

Uso Básico de API

Ricardo R. Palma

2023-03-29

##Introducción a las API con R

“**API**” es un término general para el lugar donde un programa de computadora interactúa con otro, o consigo mismo. En este tutorial, trabajaremos específicamente con API web, donde dos computadoras diferentes, un cliente y un servidor, interactuarán entre sí para solicitar y proporcionar datos, respectivamente.

Las API ofrecen a los científicos de datos una forma pulida de solicitar datos limpios y seleccionados de un sitio web. Cuando un sitio web como Facebook configura una API, básicamente está configurando una computadora que espera solicitudes de datos.

Una vez que esta computadora recibe una solicitud de datos, hará su propio procesamiento de los datos y los enviará a la computadora que los solicitó. Desde nuestra perspectiva como solicitante, necesitaremos escribir código en R que cree la solicitud y le diga a la computadora que ejecuta la API lo que necesitamos. Esa computadora luego leerá nuestro código, procesará la solicitud y devolverá datos con un formato agradable que las bibliotecas R existentes pueden analizar fácilmente.

¿Por qué es esto valioso? Compare el enfoque de la API con el web scraping puro. Cuando un programador raspa una página web, recibe los datos en un fragmento desordenado de HTML. Vea la biblioteca **curl** por ejemplo. Si bien ciertamente existen bibliotecas que facilitan el análisis de texto HTML, ¡todos estos son pasos de limpieza que deben tomarse antes de que incluso tengamos en nuestras manos los datos que queremos!

A menudo, podemos usar inmediatamente los datos que obtenemos de una API, lo que nos ahorra tiempo y frustraciones.

Hacer solicitudes de API en R

Para trabajar con API en R, necesitamos incorporar algunas bibliotecas. Estas bibliotecas toman todas las complejidades de una solicitud de API y las envuelven en funciones que podemos usar en líneas de código individuales. Las bibliotecas R que usaremos son `httr` y `jsonlite`. Cumplen diferentes funciones en nuestra introducción de las API, pero ambas son esenciales.

Si no tiene ninguna de estas bibliotecas en su consola R o RStudio, primero deberá descargarlas. Use la función `install.packages()` para traer estos paquetes.

```
# install.packages(c("httr", "jsonlite"))
installed.packages("httr")
```

```
##      Package LibPath Version Priority Depends Imports LinkingTo Suggests
##      Enhances License License_is_FOSS License_restricts_use OS_type Archs
##      MD5sum NeedsCompilation Built
```

```
installed.packages("jsonlite")
```

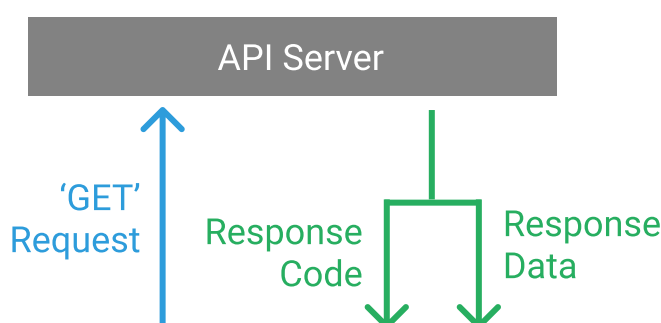
```
## Package LibPath Version Priority Depends Imports LinkingTo Suggests
## Enhances License License_is_FOSS License_restricts_use OS_type Archs
## MD5sum NeedsCompilation Built
```

Cumplido este paso cargaremos las bibliotecas

```
library(httr)
library(jsonlite)
```

Haciendo nuestra primera solicitud de API

Si bien algunos servidores de API te requerirán contraseña y usuario, este ejemplo usa un servidor que omite esta verificación. El primer paso para obtener datos de una API es realizar la solicitud real en R. Esta solicitud se enviará al servidor de la computadora que tiene la API y, suponiendo que todo transcurra sin problemas, enviará una respuesta. El siguiente gráfico ilustra esto:



Aquí hay varios tipos de

solicitudes que uno puede hacer a un servidor API. Este tipo de solicitudes corresponden a diferentes acciones que desea que realice el servidor.

Para nuestros propósitos, solo solicitaremos datos, lo que corresponde a una solicitud GET. Otros tipos de solicitudes son **POST** y **PUT**, pero no tendremos que preocuparnos por ellas a los efectos de este tutorial de API de R centrado en la ciencia de datos.

Para crear una solicitud GET, necesitamos usar la función **GET()** de la biblioteca `httr`. La función `GET()` requiere una URL, que especifica la dirección del servidor al que debe enviarse la solicitud.

La función **GET()** encapsula toda la complejidad de una solicitud GET. Para nuestro ejemplo, trabajaremos con *Open Notify API*, que abre datos sobre varios proyectos de la NASA. Usando la API Open Notify, podemos conocer la ubicación de la Estación Espacial Internacional y cuántas personas hay actualmente en el espacio y otros datos.

Primero trabajaremos con la última API. Comenzaremos haciendo nuestra solicitud usando la función **GET()** y especificando la URL de la API:

```
res = GET("http://api.open-notify.org/astros.json")
res
```

```
## Response [http://api.open-notify.org/astros.json]
## Date: 2023-03-29 14:54
## Status: 200
## Content-Type: application/json
## Size: 500 B
```

El resultado de la función `GET()` es una lista que contiene toda la información que devuelve el servidor API. En otras palabras, la variable `res` contiene la respuesta del servidor API a nuestra solicitud

Examinando la salida `GET()`

Echemos un vistazo a cómo se ve la variable `res` en la consola R:

```
>> res
Respuesta [https://api.open-notify.org/astros.json]
Fecha: 2020-01-30 18:07
Estado: 200
Tipo de contenido: aplicación/json
Tamaño: 314B
```

Investigar la variable `res` nos da una mirada resumida a la respuesta resultante. Lo primero que debe notar es que contiene la URL a la que se envió la solicitud `GET`. También podemos ver la fecha y hora en que se realizó la solicitud, así como el tamaño de la respuesta.

El tipo de contenido nos da una idea de qué forma toman los datos. Esta respuesta en particular dice que los datos adquieren un formato `json`, lo que da una pista sobre por qué necesitamos la biblioteca `jsonlite`.

El estado merece una atención especial. “Estado” se refiere al éxito o fracaso de la solicitud de la API y se presenta en forma de número. El número devuelto le indica si la solicitud fue exitosa o no y también puede detallar algunas razones por las que podría haber fallado.

El número 200 es lo que queremos ver; corresponde a una solicitud exitosa, y eso es lo que tenemos aquí. Hay más información sobre otros códigos de estado que podemos encontrar aquí. Dado que tenemos una respuesta de estado 200 exitosa, sabemos que tenemos los datos disponibles y podemos comenzar a trabajar con ellos.

Manejo de datos JSON

JSON significa Notación de objetos de JavaScript. Si bien JavaScript es otro lenguaje de programación, nuestro enfoque en JSON es su estructura. JSON es útil porque una computadora lo puede leer fácilmente y, por esta razón, se ha convertido en la forma principal en que los datos se transportan a través de las API. La mayoría de las API enviarán sus respuestas en formato JSON.

JSON tiene el formato de una serie de pares clave-valor, donde una palabra en particular (“clave”) se asocia con un valor en particular. A continuación se muestra un ejemplo de esta estructura clave-valor:

```
{
  "name": "Jane Doe",
  "number_of_skills": 2
}
```

Es claro que este tutorial no funcionará con el nombre de Jane si repetimos estos pasos en años posteriores a la escritura de este texto.

En su estado actual, los datos de la variable `res` no se pueden utilizar. Los datos reales están contenidos como Unicode sin procesar en la lista `res`, que finalmente debe convertirse al formato JSON.

Para hacer esto, primero debemos convertir el Unicode sin procesar en un vector de caracteres que se asemeje al formato JSON que se muestra arriba. La función **`rawToChar()`** realiza solo esta tarea, como se muestra a continuación:

Loading [MathJax]/extensions/MathZoom.js

```
rawToChar(res$content)
```

```
## [1] "{\"message\": \"success\", \"number\": 10, \"people\": [{\"craft\": \"ISS\", \"name\": \"Sergey Prokopyev\"}, {\"craft\": \"ISS\", \"name\": \"Dmitry Petelin\"}, {\"craft\": \"ISS\", \"name\": \"Frank Rubio\"}, {\"craft\": \"Shenzhou 15\", \"name\": \"Fei Junlong\"}, {\"craft\": \"Shenzhou 15\", \"name\": \"Deng Qingming\"}, {\"craft\": \"Shenzhou 15\", \"name\": \"Zhang Lu\"}, {\"craft\": \"ISS\", \"name\": \"Stephen Bowen\"}, {\"craft\": \"ISS\", \"name\": \"Warren Hoburg\"}, {\"craft\": \"ISS\", \"name\": \"Sultan Alneyadi\"}, {\"craft\": \"ISS\", \"name\": \"Andrey Fedyaev\"}]}"
```

Si bien la cadena resultante parece desordenada, es realmente la estructura JSON en formato de caracteres.

Desde un vector de caracteres, podemos convertirlo en una estructura de datos de lista usando la función **fromJSON()** de la biblioteca jsonlite.

La función **fromJSON()** necesita un vector de caracteres que contenga la estructura JSON, que es lo que obtuvimos de la salida **rawToChar()**. Entonces, si unimos estas dos funciones, obtendremos los datos que queremos en un formato que podemos manipular más fácilmente en R.

```
data = fromJSON(rawToChar(res$content))
names(data)
```

```
## [1] "message" "number" "people"
```

En nuestra variable de datos, el conjunto de datos que nos interesa ver está contenido en el marco de datos de personas. Podemos usar el operador **\$** para ver directamente este marco de datos:

```
data$people
```

```
##      craft      name
## 1      ISS Sergey Prokopyev
## 2      ISS  Dmitry Petelin
## 3      ISS   Frank Rubio
## 4 Shenzhou 15   Fei Junlong
## 5 Shenzhou 15  Deng Qingming
## 6 Shenzhou 15    Zhang Lu
## 7      ISS   Stephen Bowen
## 8      ISS   Warren Hoburg
## 9      ISS Sultan Alneyadi
## 10     ISS  Andrey Fedyaev
```

Entonces, ahí está nuestra respuesta: al momento de escribir esta publicación de blog, hay 6 personas en el espacio. Pero si lo prueba por sí mismo, es probable que obtenga diferentes nombres y un número diferente. Esa es una de las ventajas de las API: a diferencia de los conjuntos de datos descargables, generalmente se actualizan en tiempo real o casi en tiempo real, por lo que son una excelente manera de obtener acceso a datos muy actualizados.

Un marco de datos, como el que vemos arriba, es cómo normalmente almacenaríamos datos estructurados para su posterior análisis en las bibliotecas de tidyverse que aprendemos en el plan de estudios de Dataquest. (Puede obtener más información sobre esto en la ayuda de datos en R).

Arriba, hemos recorrido un flujo de trabajo de API muy sencillo. La mayoría de las API requerirán que siga este mismo patrón general, pero pueden ser más complejas.

La URL de la API que usamos anteriormente solo requería que hiciéramos una solicitud sin ningún detalle adicional. Algunas API requieren más información del usuario. En la última parte de este tutorial, cubriremos cómo proporcionar información adicional a la API con tu solicitud.

Entonces, ahí está nuestra respuesta: al momento de escribir esta publicación de blog, hay 6 personas en el espacio. Pero si lo prueba por sí mismo, es probable que obtenga diferentes nombres y un número diferente. Esa es una de las ventajas de las API: a diferencia de los conjuntos de datos descargables, generalmente se actualizan en tiempo real o casi en tiempo real, por lo que son una excelente manera de obtener acceso a datos muy actualizados.

API y parámetros de consulta

¿Qué pasaría si quisiéramos saber cuándo la ISS va a pasar sobre un lugar determinado de la Tierra? A diferencia de la API People in Space, la API ISS Pass Times de Open Notify requiere que proporcionemos parámetros adicionales antes de que pueda devolver los datos que queremos.

Específicamente, necesitaremos especificar la latitud y la longitud de la ubicación sobre la que preguntamos como parte de nuestra solicitud **GET()**. Una vez que se especifican la latitud y la longitud, se combinan con la URL original como parámetros de consulta.

Usemos esta API para averiguar cuándo pasará la ISS sobre la Universidad Nacional de Cuyo en Mendoza (que se encuentra aproximadamente en la Latitud -32.87, Longitud -68.87):

```
# http://api.open-notify.org/  
res = GET("http://api.open-notify.org/iss-pass.json", query = list(lat = -32, lon =  
-68))  
res
```

```
## Response [http://api.open-notify.org/iss-pass.json?lat=-32&lon=-68]  
##   Date: 2023-03-29 14:54  
##   Status: 404  
##   Content-Type: text/html  
##   Size: 169 B  
## <html>  
## <head><title>404 Not Found</title></head>  
## <body bgcolor="white">  
## <center><h1>404 Not Found</h1></center>  
## <hr><center>nginx/1.10.3</center>  
## </body>  
## </html>
```

Entonces, ahí está nuestra respuesta: al momento de escribir esta publicación de blog, hay 6 personas en el espacio. Pero si lo prueba por sí mismo, es probable que obtenga diferentes nombres y un número diferente. Esa es una de las ventajas de las API: a diferencia de los conjuntos de datos descargables, generalmente se actualizan en tiempo real o casi en tiempo real, por lo que son una excelente manera de obtener acceso a datos muy actualizados.

Los diferentes elementos de la lista que pasamos al argumento de consulta en **GET()** tienen el formato correcto en la URL, por lo que no tiene que hacerlo usted mismo.

Deberá consultar la documentación de la API con la que está trabajando para ver si hay algún parámetro de consulta requerido. Aquí está la documentación API para ISS Pass Times Link ISS (<https://open-notify.org/Open-Notify-API/ISS-Pass-Times/>).

La gran mayoría de las API a las que puede querer acceder tendrán documentación que puede (y debe) leer para comprender claramente qué parámetros requiere su solicitud. Un parámetro común requerido por muchas API, aunque no las que estamos usando para este tutorial, es una clave de API o alguna otra forma de autenticación. Consulte la documentación de la API para saber cómo obtener una clave de API y cómo insertarla en su solicitud.

De todos modos, ahora que hemos realizado nuestra solicitud, incluidos los parámetros de ubicación, podemos verificar la respuesta usando las mismas funciones que usamos anteriormente. Extraigamos los datos de la respuesta:

```
# http://api.open-notify.org/  
# res = GET("http://api.open-notify.org/iss-pass.json", query = list(lat = 40.7, lon  
# = -74))  
# data = fromJSON(rawToChar(res$content))  
  
# data$response
```

Lamentablemente esta parte de la API ha sido removida en 2023.

API Removida (<http://open-notify.org/Open-Notify-API/ISS-Pass-Times/>)

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see <http://rmarkdown.rstudio.com> (<http://rmarkdown.rstudio.com>).

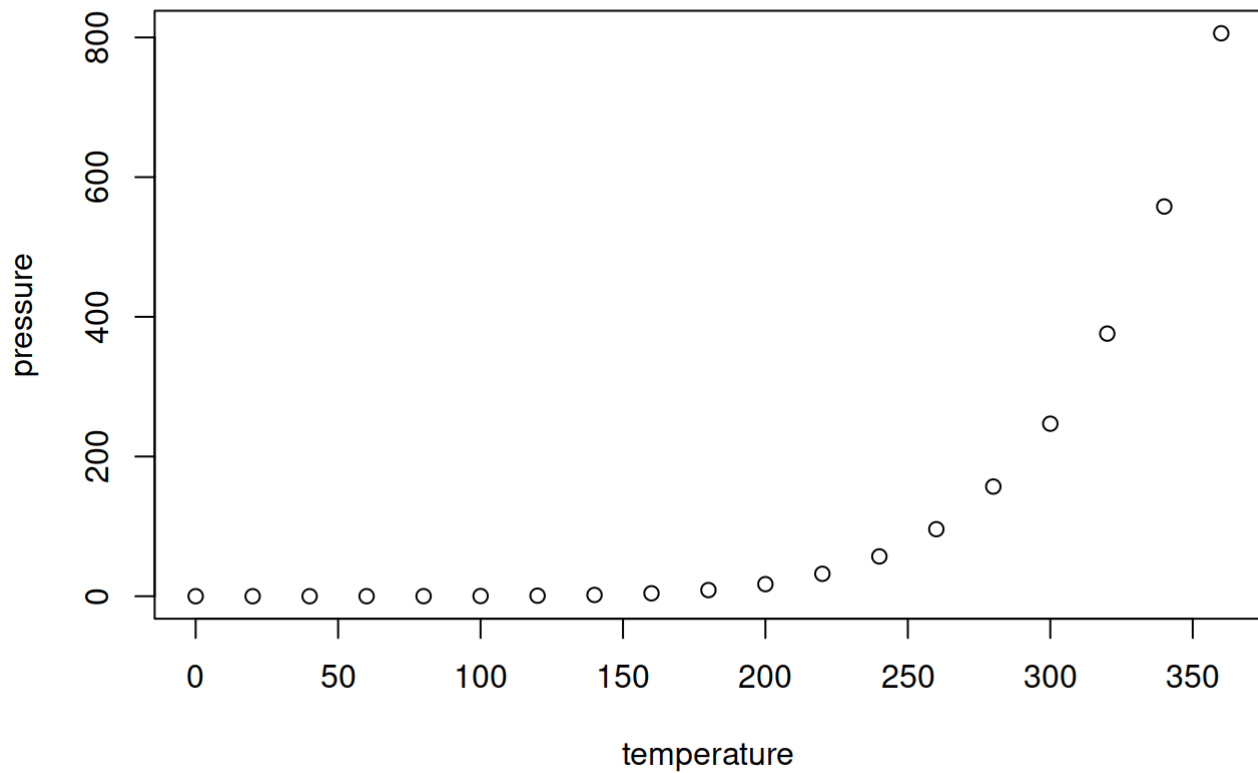
When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```
summary(cars)
```

```
##      speed      dist  
##  Min.   : 4.0    Min.   : 2.00  
##  1st Qu.:12.0    1st Qu.: 26.00  
##  Median :15.0    Median : 36.00  
##  Mean   :15.4    Mean    : 42.98  
##  3rd Qu.:19.0    3rd Qu.: 56.00  
##  Max.   :25.0    Max.    :120.00
```

Including Plots

You can also embed plots, for example:



Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.