pyplot

May 11, 2022

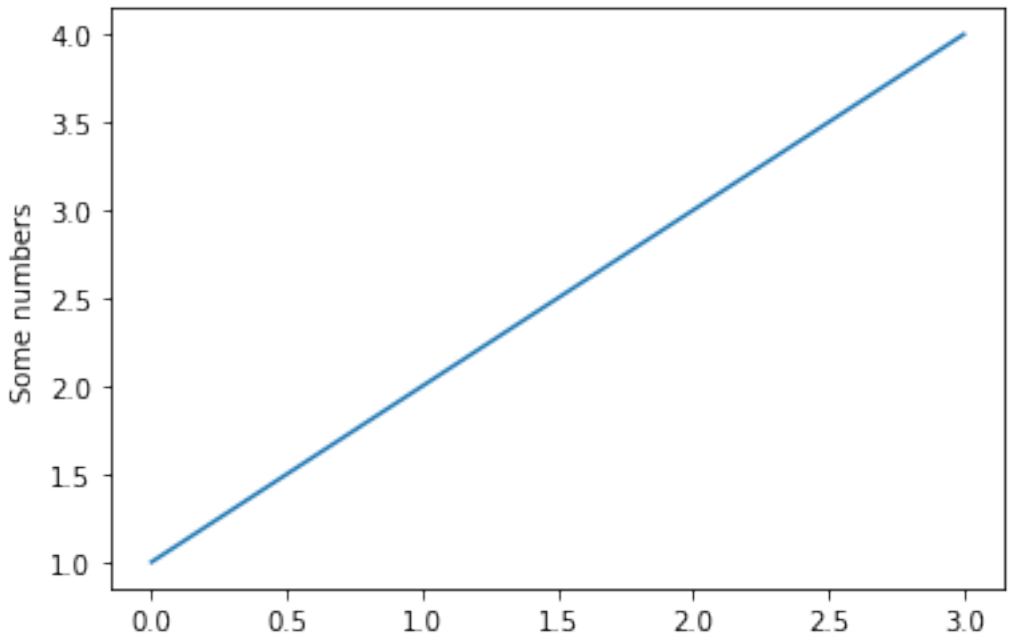
1 Intro to pyplot

matplotlib.pyplot is a collection of functions that make matplotlib work like MATLAB. Each `pyplot` function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc.

In **matplotlib.pyplot** various states are preserved across function calls, so that it keeps track of things like the current figure and plotting area, and the plotting functions are directed to the current axes (please note that "axes" here and in most places in the documentation refers to the axes part of a figure and not the strict mathematical term for more than one axis).

Generating visualizations with pyplot is very quickly

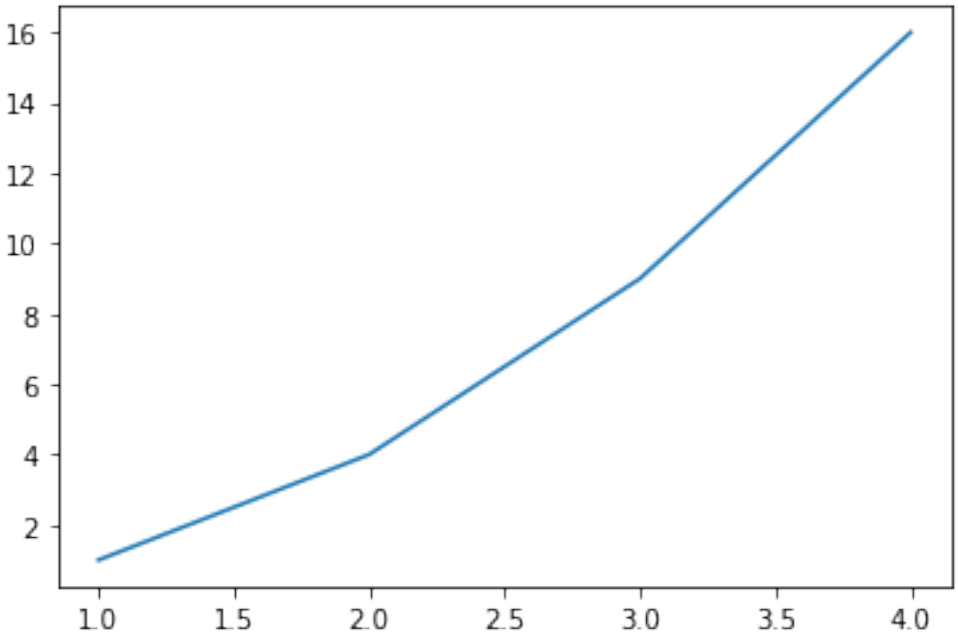
[1]:
```
import matplotlib.pyplot as plt

plt.plot([1, 2, 3, 4])
plt.ylabel('Some numbers')
plt.show()
```



plot is a versatile function, and will take an arbitrary number of arguments. For example, to plot x versus y, you can write

[2]:
```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
```

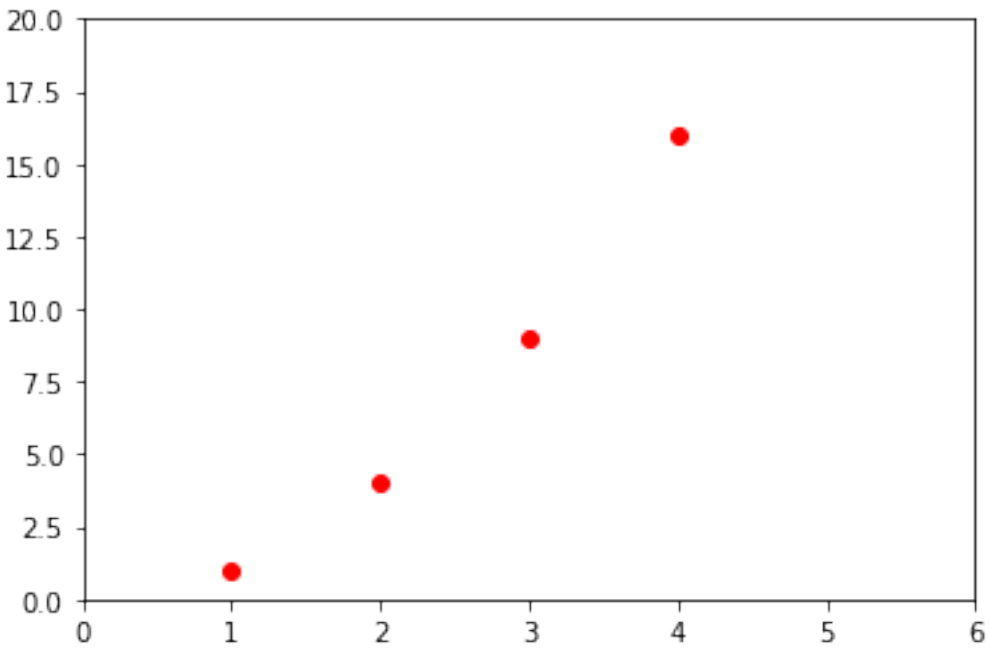
[2]:
```
[<matplotlib.lines.Line2D at 0x7f6e8de56050>]
```



1.1 Formatting the style of your plot

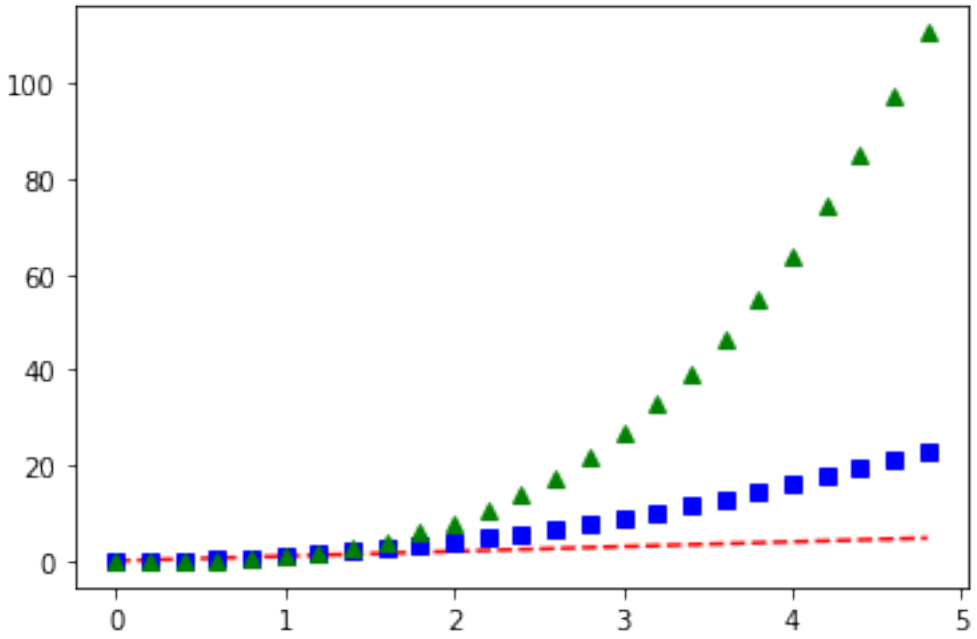
For every x, y pair of arguments, there is an optional third argument which is the format string that indicates the color and line type of the plot. The letters and symbols of the format string are from MATLAB, and you concatenate a color string with a line style string. The default format string is 'b-', which is a solid blue line. For example, to plot the above with red circles, you would issue

[3]:
```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'ro')
plt.axis([0, 6, 0, 20])
plt.show()
```



If matplotlib were limited to working with lists, it would be fairly useless for numeric processing. Generally, you will use numpy arrays. In fact, all sequences are converted to numpy arrays internally. The example below illustrates plotting several lines with different format styles in one function call using arrays.

[4]:
```
import numpy as np

t = np.arange(0., 5., .2)

plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
plt.show()
```

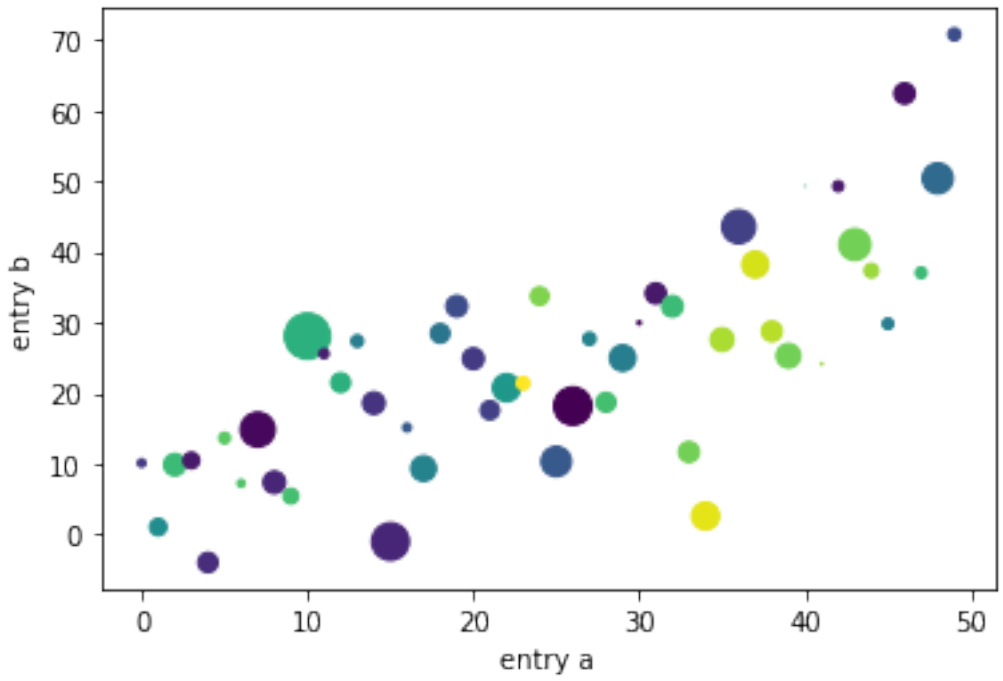


1.2 Plotting with keyword strings

There are some instances where you have data in a format that lets you access particular variables with strings. For example, with **numpy.recarray** or **pandas.DataFrame**.

Matplotlib allows you provide such an object with the **data** keyword argument. If provided, then you may generate plots with the strings corresponding to these variables.

[5]:
```
data = {'a': np.arange(50),
        'c': np.random.randint(0, 50, 50),
        'd': np.random.randn(50)}
data['b'] = data['a'] + 10 * np.random.randn(50)
data['d'] = np.abs(data['d']) * 100

plt.scatter('a', 'b', c='c', s='d', data=data)
plt.xlabel('entry a')
plt.ylabel('entry b')
plt.show()
```

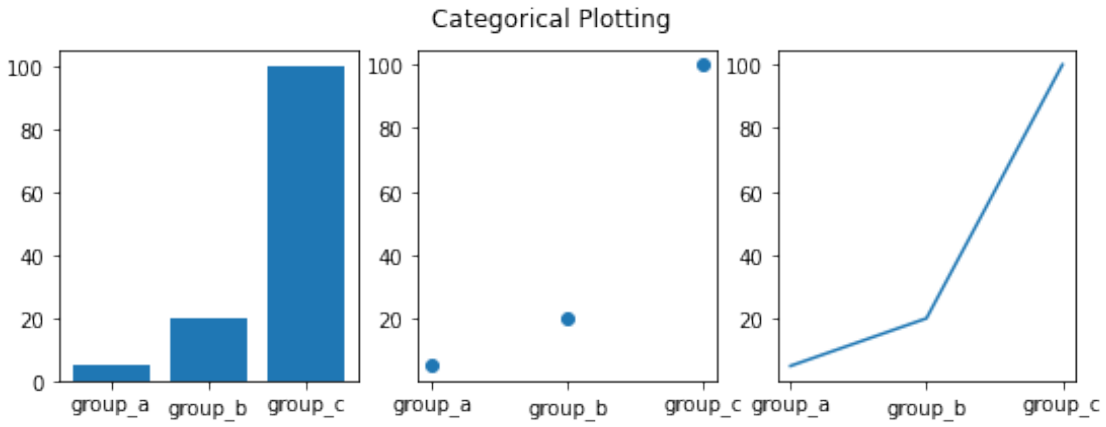


1.3 Plotting with cagorical variables

It is also possible to create a plot using categorical variables. Matplotlib allows you to pass categorical variables directly to many plotting functions. For example

[6]:
```
names = ['group_a', 'group_b', 'group_c']
values = [5, 20, 100]

fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(9, 3))
ax1.bar(names, values)
ax2.scatter(names, values)
ax3.plot(names, values)
plt.suptitle('Categorical Plotting')
```

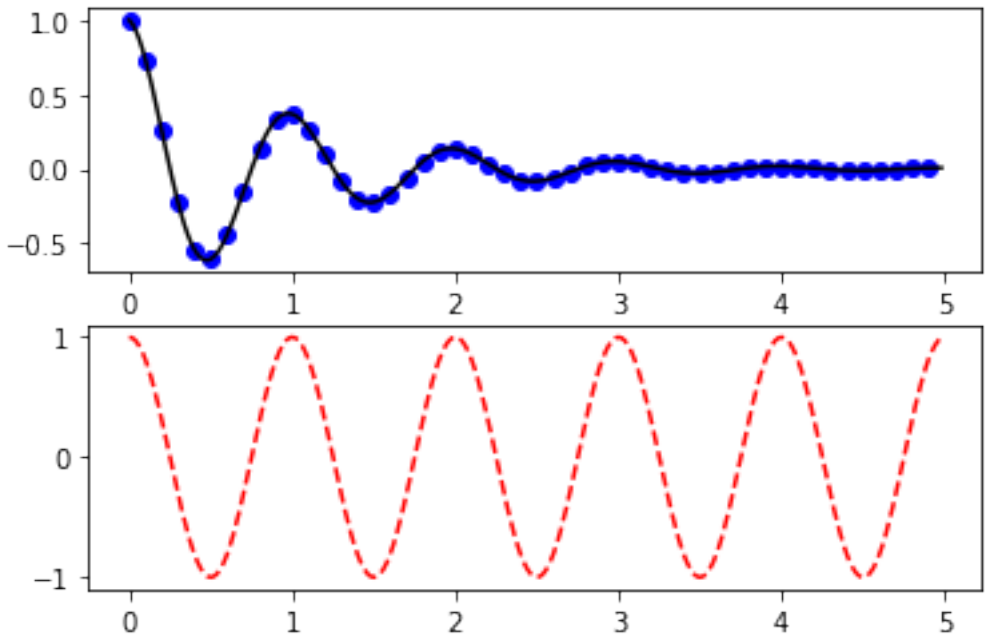
[6]:
```
Text(0.5, 0.98, 'Categorical Plotting')
```



1.4 Working with multiple figures and axes

MATLAB, and **pyplot**, have the concept of the current figure and the current axes. All plotting functions apply to the current axes. The function **gca** returns the current axes (a **matplotlib.axes.Axes** instance), and **gcf** returns the current figure (a **matplotlib.figure.Figure** instance). Normally, you don't have to worry about this, because it is all taken care of behind the scenes. Below is a script to create two subplots.

[7]:
```
def f(t):
    return np.exp(-t) * np.cos(2*np.pi*t)

t1 = np.arange(0., 5., .1)
t2 = np.arange(0., 5., .02)

plt.figure()
plt.subplot(211)
plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')

plt.subplot(212)
plt.plot(t2, np.cos(2*np.pi*t2), 'r--')
plt.show()
```



The **figure** call here is optional because a figure will be created if none exists, just as an axes will be created (equivalent to an explicit subplot() call) if none exists. The **subplot** call specifies `numrows`, `numcols`, `plot_number` where plot_number ranges from 1 to `numrows*numcols`. The commas in the subplot call are optional if `numrows*numcols < 10`. So `subplot(211)` is identical to `subplot(2, 1, 1)`.

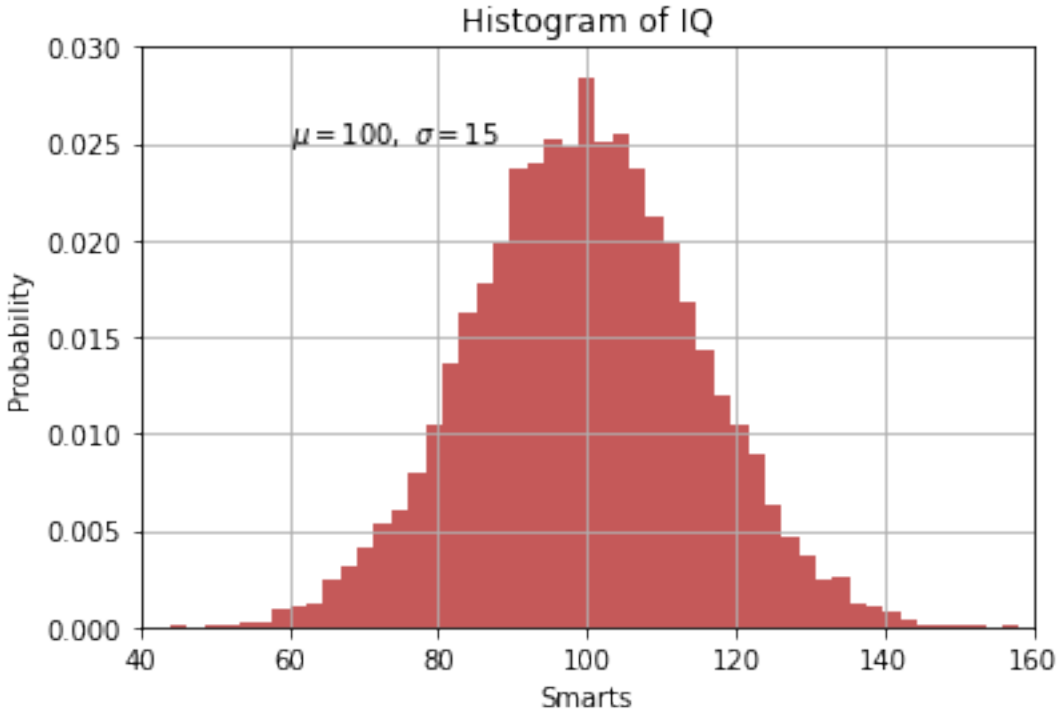
You can create an arbitrary number of subplots and axes. If you want to place an axes manually, i.e., not on a rectangular grid, use **axes**, which allows you to specify the location as `axes([left, bottom, width, height])` where all values are in fractional (0 to 1) coordinates. See Axes Demo for an example of placing axes manually and Multiple subplots for an example with lots of subplots.

1.5 Working with text

text can be used to add text in an arbitrary location, and **xlabel**, **ylabel** and **title** are used to add text in the indicated locations.

[8]:
```
mu, sigma = 100, 15
x = mu + sigma * np.random.randn(10000)

n, bins, patches = plt.hist(x, bins=50, density=1, facecolor='firebrick',␣
 ↪alpha=0.75)

plt.xlabel('Smarts')
plt.title('Histogram of IQ')
plt.ylabel('Probability')
plt.text(60, .025, r'$\mu=100,\ \sigma=15$')
```

```
plt.axis([40, 160, 0, 0.03])
plt.grid(True)
plt.show()
```



Created in Deepnote