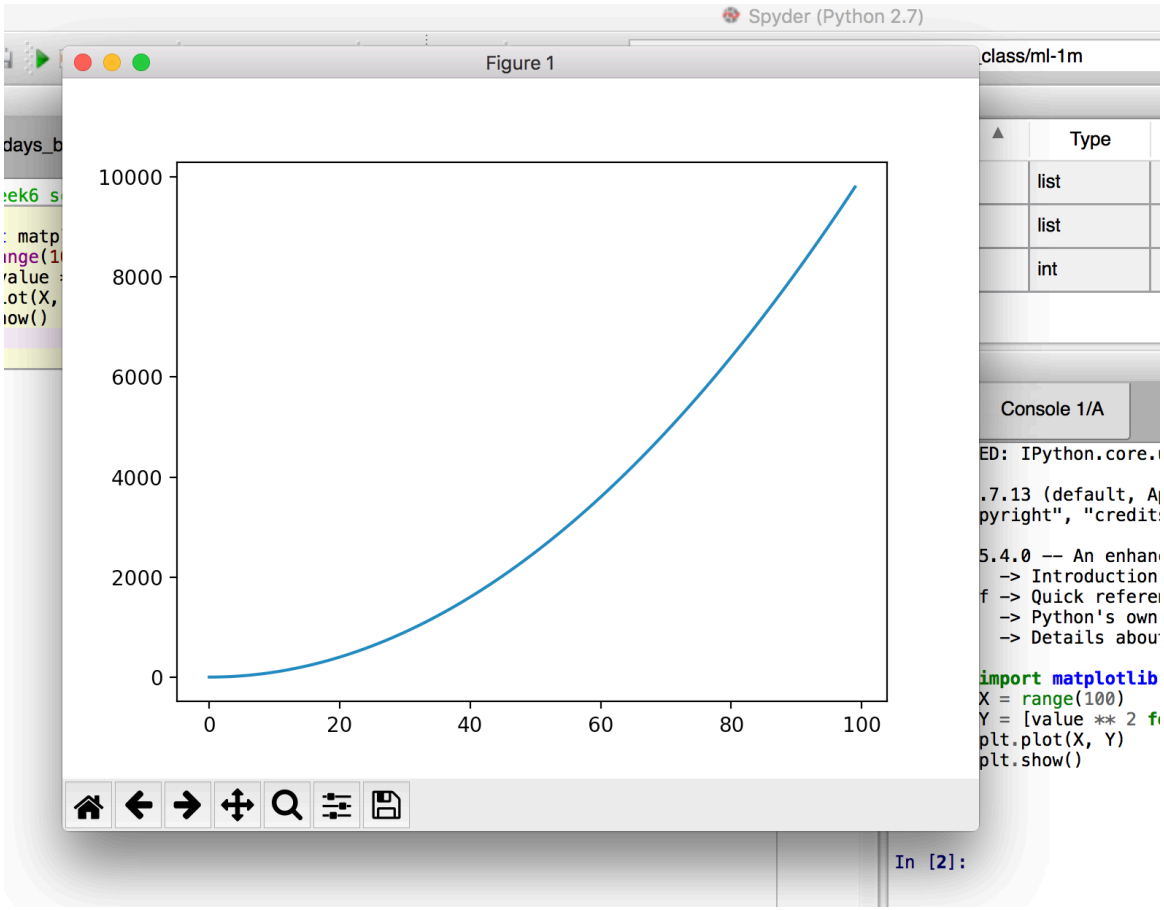
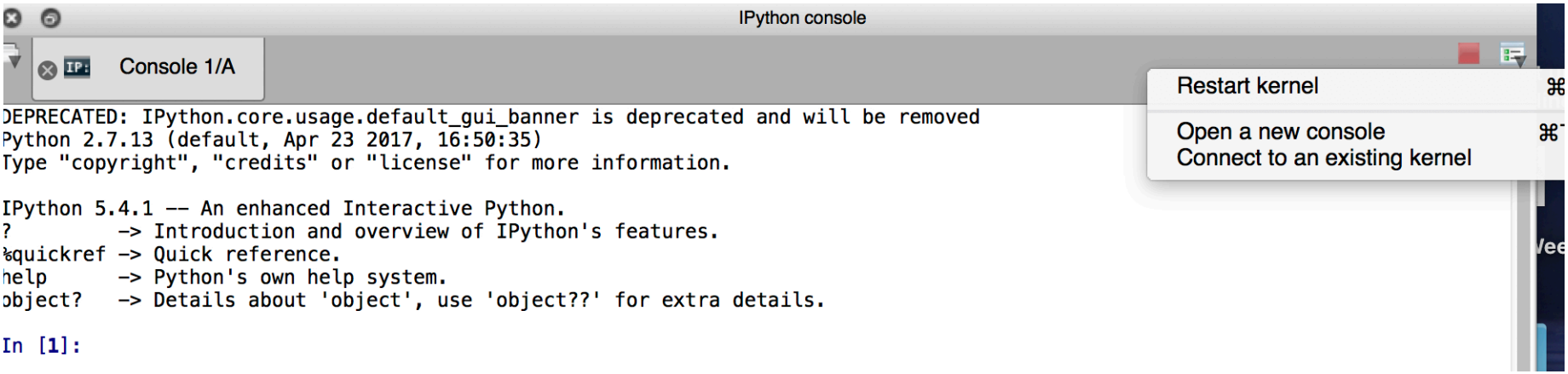
Matplotlib Applications

```
In [1]: import matplotlib.pyplot as plt
   ...: X = range(100)
   ...: Y = [value ** 2 for value in X]
   ...: plt.plot(X, Y)
   ...: plt.show()
   ...:
```



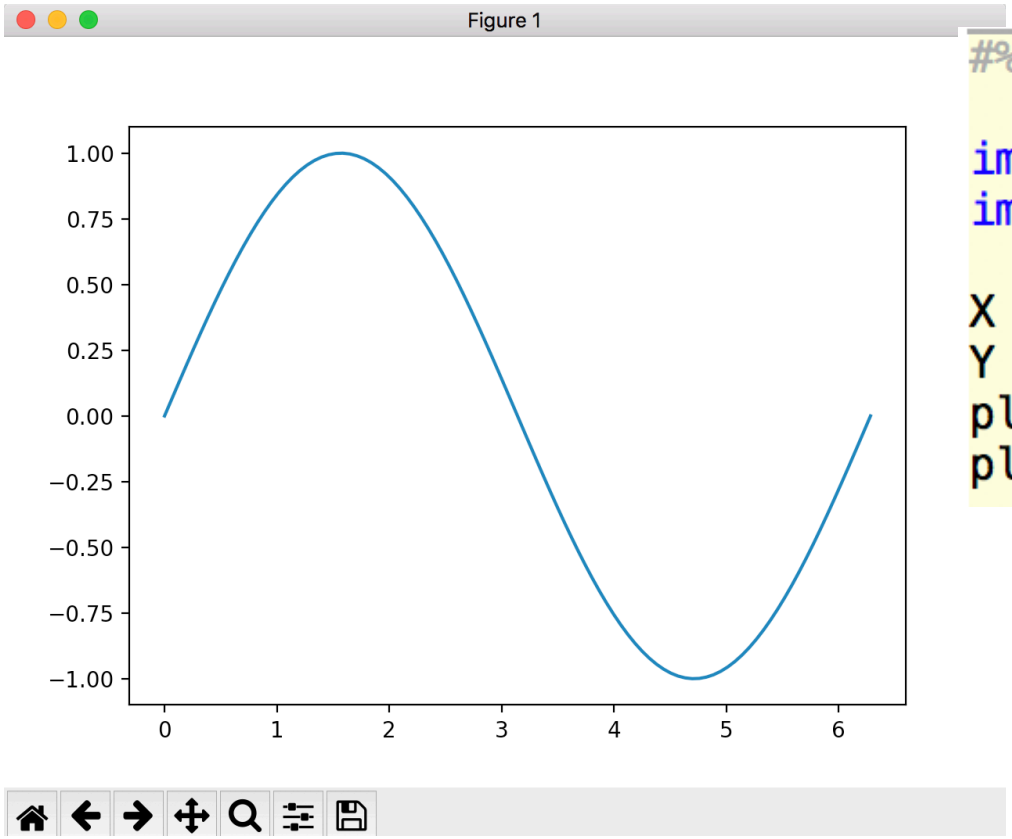
Matplotlib.pyplot

- In spyder, if you want to show plots not inside the console, go to options,Ipython console,Graphics and from Graphics backend menu select Automatic.
- If the plot is still produced in the console restart kernel of the console.



Producing plots using numpy

- np.linspace returns evenly spaced numbers of a specified interval.

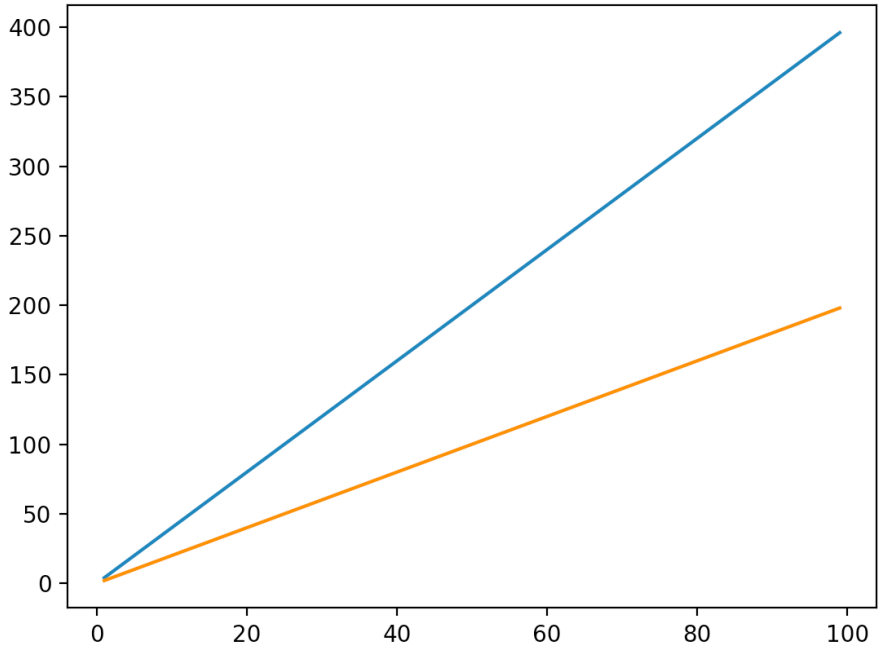



```
#%%

import numpy as np
import matplotlib.pyplot as plt

X = np.linspace(0, 2 * np.pi, 100)
Y = np.sin(X)
plt.plot(X, Y)
plt.show()
```

- Np.arange is also commonly used to create data on X axis.

Plotting multiple curves on the same figure

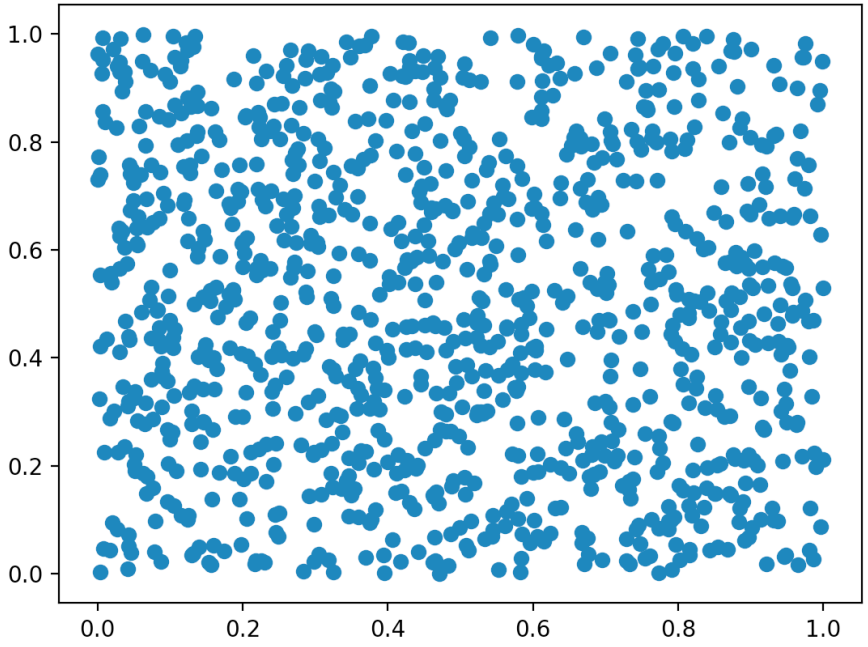
- You can declare what you render as and when it suits you.
- The graph will be rendered only when you call plt.show().
- The two curves show up with a different color automatically picked up by matplotlib.




```
import numpy as np
import matplotlib.pyplot as plt

X =np.arange(1,100)
Y=X*4
Z=X*2
plt.plot(X, Y)
plt.plot(X,Z)
plt.show()
```

Plotting points: Scatter plot

- Previous example was a time-series plot. Here we plot two variables together.

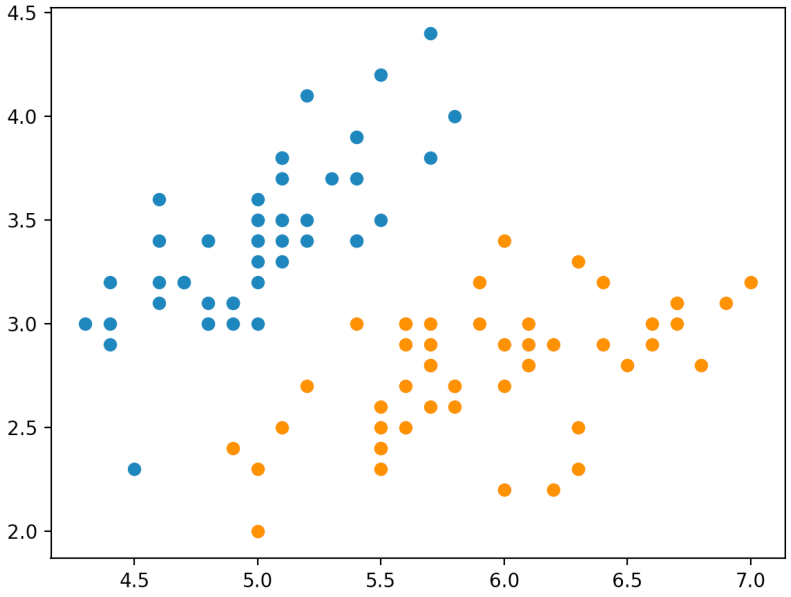



```
#%%
#scatter plot example
import numpy as np
import matplotlib.pyplot as plt
data = np.random.rand(1024, 2)
plt.scatter(data[:,0], data[:,1])
plt.show()
```

Plotting points: Scatter plot

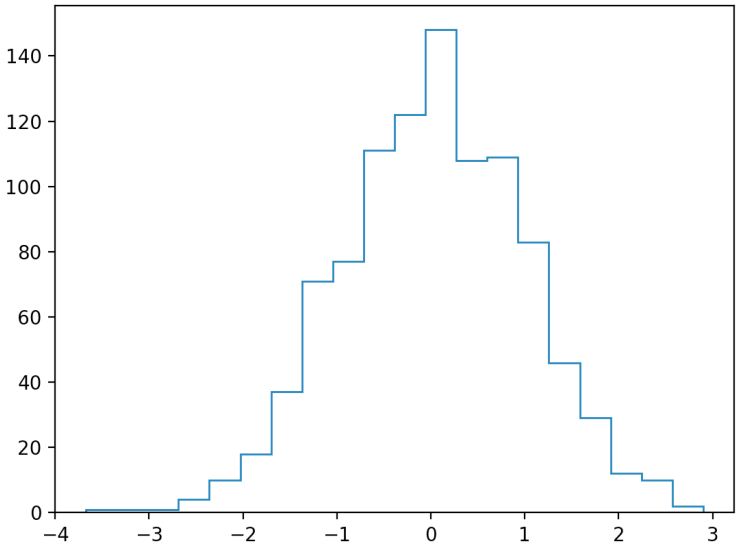
- Scatter plot of iris dataset.

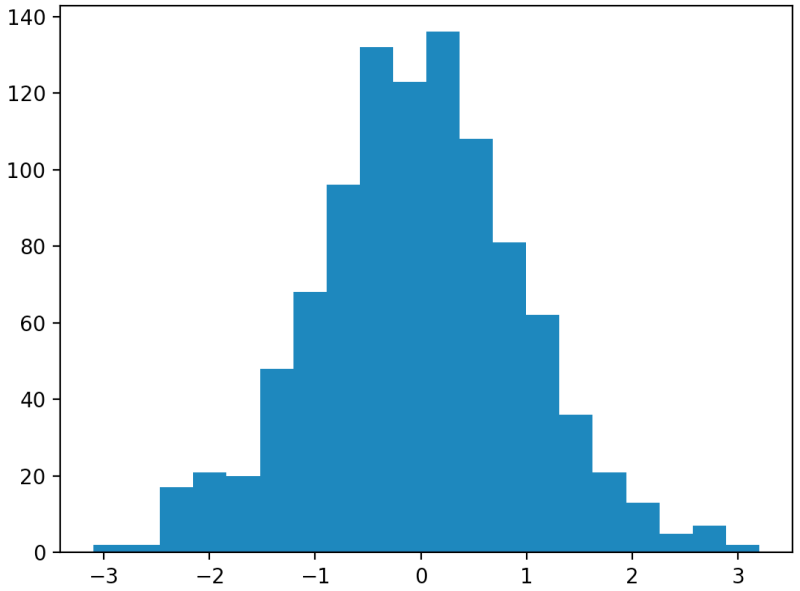
```
#%%
#scatter plot of iris data
import pandas as pd
import matplotlib.pyplot as plt
url="https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
names=["sepal-length","sepal-width","petal-length","petal-width","class"]
dataset=pd.read_csv(url,names=names)

x1=dataset[dataset["class"]=="Iris-setosa"]["sepal-length"]
y1=dataset[dataset["class"]=="Iris-setosa"]["sepal-width"]
x2=dataset[dataset["class"]=="Iris-versicolor"]["sepal-length"]
y2=dataset[dataset["class"]=="Iris-versicolor"]["sepal-width"]
plt.scatter(x1,y1)
plt.scatter(x2,y2)
```



Plotting histograms

- Histograms are graphical representations of a probability distribution.
- In fact, a histogram is just a specific kind of a bar chart.

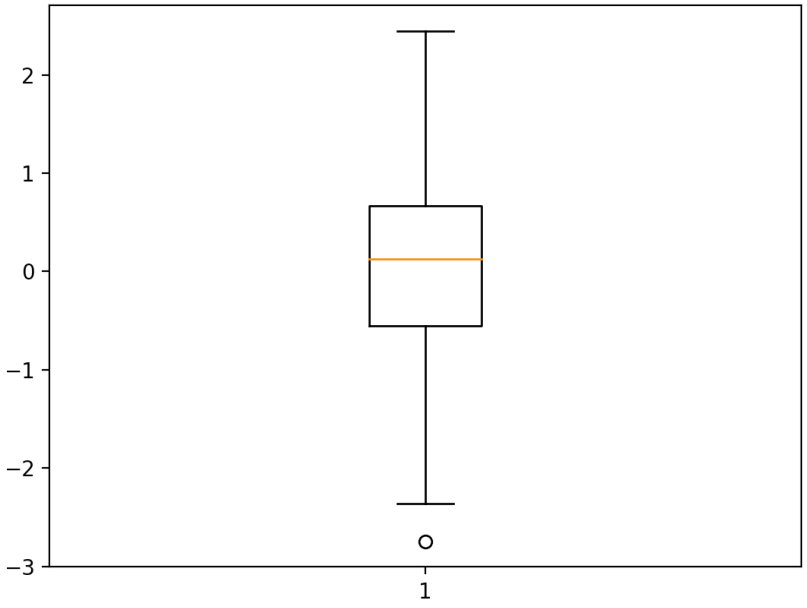


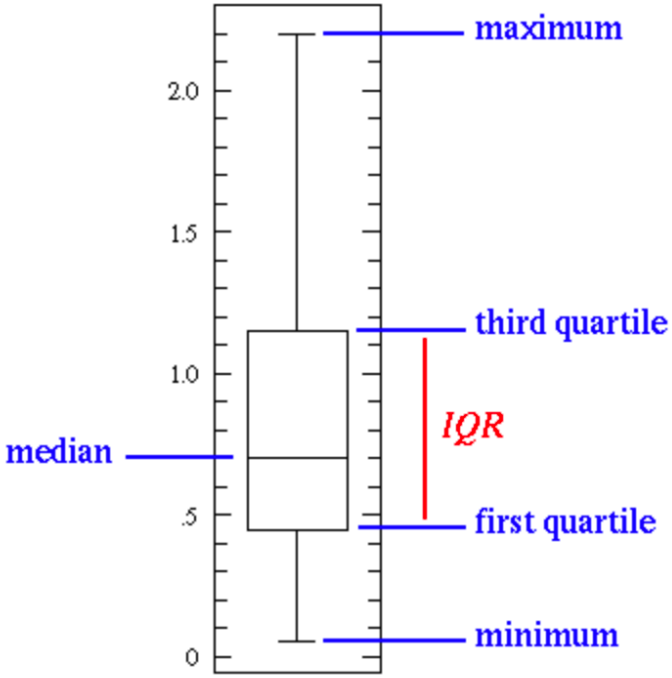



```
#histogram plots
import numpy as np
import matplotlib.pyplot as plt
X = np.random.randn(1000)
#plt.hist(X, bins = 20,histtype="step")
plt.hist(X, bins = 20)
plt.show()
```

Plotting boxplots

- Boxplot allows you to compare distributions of values by conveniently showing the median, quartiles, maximum, and minimum of a set of values.

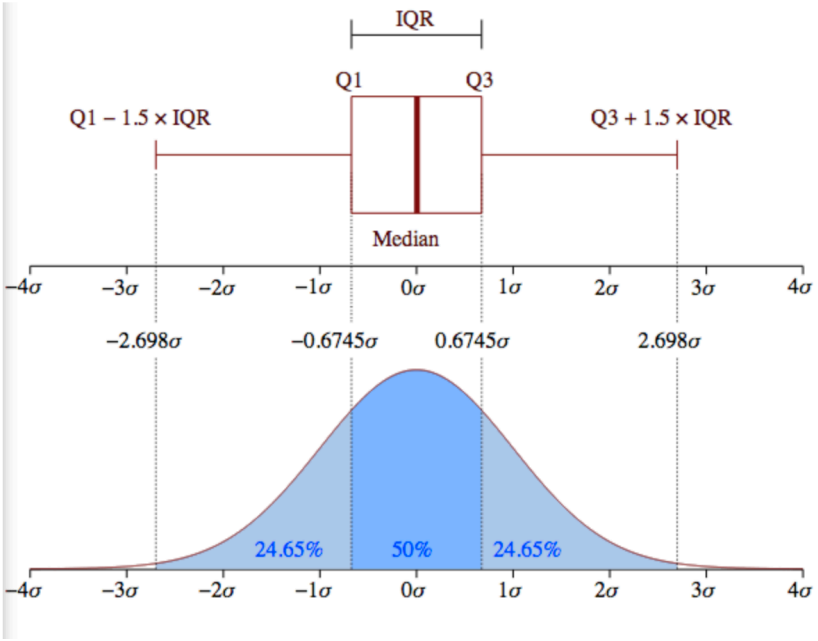




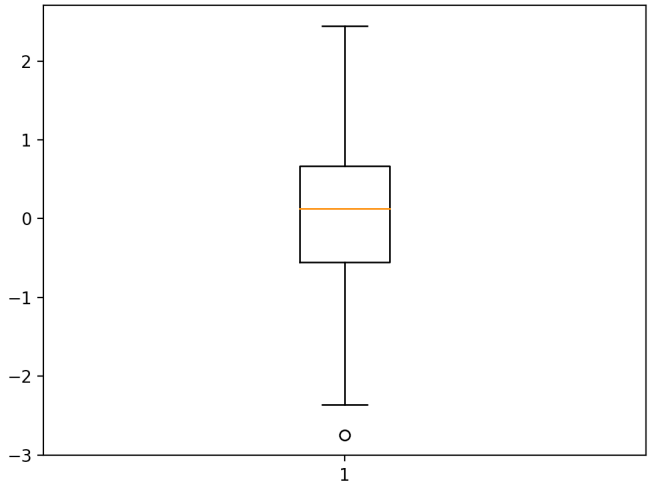

```
#%%
#boxplot example
import numpy as np
import matplotlib.pyplot as plt
data = np.random.randn(100)
h=plt.boxplot(data)
plt.show()
```

Plotting boxplots

- Boxplot allows you to compare distributions of values by conveniently showing the median, quartiles, maximum, and minimum of a set of values.



- The middle "box" represents the middle 50% of values for the dataset.
- The range of scores from lower to upper quartile is referred to as the inter-quartile range (IQR).
- Whiskers. Q-1.5xIQR , Q3+1.5x1QR
- Points staying outside whiskers are assumed to be outliers.



Plotting boxplots with pandas

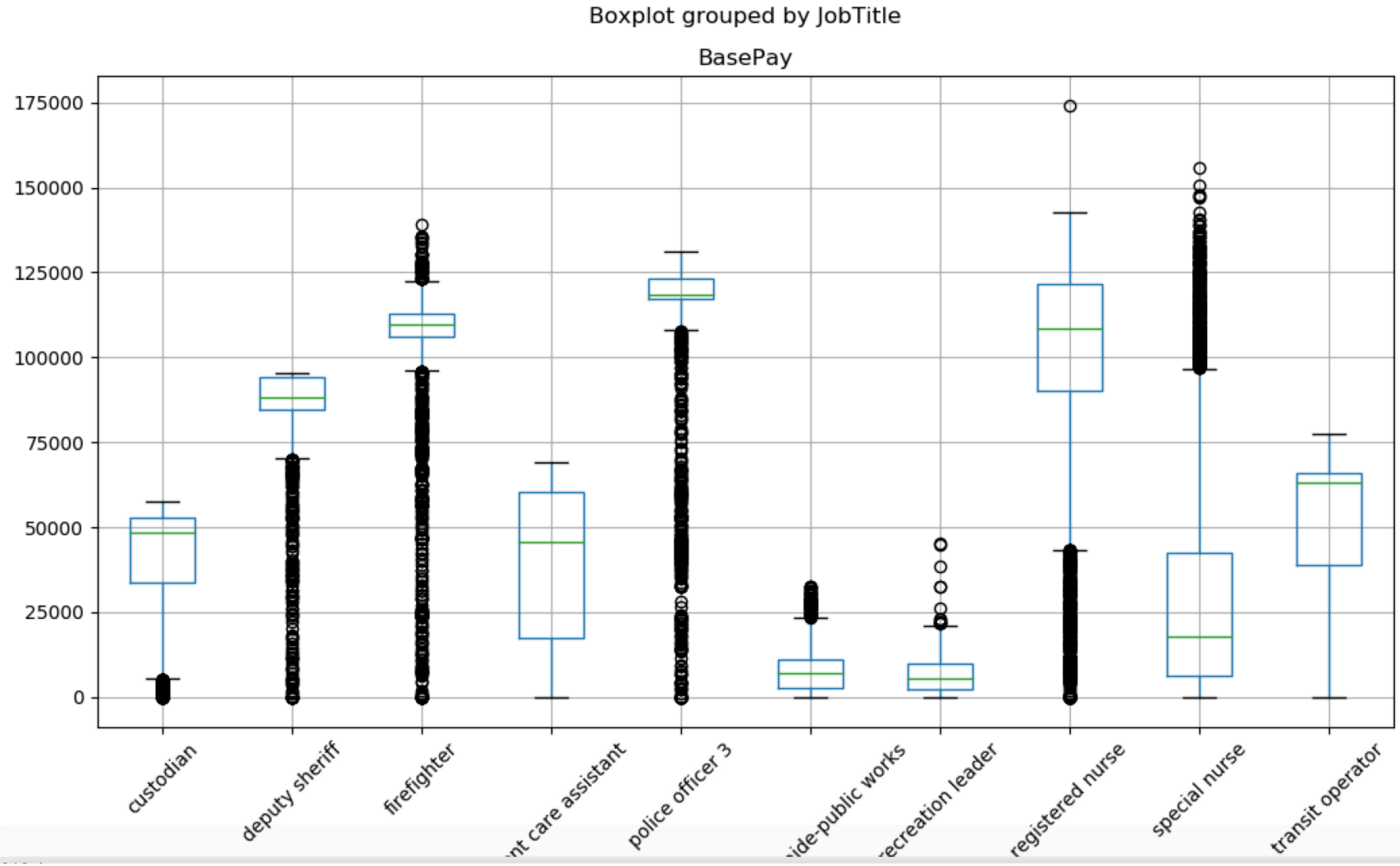
- Let's plot Basepay ditributions for SF salary data.
- Suppose we have a data_10jobtitle dataframe for 10 most frequent job title.

```
In [20]: data_10jobtitle.head()
Out[20]:
      Id      EmployeeName     JobTitle    BasePay  OvertimePay  OtherPay  \
43    44  michael thompson  firefighter  123013.02    111729.65  15575.26
102  103   lauifi seumaala  firefighter  105934.69     98534.35  18890.96
104  105     patric steele  firefighter  105934.64     97395.59  18760.77
105  106     michael walsh  firefighter  110474.93     83670.04  27043.61
109  110    scott scholzen  firefighter  105934.67     96154.33  18655.85

     Benefits   TotalPay  TotalPayBenefits  Year  Notes         Agency Status
43        NaN  250317.93         250317.93  2011    NaN  San Francisco    NaN
102       NaN  223360.00         223360.00  2011    NaN  San Francisco    NaN
104       NaN  222091.00         222091.00  2011    NaN  San Francisco    NaN
105       NaN  221188.58         221188.58  2011    NaN  San Francisco    NaN
109       NaN  220744.85         220744.85  2011    NaN  San Francisco    NaN
```

Plotting boxplots with pandas

- We want to plot boxplot for the basepay salaries for each job title.

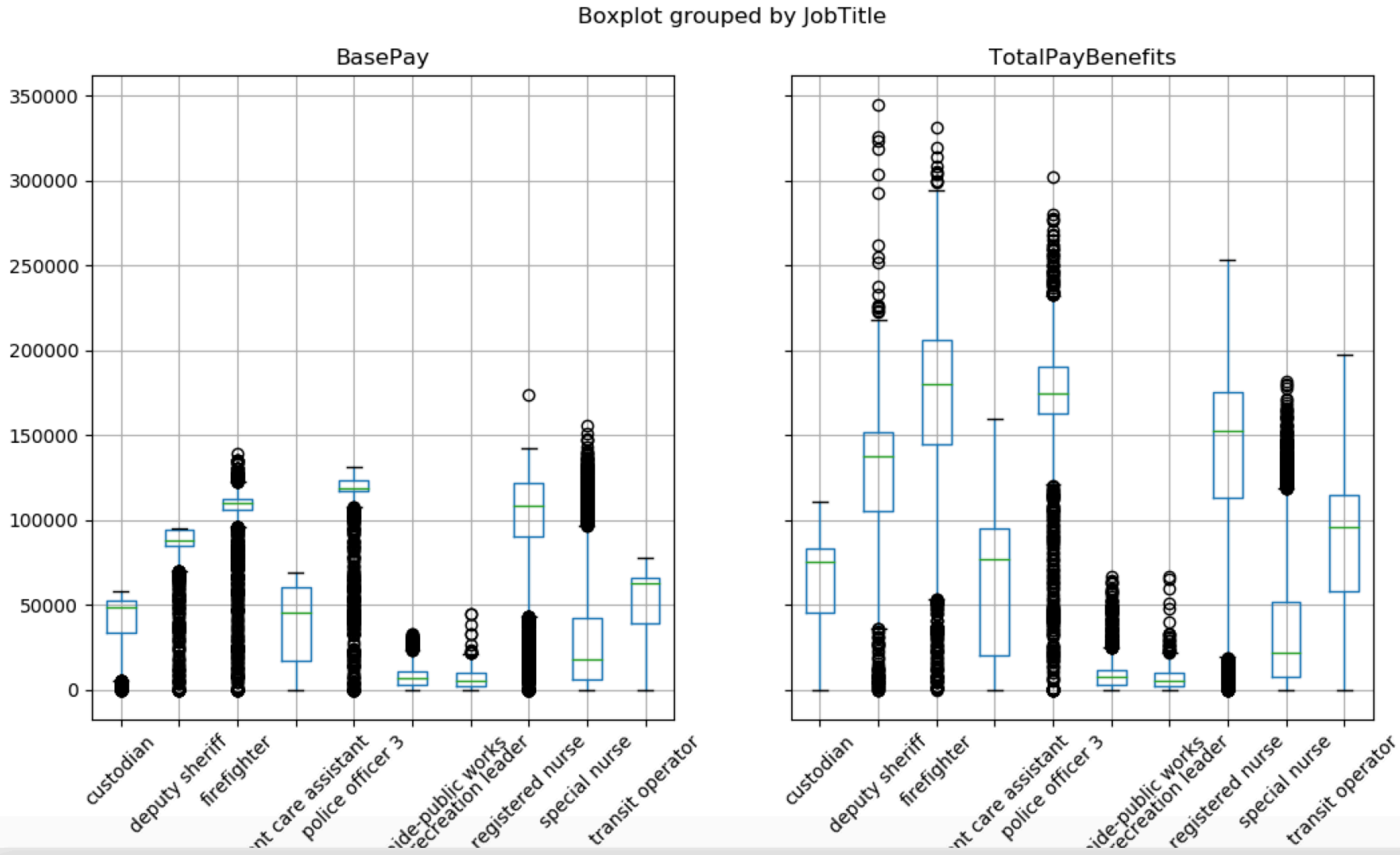
```
In [22]: data_10jobtitle.boxplot(column="BasePay",by="JobTitle",rot=45,figsize=(12,12))
Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x12855c6a0>
```



Plotting boxplots with pandas

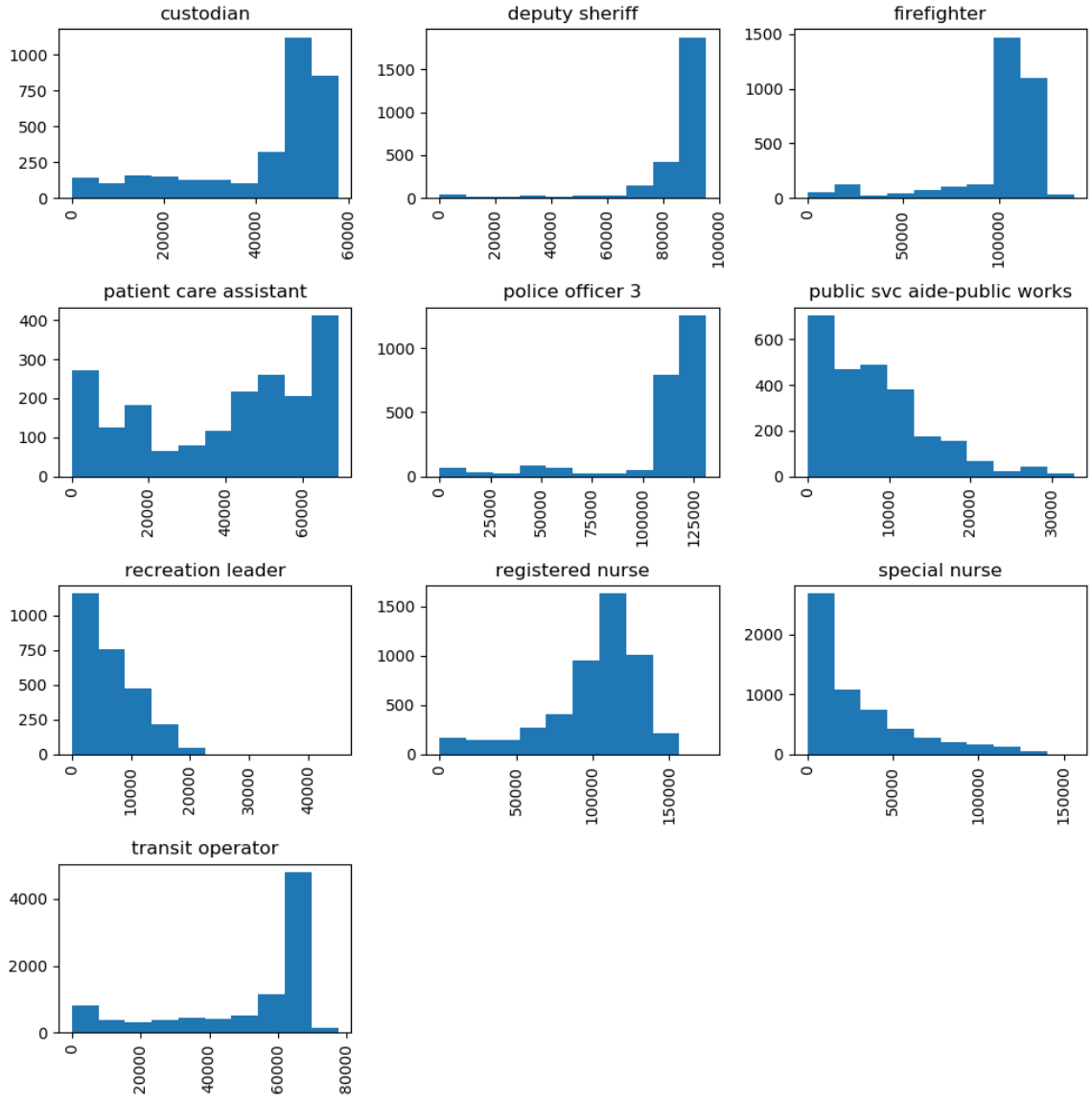
- Selecting more than two columns for boxplot

```
In [24]: data_10jobtitle.boxplot(column=["BasePay","TotalPayBenefits"],by="JobTitle",rot=45,figsize=(12,12))
```



Plotting histogram plots with pandas

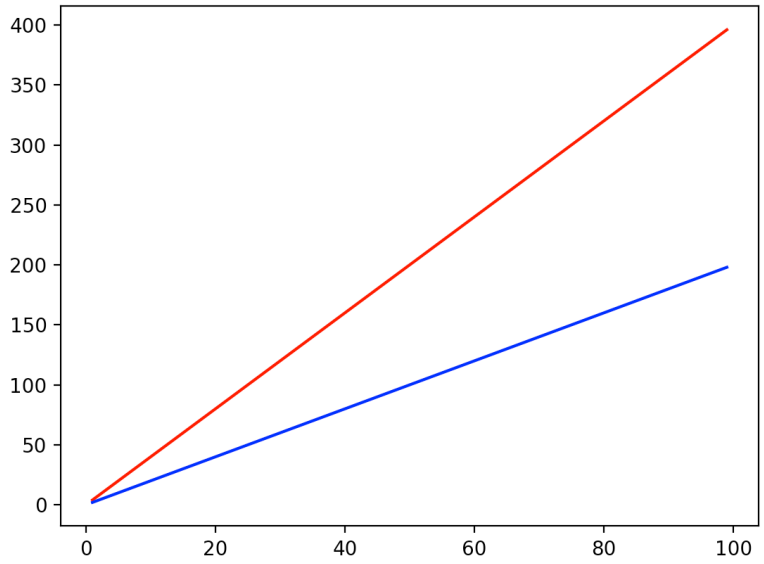
```
data_10jobtitle.hist(column="BasePay",by="JobTitle",figsize=(10,10))
plt.tight_layout()
plt.savefig("test.png")
```



Customizing color and styles

```
plot(x, y, color='green', linestyle='dashed', marker='o',
     markerfacecolor='blue', markersize=12).
```

- **Predefined names**:
- matplotlib will interpret standard HTML color names as an actual color.
- For instance, the string red will be accepted as a color and will be interpreted as a bright red. A few colors have a one-letter alias are shown below:
- b: blue g: green r: red c: cyan m: magenta y: yellow k: black w: white




```
#%%
import numpy as np
X=np.arange(1,100)
Y=X*4
Z=X*2
plt.plot(X, Y,color="r")
plt.plot(X,Z,color="b")
plt.show()
```

Customizing color and styles

- **Predefined names**:
- https://matplotlib.org/examples/color/named_colors.html

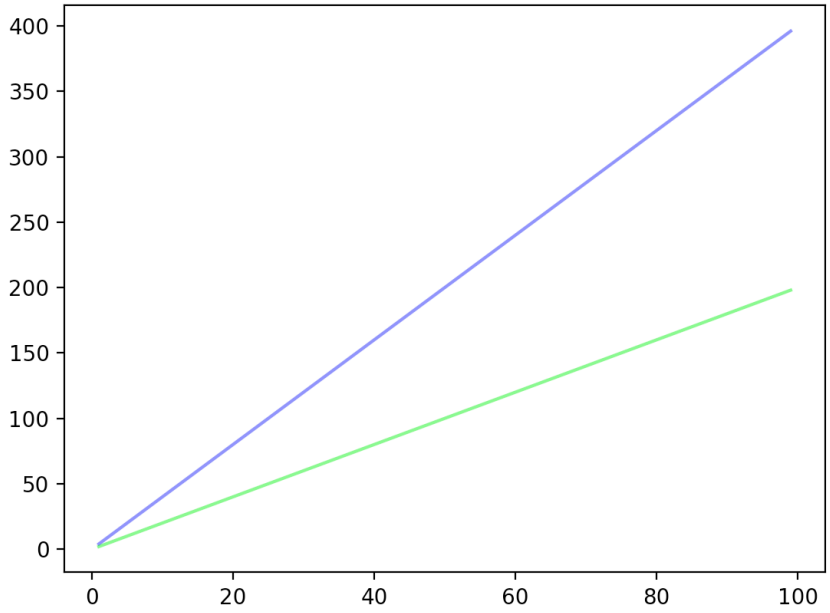
Customizing color and styles

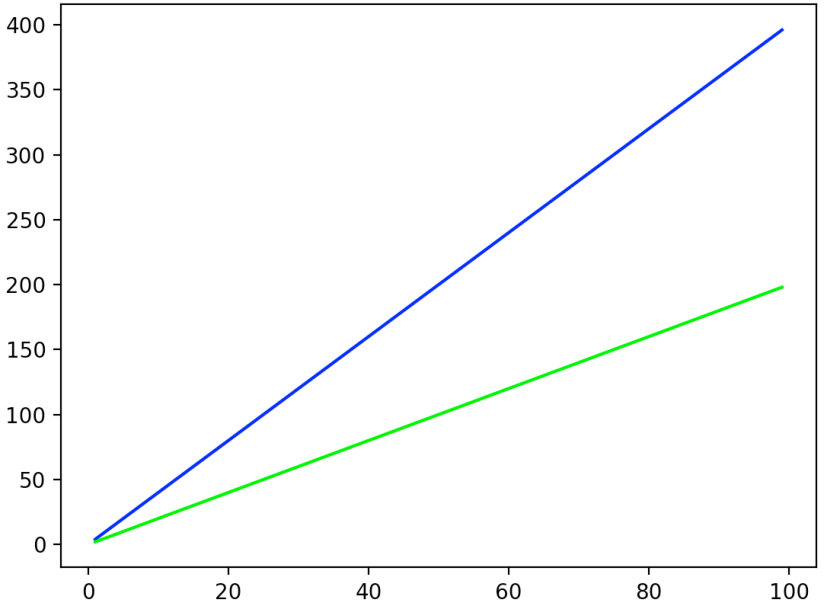
- **Triplets**:
- These colors can be described as a real value triplet—the red, blue, and green components of a color. The components have to be in the [0, 1] interval.
- Thus, the Python syntax (1.0, 0.0, 0.0) will code a pure, bright red, while (1.0, 0.0, 1.0) appears as a strong pink.
- **Quadruplets**: These work as triplets, and the fourth component defines a transparency value.
- This value should also be in the [0, 1] interval. When rendering a figure to a picture file, using transparent colors allows for making figures that blend with a background.

Customizing color and styles

```
import numpy as np
X=np.arange(1,100)
Y=X*4
Z=X*2
plt.plot(X, Y,color=(0, 0, 1,0.5))
plt.plot(X,Z,color=(0 ,1 ,0,0.5))
plt.show()
```

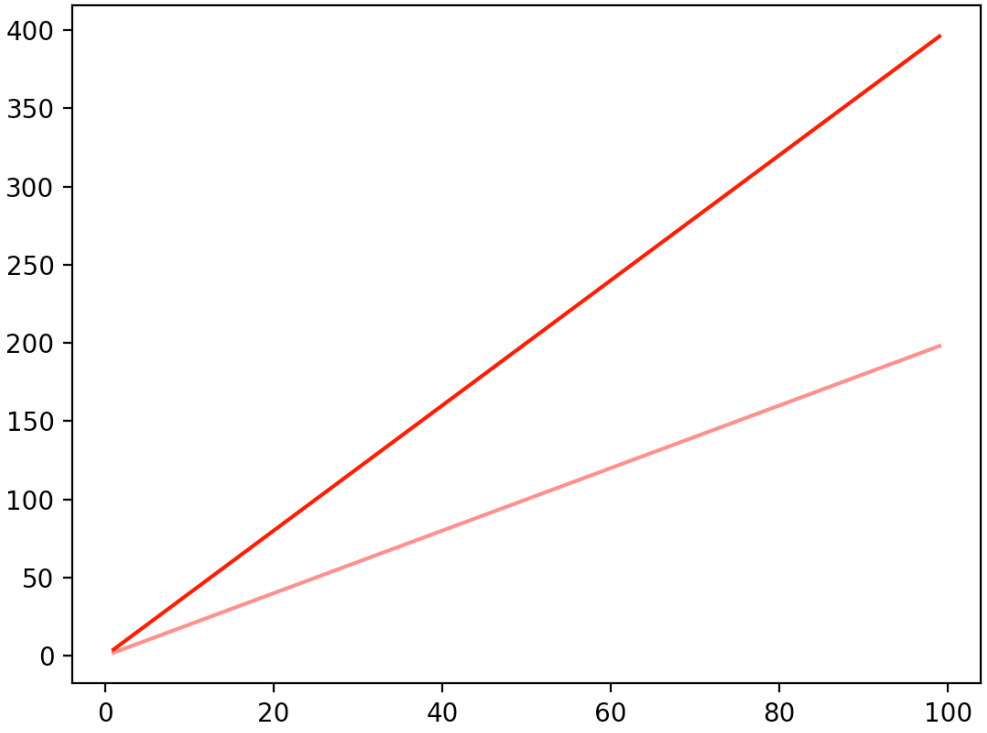
```
import numpy as np
X=np.arange(1,100)
Y=X*4
Z=X*2
plt.plot(X, Y,color=(0, 0, 1))
plt.plot(X,Z,color=(0 ,1 ,0))
plt.show()
```





Customizing color and styles

- Alpha option could be also used to set the transparency of the plot.
- float (0.0 transparent through 1.0 opaque)

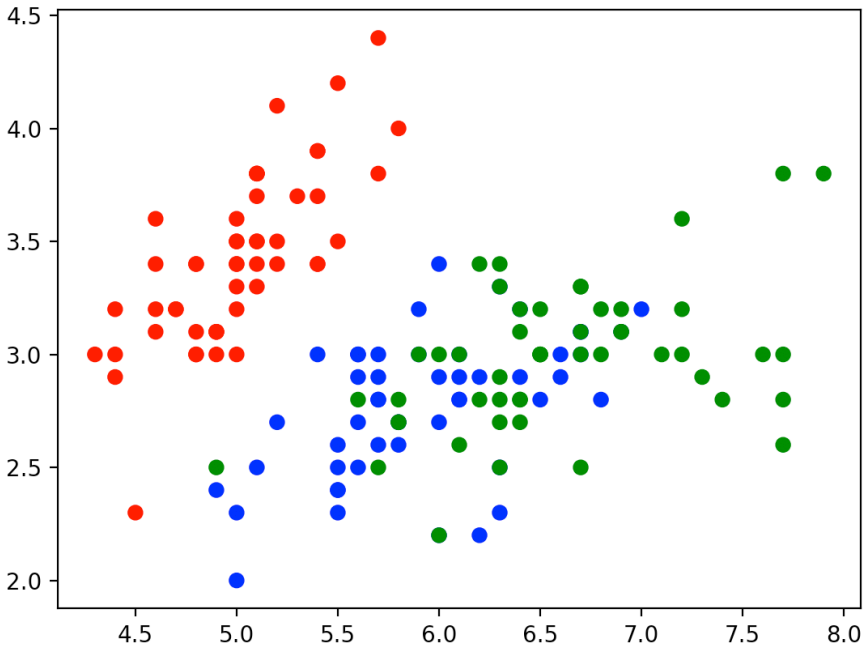



```
import numpy as np
X=np.arange(1,100)
Y=X*4
Z=X*2
plt.plot(X, Y,color="r",alpha=1)
plt.plot(X,Z,color="r",alpha=0.5)
plt.show()
```

Using custom colors for scatter plots

- We can provide which color should be used for each data point.

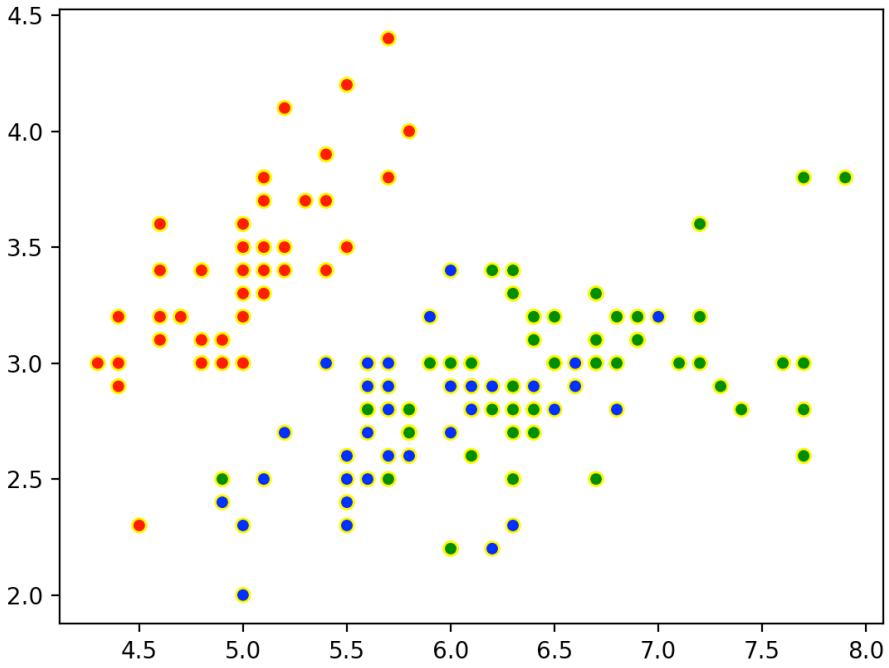
```
import pandas as pd
import matplotlib.pyplot as plt
url="https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
names=["sepal-length","sepal-width","petal-length","petal-width","class"]
dataset=pd.read_csv(url,names=names)

datadict={"Iris-setosa":"r","Iris-versicolor":"b","Iris-virginica":"g"}
colorlist=[datadict[label] for label in dataset["class"]]
plt.scatter(dataset["sepal-length"],dataset["sepal-width"],color=colorlist)
```



Using custom colors for scatter plots

- We can change color of the edges of each point using edgecolor option.

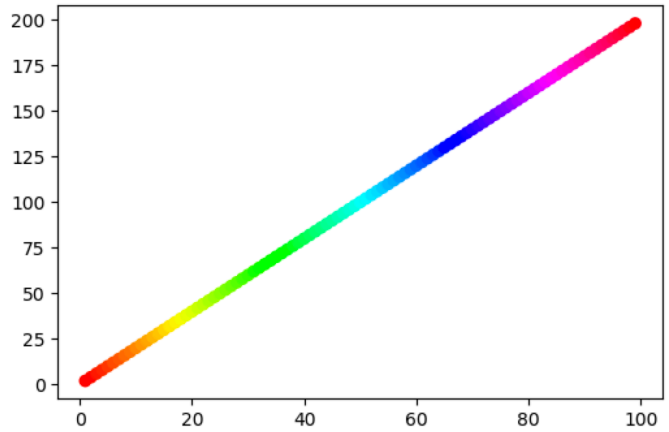
```
import pandas as pd
import matplotlib.pyplot as plt
url="https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
names=["sepal-length","sepal-width","petal-length","petal-width","class"]
dataset=pd.read_csv(url,names=names)

datadict={"Iris-setosa":"r","Iris-versicolor":"b","Iris-virginica":"g"}
colorlist=[datadict[label] for label in dataset["class"]]
plt.scatter(dataset["sepal-length"],dataset["sepal-width"],color=colorlist,edgecolor="yellow")
plt.show()
```



Using colormaps for scatter plots

- When using a lot of colors, de ning each color one by one is difficult.
- Building a good set of colors is a problem in itself.
- Colormaps define colors with a continuous function of one variable to one value, corresponding to one color.
- With cmap parameter we can define which colormap wanted to be used.

```
Out[7]: <matplotlib.collections.PathCollection at 0x110907b38>
```



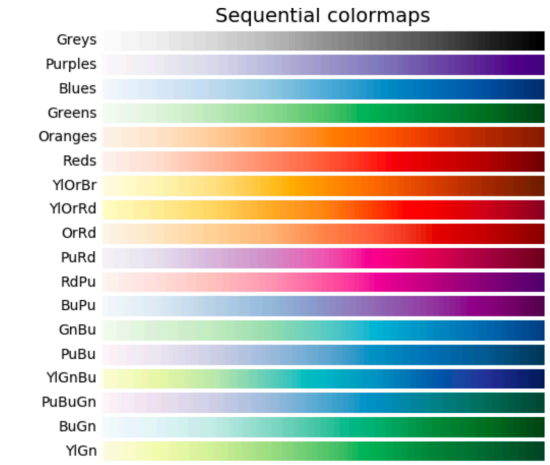

```
In [8]:
```

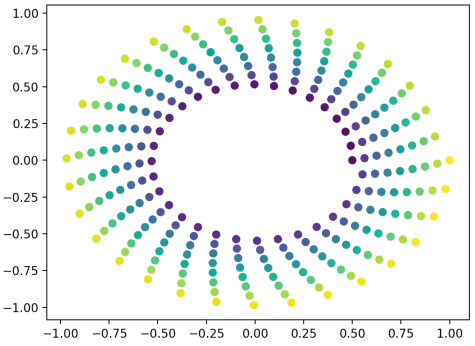
```
#%%
import numpy as np
import matplotlib.cm as cm
x=np.arange(1,100)
y=2*np.arange(1,100)
z=np.arange(1,100)
plt.scatter(x,y,c=z,cmap=cm.hsv)
```

Using colormaps for scatter plots

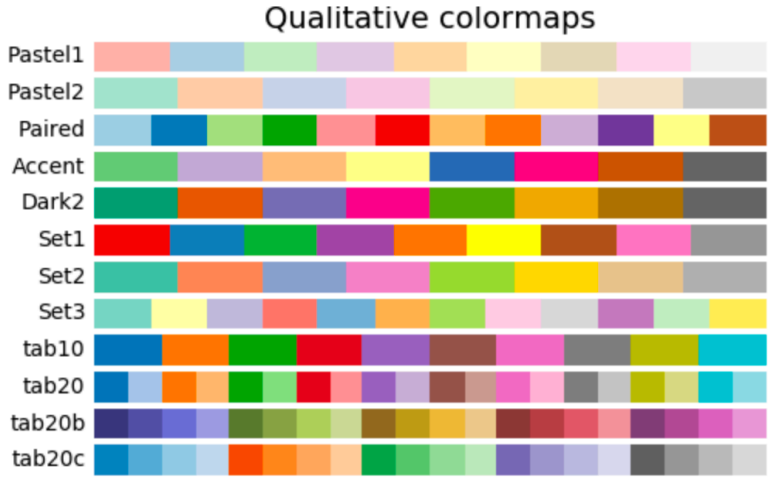
- For more colormaps see:
- https://matplotlib.org/examples/color/colormaps_reference.html

(png, pdf)

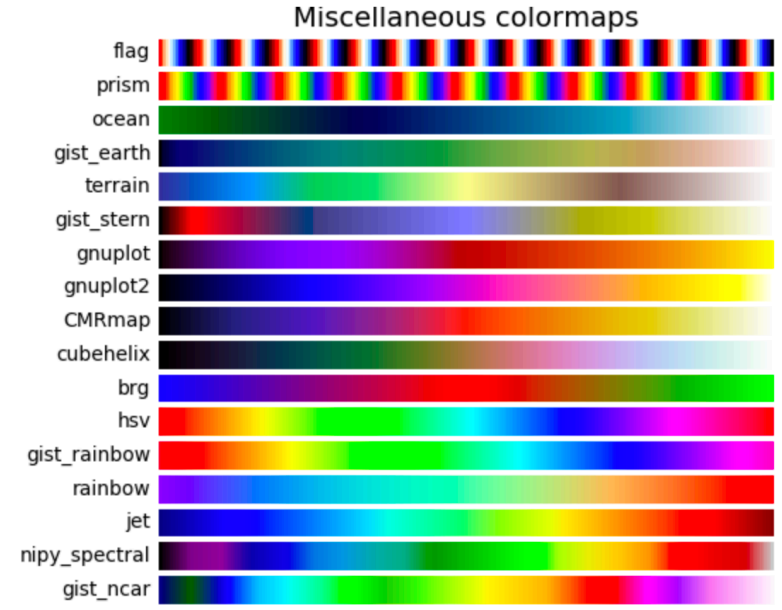




(png, pdf)



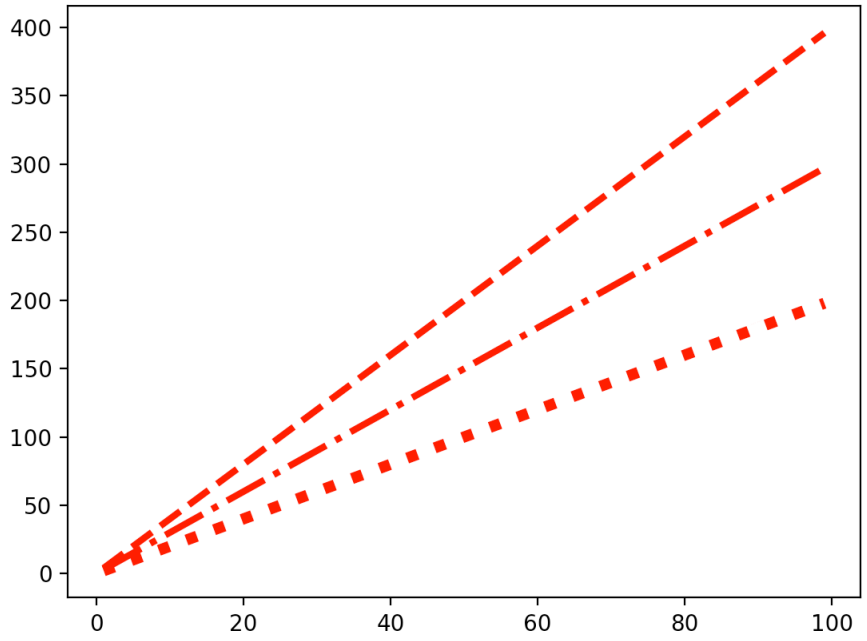
(png, pdf)



Controlling a line pattern and thickness

- Linestyle parameter for linetype, Linewidth parameter for the thickness.

linestyle	description
'-' or 'solid'	solid line
'--' or 'dashed'	dashed line
'-.' or 'dashdot'	dash-dotted line
':' or 'dotted'	dotted line
'None'	draw nothing
' '	draw nothing
''	draw nothing

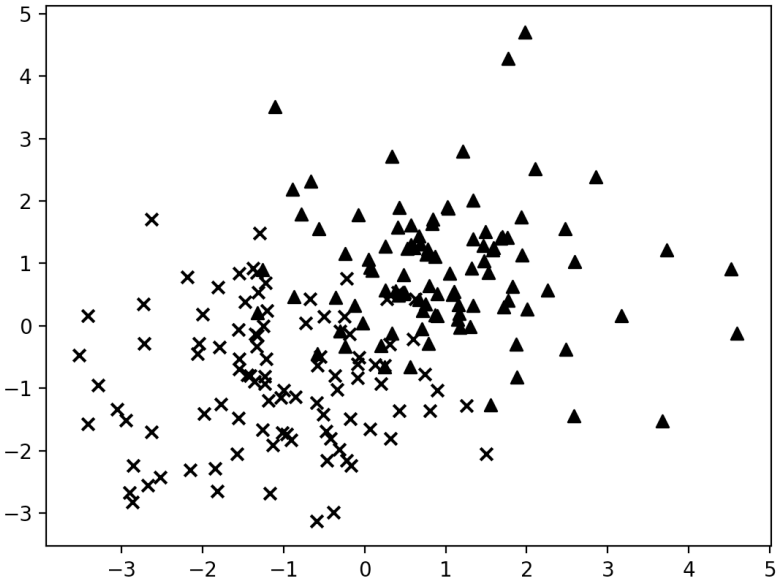



```
import numpy as np
X=np.arange(1,100)
Y=X*4
Z=X*2
plt.plot(X, Y,color="r",linestyle = 'dashed',linewidth=3.0)
plt.plot(X, X*3,color="r",linestyle = 'dashdot',linewidth=3.0)
plt.plot(X,Z,color="r",linestyle = 'dotted',linewidth=5.0)
plt.show()
```

Controlling Marker's style

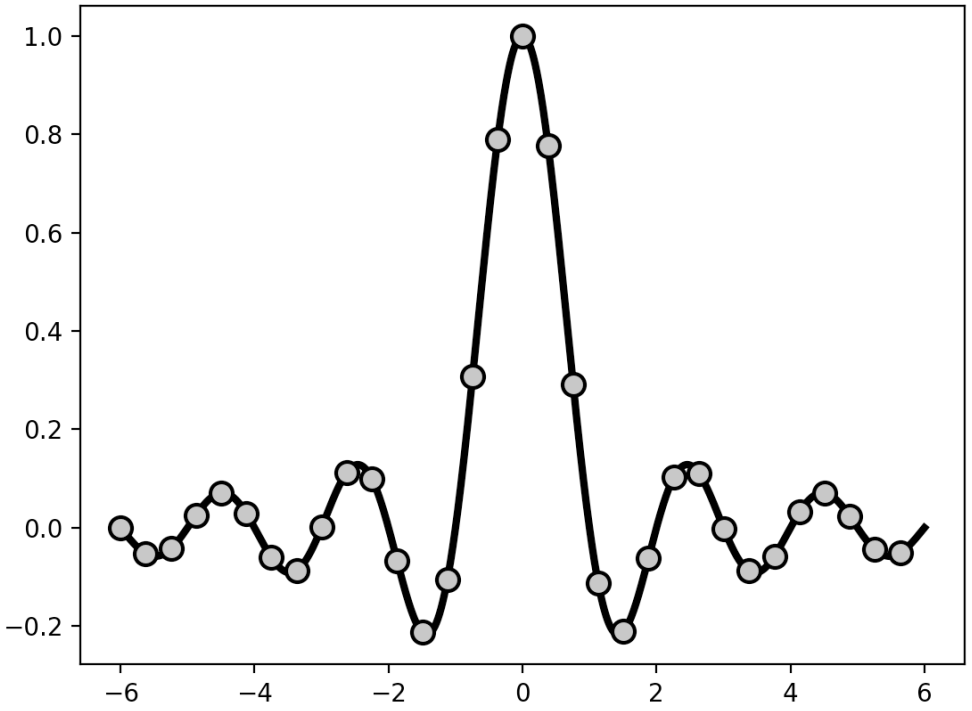
- https://matplotlib.org/api/markers_api.html

```
import numpy as np
import matplotlib.pyplot as plt
A = np.random.standard_normal((100, 2))
A += np.array((-1, -1))
B = np.random.standard_normal((100, 2))
B += np.array((1, 1))
plt.scatter(A[:,0], A[:,1], color = 'k', marker = 'x')
plt.scatter(B[:,0], B[:,1], color = 'k', marker = '^')
plt.show()
```



marker	description
"."	point
","	pixel
"o"	circle
"v"	triangle_down
"^"	triangle_up
"<"	triangle_left
">"	triangle_right
"1"	tri_down
"2"	tri_up
"3"	tri_left
"4"	tri_right
"8"	octagon
"s"	square
"p"	pentagon
"P"	plus (filled)
"*"	star
"h"	hexagon1

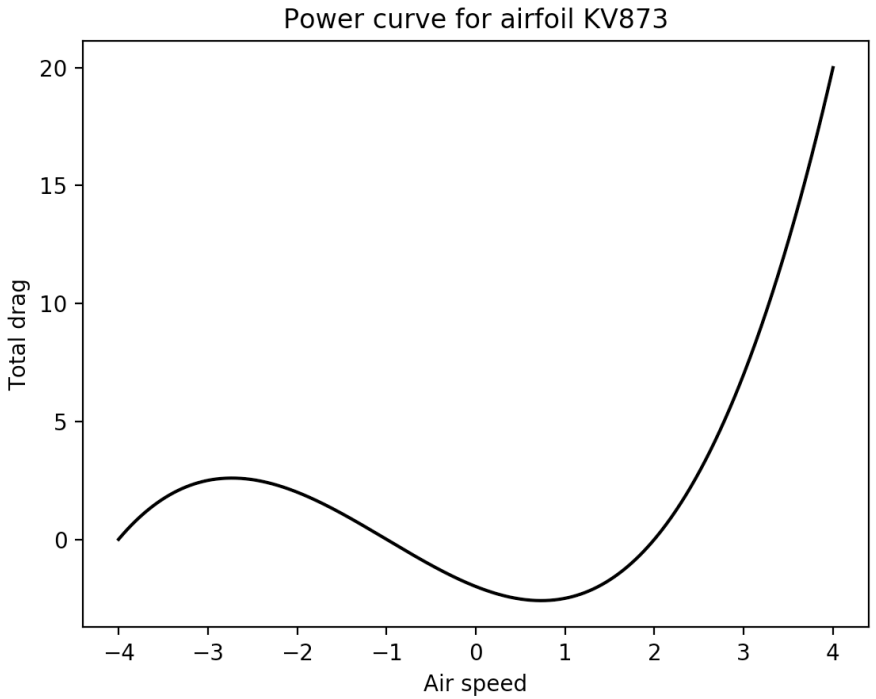
Controlling Marker's Properties




```
import numpy as np
import matplotlib.pyplot as plt
X = np.linspace(-6, 6, 1024)
Y = np.sinc(X)
plt.plot(X, Y,
  linewidth = 3.,
  color = 'k',
  markersize = 9,
  markeredgewidth = 1.5,
  markerfacecolor = '.75',
  markeredgecolor = 'k',
  marker = 'o',
  markevery = 32)
plt.show()
```

Adding labels to axis

- pyplot.xlabel() and pyplot.ylabel()functions to add a description of the horizontal axis and the vertical axis,

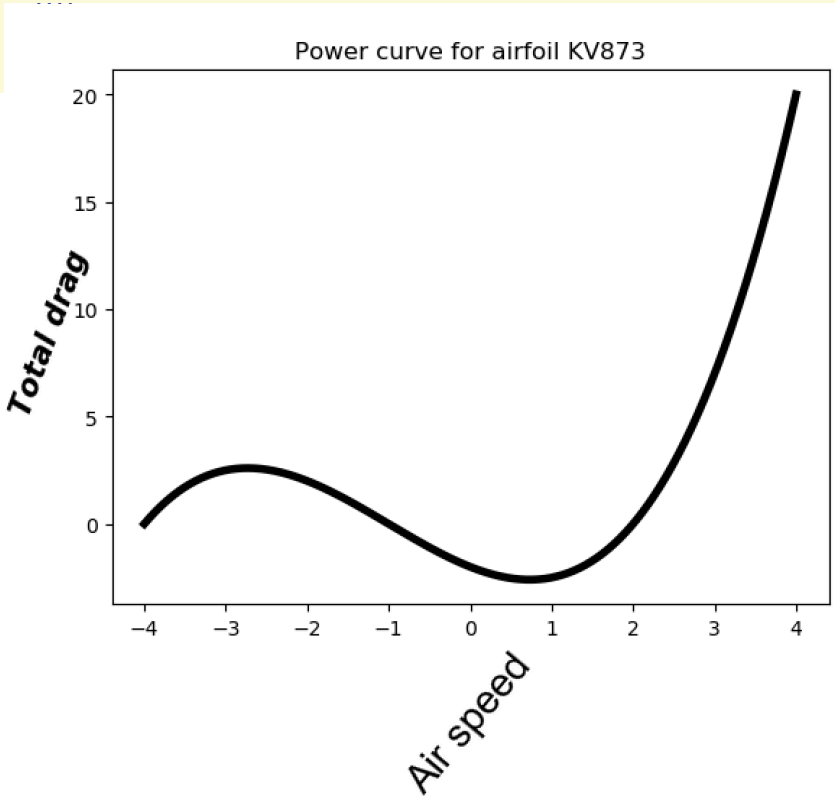



```
import numpy as np
import matplotlib.pyplot as plt
X = np.linspace(-4, 4, 1024)
Y = .25 * (X + 4.) * (X + 1.) * (X - 2.)
plt.title('Power curve for airfoil KV873')
plt.xlabel('Air speed')
plt.ylabel('Total drag')
plt.plot(X, Y, c = 'k')
plt.show()
```

Adding labels to axis

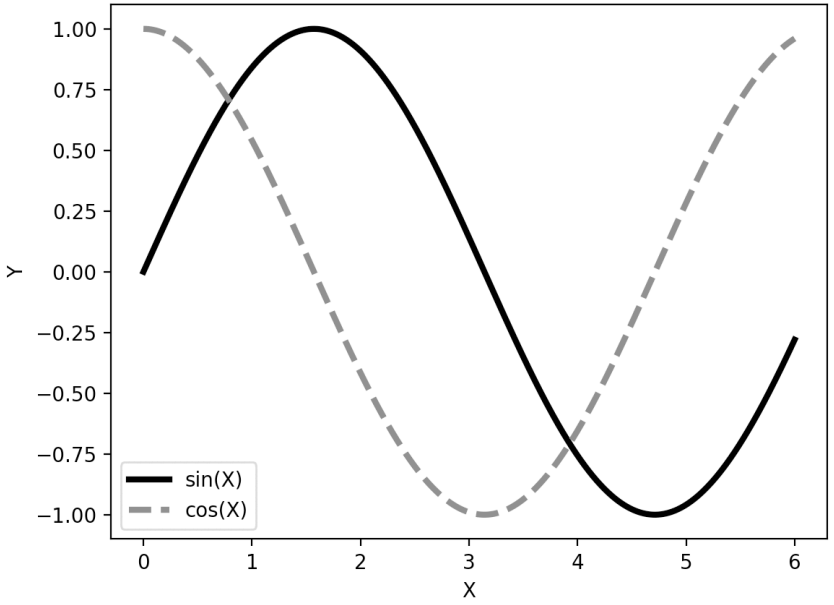
- We change styles in the labels.

```
import numpy as np
import matplotlib.pyplot as plt
X = np.linspace(-4, 4, 1024)
Y = .25 * (X + 4.) * (X + 1.) * (X - 2.)
plt.title('Power curve for airfoil KV873')
plt.xlabel('Air speed',fontsize=20,fontname="arial",rotation=50)
plt.ylabel('Total drag',fontsize=15,style="italic",fontweight="bold",rotation=70)
plt.plot(X, Y, c = 'k')
plt.show()
```



Adding a legend

- pyplot.legend() function adds a legend.
- Pyplot function has an optional label parameter to name an element, such as curve, histogram, and so on, of a figure. matplotlib keeps a track of these label.
- These labels are rendered when pylot.legend() function is called.




```
import numpy as np
import matplotlib.pyplot as plt
X = np.linspace(0, 6, 1024)
Y1 = np.sin(X)
Y2 = np.cos(X)
plt.xlabel('X')
plt.ylabel('Y')
plt.plot(X, Y1, c = 'k',  lw = 3.,label="sin(X)")
plt.plot(X, Y2, c = '.5', lw = 3., ls = '--', label = 'cos(X)')
plt.legend()
plt.show()
```

loc : int or string or pair of floats, default: 'upper right'

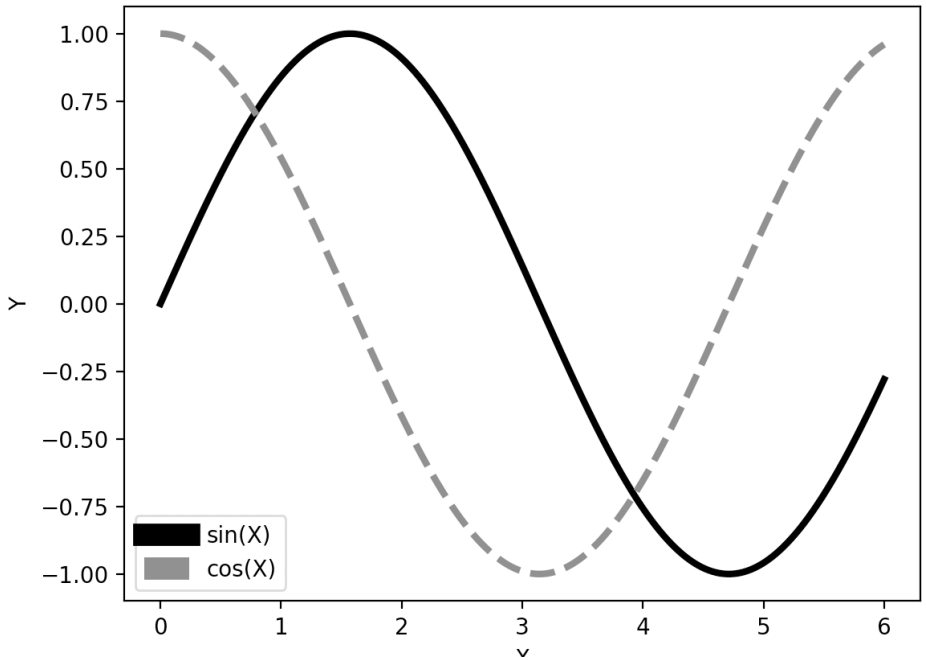
The location of the legend. Possible codes are:

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

Adding a legend

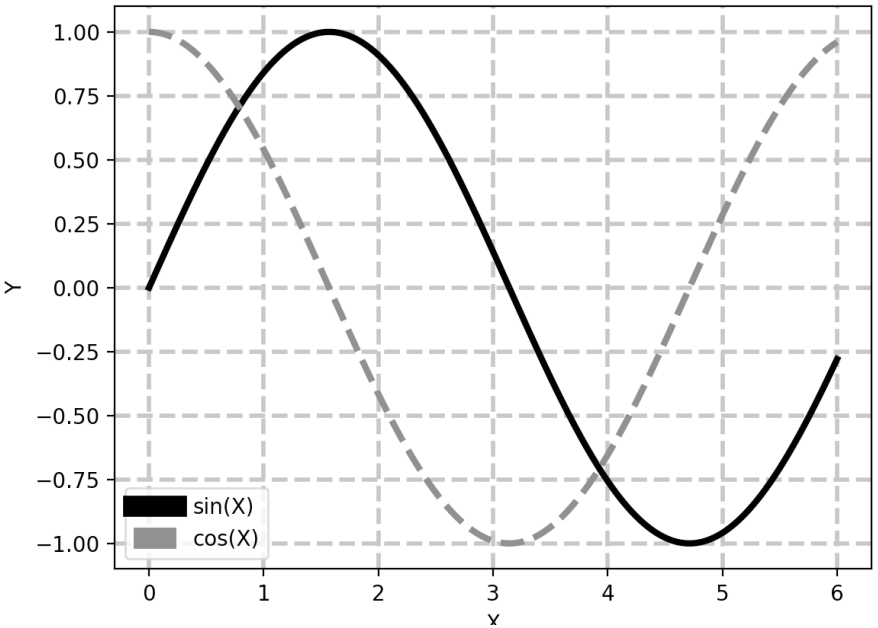
- Changing the line width inside the legend.

```
#%%
import numpy as np
import matplotlib.pyplot as plt
X = np.linspace(0, 6, 1024)
Y1 = np.sin(X)
Y2 = np.cos(X)
plt.xlabel('X')
plt.ylabel('Y')
plt.plot(X, Y1, c = 'k',  lw = 3.,label="sin(X)")
plt.plot(X, Y2, c = '.5', lw = 3., ls = '--', label = 'cos(X)')
leg=plt.legend()
for label in leg.get_lines():
    label.set_linewidth(10)  # the legend line width

plt.show()
```



Adding a grid

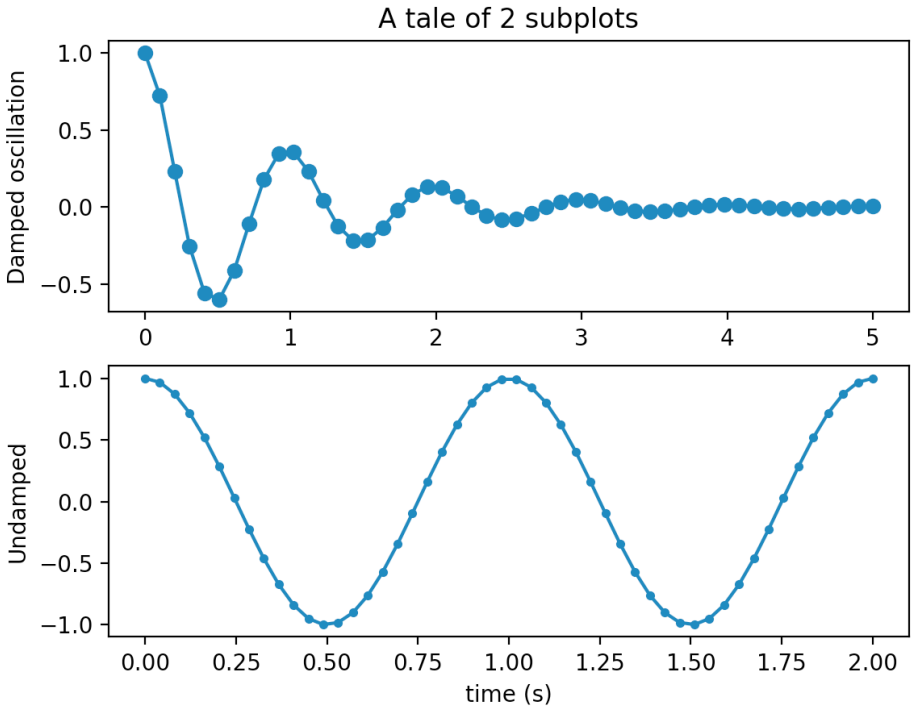
- Adding a grid to the figure is a natural way to improve the readability of a figure.




```
import numpy as np
import matplotlib.pyplot as plt
X = np.linspace(0, 6, 1024)
Y1 = np.sin(X)
Y2 = np.cos(X)
plt.xlabel('X')
plt.ylabel('Y')
plt.plot(X, Y1, c = 'k',  lw = 3.,label="sin(X)")
plt.plot(X, Y2, c = '.5', lw = 3., ls = '--', label = 'cos(X)')
leg=plt.legend()
for label in leg.get_lines():
    label.set_linewidth(10)  # the legend line width

plt.grid(True, lw = 2, ls = '--', c = '.75')
plt.show()
```

Multiple plots

- Subplots() function creates a figure and subplots.

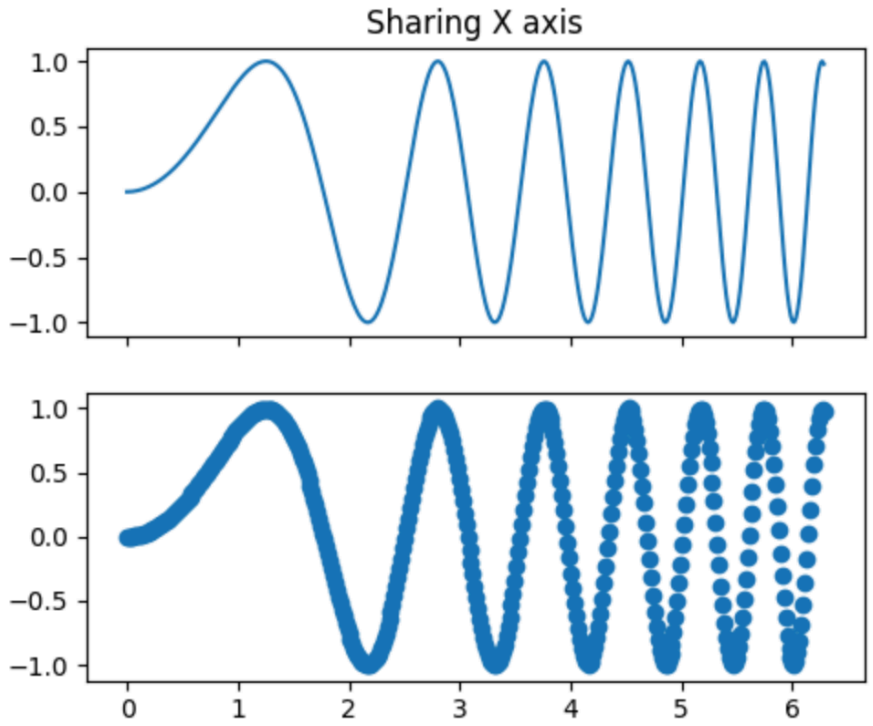



```
import numpy as np
import matplotlib.pyplot as plt

x1 = np.linspace(0.0, 5.0)
x2 = np.linspace(0.0, 2.0)

y1 = np.cos(2 * np.pi * x1) * np.exp(-x1)
y2 = np.cos(2 * np.pi * x2)

plt.subplot(2, 1, 1)
plt.plot(x1, y1, 'o-')
plt.title('A tale of 2 subplots')
plt.ylabel('Damped oscillation')

plt.subplot(2, 1, 2)
plt.plot(x2, y2, '.-')
plt.xlabel('time (s)')
plt.ylabel('Undamped')
plt.show()
```

- subplot(nrows, ncols, plot_number)
- If a subplot is applied to a figure, the figure will be notionally split into 'nrows' * 'ncols' sub-axes.
- The parameter 'plot_number' identifies the subplot that the function call has to create.

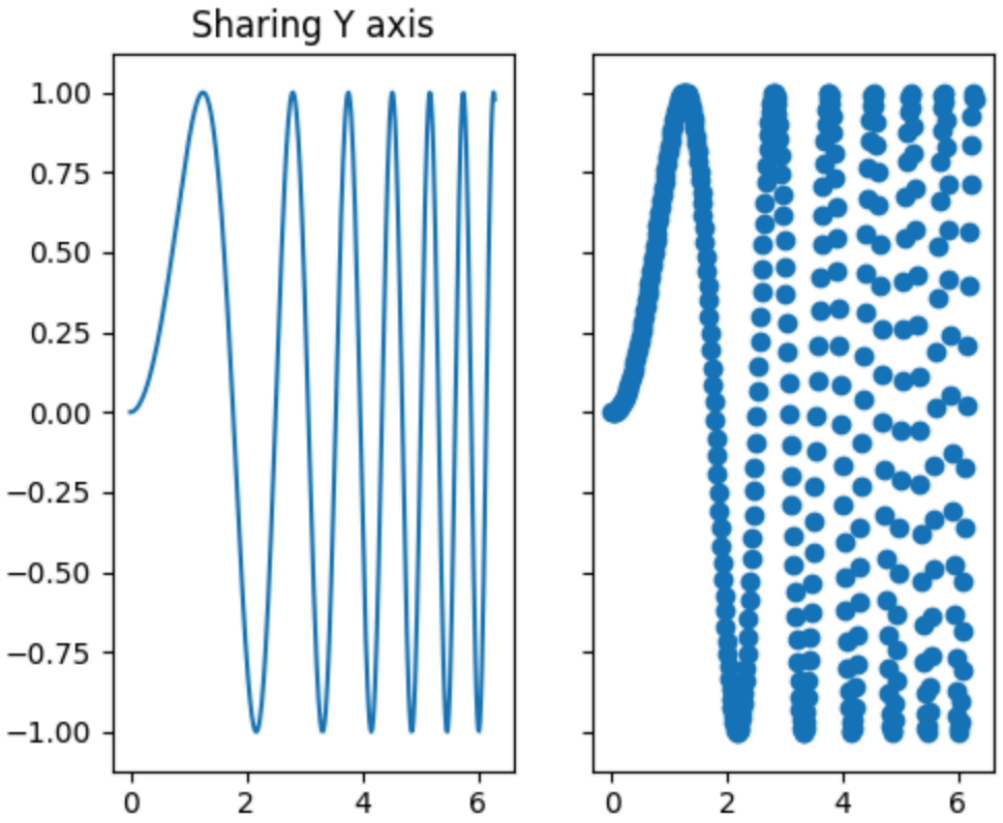
Multiple plots

```
# Two subplots, the axes array is 1-d
f, axarr = plt.subplots(2, sharex=True)
axarr[0].plot(x, y)
axarr[0].set_title('Sharing X axis')
axarr[1].scatter(x, y)
```



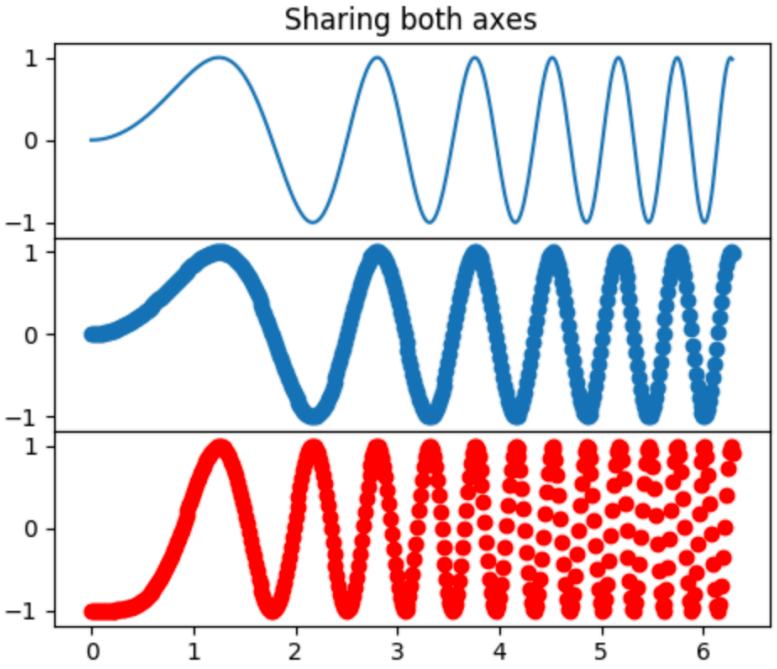
Multiple plots

```
# Two subplots, unpack the axes array immediately
f, (ax1, ax2) = plt.subplots(1, 2, sharey=True)
ax1.plot(x, y)
ax1.set_title('Sharing Y axis')
ax2.scatter(x, y)
```



Multiple plots

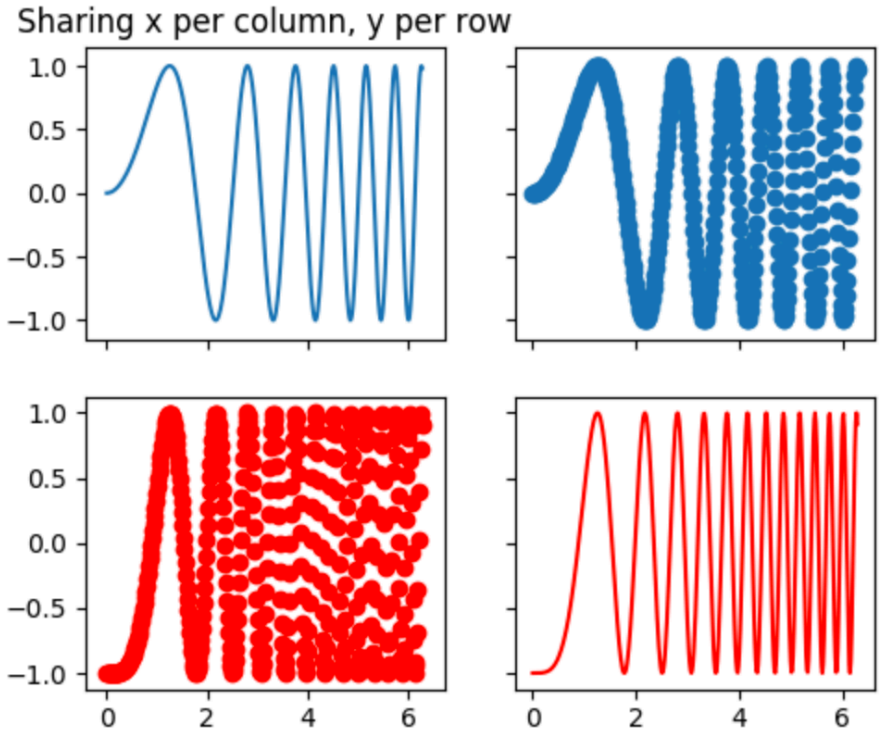
```
# Three subplots sharing both x/y axes
f, (ax1, ax2, ax3) = plt.subplots(3, sharex=True, sharey=True)
ax1.plot(x, y)
ax1.set_title('Sharing both axes')
ax2.scatter(x, y)
ax3.scatter(x, 2 * y ** 2 - 1, color='r')
# Fine-tune figure; make subplots close to each other and hide x ticks for
# all but bottom plot.
f.subplots_adjust(hspace=0)
plt.setp([a.get_xticklabels() for a in f.axes[:-1]], visible=False)
```



- hspace = 0.2 the amount of height reserved for white space between subplots, expressed as a fraction of the average axis height
- Plt.setp() sets a property
- If sharex=False, then each subplot x- or y-axis will be independent.

Multiple plots

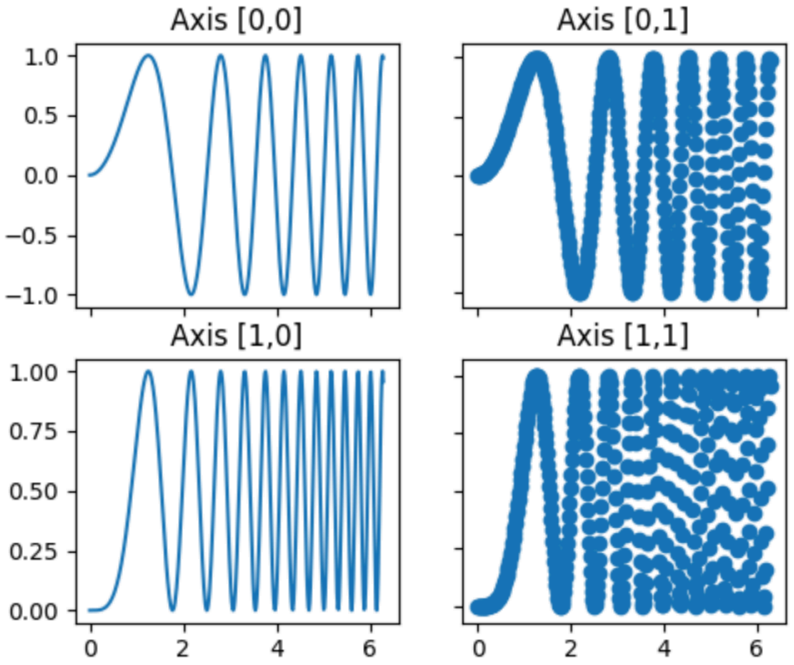
```
# row and column sharing
f, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, sharex='col', sharey='row')
ax1.plot(x, y)
ax1.set_title('Sharing x per column, y per row')
ax2.scatter(x, y)
ax3.scatter(x, 2 * y ** 2 - 1, color='r')
ax4.plot(x, 2 * y ** 2 - 1, color='r')
```



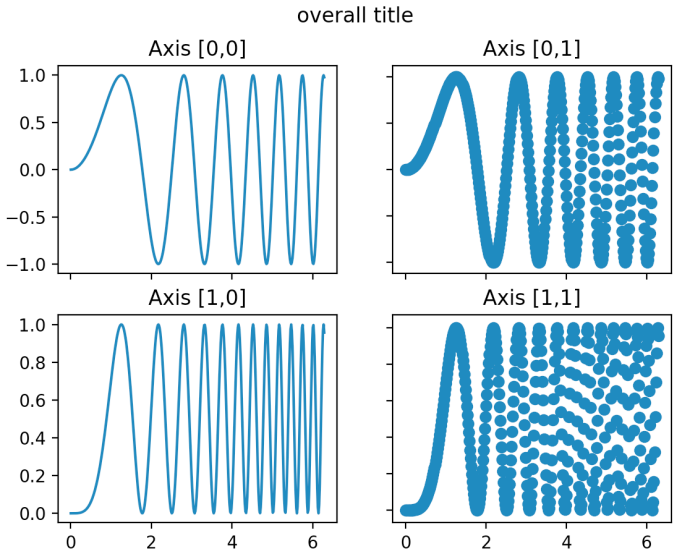
- • ‘row’: each subplot row will share an x- or y-axis.
- • ‘col’: each subplot column will share an x- or y-axis.

Multiple plots

```
# Four axes, returned as a 2-d array
f, axarr = plt.subplots(2, 2)
axarr[0, 0].plot(x, y)
axarr[0, 0].set_title('Axis [0,0]')
axarr[0, 1].scatter(x, y)
axarr[0, 1].set_title('Axis [0,1]')
axarr[1, 0].plot(x, y ** 2)
axarr[1, 0].set_title('Axis [1,0]')
axarr[1, 1].scatter(x, y ** 2)
axarr[1, 1].set_title('Axis [1,1]')
# Fine-tune figure; hide x ticks for top plots and y ticks for right plots
plt.setp([a.get_xticklabels() for a in axarr[0, :]], visible=False)
plt.setp([a.get_yticklabels() for a in axarr[:, 1]], visible=False)
```



Multiple plots




```python
import matplotlib.pyplot as plt
import numpy as np

# Simple data to display in various forms
x = np.linspace(0, 2 * np.pi, 400)
y = np.sin(x ** 2)

# Four axes, returned as a 2-d array
f, axarr = plt.subplots(2, 2)
axarr[0, 0].plot(x, y)
axarr[0, 0].set_title('Axis [0,0]')
axarr[0, 1].scatter(x, y)
axarr[0, 1].set_title('Axis [0,1]')
axarr[1, 0].plot(x, y ** 2)
axarr[1, 0].set_title('Axis [1,0]')
axarr[1, 1].scatter(x, y ** 2)
axarr[1, 1].set_title('Axis [1,1]')
# Fine-tune figure; hide x ticks for top plots and y ticks for right plots
plt.setp([a.get_xticklabels() for a in axarr[0, :]], visible=False)
plt.setp([a.get_yticklabels() for a in axarr[:, 1]], visible=False)
f.suptitle("overall title")
```

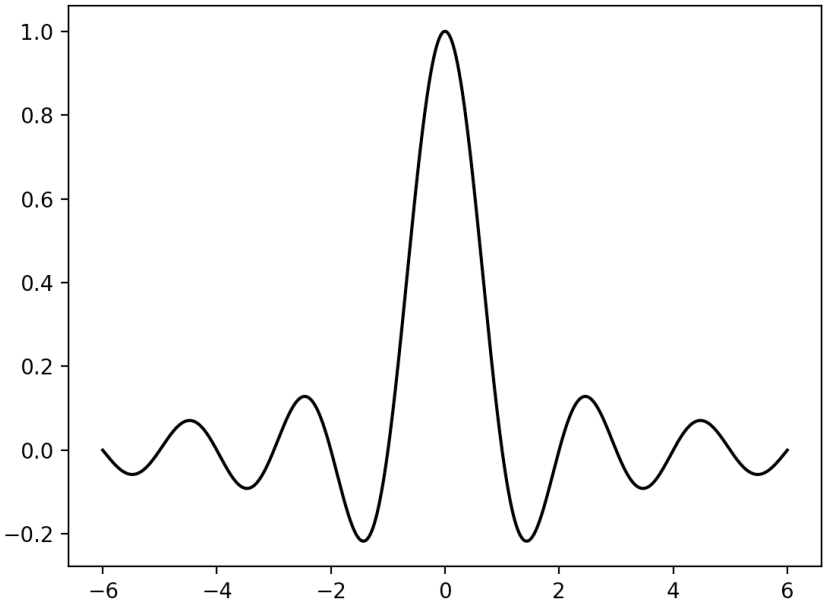
- F.suptitle() allows to put a overall time for the figure

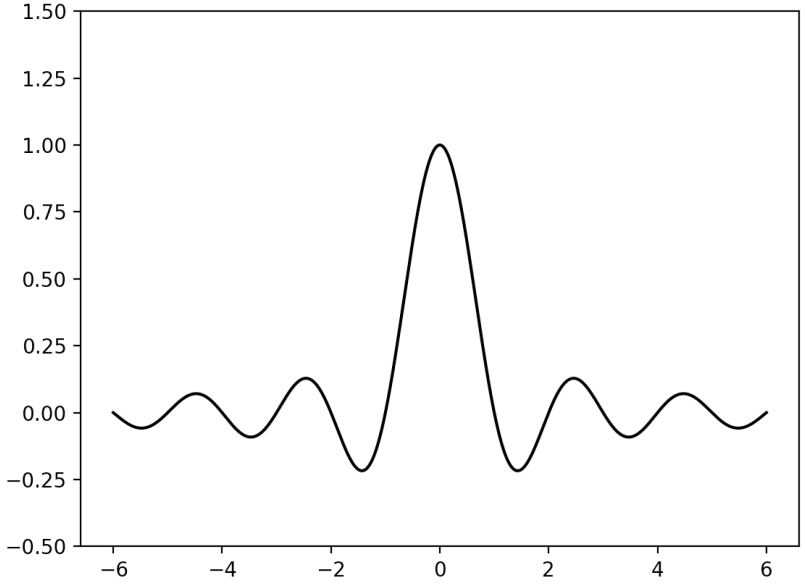
Setting an axis range

- By default, matplotlib will find the minimum and maximum of your data on both axes and use this as the range to plot your data.
- However, sometimes you may prefer explicitly define the data range.

```
#%%
import numpy as np
import matplotlib.pyplot as plt
X = np.linspace(-6, 6, 1024)
plt.plot(X, np.sinc(X), c = 'k')
plt.show()
```

```
#%%
import numpy as np
import matplotlib.pyplot as plt
X = np.linspace(-6, 6, 1024)
plt.ylim(-.5, 1.5)
plt.plot(X, np.sinc(X), c = 'k')
plt.show()
```





Saving figures as files

- pyplot.savefig() function saves a figure into a file.

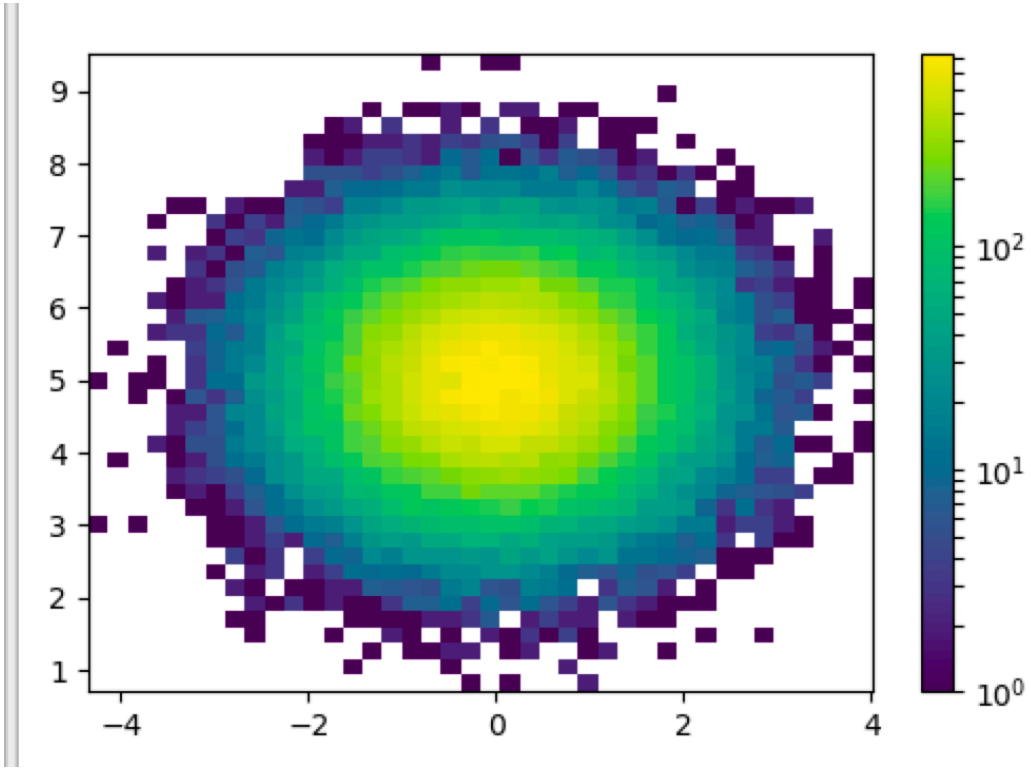
```
savefig(fname, dpi=None, facecolor='w', edgecolor='w',
        orientation='portrait', papertype=None, format=None,
        transparent=False, bbox_inches=None, pad_inches=0.1,
        frameon=None)
```

- plt.savefig('sinc.png', dpi = 300) controlling output resolution
- Dpi parameter controls the resolution of the picture expressed in DPI (Dots Per Inches).
- By default, matplotlib will output a figure of 4/3 aspect ratio.
  Thus, by default, matplotlib will give a picture file of 640x 480 pixels.
- You can change figure size plt.figure(figsize=(10.24, 2.56))

- .

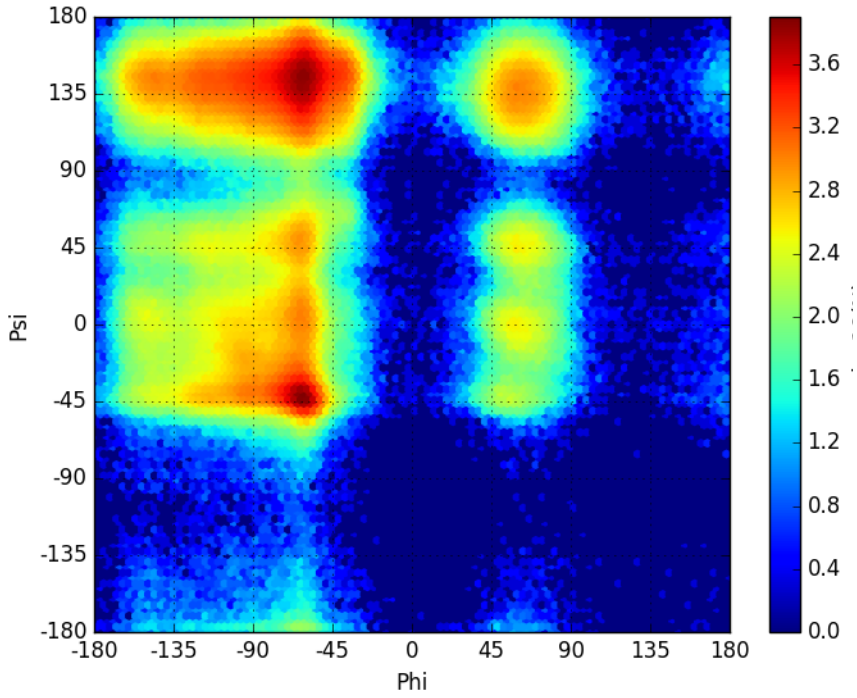
2D histogram plots

```
from matplotlib.colors import LogNorm
import matplotlib.pyplot as plt
import numpy as np

# normal distribution center at x=0 and y=5
x = np.random.randn(100000)
y = np.random.randn(100000) + 5

plt.hist2d(x, y, bins=40, norm=LogNorm())
plt.colorbar()
plt.show()
```



Hexbin Plots

- 2-D histogram with hexagonal cells




```
fig, ax1=plt.subplots()
ax1.set_axis_bgcolor('white')
plt.hexbin(phidata, psidata, bins='log', cmap=plt.get_cmap('jet'
plt.axis([-180, 180, -180, 180])
#plt.hist2d(phidata, psidata, bins=20, norm=LogNorm())
cb = plt.colorbar()
cb.set_label('log10(N)')
plt.grid(True)

ax1Xs = [ -180, -135, -90, -45, 0, 45,90, 135,180]
ax1.set_xticks(ax1Xs)
ax1.set_xticklabels(ax1Xs,fontsize=12, color='black')

ax1Ys = [ -180, -135, -90, -45, 0, 45,90, 135,180]
ax1.set_yticks(ax1Ys)
ax1.set_yticklabels(ax1Ys,fontsize=12, color='black')

plt.xlabel('Phi')
plt.ylabel('Psi')
```