



# Introduction à l'analyse de données avec R

---

Guide d'analyse des données de  
santé publique avec R

# Notes de leçon | Installation de R et RStudio

GRAPH Network & OMS, soutenu par le Fonds Mondial

January 2024

Ce cours a été créé par le Réseau GRAPH, une organisation à but non lucratif basée à l'Institut de santé globale de l'Université de Genève, en collaboration avec l'Organisation mondiale de la Santé, dans le cadre d'une subvention du Fonds mondial pour créer des cours afin de renforcer les capacités nationales en matière d'analyse épidémiologique.



Introduction .....
Travailler localement vs sur le cloud .....
RStudio Cloud .....
Installation sur Windows .....
Télécharger et installer R .....
Télécharger, installer et exécuter RStudio .....
Installation sur macOS .....
Télécharger et installer R .....
Télécharger, installer et exécuter RStudio .....
Conclusion .....

---

## Objectif d'apprentissage

1. Vous pouvez accéder à R et RStudio, soit par le biais de RStudio.cloud, soit en téléchargeant et en installant ces logiciels sur votre ordinateur.

## Introduction

Pour commencer votre parcours avec R, nous devrons vous configurer avec les logiciels nécessaires, R et RStudio. **R** est le langage de programmation que vous utiliserez pour écrire du code, tandis que **RStudio** est un environnement de développement intégré (IDE) qui facilite l'utilisation de R.

## Travailler localement vs sur le cloud

Vous pouvez accéder et travailler avec R et RStudio de deux manières principales : en les téléchargeant sur votre ordinateur ou en utilisant un serveur web pour y accéder dans le cloud.

L'utilisation de R et RStudio sur le cloud est l'option la moins courante, mais cela peut être le bon choix si vous débutez avec la programmation et que vous ne voulez pas encore vous soucier de l'installation de logiciels. Vous pouvez également préférer l'option cloud si votre ordinateur local est vieux, lent ou inadapté à l'exécution de R.

Ci-dessous, nous passons en revue le processus de configuration de RStudio Cloud, Rstudio sur Windows et RStudio sur macOS séparément. Passez à la section qui vous concerne !

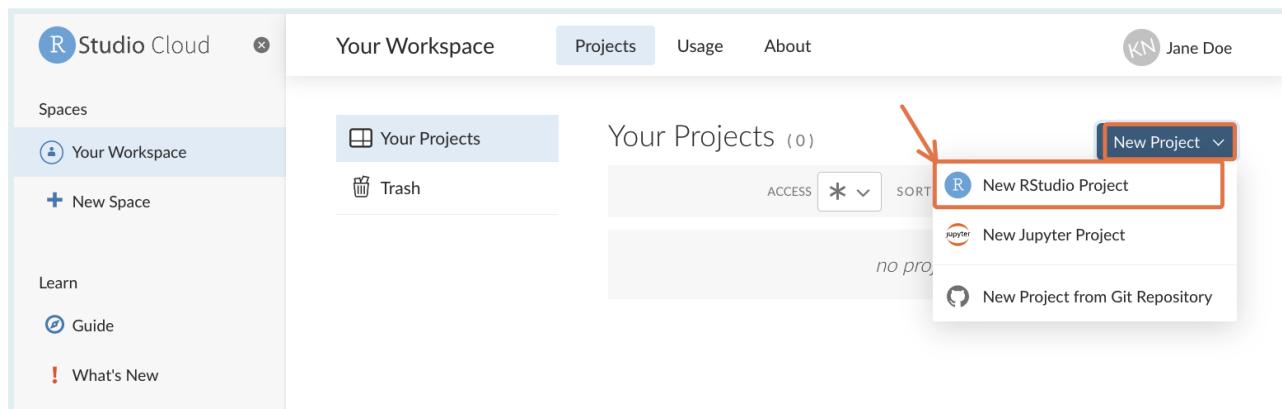
RStudio Cloud ne vous offre que 25 heures de projet gratuites par mois. Après cela, vous devrez passer à un plan payant. Si vous pensez que vous aurez besoin de plus de 25 heures

par mois, il est préférable d'éviter cette option.

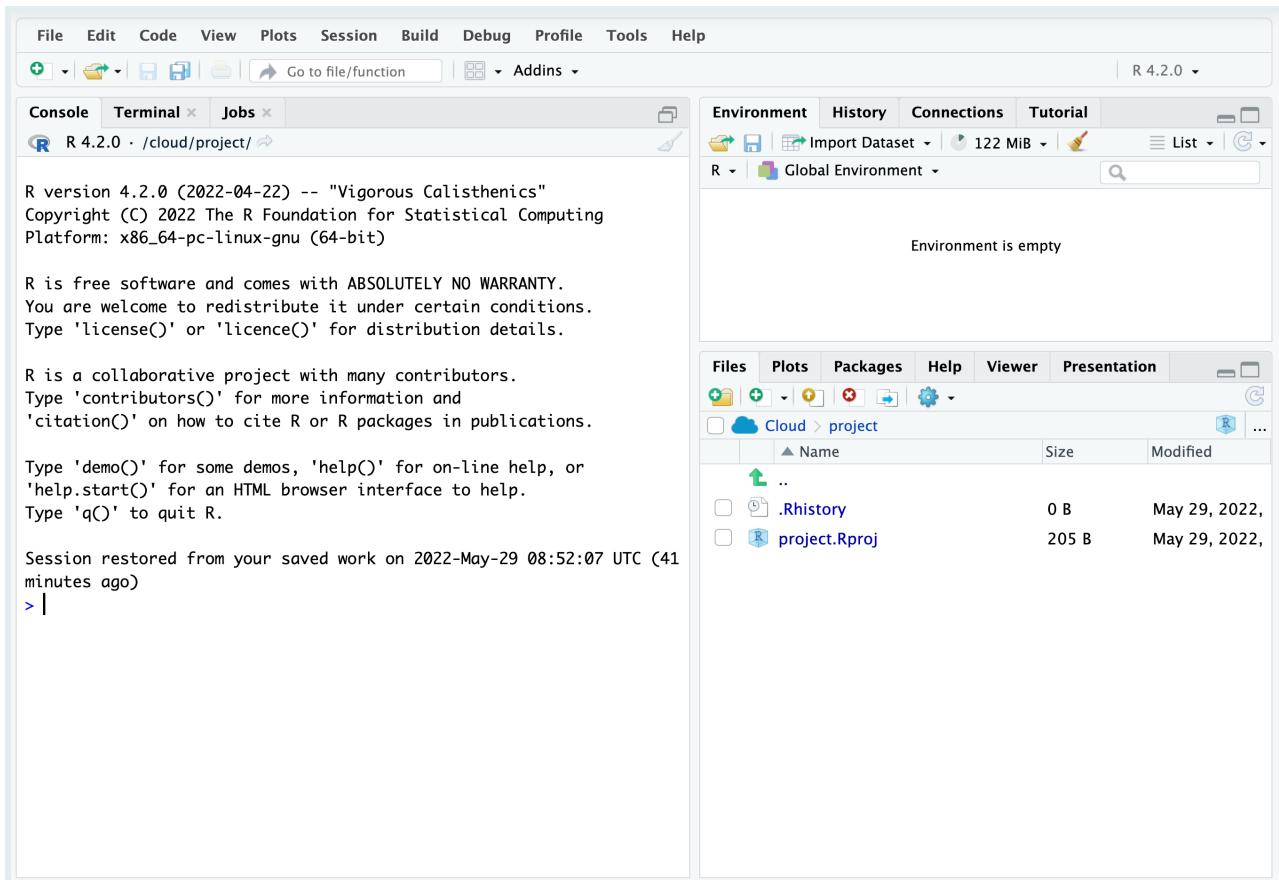
## RStudio Cloud

Si vous comptez travailler sur le cloud, suivez les étapes ci-dessous :

1. Accédez au site Web [rstudio.cloud](https://rstudio.cloud) et suivez les instructions pour créer un compte gratuit. (Nous vous recommandons de vous inscrire avec Google si vous disposez d'un compte Google, afin de ne pas avoir à vous souvenir de nouveaux mots de passe).
2. Une fois que vous avez terminé, cliquez sur l'icône "New project" en haut à droite et sélectionnez "New RStudio project".

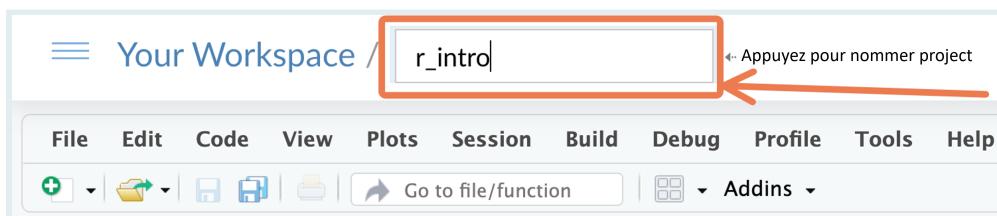


Vous devriez voir un écran comme celui-ci :



Voici RStudio, votre nouvelle maison pour longtemps !

En haut de l'écran, renommez le nom de projet "Untitled Project" en quelque chose comme "r\_intro".



Vous pouvez commencer à utiliser R en saisissant du code dans l'onglet "console" à gauche :

```
R version 4.2.0 (2022-04-22) -- "Vigorous Calisthenics"
Copyright (C) 2022 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

Session restored from your saved work on 2022-May-29 08:52:07 UTC (41
minutes ago)
> 2 + 2
```

Ecrivez code R ici!

Essayez d'utiliser R comme calculatrice ici ; tapez '2 + 2' et appuyez sur Entrée.

Et voilà, vous êtes prêt à vous lancer. Chaque fois que vous souhaitez rouvrir RStudio, rendez-vous sur [rstudio.cloud](#).

Passez à la section "Conclusion" du cours.

---

## Installation sur Windows

### Télécharger et installer R

Si vous travaillez sous Windows, suivez les étapes ci-dessous pour télécharger et installer R :

1. Rendez-vous sur [cran.rstudio.com](https://cran.rstudio.com) pour accéder à la page d'installation de R. Cliquez ensuite sur le lien de téléchargement pour Windows :

## The Comprehensive R Archive Network

### Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- [Download R for Linux \(Debian, Fedora/Redhat, Ubuntu\)](#)
- [Download R for macOS](#)
- [Download R for Windows](#)

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

2. Choisissez le sous-répertoire "base".

### R for Windows

Subdirectories:

[base](#)

[contrib](#)

[old contrib](#)

Binaries for base distribution. This is what you want to [install R for the first time](#).

Binaries of contributed CRAN packages (for R >= 3.4.x).

Binaries of contributed CRAN packages for outdated versions of R (for R < 3.4.x).

3. Cliquez ensuite sur le lien de téléchargement en haut de la page pour télécharger la dernière version de R :

### R-4.2.0 for Windows

[Download R-4.2.0 for Windows \(79 megabytes, 64 bit\)](#)

[README on the Windows binary distribution](#)

[New features in this version](#)

This build requires UCRT, which is part of Windows since Windows 10 and Windows Server 2016. On older systems, UCRT has to be installed manually from [here](#).

Notez que la capture d'écran ci-dessus peut ne pas afficher la dernière version.

4. Une fois le téléchargement terminé, cliquez sur le fichier téléchargé, puis suivez les instructions de la fenêtre d'installation. Pendant l'installation, vous ne devriez pas avoir à modifier les paramètres par défaut ; il vous suffit de cliquer sur "Suivant" jusqu'à ce que l'installation soit terminée.

Voilà qui est fait ! Vous devriez maintenant avoir R sur votre ordinateur. Mais vous n'aurez probablement jamais besoin d'interagir avec R directement. Vous utiliserez plutôt l'IDE RStudio pour travailler avec R. Suivez les instructions de la section suivante pour obtenir RStudio.

## Télécharger, installer et exécuter RStudio

Pour télécharger RStudio, rendez-vous sur

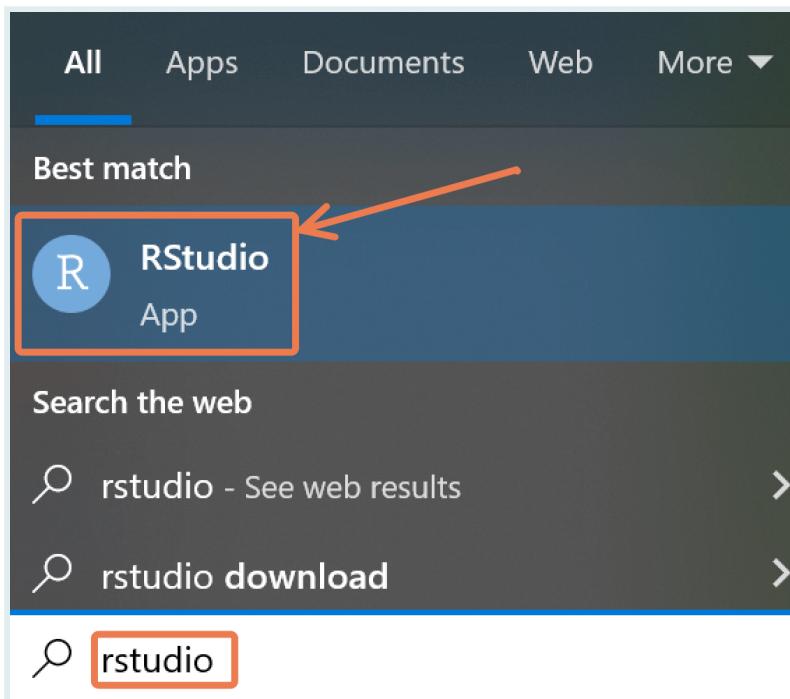
[rstudio.com/products/rstudio/download/#download](http://rstudio.com/products/rstudio/download/#download) et téléchargez la version Windows.

## 2. Download RStudio Desktop. Recommended for your system:

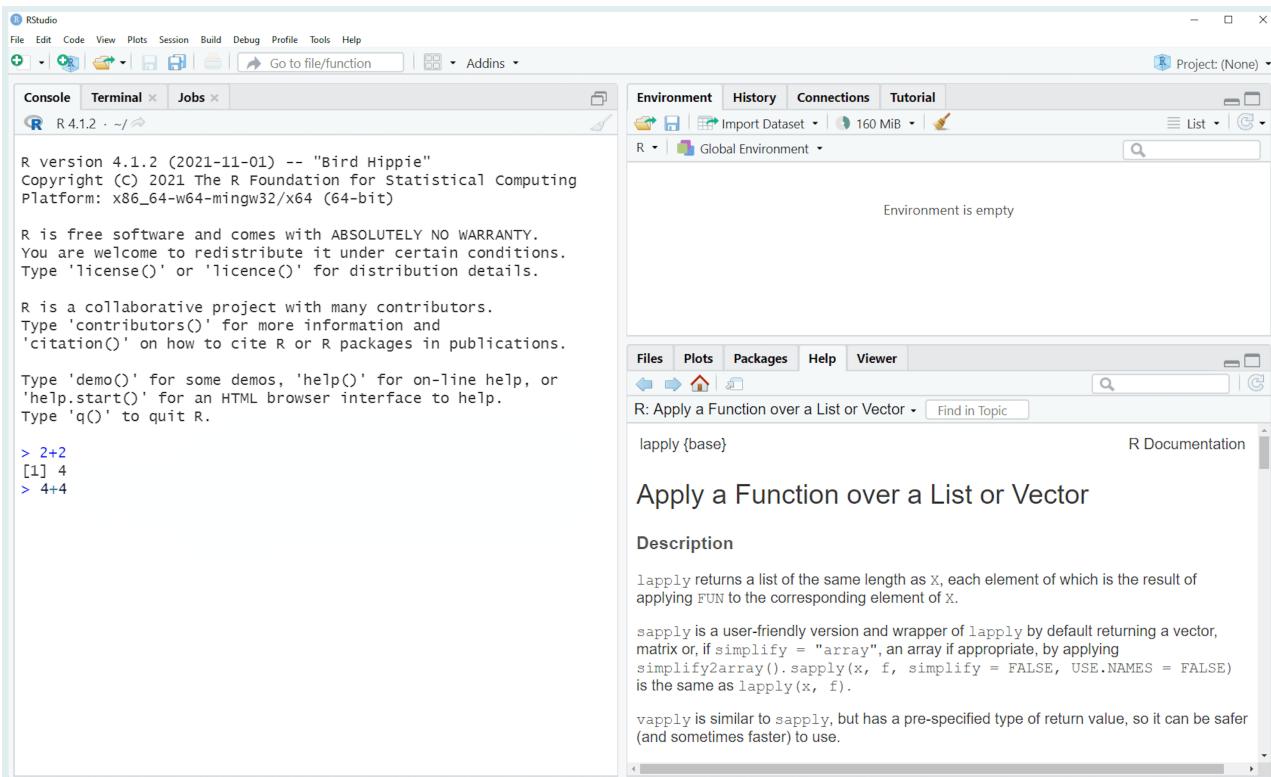


Une fois le téléchargement terminé, cliquez sur le fichier téléchargé et suivez les instructions d'installation.

Une fois installé, RStudio peut être ouvert comme n'importe quelle application sur votre ordinateur : appuyez sur la touche Windows pour ouvrir le menu Démarrer, et recherchez « rstudio ». Cliquez pour ouvrir l'application :

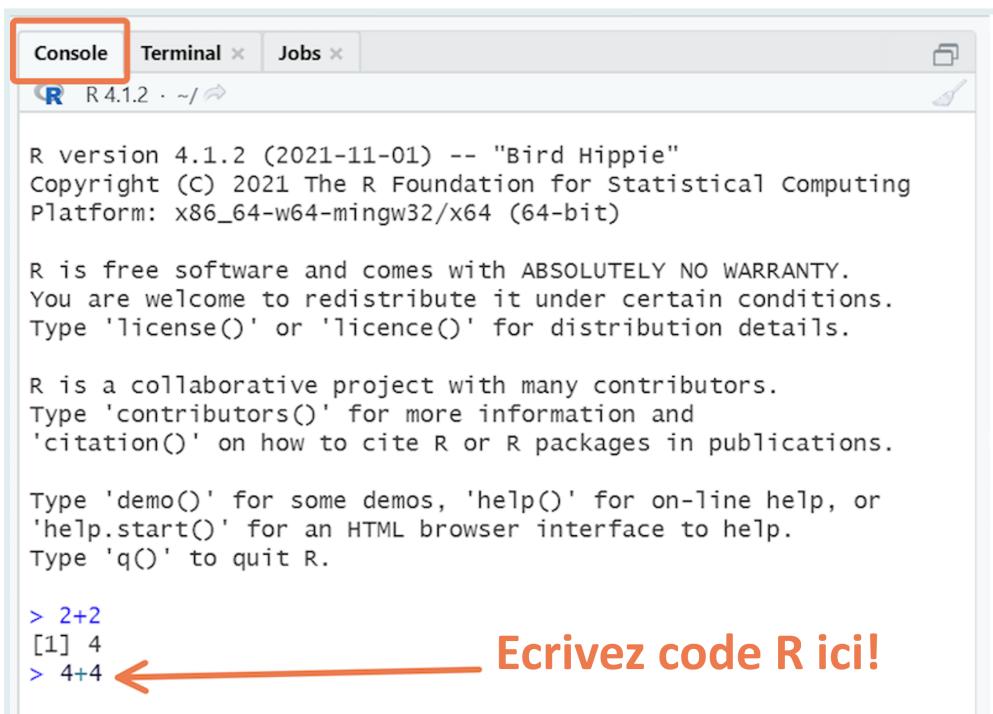


Vous devriez voir une fenêtre comme celle-ci :



Voici RStudio, votre nouvelle maison pour longtemps !

Vous pouvez commencer à utiliser R en saisissez du code dans l'onglet "console" à gauche :



Essayez d'utiliser R comme calculatrice ici ; tapez '2 + 2' et appuyez sur Entrée.

Voilà, vous êtes prêt à vous lancer. Passez à la section “Conclusion” du cours.

## Installation sur macOS

### Télécharger et installer R

Si vous travaillez sur macOS, suivez les étapes ci-dessous pour télécharger et installer R :

1. Rendez-vous sur [cran.rstudio.com](https://cran.rstudio.com) pour accéder à la page d'installation de R. Cliquez ensuite sur le lien pour macOS :

The Comprehensive R Archive Network

#### Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- [Download R for Linux \(Debian, Fedora/Redhat, Ubuntu\)](#)
- [Download R for macOS](#)
- [Download R for Windows](#)

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

2. Téléchargez et installez la version de R correspondant à votre Mac. Pour la plupart des gens, c'est la première option de la rubrique “Dernière version” qu'il faudra choisir.

Latest release:

For Apple silicon (M1/M2) Macs:  
[R-4.3.1-arm64.pkg](#)

Dernière version pour M1/M2 Mac

For older Intel Macs:  
[R-4.3.1-x86\\_64.pkg](#)

Dernière version pour Intel Mac

**R 4.3.1** binary for macOS 11 (**Big Sur**) and higher, signed and notarized packages.

Contains R 4.3.1 framework, R.app GUI 1.79, Tcl/Tk 8.6.12 X11 libraries and Texinfo 6.8. The latter two components are optional and can be omitted when choosing "custom install", they are only needed if you want to use the `tcltk` R package or build package documentation from sources.

macOS Ventura users: there is a known bug in Ventura preventing installations from some locations without a prompt. If the installation fails, move the downloaded file away from the `Downloads` folder (e.g., to your home or Desktop)

Note: the use of X11 (including `tcltk`) requires [XQuartz](#) (version 2.8.5 or later). Always re-install XQuartz when upgrading your macOS to a new major version.

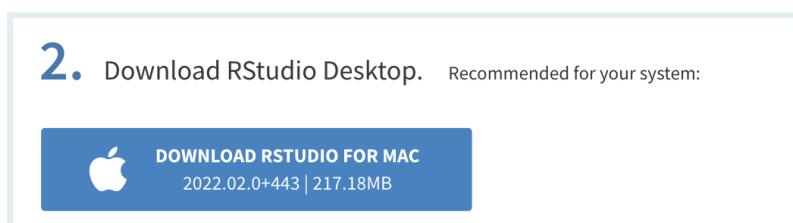
This release uses Xcode 14.2/14.3 and GNU Fortran 12.2. If you wish to compile R packages which contain Fortran code, you may need to download the corresponding GNU Fortran compiler from <https://mac.R-project.org/tools>. Any external libraries and tools are expected to live in `/opt/R/arm64` (Apple silicon) or `/opt/R/x86_64` (Intel).

3. Une fois le téléchargement terminé, cliquez sur le fichier téléchargé, puis suivez les instructions de la fenêtre d'installation.

Voilà, c'est fait ! Vous devriez maintenant avoir R sur votre ordinateur. Mais vous n'aurez probablement jamais besoin d'interagir avec R directement. Vous utiliserez plutôt l'IDE RStudio pour travailler avec R. Suivez les instructions de la section suivante pour obtenir RStudio.

### Télécharger, installer et exécuter RStudio

Pour télécharger RStudio, rendez-vous sur [rstudio.com/products/rstudio/download/#download](https://rstudio.com/products/rstudio/download/#download) et téléchargez la version pour macOS.

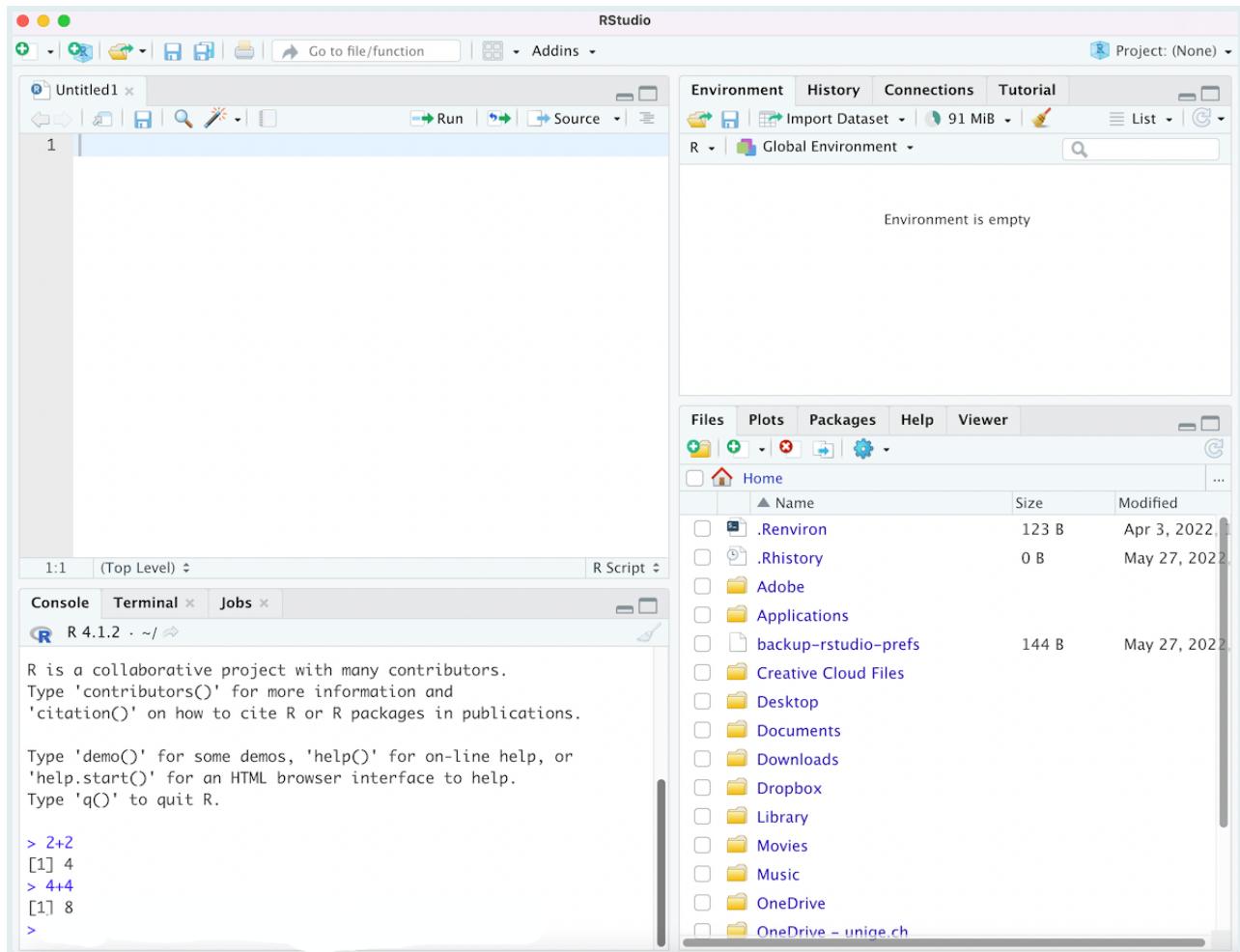


Une fois le téléchargement terminé, cliquez sur le fichier téléchargé et suivez les instructions d'installation.

Une fois installé, RStudio peut être ouvert comme n'importe quelle application sur votre ordinateur : appuyez sur "Commande" + "Espace" pour ouvrir Spotlight, puis recherchez "rstudio". Cliquez pour ouvrir l'application.

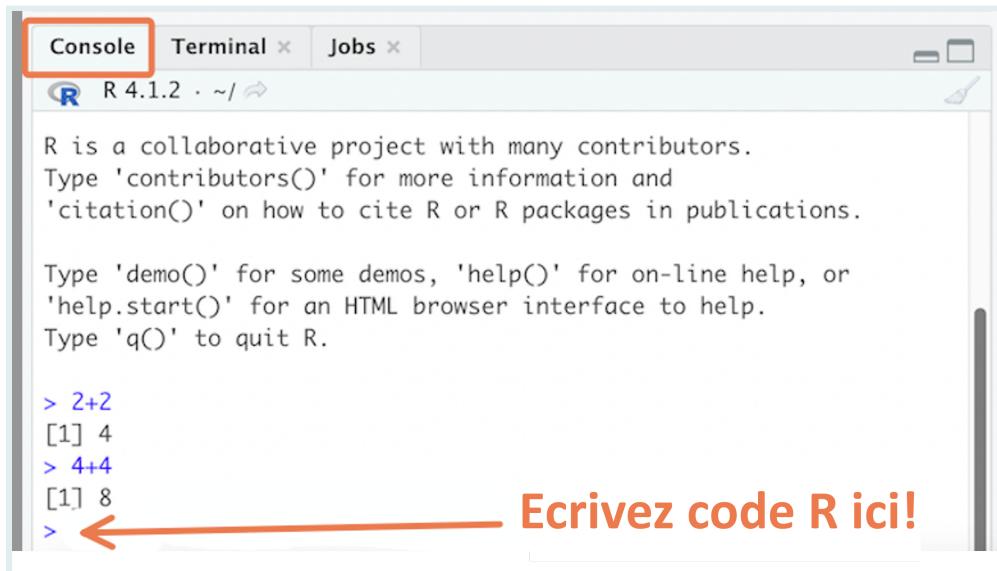


Vous devriez voir une fenêtre comme celle-ci :



Voici RStudio, votre nouvelle maison pour longtemps !

Vous pouvez commencer à utiliser R en saisissant du code dans l'onglet "console" à gauche :



Essayez d'utiliser R comme calculatrice ici ; tapez '2 + 2' et appuyez sur Entrée.

---

## Conclusion

Vous devriez maintenant avoir accès à R et RStudio, vous êtes donc prêt à commencer l'apprentissage de l'utilisation de ces outils extrêmement puissants. Rendez-vous à la prochaine session !

---

## Contributeurs

Les membres de l'équipe suivants ont contribué à cette leçon :



### KENE DAVID NWOSU

Data analyst, the GRAPH Network  
Passionate about world improvement



### LAMECK AGASA

Statistician/Data Scientist



### MICHAL SHRESTHA

Global Health Researcher, the GRAPH Network  
An advocate of health equity & justice through equal access to health data



### ELTON MUKONDA

Data analyst, the GRAPH Network  
A data enthusiast with a passion for population health research



### LAURE NGUEMO

Data Science Education Officer  
Gets very excited at the mention of data, especially health related data



### SABINA RODRIGUEZ VELÁSQUEZ

Project Manager and Scientific Collaborator, The GRAPH Network  
Infectiously enthusiastic about microbes and Global Health

---

## Références

Certains éléments de cette leçon ont été adaptés à partir des sources suivantes :

- Nordmann, Emily et Heather Cleland-Woods. *Chapitre 2 Principes de base de la programmation / Compétences en données.* psyteachr.github.io, <https://psyteachr.github.io/data-skills-v1/programming-basics.html> Consulté le 23 février 2022.

This work is licensed under the [Creative Commons Attribution Share Alike](#) license.



# Notes de leçon | Utilisation de RStudio

**GRAPH Network & OMS, soutenu par le Fonds Mondial**

**January 2024**

Ce cours a été créé par le Réseau GRAPH, une organisation à but non lucratif basée à l'Institut de santé globale de l'Université de Genève, en collaboration avec l'Organisation mondiale de la Santé, dans le cadre d'une subvention du Fonds mondial pour créer des cours afin de renforcer les capacités nationales en matière d'analyse épidémiologique.



Objectifs d'apprentissage	.....
Introduction	.....
Les volets de RStudio	.....
Source/Editeur	.....
Console	.....
Environment	.....
History	.....
Files	.....
Plots	.....
Packages	.....
Viewer	.....
Help	.....
Palette de commandes	.....
Conclusion	.....
Autres ressources	.....
Références	.....

---

## Objectifs d'apprentissage

1. Vous pouvez identifier et utiliser les onglets suivants dans RStudio : Source, Console, Environment, History, Files, Plots, Packages, Help et Viewer
2. Vous pouvez modifier les options d'interface de RStudio en fonction de vos besoins.

---

## Introduction

Maintenant que vous avez accès à R et RStudio, nous allons rapidement passer en revue l'interface de RStudio, votre maison numérique pour très longtemps.

Nous allons couvrir beaucoup de choses rapidement. Ne paniquez pas. On ne s'attend pas à ce que vous vous souveniez de tout cela. Au contraire, vous reverrez ces sujets encore et encore tout au long du cours, et vous les assimilerez naturellement de cette façon.

Vous pouvez également vous référer à cette leçon au fur et à mesure de votre progression.

Le but ici est simplement de vous faire connaître les outils à votre disposition au sein de RStudio.

---

Pour commencer, vous devez ouvrir l'application RStudio :

- Si vous travaillez avec RStudio Cloud, allez sur [rstudio.cloud](https://rstudio.cloud), connectez-vous, puis cliquez sur le projet “r\_intro” que vous avez créé dans la dernière leçon. (Si vous ne le voyez pas, créez simplement un nouveau projet R en utilisant l’icône “New project” en haut à droite).
- Si vous travaillez sur votre ordinateur local, allez dans votre dossier d’applications et double-cliquez sur l’icône RStudio. Vous pouvez également rechercher cette application dans le menu Démarrer (Windows) ou dans Spotlight (Mac).

---

## Les volets de RStudio

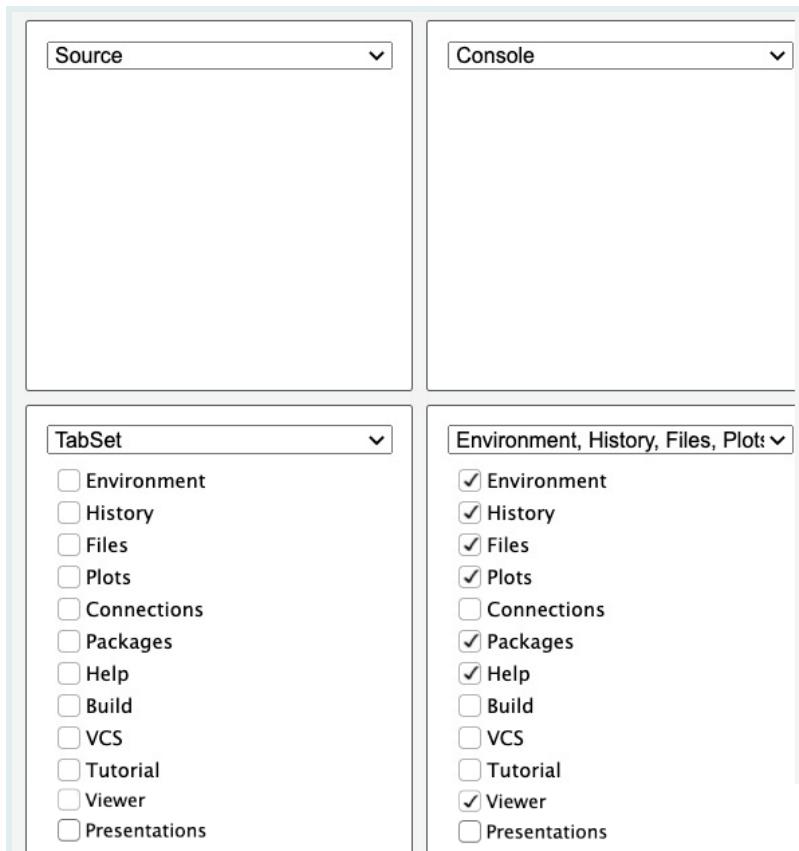
Par défaut, RStudio est organisé en quatre volets.

Si vous ne voyez que trois volets, ouvrez un nouveau script avec **File > New File > R Script**. Cela devrait faire apparaître un volet supplémentaire.



Avant d’aller plus loin, nous allons réorganiser ces volets pour améliorer la convivialité de l’interface.

Pour ce faire, dans le menu RStudio en haut de l’écran, sélectionnez **Tools > Global Options** pour afficher les options de RStudio. Ensuite, sous **Pane Layout**, ajustez la disposition des volets. La disposition que nous recommandons est illustrée ci-dessous.



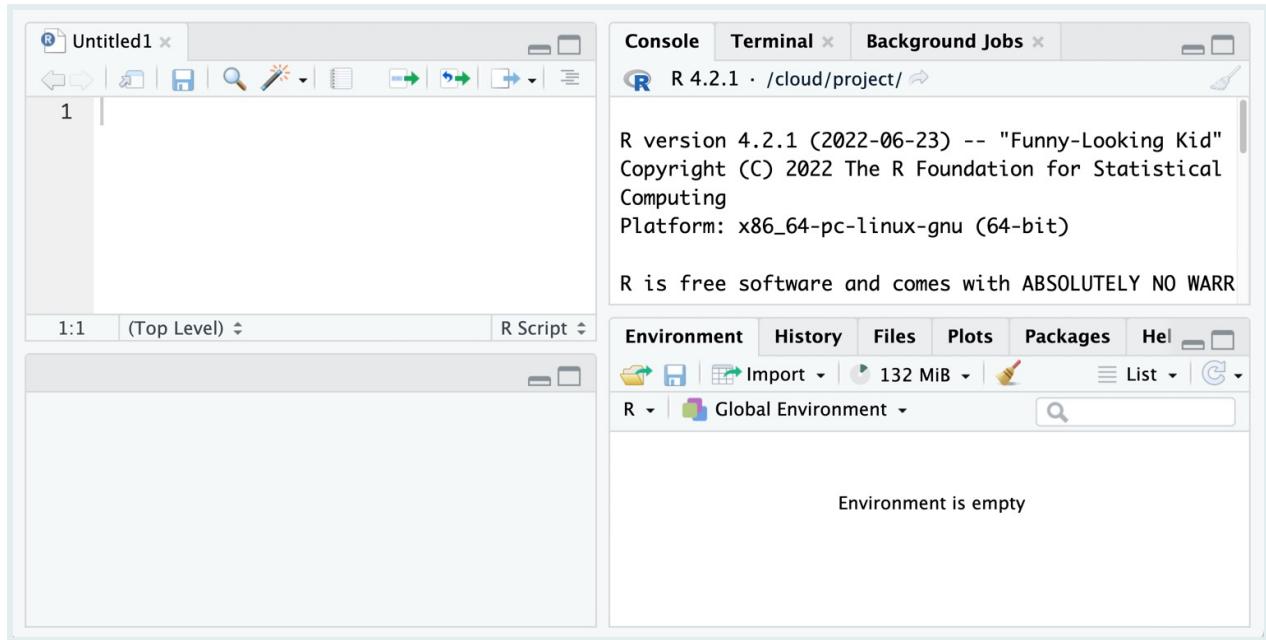
Dans le volet supérieur gauche se trouve l'onglet Source et dans le volet supérieur droit, vous devriez avoir l'onglet Console.

Ensuite, dans le volet inférieur gauche, aucune option d'onglet ne doit être cochée – cette section doit être laissée vide, la liste déroulante indiquant simplement "TabSet".

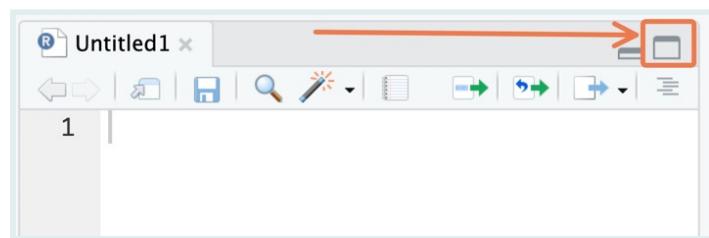
Enfin, dans le volet inférieur droit, vous devez cocher les onglets suivants : Environment, History, Files, Plots, Packages, Help et Viewer.

---

Génial, vous devriez maintenant avoir une fenêtre RStudio qui ressemble à ceci :



Le volet supérieur gauche est l'endroit où vous ferez la plupart du codage. Agrandissez-le en cliquant sur son icône d agrandissement :



Notez que vous pouvez faire glisser la barre qui sépare les volets de la fenêtre pour les redimensionner.



Examinons maintenant chacun des onglets de RStudio un par un. Vous trouverez ci-dessous une image récapitulative de ce dont nous allons discuter :

The screenshot shows the RStudio interface with the following panes:

- Editeur**: Ecrivez et sauvegardez code en scripts. Contains the script `rstudio\_intro.R` with code:

```

1 print("excited for R")
2 print("and RStudio")
3
4 View(women)
5
6 ebola_data <- read.csv("https://tinyurl.com/ebol")
7
8 plot(women)
9

```
- Console**: Exécutez code. Shows the output of the script:

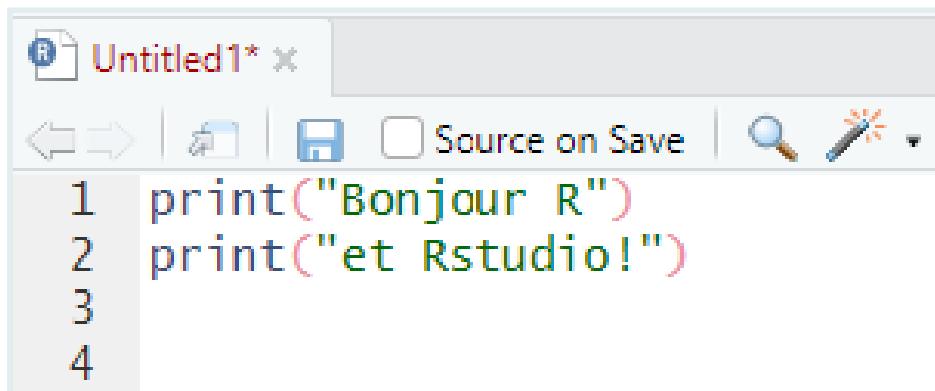
```

[1] "and RStudio"
> View(women)
> 2 + 2
[1] 4
> 3 + 3
[1] 6
> ebola_data <- read.csv("https://tinyurl.com/eboa-data-sample")
> plot(women)
>

```
- Environnement**: Voyez jeux de données & autres objets. Shows the `ebola\_data` dataset:

		200 obs. of 7 variables
\$ id	:	int 167 129 2...
\$ age	:	num 55 41 12 ...
- Historique**: Voyez et cherchez dernières commandes. Shows the history of commands run in the console.
- Paquets**: Installez et chargez paquets. Shows the System Library with packages like `base`, `boot`, and `class`.
- Fichiers**: Interagissez avec les fichiers de votre ordinateur. Shows the project directory structure.
- Aide**: Consultez documentation R. Shows information about the `women` dataset.

## Source/Éditeur



La source ou l'éditeur est l'endroit où se trouvent vos “scripts” R. Un script est un document texte dans lequel vous écrivez et enregistrez du code.

Étant donné que c'est là que vous effectuerez la majeure partie de votre codage, il est important que vous disposiez de beaucoup d'espace visuel. C'est pourquoi nous avons réorganisé la disposition du volet RStudio ci-dessus – afin de donner plus d'espace à l'éditeur.

Voyons maintenant comment utiliser cet éditeur.

---

Tout d'abord, **ouvrez un nouveau script** dans le menu File si vous n'en avez pas encore ouvert un: File > New File > R Script. Dans le script, tapez ce qui suit :

```
print("Bonjour R")
```

Pour **exécuter le code**, placez votre curseur n'importe où dans le code, puis appuyez sur "Commande" + "Entrée" sous macOS ou sur "Contrôle" + "Entrée" sous Windows.

Cela devrait envoyer le code à la console et l'exécuter.

---

Vous pouvez également **exécuter plusieurs lignes à la fois**. Pour essayer cela, ajoutez une deuxième ligne à votre script, de sorte qu'il se lise maintenant comme suit :

```
print("Bonjour R")
print("et RStudio!")
```

Faites maintenant glisser votre curseur pour mettre en surbrillance les deux lignes et appuyez sur Commande/Contrôle + Entrée.

Pour **exécuter le script entier**, vous pouvez utiliser Commande/Contrôle + A pour sélectionner tout le code, puis appuyez sur Commande/Contrôle + Entrée. Essayez ceci maintenant. Désélectionnez votre code, puis essayez le raccourci pour tout sélectionner.

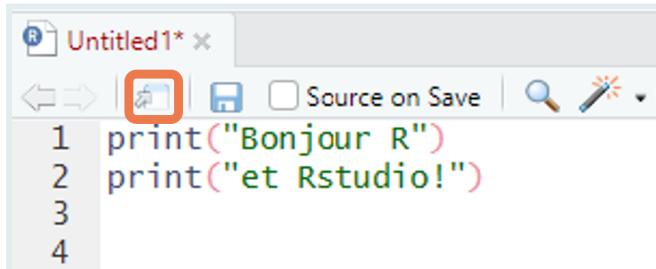
Il y a aussi un bouton Run en haut à droite de l'onglet source (



), qui permet d'exécuter le code (soit la ligne en cours, soit tout le code surligné). Mais vous devriez essayer d'utiliser le raccourci clavier à la place.

---

Pour **ouvrir le script dans une nouvelle fenêtre**, cliquez sur la troisième icône dans la barre d'outils directement au-dessus du script.



Pour remettre la fenêtre en place, cliquez sur le même bouton sur la fenêtre désormais externe.

---

Ensuite, **enregistrez le script**. Appuyez sur Commande/Contrôle + S pour faire apparaître la boîte de dialogue Enregistrer. Donnez-lui un nom de fichier comme "rstudio\_intro".

- Si vous travaillez avec RStudio Cloud, le fichier sera enregistré dans votre dossier de projet.
  - Si vous travaillez sur votre ordinateur local, enregistrez le fichier dans un endroit facile à localiser de votre ordinateur, par exemple votre bureau. (Plus tard, nous réfléchirons à la “bonne” façon d’organiser et de stocker les scripts).
- 

Vous pouvez **afficher des jeux de données** (qui sont comme des feuilles de calcul dans R) dans le même volet. Pour observer cela, saisissez et exécutez le code ci-dessous sur une nouvelle ligne de votre script :

```
View(women)
```

Remarquez le “V” majuscule dans `View()`.

	height	weight
1	58	115
2	59	117
3	60	120
4	61	123
5	62	126

`women` est le nom d’une trame de données chargé avec R. Il donne les tailles et poids moyens des femmes américaines âgées de 30 à 39 ans.

Vous pouvez cliquer sur l’icône “x” à droite de l’onglet “women” pour fermer ce visualiseur de données.

---

## Console

La *console*, en bas à gauche, est l’endroit où le **code est exécuté**. Vous pouvez saisir le code directement ici, mais il ne sera pas enregistré.

Saisissez un bout de code au hazard (peut-être un calcul comme ‘3 + 3’) et appuyez sur ‘Entrée’.

The screenshot shows the RStudio Console window. At the top, there are tabs for 'Console', 'Terminal', and 'Jobs'. The 'Console' tab is active. Below the tabs, the R logo and the text 'R 4.1.3 · /cloud/project/' are displayed. The main area of the console shows the following text:

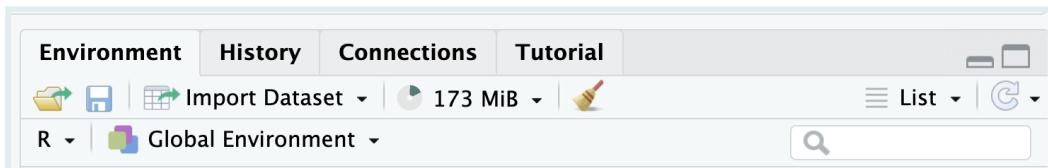
```
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> View(cars)
> 1 + 1
[1] 2
> 2 + 2
[1] 4
> 3 + 3
```

Si vous placez votre curseur sur la dernière ligne de la console et que vous appuyez sur la **flèche vers le haut**, vous pouvez revenir au dernier code exécuté. Continuez à appuyer dessus pour revenir aux lignes précédentes.

Pour exécuter l'une de ces lignes précédentes, appuyez sur *Entrée*.

## Environment



En haut à droite de la fenêtre RStudio, vous devriez voir l'onglet **Environment**.

L'onglet Environment affiche les jeux de données et autres objets qui sont chargés dans la mémoire de travail de R, ou "espace de travail".

Pour explorer cet onglet, importons une trame de données dans votre environnement à partir du Web. Saisissez le code ci-dessous dans votre script et exécutez-le :

```
ebola_data <- read.csv("https://tinyurl.com/ebola-data-sample")
```

Vous n'avez pas besoin de comprendre exactement ce que fait le code ci-dessus pour l'instant. Nous souhaitons simplement vous montrer rapidement les fonctionnalités de base de l'onglet Environment ; nous examinerons l'importation de données en détail plus tard.

De plus, si vous ne disposez pas d'un accès Internet actif, le code ci-dessus ne fonctionnera pas. Vous pouvez ignorer cette section et passer à l'onglet "History".

Vous avez maintenant importé le jeu de données et l'avez stocké dans un *objet* nommé ebola\_data. (Vous auriez pu nommer l'objet comme vous voulez.)

Maintenant que le jeu de données est stocké par R, vous devriez pouvoir le voir dans l'onglet Environment. Si vous cliquez sur l'icône déroulante bleue à côté du nom de l'objet dans l'onglet Environment, un résumé s'affiche.

The screenshot shows the RStudio interface with the Environment tab selected. In the Global Environment pane, there is a table with one row for the 'ebola\_data' object. The table has two columns: the first column contains the object name and its type, and the second column contains the object's structure. A red arrow points to the blue disclosure triangle icon next to the object name 'ebola\_data'.

ebola_data	200 obs. of 7 variables
\$ id : int	167 129 270 187 85 2
\$ age : num	55 41 12 NA 20 30 62
\$ sex : chr	"M" "M" "F" "F" ...
\$ status : chr	"confirmed" "confirm
\$ date_of_onset : chr	"2014-06-15" "2014-0
\$ date_of_sample: chr	"2014-06-21" "2014-0
\$ district : chr	"Kenema" "Kailahun"

Essayez de cliquer directement sur le jeu de données ebola\_data à partir l'onglet Environment. Cela l'ouvre dans un onglet "Viewer".

Vous pouvez **supprimer un objet de l'espace de travail** avec la fonction rm(). Tapez et exécutez ce qui suit dans une nouvelle ligne de votre script R.

```
rm(ebola_data)
```

Notez que l'objet ebola\_data n'apparaît plus dans votre environnement après avoir exécuté ce code.

L'icône du balai, en haut de l'onglet Environment, peut également être utilisée pour nettoyer votre espace de travail.



Pour vous entraîner à l'utiliser, essayez de réexécuter la ligne ci-dessus qui importe le jeu de données Ebola, puis effacez l'objet à l'aide de l'icône en forme de balai.

## History

Ensuite, l'onglet **History** affiche les commandes précédentes que vous avez exécutées.

```

Environment History Connections Tutorial
2 + 2 | To Console | To Source | X | Pencil | Search Bar
2 + 2
2 +
4
ebola_data <- read.csv("https://tinyurl.com/ebola-data-sample")
View(ebola_data)

```

Vous pouvez cliquer sur une ligne pour la mettre en surbrillance, puis l'envoyer à la console ou à votre script à l'aide des icônes "To Console" et "To Source" situées en haut de cet onglet.

Pour sélectionner plusieurs lignes, utilisez la méthode "Shift-clic": cliquez sur le premier élément que vous souhaitez sélectionner, puis maintenez la touche "Shift" enfoncée et cliquez sur le dernier élément que vous souhaitez sélectionner.

Enfin, vous remarquerez la présence d'une barre de recherche en haut à droite de l'onglet History, qui vous permet de rechercher les commandes passées que vous avez exécutées.

## Files

Ensuite, l'onglet **Files**. Il affiche les fichiers et les dossiers du dossier dans lequel vous travaillez.

Name	Size	Modified
rstudio_intro.R	219 B	Mar 18, 2022, 10:21 PM

Cet onglet vous permet d'interagir avec le système de fichiers de votre ordinateur.

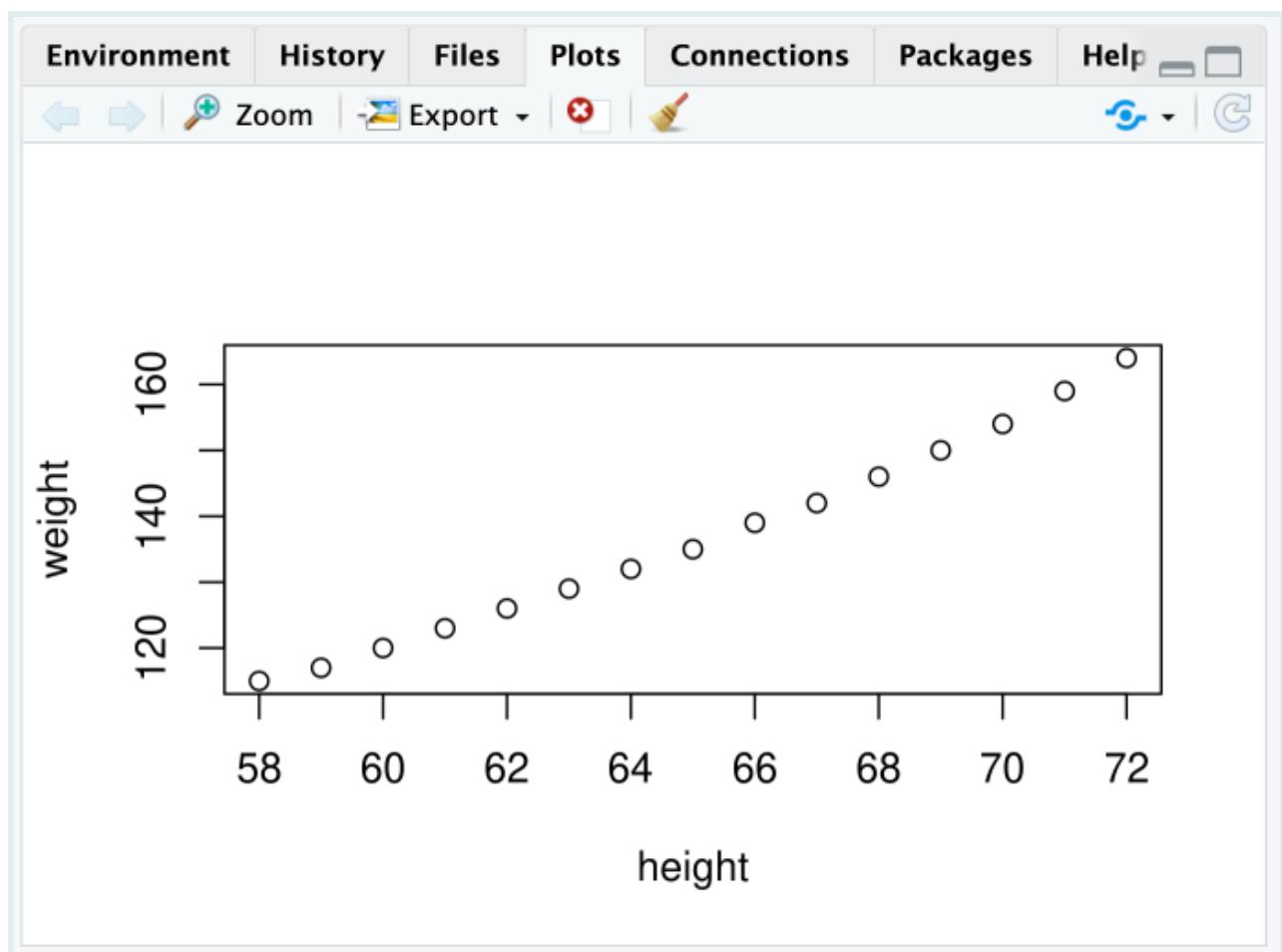
Essayez de jouer avec certains des boutons ici, pour voir ce qu'ils font. Vous devriez au moins essayer les options suivantes :

- Créer un nouveau dossier
- Supprimer ce dossier
- Créer un nouveau script R
- Renommer ce script

## Plots

Ensuite, l'onglet **Plots**. C'est ici que les figures ou graphiques générées par R apparaîtront. Essayez de créer un graphique simple avec le code suivant :

```
plot(women)
```

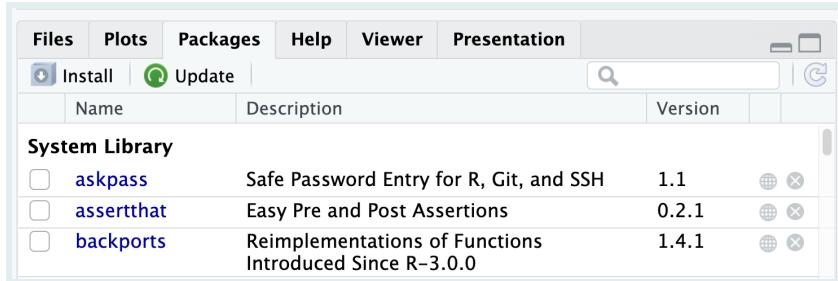


Ce code crée un graphique des deux variables dans le jeu de données "women". Vous devriez voir cette figure dans l'onglet Plots.

Maintenant, testez les boutons en haut de cet onglet pour explorer ce qu'ils font. En particulier, essayez d'exporter un graphique sur votre ordinateur.

## Packages

Regardons ensuite l'onglet **Packages**.

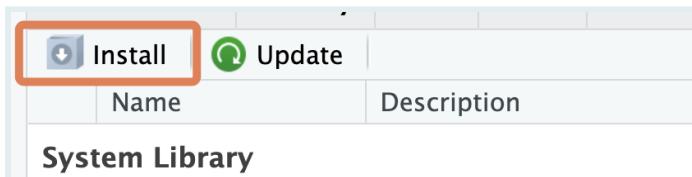


Les packages sont des collections de code R qui étendent les fonctionnalités de R. Nous aborderons les packages en détail dans une prochaine leçon.

Pour l'instant, il est important de savoir que pour utiliser un package, vous devez l'*installer* puis le *charger*. Les packages ne doivent être installés qu'une seule fois, mais doivent être chargés à chaque nouvelle session R.

Tous les noms de packages que vous voyez (en police bleue) sont des packages installés sur votre système. Les packages cochés sont des packages qui sont *chargés* dans la session en cours.

Vous pouvez installer un package avec le bouton Install de l'onglet Packages.



Mais il est préférable d'installer et de charger les packages avec le code R, plutôt qu'avec le bouton Install. Essayons ceci. Saisissez et exécutez le code ci-dessous pour installer le package {highcharter}.

```
install.packages("highcharter")
library(highcharter)
```

La première ligne installe le package. La deuxième ligne *charge* le package depuis votre bibliothèque de packages.

Comme vous n'avez besoin d'installer un package qu'une seule fois, vous pouvez maintenant supprimer la ligne d'installation de votre script.

---

Maintenant que le package {highcharter} a été installé et chargé, vous pouvez utiliser les fonctions qu'il contient. Pour essayer cela, saisissez et exécutez le code ci-dessous :

```
highcharter::hchart(women$weight)
```

Ce code utilise la *fonction* `hchart()` du package `{highcharter}` pour tracer un histogramme interactif montrant la distribution des poids dans le jeu de données `women`.

(Bien sûr, vous ne savez peut-être pas encore ce qu'est une fonction. Nous y reviendrons bientôt.)

## Viewer

Notez que l'histogramme ci-dessus apparaît dans un onglet **Viewer**. Cet onglet vous permet de prévisualiser les fichiers HTML et les objets interactifs.

## Help

Enfin, l'onglet **Help** affiche la documentation des différents objets R. Essayez de taper et d'exécuter chaque ligne ci-dessous pour voir à quoi ressemble cette documentation.

```
?hchart  
?women  
?read.csv
```

The screenshot shows the RStudio interface with the 'Viewer' tab selected in the top menu bar. Below the menu is a toolbar with icons for back, forward, search, and print. A search bar contains the text 'R: Create a highchart object from a particular data type'. The main content area displays the documentation for the `hchart` function from the `{highcharter}` package. The title is 'Create a highchart object from a particular data type'. The 'Description' section states: 'hchart uses highchart to draw a particular plot for an object of a particular class in a single command. This defines the S3 generic that other classes and packages can extend.'

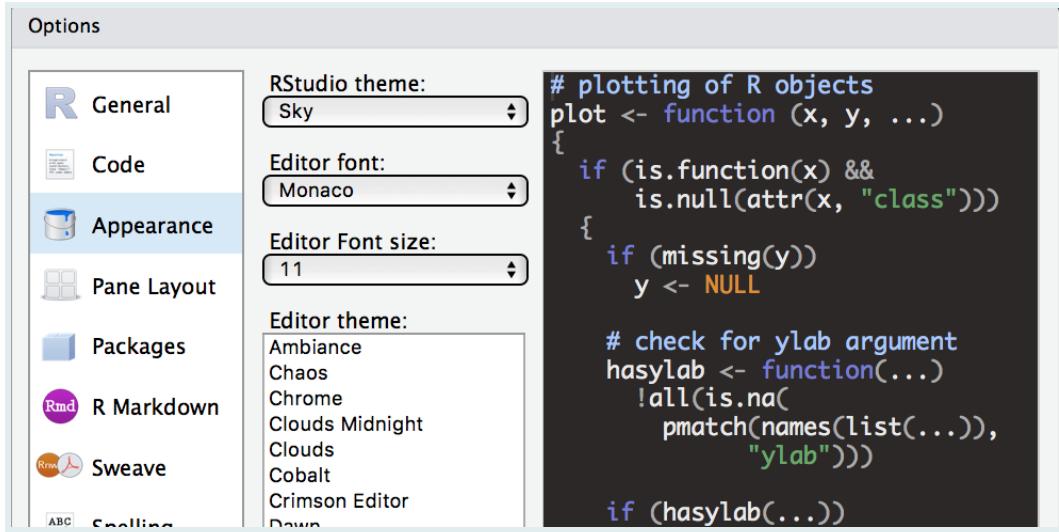
Les fichiers d'aide ne sont pas toujours très faciles à comprendre pour les débutants, mais avec le temps, ils deviendront plus utiles.

## #Options de RStudio

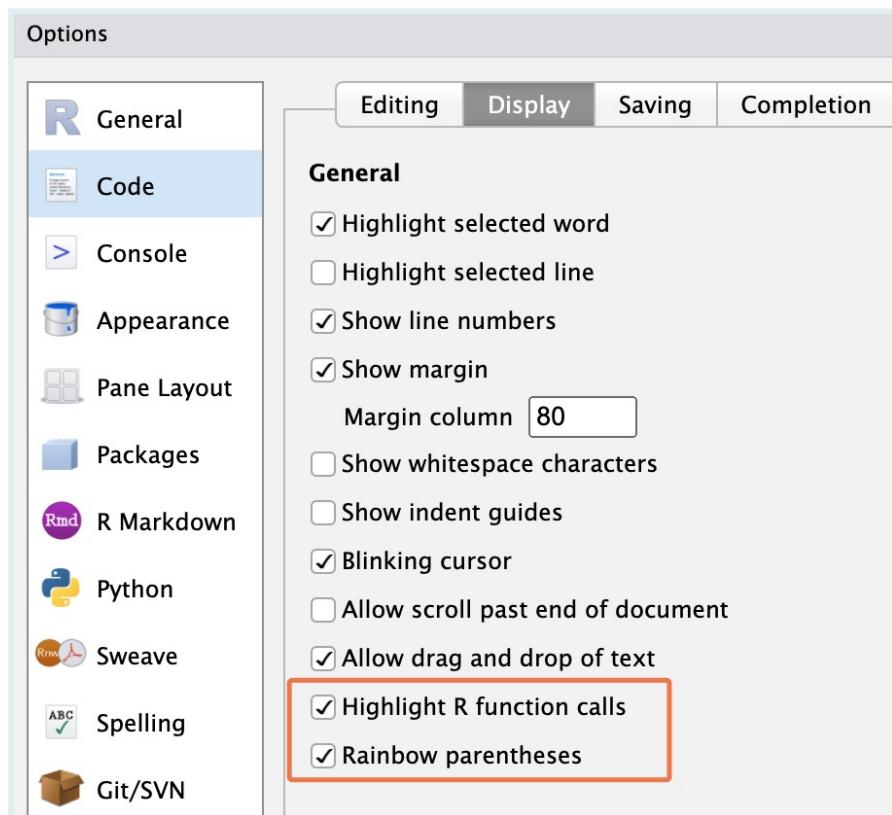
RStudio dispose d'un certain nombre d'options utiles pour modifier son apparence et ses fonctionnalités. Essayons-les. Il se peut que vous ne compreniez pas toutes les modifications apportées pour l'instant. Ce n'est pas grave.

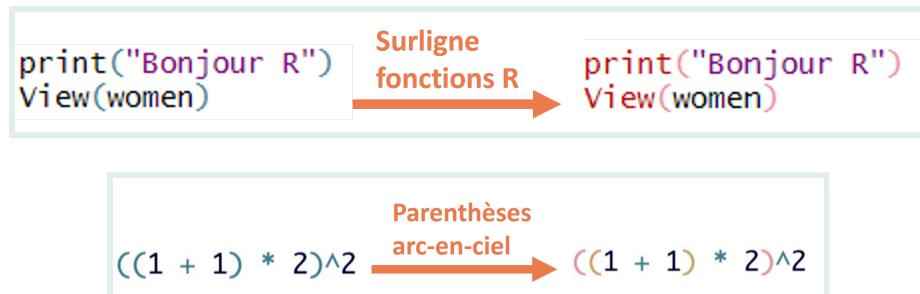
Dans le menu RStudio en haut de l'écran, sélectionnez Tools > Global Options pour faire apparaître les options de RStudio.

- Maintenant, sous "Appearance", choisissez votre thème idéal. (Nous aimons les thèmes "Crimson Editor" et "Tomorrow Night".)



- Sous Code > Display, cochez “Highlight R function calls”. Cela donne à vos *fonctions* R une couleur unique, améliorant ainsi la lisibilité. Vous comprendrez cela plus tard.
- Toujours sous Code > Display, cochez “Rainbow parentheses”. Cela rend vos “parenthèses imbriquées” plus faciles à lire en donnant à chaque paire une couleur unique.





- Enfin, sous General > Basic, **décochez** la case indiquant "**Restaurer .RData dans l'espace de travail au démarrage**". Vous ne souhaitez pas restaurer de données dans votre espace de travail (ou \* Environment)\* lorsque vous démarrez RStudio. Commencer avec un espace de travail propre à chaque fois est moins susceptible de conduire à des erreurs.

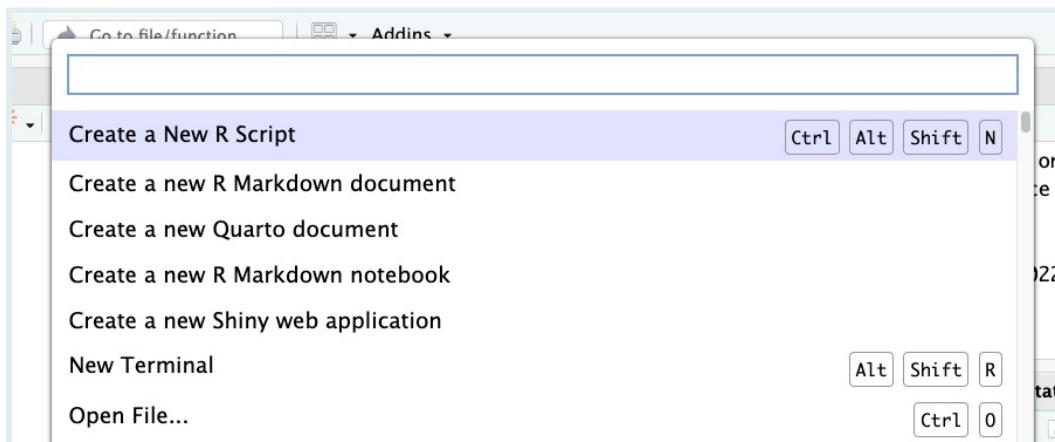
Cela signifie également que vous ne voulez jamais "**enregistrer votre espace de travail dans .RData à la sortie**", alors réglez-le sur **Jamais**.

## Palette de commandes

La palette de commandes Rstudio donne un accès instantané et consultable à de nombreuses options de menu et paramètres de RStudio que nous avons vus jusqu'à présent.

La palette peut être invoquée avec le raccourci clavier Ctrl + Shift + P (Cmd + Shift + P sur macOS).

Il est également disponible dans le menu *Tools* (*Tools* -> *Show Command Palette*).



Essayez de l'utiliser pour :

- Créez un nouveau script (Recherchez "New script" et cliquez sur l'option appropriée)

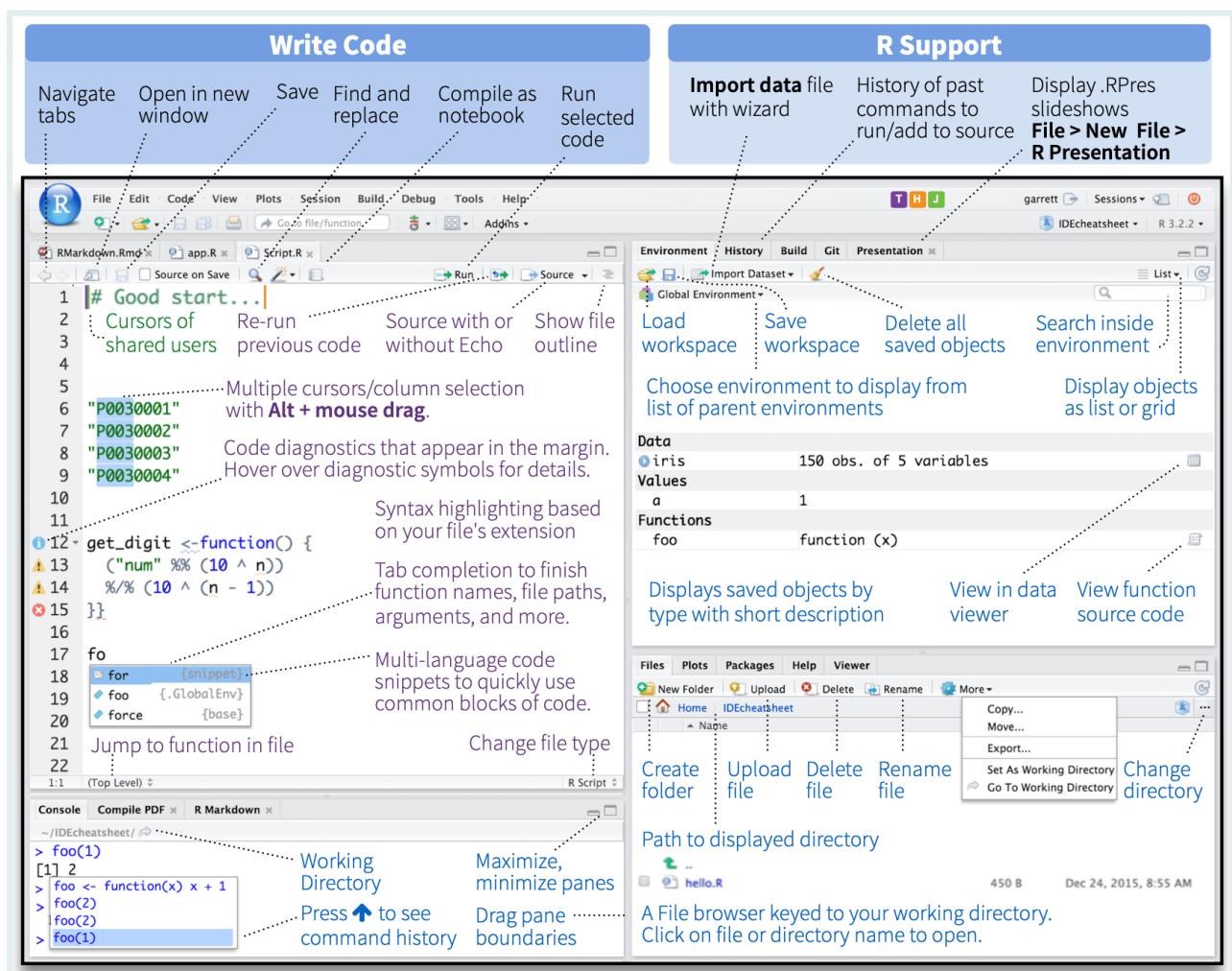
- Renommer un script (Recherchez "rename" et cliquez sur l'option appropriée)

## Conclusion

Toutes nos félicitations! Vous êtes maintenant un nouveau citoyen de RStudio.

Bien entendu, vous n'avez fait qu'effleurer la surface des fonctionnalités de RStudio. Au fur et à mesure que vous avancerez dans votre parcours R, vous découvrirez de nouvelles fonctionnalités et, nous l'espérons, vous aimerez de plus en plus ce merveilleux environnement de développement intégré (IDE) qu'est RStudio. Un bon point de départ est la fiche de contrôle officielle de l'IDE RStudio [cheatsheet](#).

Voici une section de cette fiche :



Rendez-vous dans la prochaine leçon!

---

## Autres ressources

1. 23 trucs, astuces et raccourcis de RStudio

---

## Contributeurs

Les membres de l'équipe suivants ont contribué à cette leçon :



### KENE DAVID NWOSU

Data analyst, the GRAPH Network  
Passionate about world improvement



### LAMECK AGASA

Statistician/Data Scientist



### LAURE NGUEMO

Data Science Education Officer  
Gets very excited at the mention of data, especially health related data



### SABINA RODRIGUEZ VELÁSQUEZ

Project Manager and Scientific Collaborator, The GRAPH Network  
Infectiously enthusiastic about microbes and Global Health

---

## Références

Certains éléments de cette leçon ont été adaptés à partir des sources suivantes :

- “Rstudio Cheatsheets.” *RStudio*, <https://www.rstudio.com/resources/cheatsheets/>.
- “Chapitre 1 Premiers pas : Compétences en matière de données pour une recherche reproductible.” *Chapitre 1 Mise en route / Compétences en données pour une recherche reproductible*, <https://psyteachr.github.io/reprores-v2/intro.html>.

This work is licensed under the [Creative Commons Attribution Share Alike](#) license.



# Notes de leçon | Les bases du codage

**GRAPH Network & OMS, soutenu par le Fonds Mondial**

**January 2024**

Ce cours a été créé par le Réseau GRAPH, une organisation à but non lucratif basée à l'Institut de santé globale de l'Université de Genève, en collaboration avec l'Organisation mondiale de la Santé, dans le cadre d'une subvention du Fonds mondial pour créer des cours afin de renforcer les capacités nationales en matière d'analyse épidémiologique.



Introduction .....
Commentaires .....
R comme une calculatrice .....
Mise en forme du code .....
Objets dans R .....
Créer un objet .....
Qu'est-ce qu'un objet ? .....
Les jeux de données sont aussi des objets .....
Renommer un objet .....
Écraser un objet .....
Travailler avec des objets .....
Quelques erreurs avec les objets .....
Nommer les objets .....
Les fonctions .....
Syntaxe de la fonction de base .....
L'imbrication des fonctions .....
Les packages .....
Un premier exemple : le package {tableone} .....
Signifiants complets .....
pacman::p_load() .....
Conclusion .....

---

## Objectifs d'apprentissage

1. Vous pouvez écrire des commentaires dans R.
2. Vous pouvez créer des en-têtes de section dans RStudio.
3. Vous savez utiliser R comme calculatrice.
4. Vous pouvez créer, écraser et manipuler des objets R.
5. Vous comprenez les règles de base pour nommer les objets R.
6. Vous comprenez la syntaxe pour appeler les fonctions R.
7. Vous savez comment imbriquer plusieurs fonctions.
8. Vous pouvez utiliser, installer et charger des compléments de packages R et appeler des fonctions à partir de ces packages.

---

## Introduction

Dans la dernière leçon, vous avez appris à utiliser RStudio, le merveilleux environnement de développement intégré (IDE) qui facilite grandement le travail avec R. Dans cette leçon, vous apprendrez les bases de l'utilisation de R lui-même.

Pour commencer, ouvrez RStudio et ouvrez un nouveau script avec **File > New File > R Script** dans le menu RStudio.



Ensuite, **enregistrez le script** avec **File > Save** dans le menu de RStudio ou en utilisant le raccourci Commande/Contrôle + S . Cela devrait faire apparaître la boîte de dialogue "Save file". Enregistrez le fichier avec un nom comme "coding\_basics".

Vous devez maintenant saisir tout le code de cette leçon dans ce script.

---

## Commentaires

Il existe deux principaux types de texte dans un script R : les commandes et les commentaires. Une commande est une ou plusieurs lignes de code R qui ordonne à R de faire quelque chose (par exemple `2 + 2`).

Un commentaire est un texte qui est ignoré par l'ordinateur.

Tout ce qui suit le symbole `#` (prononcé "dièse") sur une ligne donnée est un commentaire. Essayez de taper et d'exécuter le code ci-dessous pour voir ceci :

```
# Un commentaire
2 + 2 # Un autre commentaire
# 2 + 2
```

Puisqu'ils sont ignorés par l'ordinateur, les commentaires sont destinés aux *humains*. Ils vous aident, vous et les autres, à garder une trace de ce que fait votre code. Utilisez-les souvent ! Comme votre mère le dit toujours, "Tout excès nuit, sauf les commentaires R".

### Question 1

Vrai ou Faux : les deux morceaux de code ci-dessous sont des façons valables de commenter du code ?

```
# Additionnez deux nombres  
2 + 2
```

```
2 + 2 # Additionnez deux nombres
```

**Remarque:** Toutes les réponses aux questions se trouvent à la fin de la leçon.

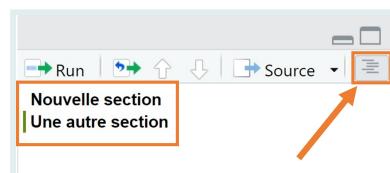
Une utilisation fantastique des commentaires consiste à séparer vos scripts en sections. Si vous mettez quatre tirets après un commentaire, RStudio créera une nouvelle section dans votre code :

```
# Nouvelle section ----
```

Cela présente deux avantages intéressants. Tout d'abord, vous pouvez cliquer sur la petite flèche située à côté de l'en-tête de la section pour replier ou réduire cette section de code :



Deuxièmement, vous pouvez cliquer sur l'icône « Outline » en haut à droite de l'Éditeur pour afficher et parcourir tout le contenu de votre script :



---

## R comme une calculatrice

R fonctionne comme une calculatrice et respecte l'ordre correct des opérations. Saisissez et exécutez les expressions suivantes et observez leur résultat :

```
2 + 2
```

```
## [1] 4
```

```
2 - 2
```

```
## [1] 0
```

```
2 * 2 # deux fois deux
```

```
## [1] 4
```

```
2 / 2 # deux divisé par deux
```

```
## [1] 1
```

```
2 ^ 2 # deux élevé à la puissance deux
```

```
## [1] 4
```

```
2 + 2 * 2 # ceci est évalué suivant l'ordre des opérations
```

```
## [1] 6
```

```
sqrt(100) # racine carrée
```

```
## [1] 10
```

La commande de racine carrée affichée sur la dernière ligne est un bon exemple de *fonction R*, où *100* est l'*argument* de la fonction. Vous verrez bientôt d'autres fonctions.

Nous espérons que vous vous souvenez du raccourci pour exécuter le code !

**REMINDER**



Pour **exécuter une seule ligne de code**, placez votre curseur n'importe où sur cette ligne, puis appuyez sur Commande + Entrée sur macOS, ou Contrôle + Entrée sur Windows.

Pour **exécuter plusieurs lignes**, faites glisser votre curseur pour surligner les lignes concernées, puis appuyez à nouveau sur Commande/Contrôle + Entrée.

## Question 2

Dans l'expression suivante, quel signe est évalué en premier par R, le moins ou la division ?

```
2 - 2 / 2
```

```
## [1] 1
```

---

## Mise en forme du code

R ne se soucie pas de la façon dont vous choisissez d'espacer votre code.

Pour les opérations mathématiques que nous avons effectuées ci-dessus, tout ce qui suit serait un code valide :

```
2+2
```

```
## [1] 4
```

```
2 + 2
```

```
## [1] 4
```

```
2
```

```
+
```

```
2
```

```
## [1] 4
```

De même, pour la fonction `sqrt()` utilisée ci-dessus, n'importe lequel de ces codes serait valide :

```
sqrt(100)
```

```
## [1] 10
```

```
sqrt( 100 )
```

```
## [1] 10
```

```
# vous pouvez même espacer la commande sur plusieurs lignes
sqrt(
  100
)
```

```
## [1] 10
```

Mais bien sûr, vous devriez essayer d'espacer votre code de manière raisonnable. Qu'entend-on exactement par "raisonnable" ? Il peut être difficile pour vous de le savoir pour le moment. Avec le temps, en lisant le code d'autres personnes, vous apprendrez qu'il existe certaines *conventions R* pour l'espacement et la mise en forme du code.

En attendant, vous pouvez demander à RStudio de vous aider à formater votre code. Pour ce faire, mettez en évidence une section de code que vous voulez reformater, et, dans le menu de RStudio, allez à **Code > Reformat Code**, ou utilisez le raccourci Maj + Commande/Contrôle + A.

En attendant, vous pouvez demander à RStudio de vous aider à structurer votre code. Pour ce faire, mettez en surbrillance n'importe quelle section de code que vous souhaitez restructurer et, dans le menu de RStudio, accédez à **Code > Reformat Code**, ou utilisez le raccourci Maj + Commande/Contrôle + A.

### Coincé sur le signe +

Si vous exécutez une ligne de code incomplète, R affichera un signe "+" pour indiquer qu'il attend que vous terminiez le code.

Par exemple, si vous exécutez le code suivant :

#### WATCH OUT



```
sqrt(100
```

vous n'obtiendrez pas la sortie que vous attendez (10). La console affichera plutôt sqrt( et un signe + :

```
> sqrt(100
+
```

R attend que vous complétez la parenthèse fermante. Vous pouvez compléter le code et vous débarrasser du "+" en saisissant simplement la parenthèse manquante :

```
)
```

### WATCH OUT



```
> sqrt(100)
+ )
[1] 10
```

Vous pouvez également appuyer sur la touche d'échappement “`ESC`” pendant que votre curseur est dans la console pour recommencer.

## Objets dans R

### Créer un objet

Lorsque vous exécutez du code comme nous l'avons fait ci-dessus, le résultat de la commande (ou sa *valeur*) est simplement affiché dans la console – il n'est stocké nulle part.

```
2 + 2 # R imprime ce résultat, 4, mais ne le stocke pas
```

```
## [1] 4
```

Pour stocker une valeur pour une utilisation future, attribuez-la à un *objet* à l'aide de l'*opérateur d'attribution*, `<-` :

```
mon_obj <- 2 + 2 # attribue le résultat de `2 + 2` à l'objet appelé `mon_obj`
mon_obj # imprime mon_obj
```

```
## [1] 4
```

L'*opérateur d'attribution*, `<-`, est composé du signe ‘inférieur à’, `<`, et d'un signe moins, `-`. Vous l'utiliserez des milliers de fois au cours de votre vie R, alors ne le saisissez pas manuellement ! Utilisez plutôt le raccourci de RStudio, **alt + -** (**alt ET moins**) sous Windows ou **option + -** (**option ET moins**) sur macOS.

### SIDE NOTE



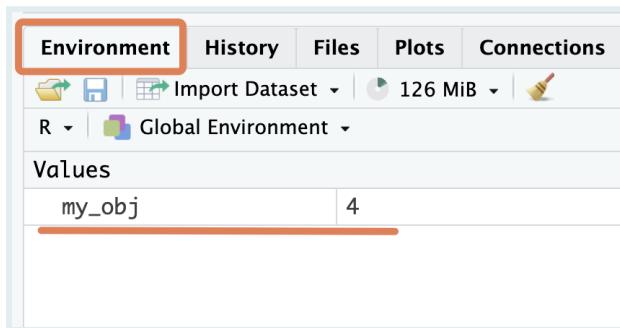
Notez également que vous pouvez utiliser le signe *égal*, `=`, pour l'attribution.

```
mon_obj = 2 + 2
```

**SIDE NOTE**

également. Suivez la convention et utilisez <-.

Maintenant que vous avez créé l'objet `mon_obj`, R sait tout à son sujet et en gardera la trace pendant cette session R. Vous pouvez voir tous les objets créés dans l'onglet *Environment* de RStudio.

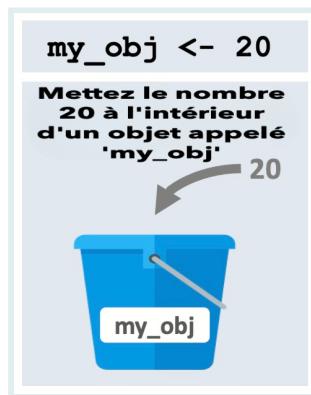


### Qu'est-ce qu'un objet ?

Alors qu'est-ce qu'un objet exactement ? Considérez-le comme un seau nommé qui peut contenir n'importe quoi. Lorsque vous exécutez le code ci-dessous :

```
mon_obj <- 20
```

vous dites à R, "mettez le nombre 20 dans un seau nommé 'mon\_obj'".



Une fois le code exécuté, nous dirions, en termes R, que "la valeur de l'objet appelé `mon_obj` est 20".

Et si vous exécutez ce code :

```
prenom <- "Joanna"
```

vous demandez à R de "mettre la valeur 'Joanna' dans le seau appelé `prenom`.

```
first_name <- "Joanna"
```

**Mettez la valeur  
"Joanna" dans  
un objet appelé  
'first\_name'**



Une fois le code exécuté, nous dirions, en termes R, que "la valeur de l'objet prenom est Joanna".

Notez que R évalue le code \* avant \* de le placer dans le seau.

Donc, avant l'exécution de ce code,

```
mon_obj <- 2 + 2
```

R effectue d'abord le calcul de  $2 + 2$ , puis stocke le résultat, 4, à l'intérieur de l'objet.

```
my_obj <- 2 + 2
```

Évaluez  $2 + 2$  puis stockez le résultat dans un objet appelé 'my\_obj'



### Question 3

Considérez le morceau de code ci-dessous :

```
resultat <- 2 + 2 + 2
```

Quelle est la valeur de l'objet resultat créé ?

- A.  $2 + 2 + 2$
- B. numérique
- C. 6

## Les jeux de données sont aussi des objets

Jusqu'à présent, vous avez travaillé avec des objets très simples. Vous pensez peut-être "Où sont les feuilles de calcul et les jeux de données ? Pourquoi écrivons-nous `mon_obj` `<- 2 + 2` ? Est-ce un cours de mathématiques à l'école primaire ?!"

Soyez patient.

Voyons maintenant un aperçu de cela. Saisissez le code ci-dessous pour télécharger un jeu de données sur les cas d'Ebola que nous avons stocké sur Google Drive et placez-le dans l'objet `ebola_sierra_leone_data`.

```
ebola_sierra_leone_data <- read.csv("https://tinyurl.com/ebola-data-sample")
ebola_sierra_leone_data # Affiche ebola_data
```

```
##   id age sex   status date_of_onset date_of_sample district
## 1 167  55   M confirmed    2014-06-15    2014-06-21 Kenema
## 2 129  41   M confirmed    2014-06-13    2014-06-18 Kailahun
## 3 270  12   F confirmed    2014-06-28    2014-07-03 Kailahun
## 4 187   NA   F confirmed    2014-06-19    2014-06-24 Kailahun
## 5  85  20   M confirmed    2014-06-08    2014-06-24 Kailahun
```

Ces données contiennent un échantillon d'informations sur les patients de l'épidémie d'Ebola de 2014 à 2016 en Sierra Leone.

---

Étant donné que vous pouvez stocker des jeux de données en tant qu'objets, il est très facile de travailler avec plusieurs jeux de données en même temps.

Ci-dessous, nous importons et visualisons un autre jeu de données à partir du Web :

```
diabete_chine <- read.csv("https://tinyurl.com/diabetes-china")
```

Étant donné que le jeu de données ci-dessus est assez volumineux, il peut être utile de le consulter dans le visualiseur de données :

```
View(diabete_chine)
```

Notez que les deux jeux de données apparaissent maintenant dans votre onglet *Environment*.

### SIDE NOTE



Plutôt que de lire les données à partir d'un lecteur Internet comme nous l'avons fait ci-dessus, il est plus probable que vous ayez les données sur votre ordinateur et que vous souhaitiez les lire dans R à partir de là. Nous aborderons ce point dans une prochaine leçon.

#### SIDE NOTE



Plus tard dans le cours, nous vous montrerons également comment stocker et lire des données à partir d'un service Web comme Google Drive, ce qui est pratique pour faciliter la portabilité.

## Renommer un objet

Vous souhaitez parfois renommer un objet. Il n'est pas possible de le faire directement.

Pour renommer un objet, faites une copie de l'objet avec un nouveau nom et supprimez l'original.

Par exemple, nous pouvons décider que le nom de l'objet `ebola_sierra_leone_data` est trop long. Pour le remplacer par le nom plus court “`ebola_data`” et exécuter :

```
ebola_data <- ebola_sierra_leone_data
```

Cela a copié le contenu du `seau ebola_sierra_leone_data` vers un nouveau `seau ebola_data`.

Vous pouvez maintenant vous débarrasser de l'ancien `seau ebola_sierra_leone_data` avec la fonction `rm()`, qui signifie “supprimer”:

```
rm(ebola_sierra_leone_data)
```

## Écraser un objet

Écraser un objet revient à changer le *contenu* d'un *seau*.

Par exemple, nous avons précédemment exécuté ce code pour stocker la valeur “`Joanna`” dans l'objet `prenom`:

```
prenom <- "Joanna"
```

Pour changer cette valeur, réexécutez simplement la ligne avec une valeur différente :

```
prenom <- "Luigi"
```

Vous pouvez jeter un coup d'œil à l'onglet Environment pour observer le changement.

## Travailler avec des objets

Vous passerez la plupart de votre temps dans R à manipuler des objets R. Voyons quelques exemples rapides.

Vous pouvez exécuter des commandes simples sur les objets. Par exemple, ci-dessous, nous stockons la valeur `100` dans un objet, puis nous prenons la racine carrée de l'objet :

Vous pouvez exécuter des commandes simples sur des objets. Par exemple, ci-dessous, nous stockons la valeur 100 dans un objet, puis nous prenons la racine carrée de l'objet :

```
mon_nombre <- 100  
sqrt(mon_nombre)
```

```
## [1] 10
```

R “voit” `mon_nombre` comme le nombre 100, et est donc capable d’évaluer sa racine carrée.

Vous pouvez également combiner des objets existants pour en créer de nouveaux. Par exemple, saisissez le code ci-dessous pour ajouter `mon_nombre` à lui-même et stockez le résultat dans un nouvel objet appelé `ma_somme` :

```
ma_somme <- mon_nombre + mon_nombre
```

Quelle devrait être la valeur de `ma_somme`? Essayez d’abord de deviner, puis vérifiez.



**SIDE NOTE** Pour vérifier la valeur d’un objet, tel que `ma_somme`, vous pouvez saisir et exécuter uniquement le code `ma_somme` dans la console ou l’éditeur. Alternativement, vous pouvez simplement mettre en surbrillance la valeur `ma_somme` dans le code existant et appuyer sur Commande/Contrôlée + Entrée.

Mais bien sûr, la plupart de votre analyse impliquera de travailler avec des objets *données*, tels que l’objet `ebola_data` que nous avons créé précédemment.

Voyons un exemple très simple de la façon d’interagir avec un objet de données ; nous l’aborderons plus en détail dans la prochaine leçon.

Pour obtenir un tableau de la répartition des patients par sexe dans l’objet `ebola_data`, nous pouvons exécuter ce qui suit :

```
table(ebola_data$sex)
```

```
##  
##   F     M  
## 124   76
```

Le symbole du dollar, \$, ci-dessus nous a permis de créer un sous-ensemble dans une colonne spécifique.

#### Question 4

- a. Considérez le code ci-dessous. Quelle est la valeur de l'objet reponse ?

```
huit <- 9  
reponse <- huit - 8
```

- b. Utilisez `table()` pour créer un tableau montrant la répartition des patients dans les districts dans l'objet `ebola_data`.

#### Quelques erreurs avec les objets

```
prenom <- "Luigi"  
nom <- "Fenway"
```

```
nom_complet <- prenom + nom
```

Erreur dans `prenom + nom` : argument non numérique à l'opérateur binaire

Le message d'erreur vous indique que ces objets ne sont pas des nombres et ne peuvent donc pas être additionnés avec `+`. Il s'agit d'un type d'erreur assez courant, causé par la tentative de faire des choses inappropriées à vos objets. Soyez prudent à ce sujet.

Dans ce cas particulier, nous pouvons utiliser la fonction `paste()` pour assembler ces deux objets :

```
nom_complet <- paste(prenom, nom)  
nom_complet
```

```
## [1] "Luigi Fenway"
```

---

Une autre erreur que vous obtiendrez souvent est Erreur : l'objet 'XXX' est introuvable. Par exemple:

```
mon_nombre <- 48 # définit `mon_nombre`  
Mon_nombre + 2 # essaie d'ajouter 2 à `Mon_nombre`
```

```
Erreur : objet 'Mon_nombre' introuvable
```

Ici, R renvoie un message d'erreur car nous n'avons pas encore créé (ou *défini*) l'objet `Mon_nombre`. (Rappelez-vous que R est sensible aux majuscules et minuscules.)

Lorsque vous commencez à apprendre R, gérer les erreurs peut être frustrant. Elles sont souvent difficiles à comprendre (par exemple, que signifie exactement “*argument non numérique à l'opérateur binaire*” ?).

Essayez de chercher sur Google tous les messages d'erreur que vous obtenez et parcourez les premiers résultats. Cela vous mènera à des forums (par exemple [stackoverflow.com](https://stackoverflow.com)) où d'autres apprenants de R se sont plaints de la même erreur. Vous y trouverez des explications et des solutions à vos problèmes.

## Question 5

a. Le code ci-dessous renvoie une erreur. Pourquoi?

```
mon_prenom <- "Kene"  
mon_nom <- "Nwosu"  
mon_prenom + mon_nom
```

b. Le code ci-dessous renvoie une erreur. Pourquoi? (Regardez attentivement)

```
mon_1er_prenom <- "Kene"  
mon_nom <- "Nwosu"  
  
paste(mon_1er_prenom, mon_nom)
```

## Nommer les objets

Il n'y a que *deux choses difficiles* en informatique : l'invalidation du cache et *nommer les objets*.

– Phil Karlton.

Étant donné qu'une grande partie de votre travail dans R implique d'interagir avec des objets que vous avez créés, il est important de choisir des noms intelligents pour ces objets.

Nommer des objets est difficile car les noms doivent être à la fois **courts** (afin que vous puissiez les taper rapidement) et **informatifs** (afin que vous puissiez facilement vous souvenir de ce qu'il y a à l'intérieur de l'objet), et ces deux objectifs sont souvent en conflit.

Ainsi, les noms trop longs, comme celui ci-dessous, sont mauvais parce qu'ils sont longs à saisir.

```
exemple_de donnees_sur_lepidemie_debola_en_sierra_leone_en_2014
```

Et un nom comme données est mauvais car il n'est pas informatif ; le nom ne donne pas une bonne idée de ce qu'est l'objet.

Au fur et à mesure que vous écrivez du code R, vous apprendrez à écrire des noms courts et informatifs.

---

Pour les noms composés de plusieurs mots, il existe quelques conventions pour séparer les mots :

```
forme_serpent <- "La forme serpent utilise les traits de soulignement"  
forme_pointe <- "La forme pointe utilise des points"  
formeChameau <- "La forme chameau met en majuscule les nouveaux mots (Mais pas  
le premier mot)"
```

Nous recommandons la forme\_serpent, qui utilise tous les mots en minuscules et sépare les mots par \_.

---

Notez également qu'il existe certaines restrictions sur les noms d'objets :

- les noms doivent commencer par une lettre. Ainsi, 2014\_data n'est pas un nom valide (car il commence par un chiffre).
- les noms ne peuvent contenir que des lettres, des chiffres, des points (.) et des traits de soulignement (\_). Donc ebola-data ou ebola~data ou ebola data avec un espace ne sont pas des noms valides.

Si vous voulez vraiment utiliser ces caractères dans vos noms d'objets, vous pouvez entourer les noms avec des accents graves :

```
`ebola-données`  
`ebola~données`  
`données Ebola`
```

Tous les noms ci-dessus sont des noms d'objet R valides. Par exemple, saisissez et exéutez le code suivant :

```
`ebola~data` <- ebola_data  
`ebola~data`
```

Mais en général, vous devriez éviter d'utiliser des accents graves pour remédier aux mauvais noms d'objets. Écrivez simplement les noms propres.

## Question 6

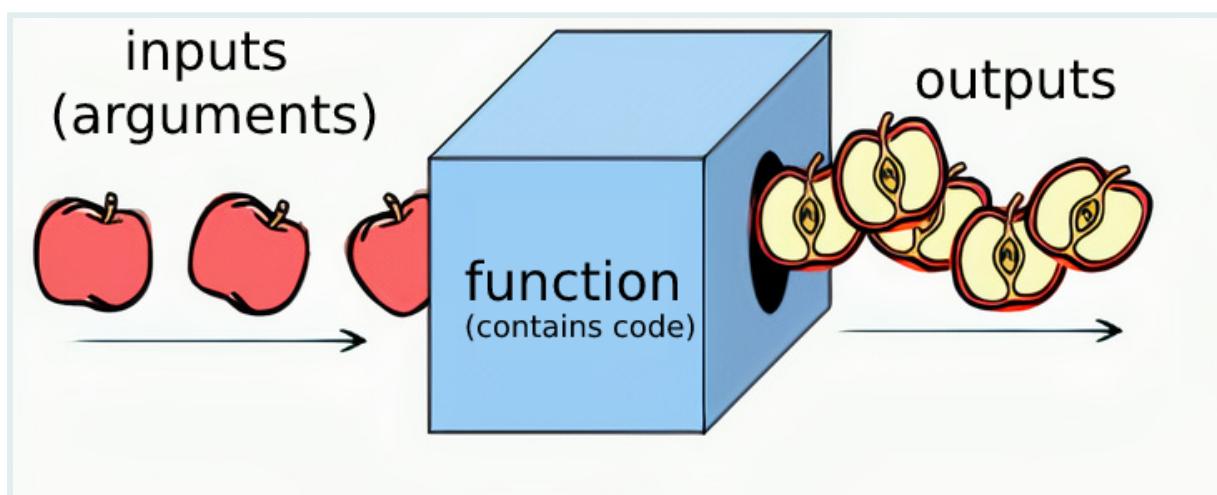
Dans le morceau de code ci-dessous, nous essayons de prendre les 20 premières lignes du tableau ebola\_data. Toutes ces lignes sauf une comportent une erreur. Quelle ligne fonctionnera correctement ?

```
20_premieres_lignes <- head(ebola_data, 20)
vingt-premieres-lignes <- head(ebola_data, 20)
premiere_20_lignes <- head(ebola_data, 20)
```

## Les fonctions

Une grande partie de votre travail dans R consistera à appeler des *fonctions*.

Vous pouvez considérer chaque fonction comme une machine qui reçoit certaines entrées (ou *arguments*) et renvoie des données de sortie.



Jusqu'à présent, vous avez déjà vu de nombreuses fonctions, notamment `sqrt()`, `paste()` et `plot()`. Exécutez les lignes ci-dessous pour vous rafraîchir la mémoire :

```
sqrt(100) # Racine carrée
paste("Je suis le nombre", 2 + 2) # Coller
plot(women) # Fait ressortir le graphique
```

### Syntaxe de la fonction de base

La façon standard d'appeler une fonction est de fournir une *valeur* pour chaque *argument* :

```
nom_de_la_fonction(argument1 = "valeur", argument2 = "valeur")
```

Démontrons ceci avec la fonction `head()`, qui renvoie les premiers éléments d'un objet.

Pour retourner les trois premières lignes de le jeu de données Ebola, exécutez :

```
head(x = ebola_data, n = 3)
```

```
##   id age sex   status date_of_onset date_of_sample district
## 1 167  55   M confirmed  2014-06-15  2014-06-21 Kenema
## 2 129  41   M confirmed  2014-06-13  2014-06-18 Kailahun
## 3 270  12   F confirmed  2014-06-28  2014-07-03 Kailahun
```

Dans le code ci-dessus, `head()` prends deux arguments :

- `x`, l'objet qui nous intéresse, et
- `n`, le nombre d'éléments à retourner.

On peut aussi permuter l'ordre des arguments :

```
head(n = 3, x = ebola_data)
```

```
##   id age sex   status date_of_onset date_of_sample district
## 1 167  55   M confirmed  2014-06-15  2014-06-21 Kenema
## 2 129  41   M confirmed  2014-06-13  2014-06-18 Kailahun
## 3 270  12   F confirmed  2014-06-28  2014-07-03 Kailahun
```

---

Si vous placez les valeurs des arguments dans le bon ordre, vous pouvez éviter de saisir de leurs noms. Ainsi, les deux lignes de code suivantes sont équivalentes et s'exécutent toutes les deux :

```
head(x = ebola_data, n = 3)
```

```
##   id age sex   status date_of_onset date_of_sample district
## 1 167  55   M confirmed  2014-06-15  2014-06-21 Kenema
## 2 129  41   M confirmed  2014-06-13  2014-06-18 Kailahun
## 3 270  12   F confirmed  2014-06-28  2014-07-03 Kailahun
```

```
head(ebola_data, 3)
```

```
##   id age sex   status date_of_onset date_of_sample district
## 1 167  55   M confirmed  2014-06-15  2014-06-21 Kenema
## 2 129  41   M confirmed  2014-06-13  2014-06-18 Kailahun
## 3 270  12   F confirmed  2014-06-28  2014-07-03 Kailahun
```

Mais si les valeurs des arguments sont dans le mauvais ordre, vous obtiendrez une erreur si vous ne saisissez pas les noms des arguments. Ci-dessous, la première ligne fonctionne mais la seconde ne fonctionne pas :

```
head(n = 3, x = ebola_data)
head(3, ebola_data)
```

(Pour connaître “l’ordre correct” des arguments, consultez le fichier d’aide de la fonction `head()`)

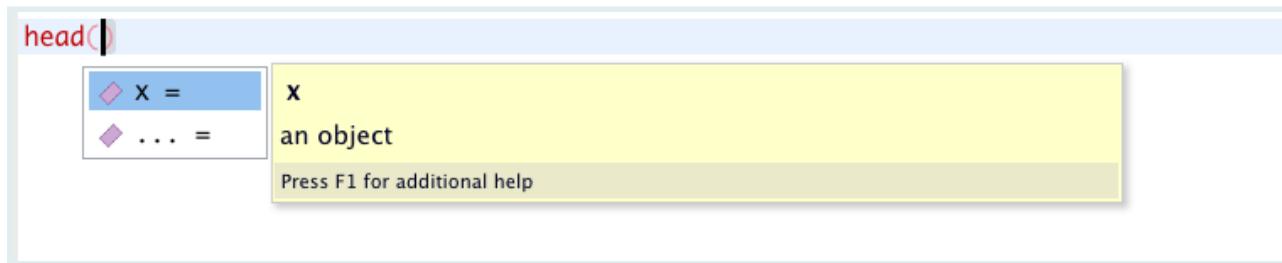
Certains arguments de fonction peuvent être ignorés, car ils ont des valeurs *par défaut*.

Par exemple, avec `head()`, la valeur par défaut de `n` est 6, donc exécuter uniquement `head(ebola_data)` renverra les 6 premières lignes.

```
head(ebola_data)
```

```
##   id age sex    status date_of_onset date_of_sample district
## 1 167  55   M confirmed  2014-06-15   2014-06-21   Kenema
## 2 129  41   M confirmed  2014-06-13   2014-06-18 Kailahun
## 3 270  12   F confirmed  2014-06-28   2014-07-03 Kailahun
## 4 187   NA   F confirmed  2014-06-19   2014-06-24 Kailahun
## 5  85   20   M confirmed  2014-06-08   2014-06-24 Kailahun
## 6 277  30   F confirmed  2014-06-29   2014-07-01   Kenema
```

Pour voir les arguments d’une fonction, appuyez sur la touche **Tab** lorsque votre curseur se trouve à l’intérieur des parenthèses de la fonction :



## Question 7

Dans les lignes de code ci-dessous, nous essayons de prendre les 6 premières lignes de le jeu de données `women` (qui est intégré à R). Quelle ligne n'est pas valide ?

```
head(women)
head(women, 6)
head(x = women, 6)
head(x = women, n = 6)
head(6, women)
```

(Si vous n’êtes pas sûr, essayez simplement de saisir et d’exécuter chaque ligne. N’oubliez pas que le but ici est que vous faire acquérir un peu de pratique.)

n'importe quel nombre d'arguments d'entrée.

Vous pouvez donc avoir soit deux arguments :

```
paste("Luigi", "Fenway")
```

```
## [1] "Luigi Fenway"
```

Soit quatre arguments :

```
paste("Luigi", "Fenway", "Luigi", "Fenway")
```

```
## [1] "Luigi Fenway Luigi Fenway"
```

Et ainsi de suite jusqu'à l'infini.

Et comme vous vous en souvenez peut-être, nous pouvons également coller (`paste()`) des objets nommés :

```
prenom <- "Luigi"  
paste("Mon prenom est", prenom, "et mon nom est", nom)
```

```
## [1] "Mon prenom est Luigi et mon nom est Fenway"
```



Les fonctions comme `paste()` peuvent prendre plusieurs valeurs car elles ont un argument spécial, des points de suspension : ... Si vous consultez le fichier d'aide de la fonction `paste`, vous verrez ceci :

#### Arguments

... one or more R objects, to be converted to character vectors.

Un autre argument utile pour `paste()` est appelé `sep`. Il indique à R quel caractère utiliser pour séparer les termes :

```
paste("Luigi", "Fenway", sep = "-")
```

```
## [1] "Luigi-Fenway"
```

## L' imbrication des fonctions

La résultante d'une fonction peut être immédiatement prise en compte par une autre fonction. C'est ce qu'on appelle l'imbrication de fonctions.

Par exemple, la fonction `tolower()` convertit une chaîne de caractères en minuscules.

```
tolower("LUIGI")
```

```
## [1] "luigi"
```

Vous pouvez prendre la sortie de cette fonction et la passer directement dans une autre fonction :

```
paste(tolower("LUIGI"), "est mon nom")
```

```
## [1] "luigi est mon nom"
```

---

Sans cette option d'imbrication, il faudrait affecter un objet intermédiaire :

```
mon_nom_minuscule <- tolower("LUIGI")
paste(mon_nom_minuscule, "est mon nom")
```

```
## [1] "luigi est mon nom"
```

L'imbrication des fonctions sera bientôt très utile.

## Question 8

Les morceaux de code ci-dessous sont tous des exemples d'imbrication de fonctions. Une des lignes contient une erreur. De quelle ligne s'agit-il et quelle est l'erreur ?

```
sqrt(head(women))
```

```
paste(sqrt(9), "plus 1 est", sqrt(16))
```

```
sqrt(tolower("LUIGI"))
```

## Les packages

Comme nous l'avons mentionné précédemment, R est formidable car il est extensible par l'utilisateur : n'importe qui peut créer un *package* logiciel qui ajoute de nouvelles fonctionnalités. La majeure partie de la puissance de R provient de ces packages.

Dans la leçon précédente, vous avez installé et chargé le package {highcharter} à l'aide des fonctions `install.packages()` et `library()`. Apprenons-en un peu plus sur les packages maintenant.

### Un premier exemple : le package {tableone}

Installons et utilisons maintenant un autre package R, appelé `tableone` :

```
install.packages("tableone")
```

```
library(tableone)
```

Notez que vous n'avez besoin d'installer un package qu'une seule fois, mais vous devez le charger avec `library()` chaque fois que vous voulez l'utiliser. Cela signifie que vous devez généralement exécuter la ligne `install.packages()` directement depuis la console, plutôt que de la saisir dans votre script.

Le package facilite la construction du "Tableau 1", c'est-à-dire un tableau avec les caractéristiques de l'échantillon d'étude que l'on trouve couramment dans les articles de recherche biomédicale.

Le cas d'utilisation le plus simple consiste à résumer le jeu de données. Il suffit d'introduire la base de données dans l'argument "data" de la fonction principale "CreateTableOne()".

```
CreateTableOne(data = ebola_data)
```

```
##                                     Overall
##   n                               200
##   id (mean (SD))      146.00 (82.28)
##   age (mean (SD))      33.12 (17.85)
##   sex = M (%)          76 (38.0)
##   status = suspected (%) 18 ( 9.0)
##   date_of_onset (%)    1 ( 0.5)
##   2014-05-18            1 ( 0.5)
##   2014-05-20            1 ( 0.5)
##   2014-05-21            1 ( 0.5)
##   2014-05-22            2 ( 1.0)
##   2014-05-23            1 ( 0.5)
##   2014-05-24            2 ( 1.0)
```

```

##      2014-05-26          8 ( -4.0)
##      2014-05-27          7 ( -3.5)
##      2014-05-28          1 ( -0.5)
##      2014-05-29          9 ( -4.5)
##      2014-05-30          4 ( -2.0)
##      2014-05-31          2 ( -1.0)
##      2014-06-01          2 ( -1.0)
##      2014-06-02          1 ( -0.5)
##      2014-06-03          1 ( -0.5)
##      2014-06-05          1 ( -0.5)
##      2014-06-06          5 ( -2.5)
##      2014-06-07          3 ( -1.5)
##      2014-06-08          4 ( -2.0)
##      2014-06-09          1 ( -0.5)
##      2014-06-10         22 (11.0)
##      2014-06-11          1 ( -0.5)
##      2014-06-12          7 ( -3.5)
##      2014-06-13         15 ( -7.5)
##      2014-06-14          8 ( -4.0)
##      2014-06-15          3 ( -1.5)
##      2014-06-16          1 ( -0.5)
##      2014-06-17          4 ( -2.0)
##      2014-06-18          5 ( -2.5)
##      2014-06-19          8 ( -4.0)
##      2014-06-20          7 ( -3.5)
##      2014-06-21          2 ( -1.0)
##      2014-06-22          1 ( -0.5)
##      2014-06-23          2 ( -1.0)
##      2014-06-24          8 ( -4.0)
##      2014-06-25          6 ( -3.0)
##      2014-06-26         10 ( -5.0)
##      2014-06-27          9 ( -4.5)
##      2014-06-28         17 ( -8.5)
##      2014-06-29          7 ( -3.5)
## date_of_sample (%) 
##      2014-05-23          1 ( -0.5)
##      2014-05-25          1 ( -0.5)
##      2014-05-26          1 ( -0.5)
##      2014-05-27          2 ( -1.0)
##      2014-05-28          1 ( -0.5)
##      2014-05-29          2 ( -1.0)
##      2014-05-31          9 ( -4.5)
##      2014-06-01          6 ( -3.0)
##      2014-06-02          1 ( -0.5)
##      2014-06-03          9 ( -4.5)
##      2014-06-04          4 ( -2.0)
##      2014-06-05          1 ( -0.5)
##      2014-06-06          2 ( -1.0)
##      2014-06-07          2 ( -1.0)
##      2014-06-10          2 ( -1.0)
##      2014-06-11          4 ( -2.0)
##      2014-06-12          3 ( -1.5)
##      2014-06-13          3 ( -1.5)
##      2014-06-14          1 ( -0.5)
##      2014-06-15         21 (10.5)
##      2014-06-16          1 ( -0.5)

```

```

##      2014-06-17      5 ( 2.5)
##      2014-06-18     13 ( 6.5)
##      2014-06-19      9 ( 4.5)
##      2014-06-21      8 ( 4.0)
##      2014-06-22      7 ( 3.5)
##      2014-06-23      6 ( 3.0)
##      2014-06-24      6 ( 3.0)
##      2014-06-25      3 ( 1.5)
##      2014-06-27      5 ( 2.5)
##      2014-06-28      2 ( 1.0)
##      2014-06-29      8 ( 4.0)
##      2014-06-30      6 ( 3.0)
##      2014-07-01      4 ( 2.0)
##      2014-07-02     16 ( 8.0)
##      2014-07-03     13 ( 6.5)
##      2014-07-04      2 ( 1.0)
##      2014-07-05      2 ( 1.0)
##      2014-07-06      1 ( 0.5)
##      2014-07-08      3 ( 1.5)
##      2014-07-12      1 ( 0.5)
##      2014-07-14      1 ( 0.5)
##      2014-07-17      1 ( 0.5)
##      2014-07-21      1 ( 0.5)
##      district (%)
##      Bo              4 ( 2.0)
##      Kailahun        146 (73.0)
##      Kenema          41 (20.5)
##      Kono             2 ( 1.0)
##      Port Loko        2 ( 1.0)
##      Western Urban    5 ( 2.5)

```

Vous pouvez voir qu'il y a 200 patients dans ce jeu de données, l'âge moyen est de 33 ans et 38% de l'échantillon de l'échantillon est de sexe masculin, entre autres détails.

Très cool! (Un problème est que le package suppose que les variables de date sont nominales; à cause de cela, le tableau de sortie est beaucoup trop long!)

Le but de cette démonstration de {tableone} est de vous montrer qu'il y a beaucoup de puissance dans les packages R externes. C'est une grande force de travailler avec R, un langage open-source avec un écosystème dynamique de contributeurs. Des milliers de personnes travaillent actuellement sur des packages qui pourraient vous être utiles un jour.

Vous pouvez rechercher sur Google “Cool R Packages” et parcourir les réponses si vous souhaitez en savoir plus sur d'autres packages R.



**SIDE NOTE** Vous avez peut-être remarqué que nous mettons les noms de packages entre accolades, par ex. {tableone}. Il s'agit simplement d'une convention de style entre les utilisateurs/enseignants de R. Les accolades ne signifient rien.

## Signifiants complets

Le *signifiant complet* d'une fonction inclut à la fois le nom du package et le nom de la fonction : `package::function()`.

Ainsi par exemple, au lieu d'écrire :

```
CreateTableOne(data = ebola_data)
```

Nous pourrions écrire cette fonction avec son signifiant complet, `package::function()` :

```
tableone::CreateTableOne(data = ebola_data)
```

Vous n'avez généralement pas besoin d'utiliser ces signifiants complets dans vos scripts. Mais dans certaines situations, cela peut s'avérer utile :

La raison la plus courante est que vous souhaitez indiquer très clairement de quel package provient une fonction.

Deuxièmement, vous souhaitez parfois éviter d'avoir à exécuter `library(package)` avant d'accéder aux fonctions d'un package. En d'autres termes, vous souhaitez utiliser une fonction d'un package sans d'abord charger ce package à partir de la bibliothèque. Dans ce cas, vous pouvez utiliser la syntaxe complète du signifiant.

Donc ce qui suit :

```
tableone::CreateTableOne(data = ebola_data)
```

est équivalent à:

```
library(tableone)
CreateTableOne(data = ebola_data)
```

## Question 9

Considérez le code ci-dessous :

```
tableone::CreateTableOne(data = ebola_data)
```

Lequel des énoncés suivants est une interprétation correcte de ce que signifie ce code :

- A. Le code applique la fonction `CreateTableOne` du package `{tableone}` sur l'objet `ebola_data`.
- B. Le code applique l'argument `CreateTableOne` de la fonction `{tableone}` sur le package `ebola_data`.

C. Le code applique la fonction `CreateTableOne` du package `{tableone}` sur le package `ebola_data`.

### `pacman::p_load()`

Plutôt que d'utiliser deux fonctions distinctes, `install.packages()` puis `library()`, pour installer puis charger des packages, vous pouvez utiliser une seule fonction, `p_load()`, du package `{pacman}` pour installer automatiquement un package s'il n'est pas encore installé, et chargez le package. Nous encourageons cette approche dans la suite de ce cours.

Installez `{pacman}` maintenant en exécutant ceci dans votre console :

```
install.packages("pacman")
```

À partir de maintenant, lorsque vous découvrirez un nouveau package, vous pouvez simplement utiliser `pacman::p_load(nom_du_package)` pour installer et charger le package :

Essayez ceci maintenant pour le package “outbreaks”, que nous utiliserons bientôt :

```
pacman::p_load(outbreaks)
```

---

Maintenant, nous avons un petit problème. La merveilleuse fonction `pacman::p_load()` installe et charge automatiquement les packages.

Mais ce serait bien d'avoir du code qui installe automatiquement le package `{pacman}` lui-même, s'il est manquant sur l'ordinateur d'un utilisateur.

Mais si vous mettez la ligne `install.packages()` dans un script, comme ceci :

```
install.packages("pacman")
pacman::p_load(here, rmarkdown)
```

vous perdrez beaucoup de temps. Parce qu'à chaque fois qu'un utilisateur ouvre et exécute un script, il va *réinstaller* `{pacman}`, ce qui peut prendre un certain temps. Au lieu de cela, nous avons besoin d'un code qui vérifie d'abord si `pacman` n'est pas encore installé et l'installe si ce n'est pas le cas.

Nous pouvons le faire avec le code suivant :

```
if(!require(pacman)) install.packages("pacman")
```

Vous n'avez pas besoin de le comprendre pour le moment, car il utilise une syntaxe que vous n'avez pas encore apprise. Notez simplement que dans les prochains chapitres, nous commencerons souvent un script avec un code comme celui-ci :

```
if(!require(pacman)) install.packages("pacman")
pacman::p_load(here, rmarkdown)
```

La première ligne installe {pacman} s'il n'est pas encore installé. La deuxième ligne utilise la fonction `p_load()` de {pacman} pour charger les packages restants (et `pacman::p_load()` installe tous les packages qui ne sont pas encore installés).

Ouf! J'espère que votre tête est encore intacte.

### Question 10

Au début d'un script R, nous aimerais installer et charger le package appelé {janitor}. Parmi les morceaux de code suivants, quels sont ceux que nous vous recommandons d'inclure dans votre script ?

A.

```
if(!require(pacman)) install.packages("pacman")
pacman::p_load(janitor)
```

B

```
install.packages("janitor")
library(janitor)
```

C

```
install.packages("janitor")
pacman::p_load(janitor)
```

---

## Conclusion

Avec vos nouvelles connaissances sur les objets R, les fonctions R et les packages d'où proviennent les fonctions, vous êtes prêt, croyez-le ou non, à effectuer une analyse de données de base dans R. Nous allons nous lancer tête baissée dans la prochaine leçon. Au plaisir de vous y retrouver !

### Réponses

1. Vrai.
2. Le signe de division est évalué en premier.
3. La réponse est C. Le code `2 + 2 + 2` est évalué avant d'être stocké dans l'objet.
4. a. La valeur est 1. Le code est évalué à '9-8'.

- b. `table(ebola_data$district)`
5. a. Il n'est pas possible d'ajouter deux chaînes de caractères. L'addition ne fonctionne que pour les nombres.
- b. `mon_1er_nom` est initialement saisi avec le chiffre 1, mais dans la commande `paste()`, il est saisi avec la lettre "l".
6. La troisième ligne est la seule ligne avec un nom d'objet valide :  
`premieres_20_lignes`
7. La dernière ligne, `head(6, women)`, n'est pas valide car les arguments sont dans le mauvais ordre et ils ne sont pas nommés.
8. Le troisième morceau de code a un problème. Il tente de trouver la racine carrée d'un caractère, ce qui est impossible.
9. La première ligne, A, est l'interprétation correcte.
10. Le premier morceau de code est la méthode recommandée pour installer et charger le package {janitor}

## Contributeurs

Les membres de l'équipe suivants ont contribué à cette leçon :



### KENE DAVID NWOSU

Data analyst, the GRAPH Network  
Passionate about world improvement



### LAMECK AGASA

Statistician/Data Scientist



### OLIVIA KEISER

Head of division of Infectious Diseases and Mathematical Modelling,  
University of Geneva



### LAURE NGUEMO

Data Science Education Officer  
Gets very excited at the mention of data, especially health related data

## Références

Certains éléments de cette leçon ont été adaptés à partir des sources suivantes :

- “File:Apple slicing function.png.” *Wikimedia Commons, le référentiel multimédia gratuit.* 1er octobre 2021, 04:26 UTC. 20 mars 2022, 17:27 <[https://commons.wikimedia.org/w/index.php?title=File:Apple\\_slicing\\_function.png&oldid=594767630](https://commons.wikimedia.org/w/index.php?title=File:Apple_slicing_function.png&oldid=594767630)>.
- “PsyteachR | Compétences en données pour une recherche reproductible.” 2021. Github.io. 2021. <https://psyteahr.github.io/reprores-v2/index.html>.
- Douglas, Alex, Deon Roos, Francesca Mancini, Ana Couto et David Lusseau. 2022. “Une introduction à R.” Intro2r.com. 27 janvier 2022. <https://intro2r.com/>.

This work is licensed under the [Creative Commons Attribution Share Alike](#) license.



# Notes de leçon | Exploration de données: Ebola en Sierra Leone

GRAPH Network & OMS, soutenu par le Fonds Mondial

January 2024

Ce cours a été créé par le Réseau GRAPH, une organisation à but non lucratif basée à l'Institut de santé globale de l'Université de Genève, en collaboration avec l'Organisation mondiale de la Santé, dans le cadre d'une subvention du Fonds mondial pour créer des cours afin de renforcer les capacités nationales en matière d'analyse épidémiologique.



Introduction .....
Configuration du script .....
En-tête .....
Les packages .....
Importer des données dans R .....
Introduction à la reproductibilité .....
Exploration rapide des données .....
<code>vis_dat()</code> .....
<code>inspect_cat()</code> et <code>inspect_num()</code> .....
Analyse d'une seule variable numérique .....
Extraire un vecteur colonne avec <code>\$</code> .....
Opérations de base sur une variable numérique .....
Visualisation d'une variable numérique .....
Analyse d'une seule variable nominale .....
Tableaux de fréquence .....
Visualisation d'une variable nominale .....
Réponses aux questions sur l'épidémie .....
Conclusion .....

---

## Objectifs d'apprentissage

1. Vous pouvez utiliser l'interface graphique de RStudio pour importer des données CSV dans R.
2. Vous pouvez expliquer le concept de reproductibilité.
3. Vous pouvez utiliser les fonctions `nrow()`, `ncol()` et `dim()` pour obtenir les dimensions d'un jeu de données, et la fonction `summary()` pour obtenir un résumé des variables du jeu de données.
4. Vous pouvez utiliser les fonctions `vis_dat()`, `inspect_num()` et `inspect_cat()` pour obtenir des résumés visuels d'un jeu de données.
5. Vous pouvez inspecter une variable numérique :
  - avec les fonctions récapitulatives `mean()`, `median()`, `max()`, `min()`, `length()` et `sum()` ;
  - avec le code `ggplot2` généré par `esquisse`.
6. Vous pouvez inspecter une variable nominale :
  - avec la fonction `janitor:::tabyl()` ;

- avec le code ggplot2 généré par esquisse.

---

## Introduction

Grâce à vos connaissances nouvellement acquises sur les fonctions et les objets, vous disposez désormais des éléments de base nécessaires pour effectuer une analyse de données simple dans R. Alors commençons. L'objectif est de commencer à travailler avec des données le plus rapidement possible, avant même que vous ne vous sentiez prêt.

Ici, vous allez analyser un jeu de données sur les cas confirmés et suspects de fièvre hémorragique Ebola en Sierra Leone en mai et juin 2014 (Fang et al., 2016). Les données sont présentées ci-dessous :

Vous allez importer et explorer ce jeu de données, puis utiliser R pour répondre aux questions suivantes sur l'épidémie :

- **Quand le premier cas a-t-il été signalé ?**
- **Quel était l'âge médian des personnes concernées ?**
- **Les cas ont-ils été plus nombreux chez les hommes ou chez les femmes ?**
- **Quel district avait eu le plus de cas signalés ?**
- **À la fin du mois de juin 2014, l'épidémie était-elle en croissance ou en régression ?**

---

## Configuration du script

Tout d'abord, ouvrez un nouveau script dans RStudio avec **File > New File > R Script**. (Si vous êtes sur RStudio Cloud, vous pouvez ouvrir n'importe lequel de vos projets créés précédemment.)



Ensuite, enregistrez le script avec **File > Save as** ou appuyez sur Commande/Contrôle + S. Enregistrez le fichier sous le nom "ebola\_analysis" ou quelque chose de similaire.

---

### SIDE NOTE



Videz votre environnement au début de l'analyse

**SIDE NOTE**

Lorsque vous démarrez une nouvelle analyse, votre environnement R devrait normalement être vide. Vérifiez cela en ouvrant l'onglet *Environment* ; il devrait dire “Environment is empty”. Si au contraire, il affiche des objets précédemment chargés, il est recommandé de redémarrer R en allant dans l'option de menu Session > Restart R

## En-tête

Ajoutez un titre, un nom et une date au début du script, sous forme de commentaires de code. Il s'agit généralement d'une bonne pratique pour l'écriture de scripts R:

```
# Analyse Ebola Sierra Leone  
# John Example-de-Nom Doe  
# 2024-01-01
```

## Les packages

Ensuite, utilisez la fonction `p_load()` de `{pacman}` pour charger les packages que vous utiliserez. Placez ceci sous un en-tête de section appelé “Load packages”, avec quatre traits d'union, comme indiqué ci-dessous :

```
# Charger les packages ----  
if(!require(pacman)) install.packages("pacman")  
pacman::p_load(  
  tidyverse, # métapackage  
  inspectdf,  
  plotly,  
  janitor,  
  visdat,  
  esquisse  
)
```

**REMINDER**

N'oubliez pas que le *signifiant complet* d'une fonction inclut à la fois le nom du package et le nom de la fonction, `package::function()`. Ce signifiant complet est pratique si vous souhaitez utiliser une fonction avant d'avoir chargé son package source. C'est le cas dans le code ci-dessus : nous voulons utiliser `p_load()` de `{pacman}` sans charger formellement le package `{pacman}`, nous saisissons donc `pacman::p_load()`

Nous pourrions aussi d'abord charger `{pacman}` avant d'utiliser la fonction `p_load` :

**REMINDER**

```
library(pacman) # charge d'abord {pacman}  
p_load(tidyverse) # utilise `p_load` de {pacman} pour charger  
d'autres packages
```

(Rappelez-vous également que l'avantage de `p_load()` est qu'il installe automatiquement un package s'il n'est pas encore installé. Sans `p_load()`, vous devez d'abord installer le package avec `install.packages()` avant de pouvoir le charger avec `library()`.)

## Importer des données dans R

Maintenant que les packages nécessaires sont chargés, vous devez importer le jeu de données.

**SIDE NOTE**

### À propos du jeu de données Ebola

Les données sur lesquelles vous allez travailler contiennent un échantillon d'informations sur les patients de l'épidémie d'Ebola de 2014-2016 en Sierra Leone. Elles proviennent d'un document de recherche qui a analysé la dynamique de transmission de cette épidémie. Les variables clés incluent le statut d'un cas, si le cas a été confirmé ou suspecté ; la date d'apparition, lorsque des symptômes de type Ebola sont apparus chez un patient ; et la date\_de\_l'échantillon, date à laquelle l'échantillon de test a été prélevé. Pour en savoir plus sur ces données, consultez la publication source ici : [bit.ly/ebola-data-source](http://bit.ly/ebola-data-source). Ou recherchez le DOI suivant sur DOI.org : 10.1073/pnas.1518587113.

Accédez à [bit.ly/view-ebola-data](http://bit.ly/view-ebola-data) pour afficher le jeu de données sur lequel vous allez travailler. Cliquez ensuite sur l'icône de téléchargement en haut de la page pour le télécharger sur votre ordinateur.

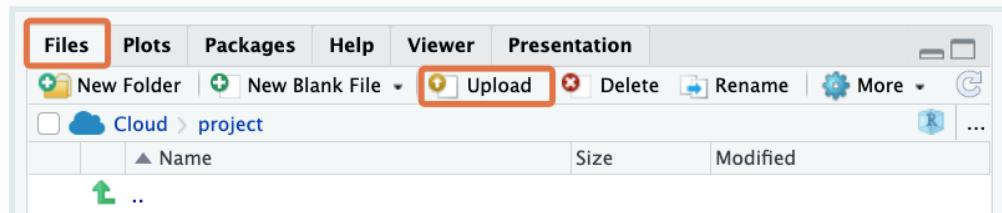
	A	B	C	D	E	F	G
1	id	age	sex	status	date_of_onset	date_of_sample	district
2	92	6	M	confirmed	2014-06-10	2014-06-15	Kailahun
3	51	46	F	confirmed	2014-05-30	2014-06-04	Kailahun

Vous pouvez laisser le jeu de données dans votre dossier de téléchargements ou le déplacer vers un endroit plus respectable ; les prochaines étapes fonctionneront

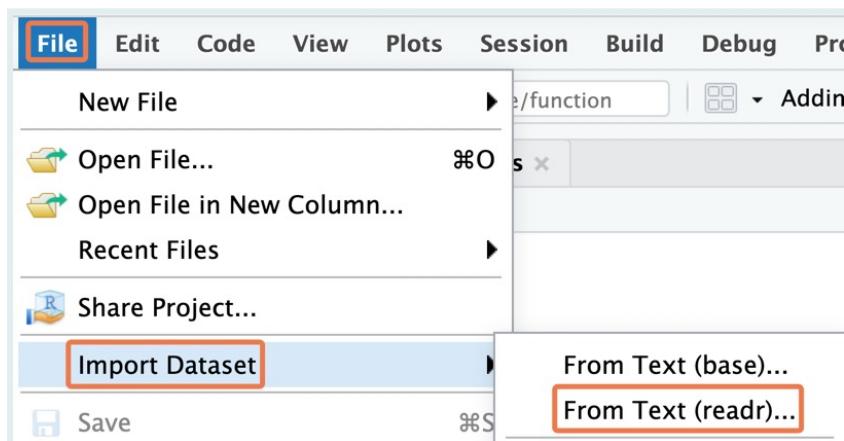
indépendamment de l'endroit où les données sont stockées. Dans la leçon suivante, vous apprendrez à organiser correctement vos projets d'analyse de données, et nous réfléchirons à la configuration idéale des dossiers pour le stockage des données.



REMARQUE : Si vous utilisez RStudio Cloud, vous devez importer votre jeu de données sur le cloud. Faites cela dans l'onglet "Files" en cliquant sur le bouton "Upload".



Ensuite, dans le menu de RStudio, allez dans **File > Import Dataset > From Text** (readr).



Parcourez les fichiers de l'ordinateur et naviguez jusqu'au jeu de données téléchargé. Cliquez pour l'ouvrir. Vous devriez voir une boîte de dialogue d'importation comme celle-ci :

**Import Text Data**

File/URL:

Data Preview:

...	...	...	...	...	...	...	...	...	...
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

Import Options:

Name: <input type="text" value="ebola_sierra_leone"/>	<input checked="" type="checkbox"/> First Row as Names	Delimiter: <input type="button" value="Comma"/>	Escape: <input type="button" value="None"/>
Skip: <input type="text" value="0"/>	<input checked="" type="checkbox"/> Trim Spaces	Quotes: <input type="button" value="Default"/>	Comment: <input type="button" value="Default"/>
	<input checked="" type="checkbox"/> Open Data Viewer	Locale: <input type="button" value="Configure..."/>	NA: <input type="button" value="Default"/>

Laissez tous les paramètres d'importation aux valeurs par défaut ; cliquez simplement sur "Import" en bas ; cela devrait charger le jeu de données dans R.

Vous devez voir dans le volet d'environnement qu'un objet appelé « ebola\_sierra\_leone » ou quelque chose de similaire a été importé :

R | Global Environment

Data

ebola\_sierra\_leone | 200 obs. of 7 variables

RStudio devrait également avoir appelé la fonction `View()` sur votre jeu de données, vous devriez donc voir une vue familière de ces données sur une feuille de calcul :

	<input type="button" value="◀"/>	<input type="button" value="▶"/>	<input type="button" value="↶"/>	<input type="button" value="↷"/>	<input type="button" value="Filter"/>				
	▲	id	age	sex	status	date_of_onset	date_of_sample	district	▼
1		92	6.0	M	confirmed	2014-06-10	2014-06-15	Kailahun	
2		51	46.0	F	confirmed	2014-05-30	2014-06-04	Kailahun	
3		230	NA	M	confirmed	2014-06-26	2014-06-30	Kenema	
4		139	25.0	F	confirmed	2014-06-13	2014-06-18	Kailahun	

Jetez maintenant un coup d'œil à votre console. Observez-vous que vos actions dans l'interface graphique ont en fait déclenché l'exécution d'un code R ? Copiez la ligne de code qui inclut la fonction `read_csv()`, en omettant le symbole >.

```
>
>
> library(readr)
> ebola_sierra_leone <- read_csv("ebola_sierra_leone.csv")
Rows: 200 Columns: 7
#> #> #> #> #> #> #>
```

Copy this  
(or something similar)

Collez le code copié dans votre script R et intitulez cette section “Charger les données”. Cela peut ressembler à ce qui suit (le chemin du fichier entre guillemets diffère d'un ordinateur à l'autre).

```
# Charger les données ----  
ebola_sierra_leone <- read_csv("~/Téléchargements/ebola_sierra_leone.csv")
```

Beau travail jusqu'à présent!

Votre script R devrait ressembler à ceci :

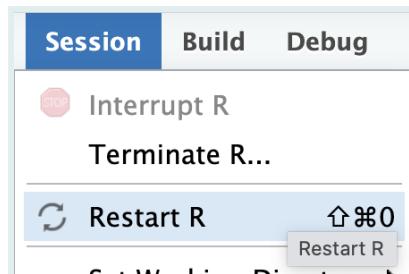
```
# Analyse Ebola Sierra Leone  
# John Example-de-Nom Doe  
# 2024-01-01  
  
# Charger les packages ----  
if(!require(pacman)) install.packages("pacman")  
pacman::p_load(  
  tidyverse,  
  inspectdf,  
  plotly,  
  janitor,  
  visdat  
)  
  
# Charger les données ----  
ebola_sierra_leone <-  
read_csv("~/Téléchargements/ebola_sierra_leone.csv")
```



## Introduction à la reproductibilité

Maintenant que le code d'importation des données se trouve dans votre script R, vous pouvez facilement réexécuter ce script à tout moment pour réimporter le jeu de données ; il ne sera pas nécessaire de refaire la procédure manuelle de pointer-cliquer pour l'importation des données.

Essayez de redémarrer R et de réexécuter le script maintenant. Enregistrez votre script avec Contrôle/Commande + s , puis redémarrez R avec le menu RStudio, à Session > Restart R. L'option de menu ressemble à ceci :



Si le redémarrage réussit, votre console devrait afficher le message "Restarting R Session":

```
Restarting R session...> |
```

A screenshot of the RStudio console window. It shows the message "Restarting R session..." followed by a cursor symbol. The console has a light gray background and a dark gray border.

Vous devriez également voir la phrase "Environment is empty" dans l'onglet Environnement, indiquant que le jeu de données que vous avez importé n'est plus stocké par R – vous commencez avec un nouvel espace de travail.



Pour relancer votre script, utilisez Commande/Contrôle + a pour surligner tout le code, puis Commande/Contrôle + Enter pour l'exécuter.

Si cela a fonctionné, félicitations ; vous avez les prémisses de votre premier script d'analyse "reproductible" !

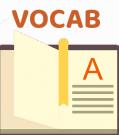
### Que signifie "reproductible" ?



Lorsque vous faites des choses avec du code plutôt qu'en pointant et en cliquant, il est facile pour n'importe qui de réexécuter, ou de *reproduire* ces étapes, en réexécutant simplement votre script.

Bien que vous puissiez utiliser l'interface graphique de RStudio pour pointer et cliquer tout au long du processus d'importation de données, vous devez toujours copier le code pertinent dans votre script afin que

votre script reste un enregistrement reproductible de toutes vos étapes d'analyse.

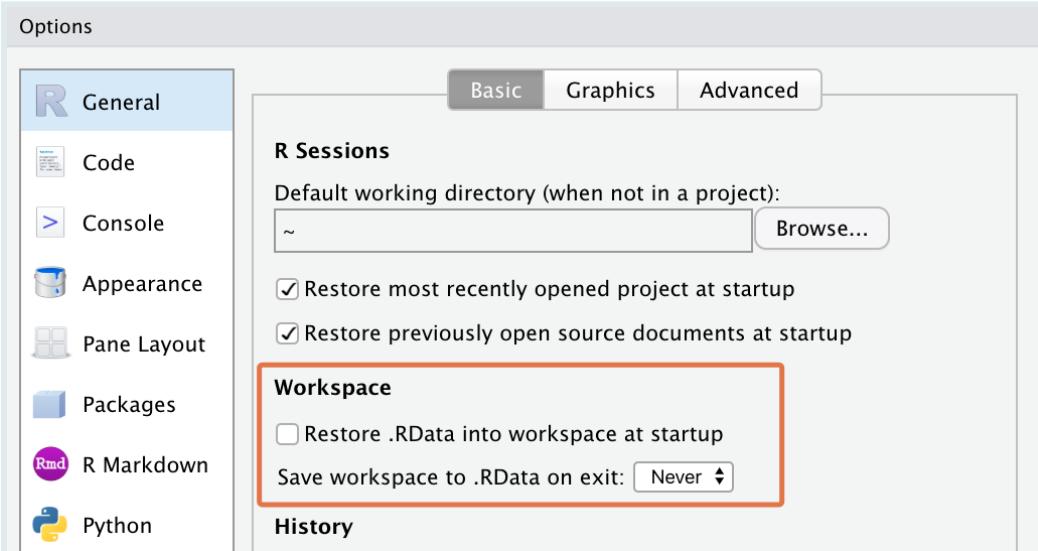


Bien sûr, votre script jusqu'à présent n'est pas encore *entièrement* reproductible, car le chemin d'accès au jeu de données est spécifique à votre ordinateur uniquement. Plus tard, nous verrons comment utiliser des chemins d'accès relatifs, afin que le code d'importation de données puisse fonctionner sur n'importe quel ordinateur.

Si votre environnement n'était pas vide après le redémarrage de R, cela signifie que vous avez sauté une étape dans une leçon précédente. Faites ceci maintenant :

- Dans le menu de RStudio, allez à Tools > Global Options pour afficher la boîte de dialogue des options de RStudio.
- Ensuite, allez dans General > Basic et **décochez** la case qui dit "Restore .RData into workspace at startup".
- Pour l'option "save your workspace to .RData on exit", réglez-la sur "Never".

#### WATCH OUT



## Exploration rapide des données

Passons maintenant à l'exploration des données. Nous verrons une série de fonctions qui peuvent être utilisées pour obtenir une vue d'ensemble de vos données.

Pour afficher les 6 lignes supérieures et inférieures du jeu de données, vous pouvez utiliser les fonctions `head()` et `tail()` :

```
# Explorer les données ----  
head(ebola_sierra_leone)
```

```
## # A tibble: 6 × 7  
##       id   age sex status date_of_onset date_of_sample  
##   <dbl> <dbl> <chr> <chr>   <date>        <date>  
## 1     92     6 M confirmed 2014-06-10 2014-06-15  
## 2     51    46 F confirmed 2014-05-30 2014-06-04  
## 3    230    NA M confirmed 2014-06-26 2014-06-30  
## 4    139    25 F confirmed 2014-06-13 2014-06-18  
## 5      8     8 F confirmed 2014-05-22 2014-05-27  
## 6    215    49 M confirmed 2014-06-24 2014-06-29  
## # i 1 more variable: district <chr>
```

```
tail(ebola_sierra_leone)
```

```
## # A tibble: 6 × 7  
##       id   age sex status date_of_onset date_of_sample  
##   <dbl> <dbl> <chr> <chr>   <date>        <date>  
## 1    214     6 F confirmed 2014-06-24 2014-06-30  
## 2     28    45 F confirmed 2014-05-27 2014-06-01  
## 3     12    27 F confirmed 2014-05-22 2014-05-27  
## 4    110     6 M confirmed 2014-06-10 2014-06-15  
## 5    209    40 F confirmed 2014-06-24 2014-06-27  
## 6     35    29 M suspected 2014-05-28 2014-06-01  
## # i 1 more variable: district <chr>
```

Pour afficher l'intégralité du jeu de données, utilisez la fonction `View()`.

```
View(ebola_sierra_leone)
```

Cette fonction ouvrira à nouveau une vue familière des données sous forme de feuille de calcul :

	<b>id</b>	<b>age</b>	<b>sex</b>	<b>status</b>	<b>date_of_onset</b>	<b>date_of_sample</b>	<b>district</b>
1	92	6.0	M	confirmed	2014-06-10	2014-06-15	Kailahun
2	51	46.0	F	confirmed	2014-05-30	2014-06-04	Kailahun
3	230	NA	M	confirmed	2014-06-26	2014-06-30	Kenema
4	139	25.0	F	confirmed	2014-06-13	2014-06-18	Kailahun

Vous pouvez fermer cet onglet et revenir à votre script.

---

Les fonctions `nrow()`, `ncol()` et `dim()` vous donnent les dimensions de votre jeu de données :

```
nrow(ebola_sierra_leone) # nombre de lignes
```

```
## [1] 200
```

```
ncol(ebola_sierra_leone) # nombre de colonnes
```

```
## [1] 7
```

```
dim(ebola_sierra_leone) # nombre de lignes et de colonnes
```

```
## [1] 200 7
```

#### REMINDER



Si vous n'êtes pas sûr de ce que fait une fonction, n'oubliez pas que vous pouvez obtenir de l'aide sur la fonction avec le symbole du point d'interrogation. Par exemple, pour obtenir de l'aide sur la fonction `ncol()`, exéutez :

```
?ncol
```

Une autre fonction souvent utile est `summary()` :

```
summary(ebola_sierra_leone)
```

```
##           id              age             sex
##  Min.   : 1.00   Min.   : 1.80   Length:200
##  1st Qu.: 62.75  1st Qu.:20.00  Class :character
```

```

## Median :131.50   Median :35.00   Mode  :character
## Mean    :136.72   Mean   :33.85
## 3rd Qu.:208.25   3rd Qu.:45.00
## Max.   :285.00   Max.   :80.00
## NA's    :4
##      status          date_of_onset      date_of_sample
## Length:200          Min.   :2014-05-18  Min.   :2014-05-23
## Class  :character   1st Qu.:2014-06-01  1st Qu.:2014-06-07
## Mode   :character   Median  :2014-06-13  Median  :2014-06-18
##                  Mean   :2014-06-12  Mean   :2014-06-17
##                  3rd Qu.:2014-06-23  3rd Qu.:2014-06-29
##                  Max.   :2014-06-29  Max.   :2014-07-17
##
##      district
## Length:200
## Class  :character
## Mode   :character
##
##
##
##

```

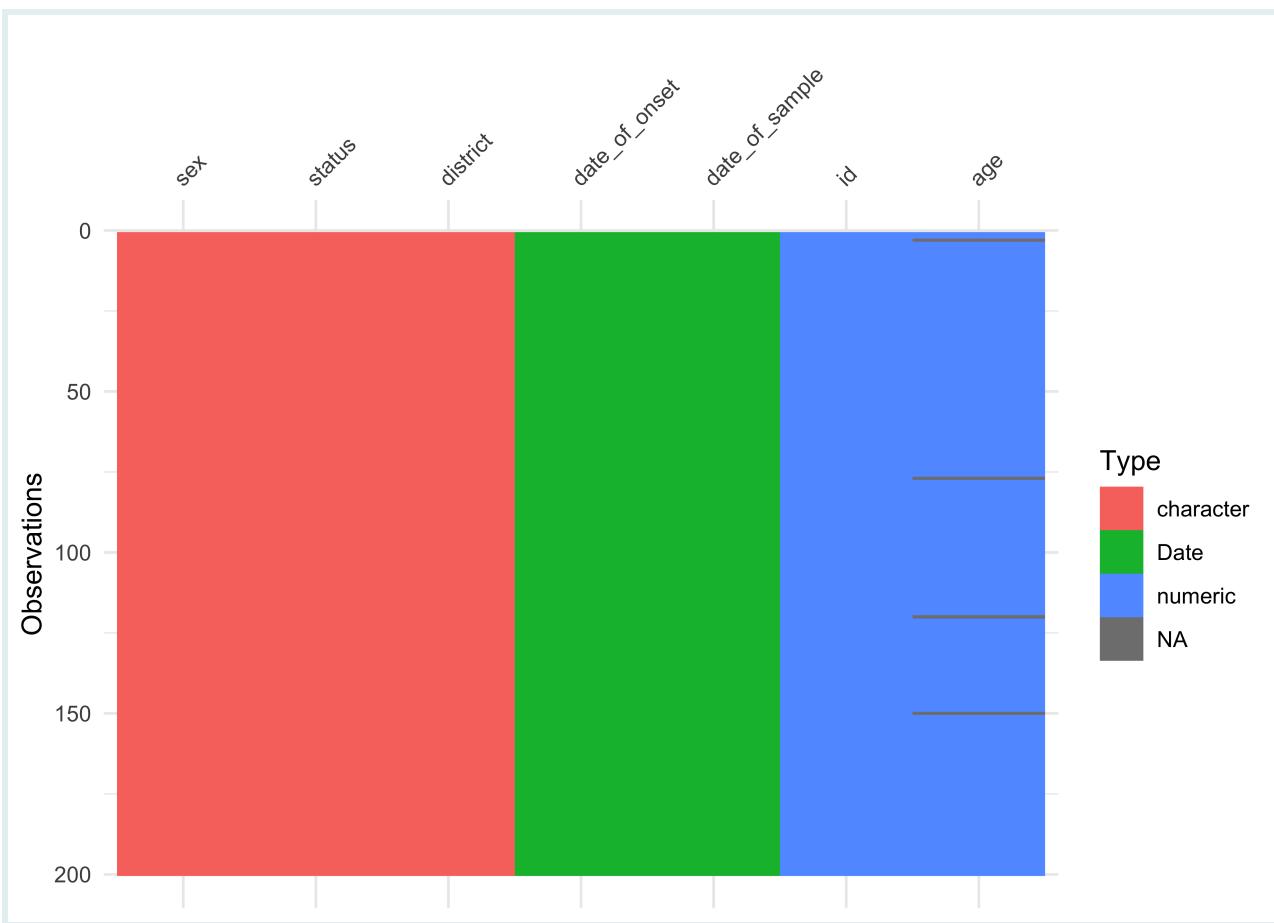
Comme vous pouvez le voir, pour les colonnes numériques de votre jeu de données, `summary()` vous donne la valeur minimale, la valeur maximale, la moyenne, la médiane et les 1er et 3e quartiles.

Pour les colonnes de caractères, il vous donne juste la longueur de la colonne (le nombre de lignes), la “classe” et le “mode”. Nous discuterons de ce que “classe” et “mode” signifient plus tard.

## `vis_dat()`

La fonction `vis_dat()` du package `{visdat}` est un excellent moyen de visualiser rapidement les types de données et les valeurs manquantes dans un jeu de données. Essayez ceci maintenant :

```
vis_dat(ebola_sierra_leone)
```



À partir de cette figure, vous pouvez voir rapidement les types de données caractère, date et numérique, et vous pouvez noter que l'âge est manquant dans certains cas.

### `inspect_cat()` et `inspect_num()`

Ensuite, `inspect_cat()` et `inspect_num()` du package `{inspectdf}` vous donnent des résumés visuels de la distribution des variables dans le jeu de données.

Si vous exécutez `inspect_cat()` sur l'objet de données, vous obtenez un résumé tabulaire des variables **nominales** dans le jeu de données , avec quelques informations cachées dans la colonne `levels` (vous apprendrez plus tard comment extraire ces informations).

```
inspect_cat(ebola_sierra_leone)
```

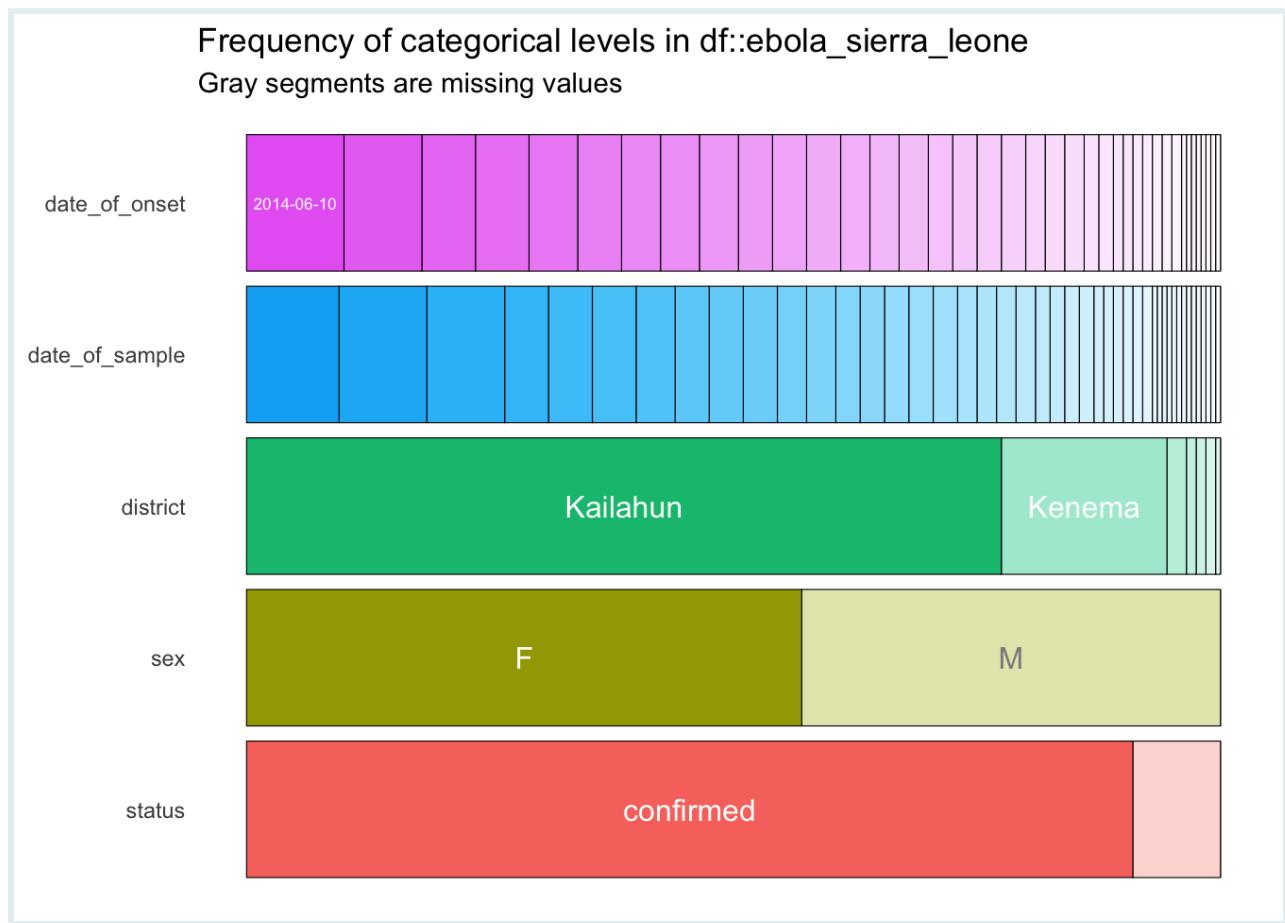
```
## # A tibble: 5 × 5
##   col_name      cnt common    common_pcnt levels
##   <chr>        <int> <chr>       <dbl> <named list>
## 1 date_of_onset     39 2014-06-10      10 <tibble>
## 2 date_of_sample    45 2014-06-15      9.5 <tibble>
## 3 district          7 Kailahun     77.5 <tibble>
```

```
## 4 sex          2 F           57  <tibble>
## 5 status       2 confirmed  91   <tibble>
```

Mais la magie opère lorsque vous exécutez `show_plot()` sur le résultat de `inspect_cat()` :

```
# stocker la sortie de `inspect_cat()` dans `resume_cat`
resume_cat <- inspect_cat(ebola_sierra_leone)

# appelle la fonction `show_plot()` sur ce résumé.
show_plot(resume_cat)
```



Vous obtenez une figure magnifique montrant la distribution de toutes les variables nominales et de date !

**SIDE NOTE**

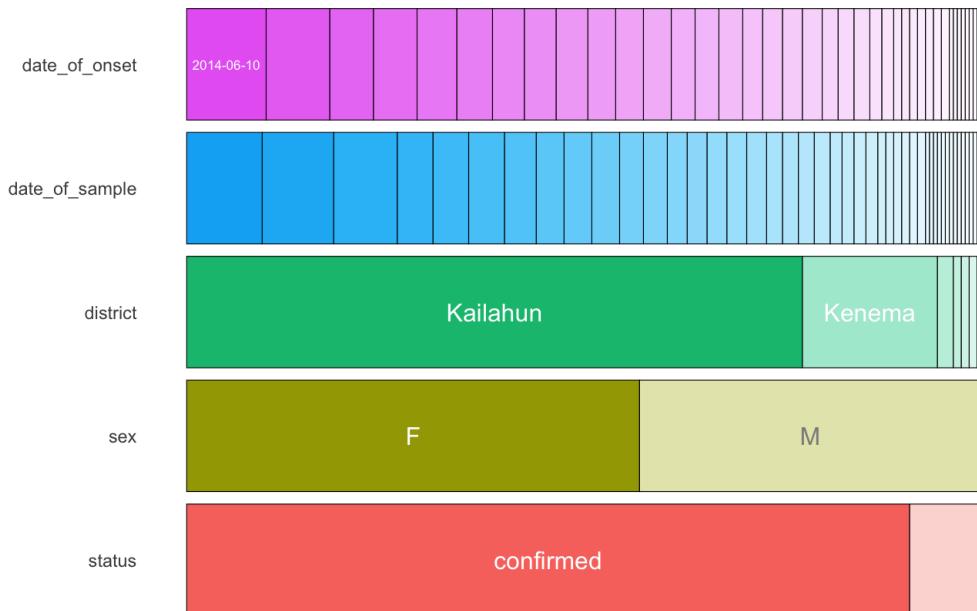


Vous pouvez également exécuter :

```
show_plot(inspect_cat(ebola_sierra_leone))
```

**SIDE NOTE**

Frequency of categorical levels in df::ebola\_sierra\_leone  
Gray segments are missing values



À partir de ce graphique, vous pouvez rapidement voir que la plupart des cas se trouvent à Kailahun et qu'il y a plus de cas chez les femmes que chez les hommes ("F" signifie "femelle").

Un problème est que dans ce graphique, les plus petites catégories ne sont pas étiquetées. Ainsi, par exemple, nous ne savons pas quelle valeur est représentée par la section blanche pour "statut" en bas à droite. Pour voir les étiquettes sur ces catégories plus petites, vous pouvez transformer cela en un graphique interactif avec la fonction `ggplotly()` du package `{plotly}`.

```
resume_cat_graph <- show_plot(resume_cat)
ggplotly(resume_cat_graph)
```

Merveilleux! Vous pouvez maintenant survoler chacune des barres pour voir la proportion de chaque section de la barre. Par exemple, vous pouvez maintenant dire que 9 % (0,090) des cas ont un statut suspect :

status

col\_name: status  
prop: 0.090  
new\_level\_key: suspected-status-ebola\_sierra\_leone

**REMINDER**

**REMINDER**

La flèche d'affectation, `<-`, peut être écrite avec le raccourci RStudio **alt + - (alt ET moins)** sous Windows ou **option + - (option ET moins)** sur macOS.

Vous pouvez obtenir un graphique similaire pour les variables numériques (continues) dans le jeu de données avec `inspect_num()`. Ici, nous montrons les trois étapes en une seule fois.

```
resume_num <- inspect_num(ebola_sierra_leone)
resume_num_graph <- show_plot(resume_num)
ggplotly(resume_num_graph)
```

Cela vous donne un aperçu des colonnes numériques `age` et `id`. Bien entendu, la distribution de la variable `id` n'est pas véritablement numérique, mais représente plutôt des identifiants uniques. Nous verrons plus tard comment changer le type de données des colonnes dans R.

## Analyse d'une seule variable numérique

Maintenant que vous avez une idée de ce à quoi ressemble le jeu de données, vous pouvez isoler et analyser une seule variable à la fois – c'est ce qu'on appelle *l'analyse univariée*.

Allez-y et créez une nouvelle section dans votre script pour cette analyse univariée.

```
# Analyse univariée, variables numériques ----
```

Commençons par analyser la variable numérique "age".

### Extraire un vecteur colonne avec \$

Pour extraire une seule variable/colonne d'un jeu de données, utilisez le signe du dollar, l'opérateur `$` :

```
ebola_sierra_leone$age # extrait la colonne 'age' dans le jeu de données
```

```
## [1] 6.0 46.0 NA 25.0 8.0 49.0 13.0 50.0 35.0 38.0 60.0 18.0
## [13] 10.0 14.0 50.0 35.0 43.0 17.0 3.0 60.0 38.0 41.0 49.0 12.0
## [25] 74.0 21.0 27.0 41.0 42.0 60.0 30.0 50.0 50.0 22.0 40.0 35.0
## [37] 19.0 3.0 34.0 21.0 73.0 65.0 30.0 70.0 12.0 15.0 42.0 60.0
## [49] 14.0 40.0 33.0 43.0 45.0 14.0 14.0 40.0 35.0 30.0 17.0 39.0
## [61] 20.0 8.0 40.0 42.0 53.0 18.0 40.0 20.0 45.0 40.0 60.0 44.0
## [73] 33.0 23.0 45.0 7.0 NA 35.0 36.0 42.0 35.0 25.0 30.0 30.0
## [85] 28.0 14.0 20.0 60.0 67.0 35.0 50.0 4.0 28.0 38.0 30.0 26.0
```

```
## [97] 37.0 30.0 3.0 56.0 32.0 35.0 54.0 42.0 48.0 11.0 1.8 63.0
## [109] 55.0 20.0 62.0 62.0 42.0 65.0 29.0 20.0 33.0 30.0 35.0 NA
## [121] 50.0 16.0 3.0 22.0 7.0 50.0 17.0 40.0 21.0 9.0 27.0 52.0
## [133] 50.0 25.0 10.0 30.0 32.0 38.0 30.0 50.0 26.0 35.0 3.0 50.0
## [145] 60.0 40.0 34.0 4.0 42.0 NA 54.0 18.0 45.0 30.0 35.0 35.0
## [157] 16.0 26.0 23.0 45.0 45.0 45.0 38.0 45.0 35.0 30.0 60.0 5.0
## [169] 18.0 2.0 70.0 35.0 3.0 30.0 80.0 62.0 20.0 45.0 18.0 28.0
## [181] 48.0 38.0 39.0 26.0 60.0 35.0 20.0 50.0 11.0 36.0 29.0 57.0
## [193] 35.0 26.0 6.0 45.0 27.0 6.0 40.0 29.0
```

### VOCAB



Cette liste de valeurs est appelée un *vecteur* dans R. Un vecteur est une sorte de structure de données dont les éléments sont d'un seul *type*. Dans ce cas, le type est "numérique". Nous vous présenterons formellement les vecteurs et autres structures de données dans un prochain chapitre. Dans cette leçon, vous pouvez considérer que "vecteur" et "variable" sont synonymes.

## Opérations de base sur une variable numérique

Pour obtenir la moyenne de ces âges, vous pouvez exécuter :

```
mean(ebola_sierra_leone$age)
```

```
## [1] NA
```

Mais il semble que nous ayons un problème. R dit que la moyenne est 'NA', ce qui signifie "non applicable" ou "non disponible". C'est parce qu'il y a des valeurs manquantes dans le vecteur des âges. (Avez-vous remarqué cela lorsque vous avez imprimé le vecteur ?) Par défaut, R ne peut pas trouver la moyenne s'il y a des valeurs manquantes. Pour ignorer ces valeurs, utilisez l'argument `na.rm` (qui signifie "Supprimer NA") en lui donnant la valeur T ou TRUE :

```
mean(ebola_sierra_leone$age, na.rm = T)
```

```
## [1] 33.84592
```

Super! Cette nécessité de supprimer les "NA" avant de calculer une statistique s'applique à de nombreuses fonctions. La fonction `median()`, par exemple, renverra également NA par défaut si elle est appelée sur un vecteur avec des NA :

```
median(ebola_sierra_leone$age) # ne fonctionne pas
```

```
## [1] NA  
  
median(ebola_sierra_leone$age, na.rm = T) # fonctionne
```

```
## [1] 35
```

---

mean et median ne sont que deux des nombreuses fonctions R qui peuvent être utilisées pour inspecter une variable numérique. Regardons quelques autres.

Mais d'abord, nous pouvons assigner le vecteur d'âge à un nouvel objet, de sorte que vous n'ayez pas à saisir ebola\_sierra\_leone\$age à chaque fois.

```
age_vec <- ebola_sierra_leone$age # assigne le vecteur à l'objet "age_vec"
```

Exécutez maintenant ces fonctions sur age\_vec et observez leurs résultats :

```
sd(age_vec, na.rm = T) # écart-type
```

```
## [1] 17.26864
```

```
max(age_vec, na.rm = T) # âge maximum
```

```
## [1] 80
```

```
min(age_vec, na.rm = T) # âge minimum
```

```
## [1] 1.8
```

```
summary(age_vec) # min, max, moyenne, quartiles et NAs
```

```
##   Min. 1st Qu. Median    Mean 3rd Qu.    Max.    NA's  
##   1.80  20.00  35.00  33.85  45.00  80.00      4
```

```
length(age_vec) # nombre d'éléments dans le vecteur
```

```
## [1] 200
```

```
sum(age_vec, na.rm = T) # somme de tous les éléments du vecteur
```

```
## [1] 6633.8
```

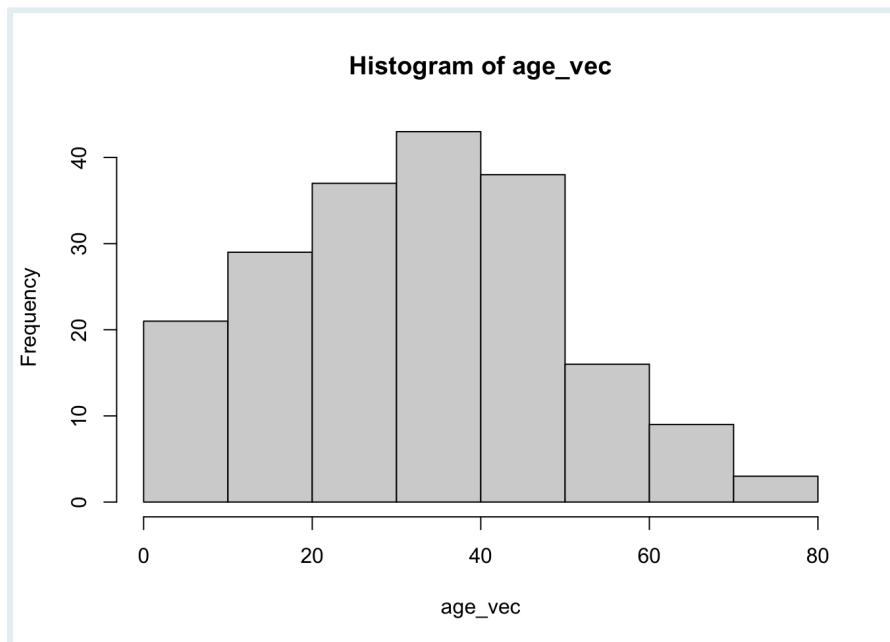
Ne vous sentez pas intimidé par la longue liste de fonctions ! Vous ne devriez pas avoir à les mémoriser ; vous devriez plutôt vous sentir libre de chercher sur Google la fonction correspondant à l'opération que vous voulez effectuer. Vous pouvez rechercher quelque chose comme "quelle est la fonction pour l'écart type dans R". L'un des premiers résultats devrait vous conduire à ce dont vous avez besoin.

## Visualisation d'une variable numérique

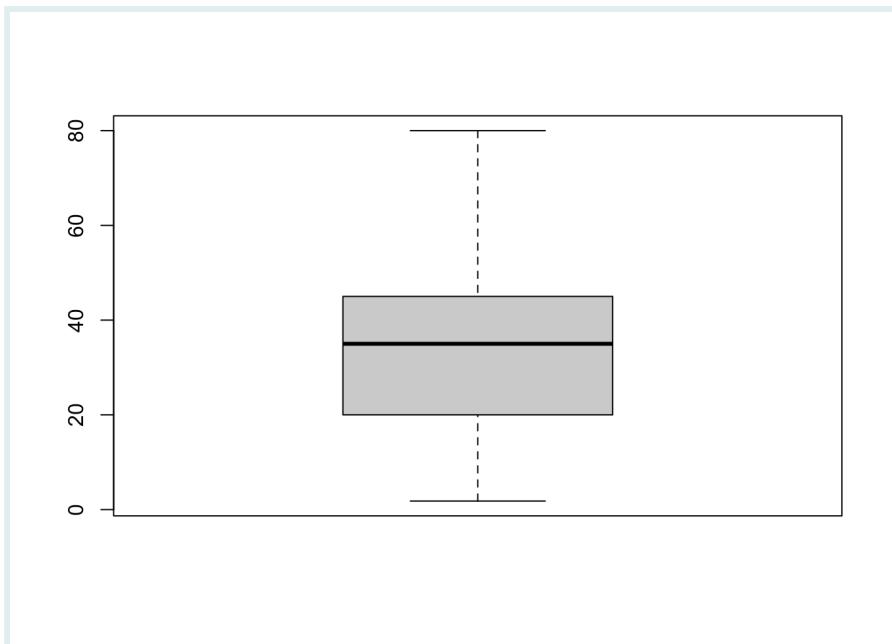
Créons maintenant un graphique pour visualiser la variable 'age'. Les deux graphiques les plus courants pour inspecter la distribution des variables numériques sont les **histogrammes** (comme la sortie de la fonction `inspect_num()` que vous avez vue précédemment) et **boxplots**.

R a des fonctions intégrées pour ceux-ci :

```
hist(age_vec)
```



```
boxplot(age_vec)
```



Agréable et facile !

Les fonctions graphiques telles que `boxplot()` et `hist()` font partie du package graphique de base de R. Ces fonctions sont simples et rapides à utiliser, mais elles n'offrent pas beaucoup de flexibilité , et il est difficile de faire de beaux graphiques avec elles. C'est pourquoi la plupart des membres de la communauté R utilisent un package d'extension, `{ggplot2}`, pour la visualisation de leurs données.

Dans ce cours, nous utiliserons indirectement `ggplot` ; en utilisant le package `{esquisse}`, qui fournit une interface conviviale pour créer des graphiques `ggplot2`.

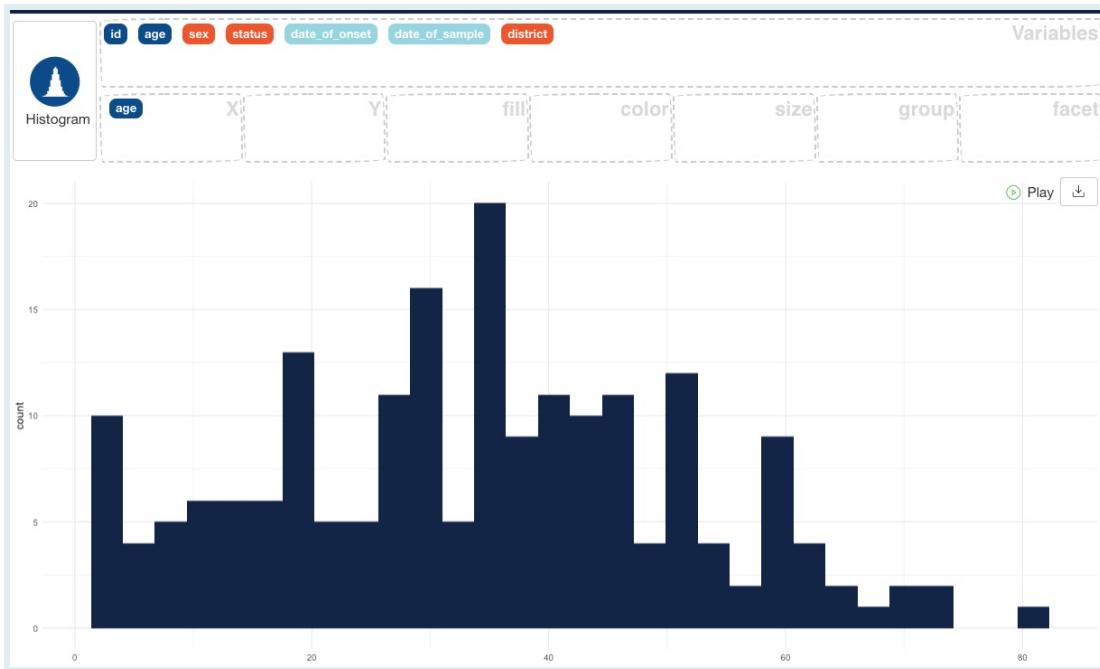
La fonction principale du package `{esquisse}` est `esquisser()`, et cette fonction prend un seul argument – le jeu de données que vous souhaitez visualiser. Nous pouvons donc exécuter :

```
esquisser(ebola_sierra_leone)
```

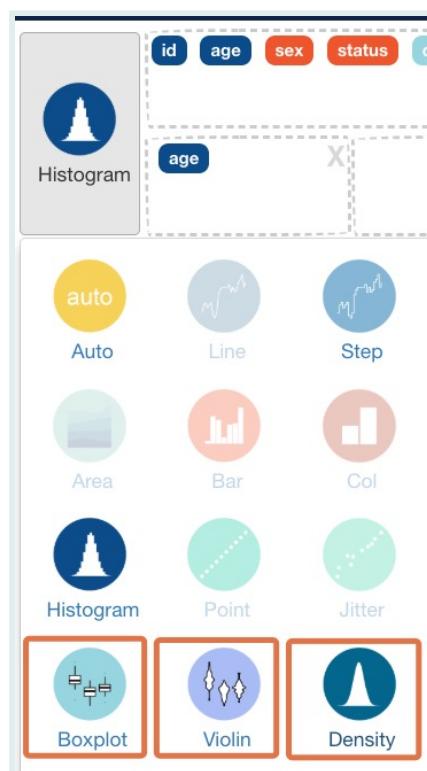
Cela devrait apporter une interface graphique que vous pouvez utiliser pour visualiser différentes variables. Pour visualiser la variable d'âge, faites simplement glisser "age" de la liste des variables vers la zone de l'axe des x :



Lorsque age est dans la case de l'axe des x, vous devriez automatiquement obtenir un histogramme des âges :



Vous pouvez modifier le type de graphique en cliquant sur le bouton "Histogram" et en sélectionnant l'un des autres types de graphique valides. Essayez lediagramme en boîte, le diagramme en violon et le diagramme de densité et observez les résultats.



Lorsque vous avez terminé de créer un graphique avec {esquisse}, vous devez copier le code qui a été créé en cliquant sur le bouton "Code" en bas à droite puis "Copy to clipboard" :



Maintenant, collez ce code dans votre script et assurez-vous que vous pouvez l'exécuter à partir de là. Le code devrait ressembler à ceci :

```
ggplot(ebola_sierra_leone) +  
  aes(x = age) +  
  geom_histogram(bins = 30L, fill = "#112446") +  
  theme_minimal()
```

En copiant le code généré dans votre script, vous vous assurez que la visualisation de données que vous avez créée est entièrement reproductible.



**PRO TIP** {esquisse} ne peut créer que des graphiques assez simples, donc lorsque vous souhaitez créer des graphiques hautement personnalisés ou complexes, vous devrez apprendre à écrire le code {ggplot} manuellement. Ce sera l'objet d'un cours ultérieur.

Vous devriez également tester les autres onglets de la barre d'outils inférieure pour voir ce qu'ils font : Labels & Title, Plot options, Appearance and Data.

#### CHALLENGE



#### graphiques bivariés et multivariés faciles

Dans cette leçon, nous nous concentrerons sur l'analyse univariée : explorer et visualiser une variable à la fois. Mais avec esquisse, il est \* si \* facile de

créer un graphique bivarié ou multivarié, vous pouvez donc déjà vous familiariser avec cela.

Essayez les graphiques suivants :

### CHALLENGE



- Faites glisser age dans la case X et sex dans la case Y.
- Faites glisser age dans la case X, sex dans la case Y et sex dans la case de remplissage.
- Faites glisser age dans la case X et district dans la case Y.

## Analyse d'une seule variable nominale

Examinons ensuite une variable nominale, les districts des cas signalés :

```
# Analyse univariée, variables nominales ----  
ebola_sierra_leone$district
```

```
## [1] "Kailahun"      "Kailahun"      "Kenema"       "Kailahun"  
## [5] "Kailahun"      "Kailahun"      "Kailahun"      "Kailahun"  
## [9] "Kenema"        "Kailahun"      "Kailahun"      "Kailahun"  
## [13] "Kailahun"      "Kailahun"      "Kailahun"      "Kailahun"  
## [17] "Kailahun"      "Kenema"        "Kono"         "Kailahun"  
## [21] "Kailahun"      "Kailahun"      "Kenema"        "Kailahun"  
## [25] "Kailahun"      "Kailahun"      "Kailahun"      "Kailahun"  
## [29] "Kenema"        "Kenema"        "Kenema"        "Kailahun"  
## [33] "Kailahun"      "Bo"            "Kailahun"      "Kailahun"  
## [37] "Kailahun"      "Kenema"        "Kenema"        "Kenema"  
## [41] "Kailahun"      "Kailahun"      "Kailahun"      "Kailahun"  
## [45] "Kailahun"      "Kailahun"      "Western Urban" "Kailahun"  
## [49] "Kailahun"      "Kailahun"      "Kailahun"      "Kailahun"  
## [53] "Kailahun"      "Kailahun"      "Kailahun"      "Kailahun"  
## [57] "Kailahun"      "Kailahun"      "Kailahun"      "Kailahun"  
## [61] "Kailahun"      "Kenema"        "Western Urban" "Kambia"  
## [65] "Kailahun"      "Kailahun"      "Kailahun"      "Kailahun"  
## [69] "Kailahun"      "Kailahun"      "Kailahun"      "Kailahun"  
## [73] "Kenema"        "Kailahun"      "Kailahun"      "Kenema"  
## [77] "Kailahun"      "Kailahun"      "Kenema"        "Kailahun"  
## [81] "Kailahun"      "Kailahun"      "Kailahun"      "Kailahun"  
## [85] "Kailahun"      "Kailahun"      "Kailahun"      "Kailahun"  
## [89] "Kailahun"      "Kenema"        "Kailahun"      "Kailahun"  
## [93] "Kailahun"      "Kono"          "Port Loko"    "Kenema"  
## [97] "Kailahun"      "Kailahun"      "Kailahun"      "Kailahun"  
## [101] "Kenema"       "Kailahun"      "Kailahun"      "Kenema"
```

```

## [105] "Kailahun"      "Kailahun"      "Kailahun"      "Kailahun"
## [109] "Kailahun"       "Kailahun"      "Kenema"        "Western Urban"
## [113] "Kailahun"       "Kailahun"      "Kailahun"      "Kailahun"
## [117] "Kailahun"       "Kailahun"      "Kailahun"      "Kailahun"
## [121] "Kailahun"       "Kailahun"      "Kenema"        "Kailahun"
## [125] "Kailahun"       "Kenema"        "Kailahun"      "Port Loko"
## [129] "Kailahun"       "Kailahun"      "Kailahun"      "Kailahun"
## [133] "Kailahun"       "Kailahun"      "Kailahun"      "Kailahun"
## [137] "Kailahun"       "Kailahun"      "Kailahun"      "Kailahun"
## [141] "Kailahun"       "Kailahun"      "Kailahun"      "Kenema"
## [145] "Kenema"         "Kailahun"      "Kenema"        "Kailahun"
## [149] "Kailahun"       "Kailahun"      "Kailahun"      "Kailahun"
## [153] "Kenema"         "Kailahun"      "Kailahun"      "Kenema"
## [157] "Kailahun"       "Kenema"        "Kailahun"      "Kailahun"
## [161] "Kenema"         "Kailahun"      "Kailahun"      "Kailahun"
## [165] "Kailahun"       "Bo"            "Kailahun"      "Kailahun"
## [169] "Kailahun"       "Kailahun"      "Kailahun"      "Kailahun"
## [173] "Kenema"         "Kailahun"      "Kailahun"      "Kenema"
## [177] "Kailahun"       "Kailahun"      "Kailahun"      "Kailahun"
## [181] "Kailahun"       "Kailahun"      "Kailahun"      "Western Urban"
## [185] "Kailahun"       "Kailahun"      "Kenema"        "Kailahun"
## [189] "Kailahun"       "Kailahun"      "Kailahun"      "Kailahun"
## [193] "Kailahun"       "Kenema"        "Kenema"        "Kailahun"
## [197] "Kailahun"       "Kailahun"      "Kailahun"      "Kenema"

```

Désolé d'avoir imprimé ce très long vecteur !

## Tableaux de fréquence

Vous pouvez utiliser la fonction `table()` pour créer un tableau de fréquence d'une variable nominale :

```
table(ebola_sierra_leone$district)
```

```

##
##          Bo      Kailahun      Kambia      Kenema
##          2           155           1          34
##      Kono      Port Loko Western Urban
##          2                2             4

```

Vous pouvez voir que la plupart des cas se trouvent à Kailahun et à Kenema.

`table()` est une fonction “de base” utile. Mais il existe une meilleure fonction pour créer des tableaux de fréquence, appelée `tabyl()`, à partir du package `{janitor}`.

Pour l'utiliser, vous devez fournir le nom de votre base de données en premier argument, puis le nom de la variable à tabuler :

```
tabyl(ebola_sierra_leone, district)
```

```

##      district  n percent
##        Bo     2   0.010
##    Kailahun 155   0.775
##      Kambia   1   0.005
##      Kenema   34   0.170
##       Kono    2   0.010
##    Port Loko   2   0.010
## Western Urban   4   0.020

```

Comme vous pouvez le voir, `tabyl()` vous donne à la fois les nombres et les pourcentages de chaque valeur. Il possède également d'autres fonctionnalités intéressantes que vous verrez plus tard.

You can also easily create cross-tabulations with `tabyl()`. You just need to add additional variables separated by a comma. For example, to create a cross-tabulation by district and sex, run:

```
tabyl(ebola_sierra_leone, district, sex)
```



```

##      district  F  M
##        Bo     0  2
##    Kailahun 91 64
##      Kambia   0  1
##      Kenema 20 14
##       Kono    0  2
##    Port Loko   1  1
## Western Urban   2  2

```

The result shows that there were 0 women in the Bo district, 2 men in the Bo district, 91 women in the Kailahun district, etc.

## Visualisation d'une variable nominale

Now, let's try to visualize the `district` variable. As we did previously, the best way to do this is to use the `esquisser()` function from the `esquisse` package. Run the following code again:

```
esquisser(ebola_sierra_leone)
```

Then, drag the `district` variable into the X-axis area:

Vous devriez obtenir un graphique à barres indiquant le nombre d'individus dans les districts. Copiez le code généré et collez-le dans votre script.

## Réponses aux questions sur l'épidémie

Avec les fonctions que vous venez d'apprendre, vous disposez des outils nécessaires pour répondre aux questions sur l'épidémie d'Ebola qui ont été énumérées en haut. Essayez-le vous-même, puis regardez les solutions.

- **Quand le premier cas a-t-il été signalé ? (regardez la date de l'échantillon)**
- **Quel était l'âge médian des personnes touchées ?**
- **Y avait-il eu plus de cas chez les hommes ou chez les femmes ?**
- **Quel est le district qui a enregistré le plus grand nombre de cas signalés ?**
- **À la fin du mois de juin 2014, l'épidémie était-elle en croissance ou en régression ?**

### Solutions

- **Quand le premier cas a-t-il été signalé ?**

```
min(ebola_sierra_leone$date_of_sample)
```

```
## [1] "2014-05-23"
```

Nous n'avons pas la date du rapport, mais la première “date\_of\_sample” (date à laquelle l'échantillon de test Ebola a été prélevé sur le patient) est le 23 mai. Nous pouvons l'utiliser comme approximation de la date du premier rapport.

- **Quel était l'âge médian des cas ?**

```
median(ebola_sierra_leone$age, na.rm = T)
```

```
## [1] 35
```

L'âge médian des cas était de 35 ans.

- **Y a-t-il plus de cas chez les hommes ou chez les femmes ?**

```
tabyl(ebola_sierra_leone$sex)
```

```
## ebola_sierra_leone$sex    n percent
##                               F 114    0.57
##                               M  86    0.43
```

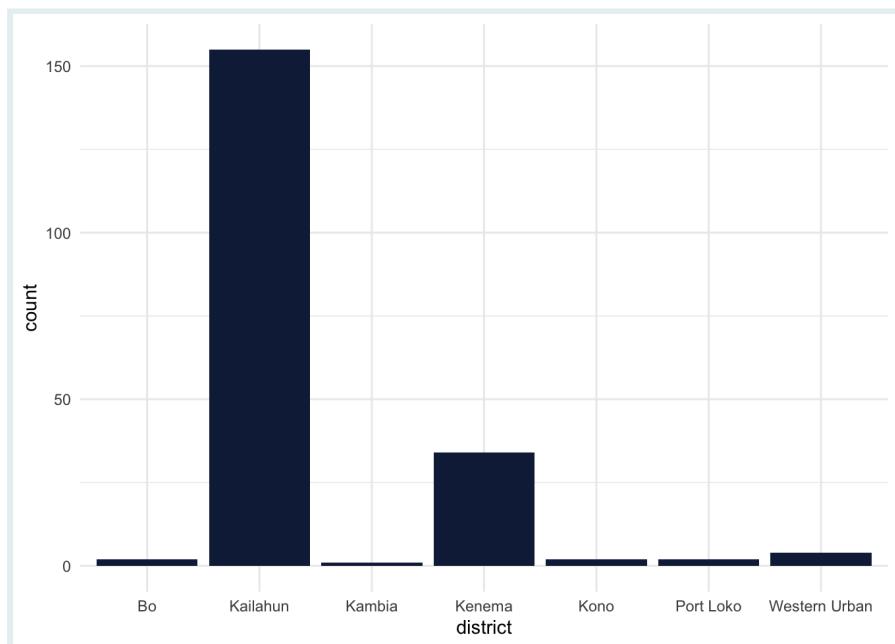
Comme on le voit dans le tableau, il y avait plus de cas chez les femmes. Plus précisément, 57 % des cas concernent des femmes.

- **Quel district a enregistré le plus de cas signalés ?**

```
tabyl(ebola_sierra_leone$district)
```

```
## ebola_sierra_leone$district    n percent
##                               Bo    2    0.010
##                               Kailahun 155   0.775
##                               Kambia   1    0.005
##                               Kenema   34   0.170
##                               Kono    2    0.010
##                               Port Loko  2    0.010
##                               Western Urban 4    0.020
```

```
# On peut aussi tracer le graphique suivant (généré avec esquisse)
ggplot(ebola_sierra_leone) +
  aes(x = district) +
  geom_bar(fill = "#112446") +
  theme_minimal()
```

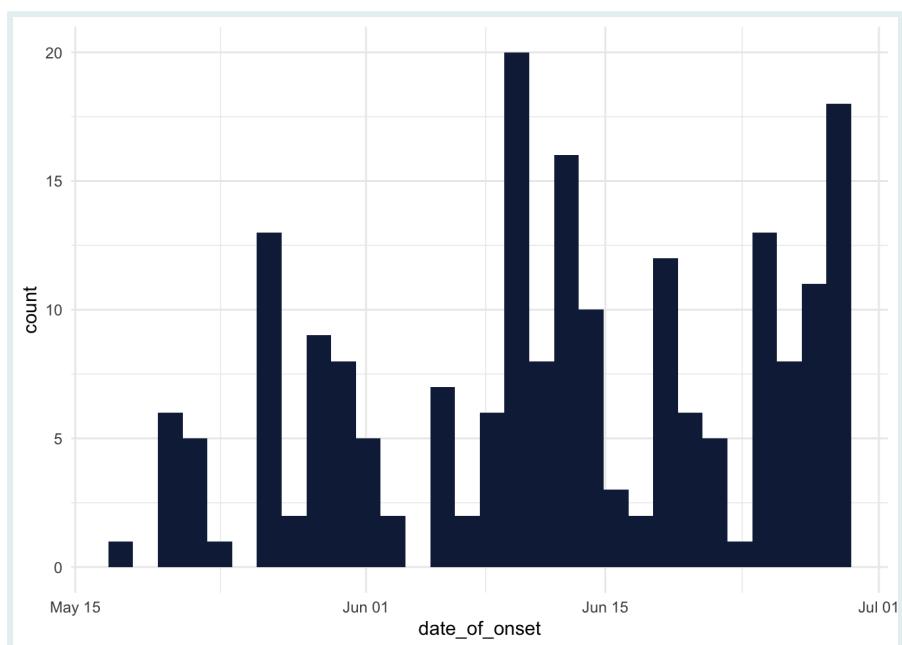


Comme on l'a vu, le district de Kailahun comptait la majorité des cas.

- **À la fin du mois de juin 2014, l'épidémie était-elle en croissance ou en recul ?**

Pour cela, nous pouvons utiliser esquisse pour générer un diagramme à barres qui montre le nombre de cas chaque jour. Faites simplement glisser la variable date\_of\_onset sur l'axe des x. Le code de sortie d'esquisse doit ressembler à ce qui suit :

```
ggplot(ebola_sierra_leone) +  
  aes(x = date_of_onset) +  
  geom_histogram(bins = 30L, fill = "#112446") +  
  theme_minimal()
```



Super ! Mais il est difficile de dire si l'épidémie augmentait ou reculait fin juin 2014 ; une tendance précise n'est pas vraiment évidente !

---

## Conclusion

Toutes nos félicitations! Vous avez maintenant fait vos premiers pas dans l'analyse des données avec R : vous avez importé un jeu de données, exploré sa structure, effectué une analyse et une visualisation univariées de base sur ses variables numériques et nominales, et vous avez pu répondre à des questions importantes sur l'épidémie en vous basant sur cette base.

Bien sûr, ce n'était qu'un *aperçu* du processus d'analyse des données – beaucoup a été laissé de côté. Avec un peu de chance, cependant, cet aperçu vous a un peu enthousiasmé par ce que vous pouvez faire avec R. Et j'espère que vous pourrez déjà commencer à appliquer certaines d'entre elles à vos propres jeux de données. Le voyage ne fait que commencer ! À bientôt.

---

## Contributeurs

Les membres de l'équipe suivants ont contribué à cette leçon :



### KENE DAVID NWOSU

Data analyst, the GRAPH Network  
Passionate about world improvement

---



### LAURE NGUEMO

Data Science Education Officer  
Gets very excited at the mention of data, especially health related data

---

---

## Références

Certains éléments de cette leçon ont été adaptés à partir des sources suivantes :

- Barnier, Julien. "Introduction à R Et Au Tidyverse." Partie 13 Diffuser et publier avec rmarkdown, 24 mai 2022. <https://juba.github.io/tidyverse/13-rmarkdown.html>.
- Yihui Xie, J. J. Allaire et Garrett Grolemund. "R Markdown : le guide définitif." Accueil, 11 avril 2022. <https://bookdown.org/yihui/rmarkdown/>.

This work is licensed under the [Creative Commons Attribution Share Alike](#) license.



# Notes de leçon | Projets RStudio

## GRAPH Network & OMS, soutenu par le Fonds Mondial

January 2024

Ce cours a été créé par le Réseau GRAPH, une organisation à but non lucratif basée à l'Institut de santé globale de l'Université de Genève, en collaboration avec l'Organisation mondiale de la Santé, dans le cadre d'une subvention du Fonds mondial pour créer des cours afin de renforcer les capacités nationales en matière d'analyse épidémiologique.



Premiers pas: projets RStudio .....
Objectifs d'apprentissage .....
Introduction .....
Création d'un nouveau Projet RStudio .....
Création de sous-dossiers de projet .....
Ajout d'un jeu de données au dossier "data" .....
Création d'un script dans le dossier "scripts" .....
Importation de données depuis le dossier "data" .....
Exportation des données vers le dossier "outputs" .....
Exportation des graphiques vers le dossier "outputs" .....
Partager un projet .....
Conclusion .....

---

## Premiers pas: projets RStudio

### Objectifs d'apprentissage

1. Vous pouvez configurer un projet RStudio et créer des sous-répertoires pour les données d'entrée, les scripts et les résultats analytiques.
2. Vous pouvez importer et exporter des données dans un projet RStudio.
3. Vous comprenez la différence entre les chemins de fichiers relatifs et absolus.
4. Vous reconnaissiez la valeur des Projets pour organiser et partager vos analyses.

### Introduction

Précédemment, vous avez parcouru certaines des étapes essentielles de l'analyse des données, de l'importation des données au calcul des statistiques de base. Mais vous avez sauté une étape cruciale : la mise en place d'un *projet* d'analyse de données.

Les analystes de données expérimentés conservent tous les fichiers associés à une analyse spécifique — données d'entrée, scripts R et résultats analytiques — ensemble dans un seul dossier. Ces dossiers sont appelés *projets* (p minuscule), et RStudio les prend en charge via RStudio *Projets* (P majuscule).

Dans cette leçon, vous apprendrez à utiliser ces projets RStudio pour organiser votre analyse de données de manière cohérente et améliorer la reproductibilité de votre travail. Vous répliquerez une partie de l'analyse que vous avez effectuée lors de la dernière leçon d'exploration des données, mais dans le contexte d'un projet RStudio.

Allons-y.

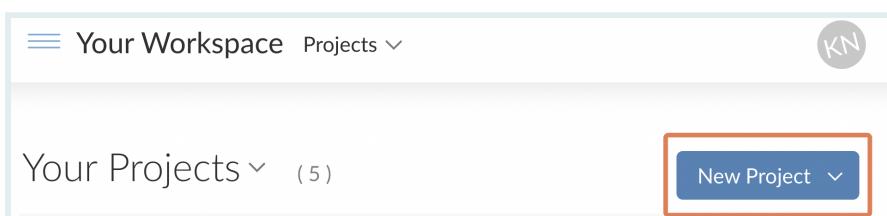
## Création d'un nouveau Projet RStudio

La création d'un nouveau projet RStudio est différente si vous êtes sur un ordinateur local et si vous êtes sur RStudio Cloud. Passez à la section qui vous concerne.

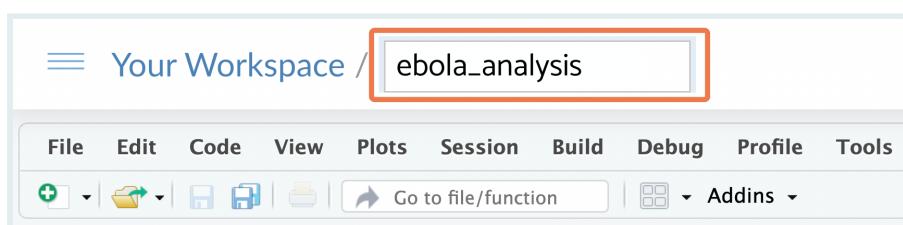
### Sur RStudio Cloud

Si vous utilisez RStudio Cloud, vous avez probablement déjà créé un projet, car vous ne pouvez pas faire d'analyse sans projets.

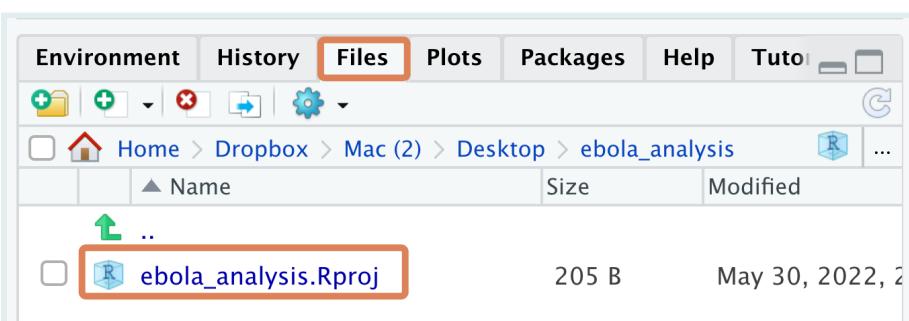
Les étapes sont assez simples : accédez à votre page d'accueil Cloud, [rstudio.cloud](https://rstudio.cloud), et cliquez sur le bouton “New Project”.



Nommez votre projet quelque chose comme `analyse_ebola` ou `analyse_ebola_proj` si vous avez déjà un projet nommé `analyse_ebola`.



Le projet RStudio que vous venez de créer n'est qu'un dossier sur un ordinateur virtuel, qui contient un fichier .Rproj (et peut-être un fichier .RHistory). Vous devriez pouvoir voir ce fichier .Rproj dans l'onglet Files de RStudio :



#### KEY POINT



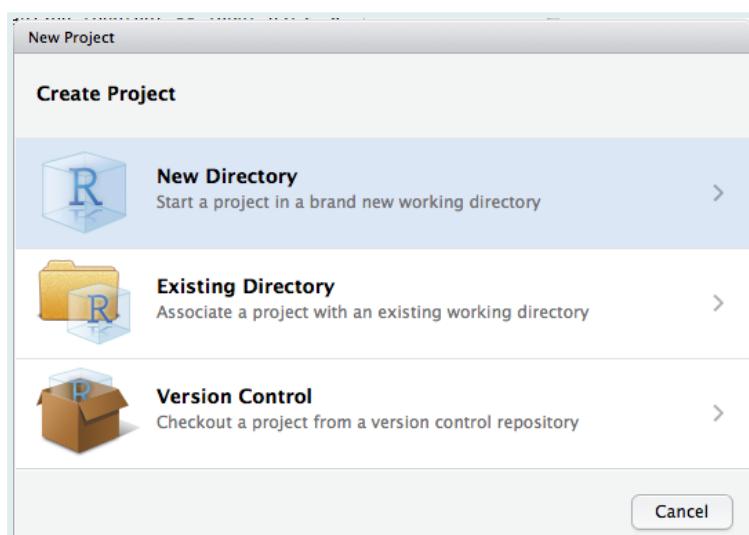
Le fichier .RProj est ce qui transforme un dossier ordinaire de l'ordinateur en un “Projet RStudio”.

## Sur un ordinateur local

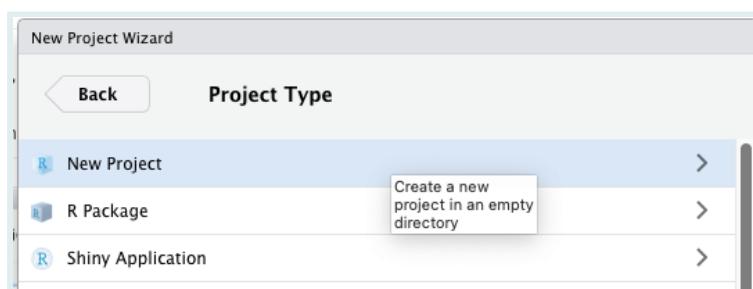
Si vous êtes sur un ordinateur local, ouvrez RStudio, puis dans le menu RStudio, allez dans “File > New Project”. Vos options peuvent être légèrement différentes des captures d’écran ci-dessous en fonction de votre système d’exploitation.



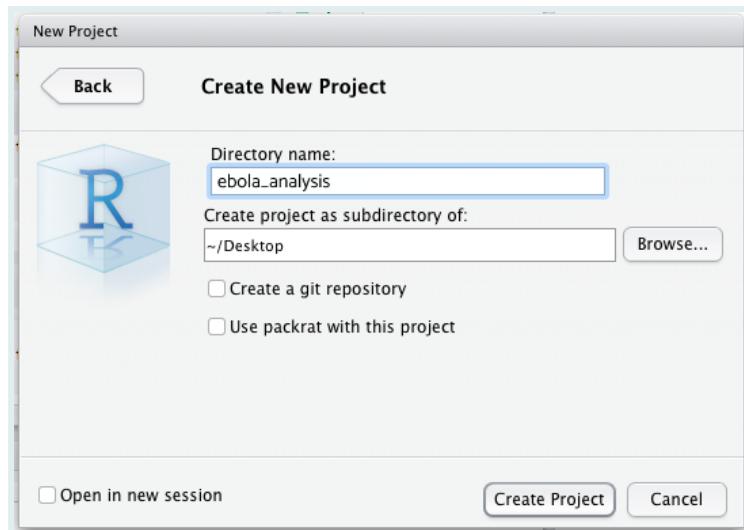
Choisissez “New directory”



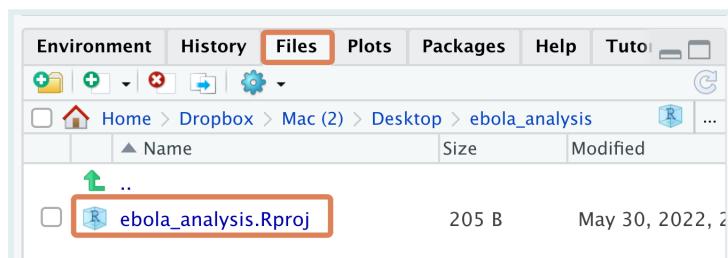
Choisissez ensuite “New Project”:



Vous pouvez appeler votre projet quelque chose comme ” analyse\_ebola” et en faire un “sous-répertoire” d’un dossier facile à trouver, tel que votre bureau. (L’expression “Créer un projet en tant que sous-répertoire de” semble effrayante, mais ce n’est pas le cas ; RStudio demande simplement : “Où dois-je mettre le dossier du projet” ?)



Le projet RStudio que vous avez créé n'est qu'un dossier contenant un fichier .Rproj (et peut-être un fichier .RHistory). Vous devriez pouvoir voir ce fichier .Rproj dans l'onglet Files de RStudio :



### Cliquez sur le fichier .Rproj pour ouvrir votre projet

#### KEY POINT

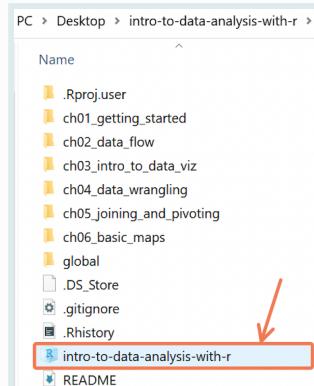


Le fichier .RProj est ce qui transforme un dossier ordinaire de l'ordinateur en un "Projet RStudio".

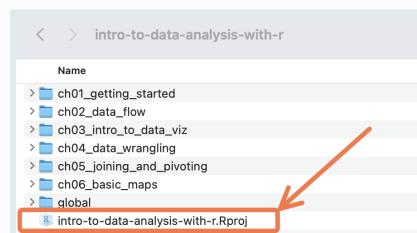
Désormais, pour ouvrir votre projet, vous devez double-cliquer sur ce fichier .RProj à partir du Finder/Explorateur de fichiers de votre ordinateur.

Sous Windows, voici un exemple de ce à quoi ressemble un fichier .Rproj dans l'explorateur de fichiers :

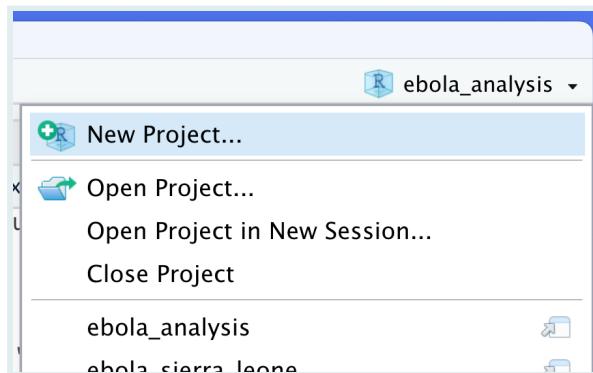
### KEY POINT



Sur macOS, voici un exemple de ce à quoi ressemble un fichier .Rproj dans le Finder :

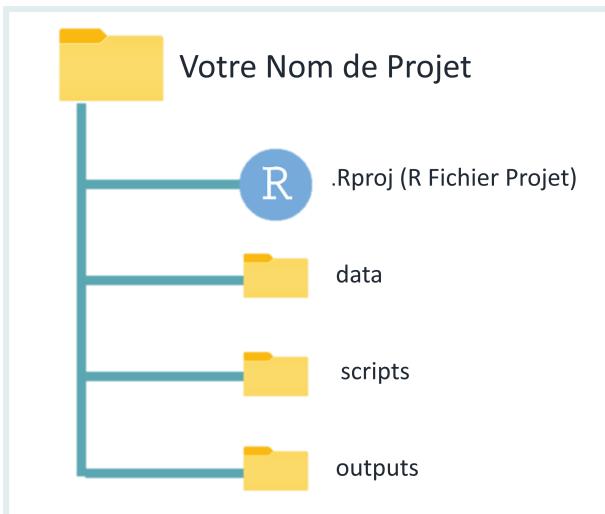


Notez également qu'il y a un en-tête en haut à droite de la fenêtre RStudio qui vous indique quel projet vous avez actuellement ouvert. En cliquant sur cet en-tête, vous pouvez accéder à d'autres options du projet. Vous pouvez notamment créer un nouveau projet, fermer un projet et ouvrir des projets récents.



### Création de sous-dossiers de projet

Les projets d'analyse de données comportent généralement au moins trois sous-dossiers : un pour les données, un autre pour les scripts et un troisième pour les résultats, comme indiqué ci-dessous :



Regardons les sous-dossiers un par un :

- **data:** Il contient les fichiers de données source (brutes) que vous utiliserez dans l'analyse. Il peut s'agir de fichiers CSV ou Excel, par exemple.
- **scripts:** Ce sous-dossier est l'endroit où vous conservez vos scripts R. Vous pouvez également enregistrer les fichiers RMarkdown dans ce dossier. (Vous découvrirez bientôt les fichiers RMarkdown.)
- **outputs:** Ici, vous enregistrez les résultats de votre analyse, comme les graphiques et les tableaux récapitulatifs. Ces résultats doivent être *jetables* et *reproductibles*. Autrement dit, vous devriez pouvoir régénérer les résultats en exécutant le code dans vos scripts. Vous le comprendrez mieux bientôt.

---

Créez maintenant ces trois sous-dossiers, “data”, “scripts” et “outputs” dans votre dossier RStudio projet. Pour ce faire, Vous devez utiliser le bouton “New Folder” (Nouveau dossier) dans l’onglet Files de RStudio:



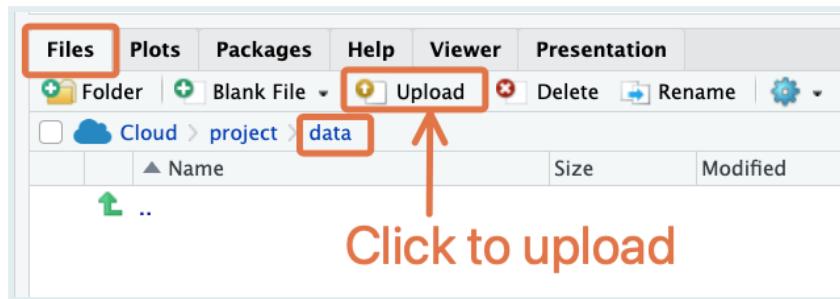
### Ajout d'un jeu de données au dossier “data”

Ensuite, vous devez déplacer le jeu de données Ebola que vous avez téléchargé dans la leçon précédente vers le sous-dossier “data” nouvellement créé (vous pouvez retélécharger ce jeu de données sur [bit.ly/ebola-data](http://bit.ly/ebola-data) si vous ne parvenez pas à retrouver l’endroit où vous l’avez stocké).

La procédure de déplacement de ce jeu de données vers le dossier “data” est différente pour les utilisateurs de RStudio Cloud et ceux utilisant un ordinateur local. Passez à la section qui vous concerne.

## Sur RStudio Cloud

Si vous êtes sur RStudio Cloud, l'ajout du jeu de données à votre dossier "data" est simple. Naviguez simplement jusqu'au dossier dans l'onglet Files, puis cliquez sur le bouton "Upload":

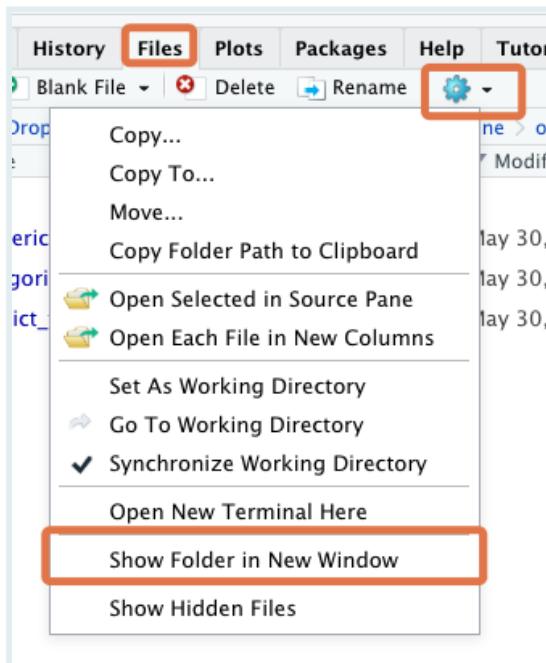


Cela fera apparaître une boîte de dialogue dans laquelle vous pourrez sélectionner le fichier à télécharger.

## Sur un ordinateur local

Sur un ordinateur local, cette étape doit être effectuée avec l'Explorateur de fichiers/Finder de votre ordinateur.

- Tout d'abord, localisez le dossier project avec l'Explorateur de fichiers/Finder de votre ordinateur. Si vous rencontrez des difficultés pour le localiser, RStudio peut vous aider : allez dans l'onglet "Files", cliquez sur "More" (l'icône d'engrenage), puis cliquez sur "Show Folder in New Window".

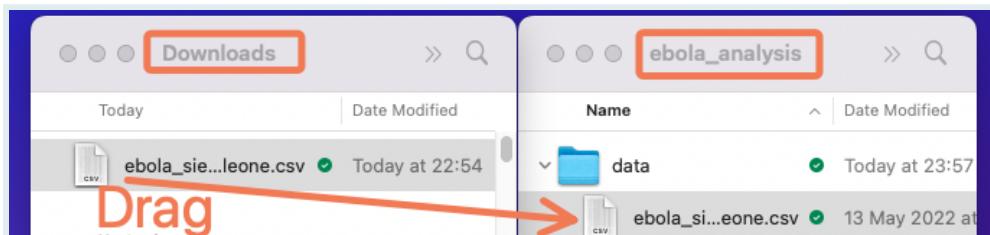


Vous accéderez ainsi au dossier Projet dans l'Explorateur de fichiers/Finder de votre ordinateur.

- Maintenant, déplacez le jeu de données Ebola que vous avez téléchargé dans la leçon précédente vers le sous-dossier "data" nouvellement créé.

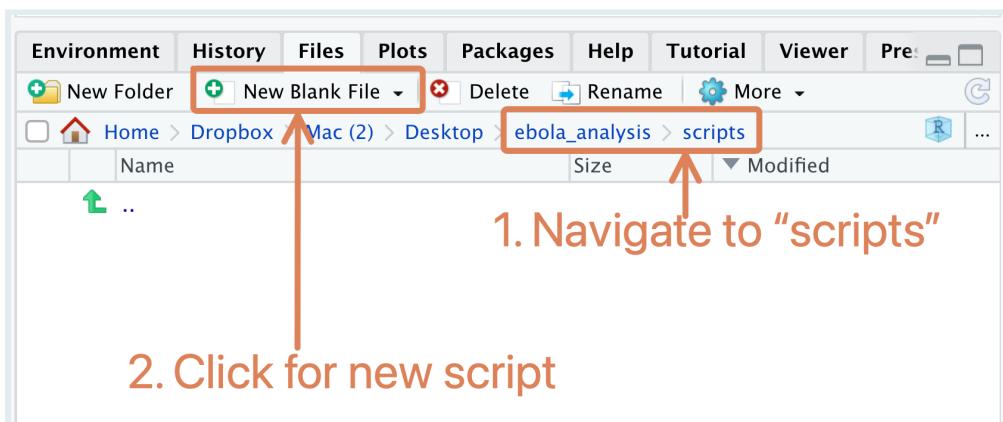
Voici à quoi ressemblerait le déplacement du fichier sur macOS :

##



### Création d'un script dans le dossier "scripts"

Ensuite, créez et enregistrez un nouveau script R dans le dossier "scripts". Vous pouvez appeler cela "analyse\_principale" ou quelque chose de similaire. Pour créer un nouveau script R dans un dossier, accédez d'abord à ce dossier dans l'onglet Files, puis cliquez sur le bouton "New Blank File" et sélectionnez "Script R" dans la liste déroulante :



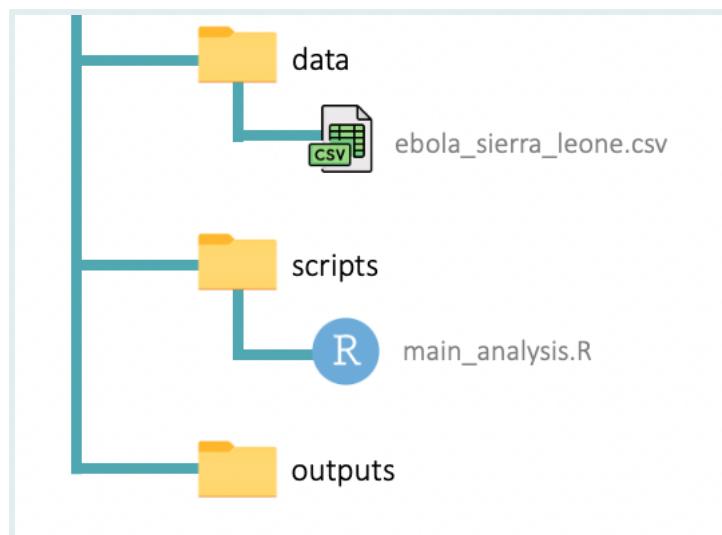
#### SIDE NOTE



Notez que ceci est différent de ce que vous avez fait jusqu'à présent lors de la création d'un nouveau script (avant, vous utilisiez l'option de menu "File > New File > New Script"). L'ancienne méthode est toujours valable; mais ce bouton "New Blank File" sera probablement plus rapide pour vous.

Excellent travail jusqu'à présent ! Maintenant, votre dossier project devrait avoir la structure ci-dessous, avec le jeu de données "ebola\_sierra\_leone.csv" dans le

dossier “data” et le script “analyse\_principale.R” (toujours vide) dans le dossier “scripts”:



Il s’agit d’un processus que vous devez suivre au début de chaque projet d’analyse de données : créez un projet RStudio, créez les sous-dossiers nécessaires et placez vos jeux de données et vos scripts dans les sous-dossiers appropriés. Cela peut être un peu pénible, mais cela sera payant à long terme.

---

Le reste de cette leçon vous apprendra comment effectuer votre analyse dans le contexte de cette configuration ou création de dossier. À la fin, vous aurez un flux global de données et de résultats qui ressemble au schéma ci-dessous :

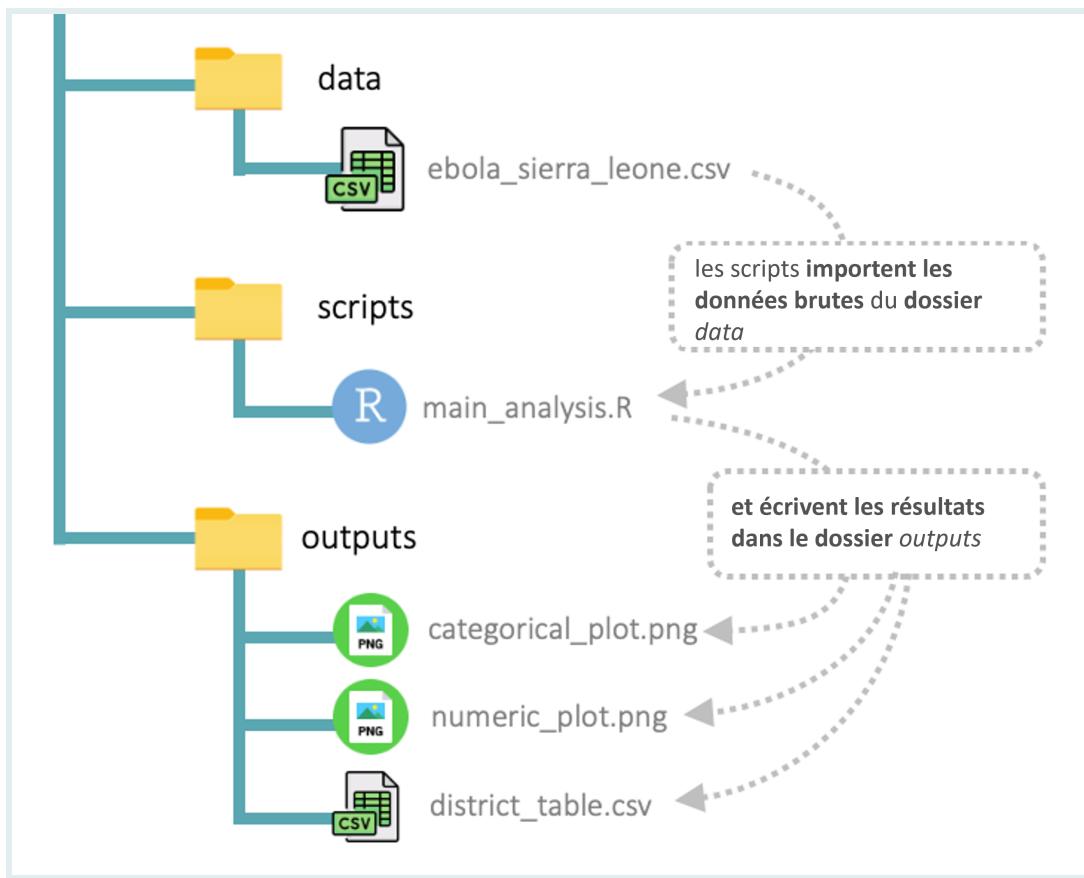


Figure : Flux de données dans un projet R. Les scripts du dossier “scripts” importent les données du dossier “data” et exportent les données et les graphiques vers le dossier “outputs”

Vous devriez vous référer à ce diagramme au fur et à mesure que vous avancez dans les sections ci-dessous, afin de vous aider à vous orienter.

### Importation de données depuis le dossier “data”

Nous utiliserons l'extrait de code ci-dessous pour illustrer le flux de données dans un projet. Copiez et collez cet extrait dans votre script “analyse\_principale.R” (mais ne l'exéutez pas encore). Le code reproduit certaines parties de l'analyse de la leçon sur l'exploration des données.

```

se Ebola Sierra Leone
Exemple-de-Nom Doe
01-01

Charger les packages ----
require(pacman) install.packages("pacman")
p_load(
  rperse,
  or,
  ctdf,
  # nouveau package que nous utiliserons bientôt

Charger les données ----
sierra_leone <- read_csv("") # DONNÉES EN ATTENTE ! NOUS LE METTRONS À JOUR CI-DESSOUS.

Afficher départements ----
d_tab <- tabyl(ebola_sierra_leone, district)
d_tab

Lister les variables nominales ----
n_graph<- show_plot(inspect_cat(ebola_sierra_leone))
n_graph

Lister les variables numériques ----
n_graph <- show_plot(inspect_num(ebola_sierra_leone))
n_graph

```

Commencez par exécuter la section “Charger les packages” pour installer et/ou charger les packages nécessaires.

Passez ensuite à la section “Charger les données”, qui ressemble à ceci :

```

Charger les données ----
sierra_leone <- read_csv("") # DONNÉES EN ATTENTE ! NOUS LE METTRONS À JOUR CI-DESSOUS.

```

Ici, vous souhaitez importer le jeu de données Ebola que vous avez précédemment placé dans le dossier “data” du projet. Pour ce faire, vous devez fournir le chemin du fichier de ce jeu de données comme premier argument de `read_csv()`.

Comme vous utilisez un projet RStudio, ce chemin peut être obtenu très facilement : placez votre curseur entre les guillemets dans la fonction `read_csv()` et appuyez sur la touche `Tab` de votre clavier. Vous devriez voir une liste des sous-dossiers disponibles dans votre projet. Quelque chose comme ça:

```

# Load data ----
ebola_sierra_leone <- read_csv("data/ebola_sierra_leone.csv") # DATA PENDING! WE WILL UPDATE THIS

# Which districts have the most cases?
district_vec <- ebola_sierra_leone$district
tabyl(district_vec)

```

The screenshot shows an RStudio interface. In the code editor, there is a line of code: `ebola\_sierra\_leone <- read\_csv("data/ebola\_sierra\_leone.csv")`. A red arrow points from the word "data" in this line to a file browser window on the right. The file browser shows a project structure with a "data" folder highlighted. Other folders in the project include "outputs" and "scripts".

Cliquez sur le dossier “data”, puis appuyez à nouveau sur Tab. Puisque vous n’avez qu’un seul fichier dans le dossier “data”, RStudio devrait automatiquement remplir son nom. Vous devriez maintenant voir :

```
ebola_sierra_leone <- read_csv("data/ebola_sierra_leone.csv")
```

Merveilleux! Exécutez maintenant cette ligne de code pour importer les données.

Si cela réussit, vous devriez voir apparaître les données dans l’onglet Environment de RStudio :



### Chemins relatifs

Le chemin que vous avez utilisé ici, “data/ebola\_sierra\_leone.csv”, est appelé un **chemin relatif**, car il est relatif à la **racine** (ou à la **base**) de votre projet.

#### KEY POINT



Comment R sait-il où se trouve la racine de votre projet ? C'est là qu'intervient le fichier .RProj. Ce fichier, qui réside dans le dossier “analyse\_ebola”, indique à R “ici ! Ici ! Je suis dans le dossier ‘analyse\_ebola’, c'est donc la racine !”. Ainsi, vous n'avez qu'à spécifier les composants de chemin qui sont *plus profonds* que cette racine.

Les projets RStudio et les chemins relatifs qu'ils vous permettent d'utiliser sont importants pour la reproductibilité. Les projets qui utilisent des chemins relatifs peuvent être exécutés sur l'ordinateur de n'importe qui, et le code d'importation et d'exportation devrait fonctionner sans problème. Cela signifie que vous pouvez envoyer à quelqu'un un dossier de projet RStudio et que le code devrait s'exécuter sur sa machine comme il s'est exécuté sur la vôtre !

**KEY POINT**

Ce ne serait pas le cas si vous utilisiez un chemin *absolu*, quelque chose comme “~/Bureau/mon\_analyse\_de\_donnees/formation\_r/ebola\_sierra\_leone.csv”, dans votre script. Les chemins absous donnent l’adresse complète d’un fichier et ne fonctionnent généralement pas sur l’ordinateur de quelqu’un d’autre, où les fichiers et les dossiers seront organisés différemment.



Notez que si vous utilisez RStudio Cloud, vous êtes *forcés* d’utiliser des chemins relatifs, car vous ne pouvez pas accéder au système de fichiers général de l’ordinateur virtuel ; vous ne pouvez travailler que dans des dossiers de projet spécifiques.

### Utilisation de `here::here()`

Comme vous avez pu le constater, les projets RStudio simplifient le processus d’importation des données et améliorent la reproductibilité de votre analyse, principalement parce qu’ils vous permettent d’utiliser des chemins relatifs.

Mais il y a une étape supplémentaire que nous recommandons lorsque vous utilisez des chemins relatifs : plutôt que de laisser votre chemin *nu*, enveloppez-le dans la fonction `here()` du package `{here}`.

Ainsi, dans la section d’importation de données de votre script, modifiez l’entrée de `read_csv()` de “`data/ebola_sierra_leone.csv`” à `here("data/ebola_sierra_leone.csv")`:

```
ebola_leone <- read_csv(here("data/ebola_sierra_leone.csv"))
```

Quel est l’intérêt d’envelopper le chemin dans `here()` ? Eh bien, techniquement, il n’y a pas de réel intérêt à le faire dans un script *R* ; le code d’importation fonctionne bien sans cela. Mais cela sera nécessaire lorsque vous commencerez à utiliser les scripts *RMarkdown* (qui vous seront bientôt présentés), car les chemins non enveloppés dans `here()` sont problématiques dans le contexte *RMarkdown*.

Donc, pour garder les choses cohérentes, nous vous recommandons toujours d’utiliser `here()` pour pointez vers des chemins, que ce soit dans un script *R* ou un script *RMarkdown*

### Exportation des données vers le dossier “outputs”

L’importation de données n’est pas le seul avantage de RStudio Projects ; l’exportation de données est également simplifiée lorsque vous utilisez Projects. Voyons maintenant ce qu’il en est.

Dans la section “Cas par départements” de votre script, vous devriez avoir :

```
ar départements ----  
:_tab <- tabyl(ebola_sierra_leone, district)  
:_tab
```

Exécutez ce code maintenant ; vous devriez obtenir le tableau suivant :

```
##          district    n percent  
##            Bo      2   0.010  
##      Kailahun 155   0.775  
##        Kambia     1   0.005  
##       Kenema     34   0.170  
##         Kono      2   0.010  
##      Port Loko    2   0.010  
##  Western Urban    4   0.020
```

Maintenant, imaginez que vous souhaitez exporter ce tableau au format CSV. Ce serait bien s'il existait un dossier spécifique désigné pour ces exportations. Eh bien, c'est le cas! Il s'agit du dossier “outputs” que vous avez créé précédemment. Exportons votre tableau dans ce dossier. Saisissez le code ci-dessous (mais ne l'exécutez pas encore):

```
sv(x = district_tab, file = "")
```

Avec la fonction `write_csv()`, vous allez “écrire” (ou “enregistrer”) le tableau `district_tab` sous forme de fichier CSV.

L'argument `x` de `write_csv()` prend en compte l'objet à sauvegarder (dans ce cas `district_tab`). Et l'argument `file` prend en compte le chemin du fichier cible. Ce chemin de fichier cible peut être un simple chemin relatif : “outputs/departement\_tableau.csv”. (Et, comme mentionné précédemment, nous devrions envelopper le chemin dans `here()`.) Saisissez ceci et exécutez-le maintenant :

```
sv(x = district_tab, file = here("outputs/departement_tableau.csv"))
```

Le chemin “outputs/departement\_tableau.csv” indique à `write_csv()` d'enregistrer le graphique dans un fichier CSV nommé “departement\_tableau” dans le dossier “outputs” du projet.

#### SIDE NOTE

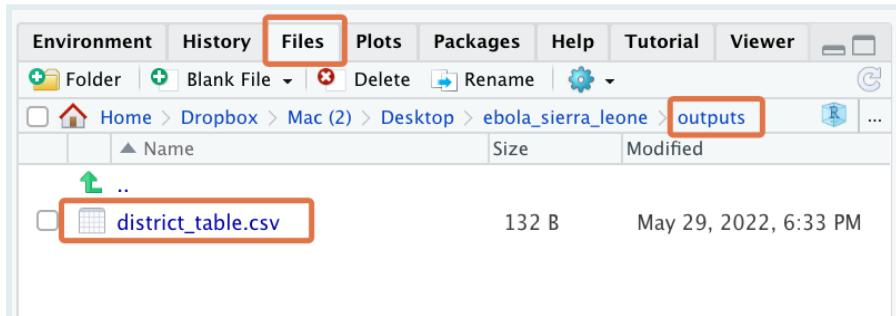


Vous pouvez remplacer “departement\_tableau.csv” par tout autre nom approprié, par exemple “tableau de freq par départements.csv” :

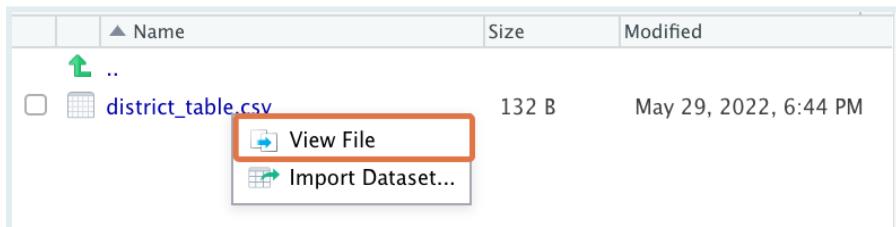
**SIDE NOTE**

```
sv(x = district_tab, file = here("outputs/tableau de freq par  
départements.csv"))
```

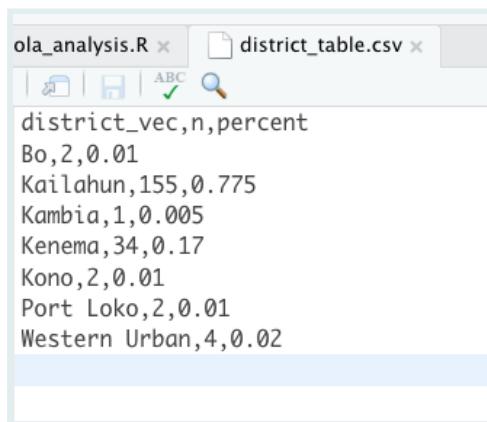
Bon travail! Maintenant, si vous allez dans l'onglet Files et naviguez jusqu'au dossier outputs de votre projet, vous devriez voir ce fichier nouvellement créé :



Vous pouvez cliquer sur le fichier pour le visualiser dans RStudio en tant que fichier CSV brut :



Cela devrait faire apparaître une fenêtre de visualisation RStudio :



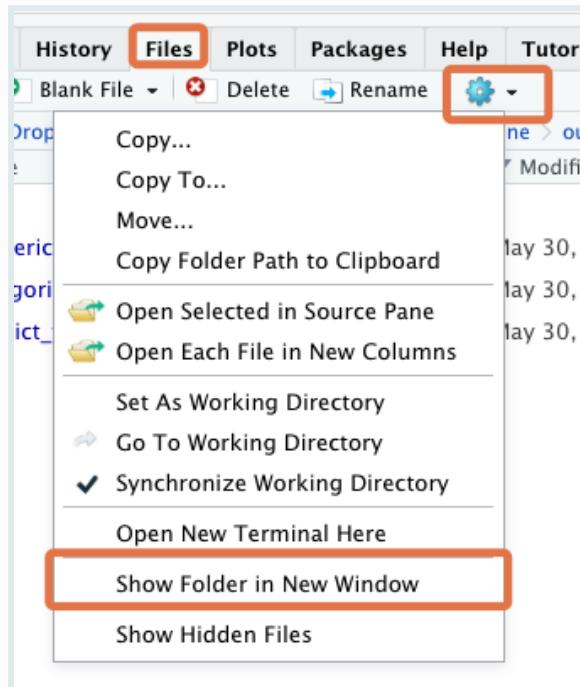
Si vous souhaitez plutôt visualiser le CSV dans Microsoft Excel, vous pouvez accéder au même fichier dans le Finder/Explorateur de fichiers de votre ordinateur et double-cliquer dessus à partir de là.

**REMINDER**

### REMINDER



Pour localiser votre dossier de projet dans le Finder/Explorateur de fichiers de votre ordinateur, allez dans l'onglet “Files”, cliquez sur l'icône d'engrenage, puis cliquez sur “Show Folder in New Window”.



Si vous êtes sur RStudio cloud, vous ne pourrez pas visualiser le CSV dans Microsoft Excel tant que vous ne l'aurez pas “exporté”. Utilisez l'option de menu “Export” dans l'onglet Files. Si cette option n'est pas immédiatement visible, cliquez sur l'icône d'engrenage pour afficher les options “More”, puis faites défiler l'écran pour trouver l'option “Export”.

## Remplacement des données

Si vous avez besoin de mettre à jour le CSV de sortie, vous pouvez simplement réexécuter la fonction `write_csv()` avec l'objet de données mis à jour.

Pour tester cela, remplacez la section “Cas par département” de votre script par le code suivant. Il utilise la fonction `arrange()` pour organiser le tableau dans l'ordre du nombre de cas, `n` :

```
ar département ----  
:_tab <- tabyl(ebola_sierra_leone, district)  
:_tab_arranged <- arrange(district_tab, -n)  
:_tab_arranged
```

( `-n` signifie “trier par ordre décroissant de la variable `n`” ; nous vous présenterons correctement la fonction `arrange` plus tard.)

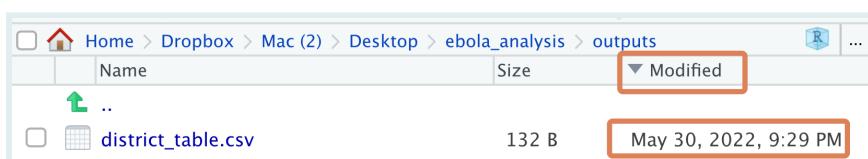
La sortie devrait être :

```
##      district   n percent
##    Kailahun 155 0.775
##    Kenema    34 0.170
##  Western Urban    4 0.020
##      Bo       2 0.010
##      Kono    2 0.010
##  Port Loko    2 0.010
##    Kambia    1 0.005
```

Vous pouvez maintenant remplacer l’ancien fichier “`departement_tableau.csv`” en réexécutant la fonction `write_csv` avec l’objet `district_tab_arrange` :

```
sv(x = district_tab_arrange, file = here("outputs/departement_tableau.csv"))
```

Pour vérifier que le jeu de données a bien été mis à jour, observez la section “Modified” dans l’onglet Files de RStudio :



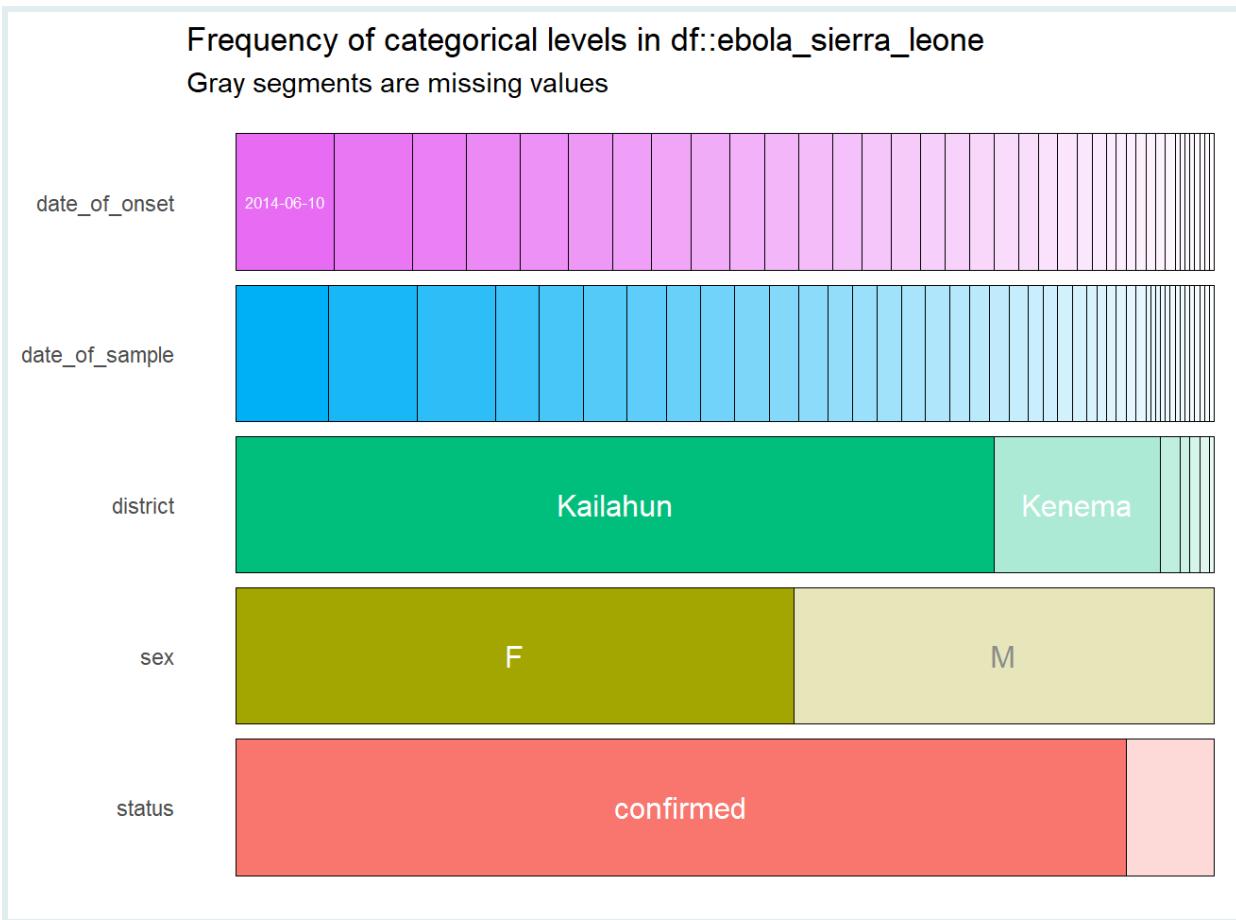
## Exportation des graphiques vers le dossier “outputs”

Enfin, regardons l’exportation de graphiques dans le contexte d’un projet RStudio.

Dans la section “Visualiser les variables nominales” de votre script, vous devez avoir :

```
liser les variables nominales ---
i_graph<- show_plot(inspect_cat(ebola_sierra_leone))
i_graph
```

L’exécution de ces lignes de code devrait vous donner ce résultat :



Sous ces lignes, tapez la commande `ggsave()` ci-dessous (mais ne l'exécutez pas encore) :

```
filename = "", plot = vars_nom_graph)
```

Cette commande utilise la fonction `ggsave()` pour exporter la figure `vars_nom_graph`. L'argument `plot` de `ggsave()` prend en compte l'objet à enregistrer (dans ce cas `vars_nom_graph`), et l'argument `filename` prend en compte le chemin du fichier cible pour le graphique.

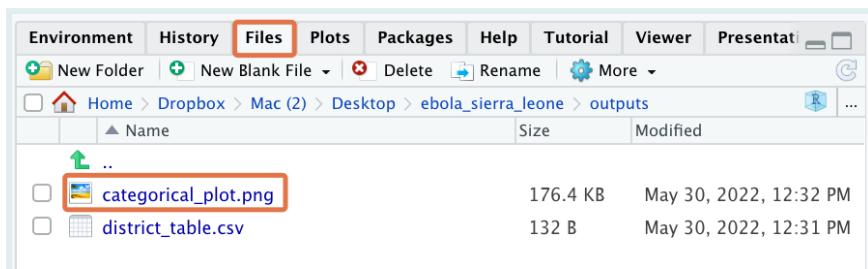
Comme vous l'avez vu lors de l'exportation de données, ce chemin de fichier cible est assez simple car vous travaillez dans un projet RStudio. Dans ce cas, vous avez :

```
filename = "outputs/graphique_nominal.png", plot = vars_nom_graph)
```

Exécutez maintenant cette commande `ggsave()`. Le chemin “outputs/graphique\_nominal.png” indique à `ggsave()` d'enregistrer le graphique en tant que fichier PNG nommé “graphique\_nominal” dans le dossier “outputs” du projet.

Pour voir ce graphique nouvellement enregistré, accédez à l'onglet Files. Vous pouvez cliquer dessus pour l'ouvrir avec le visualiseur d'images par défaut de votre

ordinateur :



Notez également que la fonction `ggsave()` vous permet d'enregistrer des graphiques dans plusieurs formats d'image. Par exemple, vous pourriez plutôt écrire :

```
filename = "outputs/graphique_nominal.pdf", plot = vars_nom_graph)
```

pour enregistrer le graphique au format PDF. Exécutez `?ggsave` pour voir quels autres formats sont possibles.

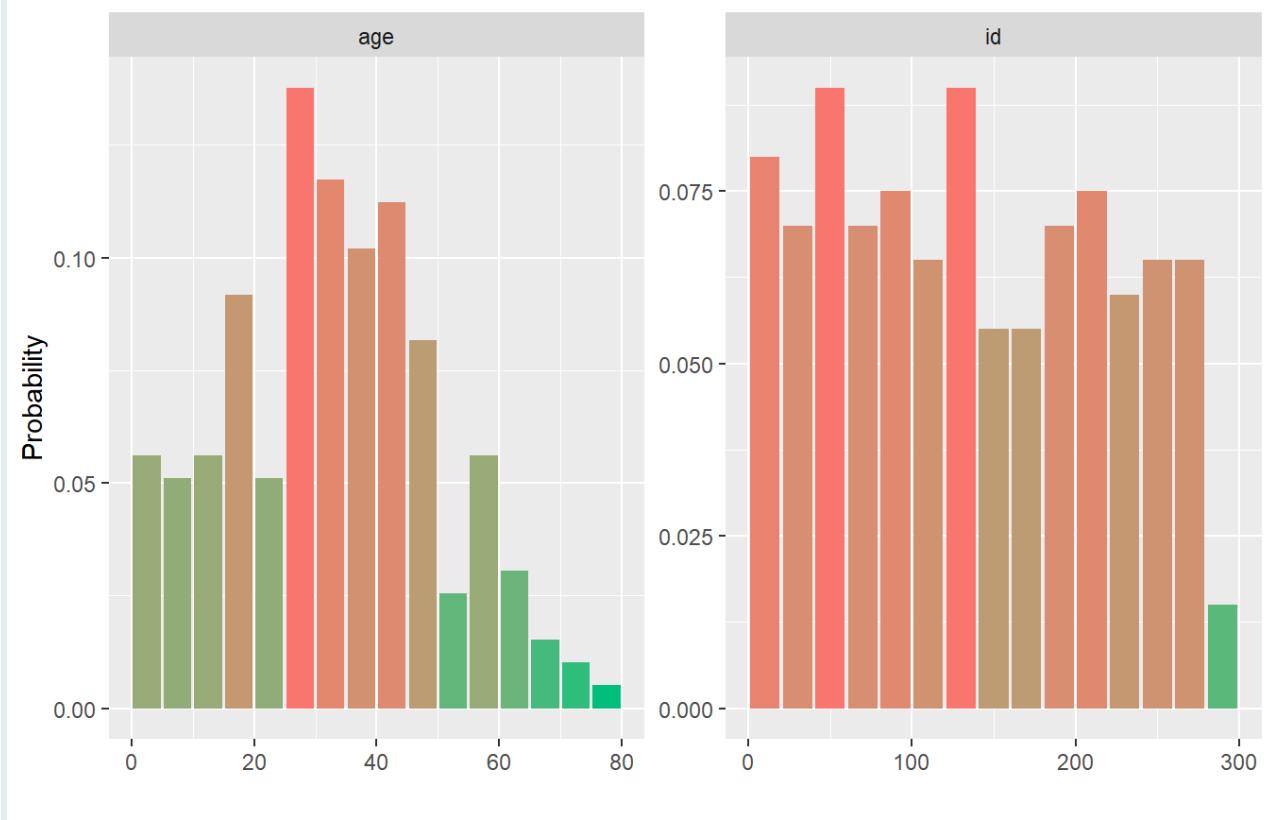
---

Exportons maintenant le deuxième graphique, le résumé numérique. Dans la section de votre script intitulée “Visualiser les variables numériques”, vous devriez avoir :

```
liser les variables numériques ----  
graph <- show_plot(inspect_num(ebola_sierra_leone))  
graph
```

L'exécution de ces lignes de code devrait vous donner ce résultat :

## Histograms of numeric columns in df::ebola\_sierra\_leone



Pour exporter ce graphique, saisissez et exécutez le code suivant :

```
filename = "outputs/graph_numerique.png", plot = vars_num_graph)
```

Merveilleux!

## Partager un projet

Les projets sont également parfaits pour partager votre analyse avec des collaborateurs.

Vous pouvez compresser votre dossier project et l'envoyer à un collègue par e-mail ou via un service de partage de fichiers comme Dropbox. Le collègue peut alors décompresser le dossier, cliquer sur le fichier .Rproj pour ouvrir le projet dans RStudio, et refaire et modifier toutes vos étapes d'analyse.

C'est une configuration convenable, mais l'envoi de projets dans les deux sens n'est pas toujours idéal pour une collaboration à long terme. C'est pourquoi les analystes expérimentés utilisent une technologie appelée *git* pour collaborer sur des projets. Mais ce sujet est un peu trop avancé pour ce cours ; nous le détaillerons dans un prochain cours. Si vous êtes impatient, vous pouvez consulter ce chapitre de livre : [https://intro2r.com/github\\_r.html](https://intro2r.com/github_r.html)

## Conclusion

Toutes nos félicitations! Vous savez maintenant comment configurer et utiliser les projets RStudio !

J'espère que vous voyez l'intérêt d'organiser vos scripts d'analyse, vos données et vos résultats de cette manière. Les projets sont une manière cohérente de structurer vos analyses et facilitent la réexamen, la révision et le partage de votre travail. Ils constitueront la base d'une grande partie de votre travail en tant qu'analyste de données à l'avenir.

C'est tout pour le moment. Rendez-vous à la prochaine leçon.

## Contributeurs

Les membres de l'équipe suivants ont contribué à cette leçon :



### KENE DAVID NWOSU

Data analyst, the GRAPH Network  
Passionate about world improvement

---



### LAURE NGUEMO

Data Science Education Officer  
Gets very excited at the mention of data, especially health related data

---

## Références

Certains éléments de cette leçon ont été adaptés à partir des sources suivantes :

- Wickham, H., & Grolemund, G. (n.d.). *R pour science des données. 8 Workflow : projets | R pour Data Science*. Extrait le 31 mai 2022 de <https://r4ds.had.co.nz/workflow-projects.html>

This work is licensed under the [Creative Commons Attribution Share Alike](#) license.



# Notes de leçon | R Markdown

GRAPH Network & OMS, soutenu par le Fonds Mondial

January 2024

Ce cours a été créé par le Réseau GRAPH, une organisation à but non lucratif basée à l'Institut de santé globale de l'Université de Genève, en collaboration avec l'Organisation mondiale de la Santé, dans le cadre d'une subvention du Fonds mondial pour créer des cours afin de renforcer les capacités nationales en matière d'analyse épidémiologique.



Introduction .....
Objectifs d'apprentissage .....
Configuration du projet .....
Créer un nouveau document .....
L'en-tête Rmarkdown (YAML) .....
Mode Visuel vs Source .....
Syntaxe Markdown .....
Personnalisation du document généré .....
Blocs de code R .....
Sortie de bloc en ligne vs dans la console .....
Options de bloc de code R .....
Code entre les lignes .....
Affichage des tableaux .....
Modèles de documents .....
Ressources .....

---

## Introduction

Le package {rmarkdown} vous permet de générer dynamiquement des documents en mélangeant du texte formaté et des résultats produits par le code R. Les documents générés peuvent être en HTML, PDF, Word, et bien d'autres. C'est donc un outil très pratique pour exporter, communiquer et diffuser des résultats d'analyse.

Il existe un livre entier sur Rmarkdown, donc nous ne pouvons couvrir ici que quelques éléments essentiels.

Ce document a lui-même été généré à partir de fichiers R Markdown.

---

## Objectifs d'apprentissage

- Vous pouvez créer et compiler un document Rmarkdown contenant du code et du texte libre.
- Vous pouvez générer des documents dans de multiples formats incluant HTML, PDF, Word, Powerpoint et flexdashboards.
- Vous comprenez la syntaxe de base de Markdown.
- Vous pouvez utiliser les options de chunk R, incluant *eval*, *echo*, et *message*.
- Vous connaissez la syntaxe pour le code R en ligne.
- Vous reconnaissiez quelques packages utiles pour la mise en forme de tableaux dans Rmarkdown.

- Vous comprenez comment utiliser le package {here} pour forcer les fichiers RMarkdown à utiliser le dossier du projet comme répertoire de travail.

## Configuration du projet

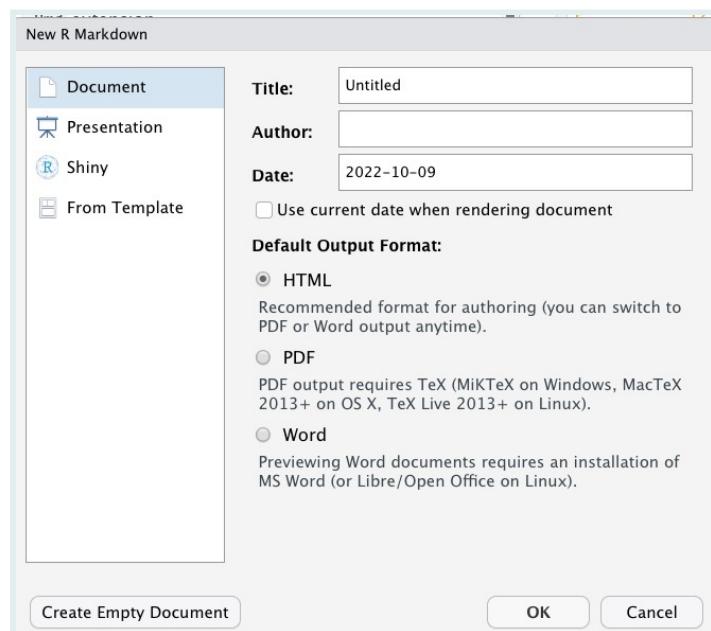
Dans RStudio, cliquez sur le menu *File*, puis sélectionnez *New project...*. Cliquez ensuite sur *New repository*. Donnez un nom à votre projet, et sélectionnez un répertoire dans lequel le placer. (Assurez-vous de vous souvenir de l'endroit vous l'avez mis !) Lorsque ces champs sont remplis, cliquez sur *Create Project*.

Ensuite, nous allons créer quelques dossiers à l'intérieur du projet. Allez dans l'onglet *Files*, et cliquez sur *New folder*. Nommez ce dossier “data”, et cliquez sur *OK*. C'est ici que vous placerez les données relatives à ce projet. Créez un autre dossier nommé “rmd”. Les documents R Markdown iront ici.

## Créer un nouveau document

Un document R Markdown est un simple fichier texte sauvegardé avec l'extension `.Rmd`.

Dans RStudio, vous pouvez créer un nouveau document en allant dans le menu *File* puis en choisissant *New file* puis *R Markdown...*. La première fois que vous créez un document R Markdown, il est possible que l'on vous demande d'installer plusieurs packages. Allez-y et installez ces packages. Une fois que RStudio a les packages appropriés, la boîte de dialogue suivante apparaît :



Pour le moment, vous pouvez laisser toutes les valeurs par défaut et cliquer sur OK. Un fichier avec un contenu d'exemple est alors affiché.

Essayez de modifier une partie du texte dans le fichier. Notez qu'il est constitué de texte libre et de sections de code.

Enregistrez votre fichier avec Cmd/Ctrl + S, en vous rappelant de lui donner l'extension .Rmd. Par exemple "analyse\_ebola.Rmd". Assurez-vous de le sauvegarder dans le dossier "rmd" que vous venez de créer.

Vous pouvez maintenant essayer de compiler le document en cliquant sur le bouton "knit" en haut à droite :



Ceci va créer une sortie HTML qui ressemble à ceci :

# Untitled

2022-10-09

## 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>.

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:

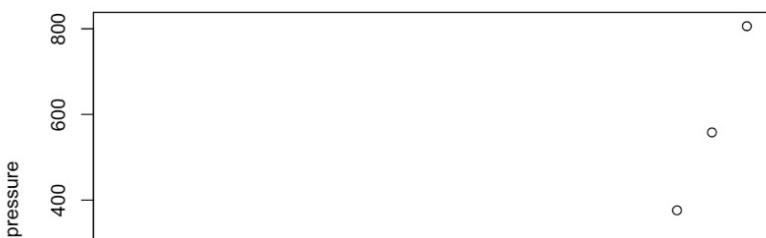
```
summary(cars)
```

```
##      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:



Ce nouveau fichier compilé est stocké dans le même répertoire que votre fichier Rmd. Il a le même nom, sauf qu'il se termine par ".html" au lieu de ".rmd".

HTML signifie Hyper Text Markup Language et est le format utilisé pour la plupart des documents sur le web.

---

## L'en-tête Rmarkdown (YAML)

Revenons maintenant au reste du Rmd pour l'examiner partie par partie.

La première partie du document est son \*en-tête\*. (Il est aussi appelé “YAML”, ce qui signifie “Yet another markup language”.) (Le nom se veut humoristique.)

```
---
```

```
title: "Sans titre"
output: html_document
date: "2022-10-09"
---
```

L'en-tête YAML doit être placé tout au début du document, délimité par trois tirets (---) avant et après.

Cet en-tête contient les métadonnées du document, comme son titre, son auteur, sa date, ainsi qu'une multitude d'options possibles qui vous permettront de configurer ou personnaliser l'ensemble du document et son rendu. Ici, par exemple, la ligne `output: html_document` indique que le document généré doit être au format HTML.

Nous pouvons modifier le texte `html_document` pour essayer d'autres formats.

Tout d'abord, vous pouvez faire ceci:

Avec la sortie fixée à “`word_document`”, nous obtenons quelque chose comme ceci :

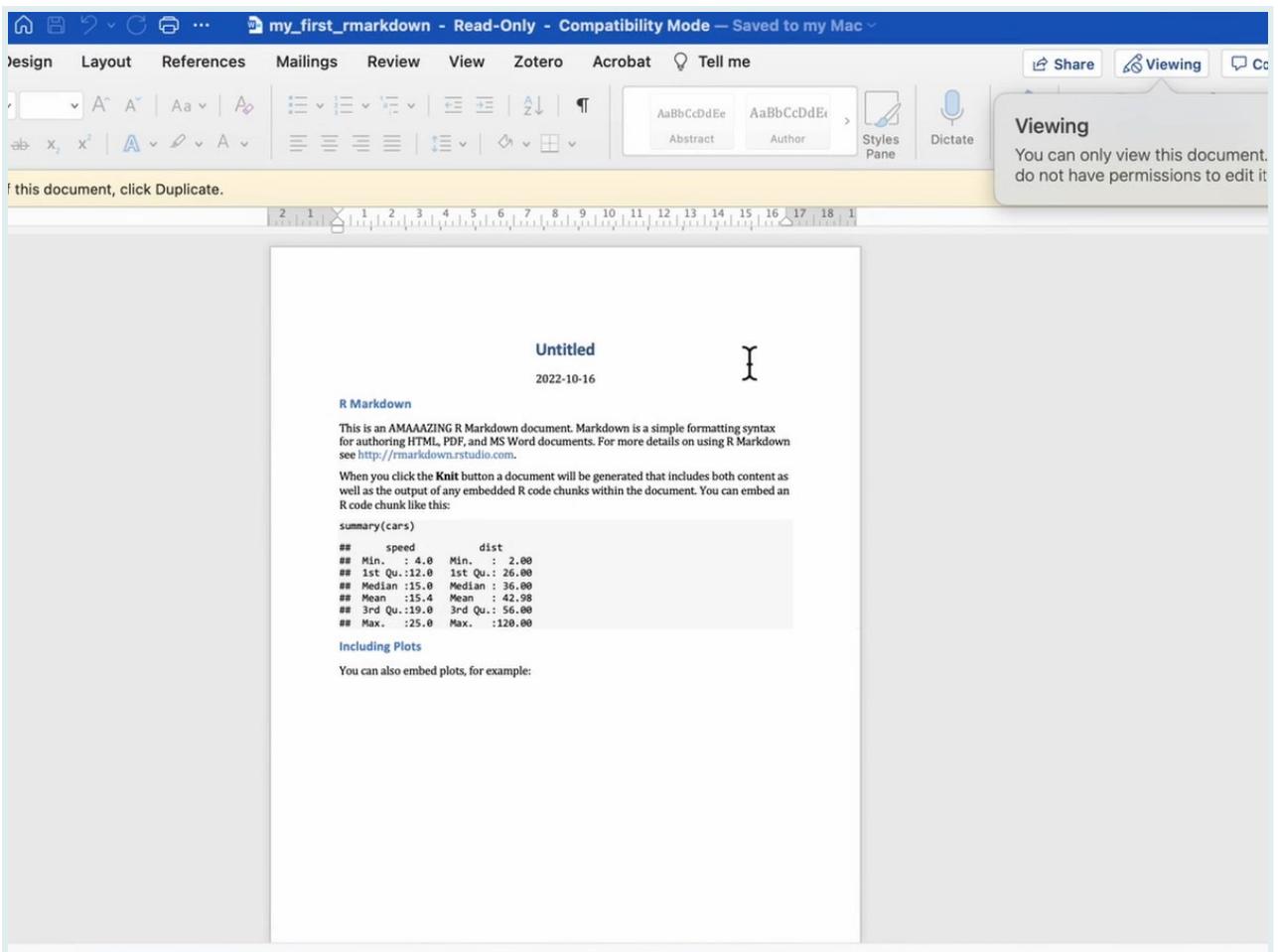


Image du document R markdown ouvert dans le programme Microsoft Word

Notez que cela crée une version ".docx" de notre document dans le dossier "rmd".

Avec la sortie définie sur "powerpoint\_presentation", cela donne ceci :

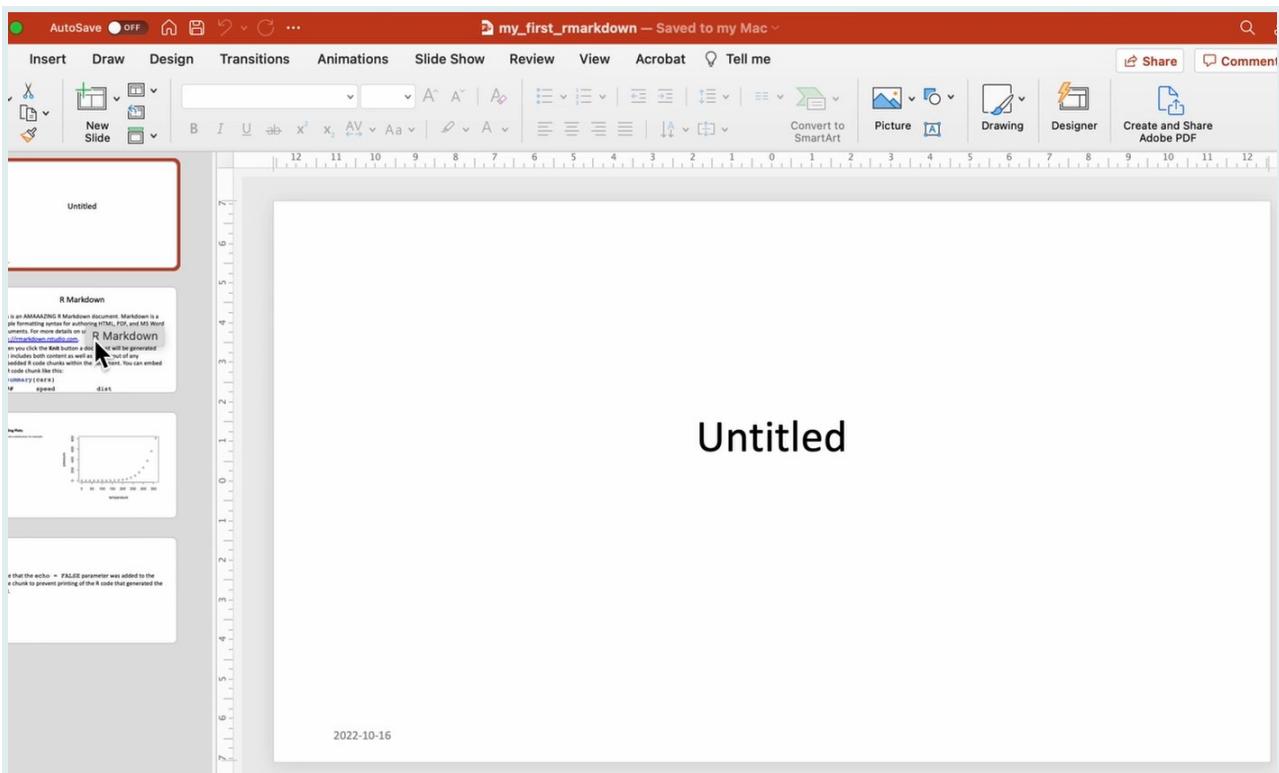
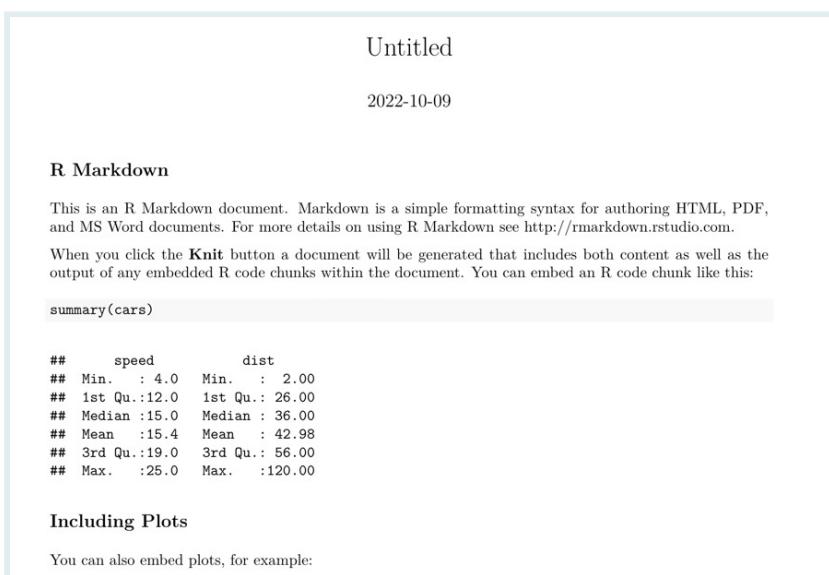


Image du document R markdown ouvert dans le programme Microsoft Powerpoint

Si nous changeons le paramètre de sortie pour “pdf\_document”, nous pouvons obtenir le même document au format PDF (cela peut vous demander d’installer tinytex sur votre ordinateur, voir ci-dessous) :



Pour la génération de PDF, vous devez avoir une installation LaTeX fonctionnelle sur votre système. Sinon, l’extension `tinytex` de Yihui Xie vise à faciliter l’installation d’une distribution LaTeX minimale quel que soit le système d’exploitation de votre machine. Pour l’utiliser, vous devez d’abord installer l’extension avec `install.packages('tinytex')`, puis exécuter la commande suivante dans la console

(attendez-vous à un téléchargement d'environ 200 Mo) : `tinytex::install_tinytex()`  
Plus d'informations sur [le site web de tinytex](#).

Il existe également un format de fichier appelé "prettydoc". Pour l'essayer, saisissez `install.packages('prettydoc')` dans la console et appuyez sur *entrée*. Le format de sortie pour prettydoc est un peu différent des trois précédents que nous avons vus, vous devez saisir `prettydoc::html_pretty` dans la section `output`. Lorsque vous compilez un prettydoc, vous devriez voir quelque chose comme ceci :

The screenshot shows a browser window titled "my\_first\_rmarkdown.html". The page has a dark blue header with the title "Untitled" and the date "2022-10-16". The main content area contains the following text:

## R Markdown

This is an AMAAAZING 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>.

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)
```

Image du document R markdown en tant que prettydoc

Nous pouvons même obtenir un format de tableau de bord simple. Tout d'abord, nous devons `install.packages ('flexdashboard')`. Puis si nous définissons le `output` sur `flexdashboard::flex_dashboard`, et compilons, nous obtenons quelque chose comme ceci :

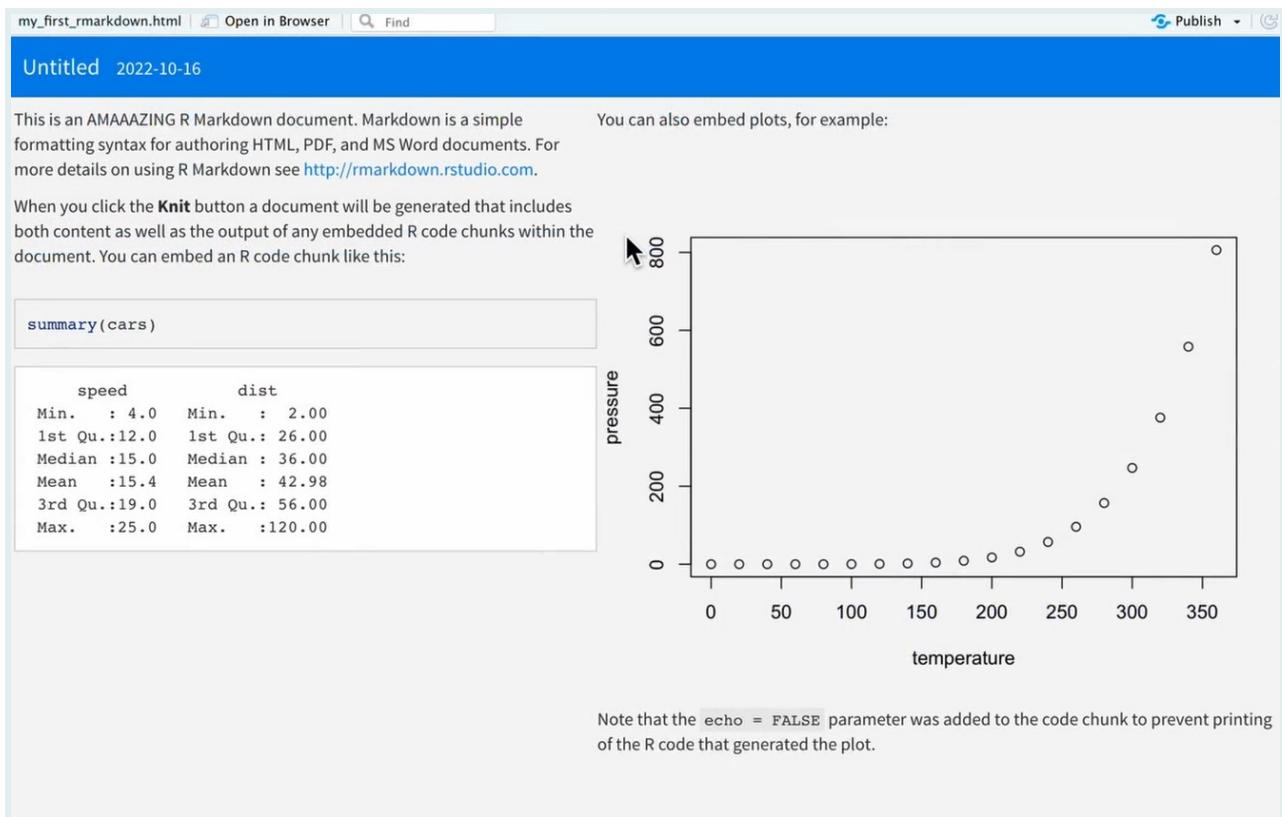


Image du document R markdown en tant que flexdashboard

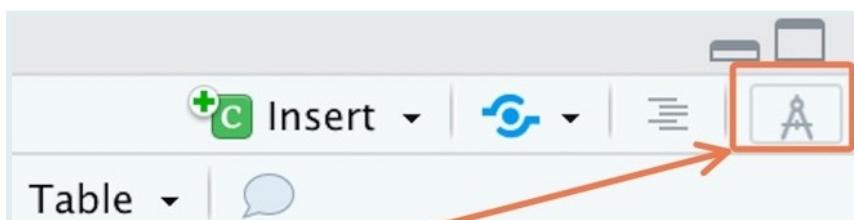
Notez qu'il n'a pas encore d'onglets. Pour créer des onglets dans un flexdashboard, changez certains de vos dièses doubles ## en dièses simples #. Cela changera le style d'en-tête pour ces sections, et fera que flexdashboard les rende en tant qu'onglets au lieu de sous-titres.

De nombreux autres formats sont possibles, et nous vous encourageons à explorer par vous-même !

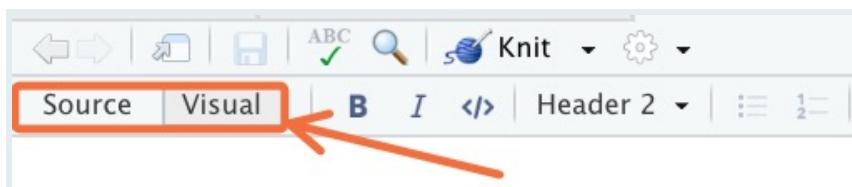
## Mode Visuel vs Source

Les documents Rmarkdown peuvent être édités soit en mode "Source" soit en mode "Visuel".

Vous pouvez passer en mode visuel pour un document donné à l'aide des barres d'outils. Pour les anciennes versions de RStudio, vous pouvez avoir un bouton A en haut à droite de la barre d'outils du document



Pour les versions plus récentes de RStudio, il y a une paire de boutons pour basculer entre les modes :



Quelle est la différence entre ces deux modes ?

En mode source, vous voyez la syntaxe brute de Markdown.

```
## 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>.

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:

```{r cars}
summary(cars)
```
```

Markdown est un ensemble simple de conventions permettant d'ajouter de la mise en forme à du texte brut. Par exemple, pour mettre du texte en italique, vous l'enveloppez dans une astérisque \*texte ici\*, et pour commencer un nouvel en-tête, vous utilisez le signe dièse #. Nous allons apprendre ces éléments en détail ci-dessous.

Mais en mode visuel, vous voyez plutôt une vue semblable à celle de Microsoft Word :

## 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>.

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:

```
{r cars}
summary(cars)
```

avec une barre d'outils pour une mise en forme facile :



Cela signifie que vous n'avez pas à vous souvenir de la syntaxe des éléments Markdown. Par exemple, si vous voulez mettre une section de texte en gras, vous pouvez simplement mettre en surbrillance cette partie de texte et cliquer sur le bouton "gras" dans la barre d'outils.

Maintenant, bien que le mode visuel soit beaucoup plus facile à utiliser, nous vous apprendrons ici la syntaxe Markdown pour trois raisons :

- Le mode visuel est parfois une source de bugs, et pour déboguer cela, vous devrez passer en mode source
- Comprendre la syntaxe Markdown est utile en dehors de RMarkdown
- Le mode visuel n'est pas disponible dans le mode collaboratif de RStudio, que vous pourriez être amené à utiliser

---

## Syntaxe Markdown

Dans l'onglet "Help", si vous recherchez "Markdown Quick Reference", vous pourrez trouver une large variété d'options RMD à votre disposition.

Vous pouvez définir des titres de différents niveaux en commençant une ligne avec un ou plusieurs # :

```
# Titre de niveau 1  
## Titre de niveau 2  
### Titre de niveau 3
```

Le corps du document est constitué de texte qui suit la syntaxe *Markdown*. Un fichier Markdown est un fichier texte qui contient des balises légères qui aident à définir les niveaux d'en-tête ou à mettre en forme du texte. Par exemple, le texte suivant :

Ceci est du texte avec de \*l'*italique*\* et du \*\*gras\*\*.

Vous pouvez définir des listes à puces :

- premier élément
- deuxième élément

Va générer le texte formaté suivant :

Ceci est du texte avec de *l'italique* et du **gras**.

Vous pouvez définir des listes à puces :

- premier élément
- deuxième élément

Notez que vous avez besoin d'espaces avant et après les listes, ainsi que de garder les éléments énumérés sur des lignes séparées, sinon ils vont s'agglutiner ensemble au lieu de former une liste.

Nous voyons que les mots placés entre astérisques sont mis en italique, les lignes qui commencent par un tiret sont transformées en liste à puces, etc.

La syntaxe Markdown permet d'autres mises en forme, comme la possibilité d'insérer des liens ou des images. Par exemple, le code suivant :

[Exemple de lien](<https://example.com>)

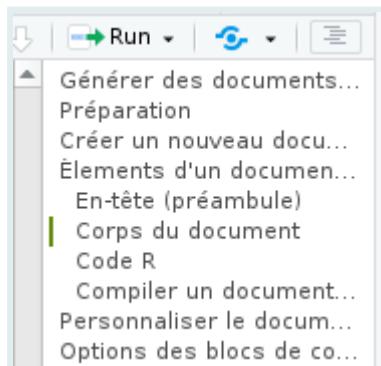
... donnera le lien suivant :

### Exemple de lien

Nous pouvons également intégrer des images. Si vous êtes en mode *Source*, tapez :

[ce que vous voulez comme sous-titre](images/nom\_image.jpg), en remplaçant "ce que vous voulez comme sous-titre" (ça peut aussi être vide), "images" par le nom du dossier d'images dans votre projet, et "nom\_image.jpg" par le nom de l'image que vous voulez utiliser. Bien sûr, c'est plus facile à faire en mode *Visuel*. À partir de là, vous pouvez simplement ouvrir le dossier qui contient votre image sur votre ordinateur et faire glisser l'image du dossier sur la page que vous construisez. Ou vous pouvez placer le curseur là où vous voulez que l'image soit, cliquez sur le bouton marqué d'une icône "image", suivez les invites, et insérez votre image où se trouve le curseur. Notez que cela créera également un dossier "images" dans votre projet (s'il n'existe pas déjà) et mettra le fichier image dans le dossier "images".

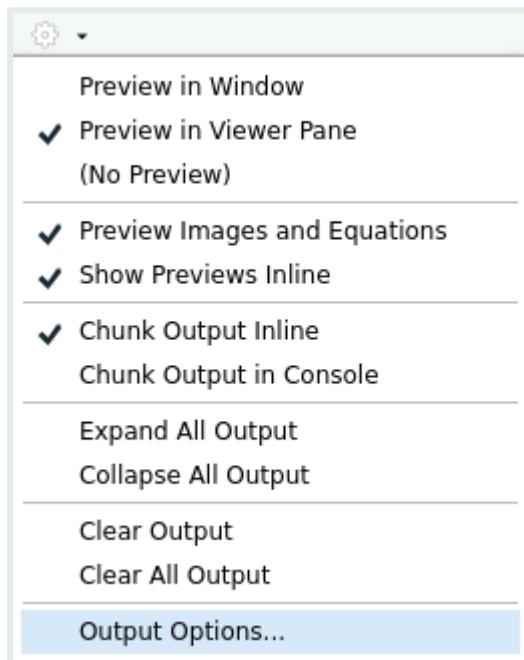
Lorsque des titres ont été définis, si vous cliquez sur l'icône *Show document outline* complètement à droite de la barre d'outils associée au fichier R Markdown, une table des matières générée automatiquement à partir des titres est affichée et vous permet de naviguer facilement dans le document :



TOC dynamique

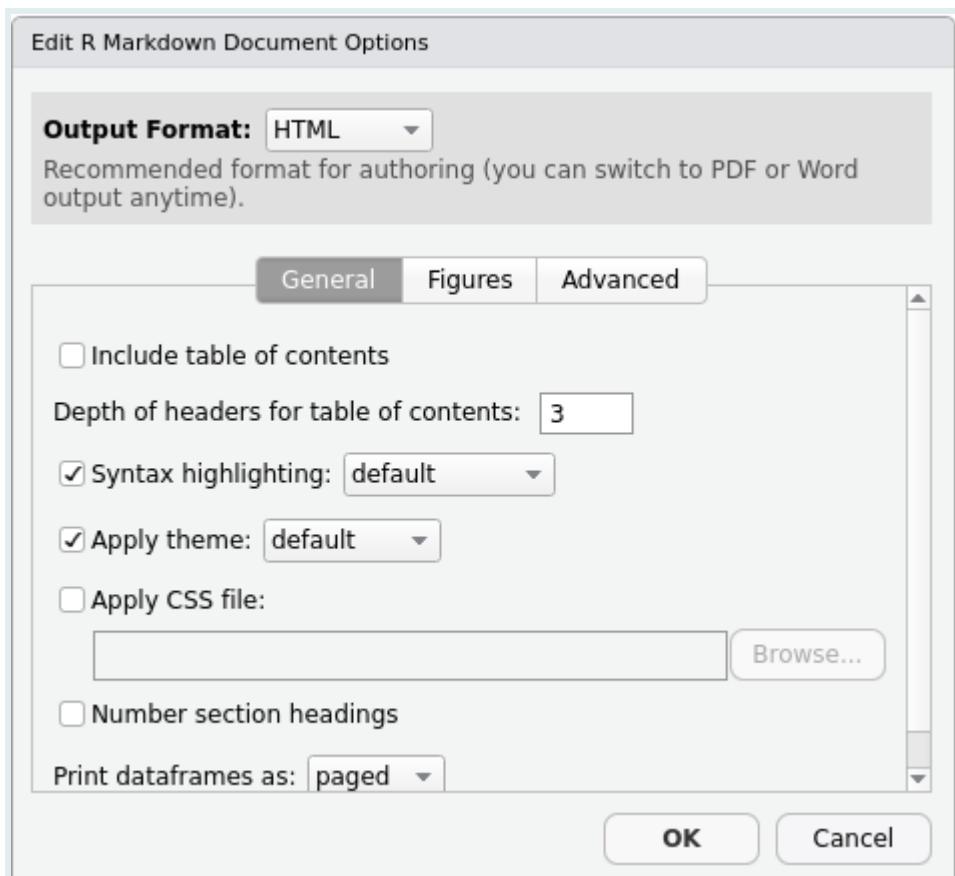
## Personnalisation du document généré

La personnalisation du document généré se fait en modifiant les options dans le préambule du document. Cependant, RStudio offre une petite interface graphique pour modifier ces options plus facilement. Pour cela, cliquez sur l'icône d'engrenage à droite du bouton *Knit* et choisissez *Output Options...*



Options de sortie R Markdown

Une boîte de dialogue apparaît vous permettant de sélectionner le format de sortie souhaité et, selon le format, différentes options :



Boîte de dialogue des options de sortie R Markdown

Pour le format HTML par exemple, l'onglet *General* vous permet de spécifier si vous voulez une table des matières, sa profondeur, les thèmes à appliquer pour le document et la coloration syntaxique des blocs R, etc. L'onglet *Figures* permet de modifier les dimensions par défaut des graphiques générés.

Lorsque vous modifiez des options, RStudio modifiera en fait le préambule de votre document. Ainsi, si vous choisissez d'afficher une table des matières et de changer le thème de coloration syntaxique, votre en-tête deviendra quelque chose comme :

```
---
```

```
title: "Révision R Markdown"
output:
  html_document:
    highlight: kate
    knock: yes
---
```

Vous pouvez modifier les options directement en éditant le préambule.

Notez qu'il est possible de spécifier des options différentes selon le format, par exemple :

```
---
```

```
title: "Révision R Markdown"
```

```
output:  
  html_document:  
    highlight: kate  
    knock: yes  
  pdf_document:  
    fig_caption: yes  
    highlight: kate  
---
```

La liste complète des options possibles est présente sur [le site de documentation officiel](#) (très complet et bien fait) ainsi que dans la fiche aide-mémoire et le guide de référence, accessibles depuis RStudio via le menu *Help* puis *Cheatsheets*.

## Blocs de code R

En plus du texte libre au format Markdown, un document R Markdown contient, comme son nom l'indique, du code R. Celui-ci est inclus dans des blocs (*chunks*) écrits de la façon suivante en mode *Source* :

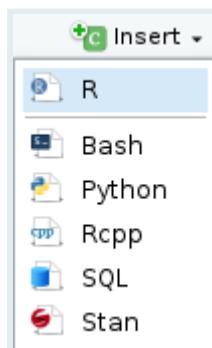
```
code_r <- 2+2
```

Ce qui produira ce qui suit en mode *Visuel* :

```
code_r <- 2+2
```

Comme cette séquence de caractères n'est pas très facile à saisir, vous pouvez utiliser le menu *Insert* de RStudio et choisir *R*[^3], ou utiliser le raccourci clavier Commande+Option+i sur Mac ou Ctrl+Alt+i sur Windows.

Notez qu'il est possible d'utiliser d'autres langages dans les blocs de code.



Menu d'insertion de bloc de code

Dans RStudio, les blocs de code R sont généralement affichés avec une couleur de fond légèrement différente pour les distinguer du reste du document.

Lorsque votre curseur est dans un bloc, vous pouvez saisir le code R que vous voulez et l'exécuter avec Commande + Entrée. Vous pouvez également exécuter tout le code

contenu dans un bloc en cliquant sur le bouton vert “lecture” en haut à droite du bloc de code.

### Sortie de bloc en ligne vs dans la console

Dans RStudio, par défaut, les résultats d'un bloc de code (texte, tableau ou graphique) sont affichés directement *dans* la fenêtre d'édition du document, permettant de les visualiser et de les conserver facilement pendant la durée de la session.

Ce comportement peut être modifié en cliquant sur l'icône d'engrenage dans la barre d'outils et en choisissant *Chunk Output in Console*.

### Options de bloc de code R

Il est également possible de passer des options à chaque bloc de code R pour modifier son comportement.

Rappelez-vous qu'un bloc de code ressemble à ceci :

```
```{r}
x <- 1:5
```

Les options d'un bloc de code doivent être placées à l'intérieur des accolades {r}, avec une virgule séparant chaque option.

#### Nom du bloc

La première possibilité est de donner un *nom* au bloc. Ceci est indiqué directement après le r :

```
{r nom_bloc}
```

Il n'est pas obligatoire de nommer un bloc, mais cela peut être utile en cas d'erreur de compilation, pour identifier le bloc qui a provoqué le problème. Attention, vous ne pouvez pas avoir deux blocs avec le même nom.

#### Options

En plus d'un nom, un bloc peut se voir passer une série d'options sous la forme *option=valeur*. Voici un exemple de bloc avec un nom et des options :

```
```{r NomBloc, echo = FALSE, warning = TRUE}
x <- 1:5
```

Et un exemple de bloc sans nom avec des options :

```
```{r echo = FALSE, warning = FALSE}
x <- 1:5
```

L'une des options utiles est l'option echo. Par défaut echo vaut TRUE, et le bloc de code R est inséré dans le document généré, comme ceci :

```
x <- 1:5
print(x)
```

```
## [1] 1 2 3 4 5
```

Mais si on définit l'option echo=FALSE, alors le code R n'est plus inséré dans le document, et seul le résultat est visible :

```
## [1] 1 2 3 4 5
```

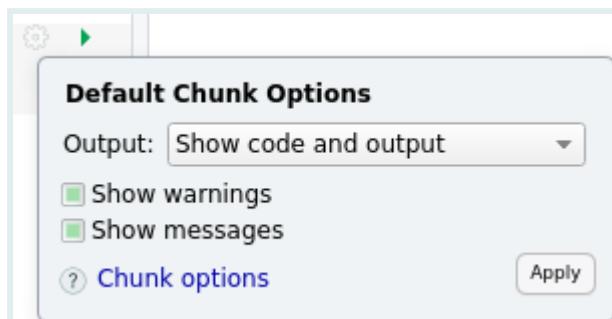
Voici une liste de certaines des options disponibles :

Option	Valeurs	Description
echo	TRUE/FALSE	Afficher (ou masquer) ce bloc de code R dans le document compilé résultant
eval	TRUE/FALSE	Exécuter (ou non) le code dans ce bloc de code dans le document compilé résultant
include	TRUE/FALSE	Combiner les options "echo" et "eval" ; soit afficher et exécuter, soit masquer et ne pas exécuter
message	TRUE/FALSE	Afficher (ou masquer) tous les messages système générés par l'exécution de ce bloc de code dans le document compilé résultant
warning	TRUE/FALSE	Afficher (ou masquer) tous les avertissements générés par l'exécution de ce bloc de code dans le document compilé résultant

Il existe de nombreuses autres options décrites notamment dans le [guide de référence R Markdown](#) (PDF en anglais).

## Modification des options

Il est possible de modifier les options manuellement en éditant l'en-tête du bloc de code, mais vous pouvez également utiliser une petite interface graphique proposée par RStudio. Pour cela, il suffit de cliquer sur l'icône d'engrenage située à droite de la ligne d'en-tête de chaque bloc :



Menu Options de bloc de code

Vous pouvez alors modifier les options les plus courantes, et cliquer sur *Apply* pour les appliquer.

## Options globales

Vous souhaiterez peut-être appliquer une option à tous les blocs d'un document. Par exemple, on peut vouloir par défaut ne pas afficher le code R de chaque bloc dans le document final.

Vous pouvez définir une option globalement en utilisant la fonction `knitr::opts_chunk$set()`. Par exemple, insérer `knitr::opts_chunk$set(echo = FALSE)` dans un bloc de code va définir l'option `echo = FALSE` par défaut pour tous les blocs suivants.

En général, nous plaçons toutes ces modifications globales dans un bloc spécial appelé `setup` qui est le premier bloc du document.

```
```{r, include=FALSE}
knitr::opts_chunk$set(echo = FALSE)
```

## Code entre les lignes

Il est également possible d'écrire des blocs de code intégrés dans le texte. Si vous allez en mode `Source` et que vous tapez [ ` ] avant et après:

“La somme d'une paire de 2 est 4”

et que vous compilez le RMD, le document résultant évaluera le code R entre les “`“. Notez que vous devez inclure le “r” au début de votre bloc de code entre les lignes pour

qu'il le reconnaisse comme du code R.

Vous pourriez également passer des variables dans votre document comme dans un programme R normal. Par exemple, sur une ligne vous pourriez exécuter :

```
taille_max <- max(women$height)
```

“La taille maximale dans le jeu de données women est 72.”

Les avantages d'un tel système sont nombreux :

- un seul document peut montrer l'ensemble de votre flux d'analyse, puisque le code, les résultats et les explications textuelles sont inclus
- le document peut être très facilement regénéré et mis à jour, par exemple si les données sources ont été modifiées.
- la variété des formats de sortie (HTML, PDF, Word, diapositives, tableaux de bord, etc.) facilite la présentation de votre travail à d'autres personnes.

## Affichage des tableaux

Les documents R Markdown peuvent afficher des tableaux de données de différentes manières. Pour commencer, vous pouvez voir comment notre RMD affiche un tableau sans mise en forme :

```
women
```

```
##     height weight
## 1      58    115
## 2      59    117
## 3      60    120
## 4      61    123
## 5      62    126
## 6      63    129
## 7      64    132
## 8      65    135
## 9      66    139
## 10     67    142
## 11     68    146
## 12     69    150
## 13     70    154
## 14     71    159
## 15     72    164
```

Il a l'air assez basique. Ensuite, pour suivre, vous devrez charger les packages suivants :

```
pacman::p_load(flextable, gt, reactable)
```

Flextable est mieux pour afficher des tableaux simples pris en charge par de nombreux formats. GT est mieux pour afficher des tableaux complexes dans des documents HTML. Reactable est plus adapté à l'affichage de très grands tableaux en HTML en donnant à votre public la possibilité de faire défiler les tableaux.

```
# Ceci est un tableau flextable  
flextable::flextable(women)
```

height	weight
58	115
59	117
60	120
61	123
62	126
63	129
64	132
65	135
66	139
67	142
68	146
69	150
70	154
71	159
72	164

```
# Ceci est un tableau GT  
gt::gt(women)
```

height	weight
58	115
59	117
60	120
61	123
62	126
63	129
64	132
65	135
66	139
67	142
68	146
69	150
70	154
71	159
72	164

```
# Ceci est un tableau reactable
reactable::reactable(women)
```

Vous pouvez voir de nombreux autres types de formats de tableau que les gens ont créés sur <https://www.rstudio.com/blog/rstudio-table-contest-2022/>

## Modèles de documents

Nous avons vu ici la production de documents “classiques”, mais R Markdown vous permet de créer de nombreuses autres choses.

Le site de documentation de l’extension propose [une galerie](#) des différentes sorties possibles. Vous pouvez créer des diaporamas, des sites web ou même des livres entiers, comme ce document.

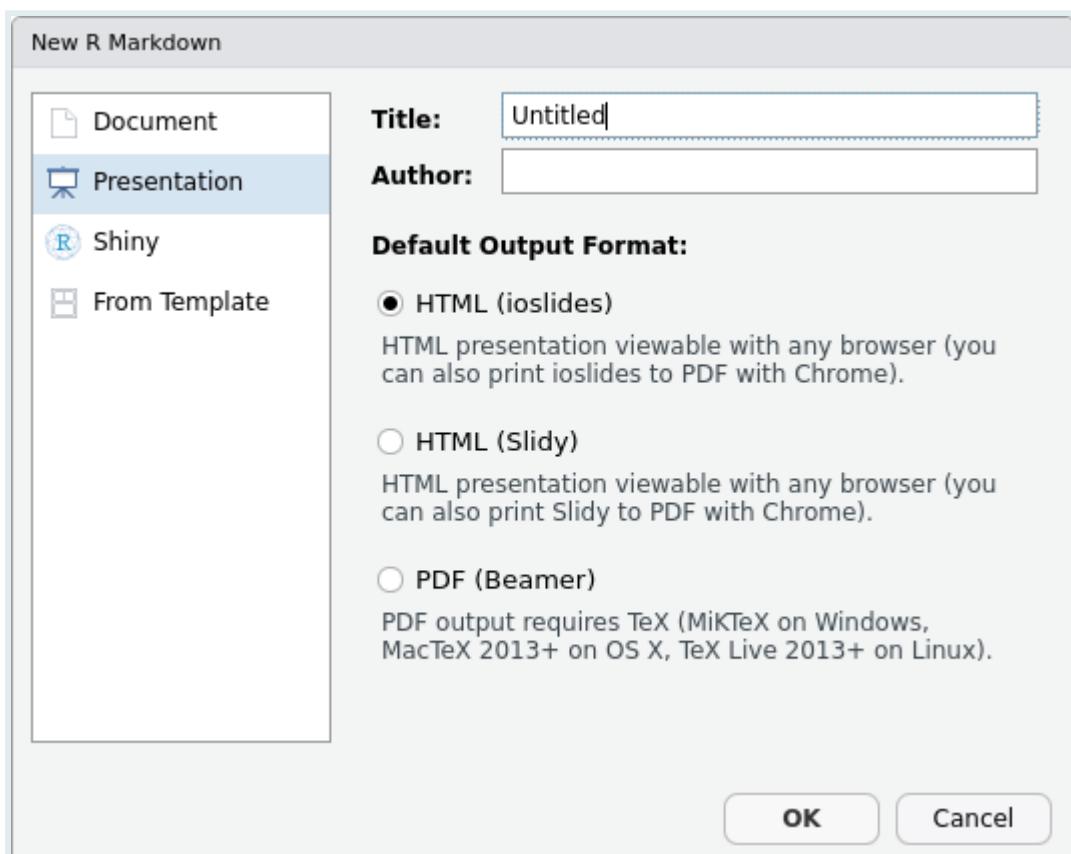
## Diapositives

Un usage intéressant est la création de diaporamas pour des présentations sous forme de diapositives. Le principe reste le même : nous mélangeons du texte au format Markdown et du code R, et R Markdown transforme le tout en présentations au format HTML ou PDF. En général, les différentes diapositives sont séparées à certains niveaux de titres.

Certains modèles de diapositives sont inclus dans R Markdown, notamment :

- `ioslides` et `Slidy` pour les présentations HTML
- `beamer` pour les présentations PDF via LaTeX

Lorsque vous créez un nouveau document dans RStudio, ces modèles sont accessibles via l'entrée *Presentation* :



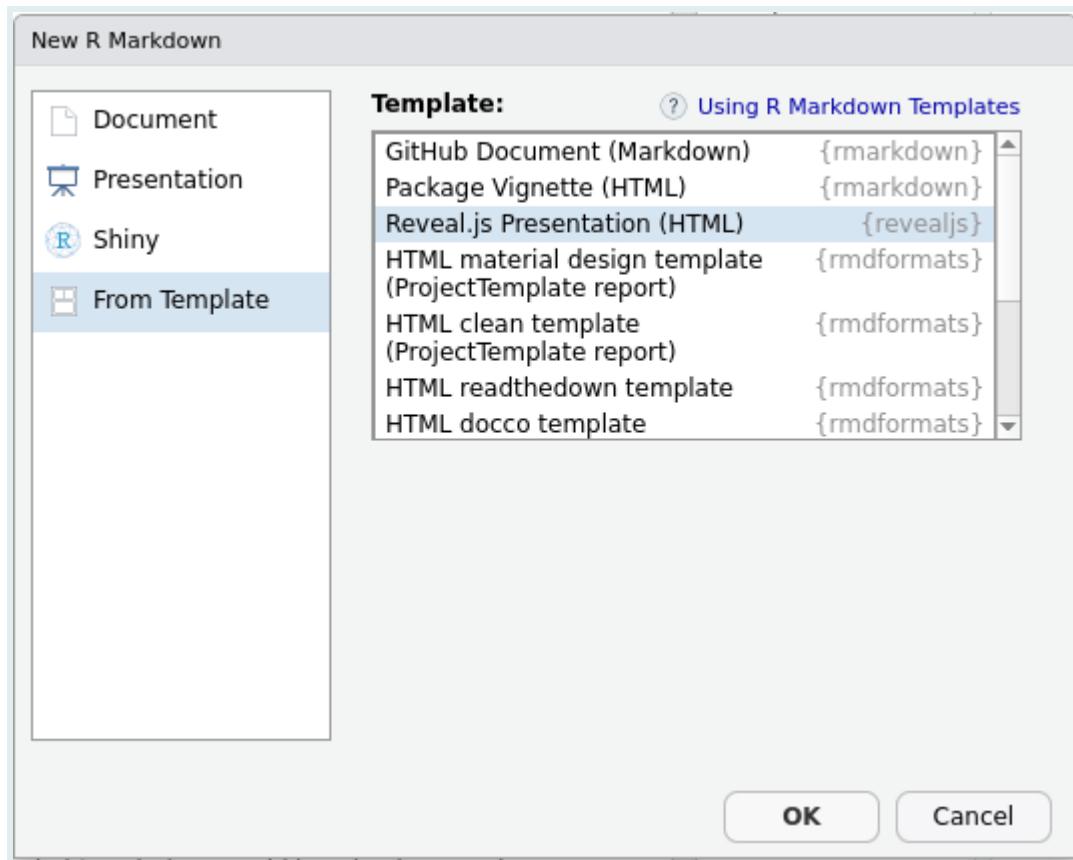
Créer une présentation R Markdown

D'autres extensions, qui doivent être installées séparément, permettent également de réaliser des diaporamas dans divers formats. Il s'agit notamment de :

- `xaringan` pour les présentations HTML basées sur `remark.js`
- `revealjs` pour les présentations HTML basées sur `reveal.js`
- `rmdshower` pour les diaporamas HTML basés sur `shower`

Une fois l'extension installée, elle offre généralement un *modèle* de départ lors de la création d'un nouveau document dans RStudio. Ceux-ci sont accessibles à partir de

l'entrée *From Template*.



Créer une présentation à partir d'un modèle

## Modèles

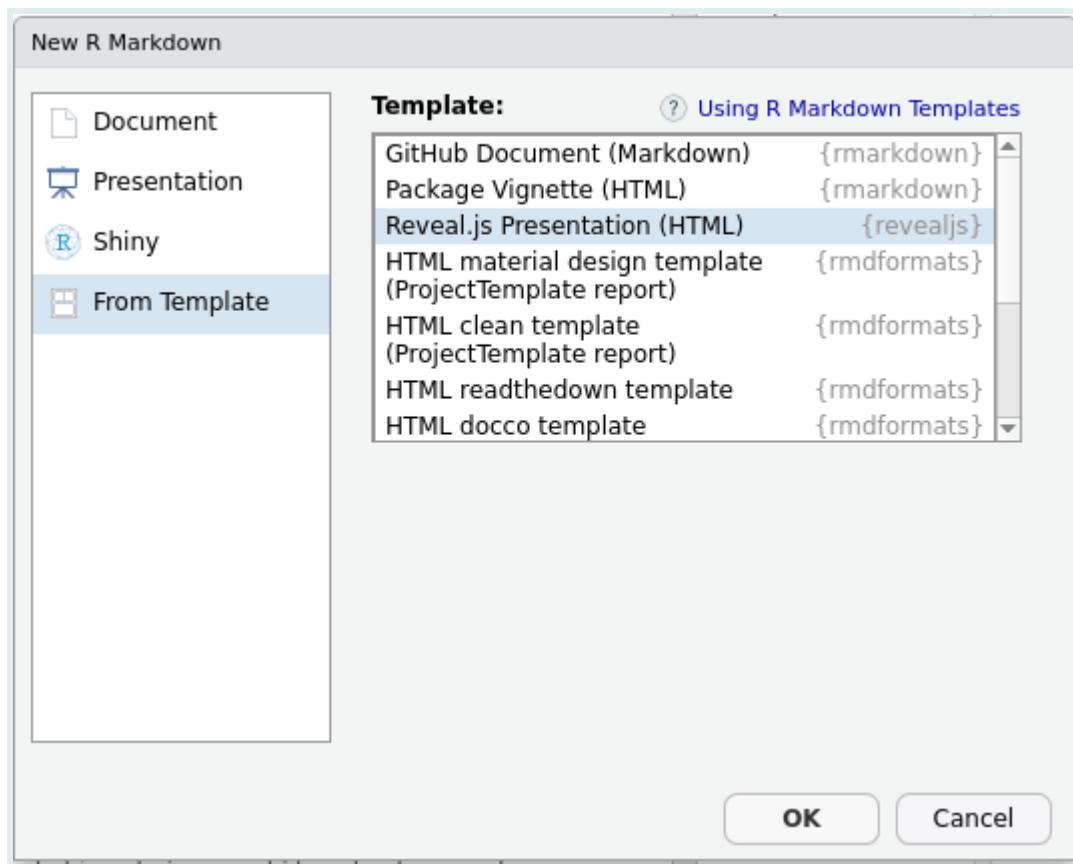
Il existe également différents *modèles* vous permettant de changer le format et la présentation des documents générés. Une liste de ces formats et leur documentation associée est disponible sur la page de documentation [formats](#).

Notez en particulier :

- le format [Distill](#), adapté aux publications scientifiques ou techniques sur le Web
- le format [Tufte Handouts](#) qui permet de produire des documents PDF ou HTML dans un format similaire à celui utilisé par Edward Tufte pour certaines de ses publications
- [rticles](#), package qui propose des modèles LaTeX pour plusieurs revues scientifiques.

Enfin, l'extension [rmdformats](#) propose plusieurs modèles HTML particulièrement adaptés pour les documents longs.

Là encore, la plupart du temps, ces modèles de documents offrent un modèle de départ lors de la création d'un nouveau document dans RStudio (entrée *From Template*) :



Créer un document à partir d'un modèle

## Ressources

Les ressources suivantes sont toutes en anglais...

Le livre *R for data science*, disponible en ligne, contient [un chapitre dédié à R Markdown](#).

[Le site officiel de l'extension](#) contient une documentation très complète, tant pour les débutants que pour les utilisateurs avancés.

Enfin, l'aide de RStudio (menu *Help* puis *Cheatsheets*) donne accès à deux documents de synthèse : une “fiche de triche” synthétique (*R Markdown Cheat Sheet*) et un “guide de référence” plus complet (*R Markdown Reference Guide*).

# Notes de leçon | Structures de données

GRAPH Network & OMS, soutenu par le Fonds Mondial

January 2024

Ce cours a été créé par le Réseau GRAPH, une organisation à but non lucratif basée à l'Institut de santé globale de l'Université de Genève, en collaboration avec l'Organisation mondiale de la Santé, dans le cadre d'une subvention du Fonds mondial pour créer des cours afin de renforcer les capacités nationales en matière d'analyse épidémiologique.



Intro .....
Objectifs d'apprentissage .....
Packages .....
Introduction aux vecteurs .....
Manipulation de vecteurs .....
Des vecteurs aux data frames .....
Tibbles .....
<code>read_csv()</code> crée des tibbles .....
Conclusion .....
Solutions .....

---

## Intro

Dans cette leçon, nous allons examiner les structures de données dans R. Nous commencerons par étudier les vecteurs, qui sont la structure de données fondamentale dans R. Puis, nous apprendrons à combiner des vecteurs dans des data frames, qui sont la structure la plus utilisée pour organiser et analyser les données. Alors, commençons !

---

## Objectifs d'apprentissage

1. Vous pouvez créer des vecteurs avec la fonction `c()`.
2. Vous pouvez combiner des vecteurs dans des data frames.
3. Vous comprenez la différence entre un tibble et un data frame.

---

## Packages

Veuillez charger les packages nécessaires pour cette leçon avec le code ci-dessous :

```
if(!require(pacman)) install.packages("pacman")
pacman::p_load(tidyverse)
```

---

## Introduction aux vecteurs

Les vecteurs sont les structures de données les plus fondamentales dans R. Ils sont une collection de valeurs qui partagent toutes la même classe (par exemple, toutes numériques ou toutes caractères).

Les vecteurs peuvent être créés en utilisant la fonction `c()` :

On écrit :

```
c(1, 2, 3)
```

```
## [1] 1 2 3
```

Nous pouvons assigner ce vecteur à un objet appelé `mon_vec` de cette manière :

```
mon_vec <- c(1, 2, 3)
```

Définissons maintenant un ensemble de quatre vecteurs différents pour trois personnes :

```
age <- c(18, 25, 46) # age est égal à dix-huit, vingt-cinq et quarante-six
sexe <- c('H', 'F', 'F') # sexe est égal à c, "H" pour homme, "F" pour femme,
# "F" pour femme
test_positif <- c(T, T, F) # test_positif est égal à c T, qui représente TRUE,
# T, F, qui représente FALSE
id <- 1:3 # et enfin, id
```

On peut également vérifier les classes de ces vecteurs :

```
class(age)
```

```
## [1] "numeric"
```

```
class(sexe)
```

```
## [1] "character"
```

```
class(test_positif)
```

```
## [1] "logical"
```

```
class(id)
```

```
## [1] "integer"
```

Chaque ligne de code ci-dessous tente de définir un vecteur avec trois éléments mais contient une erreur. Corrigez les erreurs et effectuez les commandes.

```
mon_vec_1 <- (1, 2, 3)
mon_vec_2 <- c("Obi", "Chika" "Nonso")
```

## Manipulation de vecteurs

La plupart des fonctions et opérations que vous connaissez déjà peuvent être appliquées à des vecteurs.

Par exemple, considérons le vecteur age, nous pouvons multiplier age par 2 :

```
age
```

```
## [1] 18 25 46
```

```
age * 2
```

```
## [1] 36 50 92
```

Remarquez que chaque élément du vecteur a été multiplié par 2.

Ou, ci-dessous nous prenons la racine carrée de age :

```
age
```

```
## [1] 18 25 46
```

```
sqrt(age)
```

```
## [1] 4.242641 5.000000 6.782330
```

Vous pouvez également additionner des vecteurs :

Par exemple, si nous voulons ajouter le vecteur age au vecteur id.

Regardons d'abord le vecteur age :

```
age
```

```
## [1] 18 25 46
```

et id :

```
id
```

```
## [1] 1 2 3
```

Nous pouvons simplement les additionner de cette manière :

```
age + id
```

```
## [1] 19 27 49
```

Notez que le premier élément de age est ajouté au premier élément de id et le deuxième élément de age est ajouté au deuxième élément de id, et ainsi de suite.

---

## Des vecteurs aux data frames

Maintenant que nous maîtrisons la création de vecteurs, passons à l'objet le plus utilisé dans l'analyse de données : les data frames

Un data frame est une collection de vecteurs de même longueur. Nous pouvons créer un data frame en utilisant la fonction `data.frame()`.

Rappelez-vous que nous avons précédemment créé trois objets vecteurs, (`id`, `age`, `sex` et `test_positif`) pour trois individus :

Nous pouvons maintenant utiliser la fonction `data.frame()` pour les combiner dans une seule data frame :

```
donnees.epi <- data.frame(id, age, sexe, test_positif)  
donnees.epi
```

```
##   id age sexe test_positif  
## 1  1   18    H      TRUE  
## 2  2   25    F      TRUE  
## 3  3   46    F     FALSE
```

Au lieu de créer chaque vecteur séparément, vous pouvez créer votre data frame en définissant chacun des vecteurs à l'intérieur de la fonction `data.frame()`.

```
donnees.epi_2 <- data.frame(age = c(18, 25, 46),  
                           sexe = c('M', 'F', 'F'))
```

```
donnees.epi_2
```

```
##   age sexe  
## 1  18    M  
## 2  25    F  
## 3  46    F
```

**SIDE NOTE**



La plupart du temps lorsque vous travaillez avec des données dans R, vous les importerez à partir de contextes externes. Mais il est parfois utile de créer des jeux de données *dans R* lui-même. C'est dans de tels cas que la fonction `data.frame()` sera utile.

Pour extraire les vecteurs du data frame, utilisez la syntaxe `$`, appelée le signe dollar. Exécutez les lignes de code suivantes dans votre console pour observer cela.

```
donnees.epi$age  
is.vector(donnees.epi$age) # vérifie que cette colonne est bien un vecteur  
class(donnees.epi$age) # vérifie la classe du vecteur
```

Combinez les vecteurs ci-dessous dans un data frame, avec les noms de colonnes suivants : "nom" pour le vecteur de caractères, "nb\_enfants" pour le vecteur numérique et "est\_marie" pour le vecteur logique.

```
vecteur_caracteres <- c("Bob", "Jane", "Joe")  
vecteur_numerique <- c(1, 2, 3)  
vecteur_logique <- c(T, F, F)
```

Utilisez la fonction `data.frame()` pour définir dans R un data frame qui ressemble au tableau suivant :

salle	nb_fenetres
cuisine	2
chambre	5

---

## Tibbles

La version par défaut des données tabulaires dans R s'appelle un data frame, mais il existe une autre représentation des données tabulaires fournie par le package *tidyverse*. Elle s'appelle un tibble, et c'est une version améliorée du data frame.

Vous pouvez convertir un data frame en tibble avec la fonction `as_tibble()` :

```
donnees.epi
```

```
##   id age sexe test_positif
## 1  1   18    H      TRUE
## 2  2   25    F      TRUE
## 3  3   46    F     FALSE
```

```
tibble.epi <- as_tibble(donnees.epi)
tibble.epi
```

```
## # A tibble: 3 × 4
##       id   age sexe test_positif
##   <int> <dbl> <chr> <lgl>
## 1     1     18  H     TRUE
## 2     2     25  F     TRUE
## 3     3     46  F    FALSE
```

Remarquez que le tibble affiche les dimensions des données dans la première ligne :

```
# A tibble: 3 × 4
#>       id   age sexe test_positif
#>   <int> <dbl> <chr> <lgl>
#> 1     1     18  M     VRAI
#> 2     2     25  F     VRAI
#> 3     3     46  F    FAUX
```

Et indique également les types de données en haut de chaque colonne.

Ici, "int" signifie "entier" ("integer"), "dbl" signifie "double" (qui est un type numérique), "chr" signifie "caractère" et "lgl" signifie "logique".

L'autre avantage des tibbles est qu'ils évitent d'inonder votre console lorsque vous imprimez un grand tableau.

Considérez, par exemple, ce que vous voyez dans votre console lorsque vous exécutez les lignes ci-dessous :

```
# imprime le data frame infert (un jeu de données R intégré)
infert # Impression très longue
as_tibble(infert) # impression plus gérable
```

Pour la plupart de vos besoins d'analyse de données, vous devriez préférer les tibbles aux data frames classiques.

### read\_csv() crée des tibbles

Lorsque vous importez des données avec la fonction `read_csv()` de `{readr}`, vous obtenez un tibble :

```
ebola_tib <- read_csv("https://tinyurl.com/ebola-data-sample") # Nécessite
internet pour fonctionner
class(ebola_tib)
```

```
## [1] "spec_tbl_df" "tbl_df"        "tbl"          "data.frame"
```

Mais lorsque vous importez des données avec la fonction `read.csv()` de base, vous obtenez un `data.frame` :

```
ebola_df <- read.csv("https://tinyurl.com/ebola-data-sample") # Nécessite
internet pour fonctionner
class(ebola_df)
```

```
## [1] "data.frame"
```

Essayez d'imprimer `ebola_tib` et `ebola_df` dans votre console pour observer le comportement d'impression différent des tibbles et des data frames.

C'est une des raisons pour lesquelles nous recommandons d'utiliser `read_csv()` plutôt que `read.csv()`.

---

## Conclusion

Avec votre compréhension des classes et des structures de données, vous êtes désormais bien équipé pour effectuer des tâches de manipulation de données dans R. Dans les prochaines leçons, nous explorerons les puissantes capacités de transformation de données du package `dplyr`, qui amélioreront encore vos compétences en analyse de données.

Félicitations d'être arrivé jusque-là ! Vous avez beaucoup appris et vous pouvez être fiers de vous.

## Solutions

Solution au premier bloc r-practice :

```
mon_vec_1 <- c(1,2,3) # Utilisez la fonction 'c' pour créer un vecteur  
mon_vec_2 <- c("Obi", "Chika", "Nonso") # Séparez chaque chaîne par une  
virgule
```

Solution au deuxième bloc r-practice :

```
df <- data.frame(nom = vecteur_caracteres,  
                  nb_enfants = vecteur_numerique,  
                  est_marie = vecteur_logique)
```

Solution au troisième bloc r-practice :

```
# Solution au troisième bloc r-practice  
pieces <- data.frame(piece = c("salle à manger", "cuisine", "chambre"),  
                      nb_fenetres = c(3, 2, 5))
```

## Contributeurs

L'équipe suivante a contribué à cette leçon :



### DANIEL CAMARA

Data Scientist at the GRAPH Network and fellowship as Public Health researcher at Fiocruz, Brazil

Passionate about lots of things, especially when it involves people leading lives with more equality and freedom



### EDUARDO ARAUJO

Student at Universidade Tecnologica Federal do Parana  
Passionate about reproducible science and education



### LAURE VANCAUWENBERGHE

Data analyst, the GRAPH Network  
A firm believer in science for good, striving to ally programming, health and education



### KENE DAVID NWOSU

Data analyst, the GRAPH Network  
Passionate about world improvement

## Références

Certains éléments de cette leçon ont été adaptés des sources suivantes :

- Wickham, H., & Grolemund, G. (s.d.). *R for data science*. 15 Factors | R for Data Science. Consulté le 26 octobre 2022. <https://r4ds.had.co.nz/factors.html>.

This work is licensed under the [Creative Commons Attribution Share Alike](#) license.



# Notes de cours | Sélectionner et renommer les colonnes

February 2024



Introduction	.....
Objectifs d'apprentissage	.....
L'ensemble de données COVID-19 de Yaoundé	.....
Introduction à <code>select()</code>	.....
Sélectionner des gammes de colonnes avec :	.....
Exclure des colonnes avec !	.....
Fonctions auxiliaires pour <code>select()</code>	.....
<code>starts_with()</code> et <code>ends_with()</code>	.....
<code>contains()</code>	.....
<code>everything()</code>	.....
Changer les noms de colonnes avec <code>rename()</code>	.....
Renommer dans <code>select()</code>	.....
Conclusion !	.....

---

## Introduction

Aujourd'hui, nous commencerons notre exploration du package {dplyr} ! Notre premier verbe sur la liste est `select` qui permet de conserver ou de supprimer des variables de votre dataframe. Choisir vos variables est la première étape du nettoyage de vos données.



Fig: la fonction `select()`.

C'est parti !

---

## Objectifs d'apprentissage

- Vous pouvez conserver ou supprimer des colonnes d'un dataframe en utilisant la fonction `dplyr::select()` du package {dplyr}.
- Vous pouvez sélectionner une plage ou une combinaison de colonnes à l'aide d'opérateurs comme le deux-points (:), le point d'exclamation (!) et la fonction

c().

- Vous pouvez sélectionner des colonnes en fonction des motifs de leurs noms avec des fonctions d'aide comme `starts_with()`, `ends_with()`, `contains()` et `everything()`.
- Vous pouvez utiliser `rename()` et `select()` pour changer les noms des colonnes.

---

## L'ensemble de données COVID-19 de Yaoundé

Dans cette leçon, nous analysons les résultats d'une enquête sérologique sur la COVID-19 menée à Yaoundé, au Cameroun, fin 2020. L'enquête estimait combien de personnes avaient été infectées par la COVID-19 dans la région, en testant les anticorps IgG et IgM. L'ensemble complet des données peut être obtenu sur [Zenodo](#), et l'article peut être consulté [ici](#).

Passez un peu de temps à parcourir cet ensemble de données. Chaque ligne correspond à un patient interrogé. Il y a des variables démographiques, socio-économiques et liées à la COVID. Les résultats des tests d'anticorps IgG et IgM sont dans les colonnes `igg_result` et `igm_result`.

```
<- read_csv(here:::here("data/yaounde_data.csv"))
```

```
## # A tibble: 5 × 53
##   id                  date_surveyed    age
##   <chr>                <date>        <dbl>
## 1 BRIQUETERIE_000_0001 2020-10-22     45
## 2 BRIQUETERIE_000_0002 2020-10-24     55
## 3 BRIQUETERIE_000_0003 2020-10-24     23
## 4 BRIQUETERIE_002_0001 2020-10-22     20
## 5 BRIQUETERIE_002_0002 2020-10-22     55
## #> #>   age_category age_category_3 sex
## #> #>   <chr>       <chr>        <chr>
## #> #>   1 45 - 64   Adult         Female
## #> #>   2 45 - 64   Adult         Male
## #> #>   3 15 - 29   Adult         Male
## #> #>   4 15 - 29   Adult         Female
## #> #>   5 45 - 64   Adult         Female
## #> #> #> i 47 more variables: highest_education <chr>,
## #> #> #>   occupation <chr>, weight_kg <dbl>, height_cm <dbl>, ...
```

---

**SIDE NOTE**



Notez que le jeu de donnée COVID-19 Yaoundé est en anglais !

**SIDE NOTE**

Pour cette leçon, nous utiliserons cette version en anglais. Mais dans d'autres leçons, nous utiliserons une version partialement en français.



Gauche : l'équipe d'enquête de Yaoundé. Droite : un test d'anticorps étant administré.

## Introduction à `select()`

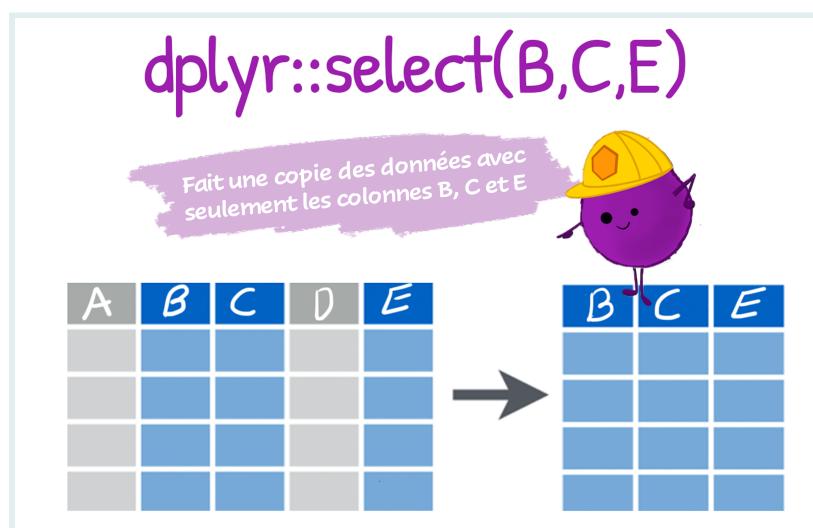


Fig : la fonction `select()`. (Dessin adapté d'Allison Horst).

`dplyr::select()` nous permet de choisir quelles colonnes (variables) conserver ou supprimer.

Nous pouvons sélectionner une colonne **par son nom** :

```
%>% select(age)
```

```
## # A tibble: 5 × 1
##       age
##   <dbl>
## 1     45
## 2     55
## 3     23
## 4     20
## 5     55
```

Ou nous pouvons sélectionner une colonne **par sa position** :

```
%>% select(3) # `age` est la 3ème colonne
```

```
## # A tibble: 5 × 1
##       age
##   <dbl>
## 1     45
## 2     55
## 3     23
## 4     20
## 5     55
```

Pour sélectionner **plusieurs variables**, nous les séparons par des virgules :

```
%>% select(age, sex, igg_result)
```

```
## # A tibble: 5 × 3
##       age   sex   igg_result
##   <dbl> <chr> <chr>
## 1     45 Female Negative
## 2     55 Male   Positive
## 3     23 Male   Negative
## 4     20 Female Positive
## 5     55 Female Positive
```



- Sélectionnez les variables poids et taille dans le dataframe `yaounde`.
- Sélectionnez les colonnes 16 et 22 dans le dataframe `yaounde`.

Pour la prochaine partie du tutoriel, créons un sous-ensemble plus petit des données, appelé `yao`.

```
ie %>% select(age,  
               sex,  
               highest_education,  
               occupation,  
               is_smoker,  
               is_pregnant,  
               igg_result,  
               igm_result)
```

```
## # A tibble: 5 × 8  
##   age   sex   highest_education  
##   <dbl> <chr> <chr>  
## 1    45 Female Secondary  
## 2    55 Male   University  
## 3    23 Male   University  
## 4    20 Female Secondary  
## 5    55 Female Primary  
## # ... with 2 more variables: occupation <chr>,  
## #   is_smoker <chr>, is_pregnant <chr>  
## # # i 2 more variables: igg_result <chr>, igm_result <chr>
```

Sélectionner des gammes de colonnes avec :

L'opérateur : permet de sélectionner une **plage de variables consécutives** :

```
select(age:occupation) # Sélectionnez toutes les colonnes de `age` à  
`occupation`
```

```
## # A tibble: 5 × 4  
##   age   sex   highest_education  
##   <dbl> <chr> <chr>  
## 1    45 Female Secondary  
## 2    55 Male   University  
## 3    23 Male   University  
## 4    20 Female Secondary  
## 5    55 Female Primary  
## # ... with 1 more variable:  
## #   occupation <chr>  
## # # i 1 Informal worker  
## # # 2 Salaried worker  
## # # 3 Student
```

```
## 4 Student  
## 5 Trader--Farmer
```

Nous pouvons également spécifier une plage avec des numéros de colonne :

```
select(1:4) # Sélectionnez les colonnes 1 à 4
```

```
## # A tibble: 5 × 4  
##   age sex   highest_education  
##   <dbl> <chr> <chr>  
## 1     45 Female Secondary  
## 2     55 Male   University  
## 3     23 Male   University  
## 4     20 Female Secondary  
## 5     55 Female Primary  
## # A tibble: 5 × 4  
##   occupation  
##   <chr>  
## 1 Informal worker  
## 2 Salaried worker  
## 3 Student  
## 4 Student  
## 5 Trader--Farmer
```



- Avec le dataframe `yaounde`, sélectionnez les colonnes entre `symptoms` et `sequelae`, inclusivement. ("Inclusivement" signifie que vous devriez également inclure `symptoms` et `sequelae` dans la sélection.)

Exclure des colonnes avec !

Le point d'exclamation nie une sélection :

```
select(!age) # Sélectionnez toutes les colonnes sauf `age`
```

```
## # A tibble: 5 × 7  
##   sex   highest_education occupation  
##   <chr> <chr> <chr>  
## 1 Female Secondary Informal worker  
## 2 Male   University Salaried worker  
## 3 Male   University Student  
## 4 Female Secondary Student  
## 5 Female Primary Trader--Farmer  
## # A tibble: 5 × 7  
##   is_smoker is_pregnant igg_result igm_result  
##   <chr>      <chr>      <chr>      <chr>  
## 1 Non-smoker No        Negative  Negative  
## 2 Ex-smoker  <NA>       Positive  Negative
```

```

## 3 Smoker      <NA>          Negative  Negative
## 4 Non-smoker No           Positive  Negative
## 5 Non-smoker No           Positive  Negative

```

Pour supprimer une plage de colonnes consécutives, nous utilisons, par exemple, `!age:occupation`:

```
select(!age:occupation) # Supprimez les colonnes de `age` à `occupation`
```

```

## # A tibble: 5 × 4
##   is_smoker is_pregnant igg_result igm_result
##   <chr>     <chr>       <chr>       <chr>
## 1 Non-smoker No          Negative  Negative
## 2 Ex-smoker  <NA>        Positive  Negative
## 3 Smoker    <NA>        Negative  Negative
## 4 Non-smoker No          Positive  Negative
## 5 Non-smoker No          Positive  Negative

```

Pour supprimer plusieurs colonnes non consécutives, placez-les à l'intérieur de `!c()`:

```
select(!c(age, sex, igg_result)) # Supprimez les colonnes de `age`, `sex`, et
`igg_result`
```

```

## # A tibble: 5 × 5
##   highest_education occupation      is_smoker
##   <chr>              <chr>       <chr>
## 1 Secondary          Informal worker Non-smoker
## 2 University         Salaried worker Ex-smoker
## 3 University         Student      Smoker
## 4 Secondary          Student      Non-smoker
## 5 Primary            Trader--Farmer Non-smoker
##   is_pregnant igm_result
##   <chr>       <chr>
## 1 No             Negative
## 2 <NA>          Negative
## 3 <NA>          Negative
## 4 No             Negative
## 5 No             Negative

```



- Du dataframe `yaounde`, retirez toutes les colonnes entre `highest_education` et `consultation`, inclusivement.

---

## Fonctions auxiliaires pour `select()`

`dplyr` possède un certain nombre de fonctions auxiliaires pour faciliter la sélection en utilisant des motifs des noms de colonnes. Jetons un œil à certaines d'entre elles.

`starts_with()` et `ends_with()`

Ces deux fonctions auxiliaires fonctionnent exactement comme leurs noms l'indiquent !

```
select(starts_with("is_")) # Colonnes qui commencent par "is"
```

```
## # A tibble: 5 × 2
##   is_smoker  is_pregnant
##   <chr>       <chr>
## 1 Non-smoker No
## 2 Ex-smoker  <NA>
## 3 Smoker     <NA>
## 4 Non-smoker No
## 5 Non-smoker No
```

```
select(ends_with("_result")) # Colonnes qui finissent par "result"
```

```
## # A tibble: 5 × 2
##   igg_result igm_result
##   <chr>        <chr>
## 1 Negative    Negative
## 2 Positive    Negative
## 3 Negative    Negative
## 4 Positive    Negative
## 5 Positive    Negative
```

`contains()`

`contains()` aide à sélectionner les colonnes qui contiennent une certaine chaîne :

```
%>% select(contains("drug")) # Colonnes contenant la chaîne "drug"
```

```
## # A tibble: 5 × 12
##   drugsource      is_drug_parac is_drug_antibio
##   <chr>           <dbl>          <dbl>
## 1 Self or famili...     1              0
## 2 <NA>             NA             NA
## 3 <NA>             NA             NA
## 4 Self or famili...     0              1
```

```

## 5 <NA> NA NA
##   is_drug_hydrocortisone is_drug_other_anti_i...1
##   <dbl> <dbl>
## 1 0 0
## 2 NA NA
## 3 NA NA
## 4 0 0
## 5 NA NA
## # i abbreviated name: 1is_drug_other_anti_inflam
## # i 7 more variables: is_drug_antiviral <dbl>, ...

```

`everything()`

Une autre fonction auxiliaire, `everything()`, correspond à toutes les variables qui n'ont pas encore été sélectionnées.

```

cd, `is_pregnant`, puis toutes les autres colonnes.
select(is_pregnant, everything())

```

```

## # A tibble: 5 × 8
##   is_pregnant age sex   highest_education
##   <chr>        <dbl> <chr> <chr>
## 1 No           45 Female Secondary
## 2 <NA>         55 Male   University
## 3 <NA>         23 Male   University
## 4 No           20 Female Secondary
## 5 No           55 Female Primary
##   occupation    is_smoker  igg_result
##   <chr>          <chr>      <chr>
## 1 Informal worker Non-smoker Negative
## 2 Salaried worker Ex-smoker  Positive
## 3 Student       Smoker     Negative
## 4 Student       Non-smoker Positive
## 5 Trader--Farmer Non-smoker Positive
## # i 1 more variable: igm_result <chr>

```

C'est souvent utile pour établir l'ordre des colonnes.

Disons que nous voulions amener la colonne `is_pregnant` au début du dataframe `yao`, nous pourrions écrire manuellement tous les noms des colonnes :

```

select(is_pregnant,
       age,
       sex,
       highest_education,
       occupation,
       is_smoker,
       igg_result,
       igm_result)

```

```

## # A tibble: 5 × 8
##   is_pregnant    age sex   highest_education
##   <chr>        <dbl> <chr> <chr>
## 1 No            45 Female Secondary
## 2 <NA>          55 Male   University
## 3 <NA>          23 Male   University
## 4 No            20 Female Secondary
## 5 No            55 Female Primary
## # ℹ 2 more variables: occupation <chr>
## #   is_smoker     <chr> igg_result <chr>
## #   <chr>           <chr>      <chr>
## # 1 Informal worker Non-smoker Negative
## # 2 Salaried worker Ex-smoker Positive
## # 3 Student       Smoker      Negative
## # 4 Student       Non-smoker Positive
## # 5 Trader--Farmer Non-smoker Positive
## # ℹ 1 more variable: igm_result <chr>

```

Mais ce serait pénible pour des dataframes plus grands, comme notre dataframe original `yaounde`. Dans un tel cas, nous pouvons utiliser `everything()` :

```

` `is_pregnant` à l'avant du dataframe
%>% select(is_pregnant, everything())

```

```

## # A tibble: 5 × 53
##   is_pregnant id                  date_surveyed
##   <chr>        <chr>                <date>
## 1 No          BRIQUETERIE_000_00... 2020-10-22
## 2 <NA>        BRIQUETERIE_000_00... 2020-10-24
## 3 <NA>        BRIQUETERIE_000_00... 2020-10-24
## 4 No          BRIQUETERIE_002_00... 2020-10-22
## 5 No          BRIQUETERIE_002_00... 2020-10-22
## # ℹ 2 more variables: age <dbl>
## #   age_category <chr>   age_category_3 <chr>
## #   <dbl>         <chr>   <chr>           <chr>
## # 1 45           45 - 64   Adult   Female
## # 2 55           45 - 64   Adult   Male
## # 3 23           15 - 29   Adult   Male
## # 4 20           15 - 29   Adult   Female
## # 5 55           45 - 64   Adult   Female
## # ℹ 46 more variables: highest_education <chr>,
## #   occupation <chr>, weight_kg <dbl>, height_cm <dbl>, ...

```

Cette fonction auxiliaire peut être combinée avec beaucoup d'autres.

```

` les colonnes qui se terminent par "result" à l'avant du dataframe
%>% select(ends_with("result"), everything())

```

```

## # A tibble: 5 × 53
##   igg_result igm_result id
##   <chr>      <chr>     <chr>

```

```

## 1 Negative    Negative    BRIQUETERIE_000_0001
## 2 Positive    Negative    BRIQUETERIE_000_0002
## 3 Negative    Negative    BRIQUETERIE_000_0003
## 4 Positive    Negative    BRIQUETERIE_002_0001
## 5 Positive    Negative    BRIQUETERIE_002_0002
##   date_surveyed    age    age_category
##   <date>        <dbl> <chr>
## 1 2020-10-22      45 45 - 64
## 2 2020-10-24      55 45 - 64
## 3 2020-10-24      23 15 - 29
## 4 2020-10-22      20 15 - 29
## 5 2020-10-22      55 45 - 64
## # i 47 more variables: age_category_3 <chr>, sex <chr>,
## #   highest_education <chr>, occupation <chr>, ...

```



- Sélectionnez toutes les colonnes dans le dataframe `yaounde` qui commencent par “`is_`”.
- Déplacez les colonnes qui commencent par “`is_`” au début du dataframe `yaounde`.

## Changer les noms de colonnes avec `rename()`

`dplyr::rename()` est utilisée pour changer les noms des colonnes :

```

nez `age` et `sex` en `patient_age` et `patient_sex`
%>%
`(.patient_age = age,
 .patient_sex = sex)`

```

```

## # A tibble: 5 × 53
##   id                  date_surveyed patient_age
##   <chr>                <date>          <dbl>
## 1 BRIQUETERIE_000_00... 2020-10-22      45
## 2 BRIQUETERIE_000_00... 2020-10-24      55
## 3 BRIQUETERIE_000_00... 2020-10-24      23
## 4 BRIQUETERIE_002_00... 2020-10-22      20
## 5 BRIQUETERIE_002_00... 2020-10-22      55
##   age_category age_category_3 patient_sex
##   <chr>         <chr>           <chr>
## 1 45 - 64     Adult            Female
## 2 45 - 64     Adult            Male
## 3 15 - 29     Adult            Male
## 4 15 - 29     Adult            Female

```

```
## 5 45 - 64      Adult      Female
## # i 47 more variables: highest_education <chr>,
## #   occupation <chr>, weight_kg <dbl>, height_cm <dbl>, ...
```

### WATCH OUT



Le fait que le nouveau nom vienne en premier dans la fonction (`rename(NOUVEAUNOM = VIEUXNOM)`) est parfois déroutant. Vous devriez vous y habituer avec le temps.

## Renommer dans `select()`

Vous pouvez également renommer des colonnes tout en les sélectionnant :

```
cionnez `age` et `sex`, et renommez-les en `patient_age` et `patient_sex`%
%>%
:(patient_age = age,
 patient_sex = sex)
```

```
## # A tibble: 5 × 2
##   patient_age patient_sex
##       <dbl> <chr>
## 1         45 Female
## 2         55 Male
## 3         23 Male
## 4         20 Female
## 5         55 Female
```

## Conclusion !

J'espère que cette première leçon vous a permis de voir à quel point les verbes `{dplyr}` sont intuitifs et utiles ! Ceci est la première d'une série de verbes de base pour la manipulation de données : rendez-vous à la prochaine leçon pour en savoir plus.



Fig: Basic Data Wrangling Dplyr Verbs.

## Contributeurs

Les membres de l'équipe suivants ont contribué à cette leçon :



### LAURE VANCAUWENBERGHE

Data analyst, the GRAPH Network

A firm believer in science for good, striving to ally programming, health and education



### ANDREE VALLE CAMPOS

R Developer and Instructor, the GRAPH Network

Motivated by reproducible science and education



### KENE DAVID NWOSU

Data analyst, the GRAPH Network

Passionate about world improvement



### SABINA RODRIGUEZ VELÁSQUEZ

Project Manager and Scientific Collaborator, The GRAPH Network

Infectiously enthusiastic about microbes and Global Health

## Références

Certains matériaux de cette leçon ont été adaptés des sources suivantes :

- Horst, A. (2021). *Dplyr-learnr*. <https://github.com/allisonhorst/dplyr-learnr> (Œuvre originale publiée en 2020)
- Sélectionner des colonnes en utilisant leurs noms et types---*Select*. (n.d.). Récupéré le 31 décembre 2021, de <https://dplyr.tidyverse.org/reference/select.html>

L'artwork a été adapté de :

- Horst, A. (2021). *Illustrations R & stats par Allison Horst*. <https://github.com/allisonhorst/stats-illustrations> (Œuvre originale publiée en 2018)

# Notes de leçon | Filtrer les lignes

February 2024



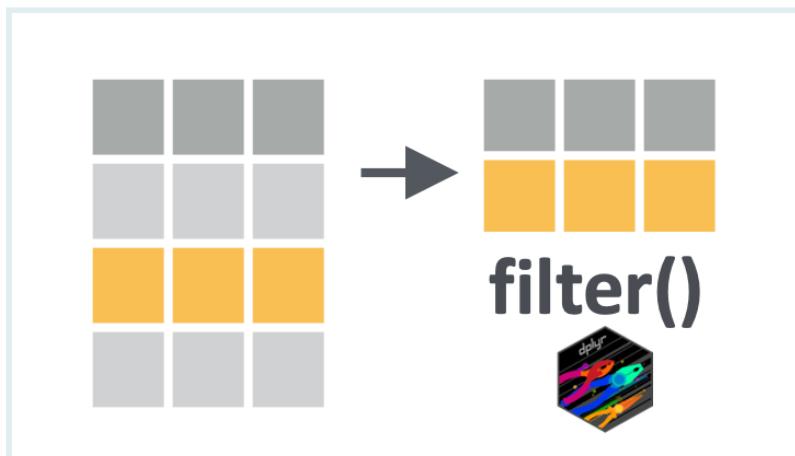
Introduction	.....
Objectifs d'apprentissage	.....
L'ensemble de données COVID-19 de Yaoundé	.....
Introduction à <code>filter()</code>	.....
Opérateurs relationnels	.....
Combiner des conditions avec & et	.....
Négation des conditions avec !	.....
Valeurs NA	.....
En Résumé !	.....

---

## Introduction

Poursuivons avec le package `{dplyr}`, en découvrant le verbe `filter`. La dernière fois, nous avons vu comment `select` sélectionne des variables (colonnes) et aujourd'hui nous verrons comment conserver ou supprimer des entrées de données, des lignes, en utilisant `filter`. Supprimer des entrées de données anormales ou conserver des sous-ensembles de vos points de données est un autre aspect essentiel de la manipulation des données.

Allons-y !



---

## Objectifs d'apprentissage

1. Vous pouvez utiliser `dplyr::filter()` pour conserver ou supprimer des lignes d'un dataframé.
2. Vous pouvez filtrer les lignes en spécifiant des conditions sur des nombres ou des chaînes en utilisant des opérateurs relationnels comme supérieur à (`>`), inférieur à (`<`), égal à (`==`), et différent de (`!=`).
3. Vous pouvez filtrer les lignes en combinant des conditions avec des opérateurs logiques comme le esperluette (`&`) et la barre verticale (`|`).
4. Vous pouvez filtrer les lignes en négatif des conditions avec l'opérateur logique point d'exclamation (`!`).

5. Vous pouvez filtrer les lignes avec des valeurs manquantes en utilisant la fonction `is.na()`.

## L'ensemble de données COVID-19 de Yaoundé

Dans cette leçon, nous utiliserons à nouveau les données de l'enquête sérologique COVID-19 réalisée à Yaoundé, au Cameroun.

```
<- read_csv(here:::here('data/yaounde_data.csv'))  
is-ensemble plus petit de variables  
yaounde %>%  
  (age, sex, weight_kg, highest_education, neighborhood,  
   occupation, is_smoker, is_pregnant,  
   igg_result, igm_result)
```

```
## # A tibble: 5 × 10  
##       age   sex   weight_kg highest_education  
##     <dbl> <chr>      <dbl> <chr>  
## 1     45 Female        95 Secondary  
## 2     55 Male          96 University  
## 3     23 Male          74 University  
## 4     20 Female        70 Secondary  
## 5     55 Female        67 Primary  
## # ... with 3 more variables: neighborhood <chr>  
## #   occupation <chr> is_smoker <chr>  
## #   1 Briqueterie Informal worker Non-smoker  
## #   2 Briqueterie Salaried worker Ex-smoker  
## #   3 Briqueterie Student Smoker  
## #   4 Briqueterie Student Non-smoker  
## #   5 Briqueterie Trader--Farmer Non-smoker  
## # i 3 more variables: is_pregnant <chr>, igg_result <chr>,  
## #   igm_result <chr>
```

**SIDE NOTE**



Notez que le jeu de donnée COVID-19 Yaoundé est en anglais !

Pour cette leçon, nous utiliserons cette version en anglais. Mais dans d'autres leçons, nous utiliserons une version partialement en français.

## Introduction à `filter()`

Nous utilisons `filter()` pour conserver les lignes qui satisfont à un ensemble de conditions. Prenons un exemple simple. Si nous voulons conserver uniquement les enregistrements masculins, nous exécutons :

```
filter(sex == "Male")
```

```
## # A tibble: 5 × 10
##       age sex    weight_kg highest_education
##   <dbl> <chr>     <dbl> <chr>
## 1     55 Male      96 University
## 2     23 Male      74 University
## 3     28 Male      62 Doctorate
## 4     30 Male      73 Secondary
## 5     42 Male      71 Secondary
## #   neighborhood occupation      is_smoker
## #   <chr>          <chr>          <chr>
## 1 Briqueterie Salaried worker Ex-smoker
## 2 Briqueterie Student        Smoker
## 3 Briqueterie Student        Non-smoker
## 4 Briqueterie Trader         Non-smoker
## 5 Briqueterie Trader         Ex-smoker
## # ℹ 3 more variables: is_pregnant <chr>, igg_result <chr>,
## #   igm_result <chr>
```

Notez l'utilisation du double signe égal `==` plutôt que le signe égal simple `=`. Le signe `==` teste l'égalité, comme le montre l'exemple ci-dessous :

```
l'objet `sex_vector` avec trois éléments
:or <- c("Male", "Female", "Female")
: quels éléments sont égaux à "Male"
:or == "Male"
```

```
## [1] TRUE FALSE FALSE
```

Donc le code `yao %>% filter(sex == "Male")` conservera toutes les lignes où le test d'égalité `sex == "Male"` évalue à `TRUE`.

Il est souvent utile de chaîner `filter()` avec `nrow()` pour obtenir le nombre de lignes remplissant une condition.

```
en de répondants étaient des hommes?
:(sex == "Male") %>%
```

```
## [1] 422
```

**KEY POINT**

Le double signe égal, `==`, teste l'égalité, tandis que le signe égal simple, `=`, est utilisé pour spécifier des valeurs aux arguments à l'intérieur des fonctions.

**PRACTICE****(in RMD)**

Filtrez le dataframe `yao` pour sélectionner les répondants qui étaient enceintes lors de l'enquête.

Combien de répondants étaient des femmes? (Utilisez `filter()` et `nrow()`)

## Opérateurs relationnels

L'opérateur `==` introduit ci-dessus est un exemple d'un opérateur "relationnel", car il teste la relation entre deux valeurs. Voici une liste de certains de ces opérateurs :

**Opérateur est VRAI si**

`A < B` A est **inférieur à B**

`A <= B` A est **inférieur ou égal à B**

`A > B` A est **supérieur à B**

`A >= B` A est **supérieur ou égal à B**

`A == B` A est **égal à B**

`A != B` A est **différent de B**

`A %in% B` A est **un élément de B**

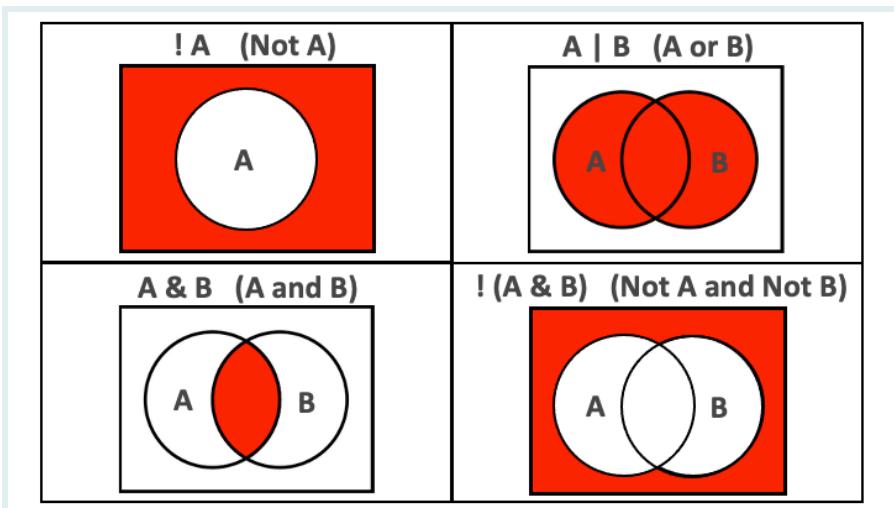


Fig: Opérateurs AND et OR visualisés.

Voyons comment les utiliser avec `filter()` :

```
filter(_____) ## gardez les lignes où `sex` n'est pas "Male"
filter(_____) ## gardez les répondants de moins de 6 ans
filter(_____) ## gardez les répondants âgés d'au moins 70 ans

# les répondants dont le niveau d'éducation le plus élevé est "Primary" ou
# "Secondary"
filter(_____)
```

### PRACTICE



(in RMD)

De `yao`, conservez uniquement les répondants qui étaient des enfants (moins de 18 ans).

Avec `%in%`, conservez uniquement les répondants qui vivent dans les quartiers “Tsinga” ou “Messa”.

## Combiner des conditions avec & et |

Nous pouvons passer plusieurs conditions à une seule instruction `filter()` séparées par des virgules:

```
#ver les répondants qui sont enceintes et qui sont d'anciens fumeurs
filter(is_pregnant == "Yes", is_smoker == "Ex-smoker") ## seulement une ligne
```

```
## # A tibble: 1 × 10
##       age   sex   weight_kg highest_education
##     <dbl> <chr>      <dbl> <chr>
## 1     25 Female        90 Secondary
```

```

##   neighborhood occupation is_smoker is_pregnant
##   <chr>          <chr>          <chr>          <chr>
## 1 Carriere       Home-maker   Ex-smoker   Yes
## # i 2 more variables: igg_result <chr>, igm_result <chr>

```

Lorsque plusieurs conditions sont séparées par une virgule, elles sont implicitement combinées avec un **et** (`&`).

Il est préférable de remplacer la virgule par `&` pour rendre cela plus explicite.

```

résultat qu'auparavant, mais `&` est plus explicite
filter(is_pregnant == "Yes" & is_smoker == "Ex-smoker")

```

```

## # A tibble: 1 × 10
##       age sex    weight_kg highest_education
##     <dbl> <chr>      <dbl> <chr>
## 1     25 Female      90 Secondary
##   neighborhood occupation is_smoker is_pregnant
##   <chr>          <chr>          <chr>          <chr>
## 1 Carriere       Home-maker   Ex-smoker   Yes
## # i 2 more variables: igg_result <chr>, igm_result <chr>

```

Ne confondez pas :

#### SIDE NOTE



- la “,” pour lister plusieurs conditions dans `filter(A, B)`  
c'est-à-dire filtrer sur la condition A et (`&`) la condition B
- la “,” dans les listes `c(A, B)` qui énumère différents composants de la liste (et n'a rien à voir avec l'opérateur `&`)

Si nous voulons combiner des conditions avec un **ou**, nous utilisons le symbole de la barre verticale, `|`.

```

dants qui sont enceintes OU qui sont d'anciens fumeurs
filter(is_pregnant == "Yes" | is_smoker == "Ex-smoker")

```

```

## # A tibble: 5 × 10
##       age sex    weight_kg highest_education
##     <dbl> <chr>      <dbl> <chr>
## 1     55 Male       96 University
## 2     42 Male       71 Secondary
## 3     38 Male       71 University
## 4     69 Male      108 University
## 5     65 Male       93 Secondary

```

```

##   neighborhood occupation      is_smoker
##   <chr>        <chr>          <chr>
## 1 Briqueterie Salaried worker Ex-smoker
## 2 Briqueterie Trader          Ex-smoker
## 3 Briqueterie Informal worker Ex-smoker
## 4 Briqueterie Retired         Ex-smoker
## 5 Briqueterie Retired         Ex-smoker
## # i 3 more variables: is_pregnant <chr>, igg_result <chr>,
## #   igm_result <chr>

```



Filtrez `yao` pour ne conserver que les hommes qui ont été testés positifs pour les IgG.

Filtrez `yao` pour conserver à la fois les enfants (moins de 18 ans) et toute personne dont le niveau d'éducation le plus élevé est l'école primaire.

## Négation des conditions avec !

Pour nier les conditions, nous les enveloppons dans `!()`.

Ci-dessous, nous supprimons les répondants qui sont des enfants (moins de 18 ans) ou qui pèsent moins de 30 kg :

```

# supprimer les répondants < 18 ans OU < 30 kg
filter(!(age < 18 | weight_kg < 30))

```

```

## # A tibble: 5 × 10
##       age sex    weight_kg highest_education
##       <dbl> <chr>     <dbl> <chr>
## 1     45 Female      95 Secondary
## 2     55 Male        96 University
## 3     23 Male        74 University
## 4     20 Female      70 Secondary
## 5     55 Female      67 Primary
##   neighborhood occupation      is_smoker
##   <chr>        <chr>          <chr>
## 1 Briqueterie Informal worker Non-smoker
## 2 Briqueterie Salaried worker Ex-smoker
## 3 Briqueterie Student        Smoker
## 4 Briqueterie Student        Non-smoker
## 5 Briqueterie Farmer--Trader Non-smoker
## # i 3 more variables: is_pregnant <chr>, igg_result <chr>,
## #   igm_result <chr>

```

L'opérateur ! est également utilisé pour nier %in% car R n'a pas d'opérateur pour NOT in.

```
mer les répondants dont le niveau d'éducation le plus élevé n'est PAS  
"Primaire" ou "Secondaire"  
filter(!(highest_education %in% c("Primary", "Secondary")))
```

```
## # A tibble: 5 × 10  
##   age sex   weight_kg highest_education  
##   <dbl> <chr>    <dbl> <chr>  
## 1    55 Male      96 University  
## 2    23 Male      74 University  
## 3    28 Male      62 Doctorate  
## 4    38 Male      71 University  
## 5    54 Male      71 University  
## # ... with 3 more variables: neighborhood <chr>, occupation <chr>  
## # ... with 3 more variables: is_smoker <chr>,  
## #   is_pregnant <chr>, igg_result <chr>,  
## #   igm_result <chr>
```

Il est plus facile de lire les instructions `filter()` comme des instructions **conserver**, pour éviter toute confusion sur le fait de savoir si nous filtrons **dans** ou filtrons **hors**!

Ainsi, le code ci-dessous se lirait : “**conserver** les répondants qui ont moins de 18 ans ou qui pèsent moins de 30 kg”.

#### KEY POINT



```
filter(age < 18 | weight_kg < 30)
```

Et lorsque nous enveloppons des conditions dans `!()`, nous pouvons alors lire les instructions `filter()` comme des instructions **supprimer**.

Ainsi, le code ci-dessous se lirait : “**supprimer** les répondants qui ont moins de 18 ans ou qui pèsent moins de 30 kg”.

```
filter(! (age < 18 | weight_kg < 30))
```

#### PRACTICE





Dans `yao`, supprimez les répondants qui vivent dans les quartiers Tsinga ou Messa.

## Valeurs NA

Les opérateurs relationnels introduits jusqu'à présent ne fonctionnent pas avec `NA`.

Créons un sous-ensemble de données pour illustrer cela.

```
<- yao %>%
  (sex, is_pregnant) %>%
(1,11,50,2) # ordre de ligne personnalisé
```

```
## # A tibble: 4 × 2
##   sex    is_pregnant
##   <chr>  <chr>
## 1 Female No
## 2 Female No response
## 3 Female Yes
## 4 Male   <NA>
```

Dans `yao_mini`, le dernier répondant a une valeur `NA` pour la colonne `is_pregnant` car il est un homme.

Essayer de sélectionner cette ligne en utilisant `== NA` ne fonctionnera pas.

```
%>% filter(is_pregnant == NA) # ne fonctionne pas
```

```
## # A tibble: 0 × 2
## # i 2 variables: sex <chr>, is_pregnant <chr>
```

```
%>% filter(is_pregnant == "NA") # ne fonctionne pas
```

```
## # A tibble: 0 × 2
## # i 2 variables: sex <chr>, is_pregnant <chr>
```

C'est parce que `NA` est une valeur inexistante. Ainsi, R ne peut pas évaluer si elle est “équivalente à” ou “différente de” quoi que ce soit.

La fonction spéciale `is.na()` est donc nécessaire :

```
les lignes où `is_pregnant` est NA  
%>% filter(is.na(is_pregnant))
```

```
## # A tibble: 1 × 2  
##   sex    is_pregnant  
##   <chr>  <chr>  
## 1 Male   <NA>
```

Cette fonction peut être niée avec ! :

```
!me les lignes où `is_pregnant` est NA  
%>% filter(!is.na(is_pregnant))
```

```
## # A tibble: 3 × 2  
##   sex    is_pregnant  
##   <chr>  <chr>  
## 1 Female No  
## 2 Female No response  
## 3 Female Yes
```

Pour les tibbles, RStudio mettra en évidence les valeurs `NA` en rouge vif pour les distinguer des autres valeurs :

**SIDE NOTE**



```
# A tibble: 5 × 3  
  age   sex    is_pregnant  
  <dbl> <chr>  <chr>  
1 32   Male   NA  
2 23   Female Yes  
3 35   Male   NA  
4 31   Female No  
5 17   Female No response
```

Une erreur courante avec `NA`

**SIDE NOTE**



Les valeurs `NA` peuvent être identifiées, mais toute autre codification telle que "`NA`" ou "`NaN`", qui sont codées comme des chaînes, sera imperceptible pour les fonctions (ce sont des chaînes, comme toutes les autres).



Dans l'ensemble de données `yao`, conservez tous les répondants qui avaient des dossiers manquants concernant le signalement de leur statut de fumeur.



Pour certains répondants, la fréquence respiratoire, en respirations par minute, a été enregistrée dans la colonne `respiration_frequency`.

Dans `yaounde`, éliminez ceux ayant une fréquence respiratoire inférieure à 20. Pensez aux NAs lors de cette opération ! Vous devriez également éviter de supprimer les valeurs NA.

## En Résumé !

Maintenant, vous connaissez les deux verbes essentiels pour sélectionner (`select()`) les colonnes et filtrer (`filter()`) les lignes. De cette manière, vous conservez les variables qui vous intéressent en sélectionnant vos colonnes et vous conservez les entrées de données que vous jugez pertinentes en filtrant vos lignes.

Mais qu'en est-il de la modification, de la transformation de vos données ? Nous en apprendrons davantage à ce sujet dans la prochaine leçon. À bientôt !

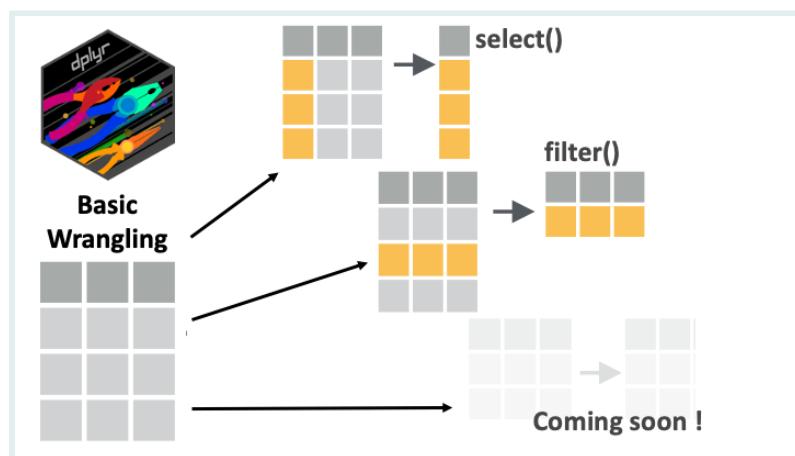


Fig: Manipulation de données de base : `select()` et `filter()`.

---

## Contributeurs

Les membres de l'équipe suivants ont contribué à cette leçon :



### LAURE VANCAUWENBERGHE

Data analyst, the GRAPH Network

A firm believer in science for good, striving to ally programming, health and education

---



### ANDREE VALLE CAMPOS

R Developer and Instructor, the GRAPH Network

Motivated by reproducible science and education

---



### KENE DAVID NWOSU

Data analyst, the GRAPH Network

Passionate about world improvement

---



### SABINA RODRIGUEZ VELÁSQUEZ

Project Manager and Scientific Collaborator, The GRAPH Network

Infectiously enthusiastic about microbes and Global Health

---

---

## Références

Certains matériaux de cette leçon ont été adaptés des sources suivantes :

- Horst, A. (2021). *Dplyr-learnr*. <https://github.com/allisonhorst/dplyr-learnr> (Œuvre originale publiée en 2020)
- Sélectionner des lignes en utilisant les valeurs des colonnes—Filter. (n.d.). Consulté le 12 janvier 2022, à partir de <https://dplyr.tidyverse.org/reference/filter.html>

Les œuvres d'art ont été adaptées de :

- Horst, A. (2021). *Illustrations R & stats par Allison Horst*. <https://github.com/allisonhorst/stats-illustrations> (Œuvre originale publiée en 2018)

# Notes de leçon | Modifier les colonnes

February 2024



Introduction	.....
Objectifs d'apprentissage	.....
Packages	.....
Jeux de données	.....
Introduction à <code>mutate()</code>	.....
Création d'une variable à partir de zéro (indice de ligne)	.....
Création d'une variable booléenne	.....
Création d'une variable numérique basée sur une formule	.....
Changer le type d'une variable	.....
Entier : <code>as.integer</code>	.....
En Résumé !	.....

---

## Introduction

Vous savez désormais comment conserver ou supprimer des colonnes et des lignes de votre ensemble de données. Aujourd'hui, vous allez apprendre comment modifier des variables existantes ou en créer de nouvelles, en utilisant la fonction `mutate()` de `{dplyr}`. C'est une étape essentielle dans la plupart des projets d'analyse de données.

Allons-y!

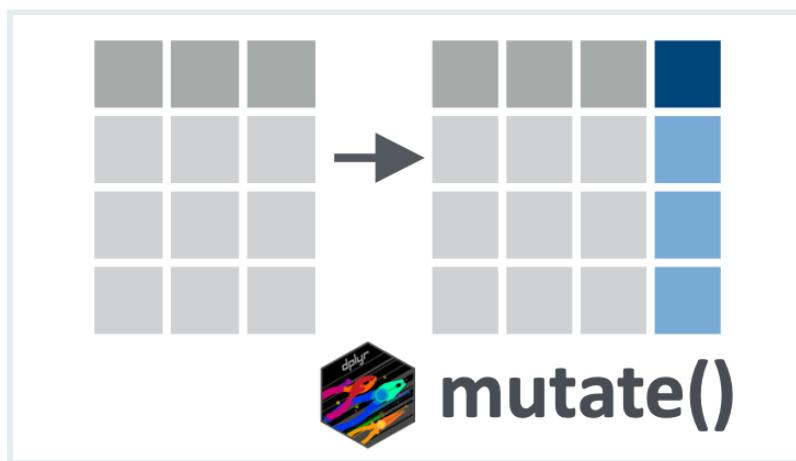


Fig: la fonction `mutate()`.

---

## Objectifs d'apprentissage

1. Vous pouvez utiliser la fonction `mutate()` du package `dplyr` pour créer de nouvelles variables ou modifier des variables existantes.
  2. Vous pouvez créer de nouvelles variables numériques, caractères, facteurs et booléennes.
- 

## Packages

Cette leçon nécessitera les “packages” chargés ci-dessous:

```
uire(pacman)) install.packages("pacman")
p_load(here,
       janitor,
       tidyverse)
```

---

## Jeux de données

Dans cette leçon, nous utiliserons à nouveau les données de l'enquête sérologique COVID-19 menée à Yaoundé, au Cameroun. Ci-dessous, nous importons l'ensemble de données `yaounde` et créons un sous-ensemble plus petit appelé `yao`. Notez que cet ensemble de données est légèrement différent de celui utilisé dans la leçon précédente.

```
<- read_csv(here::here('data/yaounde_data.csv'))

sous-ensemble plus petit de variables
yaounde %>% select(date_surveyed,
                      age,
                      weight_kg, height_cm,
                      symptoms, is_smoker)
```

<b>date_surveyed</b>	<b>age</b>	<b>weight_kg</b>	<b>height_cm</b>	<b>symptoms</b>	<b>is_smoker</b>
2020-10-22	45	95	169	Muscle pain	Non-smoker
2020-10-24	55	96	185	No symptoms	Ex-smoker
2020-10-24	23	74	180	No symptoms	Smoker
2020-10-22	20	70	164	Rhinitis	Non-smoker
2020-10-22	55	67	147	No symptoms	Non-smoker
2020-10-25	17	65	162	Fever-Cough	Non-smoker
2020-10-25	13	65	150	Sneezing	Non-smoker
2020-10-24	28	62	173	Headache	Non-smoker
2020-10-24	30	73	170	Fever-Rash	Non-smoker
2020-10-24	13	56	153	No symptoms	Non-smoker

1–10 of 971 rows

Previous 1 2 3 4 5 ... 98 Next

Nous utiliserons également un jeu de données issu d'une étude transversale visant à déterminer la prévalence de la sarcopénie chez la population âgée (>60 ans) au Karnataka, en Inde. La sarcopénie est une condition courante chez les personnes âgées, caractérisée par une perte progressive et généralisée de la masse et de la force musculaires squelettiques. Les données ont été obtenues de Zenodo [ici](#), et la publication source peut être trouvée [ici](#).

Ci-dessous, nous importons et visualisons cet ensemble de données :

```
mia <- read_csv(here::here('data/sarcopenia_elderly.csv'))
```

number	age	age_gr...	sex_ma...	marital_...	height_...	wei...
7	60.8	Sixties	0	married	1.57	58
8	72.3	Seventies	1	married	1.65	72
9	62.6	Sixties	0	married	1.59	64
12	72	Seventies	0	widow	1.473	54.5
13	60.1	Sixties	0	married	1.55	47
19	60.6	Sixties	0	married	1.422	64
45	60.1	Sixties	1	widower	1.68	60
46	60.2	Sixties	0	married	1.8	64.6
51	63	Sixties	0	married	1.6	57.8
56	60.4	Sixties	0	married	1.6	71.8

1–10 of 239 rows

Previous    1    2    3    4    5    ...    24    Next

**SIDE NOTE**



Notez que le jeu de donnée COVID-19 Yaoundé et sarcopénie sont en anglais !

Pour cette leçon, nous utiliserons ces versions en anglais. Mais dans d'autres leçons, nous utiliserons des versions partialement en français.

## Introduction à `mutate()`



La fonction `mutate()`. (Dessin adapté d'Allison Horst)

Nous utilisons `dplyr::mutate()` pour créer de nouvelles variables ou modifier des variables existantes. La syntaxe est assez intuitive et ressemble généralement à `df %>% mutate(nouveau_nom_de_colonne = ce_qu'il_contient)`.

Voyons un exemple rapide.

L'ensemble de données `yaounde` contient actuellement une colonne appelée `height_cm`, qui montre la taille, en centimètres, des répondants à l'enquête. Créons un `dataframe`, `yao_height`, avec juste cette colonne, pour une illustration facile :

```
jht <- yaounde %>% select(height_cm)  
jht
```

```
## # A tibble: 5 × 1  
##   height_cm  
##       <dbl>  
## 1     169  
## 2     185  
## 3     180  
## 4     164  
## 5     147
```

Que faire si vous voulez créer une nouvelle variable, appelée `height_meters`, où les hauteurs sont converties en mètres? Vous pouvez utiliser `mutate()` pour cela, avec l'argument `height_meters = height_cm/100`:

```
jht %>%  
  (height_meters = height_cm/100)
```

```
## # A tibble: 5 × 2  
##   height_cm     height_meters  
##       <dbl>           <dbl>  
## 1       169            1.69  
## 2       185            1.85  
## 3       180            1.8  
## 4       164            1.64  
## 5       147            1.47
```

Super. La syntaxe est magnifiquement simple, n'est-ce pas?

Il est parfois utile de penser aux fonctions de manipulation de données dans le contexte d'un logiciel de tableur familier. Voici à quoi correspondrait la commande R `mutate(height_m = height_cm/100)` dans Google Sheets :

**SIDE NOTE**



	A	B
1	height_cm	
2	169	
3	185	
4	180	
5	164	
6	147	
7	162	
8	150	
9		

Maintenant, imaginez qu'il y avait une petite erreur dans l'équipement utilisé pour mesurer les hauteurs des répondants, et que toutes les hauteurs sont inférieures de 5 cm. Vous aimeriez donc ajouter 5 cm à toutes les hauteurs de l'ensemble de données. Pour ce faire, plutôt que de créer une nouvelle variable comme vous l'avez fait auparavant, vous pouvez **modifier la variable existante** avec `mutate` :

```
jht %>%  
  (height_cm = height_cm + 5)
```

```
## # A tibble: 5 × 1  
##   height_cm
```

```
##      <dbl>
## 1      174
## 2      190
## 3      185
## 4      169
## 5      152
```

Encore une fois, très facile à faire!



La dataframe `sarcopenia` a une variable `weight_kg`, qui contient le poids des répondants en kilogrammes. Créez une nouvelle colonne, appelée `weight_grams`, avec le poids des répondants en grammes. Stockez votre réponse dans l'objet `Q_weight_to_g`. (1 kg équivaut à 1000 grammes.)

Étez le code avec votre réponse:

```
Q_weight_to_g <-  
  sarcopenia %>%
```

J'espère que vous voyez maintenant que la fonction `mutate` est très conviviale. En théorie, nous pourrions terminer la leçon ici, car vous savez maintenant comment utiliser `mutate()` 😊. Mais bien sûr, le diable est dans les détails - ce qui est intéressant, ce n'est pas `mutate()` lui-même mais ce qui se trouve à l'intérieur de l'appel `mutate()`.

Le reste de la leçon traitera de quelques cas d'utilisation du verbe `mutate()`. Dans ce processus, nous aborderons plusieurs nouvelles fonctions que vous n'avez pas encore rencontrées.

## Création d'une variable à partir de zéro (indice de ligne)

Souvent, vous allez créer des variables qui **référencent des variables existantes** (par exemple, la variable `height_meters` créée ci-dessus fait référence à la variable `height_cm`).

Parfois, vous créez des variables “à partir de zéro” sans faire référence à d'autres.

Regardons un exemple. Nous allons créer un indice de ligne avec `mutate()` et `1:n()` pour générer une séquence d'indices de lignes.

Le `n()` dans `dplyr` retourne le nombre de lignes du dataframe.

```
jht %>%  
  (row_index = 1:n())
```

```
## # A tibble: 5 × 2  
##   height_cm row_index  
##       <dbl>     <int>  
## 1       169         1  
## 2       185         2  
## 3       180         3  
## 4       164         4  
## 5       147         5
```

Ajoutez une variable `respondent_id` à `sarcopenia` pour contenir le numéro de ligne.

```
étez le code :  
penia_resp_id <-  
penia %>%  
  (_____)
```

Note : La base de données `sarcopenia` comporte 10 colonnes, qui ne tiendront probablement pas toutes dans votre console lorsque vous imprimerez cette base de données. Votre nouvelle colonne, `respondent_id` sera probablement cachée.



```
5   13 60.6          0 above poverty l... married  
6   19 60.5          0 above poverty l... married  
7   45 60.4          1 above poverty l... widower  
8   46 60.4          0 above poverty l... married  
9   51 63            0 above poverty l... married  
10  56 60.3          0 above poverty l... married  
# ... with 229 more rows, and 6 more variables:  
#   height_meters <dbl>, weight_kg <dbl>,  
#   gait_speed_meters_per_second <chr>, gripstrength_kg <dbl>,  
#   skeletal_muscle_index <dbl>, respondent_id <int>  
> | Column hidden
```

Ainsi, pour voir réellement la nouvelle colonne, vous pouvez utiliser `View()` pour visualiser l'ensemble du cadre de données ou `select()` pour sélectionner les colonnes concernées.

## Création d'une variable booléenne

Vous pouvez utiliser `mutate()` pour créer une variable booléenne pour catégoriser une partie de votre population.

Ci-dessous, nous créons une variable booléenne, `is_child` qui est soit `TRUE` si le sujet est un enfant ou `FALSE` si le sujet est un adulte (d'abord, nous sélectionnons uniquement la variable `age` pour qu'il soit facile de voir ce qui est fait ; il est probable que vous n'ayez pas besoin de cette présélection pour vos propres analyses).

```
: (age) %>%
  (is_child = age <= 18)
```

```
## # A tibble: 5 × 2
##       age is_child
##   <dbl> <lgl>
## 1     45 FALSE
## 2     55 FALSE
## 3     23 FALSE
## 4     20 FALSE
## 5     55 FALSE
```

Le code `age <= 18` évalue si chaque âge est inférieur ou égal à 18. Les âges qui correspondent à cette condition (18 ans et moins) sont `TRUE` et ceux qui ne répondent pas à la condition sont `FALSE`.

Une telle variable est utile pour, par exemple, compter le nombre d'enfants dans le jeu de données. Le code ci-dessous le fait avec la fonction `janitor::tabyl()` :

```
  (is_child = age <= 18) %>%
  (is_child)
```

```
##   is_child    n   percent
##   FALSE  662  0.6817714
##   TRUE   309  0.3182286
```

Vous pouvez observer que 31,8% (0,318...) des répondants de l'ensemble de données sont des enfants.

Voyons un autre exemple, car le concept de variables booléennes peut être un peu déroutant. La variable `symptoms` rapporte tous les symptômes respiratoires ressentis par le patient :

```
(symptoms)
```

```
## # A tibble: 5 × 1
##   symptoms
##   <chr>
## 1 Muscle pain
## 2 No symptoms
```

```
## 3 No symptoms
## 4 Rhinitis--Sneezing--Anosmia or ageusia
## 5 No symptoms
```

Vous pourriez créer une variable booléenne, appelée `has_no_symptoms`, qui est définie sur `TRUE` si le répondant n'a signalé aucun symptôme :

```
: (symptoms) %>%
  (has_no_symptoms = symptoms == "No symptoms")
```

```
## # A tibble: 5 × 2
##   symptoms
##   <chr>
## 1 Muscle pain
## 2 No symptoms
## 3 No symptoms
## 4 Rhinitis--Sneezing--Anosmia or ageusia
## 5 No symptoms
##   has_no_symptoms
##   <lgl>
## 1 FALSE
## 2 TRUE
## 3 TRUE
## 4 FALSE
## 5 TRUE
```

De même, vous pourriez créer une variable booléenne appelée `has_any_symptoms` qui est définie sur `TRUE` si le répondant a signalé des symptômes. Pour cela, vous échangerez simplement le code `symptoms == "No symptoms"` pour `symptoms != "No symptoms"` :

```
: (symptoms) %>%
  (has_any_symptoms = symptoms != "No symptoms")
```

```
## # A tibble: 5 × 2
##   symptoms
##   <chr>
## 1 Muscle pain
## 2 No symptoms
## 3 No symptoms
## 4 Rhinitis--Sneezing--Anosmia or ageusia
## 5 No symptoms
##   has_any_symptoms
##   <lgl>
## 1 TRUE
## 2 FALSE
## 3 FALSE
```

```
## 4 TRUE  
## 5 FALSE
```

Toujours confus par les exemples booléens? C'est normal. Prenez une pause et jouez un peu avec le code ci-dessus. Ensuite, essayez la question de pratique ci-dessous.



Les femmes ayant une force de préhension inférieure à 20 kg sont considérées comme ayant une faible force de préhension. Avec un sous-ensemble féminin du dataframe `sarcopenia`, ajoutez une variable appelée `low_grip_strength` qui est `TRUE` pour les femmes avec une force de préhension < 20 kg et `FALSE` pour les autres femmes.

```
Étez le code avec votre réponse :  
women_low_grip_strength <-  
  sarcopenia %>%  
  filter(sex_male_1_female_0 == 0) %>% # d'abord, nous filtrons  
  # l'ensemble de données pour seulement les femmes  
  mutate(_____) # code mutate ici
```

Quel pourcentage de femmes interrogées ont une faible force de préhension selon la définition ci-dessus? Entrez votre réponse sous forme de nombre sans guillemets (par exemple, 43.3 ou 12.2), à une décimale près.

```
women_low_grip_strength <- "VOTRE RÉPONSE"
```

## Création d'une variable numérique basée sur une formule

Maintenant, regardons un exemple de création d'une variable numérique, l'indice de masse corporelle (IMC), qui est un indicateur de santé couramment utilisé. La formule de l'indice de masse corporelle peut être écrite comme suit :

$$IMC = \frac{\text{poids(kilogrammes)}}{\text{taille(mètres)}^2}$$

Vous pouvez utiliser `mutate()` pour calculer l'IMC dans l'ensemble de données `yao` comme suit :

```


`> (weight_kg, height_cm) %>%
`> #ord obtenir la taille en mètres
`> (height_meters = height_cm/100) %>%
`> # utiliser la formule de l'IMC
`> (bmi = weight_kg / (height_meters)^2)


```

```


## # A tibble: 5 × 4
##   weight_kg height_cm height_meters   bmi
##       <dbl>     <dbl>         <dbl> <dbl>
## 1       95      169        1.69  33.3
## 2       96      185        1.85  28.0
## 3       74      180        1.8    22.8
## 4       70      164        1.64  26.0
## 5       67      147        1.47  31.0


```

Sauvegardons le dataframe avec les IMC pour plus tard. Nous l'utiliserons dans la section suivante.

```


<-
`> %
`> (weight_kg, height_cm) %>%
`> #ord obtenir la taille en mètres
`> (height_meters = height_cm/100) %>%
`> # utiliser la formule de l'IMC
`> (bmi = weight_kg / (height_meters)^2)


```

La masse musculaire appendiculaire (MMA), un indicateur de santé utile, est la somme de la masse musculaire dans les 4 membres. Elle peut être prédite avec la formule suivante, appelée équation de Lee :



$$MMA(kg) = (0.244 \times poids(kg)) + (7.8 \times taille(m)) + (6.6 \times sexe) - 4.5$$

La variable `sex` dans la formule suppose que les hommes sont codés comme 1 et les femmes comme 0 (ce qui est déjà le cas pour notre ensemble de données `sarcopenia`). Le `- 4.5` à la fin est une constante utilisée pour les Asiatiques.

Calculez la valeur MMA pour tous les individus dans l'ensemble de données `sarcopenia`. Cette valeur devrait être dans une nouvelle colonne appelée `asm`



étez le code avec votre réponse:

```
calculation <-
penia %>%
  `as.m = _____` )
```

## Changer le type d'une variable

Dans votre flux de travail d'analyse de données, vous avez souvent besoin de redéfinir les *types* de variables. Vous pouvez le faire avec des fonctions telles que `as.integer()`, `as.factor()`, `as.character()` et `as.Date()` dans votre appel `mutate()`. Voyons un exemple de cela.

Entier : `as.integer`

`as.integer()` convertit toutes les valeurs numériques en entiers :

```
%>%
  bmi_integer = as.integer(bmi))
```

```
## # A tibble: 5 × 5
##   weight_kg height_cm height_meters   bmi
##       <dbl>     <dbl>         <dbl> <dbl>
## 1       95      169        1.69  33.3
## 2       96      185        1.85  28.0
## 3       74      180        1.8    22.8
## 4       70      164        1.64  26.0
## 5       67      147        1.47  31.0
##   bmi_integer
##   <int>
## 1 33
## 2 28
## 3 22
## 4 26
## 5 31
```

Notez que cela *tronque* les entiers plutôt que de les arrondir vers le haut ou le bas, comme vous pourriez vous y attendre. Par exemple, le BMI 22.8 à la troisième ligne est tronqué à 22. Si vous voulez des nombres arrondis, vous pouvez utiliser la fonction `round` de base R.





**PRO TIP** Utiliser `as.integer()` sur une variable factorielle est un moyen rapide de codifier des chaînes en nombres. Cela peut être essentiel pour certains traitements de données d'apprentissage automatique.

```
%>%  
  bmi_integer = as.integer(bmi),  
  bmi_rounded = round(bmi))
```

```
## # A tibble: 5 × 6  
##   weight_kg height_cm height_meters   bmi  
##       <dbl>     <dbl>          <dbl> <dbl>  
## 1       95      169        1.69  33.3  
## 2       96      185        1.85  28.0  
## 3       74      180        1.8    22.8  
## 4       70      164        1.64  26.0  
## 5       67      147        1.47  31.0  
  
##   bmi_integer bmi_rounded  
##       <int>      <dbl>  
## 1       33        33  
## 2       28        28  
## 3       22        23  
## 4       26        26  
## 5       31        31
```

**SIDE NOTE**



La fonction `round()` de base R arrondit “à la moitié vers le bas”. C'est-à-dire que le nombre 3.5, par exemple, est arrondi à 3 par `round()`. C'est étrange. La plupart des gens s'attendent à ce que 3.5 soit arrondi *vers le haut* à 4, et non pas *vers le bas* à 3. La plupart du temps, vous voudrez donc utiliser la fonction `round_half_up()` de `janitor`.

**CHALLENGE**



Dans les leçons futures, vous découvrirez comment manipuler les dates et comment convertir en un type de date en utilisant `as.Date()`.

**PRACTICE**



Utilisez `as_integer()` pour convertir les âges des répondants dans l'ensemble de données `sarcopenia` en entiers (en les tronquant au

passage). Cela devrait aller dans une nouvelle colonne appelée `age_integer`.



Étez le code avec votre réponse :

```
integer <-  
  openia %>%  
  e(age_integer = _____)
```

## En Résumé !

Comme vous pouvez l'imaginer, transformer des données est une étape essentielle dans tout flux de travail d'analyse de données. Il est souvent nécessaire de nettoyer les données et de les préparer pour d'autres analyses statistiques ou pour créer des graphiques. Et comme vous l'avez vu, il est assez simple de transformer des données avec la fonction `mutate()` de `dplyr`, bien que certaines transformations soient plus délicates à réaliser que d'autres.

Félicitations d'être arrivé jusqu'ici.

Mais votre périple de manipulation de données n'est pas encore terminé ! Dans nos prochaines leçons, nous apprendrons à créer des résumés de données complexes et comment créer et travailler avec des groupes de data frame. Intrigué ? À bientôt dans la prochaine leçon.

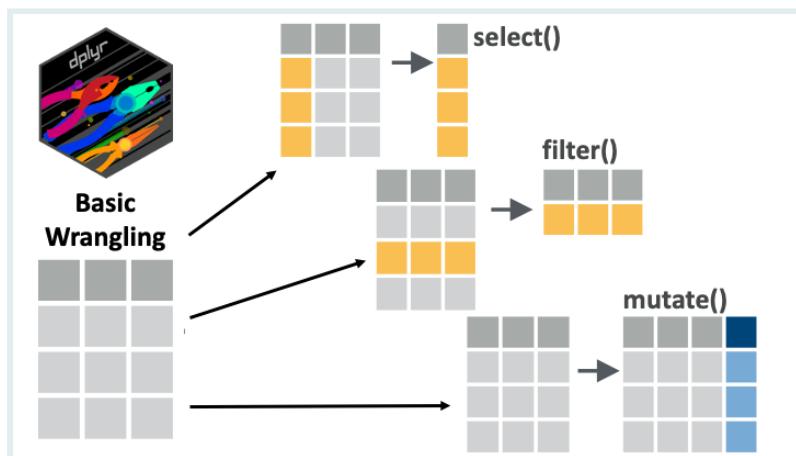


Fig : Manipulation basique des données avec `select()`, `filter()`, et `mutate()`.

## Contributeurs

Les membres de l'équipe suivants ont contribué à cette leçon :



## LAURE VANCAUWENBERGHE

Data analyst, the GRAPH Network

A firm believer in science for good, striving to ally programming, health and education

---



## ANDREE VALLE CAMPOS

R Developer and Instructor, the GRAPH Network

Motivated by reproducible science and education

---



## KENE DAVID NWOSU

Data analyst, the GRAPH Network

Passionate about world improvement

---



## SABINA RODRIGUEZ VELÁSQUEZ

Project Manager and Scientific Collaborator, The GRAPH Network

Infectiously enthusiastic about microbes and Global Health

---

## Références

Certains matériaux de cette leçon ont été adaptés des sources suivantes :

- Horst, A. (2022). *Dplyr-learnr*. <https://github.com/allisonhorst/dplyr-learnr> (Travail original publié en 2020)
- *Créer, modifier, et supprimer des colonnes — Mutate*. (n.d.). Consulté le 21 février 2022, à partir de <https://dplyr.tidyverse.org/reference/mutate.html>
- *Appliquer une fonction (ou des fonctions) sur plusieurs colonnes — Across*. (n.d.). Consulté le 21 février 2022, à partir de <https://dplyr.tidyverse.org/reference/across.html>

Les illustrations ont été adaptées de :

- Horst, A. (2022). *Illustrations R & stats par Allison Horst*. <https://github.com/allisonhorst/stats-illustrations> (Travail original publié en 2018)

Autres références :

- Lee, Robert C, ZiMian Wang, Moonseong Heo, Robert Ross, Ian Janssen, et Steven B Heymsfield. “Total-Body Skeletal Muscle Mass: Development and Cross-Validation of Anthropometric Prediction Models.” *The American Journal*

*of Clinical Nutrition* 72, no. 3 (2000): 796–803. <https://doi.org/10.1093/ajcn/72.3.796>.

# Notes de leçon | Transformation conditionnelle

February 2024



Introduction	.....
Objectifs d'apprentissage	.....
Packages	.....
Jeux de données	.....
Rappel : opérateurs relationnels (comparateurs) sur R	.....
Introduction à <code>case_when()</code>	.....
L'argument par défaut <code>TRUE</code>	.....
Appariement des NA avec <code>is.na()</code>	.....
Conserver les valeurs par défaut d'une variable	.....
Conditions multiples sur une seule variable	.....
Conditions multiples sur variables multiples	.....
Ordre de priorité des conditions dans <code>case_when()</code>	.....
Conditions superposées avec <code>case_when()</code>	.....
Conditions binaires : <code>dplyr::if_else()</code>	.....
Récapitulatif	.....

## Introduction

Dans la dernière leçon, vous avez appris les bases de la transformation des données en utilisant la fonction `mutate()` de `{dplyr}`.

Dans cette leçon, nous avons principalement examiné les transformations *globales* ; c'est-à-dire, des transformations qui font la même chose à une variable entière.

Dans cette leçon, nous allons voir comment manipuler *conditionnellement* certaines lignes en fonction de si elles répondent ou non à des critères définis.

Pour cela, nous utiliserons principalement la fonction `case_when()`, que vous considérerez probablement comme l'une des fonctions les plus importantes de `{dplyr}` pour les tâches de préparation des données.

Commençons.



Fig: les conditions `case_when()`.

---

## Objectifs d'apprentissage

1. Vous pouvez transformer ou créer de nouvelles variables en fonction des conditions en utilisant `dplyr::case_when()`
2. Vous savez comment utiliser la condition `TRUE` dans `case_when()` pour faire correspondre les cas non appariés.
3. Vous pouvez gérer les valeurs `NA` dans les transformations `case_when()`.
4. Vous comprenez comment conserver les valeurs par défaut d'une variable dans une formule `case_when()`.
5. Vous pouvez écrire des conditions `case_when()` impliquant plusieurs comparateurs et plusieurs variables.
6. Vous comprenez l'ordre de priorité des conditions `case_when()`.
7. Vous pouvez utiliser `dplyr::if_else()` pour l'assignation conditionnelle binaire.

---

## Packages

Cette leçon nécessitera la suite de package tidyverse :

```
aire(pacman) install.packages("pacman")
p_load(tidyverse)
```

---

## Jeux de données

Dans cette leçon, nous utiliserons à nouveau les données de l'enquête sérologique COVID-19 menée à Yaoundé, au Cameroun.

```

ter et visualiser le jeu de données
<- read_csv(here::here('data/fr_yaounde_data.csv')) %>%
  #indre manquant chaque 5ème âge
  #e(age = case_when(row_number() %in% seq(5, 900, by = 5) ~ NA_real_,
  #                  TRUE ~ age)) %>%
  #ommer la variable âge
  #e(age_annees = age) %>%
  #rimer la colonne de catégorie d'âge
  :(-cat_age)

```

Notez que dans le bloc de code ci-dessus, nous avons légèrement modifié la colonne d'âge, en introduisant artificiellement quelques valeurs manquantes, et nous avons également supprimé la colonne `cat_age`. Ceci pour aider à illustrer certains points clés du tutoriel.

Pour les questions de pratique, nous utiliserons également une liste d'une épidémie de 136 cas d'influenza A H7N9 lors d'une **épidémie en 2013** en Chine. Il s'agit d'une version modifiée d'un ensemble de données compilé par Kucharski et al. (2014).

```

ter et afficher l'ensemble de données
flu <- read_csv(here::here('data/fr_fiu_h7n9_china_2013.csv'))
flu

```

## Rappel : opérateurs relationnels (comparateurs) sur R

Tout au long de cette leçon, vous utiliserez beaucoup d'opérateurs relationnels en R. Rappelons que les opérateurs relationnels, parfois appelés "comparateurs", testent la relation entre deux valeurs, et renvoient `TRUE`, `FALSE` ou `NA`.

Une liste des opérateurs les plus couramment utilisés est donnée ci-dessous :

<b>Opérateur est VRAI si</b>	
<code>A &lt; B</code>	A est <b>inférieur à</b> B
<code>A &lt;= B</code>	A est <b>inférieur ou égal à</b> B
<code>A &gt; B</code>	A est <b>supérieur à</b> B
<code>A &gt;= B</code>	A est <b>supérieur ou égal à</b> B
<code>A == B</code>	A est <b>égal à</b> B
<code>A != B</code>	A est <b>différent de</b> B
<code>A %in% B</code>	A est un élément de B

## Introduction à `case_when()`

Pour se familiariser avec `case_when()`, commençons par une simple transformation conditionnelle sur la colonne `age_annees` de l'ensemble de données `yaounde`. Nous commençons par extraire uniquement la colonne `age_annees` du jeu de données pour illustrer facilement :

```
age <-
ie %>%
  (age_annees)

age
```

Maintenant, en utilisant `case_when()`, nous pouvons créer une nouvelle colonne, appelée “`age_groupe`”, qui a la valeur “Enfant” si la personne a moins de 18 ans, et “Adulte” si la personne a 18 ans et plus :

```
age %>%
  (age_groupe = case_when(age_annees < 18 ~ "Enfant",
                           age_annees >= 18 ~ "Adulte"))
```

La syntaxe de `case_when()` peut sembler un peu étrangère, mais elle est assez simple : du côté gauche (LHS) du signe `~` (appelé “tilde”), vous fournissez la ou les conditions que vous voulez évaluer, et du côté droit (RHS), vous fournissez une valeur à insérer si la condition est vraie.

Donc, la déclaration `case_when(age_annees < 18 ~ "Enfant", age_annees >= 18 ~ "Adulte")` peut se lire comme suit : “si `age_annees` est inférieur à 18, insérez ‘Enfant’, sinon si `age_annees` est supérieur ou égal à 18, insérez ‘Adulte’”.

### Formules, LHS et RHS

#### VOCAB



Chaque ligne d'un appel à `case_when()` est appelée une “formule” ou, parfois, une “formule à deux côtés”. Et chaque formule a un côté gauche (LHS) et un côté droit (RHS).

Par exemple, le code `age_annees < 18 ~ "Enfant"` est une “formule”, son LHS est `age_annees < 18` tandis que son RHS est “`Enfant`”.

Vous allez probablement rencontrer ces termes en lisant la documentation pour la fonction `case_when()`, et nous les utiliserons également dans cette leçon.

Après avoir créé une nouvelle variable avec `case_when()`, il est recommandé de l'inspecter minutieusement pour s'assurer qu'elle a fonctionné comme prévu.

Pour inspecter la variable, vous pouvez passer votre jeu de données dans la fonction `View()` pour la visualiser sous forme de tableau :

```
age %>%
  age_groupe = case_when(age_annees < 18 ~ "Enfant",
                         age_annees >= 18 ~ "Adulte")) %>%
```

Cela ouvrirait un nouvel onglet dans RStudio où vous devriez manuellement scanner la nouvelle colonne, `age_groupe` et la colonne référencée `age_annees` pour vous assurer que votre déclaration `case_when()` a fait ce que vous vouliez qu'elle fasse.

Vous pourriez aussi passer la nouvelle colonne dans la fonction `tabyl()` pour vous assurer que les proportions “ont du sens” :

```
age %>%
  age_groupe = case_when(age_annees < 18 ~ "Enfant",
                         age_annees >= 18 ~ "Adulte")) %>%
  (age_groupe)
```

Avec les données `liste_influ`, créez une nouvelle colonne, appelée `age_groupe`, qui a la valeur “Moins de 50” pour les personnes de moins de 50 ans et “50 et plus” pour les personnes âgées de 50 ans et plus. Utilisez la fonction `case_when()`.

étez le code avec votre réponse :

```
coupe <- liste_influ %>%
  (age_groupe = _____)
```



Parmi l'ensemble des individus dans le jeu de données `liste_influ`, quel pourcentage est confirmé comme étant inférieur à 60 ans ? (Répétez la procédure ci-dessus mais avec le seuil de 60, puis appelez `tabyl()` sur la variable de groupe d'âge. Utilisez la colonne `percent`, pas la colonne `valid_percent`).

ez votre réponse sous forme d'un nombre ENTIER sans guillemets :

```
coupe_pourcentage(_____)
```

## L'argument par défaut TRUE

Dans une déclaration `case_when()`, vous pouvez utiliser une condition littérale `TRUE` pour faire correspondre toutes les lignes non encore appariées avec les conditions fournies.

Par exemple, si nous ne gardons que la première condition de l'exemple précédent, `age_annees < 18`, et définissons la valeur par défaut à `TRUE ~ "Pas enfant"`, alors tous les adultes et les valeurs `NA` dans l'ensemble de données seront étiquetés "Pas enfant" par défaut.

```
age %>%  
  age_groupe = case_when(age_annees < 18 ~ "Enfant",  
                         TRUE ~ "Pas enfant"))
```

Cette condition `TRUE` peut être lue comme "pour tout le reste..." .

Ainsi, la déclaration `case_when()` utilisée précédemment, `age_annees < 18 ~ "Enfant"`, `TRUE ~ "Pas enfant"`, se lirait alors comme suit : "si l'âge est inférieur à 18, entrez 'Enfant' et pour tout le monde qui n'a pas encore été apparié, entrez 'Pas enfant'" .

Il est important d'utiliser `TRUE` comme condition finale dans `case_when()`. Si vous l'utilisez comme première condition, elle aura la priorité sur toutes les autres, comme on peut le voir ici :

### WATCH OUT



```
age %>%  
  age_groupe = case_when(TRUE ~ "Pas enfant",  
                         age_annees < 18 ~ "Enfant"))
```

Comme vous pouvez le constater, tous les individus sont maintenant codés comme "Pas enfant", parce que la condition `TRUE` a été placée en premier, et a donc pris le pas. Nous examinerons plus loin la question de la priorité.

## Appariement des NA avec `is.na()`

Nous pouvons appairer les valeurs manquantes manuellement avec `is.na()`. Ci-dessous, nous appairons les âges `NA` avec `is.na()` et définissons leur groupe d'âge à "Age manquant" :

```
age %>%
  age_groupe = case_when(age_annees < 18 ~ "Enfant",
                         age_annees >= 18 ~ "Adulte",
                         is.na(age_annees) ~ "Age manquant"))
```



Comme précédemment, en utilisant les données `liste_influ`, créez une nouvelle colonne, appelée `age_groupe`, qui a la valeur “Moins de 60” pour les personnes de moins de 60 ans et “60 et plus” pour les personnes âgées de 60 ans et plus. Mais cette fois, affectez également à ceux dont l’âge est manquant la valeur “Age manquant”.

Étez le code avec votre réponse :

```
coupe_nas <-
  influ %>% _____
```



La colonne `sexé` du jeu de données `liste_influ` contient les valeurs “f”, “m” et `NA` :

```
influ %>%
  (sexé)
```

Recodez “f”, “m” et `NA` respectivement en “Femme”, “Homme” et “Sexe manquant”. Vous devez modifier la colonne `sexé` existante, pas créer une nouvelle colonne.

Étez le code avec votre réponse :

```
recode <-
  influ %>%
  (sexé = _____)
```

## Conserver les valeurs par défaut d’une variable

Le côté droit (RHS) d’une formule `case_when()` peut également prendre une variable de votre jeu de données. C’est souvent utile lorsque vous voulez changer seulement quelques valeurs dans une colonne.

Voyons un exemple avec la colonne `edu_haute`, qui contient le plus haut niveau d’éducation atteint par un répondant :

```
educ <-
ie %>%
:(edu_haute)
educ
```

Ci-dessous, nous créons une nouvelle colonne, `edu_haute_recode`, où nous recodons à la fois “University” et “Doctorate” par la valeur “Post-secondary” :

```
educ %>%
`:(edu_haute_recode =
  case_when(
    edu_haute %in% c("University", "Doctorate") ~ "Post-secondary"
  ))
```

Ça a fonctionné, mais maintenant nous avons des `NA` pour toutes les autres lignes. Pour conserver ces autres lignes à leurs valeurs par défaut, nous pouvons ajouter la ligne `TRUE ~ edu_haute` (avec une variable, `edu_haute`, sur le côté droit d'une formule) :

```
educ %>%
`:(edu_haute_recode =
  case_when(
    edu_haute %in% c("University", "Doctorate") ~ "Post-secondary",
    TRUE ~ edu_haute
  ))
```

Maintenant, la déclaration `case_when()` se lit : ‘Si l’éducation la plus élevée est “University” ou “Doctorate”, inscrivez “Post-secondary”. Pour tout le monde, inscrivez la valeur de `edu_haute`’.

Ci-dessus, nous avons placé les valeurs recodées dans une colonne séparée, `edu_haute_recode`, mais pour ce type de remplacement, il est plus courant de simplement écraser la colonne existante :

```
educ %>%
`:(edu_haute =
  case_when(
    edu_haute %in% c("University", "Doctorate") ~ "Post-secondary",
    TRUE ~ edu_haute
  ))
```

Nous pouvons lire cette dernière déclaration `case_when()` comme suit : ‘Si l’éducation la plus élevée est “University” ou “Doctorate”, changez la valeur en “Post-secondary”. Pour tout le monde, laissez la valeur de `edu_haute`’.



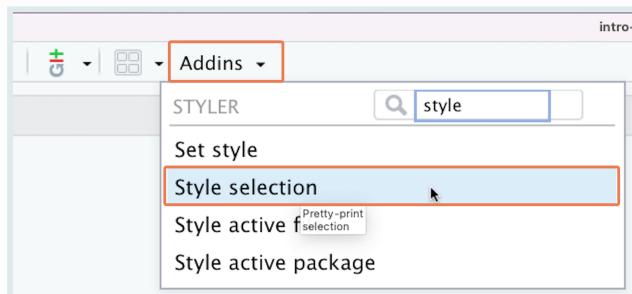
étez le code avec votre réponse :

```
er_recovery <-
  influ %>%
  `>(devenir = _____)
```

(Nous savons que c'est beaucoup de code pour un si petit changement. Plus tard, vous verrez des façons plus simples de faire cela.)

**Éviter les lignes de code trop longues** Au fur et à mesure que vous commencez à écrire des instructions `case_when()` de plus en plus complexes, il devient utile d'utiliser des sauts de ligne pour éviter des lignes de code trop longues.

Pour vous aider à créer des sauts de ligne, vous pouvez utiliser le package `{styler}`. Installez-le avec `pacman::p_load(styler)`. Ensuite, pour reformater n'importe quelle partie du code, mettez en surbrillance le code, cliquez sur le bouton “Addins” dans RStudio, puis cliquez sur “Style selection” :



Alternativement, vous pourriez mettre en surbrillance le code et utiliser le raccourci `Shift + Command/Control + A` pour utiliser le reformateur de code intégré de RStudio. Parfois, `{styler}` fait un meilleur travail de reformatage. Parfois, le reformateur intégré fait un meilleur travail.

## Conditions multiples sur une seule variable

Les conditions LHS dans les formules `case_when()` peuvent avoir plusieurs parties. Voyons un exemple de cela.

Mais tout d'abord, nous allons nous inspirer de ce que nous avons appris dans la leçon `mutate()` et recréer la variable IMC. Cela implique d'abord de convertir la variable `taille_cm` en mètres, puis de calculer l'IMC.

```
IMC <-  
  le %>%  
    vertissons notre taille en mètres et définissons l'IMC  
    : (taille_m = taille_cm/100,  
      IMC = (poids_kg / (taille_m)^2)) %>%  
    ectionnons uniquement l'IMC  
    : (IMC)  
  
IMC
```

Rappelez-vous les catégories suivantes pour l'IMC:

- Si l'IMC est inférieur à 18,5, la personne est considérée comme étant en sous-poids.
- Un IMC normal est supérieur ou égal à 18,5 et inférieur à 25.
- Un IMC en surpoids est supérieur ou égal à 25 et inférieur à 30.
- Un IMC obèse est supérieur ou égal à 30.

La condition `IMC >= 18.5 & IMC < 25` pour définir Poids normal **est une condition composée car elle a deux comparateurs** : `>=` et `<`.

```
IMC <- yaounde_IMC %>%  
  (classification_IMC = case_when(IMC < 18.5 ~ 'Sous-poids',  
                                    IMC >= 18.5 & IMC < 25 ~ 'Poids normal',  
                                    IMC >= 25 & IMC < 30 ~ 'Surpoids',  
                                    IMC >= 30 ~ 'Obèse'))  
IMC
```

Utilisons `tabyl()` pour jeter un coup d'œil à nos données :

```
IMC %>%  
  (classification_IMC)
```

Mais vous pouvez voir que les niveaux d'IMC sont définis par ordre alphabétique de Obèse à Surpoids, au lieu de aller du plus léger (Sous-poids) au plus lourd (Obèse). Rappelez-vous que si vous voulez avoir un certain ordre, vous pouvez faire de `classification_IMC` un facteur en utilisant `mutate()` et définir ses niveaux.

```
IMC %>%  
  (classification_IMC = factor(classification_IMC, levels=c("Obèse",  
                                         "Surpoids",  
                                         "Poids normal",  
                                         "Sous-poids")) %>%  
classification_IMC
```

Avec les conditions composées, il faut se rappeler d'entrer le nom de la variable à chaque fois qu'il y a un comparateur. Les apprenants R oublient souvent cela et essaieront d'exécuter du code qui ressemble à ceci :

**WATCH OUT**



```
IMC %>%
  classification_IMC = case_when(IMC < 18.5 ~ 'Sous-poids',
                                    IMC >= 18.5 & < 25 ~ 'Poids
                                    normal',
                                    IMC >= 25 & < 30 ~ 'Surpoids',
                                    IMC >= 30 ~ 'Obèse'))
```

Les définitions pour les catégories “Poids normal” et “Surpoids” sont erronées. Voyez-vous le problème ? Essayez d'exécuter le code pour repérer l'erreur.

**PRACTICE**



(in RMD)

Avec les données `liste_influ`, créez une nouvelle colonne, appelée `adolescent`, qui a la valeur “Oui” pour les personnes âgées de 10 à 19 ans (au moins 10 ans et moins de 20 ans), et “Non” pour tous les autres.

Étez le code avec votre réponse :

```
adolescent_grouping <-
  influ %>% _____
```

## Conditions multiples sur variables multiples

Dans tous les exemples vus jusqu'à présent, vous n'avez utilisé que des conditions impliquant une seule variable à la fois. Mais les conditions de LHS se réfèrent souvent à plusieurs variables à la fois.

Voyons un exemple simple avec l'âge et le sexe dans le jeu de données `yaounde`. Tout d'abord, nous sélectionnons uniquement ces deux variables pour une illustration facile :

```
age_sexe <-
  yaounde %>%
  select(-age_annees, -sexe)

age_sexe
```

Maintenant, imaginons que nous voulons recruter des femmes et des hommes dans le groupe d'âge 20-29 ans dans deux études. Pour cela, nous aimerais créer une colonne, appelée `recruit`, avec le schéma suivant :

- Les femmes âgées de 20 à 29 ans devraient avoir la valeur “Recruter pour l'étude féminine”
- Les hommes âgés de 20 à 29 ans devraient avoir la valeur “Recruter pour l'étude masculine”
- Tous les autres devraient avoir la valeur “Ne pas recruter”

Pour faire cela, nous exécutons l'instruction `case_when` suivante :

```
age_sexe %>%
  recruit = case_when(
    == "Female" & age_annees >= 20 & age_annees <= 29 ~ "Recruter pour l'étude féminine",
    == "Male" & age_annees >= 20 & age_annees <= 29 ~ "Recruter pour l'étude masculine",
    ~ "Ne pas recruter"
```

Vous pourriez également ajouter des paires supplémentaires de parenthèses autour des critères d'âge dans chaque condition :

```
age_sexe %>%
  recruit = case_when(
    == "Female" & (age_annees >= 20 & age_annees <= 29) ~ "Recruter pour l'étude féminine",
    == "Male" & (age_annees >= 20 & age_annees <= 29) ~ "Recruter pour l'étude masculine",
    ~ "Ne pas recruter"
```

Cette paire supplémentaire de parenthèses ne change pas la sortie du code, mais elle améliore la cohérence parce que le lecteur peut voir visuellement que votre condition est composée de deux parties, une pour le sexe, `sexe == "Female"`, et une autre pour l'âge, `(age_annees >= 20 & age_annees <= 29)`.



Avec les données `liste_influ`, créez une nouvelle colonne, appelée `recruter` avec le schéma suivant :

- Les personnes âgées de 30 à 59 ans (au moins 30 ans, moins de 60 ans) de la province du Jiangsu devraient avoir la valeur “Recruter pour l'étude Jiangsu”

- Les personnes âgées de 30 à 59 ans de la province du Zhejiang devraient avoir la valeur “Recruter pour l'étude Zhejiang”
- Tous les autres devraient avoir la valeur “Ne pas recruter”



Étez le code avec votre réponse :

```
province_grouping <-
  influ %>%
  >(recruter = _____)
```

## Ordre de priorité des conditions dans `case_when()`

Notez que l'ordre des conditions est important, car les conditions listées en haut de votre instruction `case_when()` sont prioritaires sur les autres.

Pour comprendre cela, exécutez l'exemple ci-dessous :

```
age_sexe %>%
  age_groupe = case_when(age_annees < 18 ~ "Enfant",
                         age_annees < 30 ~ "Adulte jeune",
                         age_annees < 50 ~ "Adulte moyen",
                         age_annees < 120 ~ "Adulte agé"))
```

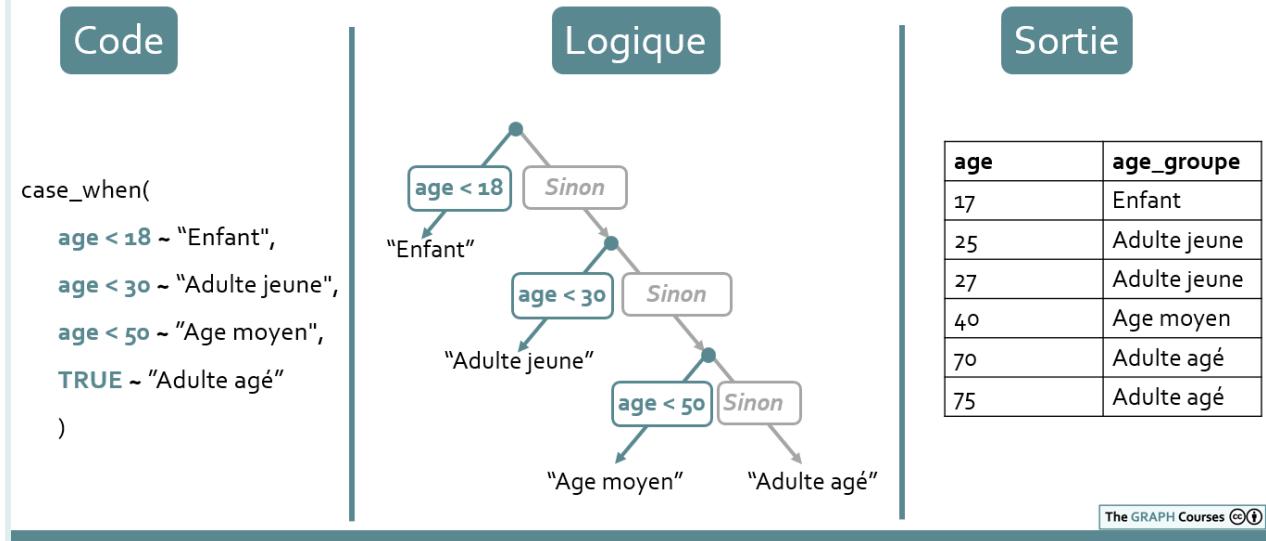
Au premier abord, cela ressemble à une instruction `case_when()` défectueuse car les conditions d'âge se chevauchent. Par exemple, l'instruction `age_annees < 120 ~ "Adulte agé"` (qui se lit “si l'âge est inférieur à 120, saisissez ‘Adulte agé’”) suggère que *n'importe qui* entre 0 et 120 ans (même un bébé de 1 an !) serait codé comme “Adulte agé”.

Mais comme vous l'avez vu, le code fonctionne bien ! Les personnes de moins de 18 ans sont toujours codées comme “Enfant”.

Que se passe-t-il ? Essentiellement, l'instruction `case_when()` est interprétée comme une série d'étapes logiques de branchement, en commençant par la première condition. Cette instruction particulière peut donc se lire comme suit : “Si l'âge est inférieur à 18 ans, saisissez ‘Enfant’, *sinon*, si l'âge est inférieur à 30 ans, saisissez ‘Adulte jeune’, *sinon*, si l'âge est inférieur à 120 ans, saisissez ‘Adulte agé’”.

Ceci est illustré dans le schéma ci-dessous :

## Ordre d'évaluation avec dplyr::case\_when



The GRAPH Courses

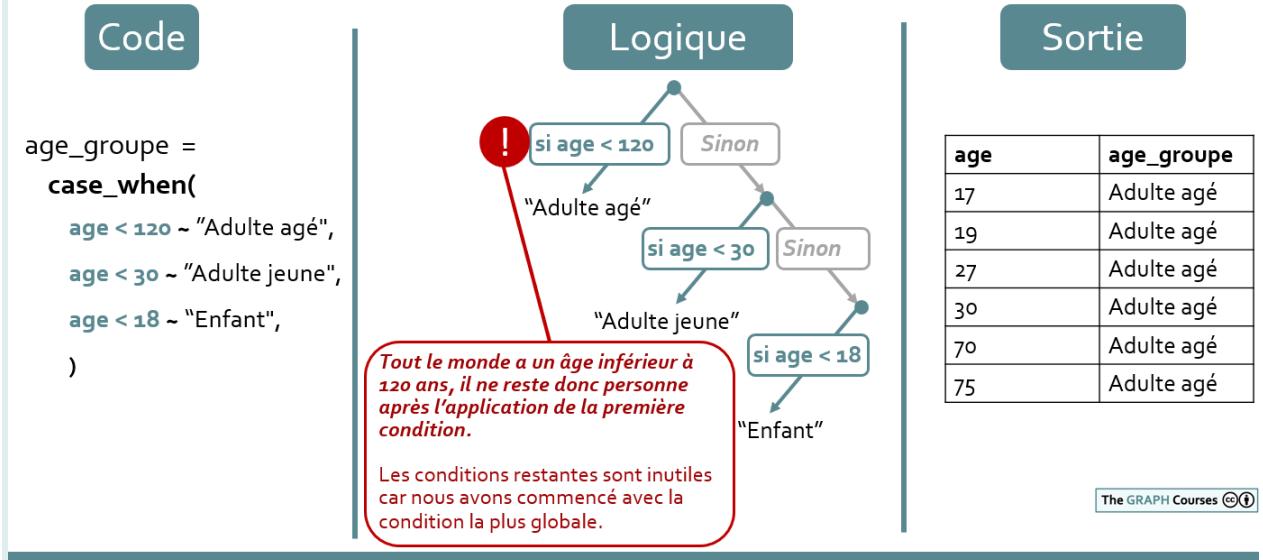
Cela signifie que si vous inversez l'ordre des conditions, vous obtiendrez une instruction `case_when()` érronée:

```
age %>%
  age_groupe = case_when(age_annees < 120 ~ "Adulte agé",
                         age_annees < 50 ~ "Adulte moyen",
                         age_annees < 30 ~ "Adulte jeune",
                         age_annees < 18 ~ "Enfant"))
```

Comme vous pouvez le constater, tout le monde est codé comme “Adulte agé”. Cela se produit parce que la première condition correspond à tout le monde, il ne reste donc plus personne pour correspondre aux conditions suivantes. L'instruction peut se lire “Si l'âge est inférieur à 120 ans, saisissez ‘Adulte agé’, *sinon* si l'âge est inférieur à 30 ans...”. Mais il n'y a pas de “sinon” car tout le monde a déjà été mis en correspondance !

Ceci est illustré dans le diagramme ci-dessous :

## Une instruction case\_when érronée



Bien que nous ayons passé beaucoup de temps à expliquer l'importance de l'ordre des conditions, dans cet exemple précis, il y aurait une façon beaucoup plus claire d'écrire ce code qui ne dépendrait pas de l'ordre des conditions. Plutôt que de laisser les groupes d'âge ouverts comme ceci :

```
age_annees < 120 ~ "Adulte agé"
```

vous devriez en fait utiliser des limites d'âge *fermées* comme ceci :

```
age_annees >= 30 & age_annees < 120 ~ "Adulte agé"
```

qui se lit : "si l'âge est supérieur ou égal à 30 ans et inférieur à 120 ans, saisissez 'Adulte agé'".

Avec de telles conditions fermées, l'ordre des conditions n'a plus d'importance. Vous obtenez le même résultat quel que soit l'arrangement des conditions :

```
icer avec la condition "Adulte agé"  
age %>%  
ivons des conditions de bornes fermées sur l'âge  
e(age_groupe = case_when(age_annees >= 30 & age_annees < 120 ~ "Adulte agé",  
                           age_annees >= 18 & age_annees < 30 ~ "Adulte jeune",  
                           age_annees >= 0 & age_annees < 18 ~ "Enfant"))
```

```
icer avec la condition "Enfant"  
age %>%  mutate(age_group = case_when(age_annees >= 0 & age_annees < 18 ~  
"Enfant",  
                           age_annees >= 18 & age_annees < 30 ~ "Adulte jeune",  
                           age_annees >= 30 & age_annees < 120 ~ "Adulte agé"))
```

Bien net et clair !

Alors pourquoi avons-nous passé autant de temps à expliquer l'importance de l'ordre des conditions si vous pouvez simplement éviter les catégories ouvertes et ne pas avoir à vous soucier de l'ordre des conditions ?

Une raison est que la compréhension de l'ordre des conditions devrait maintenant vous aider à voir pourquoi il est important de placer la condition `TRUE` comme dernière ligne de votre instruction `case_when()`. La condition `TRUE` correspond à *chaque ligne qui n'a pas encore été mise en correspondance*, donc si vous l'utilisez en premier dans le `case_when()`, elle correspondra à *tout le monde* !

L'autre raison est qu'il existe certains cas où vous voudrez *peut-être* utiliser des conditions qui se chevauchent et ouvertes, et vous devrez donc faire attention à l'ordre des conditions. Voyons un tel exemple maintenant : l'identification des symptômes de type COVID. Notez qu'il s'agit d'un matériel un peu avancé, probablement un peu au-dessus de vos besoins actuels. Nous l'introduisons maintenant pour que vous en soyez conscient et que vous restiez vigilant avec `case_when()` à l'avenir.

### Conditions superposées avec `case_when()`

Nous voulons identifier les symptômes semblables à ceux de la COVID-19 dans nos données. Considérons les colonnes de symptômes dans le jeu de données `yaounde`, qui indiquent quels symptômes ont été ressentis par les répondants sur une période de 6 mois :

```
%>%
  starts_with("symp_"))
```

Nous aimerais utiliser cela pour évaluer si une personne a pu avoir la COVID-19, en suivant partiellement les directives recommandées par l'[OMS](#).

- Les individus avec toux doivent être classés comme “cas possibles de COVID-19”
- Les individus avec anosmie/agueusie (perte d'odeur ou de goût) doivent être classés comme “cas probables de COVID-19”.

Maintenant, en gardant ces critères à l'esprit, considérons une personne, appelons-la Osma, qui a à la fois de la toux ET de l'anosmie/agueusie ? Comment devrions-nous classer Osma ?

Elle remplit les critères pour être un “cas possible de COVID-19” (parce qu'elle a de la toux), mais elle remplit *aussi* les critères pour être un “cas probable de COVID-19” (parce qu'elle a de l'anosmie/agueusie). Alors, dans quel groupe devrait-elle être classée, “cas possible de COVID-19” ou “cas probable de COVID-19” ? Pensez-y pendant une minute.

Vous avez probablement deviné qu'elle devrait être classée comme un “cas probable de COVID-19”. “Probable” est plus probable que “Possible” ; et le symptôme de l'anosmie/agueusie est plus *significatif* que le symptôme de la toux. On pourrait dire

que le critère pour “cas probable de COVID-19” a une spécificité plus élevée ou une présence plus élevée que le critère pour “cas possible de COVID-19”.

Par conséquent, lors de la construction d'une déclaration `case_when()`, la condition “cas probable de COVID-19” devrait également prendre une présence plus élevée - elle devrait venir *en premier* dans les conditions fournies à `case_when()`. Voyons cela maintenant.

D'abord, nous sélectionnons les variables pertinentes, pour une illustration facile. Nous identifions également et `slice()` des lignes spécifiques qui sont utiles pour la démonstration :

```
_symptomes_slice <-
ie %>%
  (symp_toux, symp_anosmie_agueusie) %>%
  ce de lignes spécifiques utiles pour la démo
fois que vous trouvez le bon code, vous devriez supprimer ce slice
(32, 711, 625, 651)

_symptomes_slice
```

Maintenant, la déclaration `case_when()` correcte, qui a la condition “COVID-19 probable” en premier :

```
_symptomes_slice %>%
  (statut_covid = case_when(
  _anosmie_agueusie == "Yes" ~ "COVID-19 Probable",
  _toux == "Yes" ~ "COVID-19 Possible"
```

Cette déclaration `case_when()` peut être lue en termes simples comme ‘Si la personne a de l’anosmie/agueusie, inscrire “COVID-19 Probable”, sinon, si la personne a de la toux, inscrire “COVID-19 Possible”’.

Maintenant, passez du temps à examiner le jeu de données de sortie, en particulier les trois derniers individus. L’individu de la ligne 2 remplit le critère pour être un “cas possible de COVID-19” parce qu’il a de la toux (`symp_toux == "Yes"`), et l’individu de la ligne 3 remplit le critère pour être un “cas probable de COVID-19” parce qu’il a de l’anosmie/agueusie (`symp_anosmie_agueusie == "Yes"`).

L’individu de la ligne 4 est Osma, qui remplit à la fois les critères pour être un “cas possible de COVID-19” *et* pour un “cas probable de COVID-19”. Et parce que nous avons organisé nos conditions `case_when()` dans le bon ordre, elle est correctement codée comme “COVID-19 probable”. Super !

Mais remarquez ce qui se passe si nous échangeons l’ordre des conditions :

```

symptomes_slice %>%
  (statut_covid = case_when(
    toux == "Yes" ~ "COVID-19 Possible",
    anosmie_agueusie == "Yes" ~ "COVID-19 Probable"
  )

```

Oh non ! Osma à la ligne 4 est maintenant mal classée comme “COVID-19 Possible” alors qu’elle a le symptôme plus significatif d’anosmie/agueusie. C’est parce que la première condition `symp_toux == "Yes"` l’a correspondue en premier, et donc la deuxième condition n’a pas pu la correspondre !

Vous voyez maintenant pourquoi vous devez parfois réfléchir profondément à l’ordre de vos conditions `case_when()`. C’est un point mineur, mais il peut vous mordre à des moments inattendus. Même les analystes expérimentés ont tendance à faire des erreurs qui peuvent être attribuées à un mauvais arrangement des déclarations `case_when()`.

#### CHALLENGE



En réalité, il existe encore une autre solution pour éviter de mal classer la personne qui a de la toux et de l’anosmie/agueusie. C’est d’ajouter `symp_anosmie_agueusie != "Yes"` (n’est pas égal à “Yes”) aux conditions pour “COVID-19 Possible”. Pouvez-vous penser à pourquoi cela fonctionne ?

```

symptomes_slice %>%
  (statut_covid = case_when(
    toux == "Yes" & symp_anosmie_agueusie != "Yes" ~ "COVID-19
      Possible",
    anosmie_agueusie == "Yes" ~ "COVID-19 Probable"))

```

#### PRACTICE



Avec le jeu de données `liste_influ`, créez une nouvelle colonne appelée `priorite_de_suivi` qui implémente le schéma suivant :

- Les femmes doivent être considérées comme “Haute priorité”
- Tous les enfants (de moins de 18 ans) de n’importe quel sexe doivent être considérés comme “Priorité la plus élevée”.
- Tous les autres doivent avoir la valeur “Pas de priorité”

Étez le code avec votre réponse :

```

priorite_groupes <-
influ %>%
  (priorite_de_suivi = _____)

```

## Conditions binaires : `dplyr::if_else()`

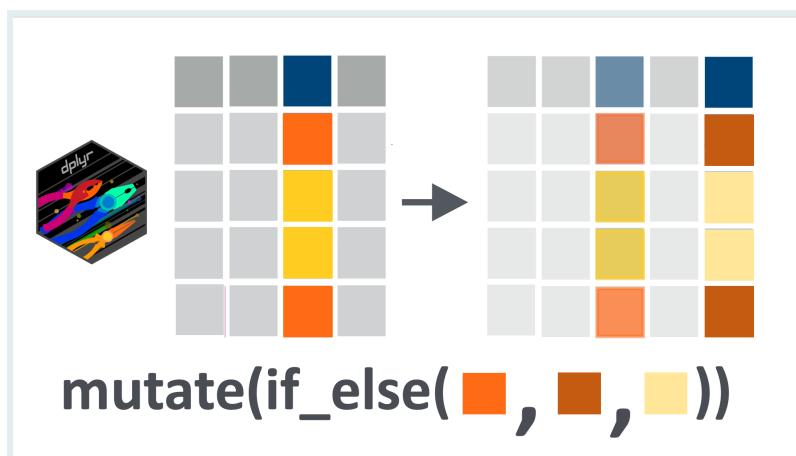


Fig: les conditions `if_else()`.

Il existe une autre fonction `{dplyr}` similaire à `case_when()` pour lorsque nous voulons appliquer une condition binaire à une variable : `if_else()`. Une condition binaire est soit `TRUE` soit `FALSE`. `if_else()` a une application similaire à `case_when()` : si la condition est vraie, une opération est appliquée, si la condition est fausse, l'alternative est appliquée. La syntaxe est : `if_else(CONDITION, SI_VRAI, SI_FAUX)`. Comme vous pouvez le voir, cela ne permet que d'appliquer une condition binaire (et non des cas multiples, comme avec `case_when()`).

Si nous prenons l'un des premiers exemples sur le recodage de la variable `highest_education`, nous pouvons l'écrire soit avec `case_when()` soit avec `if_else()`.

Voici la version que nous avons déjà explorée :

```
educ %>%
  `#(edu_haute =
    case_when(
      edu_haute %in% c("University", "Doctorate") ~ "Post-secondary",
      TRUE ~ edu_haute
    )`
```

Et voici comment nous l'écririons en utilisant `if_else()` :

```
educ %>%
  `#(edu_haute =
    if_else(
      edu_haute %in% c("University", "Doctorate"),
      # si TRUE alors on recodifie
      "Post-secondary",
      # si FALSE alors on garde la valeur par défaut
      edu_haute
    )`
```

Comme vous pouvez le voir, nous obtenons le même résultat, que nous utilisions `if_else()` ou `case_when()`.

### PRACTICE



(in RMD)

Avec les données `liste_influ`, créez une nouvelle colonne, appelée `age_groupe`, qui a la valeur “Moins de 50” pour les personnes de moins de 50 ans et “50 ans et plus” pour les personnes âgées de 50 ans et plus. Utilisez la fonction `if_else()`.

C'est exactement la même question que votre première question de pratique, mais cette fois vous devez utiliser `if_else()`.

Étez le code avec votre réponse :

```
coupe_if_else <-
  influ %>%
  mutate(age_groupe = if_else(____))
```

## Récapitulatif

Modifier ou construire vos variables en fonction de conditions sur d'autres variables est l'une des tâches de nettoyage de données les plus répétées. À tel point que cela méritait sa propre leçon !

J'espère maintenant que vous vous sentirez à l'aise pour utiliser `case_when()` et `if_else()` dans `mutate()` et que vous êtes enthousiaste à l'idée d'apprendre des opérations `{dplyr}` plus complexes comme le groupement de variables et leur résumé. À la prochaine !



Fig: Les conditions `if_else()` et `'case_when()'`.

---

## Contributeurs

L'équipe suivante a contribué à cette leçon :



### LAURE VANCAUWENBERGHE

Data analyst, the GRAPH Network

A firm believer in science for good, striving to ally programming, health and education

---



### KENE DAVID NWOSU

Data analyst, the GRAPH Network

Passionate about world improvement

---



### GUY WAFFEU

R Instructor and Public Health Physician

Committed to improving the quality of data analysis

---



### SABINA RODRIGUEZ VELÁSQUEZ

Project Manager and Scientific Collaborator, The GRAPH Network

Infectiously enthusiastic about microbes and Global Health

---

---

## Références

Certains matériaux de cette leçon ont été adaptés des sources suivantes :

- Horst, A. (2022). *Dplyr-learnr*. <https://github.com/allisonhorst/dplyr-learnr> (Œuvre originale publiée en 2020)
- *Créer, modifier, et supprimer des colonnes — Mutate*. (s.d.). Consulté le 21 février 2022, à partir de <https://dplyr.tidyverse.org/reference/mutate.html>

L'artwork a été adapté de :

- Horst, A. (2022). *Illustrations R & stats par Allison Horst*. <https://github.com/allisonhorst/stats-illustrations> (Œuvre originale publiée en 2018)

# Notes de leçon | Regroupement et résumé des données

February 2024



Introduction	.....
Objectifs d'apprentissage	.....
Le jeu de données COVID-19 de Yaoundé	.....
Qu'est-ce qu'une statistique récapitulative ?	.....
Introduction à <code>dplyr::summarize()</code>	.....
Résumés groupés avec <code>dplyr::group_by()</code>	.....
Regroupement par plusieurs variables (groupement imbriqué)	.....
Dégroupement avec <code>dplyr::ungroup()</code> (pourquoi et comment)	.....
Comptage des lignes	.....
Compter les lignes qui répondent à une condition	.....
<code>dplyr::count()</code>	.....
Inclure les combinaisons manquantes dans les statistiques récapitulatives	.....
Conclusion	.....

---

## Introduction

Vous savez déjà comment conserver les jeux données qui vous intéressent, comment garder les variables pertinentes et comment les modifier ou en créer de nouvelles.

Maintenant, nous allons aller un peu plus loin dans la manipulation de vos données en comprenant comment extraire facilement des statistiques récapitulatives, grâce au verbe `summarize()`, comme le calcul de la moyenne d'une variable.

De plus, nous commencerons à explorer un verbe crucial, `group_by()`, capable de regrouper vos variables ensemble pour effectuer des opérations groupées sur votre jeu de données.

Allons-y !

---

## Objectifs d'apprentissage

1. Vous pouvez utiliser `dplyr::summarize()` pour extraire des statistiques récapitulatives des jeux de données.
2. Vous pouvez utiliser `dplyr::group_by()` pour regrouper les données par une ou plusieurs variables avant d'effectuer des opérations sur elles.
3. Vous comprenez pourquoi et comment dégroupier les jeux de données groupés.
4. Vous pouvez utiliser `dplyr::n()` avec `group_by()-summarize()` pour compter les lignes par groupe.

5. Vous pouvez utiliser `sum()` avec `group_by()-summarize()` pour compter les lignes qui répondent à une condition.
  6. Vous pouvez utiliser `dplyr::count()` comme une fonction pratique pour compter les lignes par groupe.
- 

## Le jeu de données COVID-19 de Yaoundé

Dans cette leçon, nous allons à nouveau utiliser les données de l'enquête sérologique COVID-19 menée à Yaoundé, au Cameroun.

```
<- read_csv(here::here('data/fr_yaounde_data.csv'))  
  
#> # Source: CSV  
#> # Read with na越过  
#> # This关切  
#> # A tibble: 971 × 15  
#>   age cat_age_3 sexe   poids_kg  
#>   <dbl> <chr>   <chr>    <dbl>  
#> 1 45 Adult   Female  95  
#> 2 55 Adult   Male    96  
#> 3 23 Adult   Male    74  
#> 4 20 Adult   Female  70  
#> 5 55 Adult   Female  67  
#> 6 17 Child   Female  65  
#> 7 13 Child   Female  65  
#> 8 28 Adult   Male    62  
#> 9 30 Adult   Male    73  
#> 10 13 Child  Female  56  
#> # ... with 961 more rows, and 1 more variable:  
#> #   quartier <chr>  
#> #   fumeur   <chr>  
#> #   ...
```

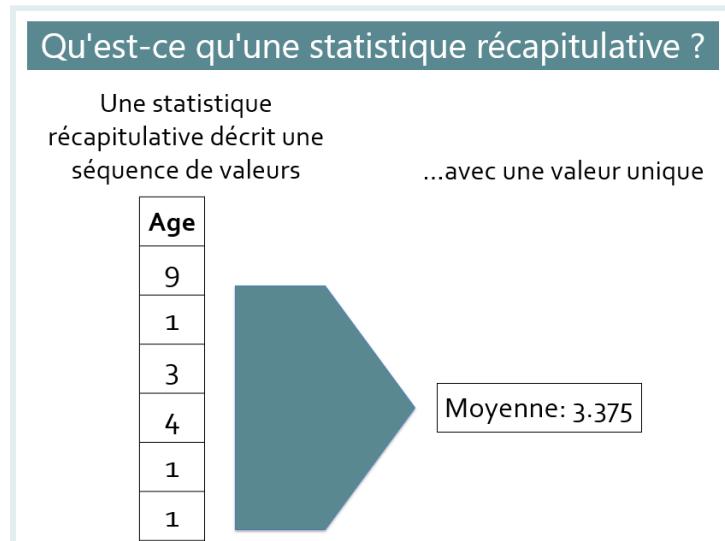
```
## # A tibble: 971 × 15  
##   age cat_age_3 sexe   poids_kg  
##   <dbl> <chr>   <chr>    <dbl>  
## 1 45 Adult   Female  95  
## 2 55 Adult   Male    96  
## 3 23 Adult   Male    74  
## 4 20 Adult   Female  70  
## 5 55 Adult   Female  67  
## 6 17 Child   Female  65  
## 7 13 Child   Female  65  
## 8 28 Adult   Male    62  
## 9 30 Adult   Male    73  
## 10 13 Child  Female  56  
## # ... with 961 more rows, and 1 more variable:  
## #   quartier <chr>  
## #   fumeur   <chr>  
## #   ...
```

```
## # i 961 more rows
## # i 8 more variables: enceinte <chr>, occupation <chr>, ...
```

Consultez la première leçon de ce chapitre pour plus d'informations sur ce jeu de données.

## Qu'est-ce qu'une statistique récapitulative ?

Une statistique récapitulative est une valeur unique (telle qu'une moyenne ou une médiane) qui décrit une séquence de valeurs (généralement une colonne dans votre jeu de données).



Les statistiques récapitulatives peuvent décrire le centre, la dispersion ou l'étendu d'une variable, ou les nombres et les positions des valeurs au sein de cette variable. Certaines statistiques récapitulatives courantes sont présentées dans le diagramme ci-dessous :

## Exemples de statistiques récapitulatives

```
age <- (9, 1, 4, 2, 2, 2)
```

Statistique récapitulative	Code R	Résultat
<b>Nombres</b>		
No. d'éléments	dplyr::n(age)	6
No. d'éléments distincts	dplyr::n_distinct(age)	4
<b>Position</b>		
Premier élément	dplyr::first(age)	9
Dernier élément	dplyr::last(age)	2
3 <sup>ème</sup> élément	dplyr::nth(age, 3)	4
<b>Centre</b>		
Moyenne	mean(age)	3.3
Médiane	median(age)	2
<b>Dispersion</b>		
Ecart-type	sd(age)	2.9
Ecart interquartile	IQR(age)	1.5
<b>Etendu</b>		
Minimum	min(age)	1
Maximum	max(age)	9
25 <sup>ème</sup> quantile	quantile(age, 0.25)	2

Le calcul des statistiques récapitulatives est une opération très courante dans la plupart des processus d'analyse de données, il sera donc important de devenir compétent pour les extraire de vos jeux de données. Et pour cette tâche, il n'y a pas de meilleur outil que la fonction `summarize()` de `{dplyr}` ! Alors voyons comment utiliser cette puissante fonction.

### Introduction à `dplyr::summarize()`

Pour commencer, il est préférable de voir d'abord comment obtenir des statistiques récapitulatives simples *sans utiliser* `summarize()`, puis nous verrons pourquoi vous devriez *réellement utiliser* `summarize()`.

Imaginez que l'on vous demande de trouver l'âge moyen des répondants dans le jeu de données `yao`. Comment pourriez-vous le faire en R de base ?

Tout d'abord, rappelons que la fonction du signe dollar, `$`, vous permet d'extraire une colonne d'un jeu de données vers un vecteur :

```
# extraire la colonne `age` de `yao`
```

Pour obtenir la moyenne, vous passez simplement ce vecteur `yao$age` dans la fonction `mean()` :

```
age)
```

```
## [1] 29.01751
```

Et c'est tout ! Vous avez maintenant une statistique récapitulative simple. Extrêmement facile, n'est-ce pas ?

Alors, pourquoi avons-nous besoin de `summarize()` pour obtenir des statistiques récapitulatives si le processus est déjà si simple sans lui ? Nous reviendrons sur la question du *pourquoi* bientôt. D'abord, voyons *comment* obtenir des statistiques récapitulatives avec `summarize()`.

En revenant à l'exemple précédent, la syntaxe correcte pour obtenir l'âge moyen avec `summarize()` serait :

```
size(mean_age = mean(age))
```

```
## # A tibble: 1 × 1
##   mean_age
##       <dbl>
## 1     29.0
```

L'anatomie de cette syntaxe est présentée ci-dessous. Vous devez simplement entrer le nom de la nouvelle colonne (par exemple `mean_age`), la fonction récapitulative (par exemple `mean()`), et la colonne à résumer (par exemple `age`).

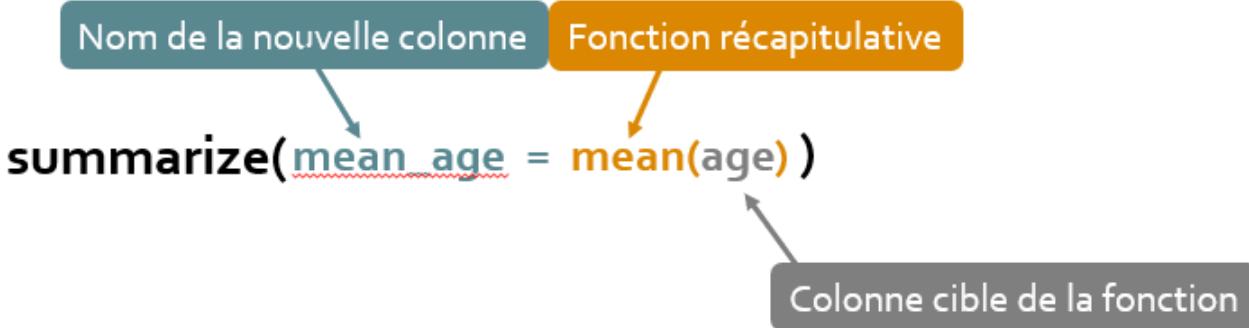


Fig. Syntaxe de base pour la fonction `summarize()`.

Vous pouvez également calculer plusieurs statistiques récapitulatives dans une seule commande `summarize()`. Par exemple, si vous voulez à la fois l'âge moyen et l'âge médian, vous pourriez exécuter :

```
size(mean_age = mean(age),
      median_age = median(age))
```

```
## # A tibble: 1 × 2
##   mean_age median_age
##       <dbl>      <dbl>
## 1     29.0       26
```

Sympa !

Maintenant, vous devriez vous demander pourquoi `summarize()` place les statistiques récapitulatives dans un jeu de données, avec chaque statistique dans une colonne différente.

Le principal avantage de cette structure de jeu de données est de faciliter la production de résumés *groupés* (et la création de tels résumés groupés sera le principal avantage de l'utilisation de `summarize()`).

Nous examinerons ces résumés groupés dans la section suivante. Pour l'instant, essayez de répondre aux questions de pratique ci-dessous.

Utilisez `summarize()` et les fonctions récapitulatives pertinentes pour obtenir la moyenne, la médiane et l'écart type des poids des répondants à partir de la variable `poids_kg` du jeu de données `yao`.

Votre sortie doit être un jeu de données avec trois colonnes nommées comme indiqué ci-dessous :

**mean\_poids\_kg median\_poids\_kg sd\_poids\_kg**

```
_poids <-  
% "ÉCRIVEZ_VOTRE_RÉPONSE_ICI"
```

Utilisez `summarize()` et les fonctions récapitulatives pertinentes pour obtenir les tailles minimale et maximale des répondants à partir de la variable `taille_cm` du jeu de données `yao`.



Votre sortie doit être un jeu de données avec deux colonnes nommées comme indiqué ci-dessous :

**min\_taille\_cm max\_taille\_cm**

```
_taille <-  
%  
EZ_VOTRE_RÉPONSE_ICI"
```

---

## Résumés groupés avec `dplyr::group_by()`

Comme son nom l'indique, `dplyr::group_by()` vous permet de regrouper un jeu de données par les valeurs d'une variable (par exemple le sexe masculin vs féminin). Vous pouvez ensuite effectuer des opérations qui sont divisées selon ces groupes.

Quel effet `group_by()` a-t-il sur un jeu de données ? Essayons de regrouper le jeu de données `yao` par sexe et observons l'effet :

```
_by(sexe)
```

```
## # A tibble: 971 × 15
## # Groups: sexe [2]
##       age cat_age_3 sexe   poids_kg
##     <dbl> <chr>    <chr>    <dbl>
## 1     45 Adult    Female    95
## 2     55 Adult    Male     96
## 3     23 Adult    Male     74
## 4     20 Adult    Female    70
## 5     55 Adult    Female    67
## 6     17 Child    Female    65
## 7     13 Child    Female    65
## 8     28 Adult    Male     62
## 9     30 Adult    Male     73
## 10    13 Child    Female    56
## # ... with 961 more rows
## # ... with 8 more variables: enceinte <chr>, occupation <chr>, ...
```

Hmm. Apparemment, rien ne s'est passé. La seule chose que vous pourriez remarquer est une nouvelle section dans l'en-tête qui vous indique la variable groupée—`sex`—et le nombre de groupes—2 :

```
# A tibble: 971 × 10
👉 # Groups: sexe [2] 👈
```

Mis à part cet en-tête, cependant, le jeu de données semble inchangé.

Mais voyez ce qui se passe lorsque nous chaînons le `group_by()` avec l'appel `summarize()` que nous avons utilisé dans la section précédente :

```
by(sexe) %>%  
  summarize(mean_age = mean(age))
```

```
## # A tibble: 2 × 2  
##   sexe     mean_age  
##   <chr>      <dbl>  
## 1 Female     29.5  
## 2 Male       28.4
```

Vous obtenez une statistique récapitulative différente pour chaque groupe ! Les statistiques pour les femmes sont dans une ligne et celles pour les hommes sont dans une autre. (À partir de ce jeu de données de sortie, vous pouvez dire par exemple que, l'âge moyen pour les répondantes est de 29.5, tandis que pour les répondants masculins, il est de 28.4)

Comme mentionné précédemment, ce type de résumé groupé est la raison principale pour laquelle la fonction `summarize()` est si utile !

---

Voyons un autre exemple d'une opération simple `group_by() + summarize()`.

Supposons que l'on vous ait demandé d'obtenir les poids maximum et minimum pour les individus dans différents quartiers dans le jeu de données `yao`. D'abord, vous feriez un `group_by()` sur la variable `neighbourhood`, puis vous appeleriez les fonctions `max()` et `min()` à l'intérieur de `summarize()` :

```
by(quartier) %>%  
  summarize(max_poids = max(poids_kg),  
           min_poids = min(poids_kg))
```

```
## # A tibble: 9 × 3  
##   quartier     max_poids min_poids  
##   <chr>        <dbl>     <dbl>  
## 1 Briqueterie    128      20  
## 2 Carriere       129      14  
## 3 Cité Verte     118      16  
## 4 Ekoudou        135      15  
## 5 Messa           96       19  
## 6 Mokolo         162      16  
## 7 Nkomkana       161      15  
## 8 Tsinga          105      15  
## 9 Tsinga Oliga    100      17
```

Super ! Avec seulement quelques lignes de code, vous êtes capable d'extraire beaucoup d'informations.

---

Voyons encore un exemple pour faire bonne mesure. La variable `jours_absence_travail` nous indique le nombre de jours où les répondants ont été absents au travail en raison de symptômes similaires à ceux du COVID. Les individus qui n'ont signalé aucun symptôme semblable à celui du COVID ont un `NA` pour cette variable :

```
: (jours_absence_travail)
```

```
## # A tibble: 971 × 1
##   jours_absence_travail
##       <dbl>
## 1 0
## 2 NA
## 3 NA
## 4 7
## 5 NA
## 6 7
## 7 0
## 8 0
## 9 0
## 10 NA
## # i 961 more rows
```

Pour compter le nombre total de jours de travail manqués pour chaque groupe de sexe, vous pourriez essayer d'exécuter la fonction `sum()` sur la variable `jours_absence_travail` :

```
by(sexe) %>%
  use(total_jours_absence = sum(jours_absence_travail))
```

```
## # A tibble: 2 × 2
##   sexe   total_jours_absence
##   <chr>      <dbl>
## 1 Female        NA
## 2 Male          NA
```

Hmmm. Cela vous donne des résultats `NA` car certaines lignes dans la colonne `jours_absence_travail` ont des `NA` en elles, et R ne peut pas trouver la somme de valeurs contenant un `NA`. Pour résoudre ce problème, l'argument `na.rm = TRUE` est nécessaire :

```
by(sexe) %>%  
  use(total_jours_absence = sum(jours_absence_travail, na.rm = TRUE))
```

```
## # A tibble: 2 × 2  
##   sexe   total_jours_absence  
##   <chr>          <dbl>  
## 1 Female           256  
## 2 Male             272
```

La sortie nous dit qu'au total, parmi toutes les femmes de l'échantillon, 256 jours de travail ont été manqués en raison de symptômes similaires à ceux du COVID, et parmi tous les hommes, 272 jours.

J'espère que vous voyez maintenant pourquoi `summarize()` est si puissant. En combinaison avec `group_by()`, il vous permet d'obtenir des résumés de vos jeux de données groupés très informatifs avec très peu de lignes de code.

Produire de tels résumés est une partie très importante de la plupart des processus d'analyse de données, cette compétence sera donc probablement utile très prochainement !

### VOCAB



#### `summarize()` produit des “Tableaux croisés dynamiques”

Les jeux de données récapitulatifs créés par `summarize()` sont souvent appelés des tableaux croisés dynamiques dans le contexte des logiciels de tableur comme Microsoft Excel.

### PRACTICE (in RMD)

Utilisez `group_by()` et `summarize()` pour obtenir le poids moyen (kg) en fonction du statut de fumeur dans le jeu de données `yao`. Nommez la colonne de poids moyen `poids_moyen`

Le jeu de données de sortie doit ressembler à ceci :

fumeur	poids_moyen
Ex-fumeur	
Non-fumeur	
Fumeur	
NA	

**PRACTICE**  
  
(in RMD)

```
selon_statut_fumeur <-
>%
EZ_VOTRE_RÉPONSE_ICI"
```

Utilisez `group_by()`, `summarize()` et les fonctions de statistiques récapitulatives pertinentes pour obtenir les tailles minimum et maximum pour chaque sexe dans le jeu de données `yao`.

**PRACTICE**  
  
(in RMD)

Votre sortie doit être un jeu de données avec trois colonnes nommées comme indiqué ci-dessous :

sexe	taille_min_cm	taille_max_cm
Female		
Male		

```
ax_taille_selon_sexe <-
>%
EZ_VOTRE_RÉPONSE_ICI"
```

Utilisez `group_by()`, `summarize()`, et la fonction `sum()` pour calculer le nombre total de jours alités (de la variable `jours_alite`) rapportés par les répondants de chaque sexe.

**PRACTICE**  
  
(in RMD)

Votre sortie doit être un jeu de données avec deux colonnes nommées comme indiqué ci-dessous :

sexe	total_jours_alite
Female	
Male	

```
jours_alite <-
>%
EZ_VOTRE_RÉPONSE_ICI"
```

## Regroupement par plusieurs variables (groupement imbriqué)

Il est possible de regrouper un jeu de données par plus d'une variable. Ceci est parfois appelé "groupement imbriqué".

Prenons un exemple. Supposons que vous voulez connaître l'âge moyen des hommes et des femmes *dans chaque quartier* (plutôt que l'âge moyen de *toutes les femmes*), vous pourriez mettre à la fois sexe et quartier dans l'instruction group\_by() :

```
by(sexe, quartier) %>%  
  size(age_moyen = mean(age))
```

```
## `summarise()` has grouped output by  
## 'sexe'. You can override using the  
## `groups` argument.
```

```
## # A tibble: 18 × 3  
## # Groups: sexe [2]  
##   sexe   quartier     age_moyen  
##   <chr>  <chr>        <dbl>  
## 1 Female Briqueterie    31.6  
## 2 Female Carriere      28.2  
## 3 Female Cité Verte    31.8  
## 4 Female Ekoudou       29.3  
## 5 Female Messa         30.2  
## 6 Female Mokolo        28.0  
## 7 Female Nkomkana      33.0  
## 8 Female Tsinga        30.6  
## 9 Female Tsinga Oliga  24.3  
## 10 Male   Briqueterie   33.7  
## 11 Male   Carriere      30.0  
## 12 Male   Cité Verte    27.0  
## 13 Male   Ekoudou       25.2  
## 14 Male   Messa          23.9  
## 15 Male   Mokolo         30.5  
## 16 Male   Nkomkana      29.8  
## 17 Male   Tsinga         28.8  
## 18 Male   Tsinga Oliga  24.3
```

À partir de ce jeu de données de sortie, vous pouvez voir que, par exemple, les femmes de Briqueterie ont un âge moyen de 31,6 ans, tandis que les hommes de Briqueterie ont un âge moyen de 33,7 ans.

L'ordre des colonnes listées dans group\_by() est interchangeable. Donc, si vous exécutez group\_by(quartier, sexe) au lieu de group\_by(sexe, quartier), vous obtiendrez le même résultat, bien qu'il soit ordonné différemment :

```
by(quartier, sexe) %>%  
  size(age_moyen = mean(age))
```

```

## `summarise()` has grouped output by
## 'quartier'. You can override using
## the ` `.groups` argument.

## # A tibble: 18 × 3
## # Groups: quartier [9]
##   quartier sexe   age_moyen
##   <chr>    <chr>     <dbl>
## 1 Briqueterie Female   31.6
## 2 Briqueterie Male    33.7
## 3 Carriere   Female   28.2
## 4 Carriere   Male    30.0
## 5 Cité Verte Female   31.8
## 6 Cité Verte Male    27.0
## 7 Ekoudou    Female   29.3
## 8 Ekoudou    Male    25.2
## 9 Messa      Female   30.2
## 10 Messa     Male    23.9
## 11 Mokolo    Female   28.0
## 12 Mokolo    Male    30.5
## 13 Nkomkana  Female   33.0
## 14 Nkomkana  Male    29.8
## 15 Tsinga    Female   30.6
## 16 Tsinga    Male    28.8
## 17 Tsinga Oliga Female  24.3
## 18 Tsinga Oliga Male   24.3

```

Maintenant, l'ordre des colonnes est différent : `quartier` est la première colonne, et `sexe` est la deuxième. Et l'ordre des lignes est également différent : les lignes sont d'abord ordonnées par `quartier`, puis ordonnées par `sexe` à l'intérieur de chaque quartier.

Mais les statistiques de résumé sont les mêmes. Par exemple, vous pouvez à nouveau voir que les femmes de Briqueterie ont un âge moyen de 31,6 ans, tandis que les hommes de Briqueterie ont un âge moyen de 33,7 ans.



En utilisant le jeu de données `yao`, groupez vos données par sexe (`sexe`) et traitements (`combinaisons_traitement`) en utilisant `group_by`. Ensuite, en utilisant `summarize()` et la fonction de statistique récapitulative appropriée, calculez le poids moyen (`poids_kg`) pour chaque groupe.

Votre sortie doit être un jeu de données avec trois colonnes nommées comme indiqué ci-dessous :

<code>sexe</code>	<code>combinaisons_traitement</code>	<code>poids_moyen_kg</code>
-------------------	--------------------------------------	-----------------------------



```
selon_sexe_traitement <-
>%>
EZ_VOTRE_RÉPONSEICI"
```



En utilisant le jeu de données `yao`, groupez vos données par catégorie d'âge (`cat_age_3`), genre (`sexe`), et résultats d'IgG (`resultat_igg`) en utilisant `group_by`. Ensuite, en utilisant `summarize()` et la fonction de statistique récapitulative appropriée, calculez le nombre moyen de jours alités (`moyenne_jours_alite`) pour chaque groupe.

Votre sortie doit être un jeu de données avec quatre colonnes nommées comme indiqué ci-dessous :

`cat_age_3 sexe resultat_igg moyenne_jours_alite`

```
alite_age_sex_igg <-
>%>
EZ_VOTRE_RÉPONSEICI"
```

## Dégroupement avec `dplyr::ungroup()` (pourquoi et comment)

Quand vous utilisez `group_by()` pour plus d'une variable avant d'utiliser `summarize()`, le jeu de données de sortie reste groupé. Ce regroupement persistant peut avoir des effets indésirables en aval, vous devrez donc parfois utiliser `dplyr::ungroup()` pour dégroupier les données avant de faire une analyse plus poussée.

Pour comprendre *pourquoi* vous devriez utiliser `ungroup()` sur les données, considérez d'abord l'exemple suivant, où nous ne regroupons qu'une seule variable avant de calculer une statistique récapitulative :

```
by(sexe) %>%
  size(mean_age = mean(age))
```

```
## # A tibble: 2 × 2
##   sexe    mean_age
##   <chr>     <dbl>
```

```
## 1 Female      29.5
## 2 Male        28.4
```

Les données sont produites comme un jeu de données normal ; il n'est pas groupé. Vous pouvez le voir parce qu'il n'y a pas d'information sur les groupes dans l'en-tête.

Mais considérez maintenant quand vous regroupez par deux variables avant de calculer une statistique récapitulative :

```
by(sexe, quartier) %>%
  summarize(mean_age = mean(age))
```

```
## `summarise()` has grouped output by
## 'sexe'. You can override using the
## `.` argument.
```

```
## # A tibble: 18 × 3
## # Groups:   sexe [2]
##       sexe   quartier   mean_age
##       <chr>   <chr>     <dbl>
## 1 Female Briqueterie    31.6
## 2 Female Carriere      28.2
## 3 Female Cité Verte    31.8
## 4 Female Ekoudou       29.3
## 5 Female Messa         30.2
## 6 Female Mokolo        28.0
## 7 Female Nkomkana      33.0
## 8 Female Tsinga        30.6
## 9 Female Tsinga Oliga  24.3
## 10 Male   Briqueterie   33.7
## 11 Male   Carriere      30.0
## 12 Male   Cité Verte    27.0
## 13 Male   Ekoudou       25.2
## 14 Male   Messa          23.9
## 15 Male   Mokolo         30.5
## 16 Male   Nkomkana      29.8
## 17 Male   Tsinga         28.8
## 18 Male   Tsinga Oliga  24.3
```

Maintenant, l'en-tête vous indique que les données sont toujours groupées par la première variable dans `group_by()`, ici `sexe` :

```
# A tibble: 18 × 3
👉# Groups:   sexe [2] 👈
```

Quelle est l'implication de ce regroupement persistant dans le jeu de données ? Cela signifie que le jeu de données peut montrer un comportement qui semble étrange

lorsque vous essayez d'appliquer certaines fonctions {dplyr} dessus.

Par exemple, si vous essayez de `select()` une seule variable, peut-être la variable `mean_age`, vous devriez normalement pouvoir utiliser `select(mean_age)` :

```
by(sexe, quartier) %>%  
  size(mean_age = mean(age)) %>%  
  (mean_age) # ne fonctionne pas comme prévu
```

```
## `summarise()` has grouped output by  
## 'sexe'. You can override using the  
## `groups` argument.  
## Adding missing grouping variables:  
## `sexe`  
  
## # A tibble: 18 × 2  
## Groups: sexe [2]  
##   sexe     mean_age  
##   <chr>     <dbl>  
## 1 Female    31.6  
## 2 Female    28.2  
## 3 Female    31.8  
## 4 Female    29.3  
## 5 Female    30.2  
## 6 Female    28.0  
## 7 Female    33.0  
## 8 Female    30.6  
## 9 Female    24.3  
## 10 Male     33.7  
## 11 Male     30.0  
## 12 Male     27.0  
## 13 Male     25.2  
## 14 Male     23.9  
## 15 Male     30.5  
## 16 Male     29.8  
## 17 Male     28.8  
## 18 Male     24.3
```

Mais comme vous pouvez le voir, la variable groupée, `sex`, est toujours sélectionnée, même si nous n'avons demandé que `mean_age` dans l'instruction `select()`.

C'est l'un des nombreux exemples de comportements uniques des jeux de données groupés. D'autres verbes dplyr comme `filter()`, `mutate()` et `arrange()` agissent également de manière spéciale sur les données groupées. Nous aborderons cela en détail dans une leçon future.

Vous savez donc maintenant *pourquoi* vous devriez dégrouper les données lorsque vous n'en avez plus besoin. Voyons maintenant *comment* dégrouper les données.

C'est assez simple : il suffit d'ajouter la fonction `ungroup()` à votre chaîne de pipe. Par exemple :

```
by(sexe, quartier) %>%  
  size(mean_age = mean(age)) %>%  
  ungroup()
```

```
## `summarise()` has grouped output by  
## 'sexe'. You can override using the  
## `.`groups` argument.
```

```
## # A tibble: 18 × 3  
##   sexe   quartier     mean_age  
##   <chr>  <chr>        <dbl>  
## 1 Female Briqueterie    31.6  
## 2 Female Carriere      28.2  
## 3 Female Cité Verte    31.8  
## 4 Female Ekoudou       29.3  
## 5 Female Messa         30.2  
## 6 Female Mokolo        28.0  
## 7 Female Nkomkana      33.0  
## 8 Female Tsinga        30.6  
## 9 Female Tsinga Oliga  24.3  
## 10 Male   Briqueterie   33.7  
## 11 Male   Carriere     30.0  
## 12 Male   Cité Verte    27.0  
## 13 Male   Ekoudou       25.2  
## 14 Male   Messa         23.9  
## 15 Male   Mokolo        30.5  
## 16 Male   Nkomkana      29.8  
## 17 Male   Tsinga        28.8  
## 18 Male   Tsinga Oliga  24.3
```

Maintenant que le jeu de données est dégroupé, il se comportera à nouveau comme un jeu de données normal. Par exemple, vous pouvez `select()` n'importe quelle colonne(s) que vous voulez ; vous n'aurez pas certaines colonnes indésirables qui vous suivent :

```
by(sexe, quartier) %>%  
  size(mean_age = mean(age)) %>%  
  ungroup() %>%  
  select(-mean_age)
```

```
## `summarise()` has grouped output by  
## 'sexe'. You can override using the  
## `.`groups` argument.
```

```
## # A tibble: 18 × 1
##   mean_age
##   <dbl>
## 1 31.6
## 2 28.2
## 3 31.8
## 4 29.3
## 5 30.2
## 6 28.0
## 7 33.0
## 8 30.6
## 9 24.3
## 10 33.7
## 11 30.0
## 12 27.0
## 13 25.2
## 14 23.9
## 15 30.5
## 16 29.8
## 17 28.8
## 18 24.3
```

---

## Comptage des lignes

Vous pouvez faire beaucoup de science des données en *comptant* simplement et occasionnellement en *divisant*. - Hadley Wickham, Scientifique Senior chez RStudio

Une tâche courante de statistique récapitulative des données est de compter combien d'observations (lignes) il y a pour chaque groupe. Vous pouvez y parvenir avec la fonction spéciale `n()` de `{dplyr}`, qui est spécifiquement conçue pour être utilisée dans `summarise()`.

Par exemple, si vous voulez compter combien d'individus se trouvent dans chaque groupe de quartier, vous exécuteriez :

```
by(quartier) %>%
  size(nombre = n())
```

```
## # A tibble: 9 × 2
##   quartier     nombre
##   <chr>       <int>
## 1 Briqueterie    106
## 2 Carriere        236
## 3 Cité Verte      72
## 4 Ekoudou        190
```

```

## 5 Messa          48
## 6 Mokolo        96
## 7 Nkomkana      75
## 8 Tsinga         81
## 9 Tsinga Oliga   67

```

Comme vous pouvez le voir, la fonction `n()` ne nécessite aucun argument. Elle “connait son travail” dans le jeu de données !

Bien sûr, vous pouvez inclure d’autres statistiques récapitulatives dans le même appel `summarize()`. Par exemple, ci-dessous, nous calculons également l’âge moyen par quartier.

```

by(quartier) %>%
  size(nombre = n(),
       mean_age = mean(age))

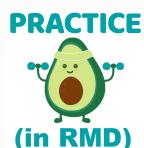
```

```

## # A tibble: 9 × 3
##   quartier     nombre  mean_age
##   <chr>       <int>    <dbl>
## 1 Briqueterie 106     32.5
## 2 Carriere     236     28.9
## 3 Cité Verte   72      29.9
## 4 Ekoudou     190     27.6
## 5 Messa        48      27.3
## 6 Mokolo       96      29.1
## 7 Nkomkana     75      31.7
## 8 Tsinga       81      29.7
## 9 Tsinga Oliga 67      24.3

```

Groupez votre jeu de données `yao` par l’occupation des répondants (`occupation`) et utilisez `summarize()` pour créer des colonnes qui montrent :



- combien d’individus il y a avec chaque occupation (pensez à la fonction `n()`)
- le nombre moyen de jours de travail manqués (`moyenne_age`) par ceux ayant cette occupation

Votre sortie doit être un jeu de données avec trois colonnes nommées comme indiqué ci-dessous :

occupation	nombre	moyenne_age
------------	--------	-------------



```
#_occupation <-
>%
'EZ_VOTRE_RÉPONSEICI"
```

## Compter les lignes qui répondent à une condition

Plutôt que de compter *toutes* les lignes comme ci-dessus, il est parfois plus utile de compter seulement les lignes qui répondent à des conditions spécifiques. Cela peut être fait facilement en plaçant les conditions requises dans la fonction `sum()`.

Par exemple, pour compter le nombre de personnes de moins de 18 ans dans chaque quartier, vous placez la condition `age < 18` à l'intérieur de `sum()` :

```
by(quartier) %>%
size(nombre_inferieur_18 = sum(age < 18))
```

```
## # A tibble: 9 × 2
##   quartier      nombre_inferieur_18
##   <chr>                 <int>
## 1 Briqueterie            28
## 2 Carriere                58
## 3 Cité Verte              19
## 4 Ekoudou                  66
## 5 Messa                      18
## 6 Mokolo                    32
## 7 Nkomkana                  22
## 8 Tsinga                     23
## 9 Tsinga Oliga              25
```

---

De même, pour compter le nombre de personnes ayant un doctorat dans chaque quartier, vous placez la condition `edu_haute == "Doctorate"` à l'intérieur de `sum()` :

```
by(quartier) %>%
size(nombre_avec_doctorates = sum(edu_haute == "Doctorate"))
```

```
## # A tibble: 9 × 2
##   quartier      nombre_avec_doctorates
##   <chr>                 <int>
## 1 Briqueterie                   2
## 2 Carriere                      1
## 3 Cité Verte                     1
## 4 Ekoudou                      1
## 5 Messa                         2
## 6 Mokolo                        0
## 7 Nkomkana                     4
```

```
## 8 Tsinga          3  
## 9 Tsinga Oliga    3
```

## Sous le capot : compter avec des conditions

Pourquoi pouvez-vous utiliser `sum()`, qui est censé ajouter des nombres, sur une condition comme `edu_haute == "Doctorate"` ?

Utiliser `sum()` sur une condition fonctionne parce que la condition évalue aux valeurs booléennes `TRUE` et `FALSE`. Et ces valeurs booléennes sont traitées comme des nombres (où `TRUE` est égal à 1 et `FALSE` est égal à 0), et les nombres peuvent, bien sûr, être sommés.

Le code ci-dessous démontre ce qui se passe sous le capot de manière étape par étape. Exécutez-le et voyez si vous pouvez suivre.

```
condition_sums <- yao %>%  
  (edu_haute) %>%  
  (avec_doctorate = edu_haute == "Doctorate") %>%  
  (numerique_avec_doctorate = as.numeric(avec_doctorate))  
  
condition_sums
```

CHALLENGE



```
## # A tibble: 971 × 3  
##   edu_haute   avec_doctorate  
##   <chr>        <lgl>  
##   1 Secondary  FALSE  
##   2 University FALSE  
##   3 University FALSE  
##   4 Secondary  FALSE  
##   5 Primary    FALSE  
##   6 Secondary  FALSE  
##   7 Secondary  FALSE  
##   8 Doctorate  TRUE  
##   9 Secondary  FALSE  
##  10 Secondary  FALSE  
##   numerique_avec_doctorate  
##                           <dbl>  
##   1                      0  
##   2                      0  
##   3                      0  
##   4                      0  
##   5                      0  
##   6                      0  
##   7                      0  
##   8                      1  
##   9                      0
```

```
## 10
## # i 961 more rows
0
```



Les valeurs numériques peuvent ensuite être ajoutées pour produire un décompte des lignes remplissant la condition `edu_haute == "Doctorate"` :

```
_condition_sums %>%
  size(nombre_avec_doctorate = sum(numerique_avec_doctorate))
```

```
## # A tibble: 1 × 1
##   nombre_avec_doctorate
##   <dbl>
## 1 17
```

Pour une illustration finale du comptage avec des conditions, considérez la variable `combinaisons_traitement`, qui liste les traitements reçus par les personnes présentant des symptômes similaires à ceux du COVID. Les personnes qui n'ont reçu aucun traitement ont une valeur `NA` :

```
(combinaisons_traitement)
```

```
## # A tibble: 971 × 1
##   combinaisons_traitement
##   <chr>
## 1 Paracetamol
## 2 <NA>
## 3 <NA>
## 4 Antibiotics
## 5 <NA>
## 6 Paracetamol--Antibiotics
## 7 Traditional meds.
## 8 Paracetamol
## 9 Paracetamol--Traditional meds.
## 10 <NA>
## # i 961 more rows
```

Si vous voulez compter le nombre de personnes qui n'ont reçu *aucun traitement*, vous additionneriez celles qui répondent à la condition `is.na(combinaisons_traitement)` :

```
_by(quartier) %>%
  size(traitement_inconnu = sum(is.na(combinaisons_traitement)))
```

```
## # A tibble: 9 × 2
##   quartier      traitement_inconnu
##   <chr>              <int>
## 1 Briqueterie          82
## 2 Carriere             192
## 3 Cité Verte            46
## 4 Ekoudou              133
## 5 Messa                 35
## 6 Mokolo                65
## 7 Nkomkana              53
## 8 Tsinga                56
## 9 Tsinga Oliga           47
```

Ce sont les personnes ayant des valeurs NA pour la colonne combinaisons\_traitement.

Pour compter les personnes qui *ont* reçu un traitement, vous pouvez simplement nier la fonction `is.na()` avec ! :

```
group_by(quartier) %>%
  size(traitement_connu = sum(!is.na(combinaisons_traitement)))
```

```
## # A tibble: 9 × 2
##   quartier      traitement_connu
##   <chr>              <int>
## 1 Briqueterie          24
## 2 Carriere             44
## 3 Cité Verte            26
## 4 Ekoudou              57
## 5 Messa                 13
## 6 Mokolo                31
## 7 Nkomkana              22
## 8 Tsinga                25
## 9 Tsinga Oliga           20
```

## dplyr::count()

La fonction `dplyr::count()` regroupe plusieurs choses en une seule ligne de code conviviale pour vous aider à trouver les comptages d'observations par groupe.

Utilisons `dplyr::count()` sur notre variable occupation :

```
(occupation)
```

```

## # A tibble: 28 × 2
##   occupation     n
##   <chr>       <int>
## 1 Farmer           5
## 2 Farmer--Other      1
## 3 Home-maker        65
## 4 Home-maker--Farmer    2
## 5 Home-maker--Informal worker 3
## 6 Home-maker--Informal worker--Farmer 1
## 7 Home-maker--Trader    3
## 8 Informal worker      189
## 9 Informal worker--Other 2
## 10 Informal worker--Trader 4
## # i 18 more rows

```

Notez que c'est la même sortie que :

```

by(occupation) %>%
size(n = n())

```

```

## # A tibble: 28 × 2
##   occupation     n
##   <chr>       <int>
## 1 Farmer           5
## 2 Farmer--Other      1
## 3 Home-maker        65
## 4 Home-maker--Farmer    2
## 5 Home-maker--Informal worker 3
## 6 Home-maker--Informal worker--Farmer 1
## 7 Home-maker--Trader    3
## 8 Informal worker      189
## 9 Informal worker--Other 2
## 10 Informal worker--Trader 4
## # i 18 more rows

```

```
## 5      3
## 6      1
## 7      3
## 8    189
## 9      2
## 10     4
## # i 18 more rows
```

Vous pouvez également appliquer `dplyr::count()` de manière imbriquée :

```
(sexe, occupation)
```

```
## # A tibble: 40 × 3
##       sexe   occupation     n
##       <chr>   <chr>     <int>
## 1 Female  Farmer            3
## 2 Female  Home-maker        65
## 3 Female  Home-maker--Farmer  2
## 4 Female  Home-maker--Informal worker  3
## 5 Female  Home-maker--Informal worker--... 1
## 6 Female  Home-maker--Trader    3
## 7 Female  Informal worker     77
## 8 Female  Informal worker--Trader  1
## 9 Female  No response        8
## 10 Female Other              6
## # i 30 more rows
```



Le verbe `count()` vous donne des informations clés sur votre ensemble de données de manière très rapide. Regardons nos résultats IgG stratifiés par catégorie d'âge et par sexe en une seule ligne de code.

En utilisant le jeu de données `yao`, comptez les différentes combinaisons de genre (`sex`), de catégories d'âge (`cat_age_3`) et de résultats IgG (`resultat_igg`).

**PRACTICE**

(in RMD)

Votre sortie doit être un jeu de données avec quatre colonnes nommées comme indiqué ci-dessous :

sex sex cat\_age\_3 resultat\_igg n

```
_resultats <-  
>%  
EZ_VOTRE_RÉPONSE_ICI"
```

**PRACTICE**

(in RMD)

En utilisant le jeu de données `yao`, comptez les différentes combinaisons de catégories d'âge (`cat_age_3`) et de nombre de jours alités (`total_jours_alite`).

Votre sortie doit être un jeu de données avec trois colonnes nommées comme indiqué ci-dessous :

cat\_age\_3 total\_jours\_alite n

```
_jours_alite_age <-  
>%  
EZ_VOTRE_RÉPONSE_ICI"
```

L'inconvénient de `count()` est qu'il ne peut vous donner qu'une seule statistique récapitulative dans le jeu de données. Lorsque vous utilisez `summarize()` et `n()`, vous pouvez inclure plusieurs statistiques récapitulatives. Par exemple :

```
by(sexe, quartier) %>%  
size(count = n(),  
median_age = median(age))
```

```
## `summarise()` has grouped output by  
## 'sexe'. You can override using the  
## `.` argument.
```

```
## # A tibble: 18 × 4  
## # Groups: sexe [2]  
##   sexe quartier count  
##   <chr> <chr>    <int>  
## 1 Female Briqueterie     61  
## 2 Female Carriere      140  
## 3 Female Cité Verte     44  
## 4 Female Ekoudou       110
```

```

## 5 Female Messa          26
## 6 Female Mokolo         53
## 7 Female Nkomkana       43
## 8 Female Tsinga          42
## 9 Female Tsinga Oliga    30
## 10 Male   Briqueterie     45
## 11 Male   Carriere        96
## 12 Male   Cité Verte       28
## 13 Male   Ekoudou          80
## 14 Male   Messa            22
## 15 Male   Mokolo           43
## 16 Male   Nkomkana          32
## 17 Male   Tsinga            39
## 18 Male   Tsinga Oliga      37
##   median_age
##   <dbl>
## 1 28
## 2 25.5
## 3 28
## 4 26.5
## 5 27.5
## 6 23
## 7 28
## 8 29
## 9 23.5
## 10 28
## 11 27
## 12 22.5
## 13 21.5
## 14 24.5
## 15 32
## 16 27
## 17 27
## 18 21

```

Mais `count()` ne peut produire que des comptages :

```

by(sexe, quartier) %>%
()
```

```

## # A tibble: 18 × 3
## # Groups:   sexe, quartier [18]
##   sexe   quartier     n
##   <chr>  <chr>     <int>
## 1 Female Briqueterie     61
## 2 Female Carriere        140
## 3 Female Cité Verte       44
## 4 Female Ekoudou         110
## 5 Female Messa            26
## 6 Female Mokolo           53
## 7 Female Nkomkana          43
## 8 Female Tsinga            42
## 9 Female Tsinga Oliga      30

```

```

## 10 Male    Briqueterie      45
## 11 Male    Carriere        96
## 12 Male    Cité Verte      28
## 13 Male    Ekoudou         80
## 14 Male    Messa            22
## 15 Male    Mokolo           43
## 16 Male    Nkomkana        32
## 17 Male    Tsinga           39
## 18 Male    Tsinga Oliga     37

```

## Inclure les combinaisons manquantes dans les statistiques récapitulatives

Lorsque vous utilisez `group_by()` et `summarize()` sur plusieurs variables, vous obtenez une statistique récapitulative pour chaque combinaison unique des variables groupées. Par exemple, considérez le code et la sortie ci-dessous, qui comptent le nombre d'individus dans chaque groupe d'âge et de sexe :

```

by(sexe, cat_age_3) %>%
  summarise(nombre_d_individus = n())

```

```

## `summarise()` has grouped output by
## 'sexe'. You can override using the
## `.`groups` argument.

## # A tibble: 6 × 3
## # Groups:   sexe [2]
##   sexe   cat_age_3 nombre_d_individus
##   <chr>  <chr>          <int>
## 1 Female Adult            368
## 2 Female Child            155
## 3 Female Senior           26
## 4 Male   Adult            267
## 5 Male   Child            136
## 6 Male   Senior           19

```

Dans le jeu de données de sortie, il y a une ligne pour chaque combinaison de sexe et de groupe d'âge (Femme—Adulte, Femme—Enfant, etc.).

Mais que se passe-t-il si l'une de ces combinaisons n'est pas présente dans les données ?

Créons un exemple artificiel pour observer cela. Avec le code ci-dessous, nous supprimons artificiellement tous les enfants de sexe masculin du jeu de données `yao`

:

```
male_children <-
>%>
filter(!(sexe == "Male" & cat_age_3 == "Child"))
```

Maintenant, si vous exécutez le même appel `group_by()` et `summarize()` sur `yao_no_male_children`, vous remarquerez la combinaison manquante :

```
male_children %>%
  group_by(sexe, cat_age_3) %>%
  summarise(number_of_individuals = n())
```

```
## `summarise()` has grouped output by
## 'sexe'. You can override using the
## `groups` argument.
```

```
## # A tibble: 5 × 3
## # Groups:   sexe [2]
##   sexe   cat_age_3
##   <chr>  <chr>
## 1 Female  Adult
## 2 Female  Child
## 3 Female  Senior
## 4 Male    Adult
## 5 Male    Senior
##   number_of_individuals
##   <int>
## 1 368
## 2 155
## 3 26
## 4 267
## 5 19
```

En effet, il n'y a pas de ligne pour les enfants de sexe masculin.

Mais parfois, il est utile d'inclure ces combinaisons manquantes dans le jeu de données de sortie, avec une valeur `NA` ou `0` pour la statistique récapitulative.

Pour ce faire, vous pouvez exécuter le code suivant à la place :

```
male_children %>%
  # convertir les variables en facteurs
  mutate(sexe = as.factor(sexe),
        cat_age_3 = as.factor(cat_age_3)) %>%
  # ne pas supprimer l'argument .drop = FALSE
  group_by(sexe, cat_age_3, .drop = FALSE) %>%
  summarise(number_of_individuals = n())
```

```

## `summarise()` has grouped output by
## 'sexe'. You can override using the
## `.groups` argument.

## # A tibble: 6 × 3
## # Groups: sexe [2]
##   sexe   cat_age_3
##   <fct>  <fct>
## 1 Female Adult
## 2 Female Child
## 3 Female Senior
## 4 Male   Adult
## 5 Male   Child
## 6 Male   Senior
## #>   number_of_individuals
## #>             <int>
## #> 1            368
## #> 2            155
## #> 3            26
## #> 4            267
## #> 5             0
## #> 6            19

```

## Que fait ce code ?

- D'abord, il convertit les variables de regroupement en facteurs avec `as.factor()` (dans un appel à `mutate()`)
- Ensuite, il utilise l'argument `.drop = FALSE` dans la fonction `group_by()` pour éviter de supprimer les combinaisons manquantes.

Vous avez maintenant un compte clair de 0 pour le nombre d'enfants de sexe masculin !

---

Voyons un autre exemple, cette fois sans modifier artificiellement nos données.

Le code ci-dessous calcule l'âge moyen par sexe et par niveau d'éducation :

```

by(sexe, edu_haute) %>%
  use(mean_age = mean(age))

```

```

## `summarise()` has grouped output by
## 'sexe'. You can override using the
## `.groups` argument.

```

```

## # A tibble: 13 × 3
## # Groups:   sexe [2]
##   sexe   edu_haute
##   <chr>  <chr>
## 1 Female Doctorate
## 2 Female No formal instruction
## 3 Female No response
## 4 Female Primary
## 5 Female Secondary
## 6 Female University
## 7 Male   Doctorate
## 8 Male   No formal instruction
## 9 Male   No response
## 10 Male  Other
## 11 Male  Primary
## 12 Male  Secondary
## 13 Male  University
##   mean_age
##   <dbl>
## 1 28
## 2 45.6
## 3 35
## 4 26.8
## 5 28.8
## 6 31.5
## 7 42.2
## 8 37.9
## 9 22
## 10 5.5
## 11 22.9
## 12 29.4
## 13 31.9

```

Remarquez que dans le jeu de données de sortie, il y a 7 lignes pour les hommes mais seulement 6 lignes pour les femmes, car aucune femme n'a répondu "Autre" à la question sur le niveau d'éducation le plus élevé.

Si vous voulez néanmoins inclure la ligne "Femme—Autre" dans le jeu de données de sortie, vous exécuteriez :

```

`(.sex = as.factor(sexe),
 `edu_haute = as.factor(edu_haute)) %>%
 `by(sexe, edu_haute, .drop = FALSE) %>%
 `summarise(mean_age = mean(age))

```

```

## `summarise()` has grouped output by
## 'sexe'. You can override using the
## `groups` argument.

```

```

## # A tibble: 14 × 3
## # Groups:   sexe [2]
##   sexe   edu_haute
##   <fct>  <fct>
## 1 Female Doctorate
## 2 Female No formal instruction
## 3 Female No response
## 4 Female Other
## 5 Female Primary
## 6 Female Secondary
## 7 Female University
## 8 Male   Doctorate
## 9 Male   No formal instruction
## 10 Male  No response
## 11 Male  Other
## 12 Male  Primary
## 13 Male  Secondary
## 14 Male  University
## #     mean_age
## #   <dbl>
## 1    28
## 2   45.6
## 3    35
## 4    NaN
## 5   26.8
## 6   28.8
## 7   31.5
## 8   42.2
## 9   37.9
## 10   22
## 11   5.5
## 12   22.9
## 13   29.4
## 14   31.9

```



En utilisant le jeu de données `yao`, calculons l'âge médian en regroupant par quartier, catégorie d'âge et sexe.

Notez que nous voulons toutes les combinaisons possibles de ces trois variables (pas seulement celles présentes dans nos données).

Faites attention à deux impératifs de préparation des données !

- convertissez vos variables de regroupement en facteurs au préalable en utilisant `mutate()`
- calculez votre statistique, la médiane, tout en supprimant les valeurs NA.

**PRACTICE****(in RMD)**

Votre sortie doit être un jeu de données avec quatre colonnes nommées comme indiqué ci-dessous :

**quartier cat\_age\_3 sexe median\_age**

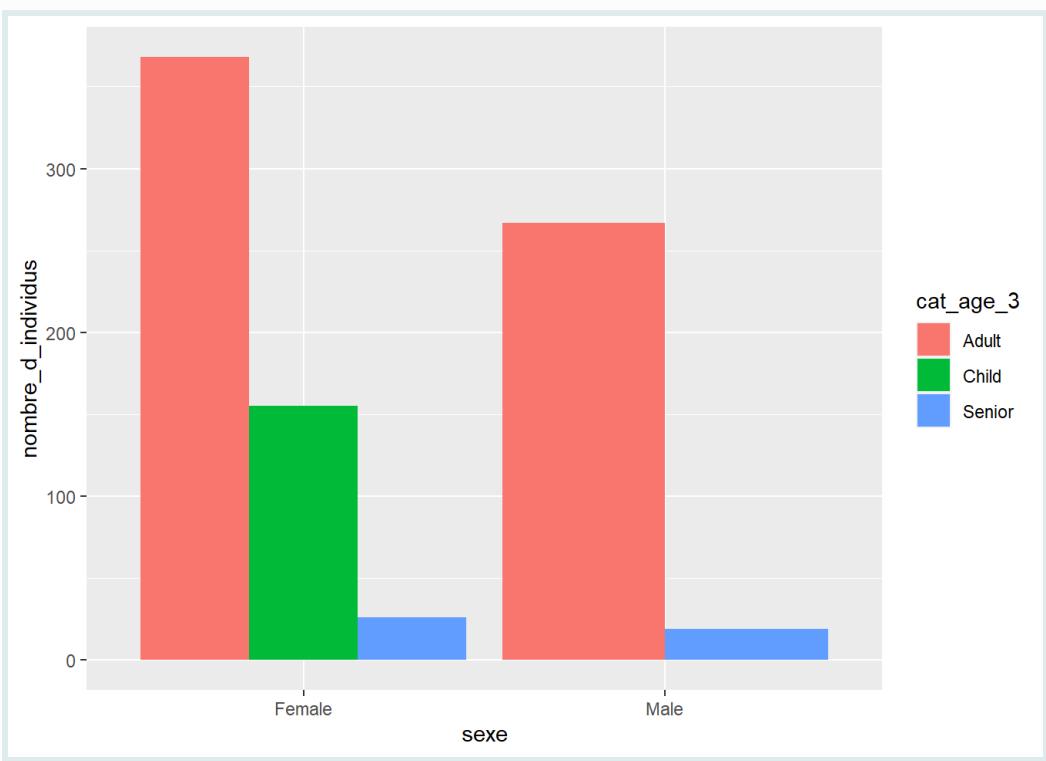
```
me_quartier_age_sexe <-  
>%  
EZ_VOTRE_RÉPONSEICI"
```

**SIDE NOTE**

Pour réaliser un diagramme à barres composé avec les comptes d'âge sexe de `yao_no_male_children`, vous pourriez exécuter :

```
ale_children %>%  
by(sexe, cat_age_3) %>%  
ise(nombre_d_individus = n()) %>%  
ip() %>%  
  
ismettre la sortie à ggplot  
+  
col(aes(x = sexe, y = nombre_d_individus, fill = cat_age_3),  
position = "dodge")
```

```
## `summarise()` has grouped output by  
## 'sexe'. You can override using the  
## `groups` argument.
```



#### SIDE NOTE

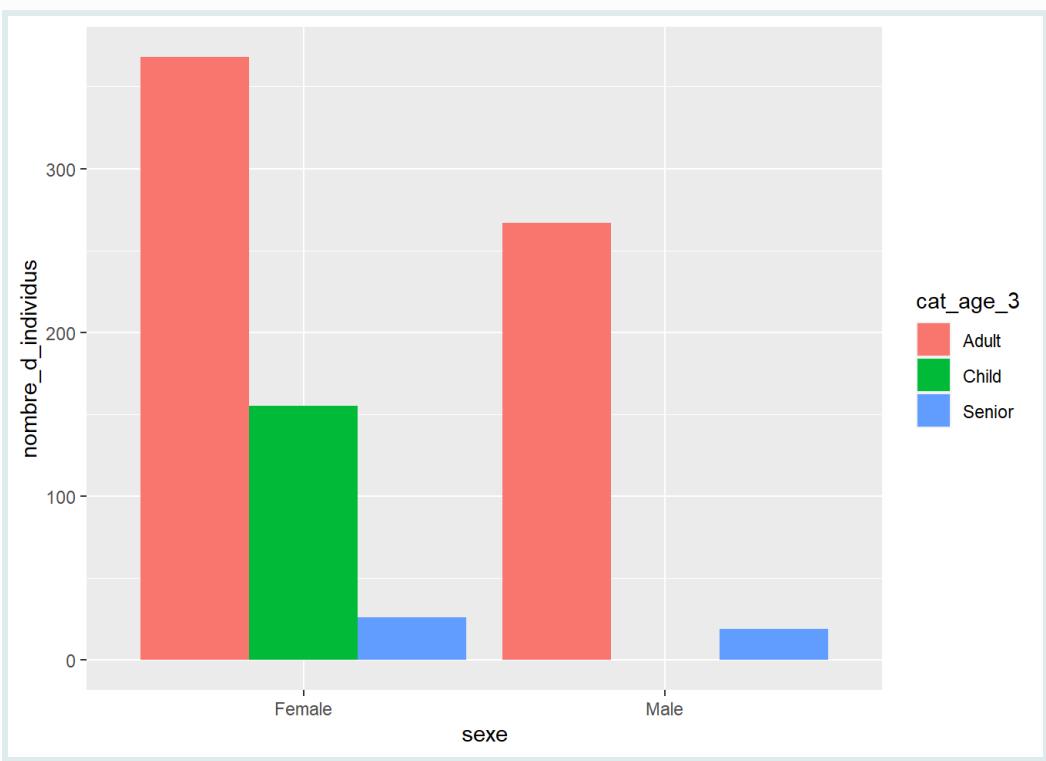


Pas très élégant ! Idéalement, il devrait y avoir un espace vide indiquant 0 pour le nombre d'enfants de sexe masculin.

Si vous mettez en œuvre la procédure pour inclure les combinaisons manquantes, vous obtenez un diagramme à barres composé plus naturel, avec un espace vide pour les enfants de sexe masculin :

```
male_children %>%
  `summarise`(
    .by(sexe = as.factor(sexe),
        cat_age_3 = as.factor(cat_age_3)) %>%
    `group_by`(sexe, cat_age_3, .drop = FALSE) %>%
    `summarise`(nombre_d_individus = n()) %>%
    `ungroup`() %>%
    `mutate`(
      .by(sexe, cat_age_3),
      nombre_d_individus = ifelse(sexe == "Male" & cat_age_3 == "Child", 0, nombre_d_individus))
  ) %>%
  `ggplot`(.,
    aes(x = sexe, y = nombre_d_individus, fill = cat_age_3),
    position = "dodge")
```

```
## `summarise()` has grouped output by
## 'sexe'. You can override using the
## `.` argument.
```



#### SIDE NOTE



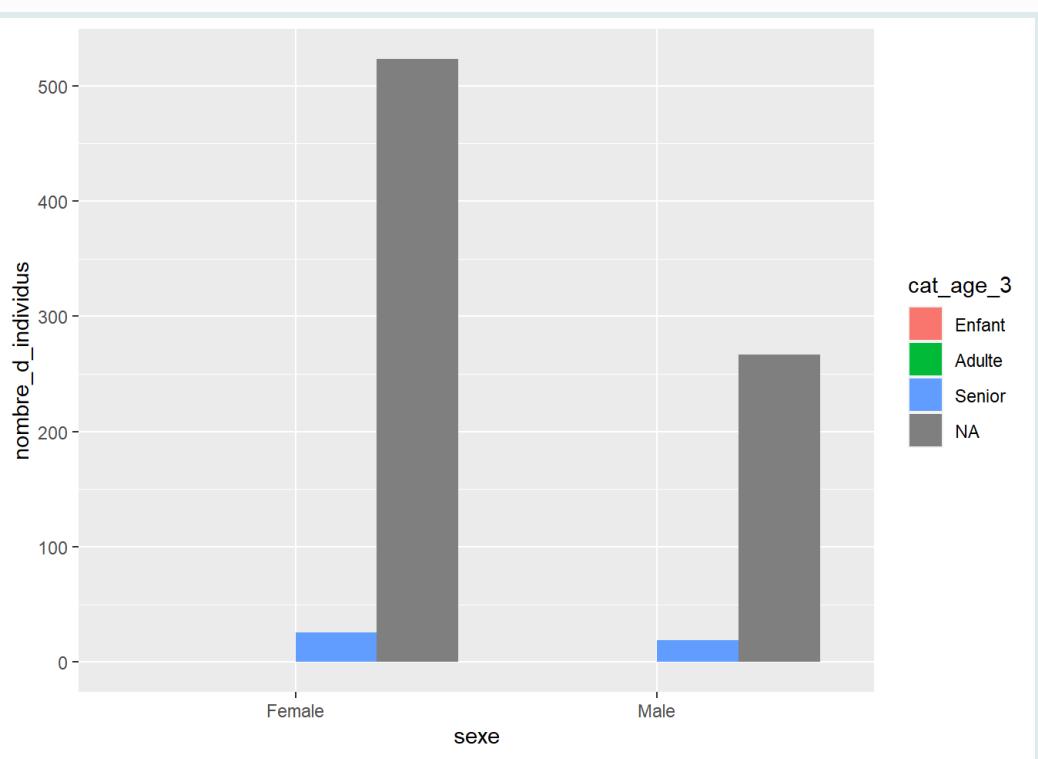
Beaucoup mieux !

Au fait, cette sortie peut être légèrement améliorée en définissant les niveaux de facteur pour l'âge dans leur ordre croissant correct : d'abord “Enfant”, puis “Adulte” puis “Senior” :

```
male_children %>%
  e(sexe = as.factor(sexe),
    cat_age_3 = factor(cat_age_3,
      levels = c("Enfant",
                 "Adulte",
                 "Senior")))) %>%
  by(sexe, cat_age_3, .drop = FALSE) %>%
  use(nombrer_d_individus = n()) %>%
  ip() %>%
  ismettre la sortie à ggplot
  () +
  col(aes(x = sexe, y = nombrer_d_individus, fill = cat_age_3),
      position = "dodge")
```

## `summarise()` has grouped output by  
## 'sexe'. You can override using the  
## `groups` argument.

### SIDE NOTE



## Conclusion

Vous avez maintenant vu comment obtenir des statistiques récapitulatives rapides à partir de vos données, soit pour l'exploration de données, soit pour une présentation ou une visualisation de données supplémentaires.

De plus, vous avez découvert l'une des merveilles de {dplyr}, la possibilité de grouper vos données à l'aide de `group_by()`.

`group_by()` combiné avec `summarize()` est l'une des manipulations de regroupement les plus courantes.

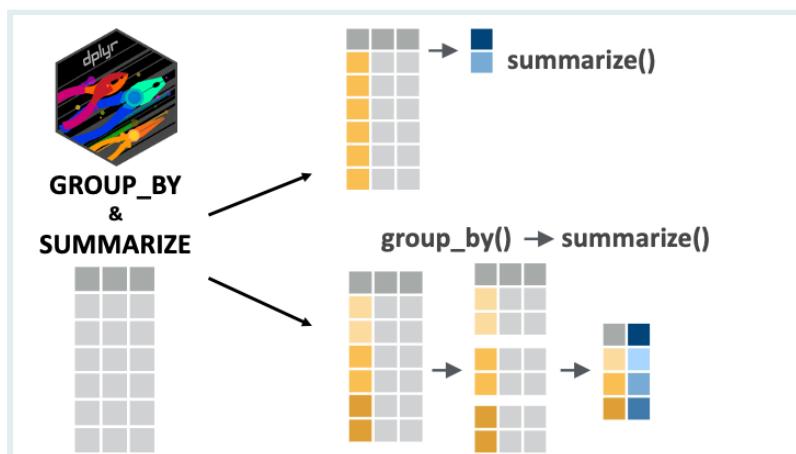


Fig: summarize() et son utilisation combinée avec group\_by().

Cependant, vous pouvez également combiner `group_by()` avec de nombreux autres verbes {dplyr} : c'est ce que nous couvrirons dans notre prochaine leçon. À bientôt !

---

## Contributeurs

Les membres suivants de l'équipe ont contribué à cette leçon:



### LAURE VANCAUWENBERGHE

Data analyst, the GRAPH Network

A firm believer in science for good, striving to ally programming, health and education

---



### ANDREE VALLE CAMPOS

R Developer and Instructor, the GRAPH Network

Motivated by reproducible science and education

---



### KENE DAVID NWOSU

Data analyst, the GRAPH Network

Passionate about world improvement

---



### SABINA RODRIGUEZ VELÁSQUEZ

Project Manager and Scientific Collaborator, The GRAPH Network

Infectiously enthusiastic about microbes and Global Health

---

Merci à [Alice Osmaston](#) et [Saifeldin Shehata](#) pour leurs commentaires et leur revue.

---

## Références

Certaines informations de cette leçon ont été adaptées des sources suivantes :

- Horst, A. (2022). *Dplyr-learnr*. <https://github.com/allisonhorst/dplyr-learnr> (Travail original publié en 2020)
- *Group by one or more variables*. (s.d.). Consulté le 21 février 2022, sur [https://dplyr.tidyverse.org/reference/group\\_by.html](https://dplyr.tidyverse.org/reference/group_by.html)

- *Summarise each group to fewer rows.* (s.d.). Consulté le 21 février 2022, sur <https://dplyr.tidyverse.org/reference/summarize.html>
- The Carpentries. (s.d.). *Grouped operations using `dplyr`*. Grouped operations using `dplyr` – Introduction to R/tidyverse for Exploratory Data Analysis. Consulté le 28 juillet 2022, sur [https://tavareshugo.github.io/r-intro-tidyverse-gapminder/06-grouped\\_operations\\_dplyr/index.html](https://tavareshugo.github.io/r-intro-tidyverse-gapminder/06-grouped_operations_dplyr/index.html)

L'œuvre d'art a été adaptée de :

- Horst, A. (2022). *R & stats illustrations by Allison Horst*. <https://github.com/allisonhorst/stats-illustrations> (Travail original publié en 2018)

# Notes de leçon | Filtrer, modifier et organiser par groupes

February 2024



Introduction	.
Objectifs d'apprentissage	.
Packages	.
Jeux de données	.
Organisation par groupe	.
<code>arrange()</code> peut regrouper automatiquement	.
Filtrage par groupe	.
Filtrage avec des groupements imbriqués	.
Modification par groupe	.
Modification avec des groupes imbriqués	.
Conclusion	.

---

## Introduction

La manipulation de données implique souvent d'appliquer les mêmes opérations séparément à différents groupes au sein des données. Ce motif, parfois appelé “split-apply-combine”, est facilement réalisable dans `{dplyr}` en enchaînant le verbe `group_by()` avec d'autres verbes de manipulation tels que `filter()`, `mutate()` et `arrange()` (que vous avez déjà vus !).

Dans cette leçon, vous deviendrez confiant avec ce type de manipulations groupées.

Commençons.

---

## Objectifs d'apprentissage

1. Vous pouvez utiliser `group_by()` avec `arrange()`, `filter()` et `mutate()` pour effectuer des opérations groupées sur un jeu de données.

---

## Packages

Cette leçon exigera la suite de packages `{tidyverse}` et le package `{here}` :

```
aire(pacman)) install.packages("pacman")
p_load(tidyverse, here)
```

## Jeux de données

Dans cette leçon, nous utiliserons encore les données de l'enquête sérologique COVID-19 menée à Yaoundé, au Cameroun. Ci-dessous, nous importons les données, créons un petit sous-ensemble du jeu de données, `yao` et un sous-ensemble encore plus petit, `yao_sexe_poids`.

```
csv(here::here('data/fr_yaounde_data.csv')) %>%  
  t(sexe, age, cat_age, poids_kg, occupation, resultat_igg, resultat_igm)
```

```
## # A tibble: 5 × 7  
##   sexe      age  cat_age  poids_kg  
##   <chr>     <dbl> <chr>       <dbl>  
## 1 Female    45  45 - 64      95  
## 2 Male      55  45 - 64      96  
## 3 Male      23  15 - 29      74  
## 4 Female    20  15 - 29      70  
## 5 Female    55  45 - 64      67  
##   occupation      resultat_igg  
##   <chr>            <chr>  
## 1 Informal worker Negative  
## 2 Salaried worker Positive  
## 3 Student        Negative  
## 4 Student        Positive  
## 5 Trader--Farmer Positive  
## # 1 more variable: resultat_igm <chr>
```

```
_poids <-  
%>%  
  t(sexe, poids_kg)  
_poids
```

```
## # A tibble: 5 × 2  
##   sexe  poids_kg  
##   <chr>    <dbl>  
## 1 Female    95  
## 2 Male      96  
## 3 Male      74  
## 4 Female    70  
## 5 Female    67
```

Pour les questions pratiques, nous utiliserons également le jeu de données sur la sarcopénie que vous avez déjà vu :

```
nia <- read_csv(here::here('data/fr_sarcopenia_elderly.csv'))  
nia
```

```
## # A tibble: 5 × 9  
##   numero    age groupe_d_age  
##   <dbl> <dbl> <chr>  
## 1     7  60.8 Sixties  
## 2     8  72.3 Seventies  
## 3     9  62.6 Sixties  
## 4    12    72  Seventies  
## 5    13  60.1 Sixties  
## #> #>   sexe_homme_1_femme_0 statut_marital  
## #> #>   <dbl> <chr>  
## #> 1           0 married  
## #> 2           1 married  
## #> 3           0 married  
## #> 4           0 widow  
## #> 5           0 married  
## #> #> 4 more variables: taille_metros <dbl>, poids_kg <dbl>,  
## #> #>   force_de_prehension_kg <dbl>, ...
```

---

## Organisation par groupe

La fonction `arrange()` ordonne les lignes d'un jeu de données par les valeurs des colonnes sélectionnées. Cette fonction est sensible aux groupements uniquement lorsque nous définissons son argument `.by_group` à `TRUE`. Pour illustrer cela, considérons le jeu de données `yao_sexe_poids` :

```
_poids
```

```
## # A tibble: 5 × 2  
##   sexe   poids_kg  
##   <chr>    <dbl>  
## 1 Female    95  
## 2 Male      96  
## 3 Male      74  
## 4 Female    70  
## 5 Female    67
```

Nous pouvons organiser ce jeu de données par poids de cette façon :

```
_poids %>%
  `by`(poids_kg)
```

```
## # A tibble: 5 × 2
##   sexe   poids_kg
##   <chr>     <dbl>
## 1 Female      14
## 2 Male        15
## 3 Male        15
## 4 Male        15
## 5 Female      15
```

Comme prévu, les poids les plus faibles ont été placés en haut de la trame de données.

Si nous regroupons d'abord les données, nous pourrions nous attendre à un résultat différent :

```
_poids %>%
  `by`(sexe) %>%
  `re`(poids_kg)
```

```
## # A tibble: 5 × 2
## # Groups:   sexe [2]
##   sexe   poids_kg
##   <chr>     <dbl>
## 1 Female      14
## 2 Male        15
## 3 Male        15
## 4 Male        15
## 5 Female      15
```

Mais comme vous le voyez, l'organisation est toujours la même.

Ce n'est que lorsque nous définissons l'argument `.by_group = TRUE` que nous obtenons quelque chose de différent :

```
_poids %>%
  `by`(sexe) %>%
  `re`(poids_kg, .by_group = TRUE)
```

```
## # A tibble: 5 × 2
## # Groups:   sexe [1]
##   sexe   poids_kg
##   <chr>     <dbl>
## 1 Female      14
## 2 Female      15
## 3 Female      16
```

```
## 4 Female      16
## 5 Female      18
```

Maintenant, les données sont *d'abord* triées par sexe (toutes les femmes en premier), puis par poids.

`arrange()` peut regrouper automatiquement

En réalité, nous n'avons pas besoin de `group_by()` pour organiser par groupe ; nous pouvons simplement mettre plusieurs variables dans la fonction `arrange()` pour obtenir le même effet.

Ainsi, cette simple commande `arrange()` :

```
_poids %>%
  by(sexe, poids_kg)
```

```
## # A tibble: 5 × 2
##   sexe     poids_kg
##   <chr>    <dbl>
## 1 Female     14
## 2 Female     15
## 3 Female     16
## 4 Female     16
## 5 Female     18
```

est équivalente à la commande plus complexe `group_by()`, `arrange()` utilisée précédemment :

```
_poids %>%
  by(sexe) %>%
  by(poids_kg, .by_group = TRUE)
```

La commande `arrange(sexe, poids_kg)` dit à R d'organiser les lignes *d'abord* par sexe, puis par poids.

Évidemment, cette syntaxe, avec juste `arrange()`, et pas de `group_by()` est plus simple, donc vous pouvez vous y tenir.

## Rappel

`desc()` pour l'ordre décroissant

Rappelez-vous que pour classer *en ordre décroissant*, nous pouvons mettre la variable cible dans `desc()`. Donc, par exemple, pour trier par sexe et poids, mais avec les personnes les plus lourdes en haut, nous pouvons exécuter :

```
_poids %>%
  by(sexe, desc(poids_kg))
```

```
## # A tibble: 5 × 2
##   sexe     poids_kg
##   <chr>     <dbl>
## 1 Female     162
## 2 Female     161
## 3 Female     158
## 4 Female     135
## 5 Female     129
```



Avec une commande `arrange()`, triez les données `sarcopenia` d'abord par sexe, puis par force de préhension. (Si c'est fait correctement, la première ligne devrait être celle d'une femme avec une force de préhension de 1,3 kg). Pour rendre l'organisation claire, vous devriez d'abord `select()` les variables de sexe et de force de préhension.

```
étez le code avec votre réponse :
_de_prehension_arrangee <-
penia %>%
  : (_____) %>%
  ge (_____)
```



Le jeu de données `sarcopenia` contient une colonne `groupe_d_age`, qui stocke les groupes d'âge en chaîne de caractères (les groupes d'âge sont "Sixties", "Seventies" et "Eighties"). Convertissez cette variable en un facteur avec les niveaux dans le bon ordre (d'abord "Sixties", puis "Seventies" et ainsi de suite). (Astuce : Revenez sur la leçon `case_when()` si vous ne voyez pas comment réattribuer un niveau à un facteur).

Ensuite, avec une commande imbriquée à `arrange()`, classez les données d'abord par la variable facteur `groupe_d_age` nouvellement créée (les individus les plus jeunes en premier) puis par `taille_metros`, avec les individus les plus petits en premier.

```
étez le code avec votre réponse :
_d_age_taille <-
penia %>%
"EZ_VOTRE_RÉPONSEICI"
```

## Filtrage par groupe

La fonction `filter()` conserve ou supprime des lignes en fonction d'une condition. Si `filter()` est appliquée à des données groupées, l'opération de filtrage est réalisée séparément pour chaque groupe.

Pour illustrer cela, considérons à nouveau le jeu de données `yao_sexe_poids` :

```
yao_poids
```

```
## # A tibble: 5 × 2
##   sexe     poids_kg
##   <chr>     <dbl>
## 1 Female     95
## 2 Male       96
## 3 Male       74
## 4 Female     70
## 5 Female     67
```

Si nous voulons filtrer les données pour la personne la plus lourde, nous pourrions exécuter :

```
yao_poids %>%
  filter(poids_kg == max(poids_kg))
```

```
## # A tibble: 1 × 2
##   sexe     poids_kg
##   <chr>     <dbl>
## 1 Female     162
```

Mais si nous voulons obtenir la personne la plus lourde par groupe de sexe (l'homme le plus lourd *et* la femme la plus lourde), nous pouvons utiliser `group_by(sex)` puis `filter()` :

```
yao_poids %>%
  group_by(sexe) %>%
  filter(poids_kg == max(poids_kg))
```

```
## # A tibble: 2 × 2
## # Groups:   sexe [2]
##   sexe     poids_kg
##   <chr>     <dbl>
## 1 Male      128
## 2 Female    162
```

Parfait ! Le code ci-dessus peut être traduit par “Pour chaque groupe de sexe, conserve la ligne avec la valeur maximale `poids_kg`”.

## Filtrage avec des groupements imbriqués

La fonction `filter()` fonctionne bien avec un nombre quelconque de groupements imbriqués.

Par exemple, si nous voulons voir l’homme le plus lourd et la femme la plus lourde *par groupe d’âge*, nous pourrions exécuter le code suivant sur le jeu de données `yao` :

```
by(sexe, cat_age) %>%
  (poids_kg == max(poids_kg))
```

Ce code regroupe par sexe *et* catégorie d’âge, puis trouve la personne la plus lourde dans chaque sous-catégorie.

(Pourquoi avons-nous 10 lignes dans la sortie ? Eh bien, 2 groupes de sexe x 5 groupes d’âge = 10 groupements uniques.)

La sortie est un peu dispersée, donc nous pouvons enchaîner cela avec la fonction `arrange()`, pour organiser par sexe et groupe d’âge.

```
by(sexe, cat_age) %>%
  (poids_kg == max(poids_kg)) %>%
  arrange(sexe, cat_age)
```

Maintenant, les données sont plus faciles à lire. Toutes les femmes viennent en premier, puis les hommes. Mais nous remarquons un arrangement étrange des groupes d’âge ! Ceux âgés de 5 à 14 ans devraient venir *en premier* dans l’arrangement. Bien sûr, nous avons appris comment corriger cela avec la fonction `factor()` et son argument `levels` :

```
cat_age = factor(
  age,
  levels = c("5 - 14", "15 - 29", "30 - 44", "45 - 64", "65 +"))
by(sexe, cat_age) %>%
  (poids_kg == max(poids_kg)) %>%
  arrange(sexe, cat_age)
```

Maintenant, nous avons une sortie bien organisée !

### PRACTICE



**PRACTICE**

Groupez le jeu de données `sarcopenia` par groupe d'âge et sexe, puis filtrez pour obtenir l'index de masse musculaire le plus élevé dans chaque groupe (imbriqué).

```
étez le code avec votre réponse :  
de_muscle_squelettique_max <-  
sarcopenia %>%  
  EZ_VOTRE_RÉPONSEICI"
```

## Modification par groupe

`mutate()` est utilisé pour modifier les colonnes ou pour en créer de nouvelles. Avec des données groupées, `mutate()` opère sur chaque groupe indépendamment.

Considérons d'abord un appel régulier à `mutate()`, pas un groupé. Imaginez que vous voulez ajouter une colonne qui classe les répondants par poids. Cela peut être fait avec la fonction `rank()` à l'intérieur d'un appel à `mutate()` :

```
%>%  
  (poids_ordre = rank(poids_kg))
```

```
## # A tibble: 5 × 3  
##   sexe   poids_kg   poids_ordre  
##   <chr>     <dbl>       <dbl>  
## 1 Female      95        901  
## 2 Male        96        908  
## 3 Male        74        640.  
## 4 Female      70        564.  
## 5 Female      67        502.
```

La sortie montre que la première ligne est le 901ème individu le plus léger. Mais il serait plus intuitif de classer dans l'ordre décroissant avec la personne la plus lourde en premier. Nous pouvons le faire avec la fonction `desc()` :

```
%>%  
  (poids_ordre = rank(desc(poids_kg)))
```

```
## # A tibble: 5 × 3  
##   sexe   poids_kg   poids_ordre  
##   <chr>     <dbl>       <dbl>  
## 1 Female      95        71  
## 2 Male        96        64  
## 3 Male        74        332.
```

```
## 4 Female      70      408.  
## 5 Female      67      470.
```

La sortie montre que la personne de la première ligne est la 71ème personne la plus lourde.

Maintenant, essayons d'écrire un appel groupé à `mutate()`. Imaginez que nous voulons ajouter cette colonne de classement par poids *par groupe de sexe* dans le jeu de données. Autrement dit, nous voulons connaître le classement de chaque personne par poids dans leur catégorie de sexe. Dans ce cas, nous pouvons combiner `group_by(sex)` avec `mutate()` :

```
_poids %>%  
  by(sexe) %>%  
  (poids_ordre = rank(desc(poids_kg)))
```

```
## # A tibble: 5 × 3  
## # Groups:   sexe [2]  
##   sexe     poids_kg   poids_ordre  
##   <chr>     <dbl>       <dbl>  
## 1 Female     95       53.5  
## 2 Male       96      13.5  
## 3 Male       74      148  
## 4 Female     70      220.  
## 5 Female     67      250.
```

Nous voyons maintenant que la personne de la première ligne est la 53ème femme la plus lourde. (Le .5 indique que ce rang est à égalité avec quelqu'un d'autre dans les données.)

Nous pourrions également organiser les données pour rendre les choses plus claires :

```
_poids %>%  
  by(sexe) %>%  
  (poids_ordre = rank(desc(poids_kg))) %>%  
  ge(sexe, poids_ordre)
```

```
## # A tibble: 5 × 3  
## # Groups:   sexe [1]  
##   sexe     poids_kg   poids_ordre  
##   <chr>     <dbl>       <dbl>  
## 1 Female     162       1  
## 2 Female     161       2  
## 3 Female     158       3  
## 4 Female     135       4  
## 5 Female     129       5
```

## Modification avec des groupes imbriqués

Bien sûr, comme avec les autres verbes que nous avons vus, `mutate()` fonctionne aussi avec des groupes imbriqués.

Par exemple, ci-dessous nous créons le groupe imbriqué d'âge *et* de sexe avec le jeu de données `yao`, puis nous ajoutons une colonne de rang avec `mutate()` :

```
by(sexe, cat_age) %>%  
  poids_ordre = rank(desc(poids_kg))
```

```
## # A tibble: 5 × 8  
## # Groups: sexe, cat_age [4]  
##   sexe      age cat_age poids_kg  
##   <chr>     <dbl> <chr>      <dbl>  
## 1 Female    45  45 - 64       95  
## 2 Male      55  45 - 64       96  
## 3 Male      23  15 - 29       74  
## 4 Female    20  15 - 29       70  
## 5 Female    55  45 - 64       67  
## # ... with 2 more variables: resultat_igm <chr>, poids_ordre <dbl>
```

La sortie montre que la personne de la première ligne est la 20ème femme la plus lourde *dans le groupe d'âge de 45 à 64 ans*.



Avec les données `sarcopenia`, **groupez par** `groupe_d_age`, **puis dans une nouvelle variable appelée** `force_prehension_ordre`, **calculez le rang de la force de préhension de chaque individu par groupe d'âge**. (Pour calculer le rang, utilisez `mutate()` et la fonction `rank()` avec sa méthode par défaut pour les égalités.)

```
étez le code avec votre réponse :  
ement_force_de_prehension <-  
penia %>%  
EZ_VOTRE_RÉPONSEICI"
```

**WATCH OUT**

## Attention

**N'oubliez pas de dégrouper les données avant de procéder à une analyse plus approfondie**

Comme il a été mentionné auparavant, il est important de dégrouper vos données avant de faire une analyse plus détaillée.

Considérez ce dernier exemple, où nous avons calculé le rang de poids des individus par groupe d'âge et de sexe :

```
by(sexe, cat_age) %>%
  `(poids_ordre = rank(desc(poids_kg)))
```

**WATCH OUT**



```
## # A tibble: 5 × 8
## # Groups:   sexe, cat_age [4]
##   sexe      age cat_age poids_kg
##   <chr>     <dbl> <chr>    <dbl>
## 1 Female    45  45 - 64     95
## 2 Male      55  45 - 64     96
## 3 Male      23  15 - 29     74
## 4 Female    20  15 - 29     70
## 5 Female    55  45 - 64     67
## #     occupation      resultat_igg
## #   <chr>            <chr>
## 1 Informal worker Negative
## 2 Salaried worker Positive
## 3 Student        Negative
## 4 Student        Positive
## 5 Trader--Farmer Positive
## # # i 2 more variables: resultat_igm <chr>, poids_ordre <dbl>
```

Si, dans le processus d'analyse, vous avez stocké cette sortie comme un nouveau jeu de données :

```
.fie <-
>%>%
  by(sexe, cat_age) %>%
  `(poids_ordre = rank(desc(poids_kg)))
```

Et puis, plus tard, vous avez repris le jeu de données et essayé une autre analyse, par exemple, en filtrant pour obtenir la personne la plus âgée dans les données :

```

.fie %>%
: (age == max(age))

## # A tibble: 5 × 8
## # Groups: sexe, cat_age [5]
##   sexe      age cat_age poids_kg
##   <chr>    <dbl> <chr>     <dbl>
## 1 Male       65 45 - 64      93
## 2 Male       78 65 +        95
## 3 Male       14 5 - 14      44
## 4 Female     44 30 - 44      67
## 5 Female     79 65 +        40
##   occupation      resultat_igg
##   <chr>           <chr>
## 1 Retired        Negative
## 2 Retired--Informal work... Positive
## 3 Student         Negative
## 4 Home-maker     Positive
## 5 Retired        Negative
## # i 2 more variables: resultat_igm <chr>, poids_ordre <dbl>

```

### WATCH OUT



Vous pourriez être confus par la sortie! Pourquoi y a-t-il 55 lignes de "personnes les plus âgées"?

Cela serait dû au fait que vous avez oublié de dégrouper les données avant de les stocker pour une analyse plus approfondie. Faisons cela correctement maintenant

```

.fie <-
>%>
by(sexe, cat_age) %>%
  (poids_ordre = rank(desc(poids_kg)) ) %>%
  ip()

```

Maintenant, nous pouvons obtenir correctement la/les personne(s) la/les plus âgée(s) dans le jeu de données :

```

.fie %>%
: (age == max(age))

```

```

## # A tibble: 2 × 8
##   sexe      age cat_age poids_kg
##   <chr>    <dbl> <chr>     <dbl>
## 1 Female     79 65 +        40
## 2 Female     79 65 +        81
##   occupation resultat_igg resultat_igm
##   <chr>           <chr>           <chr>

```

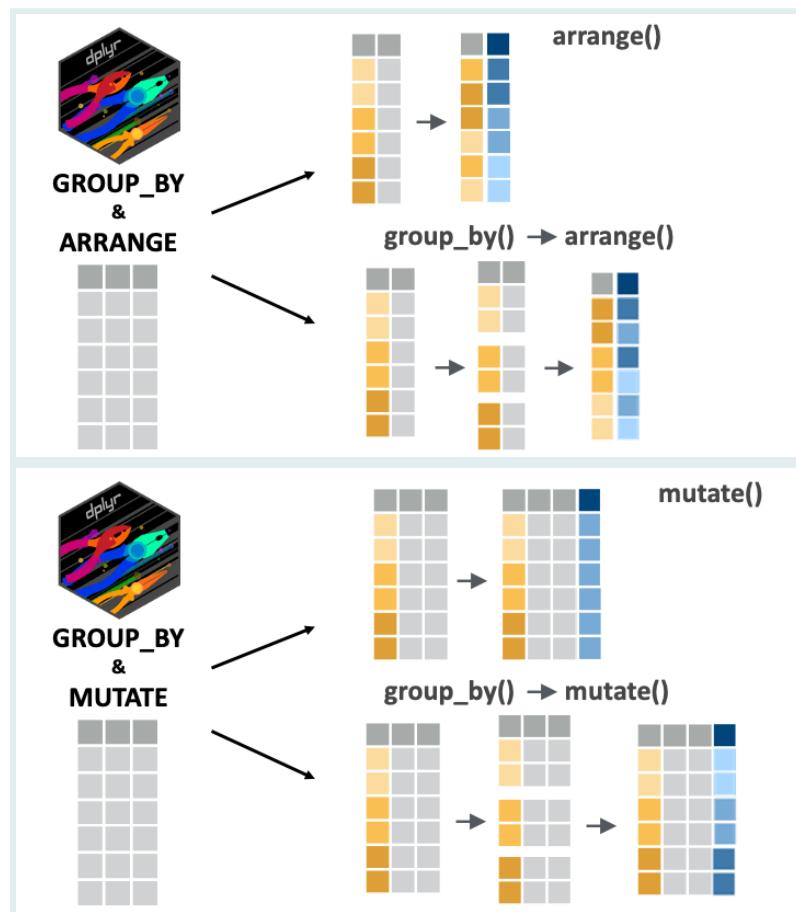
### WATCH OUT



```
## 1 Retired    Negative    Negative  
## 2 Home-maker Negative    Negative  
## # i 1 more variable: poids_ordre <dbl>
```

## Conclusion

`group_by()` est un outil merveilleux pour arranger, modifier, filtrer en fonction des groupes au sein d'une ou plusieurs variables.



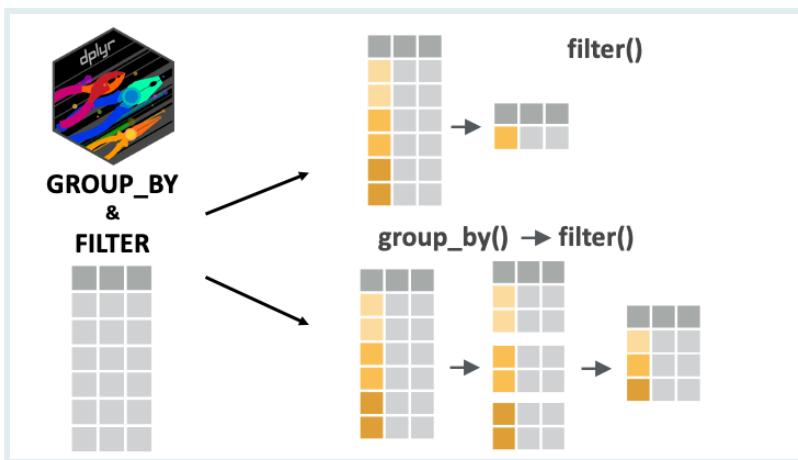


Fig: `filter()` et son utilisation combinée avec `group_by()`

Il existe de nombreuses façons de combiner ces verbes pour manipuler vos données. Nous vous invitons à prendre un peu de temps pour essayer ces verbes dans différentes combinaisons !

À la prochaine !

## Contributeurs

Les membres de l'équipe suivants ont contribué à cette leçon :



### LAURE VANCAUWENBERGHE

Data analyst, the GRAPH Network  
A firm believer in science for good, striving to ally programming, health and education



### KENE DAVID NWOSU

Data analyst, the GRAPH Network  
Passionate about world improvement



### GUY WAFFEU

R Instructor and Public Health Physician  
Committed to improving the quality of data analysis



### SABINA RODRIGUEZ VELÁSQUEZ

Project Manager and Scientific Collaborator, The GRAPH Network  
Infectiously enthusiastic about microbes and Global Health

---

## Références

Certains matériaux de cette leçon ont été adaptés des sources suivantes :

- Horst, A. (2022). *Dplyr-learnr*. <https://github.com/allisonhorst/dplyr-learnr> (Travail original publié en 2020)
- *Grouper par une ou plusieurs variables*. (n.d.). Consulté le 21 Février 2022, sur [https://dplyr.tidyverse.org/reference/group\\_by.html](https://dplyr.tidyverse.org/reference/group_by.html)
- *Créer, modifier et supprimer des colonnes — Mutate*. (n.d.). Consulté le 21 Février 2022, sur <https://dplyr.tidyverse.org/reference/mutate.html>
- *Sélectionner des lignes en utilisant les valeurs des colonnes — Filter*. (n.d.). Consulté le 21 Février 2022, sur <https://dplyr.tidyverse.org/reference/filter.html>
- *Arranger les lignes par valeurs de colonnes — Arrange*. (n.d.). Consulté le 21 Février 2022, sur <https://dplyr.tidyverse.org/reference/arrange.html>

L'œuvre d'art a été adaptée de :

- Horst, A. (2022). *Illustrations R & stats par Allison Horst*. <https://github.com/allisonhorst/stats-illustrations> (Travail original publié en 2018)

# Notes de leçon | Pivoter des données

February 2024



Introduction .....
Objectifs d'apprentissage .....
Packages .....
Que signifient large et long ? .....
Quand devez-vous utiliser des données en format large ou long ? .....
Pivoter de large à long .....
Pivoter de long à large .....
Pourquoi les données au format long sont-elles meilleures pour l'analyse? .....
Filtrer les données groupées .....
Résumer les données regroupées .....
Conception des graphiques .....
Le pivot peut être difficile .....
Conclusion ! .....

---

## Introduction

Le pivotement ou la mise en forme est une technique de manipulation de données qui consiste à réorienter les lignes et les colonnes d'un ensemble de données. Cela est parfois nécessaire pour faciliter l'analyse des données ou pour rendre les données plus compréhensibles.

Dans cette leçon, nous verrons comment pivoter efficacement les données en utilisant `pivot_longer()` et `pivot_wider()` du package `tidyverse`.

---

## Objectifs d'apprentissage

- Vous comprendrez ce que sont le format de données large et le format de données long.
- Vous saurez comment pivoter des données longues en données larges en utilisant `pivot_long()`
- Vous saurez comment pivoter des données larges en données longues en utilisant `pivot_wider()`
- Vous comprendrez pourquoi le format de données long est plus facile pour le tracé et la manipulation dans R.

## Packages

```
er les packages
uire(pacman)) install.packages("pacman")
p_load(tidyverse, outbreaks, janitor, rio, here, knitr)
```

## Que signifient large et long ?

Les termes large et long sont mieux compris dans le contexte des jeux de données exemplaires. Examinons-en quelques-uns maintenant.

Imaginez que vous avez trois patients chez qui vous collectez des données de pression artérielle sur trois jours.

Vous pouvez enregistrer les données dans un format large comme celui-ci :

patient	pression_arterielle_jour_1	pression_arterielle_jour_2	pression_arterielle_jour_3
A	110	112	114
B	120	122	124
C	100	104	105

Fig: ensemble de données large pour une série temporelle de patients.

Ou vous pourriez enregistrer les données dans un format long comme suit :

patient	jour	pression_arterielle
A	1	110
	2	112
	3	114
B	1	120
	2	122
	3	124
C	1	100
	2	104
	3	105

Fig: ensemble de données long pour une série temporelle de patients.

Prenez une minute pour étudier les deux ensembles de données pour vous assurer que vous comprenez la relation entre eux.

Dans l'ensemble de données large, chaque unité d'observation (chaque patient) n'occupe qu'une seule ligne. Et chaque mesure (pression artérielle jour 1, pression artérielle jour 2...) se trouve dans une colonne séparée.

Dans l'ensemble de données long, en revanche, chaque unité d'observation (chaque patient) occupe plusieurs lignes, avec une ligne pour chaque mesure.

---

Voici un autre exemple avec des données fictives, dans lequel les unités d'observation sont des pays :

pays	annee	mesure
x	1960	10
x	1970	13
x	2010	15
y	1960	20
y	1970	23
y	2010	25
z	1960	30
z	1970	33
z	2010	35

Fig: ensemble de données au format long où l'unité d'observation unique est un pays.

pays	annee1960	annee1970	annee2010
x	10	13	15
y	20	23	25
z	30	33	35

Fig: l'ensemble de données au format large équivalent

---

Les exemples ci-dessus sont tous deux des ensembles de données de séries temporelles, car les mesures sont répétées dans le temps (jour 1, jour 2 et ainsi de suite). Mais les concepts de long et de large sont pertinents pour d'autres types de données également, pas seulement les données de séries temporelles.

Considérez l'exemple ci-dessous, qui montre le nombre de patients dans différentes unités de trois hôpitaux :

Hopital	Unite maternite	Unite soins intensifs
Hopital A	4	2
Hopital B	5	2
Hopital C	6	3

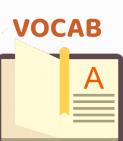
Fig: ensemble de données au format large, où chaque hôpital est une unité d'observation

Hopital	Unite	Nbr. de patients
Hopital A	Maternite	4
	Soins intensifs	2
Hopital B	Maternite	5
	Soins intensifs	2
Hopital C	Maternite	6
	Soins intensifs	3

Fig: l'ensemble de données au format long équivalent

Dans l'ensemble de données au format large, encore une fois, chaque unité d'observation (chaque hôpital) n'occupe qu'une seule ligne, avec les mesures répétées pour cette unité (nombre de patients dans différentes salles) réparties sur deux colonnes.

Dans l'ensemble de données au format long, chaque unité d'observation est répartie sur plusieurs lignes.



**VOCAB**  
Les “unités d’observation”, parfois appelées “unités statistiques” d’un ensemble de données, sont les entités ou les éléments principaux décrits par les colonnes de cet ensemble de données.

Dans le premier exemple, les unités d’observation/statistiques étaient les patients ; dans le deuxième exemple, les pays, et dans le troisième exemple, les hôpitaux.



Considérez l'ensemble de données fictif créé ci-dessous :

```
countries <-  
  frame(  
    s = c("Suède", "Danemark", "Norvège"),  
    moy.1994 = 1:3,  
    moy.1995 = 3:5,  
    moy.1996 = 5:7)  
countries
```

```

##      pays tempmoy.1994 tempmoy.1995
## 1   Suède          1          3
## 2 Danemark         2          4
## 3 Norvège          3          5
##   tempmoy.1996
## 1           5
## 2           6
## 3           7

```

## Ces données sont-elles dans un format large ou long ?

*la chaîne "large" ou la chaîne "long"*  
`donnees <- "_____"`

*ez votre réponse*  
`?_type_donnees()`

### PRACTICE



**(in RMD)**

```

## Vous n'avez pas encore défini l'objet de réponse,
`Q_type_donnees`.
## ▷1◁ 2 3 4 5

```

`_type_donnees()`

```

##
## INDICE.
##   Chaque unité observationnelle (chaque pays) occupe que
une seule ligne.

```

*obtenir la solution, exécutez la ligne ci-dessous !*  
`ON_Q_type_donnees()`

```

##
## SOLUTION
##   "large"

```

*question a une fonction solution similaire à celle-ci.  
INT est remplacé par SOLUTION dans le nom de la fonction.)  
vous devrez taper le nom de la fonction par vous-même.  
vise à vous dissuader de regarder la solution avant de répondre  
à la question.)*

---

## Quand devez-vous utiliser des données en format large ou long ?

La vérité est que cela dépend vraiment de ce que vous voulez faire ! Le format large est parfait pour *afficher les données* car il est facile de comparer visuellement les valeurs de cette manière. Le format long est idéal pour certaines tâches d'analyse de données, comme le regroupement et la conception des graphiques.

Il sera donc essentiel pour vous de savoir comment passer facilement de l'un à l'autre format. Passer du format large au format long, ou vice versa, est appelé **pivotement**.

---

### Pivoter de large à long

Pour pratiquer le pivotement d'un format large à un format long, nous allons considérer les données de [Gapminder](#) sur le **nombre de décès d'enfants en bas âge** dans certains pays au fil des ans.

**SIDE NOTE**

Gapminder est une bonne source de jeux de données riches et pertinents pour la santé. Nous vous encourageons à explorer leurs collections.

Ci-dessous, nous lisons et visualisons ces données sur les décès d'enfants en bas âge :

```
infants_large <- read_csv(here("data/fr_gapminder_infant_deaths.csv"))  
infants_large
```

```
## # A tibble: 5 × 7  
##   pays           x2010 x2011 x2012 x2013  
##   <chr>          <dbl> <dbl> <dbl> <dbl>  
## 1 Afghanistan    74600  72000  69500  67100  
## 2 Angola          79100  76400  73700  71200  
## 3 Albania          420    384    354    331  
## 4 United Arab Emirates  683    687    686    681  
## 5 Argentina        9550   9230   8860   8480  
##   x2014 x2015  
##   <dbl> <dbl>  
## 1 64800 62700  
## 2 69000 67200  
## 3 313   301
```

```
## 4    672    658
## 5    8100   7720
```

Nous observons que chaque unité d'observation (chaque pays) occupe une seule ligne, les mesures répétées étant réparties sur plusieurs colonnes. Par conséquent, cet ensemble de données est dans un format large.

Pour convertir en format long, nous pouvons utiliser une fonction pratique `pivot_longer`. Dans `pivot_longer`, nous définissons, à l'aide de l'argument `cols`, les colonnes que nous voulons pivoter :

```
infants_large %>%
  longer(cols = x2010:x2015)
```

```
## # A tibble: 5 × 3
##   pays      name  value
##   <chr>     <chr> <dbl>
## 1 Afghanistan x2010 74600
## 2 Afghanistan x2011 72000
## 3 Afghanistan x2012 69500
## 4 Afghanistan x2013 67100
## 5 Afghanistan x2014 64800
```

Très facile !

Nous pouvons observer que l'ensemble de données au format long résultant a chaque pays occupant 5 lignes (une par année entre 2010 et 2015). Les années sont indiquées dans la variable `names`, et toutes les valeurs de décompte de décès occupent une seule variable, `values`.

Une façon utile de penser à cette transformation est que les valeurs des décès d'enfants étaient auparavant en format matriciel (2 dimensions ; 2D), mais qu'elles sont maintenant en format vectoriel (1 dimension ; 1D).

Cet ensemble de données long sera beaucoup plus pratique pour de nombreuses procédures d'analyse de données.

En tant que bon analyste de données, vous pouvez trouver que les noms par défaut des variables, `names` et `values`, ne sont pas satisfaisants ; ils ne décrivent pas adéquatement ce que contiennent les variables. Pas de souci ; vous pouvez donner des noms de colonnes personnalisés, en utilisant les arguments `names_to` et `values_to` :

```
infants_large %>%
  longer(cols= x2010:x2015,
         names_to = "année",
         values_to = "nombre_deces")
```

```

## # A tibble: 5 × 3
##   pays      année nombre_deces
##   <chr>    <chr>     <dbl>
## 1 Afghanistan x2010     74600
## 2 Afghanistan x2011     72000
## 3 Afghanistan x2012     69500
## 4 Afghanistan x2013     67100
## 5 Afghanistan x2014     64800

```

**SIDE NOTE**



Remarquez que le format long est plus informatif que le format large original. Pourquoi ? À cause du nom de colonne informatif “nombre\_deces”. Dans le format large, à moins que le CSV ne soit nommé quelque chose comme `nombre_deces_enfants`, ou que quelqu’un vous dise “ce sont les comptes de décès d’enfants par pays et par année”, vous n’avez aucune idée de ce que représentent les chiffres dans les cellules.

Vous voudrez peut-être aussi supprimer le `x` devant chaque année. Cela peut être réalisé avec la fonction pratique `parse_number()` du package `{readr}` (partie de `tidyverse`), qui extrait les chiffres des chaînes :

```

enfants_large %>%
  longer(cols = x2010:x2015,
         names_to = "année",
         values_to = "nombre_deces") %>%
  mutate(année = parse_number(année))

```

```

## # A tibble: 5 × 3
##   pays      année nombre_deces
##   <chr>    <dbl>     <dbl>
## 1 Afghanistan 2010     74600
## 2 Afghanistan 2011     72000
## 3 Afghanistan 2012     69500
## 4 Afghanistan 2013     67100
## 5 Afghanistan 2014     64800

```

Super ! Nous avons maintenant un ensemble de données au format long et propres.

Pour une utilisation ultérieure, enregistrons maintenant ces données :

```

enfants_long <-
enfants_large %>%
  longer(cols = x2010:x2015,
         names_to = "annee",
         values_to = "nombre_deces")

```

Pour cette question de pratique, vous utiliserez le jeu de données `fr_euro_births_wide` provenant d'Eurostat. Il montre le nombre annuel de naissances dans 50 pays européens :

```
ces_euro_large <-  
  csv(here("data/fr_euro_births_wide.csv"))  
  naissances_euro_large)
```

```
## # A tibble: 5 × 8  
##   pays      x2015  x2016  x2017  x2018  x2019  
##   <chr>     <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  
## 1 Belgium  122274 121896 119690 118319 117695  
## 2 Bulgaria  65950  64984  63955  62197  61538  
## 3 Czechia  110764 112663 114405 114036 112231  
## 4 Denmark   58205  61614  61397  61476  61167  
## 5 Germany  737575 792141 784901 787523 778090  
##   x2020  x2021  
##   <dbl>  <dbl>  
## 1 114350 118349  
## 2 59086  58678  
## 3 110200 111793  
## 4 60937  63473  
## 5 773144 795492
```



Les données sont dans un format large. Convertissez-le en un jeu de données au format long qui a les noms de colonnes suivants : “pays”, “annee” et “nombre\_de\_naissances”

```
naissance_long %>% # complétez le code avec votre réponse
```

```
[ez votre réponse  
_euro_naissance_long()
```

```
## Vous n'avez pas encore défini l'objet de réponse,  
`_Q_euro_naissance_long`.  
## 1 ▷2◀ 3 4 5
```

```
_euro_naissance_long()
```

```
##  
## INDICE.  
## Votre réponse devrait ressembler à cela :  
##
```



```
##    naissances_euro_large %>%
##      pivot_longer(COLS A PIVOTER,
##                    names_to = NOM_COL,
##                    values_to = VALEUR_COL)
```

## Pivoter de long à large

Maintenant que vous savez comment pivoter de large à long avec `pivot_longer()`. Comment faire l'inverse, de long à large ? Pour cela, vous pouvez utiliser la fonction bien nommée `pivot_wider()`.

Mais avant de considérer comment utiliser cette fonction pour manipuler des données longues, considérons d'abord où vous êtes susceptible de rencontrer des données longues.

Alors que les données larges proviennent généralement de sources externes (comme nous l'avons vu ci-dessus), les données longues, en revanche, sont susceptibles d'être créées par vous lors de la manipulation des données, notamment lors des manipulations `group_by()-summarize()`.

Voyons un exemple maintenant.

Nous utiliserons un jeu de données de dossiers de patients provenant d'une épidémie d'Ebola en Sierra Leone en 2014. Ci-dessous, nous extrayons ces données du package `{outbreaks}` et effectuons quelques manipulations de simplification.

```
outbreaks::ebola_sierraleone_2014 %>%
  select(-date_of_onset) %>%
  mutate(annee = lubridate::year(date_of_onset)) %>% # extraire l'année de la date
  select(id_patient, district, annee_de_debut = annee) # sélectionner et renommer
```

```
## # A tibble: 5 × 3
##   id_patient district annee_de_debut
##       <int> <fct>          <dbl>
## 1 1         Kailahun 2014
## 2 2         Kailahun 2014
## 3 3         Kailahun 2014
## 4 4         Kailahun 2014
## 5 5         Kailahun 2014
```

Chaque ligne correspond à un patient, et nous avons le numéro d'identifiant de chaque patient, leur district et l'année où ils ont contracté Ebola.

Maintenant, considérez la résumé groupé suivant du jeu de données `ebola`, qui compte le nombre de patients enregistrés dans chaque district chaque année :

```
district_par_annee <-  
%>%  
by(district) %>%  
(annee_de_debut) %>%  
ip()  
  
district_par_annee
```

```
## # A tibble: 5 × 3  
##   district annee_de_debut     n  
##   <fct>          <dbl> <int>  
## 1 Bo            2014    397  
## 2 Bo            2015    209  
## 3 Bombali      2014   1070  
## 4 Bombali      2015    120  
## 5 Bonthe       2014     7
```

La sortie de cette opération groupée est un jeu de données typiquement “long” ! Chaque unité d’observation (chaque district) occupe plusieurs lignes (deux lignes par district, pour être exact), avec une ligne pour chaque mesure (chaque année).

Ainsi, comme vous le voyez maintenant, les données longues peuvent souvent arriver comme une sortie de résumés groupés, entre autres manipulations de données.

Maintenant, voyons comment convertir de telles données longues en un format large avec `pivot_wider()`.

Le code est assez simple :

```
district_par_annee %>%  
wider(values_from = n,  
      names_from = annee_de_debut)
```

```
## # A tibble: 5 × 3  
##   district `2014` `2015`  
##   <fct>    <int> <int>  
## 1 Bo        397    209  
## 2 Bombali  1070    120  
## 3 Bonthe   7       77  
## 4 Kailahun 535    35  
## 5 Kambia   127    294
```

Comme vous pouvez le voir, `pivot_wider()` a deux arguments importants : `values_from` et `names_from`. L'argument `values_from` définit quelles valeurs deviendront le cœur du format de données large (en d'autres termes : quel vecteur 1D deviendra une matrice 2D). Dans notre cas, ces valeurs étaient dans la variable `n`. Et `names_from` identifie quelle variable utiliser pour définir les noms de colonnes dans le format large. Dans notre cas, c'était la variable `annee_debut`.

Vous voudrez peut-être aussi faire des *années* votre unité d'observation/statistique principale, avec chaque année occupant une ligne. Cela peut être réalisé de manière similaire à l'exemple ci-dessus, mais la variable `district` sera fournie en argument à `names_from`, au lieu de `annee_debut`.

```
district_par_annee %>%  
  wider(values_from = n,  
        names_from = district)
```

#### SIDE NOTE



```
## # A tibble: 2 × 15  
##   annee_debut     Bo Bombali Bonthe Kailahun  
##             <dbl> <int> <int> <int> <int>  
## 1           2014    397   1070     7   535  
## 2           2015    209    120    77   35  
##   Kambia Kenema Koinadugu Kono Moyamba  
##   <int> <int> <int> <int> <int>  
## 1     127    641     142    328    258  
## 2     294    139      15    223    191  
## # i 5 more variables: `Port Loko` <int>, Pujehun <int>,  
## # Tonkolili <int>, `Western Rural` <int>, ...
```

Ici, les unités d'observation uniques (nos lignes) sont maintenant les années (2014, 2015).

#### PRACTICE (in RMD)

L'ensemble de données `population` du package `tidyverse` montre les populations de 219 pays au fil du temps.

Pivoter ces données en un format large. Votre réponse devrait comprendre 20 colonnes et 219 lignes.

```
ation_large <-  
:population %>% # CODE ICI
```

```
ez votre réponse  
population_large()
```

```
## Vous n'avez pas encore défini l'objet de réponse,  
'Q_population_large'.  
## 1 2 ▷3◁ 4 5
```

## PRACTICE



(in RMD)

```
population_large()
```

```
##  
## INDICE.  
## Votre réponse devrait ressembler à cela :  
##  
## tidyverse::population %>%  
##   pivot_wider(names_from = COLONNES_FUTURE_NOMS,  
##                 values_from = COLONNES_PIVOTER)
```

## Pourquoi les données au format long sont-elles meilleures pour l'analyse?

Ci-dessus, nous avons mentionné que les données au format long sont les meilleures pour la majorité des tâches d'analyse de données. Maintenant, nous pouvons justifier pourquoi. Dans les sections ci-dessous, nous passerons en revue quelques opérations courantes que vous devrez effectuer avec des données au format long. Dans chaque cas, vous observerez que les manipulations similaires sur des données larges seraient assez délicates.

### Filtrer les données groupées

Tout d'abord, parlons du filtrage des données groupées, qui est très facile à faire sur des données au format long, mais difficile sur des données au format large.

Voici un exemple avec le jeu de données sur la mortalité infantile. Imaginez que nous voulons répondre à la question suivante : **Pour chaque pays, quelle année a connu le plus grand nombre de décès d'enfants ?**

Voici comment nous le ferions avec le format long des données :

```
infants_long %>%
  by(pays) %>%
  filter(nombre_deces == max(nombre_deces))
```

```
## # A tibble: 5 × 3
## # Groups:   pays [5]
##   pays           annee nombre_deces
##   <chr>          <chr>      <dbl>
## 1 Afghanistan  x2010      74600
## 2 Angola        x2010      79100
## 3 Albania       x2010      420
## 4 United Arab Emirates x2011      687
## 5 Argentina     x2010      9550
```

Facile, n'est-ce pas ? Nous pouvons facilement voir, par exemple, que l'Afghanistan a eu son plus haut taux de mortalité infantile en 2010, et les Émirats arabes unis en 2011.

Si vous vouliez faire la même chose avec des données au format large, ce serait beaucoup plus difficile. Vous pourriez essayer une approche comme celle-ci avec `rowwise()` :

```
infants_large %>%
  rowwise() %>%
  mutate(max_decompte = max(x2010, x2011, x2012, x2013, x2014, x2015))
```

```
## # A tibble: 5 × 8
## # Rowwise:
##   pays           x2010 x2011 x2012 x2013
##   <chr>          <dbl> <dbl> <dbl> <dbl>
## 1 Afghanistan  74600  72000  69500  67100
## 2 Angola        79100  76400  73700  71200
## 3 Albania       420    384    354    331
## 4 United Arab Emirates  683    687    686    681
## 5 Argentina     9550   9230   8860   8480
##   x2014 x2015 max_decompte
##   <dbl> <dbl>      <dbl>
## 1 64800 62700      74600
## 2 69000 67200      79100
## 3 313   301       420
## 4 672   658       687
## 5 8100  7720      9550
```

Cela fonctionne presque - nous avons, pour chaque pays, le nombre maximum de décès d'enfants rapportés - mais nous ne savons toujours pas à quelle année est attachée cette valeur dans `max_decompte`. Nous devrions prendre cette valeur et l'indexer à sa colonne d'année respective d'une manière ou d'une autre... quelle corvée ! Il y a des solutions pour trouver cela mais toutes sont très pénibles.

Pourquoi rendre votre vie compliquée quand vous pouvez simplement pivoter vers le format long et utiliser la beauté de `group_by()` et `filter()` ?

Ici, nous avons utilisé une fonction spéciale de `{dplyr}` : `rowwise()`. `rowwise()` permet d'appliquer d'autres opérations *par ligne*. Il est équivalent à la création d'un groupe pour chaque ligne (`group_by(row_number())`).

Sans `rowwise()`, vous obtiendriez ceci :

```
infants_large %>%  
  summarise(max_decompte = max(x2010, x2011, x2012, x2013, x2014, x2015))
```

**SIDE NOTE**



```
## # A tibble: 5 × 8  
##   pays      x2010 x2011 x2012 x2013  
##   <chr>     <dbl>  <dbl>  <dbl>  <dbl>  
## 1 Afghanistan 74600  72000  69500  67100  
## 2 Angola      79100  76400  73700  71200  
## 3 Albania      420    384    354    331  
## 4 United Arab Emirates  683    687    686    681  
## 5 Argentina    9550   9230   8860   8480  
## # A tibble: 5 × 3  
##   pays      x2014 x2015 max_decompte  
##   <chr>     <dbl>  <dbl>        <dbl>  
## 1 Afghanistan 64800  62700       1170000  
## 2 Angola      69000  67200       1170000  
## 3 Albania      313    301       1170000  
## 4 United Arab Emirates  672    658       1170000  
## 5 Argentina    8100   7720       1170000
```

... le compte maximum sur TOUTES les lignes de l'ensemble de données.

**PRACTICE**



Pour cette question de pratique, vous effectuerez un filtre groupé sur le jeu de données `population` en format long du package `tidyverse`. Utilisez `group_by()` et `filter()` pour obtenir un ensemble de données qui montre la population maximale enregistrée pour chaque pays, et l'année où cette population maximale a été enregistrée.

```
population_max <-  
  population %>% # CODE ICI
```

```
ez votre réponse  
population_max()
```

```
## Vous n'avez pas encore défini l'objet de réponse,  
`Q_population_max`.  
## 1 2 3 ▷4◁ 5
```

### PRACTICE



(in RMD)

```
population_max()
```

```
##  
## INDICE.  
## Votre réponse devrait ressembler à cela :  
##  
## tidyverse::population %>%  
##   group_by(country) %>%  
##     FILTRE_POUR_MAX_POPULATION_PAR_GROUPE  
##   ungroup()
```

## Résumer les données regroupées

Les résumés regroupés sont également difficiles à réaliser sur des données au format large. Par exemple, en considérant à nouveau le jeu de données `deces_enfants_long`, si vous voulez demander : **Pour chaque pays, quel était le nombre moyen de décès de nourrissons et l'écart-type (variation) des décès ?**

Avec des données au format long, c'est simple :

```
nfants_long %>%  
  by(pays) %>%  
  size(moyen_deces = mean(nombre_deces),  
       sd_deces = sd(nombre_deces))
```

```
## # A tibble: 5 × 3  
##   pays           moyen_deces    sd_deces  
##   <chr>          <dbl>      <dbl>  
## 1 Afghanistan    68450.    4466.  
## 2 Albania         350.     45.2  
## 3 Algeria        21033.    484.  
## 4 Angola          72767.   4513.  
## 5 Antigua and Barbuda    10.7    0.816
```

Avec des données au format large, par contre, trouver la moyenne est moins intuitif...

```

infants_large %>%
  se() %>%
  `moyen_deces` = sum(x2010, x2011, x2012,
    x2013, x2014, x2015, na.rm = T) / 6)

```

```

## # A tibble: 5 × 8
## # Rowwise:
##   pays           x2010 x2011 x2012 x2013
##   <chr>          <dbl>  <dbl>  <dbl>  <dbl>
## 1 Afghanistan  74600  72000  69500  67100
## 2 Angola        79100  76400  73700  71200
## 3 Albania       420    384    354    331
## 4 United Arab Emirates 683    687    686    681
## 5 Argentina     9550   9230   8860   8480
## # ... with row labels `x2014` and `x2015` and
## # ... values `moyen_deces`:
## #   <dbl> <dbl> <dbl>
## # 1 64800 62700 68450
## # 2 69000 67200 72767.
## # 3 313   301   350.
## # 4 672   658   678.
## # 5 8100  7720  8657.

```

Et trouver l'écart-type serait très difficile. (Nous ne pouvons penser à aucune façon d'y parvenir, en fait.)

Pour cette question de pratique, vous travaillerez à nouveau avec le jeu de données `population` en format long du package `tidyverse`.

Utilisez `group_by()` et `summarize()` pour obtenir, pour chaque pays, la population maximale signalée, la population minimale signalée, et la population moyenne signalée sur les années disponibles dans les données. Vos données devraient avoir quatre colonnes, “country”, “max\_population”, “min\_population” et “moyen\_population”.

### PRACTICE



```

population_resumer <-
  population %>% # CODE ICI

```

```

lez votre réponse
Q_population_resumer()

```

```

## Vous n'avez pas encore défini l'objet de réponse,
`Q_population_resumer`.
## 1 2 3 4 ▷5◀

```



```
population_resumer()
```

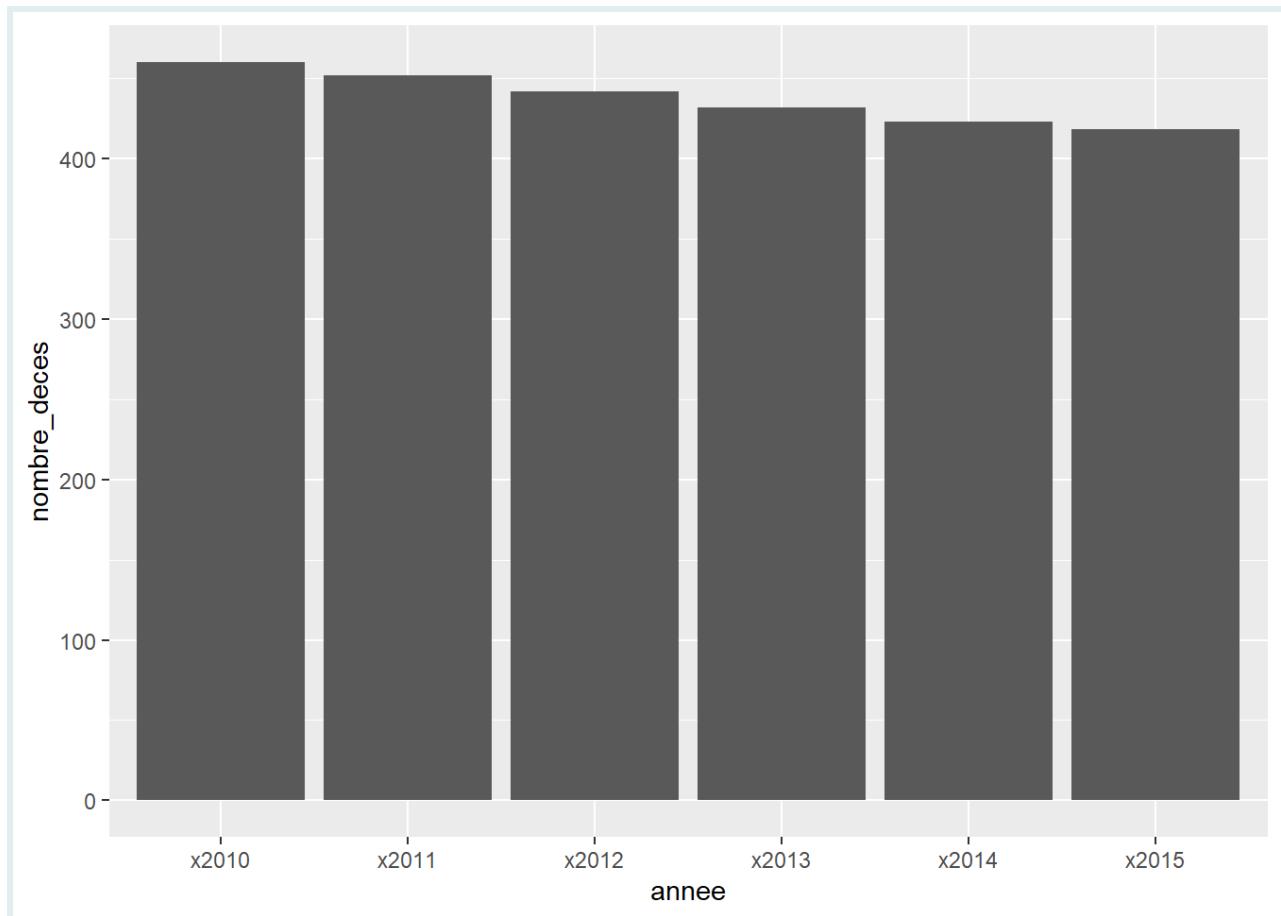
```
## INDICE.  
## Votre réponse devrait ressembler à cela :  
## population %>%  
## group_by(country) %>%  
## summarise(max_population = ,  
##            min_population = ,  
##            mean_population = )
```

## Conception des graphiques

Enfin, l'une des tâches d'analyse de données qui est LE PLUS entravée par les formats larges est la conception de graphiques. Vous n'avez peut-être pas encore de connaissance préalable de {ggplot} et de comment concevoir des graphiques, nous verrons donc les figures sans approfondir le code. Ce que vous devez retenir est : de nombreux graphiques avec ggplot sont également possibles uniquement avec des données au format long.

Considérez à nouveau les données sur les décès de nourrissons `deces_enfants_long`. Nous allons représenter le nombre de décès pour la Belgique par année :

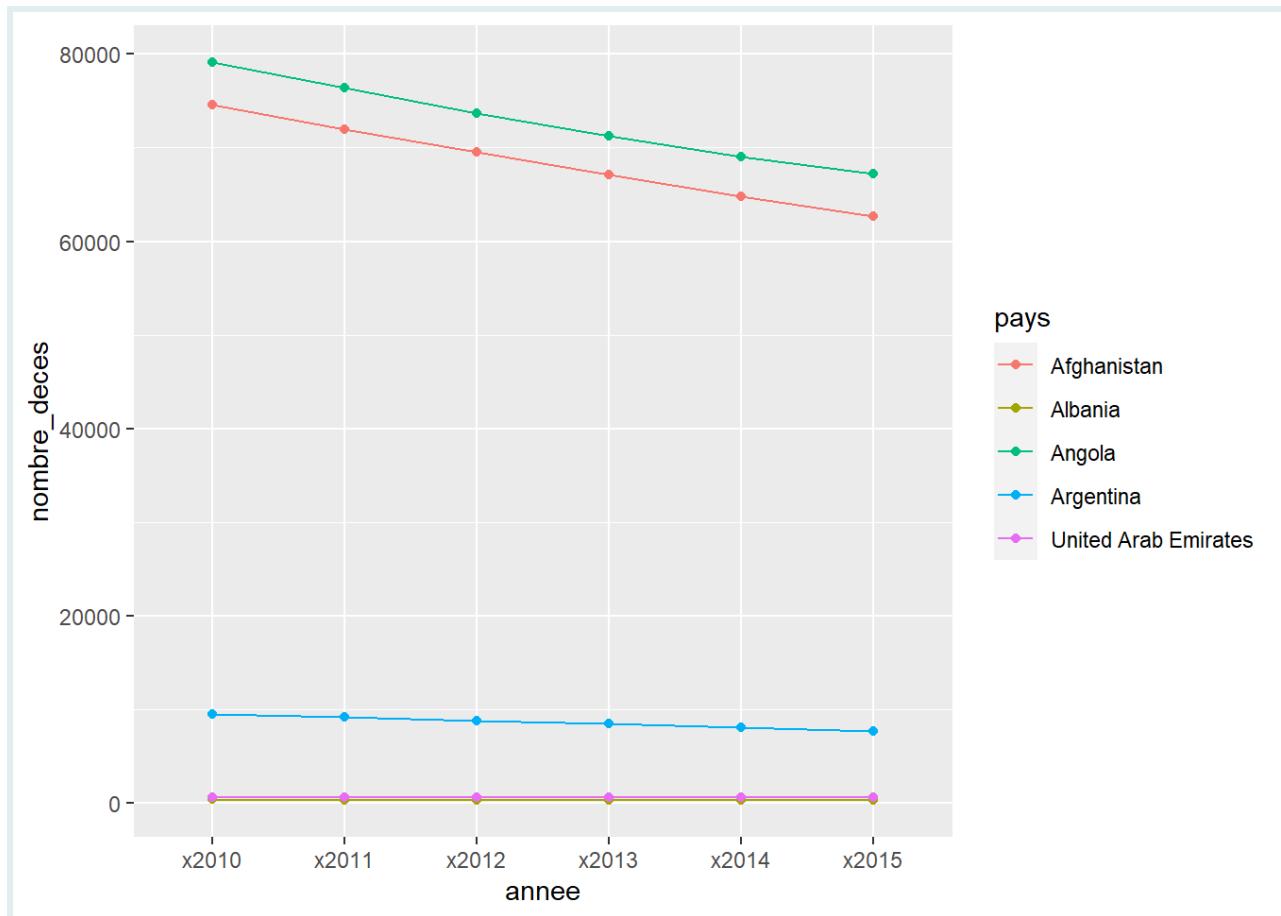
```
enfants_long %>%  
  filter(pays == "Belgium") %>%  
  group_by(annee) +  
  summarise(nombre_deces = sum(nombre_deces))
```



La conception du graphique fonctionne parce que nous pouvons donner la variable `annee` pour l'axe des x. Dans le format long, `annee` est une variable à part entière. Dans le format large, il n'y aurait pas une telle variable à passer à l'axe des x.

Un autre graphique qui ne serait pas possible sans un format long :

```
nfants_long %>%
  aes(x = annee, y = nombre_deces, group = pays, color = pays)) +
  line() +
  point()
```



Encore une fois, la raison est la même, nous devons indiquer au graphique ce qu'il faut utiliser comme axe des x et des y et il est nécessaire d'avoir ces variables dans leurs propres colonnes (comme organisé dans le format long).

## Le pivot peut être difficile

Nous avons principalement examiné ici des exemples très simples de pivot, mais dans la pratique, le pivot peut être très difficile à réaliser avec précision. C'est parce que les données avec lesquelles vous travaillez peuvent ne pas avoir toutes les informations nécessaires pour un pivot réussi, ou les données peuvent contenir des erreurs qui vous empêchent de pivoter correctement.

Lorsque vous rencontrez de tels cas, nous vous recommandons de consulter la [documentation officielle](#) du pivot de l'équipe `tidyverse`, car elle est assez riche en exemples. Vous pourriez également poster vos questions sur le pivot sur des forums comme Stack Overflow.

---

## Conclusion !

Vous avez maintenant exploré différents jeux de données et comment ils sont soit en format long, soit en format large. En fin de compte, il s'agit simplement de la façon dont vous présentez l'information. Parfois, un format sera plus pratique, et d'autres fois un autre pourrait être le meilleur. Maintenant, vous n'êtes plus limité par le format de vos données : ça ne vous plaît pas ? changez-le !

---

## Contributeurs

Les membres de l'équipe suivants ont contribué à cette leçon :



### KENE DAVID NWOSU

Data analyst, the GRAPH Network  
Passionate about world improvement

---



### LAURE VANCAUWENBERGHE

Data analyst, the GRAPH Network  
A firm believer in science for good, striving to ally programming, health and education

---



### CAMILLE BEATRICE VALERA

Project Manager and Scientific Collaborator, The GRAPH Network

---

This work is licensed under the [Creative Commons Attribution Share Alike](#) license.



### Solution exercice pratique 5

```
%>%  
by(country) %>%  
  use(max_population = max(population),  
      min_population = min(population),  
      moyen_population = mean(population))
```

```
## # A tibble: 5 × 4  
##   country       max_population min_population  
##   <chr>          <dbl>           <dbl>  
## 1 Afghanistan     30551674     17586073
```

```
## 2 Albania          3357858      3150143
## 3 Algeria          39208194      29315463
## 4 American Samoa   59262        52874
## 5 Andorra           81877        63799
##   moyen_population
##                   <dbl>
## 1             23960450.
## 2             3235417
## 3             33826305.
## 4             56705.
## 5             73485.
```

# Notes de cours | Pivot avancé

February 2024



Introduction	.
Objectifs d'apprentissage	.
Packages	.
Jeux de données	.
Du format large au format long	.
Comprendre <code>names_sep</code> et “ <code>.value</code> ”	.
Type de valeur <i>avant</i> le séparateur	.
Un exemple qui n'est pas une série temporelle	.
Echapper le séparateur de point	.
Que faire quand vous n'avez pas un séparateur net ?	.
Du format long au format large	.
Bilan !	.
Références	.

---

## Introduction

Vous connaissez les opérations de pivot de base des jeux de données du format long au format large et vice versa. Cependant, comme c'est souvent le cas, les manipulations de base ne sont pas suffisantes pour le traitement des données que vous devez faire. Voyons maintenant le niveau suivant. Allons-y !

---

## Objectifs d'apprentissage

1. Maîtriser le pivot complexe du format large au format long et du format long au format large
2. Savoir utiliser les séparateurs comme outil de pivot

---

## Packages

```
er les packages
aire(pacman)) install.packages("pacman")
p_load(tidyverse, outbreaks, janitor, rio, here, knitr)
```

---

## Jeux de données

Nous présenterons ces jeux de données au fur et à mesure, mais voici un aperçu :

- Données d'enquête d'une étude menée en Inde sur les dépenses des patients pour le traitement de la tuberculose
  - Données d'une étude sur les biomarqueurs des entéropathogènes en Zambie
  - Une enquête alimentaire au Vietnam
- 

## Du format large au format long

Parfois, vous avez plusieurs types de données au format large dans le même jeu de données. Considérez cet exemple factice de la taille et du poids des enfants sur deux ans :

```
stats <-  
  :::tribble(  
    ~enfant, ~annee1_taille, ~annee2_taille, ~annee1_poids, ~annee2_poids,  
    'A', "80cm", "85cm", "5kg", "10kg",  
    'B', "85cm", "90cm", "7kg", "12kg",  
    'C', "90cm", "100cm", "6kg", "14kg"  
  
stats
```

```
## # A tibble: 3 × 5  
##   enfant annee1_taille annee2_taille  
##   <chr>     <chr>      <chr>  
## 1 A         80cm        85cm  
## 2 B         85cm        90cm  
## 3 C         90cm       100cm  
##   annee1_poids annee2_poids  
##   <chr>      <chr>  
## 1 5kg        10kg  
## 2 7kg        12kg  
## 3 6kg        14kg
```

Si vous pivotez toutes les colonnes des mesures, vous obtiendrez des données trop longues :

```
stats %>%  
  longer(2:5)
```

```
## # A tibble: 5 × 3  
##   enfant name      value  
##   <chr>  <chr>      <chr>  
## 1 A      annee1_taille 80cm  
## 2 A      annee2_taille 85cm  
## 3 A      annee1_poids  5kg
```

```
## 4 A      annee2_poids 10kg
## 5 B      annee1_taille 85cm
```

Ce n'est (généralement) pas ce que nous recherchons, car maintenant vous avez deux données différentes dans la même colonne- le poids et la taille.

Pour obtenir le bon format, vous devez utiliser l'argument `names_sep` et l'identifiant `".value"` :

```
stats %>%
  longer(2:5,
         names_sep = "_",
         names_to = c("periode", ".value"))
```

```
## # A tibble: 5 × 4
##   enfant periode taille poids
##   <chr>   <chr>   <chr>  <chr>
## 1 A       annee1  80cm   5kg
## 2 A       annee2  85cm   10kg
## 3 B       annee1  85cm   7kg
## 4 B       annee2  90cm   12kg
## 5 C       annee1  90cm   6kg
```

Maintenant, nous avons une ligne pour chaque combinaison enfant-période, un format long correct !

Ce que fait le code ci-dessus peut ne pas être clair, mais vous devriez déjà pouvoir répondre à l'exercice ci-dessous en reproduisant la syntaxe de l'exemple précédent. Après cet exercice , nous expliquerons l'argument `names_sep` et l'identifiant `".value"` plus en détail.



Considérez cet autre ensemble de données factice :

```
stats <-
  ::tribble(
    ~alte, ~annee1_IMC, ~annee2_IMC, ~annee1_VIH, ~annee2_VIH,
    "A", 25, 30, "Positive", "Positive",
    "B", 34, 28, "Negative", "Positive",
    "C", 19, 17, "Negative", "Negative"
  )
stats
```

```
## # A tibble: 3 × 5
##   adulte annee1_IMC annee2_IMC annee1_VIH
```

```

## 2 B           34      28 Negative
## 3 C           19      17 Negative
##   annee2_VIH
##   <chr>
## 1 Positive
## 2 Positive
## 3 Negative

```

Pivotez les données en un format long pour obtenir la structure suivante :

**adulte annee IMC VIH**

```

`_long <-
`_stats %>%
longer(______)

```

```

ez votre réponse
)`_adulte_long()

```

### PRACTICE



(in RMD)

```

## Vous n'avez pas encore défini l'objet de réponse,
`Q_adulte_long`.
## ▷1◁ 2 3 4 5 6

```

```

`adulte_long()

```

```

## 
## INDICE.
## 
## Utilisez la fonction `pivot_longer` du package
`tidyverse`.
## - Spécifiez `cols = 2:5` pour sélectionner les colonnes à
pivoter.
## - Utilisez `names_sep = "_"` et `names_to = c("year",
".value")` pour formater les noms.
## 

```

obtenir la solution, exécutez la ligne ci-dessous !  
`N\_Q\_adulte\_long()

```

## 
## SOLUTION
## 
##   adulte_stats %>%
##     pivot_longer(cols = 2:5,

```

```
##             names_sep = "_",
##             names_to = c("annee", ".value"))
```

### PRACTICE



La question a une fonction solution similaire à celle-ci.  
INT est remplacé par SOLUTION dans le nom de la fonction.)  
vous devrez taper le nom de la fonction par vous-même.  
vise à vous dissuader de regarder la solution avant de répondre  
à la question.)

L'exemple ci-dessus enfant\_stats a des nombres stockés en tant que caractères [...]

Comme vous l'avez vu dans la leçon précédente, vous pouvez facilement extraire les nombres à partir du jeux de données de sortie au format long en utilisant la fonction `parse_number()` de `readr` :

```
stats_long <-
  stats %>%
  longer(2:5,
         names_sep = "_",
         names_to = c("periode", ".value"))

stats_long
```

### SIDE NOTE



```
## # A tibble: 5 × 4
##   enfant periode taille poids
##   <chr>    <chr>   <chr>  <chr>
## 1 A        annee1  80cm   5kg
## 2 A        annee2  85cm   10kg
## 3 B        annee1  85cm   7kg
## 4 B        annee2  90cm   12kg
## 5 C        annee1  90cm   6kg
```

```
stats_long %>%
  `#(taille = parse_number(taille),
  poids = parse_number(poids))`
```

```
## # A tibble: 5 × 4
##   enfant periode taille poids
##   <chr>    <chr>   <dbl> <dbl>
## 1 A        annee1     80      5
## 2 A        annee2     85      10
## 3 B        annee1     85      7
```

**SIDE NOTE**

```
## 4 B      annee2      90     12
## 5 C      annee1      90      6
```

## Comprendre `names_sep` et “`.value`”

Maintenant, décomposons l'appel `pivot_longer()` que nous avons vu ci-dessus :

```
stats
```

```
## # A tibble: 3 × 5
##   enfant annee1_taille annee2_taille
##   <chr>    <chr>          <chr>
## 1 A        80cm           85cm
## 2 B        85cm           90cm
## 3 C        90cm           100cm
## # ... with 2 more variables: annee1_poids <chr>,
## #   annee2_poids <chr>
## #   1 5kg          10kg
## #   2 7kg          12kg
## #   3 6kg          14kg
```

```
stats %>%
  longer(2:5,
         names_sep = "_",
         names_to = c("periode", ".value"))
```

```
## # A tibble: 5 × 4
##   enfant periode taille poids
##   <chr>    <chr>  <chr> <chr>
## 1 A        annee1  80cm  5kg
## 2 A        annee2  85cm  10kg
## 3 B        annee1  85cm  7kg
## 4 B        annee2  90cm  12kg
## 5 C        annee1  90cm  6kg
```

Remarquez que les noms de colonnes dans le dataframe `enfant_stats` d'origine (`annee1_taille`, `annee2_taille` etc.) sont composés de trois parties :

- la période référencée : par exemple “`annee1`”
- un séparateur de soulignement, “`_`”;
- et le type de valeur enregistrée “`taille`” ou “`poids`”

Nous pouvons faire un tableau avec ces parties :

nom_colonne	periode	separateur “.value”
annee1_taille	annee1	taille
annee2_taille	annee2	taille
annee1_poids	annee1	poids
annee2_poids	annee2	poids

Sur la base de ce tableau, il devrait maintenant être plus facile de comprendre les arguments `names_sep` et `names_to` que nous avons fournis à `pivot_longer()` :

```
names_sep = "_":
```

C'est le séparateur entre l'indicateur de période (année) et les valeurs (taille et poids) enregistrées.

Si nous utilisons un séparateur différent, l'argument va aussi changer. Par exemple, si le séparateur est un espace vide, "", vous aurez `names_sep = " "`, comme on le voit dans l'exemple ci-dessous :

```
stats_espace_sep <-
:::tribble(
!ant, `~`ann1 taille`, `~`ann2 taille`, `~`ann1 poids`, `~`ann2 poids`,
'A',      "80cm",           "85cm",           "5kg",          "10kg",
'B',      "85cm",           "90cm",           "7kg",          "12kg",
'C',      "90cm",           "100cm",          "6kg",          "14kg"

stats_espace_sep %>%
longer(2:5,
  names_sep = " ",
  names_to = c("periode", ".value"))
```

```
## # A tibble: 5 × 4
##   enfant periode taille poids
##   <chr>    <chr>  <chr> <chr>
## 1 A        ann1    80cm   5kg
## 2 A        ann2    85cm   10kg
## 3 B        ann1    85cm   7kg
## 4 B        ann2    90cm   12kg
## 5 C        ann1    90cm   6kg
```

```
names_to = c("periode", ".value")
```

Ensuite, l'argument `names_to` indique comment les données doivent être restructurées. Nous avons passé un vecteur de deux chaînes de caractères, "periode" et ".value" à cet argument. Voyons le rôle de chaque élément :

**La chaîne "periode"** indique que nous voulons placer les données de chaque année (ou période) dans une ligne séparée. Notez qu'il n'y a rien de spécial dans le

mot “periode” utilisé ici ; nous pourrions changer cela par n’importe quelle autre chaîne. Donc, au lieu de “periode”, vous auriez pu écrire “temps” ou “annee\_de\_mesure” ou autre chose :

```
stats %>%
  longer(2:5,
         names_sep = " _ ",
         names_to = c("annee_de_mesure", ".value"))
```

```
## # A tibble: 5 × 4
##   enfant annee_de_mesure taille poids
##   <chr>    <chr>      <chr>  <chr>
## 1 A        anneel       80cm   5kg
## 2 A        annee2       85cm   10kg
## 3 B        anneel       85cm   7kg
## 4 B        annee2       90cm   12kg
## 5 C        anneel       90cm   6kg
```

Maintenant, le placeholder “.value” est un indicateur spécial, qui indique à `pivot_longer()` de créer une colonne séparée pour chaque valeur distincte qui apparaît après le séparateur. Dans notre exemple, ces valeurs sont “taille” et “poids”.

La chaîne “.value” ne peut pas être remplacée arbitrairement. Par exemple, ceci ne fonctionnera pas :

```
stats %>%
  longer(2:5,
         names_sep = " _ ",
         names_to = c("periode", "valeurs"))
```

```
## # A tibble: 5 × 4
##   enfant periode valeurs value
##   <chr>    <chr>    <chr>  <chr>
## 1 A        anneel   taille  80cm
## 2 A        annee2   taille  85cm
## 3 A        anneel   poids   5kg
## 4 A        annee2   poids   10kg
## 5 B        anneel   taille  85cm
```

---

Autrement dit, le placeholder “.value” indique à `pivot_longer()` que nous voulons séparer les valeurs “taille” et “poids” dans deux colonnes séparées, car nous avons deux types de valeurs après le séparateur “\_” dans les noms de colonnes.

Cela signifie que si vous avez un jeu de données au format large avec trois types de valeurs, vous obtiendrez trois colonnes séparées, une pour chaque type de valeur. Par exemple, considérez le jeu de données fictif ci-dessous qui montre les enregistrements d’enfants, à deux moments, pour les variables suivantes :

- âge en mois,
- pourcentage de graisse corporelle
- IMC

```
stats_trois_valeurs <-
:::tribble(
, ~t1_age, ~t2_age, ~t1_graisse, ~t2_graisse, ~t1_imc, ~t2_imc,
, "5 mois", "8 mois", "13%", "15%", 14, 15,
, "7 mois", "9 mois", "15%", "17%", 16, 18

stats_trois_valeurs
```

```
## # A tibble: 2 × 7
##   enfant t1_age t2_age t1_graisse t2_graisse
##   <chr>    <chr>    <chr>    <chr>
## 1 a        5 mois  8 mois  13%      15%
## 2 b        7 mois  9 mois  15%      17%
##   t1_imc t2_imc
##   <dbl>   <dbl>
## 1     14     15
## 2     16     18
```

Ici, dans les noms de colonnes, il y a trois types de valeurs qui apparaissent après le séparateur “\_” : age, graisse et imc; la chaîne “.value” indique à `pivot_longer()` de créer une nouvelle colonne pour chaque type de valeur :

```
stats_trois_valeurs %>%
longer(2:7,
       names_sep = "-",
       names_to = c("temps", ".value")
     )
```

```
## # A tibble: 4 × 5
##   enfant temps age     graisse   imc
##   <chr>   <chr> <chr>    <chr>    <dbl>
## 1 a       t1    5 mois  13%      14
## 2 a       t2    8 mois  15%      15
## 3 b       t1    7 mois  15%      16
## 4 b       t2    9 mois  17%      18
```

Un pédiatre enregistre les informations suivantes pour un ensemble d'enfants sur deux ans :

- périmètre crânien ;
- circonférence du cou ; et
- tour de hanches

le tout en centimètres.

Voici le tableau de sortie :

```
practice_stats <-  
  ::tribble(  
    ~fant, ~ann1_tete, ~ann2_tete, ~ann1_cou, ~ann2_cou, ~ann1_hanche, ~ann2_hanche  
    'a', 45, 48, 23, 24, 51,  
    52,  
    'b', 48, 50, 24, 26, 52,  
    52,  
    'c', 50, 52, 24, 27, 53,  
    54  
  
practice_stats
```



```
## # A tibble: 3 × 7  
##   enfant ann1_tete ann2_tete ann1_cou ann2_cou  
##   <chr>     <dbl>     <dbl>     <dbl>     <dbl>  
## 1 a         45        48        23        24  
## 2 b         48        50        24        26  
## 3 c         50        52        24        27  
##   ann1_hanche ann2_hanche  
##   <dbl>       <dbl>  
## 1          51          52  
## 2          52          52  
## 3          53          54
```

Pivotez les données en un format long pour obtenir la structure suivante :

enfant	annee	tete	cou	hanche
--------	-------	------	-----	--------

```
sance_stats_long <-  
  sance_stats %>%  
  longer(_____)
```

```
ez votre réponse  
Q_croissance_stats_long()
```

```
## Vous n'avez pas encore défini l'objet de réponse,  
`Q_croissance_stats_long`.  
## 1 ▷2◁ 3 4 5 6
```

## PRACTICE



(in RMD)

```
croissance_stats_long()
```

```
##  
## INDICE.  
##  
## Utilisez la fonction `pivot_longer` du package  
`tidyverse`.  
## - Spécifiez `cols = 2:7` pour sélectionner les colonnes à  
pivoter.  
## - Utilisez `names_to = c("year", ".value")` et `names_sep  
= "_"` pour formater les noms.  
##
```

## Type de valeur avant le séparateur

Dans tous les exemples que nous avons utilisés jusqu'à présent, les noms de colonnes étaient construits de telle sorte que le type de valeur venait après le séparateur. Rappelez-vous notre tableau :

nom_colonne	periode	separateur	".value"
annee1_taille	annee1	_	taille
annee2_taille	annee2	_	taille
annee1_poids	annee1	_	poids
annee2_poids	annee2	_	poids

Mais bien sûr, les noms de colonnes pourraient être construits différemment, avec les types de valeurs venant avant le séparateur, comme dans cet exemple :

```
stats2 <-  
tribble(  
!ant, ~taille_annee1, ~taille_annee2, ~poids_annee1, ~poids_annee2,  
"A", "80cm", "85cm", "5kg", "10kg",  
"B", "85cm", "90cm", "7kg", "12kg",  
"C", "90cm", "100cm", "6kg", "14kg"  
  
stats2
```

```

## # A tibble: 3 × 5
##   enfant taille_annee1 taille_annee2
##   <chr>    <chr>      <chr>
## 1 A        80cm       85cm
## 2 B        85cm       90cm
## 3 C        90cm      100cm
##   poids_annee1 poids_annee2
##   <chr>      <chr>
## 1 5kg       10kg
## 2 7kg       12kg
## 3 6kg       14kg

```

Ici, les types de valeurs (taille et poids) viennent avant le “\_” séparateur.

Comment notre commande `pivot_longer()` peut-elle s'adapter à cela ? C'est simple ! Il suffit d'inverser l'ordre du vecteur donné à l'argument `names_to` :

Donc, au lieu de `names_to = c("temps", ".value")`, vous aurez `names_to = c(".value", "temps")` :

```

stats2 %>%
  longer(2:5,
         names_sep = "_",
         names_to = c(".value", "temps"))

```

```

## # A tibble: 5 × 4
##   enfant temps  taille poids
##   <chr>   <chr>  <chr> <chr>
## 1 A        annee1 80cm   5kg
## 2 A        annee2 85cm   10kg
## 3 B        annee1 85cm   7kg
## 4 B        annee2 90cm   12kg
## 5 C        annee1 90cm   6kg

```

Et voilà !

Considérez le jeu de données suivant de la Zambie sur les entéropathogènes et leurs biomarqueurs.



```

enteropathogenes_zambie_large<-
  read_csv(here("data/fr_enteropathogenes_zambie_large.csv"))

enteropathogenes_zambie_large

```

```

## # A tibble: 5 × 7
##   ID LPS_1 LPS_2  LBP_1 LBP_2 IFABP_1
##   <chr> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 1     0.12  0.15  0.08  0.05  0.03
## 2 2     0.15  0.18  0.09  0.06  0.04
## 3 3     0.18  0.22  0.11  0.07  0.05
## 4 4     0.22  0.25  0.13  0.08  0.06
## 5 5     0.25  0.28  0.15  0.09  0.07

```

```

##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 1002  222. 390. 38414. 6840. 1294.
## 2 1003  181. NA 26888. NA 22.5
## 3 1004  257. 221. 49183. 5426. 0
## 4 1005    NA 369. NA 1938. 0
## 5 1006  275. NA 61758. NA 0
##   IFABP_2
##   <dbl>
## 1    610.
## 2     NA
## 3      0
## 4   1010.
## 5     NA

```

Ce jeu de données se compose des colonnes suivantes :



- LPS\_1 et LPS\_2 : niveau des lipopolysaccharides, mesuré par Pyrochrome LAL, en EU/mL
- LBP\_1 et LBP\_2 : niveau des protéines de liaison au LPS, en pg/mL
- IFABP\_1 et IFAPB\_2 : niveau des protéines de liaison aux acides gras de type intestinal, en pg/mL

Pivotez le jeu de données pour qu'il ressemble à la structure suivante :

ID	numero_echantillon	LPS	LBP	IFABP
----	--------------------	-----	-----	-------

```

>pathogenes_zambie_large <-
>pathogenes_zambie_large %>%
|longer(____)

```

```

ez votre réponse
Q_enteropathogenes_zambie_large()

```

```

## Vous n'avez pas encore défini l'objet de réponse,
`Q_enteropathogenes_zambie_large`.
##   1   2  ▷3◁  4   5   6

```

```

enteropathogenes_zambie_large()

```



```
##  
## INDICE.  
##  
## Utilisez la fonction `pivot_longer` du package  
`tidyverse` pour réaliser la pivotisation.  
## - Spécifiez les arguments `names_to = c(".value",  
"numero_echantillon")` et `names_sep = "_"`.  
##
```

## Un exemple qui n'est pas une série temporelle

Jusqu'à présent, nous avons utilisé des ensembles de données personne-période (séries temporelles) pour illustrer l'idée de pivots complexes avec plusieurs types de valeurs.

Mais comme nous l'avons mentionné, tous les jeux de données nécessitant une restructuration ne sont pas forcément des données de séries temporelles. Voyons un exemple rapide qui n'est pas une série temporelle.

Vous pourriez mesurer la taille (cm) et le poids (kg) d'une série de couples parentaux dans un tableau comme celui-ci :

```
stats <-  
:::tribble(  
e, ~pere_taille, ~pere_poids, ~mere_taille, ~mere_poids,  
1", 180, 80, 160, 70,  
2", 185, 90, 150, 76,  
3", 182, 93, 143, 78  
stats
```

```
## # A tibble: 3 × 5  
##   couple pere_taille pere_poids mere_taille  
##   <chr>     <dbl>      <dbl>      <dbl>  
## 1 a         180        80        160  
## 2 b         185        90        150  
## 3 c         182        93        143  
##   mere_poids  
##   <dbl>  
## 1 70  
## 2 76  
## 3 78
```

Ici, nous avons deux types de valeurs différents (poids et taille) pour chaque personne du couple.

Pour pivoter à une ligne par personne, nous aurons encore besoin des arguments `names_sep` et `names_to` :

```
stats %>%
  longer(2:5,
         names_sep = "_",
         names_to = c("personne", ".value"))
```

```
## # A tibble: 5 × 4
##   couple personne taille poids
##   <chr>   <chr>     <dbl> <dbl>
## 1 a       pere      180    80
## 2 a       mere      160    70
## 3 b       pere      185    90
## 4 b       mere      150    76
## 5 c       pere      182    93
```

Le séparateur est un trait de soulignement, “\_”, donc nous avons utilisé `names_sep = “_”` et comme les types de valeurs viennent après le séparateur, l’identifiant “`.value`” a été placé en deuxième dans l’argument `names_to`.

## Echapper le séparateur de point

Un cas spécial que vous pourriez rencontrer est un ensemble de données où le séparateur est un point.

```
stats_point_sep <-
  tribble(
    ~ant, ~annee1.taille, ~annee2.taille, ~annee1.poids, ~annee2.poids,
    'A',      "80cm",        "85cm",        "5kg",        "10kg",
    'B',      "85cm",        "90cm",        "7kg",        "12kg",
    'C',      "90cm",        "100cm",       "6kg",        "14kg"
```

```
stats_point_sep %>%
  longer(2:5,
         names_to = c("periode", ".value"),
         names_sep = "\\".)
```

```
## # A tibble: 5 × 4
##   enfant periode taille poids
##   <chr>   <chr>   <chr>   <chr>
## 1 A       annee1  80cm   5kg
## 2 A       annee2  85cm   10kg
## 3 B       annee1  85cm   7kg
## 4 B       annee2  90cm   12kg
## 5 C       annee1  90cm   6kg
```

Ici, nous avons utilisé la chaîne “\.” pour indiquer un point “.” parce que le “.” est un caractère spécial dans R qui dans certains cas doit être échappé.

Considérez à nouveau les données `adulte_stats` que vous avez vues ci-dessus. Maintenant, les noms des colonnes ont été légèrement modifiés.

```
stats_point_sep <-
:::tribble(
  ~`IMC.annee1`, ~`IMC.annee2`, ~`VIH.annee1`,
  ~`VIH.annee2`,
  'A',           25,      30,    "Positive", "Positive",
  'B',           34,      28,    "Negative", "Positive",
  'C',           19,      17,    "Negative", "Negative"

stats_point_sep
```



```
## # A tibble: 3 × 5
##   adulte IMC.annee1 IMC.annee2 VIH.annee1
##   <chr>     <dbl>     <dbl> <chr>
## 1 A          25        30  Positive
## 2 B          34        28  Negative
## 3 C          19        17  Negative
##   VIH.annee2
##   <chr>
## 1 Positive
## 2 Positive
## 3 Negative
```

Encore une fois, pivotez les données en un format long pour obtenir la structure suivante :

adulte anneé IMC VIH

```
adulte2_long <-
stats_point_sep %>%
longer(____)
```

```
ez votre réponse
adulte2_long()
```

```
## Vous n'avez pas encore défini l'objet de réponse,
`Q_adulte2_long`.
```

```
-adulste2_long()
```



```
##  
## INDICE.  
##  
## Utilisez la fonction `pivot_longer` du package  
`tidyverse`.  
## - Spécifiez `cols = 2:5` pour sélectionner les colonnes à  
pivoter.  
## - N'oubliez pas d'échapper correctement le caractère  
point en utilisant `names_sep = "\."`  
## et utilisez `names_to = c(".value", "year")` pour  
formater les noms.  
##
```

## Que faire quand vous n'avez pas un séparateur net ?

Parfois, vous n'avez pas de séparateur net.

Considérez ces données d'une enquête menée en Inde qui examine les dépenses des patients pour le traitement de la tuberculose :

```
:es <- read_csv(here("data/fr_india_tb_pathways_and_costs_data.csv")) %>%  
names() %>%  
:(id, premiere_visite_emplacement, premiere_visite_cout,  
deuxieme_visite_emplacement, deuxieme_visite_cout,  
troisieme_visite_emplacement, troisieme_visite_cout)  
:es
```

```
## # A tibble: 5 × 7  
##      id premiere_visite_e...¹ premiere_visite...²  
##   <dbl> <chr>                  <dbl>  
## 1 100202 GH                      0  
## 2 100396 Pvt. docto              1500  
## 3 100590 Pvt. docto              2000  
## 4 100687 Pvt. hospi             20000  
## 5 100784 Pvt. docto              1000  
##   deuxieme_visite_emplacem...³ deuxieme_visite...⁴  
##   <chr>                  <dbl>  
## 1 <NA>                      0  
## 2 Pvt. clini                 1000  
## 3 Pvt. docto                3000  
## 4 Pvt. hospi                1500  
## 5 GH                        0
```

```
## # i abbreviated names: `¹premiere_visite_emplacement,
## #   `²premiere_visite_cout, `³deuxieme_visite_emplacement, ...
```

Il n'y a pas de séparateur net entre les indicateurs de temps (premier, deuxième, troisième) et le type de valeur (cout, emplacement). C'est-à-dire, au lieu de "premierevisite\_emplacement", nous avons plutôt "premiere\_visite\_emplacement", donc le trait de soulignement est utilisé pour deux buts. Pour cette raison, si vous essayez notre stratégie de pivot habituelle, vous obtiendrez un message d'erreur :

```
les %>%
  longer(2:7,
         names_to = c("numero_visite", ".value"),
         names_sep = "_")
```

```
Error in `pivot_longer()`:
! Can't combine `premiere_visite_emplacement` <character> and
`premiere_visite_cout` <double>.
Run `rlang::last_trace()` to see where the error occurred.
```

La façon la plus directe de restructurer ce jeu de données avec succès serait d'utiliser un "regex" spécial (manipulation de chaînes de caractères), mais il est probable que vous n'ayez pas encore appris cela !

Alors pour l'instant, la solution que nous recommandons est de renommer manuellement vos colonnes pour insérer un séparateur clair, "\_\_" :

```
les_renomme <-
sites %>%
  e(premiere__visite_emplacement = premiere_visite_emplacement,
    premiere__visite_cout = premiere_visite_cout,
    deuxieme__visite_emplacement = deuxieme_visite_emplacement,
    deuxieme__visite_cout = deuxieme_visite_cout,
    troisieme__visite_emplacement = troisieme_visite_emplacement,
    troisieme__visite_cout = troisieme_visite_cout)

les_renomme
```

```
## # A tibble: 5 × 7
##       id premiere__visite_...¹ premiere__visit...²
##   <dbl> <chr>                  <dbl>
## 1 100202 GH                      0
## 2 100396 Pvt. docto              1500
## 3 100590 Pvt. docto              2000
## 4 100687 Pvt. hospi             20000
## 5 100784 Pvt. docto              1000
##   deuxieme__visite_emplace...³ deuxieme__visit...⁴
##   <chr>                  <dbl>
## 1 <NA>                      0
## 2 Pvt. clinici                1000
## 3 Pvt. docto                 3000
```

```

## 4 Pvt. hospi          1500
## 5 GH                  0
## # i abbreviated names: `premiere_visite_emplacement,
## #   `premiere_visite_cout, ...

```

Maintenant, nous pouvons essayer le pivot :

```

es_long <-
sites_renomme %>%
longer(2:7,
       names_to = c("numero_visite", ".value"),
       names_sep = "__")
es_long

```

```

## # A tibble: 5 × 4
##       id numero_visite visite_emplacement
##   <dbl> <chr>           <chr>
## 1 100202 premiere        GH
## 2 100202 deuxieme       <NA>
## 3 100202 troisieme      <NA>
## 4 100396 premiere        Pvt. docto
## 5 100396 deuxieme       Pvt. clini
##   visite_cout
##       <dbl>
## 1 0
## 2 0
## 3 0
## 4 1500
## 5 1000

```

Maintenant, nettoyons le jeu de données :

```

es_long %>%
  # Remplacer les observations manquantes
  !visite_emplacement == "") %>%
  # Affecter un nom significatif aux valeurs de numero_visite
  numero_visite = case_when(numero_visite == "premiere" ~ 1,
                            numero_visite == "deuxieme" ~ 2,
                            numero_visite == "troisieme" ~ 3)) %>%
  # Assurer que visite_cout est numérique
  visite_cout = as.numeric(visite_cout)

```

```

## # A tibble: 5 × 4
##       id numero_visite visite_emplacement
##   <dbl> <dbl>           <chr>
## 1 100202 1               GH
## 2 100396 1               Pvt. docto
## 3 100396 2               Pvt. clini
## 4 100396 3               Pvt. hospi
## 5 100590 1               Pvt. docto
##   visite_cout

```

```

##      <dbl>
## 1      0
## 2  1500
## 3 1000
## 4 2500
## 5 2000

```

Ici, nous avons d'abord supprimé les observations où nous n'avons pas d'information sur l'emplacement de la visite (c'est-à-dire que nous filtrons les lignes où la variable d'emplacement de la visite est définie à ""). Nous convertissons ensuite en valeurs numériques la variable du numero de la visite, où les chaînes "premiere" à "troisieme" sont converties en valeurs numériques 1 à 3. Enfin, nous nous assurons que la variable du coût de la visite est numérique en utilisant `mutate()` et la fonction d'aide `as.numeric()`.

Nous allons utiliser les données d'une enquête alimentaire au Vietnam. Des femmes de Hanoi ont été interrogées sur leurs achats alimentaires, et les données collectées ont servi à créer un profil nutritionnel de chaque femme. Ici, nous utiliserons un sous-ensemble de ces données de 61 ménages qui sont venus pour 2 visites, enregistrant :

- `enerc_kcal_s_1` : l'apport énergétique de l'ingrédient/nourriture (Kcal) lors de la première visite (\_2 pour la deuxième visite)
- `sec_s_1` : l'apport sec de l'ingrédient/nourriture (g) lors de la première visite (\_2 pour la deuxième visite)
- `eau_s_1` : l'apport en eau de l'ingrédient/nourriture (g) lors de la première visite (\_2 pour la deuxième visite)
- `graisse_s_1` : l'apport en lipides de l'ingrédient/nourriture (g) lors de la première visite (\_2 pour la deuxième visite)



```

:e_alimentaire_vietnam_large <-
  read_csv(here("data/fr_diet_diversity_vietnam_wide.csv"))

:e_alimentaire_vietnam_large

```

```

## # A tibble: 5 × 10
##       ...1 menage_id enerc_kcal_s_1
##      <dbl>      <dbl>      <dbl>
## 1      1        348     2268.
## 2      2        354     2775.

```

```

## 5      5       211       1298.
##   enerc_kcal_s_2 sec_s_1 sec_s_2 eau_s_1
##           <dbl>    <dbl>    <dbl>    <dbl>
## 1      1386.     548.     281.     4219.
## 2      1240.     600.     284.     2376.
## 3      2075.     646.     451.     2808.
## 4      2146.     620.     807.     3457.
## 5      1191.     269.     288.     2584.
## # i 3 more variables: eau_s_2 <dbl>, graisse_s_1 <dbl>,
## #   graisse_s_2 <dbl>

```

Vous devrez d'abord vérifier si vous avez un opérateur net et renommer vos colonnes si nécessaire. Ensuite, rassemblez les données enregistrées sur les deux visite dans une colonne par type d'apport (énergétique, lipides, eau et poids sec). En d'autres termes, pivotez le jeu de données en un format long de cette forme :

menage_id	visite	enerc_kcal_s	sec_s	eau_s	graisse_s
-----------	--------	--------------	-------	-------	-----------

### PRACTICE



```

site_alimentaire_vietnam_large <-
site_alimentaire_vietnam_large %>%
longer(____)

```

```

ez votre réponse
Q_diversite_alimentaire_vietnam_large()

```

```

## Vous n'avez pas encore défini l'objet de réponse,
`Q_diversite_alimentaire_vietnam_large`.
## 1 2 3 4 ▷5◁ 6

```

```

diversite_alimentaire_vietnam_large()

```

```

##
## INDICE.
##
## Commencez par renommer les colonnes en utilisant la
fonction `rename` du package `tidyverse` afin qu'elles aient
un séparateur double underscore.
## - Ensuite, utilisez `pivot_longer` et spécifiez la plage
de colonnes `2:9`.
## - Utilisez `names_sep = "__"` et `names_to = c(".value",
"visit")` pour formater les noms.
##
```

## Du format long au format large

Nous venons de voir comment effectuer certaines opérations complexes du format large au format long, qui, comme nous l'avons vu dans la leçon précédente, sont essentielles pour tracer et manipuler les données. Passons maintenant à la transformation inverse.

Il peut être utile de passer du format long au format large pour transformer et filtrer les données ou encore pour traiter des valeurs manquantes (NA). Dans ce format, vos mesures / données collectées deviennent les colonnes du jeu de données.

Cette fois-ci, nous allons utiliser le jeu de données originel sur les entéropathogènes en Zambie. En effet, ce que vous manipulez jusqu'à présent était un jeu de données **préparé pour vous**, en format large. **Le jeu de données originel est au format long** et nous allons maintenant voir la préparation des données que j'ai faite au préalable, en coulisses. Vous êtes presque en train de devenir l'enseignant de cette leçon ;)

```
pathogenes_zambie_long <-  
  read_csv(here("data/fr_enteropathogenes_zambie_long.csv"))  
pathogenes_zambie_long
```

```
## # A tibble: 5 × 5  
##   ID group    LPS     LBP  IFABP  
##   <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1  1002     1  222. 38414. 1294.  
## 2  1002     2  390.  6840.  610.  
## 3  1003     1  181. 26888.  22.5  
## 4  1004     2  221.  5426.    0  
## 5  1004     1  257. 49183.    0
```

Voici comment nous le convertissons du format long au format large :

```
pathogenes_zambie_large <-  
  pathogenes_zambie_long %>%  
  wider(  
    .from = group,  
    .from = c(LPS, LBP, IFABP)  
  
pathogenes_zambie_large
```

```
## # A tibble: 5 × 7  
##   ID LPS_1 LPS_2  LBP_1 LBP_2  IFABP_1  
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1  1002  222.  390. 38414. 6840.  1294.  
## 2  1003  181.   NA 26888.   NA    22.5  
## 3  1004  257.  221. 49183.  5426.    0
```

```

## 4 1005 NA 369. NA 1938. 0
## 5 1006 275. NA 61758. NA 0
## IFABP_2
## <dbl>
## 1 610.
## 2 NA
## 3 0
## 4 1010.
## 5 NA

```

Vous pouvez voir que les valeurs de la variable `group` (1 ou 2) sont ajoutées aux noms des valeurs (LPS, LBP, IFABP) pour créer les nouvelles colonnes représentant différents groupes de données : par exemple, `LPS_1` et `LPS_2`.

Nous considérons que c'est une option "avancée" du pivot car nous pivotons plusieurs variables en même temps, mais comme vous pouvez le voir, la syntaxe est assez simple. Nous utilisons les mêmes arguments `names_from` et `values_from` qu'avec les pivots plus simples que nous avons vus dans la leçon précédente.

Voyons un autre exemple, en utilisant les données de l'enquête alimentaire du Vietnam que vous avez manipulées précédemment :

```

e_alimentaire_vietnam_long <-
  read_csv(here("data/fr_diet_diversity_vietnam_long.csv"))
e_alimentaire_vietnam_long

```

```

## # A tibble: 5 × 6
##   numero_visite menage_id energie_kcal_s sec_s
##           <dbl>      <dbl>      <dbl> <dbl>
## 1             1       348     2268.  548.
## 2             1       354     2775.  600.
## 3             1       53      3104.  646.
## 4             1       18      2802.  620.
## 5             1      211     1298.  269.
##   eau_s graisse_s
##   <dbl>    <dbl>
## 1 4219.    78.4
## 2 2376.    115.
## 3 2808.    127.
## 4 3457.    87.4
## 5 2584.    47.8

```

Ici, nous allons utiliser la variable `numero_visite` pour créer une nouvelle variable pour les différents apports enregistrés lors des deux visites :

```

fe_alimentaire_vietnam_large <-
site_alimentaire_vietnam_long %>%
wider(
es_from = numero_visite,
ies_from = c(enerc_kcal_s, sec_s, eau_s, graisse_s)

fe_alimentaire_vietnam_large

```

```

## # A tibble: 5 × 9
##   menage_id enerc_kcal_s_1 enerc_kcal_s_2
##       <dbl>      <dbl>      <dbl>
## 1         348.      2268.     1386.
## 2         354.      2775.     1240.
## 3         53.       3104.     2075.
## 4         18.       2802.     2146.
## 5        211.      1298.     1191.
## #>   sec_s_1 sec_s_2 eau_s_1 eau_s_2 graisse_s_1
## #>   <dbl>    <dbl>    <dbl>    <dbl>      <dbl>
## #> 1      548.    281.    4219.    1997.     78.4
## #> 2      600.    284.    2376.    3145.     115.
## #> 3      646.    451.    2808.    2305.     127.
## #> 4      620.    807.    3457.    1903.     87.4
## #> 5      269.    288.    2584.    2269.     47.8
## #> # i 1 more variable: graisse_s_2 <dbl>

```

Vous pouvez voir que les valeurs de la variable `numero_visite` (1 ou 2) sont ajoutées aux noms des valeurs (`enerc_kcal_s`, `sec_s`, `graisse_s`, `eau_s`) pour créer les nouvelles colonnes représentant différents groupes de données : par exemple, `eau_s_1` et `eau_s_2`. Nous avons pivoté en format large toutes ces variables en même temps. Maintenant, chaque mesure de l'apport par visite est représentée comme une seule variable (c'est-à-dire une colonne) dans le jeu de données.

Avec ce format, il est facile de faire la somme de l'apport énergétique par ménage par exemple :

```

fe_alimentaire_vietnam_large %>%
  (menage_id, enerc_kcal_s_1, enerc_kcal_s_2) %>%
  (energie_totale_kcal = enerc_kcal_s_1 + enerc_kcal_s_2) %>%
  fe(menage_id)

```

```

## # A tibble: 5 × 4
##   menage_id enerc_kcal_s_1 enerc_kcal_s_2
##       <dbl>      <dbl>      <dbl>
## 1         14.      1040.     1663.
## 2         17.      2100.     1286.
## 3         18.      2802.     2146.
## 4         22.      3187.     1582.
## 5         24.      2359.     2026.
## #>   energie_totale_kcal
## #>           <dbl>

```

```
## 1      2704.  
## 2      3386.  
## 3      4948.  
## 4      4769.  
## 5      4385.
```

Cependant, vous pourriez obtenir un résultat similaire avec le format long :

```
me_alimentaire_vietnam_long %>%  
  by(menage_id) %>%  
  size(energie_totale = sum(enerc_kcal_s))
```

```
## # A tibble: 5 × 2  
##   menage_id energie_totale  
##       <dbl>          <dbl>  
## 1         14        2704.  
## 2         17        3386.  
## 3         18        4948.  
## 4         22        4769.  
## 5         24        4385.
```

Prenez le jeu de données `tb_visites_long` que nous avons manipulé plus haut et pivotez-le à nouveau au format large.

```
sites_large <-  
sites_long %>%  
pivot_wider(names_from = numero_visite,  
           values_from = c(visite_emplacement, visite_cout))  
  
#ez votre réponse  
?_tb_visites_large()
```



```
## Correct !  
## 1 2 3 4 5 ▷6◁
```

```
- tb_visites_large()
```

```
##  
## INDICE.  
##  
## Commencez par utiliser la fonction `pivot_wider` du  
## package `tidyverse` sur le jeu de données `tb_visites_long`.  
## Vous voudrez spécifier quelles colonnes fourniront les  
## nouveaux noms de colonnes (`names_from`) et de quelles  
## colonnes obtenir les valeurs (`values_from`).
```



```
## Ensuite, utilisez la fonction `rename` pour changer les  
noms de colonnes comme requis.  
##  
##   tb_visites_long %>%  
##   pivot_wider(_____  
##
```

## Bilan !

Vos compétences en manipulation de données viennent d'être renforcées avec le pivot avancé. Cette compétence s'avérera souvent essentielle lors de la manipulation des données du monde réel. Je ne doute pas que vous la mettrez bientôt en pratique. Elle est également essentielle, comme nous l'avons vu, pour la conception des graphiques. J'espère donc que le pivot vous sera utile non seulement pour votre manipulation de données, mais aussi pour la conception des graphiques.

## Contributeurs

Les membres suivants de l'équipe ont contribué à cette leçon :



### KENE DAVID NWOSU

Data analyst, the GRAPH Network  
Passionate about world improvement



### LAURE VANCAUWENBERGHE

Data analyst, the GRAPH Network  
A firm believer in science for good, striving to ally programming, health and education



### CAMILLE BEATRICE VALERA

Project Manager and Scientific Collaborator, The GRAPH Network



### IMANE BENSOUDA KORACHI

R Developer and Instructor, the GRAPH Network

---

## Références

# Introduction à {ggplot2}

February 2024



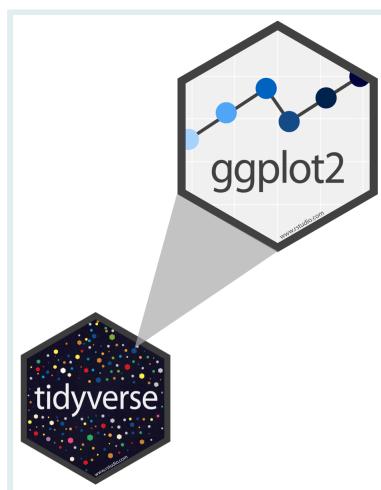
Introduction	.
Objectifs d'apprentissage	.
Les packages	.
Les épidémies de Rougeole au Niger	.
Le dataset <code>nigerm</code>	.
La grammaire des graphiques (GG)	.
Les couches essentielles	.
Construire un <code>ggplot()</code> étape par étape	.
Modifier les couches	.
Modifier le mapping <code>aes()</code>	.
Modifier la fonction <code>geom_*</code>	.
Mapper d'autres esthétiques à l'intérieur de <code>aes()</code>	.
Esthétiques fixes en dehors de <code>aes()</code>	.
Les autres couches	.
Les acquis	.
Contributeurs	.
Références	.

---

## Introduction

Bienvenue au cours de visualisation de données de The GRAPH Course !

Nous allons apprendre à utiliser le **package {ggplot2}** pour créer des graphiques de haute qualité avec R.



{ggplot2} est une extension du {tidyverse}, et le package le plus utilisé pour la visualisation de données.

C'est parti !

## Objectifs d'apprentissage

A la fin de ce cours, vous devriez être capable de :

1. Comprendre la **Grammaire des Graphiques (Grammar of Graphics)**, le cadre de visualisation de données sur lequel se base le package **{ggplot2}**.
2. Nommer et décrire les 3 couches (layers) essentielles à la construction d'un graphique : **données, esthétiques, et géométries**.
3. Écrire correctement le code pour **construire un graphique avec ggplot** en fournissant les 3 couches essentielles à la **fonction ggplot()**.
4. Créer différents types de graphiques comme les **graphiques de dispersion, les graphiques linéaires** et les **graphiques à barres**.
5. Ajouter ou modifier les propriétés visuelles d'un graphique tels que la **couleur** ou la **taille**.
6. Distinguer entre les **esthétiques mappées** et les **esthétiques fixes** et savoir les utiliser.



Illustration de Allison Horst

---

## Les packages

{ggplot2} fait partie intégrante du méga-package {tidyverse}. Il n'est donc pas nécessaire de le charger séparément. Le package {here} permet de référencer correctement les chemins de fichiers dans votre code R.

```
# Charger les packages
pacman::p_load(tidyverse,
                here)
```

---

## Les épidémies de Rougeole au Niger

Dans ce cours, nous allons étudier les épidémies de rougeole au Niger.

La rougeole est une **maladie virale hautement contagieuse** qui se propagent par les gouttelettes en suspension dans l'air.

Comme la rougeole se transmet également par contact direct, la **densité de la population** est un facteur important dans sa propagation.

### Le dataset `nigerm`

Nous allons générer des graphiques à partir d'un dataset qui recense les cas de rougeole signalés chaque semaine à l'échelle régionale au Niger.

Ces données ont été recueillies par le ministère de la Santé du Niger et couvre la période du 1er janvier 1995 au 31 décembre 2005.

Pour commencer, nous allons charger notre dataset prétraité :

```
# Importer le dataframe dans l'environnement RStudio
load(here("data/clean/nigerm_cases_rgn.RData"))
```

Ensuite, prenez le temps d'examiner les données :

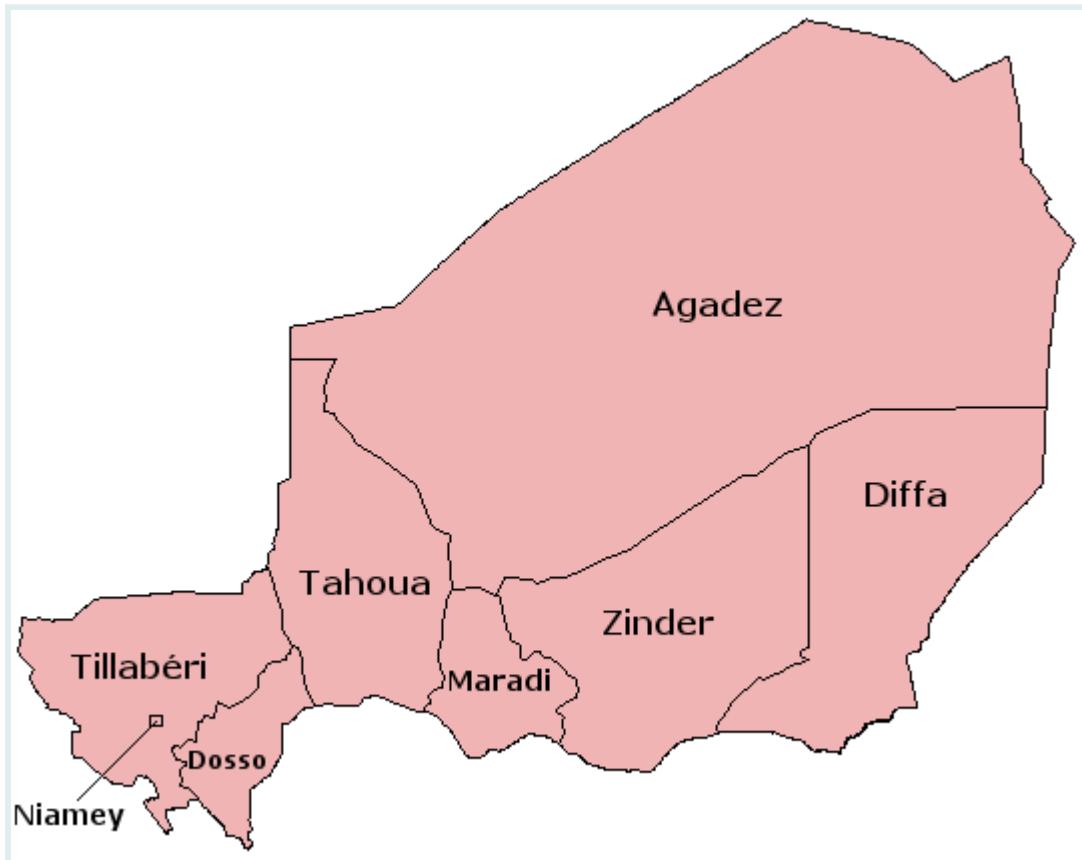
```
# Afficher le dataframe nigerm
nigerm
```

Le dataframe `nigerm` contient 4 variables (ou colonnes) :

1. `year` : Année civile (de 1995 à 2005)
2. `week` : Semaine de l'année (de 1 à 52)

3. **region** : Région dans laquelle les cas ont été enregistrés (voir figure ci-dessous)

4. **cases** : Nombre de cas de rougeole signalés



Régions du Niger

Plusieurs publications ont étudié le lien entre la rougeole et l'activité humaine, la migration, et la saisonnalité.

The screenshot shows two side-by-side journal article pages. On the left, the **JOURNAL OF THE ROYAL SOCIETY INTERFACE** page features an article titled "Investigating persistent measles dynamics in Niger and associations with rainfall" by Alexandre Blake, Ali Djibo, Ousmane Guindo and Nita Bharti, published on 26 August 2020. On the right, the **PROCEEDINGS OF THE ROYAL SOCIETY B BIOLOGICAL SCIENCES** page features an article titled "Rural–urban gradient in seasonal forcing of measles transmission in Niger" by Matthew J. Ferrari, Ali Djibo, Rebecca F. Grais, Nita Bharti, Bryan T. Grenfell and Ottar N. Bjornstad, published on 28 April 2010. Both articles have "Check for updates" buttons.

Articles de recherche qui ont utilisé ce dataset et l'ont analysé avec R !

Ces études sont nettement plus complexes que ce que nous allons faire ici, mais voyons si nous pouvons détecter des relations à l'aide d'une **visualisation exploratoire des données**.

Nous pouvons déjà obtenir certaines informations sur ces données en inspectant le résumé statistique fourni par la fonction `summary()` :

```
summary(nigerm)
```

```
##      year          week        region
##  Min.   :1995   Min.   : 1.00   Agadez : 572
##  1st Qu.:1997  1st Qu.:13.75  Diffa   : 572
##  Median :2000   Median :26.50  Dosso   : 572
##  Mean    :2000   Mean    :26.50  Maradi  : 572
##  3rd Qu.:2003  3rd Qu.:39.25  Niamey  : 572
##  Max.    :2005   Max.    :52.00  Tahoua  : 572
##                           (Other):1144
##
##      cases
##  Min.   : 0.0
##  1st Qu.: 1.0
##  Median :16.0
##  Mean   :100.3
##  3rd Qu.:86.0
##  Max.   :1887.0
##
```

`summary()` fournit les valeurs maximale, minimale et les quartiles de chaque variable numérique, ainsi que le nombre d'observations (lignes) pour chaque région. Ce résumé est utile, mais il omet une grande partie des informations du dataset.

Il faut aussi garder à l'esprit que les résumés statistiques peuvent nous induire en erreur, et qu'un simple graphique peut révéler beaucoup plus.

La manière la plus simple et la plus claire d'analyser les relations à partir de ce dataset est de les visualiser !

Dans R, `{ggplot2}` est le meilleure outils pour la visualisation de données. Voyons donc comment cela fonctionne.

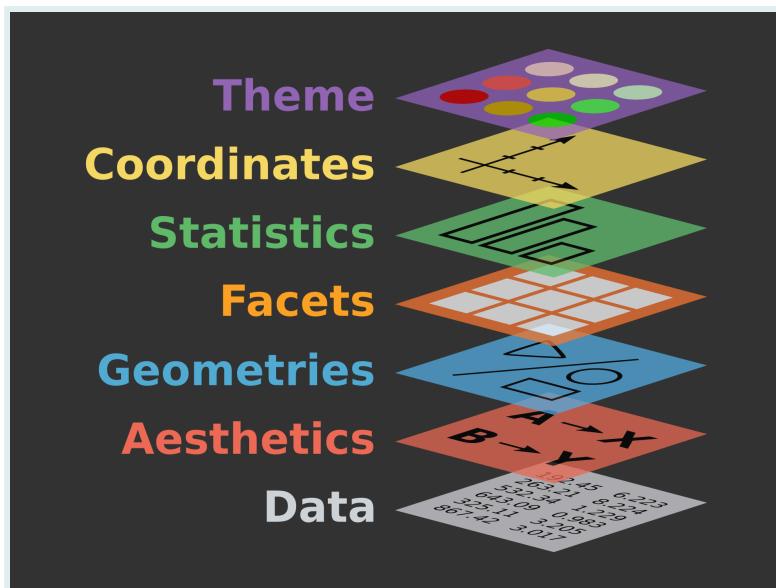
## La grammaire des graphiques (GG)

Le `gg` dans `ggplot` fait référence à “grammar of graphics” (grammaire des graphiques). C'est un cadre théorique qui décompose le processus de création d'un graphique.

Prenons pour exemple la façon de construire une phrase. Une phrase est composée d'un ensemble de mots (sujet, verbe, complément, etc.). Ces éléments ne peuvent pas être combinés dans n'importe quel ordre; ils suivent un ordre logique défini par les règles de grammaire.

De la même manière, la grammaire des graphiques (GG) définit un ensemble de règles pour construire des *graphiques* en combinant différents types d'éléments

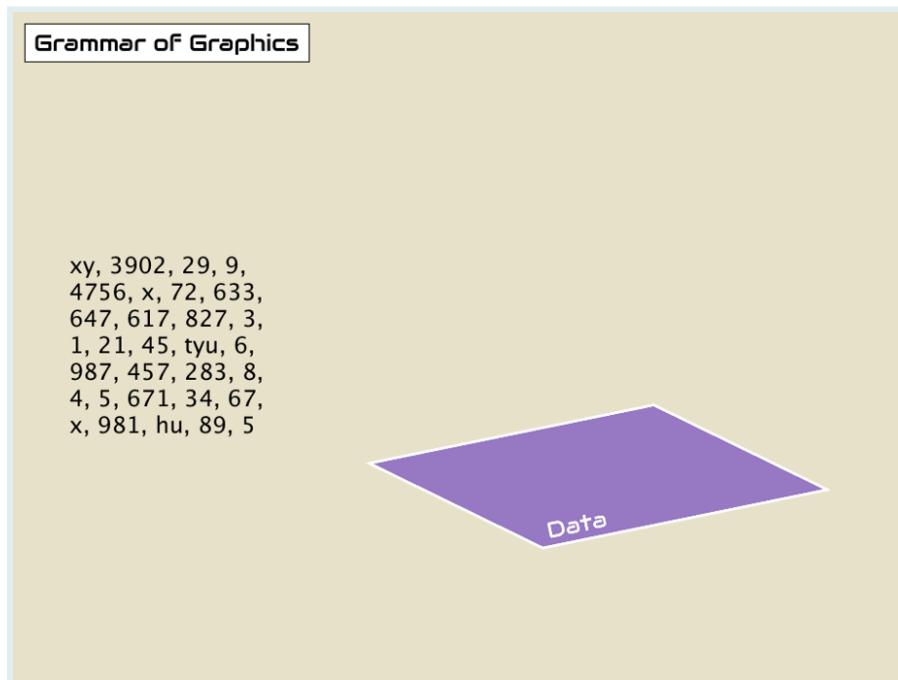
appelés *couches* (*layers*).



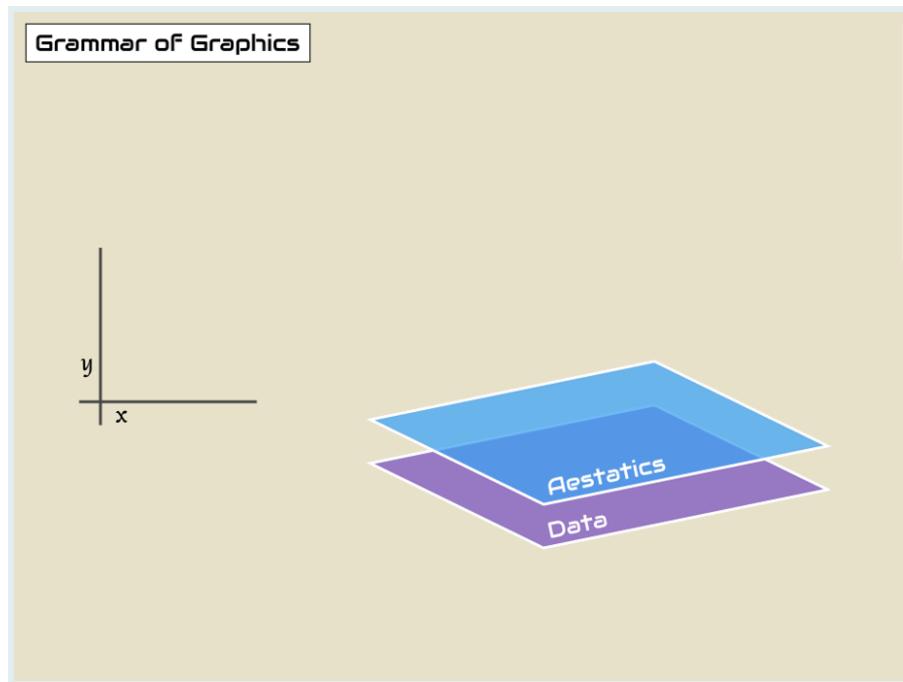
Le cadre de la grammaire des graphiques dissèque un graphique en composants individuels appartenant à sept couches distinctes. Nous combinons ces couches pour construire un graphique.

Quel que soit le type de graphique, 3 couches doivent obligatoirement être incluses :

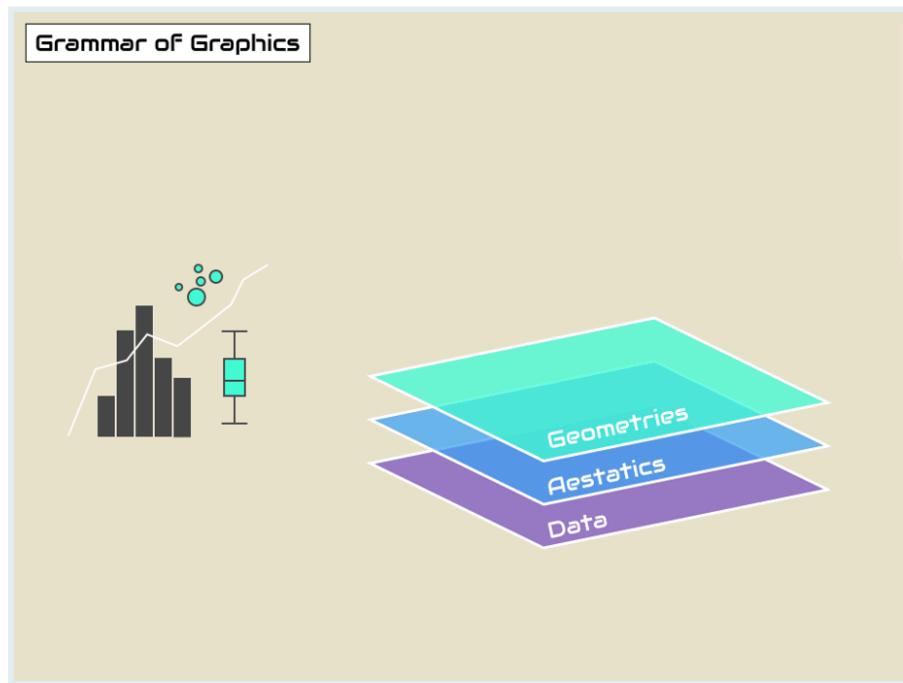
1. **data** (données) : Le jeu de données contenant les variables que vous souhaitez visualiser.



2. **aesthetics** (esthétiques) : Les propriétés visuelles attribuées aux données permettant de les représenter graphiquement.



3. **geometry** (géométrie) : Le type de représentation graphique choisi pour les données, tel que des points, des lignes ou des barres.



Vous vous demandez peut-être pourquoi nous avons écrit `data`, `geom` et `aes` dans une police de caractères de type code informatique. Vous verrez très bientôt que nous utilisons ces termes dans le code R pour représenter les couches.

### CHALLENGE



**CHALLENGE**

Les termes et la syntaxe employés pour les fonctions, les arguments et les couches de `ggplot` peuvent sembler complexes au début, mais à force de les utiliser, vous finirez par les maîtriser.

## Les couches essentielles

Créons ensemble notre premier graphique avec `{ggplot2}`. Nous allons créer un nuage de point (scatter plot) à partir des données de `nigerm`.

Tout d'abord, nous allons créer un premier subset que nous appellerons `nigerm96`, contenant uniquement les données des cas de rougeole recensés en 1996. En exécutant le code ci-dessous, vous allez créer un nouveau dataframe `nigerm96` qui sera ajouté à votre environnement RStudio :

```
# Créer le dataframe nigerm96
nigerm96 <- nigerm %>%
  filter(year == 1996)  %>% # filtrer pour inclure uniquement les lignes de
                        1996
  select(-year) # retirer la colonne année
```

**REMINDER**

Les fonctions `select()` et `filter()` font partie du package `{dplyr}`, qui est un package central du `{tidyverse}`. Ces fonctions ont déjà été abordées dans le cours sur la Manipulation des Données. Consultez le [site web](#) de The GRAPH Course pour plus d'informations.

Jetons un œil à notre nouveau dataframe `nigerm96`:

```
# Afficher nigerm96
nigerm96
```

## Construire un `ggplot()` étape par étape

Commençons à créer notre `ggplot` ! La création d'un graphique avec `{ggplot2}` débute avec l'appel de la fonction `ggplot()` à laquelle nous allons ajouter les différentes couches.

### Étape 0 : Appeler la fonction `ggplot()`

```
# Appeler la fonction `ggplot()`
ggplot()
```

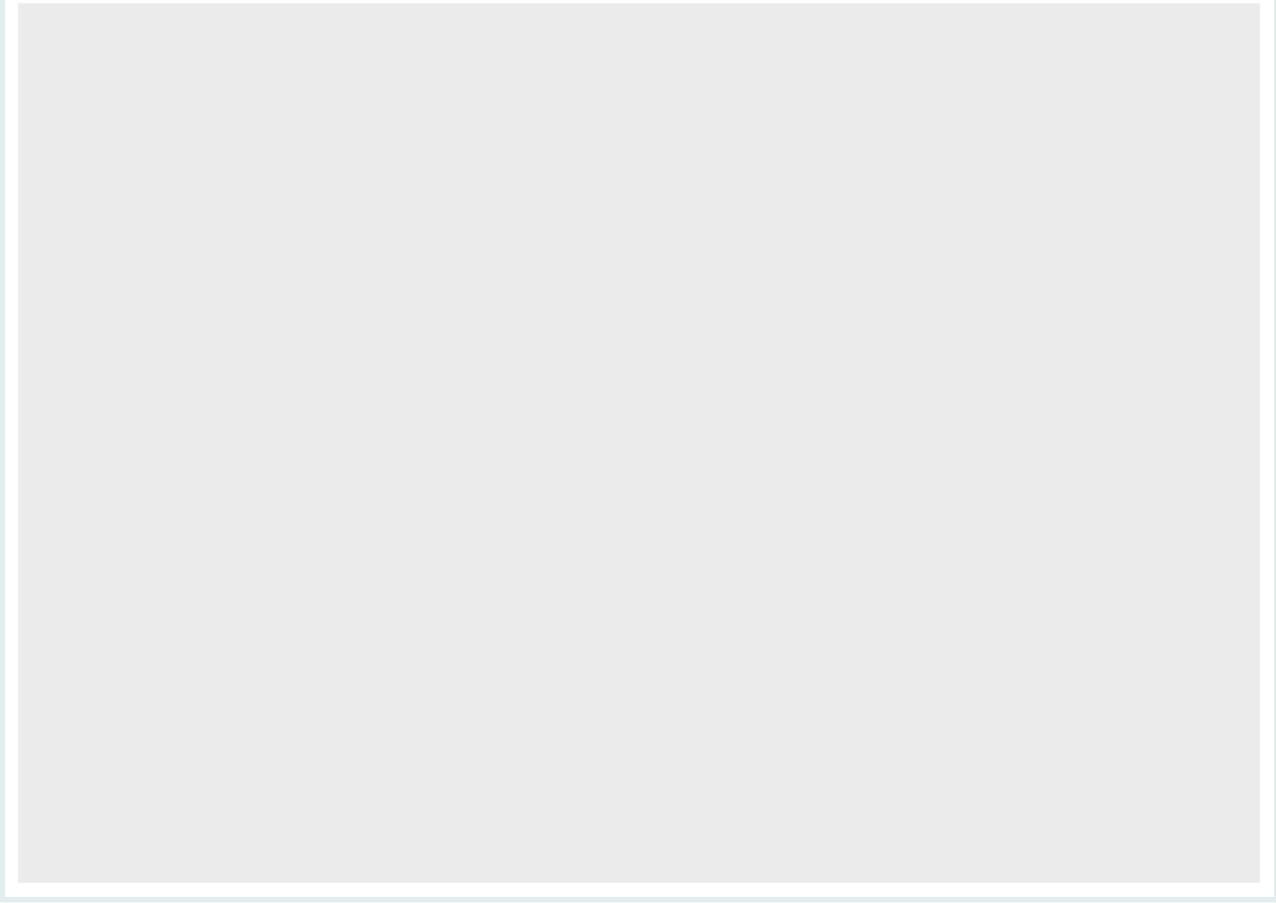


Comme vous pouvez le voir, cela ne nous donne rien d'autre qu'une toile vide. Mais ne vous inquiétez pas, nous sommes sur le point d'ajouter d'autres éléments.

## Étape 1 : Fournir les données

Le premier paramètre de la fonction `ggplot()` à inclure est la couche de données, c'est-à-dire le dataframe sur lequel le graphique va être construit, grâce à l'argument `data (data = NOM_DF)` :

```
# Couche des données
ggplot(data = nigerm96) # Inclure les données à utiliser
```

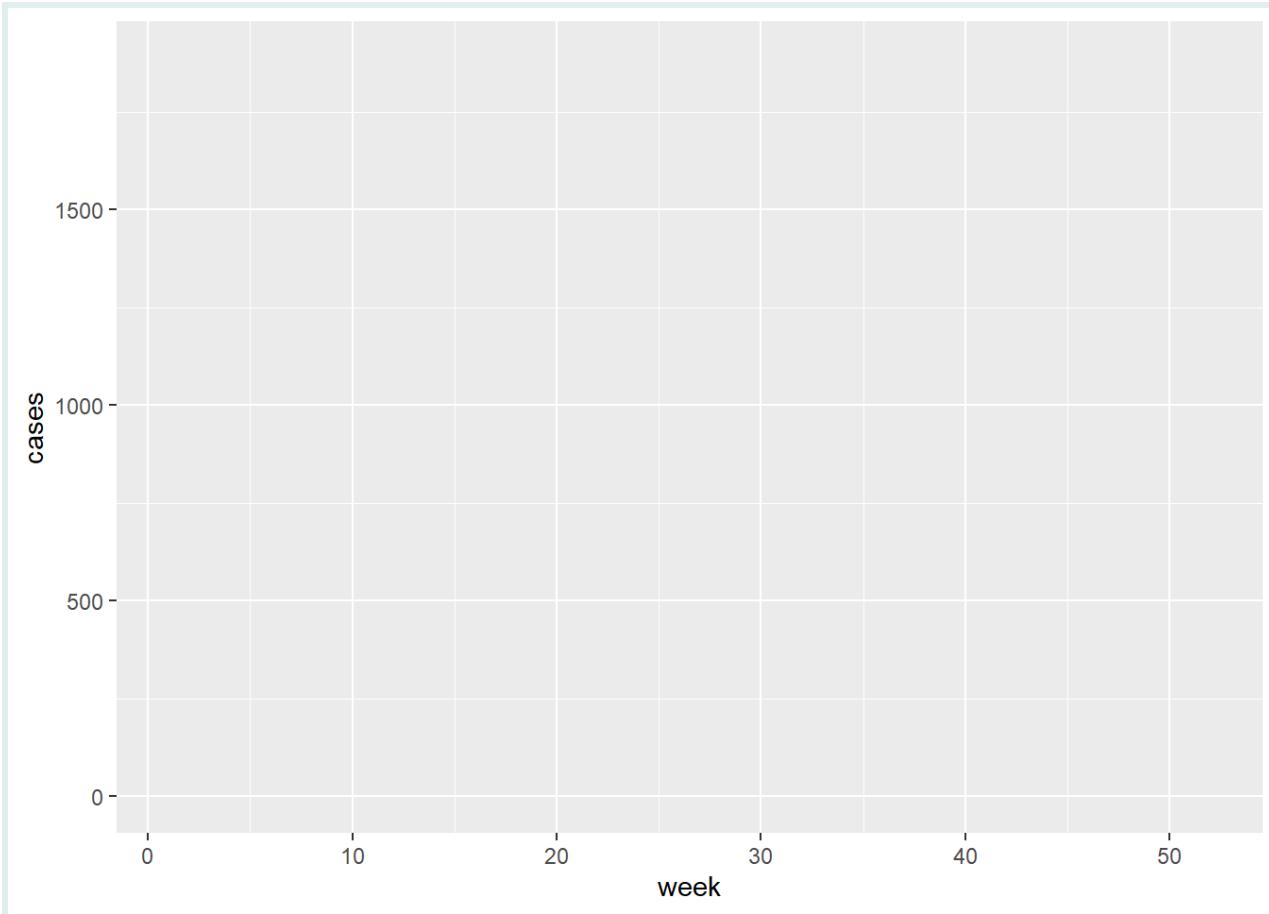


Encore une fois, nous obtenons une toile vide, car nous avons fourni seulement l'une des trois couches nécessaires à la création d'un graphique. Il nous faut maintenant associer les variables aux arguments esthétiques.

## Étape 2: Définir les variables

Quelles données représenter sur nos axes ? Imaginons que nous voulons créer un graphique de série temporelle pour suivre l'évolution d'une épidémie. Dans ce cas, nous plaçons le temps (en semaines) sur l'axe des x et l'incidence de la maladie (nombre de cas signalés) sur l'axe des y. En termes `ggplot`, nous associons (`mapping`) la variable `cases` à l'esthétique `x` et la variable `week` à l'esthétique `y`. Pour indiquer à `ggplot()` quelles variables utiliser pour les esthétiques, nous utilisons l'argument `mapping` avec la syntaxe suivante : `mapping = aes(x = VAR1, y = VAR2)`.

```
# Couche esthétique
ggplot(data = nigerm96, # inclure les données à utiliser
       mapping = aes(    # fournir un mapping sous forme d'une `esthétique`
                      x = week,      # inclure la variable à mapper sur l'axe des x
                      y = cases))   # inclure la variable à mapper sur l'axe des y
```



Même si aucune donnée n'est encore représentée, les échelles, les titres et les étiquettes des axes sont présents. Sur l'axe des x, les semaines de l'année sont marquées de 1 à 52, et sur l'axe des y, nous pouvons observer que le nombre de cas hebdomadaires recensés par région varie de 0 à environ 2000.

Il nous manque toujours la couche géométrique pour compléter le graphique.



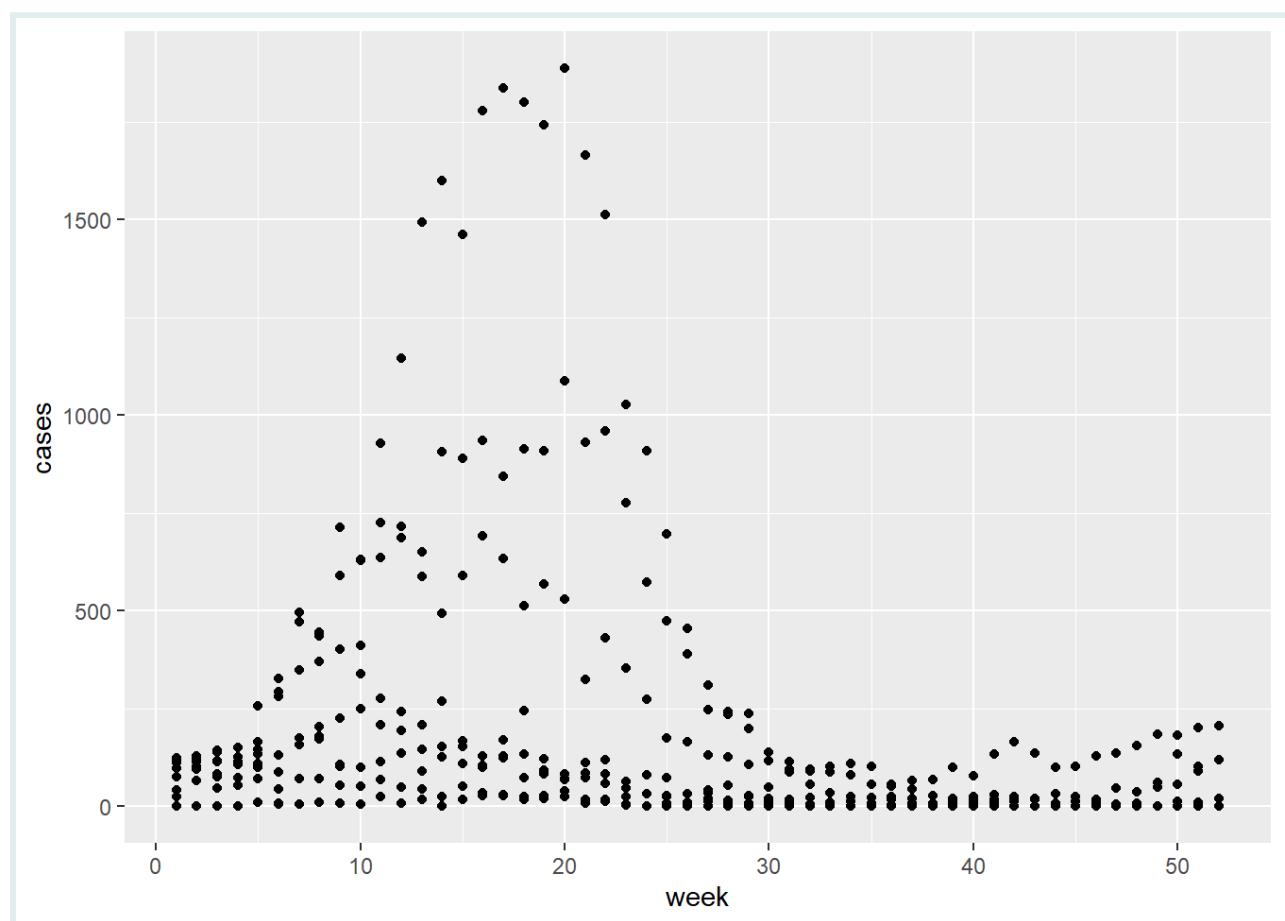
Dans `ggplot()`, `aes()` signifie aesthetics (esthétiques). Ce sont les différentes propriétés visuelles d'un graphique. Les variables sont toujours placées à l'intérieur de la fonction `aes()`, laquelle est elle-même à l'intérieur de `ggplot()`, d'où les doubles parenthèses `).`. La première appartient à `aes()`, tandis que la seconde appartient à `ggplot()`.

### Étape 3 : Préciser le type de graphique à créer

Pour finir, nous allons ajouter la couche de géométrie en utilisant la fonction `geom_*`. Cela détermine le type d'objet géométrique à utiliser pour visualiser les données.

Puisque nous voulons représenter la relation entre deux variables numériques, nous allons créer un **nuage de point (graphique de dispersion)**. Dans un nuage de point, les objets géométriques utilisés pour représenter les données sont des **points**, et la fonction `geom_*` spécifique aux nuages de point est appelée `geom_point()`. Ajoutons cette fonction en tant que nouvelle couche en utilisant l'opérateur `+` :

```
# Couche de géométrie
ggplot(data = nigerm96, # inclure les données à utiliser
       mapping = aes( # fournir un mapping sous forme d'une `esthétique`
                      x = week, # inclure la variable à mapper sur l'axe des x
                      y = cases)) + # inclure la variable à mapper sur l'axe des y
       geom_point() # ajouter une géométrie de type `point` (pour le
                     # nuage de points)
```



Maintenant que les points ont été ajoutés, nous avons un nuage de point complet ! Il y a 8 points par semaine, représentant chacune des 8 régions (mais à ce stade, nous ne pouvons pas dire à quelle région correspond chaque point).

**REMINDER**



La fonction `aes()` est imbriquée à l'intérieur de la fonction `ggplot()`. Assurez-vous de fermer les parenthèse pour les deux fonctions avant

**REMINDER**

d'ajouter le signe + pour la fonction `geom_*`, sinon votre code ne s'exécutera pas correctement.

C'est à votre tour de vous entraîner au traçage de graphique avec `ggplot()` ! Dans les exercices de ce cours, vous allez utiliser un subset de `nigerm` appelé `nigerm04`, qui contient uniquement les données de l'année 2004 :

Tracer avec un ensemble de données différent vous permettra également de vérifier si ce que nous observons pour 1996 est également valable pour 2004.

Utilisez le dataframe `nigerm04` pour générer le code `ggplot` qui représentera graphiquement la relation entre les cas `cases` (sur l'axe des ordonnées) et les semaines `week` (sur l'axe des abscisses) sous forme d'un nuage de points.

```
# Lorsque vous pensez avoir la bonne réponse, soumettez-la en  
# faisant ce qui suit : supprimez "VOTRE_RÉPONSEICI",  
# remplacez-la par votre code et exécutez ces lignes.
```

```
nigerm04_scatter <- "VOTRE_RÉPONSEICI"
```

```
# Assurez-vous que "nigerm04_scatter" apparaît dans votre  
# onglet Environnement.
```

```
# Vérifiez votre réponse en exécutant cette fonction de  
# vérification (aucune entrée requise). La sortie vous  
# indiquera si vous avez répondu correctement ou non.
```

```
.CHECK_nigerm04_scatter()
```

```
# Vous pouvez demander un indice en exécutant cette fonction  
# d'indice (aucune entrée requise).
```

```
.HINT_nigerm04_scatter()
```

```
# Obtenez la solution complète en tapant la fonction de  
# solution :
```

```
.SOLUTION_nigerm04_scatter()
```

**PRACTICE**  
  
**(in RMD)**

## Modifier les couches

La grammaire des graphiques offre un haut degré de personnalisation pour les graphiques. De plus, elle propose une structure cohérente pour faciliter la modification des graphiques.

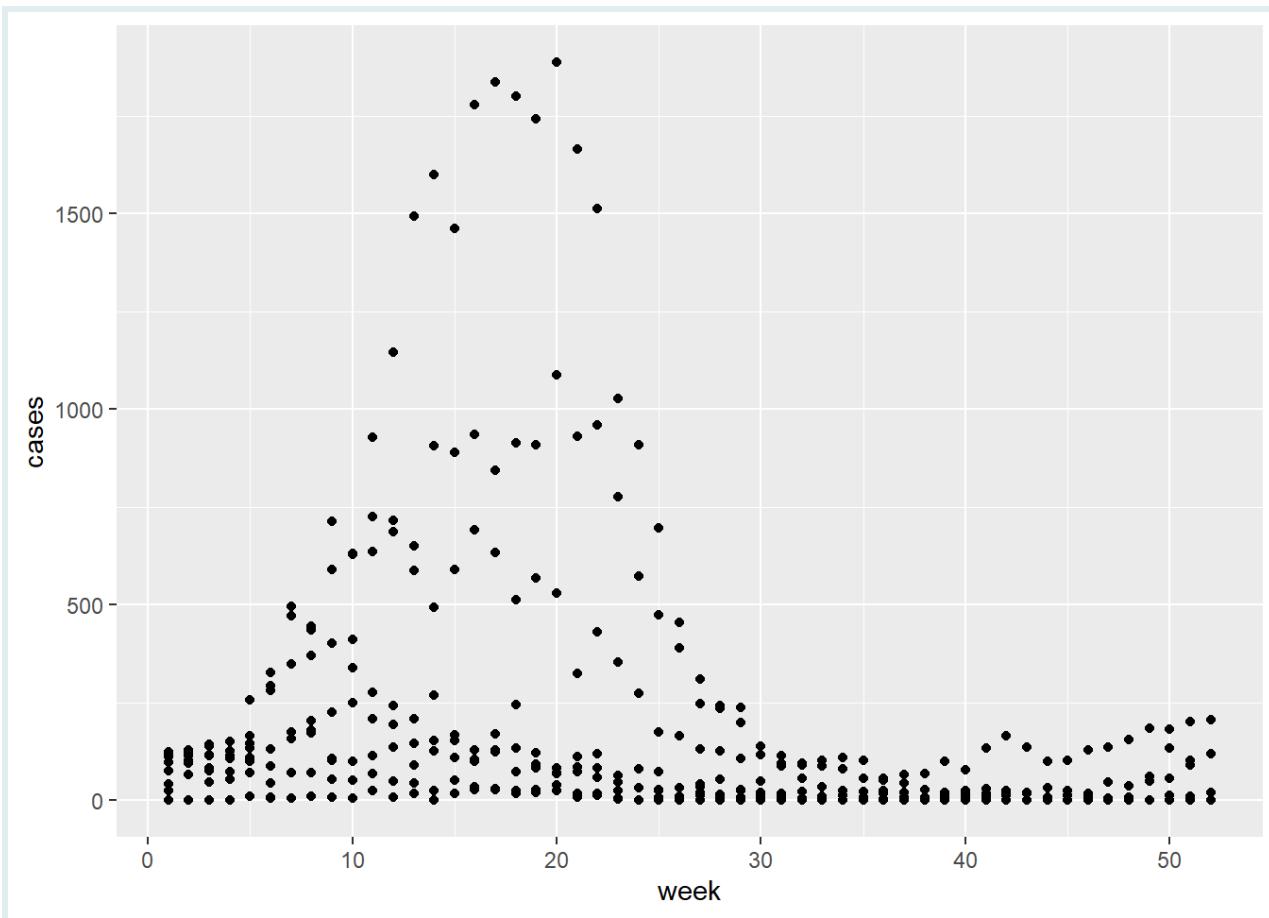
Nous pouvons ajuster un code existant en modifiant les données, les esthétiques et les éléments géométriques fournis à la fonction `ggplot()`. Cela nous permet de créer des variations du graphique originel. D'ailleurs, c'est exactement ce que nous avons fait lorsque nous sommes passés de `nigerm96` à `nigerm04` dans l'exercice précédent.

De la même manière, les couches esthétiques (`aes`) et géométriques (`geom`) peuvent être modifiées pour créer différents rendus visuels. Dans les sections suivantes, nous allons partir du nuage de points que nous avons précédemment créé et ajuster différents éléments du code d'origine.

### Modifier le mapping `aes()`

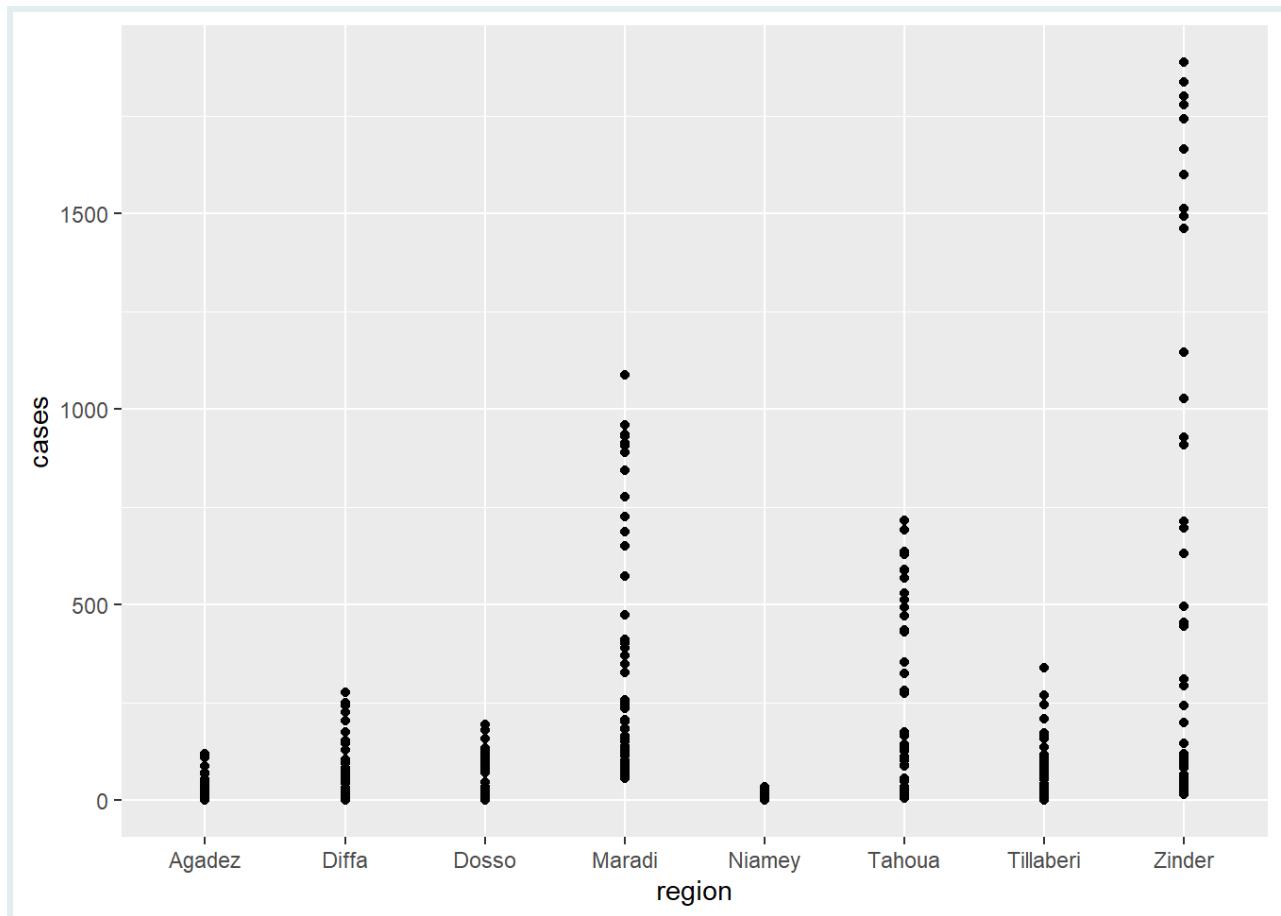
Nous avons créé un nuage de point pour visualiser le nombre de cas `cases` par semaine `week` dans `nigerm96` avec ce code :

```
ggplot(data = nigerm96,  
       mapping = aes(x = week,  
                      y = cases)) +  
  geom_point()
```



Si nous prenons le même code et que nous remplaçons uniquement la variable associée à `x` de `week` (variable numérique) à `region` (variable catégorielle), nous obtenons ce qu'on appelle un **strip plot** :

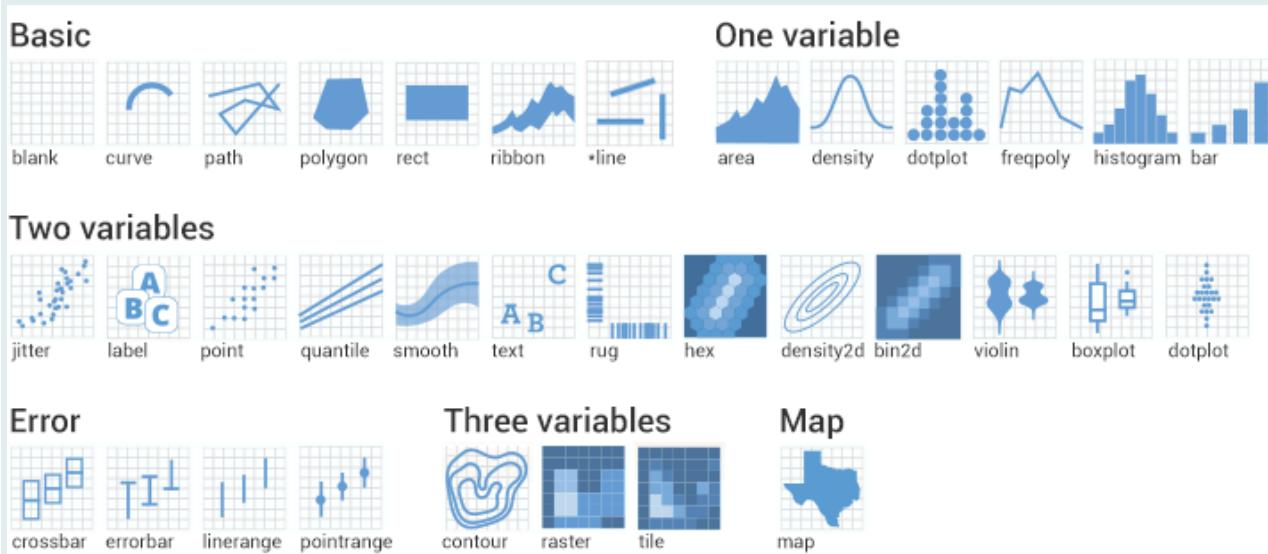
```
ggplot(data = nigerm96,  
       mapping = aes(x = region, # modifier le mapping sur l'axe des x  
                      y = cases)) +  
  geom_point()
```



Si les valeurs sur l'axe des ordonnées n'ont pas changé, le mapping sur l'axe des abscisses a considérablement changé. Les valeurs sont désormais associées à 8 positions distinctes et chaque point est aligné horizontalement le long de l'axe des x en fonction de la région qu'il représente.

#### Modifier la fonction `geom_*`

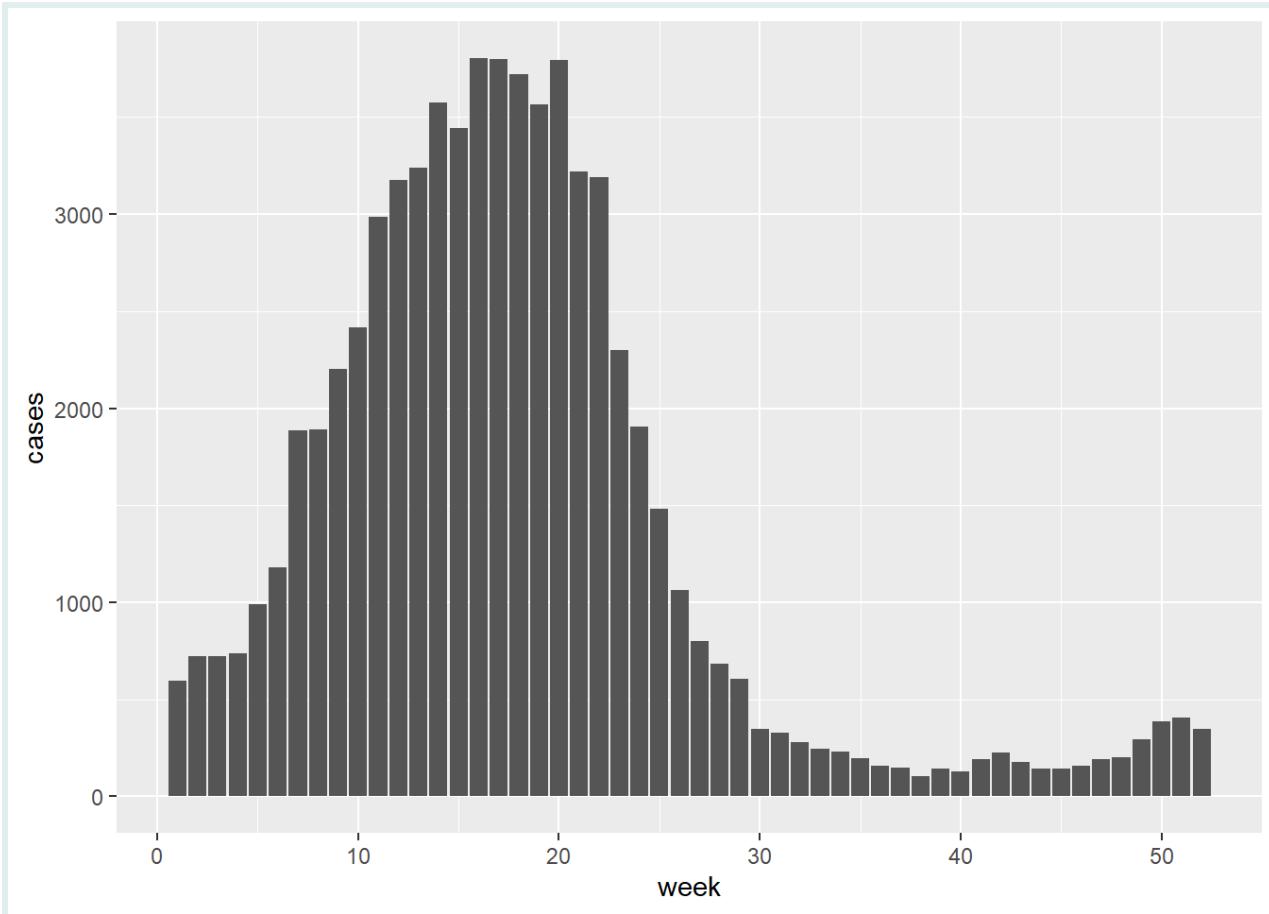
De la même manière, nous pouvons modifier la couche de géométrie pour créer un type de graphique différent, tout en conservant les mêmes mappings esthétiques.



{ggplot2} propose une variété de fonctions `geom_*` et d'objets géométriques que vous pouvez utiliser pour visualiser vos données. Voici quelques exemples de différents types de geoms qui peuvent être utilisés avec `ggplot()`.

Nous allons réutiliser le code original du nuage de points, mais cette fois nous allons uniquement modifier la fonction `geom_*` au lieu de l'esthétique `x`. Si nous remplaçons `geom_point()` par `geom_col()`, nous obtenons un diagramme en barres (appelé aussi un diagramme en colonne) :

```
ggplot(data = nigerm96,
       mapping = aes(x = week,
                     y = cases)) +
  geom_col() # préciser le type de graphique (ici, diagramme en colonne)
```



Encore une fois, le reste du code est toujours le même - nous avons simplement changé le mot clé de la fonction `geom_*`. Cependant, le graphique est significativement différent du nuage de points ou du strip plot.

Notez que l'échelle de l'axe des y a été modifiée. La hauteur de chaque barre représente désormais le nombre cumulé de cas hebdomadaires, c'est-à-dire le nombre total de cas signalés dans l'ensemble des régions durant chaque semaine. Cela diffère de la représentation précédente où chaque région était affichée comme un point de données distinct.



Tous les types de graphiques ne sont pas interchangeables. L'utilisation d'une fonction `geom_*` qui n'est pas compatible avec le type de variable que vous avez choisi dans `aes()` vous donnera une erreur. Par exemple, remplaçons `geom_point()` par `geom_histogram` :

```
ggplot(data = nigerm96,
       mapping = aes(x = week,
                      y = cases)) +
  geom_histogram()
```

**Error?**

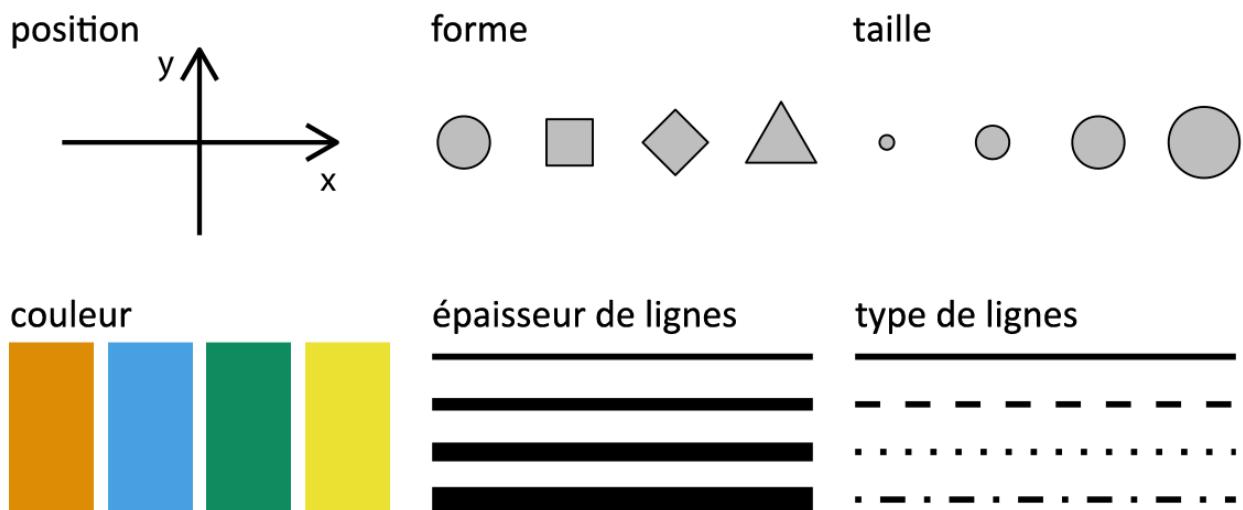
C'est parce qu'un histogramme montre la répartition d'une seule variable numérique. `ggplot()` ne peut pas mapper deux variables à la fois sur les positions des axes `x` et `y` avec un histogramme, donc il renvoie un message d'erreur.

**PRACTICE**

Utilisez le dataframe `nigerm04` pour créer un graphique à barres des cas hebdomadaires avec la fonction `geom_col()`. Mappez `cases` sur l'axe des `y` et `week` sur l'axe des `x`.

## Mapper d'autres esthétiques à l'intérieur de `aes()`

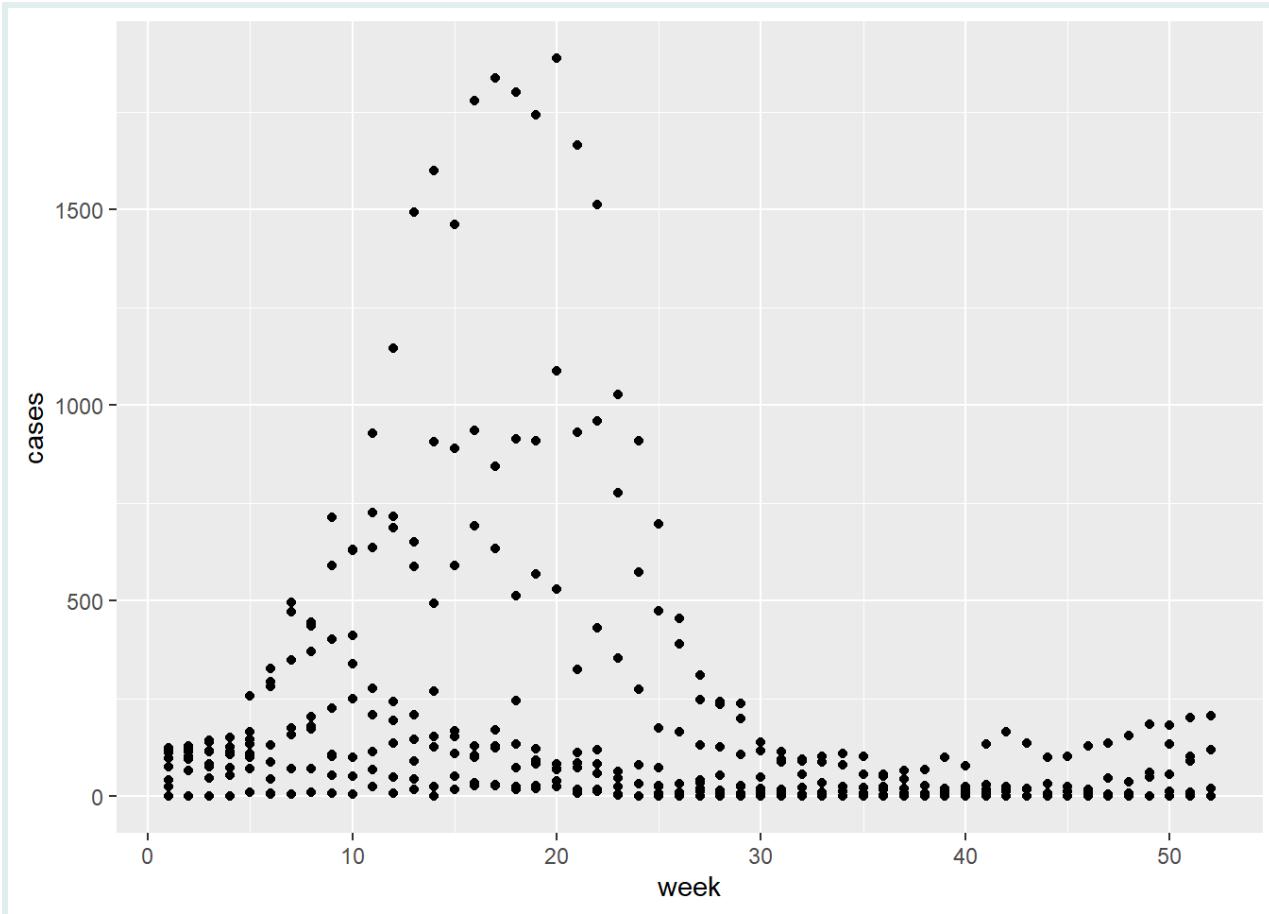
Jusqu'à présent, nous avons mappé nos variables uniquement aux attributs esthétiques `x` et `y`. Nous pouvons également mapper les variables à d'autres esthétiques comme la couleur, la taille ou la forme.



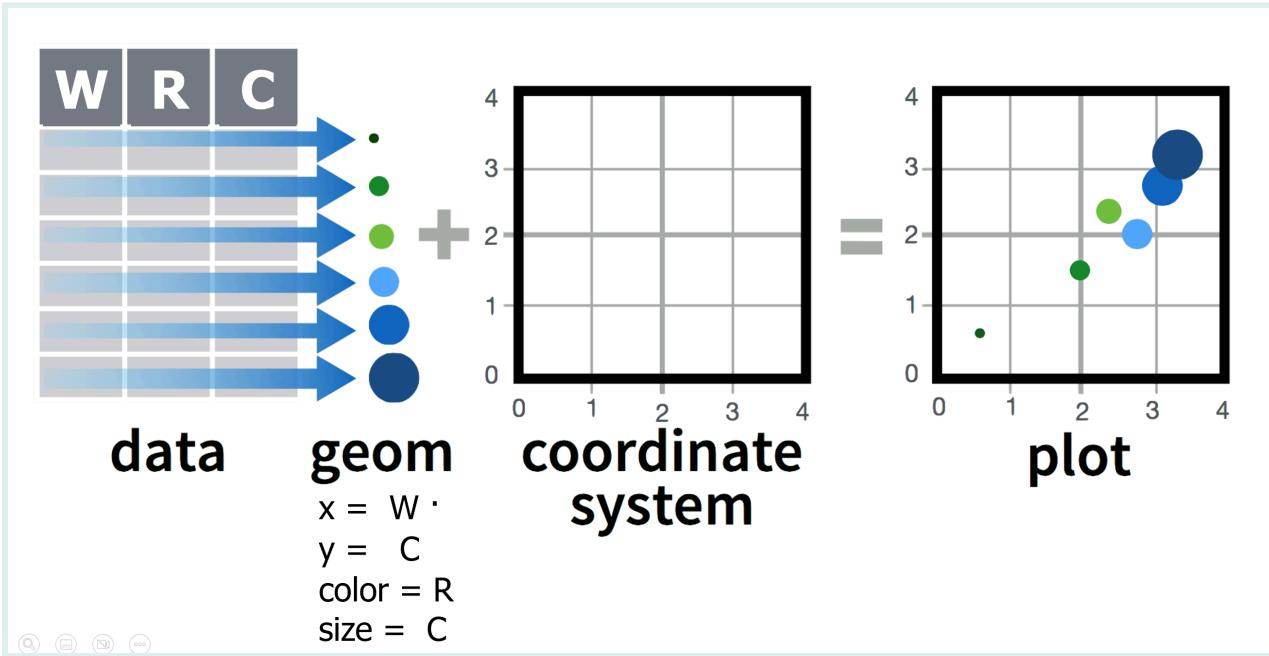
Attributs esthétiques couramment utilisés dans les graphiques `ggplot`.

Revenons à notre diagramme de dispersion original (`cases` vs `week`) :

```
ggplot(data = nigerm96,  
       mapping = aes(x = week,  
                      y = cases)) +  
  geom_point()
```



Nous pouvons ajouter d'autres esthétiques, comme la couleur ou la taille.



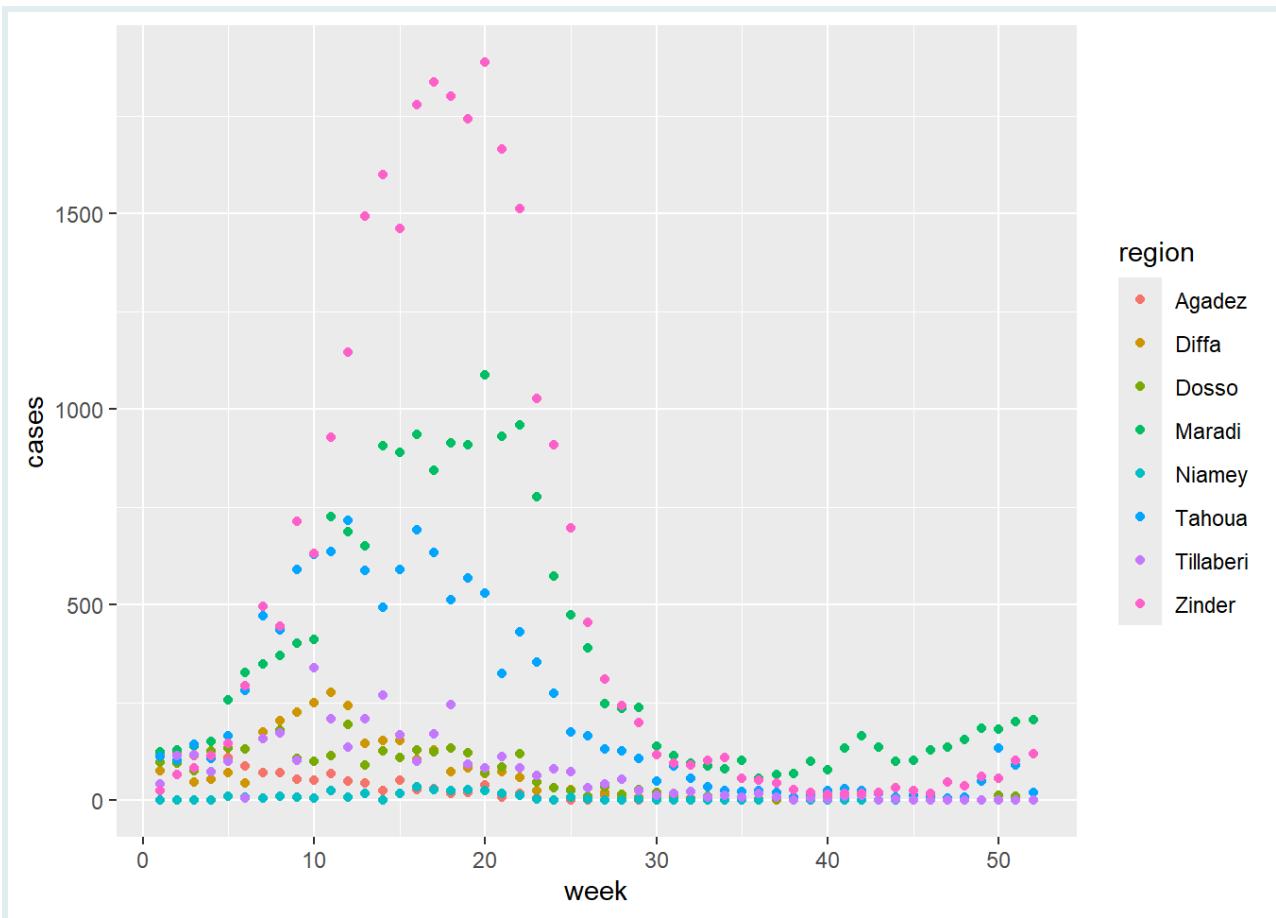
**PRO TIP**

**PRO TIP**

Pour voir la liste complète des esthétiques qui peuvent être utilisées avec une fonction `geom_*` spécifique, référez-vous à la documentation de la fonction. Vous pouvez accéder à cette documentation en appuyant sur F1 après avoir tapé le nom de la fonction, par exemple `geom_point()`. Cela ouvrira l'onglet d'aide où vous pouvez faire défiler jusqu'à la section "Aesthetics", où vous trouverez une liste complète des esthétiques que vous pouvez personnaliser. Une alternative est de taper et d'exécuter ? `geom_point` dans votre onglet Console, ce qui vous dirigera directement vers la documentation de la fonction `geom_point()`.

Ajoutons de la couleur à notre nuage de points. Nous pouvons mapper la variable catégorielle `region` à l'esthétique `color`. Pour cela, nous allons modifier le code originel pour ajouter un nouvel argument à l'intérieur de `mapping = aes()`. Voyons ce qui se passe lorsque nous ajoutons `color = region` à l'intérieur de `aes()`:

```
ggplot(data = nigerm96,
        mapping = aes(x = week,
                      y = cases,
                      color = region)) + # utilise une couleur différente
        pour chaque région
geom_point()
```



Nous avons maintenant un nuage de points coloré ! Chaque point est coloré selon la région à laquelle il appartient. Cela nous permet de mieux distinguer les régions.

Notez que `ggplot()` fournit automatiquement une légende des couleurs à gauche.

#### SIDE NOTE



Les couleurs proviennent de la palette de couleurs arc-en-ciel par défaut de `{ggplot2}`. Dans les cours à venir, nous apprendrons comment personnaliser les échelles de couleurs et les palettes.

En examinant le tracé de couleurs, vous pouvez distinguer la forme classique en cloche des courbes épidémiques montrant une augmentation et une diminution de l'incidence de la rougeole pour chaque région.

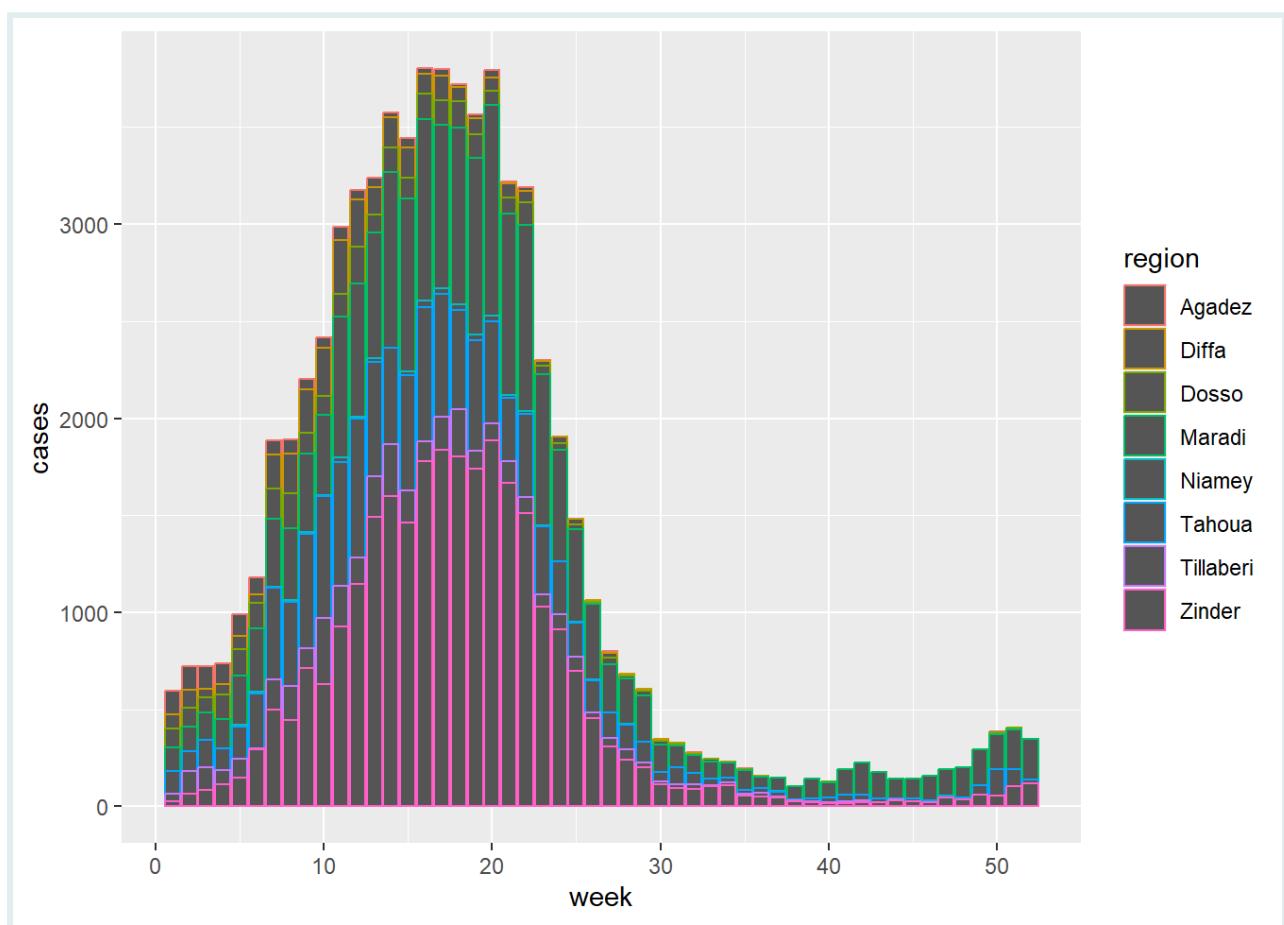
Zinder avait le plus grand nombre de cas et la courbe épidémique la plus raide, suivie de Maradi et Niamey.

Si l'ajout de couleur fournit plus d'informations sur l'incidence de la rougeole au niveau régional, ce graphique est toujours chargé en informations et difficile à interpréter. Un graphique différent serait plus approprié.

Essayons plutôt un graphique à barres, puis un graphique en ligne.

Essayons le même mapping esthétique `color = region` avec `geom_col()` à la place :

```
ggplot(data = nigerm96,
       mapping = aes(x = week,
                      y = cases,
                      color = region)) + # utilise une couleur de contour
                           différente pour chaque région
       geom_col()
```

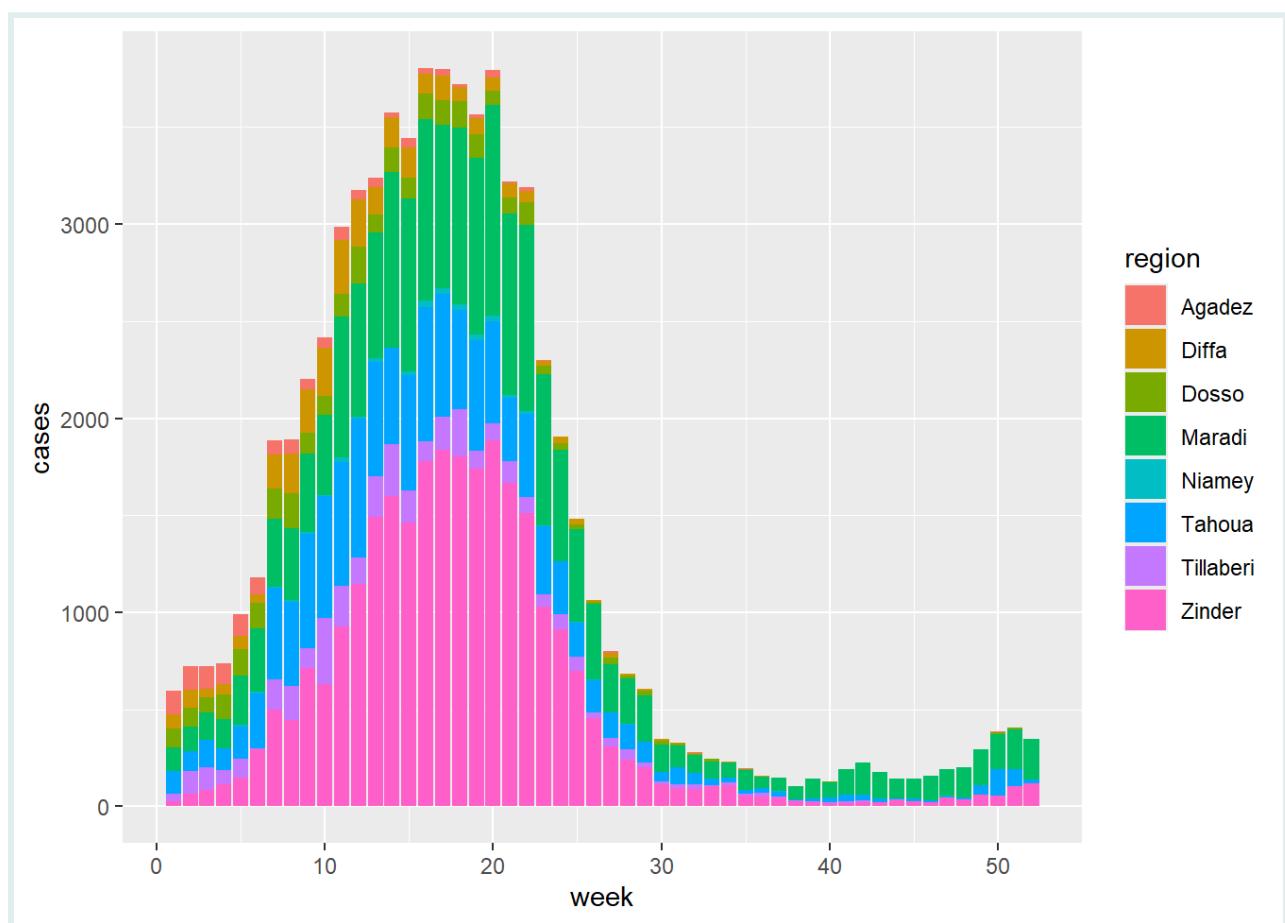


Cela nous donne un graphique à barres empilées, où les barres sont subdivisées en sections plus petites. Cela nous permet de voir la contribution proportionnelle de chaque région (c'est-à-dire que la hauteur ou la longueur de chaque sous-section représente la part de chaque région dans le nombre total de cas par semaine).

Le graphique à barres empilées que vous voyez est délimité par la couleur. Cela est dû au fait que dans {ggplot2}, l'attribut esthétique “color” fait généralement référence à la bordure autour d'une forme. Cet attribut ne s'applique pas aux formes pleines de notre nuage de points créé avec `geom_point()`, mais il est utilisé pour délimiter les barres dans un graphique à barres empilées créé avec `geom_col()`.

Il serait plus intéressant de colorer l'intérieur des barres à la place. Cela se fait en mappant notre variable à l'esthétique `fill`. Nous pouvons copier le code ci-dessus et simplement changer `color` par `fill` à l'intérieur de `aes()`:

```
ggplot(data = nigerm96,  
       mapping = aes(x = week,  
                      y = cases,  
                      fill = region)) + # attribue une couleur de  
# remplissage différente pour chaque région  
geom_col()
```



Voilà! Nous avons ajouté une couleur de remplissage des barres.

Maintenant, entraînez-vous à utiliser l'esthétique `color` avec un nouveau type de graphique : les graphiques linéaires. Les graphiques linéaires sont considérés comme l'un des meilleurs types de graphiques pour les données de séries temporelles.

#### PRACTICE



Utilisez le dataframe `nigerm04` pour créer un graphique linéaire des cas par semaine, coloré par `region`. Mappez `cases` sur l'axe des y,



week sur l'axe des x, et region à la couleur. La fonction geom\_\* pour un graphique linéaire est geom\_line().

## Esthétiques fixes en dehors de aes()

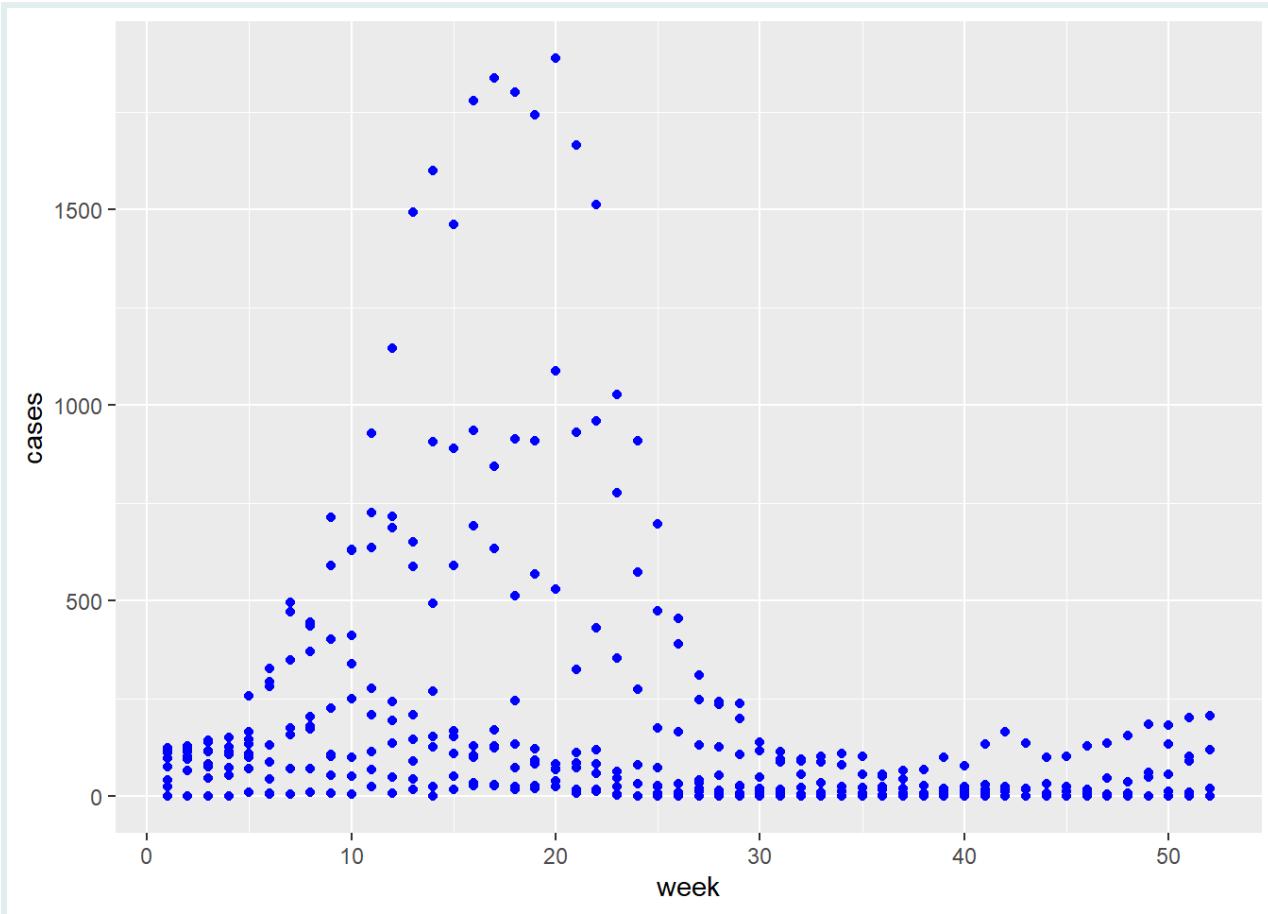
Il est très important de comprendre la différence entre les **mappings esthétiques** et les **esthétiques fixes**. Les principales esthétiques dans ggplot sont : x, y, color, fill, et size, et chacun de ces éléments peut être utilisés soit comme mapping, soit comme valeur fixe. Cela dépend de s'ils apparaissent à l'intérieur ou à l'extérieur de la fonction aes().

Lorsque nous appliquons une esthétique pour modifier les objets géométriques en fonction d'une variable (par exemple, la couleur des points change en fonction de la variable de région), c'est un mapping esthétique. Cela doit toujours être défini à l'intérieur de mapping = aes(), comme nous venons de le faire dans les exemples précédents.

Mais si vous voulez appliquer une modification visuelle à *tous* les objets géométriques de manière égale (par exemple, changer manuellement la couleur de tous les points pour qu'ils soient d'une seule couleur), c'est une esthétique fixe. Nous devons associer les esthétiques fixes à une valeur constante **en dehors** de mapping = aes() et directement à l'intérieur de la fonction geom\_\* - par exemple, geom\_point(color = "NOM\_DE\_LA\_COULEUR").

Ici, changeons la couleur de tous les points de notre nuage de points en bleu :

```
ggplot(data = nigerm96,  
       mapping = aes(x = week,  
                      y = cases)) +  
  geom_point(color = "blue")      # utilise la même couleur pour tous les  
                                # points
```



Cela colore chaque point avec la même couleur R (“blue”). Dans ce graphique, la couleur des points ne représente aucune valeur du dataframe. Notez que les noms de couleurs dans R sont des chaînes de caractères, donc ils doivent être entre guillemets.

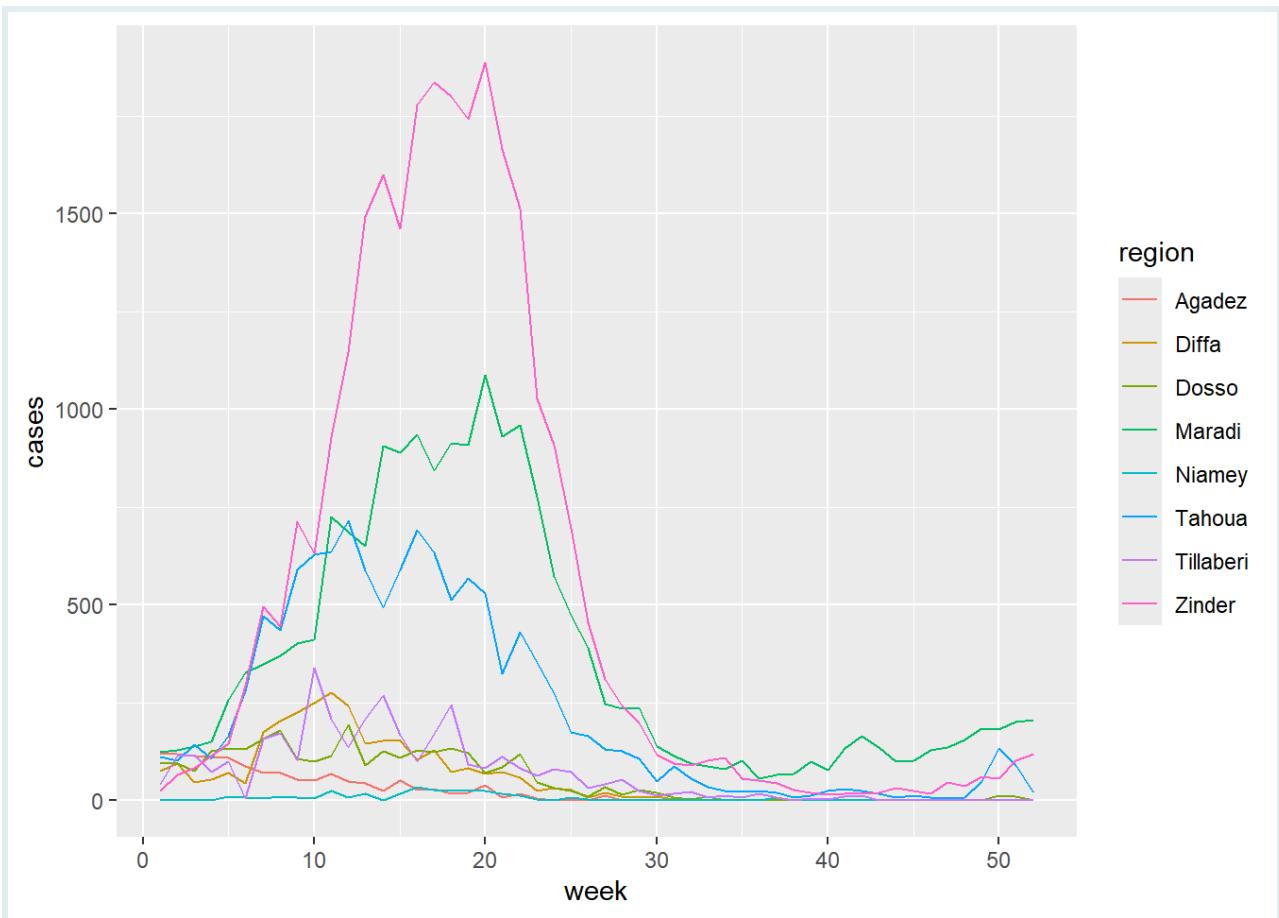
#### SIDE NOTE



Si vous êtes curieux, exécutez `colors()` dans votre console pour voir tous les choix possibles de couleurs dans R ! Pour savoir combien d'options cela représente, exécutez `colors() %>% length()`.

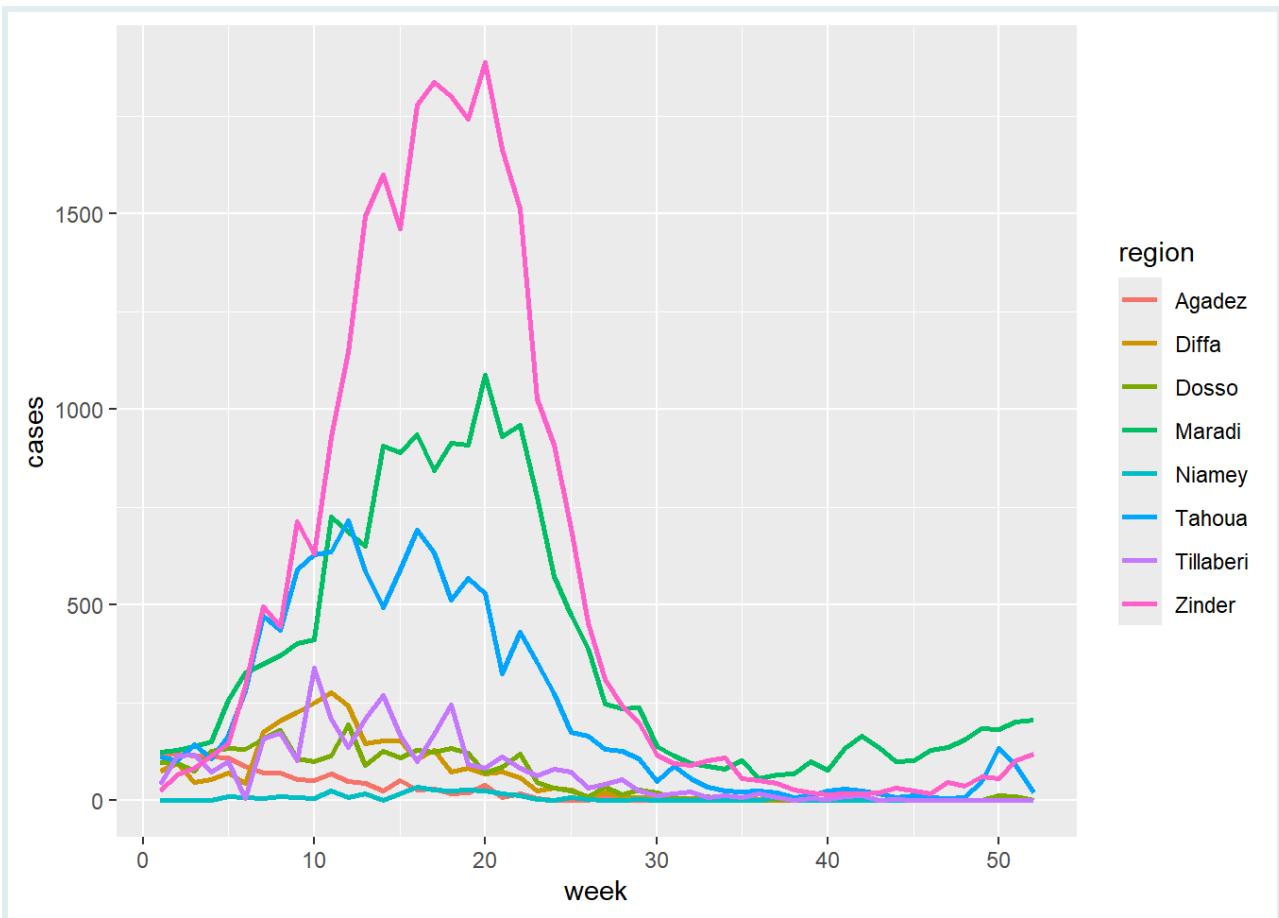
Maintenant, ajoutons une esthétique fixe appelée `size`. La largeur de ligne par défaut utilisée par `geom_line()` est de 0,5 mm, ce qui ressemble à ceci:

```
ggplot(data = nigerm96,
        mapping = aes(x = week,
                      y = cases,
                      color = region)) +
  geom_line()
```



Pour rendre toutes les lignes de notre figure un peu plus épaisses, fixons cette esthétique à 1 mm. Nous faisons cela en ajoutant `size = 1` à l'intérieur de la fonction `geom_line()` :

```
ggplot(data = nigerm96,
       mapping = aes(x = week,
                      y = cases,
                      color = region)) +
  geom_line(size = 1)
```



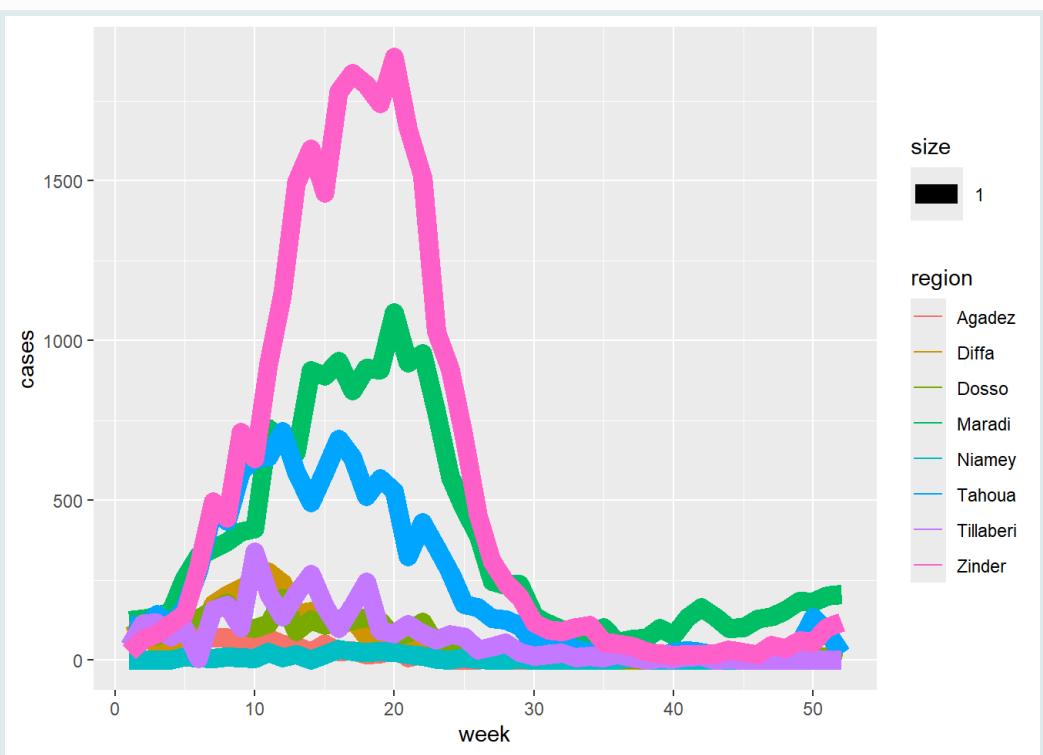
Toutes les lignes du graphique ont été rendues plus épaisses, et la largeur de la ligne est fixée à une valeur constante de 1 mm. Notez qu'ici la valeur de la taille est numérique, donc elle ne doit pas être entre guillemets.

#### WATCH OUT



N'oubliez pas que les esthétiques fixes sont manuellement définies à une valeur constante (et non à une variable), et vont directement dans la fonction `geom_*`, et non pas à l'intérieur de `aes()`. Si vous essayez de mettre une esthétique fixe dans `aes()`, vous pourriez obtenir un résultat étrange. Par exemple, essayons de déplacer l'esthétique `size = 1` de `geom_line()` à `aes()` pour voir comment cela peut mal tourner :

```
ggplot(data = nigerm96,
       mapping = aes(x = week,
                     y = cases,
                     color = region,
                     size = 1)) +      # placement
  INCORRECT
  geom_line()
```



`aes()` est une fonction de mapping qui modifie les graphiques en fonction des variables. Comme il n'y a pas de variable appelée “1” dans le dataframe `nigerm96`, `aes()` ne peut pas traiter ou mapper correctement cette esthétique.

Entraînez-vous à utiliser `fill` comme esthétique fixe pour un graphique à barres.



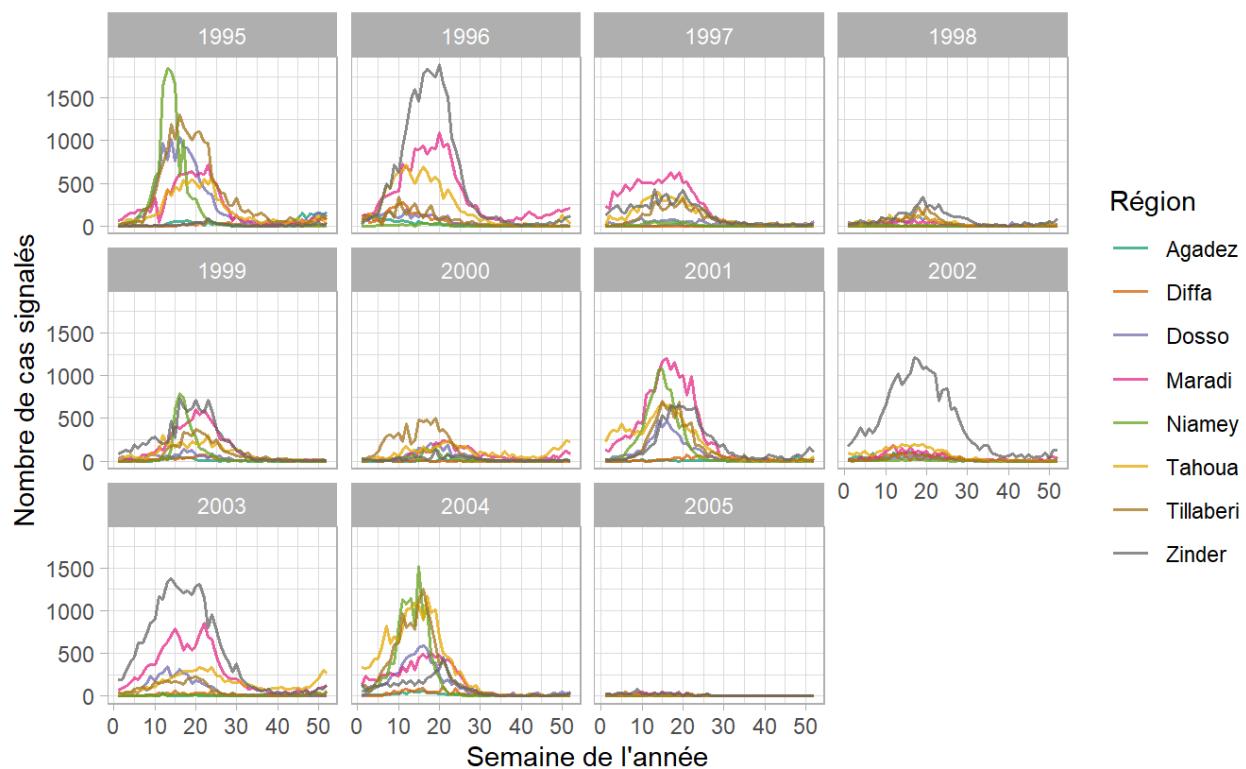
Utilisez le dataframe `nigerm04` pour créer un graphique à barres des cas hebdomadaires, et remplissez toutes les barres avec la même couleur. Mappez `cases` sur l'axe des y, `week` sur l'axe des x, et fixez l'esthétique `color` des barres à la couleur R “hotpink”.

## Les autres couches

Pour ce premier cours, nous nous sommes concentrés sur l'utilisation des trois couches essentielles. Au fur et à mesure que vous utiliserez {ggplot2}, vous découvrirez aussi les autres couches. Bientôt, vous pourrez créer des graphiques plus complexes, comme celui-ci :

## Incidence de la rougeole au Niger par saison

Rapport hebdomadaire au niveau régional (1995-2005)



### RECAP



Pour créer un graphique complet avec `ggplot`, vous devez d'abord fournir un jeu de données à l'aide de l'argument `data` de `ggplot()`. Ensuite, définissez les variables que vous souhaitez visualiser et mappez-les aux esthétiques à l'intérieur de la fonction `aes()` en utilisant l'argument `mapping` de `ggplot()`. Enfin, créez une nouvelle couche avec le signe `+` et spécifiez le type de graphique que vous souhaitez à l'aide de la fonction `geom_*` appropriée. Vous pouvez utiliser ce modèle de code et l'ajuster pour créer différents types de graphiques avec `ggplot` :

```
ggplot(data = NOM_DF,
       mapping = aes(AES1 = VAR1,
                     AES2 = VAR2,
                     AES3 = VAR3,
                     ...)) +
  geom_FONCTION()
```

---

## Les acquis

1. Comprendre la **Grammaire des Graphiques (Grammar of Graphics)**, le cadre de visualisation de données sur lequel se base le package **{ggplot2}**.
2. Nommer et décrire les 3 couches (layers) essentielles à la construction d'un graphique : **données, esthétiques, et géométries**.
3. Écrire correctement le code pour **construire un graphique avec ggplot** en fournissant les 3 couches essentielles à la fonction **ggplot()**.
4. Créer différents types de graphiques comme les **graphiques de dispersion, les graphiques linéaires et les graphiques à barres**.
5. Ajouter ou modifier les propriétés visuelles d'un graphique tels que la **couleur** ou la **taille**.
6. Distinguer entre les **esthétiques mappées** et les **esthétiques fixes** et savoir les utiliser.

---

## Contributeurs

Les membres suivants ont contribué à ce cours :



**JOY VAZ**

R Developer and Instructor, the GRAPH Network  
Loves doing science and teaching science



**IMANE BENSOUDA KORACHI**

R Developer and Instructor, the GRAPH Network



**SABINA RODRIGUEZ VELÁSQUEZ**

Project Manager and Scientific Collaborator, The GRAPH Network  
Infectiously enthusiastic about microbes and Global Health

---

## Références

Le contenu de ce cours a été en partie adapté des sources suivantes :

- Blake, Alexandre, Ali Djibo, Ousmane Guindo, and Nita Bharti. 2020. “Investigating Persistent Measles Dynamics in Niger and Associations with Rainfall.” *Journal of The Royal Society Interface* 17 (169) : 20200480. <https://doi.org/10.1098/rsif.2020.0480>.
- Cmprince. *Administrative divisions of Niger: Departments and Regions*. 29 October 2017. Wikimedia Commons. Consulté le 14 octobre 2022. [https://commons.wikimedia.org/wiki/File:Niger\\_administrative\\_divisions.svg](https://commons.wikimedia.org/wiki/File:Niger_administrative_divisions.svg)
- DeBruine, Lisa, and Dale Barr. 2022. *Chapter 3 Data Visualisation | Data Skills for Reproducible Research*. <https://psyteachr.github.io/reprores-v3/ggplot.html>.
- Franke, Michael. n.d. *6 Data Visualization | An Introduction to Data Analysis*. Consulté le 12 octobre 2022. <https://michael-franke.github.io/intro-data-analysis/Chap-02-02-visualization.html>.
- Geography Now, dir. 2019. *Geography Now! NIGER*. <https://www.youtube.com/watch?v=AHeq99pojLo>.
- Giroux-Bougard, Xavier, Maxwell Farrell, Amanda Winegardner, Étienne Low-Decarie and Monica Granados. 2020. *Workshop 3: Introduction to Data Visualisation with Ggplot2*. <http://r.qcbs.ca/workshop03/book-en/>.
- Ismay, Chester, and Albert Y. Kim. 2022. *A ModernDive into R and the Tidyverse*. <https://moderndive.com/>.
- Kabacoff, Rob. 2020. *Data Visualization with R*. <https://rkabacoff.github.io/datavis/>.
- Lisa DeBruine. 2020. *Basic Plots*. <https://www.youtube.com/watch?v=tOFQFPRgZ3M>.
- Pius, Ewen Harrison and Riinu. n.d. *R for Health Data Science*. Consulté le 11 octobre 2022. [https://argoshare.is.ed.ac.uk/healthyr\\_book/](https://argoshare.is.ed.ac.uk/healthyr_book/).
- Prabhakaran, Selva. 2016. “How to Make Any Plot in Ggplot2? | Ggplot2 Tutorial.” 2016. <http://r-statistics.co/ggplot2-Tutorial-With-R.html>.

This work is licensed under the [Creative Commons Attribution Share Alike](#) license.



# Les graphiques de dispersion et lignes de lissage

February 2024



Introduction	.....
Objectifs d'apprentissage	.....
Maladies diarrhéiques chez les enfants au Mali	.....
Graphiques de dispersion avec <code>geom_point()</code>	.....
Modifier les esthétiques	.....
Mapping des données aux esthétiques	.....
Définir les esthétiques fixes	.....
Ajouter une ligne de tendance	.....
En résumé	.....
Contributeurs	.....
Références	.....

---

## Introduction

Les **diagrammes de dispersion** ou **nuages de points** (en anglais, **scatter plots**) sont des outils visuels puissants permettant d'explorer la **relation** entre deux variables quantitatives.

Ils sont très populaires dans le domaine de la visualisation de données, car ils offrent une vue instantanée de la manière dont une variable quantitative évolue par rapport à une autre.

Les diagrammes de dispersion permettent aussi de représenter plusieurs relations en associant une variable supplémentaire à des propriétés esthétiques, comme la couleur des points.

De plus, les tendances et les relations sur un diagramme de dispersion peuvent être rendues plus claires en ajoutant une ligne de lissage.

Nous allons utiliser ggplot pour faire tout cela et plus encore. C'est parti !

## Objectifs d'apprentissage

1. Visualiser les relations entre des variables quantitatives en utilisant des **diagrammes de dispersion** avec `geom_point()`.
2. Utiliser `color` comme argument esthétique pour mapper les variables d'un dataset sur les points.
3. Changer la taille, la forme, la couleur, le remplissage et l'opacité des objets géométriques en définissant des **esthétiques fixes**.
4. Ajouter une **ligne de tendance** à un diagramme de dispersion avec `geom_smooth()`.

## Maladies diarrhéiques chez les enfants au Mali

Nous allons utiliser les données recueillies lors d'une étude observationnelle prospective sur la **diarrhée aiguë chez les enfants** âgés de 0 à 59 mois. L'étude a été menée au Mali au début de l'année 2020.

Le dataset complet est disponible sur [Dryad](#), et l'article peut être consulté [ici](#).



### VOCAB

Une étude prospective suit l'évolution d'un événement (par exemple, le développement d'une maladie ou d'autres problèmes de santé) pendant la période de l'étude, et examine son lien avec d'éventuels facteurs de risque ou de protection.

Explorons ce dataset. Il comprend des variables démographiques, physiologiques, cliniques, socioéconomiques et géographiques. Chaque ligne correspond à un patient interrogé.

Nous allons commencer par visualiser la relation entre les deux variables quantitatives suivantes :

1. `age_months` : l'**âge** du patient en mois sur l'axe des **x** et
2. `viral_load` : la **charge virale** du patient sur l'axe des **y**.

## Graphiques de dispersion avec `geom_point()`

Nous allons explorer la relations entre des variables quantitatives du dataframe `malidd`.

Commençons par exécuter le code pour créer le graphique de dispersion souhaité, tout en gardant à l'esprit le cadre GG. Examinons le code et décomposons-le couche par couche.

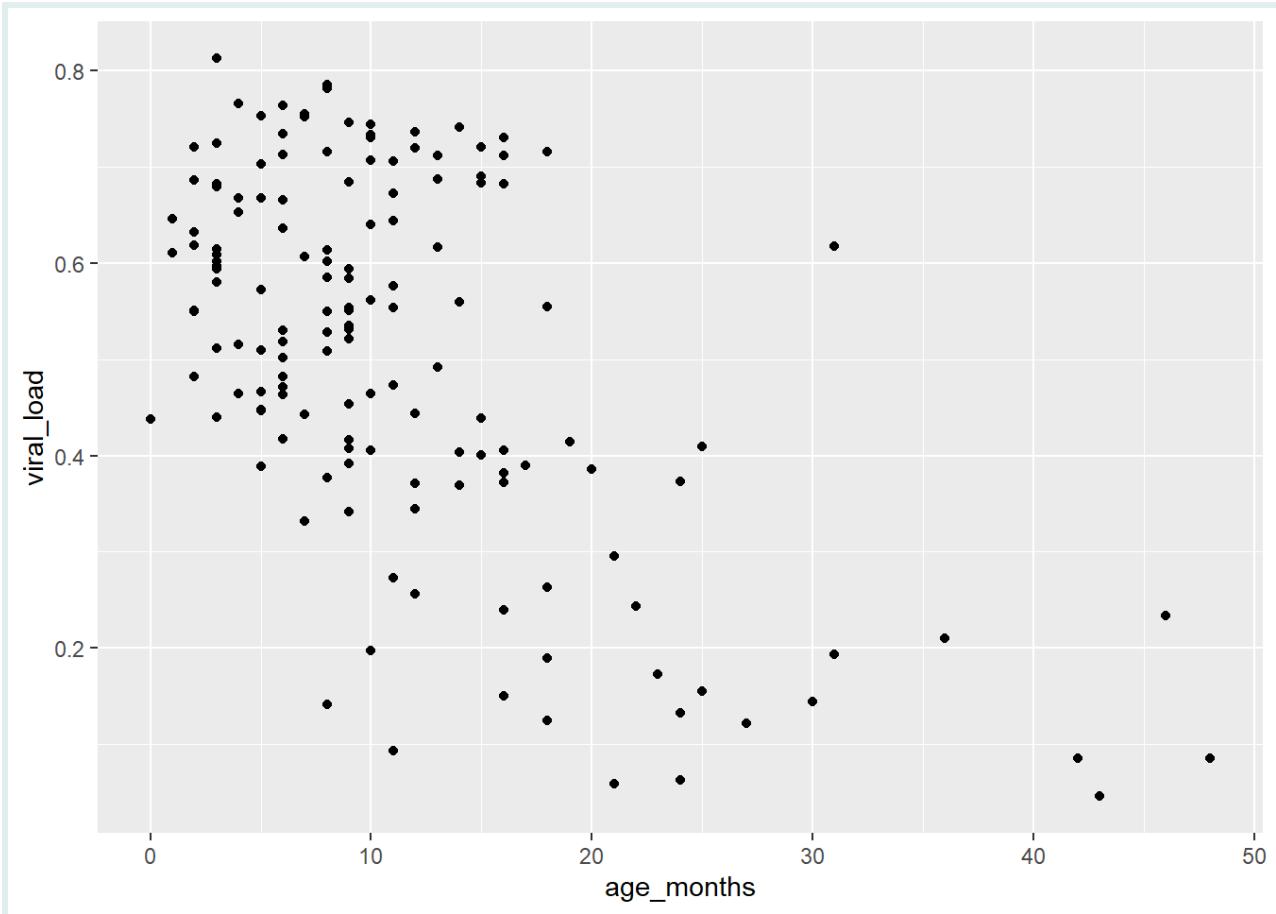
N'oubliez pas d'inclure les deux premières couches à l'intérieur de la fonction `ggplot()` :

1. Nous utilisons le dataframe `malidd` comme argument `data`, en spécifiant `data = malidd`.
2. Dans la fonction `aesthetics` de l'argument `mapping`, nous définissons les variables à représenter en utilisant `mapping = aes(x = age_months, y = viral_load)`. Plus précisément, la variable `age_months` est associée à l'axe des `x`, tandis que la variable `viral_load` est associée à l'axe des `y`.

Ensuite, nous ajoutons une nouvelle couche à l'aide de l'opérateur `+` et la fonction `geom_*` (). Pour un graphique de dispersion, les objets géométriques nécessaires sont des points, donc nous ajoutons `geom_point()`.

En exécutant le code suivant, vous obtiendrez le graphique de dispersion illustré ci-dessous :

```
# Graphique de dispersion basique de la charge virale en fonction de l'âge
ggplot(data = malidd,
        mapping = aes(x = age_months,
                      y = viral_load)) +
  geom_point()
```



Cela suggère que la charge virale **diminue** généralement avec l'âge.



- En utilisant le dataframemalidd, créez un graphique de dispersion montrant la relation entre l'âge et la taille (height\_cm).

## Modifier les esthétiques

Une esthétique est une propriété visuelle des objets géométriques (`geom`) de votre graphique. Les esthétiques incluent entre autres la taille, la forme ou la couleur de vos points. Vous pouvez présenter un point de différentes manières en modifiant les valeurs de ses propriétés esthétiques.

Rappelez-vous, il y a deux manières de modifier les propriétés esthétiques de vos éléments géométriques `geom` (ici, des points) :

1. **Mapper les esthétiques aux variables de vos données**, en utilisant la fonction `aes()` dans l'argument `mapping` pour associer le nom de l'esthétique à une variable à afficher.
2. **Définir manuellement les esthétiques** en spécifiant l'esthétique en tant qu'argument de la fonction `geom_*`(), *en dehors* de `aes()`. Dans ce cas, l'esthétique ne véhicule pas d'information sur une variable, mais sert uniquement à modifier l'apparence du graphique.

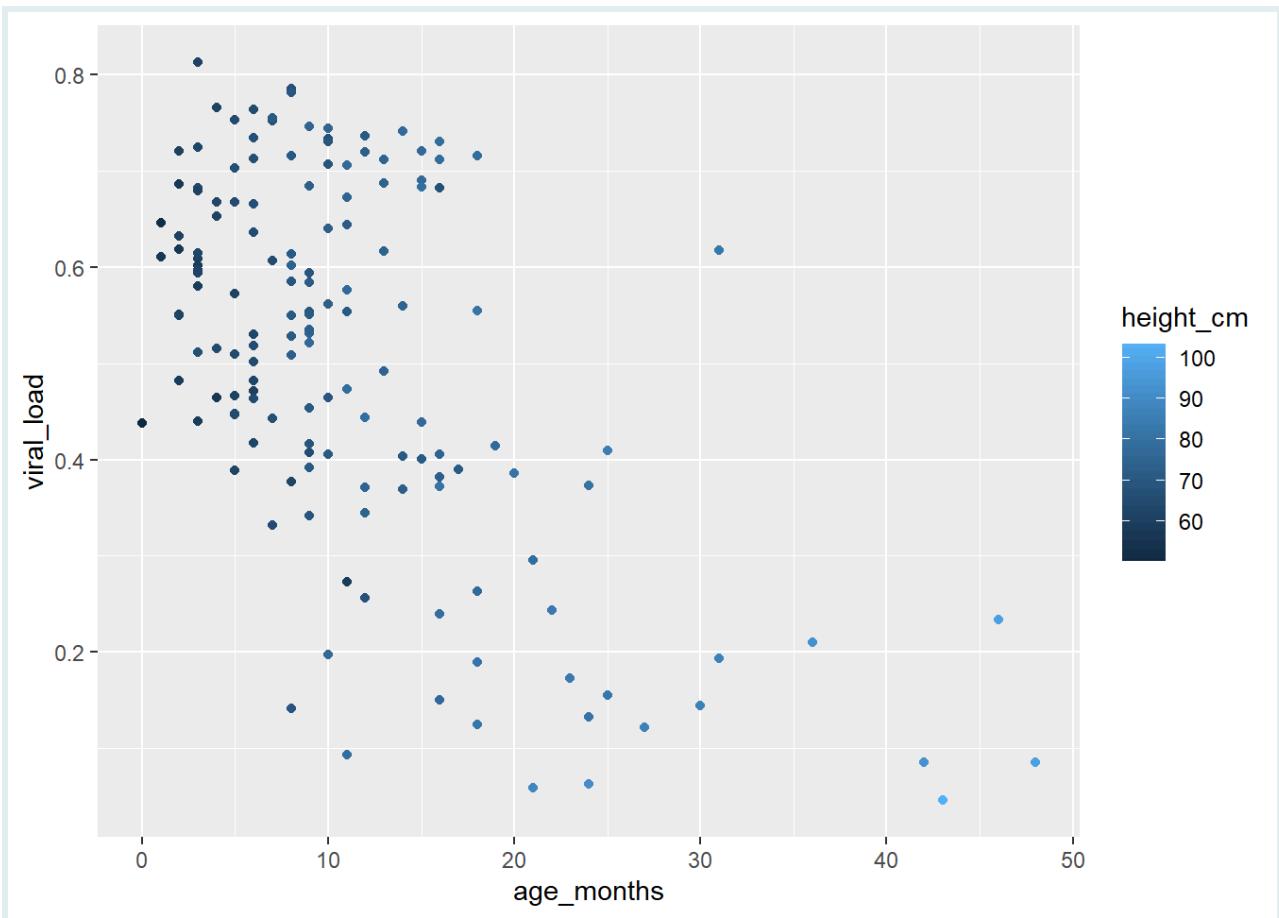
## Mapping des données aux esthétiques

En plus de mapper des variables aux axes des **x** et des **y** comme nous l'avons fait précédemment, les variables peuvent être mappées à la couleur, à la forme, à la taille, à l'opacité et à d'autres caractéristiques visuelles du `geom`. Cela permet de superposer plusieurs observations sur un seul graphique.

Pour mapper une variable à une esthétique, il faut associer le nom de l'esthétique au nom de la variable à l'intérieur de `aes()`. De cette façon, vous pouvez visualiser une troisième variable dans un graphique de dispersion à deux dimensions en la mappant à une nouvelle esthétique.

Par exemple, supposons que nous voulons montrer comment la taille (`height_cm`) varie avec l'âge et la charge virale. Nous pouvons mapper la variable `height_cm` à la couleur de nos points de la manière suivante :

```
ggplot(data = malidd,  
       mapping = aes(x = age_months,  
                      y = viral_load)) +  
  geom_point(mapping = aes(color = height_cm))
```



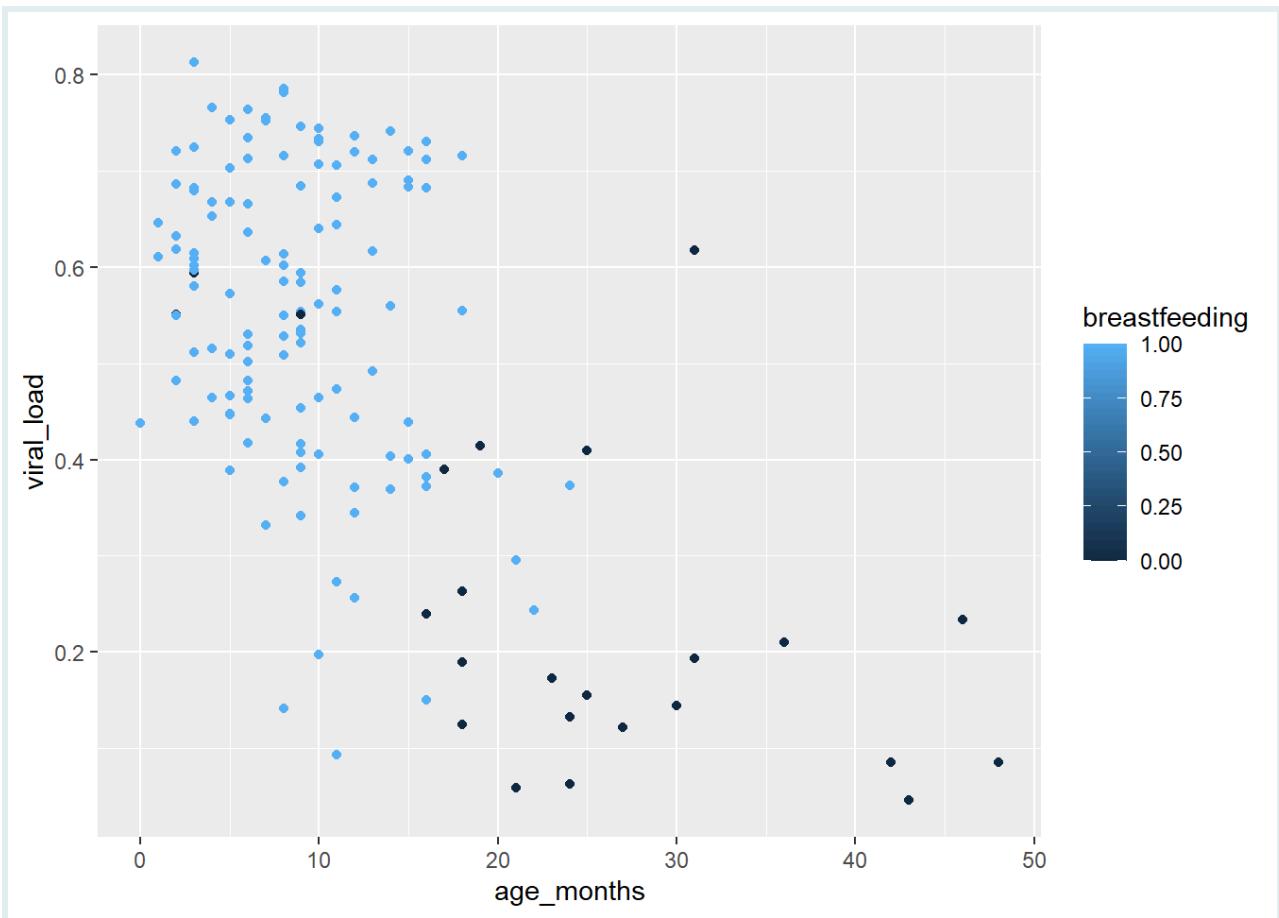
Comme vous pouvez le constater, {ggplot2} a automatiquement assigné les valeurs de notre variable à une esthétique, un processus connu sous le nom de **scaling**. {ggplot2} ajoutera également une légende pour expliquer quelles couleurs correspondent à quelles valeurs.

Dans cet exemple, les points sont colorés par différentes nuances de bleu, avec des couleurs plus foncées représentant des valeurs plus basses.

Cela nous indique que la taille augmente avec l'âge, comme on pouvait s'y attendre.

Au lieu d'une variable continue comme `height_cm`, nous pouvons également mapper une variable binaire comme `breastfeeding`, pour indiquer les enfants qui sont allaités et ceux qui ne le sont pas :

```
ggplot(data = malidd,
       mapping = aes(x = age_months,
                     y = viral_load)) +
  geom_point(mapping = aes(color = breastfeeding))
```



Nous obtenons la même échelle de couleurs que celle obtenue avec la variable `height_cm`. Cette échelle de couleurs suggère une variabilité continue, ce qui n'est pas idéal dans notre cas, car notre variable `breastfeeding` ne peut prendre que deux valeurs distinctes : 0 ou 1.

Cela est dû à la classe de la variable `breastfeeding` dans `malidd` :

```
class(malidd$breastfeeding)
```

```
## [1] "numeric"
```

Même si cette variable est numérique, elle ne peut prendre que deux valeurs. Par conséquent, une échelle de couleurs continue n'est pas la représentation la plus appropriée pour ce cas.

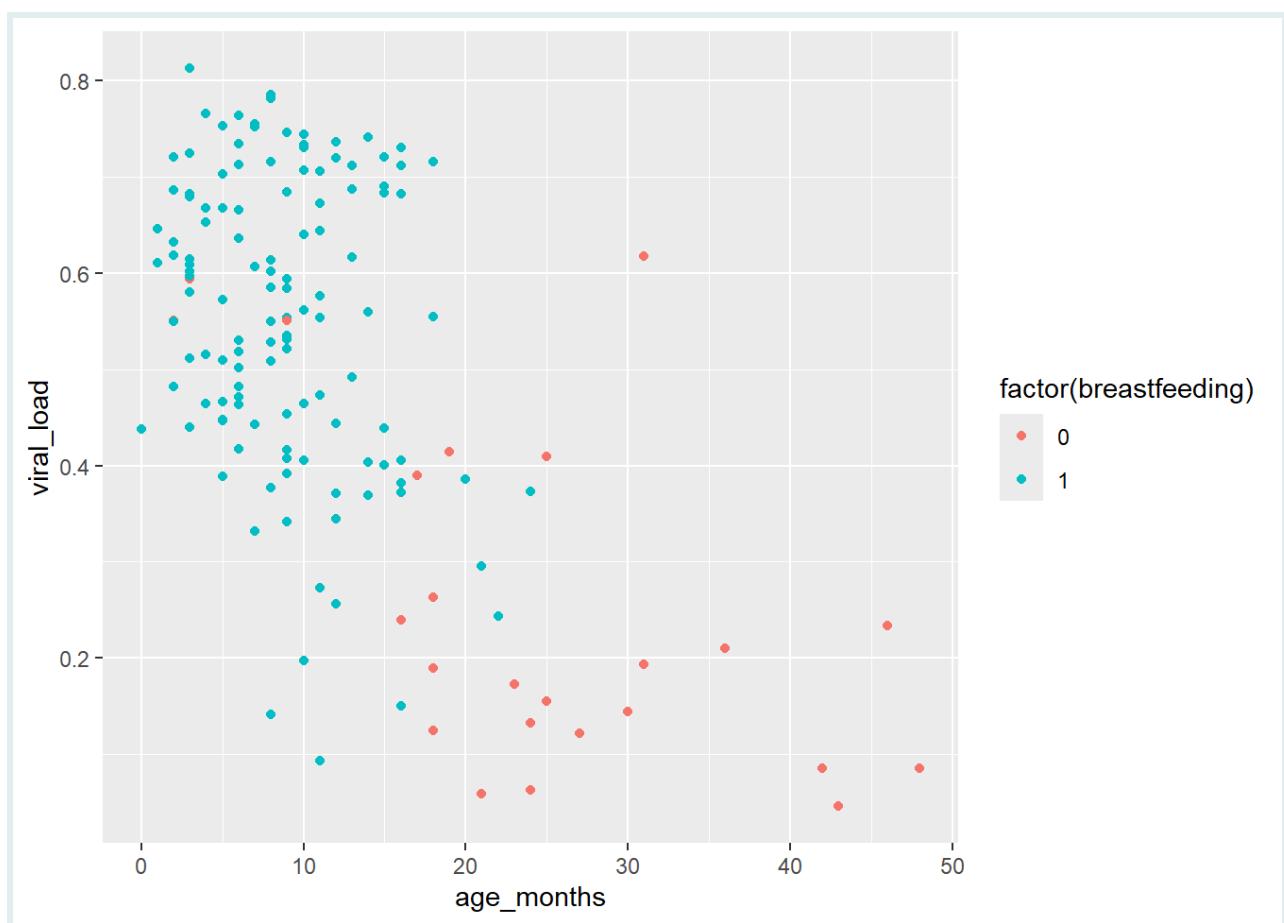
Dans des cas comme celui-ci, nous ajoutons la fonction `factor()` autour de la variable `breastfeeding` pour dire à `ggplot()` de traiter la variable comme un facteur. Voyons ce qui se passe lorsque nous faisons cela :

```
ggplot(data = malidd,
       mapping = aes(x = age_months,
```

```

y = viral_load)) +
geom_point(mapping = aes(color = factor(breastfeeding)))

```



Lorsque la variable est traitée comme un facteur, les couleurs attribuées sont nettement distinctes. Avec les facteurs, {ggplot2} attribue automatiquement un niveau unique de l'esthétique (dans ce cas, une couleur unique) à chaque valeur de la variable. C'est ce qui s'est produit avec la variable `region` du dataframe `malidd` que nous avons utilisé dans la leçon précédente.

Ce graphique met en évidence une relation claire entre l'âge et l'allaitement maternel, comme on pouvait s'y attendre. Les enfants sont susceptibles d'arrêter l'allaitement vers 20 mois. Dans cette étude, aucun enfant de 25 mois ou plus n'était allaité.

L'ajout de couleurs à notre diagramme de dispersion nous a permis de visualiser une **troisième variable** en plus de la relation entre l'âge et la charge virale. Cette troisième variable peut être soit discrète (comme dans le cas de l'allaitement), soit continue.



- Créez un graphique de dispersion à partir du dataframe `malidd`, montrant la relation entre l'âge et la charge virale, et mappez



une troisième variable `freqresp` à la couleur.

- Recréez le même graphique de dispersion âge vs. taille, mais cette fois, mappez la variable binaire `fever` à la couleur des points. N'oubliez pas que `fever` doit être traité comme un facteur.

## Définir les esthétiques fixes

Les esthétiques fixes sont constantes sur tout le graphique et ne varient pas en fonction des données. Pour ajouter une esthétique fixe, nous l'ajoutons comme argument direct de la fonction `geom_*` (*en dehors* de `mapping = aes()`).

Voici certains des arguments esthétiques que nous pouvons placer directement dans `geom_point()` pour apporter des modifications visuelles aux points de notre graphique de dispersion :

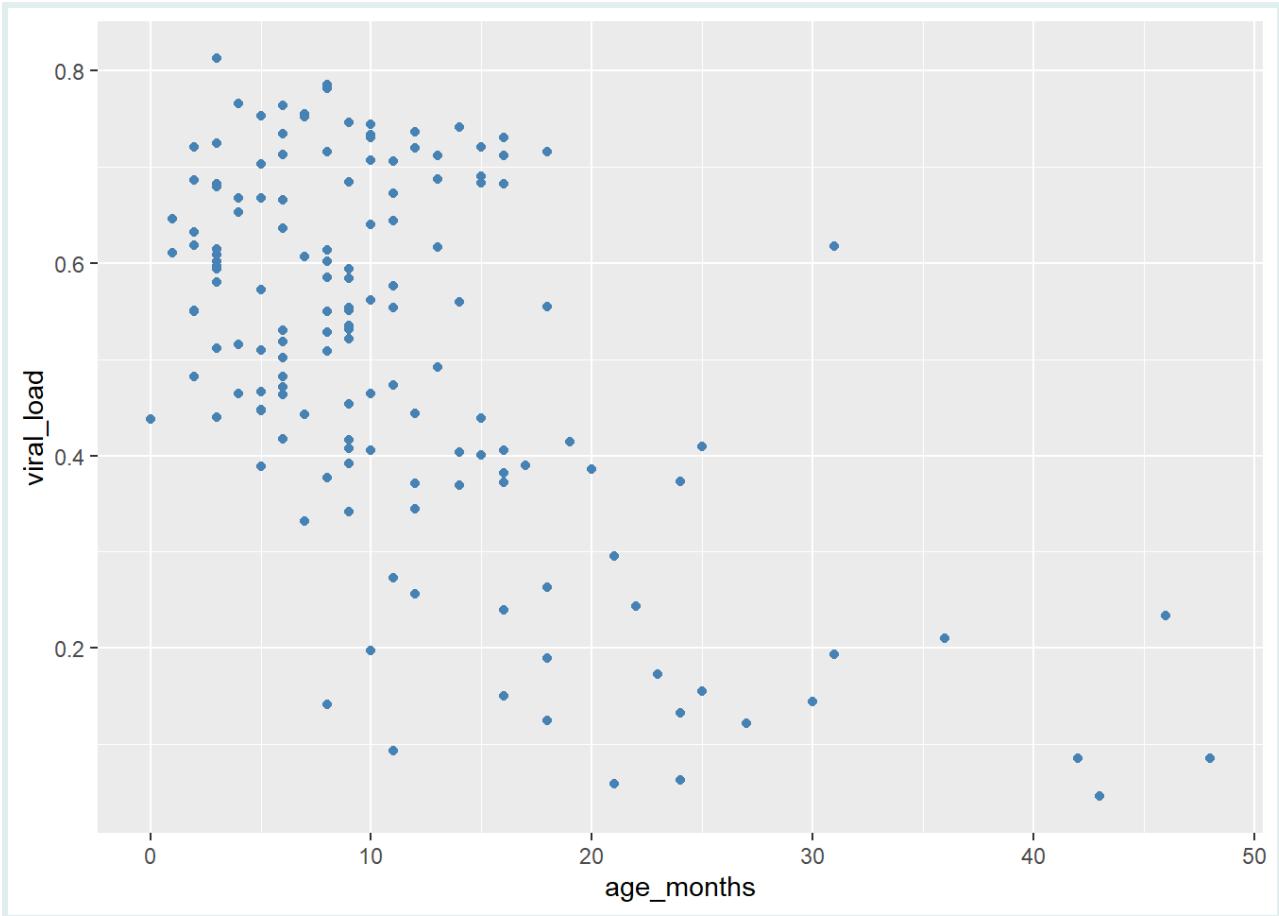
- `color` - couleur du point ou couleur du contour du point
- `size` - taille du point
- `alpha` - opacité du point
- `shape` - forme du point
- `fill` - couleur de remplissage du point (ne s'applique que si le point a un contour)

Pour utiliser ces options afin de créer un graphique de dispersion plus attractif, vous devrez choisir une valeur appropriée pour chaque argument esthétique, comme le montrent les exemples ci-dessous.

### Modifier `color`, `size` et `alpha`

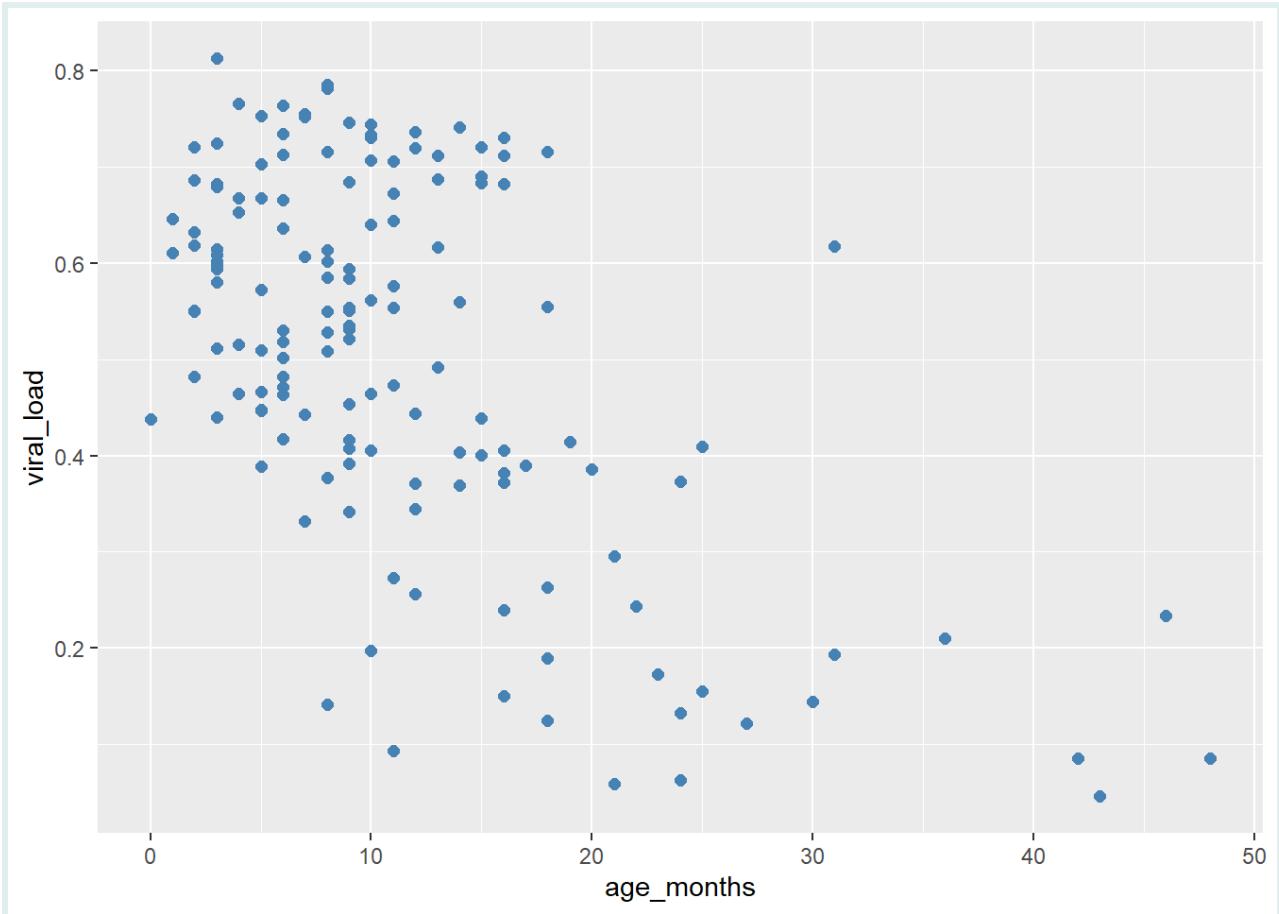
Pour modifier la couleur des points en leur attribuant une valeur fixe, il faut positionner l'argument `color` directement dans `geom_point()`. La couleur que nous choisissons doit être une chaîne de caractères que R reconnaît comme une couleur. Pour définir la couleur des points en bleu acier (“steelblue”), vous pouvez utiliser le code suivant :

```
# Modifier le graphique de dispersion d'origine en définissant `color =
  "steelblue"`
ggplot(data = malidd,
       mapping = aes(x = age_months,
                     y = viral_load)) +
  geom_point(color = "steelblue")      # définir la couleur
```



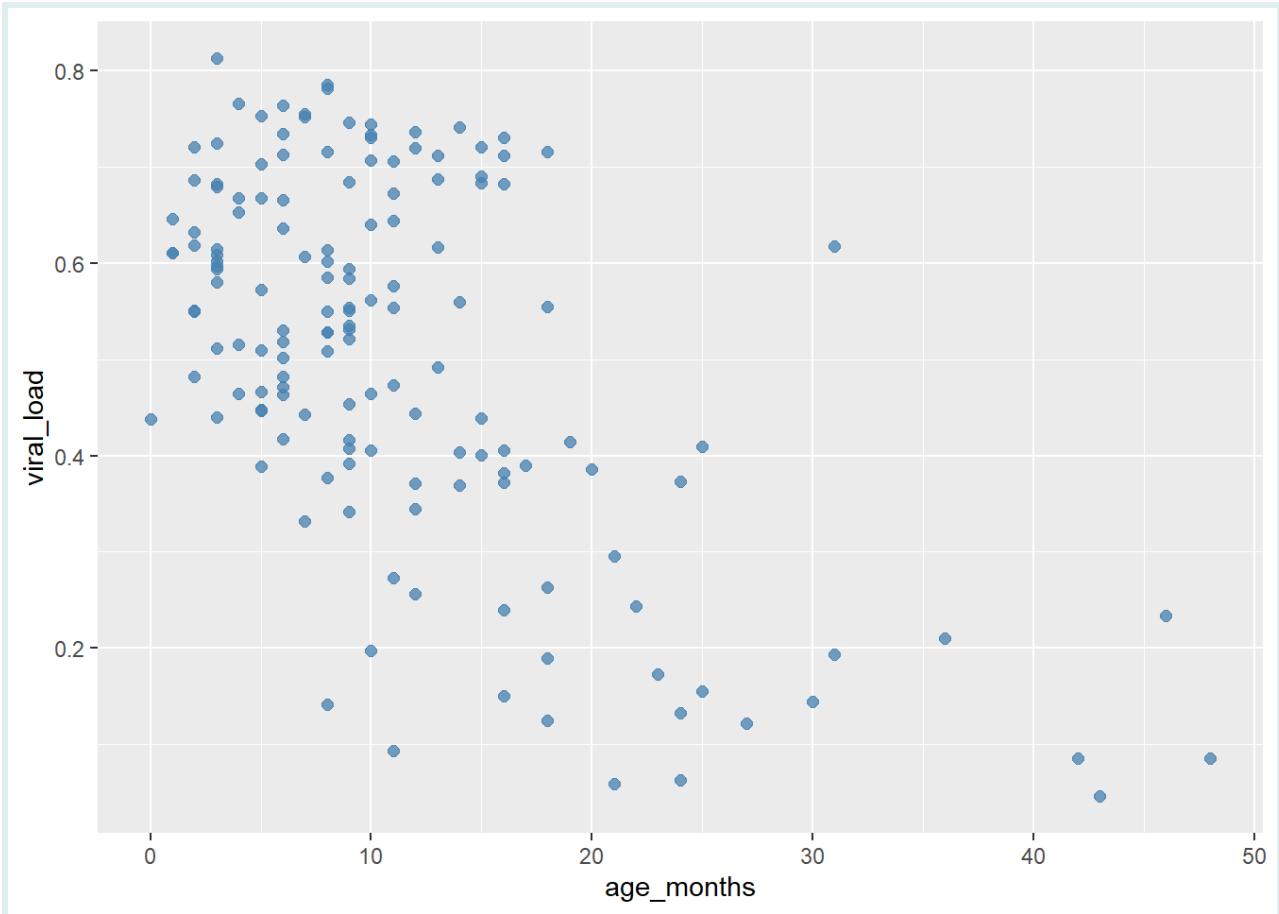
En plus de changer la couleur par défaut, nous allons maintenant modifier l'esthétique `size` des points en lui attribuant un nombre fixe (en millimètres). La taille par défaut est de 1 mm, nous allons donc choisir une valeur plus grande :

```
# Définir la taille à 2 mm en ajoutant `size = 2`
ggplot(data = malidd,
        mapping = aes(x = age_months,
                      y = viral_load)) +
  geom_point(color = "steelblue",           # définir la couleur
             size = 2)                  # définir la taille (mm)
```



L'esthétique `alpha` contrôle le niveau d'opacité des geom. `alpha` est une valeur numérique qui varie de 0 (complètement transparent) à la valeur par défaut de 1 (complètement opaque). Nous allons réduire l'opacité des points en diminuant la valeur de `alpha` :

```
# Définir l'opacité à 75% en ajoutant `alpha = 0.75`
ggplot(data = malidd,
        mapping = aes(x = age_months,
                      y = viral_load)) +
  geom_point(color = "steelblue",           # définir la couleur
             size = 2,                  # définir la taille (mm)
             alpha = 0.75)              # définir le niveau d'opacité
```



En rendant les points semi-transparents, nous pouvons voir où les points se chevauchent. C'est particulièrement utile pour les graphiques de dispersion où il y a beaucoup de **superposition** de points.

Souvenez-vous, changer la couleur, la taille, ou l'opacité de nos points ici ne transmet aucune information supplémentaire sur les données - ce sont des choix de conception que nous faisons pour rendre nos graphiques plus esthétiques.



- Créez un graphique de dispersion avec les mêmes variables que dans l'exemple précédent, mais changez la couleur des points à `cornflowerblue`, augmentez la taille des points à 3 mm et réglez l'opacité à 60%.

### Modifier `shape` et `fill`

Nous pouvons modifier l'apparence des points d'un graphique de dispersion avec l'esthétique `shape`.

Pour fixer la forme de vos `geom` à une valeur spécifique, vous pouvez définir l'esthétique `shape` au nombre correspondant à la forme souhaitée.

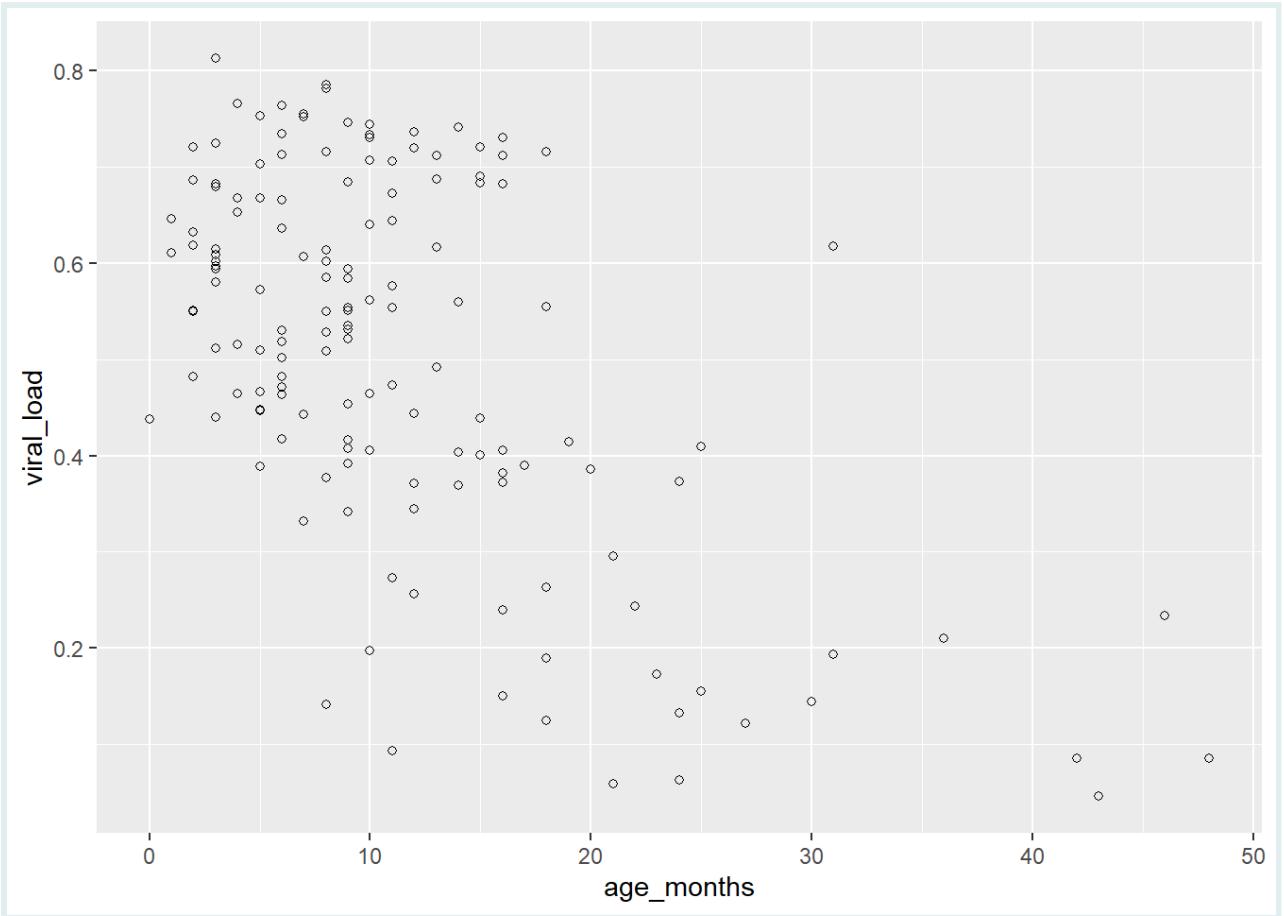
Dans {ggplot2}, les formes des points sont définies par les numéros 0 à 24 :

0	1	2	3	4
□	○	△	+	×
5	6	7	8	9
◇	▽	▣	✳	❖
10	11	12	13	14
⊕	⊗	田	⊗	□
15	16	17	18	19
■	●	▲	◆	●
20	21	22	23	24
●	●	■	◆	▲

Notez que les formes 21 à 24 peuvent être remplies (`fill`) et avoir une couleur de contour (`color`), tandis que les autres formes ne sont sensibles qu'à la couleur de contour `color`.

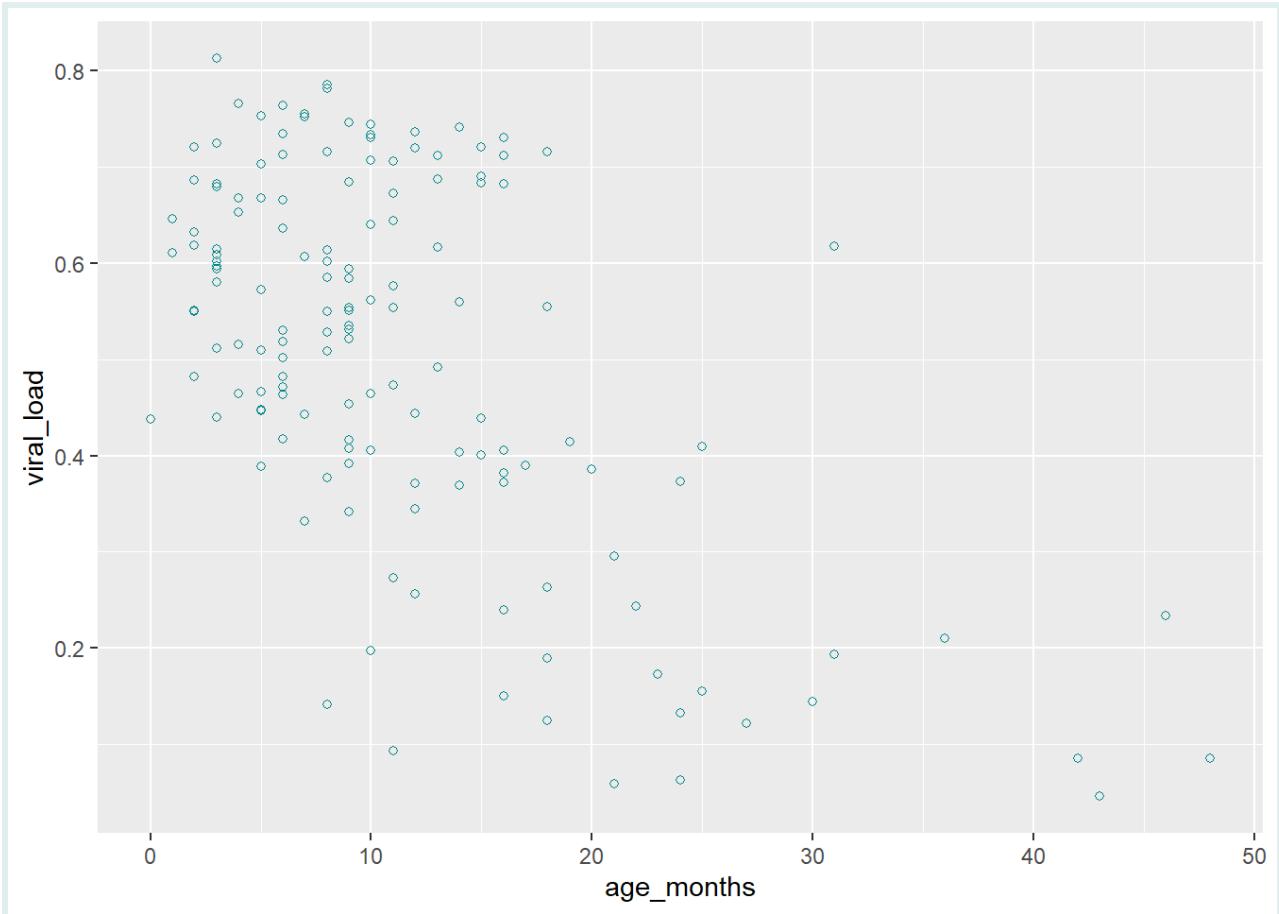
Tout d'abord, modifions notre graphique de dispersion d'origine en choisissant une forme de point remplissable :

```
# Définir la forme à des cercles remplissables en ajoutant `shape = 21`  
  
ggplot(data = malidd,  
        mapping = aes(x = age_months,  
                      y = viral_load)) +  
  geom_point(shape = 21) # définir la forme
```



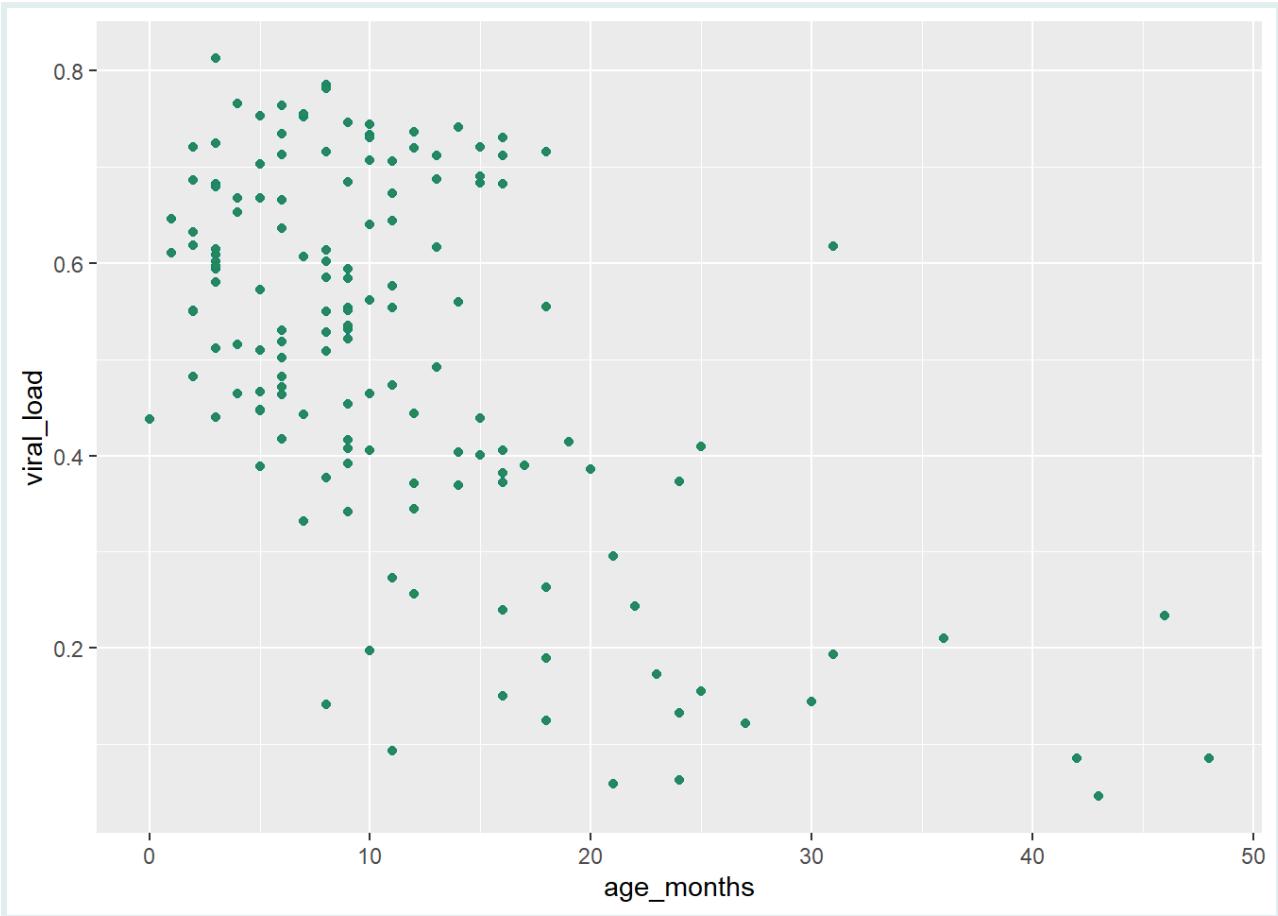
Les formes remplies peuvent avoir des couleurs différentes pour le contour et l'intérieur. Modifier l'esthétique `color` ne changera que le contour de nos points :

```
# Définir la couleur du contour des formes en ajoutant `color = cyan4`
ggplot(data = malidd,
        mapping = aes(x = age_months,
                      y = viral_load)) +
  geom_point(shape = 21,                      # définir la forme
             color = "cyan4")                # définir la couleur du contour
```



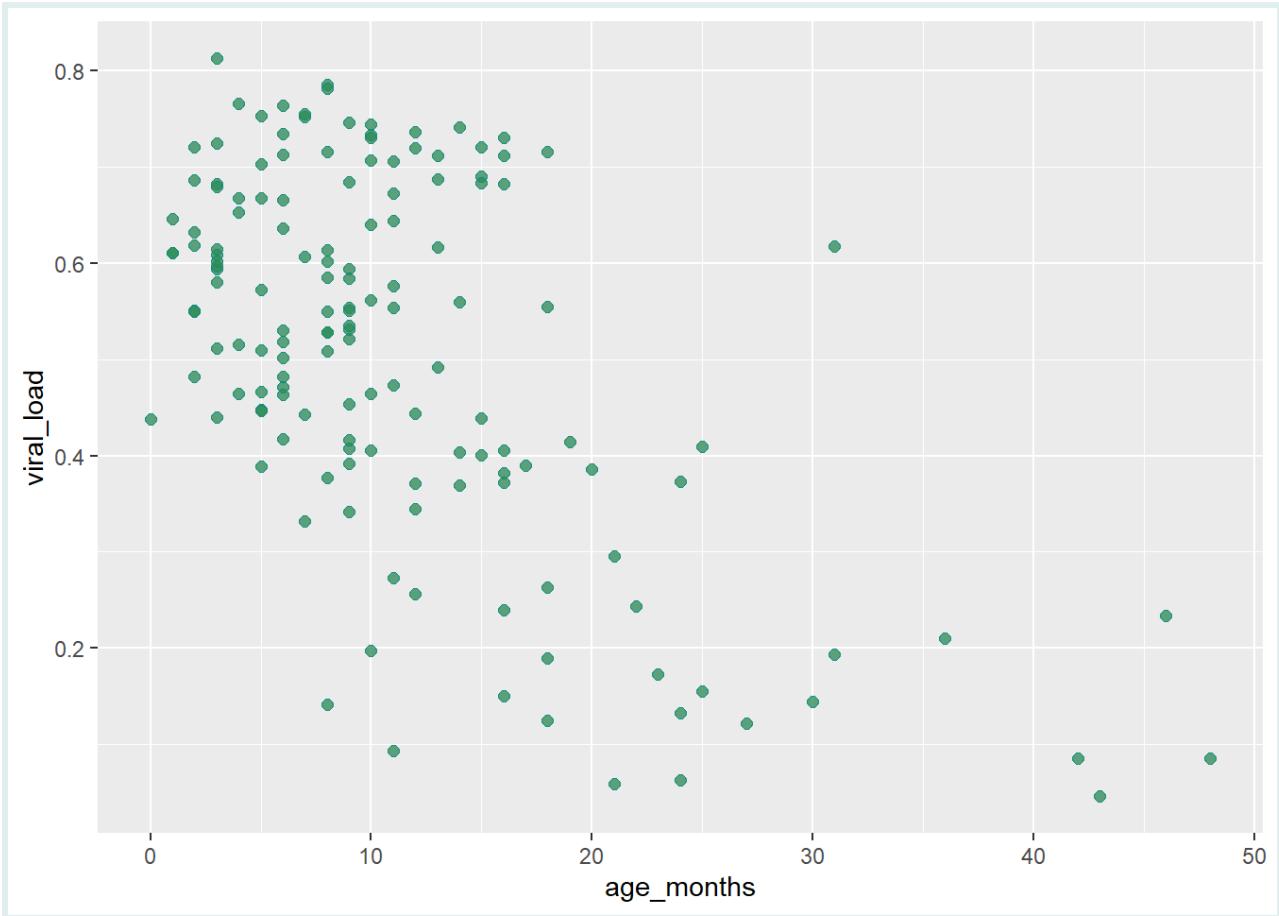
Maintenant, remplissons les points :

```
# Définir la couleur intérieure des formes en ajoutant `fill = "seagreen"`
ggplot(data = malidd,
       mapping = aes(x = age_months,
                      y = viral_load)) +
  geom_point(shape = 21,                         # définir la forme
             color = "cyan4",                   # définir la couleur du contour
             fill = "seagreen")                 # définir la couleur de remplissage
```



Nous pouvons améliorer la lisibilité en augmentant la taille et en réduisant l'opacité avec `size` et `alpha`, comme nous l'avons fait précédemment :

```
ggplot(data = malidd,
       mapping = aes(x = age_months,
                     y = viral_load)) +
  geom_point(shape = 21,                      # définir la forme
             color = "cyan4",                  # définir la couleur du contour
             fill = "seagreen",                # définir la couleur de remplissage
             size = 2,                        # définir la taille (mm)
             alpha = 0.75)                   # définir le niveau d'opacité
```



## Ajouter une ligne de tendance

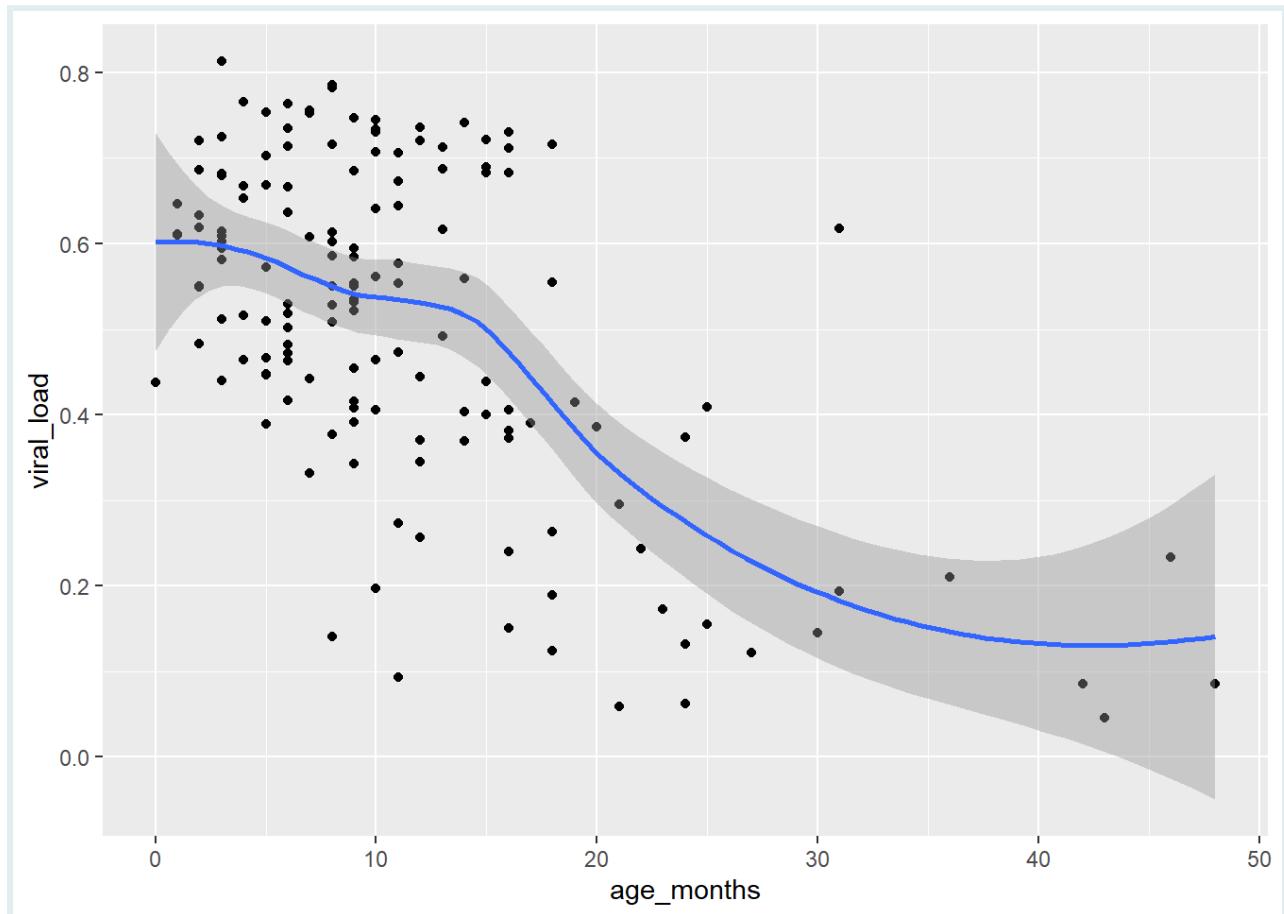
Il peut être difficile de discerner à première vue des relations ou des tendances en se basant uniquement sur les points du graphique. Une solution courante pour améliorer la lisibilité de ces tendances consiste à ajouter une ligne de lissage. Ceci est particulièrement utile lorsque nous cherchons à comprendre les relations de régression entre les variables.

Pour avoir une meilleure idée de la relation entre ces deux variables, nous pouvons ajouter une ligne de tendance (également connue sous le nom de ligne de meilleure adéquation ou ligne de lissage).

Pour ce faire, nous ajoutons la fonction `geom_smooth()` à notre code :

```
ggplot(data = malidd,
       mapping = aes(x = age_months,
                      y = viral_load)) +
  geom_point() +
  geom_smooth()
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~  
## x'
```



La ligne de lissage est ajoutée à notre graphique comme une autre couche géométrique et vient après les points.

La fonction de lissage par défaut utilisée dans ce graphique de dispersion est “loess” qui signifie **locally weighted scatter plot smoothing**. Le lissage loess est une technique couramment employée par de nombreux logiciels de statistiques. Dans {ggplot2}, l’usage de loess est généralement recommandé lorsque vous avez moins de 1000 points, car son calcul prend beaucoup de temps.

De nombreuses fonctions de lissage peuvent également être utilisées dans `geom_smooth()`.

Cette fois-ci, utilisons plutôt une méthode de régression linéaire. Nous allons opter pour un modèle linéaire généralisé. Pour ce faire, nous définissons l’argument `method` à l’intérieur de `geom_smooth()` :

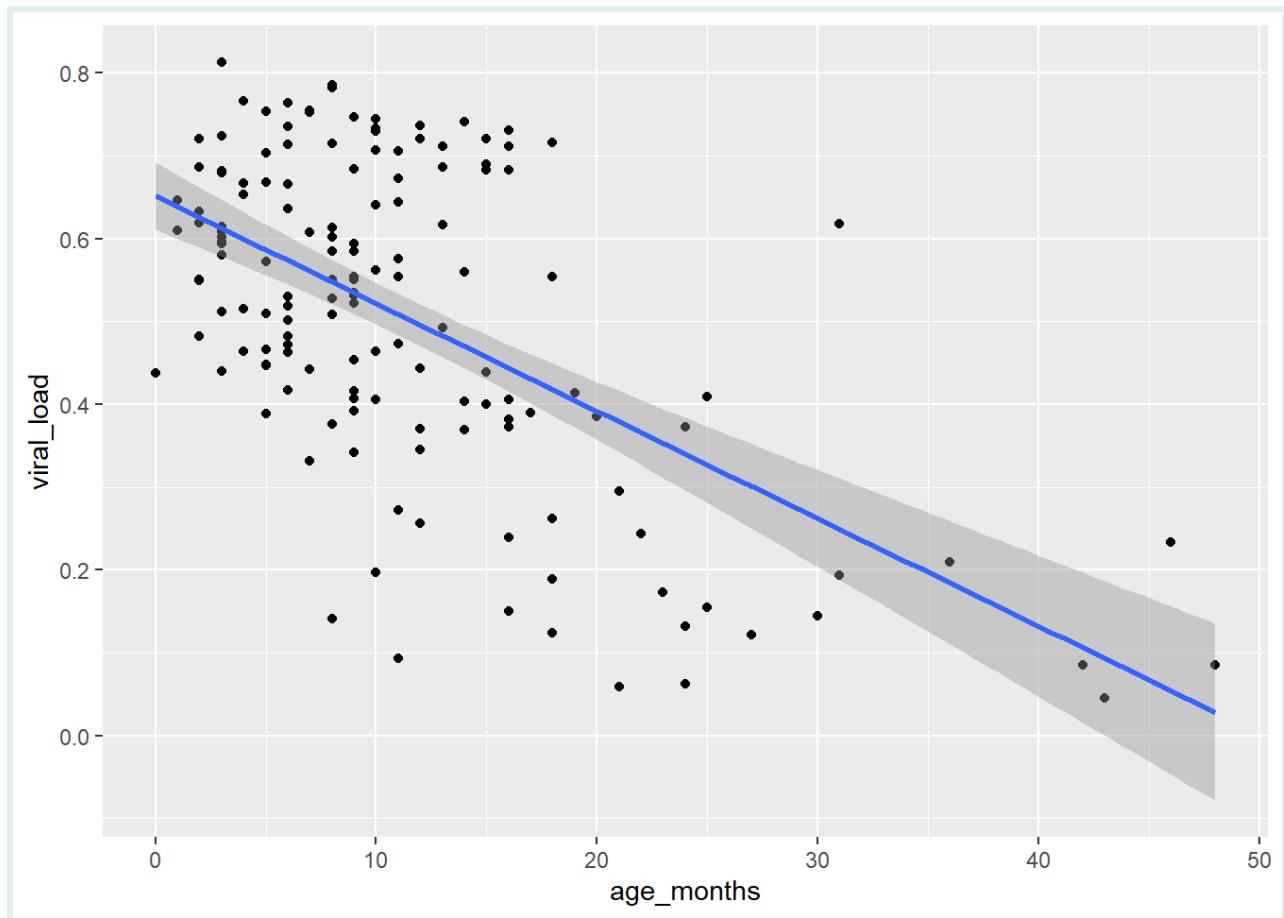
```
# Changer pour une fonction de lissage par régression linéaire avec `method =  
# "glm"`,  
ggplot(data = malidd,
```

```

mapping = aes(x = age_months,
              y = viral_load)) + geom_point() +
geom_smooth(method = "glm")

## `geom_smooth()` using formula = 'y ~ x'

```



Par défaut, des intervalles de confiance à 95% sont affichés autour de ces lignes de lissage.

Pour supprimer ces intervalles, vous pouvez ajouter l'argument `se = FALSE` à l'intérieur de `geom_smooth()` :

```

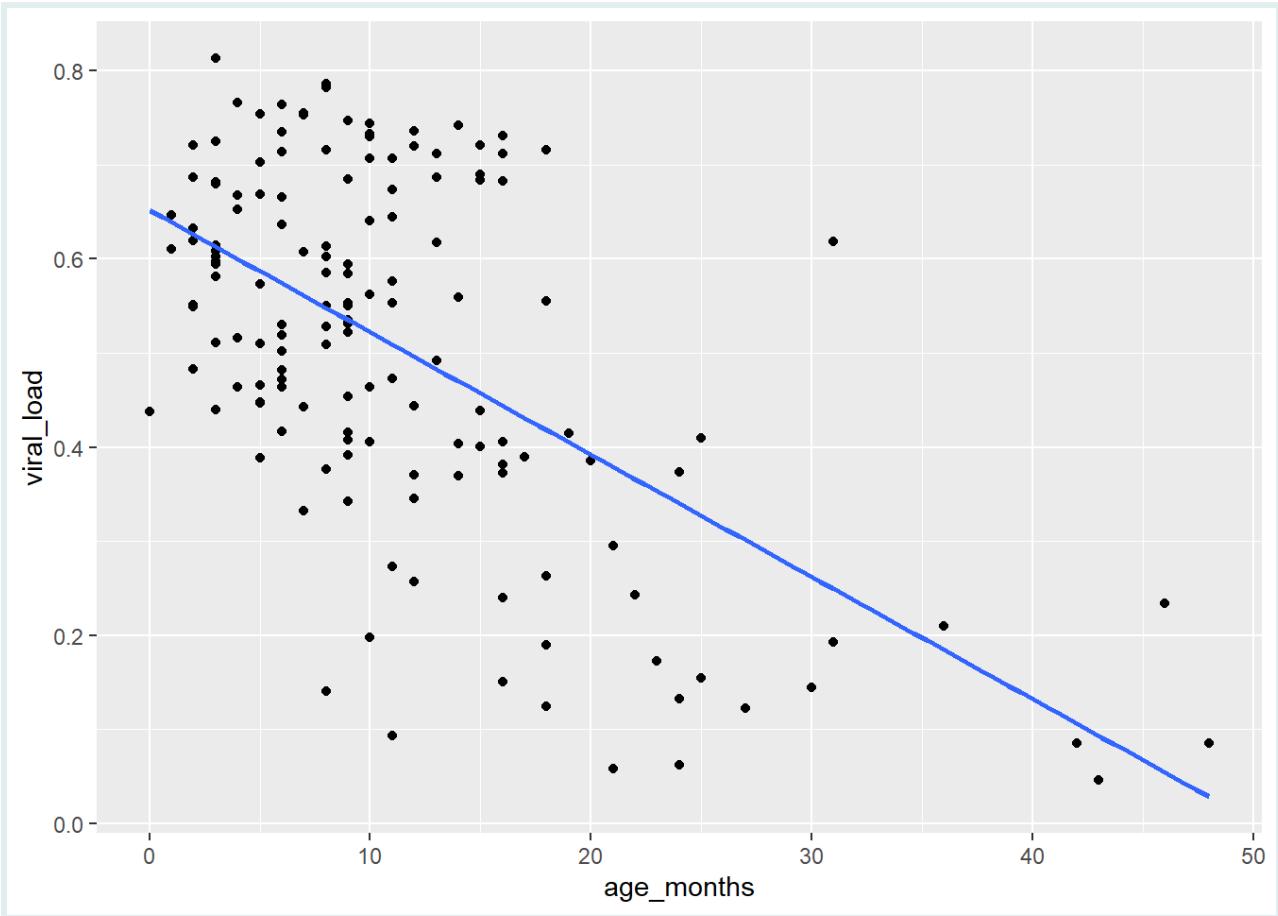
# Supprimer les intervalles de confiance en ajoutant `se = FALSE`
ggplot(data = malidd,
        mapping = aes(x = age_months,
                      y = viral_load)) +
  geom_point() +
  geom_smooth(method = "glm",
              se = FALSE)

```

```

## `geom_smooth()` using formula = 'y ~ x'

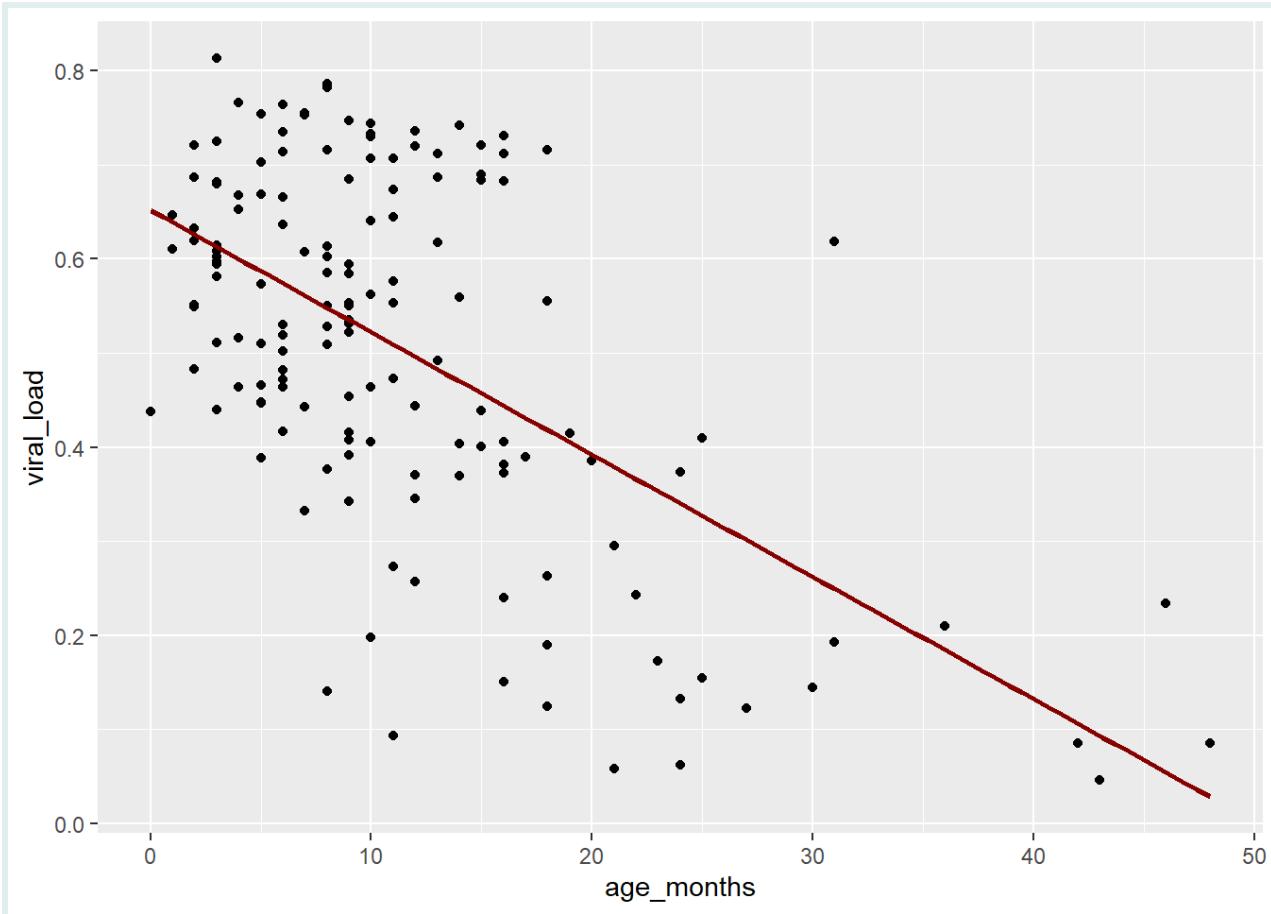
```



Ajoutons aussi l'argument `color` à l'intérieur de `geom_smooth()` pour changer la couleur de la ligne :

```
# Changer la couleur de la ligne de tendance en ajoutant `color = "darkred"`
ggplot(data = malidd,
       mapping = aes(x = age_months,
                      y = viral_load)) +
  geom_point() +
  geom_smooth(method = "glm",
              se = FALSE,
              color = "darkred")

## `geom_smooth()` using formula = 'y ~ x'
```

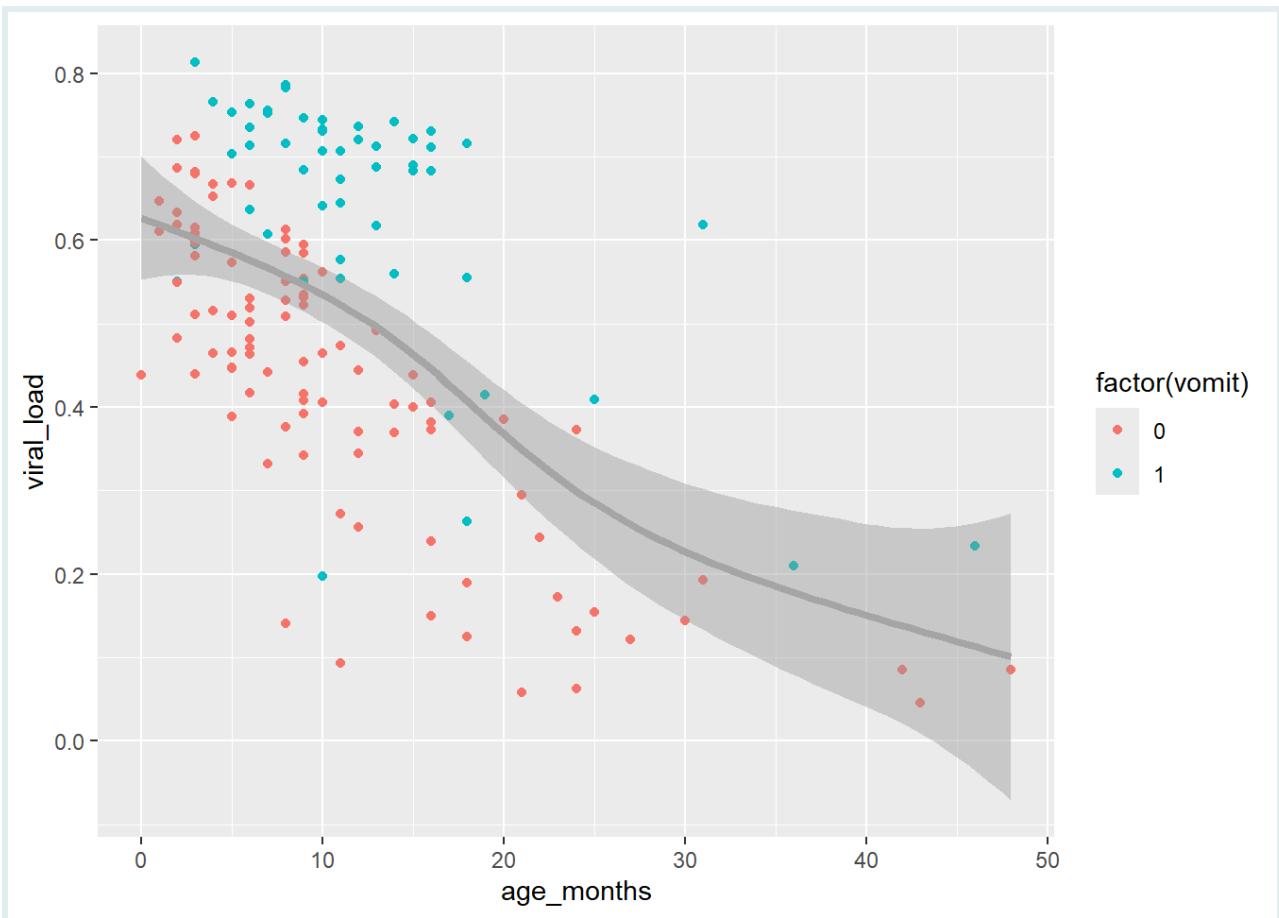


Cette régression linéaire confirme ce que nous avions initialement observé dans le premier nuage de points : Une *relation négative* existe entre `age_months` et `viral_load` : lorsque l'âge augmente, la charge virale tend à diminuer.

Introduisons maintenant une troisième variable du dataset `malidd` appelée `vomit`. Cette variable binaire enregistre si le patient a vomi ou non. Nous allons incorporer la variable `vomit` au graphique en la mappant à l'esthétique de la couleur. Nous allons changer à nouveau la méthode de lissage pour le modèle additif généralisé ("gam") et nous apporterons quelques modifications esthétiques à la ligne dans la couche `geom_smooth()`.

```
ggplot(data = malidd,
       mapping = aes(x = age_months,
                      y = viral_load)) +
  geom_point(mapping = aes(color = factor(vomit))) +
  geom_smooth(method = "gam",
              size = 1.5,
              color = "darkgray")
```

```
## `geom_smooth()` using formula = 'y ~ s(x, bs = "cs")'
```



Notez la distribution des points bleus (représentant les enfants qui ont vomi) par rapport aux points rouges (représentant les enfants qui n'ont pas vomi). Les points bleus se situent principalement au-dessus de la ligne de tendance, ce qui indique que les charges virales plus élevées étaient non seulement associées à des enfants plus jeunes, mais aussi à des enfants plus susceptibles de présenter des symptômes de vomissements.



- Créez un graphique de dispersion avec les variables `age_months` et `viral_load`. Définissez la couleur des points à “steelblue”, la taille à 2,5 mm et l’opacité à 80%. Puis ajoutez une ligne de tendance avec la méthode de lissage “lm” (modèle linéaire). Pour faire ressortir la ligne de tendance, définissez sa couleur à “indianred3”.
- Recréez le graphique de la question précédente, mais cette fois adaptez le code pour changer la forme des points à la forme numéro 23, et ajoutez la variable de température corporelle (`temp`) en la mappant à la couleur de remplissage des points.

---

## En résumé

Les graphiques de dispersion permettent de visualiser la relation entre deux variables quantitatives.

Avec un dataset de taille moyenne à grande, vous pouvez expérimenter avec différentes variations des éléments d'un graphique de dispersion que nous avons vus, comme l'ajout de lignes de tendance, la modification de la couleur, de la taille, de la forme, du remplissage ou de l'opacité des points. Cette personnalisation est souvent un aspect ludique de la visualisation de données, car elle vous permet de découvrir différentes relations qui se dévoilent au fur et à mesure que vous ajustez vos graphiques.

---

## Contributeurs

Ont contribué à ce cours :



### JOY VAZ

R Developer and Instructor, the GRAPH Network  
Loves doing science and teaching science



### IMANE BENSOUDA KORACHI

R Developer and Instructor, the GRAPH Network



### ADMIN TEAM

GRAPH Courses Administration Team

The GRAPH Courses team is building epidemiological training courses to enhance disease surveillance and data science for public health across the globe

---

## Références

Le contenu de ce cours a été adapté en partie des sources suivantes :

- Ismay, Chester, and Albert Y. Kim. 2022. *A ModernDive into R and the Tidyverse*. <https://moderndive.com/>.

- Kabacoff, Rob. 2020. *Data Visualization with R*. [https://rkabacoff.github.io/  
datavis/](https://rkabacoff.github.io/datavis/).
- Giroux-Bougard, Xavier, Maxwell Farrell, Amanda Winegardner, Étienne Low-Decarie and Monica Granados. 2020. *Workshop 3: Introduction to Data Visualisation with {ggplot2}*. <http://r.qcbs.ca/workshop03/book-en/>.

This work is licensed under the [Creative Commons Attribution Share Alike](#) license.



# Lignes, échelles et étiquettes

March 2024

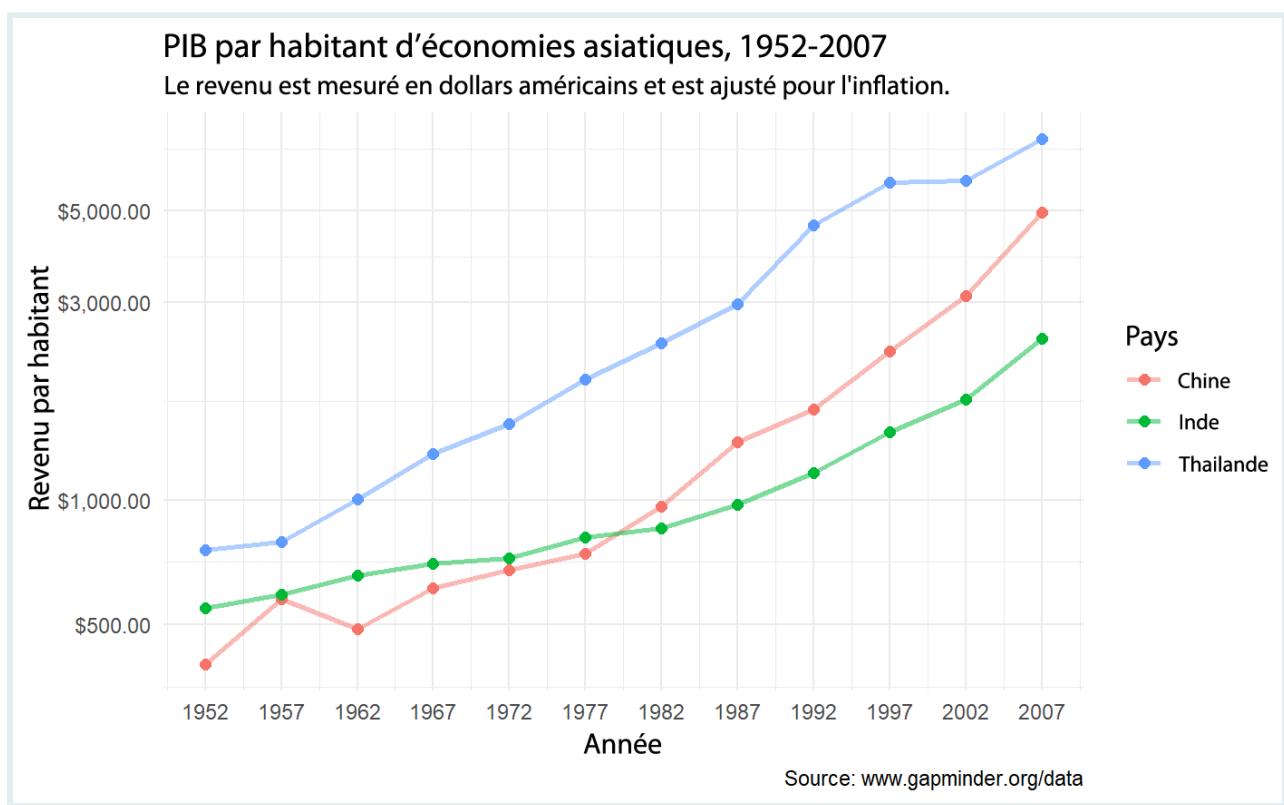


Objectifs d'Apprentissage . . . . .
Introduction . . . . .
Packages . . . . .
Le dataframe <code>gapminder</code> . . . . .
Graphiques linéaires avec <code>geom_line()</code> . . . . .
Les esthétiques fixes dans <code>geom_line()</code> . . . . .
Combiner les éléments géométriques . . . . .
Mapper les données sur plusieurs lignes . . . . .
Modifier les échelles continues x et y . . . . .
Modifier les graduations des axes . . . . .
Définir une échelle logarithmique . . . . .
Étiquetage avec <code>labs()</code> . . . . .
Preview : Les thèmes . . . . .
En résumé . . . . .
Contributeurs . . . . .
Références . . . . .

---

## Objectifs d'Apprentissage

1. Créer des **graphiques linéaires** pour visualiser les relations entre deux variables numériques avec `geom_line()`.
2. Ajouter des **points** à un graphique linéaire avec `geom_point()`.
3. Utiliser des esthétiques comme `size`, `color`, et `linetype` pour modifier les graphiques linéaires.
4. Manipuler les échelles (`scale`) des axes de données continues avec `scale_*_continuous()` et `scale_*_log10()`.
5. Ajouter des étiquettes (**labels**) à un graphique tels que `title`, `subtitle`, ou `caption` avec la fonction `labs()`.



## Introduction

Les graphiques linéaires sont utilisés pour montrer les **relations** entre deux **variables numériques**, tout comme les nuages de points. Ils sont particulièrement utiles lorsque la variable sur l'axe des x, également appelée variable *explicative*, est de nature **séquentielle**. En d'autres termes, il y a un ordre inhérent à la variable.

Les exemples les plus courants de graphiques linéaires incluent une composante **temporelle sur l'axe des x**, tels que les heures, les jours, les semaines ou les années. Étant donné que le temps est une séquence, nous relierons les observations

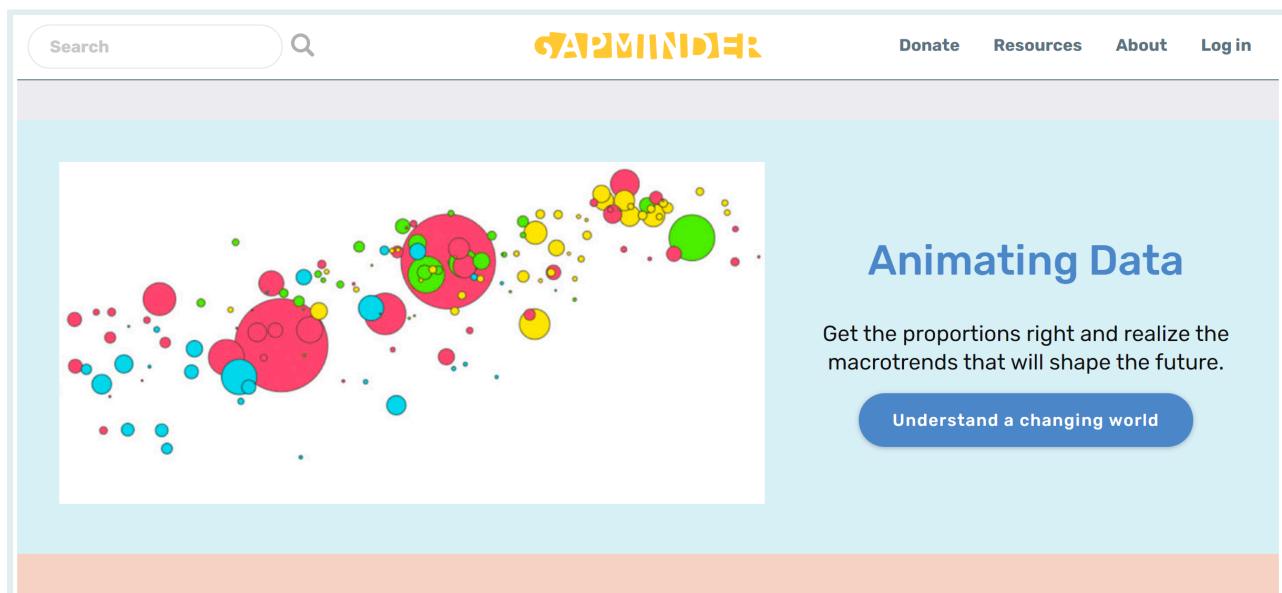
consécutives de la variable sur l'axe des y avec une ligne. Ces graphiques linéaires, qui intègrent une notion de temps sur l'axe des x, sont communément appelés des **graphiques de série temporelle**.

## Packages

```
# Charger les packages  
pacman::p_load(tidyverse,  
                 gapminder,  
                 here)
```

## Le dataframe gapminder

En février 2006, un médecin suédois et défenseur des données nommé Hans Rosling a donné une célèbre conférence TED intitulée “[Les meilleures statistiques jamais vues](#)” où il a présenté des données économiques, sanitaires et de développement mondial compilées par la Fondation Gapminder.



Nous pouvons accéder à un subset de ces données avec le package R **{gapminder}**, que nous venons de charger.

```
# Charger le dataframe gapminder à partir du package gapminder  
data(gapminder, package="gapminder")
```

```
# Afficher le dataframe gapminder
```

Chaque ligne de ce tableau correspond à une combinaison pays-année. Pour chaque ligne, nous avons 6 colonnes :

1. `country` : Nom du pays
2. `continent` : Région géographique du monde
3. `year` : Année calendaire
4. `lifeExp` : L'espérance de vie à la naissance en années
5. `pop` : Population totale
6. `gdpPercap` : Produit intérieur brut par personne (en dollar américain ajusté en fonction de l'inflation)

La fonction `str()` peut nous en apprendre plus sur ces variables.

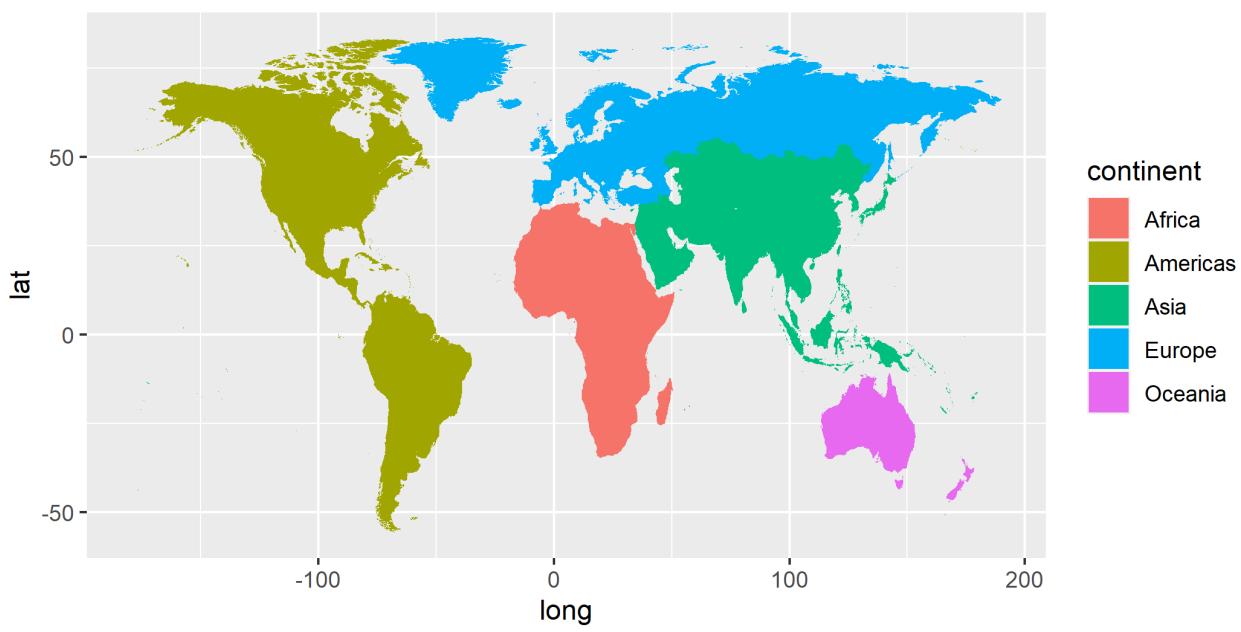
```
# Structure des données
str(gapminder)
```

```
## # tibble [1,704 x 6] (S3: tbl_df/tbl/data.frame)
## $ country : Factor w/ 142 levels "Afghanistan",...: 1 1 1 1 1 1 1 1 1 1 ...
## ...
## $ continent: Factor w/ 5 levels "Africa", "Americas", ...: 3 3 3 3 3 3 3 3 3 3 ...
## $ year     : int [1:1704] 1952 1957 1962 1967 1972 1977 1982 1987 1992 1997 ...
## $ lifeExp   : num [1:1704] 28.8 30.3 32 34 36.1 ...
## $ pop       : int [1:1704] 8425333 9240934 10267083 11537966 13079460 14880372 12881816 13867957 16317921 22227415 ...
## $ gdpPercap : num [1:1704] 779 821 853 836 740 ...
```

Cette version du dataset `gapminder` contient des informations sur **142 pays** répartis en **5 continents**.

## Gapminder world regions

Five regions in the `continent` variable of `gapminder`



```
# Résumé des données
summary(gapminder)
```

```
##           country      continent
## Afghanistan: 12     Africa :624
## Albania     : 12    Americas:300
## Algeria     : 12     Asia   :396
## Angola      : 12    Europe :360
## Argentina   : 12   Oceania : 24
## Australia   : 12
## (Other)      :1632
##           year      lifeExp
## Min.   :1952   Min.   :23.60
## 1st Qu.:1966  1st Qu.:48.20
## Median :1980  Median :60.71
## Mean   :1980  Mean   :59.47
## 3rd Qu.:1993  3rd Qu.:70.85
## Max.   :2007   Max.   :82.60
##
##           pop      gdpPercap
## Min.   :6.0001e+04  Min.   : 241.2
## 1st Qu.:2.794e+06  1st Qu.: 1202.1
## Median :7.024e+06  Median : 3531.8
## Mean   :2.960e+07  Mean   : 7215.3
## 3rd Qu.:1.959e+07  3rd Qu.: 9325.5
## Max.   :1.319e+09  Max.   :113523.1
##
```

Les données sont enregistrées tous les 5 ans de 1952 à 2007 (soit un total de 12 années).

Supposons que nous voulions visualiser la relation entre le temps (`year`) et l'espérance de vie (`lifeExp`).

Pour l'instant, concentrons-nous uniquement sur un pays - les États-Unis. D'abord, nous devons créer un nouveau dataframe avec uniquement les données de ce pays.

```
# Sélectionner les cas US
gap_US <- dplyr::filter(gapminder,
                        country == "United States")

gap_US
```

**REMINDER**



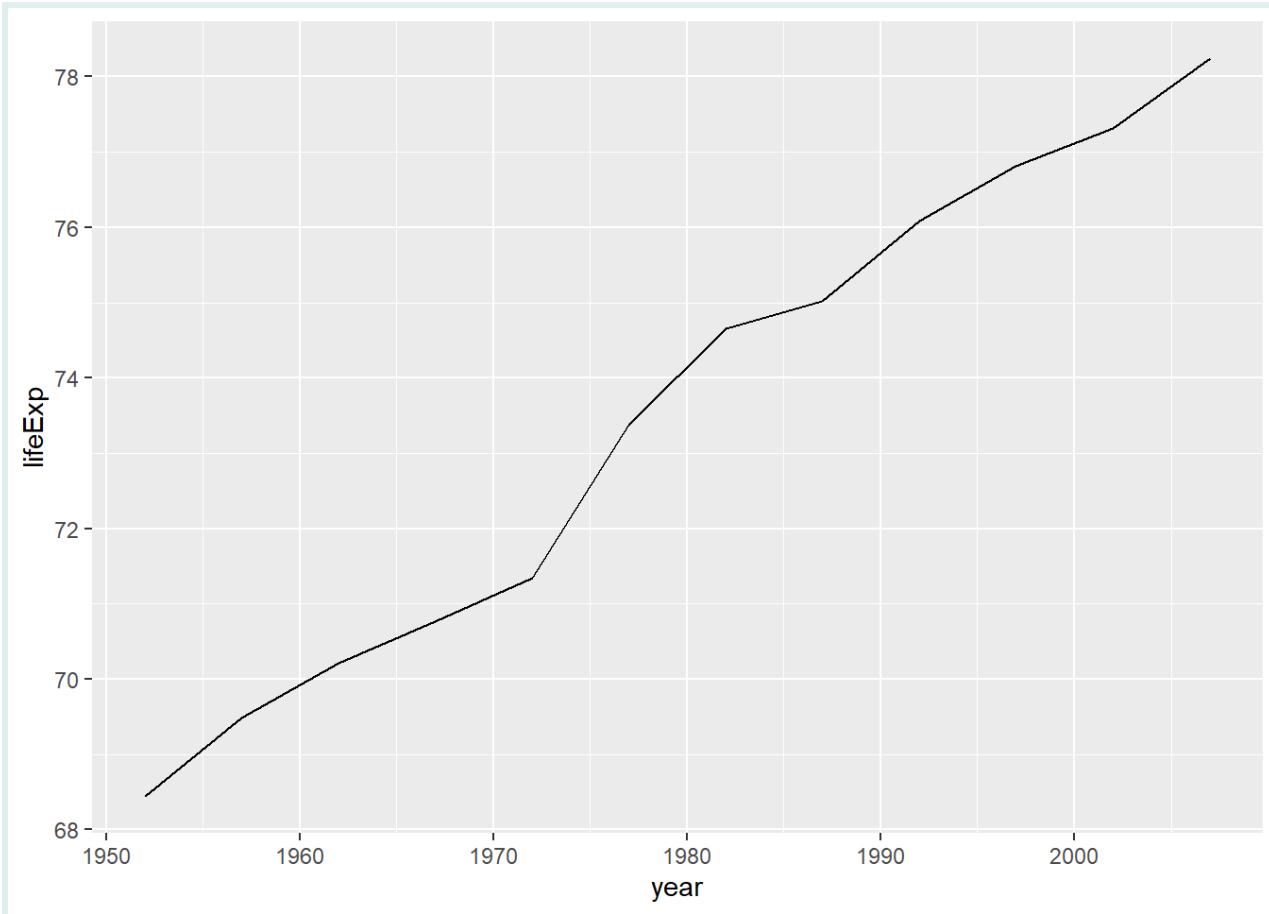
Le code ci-dessus est couvert dans notre cours sur la manipulation de données en utilisant le package `{dplyr}`. La manipulation de données est le processus de transformation et de nettoyage des données dans le but de les rendre plus appropriées à des fins d'analyse. Par exemple, ce code utilise la fonction `filter()` pour créer un nouveau dataframe (`gap_US`) en incluant uniquement les lignes du dataframe `gapminder` qui ont "United States" dans la colonne `country`.

## Graphiques linéaires avec `geom_line()`

Nous allons utiliser le dataframe `gap_US` avec `ggplot()` pour tracer **le temps** en années sur l'axe des abscisses x et **l'espérance de vie** sur l'axe des ordonnées y.

Nous pouvons visualiser les données de séries temporelles en utilisant `geom_line()` pour créer un graphique linéaire, au lieu d'utiliser `geom_point()` comme nous l'avons fait précédemment pour créer un nuage de points :

```
# Graphique linéaire simple
ggplot(data = gap_US,
       mapping = aes(x = year,
                     y = lifeExp)) +
  geom_line()
```



Tout comme avec le code `ggplot()` qui a créé le nuage de points de l'âge et de la charge virale avec `geom_point()`, décomposons ce code couche par couche en utilisant la grammaire des graphiques :

Dans l'appel de la fonction `ggplot()`, nous précisons deux des composants de la grammaire des graphiques comme arguments :

1. Le dataframe `gap_US` comme couche de données en réglant `data = gap_US`.
2. Le mapping esthétique `aes` en réglant `mapping = aes(x = year, y = lifeExp)`. Plus précisément, la variable `year` est associée à l'esthétique de position `x`, tandis que la variable `lifeExp` est associée à l'esthétique de position `y`.

Après avoir précisé à R quelles données et quelles correspondances esthétiques nous voulions tracer, nous allons ajouté le troisième composant essentiel, l'objet géométrique en utilisant l'opérateur `+`. Dans ce cas, l'objet géométrique a été réglé sur des lignes en utilisant `geom_line()`.





**PRACTICE**  
Créez un graphique de série temporelle du PIB par habitant (`gdpPercap`) à partir du dataframé `gap_US` en utilisant `geom_line()` pour créer un graphique linéaire.

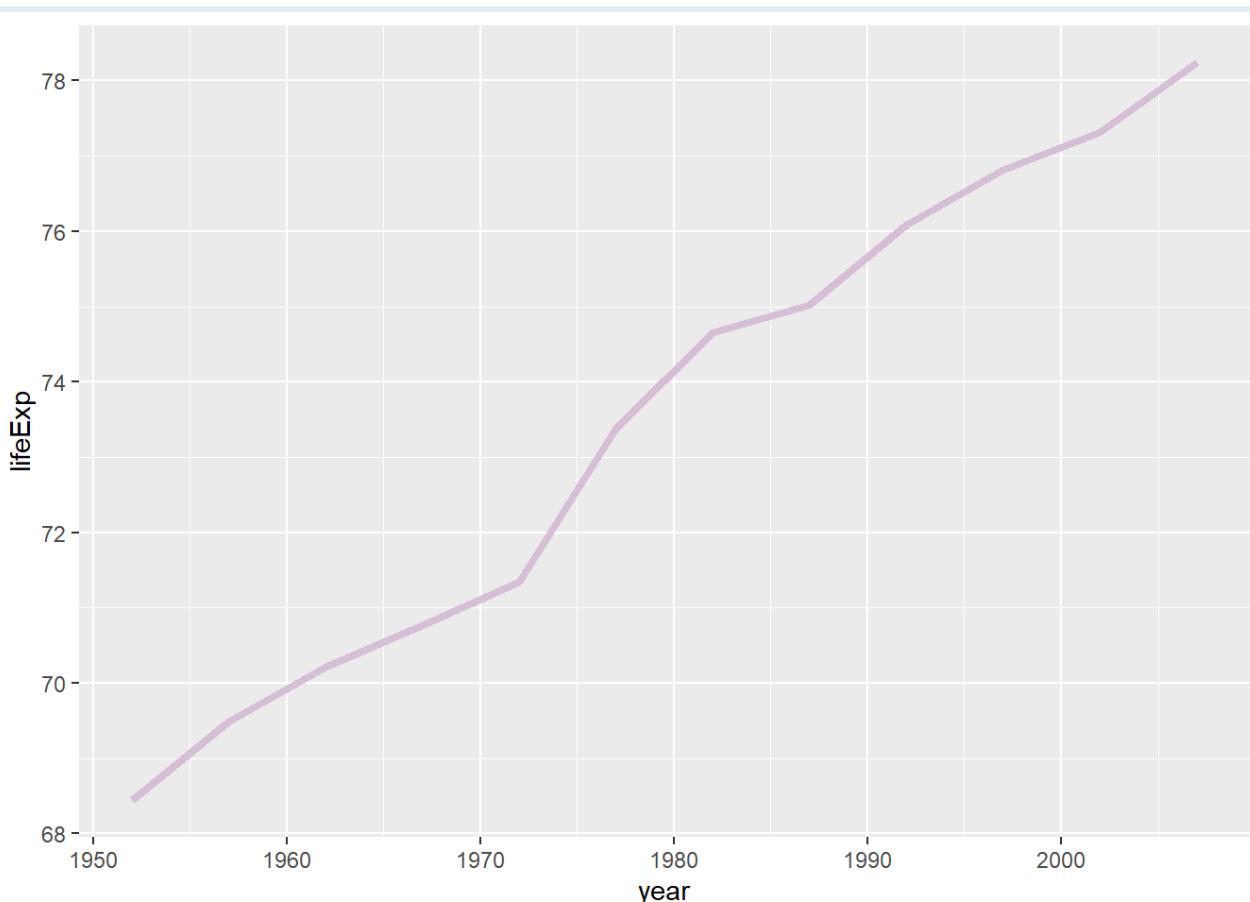
## Les esthétiques fixes dans `geom_line()`

La couleur, l'épaisseur des lignes et le type de ligne du graphique linéaire peuvent être personnalisés en utilisant les arguments `color`, `size` et `linetype`, respectivement.

Nous avons changé la couleur et la taille des géométries dans le cours précédent.

Nous allons les réutiliser comme esthétiques fixes :

```
# Améliorer le graphique linéaire en ajoutant la couleur et la taille comme
# esthétiques fixes
ggplot(data = gap_US,
        mapping = aes(x = year,
                      y = lifeExp)) +
  geom_line(color = "thistle",
            size = 1.5)
```

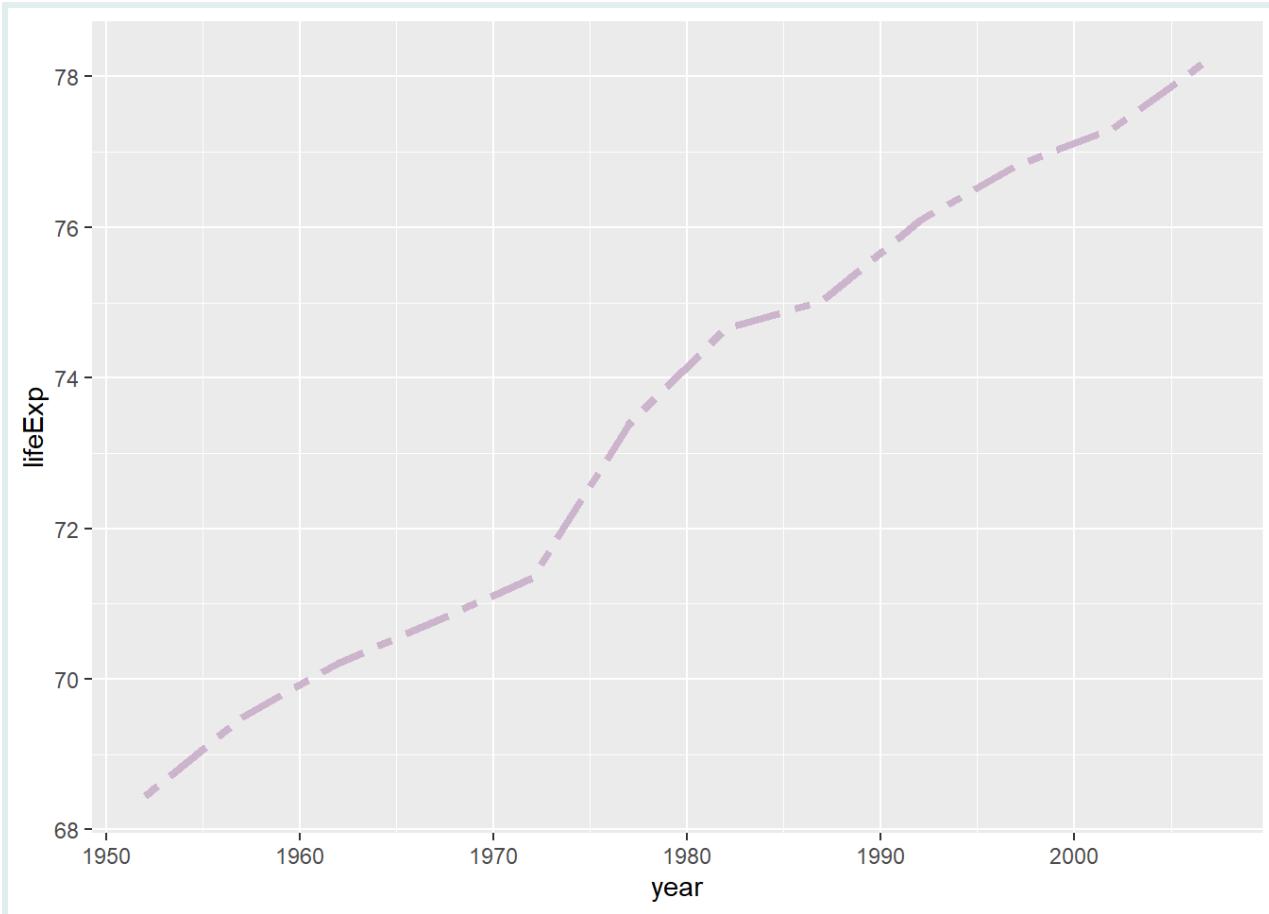


Nous allons maintenant introduire une nouvelle esthétique fixe qui est spécifique aux graphiques linéaires : `linetype` (ou `lty` en abrégé).

—	<code>lty = 0 or 'blank'</code>
—	<code>lty = 1 or 'solid'</code>
- - - - -	<code>lty = 2 or 'dashed'</code>
- - - - -	<code>lty = 3 or 'dotted'</code>
- - - - -	<code>lty = 4 or 'dotdash'</code>
- - - - -	<code>lty = 5 or 'longdash'</code>
- - - - -	<code>lty = 6 or 'twodash'</code>

Le type de ligne dans un graphique peut être spécifié en utilisant un nom ou un entier. Les types de ligne valides peuvent être définis à l'aide de chaînes de caractères compréhensibles : "blank", "solid", "dashed", "dotted", "dotdash", "longdash", et "twodash" sont tous compris par `linetype` ou `lty`.

```
# Améliorer le graphique linéaire en ajoutant la couleur, l'épaisseur et le
# type de ligne comme esthétiques fixes
ggplot(data = gap_US,
        mapping = aes(x = year,
                      y = lifeExp)) +
  geom_line(color = "thistle3",
            size = 1.5,
            linetype = "twodash")
```



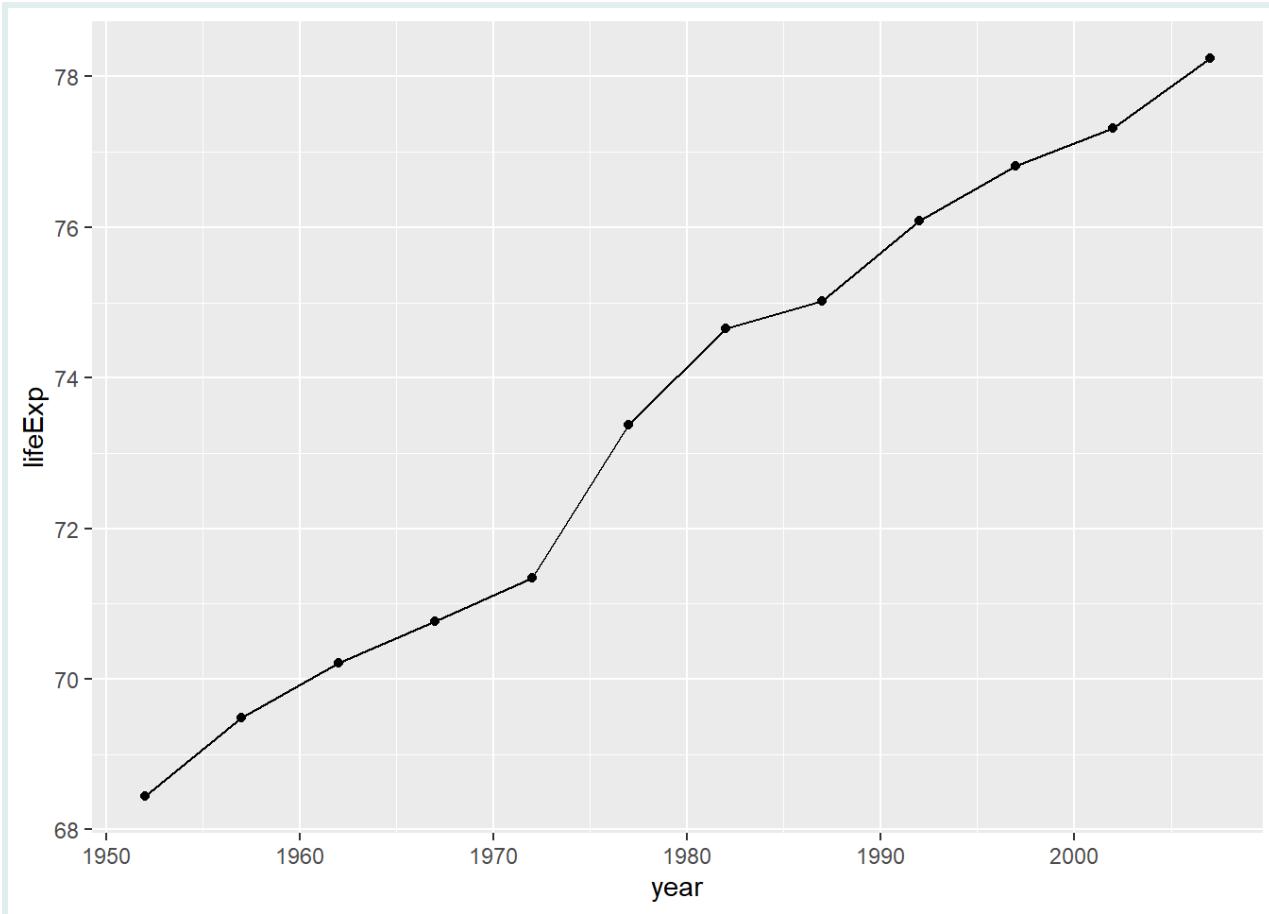
Dans les graphiques linéaires, il peut parfois être difficile de déterminer l'emplacement exact des points de données. Dans le graphique suivant, nous allons inclure des points pour une meilleure visualisation.

## Combiner les éléments géométriques

Tant que les géométries sont compatibles, nous pouvons les superposer les unes sur les autres pour personnaliser davantage un graphique.

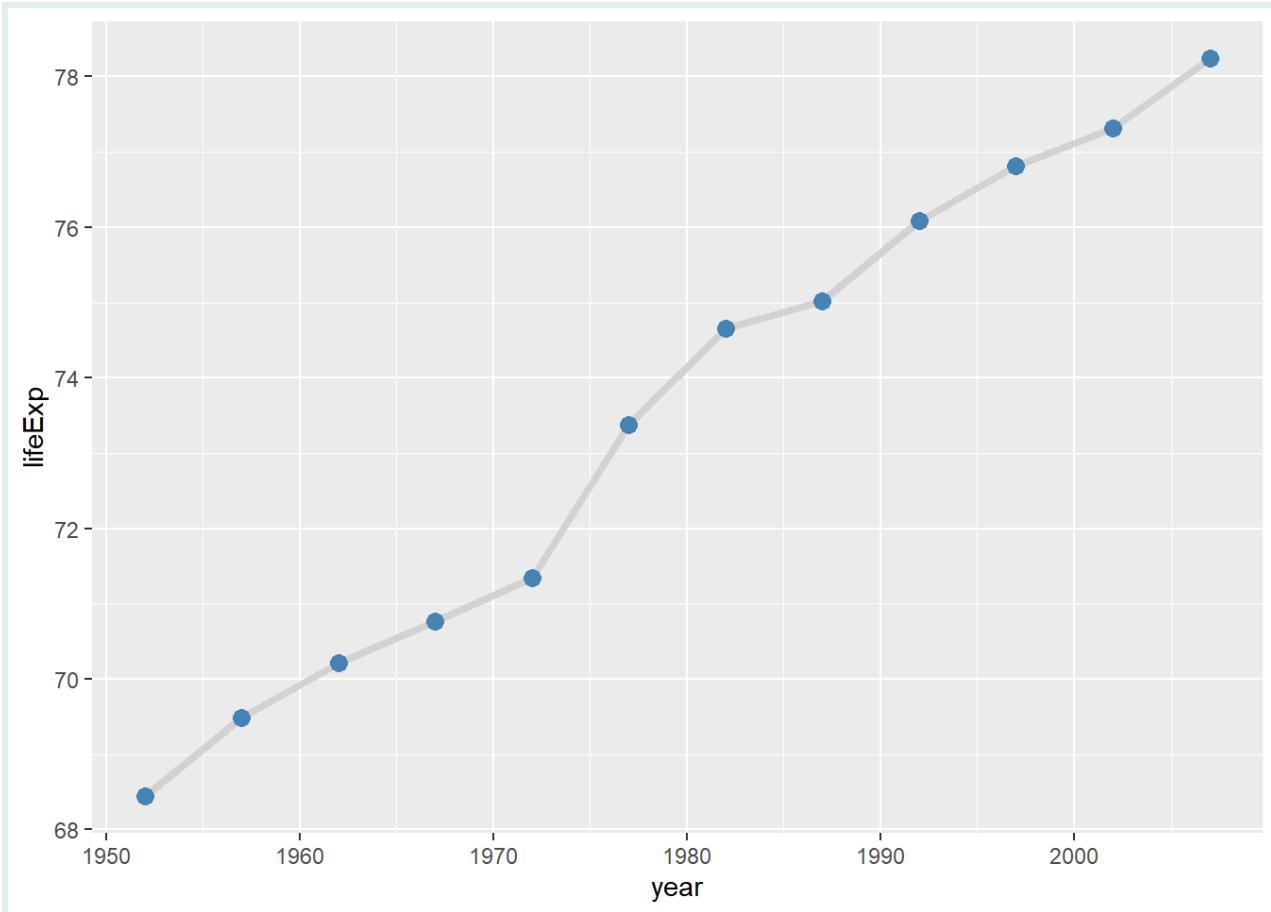
Par exemple, nous pouvons ajouter des points à notre graphique linéaire en utilisant l'opérateur `+` pour ajouter une seconde couche de `geom` avec `geom_point()` :

```
# Graphique linéaire simple avec des points
ggplot(data = gap_US,
       mapping = aes(x = year,
                     y = lifeExp)) +
  geom_line() +
  geom_point()
```



Nous pouvons améliorer l'apparence du graphique en personnalisant la taille et la couleur de nos géométries.

```
# Graphique linéaire avec des points et des esthétiques fixes
ggplot(data = gap_US,
        mapping = aes(x = year,
                      y = lifeExp)) +
  geom_line(size = 1.5,
            color = "lightgrey") +
  geom_point(size = 3,
             color = "steelblue")
```



En vous basant sur le code ci-dessus, visualisez la relation entre le temps et le **PIB par habitant** (`gdpPercap`) à partir du dataframe `gap_US`.

Utilisez à la fois des points et des lignes pour représenter les données.

Changez le type de ligne et la couleur des points par n'importe quelle valeur valide de votre choix.

## Mapper les données sur plusieurs lignes

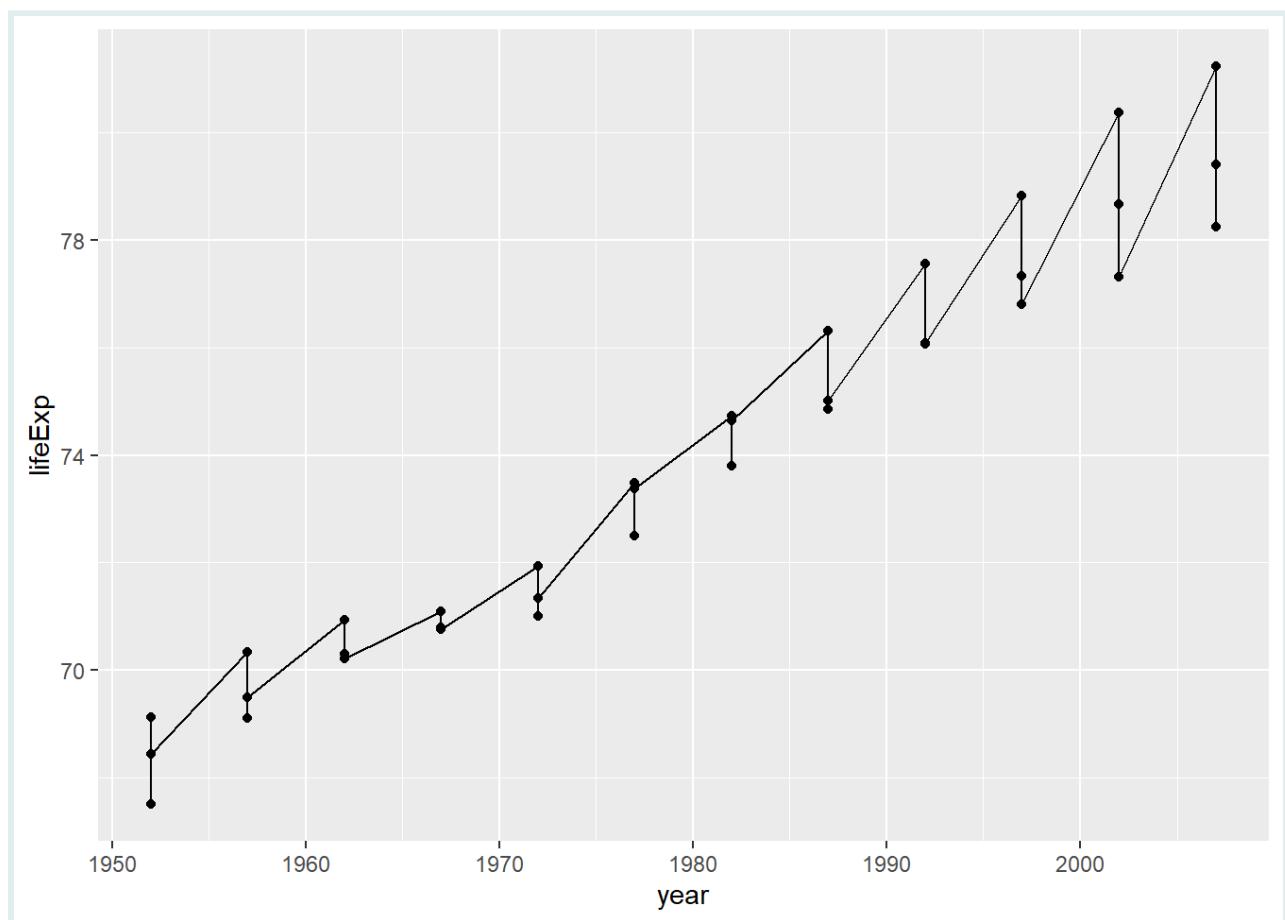
Dans la section précédente, nous n'avons examiné les données que d'un seul pays, mais que faire si nous voulons tracer les données de plusieurs pays et les comparer ?

D'abord, ajoutons deux autres pays à notre subset :

```
# Créer un subset pour visualiser plusieurs catégories
gap_mini <- filter(gapminder,
                     country %in% c("United States",
                     "Australia",
                     "Germany"))
gap_mini
```

Lorsque nous remplaçons `gap_US` par `gap_mini` dans notre code, les lignes ne sont pas automatiquement séparées par pays :

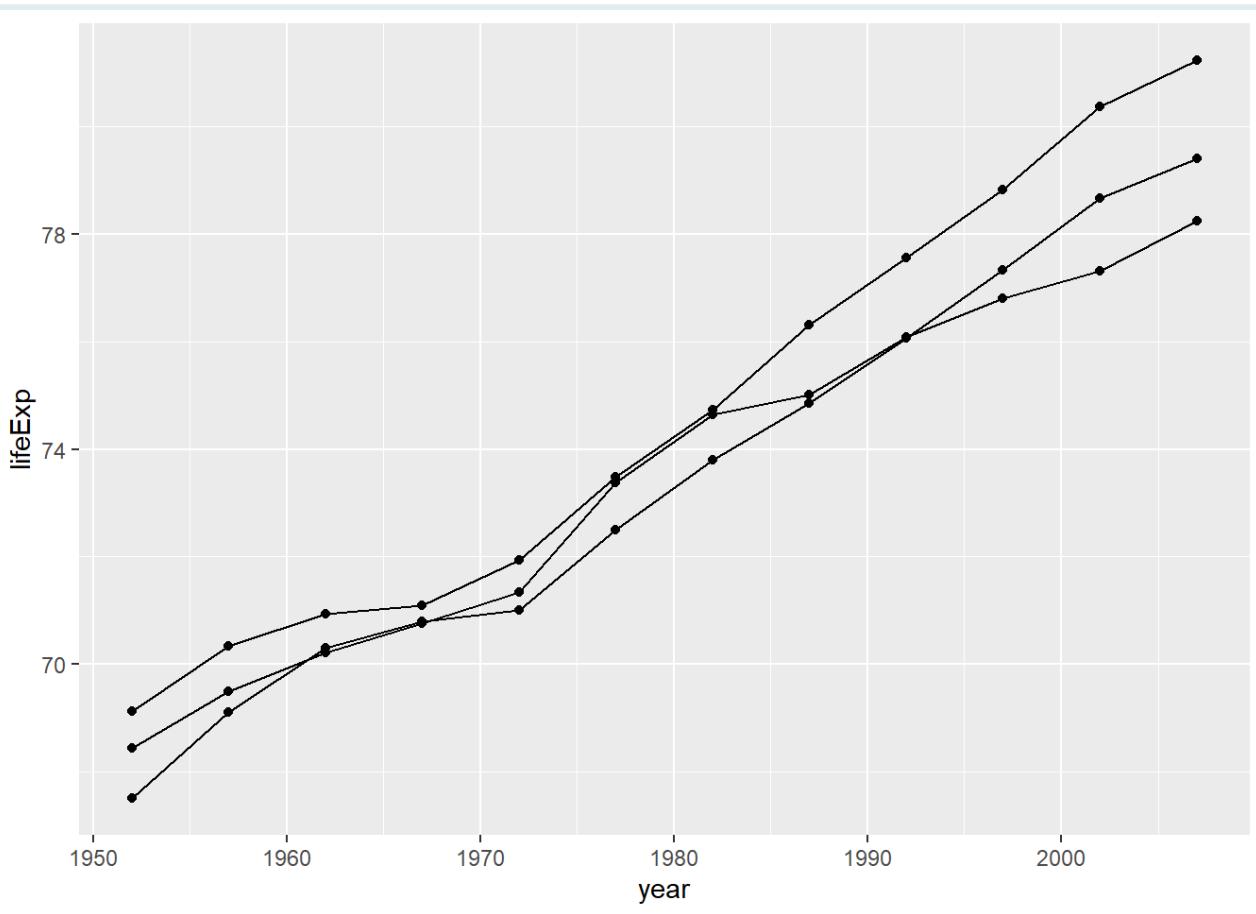
```
# Graphique en ligne sans esthétique de groupe
ggplot(data = gap_mini,
       mapping = aes(y = lifeExp,
                     x = year)) +
  geom_line() +
  geom_point()
```



Ce graphique n'est pas très utile pour faire des comparaisons entre les groupes.

Pour indiquer à `ggplot()` de mapper les données de chaque pays séparément, nous pouvons utiliser l'argument `group` comme mapping esthétique :

```
# Graphique linéaire avec regroupement par une variable catégorielle
ggplot(data = gap_mini,
       mapping = aes(y = lifeExp,
                     x = year,
                     group = country)) +
  geom_line() +
  geom_point()
```



Maintenant que les données sont groupées par pays, nous avons 3 lignes séparées - une pour chaque modalité de la variable `country`.

Nous pouvons également appliquer des esthétiques fixes aux couches géométriques.

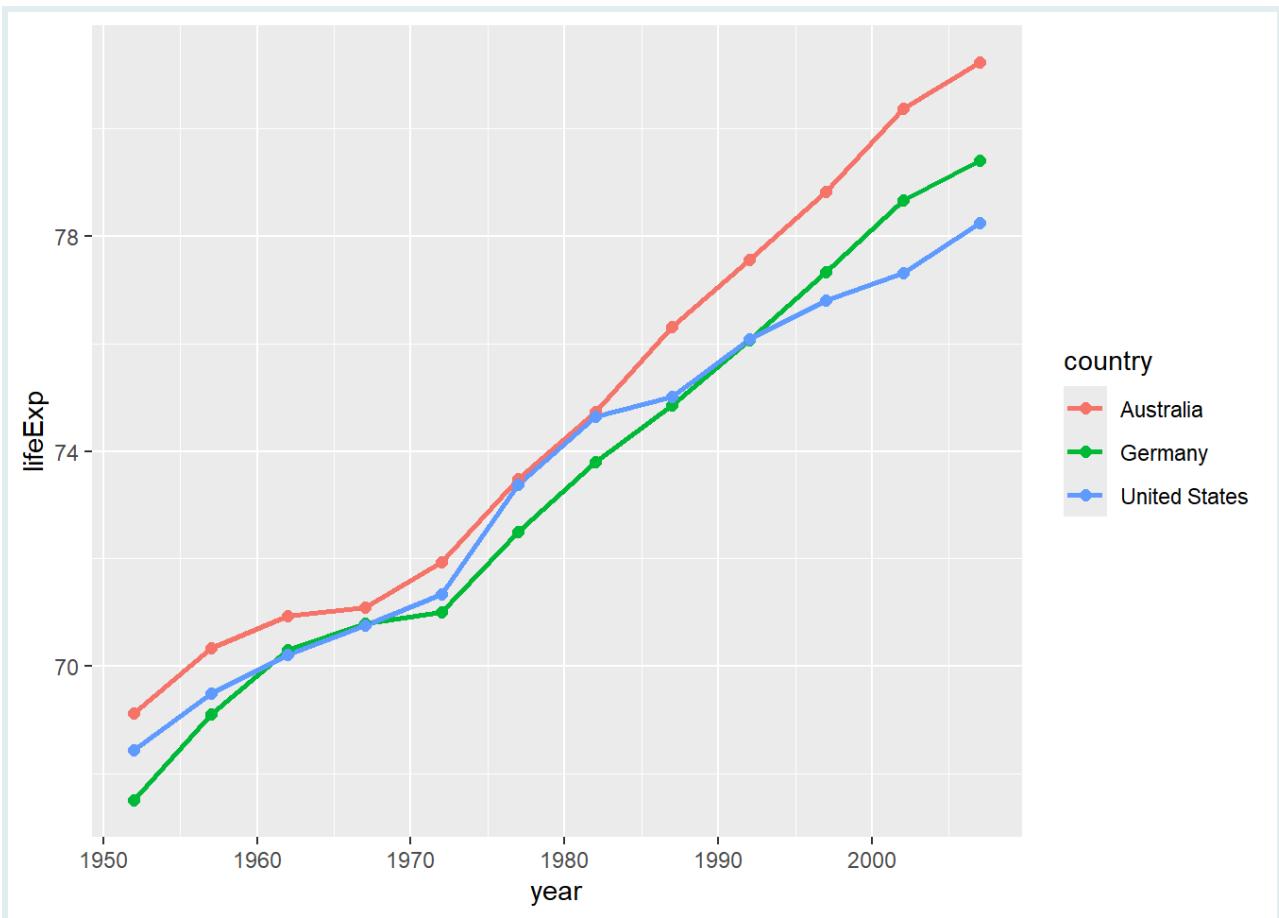
```
# Appliquer des esthétiques fixes à plusieurs lignes
ggplot(data = gap_mini,
       mapping = aes(y = lifeExp,
                     x = year,
                     group = country)) +
  geom_line(linetype="longdash",           # définir le type de ligne
            color="tomato",             # définir la couleur de la ligne
            size=1) +                  # définir l'épaisseur de la ligne
  geom_point(size = 2)                  # définir la taille du point
```



Dans le graphiques ci-dessus, le type, la couleur et l'épaisseur des lignes sont les mêmes pour les trois groupes.

Cela ne nous permet pas de distinguer les groupes. Il faut ajouter un mapping esthétique qui peut nous aider à identifier à quel pays appartient chaque ligne, comme la couleur ou le type de ligne.

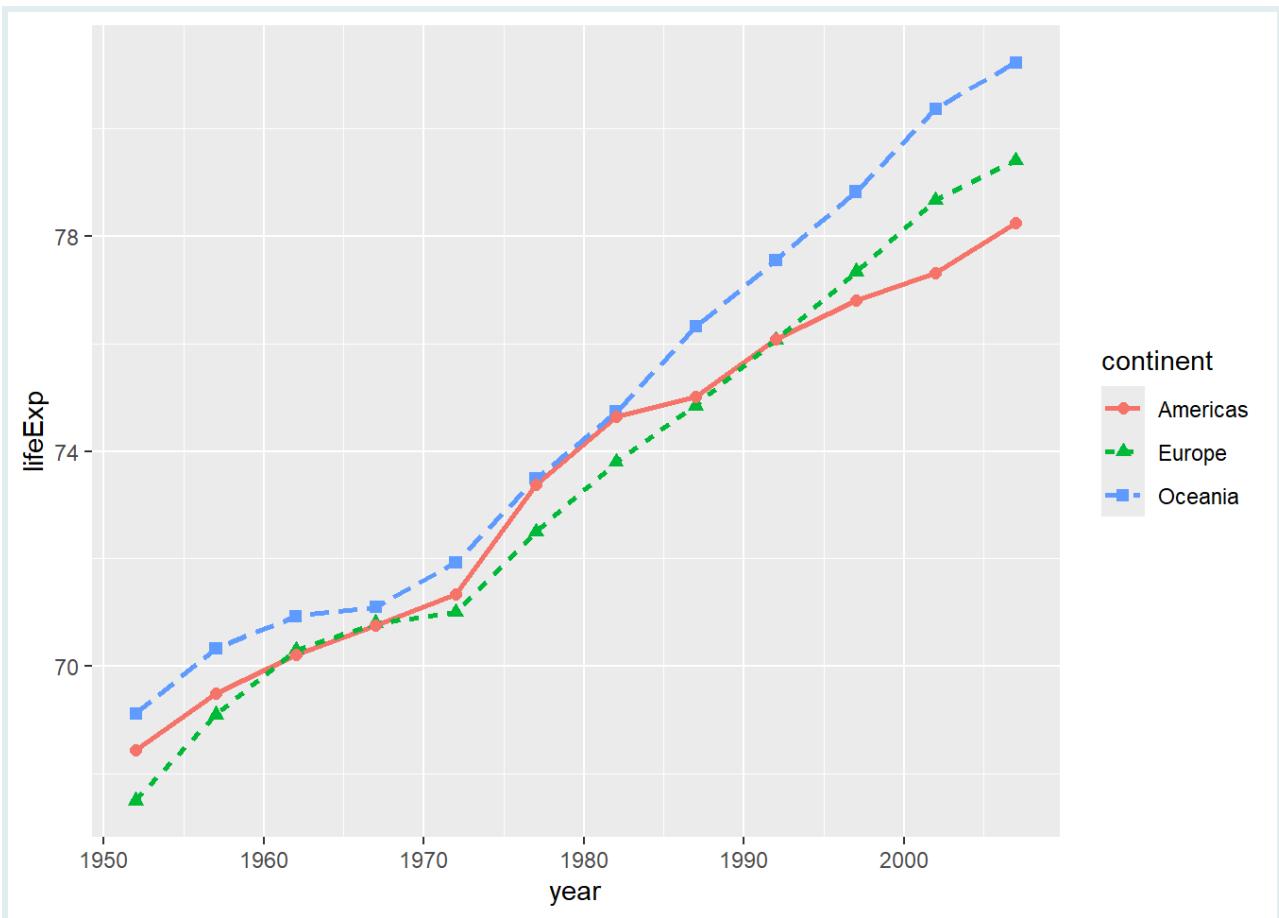
```
# Mapper le pays à la couleur
ggplot(data = gap_mini,
        mapping = aes(y = lifeExp, x = year,
                      group = country,
                      color = country)) +
  geom_line(size = 1) +
  geom_point(size = 2)
```



Les mappings esthétiques spécifiés dans l'appel de la fonction `ggplot()` sont transmis aux couches ultérieures.

Au lieu de grouper par `country`, nous pouvons également grouper par `continent` :

```
# Mapper le continent à la couleur, au type de ligne et à la forme des points
ggplot(data = gap_mini,
        mapping = aes(x = year,
                      y = lifeExp,
                      color = continent,
                      lty = continent,
                      shape = continent)) +
  geom_line(size = 1) +
  geom_point(size = 2)
```



Lorsque nous fournissons plusieurs mappings et geoms, {ggplot2} peut discerner quels mappings s'appliquent à quels geoms.

Ici, `color` a été appliqué aux points et aux lignes, mais `lty` a été ignoré par `geom_point()` et `shape` a été ignoré par `geom_line()`, puisqu'ils ne peuvent pas être appliqués.

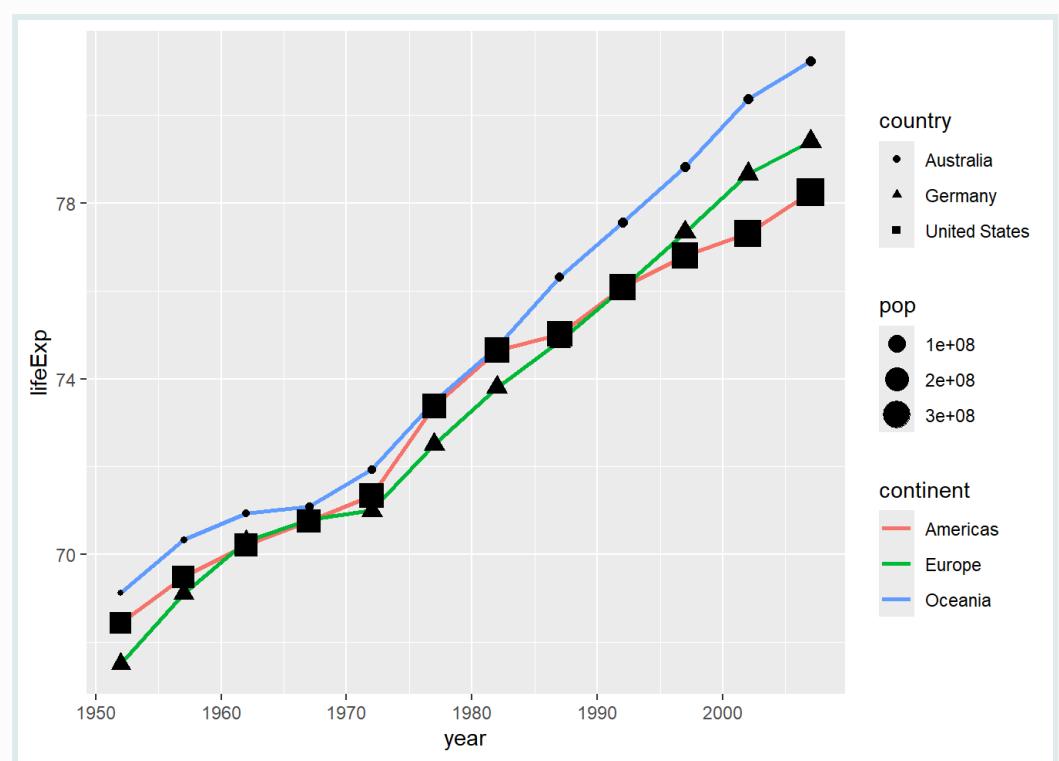
Les mappings sont inclus soit dans la fonction `ggplot()`, soit dans la couche `geom_*`.

### CHALLENGE



Par exemple, les mappings esthétiques peuvent aller dans `geom_line()` et ne seront appliqués qu'à cette couche :

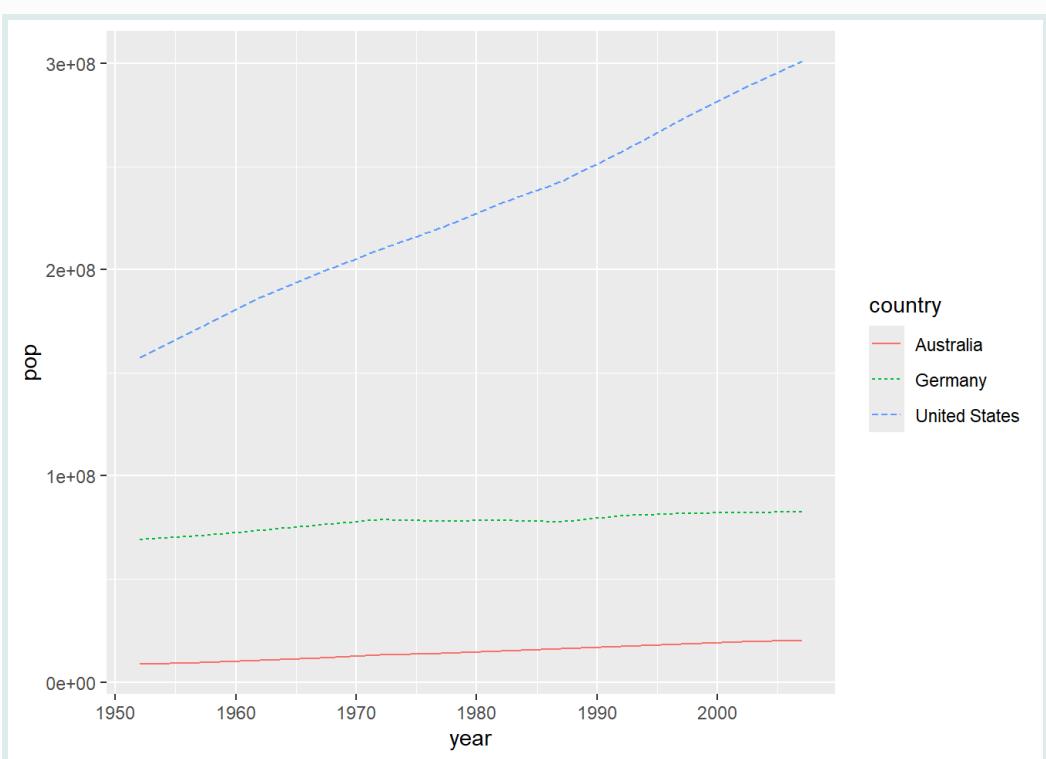
```
ggplot(data = gap_mini,
       mapping = aes(x = year,
                     y = lifeExp)) +
  geom_line(size = 1, mapping = aes(color = continent)) +
  geom_point(mapping = aes(shape = country,
                           size = pop))
```



Essayez d'ajouter `mapping= aes()` dans `geom_point()` et mappez continent à une esthétique appropriée !



En utilisant le dataframe `gap_mini`, créez un graphique de la croissance de la population avec ces mappings esthétiques :

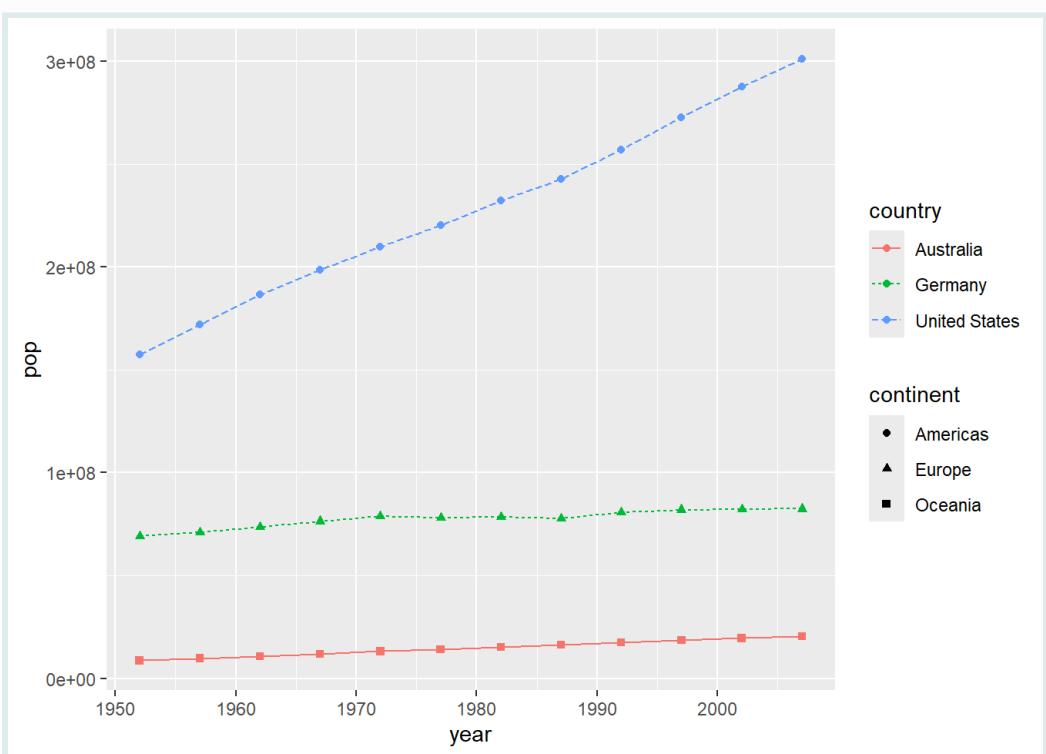


## PRACTICE



(in RMD)

Ensuite, ajoutez une couche de points au graphique précédent, et ajoutez les mappings esthétiques requis pour produire un graphique qui ressemble à ceci :



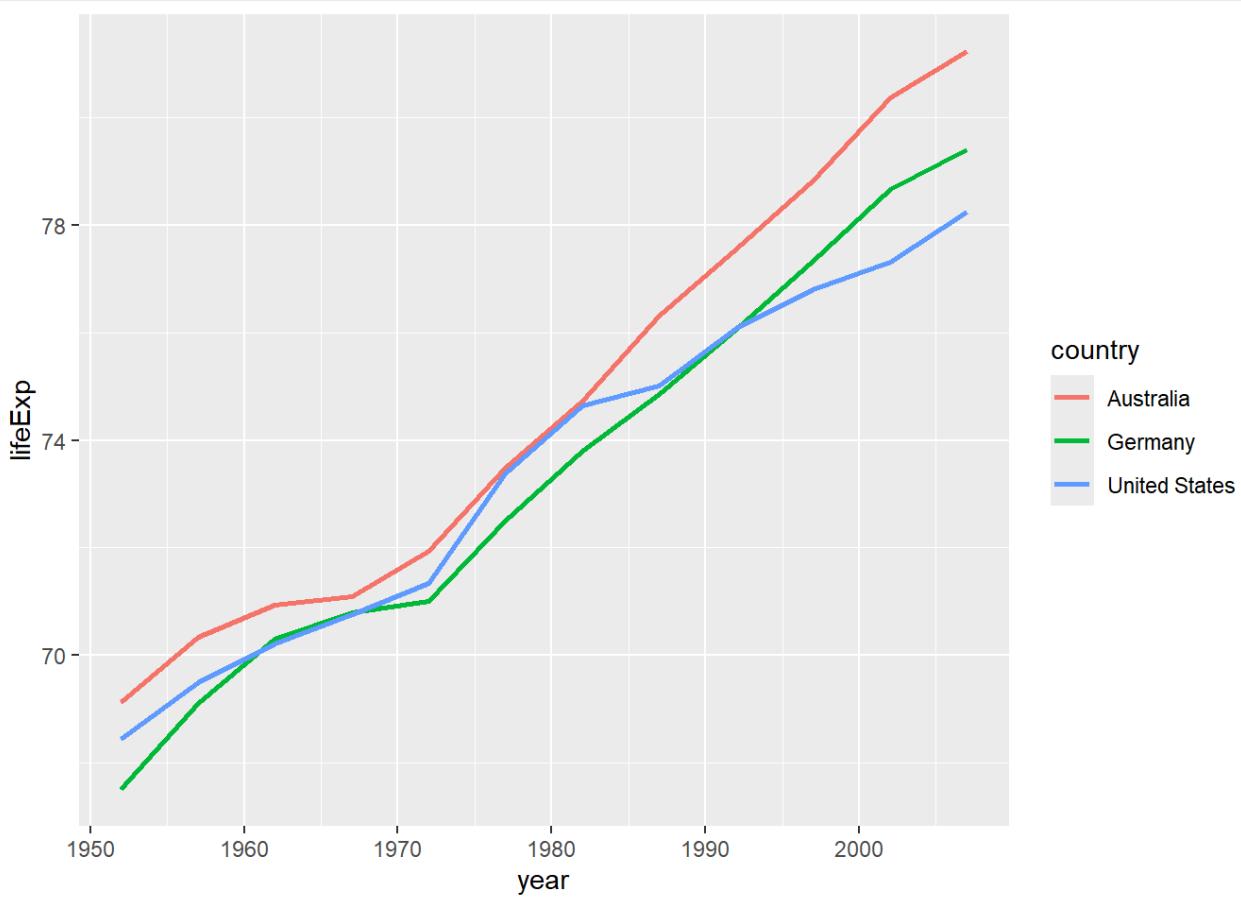


Ne vous souciez pas des esthétiques fixes, assurez-vous seulement que le mapping des variables est le même.

## Modifier les échelles continues x et y

{ggplot2} choisit automatiquement l'échelle à utiliser en fonction du type de variable.

```
# Echelle par défaut pour x, y et color
ggplot(data = gap_mini,
       mapping = aes(x = year,
                      y = lifeExp,
                      color = country)) +
  geom_line(size = 1)
```



Dans certains cas, il faut transformer l'échelle des axes pour une meilleure visualisation. Nous pouvons personnaliser ces échelles avec la famille de fonctions `scale_*`.

```

ggplot(data = <DATAFRAME>,
       mapping = aes(<VARS À MAPPER>) + ] OBLIGATOIRE
<FONCTION_GEOM>() +
       stat = <STAT>, position = <POSITION> ) + ] OPTIONNELLE
       <FONCTION_COORDONNEES> +
       <FONCTION_FACET> +
       <FONCTION_SCALE> +
       <FONCTION_THEME>

```

`scale_x_continuous()` et `scale_y_continuous()` sont les fonctions utilisées pour personnaliser les échelle x et y d'un graphique lorsque les données sur ces axes sont continues.

#### Scales couramment utilisées

A utiliser avec tous paramètres esthétiques:

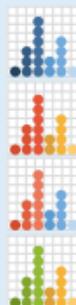
- `scale_*_continuous()` - échelle continue
- `scale_*_discrete()` - échelle discrète
- `scale_*_identity()` - échelle identité
- `scale_*_manual(values = c())` - permet de choisir manuellement les valeurs de l'échelle
- `scale_*_date(date_labels = "%m/%d"), date_breaks = "2 weeks")` - considère les valeurs en tant que date.
- `scale_*_datetime()` considère les valeurs de x en tant que datetime. Utilise les mêmes arguments que `scale_x_date()`.

#### Scales associées à X et Y

A utiliser avec le paramètre esthétique x ou y (exemple ici avec x)

- `scale_x_log10()` - échelle logarithmique pour l'axe x
- `scale_x_reverse()` - inverse l'axe des x
- `scale_x_sqrt()` - échelle « racine carrée » pour l'axe x

#### Scales de couleur et remplissage (continue)



- `o <- a + geom_dotplot(aes(fill = ..x..))`
- `o + scale_fill_distiller(palette = «Blues»))`
- `o + scale_fill_gradient(low = "red", high = "yellow")`
- `o + scale_fill_gradient2(low = "red", high = "blue", mid = "white", midpoint = 25)`
- `o + scale_fill_gradientn(colours = topo.colors(6))`  
voir : `rainbow()`, `heat.colors()`, `terrain.colors()`, `cm.colors()`, `RColorBrewer::brewer.pal()`

#### Scales de forme et de taille



- `p <- e + geom_point(aes(shape = fl, size = cyl))`
- `p + scale_shape() + scale_size()`
- `p + scale_shape_manual(values = c(3:7))`
- `0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25`
- `□○△+×◊▽▣*◊⊕▣田▣□○△◊○○○□◊△▽`
- `p + scale_radius(range = c(1,6))`
- `p + scale_size_area(max_size = 6)`

## Modifier les graduations des axes

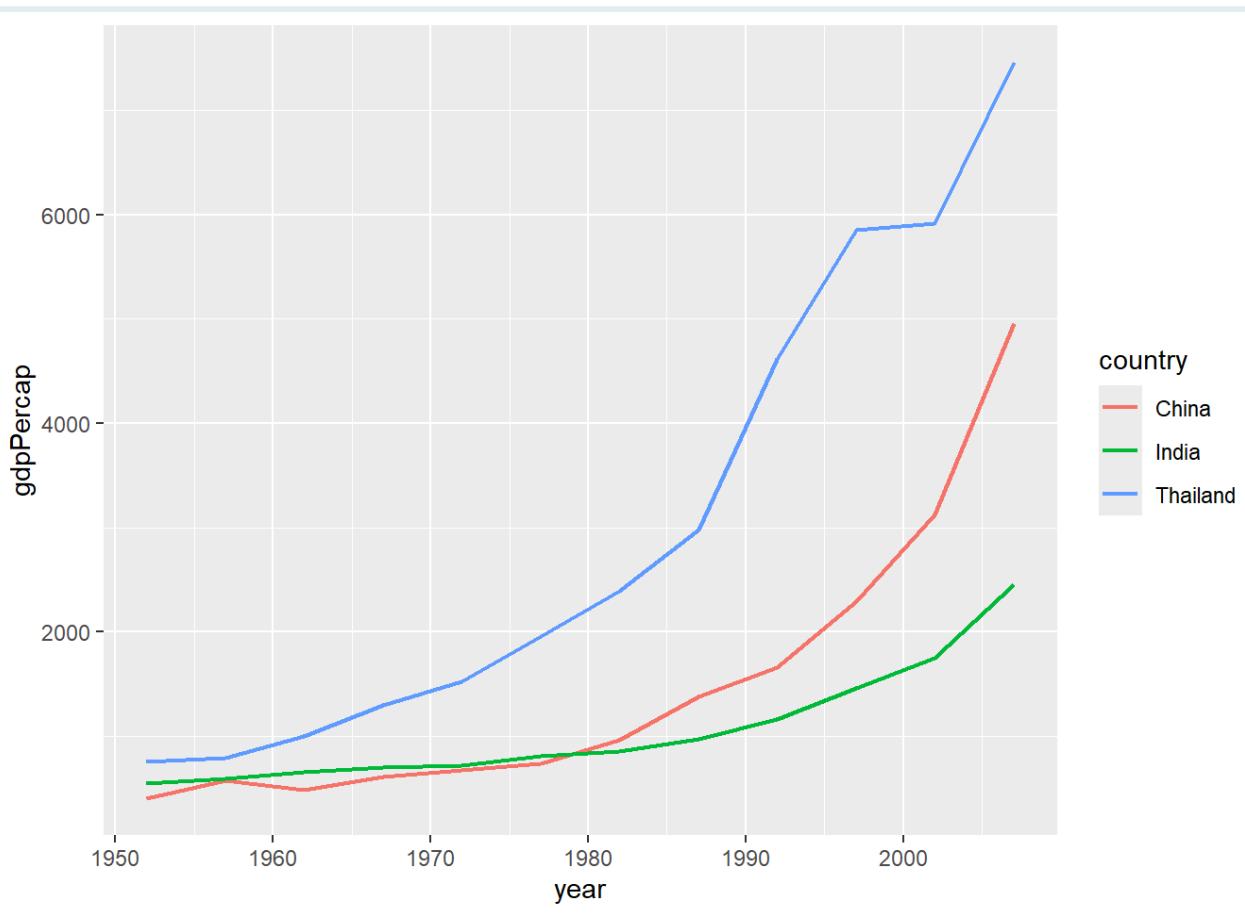
Créons un nouveau subset de pays à partir de `gapminder`. Cette fois, nous allons tracer l'évolution du PIB au fil du temps.

```
# Subset de données incluant l'Inde, la Chine et la Thaïlande
gap_mini2 <- filter(gapminder,
                      country %in% c("India",
```

```
"China",
"Thailand"))
```

Ici, nous allons changer le mapping de l'axe des y de lifeExp à gdpPercap :

```
ggplot(data = gap_mini2,
       mapping = aes(x = year,
                     y = gdpPercap,
                     group = country,
                     color = country)) +
  geom_line(size = 0.75)
```



Les étiquettes de l'axe des x pour year ne correspondent pas aux années dans le dataset.

```
gap_mini2$year %>% unique()
```

```
## [1] 1952 1957 1962 1967 1972 1977 1982 1987
## [9] 1992 1997 2002 2007
```

Nous pouvons spécifier exactement où étiqueter l'axe en fournissant un vecteur numérique.

```
# Vous pouvez entrer manuellement les graduations (ne faites pas ça)
c(1952, 1957, 1962, 1967, 1972, 1977, 1982, 1987, 1992, 1997, 2002, 2007)
```

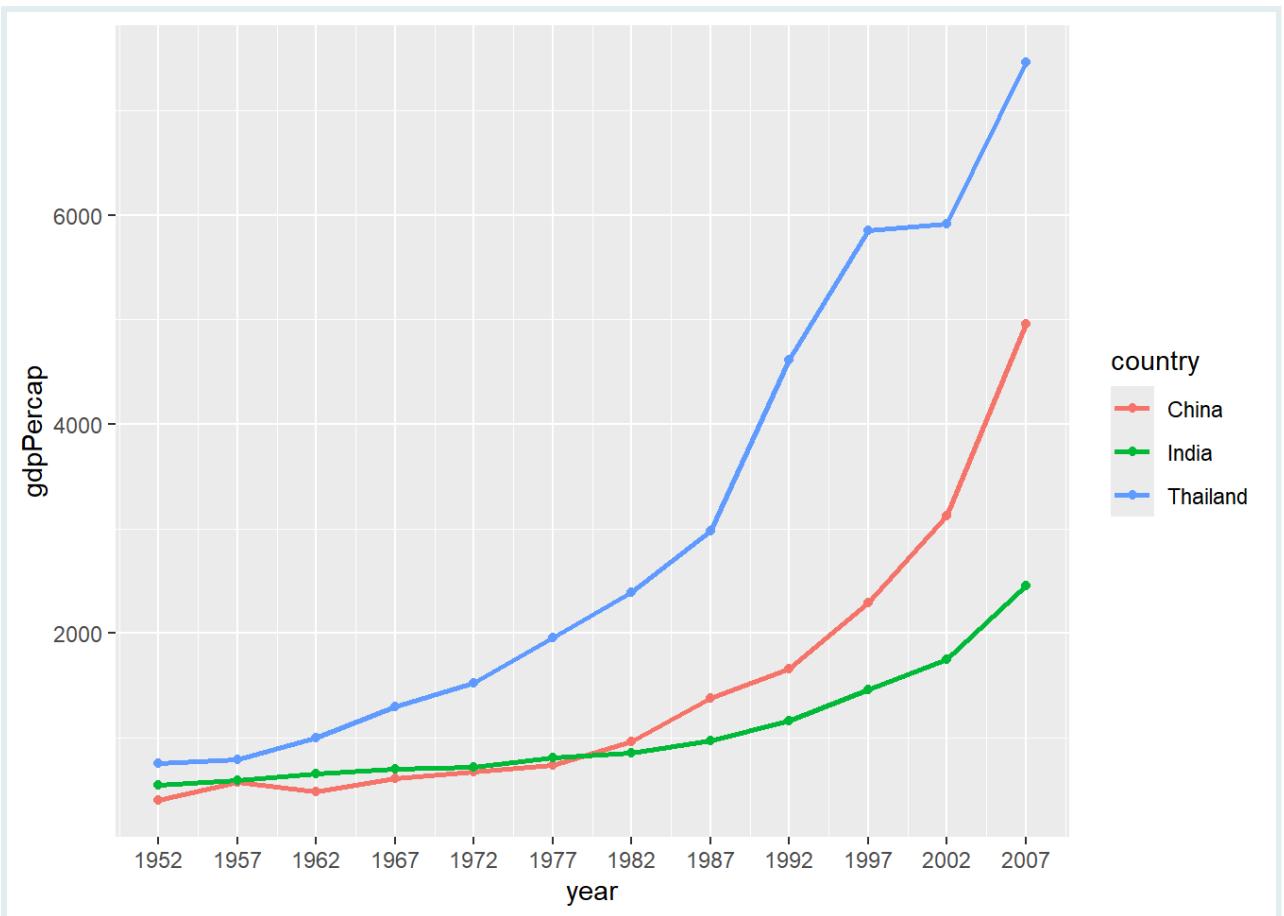
```
## [1] 1952 1957 1962 1967 1972 1977 1982 1987
## [9] 1992 1997 2002 2007
```

```
# Il est préférable de créer le vecteur avec seq()
seq(from = 1952, to = 2007, by = 5)
```

```
## [1] 1952 1957 1962 1967 1972 1977 1982 1987
## [9] 1992 1997 2002 2007
```

Utilisez `scale_x_continuous` pour faire correspondre les graduations avec le dataset :

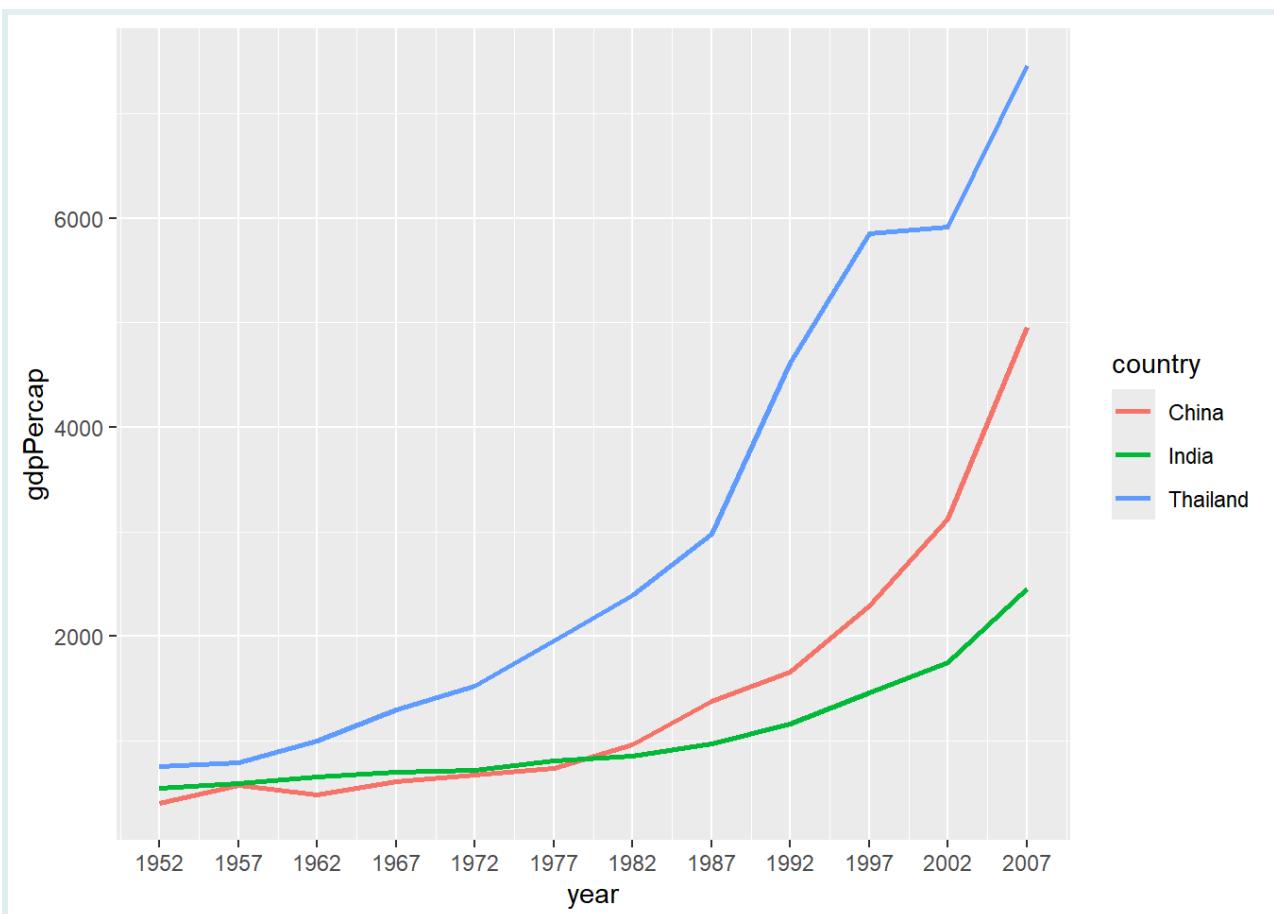
```
# Personnalisez les graduations des x avec `scale_x_continuous (breaks =
VECTEUR)
ggplot(data = gap_mini2,
       mapping = aes(x = year,
                     y = gdpPercap,
                     color = country)) +
  geom_line(size = 1) +
  scale_x_continuous(breaks = seq(from = 1952,
                                   to = 2007,
                                   by = 5)) +
  geom_point()
```



Stockez les valeurs des graduations dans un objet R pour les référencer plus facilement.

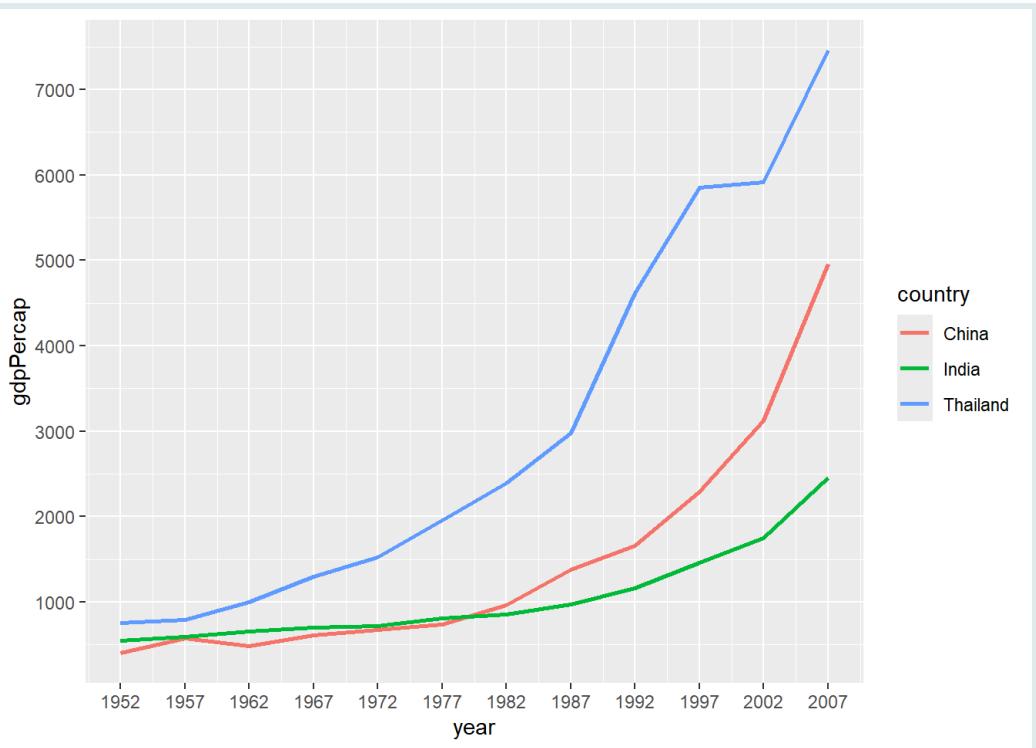
```
# Stocker le vecteur numérique dans un objet
gap_years <- seq(from = 1952,
                  to = 2007,
                  by = 5)
```

```
# Remplacez le code seq() par l'objet R
ggplot(data = gap_mini2,
        mapping = aes(x = year,
                      y = gdpPercap,
                      color = country)) +
  geom_line(size = 1) +
  scale_x_continuous(breaks = gap_years)
```



Nous pouvons personnaliser les graduations de l'axe des y continu avec `scale_y_continuous()`.

Copiez le code du dernier exemple et ajoutez `scale_y_continuous()` pour ajouter les graduations de l'axe des y suivantes :



## Définir une échelle logarithmique

Dans les deux derniers mini-datasets, nous avons choisi trois pays qui avaient des PIB ou une espérance de vie similaires afin que nous puissions les comparer facilement.

Mais si nous ajoutons un pays qui diffère significativement, l'échelle utilisée par défaut ne convient plus dans ce cas.

Nous allons voir un exemple où il vaut mieux convertir les axes de l'échelle linéaire par défaut à une échelle logarithmique.

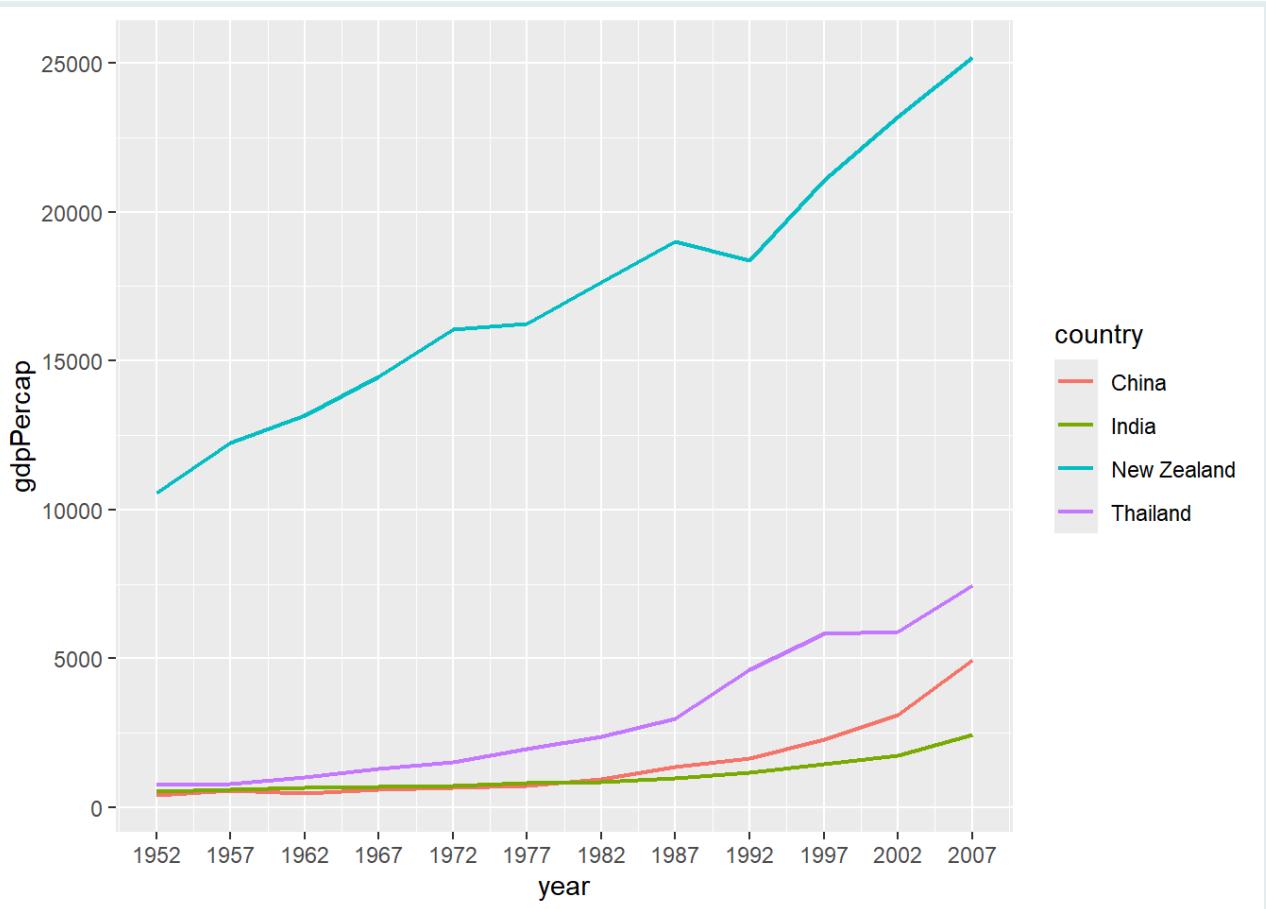
Ajoutons la Nouvelle-Zélande au dataset précédent et créons `gap_mini3` :

```
# Nouveau subset pour inclure l'Inde, la Chine, la Thaïlande et la Nouvelle-Zélande
gap_mini3 <- filter(gapminder,
                     country %in% c("India",
                                   "China",
                                   "Thailand",
                                   "New Zealand"))
```

gap\_mini3

Maintenant, nous allons recréer le graphique du PIB au fil du temps avec le nouveau subset :

```
ggplot(data = gap_mini3,
       mapping = aes(x = year,
                     y = gdpPercap,
                     color = country)) +
  geom_line(size = 0.75) +
  scale_x_continuous(breaks = gap_years)
```

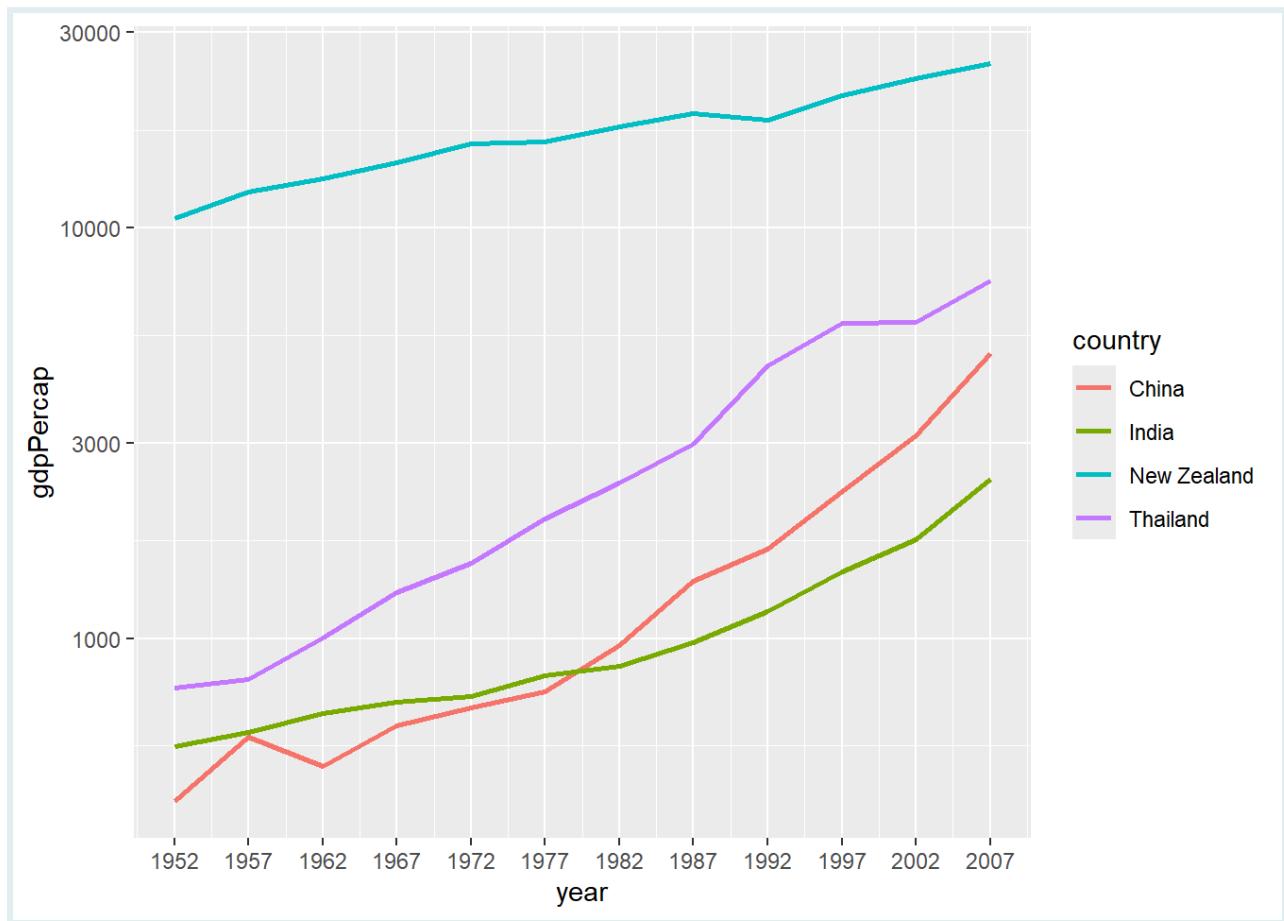


Les courbes pour l'Inde et la Chine montrent une augmentation exponentielle du PIB par habitant. Cependant, les valeurs de l'axe des **y** pour ces deux pays sont beaucoup plus faibles que celle de la Nouvelle-Zélande, donc les lignes sont un peu "tassées". Cela rend les données difficiles à lire. De plus, nous nous retrouvons avec une grande zone vide au milieu.

Nous pouvons résoudre ce problème avec une transformation logarithmique de l'axe des **y** en utilisant `scale_y_log10()`. Nous ajouterons cette fonction en tant que nouvelle couche avec l'opérateur `+`, comme d'habitude :

```
# Ajouter scale_y_log10()
ggplot(data = gap_mini3,
       mapping = aes(x = year,
                     y = gdpPercap,
                     color = country)) +
  geom_line(size = 1) +
```

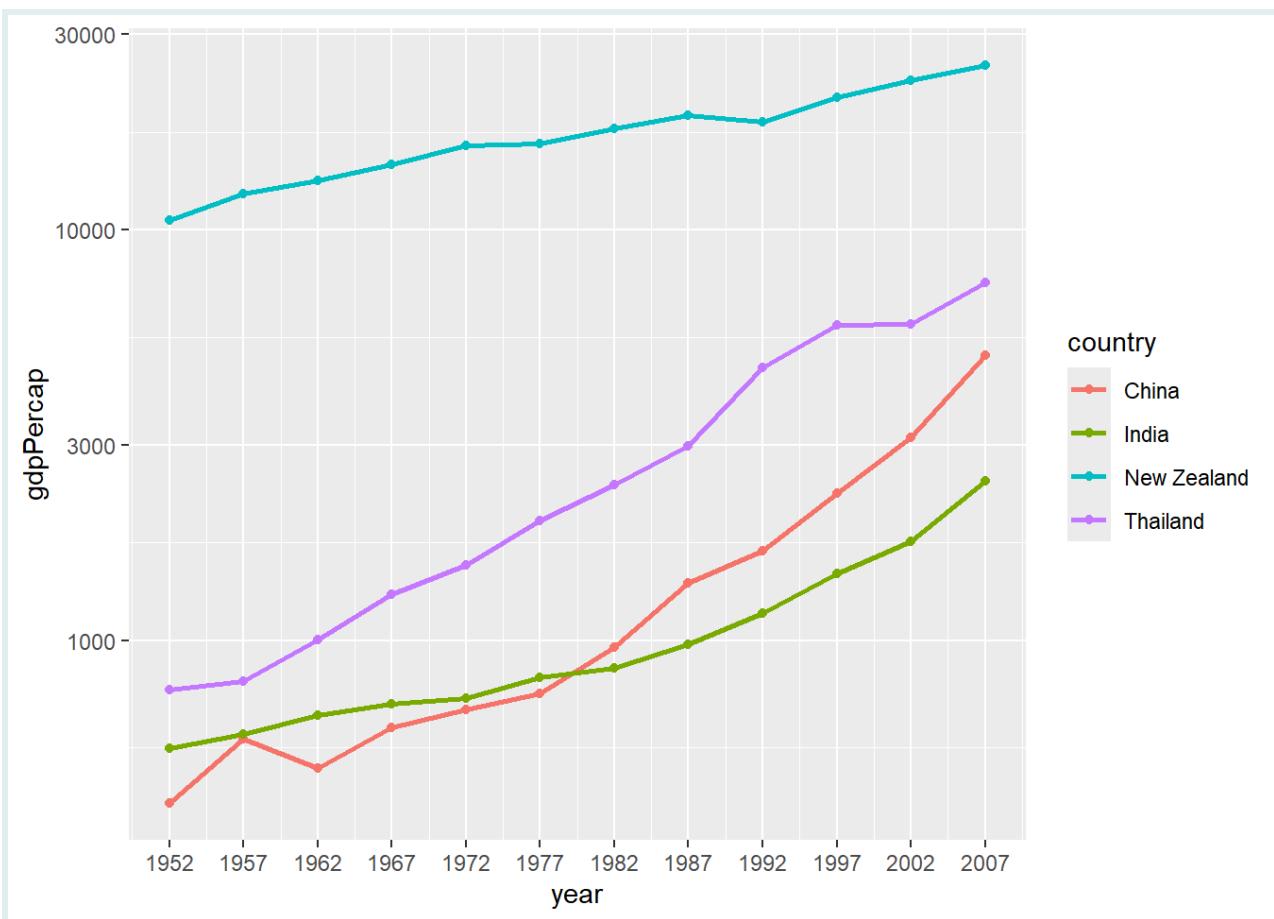
```
scale_x_continuous(breaks = gap_years) + scale_y_log10()
```



Nous avons changé l'échelle de l'axe des y, et les étiquettes des graduations d'échelle nous indiquent qu'elle est non linéaire.

Nous pouvons ajouter une couche de points pour rendre cela plus clair :

```
ggplot(data = gap_mini3,
       mapping = aes(x = year,
                      y = gdpPercap,
                      color = country)) +
  geom_line(size = 1) +
  scale_x_continuous(breaks = gap_years) +
  scale_y_log10() +
  geom_point()
```



### PRACTICE



Tout d'abord, créez un subest à partir de `gapminder` contenant uniquement les données de l'Uganda :

Maintenant, utilisez `gap_uganda` pour créer un graphique de série temporelle de la population (`pop`) au fil du temps (`year`). Transformez l'échelle de l'axe des y en une échelle logarithmique, modifiez les graduations en utilisant `gap_years`, changez la couleur de la ligne à `forestgreen` et l'épaisseur à 1mm.

Ensuite, nous pouvons changer le texte des étiquettes des axes pour qu'il soit plus descriptif, et ajouter un titre, un sous-titre et d'autres étiquettes informatives au graphique.

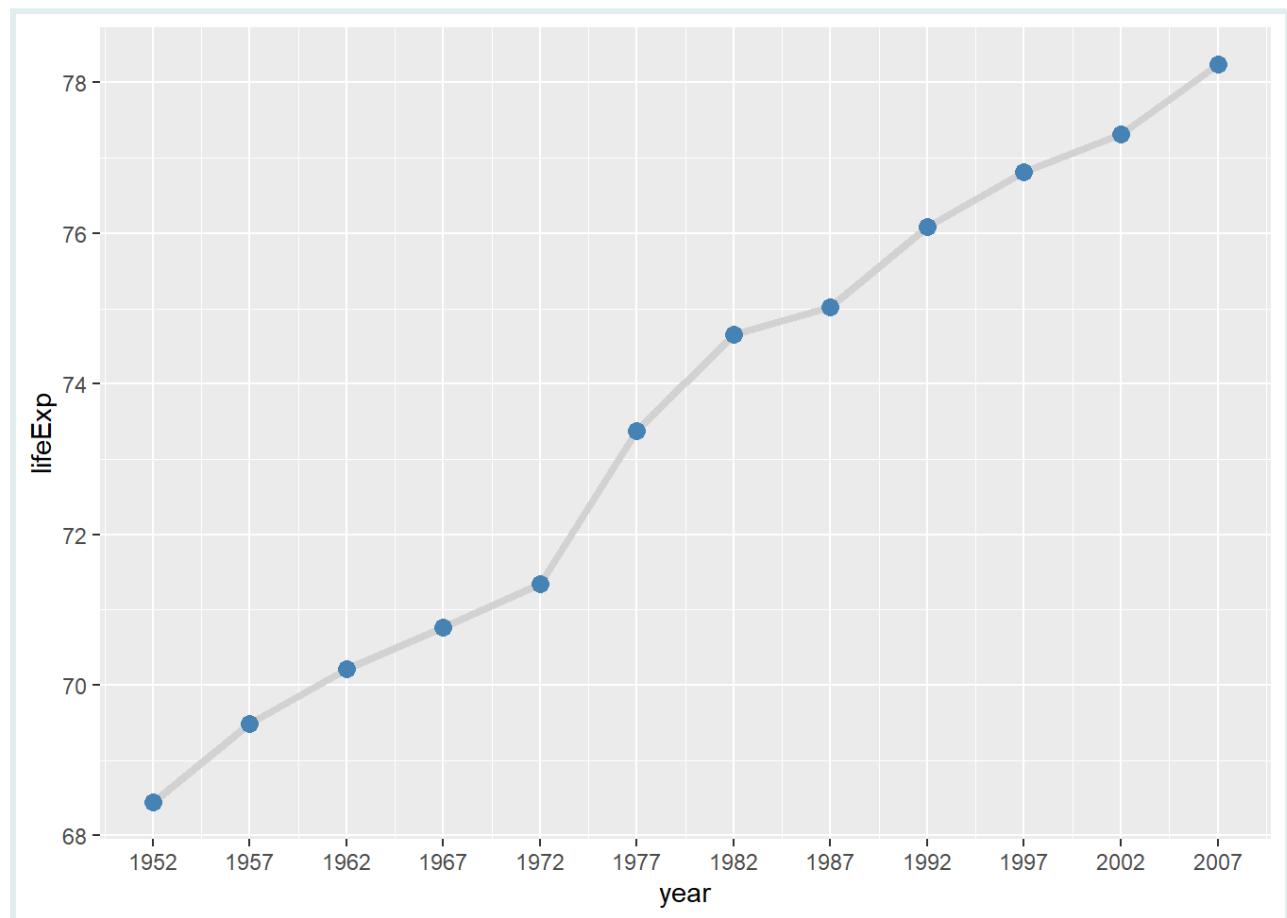
## Étiquetage avec `labs()`

Vous pouvez ajouter des étiquettes (labels) à un graphique avec la fonction `labs()`. Les arguments que nous pouvons spécifier avec la fonction `labs()` incluent :

- **title**: Changer ou ajouter un titre
- **subtitle**: Ajouter un sous-titre sous le titre
- **x**: Renommer l'axe des x
- **y**: Renommer l'axe des y
- **caption**: Ajouter une note de bas de page sous le graphique

Commençons par ajouter des étiquettes à ce graphique :

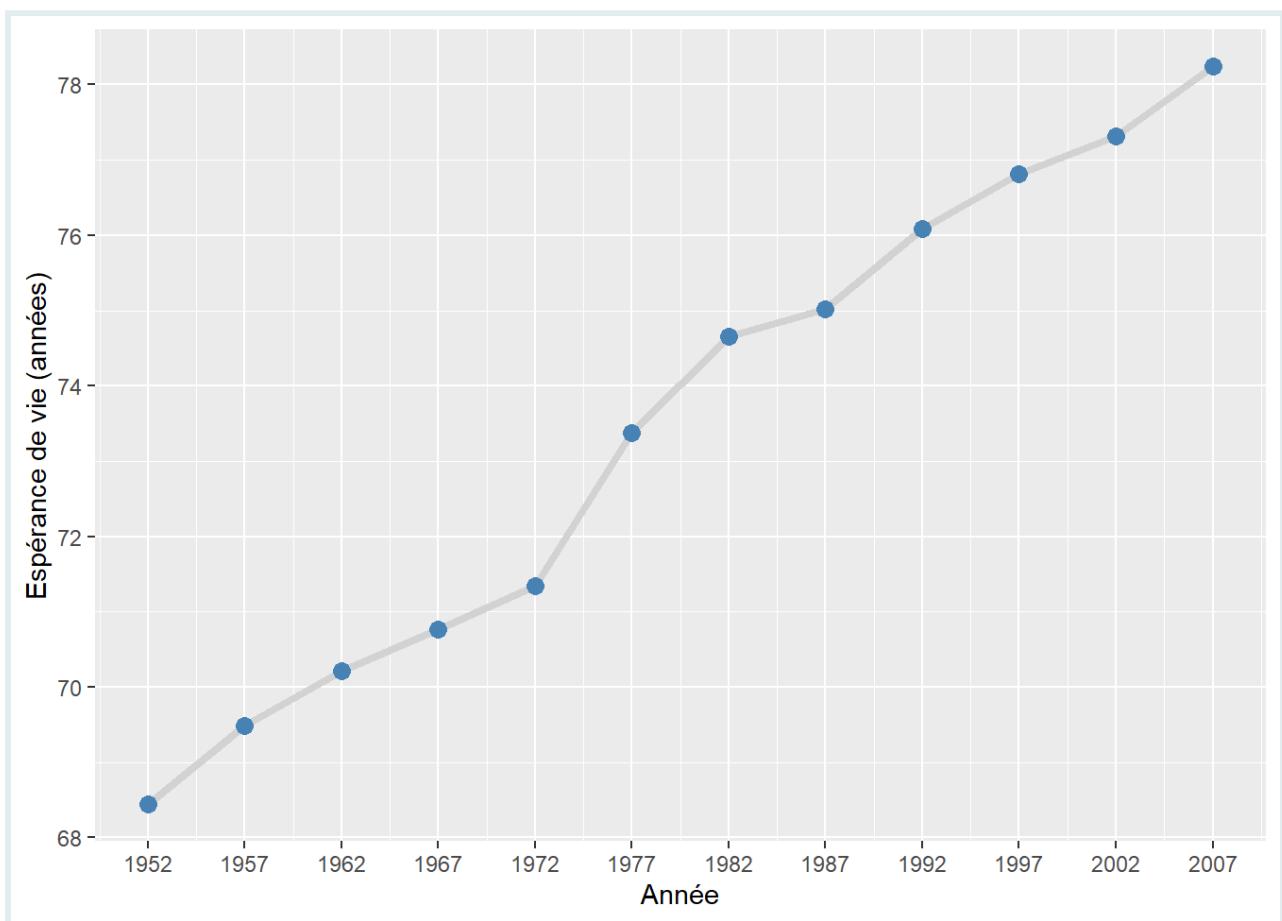
```
# Série temporelle de l'espérance de vie aux États-Unis
ggplot(data = gap_US,
       mapping = aes(x = year,
                      y = lifeExp)) +
  geom_line(size = 1.5,
            color = "lightgrey") +
  geom_point(size = 3,
             color = "steelblue") +
  scale_x_continuous(breaks = gap_years)
```



Nous ajoutons `labs()` à notre code en utilisant l'opérateur `+`.

D'abord, nous allons ajouter les arguments `x` et `y` à `labs()`, et changer les titres des axes de la valeur par défaut (nom de la variable) à quelque chose de plus informatif.

```
# Changer les titres des axes
ggplot(data = gap_US,
       mapping = aes(x = year,
                      y = lifeExp)) +
  geom_line(size = 1.5,
            color = "lightgrey") +
  geom_point(size = 3,
             color = "steelblue") +
  scale_x_continuous(breaks = gap_years) +
  labs(x = "Année",
       y = "Espérance de vie (années)")
```

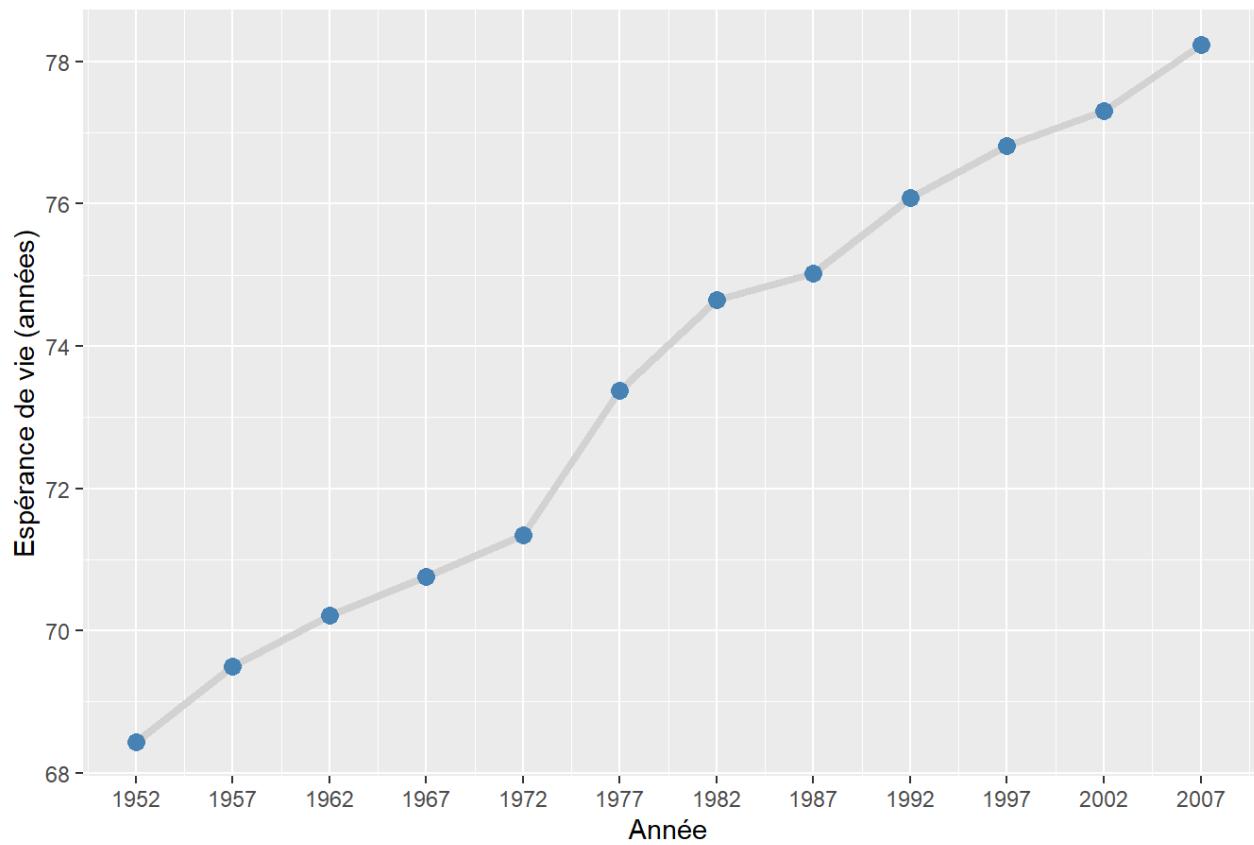


Ensuite, nous allons ajouter un titre au-dessus du graphique avec l'argument `title`.

```
# Ajouter un titre principal: "L'espérance de vie augmente avec le temps"
ggplot(data = gap_US,
       mapping = aes(x = year,
                      y = lifeExp)) +
  geom_line(size = 1.5,
            color = "lightgrey") +
  geom_point(size = 3,
             color = "steelblue") +
  scale_x_continuous(breaks = gap_years) +
  labs(x = "Année",
```

```
y = "Espérance de vie (années)",  
title = "L'espérance de vie augmente avec le temps")
```

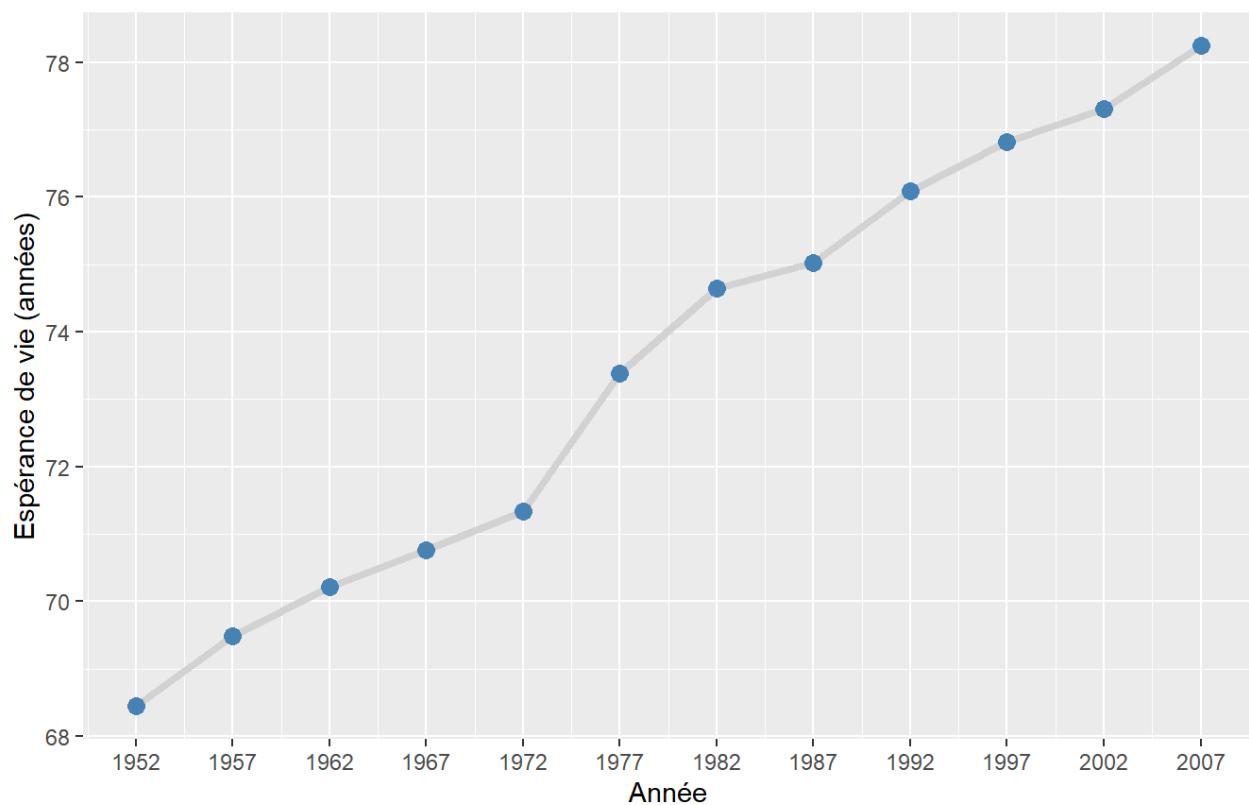
L'espérance de vie augmente avec le temps



L'argument `subtitle` ajoute un sous-titre sous le titre principal.

```
# Ajouter un sous-titre avec le lieu et la période  
ggplot(data = gap_US,  
        mapping = aes(x = year,  
                      y = lifeExp)) +  
  geom_line(size = 1.5,  
            color = "lightgrey") +  
  geom_point(size = 3,  
             color = "steelblue") +  
  scale_x_continuous(breaks = gap_years) +  
  labs(x = "Année",  
       y = "Espérance de vie (années)",  
       title = "Changement de l'espérance de vie au fil du temps",  
       subtitle = "États-Unis (1952-2007)")
```

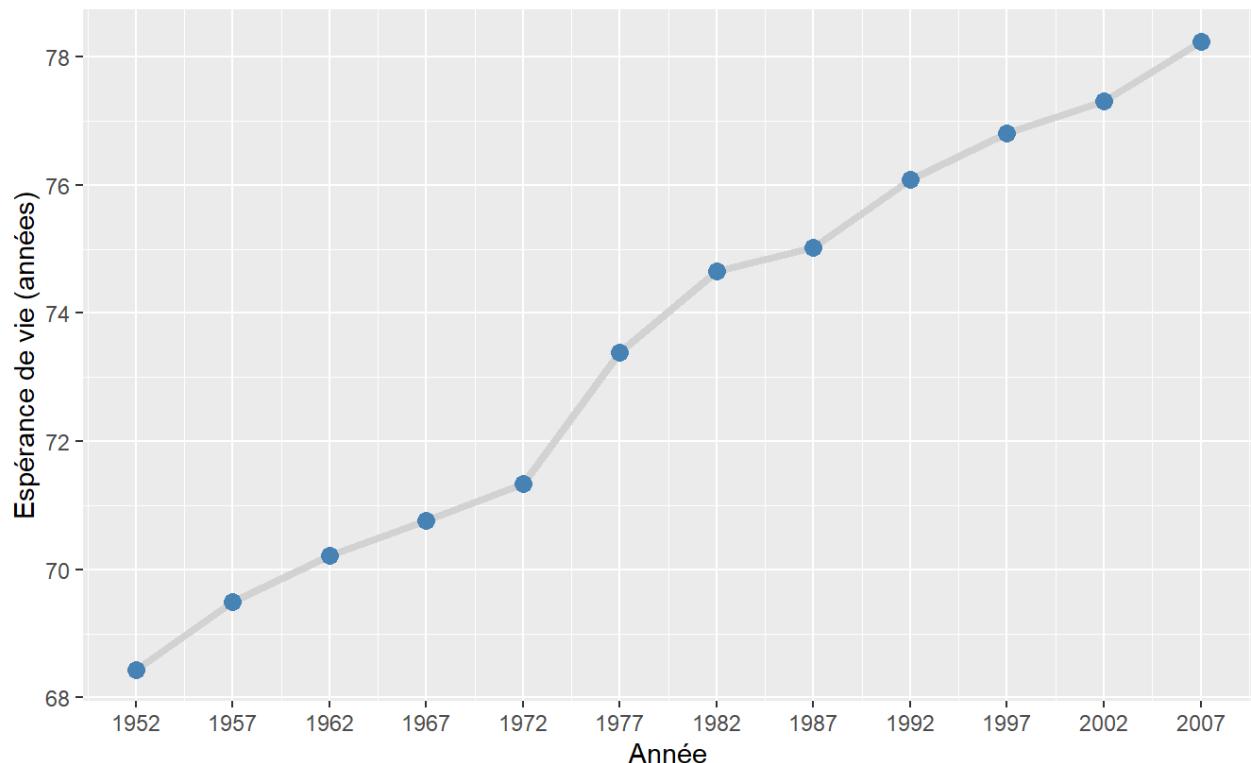
## Changement de l'espérance de vie au fil du temps États-Unis (1952-2007)



Enfin, nous pouvons fournir l'argument `caption` pour ajouter une note de bas de page sous le graphique.

```
# Ajouter une note de bas de page avec la source des données : "Source :  
# www.gapminder.org/data"  
ggplot(data = gap_US,  
       mapping = aes(x = year,  
                      y = lifeExp)) +  
  geom_line(size = 1.5,  
            color = "lightgrey") +  
  geom_point(size = 3,  
             color = "steelblue") +  
  scale_x_continuous(breaks = gap_years) +  
  labs(x = "Année",  
       y = "Espérance de vie (années)",  
       title = "Changement de l'espérance de vie au fil du temps",  
       subtitle = "États-Unis (1952-2007)",  
       caption = "Source: http://www.gapminder.org/data/")
```

## Changement de l'espérance de vie au fil du temps États-Unis (1952-2007)



Lorsque vous utilisez un mapping esthétique (par exemple, `color` ou `size`), `{ggplot2}` crée automatiquement une échelle pour cette esthétique en fonction des données fournies et ajoute une légende.

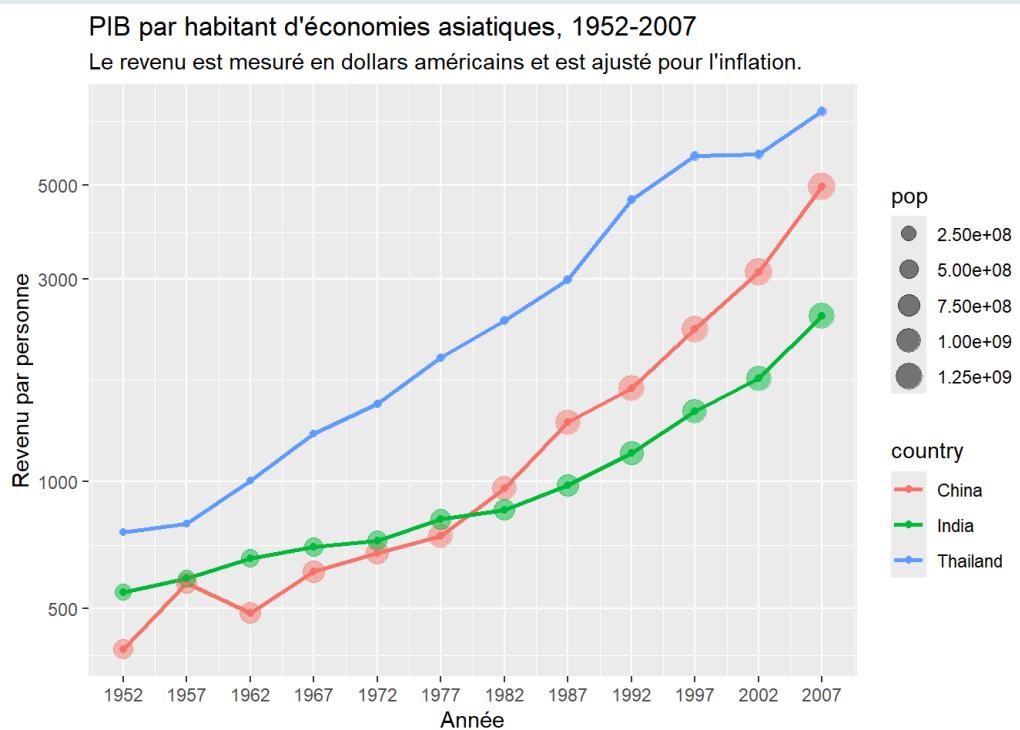
### CHALLENGE



Voici une version mise à jour du graphique `gap_mini3` que nous avons réalisé précédemment. Nous changeons la couleur des points et des lignes en définissant `aes(color = country)` dans `ggplot()`, et la taille des points en définissant `aes(fill = pop)`. Notez que `labs()` est utilisé pour changer le titre, le sous-titre et les étiquettes des axes.

```
ggplot(data = gap_mini2,
        mapping = aes(x = year,
                      y = gdpPercap,
                      color = country)) +
  geom_line(size = 1) +
  geom_point(mapping = aes(size = pop),
             alpha = 0.5) +
  geom_point() +
  scale_x_continuous(breaks = gap_years) +
  scale_y_log10()
```

```
labs(x = "Année",
      y = "Revenu par personne",
      title = "PIB par habitant d'économies asiatiques, 1952-2007",
      subtitle = "Le revenu est mesuré en dollars américains et est ajusté pour l'inflation.")
```



Le titre par défaut d'une légende est le nom de la variable à laquelle elle correspond. Ici, la légende de `color` est intitulée `country`, et la légende de `size` est intitulée `pop`.

Nous pouvons également les modifier dans `labs()` en définissant `NOM_AES = "NOUVEAU_TITRE"`.

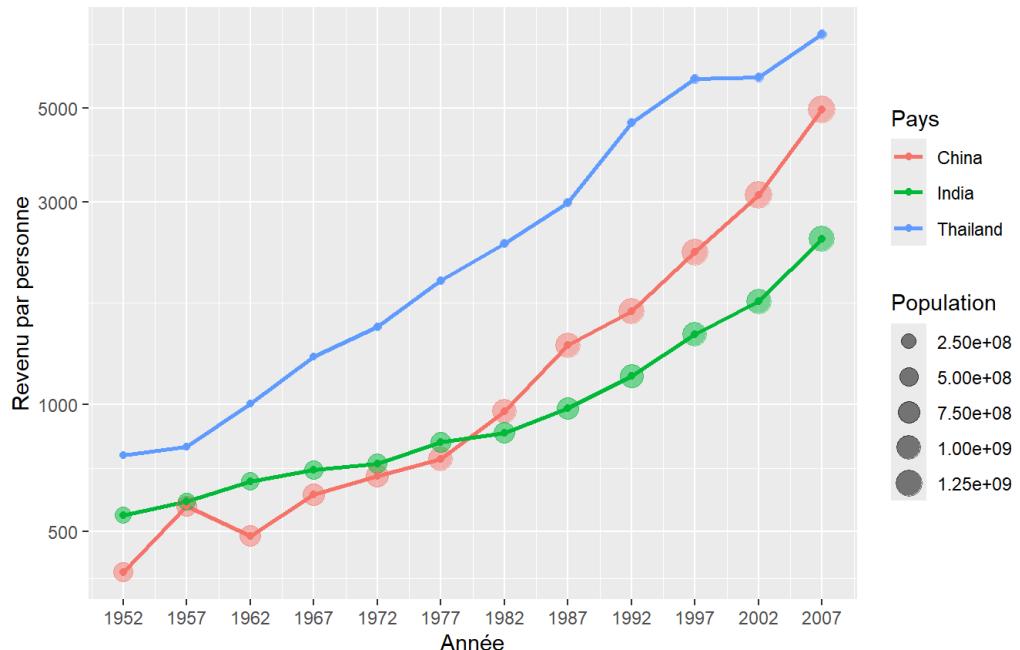
```
ggplot(data = gap_mini2,
       mapping = aes(x = year,
                      y = gdpPercap,
                      color = country)) +
  geom_line(size = 1) +
  geom_point(mapping = aes(size = pop),
             alpha = 0.5) +
  geom_point() +
  scale_x_continuous(breaks = gap_years) +
  scale_y_log10() +
  labs(x = "Année",
       y = "Revenu par personne",
       title = "PIB par habitant d'économies asiatiques, 1952-2007",
```



```
subtitle = "Le revenu est mesuré en dollars américains  
et est ajusté pour l'inflation.",  
color = "Pays",           size = "Population")
```

### PIB par habitant d'économies asiatiques, 1952-2007

Le revenu est mesuré en dollars américains et est ajusté pour l'inflation.



La même syntaxe peut être utilisée pour modifier les titres des légendes des autres mappings esthétiques. Une erreur courante est d'utiliser le nom de la variable au lieu du nom de l'esthétique dans `labs()`, donc faites attention !



Créez un graphique de série temporelle comparant les tendances du PIB par habitant de 1952 à 2007 pour **trois pays** dans le dataframe gapminder.

Tout d'abord, créez un subset contenant les données de trois pays de votre choix :

Utilisez `my_gap_mini` pour créer un graphique avec les attributs suivants :

- Ajoutez des points au graphique linéaire

- Augmentez l'épaisseur des lignes à 1mm et la taille des points à 2mm
- Rendez les lignes transparentes à 50%
- Changez les intervalles de l'échelle de l'axe des x pour correspondre aux années dans le jeu de données

Enfin, ajoutez les étiquettes suivantes à votre graphique :

- Titre : “Santé & richesse des nations”
- Titres d'axes : “Longévité” et “Année”
- Titre de la légende : “Pays”



```
# Écrivez le code pour créer votre graphique :
q8 <- "ÉCRIVEZ_VOTRE_CODEICI"

# Vérifiez votre réponse
	CHECK_q8()

## Incorrect. Votre résultat devrait être un objet ggplot2.
Veuillez réessayer.
## 1 2 3 4 5 6 7 ▷8◁

.HINT_q8()

## 
## Assurez-vous de ne pas ajouter de modifications
## supplémentaires que la question n'a pas demandées, sinon la
## fonction CHECK le considérera comme incorrect.
```

## Preview : Les thèmes

Dans le prochain cours, nous allons apprendre à utiliser les fonctions `theme`.

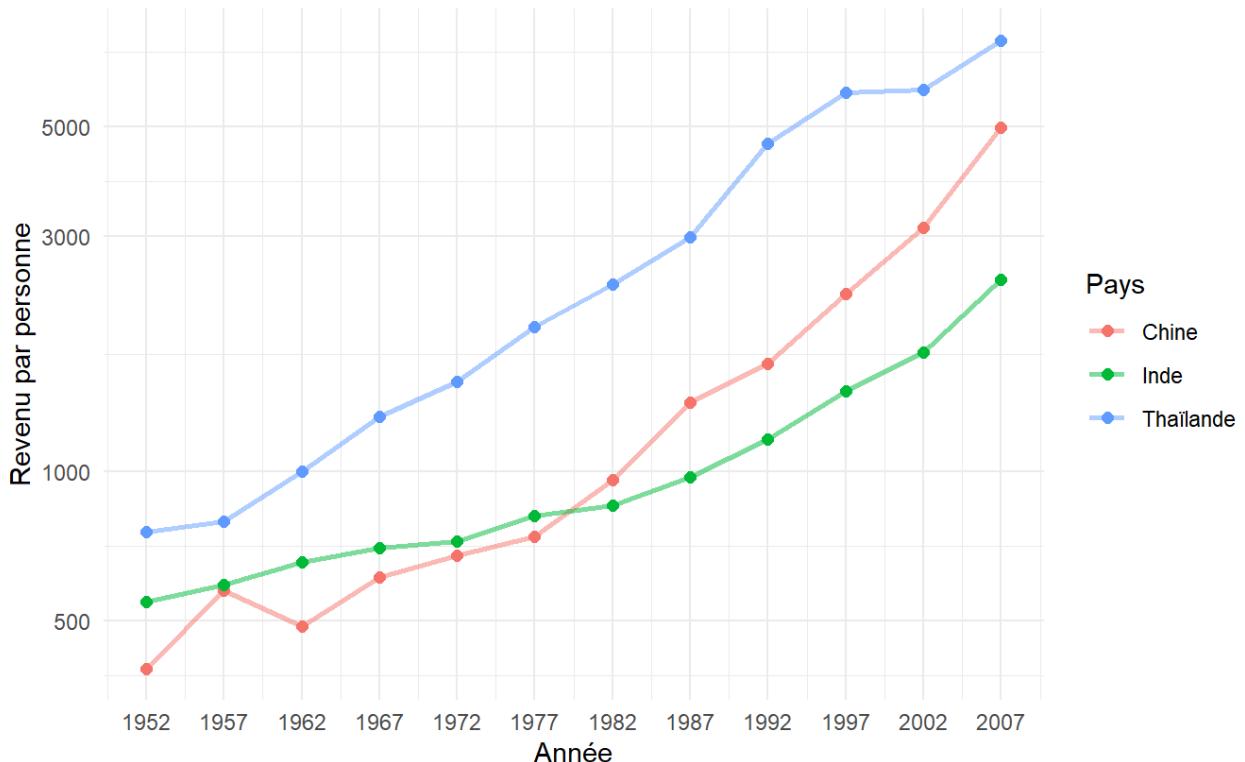
```

# Utiliser theme_minimal()
ggplot(data = gap_mini2,
       mapping = aes(x = year,
                     y = gdpPercap,
                     color = country)) +
  geom_line(size = 1, alpha = 0.5) +
  geom_point(size = 2) +
  scale_x_continuous(breaks = gap_years) +
  scale_y_log10() +
  labs(x = "Année",
       y = "Revenu par personne",
       title = "PIB par habitant d'économies asiatiques, 1952-2007",
       subtitle = "Le revenu est mesuré en dollars américains et est ajusté pour l'inflation.",
       caption = "Source : www.gapminder.org/data",
       color = "Pays") +
  scale_color_discrete(labels = c("China" = "Chine", "India" = "Inde",
                                 "Thailand" = "Thaïlande")) +
  theme_minimal()

```

## PIB par habitant d'économies asiatiques, 1952-2007

Le revenu est mesuré en dollars américains et est ajusté pour l'inflation.



---

## En résumé

Les graphiques linéaires, à l'instar des graphiques de dispersion, sont des outils efficaces pour représenter la relation entre deux variables numériques. Lorsque l'une de ces variables est temporelle, l'usage d'un graphique linéaire est plus adapté. De ce fait, il est conseillé d'opter pour des graphiques linéaires plutôt que des graphiques de dispersion lorsque la variable sur l'axe des abscisses (c'est-à-dire, la variable explicative) possède un ordre intrinsèque, comme c'est le cas pour une variable temporelle telle que `year` dans le dataframe `gapminder`.

Il est possible de transformer les échelles et de modifier les graduations des axes afin de rendre nos graphiques plus lisibles. De plus, l'ajout d'étiquettes permet d'incorporer davantage d'informations.

J'espère que vous avez trouvé ce cours instructif !

---

## Contributeurs

Les membres suivants ont contribué à ce cours :



### JOY VAZ

R Developer and Instructor, the GRAPH Network  
Loves doing science and teaching science



### IMANE BENSOUDA KORACHI

R Developer and Instructor, the GRAPH Network



### ADMIN TEAM

GRAPH Courses Administration Team  
The GRAPH Courses team is building epidemiological training courses to enhance disease surveillance and data science for public health across the globe

---

## Références

Le contenu de ce cours a été adapté en partie des sources suivantes :

- Ismay, Chester, and Albert Y. Kim. 2022. *A ModernDive into R and the Tidyverse*. <https://moderndive.com/>.
- Kabacoff, Rob. 2020. *Data Visualization with R*. <https://rkabacoff.github.io/datavis/>.
- [https://www.rebeccabarter.com/blog/2017-11-17-ggplot2\\_tutorial/](https://www.rebeccabarter.com/blog/2017-11-17-ggplot2_tutorial/)

This work is licensed under the [Creative Commons Attribution Share Alike](#) license.



# Les histogrammes avec {ggplot2}

March 2024



Objectifs d'apprentissage . . . . .
Introduction . . . . .
Les packages . . . . .
Maladies diarrhéiques chez l'enfant au Mali . . . . .
Histogramme basique avec <code>geom_histogram()</code> . . . . .
Ajuster les classes dans un histogramme . . . . .
Définir le nombre de classes avec <code>bins</code> . . . . .
Définir la largeur des classes avec <code>binwidth</code> . . . . .
Modifier les limites des classes avec <code>breaks</code> . . . . .
En résumé . . . . .
Contributeurs . . . . .
Références . . . . .

---

## Objectifs d'apprentissage

À la fin de cette leçon, vous serez capable de :

1. Tracer un histogramme pour visualiser la distribution de variables continues en utilisant `geom_histogram()`.
  2. Ajuster le nombre ou la largeur des classes (`bins`) d'un histogramme à l'aide des paramètres `bins` OU `binwidth`.
  3. Décaler et aligner les classes d'un histogramme avec l'argument `boundary`.
  4. Définir les limites des classes avec l'argument `breaks`.
- 

## Introduction

Un histogramme est un graphique qui visualise la *distribution* d'une variable numérique comme suit :

1. Nous divisons d'abord l'axe des abscisses en une série de *classes*, chaque classe représentant un intervalle de valeurs.
2. Pour chaque classe, nous comptons le nombre d'observations qui se situent dans son intervalle.
3. Puis pour chaque classe, nous traçons une barre dont la hauteur indique le nombre correspondant d'observations.

## Les packages

```
pacman::p_load(tidyverse,  
                here)
```

## Maladies diarrhéiques chez l'enfant au Mali

Nous allons visualiser la distribution des variables numériques du dataframe `malidd` que nous avons vu dans les cours précédents.

```
# Importer les données du CSV  
malidd <- read_csv(here::here("data/clean/malidd.csv"))
```



Ces données ont été collectées dans le cadre d'une étude observationnelle sur la diarrhée aiguë chez les enfants âgés de 0 à 59 mois. L'étude a été menée au Mali début 2020. La base de données enregistre les informations démographiques et cliniques de 150 patients.

```
# Afficher les premières lignes du dataframe  
head(malidd)
```

Le dataframe contient 21 variables. Plusieurs d'entre elles sont continues, comme `height_cm` (taille en cm), `viral_load` (charge virale), et `freqrespi` (fréquence respiratoire).

## Histogramme basique avec `geom_histogram()`

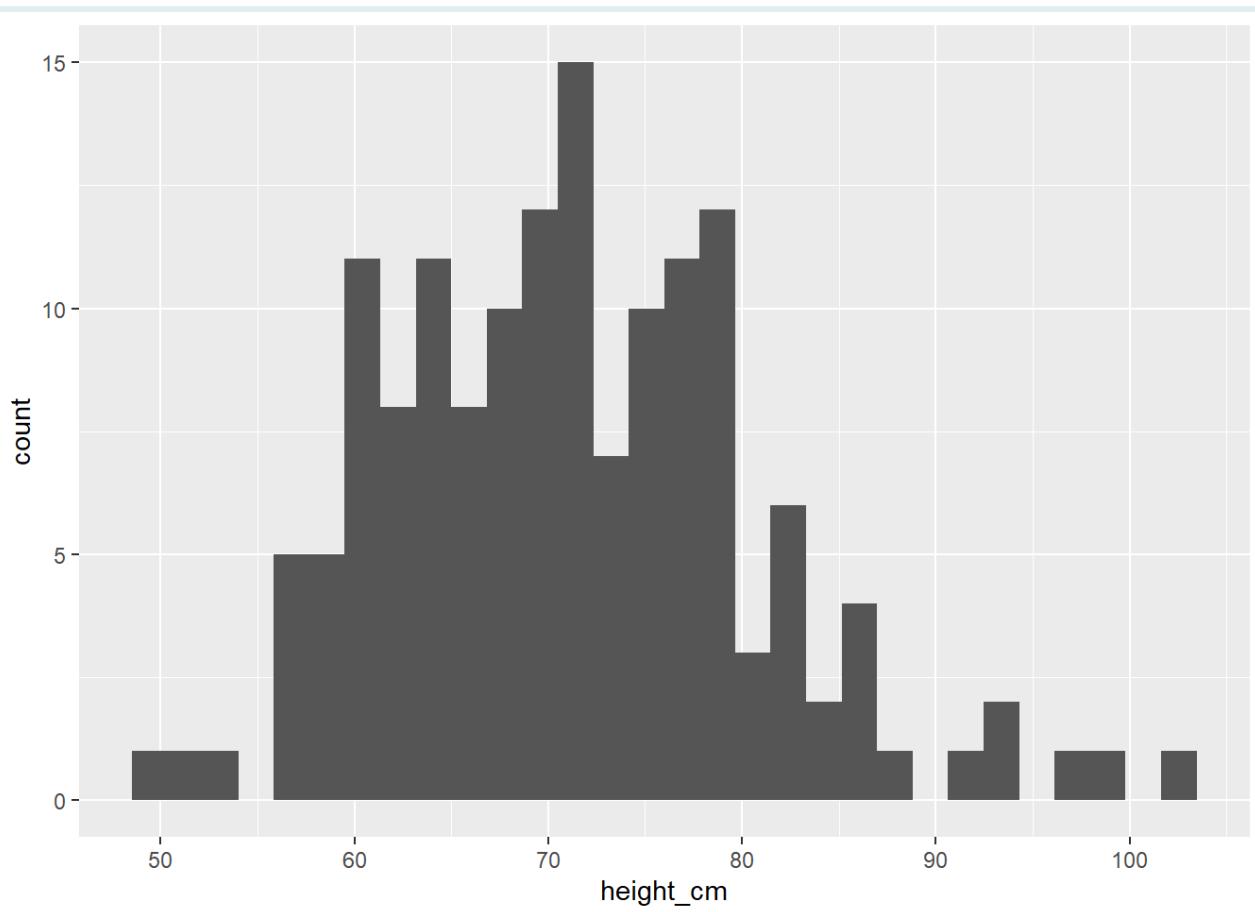
Nous allons utiliser `{ggplot2}` pour visualiser la distribution de la taille des enfant. Les données sont enregistrées dans la colonne `height_cm` de `malidd`.

La fonction `geom_*` utilisée pour les histogrammes est `geom_histogram()`

```
# Histogramme basique montrant la distribution de height_cm  
ggplot(data = malidd,
```

```
mapping = aes(x = height_cm)) + geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better  
## value with `binwidth`.
```



**SIDE NOTE**

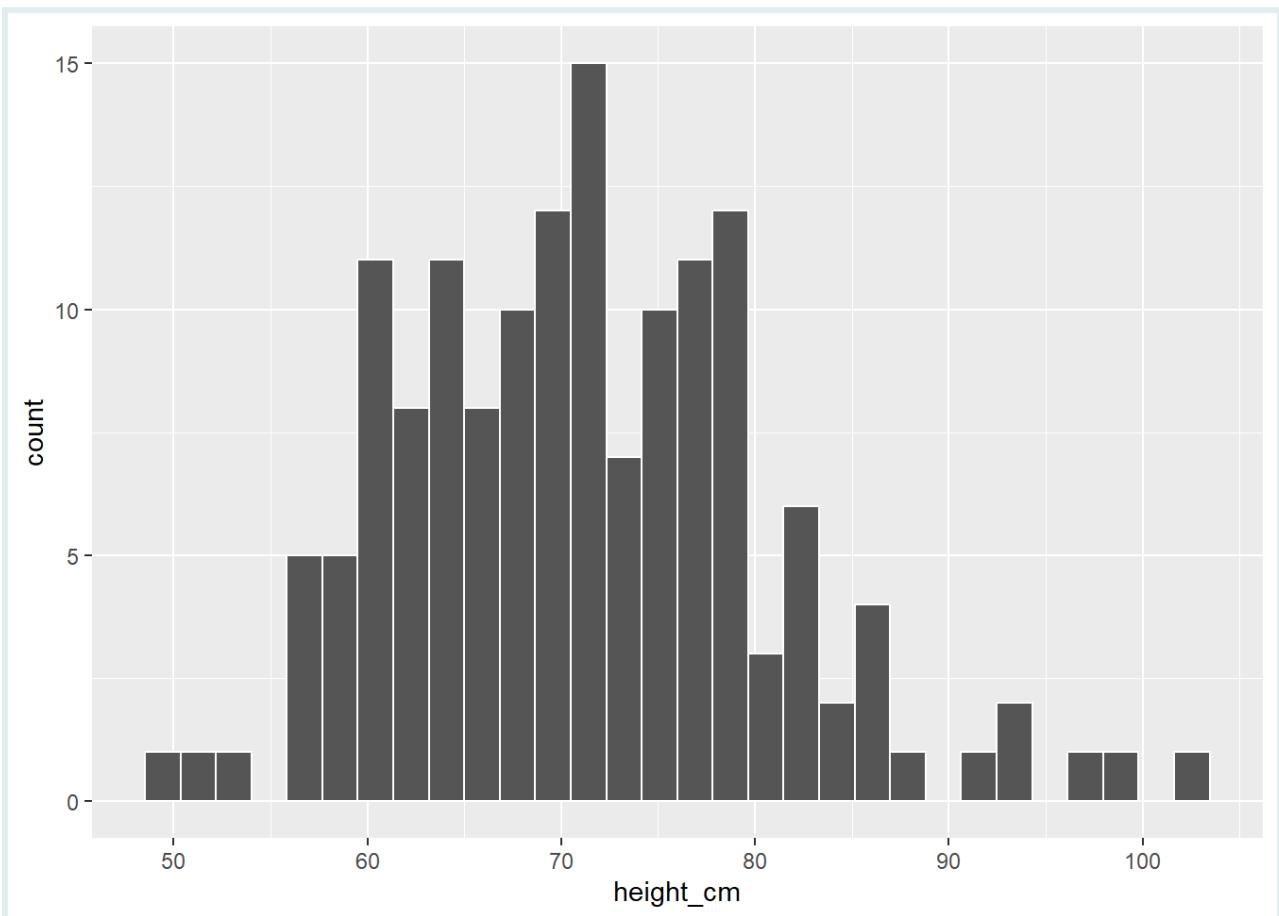


Si nous n'ajustons pas les classes dans `geom_histogram()`, nous obtenons un message d'avertissement. Vous pouvez ignorer ce message pour le moment. Vous apprendrez comment personnaliser les classes dans la prochaine section.

Dans l'histogramme précédent, il est difficile de visualiser les limites de chaque classe. Par conséquent, nous allons ajouter des bordures aux barres :

```
# Définir la couleur de la bordure à "white"  
ggplot(data = malidd ,  
        mapping = aes(x = height_cm)) +  
        geom_histogram(color = "white")
```

```
## `stat_bin()` using `bins = 30`. Pick better  
## value with `binwidth`.
```

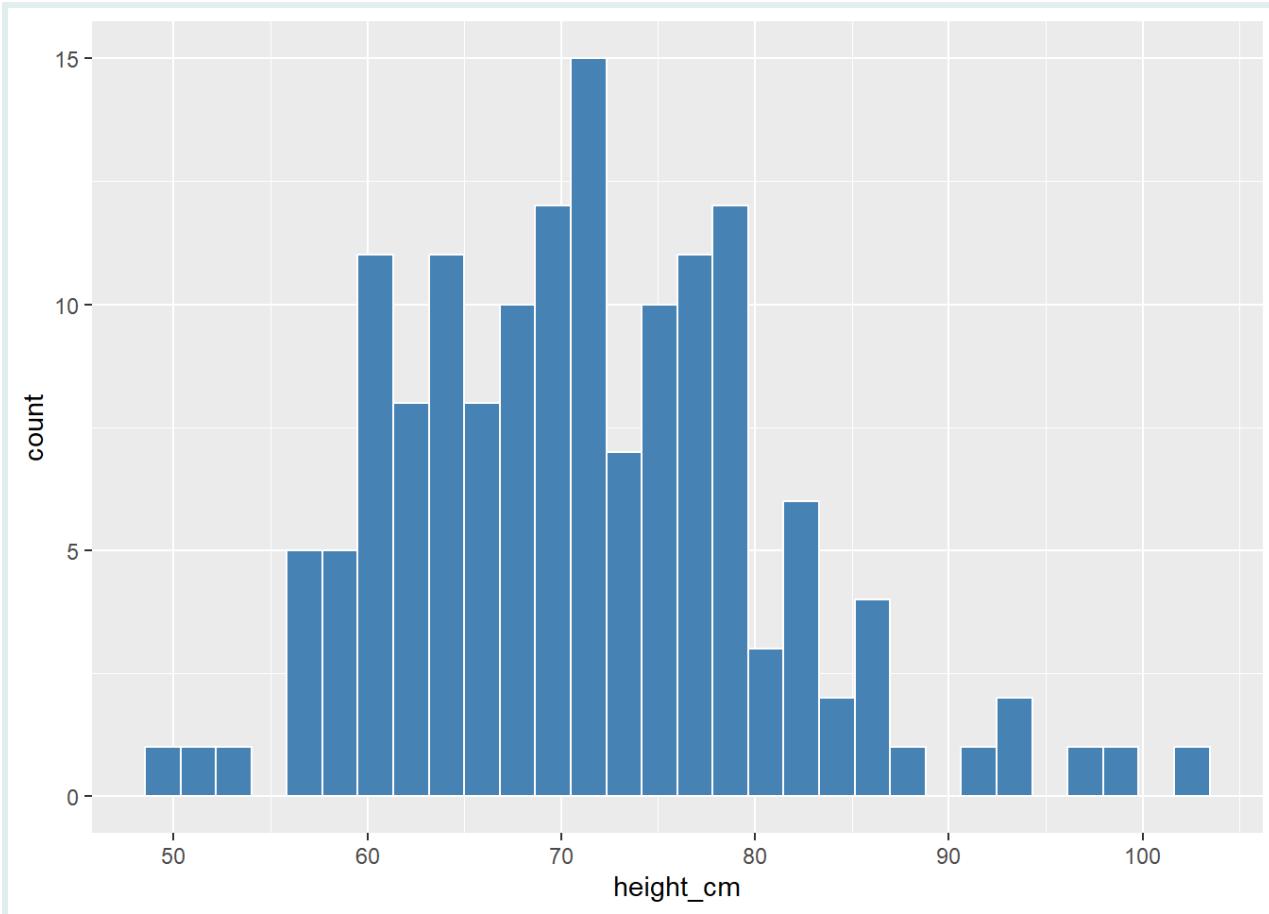


Il est maintenant plus facile d'associer chaque barre à la classe à laquelle elle correspond.

Nous pouvons également modifier la couleur des barres en ajoutant l'argument `fill`:

```
# Définir la couleur de remplissage à "steelblue"  
ggplot(data = malidd ,  
        mapping = aes(x = height_cm)) +  
geom_histogram(color = "white",  
               fill = "steelblue")
```

```
## `stat_bin()` using `bins = 30`. Pick better  
## value with `binwidth`.
```



Maintenant que nous sommes en mesure de distinguer les barres, analysons l'histogramme obtenu. Voici quelques questions auxquelles nous pourrions chercher à répondre :

1. Quelles sont les valeurs minimale et maximale ?
2. Quelle est la valeur “centrale” ou la valeur “la plus courante” ?
3. Comment sont réparties les valeurs ?
4. Quelles sont les valeurs les plus fréquentes et celles qui le sont moins ?

Nous pouvons voir que les tailles varient de 50 à 105 cm. Le centre se situe aux alentours de 70 cm, la majorité des patients ayant une taille comprise entre 60 et 80 cm, avec très peu de personnes mesurant moins de 55 cm ou plus de 90 cm. Il est à noter que l'histogramme présente une forme de cloche, ce qui suggère que la variable suit une *distribution normale* (à peu près).



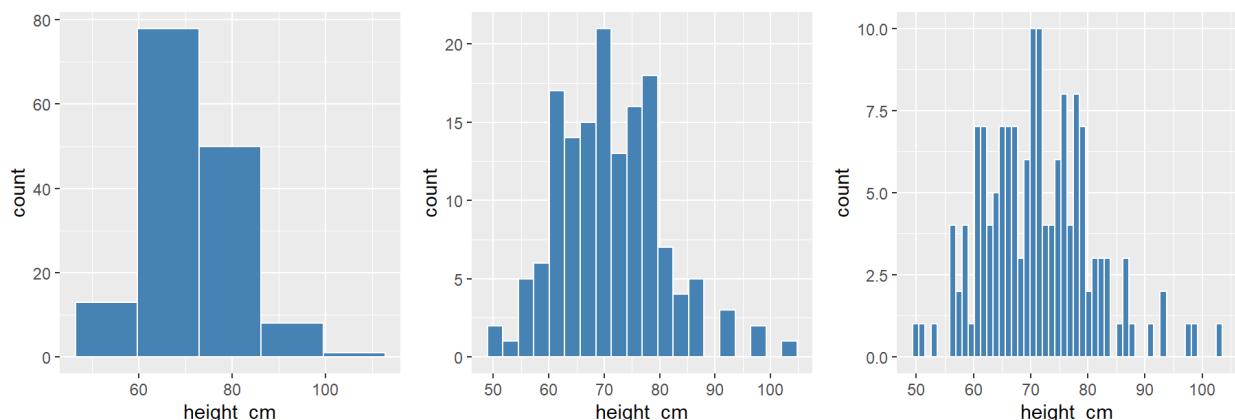
- Tracez un histogramme montrant la distribution de l'âge (`age_months`) dans `malidd`. Définissez les bordures et le



remplissage des barres à “seagreen”, et réduisez l’opacité à 40%.

- En vous basant sur le code précédent, modifiez les titres des axes pour “Âge (mois)” et “Nombre d’enfants”, respectivement.

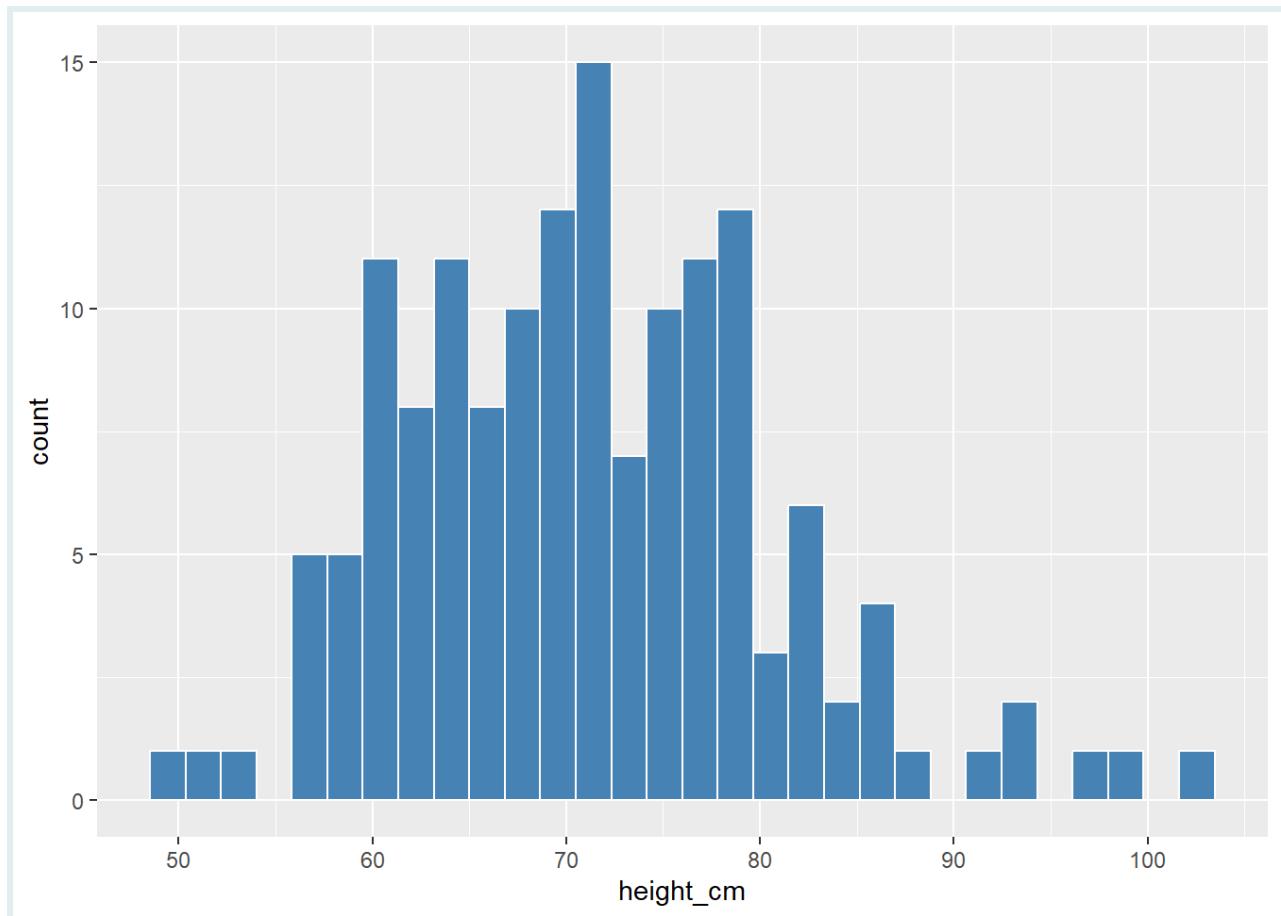
## Ajuster les classes dans un histogramme



Histogrammes représentant la même variable avec différents réglages des classes.

Lorsque nous avons exécuté le code dans les exemples précédents, nous avons obtenu un histogramme ainsi qu’un message d’avertissement relatif au nombre et à la largeur des classes. Ce message d’avertissement nous indique que `bins = 30`, ce qui correspond à 30 classes également espacées.

```
# Le message d'avertissement nous dit de changer la valeur par défaut de 30
# classes
ggplot(data = malidd,
        mapping = aes(x = height_cm)) +
  geom_histogram(color = "white",
                 fill = "steelblue")  
  
## `stat_bin()` using `bins = 30`. Pick better
## value with `binwidth`.
```



À moins que vous ne spécifiez explicitement le nombre de classes, R continuera à donner ce message.

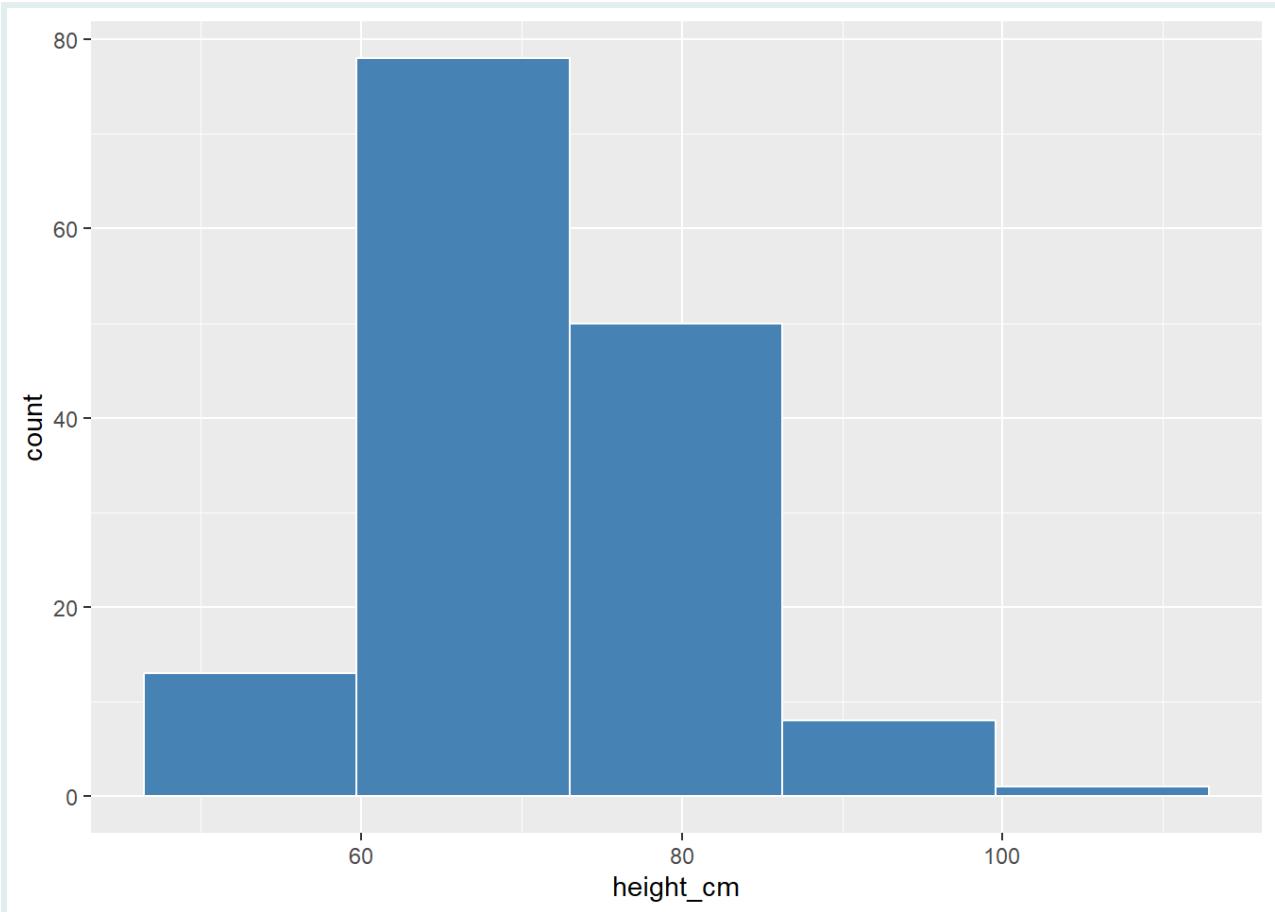
Nous pouvons modifier le nombre de classes en utilisant l'un de ces trois arguments de `geom_histogram()` :

1. Définir le nombre de classes avec `bins`
2. Définir la largeur des classes avec `binwidth`
3. Définir les limites des classes avec `breaks`

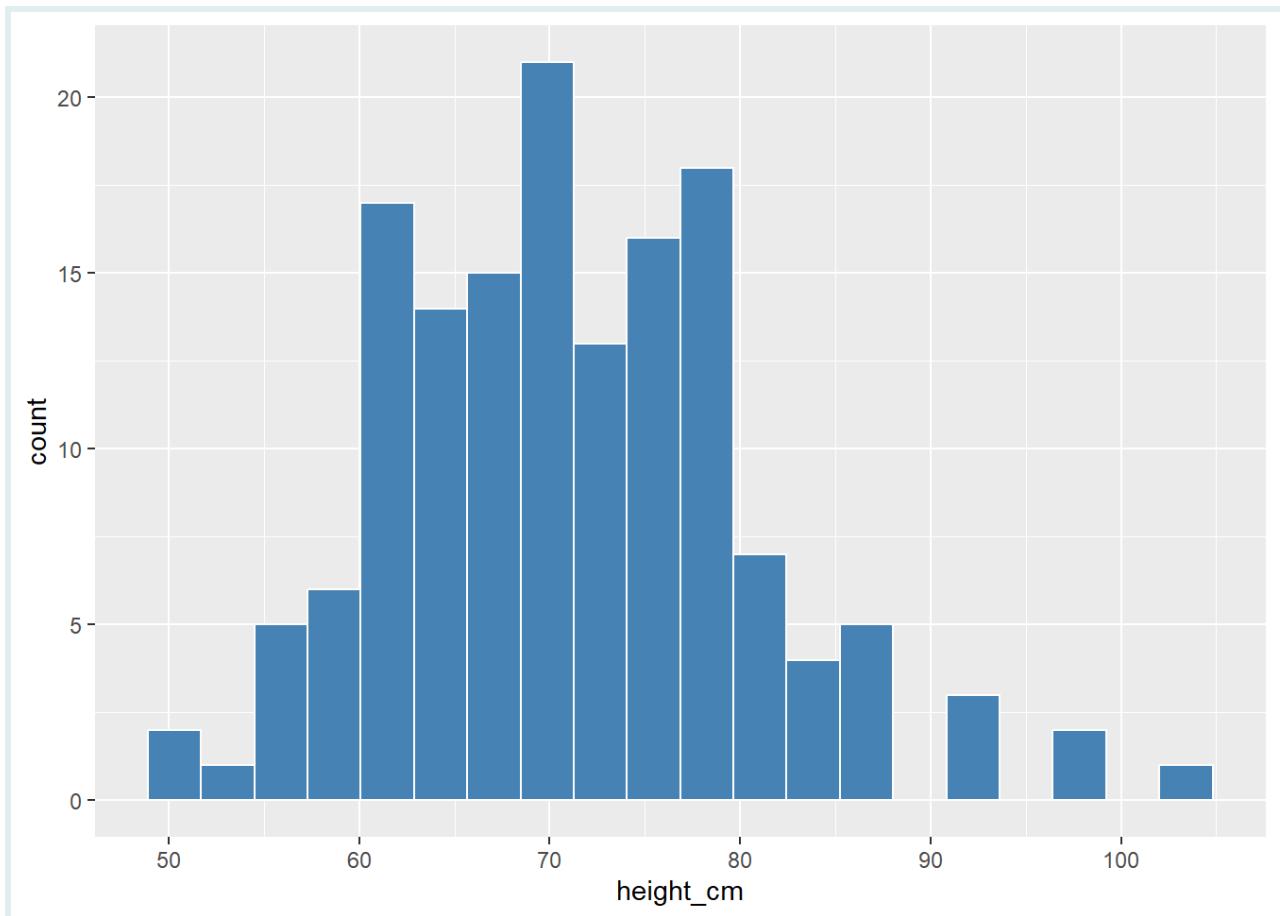
### Définir le nombre de classes avec `bins`

Avec la première méthode, nous pouvons spécifier le nombre de classes sur l'axe des x en réglant `bins = ENTIER`:

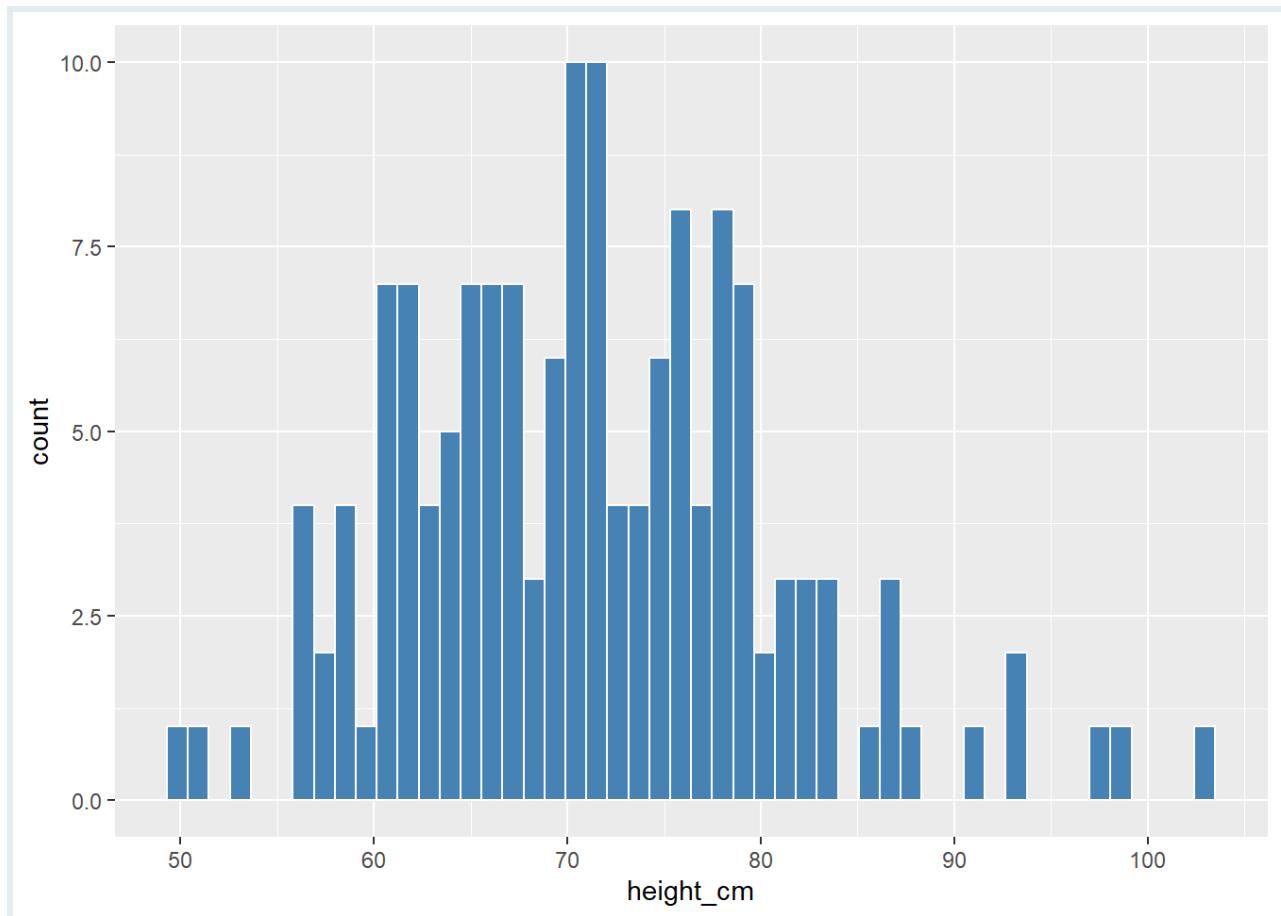
```
# Essayer différents nombres de classes
ggplot(data = malidd,
        mapping = aes(x = height_cm)) +
  geom_histogram(bins = 5,
                 color = "white",
                 fill = "steelblue")
```



```
ggplot(data = malidd,  
       mapping = aes(x = height_cm)) +  
  geom_histogram(bins = 20,  
                 color = "white",  
                 fill = "steelblue")
```



```
ggplot(data = malidd,
       mapping = aes(x = height_cm)) +
  geom_histogram(bins = 50,
                 color = "white",
                 fill = "steelblue")
```



Traquez un histogramme de la fréquence respiratoire (`freqrespi`), qui est mesurée en respirations par minute. Réglez la couleur de remplissage à “`indianred3`” et la couleur de la bordure à “`lightgray`”.

#### PRACTICE



Notez qu’avec les 30 classes par défaut, il existe certains intervalles pour lesquels aucune barre n’est tracée (c’est-à-dire qu’il n’y avait aucune observation dans cet intervalle).

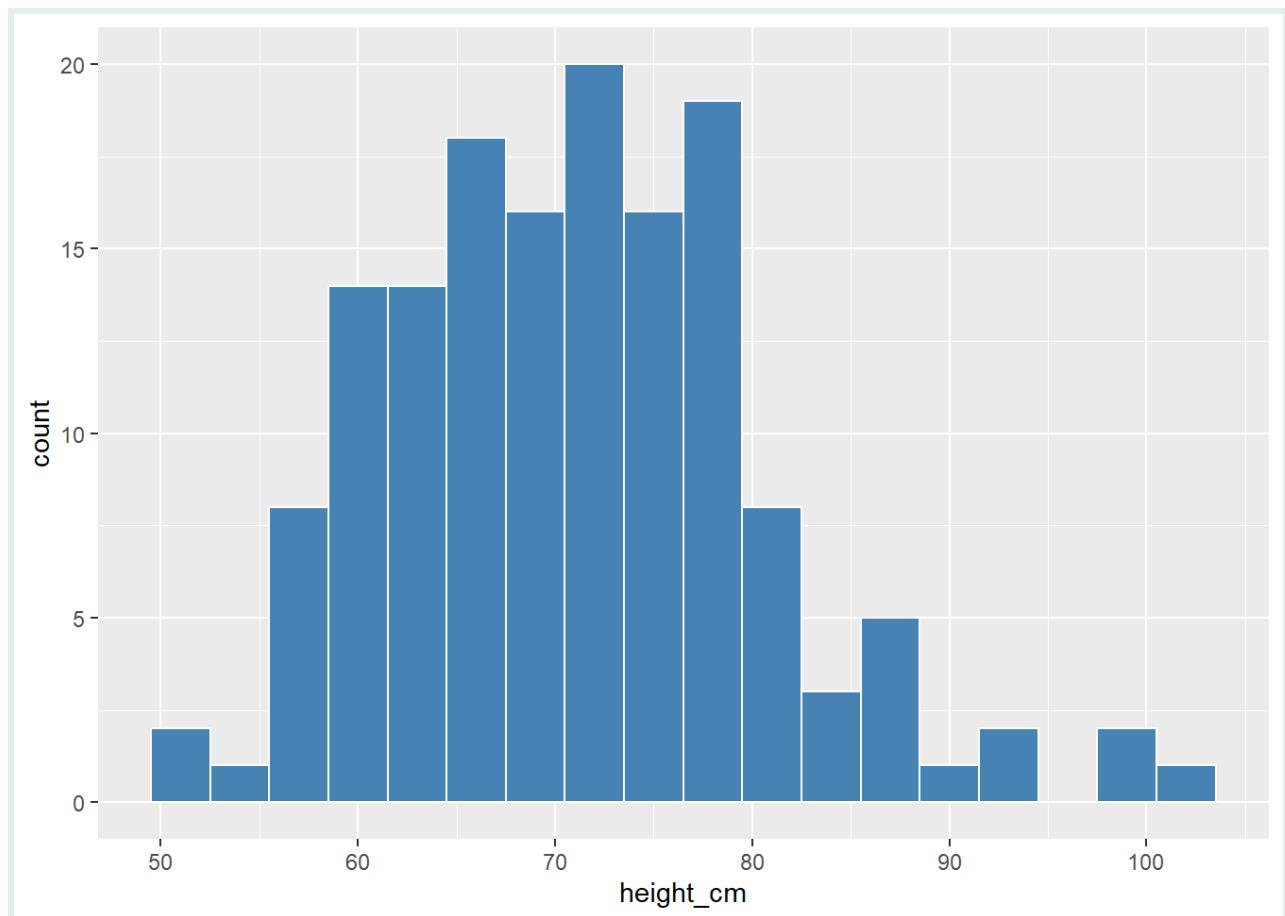
Diminuez le nombre de classes jusqu’à ce qu’il n’y ait plus d’intervalles vides. Sélectionnez la valeur la plus élevée pour laquelle il n’y a pas d’espaces vides.

#### Définir la largeur des classes avec `binwidth`

Avec la deuxième méthode, au lieu de spécifier le nombre de classes, nous spécifions leur largeur en utilisant l’argument `binwidth` dans `geom_histogram()`.

```
# Essayer différentes largeurs de classes
ggplot(data = malidd,
```

```
mapping = aes(x = height_cm)) +  
geom_histogram(color = "white",  
               fill = "steelblue",  
               binwidth = 3)
```

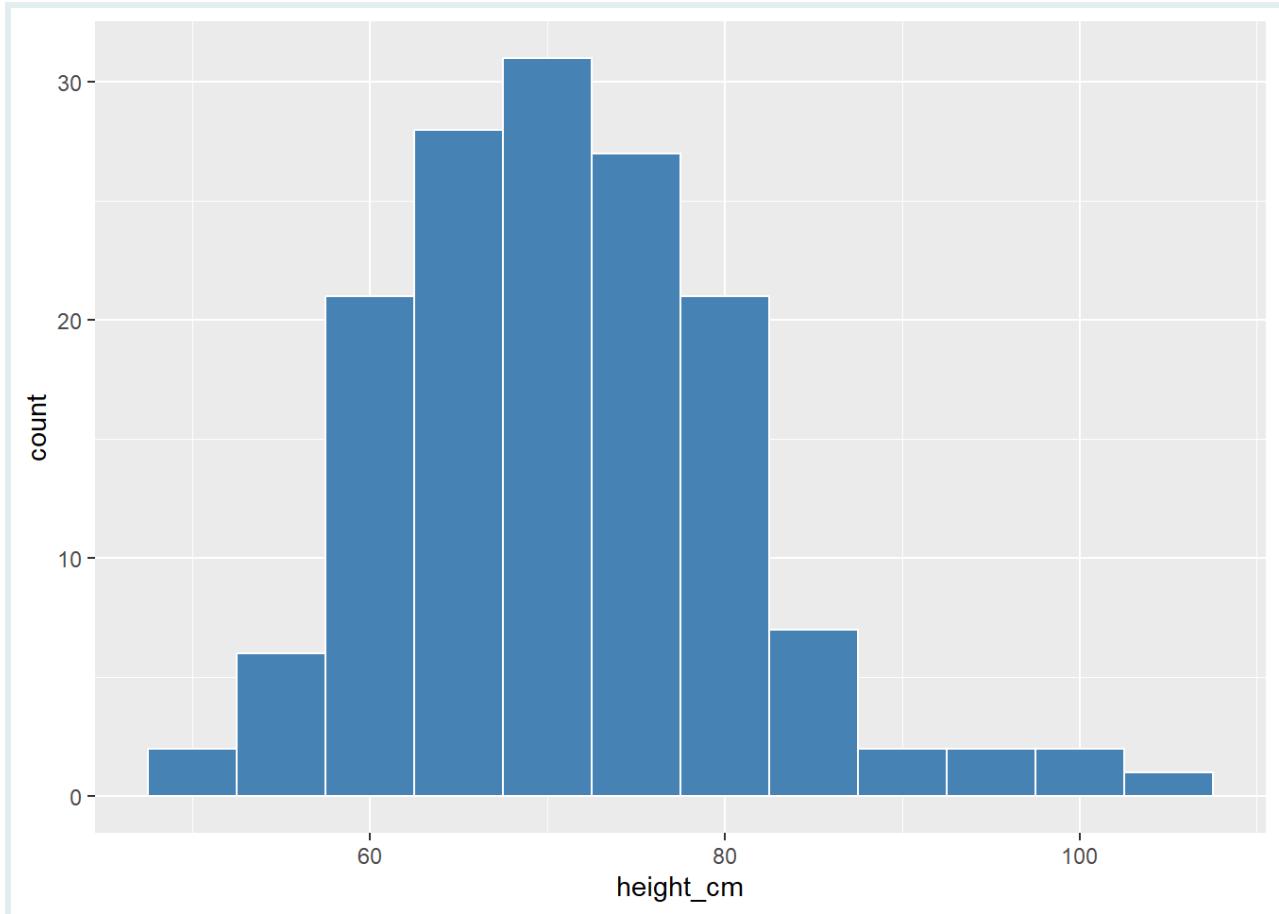


En examinant l'étendue des valeurs, nous pouvons choisir une largeur de classes appropriée.

```
range(malidd$height_cm)
```

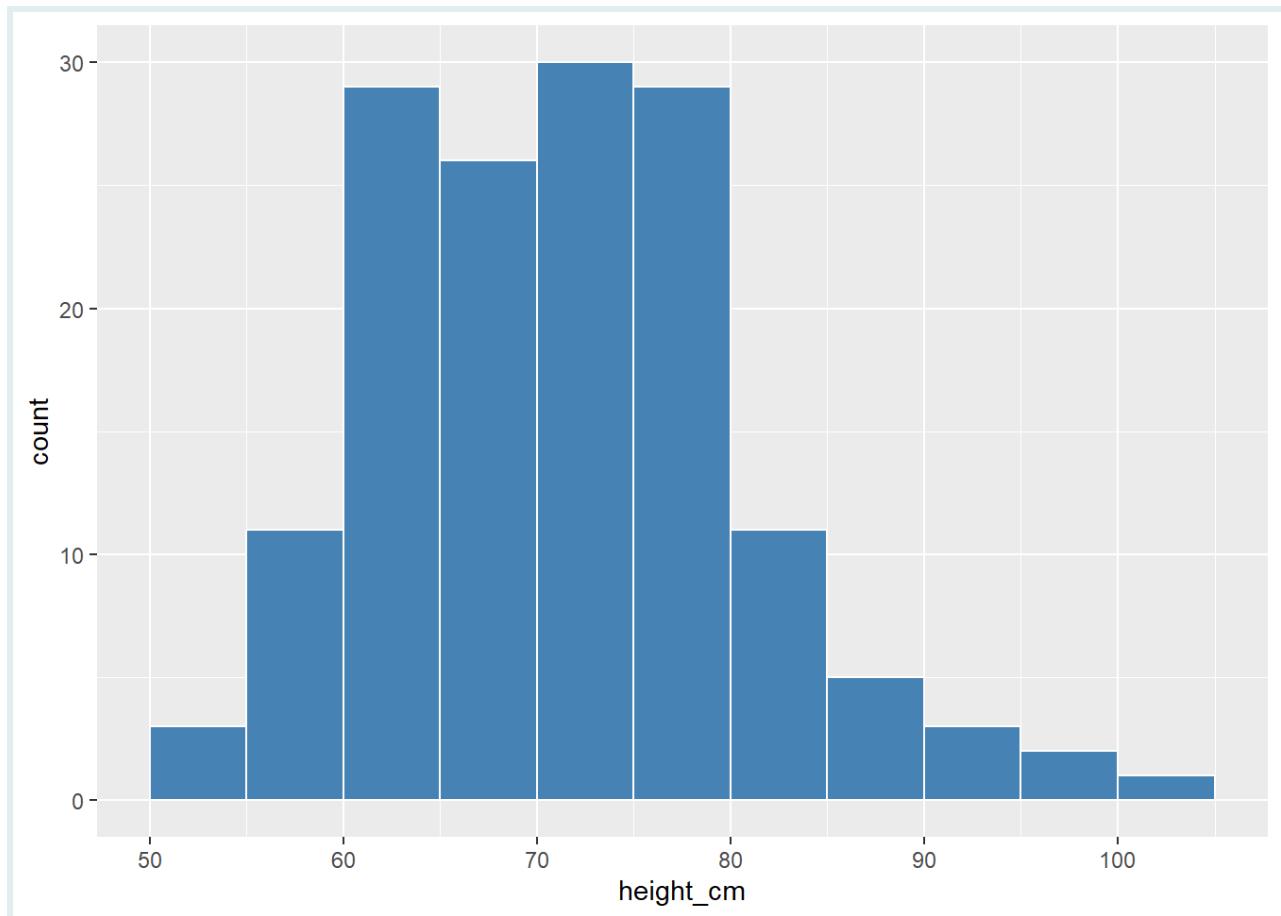
```
## [1] 50.3 103.4
```

```
ggplot(data = malidd,  
       mapping = aes(x = height_cm)) +  
geom_histogram(color = "white",  
               fill = "steelblue",  
               binwidth = 5)
```



Nous pouvons aussi utiliser l'argument `boundary` pour aligner les classes aux intervalles correspondantes sur l'axe des x.

```
# Mettre `boundary` égal à la valeur minimale de la variable
ggplot(data = malidd,
       mapping = aes(x = height_cm)) +
  geom_histogram(color = "white",
                 fill = "steelblue",
                 binwidth = 5,
                 boundary = 50)
```



### PRACTICE

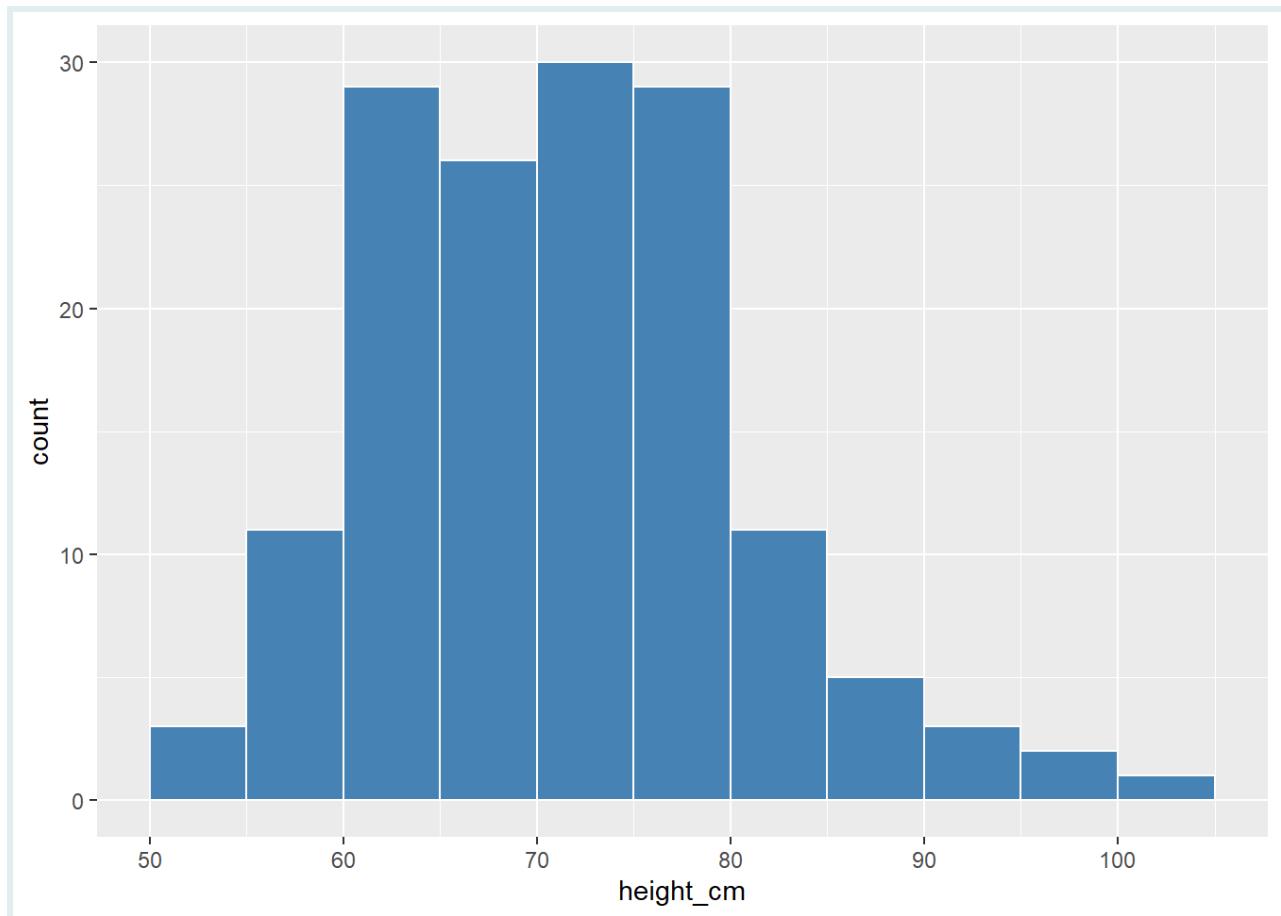


Créez le même histogramme de la variable `freqrespi` en définissant cette fois-ci la largeur des classes de manière à obtenir 18 classes. Ensuite, alignez les classes sur les valeurs de l'axe des x en ajustant les limites des classes.

### Modifier les limites des classes avec `breaks`

Pour modifier les limites des classes, définissez `breaks` à un vecteur numérique dans `geom_histogram()` :

```
# Fournir un vecteur qui couvre l'étendue des valeurs de height_cm
ggplot(data = malidd,
       mapping = aes(x = height_cm)) +
  geom_histogram(color = "white",
                 fill = "steelblue",
                 breaks = seq(50, 105, 5))
```



**PRACTICE**



(in RMD)

Tracez l'histogramme de `freqrespi` avec des limites allant de la valeur la plus basse de `freqrespi` à la plus élevée et des intervalles de 4.

Ensuite, ajustez l'échelle de l'axe des abscisses en ajoutant une fonction `scale_*`(). Configurez la plage valeurs de 24 à 60, avec des intervalles de 8.

## En résumé

Contrairement aux nuages de points et aux graphiques linéaires, les histogrammes représentent les informations d'une seule variable numérique. Plus précisément, ils permettent de visualiser la distribution de la variable numérique en question.

---

## Contributeurs

Les membres suivants ont contribué à ce cours :



### JOY VAZ

R Developer and Instructor, the GRAPH Network  
Loves doing science and teaching science

---



### IMANE BENSOUDA KORACHI

R Developer and Instructor, the GRAPH Network

---



### ADMIN TEAM

GRAPH Courses Administration Team  
The GRAPH Courses team is building epidemiological training courses to enhance disease surveillance and data science for public health across the globe

---

---

## Références

Le contenu de ce cours a été en partie adapté des sources suivantes :

- Ismay, Chester, and Albert Y. Kim. 2022. *A ModernDive into R and the Tidyverse*. <https://moderndive.com/>.
- Chang, Winston. 2013. *R Graphics Cookbook: Practical Recipes for Visualizing Data*. 1st edition. Beijing Köln: O'Reilly Media.

This work is licensed under the [Creative Commons Attribution Share Alike](#) license.



March 2024



Les boîtes à moustaches avec {ggplot2} .....
Objectifs d'apprentissage .....
Introduction .....
Charger les packages .....
Le dataset <code>gapminder</code> .....
Boîte à moustaches avec <code>geom_boxplot()</code> .....
Réorganiser les boîtes avec <code>reorder()</code> .....
Ajouter des points de données avec <code>geom_jitter()</code> .....
En résumé .....
Les acquis .....
Contributeurs .....
Références .....

---

## Les boîtes à moustaches avec {ggplot2}

### Objectifs d'apprentissage

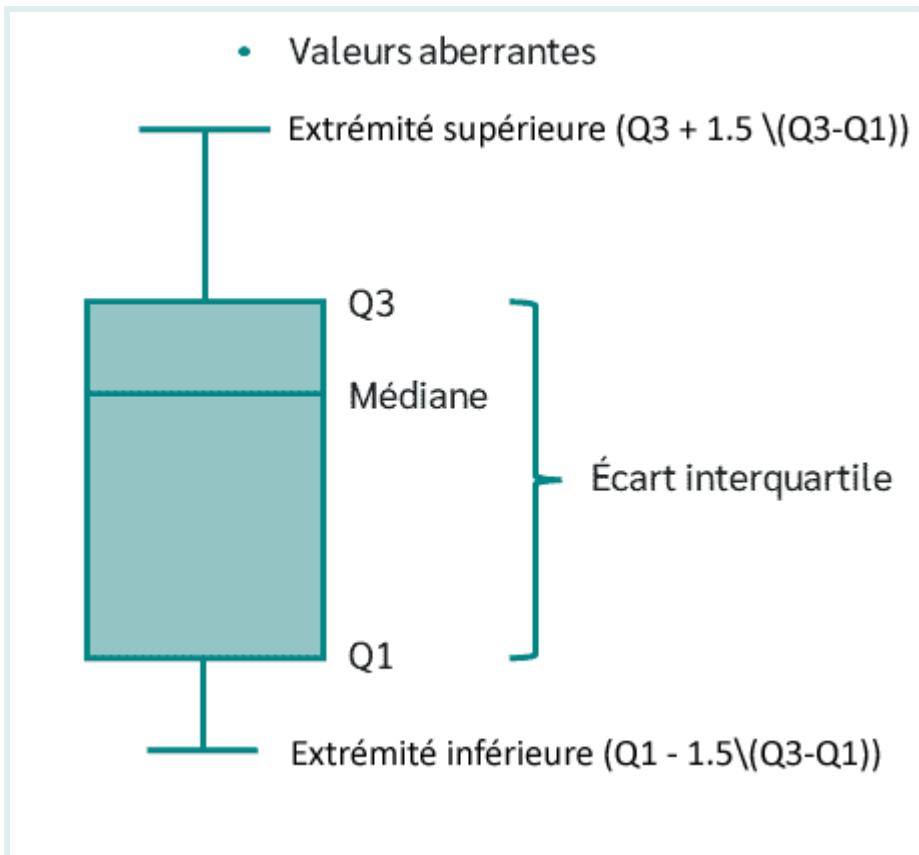
À la fin de ce cours, vous serez capable de :

1. Tracer une boîte à moustaches (boxplot) pour visualiser la distribution de données continues en utilisant `geom_boxplot()`.
2. Réorganiser les boîtes à moustaches avec la fonction `reorder()`.
3. Ajouter une couche de points de données sur une boîte à moustaches en utilisant `geom_jitter()`.

### Introduction

#### Structure d'une boîte à moustaches

La boîte à moustaches ou diagramme en boîte permet de visualiser la **distribution de variables numériques**.



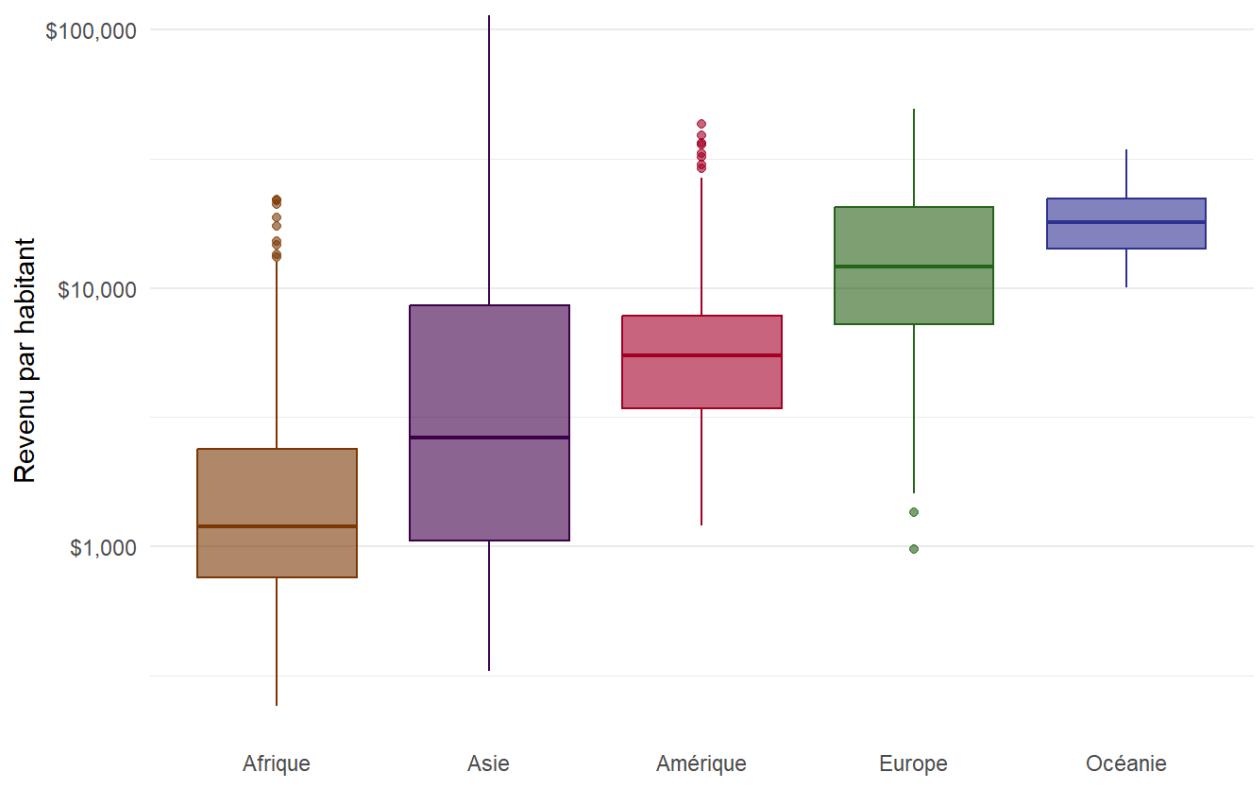
Elle se compose de deux éléments :

1. **Boîte** — S'étend du premier au troisième quartile (Q1 à Q3) avec une ligne au milieu qui représente la *médiane*. La plage de valeurs entre Q1 et Q3 est également connue sous le nom d'*écart interquartile*.
2. **Moustaches** — Des lignes partant des deux bords de la boîte indiquent la variabilité en dehors de Q1 et Q3. Les valeurs minimales/maximales des moustaches sont calculées selon la formule  $Q1 - 1.5 \setminus (Q3 - Q1)$  et  $Q3 + 1.5 \setminus (Q3 - Q1)$ . Tout ce qui se trouve au-delà est considéré comme une *valeur aberrante* et est représenté par des points ou d'autres symboles.

Les boîtes à moustaches sont généralement utilisées pour examiner la distribution d'une variable numérique pour chaque modalité d'une deuxième variable catégorielle.

### PIB par habitant groupé par continent

Données Gapminder de 142 pays (1952-2007)



Ici, nous comparons le PIB par habitant (variable continue) entre différentes régions du monde (variable catégorielle).

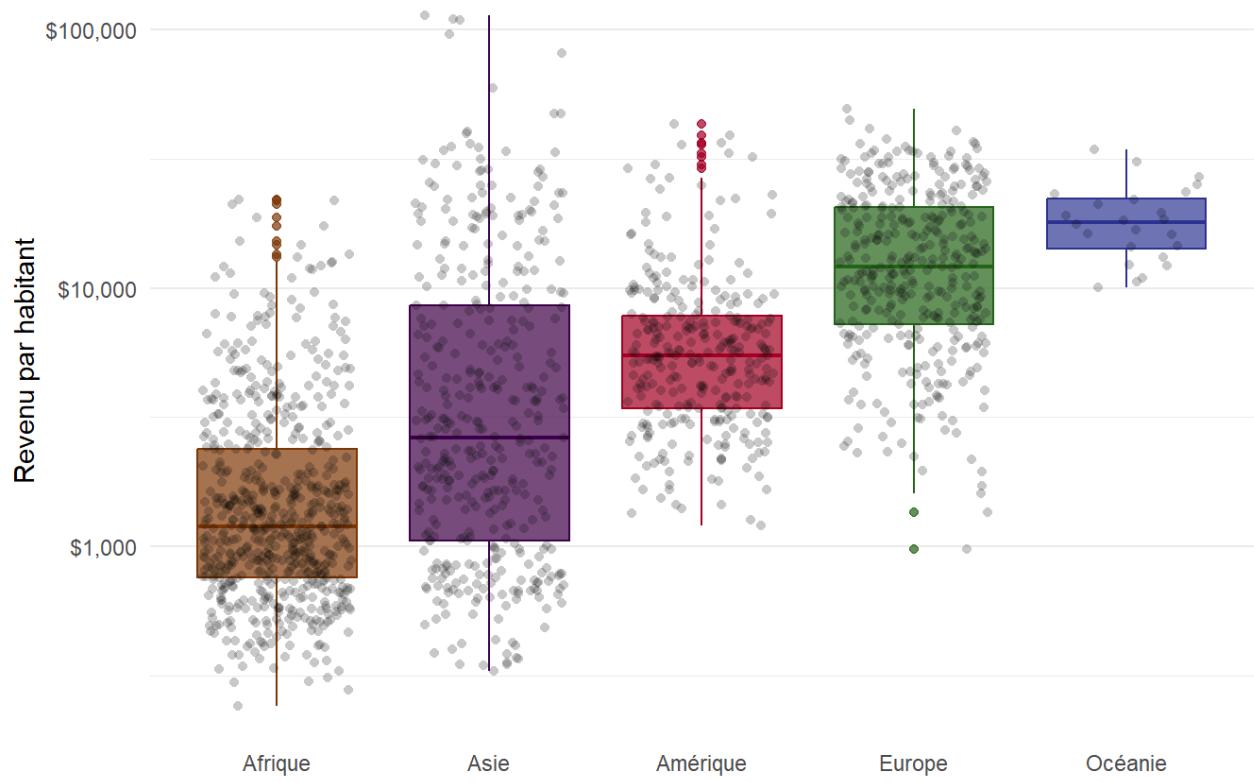
#### Pièges potentiels

Les boîtes à moustaches résument les données en cinq nombres, ce qui pourrait nous faire passer à côté d'informations importantes sur les données.

Si le volume de données que vous manipulez n'est pas trop grand, ajouter des points de données individuels peut rendre le graphique plus informatif.

## PIB par habitant groupé par continent

Données Gapminder de 142 pays (1952-2007)



## Charger les packages

```
pacman::p_load(tidyverse,  
                 gapminder,  
                 here)
```

## Le dataset gapminder

Pour ce cours, nous allons visualiser des données mondiales sur la santé et l'économie à partir du dataframe `gapminder` que nous avons déjà utilisé dans les cours précédents.

```
# Afficher les premières lignes du dataframe  
head(gapminder)
```

country	continent	year	lifeExp	pop	gdpPercap
Afghanistan	Asia	1952	28.801	8425333	779.4453...
Afghanistan	Asia	1957	30.332	9240934	820.8530...
Afghanistan	Asia	1962	31.997	10267083	853.10071
Afghanistan	Asia	1967	34.02	11537966	836.1971...
Afghanistan	Asia	1972	36.088	13079460	739.9811...
Afghanistan	Asia	1977	38.438	14880372	786.11336

Gapminder est un dataset pays-année. Il contient des informations sur 142 pays divisés en 5 “continents” ou régions du monde.



```
# Résumé des données
summary(gapminder)

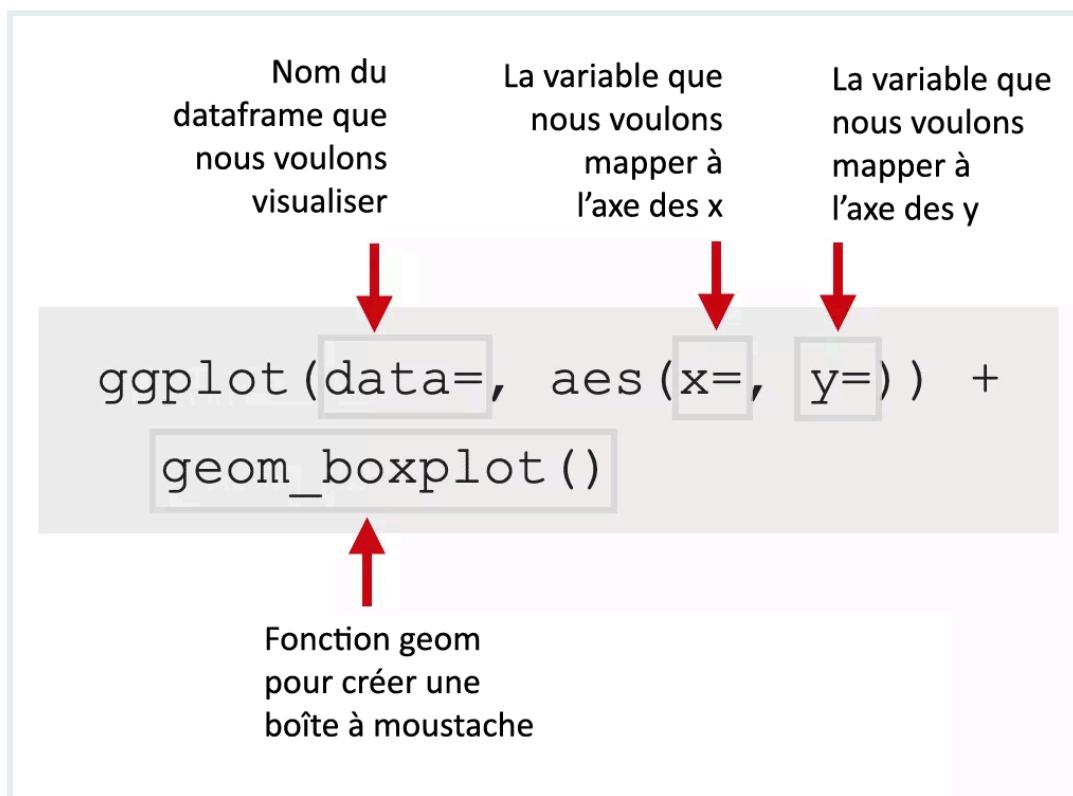
##          country      continent
## Afghanistan: 12    Africa   :624
## Albania     : 12    Americas:300
## Algeria     : 12    Asia     :396
## Angola      : 12    Europe   :360
## Argentina   : 12    Oceania  : 24
## Australia   : 12
## (Other)     :1632
##          year       lifeExp
## Min.   :1952   Min.   :23.60
## 1st Qu.:1966  1st Qu.:48.20
## Median  :1980  Median :60.71
## Mean    :1980  Mean   :59.47
## 3rd Qu.:1993  3rd Qu.:70.85
## Max.   :2007  Max.   :82.60
##
##          pop        gdpPercap
## Min.   :6.001e+04  Min.   : 241.2
## 1st Qu.:2.794e+06  1st Qu.: 1202.1
## Median :7.024e+06  Median : 3531.8
## Mean   :2.960e+07  Mean   : 7215.3
## 3rd Qu.:1.959e+07  3rd Qu.: 9325.5
## Max.   :1.319e+09  Max.   :113523.1
##
```



Les données sont enregistrées tous les 5 ans de 1952 à 2007 (soit un total de 12 années).

## Boîte à moustaches avec `geom_boxplot()`

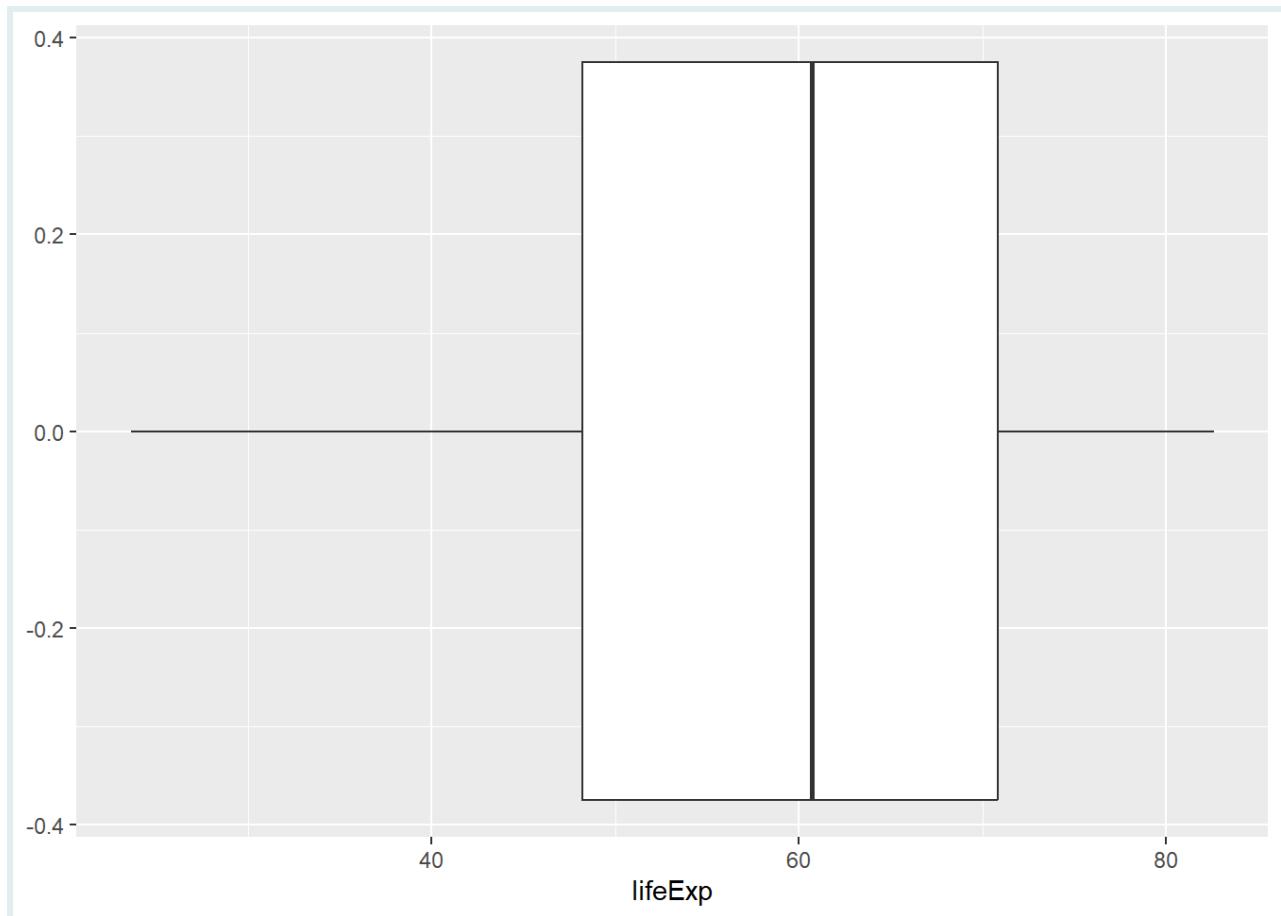
La fonction pour créer des boîtes à moustaches avec {ggplot2} est `geom_boxplot()`.



Nous allons commencer par tracer une boîte à moustaches basique, puis ajouter des esthétiques et couches supplémentaires.

Commençons par créer une boîte à moustaches simple en associant une variable numérique de `gapminder`, l'espérance de vie (`lifeExp`) à l'axe des `x`.

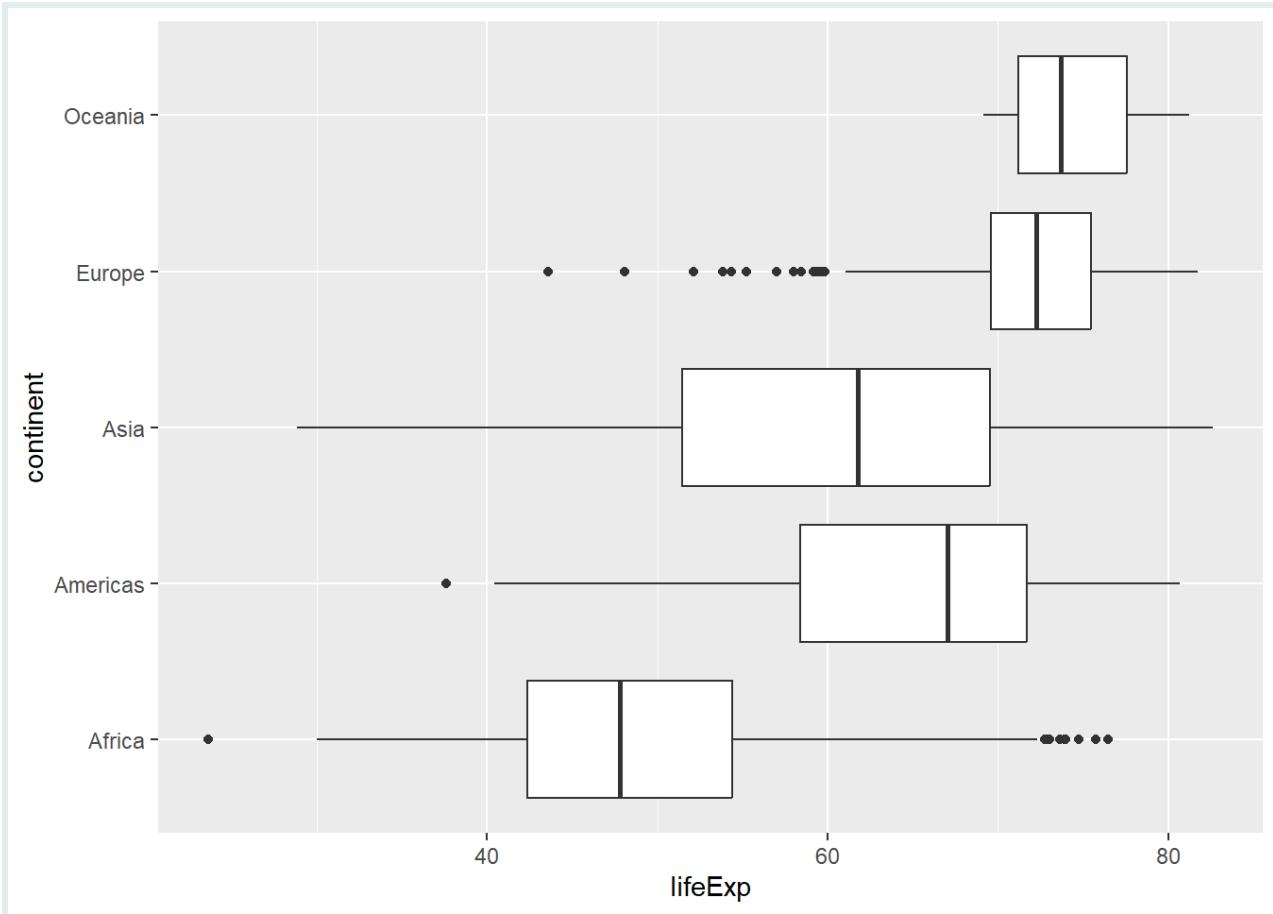
```
# Boîte à moustaches simple de lifeExp  
ggplot(data = gapminder,  
       mapping = aes(x = lifeExp)) +  
  geom_boxplot()
```



Ensuite, nous ajoutons une variable catégorielle à l'esthétique de position `y`.

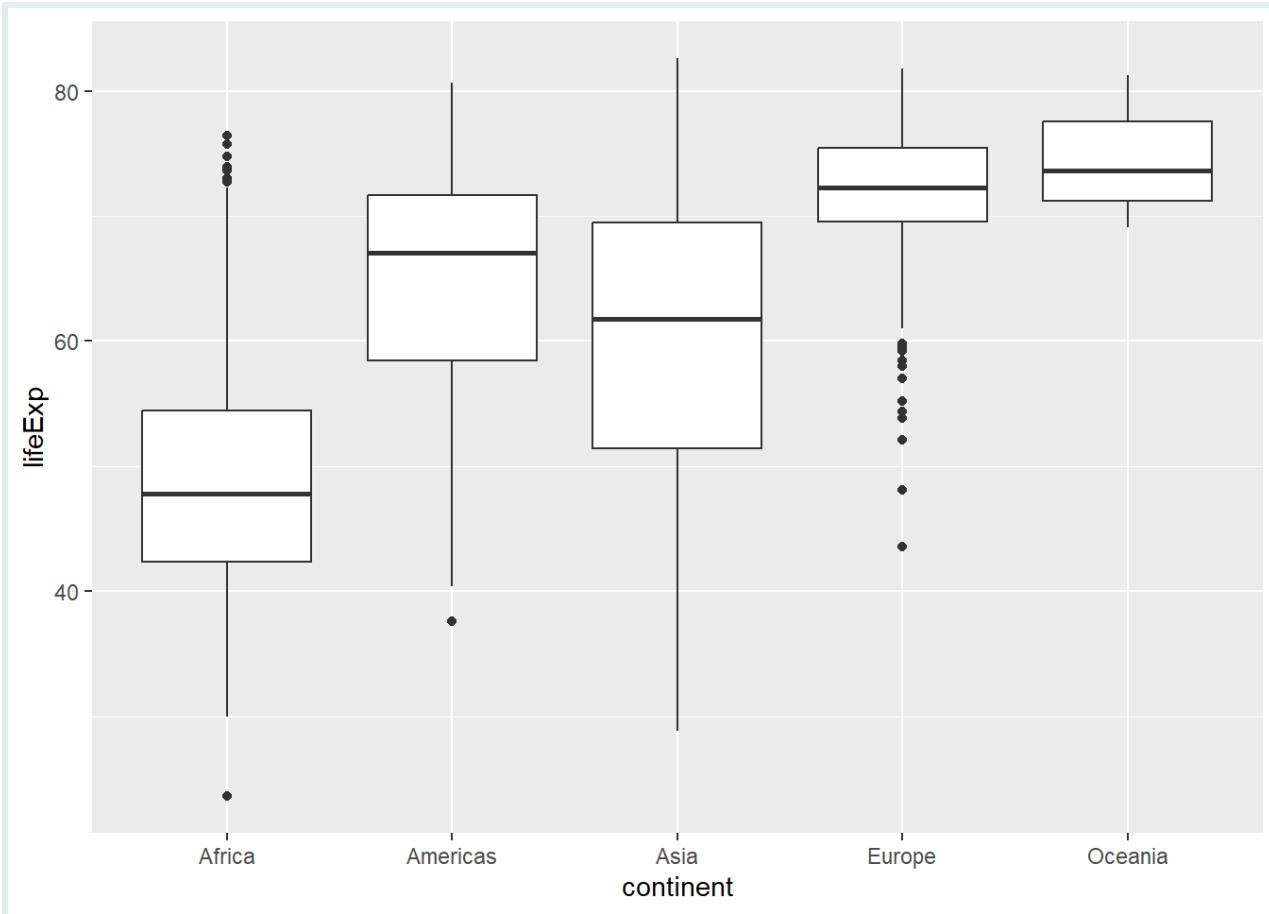
Comparons la distribution de l'espérance de vie entre les continents - c'est-à-dire, la distribution de `lifeExp` par modalité de la variable `continent`.

```
# Boîte à moustaches lifeExp par modalité de continent
ggplot(gapminder,
       aes(x = lifeExp,
            y = continent)) +
  geom_boxplot()
```



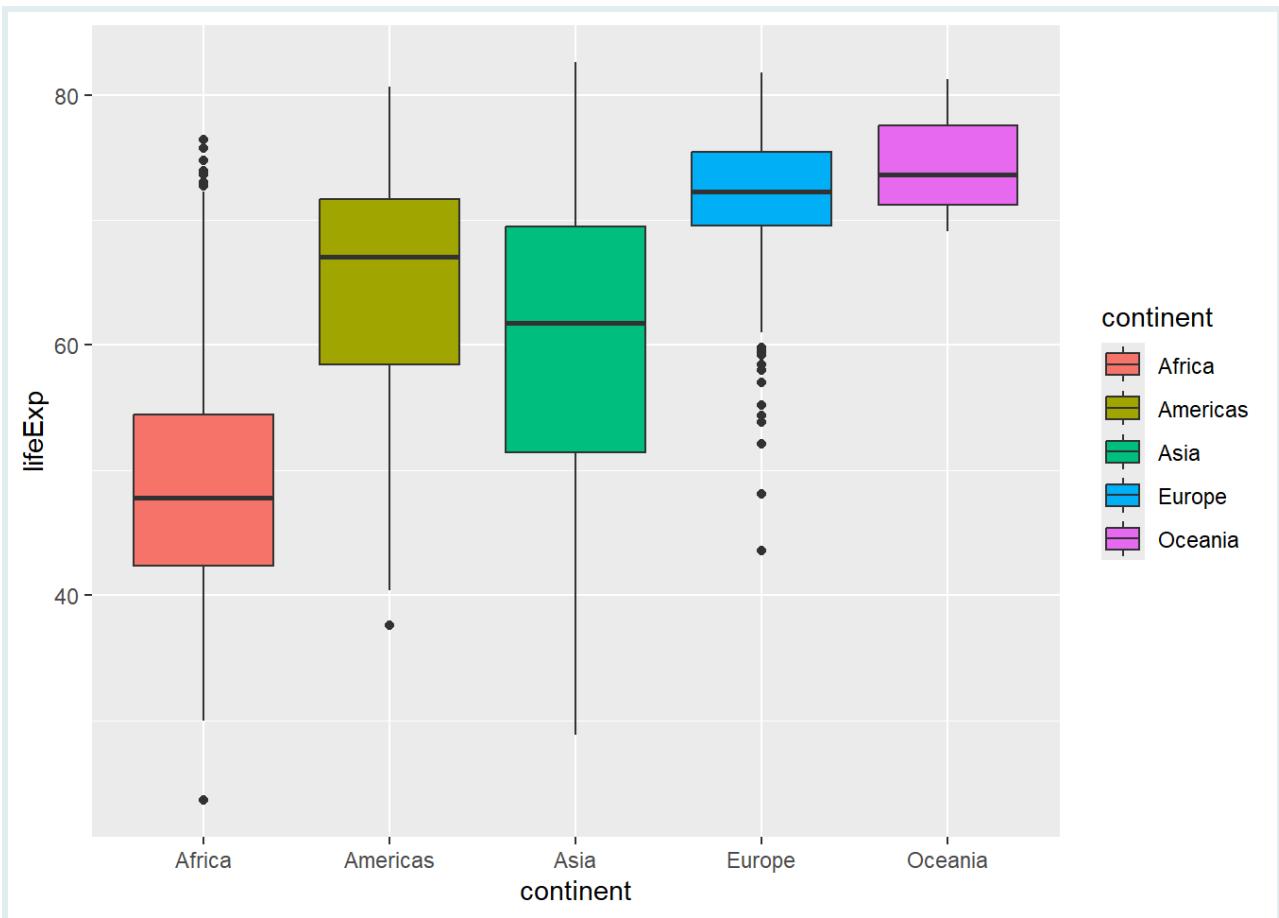
Le résultat est une boîte à moustaches basique de `lifeExp` par continent.

```
# Boîte à moustaches de lifeExp par continent (verticale)
ggplot(data = gapminder,
       mapping = aes(x = continent,
                      y = lifeExp)) +
  geom_boxplot()
```



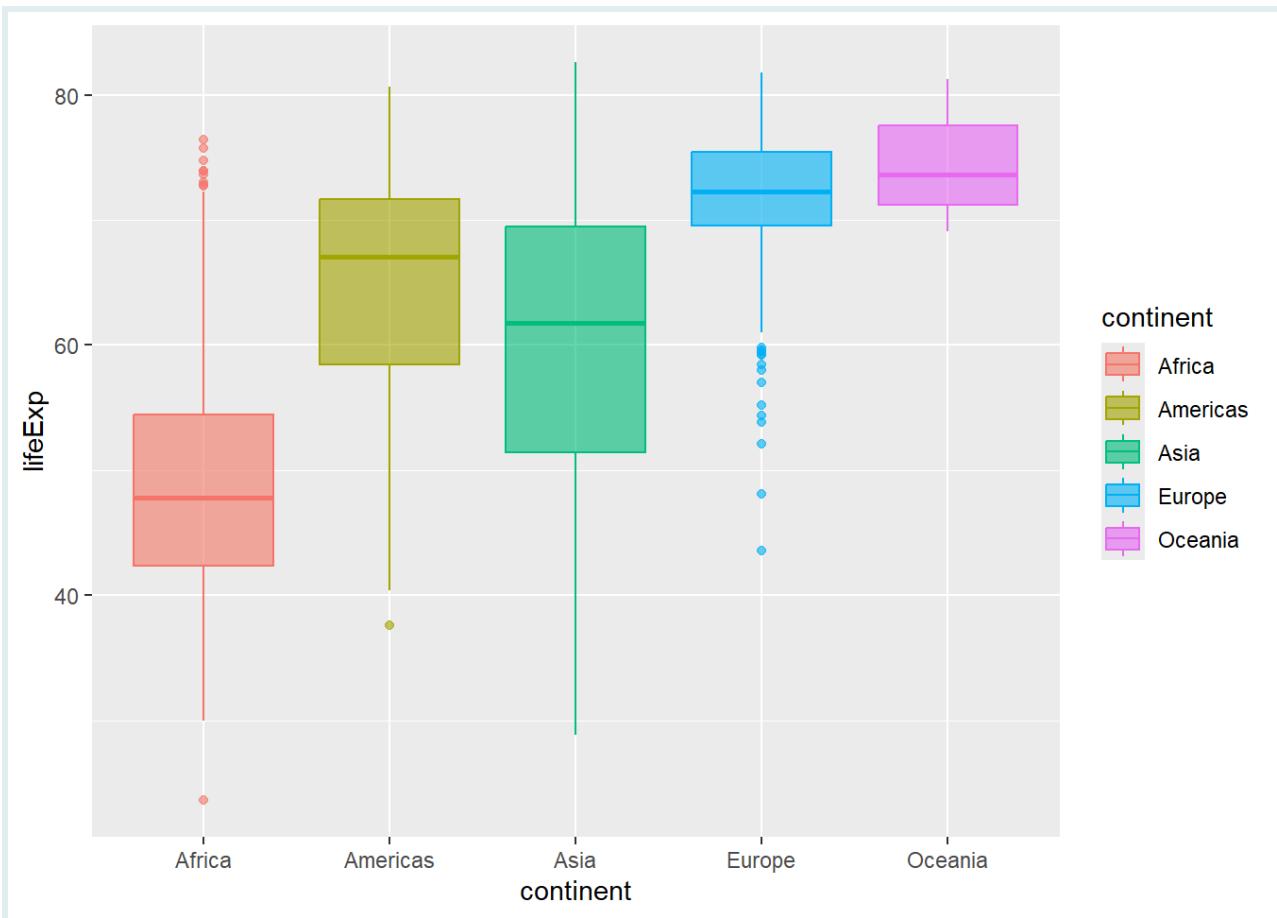
Ajoutons des couleurs aux boîtes. Nous pouvons associer la variable `continent` à `fill` pour que chaque boîte soit colorée en fonction du continent qu'elle représente.

```
# Attribuer une couleur différente à chaque continent avec fill
ggplot(gapminder,
       aes(x = continent,
           y = lifeExp,
           fill = continent)) +
  geom_boxplot()
```



Nous pouvons également ajouter les esthétiques `color` et `alpha` pour modifier la couleur du contour et la transparence.

```
# Changer la couleur du contour et augmenter la transparence
ggplot(gapminder,
       aes(x = continent,
            y = lifeExp,
            fill = continent,
            color = continent)) +
  geom_boxplot(alpha = 0.6)
```



- En utilisant le dataframe `gapminder`, créez une boîte à moustaches comparant la distribution du **PIB par habitant** (`gdpPerCap`) entre les continents. Associez la **couleur de remplissage** des boîtes à `continent`, et réglez l'**épaisseur de ligne** à 1.
- En vous basant sur le code de la question précédente, ajoutez une fonction `scale_*`() qui transforme l'axe des y en une échelle logarithmique.

## Réorganiser les boîtes avec `reorder()`



Réorganiser les boîtes à moustaches en fonction de l'espérance de vie plutôt que par ordre alphabétique.

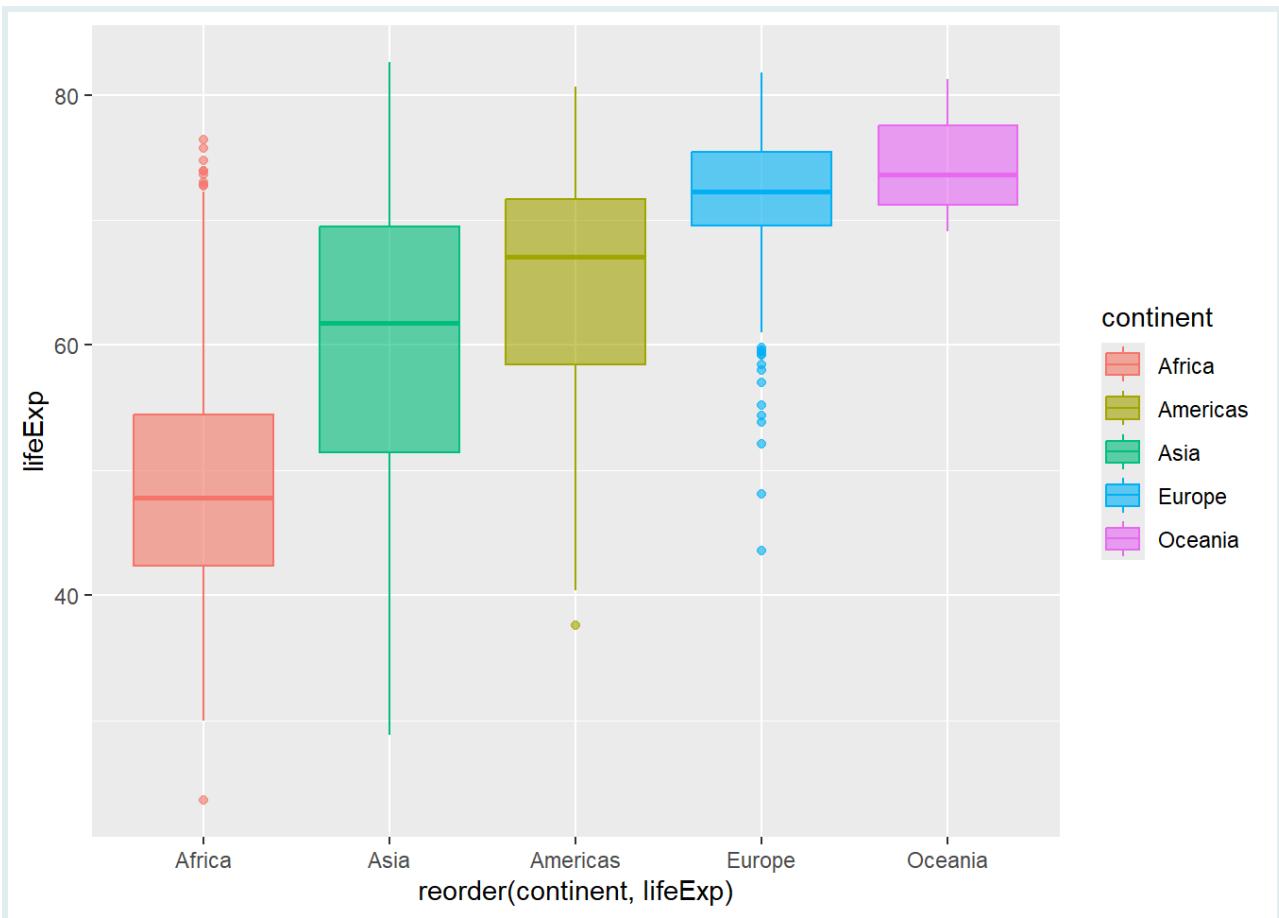
Les modalités de la variable `continent` sont ordonnées par défaut par ordre alphabétique. Si vous regardez l'axe des x, il commence avec l'Afrique et va alphabétiquement jusqu'à l'Océanie. Il serait plus intéressant de les ordonner selon l'espérance de vie, la variable de l'axe des y.

Nous pouvons changer l'ordre des niveaux d'un facteur dans R en utilisant la fonction `reorder()`. Si nous réorganisons les modalités de la variable `continent`, les boîtes seront tracées sur l'axe des x dans cet ordre. `reorder()` traite son premier argument comme une variable catégorielle, et réorganise ses niveaux en fonction des valeurs d'une seconde variable numérique.

Pour réorganiser les modalités de la variable `continent` en fonction de `lifeExp`, nous utiliserons la syntaxe `reorder(VAR_CATEGORIELLE, VAR_NUMERIQUE)` comme ceci : `reorder(continent, lifeExp)`.

Ici, nous allons modifier l'argument `x` et dire à `ggplot()` de réorganiser la variable.

```
ggplot(gapminder,
       aes(x = reorder(continent, lifeExp),
           y = lifeExp,
           fill = continent,
           color = continent)) +
  geom_boxplot(alpha = 0.6)
```



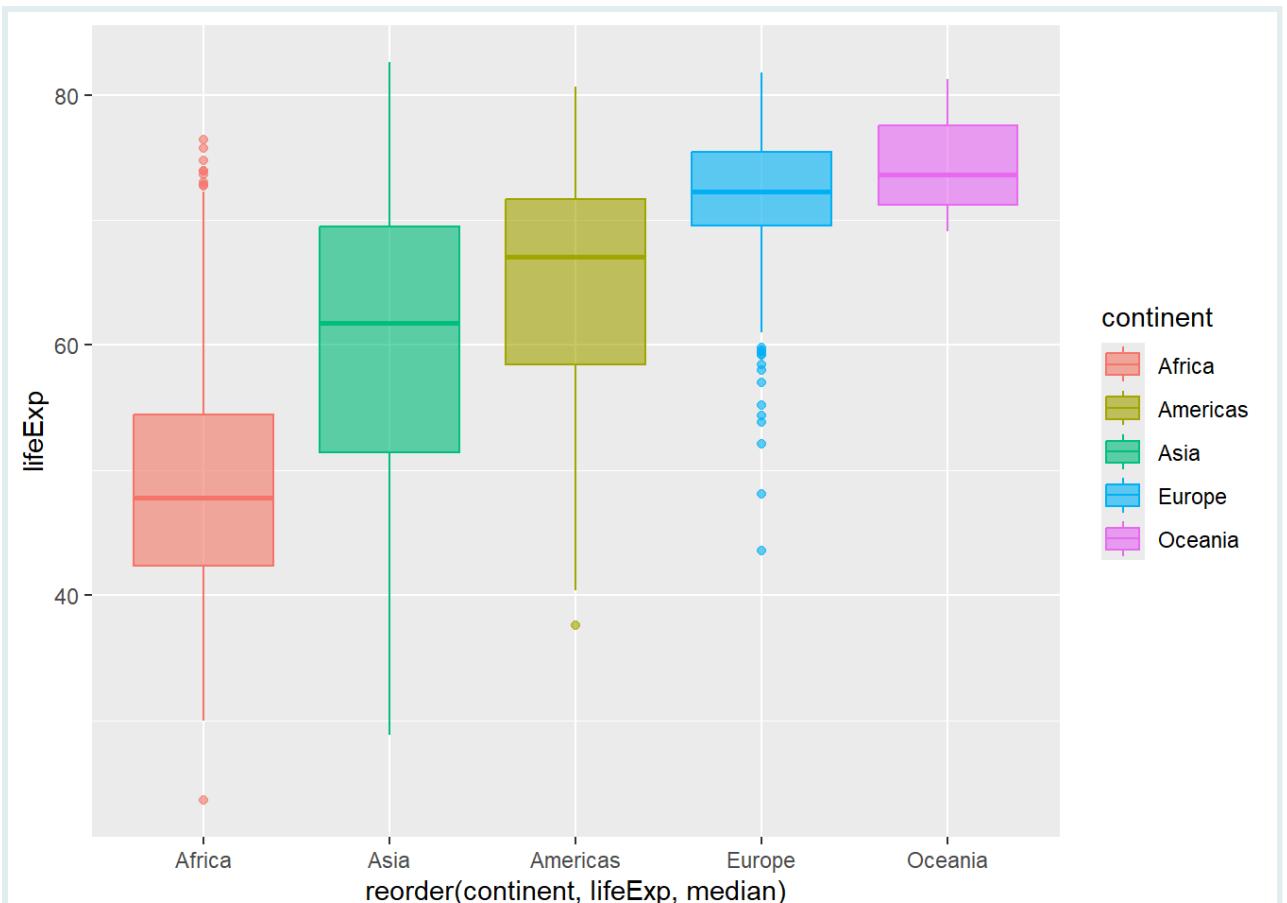
Nous pouvons voir qu'il y a des différences notables dans l'espérance de vie médiane entre les continents. Cependant, il y a beaucoup de chevauchement entre les plages de valeurs des continents. Par exemple, l'espérance de vie médiane du continent africain est inférieure à celle de l'Europe, mais plusieurs pays africains ont des valeurs d'espérance de vie supérieures à celles de la majorité des pays européens.

### Choisir la méthode de réorganisation

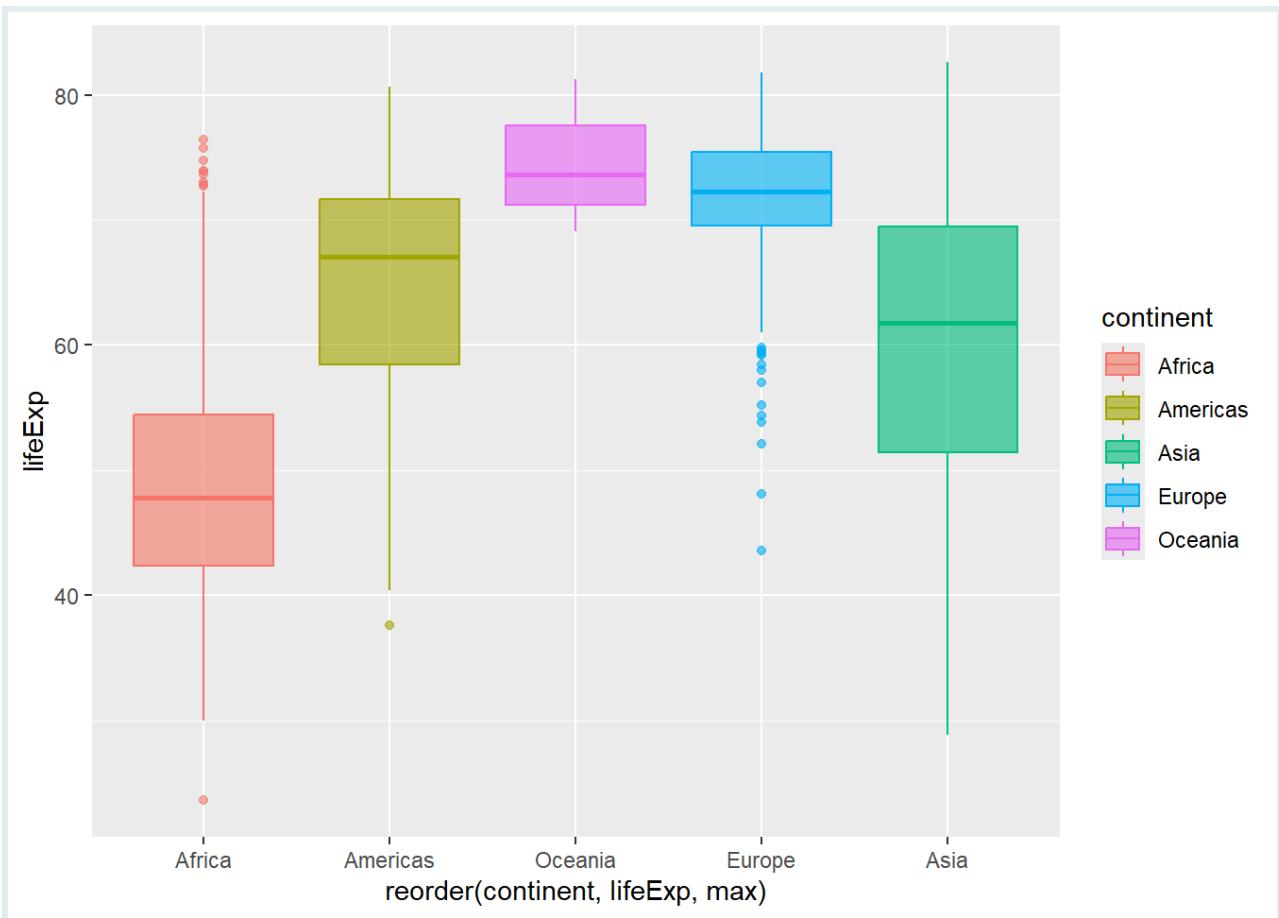
La méthode par défaut réorganise le facteur en fonction de la **moyenne** de la variable numérique.

Nous pouvons ajouter un troisième argument pour choisir une méthode différente, comme la **médiane** ou le **maximum**.

```
# Organiser les boîtes à moustaches par espérance de vie médiane
ggplot(gapminder,
       aes(x = reorder(continent, lifeExp, median),
            y = lifeExp,
            fill = continent,
            color = continent)) +
  geom_boxplot(alpha = 0.6)
```



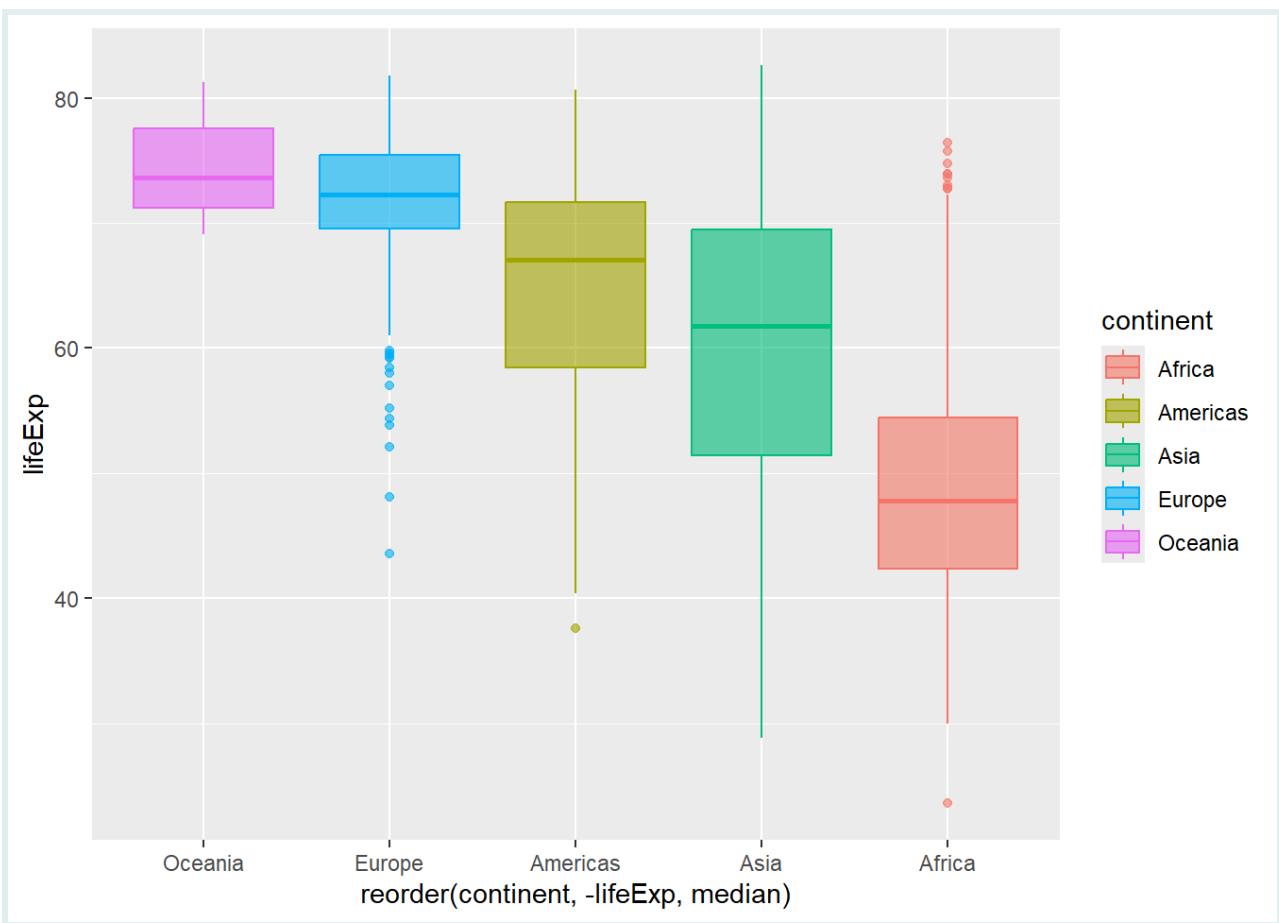
```
# Organiser les boîtes à moustaches par espérance de vie maximale
ggplot(gapminder,
       aes(x = reorder(continent, lifeExp, max),
            y = lifeExp,
            fill = continent,
            color = continent)) +
  geom_boxplot(alpha = 0.6)
```



Les boîtes à moustaches sont organisées dans un ordre **croissant**.

Pour trier les boîtes par ordre **décroissant**, nous ajoutons une **négation** à `lifeExp` dans la fonction `reorder()`.

```
# Organiser les boîtes à moustaches par espérance de vie médiane décroissante
ggplot(gapminder,
       aes(x = reorder(continent, -lifeExp, median),
            y = lifeExp,
            fill = continent,
            color = continent)) +
  geom_boxplot(alpha = 0.6)
```



Créez une boîte à moustaches montrant la distribution du PIB par habitant pour chaque continent, comme vous l'avez fait dans l'exercice 1. Conservez le remplissage, l'épaisseur de ligne et l'échelle de l'axe des y de ce graphique.

### PRACTICE



(in RMD)

Maintenant, **réorganisez** les boîtes en fonction de la **moyenne** de `gdpPercap` et par ordre **décroissant**.

En vous basant sur le code de la question précédente, ajoutez des **étiquettes** à votre graphique.

- Définissez le **titre principal** à “Variation du PIB par habitant à travers les continents (1952-2007)”
- Changez le **titre de l'axe des x** pour “Continent”,



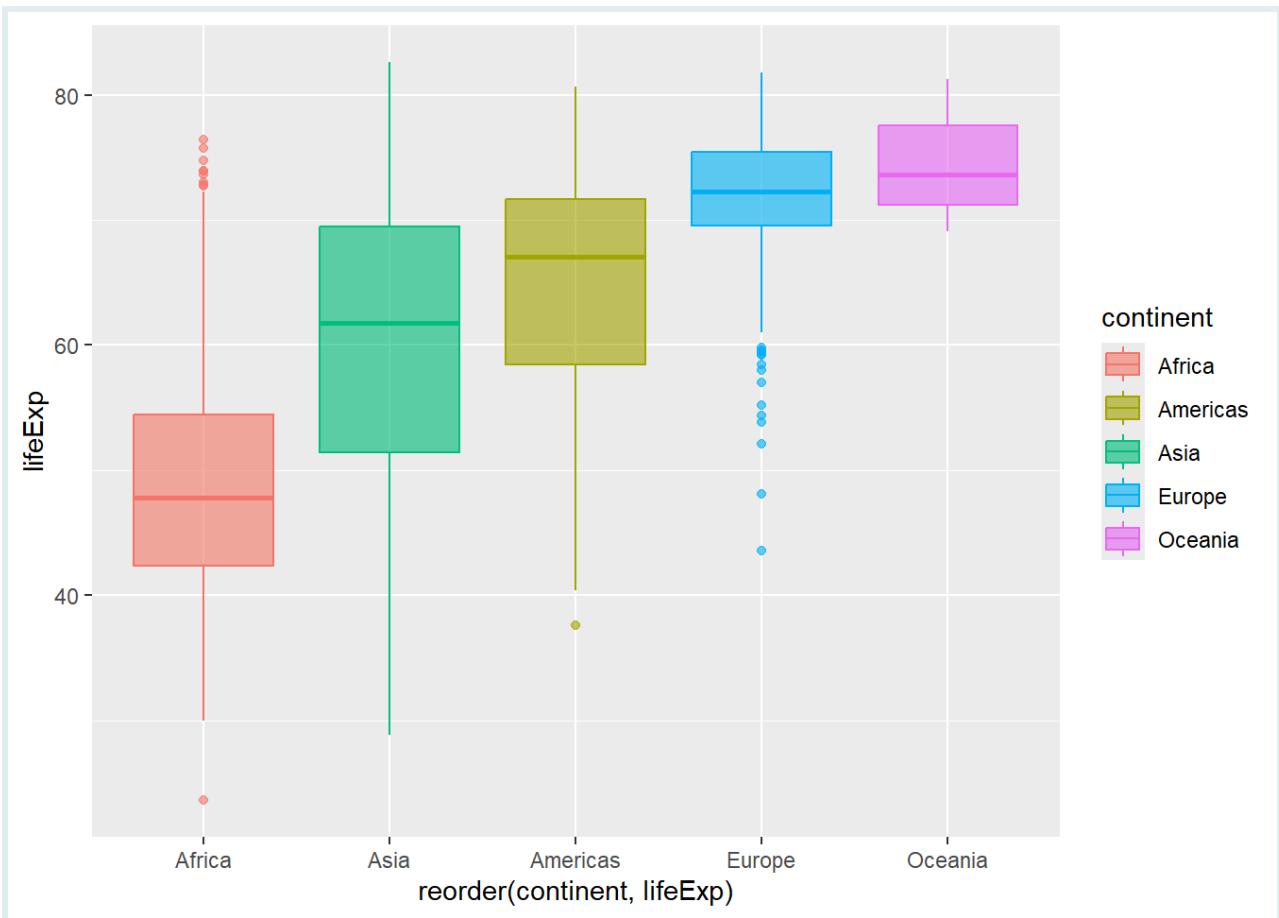
- Changez le **titre de l'axe des y** pour “Revenu par personne (USD)”.

## Ajouter des points de données avec `geom_jitter()`

Les boîtes à moustaches fournissent un résumé de la distribution d'une variable numérique pour plusieurs groupes. Sauf que résumer signifie aussi perdre de l'information.

Si nous prenons pour exemple notre boîte à moustache de `lifeExp`, il est facile de conclure que l'Océanie a une espérance de vie plus élevée que les autres continents. Cependant, nous ne pouvons pas voir la distribution sous-jacente des points de données dans chaque groupe ou le nombre d'observations.

```
# Boîte à moustache basique de lifeExp précédente
ggplot(gapminder,
       aes(x = reorder(continent, lifeExp),
           y = lifeExp,
           fill = continent,
           color = continent)) +
  geom_boxplot(alpha = 0.6)
```

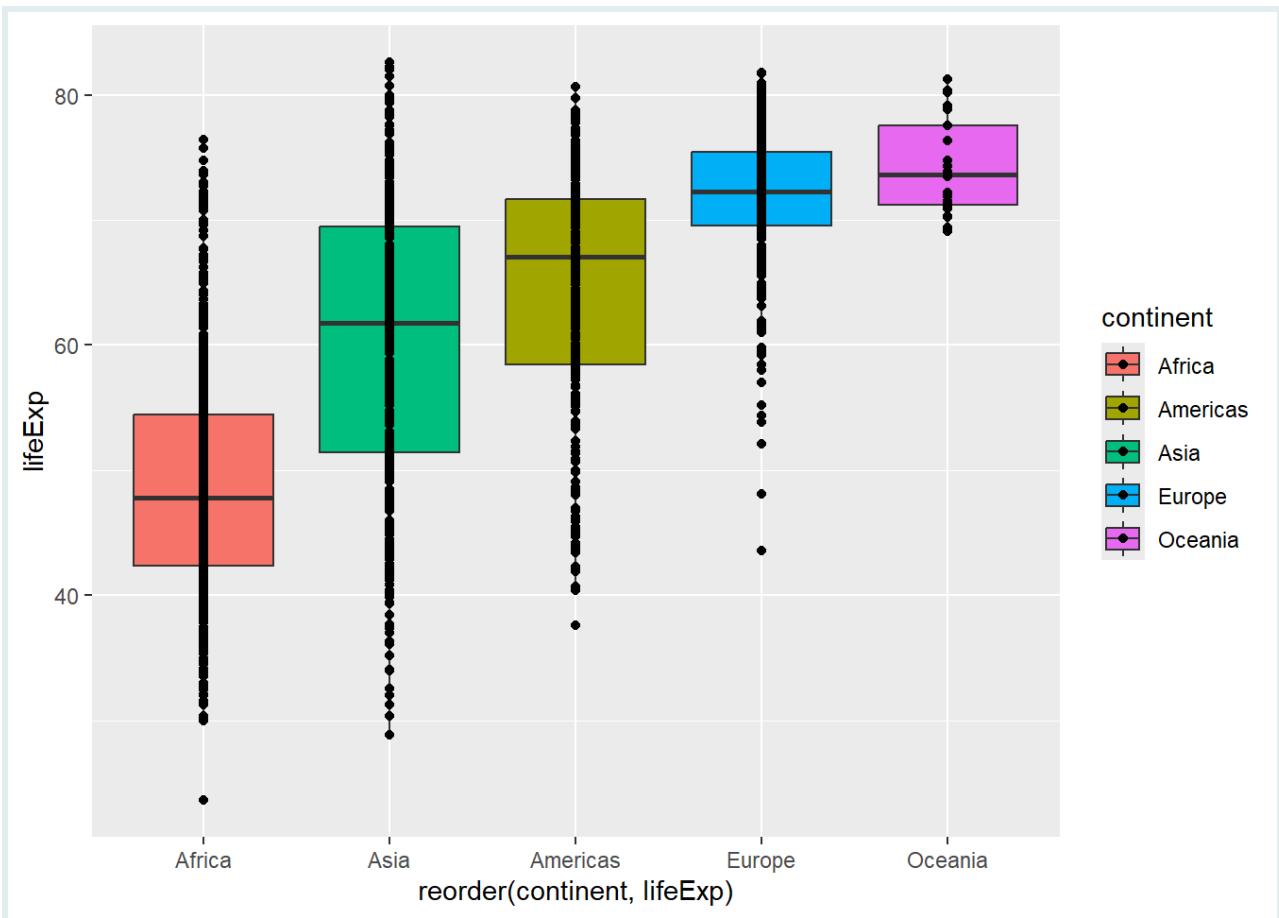


Voyons ce qui se passe lorsque la boîte à moustaches est améliorée en utilisant des éléments supplémentaires.

Une façon d'afficher la distribution des points de données individuels est de tracer une **couche supplémentaire de points** sur la boîte à moustaches.

Nous *pourrions* faire cela en ajoutant simplement la fonction `geom_point()`.

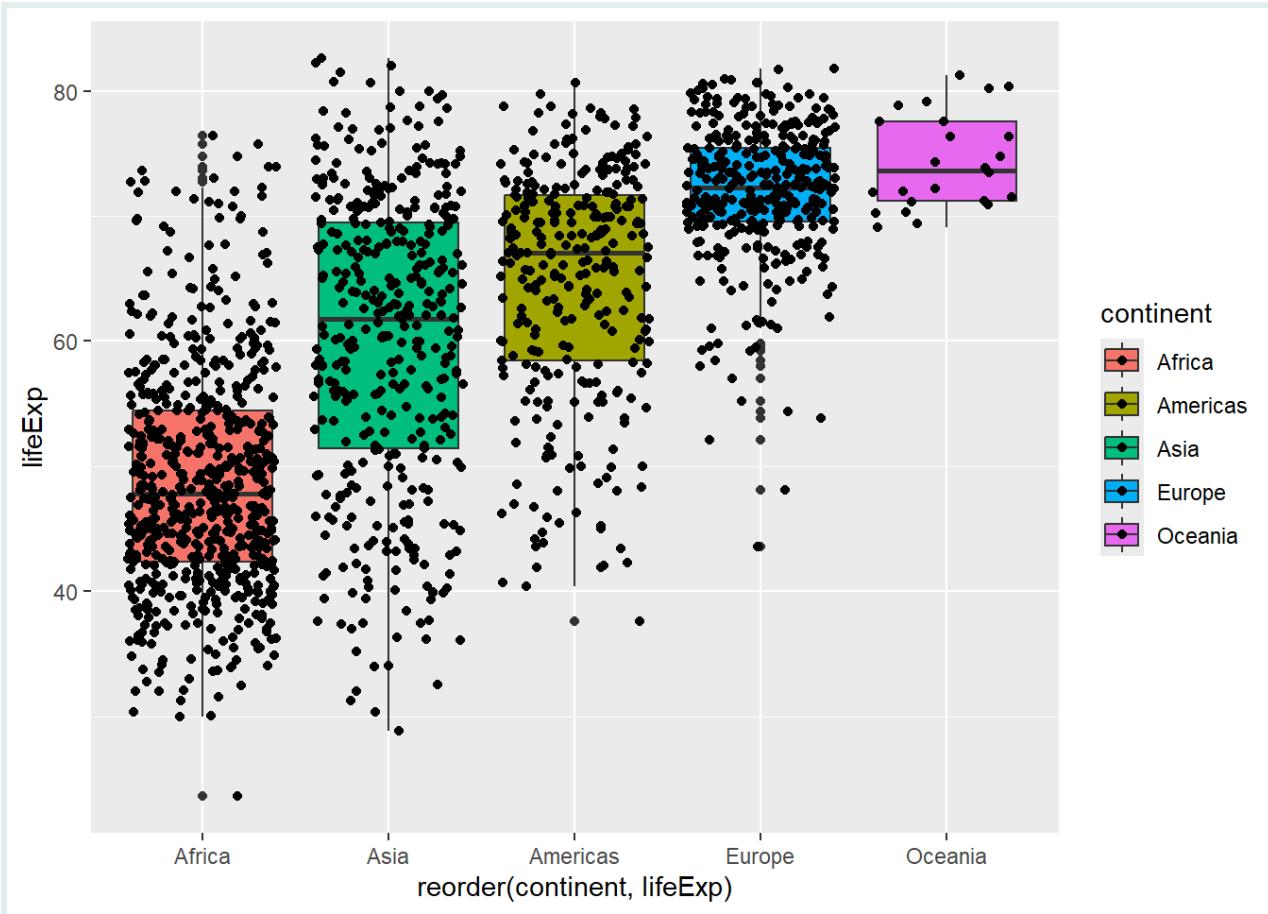
```
ggplot(gapminder,
       aes(x = reorder(continent, lifeExp),
           y = lifeExp,
           fill = continent)) +
  geom_boxplot() +
  geom_point()
```



Cependant, `geom_point()` a tracé tous les points de données sur une ligne verticale. Ce n'est pas très utile car tous les points d'un même continent se chevauchent et sont tracés les uns sur les autres.

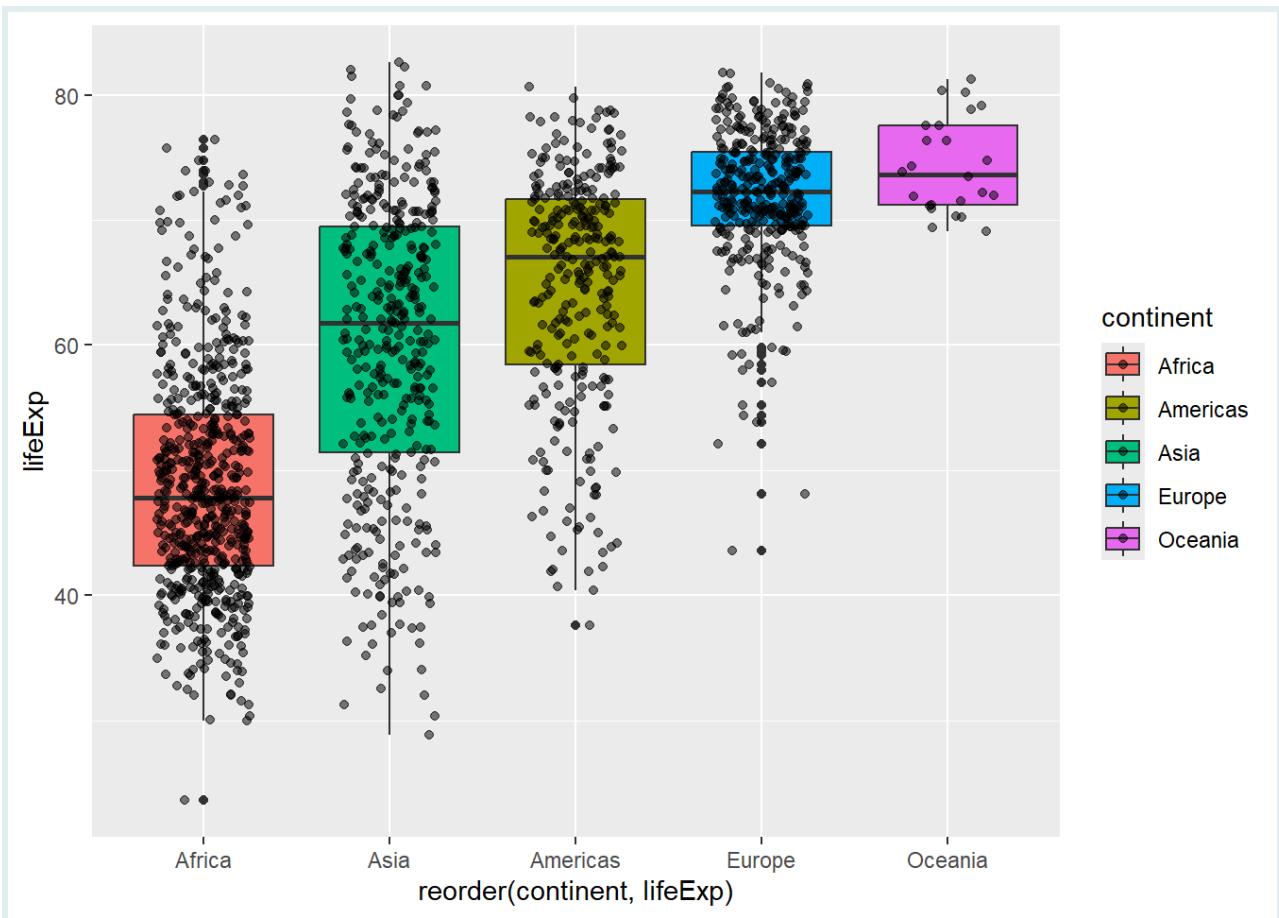
Une solution à cela est de décaler aléatoirement (jitter) les points de données horizontalement. `ggplot` vous permet de le faire avec la fonction `geom_jitter()`.

```
ggplot(gapminder,
       aes(x = reorder(continent, lifeExp),
           y = lifeExp,
           fill = continent)) +
  geom_boxplot() +
  geom_jitter()
```



Vous pouvez également contrôler la quantité de “jitter” avec l’argument `width` et spécifier l’opacité des points avec `alpha`.

```
ggplot(gapminder,
       aes(x = reorder(continent, lifeExp),
           y = lifeExp,
           fill = continent)) +
  geom_boxplot() +
  geom_jitter(width = 0.25,
              alpha = 0.5)
```



Ici, on se rend compte que l'Océanie a une plus petite taille d'échantillon par rapport aux autres groupes. C'est un élément important à prendre en compte avant de dire que l'Océanie a une espérance de vie plus élevée que les autres continents.



Les boîtes à moustaches sont limitées par le fait qu'elles résument les données en cinq nombres : le 1er quartile, la médiane (le 2ème quartile), le 3ème quartile, et les moustaches supérieure et inférieure. En faisant cela, nous pourrions passer à côté d'informations importantes sur les données. Une façon d'éviter cela est de visualiser les données avec des points.



- Créez une boîte à moustaches montrant la distribution du PIB par habitant pour chaque continent, comme vous l'avez fait dans l'exercice 2. Puis ajoutez une couche de points décalés.



- Ajustez votre réponse à la question précédente pour rendre les points 45% transparents et changez le “jitter” à 0.3mm.

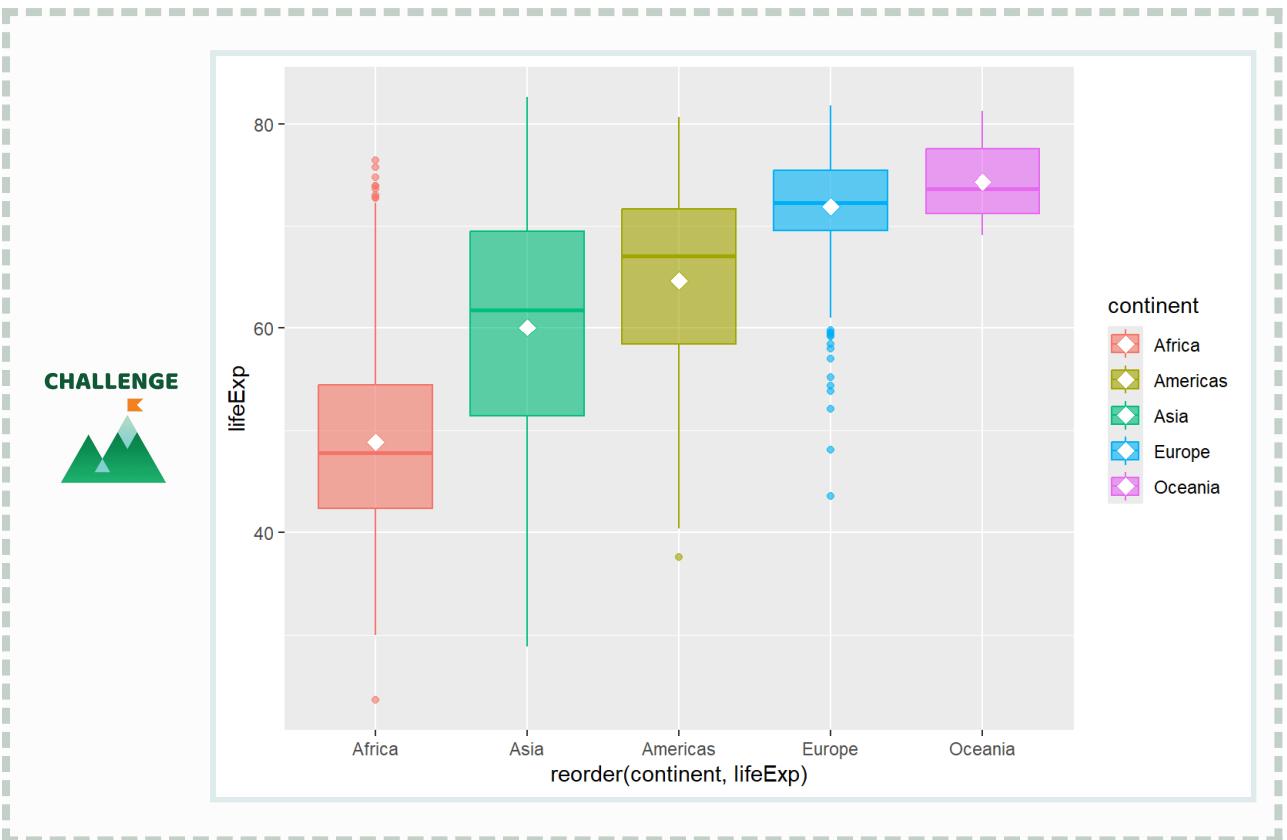


## Ajouter un indicateur de moyenne dans une boîte à moustaches

Il peut être intéressant de visualiser la moyenne des distributions sur une boîte à moustaches.

Nous pouvons le faire en ajoutant une couche de statistiques avec la fonction `stat_summary()`.

```
# Ajouter un indicateur pour montrer la moyenne
ggplot(gapminder,
       aes(x = reorder(continent, lifeExp),
           y = lifeExp,
           fill = continent,
           color = continent)) +
  geom_boxplot(alpha = 0.6) +
  stat_summary(fun = "mean",
              geom = "point",
              size = 3,
              shape = 23,
              fill = "white")
```



## En résumé

Les graphiques en boîte permettent de comparer la distribution d'une variable continue pour chaque niveau d'une autre variable. On peut voir où la médiane se situe pour les différents groupes en comparant les lignes à l'intérieur des boîtes.

Pour étudier la dispersion d'une variable continue à l'intérieur de l'une des boîtes, observez à la fois la longueur de la boîte et aussi jusqu'où les moustaches s'étendent de chaque extrémité de la boîte. Les valeurs aberrantes sont plus facilement identifiables sur une boîte à moustache qu'un histogramme car elles sont représentées par des points distincts.

## Les acquis

1. Tracer une boîte à moustaches (boxplot) pour visualiser la distribution de données continues en utilisant `geom_boxplot()`.
2. Réorganiser les boîtes à moustaches avec la fonction `reorder()`.
3. Ajouter une couche de points de données sur une boîte à moustaches en utilisant `geom_jitter()`.

## Contributeurs

Les membres suivants ont contribué à ce cours :



## JOY VAZ

R Developer and Instructor, the GRAPH Network  
Loves doing science and teaching science

---



## IMANE BENSOUDA KORACHI

R Developer and Instructor, the GRAPH Network

---



## ADMIN TEAM

GRAPH Courses Administration Team

The GRAPH Courses team is building epidemiological training courses to enhance disease surveillance and data science for public health across the globe

---

## Références

Le contenu de ce cours est en partie adapté des sources suivantes :

- Ismay, Chester, and Albert Y. Kim. 2022. *A ModernDive into R and the Tidyverse*. <https://moderndive.com/>.

This work is licensed under the [Creative Commons Attribution Share Alike](#) license.

