Week3

July 8, 2022

1 Subplots

In [1]:
```
%matplotlib notebook

import matplotlib.pyplot as plt
import numpy as np
# Desactivo el límite de figuras que permite tener abiertas matplotlib de m
import matplotlib as mpl
mpl.rc('figure', max_open_warning = 0)

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 0x7f09d43dedd8>]
```

In [3]:
```
# Notese que los ejes Y de arriba tienen distintos tamaños

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 0x7f09d1dbe048>]
```

In [4]:
```
# Si ponemos ambos datos (lineales y exponenciales) dentro del mismo subplo
# eje Y de ambos subplots tenían distintos tamaños

plt.figure()
plt.subplot(1, 2, 1)
plt.plot(linear_data, '-o')
plt.plot(exponential_data, '-x')
plt.subplot(1, 2, 2)
plt.plot(exponential_data, '-x')
```

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

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

Out[4]:
```
[<matplotlib.lines.Line2D at 0x7f09d1d31048>]
```

In [5]:
```
# Compartir eje Y entre dos subplots para que tengan las mismas dimensiones

plt.figure()
ax1 = plt.subplot(1, 2, 1)
plt.plot(linear_data, '-o')

# Acá nos aseguramos de que el segundo subplot (1,2,2) comparta el eje y co
# 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 0x7f09d1c3b2e8>]
```

In [6]:
```
# Vemos que podemos definir los subplot con comas o con números enteros sin

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

Out[6]:
```
True
```

In [7]:
```
# Podemos fijar los ejes X e Y de varios subplots con el comando plt.subplo

# create a 3x3 grid of subplots
# Acá haremos el tuple unpacking de la figura y los 9 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, '-')

        # Agrego datos en el subplot9 = ax9
        ax9.plot(linear_data*2, 'r^')
```

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

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

```
Out[7]: [<matplotlib.lines.Line2D at 0x7f09d17464e0>]
```

```
In [8]: # Fig es el objeto figura, o sea, la figura que contiene los distintos subp

        type(fig)
```

```
Out[8]: matplotlib.figure.Figure
```

```
In [9]: plt.gcf() == fig
```

```
Out[9]: True
```

```
In [13]: # Obtengo los 9 subplots (cada subplot es un objeto axes)
         plt.gcf().get_axes()
```

```
Out[13]: [<matplotlib.axes._subplots.AxesSubplot at 0x7f09d1b1b550>,
          <matplotlib.axes._subplots.AxesSubplot at 0x7f09d1a8ca58>,
          <matplotlib.axes._subplots.AxesSubplot at 0x7f09d1a5fa90>,
          <matplotlib.axes._subplots.AxesSubplot at 0x7f09d19dd080>,
          <matplotlib.axes._subplots.AxesSubplot at 0x7f09d1935f60>,
          <matplotlib.axes._subplots.AxesSubplot at 0x7f09d1932940>,
          <matplotlib.axes._subplots.AxesSubplot at 0x7f09d189c128>,
          <matplotlib.axes._subplots.AxesSubplot at 0x7f09d1854668>,
          <matplotlib.axes._subplots.AxesSubplot at 0x7f09d17e0630>]
```

```
In [11]: # set inside tick labels to visible
         # acá plt.gcf() es == fig
         # la figura tiene 9 subplots, cada uno con sus axes

         for ax in plt.gcf().get_axes():
             for label in ax.get_xticklabels() + ax.get_yticklabels():
                 label.set_visible(True)
```

```
In [14]: # La celda de arriba puede reescribirse como esta
         # plt.gcf() puede reemplazarse por fig que definimos en la línea 21

         for ax in fig.get_axes():
             for label in ax.get_xticklabels() + ax.get_yticklabels():
                 label.set_visible(True)
```

```
In [13]: fig
```

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

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

```
In [15]: # necessary on some systems to update the plot
         plt.gcf().canvas.draw()
```

```
In [16]: # Ahora pruebo con gráficos que no tienen subplots

         plt.figure()
         plt.plot([1,2,3],[0,1,2])
```

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

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

```
Out[16]: [<matplotlib.lines.Line2D at 0x7f09d168dac8>]
```

```
In [17]: ax = plt.gca()

         # Obtengo la posiciones de los marcadores en X
         ax.get_xticks(), ax.get_yticks()
```

```
Out[17]: (array([ 0.75,  1.  ,  1.25,  1.5 ,  1.75,  2.  ,  2.25,  2.5 ,  2.75,
                  3.  ,  3.25]),
          array([-0.25,  0.  ,  0.25,  0.5 ,  0.75,  1.  ,  1.25,  1.5 ,  1.75,
                  2.  ,  2.25]))
```

```
In [19]: # Vemos la equivalencia
         print(ax.set_xticks([1,3]) == plt.gca().set_xticks([1,3]))


         # Le cambio los xticks del gráfico
         # Ahora solo aparecerán en el eje X los ticks 1 y 3
         ax.set_xticks([1,3])
         plt.gca()
```

```
True
```

```
Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0x7f09d172e6d8>
```

```
In [20]: plt.gcf()
```

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

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

2 Histograms

In [21]:
```
# create 2x2 grid of axis subplots
# Adicionalmente compartimos los ejes X para que tengan las mismas dimensi


fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, sharex=True)
axs = [ax1,ax2,ax3,ax4]

# Vamos a graficar cómo es la distribución normal a medida que se aumenta
# 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 [23]:
```
# Lo mismo pero ahora aumentamos el nro de bins, o sea, el nro de rangos d
# 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 [24]:
```
# Vamos a graficar datos al azar con distribución normal en el eje Y y val


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

# Al graficar vemos que no se puede apreciar ninguna distribución dentro d
# ¿Pero qué pasaría si graficamos la distribución de cada variable por sep
```

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


<IPython.core.display.HTML object>


Out[24]: <matplotlib.collections.PathCollection at 0x7f09d0d69a90>

In [25]: # 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 [26]: 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:])

         # Hacemos una primera prueba de los gráficos

         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')

<IPython.core.display.Javascript object>


<IPython.core.display.HTML object>


In [27]: # Pero podemos mejorarlos los gráficos.
         # 1- podemos hacer que los ejes Y de los histogramas demuestre frecuencias
         # 2- Podemos invertir el eje x (sería el Y de los datos) del gráfico del s
         # .clear() saca todos los datos del subplot
```

```
        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:])


        # 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)
        lower_right.scatter(X, Y)
        # flip the side histogram's x axis
        side_histogram.invert_xaxis()
```

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

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

```
In [32]: # Vemos que top_histogram y lower_right son subplots dado que son del tipo
         # propiedades tales como ax.set_xlim(xmin, xmax)
         top_histogram, lower_right
```

```
Out[32]: (<matplotlib.axes._subplots.AxesSubplot at 0x7f09d0288c18>,
          <matplotlib.axes._subplots.AxesSubplot at 0x7f09d01c9dd8>)
```

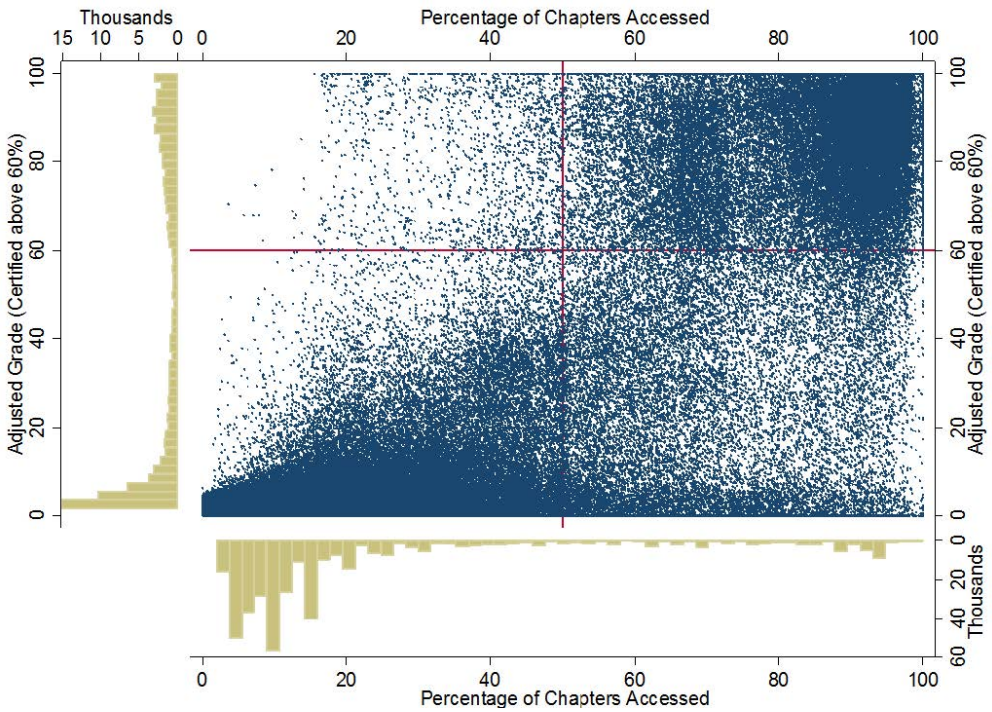
```
In [28]: # Podemos hacer que las dimensiones de los ejes coincidan entre los distin

         # change axes limits
         # para eso iteramos sobre los axes de los subplots top y lower y definimos
         for ax in [top_histogram, lower_right]:
             ax.set_xlim(0, 1)


         # despues iteramos sobres los axes de los subplots side y lower y definimo
         for ax in [side_histogram, lower_right]:
             ax.set_ylim(-5, 5)
```

3 Box and Whisker Plots

```
In [33]: import pandas as pd
         normal_sample = np.random.normal(loc=0.0, scale=1.0, size=10000)
```



MOOC DATA

```
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 [34]:
```
# Se crea una df con 3 columnas y 10.000 datos generados al azar en cada u
# normal, azar, y gamma.

df.head()
```

Out[34]:

	gamma	normal	random
0	1.332684	-0.445734	0.021583
1	1.186587	-0.093500	0.157474
2	4.282165	-0.411930	0.481617
3	2.829093	-0.722599	0.542849
4	2.518719	0.981192	0.549905

In [35]:
```
# Creo una figura y sus subplots indicando que compartan eje Y
fig, (ax1,ax2,ax3) = plt.subplots(1,3, sharey=True)


# Grafico cada una de las distribuciones y selecciono un nro de bins = 100
_1 = ax1.hist(normal_sample, bins=100)
_2 = ax2.hist(random_sample, bins=100)
```

```
        _3 = ax3.hist(gamma_sample, bins=100)

        # asigné los gráficos a variables (_x) porque sino el Jupyter notebook imp
        # funciona perfectamente sin asignarle nombres, es solo para que quedé too
```

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

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

```
In [36]: df.describe()
```

Out[36]:

	gamma	normal	random
count	10000.000000	10000.000000	10000.000000
mean	1.987955	0.012851	0.499766
std	1.405258	0.994221	0.287582
min	0.021030	-3.955414	0.000117
25%	0.971314	-0.655022	0.254412
50%	1.644274	0.011090	0.503329
75%	2.681171	0.687221	0.749995
max	11.852059	3.862908	0.999977

```
In [37]: plt.figure()

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

        # igual que antes, se asigna el boxplot a una variables "_" para que no im
        # las variables "_" no aportan info y se suelen usar para datos que no vol
```

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

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

```
In [38]: plt.figure()

        # Si usamos el parámetro whis='range' lo que hace es gráficar los máximos
        # Si no lo usamos, como en la figura de arriba, los whiskers son el 50% de
        _ = plt.boxplot(df['normal'], whis='range')
```

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

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

```
In [43]: # Graficamos las 3 distribuciones de datos creadas anteriormente

         plt.figure()
         # plot boxplots for all three of df's columns
         _ = plt.boxplot([ df['normal'], df['random'], df['gamma'] ], whis='range')
         plt.xticks([1, 2, 3], ['normal', 'random', 'gamma'])
```

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

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

```
Out[43]: ([<matplotlib.axis.XTick at 0x7f0a101e40b8>,
           <matplotlib.axis.XTick at 0x7f0a101f0f98>,
           <matplotlib.axis.XTick at 0x7f0a101e9208>],
          <a list of 3 Text xticklabel objects>)
```

```
In [44]: plt.figure()
         _ = plt.hist(df['gamma'], bins=100)
```

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

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

```
In [45]: # Insert axis

         # Sirve para insertar un axes (gráfico adicional) sobre un axes base.

         import mpl_toolkits.axes_grid1.inset_locator as mpl_il

         plt.figure()
         # Definimos el gráfico de base, o sea, el axes de base
         plt.boxplot([ df['normal'], df['random'], df['gamma'] ], whis='range')

         # Aora creamos axes sobre el axes de base (plt.gca()), indicamos su tamañ
         ax2 = mpl_il.inset_axes(plt.gca(), width='60%', height='40%', loc=2)

         # Ahora definimo qué gráfico será el que ponemos sobre el gráfico de base
         ax2.hist(df['gamma'], bins=100)

         #  Le agrega un margen al x, y lo desplaza hacia la derecha
         ax2.margins(x=0.5)
```

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

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

```
In [46]: # Invertimos de lado los labels del gráfico insertado, pasan de la izq hac
         ax2.yaxis.tick_right()
```

```
In [48]: # Si a un boxplot no se le pasa un valor al parámetro "whis" directamente
         # y los puntos que quedan fuera del whisker se podrían considerer outlayer

         # 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 [49]: 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=100)
```

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

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

```
In [50]: plt.figure()
         _ = plt.hist2d(X, Y, bins=100)
```

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

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

```
In [39]: # add a colorbar legend
         plt.colorbar()
```

```
Out[39]: <matplotlib.colorbar.Colorbar at 0x7f01b8e25748>
```

```
In [40]: # Hay que tener cuidado con la cantidad de bins que se definan, porque si
         # será una categoría y no habrían distintos colores para graficar.
```

In [51]:
```
# Agrego unos subplots para comprender mejor qué está haciendo el hist2d c

fig, ((ax1,ax2,ax3)) = plt.subplots(1, 3)

ax1.hist(X, bins=100)
ax2.hist2d(X, Y, bins=100)

# Ax3 lo giro y le invierto el sentido
ax3.hist(Y, bins=100, orientation='horizontal')
ax3.invert_xaxis()

# Podemos ver que el heat map muestra con color, los puntos donde hay mayo
```

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

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

5 Animations

In [52]:
```
import matplotlib.animation as animation

#Generamos 100 datos al azar con una distribución normal

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

In [53]:
```
# create the function that will do the plotting for every frame
# Esta función sera llamada por FuncAnim que definimos en la próxima celda

def update(frame):
    # check if animation is at the last frame, and if so, stop the animati
    if frame == n:
        myanimation.event_source.stop()
    plt.cla()
    bins = np.arange(-4, 4, 0.5)
    plt.hist(x[:frame], 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(frame), [3,27])
```

In [54]:
```
# Definimos el objeto FuncAnimation y le decimos que utilice la figura() p
# "update" cada 100 ms. Como frames=None, la función update recibe valores
# que genera valores empezando desde 0 y hasta infinito.
```

```
        # FuncAnimation lo que hara cada 100 ms es:
        # update(0) -->100 ms --> update(1) --> ... --> update(100) y en este fra
        # con el comando anim.event_source.stop()

        fig = plt.figure()
        myanimation = animation.FuncAnimation(fig, update, frames=None, interval=1
```

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

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

6 Interactivity

```
In [55]: 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[55]: 7
```

```
In [57]: from random import shuffle
         import pandas as pd

         origins = ['China', 'Brazil', 'India', 'USA', 'Canada', 'UK', 'Germany', '

         shuffle(origins)

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

Out[57]:

	height	origin	weight
0	0.160625	UK	0.962878
1	0.452725	Chile	0.290085
2	0.129633	Germany	0.654887
3	0.657525	India	0.804751
4	0.135526	Brazil	0.481267
5	0.042941	Iraq	0.430644
6	0.901766	China	0.487353
7	0.976315	Canada	0.290978
8	0.542912	USA	0.098949
9	0.353295	Mexico	0.413355

In [58]:
```
# Plteamos los datos de altura y peso del dataframe

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[58]:
```
<matplotlib.text.Text at 0x7f0a0dfed940>
```

In [64]:
```
# Generamos la interacción con el gráfico, de modo que cuando se seleccion
# corresponden esos datos

# event.ind devuelve [posición del dato seleccionado], por eso le aplica l
# el valor de 'origin'


def onpick(event):
    origin = df.iloc[event.ind[0]]['origin'] # mirar el comentario inmedia
    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)

# Luego de ejecutar la celda ir y hacer click sobre uno de los puntos
```

In [62]:
```
# df.iloc[event.ind[0]]['origin'] sería como la línea de abajo
# Esa línea de código le permite a la función onpick encontrar en la df el

df.iloc[[0][0]]['origin']
```

```
Out[62]: 'UK'
```

```
In [ ]:
```