Data Visualization

- Data visualization makes it easier to sea patterns in the data than just looking at tables of numbers
- The Anscombe data set contains four sets of data, each contains two continuous variables
- Each set has the same mean, variance, correlation, and regression line
- Only when the data are visualized does it become obvious that each set does not follow the same pattern

The Anscombe Data Set

```
import seaborn as sns
anscombe = sns.load_dataset("anscombe")
print(anscombe)
```

```
   dataset     x      y
0        I  10.0   8.04
1        I   8.0   6.95
2        I  13.0   7.58
3        I   9.0   8.81
4        I  11.0   8.33
5        I  14.0   9.96
6        I   6.0   7.24
7        I   4.0   4.26
8        I  12.0  10.84
9        I   7.0   4.82
10       I   5.0   5.68
11      II  10.0   9.14
12      II   8.0   8.14
13      II  13.0   8.74
14      II   9.0   8.77
15      II  11.0   9.26
16      II  14.0   8.10
17      II   6.0   6.13
18      II   4.0   3.10
19      II  12.0   9.13
20      II   7.0   7.26
21      II   5.0   4.74
22     III  10.0   7.46
23     III   8.0   6.77
24     III  13.0  12.74
25     III   9.0   7.11
26     III  11.0   7.81
27     III  14.0   8.84
28     III   6.0   6.08
29     III   4.0   5.39
30     III  12.0   8.15
31     III   7.0   6.42
32     III   5.0   5.73
33      IV   8.0   6.58
34      IV   8.0   5.76
35      IV   8.0   7.71
36      IV   8.0   8.84
37      IV   8.0   8.47
38      IV   8.0   7.04
39      IV   8.0   5.25
40      IV  19.0  12.50
41      IV   8.0   5.56
42      IV   8.0   7.91
43      IV   8.0   6.89
```

In [ ]:

The Anscombe Data Set

Jupyter QtConsole

File Edit View Kernel Window Help

```
In [29]: # create subsets of the anscombe data
    ...: dataset_1 = anscombe[anscombe['dataset'] == 'I']
    ...: dataset_2 = anscombe[anscombe['dataset'] == 'II']
    ...: dataset_3 = anscombe[anscombe['dataset'] == 'III']
    ...: dataset_4 = anscombe[anscombe['dataset'] == 'IV']

In [30]: for d in [dataset_1, dataset_2, dataset_3, dataset_4]:
    ...:     print("mean:\n", d.mean())
    ...:     print("variance:\n", d.var())
    ...:     print("correlation: ", d['x'].corr(d['y']))
    ...:     print()
```

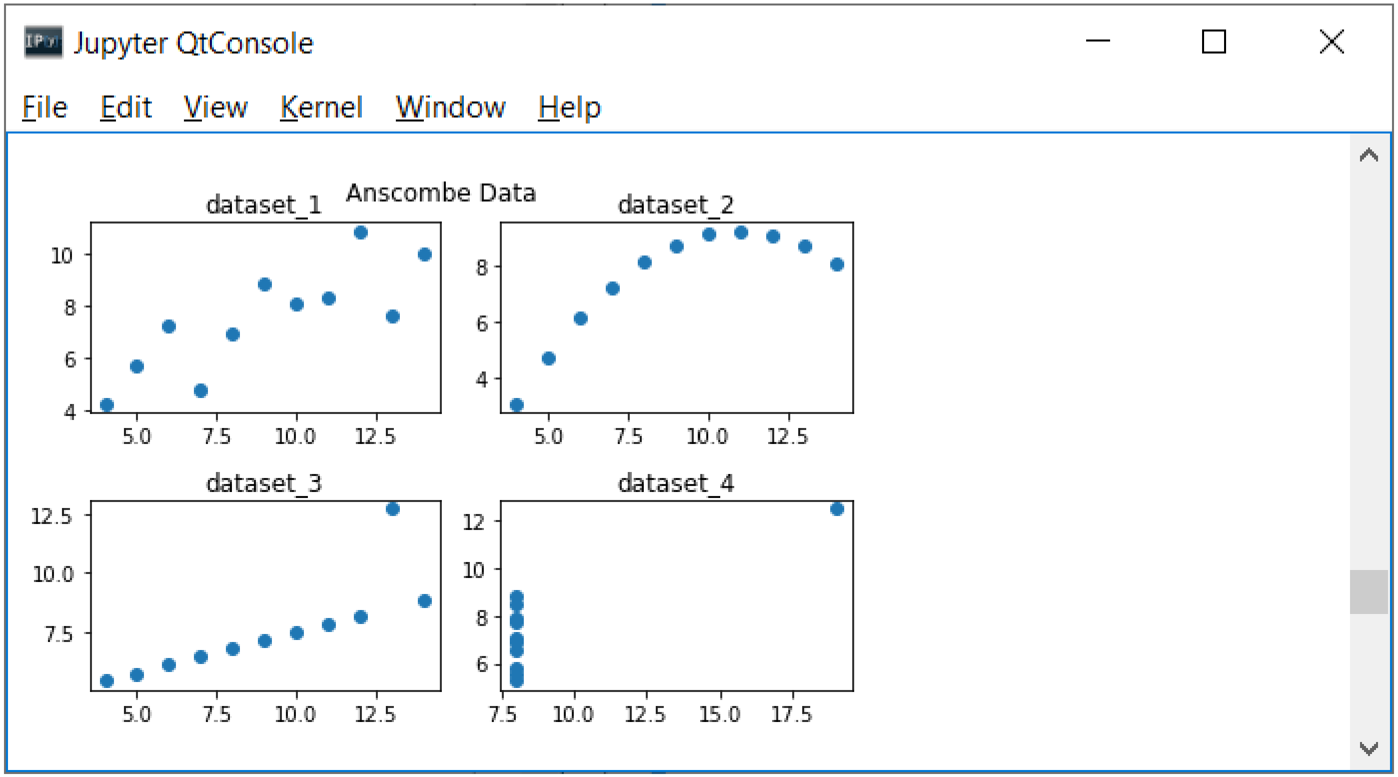
The Anscombe Data Set

Jupyter QtConsole

File Edit View Kernel Window Help

```
    ...:     print()
mean:
 x    9.000000
y    7.500909
dtype: float64
variance:
 x    11.000000
y     4.127269
dtype: float64
correlation:  0.81642051634484

mean:
 x    9.000000
y    7.500909
dtype: float64
variance:
 x    11.000000
y     4.127629
dtype: float64
correlation:  0.8162365060002428

mean:
 x    9.0
y    7.5
dtype: float64
variance:
 x    11.00000
y     4.12262
dtype: float64
correlation:  0.8162867394895984

mean:
 x    9.000000
y    7.500909
dtype: float64
variance:
 x    11.000000
y     4.123249
dtype: float64
correlation:  0.8165214368885028


In [31]:
```

Plot of the Anscombe Data Set



Matplotlib

- matplotlib is Python's fundamental plotting library

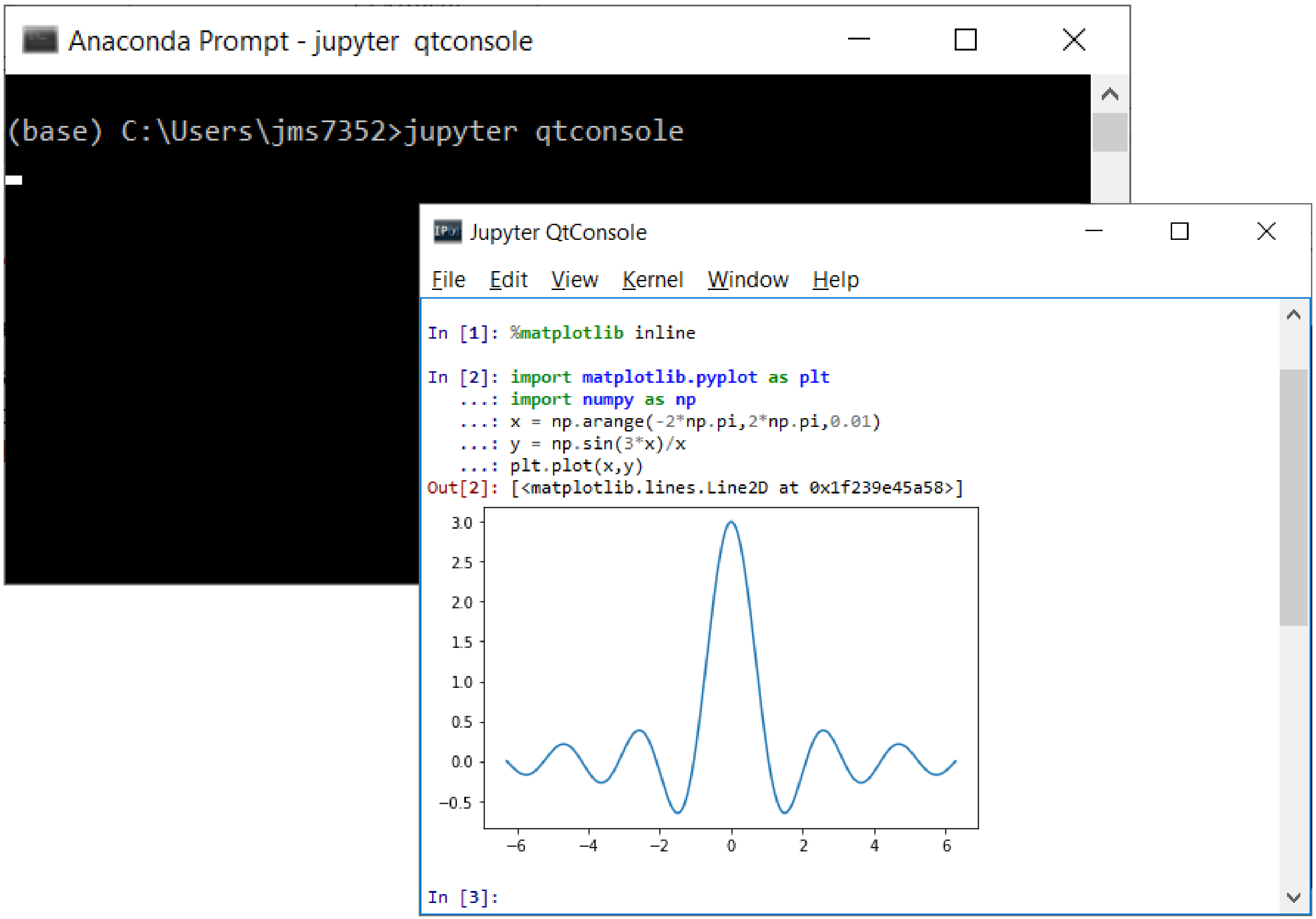
```
import matplotlib.pyplot as plt
```

- Calling plt.plot with two iterable objects of the same length (e.g., lists of numbers or NumPy arrays) creates a line plot

Launching the Qtconsole

- Open anaconda command prompt
- Type
  jupyter qtconsole
- To be able to se the charts in the console, type
  %matplotlib inline
- Same for jupyter notebook
- In jupyter notebook, you may need plt.show() after writing the code for a plot

Launching the Qtconsole



Creating a Line Plot

Trusted | Python 3

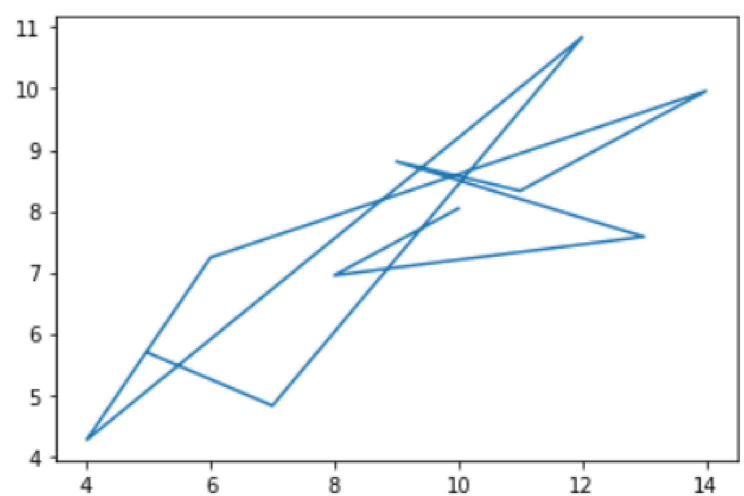
File Edit View Insert Cell Kernel Widgets Help

Run | Code

```
In [4]: import matplotlib.pyplot as plt
```

```
In [5]: # create a subset of the data
        # contains only data set 1 from anscombe
        dataset_1 = anscombe[anscombe['dataset'] == 'I']
        plt.plot(dataset_1['x'], dataset_1['y'])
```

```
Out[5]: [<matplotlib.lines.Line2D at 0x220d8f36d30>]
```



Jupyter Qt...

File Edit View Kernel Window

```
In [13]: dataset_1
Out[13]:
```

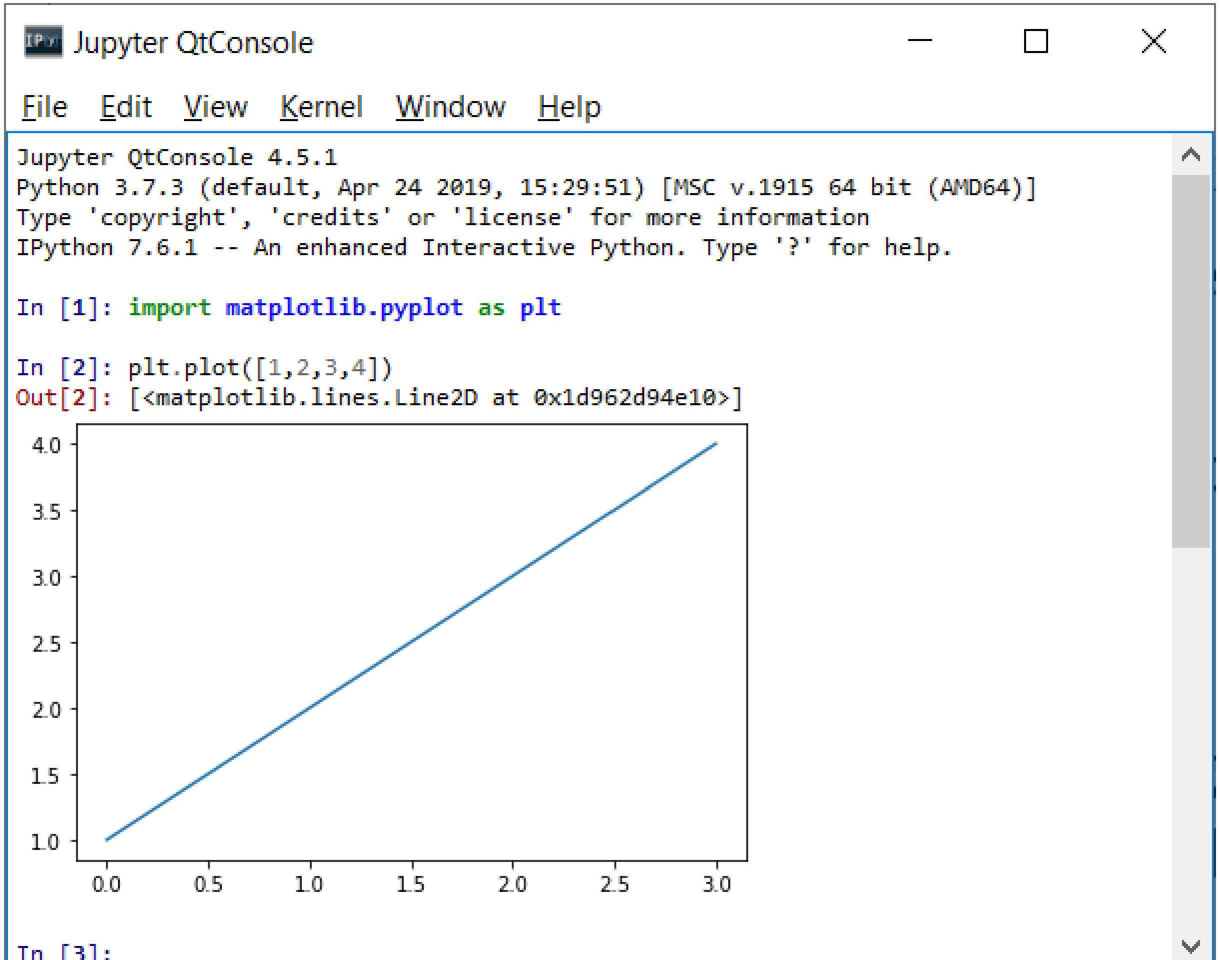
	dataset	x	y
0	I	10.0	8.04
1	I	8.0	6.95
2	I	13.0	7.58
3	I	9.0	8.81
4	I	11.0	8.33
5	I	14.0	9.96
6	I	6.0	7.24
7	I	4.0	4.26
8	I	12.0	10.84
9	I	7.0	4.82
10	I	5.0	5.68

```
In [14]:
```

Remarks

- A plot represents value pairs (x, y)
  - need two arrays
  - x contains values on the x-axis
  - y contains values on the y-axis
- If you pass only one array to the plt.plot() function, matplotlib assumes it is the sequence of y values
  - it associates the y values with the sequence x: 0,1,2,3, ...

Passing One Set of Values



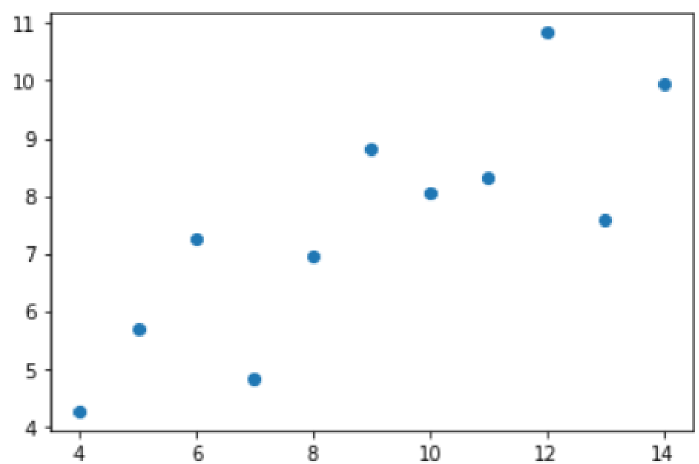
Creating a Scatter Plot

File Edit View Insert Cell Kernel Widgets Help

Run Code

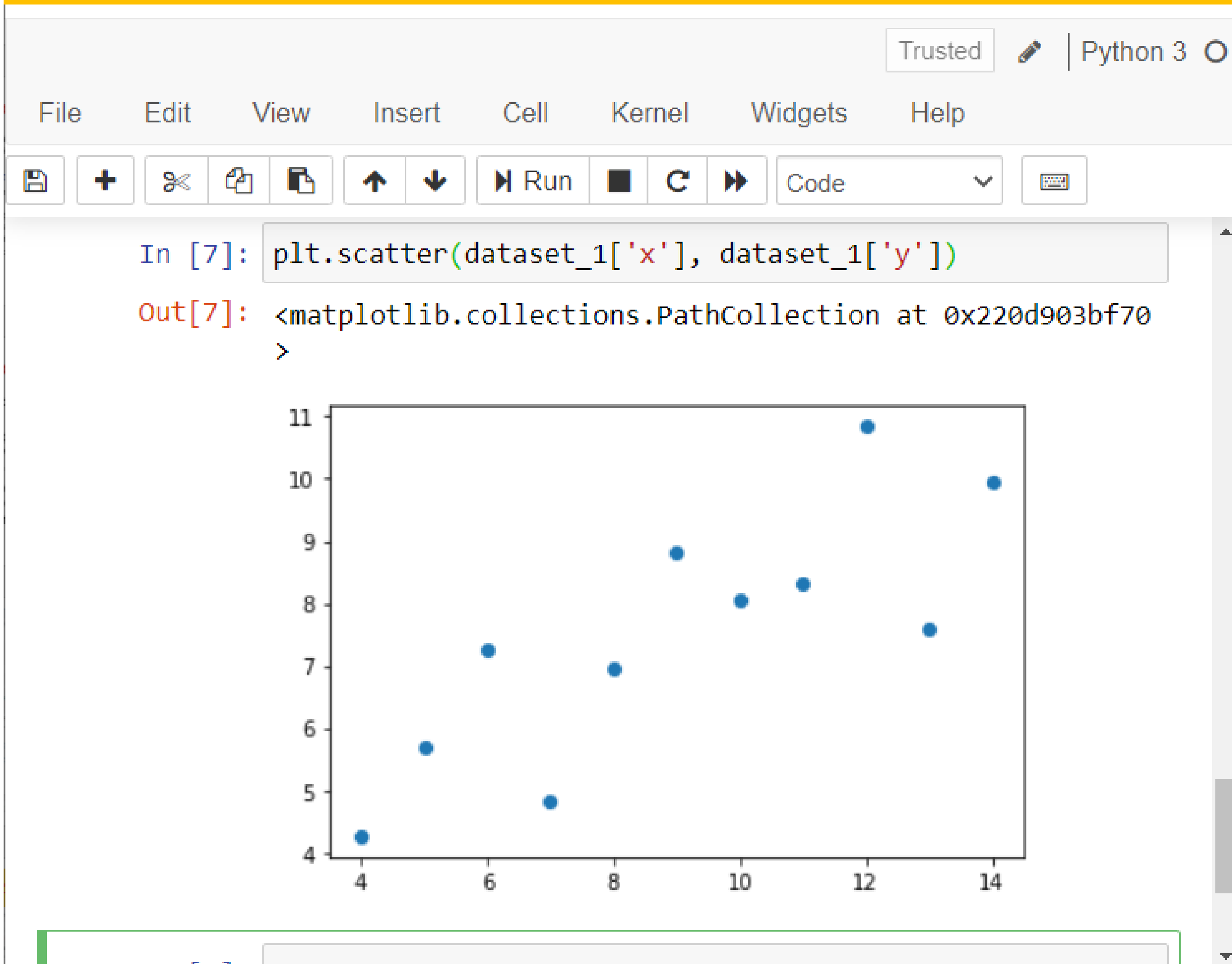
In [6]:
```
plt.plot(dataset_1['x'], dataset_1['y'], 'o')
```

Out[6]:
```
[<matplotlib.lines.Line2D at 0x220d8fe17c0>]
```



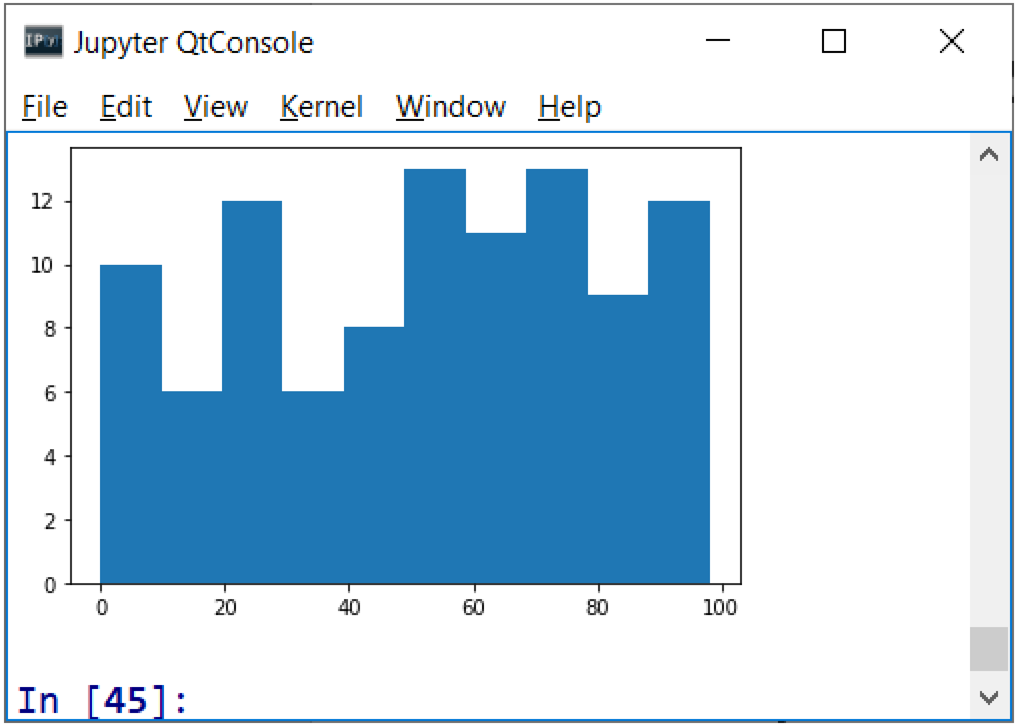
In [ ]:

Creating a Scatter Plot



Creating a Histogram

- A histograms shows the distribution of a variable
- pylot.hist(array, bins = n) draws a histogram

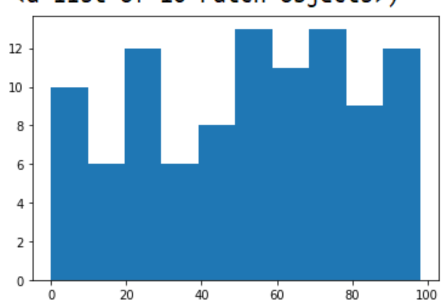


plt.hist()

Jupyter QtConsole

File Edit View Kernel Window Help

```
In [43]: import numpy as np
    ...: pop = np.random.randint(0,100,100)
    ...: pop
Out[43]:
array([60, 43, 58, 85, 59, 21, 42, 89, 91, 20, 98, 53,  0, 20, 50, 26, 71,
       11, 49, 31, 26, 59, 14, 19, 50, 96, 85, 85, 94, 20, 89, 20, 66, 81,
       19, 77, 83, 36, 33, 52,  9, 59, 11, 22, 42, 28, 71, 28, 57, 59, 51,
       70,  2, 74, 75,  8, 22,  5, 93, 88,  5, 45, 96, 33, 60, 78, 36, 55,
       33, 43, 47, 51, 17, 65, 91, 62, 80, 70, 50, 77,  8, 81, 61, 55, 24,
       90, 78, 53,  4, 77,  8,  1, 71, 76, 45, 47, 91, 86, 64, 92])

In [44]: plt.hist(pop, bins=10)
Out[44]:
(array([10.,  6., 12.,  6.,  8., 13., 11., 13.,  9., 12.]),
 array([ 0. ,  9.8, 19.6, 29.4, 39.2, 49. , 58.8, 68.6, 78.4, 88.2, 98. ]),
 <a list of 10 Patch objects>)
```




```
In [45]:
```

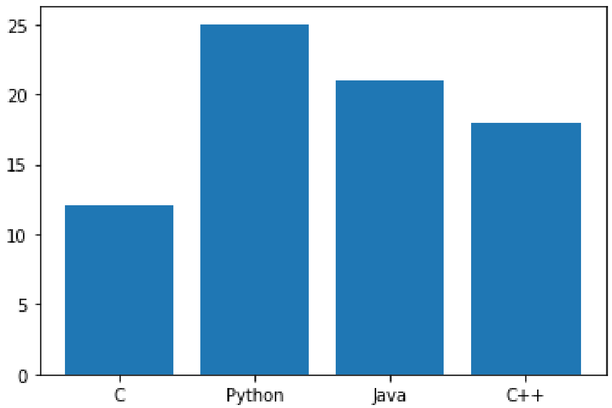
Creating a Bar Chart

- A bar chart can be used to compare variables
- Very example, to compare the favorite programming language for students in a CS class
- plt.bar(x, y)
- plt.barh(x, y) draws a horizontal bar chart

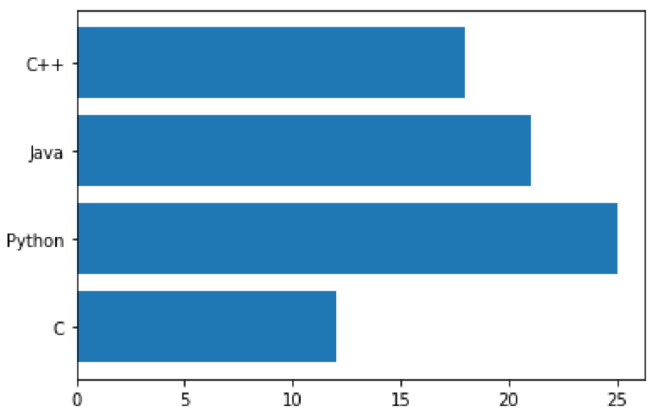
Jupyter QtConsole

File Edit View Kernel Window Help

```
In [45]: favorite_language = ['C', 'Python', 'Java', 'C++']
    ...: number_students = [12, 25, 21, 18]
    ...: plt.bar(favorite_language, number_students)
Out[45]: <BarContainer object of 4 artists>
```




```
In [46]: plt.barh(favorite_language, number_students)
Out[46]: <BarContainer object of 4 artists>
```



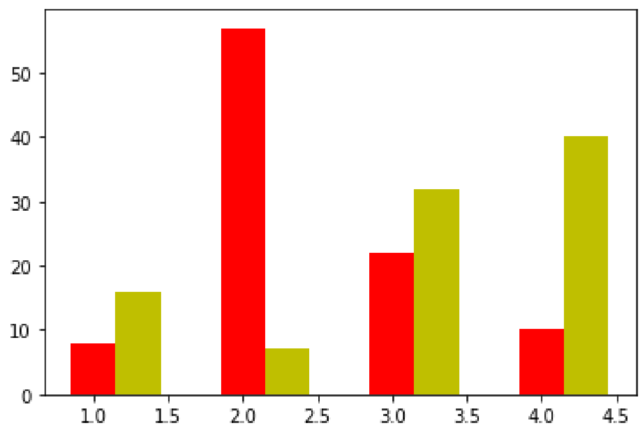

```
In [47]:
```

Multiple Bar Charts

Jupyter QtConsole

File Edit View Kernel Window Help

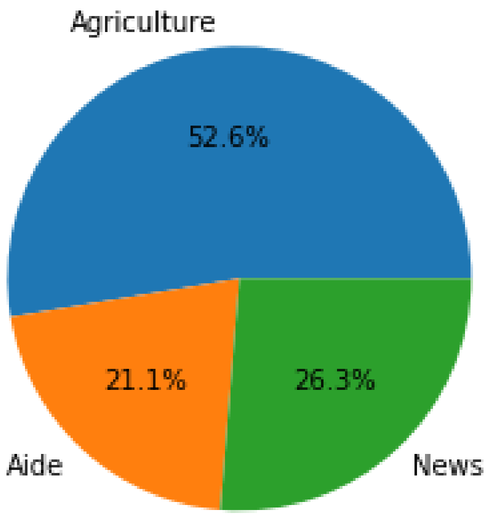
```
In [51]: # Bar chart comparing quarterly sales over two years
    ...: import numpy as np
    ...: quarters = np.arange(1, 5)
    ...: year1 = [8, 57, 22, 10]
    ...: year2 = [16, 7, 32, 40]
    ...: plt.bar(quarters, year1, color = 'r', width = 0.3)
    ...: plt.bar(quarters + 0.3, year2, color = 'y', width = 0.3)
Out[51]: <BarContainer object of 4 artists>
```




```
In [52]:
```

Pie Chart

- A pie chart represents the data as if to fit into a circle
- The individual data points are expressed as sectors of a circle that add up to 360 degrees.
- A pie chart chart is good for displaying categorical data and summaries
- data = [500, 200, 250] ; labels = ["Agriculture", "Aide", "News"]
- plt.pie(data, labels=labels, autopct=‘%0.1f')

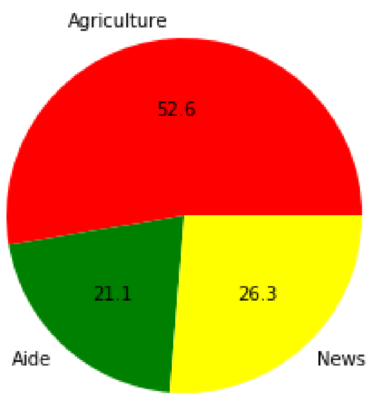


Pie Chart – colors & axis

Jupyter QtConsole

File Edit View Kernel Window Help

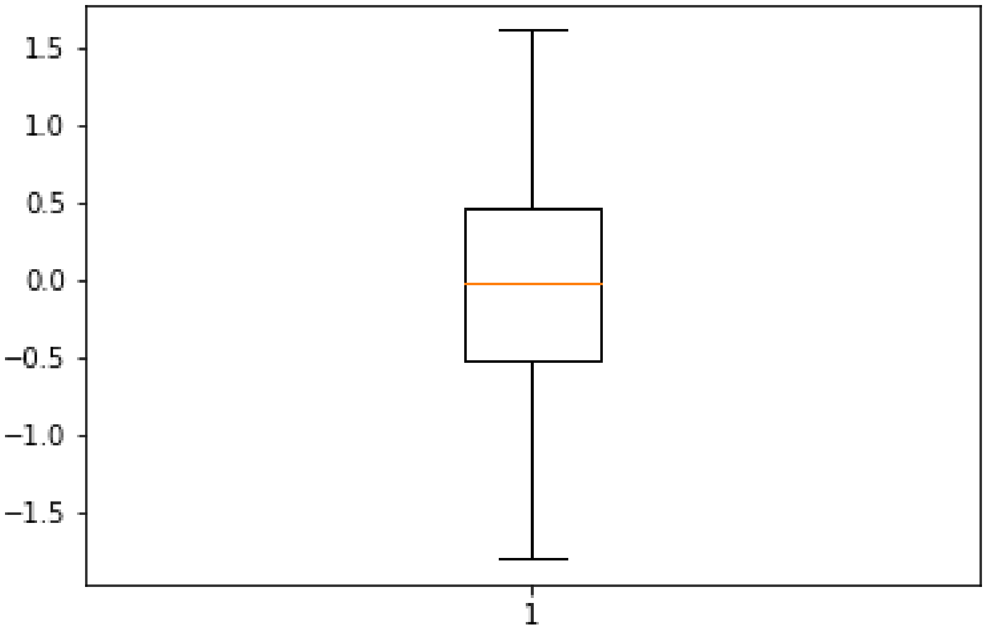
```
In [67]: data = [500, 200, 250]
    ...: labels = ["Agriculture", "Aide", "News"]
    ...: plt.pie(data, labels=labels, colors = ['red', 'green', 'yellow'], autopct='%0.1f')
    ...: plt.axis('equal')
Out[67]:
(-1.109577131582004,
 1.1004560810108412,
 -1.1088037134860909,
 1.1088037238273893)
```




```
In [68]:
```

Box Plot

- A box plot is used to visualize the median value and low and high ranges of a distribution
- data = np.random.randn(50) #Z(0,1)
- plt.boxplot(data)

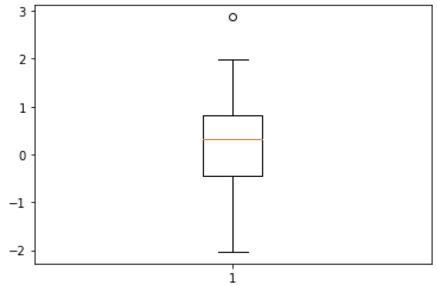


Box Plot

Jupyter QtConsole

File Edit View Kernel Window Help

```
In [68]: data = np.random.randn(50)

In [69]: data
Out[69]:
array([-0.05377518, -0.93720463,  0.39382558, -0.43101858, -0.36497506,
        0.83484773,  0.31094051, -0.41445169, -0.43602979,  0.67991143,
        1.10970474, -1.33534333,  0.54048272,  1.03834793, -0.73915858,
        0.82198128,  0.61442846,  1.7093854 ,  0.9193192 , -0.51540242,
       -0.46998441, -2.03363986, -0.14898594,  0.32354872,  1.0525657 ,
        0.82693804,  0.54274661,  0.33585519,  1.98949379,  0.60989897,
        2.87334636, -1.64489746, -0.79576625,  0.37307228, -0.66777782,
        0.38140648, -0.10218382, -1.00569866,  0.58048467,  0.25538954,
        0.95151777,  0.08977108,  0.60736969, -0.93529634,  1.25674806,
       -0.87832726,  1.25525136, -0.26461772,  1.22089877, -0.44162983])

In [70]: plt.boxplot(data)
Out[70]:
{'whiskers': [<matplotlib.lines.Line2D at 0x1f2407f4898>,
  <matplotlib.lines.Line2D at 0x1f2407f4c50>],
 'caps': [<matplotlib.lines.Line2D at 0x1f2407f4f98>,
  <matplotlib.lines.Line2D at 0x1f2407f4f28>],
 'boxes': [<matplotlib.lines.Line2D at 0x1f2407f44a8>],
 'medians': [<matplotlib.lines.Line2D at 0x1f2407fe668>],
 'fliers': [<matplotlib.lines.Line2D at 0x1f2407fe9b0>],
 'means': []}
```



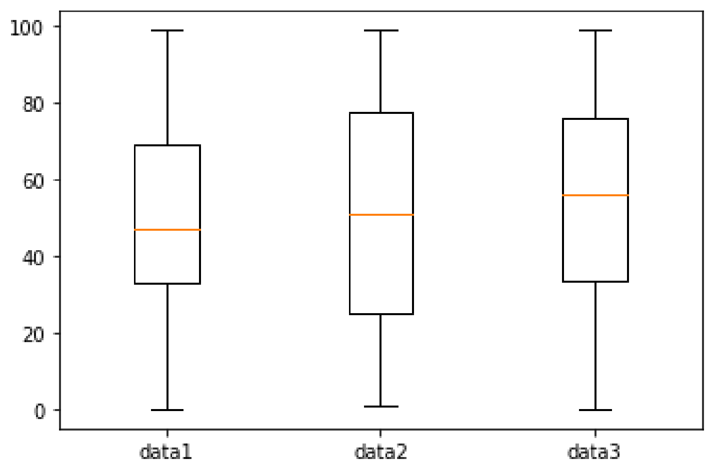

```
In [71]: data.max
Out[71]: <function ndarray.max>

In [72]: data.max()
Out[72]: 2.8733463569779207

In [73]: data.min()
Out[73]: -2.03363985914945

In [74]:
```

Displaying Multiple Box Plots

Jupyter QtConsole

File Edit View Kernel Window Help

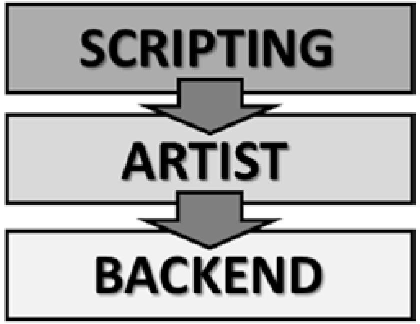
```
In [84]: data1 = np.random.randint(0,100,100)
    ...: data2 = np.random.randint(0,100,100)
    ...: data3 = np.random.randint(0,100,100)
    ...: data = [data1, data2, data3]
    ...: plt.boxplot(data, labels = ['data1', 'data2', 'data3'])
    ...: plt.savefig("boxplot2")
```




```
In [85]:
```

Architecture of matplotlib

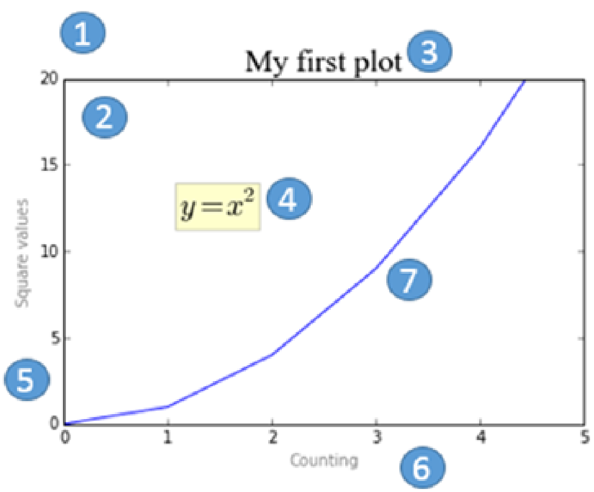
- The architecture of matplotlib is logically structured into three layers as in the figure

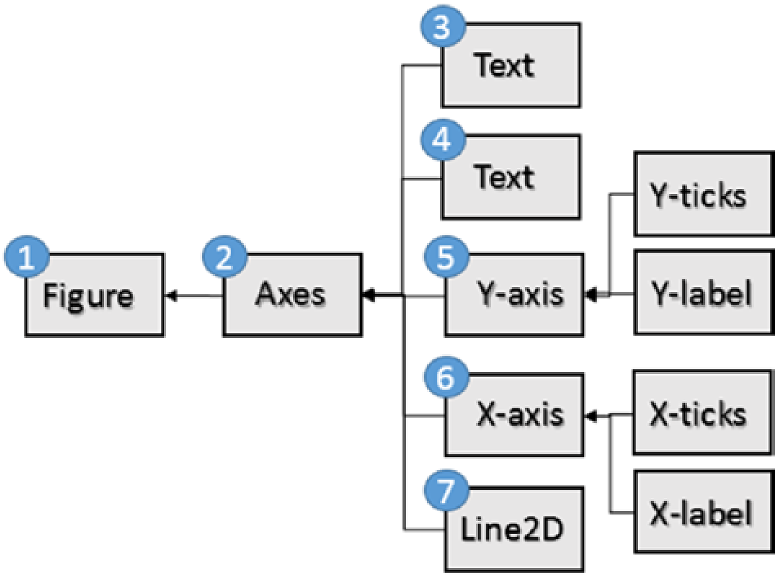


- The communication is unidirectional
- The three layers are:
  1. Scripting: as data analysts, you work at this layer
  2. Artist: see following slides
  3. Backend: classes that implement the graphic elements at a low level

Artist Layer

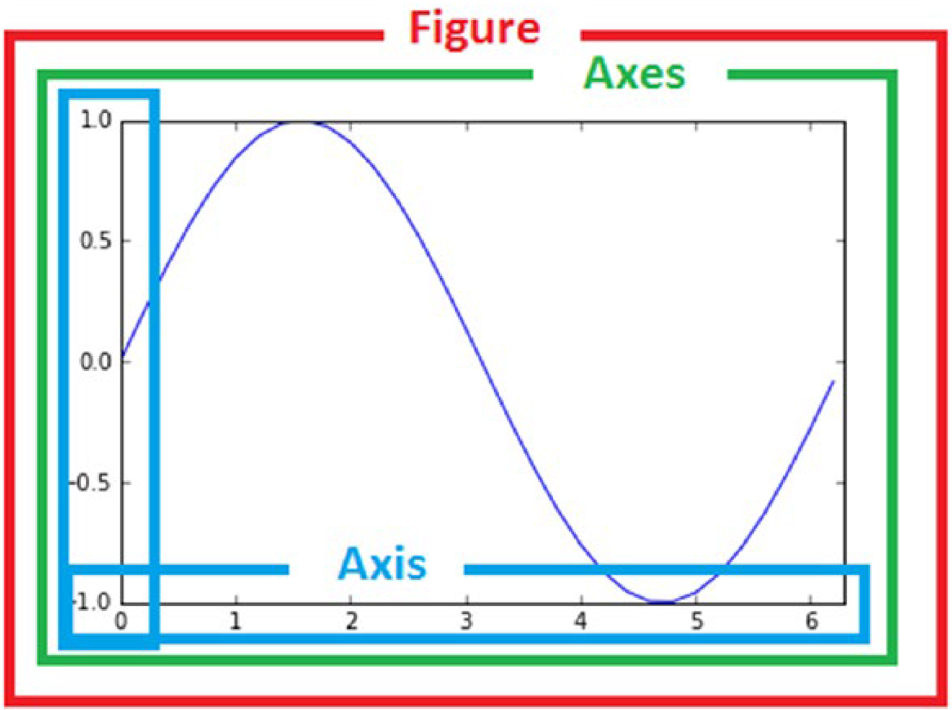
- All the elements that make up a chart, such as the title, axis labels, … etc., are instances of the Artist object with a hierarchical structure representing the different components on a chart as in the Figure below
- There are two types of Artist classes: primitive and composite





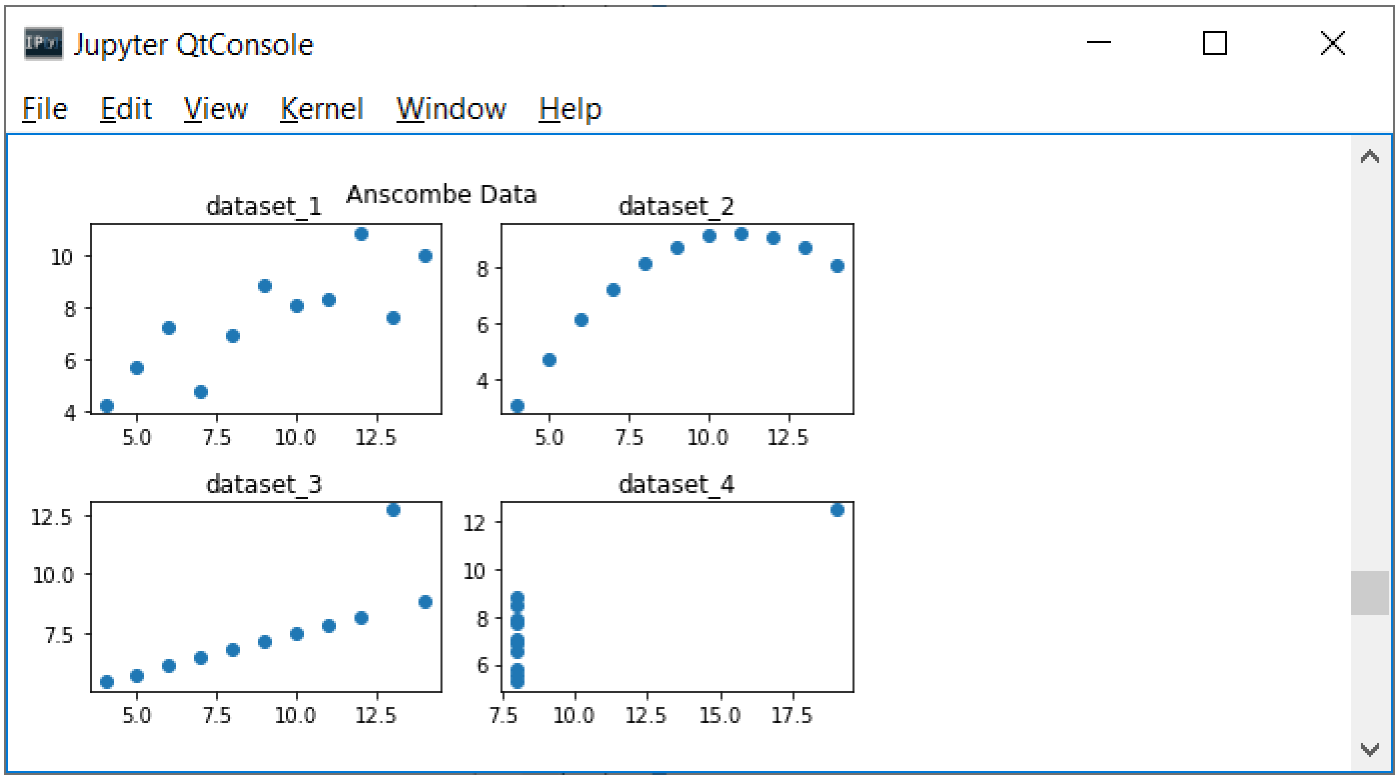
The Main Artist objects in the Hierarchy of the Artist Layer

1. Figure corresponds to the entire graphical representation and can contain many Axes
2. Axes corresponds to plot or chart.
Each Axes object belongs to only one Figure, and is characterized by two Artist Axis.
Objects such as title, x label, and the y label, belong to this composite artist.

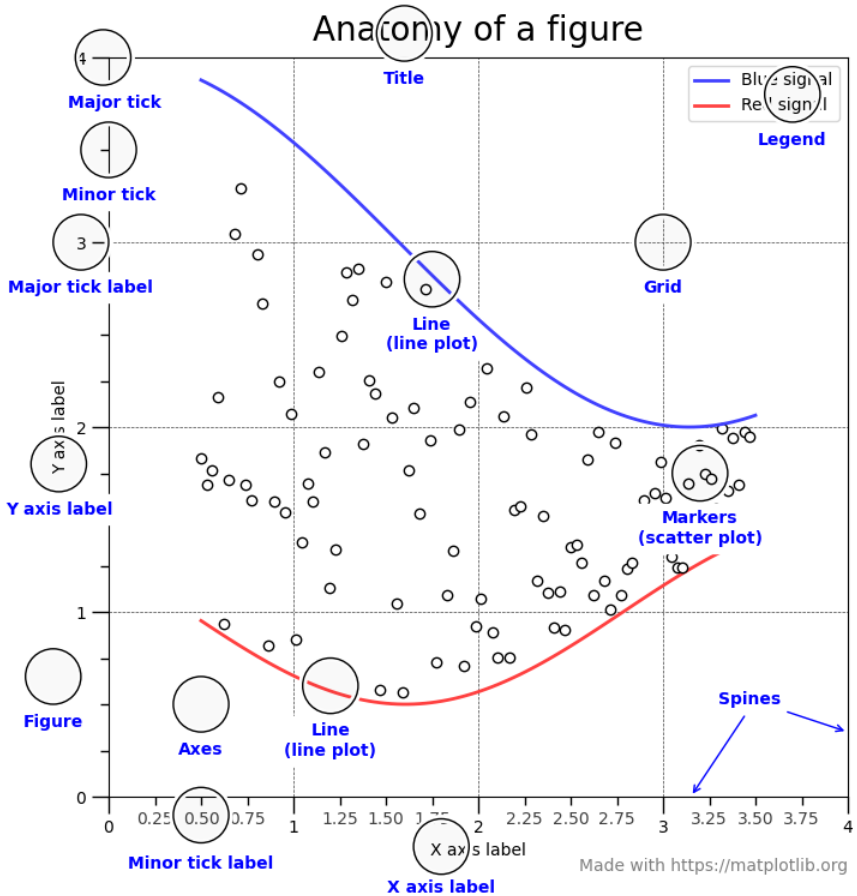


3. Axis objects that take into account the numerical values to be represented on the x axis and y axis, define the limits and manage the ticks and tick labels

Axes vs Axis


Jupyter QtConsole
File Edit View Kernel Window Help
Anscombe Data
dataset_1
dataset_2
dataset_3
dataset_4
10
8
6
4
5.0
7.5
10.0
12.5
12.5
10.0
7.5
12
15.0
17.5

Parts of a Figure

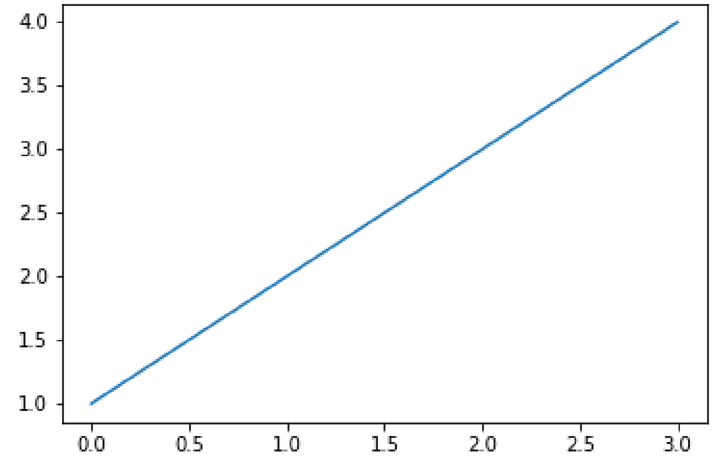


Scripting Layer (pyplot)

- The scripting layer is best suited for data analysts to manipulate and visualize the data
- This layer consists of an interface called **pyplot**

Adding More Features to Plots

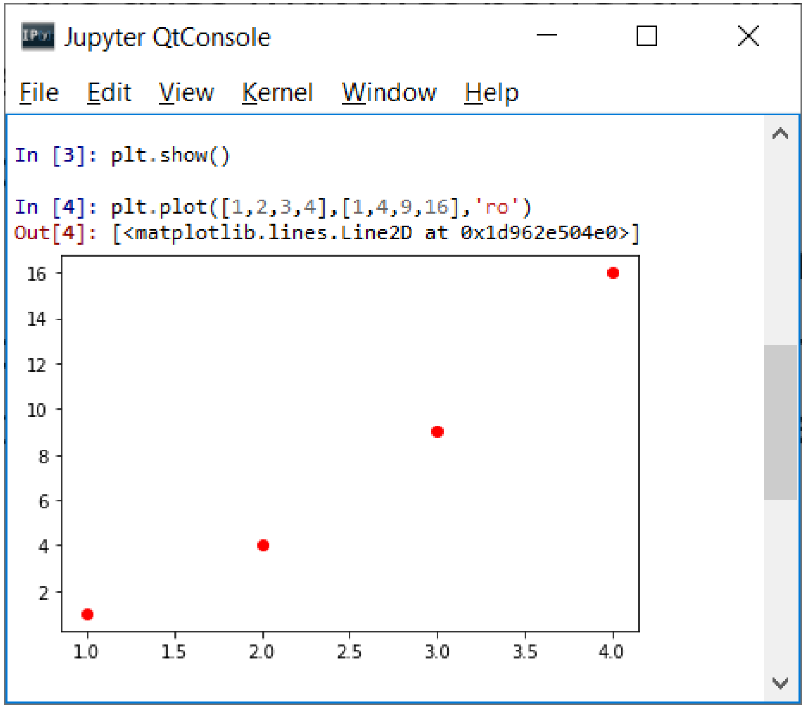
- plt.plot([1,2,3,4])
- Default configuration:
- Blue line
- Size of axis matches range of input values



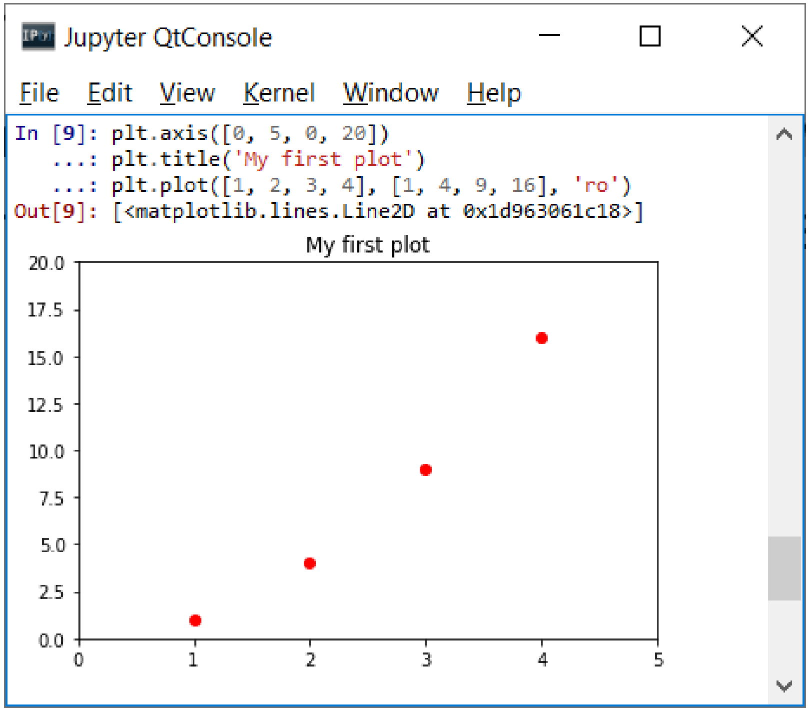
- There is neither a title nor axis labels
- There is no legend

Setting Dimensions & Title

- Specify the dimensions of the plot by calling plt.axis([xmin, xmax, ymin, ymax])
- A title can be specified using plt.title()



Before

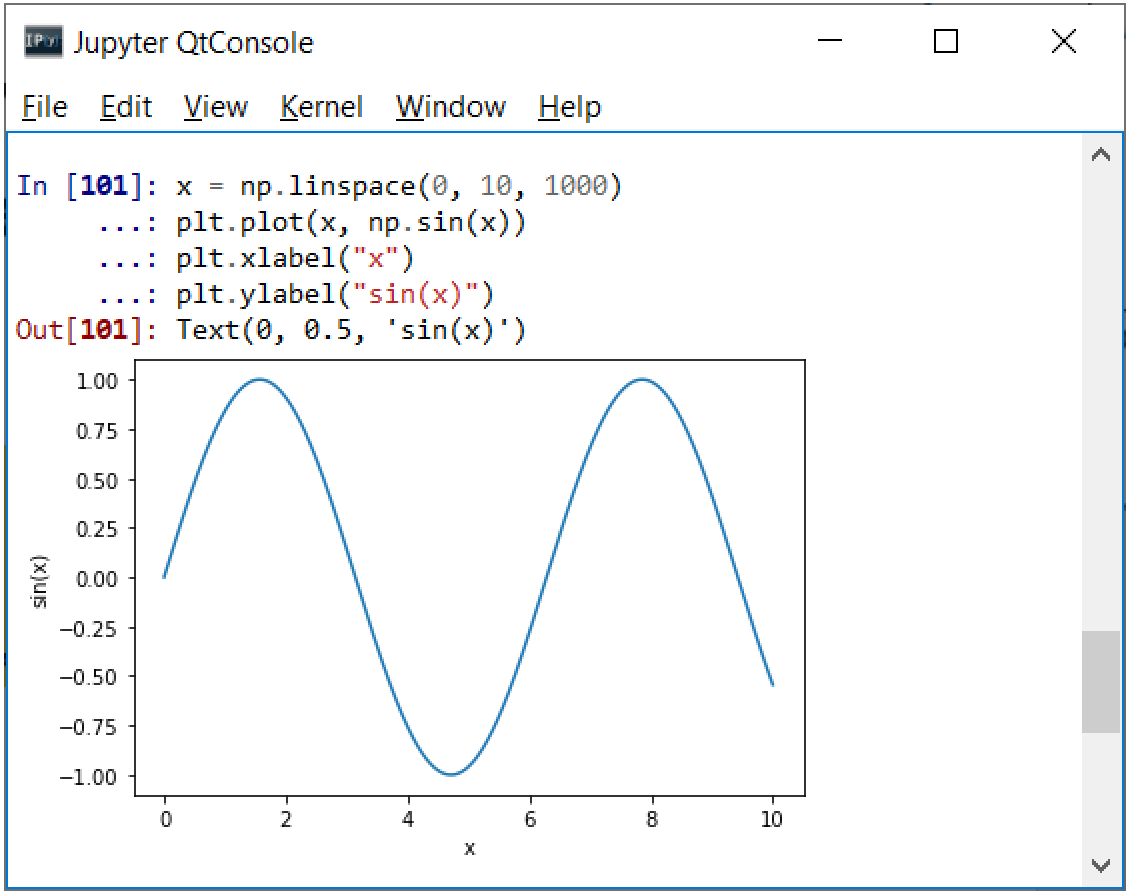


After

Adding Axis Labels

- plt.xlabel("string")
- plt.ylabel("string")

- np.linspace creates an array of regularly spaced values inclusive of the end points




```
In [15]: x = np.linspace(2, 3, 11)
         x
Out[15]: array([ 2. , 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9, 3. ])
```

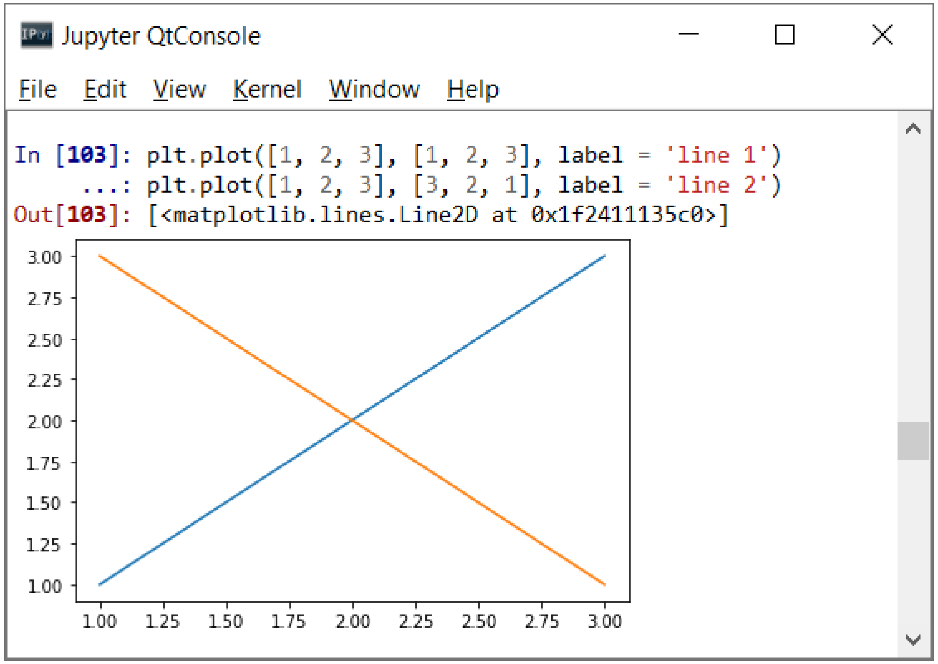
Adding Legend & Line Labels

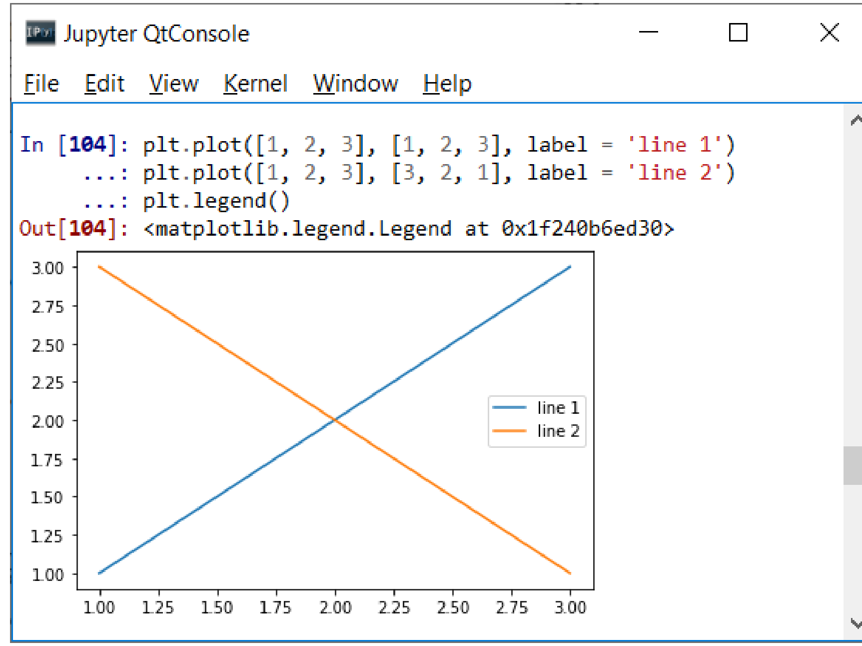
- Each line in a plot can be given a label by using the label argument of the plot function
- The label won't appear on the plot unless you also call plt.legend() to add a legend
- Legend location can be customized by using a loc argument of the legend method
- loc takes a string or integer value as in the table in the next slide

Values of loc argument

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

Labels and Legends - Example



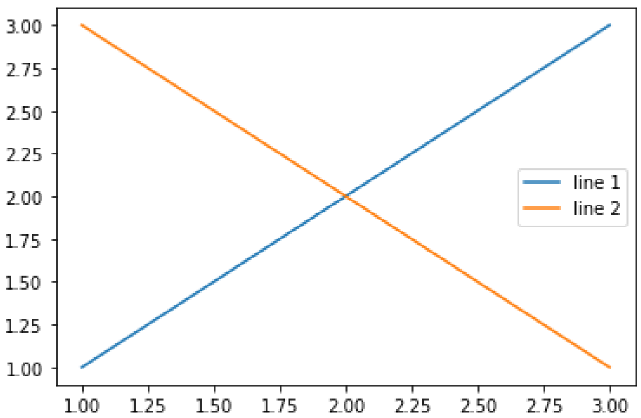


Labels and Legends - Example

Jupyter QtConsole

File Edit View Kernel Window Help

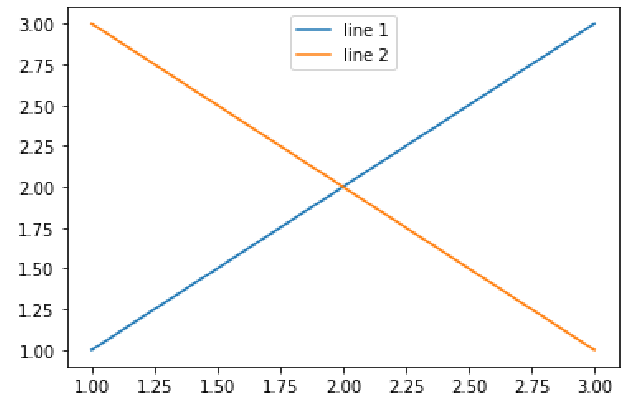
```
In [105]: plt.plot([1, 2, 3], [1, 2, 3], label = 'line 1')
     ...: plt.plot([1, 2, 3], [3, 2, 1], label = 'line 2')
     ...: plt.legend(loc=0)
Out[105]: <matplotlib.legend.Legend at 0x1f243611198>
```



Jupyter QtConsole

File Edit View Kernel Window Help

```
In [106]: plt.plot([1, 2, 3], [1, 2, 3], label = 'line 1')
     ...: plt.plot([1, 2, 3], [3, 2, 1], label = 'line 2')
     ...: plt.legend(loc='upper center')
Out[106]: <matplotlib.legend.Legend at 0x1f24302bf60>
```

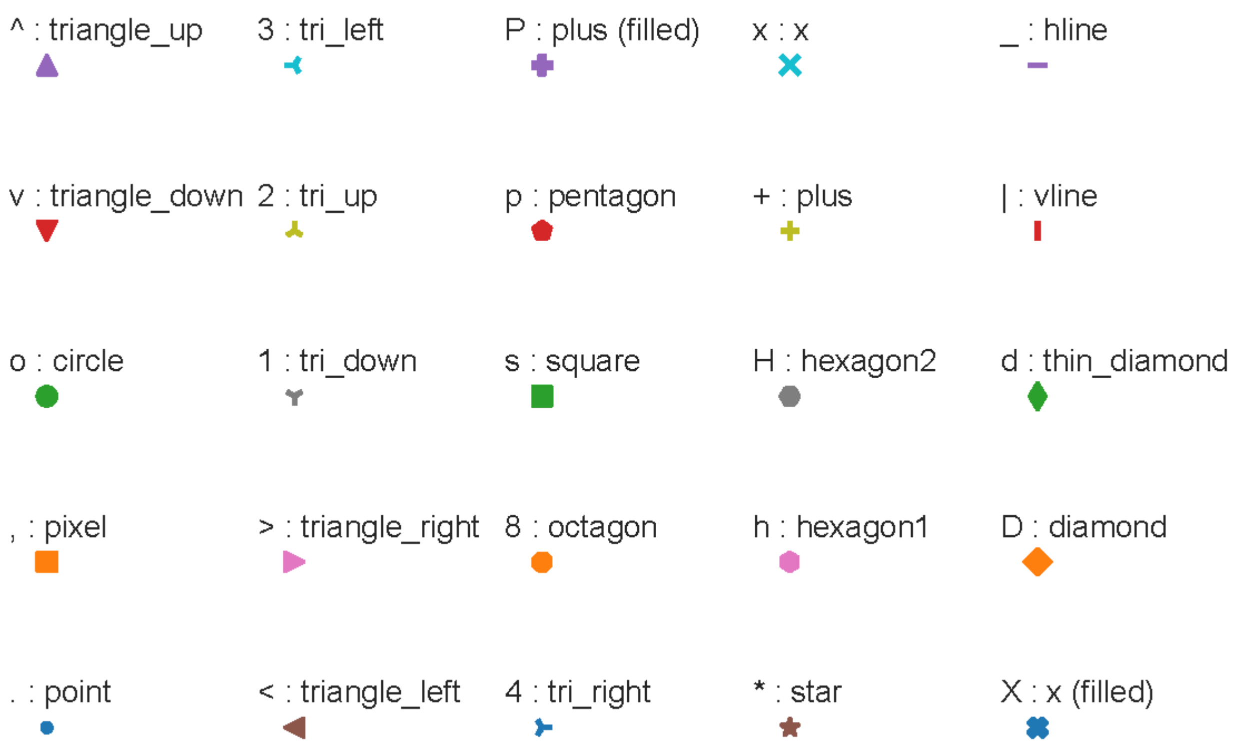



```
In [107]:
```

Customizing Plots

- The plot attributes
  - **marker** is used to add a marker
  - **color** is used to add a color

Some Matplotlib marker Styles

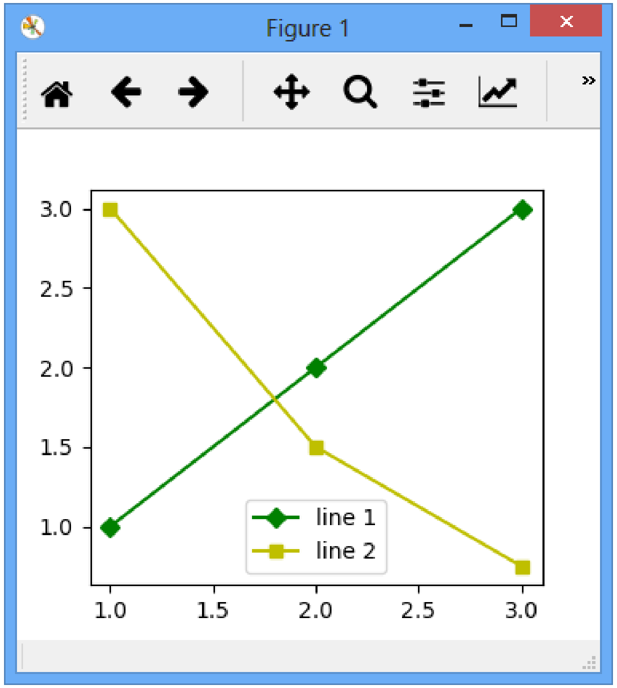


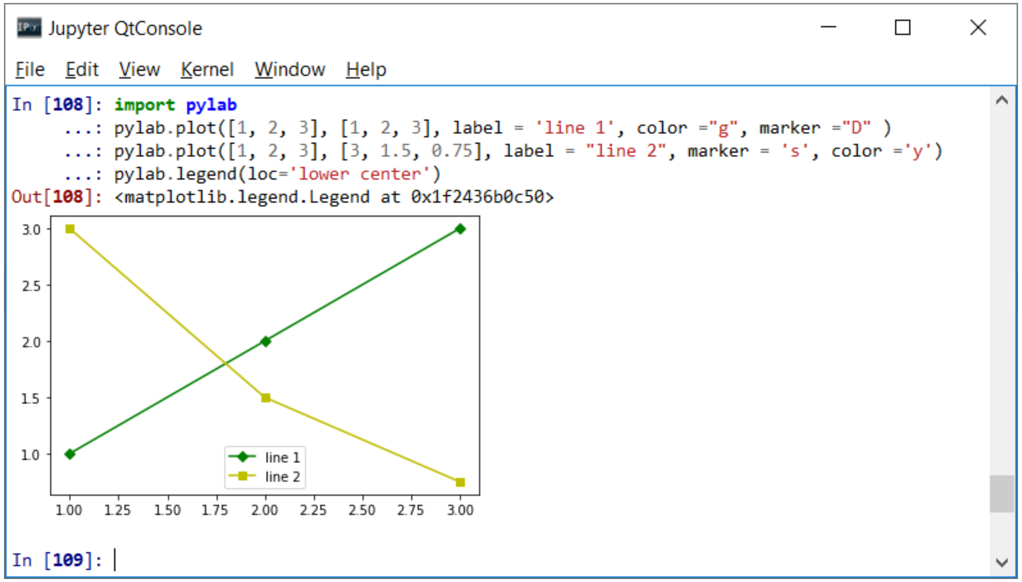
Some Matplotlib color Code Letters

- Can use several formats to specify colors
  - as in html (e.g., color ='#rrggbb')
  - shades of gray can be specified as a string representing a float in the range '0.0' to '1.0' ('0.0' black to '1.0' white)
- For basic built-in colors, you can use a single letter
  - b: blue
  - g: green
  - r: red
  - c: cyan
  - m: magenta
  - y: yellow
  - k: black
  - w: white

Markers and Colors - Example

```
import pylab
pylab.plot([1, 2, 3], [1, 2, 3], label = 'line 1', color ="g", marker ="D" )
pylab.plot([1, 2, 3], [3, 1.5, 0.75], label = "line 2", marker = 's', color ='y')
pylab.legend(loc='lower center')
```



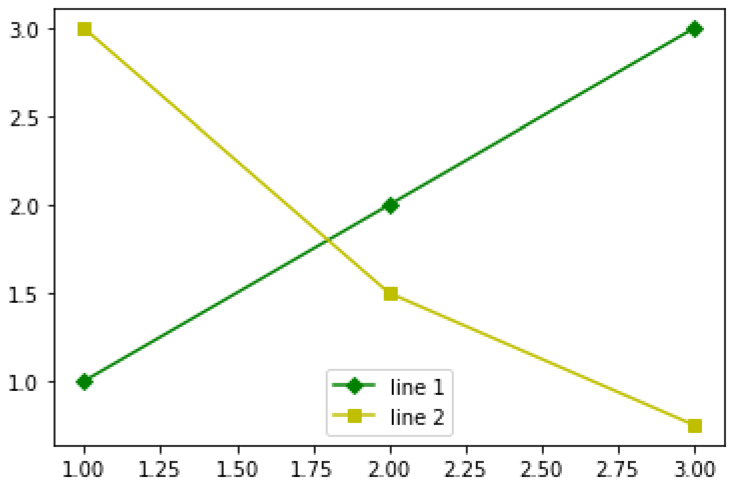


Markers and Colors - Example

Jupyter QtConsole

File Edit View Kernel Window Help

```
In [110]: plt.plot([1, 2, 3], [1, 2, 3], label = 'line 1', color ="g", marker ="D" )
     ...: plt.plot([1, 2, 3], [3, 1.5, 0.75], label = "line 2", marker = 's', color ='y')
     ...: plt.legend(loc='lower center')
Out[110]: <matplotlib.legend.Legend at 0x1f2436c9908>
```



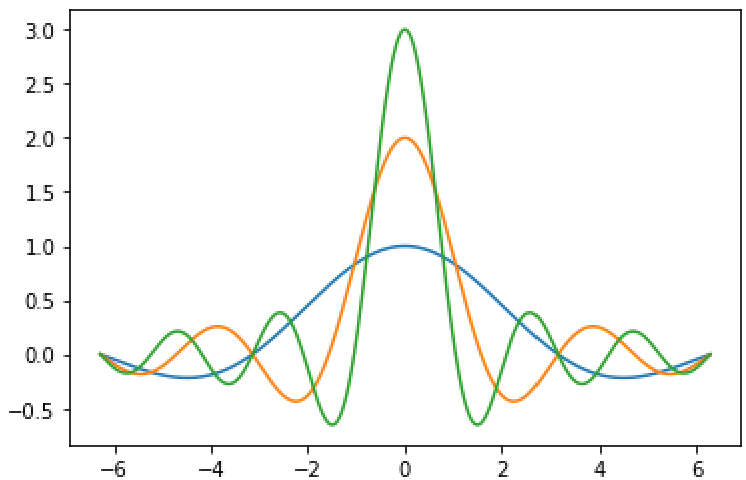

```
In [111]:
```

Linewidth & linstyle

Jupyter QtConsole

File Edit View Kernel Window Help

```
In [3]: x = np.arange(-2*np.pi,2*np.pi,0.01)
   ...: y = np.sin(x)/x
   ...: y2 = np.sin(2*x)/x
   ...: y3 = np.sin(3*x)/x
   ...: plt.plot(x,y)
   ...: plt.plot(x,y2)
   ...: plt.plot(x,y3)
Out[3]: [<matplotlib.lines.Line2D at 0x1f239ec54a8>]
```




```
In [4]:
```

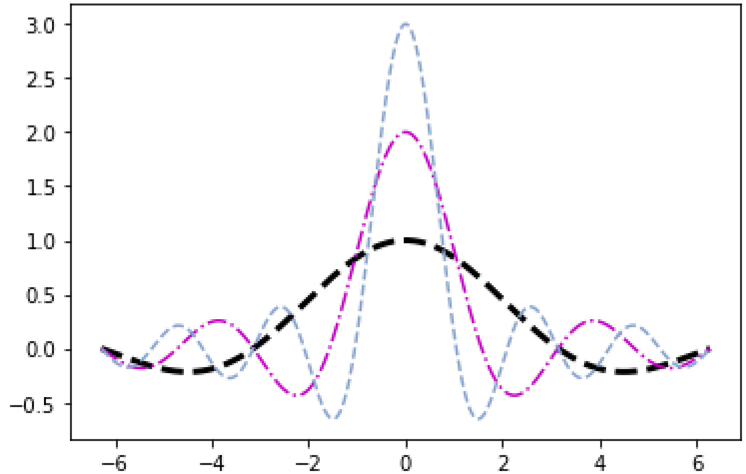
Linewidth & linstyle (cont)

- plot attributes **linewidth** and **linestyle**

Jupyter QtConsole

File Edit View Kernel Window Help

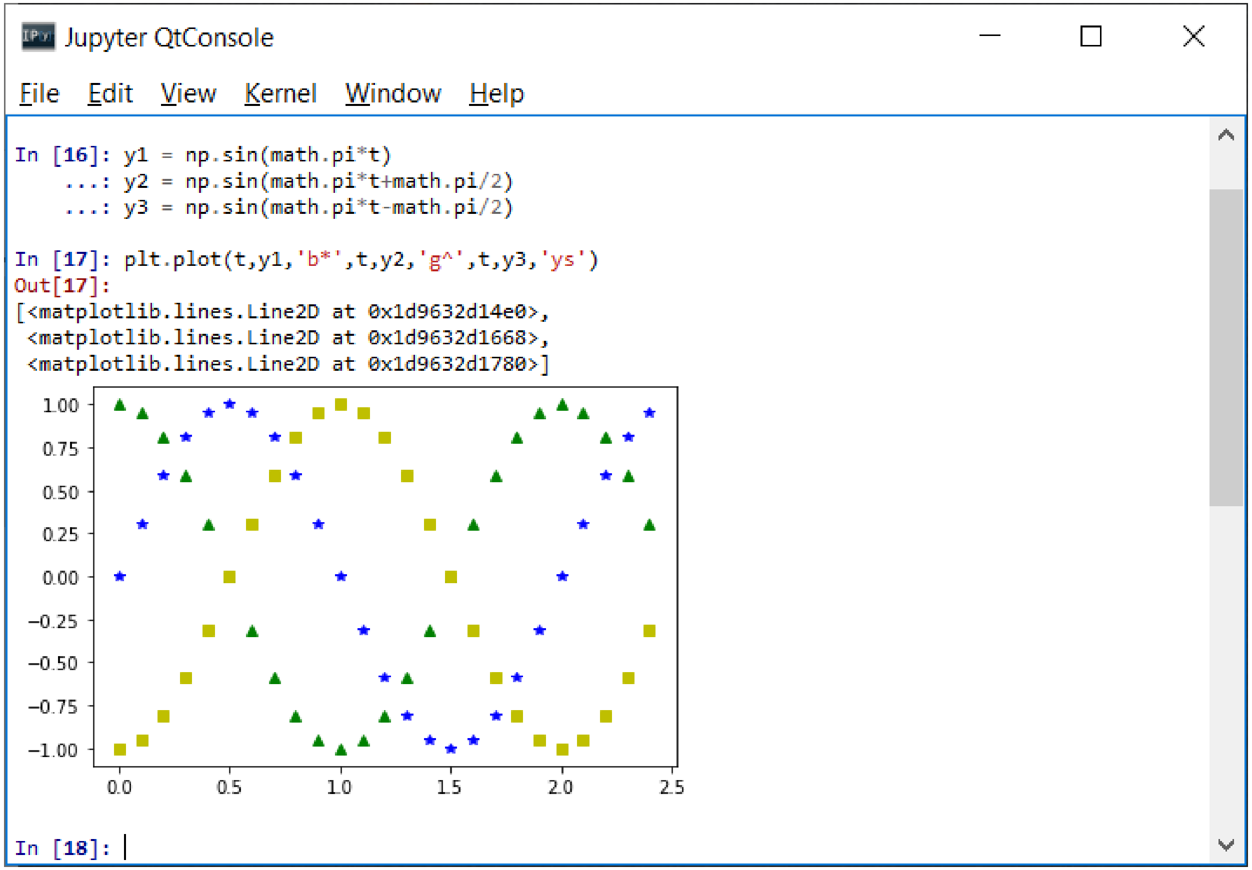
```
In [4]: x = np.arange(-2*np.pi,2*np.pi,0.01)
   ...: y = np.sin(x)/x
   ...: y2 = np.sin(2*x)/x
   ...: y3 = np.sin(3*x)/x
   ...: plt.plot(x,y,'k--',linewidth=3)
   ...: plt.plot(x,y2,'m-.')
   ...: plt.plot(x,y3,color='#87a3cc',linestyle='--')
Out[4]: [<matplotlib.lines.Line2D at 0x1f239f77e48>]
```




```
In [5]:
```

Linewidth & linstyle (cont)

- We can pass a third argument to the plot() function to specify some codes that correspond to the colors, styles, … etc.



Working with Multiple Subplots

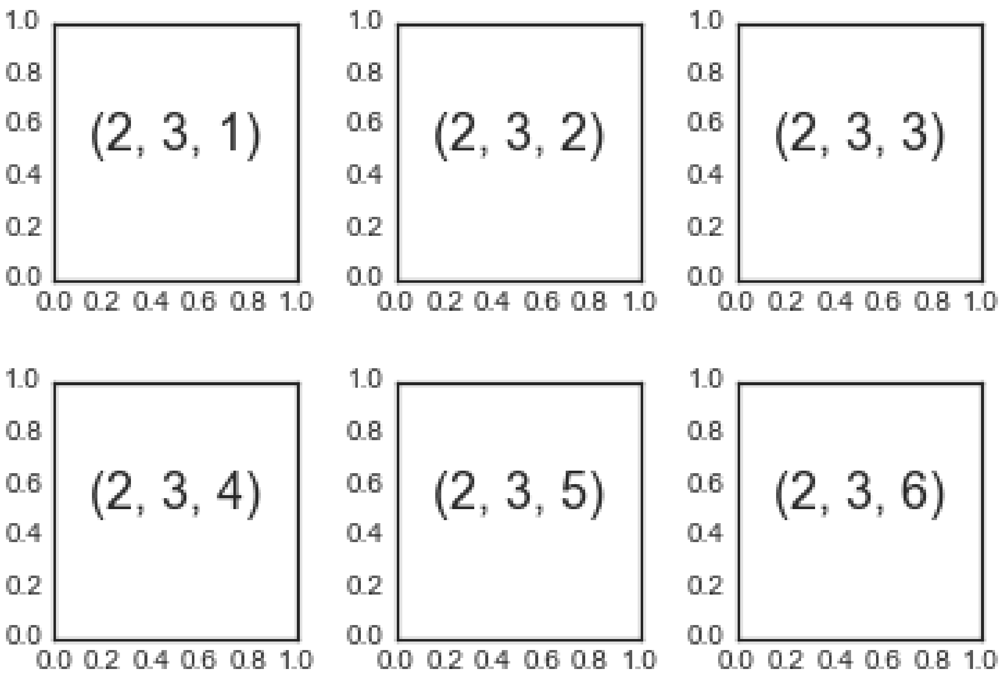
- Within a figure, you can have multiple subplots
- Each subplot specifies a different drawing region
- Each subplot corresponds to a separate Axes object
- One way of accomplishing this is by using pyplot.subplot()

The Function pyplot.subplot()

- The subplot() function, takes either a 3-digit integer or 3-separate integers as parameters
- They specify how the figure is divided into different subplot regions, and which subplot is active and will receive the next pyplot command
  - The first digit or number **specifies the number of rows**
  - The second digit or number **specifies the number of columns**
  - The third digit or number is **the index/location of the current subplot** that will receive pyplot ommands

The Function pyplot.subplot() (cont.)

- The subplot **location/index** is sequentially numbered, and plots are placed first in a left-to-right direction, then from top to bottom

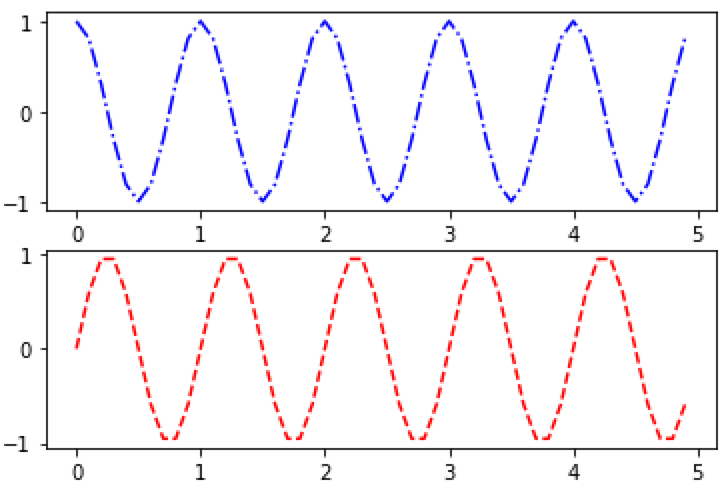


Multiple Subplots Example

Jupyter QtConsole

File Edit View Kernel Window Help

```
In [22]: t = np.arange(0,5,0.1)
    ...: y1 = np.cos(2*np.pi*t)
    ...: y2 = np.sin(2*np.pi*t)
    ...:
    ...: plt.subplot(211)
    ...: plt.plot(t,y1,'b-.')
    ...: plt.subplot(212)
    ...: plt.plot(t,y2,'r--')
Out[22]: [<matplotlib.lines.Line2D at 0x1d962da5b00>]
```



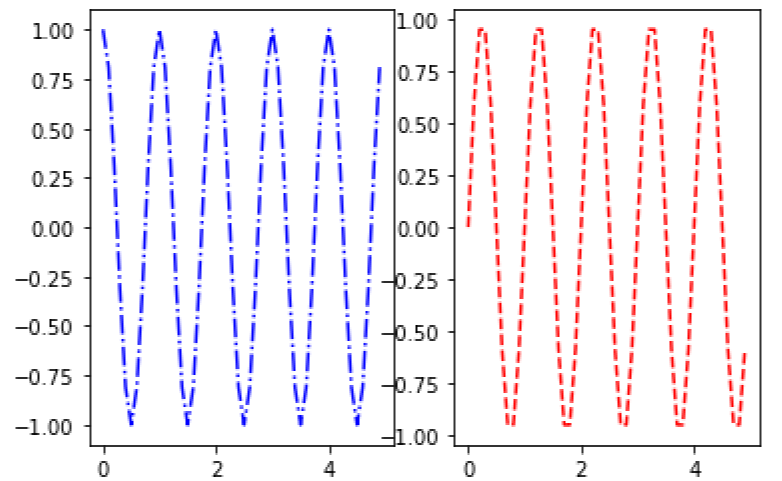

```
In [23]:
```

The figure has been divided into two vertical subplots

Jupyter QtConsole

File Edit View Kernel Window Help

```
In [23]: plt.subplot(121)
    ...: plt.plot(t,y1,'b-.')
    ...: plt.subplot(122)
    ...: plt.plot(t,y2,'r--')
Out[23]: [<matplotlib.lines.Line2D at 0x1d9634c5a58>]
```




```
In [24]:
```

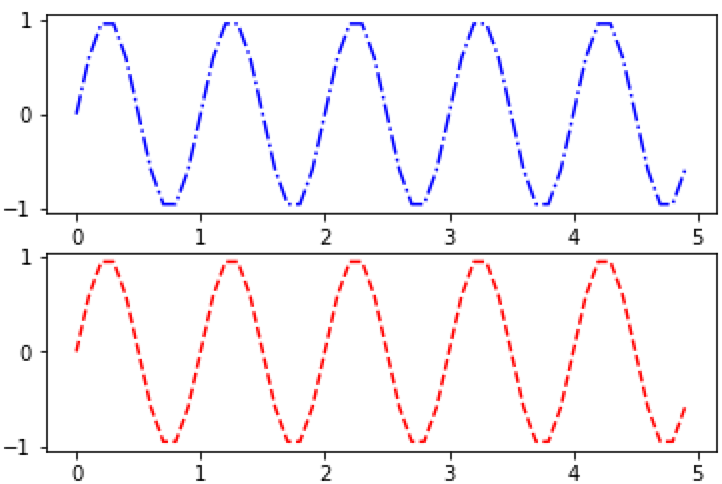
The figure has been divided into two horizontal subplots

pyplot.subplot() – Specifying Parameters As Separate Integers

Jupyter QtConsole

File Edit View Kernel Window Help

```
In [24]: # two rows and 1 column of subplots
    ...: plt.subplot(2, 1, 1)
    ...: plt.plot(t,y1,'b-.')
    ...: plt.subplot(2, 1, 2)
    ...: plt.plot(t,y2,'r--')
    ...:
    ...:
Out[24]: [<matplotlib.lines.Line2D at 0x1d36500ed68>]
```

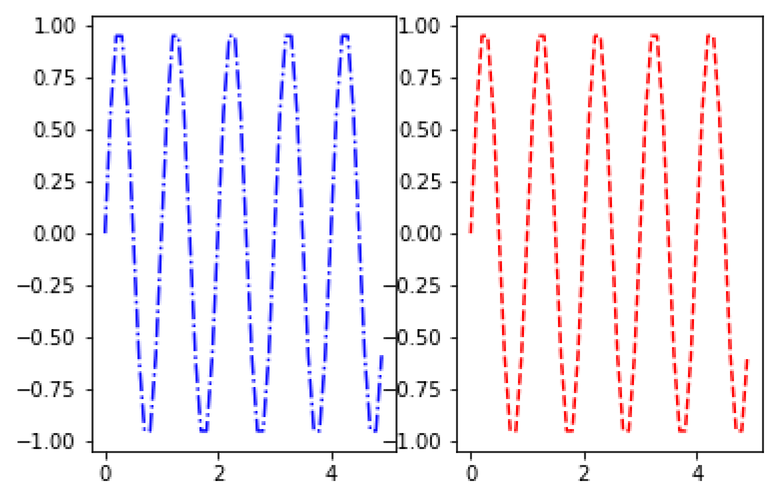



```
In [25]:
```

Jupyter QtConsole

File Edit View Kernel Window Help

```
In [25]: # one row and 2 columns of subplots
    ...: plt.subplot(1, 2, 1)
    ...: plt.plot(t,y1,'b-.')
    ...: plt.subplot(1, 2, 2)
    ...: plt.plot(t,y2,'r--')
    ...:
    ...:
Out[25]: [<matplotlib.lines.Line2D at 0x1d363d6f4e0>]
```




```
In [26]:
```

Subplots – Alternative Approach

- Create a figure object

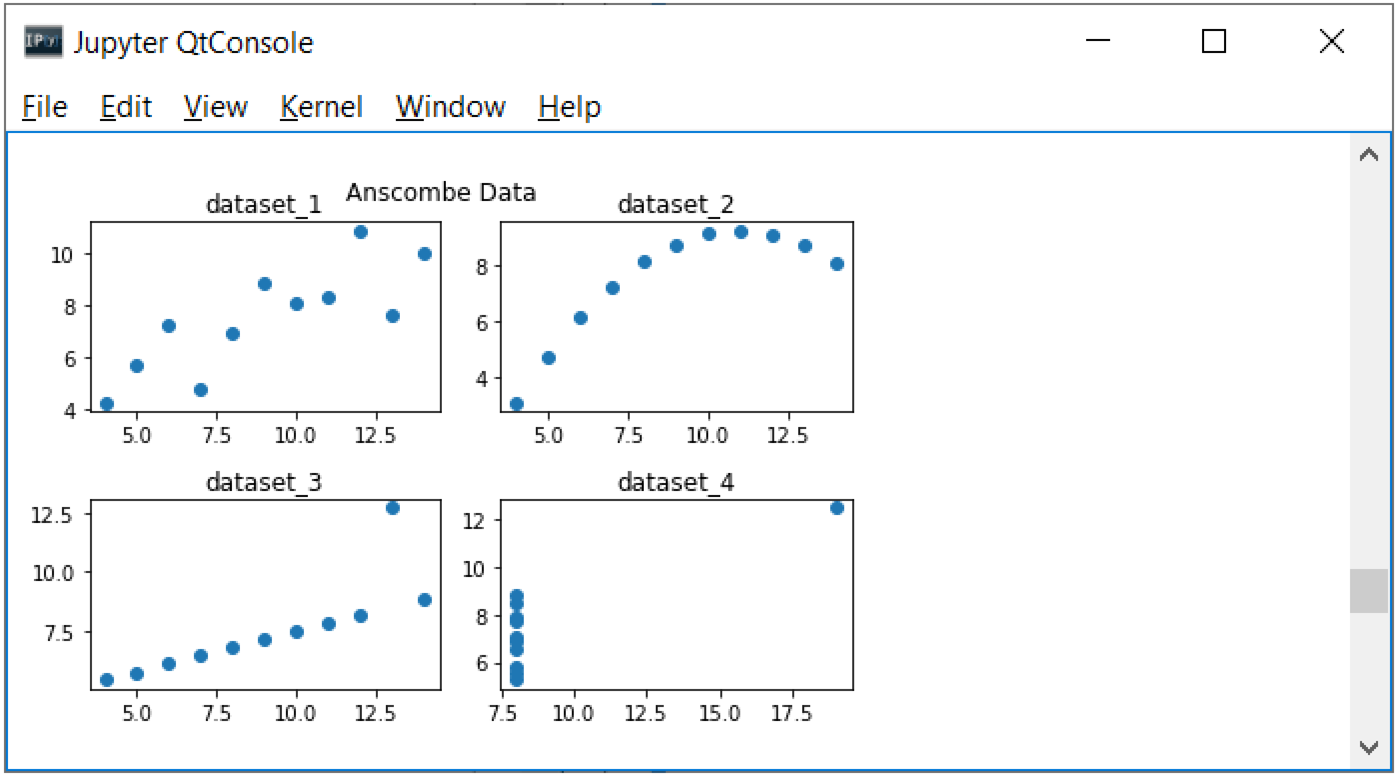
```
fig = plt.figure(figsize=(6,4))
```

  - figsize: specifies figure dimension (width, height) in inches; default is [6.4, 4.8]

- Add a subplot

```
ax = fig.add_subplot(nrows, ncols, index)
```

Example – Anscombe Data



Loading the Data

Jupyter QtConsole

File Edit View Kernel Window Help

```
In [27]: # load the Anscombe dataset
    ...: import seaborn as sns
    ...: anscombe = sns.load_dataset("anscombe")
    ...: #print(anscombe)
    ...:
    ...: # create subsets of the anscombe data
    ...: dataset_1 = anscombe[anscombe['dataset'] == 'I']
    ...: dataset_2 = anscombe[anscombe['dataset'] == 'II']
    ...: dataset_3 = anscombe[anscombe['dataset'] == 'III']
    ...: dataset_4 = anscombe[anscombe['dataset'] == 'IV']
    ...:
    ...:
```

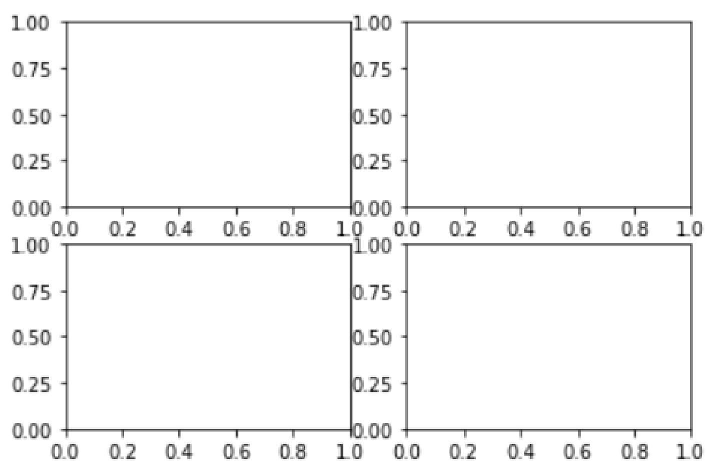
Specifying the Subplots

File Edit View Insert Cell Kernel Widgets Help

Run Code

In [8]:

```
# create the entire figure where our subplots will go
fig = plt.figure()
# tell the figure how the subplots should be laid out
# in the example, we will have
# 2 row of plots, and each row will have 2 plots
# subplot has 2 rows and 2 columns, plot location 1
axes1 = fig.add_subplot(2, 2, 1)
# subplot has 2 rows and 2 columns, plot location 2
axes2 = fig.add_subplot(2, 2, 2)
# subplot has 2 rows and 2 columns, plot location 3
axes3 = fig.add_subplot(2, 2, 3)
# subplot has 2 rows and 2 columns, plot location 4
axes4 = fig.add_subplot(2, 2, 4)
```



Matplotlib figure with four empty axes or subplot areas

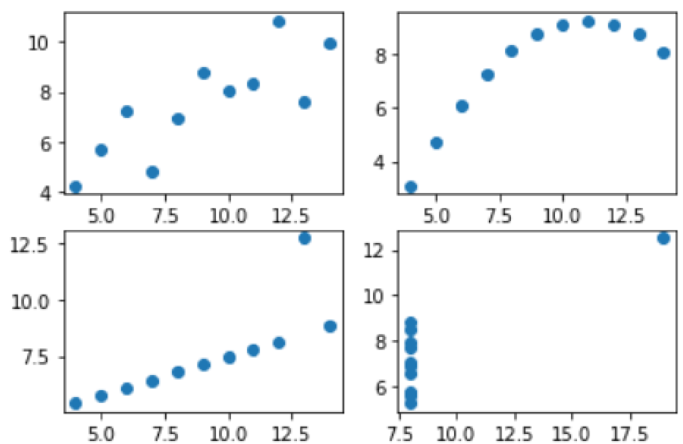
Plotting The Data

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

In [13]:
```
fig = plt.figure()
axes1 = fig.add_subplot(2, 2, 1)
axes2 = fig.add_subplot(2, 2, 2)
axes3 = fig.add_subplot(2, 2, 3)
axes4 = fig.add_subplot(2, 2, 4)

# add a plot to each of the axes created above
axes1.plot(dataset_1['x'], dataset_1['y'], 'o')
axes2.plot(dataset_2['x'], dataset_2['y'], 'o')
axes3.plot(dataset_3['x'], dataset_3['y'], 'o')
axes4.plot(dataset_4['x'], dataset_4['y'], 'o')
```

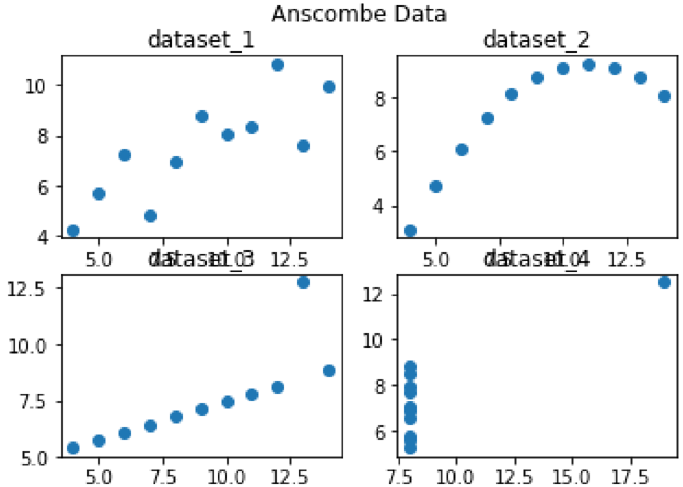
Out[13]: [<matplotlib.lines.Line2D at 0x182e898ab38>]



In [ ]:

Adding Titles

```
In [29]: fig = plt.figure()
    ...: axes1 = fig.add_subplot(2, 2, 1)
    ...: axes2 = fig.add_subplot(2, 2, 2)
    ...: axes3 = fig.add_subplot(2, 2, 3)
    ...: axes4 = fig.add_subplot(2, 2, 4)
    ...:
    ...: axes1.plot(dataset_1['x'], dataset_1['y'], 'o')
    ...: axes2.plot(dataset_2['x'], dataset_2['y'], 'o')
    ...: axes3.plot(dataset_3['x'], dataset_3['y'], 'o')
    ...: axes4.plot(dataset_4['x'], dataset_4['y'], 'o')
    ...:
    ...: # add a small title to each subplot
    ...: axes1.set_title("dataset_1")
    ...: axes2.set_title("dataset_2")
    ...: axes3.set_title("dataset_3")
    ...: axes4.set_title("dataset_4")
    ...:
    ...: # add a title for the entire figure
    ...: fig.suptitle("Anscombe Data")
    ...:
    ...:
Out[29]: Text(0.5,0.98,'Anscombe Data')
```

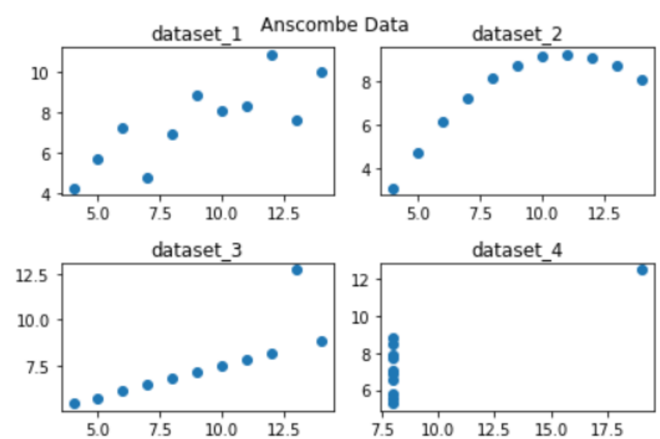



```
In [30]:
```

Adjusting Things

In [16]:
```
fig = plt.figure()
axes1 = fig.add_subplot(2, 2, 1)
axes2 = fig.add_subplot(2, 2, 2)
axes3 = fig.add_subplot(2, 2, 3)
axes4 = fig.add_subplot(2, 2, 4)

# add a plot to each of the axes created above
axes1.plot(dataset_1['x'], dataset_1['y'], 'o')
axes2.plot(dataset_2['x'], dataset_2['y'], 'o')
axes3.plot(dataset_3['x'], dataset_3['y'], 'o')
axes4.plot(dataset_4['x'], dataset_4['y'], 'o')

# add a small title to each subplot
axes1.set_title("dataset_1")
axes2.set_title("dataset_2")
axes3.set_title("dataset_3")
axes4.set_title("dataset_4")

# add a title for the entire figure
fig.suptitle("Anscombe Data")

# use a tight layout
fig.tight_layout()
```

An alternative approach is to use:
fig.subplots_adjust(hspace=0.5)



Saving a Plot to a File

- To save the plot as an image, use the code
  `plt.savefig(filename)`

- The file format can be deduced from the file name extension
- `pylab.savefig('plot.png')`
  `# save as a png image`
- `pylab.savefig('plot.pdf')`
  `# save as a pdf`
- `pylab.savefig('plot.eps')`
  `# save in Encapsulated Postscript format`

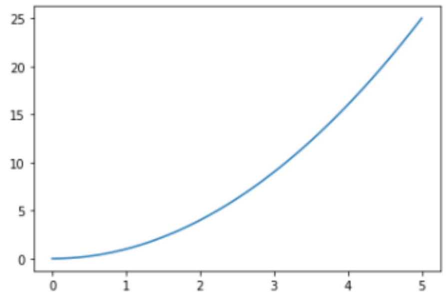
visualization - Jupyter Notebook

localhost:8888/noteboo...

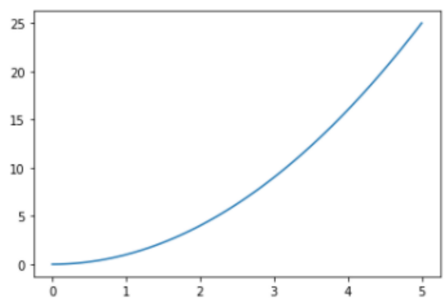
jupyter visualization

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [50]:
```python
import numpy as np
x = np.linspace(0,5,100)
y = x**2
plt.plot(x, y)
plt.savefig("./chapter1/fig2", format ="jpeg")
```



In [52]:
```python
x = np.linspace(0,5,100)
y = x**2
plt.plot(x, y)
plt.savefig("D:\\Jamil\\fig1.pdf")
```



In [ ]:

Saving the Code

- In [171]: import matplotlib.pyplot as plt
  ...
- The magic command %save followed by filename followed by number of input prompts can be used save code in .py file .py file.

- In [185]: %save my_first_chart 175

- In [186]: %save my_first_chart 171-175
  #no space and consecutive line numbers

- After you launch the command, you will find the my_first_chart.py file in your working directory

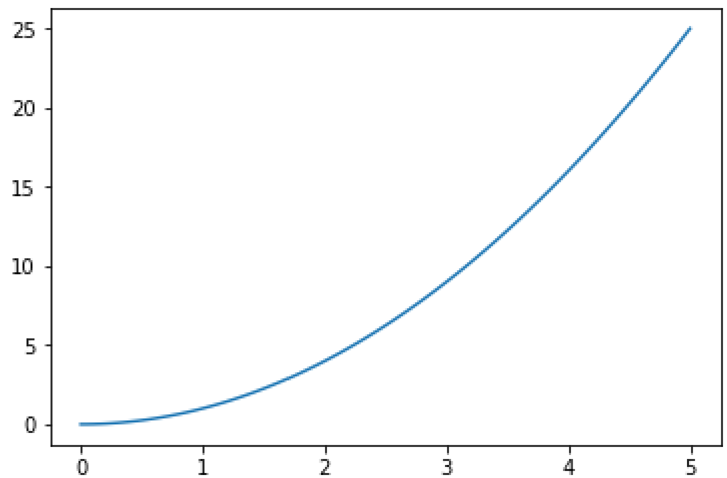
- %load my_first_chart.py   #loads the file
- %run my_first_chart.py   #runs the file

Saving the Code

Jupyter QtConsole

File Edit View Kernel Window Help

```
In [44]: import numpy as np
    ...: x = np.linspace(0,5,100)
    ...: y = x**2
    ...: plt.plot(x, y)
    ...:
    ...:
Out[44]: [<matplotlib.lines.Line2D at 0x1d366e194a8>]
```



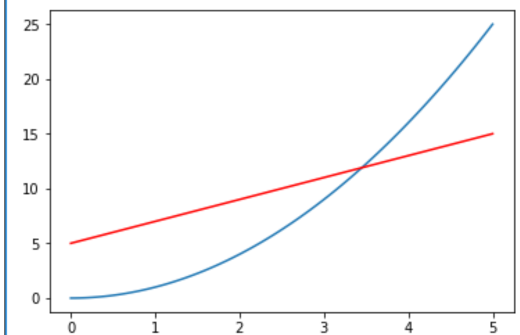

```
In [45]: %save my_first_chart 41

File `my_first_chart.py` exists. Overwrite (y/[N])?  y
The following commands were written to file `my_first_chart.py`:
get_ipython().run_line_magic('save', 'my_first_chart 40')
```

Saving the Code

Jupyter QtConsole

File Edit View Kernel Window Help

```
In [46]: import matplotlib.pyplot as plt
    ...: import numpy as np
    ...: x = np.linspace(0,5,100)
    ...:
    ...:

In [47]: y = x**2
    ...: z = 2*x + 5
    ...:
    ...:

In [48]: plt.plot(x, y)
    ...: plt.plot(x, z, color = "red")
    ...:
    ...:
Out[48]: [<matplotlib.lines.Line2D at 0x1d366de1dd8>]
```



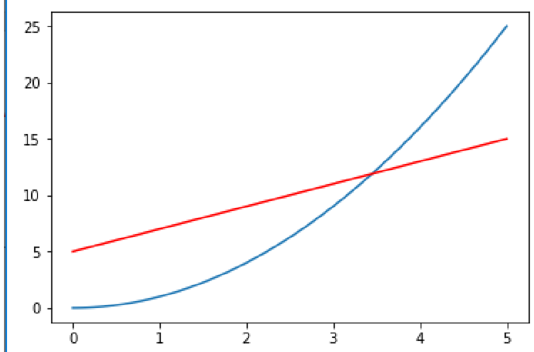

```
In [49]: %save myFirstChart 46-48
The following commands were written to file `myFirstChart.py`:
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0,5,100)
y = x**2
z = 2*x + 5
plt.plot(x, y)
plt.plot(x, z, color = "red")

In [50]:
```

Loading and Running the File

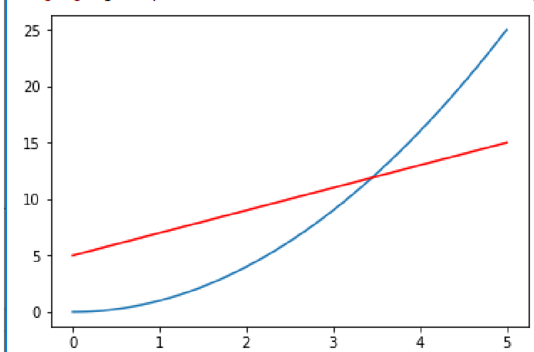
Jupyter QtConsole

File Edit View Kernel Window Help

```
In [55]: %run myFirstChart.py
```




```
In [56]: %load myFirstChart.py

In [57]: # %load myFirstChart.py
    ...: import matplotlib.pyplot as plt
    ...: import numpy as np
    ...: x = np.linspace(0,5,100)
    ...: y = x**2
    ...: z = 2*x + 5
    ...: plt.plot(x, y)
    ...: plt.plot(x, z, color = "red")
    ...:
    ...:
Out[57]: [<matplotlib.lines.Line2D at 0x1d366f94278>]
```




```
In [58]:
```

Good resources

- https://matplotlib.org/
- https://www.w3schools.com/python/
- https://www.geeksforgeeks.org/python-programming-language/
- Jupyter Notebook: https://www.dataquest.io/blog/jupyter-notebook-tutorial/
- NumPy: https://www.machinelearningplus.com/python/numpy-tutorial-part1-array-python-examples/
- Data Science: https://www.datacamp.com/courses/tech:python