



## 0.3 Plotting multiple figure

**fig,ax= plt.subplot(numrows, numcols, fignum)** *where fignum ranges from 1 to numrows X numcols* **- fig: matplotlib.figure.Figure object - ax: Axes object or array of Axes objects. ax can be either a single matplotlib.axes.Axes object or an array of Axes objects**

Let us plot above graph of sin and cosine function on separated figures

## 0.4 Adding Text at specihic location in graph

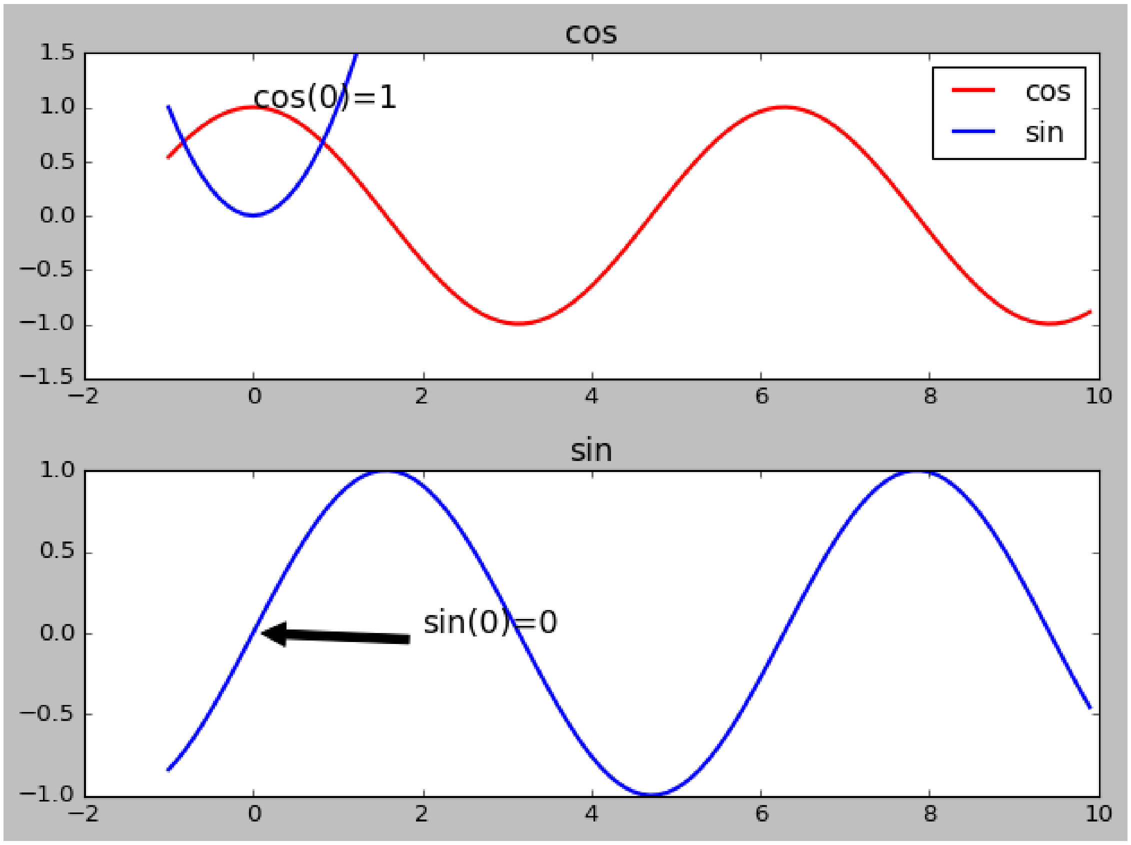
- ** plt.text(x, y, "text")** *where x and y are x and y coordinates*

## 0.5 Annotating text

- ** plt.annotate("text", xy=(x_coordinate, y_coordinate), xytext=(x'_xoordinate, y'_coordiante) , arrowdrops= dict(facecolor="black", shrink="0.05"))**

```
In [7]: import matplotlib.pyplot as plt
        import numpy as np
        from matplotlib import style
        plt.style.use('classic')

        arr= np.arange(-1, 10, 0.1)
        ax=plt.subplot(2,1,1)
        print(type(ax))
        plt.text(0, 1, "cos(0)=1", fontsize="16")  #adding text
        plt.axis([-2,10,-1.5,1.5])
        plt.title("cos",fontsize="16") #add title to figure
        line1,= plt.plot(arr, np.cos(arr), 'r-', linewidth='2.0')
        line2,=plt.plot(arr, np.power(arr,2), 'b-', linewidth='2.0')
        plt.legend([line1,line2],["cos","sin"])  #cos and sin are label for line1 and line2 resp

        plt.subplot(2,1,2)
        plt.title("sin",fontsize="16") #add title to figure
        plt.annotate("sin(0)=0", xy=(0,0), xytext=(2,0), arrowprops=dict(facecolor="black", shri
        plt.plot(arr, np.sin(arr), 'b-', linewidth='2.0')
        plt.tight_layout()  #Just use this command before plt.show()
        #plt.tight_layout() will also adjust spacing between subplots to minimize the overlaps
        plt.show()
```

```
<class 'matplotlib.axes._subplots.AxesSubplot'>
```



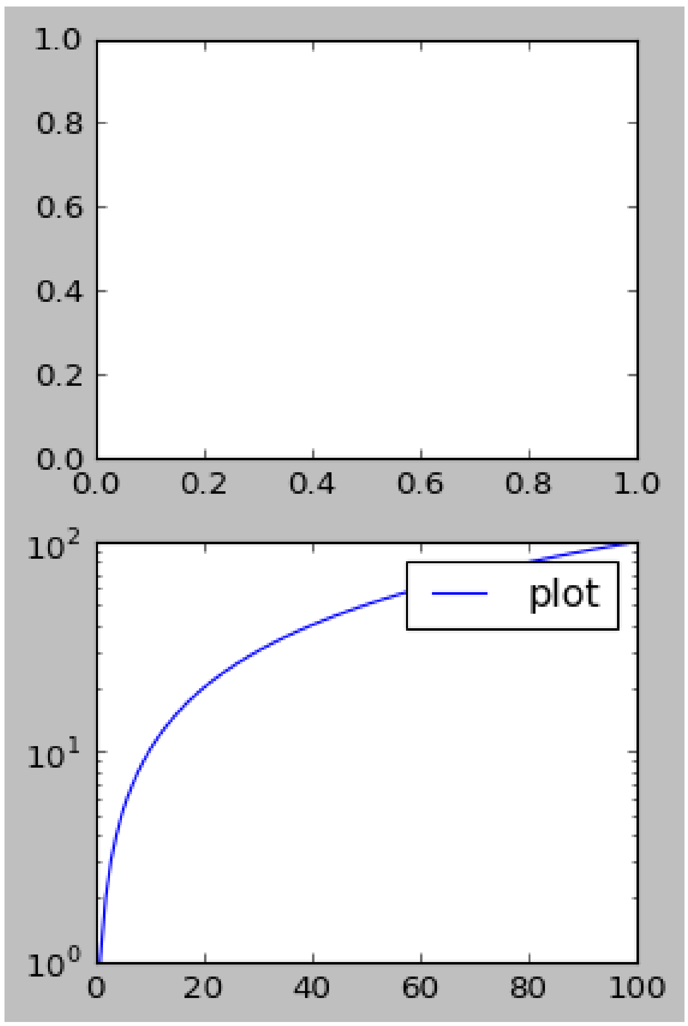

### 0.5.1 Logarithmic and other nonlinear axis

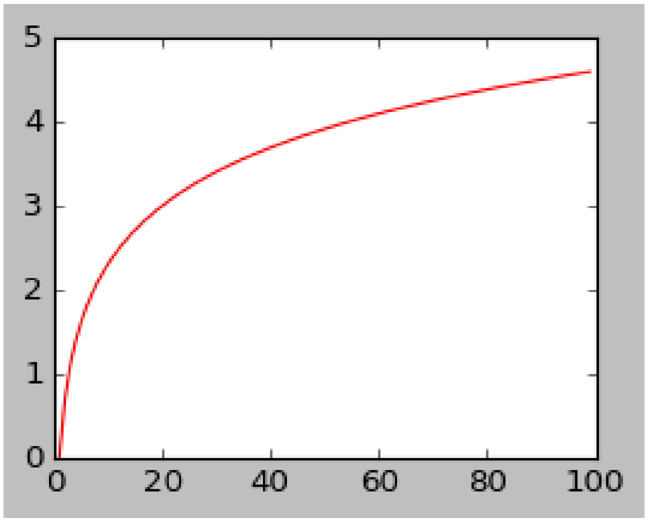
- *A logarithmic scale is a nonlinear scale used when there is a large range of quantities*
- **plt.xscale('log'), plt.xscale('linear'), plt.xscale('logit'), plt.xscale('symlog')**

In [8]:
```
import matplotlib.pyplot as plt
import numpy as np
import math
from matplotlib import style
plt.style.use('classic')

x=np.arange(0,100,1)
y= np.arange(0,100,1)
plt.figure(1)
plt.subplot(2,2,1)
ax1=plt.subplot(2,2,3)
plt.yscale('log')
line,=ax1.plot(x,y)
plt.setp(line,label="plot")
plt.legend()
plt.figure(2)
```

```
plt.subplot(2,2,2)
# plt.axis([0,9,0,10])
plt.plot(x,np.log(y),'r')
print((math.log(100)))

plt.show()
```

```
/usr/local/lib/python3.5/dist-packages/ipykernel_launcher.py:19: RuntimeWarning: divide by zero


4.605170185988092
```






# 1 Adjusting subplots

1. **subplots_adjust(left=None, bottom=None, right=None, top=None, wspace=None, hspace=None)**
   - *where,*
     - left = the left side of the subplots of the figure
     - right = the right side of the subplots of the figure
     - top = the bottom of the subplots of the figure
     - bottom = the top of the subplots of the figure
     - wspace = the amount of width reserved for blank space between subplots,

       *

## 2 expressed as a fraction of the average axis width

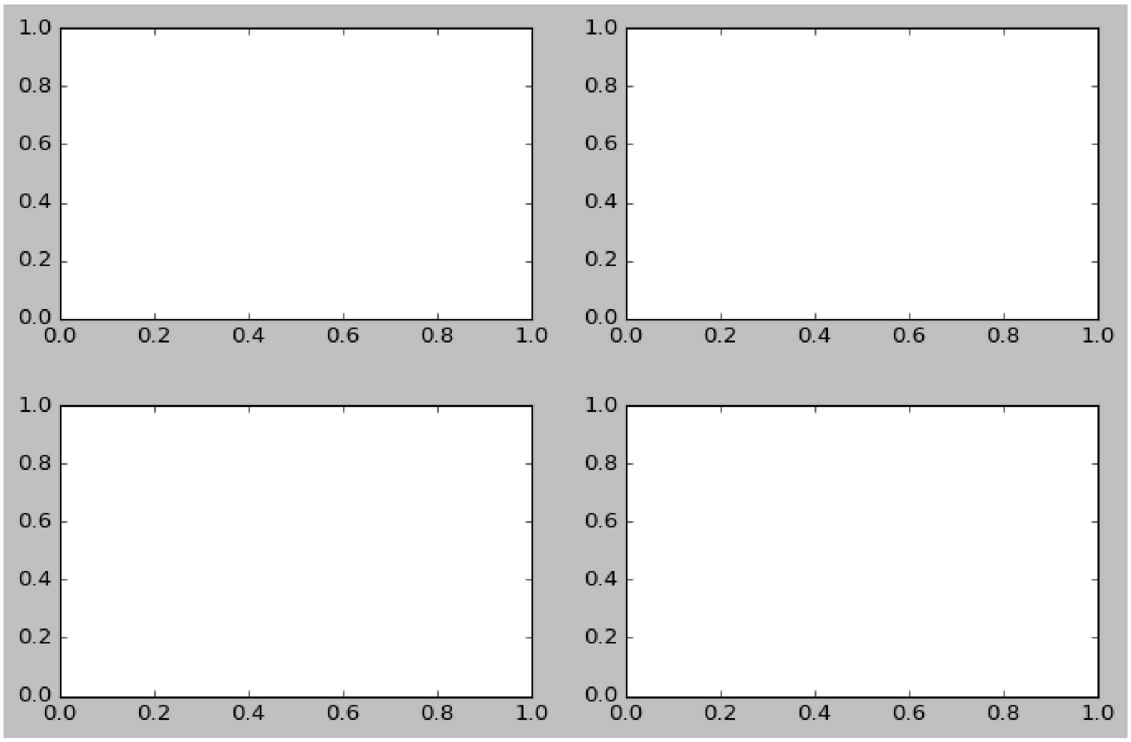
- hspace = the amount of height reserved for white space between subplots,

  *

## 3 expressed as a fraction of the average axis height

```
In [9]: import numpy as np
        import matplotlib.pyplot as plt
        import math
        from matplotlib import style
        plt.style.use('classic')

        plt.subplot(2,2,1)
```

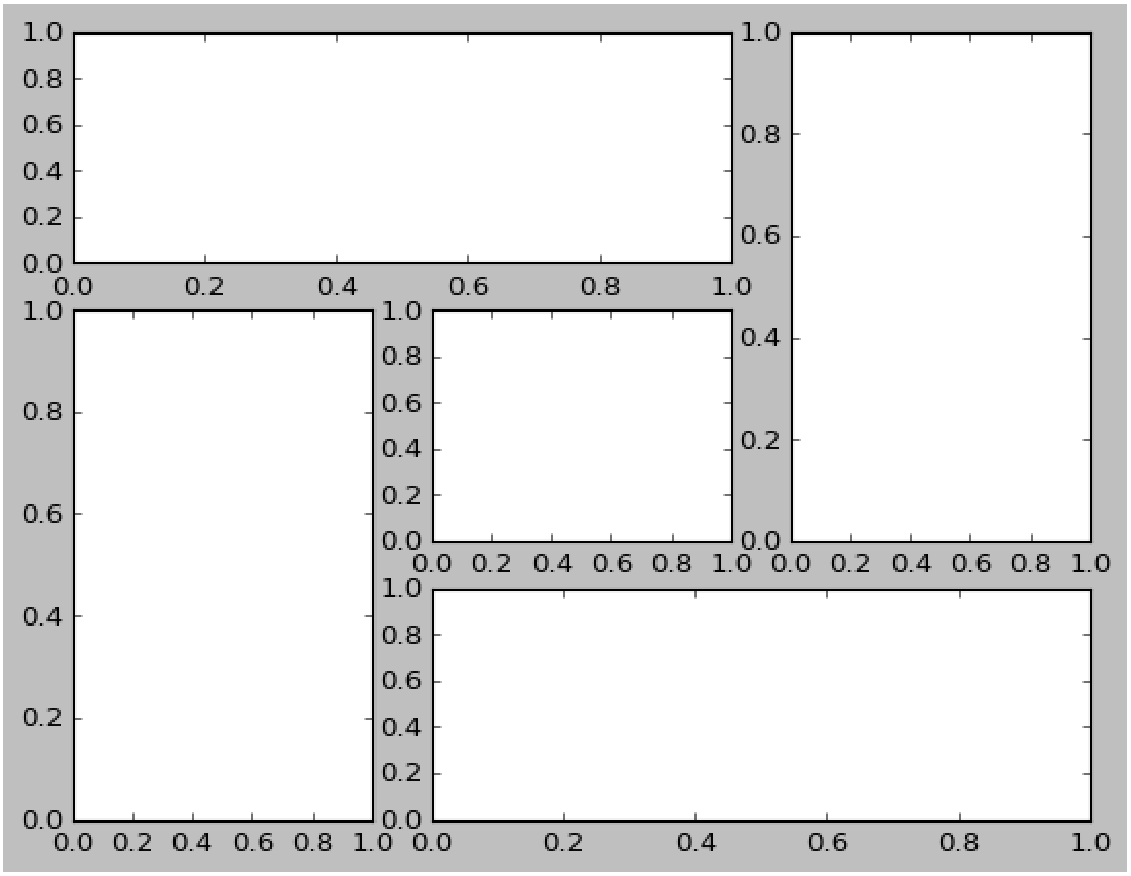
```
plt.subplot(2,2,2)
plt.subplot(2,2,3)
plt.subplot(2,2,4)
plt.subplots_adjust(left=0, bottom =.08, right=.98, top=0.92, wspace= 0.2, hspace= 0.3)
plt.show()
```




### 3.0.1 2. Customizing Location of Subplot Using subplot2grid

- **plt.subplot2grid(shape, loc, rowspan=1, colspan=1)**
  - shape of grid
  - loc-location of subplot(starting point)
  - rowspan: number of row occupied by subplot
  - colspan: number of row occupied by subplot

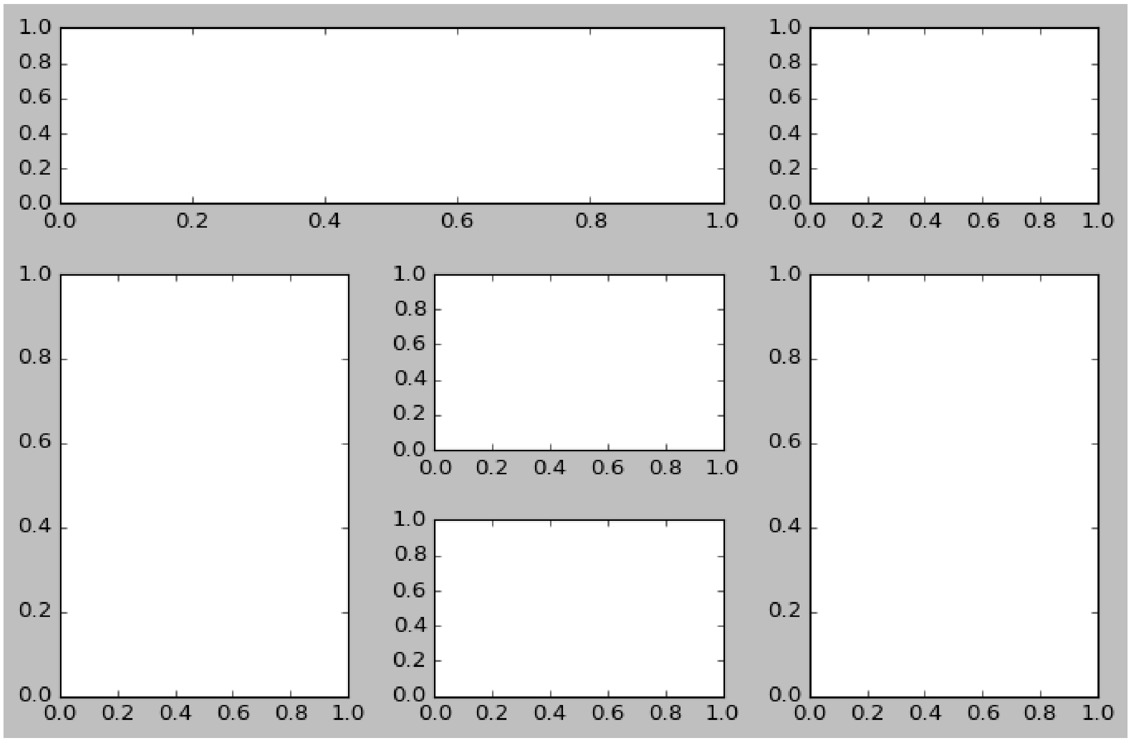
```
In [10]: import matplotlib.pyplot as plt
         from matplotlib import style
         plt.style.use('classic')

         plt.subplot2grid((3,3),(0,0), colspan=2)
         plt.subplot2grid((3,3),(0,2), rowspan=2)
         plt.subplot2grid((3,3),(1,0),rowspan=2)
         plt.subplot2grid((3,3),(1,1))
         plt.subplot2grid((3,3),(2,1),colspan=2)
         plt.show()
```



### 3.0.2 3. Customizing Location of Subplot using GridSpec and SubplotSpec

- **import matplotlib.gridspec as gridspec**
- **A gridspec instance provides array-like (2d or 1d) indexing that returns the SubplotSpec instance. For, SubplotSpec that spans multiple cells, use slice.**
- **gs= gridspec.GridSpec(3,3)**
  - gs is array
  - 3,3 is shape of grid

```
In [11]: import matplotlib.gridspec as gridspec
         gs =gridspec.GridSpec(3,3)
         ax1= plt.subplot(gs[0,:-1])
         ax2= plt.subplot(gs[0,-1])
         ax3= plt.subplot(gs[1:,0])
         ax4= plt.subplot(gs[1,1])
         ax5= plt.subplot(gs[1:,2])
         ax6= plt.subplot(gs[2,1])
         plt.subplots_adjust(left=0, bottom =.08, right=.98, top=0.92, wspace= 0.3, hspace= 0.4)
         plt.show()
```

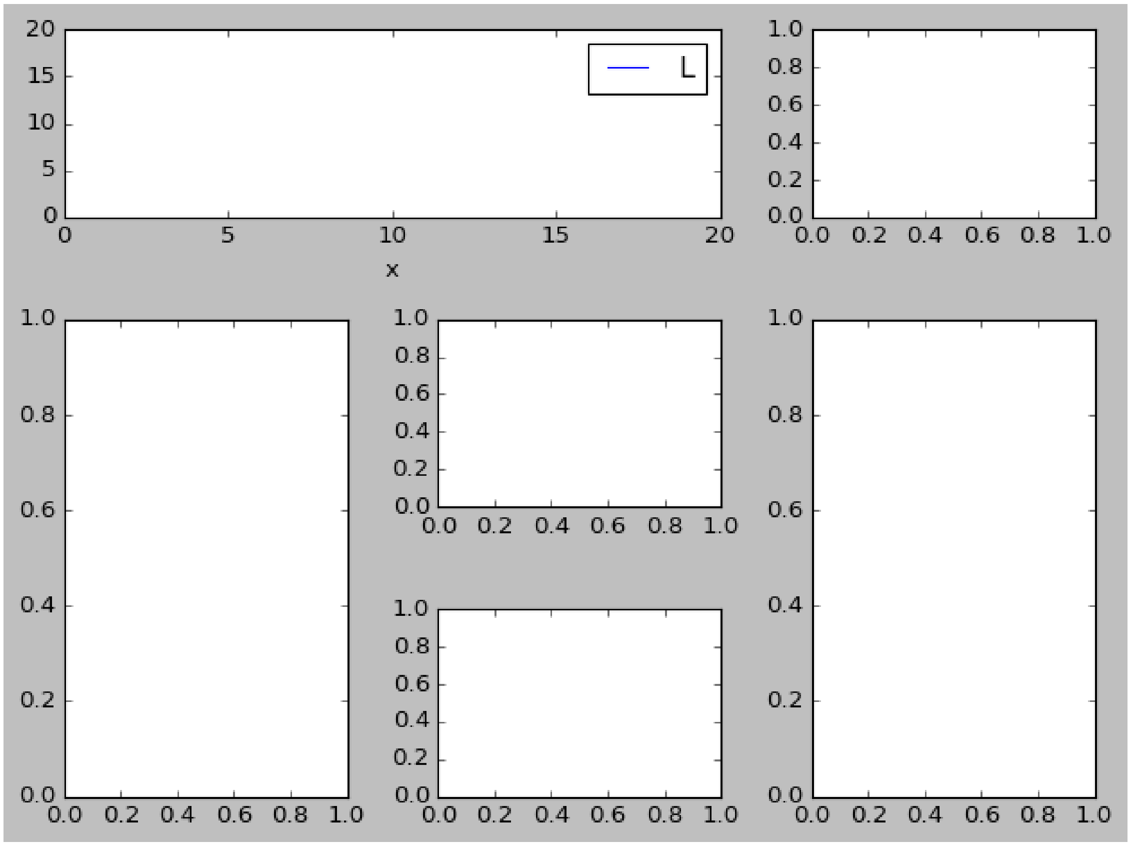



### 3.1 Tight layout guide

- It can happen that your axis labels or titles (or sometimes even ticklabels) go outside the figure area.
- tight_layout automatically adjusts subplot params so that the subplot(s) fits in to the figure area.
- **plt.tight_layout()** *will also adjust spacing between subplots to minimize the overlaps*

In [12]:
```
import matplotlib.gridspec as gridspec
import numpy as np
gs =gridspec.GridSpec(3,3)
ax1= plt.subplot(gs[0,:-1])
line,=ax1.plot(np.arange(1,20,1),np.arange(1,20,1))
line.set_label("L")

ax1.set_xlabel("x")
# plt.xlabel("x")
ax1.legend()
line.set_visible(False)
ax2= plt.subplot(gs[0,-1])
ax3= plt.subplot(gs[1:,0])
ax4= plt.subplot(gs[1,1])
ax5= plt.subplot(gs[1:,2])
ax6= plt.subplot(gs[2,1])
```

```
plt.tight_layout()
# plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=1.0)
plt.show()
```




```
In [13]: ax1.*?
```

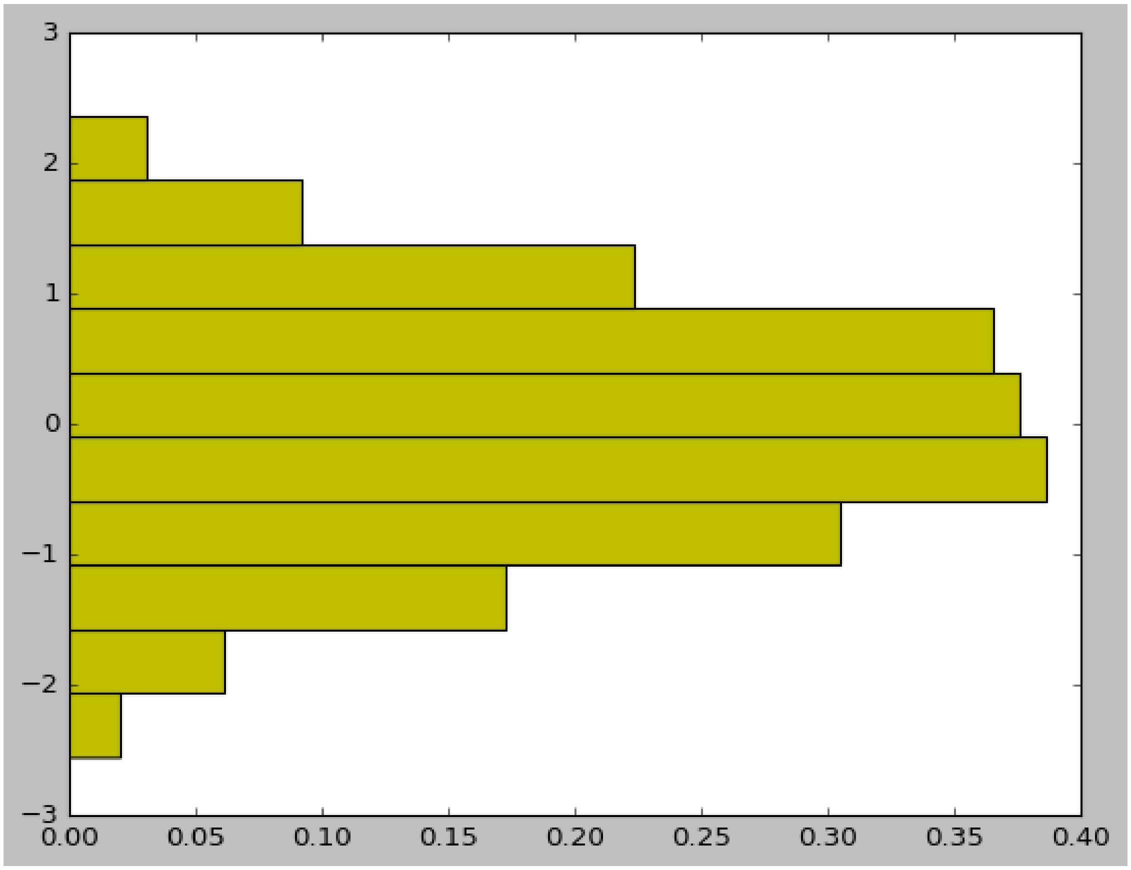
# 4 Histogram

### 4.0.1 - 1D histogram

**- h= plt.hist(x, bins=None, range=None, normed=False, weights=None, cumulative=False, bottom=None, histtype='bar', align='mid', orientation='vertical', rwidth=None, log=False, color=None, label=None, stacked=False, hold=None, data=None, **kwargs) - Except *x* all parameters are optional - x: - bins: - range: The lower and upper range of the bins - normed: - cumulative: If True, then a histogram is computed where each bin gives the counts in that bin plus all bins for smaller values - orientation: "vertical or horizontal - h: return value**

```
In [14]: import matplotlib.pyplot as plt
         import numpy as np
         import cv2
         from matplotlib import style
```

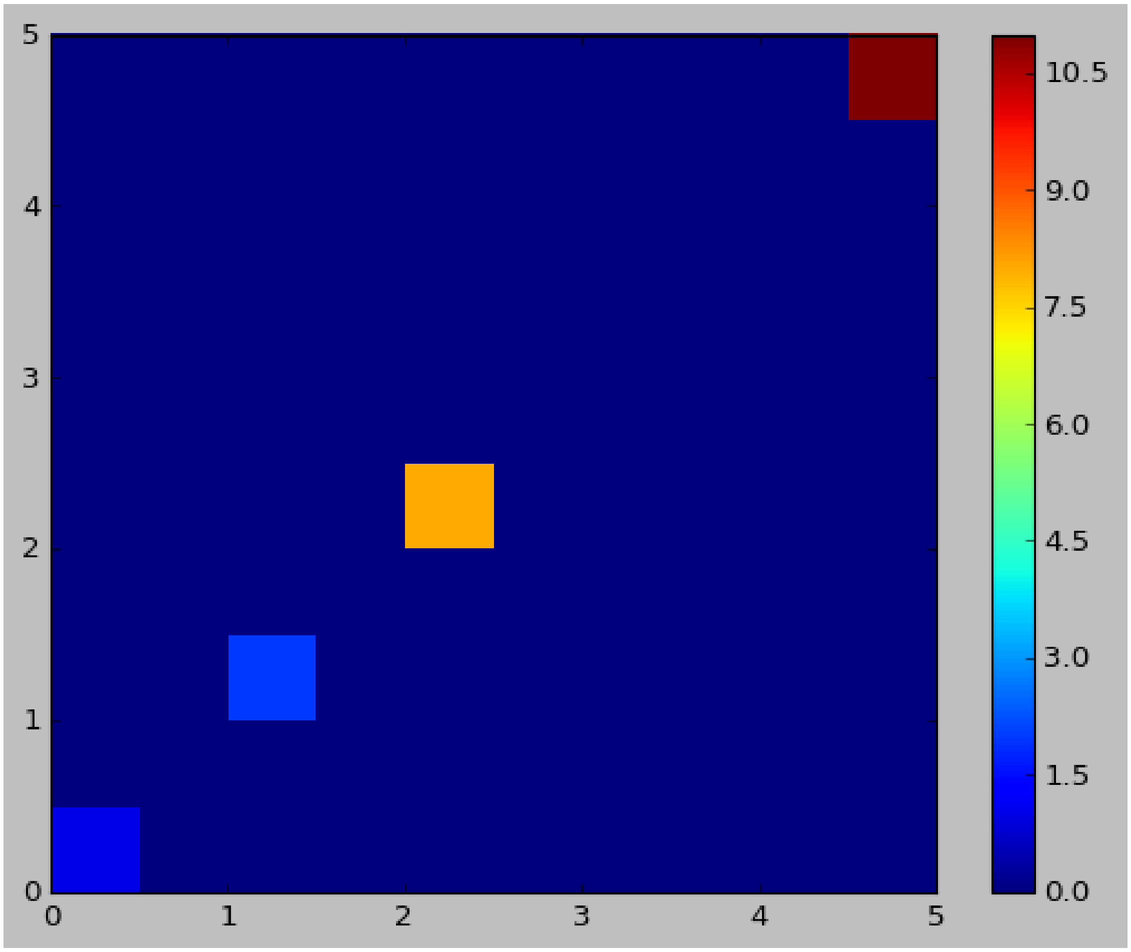
```
plt.style.use('classic')
x=np.random.randn(200)
plt.hist(x,bins=10, color="y",normed=1,orientation="horizontal")
plt.show()
```




**2D histogram**

- ** h = plt.hist2d(x, y, bins=10, range=None, normed=False, weights=None, cmin=None, cmax=None, hold=None, data=None, **kwargs)
- **h** is return value = (counts, xedges, yedges, Image).

```
In [15]: import matplotlib.pyplot as plt
         import numpy as np
         from matplotlib import style
         plt.style.use('classic')
         x=np.array([0,1,2,2,2,1,2,2,2,2,2,5,5,5,5,5,5,5,5,5,5,5])
         y=np.array([0,1,2,2,2,1,2,2,2,2,2,5,5,5,5,5,5,5,5,5,5,5])
         h=plt.hist2d(x,y)
         plt.colorbar(h[3])
         plt.show()
```




```
In [16]: plt.*hist*?
```

## 4.1 3Dimensional plot

### 4.1.1 1.

- **plt.figure**(figure_number,kargs) returns **"class 'matplotlib.figure.Figure"**. Now kargs can be the all attributes in **"class 'matplotlib.figure.Figure"**. Here are attributes
  - So **kargs** can be following - 1.facecolor - 2. fig.add_subplot - 3. no attribute fig.subplot()
- **plt.gca(kargs)** AND **plt.subplot(num_row,num_column,num_fig, kargs)** return ** "class'matplotlib.axes._subplots.Axes3DSubplot"**OR** "class 'matplotlib.axes._subplots.AxesSubplot"** depending on parameter "projection"
  - Now kargs can be all attributes in **"class 'matplotlib.axes._subplots.AxesSubplot"**. Here are attributes
  - So **kargs** can be following
    - projection
    - axisbg

    * title
    * xlabel
    * ylabel
    * xscale
    * yscale
  – If ax=plt.gca()(**OR fig.gca where fig=plt.figure**) ax=plt.subplot()(**fig.subplot is wrong**).
    * Then we can use these things **AND** as you know we can use * **plt** instead of ax* So below statements are also applicable for **plt**.

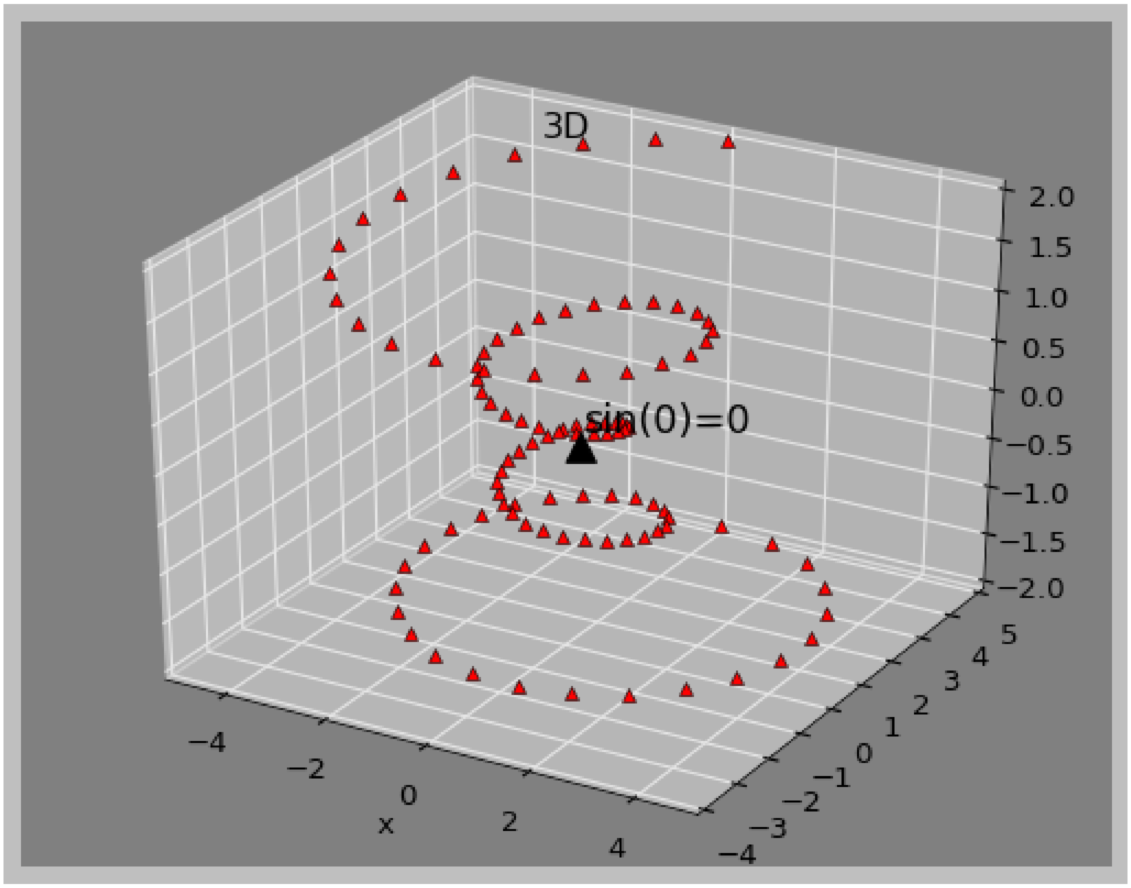
| ax | plt |
|---|---|
| **ax.annotate** | **plt.annotate** |
| **ax. text** | **plt.text** |
| **ax.set_title** | **plt.title** |
| **ax.set_xlabel** | **plt.xlabel** |
| **ax.legend** | **plt.legend** |
| **ax.set_xscale** | **plt.xscale** |
| **ax.hist** | **plt.hist** |
| **ax.hist2d** | **plt.hist2d** |
| **ax.pie** | **plt.pie** |
| **ax.plot** | **plt.plot** |
| **ax.scatter** | **plt.scatter** |
| **ax.plot_wireframe** | **NOT APPLICABLE** |

- **plt.plot(krags)** OR **ax.plot(kargs)** return ""*class 'matplotlib.lines.Line2D"* Now krags can be all attributes supported in *class 'matplotlib.lines.Line2D* Here are attributes
  – So krgs can be following:
    * linewidth
    * linestyle
    * color
    * marker
    * markersize *or* ms
    * label
    * markerfacecolor
    * markeredgecolor
  – line,=ax.plot() or line,=plt.plot() then We can use plt.setp() to change the chrcteristics of plot using above krgs.
    * plt.setp(line,color='r', label='line1', ms=3)

In [17]:
```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
from matplotlib import style
plt.style.use('classic')
theta = np.linspace(-4 * np.pi, 4 * np.pi, 100)
z = np.linspace(-2, 2, 100)
```

```
r = z**2 + 1
x = r * np.sin(theta)
y = r * np.cos(theta)
fig= plt.figure(1)
print(type(fig))
ax= plt.gca(projection="3d",title="3D1",xlabel="x",axisbg='gray')
# OR ax= fig.gca()
print(type(ax))
line,=plt.plot(x,y,z,'^' ,color='r')
ax.annotate("sin(0)=0", xy=(0,0), xytext=(0,0), arrowprops=dict(facecolor="black", shri
# plt.setp(line,marker='o')
ax.set_title("3D")
plt.show()

print(type(fig.gca()))
```

```
<class 'matplotlib.figure.Figure'>
<class 'matplotlib.axes._subplots.Axes3DSubplot'>
```



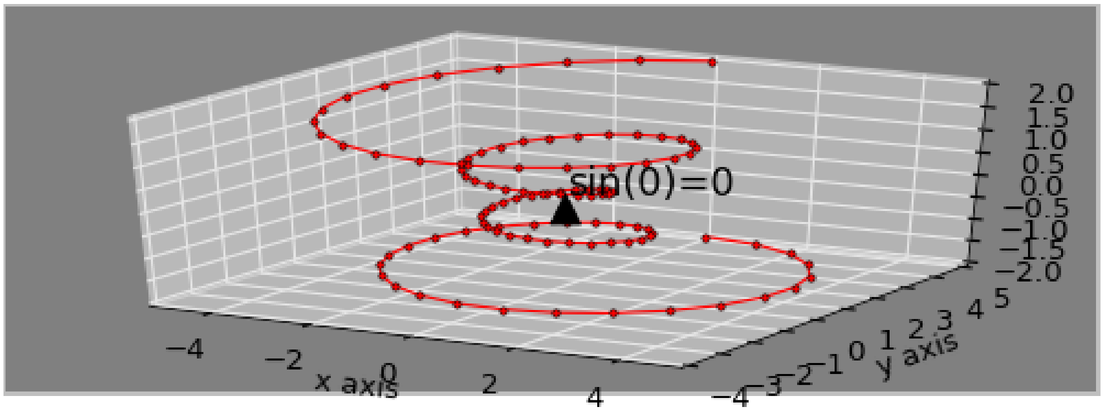
```
<class 'matplotlib.axes._subplots.Axes3DSubplot'>
```

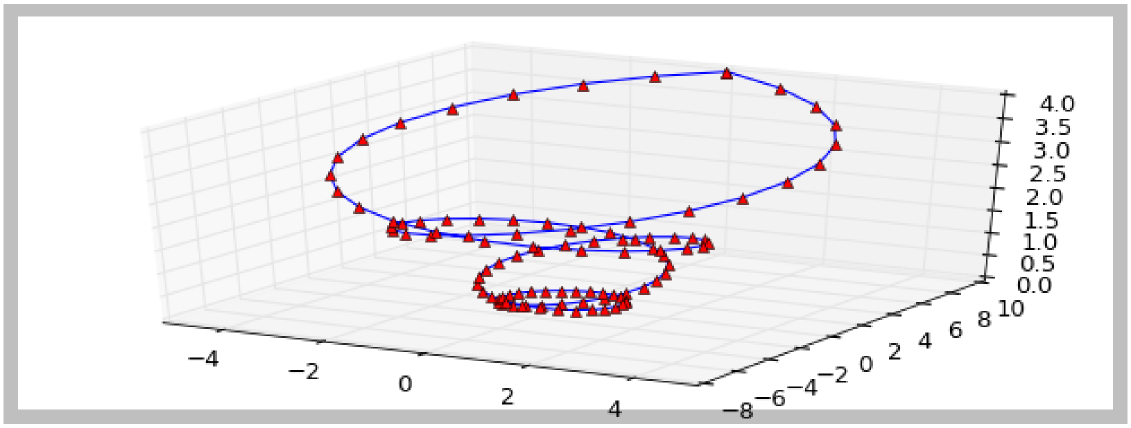
#### 4.1.2 2. plt.subplot(nrows, ncols, plot_number, kargs) - krgs

```
- projection (like projecion="3d")
- axisbg : changes background of plot, e.g axisbg="y".
```

```
In [18]: import matplotlib.pyplot as plt
         import numpy as np
         from mpl_toolkits.mplot3d import Axes3D
         from matplotlib import style
         plt.style.use('classic')

         theta = np.linspace(-4 * np.pi, 4 * np.pi, 100)
         z = np.linspace(-2, 2, 100)

         r = z**2 + 1
         x = r * np.sin(theta)
         y = r * np.cos(theta)
         fig=plt.figure(1,linewidth=10)
         ax=fig.add_subplot(211,projection="3d",axisbg='gray')
         # ax=fig.add_subplot
         plt.xlabel('x axis')
         plt.ylabel('y axis')
         print(type(ax))
         # fig.set_facecolor('y')
         ax.plot(x,y,z,color="red",marker='o',ms=3)
         ax.annotate("sin(0)=0", xy=(0,0), xytext=(0,0), arrowprops=dict(facecolor="black", shri
         ax1=plt.figure()
         ax1=ax1.add_subplot(212,projection="3d",axisbg='white')
         ax1.plot(x,y*2,z**2,color="b",marker='^',ms=5,markerfacecolor='r')
         plt.tight_layout(h_pad=1)
         plt.show()
```

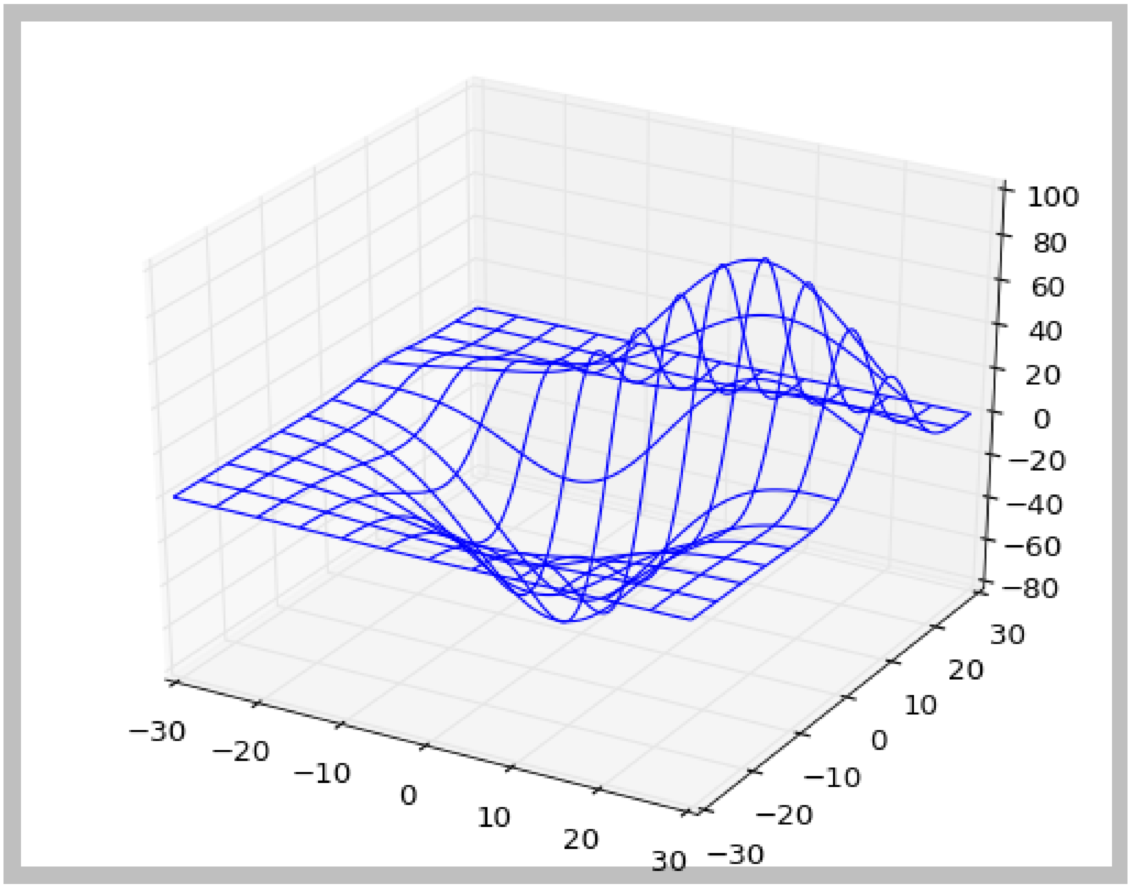
```
<class 'matplotlib.axes._subplots.Axes3DSubplot'>
```

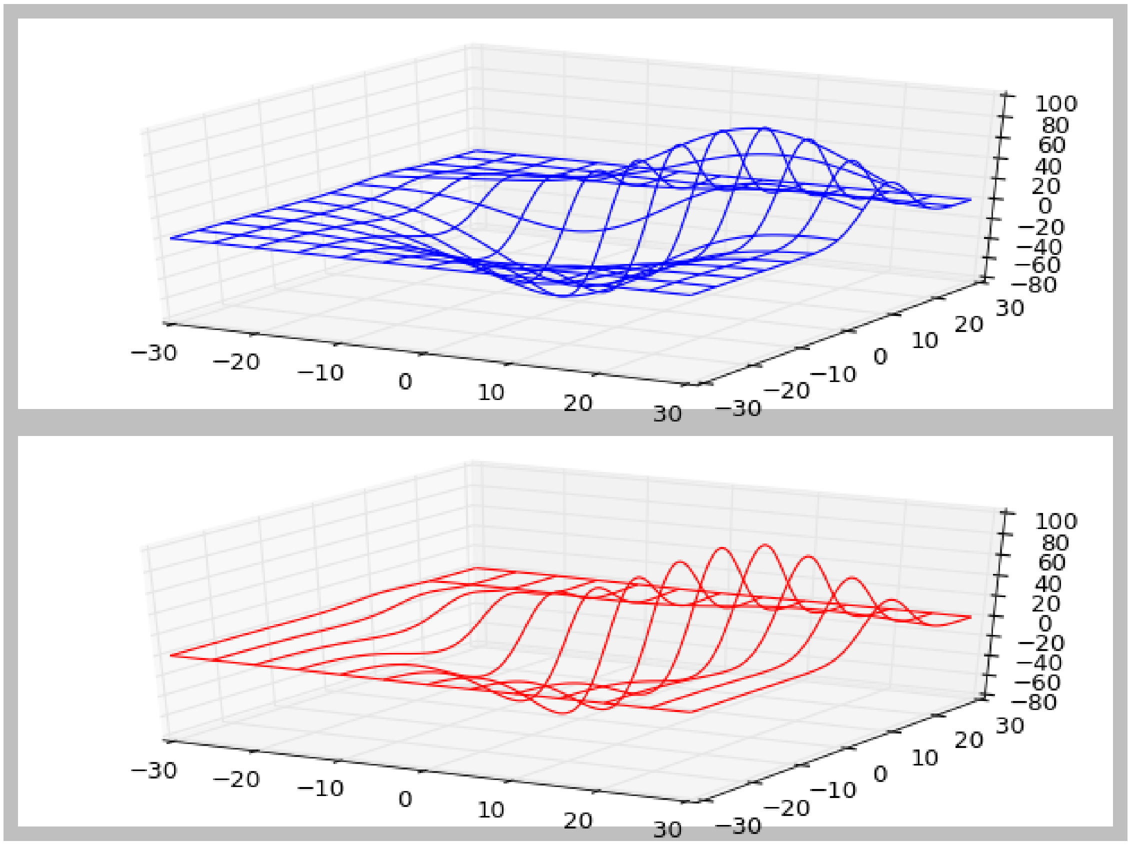





#### 4.1.3 3. ax.plot_wireframe(X,Y,Z, **kargs) - kargs can be following: - rstride : Array row stride (step size), defaults to 1 - cstride : Array column stride (step size), defaults to 1 - NOTE: We can't use plt.plot_wireframe.

In [19]:
```
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
from matplotlib import style
plt.style.use('classic')

fig = plt.figure(1)
ax = fig.gca(projection='3d')

# Grab some test data.
X, Y, Z = axes3d.get_test_data(0.05)

# Plot a basic wireframe.
ax.plot_wireframe(X, Y, Z, rstride=10, cstride=10)  ####X, Y,        Data values as 2D

fig2= plt.figure(2)
ax1= plt.subplot(211, projection='3d')
ax1.plot_wireframe(X, Y, Z, rstride=10, cstride=10)

ax2= plt.subplot(212, projection='3d')
ax2.plot_wireframe(X, Y, Z, rstride=100, cstride=10, color='r')

#NOTE: We can't use plt.plot_wireframe.
plt.tight_layout()

plt.show()
```


100
80
60
40
20
0
−20
−40
−60
−80
−30
−20
−10
0
10
20
30
−30
−20
−10
0
10
20
30

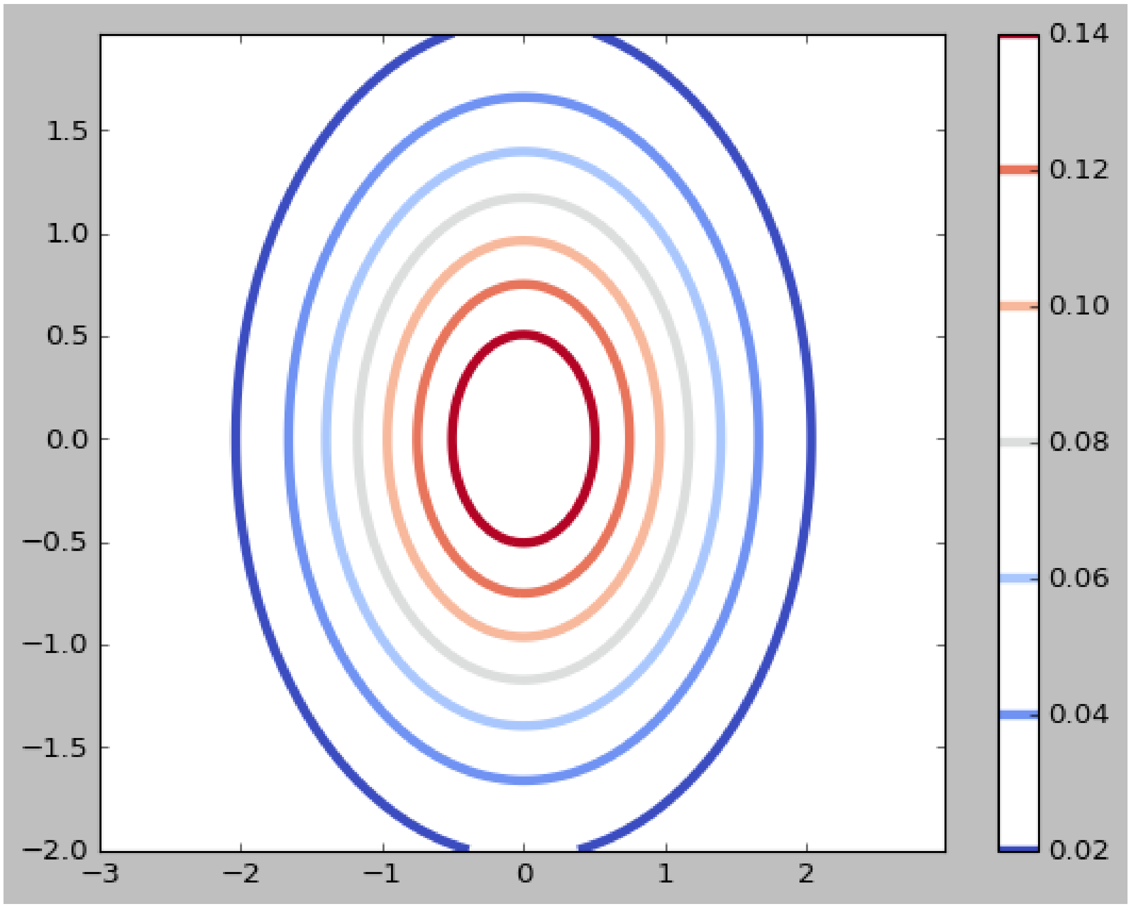



### 4.1.4 Contours

- 1. CS= plt.contour(X,Y,Z, kargs)
  - kargs can be folloeing:
    - cmap: A cm Colormap instance or None. If cmap is None and colors is None, a default Colormap is used.
  - Adding colorbar - plt.colorbar(CS)

In [20]:
```
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.cm as cm
import matplotlib.mlab as mlab


delta = 0.025
x = np.arange(-3.0, 3.0, delta)
y = np.arange(-2.0, 2.0, delta)
X, Y = np.meshgrid(x, y)
Z1 = mlab.bivariate_normal(X, Y, 1.0, 1.0, 0.0, 0.0)
CS=plt.contour(X,Y,Z1,cmap=cm.coolwarm, linewidths=4)
```

```
plt.colorbar(CS)
plt.show()
```




```
In [21]: cm.*?
```

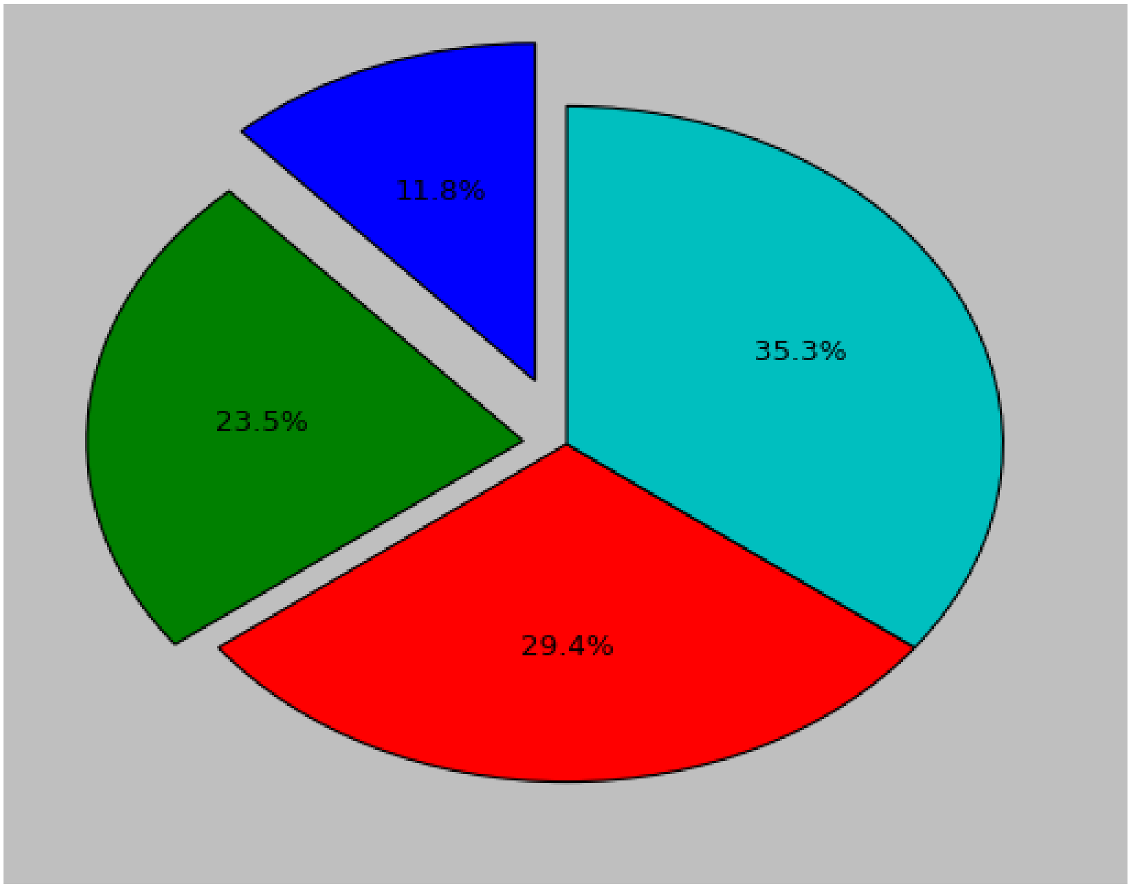
### 4.1.5 PIE CHART

### 4.1.6 plt.pie(or ax.pie)(x, kargs)

```
- x:data to be represented
- kargs can be following:
    - colors : array of colors
    - labels : label should be a ist of length len(x)
    - shadow :
    - explode:
    - autopct: shows percentage...always set to '%1.1f%%'
    - startangle: angle from where pie chartis started
```

```
In [22]: import numpy as np
         import matplotlib.pyplot as plt
```

```
from matplotlib import style
plt.style.use('classic')
plt.pie(np.array([2,4,5,6]), autopct='%1.1f%%',explode=(0.2,0.1,0,0),startangle=90)
plt.show()
```




END