# 16 Descargar datos



En este capítulo, descargarás conjuntos de datos de fuentes online y crearás visualizaciones de trabajo de esos datos. Puedes encontrar una increíble variedad de datos en Internet, muchos de los cuales no se han examinado a fondo. La capacidad de analizar estos datos te permite descubrir patrones y conexiones que nadie más ha encontrado.

Accederemos y visualizaremos datos almacenados en dos formatos de datos comunes: CSV y JSON. Utilizaremos el módulo csv de Python para procesar los datos meteorológicos almacenados en el formato CSV y analizar las temperaturas altas y bajas a lo largo del tiempo en dos lugares diferentes. A continuación, utilizaremos Matplotlib para generar un gráfico basado en nuestros datos descargados para mostrar las variaciones de temperatura en dos entornos distintos: Sitka, Alaska, y el Valle de la Muerte, California. Más adelante en el capítulo, utilizaremos el módulo json para acceder a los datos de terremotos almacenados en formato GeoJSON y utilizaremos Plotly para dibujar un mapa del mundo que muestre las ubicaciones y magnitudes de los terremotos recientes.

Al final de este capítulo, estarás preparado para trabajar con varios tipos de conjuntos de datos en diferentes formatos, y tendrás un conocimiento más profundo de cómo construir visualizaciones complejas. Ser capaz de acceder y visualizar datos en línea es esencial para trabajar con una amplia variedad de conjuntos de datos del mundo real.

## El formato de archivo CSV

Una forma sencilla de almacenar datos en un archivo de texto es escribir los datos como una serie de valores separados por comas, denominados *comma-separated values*. Los archivos resultantes son archivos *CSV*. Por ejemplo, aquí tienes un trozo de datos meteorológicos en formato CSV:

"USW00025333","SITKA AIRPORT, AK US","2021-01-01",,"44","40"

Este es un extracto de los datos meteorológicos del 1 de enero de 2021, en Sitka, Alaska. Incluye las temperaturas máxima y mínima del día, así como otras mediciones de ese día. Los archivos CSV pueden resultar tediosos de leer para los humanos, pero los programas pueden procesar y extraer información de ellos con rapidez y precisión.

Empezaremos con un pequeño conjunto de datos meteorológicos en formato CSV registrados en Sitka; está disponible en los recursos de este libro en [https://ehmatthes.github.io/pcc\_3e.](https://ehmatthes.github.io/pcc_3e) Crea una carpeta llamada *weather\_data* dentro de la carpeta donde guardes los programas de este capítulo. Copia el archivo *sitka\_weather\_07-2021\_simple.csv* en esta nueva carpeta. (Después de descargar los recursos de este libro, tendrás todos los archivos que necesitas para este proyecto).

## Nota

Los datos meteorológicos de este proyecto se descargaron originalmente de [https://ncdc.noaa.gov/cdo-web.](https://ncdc.noaa.gov/cdo-web)

### Análisis de las cabeceras del archivo CSV

El módulo csv de la biblioteca estándar de Python analiza las líneas de un archivo CSV y nos permite extraer rápidamente los valores que nos interesan. Empecemos examinando la primera línea del archivo, que contiene una serie de cabeceras para los datos. Estas cabeceras nos indican qué tipo de información contienen los datos:

**sitka\_highs.py**

from pathlib import Path  
import csv  
  
❶ path = Path('weather\_data/sitka\_weather\_07-2021\_simple.csv')  
lines = path.read\_text().splitlines()  
  
❷ reader = csv.reader(lines)  
❸ header\_row = next(reader)  
print(header\_row)

Primero importamos Path y el módulo csv. A continuación, construimos un objeto Path que busca en la carpeta *weather\_data*, y apunta al archivo de datos meteorológicos concreto con el que queremos trabajar ❶. Leemos el archivo y encadenamos el método splitlines() para obtener una lista de todas las líneas del archivo, que asignamos a lines.

A continuación, construimos un objeto reader ❷. Éste es un objeto que puede utilizarse para analizar cada línea del archivo. Para crear un objeto lector, llama a la función csv.reader() y pásale la lista de líneas del archivo CSV.

Cuando se le da un objeto reader, la función next() devuelve la siguiente línea del archivo, empezando por el principio del archivo. Aquí llamamos a next() una sola vez, por lo que obtenemos la primera línea del archivo, que contiene las cabeceras del archivo ❸. Asignamos los datos devueltos a header\_row. Como puedes ver, header\_row contiene cabeceras significativas, relacionadas con el tiempo, que nos indican qué información contiene cada línea de datos:

['STATION', 'NAME', 'DATE', 'TAVG', 'TMAX', 'TMIN']

El objeto reader procesa la primera línea de valores separados por comas del archivo y almacena cada valor como un elemento de una lista. La cabecera STATION representa el código de la estación meteorológica que registró estos datos. La posición de esta cabecera nos indica que el primer valor de cada línea será el código de la estación meteorológica. La cabecera NAME indica que el segundo valor de cada línea es el nombre de la estación meteorológica que realizó la grabación. El resto de las cabeceras especifican qué tipo de información se registró en cada lectura. Los datos que más nos interesan por ahora son la fecha (DATE), la temperatura alta (TMAX) y la temperatura baja (TMIN). Se trata de un conjunto de datos sencillo que sólo contiene datos relacionados con la temperatura. Cuando descargues tus propios datos meteorológicos, puedes optar por incluir otra serie de mediciones relacionadas con la velocidad y dirección del viento, y datos sobre precipitaciones.

### Imprimir las cabeceras y sus posiciones

Para facilitar la comprensión de los datos de cabecera del archivo, vamos a imprimir cada cabecera y su posición en la lista:

**sitka\_highs.py**

--snip--  
reader = csv.reader(lines)  
header\_row = next(reader)  
  
for index, column\_header in enumerate(header\_row):  
 print(index, column\_header)

La función enumerate() devuelve tanto el índice de cada elemento como el valor de cada elemento al recorrer una lista. (Observa que hemos eliminado la línea print(header\_row) en favor de esta versión más detallada).

Aquí tienes la salida mostrando el índice de cada encabezado:

0 STATION  
1 NAME  
2 DATE  
3 TAVG  
4 TMAX  
5 TMIN

Podemos ver que las fechas y sus altas temperaturas se almacenan en las columnas 2 y 4. Para explorar estos datos, procesaremos cada fila de datos en *sitka\_weather\_07-2021\_simple.csv* y extraeremos los valores con los índices 2 y 4.

### Extraer y leer datos

Ahora que sabemos qué columnas de datos necesitamos, vamos a leer algunos de esos datos. En primer lugar, leeremos la temperatura máxima de cada día:

**sitka\_highs.py**

--snip--  
reader = csv.reader(lines)  
header\_row = next(reader)  
  
# Extract high temperatures.  
❶ highs = []  
❷ for row in reader:  
❸ high = int(row[4])  
 highs.append(high)  
  
print(highs)

Creamos una lista vacía llamada highs ❶ y luego recorremos en bucle las filas restantes del archivo ❷. El objeto reader continúa desde donde lo dejó en el archivo CSV y devuelve automáticamente cada línea siguiente a su posición actual. Como ya hemos leído la fila de cabecera, el bucle comenzará en la segunda línea, donde empiezan los datos reales. En cada pasada por el bucle, extraemos los datos del índice 4, correspondiente a la cabecera TMAX, y los asignamos a la variable high ❸. Utilizamos la función int() para convertir los datos, que se almacenan como una cadena, a un formato numérico para que podamos utilizarlos. A continuación, añadimos este valor a highs.

El siguiente listado muestra los datos almacenados ahora en highs:

[61, 60, 66, 60, 65, 59, 58, 58, 57, 60, 60, 60, 57, 58, 60, 61, 63, 63, 70, 64, 59, 63, 61, 58, 59, 64, 62, 70, 70, 73, 66]

Hemos extraído la temperatura alta para cada fecha y almacenado cada valor en una lista. Ahora vamos a crear una visualización de estos datos.

### Trazar los datos en un gráfico de temperatura

Para visualizar los datos de temperatura que tenemos, primero crearemos un gráfico simple de las máximas diarias utilizando Matplotlib, como se muestra aquí:

**sitka\_highs.py**

from pathlib import Path  
import csv  
  
import matplotlib.pyplot as plt  
  
path = Path('weather\_data/sitka\_weather\_07-2021\_simple.csv')  
lines = path.read\_text().splitlines()  
 --snip--  
  
# Plot the high temperatures.  
plt.style.use('seaborn')  
fig, ax = plt.subplots()  
❶ ax.plot(highs, color='red')  
  
# Format plot.  
❷ ax.set\_title("Daily High Temperatures, July 2021", fontsize=24)  
❸ ax.set\_xlabel('', fontsize=16)  
ax.set\_ylabel("Temperature (F)", fontsize=16)  
ax.tick\_params(labelsize=16)  
  
plt.show()

Pasamos la lista de máximos a plot() y pasamos a color='red' para que trace los puntos en rojo ❶.(Trazaremos los máximos en rojo y los mínimos en azul.) A continuación, especificamos algunos otros detalles de formato, como el título, el tamaño de la fuente y las etiquetas ❷, tal como hicimos en el Capítulo 15. Como aún tenemos que añadir las fechas, no etiquetaremos el eje *x*-, pero ax.set\_xlabel() sí modifica el tamaño de la fuente para que las etiquetas por defecto sean más legibles ❸. [La Figura 16-1](#figure16-1) muestra el gráfico resultante: un simple gráfico lineal de las altas temperaturas de julio de 2021 en Sitka, Alaska.

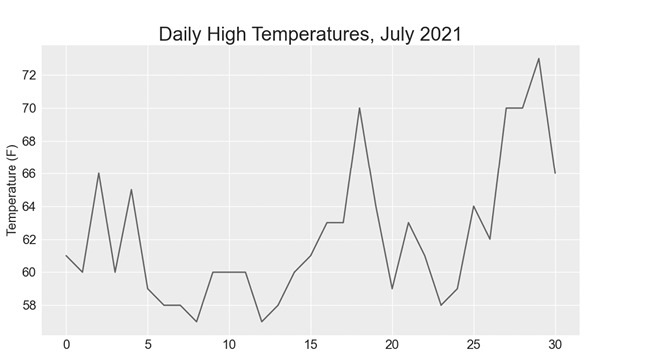


Figura 16-1: Gráfico lineal de las temperaturas máximas diarias de julio de 2021 en Sitka,

Alaska

### El módulo datetime

Vamos a añadir fechas a nuestro gráfico para hacerlo más útil. La primera fecha del archivo de datos meteorológicos está en la segunda fila del archivo:

"USW00025333","SITKA AIRPORT, AK US","2021-07-01",,"61","53"

Los datos se leerán como una cadena, por lo que necesitamos una forma de convertir la cadena "2021-07-01" en un objeto que represente esta fecha. Podemos construir un objeto que represente el 1 de julio de 2021, utilizando el método strptime() del módulo datetime. Veamos cómo funciona strptime() en una sesión de terminal:

>>> from datetime import datetime  
>>> first\_date = datetime.strptime('2021-07-01', '%Y-%m-%d')  
>>> print(first\_date)  
2021-07-01 00:00:00

Primero importamos la clase datetime del módulo datetime. Luego llamamos al método strptime() con la cadena que contiene la fecha que queremos procesar como primer argumento. El segundo argumento indica a Python el formato de la fecha. En este ejemplo, '%Y-' indica a Python que busque un año de cuatro dígitos antes del primer guión; '%m-' indica un mes de dos dígitos antes del segundo guión; y '%d' significa que la última parte de la cadena es el día del mes, del 1 al 31.

El método strptime() puede tomar diversos argumentos para determinar cómo interpretar la fecha. [La Tabla 16-1](#table16-1) muestra algunos de estos argumentos.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| | **Argumento** | **Significado** | | --- | --- | | %A | Nombre del día de la semana, como Lunes | | %B | Nombre del mes, como Enero | | %m | Mes, como número (01 a 12) | | %d | Día del mes, como número (01 a 31) | | %Y | Año de cuatro dígitos, como 2019 | | %y | Año de dos dígitos, como 19 | | %H | Hora, en formato de 24 horas (00 a 23) | | %I | Hora, en formato de 12 horas (01 a 12) | | %p | AM o PM | | %M | Minutos (00 a 59) | | %S | Segundos (00 a 61) | |

Tabla 16-1: Argumentos de formato de fecha y hora del módulo datetime

### Trazar fechas

Podemos mejorar nuestro gráfico extrayendo las fechas de las lecturas diarias de temperatura alta, y utilizando estas fechas en el eje *x*:

**sitka\_highs.py**

from pathlib import Path  
import csv  
from datetime import datetime  
  
import matplotlib.pyplot as plt  
  
path = Path('weather\_data/sitka\_weather\_07-2021\_simple.csv')  
lines = path.read\_text().splitlines()  
  
reader = csv.reader(lines)  
header\_row = next(reader)  
  
# Extract dates and high temperatures.  
❶ dates, highs = [], []  
for row in reader:  
❷ current\_date = datetime.strptime(row[2], '%Y-%m-%d')  
 high = int(row[4])  
 dates.append(current\_date)  
 highs.append(high)  
  
# Plot the high temperatures.  
plt.style.use('seaborn')  
fig, ax = plt.subplots()  
❸ ax.plot(dates, highs, color='red')  
  
# Format plot.  
ax.set\_title("Daily High Temperatures, July 2021", fontsize=24)  
ax.set\_xlabel('', fontsize=16)  
❹ fig.autofmt\_xdate()  
ax.set\_ylabel("Temperature (F)", fontsize=16)  
ax.tick\_params(labelsize=16)  
  
plt.show()

Creamos dos listas vacías para almacenar las fechas y las altas temperaturas del archivo ❶. A continuación, convertimos los datos que contienen la información de la fecha (row[2]) en un objeto datetime ❷ y lo añadimos a dates. Pasamos las fechas y los valores de las altas temperaturas a plot() ❸. La llamada a fig.autofmt\_xdate() ❹ dibuja las etiquetas de las fechas en diagonal para evitar que se solapen. La [Figura 16-2](#figure16-2) muestra el gráfico mejorado.

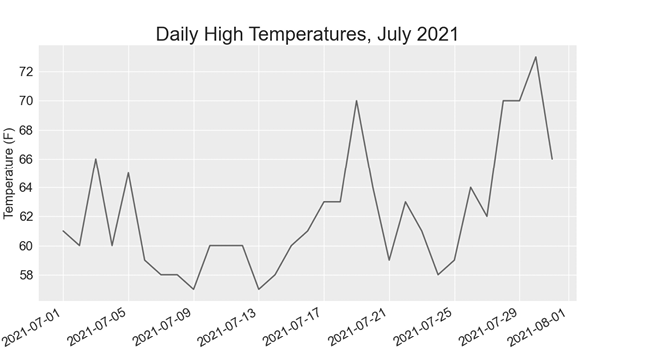


Figura 16-2: El gráfico es más significativo, ahora que tiene fechas

en el eje



*x*.

### Trazar un marco temporal más largo

Con nuestro gráfico preparado, vamos a incluir datos adicionales para obtener una imagen más completa del tiempo en Sitka. Copia el archivo *sitka\_weather\_2021\_simple.csv*, que contiene un año completo de datos meteorológicos de Sitka, en la carpeta donde guardas los datos para los programas de este capítulo.

Ahora podemos generar un gráfico del tiempo de todo el año:

**sitka\_highs.py**

--snip--  
path = Path('weather\_data/sitka\_weather\_2021\_simple.csv')  
lines = path.read\_text().splitlines()  
--snip--  
# Format plot.  
ax.set\_title("Daily High Temperatures, 2021", fontsize=24)  
ax.set\_xlabel('', fontsize=16)  
--snip--

Modificamos el nombre del archivo para utilizar el nuevo archivo de datos *sitka\_weather\_2021\_simple.csv*, y actualizamos el título de nuestro gráfico para reflejar el cambio en su contenido. [La Figura 16-3](#figure16-3) muestra el gráfico resultante.

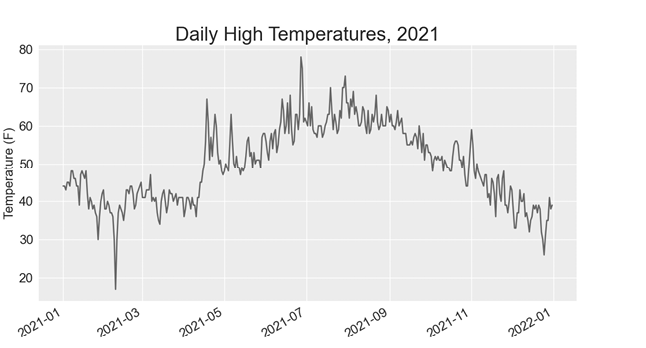


Figura 16-3: Un año

de datos

### Trazar una segunda serie de datos

Podemos hacer que nuestro gráfico sea aún más útil incluyendo las bajas temperaturas. Tenemos que extraer las temperaturas bajas del archivo de datos y añadirlas a nuestro gráfico, como se muestra aquí:

**sitka\_highs\_lows.py**

--snip--  
reader = csv.reader(lines)  
header\_row = next(reader)  
  
# Extract dates, and high and low temperatures.  
❶ dates, highs, lows = [], [], []  
for row in reader:  
 current\_date = datetime.strptime(row[2], '%Y-%m-%d')  
 high = int(row[4])  
❷ low = int(row[5])  
 dates.append(current\_date)  
 highs.append(high)  
 lows.append(low)  
  
# Plot the high and low temperatures.  
plt.style.use('seaborn')  
fig, ax = plt.subplots()  
ax.plot(dates, highs, color='red')  
❸ ax.plot(dates, lows, color='blue')  
  
# Format plot.  
❹ ax.set\_title("Daily High and Low Temperatures, 2021", fontsize=24)  
--snip--

Añadimos la lista vacía lows para contener las temperaturas bajas ❶, y luego extraemos y almacenamos la temperatura baja de cada fecha a partir de la sexta posición de cada fila (row[5]) ❷. Añadimos una llamada a plot() para las temperaturas bajas y coloreamos estos valores de azul ❸. Por último, actualizamos el título ❹. [La Figura 16-4](#figure16-4) muestra el gráfico resultante.

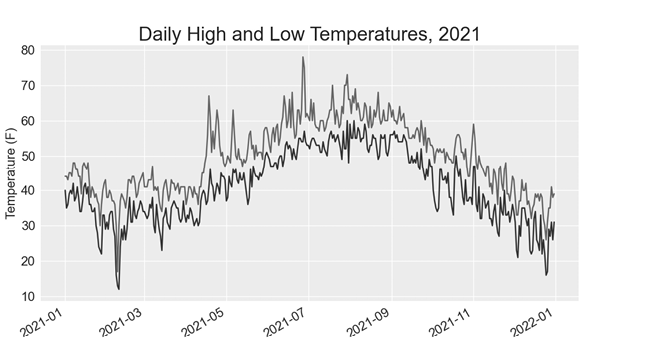


Figura 16-4: Dos series de datos en el mismo

gráfico

### Sombrear un área del gráfico

Habiendo añadido dos series de datos, ahora podemos examinar el rango de temperaturas de cada día. Vamos a dar un toque final al gráfico utilizando el sombreado para mostrar el rango entre las temperaturas máximas y mínimas de cada día. Para ello, utilizaremos el método fill\_between(), que toma una serie de valores *x* y dos series de valores *y* y rellena el espacio entre las dos series de valores *y*:

**sitka\_highs\_lows.py**

--snip--  
# Plot the high and low temperatures.  
plt.style.use('seaborn')  
fig, ax = plt.subplots()  
❶ ax.plot(dates, highs, color='red', alpha=0.5)  
ax.plot(dates, lows, color='blue', alpha=0.5)  
❷ ax.fill\_between(dates, highs, lows, facecolor='blue', alpha=0.1)  
--snip--

El argumento alpha controla la transparencia ❶ de un color . Un valor alpha de 0 es completamente transparente, y un valor de 1 (por defecto) es completamente opaco. Ajustando alpha a 0,5, hacemos que las líneas de trazado rojas y azules parezcan más claras.

Pasamos a fill\_between() la lista dates para los *x*-valores y, a continuación, las dos series *y*-valores highs y lows ❷. El argumento facecolor determina el color de la región sombreada; le damos un valor alpha bajo de 0,1 para que la región rellena conecte las dos series de datos sin distraer de la información que representan. [La Figura 16-5](#figure16-5) muestra el gráfico con la región sombreada entre los máximos y los mínimos.

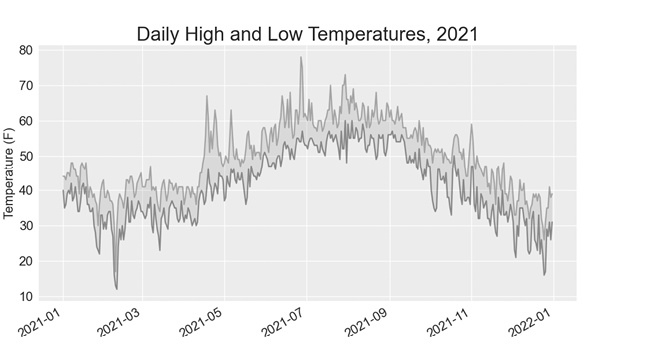


Figura 16-5: La región entre las dos series de datos está sombreada.

El sombreado ayuda a que el rango entre los dos conjuntos de datos sea inmediatamente aparente.

### Comprobación de errores

Deberíamos poder ejecutar el código *sitka\_highs\_lows.py* con los datos de cualquier lugar. Pero algunas estaciones meteorológicas recogen datos distintos de otras, y algunas funcionan mal ocasionalmente y no recogen algunos de los datos que se supone que deben recoger. La falta de datos puede dar lugar a excepciones que bloqueen nuestros programas, a menos que las manejemos adecuadamente.

Por ejemplo, veamos qué ocurre cuando intentamos generar un gráfico de temperatura del Valle de la Muerte, California. Copia el archivo *death\_valley\_2021\_simple.csv* en la carpeta donde guardes los datos para los programas de este capítulo.

Primero, vamos a ejecutar el código para ver las cabeceras que se incluyen en este archivo de datos:

**death\_valley\_highs\_lows.py**

from pathlib import Path  
import csv  
  
path = Path('weather\_data/death\_valley\_2021\_simple.csv')  
lines = path.read\_text().splitlines()  
  
reader = csv.reader(lines)  
header\_row = next(reader)  
  
for index, column\_header in enumerate(header\_row):  
 print(index, column\_header)

Éste es el resultado:

0 STATION  
1 NAME  
2 DATE  
3 TMAX  
4 TMIN  
5 TOBS

La fecha está en la misma posición, en el índice 2. Pero las temperaturas alta y baja están en los índices 3 y 4, por lo que tendremos que cambiar los índices en nuestro código para reflejar estas nuevas posiciones. En lugar de incluir una lectura de la temperatura media del día, esta estación incluye TOBS, una lectura para una hora de observación concreta.

Cambia *sitka\_highs\_lows.py* para generar un gráfico para el Valle de la Muerte utilizando los índices que acabamos de anotar, y observa qué ocurre:

**death\_valley\_highs\_lows.py**

--snip--  
path = Path('weather\_data/death\_valley\_2021\_simple.csv')  
lines = path.read\_text().splitlines()  
 --snip--  
# Extract dates, and high and low temperatures.  
dates, highs, lows = [], [], []  
for row in reader:  
 current\_date = datetime.strptime(row[2], '%Y-%m-%d')  
 high = int(row[3])  
 low = int(row[4])  
 dates.append(current\_date)  
--snip--

Actualizamos el programa para que lea del archivo de datos del Valle de la Muerte, y cambiamos los índices para que correspondan a las posiciones TMAX y TMIN de este archivo.

Cuando ejecutamos el programa, obtenemos un error:

Traceback (most recent call last):  
 File "death\_valley\_highs\_lows.py", line 17, in <module>  
 high = int(row[3])  
❶ ValueError: invalid literal for int() with base 10: ''

El rastreo nos dice que Python no puede procesar la temperatura alta de una de las fechas porque no puede convertir una cadena vacía ('') en un entero ❶. En lugar de buscar entre los datos para averiguar qué lectura falta, trataremos directamente los casos de datos que faltan.

Ejecutaremos código de comprobación de errores cuando se estén leyendo los valores del archivo CSV para gestionar las excepciones que puedan surgir. He aquí cómo hacerlo:

**death\_valley\_highs\_lows.py**

--snip--  
for row in reader:  
 current\_date = datetime.strptime(row[2], '%Y-%m-%d')  
❶ try:  
 high = int(row[3])  
 low = int(row[4])  
 except ValueError:  
❷ print(f"Missing data for {current\_date}")  
❸ else:  
 dates.append(current\_date)  
 highs.append(high)  
 lows.append(low)  
  
# Plot the high and low temperatures.  
--snip--  
  
# Format plot.  
❹ title = "Daily High and Low Temperatures, 2021\nDeath Valley, CA"  
ax.set\_title(title, fontsize=20)  
ax.set\_xlabel('', fontsize=16)  
--snip--

Cada vez que examinamos una fila, intentamos extraer la fecha y la temperatura alta y baja ❶. Si falta algún dato, Python lanzará un ValueError y lo manejamos imprimiendo un mensaje de error que incluye la fecha del dato que falta ❷. Tras imprimir el error, el bucle continuará procesando la siguiente fila. Si todos los datos de una fecha se recuperan sin error, se ejecutará el bloque else y los datos se añadirán a las listas correspondientes ❸. Como estamos trazando la información de una nueva ubicación, actualizamos el título para incluir la ubicación en el gráfico, y utilizamos un tamaño de letra más pequeño para acomodar el título más largo ❹.

Cuando ejecutes ahora *death\_valley\_highs\_lows.py*, verás que sólo faltaban datos de una fecha:

Missing data for 2021-05-04 00:00:00

Como el error se gestiona adecuadamente, nuestro código puede generar un gráfico que omite los datos que faltan. [La Figura 16-6](#figure16-6) muestra el gráfico resultante.

Comparando este gráfico con el de Sitka, podemos ver que el Valle de la Muerte es más cálido en general que el sureste de Alaska, como esperábamos. Además, el rango de temperaturas de cada día es mayor en el desierto. La altura de la región sombreada lo pone de manifiesto.

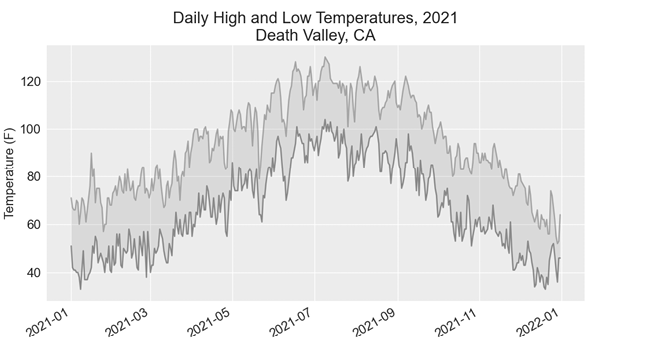


Figura 16-6: Temperaturas máximas y mínimas diarias

del Valle de la Muerte

Muchos conjuntos de datos con los que trabajes tendrán datos ausentes, mal formateados o incorrectos. Puedes utilizar las herramientas que aprendiste en la primera mitad de este libro para manejar estas situaciones. Aquí utilizamos un bloque try-except-else para gestionar los datos que faltan. A veces utilizarás continue para saltarte algunos datos, o utilizarás remove() o del para eliminar algunos datos después de haberlos extraído. Utiliza cualquier método que funcione, siempre que el resultado sea una visualización significativa y precisa.

### Descargar tus propios datos

Para descargar tus propios datos meteorológicos, sigue estos pasos:

1. Visita el sitio Datos Climáticos Online de la NOAA en [https://www.ncdc.noaa.gov/cdo-web.](https://www.ncdc.noaa.gov/cdo-web) En la sección Descubre los datos por, haz clic en **Herramienta de búsqueda**. En la casilla Selecciona un conjunto de datos, elige **Resúmenes diarios**.
2. Selecciona un intervalo de fechas y, en la sección Buscar por, elige **Códigos** postales. Introduce el código postal que te interesa y haz clic en **Buscar**.
3. En la página siguiente, verás un mapa y alguna información sobre la zona en la que te estás centrando. Debajo del nombre de la ubicación, haz clic en **Ver detalles completos**, o haz clic en el mapa y luego en **Detalles completos**.
4. Desplázate hacia abajo y haz clic en **Lista de estaciones** para ver las estaciones meteorológicas disponibles en esta zona. Haz clic en el nombre de una de las estaciones y luego haz clic en **Añadir a la Cesta**. Estos datos son gratuitos, aunque el sitio utiliza un icono de carrito de la compra. En la esquina superior derecha, haz clic en el carrito.
5. En Seleccionar el formato de salida, elige **Personalizado GHCN-Daily CSV**. Asegúrate de que el intervalo de fechas es correcto y haz clic en **Continuar**.
6. En la página siguiente, puedes seleccionar los tipos de datos que deseas. Puedes descargar un tipo de datos (por ejemplo, centrarte en la temperatura del aire) o puedes descargar todos los datos disponibles de esta estación. Haz tu elección y haz clic en **Continuar**.
7. En la última página, verás un resumen de tu pedido. Introduce tu dirección de correo electrónico y haz clic en **Enviar pedido**. Recibirás una confirmación de que se ha recibido tu pedido y, en unos minutos, deberías recibir otro correo electrónico con un enlace para descargar tus datos.

Los datos que descargues deben tener la misma estructura que los datos con los que hemos trabajado en esta sección. Puede que tengan cabeceras diferentes a las que has visto en esta sección, pero si sigues los mismos pasos que hemos utilizado aquí, deberías poder generar visualizaciones de los datos que te interesan.

## Pruébalo tú mismo

16-1. Precipitaciones en Sitka: Sitka está situada en un bosque templado lluvioso, por lo que recibe una buena cantidad de precipitaciones. En el archivo de datos *sitka\_weather\_2021\_full.csv* hay una cabecera llamada PRCP, que representa las cantidades de lluvia diarias. Haz una visualización centrada en los datos de esta columna. Puedes repetir el ejercicio para el Valle de la Muerte si tienes curiosidad por saber lo poco que llueve en un desierto.

16-2. Comparación Sitka-Valle de la Muerte: Las escalas de temperatura de los gráficos de Sitka y del Valle de la Muerte reflejan los diferentes rangos de datos. Para comparar con precisión el rango de temperaturas de Sitka con el del Valle de la Muerte, necesitas escalas idénticas en el eje *y*-. Cambia los ajustes del eje *y* en uno o ambos gráficos de las Figuras 16-5 y 16-6. Luego haz una comparación directa entre los rangos de temperatura de Sitka y del Valle de la Muerte (o de cualquier otro lugar que quieras comparar).

16-3. San Francisco: ¿Se parecen más las temperaturas de San Francisco a las de Sitka o a las del Valle de la Muerte? Descarga algunos datos de San Francisco, y genera un gráfico de temperaturas altas-bajas de San Francisco para hacer una comparación.

16-4. Índices automáticos: En esta sección, hemos codificado los índices correspondientes a las columnas TMIN y TMAX. Utiliza la fila de cabecera para determinar los índices de estos valores, de forma que tu programa pueda funcionar para Sitka o el Valle de la Muerte. Utiliza también el nombre de la estación para generar automáticamente un título apropiado para tu gráfico.

16-5. Explora: Genera algunas visualizaciones más que examinen cualquier otro aspecto meteorológico que te interese para cualquier ubicación sobre la que sientas curiosidad.

## Cartografía de Conjuntos de Datos Globales: Formato GeoJSON

En esta sección, descargarás un conjunto de datos que representa todos los terremotos ocurridos en el mundo durante el mes anterior. Luego harás un mapa que muestre la ubicación de estos terremotos y la importancia de cada uno de ellos. Como los datos están almacenados en formato GeoJSON, trabajaremos con ellos utilizando el módulo json. Utilizando el gráfico scatter\_geo() de Plotly, crearás visualizaciones que muestren claramente la distribución global de los terremotos.

### Descargar datos de terremotos

Crea una carpeta llamada *eq\_data* dentro de la carpeta donde guardas los programas de este capítulo. Copia el archivo *eq\_1\_day\_m1.geojson* en esta nueva carpeta. Los terremotos se clasifican por su magnitud en la escala de Richter. Este archivo incluye los datos de todos los terremotos de magnitud M1 o superior ocurridos en las últimas 24 horas (en el momento de escribir esto). Estos datos proceden de una de las fuentes de datos sobre terremotos del Servicio Geológico de Estados Unidos, en [https://earthquake.usgs.gov/earthquakes/feed.](https://earthquake.usgs.gov/earthquakes/feed)

### Examinar los datos GeoJSON

Cuando abras *eq\_1\_day\_m1.geojson*, verás que es muy denso y difícil de leer:

{"type":"FeatureCollection","metadata":{"generated":1649052296000,...  
{"type":"Feature","properties":{"mag":1.6,"place":"63 km SE of Ped...  
{"type":"Feature","properties":{"mag":2.2,"place":"27 km SSE of Ca...  
{"type":"Feature","properties":{"mag":3.7,"place":"102 km SSE of S...  
{"type":"Feature","properties":{"mag":2.92000008,"place":"49 km SE...  
{"type":"Feature","properties":{"mag":1.4,"place":"44 km NE of Sus...  
--snip--

Este archivo está formateado más para máquinas que para humanos. Pero podemos ver que el archivo contiene algunos diccionarios, así como información que nos interesa, como las magnitudes y ubicaciones de los terremotos.

El módulo json proporciona diversas herramientas para explorar y trabajar con datos JSON. Algunas de estas herramientas nos ayudarán a reformatear el archivo para que podamos ver los datos en bruto más fácilmente antes de trabajar con ellos mediante programación.

Empecemos por cargar los datos y mostrarlos en un formato más fácil de leer. Se trata de un archivo de datos largo, así que en lugar de imprimirlo, reescribiremos los datos en un nuevo archivo. Así podremos abrir ese archivo y desplazarnos hacia adelante y hacia atrás por los datos más fácilmente:

**eq\_explore\_data.py**

from pathlib import Path  
import json  
  
# Read data as a string and convert to a Python object.  
path = Path('eq\_data/eq\_data\_1\_day\_m1.geojson')  
contents = path.read\_text()  
❶ all\_eq\_data = json.loads(contents)  
  
# Create a more readable version of the data file.  
❷ path = Path('eq\_data/readable\_eq\_data.geojson')  
❸ readable\_contents = json.dumps(all\_eq\_data, indent=4)  
path.write\_text(readable\_contents)

Leemos el archivo de datos como una cadena, y utilizamos json.loads() para convertir la representación de cadena del archivo en un objeto ❶ de Python . Este es el mismo enfoque que utilizamos en el Capítulo 10. En este caso, todo el conjunto de datos se convierte en un único diccionario, que asignamos a all\_eq\_data. A continuación, definimos un nuevo path en el que podemos escribir estos mismos datos en un formato ❷ más legible . La función json.dumps() que viste en el Capítulo 10 puede tomar un argumento opcional indent ❸, que le indica cuánto sangrar los elementos anidados en la estructura de datos.

Cuando busques en tu directorio *eq\_data* y abras el archivo *readable\_eq\_data.json*, esto es lo primero que verás:

**readable\_eq\_data.json**

{  
 "type": "FeatureCollection",  
❶ "metadata": {  
 "generated": 1649052296000,  
 "url": "https://earthquake.usgs.gov/earthquakes/.../1.0\_day.geojson",  
 "title": "USGS Magnitude 1.0+ Earthquakes, Past Day",  
 "status": 200,  
 "api": "1.10.3",  
 "count": 160  
 },  
❷ "features": [  
 --snip--

La primera parte del archivo incluye una sección con la clave "metadata"❶. Esto nos indica cuándo se generó el archivo de datos y dónde podemos encontrar los datos en Internet. También nos da un título legible por humanos y el número de terremotos incluidos en este archivo. En este periodo de 24 horas, se registraron 160 terremotos.

Este archivo GeoJSON tiene una estructura que resulta útil para los datos basados en la ubicación. La información se almacena en una lista asociada a la clave "features" ❷. Como este archivo contiene datos sobre terremotos, los datos están en forma de lista, donde cada elemento de la lista corresponde a un único terremoto. Esta estructura puede parecer confusa, pero es bastante potente. Permite a los geólogos almacenar toda la información que necesiten en un diccionario sobre cada terremoto, y luego meter todos esos diccionarios en una gran lista.

Veamos un diccionario que representa un único terremoto:

**readable\_eq\_data.json**

--snip--  
 {  
 "type": "Feature",  
❶ "properties": {  
 "mag": 1.6,  
 --snip--  
❷ "title": "M 1.6 - 27 km NNW of Susitna, Alaska"  
 },  
❸ "geometry": {  
 "type": "Point",  
 "coordinates": [  
❹ -150.7585,  
❺ 61.7591,  
 56.3  
 ]  
 },  
 "id": "ak0224bju1jx"  
 },

La clave "properties" contiene mucha información sobre cada terremoto ❶. Nos interesa sobre todo la magnitud de cada terremoto, asociada a la clave "mag". También nos interesa el "title" de cada suceso, que proporciona un buen resumen de su magnitud y localización ❷.

La clave "geometry" nos ayuda a comprender dónde se produjo el terremoto ❸. Necesitaremos esta información para cartografiar cada suceso. Podemos encontrar la longitud ❹ y la latitud ❺ de cada terremoto en una lista asociada a la clave "coordinates".

Este archivo contiene mucho más anidamiento del que utilizaríamos en el código que escribimos, así que si te parece confuso, no te preocupes: Python se encargará de la mayor parte de la complejidad. Sólo trabajaremos con uno o dos niveles de anidamiento cada vez. Empezaremos sacando un diccionario para cada terremoto que se haya registrado en el periodo de 24 horas.

## Nota

Cuando hablamos de lugares, a menudo decimos primero la latitud del lugar, seguida de su longitud. Esta convención surgió probablemente porque los humanos descubrimos la latitud mucho antes de que desarrolláramos el concepto de longitud. Sin embargo, muchos marcos geoespaciales indican primero la longitud y luego la latitud, porque esto se corresponde con la convención (*x*, *y*) que utilizamos en las representaciones matemáticas. El formato GeoJSON sigue la convención (longitud, latitud). Si utilizas un marco diferente, es importante que aprendas qué convención sigue ese marco.

### Hacer una lista de todos los terremotos

En primer lugar, haremos una lista que contenga toda la información sobre cada terremoto ocurrido.

**eq\_explore\_data.py**

from pathlib import Path  
import json  
  
# Read data as a string and convert to a Python object.  
path = Path('eq\_data/eq\_data\_1\_day\_m1.geojson')  
contents = path.read\_text()  
all\_eq\_data = json.loads(contents)  
  
# Examine all earthquakes in the dataset.  
all\_eq\_dicts = all\_eq\_data['features']  
print(len(all\_eq\_dicts))

Tomamos los datos asociados a la clave 'features' del diccionario all\_eq\_data, y los asignamos a all\_eq\_dicts. Sabemos que este archivo contiene registros de 160 terremotos, y la salida verifica que hemos capturado todos los terremotos del archivo:

160

Fíjate en lo corto que es este código. El archivo *readable\_eq\_data.json*, pulcramente formateado, tiene más de 6.000 líneas. Pero en sólo unas líneas, podemos leer todos esos datos y almacenarlos en una lista de Python. A continuación, sacaremos las magnitudes de cada terremoto.

### Extraer magnitudes

Podemos recorrer la lista que contiene los datos de cada terremoto y extraer la información que queramos. Extraigamos la magnitud de cada terremoto:

**eq\_explore\_data.py**

--snip--  
all\_eq\_dicts = all\_eq\_data['features']  
  
❶ mags = []  
for eq\_dict in all\_eq\_dicts:  
❷ mag = eq\_dict['properties']['mag']  
 mags.append(mag)  
  
print(mags[:10])

Creamos una lista vacía para almacenar las magnitudes, y luego hacemos un bucle a través de la lista all\_eq\_dicts ❶. Dentro de este bucle, cada terremoto está representado por el diccionario eq\_dict. La magnitud de cada terremoto se almacena en la sección 'properties' de este diccionario, bajo la clave 'mag' ❷. Almacenamos cada magnitud en la variable mag y luego la añadimos a la lista mags.

Imprimimos las primeras magnitudes 10, para saber si estamos obteniendo los datos correctos:

[1.6, 1.6, 2.2, 3.7, 2.92000008, 1.4, 4.6, 4.5, 1.9, 1.8]

A continuación, sacaremos los datos de localización de cada terremoto, y así podremos hacer un mapa de los terremotos.

### Extraer los datos de localización

Los datos de localización de cada terremoto se almacenan bajo la clave "geometry". Dentro del diccionario de geometría hay una clave "coordinates", y los dos primeros valores de esta lista son la longitud y la latitud. He aquí cómo extraeremos estos datos:

**eq\_explore\_data.py**

--snip--  
all\_eq\_dicts = all\_eq\_data['features']  
  
mags, lons, lats = [], [], []  
for eq\_dict in all\_eq\_dicts:  
 mag = eq\_dict['properties']['mag']  
❶ lon = eq\_dict['geometry']['coordinates'][0]  
 lat = eq\_dict['geometry']['coordinates'][1]  
 mags.append(mag)  
 lons.append(lon)  
 lats.append(lat)  
  
print(mags[:10])  
print(lons[:5])  
print(lats[:5])

Hacemos listas vacías para las longitudes y latitudes. La clave eq\_dict['geometry'] accede al diccionario que representa el elemento de geometría del terremoto ❶. La segunda clave, 'coordinates', extrae la lista de valores asociada a 'coordinates'. Por último, el índice 0 pide el primer valor de la lista de coordenadas, que corresponde a la longitud de un terremoto.

Cuando imprimimos las primeras longitudes y latitudes de 5, la salida muestra que estamos extrayendo los datos correctos:

[1.6, 1.6, 2.2, 3.7, 2.92000008, 1.4, 4.6, 4.5, 1.9, 1.8]  
[-150.7585, -153.4716, -148.7531, -159.6267, -155.248336791992]  
[61.7591, 59.3152, 63.1633, 54.5612, 18.7551670074463]

Con estos datos, podemos pasar a cartografiar cada terremoto.

### Construir un mapa del mundo

Con la información que hemos obtenido hasta ahora, podemos construir un sencillo mapamundi. Aunque todavía no tendrá un aspecto presentable, queremos asegurarnos de que la información se muestra correctamente antes de centrarnos en cuestiones de estilo y presentación. Éste es el mapa inicial:

**eq\_world\_map.py**

from pathlib import Path  
import json  
  
import plotly.express as px  
  
--snip--  
for eq\_dict in all\_eq\_dicts:  
 --snip--  
  
title = 'Global Earthquakes'  
❶ fig = px.scatter\_geo(lat=lats, lon=lons, title=title)  
fig.show()

Importamos plotly.express con el alias px, igual que hicimos en el Capítulo 15. La función scatter\_geo() ❶ te permite superponer un gráfico de dispersión de datos geográficos sobre un mapa. En el uso más sencillo de este tipo de gráfico, sólo necesitas proporcionar una lista de latitudes y otra de longitudes. Pasamos la lista lats al argumento lat, y lons al argumento lon.

Cuando ejecutes este archivo, deberías ver un mapa parecido al de la [Figura 16-7](#figure16-7). Esto demuestra una vez más la potencia de la biblioteca Plotly Express; en sólo tres líneas de código, tenemos un mapa de la actividad sísmica mundial.

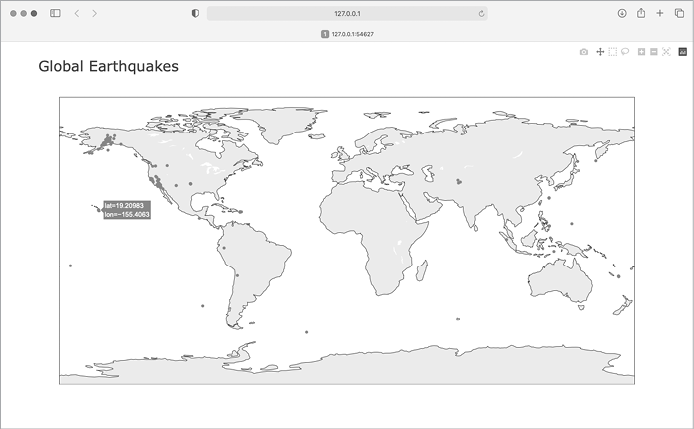


Figura 16

-7



: Un sencillo mapa que muestra dónde se produjeron todos los terremotos de las últimas 24

horas

Ahora que sabemos que la información de nuestro conjunto de datos se traza correctamente, podemos hacer algunos cambios para que el mapa tenga más sentido y sea más fácil de leer.

### Representación de magnitudes

Un mapa de actividad sísmica debe mostrar la magnitud de cada terremoto. También podemos incluir más datos, ahora que sabemos que los datos se trazan correctamente.

--snip--  
# Read data as a string and convert to a Python object.  
path = Path('eq\_data/eq\_data\_30\_day\_m1.geojson')  
contents = path.read\_text()  
--snip--  
  
title = 'Global Earthquakes'  
fig = px.scatter\_geo(lat=lats, lon=lons, size=mags, title=title)  
fig.show()

Cargamos el archivo *eq\_data\_30\_day\_m1.geojson*, para incluir 30 días completos de actividad sísmica. También utilizamos el argumento tamaño en la llamada a px.scatter\_geo(), que especifica cómo se dimensionarán los puntos del mapa. Pasamos la lista mags a size, para que los terremotos de mayor magnitud aparezcan como puntos más grandes en el mapa.

El mapa resultante se muestra en la [Figura 16-8](#figure16-8). Los terremotos suelen producirse cerca de los límites de las placas tectónicas, y el periodo más largo de actividad sísmica incluido en este mapa revela las ubicaciones exactas de estos límites.

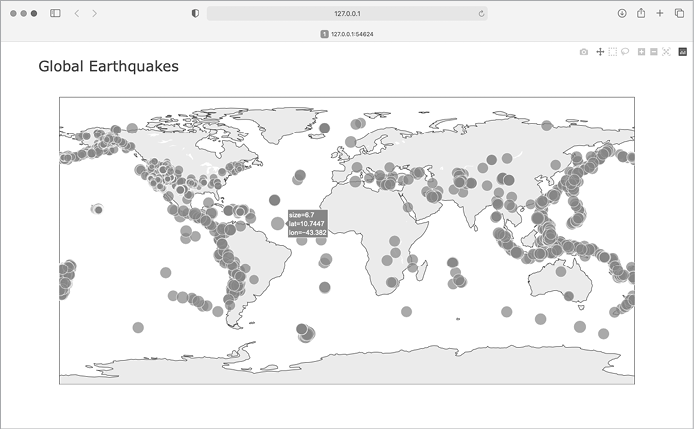


Figura16-8: El mapa muestra ahora la magnitud de todos los terremotos de los últimos 30 días.

Este mapa es mejor, pero sigue siendo difícil distinguir qué puntos representan los terremotos más importantes. Podemos mejorarlo aún más utilizando también el color para representar las magnitudes.

### Personalizar los colores de los marcadores

Podemos utilizar las escalas de color de Plotly para personalizar el color de cada marcador, según la gravedad del terremoto correspondiente. También utilizaremos una proyección diferente para el mapa base.

**eq\_world\_map.py**

--snip--  
fig = px.scatter\_geo(lat=lats, lon=lons, size=mags, title=title,  
❶ color=mags,  
❷ color\_continuous\_scale='Viridis',  
❸ labels={'color':'Magnitude'},  
❹ projection='natural earth',  
 )  
fig.show()

Todos los cambios significativos aquí se producen en la llamada a la función px.scatter\_geo(). El argumento color indica a Plotly qué valores debe utilizar para determinar dónde cae cada marcador en la escala de colores ❶. Utilizamos la lista mags para determinar el color de cada punto, igual que hicimos con el argumento size.

El argumento color\_continuous\_scale indica a Plotly qué escala de color debe utilizar ❷. *Viridis* es una escala de color que va del azul oscuro al amarillo brillante, y funciona bien para este conjunto de datos. Por defecto, la escala de colores de la derecha del mapa está etiquetada como *color*; esto no es representativo de lo que significan realmente los colores. El argumento labels, mostrado en el Capítulo 15, toma como valor un diccionario ❸. Sólo tenemos que establecer una etiqueta personalizada en este gráfico, asegurándonos de que la escala de colores se etiqueta *Magnitude* en lugar de *color*.

Añadimos un argumento más, para modificar el mapa base sobre el que se trazan los terremotos. El argumento projection acepta una serie de proyecciones cartográficas ❹ comunes . Aquí utilizamos la proyección 'natural earth', que redondea los extremos del mapa. Fíjate también en la coma que aparece después de este último argumento. Cuando una llamada a una función tiene una larga lista de argumentos que abarca varias líneas como ésta, es práctica común añadir una coma al final para que siempre estés preparado para añadir otro argumento en la línea siguiente.

Cuando ejecutes ahora el programa, verás un mapa de aspecto mucho más agradable. En [la Figura 16-9](#figure16-9), la escala de colores muestra la gravedad de los terremotos individuales; los terremotos más graves destacan como puntos de color amarillo claro, en contraste con muchos puntos más oscuros. También puedes saber qué regiones del mundo tienen una actividad sísmica más significativa.

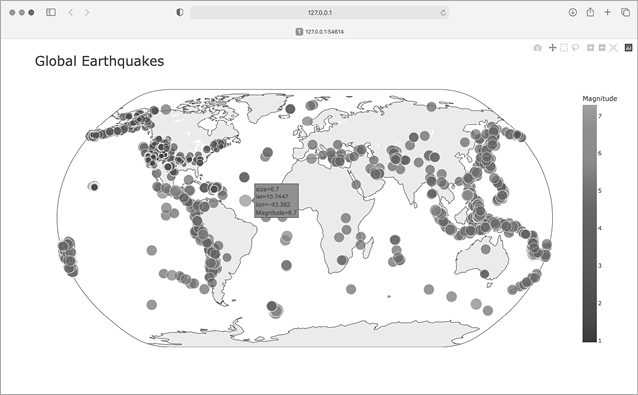


Figura 16-9: En 30 días de terremotos, el color y el tamaño se utilizan para representar la magnitud de cada terremoto.

### Otras escalas de color

Puedes elegir entre otras escalas de color. Para ver las escalas de color disponibles, introduce las dos líneas siguientes en una sesión de terminal Python:

>>> import plotly.express as px  
>>> px.colors.named\_colorscales()  
['aggrnyl', 'agsunset', 'blackbody', ..., 'mygbm']

No dudes en probar estas escalas de color en el mapa de terremotos, o con cualquier conjunto de datos en el que la variación continua de los colores pueda ayudar a mostrar patrones en los datos.

### Añadir texto sobreimpresionado

Para terminar este mapa, añadiremos un texto informativo que aparecerá cuando pases el ratón por encima del marcador que representa un terremoto. Además de mostrar la longitud y la latitud, que aparecen por defecto, mostraremos también la magnitud y proporcionaremos una descripción de la ubicación aproximada.

Para realizar este cambio, necesitamos extraer un poco más de datos del archivo:

**eq\_world\_map.py**

--snip--  
❶ mags, lons, lats, eq\_titles = [], [], [], []  
 mag = eq\_dict['properties']['mag']  
 lon = eq\_dict['geometry']['coordinates'][0]  
 lat = eq\_dict['geometry']['coordinates'][1]  
❷ eq\_title = eq\_dict['properties']['title']  
 mags.append(mag)  
 lons.append(lon)  
 lats.append(lat)  
 eq\_titles.append(eq\_title)  
  
title = 'Global Earthquakes'  
fig = px.scatter\_geo(lat=lats, lon=lons, size=mags, title=title,  
 --snip--  
 projection='natural earth',  
❸ hover\_name=eq\_titles,  
 )  
fig.show()

Primero hacemos una lista llamada eq\_titles para almacenar el título de cada terremoto ❶. La sección 'title' de los datos contiene un nombre descriptivo de la magnitud y ubicación de cada terremoto, además de su longitud y latitud. Extraemos esta información y la asignamos a la variable eq\_title ❷, y luego la añadimos a la lista eq\_titles.

En la llamada a px.scatter\_geo(), pasamos eq\_titles al argumento hover\_name ❸. Plotly añadirá ahora la información del título de cada terremoto al texto hover de cada punto. Cuando ejecutes este programa, deberías poder pasar el ratón por encima de cualquier marcador, ver una descripción de dónde tuvo lugar ese terremoto y leer su magnitud exacta. En la [Figura 16-10](#figure16-10) se muestra un ejemplo de esta información.

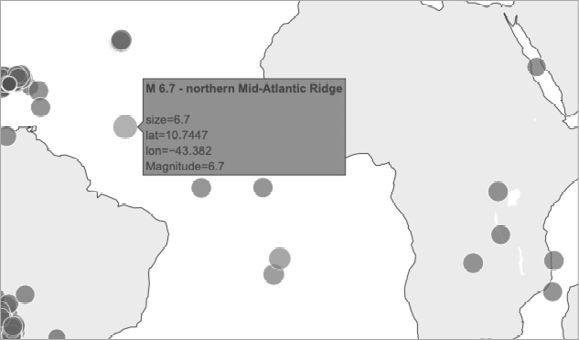


Figura16-10: El texto que aparece al pasar el ratón incluye ahora un resumen de cada terremoto.

¡Es impresionante! En menos de 30 líneas de código, hemos creado un mapa visualmente atractivo y significativo de la actividad sísmica mundial que también ilustra en la estructura geológica del planeta. Plotly ofrece una amplia gama de formas de personalizar el aspecto y el comportamiento de tus visualizaciones. Utilizando las numerosas opciones de Plotly, puedes hacer gráficos y mapas que muestren exactamente lo que quieres que muestren.

## Pruébalo tú mismo

16-6. Refactorización: El bucle que extrae datos de all\_eq\_dicts utiliza variables para la magnitud, la longitud, la latitud y el título de cada terremoto antes de añadir estos valores a sus listas correspondientes. Este enfoque se eligió para aclarar cómo extraer datos de un archivo GeoJSON, pero no es necesario en tu código. En lugar de utilizar estas variables temporales, extrae cada valor de eq\_dict y añádelo a la lista correspondiente en una sola línea. Haciendo esto acortarás el cuerpo de este bucle a sólo cuatro líneas.

16-7. Título automatizado: En esta sección, hemos utilizado el título genérico *Global Earthquakes*. En su lugar, puedes utilizar el título del conjunto de datos en la parte de metadatos del archivo GeoJSON. Extrae este valor y asígnalo a la variable title.

16-8. Terremotos recientes: Puedes encontrar en Internet archivos de datos que contienen información sobre los terremotos más recientes en periodos de 1 hora, 1 día, 7 días y 30 días. Entra en <https://earthquake.usgs.gov/earthquakes/feed/v1.0/geojson.php> y verás una lista de enlaces a conjuntos de datos para varios periodos de tiempo, centrados en terremotos de diferentes magnitudes. Descarga uno de estos conjuntos de datos y crea una visualización de la actividad sísmica más reciente.

16-9. Incendios en el Mundo: En los recursos para este capítulo, encontrarás un archivo llamado *world\_fires\_1\_day.csv*. Este archivo contiene información sobre los incendios que arden en diferentes lugares del planeta, incluyendo la latitud, la longitud y la luminosidad de cada incendio. Utilizando el trabajo de procesamiento de datos de la primera parte de este capítulo y el trabajo de cartografía de esta sección, haz un mapa que muestre qué partes del mundo están afectadas por incendios.

Puedes descargar versiones más recientes de estos datos en [https://earthdata.nasa.gov/earth-observation-data/near-real-time/firms/active-fire-data.](https://earthdata.nasa.gov/earth-observation-data/near-real-time/firms/active-fire-data) Puedes encontrar enlaces a los datos en formato CSV en la sección *SHP, KML, and TXT Files*.

## Resumen

En este capítulo has aprendido a trabajar con conjuntos de datos del mundo real. Procesaste archivos CSV y GeoJSON, y extrajiste los datos en los que querías centrarte. Utilizando datos meteorológicos históricos, aprendiste más sobre cómo trabajar con Matplotlib, incluyendo cómo utilizar el módulo datetime y cómo trazar múltiples series de datos en un gráfico. Trazaste datos geográficos en un mapa del mundo en Plotly, y aprendiste a personalizar el estilo del mapa.

A medida que adquieras experiencia trabajando con archivos CSV y JSON, podrás procesar casi cualquier dato que quieras analizar. Puedes descargar la mayoría de los conjuntos de datos en línea en uno de estos formatos o en ambos. Trabajando con estos formatos de , también podrás aprender a trabajar con otros formatos de datos más fácilmente.

En el próximo capítulo, escribirás programas que recojan automáticamente sus propios datos de fuentes online, y luego crearás visualizaciones de esos datos. Estas son habilidades divertidas si quieres programar como hobby y son habilidades fundamentales si te interesa programar profesionalmente.