Out[2]:

```
x:
[[1 2 3]
 [4 5 6]]
```

We will be using NumPy *a lot* in this book, and we will refer to objects of the NumPy `ndarray` class as "NumPy arrays" or just "arrays."

SciPy

SciPy is a collection of functions for scientific computing in Python. It provides, among other functionality, advanced linear algebra routines, mathematical function optimization, signal processing, special mathematical functions, and statistical distributions. `scikit-learn` draws from SciPy's collection of functions for implementing its algorithms. The most important part of SciPy for us is `scipy.sparse`: this provides *sparse matrices*, which are another representation that is used for data in `scikit-learn`. Sparse matrices are used whenever we want to store a 2D array that contains mostly zeros:

In[3]:

```python
from scipy import sparse

# Create a 2D NumPy array with a diagonal of ones, and zeros everywhere else
eye = np.eye(4)
print("NumPy array:\n{}".format(eye))
```

Out[3]:

```
NumPy array:
[[ 1.  0.  0.  0.]
 [ 0.  1.  0.  0.]
 [ 0.  0.  1.  0.]
 [ 0.  0.  0.  1.]]
```

In[4]:

```python
# Convert the NumPy array to a SciPy sparse matrix in CSR format
# Only the nonzero entries are stored
sparse_matrix = sparse.csr_matrix(eye)
print("\nSciPy sparse CSR matrix:\n{}".format(sparse_matrix))
```

Out[4]:

```
SciPy sparse CSR matrix:
  (0, 0)    1.0
  (1, 1)    1.0
  (2, 2)    1.0
  (3, 3)    1.0
```

Usually it is not possible to create dense representations of sparse data (as they would not fit into memory), so we need to create sparse representations directly. Here is a way to create the same sparse matrix as before, using the COO format:

In[5]:

```
data = np.ones(4)
row_indices = np.arange(4)
col_indices = np.arange(4)
eye_coo = sparse.coo_matrix((data, (row_indices, col_indices)))
print("COO representation:\n{}".format(eye_coo))
```

Out[5]:

```
COO representation:
  (0, 0)    1.0
  (1, 1)    1.0
  (2, 2)    1.0
  (3, 3)    1.0
```

More details on SciPy sparse matrices can be found in the SciPy Lecture Notes.

matplotlib

`matplotlib` is the primary scientific plotting library in Python. It provides functions for making publication-quality visualizations such as line charts, histograms, scatter plots, and so on. Visualizing your data and different aspects of your analysis can give you important insights, and we will be using `matplotlib` for all our visualizations. When working inside the Jupyter Notebook, you can show figures directly in the browser by using the `%matplotlib notebook` and `%matplotlib inline` commands. We recommend using `%matplotlib notebook`, which provides an interactive environment (though we are using `%matplotlib inline` to produce this book). For example, this code produces the plot in Figure 1-1:

In[6]:

```
%matplotlib inline
import matplotlib.pyplot as plt

# Generate a sequence of numbers from -10 to 10 with 100 steps in between
x = np.linspace(-10, 10, 100)
# Create a second array using sine
y = np.sin(x)
# The plot function makes a line chart of one array against another
plt.plot(x, y, marker="x")
```