Capítulo 3: Describir datos con estadísticas

### **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.

[anterior](ch03_7.html)[Subtema 8 de 10: (Ver todo)](ch03.html)[siguiente](ch03_9.html)