Controlling figure aesthetics

Drawing attractive figures is important. When making figures for yourself, as you explore a dataset, it's nice to have plots that are pleasant to look at. Visualizations are also central to communicating quantitative insights to an audience, and in that setting it's even more necessary to have figures that catch the attention and draw a viewer in.

Matplotlib is highly customizable, but it can be hard to know what settings to tweak to achieve an attractive plot. Seaborn comes with a number of customized themes and a high-level interface for controlling the look of matplotlib figures.

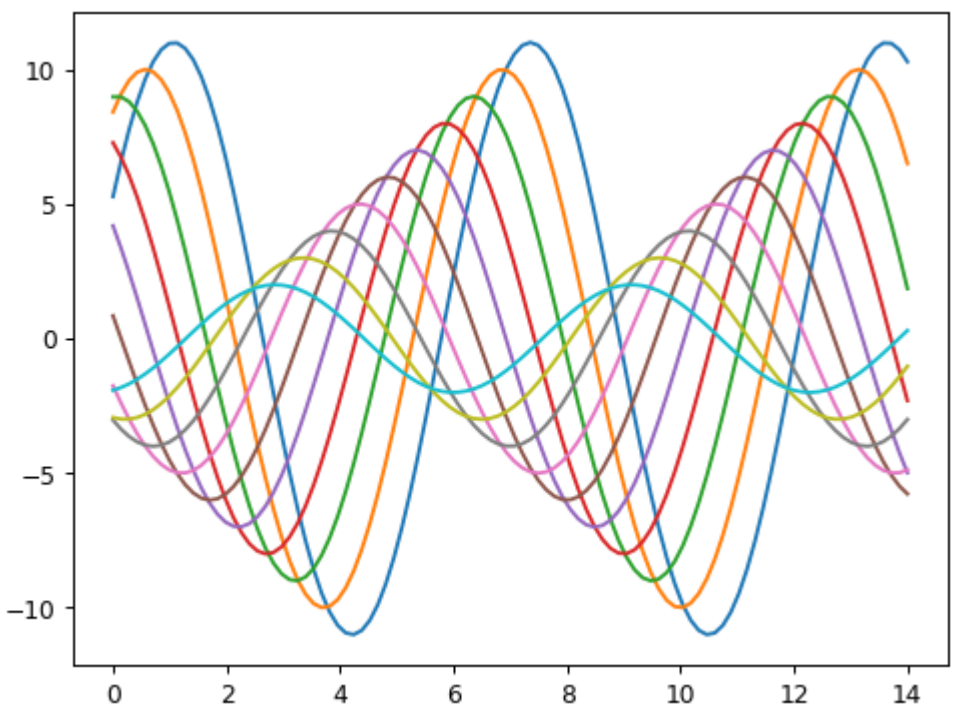
```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

Let's define a simple function to plot some offset sine waves, which will help us see the different stylistic parameters we can tweak.

```
def sinplot(n=10, flip=1):
    x = np.linspace(0, 14, 100)
    for i in range(1, n + 1):
        plt.plot(x, np.sin(x + i * .5) * (n + 2 - i) * flip)
```

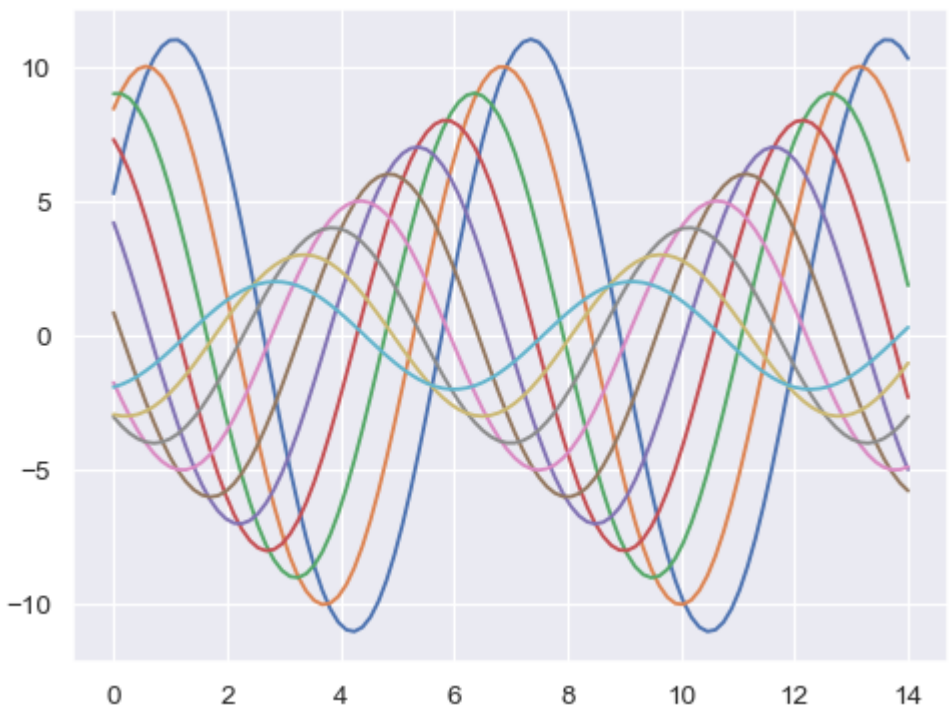
This is what the plot looks like with matplotlib defaults:

```
sinplot()
```



To switch to seaborn defaults, simply call the `set_theme()` function.

```
sns.set_theme()
sinplot()
```



(Note that in versions of seaborn prior to 0.8, `set_theme()` was called on import. On later versions, it must be explicitly invoked).

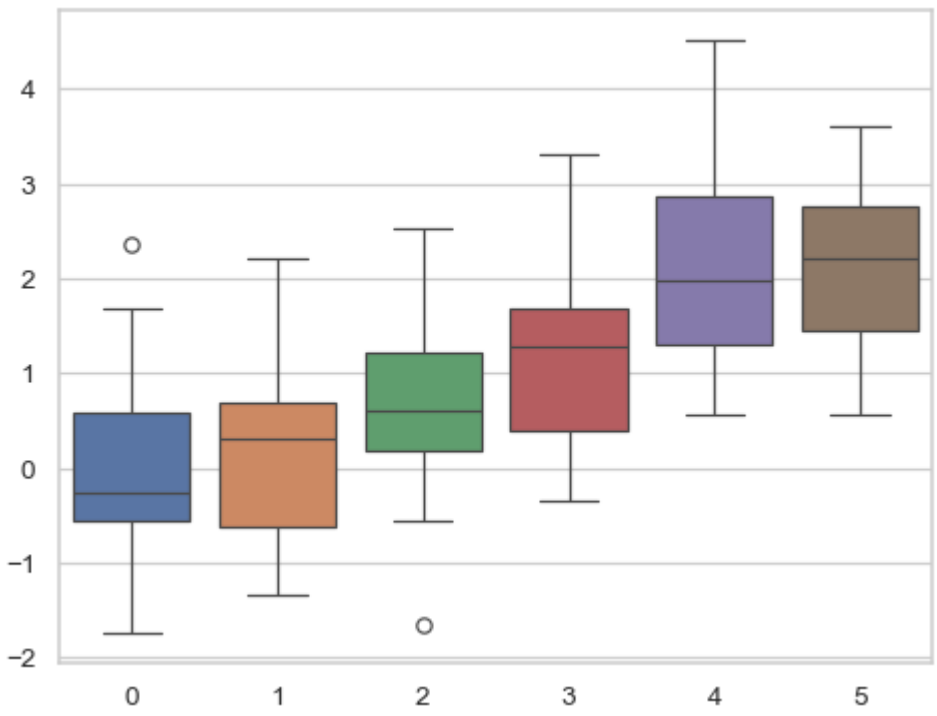
Seaborn splits matplotlib parameters into two independent groups. The first group sets the aesthetic style of the plot, and the second scales various elements of the figure so that it can be easily incorporated into different contexts.

The interface for manipulating these parameters are two pairs of functions. To control the style, use the `axes_style()` and `set_style()` functions. To scale the plot, use the `plotting_context()` and `set_context()` functions. In both cases, the first function returns a dictionary of parameters and the second sets the matplotlib defaults.

Seaborn figure styles

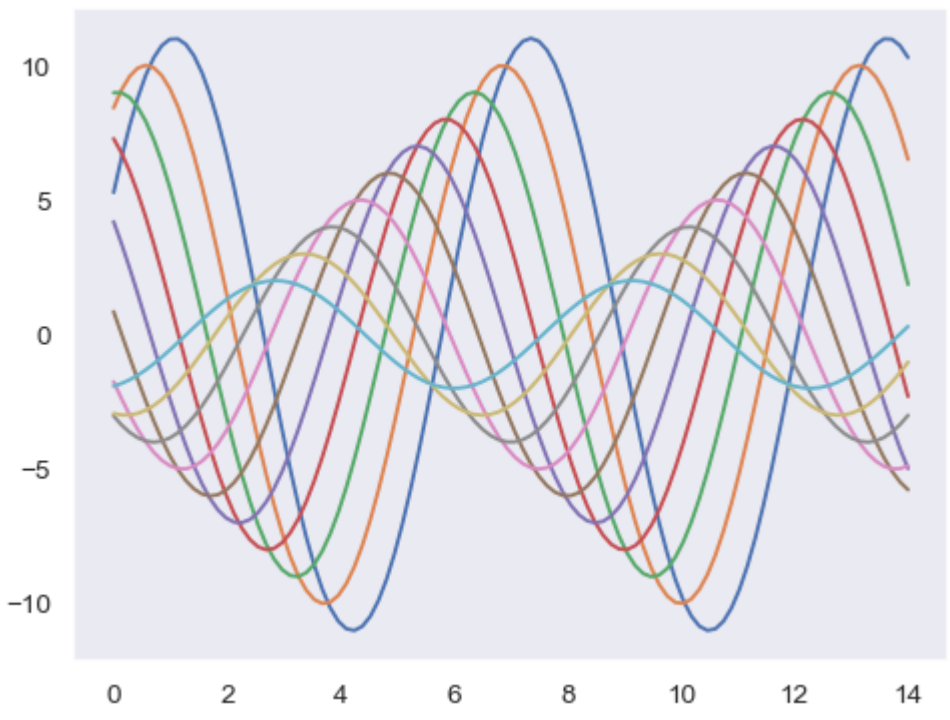
There are five preset seaborn themes: `darkgrid`, `whitegrid`, `dark`, `white`, and `ticks`. They are each suited to different applications and personal preferences. The default theme is `darkgrid`. As mentioned above, the grid helps the plot serve as a lookup table for quantitative information, and the white-on grey helps to keep the grid from competing with lines that represent data. The `whitegrid` theme is similar, but it is better suited to plots with heavy data elements:

```
sns.set_style("whitegrid")
data = np.random.normal(size=(20, 6)) + np.arange(6) / 2
sns.boxplot(data=data);
```

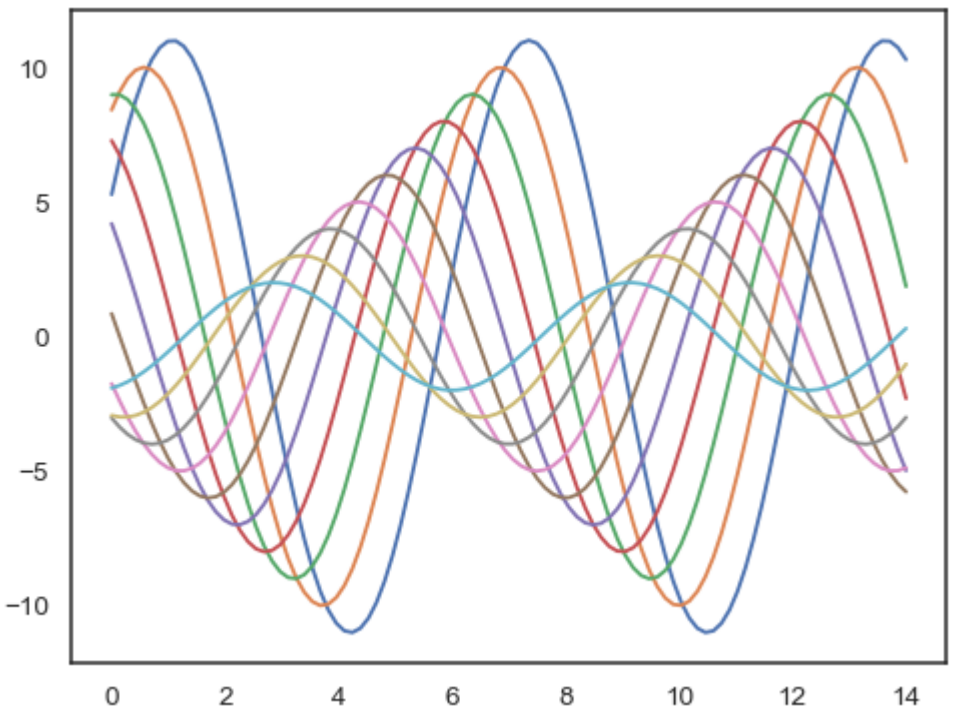


For many plots, (especially for settings like talks, where you primarily want to use figures to provide impressions of patterns in the data), the grid is less necessary.

```
sns.set_style("dark")
sinplot()
```

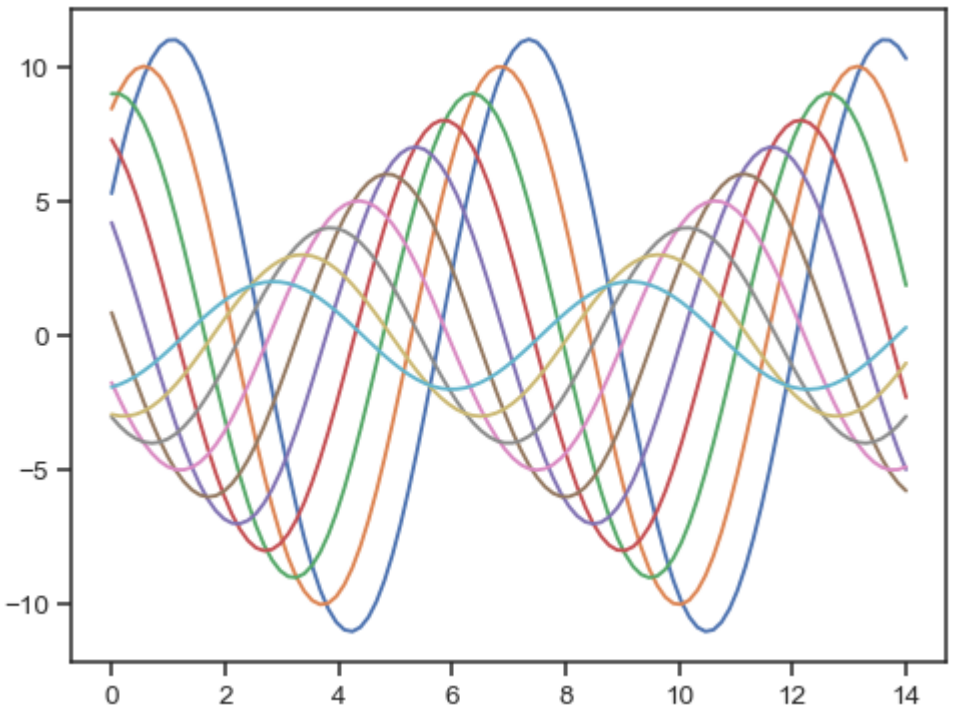



```
sns.set_style("white")
sinplot()
```



Sometimes you might want to give a little extra structure to the plots, which is where ticks come in handy:

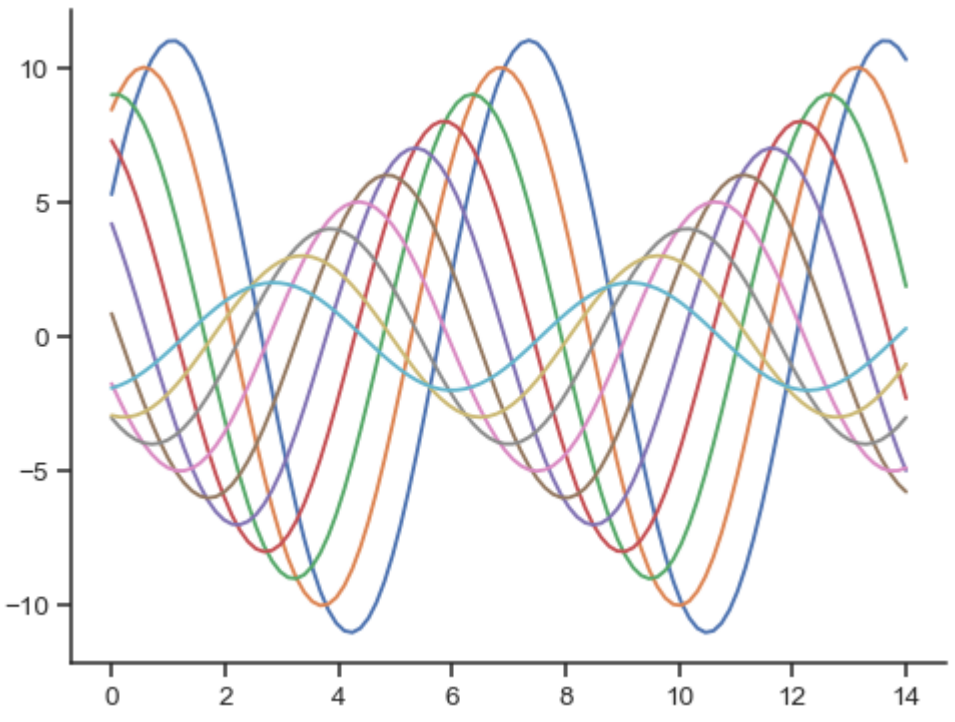
```
sns.set_style("ticks")
sinplot()
```



Removing axes spines

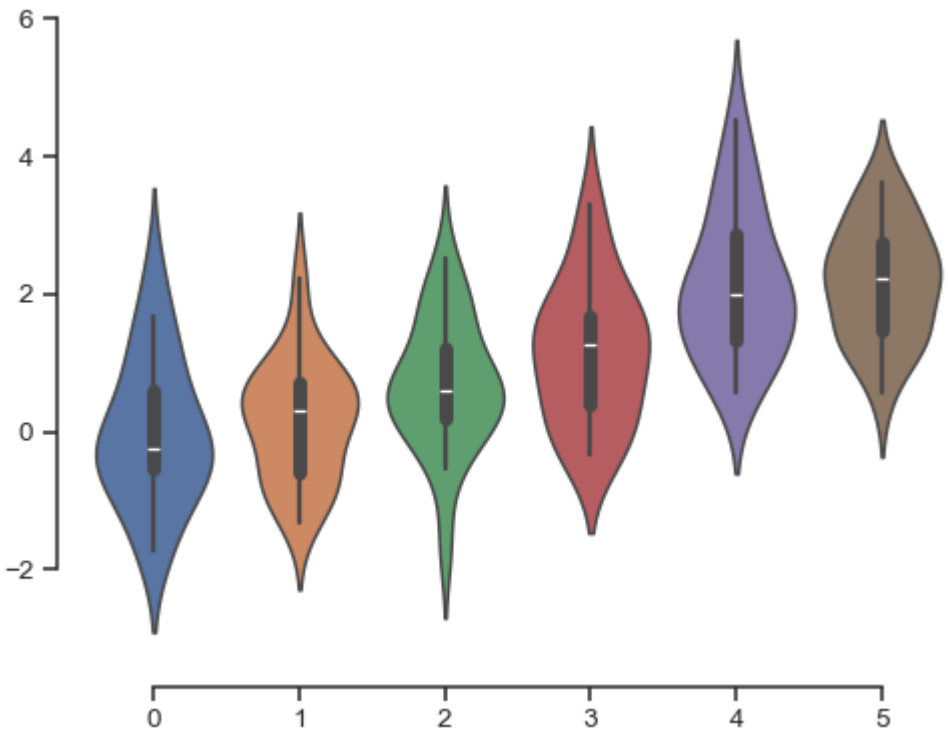
Both the `white` and `ticks` styles can benefit from removing the top and right axes spines, which are not needed. The seaborn function `despine()` can be called to remove them:

```
sinplot()
sns.despine()
```



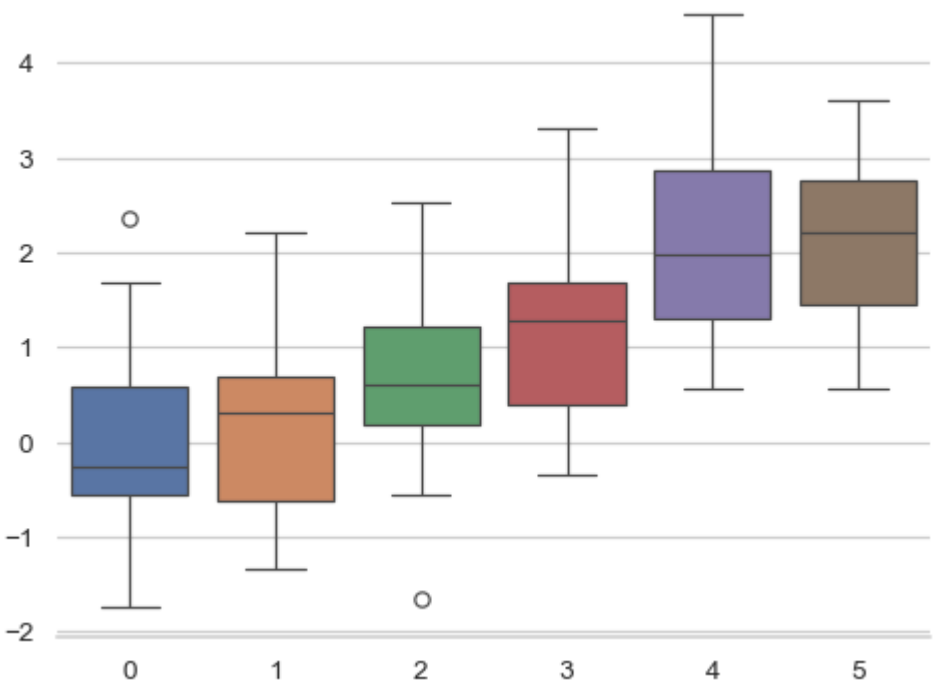
Some plots benefit from offsetting the spines away from the data, which can also be done when calling `despine()`. When the ticks don't cover the whole range of the axis, the `trim` parameter will limit the range of the surviving spines.

```
f, ax = plt.subplots()
sns.violinplot(data=data)
sns.despine(offset=10, trim=True);
```



You can also control which spines are removed with additional arguments to `despine()`:

```
sns.set_style("whitegrid")
sns.boxplot(data=data, palette="deep")
sns.despine(left=True)
```



Temporarily setting figure style

Although it's easy to switch back and forth, you can also use the `axes_style()` function in a `with` statement to temporarily set plot parameters. This also allows you to make figures with differently-styled axes:

```
f = plt.figure(figsize=(6, 6))
gs = f.add_gridspec(2, 2)

with sns.axes_style("darkgrid"):
    ax = f.add_subplot(gs[0, 0])
    sinplot(6)

with sns.axes_style("white"):
    ax = f.add_subplot(gs[0, 1])
    sinplot(6)

with sns.axes_style("ticks"):
    ax = f.add_subplot(gs[1, 0])
    sinplot(6)

with sns.axes_style("whitegrid"):
    ax = f.add_subplot(gs[1, 1])
    sinplot(6)

f.tight_layout()
```

Overriding elements of the seaborn styles

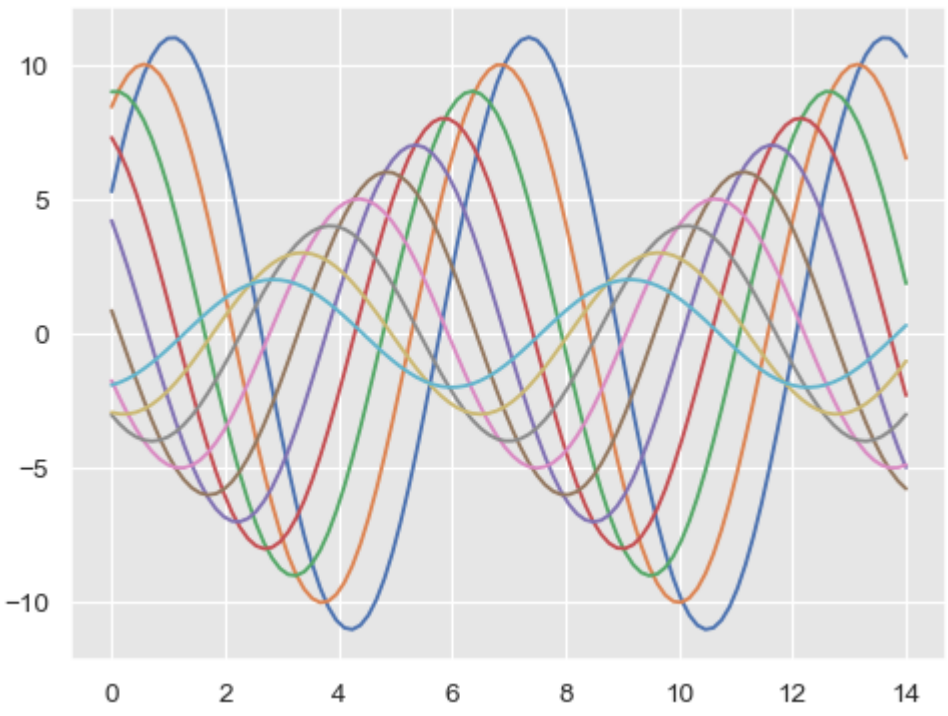
If you want to customize the seaborn styles, you can pass a dictionary of parameters to the `rc` argument of `axes_style()` and `set_style()`. Note that you can only override the parameters that are part of the style definition through this method. (However, the higher-level `set_theme()` function takes a dictionary of any matplotlib parameters).

If you want to see what parameters are included, you can just call the function with no arguments, which will return the current settings:

```
sns.axes_style()
```

You can then set different versions of these parameters:

```
sns.set_style("darkgrid", {"axes.facecolor": ".9"})
sinplot()
```



Scaling plot elements

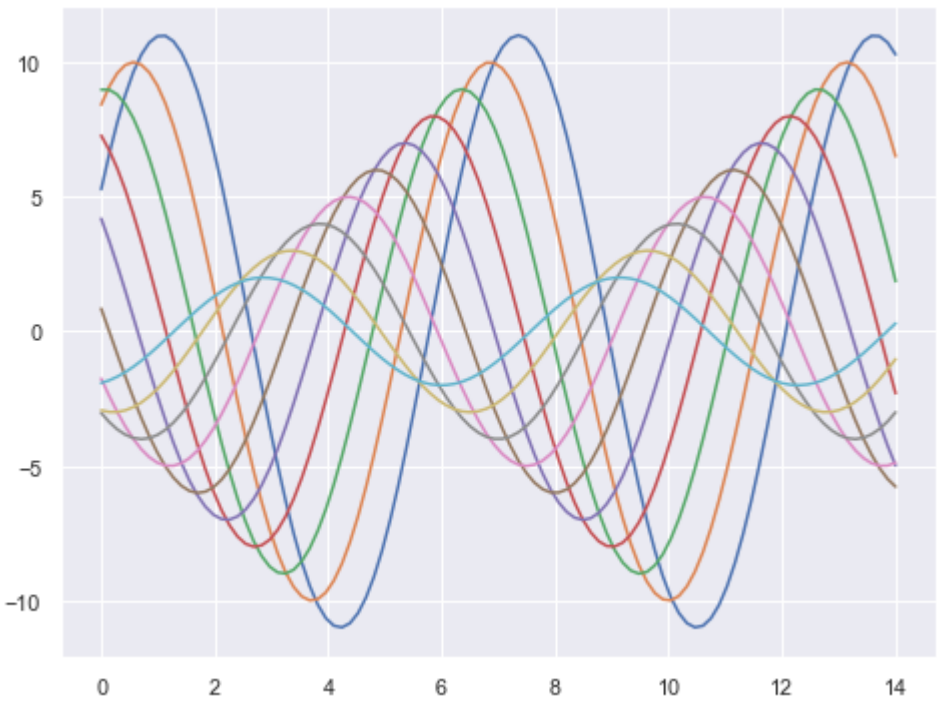
A separate set of parameters control the scale of plot elements, which should let you use the same code to make plots that are suited for use in settings where larger or smaller plots are appropriate.

First let's reset the default parameters by calling `set_theme()`:

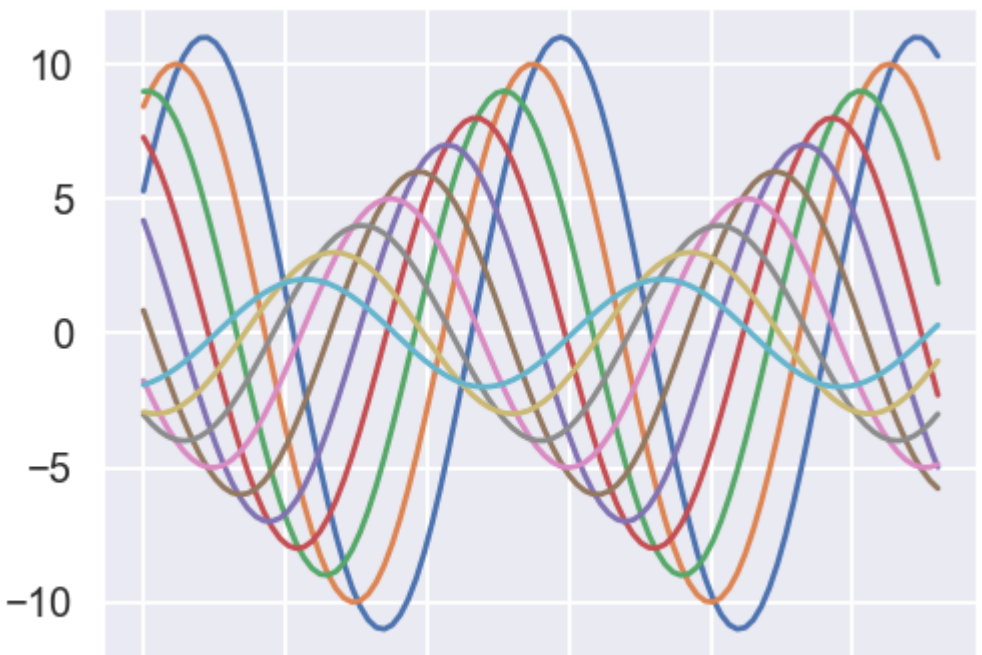
```
sns.set_theme()
```

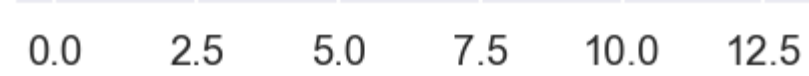
The four preset contexts, in order of relative size, are `paper`, `notebook`, `talk`, and `poster`. The `notebook` style is the default, and was used in the plots above.

```
sns.set_context("paper")
sinplot()
```

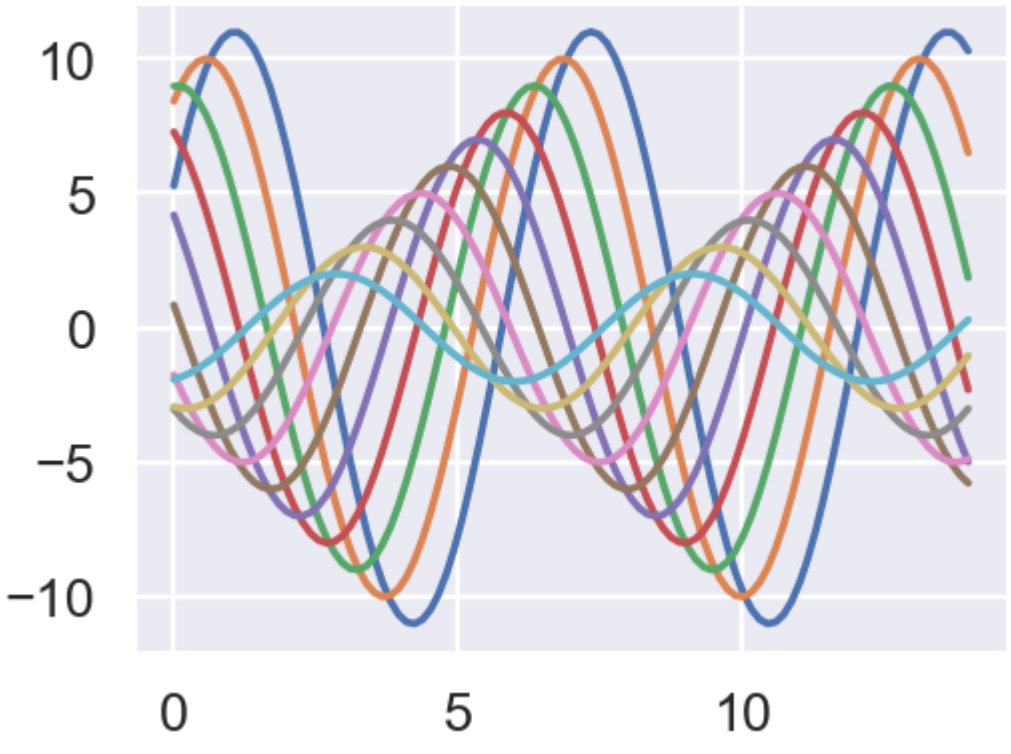



```
sns.set_context("talk")
sinplot()
```






```
sns.set_context("poster")
sinplot()
```

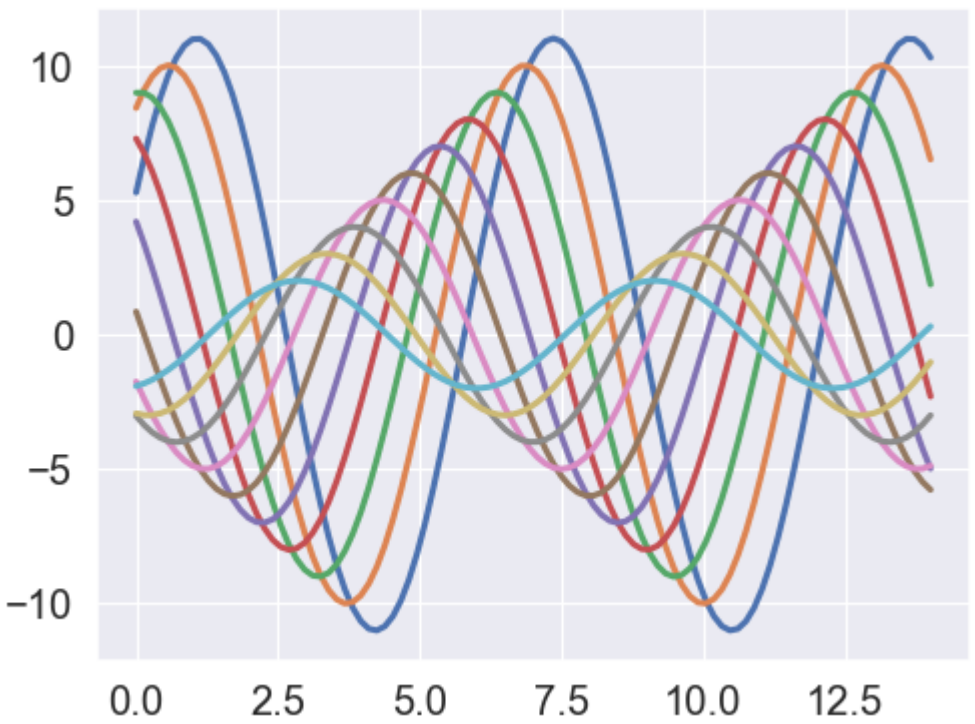


Most of what you now know about the style functions should transfer to the context functions.

You can call `set_context()` with one of these names to set the parameters, and you can override the parameters by providing a dictionary of parameter values.

You can also independently scale the size of the font elements when changing the context. (This option is also available through the top-level `set()` function).

```
sns.set_context("notebook", font_scale=1.5, rc={"lines.linewidth": 2.5})
sinplot()
```



Similarly, you can temporarily control the scale of figures nested under a `with` statement.

Both the style and the context can be quickly configured with the `set()` function. This function also sets the default color palette, but that will be covered in more detail in the next section of the tutorial.