By plotting empty lists, we create labeled plot objects that are picked up by the legend, and now our legend tells us some useful information. This strategy can be useful for creating more sophisticated visualizations.

Finally, note that for geographic data like this, it would be clearer if we could show state boundaries or other map-specific elements. For this, an excellent choice of tool is Matplotlib's Basemap add-on toolkit, which we'll explore in "Geographic Data with Basemap" on page 298.

Multiple Legends

Sometimes when designing a plot you'd like to add multiple legends to the same axes. Unfortunately, Matplotlib does not make this easy: via the standard `legend` interface, it is only possible to create a single legend for the entire plot. If you try to create a second legend using `plt.legend()` or `ax.legend()`, it will simply override the first one. We can work around this by creating a new legend artist from scratch, and then using the lower-level `ax.add_artist()` method to manually add the second artist to the plot (Figure 4-48):

```
In[10]: fig, ax = plt.subplots()

        lines = []
        styles = ['-', '--', '-.', ':']
        x = np.linspace(0, 10, 1000)

        for i in range(4):
            lines += ax.plot(x, np.sin(x - i * np.pi / 2),
                             styles[i], color='black')
        ax.axis('equal')

        # specify the lines and labels of the first legend
        ax.legend(lines[:2], ['line A', 'line B'],
                  loc='upper right', frameon=False)

        # Create the second legend and add the artist manually.
        from matplotlib.legend import Legend
        leg = Legend(ax, lines[2:], ['line C', 'line D'],
                     loc='lower right', frameon=False)
        ax.add_artist(leg);
```

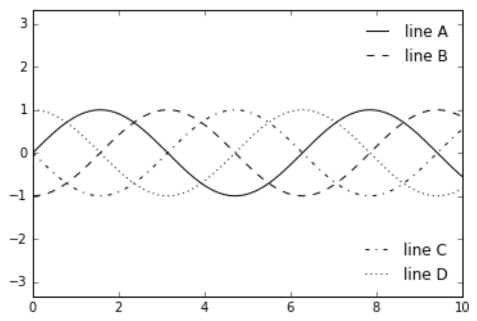

Figure 4-48. A split plot legend

This is a peek into the low-level artist objects that compose any Matplotlib plot. If you examine the source code of `ax.legend()` (recall that you can do this within the IPython notebook using `ax.legend??`) you'll see that the function simply consists of some logic to create a suitable `Legend` artist, which is then saved in the `legend_` attribute and added to the figure when the plot is drawn.

Customizing Colorbars

Plot legends identify discrete labels of discrete points. For continuous labels based on the color of points, lines, or regions, a labeled colorbar can be a great tool. In Matplotlib, a colorbar is a separate axes that can provide a key for the meaning of colors in a plot. Because the book is printed in black and white, this section has an accompanying online appendix where you can view the figures in full color (*https://github.com/jakevdp/PythonDataScienceHandbook*). We'll start by setting up the notebook for plotting and importing the functions we will use:

```
In[1]: import matplotlib.pyplot as plt
       plt.style.use('classic')
```

```
In[2]: %matplotlib inline
       import numpy as np
```

As we have seen several times throughout this section, the simplest colorbar can be created with the `plt.colorbar` function (Figure 4-49):

```
In[3]: x = np.linspace(0, 10, 1000)
       I = np.sin(x) * np.cos(x[:, np.newaxis])

       plt.imshow(I)
       plt.colorbar();
```