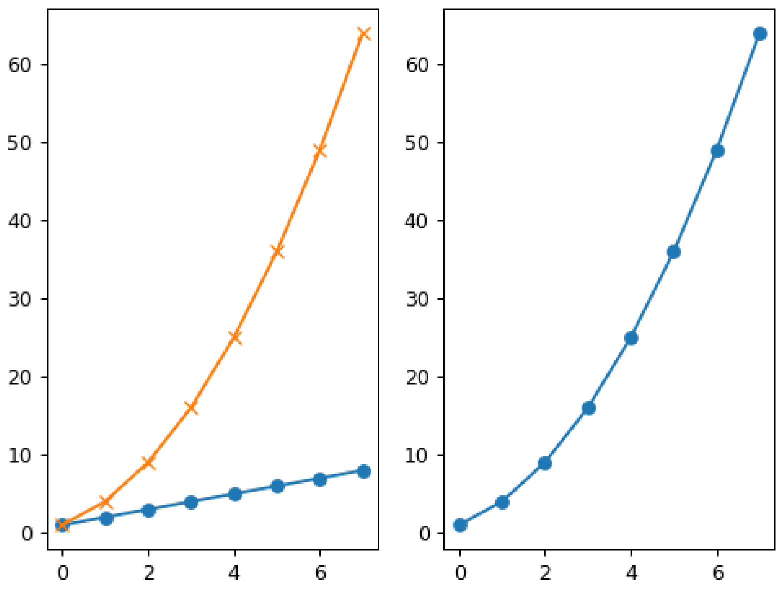
Subplots

```
%matplotlib notebook

import matplotlib.pyplot as plt
import numpy as np

# plt.subplot?
```

```
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')
```

```
<IPython.core.display.Javascript object>
```




```
[<matplotlib.lines.Line2D at 0x7fdd683cf630>]
```

```
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')
```

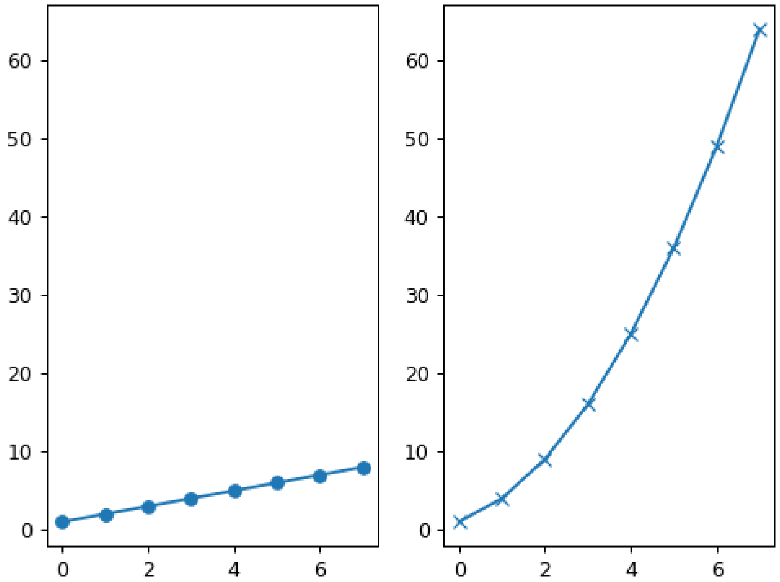
```
[<matplotlib.lines.Line2D at 0x7fdd660f43c8>]
```

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

```
[<matplotlib.lines.Line2D at 0x7fdd660c6390>]
```

```
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')
```

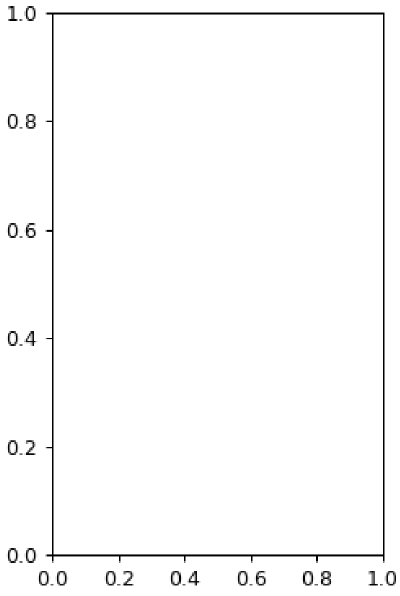
```
<IPython.core.display.Javascript object>
```




```
[<matplotlib.lines.Line2D at 0x7fdd65fff1d0>]
```

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

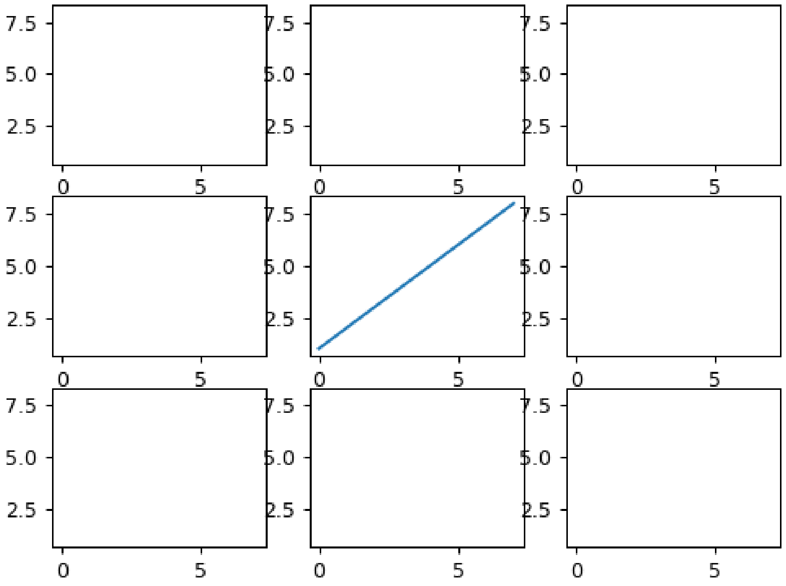
```
<IPython.core.display.Javascript object>
```




```
True
```

```python
# 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, '-')
```

```
<IPython.core.display.Javascript object>
```




```
[<matplotlib.lines.Line2D at 0x7fdd65f03208>]
```

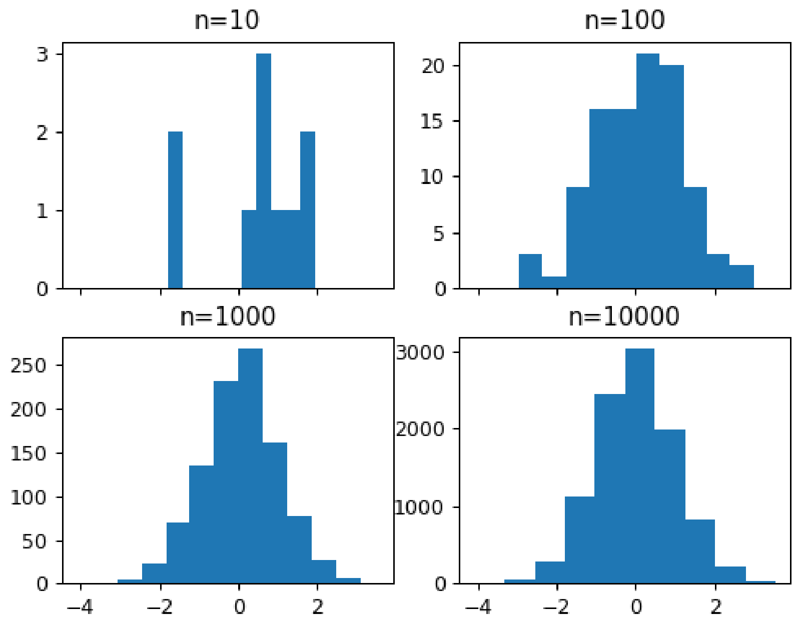
```python
# set inside tick labels to visible
for ax in plt.gcf().get_axes():
    for label in ax.get_xticklabels() + ax.get_yticklabels():
        label.set_visible(True)
```

```
# necessary on some systems to update the plot
plt.gcf().canvas.draw()
```

Histograms

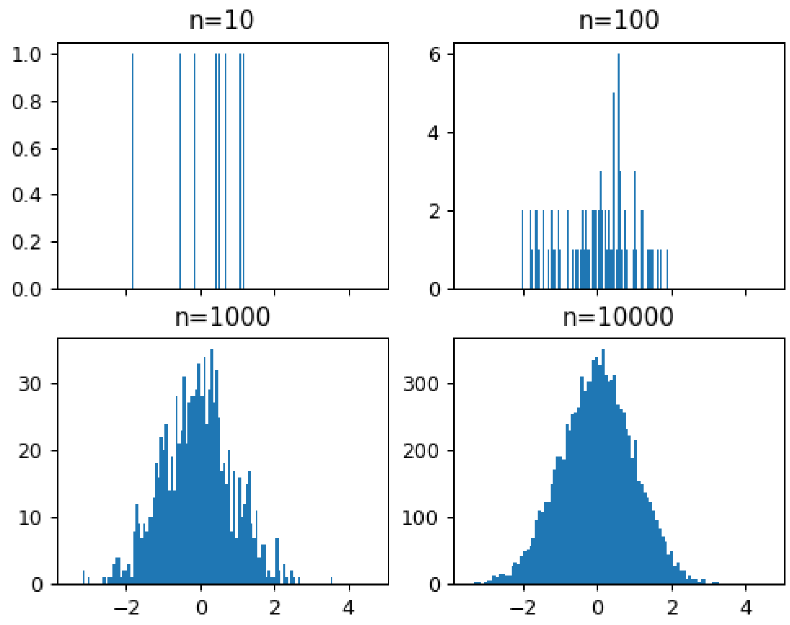
```
# create 2x2 grid of axis subplots
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, sharex=True)
axs = [ax1,ax2,ax3,ax4]

# 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))
```

```
<IPython.core.display.Javascript object>
```



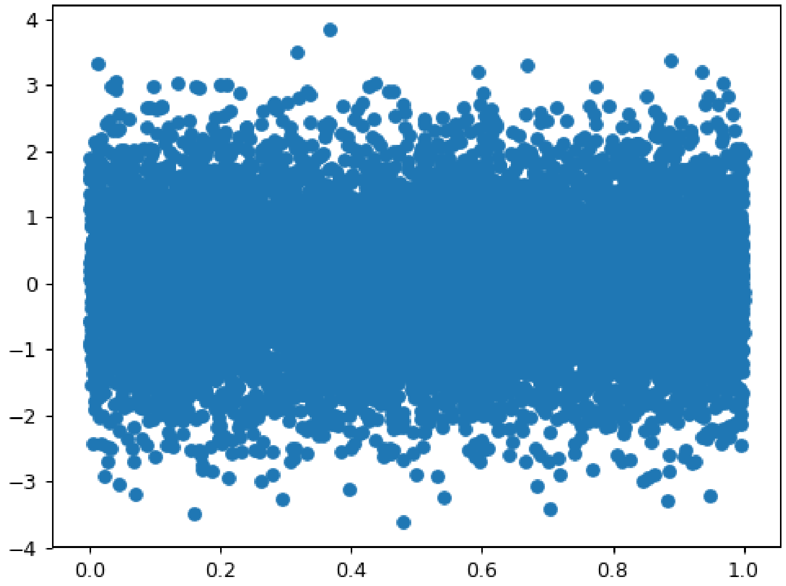

```
# repeat with number of bins set to 100
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, sharex=True)
axs = [ax1,ax2,ax3,ax4]

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))
```

```
<IPython.core.display.Javascript object>
```




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

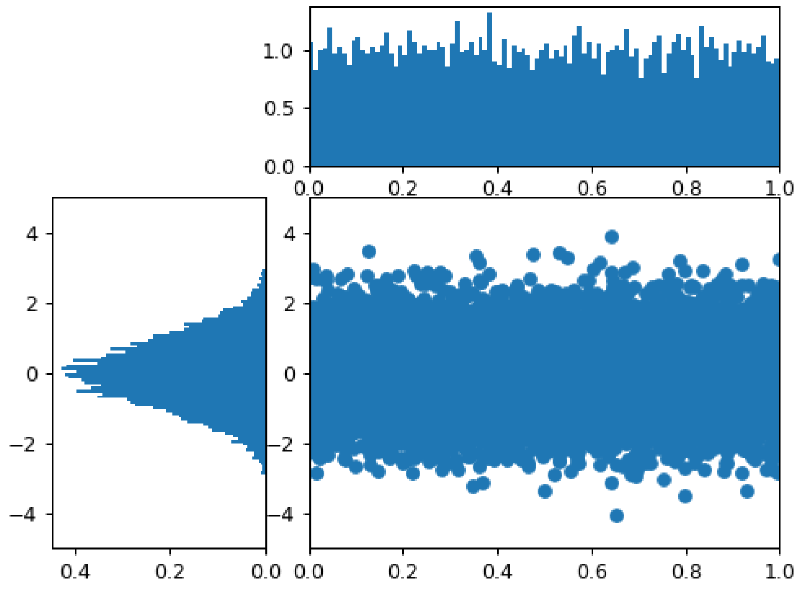
```
<IPython.core.display.Javascript object>
```



```
<matplotlib.collections.PathCollection at 0x7fdd64dcf1d0>
```

```
# 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:])
```

<IPython.core.display.Javascript object>




```
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')
```

```
# 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()
```

```
# 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)
```

MOOC DATA

Box and Whisker Plots

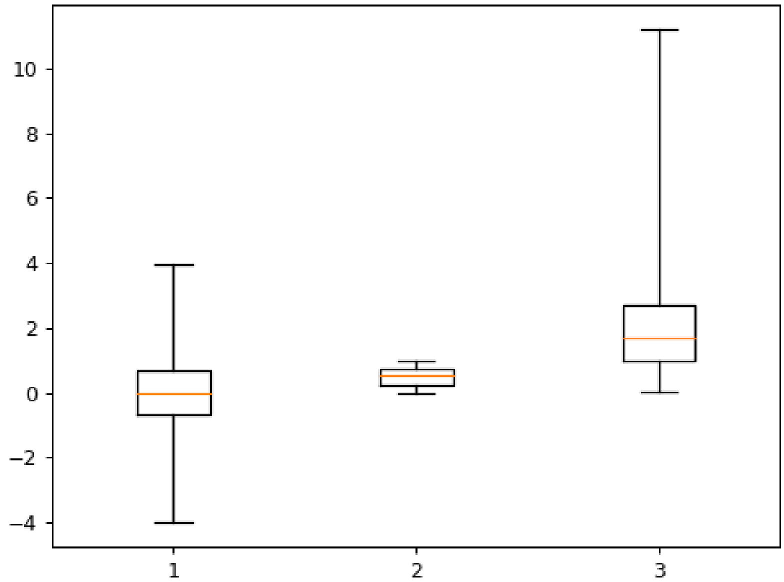
```
import pandas as pd
normal_sample = np.random.normal(loc=0.0, scale=1.0, size=10000)
random_sample = np.random.random(size=10000)
gamma_sample = np.random.gamma(2, size=10000)

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

```
df.describe()
```

	gamma	normal	random
count	10000.000000	10000.000000	10000.000000
mean	2.013573	-0.001622	0.505779
std	1.428351	1.006206	0.288896
min	0.010999	-4.012801	0.000023
25%	0.983298	-0.662282	0.256625
50%	1.679789	0.001882	0.511022
75%	2.694919	0.668334	0.754551
max	11.202994	3.963881	0.999993

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

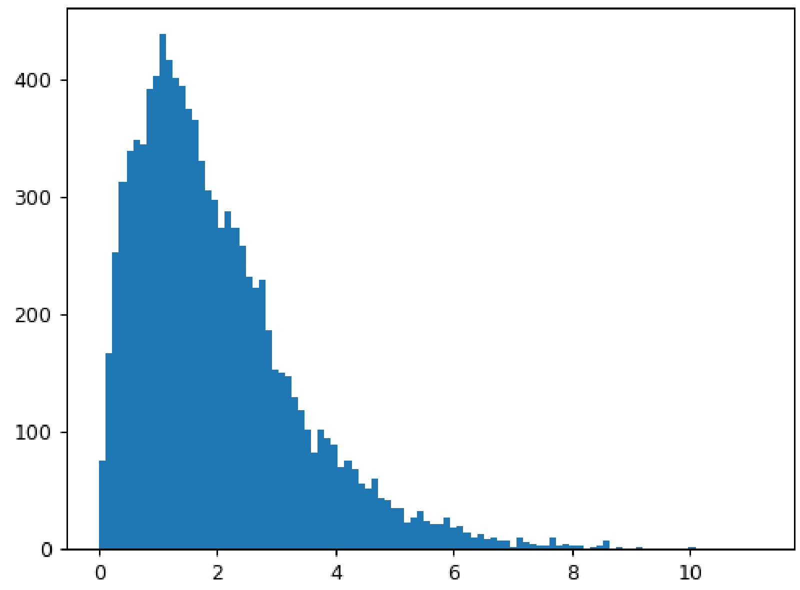
```
<IPython.core.display.Javascript object>
```




```
# 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')
```

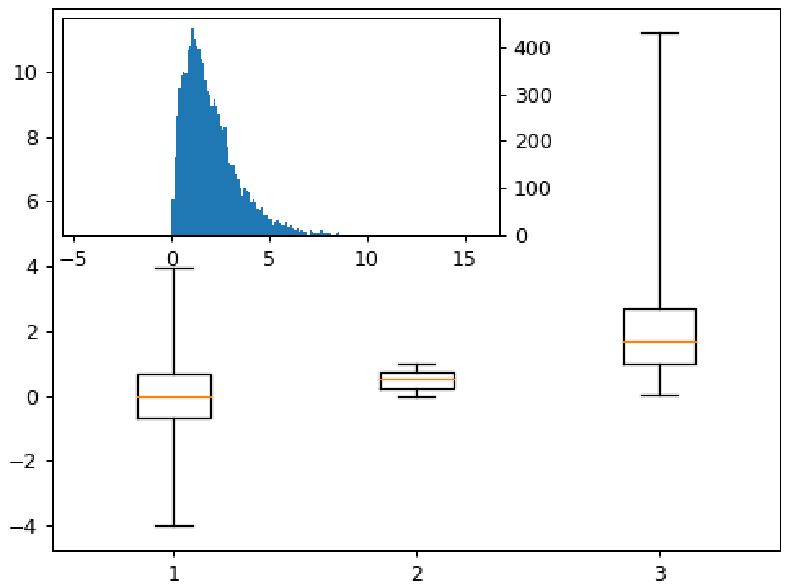
```
plt.figure()
_ = plt.hist(df['gamma'], bins=100)
```

```
<IPython.core.display.Javascript object>
```




```python
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)
```

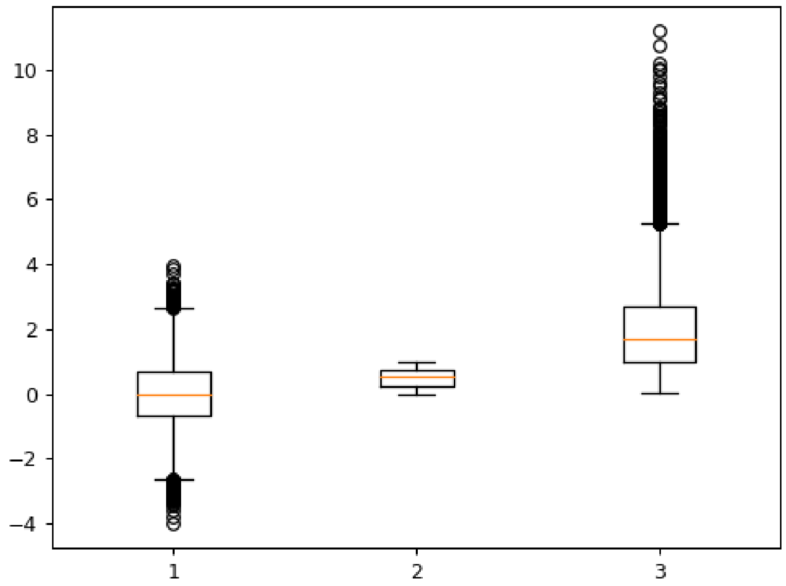
```
<IPython.core.display.Javascript object>
```




```python
# switch the y axis ticks for ax2 to the right side
ax2.yaxis.tick_right()
```

```python
# if `whis` argument isn't passed, boxplot defaults to showing 1.5*interquartile (IQR) whiskers with outliers
# whisç”¨æŒ‡å®šä¸Šä¸‹é¡»ä¸Žä¸Šä¸‹å››åˆ†ä½çš„è·ç¦»ï¼Œå³å›¾1é‡Œä¸Šé¡»å€¼3åˆ°ä¸Šå››åˆ†ä½å€¼0.7çš„è·ç¦»
plt.figure()
_ = plt.boxplot([ df['normal'], df['random'], df['gamma'] ] )
```

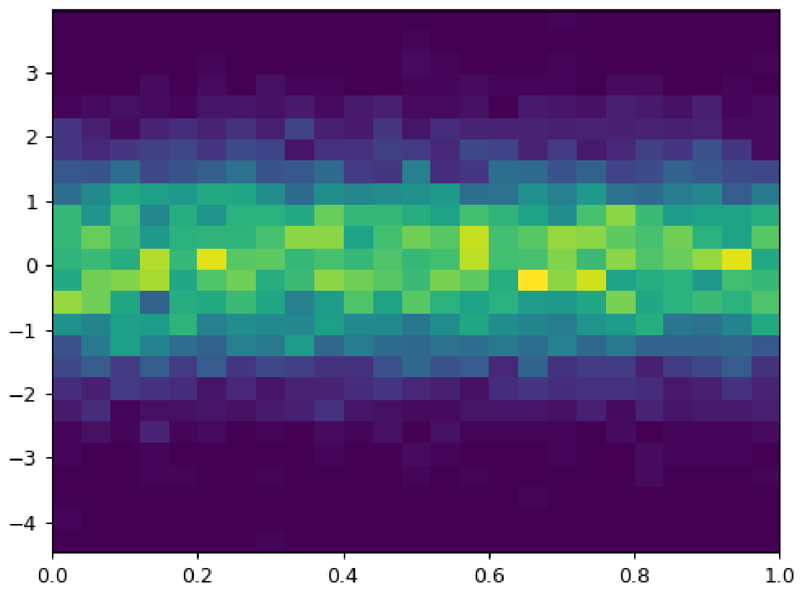
```
<IPython.core.display.Javascript object>
```



Heatmaps

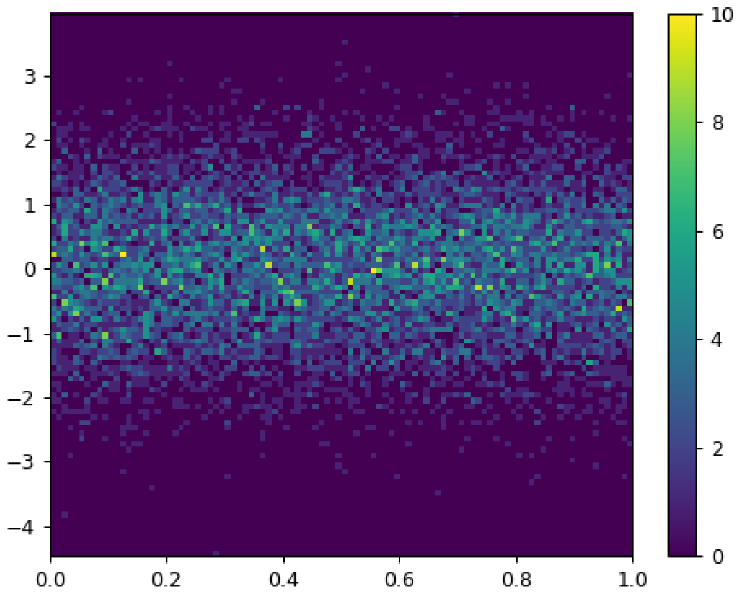
```
plt.figure()

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

```
<IPython.core.display.Javascript object>
```




```
plt.figure()
_ = plt.hist2d(X, Y, bins=100)
```

```
<IPython.core.display.Javascript object>
```




```python
# add a colorbar legend
plt.colorbar()
```

```
<matplotlib.colorbar.Colorbar at 0x7fdd5c6b7e48>
```

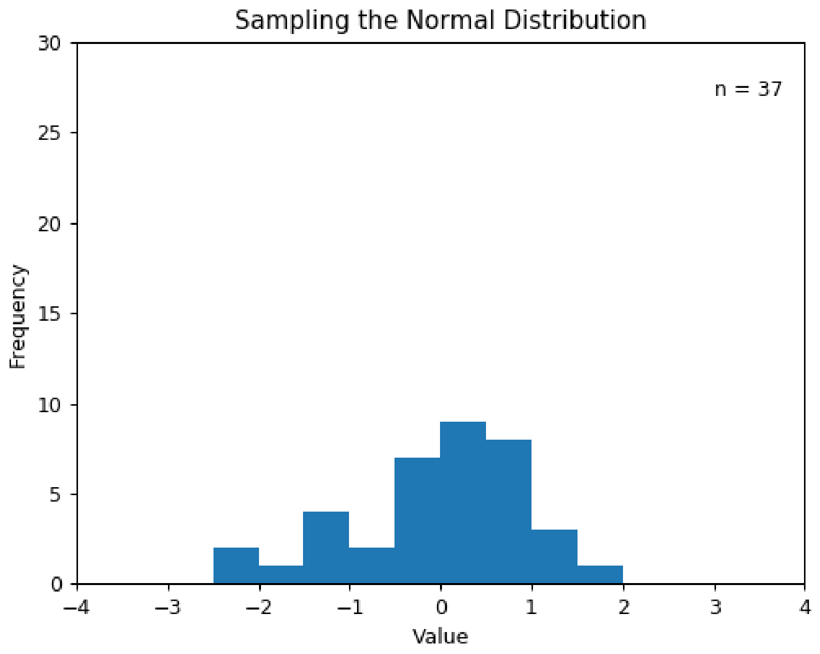
Animations

```python
import matplotlib.animation as animation

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

```python
# 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])
```

```python
fig = plt.figure()
a = animation.FuncAnimation(fig, update, interval=100)
```

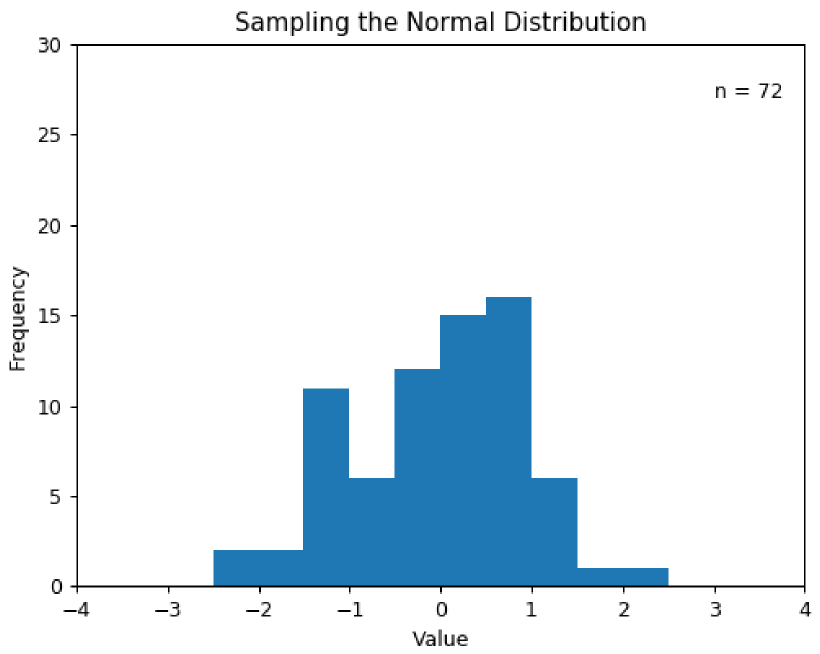
```
<IPython.core.display.Javascript object>
```



Interactivity

```
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)
```

```
<IPython.core.display.Javascript object>
```



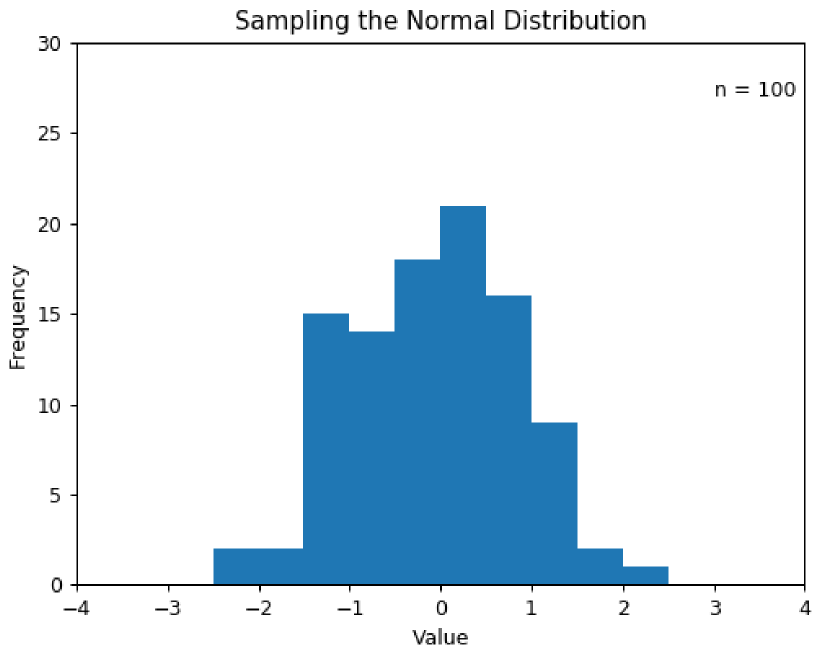
```
7
```

```python
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
```

	height	origin	weight
0	0.773199	China	0.927905
1	0.083552	Chile	0.387157
2	0.746691	Germany	0.871206
3	0.374043	UK	0.184381
4	0.764909	India	0.734964
5	0.554867	Canada	0.484849
6	0.504376	Brazil	0.454651
7	0.436940	Mexico	0.453248
8	0.932848	Iraq	0.413904
9	0.575899	USA	0.200458

```python
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')
```

```
<IPython.core.display.Javascript object>
```



```
<matplotlib.text.Text at 0x7fdda4fc8780>
```

```
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)
```

```
7
```