

modern APIs—for example, Seaborn (discussed in [“Visualization with Seaborn” on page 311](#)), ggplot, HoloViews, Altair, and even Pandas itself can be used as wrappers around Matplotlib’s API. Even with wrappers like these, it is still often useful to dive into Matplotlib’s syntax to adjust the final plot output. For this reason, I believe that Matplotlib itself will remain a vital piece of the data visualization stack, even if new tools mean the community gradually moves away from using the Matplotlib API directly.

General Matplotlib Tips

Before we dive into the details of creating visualizations with Matplotlib, there are a few useful things you should know about using the package.

Importing matplotlib

Just as we use the `np` shorthand for NumPy and the `pd` shorthand for Pandas, we will use some standard shorthands for Matplotlib imports:

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

The `plt` interface is what we will use most often, as we’ll see throughout this chapter.

Setting Styles

We will use the `plt.style` directive to choose appropriate aesthetic styles for our figures. Here we will set the `classic` style, which ensures that the plots we create use the classic Matplotlib style:

```
In[2]: plt.style.use('classic')
```

Throughout this section, we will adjust this style as needed. Note that the stylesheets used here are supported as of Matplotlib version 1.5; if you are using an earlier version of Matplotlib, only the default style is available. For more information on stylesheets, see [“Customizing Matplotlib: Configurations and Stylesheets” on page 282](#).

`show()` or No `show()`? How to Display Your Plots

A visualization you can’t see won’t be of much use, but just how you view your Matplotlib plots depends on the context. The best use of Matplotlib differs depending on how you are using it; roughly, the three applicable contexts are using Matplotlib in a script, in an IPython terminal, or in an IPython notebook.