

Simple Scatter Plots

Another commonly used plot type is the simple scatter plot, a close cousin of the line plot. Instead of points being joined by line segments, here the points are represented individually with a dot, circle, or other shape. We'll start by setting up the notebook for plotting and importing the functions we will use:

```
In[1]: %matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
import numpy as np
```

Scatter Plots with plt.plot

In the previous section, we looked at `plt.plot/ax.plot` to produce line plots. It turns out that this same function can produce scatter plots as well (Figure 4-20):

```
In[2]: x = np.linspace(0, 10, 30)
y = np.sin(x)

plt.plot(x, y, 'o', color='black');
```

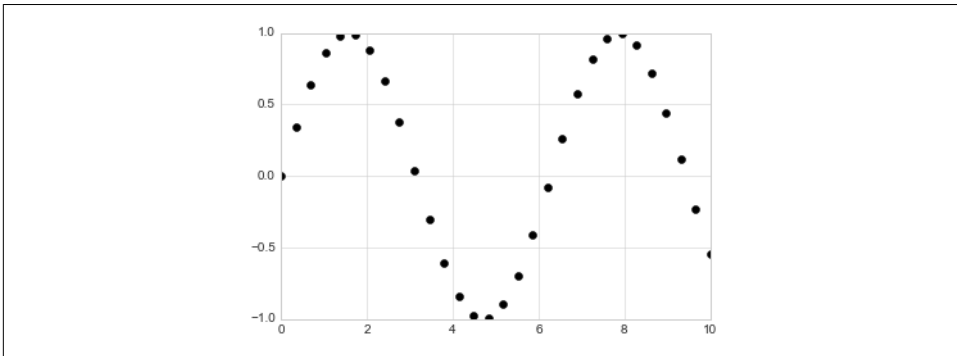


Figure 4-20. Scatter plot example

The third argument in the function call is a character that represents the type of symbol used for the plotting. Just as you can specify options such as '-' and '--' to control the line style, the marker style has its own set of short string codes. The full list of available symbols can be seen in the documentation of `plt.plot`, or in Matplotlib's online documentation. Most of the possibilities are fairly intuitive, and we'll show a number of the more common ones here (Figure 4-21):

```
In[3]: rng = np.random.RandomState(0)
for marker in ['o', '.', ',', 'x', '+', 'v', '^', '<', '>', 's', 'd']:
    plt.plot(rng.rand(5), rng.rand(5), marker,
             label="marker='{0}'".format(marker))
```

```
plt.legend(numpoints=1)
plt.xlim(0, 1.8);
```

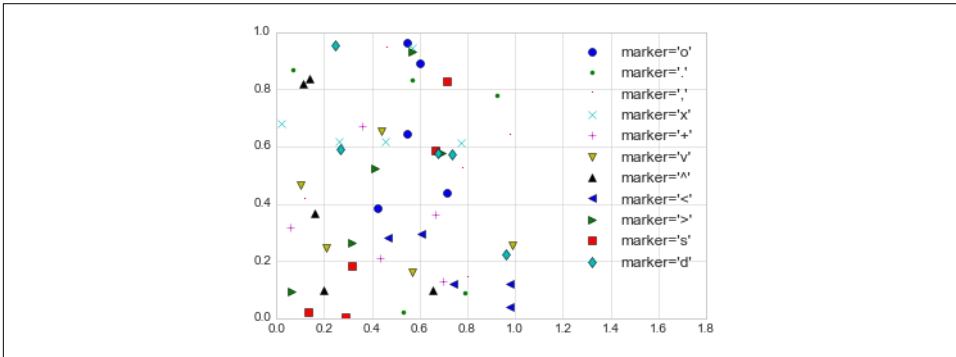


Figure 4-21. Demonstration of point numbers

For even more possibilities, these character codes can be used together with line and color codes to plot points along with a line connecting them (Figure 4-22):

```
In[4]: plt.plot(x, y, '-ok'); # line (-), circle marker (o), black (k)
```

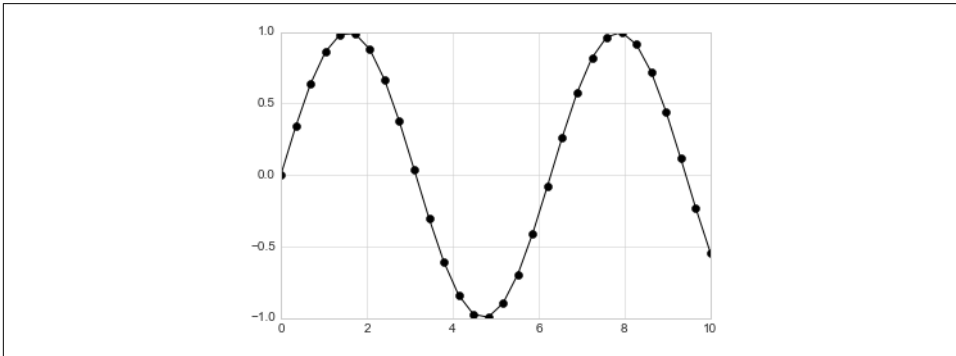


Figure 4-22. Combining line and point markers

Additional keyword arguments to `plt.plot` specify a wide range of properties of the lines and markers (Figure 4-23):

```
In[5]: plt.plot(x, y, '-p', color='gray',
               markersize=15, linewidth=4,
               markerfacecolor='white',
               markeredgecolor='gray',
               markeredgewidth=2)
plt.ylim(-1.2, 1.2);
```

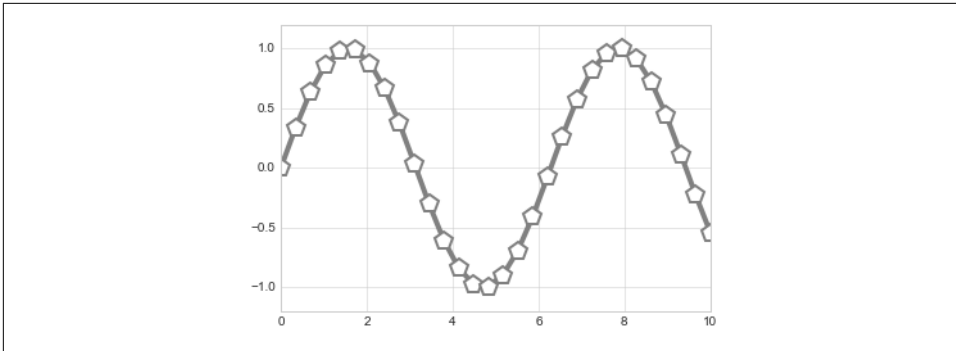


Figure 4-23. Customizing line and point numbers

This type of flexibility in the `plt.plot` function allows for a wide variety of possible visualization options. For a full description of the options available, refer to the `plt.plot` documentation.

Scatter Plots with `plt.scatter`

A second, more powerful method of creating scatter plots is the `plt.scatter` function, which can be used very similarly to the `plt.plot` function (Figure 4-24):

```
In[6]: plt.scatter(x, y, marker='o');
```

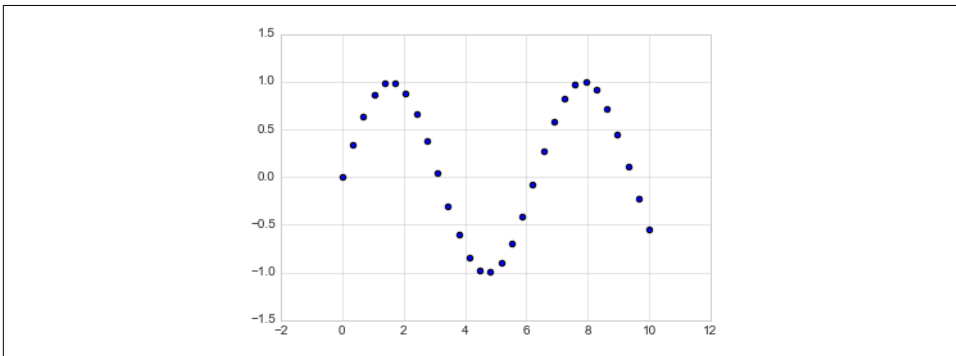


Figure 4-24. A simple scatter plot

The primary difference of `plt.scatter` from `plt.plot` is that it can be used to create scatter plots where the properties of each individual point (size, face color, edge color, etc.) can be individually controlled or mapped to data.

Let's show this by creating a random scatter plot with points of many colors and sizes. In order to better see the overlapping results, we'll also use the `alpha` keyword to adjust the transparency level (Figure 4-25):