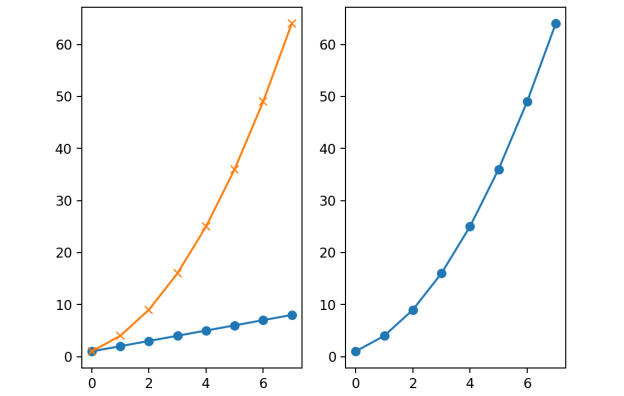
Subplots

In [1]:
```
%matplotlib notebook

import matplotlib.pyplot as plt
import numpy as np

plt.subplot?
```

In [2]:
```
plt.figure()
# subplot with 1 row, 2 columns, and current axis is 1st subplot axes
plt.subplot(1, 2, 1)

linear_data = np.array([1,2,3,4,5,6,7,8])

plt.plot(linear_data, '-o')
```

Figure 1



Out[2]: [<matplotlib.lines.Line2D at 0x7fcab58ae4e0>]

In [3]:
```
exponential_data = linear_data**2

# subplot with 1 row, 2 columns, and current axis is 2nd subplot axes
plt.subplot(1, 2, 2)
plt.plot(exponential_data, '-o')
```

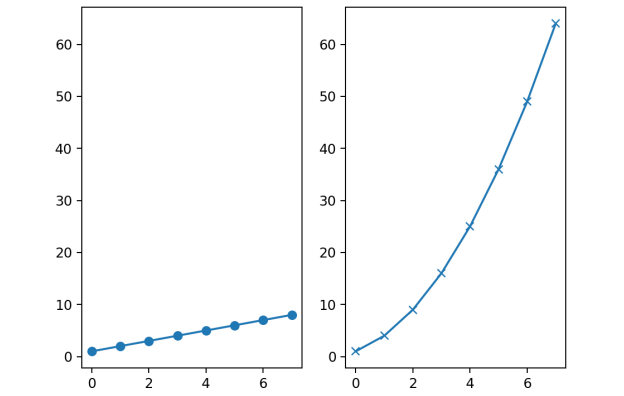
Out[3]: [<matplotlib.lines.Line2D at 0x7fcab58cbac8>]

In [4]:
```
# plot exponential data on 1st subplot axes
plt.subplot(1, 2, 1)
plt.plot(exponential_data, '-x')
```

Out[4]: [<matplotlib.lines.Line2D at 0x7fcab58cba20>]

In [5]:
```
plt.figure()
ax1 = plt.subplot(1, 2, 1)
plt.plot(linear_data, '-o')
# pass sharey=ax1 to ensure the two subplots share the same y axis
ax2 = plt.subplot(1, 2, 2, sharey=ax1)
plt.plot(exponential_data, '-x')
```

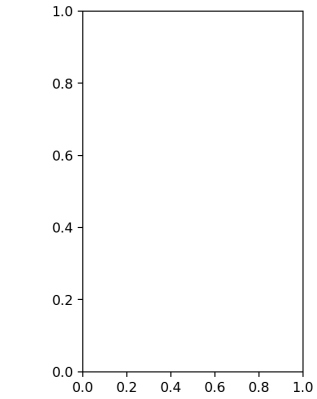
Figure 2



Out[5]: [<matplotlib.lines.Line2D at 0x7fcab44d1da0>]

In [6]:
```
plt.figure()
# the right hand side is equivalent shorthand syntax
plt.subplot(1,2,1) == plt.subplot(121)
```

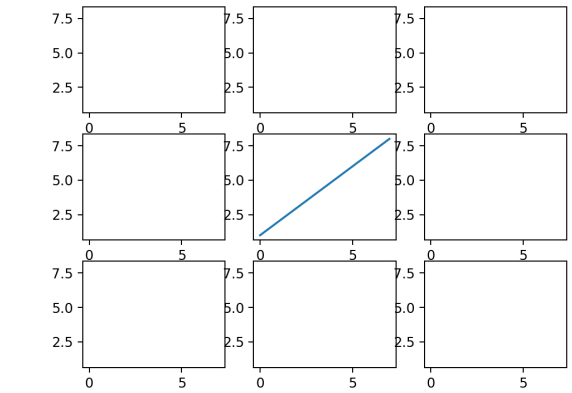
Figure 3



Out[6]: True

In [7]:
```
# create a 3x3 grid of subplots
fig, ((ax1,ax2,ax3), (ax4,ax5,ax6), (ax7,ax8,ax9)) = plt.subplots(3, 3, sharex=True, sharey=True)
# plot the linear_data on the 5th subplot axes
ax5.plot(linear_data, '-')
```

Figure 4



Out[7]: [<matplotlib.lines.Line2D at 0x7fcab44f7eb8>]

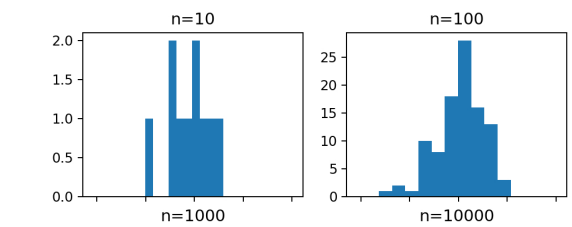
In [8]:
```
# set inside tick labels to visible
for ax in plt.gcf().get_axes():
    for label in ax.get_xticklabels() + ax.get_yticklabels():
        label.set_visible(True)
```

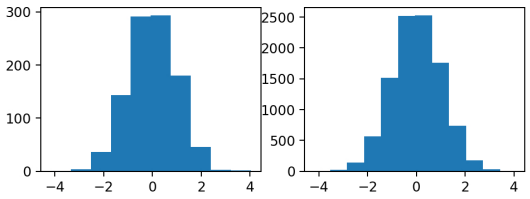
In [9]:
```
# necessary on some systems to update the plot
plt.gcf().canvas.draw()
```

Histograms

In [10]:
```
# create 2x2 grid of axis subplots
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, sharex=True)
axs = [ax1,ax2,ax3,ax4]

# draw n = 10, 100, 1000, and 10000 samples from the normal distribution and plot corresponding histograms
for n in range(0,len(axs)):
    sample_size = 10**(n+1)
    sample = np.random.normal(loc=0.0, scale=1.0, size=sample_size)
    axs[n].hist(sample)
    axs[n].set_title('n={}'.format(sample_size))
```

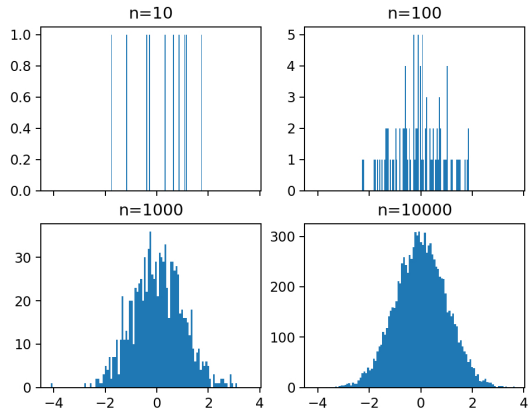
Figure 5





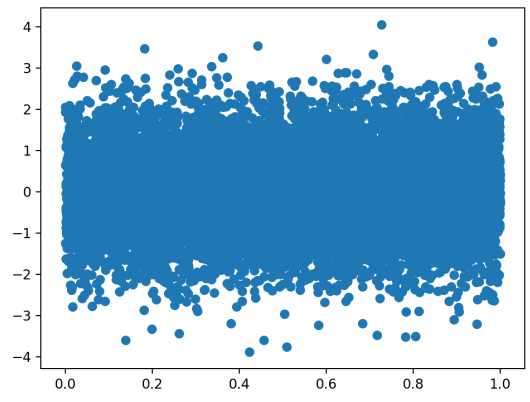
In [11]:
```
# repeat with number of bins set to 100
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, sharex=True)
axs = [ax1,ax2,ax3,ax4]

for n in range(0,len(axs)):
    sample_size = 10**(n+1)
    sample = np.random.normal(loc=0.0, scale=1.0, size=sample_size)
    axs[n].hist(sample, bins=100)
    axs[n].set_title('n={}'.format(sample_size))
```

Figure 6



In [12]:
```
plt.figure()
Y = np.random.normal(loc=0.0, scale=1.0, size=10000)
X = np.random.random(size=10000)
plt.scatter(X,Y)
```

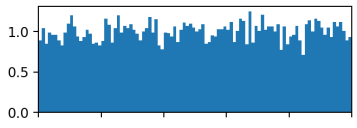
Figure 7

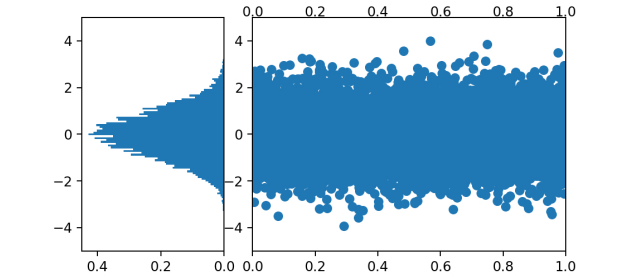


Out[12]: <matplotlib.collections.PathCollection at 0x7fca54470b70>

In [13]:
```
# use gridspec to partition the figure into subplots
import matplotlib.gridspec as gridspec

plt.figure()
gspec = gridspec.GridSpec(3, 3)

top_histogram = plt.subplot(gspec[0, 1:])
side_histogram = plt.subplot(gspec[1:, 0])
lower_right = plt.subplot(gspec[1:, 1:])
```

Figure 8

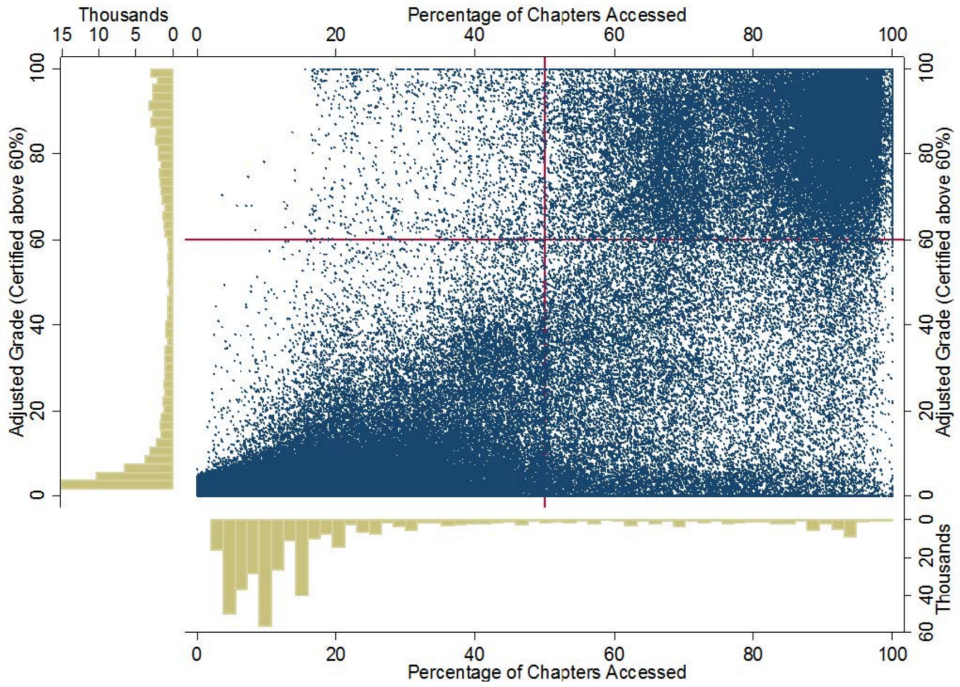




In [14]:
```
Y = np.random.normal(loc=0.0, scale=1.0, size=10000)
X = np.random.random(size=10000)
lower_right.scatter(X, Y)
top_histogram.hist(X, bins=100)
s = side_histogram.hist(Y, bins=100, orientation='horizontal')
```

In [15]:
```
# clear the histograms and plot normed histograms
top_histogram.clear()
top_histogram.hist(X, bins=100, normed=True)
side_histogram.clear()
side_histogram.hist(Y, bins=100, orientation='horizontal', normed=True)
# flip the side histogram's x axis
side_histogram.invert_xaxis()
```

In [16]:
```
# change axes limits
for ax in [top_histogram, lower_right]:
    ax.set_xlim(0, 1)
for ax in [side_histogram, lower_right]:
    ax.set_ylim(-5, 5)
```



Box and Whisker Plots

In [17]:
```
import pandas as pd
normal_sample = np.random.normal(loc=0.0, scale=1.0, size=10000)
random_sample = np.random.random(size=10000)
gamma_sample = np.random.gamma(2, size=10000)

df = pd.DataFrame({'normal': normal_sample,
                   'random': random_sample,
                   'gamma': gamma_sample})
```

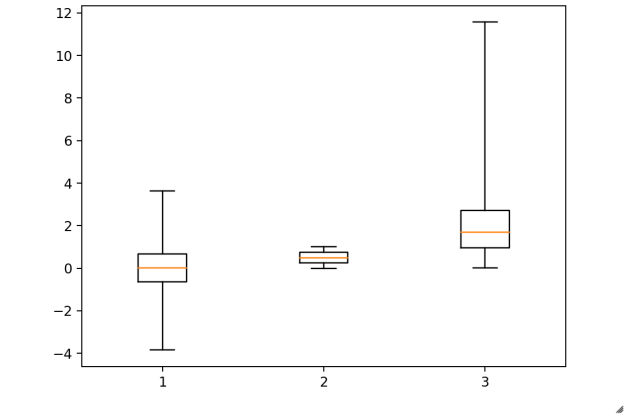
In [18]:
```
df.describe()
```

Out[18]:

	gamma	normal	random
count	10000.000000	10000.000000	10000.000000
mean	2.012147	0.006002	0.498710
std	1.417666	0.983814	0.290119
min	0.005748	-3.843864	0.000132
25%	0.960191	-0.648371	0.245849
50%	1.693698	0.011603	0.498361
75%	2.720589	0.672315	0.751416
max	11.571223	3.643972	0.999934

In [19]:
```
plt.figure()
# create a boxplot of the normal data, assign the output to a variable to supress output
_ = plt.boxplot(df['normal'], whis='range')
```

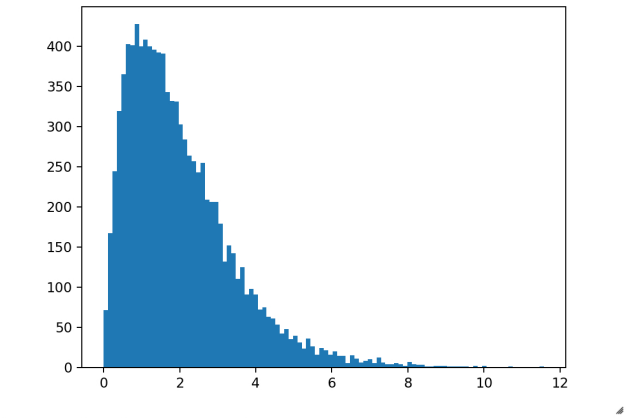
Figure 9



In [20]:
```
# clear the current figure
plt.clf()
# plot boxplots for all three of df's columns
_ = plt.boxplot([ df['normal'], df['random'], df['gamma'] ], whis='range')
```

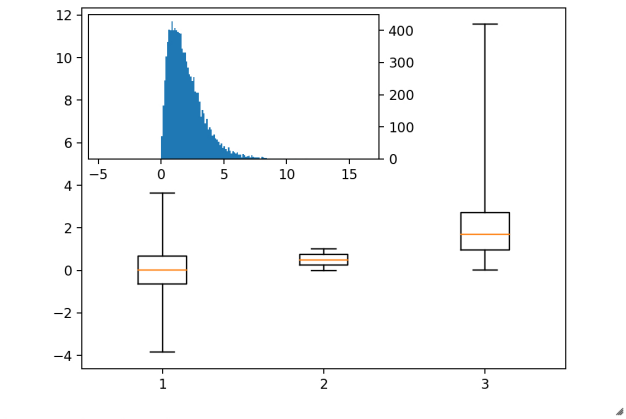
In [21]:
```
plt.figure()
_ = plt.hist(df['gamma'], bins=100)
```

Figure 10



In [22]:
```
import mpl_toolkits.axes_grid1.inset_locator as mpl_il

plt.figure()
plt.boxplot([ df['normal'], df['random'], df['gamma'] ], whis='range')
# overlay axis on top of another
ax2 = mpl_il.inset_axes(plt.gca(), width='60%', height='40%', loc=2)
ax2.hist(df['gamma'], bins=100)
ax2.margins(x=0.5)
```

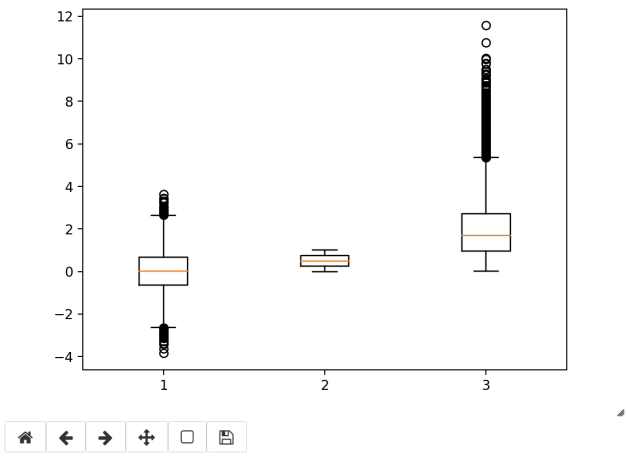
Figure 11



In [23]:
```
# switch the y axis ticks for ax2 to the right side
ax2.yaxis.tick_right()
```

In [24]:
```
# if `whis` argument isn't passed, boxplot defaults to showing 1.5*interquartile (IQR) whiskers with outliers
plt.figure()
_ = plt.boxplot([ df['normal'], df['random'], df['gamma'] ] )
```

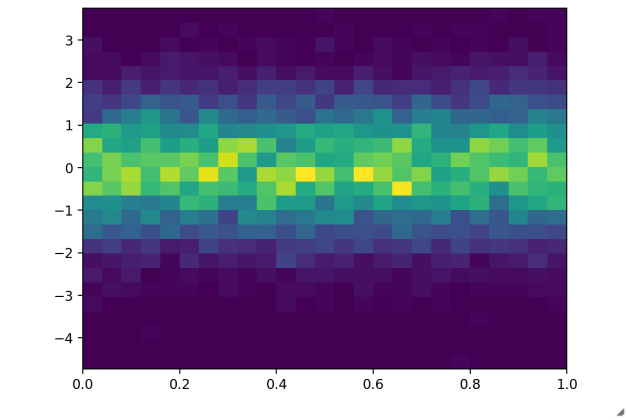
Figure 12



Heatmaps

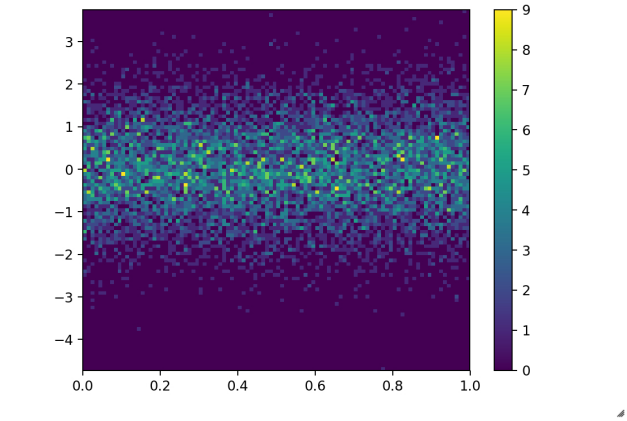
In [25]:
```
plt.figure()

Y = np.random.normal(loc=0.0, scale=1.0, size=10000)
X = np.random.random(size=10000)
_ = plt.hist2d(X, Y, bins=25)
```

Figure 13



In [26]:
```
plt.figure()
_ = plt.hist2d(X, Y, bins=100)
```

Figure 14



In [27]:
```
# add a colorbar legend
plt.colorbar()
```

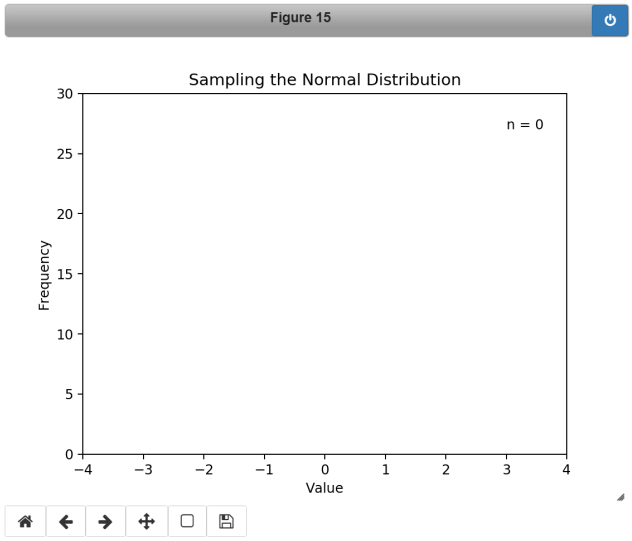
Out[27]:
```
<matplotlib.colorbar.Colorbar at 0x7fca45ff1ba8>
```

Animations

In [28]:
```
import matplotlib.animation as animation

n = 100
x = np.random.randn(n)
```

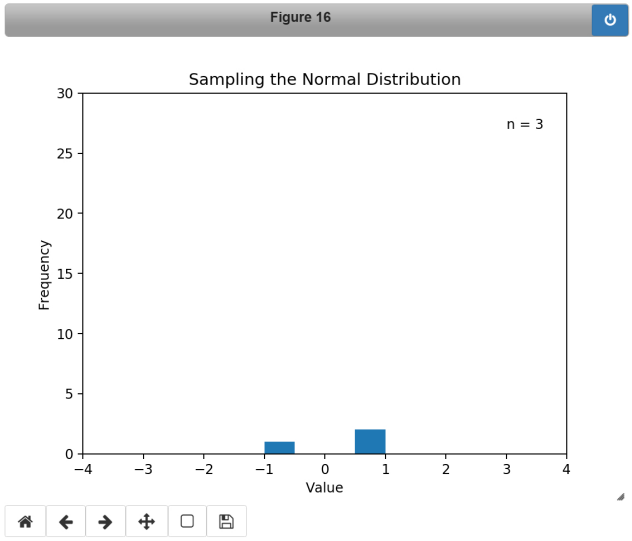
In [29]:
```
# create the function that will do the plotting, where curr is the current frame
def update(curr):
    # check if animation is at the last frame, and if so, stop the animation a
    if curr == n:
        a.event_source.stop()
    plt.cla()
    bins = np.arange(-4, 4, 0.5)
    plt.hist(x[:curr], bins=bins)
    plt.axis([-4,4,0,30])
    plt.gca().set_title('Sampling the Normal Distribution')
    plt.gca().set_ylabel('Frequency')
    plt.gca().set_xlabel('Value')
    plt.annotate('n = {}'.format(curr), [3,27])
```

In [30]:
```
fig = plt.figure()
a = animation.FuncAnimation(fig, update, interval=100)
```



Interactivity

In [31]:
```
plt.figure()
data = np.random.rand(10)
plt.plot(data)

def onclick(event):
    plt.cla()
    plt.plot(data)
    plt.gca().set_title('Event at pixels {},{} \nand data {},{}'.format(event.x, event.y, event.xdata, event.ydata))

# tell mpl_connect we want to pass a 'button_press_event' into onclick when the event is detected
plt.gcf().canvas.mpl_connect('button_press_event', onclick)
```



Out[31]: 7

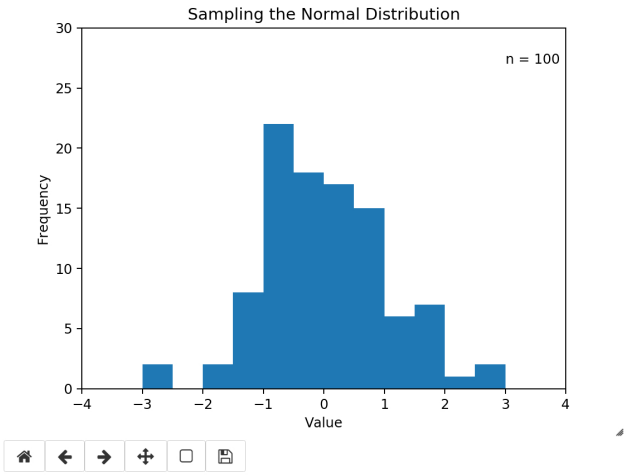
In [32]:
```
from random import shuffle
origins = ['China', 'Brazil', 'India', 'USA', 'Canada', 'UK', 'Germany', 'Iraq', 'Chile', 'Mexico']

shuffle(origins)

df = pd.DataFrame({'height': np.random.rand(10),
                   'weight': np.random.rand(10),
                   'origin': origins})
df
```

Out[32]:

	height	origin	weight
0	0.070259	China	0.300175
1	0.155428	India	0.060695
2	0.858719	Germany	0.765097
3	0.525608	Brazil	0.713885
4	0.641282	Mexico	0.448949
5	0.751197	UK	0.537409
6	0.848622	Iraq	0.954888
7	0.956217	Canada	0.201536
8	0.016385	USA	0.975615
9	0.510486	Chile	0.470568

In [33]:
```
plt.figure()
# picker=5 means the mouse doesn't have to click directly on an event, but can be up to 5 pixels away
plt.scatter(df['height'], df['weight'], picker=5)
plt.gca().set_ylabel('Weight')
plt.gca().set_xlabel('Height')
```

Figure 17



Out[33]: <matplotlib.text.Text at 0x7fca8386d2e8>

In [34]:
```
def onpick(event):
    origin = df.iloc[event.ind[0]]['origin']
    plt.gca().set_title('Selected item came from {}'.format(origin))

# tell mpl_connect we want to pass a 'pick_event' into onpick when the event is detected
plt.gcf().canvas.mpl_connect('pick_event', onpick)
```

Out[34]: 7

In [ ]: