집계: 최소값, 최댓값, 그리고 그사이의 모든 것

대용량 데이터에 직면했을 때 첫 번째단계는 궁금한 데이터에 대한 요약 통계를 계산하는 것이다. 가장 보편적인 요약 통계는 데이터세트의 '전형적인' 값을 요약할 수 있는 평균과 표준 편차겠지만 다른 집계 연산도 유용하다.(합, 곱, 중 앙값, 최댓값, .분위수 등)

배열의 값의 합 구하기

간단한 예로 배열 내 모든 것을 생각해 보자. 이때는 파이썬 자체에 내장된 함수 sum 을 사용할 수 있다.

```
In [1]:import numpy as np
In [2]:L = np.random.random(100)
        sum(L)
Out[2]:55.61209116604941
```

구문은 NumPy 의 sum 함수와 매우 유사하며, 가장 간단한 계산에서는 두 함수의 결과가 같다

```
In [3]:np.sum(L)
Out[3]:55.612091166049424
```

그러나 이 연산이 컴파일된 코드에서 실행되기 때문에 NumPy 에서의 연산이 훨씬 더 빠르다.

```
In [4]:big_array = np.random.rand(1000000)
        %timeit sum(big_array)
        %timeit np.sum(big_array)
150 ms ± 6.47 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
536 µs ± 20.7 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

그렇더라도 sum 함수와 np.sum 함수가 같은 함수는 아니라서 때로는 혼선을 일으킬 수 있다는 점에 주의하자. 특히 그 함수들의 선택적 인수는 다른 의미를 갖고 있다.

최소값과 최댓값¶

파이썬에는 배열의 최소값과 최대값을 찾는 데 사용하는 내장 함수인 min 과 max 가 있다.

```
In [5]:min(big_array), max(big_array)
Out[5]:(1.1717128136634614e-06, 0.9999976784968716)
```

이에 대응하는 NumPy 함수도 유사한 구문을 가지고 있으며, 그것 역시 훨씬 더 빨리 계산한다.

```
In [6]:np.min(big_array), np.max(big_array)
Out[6]:
(1.1717128136634614e-06, 0.9999976784968716)
In [7]:%timeit min(big_array)
     %timeit np.min(big_array)
103 ms ± 7.78 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
505 µs ± 5.96 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

min, max, sum 을 비롯한 다른 여러 NumPy 집계 함수의 경우, 배열 객체 자체의 메서드를 사용하는 더 잛은 구문이 존재한다.

```
In [8]:print(big_array.min(), big_array.max(), big_array.sum())
1.17171281366e-06 0.999997678497 499911.628197
```

NumPy 배열을 다룰 때는 가능한 이 NumPy 버전의 집계 함수를 사용하라.

다차원 집계¶

집계 연산의 보편적인 유형은 행이나 열을 기준으로 집계하는 것이다. 2 차원 배열에 저장된 데이터를 가지고 있다고 해보자.

```
In [9]:M = np.random.random((3, 4))
       print(M)
[[ 0.8967576   0.03783739  0.75952519  0.06682827]
 [ 0.8354065   0.99196818  0.19544769  0.43447084]
 [ 0.66859307  0.15038721  0.37911423  0.6687194 ]]

In [10]:M.sum()
Out[10]:6.0850555667307118
```

집계 함수는 어느 축(axis)을 따라 집계할 것인지를 지정하는 추가적인 인수를 취한다. 예를 들어 각 열의 최소값을 찾으려면 axis=0 으로 지정하면 된다.

```
In [11]:M.min(axis=0)
Out[11]:array([ 0.66859307,  0.03783739,  0.19544769,  0.06682827])
```

이 함수는 네 개의 열에 대응하는 값 네개를 반환한다.
비슷하게 각 행의 최소값을 찾을 수 있다.

```
In [12]:M.max(axis=1)
Out[12]:array([ 0.8967576 ,  0.99196818,  0.6687194 ])
```

여기서 축을 지정하는 방식은 다른 언어를 사용하던 사람이라면 혼란스러울 수 있다. Axis 키워드는 반환 차원이 아니라 축소할 배열의 차원을 지정한다. 따라서 axis=0 으로 지정하는 것은 첫 번째 축을 축소한다는 의미가 된다. 2 차원 배열이라면 각 열의 값들이 집계된다는 뜻이다.

기타 집계함수¶

NumPy 는 이 밖에도 수많은 집계 함수를 제공하지만, 여기서 자세히 다루지는 않을 것이다. 아울러 대부분의 집계 함수에는 누락된 값을 무시한 채 값을 계산하는 NaN 안전 모드가 있으며, 이 경우 IEEE 부동소수점 NaN 값으로 표시된다. 일부 NaN 안전 함수는 NumPy 1.8 버전까지는 존재하지 않았다.

NumPy 에서 사용할 수 있는 집계 함수:

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 mean 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

이 집계 함수들은 이 책의 나머지 부분에서도 종종 보게 될 것이다..

예제: 미국 대통령의 평균 신장은 얼마인가?¶

NumPy에서 사용 가능한 집계 함수는 일련의 값을 요약할 때 매우 유용하다. 가단한 예로, 역대 미국 대통령의 키를 생각해 보자. 이 데이터는 President_heights.csv 파일에 레이블과 값을 콤마로 구분한 간단한 목록 형태로 존재한다.

```
In [13]:!head -4 data/president_heights.csv
order,name,height(cm)
1,George Washington,189
2,John Adams,170
3,Thomas Jefferson,189
```

```
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]
```

이 데이터 배열이 있으니 이제 다양한 요약 통계를 계산할 수 있다.

```
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
```

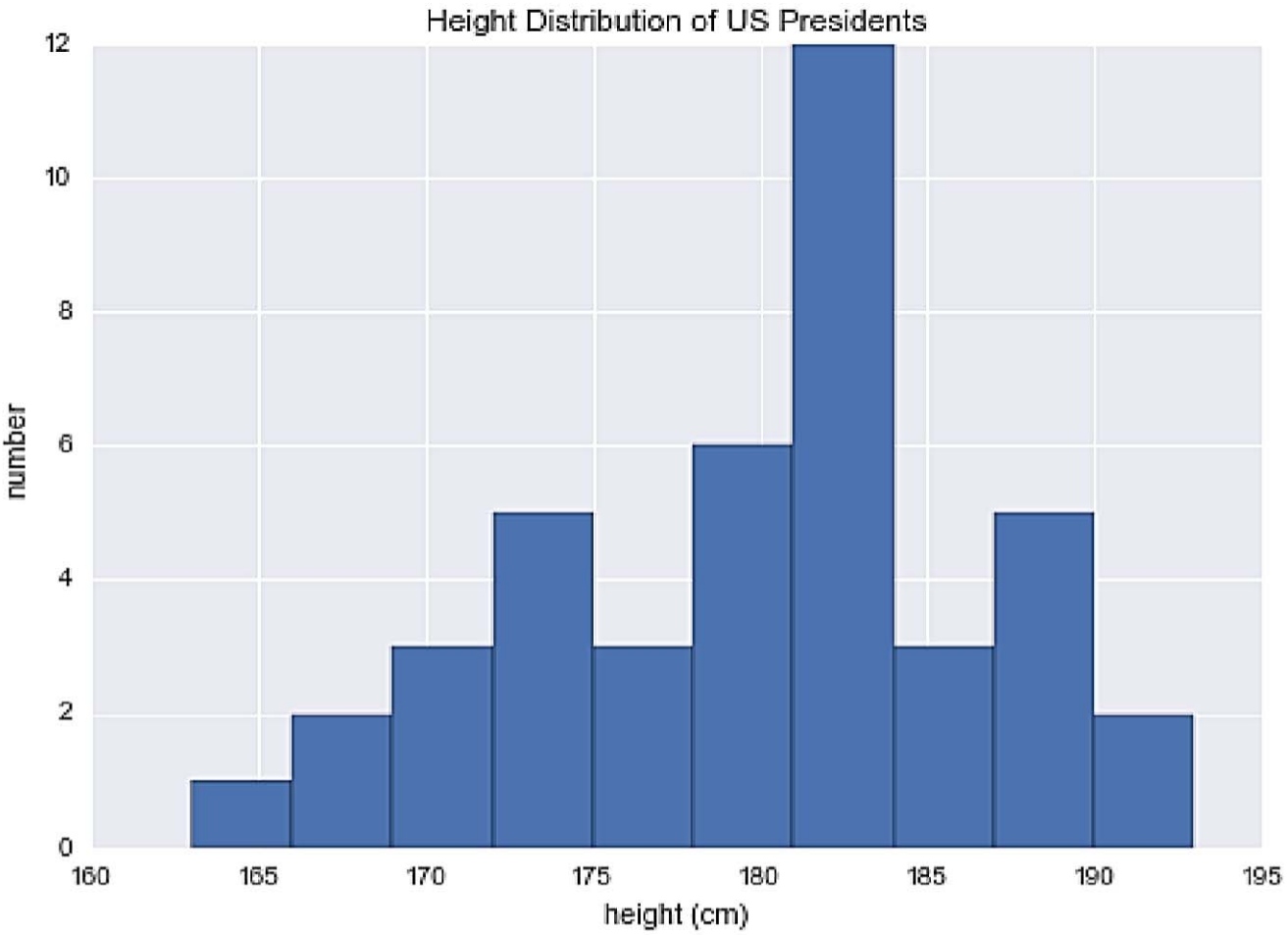
매번 집계 연산이 전체 배열을 하나의 요약 값으로 축소해서 배열 값의 분포에 대한 정보를 제공한다는 점을 알아두자. 또한 사분위수를 계산할 수도 있다.

```
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
```

미국 대통령 신장의 중 앙값은 182cm, 즉 6 피트에서 약간 모자란다는 것을 알 수 있다.

물론 이 데이터를 Matplotlib 의 도구를 이용해 시각적으로 표시하는 것이 더 유용할 때도 있다.

```
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');
```


Height Distribution of US Presidents
number
height (cm)
0
2
4
6
8
10
12
160
165
170
175
180
185
190
195