

Figure 2-5. Visualization of a 2D array

## Comparisons, Masks, and Boolean Logic

This section covers the use of Boolean masks to examine and manipulate values within NumPy arrays. Masking comes up when you want to extract, modify, count, or otherwise manipulate values in an array based on some criterion: for example, you might wish to count all values greater than a certain value, or perhaps remove all outliers that are above some threshold. In NumPy, Boolean masking is often the most efficient way to accomplish these types of tasks.

### Example: Counting Rainy Days

Imagine you have a series of data that represents the amount of precipitation each day for a year in a given city. For example, here we'll load the daily rainfall statistics for the city of Seattle in 2014, using Pandas (which is covered in more detail in [Chapter 3](#)):

```
In[1]: import numpy as np
import pandas as pd

# use Pandas to extract rainfall inches as a NumPy array
rainfall = pd.read_csv('data/Seattle2014.csv')['PRCP'].values
inches = rainfall / 254 # 1/10mm -> inches
inches.shape

Out[1]: (365,)
```

The array contains 365 values, giving daily rainfall in inches from January 1 to December 31, 2014.

As a first quick visualization, let's look at the histogram of rainy days shown in [Figure 2-6](#), which was generated using Matplotlib (we will explore this tool more fully in [Chapter 4](#)):

```
In[2]: %matplotlib inline
import matplotlib.pyplot as plt
import seaborn; seaborn.set() # set plot styles

In[3]: plt.hist(inches, 40);
```

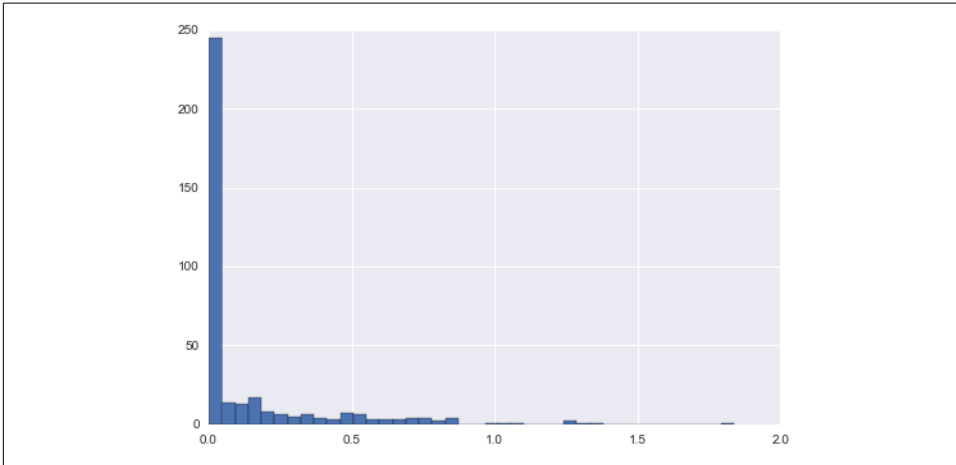


Figure 2-6. Histogram of 2014 rainfall in Seattle

This histogram gives us a general idea of what the data looks like: despite its reputation, the vast majority of days in Seattle saw near zero measured rainfall in 2014. But this doesn't do a good job of conveying some information we'd like to see: for example, how many rainy days were there in the year? What is the average precipitation on those rainy days? How many days were there with more than half an inch of rain?

### Digging into the data

One approach to this would be to answer these questions by hand: loop through the data, incrementing a counter each time we see values in some desired range. For reasons discussed throughout this chapter, such an approach is very inefficient, both from the standpoint of time writing code and time computing the result. We saw in [“Computation on NumPy Arrays: Universal Functions” on page 50](#) that NumPy's ufuncs can be used in place of loops to do fast element-wise arithmetic operations on arrays; in the same way, we can use other ufuncs to do element-wise *comparisons* over arrays, and we can then manipulate the results to answer the questions we have. We'll leave the data aside for right now, and discuss some general tools in NumPy to use *masking* to quickly answer these types of questions.

## Comparison Operators as ufuncs

In [“Computation on NumPy Arrays: Universal Functions” on page 50](#) we introduced ufuncs, and focused in particular on arithmetic operators. We saw that using `+`, `-`, `*`, `/`,