Week3

May 13, 2018

1 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')
```

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


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

Out[2]:
```
[<matplotlib.lines.Line2D at 0x233e0318f98>]
```

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 0x233e03156a0>]
```

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

```
C:\Users\apday\AppData\Local\Continuum\anaconda3\lib\site-packages\matplotlib\cbook\deprecatio
  warnings.warn(message, mplDeprecation, stacklevel=1)
```

```
Out[4]: [<matplotlib.lines.Line2D at 0x233e0d20d68>]
```

```
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')
```

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

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

```
Out[5]: [<matplotlib.lines.Line2D at 0x233e0df7f98>]
```

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

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

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

```
C:\Users\apday\AppData\Local\Continuum\anaconda3\lib\site-packages\matplotlib\cbook\deprecatio
  warnings.warn(message, mplDeprecation, stacklevel=1)
```

```
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, sh
        # plot the linear_data on the 5th subplot axes
        ax5.plot(linear_data, '-')
```

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

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

```
Out[7]: [<matplotlib.lines.Line2D at 0x233e15116a0>]
```

```
In [10]: # 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 [11]: # necessary on some systems to update the plot
         plt.gcf().canvas.draw()
```

2 Histograms

In [12]:
```
# 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 cor
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>
```

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

In [13]:
```
# 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>
```

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

In [14]:
```
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>
```

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

Out[14]:
```
<matplotlib.collections.PathCollection at 0x233e4815e48>
```

```
In [7]: # use gridspec to partition the figure into subplots
        # https://matplotlib.org/users/gridspec.html
        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>
```

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

```
In [8]: 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 [9]: # 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 [10]: # 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)
```

```
In [6]: %%HTML
        <img src='http://educationxpress.mit.edu/sites/default/files/journal/WP1-Fig13.jpg' />
```

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

3 Box and Whisker Plots

```
In [11]: 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 [12]:
```
df.describe()
```

Out[12]:

	gamma	normal	random
count	10000.000000	10000.000000	10000.000000
mean	1.996480	-0.004223	0.504650
std	1.401725	0.999847	0.287764
min	0.013208	-3.807651	0.000058
25%	0.966471	-0.663231	0.256471
50%	1.683910	0.004641	0.503712
75%	2.685069	0.668594	0.757300
max	11.283952	3.842116	0.999955

In [13]:
```
plt.figure()
# create a boxplot of the normal data, assign the output to a variable to supress out
_ = plt.boxplot(df['normal'], whis='range')
```

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

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

In [14]:
```
# 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 [15]:
```
plt.figure()
_ = plt.hist(df['gamma'], bins=100)
```

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

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

In [16]:
```
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>
```

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

```
In [17]: # switch the y axis ticks for ax2 to the right side
         ax2.yaxis.tick_right()

In [18]: # if `whis` argument isn't passed, boxplot defaults to showing 1.5*interquartile (IQR)
         plt.figure()
         _ = plt.boxplot([ df['normal'], df['random'], df['gamma'] ] )
```

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

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

4 Heatmaps

```
In [19]: 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>
```

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

```
In [20]: plt.figure()
         _ = plt.hist2d(X, Y, bins=100)
```

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

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

```
In [21]: # add a colorbar legend
         plt.colorbar()

Out[21]: <matplotlib.colorbar.Colorbar at 0x1f040b0e588>
```

5 Animations

In [22]:
```
import matplotlib.animation as animation

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

In [23]:
```
# 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 [24]:
```
fig = plt.figure()
a = animation.FuncAnimation(fig, update, interval=100)
```

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

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

6 Interactivity

In [25]:
```
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

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

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

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

Out[25]: 8

In [26]:
```
from random import shuffle
origins = ['China', 'Brazil', 'India', 'USA', 'Canada', 'UK', 'Germany', 'Iraq', 'Chi

shuffle(origins)

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

Out[26]:

	height	origin	weight
0	0.234522	USA	0.135443
1	0.722963	UK	0.696778
2	0.487043	India	0.167485
3	0.991421	Germany	0.578164
4	0.702518	China	0.872935
5	0.140389	Chile	0.536425
6	0.228273	Iraq	0.354585
7	0.795987	Canada	0.064228
8	0.739887	Brazil	0.233569
9	0.503621	Mexico	0.965329

In [27]:
```
plt.figure()
# picker=5 means the mouse doesn't have to click directly on an event, but can be up
plt.scatter(df['height'], df['weight'], picker=5)
plt.gca().set_ylabel('Weight')
plt.gca().set_xlabel('Height')
```

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

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

Out[27]: Text(0.5,0,'Height')

In [30]:
```
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 detec
plt.gcf().canvas.mpl_connect('pick_event', onpick)
```

Out[30]: 10