DATA MINING

Master Meci - Parcours Data - Options PISE et CCESE

Claude Grasland, Professeur de Géographie, Université de Paris (Diderot)

2021-02-23

Table des Matières

Présentation **3**

À propos de ce document . . . 3
Prérequis . . . 3
Remerciements . . . 3
Licence . . . 3

1 Collecter des données à l'aide d'une API **5**

1.1 Qu'est-ce qu'une API ? . . . 6
1.2 Comment utiliser une API dans R ? . . . 6
1.3 API ou data packages ? . . . 12
1.4 Exercices . . . 16

2 Collecter des données à l'aide d'une API **19**

2.1 Objectifs . . . 19
2.2 Le tableau "countries" . . . 20
2.3 Le tableau indicators . . . 21
2.4 L'extraction des données . . . 23
2.5 Exercices . . . 27
2.6 Exercice 3 . . . 27

Présentation

À propos de ce document

Ce document est la première version du cours de Data Mining dispensé aux étudiants de deuxième année de l' option Data du master MECI

Il est basé sur R version 4.0.2 (2020-06-22).

Ce document est régulièrement corrigé et mis à jour. La version de référence est disponible en ligne à l'adresse :

- https://ClaudeGrasland.github.io/DataMining1.

Pour toute suggestion ou correction, il est possible de me contacter par mail

Prérequis

Le seul prérequis pour suivre ce document est d'avoir installé R et RStudio sur votre ordinateur. Il s'agit de deux logiciels libres, gratuits, téléchargeables en ligne et fonctionnant sous PC, Mac et Linux.

Pour installer R, il suffit de se rendre sur une des pages suivantes [1] :

- Installer R sous Windows
- Installer R sous Mac

Pour installer RStudio, rendez-vous sur la page suivante et téléchargez la version adaptée à votre système :

- https://www.rstudio.com/products/rstudio/download/#download

Remerciements

Ce document a bénéficié de la relecture et des suggestions ... des étudiants qui en ont été les cobayes des premières versions.

Ce document est généré par l'excellente extension bookdown de Yihui Xie et il s'est servi du template proposé par Julien Barnier pour introduire des exercices interactifs dans son cours de tidyverse.

Licence

Figure 1: Licence Creative Commons

[1] Sous Linux, utilisez votre gestionnaire de packages habituel.

Chapitre 1

Collecter des données à l'aide d'une API

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

```
Attachement du package : 'jsonlite'
```

```
The following object is masked from 'package:purrr':

    flatten
```

```
library(insee)
```

```
Warning: le package 'insee' a été compilé avec la version R 4.0.3
```

```
library(dplyr)
library(lubridate)
```

```
Warning: le package 'lubridate' a été compilé avec la version R 4.0.3
```

```
Attachement du package : 'lubridate'
```

```
The following objects are masked from 'package:base':

    date, intersect, setdiff, union
```

1.1 Qu'est-ce qu'une API ?

1.1.1 Définitions

On peut partir de la définition suivante

> En informatique, API est l'acronyme d'*Application Programming Interface*, que l'on traduit en français par **interface de programmation applicative** ou **interface de programmation d'application**. L'API peut être résumée à une solution informatique qui permet à des applications de communiquer entre elles et de s'échanger mutuellement des services ou des données. Il s'agit en réalité d'un ensemble de fonctions qui facilitent, via un langage de programmation, l'accès aux services d'une application. (Source : Journal du Net)

1.1.2 Domaine d'application

Une API peut remplir des fonctions très diverses :

> Dans le domaine d'internet, l'API permet aux développeurs de pouvoir utiliser un programme sans avoir à se soucier du fonctionnement complexe d'une application. Les API peuvent par exemple être utilisées pour déclencher des campagnes publicitaires d'e-mailing de façon automatique sans avoir à passer par la compréhension d'une telle application (c'est le cas avec l'API AdWords de Google, par exemple). On les retrouve aujourd'hui dans de nombreux logiciels, en particulier dans les systèmes d'exploitation, les serveurs d'applications, dans le monde du graphisme (OpenGL), dans les applications SaaS (Office 365, G Suite, Salesforce…), les bases de données, l'open data, etc.(Source : Journal du Net)

1.1.3 Système client-serveur

D'une manière générale, les API supposent un échange d'informations entre un *client* et un *serveur*.

> Ces échanges d'informations suivent un *protocole* c'est-à-dire un ensemble de règles. Il existe deux grands protocoles de communication sur lesquels s'adossent les API : Simple Object Access Protocol (SOAP) et Representational State Transfer (REST). Le second s'est désormais largement imposé face au premier car il est plus flexible. Il a donné naissance aux API dites REST ou RESTful (Source : Journal du Net)

1.2 Comment utiliser une API dans R ?

Le métier de data analyst implique presque nécessairement l'emploi d'API. Les langages de programmation R ou Python ont donc l'un comme l'autre mis au point des packages pour faciliter l'envoi de requêtes sur des serveurs dotés d'API. A titre d'introduction, nous allons reprendre (et traduire en français) quelques extraits d'un billet proposé par un étudiant en doctorat de biostatistiques à l'université de Californie San Diego.

- Pascual C., 2020, Getting Started with APIs in R

1.2.1 Pourquoi utiliser des API ?

> «API» est un terme général désignant le lieu où un programme informatique interagit avec un autre ou avec lui-même. Dans ce didacticiel, nous travaillerons spécifiquement avec des API Web, où deux ordinateurs différents - un client et un serveur - interagiront l'un avec l'autre pour demander et fournir des données, respectivement.
>
> Les API offrent aux scientifiques des données un moyen raffiné de demander des données propres et organisées à partir d'un site Web. Lorsqu'un site Web comme Facebook met en place une API, il met essentiellement en place un ordinateur qui attend les demandes de données.
>
> Une fois que cet ordinateur reçoit une demande de données, il effectuera son propre traitement des données et les enverra à l'ordinateur qui l'a demandé. De notre point de vue en tant que demandeur, nous devrons écrire du code dans R qui crée la demande et indique à l'ordinateur exécutant l'API ce dont nous avons besoin. Cet ordinateur lira ensuite notre code, traitera la requête et renverra des données bien formatées qui peuvent être facilement analysées par les bibliothèques R existantes.
>
> Pourquoi est-ce précieux? Comparez l'approche API au scraping Web pur. Lorsqu'un programmeur gratte une page Web, il reçoit les données dans un morceau de HTML désordonné. Bien qu'il existe certainement des bibliothèques qui facilitent l'analyse du texte HTML, ce sont toutes des étapes de nettoyage qui doivent être prises avant même de mettre la main sur les données que nous voulons!
>
> Souvent, nous pouvons immédiatement utiliser les données que nous obtenons d'une API, ce qui nous fait gagner du temps et de la frustration.

Source : Traduction française d'un billet de Pascual C., 2020

1.2.2 Installer les packages jsonlite et httr

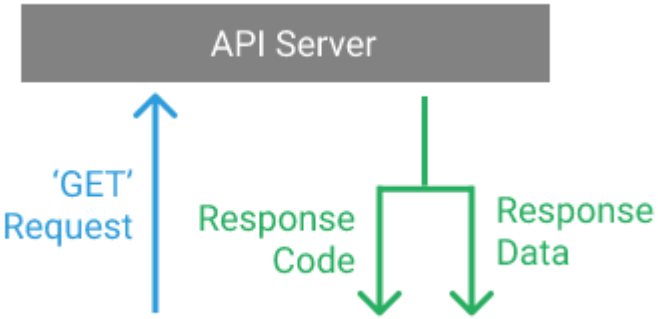
Pour travailler avec des API dans R, nous devons intégrer certaines bibliothèques (*library*). Ces bibliothèques prennent toutes les complexités d'une requête d'API et les enveloppent dans des fonctions que nous pouvons utiliser dans des lignes de code uniques. Les bibliothèques R que nous utiliserons sont `httr` et `jsonlite`. Elles remplissent des rôles différents dans notre introduction des API, mais les deux sont essentiels.Si vous ne disposez pas de ces bibliothèques dans votre console R ou RStudio, vous devez d'abord les télécharger.

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

1.2.3 Structure d'une requête

Une requête adressé à une API va suivre le schéma suivant :

```
knitr::include_graphics("img/API_GET.png",)
```



Il existe plusieurs types de requêtes que l'on peut adresser à un serveur API. Pour nos besoins, nous allons simplement demander des données, ce qui correspond à une demande **GET**. Les autres types de requêtes sont POST et PUT, mais nous n'avons pas à nous en préoccuper dans l'immédiat

Afin de créer une requête GET, nous devons utiliser la fonction *GET()* de la bibliothèque `httr`. La fonction GET() nécessite une URL, qui spécifie l'adresse du serveur auquel la demande doit être envoyée. A titre d'exemple, C. Pascual propose de travailler avec l'**API Open Notify**, qui donne accès à des données sur divers projets de la NASA. À l'aide de l'API Open Notify, nous pouvons notamment en savoir plus sur l'emplacement de la Station spatiale internationale et sur le nombre de personnes actuellement dans l'espace.

Notre programme télécharge les données disponibles à l'adresse du serveur et les stocke dans un objet auquel on peut donner le nom que l'on souhaite, par exemple *toto*

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

```
Response [http://api.open-notify.org/astros.json]
  Date: 2021-02-23 12:23
  Status: 200
  Content-Type: application/json
  Size: 356 B
```

Lorsqu'on affiche la réponse, on obtient ici quatre informations :

- **Date** : le moment exact du téléchargement, très utile pour suivre les mises à jour
- **Status** : le code informatique de résultat de la requête. La valeur *200* indique un succès alors que les autres valeurs signaleront un problème.
- **Content-Type** : le type d'information recueillie. Ici, une application au format json
- **Size** : la taille du fichier résultant du transfert.

On pourrait également en savoir plus en tapant la commande *str()* qui nous indique que le résultat est une liste comportant 10 branches et de nombreuses sous-branches :

```
str(toto)
```

```
List of 10
 $ url        : chr "http://api.open-notify.org/astros.json"
 $ status_code: int 200
 $ headers    :List of 6
  ..$ server                      : chr "nginx/1.10.3"
  ..$ date                        : chr "Tue, 23 Feb 2021 12:23:23 GMT"
  ..$ content-type                : chr "application/json"
  ..$ content-length              : chr "356"
  ..$ connection                  : chr "keep-alive"
  ..$ access-control-allow-origin: chr "*"
  ..- attr(*, "class")= chr [1:2] "insensitive" "list"
 $ all_headers:List of 1
  ..$ :List of 3
  .. ..$ status : int 200
  .. ..$ version: chr "HTTP/1.1"
  .. ..$ headers:List of 6
  .. .. ..$ server                      : chr "nginx/1.10.3"
```

```
  .. .. ..$ date                        : chr "Tue, 23 Feb 2021 12:23:23 GMT"
  .. .. ..$ content-type                : chr "application/json"
  .. .. ..$ content-length              : chr "356"
  .. .. ..$ connection                  : chr "keep-alive"
  .. .. ..$ access-control-allow-origin: chr "*"
  .. .. ..- attr(*, "class")= chr [1:2] "insensitive" "list"
 $ cookies    :'data.frame':    0 obs. of  7 variables:
  ..$ domain    : logi(0)
  ..$ flag      : logi(0)
  ..$ path      : logi(0)
  ..$ secure    : logi(0)
  ..$ expiration: 'POSIXct' num(0)
  ..$ name      : logi(0)
  ..$ value     : logi(0)
 $ content    : raw [1:356] 7b 22 6d 65 ...
 $ date       : POSIXct[1:1], format: "2021-02-23 12:23:23"
 $ times      : Named num [1:6] 0 0.00283 0.15962 0.15975 0.30635 ...
  ..- attr(*, "names")= chr [1:6] "redirect" "namelookup" "connect" "pretransfer" ...
 $ request    :List of 7
  ..$ method    : chr "GET"
  ..$ url       : chr "http://api.open-notify.org/astros.json"
  ..$ headers   : Named chr "application/json, text/xml, application/xml, */*"
  .. ..- attr(*, "names")= chr "Accept"
  ..$ fields    : NULL
  ..$ options   :List of 2
  .. ..$ useragent: chr "libcurl/7.64.1 r-curl/4.3 httr/1.4.2"
  .. ..$ httpget  : logi TRUE
  ..$ auth_token: NULL
  ..$ output    : list()
  .. ..- attr(*, "class")= chr [1:2] "write_memory" "write_function"
  ..- attr(*, "class")= chr "request"
 $ handle     :Class 'curl_handle' <externalptr>
 - attr(*, "class")= chr "response"
```

La branche qui nous intéresse le plus est *content* puisque c'est celle qui contient les données.

1.2.4 Extraction des données

Les données contenues dans la réponse ont été stockées au format *JSON (JavaScript Object Notation)* qui est devenu un standard pour les échanges de données. Sans entrer dans le détail de ce langage, on retiendra qu'il va falloir convertir les données JSON dans un format de tableau lisible par R ce qui se fait ici en deux étapes.

Tout d'abord extraire le champ content et le convertir en mode caractère :

```
# conversion du contenu de toto en mode character
toto2<-rawToChar(toto$content)
toto2
```

```
[1] "{\"message\": \"success\", \"number\": 7, \"people\": [{\"craft\": \"ISS\", \"name\": \"Sergey Ryz
```

```
str(toto2)
```

Table 1.1: Passagers de l'ISS en temps réel

craft	name
ISS	Sergey Ryzhikov
ISS	Kate Rubins
ISS	Sergey Kud-Sverchkov
ISS	Mike Hopkins
ISS	Victor Glover
ISS	Shannon Walker
ISS	Soichi Noguchi

```
 chr "{\"message\": \"success\", \"number\": 7, \"people\": [{\"craft\": \"ISS\", \"name\": \"Sergey Ry
```

Puis convertir ces données de type JSON en données utilisables par R à l'aide de la fonction *fromJson()* du package `jsonlite()`

```
toto3 <- fromJSON(toto2)
str(toto3)
```

```
List of 3
 $ message: chr "success"
 $ number : int 7
 $ people :'data.frame':    7 obs. of  2 variables:
  ..$ craft: chr [1:7] "ISS" "ISS" "ISS" "ISS" ...
  ..$ name : chr [1:7] "Sergey Ryzhikov" "Kate Rubins" "Sergey Kud-Sverchkov" "Mike Hopkins" ...
```

On obtient finalement une liste de trois éléments dont le dernier est un *data.frame* décrivant les astronautes présents dans la station spatiale internationale au moment de l'execution du programme.

```
toto4<-toto3$people
str(toto4)
```

```
'data.frame':   7 obs. of  2 variables:
 $ craft: chr  "ISS" "ISS" "ISS" "ISS" ...
 $ name : chr  "Sergey Ryzhikov" "Kate Rubins" "Sergey Kud-Sverchkov" "Mike Hopkins" ...
```

```
kable(toto4,caption = "Passagers de l'ISS en temps réel")
```

1.2.5 API et mise à jour en temps réel

Sur le site web du billet proposé par C. Pascual en février 2020, on trouve une autre liste ne comportant que 6 passagers et avec des noms totalement différents :

Table 1.2: Passagers de l'ISS en février 2020

craft	name
ISS	Christina Koch
ISS	Alexander Skvortsov
ISS	Luca Parmitano
ISS	Andrew Morgan
ISS	Oleg Skripochka
ISS	Jessica Meir

En effet, l'API renvoie les résultats au moment de l'execution de la fonction *GET()* ce qui correspond à février 2020 pour le billet de blog. Or, les astronautes sont remplacés au plus tous les six mois ce qui explique que tous les noms soient différents un an après.

NB : Cet exemple permet de mettre en évidence une fonction centrale des API qui est la mise à jour en temps réel des données !

1.2.6 API et requête paramétrique

L'exemple précédent consistait à télécharger la totalité d'un tableau et ne demandait donc pas de paramètres particuliers. Mais il peut aussi arriver (par exemple si une base de données est très volumineuse) que l'on précise à l'aide de paramètres ce que l'on veut précisément télécharger.

A titre d'exemple, C. Pascual propose d'utiliser une autre API de la NASA intitulée ISS Pass Time qui permet de savoir à quel moment la station ISS passera au dessus d'un certain point du globe.

L'exemple choisi par C.Pascual est la recherche des trois prochaines dates de passage de l'ISS au dessus de New York dont les coordonnées de latitude et de longitude sont 40.7 et -74.0 :

```
titi <- GET("http://api.open-notify.org/iss-pass.json",
            query = list(lat = 40.7, lon = -74, n=3))
titi2 <- fromJSON(rawToChar(titi$content))
titi3 <- titi2$response
titi3
```

```
  duration   risetime
1      560 1614085817
2      622 1614091649
3      648 1614097455
```

Le résultat paraît à première vue assez déconcertant. Mais la lecture de la documentation de l'API indique que les deux variables du tableau correspondent respectivement :

- *duration* : nombre de secondes pendant lesquelles la station sera à la verticale du point avec un angle de + ou - 10 degrés.
- *risetime* : moment de passage exprimé en Unix Time c'est-à-dire en nombre de secondes écoulées depuis le 1er Janvier 1970 UTC.

Si l'on veut se ramener à une date précise, il faut donc convertir ce temps à l'aide d'une fonction R. Le plus simple est pour cela d'utiliser la fonction *as_datetime()* du package `lubridate`.

```
library(lubridate)
titi3$risetime<-as_datetime(titi3$risetime)
kable(titi3)
```

duration	risetime
560	2021-02-23 13:10:17
622	2021-02-23 14:47:29
648	2021-02-23 16:24:15

1.3 API ou data packages ?

L'utilisation d'API à l'aide des fonctions de base `httpr` et `jsonlite` constitue à moyen terme une étape indispensable de la formation d'un data analyste. Mais heureusement elle n'est pas toujours indispensable pour le débutant car plusieurs packages R (ou Python) ont été développées par des programmeurs pour faciliter l'usage des API.

Ces packages executent en pratique les commandes de l'API, mais sans que l'utilisateur ait besoin d'avoir aucune connaissance sur la syntaxe de la fonction *GET()* qui a collecté les données ni des transformations effectuées sur les résultats pour transformer les données JSON en *data.frame* ou *tibble*. La connaissance de ces packages spécialisées offre donc une grosse économie de temps … s'ils ont été bien conçus.

On va prendre comme exemple le package `insee` mis au point récemment pour faciliter l'accès aux données de cette organisation. La documentation du package est accessible par le lien ci-dessous

https://www.data.gouv.fr/fr/reuses/insee-package-r/

Cette page renvoie vers une "vignette" c'est-à-dire une suite de programmes exemples.

https://inseefr.github.io/R-Insee-Data/

1.3.1 Installation et chargement du package

On commence par installer le package `insee` ce qui peut prendre quelques minutes mais sera fait une seule fois (sauf mise à jour).

```
# install.packages("insee")
```

On peut ensuite lancer le package pour l'utiliser avec *library()* et on ajoute le package *tidyverse* que l'INSEE semble privilégier pour l'exploitation des données :

```
library(insee)
library(tidyverse,warn.conflicts = F)
```

1.3.2 Chargement de la liste des tableaux

On commence par télécharger le catalogue des tableaux de données disponibles,à l'aide de la commande *get_dataset_list()*

```
catalogue = get_dataset_list()
kable(head(catalogue))
```

id	Name.fr
BALANCE-PAIEMENTS	Balance des paiements
CHOMAGE-TRIM-NATIONAL	Chômage, taux de chômage par sexe et âge (sens BIT)
CLIMAT-AFFAIRES	Indicateurs synthétiques du climat des affaires
CNA-2010-CONSO-MEN	Consommation des ménages - Résultats par produit, fonction et durabilité
CNA-2010-CONSO-SI	Dépenses de consommation finale par secteur institutionnel - Résultats par opération e
CNA-2010-CPEB	Comptes de production et d'exploitation par branche

Chaque tableau comporte un très grand nombre de séries chronologiques parmi lesquelles il faut opérer un choix afin d'extraire exactement ce que l'on veut.

1.3.3 Examen des séries présentes dans un tableau

Une fois que l'on a choisi un tableau, on peut examiner plus en détail les différentes séries qui y sont présentes à l'aide de la commande *get_idbank_list()*. On va par exemple examiner le contenu de la base de données "DECES-MORTALITE" :

```
var<-get_idbank_list("DECES-MORTALITE")
str(var)
```

```
FALSE tibble [1,905 x 39] (S3: tbl_df/tbl/data.frame)
FALSE  $ nomflow              : chr [1:1905] "DECES-MORTALITE" "DECES-MORTALITE" "DECES-MORTALITE" "DEC
FALSE  $ idbank               : chr [1:1905] "000436398" "001641606" "000869058" "001780755" ...
FALSE  $ cleFlow              : chr [1:1905] "M.TAUX_MORTALITE.TAUX.TXMORINF.FM.0.SO.SO.BRUT" "M.TAUX_M
FALSE  $ FREQ                 : chr [1:1905] "M" "M" "A" "A" ...
FALSE  $ INDICATEUR           : chr [1:1905] "TAUX_MORTALITE" "TAUX_MORTALITE" "DECES_DOMICILIES" "DECE
FALSE  $ NATURE               : chr [1:1905] "TAUX" "TAUX" "VALEUR_ABSOLUE" "VALEUR_ABSOLUE" ...
FALSE  $ DEMOGRAPHIE          : chr [1:1905] "TXMORINF" "TXMORINF" "DECES-DOM" "DECES-DOM" ...
FALSE  $ REF_AREA             : chr [1:1905] "FM" "FR-D976" "AU" "F_H_IDF" ...
FALSE  $ SEXE                 : chr [1:1905] "0" "0" "SO" "SO" ...
FALSE  $ AGE                  : chr [1:1905] "SO" "SO" "SO" "SO" ...
FALSE  $ UNIT_MEASURE         : chr [1:1905] "SO" "SO" "NOMBRE" "NOMBRE" ...
FALSE  $ CORRECTION           : chr [1:1905] "BRUT" "BRUT" "BRUT" "BRUT" ...
FALSE  $ FREQ_label_fr        : chr [1:1905] "Mensuelle" "Mensuelle" "Annuelle" "Annuelle" ...
FALSE  $ FREQ_label_en        : chr [1:1905] "Monthly" "Monthly" "Annual" "Annual" ...
FALSE  $ INDICATEUR_label_fr  : chr [1:1905] "Taux de mortalité" "Taux de mortalité" "Décès domiciliés"
FALSE  $ INDICATEUR_label_en  : chr [1:1905] "Mortality rate" "Mortality rate" "Deaths domiciled" "Deat
FALSE  $ NATURE_label_fr      : chr [1:1905] "Taux" "Taux" "Valeur absolue" "Valeur absolue" ...
FALSE  $ NATURE_label_en      : chr [1:1905] "Rate" "Rate" "Absolute value" "Absolute value" ...
FALSE  $ DEMOGRAPHIE_label_fr : chr [1:1905] "Taux de mortalité infantile" "Taux de mortalité infantile
FALSE  $ DEMOGRAPHIE_label_en : chr [1:1905] "Infant mortality rate" "Infant mortality rate" "Deaths of
FALSE  $ REF_AREA_label_fr    : chr [1:1905] "France métropolitaine" "France hors Mayotte" "Territoires
FALSE  $ REF_AREA_label_en    : chr [1:1905] "Metropolitan France" "France excluding Mayotte" "French o
FALSE  $ SEXE_label_fr        : chr [1:1905] "Ensemble" "Ensemble" "Sans objet" "Sans objet" ...
FALSE  $ SEXE_label_en        : chr [1:1905] "All" "All" "Not applicable" "Not applicable" ...
FALSE  $ AGE_label_fr         : chr [1:1905] "Sans objet" "Sans objet" "Sans objet" "Sans objet" ...
FALSE  $ AGE_label_en         : chr [1:1905] "Not applicable" "Not applicable" "Not applicable" "Not ap
FALSE  $ UNIT_MEASURE_label_fr: chr [1:1905] "sans objet" "sans objet" "nombre" "nombre" ...
FALSE  $ UNIT_MEASURE_label_en: chr [1:1905] "not applicable" "not applicable" "number" "number" ...
FALSE  $ CORRECTION_label_fr  : chr [1:1905] "Non corrigé" "Non corrigé" "Non corrigé" "Non corrigé" ..
FALSE  $ CORRECTION_label_en  : chr [1:1905] "Uncorrected" "Uncorrected" "Uncorrected" "Uncorrected" ..
FALSE  $ dim1                 : chr [1:1905] "M" "M" "A" "A" ...
FALSE  $ dim2                 : chr [1:1905] "TAUX_MORTALITE" "TAUX_MORTALITE" "DECES_DOMICILIES" "DECE
FALSE  $ dim3                 : chr [1:1905] "TAUX" "TAUX" "VALEUR_ABSOLUE" "VALEUR_ABSOLUE" ...
FALSE  $ dim4                 : chr [1:1905] "TXMORINF" "TXMORINF" "DECES-DOM" "DECES-DOM" ...
FALSE  $ dim5                 : chr [1:1905] "FM" "FR-D976" "AU" "F_H_IDF" ...
FALSE  $ dim6                 : chr [1:1905] "0" "0" "SO" "SO" ...
FALSE  $ dim7                 : chr [1:1905] "SO" "SO" "SO" "SO" ...
FALSE  $ dim8                 : chr [1:1905] "SO" "SO" "NOMBRE" "NOMBRE" ...
FALSE  $ dim9                 : chr [1:1905] "BRUT" "BRUT" "BRUT" "BRUT" ...
```

Le résultat est un tibble comportant 1905 lignes et 39 colonnes. Il correspond en pratique aux 1905 séries chronologiques que l'on peut extraire de la base de données. Chaque série dispose d'un code unique contenu dans la variable *idbank*.

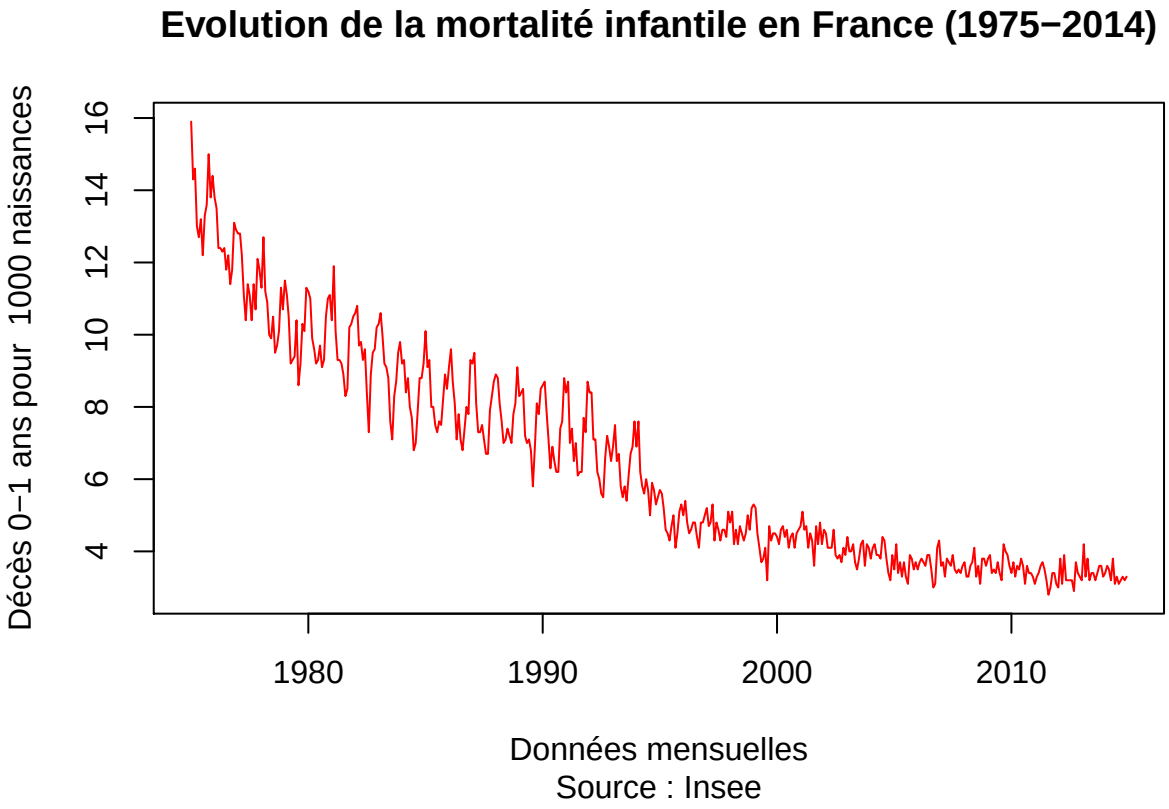
1.3.4 Extraction d'une série à l'aide de son identifiant

Une première solution pour extraire une série consiste à parcourir le tableau des variables jusqu'à repérer la ligne qui nous intéresse puis à noter son idbank et à extraire la série correspondante à l'aide de la fonction **get_insee_idbank()**. Par exemple, la première ligne du tableau des variables dont le code est "000436398" va renvoyer un tableau du taux brut de mortalité infantile en France métropolitaine de Janvier 1975 à Décembre 2014. On peut en faire rapidement un graphique avec la fonction *plot()* de R-Base

```
don<-get_insee_idbank("000436398")
```

```
FALSE   |                                                                      |
```

```
don<-don[order(don$DATE),1:3]
plot(don$DATE,don$OBS_VALUE,
     type ="l",
     col="red",
     ylab = "Décès 0-1 ans pour  1000 naissances",
     xlab = "Données mensuelles",
     main = "Evolution de la mortalité infantile en France (1975-2014)",
     sub = "Source : Insee")
```



On remarque que la courbe a des oscillations saisonnières beaucoup moins fortes après 1995 ce qui est sans doute lié à un changement dans le mode de collecte des données plutôt qu'à la réalité.

On note aussi que les données s'arrêtent en 2014 ce qui est bizarre puisque l'API devrait nous donner les chiffres les plus récents. en fait les données plus récentes sont disponibles mais elles font partie d'une autre série de données.

1.3.5 Extraction d'un ensemble de séries d'un même tableau

Supposons que l'on veuille extraire trois courbes décrivant l'espérance de vie des hommes en France métropolitaine, à 20, 40 et 60 ans. Nous lançons alors une requête pour ne retenir dans le tableau des variables que les lignes qui nous intéressent.

```
sel  =
  get_idbank_list("DECES-MORTALITE") %>%
  filter(SEXE == "1") %>%
  filter(FREQ == "A") %>% #données annuelles
  filter(REF_AREA == "FM") %>% #France métropolitaine
  filter(DEMOGRAPHIE %in% c("ESPV-20","ESPV-40","ESPV-60")) # Espérance de vie

kable(head(sel))
```

nomflow	idbank	cleFlow
DECES-MORTALITE	001686948	A.ESPERANCE_VIE.VALEUR_ABSOLUE.ESPV-20.FM.1.SO.ANNEES.BRUT
DECES-MORTALITE	001686949	A.ESPERANCE_VIE.VALEUR_ABSOLUE.ESPV-40.FM.1.SO.ANNEES.BRUT
DECES-MORTALITE	001686950	A.ESPERANCE_VIE.VALEUR_ABSOLUE.ESPV-60.FM.1.SO.ANNEES.BRUT
DECES-MORTALITE	010536470	A.ESPERANCE_VIE.VALEUR_ABSOLUE.ESPV-20.FM.1.SO.ANNEES.BRUT
DECES-MORTALITE	010536474	A.ESPERANCE_VIE.VALEUR_ABSOLUE.ESPV-40.FM.1.SO.ANNEES.BRUT
DECES-MORTALITE	010536478	A.ESPERANCE_VIE.VALEUR_ABSOLUE.ESPV-60.FM.1.SO.ANNEES.BRUT

On découvre que le programme renvoie **6 lignes au lieu de 3**. Pourquoi ? Parce que l'INSEE stocke différemment des séries anciennes et des séries récentes. Il faut donc effectuer une requête sur les 4 codes à la fois pour avoir la série la plus longue.

1.3.6 Recupération et nettoyage des données

On récupère les données puis on procède à un petit nettoyage du tableau pour ne conserver que les colonnes utiles.

```
don = get_insee_idbank(sel$idbank)
```

```
FALSE   |                                                                      |
```

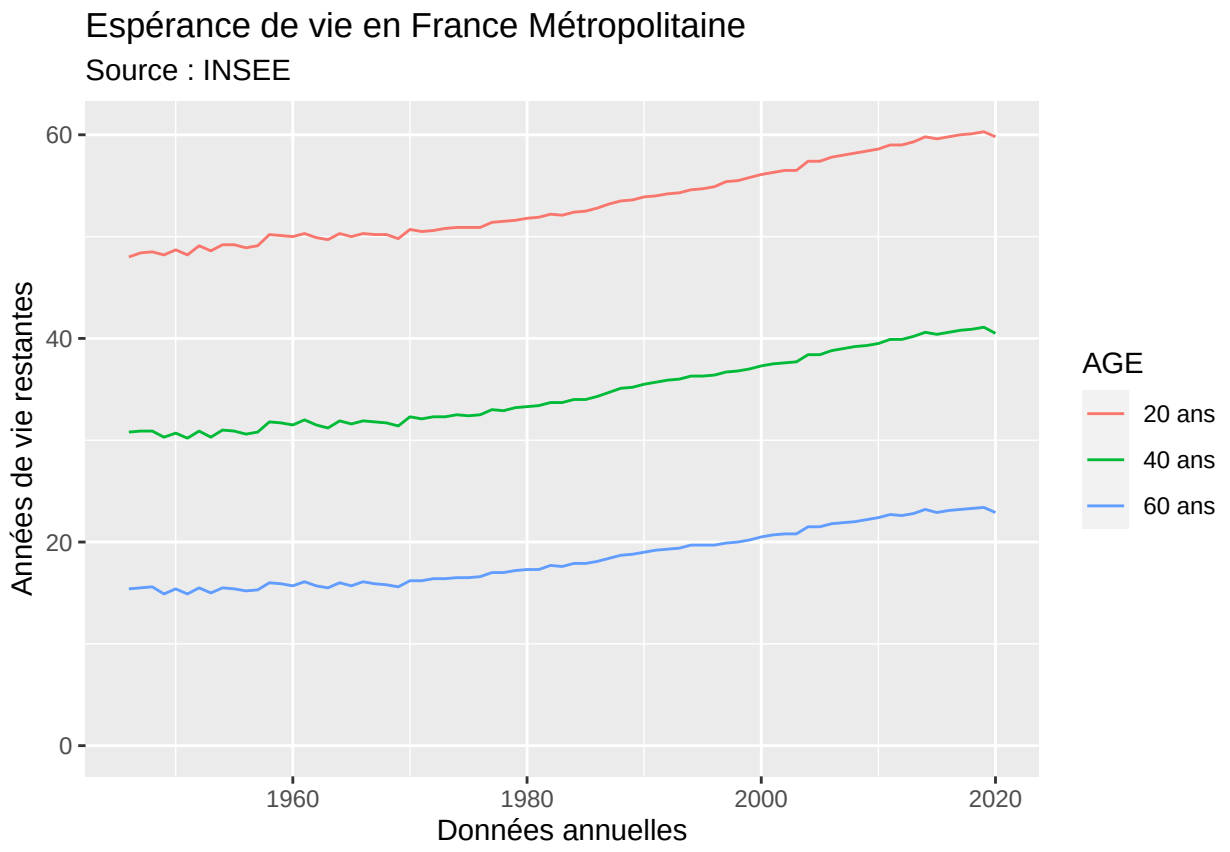
```
don2<-don %>% select(ANNEE = DATE, ESPVIE= OBS_VALUE, AGE = TITLE_FR) %>%
              mutate(AGE = as.factor(AGE)) %>%
              arrange(AGE, ANNEE)
levels(don2$AGE) <- c("20 ans", "40 ans","60 ans")
kable(head(don2))
```

ANNEE	ESPVIE	AGE
1946-01-01	48.0	20 ans
1947-01-01	48.4	20 ans
1948-01-01	48.5	20 ans
1949-01-01	48.2	20 ans
1950-01-01	48.7	20 ans
1951-01-01	48.2	20 ans

1.3.7 Construction d'un graphique

On peut maintenant construire notre graphique à l'aide par exemple de `ggplot2` :

```
p<-ggplot(don2) +
   aes(x=ANNEE,y=ESPVIE, color = AGE) +
    geom_line() +
    ggtitle(label= "Espérance de vie en France Métropolitaine",
            subtitle = "Source : INSEE")+
    scale_x_date("Données annuelles") +
    scale_y_continuous("Années de vie restantes",limits = c(0,NA))
p
```



1.3.8 Discussion

Comme on peut le voir, l'utilisation d'un package simplifie l'usage des API mais ne dispense pas d'un apprentissage souvent long pour comprendre toutes les finesses du package (et parfois ses bugs …). Dans le cas du package INSEE, l'utilisation s'avère assez lourde mais permet d'accéder à un nombre considérable de données !

1.4 Exercices

1.4.1 Exercice 1 : utilisation de httr et jsonlite

Déterminer la durée et la date des 10 prochains dates de passage de l'ISS au dessus de Paris (Latitude = 48.86, Longitude = 2.35)

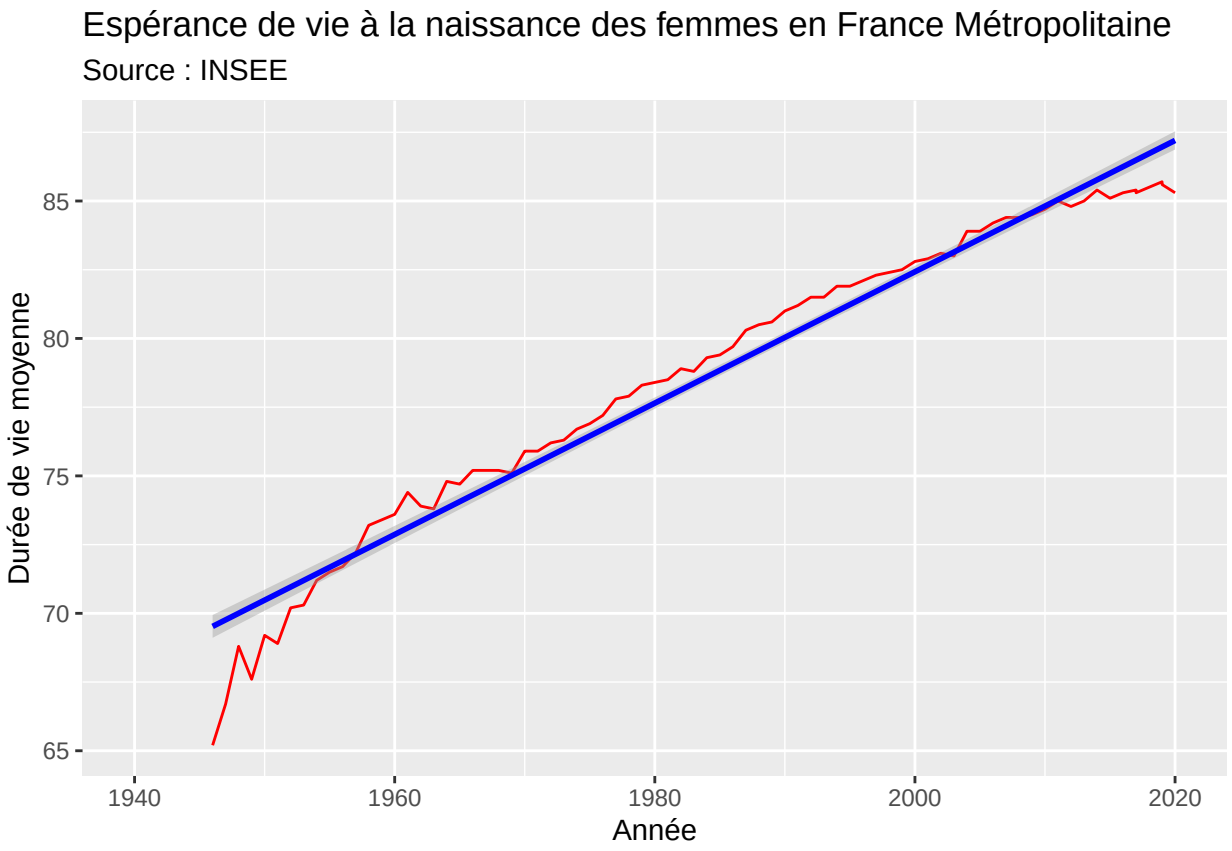
Table 1.3: Prochains passages de l'ISS au dessus de Paris

duration	risetime
356	2021-02-24 02:57:06
621	2021-02-24 04:30:47
655	2021-02-24 06:07:06
651	2021-02-24 07:44:10
655	2021-02-24 09:21:07
591	2021-02-24 10:58:05
70	2021-02-25 02:12:25
591	2021-02-25 03:43:45
654	2021-02-25 05:19:40
651	2021-02-25 06:56:40

1.4.2 Exercice 2 : utilisation du package 'insee'

Construire à l'aide du package INSEE un graphique de l'évolution mensuelle de l'espérance de vie des femmes à la naissance en France Métropolitaine de 1945 à 2020.

```
FALSE   |                                                                      |
```



1.4.3 Exercice 3 : Osrm

On se propose de calculer la distance routière Paris Rouen en temps et en kilomètre à l'aide de l'API osrm.

Paris (48.863186 ; 2.339754) Rouen (49.443232.; 1.099971)

Vous pouvez effectuer le calcul :

- soit avec l'API osrm

http://project-osrm.org/docs/v5.15.2/api/#general-options

- soit avec le package R osrm de Thimotee Giraud qui émule l'API

https://rgeomatic.hypotheses.org/1798

1.4.4 Exercice 4 : Exploration de nouvelles API

Vous devez identifier une API intéressante, accessible soit par un package R, soit par une combinaison de comandes GET() puis montrer son utilisation à l'aide d'un exemple de création d'un tableau puis d'un graphique.

Vous présenterez le résultat sous la forme d'un document markdown d'une à deux pages maximum.

Chapitre 2

Collecter des données à l'aide d'une API

```
## Global options
library(knitr)
library(dplyr)
opts_chunk$set(echo=TRUE,
               cache=TRUE,
               prompt=FALSE,
               tidy=FALSE,
               comment=NA,
               message=FALSE,
               warning=FALSE,
               options(scipen=999))
```

2.1 Objectifs

Supposons que l'on souhaite télécharger la population, le PIB et les émisssions de CO2 des pays du monde de 2000 à 2015. Plutôt que d'aller chercher des fichiers sur un site web, nous allons utiliser une API proposée par la Banque Mondial qui permet de télécharger les données facilement et surtout de les mettre à jour régulièrement. Pour cela on va installer le package R correspondant à l'API `wbstats` de la Banque mondiale.

https://cran.r-project.org/web/packages/wbstats/vignettes/Using_the_wbstats_package.html

Au moment du chargement du package, il est créé un fichier wb_cachelist qui fournit l'ensemble des donnes disponibles sous la forme d'une liste de tableaux de méta-données.

```
library("wbstats")
cat<-wb_cachelist
str(cat,max.level = 1)
```

```
List of 8
 $ countries    : tibble [304 x 18] (S3: tbl_df/tbl/data.frame)
 $ indicators   : tibble [16,649 x 8] (S3: tbl_df/tbl/data.frame)
 $ sources      : tibble [63 x 9] (S3: tbl_df/tbl/data.frame)
 $ topics       : tibble [21 x 3] (S3: tbl_df/tbl/data.frame)
 $ regions      : tibble [48 x 4] (S3: tbl_df/tbl/data.frame)
```

```
 $ income_levels: tibble [7 x 3] (S3: tbl_df/tbl/data.frame)
 $ lending_types: tibble [4 x 3] (S3: tbl_df/tbl/data.frame)
 $ languages    : tibble [23 x 3] (S3: tbl_df/tbl/data.frame)
```

2.2 Le tableau "countries"

Il fournit des renseignements de base sur les différents pays, leurs codes, etc.

```
str(cat$countries)
```

```
tibble [304 x 18] (S3: tbl_df/tbl/data.frame)
 $ iso3c             : chr [1:304] "ABW" "AFG" "AFR" "AGO" ...
 $ iso2c             : chr [1:304] "AW" "AF" "A9" "AO" ...
 $ country           : chr [1:304] "Aruba" "Afghanistan" "Africa" "Angola" ...
 $ capital_city      : chr [1:304] "Oranjestad" "Kabul" NA "Luanda" ...
 $ longitude         : num [1:304] -70 69.2 NA 13.2 19.8 ...
 $ latitude          : num [1:304] 12.52 34.52 NA -8.81 41.33 ...
 $ region_iso3c      : chr [1:304] "LCN" "SAS" NA "SSF" ...
 $ region_iso2c      : chr [1:304] "ZJ" "8S" NA "ZG" ...
 $ region            : chr [1:304] "Latin America & Caribbean" "South Asia" "Aggregates" "Sub-Saharan A
 $ admin_region_iso3c: chr [1:304] NA "SAS" NA "SSA" ...
 $ admin_region_iso2c: chr [1:304] NA "8S" NA "ZF" ...
 $ admin_region      : chr [1:304] NA "South Asia" NA "Sub-Saharan Africa (excluding high income)" ...
 $ income_level_iso3c: chr [1:304] "HIC" "LIC" NA "LMC" ...
 $ income_level_iso2c: chr [1:304] "XD" "XM" NA "XN" ...
 $ income_level      : chr [1:304] "High income" "Low income" "Aggregates" "Lower middle income" ...
 $ lending_type_iso3c: chr [1:304] "LNX" "IDX" NA "IBD" ...
 $ lending_type_iso2c: chr [1:304] "XX" "XI" NA "XF" ...
 $ lending_type      : chr [1:304] "Not classified" "IDA" "Aggregates" "IBRD" ...
```

Le tableau comporte 304 observation et il mélange des pays (France), des fragments de pays (Réunion) et des agrégats de pays (Europe). Il faudra donc bien faire attention lors de l'extraction à réfléchir à ce que l'on souhaite utiliser. Par exemple, si l'on veut juste les pays :

```
## Programme en langage R_base
# pays<-cat$countries[cat$countries$income_level!="Aggregates",c("iso3c", "country","capital_city","lon


## Programme en langage dplyr

pays <- cat$countries %>%
          filter(income_level !="Aggregates") %>%
          select(iso3c,country, capital_city, latitude, longitude, region, income_level)


kable(head(pays))
```

iso3c	country	capital_city	latitude	longitude	region	income_level
ABW	Aruba	Oranjestad	12.51670	-70.0167	Latin America & Caribbean	High income
AFG	Afghanistan	Kabul	34.52280	69.1761	South Asia	Low income
AGO	Angola	Luanda	-8.81155	13.2420	Sub-Saharan Africa	Lower middle
ALB	Albania	Tirane	41.33170	19.8172	Europe & Central Asia	Upper middle
AND	Andorra	Andorra la Vella	42.50750	1.5218	Europe & Central Asia	High income
ARE	United Arab Emirates	Abu Dhabi	24.47640	54.3705	Middle East & North Africa	High income

2.3 Le tableau indicators

Il comporte pas loin de 17000 variables … Autant dire qu'il est difficile de l'explorer facilement si l'on ne sait pas ce que l'on cherche.

```
indic<-cat$indicators
dim(indic)
```

```
[1] 16649     8
```

```
kable(head(indic))
```

indicator_id	indicator	unit	indicator_desc
1.0.HCount.1.90usd	Poverty Headcount ($1.90 a day)	NA	The poverty headcount index measures the pr
1.0.HCount.2.5usd	Poverty Headcount ($2.50 a day)	NA	The poverty headcount index measures the pr
1.0.HCount.Mid10to50	Middle Class ($10-50 a day) Headcount	NA	The poverty headcount index measures the pr
1.0.HCount.Ofcl	Official Moderate Poverty Rate-National	NA	The poverty headcount index measures the pr
1.0.HCount.Poor4uds	Poverty Headcount ($4 a day)	NA	The poverty headcount index measures the pr
1.0.HCount.Vul4to10	Vulnerable ($4-10 a day) Headcount	NA	The poverty headcount index measures the pr

2.3.1 Recherche du code d'un indicateur

Supposons qu'on recherche les données récentes sur les émissions de CO2. On va utiliser le mot-clé *CO2* pour rechercher les variables correspondantes dans le catalogue à l'aide de la fonction `wbsearch`, ce qui donne 45 réponses

```
vars <- wbsearch(pattern = "CO2",fields="indicator")
kable(vars)
```

	indicatorID	indicator
5294	EN.ATM.CO2E.CP.KT	CO2 emissions from cement production (thousand metric tons)
5295	EN.ATM.CO2E.EG.ZS	CO2 intensity (kg per kg of oil equivalent energy use)
5296	EN.ATM.CO2E.FF.KT	CO2 emissions from fossil-fuels, total (thousand metric tons)
5297	EN.ATM.CO2E.FF.ZS	CO2 emissions from fossil-fuels (% of total)
5298	EN.ATM.CO2E.GDP	CO2 emissions, industrial (kg per 1987 US$ of GDP)
5299	EN.ATM.CO2E.GF.KT	CO2 emissions from gaseous fuel consumption (kt)
5300	EN.ATM.CO2E.GF.ZS	CO2 emissions from gaseous fuel consumption (% of total)
5301	EN.ATM.CO2E.GL.KT	CO2 emissions from gas flaring (thousand metric tons)
5302	EN.ATM.CO2E.KD.87.GD	CO2 emissions, industrial (kg per 1987 US$ of GDP)
5303	EN.ATM.CO2E.KD.GD	CO2 emissions (kg per 2010 US$ of GDP)
5304	EN.ATM.CO2E.KT	CO2 emissions (kt)
5305	EN.ATM.CO2E.LF.KT	CO2 emissions from liquid fuel consumption (kt)
5306	EN.ATM.CO2E.LF.ZS	CO2 emissions from liquid fuel consumption (% of total)
5307	EN.ATM.CO2E.PC	CO2 emissions (metric tons per capita)
5308	EN.ATM.CO2E.PP.GD	CO2 emissions (kg per PPP $ of GDP)
5309	EN.ATM.CO2E.PP.GD.KD	CO2 emissions (kg per 2017 PPP $ of GDP)
5310	EN.ATM.CO2E.SF.KT	CO2 emissions from solid fuel consumption (kt)
5311	EN.ATM.CO2E.SF.ZS	CO2 emissions from solid fuel consumption (% of total)
5312	EN.ATM.GHGO.KT.CE	Other greenhouse gas emissions, HFC, PFC and SF6 (thousand metric tons of CO2
5314	EN.ATM.GHGT.KT.CE	Total greenhouse gas emissions (kt of CO2 equivalent)
5316	EN.ATM.HFCG.KT.CE	HFC gas emissions (thousand metric tons of CO2 equivalent)
5317	EN.ATM.METH.AG.KT.CE	Agricultural methane emissions (thousand metric tons of CO2 equivalent)
5319	EN.ATM.METH.EG.KT.CE	Methane emissions in energy sector (thousand metric tons of CO2 equivalent)
5322	EN.ATM.METH.KT.CE	Methane emissions (kt of CO2 equivalent)
5323	EN.ATM.METH.PC	Methane emissions (kt of CO2 equivalent per capita)
5325	EN.ATM.NOXE.AG.KT.CE	Agricultural nitrous oxide emissions (thousand metric tons of CO2 equivalent)
5327	EN.ATM.NOXE.EG.KT.CE	Nitrous oxide emissions in energy sector (thousand metric tons of CO2 equivalent)
5330	EN.ATM.NOXE.IN.KT.CE	Industrial nitrous oxide emissions (thousand metric tons of CO2 equivalent)
5332	EN.ATM.NOXE.KT.CE	Nitrous oxide emissions (thousand metric tons of CO2 equivalent)
5333	EN.ATM.NOXE.MT.CE	Nitrous oxide emissions (metric tons of CO2 equivalent)
5334	EN.ATM.NOXE.PC	Nitrous oxide emissions (metric tons of CO2 equivalent per capita)
5336	EN.ATM.PFCG.KT.CE	PFC gas emissions (thousand metric tons of CO2 equivalent)
5343	EN.ATM.SF6G.KT.CE	SF6 gas emissions (thousand metric tons of CO2 equivalent)
5348	EN.CLC.GHGR.MT.CE	GHG net emissions/removals by LUCF (Mt of CO2 equivalent)
5353	EN.CO2.BLDG.MT	CO2 emissions from residential buildings and commercial and public services (milli
5354	EN.CO2.BLDG.ZS	CO2 emissions from residential buildings and commercial and public services (% of
5355	EN.CO2.ETOT.MT	CO2 emissions from electricity and heat production, total (million metric tons)
5356	EN.CO2.ETOT.ZS	CO2 emissions from electricity and heat production, total (% of total fuel combusti
5357	EN.CO2.MANF.MT	CO2 emissions from manufacturing industries and construction (million metric tons
5358	EN.CO2.MANF.ZS	CO2 emissions from manufacturing industries and construction (% of total fuel com
5359	EN.CO2.OTHX.MT	CO2 emissions from other sectors, excluding residential buildings and commercial a
5360	EN.CO2.OTHX.ZS	CO2 emissions from other sectors, excluding residential buildings and commercial a
5361	EN.CO2.TRAN.MT	CO2 emissions from transport (million metric tons)
5362	EN.CO2.TRAN.ZS	CO2 emissions from transport (% of total fuel combustion)
8231	IN.ENV.CO2.CONC	CO2 Emission (in thousand metric tons of Carbon)

On va finalement trouver le code de la variable recherchée

- *EN.ATM.CO2E.KT* : émissions de CO2 en kilotonnes

Les deux autres variables dont nous avons besoin ont pour code

- *NY.GDP.MKTP.CD* : PIB en parités de pouvoir d'achat
- *SP.POP.TOTL* : Population totale

2.3.2 Extraction des métadonnées

Une fois que l'on pense connaître le code de nos variables, on peut extraire les métadonnés pour vérifier qu'il s'agit bien de ce que l'on cherche, quelle est la source exacte, quelle est l'unité de mesure …

```
# Programme R-base
meta<-cat$indicators[cat$indicators$indicator_id %in% c("SP.POP.TOTL","NY.GDP.MKTP.CD","EN.ATM.CO2E.KT"

# Programme dplyr
meta<-cat$indicators %>%
        filter(indicator_id %in% c("SP.POP.TOTL","NY.GDP.MKTP.CD","EN.ATM.CO2E.KT"))

kable(meta)
```

indicator_id	indicator	unit	indicator_desc
EN.ATM.CO2E.KT	CO2 emissions (kt)	NA	Carbon dioxide emissions are those stemming from the burning of fos
NY.GDP.MKTP.CD	GDP (current US$)	NA	GDP at purchaser's prices is the sum of gross value added by all resi
SP.POP.TOTL	Population, total	NA	Total population is based on the de facto definition of population, wh

2.4 L'extraction des données

Elle se fait à l'aide de la fonction `wb_data` qui comporte de nombreuses options.

2.4.1 le paramètre `indicator =`

Ce paramètre permet de choisir les indicateurs à collecter, ce qui suppose que l'on connaisse leur code. Par exemple, supposons que l'on veuille extraire la population et le PIB pour pouvoir calculer ensuite le PIB par habitant

```
df   <- wb_data(indicator  = c("NY.GDP.MKTP.CD","SP.POP.TOTL"))
dim(df)
```

```
[1] 13237     6
```

```
kable(head(df,6))
```

iso2c	iso3c	country	date	NY.GDP.MKTP.CD	SP.POP.TOTL
AW	ABW	Aruba	1960	NA	54211
AW	ABW	Aruba	1961	NA	55438
AW	ABW	Aruba	1962	NA	56225
AW	ABW	Aruba	1963	NA	56695
AW	ABW	Aruba	1964	NA	57032
AW	ABW	Aruba	1965	NA	57360

- **commentaire** : Nous obtenons un tableau très grand (> 13000 lignes) qui comporte les valeurs pour toutes les dates disponibles depuis 1960 et pour tous les pays, même si les valeurs sont souvent manquantes.

2.4.2 le choix d'une période de temps

2.4.2.1 les paramètres `startdate =` et `startdate =`

Ces deux paramètres permettent de choisir une plage de temps. On peut par exemple décider de ne collecter que les données relatives aux années 2014, 2015 et 2016

```
df   <- wb_data(indicator  = c("NY.GDP.MKTP.CD","SP.POP.TOTL"),
                start_date = 2014,
                end_date = 2016)
dim(df)
```

```
[1] 651   6
```

```
kable(head(df,6))
```

iso2c	iso3c	country	date	NY.GDP.MKTP.CD	SP.POP.TOTL
AW	ABW	Aruba	2014	2765363128	103774
AW	ABW	Aruba	2015	2919553073	104341
AW	ABW	Aruba	2016	2965921788	104872
AF	AFG	Afghanistan	2014	20484885120	33370794
AF	AFG	Afghanistan	2015	19907111419	34413603
AF	AFG	Afghanistan	2016	18017749074	35383128

- **commentaire** : Le tableau ne comporte donc plus que 651 lignes correspondant aux trois dates pour les différents pays du Monde.

2.4.2.2 Le paramètre `mrv` (most recent value)

Lorsque l'on souhaite juste obtenir les données les plus récentes, on peut remplacer les paramètres `startdate =` et `startdate =` par le paramètre `mrv =` suivit d'un chiffre indiquant le nombre d'années que l'on souhaite à partir de la date la plus récente. Avec mrv=1 on récupère uniquement la dernière année disponible pour au moins l'une des variables.

```
df   <- wb_data(indicator  = c("NY.GDP.MKTP.CD","SP.POP.TOTL"),
                mrv = 1)
dim(df)
```

```
[1] 217   6
```

```
kable(head(df,6))
```

iso2c	iso3c	country	date	NY.GDP.MKTP.CD	SP.POP.TOTL
AW	ABW	Aruba	2019	NA	106314
AF	AFG	Afghanistan	2019	19291104008	38041754
AO	AGO	Angola	2019	88815697793	31825295
AL	ALB	Albania	2019	15279183290	2854191
AD	AND	Andorra	2019	3154057987	77142
AE	ARE	United Arab Emirates	2019	421142267938	9770529

L'inconvénient de cette méthode est que cela peut aboutir à un grand nombre de valeurs manquantes si l'une des variables recherchée n'a pas été mise à jour. Par exemple, la variable relative au CO2 n'est pas disponible après 2016 et du coup le tableau va mélanger des dates différentes.

```
df   <- wb_data(indicator  = c("NY.GDP.MKTP.CD","SP.POP.TOTL","EN.ATM.CO2E.KT" ),
                mrv =1)
dim(df)
```

```
[1] 434   7
```

```
kable(head(df,6))
```

iso2c	iso3c	country	date	EN.ATM.CO2E.KT	NY.GDP.MKTP.CD	SP.POP.TOTL
AW	ABW	Aruba	2016	883.747	NA	NA
AW	ABW	Aruba	2019	NA	NA	106314
AF	AFG	Afghanistan	2016	8672.455	NA	NA
AF	AFG	Afghanistan	2019	NA	19291104008	38041754
AO	AGO	Angola	2016	34693.487	NA	NA
AO	AGO	Angola	2019	NA	88815697793	31825295

Il est donc préférable de sélectioner une période plus longue mrv=5 et de faire ensuite soi-même le tri :

2.4.3 Le choix des unités géographiques

Le paramètre `country =` permet de choisir les entités spatiales à collecter, soit sous forme de liste de codes, soit à l'aide de valeurs spéciales. Par défaut; il renvoie la liste de tous les pays, mais on peut se limiter à quelques uns seulement à l'aide de leur nom en anglais (risqué ...) ou de leur code ISO3 (plus sûr)

2.4.3.1 sélection de pays

```
df   <- wb_data(indicator  = c("NY.GDP.MKTP.CD","SP.POP.TOTL"),
                start_date = 2018,
                end_date = 2018,
                country = c("USA","CHN"))
df$GDP.per.capita <- round(df$NY.GDP.MKTP.CD / df$SP.POP.TOTL,0)
kable(head(df,6))
```

iso2c	iso3c	country	date	NY.GDP.MKTP.CD	SP.POP.TOTL	GDP.per.capita
CN	CHN	China	2018	13894817549374	1392730000	9977
US	USA	United States	2018	20580159776000	326687501	62996

- **commentaire** : Il est donc facile de travailler sur un petit nombre de pays que l'on souhaite comparer.

2.4.3.2 Opérateurs spéciaux

Il existe un certain nombre de paramètres spéciaux que l'on peut utiliser à la place de la liste des pays :

- "countries_only" (Default)
- "regions_only"
- "admin_regions_only"
- "income_levels_only"
- "aggregates_only"
- "all"

```
df   <- wb_data(indicator  = c("NY.GDP.MKTP.CD","SP.POP.TOTL"),
                start_date = 2018,
                end_date = 2018,
                country = "regions_only")
df$GDP.per.capita <- round(df$NY.GDP.MKTP.CD / df$SP.POP.TOTL,0)
kable(df)
```

iso2c	iso3c	country	date	NY.GDP.MKTP.CD	SP.POP.TOTL	GDP.per.capita
Z4	EAS	East Asia & Pacific	2018	26351346134238	2328138066	11319
Z7	ECS	Europe & Central Asia	2018	23145668465857	917922619	25215
ZJ	LCN	Latin America & Caribbean	2018	5823558182038	640467174	9093
ZQ	MEA	Middle East & North Africa	2018	3566140736119	448912962	7944
XU	NAC	North America	2018	22303646726082	363809186	61306
8S	SAS	South Asia	2018	3445471615275	1814388744	1899
ZG	SSF	Sub-Saharan Africa	2018	1721156393780	1078306520	1596

- **commentaire** : Nous avons extrait les données par grandes régions du Monde pour l'année 2016

2.4.4 Le format de sortie du tableau

Il existe deux façons d'extraire un tableau comprenant plusieurs variables ou plusieurs dates, selon que l'on veut un tableau large (wide) ou étroit. On peut régler la sortie à l'aide du paramètre `return_wide` qui est TRUE par défaut mais que l'on peut régler sur FALSE.

2.4.4.1 `return_wide` = FALSE

```
df   <- wb_data(indicator  = c("NY.GDP.MKTP.CD","SP.POP.TOTL"),
                return_wide = TRUE,
                start_date = 2016,
                end_date = 2018,
                country = c("USA","CHN"))
df
```

```
# A tibble: 6 x 6
  iso2c iso3c country        date NY.GDP.MKTP.CD SP.POP.TOTL
  <chr> <chr> <chr>         <dbl>          <dbl>       <dbl>
1 CN    CHN   China          2016        1.12e13  1378665000
2 CN    CHN   China          2017        1.23e13  1386395000
3 CN    CHN   China          2018        1.39e13  1392730000
4 US    USA   United States  2016        1.87e13   322941311
5 US    USA   United States  2017        1.95e13   324985539
6 US    USA   United States  2018        2.06e13   326687501
```

2.4.4.2 `return_wide` = FALSE

```
df   <- wb_data(indicator  = c("NY.GDP.MKTP.CD","SP.POP.TOTL"),
                return_wide = FALSE,
                start_date = 2016,
```

```
                end_date = 2018,
                country = c("USA","CHN"))
df[,1:7]
```

```
# A tibble: 12 x 7
   indicator_id   indicator         iso2c iso3c country        date   value
   <chr>          <chr>             <chr> <chr> <chr>         <dbl>   <dbl>
 1 NY.GDP.MKTP.CD GDP (current US$) CN    CHN   China          2018 1.39e13
 2 NY.GDP.MKTP.CD GDP (current US$) CN    CHN   China          2017 1.23e13
 3 NY.GDP.MKTP.CD GDP (current US$) CN    CHN   China          2016 1.12e13
 4 NY.GDP.MKTP.CD GDP (current US$) US    USA   United States  2018 2.06e13
 5 NY.GDP.MKTP.CD GDP (current US$) US    USA   United States  2017 1.95e13
 6 NY.GDP.MKTP.CD GDP (current US$) US    USA   United States  2016 1.87e13
 7 SP.POP.TOTL    Population, total CN    CHN   China          2018 1.39e 9
 8 SP.POP.TOTL    Population, total CN    CHN   China          2017 1.39e 9
 9 SP.POP.TOTL    Population, total CN    CHN   China          2016 1.38e 9
10 SP.POP.TOTL    Population, total US    USA   United States  2018 3.27e 8
11 SP.POP.TOTL    Population, total US    USA   United States  2017 3.25e 8
12 SP.POP.TOTL    Population, total US    USA   United States  2016 3.23e 8
```

2.5 Exercices

2.5.1 Exercice 1

Extraire les métadonnées relatives à la variable *SP.URB.TOTL*

indicator_id	indicator	unit	indicator_desc
SP.URB.TOTL	Urban population	NA	Urban population refers to people living in urban areas as defined by nation

2.5.2 Exercice 2

Créer un tableau de la population des pays du monde en 2000, triez le par ordre décroissant et affichez les 10 pays les plus peuplés avec leur nom,leur code et la population en millions

Code	Pays	Population
CHN	China	1262.6
IND	India	1056.6
USA	United States	282.2
IDN	Indonesia	211.5
BRA	Brazil	174.8
RUS	Russian Federation	146.6
PAK	Pakistan	142.3
BGD	Bangladesh	127.7
JPN	Japan	126.8
NGA	Nigeria	122.3

2.6 Exercice 3

On se propose de comparer l'évolution des émissions de CO2 de la Chine (CHN), l'Inde (IND), la Russie (RUS) le Japon (JPN) et des Etats-Unis d'Amérique (USA) de 1995 à 2015.