Intermediate API in R

Hedia Tnani

# Intermediate API in R

## 1. Basic API in R

Lat time we saw how to make a GET request to retrieve data from an API. We’ll remember again the different steps. We’ll also add query parameters and authentication.

Let’s remember what we did last time using the Ensembl Rest API.

First we prepare the URL.

library(httr) # accessing APIs  
# Preparing the URL   
baseURL <- "https://rest.ensembl.org"  
ext <- "/regulatory/species/homo\_sapiens/microarray?"  
URL <- modify\_url(baseURL, path = ext)

As we saw last time a **Uniform Resource Identifier (URI)** **resource** specifies which resource you want to access. A URI is a generalization of a URL (Uniform Resource Locator)—what you commonly think of as "web addresses".

Then we query the database.

response = GET(URL)  
response

Response [https://rest.ensembl.org/regulatory/species/homo\_sapiens/microarray]  
 Date: 2023-03-16 14:38  
 Status: 200  
 Content-Type: application/json  
 Size: 3.46 kB

Then we convert the JSON format to a dataframe by using the fromJSON function.

library(jsonlite)  
fromJSON(content(response,as = "text", encoding = "UTF-8" ))

description type format vendor  
1 <NA> OLIGO EXPRESSION PHALANX  
2 <NA> OLIGO EXPRESSION CODELINK  
3 <NA> OLIGO EXPRESSION ILLUMINA  
4 <NA> OLIGO EXPRESSION AGILENT  
5 <NA> OLIGO METHYLATION ILLUMINA  
6 <NA> OLIGO EXPRESSION ILLUMINA  
7 <NA> OLIGO EXPRESSION AGILENT  
8 <NA> OLIGO CGH AGILENT  
9 <NA> OLIGO EXPRESSION AGILENT  
10 <NA> OLIGO EXPRESSION AFFY  
11 <NA> OLIGO EXPRESSION AGILENT  
12 <NA> OLIGO EXPRESSION AFFY  
13 <NA> OLIGO EXPRESSION AGILENT  
14 <NA> OLIGO EXPRESSION AGILENT  
15 <NA> OLIGO EXPRESSION AGILENT  
16 <NA> OLIGO EXPRESSION AFFY  
17 <NA> OLIGO EXPRESSION AGILENT  
18 <NA> OLIGO METHYLATION ILLUMINA  
19 <NA> OLIGO EXPRESSION AFFY  
20 Human Transcriptome Array 2.0 OLIGO EXPRESSION AFFY  
21 <NA> OLIGO EXPRESSION AFFY  
22 <NA> OLIGO EXPRESSION AFFY  
23 <NA> OLIGO EXPRESSION AFFY  
24 <NA> OLIGO EXPRESSION AFFY  
25 <NA> OLIGO EXPRESSION AFFY  
26 <NA> OLIGO EXPRESSION AFFY  
27 <NA> OLIGO EXPRESSION AFFY  
28 <NA> OLIGO EXPRESSION AFFY  
29 <NA> OLIGO EXPRESSION AFFY  
30 <NA> OLIGO EXPRESSION AFFY  
31 <NA> OLIGO EXPRESSION AFFY  
32 <NA> OLIGO EXPRESSION AFFY  
33 <NA> OLIGO EXPRESSION AFFY  
34 <NA> OLIGO EXPRESSION AFFY  
35 <NA> OLIGO EXPRESSION AFFY  
 array  
1 OneArray  
2 CODELINK  
3 HumanWG\_6\_V3  
4 GPL6848  
5 HumanMethylation450  
6 HumanRef-8\_V3  
7 WholeGenome  
8 CGH\_44b  
9 SurePrint\_G3\_GE\_8x60k  
10 HC-G110  
11 SurePrint\_G3\_GE\_8x60k\_v2  
12 HG-Focus  
13 WholeGenome\_4x44k\_v1  
14 WholeGenome\_4x44k\_v2  
15 GPL26966  
16 HG-U133A\_2  
17 GPL19072  
18 HumanMethylation27  
19 HG-U133B  
20 HTA-2\_0  
21 HG-U133\_Plus\_2  
22 HG\_U95Av2  
23 HG-U95B  
24 HG-U95C  
25 HG-U95D  
26 HG-U95E  
27 HG\_U95A  
28 HuGeneFL  
29 U133\_X3P  
30 PrimeView  
31 HT\_HG-U133\_Plus\_PM  
32 HuEx-1\_0-st-v2  
33 HuGene-2\_0-st-v1  
34 HuGene-1\_0-st-v1  
35 HuGene-2\_1-st-v1

Then we saw how to make a function.

library(httr)  
library(jsonlite)  
  
get\_microarrays <- function(baseURL, ext, content\_type){  
  
 r <- GET(paste(baseURL, ext, sep = ""), accept(content\_type))  
  
 stop\_for\_status(r)  
  
 if (content\_type == 'application/json'){  
 return (fromJSON(content(r, "text", encoding = "UTF-8")))  
 } else {  
 return (content(r, "text", encoding = "UTF-8"))  
 }  
}  
  
baseURL <- "https://rest.ensembl.org"  
ext <- "/regulatory/species/homo\_sapiens/microarray?"  
con <- "application/json"  
  
  
get\_microarrays(baseURL, ext, con)

format vendor type description  
1 EXPRESSION PHALANX OLIGO <NA>  
2 EXPRESSION CODELINK OLIGO <NA>  
3 EXPRESSION ILLUMINA OLIGO <NA>  
4 EXPRESSION AGILENT OLIGO <NA>  
5 METHYLATION ILLUMINA OLIGO <NA>  
6 EXPRESSION ILLUMINA OLIGO <NA>  
7 EXPRESSION AGILENT OLIGO <NA>  
8 CGH AGILENT OLIGO <NA>  
9 EXPRESSION AGILENT OLIGO <NA>  
10 EXPRESSION AFFY OLIGO <NA>  
11 EXPRESSION AGILENT OLIGO <NA>  
12 EXPRESSION AFFY OLIGO <NA>  
13 EXPRESSION AGILENT OLIGO <NA>  
14 EXPRESSION AGILENT OLIGO <NA>  
15 EXPRESSION AGILENT OLIGO <NA>  
16 EXPRESSION AFFY OLIGO <NA>  
17 EXPRESSION AGILENT OLIGO <NA>  
18 METHYLATION ILLUMINA OLIGO <NA>  
19 EXPRESSION AFFY OLIGO <NA>  
20 EXPRESSION AFFY OLIGO Human Transcriptome Array 2.0  
21 EXPRESSION AFFY OLIGO <NA>  
22 EXPRESSION AFFY OLIGO <NA>  
23 EXPRESSION AFFY OLIGO <NA>  
24 EXPRESSION AFFY OLIGO <NA>  
25 EXPRESSION AFFY OLIGO <NA>  
26 EXPRESSION AFFY OLIGO <NA>  
27 EXPRESSION AFFY OLIGO <NA>  
28 EXPRESSION AFFY OLIGO <NA>  
29 EXPRESSION AFFY OLIGO <NA>  
30 EXPRESSION AFFY OLIGO <NA>  
31 EXPRESSION AFFY OLIGO <NA>  
32 EXPRESSION AFFY OLIGO <NA>  
33 EXPRESSION AFFY OLIGO <NA>  
34 EXPRESSION AFFY OLIGO <NA>  
35 EXPRESSION AFFY OLIGO <NA>  
 array  
1 OneArray  
2 CODELINK  
3 HumanWG\_6\_V3  
4 GPL6848  
5 HumanMethylation450  
6 HumanRef-8\_V3  
7 WholeGenome  
8 CGH\_44b  
9 SurePrint\_G3\_GE\_8x60k  
10 HC-G110  
11 SurePrint\_G3\_GE\_8x60k\_v2  
12 HG-Focus  
13 WholeGenome\_4x44k\_v1  
14 WholeGenome\_4x44k\_v2  
15 GPL26966  
16 HG-U133A\_2  
17 GPL19072  
18 HumanMethylation27  
19 HG-U133B  
20 HTA-2\_0  
21 HG-U133\_Plus\_2  
22 HG\_U95Av2  
23 HG-U95B  
24 HG-U95C  
25 HG-U95D  
26 HG-U95E  
27 HG\_U95A  
28 HuGeneFL  
29 U133\_X3P  
30 PrimeView  
31 HT\_HG-U133\_Plus\_PM  
32 HuEx-1\_0-st-v2  
33 HuGene-2\_0-st-v1  
34 HuGene-1\_0-st-v1  
35 HuGene-2\_1-st-v1

## 2. Intermediate API in R

### **2.1. Adding Query Parameters**

What if we need to add query parameters ?

Query parameters are the extra **parameters** (arguments) about what resource to access.

We can pass query parameters as a named list, where the names are the API fields for each query parameter.

We can use the query parameter of the **GET() function** from the httr package to add the query parameters to our request.

So let’s edit the previous get\_microarrays() function.

* Add a **new parameter** named queries to the function and assign list() to it to make it an optional parameter: queries = list(). We assign a list() here so that multiple optional parameters can be passed to our function.
* Edit the GET() function to set its query parameter to our function parameter queries.

Let’s take again the same example using the Ensembl rest API.

library(httr)  
library(jsonlite)  
  
get\_microarrays <- function(baseURL, endpoint, queries){  
  
 r <- GET(paste(baseURL, endpoint, sep = ""), queries)  
  
 stop\_for\_status(r)  
  
 return (content(r, "text", encoding = "UTF-8"))  
 }  
  
baseURL <- "https://rest.ensembl.org"  
endpoint <- "/regulatory/species/homo\_sapiens/microarray?"  
queries = list(format="json")  
  
  
get\_microarrays(baseURL, endpoint, queries)

[1] "[{\"array\":\"OneArray\",\"description\":null,\"type\":\"OLIGO\",\"format\":\"EXPRESSION\",\"vendor\":\"PHALANX\"},{\"type\":\"OLIGO\",\"description\":null,\"array\":\"CODELINK\",\"vendor\":\"CODELINK\",\"format\":\"EXPRESSION\"},{\"description\":null,\"array\":\"HumanWG\_6\_V3\",\"type\":\"OLIGO\",\"format\":\"EXPRESSION\",\"vendor\":\"ILLUMINA\"},{\"format\":\"EXPRESSION\",\"vendor\":\"AGILENT\",\"type\":\"OLIGO\",\"array\":\"GPL6848\",\"description\":null},{\"format\":\"METHYLATION\",\"vendor\":\"ILLUMINA\",\"type\":\"OLIGO\",\"description\":null,\"array\":\"HumanMethylation450\"},{\"vendor\":\"ILLUMINA\",\"format\":\"EXPRESSION\",\"description\":null,\"array\":\"HumanRef-8\_V3\",\"type\":\"OLIGO\"},{\"type\":\"OLIGO\",\"array\":\"WholeGenome\",\"description\":null,\"format\":\"EXPRESSION\",\"vendor\":\"AGILENT\"},{\"format\":\"CGH\",\"vendor\":\"AGILENT\",\"description\":null,\"array\":\"CGH\_44b\",\"type\":\"OLIGO\"},{\"vendor\":\"AGILENT\",\"format\":\"EXPRESSION\",\"type\":\"OLIGO\",\"description\":null,\"array\":\"SurePrint\_G3\_GE\_8x60k\"},{\"format\":\"EXPRESSION\",\"vendor\":\"AFFY\",\"description\":null,\"array\":\"HC-G110\",\"type\":\"OLIGO\"},{\"description\":null,\"array\":\"SurePrint\_G3\_GE\_8x60k\_v2\",\"type\":\"OLIGO\",\"format\":\"EXPRESSION\",\"vendor\":\"AGILENT\"},{\"description\":null,\"array\":\"HG-Focus\",\"type\":\"OLIGO\",\"vendor\":\"AFFY\",\"format\":\"EXPRESSION\"},{\"type\":\"OLIGO\",\"description\":null,\"array\":\"WholeGenome\_4x44k\_v1\",\"vendor\":\"AGILENT\",\"format\":\"EXPRESSION\"},{\"description\":null,\"array\":\"WholeGenome\_4x44k\_v2\",\"type\":\"OLIGO\",\"vendor\":\"AGILENT\",\"format\":\"EXPRESSION\"},{\"array\":\"GPL26966\",\"description\":null,\"type\":\"OLIGO\",\"vendor\":\"AGILENT\",\"format\":\"EXPRESSION\"},{\"format\":\"EXPRESSION\",\"vendor\":\"AFFY\",\"type\":\"OLIGO\",\"array\":\"HG-U133A\_2\",\"description\":null},{\"format\":\"EXPRESSION\",\"vendor\":\"AGILENT\",\"type\":\"OLIGO\",\"array\":\"GPL19072\",\"description\":null},{\"format\":\"METHYLATION\",\"vendor\":\"ILLUMINA\",\"array\":\"HumanMethylation27\",\"description\":null,\"type\":\"OLIGO\"},{\"format\":\"EXPRESSION\",\"vendor\":\"AFFY\",\"type\":\"OLIGO\",\"description\":null,\"array\":\"HG-U133B\"},{\"format\":\"EXPRESSION\",\"vendor\":\"AFFY\",\"type\":\"OLIGO\",\"description\":\"Human Transcriptome Array 2.0\",\"array\":\"HTA-2\_0\"},{\"format\":\"EXPRESSION\",\"vendor\":\"AFFY\",\"array\":\"HG-U133\_Plus\_2\",\"description\":null,\"type\":\"OLIGO\"},{\"type\":\"OLIGO\",\"array\":\"HG\_U95Av2\",\"description\":null,\"vendor\":\"AFFY\",\"format\":\"EXPRESSION\"},{\"type\":\"OLIGO\",\"array\":\"HG-U95B\",\"description\":null,\"vendor\":\"AFFY\",\"format\":\"EXPRESSION\"},{\"format\":\"EXPRESSION\",\"vendor\":\"AFFY\",\"array\":\"HG-U95C\",\"description\":null,\"type\":\"OLIGO\"},{\"format\":\"EXPRESSION\",\"vendor\":\"AFFY\",\"array\":\"HG-U95D\",\"description\":null,\"type\":\"OLIGO\"},{\"description\":null,\"array\":\"HG-U95E\",\"type\":\"OLIGO\",\"vendor\":\"AFFY\",\"format\":\"EXPRESSION\"},{\"type\":\"OLIGO\",\"array\":\"HG\_U95A\",\"description\":null,\"vendor\":\"AFFY\",\"format\":\"EXPRESSION\"},{\"type\":\"OLIGO\",\"description\":null,\"array\":\"HuGeneFL\",\"vendor\":\"AFFY\",\"format\":\"EXPRESSION\"},{\"array\":\"U133\_X3P\",\"description\":null,\"type\":\"OLIGO\",\"vendor\":\"AFFY\",\"format\":\"EXPRESSION\"},{\"description\":null,\"array\":\"PrimeView\",\"type\":\"OLIGO\",\"vendor\":\"AFFY\",\"format\":\"EXPRESSION\"},{\"array\":\"HT\_HG-U133\_Plus\_PM\",\"description\":null,\"type\":\"OLIGO\",\"format\":\"EXPRESSION\",\"vendor\":\"AFFY\"},{\"vendor\":\"AFFY\",\"format\":\"EXPRESSION\",\"type\":\"OLIGO\",\"description\":null,\"array\":\"HuEx-1\_0-st-v2\"},{\"description\":null,\"array\":\"HuGene-2\_0-st-v1\",\"type\":\"OLIGO\",\"vendor\":\"AFFY\",\"format\":\"EXPRESSION\"},{\"description\":null,\"array\":\"HuGene-1\_0-st-v1\",\"type\":\"OLIGO\",\"vendor\":\"AFFY\",\"format\":\"EXPRESSION\"},{\"type\":\"OLIGO\",\"description\":null,\"array\":\"HuGene-2\_1-st-v1\",\"format\":\"EXPRESSION\",\"vendor\":\"AFFY\"}]"

## 2.2 ChatGPT’s **API Authentification**

To authenticate with the ChatGPT’s API, we need to use an access token. An access token is a credential we can **generate** on the OpenAI’s website. The token is a string that the API can read and associate with our account.

Typically APIs use the following authentication types:

* **Username-password authentication**: requires a username and password. We can use the authenticate("username", "password") function in the API requests to achieve this.

**API key authentication**: requires an API “key” or “token.” We can use the api\_key = "key" parameter request to achieve this.

Access tokens can have scopes and specific permissions. Using read-access-only tokens in potentially insecure or shared scripts improves security.

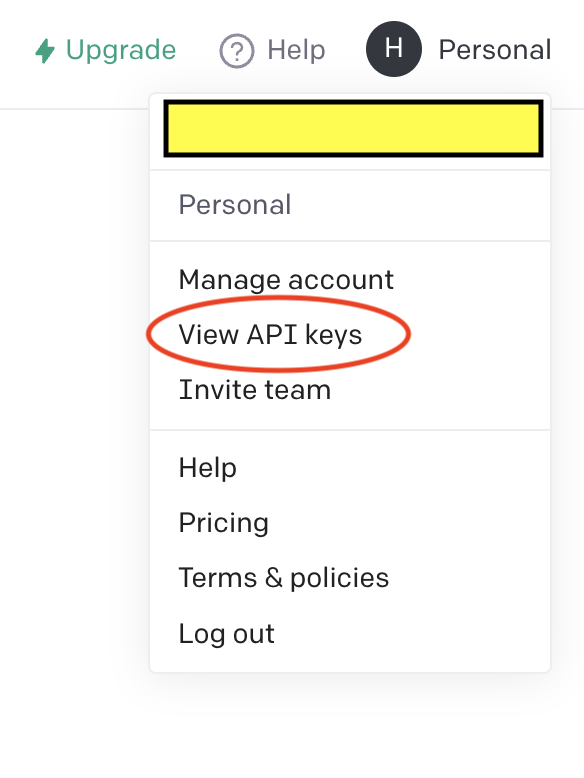
APIs also use authentication for **rate limiting**. Developers typically use APIs to build interesting applications or services. Ensuring an API is available and responsive for all users will prevent us from making too many requests too quickly. We call this **restriction rate limiting**.

### 2.2.1 Create OpenAI API Keys

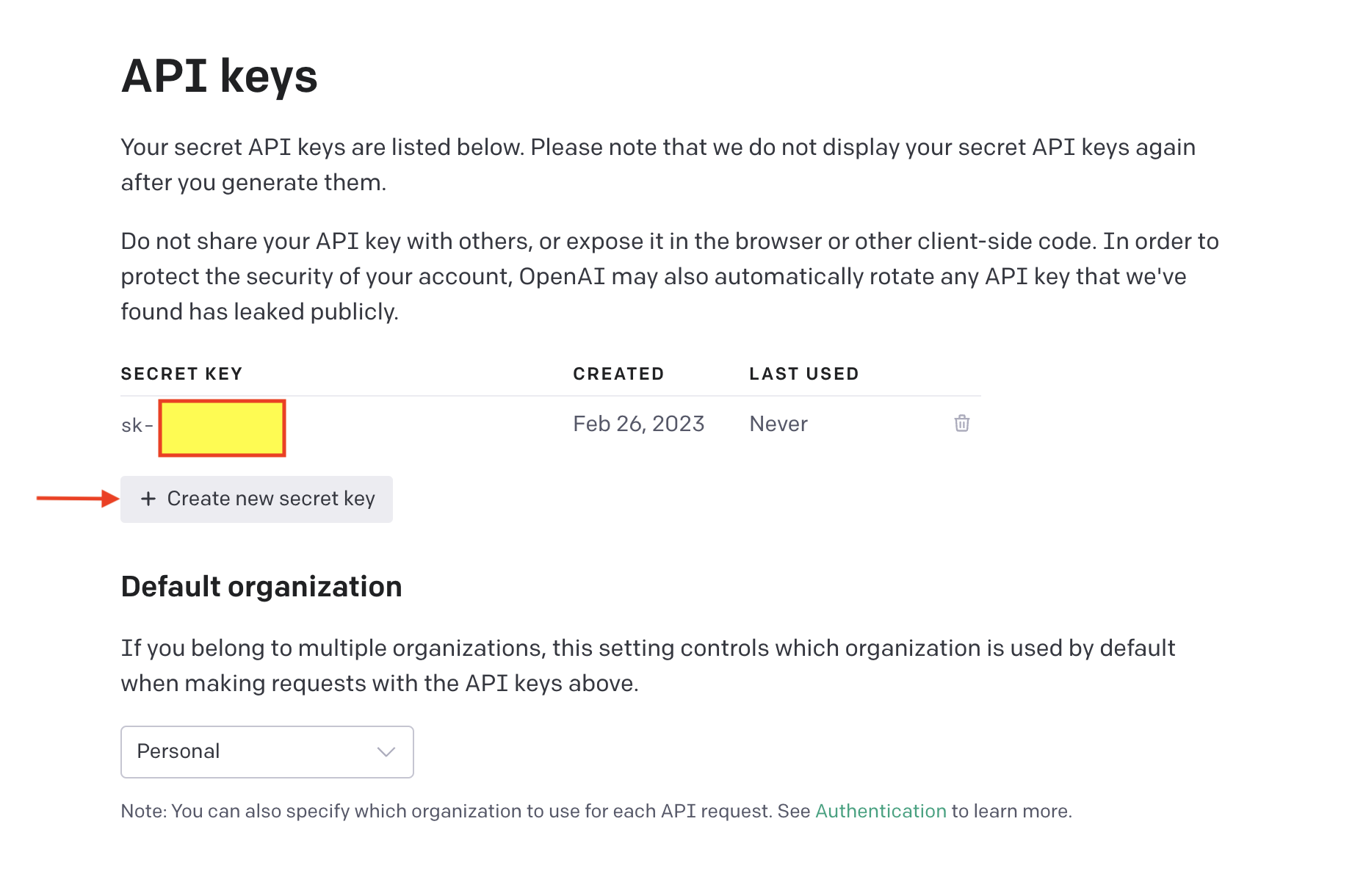
To use OpenAI through API, you must create a free account and generate keys.

1. Sign up here <https://beta.openai.com/signup>. You can use your Google or Microsoft account to sign up if you don’t want to create using an email/password combination. You may need a valid mobile number to verify your account.

2. Now, visit your OpenAI key page <https://beta.openai.com/account/api-keys> or click the menu item “View API keys”



3. Create a new key by clicking on **Personal** and then “Create new secret key” button. See the image below for reference.



Et voila !!!

OpenAI gives $18 free credits which you can consume within the first 3 months.

### 2.2.2 Use OpenAI API Keys

Here we’ll explain the different ways that are usually used to set the API key up in RStudio :

* By default, API calls will look for OPENAI\_API\_KEY environment variable. If you want to set a global environment variable, you can use the following command, where "<APIKEY>" should be replaced with your actual key:

Sys.setenv(OPENAI\_API\_KEY = "<OPENAI\_API\_KEY>")

* Alternatively, you can set the key in your .Renviron file.

The following commands will open .Renviron for editing:

require(usethis)  
edit\_r\_environ()

You can add the following line to the file (again, replace <my\_api\_key> with your actual key):

OPENAI\_API\_KEY=<my\_api\_key>

Make sure your ‘.Renviron’ ends with a newline. Don't forget to save and restart R for this change to take effect. This now set the API key. This is a less secure approach.

* Use a more secure approach by using ask\_pass that if possible uses askpass::askpass() to interactively safely prompt you for the values
* For that we’ll be using this function.
* set\_api\_key <- function(key = NULL) {   
   if (is.null(key)) {   
   key <- askpass::askpass("Please enter your API key") }   
   Sys.setenv("OPENAI\_KEY" = key) }

httr2 provides secret\_encrypt() and secret\_decrypt() to scramble secrets so that you can include them in your public source code without worrying that others can read them. There are three basic steps to this process:

1. You create an **encryption** key with secret\_make\_key() that is used to scramble and descramble secrets using symmetric cryptography.
2. You scramble your secrets with secret\_encrypt() and store the resulting text directly in the source code of your package.
3. When needed, you descramble the secret using secret\_decrypt().

## Hands-on exercice using ChatGPT in R

Let’s apply what we’ve learned so far.

Let’s set up our token.

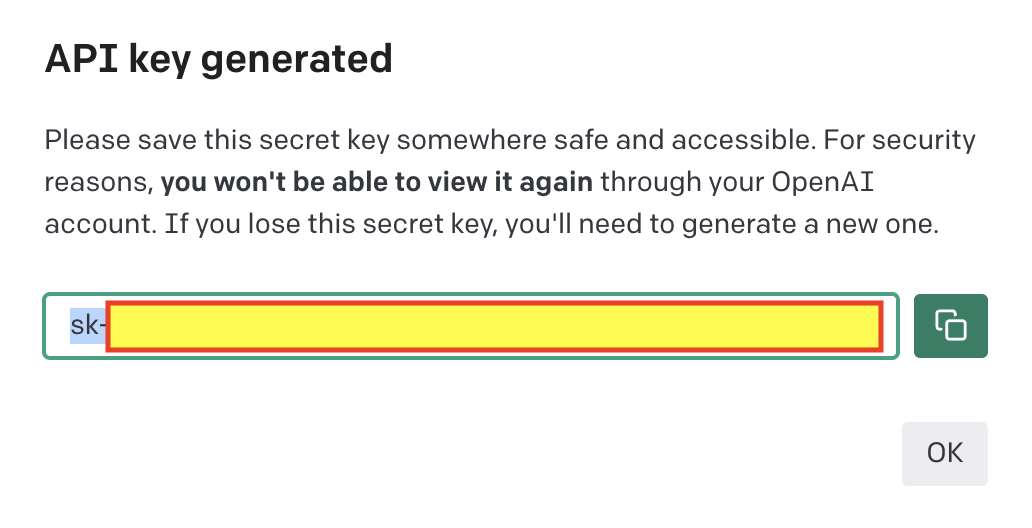
First let’s run the set\_api\_key( )

set\_api\_key <- function(key = NULL) {   
 if (is.null(key)) {   
 key <- askpass::askpass("Please enter your OpenAI API key") }   
 Sys.setenv("OPENAI\_KEY" = key) }

**Sys.setenv** sets environment variables. Then let’s call the function.

set\_api\_key()

You’ll get a prompt asking for the token. For that, go to OpenAI and click on “Create new secret key”.



Copy the API key and paste it in the prompt.

key <- Sys.getenv("OPENAI\_KEY")  
is.null(key)

We can pass our token to the OpenAI API through the query parameter api\_key, just like we did before.

library(httr)   
 key <- Sys.getenv("OPENAI\_KEY")   
 headers = c( `Authorization` = paste('Bearer', key), `Content-Type` = 'application/json' )   
res <- httr::GET(url = 'https://api.openai.com/v1/models/text-davinci-003', httr::add\_headers(.headers=headers))   
 res

As we saw last time the status code is a three digit number that summarises whether or not the request was successful.

http\_status(res)

GPT-3.5 models can understand and generate natural language or code. text-davinci-003 is Can do any language task with better quality, longer output, and consistent instruction-following than the curie, babbage, or ada models. Also supports inserting completions within text. For more information about the different models go to this [link](https://platform.openai.com/docs/models/gpt-3-5).

You can change the model. The most capable and cost effective model in the GPT-3.5 family is **gpt-3.5-turbo** which has been optimized for chat but works well for traditional completions tasks as well.

library(httr)   
 key <- Sys.getenv("OPENAI\_KEY")   
 headers = c( `Authorization` = paste('Bearer', key), `Content-Type` = 'application/json' )   
res <- httr::GET(url = 'https://api.openai.com/v1/models/gpt-3.5-turbo', httr::add\_headers(.headers=headers))   
 res

### POST requests

We use POST requests to send information (instead of retrieving it) and create objects on the API’s server.

When POST()ing, you can include data in the body of the request. httr allows you to supply this in a number of different ways. The most common way is a named list. If the body is a named list you should specify how it will be encoded.

# JSON encoded  
r <- POST(url, body = body, encode = "json")

Let’s say you want to optimize your R code.

code = "i <- 10\nwhile (i > 0) {\n i <- i - 1\n print(i)\n}"   
cat(code,"\n")

In the prompt you have to ask

prompt <- paste0('Optimize the following R code: "', code, '"')   
prompt

You have to give the body of the POST request.

params <- list(  
model = "text-davinci-003", max\_tokens = 256, temperature =0.7, top\_p = 1, frequency\_penalty = 0, presence\_penalty = 0)

Let’s run our POST request

r = content(POST( "https://api.openai.com/v1/completions", add\_headers("Authorization" = paste('Bearer', key)), content\_type\_json(), body = toJSON(c(params, list(prompt = prompt)), auto\_unbox = TRUE) ))  
cat(r$choices[[1]]$text,"\n")

## Exercice

Use the example above to do the same task as above using httr2 .

### ChatGPT R packages

https://github.com/jcrodriguez1989/chatgpt

https://jameshwade.github.io/gpttools/

https://irudnyts.github.io/openai/

## References

https://info201.github.io/apis.html

https://observablehq.com/@periscopic/cozy-collecting-part-1

https://observablehq.com/@periscopic/cozy-collecting-part-2

https://observablehq.com/@periscopic/cozy-collecting-case-study