04 - Python_Matplotlib_Course_Notes

November 25, 2021

1 Visualization with Matplotlib

Course notes by - Himalaya Kakshapati

Matplotlib is a multiplatform data visualization library built on NumPy arrays.

2 Import matplotlib

We can import matplotlib in the following way.

```
[ ]: import matplotlib as mpl
     import matplotlib.pyplot as plt
```

The `plt` interface is used the most often.

3 Setting styles

We can use `plt.style()` to choose appropriate styles for our figures.

Let's use the `classic` style, so that the plot we create uses classic Matplotlib style.

```
[ ]: plt.style.use('classic')
```

4 show() or No show()? How to display your plots

There are 3 applicable contexts of using Matplotlib. They are, using Matplotlib in:

- script
- IPython terminal
- IPython notebook

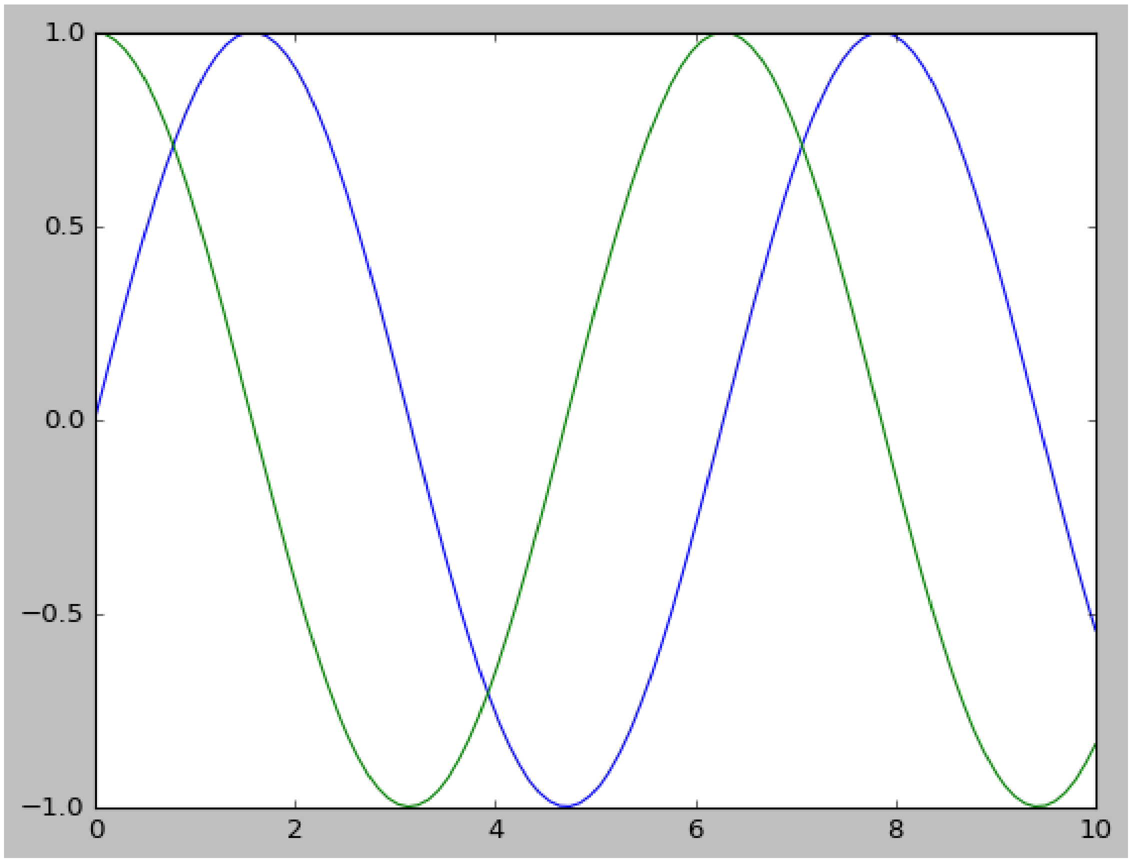
4.1 Plotting from a script

If you are using Matplotlib in a script (file with .py extension), you need to use `plt.show()` to display your graph.

`plt.show()` starts an event loop, looks for all currently active figure objects, and opens one or more interactive windows that display your figure/s.

For example, if you have a file named 'myplot.py', you can use Matplotlib as follows.

[ ]:
```
 # ---- file: myplot.py -------
 import matplotlib.pyplot as plt
 import numpy as np

x = np.linspace(0, 10, 100)

plt.plot(x, np.sin(x))
plt.plot(x, np.cos(x))

plt.show()
```



Then, you can run the script from the command line as follows.

```
$ python myplot.py
```

When the command is executed, you will see a plot as shown above.

The `plt.show()` command does a lot under the hood, as it interacts with your system's graphical backend.

Please note that `plt.show()` command can be used *only once* per Python session. So it is most often put at the very end of the script.

Multiple invokation of `show()` should be avoided.

4.2 Plotting from IPython shell

In Ipython shell, you can enable Matplotlib mode by using the `%matplotlib` magic command.

```
[ ]: %matplotlib
```

```
Using matplotlib backend: agg
```

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

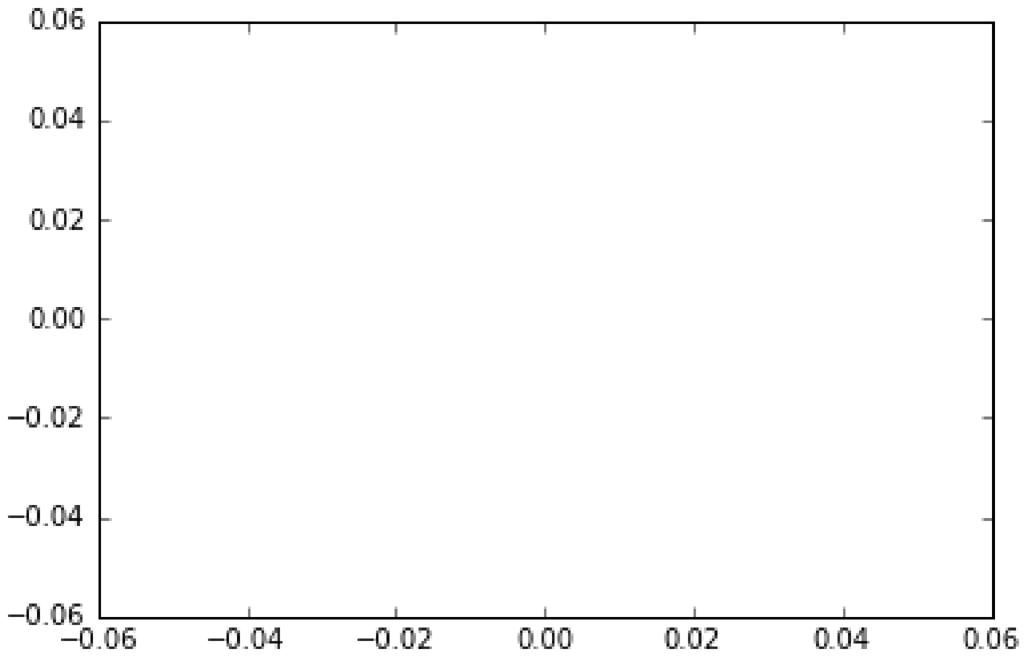
At this point, any `plt` plot command causes a figure window to open, and further commands can be run to update the plot.

Some changes, such as modifying properties of lines that are already drawn, will not draw automatically. So in order to force an update, you can use `plt.draw()` command.

`plot.show()` is not required in Matplotlib mode.

```
[ ]: plt.plot()
```

```
[ ]: []
```



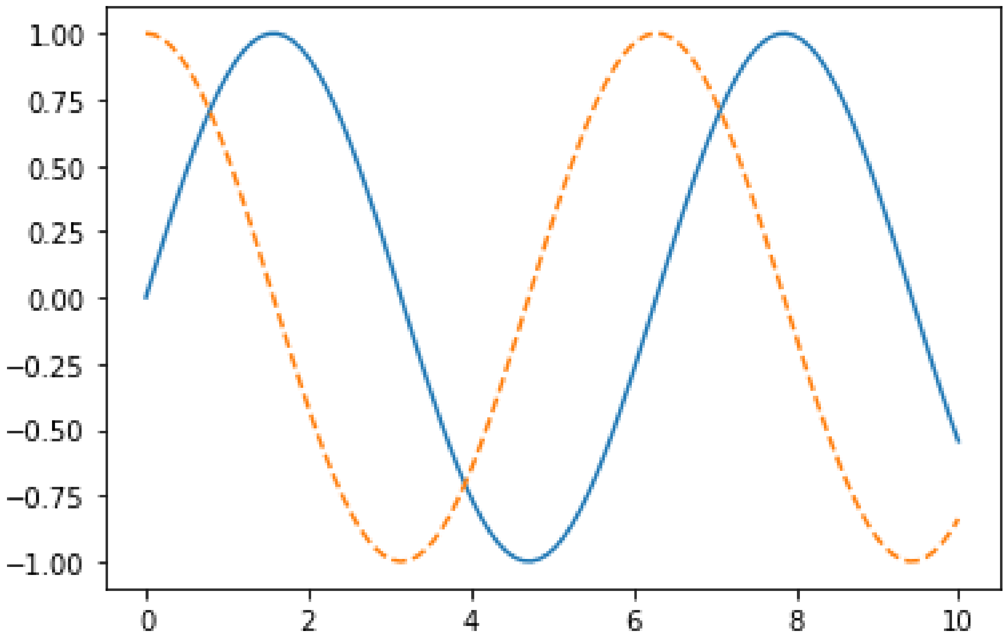
4.3 Plotting from an IPython notebook

Plotting interactively within an IPython notebook can be done with the `%matplotlib inline` command, and works in similar way to the IPython shell.

[ ]:
```
%matplotlib inline
```

After we run the above command (it needs to be done only once per kernel/session), any cell within the notebook that creates a plot will embed a PNG image of the resulting graph.

[ ]:
```
import numpy as np

x = np.linspace(0, 10, 100)

fig = plt.figure()

plt.plot(x, np.sin(x), '-')
plt.plot(x, np.cos(x), '--');
```



5 Two interfaces for the price of one

Matplotlib has dual interfaces: * Matlab-style state-based interface * object-oriented interface

Matlab-style interface is convenient, and object-oriented interface is more powerful.

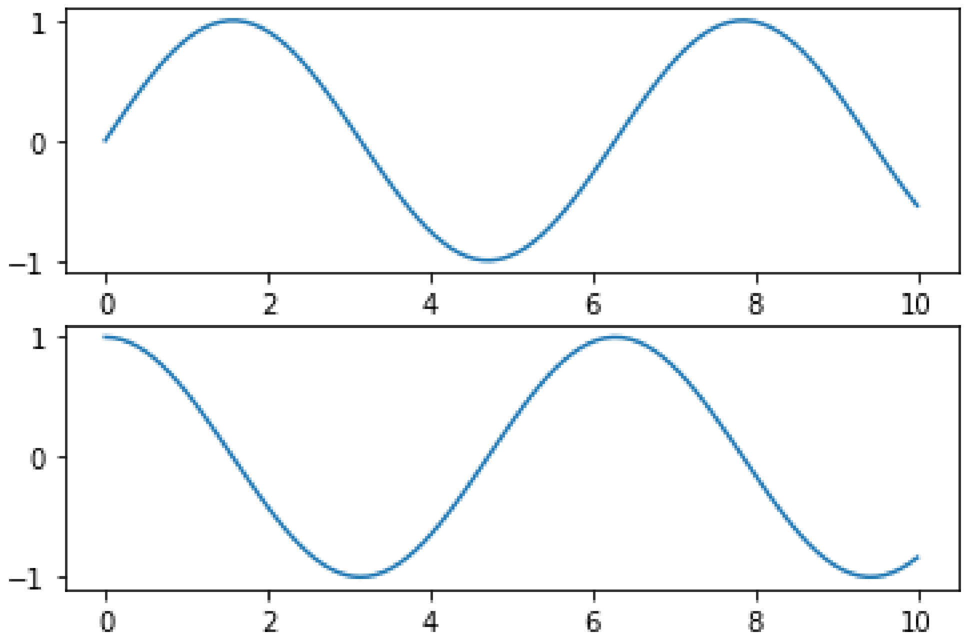
5.1 Matlab-style interface

Matplotlib was originally written as a Python alternative for Matlab users, and much of its syntax reflects that fact.

The Matlab-style tools are contained in the pyplot (`plt`) interface.

```
[ ]: plt.figure() # create a plot figure

     # create the first of two panels and set the current axis
     plt.subplot(2, 1, 1) # (rows, cols, panel number)
     plt.plot(x, np.sin(x))

     # create the second panel and set the current axis
     plt.subplot(2, 1, 2)
     plt.plot(x, np.cos(x))
```

```
[ ]: [<matplotlib.lines.Line2D at 0x7f25fa1918d0>]
```



This stateful interface is fast and convenient for simple plots, it can run into problems for more complex graphs.

For example, once the second panel is created, how can we go back and add something to the first?

There is a better way.

5.2 Object-oriented interface

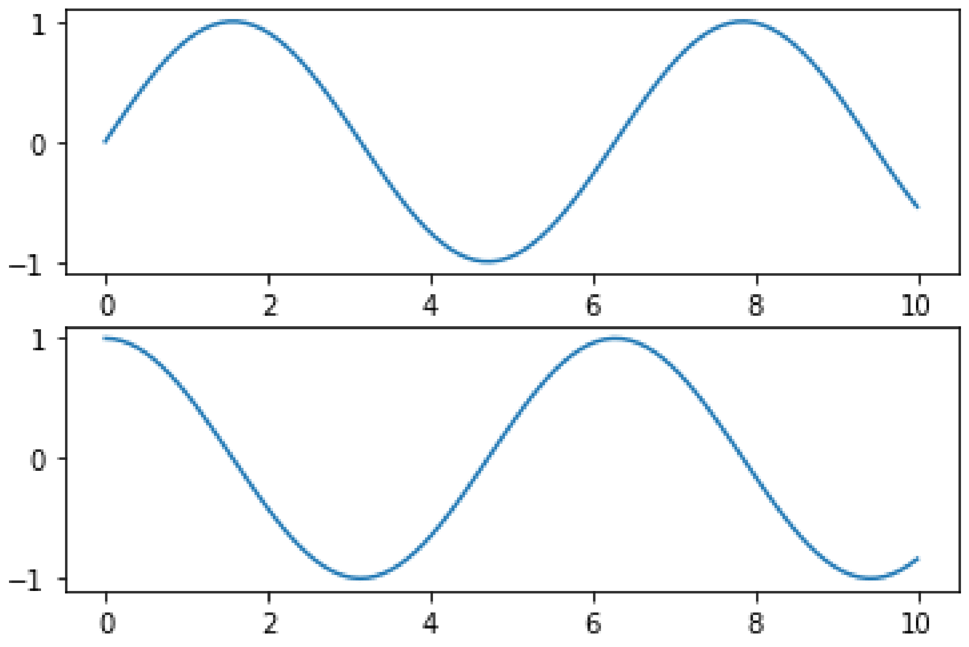
The object-oriented interface is available for more complicated situations, and for when you want more control over your figure.

In the object-oriented interface the plotting functions are *methods* of explicit `Figure` and `Axes` objects.

Let's recreate the previous graph with object-oriented interface.

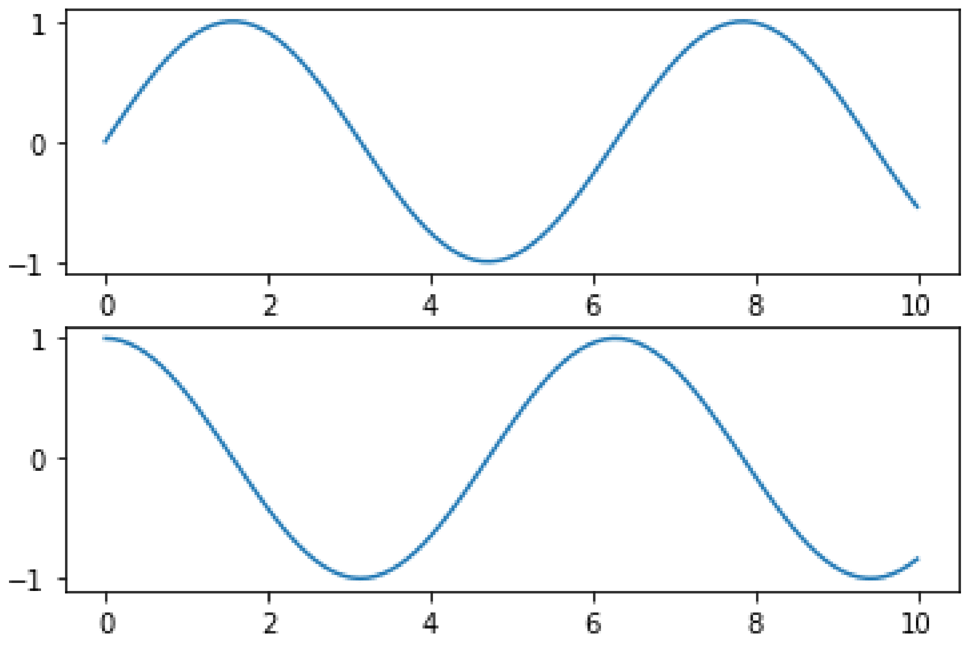
```
[ ]: # First create a grid of plots
     # ax will be an array of two Axes objects
     fig, ax = plt.subplots(2)

     # Call plot() method on the appropriate object
     ax[0].plot(x, np.sin(x))
     ax[1].plot(x, np.cos(x));
     ax
```

```
[ ]: array([<matplotlib.axes._subplots.AxesSubplot object at 0x7f25fa06f090>,
            <matplotlib.axes._subplots.AxesSubplot object at 0x7f25fa011190>],
           dtype=object)
```



```
[ ]: fig
```

```
[ ]:
```



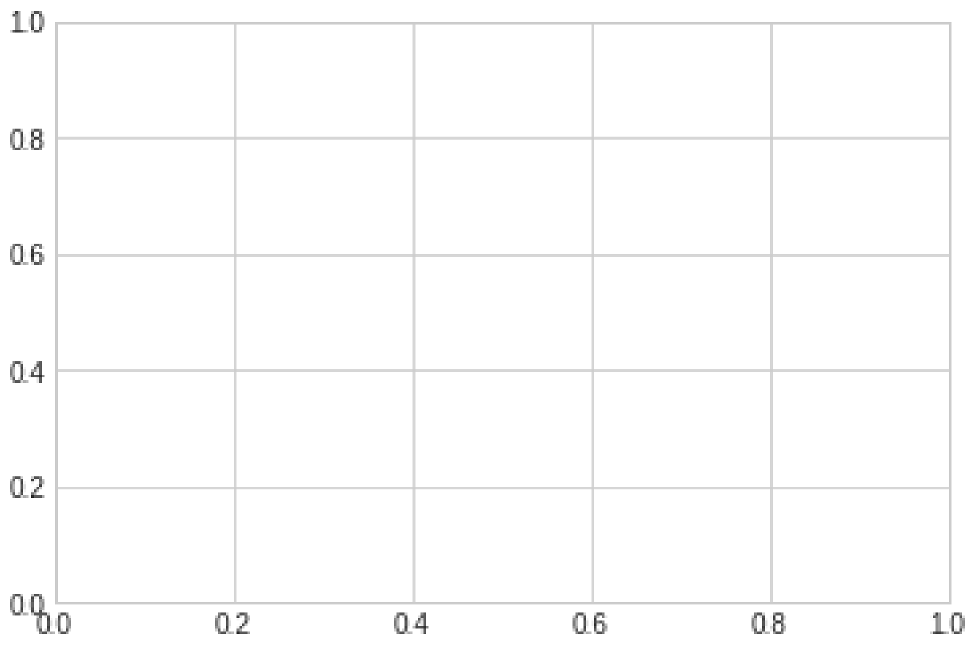
6 Simple Line Plots

[ ]:
```
%matplotlib inline
import matplotlib.pyplot as plt

plt.style.use('seaborn-whitegrid')
import numpy as np
```

For all Matplotlib plots, we start by creating a figure and axes.

Figure and axes can be created as follows.

[ ]:
```
fig = plt.figure()
ax = plt.axes()
```

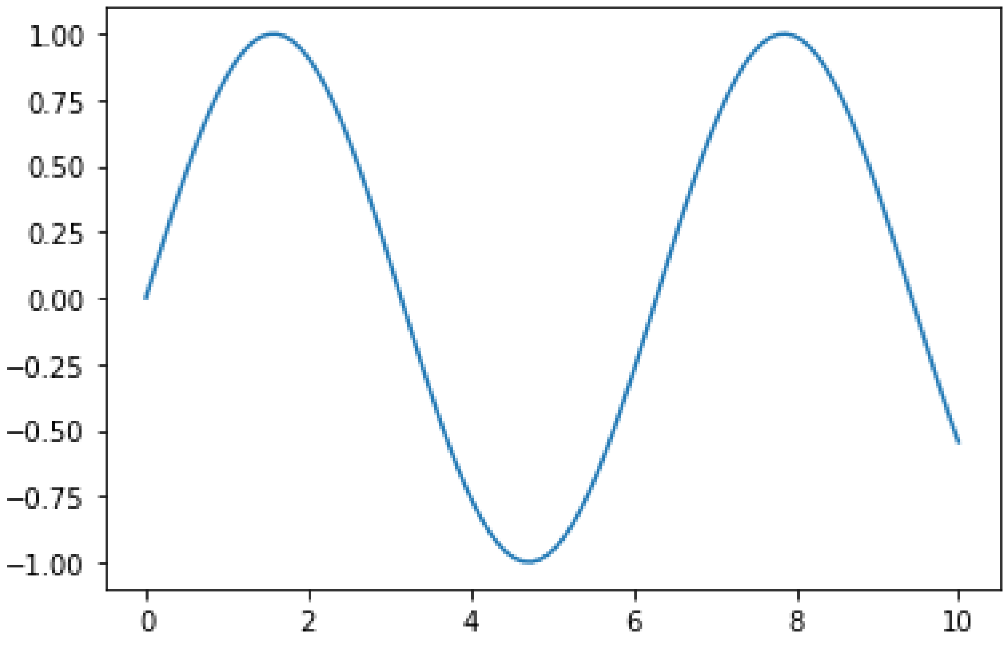


In Matplotlib, the *figure* (which is an instance of `plt.Figure`) can be taken as a single container that contains all the objects representing axes, graphics, text, and labels.

Once we have created the axes, we can use the `ax.plot()` function to plot some data.

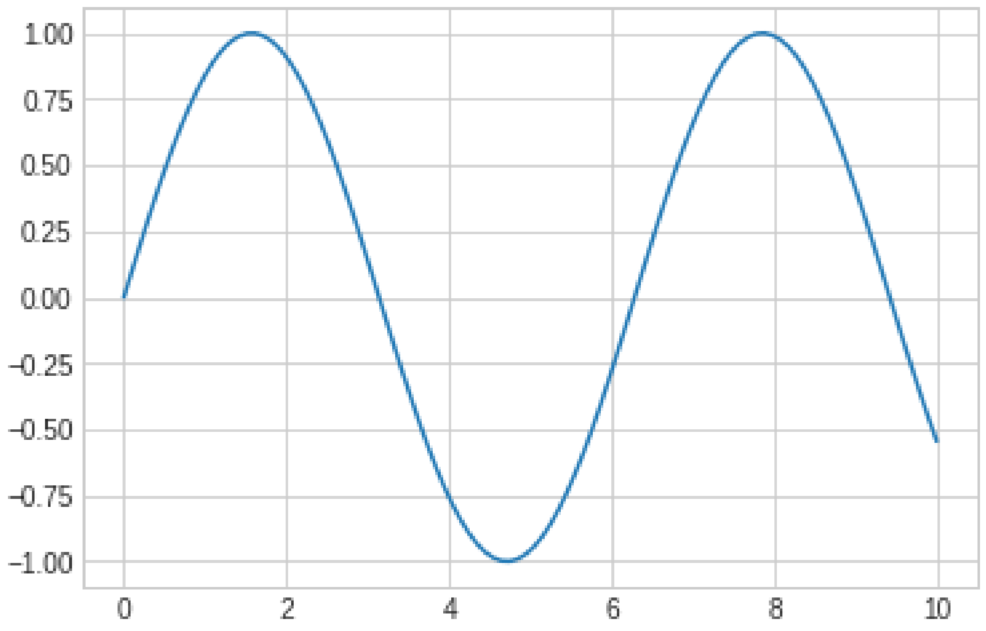
Example:

[ ]:
```
fig = plt.figure()
ax = plt.axes()

# plot a sinusoidal curve
x = np.linspace(0, 10, 100)
ax.plot(x, np.sin(x));
```



Alternatively, we can use the pylab interface and let the figure and axes be created for us in the background.

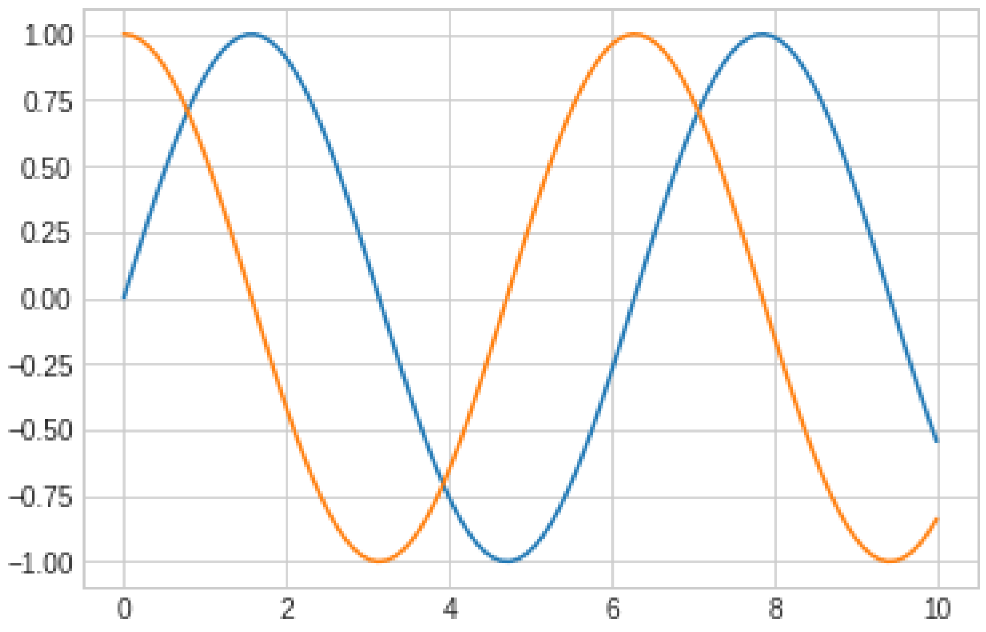
```
[ ]: plt.plot(x, np.sin(x));
```



If we want to create a single figure with multiple curves, we can call `plot()` function multiple times.

```
[ ]: # two curves on the same figure
     plt.plot(x, np.sin(x)) # sin
     plt.plot(x, np.cos(x)) # cos
```

```
[ ]: [<matplotlib.lines.Line2D at 0x7f416d8ffd90>]
```



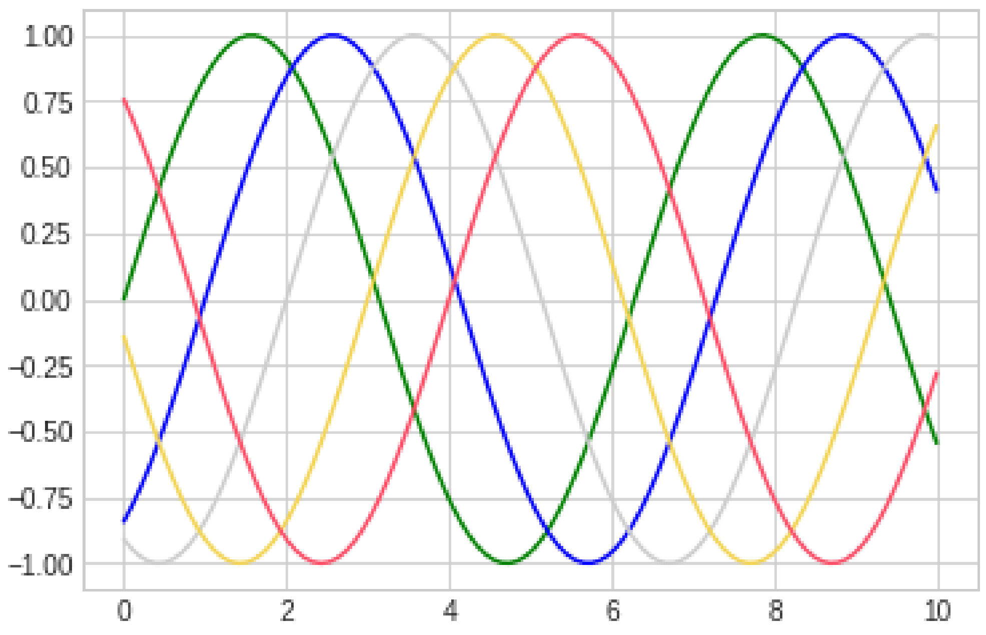
6.1 Adjusting the Plot: Line Colors and Styles

We can choose line colors and styles by using the additional arguments in `plt.plot()`.

To adjust the color, we can use the `color` keyword, which accepts a string argument representing virtually any color.

```
[ ]: plt.plot(x, np.sin(x - 0), color='green') # color by name
     plt.plot(x, np.sin(x - 1), color='b') # short color code (rgbcmyk)
     plt.plot(x, np.sin(x - 2), color='0.8') # grayscale (between 0 and 1)
     plt.plot(x, np.sin(x - 3), color='#F2D354') # Hex code (RRGGBB from 00 to FF)
     plt.plot(x, np.sin(x - 4), color=(1.0, 0.3, 0.4)) # RGB tuple, values between 0␣
      ↪and 1
```

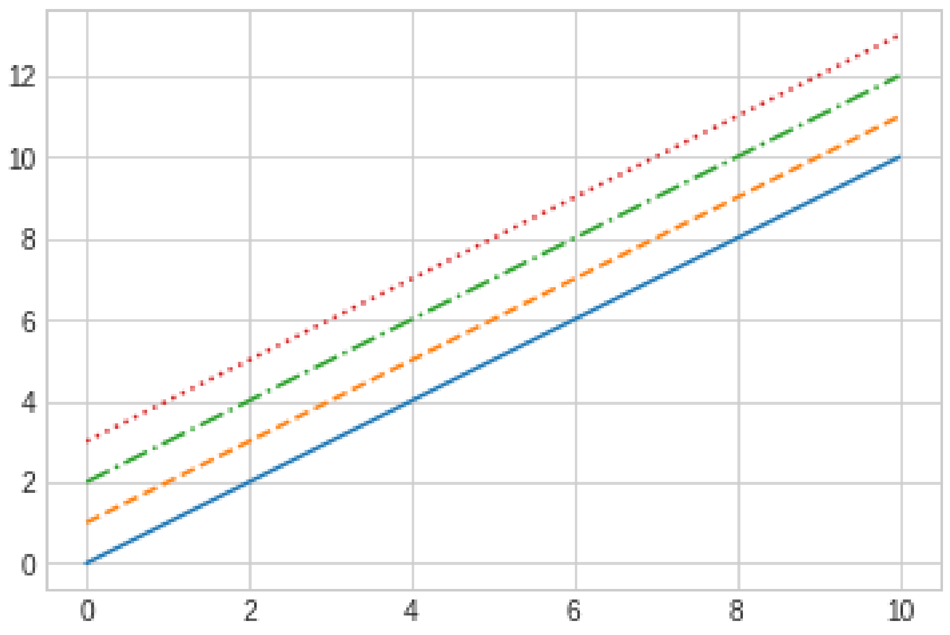
```
[ ]: [<matplotlib.lines.Line2D at 0x7f416d7b1850>]
```



We can adjust the line style using the `linestyle` keyword.

```
[ ]: plt.plot(x, x + 0, linestyle='solid')
     plt.plot(x, x + 1, linestyle='dashed')
     plt.plot(x, x + 2, linestyle='dashdot')
     plt.plot(x, x + 3, linestyle='dotted');
```

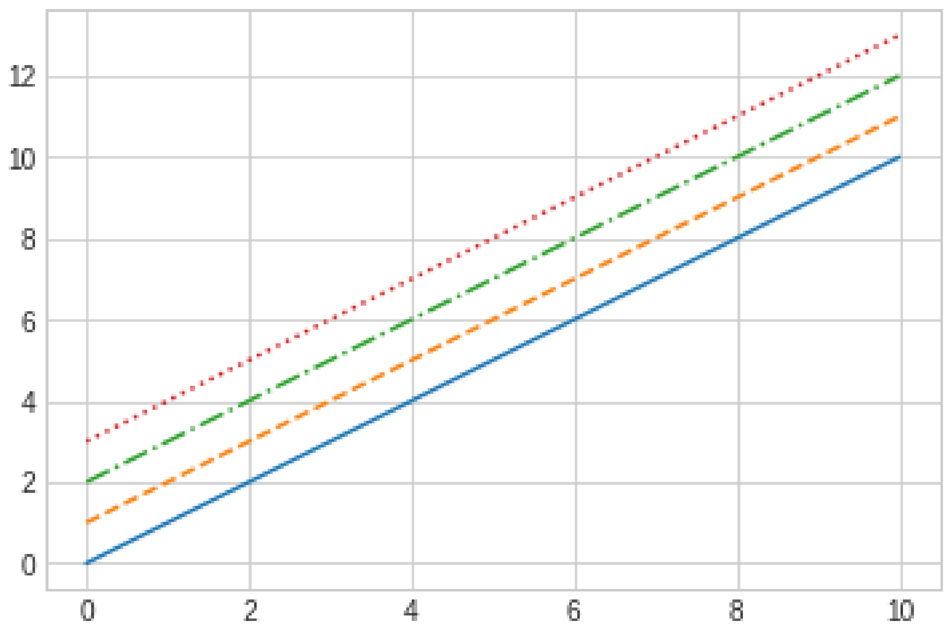
```
[ ]: [<matplotlib.lines.Line2D at 0x7f416dba4c90>]
```



We can also use the short codes as follows.

```
[ ]: plt.plot(x, x + 0, linestyle='-') # solid
     plt.plot(x, x + 1, linestyle='--') # dashed
     plt.plot(x, x + 2, linestyle='-.') # dashed dot
     plt.plot(x, x + 3, linestyle=':'); # dotted
```

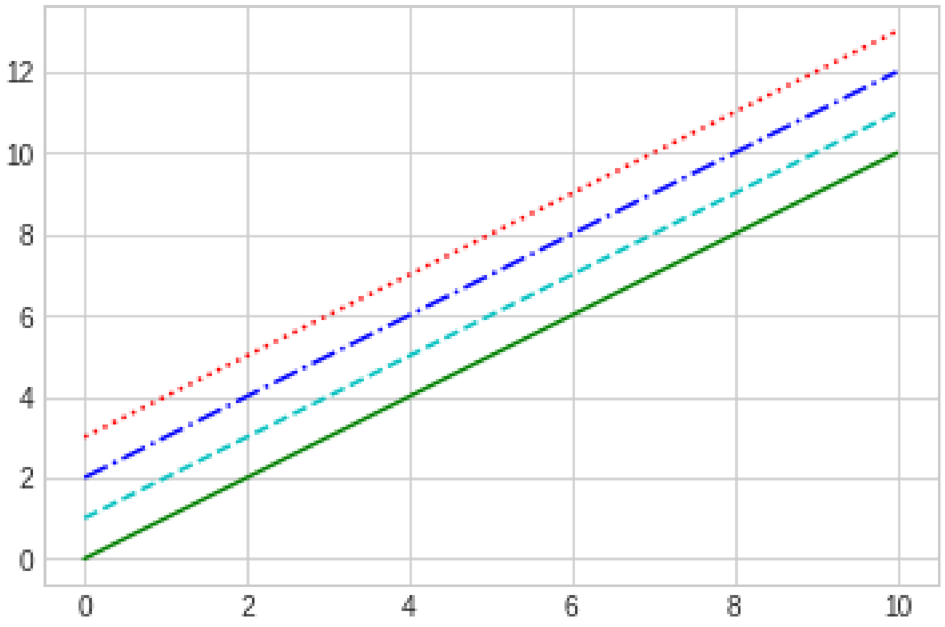
```
[ ]: [<matplotlib.lines.Line2D at 0x7f416d98a950>]
```



The `linestyle` and `color` codes can be combined.

```
[ ]: # rgb => (red, green, blue); cmyk => (cyan, magenta, yellow, black)
     plt.plot(x, x + 0, '-g') # solid green
     plt.plot(x, x + 1, '--c') # dashed cyan
     plt.plot(x, x + 2, '-.b') # dashed dot blue
     plt.plot(x, x + 3, ':r') # dotted red
```

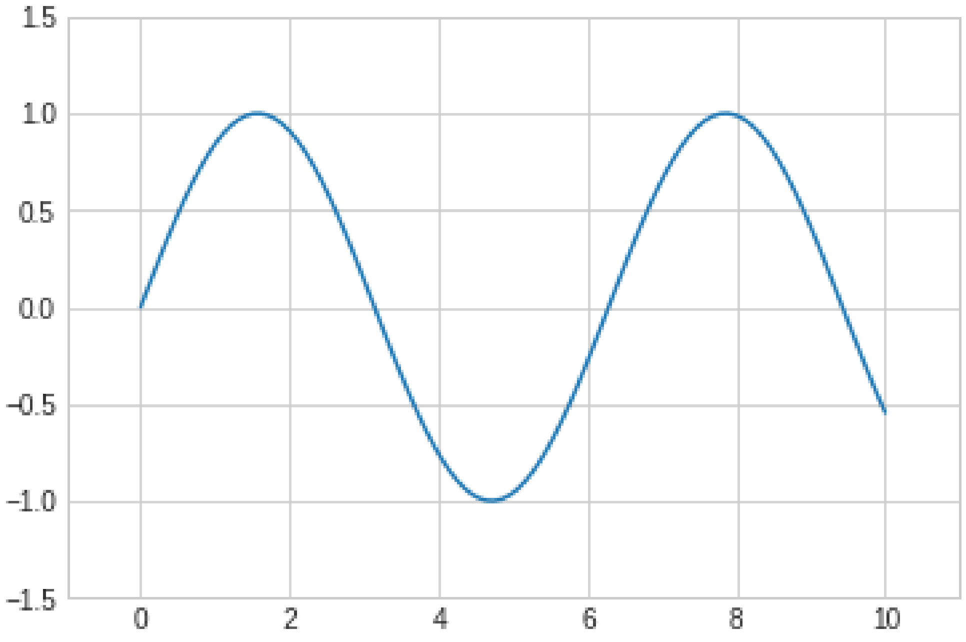
```
[ ]: [<matplotlib.lines.Line2D at 0x7f416d5c73d0>]
```



6.2 Adjusting the Plot: Axes Limits

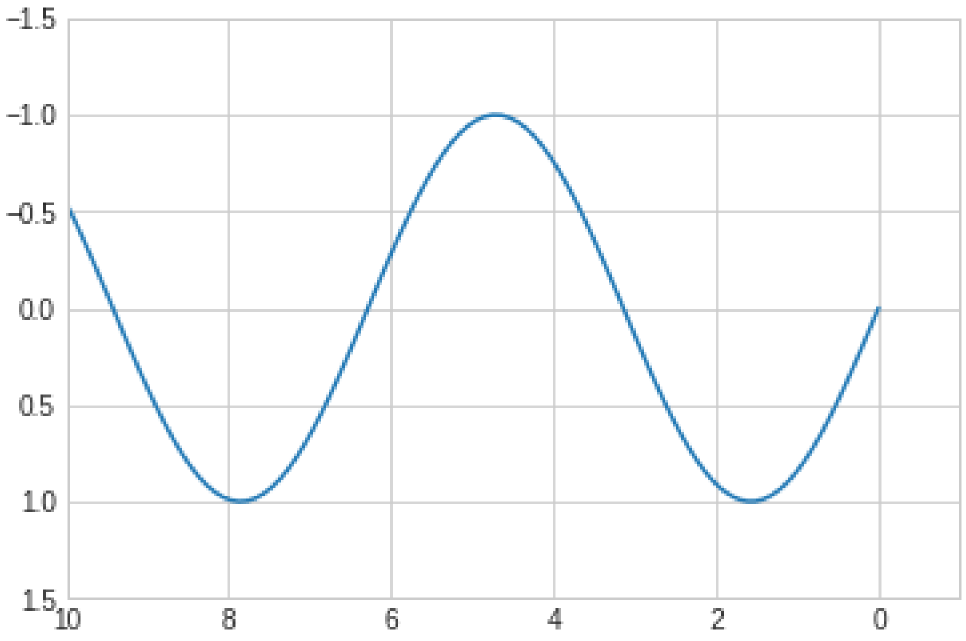
We can adjust the axis limits using `plt.xlim()` and `plt.ylim()` methods.

```
[ ]: %matplotlib inline
     import matplotlib.pyplot as plt
     import numpy as np

     plt.style.use('seaborn-whitegrid')
     plt.plot(x, np.sin(x))

     plt.xlim(-1, 11) # x axis from -1 to 11
     plt.ylim(-1.5, 1.5); # axis from -1.5 to 1.5
```



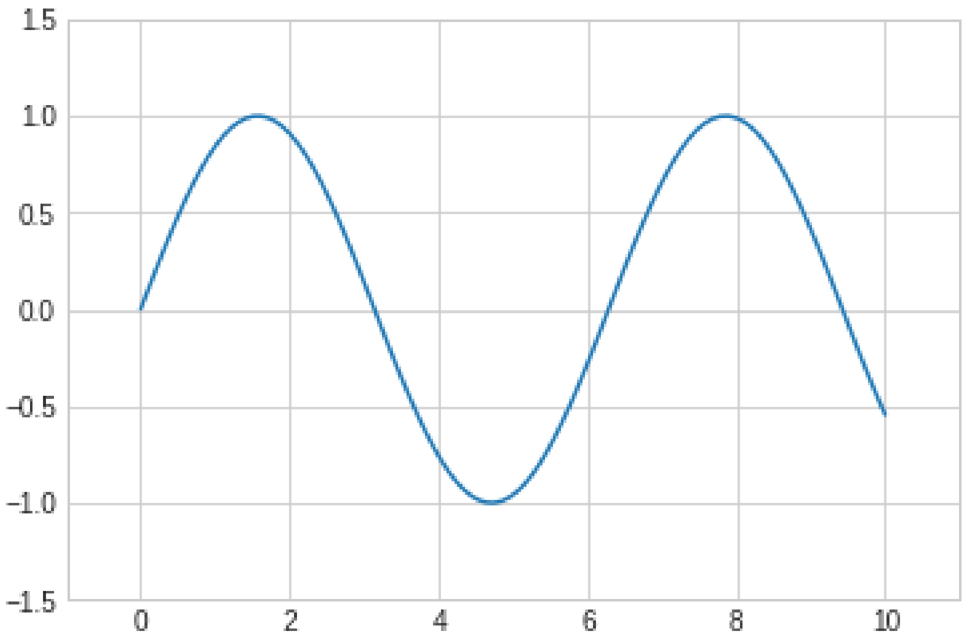
If you want to display either axis to be displayed in reverse, you can reverse the order of arguments.

[ ]:
```
plt.plot(x, np.sin(x))

plt.xlim(10, -1) # x axis from 10 to -1
plt.ylim(1.5, -1.5); # axis from 1.5 to -1.5
```



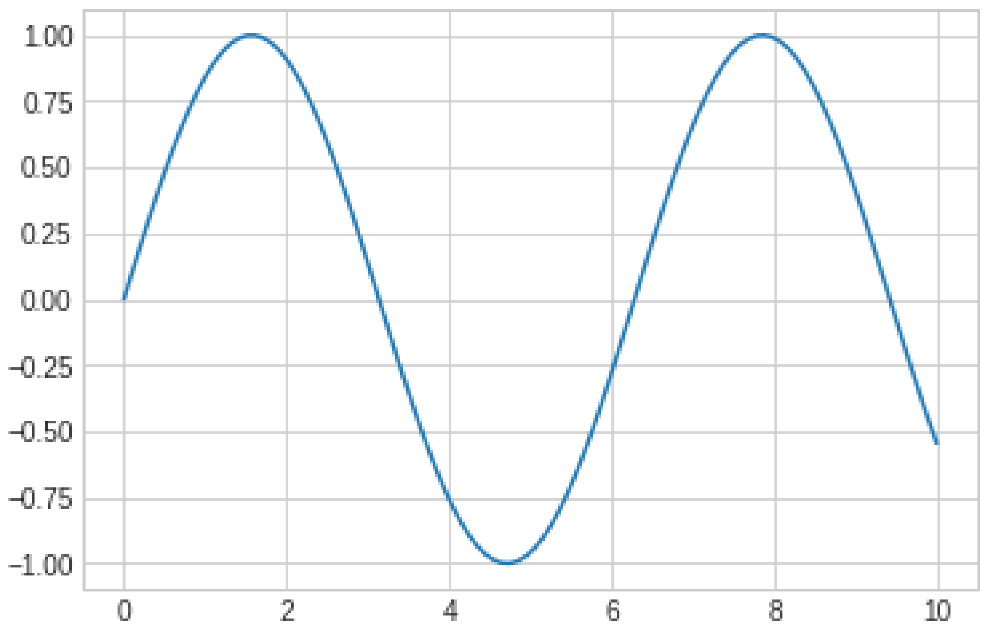
The `plt.axis()` method allows you to set the x and y limits with a single call, passing a list that specifies `[xmin, xmax, ymin, ymax]`.

```
[ ]: plt.plot(x, np.sin(x))
     plt.axis([-1, 11, -1.5, 1.5]);
```



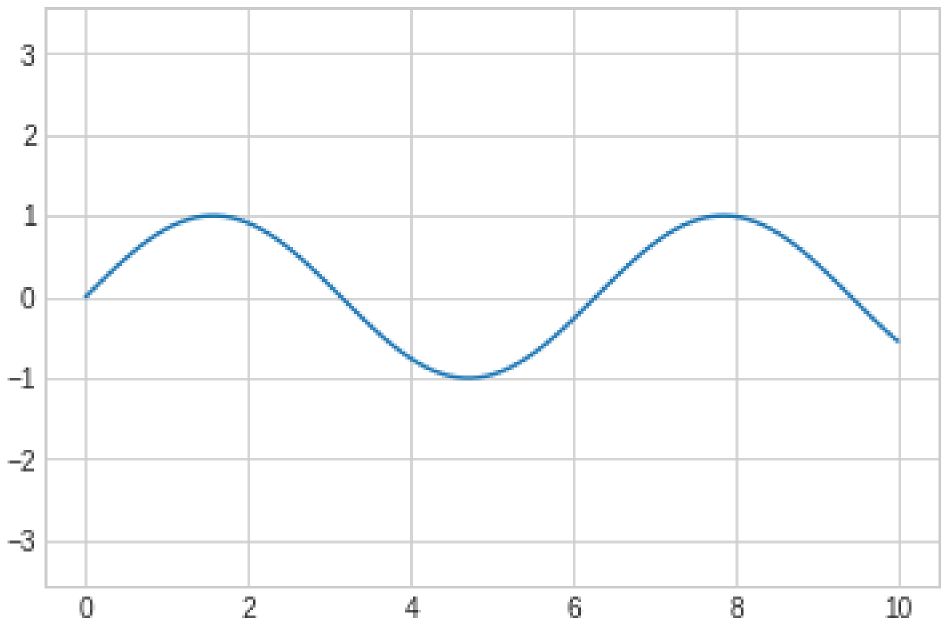
You can automatically tighten the bounds around the current plot as follows.

```
[ ]: plt.plot(x, np.sin(x))
     plt.axis('tight');
```



We can ensure an equal aspect ratio.

```
[ ]: plt.plot(x, np.sin(x))
     plt.axis('equal');
```

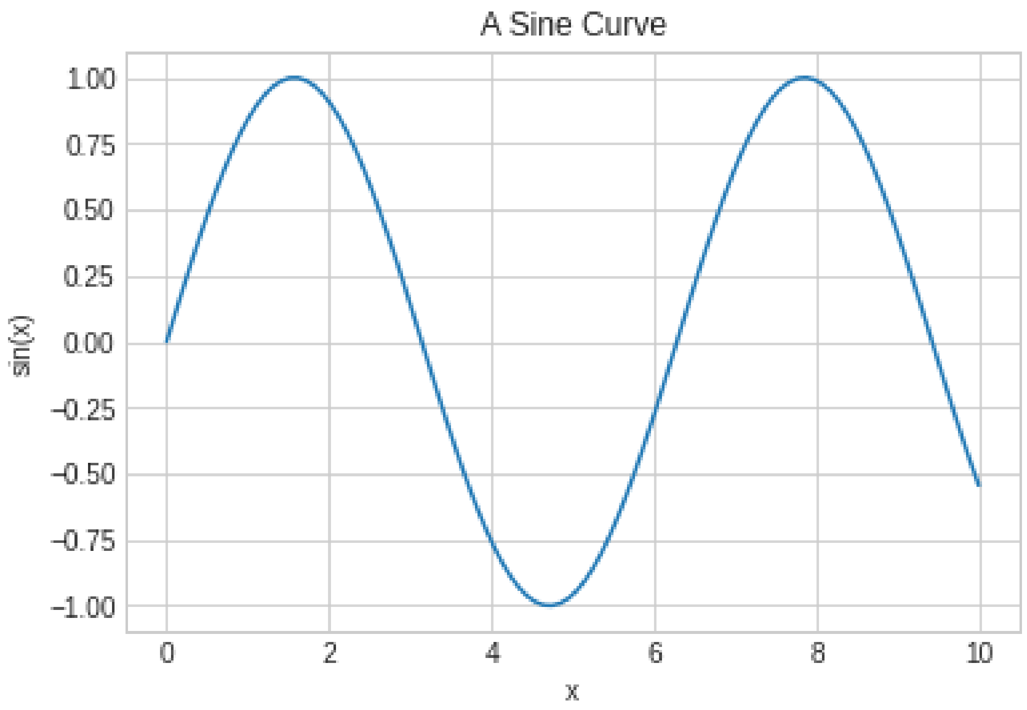


6.3 Labeling Plots

Let's look at how we can label plots: titles, axis labels, and simple legends.

```
[ ]: plt.plot(x, np.sin(x))
     plt.title("A Sine Curve"); # setting the title of the graph'
     plt.xlabel("x") # x label
     plt.ylabel("sin(x)") # y label
```

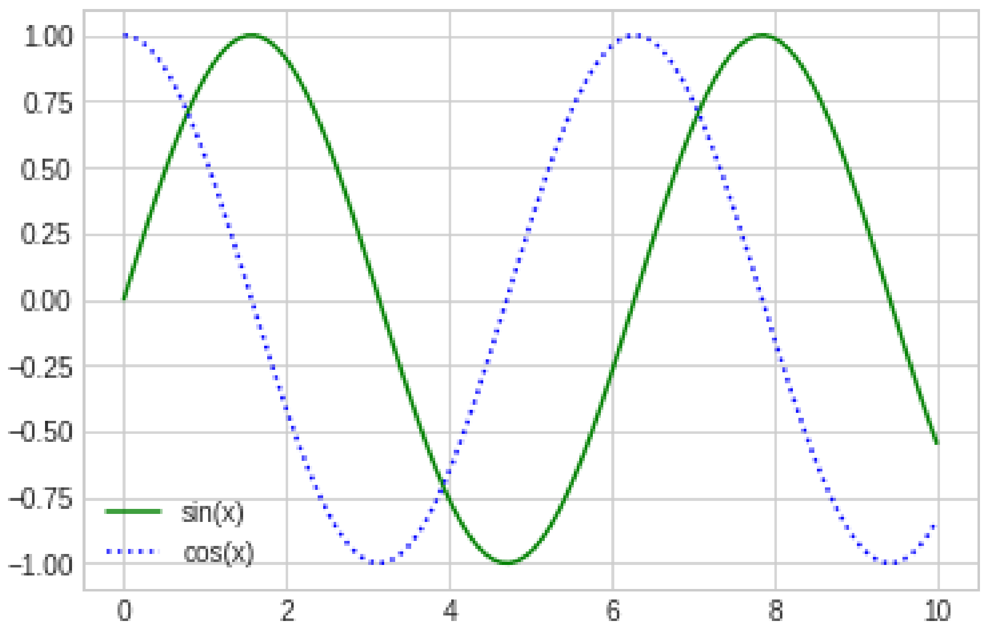
```
[ ]: Text(0, 0.5, 'sin(x)')
```



We can create legends using the `plt.legend()` method.

```
[ ]: plt.plot(x, np.sin(x), '-g', label='sin(x)')
     plt.plot(x, np.cos(x), ':b', label='cos(x)')

     plt.legend() # shows the legend
```

```
[ ]: <matplotlib.legend.Legend at 0x7f89e4b0c210>
```



7 Simple Scatter Plots

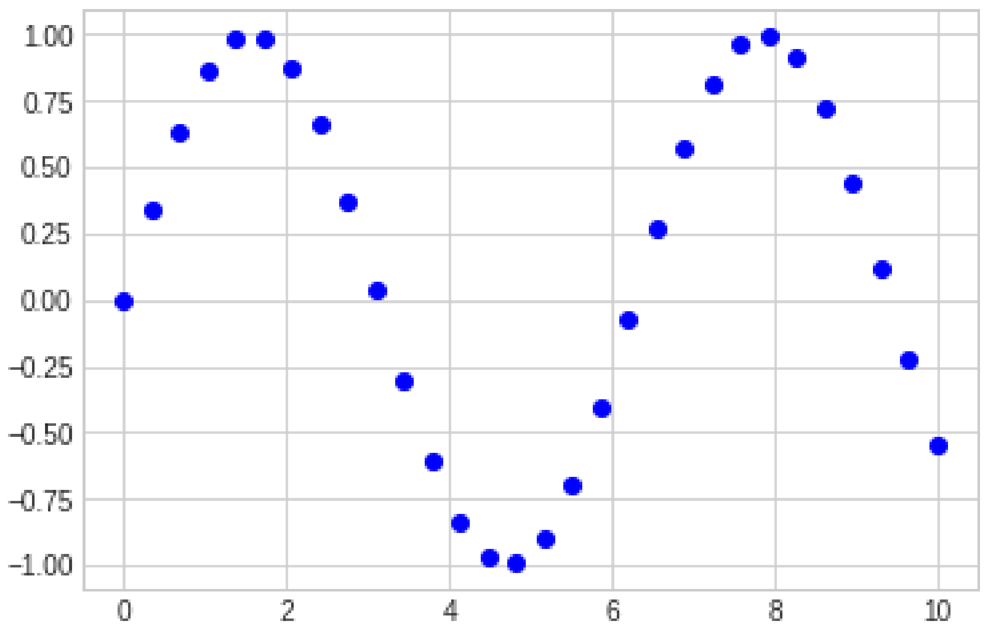
Another commonly used plot type is the simple scatter plot.

```
[ ]: %matplotlib inline
     import matplotlib.pyplot as plt
     plt.style.use('seaborn-whitegrid')
     import numpy as np
```

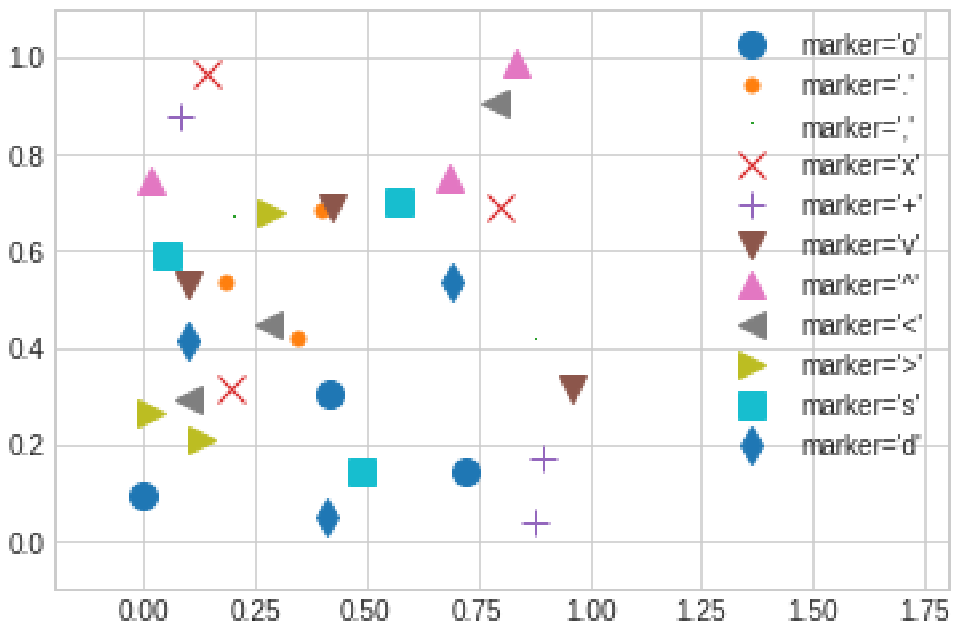
7.1 Scatter Plots with `plt.plot`

We can use `plt.plot()` method to plot scatter plots.

```
[ ]: x = np.linspace(0, 10, 30)
     y = np.sin(x)

     plt.plot(x, y, 'o', color='blue');
```

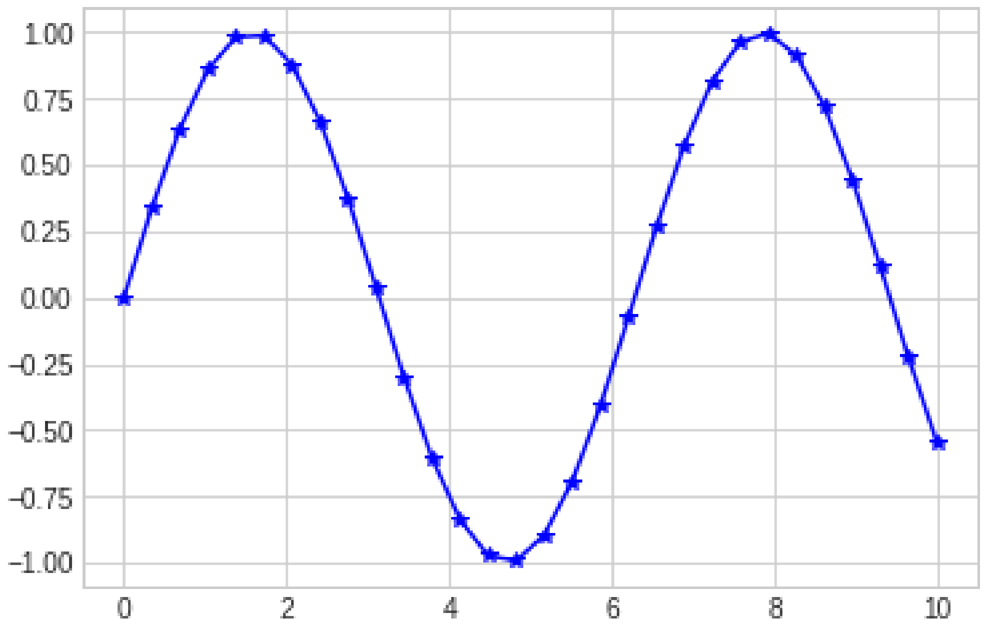


```
[ ]: rng = np.random.RandomState(1)

     for marker in ['o', '.', ',', 'x', '+', 'v', '^', '<', '>', 's', 'd']:
       plt.plot(rng.rand(3), rng.rand(3), marker, label="marker='{0}'".
      ↪format(marker), markersize=10)

     plt.legend()
     plt.xlim(-0.2, 1.8)
     plt.ylim(-0.1, 1.1);
```



These character codes can be used together with line and color codes to plot points along with the line connecting them.

```
[ ]: plt.plot(x, y, '-*b');
```

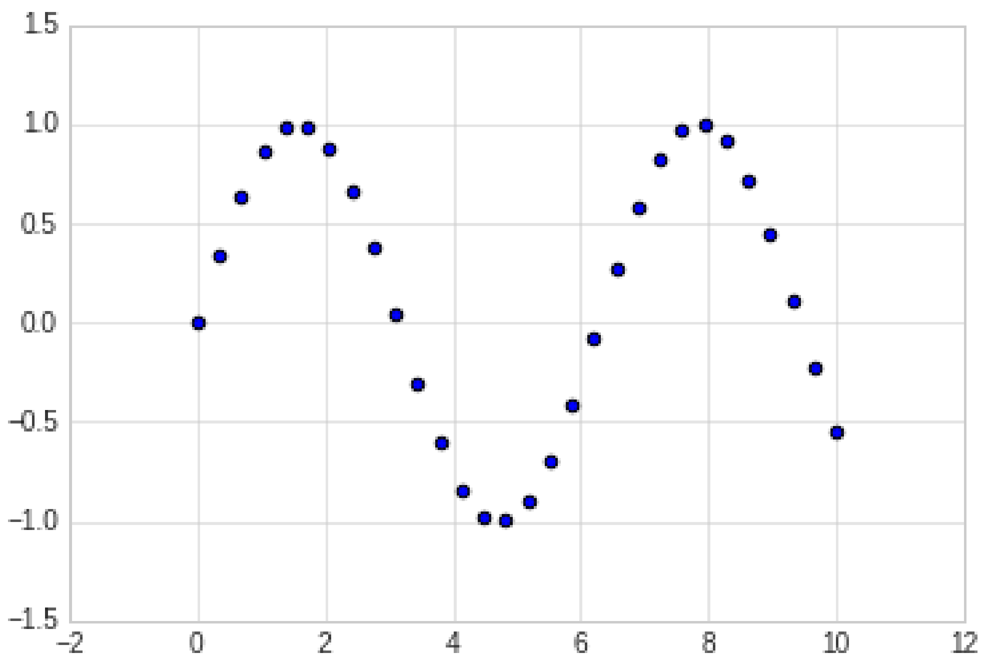


7.2 Scatter Plots with `plt.scatter`

A second, more powerful method of creating scatter plots is the `plt.scatter()` function, which can be used just like `plt.plot()`.

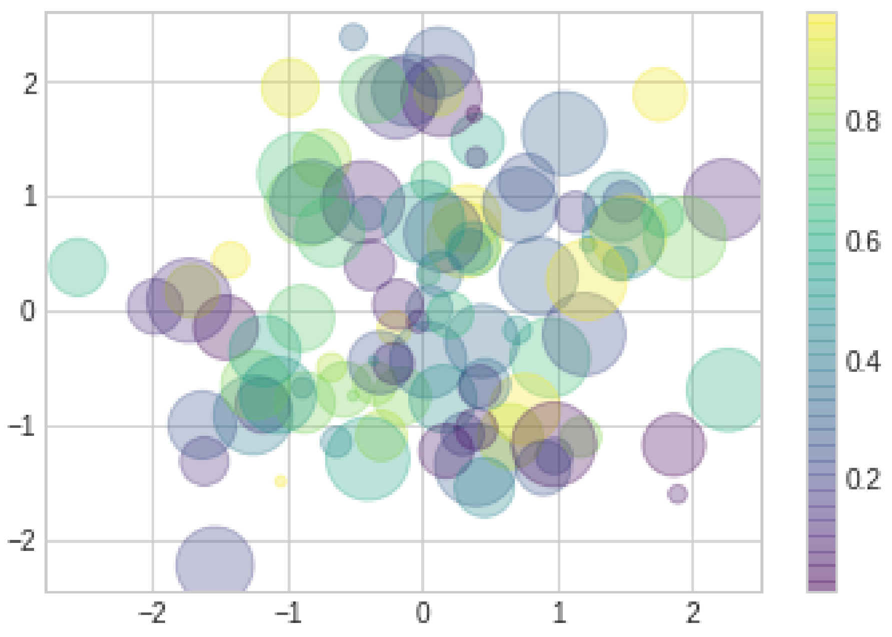
```
[ ]: plt.scatter(x, y, marker='o')
```

```
[ ]: <matplotlib.collections.PathCollection at 0x7f3d80c31a50>
```



The main difference between `plt.scatter` and `plt.plot` is that `plt.scatter` can be used to create scatter plots where the properties of each indiviual points (size, face color, edge color etc.) can be individually controlled or mapped to data.

```
[ ]: import numpy as np
     rng = np.random.RandomState(0)
     x = rng.randn(100)
     y = rng.randn(100)

     colors = rng.rand(100)
     sizes = 1000 * rng.rand(100)

     plt.scatter(x, y, c=colors, s=sizes, alpha=0.3, cmap='viridis')

     plt.colorbar(); # show color scale
```



Note that the `color` argument is automatically mapped to the color scale, and the size argument is given in pixels.

In this way, the color and size of points can be used to convey information in visualization in order to illustrate multi-dimensional data.

Let's look at an example of Iris dataset from scikit-learn.

```
[ ]: from sklearn.datasets import load_iris
     iris = load_iris()
     print(iris)
```

```
{'data': array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
       [5.1, 3.8, 1.5, 0.3],
       [5.4, 3.4, 1.7, 0.2],
       [5.1, 3.7, 1.5, 0.4],
       [4.6, 3.6, 1. , 0.2],
       [5.1, 3.3, 1.7, 0.5],
       [4.8, 3.4, 1.9, 0.2],
       [5. , 3. , 1.6, 0.2],
       [5. , 3.4, 1.6, 0.4],
       [5.2, 3.5, 1.5, 0.2],
       [5.2, 3.4, 1.4, 0.2],
       [4.7, 3.2, 1.6, 0.2],
       [4.8, 3.1, 1.6, 0.2],
       [5.4, 3.4, 1.5, 0.4],
       [5.2, 4.1, 1.5, 0.1],
       [5.5, 4.2, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.2],
       [5. , 3.2, 1.2, 0.2],
       [5.5, 3.5, 1.3, 0.2],
       [4.9, 3.6, 1.4, 0.1],
       [4.4, 3. , 1.3, 0.2],
       [5.1, 3.4, 1.5, 0.2],
       [5. , 3.5, 1.3, 0.3],
       [4.5, 2.3, 1.3, 0.3],
       [4.4, 3.2, 1.3, 0.2],
       [5. , 3.5, 1.6, 0.6],
       [5.1, 3.8, 1.9, 0.4],
       [4.8, 3. , 1.4, 0.3],
       [5.1, 3.8, 1.6, 0.2],
       [4.6, 3.2, 1.4, 0.2],
       [5.3, 3.7, 1.5, 0.2],
       [5. , 3.3, 1.4, 0.2],
       [7. , 3.2, 4.7, 1.4],
       [6.4, 3.2, 4.5, 1.5],
       [6.9, 3.1, 4.9, 1.5],
       [5.5, 2.3, 4. , 1.3],
       [6.5, 2.8, 4.6, 1.5],
       [5.7, 2.8, 4.5, 1.3],
       [6.3, 3.3, 4.7, 1.6],
       [4.9, 2.4, 3.3, 1. ],
       [6.6, 2.9, 4.6, 1.3],
       [5.2, 2.7, 3.9, 1.4],
       [5. , 2. , 3.5, 1. ],
       [5.9, 3. , 4.2, 1.5],
       [6. , 2.2, 4. , 1. ],
       [6.1, 2.9, 4.7, 1.4],
       [5.6, 2.9, 3.6, 1.3],
       [6.7, 3.1, 4.4, 1.4],
       [5.6, 3. , 4.5, 1.5],
       [5.8, 2.7, 4.1, 1. ],
       [6.2, 2.2, 4.5, 1.5],
       [5.6, 2.5, 3.9, 1.1],
       [5.9, 3.2, 4.8, 1.8],
       [6.1, 2.8, 4. , 1.3],
       [6.3, 2.5, 4.9, 1.5],
       [6.1, 2.8, 4.7, 1.2],
       [6.4, 2.9, 4.3, 1.3],
       [6.6, 3. , 4.4, 1.4],
       [6.8, 2.8, 4.8, 1.4],
       [6.7, 3. , 5. , 1.7],
       [6. , 2.9, 4.5, 1.5],
       [5.7, 2.6, 3.5, 1. ],
       [5.5, 2.4, 3.8, 1.1],
       [5.5, 2.4, 3.7, 1. ],
       [5.8, 2.7, 3.9, 1.2],
       [6. , 2.7, 5.1, 1.6],
       [5.4, 3. , 4.5, 1.5],
       [6. , 3.4, 4.5, 1.6],
       [6.7, 3.1, 4.7, 1.5],
       [6.3, 2.3, 4.4, 1.3],
       [5.6, 3. , 4.1, 1.3],
       [5.5, 2.5, 4. , 1.3],
       [5.5, 2.6, 4.4, 1.2],
       [6.1, 3. , 4.6, 1.4],
       [5.8, 2.6, 4. , 1.2],
       [5. , 2.3, 3.3, 1. ],
       [5.6, 2.7, 4.2, 1.3],
       [5.7, 3. , 4.2, 1.2],
       [5.7, 2.9, 4.2, 1.3],
       [6.2, 2.9, 4.3, 1.3],
       [5.1, 2.5, 3. , 1.1],
       [5.7, 2.8, 4.1, 1.3],
       [6.3, 3.3, 6. , 2.5],
       [5.8, 2.7, 5.1, 1.9],
       [7.1, 3. , 5.9, 2.1],
       [6.3, 2.9, 5.6, 1.8],
       [6.5, 3. , 5.8, 2.2],
       [7.6, 3. , 6.6, 2.1],
       [4.9, 2.5, 4.5, 1.7],
       [7.3, 2.9, 6.3, 1.8],
       [6.7, 2.5, 5.8, 1.8],
       [7.2, 3.6, 6.1, 2.5],
       [6.5, 3.2, 5.1, 2. ],
       [6.4, 2.7, 5.3, 1.9],
       [6.8, 3. , 5.5, 2.1],
       [5.7, 2.5, 5. , 2. ],
       [5.8, 2.8, 5.1, 2.4],
       [6.4, 3.2, 5.3, 2.3],
       [6.5, 3. , 5.5, 1.8],
       [7.7, 3.8, 6.7, 2.2],
       [7.7, 2.6, 6.9, 2.3],
       [6. , 2.2, 5. , 1.5],
       [6.9, 3.2, 5.7, 2.3],
       [5.6, 2.8, 4.9, 2. ],
       [7.7, 2.8, 6.7, 2. ],
       [6.3, 2.7, 4.9, 1.8],
       [6.7, 3.3, 5.7, 2.1],
       [7.2, 3.2, 6. , 1.8],
       [6.2, 2.8, 4.8, 1.8],
       [6.1, 3. , 4.9, 1.8],
       [6.4, 2.8, 5.6, 2.1],
       [7.2, 3. , 5.8, 1.6],
       [7.4, 2.8, 6.1, 1.9],
       [7.9, 3.8, 6.4, 2. ],
       [6.4, 2.8, 5.6, 2.2],
       [6.3, 2.8, 5.1, 1.5],
       [6.1, 2.6, 5.6, 1.4],
       [7.7, 3. , 6.1, 2.3],
       [6.3, 3.4, 5.6, 2.4],
       [6.4, 3.1, 5.5, 1.8],
       [6. , 3. , 4.8, 1.8],
       [6.9, 3.1, 5.4, 2.1],
       [6.7, 3.1, 5.6, 2.4],
       [6.9, 3.1, 5.1, 2.3],
       [5.8, 2.7, 5.1, 1.9],
       [6.8, 3.2, 5.9, 2.3],
       [6.7, 3.3, 5.7, 2.5],
       [6.7, 3. , 5.2, 2.3],
       [6.3, 2.5, 5. , 1.9],
       [6.5, 3. , 5.2, 2. ],
       [6.2, 3.4, 5.4, 2.3],
       [5.9, 3. , 5.1, 1.8]]), 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]), 'target_names':
array(['setosa', 'versicolor', 'virginica'], dtype='<U10'), 'DESCR': '..
_iris_dataset:\n\nIris plants dataset\n--------------------\n\n**Data Set
Characteristics:**\n\n    :Number of Instances: 150 (50 in each of three
classes)\n    :Number of Attributes: 4 numeric, predictive attributes and the
class\n    :Attribute Information:\n        - sepal length in cm\n        -
sepal width in cm\n        - petal length in cm\n        - petal width in cm\n
- class:\n                - Iris-Setosa\n                - Iris-Versicolour\n
- Iris-Virginica\n                \n    :Summary Statistics:\n\n
============== ==== ==== ======= ===== ====================\n
Min  Max   Mean    SD   Class Correlation\n    ============== ==== ==== =======
===== ====================\n    sepal length:   4.3  7.9   5.84   0.83
0.7826\n    sepal width:    2.0  4.4   3.05   0.43   -0.4194\n    petal length:
1.0  6.9   3.76   1.76    0.9490  (high!)\n    petal width:    0.1  2.5   1.20
0.76    0.9565  (high!)\n    ============== ==== ==== ======= =====
====================\n\n    :Missing Attribute Values: None\n    :Class
Distribution: 33.3% for each of 3 classes.\n    :Creator: R.A. Fisher\n
:Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)\n    :Date: July,
1988\n\nThe famous Iris database, first used by Sir R.A. Fisher. The dataset is
taken\nfrom Fisher\'s paper. Note that it\'s the same as in R, but not as in the
UCI\nMachine Learning Repository, which has two wrong data points.\n\nThis is
perhaps the best known database to be found in the\npattern recognition
literature.  Fisher\'s paper is a classic in the field and\nis referenced
frequently to this day.  (See Duda & Hart, for example.)  The\ndata set contains
3 classes of 50 instances each, where each class refers to a\ntype of iris
plant.  One class is linearly separable from the other 2; the\nlatter are NOT
linearly separable from each other.\n\n.. topic:: References\n\n   - Fisher,
R.A. "The use of multiple measurements in taxonomic problems"\n     Annual
Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to\n
Mathematical Statistics" (John Wiley, NY, 1950).\n   - Duda, R.O., & Hart, P.E.
(1973) Pattern Classification and Scene Analysis.\n     (Q327.D83) John Wiley &
Sons.  ISBN 0-471-22361-1.  See page 218.\n   - Dasarathy, B.V. (1980) "Nosing
Around the Neighborhood: A New System\n     Structure and Classification Rule
for Recognition in Partially Exposed\n     Environments".  IEEE Transactions on
Pattern Analysis and Machine\n     Intelligence, Vol. PAMI-2, No. 1, 67-71.\n
- Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactions\n
on Information Theory, May 1972, 431-433.\n   - See also: 1988 MLC Proceedings,
54-64.  Cheeseman et al"s AUTOCLASS II\n     conceptual clustering system finds
3 classes in the data.\n   - Many, many more …', 'feature_names': ['sepal
length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)'],
'filename': '/usr/local/lib/python3.7/dist-
packages/sklearn/datasets/data/iris.csv'}
```

[ ]:
```python
print(iris.DESCR)
```

```
.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**
```

```
    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica

    :Summary Statistics:

    ============== ==== ==== ======= ===== ====================
                    Min  Max   Mean    SD   Class Correlation
    ============== ==== ==== ======= ===== ====================
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
    ============== ==== ==== ======= ===== ====================

    :Missing Attribute Values: None
    :Class Distribution: 33.3% for each of 3 classes.
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988

The famous Iris database, first used by Sir R.A. Fisher. The dataset is taken
from Fisher's paper. Note that it's the same as in R, but not as in the UCI
Machine Learning Repository, which has two wrong data points.

This is perhaps the best known database to be found in the
pattern recognition literature.  Fisher's paper is a classic in the field and
is referenced frequently to this day.  (See Duda & Hart, for example.)  The
data set contains 3 classes of 50 instances each, where each class refers to a
type of iris plant.  One class is linearly separable from the other 2; the
latter are NOT linearly separable from each other.

.. topic:: References

   - Fisher, R.A. "The use of multiple measurements in taxonomic problems"
     Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to
     Mathematical Statistics" (John Wiley, NY, 1950).
   - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.
     (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.
   - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System
```

```
    Structure and Classification Rule for Recognition in Partially Exposed
    Environments".  IEEE Transactions on Pattern Analysis and Machine
    Intelligence, Vol. PAMI-2, No. 1, 67-71.
  - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactions
    on Information Theory, May 1972, 431-433.
  - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II
    conceptual clustering system finds 3 classes in the data.
  - Many, many more …
```

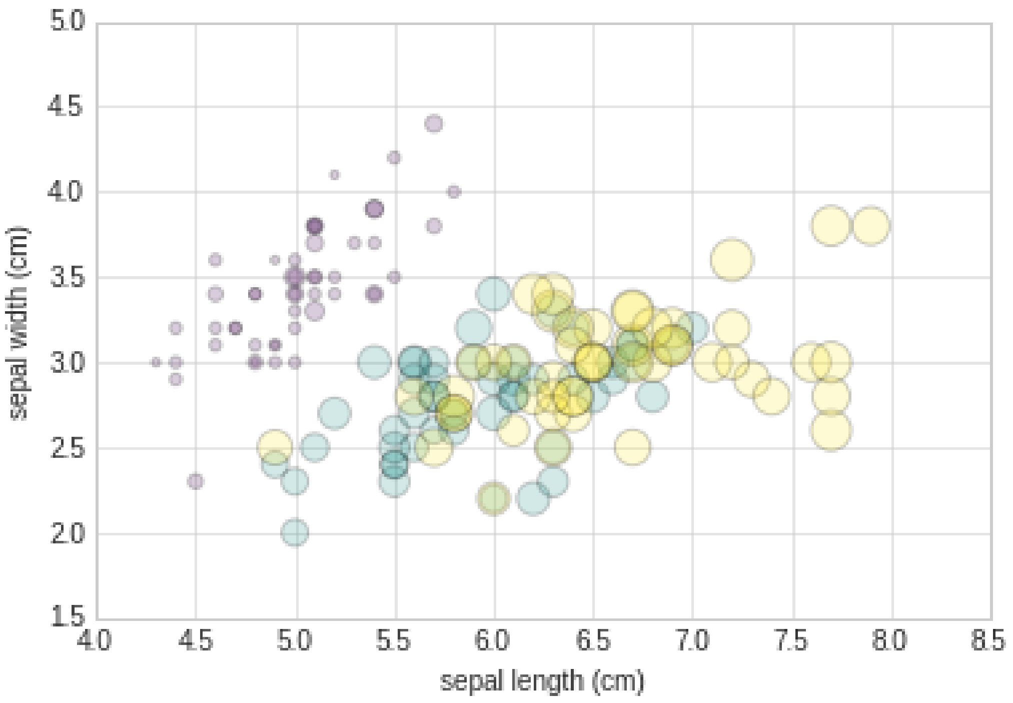
[ ]:
```
iris.data[:10]
```

[ ]:
```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1]])
```

[ ]:
```
features = iris.data.T
features # data in each column (feature)
```

[ ]:
```
array([[5.1, 4.9, 4.7, 4.6, 5. , 5.4, 4.6, 5. , 4.4, 4.9, 5.4, 4.8, 4.8,
        4.3, 5.8, 5.7, 5.4, 5.1, 5.7, 5.1, 5.4, 5.1, 4.6, 5.1, 4.8, 5. ,
        5. , 5.2, 5.2, 4.7, 4.8, 5.4, 5.2, 5.5, 4.9, 5. , 5.5, 4.9, 4.4,
        5.1, 5. , 4.5, 4.4, 5. , 5.1, 4.8, 5.1, 4.6, 5.3, 5. , 7. , 6.4,
        6.9, 5.5, 6.5, 5.7, 6.3, 4.9, 6.6, 5.2, 5. , 5.9, 6. , 6.1, 5.6,
        6.7, 5.6, 5.8, 6.2, 5.6, 5.9, 6.1, 6.3, 6.1, 6.4, 6.6, 6.8, 6.7,
        6. , 5.7, 5.5, 5.5, 5.8, 6. , 5.4, 6. , 6.7, 6.3, 5.6, 5.5, 5.5,
        6.1, 5.8, 5. , 5.6, 5.7, 5.7, 6.2, 5.1, 5.7, 6.3, 5.8, 7.1, 6.3,
        6.5, 7.6, 4.9, 7.3, 6.7, 7.2, 6.5, 6.4, 6.8, 5.7, 5.8, 6.4, 6.5,
        7.7, 7.7, 6. , 6.9, 5.6, 7.7, 6.3, 6.7, 7.2, 6.2, 6.1, 6.4, 7.2,
        7.4, 7.9, 6.4, 6.3, 6.1, 7.7, 6.3, 6.4, 6. , 6.9, 6.7, 6.9, 5.8,
        6.8, 6.7, 6.7, 6.3, 6.5, 6.2, 5.9],
       [3.5, 3. , 3.2, 3.1, 3.6, 3.9, 3.4, 3.4, 2.9, 3.1, 3.7, 3.4, 3. ,
        3. , 4. , 4.4, 3.9, 3.5, 3.8, 3.8, 3.4, 3.7, 3.6, 3.3, 3.4, 3. ,
        3.4, 3.5, 3.4, 3.2, 3.1, 3.4, 4.1, 4.2, 3.1, 3.2, 3.5, 3.6, 3. ,
        3.4, 3.5, 2.3, 3.2, 3.5, 3.8, 3. , 3.8, 3.2, 3.7, 3.3, 3.2, 3.2,
        3.1, 2.3, 2.8, 2.8, 3.3, 2.4, 2.9, 2.7, 2. , 3. , 2.2, 2.9, 2.9,
        3.1, 3. , 2.7, 2.2, 2.5, 3.2, 2.8, 2.5, 2.8, 2.9, 3. , 2.8, 3. ,
        2.9, 2.6, 2.4, 2.4, 2.7, 2.7, 3. , 3.4, 3.1, 2.3, 3. , 2.5, 2.6,
        3. , 2.6, 2.3, 2.7, 3. , 2.9, 2.9, 2.5, 2.8, 3.3, 2.7, 3. , 2.9,
        3. , 3. , 2.5, 2.9, 2.5, 3.6, 3.2, 2.7, 3. , 2.5, 2.8, 3.2, 3. ,
        3.8, 2.6, 2.2, 3.2, 2.8, 2.8, 2.7, 3.3, 3.2, 2.8, 3. , 2.8, 3. ,
        2.8, 3.8, 2.8, 2.8, 2.6, 3. , 3.4, 3.1, 3. , 3.1, 3.1, 3.1, 2.7,
        3.2, 3.3, 3. , 2.5, 3. , 3.4, 3. ],
       [1.4, 1.4, 1.3, 1.5, 1.4, 1.7, 1.4, 1.5, 1.4, 1.5, 1.5, 1.6, 1.4,
        1.1, 1.2, 1.5, 1.3, 1.4, 1.7, 1.5, 1.7, 1.5, 1. , 1.7, 1.9, 1.6,
        1.6, 1.5, 1.4, 1.6, 1.6, 1.5, 1.5, 1.4, 1.5, 1.2, 1.3, 1.4, 1.3,
        1.5, 1.3, 1.3, 1.3, 1.6, 1.9, 1.4, 1.6, 1.4, 1.5, 1.4, 4.7, 4.5,
        4.9, 4. , 4.6, 4.5, 4.7, 3.3, 4.6, 3.9, 3.5, 4.2, 4. , 4.7, 3.6,
        4.4, 4.5, 4.1, 4.5, 3.9, 4.8, 4. , 4.9, 4.7, 4.3, 4.4, 4.8, 5. ,
        4.5, 3.5, 3.8, 3.7, 3.9, 5.1, 4.5, 4.5, 4.7, 4.4, 4.1, 4. , 4.4,
        4.6, 4. , 3.3, 4.2, 4.2, 4.2, 4.3, 3. , 4.1, 6. , 5.1, 5.9, 5.6,
        5.8, 6.6, 4.5, 6.3, 5.8, 6.1, 5.1, 5.3, 5.5, 5. , 5.1, 5.3, 5.5,
        6.7, 6.9, 5. , 5.7, 4.9, 6.7, 4.9, 5.7, 6. , 4.8, 4.9, 5.6, 5.8,
        6.1, 6.4, 5.6, 5.1, 5.6, 6.1, 5.6, 5.5, 4.8, 5.4, 5.6, 5.1, 5.1,
        5.9, 5.7, 5.2, 5. , 5.2, 5.4, 5.1],
       [0.2, 0.2, 0.2, 0.2, 0.2, 0.4, 0.3, 0.2, 0.2, 0.1, 0.2, 0.2, 0.1,
        0.1, 0.2, 0.4, 0.4, 0.3, 0.3, 0.3, 0.2, 0.4, 0.2, 0.5, 0.2, 0.2,
        0.4, 0.2, 0.2, 0.2, 0.2, 0.4, 0.1, 0.2, 0.2, 0.2, 0.2, 0.1, 0.2,
        0.2, 0.3, 0.3, 0.2, 0.6, 0.4, 0.3, 0.2, 0.2, 0.2, 0.2, 1.4, 1.5,
        1.5, 1.3, 1.5, 1.3, 1.6, 1. , 1.3, 1.4, 1. , 1.5, 1. , 1.4, 1.3,
        1.4, 1.5, 1. , 1.5, 1.1, 1.8, 1.3, 1.5, 1.2, 1.3, 1.4, 1.4, 1.7,
        1.5, 1. , 1.1, 1. , 1.2, 1.6, 1.5, 1.6, 1.5, 1.3, 1.3, 1.3, 1.2,
        1.4, 1.2, 1. , 1.3, 1.2, 1.3, 1.3, 1.1, 1.3, 2.5, 1.9, 2.1, 1.8,
        2.2, 2.1, 1.7, 1.8, 1.8, 2.5, 2. , 1.9, 2.1, 2. , 2.4, 2.3, 1.8,
        2.2, 2.3, 1.5, 2.3, 2. , 2. , 1.8, 2.1, 1.8, 1.8, 1.8, 2.1, 1.6,
        1.9, 2. , 2.2, 1.5, 1.4, 2.3, 2.4, 1.8, 1.8, 2.1, 2.4, 2.3, 1.9,
        2.3, 2.5, 2.3, 1.9, 2. , 2.3, 1.8]])
```

[ ]:
```
# scatter plot of sepal length (x) and sepal width (y); size = petal width;␣
↪color = target (class of flower)
plt.scatter(features[0], features[1], alpha=0.2, s=100*features[3], c=iris.
↪target, cmap='viridis')

plt.xlabel(iris.feature_names[0])
plt.ylabel(iris.feature_names[1]);
```



We can see that this scatter plot has given us the ability to simultaneously explore four different dimensions of the data: * the (x, y) location of each point corresponds to the sepal length and sepal width * the size of the point is related to the petal width * the color is related to the particular species of flower

Multicolor and multifeature scatter plot like this can be used for both exploration and presentation of data.

As datasets get larger than a few thousand points, `plt.plot` can be noticeably more efficient that `plt.scatter`. Therefore, `plt.plot` is preferable to `plt.scatter`.

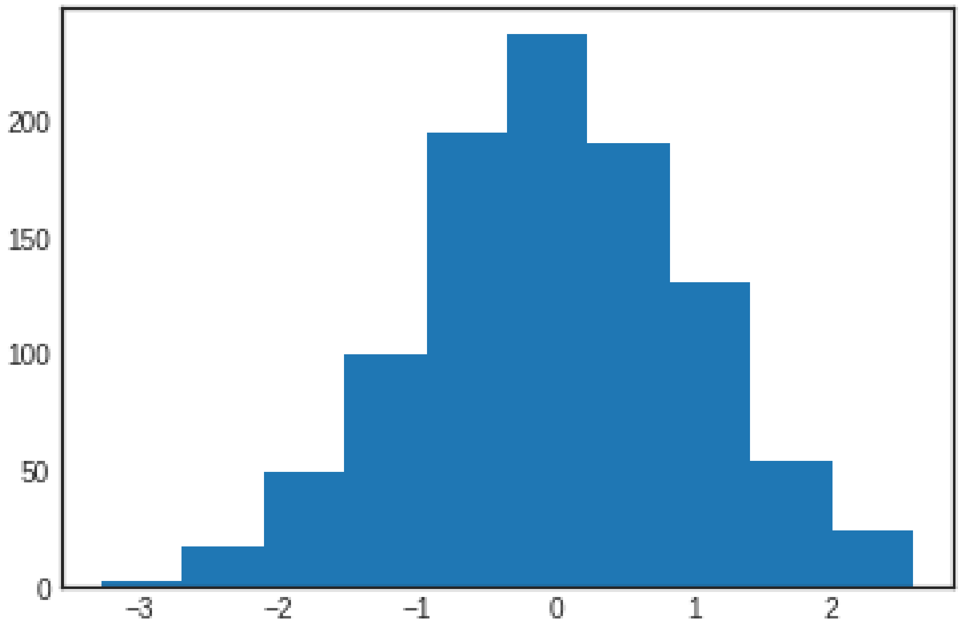
8 Histograms, Binnings, and Density

A simple histogram can be a great first step in understanding a dataset.

```
[ ]: %matplotlib inline
     import numpy as np
     import matplotlib.pyplot as plt
     plt.style.use('seaborn-white')

     data = np.random.randn(1000)
```

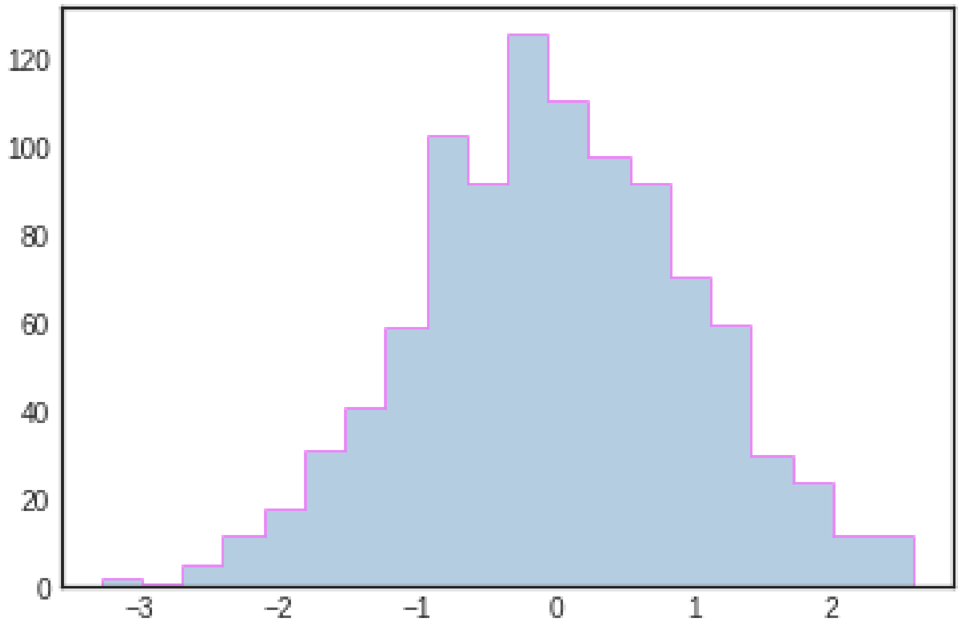
```
[ ]: data.std()
```

```
[ ]: 1.007901045644668
[ ]: plt.hist(data);
```



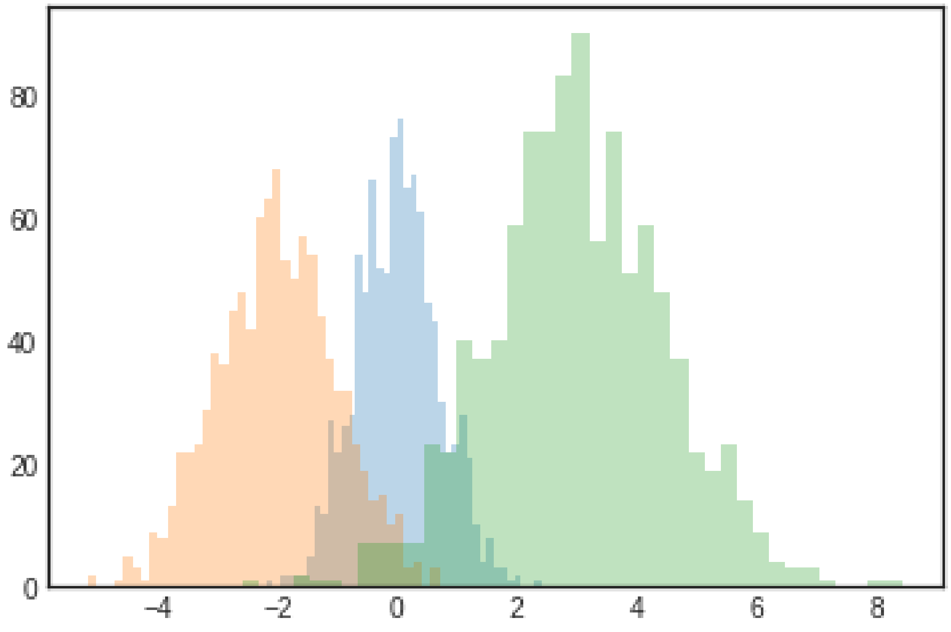
The `hist()` function has many options to tune both the calculation and the display.

```
[ ]: plt.hist(data, bins=20, alpha=0.4, histtype='stepfilled', color='steelblue',␣
      ↪edgecolor='magenta');
```



histtype='stepfilled' and alpha are very useful when comparing histograms of several distributions.

```
[ ]: x1 = np.random.normal(0, 0.7, 1000)
     x2 = np.random.normal(-2, 1, 1000)
     x3 = np.random.normal(3, 1.5, 1000)

     kwargs = dict(histtype='stepfilled', alpha=0.3, bins=40)

     plt.hist(x1, **kwargs)
     plt.hist(x2, **kwargs)
     plt.hist(x3, **kwargs);
```



9 Customizing Plot Legends

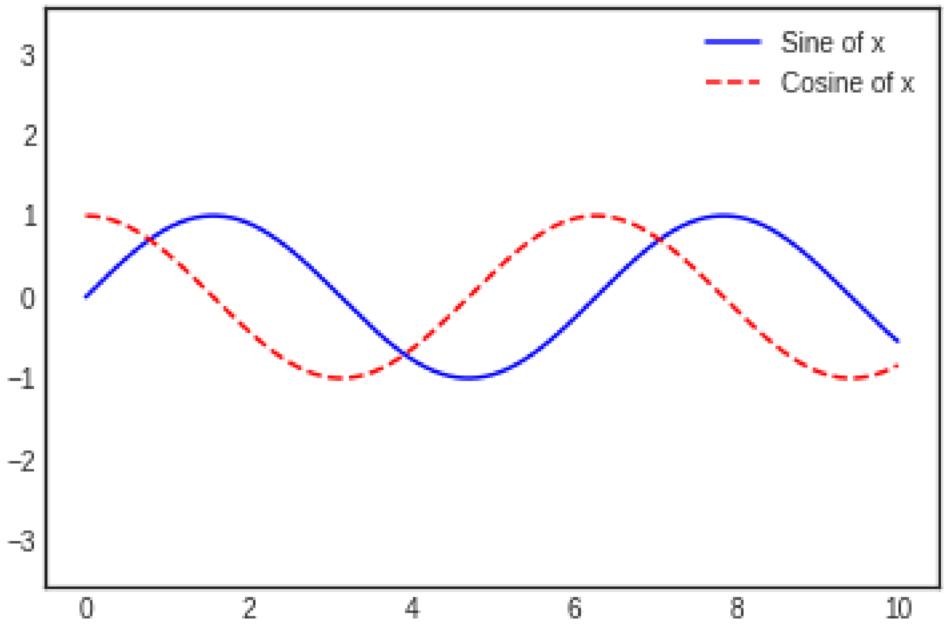
Plot legends give meaning to the visualization, assigning labels to the various plot elements.

The simplest legend can be created with the plt.legend() command, which automatically creates a legend for any labeled plot elements.

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

```
[ ]: %matplotlib inline
     import numpy as np
```

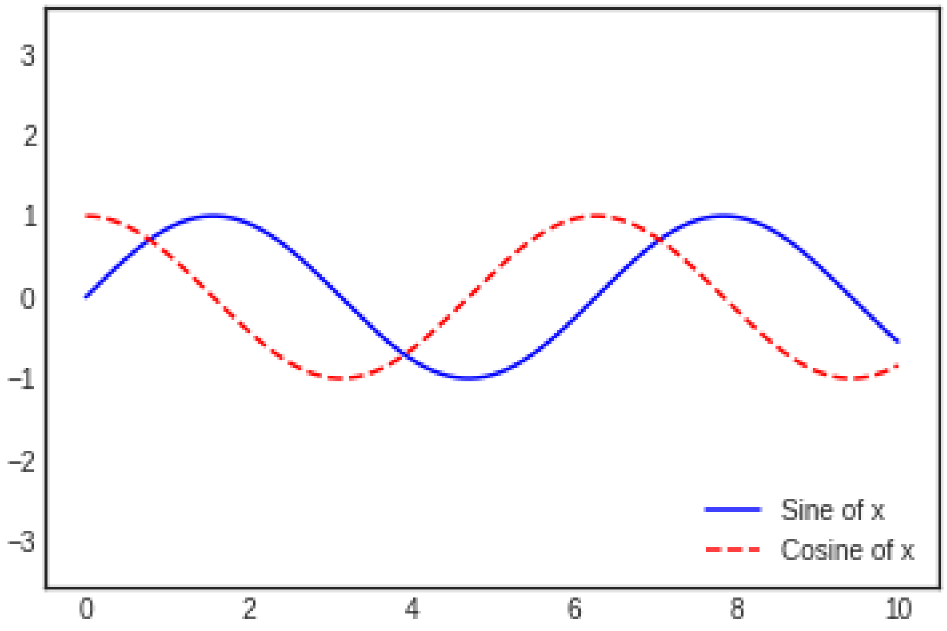
```
[ ]: x = np.linspace(0, 10, 1000)
     fig, ax = plt.subplots()

     ax.plot(x, np.sin(x), '-b', label='Sine of x')
     ax.plot(x, np.cos(x), '--r', label='Cosine of x')
     ax.axis('equal')
     leg = ax.legend();
```



We can customize the legend. For example, we can specify the location and turn off the frame.

```
[ ]: ax.legend(loc='lower right', frameon=False) # legend on upper left corner with␣
     ↪no frame
     fig
```

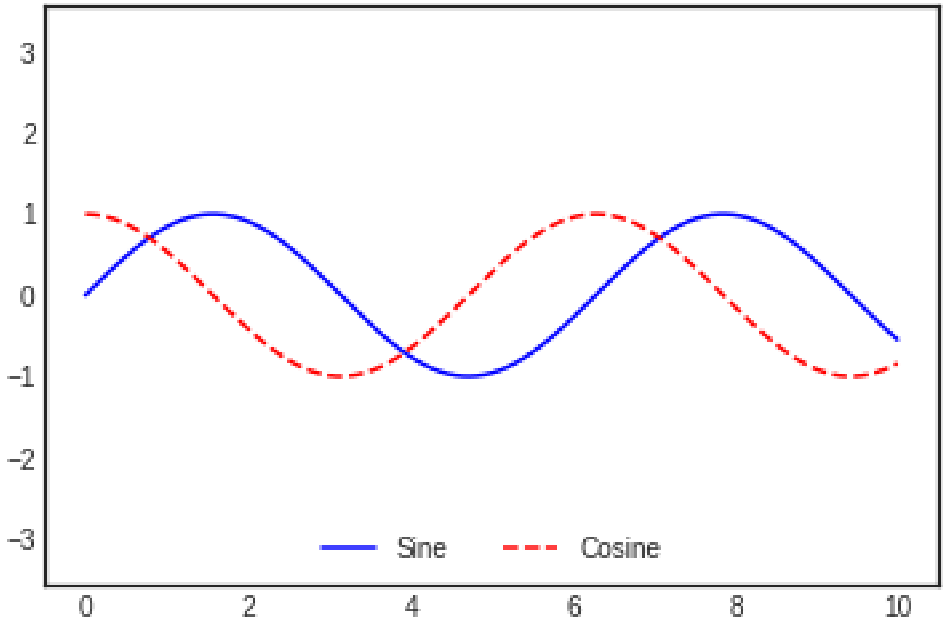
```
[ ]:
```



We can use the `ncol` command to specify the number of columns in the legend.

[ ]:
```
ax.legend(frameon= False, loc='lower center', ncol=2)
fig
```

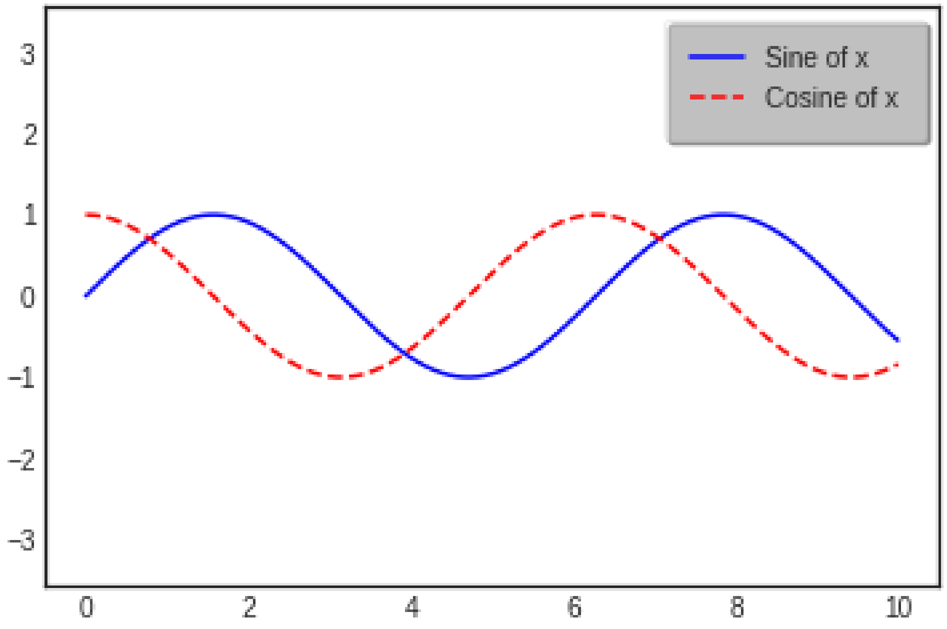
[ ]:



We can also use a rounded box (fancybox), add a shadow, change the transparency of the frame, or change the padding of the text.

[ ]:
```
ax.legend(frameon = True,fancybox=True, framealpha=0.3, shadow=True,␣
 ↪borderpad=1)
fig
```

[ ]:

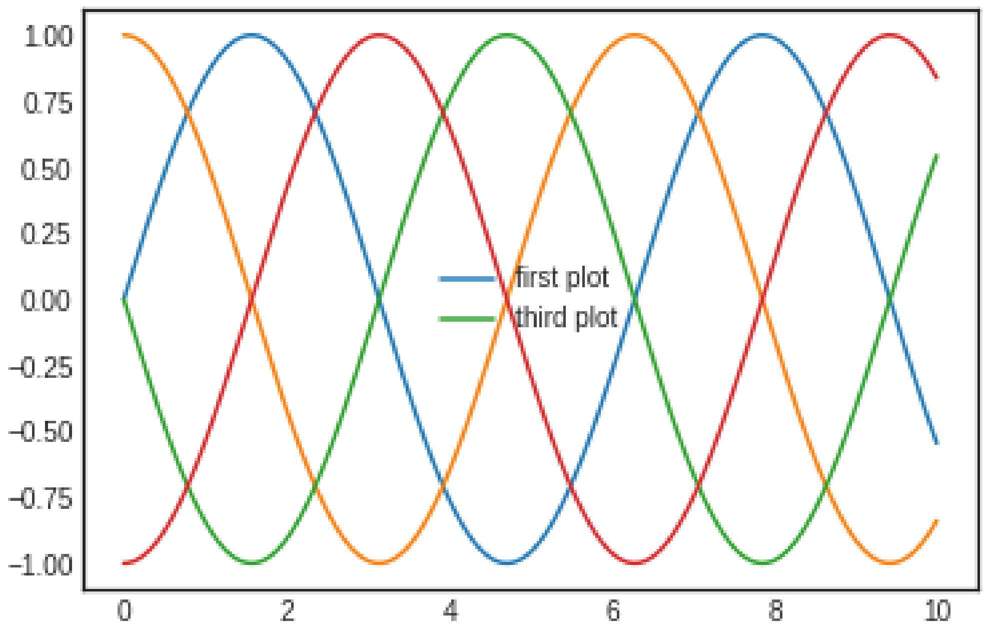


9.1 Choosing Elements for the Legend

We can fine-tune which elements and labels appear in the legend by using the objects returned by the plot commands.

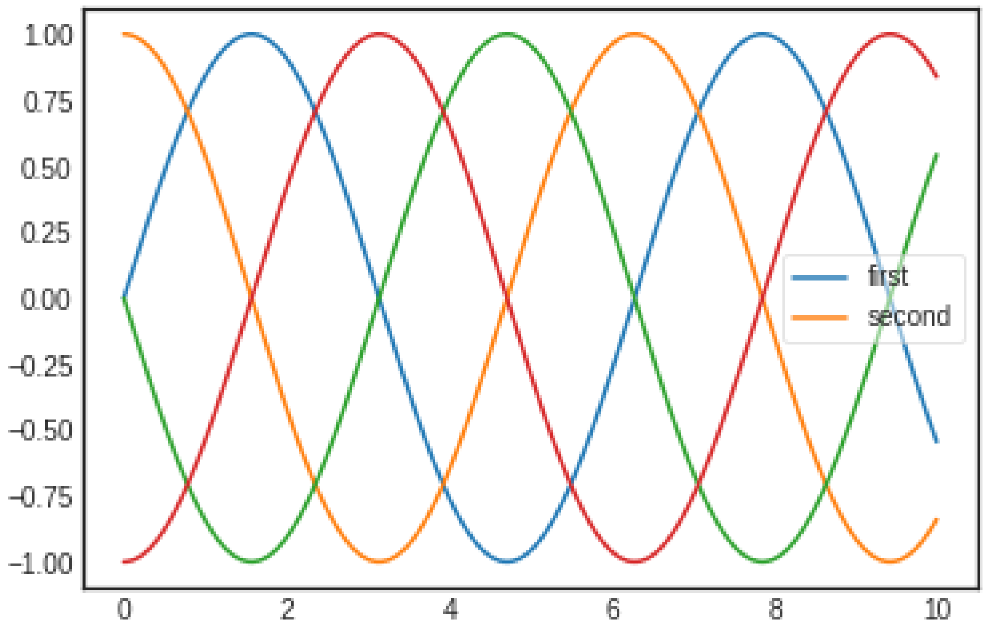
The `plt.plot()` command is able to create multiple lines at once, and returns a list of created line instances. Passing any of these to `plt.legend()` will tell it which to identify, along with the labels we would like to specify.

[ ]:
```
y = np.sin(x[:, np.newaxis] + np.pi * np.arange(0, 2, 0.5))
lines = plt.plot(x, y)

# lines is a list of plt.Line2D instances
plt.legend(lines[:2], ['first', 'second']);
```



You can also apply labels to the plot elements you would like to show on the legend as follows.

```
[ ]: plt.plot(x, y[:, 0], label='first') # plot the first column of y
     plt.plot(x, y[:, 1], label='second') # plot the second column of y
     plt.plot(x, y[:, 2:]) # plot the rest of the columns of y
     plt.legend(framealpha=0.5, frameon=True);
```



Note that by default, the legend ignores all elements without a label attribute set.

9.2 Legend for size of points

Let's look at an example of using size of points to indicate populations of California cities.

[ ]:
```
import pandas as pd
cities = pd.read_csv('/content/drive/MyDrive/Python Training/
↪PythonDataScienceHandbook-master/notebooks/data/california_cities.csv')

cities.head()
```

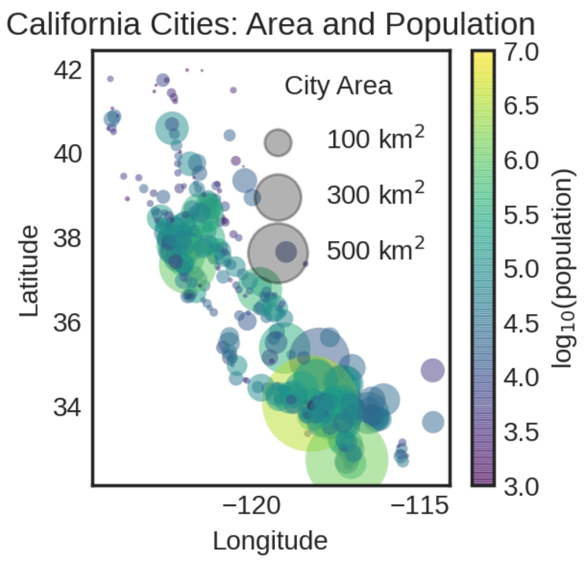
[ ]:
```
   Unnamed: 0         city  …  area_water_km2  area_water_percent
0           0     Adelanto  …           0.046                0.03
1           1  AgouraHills  …           0.076                0.37
2           2      Alameda  …          31.983               53.79
3           3       Albany  …           9.524               67.28
4           4     Alhambra  …           0.003                0.01

[5 rows x 14 columns]
```

[ ]:
```
cities.shape
```

[ ]:
```
(482, 14)
```

[ ]:
```
# extract the data we are interested in
lat, lon = cities['latd'], cities['longd']
population, area = cities['population_total'], cities['area_total_km2']
```

[ ]:
```
plt.figure(figsize=(3,3), dpi=200)

# scatter plot the points, using size and color but no label
plt.scatter(lon, lat, label=None, c=np.log10(population), cmap='viridis',␣
↪s=area, linewidth=0, alpha=0.5)

plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.colorbar(label='log$_{10}$(population)')
plt.clim(3, 7)

# Let's create a legend
# we will plot empty lists with the desired size and label
for area in [100, 300, 500]:
  plt.scatter([], [], c='k', alpha=0.3, s=area, label=str(area) + ' km$^2$')

plt.legend(scatterpoints=1, frameon=False, labelspacing=1, title='City Area')
plt.title('California Cities: Area and Population');
```



The objects we want (the gray circles in the legend) are not on the plot, so we faked them by plotting empty lists. Note that the legend only lists plot elements that have a label specified.

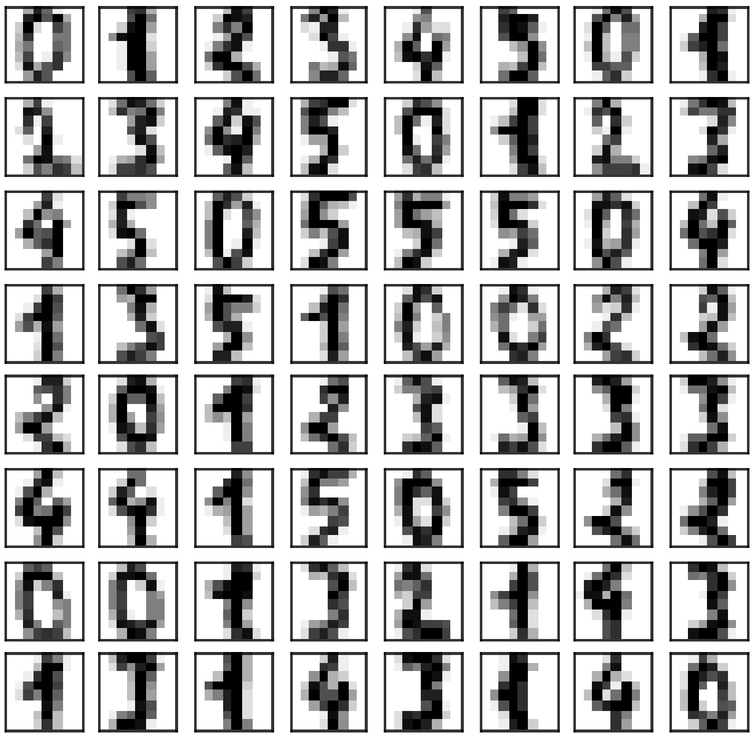
For smaller amounts of data, it does not matter which one you use - `plot()` or `scatter()`. However, when the datasets get larger than a few thousand points, `plt.plot()` can be noticeably more efficient than `plt.scatter()`.

9.3 Example: Handwritten Digits

Let's perform a visualization of some hand-written digits data, which is included in Scikit-Learn. The data consists of nearly 2,000 thumbnails of size 8x8 pixels showing handwritten digits.

We first download the data and visualize some examples with `plt.imshow()`.

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

# load images of the digits 0 to 5 and visualize several of them
from sklearn.datasets import load_digits
digits = load_digits(n_class=6)

fig, ax = plt.subplots(8, 8, figsize=(5, 5))
for i, axi in enumerate(ax.flat):
  axi.imshow(digits.images[i], cmap='binary')
  axi.set(xticks=[], yticks=[]) # no x and y ticks
```

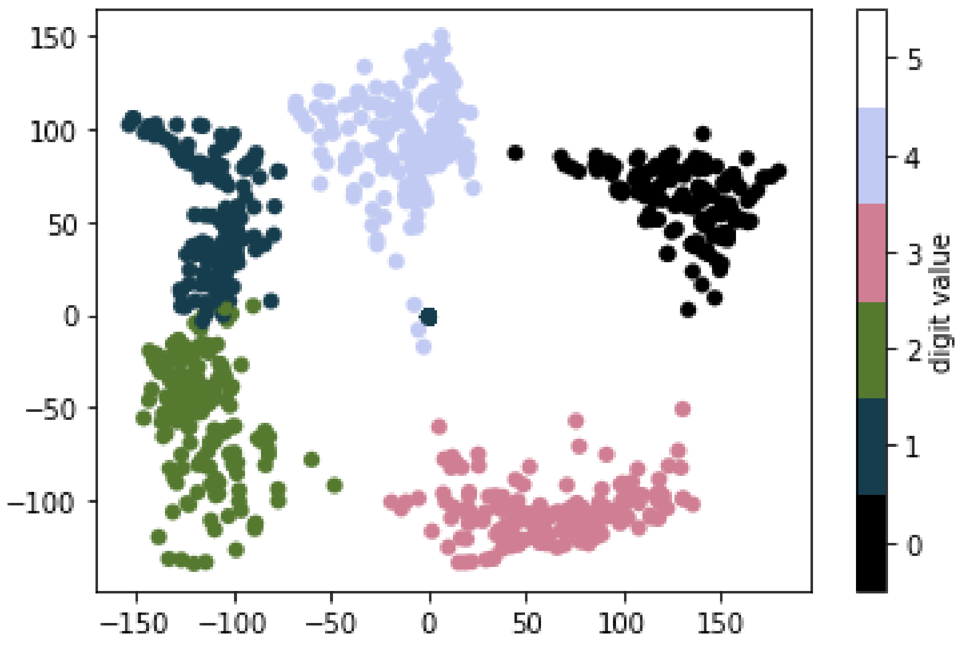


As each digit is defined by the hue of 64 pixels (8x8 pixels), we can think of each digit as a point lying in a 64-dimensional space. Visualizing relationships in such high-dimensional space is extremely difficult.

Therefore, we can reduce the dimensionality of the data while maintaining the relationships of interest.

```
[ ]: # project the digits into 2-dimensions using IsoMap
     from sklearn.manifold import Isomap
     iso = Isomap(n_components=2)
     projection = iso.fit_transform(digits.data)
```

```
[ ]: # plot the results
     plt.scatter(projection[:, 0], projection[:, 1], lw=0.1, c=digits.target,␣
      ↪cmap=plt.cm.get_cmap('cubehelix', 6))
     plt.colorbar(ticks=range(6), label='digit value')
     plt.clim(-0.5, 5.5)
```



10 Multiple subplots

Sometimes we need to compare different views of data side by side. Matplotlib has the concept of *subplots*, which are groups of smaller axes that can exist together within a figure.

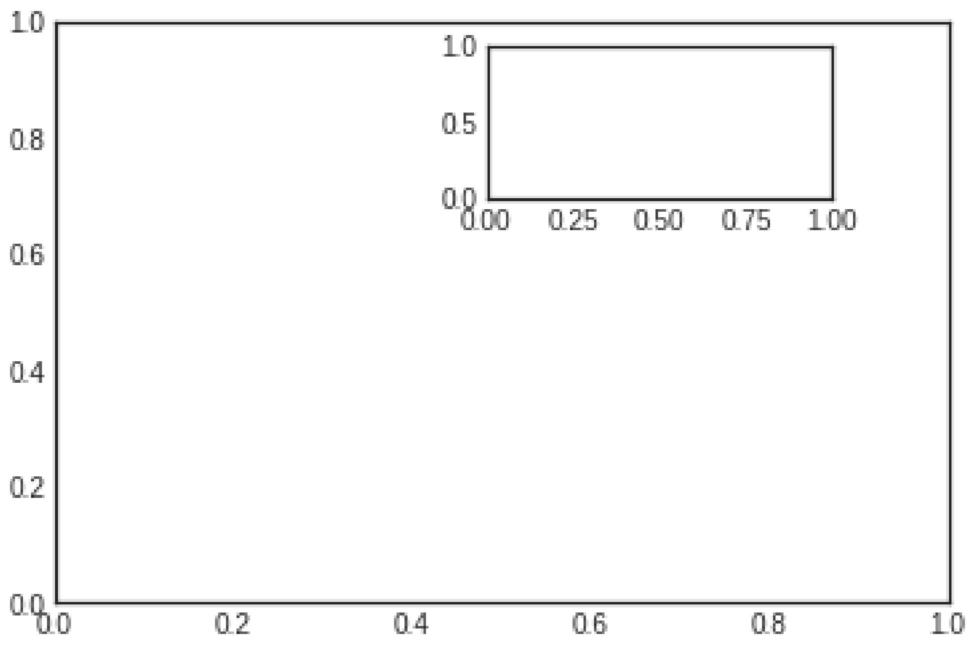
[ ]:
```
%matplotlib inline
import matplotlib.pyplot as plt

plt.style.use('seaborn-white')
import numpy as np
```

10.1 plt.axes() : Subplots by hand

`plt.axes()` takes an optional argument - a list of 4 numbers in the figure coordinate system. The numbers represent [*left, bottom, width, height*] in the figure coordinate system.

Let's create an inset axes at the top-right corner of another axes by * setting the `x` and `y` position to 0.65 (65% of the width and 65% of the height of the figure) * `x` and `y` extents to 0.2 (size of the axes: 20% of the width, and 20% of the height of the figure)

[ ]:
```
ax1 = plt.axes() # standard axes
ax2 = plt.axes([0.5, 0.65, 0.3, 0.2])
```

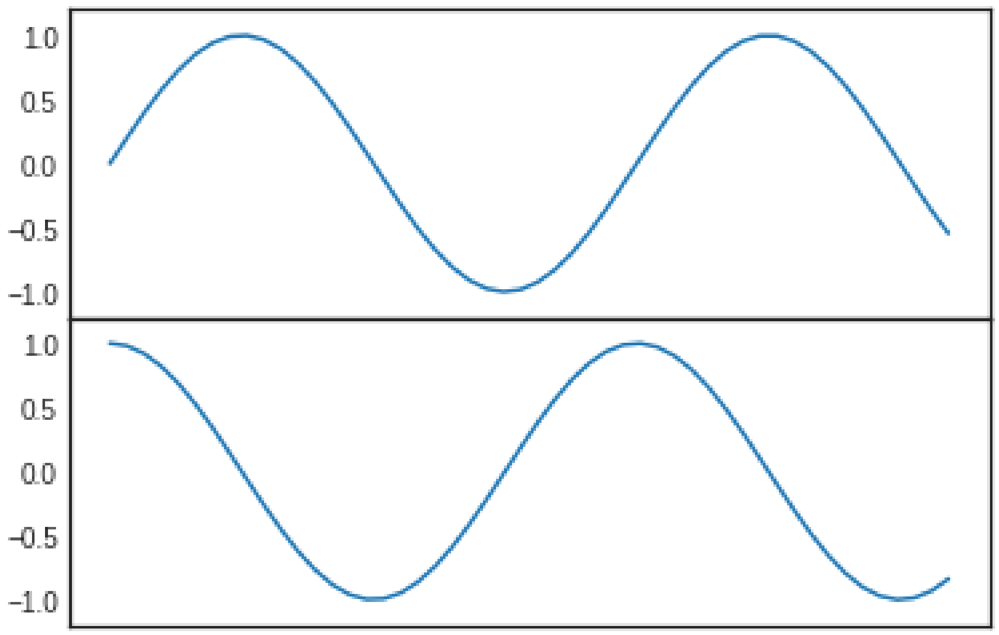


The equivalent of this command in object-oriented interface is: `fig.add_axes()`.

Let's create a vertically stacked axes using `fig.add_axes()`.

```
[ ]: fig = plt.figure()
     ax1 = fig.add_axes([0.1, 0.5, 0.8, 0.4], xticklabels=[], ylim=(-1.2, 1.2))
     ax2 = fig.add_axes([0.1, 0.1, 0.8, 0.4], xticklabels=[], ylim=(-1.2, 1.2))

     x = np.linspace(0, 10)
     ax1.plot(np.sin(x))
     ax2.plot(np.cos(x))
```

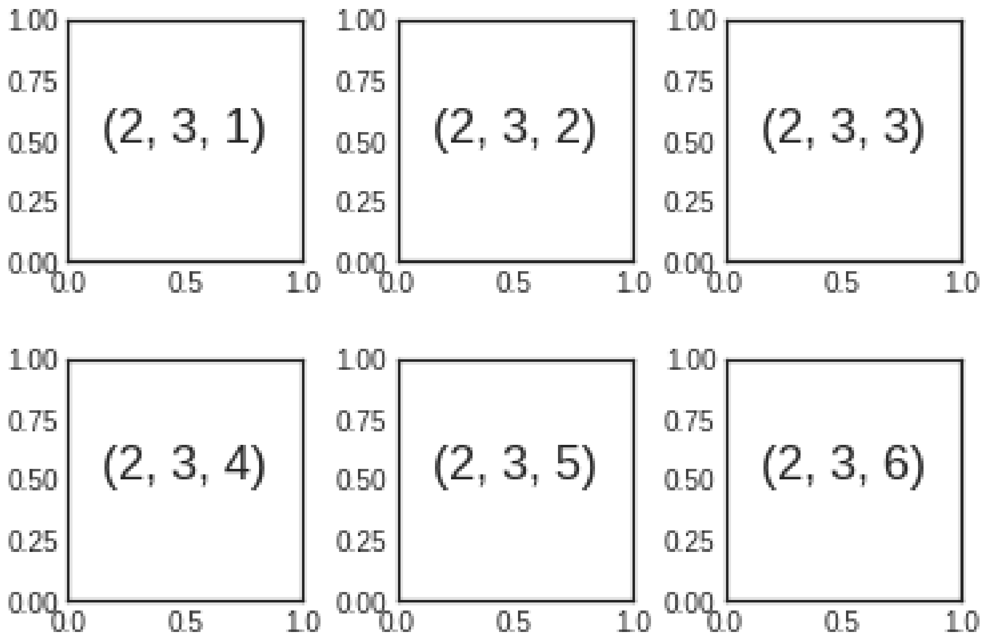
```
[ ]: [<matplotlib.lines.Line2D at 0x7fac1dc0a050>]
```



10.2 plt.subplot(): Simple grids of subplots

plt.subplot() creates a single subplot within a grid. This command takes 3 integer arguments: * number of rows * number of columns * index of the plot to be created

[ ]:
```
fig = plt.figure()

# adjust the spacing between plots (40% of the subplot width and height)
fig.subplots_adjust(hspace=0.4, wspace=0.4)

for i in range(1, 7):
  plt.subplot(2, 3, i)
  plt.text(0.5, 0.5, str((2, 3, i)), fontsize=18, ha='center')
```



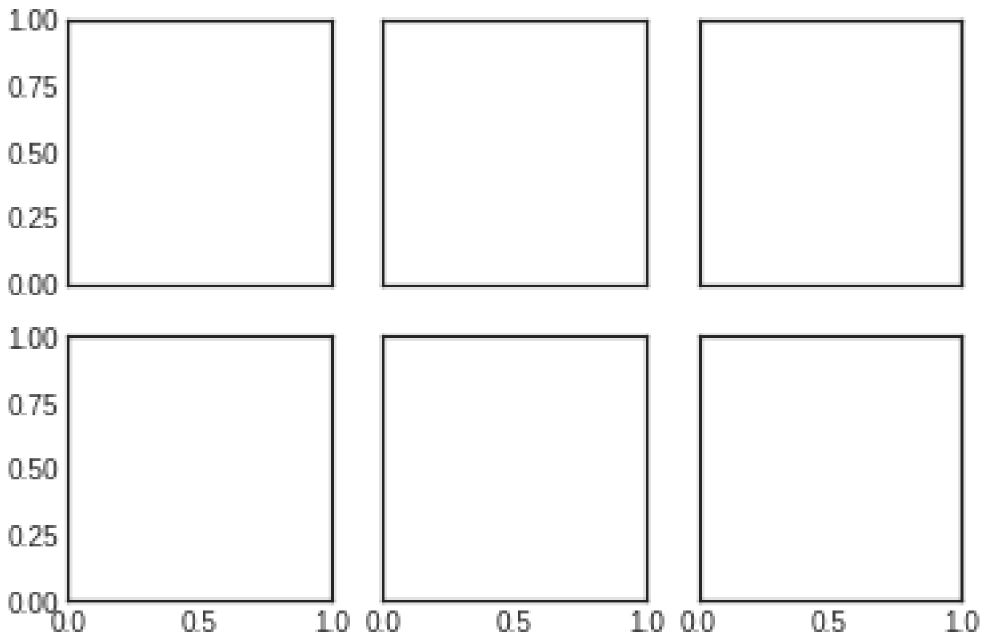
10.3 `plt.subplots`: The whole grid in one go

`plt.subplots()` is the easier tool to use that creates a full grid of subplots in a single line, returning them in a Numpy array.

The arguments are: * number of rows * number of columns * optional: `sharex` and `sharey`, which allow you to specify the relationship between different axes.

Let's create a 2x3 grid of subplots, where all axes in the same row share their y-axis scale, and all axes in the same column share their x-axis scale.

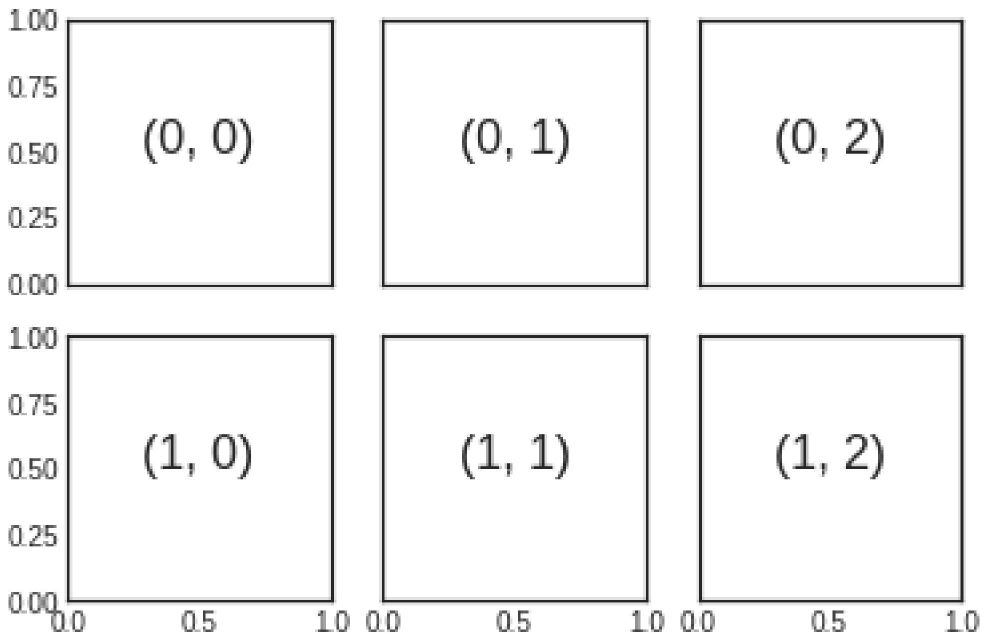
```
[ ]: fig, ax = plt.subplots(2, 3, sharex='col', sharey='row')
```



The resulting grid of axes instances is returned within a NumPy array, which allows for convenient specification of the desired axes using standard array indexing notation.

[ ]:
```
# axes are in a 2-dim array indexed by [row, col]
for i in range(2):
  for j in range(3):
    ax[i, j].text(0.5, 0.5, str((i, j)), fontsize=18, ha='center')

fig
```

[ ]:



10.4 `plt.GridSpec`: More complicated arrangements

The `plt.GridSpec()` object does not create a plot by itself. It is simply a convenient interface that is recognized by the `plt.subplot()` command.

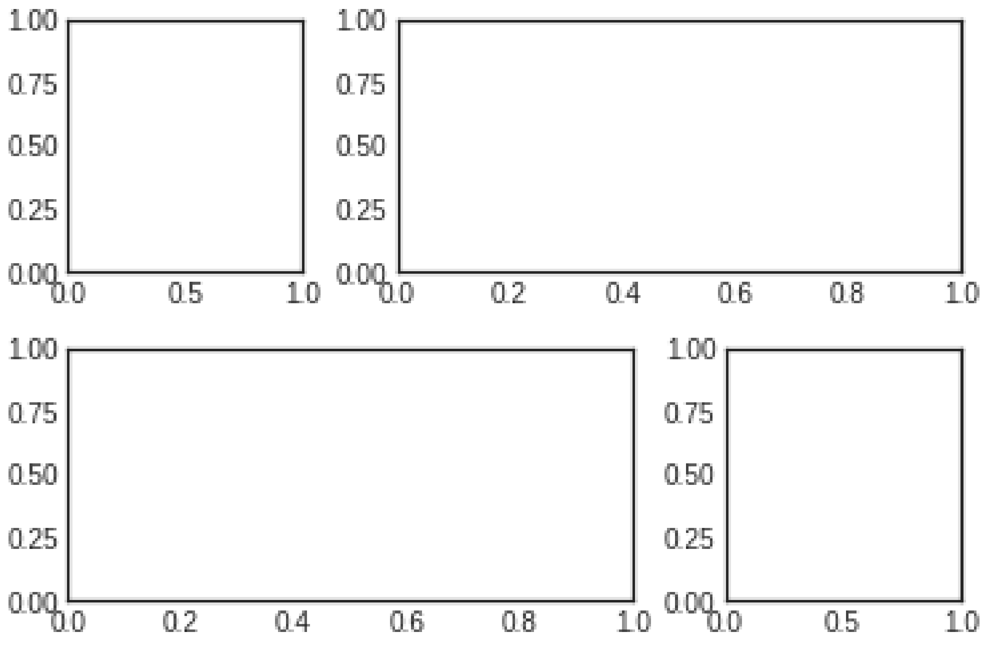
Let's create a gridspec for a grid of two rows and three columns with some specified width and height.

```
[ ]: grid = plt.GridSpec(2, 3, wspace=0.4, hspace=0.3)
```

Now, we can specify subplot locations and extents using slicing.

```
[ ]: plt.subplot(grid[0, 0])
     plt.subplot(grid[0, 1:])
     plt.subplot(grid[1, :2])
     plt.subplot(grid[1, 2])
```

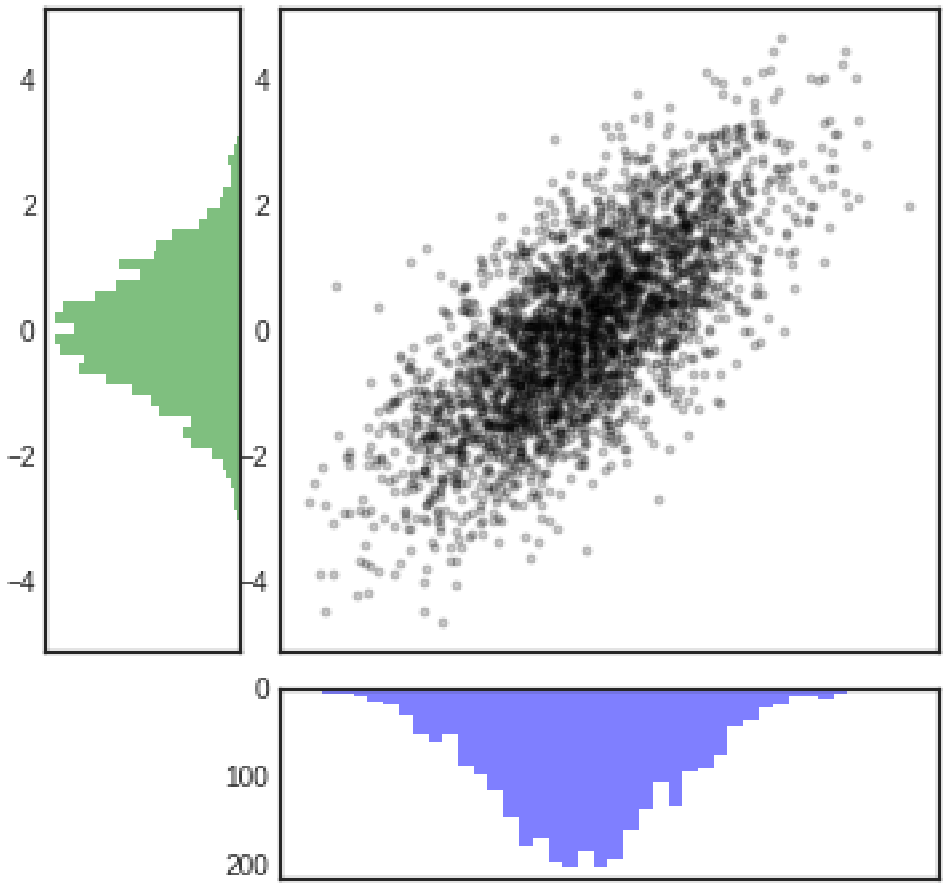
```
[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f7cf052ff10>
```



We can use grid alignment for creating multi-axes histogram plot as shown below.

[ ]:
```
# create some normally distributed data
mean = [0, 0]
cov = [[1, 1],
       [1, 2]]

x, y = np.random.multivariate_normal(mean, cov, 3000).T

# set up the axes with gridspec
fig = plt.figure(figsize=(6, 6))
grid = plt.GridSpec(4, 4, hspace=0.2, wspace=0.2)

main_ax = fig.add_subplot(grid[:-1, 1:])

y_hist = fig.add_subplot(grid[:-1, 0], xticklabels=[], sharey=main_ax)
x_hist = fig.add_subplot(grid[-1:, 1:], xticklabels=[], sharex=main_ax)

# scatter points on the main axes
main_ax.plot(x, y, 'ok', markersize=3, alpha=0.2)

# histogram on the attached axes
x_hist.hist(x, 40, histtype='stepfilled', orientation='vertical', color='blue',␣
 ↪alpha=0.5)
x_hist.invert_yaxis()
```

```
y_hist.hist(x, 40, histtype='stepfilled', orientation='horizontal',␣
 ↪color='green', alpha=0.5)
y_hist.invert_xaxis()
```



11 Visualization with Seaborn

Seaborn provides an API on top of Matplotlib that offers * choices for plot style and color defaults, * defines simple high-level functions for common statistical plot types, and * integrates with the functionality provided by Pandas `DataFrames`.

11.1 Seaborn vs. Matplotlib

Let's look at an example of a simple random-walk plot in Matplotlib, using its classic plot formatting and colors.

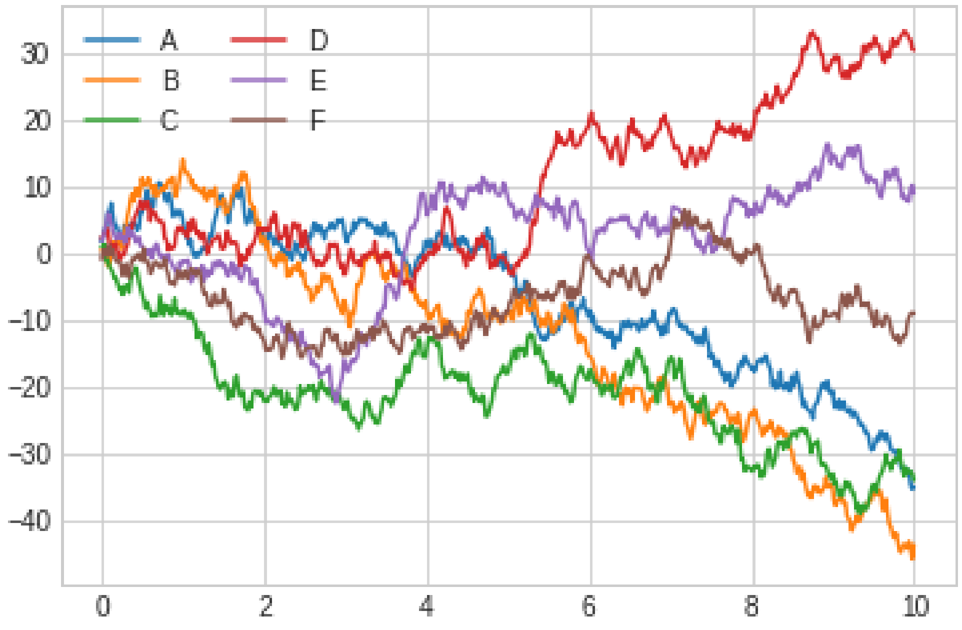
[ ]:
```
import matplotlib.pyplot as plt
plt.style.use('classic')

%matplotlib inline
```

```
import numpy as np
import pandas as pd
```

```
[ ]: # create some data
rng = np.random.RandomState(0)
x = np.linspace(0, 10, 500)
y = np.cumsum(rng.randn(500, 6), 0)
```

```
[ ]: # plot the data with Matplotlib defaults
plt.plot(x, y)
plt.legend('ABCDEF', ncol=2, loc='upper left')
```

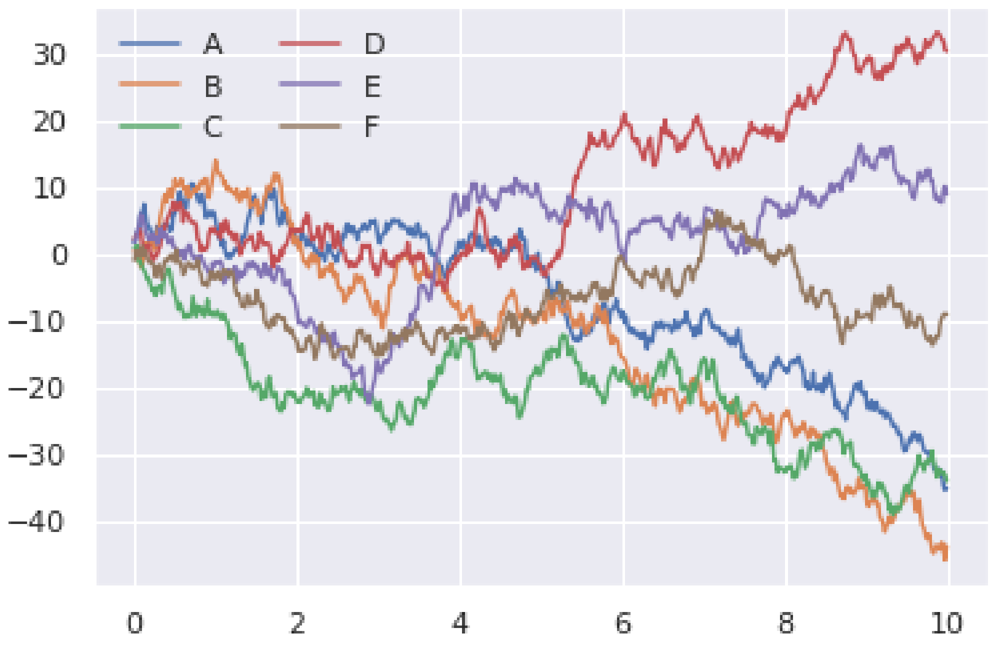
```
[ ]: <matplotlib.legend.Legend at 0x7f7cf0412a90>
```



Now, let's look at how Seaborn visualization works.

```
[ ]: import seaborn as sns
sns.set()
```

```
[ ]: # same plotting code as above
plt.plot(x, y)
plt.legend('ABCDEF', ncol=2, loc='upper left');
```



As we can see, Seaborn visualization is more visually appealing.

11.2 Exploring Seaborn Plots

Seaborn provides high-level commands to create a variey to plot types useful for data exploration, and even some statistical model fitting.

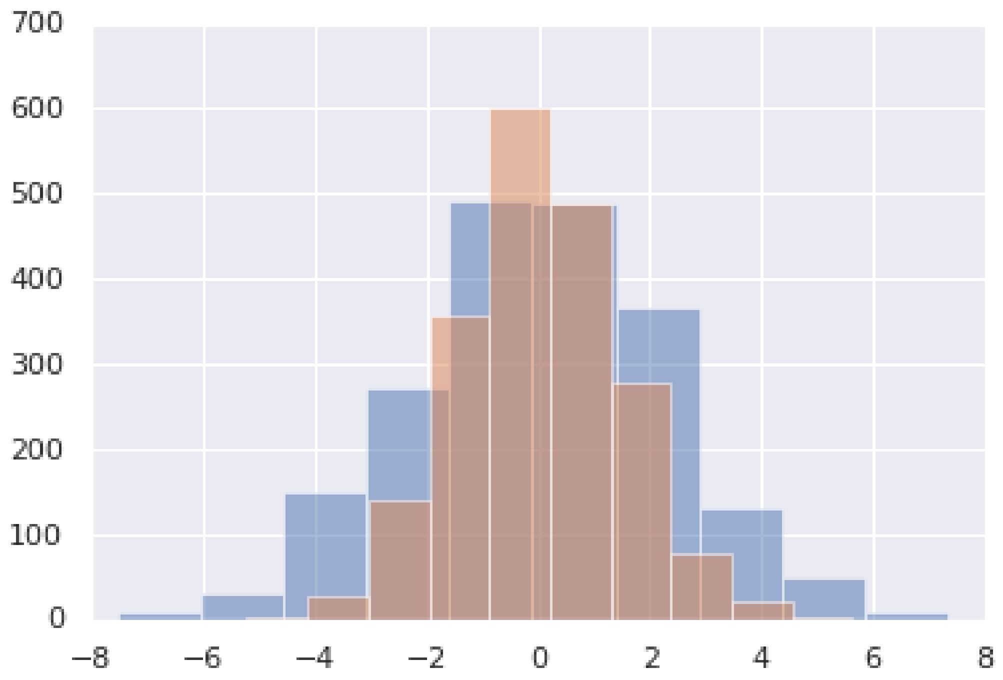
Let's look at some datasets and plot types available in Seaborn.

Histogram, KDE, and densities

In statistical data visualization, we often want to plot histograms and joint distributions of variables.

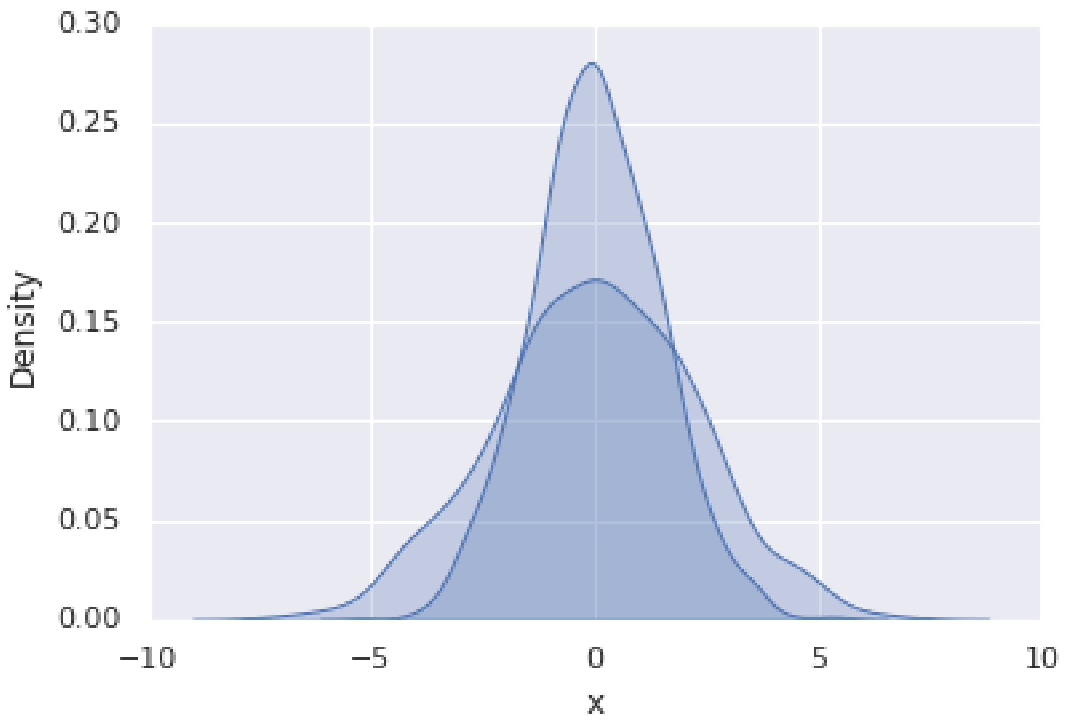
```
[ ]: import seaborn as sns
     sns.set()
```

```
[ ]: data = np.random.multivariate_normal([0, 0], [[5, 2], [2, 2]], size=2000)
     data = pd.DataFrame(data, columns=['x', 'y'])

     for col in 'xy':
       plt.hist(data[col], alpha=0.5)
```



We can get a smooth estimate of the distribution using a kernel density estimation (KDE), which Seaborn does with `sns.kdeplot`.

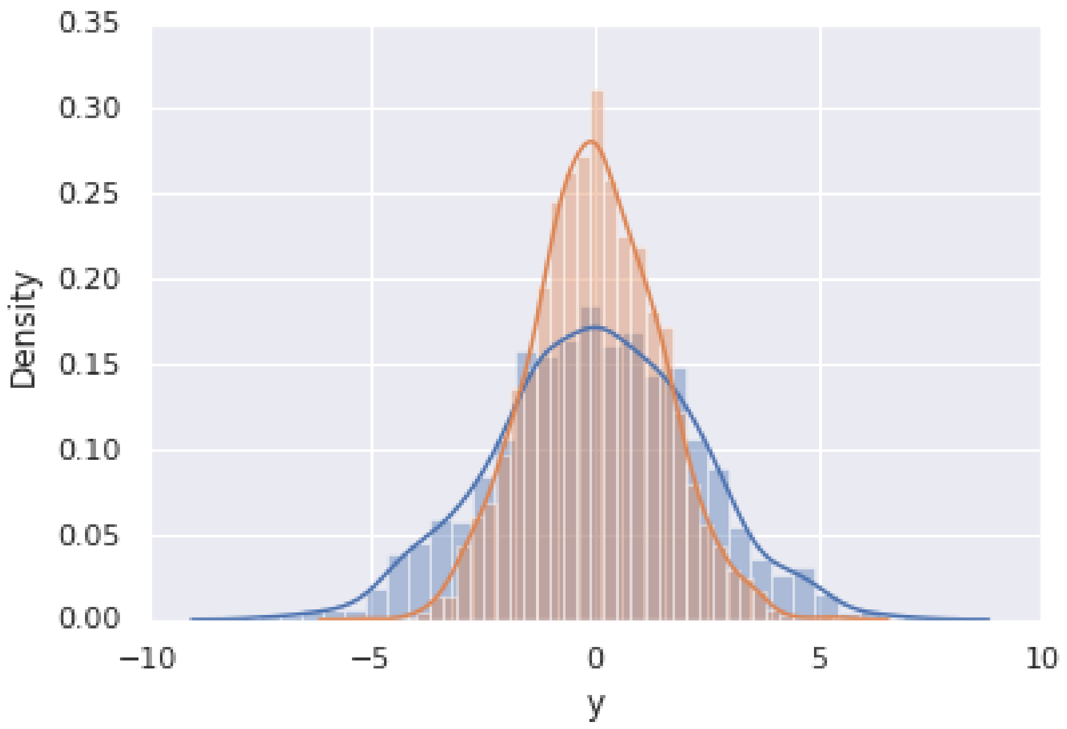
```
[ ]: for col in 'xy':
       sns.kdeplot(data[col], shade=True)
```



Histograms and KDE plots can be combined using `distplot`

```
[ ]: sns.distplot(data['x'])
     sns.distplot(data['y']);
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```



Pair Plots

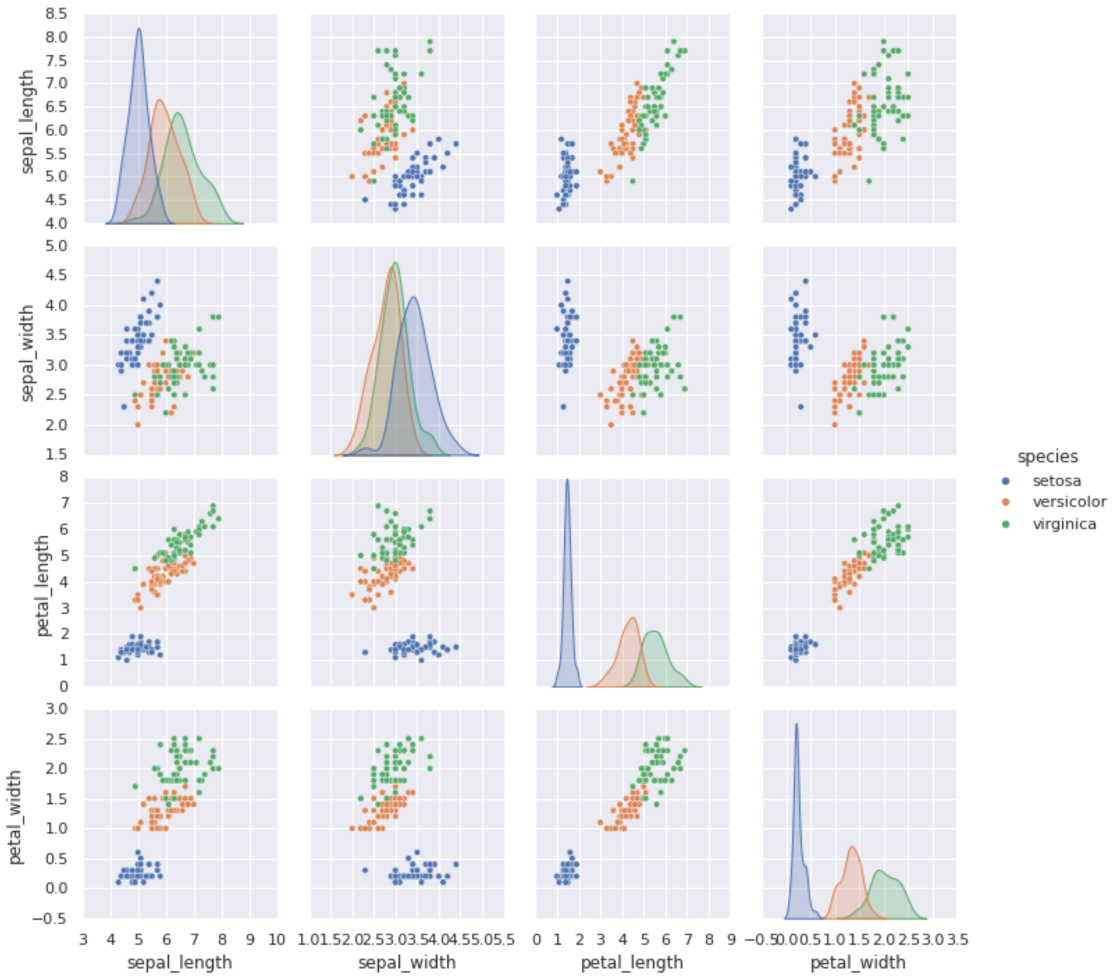
Pair plots are very useful for exploring correlations between multidimensional data, when you want to plot all pair of values against each other.

```
[ ]: iris = sns.load_dataset('iris')
     iris.head()
```

```
[ ]:    sepal_length  sepal_width  petal_length  petal_width species
     0           5.1          3.5           1.4          0.2  setosa
     1           4.9          3.0           1.4          0.2  setosa
     2           4.7          3.2           1.3          0.2  setosa
     3           4.6          3.1           1.5          0.2  setosa
     4           5.0          3.6           1.4          0.2  setosa
```

Let's visualize the multidimensional relationships among the samples.

```
[ ]: sns.pairplot(iris, hue='species', height=2.5);
```



Faceted histograms

Sometimes the best way to view data is via histograms of subsets. Seaborn's `FaceGrid` makes it very simple.

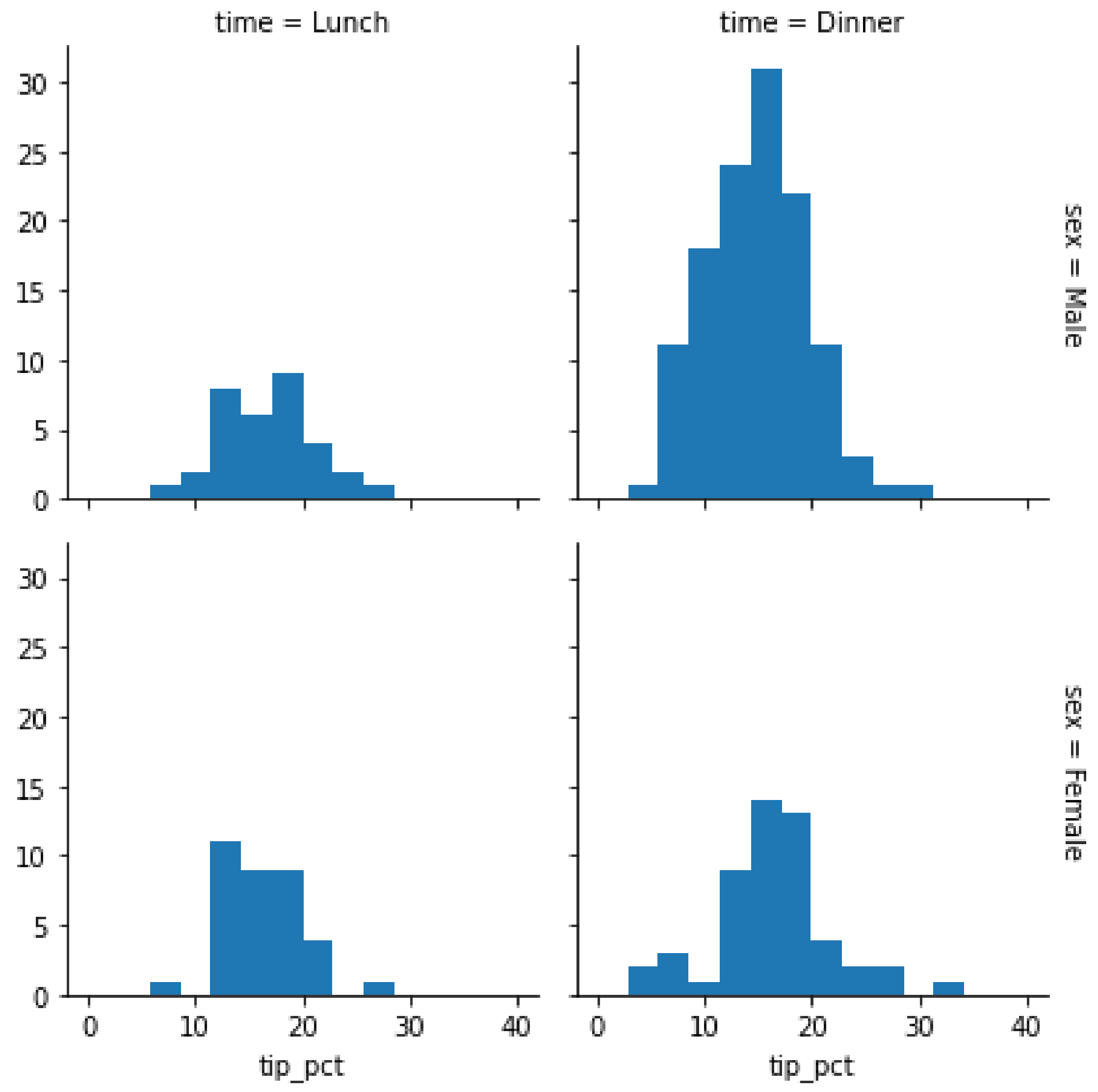
Let's take a look at some data that shows the amount that restaurant staff receive in tips based on various indicator data.

```
[ ]: import matplotlib.pyplot as plt
     import numpy as np
     import seaborn as sns

     tips = sns.load_dataset('tips')
     tips.head()
```

```
[ ]:    total_bill   tip     sex smoker  day    time  size
     0       16.99  1.01  Female     No  Sun  Dinner     2
     1       10.34  1.66    Male     No  Sun  Dinner     3
```

```
2        21.01  3.50    Male     No  Sun  Dinner     3
3        23.68  3.31    Male     No  Sun  Dinner     2
4        24.59  3.61  Female     No  Sun  Dinner     4
```

[ ]:
```
tips['tip_pct'] = 100 * tips['tip'] / tips['total_bill']

grid = sns.FacetGrid(tips, row='sex', col='time', margin_titles=True)
grid.map(plt.hist, "tip_pct", bins=np.linspace(0, 40, 15));
```



Bar plots

Time series can be plotted with `sns.factorplot`.

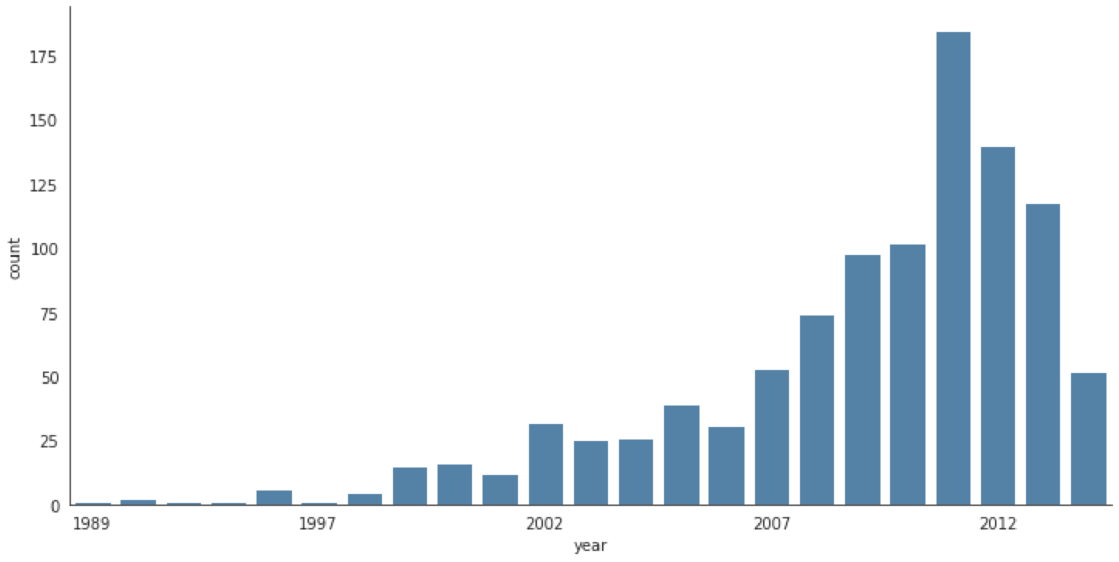
Let's visualize the Planets dataset.

```
[ ]: planets = sns.load_dataset('planets')
     planets.head()
```

```
[ ]:             method  number  orbital_period   mass  distance  year
     0  Radial Velocity       1         269.300   7.10     77.40  2006
     1  Radial Velocity       1         874.774   2.21     56.95  2008
     2  Radial Velocity       1         763.000   2.60     19.84  2011
     3  Radial Velocity       1         326.030  19.40    110.62  2007
     4  Radial Velocity       1         516.220  10.50    119.47  2009
```

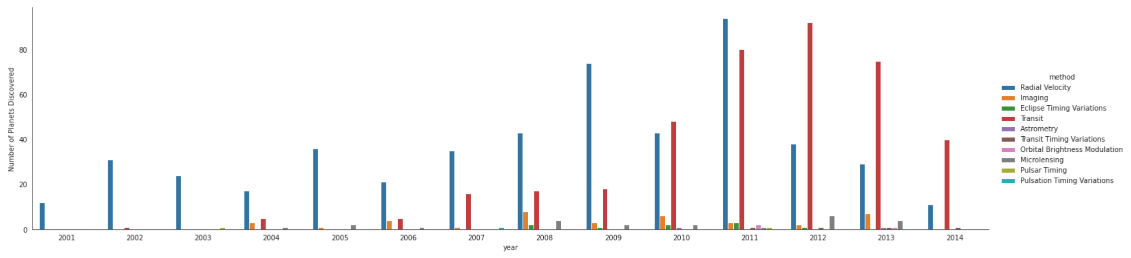
```
[ ]: with sns.axes_style('white'):
       g = sns.factorplot('year', data=planets, aspect=2, kind='count',␣
      ↪color='steelblue')
       g.set_xticklabels(step=5)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/categorical.py:3714: UserWarning:
The `factorplot` function has been renamed to `catplot`. The original name will
be removed in a future release. Please update your code. Note that the default
`kind` in `factorplot` (`'point'`) has changed `'strip'` in `catplot`.
  warnings.warn(msg)
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning:
Pass the following variable as a keyword arg: x. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without an
explicit keyword will result in an error or misinterpretation.
  FutureWarning
```



We can learn more by looking at the method of discovery of each of these planets as shown below.

```
[ ]: with sns.axes_style('white'):
       g = sns.catplot(x='year', data=planets, aspect=4.0, kind='count',␣
      ↪hue='method', order=range(2001, 2015))
       g.set_ylabels('Number of Planets Discovered')
```


Number of Planets Discovered
year
method
Radial Velocity
Imaging
Eclipse Timing Variations
Transit
Astrometry
Transit Timing Variations
Orbital Brightness Modulation
Microlensing
Pulsar Timing
Pulsation Timing Variations