```
In[8]: # For every axis, set the x and y major locator
       for axi in ax.flat:
           axi.xaxis.set_major_locator(plt.MaxNLocator(3))
           axi.yaxis.set_major_locator(plt.MaxNLocator(3))
       fig
```

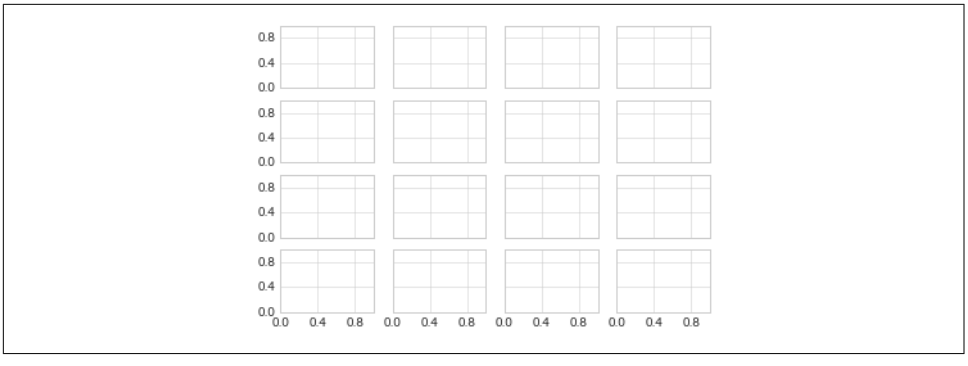


*Figure 4-77. Customizing the number of ticks*

This makes things much cleaner. If you want even more control over the locations of regularly spaced ticks, you might also use `plt.MultipleLocator`, which we'll discuss in the following section.

## Fancy Tick Formats

Matplotlib's default tick formatting can leave a lot to be desired; it works well as a broad default, but sometimes you'd like to do something more. Consider the plot shown in Figure 4-78, a sine and a cosine:

```
In[9]: # Plot a sine and cosine curve
       fig, ax = plt.subplots()
       x = np.linspace(0, 3 * np.pi, 1000)
       ax.plot(x, np.sin(x), lw=3, label='Sine')
       ax.plot(x, np.cos(x), lw=3, label='Cosine')

       # Set up grid, legend, and limits
       ax.grid(True)
       ax.legend(frameon=False)
       ax.axis('equal')
       ax.set_xlim(0, 3 * np.pi);
```

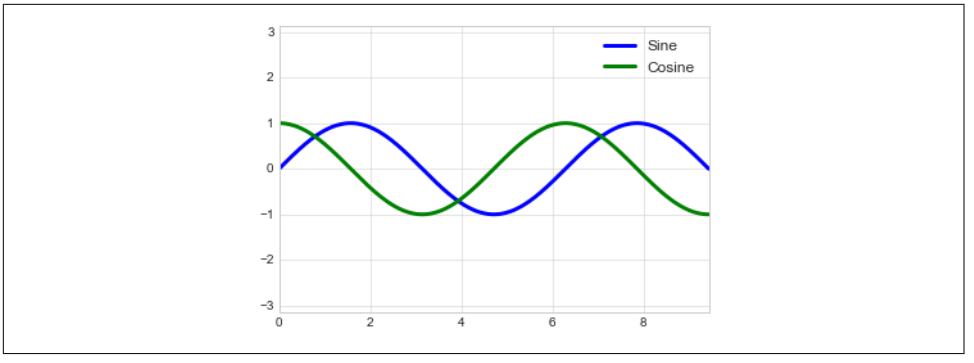


*Figure 4-78. A default plot with integer ticks*

There are a couple changes we might like to make. First, it's more natural for this data to space the ticks and grid lines in multiples of $\pi$. We can do this by setting a `MultipleLocator`, which locates ticks at a multiple of the number you provide. For good measure, we'll add both major and minor ticks in multiples of $\pi/4$ (Figure 4-79):

```
In[10]: ax.xaxis.set_major_locator(plt.MultipleLocator(np.pi / 2))
        ax.xaxis.set_minor_locator(plt.MultipleLocator(np.pi / 4))
        fig
```

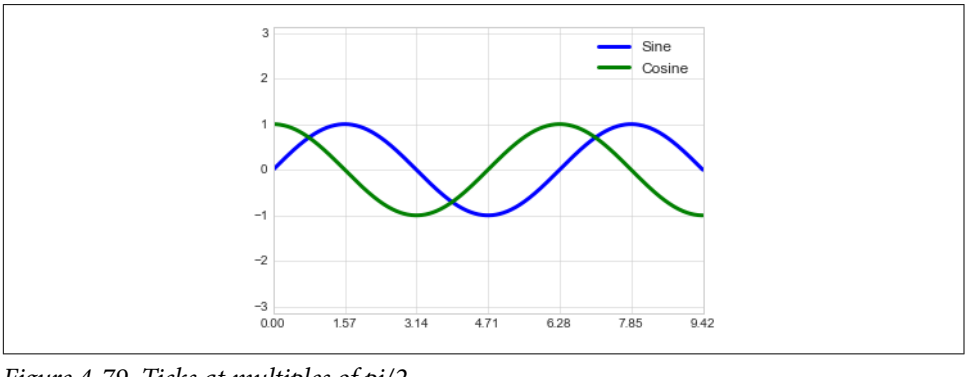


*Figure 4-79. Ticks at multiples of pi/2*

But now these tick labels look a little bit silly: we can see that they are multiples of $\pi$, but the decimal representation does not immediately convey this. To fix this, we can change the tick formatter. There's no built-in formatter for what we want to do, so we'll instead use `plt.FuncFormatter`, which accepts a user-defined function giving fine-grained control over the tick outputs (Figure 4-80):

```
In[11]: def format_func(value, tick_number):
            # find number of multiples of pi/2
            N = int(np.round(2 * value / np.pi))
            if N == 0:
                return "0"
```

```
        elif N == 1:
            return r"$\pi/2$"
        elif N == 2:
            return r"$\pi$"
        elif N % 2 > 0:
            return r"${0}\pi/2$".format(N)
        else:
            return r"${0}\pi$".format(N // 2)

    ax.xaxis.set_major_formatter(plt.FuncFormatter(format_func))
    fig
```

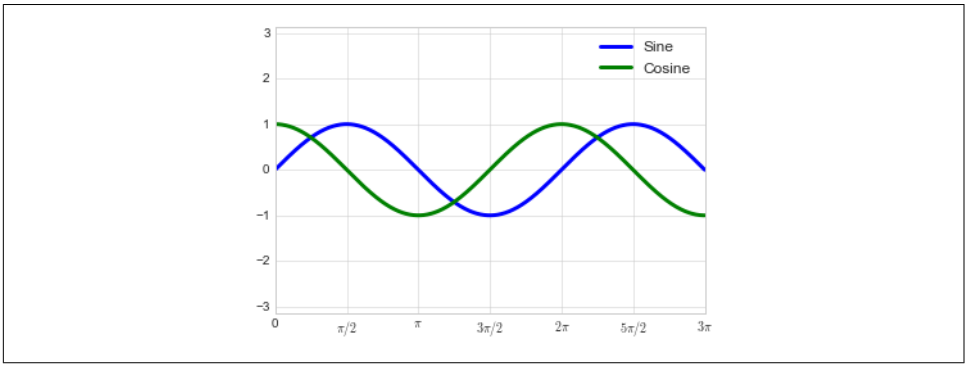


*Figure 4-80. Ticks with custom labels*

This is much better! Notice that we've made use of Matplotlib's LaTeX support, specified by enclosing the string within dollar signs. This is very convenient for display of mathematical symbols and formulae; in this case, `"$\pi$"` is rendered as the Greek character $\pi$.

The `plt.FuncFormatter()` offers extremely fine-grained control over the appearance of your plot ticks, and comes in very handy when you're preparing plots for presentation or publication.

## Summary of Formatters and Locators

We've mentioned a couple of the available formatters and locators. We'll conclude this section by briefly listing all the built-in locator and formatter options. For more information on any of these, refer to the docstrings or to the Matplotlib online documentation. Each of the following is available in the `plt` namespace:

| Locator class | Description |
|---|---|
| `NullLocator` | No ticks |
| `FixedLocator` | Tick locations are fixed |
| `IndexLocator` | Locator for index plots (e.g., where `x = range(len(y))`) |