first axis will be collapsed: for two-dimensional arrays, this means that values within each column will be aggregated.

Other aggregation functions

NumPy provides many other aggregation functions, but we won't discuss them in detail here. Additionally, most aggregates have a `NaN`-safe counterpart that computes the result while ignoring missing values, which are marked by the special IEEE floating-point `NaN` value (for a fuller discussion of missing data, see "Handling Missing Data" on page 119). Some of these `NaN`-safe functions were not added until NumPy 1.8, so they will not be available in older NumPy versions.

Table 2-3 provides a list of useful aggregation functions available in NumPy.

Table 2-3. Aggregation functions available in NumPy

Function Name	NaN-safe Version	Description
`np.sum`	`np.nansum`	Compute sum of elements
`np.prod`	`np.nanprod`	Compute product of elements
`np.mean`	`np.nanmean`	Compute median of elements
`np.std`	`np.nanstd`	Compute standard deviation
`np.var`	`np.nanvar`	Compute variance
`np.min`	`np.nanmin`	Find minimum value
`np.max`	`np.nanmax`	Find maximum value
`np.argmin`	`np.nanargmin`	Find index of minimum value
`np.argmax`	`np.nanargmax`	Find index of maximum value
`np.median`	`np.nanmedian`	Compute median of elements
`np.percentile`	`np.nanpercentile`	Compute rank-based statistics of elements
`np.any`	N/A	Evaluate whether any elements are true
`np.all`	N/A	Evaluate whether all elements are true

We will see these aggregates often throughout the rest of the book.

Example: What Is the Average Height of US Presidents?

Aggregates available in NumPy can be extremely useful for summarizing a set of values. As a simple example, let's consider the heights of all US presidents. This data is available in the file *president_heights.csv*, which is a simple comma-separated list of labels and values:

```
In[13]: !head -4 data/president_heights.csv

order,name,height(cm)
1,George Washington,189
```

```
2,John Adams,170
3,Thomas Jefferson,189
```

We'll use the Pandas package, which we'll explore more fully in Chapter 3, to read the file and extract this information (note that the heights are measured in centimeters):

```
In[14]: import pandas as pd
        data = pd.read_csv('data/president_heights.csv')
        heights = np.array(data['height(cm)'])
        print(heights)
```

```
[189 170 189 163 183 171 185 168 173 183 173 173 175 178 183 193 178 173
 174 183 183 168 170 178 182 180 183 178 182 188 175 179 183 193 182 183
 177 185 188 188 182 185]
```

Now that we have this data array, we can compute a variety of summary statistics:

```
In[15]: print("Mean height:       ", heights.mean())
        print("Standard deviation:", heights.std())
        print("Minimum height:    ", heights.min())
        print("Maximum height:    ", heights.max())
```

```
Mean height:        179.738095238
Standard deviation: 6.93184344275
Minimum height:     163
Maximum height:     193
```

Note that in each case, the aggregation operation reduced the entire array to a single summarizing value, which gives us information about the distribution of values. We may also wish to compute quantiles:

```
In[16]: print("25th percentile:   ", np.percentile(heights, 25))
        print("Median:            ", np.median(heights))
        print("75th percentile:   ", np.percentile(heights, 75))
```

```
25th percentile:    174.25
Median:             182.0
75th percentile:    183.0
```

We see that the median height of US presidents is 182 cm, or just shy of six feet.

Of course, sometimes it's more useful to see a visual representation of this data, which we can accomplish using tools in Matplotlib (we'll discuss Matplotlib more fully in Chapter 4). For example, this code generates the chart shown in Figure 2-3:

```
In[17]: %matplotlib inline
        import matplotlib.pyplot as plt
        import seaborn; seaborn.set()  # set plot style
```

```
In[18]: plt.hist(heights)
        plt.title('Height Distribution of US Presidents')
        plt.xlabel('height (cm)')
        plt.ylabel('number');
```

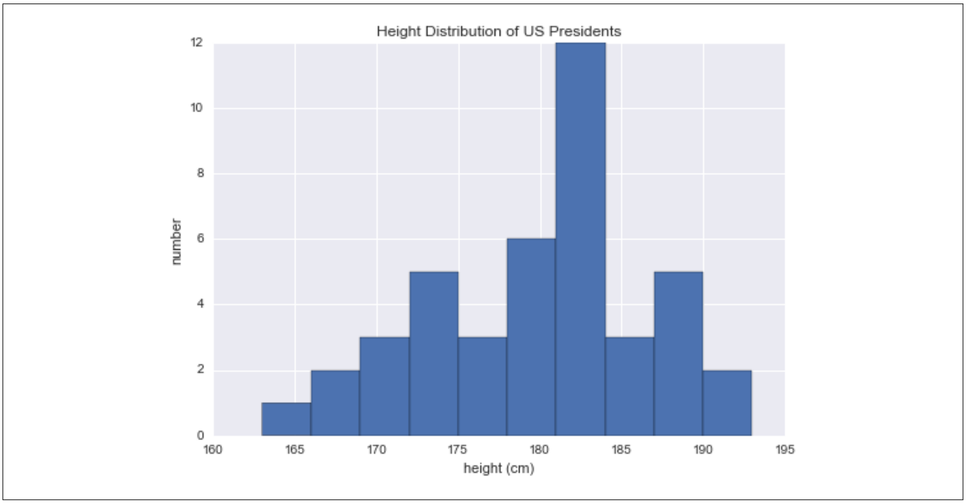

Figure 2-3. Histogram of presidential heights

These aggregates are some of the fundamental pieces of exploratory data analysis that we'll explore in more depth in later chapters of the book.

Computation on Arrays: Broadcasting

We saw in the previous section how NumPy's universal functions can be used to *vectorize* operations and thereby remove slow Python loops. Another means of vectorizing operations is to use NumPy's *broadcasting* functionality. Broadcasting is simply a set of rules for applying binary ufuncs (addition, subtraction, multiplication, etc.) on arrays of different sizes.

Introducing Broadcasting

Recall that for arrays of the same size, binary operations are performed on an element-by-element basis:

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

In[2]: a = np.array([0, 1, 2])
       b = np.array([5, 5, 5])
       a + b

Out[2]: array([5, 6, 7])
```

Broadcasting allows these types of binary operations to be performed on arrays of different sizes—for example, we can just as easily add a scalar (think of it as a zero-dimensional array) to an array: