Plotting and Visualization

Part 2

A Brief matplotlib API Primer

Part 2

Adjusting the spacing around subplots

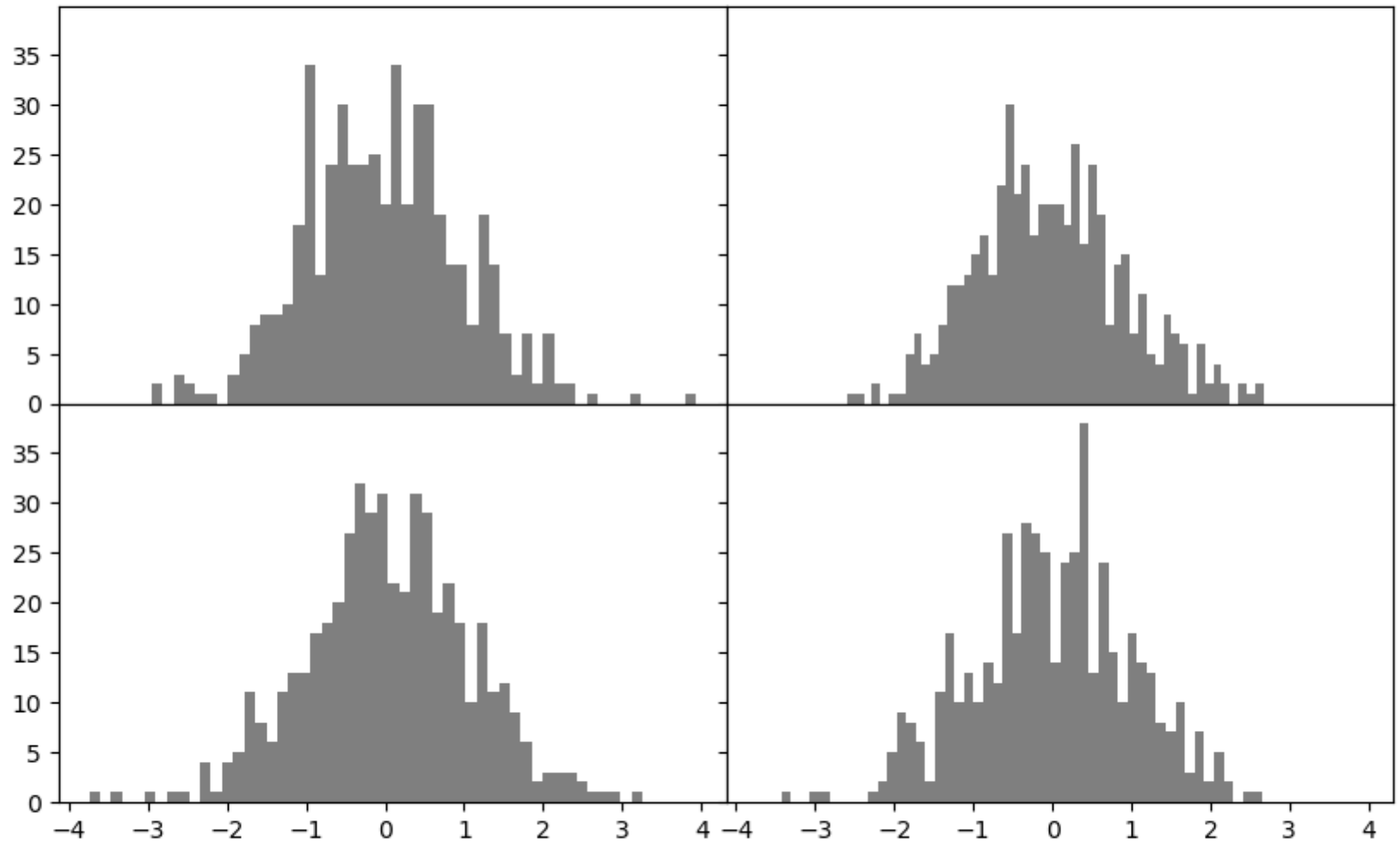
- By default matplotlib leaves a certain amount of padding around the outside of the subplots and spacing between subplots.
- This spacing is all specified relative to the height and width of the plot, so that if you resize the plot either programmatically or manually using the GUI window, the plot will dynamically adjust itself.
- You can change the spacing using the `subplots_adjust` method on `Figure` objects, also available as a top-level function:

```
subplots_adjust(left=None, bottom=None, right=None, top=None,
                wspace=None, hspace=None)
```

- `wspace` and `hspace` controls the percent of the figure width and figure height, respectively, to use as spacing between subplots.

In [16]:
```
fig, axes = plt.subplots(2, 2, sharex=True, sharey=True)
for i in range(2):
    for j in range(2):
        axes[i, j].hist(np.random.randn(500), bins=50, color='k', alpha=0.5)
plt.subplots_adjust(wspace=0, hspace=0)
```

Figure 4



x=0.595318 y=11.8

Colors, Markers, and Line Styles

- Matplotlib's main `plot` function accepts arrays of x and y coordinates and optionally a string abbreviation indicating color and line style.
- For example, to plot x versus y with green dashes, you would execute:

```
ax.plot(x, y, 'g--')
```

- This way of specifying both color and line style in a string is provided as a convenience; in practice if you were creating plots programmatically you might prefer not to have to munge strings together to create plots with the desired style.
- The same plot could also have been expressed more explicitly as:

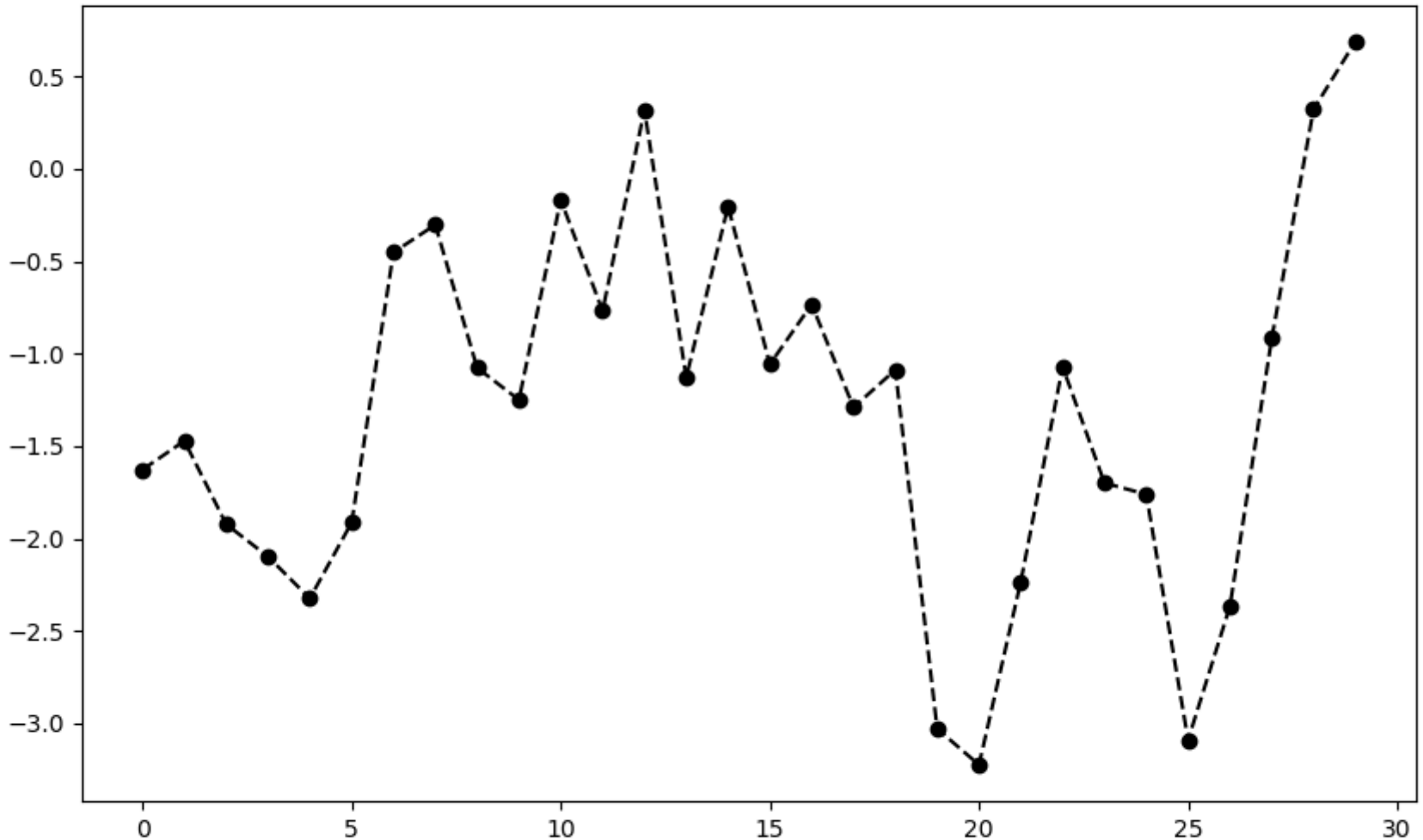
```
ax.plot(x, y, linestyle='--', color='g')
```

- There are a number of color abbreviations provided for commonly used colors, but you can use any color on the spectrum by specifying its hex code (e.g., `'#CECECE'`).
- You can see the full set of line styles by looking at the docstring for `plot` (use `plot?` in Ipython or Jupyter).

- Line plots can additionally have markers to highlight the actual data points.
- Since matplotlib creates a continuous line plot, interpolating between points, it can occasionally be unclear where the points lie.
- The marker can be part of the style string, which must have color followed by marker type and line style:

```
In [17]: plt.figure()
         from numpy.random import randn
         plt.plot(randn(30).cumsum(), 'ko--')
```

Figure 5



x=0.9744 y=-

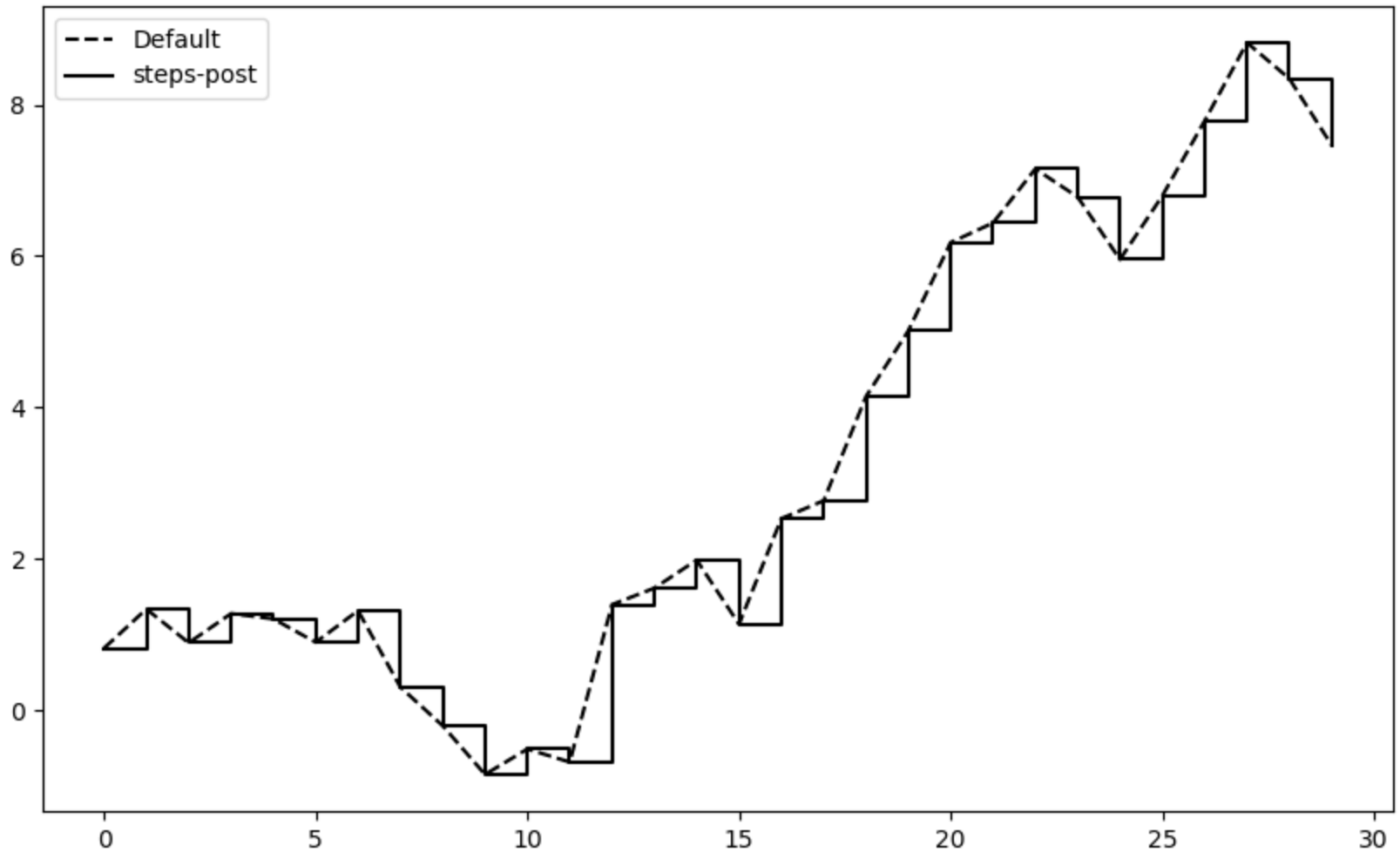
- This could also have been written more explicitly as:

```
plot(randn(30).cumsum(), color='k', linestyle='dashed', marker='o')
```

- For line plots, you will notice that subsequent points are linearly interpolated by default.
- This can be altered with the `drawstyle` option:

```
In [18]: plt.figure()
         data = np.random.randn(30).cumsum()
         plt.plot(data, 'k--', label='Default')
         plt.plot(data, 'k-', drawstyle='steps-post', label='steps-post')
         plt.legend(loc='best')
```

Figure 6



x=0.727432 y=4.45

Ticks, Labels, and Legends

- For most kinds of plot decorations, there are two main ways to do things: using the procedural `pyplot` interface (i.e., `matplotlib.pyplot`) and the more object-oriented native matplotlib API.

- The `pyplot` interface, designed for interactive use, consists of methods like `xlim`, `xticks`, and `xticklabels`.
- These control the plot range, tick locations, and tick labels, respectively.
- They can be used in two ways:
  - Called with no arguments returns the current parameter value (e.g., `plt.xlim()` returns the current x-axis plotting range)
  - Called with parameters sets the parameter value (e.g., `plt.xlim([0, 10])`, sets the x-axis range to `0` to `10`)

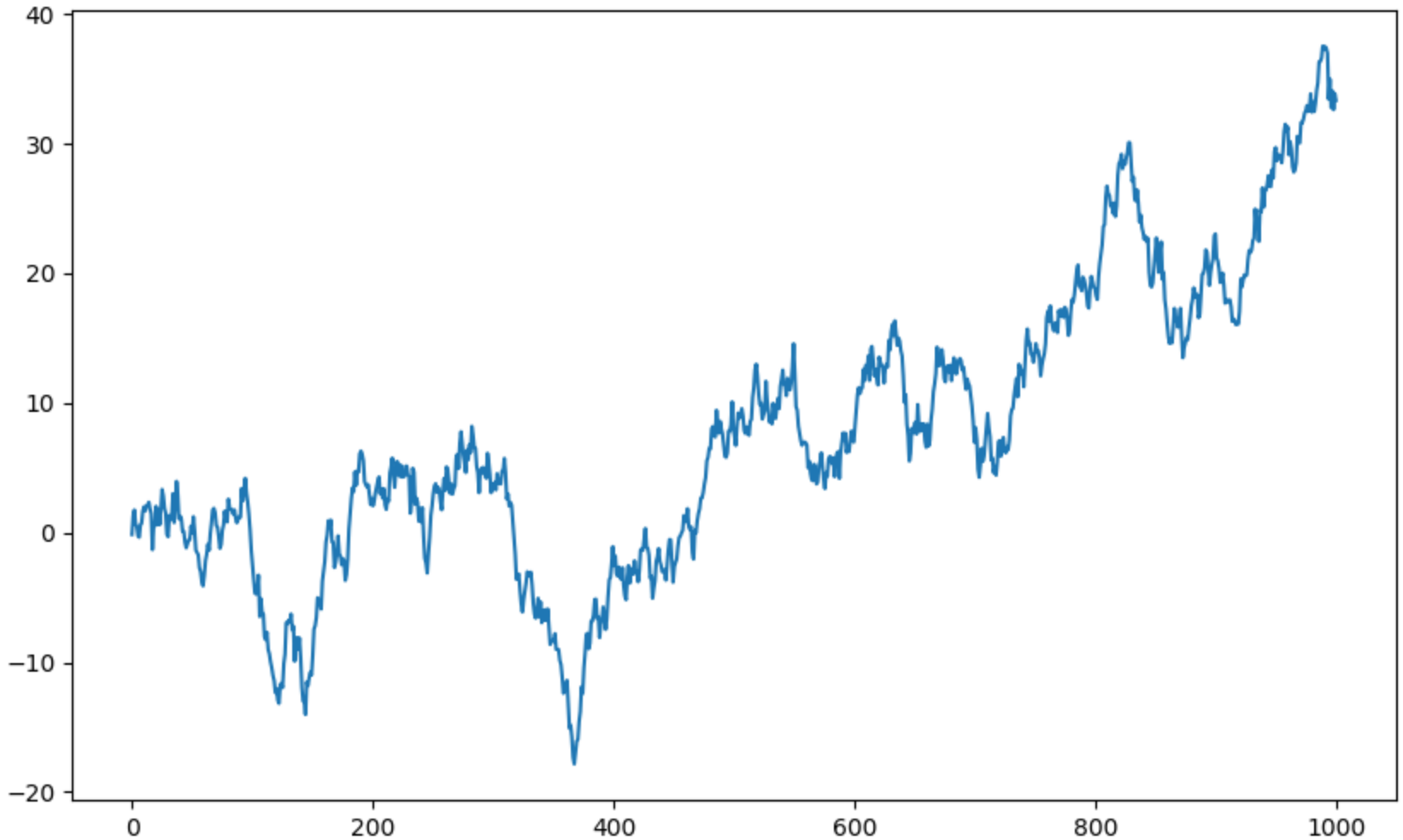
- All such methods act on the active or most recently created `AxesSubplot`.
- Each of them corresponds to two methods on the subplot object itself; in the case of `xlim` these are `ax.get_xlim` and `ax.set_xlim`.

Setting the title, axis labels, ticks, and ticklabels

- To illustrate customizing the axes, we'll create a simple figure and plot of a random walk.

```
In [19]: fig = plt.figure()
         ax = fig.add_subplot(1, 1, 1)
         ax.plot(np.random.randn(1000).cumsum())
```

Figure 7

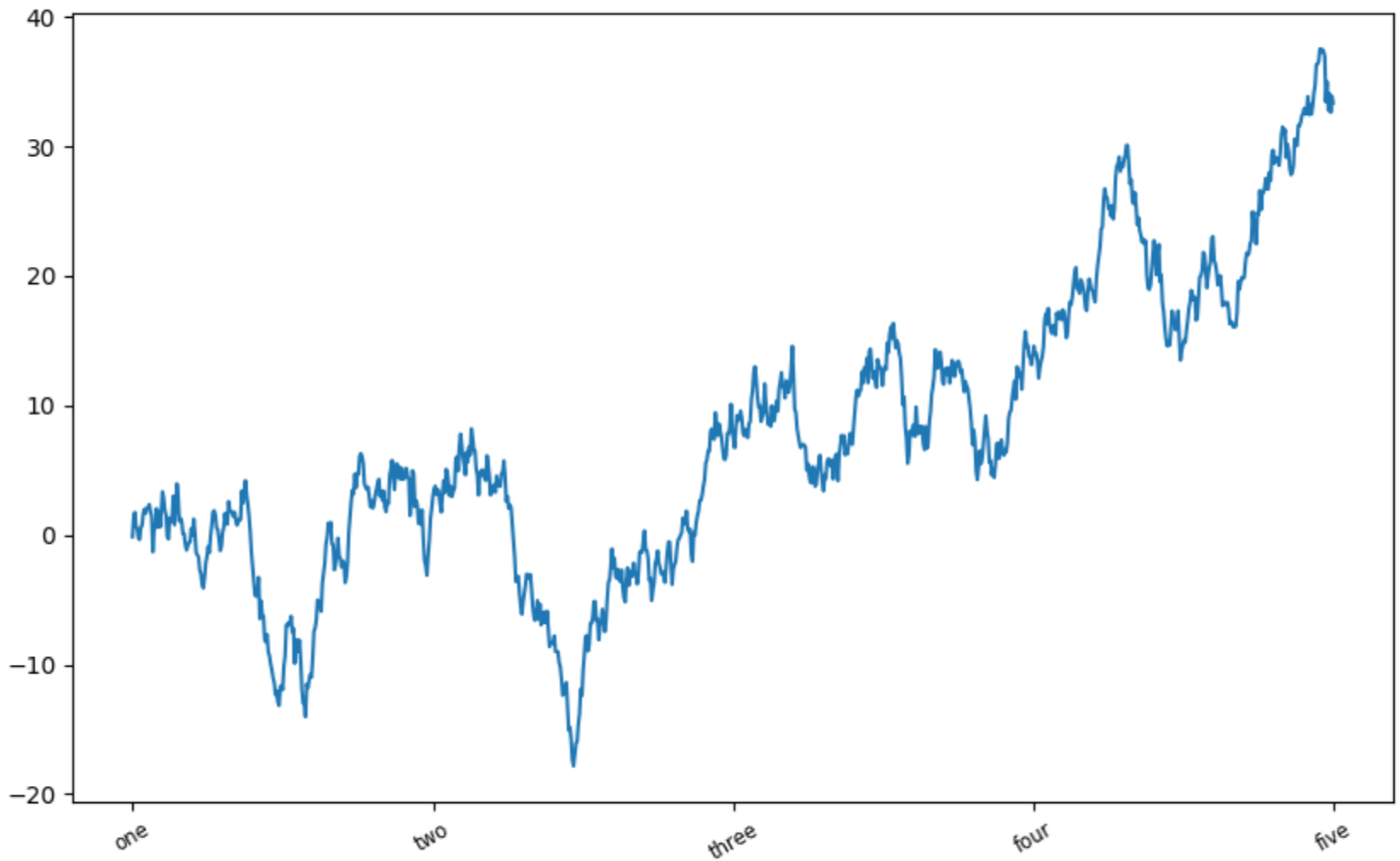


x=141.329 y=11.4

- To change the x-axis ticks, it's easiest to use `set_xticks` and `set_xticklabels`.
- The former instructs matplotlib where to place the ticks along the data range; by default these locations will also be the labels.
- But we can set any other values as the labels using `set_xticklabels`:

```
In [20]: ticks = ax.set_xticks([0, 250, 500, 750, 1000])
         labels = ax.set_xticklabels(['one', 'two', 'three', 'four', 'five'],
                                     rotation=30, fontsize='small')
```

Figure 7



- Lastly, `set_xlabel` gives a name to the x-axis and `set_title` the subplot title:

```
In [21]: ax.set_title('My first matplotlib plot')
         ax.set_xlabel('Stages')

Out[21]: Text(0.5, 23.96389197302451, 'Stages')
```

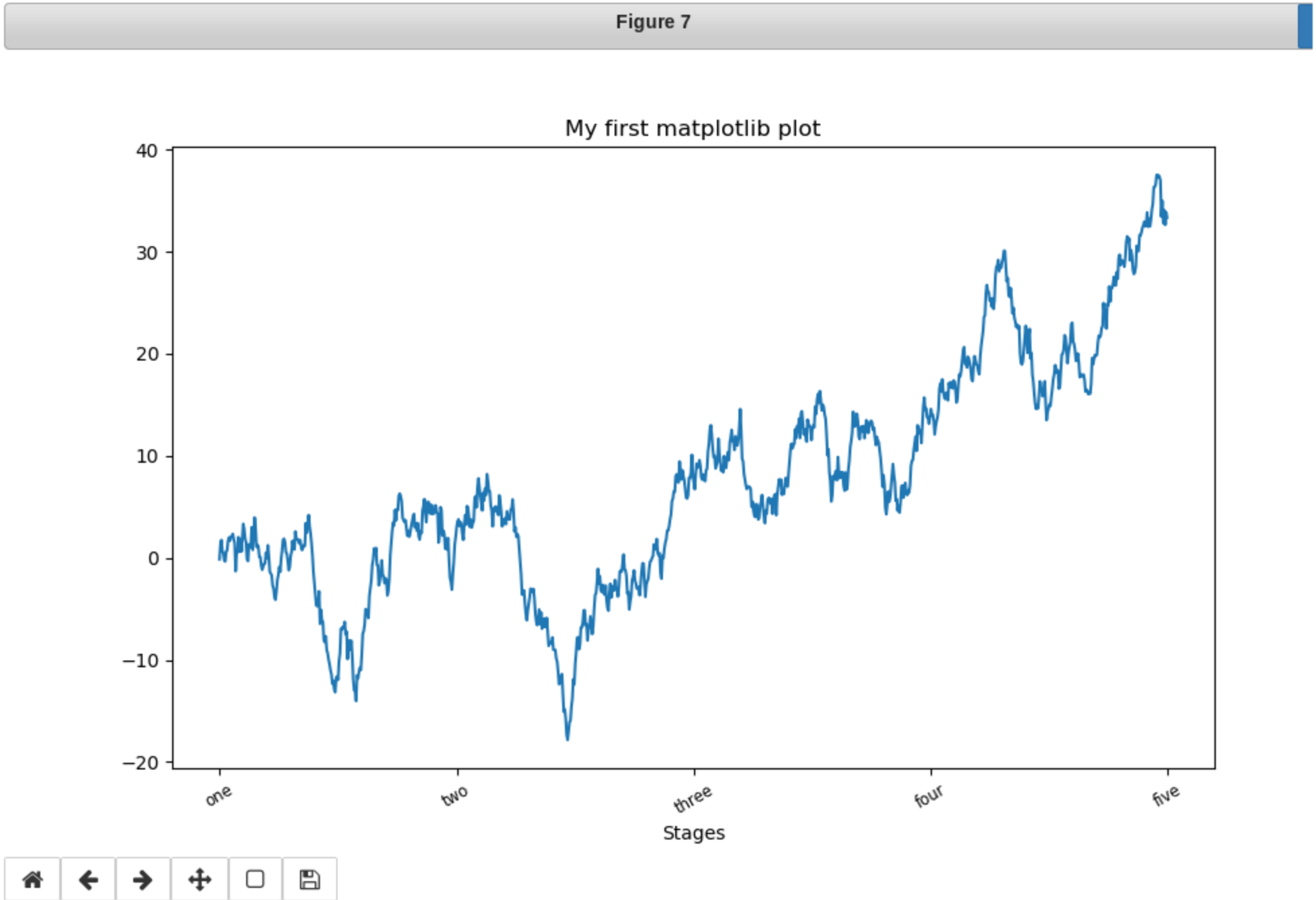


Figure 7
My first matplotlib plot
40
30
20
10
0
−10
−20
one
two
three
four
five
Stages

- Modifying the y-axis consists of the same process, substituting `y` for `x` in the above.
- The axes class has a `set` method that allows batch setting of plot properties.
- From the prior example, we could also have written:

```
props = {
    'title': 'My first matplotlib plot',
    'xlabel': 'Stages'
}
ax.set(**props)
```

Adding legends

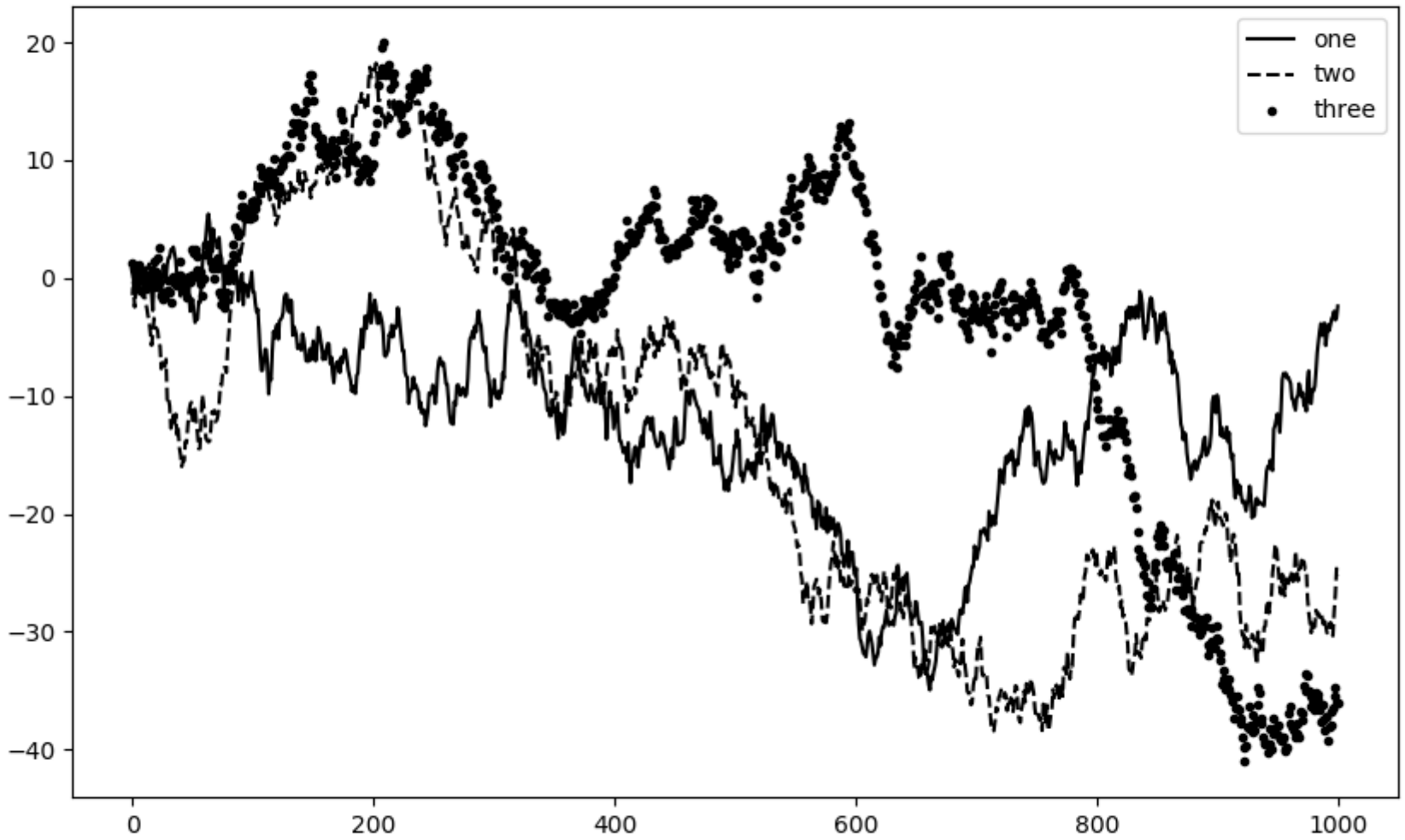
- Legends are another critical element for identifying plot elements.
- There are a couple of ways to add one.
- The easiest is to pass the `label` argument when adding each piece of the plot:

```
In [22]: from numpy.random import randn
         fig = plt.figure(); ax = fig.add_subplot(1, 1, 1)
         ax.plot(randn(1000).cumsum(), 'k', label='one')
         ax.plot(randn(1000).cumsum(), 'k--', label='two')
         ax.plot(randn(1000).cumsum(), 'k.', label='three')
```

- Once you've done this, you can either call `ax.legend()` or `plt.legend()` to automatically create a legend.

```
In [23]: ax.legend(loc='best')
Out[23]: <matplotlib.legend.Legend at 0x7fce82af1cc0>
```

Figure 8



- The `legend` method has several other choices for the location `loc` argument.
- See the docstring (with `ax.legend?`) for more information.

- The `loc` tells matplotlib where to place the plot.
- If you aren’t picky, `'best'` is a good option, as it will choose a location that is most out of the way.
- To exclude one or more elements from the legend, pass no label or `label='_nolegend_'`.