Week3

September 7, 2020

1 Subplots

```
In [2]: %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 0x7fab8f13dc18>]
```

```
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 0x7fab8cd5c0f0>]

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

```
In [ ]: 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')
```

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

```
In [ ]: # 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, '-')
```

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

2 Histograms

```
In [4]: # 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 ar
        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 [5]: # 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 [6]: 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[6]: <matplotlib.collections.PathCollection at 0x7fab8cc75e48>
```

```
In [3]: # 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:])
```

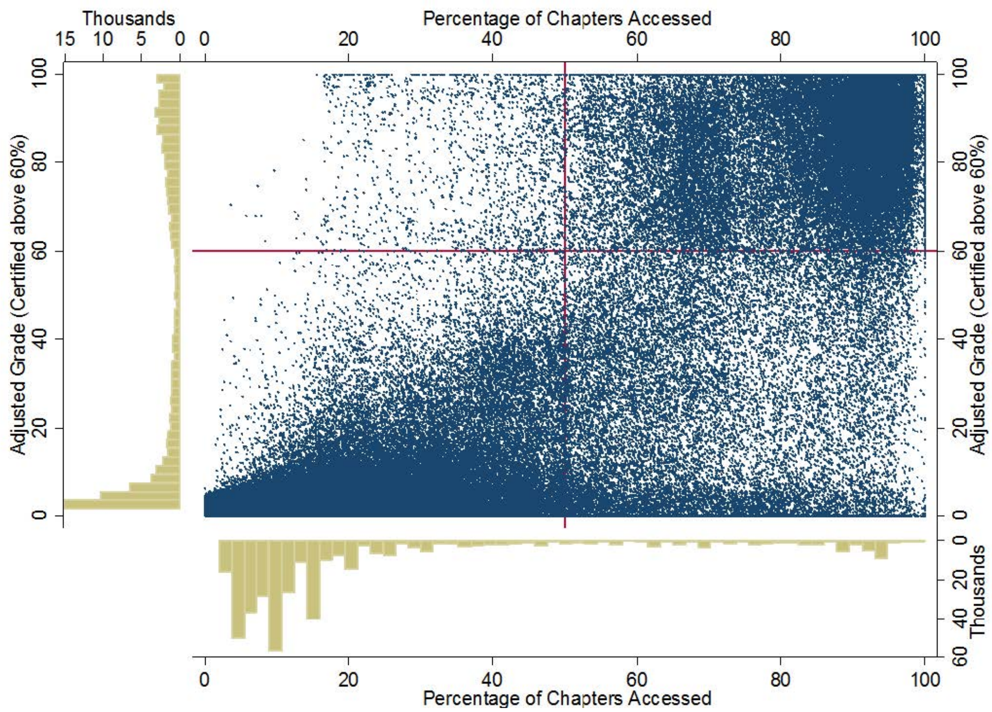
```
<IPython.core.display.Javascript object>



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

```
In [ ]: 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 [ ]: # 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()
```



MOOC DATA

```
In [ ]: # 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)
```

3 Box and Whisker Plots

```
In [4]: 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 [9]: df.describe()
```

```
Out[9]:              gamma        normal        random
        count  10000.000000  10000.000000  10000.000000
        mean       1.998293     -0.005356      0.498918
        std        1.423822      1.002115      0.290452
        min        0.011390     -3.590000      0.000005
```

```
       25%       0.955942    -0.682804     0.245649
       50%       1.670791    -0.017679     0.493155
       75%       2.669577     0.672531     0.752088
       max      11.656577     3.563709     0.999993
```

```
In [10]: 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 [11]: # 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 [12]: plt.figure()
         _ = plt.hist(df['gamma'], bins=100)
```

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


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

```
In [5]: 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 [ ]: # switch the y axis ticks for ax2 to the right side
        ax2.yaxis.tick_right()
```

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

4 Heatmaps

In [ ]:
```
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)
```

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

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

5 Animations

In [6]:
```
import matplotlib.animation as animation

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

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

6 Interactivity

In [7]:
```
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(eve
```

```
        # 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[7]: 7

```
In [8]: from random import shuffle
        origins = ['China', 'Brazil', 'India', 'USA', 'Canada', 'UK', 'Germany', 'I

        shuffle(origins)

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

Out[8]:

	height	origin	weight
0	0.328799	China	0.319679
1	0.374345	Mexico	0.876862
2	0.054500	Germany	0.561160
3	0.648827	USA	0.662121
4	0.406197	Brazil	0.227190
5	0.035085	Iraq	0.301931
6	0.367490	Canada	0.669159
7	0.352369	India	0.313088
8	0.201859	Chile	0.579325
9	0.670391	UK	0.583058

```
In [14]: 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[14]: <matplotlib.text.Text at 0x7fddaa1fb5c0>

In [12]:
```python
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[12]: 7

In [ ]: