Capítulo 3: Describir datos con estadísticas

## **3** **Describir** datos con estadísticas



En este capítulo utilizaremos Python para explorar la estadística, de modo que podamos estudiar, describir y comprender mejor conjuntos de datos. Después de ver algunas medidas estadísticas básicas -media, mediana, moda y rango-, pasaremos a algunas medidas más avanzadas, como la varianza y la desviación típica. Luego veremos cómo calcular el coeficiente de correlación, que permite cuantificar la relación entre dos conjuntos de datos. Terminaremos el capítulo aprendiendo sobre los diagramas de dispersión. Por el camino, aprenderemos más sobre el lenguaje Python y los módulos de la biblioteca estándar. Empecemos con una de las medidas estadísticas más utilizadas: la media.

**NOTA**

*En estadística, algunas medidas estadísticas se calculan de forma ligeramente diferente dependiendo de si tienes datos de toda una población o sólo de una muestra. Para simplificar las cosas, en este capítulo nos ceñiremos a los métodos de cálculo para una población.*

### **Encontrar la media**

La *media* es una forma común e intuitiva de resumir un conjunto de números. Es lo que podríamos llamar simplemente la "media" en el uso cotidiano, aunque como veremos, también hay otros tipos de medias. Tomemos un conjunto de números de muestra y calculemos la media.

Supongamos que hay una escuela benéfica que ha estado recibiendo donativos durante un periodo de tiempo que abarca los últimos 12 días (lo denominaremos periodo A). En ese tiempo, los 12 números siguientes representan el importe total en dólares de los donativos recibidos cada día: 100, 60, 70, 900, 100, 200, 500, 500, 503, 600, 1000 y 1200. Podemos calcular la media sumando estos totales y dividiendo la suma por el número de días. En este caso, la suma de los números es 5733. Si dividimos este número por 12 (el número de días), obtenemos 477,75, que es la donación *media* por día. Este número nos da una idea general de cuánto dinero se donó en un día determinado.

Dentro de un momento, escribiremos un programa que calcule e imprima la media de una colección de números. Como acabamos de ver, para calcular la media, tendremos que tomar la suma de la lista de números y dividirla por el número de elementos de la lista. Veamos dos funciones de Python que facilitan estas dos operaciones: sum() y len().

Cuando utilizas la función sum() en una lista de números, suma todos los números de la lista y devuelve el resultado:

>>> shortlist = [1, 2, 3]  
>>> sum(shortlist)  
6

Podemos utilizar la función len() para obtener la longitud de una lista:

>>> len(shortlist)  
3

Cuando utilizamos la función len() en la lista, nos devuelve 3 porque hay tres elementos en shortlist. Ahora estamos preparados para escribir un programa que calcule la media de la lista de donaciones.

'''  
Calculating the mean  
'''  
  
def calculate\_mean(numbers):  
➊ s = sum(numbers)  
➋ N = len(numbers)  
# Calculate the mean  
➌ mean = s/N  
  
return mean  
  
if \_\_name\_\_ == '\_\_main\_\_':  
➍ donations = [100, 60, 70, 900, 100, 200, 500, 500, 503, 600, 1000, 1200]  
➎ mean = calculate\_mean(donations)  
N = len(donations)  
➏ print('Mean donation over the last {0} days is {1}'.format(N, mean))

Primero, definimos una función, calculate\_mean(), que acepta el argumento numbers, que es una lista de números. En ➊, utilizamos la función sum() para sumar los números de la lista y creamos una etiqueta, s, para referirnos al total. Del mismo modo, en ➋, utilizamos la función len() para obtener la longitud de la lista y creamos una etiqueta, N, para referirnos a ella. Después, como puedes ver en ➌, calculamos la media simplemente dividiendo la suma (s) por el número de miembros (N). En ➍, creamos una lista, donations, con los valores de las donaciones enumeradas anteriormente. A continuación, llamamos a la función calculate\_mean(), pasando esta lista como argumento en ➎. Por último, imprimimos la media calculada en ➏.

Cuando ejecutes el programa, deberías ver lo siguiente:

Mean donation over the last 12 days is 477.75

La función calculate\_mean() calculará la suma y la longitud de *cualquier* lista, así que también podemos reutilizarla para calcular la media de otros conjuntos de números.

Calculamos que la media de donaciones diarias es de 477,75. Cabe señalar que las donaciones de los primeros días fueron mucho menores que la donación media que calculamos y que las donaciones de los últimos días fueron mucho mayores. La media nos da una forma de resumir los datos, pero no nos da una imagen completa. Sin embargo, hay otras medidas estadísticas que pueden decirnos más sobre los datos cuando se comparan con la media.

### **Encontrar la mediana**

La *mediana* de un conjunto de números es otro tipo de media. Para hallar la mediana, ordenamos los números en orden ascendente. Si la longitud de la lista de números es impar, el número situado en el centro de la lista es la mediana. Si la longitud de la lista de números es par, obtenemos la mediana tomando la media de los dos números centrales. Busquemos la mediana de la lista anterior de donaciones: 100, 60, 70, 900, 100, 200, 500, 500, 503, 600, 1000 y 1200.

Después de ordenar de menor a mayor, la lista de números se convierte en 60, 70, 100, 100, 200, 500, 500, 503, 600, 900, 1000 y 1200. Tenemos un número par de elementos en la lista (12), así que para obtener la mediana, tenemos que tomar la media de los dos números centrales. En este caso, los números centrales son el sexto y el séptimo -500 y 500- y la media de estos dos números es (500 + 500)/2, que da 500. Esto significa que la mediana es 500. Eso significa que la mediana es 500.

Ahora supongamos -sólo para este ejemplo- que tenemos otro total de donaciones para el 13º día, de modo que la lista tiene ahora este aspecto: 100, 60, 70, 900, 100, 200, 500, 500, 503, 600, 1000, 1200 y 800.

Una vez más, tenemos que ordenar la lista, que pasa a ser 60, 70, 100, 100, 200, 500, 500, 503, 600, 800, 900, 1000 y 1200. En esta lista hay 13 números (un número impar), por lo que la mediana de esta lista es simplemente el número del medio. En este caso, es el séptimo número, que es 500.

Antes de escribir un programa para hallar la mediana de una lista de números, pensemos cómo podríamos calcular automáticamente los elementos medios de una lista en cualquier caso. Si la longitud de una lista*(N*) es impar, el número del medio es el que está en la posición*(N* + 1)/2. Si *N* es par, los dos elementos del medio son *N/2* y*(N/2*) + 1. En nuestro primer ejemplo de esta sección, *N* = 12, por lo que los dos elementos del medio eran los elementos 12/2 (sexto) y 12/2 + 1 (séptimo). En el segundo ejemplo, *N* = 13, por lo que el séptimo elemento,*(N* + 1)/2, era el elemento medio.

Para escribir una función que calcule la mediana, también necesitaremos ordenar una lista en orden ascendente. Por suerte, el método sort() hace precisamente eso:

>>> samplelist = [4, 1, 3]  
>>> samplelist.sort()  
>>> samplelist  
[1, 3, 4]

Ahora podemos escribir nuestro siguiente programa, que halla la mediana de una lista de números:

'''  
Calculating the median  
'''  
  
def calculate\_median(numbers):  
➊ N = len(numbers)  
➋ numbers.sort()  
  
# Find the median  
if N % 2 == 0:  
# if N is even  
m1 = N/2  
m2 = (N/2) + 1  
# Convert to integer, match position  
➌ m1 = int(m1) - 1  
➍ m2 = int(m2) - 1  
➎ median = (numbers[m1] + numbers[m2])/2  
else:  
➏ m = (N+1)/2  
# Convert to integer, match position  
m = int(m) - 1  
median = numbers[m]  
  
return median  
  
if \_\_name\_\_ == '\_\_main\_\_':  
donations = [100, 60, 70, 900, 100, 200, 500, 500, 503, 600, 1000, 1200]  
  
median = calculate\_median(donations)  
N = len(donations)  
print('Median donation over the last {0} days is {1}'.format(N, median))

La estructura general del programa es similar a la del programa anterior que calcula la media. La función calculate\_median() acepta una lista de números y devuelve la mediana. En ➊, calculamos la longitud de la lista y creamos una etiqueta, N, para referirnos a ella. A continuación, en ➋, ordenamos la lista utilizando el método sort().

A continuación, comprobamos si N es par. Si es así, encontramos los elementos del medio, m1 y m2, que son los números que ocupan las posiciones N/2 y (N/2) + 1 en la lista ordenada. Las dos sentencias siguientes(➌ y ➍) ajustan m1 y m2 de dos formas. En primer lugar, utilizamos la función int() para convertir m1 y m2 en enteros. Esto se debe a que los resultados del operador de división siempre se devuelven como números en coma flotante, aunque el resultado sea equivalente a un entero. Por ejemplo:

>>> 6/2  
3.0

No podemos utilizar un número de coma flotante como índice en una lista, así que utilizamos int() para convertir ese resultado en un entero. También restamos 1 tanto a m1 como a m2 porque las posiciones en una lista empiezan por 0 en Python. Esto significa que para obtener el sexto y el séptimo número de la lista, tenemos que preguntar por los números del índice 5 y del índice 6. En ➎, calculamos la mediana tomando la media de los dos números de las posiciones intermedias.

A partir de ➏, el programa encuentra la mediana si hay un número impar de elementos en la lista, utilizando de nuevo int() y restando 1 para encontrar el índice adecuado. Por último, el programa calcula la mediana de la lista de donaciones y la devuelve. Cuando ejecutas el programa, éste calcula que la mediana es 500:

Median donation over the last 12 days is 500.0

Como puedes ver, la media (477,75) y la mediana (500) están bastante próximas en esta lista concreta, pero la mediana es un poco más alta.

### **Encontrar la moda y crear una tabla de frecuencias**

En lugar de hallar la media o la mediana de un conjunto de números, ¿qué pasaría si quisieras encontrar el número que aparece con más frecuencia? Este número se llama *moda*. Por ejemplo, considera las puntuaciones de un examen de matemáticas (sobre 10 puntos) en una clase de 20 alumnos: 7, 8, 9, 2, 10, 9, 9, 9, 9, 4, 5, 6, 1, 5, 6, 7, 8, 6, 1 y 10. La moda de esta lista te indicaría qué puntuación era la más común en la clase. En la lista puedes ver que la puntuación de 9 es la más frecuente, por lo que 9 es la moda de esta lista de números. No hay una fórmula simbólica para calcular la moda: simplemente cuentas cuántas veces aparece cada número único y encuentras el que aparece más veces.

Para escribir un programa que calcule la moda, tendremos que hacer que Python cuente cuántas veces aparece cada número dentro de una lista e imprima el que aparece con más frecuencia. La clase Counter del módulo collections, que forma parte de la biblioteca estándar, nos lo pone muy fácil.

#### ***Encontrar los elementos más comunes***

Encontrar el número más común de un conjunto de datos puede considerarse como un subproblema de encontrar un número arbitrario de números más comunes. Por ejemplo, en lugar de la puntuación más común, ¿qué pasaría si quisieras conocer las cinco puntuaciones más comunes? El método most\_common() de la clase Counter nos permite responder fácilmente a estas preguntas. Veamos un ejemplo:

>>> simplelist = [4, 2, 1, 3, 4]  
>>> from collections import Counter  
>>> c = Counter(simplelist)  
>>> c.most\_common()  
[(4, 2), (1, 1), (2, 1), (3, 1)]

Partimos de una lista de cinco números e importamos Counter del módulo de colecciones. A continuación, creamos un objeto Counter, utilizando c para referirnos al objeto. A continuación, llamamos al método most\_common(), que devuelve una lista ordenada por los elementos más comunes.

Cada miembro de la lista es una tupla. El primer elemento de la primera tupla es el número que aparece con más frecuencia, y el segundo elemento es el número de veces que aparece. La segunda, tercera y cuarta tuplas contienen los demás números junto con el recuento del número de veces que aparecen. Este resultado nos dice que el 4 es el que más veces aparece (dos veces), mientras que los demás sólo aparecen una vez. Ten en cuenta que los números que aparecen el mismo número de veces son devueltos por el método most\_common() en un orden arbitrario.

Cuando llames al método most\_common(), también puedes proporcionarle un argumento que le indique el número de elementos más comunes que quieres que devuelva. Por ejemplo, si sólo quisiéramos encontrar el elemento más común, lo llamaríamos con el argumento 1:

>>> c.most\_common(1)  
[(4, 2)]

Si vuelves a llamar al método con 2 como argumento, verás esto

>>> c.most\_common(2)  
[(4, 2), (1, 1)]

Ahora el resultado devuelto por el método most\_common es una lista con dos tuplas. La primera es el elemento más común, seguido del segundo más común. Por supuesto, en este caso, hay varios elementos empatados en el más común, por lo que el hecho de que la función devuelva aquí 1 (y no 2 ó 3) es arbitrario, como ya se ha indicado.

El método most\_common() devuelve tanto los números como el número de veces que aparecen. ¿Y si sólo queremos los números y no nos importa el número de veces que aparecen? He aquí cómo podemos recuperar esa información:

➊ >>> mode = c.most\_common(1)  
>>> mode  
[(4, 2)]  
➋ >>> mode[0]  
(4, 2)  
➌ >>> mode[0][0]  
4

En ➊, utilizamos la etiqueta mode para referirnos al resultado devuelto por el método most\_common(). Recuperamos el primer (y único) elemento de esta lista con mode[0] ➋, lo que nos da una tupla. Como sólo queremos el primer elemento de la tupla, podemos recuperarlo con mode[0][0] ➌. Esto devuelve 4- el elemento más común, o el modo.

Ahora que sabemos cómo funciona el método most\_common(), lo aplicaremos para resolver los dos problemas siguientes.

#### ***Encontrar el modo***

Ya estamos preparados para escribir un programa que encuentre el modo de una lista de números:

'''  
Calculating the mode  
'''  
  
from collections import Counter  
  
def calculate\_mode(numbers):  
➊ c = Counter(numbers)  
➋ mode = c.most\_common(1)  
➌ return mode[0][0]  
  
if \_\_name\_\_=='\_\_main\_\_':  
scores = [7,8,9,2,10,9,9,9,9,4,5,6,1,5,6,7,8,6,1,10]  
mode = calculate\_mode(scores)  
  
print('The mode of the list of numbers is: {0}'.format(mode))

La función calculate\_mode() encuentra y devuelve el modo de los números que se le pasan como parámetro. Para calcular el modo, primero importamos la clase Counter del módulo collections y la utilizamos para crear un objeto Counter en ➊. Después, en ➋, utilizamos el método most\_common(), que, como vimos antes, nos da una lista que contiene una tupla con el número más común y el número de veces que aparece. Asignamos a esa lista la etiqueta mode. Por último, utilizamos mode[0][0] ➌ para acceder al número que queremos: el número más frecuente de la lista, que es el modo.

El resto del programa aplica la función calculate\_mode a la lista de puntuaciones de exámenes que vimos antes. Cuando ejecutes el programa, deberías ver la siguiente salida:

The mode of the list of numbers is: 9

¿Qué ocurre si tienes un conjunto de datos en el que dos o más números aparecen el mismo número máximo de veces? Por ejemplo, en la lista de números 5, 5, 5, 4, 4, 4, 9, 1 y 3, tanto el 4 como el 5 están presentes tres veces. En estos casos, se dice que la lista de números tiene modos múltiples, y nuestro programa debe encontrar e imprimir todos los modos. El programa modificado es el siguiente:

'''  
Calculating the mode when the list of numbers may  
have multiple modes  
'''  
  
  
from collections import Counter  
  
  
def calculate\_mode(numbers):  
  
c = Counter(numbers)  
➊ numbers\_freq = c.most\_common()  
➋ max\_count = numbers\_freq[0][1]  
  
modes = []  
for num in numbers\_freq:  
➌ if num[1] == max\_count:  
modes.append(num[0])  
return modes  
  
if \_\_name\_\_ == '\_\_main\_\_':  
scores = [5, 5, 5, 4, 4, 4, 9, 1, 3]  
modes = calculate\_mode(scores)  
print('The mode(s) of the list of numbers are:')  
➍ for mode in modes:  
print(mode)

En ➊, en lugar de encontrar sólo el elemento más común, recuperamos todos los números y el número de veces que aparece cada uno. A continuación, en ➋, hallamos el valor de la cuenta máxima, es decir, el número máximo de veces que aparece cualquier número. Luego, para cada uno de los números, comprobamos si el número de veces que aparece es igual a la cuenta máxima ➌. Cada número que cumple esta condición es un modo, y lo añadimos a la lista modes y devolvemos la lista.

En ➍, iteramos sobre la lista devuelta por la función calculate\_mode() e imprimimos cada uno de los números.

Cuando ejecutes el programa anterior, deberías ver la siguiente salida:

The mode(s) of the list of numbers are:  
4  
5

¿Y si quisieras encontrar el número de veces que aparece cada número en lugar de sólo la moda? Una *tabla* de frecuencias, como su nombre indica, es una tabla que muestra cuántas veces aparece cada número dentro de una colección de números.

#### ***Crear una tabla de frecuencias***

Consideremos de nuevo la lista de puntuaciones del examen: 7, 8, 9, 2, 10, 9, 9, 9, 9, 4, 5, 6, 1, 5, 6, 7, 8, 6, 1 y 10. La tabla de frecuencias de esta lista se muestra en [la Tabla 3-1](ch03.html#ch3tab1). Para cada número, indicamos el número de veces que aparece en la segunda columna.

**Tabla 3-1:** Tabla de frecuencias

| **Puntuación** | **Frecuencia** |
| --- | --- |
| 1 | 2 |
| 2 | 1 |
| 4 | 1 |
| 5 | 2 |
| 6 | 3 |
| 7 | 2 |
| 8 | 2 |
| 9 | 5 |
| 10 | 2 |

Observa que la suma de las frecuencias individuales de la segunda columna suma el número total de puntuaciones (en este caso, 20).

Utilizaremos una vez más el método most\_common() para imprimir la tabla de frecuencias de un conjunto dado de números. Recuerda que cuando no proporcionamos ningún argumento al método most\_common(), éste devuelve una lista de tuplas con todos los números y el número de veces que aparecen. Podemos imprimir simplemente cada número y su frecuencia de esta lista para mostrar una tabla de frecuencias.

Éste es el programa:

'''  
Frequency table for a list of numbers  
'''  
  
from collections import Counter  
  
def frequency\_table(numbers):  
➊ table = Counter(numbers)  
print('Number\tFrequency')  
➋ for number in table.most\_common():  
print('{0}\t{1}'.format(number[0], number[1]))  
  
if \_\_name\_\_=='\_\_main\_\_':  
scores = [7,8,9,2,10,9,9,9,9,4,5,6,1,5,6,7,8,6,1,10]  
frequency\_table(scores)

La función frequency\_table() imprime la tabla de frecuencias de la lista de números que se le pasa. En ➊, primero creamos un objeto Counter y creamos la etiqueta table para referirnos a él. A continuación, utilizando un bucle for ➋, recorremos cada una de las tuplas, imprimiendo el primer miembro (el número en sí) y el segundo miembro (la frecuencia del número correspondiente). Utilizamos \t para imprimir una tabulación entre cada valor para espaciar la tabla. Cuando ejecutes el programa, verás la siguiente salida:

Number Frequency  
9 5  
6 3  
1 2  
5 2  
7 2  
8 2  
10 2  
2 1  
4 1

Aquí puedes ver que los números aparecen en orden decreciente de frecuencia porque la función most\_common() devuelve los números en este orden. Si, por el contrario, quieres que tu programa imprima la tabla de frecuencias ordenada por valor de menor a mayor, como se muestra en [la Tabla 3-1](ch03.html#ch3tab1), tendrás que volver a ordenar la lista de tuplas.

El método sort() es todo lo que necesitamos para modificar nuestro anterior programa de tabla de frecuencias:

'''  
Frequency table for a list of numbers  
Enhanced to display the table sorted by the numbers  
'''  
  
from collections import Counter  
  
def frequency\_table(numbers):  
table = Counter(numbers)  
➊ numbers\_freq = table.most\_common()  
➋ numbers\_freq.sort()  
  
print('Number\tFrequency')  
➌ for number in numbers\_freq:  
print('{0}\t{1}'.format(number[0], number[1]))  
  
if \_\_name\_\_ == '\_\_main\_\_':  
scores = [7,8,9,2,10,9,9,9,9,4,5,6,1,5,6,7,8,6,1,10]  
frequency\_table(scores)

Aquí, almacenamos la lista devuelta por el método most\_common() en numbers\_freq en ➊, y luego la ordenamos llamando al método sort() ➋. Por último, utilizamos el bucle for para repasar las tuplas ordenadas e imprimir cada número y su frecuencia ➌. Ahora, cuando ejecutes el programa, verás la siguiente tabla, que es idéntica a [la Tabla 3-1](ch03.html#ch3tab1):

Number Frequency  
1 2  
2 1  
4 1  
5 2  
6 3  
7 2  
8 2  
9 5  
10 2

En esta sección hemos tratado la media, la mediana y la moda, que son tres medidas habituales para describir una lista de números. Cada una de ellas puede ser útil, pero también pueden ocultar otros aspectos de los datos cuando se consideran aisladamente. A continuación, veremos otras medidas estadísticas más avanzadas que pueden ayudarnos a sacar más conclusiones sobre una colección de números.

### **Medir la dispersión**

Los siguientes cálculos estadísticos que veremos miden la *dispersión*, que nos indica la distancia que separa los números de un conjunto de datos de la media del conjunto. Aprenderemos a calcular tres medidas distintas de la dispersión: el rango, la varianza y la desviación típica.

#### ***Hallar el rango de un conjunto de números***

Una vez más, considera la lista de donaciones durante el periodo A: 100, 60, 70, 900, 100, 200, 500, 500, 503, 600, 1000 y 1200. Comprobamos que la media de donaciones por día es de 477,75. Pero sólo mirando la media, no tenemos ni idea de si todas las donaciones cayeron en un rango estrecho -digamos entre 400 y 500- o si variaron mucho más que eso -digamos entre 60 y 1200, como en este caso-. Para una lista de números, el *rango* es la diferencia entre el número más alto y el más bajo. Podrías tener dos grupos de números con exactamente la misma media pero con rangos muy diferentes, por lo que conocer el rango aporta más información sobre un conjunto de números que la que podemos obtener con sólo mirar la media, la mediana y la moda.

El siguiente programa halla el rango de la lista de donaciones anterior:

'''  
Find the range  
'''  
  
def find\_range(numbers):  
  
➊ lowest = min(numbers)  
➋ highest = max(numbers)  
# Find the range  
r = highest-lowest  
  
➌ return lowest, highest, r  
  
if \_\_name\_\_ == '\_\_main\_\_':  
donations = [100, 60, 70, 900, 100, 200, 500, 500, 503, 600, 1000, 1200]  
➍ lowest, highest, r = find\_range(donations)  
print('Lowest: {0} Highest: {1} Range: {2}'.format(lowest, highest, r))

La función find\_range() acepta una lista como parámetro y halla el rango. Primero, calcula los números más bajos y más altos utilizando las funciones min() y max() en ➊ y ➋. Como indican los nombres de las funciones, encuentran los valores mínimo y máximo en una lista de números.

A continuación, calculamos el rango tomando la diferencia entre los números mayor y menor, utilizando la etiqueta r para referirnos a esta diferencia. En ➌, devolvemos los tres números: el número más bajo, el número más alto y el rango. Es la primera vez en el libro que devolvemos varios valores de una función: en lugar de devolver sólo un valor, esta función devuelve tres. En ➍, utilizamos tres etiquetas para *recibir* los tres valores que devuelve la función find\_range(). Por último, imprimimos los valores. Cuando ejecutes el programa, deberías ver la siguiente salida:

Lowest: 60 Highest: 1200 Range: 1140

Esto nos indica que las donaciones totales de los días estaban bastante repartidas, con un rango de 1140, porque teníamos totales diarios tan pequeños como 60 y tan grandes como 1200.

#### ***Hallar la varianza y la desviación típica***

El rango nos indica la diferencia entre los dos extremos de un conjunto de cifras, pero ¿qué ocurre si queremos saber más sobre cómo varían todas las cifras individuales respecto a la media? ¿Eran todas similares, agrupadas cerca de la media, o eran todas diferentes, más cerca de los extremos? Hay dos medidas de dispersión relacionadas que nos dicen más sobre una lista de números en este sentido: la *varianza* y la *desviación típica*. Para calcular cualquiera de ellas, primero tenemos que hallar la diferencia de cada uno de los números respecto a la media. La varianza es la media de los cuadrados de esas diferencias. Una varianza alta significa que los valores están lejos de la media; una varianza baja significa que los valores están agrupados cerca de la media. Calculamos la varianza mediante la fórmula

image

En la fórmula, *xi* representa los números individuales (en este caso, los donativos totales diarios), xmedia representa la media de estos números (el donativo medio diario), y *n* es el número de valores de la lista (el número de días en los que se recibieron donativos). Para cada valor de la lista, tomamos la diferencia entre ese número y la media y la elevamos al cuadrado. Luego, sumamos todas esas diferencias al cuadrado y, por último, dividimos la suma total por *n* para hallar la varianza.

Si queremos calcular también la desviación típica, basta con sacar la raíz cuadrada de la varianza. Los valores que están dentro de una desviación típica de la media se pueden considerar bastante típicos, mientras que los valores que están a tres o más desviaciones típicas de la media se pueden considerar mucho más atípicos; a esos valores los llamamos *valores atípicos*.

¿Por qué tenemos estas dos medidas de dispersión: la varianza y la desviación típica? En resumen, las dos medidas son útiles en situaciones diferentes. Volviendo a la fórmula que utilizamos para calcular la varianza, puedes ver que la varianza se expresa en unidades cuadradas porque es la media de la diferencia al cuadrado respecto a la media. Para algunas fórmulas matemáticas, es más agradable trabajar con esas unidades cuadradas en lugar de sacar la raíz cuadrada para hallar la desviación típica. Por otra parte, la desviación típica se expresa en las mismas unidades que los datos de la población. Por ejemplo, si calculas la varianza de nuestra lista de donaciones (como haremos dentro de un momento), el resultado se expresa en dólares al cuadrado, lo que no tiene mucho sentido. Mientras tanto, la desviación típica se expresa simplemente en dólares, la misma unidad que cada una de las donaciones.

El siguiente programa halla la varianza y la desviación típica de una lista de números:

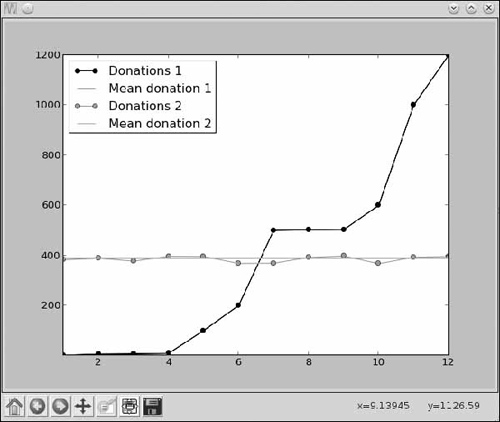
'''  
Find the variance and standard deviation of a list of numbers  
'''  
  
def calculate\_mean(numbers):  
s = sum(numbers)  
N = len(numbers)  
# Calculate the mean  
mean = s/N  
  
return mean  
  
def find\_differences(numbers):  
# Find the mean  
mean = calculate\_mean(numbers)  
# Find the differences from the mean  
diff = []  
for num in numbers:  
diff.append(num-mean)  
  
return diff  
  
def calculate\_variance(numbers):  
  
# Find the list of differences  
➊ diff = find\_differences(numbers)  
# Find the squared differences  
squared\_diff = []  
➋ for d in diff:  
squared\_diff.append(d\*\*2)  
# Find the variance  
sum\_squared\_diff = sum(squared\_diff)  
➌ variance = sum\_squared\_diff/len(numbers)  
return variance  
  
if \_\_name\_\_ == '\_\_main\_\_':  
donations = [100, 60, 70, 900, 100, 200, 500, 500, 503, 600, 1000, 1200]  
variance = calculate\_variance(donations)  
print('The variance of the list of numbers is {0}'.format(variance))  
  
➍ std = variance\*\*0.5  
print('The standard deviation of the list of numbers is {0}'.format(std))

La función calculate\_variance() calcula la varianza de la lista de números que se le pasa. Primero, llama a la función find\_differences() en ➊ para calcular la diferencia de cada uno de los números respecto a la media. La función find\_differences() devuelve la diferencia de cada donación respecto al valor medio en forma de lista. En esta función, utilizamos la función calculate\_mean() que escribimos antes para hallar la donación media. A continuación, a partir de ➋, se calculan los cuadrados de estas diferencias y se guardan en una lista denominada squared\_diff. A continuación, utilizamos la función sum() para hallar la suma de las diferencias al cuadrado y, por último, calculamos la varianza en ➌. En ➍, calculamos la desviación típica tomando la raíz cuadrada de la varianza.

Cuando ejecutes el programa anterior, deberías ver el siguiente resultado:

The variance of the list of numbers is 141047.35416666666  
The standard deviation of the list of numbers is 375.5627166887931

Tanto la varianza como la desviación típica son muy grandes, lo que significa que las donaciones totales diarias individuales varían mucho respecto a la media. Ahora, comparemos la varianza y la desviación típica de un conjunto diferente de donaciones que tienen la misma media: 382, 389, 377, 397, 396, 368, 369, 392, 398, 367, 393 y 396. En este caso, la varianza y la desviación típica resultan ser 135,38888888888889 y 11,63567311713804, respectivamente. Los valores más bajos de la varianza y la desviación típica nos indican que los números individuales están más cerca de la media. [La figura 3-1](ch03.html#ch3fig1) ilustra visualmente este punto.



*Figura 3-1: Variación de los donativos en torno al donativo medio*

Las donaciones medias de ambas listas de donaciones son similares, por lo que las dos líneas se superponen, apareciendo como una sola línea en la figura. Sin embargo, los donativos de la primera lista varían mucho respecto a la media, mientras que los donativos de la segunda lista están muy cerca de la media, lo que confirma lo que dedujimos del valor más bajo de la varianza.

### **Cálculo de la correlación entre dos conjuntos de datos**

En este apartado aprenderemos a calcular una medida estadística que nos indica la naturaleza y la fuerza de la relación entre dos conjuntos de números: el coeficiente de *correlación de Pearson*, que llamaré simplemente *coeficiente de correlación*. Observa que este coeficiente mide la fuerza de la relación *lineal*. Tendríamos que utilizar otras medidas (que no trataremos aquí) para averiguar el coeficiente cuando dos conjuntos tienen una relación no lineal. El coeficiente puede ser positivo o negativo, y su magnitud puede oscilar entre -1 y 1 (ambos inclusive).

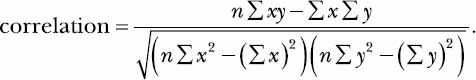
Un coeficiente de correlación de 0 indica que no hay correlación lineal entre las dos cantidades. (Ten en cuenta que esto no significa que las dos cantidades sean independientes entre sí. Podría seguir existiendo una relación no lineal entre ellas, por ejemplo). Un coeficiente de 1 o cercano a 1 indica que existe una fuerte correlación lineal positiva; un coeficiente de exactamente 1 se denomina correlación positiva perfecta. Del mismo modo, un coeficiente de correlación de -1 o cercano a -1 indica una fuerte correlación negativa, donde 1 indica una correlación negativa perfecta.

**CORRELACIÓN Y CAUSALIDAD**

En estadística, a menudo te encontrarás con la afirmación "correlación no implica causalidad". Esto es un recordatorio de que aunque dos conjuntos de observaciones estén fuertemente correlacionados entre sí, eso no significa que una variable *cause* la otra. Cuando dos variables están fuertemente correlacionadas, a veces hay un tercer factor que influye en ambas variables y explica la correlación. Un ejemplo clásico es la correlación entre las ventas de helados y los índices de delincuencia: si haces un seguimiento de estas dos variables en una ciudad típica, es probable que encuentres una correlación, pero esto no significa que las ventas de helados causen la delincuencia (o viceversa). La venta de helados y la delincuencia están correlacionadas porque ambas aumentan cuando hace más calor en verano. Por supuesto, esto tampoco significa que el calor provoque directamente el aumento de la delincuencia; también hay causas más complicadas detrás de esa correlación.

#### ***Cálculo del coeficiente de correlación***

El coeficiente de correlación se calcula mediante la fórmula



En la fórmula anterior, *n* es el número total de valores presentes en cada conjunto de números (los conjuntos tienen que tener la misma longitud). Los dos conjuntos de números se denotan por *x* e *y* (no importa cuál denotes como cuál). Los demás términos se describen como sigue

|  |  |
| --- | --- |
| *Σxy* | Suma de los productos de los elementos individuales de los dos conjuntos de números, *x* e *y* |
| *Σx* | Suma de los números del conjunto *x* |
| *Σy* | Suma de los números del conjunto *y* |
| (*Σx*)2 | Cuadrado de la suma de los números del conjunto *x* |
| (*Σy*)2 | Cuadrado de la suma de los números del conjunto *y* |
| Σx2 | Suma de los cuadrados de los números del conjunto *x* |
| Σy2 | Suma de los cuadrados de los números del conjunto *y* |

Una vez calculados estos términos, puedes combinarlos según la fórmula anterior para hallar el coeficiente de correlación. Para listas pequeñas, es posible hacerlo a mano sin demasiado esfuerzo, pero ciertamente se complica a medida que aumenta el tamaño de cada conjunto de números.

Dentro de un momento, escribiremos un programa que calcule por nosotros el coeficiente de correlación. En este programa, utilizaremos la función zip(), que nos ayudará a calcular la suma de los productos de los dos conjuntos de números. Aquí tienes un ejemplo de cómo funciona la función zip():

>>> simple\_list1 = [1, 2, 3]  
>>> simple\_list2 = [4, 5, 6]  
>>> for x, y in zip(simple\_list1, simple\_list2):  
print(x, y)  
  
1 4  
2 5  
3 6

La función zip() devuelve pares de los elementos correspondientes en x y y, que luego puedes utilizar en un bucle para realizar otras operaciones (como imprimir, como se muestra en el código anterior). Si las dos listas son desiguales en longitud, la función termina cuando se han leído todos los elementos de la lista más pequeña.

Ahora estamos listos para escribir un programa que calcule por nosotros el coeficiente de correlación:

def find\_corr\_x\_y(x,y):  
n = len(x)  
  
# Find the sum of the products  
prod = []  
➊ for xi,yi in zip(x,y):  
prod.append(xi\*yi)  
  
➋ sum\_prod\_x\_y = sum(prod)  
➌ sum\_x = sum(x)  
➍ sum\_y = sum(y)  
squared\_sum\_x = sum\_x\*\*2  
squared\_sum\_y = sum\_y\*\*2  
  
x\_square = []  
➎ for xi in x:  
x\_square.append(xi\*\*2)  
# Find the sum  
x\_square\_sum = sum(x\_square)  
  
y\_square=[]  
for yi in y:  
y\_square.append(yi\*\*2)  
# Find the sum  
y\_square\_sum = sum(y\_square)  
  
# Use formula to calculate correlation  
➏ numerator = n\*sum\_prod\_x\_y - sum\_x\*sum\_y  
denominator\_term1 = n\*x\_square\_sum - squared\_sum\_x  
denominator\_term2 = n\*y\_square\_sum - squared\_sum\_y  
➐ denominator = (denominator\_term1\*denominator\_term2)\*\*0.5  
➑ correlation = numerator/denominator  
  
return correlation

La función find\_corr\_x\_y() acepta dos argumentos, x y y, que son los dos conjuntos de números para los que queremos calcular la correlación. Al principio de esta función, hallamos la longitud de las listas y creamos una etiqueta, n, para referirnos a ella. A continuación, en ➊, tenemos un bucle for que utiliza la función zip() para calcular el producto de los valores correspondientes de cada lista (multiplicando juntos el primer elemento de cada lista, luego el segundo elemento de cada lista, y así sucesivamente). Utilizamos el método append() para añadir estos productos a la lista etiquetada prod.

En ➋, calculamos la suma de los productos almacenados en prod utilizando la función sum(). En las sentencias ➌ y ➍, calculamos la suma de los números de x y y, respectivamente (una vez más, utilizando la función sum() ). A continuación, calculamos los cuadrados de la suma de los elementos en x y y, creando las etiquetas squared\_sum\_x y squared\_sum\_y para referirnos a ellos, respectivamente.

En el bucle que comienza en ➎, calculamos el cuadrado de cada uno de los elementos de x y hallamos la suma de estos cuadrados. A continuación, hacemos lo mismo con los elementos de y. Ahora tenemos todos los términos que necesitamos para calcular la correlación, y lo hacemos en las sentencias ➏,➐ y ➑. Por último, devolvemos la correlación. La correlación es una medida muy citada en los estudios estadísticos, tanto en los medios de comunicación populares como en los artículos científicos. A veces sabemos de antemano que existe una correlación, y sólo queremos averiguar la fuerza de esa correlación. Veremos un ejemplo de esto en "[Lectura de datos de un archivo CSV](ch03.html#ch03lev2sec09)" en [la página 86](ch03.html#page_86), cuando calculemos la correlación entre los datos leídos de un archivo. Otras veces, sólo sospechamos que puede haber una correlación, y debemos investigar los datos para verificar si realmente la hay (como en el ejemplo siguiente).

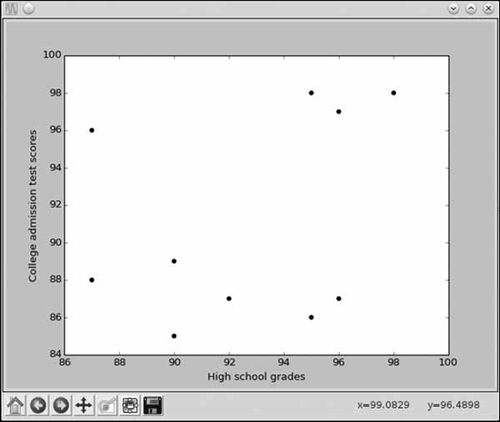
#### ***Notas de bachillerato y rendimiento en las pruebas de acceso a la universidad***

En este apartado, consideraremos un grupo ficticio de 10 estudiantes de bachillerato e investigaremos si existe una relación entre sus calificaciones en la escuela y su rendimiento en las pruebas de acceso a la universidad. [La Tabla 3-2](ch03.html#ch3tab2) enumera los datos que vamos a asumir para nuestro estudio y en los que vamos a basar nuestros experimentos. La columna "Notas de bachillerato" muestra las puntuaciones percentiles de las notas de los alumnos en el bachillerato, y la columna "Puntuaciones en el examen de acceso a la universidad" muestra sus puntuaciones percentiles en el examen de acceso a la universidad.

**Tabla 3-2:** Calificaciones de bachillerato y resultados en la prueba de acceso a la universidad

| **Notas de bachillerato** | **Puntuaciones en la prueba de acceso a la universidad** |
| --- | --- |
| 90 | 85 |
| 92 | 87 |
| 95 | 86 |
| 96 | 97 |
| 87 | 96 |
| 87 | 88 |
| 90 | 89 |
| 95 | 98 |
| 98 | 98 |
| 96 | 87 |

Para analizar estos datos, veamos un *diagrama de* dispersión. [La Figura 3-2](ch03.html#ch3fig2) muestra el diagrama de dispersión del conjunto de datos anterior, con el *eje x* representando las calificaciones de bachillerato y el *eje y* representando los resultados correspondientes de las pruebas de acceso a la universidad.



*Figura 3-2: Diagrama de dispersión de las notas de bachillerato y los resultados de las pruebas de acceso a la universidad*

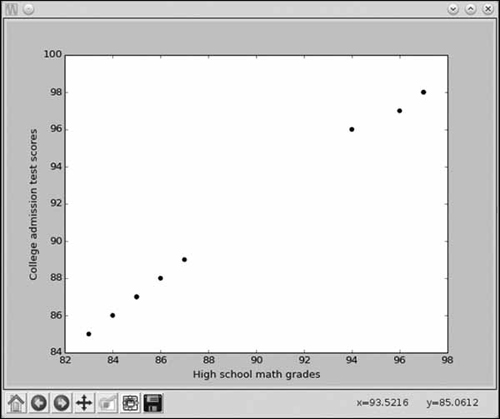
El diagrama de los datos indica que los alumnos con las notas más altas en el instituto no obtuvieron necesariamente mejores resultados en las pruebas de acceso a la universidad y viceversa. Algunos estudiantes con malas notas en el instituto obtuvieron muy buenos resultados en el examen de acceso a la universidad, mientras que otros tenían notas excelentes pero obtuvieron resultados relativamente malos en el examen de acceso a la universidad. Si calculamos el coeficiente de correlación de los dos conjuntos de datos (utilizando nuestro programa de antes), vemos que es aproximadamente 0,32. Esto significa que existe cierta correlación, pero no muy fuerte. Si la correlación fuera más cercana a 1, también lo veríamos reflejado en el diagrama de dispersión: los puntos se ajustarían más a una línea recta diagonal.

Supongamos que las notas de bachillerato que aparecen en [la Tabla 3-2](ch03.html#ch3tab2) son una media de las notas individuales en matemáticas, ciencias, inglés y ciencias sociales. Imaginemos también que el examen de la universidad hace mucho hincapié en las matemáticas, mucho más que en otras asignaturas. En lugar de fijarnos en las calificaciones generales de los alumnos en el instituto, veamos sólo sus calificaciones en matemáticas para ver si eso predice mejor cómo les fue en el examen de la universidad. [La Tabla 3-3](ch03.html#ch3tab3) muestra ahora sólo las calificaciones en matemáticas (como percentiles) y las pruebas de acceso a la universidad. El gráfico de dispersión correspondiente se muestra en [la Figura 3-3](ch03.html#ch3fig3).

**Tabla 3-3:** Notas de Matemáticas en el instituto y resultados en los exámenes de acceso a la universidad

| **Notas de matemáticas en el instituto** | **Puntuaciones en las pruebas de acceso a la universidad** |
| --- | --- |
| 83 | 85 |
| 85 | 87 |
| 84 | 86 |
| 96 | 97 |
| 94 | 96 |
| 86 | 88 |
| 87 | 89 |
| 97 | 98 |
| 97 | 98 |
| 85 | 87 |

Ahora, el diagrama de dispersión[(Figura 3-3](ch03.html#ch3fig3)) muestra que los puntos de datos se extienden casi perfectamente a lo largo de una línea recta. Esto indica que existe una alta correlación entre las puntuaciones obtenidas en matemáticas en el instituto y el rendimiento en el examen de acceso a la universidad. El coeficiente de correlación, en este caso, resulta ser aproximadamente 1. Con la ayuda del diagrama de dispersión y el coeficiente de correlación, podemos concluir que, efectivamente, existe una fuerte relación en este conjunto de datos entre las calificaciones en matemáticas del bachillerato y el rendimiento en las pruebas de acceso a la universidad.



*Figura 3-3: Diagrama de dispersión de las calificaciones en matemáticas de bachillerato y los resultados en las pruebas de acceso a la universidad*

### **Gráficos de dispersión**

En el apartado anterior, vimos un ejemplo de cómo un diagrama de dispersión puede darnos una primera indicación de la existencia de una correlación entre dos conjuntos de números. En este apartado, veremos la importancia de analizar gráficos de dispersión observando un conjunto de cuatro conjuntos de datos. Para estos conjuntos de datos, las medidas estadísticas convencionales resultan ser todas iguales, pero los gráficos de dispersión de cada conjunto de datos revelan importantes diferencias.

En primer lugar, vamos a ver cómo crear un gráfico de dispersión en Python:

>>> x = [1, 2, 3, 4]  
>>> y = [2, 4, 6, 8]  
>>> import matplotlib.pyplot as plt  
➊ >>> plt.scatter(x, y)  
<matplotlib.collections.PathCollection object at 0x7f351825d550>  
>>> plt.show()

La función scatter() se utiliza para crear un gráfico de dispersión entre dos listas de números, x y y ➊. La única diferencia entre este gráfico y los gráficos que creamos en [el Capítulo 2](ch02.html#ch02) es que aquí utilizamos la función scatter() en lugar de la función plot(). Una vez más, tenemos que llamar a show() para visualizar el gráfico.

Para aprender más sobre los gráficos de dispersión, veamos un importante estudio estadístico: "Los gráficos en el análisis estadístico", del estadístico Francis Anscombe[.](footnote.html#fn01) 1 El estudio considera cuatro conjuntos de datos diferentes -denominados *el cuarteto de Anscombe- con*idénticas propiedades estadísticas: media, varianza y coeficiente de correlación.

Los conjuntos de datos son los que se muestran en [la Tabla 3-4](ch03.html#ch3tab4) (reproducida del estudio original).

**Tabla 3-4:** Cuarteto de Anscombe: cuatro conjuntos de datos diferentes con medidas estadísticas casi idénticas

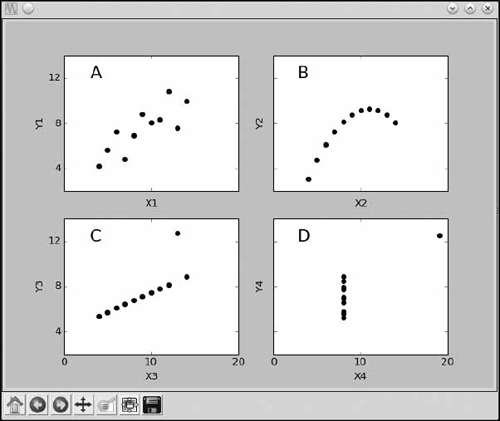
| **A** | | **B** | | **C** | | **D** | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **X1** | **Y1** | **X2** | **Y2** | **X3** | **Y3** | **X4** | **Y4** |
| 10.0 | 8.04 | 10.0 | 9.14 | 10.0 | 7.46 | 8.0 | 6.58 |
| 8.0 | 6.95 | 8.0 | 8.14 | 8.0 | 6.77 | 8.0 | 5.76 |
| 13.0 | 7.58 | 13.0 | 8.74 | 13.0 | 12.74 | 8.0 | 7.71 |
| 9.0 | 8.81 | 9.0 | 8.77 | 9.0 | 7.11 | 8.0 | 8.84 |
| 11.0 | 8.33 | 11.0 | 9.26 | 11.0 | 7.81 | 8.0 | 8.47 |
| 14.0 | 9.96 | 14.0 | 8.10 | 14.0 | 8.84 | 8.0 | 7.04 |
| 6.0 | 7.24 | 6.0 | 6.13 | 6.0 | 6.08 | 8.0 | 5.25 |
| 4.0 | 4.26 | 4.0 | 3.10 | 4.0 | 5.39 | 19.0 | 12.50 |
| 12.0 | 10.84 | 12.0 | 9.13 | 12.0 | 8.15 | 8.0 | 5.56 |
| 7.0 | 4.82 | 7.0 | 7.26 | 7.0 | 6.42 | 8.0 | 7.91 |
| 5.0 | 5.68 | 5.0 | 4.74 | 5.0 | 5.73 | 8.0 | 6.89 |

Nos referiremos a los pares (X1, Y1), (X2, Y2), (X3, Y3) y (X4, Y4) como conjuntos de datos A, B, C y D, respectivamente. [La Tabla 3-5](ch03.html#ch3tab5) presenta las medidas estadísticas de los conjuntos de datos redondeadas a dos cifras decimales.

**Tabla 3-5:** Cuarteto de Anscombe-Medidas estadísticas

| **Conjunto de datos** | **X** | | **Y** | |  |
| --- | --- | --- | --- | --- | --- |
| **Media** | **Desv. est.** | **Media** | **Desv. est.** | **Correlación** |
| A | 9.00 | 3.32 | 7.50 | 2.03 | 0.82 |
| B | 9.00 | 3.32 | 7.50 | 2.03 | 0.82 |
| C | 9.00 | 3.32 | 7.50 | 2.03 | 0.82 |
| D | 9.00 | 3.32 | 7.50 | 2.03 | 0.82 |

Los diagramas de dispersión de cada conjunto de datos se muestran en [la Figura](ch03.html#ch3fig4) 3-4.



*Figura 3-4: Diagramas de dispersión del cuarteto de Anscombe*

Si nos fijamos sólo en las medidas estadísticas tradicionales (ver [Tabla 3-5](ch03.html#ch3tab5)) -como la media, la desviación típica y el coeficiente de correlación-, estos conjuntos de datos parecen casi idénticos. Pero los gráficos de dispersión muestran que estos conjuntos de datos son en realidad muy diferentes entre sí. Así pues, los gráficos de dispersión pueden ser una herramienta importante y deben utilizarse junto con otras medidas estadísticas antes de sacar conclusiones sobre un conjunto de datos.

### **Lectura de datos de archivos**

En todos los programas de este capítulo, las listas de números que utilizamos en nuestros cálculos estaban explícitamente escritas, o *codificadas*, en los propios programas. Si quisieras encontrar las medidas para un conjunto de datos diferente, tendrías que introducir todo el nuevo conjunto de datos en el propio programa. También sabes cómo hacer programas que permitan al usuario introducir los datos como entrada, pero con grandes conjuntos de datos, no es muy conveniente hacer que el usuario introduzca largas listas de números cada vez que utiliza el programa.

Una alternativa mejor es leer los datos del usuario desde un archivo. Veamos un ejemplo sencillo de cómo podemos leer números de un fichero y realizar operaciones matemáticas con ellos. En primer lugar, te mostraré cómo leer datos de un simple fichero de texto en el que cada línea del fichero contiene un nuevo elemento de datos. Después, te mostraré cómo leer de un archivo en el que los datos están almacenados en el conocido formato CSV de , lo que te abrirá un montón de posibilidades, ya que hay montones de conjuntos de datos útiles que puedes descargar de Internet en formato CSV. (Si no estás familiarizado con el manejo de archivos en Python, consulta [el Apéndice B](app02.html#app02) para una breve introducción).

#### ***Lectura de datos de un archivo de texto***

Tomemos un archivo, *misdatos.txt*, con la lista de donaciones (una por línea) durante el periodo A que hemos considerado al principio de este capítulo:

100  
60  
70  
900  
100  
200  
500  
500  
503  
600  
1000  
1200

El siguiente programa leerá este fichero e imprimirá la suma de los números almacenados en el fichero:

# Find the sum of numbers stored in a file  
def sum\_data(filename):  
s = 0  
➊ with open(filename) as f:  
for line in f:  
➋ s = s + float(line)  
print('Sum of the numbers: {0}'.format(s))  
  
if \_\_name\_\_ == '\_\_main\_\_':  
sum\_data('mydata.txt')

La función sum\_data() abre el archivo especificado por el argumento filename en ➊ y lo lee línea a línea (f se denomina *objeto archivo*, y puedes pensar que apunta a un archivo abierto). En ➋, convertimos cada número en un número de coma flotante utilizando la función float() y seguimos sumando hasta que hayamos leído todos los números. El número final, etiquetado s, contiene la suma de los números, que se imprime al final de la función.

Antes de ejecutar el programa, debes crear un archivo llamado *misdatos.txt* con los datos apropiados y guardarlo en el mismo directorio que tu programa. Puedes crear este archivo desde el propio IDLE pulsando Archivo▸Nueva**ventana**, escribiendo los números (uno por línea) en la nueva ventana y guardando el archivo como *mydata*.txt en el mismo directorio que tu programa. Ahora, si ejecutas el programa, verás la siguiente salida:

Sum of the numbers: 5733.0

Todos nuestros programas de este capítulo han supuesto que los datos de entrada están disponibles en listas. Para utilizar nuestros programas anteriores con los datos de un fichero, primero tenemos que crear una lista a partir de esos datos. Una vez que tenemos una lista, podemos utilizar las funciones que hemos escrito antes para calcular la estadística correspondiente. El siguiente programa calcula la media de los números almacenados en el fichero *misdatos.txt*:

'''  
Calculating the mean of numbers stored in a file  
'''  
def read\_data(filename):  
  
numbers = []  
with open(filename) as f:  
for line in f:  
➊ numbers.append(float(line))  
  
return numbers  
  
def calculate\_mean(numbers):  
s = sum(numbers)  
N = len(numbers)  
mean = s/N  
  
return mean  
  
if \_\_name\_\_ == '\_\_main\_\_':  
➋ data = read\_data('mydata.txt')  
mean = calculate\_mean(data)  
print('Mean: {0}'.format(mean))

Antes de poder llamar a la función calculate\_mean(), tenemos que leer los números almacenados en el fichero y convertirlos en una lista. Para ello, utiliza la función read\_data(), que lee el fichero línea por línea. En lugar de sumar los números, esta función los convierte en números de coma flotante y los añade a la lista numbers ➊. Se devuelve la lista, y nos referimos a ella con la etiqueta data ➋. A continuación, invocamos la función calculate\_mean(), que devuelve la media de los datos. Por último, la imprimimos.

Cuando ejecutes el programa, deberías ver la siguiente salida:

Mean: 477.75

Por supuesto, verás un valor diferente para la media si los números de tu fichero son distintos de los de este ejemplo.

Consulta [el Apéndice B](app02.html#app02) para obtener consejos sobre cómo puedes pedir al usuario que introduzca el nombre del archivo y luego modificar tu programa en consecuencia. Esto permitirá al usuario de tu programa especificar cualquier archivo de datos.

#### ***Lectura de datos de un archivo CSV***

Un archivo de valores separados por comas (CSV) consta de filas y columnas con las columnas separadas entre sí por comas. Puedes visualizar un archivo CSV utilizando un editor de texto de tu sistema operativo o un software especializado, como Microsoft Excel, OpenOffice Calc o LibreOffice Calc.

Aquí tienes un ejemplo de archivo CSV que contiene unos cuantos números y sus cuadrados:

Number,Squared  
10,100  
9,81  
22,484

La primera línea se denomina *cabecera*. En este caso, nos indica que las entradas de la primera columna de este archivo son números y las de la segunda columna son las casillas correspondientes. Las tres líneas siguientes, o filas, contienen un número y su casilla separados por una coma. Es posible leer los datos de este archivo utilizando un enfoque similar al que mostré para el archivo *.txt*. Sin embargo, la biblioteca estándar de Python tiene un módulo dedicado (csv) para leer (y escribir) archivos CSV, que facilita un poco las cosas.

Guarda los números y sus cuadrados en un archivo, *números.csv*, en el mismo directorio que tus programas. El siguiente programa muestra cómo leer este archivo y crear un gráfico de dispersión que muestre los números frente a sus cuadrados:

import csv  
import matplotlib.pyplot as plt  
  
def scatter\_plot(x, y):  
plt.scatter(x, y)  
plt.xlabel('Number')  
plt.ylabel('Square')  
plt.show()  
  
def read\_csv(filename):  
  
numbers = []  
squared = []  
with open(filename) as f:  
➊ reader = csv.reader(f)  
next(reader)  
➋ for row in reader:  
numbers.append(int(row[0]))  
squared.append(int(row[1]))  
return numbers, squared  
  
if \_\_name\_\_ == '\_\_main\_\_':  
numbers, squared = read\_csv('numbers.csv')  
scatter\_plot(numbers, squared)

La función read\_csv() lee el archivo CSV utilizando la función reader() definida en el módulo csv (que se importa al principio del programa). Esta función se llama con el objeto de archivo f que se le pasa como argumento ➊. A continuación, esta función devuelve un *puntero* a la primera línea del archivo CSV. Sabemos que la primera línea del archivo es la cabecera, que queremos saltarnos, así que movemos el puntero a la línea siguiente utilizando la función next(). A continuación, leemos cada línea del archivo con cada línea referida por la etiqueta row ➋, con row[0] referida a la primera columna de los datos y row[1] referida a la segunda. Para este archivo concreto, sabemos que estos dos números son enteros, así que utilizamos la función int() para convertirlos de cadenas a enteros y almacenarlos en dos listas. A continuación, se devuelven las listas: una con los números y otra con los cuadrados.

A continuación, llamamos a la función scatter\_plot() con estas dos listas para crear el gráfico de dispersión. La función find\_corr\_x\_y() que escribimos antes también se puede utilizar fácilmente para encontrar el coeficiente de correlación entre los dos conjuntos de números.

Ahora vamos a tratar con un archivo CSV más complejo. Abre [*https://www.google.com/trends/correlate/*](https://www.google.com/trends/correlate/) en tu navegador, introduce la consulta de búsqueda que desees (por ejemplo, *verano*) y haz clic en el botón **Buscar correlaciones**. Verás que se devuelven varios resultados bajo el título "Correlacionado con el verano", y el primer resultado es el que tiene la correlación más alta (el número situado inmediatamente a la izquierda de cada resultado). Haz clic en la opción **Gráfico de dispersión** situada encima del gráfico para ver un gráfico de dispersión con el *eje x* etiquetado como *verano* y el *eje y* etiquetado con el resultado superior. Ignora los números exactos trazados en ambos ejes, ya que sólo nos interesa la correlación y el gráfico de dispersión.

Un poco más arriba del gráfico de dispersión, haz clic en **Exportar datos como CSV** y se iniciará la descarga de un archivo. Guarda este archivo en el mismo directorio que tus programas.

Este archivo CSV es ligeramente diferente del que vimos antes. Al principio del archivo, verás varias líneas en blanco y líneas con el símbolo '#' hasta que finalmente verás la cabecera y los datos. Estas líneas no nos son útiles: elimínalas a mano con el programa con el que hayas abierto el archivo, de modo que la primera línea del archivo sea la cabecera. Elimina también las líneas en blanco del final del archivo. Ahora guarda el archivo. Este paso -en el que limpiamos el archivo para que sea más fácil procesarlo con Python- suele denominarse *preprocesamiento de* los datos.

La cabecera tiene varias columnas. La primera contiene la fecha de los datos de cada fila (cada fila tiene los datos correspondientes a la semana que comenzó en la fecha de esta columna). La segunda columna es la consulta de búsqueda que has introducido, la tercera columna muestra la consulta de búsqueda con *mayor* correlación con tu consulta de búsqueda, y las demás columnas incluyen una serie de otras consultas de búsqueda ordenadas en orden decreciente de correlación con tu consulta de búsqueda introducida. Los números de estas columnas son las *puntuaciones* z de las consultas de búsqueda correspondientes. La puntuación *z* indica la diferencia entre el número de veces que se buscó un término durante una semana concreta y la media general de búsquedas semanales de ese término. Una puntuación *z* positiva indica que el número de búsquedas fue superior a la media de esa semana, y una *puntuación z* negativa indica que fue inferior.

Por ahora, vamos a trabajar sólo con la segunda y la tercera columnas. Puedes utilizar la siguiente función read\_csv() para leer estas columnas:

def read\_csv(filename):  
  
with open(filename) as f:  
reader = csv.reader(f)  
next(reader)  
  
summer = []  
highest\_correlated = []  
➊ for row in reader:  
summer.append(float(row[1]))  
highest\_correlated.append(float(row[2]))  
  
return summer, highest\_correlated

Es muy parecida a la versión anterior de la función read\_csv; el principal cambio aquí es cómo añadimos los valores a cada lista a partir de ➊: ahora estamos leyendo el segundo y el tercer miembro de cada fila, y los estamos almacenando como números en coma flotante.

El siguiente programa utiliza esta función para calcular la correlación entre los valores de la consulta de búsqueda que has proporcionado y los valores de la consulta con mayor correlación con ella. También crea un gráfico de dispersión de estos valores:

import matplotlib.pyplot as plt  
import csv  
  
if \_\_name\_\_ == '\_\_main\_\_':  
➊ summer, highest\_correlated = read\_csv('correlate-summer.csv')  
corr = find\_corr\_x\_y(summer, highest\_correlated)  
print('Highest correlation: {0}'.format(corr))  
scatter\_plot(summer, highest\_correlated)

Suponiendo que el archivo CSV se guardó como *correlate-summer.csv*, llamamos a la función read\_csv() para leer los datos de la segunda y tercera columnas ➊. A continuación, llamamos a la función find\_corr\_x\_y() que escribimos antes con las dos listas summer y highest\_correlated. Nos devuelve el coeficiente de correlación, que luego imprimimos. Ahora, volvemos a llamar a la función scatter\_plot() que escribimos antes con estas dos listas. Antes de ejecutar este programa, tendrás que incluir las definiciones de las funciones read\_csv(), find\_corr\_x\_y() y scatter\_plot().

Al ejecutarlo, verás que imprime el coeficiente de correlación y también crea un gráfico de dispersión. Ambos deberían ser muy similares a los datos mostrados en el sitio web de Google correlate.

### **Lo que has aprendido**

En este capítulo has aprendido a calcular medidas estadísticas para describir un conjunto de números y las relaciones entre conjuntos de números. También has utilizado gráficos para ayudarte a comprender estas medidas. Aprendiste una serie de nuevas herramientas y conceptos de programación mientras escribías programas para calcular estas medidas.

### **Retos de programación**

A continuación, aplica lo que has aprendido para completar los siguientes retos de programación.

#### ***#nº 1: Mejor programa de búsqueda de coeficientes de correlación***

La función find\_corr\_x\_y() que escribimos antes para hallar el coeficiente de correlación entre dos conjuntos de números supone que los dos conjuntos de números tienen la misma longitud. Mejora la función para que compruebe primero la longitud de las listas. Si son iguales, sólo entonces la función debe proceder con los cálculos restantes; de lo contrario, debe imprimir un mensaje de error indicando que no se puede encontrar la correlación.

#### ***#2: Calculadora estadística***

Implementa una calculadora estadística que tome una lista de números del archivo *misdatos.txt* y luego calcule e imprima su media, mediana, moda, varianza y desviación típica utilizando las funciones que escribimos anteriormente en este capítulo.

#### ***#3: Experimenta con otros datos CSV***

Puedes experimentar con numerosas fuentes de datos interesantes disponibles gratuitamente en Internet. El sitio web [*http://www.quandl.com/*](http://www.quandl.com/) es una de esas fuentes. Para este reto, descarga los siguientes datos como archivo CVS de [*http://www.quandl.com/WORLDBANK/USA\_SP\_POP\_TOTL/:*](http://www.quandl.com/WORLDBANK/USA_SP_POP_TOTL/) la población total de Estados Unidos al final de cada año para los años 1960 a 2012. A continuación, calcula la media, la mediana, la varianza y la desviación típica de la *diferencia* de población a lo largo de los años y crea un gráfico que muestre estas diferencias.

#### ***#nº 4: Hallar el percentil***

El percentil es una estadística de uso común que transmite el valor por debajo del cual cae un determinado porcentaje de observaciones. Por ejemplo, si un alumno obtuvo una puntuación percentil 95 en un examen, significa que el 95 por ciento de los alumnos obtuvieron una puntuación inferior o igual a la suya. Otro ejemplo: en la lista de números 5, 1, 9, 3, 14, 9 y 7, el percentil 50 es 7 y el percentil 25 es 3,5, un número que no está presente en la lista.

Hay varias formas de encontrar la observación correspondiente a un percentil determinado, pero aquí tienes un enfoque[.2](footnote.html#fn02)

Supongamos que queremos calcular la observación en el percentil *p*:

1. En orden ascendente, ordena la lista dada de números, que podríamos llamar data.

2. Calcula

image

donde *n* es el número de elementos de data.

3. Si *i* es un número entero, data[i] es el número correspondiente al percentil *p*.

4. Si *i* *no* es un número entero, establece *k* igual a la parte integral de *i* y *f* igual a la parte fraccionaria de *i*. El número (1-f)\*data[k] + f\*data[k+1] es el número correspondiente al percentil *p*.

Utilizando este enfoque, escribe un programa que tome un conjunto de números en un archivo y muestre el número que corresponde a un percentil específico suministrado como entrada al programa.

#### ***#5: Creación de una tabla de frecuencias agrupadas***

Para este reto, tu tarea consiste en escribir un programa que cree una tabla de frecuencias agrupadas a partir de un conjunto de números. Una tabla de frecuencias agrupadas muestra la frecuencia de los datos clasificados en diferentes *clases*. Por ejemplo, consideremos las puntuaciones de las que hablamos en "[Crear una](ch03.html#ch03lev2sec03)tabla de frecuencias" en [la página 69](ch03.html#page_69): 7, 8, 9, 2, 10, 9, 9, 9, 9, 4, 5, 6, 1, 5, 6, 7, 8, 6, 1 y 10. Una tabla de frecuencias agrupadas mostraría estos datos de la siguiente manera:

| **Curso** | **Frecuencia** |
| --- | --- |
| 1-6 | 6 |
| 6-11 | 14 |

La tabla clasifica los grados en dos clases: 1-6 (que incluye 1 pero no 6) y 6-11 (que incluye 6 pero no 11). Muestra frente a ellas el número de grados que pertenecen a cada categoría. Determinar el número de clases y el rango de números de cada clase son dos pasos clave para crear esta tabla. En este ejemplo, he mostrado dos clases con el rango de números de cada clase dividido a partes iguales entre las dos.

He aquí un enfoque sencillo para crear clases, que supone que el número de clases puede elegirse arbitrariamente:

def create\_classes(numbers, n):  
low = min(numbers)  
high = max(numbers)  
  
# Width of each class  
width = (high - low)/n  
classes = []  
a = low  
b = low + width  
classes = []  
while a < (high-width):  
classes.append((a, b))  
a = b  
b = a + width  
# The last class may be of a size that is less than width  
classes.append((a, high+1))  
return classes

La función create\_classes() acepta dos argumentos: una lista de números, numbers, y n, el número de clases a crear. Devolverá una lista de tuplas en la que cada tupla representa una clase. Por ejemplo, si se llama con los números 7, 8, 9, 2, 10, 9, 9, 9, 9, 4, 5, 6, 1, 5, 6, 7, 8, 6, 1, 10, y n = 4, devuelve la siguiente lista: [(1, 3.25), (3.25, 5.5), (5.5, 7.75), (7.75, 11)]. Una vez que tengas la lista, el siguiente paso es repasar cada uno de los números y averiguar a qué clase de las devueltas pertenece.

Tu reto consiste en escribir un programa que lea una lista de números de un fichero y luego imprima la tabla de frecuencias agrupadas, haciendo uso de la función create\_classes().