Week3

March 5, 2021

1 Subplots

In [ ]:
```
%matplotlib notebook

import matplotlib.pyplot as plt
import numpy as np

plt.subplot?
```

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

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

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

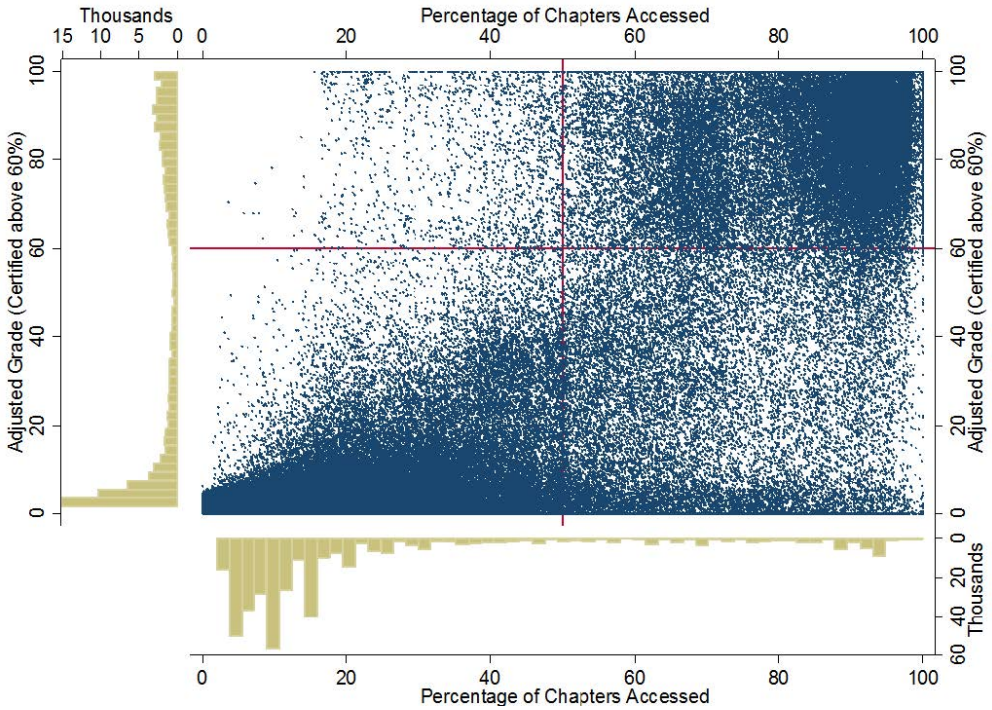
In [ ]:
```
plt.figure()
ax1 = plt.subplot(1, 2, 1)
plt.plot(linear_data, '-o')
# pass sharey=ax1 to ensure the two subplots share the same y axis
ax2 = plt.subplot(1, 2, 2, sharey=ax1)
plt.plot(exponential_data, '-x')
```

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 [ ]: # 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))

In [ ]: # 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))

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

In [ ]: # 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:])
```



MOOC DATA

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

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()
```

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

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

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

In [ ]:
```
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)
```

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 [ ]: 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 [ ]: 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 when
        plt.gcf().canvas.mpl_connect('button_press_event', onclick)

In [ ]: 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
```

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

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