Week3

May 29, 2019

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 0x7fa3b97d07b8>]

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 0x7fa3afa0a0b8>]

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 0x7fa3b97ba9e8>]
```

```
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 0x7fa3af919dd8>]
```

```
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>
```

```
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, sha
        # 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 0x7fa3a0347470>]
```

```
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()
```

2 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 a
         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 [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))
```

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

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

```
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)
```

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

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

```
Out[12]: <matplotlib.collections.PathCollection at 0x7fa3992e7668>
```

```
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:])
```

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


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

```
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)
```

```
In [18]: %%HTML
         <img src='http://educationxpress.mit.edu/sites/default/files/journal/WP1-
```

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

3 Box and Whisker Plots

```
In [19]: 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 [20]: df.describe()
```

```
Out[20]:               gamma        normal        random
         count  10000.000000  10000.000000  10000.000000
         mean       2.015305     -0.000772      0.497707
         std        1.412793      0.994862      0.287534
         min        0.012539     -4.324106      0.000090
         25%        0.993188     -0.650477      0.251112
         50%        1.701335     -0.006622      0.490552
         75%        2.689700      0.672873      0.746845
         max       12.807731      3.932277      0.999908
```

```
In [21]: plt.figure()
         # create a boxplot of the normal data, assign the output to a variable to
         _ = plt.boxplot(df['normal'], whis='range')
```

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

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

```
In [22]: # 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 [23]: plt.figure()
         _ = plt.hist(df['gamma'], bins=100)
```

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

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

```
In [24]: 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 [25]: # switch the y axis ticks for ax2 to the right side
         ax2.yaxis.tick_right()
```

```
In [26]: # if `whis` argument isn't passed, boxplot defaults to showing 1.5*interqu
         plt.figure()
         _ = plt.boxplot([ df['normal'], df['random'], df['gamma'] ] )
```

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

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

4 Heatmaps

```
In [27]: 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 [28]: plt.figure()
         _ = plt.hist2d(X, Y, bins=100)
```

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

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

```
In [29]: # add a colorbar legend
         plt.colorbar()
```

```
Out[29]: <matplotlib.colorbar.Colorbar at 0x7fa390d35240>
```

5 Animations

```
In [30]: import matplotlib.animation as animation

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

```
In [31]: # create the function that will do the plotting, where curr is the current
         def update(curr):
             # check if animation is at the last frame, and if so, stop the animati
             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 [33]: fig = plt.figure()
         a = animation.FuncAnimation(fig, update, interval=100)
```

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


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

6 Interactivity

```
In [34]: 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(ev

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

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


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

```
Out[34]: 7
```

```
In [35]: from random import shuffle
         origins = ['China', 'Brazil', 'India', 'USA', 'Canada', 'UK', 'Germany', '

         shuffle(origins)

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

```
Out[35]:      height   origin    weight
         0  0.233246   Canada  0.375660
         1  0.400265    Chile  0.526053
         2  0.429471    India  0.971926
         3  0.886119      USA  0.446377
         4  0.840650   Brazil  0.767731
         5  0.771163    China  0.119122
         6  0.251117   Mexico  0.464495
         7  0.304493  Germany  0.796560
         8  0.626227       UK  0.145462
         9  0.849131     Iraq  0.751087
```

```
In [36]: plt.figure()
         # picker=5 means the mouse doesn't have to click directly on an event, but
         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[36]: <matplotlib.text.Text at 0x7fa394cbaf98>
```

```
In [37]: 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 eve
         plt.gcf().canvas.mpl_connect('pick_event', onpick)
```

```
Out[37]: 7
```

```
In [ ]:
```