Matplotlib Tutorial: 3. Useful Plot Types

So far we have dealt with simple line and point plots using the `plot` command. There are a wide array of other plot types available in matplotlib; we'll explore a few of them here. An excellent reference for this is the Plotting Commands Summary in the matplotlib documentation. For a more visual summary of available routines, see the Gallery Page.

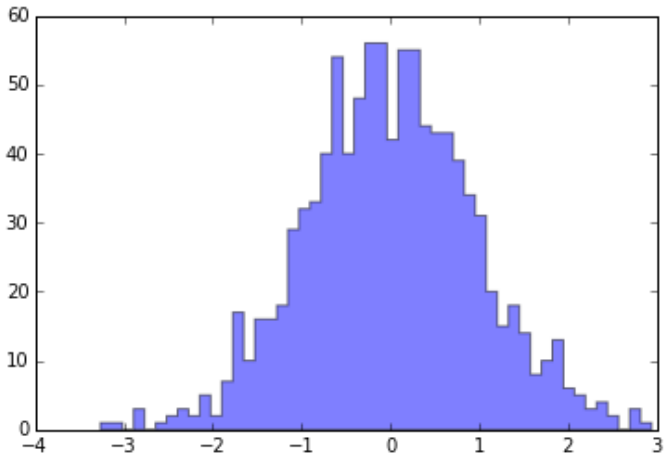
As before, we'll start by entering the matplotlib inline mode & doing some imports:

In [1]:
```
%matplotlib inline

from __future__ import print_function, division
import numpy as np
import matplotlib.pyplot as plt
```

Histograms

Histograms can be used to judge the density of 1-dimensional data. For example:

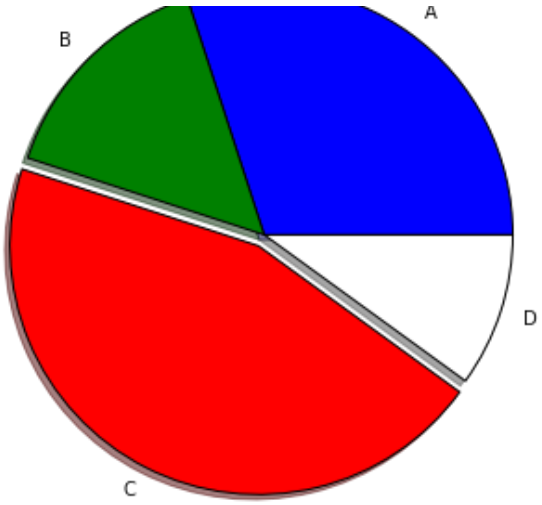
In [2]:
```
x = np.random.normal(size=1000)

fig, ax = plt.subplots()

H = ax.hist(x, bins=50, alpha=0.5, histtype='stepfilled')
```



Pie Plot

Matplotlib can create pie diagrams with the function `pie`:

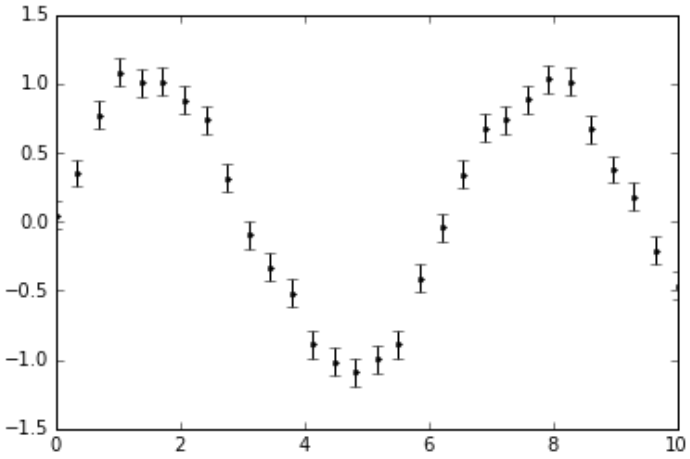
In [3]:
```
fracs = [30, 15, 45, 10]
colors = ['b', 'g', 'r', 'w']

fig, ax = plt.subplots(figsize=(6, 6))  # make the plot square
pie = ax.pie(fracs, colors=colors, explode=(0, 0, 0.05, 0), shadow=True,
             labels=['A', 'B', 'C', 'D'])
```



Errorbar Plots

Often we want to add errorbars to our points. The `errorbar` function works in a similar way to `plot`, but adds vertical and/or horizontal errorbars to the points.

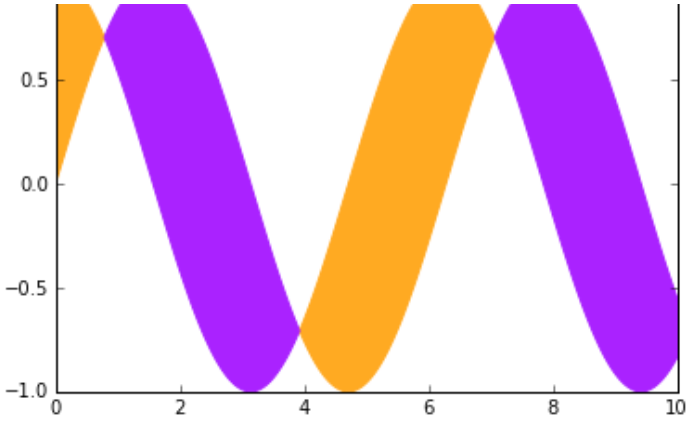
In [4]:
```
x = np.linspace(0, 10, 30)
dy = 0.1
y = np.random.normal(np.sin(x), dy)

fig, ax = plt.subplots()
plt.errorbar(x, y, dy, fmt='.k');
```



Filled Plots

Sometimes you'd like to fill the region below a curve, or between two curves. The functions `fill` and `fill_between` can be very useful for this:

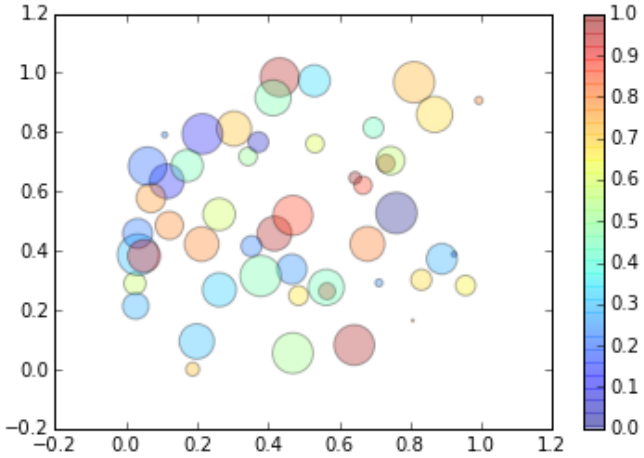
In [5]:
```
x = np.linspace(0, 10, 1000)
y1 = np.sin(x)
y2 = np.cos(x)

fig, ax = plt.subplots()
ax.fill_between(x, y1, y2, where=(y1 < y2), color='#FFAA22')
ax.fill_between(x, y1, y2, where=(y1 > y2), color='#AA22FF');
```



Scatter Plots

We have seen scatterplots before, when using point-type line styles in the `plot` command. The `scatter` command allows more flexibility in the colors and shapes of the points:

In [6]:
```
x = np.random.random(50)
y = np.random.random(50)
c = np.random.random(50)  # color of points
s = 500 * np.random.random(50)  # size of points

fig, ax = plt.subplots()
im = ax.scatter(x, y, c=c, s=s, cmap=plt.cm.jet, alpha=0.3)

# Add a colorbar
fig.colorbar(im, ax=ax)

# set the color limits - not necessary here, but good to know how.
im.set_clim(0.0, 1.0);
```

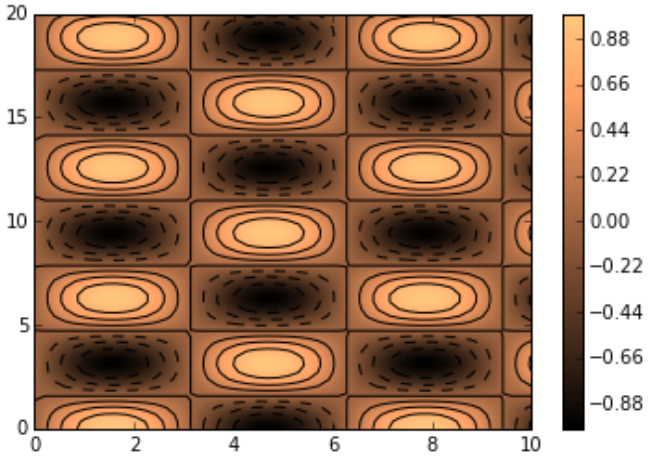


Contour Plots

Contour plots can be used to show the variation of a quantity with respect to two others:

In [7]:
```
x = np.linspace(0, 10, 50)
y = np.linspace(0, 20, 60)

z = np.cos(y[:, np.newaxis]) * np.sin(x)

fig, ax = plt.subplots()
```

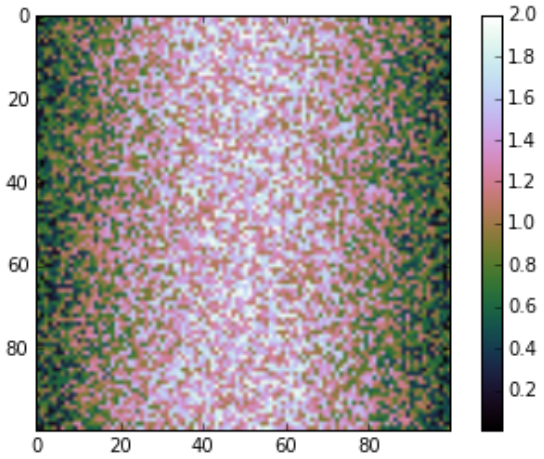
```
# filled contours
im = ax.contourf(x, y, z, 100, cmap=plt.cm.copper)

# contour lines
im2 = ax.contour(x, y, z, colors='k')

fig.colorbar(im, ax=ax);
```



Showing Images

The `imshow` command allows displaying images in a variety of formats. It can be useful for actual image data, as well as being useful for visualizing datasets in a way similar to the contour plots above.

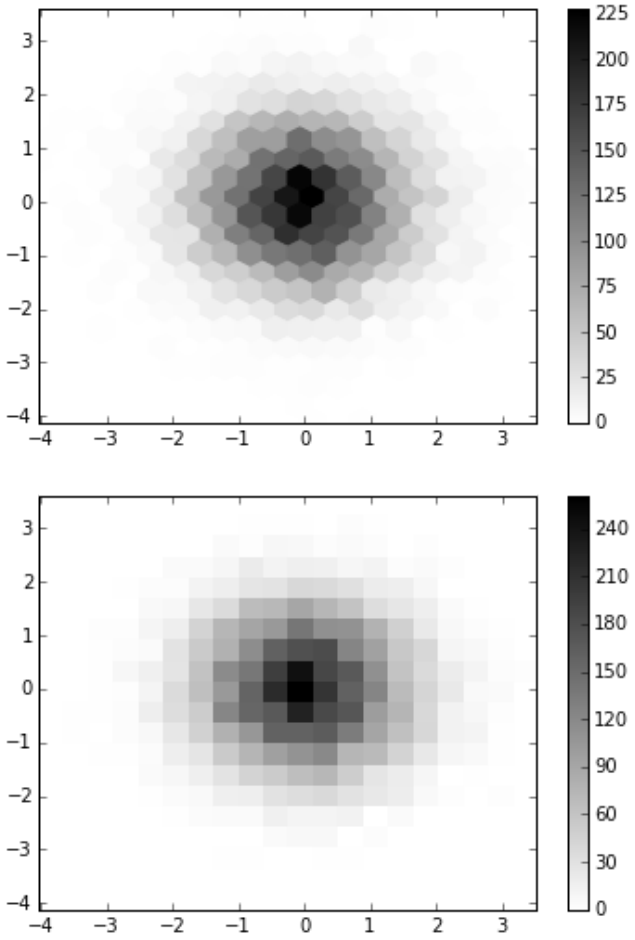
In [8]:
```
I = np.random.random((100, 100))

I += np.sin(np.linspace(0, np.pi, 100))

fig, ax = plt.subplots()

im = ax.imshow(I, cmap=plt.cm.cubehelix)

fig.colorbar(im, ax=ax);
```



2D Histograms and Hexbin

`hist2D` and `hexbin` are ways to represent binned two-dimensional data. They can be used as follows:

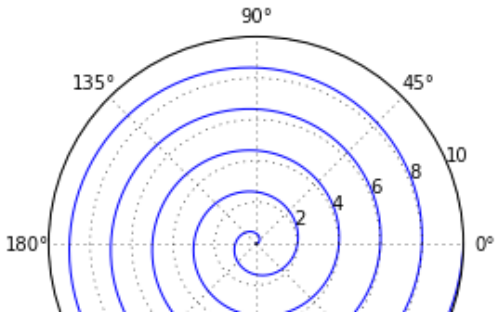
In [9]:
```
x, y = np.random.normal(size=(2, 10000))

fig, ax = plt.subplots()
im = ax.hexbin(x, y, gridsize=20, cmap='binary')
fig.colorbar(im, ax=ax)

fig, ax = plt.subplots()
H = ax.hist2d(x, y, bins=20, cmap='binary')
fig.colorbar(H[3], ax=ax);
```

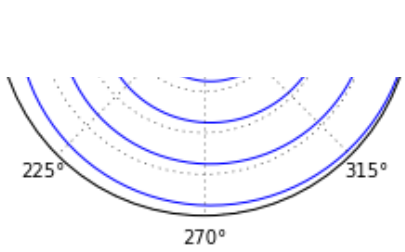


Polar Plots

It is also possible to plot data in coordinates other than Cartesian. Here we'll show how to do a polar plot

In [10]:
```
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1, projection='polar')

theta = np.linspace(0, 10 * np.pi, 1000)
r = np.linspace(0, 10, 1000)

ax.plot(theta, r);
```




225°
315°
270°