Science des données biologiques I

Philippe Grosjean & Guyliann Engels

2025-12-06

Préambule

Le contenu de ce cours est contextuel. Vérifiez les informations suivantes, s'il-vous-plait :

- Vous êtes anonyme sur ce site. **Votre progression dans les exercices ne sera pas enregistrée.**
- Cours : Science des données I : visualisation (S-BIOG-006)
- Institution : UMONS

Le contenu sera adapté en fonction de ce contexte. Vérifiez qu'il est correct, sinon fermez cette page et relancez-là depuis le système d'apprentissage en ligne de votre Université (Moodle, ...). Si les données sont toujours incorrectes, contactez vos enseignants.

Pour explorer ces pages de manière anonyme et n'enregistrer aucune activité, vous pouvez éliminez vos informations personnelles en cliquant sur le bouton juste ci-dessous.

Effacer mes données personnelles

Cet ouvrage est conçu pour être utilisé de manière interactive en ligne. En effet, vous y trouverez des vidéos, des démonstrations interactives ainsi que des exercices sous forme de questionnaires interactifs. **Ces différents éléments ne sont, bien évidemment, utilisables qu'en ligne.**

Science des données biologiques I

Philippe Grosjean & Guyliann Engels

2025-09-30

Préambule

Le contenu de ce cours est contextuel. Vérifiez les informations suivantes, s'il-vous-plait :

- Login : phgrosjean
- Email : phgrosjean@sciviews.org (email institutionnel : philippe.grosjean@umons.ac.be)
  Votre progression dans les exercices sera enregistrée sous cette identité.
- Cours : Science des données I : visualisation (S-BIOG-006)
- Institution : UMONS

Le contenu sera adapté en fonction de ce contexte. Vérifiez qu'il est correct, sinon fermez cette page et relancez-là depuis le système d'apprentissage en ligne de votre Université (Moodle, ...). Si les données sont toujours incorrectes, contactez vos enseignants.

Pour explorer ces pages de manière anonyme et n'enregistrer aucune activité, vous pouvez éliminez vos informations personnelles en cliquant sur le bouton juste ci-dessous.

Effacer mes données personnelles

Cet ouvrage est conçu pour être utilisé de manière interactive en ligne. En effet, vous y trouverez des vidéos, des démonstrations interactives ainsi que des exercices sous forme de questionnaires interactifs. **Ces différents éléments ne sont, bien évidemment, utilisables qu'en ligne.**

Informations système

Vue générale du cours

Cet ouvrage est le premier d'une série de trois volumes traitant de la science des données biologiques. L'écriture de cette suite de livres a débuté au cours de l'année académique 2018-2019. Pour l'année académique 2025-2026, ceci est le support du cours de Science des données I : visualisation et inférence au second Bachelier en Biologie en Faculté des Sciences de l'Université de Mons (UMONS), et dont le responsable est Philippe Grosjean.

La matière est divisée en dix modules représentant chacun six heures de travail en présentiel. Chaque module nécessitera environ dix heures (variable en fonction de votre rythme et de votre technique d'apprentissage, disons entre six et douze heures) de travail à domicile. **Une séance introductive de deux heures est programmée en début d'année pour installer les logiciels (SciViews Box, R, RStudio), et se familiariser avec eux, ainsi que pour expliquer la façon dont les séances vont se dérouler.**

Les cinq premiers modules au premier quadrimestre vous font découvrir les logiciels nécessaires pour effectuer les exercices de ce cours : R, RStudio, git, GitHub. Ils sont également consacrés à l'importation, le remaniement et la **visualisation** de données biologiques.

Les cinq modules suivants au second quadrimestre s'attaquent à l'**inférence** statistique, c'est-à-dire, l'art de tirer des conclusions sur l'observation d'un phénomène biologique au travers d'un échantillon, en présence d'incertitude, de variation entre individus et d'erreur de mesure qui sont impossible à éviter en biologie. Il s'agit ici d'utiliser de manière correcte les tests d'hypothèses statistiques et de réaliser des analyses de données reproductibles.

Matériel pédagogique

Le matériel pédagogique est aussi varié que possible. Vous pourrez ainsi piocher dans l'offre en fonction de vos envies et de votre profil d'apprenant pour optimiser votre travail. Vous trouverez :

- le présent ouvrage en ligne,
- des exercices H5P repérables par le logo  sur lequel vous pouvez cliquer pour avoir plus d'informations sur leur utilisation,
- des applications Shiny qui sont de véritables petits programmes avec interface Web écrits en R. Ils vous démontrent "en live" certains concepts. Ces applications doivent être lancées en cliquant sur l'image contenant le logo  et elles doivent être quittées avec le bouton `Quit` ou `Quit & Save` si l'enregistrement est activé. N'oubliez pas de soumettre votre réponse avec le bouton `Submit` . Si le serveur distant est trop lent ou indisponible, il est aussi possible de lancer ces applications Shiny directement dans RStudio (voir le message en italique qui apparaît en dessous de l'application),
- des tutoriels interactifs learnr. Vous pourrez exécuter ces tutoriels directement dans RStudio, et vous aurez alors accès à des pages Web réactives contenant des explications, des exercices et des quiz en ligne dans votre environnement de travail habituel. Ces tutoriels sont repérables par l'icône suivante  ,
- des dépôts GitHub Classroom dans la section BioDataScience-Course pour réaliser et documenter vos travaux personnels. Les liens vers ces dépôts sont repérables par l'icône GitHub  ,
- des renvois vers des documents externes en ligne, des vidéos Youtube ou Vimeo, des livres en anglais ou en français, des blogs, des tutoriels, des questions sur des sites comme Stackoverflow ou dans des mailing lists R, de X (ex Twitter)…

- des diapositives de présentations distribuées via le dépôt sdd_lessons sur BioDataScience-Course.

Tout ce matériel est accessible à partir du site Web du cours, du présent syllabus interactif et de Moodle pour les étudiants de l'UMONS.

Les travaux personnels seront à réaliser en utilisant une machine virtuelle préconfigurée, la **"SciViews Box"**, que nous installerons ensemble durant la séance introductive.

Vous pourrez poser vos questions par mail à l'adresse sdd@sciviews.org ou dans les "issues" du dépôt Github de ce cours. L'accès à ce dépôt vous sera donné au début du cours.

Un **outil d'annotation** et de surlignage est intégré dans le cours en ligne. Il vous permet :

- de personnaliser votre cours en écrivant dedans comme vous le feriez avec un syllabus papier (annotations privées… tout ce qui vous passe par la tête).
- d'échanger des informations complémentaires entre vous (par exemple, trucs et astuces supplémentaires, liens utiles, etc.) ou avec vos enseignants (passages moins clairs ou lacunes éventuelles). Soyez constructif dans vos commentaires publiques et réservez vos questions pour les "issues". Nous tiendrons compte de vos remarques pour améliorer le cours pour les années suivantes.

Pour annoter ou surligner, vous sélectionnez du texte et vous cliquez sur l'un des deux boutons `Annotate` ou `Highlight` qui apparaissent. Vous devez vous créer un compte (gratuit) sur hypothes.is auparavant. Vos annotations publiques sont à poster dans le groupe **BioDataScience-Course**. Vous devez vous y abonner en cliquant sur ce lien et ensuite sélectionner ce groupe dans la barre d'outil en haut du panneau d'annotation **avant d'ajouter vos commentaires, qu'ils soient personnels ou de groupe**.

Comment apprendre ?

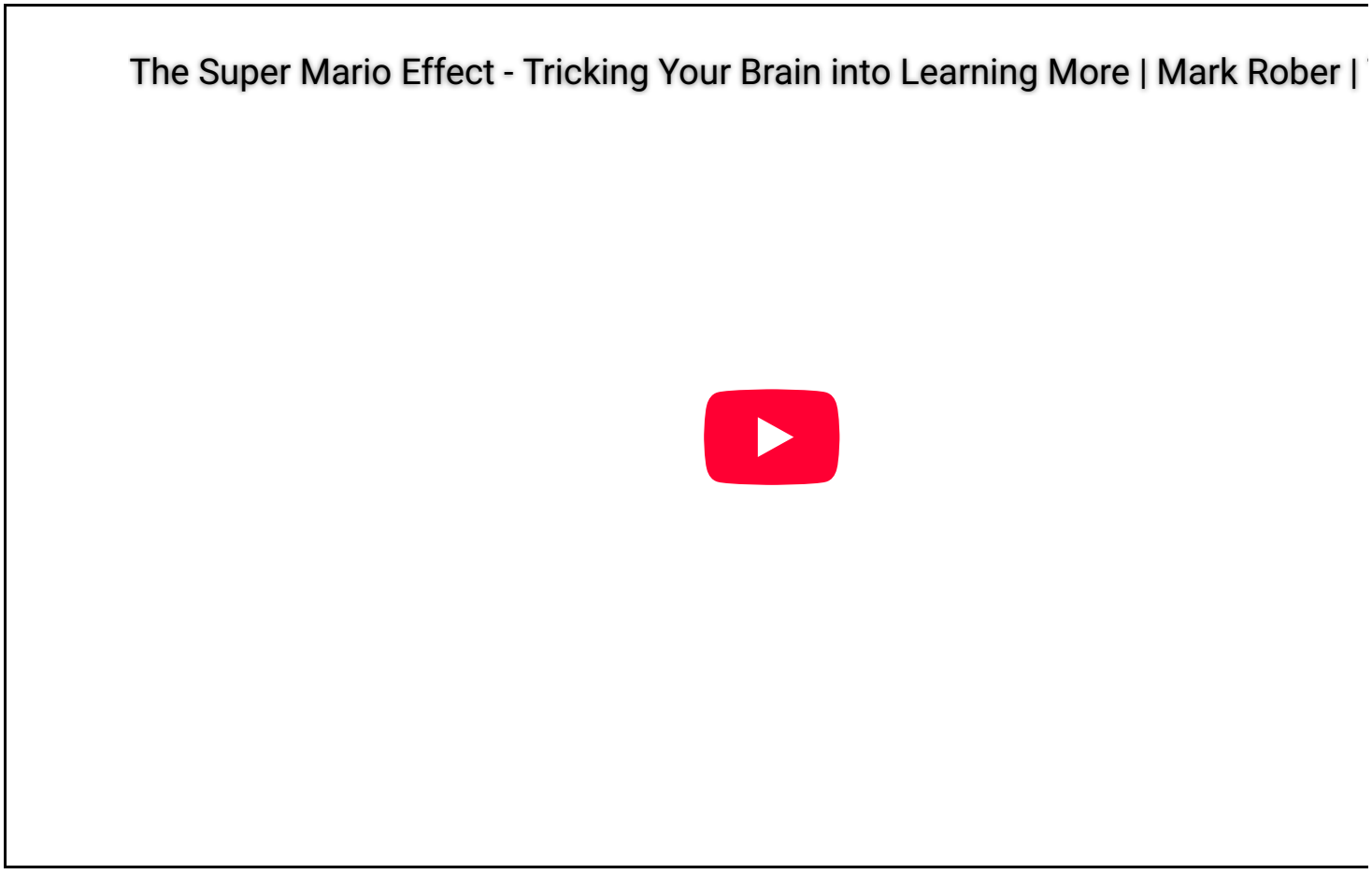
```
fortunes::fortune("brain surgery")


#
# I wish to perform brain surgery this afternoon at 4pm and don't know where
# start. My background is the history of great statistician sports legends bu
# am willing to learn. I know there are courses and numerous books on brain
# surgery but I don't have the time for those. Please direct me to the
# appropriate HowTos, and be on standby for solving any problem I may encount
# while in the operating room. Some of you might ask for specifics of the cas
# but that would require my following the posting guide and spending even mor
# time than I am already taking to write this note.
#    -- I. Ben Fooled (aka Frank Harrell)
#       R-help (April 1, 2005)
```

Version courte : **en pratiquant et en faisant des erreurs !**

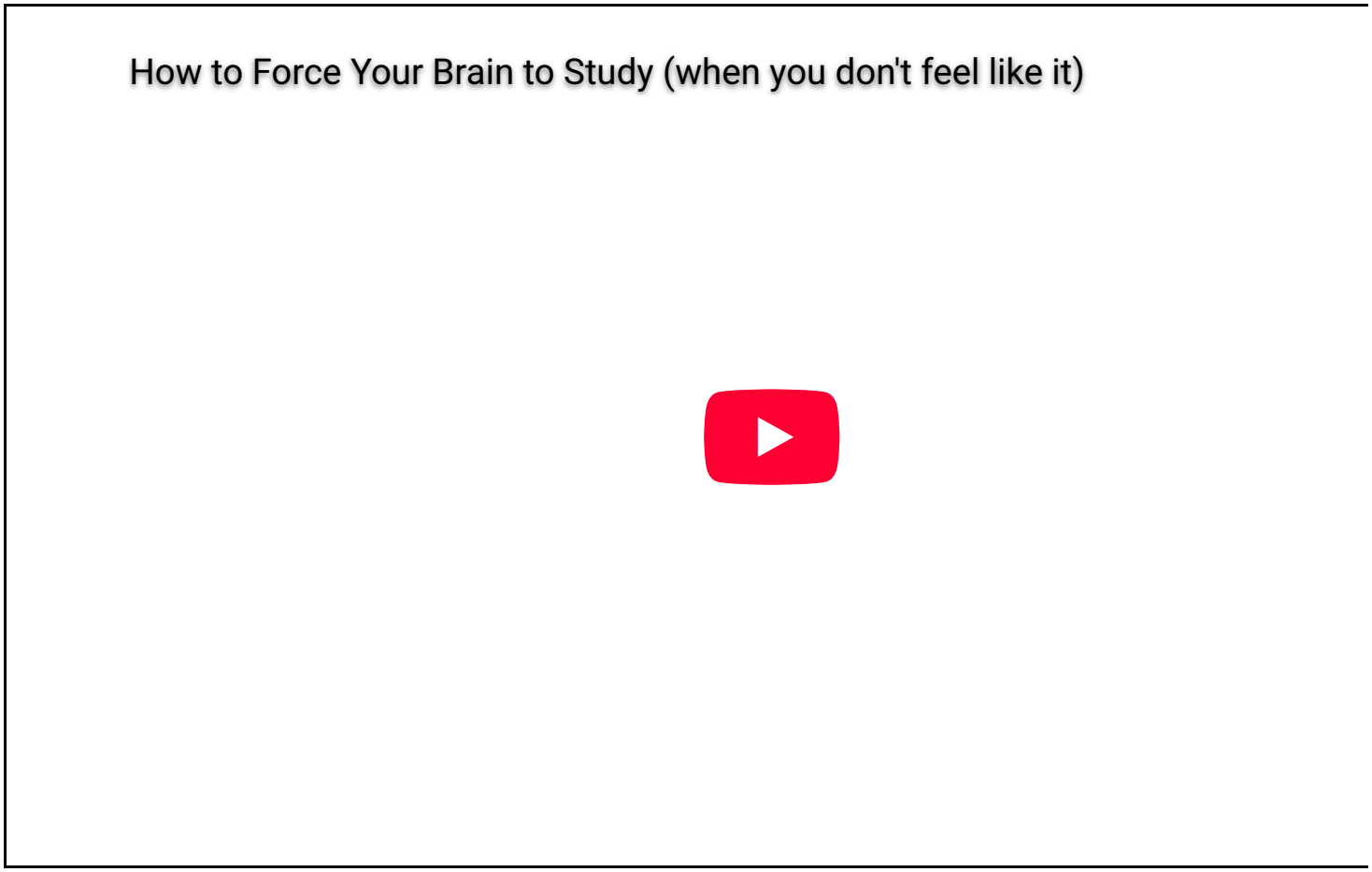
Version longue : aujourd'hui –et encore plus à l'avenir– les données sont complexes et ne se manipulent plus simplement avec un tableur comme Microsoft Excel, même si l'ajout de commandes en Python viennent considérablement l'améliorer. Vous apprendrez donc à maîtriser des outils professionnels très puissants, mais aussi relativement complexes. La méthode d'apprentissage que nous vous proposons a pour objectif prioritaire de vous faciliter la tâche, quelles que soient vos aptitudes au départ. Envisagez votre voyage en science des données comme l'apprentissage d'une nouvelle langue. **C'est en pratiquant, et en pratiquant encore sur le long terme que vous allez progresser.** Pour vous aider

dans cet apprentissage progressif et sur la durée, la formation s'étale sur quatre années, et est répartie en cinq cours de difficulté croissante. De plus l'évaluation est réalisée de manière **continue** tout au long de l'année pour favoriser ce travail sur le long terme.

N'hésitez pas à expérimenter, tester ou essayer de nouvelles idées (même au-delà de ce qui vous sera demandé dans les exercices) et surtout, **n'ayez pas peur de faire des erreurs**. Vous en ferez … beaucoup … *nous vous le souhaitons !* La meilleure manière d'apprendre, c'est en faisant des erreurs et en mettant ensuite tout en œuvre pour les comprendre et les corriger. Donc, si un message d'erreur apparaît, ne soyez pas intimidé. Prenez une bonne respiration, lisez-le attentivement, essayez de le comprendre, et au besoin, faites-vous aider : la solution est sur Internet, 'Google[1] est votre ami' ! L'idée ici n'est pas tellement éloignée de ce qui est présenté dans la vidéo suivante :



S'il vous semble que votre méthode de travail ou votre motivation n'est pas optimale, ou si vous trouvez la matière trop difficile, essayez la **méthode en échelle** qui consiste à travailler étape par étape en se concentrant d'abord sur ce qui parait facile pour vous, et ensuite, en repassant sur la matière tout en travaillant progressivement les autres aspects. Ceci est très bien expliqué dans la vidéo suivante :



1. Il existe tout de même des outils plus pointus pour obtenir de l'aide sur le logiciel R comme rseek.org ou rdrr.io. Rien ne sert de chercher "R" dans Google. De plus, l'émergence d'outils IA (intelligence artificielle) du style "chatGPT" sont aussi d'un grand secours. Nous proposons un **chatbot** taillé spécifiquement pour ce cours et accessible depuis votre SciViews Box, mais sinon, nous vous conseillons aussi Phind.↩

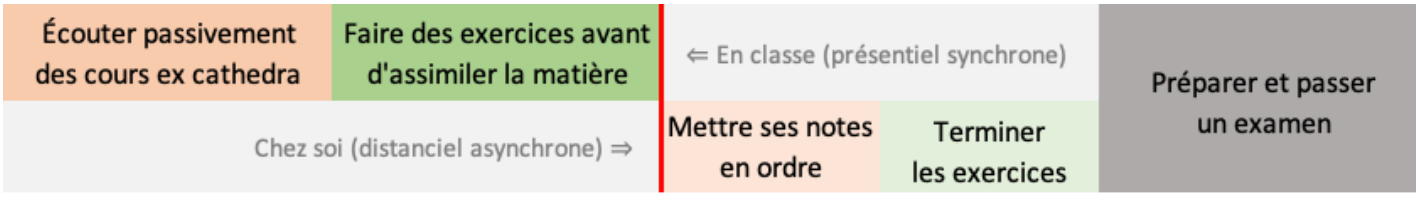
Approche pédagogique

Ce livre en ligne ainsi que tout le matériel pédagogique cité plus haut ont été développés pour être employés en pédagogie active en classe inversée. On peut résumer cette approche en deux phases. Vous apprenez d'abord à la maison, nous appliquons *ensuite* en présentiel.

Classe inversée & pédagogie active

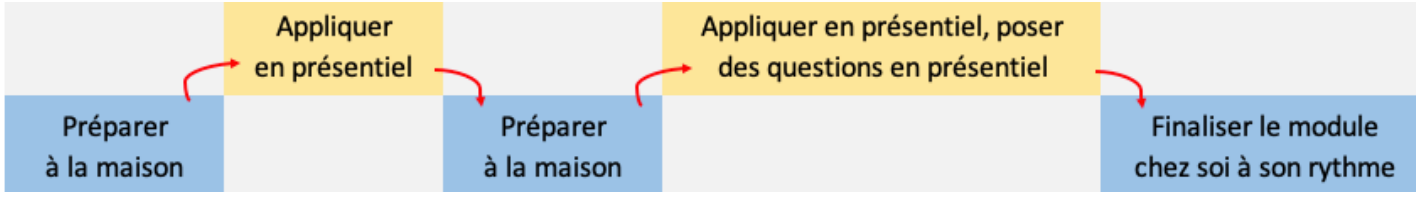
Jusqu'à présent, la plus grande partie de vos cours vous ont été donnés de manière classique. Vous avez donc suivi **passivement** un cours *ex cathedra* suivi, éventuellement, par une séance d'exercice ou de travaux pratiques.

On peut schématiser ce mode d'apprentissage comme cela



À la fin, un examen est indispensable pour valider l'acquisition des concepts.

Le cours de science des données I utilise une autre approche et se donne en **classe inversée**. Le schéma ci-dessous vous montre l'organisation d'un module du cours. Vous devrez donc préparer en distanciel la matière avant les séances en présentiel. Lors de chaque heure de travail lié à ce cours, vous serez donc **actif**.



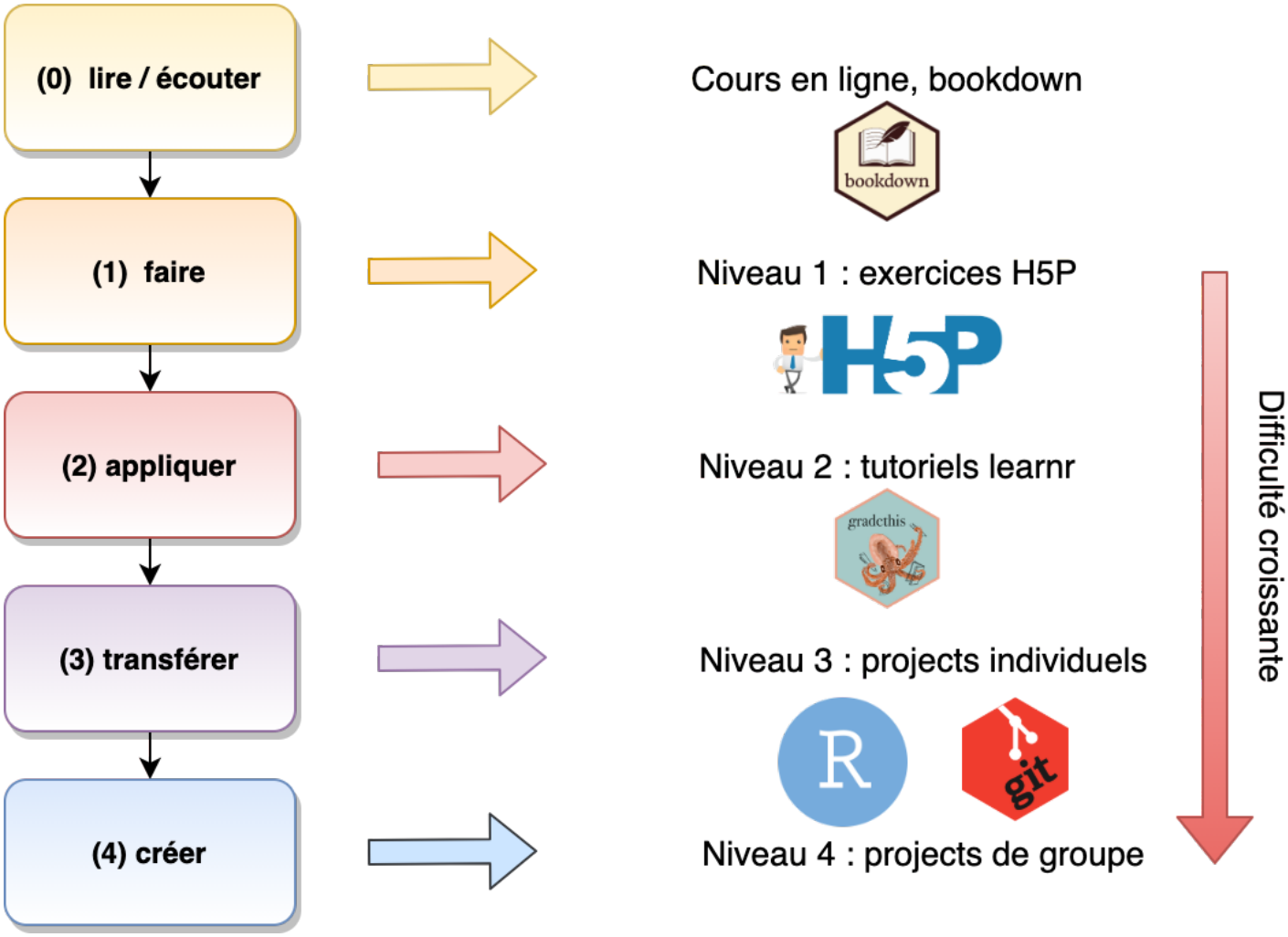
Une présentation de l'approche pédagogique du cours vous est faite durant la séance introductive obligatoire. La présentation est disponible au format PDF ici.

Quatre niveaux d'exercices

Quatre niveaux d'exercices de difficultés croissantes vous seront proposés.

- niveau I : découverir les notions dans des exercices rapides et ciblés intégrés au sein du cours en ligne.
- niveau II : appliquer les notions vues dans des exercices sous la forme de tutoriel cadré
- niveau III : transférer les notions dans des projets individuels guidés sur des données biologiques
- niveau IV : créer/réaliser des analyses dans des projets de groupe libres sur des données biologiques nouvelles grâce aux notions vues

Chaque type d'exercice est associé à un outil présenté dans le matériel pédagogique.



Toutes vos réponses sont collectées et utilisées afin de construire la note finale. Il n'est donc plus nécessaire de passer un examen classique en session d'examen. Tous ces exercices prouvent que vous avez acquis ou non les acquis d'apprentissage de ce cours. Cependant, vous aurez des **interrogations** classiques régulières sur la matière vue jusque là.

Votre progression au sein des exercices est mise à votre disposition à la fin de chaque module, ainsi que depuis Moodle.

> Les notes des exercices plus faciles de niveau 1, 2 et 3 sont comptabilisées séparément des exercices plus difficiles de niveau 4, des challenges et des interrogations, avec une pondération également en faveur de ces derniers. **Votre objectif est donc de bien vous préparer et de réussir vos projets de**

niveau 4, challenges et interrogations en présentiel. Les exercices faciles vous aident à bien vous préparer. **Faites-les sérieusement, c'est dans votre intérêt.**

L'évaluation est **continue** tout au long de l'année. Il n'y a pas d'examen en fin d'année et il n'y a pas moyen non plus de repasser un examen en seconde session. La présence aux séances est **obligatoire** pour cette raison et toute absence injustifiée vous coûtera un zéro pour la matière concernée.

Plan du cours

Le plan du cours est une forme de contrat entre les étudiants et les enseignants. Ce document structure les attentes de vos encadrants. Veuillez consulter ce document attentivement.

- Plan du cours de Science des données biologiques I : années 2025-2026

À vous de jouer !

Note : lorsque vous voyez le petit logo "H5P" comme ci-dessous, cela signifie que vous avez maintenant un exercice interactif. Cet exercice peut prendre différentes formes (quiz, présentation ou vidéo contenant des questions, vrai ou faux, cliquer sur une image …).

H5P

Combien d'heures de travail requiert un module du cours ?

- ○ Entre 9 et 12 heures
- ○ Entre 12 et 15 heures
- ○ Entre 15 et 18 heures
- ○ Entre 18 et 21 heures

Afficher la réponse

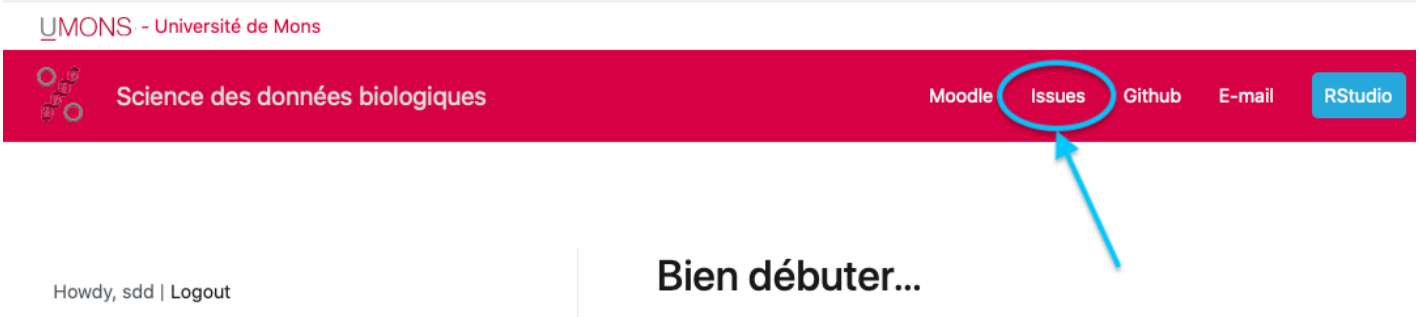
Issues

Toutes les questions relatives aux exercices sont à poser dans les **“issues” GitHub** dédiées à ce cours. Lors de la première lecture de ce préambule, vous ne connaissez pas encore les “issues”. Nous allons découvrir ce que cache ce terme lorsque nous travaillerons le premier module du cours. Vous rejoindrez cet espace de discussion par la suite.

> Rejoignez le projet **A00Qa_issues**.
>
> **Projet GitHub Classroom dédiée aux issues pour les étudiants inscrits au cours de Science des Données Biologiques I : visualisation à l’UMONS**
>
> Les issues ne sont pas accessibles aux étudiants externes.
>
> *Voyez les explications dans le fichier* `README.md` .

Note : vos enseignants sont également accessibles par mail via sdd@sciviews.org, mais uniquement si vos questions nécessitent un échange en privé, sinon, vous devez utiliser les issues.

Une fois que vous avez rejoint l’équipe **a25**, il n’est plus nécessaire d’utiliser le lien GitHub Classroom vers ces issues. Vous pouvez retrouver directement les issues liées à ce dépôt via la bannière du site.



Vous pouvez aussi directement créer une issue depuis RStudio. Voyez le fichier `README.md` dans le dépôt des issues pour les explications.

Enfin, vous apprendrez à utiliser les documents de type “learnr” dans le premier module. Vous avez à disposition un learnr de “feedback” qui vous permet de faire des suggestions sur le contenu de ce cours, module par module, ou de manière générale :

> Utilisez le document suivant pour vos suggestions sur le cours : **A99La_feedback (Learnr de feedback)**.
>
> ```
> BioDataScience1::run("A99La_feedback")
> ```

Pour terminer, encore un petit exercice rapide pour vérifier si vous avez lu suffisamment attentivement le contenu ci-dessus…

Écrivez ci-dessous la principale adresse mail à utiliser pour les questions relatives à ce cours

sdd@sciviews.org

Vérifier

Planning des séances

Dates importantes pour les activités de science des données biologiques I (à 23:59:59 pour les fins de projets N3 ou N4, à l'heure de la séance en présentiel pour les autres dates).

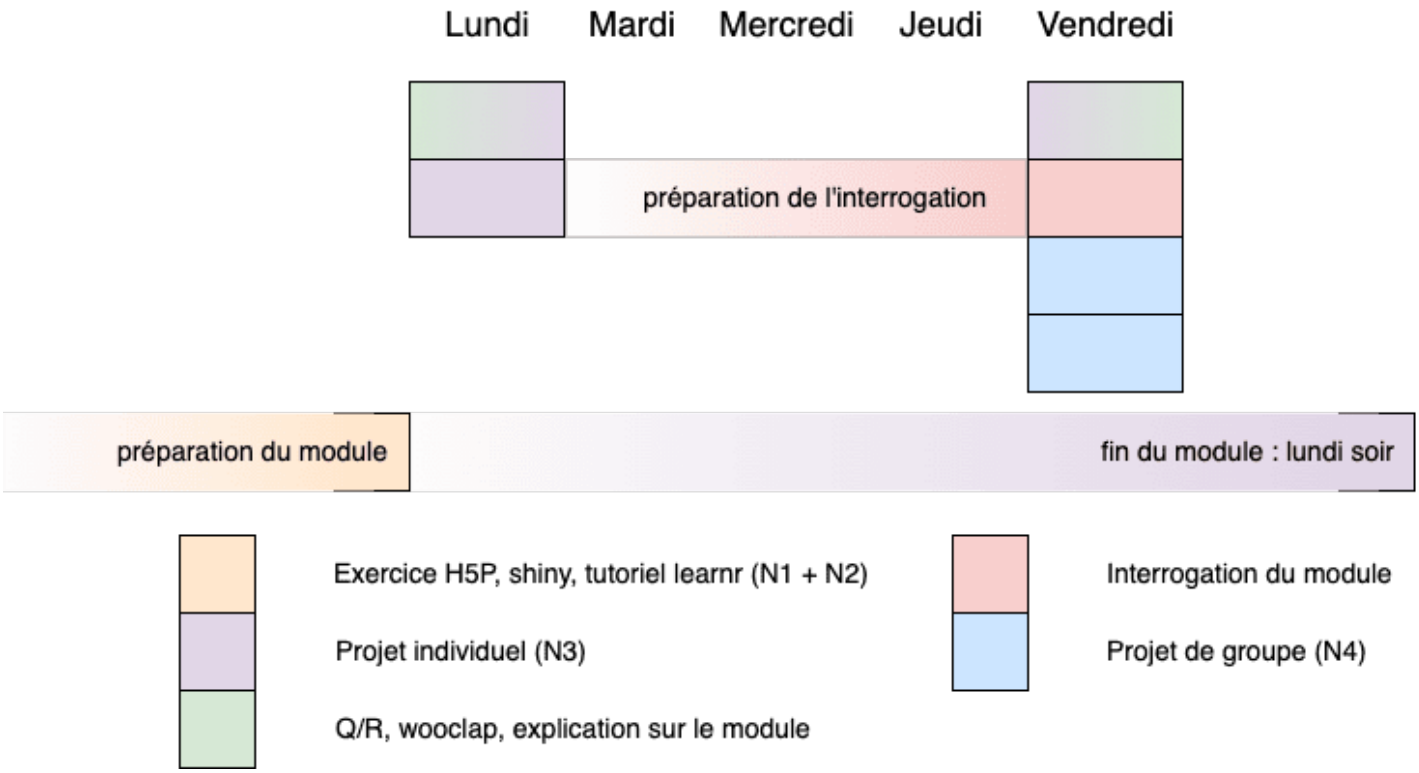
Module	Préparation pour le …	Fin N1-N3	Projet N4	Challenge	Interrogation
Quadri 1					
(install party)	2025-09-15	-	-	**-**	**-**
1 Intro. & visu. I	2025-09-22	2025-10-01	-	**-**	**2025-09-29**
2 Visu. II	2025-10-06	2025-10-13	2025-10-10	**-**	**2025-10-10**
(remédiation)	2025-10-13	-	-	**-**	**-**
(remédiation)	2025-10-20	-	-	**-**	**-**
3 Visu. III	2025-11-10	2025-11-17	continue…	**2025-11-14**	**-**
4 Trait. I	2025-11-24	2025-12-01	continue…	**-**	**2025-11-28**
5 Trait. II	2025-12-08	2025-12-15	**fin 2025-12-16**	**-**	**2025-12-12**
Quadri 2					
6 Proba. &		2026-			

corrél.	2026-02-09	02-16	-	-	**2026-02-13**
7 Chi^2	2026-02-23	2026-03-02	2026-02-27	-	**2026-02-27**
8 Moyenne	2026-03-09	2026-03-16	continue…	-	**2026-03-13**
9 Var. I	2026-03-23	2026-03-30	continue…	-	**2026-03-27**
(remédiation)	2026-04-13	-	-	-	-
10 Var. II	2026-05-04	2026-05-11	**fin 2026-05-12**	**2026-05-08**	-

- La colonne **préparation pour le…** indique la date pour laquelle vous devez avoir préparé[2] la *totalité* du chapitre correspondant du cours en ligne, y compris les exercices de niveaux 1 et 2 (H5P, learnr, shiny).
- Les **projets individuels de niveau 3** sont réalisés en présentiel. Vous ne pourrez les compléter efficacement que si vous avez bien préparé la matière. La date indiquée dans cette colonne correspond à la clôture des projets. Logiquement, vous devez avoir aussi terminé les exercices de niveaux 1 et 2 à ce moment-là.
- Les **projets de groupe de niveau 4** se font sur une plus longue durée et à cheval sur plusieurs modules. Étalez et répartissez le travail entre vous. Prévoyez le temps nécessaire pour *terminer* vos rapports avant la date de fin.
- Les **challenges** sont des exercices qui se font obligatoirement en séance sous forme de jeux, type compétition entre vous ou contre la montre. *Ils nécessitent d'être au point dans la manipulation du logiciel pour l'analyse de données biologiques en pratique.*
- Les **interrogations** sont des contrôles classiques en présentiel sur feuille de papier ou sur ordinateur et limités à 30, 45 ou 60 minutes. *Réussir implique que vous ayez assimilé toute la matière vue jusque là.* Prévoyez entre une demi et une journée et

demie pour l'étude et la révision avant cette date, selon votre rythme de travail.

Les **séances en présentiel** sont généralement réparties sur une semaine avec **deux heures le lundi et quatre heures le vendredi**, sauf quand l'horaire impose une autre configuration.



- La **première séance de deux heures du lundi** est consacrée aux questions/réponses sur la matière et aux courtes explications en classe. Le projet individuel de niveau 1 est débuté. *Les étudiants doivent étudier la matière pour l'interrogation ou le challenge* et préparer leur seconde série de questions pour la séance suivante.
- Les **deux premières heures de la seconde séance du vendredi** sont consacrées à la poursuite du projet de niveau 1 qui doit normalement se terminer à ce moment-là, ou presque. Une nouvelle séance de **questions/réponses** et de discussion sur la matière est également prévue. Elle vise à vous préparer au mieux à l'interrogation ou au challenge. Ensuite, après une courte interruption, **une interrogation écrite** a lieu durant environ une demi-heure, sauf autres directives (parfois 45 minutes ou même une heure).
- Les **deux dernières heures de la seconde séance** sont consacrées au travail sur le **projet de groupe**. *À l'issue de ces deux séances, les étudiants terminent leurs exercices de niveaux 1-3 chez eux*, généralement pour le lundi suivant.

Notez les dates clés dans vos agendas. Les informations ci-dessus sont fournies à titre indicatif pour vous aider à planifier votre travail. Cependant, elles pourraient légèrement évoluer au cours de l'année académique en fonction de l'évolution de l'apprentissage et/ou d'imprévus.

2. Préparer la matière en classe inversée signifie que vous avez **étudié** ce que vous comprenez par vous-même et que vous avez **noté** vos questions relatives à ce que vous ne comprenez pas pour les poser en séance. Il ne s'agit pas *juste* de lire de manière distraite le syllabus !↩

Module 1 Introduction & visualisation I

Objectifs

- Se familiariser avec les outils logiciels de base pour la science des données (SciViews Box, RStudio, logiciel R (R Core Team 2025), GitHub, Git).
- Réaliser son premier document écrit en Quarto/R Markdown (Allaire et al. 2024) (entêt YAML, zones Markdown et chunks de code).
- Être capable de réaliser différentes variantes d'un graphique en nuage de points dans R avec la fonction `chart()`
- Intégrer ensuite des graphiques dans un rapport et y décrire ce que vous observez

> Les liens cliquables qui vous sont proposés dans ce cours ont été spécialement sélectionné pour vous. La section **Prérequis** vous permet de vous assurer que vous avez toutes les connaissances nécessaires pour aborder le module dans les meilleures conditions possibles.
>
> **Cliquez sur ces liens**

Prérequis

La partie suivante est repliée. Il s'agit d'une partie optionnelle, mais néanmoins indispensable pour réaliser les exercices de ce cours qui vont être effectués sur ordinateur

Si vous considérez que vous maîtrisez les éléments de base d'un ordinateur, le clavier, que vous êtes capables de trouver toute une série de caractères, dont des caractères spéciaux et des raccourcis clavier, vous pouvez sauter cette partie repliée et continuer votre lecture. En cas d'hésitation, dépliez la partie afin d'approfondir vos connaissances.

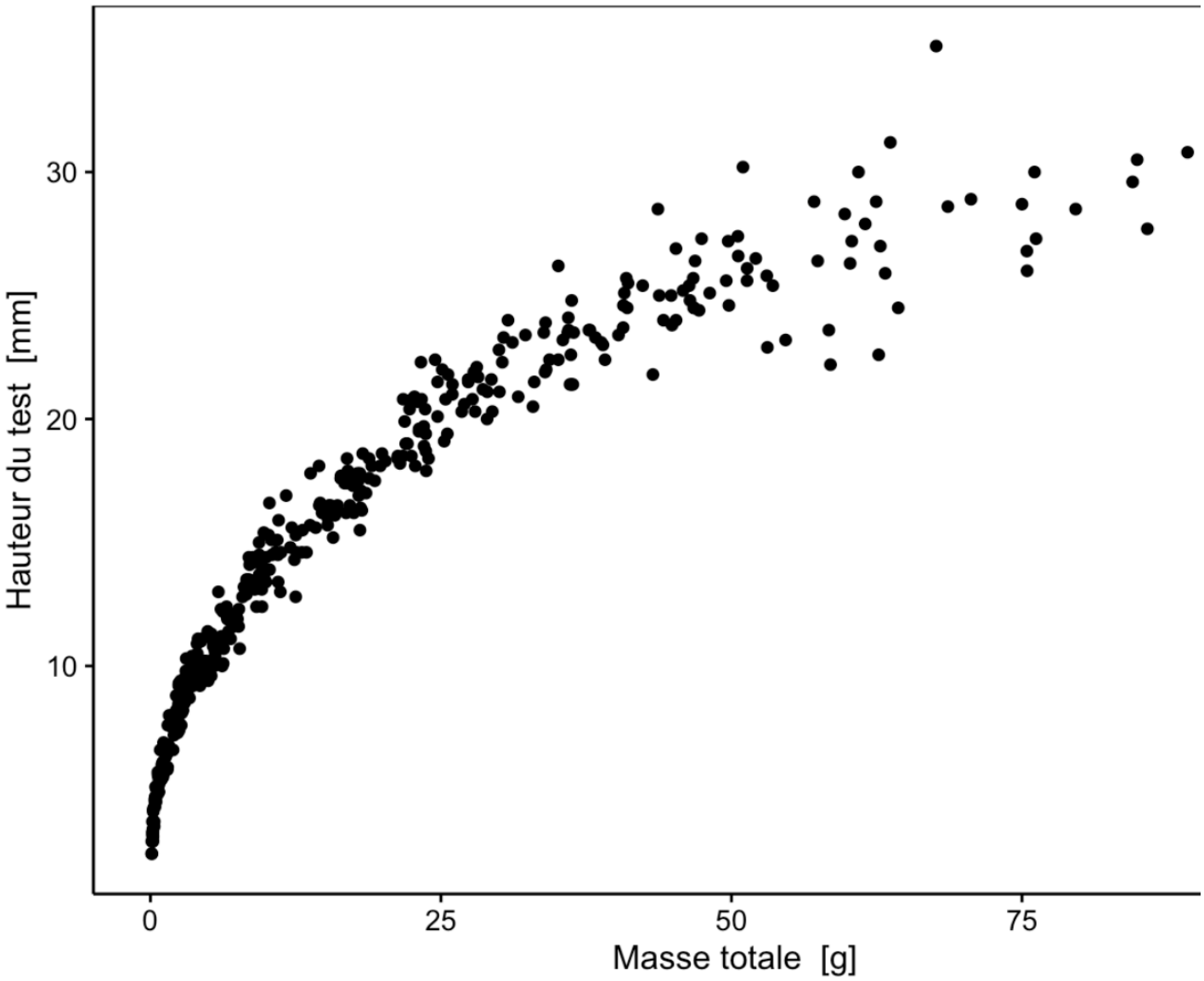
▸ Élément de base d’un ordinateur et des navigateurs web

L’objectif principal de ce module est de réaliser des graphiques en nuage de points. Il est de ce fait important de maîtriser des éléments de géométrie (origine, abscisse, ordonnée, coordonnées d’un point dans un plan) mais également des fonctions mathématiques comme les puissances et les logarithmes. Vous trouverez une nouvelle partie repliée juste après l’exercice ci-dessous pour vous rafraîchir la mémoire concernant ces notions.

À vous de jouer !

Le graphique ci-dessous présente la variation de la hauteur du test en fonction de la masse d’oursins (n’hésitez à faire des petites recherches par vous-même, si vous ne savez pas ce qu’est le “test” d’un oursin).

Sélectionnez l'axe des ordonnées sur le graphique ci-dessous



▸ Notions élémentaires de géométrie et de fonctions mathématiques

Le préambule de ce cours vous a également permis d'en apprendre davantage sur le matériel pédagogique, l'approche pédagogique ou encore la méthode d'évaluation. Vérifiez votre compréhension des notions vues avec l'exercice suivant.

À vous de jouer !

H5P

Déplacez les textes dans les emplacements qui leur correspondent.

Le cours de science des données I est donnée en . Les cours sont évités ou limités à des explications courtes. Chaque module de cours requiert de travail dont en présentiel. Les étudiants sont les de leurs apprentissages et les encadrants des .

classe inversée
6h
facilitateurs
ex cathedra
acteurs
15h

Vérifier

À vous de jouer !

Note : les tutoriels learnrs vous permettent d'autoévaluer l'acquis de connaissances et de compétences. Dans le cours, ils sont marqués d'une icône en forme de toque verte. Ils s'exécutent directement dans RStudio à l'intérieur de la SciViews Box (dans SaturnCloud). Vous copiez l'instruction `BioDataScience1::run("A00La_discovery")` *et la collez dans l'onglet Console de RStudio. Ensuite, vous validez la commande en cliquant sur la touche <entrée> et le tutoriel doit se lancer. Vous pouvez ouvrir ensuite le tutoriel dans sa propre fenêtre plus spacieuse en cliquant sur le second bouton de la barre d'outils en forme de petite fenêtre avec une flèche blanche.*

> Effectuez maintenant les exercices du tutoriel **A00La_discovery (Découverte de learnr)**.
>
> ```
> BioDataScience1::run("A00La_discovery")
> ```

Vous pouvez aussi voir (ou revoir) les dias de la présentation qui vous est faite en séance introductive.

Références

Allaire, JJ, Yihui Xie, Christophe Dervieux, Jonathan McPherson, Javier Luraschi, Kevin Ushey, Aron Atkins, et al. 2024. *Rmarkdown: Dynamic Documents for r*. https://github.com/rstudio/rmarkdown.

R Core Team. 2025. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

1.1 Découverte des outils

L'information est partout et en quantité inimaginable. Pour ne citer qu'une valeur, en 2020, la quantité d'information ajoutée chaque semaine sur Internet était de 1000 milliards de milliards de bits (unité de base en informatique représentée par des 0 ou des 1). Pour traiter même une toute petite partie de ces données en lien avec la biologie, vous devrez être un scientifique des données capable d'employer toute une série d'outils (logiciels).

À la fin de ce premier module, vous aurez réalisé votre première analyse complète en biologie, et vous la réaliserez avec des outils professionnels que vous utiliserez durant toutes vos études et même après dans votre profession en lien avec la biologie.

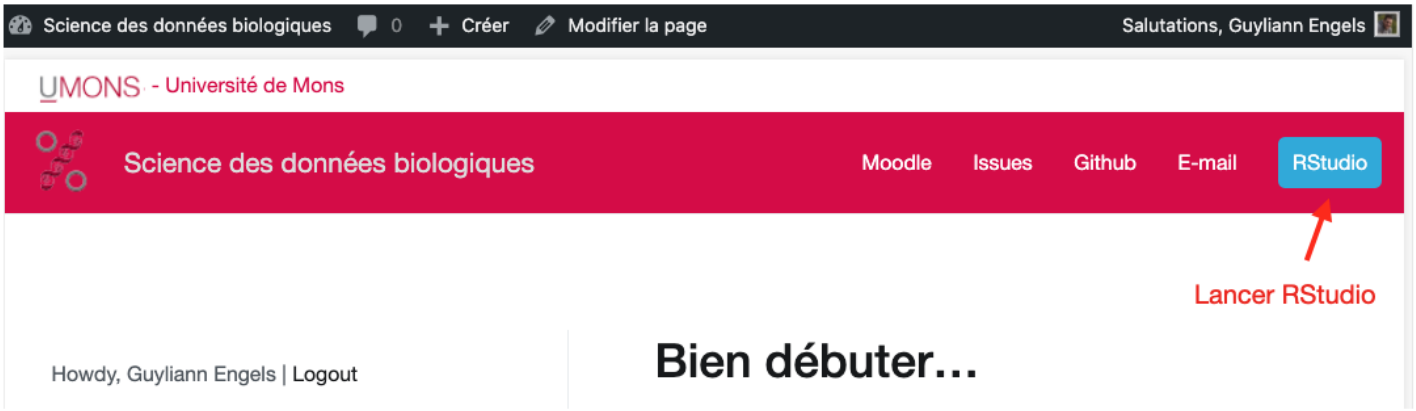
La science des données requiert d'employer des outils performants que nous avons sélectionnés pour vous parmi la multitude de logiciels disponibles, car nous faisons le pari que ce seront les outils qui vous seront les plus utiles dans les 30 prochaines années, c'est-à-dire pendant une bonne partie de votre carrière.

Les outils logiciels que vous apprendrez à utiliser dans ce cours vont littéralement vous transformer. Vos capacités d'analyse et de compréhension du vivant seront transcendées.

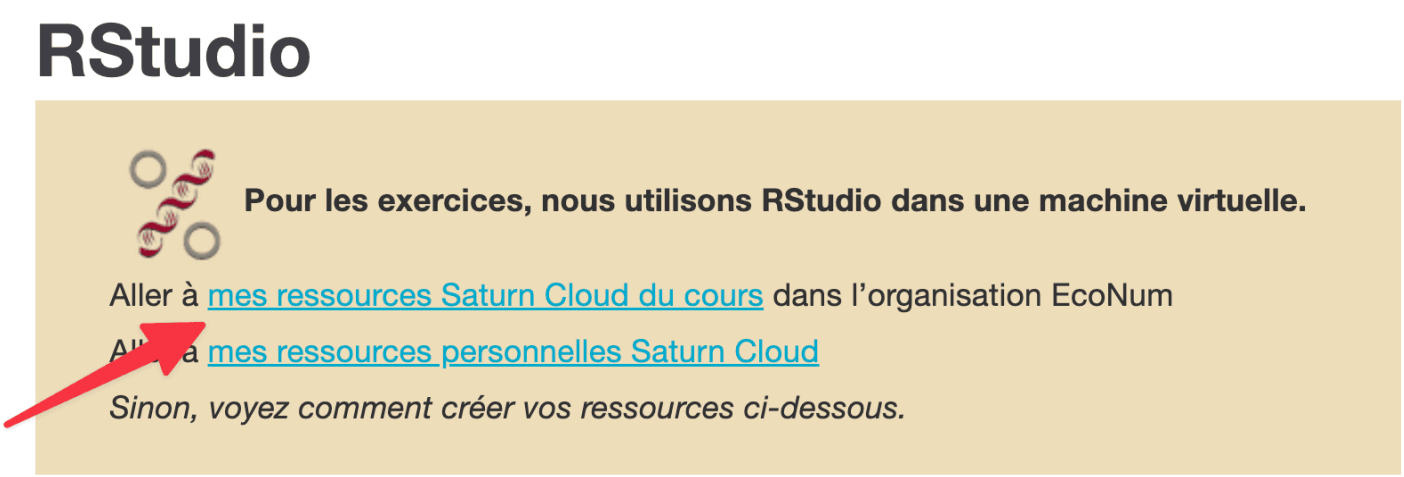
1.1.1 SciViews Box

Dans ce cours, nous utilisons différents logiciels qui nécessitent une installation et une configuration en plusieurs étapes. Pour vous économiser ces étapes fastidieuses, nous employons un système complètement préconfiguré : la **SciViews Box** pour laquelle une nouvelle version est préparée avant chaque nouvelle année académique. Il s'agit d'une **machine virtuelle**, un ordinateur complet, mais dématérialisé en quelque sorte. Nous utilisons un système fonctionnant sur le Cloud et nommé **SaturnCloud**. SaturnCloud ne nécessite pas un ordinateur puissant pour y accéder, mais il faut par contre, une bonne connexion Internet. Il faut aussi que le serveur distant soit disponible et offre les ressources suffisantes pour travailler.

Depuis le cours en ligne, on accède facilement à la SciViews Box.

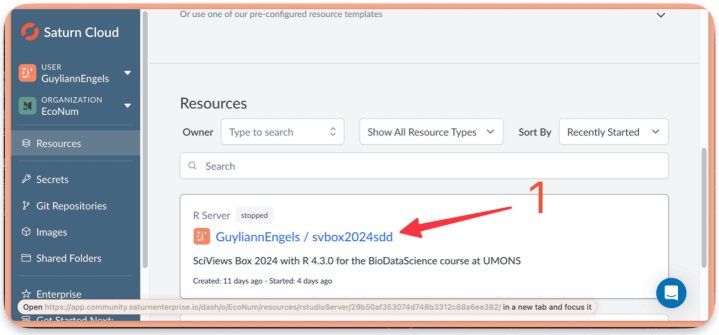


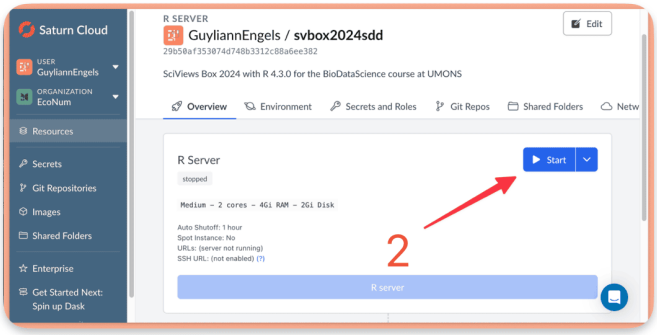
Le bouton RStudio vous conduit vers une page qui vous permet d'accéder à votre machine virtuelle hébergée sur Saturn Cloud dans l'organisation "EcoNum". Vous observez qu'il n'est pas inscrit SciViews Box mais RStudio sur le bouton. Il s'agit du logiciel qui va nous permettre de réaliser vos rapports d'analyse.

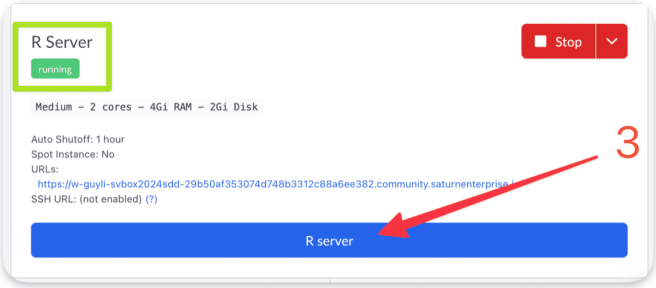


Cette page contient également toutes les instructions indispensables pour :

- installer et configurer la SciViews Box, composée entre autres des logiciels R et RStudio nécessaires pour ce cours
- allumer, éteindre, valider votre configuration.







1.1.2 RStudio

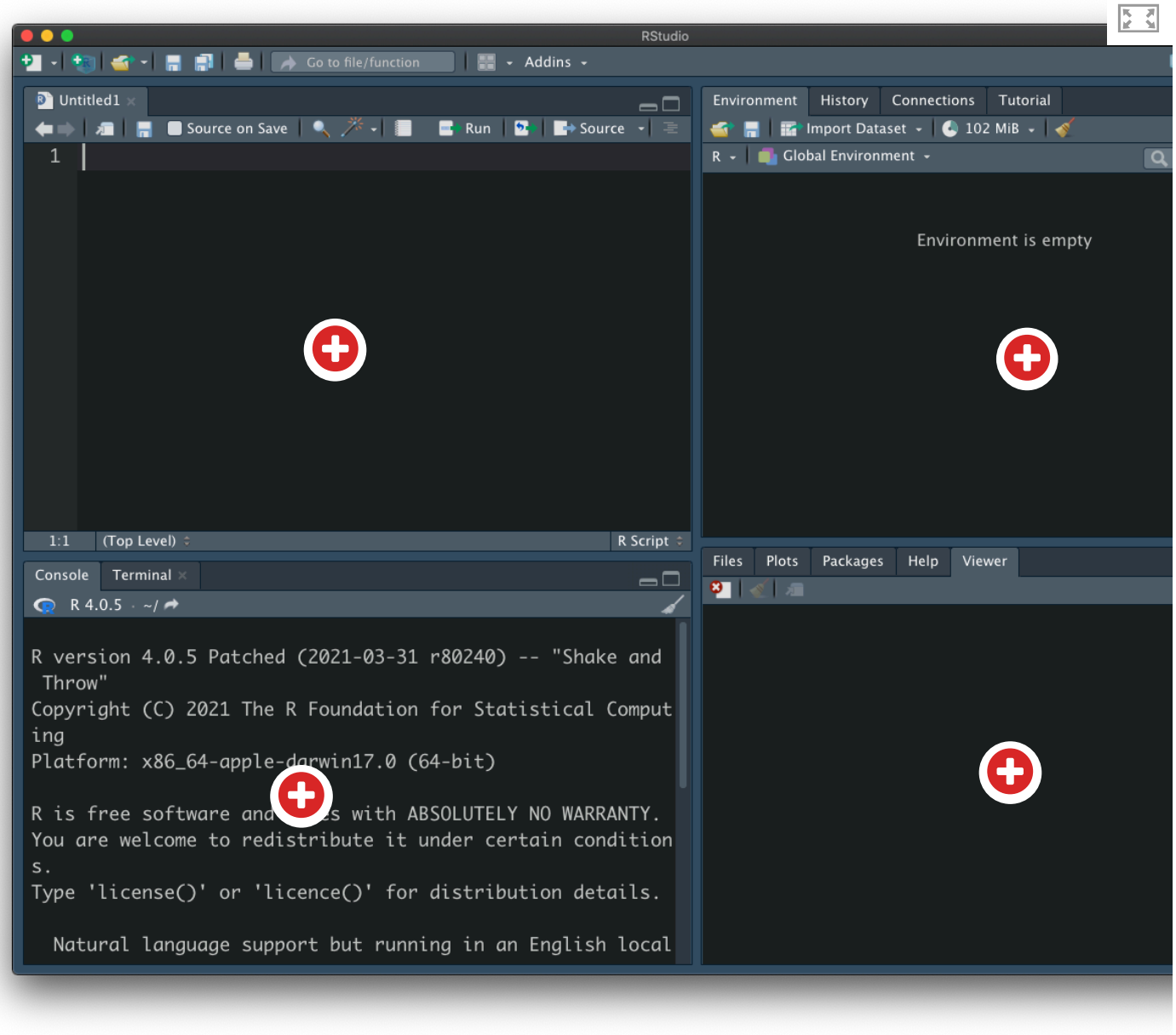
RStudio permet la rédaction des rapports dans les meilleures conditions et d'y intégrer des graphiques, des tableaux ou encore des analyses statistiques. Des éditeurs de texte classiques comme Google Docs ou Microsoft Word ne sont pas orientés vers la production de documents techniques ou scientifiques. Nous utiliserons donc un logiciel complet et optimisé pour produire de tels documents : RStudio. Celui-ci s'appuie lui-même sur R qui est un langage taillé pour traiter des données, produire des graphiques et des tableaux et réaliser des analyses statistiques.

L'interface de RStudio se présente en quatre sous-fenêtres (on dit aussi des "panneaux" ou *panes* en anglais) que vous pouvez découvrir ci-dessous.

À vous de jouer !

Cliquez sur les symboles + pour découvrir le rôle de chaque sous-fenêtre de RStudio.

H5P



Des explications détaillées se trouvent dans l'annexe B.1 qui présente les bases de l'utilisation de RStudio. Vous avez également à votre disposition un aide-mémoire pour appréhender cette interface RStudio IDE Cheat Sheet.

Pour en savoir plus

- RStudio. Site Web de RStudio qui rassemble un ensemble de ressources en anglais
- RStudio, un environnement de développement pour R. Brève explication de RStudio en français.
- RStudio : sa vie, son œuvre, ses ressources. Une autre courte page en français qui présente RStudio.

1.1.3 R, les éléments de base

Dans la section précédente, vous avez pu découvrir que le logiciel **R** allait être un outil central dans cette formation en science des données.

R est un logiciel **open source** (c'est-à-dire que son code source est disponible ; un logiciel est constitué de la **compilation** d'un code source qui peut être rendu public comme ici, ou non) spécialisé dans l'analyse de données. Le langage de programmation R qu'il implémente est mature et développé depuis 1993. Il prend ses sources dans le langage S (spécialement conçu pour les statistiques dans les années 1970). R permet, entre autres, la manipulation et la visualisation de données, ainsi que les calculs statistiques. C'est l'un des logiciels les plus utilisés et les plus puissants pour l'analyse des données, avec Python, Julia et Matlab. Donc, étudier R sera un **investissement clé** pour votre future carrière de biologiste, car des données, vous en aurez tous à en analyser !

À vous de jouer !

Au tout début de ce premier module, vous avez lancé votre premier tutoriel intitulé “A00La_discovery” qui traitait de la découverte des learnr. Il était composé de questions à choix multiples et de zones de code R. Vous l’avez utilisé comme une calculatrice. Cependant, R est bien plus puissant qu’une calculatrice. Les tutoriels servent aussi à vous entraîner à écrire des instructions en R. Avant de vous lancer dans ces exercices, consultez l’Appendice C pour apprendre à utiliser correctement ces tutoriels “learnrs” pour les questions relatives à du code R.

> Effectuez maintenant les exercices du tutoriel **A01La_base (Les bases de R)**.
>
> ```
> BioDataScience1::run("A01La_base")
> ```

Vous venez de découvrir les assignations, les fonctions, le chaînage d’instructions… Lancez-vous à présent dans une première analyse concrète de données biologiques qui sera l’occasion de découvrir comment on crée des graphiques avec R.

1.2 Nuage de points

Dès que vous commencez à maîtriser les principes de base de R, vous allez pouvoir réaliser assez rapidement de beaux graphiques. Par exemple, si vous souhaitez représenter une variable numérique en fonction d'une autre variable numérique, vous pouvez exprimer cela sous la forme d'une **formule**[3]

$$y \sim x$$

que l'on peut lire "y en fonction de x". Pour les deux variables numériques *x* et *y*, la représentation graphique la plus classique est le **nuage de points** (voir Fig. 1.1 pour un exemple).

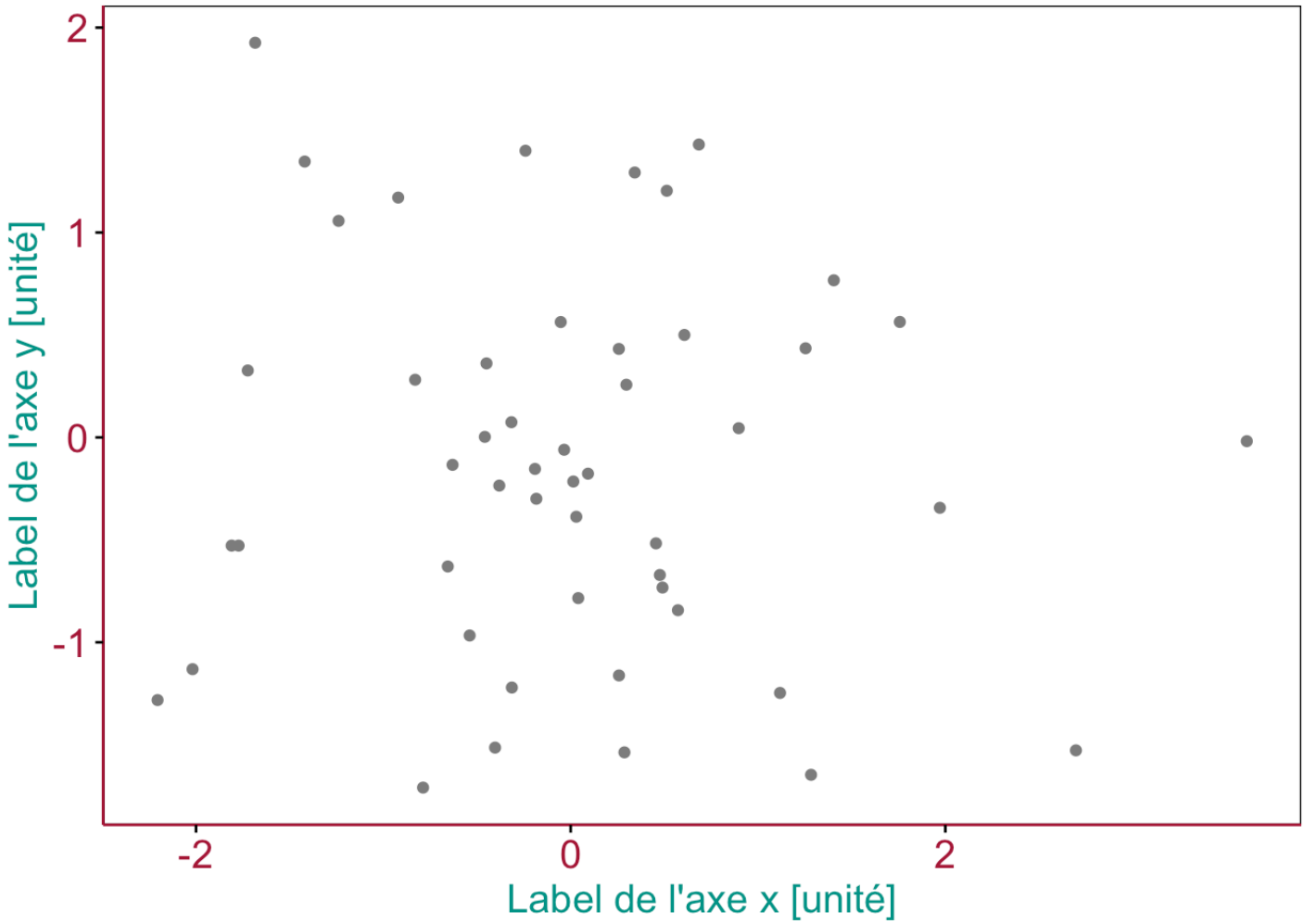


Figure 1.1: Exemple de graphique en nuage de points. Des éléments essentiels sont ici mis en évidence en couleurs (voir texte).

Les éléments indispensables à la compréhension d'un graphique en nuage de points sont mis en évidence à la Fig. 1.1 :

- Les axes avec les graduations (en rouge),
- Les labels et les unités des axes (en bleu).
- Les points représentant les différentes observations (en gris).

Les instructions dans R pour produire un tel graphique en nuage de points sont les suivantes :

- Préparation de l'environnement, importation et visualisation des données sous forme de tableau synthétique pour commencer.

```
# Chargement de SciViews::R
SciViews::R
# Importation du jeu de données
urchin <- read("urchin_bio", package = "data.io", lang = "fr")
# Visualisation des premières et dernières lignes du jeu de données
tabularise$headtail(urchin, auto.labs = FALSE)
```

origin	diameter1	diameter2	height	buoyant_weight	weight	solid_parts	integuments
Fishery	9.9	10.2	5.0		0.5215	0.4777	0.3658
Fishery	10.5	10.6	5.7		0.6418	0.5891	0.4447
Fishery	10.8	10.8	5.2		0.7336	0.6770	0.5326
Fishery	9.6	9.3	4.6		0.3697	0.3438	0.2661
Fishery	10.4	10.7	4.8		0.6097	0.5587	0.4058
...	...	...	...	...	...	...	...
Farm	16.7	17.2	8.5	0.5674	2.4300	2.2900	1.8400
Farm	16.5	16.5	7.9	0.5472	2.3200	2.1800	1.8000
Farm	16.8	16.7	8.2	0.4864	2.2200	2.1300	1.6300
Farm	17.3	17.2	8.5	0.4864	2.5200	2.3400	1.7200
Farm	17.0	16.6	7.9	0.4357	2.0500	1.9800	1.4300

First and last 5 rows of a total of 421

- Réalisation du graphique en nuage de points proprement dit.

```
chart(data = urchin, height ~ weight) +
  geom_point()
```

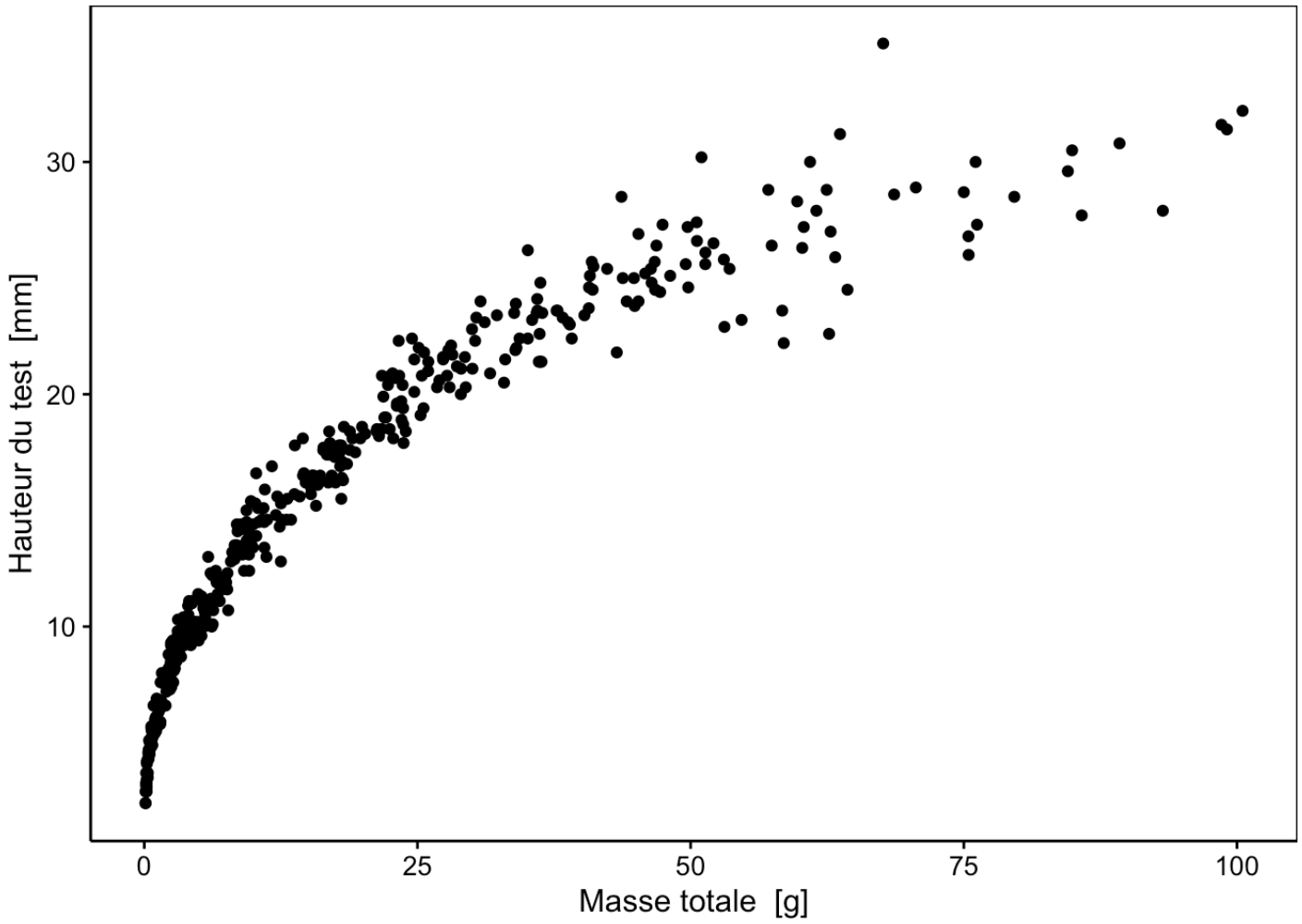
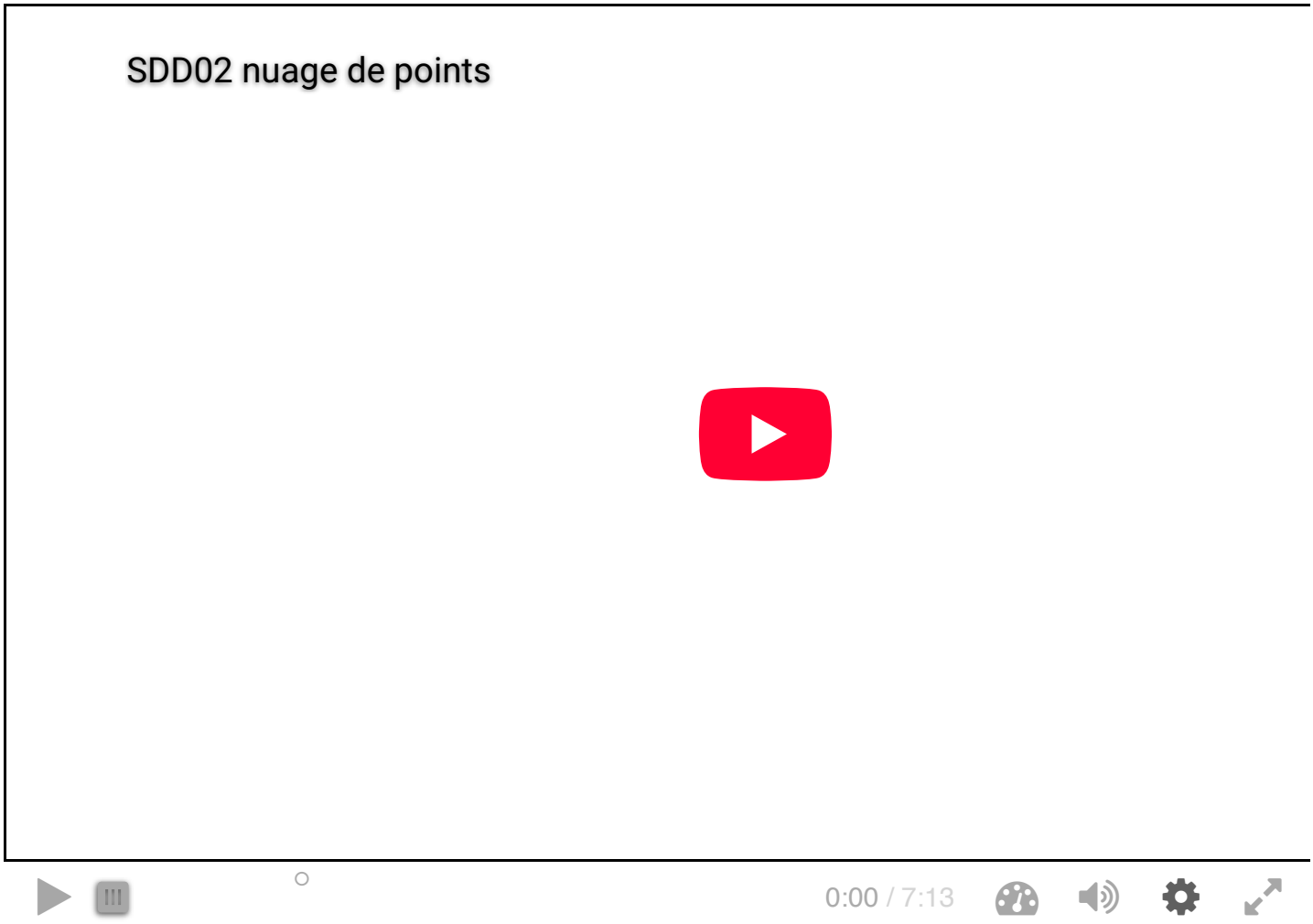


Figure 1.2: Taille (hauteur du test) d'oursins en fonction de leur masse.

La fonction `chart()` n'est pas accessible dans R de base, mais l'extension chargée grâce à l'instruction `SciViews::R` rend cette fonction disponible. Elle requiert comme argument le jeu de donnée (`data = urchin` , c'est un objet qui s'appelle un **"data frame"** dans le langage de R), ainsi que la formule à employer dans laquelle vous avez indiqué le nom des variables que vous voulez sur l'axe des ordonnées à gauche et des abscisses à droite de la formule, les deux parties étant séparées par un "tilde" (`~`). Vous voyez que le jeu de données contient beaucoup de variables (les titres des colonnes du tableau en sortie). Parmi toutes ces variables, nous avons choisi ici de représenter `height` en fonction de `weight` , la hauteur en fonction de la masse des oursins. Jusqu'ici, nous avons spécifié *ce que* nous voulons représenter, mais pas encore *comment* (sous quelle apparence), nous voulons les matérialiser sur le graphique. Pour un nuage de points, nous voulons les représenter sous forme de … points ! Donc, nous devons ajouter la fonction `geom_point()` pour indiquer cela.

À vous de jouer !

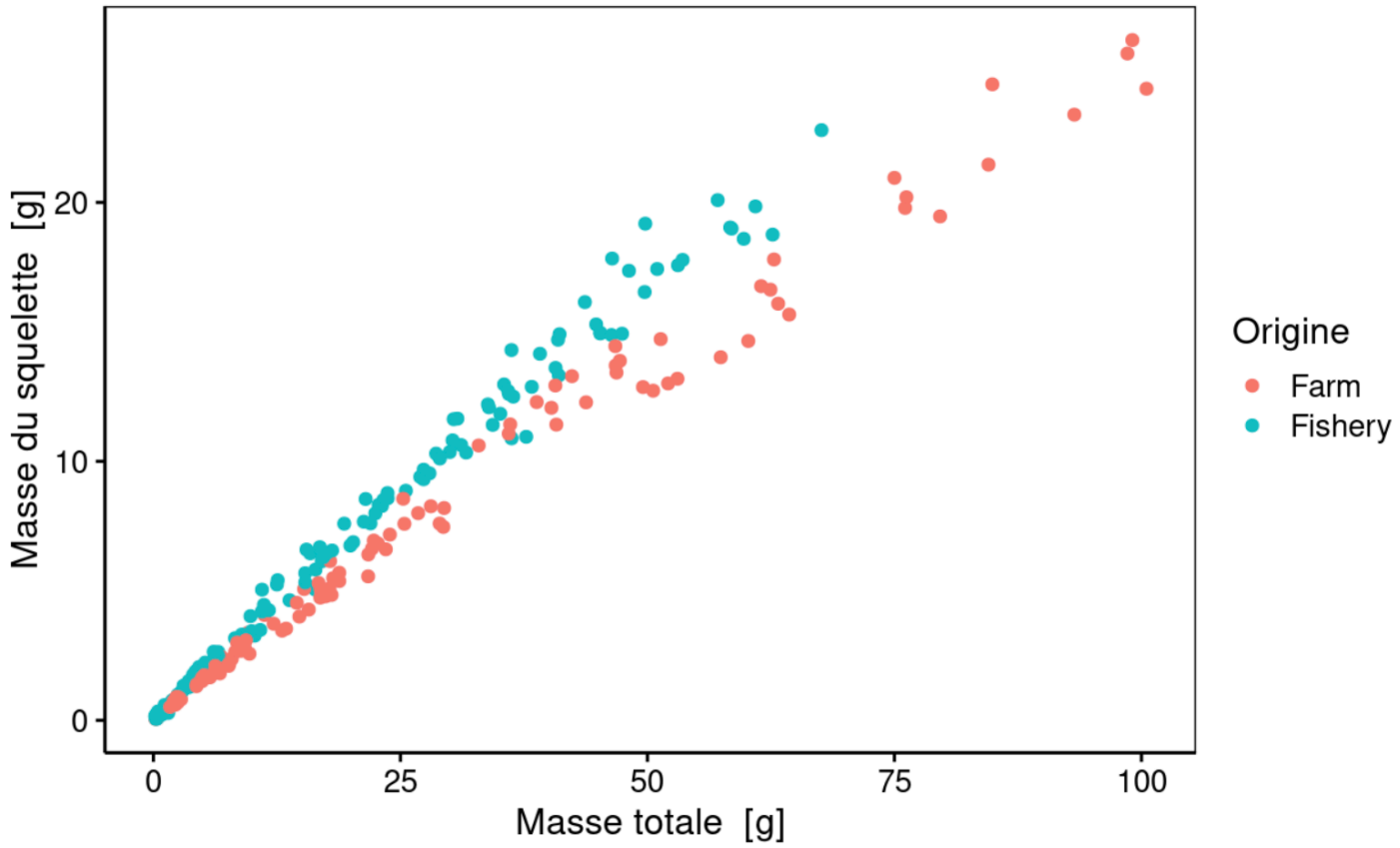
Note : la vidéo ci-dessous vous expliquant la création du nuage de points dans R sur le jeu de données `urchin` est affublée du signe “H5P”, ce qui signifie qu’elle contient également des questions auxquelles vous devez répondre dans le cadre de votre progression dans la matière.



Autoévaluez maintenant vos acquis avec le tutoriel learnr suivant.

Effectuez maintenant les exercices du tutoriel **A01Lb_scatterplot (Nuage de points)**.

```
BioDataScience1::run("A01Lb_scatterplot")
```



Complétez les instructions ci-dessous pour obtenir le graphique ci-dessus.

chart(data = urchin, skeleton ~ weight %col=% ✔ origin) +
geom_point(na.rm = TRUE)

You got 1 of 1 blanks correct.

1/1

1.2.1 Échelles de graphiques

Vous devez être particulièrement vigilant lors de la réalisation d'un graphique en nuage de point sur l'étendue des valeurs présentées sur vos axes. Vous devez utiliser votre expertise de biologiste pour vous poser les deux questions suivantes :

- Est-ce que l'axe représente des valeurs plausibles de hauteurs et de masses de ces oursins (sachant que l'espèce est *Paracentrotus lividus* vous pouvez faire une recherche pour vérifier les tailles et masses de ces animaux) ?
- Quelle est la précision des mesures ?

Par défaut, `chart()` va calculer l'étendue des axes en tenant compte de la plus petite et de la plus grande valeur observée. Dans certains cas, lorsqu'une ou plusieurs observations s'éloignent très fort du nuage de points (par exemple, une valeur mal encodée avec un zéro de trop à la fin du nombre), le graphique va compresser le nuage de points à cause de la présence de ces valeurs aberrantes. Ce n'est pas le cas ici, mais nous pouvons le simuler en distendant artificiellement soit l'axe X, soit l'axe Y, soit les deux :

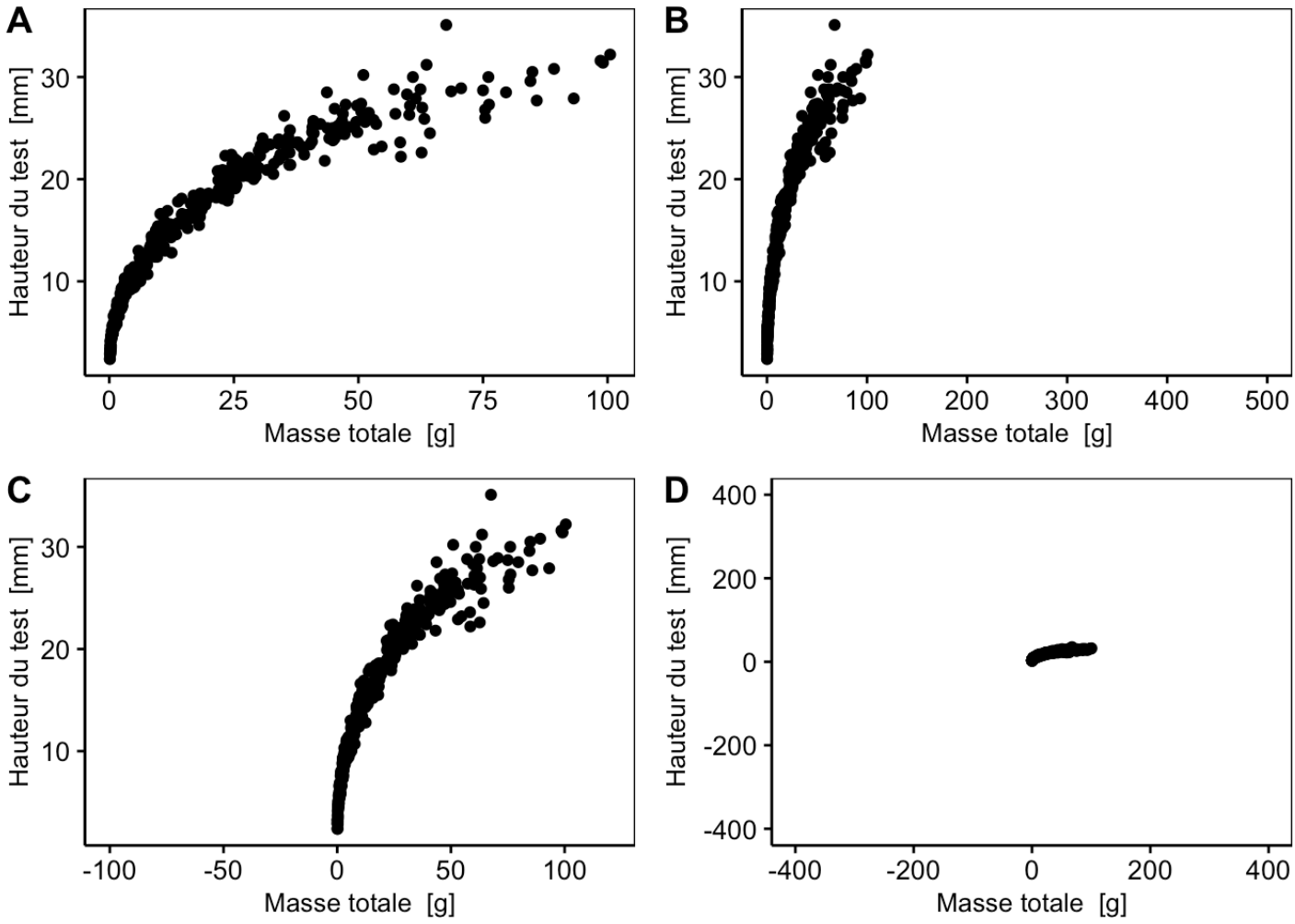
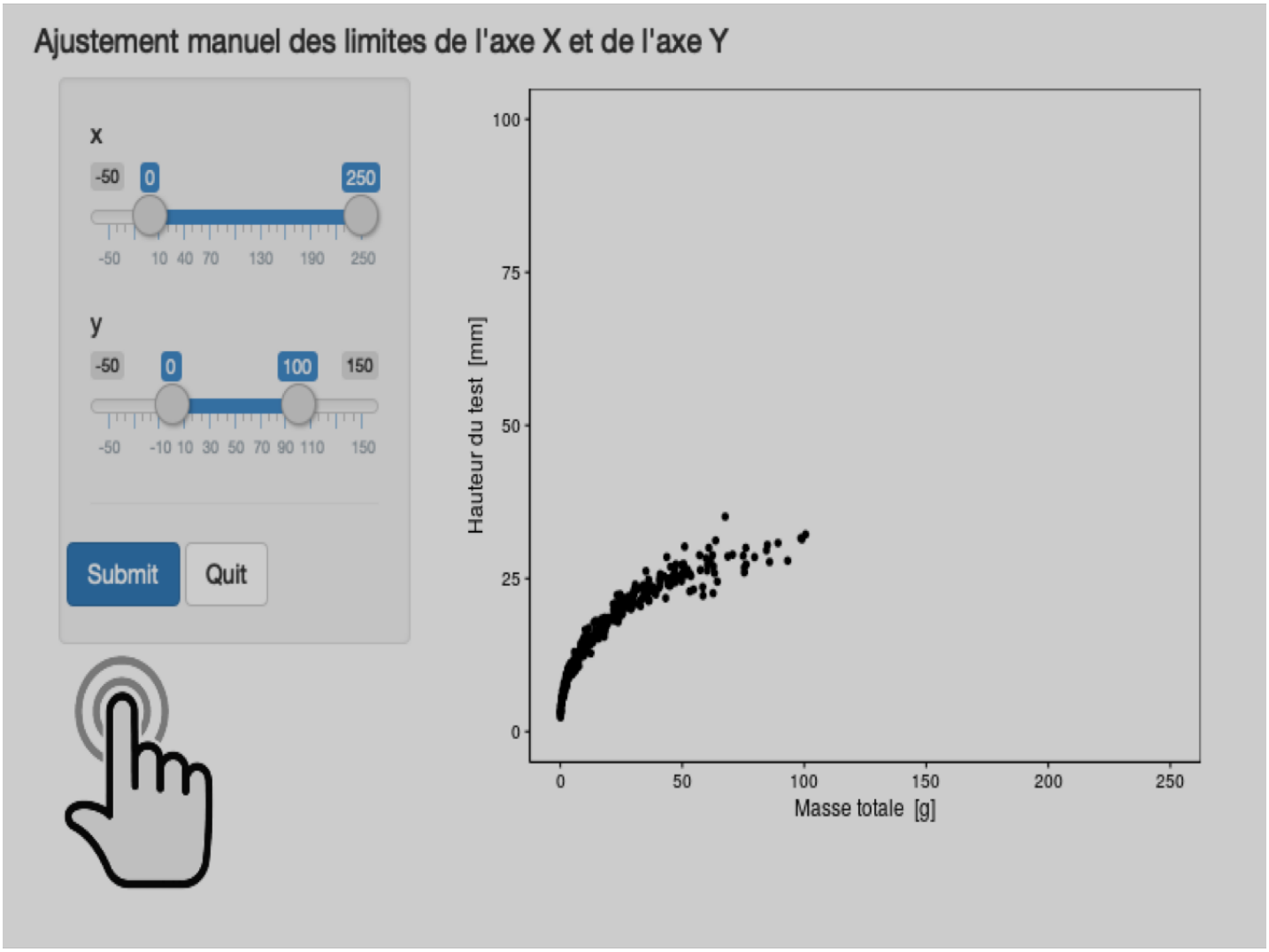


Figure 1.3: Piège du nuage de points. A) Graphique initial montrant la variation de la hauteur [mm] en fonction de la masse [g]. B) Graphique A avec la modification de l'échelle de l'axe X. C) Graphique A avec une seconde modification de l'axe X. D) Graphique A avec modification simultanée des deux axes.

Vous voyez que l'effet peut éventuellement être assez catastrophique. Faites donc bien attention à la façon dont les points se répartissent et remplissent ou non le graphique pour éviter de tomber dans ce piège !

À vous de jouer !

*Note : l'exercice suivant est une **application Shiny**. Il s'agit d'un petit programme écrit en R et qui vise à expérimenter un concept de manière interactive. Vous pouvez lancer cet exercice directement dans cette page. Vérifiez d'être bien enregistré (message à l'ouverture de l'application) et n'oubliez pas de cliquer sur le bouton `Submit` pour tester votre choix, mais également sur le bouton `Save & Quit` pour valider définitivement votre réponse pour la prendre en compte dans votre rapport de progression.*



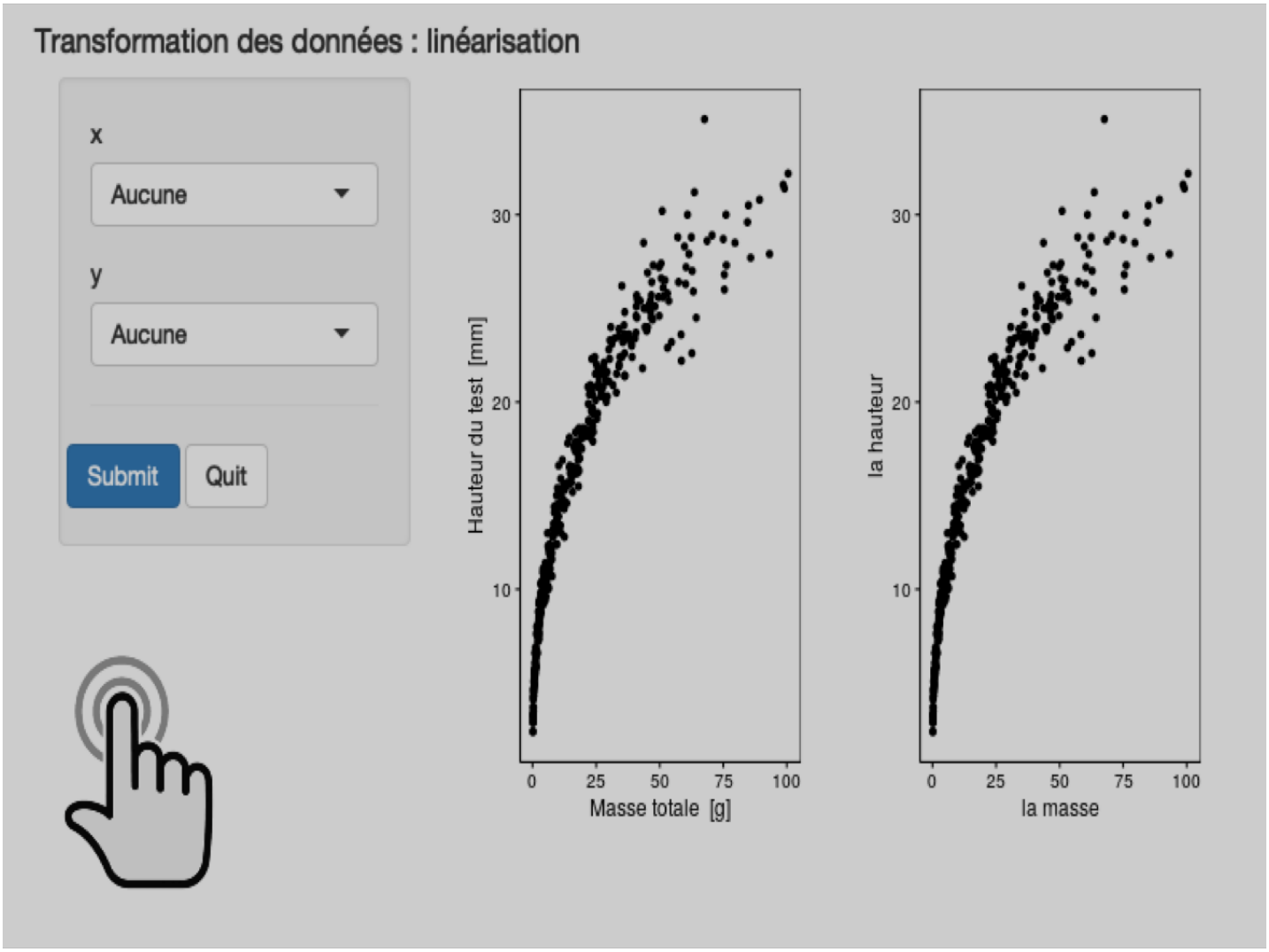
Cliquez pour lancer ou exécutez dans RStudio

```
BioDataScience1::run_app("A01Sa_limits") .
```

1.2.2 Transformation des données

Vous avez la possibilité d'appliquer une transformation à vos données (il est même conseillé de le faire) pour qu'elles soient plus facilement analysables ensuite. Par exemple, il est possible d'utiliser des fonctions de puissance, racines, logarithmes, exponentielles[4] pour modifier l'apparence du nuage de points dans le but de le rendre plus linéaire. Il est, en effet, plus facile d'analyser statistiquement des données qui s'alignent le long d'une droite.

À vous de jouer !



Cliquez pour lancer ou exécutez dans RStudio

```
BioDataScience1::run_app("A01Sb_transformation") .
```

Pièges et astuces

RStudio permet de récupérer rapidement des instructions à partir d'une banque de solutions toutes prêtes. Cela s'appelle des **snippets**. Vous avez une série de snippets disponibles dans la SciViews Box. Celui qui vous permet de réaliser un graphique en nuage de points s'appelle `.cbxy` (pour **c**hart -> **b**ivariate -> **xy**-plot). Entrez ce code et appuyez ensuite sur la tabulation dans un document de type script R, et vous verrez le code remplacé par ceci dans la fenêtre d'édition :

```
chart(data = DF, YNUM ~ XNUM) +
  geom_point()
```

Vous avez à votre disposition un ensemble de snippets que vous pouvez retrouver dans l'aide-mémoire consacré à **SciViews**. Vous avez également à votre disposition l'aide-mémoire sur la visualisation des données (**Data Visualization Cheat Sheet**) qui utilise la fonction `ggplot()` plutôt que `chart()` et une interface légèrement différente pour spécifier les variables à utiliser pour réaliser le graphique (`aes(x = ..., y = ...)`).

Prêtez une attention toute particulière à l'organisation d'un script R. En plus des instructions R, il contient aussi sous forme de commentaires, un titre, la date de la dernière mise à jour, le nom de l'auteur, et des sections qui organisent de façon claire le contenu du script. À ce sujet, vous trouverez des explications détaillées concernant l'utilisation des scripts R dans l'annexe B.1.2.

3. Dans R, une **formule** permet de spécifier les variables avec lesquelles on souhaite travailler, et leur rôle. Par exemple ici, la variable *x* sur l'axe des abscisses et la variable *y* sur l'axe des ordonnées. ↩
4. Pour les proportions (`prop`) ou les pourcentages (`perc`) (valeurs bornées entre 0 et 1 ou 0 et 100%) la transformation arc-sinus est souvent utilisée dans la littérature $prop' = \arcsin \sqrt{prop}$ ou $perc' = \arcsin \sqrt{perc/100}$. Son usage est cependant soumis à la critique et tous les statisticiens ne sont pas d'accord au sujet de cette transformation. ↩

1.3 Premier projet

Passons maintenant à la pratique. Lisez la suite de ce module lorsque vous le préparez d'avance chez vous. Cependant, nous réaliserons le projet suivant sur le nuage de points ensemble en présentiel.

1.3.1 GitHub Classroom

GitHub Classroom est une extension de GitHub. Rappelez-vous, l'une des premières étapes que vous avez réalisées dans le cadre de ce cours a été de créer votre compte GitHub. Grâce à GitHub, vous allez pouvoir sauvegarder vos travaux dans le Cloud, les partager et collaborer.

GitHub Classroom facilite le travail dans le contexte d'exercices de niveau 3 ou de niveau 4 (les projets individuels et en groupes) à réaliser dans le cadre du cours. Vous serez donc amenés à créer et éditer des projets issus de GitHub Classroom pour réaliser vos exercices. **Vos projets seront privés**. Seuls vous-mêmes et vos enseignants aurez accès à ces projets, mais vous pourrez aussi les rendre publics, si vous voulez *valoriser votre travail* de manière plus large.

L'illustration ci-dessous vous montre la boîte d'information qui vous permet d'initier votre projet. Cette boîte se trouve à la fin de ce module 1. Soyez vigilant quant à la date limite pour terminer le projet.

À vous de jouer !

Vous allez maintenant manipuler un document **Quarto** pour réaliser des graphiques en nuage de points.

Réalisez le travail **A01Ia_scatterplot**.

Travail individuel pour les étudiants inscrits au cours de Science des Données Biologiques I : visualisation à l'UMONS à terminer avant le 2024-10-14 23:59:59.

Initiez votre projet GitHub Classroom

Voyez les explications dans le fichier `README.md` .

Attention, si vous n'êtes pas reconnu, vous ne pourrez pas initier votre projet et vous serez renvoyé vers une page comme celle ci-dessous. Vous devrez alors vérifier votre identification dans le site du cours (page d'accueil accessible à partir de l'entrée de menu "Home" en haut de page) avant de pouvoir accéder à votre projet.

A01Ia_scatterplot

Vous ne semblez pas être inscrit dans un cours lié à ce travail. Si cela est une erreur, revenez à la page principale et vérifiez votre identification. Si **et seulement si** vous n'êtes pas un étudiant inscrit à ce cours, vous pouvez réaliser quand même cet exercice en **effectuant un "fork"** sur votre propre compte GitHub de ce template.

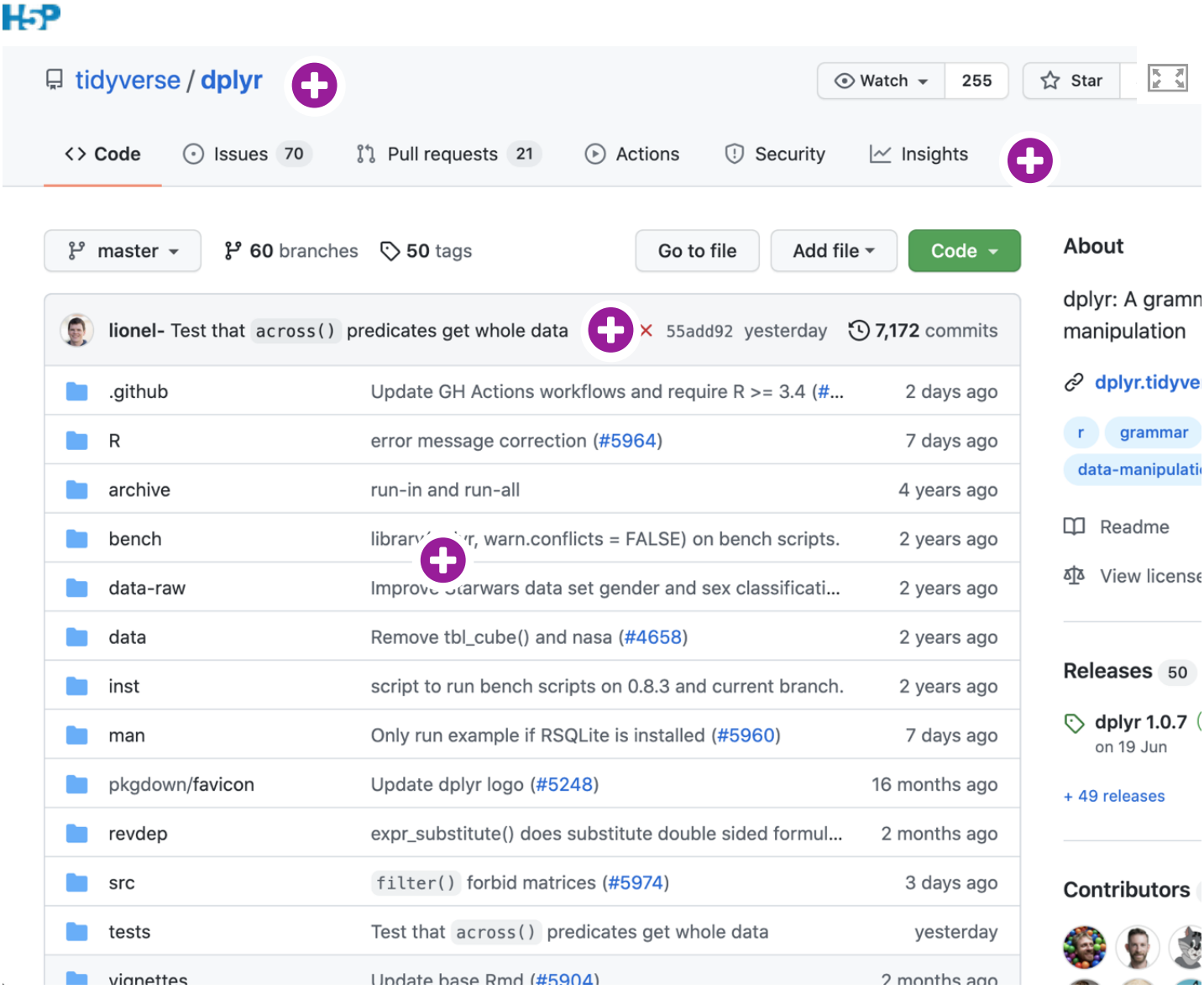
1.3.2 GitHub

GitHub est un réseau social permettant de sauvegarder vos projets dans le "Cloud", les partager et collaborer avec d'autres personnes. Les utilisateurs peuvent se regrouper sous une organisation afin de faciliter la collaboration. Ce réseau social a la particularité d'être centré sur les projets et utilise un gestionnaire de version de projet nommé Git (nous aborderons Git plus loin dans ce module). Ce nom provient de l'association "Git", le gestionnaire de version et "Hub" relatif au réseau.

Découvrons un projet qui s'intéresse à la manipulation de tableau de données (filtrer des lignes spécifiques, calculer de nouvelles colonnes, trier le tableau en fonction d'une colonne spécifique…). Nous utiliserons ce projet dans le cours. Le projet dplyr est partagé par l'organisation tidyverse. Il va nous permettre d'aborder quelques aspects de GitHub. Nous nous attardons uniquement sur les éléments clés dans le cadre du cours.

À vous de jouer !

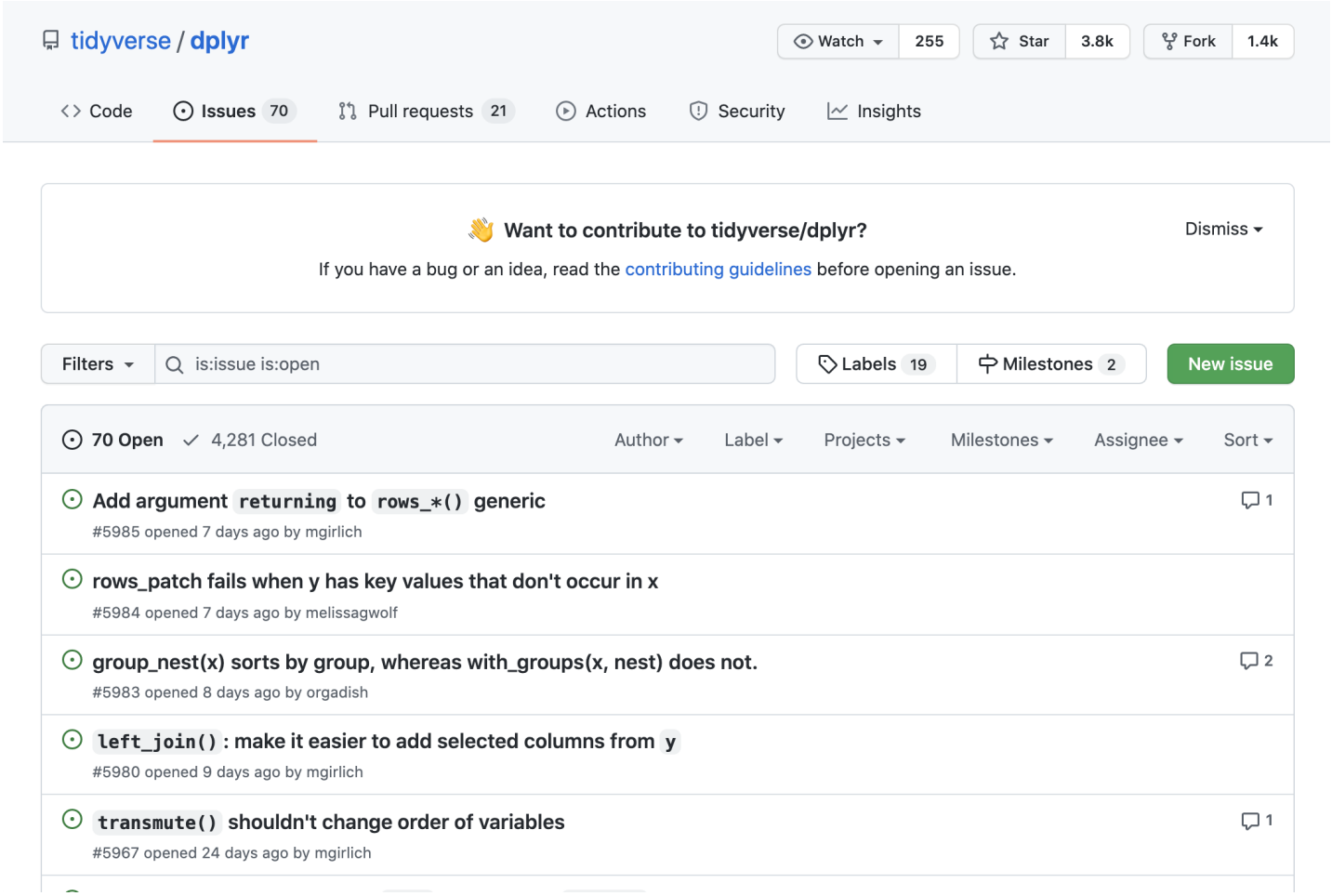
Cliquez sur les symboles + pour découvrir les parties importantes de la page principale d'un projet GitHub



Ce projet héberge une multitude de fichiers que vous ne devez pas comprendre pour le moment ;). 7112 "commits" y ont été réalisés (au moment où la capture d'écran a été faite : le projet a évolué depuis et les nombres cités ne sont certainement plus d'actualité

par rapport au site aujourd'hui). Pour le moment, retenez qu'un "commit" est un état de sauvegarde du projet. Ce projet traite de la manipulation de données. Un site Web lui est associé dplyr.tidyverse.org pour en expliquer le contenu. Un grand nombre de contributeurs ont participé à ce projet (plus de 240). Un fichier `README.md` sert de page de présentation.

En haut de la page, on a les différentes sections du site : `Code` , `Issues` , `Pull requests` , `Actions` , `Security` et `Insights` (elles ne seront pas toutes détaillées ici).



Dépôt dplyr de l'organisation tidyverse dans GitHub

Lors d'un travail en équipe, vous allez vous poser des questions et avoir envie d'échanger vos idées. GitHub met à votre disposition un espace dédié à la discussion et à la collaboration. Il s'agit des `Issues` . La section `Issues` permet de mettre en avant un problème ou une idée dans le but d'améliorer ce projet. Une issue est un espace de discussion et de collaboration centré autour d'une question. Par exemple, le projet dplyr a 70 issues ouvertes et 4281 issues fermées. On peut en déduire que 70 idées ou bogues sont en cours de correction ou de réflexion et que 4281 sont considérés comme terminés. Ce projet est assez dynamique.

Quand vous aurez un problème ou une idée, utilisez donc les issues pour discuter avec vos encadrants ou vos collaborateurs. Vous y accédez à partir de bandeau rouge au-dessus de chaque page du cours.

À vous de jouer !

Testez vos nouvelles connaissances en répondant à la question ci-dessous.

Explorez le lien suivant : sdd-umons-2020. Ensuite, déplacez les textes dans les emplacements qui leur correspondent.

Le projet est associé à l'organisation . Plus de 400 y ont été faits. Il héberge les fichiers générant le du cours de sciences de données I pour l'année académique 2020-2021. Toutes les sont fermées.

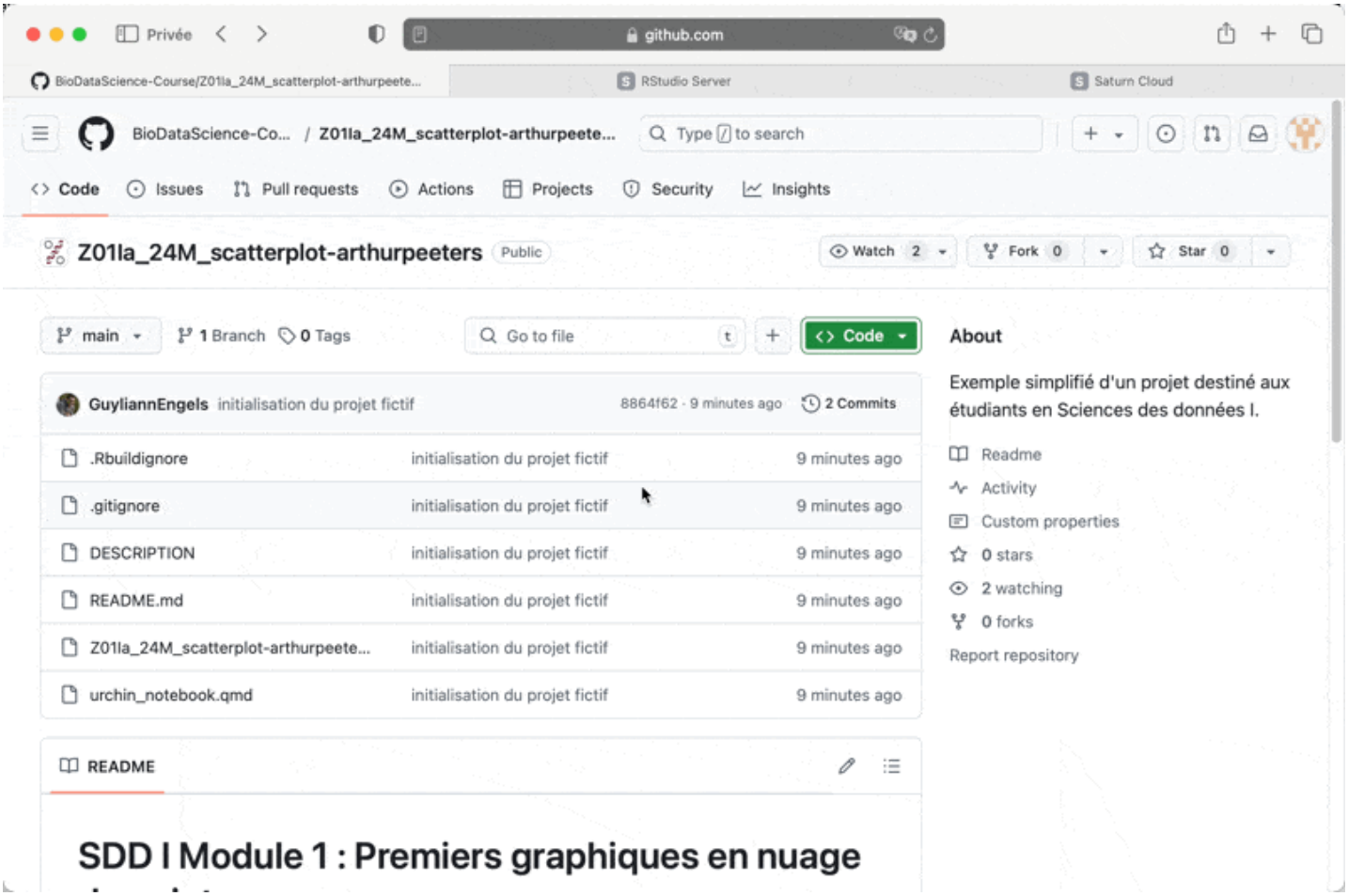
- sdd-umons-2020
- issues
- BioDataScience-Course
- commits
- livre en ligne

Vérifier

GitHub n'est pas le seul réseau social centré sur Git. D'autres réseaux équivalents existent comme Gitlab ou Bitbucket. Cependant, nous utiliserons GitHub ensemble, sachant que les notions apprises ici seront réutilisables ailleurs.

1.3.3 Projet RStudio

Démarrez votre SciViews box dans Saturn Cloud et affichez la fenêtre de RStudio maintenant (si vous ne l'avez pas encore installée, cliquez sur le bouton bleu **RStudio** en haut à droite de cette page pour voir les explications d'installation). RStudio permet de cloner (faire une copie locale sur votre ordinateur) directement un projet disponible sur GitHub en quelques étapes comme ci-dessous.



Des explications détaillées se trouvent dans l'annexe B.2.5 qui présente deux méthodes différentes pour cloner votre projet GitHub dans RStudio. Vous allez terminer ce module en réalisant votre premier projet individuel en science des données.

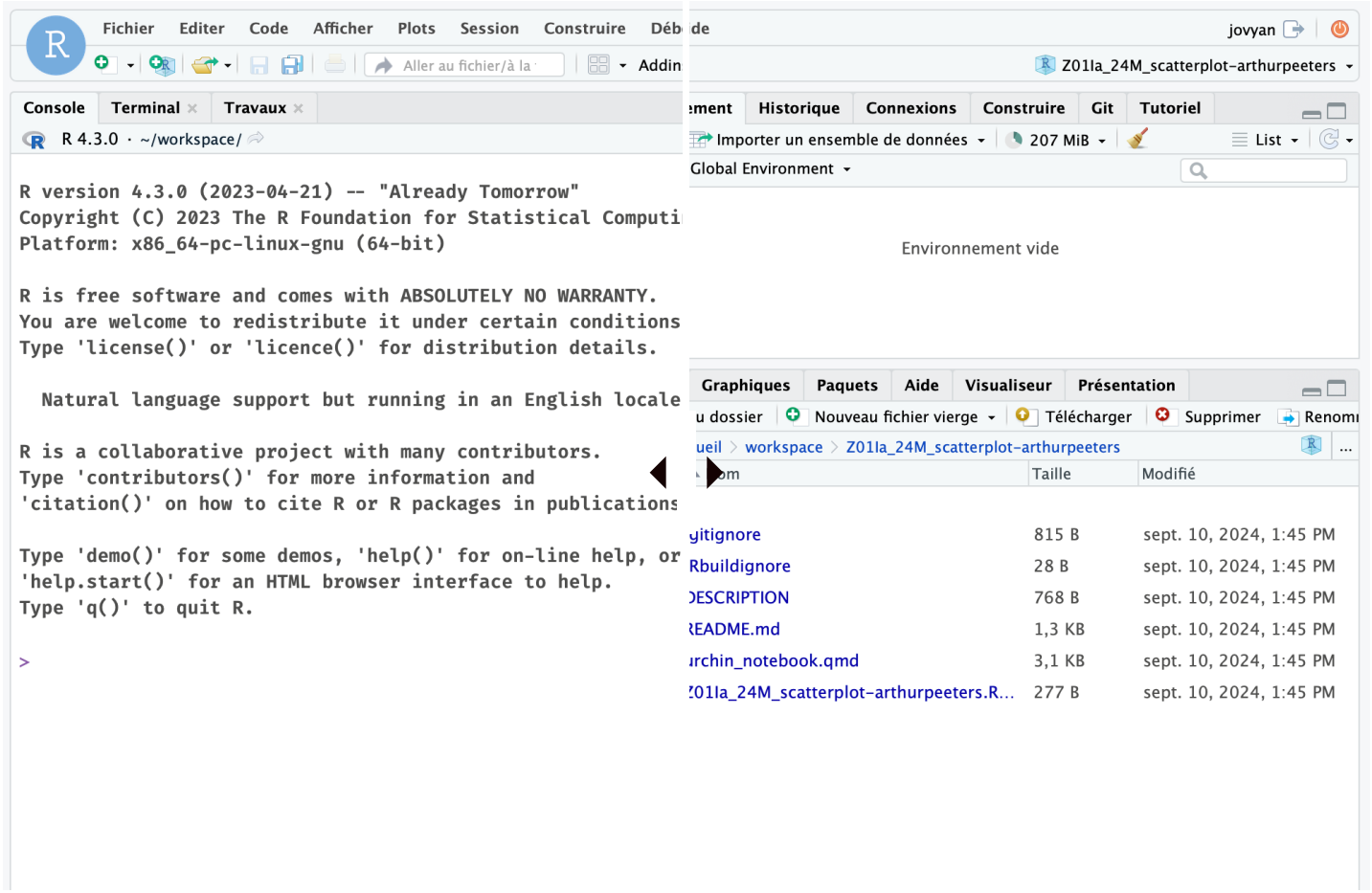
RStudio permet de gérer des projets efficacement. Un projet va regrouper l'ensemble des jeux de données, des blocs-notes, des rapports, des présentations, des scripts d'une analyse généralement en relation avec une ou plusieurs expériences ou observations réalisées sur le terrain ou en laboratoire. Il est normal que tous les éléments d'un projet ne vous soient pas encore familiers. Nous allons découvrir cela tout au long du cours de science des données.

À vous de jouer !

Observez les deux images et tentez de repérer chaque différence entre les deux interfaces, l'une hors projet, et l'autre dans un projet nommé `Z01Ia_24M_scatterplot-arthurpeeters`. Tentez de trouver les *quatre différences* avant de lire la section suivante.

Déplacez le curseur de gauche à droite pour comparer l'interface de RStudio hors projet (à gauche) et avec un projet ouvert (à droite).

Les projets dans RStudio



Aucun projet — Mon projet

1. Dans la barre d'outils supérieure, on passe de `Projet (Aucun)` vers `Z01Ia_24M_scatterplot-arthurpeeters`. On peut donc facilement repérer si l'on est dans un projet ou non, et si oui, quel est son nom.

2. Le panneau en haut à droite change et de nouveaux onglets sont présents, dont un nommé `Git` qui vient s'ajouter, mais uniquement si les versions du projet sont gérées par Git (que nous découvrirons dans la suite de ce module).

3. Le panneau en bas à droite contenant l'onglet `Fichiers` change de dossier pour afficher le dossier de base du projet. Dans ce dossier, nous retrouvons obligatoirement un fichier à l'extension `.Rproj` ainsi qu'éventuellement un second, `.gitignore`. D'autres fichiers sont présents que nous découvrirons par la suite.

- `Z01Ia_24M_scatterplot-arthurpeeters.Rproj` : À la base d'un projet RStudio, on retrouve un fichier à l'extension `.Rproj`. Ce fichier est placé automatiquement par RStudio. Il contient les paramètres de configuration de votre projet. Il ne faut pas le modifier soi-même. Il est utile aussi pour repérer tout de suite que l'on se trouve dans un dossier de base d'un projet RStudio.
- `.gitignore` : Ce fichier permet de spécifier les fichiers que l'on souhaite exclure du gestionnaire de version, par exemple, les gros fichiers de jeux de données ou bien des documents.

1.3.4 Gestionnaire de version

Nous nous sommes intéressés à la partie collaborative de GitHub. Nous avons exploré la structure d'un projet et les outils de discussions que sont les issues. Nous allons maintenant nous intéresser à Git, le gestionnaire de version de projet utilisé par GitHub et par RStudio.

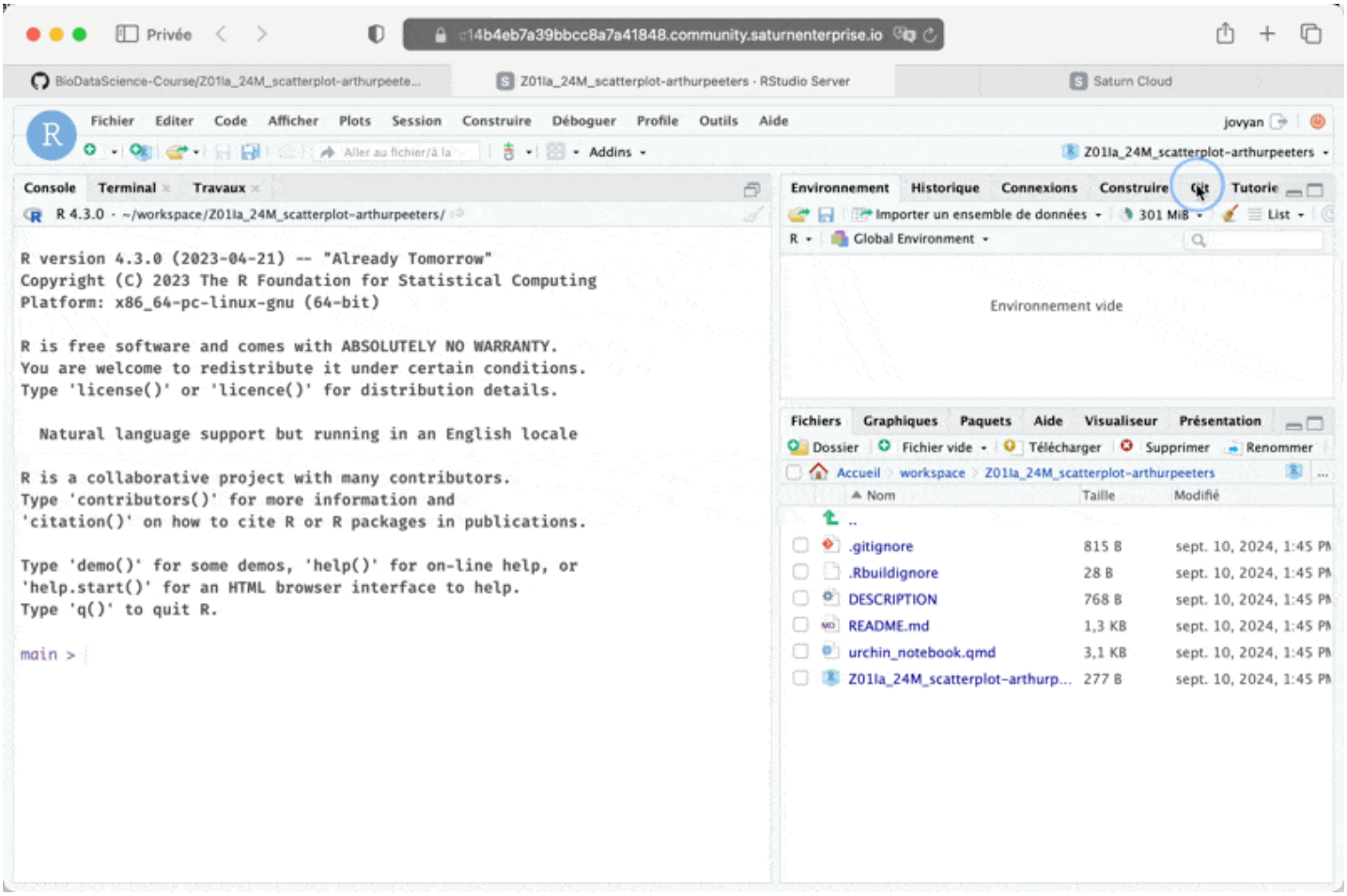
Lors de la rédaction d'un court document ou de travaux un petit peu conséquents, comme un travail de fin d'études, une publication scientifique ou un rapport volumineux, on se retrouve rapidement avec plusieurs fichiers correspondants à des états d'avancements du travail :

- TFE_final
- TFE_final1
- TFE_final2
- TFE_final3
- TFE_final…
- TFE_final99

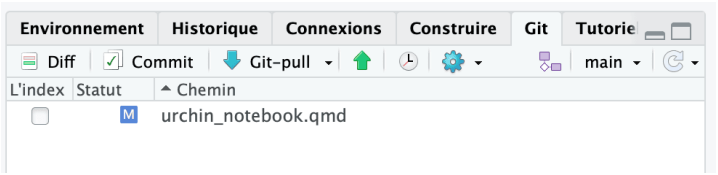
On aura tendance à tout garder dans différents fichiers afin de ne rien perdre d'important. Cette pratique, bien que très courante, comporte le gros désavantage de prendre énormément de place sur le disque de votre ordinateur et de n'être pas pratique. Les questions suivantes peuvent se poser :

Que se cache-t-il dans la version TFE_finalX ? Après un mois sans travailler sur le fichier, seriez-vous encore capable de faire facilement la différence entre TFE_final2 et TFE_final3 ?

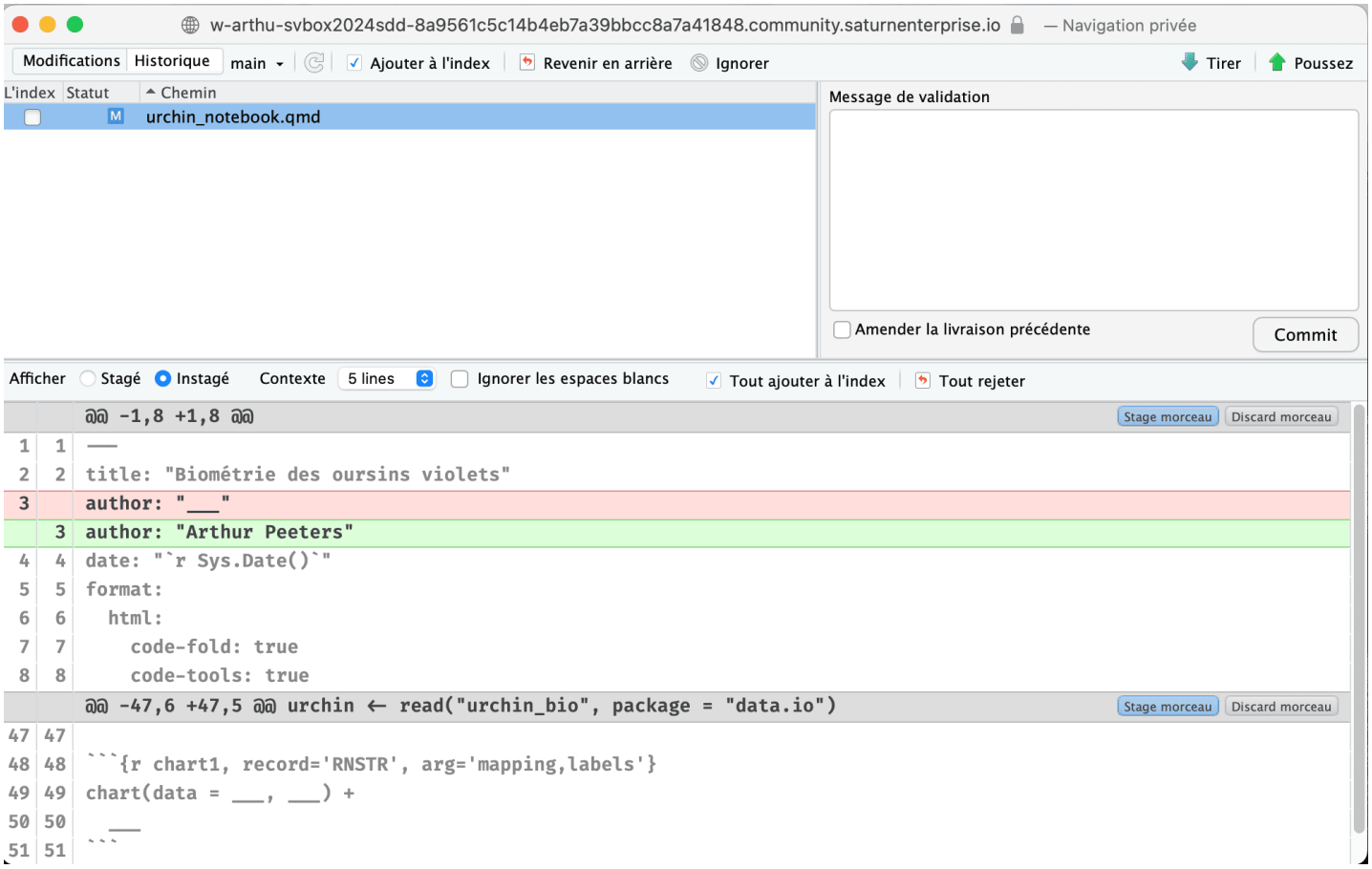
Un logiciel comme Git vous offre une solution professionnelle pour gérer vos fichiers au sein d'un projet. Pour que vous vous fassiez une idée plus précise, observez le gif animé juste en dessous. Un fichier `urchin_notebook.qmd` est ouvert dans la zone d'édition (en haut à gauche) à partir de l'onglet `Fichiers` (en bas à droite). Le nom de l'auteur est ensuite inséré dans le document. Une fois le document sauvegardé en cliquant l'icône bleue dans la barre d'outils d'édition, on retrouve dans l'onglet Git le nom du fichier modifié. Le gestionnaire de version va suivre toutes les modifications apportées aux fichiers au sein du projet (matérialisé par un dossier dans votre machine). Ce dossier se nomme un dépôt (*repository* en anglais).



Si vous sélectionnez l'onglet Git, vous verrez une barre d'outils en haut du panneau correspondant. Chaque bouton de cette barre d'outils donne accès à des fonctionnalités de Git ou de GitHub. Nous allons en découvrir une partie à présent.



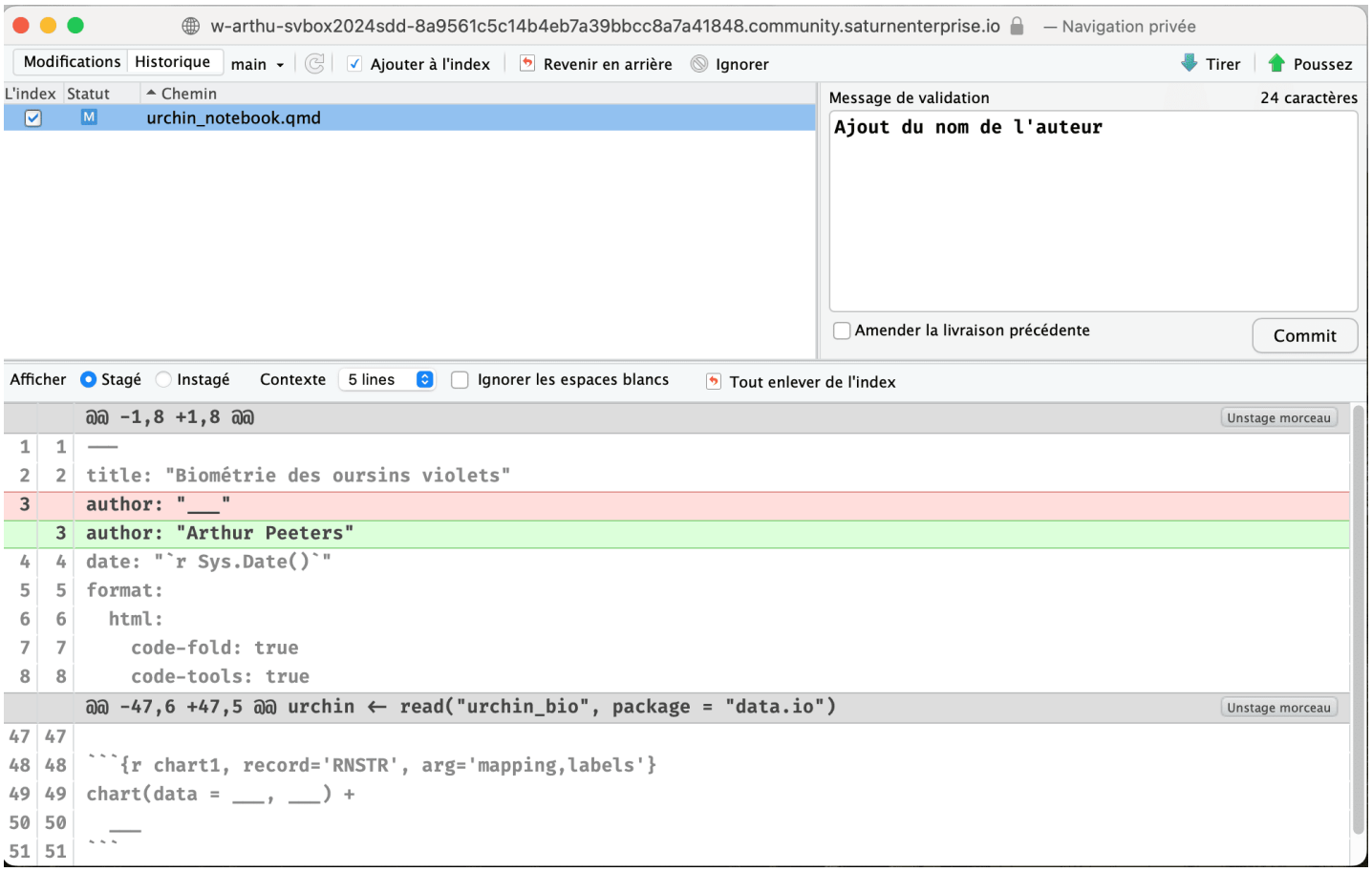
Les deux premiers boutons, `Diff` et `Commit`, ouvrent la même nouvelle fenêtre destinée à l'utilisation de Git/GitHub.



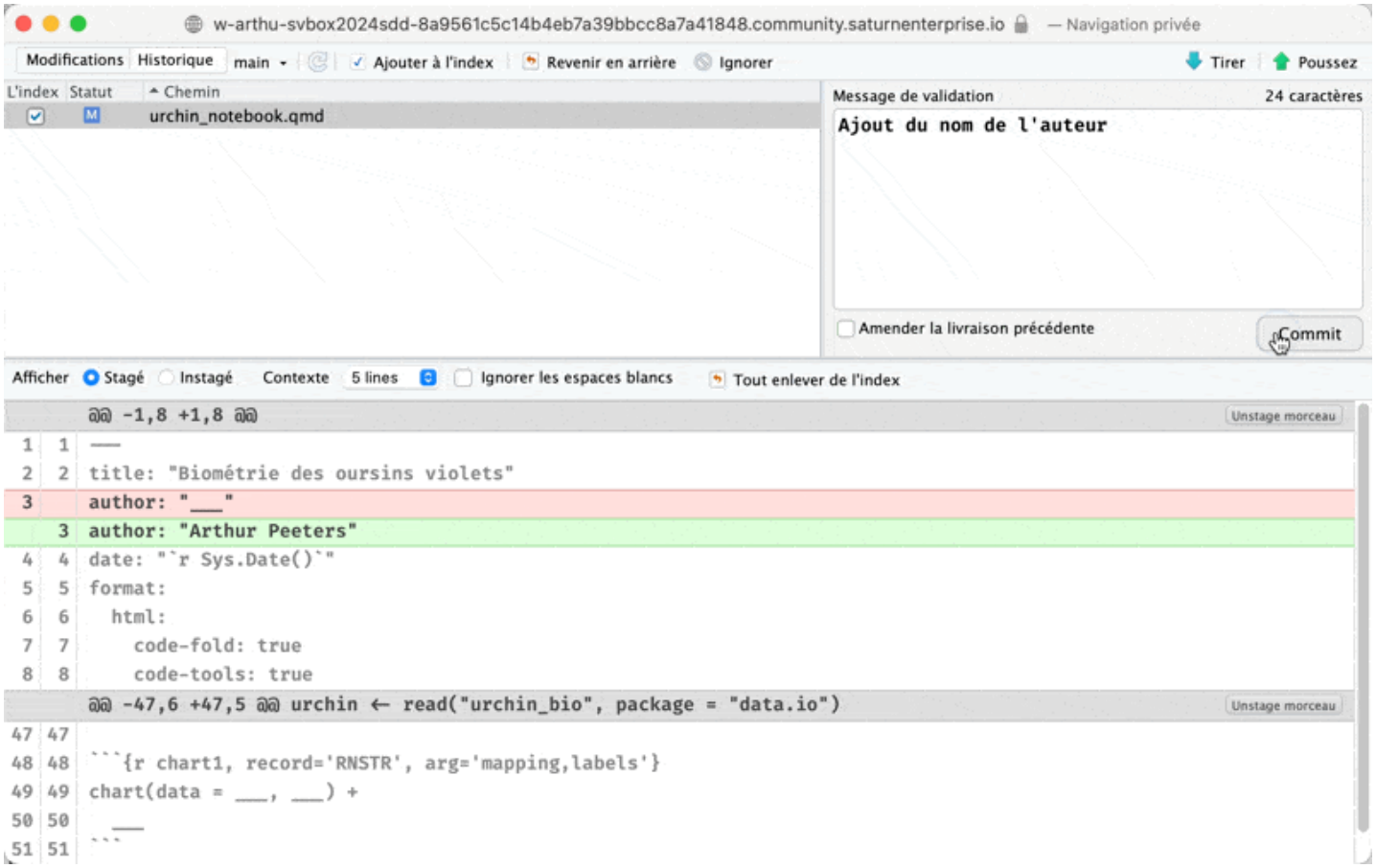
En bas de cette fenêtre, nous allons suivre les modifications apportées à nos documents. Dans notre exemple, il n'y en a qu'un seul : le nom d'auteur en ligne 3 de `urchin_notebook.qmd` (en vert pour les parties de texte ajoutées) à la place de trois tirets (en rouge pour les éléments supprimés ou remplacés). La liste des fichiers modifiés se trouve en haut à gauche de la fenêtre (un seul également). Comme vous pouvez vous en rendre compte, Git permet de suivre finement chaque modification d'un document.

Le logiciel Git permet de faire des points de sauvegarde de votre projet. Cela s'appelle réaliser un **commit**. Pour ce faire, vous devez indexer les fichiers que vous souhaitez sauvegarder. Cette étape se réalise en sélectionnant le fichier `urchin_notebook.qmd`

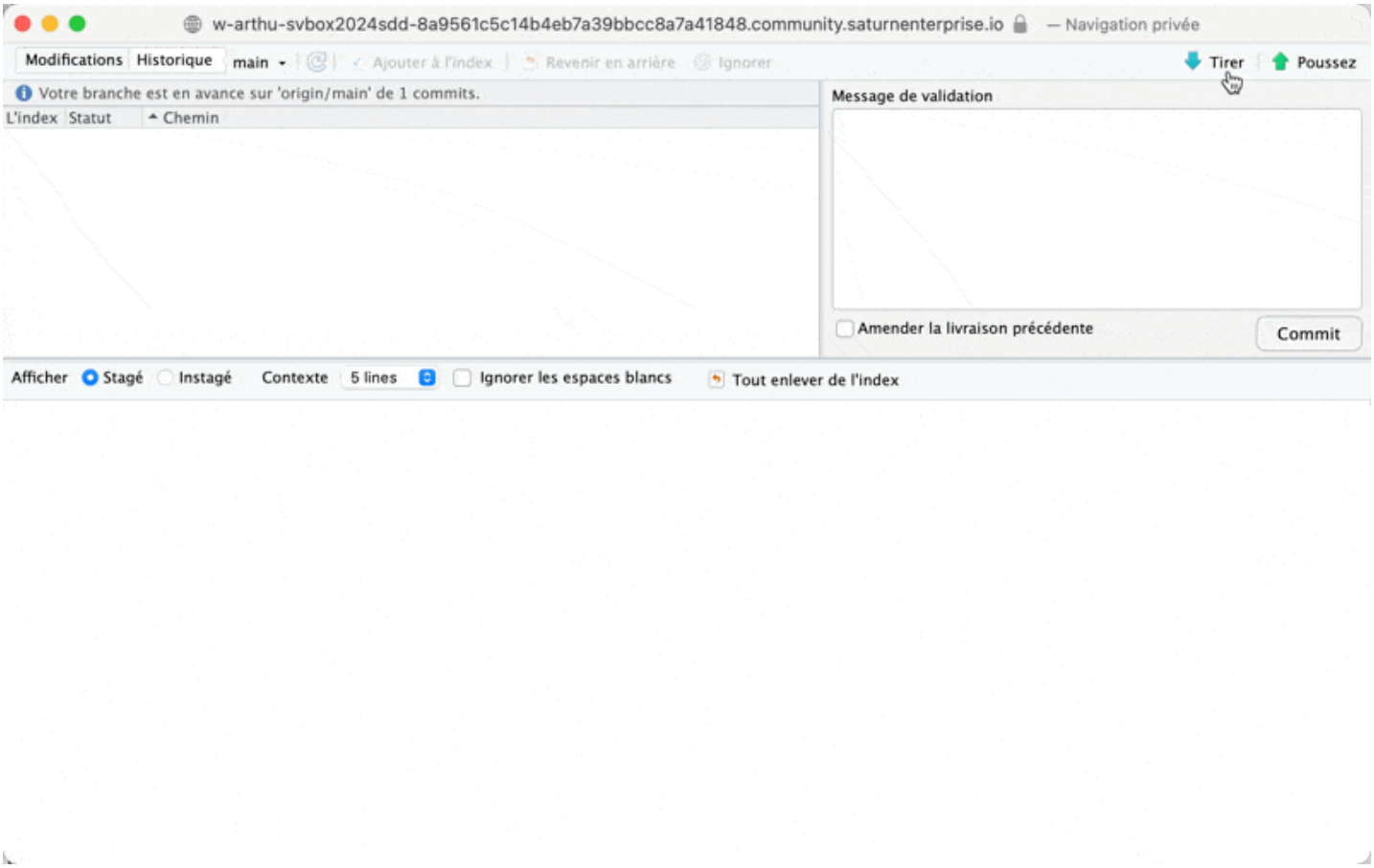
dans notre exemple. Vous devez écrire aussi un court message de validation qui fournit une description des modifications apportées dans ce commit.



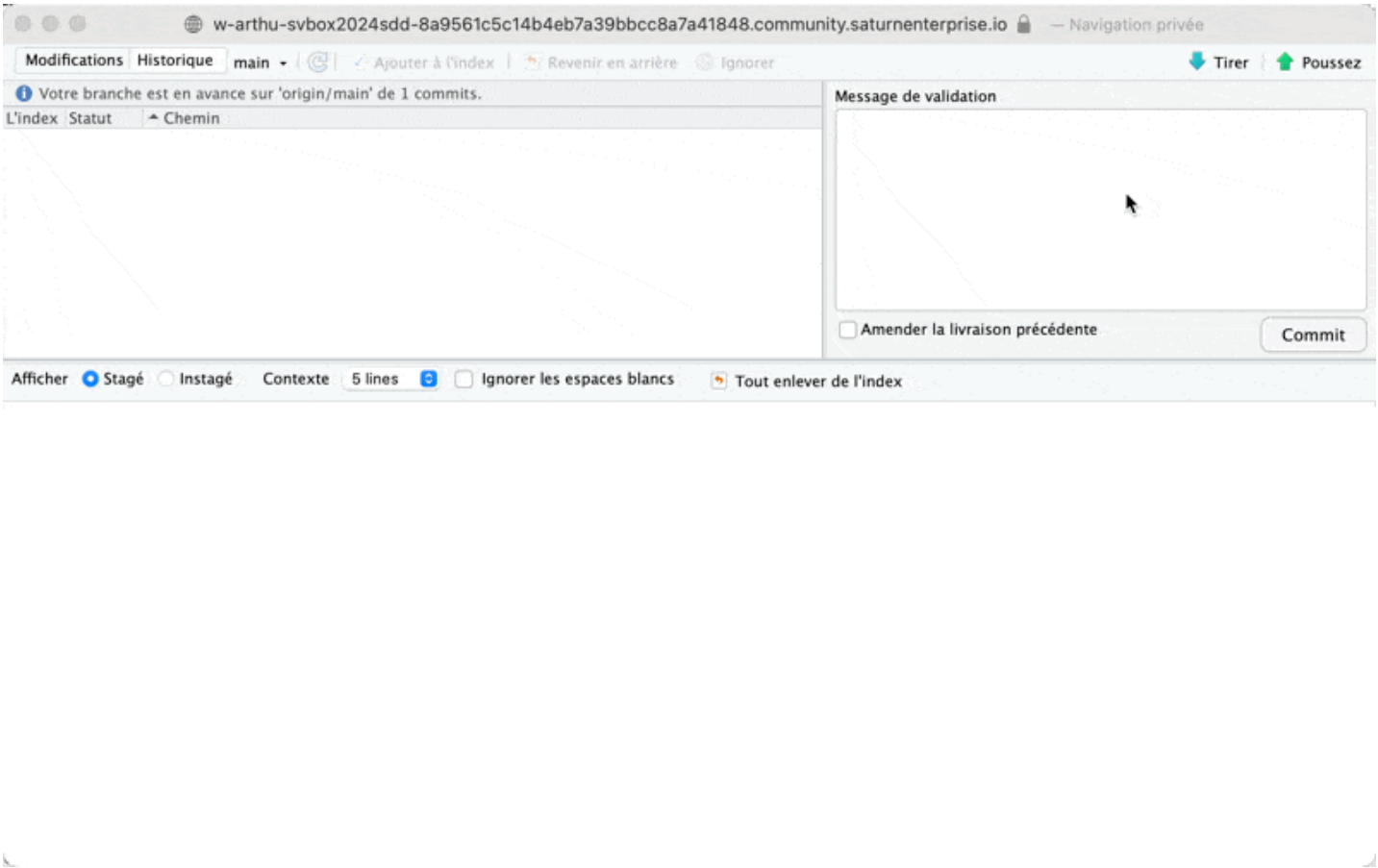
Il ne reste plus qu'à cliquer sur le bouton "Commit" pour finaliser ce point de sauvegarde. Vous pouvez observer ce qui se passe dans le gif animé juste en dessous. Une fois l'opération terminée, des informations sont affichées, comme le nombre de fichiers modifiés inclus dans ce commit.



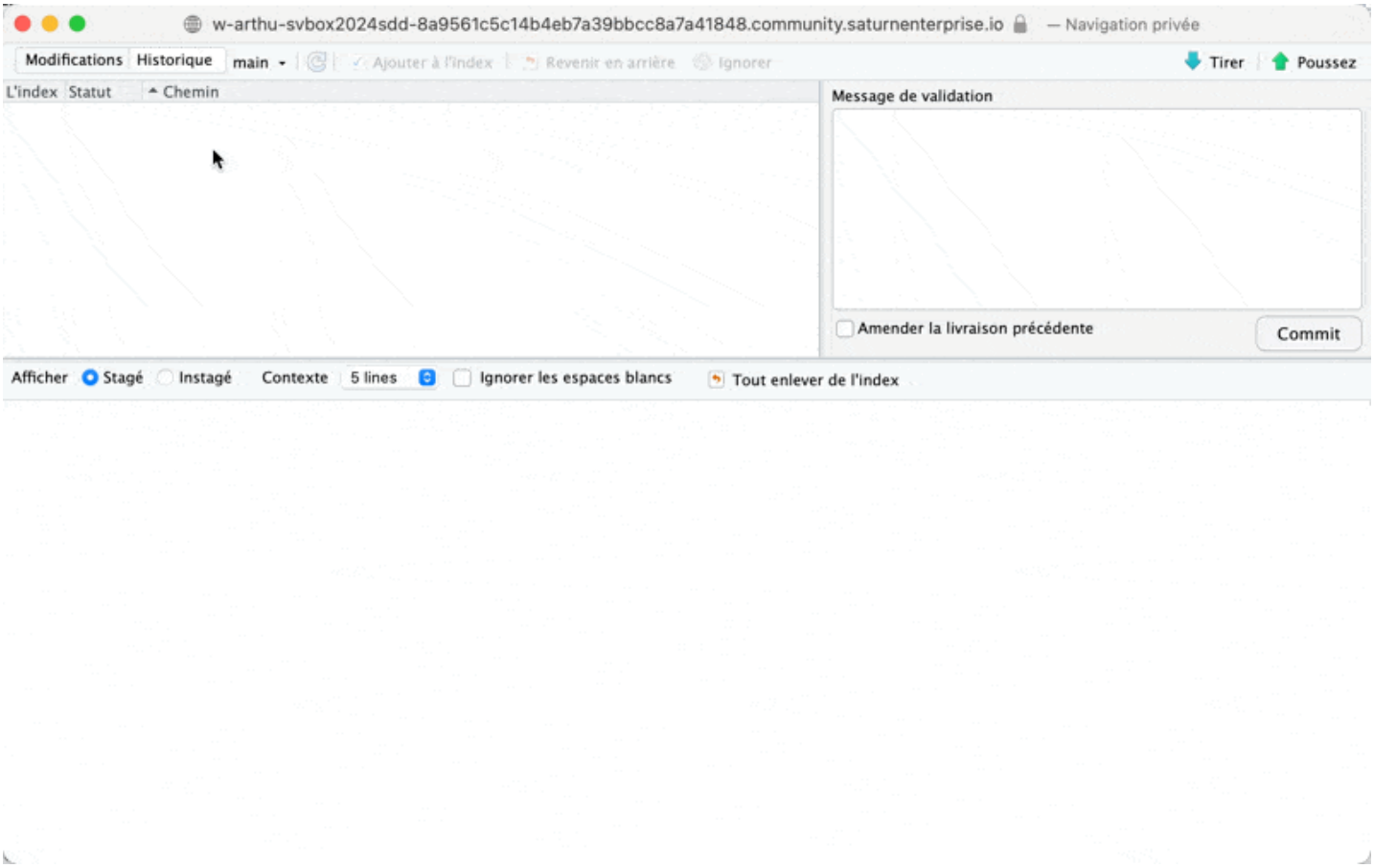
Une version de notre dépôt est également présente en ligne dans GitHub. À présent, il va falloir synchroniser les deux versions. Pour ce faire, nous vous conseillons de toujours commencer par rapatrier localement les changements réalisés dans le dépôt GitHub distant. Cette étape se nomme un **pull**. Vous pouvez cliquer sur tirer (une flèche bleue vers le bas). Elle sera cruciale lors d'un travail à plusieurs. Sur l'illustration suivante, vous pouvez observer que vous êtes à jour (le message "Already up to date" apparaît).



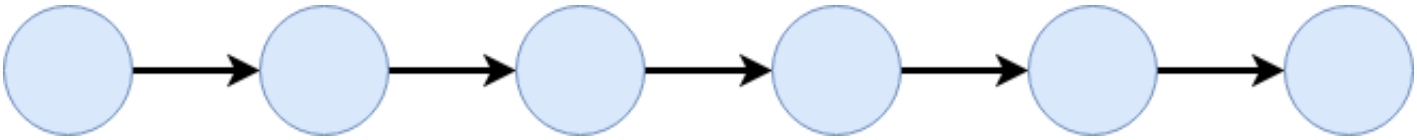
Enfin, vous allez envoyer vos changements locaux vers la version sur GitHub. Cette étape se nomme un **push**. Il s'agit de la flèche verte vers le haut "Poussez". Plusieurs **commits** peuvent être envoyés avec un seul **push**. Lisez *toujours* le message qui s'affiche à la fin de l'opération pour vous assurer que l'envoi a pu être réalisé sans erreurs.



En cliquant sur “Historique”, vous pouvez suivre la progression des commits au cours du temps. Sur le gif animé suivant, on observe trois commits successifs.



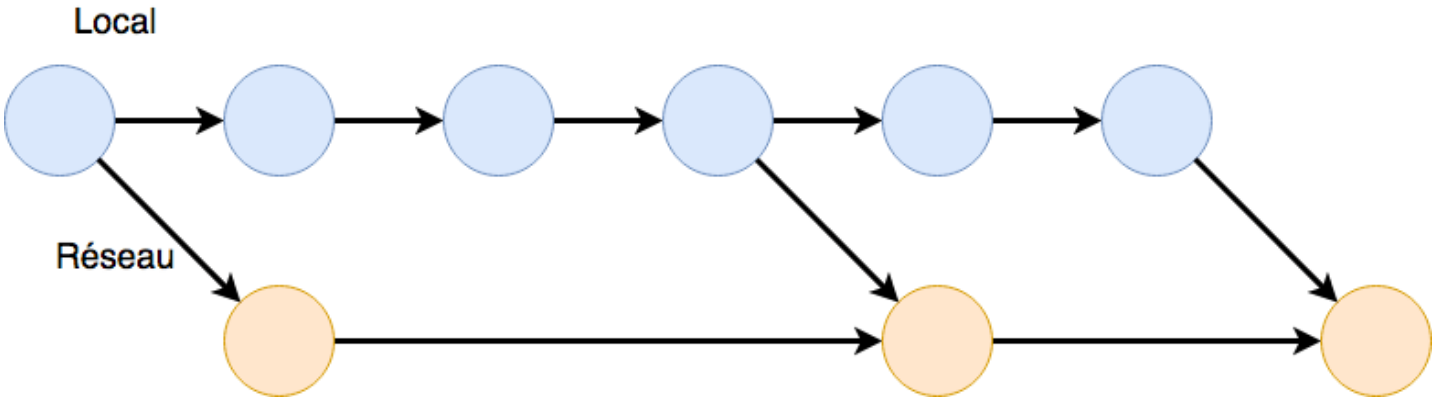
Le logiciel Git remplace les nombreuses copies d'un même fichier par différents états de votre document ou dossier, que l'on peut représenter schématiquement comme ci-dessous (chaque boule bleue représente une version enregistrée dans le système et les flèches indiquent d'où provient chaque version et où elle est utilisée ensuite) :



Représentation de la gestion de fichiers avec Git

Sur ce schéma, le flux est linéaire. Pour enregistrer une nouvelle version de votre document, vous effectuez un **commit** accompagné d'un message expliquant les modifications apportées.

Le gestionnaire Git peut être couplé à un hébergement sur le Cloud, soit pour simplement faire un backup de nos projets, soit pour pouvoir échanger et collaborer. Nous utilisons GitHub à cette fin dans le cours. Lorsque l'on travaille seul avec GitHub, l'évolution de notre projet ressemblera au schéma ci-dessous :



Représentation des versions successives d'un projet avec GitHub.
En bleu, les différentes versions locales de notre projet. En orange, les états modifiés de la version du projet sur GitHub. Les flèches indiquent le sens de la synchronisation.

> Vous venez d'apprendre le B-A-BA de la terminologie nécessaire à la bonne compréhension de Git et GitHub :
>
> - **dépôt** ou **repository** : espace de stockage sous gestion de version Git.
> - **commit** : enregistrer une version du projet.
> - **clone** : créer un double local d'un dépôt GitHub.
> - **push** : envoyer ses modifications locales vers le dépôt GitHub.
> - **pull** : rapatrier les modifications que les autres utilisateurs ont réalisé dans le dépôt GitHub vers sa propre version locale.

Ceci n'est qu'une explication très succincte. Vous trouverez plus de détails dans les liens ci-dessous et dans les modules suivants.

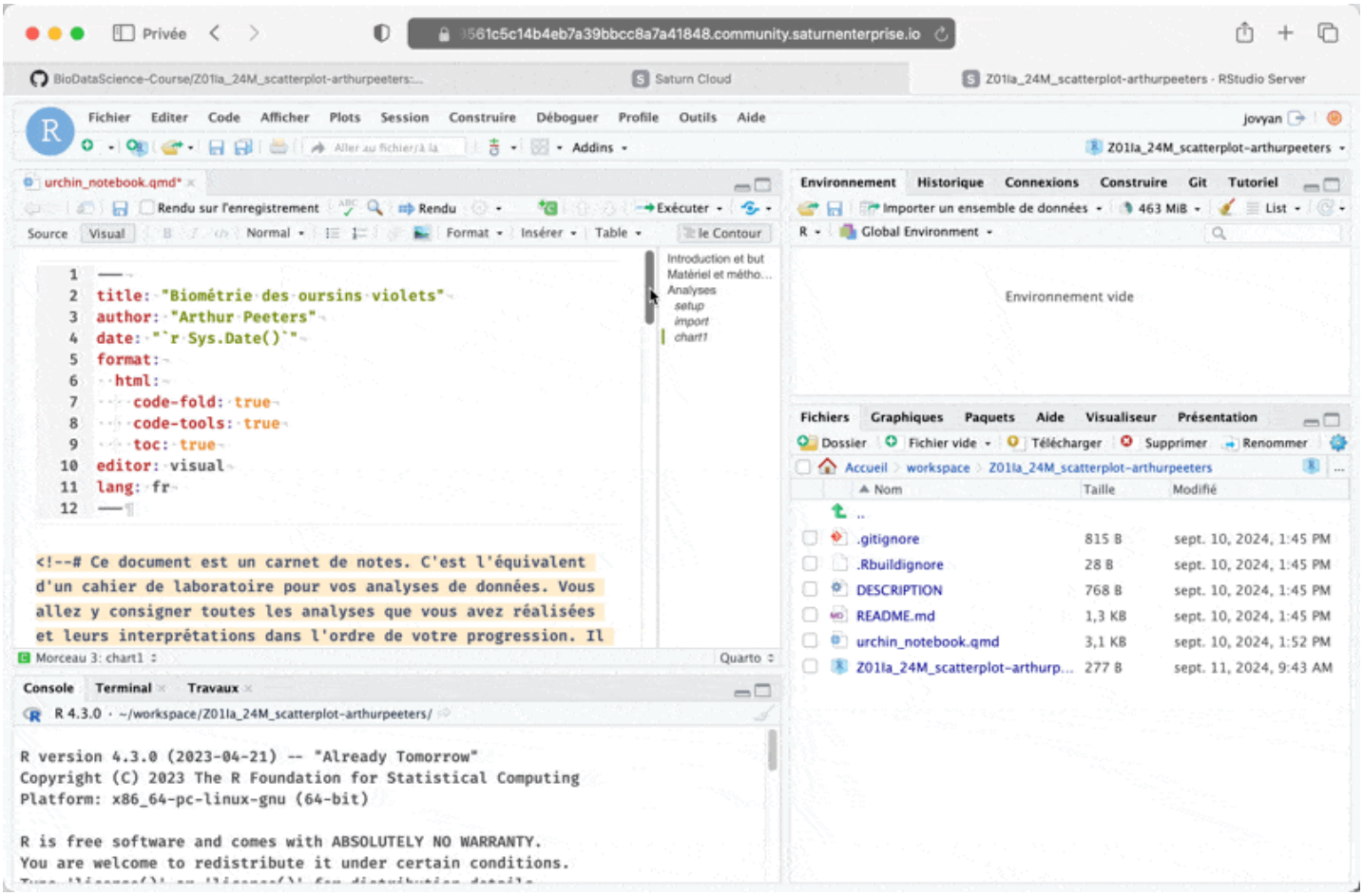
Pour en savoir plus

- Gérez vos codes sources avec Git. Explication en français sur l'utilisation de Git.
- Quel logiciel de gestion de versions devriez-vous utiliser ?. Explication en français sur l'utilisation des logiciels de gestion de versions.
- Happy Git and GitHub for the useR. Complet, mais un peu technique et en anglais.

- Git. Site en anglais rassemblant toute la documentation de Git.
- GitHub pour les nuls : pas de panique, lancez-vous !. En français.

1.3.5 Quarto/R Markdown

Le fichier édité dans la section précédente était un fichier Quarto qui a une extension `.qmd` [5]. Un fichier R Markdown est caractérisé par l'extension `.Rmd`. Ces deux formats sont très similaires, Quarto étant la version plus récente de R Markdown. Tous deux sont développés et maintenus par la même société (Posit). À partir d'ici, nous indiquerons Quarto/R Markdown lorsque la phrase s'applique indifféremment aux deux formats. Donc, Quarto/R Markdown permet de combiner le langage Markdown avec du code R, voire du code dans d'autres langages comme Python, Julia, SQL… Ces zones contenant du code exécutable en R ou autre s'appellent des **chunks**, ou **morceaux** dans l'interface RStudio en français (mais nous utiliserons préférentiellement le terme anglais dans cet ouvrage).



Lorsqu'on rédige un document, on perd beaucoup de temps à décider de l'interligne, de choisir la taille et la police… bref au niveau de la mise en page. On peut diviser la rédaction d'un document en deux grandes étapes : le fond et la forme. Nous vous proposons un outil pour booster votre **productivité** (et donc pour gagner du temps) avec Markdown. Dans un premier temps, nous allons nous intéresser uniquement au fond. Ensuite, nous déterminerons comment le document sera mis en forme.

Des **"balises"** spéciales Quarto/R Markdown vont marquer les différentes parties de texte lors de l'écriture du fond qui seront rendus différemment lors de la mise en page (la forme). Par exemple, une ligne qui commence par un ou plusieurs dièses (`#` encore appelé "hashtag") sert à indiquer un titre de niveau équivalent au nombre de dièses.

Donc, un document R Markdown/Quarto est constitué d'une succession de zones de texte Markdown et de chunks contenant du code. Le passage de zone Markdown à zone de code et inversement se fait avec des balises particulières :

- en entrée de chunk R : ` ```{r} ` seul sur une ligne. Il est aussi possible de rajouter un label, par exemple, ` ```{r graphique1} ` et/ou des options, par exemple, ` ```{r, echo=FALSE, results='hide'} ` pour cacher et le code et le résultat dans le rapport),
- en sortie de chunk R : ` ``` ` seul sur une ligne.

En mode éditeur visuel, les trois apostrophes inverses ` ``` ` en début et fin de chunk sont cachées. Mais tant en mode visuel qu'en mode source, le fond du texte est représenté dans une couleur différente pour les chunks et pour le texte Markdown, afin de bien les différencier.

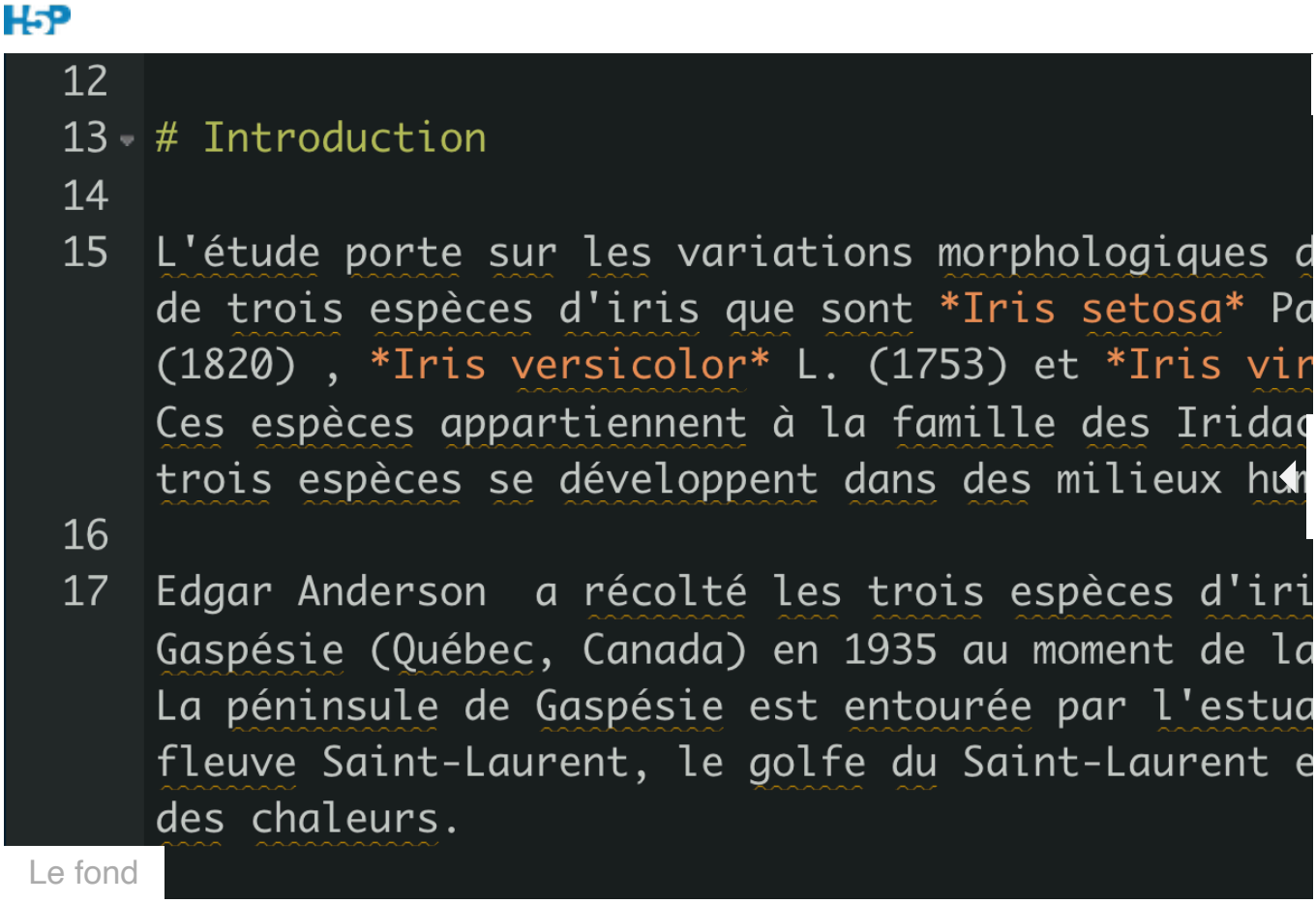
Vous devez bien entendu avoir autant de balises d'entrée que de balises de sortie. Des explications plus détaillées se trouvent dans l'annexe B.1.3 dédiée au R Markdown. De plus, l'écriture d'un texte scientifique doit respecter certaines conventions. Vous trouverez des explications à ce sujet dans l'annexe D.

> Il vous est toujours possible d'exécuter les instructions d'un chunk ligne après ligne dans la fenêtre **Console** pour les tester tout comme à partir d'un script R.

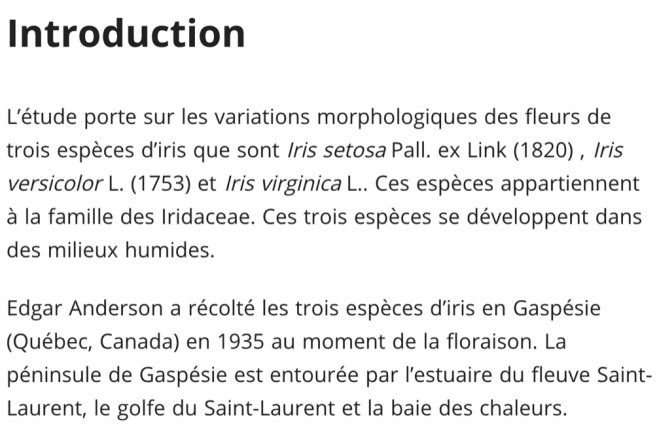
Markdown est relativement simple et intuitif à l'usage, même si un petit effort est nécessaire au début. L'apparence finale du texte sera, quant à elle, définie dans une feuille de style séparée. Par défaut, dans RStudio, un mode visuel est proposé, c'est-à-dire qu'on a la possibilité d'employer des boutons pour mettre en gras, en italique, etc. Bien que ce mode soit utile, nous vous recommandons de faire l'effort d'apprendre les principales balises du Markdown.

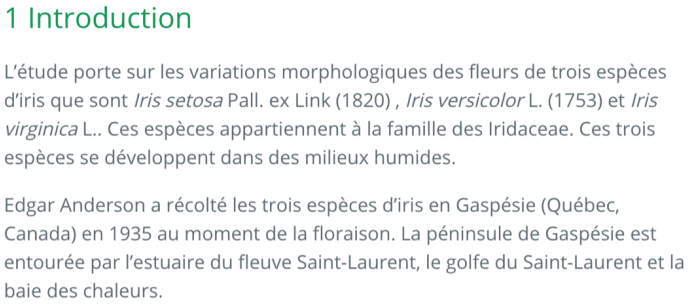
À vous de jouer !

Déplacez de droite à gauche le curseur sur l'image ci-dessous pour découvrir la forme finale que pourra prendre ce court paragraphe rédigé en Markdown (fond noir : vue dans l'éditeur, fond blanc : forme finale dans un navigateur Web).



En partant du document Markdown ci-dessus, on peut également obtenir des présentations et différents formats finaux. Ci-dessous, deux feuilles de style différentes ont été appliquées sur ce même document. La présentation diffère (police de caractères, couleurs, taille du texte…)





Le langage Markdown est également utilisé dans les issues de GitHub, dans des forums comme Reddit ou encore pour la rédaction de document avec divers éditeurs de textes.

Consultez les liens Syntaxe de base pour l'écriture et la mise en forme de GitHub et la première section de R Markdown Reference Guide afin d'en apprendre plus sur la syntaxe de Markdown.

À vous de jouer !

Cet exercice H5P reprend plusieurs questions. Assurez-vous de les passer toutes en revue l'une après l'autre.

H5P

Placez les mots à la bonne place.

Lorsque je souhaite écrire une équation en markdown, j'utilise la balise [$]. Il est aussi possible d'utiliser des indices et des exposants en dehors des équations. Pour obtenir un exposant, j'utilise ceci : [^2^]. Par contre pour obtenir un indice, j'utilise cela : [~2~].

Vérifier

Pour en savoir plus

Pour plus d'informations sur Quarto/R Markdown, vous pouvez consulter les liens suivants :

- Quarto, site contenant tout ce qu'il faut savoir sur Quarto (en anglais).
- Communicating results with R Markdown explique de manière succincte l'intérêt de R Markdown par rapport à Microsoft Office (en anglais).
- What is R Markdown?. Vidéo en anglais et site présentant les différentes possibilités, par les concepteurs de R Markdown (RStudio).
- Introduction to R Markdown. Tutoriel en anglais, par RStudio.
- R Markdown: the definitive guide est **le** manuel par excellence pour R Markdown (en anglais uniquement, malheureusement).
- Aide-mémoire R Markdown, dans les menus de RStudio : `Aide -> Cheat Sheets -> Aide-mémoire R Markdown`
- Référence rapide à Markdown, dans les menus RStudio : `Aide -> Référence rapide Markdown`

À vous de jouer !

Vous allez maintenant manipuler un document **Quarto** pour réaliser par vous-même des graphiques en nuage de points.

> Réalisez le travail **A01la_scatterplot**.
>
> **Travail individuel pour les étudiants inscrits au cours de Science des Données Biologiques I : visualisation à l'UMONS à terminer avant le 2025-10-01 23:59:59.**
>
> **Initiez votre projet GitHub Classroom**
>
> *Voyez les explications dans le fichier `README.md` .*

1.3.6 Rechercher d'aide

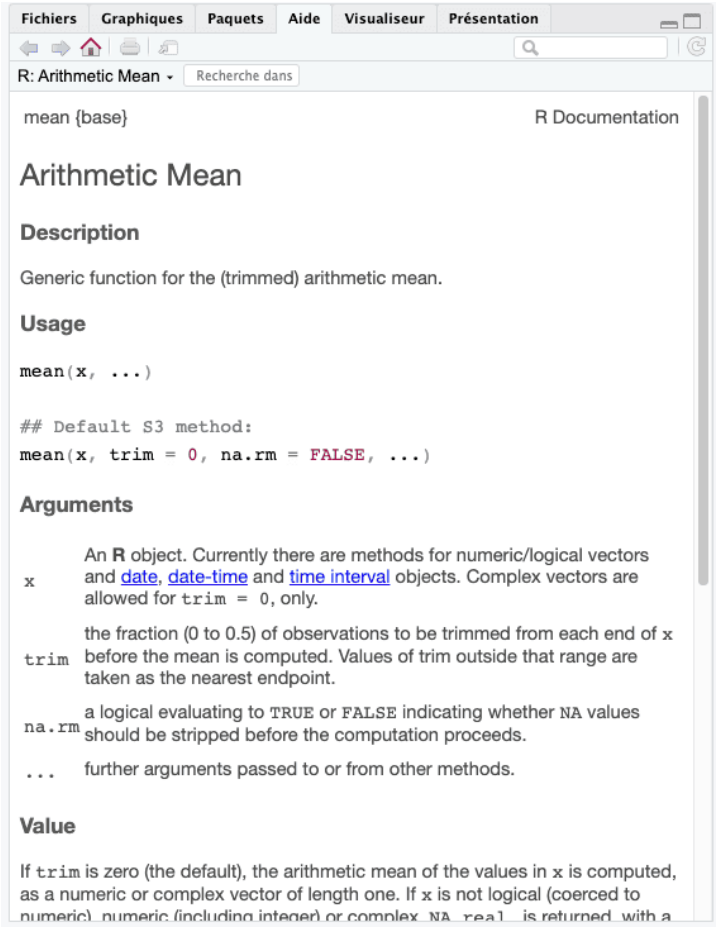
Dans le cadre de ce cours, vous allez vous poser de nombreuses questions. Ces questions pourront porter sur l'utilisation des logiciels, sur la théorie statistique derrière un concept ou encore sur l'utilisation d'une fonction particulière du langage R. Cette section va vous donner des pistes pour trouver de l'aide par vous-mêmes ou en faisant appel à la communauté. Il est indispensable d'apprendre à rechercher l'information pour répondre à vos questions. C'est seulement comme cela que vous progresserez efficacement.

1.3.6.1 Pages d'aide

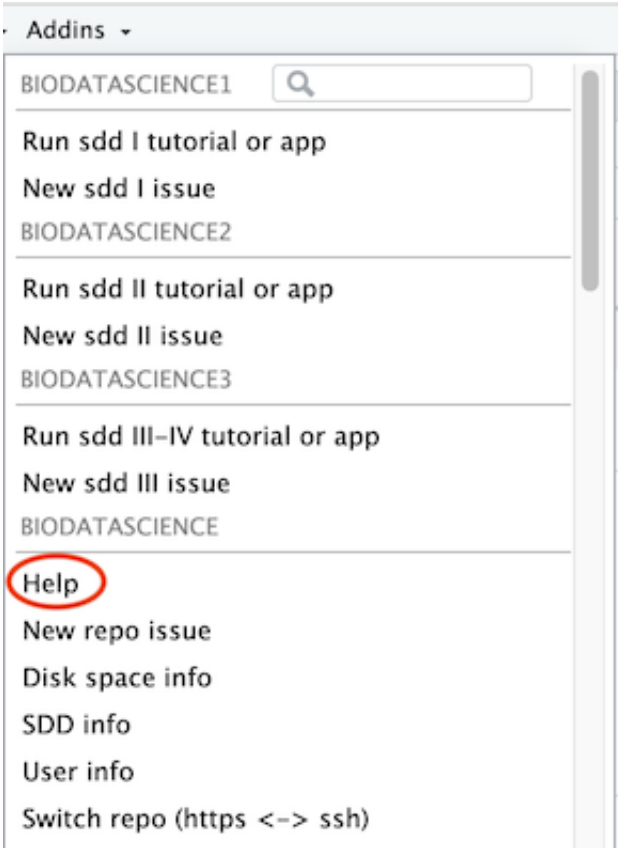
Des pages d'aide sont associées à chacune des fonctions dans R. Pour y accéder, utilisez l'opérateur `?` suivi du nom de la fonction. Dans RStudio, la page d'aide apparaît dans l'onglet **Aide**. Dans RStudio, il est encore bien plus facile de placer le curseur sur le nom de la fonction dans le code et d'appuyer sur la touche `F1` pour faire apparaître sa page d'aide.

```
?mean
```

Voici le début de la page d'aide proposée concernant la fonction mean()



Une page d'aide est en anglais et bien souvent technique. Dans certain cas, la page d'aide ne va pas être suffisante afin de vous permettre de répondre à votre interrogation. Si `?<topic>` ne fonctionne pas, essayez `??<topic>` en doublant le point d'interrogation pour une recherche approfondie dans les pages d'aide, ou encore en utilisant `.?<topic>` (point - point d'interrogation) dans SciViews-R pour encore plus de détails dans les recherches.



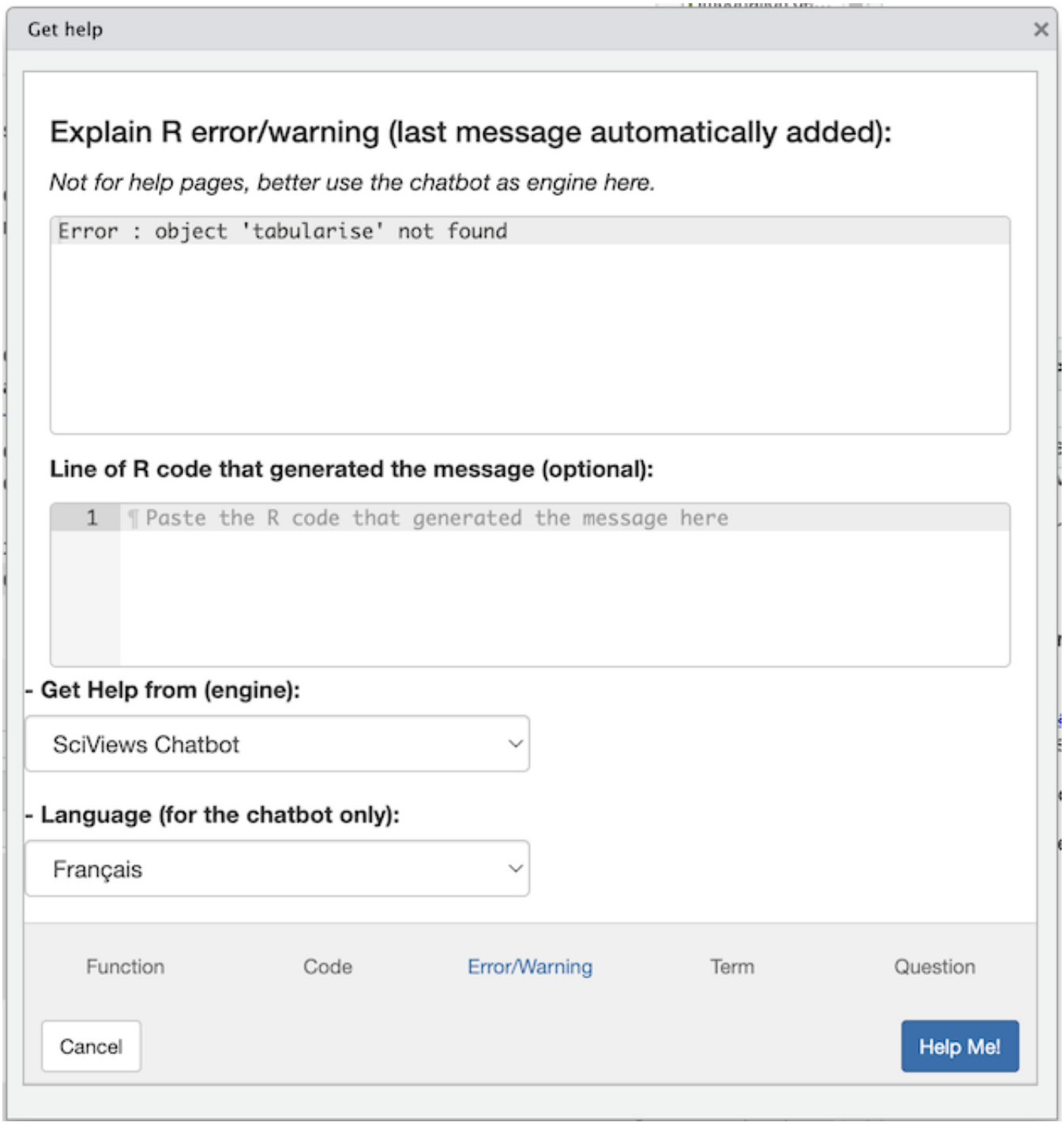
Vous pouvez aussi placer le curseur ou sélectionner le nom d'une fonction dans l'éditeur de texte de Rstudio et ensuite faire appel à l'addin **Help** pour rechercher la page d'aide qui vous intéresse. Pour certaines fonctions dites **génériques**, il existe plusieures variantes (dites **méthodes**) que l'addin **Help** vous listera.

1.3.6.2 Chatbot SciViews

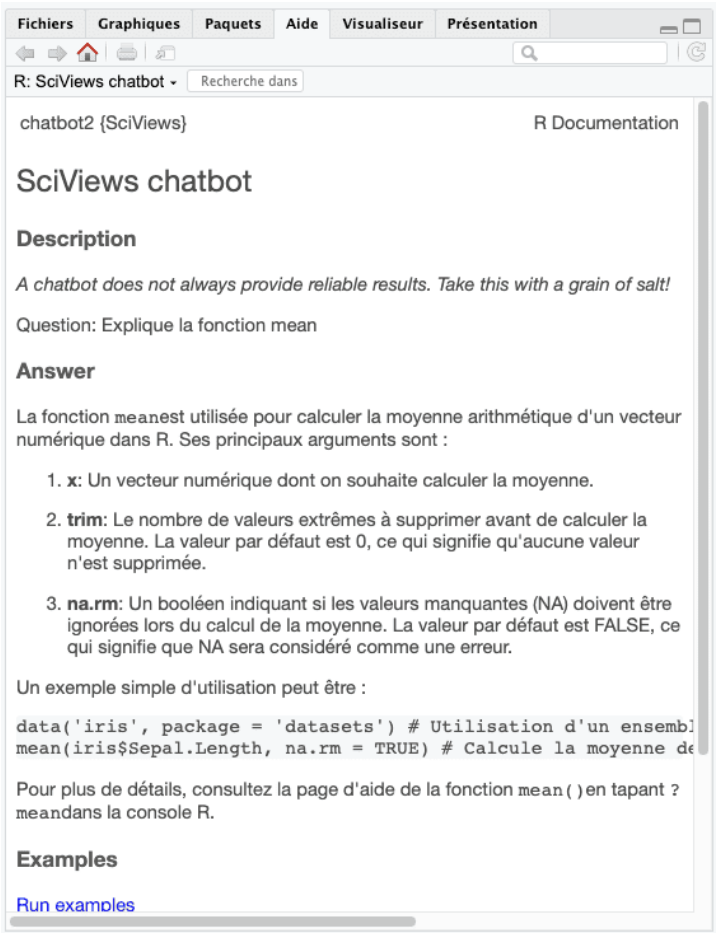
Le chatbot SciViews est un grand modèle de langage, plus connu sous sa dénomination anglaise de LLM ou encore de *large language model*. Derrière ChatGPT, Perplexity, Copilot et Gemini se cachent des LLM. Le chatbot est disponible dans votre RStudio au travers ud même addin **Help** cité un peu plus haut. Il vous est mis à disposition dans le cadre de ce cours. Il répond aux questions qui traitent du langage R, des statistiques et de la science des données.

Il peut vous préparer une page d'aide simplifiée, en anglais, mais aussi en français, sur les fonctions R. Il peut aussi servir à expliquer quelques lignes de code R, un message d'erreur renvoyé par R. Il peut aussi définir et expliquer un terme (statistique, par exemple).

Enfin, vous pouvez lui poser n'importe quelle question (mais vous comprendrez très vite qu'il se borne essentiellement à répondre à tout ce qui a un rapport avec R, la statistique ou la science des données).



Les réponses du chatbot SciViews apparaîtront comme une page d'aide dans RStudio.



> Attention, les LLM peuvent halluciner. Une hallucination est une réponse qui semble plausible, mais qui est en réalité incorrecte. Cela peut se produire lorsqu'il y a des lacunes dans les données d'entraînement du modèle ou qu'une question posée est ambiguë.
>
> **Soyez toujours critique face à l'information proposée par ce chatbot, ainsi que n'importe quel LLM. Vérifiez toujours les informations fournies par une autre voie.**

1.3.6.3 Recherche sur internet

Sur Internet, il est possible de trouver des centaines, voire des milliers d'articles de blog, des tutoriels, des aide-mémoires et des manuels en ligne en lien avec R et tout ce qui l'entoure. Les ressources sont principalement en anglais, mais on peut en trouver en français. Il est impératif de bien utiliser les moteurs de recherche pour trouver rapidement

une information de qualité. Cela nécessite de poser la bonne question. Une recherche sur Internet est un excellent complément à la réponse d'un chatbot afin de valider l'information.

Voici quelques propositions :

- Un blog en ligne : https://www.r-bloggers.com/
- Des aide-mémoires : https://rstudio.github.io/cheatsheets/
- Sites web :
  - Une documentation collaborative sur R : https://book.utilitr.org/
  - STHDA : http://www.sthda.com/french/wiki/wiki.php
- Des ouvrages en ligne :
  - R for Data Science : https://r4ds.hadley.nz/
  - Introduction à R et au tidyverse: https://juba.github.io/tidyverse/

Le manuel suivant propose une section détaillée sur comment trouver de l'aide : https://larmarange.github.io/analyse-R/ou-trouver-de-l-aide.html

L'addin **Help** dan RStudio vous permet de rechercher de l'aide dans les pages d'aide de R, mais aussi sur Internet avec le moteur **Rseek** spécialisé dans la recherche de document sur le langage R, **StackOverflow** (un forum dédié à la programmation) ou encore **Google**, **Bing** ou **DuckDukGo**.

> Généralement, les informations que vous allez retrouver devront être modifiées afin de s'adapter à votre problématique ou à la consigne de l'exercice auquel vous répondez.

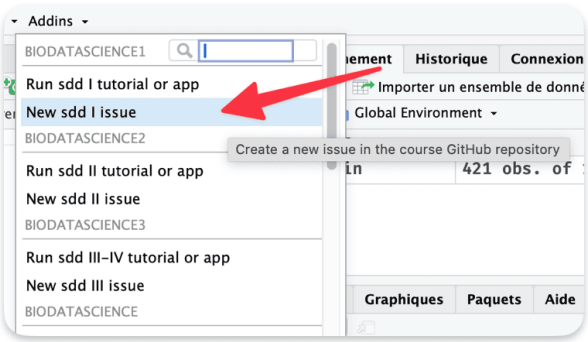
1.3.6.4 Appel à la communauté

Néanmoins, si vous ne trouvez pas la réponse à votre question malgré une recherche à l'aide des différents outils présentés ci-dessus, il vous reste la possibilité de faire appel à la communauté.

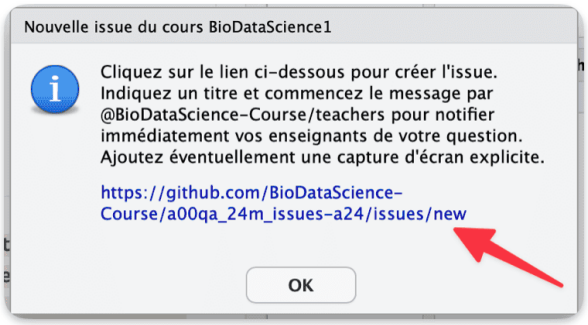
Pour tous les étudiants du cours de Sciences des données I, un espace d'échange est disponible pour poser des questions entre les membres du cours de Science des données I, vous pouvez y accéder également à partir des addins de RStudio. Vous posez facilement votre question en trois étapes.

1. Depuis RStudio, cliquez sur l'addin qui se trouve dans le menu déroulant `Addins`
2. Une fenêtre s'ouvre pour vous rediriger vers l'espace de discussion.
3. Vous posez votre question.

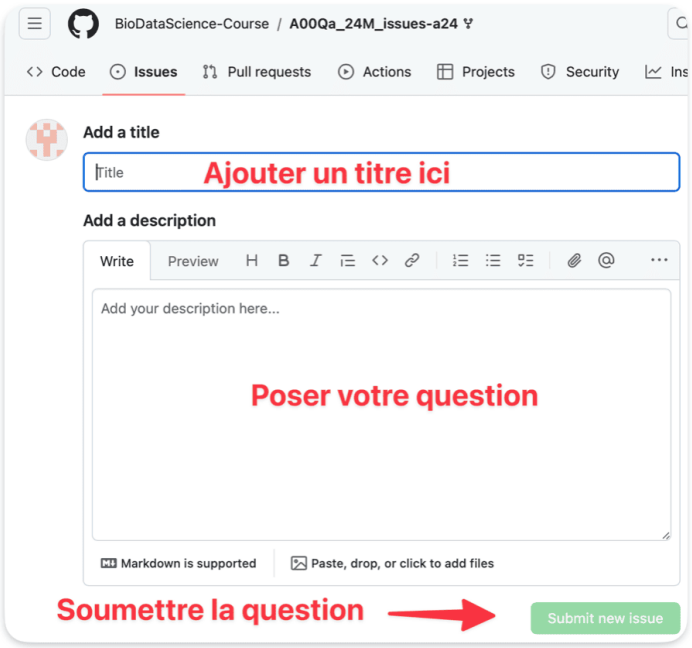
1



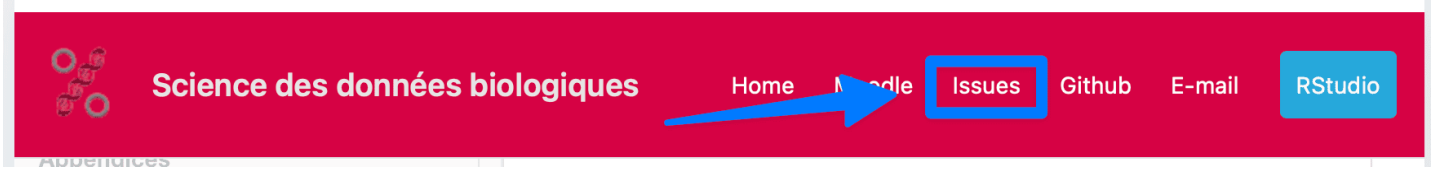
2



3



Vous pouvez également accéder à cet espace de discussion depuis n'importe quelle page de site de ce cours.



Il est possible de faire appel à une communauté bien plus grande, mais nous en resterons là pour ce premier module.

5. L'**extension d'un fichier** est un indicateur de son format à la fin de son nom. Il se présente sous forme d'un point (.) suivi d'un sigle de quelques lettres. Par exemple `.txt` caractérise un fichier texte, alors que `.png` indique qu'il s'agit d'une image au format PNG. Attention que les extensions sont parfois cachées (c'est systématique dans l'explorateur de fichiers de Windows avec la configuration par défaut, et c'est selon le fichier dans le Finder de macOS. Sous Linux, et donc dans votre machine Saturn Cloud, l'extension est *toujours* affichée heureusement.↩

1.4 Récapitulatif des exercices

Bravo ! Vous arrivez au bout de votre premier module. Cette dernière section vous permet de vérifier que vous avez bien réalisé tous les exercices et de contrôler votre progression telle qu'enregistrée dans notre base de données (étudiants UMONS uniquement). Vous aviez à réaliser les exercices suivants :

- A00La_discovery - Découverte des tutoriels de type learnr
- A01Ha_rstudio2 - L'environnement de RStudio
- A01La_base - Notions de base de R
- A01Hb_nuage2 - Vidéo d'introduction au graphique en nuage de points
- A01Lb_scatterplot - Graphiques en nuages de points
- A01Hc_chart - Les fonctions chart() et geom_point()
- A01Sa_limits - Étendue des axes d'un graphique
- A01Sb_transformation - Transformation des axes d'un graphique
- A01Hd_github_project - Le projet dplyr dans GitHub
- A01He_github - Exploration d'un dépôt GitHub
- A01Hf_rs_project2 - Les projets dans RStudio
- A01Hg_md_comp - Le fond versus la forme
- A01Hh_markdown2 - Premiers exercices relatifs à Markdown
- A01Ia_scatterplot - Graphiques avec `urchin_bio`

Progression

Module 2 Visualisation II

Objectifs

- Être capable de réaliser différentes variantes de graphiques visant à montrer *comment les données se distribuent* telles que les histogrammes, les graphiques de densité ou encore les diagrammes en violon dans R avec la fonction `chart()`
- Intégrer des graphiques dans un rapport et y décrire ce que vous observez
- Gérer des conflits dans GitHub

Prérequis

Le premier module de ce cours vous a permis de découvrir de nouveaux outils logiciels comme GitHub, Git, R Markdown/Quarto ou encore RStudio. Ces outils vont être employés tout au long de votre formation en science des données (et donc de ce cours). Il est indispensable de les maîtriser.

À vous de jouer !

H5P

Choisissez l'affirmation exacte. ✔ Progression : 0/4

- Github est un réseau social dédié à la collaboration entre utilisateurs, centré autour de projets utilisant Git.
- Github est environnement de travail optimisé pour utiliser R.
- Github est un gestionnaire de version que permet de réaliser des commits, des push et des pulls.

Ce module va vous permettre de découvrir de nouveaux graphiques à réaliser avec R. Vous devez donc maîtriser les instructions R qui permettent de réaliser un graphique.

À vous de jouer !

Déplacez les éléments dans les emplacements qui leur correspondent afin de compléter la fonction suivante.

chart (data = df, y~x) +
geom_xxxx ()

Vérifier

Ces graphiques vont être réalisés dans des scripts R et dans des documents R Markdown/Quarto. Il est dès lors indispensable de bien comprendre les différences entre ces deux approches.

À vous de jouer !

Choisissez l'affirmation exacte. ✔ Progression : 0/3

- Le R Markdown est la combinaison de R et du Markdown.
- Le R Markdown permet décrire du R et du Markdown dans Microsoft Word.
- Le R Markdown combine le PDF et le Markdown dans R.

La fin de ce module va traiter de la gestion des conflits dans GitHub. Vous devez maîtriser GitHub et Git pour apprendre ensuite à gérer ces conflits (nous vous expliquerons bien entendu de quoi il s'agit exactement).

À vous de jouer !

Lorsqu'un utilisateur souhaite faire une copie locale d'un dépôt de son compte GitHub, il doit faire un ...

- ○ pull
- ○ push
- ○ clone

Afficher la réponse

2.1 Langage R

Avant de découvrir de nouveaux types de graphiques, réalisez une analyse concrète de données biologiques dans un tutoriel learnr, prétexte pour en apprendre un peu plus sur le langage R.

À vous de jouer !

> Effectuez maintenant les exercices du tutoriel **A02La_progression (Progression R)**.
>
> ```
> BioDataScience1::run("A02La_progression")
> ```

(BioDataScience1 est un package R spécialement développé pour ce cours et qui est préinstallé dans votre SciViews Box).

H5P

Quelle est la fonction qui permet de calculer la moyenne d'un vecteur en R ?

- ○ mean(x)
- ○ exp(x)
- ○ mutate(x)

Afficher la réponse

2.1 Langage R

Avant de découvrir de nouveaux types de graphiques, réalisez une analyse concrète de données biologiques dans un tutoriel learnr, prétexte pour en apprendre un peu plus sur le langage R.

À vous de jouer !

> Effectuez maintenant les exercices du tutoriel **A02La_progression (Progression R)**.
>
> ```
> BioDataScience1::run("A02La_progression")
> ```

(BioDataScience1 est un package R spécialement développé pour ce cours et qui est préinstallé dans votre SciViews Box).

H5P

Quelle est la fonction qui permet de calculer la moyenne d'un vecteur en R ?

- ○ mean(x)
- ○ exp(x)
- ○ mutate(x)

Afficher la réponse

2.2 Histogramme

Vous souhaitez visualiser l'étalement de vos données sur un axe (on parle de **distribution**[6] en statistique) pour l'une des variables étudiées. L'histogramme est l'un des graphiques qui vous apportent cette information de manière visuelle. Il représente sous forme de barres un découpage en plusieurs **classes**[7] d'une variable numérique.

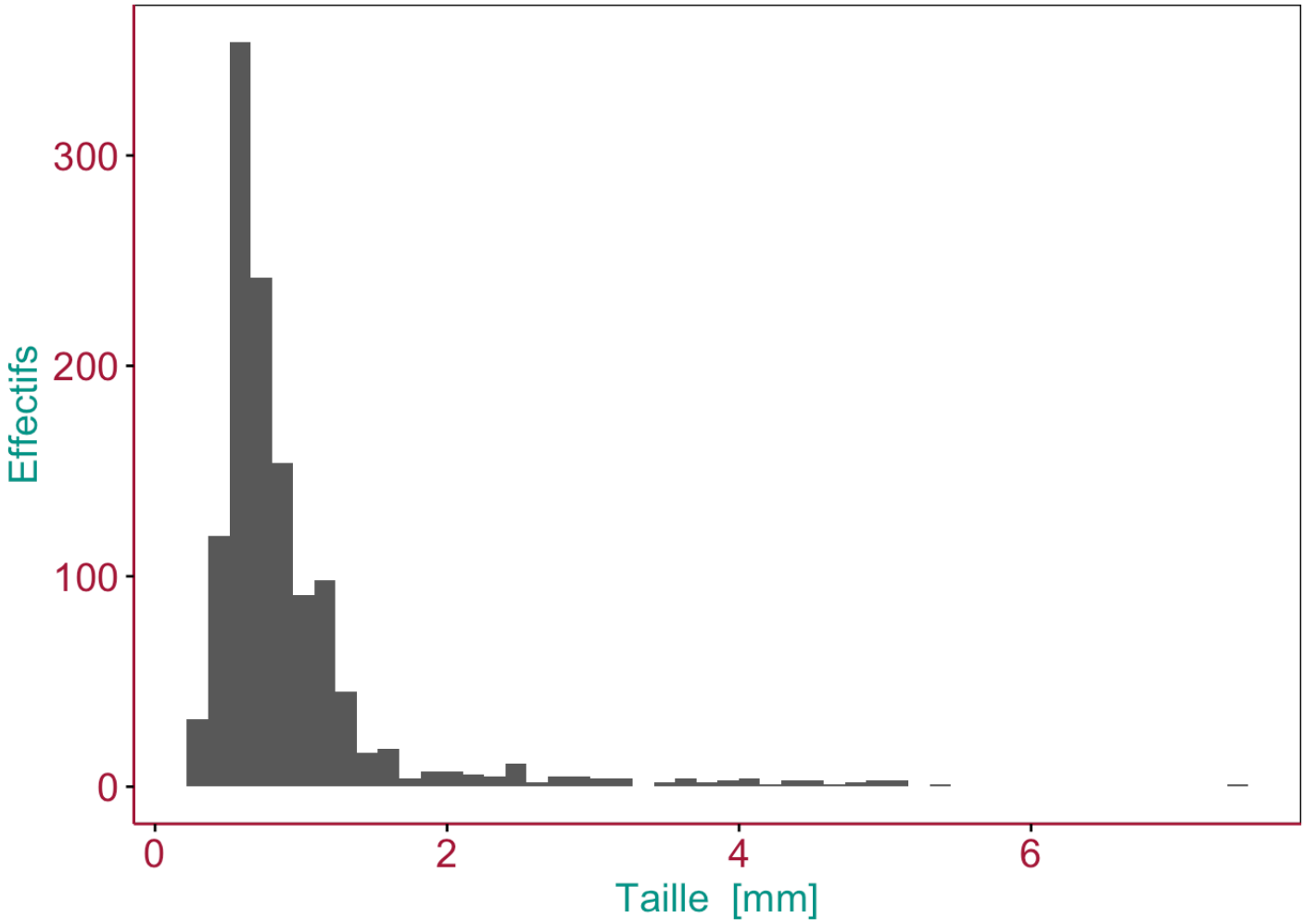


Figure 2.1: Exemple d'histogramme montrant la distribution de la taille dans un échantillon de zooplancton. Des couleurs sont utilisées pour mettre en évidence différentes parties du graphique (en pratique, les axes et leurs labels seront noirs).

Outre l'histogramme lui-même, représenté par des barres noires de hauteur équivalente au nombre d'observations dans les différentes classes, les éléments suivants sont également indispensables à la compréhension du graphique (ici mis en évidence en couleur) :

- Les axes avec les graduations (en rouge). Sur l'axe des abscisses, la variable numérique étudiée (ici la taille en mm), et sur l'axe des ordonnées, les effectifs
- les labels des axes et les unités (pour l'axe des abscisses uniquement ici) (en bleu)

Partons d'un jeu de données sur le zooplancton.

```
# Importation du jeu de données
zooplankton <- read("zooplankton", package = "data.io", lang = "FR")
# Affichage des premières et dernières lignes du jeu de données
tabularise$headtail(zooplankton)
```

Diamètre circulaire équivalent [mm]	Aire [mm^2]	Périmètre [mm]	Diamètre de Feret [mm]	Axe majeur de l'ellipsoïde [mm]	Axe mineur de l'ellipsoïde [mm]	D.O. moyenne	D.O. plus fréquente	m
0.7697829	0.4654	4.4469	1.3168	1.1640	0.5091	0.3631	0.036	
0.7004136	0.3853	2.3247	0.7281	0.7128	0.6882	0.3609	0.492	
0.8147804	0.5214	4.1509	1.3267	1.1106	0.5978	0.3082	0.032	
0.7850146	0.4840	4.4422	1.7845	1.5641	0.3940	0.3317	0.036	
0.3614338	0.1026	1.7065	0.7391	0.6940	0.1883	0.1526	0.016	
...	...	...	...	...	...	...	...	
0.5579490	0.2445	4.6214	0.9864	0.7318	0.4254	0.0329	0.012	
0.4763311	0.1782	4.2148	0.9864	0.5593	0.4057	0.1474	0.016	
0.5744769	0.2592	5.4060	0.9302	0.7010	0.4709	0.0374	0.012	
0.6375093	0.3192	6.9642	1.6955	0.7468	0.5443	0.1576	0.008	
0.5817449	0.2658	5.1730	1.0174	0.6401	0.5288	0.1292	0.008	

Premières et dernières 5 lignes d'un total de 1262

Les instructions dans R pour produire un histogramme similaire à celui présenté plus haut à l'aide de la fonction `chart()` sont :

```
chart(data = zooplankton, ~ size) +
  geom_histogram(bins = 50) + # bins = nombre de classes souhaitées
  ylab("Effectifs")
```

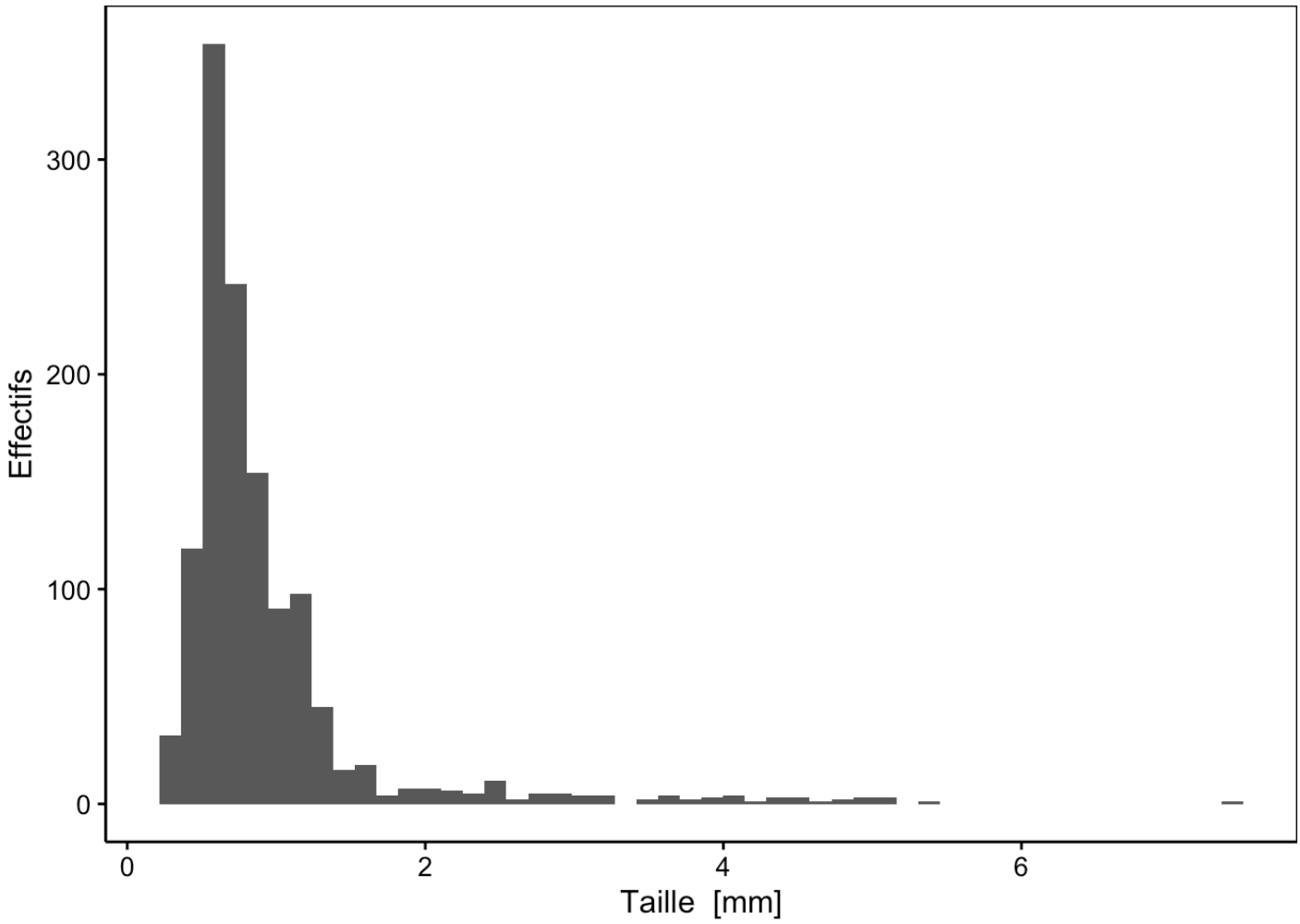


Figure 2.2: Distribution des tailles au sein d'un échantillon de zooplancton

La fonction `chart()` requiert comme argument le jeu de donnée (`data = zooplankton`), ainsi que la formule à employer dans laquelle vous avez indiqué le nom de la variable que vous voulez sur l'axe des abscisses à droite, après le tilde `~` . Parmi toutes les variables du jeu de données, nous avons choisi ici de représenter `size` . Jusqu'ici, nous avons spécifié *ce que* nous voulons représenter, mais pas encore *comment* (sous quelle apparence), nous voulons matérialiser cela sur le graphique. Avec les graphiques de type `ggplot()` que nous réalisons ici à l'aide de `chart()` , nous **ajoutons** des couches au graphique à l'aide de l'opérateur `+` . Pour un histogramme, nous devons ajouter une

couche `geom_histogram()`, tout comme pour le nuage de points, nous utilisions `geom_point()`. L'argument `bins=` dans cette fonction permet de préciser le nombre de classes souhaitées.

> Le **découpage en classes**, ici de la variable `size` se fait automatiquement. Elle consiste à diviser l'intervalle des valeurs prises par la variable en *n* classes et à comptabiliser le nombre d'observations qui se situent dans chaque classe. Ce nombre d'observations est ensuite représenté par une barre verticale dont la hauteur est proportionnelle à ce nombre dans l'histogramme.
>
> Prenons un exemple concret simple : des observations d'une variable *X* qui sont 3.1, 2.6, 5.4, 5.3, 2.4, 4.2, 5.3. Vous désirez découper en quatre classes allant de 2 à 3 (classe A), 3 à 4 (classe B), 4 à 5 (classe C) et 5 à 6 (classe D). Vous devez naturellement décider comment les bornes des classes sont définies. Par exemple, 3 est-il dans la classe A ou dans la B ? Si vous décidez que la classe inclut la borne inférieure mais pas la supérieure, c'est à dire, A = [2, 3[et B = [3, 4[en écriture mathématique, alors 3 appartient à la classe B. Dans ce cas, vous avez 2 observations dans la classe A, 1 dans la B, 1 dans la C et 3 dans la D. L'histogramme correspondant aura donc des barres de hauteur 2, 1, 1 et 3, respectivement.

À vous de jouer !

H5P

Complétez la fonction suivante pour qu'elle trace un histogramme de la variable **height** du jeu de données **bio.**

chart(data = [ ], ~[ ]) +

[ ]()

Vérifier

Vous pouvez interpréter votre histogramme sur base des **modes**[8] et de la **symétrie**[9] de ces derniers. Un histogramme peut être **unimodal** (un seul mode), **bimodal** (deux modes) ou **multimodal** (plus de deux modes). En général, s'il y a plus d'un mode, nous pouvons suspecter que des sous-populations distinctes existent dans les données (par exemple des différences morphométriques entre mâles et femelles pour une espèce au dimorphisme sexuel marqué). En effet, l'allure d'un histogramme qui représente une population homogène est pratiquement toujours unimodal.

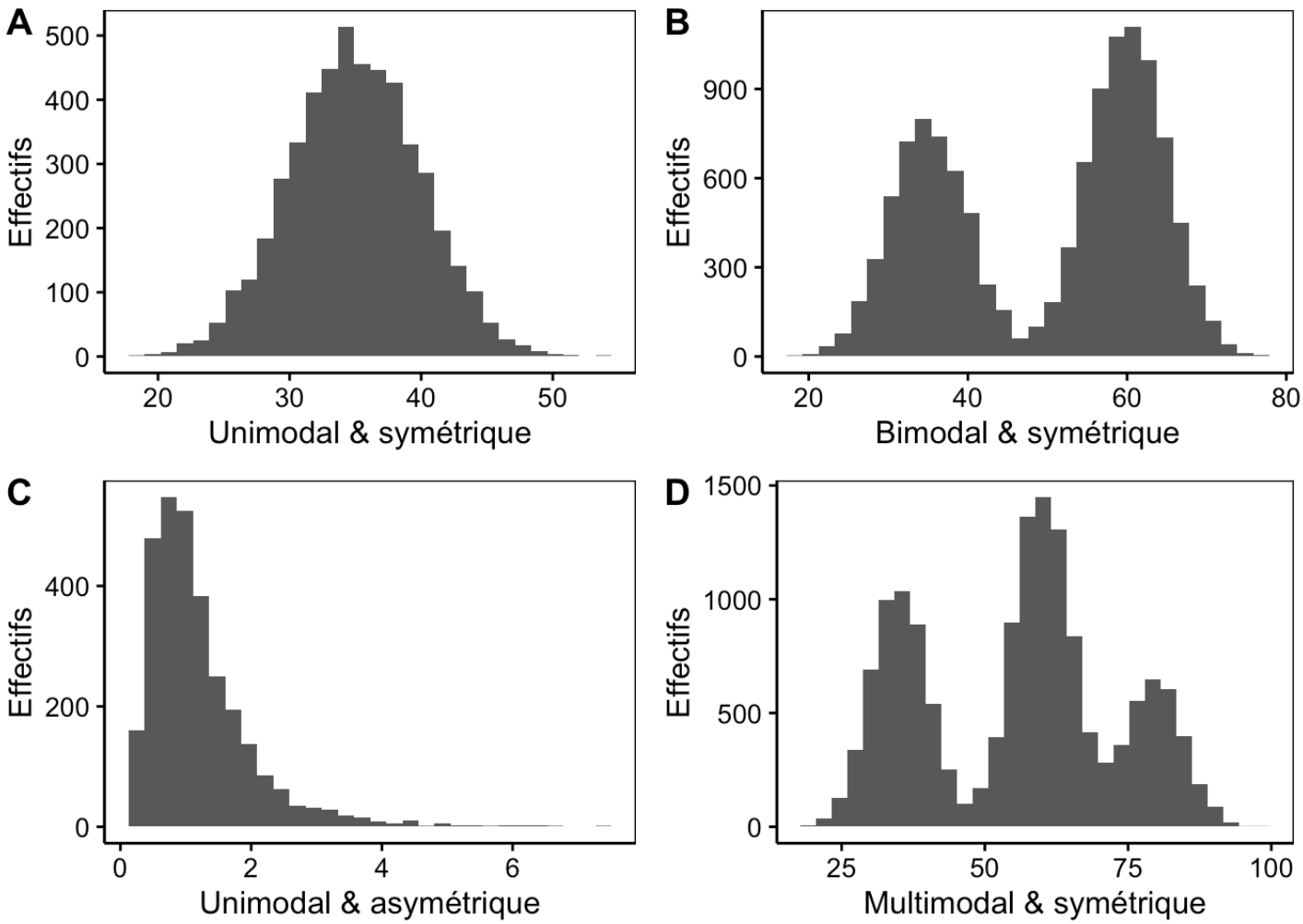
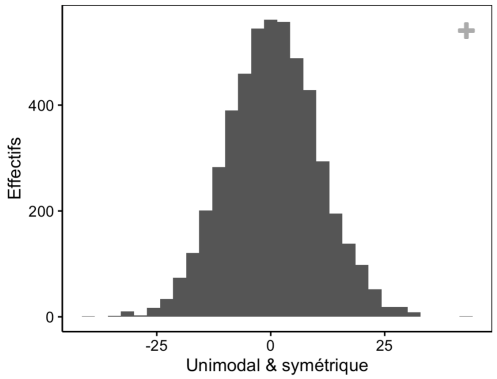


Figure 2.3: Histogrammes montrant les modes et symétries : A. histogramme unimodal et symétrique, B. histogramme bimodal et asymétrique, C. histogramme unimodal et asymétrique, D. histogramme multimodal et symétrique.

À vous de jouer !



Qualifiez la distribution mentionnée ci-dessus en termes de symétrie et de mode.

- [ ] Unimodale
- [ ] Symétrique
- [ ] Multimodale
- [ ] Asymétrique

Afficher la réponse

2.2.1 Nombre de classes

Vous devez être particulièrement vigilant lors de la réalisation d'un histogramme aux classes définies pour ce dernier.

```r
# Réalisation du graphique précédent
a <- chart(data = zooplankton, ~ size) +
  geom_histogram(bins = 50) +
  ylab("Effectifs")

# Modification du nombre de classes
b <- chart(data = zooplankton, ~ size) +
  geom_histogram(bins = 20) +
  ylab("Effectifs")

c <- chart(data = zooplankton, ~ size) +
  geom_histogram(bins = 10) +
  ylab("Effectifs")

d <- chart(data = zooplankton, ~ size) +
  geom_histogram(bins = 5) +
  ylab("Effectifs")

# Assemblage des graphiques
combine_charts(list(a, b, c, d))
```

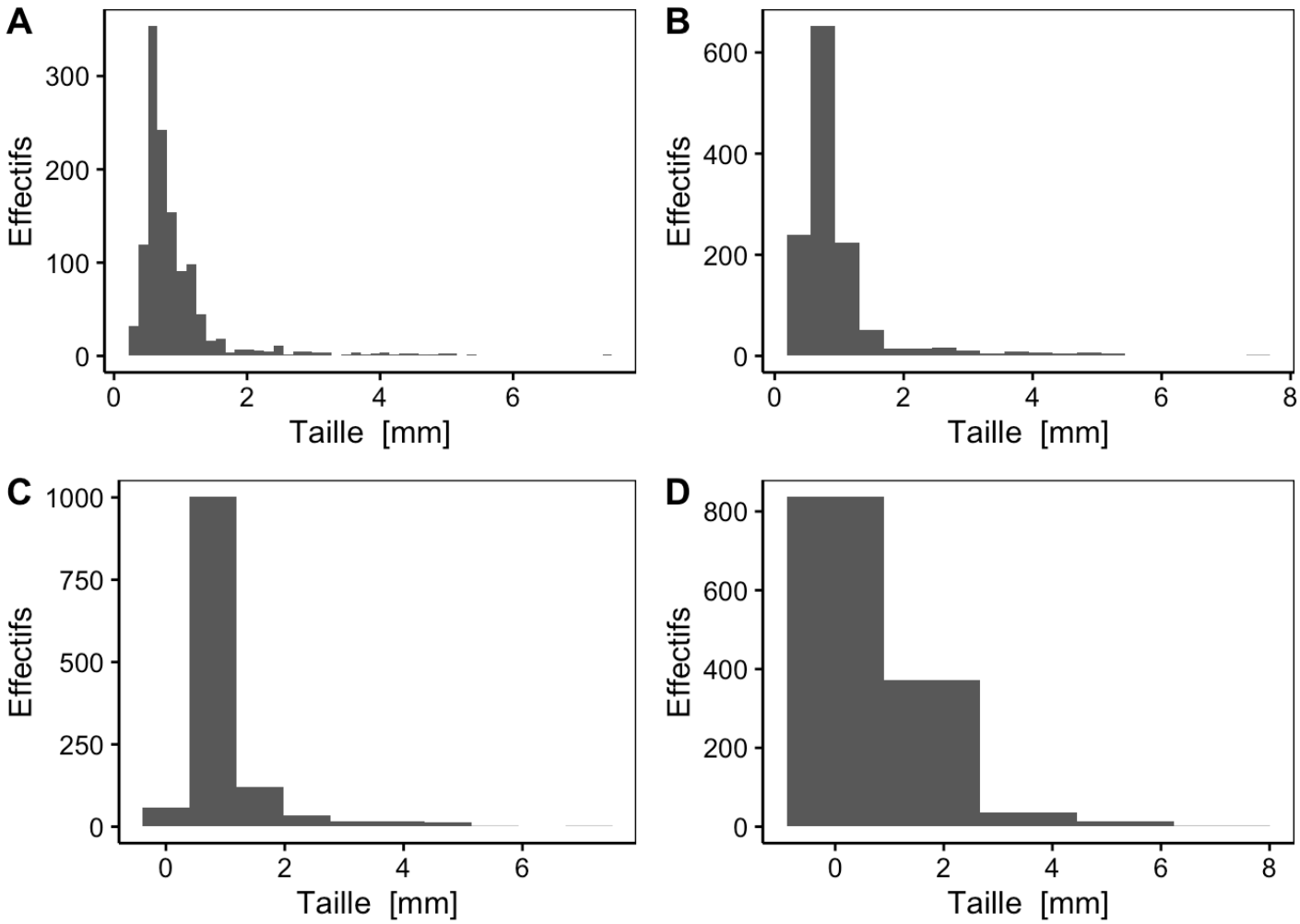


Figure 2.4: Choix des classes. A. histogramme initial montrant la répartition des tailles au sein d'organismes planctoniques. B., C., D. Même histogramme que A, mais en modifiant le nombre de classes.

Comme vous pouvez le voir à la Fig. 2.4, le changement du nombre de classes peut modifier complètement la perception des données au travers de l'histogramme (notez au passage l'utilisation de la fonction `combine_charts()` pour réaliser une figure composite, nous étudierons cette fonction plus en détail dans le prochain module). Le choix idéal est un compromis entre plus de classes (donc plus de détails), et un découpage raisonnable en fonction de la quantité de données disponibles. Si l'intervalle des classes est trop petit, l'histogramme sera illisible. Si l'intervalle des classes est trop grand, il sera impossible de visualiser correctement les différents modes. Dans la figure en exemple, les variantes A et B sont acceptables, mais les C et D manquent de détails.

À vous de jouer !

Cliquez pour lancer ou exécutez dans RStudio

```
BioDataScience1::run_app("A02Sa_histogram") .
```

Pièges et astuces

La SciViews Box propose un **snippet RStudio** pour réaliser un histogramme. Il s'appelle `.cuhist` (pour **c**hart -> **u**nivariate -> **hist**ogram). Entrez ce code dans une zone d'édition R et appuyez ensuite sur la tabulation, et vous verrez le code remplacé par ceci :

```
chart(data = DF, ~VARNUM) +
  geom_histogram(binwidth = 30)
```

L'argument `binwidth=` permet de préciser la largeur des classes. C'est une autre façon de spécifier le découpage en classes, mais vous pouvez naturellement le remplacer par l'argument `bins=` comme nous l'avons fait plus haut, si vous préférez.

Vous avez à votre disposition un ensemble de snippets que vous pouvez retrouver dans l'aide-mémoire sur **SciViews**. N'oubliez pas que vous avez également à votre disposition l'aide-mémoire sur la visualisation des données (**Data Visualization Cheat Sheet**), avec la fonction `ggplot()` à laquelle vous pouvez simplement substituer `chart()` .

2.2.2 Histogramme par facteur

Lors de l'analyse de jeux de données, vous serez amené à réaliser un histogramme par facteur (c'est-à-dire, en fonction de différents niveaux d'une variable qualitative qui divise le jeu de données en sous-groupes).

> Une **variable qualitative** ou **variable facteur** est une variable qui représente des *catégories*. Par exemple, une couleur, le genre, une classe taxonomique… Les différentes catégories possibles pour la variable sont appelées **niveaux** ou **modalités** (*levels* en anglais). Pour le genre, par exemple, nous aurons deux niveaux (principaux) : "homme" ou "femme".
>
> Les variables facteurs peuvent aussi représenter un petit nombre de classes différentes. Une variable sera considérée comme qualitative ou facteur si elle possède moins d'une dizaine de niveaux pour fixer les idées, mais il n'existe pas de limite stricte entre une variable numérique quantitative et facteur qualitatif, en réalité. C'est à votre appréciation, mais aussi en fonction du contexte. Une telle variable s'obtient par découpage d'une **variable numérique**. Par exemple, si au lieu de reprendre la taille d'un animal, nous nous contentons de déterminer s'il est "petit", "moyen" ou "grand". Dans ce cas, il existe un ordre logique entre les niveaux : petit < moyen < grand. La variable sera alors dite "qualitative ordonnée" et sera représentée par un objet **ordered** dans R. Sinon, la variable sera **qualitative non ordonnée** et sera un objet **factor** dans R.
>
> Les variables numériques sont représentées par des nombres, donc **numeric** (des nombre décimaux à virgule flottante) ou **integer** (des entiers) dans R.

> Attention que les variables facteur peuvent très bien être importées comme chaînes de caractères (objet **character**), et il faudra peut-être les convertir à l'aide des fonctions `factor()` ou `ordered()` avant de les utiliser.

Par exemple, dans un jeu de données sur des fleurs d'iris, la variable `species` [10] représente l'espèce d'iris étudiée (trois espèces différentes : *I. setosa*, *I. versicolor* et *I. virginica*).

```
# Importation du jeu de données
iris <- read("iris", package = "datasets", lang = "fr")
tabularise$headtail(iris)
```

Longueur des sépales [cm]	Largeur des sépales [cm]	Longueur des pétales [cm]	Largeur des pétales [cm]	Espèces d'Iris
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
...	...	...	...	...
6.7	3.0	5.2	2.3	virginica
6.3	2.5	5.0	1.9	virginica
6.5	3.0	5.2	2.0	virginica
6.2	3.4	5.4	2.3	virginica
5.9	3.0	5.1	1.8	virginica

Premières et dernières 5 lignes d'un total de 150

```
# Réalisation de l'histogramme par facteur
chart(data = iris, ~ sepal_length %fill=% species) +
  geom_histogram(bins = 25) +
  ylab("Effectifs") +
  scale_fill_viridis_d() # palette de couleur harmonieuse
```

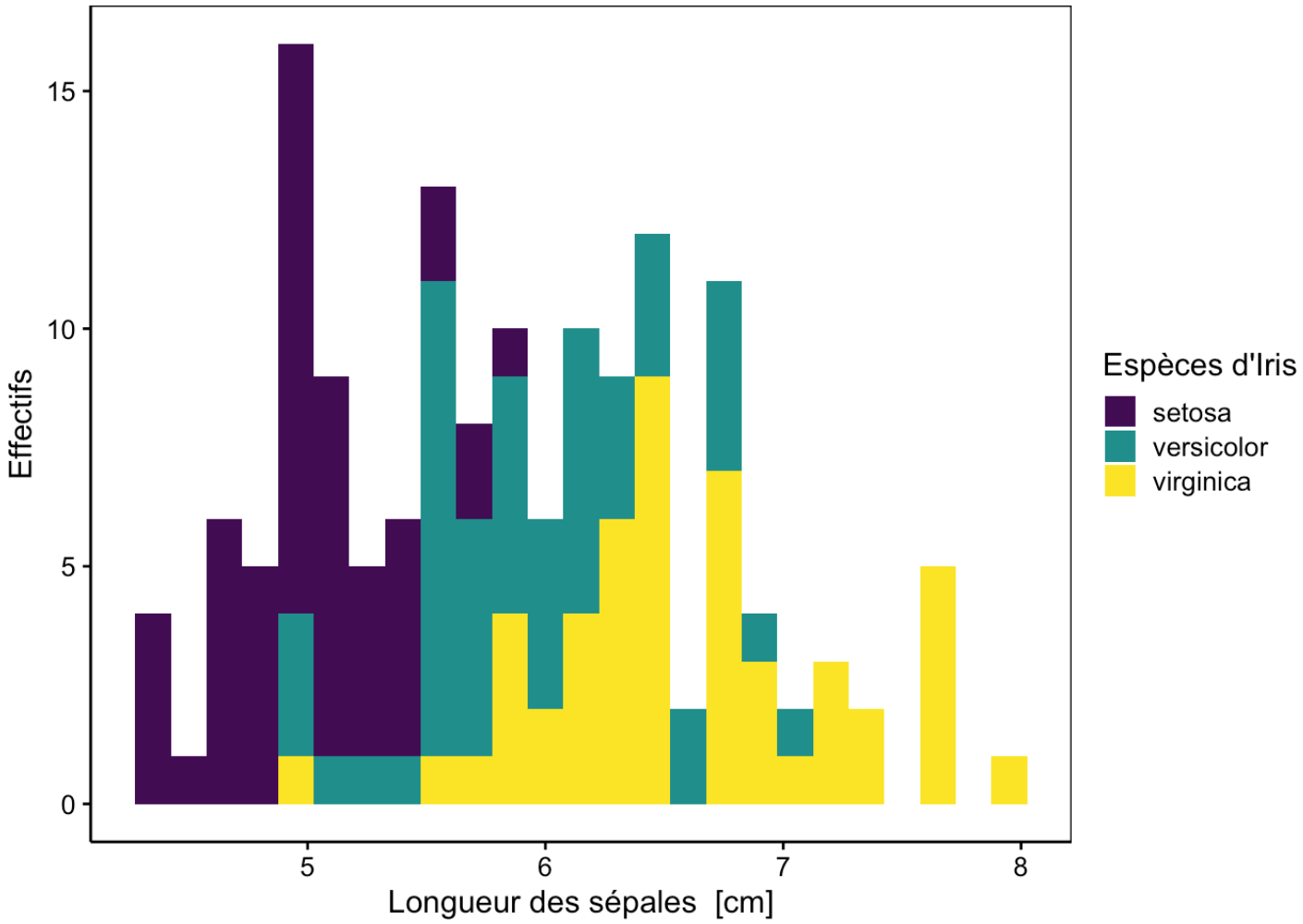


Figure 2.5: Distribution de la longueur des sépales de trois espèces d'iris.

Ici, nous avons tracé un histogramme unique, mais en prenant soin de colorier les barres en fonction de l'espèce. La formule fait toujours intervenir la variable numérique à découper en classes à la droite du tilde `~`, ici `sepal_length`, mais nous y avons ajouté une directive supplémentaire pour indiquer que le remplissage des barres (`%fill=%`) doit se faire en fonction du contenu de la variable `species`.

Nous avons ici un bon exemple d'histogramme multimodal lié à la présence de trois sous-populations (les trois espèces différentes) au sein d'un jeu de données unique. Le rendu du graphique n'est pas optimal. Voici deux astuces pour l'améliorer. La première consiste à représenter trois histogrammes séparés, mais rassemblés dans une même figure. Pour cela, nous utilisons des **facettes** (`facets`) au lieu de l'argument `%fill=%` . Dans `chart()` , les facettes peuvent être spécifiées en utilisant l'opérateur `|` dans la formule. À ce moment-là, le terme qui définit les facettes doit obligatoirement être le tout dernier (par exemple, si vous utilisez aussi `%col=%` , `%fill=%` , etc.)

```
iris <- read("iris", package = "datasets")
chart(data = iris, ~ sepal_length | species) +
  geom_histogram(bins = 25) +
  ylab("Effectifs")
```

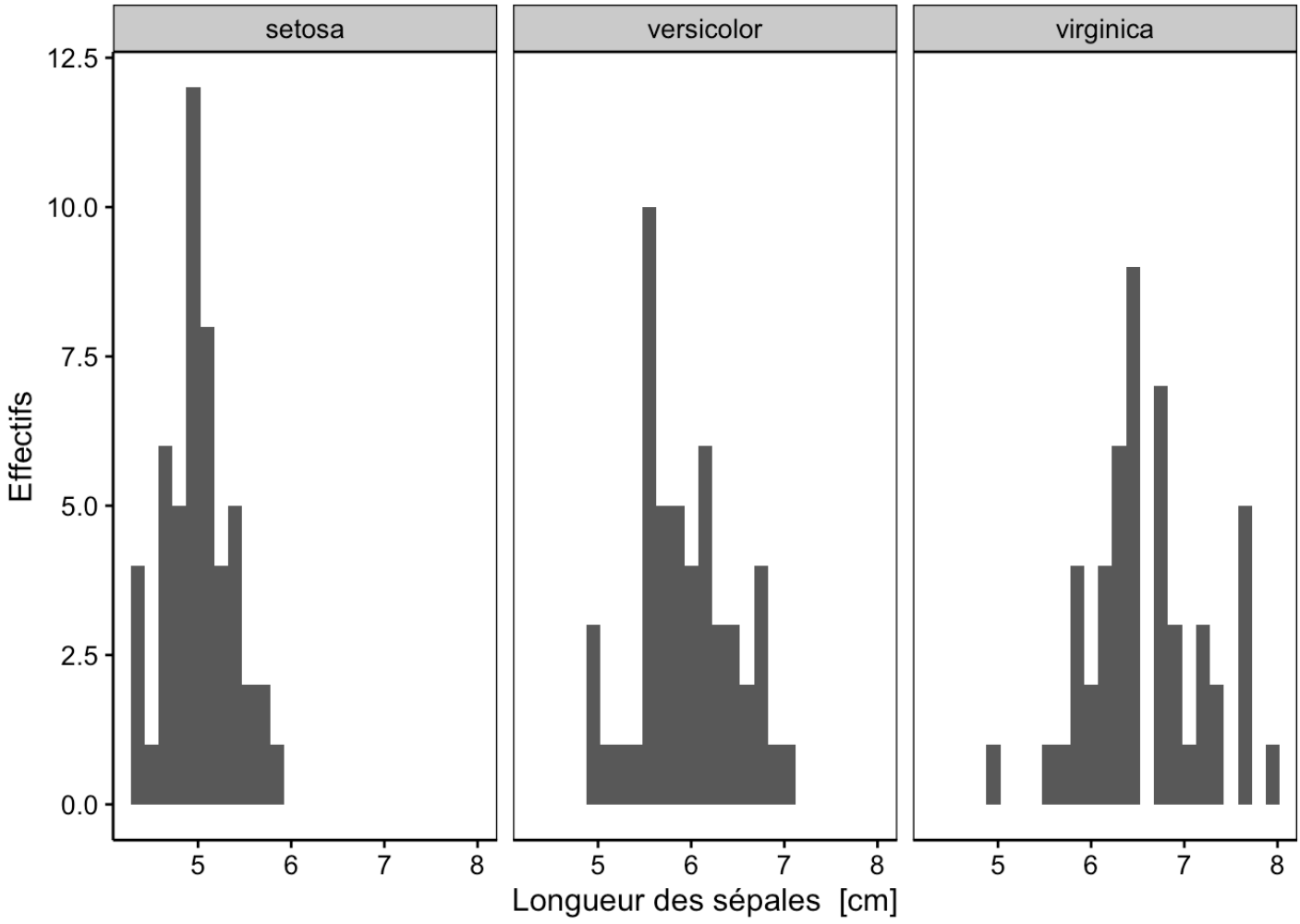


Figure 2.6: Distribution de la longueur des sépales de trois espèces d'iris (en employant les facettes pour séparer les espèces).

L'histogramme est maintenant séparé en trois en fonction des niveaux de la variable facteur `species`. Cela rend la lecture plus aisée. Une seconde solution combine les facettes avec `|` et l'argument `%fill=%` [11]. Il faut ensuite ajouter par-derrière un histogramme grisé de l'ensemble des données.

```
nbins <- 25
chart(data = iris, ~ sepal_length %fill=% species | species) +
  # histogramme d'arrière-plan en gris de toutes les données
  geom_histogram(data = select_(iris, ~-species),
    fill = "grey", bins = nbins) +
  # histogrammes par espèce
  geom_histogram(show.legend = FALSE, bins = nbins) +
  ylab("Effectifs") +
  scale_fill_viridis_d()
```

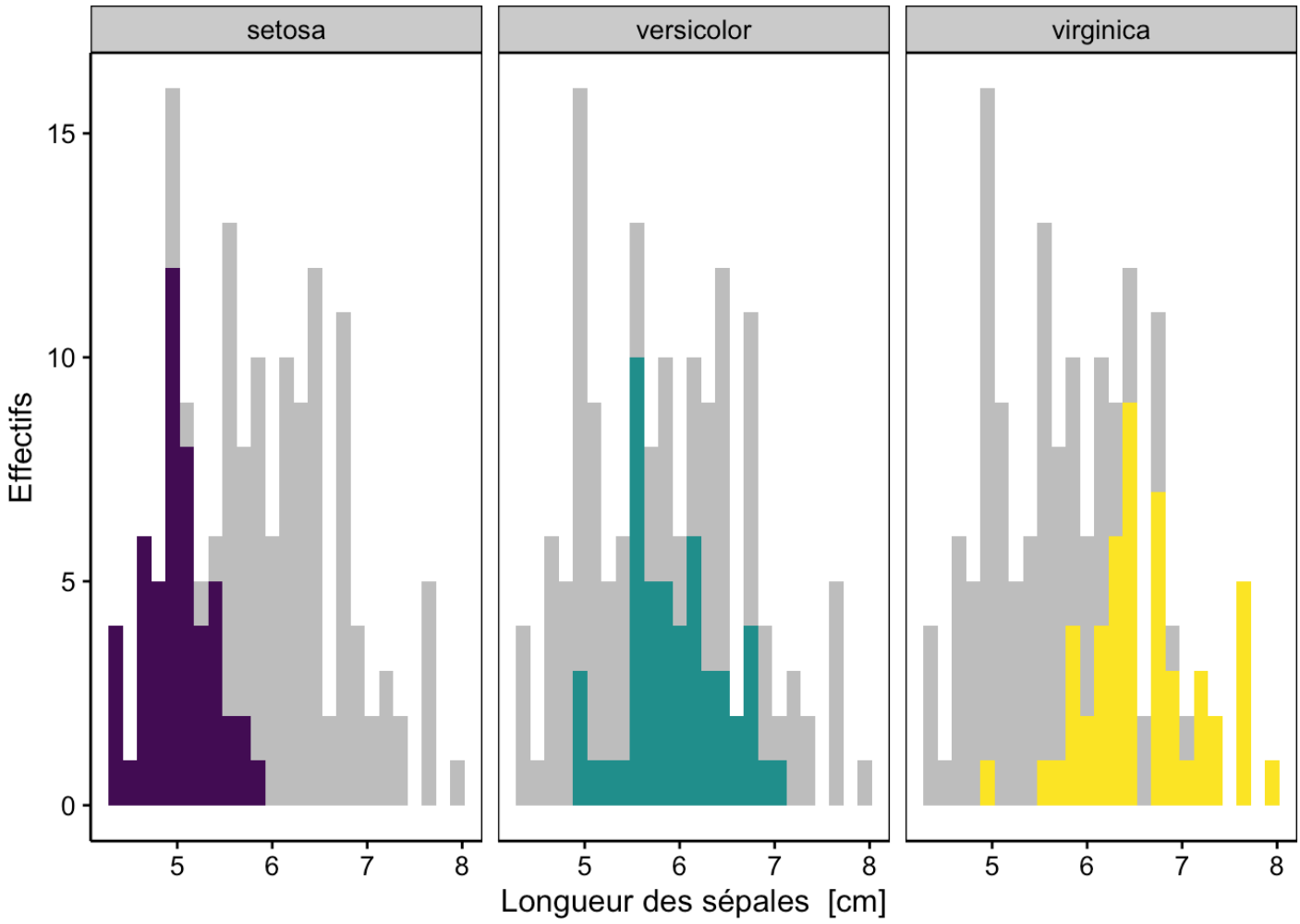


Figure 2.7: Distribution des longueurs de sépales de trois espèces d'iris (avec facettes et histogrammes complets grisés en arrière-plans).

Vous découvrez sans doute que les graphiques réalisables avec R sont modulables à souhait en ajoutant une série d'instructions successives qui créent autant de couches superposées dans le graphique. Cette approche permet de réaliser quasiment une infinité de graphiques différents en combinant quelques dizaines d'instructions. Pour s'y retrouver, les fonctions qui ajoutent des couches commencent toutes par `geom_`, et celles qui manipulent les couleurs par `scale_`, par exemple. Vous découvrirez encore d'autres fonctions graphiques plus loin.

À vous de jouer !

Note : Ce tutoriel couvre l'ensemble de la matière de ce module. N'hésitez pas à le réaliser en parallèle de votre lecture.

Effectuez maintenant les exercices du tutoriel **A02Lb_univariate (Graphiques univariés)**.

```
BioDataScience1::run("A02Lb_univariate")
```

6. La **distribution** des données en statistique se réfère à la fréquence avec laquelle les différentes valeurs d'une variable s'observent.↩
7. Une variable numérique est **découpée en classes** en spécifiant différents intervalles, et ensuite en dénombrant le nombre de fois que les observations rentrent dans ces intervalles.↩
8. Les **modes** d'un histogramme correspondent à des classes plus abondantes localement, c'est-à-dire que les classes à gauche et à droite du mode comptent moins d'occurrences que lui.↩
9. Un histogramme est dit **symétrique** lorsque son profil à gauche est identique ou très similaire à son profil à droite autour d'un mode.↩
10. Attention : le jeu de donnée `iris` est un grand classique dans R, mais lorsqu'il est chargé à l'aide de la fonction `read()` du package {data.io}, le nom de ses variables est modifié pour suivre la convention "snake_case" qui veut que seules des lettres minuscules soient utilisées et que les mots soient séparés par un trait souligné `_`. Ainsi, dans le jeu de données d'origine, les variables sont nommées `Petal.Length` ou `Species`. Ici, ces mêmes variables se nomment `petal_length` et `species`.↩
11. Astuce proposée ici.↩

2.3 Graphique de densité

L'histogramme n'est pas le seul outil à votre disposition. Vous pouvez également utiliser le **graphique de densité** qui se présente un peu comme un histogramme lissé. Le passage d'un histogramme vers un graphe de densité se base sur une **estimation par noyau gaussien**[12]

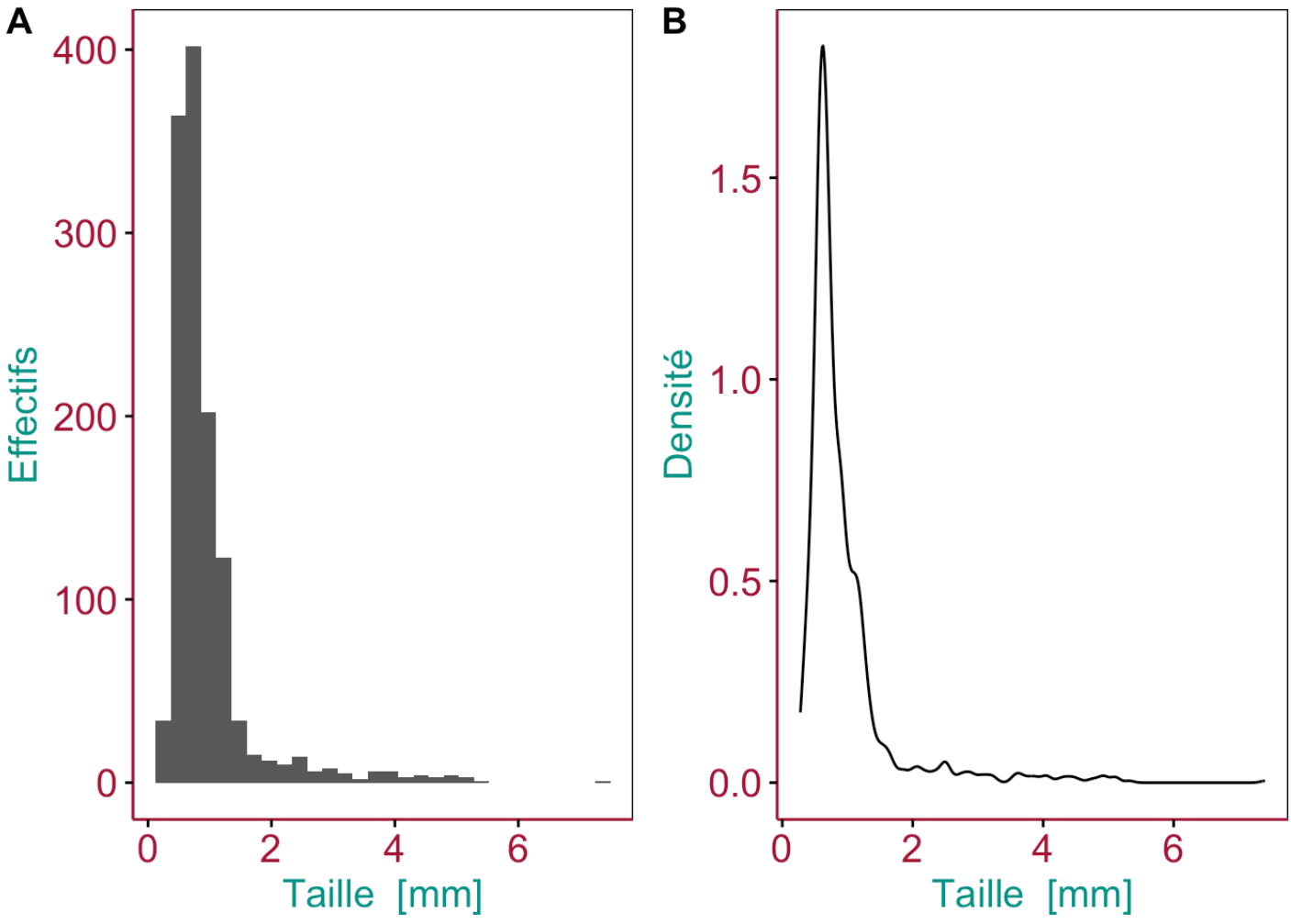


Figure 2.8: A. Histogramme et B. graphique de densité montrant la distribution de la taille de zooplancton étudié par analyse d'image. Les couleurs pour les axes et les labels servent à les mettre en évidence, mais en pratique ils seront noirs.

Comme pour les autres graphiques, veillez à soigner les indications qui permettent d'interpréter le graphique. Outre la courbe de densité ici en noir, il faut :

- Les axes avec les graduations (en rouge)
- les labels des axes, et les unités pour l'axe des abscisses (en bleu)

Les instructions en R pour produire un graphique de densité avec la fonction `chart()` sont :

```
chart(data = zooplankton, ~ size) +
  geom_density() +
  ylab("Densité")
```

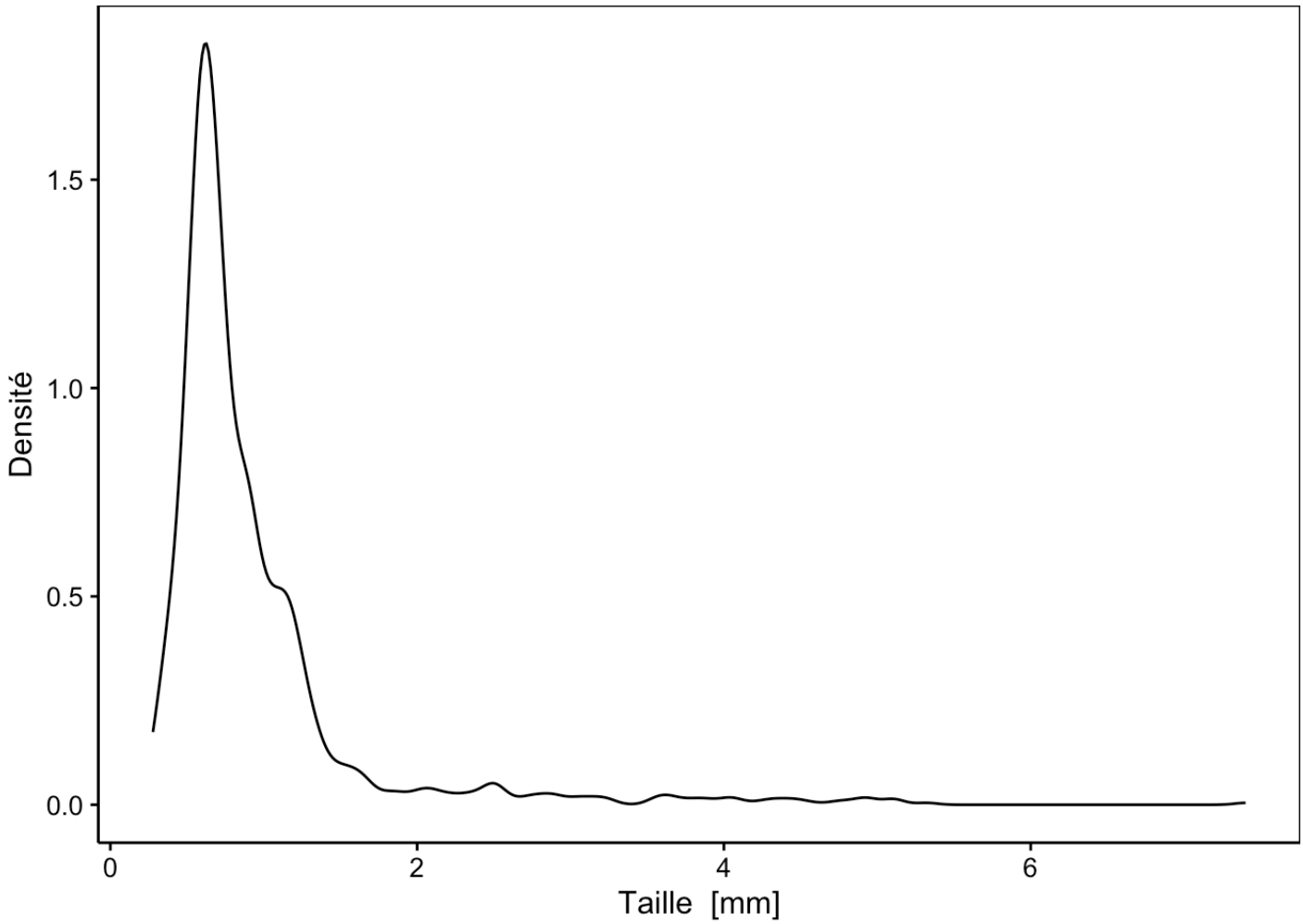


Figure 2.9: Distribution des tailles dans l'échantillon de zooplancton.

Ici, nous utilisons donc la fonction `geom_density()` .

À vous de jouer !

Complétez la fonction ci-dessous pour créer un graphique de densité. Cette fonction gère la variable '**height**' du tableau de données '**bio**'.

[ ]([ ] = bio, [ ]height) +

[ ]()

Vérifier

Réalisez le travail **A02Ia_distributions**.

Travail individuel pour les étudiants inscrits au cours de Science des Données Biologiques I : visualisation à l'UMONS à terminer avant le 2025-10-13 23:59:59.

Initiez votre projet GitHub Classroom

Voyez les explications dans le fichier `README.md` .

12. L'opération effectuée pour passer d'un histogramme à une courbe de densité consiste effectivement à lisser les pics plus ou moins fortement dans l'histogramme de départ.↩

2.4 Diagramme en violon

Le graphique en violon ou diagramme en violon est constitué de deux graphiques de densité en miroir. Le résultat fait penser un peu à la silhouette d'un violon pour une distribution bimodale. Cette représentation est visuellement très convaincante lorsque la variable étudiée contient suffisamment d'observations pour permettre de déterminer précisément sa distribution (plusieurs dizaines ou centaines d'individus mesurés).

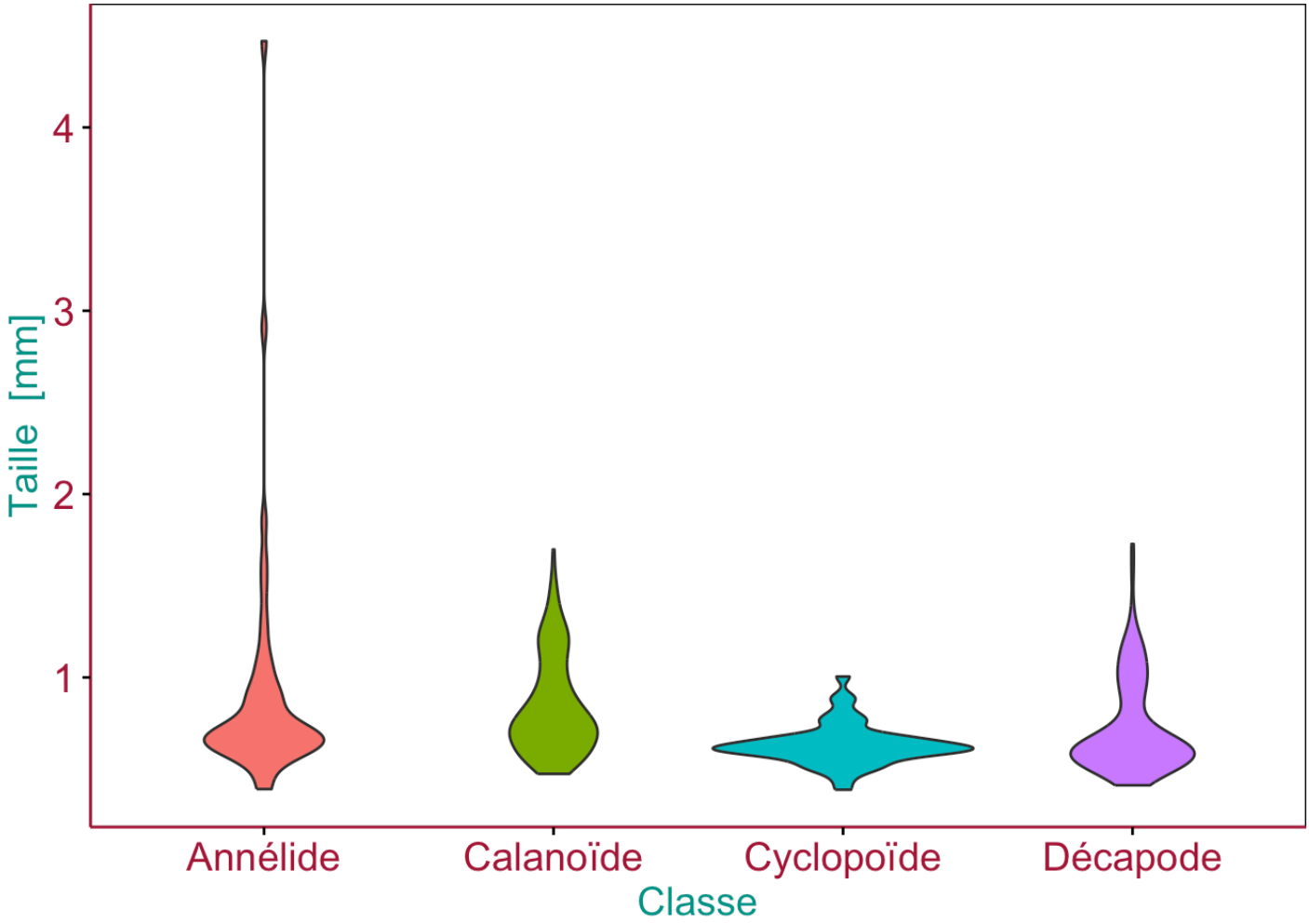


Figure 2.10: Graphe en violon de la distribution de la taille en fonction des groupes taxonomiques dans un échantillon de zooplancton. Les couleurs pour les axes servent à mettre des éléments en évidence. En pratique, les axes seront noirs.

Les instructions en R pour produire un diagramme en violon à l'aide de la fonction `chart()` sont présentées ci-dessous. Comme le jeu de données `zooplankton` contient 17 classes différentes, nous décidons ici d'en extraire un sous-tableau ne contenant que les quatre classes de crustacés copépodes à l'aide d'une fonction `filter_()`. Pour l'instant, retenez simplement que ce genre de remaniement de tableau est possible. Nous étudierons ces fonctions plus en détail dans les modules 4 et 5. Notre tableau réduit à quatre classes se nomme `zooplankton_sub` et c'est à partir de lui que nous réalisons notre graphique.

```
# Importation du jeu de données
zooplankton <- read("zooplankton", package = "data.io", lang = "FR")
# Réduction du jeu de données
zooplankton_sub <- filter_(zooplankton,
  ~class %in% c("Annélide", "Calanoïde", "Cyclopoïde", "Décapode"))
# Réalisation du graphique
chart(data = zooplankton_sub, size ~ class) +
  geom_violin()
```

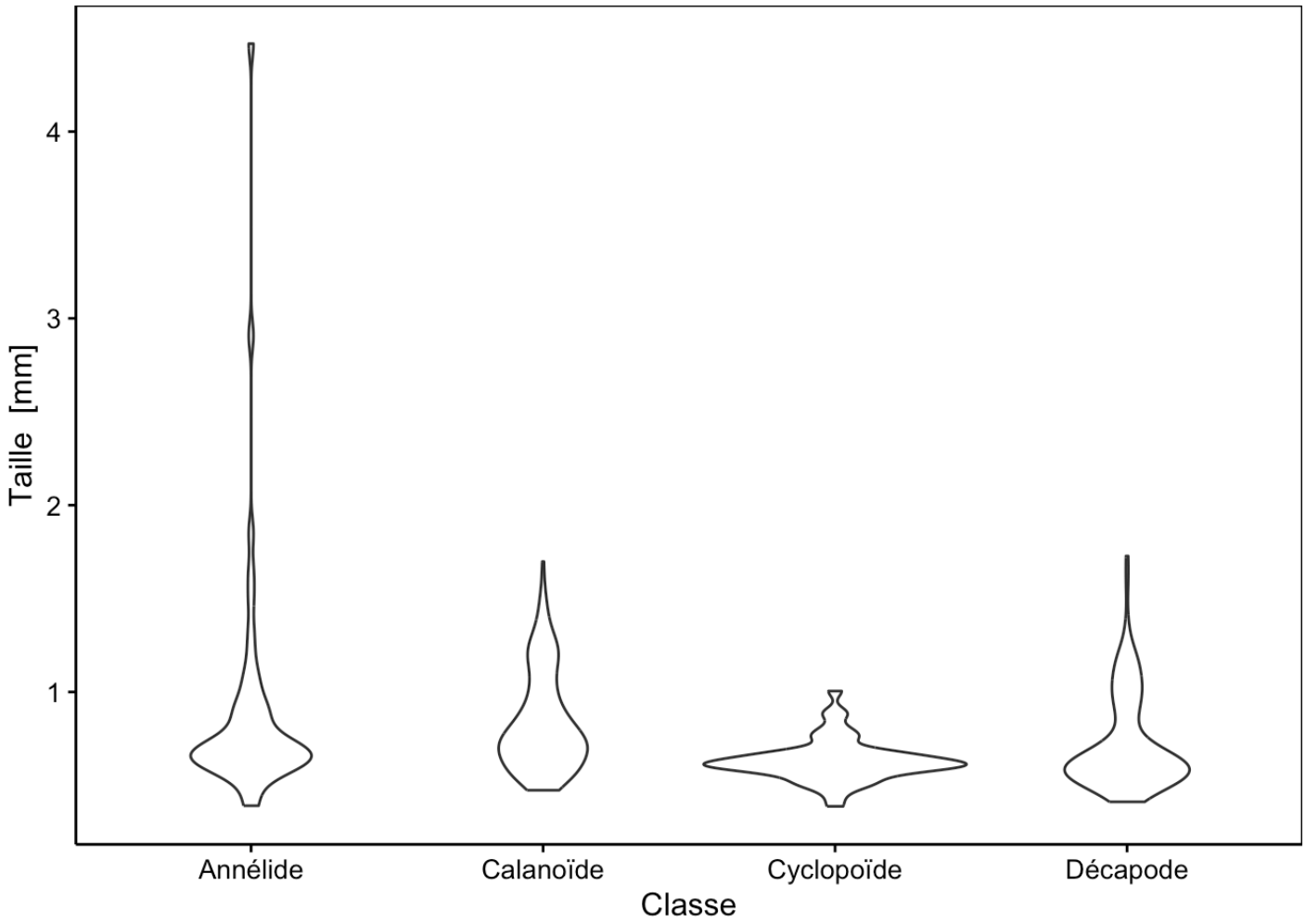
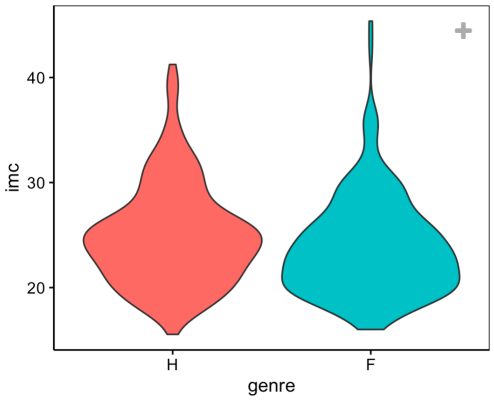


Figure 2.11: Distribution des tailles pour quatre groupes taxonomiques de zooplancton.

Ici, la formule fournie à `chart()` indique la variable numérique à représenter par un graphe de densité *dans le terme de gauche*, et la variable facteur qui découpe l'échantillon en classes *à droite* : `YNUM (size) ~ XFACT (class)`. Pour réaliser un graphique en violon, vous devez ensuite ajouter la fonction `geom_violin()`. Vous pouvez aussi utiliser `%fill=%` pour colorer vos différents graphes en fonction de la variable facteur également, comme dans la Fig. 2.10.

À vous de jouer !



Remplacez XXXXX par la bonne formule afin d'obtenir le graphique ci-dessus.

```
chart( data = bio, XXXXX) +
        geom_violin(show.legend = FALSE)
```

- ○ imc ~ genre %fill=% genre
- ○ ~ genre %fill=% genre
- ○ genre ~ imc %fill=% imc
- ○ imc ~ genre

Afficher la réponse

Pièges et astuces

Parfois, les labels sur l'axe des abscisses d'un diagramme en violon apparaissent trop rapprochés et se chevauchent, comme ci-dessous.

```
chart(data = zooplankton, size ~ class) +
  geom_violin()
```

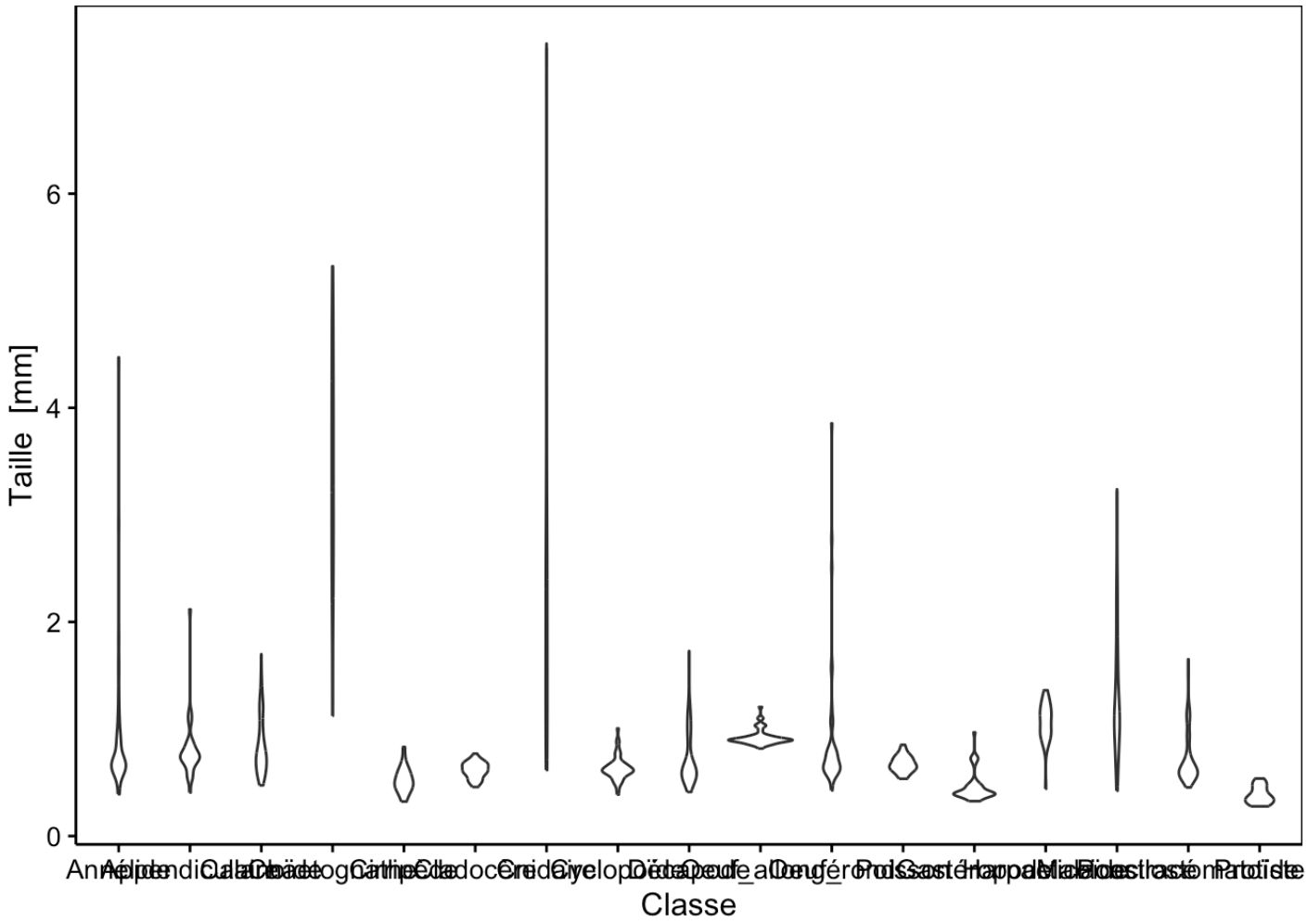


Figure 2.12: Distribution de tailles des 17 classes d'organismes planctoniques (diagramme en violon).

La fonction `coord_flip()` permute les axes. Ainsi les libellés ne se chevauchent plus sur l'axe des ordonnées.

```
chart(data = zooplankton, size ~ class) +
  geom_violin() +
  coord_flip()
```

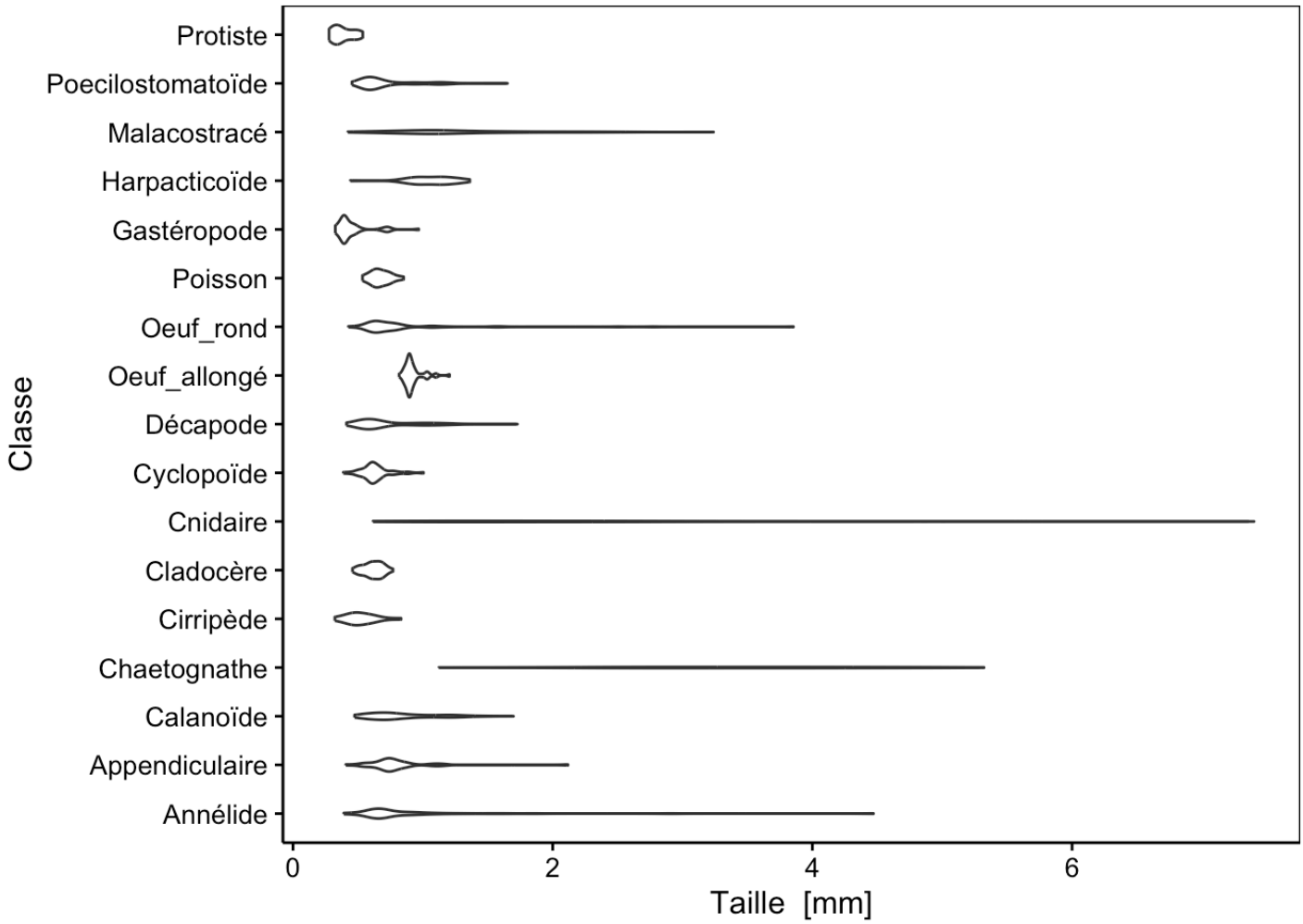


Figure 2.13: Distribution de tailles des 17 classes d'organismes planctoniques (diagramme en violon avec l'ajout de la fonction `coord_flip()`).

La fonction `geom_violin()` a aussi un argument `orientation=` qui est plus pratique pour inverser les axes directement (à utiliser préférentiellement, mais attention à bien penser d'inverser les variables dans la formule également `class ~ size` au lieu de `size ~ class`) :

```
chart(data = zooplankton, class ~ size) +
  geom_violin(orientation = "y")
```


Figure 2.14: Distribution de tailles des 17 classes d'organismes planctoniques (diagramme en violon avec `orientation = "y"`).

Le package {ggridges} offre une seconde solution basée sur le principe de graphique de densité avec la fonction `geom_density_ridges()` qui crée un graphique en lignes de crêtes. **Attention : remarquez que la notation est ici inverse du diagramme en violon vertical, soit `XFACT (class) ~ YNUM (size)` !**

```
chart(data = zooplankton, class ~ size) +
  ggridges::geom_density_ridges()
```

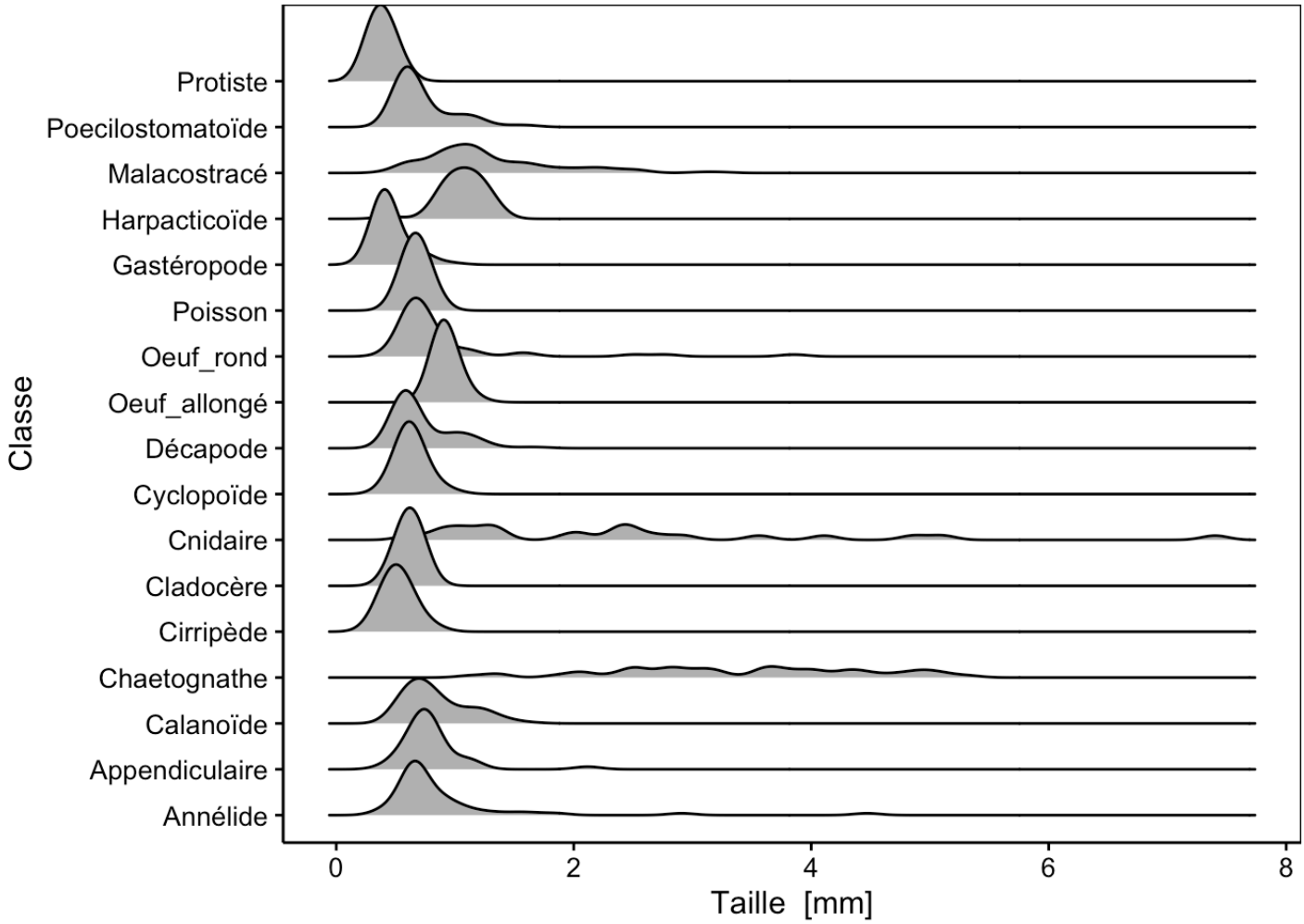


Figure 2.15: Distribution des tailles des 17 classes d'organismes planctoniques (sous forme de graphique en lignes de crêtes).

2.5 Visualiser des distributions

En pratique, vous ne représenterez pas systématiquement tous ces types de graphiques pour toutes les variables. Il faudra choisir le graphique le plus adapté à la situation. La plupart du temps, cela se fait de manière *itérative* : vous essayez diverses variantes, vous les comparez, et vous gardez celle(s) qui visualisent le mieux les données dans le cas particulier de votre étude.

Pour en savoir plus

- Si vous avez encore du mal avec la compréhension de l'histogramme, voyez cette vidéo qui vous montre comment le construire à la main.
- Dans la section "How to build a histogram" de cette page, vous verrez une animation qui visualise étape par étape la construction d'un histogramme (en anglais).
- Les histogrammes à classes de largeurs variables.

2.6 Travail collaboratif

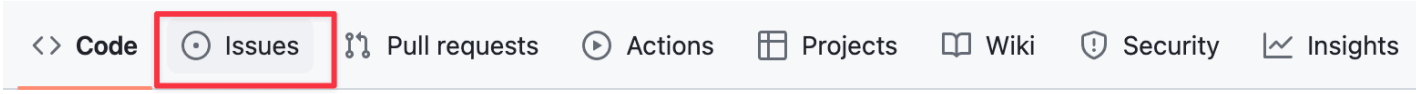
Dans le module précédent, vous avez travaillé seul sur un projet en utilisant **git** et **GitHub**. Vous avez commencé par créer une copie locale du projet, appelée **clone**. Ensuite, vous avez effectué des sauvegardes de vos fichiers au sein d'un dépôt, appelées **commit**. Pour finir, vous avez synchronisé votre dépôt local avec votre dépôt distant (sur GitHub) via un **pull** suivi d'un **push**.

Bien que GitHub et git permettent de travailler seul sur ses dépôts, ils sont conçus pour la collaboration. En tant qu'étudiant et futur scientifique, vous allez devoir collaborer abondamment à l'avenir. Il est donc indispensable de savoir utiliser des outils professionnels de collaboration.

La répartition réfléchie des différentes tâches d'un projet est un élément clé dans un travai collaboratif. Vous pourriez être tenté de diviser le travail via des échanges de mails ou encore via tout autre système de discussion instantanée, grand public ou professionnel. Le risque est de ne plus retrouver ces échanges. GitHub a la solution pour vous avec les **issues**. Ces dernières sont liées au dépôt sur lequel vous travaillez. Il est donc simple de retrouver les échanges importants sur un projet.

2.6.1 Issues

Une issue est une zone de discussion attachée à votre dépôt GitHub **pour y discuter de questions techniques et scientifiques strictement en relation avec ce dépôt**. Les issues sont accessibles dans l'entête des dépôts comme le montre l'image ci-dessous.

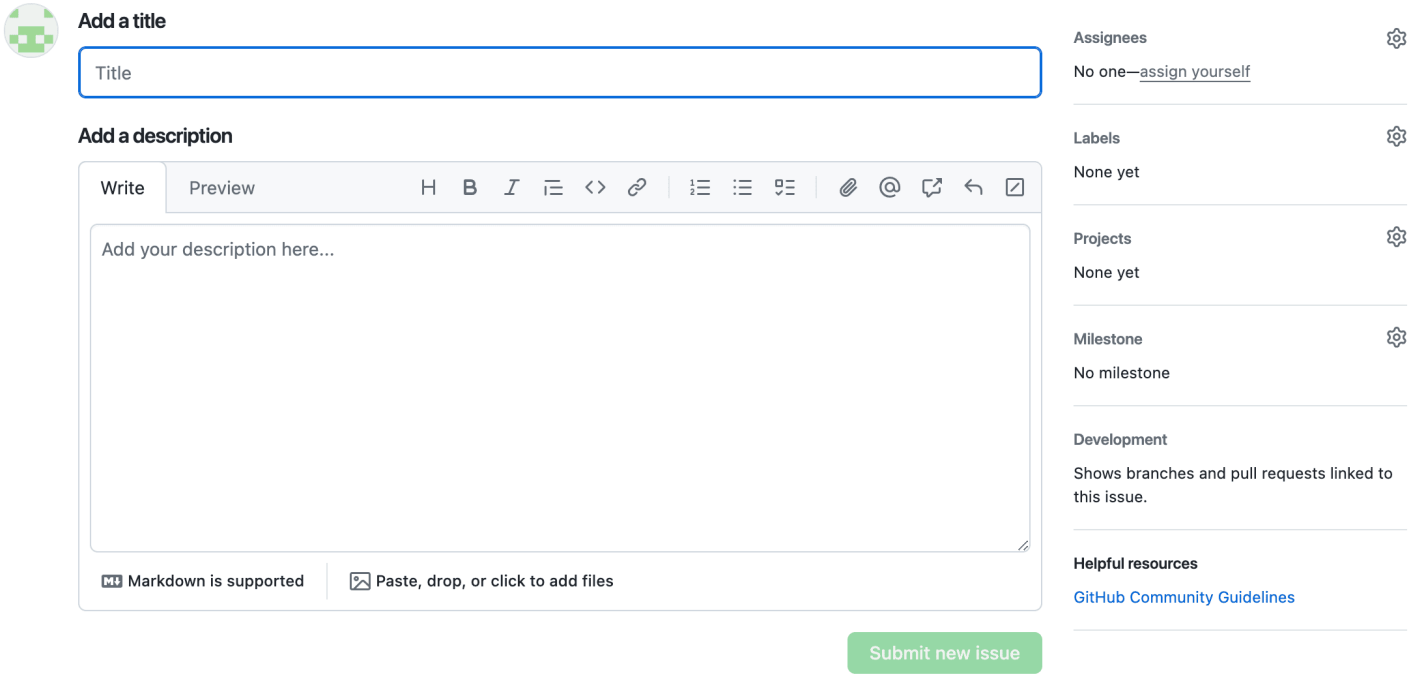


Rédiger correctement une issue nécessite une réflexion et prend du temps. Une question claire permet une réponse précise de la part des collaborateurs, ce qui représente un gain de temps important. Il ne s'agit pas d'un système de chat comme Discord ou Messenger… Vos "memes" et autres gifs animés pour exprimer vos émotions n'y ont pas leur place. De même, des échanges du genre : "tu es là ?", "oui… tu travailles sur quoi ?", "sur le premier graphique, mais sinon, je me cure les ongles, kiss kiss" n'ont absolument pas leur place dans les issues ! Vous avez un dépôt GitHub à disposition qui regroupe toutes les bonnes pratiques pour rédiger une issue : sdd_issues

La rédaction d'une question sous la forme d'une Issue se décompose en quatre étapes :

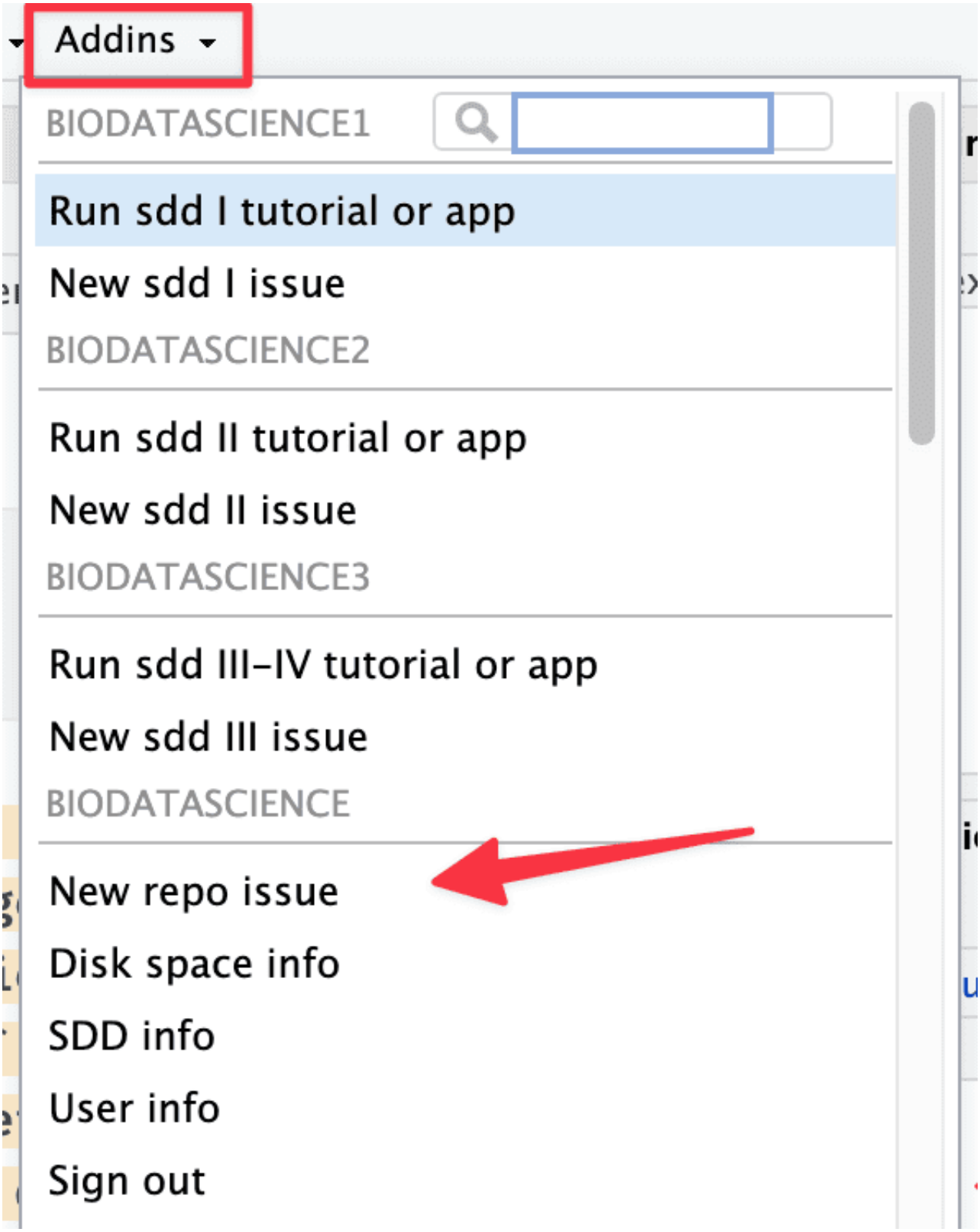
1. Choix d'un titre court qui résume la question
2. Rédaction de la question en respectant la syntaxe Markdown
3. Ajout de labels, de personnes assignées…
4. Soumission de l'issue via le bouton "Submit new issue"

Consultez le document suivant pour tout connaitre de la création d'une issue

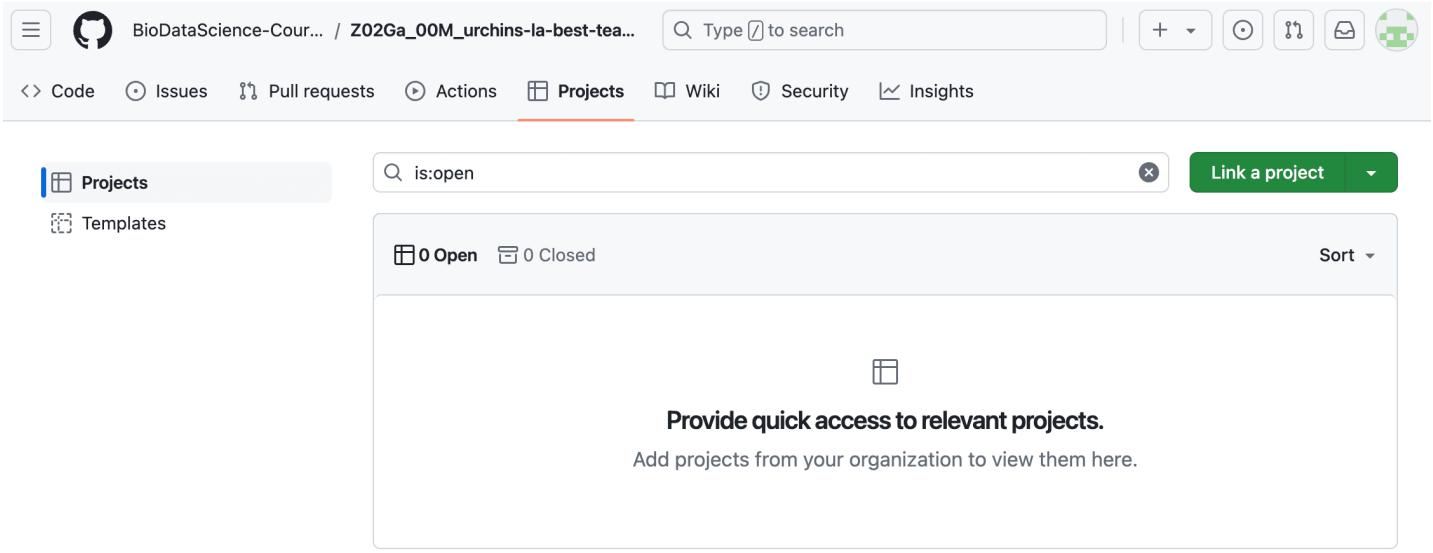


Rappelez-vous qu'une issue doit être **fermée** lorsque le sujet de cette dernière a été débattu et le problème résolu. Ouvrez une issue différente pour *chaque* problème.

Pour accéder encore plus facilement aux issues dans vos projets de groupes, utilisez l'addins dédié :



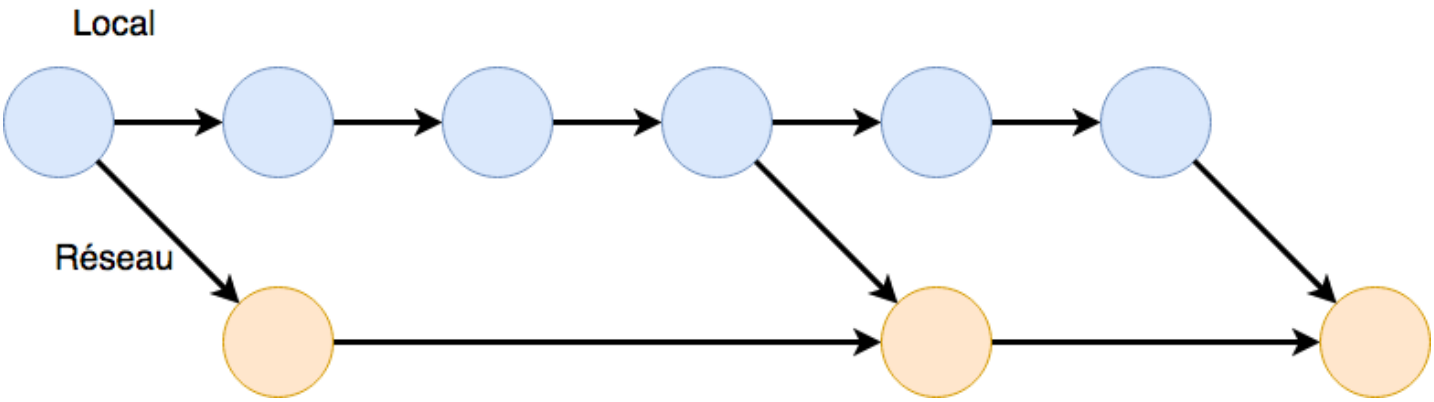
GitHub met à votre disposition plusieurs outils pour améliorer l'efficacité d'un travail en équipe. Dans les issues vous pouvez ajouter des labels à vos questions par exemple. Il existe également dans la section Projects un outil de planification de projets, très utile, bien que nous ne l'utiliserons pas dans le cadre du cours.



2.6.2 Git

Git est un outil puissant, mais complexe. Dans le premier module, nous étions arrivés à la vision schématique suivante. (chaque boule bleue représente un **commit** soit une version enregistrée dans le système et les flèches indiquent d'où provient chaque version et où elle est utilisée ensuite) :

Ce gestionnaire Git peut être couplé à un hébergement sur le cloud, soit pour simplement faire un backup de nos projets, soit pour pouvoir échanger et collaborer. Nous utiliserons GitHub à cette fin dans le cours. Lorsque l'on travaille seul avec GitHub, l'évolution de notre projet ressemblera au schéma ci-dessous :



Représentation des versions successives d'un projet avec GitHub.

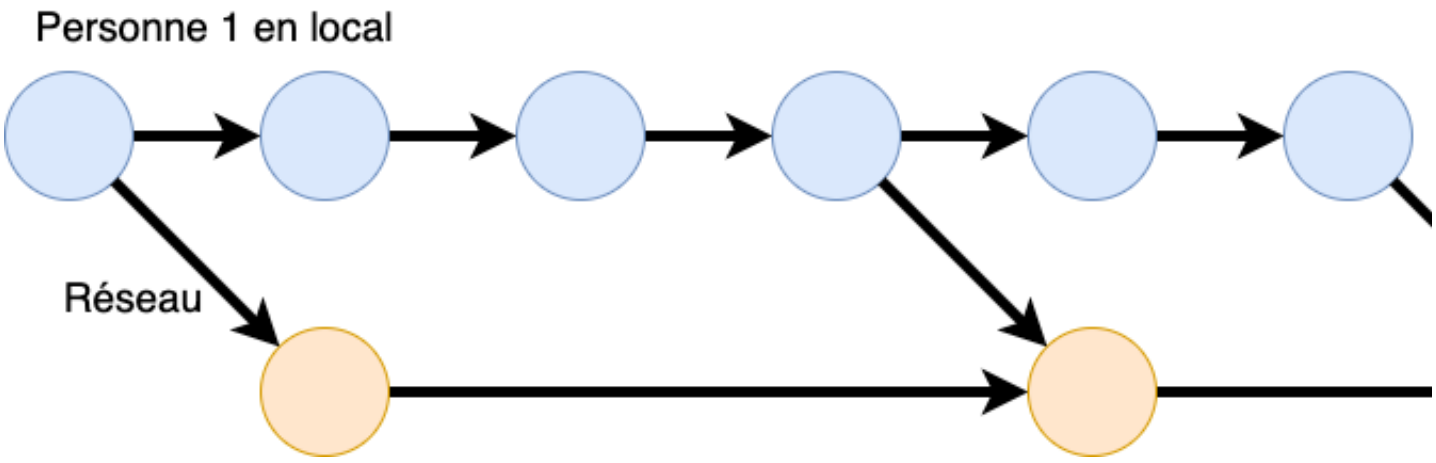
On réalise un envoi (**push**) lorsque l'on souhaite synchroniser nos changements locaux avec la version sur le "cloud". Plusieurs **commits** peuvent être envoyés avec un seul **push** sur le réseau, et c'est d'ailleurs généralement comme cela que l'on procède. L'inverse (rapatrier localement les changements que d'autres collaborateurs ont envoyés sur la version réseau de notre projet) s'appelle faire un **pull**. Par conséquent, synchroniser complètement notre dépôt GitHub avec la version locale consiste à faire les trois actions dans l'ordre : d'abord un **commit** pour créer un nouveau point d'enregistrement de l'état du système et y inclure tous les fichiers qui ont été modifiés, ensuite faire un **pull** pour rapatrier localement les changements qui ont été réalisés par d'autres ou par GitHub lui-même, et enfin, faire un **push** pour envoyer nos propres modifications dans notre projet GitHub. Les autres utilisateurs feront aussi **commit-pull-push** de leur côté.

À vous de jouer !

Note : l'image suivante est interactive. Il vous est maintenant demandé de cliquer dessus pour indiquer quelles flèches représentent une action particulière dans le schéma présenté.

H5P

Sélectionnez les flèches qui représentent un PUSH



L'avantage principal de **GitHub** ne réside pas vraiment dans la possibilité de réaliser une sauvegarde en ligne, mais plutôt dans la possibilité de collaborer avec d'autres personnes présentes sur ce réseau comme l'illustre la figure ci-dessous. Deux scientifiques (les versions locales sur leurs ordinateurs respectifs sont indiquées en bleu pour l'un et en vert pour l'autre) collaborent sur un même projet que l'on appelle un **dépôt** (**repository** en anglais) lorsqu'il est en ligne. Le premier chercheur (boules bleues) initie le dépôt et réalise

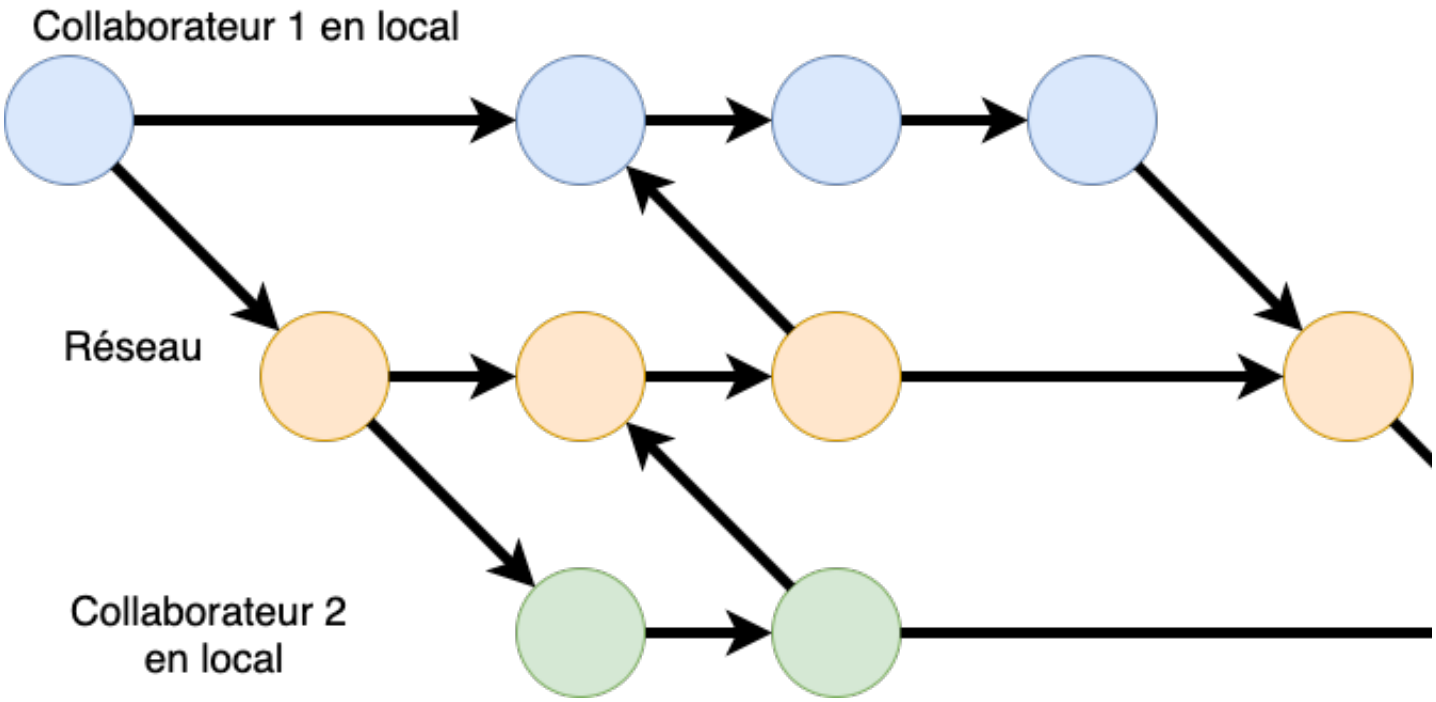
un **push** pour rendre son travail accessible sur le réseau (les versions GitHub sont représentées par des boules orange). Son collaborateur (boules vertes) **clone** ensuite le dépôt sur son ordinateur afin de pouvoir y travailler. Après avoir fait des changements, il réalise également un **push** sur le réseau. Le premier scientifique, avant de travailler à nouveau sur le projet, devra donc réaliser un **pull** pour obtenir en local l'ensemble des modifications fournies par son ou ses collaborateurs. Après ses propres modifications, il devra ensuite effectuer à nouveau un **push**.

À vous de jouer !

Pour être certain que vous ayez bien compris, encore une image interactive à cliquer…

H5P

Sélectionnez les flèches qui représentent un PULL



À vous de jouer !

Effectuez maintenant les exercices du tutoriel **A02Lc_git (Gestion de versions avec git)**.

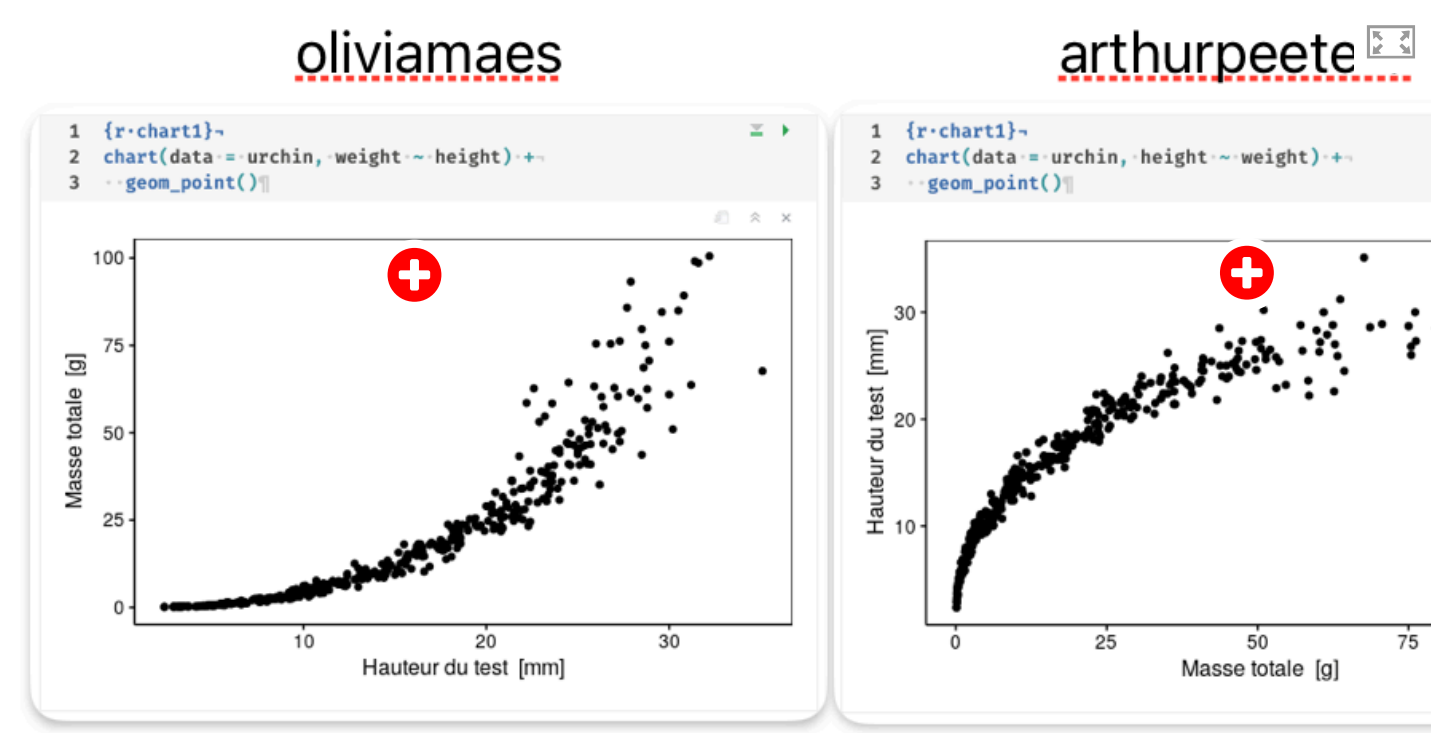
```
BioDataScience1::run("A02Lc_git")
```

2.6.3 Gestion de conflit

Malgré la répartition des tâches, la collaboration entre plusieurs personnes sur un projet commun mène à des conflits. C'est-à-dire les mêmes parties de documents au sein d'un projet qui sont modifiées par deux membres du groupe. Ne redoutez pas les conflits (dans le contexte de GitHub, du moins). Leur gestion appropriée est une compétence que nous allons développer ici.

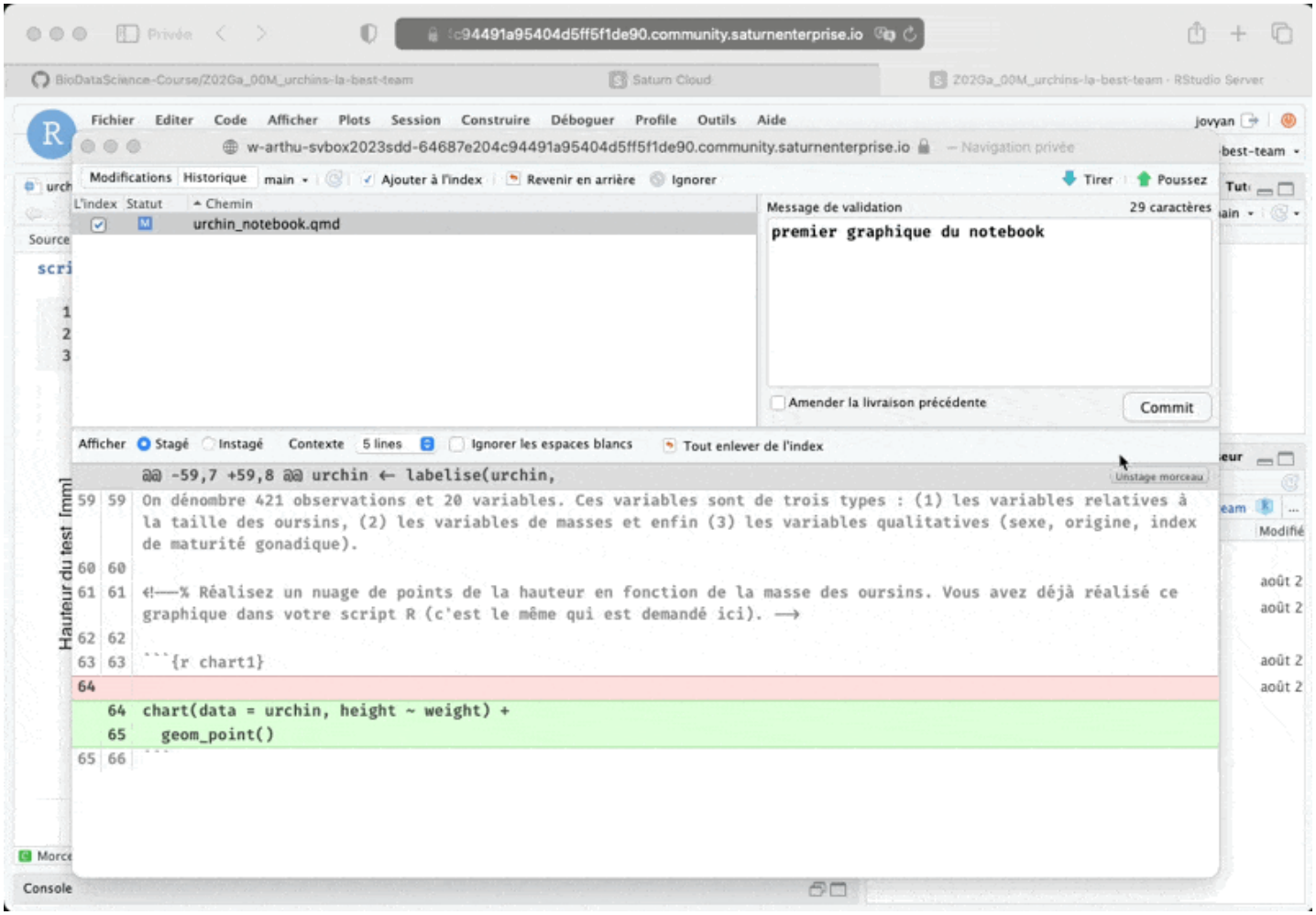
Prenons un cas concret entre des collaborateurs qui étudient deux populations de *Paracentrotus lividus* (Lamarck, 1816). Les collaborateurs sont oliviaMaes, arthurpeeters et GuyliannEngels. Le projet est disponible. Consultez-y par exemple les différents commits afin de suivre l'évolution du projet.

Arthur et Olivia ont édité la même section d'un document.

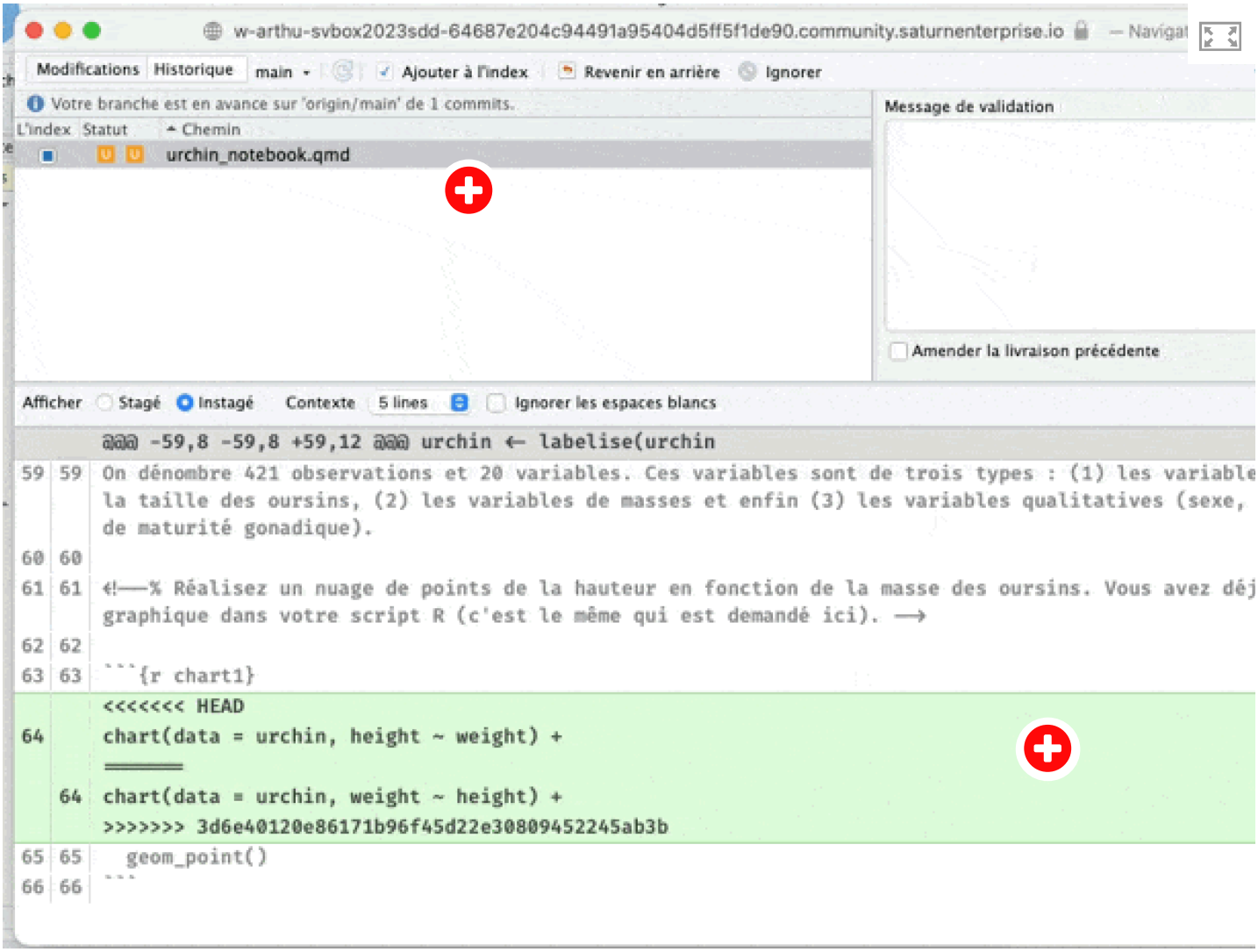


Que va-t-il se passer selon vous ? Lequel des deux collaborateurs va avoir le droit d'écraser le contenu de l'autre ? GitHub n'est pas capable de décider quelle version est la meilleure. Il va donc demander au dernier collaborateur qui souhaite ajouter ces

modifications avec le suite d'instruction, **commit -> pull -> push**, de gérer ce conflit lui-même.



Cette animation montre l'importance de toujours bien lire le message renvoyé par git. Lors du pull, il précise qu'il y a un conflit à gérer dans le fichier `urchin_notebook.qmd` .

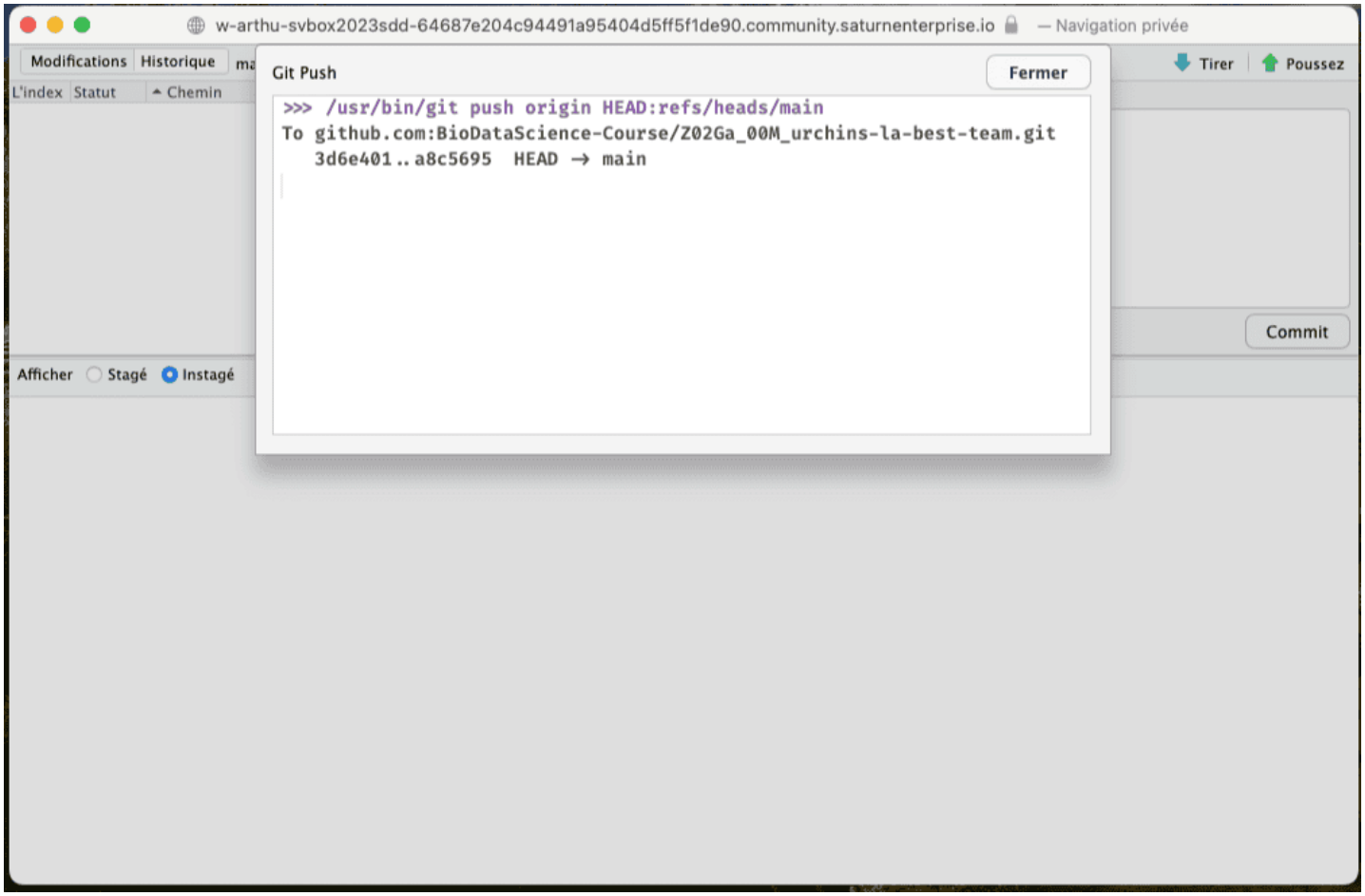


Vous pouvez ensuite vous rendre dans chaque fichier problématique pour retrouver la ou les zones de conflit (utiliser l'outil de recherche avec l'icône loupe dans l'éditeur RStudio et recherchez par exemple `<<<<<`). En fait, GitHub va placer les deux versions de la même section entre des balises spéciales pour vous permettre de les repérer : une série de signes "plus petit que" indique le début de la première version (`<<<<<<<<<<<<<<<<<<<<<<<`). Ensuite, une série de signes "égaux" indique le passage à la seconde version (`=======================`). Enfin, une série de signes "plus grand que" indique la fin de la seconde version (`>>>>>>>>>>>>>>>>>>>>>>>`). Un numéro de commit est aussi indiqué. Dans l'onglet "Git", tous les fichiers qui ont des conflits non résolus apparaissent avec une petite icône orange marquée d'un "U", pour "unresolved conflict". Cela vous aide pour déterminer dans quel(s) fichier(s) se trouve(nt) le(s) conflit(s). Ouvrez chacun de ces fichiers. Ensuite, l'outil de recherche dans l'éditeur vous positionne au début du conflit. Notez bien qu'il peut y avoir **plusieurs zones de conflit dans un**

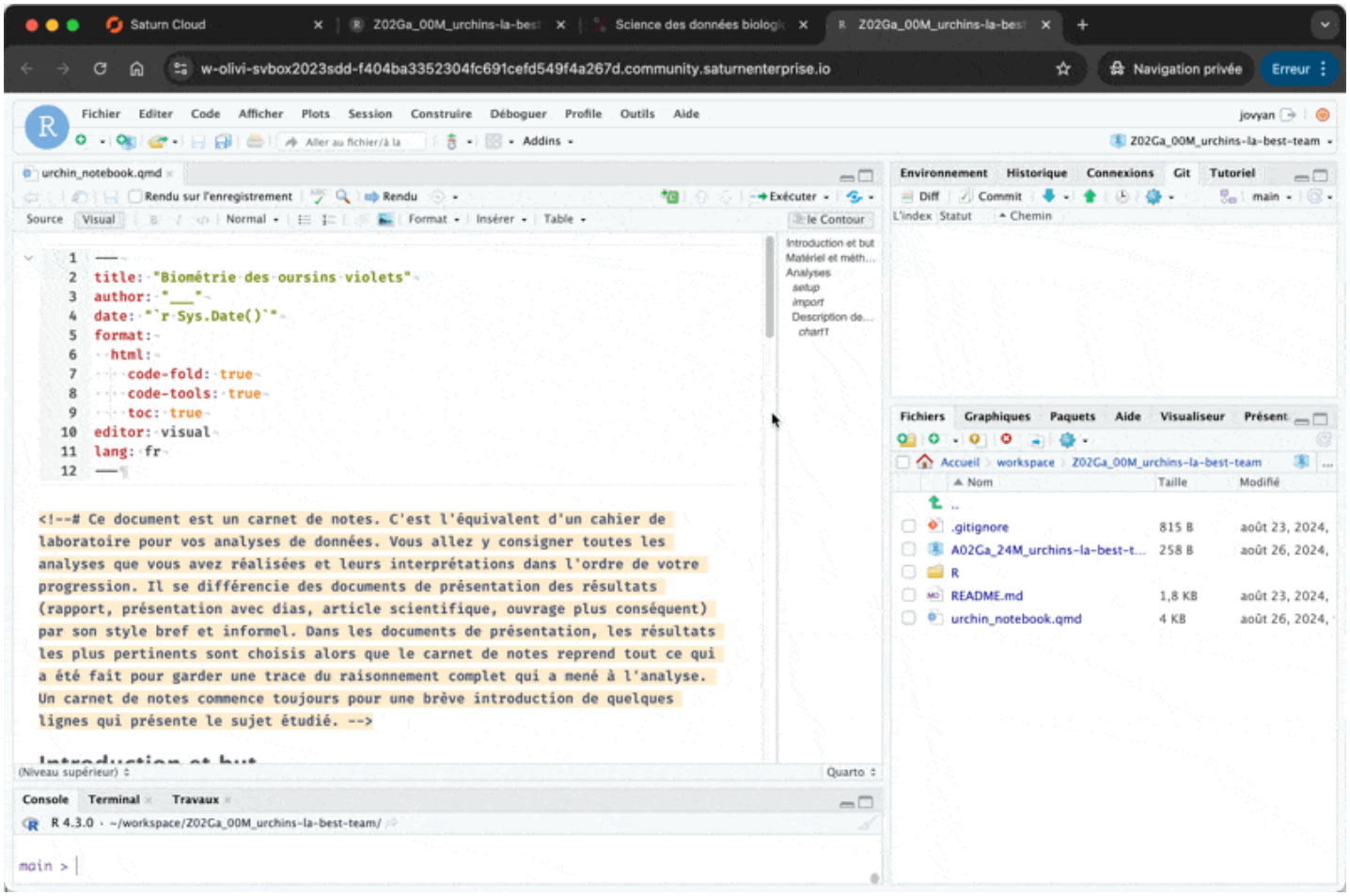
même document. Répétez la recherche jusqu'à ce que RStudio vous dise qu'il ne trouve plus `<<<<<` . Éditez ces zones pour ne garder que la bonne version et éliminez les balises. Ensuite, sauvegardez le document et passez éventuellement au suivant.

La résolution d'un conflit se fait donc en éditant le fichier concerné. Vous devez choisir la partie qui vous semble la plus pertinente et effacer tout le reste, y compris les balises ajoutées par GitHub. Terminez l'opération en enregistrant votre fichier.

La gestion du conflit se termine par la réalisation d'un "commit" avec un message qui indique la résolution du ou des conflits, suivi par un "pull" pour vérifier qu'il n'y a plus de problèmes, et enfin un "push" qui synchronise votre version locale avec GitHub.



Votre ou vos collaborateurs doivent ensuite faire un “pull” de leur côté pour se synchroniser à leur tour. Nous vous conseillons à chaque fois que vous ouvrez un projet pour y contribuer de réaliser un pull afin de récupérer toutes les modifications réalisées par les autres membres du groupe.



Lorsque vous vous lancez dans un projet ou revenez vers lui, il faut toujours faire un “pull” avant de commencer à travailler pour vous assurer que votre copie locale du projet est bien synchronisée avec la dernière version en ligne sur GitHub. C’est d’autant plus important lorsque le dépôt a plusieurs collaborateurs, car il faut traiter immédiatement les conflits et ne surtout pas les laisser s’accumuler.

Lorsque vous avez terminé de travailler dans un projet. Il est conseillé de faire toujours un “commit”, suivi d’un “pull” en vérifiant bien si des modifications sont réalisées qu’il n’y a pas de conflits (sinon, on les règle directement comme expliqué plus haut) et enfin un “push”. **Ne laissez jamais traîner un conflit ! C’est une situation transitoire normale de votre dépôt, mais qui nécessite une intervention rapide pour l’éliminer avant que d’autres “commits” ne viennent rendre la situation encore plus complexe.**

À vous de jouer !

Complétez votre projet de groupe en y ajoutant des graphiques de distribution et en les interprétant.

> Réalisez en groupe le travail **A02Ga_analysis, partie I**.
>
> **Travail en groupe de 4 pour les étudiants inscrits au cours de Science des Données Biologiques I : visualisation à l'UMONS à terminer avant le 2025-12-16 23:59:59.**
>
> **Initiez votre projet GitHub Classroom**
>
> *Voyez les explications dans le fichier* `README.md` *, partie I.*

2.7 Récapitulatif des exercices

Ce module 2 vous a permis de réaliser différents graphiques uni- et bivariés afin de visualiser la *distribution* de variables quantitatives seules ou en fonction des niveaux d'une variable qualitative (facteur). Pour évaluer votre compréhension de cette matière, vous aviez les exercices suivants à réaliser :

- A02La_progression - Progression en R
- A02Ha_mean - Sélection de la fonction adéquate
- A02Hb_geom_histogram - Les fonctions chart() et geom_histogram()
- A02Hc_histogram - Modes et symétries
- A02Sa_histogram - Nombre de classes d'un histogramme
- A02Lb_univariate - Graphiques univariés
- A02Hd_geom_density - La fonction chart() et geom_density()
- A02Ia_distributions - Graphiques de distribution des données
- A02He_geom_violin - La fonction chart() et geom_violin()
- A02Hf_git_push2 - GitHub : repérer les pushs
- A02Hg_git_pull2 - GitHub : repérer les pulls
- A02Lc_git - Autoévaluation sur git
- A02Hh_conflit - Deux versions d'un même fichier
- A02Hi_resolution_conflit - Résolution d'un conflit
- A02Hj_gestion_conflit - Résolution d'un conflit (suite)
- A02Ga_analysis - Analyse de données (partie I, par groupe de 4)

Progression

Module 3 Visualisation III

Objectifs

- Être capable de réaliser différents graphiques pour représenter des variables facteurs comme le graphique en barres, ou le graphique en camembert dans R avec la fonctior `chart()`
- Comprendre et utiliser la boite à moustaches pour synthétiser la distribution de données numériques
- Arranger différents graphiques dans une figure unique
- **De manière optionnelle**, découvrir différents systèmes graphiques (graphiques de base, {lattice}, {ggplot2}) et les comparer entre eux

Prérequis

Assurez-vous de bien maîtriser les bases relatives à la représentation graphique abordée dans le module 2 et d'être à l'aise dans l'utilisation des outils logiciels (SciViews Box, RStudio, R Markdown, Git & GitHub).

3.1 Graphique en barres

Le graphique en barres (on dit aussi graphique en bâtons) compare les effectifs pour différents niveaux (ou modalités) d'une variable qualitative ou facteur. La différence avec l'histogramme est donc subtile et tient au fait que, pour l'histogramme, nous partons d'une variable quantitative qui est découpée en classes.

3.1.1 Effectifs par facteur

La question du nombre et/ou de l'intervalle des classes ne se pose pas dans le cas du graphique en barres. Par défaut, les barres seront séparées les unes des autres par un petit espace vide pour bien indiquer visuellement qu'il n'y a pas continuité entre les classes (dans l'histogramme, les barres sont accolées les unes aux autres pour matérialiser justement cette continuité).

La formule que vous utiliserez, ici encore, ne fait appel qu'à une seule variable et s'écrira donc :

$$\sim variable\ facteur$$

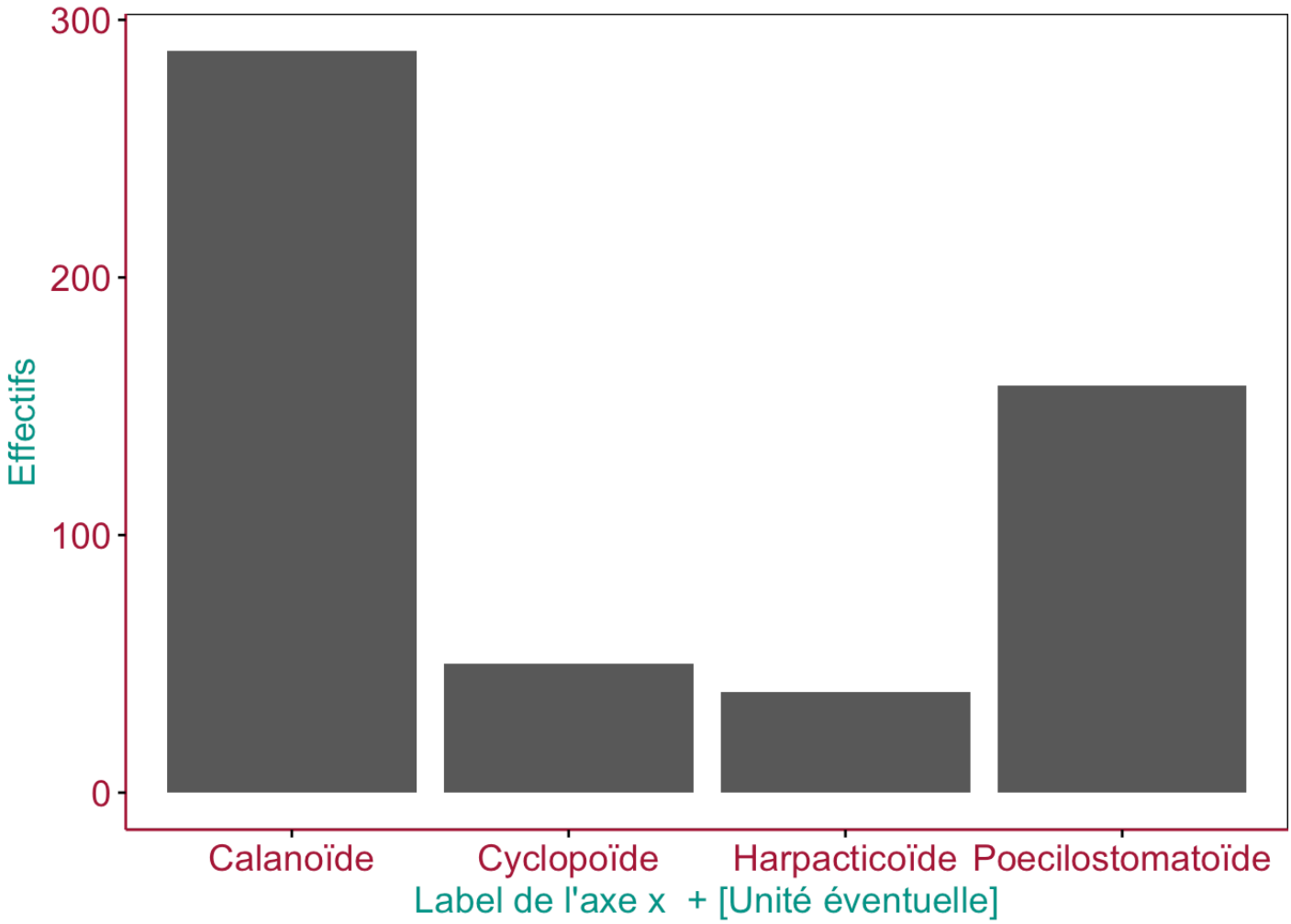


Figure 3.1: Exemple d'un graphique en barres montrant le dénombrement des niveaux d'une variable facteur, avec les éléments importants du graphique mis en évidence en couleurs.

Outre les barres elles-mêmes, prêtez toujours attention aux éléments suivants du graphique (ici mis en évidence en couleurs) :

- les axes avec les graduations (en rouge)
- les niveaux de la variable facteur (en rouge également)
- le label des axes (en bleu)

Les instructions dans R pour produire un graphique en barres à l'aide de la fonction `chart()` sont les suivantes. Nous partons d'un jeu de données `zooplankton` que nous importons et dont nous extrayons un sous-ensemble à l'aide de la fonction `filter_()` (vous étudierez en détail les fonctions de remaniement de tableaux dans les deux prochains modules) avant de réaliser notre graphique à l'aide de `chart()` :

```
# Importation du jeu de données
(zooplankton <- read("zooplankton", package = "data.io", lang = "FR"))
```

```
# # A data.trame: [1,262 × 20]
#      ecd  area perimeter feret major minor  mean  mode   min   max std_dev r
#    <dbl> <dbl>     <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>   <dbl> <
#  1 0.770 0.465      4.45 1.32  1.16  0.509 0.363 0.036 0.004 0.908   0.231 0
#  2 0.700 0.385      2.32 0.728 0.713 0.688 0.361 0.492 0.024 0.676   0.183 0
#  3 0.815 0.521      4.15 1.33  1.11  0.598 0.308 0.032 0.008 0.696   0.204 0
#  4 0.785 0.484      4.44 1.78  1.56  0.394 0.332 0.036 0.004 0.728   0.218 0
#  5 0.361 0.103      1.71 0.739 0.694 0.188 0.153 0.016 0.008 0.452   0.110 0
#  6 0.832 0.544      5.27 1.66  1.36  0.511 0.371 0.02  0.004 0.844   0.268 0
#  7 1.23  1.20      15.7  3.92  1.37  1.11  0.217 0.012 0.004 0.784   0.214 0
#  8 0.620 0.302      3.98 1.19  1.04  0.370 0.316 0.012 0.004 0.756   0.246 0
#  9 1.19  1.12      15.3  3.85  1.34  1.06  0.176 0.012 0.004 0.728   0.172 0
# 10 1.04  0.856      7.60 1.89  1.66  0.656 0.404 0.044 0.004 0.88    0.264 0
# # ℹ 1,252 more rows
# # ℹ 8 more variables: size <dbl>, aspect <dbl>, elongation <dbl>,
# #   compactness <dbl>, transparency <dbl>, circularity <dbl>, density <dbl>,
# #   class <fct>
```

```
# Réduction du jeu de données à quatre classes seulement
(copepoda <- filter_(zooplankton, ~class %in% c("Calanoïde", "Cyclopoïde",
  "Harpacticoïde",  "Poecilostomatoïde")))
```

```
# # A data.trame: [535 × 20]
#      ecd  area perimeter feret major minor  mean  mode   min   max std_dev r
#    <dbl> <dbl>     <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>   <dbl> <
#  1 0.770 0.465      4.45 1.32  1.16  0.509 0.363 0.036 0.004 0.908   0.231 0
#  2 0.815 0.521      4.15 1.33  1.11  0.598 0.308 0.032 0.008 0.696   0.204 0
#  3 0.785 0.484      4.44 1.78  1.56  0.394 0.332 0.036 0.004 0.728   0.218 0
#  4 0.361 0.103      1.71 0.739 0.694 0.188 0.153 0.016 0.008 0.452   0.110 0
#  5 0.832 0.544      5.27 1.66  1.36  0.511 0.371 0.02  0.004 0.844   0.268 0
#  6 1.23  1.20      15.7  3.92  1.37  1.11  0.217 0.012 0.004 0.784   0.214 0
#  7 0.620 0.302      3.98 1.19  1.04  0.370 0.316 0.012 0.004 0.756   0.246 0
#  8 1.19  1.12      15.3  3.85  1.34  1.06  0.176 0.012 0.004 0.728   0.172 0
#  9 1.04  0.856      7.60 1.89  1.66  0.656 0.404 0.044 0.004 0.88    0.264 0
# 10 0.725 0.412      7.14 1.90  0.802 0.655 0.209 0.008 0.004 0.732   0.202 0
# # ℹ 525 more rows
# # ℹ 8 more variables: size <dbl>, aspect <dbl>, elongation <dbl>,
# #   compactness <dbl>, transparency <dbl>, circularity <dbl>, density <dbl>,
# #   class <fct>
```

```
# Réalisation du graphique
chart(data = copepoda, ~ class) +
  geom_bar() +
  xlab("Classe") +
  ylab("Effectifs")
```

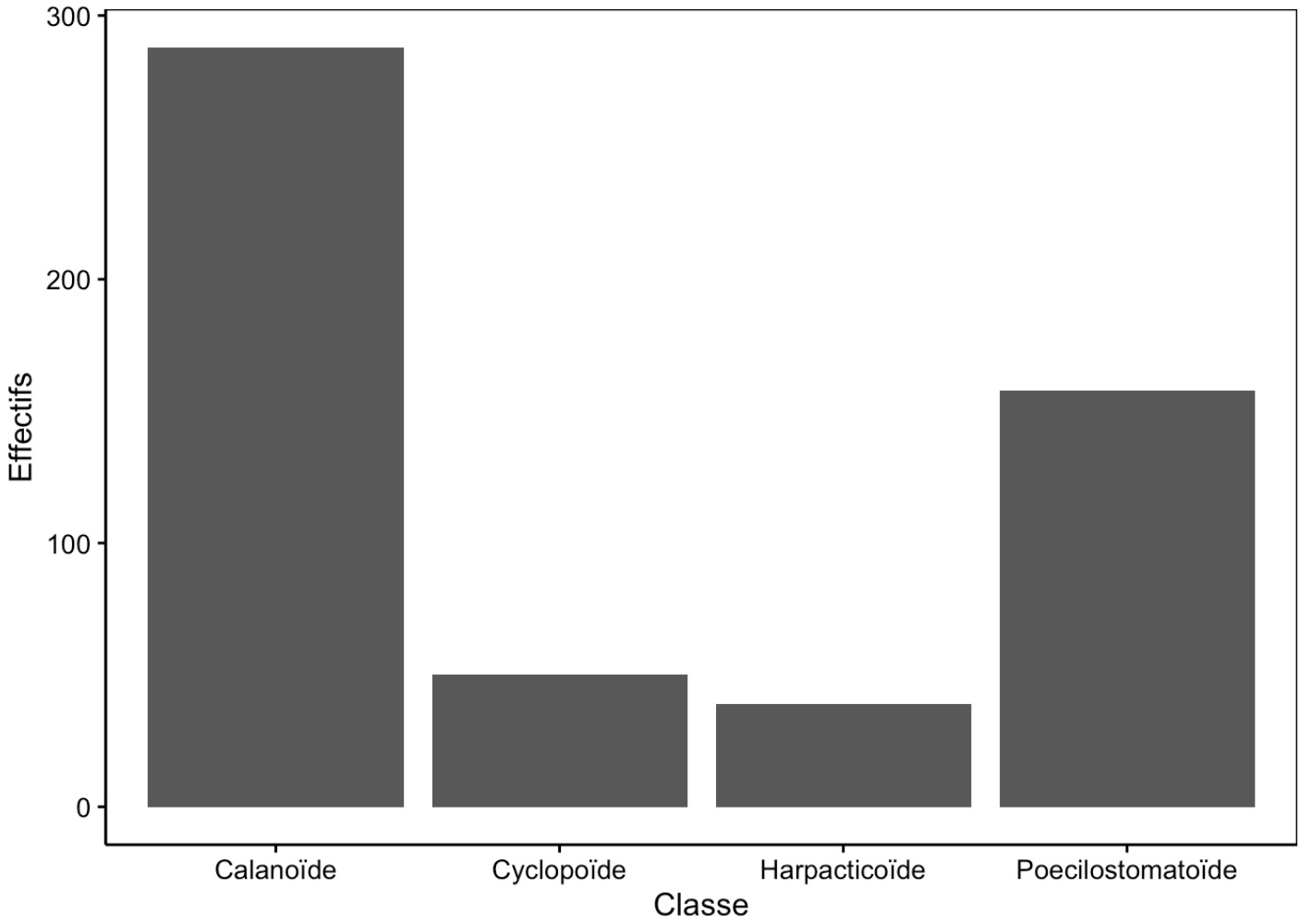


Figure 3.2: Abondances de quatre types de copépodes dans un échantillon de zooplancton.

La fonction `geom_bar()` se charge d'ajouter les barres verticales dans le graphique. La hauteur de ces barres correspond au nombre d'observations rencontrées dans le jeu de données pour chaque niveau (ou classe, ou groupe) de la variable facteur représentée.

3.1.2 Effectifs par deux facteurs

Reprenons maintenant le jeu de données `biometry` .

```
# Importation des données `biometry`
(biometry <- read("biometry", package = "BioDataScience", lang = "FR"))
```

```
# # A data.trame: [395 × 7]
#    gender day_birth  weight height wrist year_measure   age
#    <fct>  <date>      <dbl>  <dbl> <dbl>        <dbl> <dbl>
#  1 H      1995-03-11     69    182  15           2013    18
#  2 H      1998-04-03     74    190  16           2013    15
#  3 H      1967-04-04     83    185  17.5         2013    46
#  4 H      1994-02-10     60    175  15           2013    19
#  5 F      1990-12-02     48    167  14           2013    23
#  6 F      1994-07-15     52    179  14           2013    19
#  7 F      1971-03-03     72    167  15.5         2013    42
#  8 F      1997-06-24     74    180  16           2013    16
#  9 H      1972-10-26    110    189  19           2013    41
# 10 H      1945-03-15     82    160  18           2013    68
# # ℹ 385 more rows
```

Nous voulons représenter des barres pour les effectifs d'hommes et de femmes dans ce jeu de données (variable `gender`), mais en les séparant par année (variable `year_measure`). C'est faisable, mais notez que, si `gender` est déjà une variable facteur `<fct>`, `year_measure` est encodé comme variable quantitative numérique (c'est un "double" indiqué `<dbl>`, c'est-à-dire un nombre décimal par opposition à entier `<int>`). Or nous devons **absolument utiliser des variables facteur ici**. Nous allons donc effectuer la conversion avec la fonction `as.factor()` avant de réaliser notre graphique en barres. Nous en profitons pour indiquer un `label()` en français pour cette variable.

```
# Conversion de la variable year_measure de numérique à facteur
biometry$year_measure <- as.factor(biometry$year_measure)
label(biometry$year_measure) <- "Année de la mesure"
```

Notez bien comment on se réfère à la variable `year_measure` à l'intérieur du jeu de données `biometry` avec `biometry$year_measure`. Cette notation peut aussi bien être utilisée pour récupérer la colonne `year_measure` dans un argument d'une fonction (à droite), que comme résultat de l'assignation (à gauche de l'opérateur d'assignation `<-`).

Ainsi l'instruction qui transforme en facteur **remplace** la version dans le jeu de données `biometry`. Maintenant, considérons que nous nous intéressons aux mesures antérieures à 2017 (cela nous permettra d'illustrer des points importants relatifs à l'utilisation de variables facteurs).

> Les variables facteurs sont encodées dans R comme des entiers 1, 2, 3… pour les différents niveaux avec en plus, un attribut `levels` qui y associe une description textuelle à chacun des niveaux. Cela peut être perturbant quand la description textuelle est constituée d'un nombre comme ici pour `year_measure`. Mais les calculs sont prohibés sur les variables facteurs.

```
tail(biometry)$year_measure
```

```
# Année de la mesure
# [1] 2017 2017 2017 2017 2017 2017
# Levels: 2013 2014 2016 2017
```

Nous utilisons `head()` et `tail()` pour extraire les quelques lignes de début ou de fin d'un tableau. C'est utile pour en réduire la taille à l'impression. Faites très attention : lorsque vous imprimez le contenu d'une variable facteur, R *substitue* automatiquement les niveaux 1, 2, 3 … par le contenu textuel comme indiqué dans la dernière ligne `Levels: ...`, et **l'imprime sans mettre le texte entre guillemets**. Cela peut renforcer la fausse impression que c'est bien des valeurs numériques.

Vous commencez à comprendre que si vous effectuez maintenant la comparaison `year_measure < 2017` lorsque cette variable est encodée comme facteur, cela ne fonctionnera pas comme vous le souhaitez ! Si on est chanceux, un message d'erreur ou d'avis ("warning") est imprimé.

```
biometry2 <- filter_(biometry, ~year_measure < 2017)
```

```
# Warning in Ops.factor(year_measure, 2017): '<' not meaningful for factors
```

Notez au passage que `filter_()` n'utilise pas la notation `biometry$year_measure`, mais prend un premier argument qui est le jeu de données `biometry`, et la suite se réfère aux variables de ce jeu de données telles que `year_measure` en priorité aux autres variables disponibles dans l'environnement utilisateur de R dans une **formule** (un objet R qui contient une instruction et qui commence par `~` ici). Aussi `filter_()` renvoie tout le tableau remanié. Donc, nous devons l'affecter simplement à une variable qui contient ce tableau (`biometry2` ici). Vous pouvez aussi utiliser la fonction `filter()` qui s'emploie de manière similaire mais sans formule (retirez le `~` devant `year_measure < 2017`).

Mais revenons à notre variable facteur. Dans d'autres cas, le résultat peut être dramatique, car le calcul est appliqué à l'encodage des niveaux de la variable facteur. Or, avec quatre niveaux, les encodages sont 1, 2, 3 et 4, … et ils sont bien évidemment tous inférieurs à 2017 (pour rappel, "2013", "2014", "2016" et "2017" sont les libellés textuels des niveaux de la variable facteur) !

> De manière générale, n'effectuez **jamais** de calcul sur des variables facteurs. Transformez-les toujours avant. Si les libellés contiennent des valeurs numériques sur lesquelles vous voulez faire des calculs, utilisez `as.numeric(as.character(VARFACT))`, et une fois le calcul réalisé, retransformez en facteur avec `as.factor()`.

Le calcul explicite et sûr est donc le suivant :

```
# Transforme de manière sûre factor -> numeric (double)
biometry$year_measure <- as.numeric(as.character(biometry$year_measure))
# Filtre les données sur une copie du tableau
biometry2 <- filter_(biometry, ~year_measure < 2017)
# Retransforme en variable factor
biometry2$year_measure <- as.factor(biometry2$year_measure)
# Vérification
tail(biometry2)$year_measure
```

```
# [1] 2016 2016 2016 2016 2016 2016
# Levels: 2013 2014 2016
```

Naturellement ici, la bonne stratégie est d'effectuer le calcul sur le tableau de départ **avant** de transformer en facteur, mais nous sommes partis de la variable facteur à titre d'illustration d'un cas qui peut se rencontrer en pratique. À présent que nous avons notre jeu de données sans les individus de 2017, et les variables correctement encodées, nous pouvons aborder différentes représentations pour observer des dénombrements tenant compte de plusieurs variables facteurs. Par défaut, l'argument `position =` a pour valeur `"stack"` (donc, lorsque cet argument n'est pas précisé dans `geom_bar()`, les barres sont empilées par rapport à la seconde variable facteur).

```
a <- chart(data = biometry2, ~ gender) +
  geom_bar() +
  ylab("Effectifs")

b <- chart(data = biometry2, ~ gender %fill=% year_measure) +
  geom_bar() +
  ylab("Effectifs") +
  scale_fill_viridis_d()

combine_charts(list(a, b), common.legend = TRUE)
```

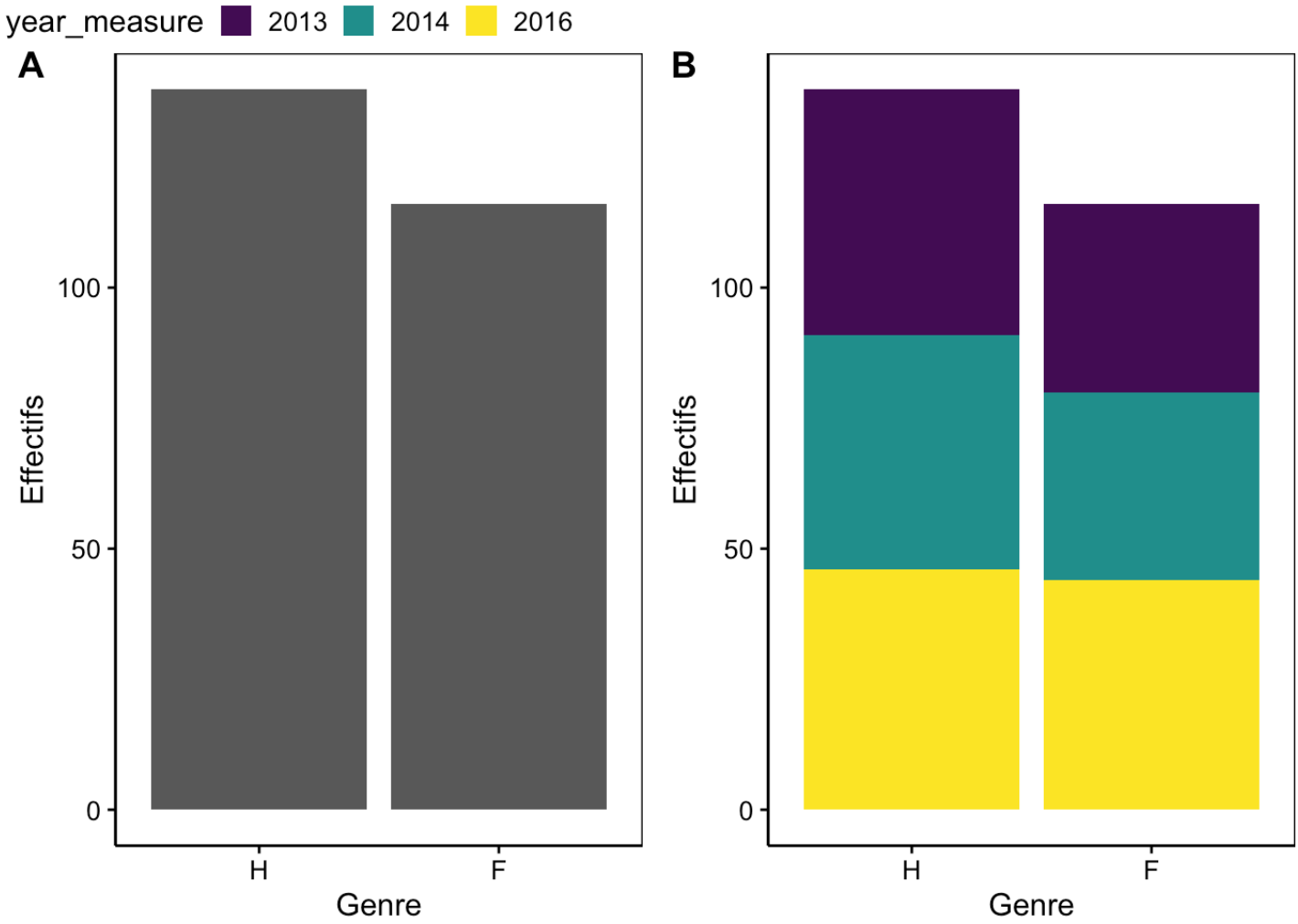


Figure 3.3: Dénombrement des hommes (H) et des femmes (F) dans l'étude sur l'obésité en Hainaut, (A) graphique utilisant un seul facteur. (B) graphique prenant en compte les années de mesure.

Il existe d'autres options en utilisant les valeurs `"dodge"` ou `"fill"` pour l'argument `position = .`

```r
a <- chart(data = biometry2, ~ gender %fill% year_measure) +
  geom_bar(position = "stack") +
  ylab("Effectifs") +
  scale_fill_viridis_d()

b <- chart(data = biometry2, ~ gender %fill% year_measure) +
  geom_bar(position = "dodge") +
  ylab("Effectifs") +
  scale_fill_viridis_d()

c <- chart(data = biometry2, ~ gender %fill% year_measure) +
  geom_bar(position = "fill") +
  ylab("Fractions") +
  scale_fill_viridis_d()

combine_charts(list(a, b, c), common.legend = TRUE)
```

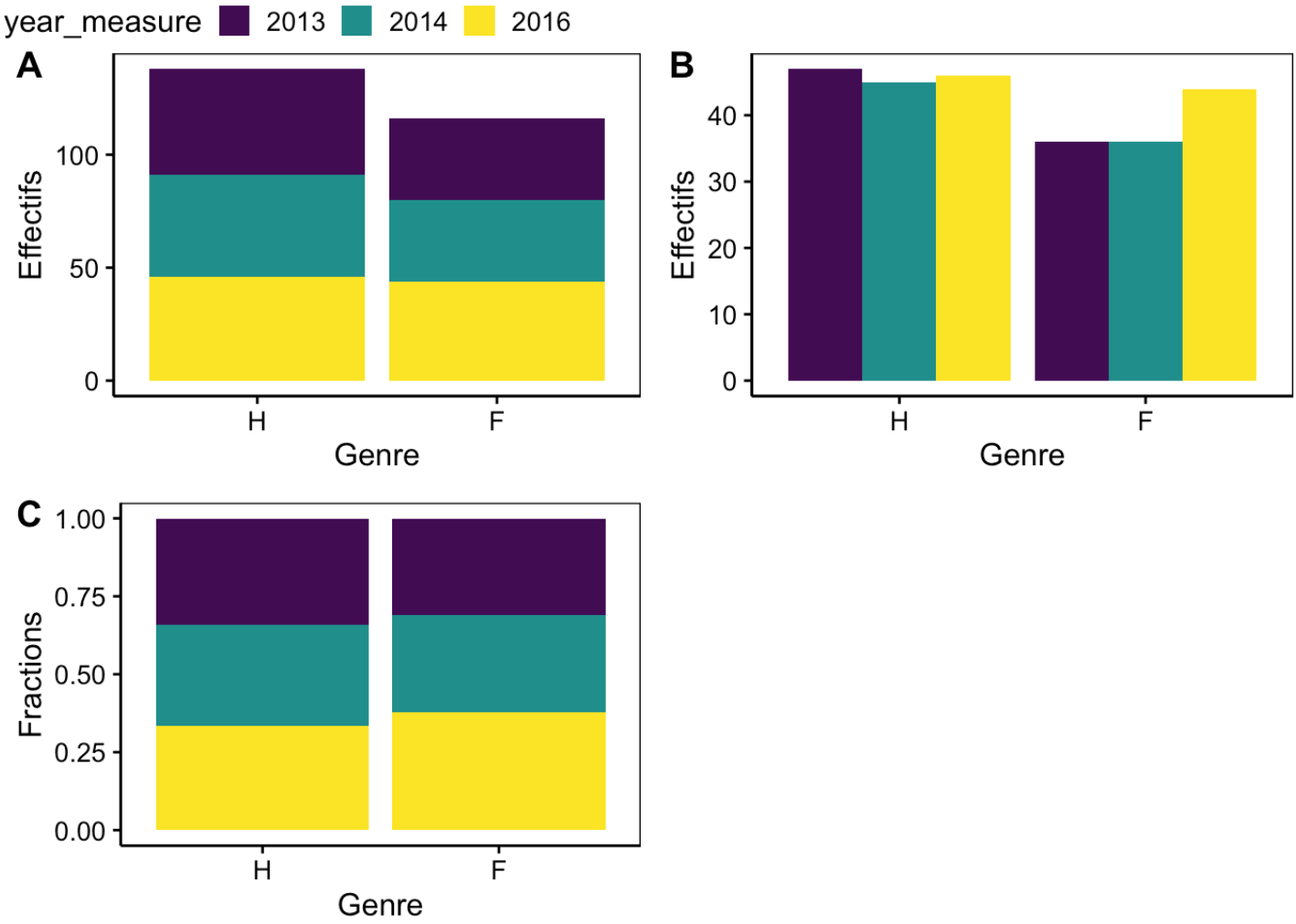
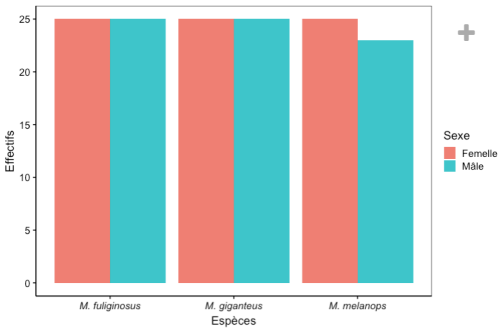


Figure 3.4: Dénombrement des hommes (H) et des femmes (F) dans l'étude sur l'obésité en Hainaut en tenant compte des années de mesure (différentes présentations).

Soyez vigilant à la différence entre l'argument `position = "stack"` (figure A) et `position = "fill"` (Figure C) qui malgré un rendu semblable ont l'axe des ordonnées qui diffèrent (dans le cas de `"fill"`, il s'agit de la **fraction** par rapport au total qui est représentée, et non pas des **effectifs** absolus dénombrés).



Sur base du graphique ci-dessus, quelle est la valeur de l'argument `position=` dans la fonction geom_bar() ?

- ○ stack
- ○ fill
- ○ dodge

Afficher la réponse

Pièges et astuces

Réordonner la variable facteur par fréquence

Vous pouvez avoir le souhait d'ordonner votre variable facteur afin d'améliorer le rendu visuel de votre graphique. Pour cela, vous pouvez employer la fonction `fct_infreq()` .

```
chart(data = copepoda, ~ fct_infreq(class)) +
  geom_bar() +
  labs(x = "Classe", y = "Effectifs")
```

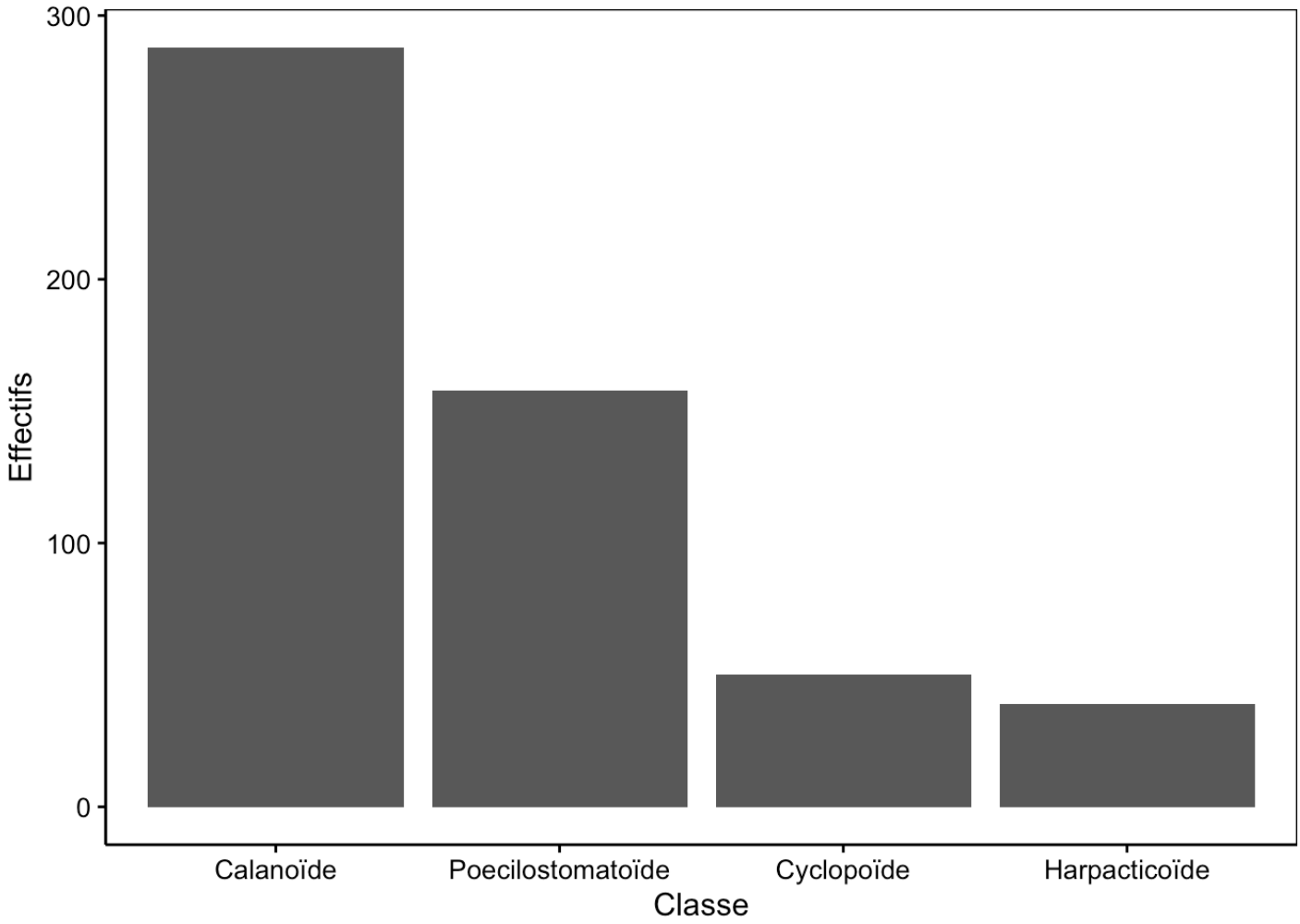


Figure 3.5: Dénombrement des classes de copépodes du jeu de données zooplankton.

Rotation des axes du graphique en barres

Lorsque les niveaux dans la variable étudiée sont trop nombreux, les légendes en abscisse risquent de se chevaucher, comme dans la Fig. 3.6

```
chart(data = zooplankton, ~ class) +
  geom_bar() +
  ylab("Effectifs")
```

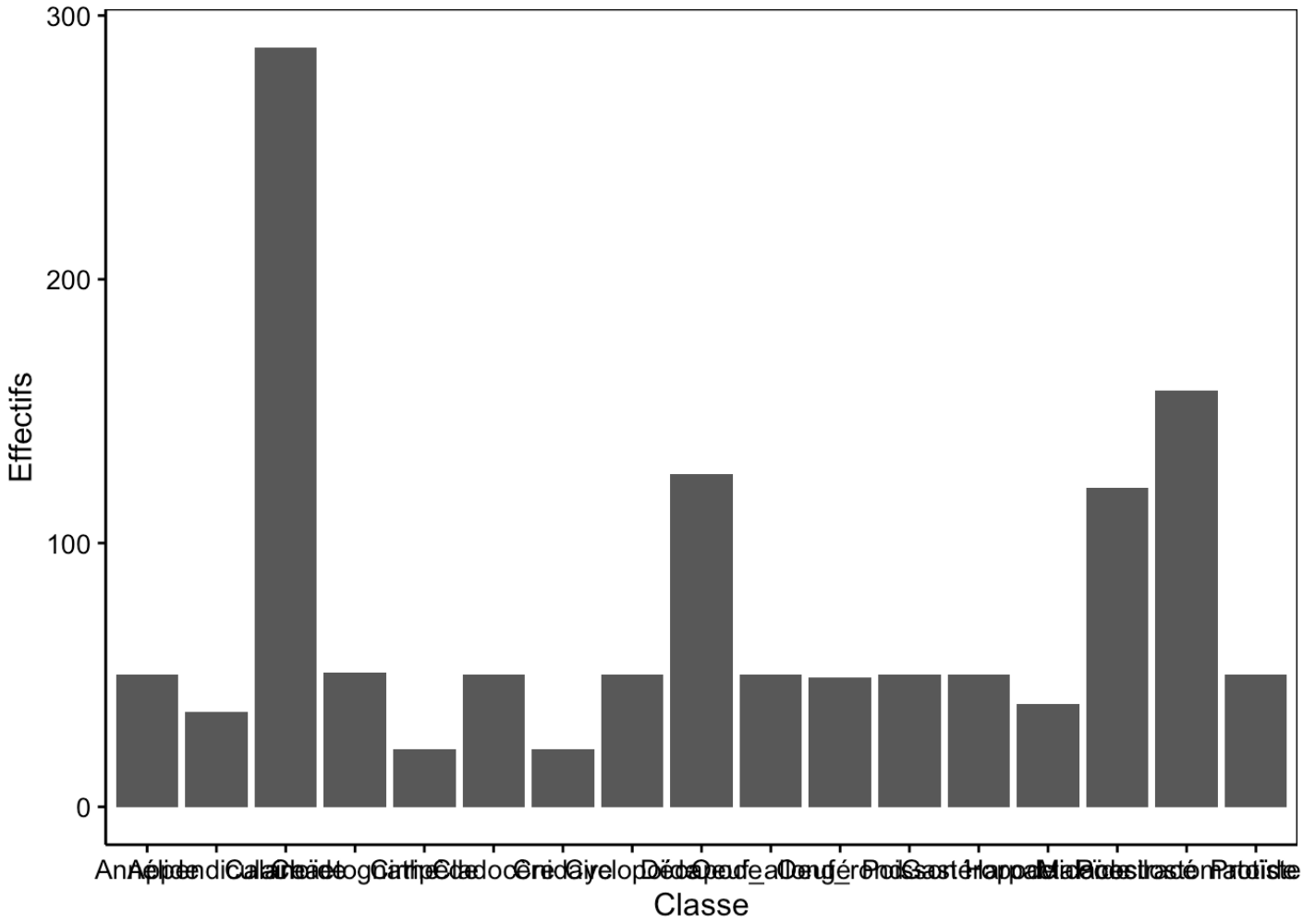


Figure 3.6: Dénombrement des classes du jeu de données zooplankton.

Dans ce cas, il est possible de réaliser un graphique en **barres horizontales** qui laisse plus de place pour le libellé sur l'axe. Il existe deux manières de le faire : préférentiellement en utilisant l'argument `orientation = "y"` de `geom_bar()` (mais alors il faut utiliser `aes(y = ...)` à la place de la formule). Une seconde option consiste à réaliser le graphique en barres verticales (tout le code reste identique), mais de rajouter `coord_flip()` tout à la fin. Utilisons successivement ces deux approches.

```
chart(data = zooplankton, aes(y = class)) +
  geom_bar(orientation = "y") +
  xlab("Effectifs")
```

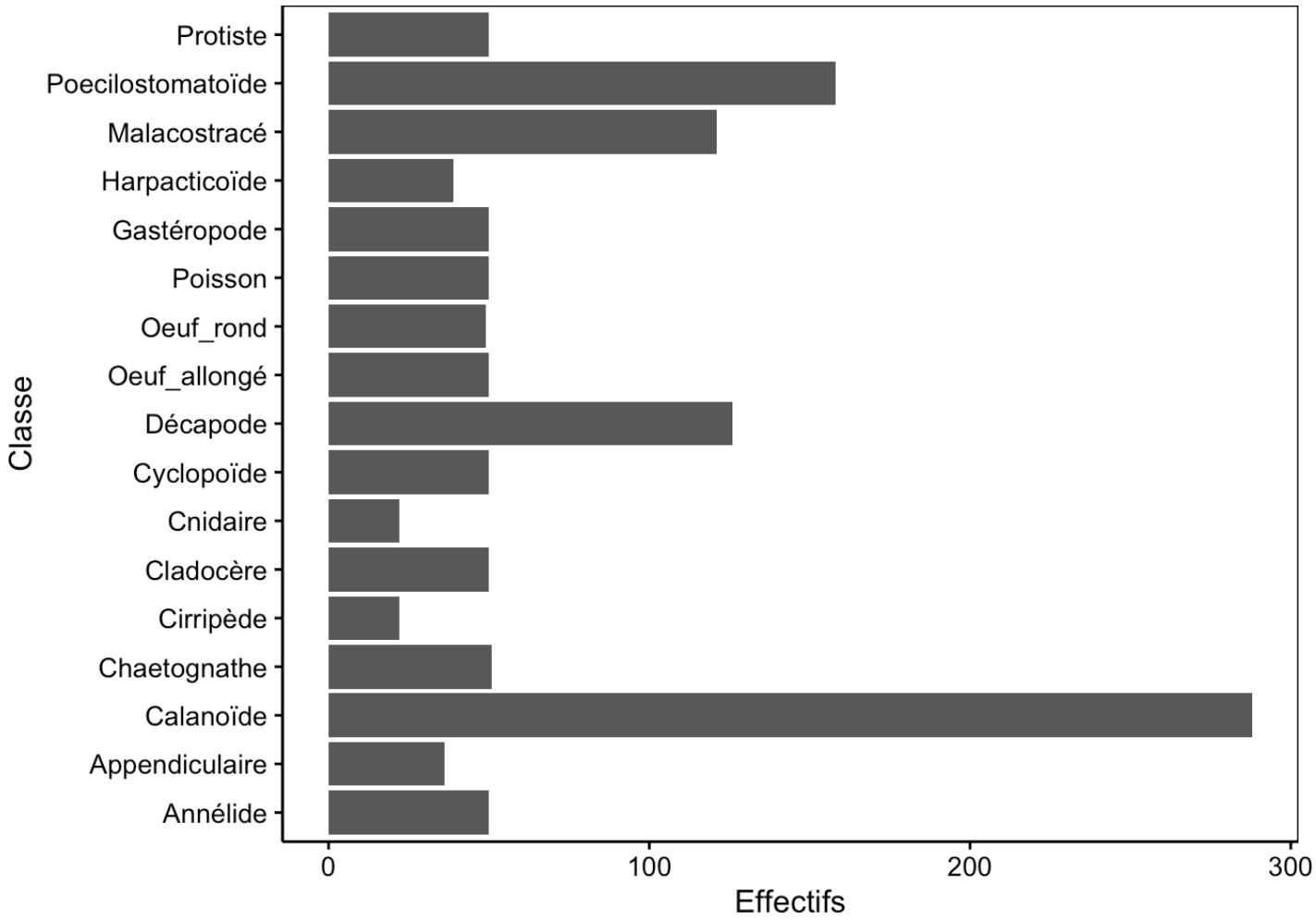


Figure 3.7: Dénombrement des classes du jeu de données zooplankton (version avec barres horizontales en utilisant l'argument `orientation=`).

Pourquoi ne peut-on pas utiliser de formule dans ce cas ? En fait, il faudrait écrire `class ~ ` , … seulement voilà, une formule ne *peut pas* avoir un membre de droite vide. Donc, on est dans une impasse et on doit utiliser la forme explicite `aes()` pour "aesthetics" en anglais qui indique quelle variable est utilisée pour quoi dans le graphique en indiquant `y = .`

Avec la fonction `coord_flip()` ajoutée à votre graphique, vous pouvez effectuer une rotation des axes (l'axe X devient Y et inversement) pour obtenir un **graphique en barres horizontales**. De plus, l'œil humain perçoit plus distinctement les différences de longueurs horizontales que verticales. Donc, de ce point de vue, le graphe en barres horizontales est considéré comme meilleur que le graphe en barres verticales.

```
chart(data = zooplankton, ~ class) +
  geom_bar() +
  ylab("Effectifs") +
  coord_flip()
```

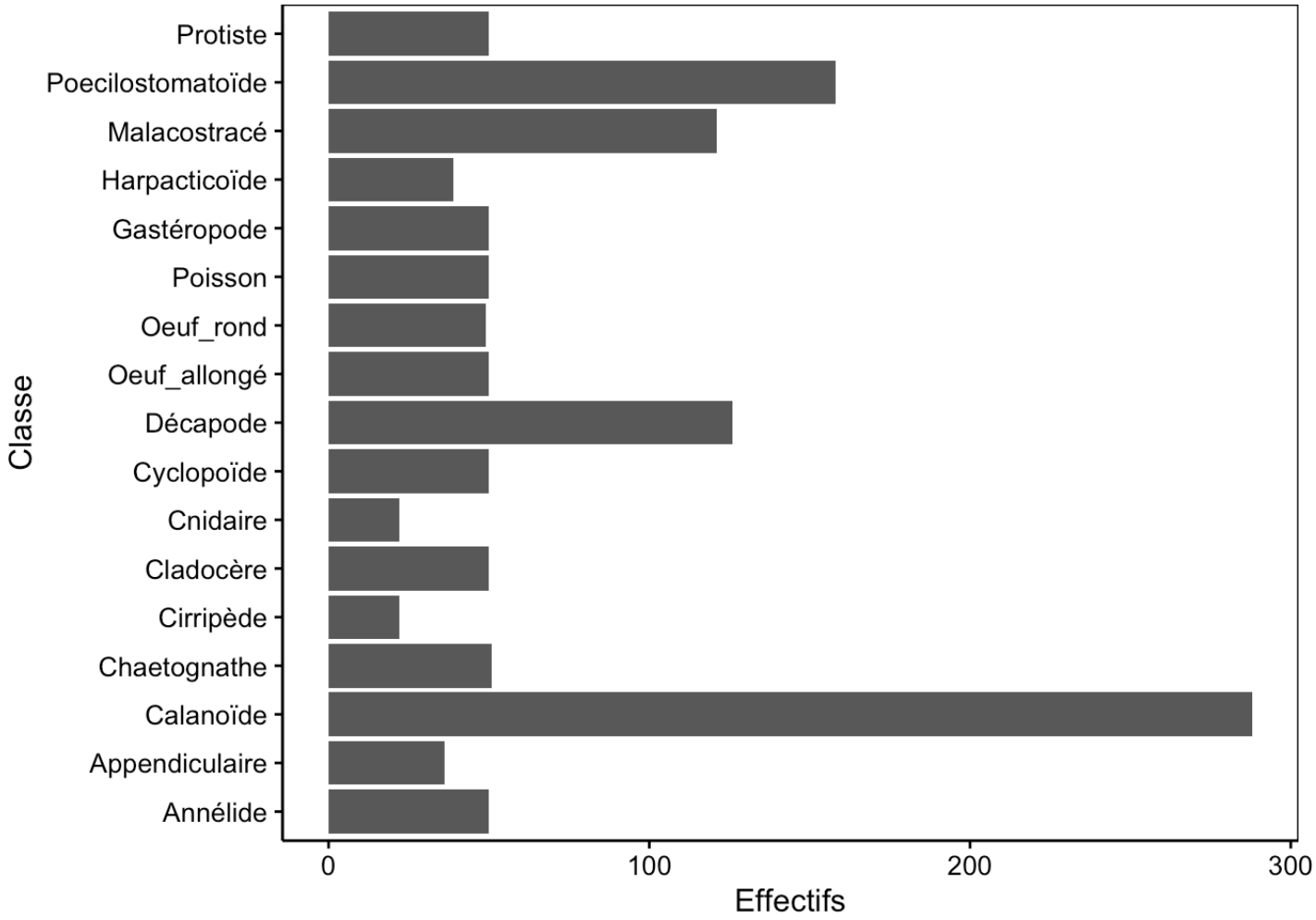


Figure 3.8: Dénombrement des classes du jeu de données zooplankton (version avec barres horizontales en utilisant `coord_flip()`).

Avec `coord_flip()`, gardez toujours à l'esprit que vos axes *finaux* X et Y sont inversés avant l'utilisation de cette instruction. Ainsi, si vous voulez changer le libellé de l'axes X sur le graphique final, mais *avant* d'avoir indiqué `coord_flip()`, c'est en fait le libellé de l'axe Y que vous devez indiquer avec, par exemple `ylab()` comme c'est le cas dans la figure précédente pour le libellé "Effectifs".

Pour en savoir plus

- Graphes en barres à l'aide de ggplot2. Un tutoriel en anglais utilisant la fonction `ggplot()` .
- Page d'aide de la fonction `geom_bar()` en anglais.
- Une page qui présente l'utilisation de geom_col(), une autre fonction qui réalise des graphiques en bâtonnets, en français.

3.1.3 Valeurs moyennes

Le graphique en barres peut être aussi employé pour résumer des données numériques via la moyenne. Il ne s'agit plus de dénombrer les effectifs d'une variable facteur, mais de résumer des données numériques en fonction d'une variable facteur. On peut exprimer cette relation dans R sous la forme de :

$$y \sim x$$

que l'on peut lire :

$$y\ en\ fonction\ de\ x$$

Avec *y* une variable numérique et *x* une variable facteur. Considérez l'échantillon suivant :

```
1, 71, 55, 68, 78, 60, 83, 120, 82 ,53, 26
```

Calculez la moyenne :

$$\bar{y} = \sum_{i=1}^{n} \frac{y_i}{n}$$

```
# Création du vecteur
x <- c(1, 71, 55, 68, 78, 60, 83, 120, 82, 53, 26)
# Calcul  de la moyenne
mean(x)
```

```
# [1] 63.36364
```

Les instructions pour produire ce graphe en barres à l'aide de `chart()` sont :

```
chart(data = copepoda, size ~ class) +
  stat_summary(geom = "col", fun = "mean") +
  xlab("Classe")
```

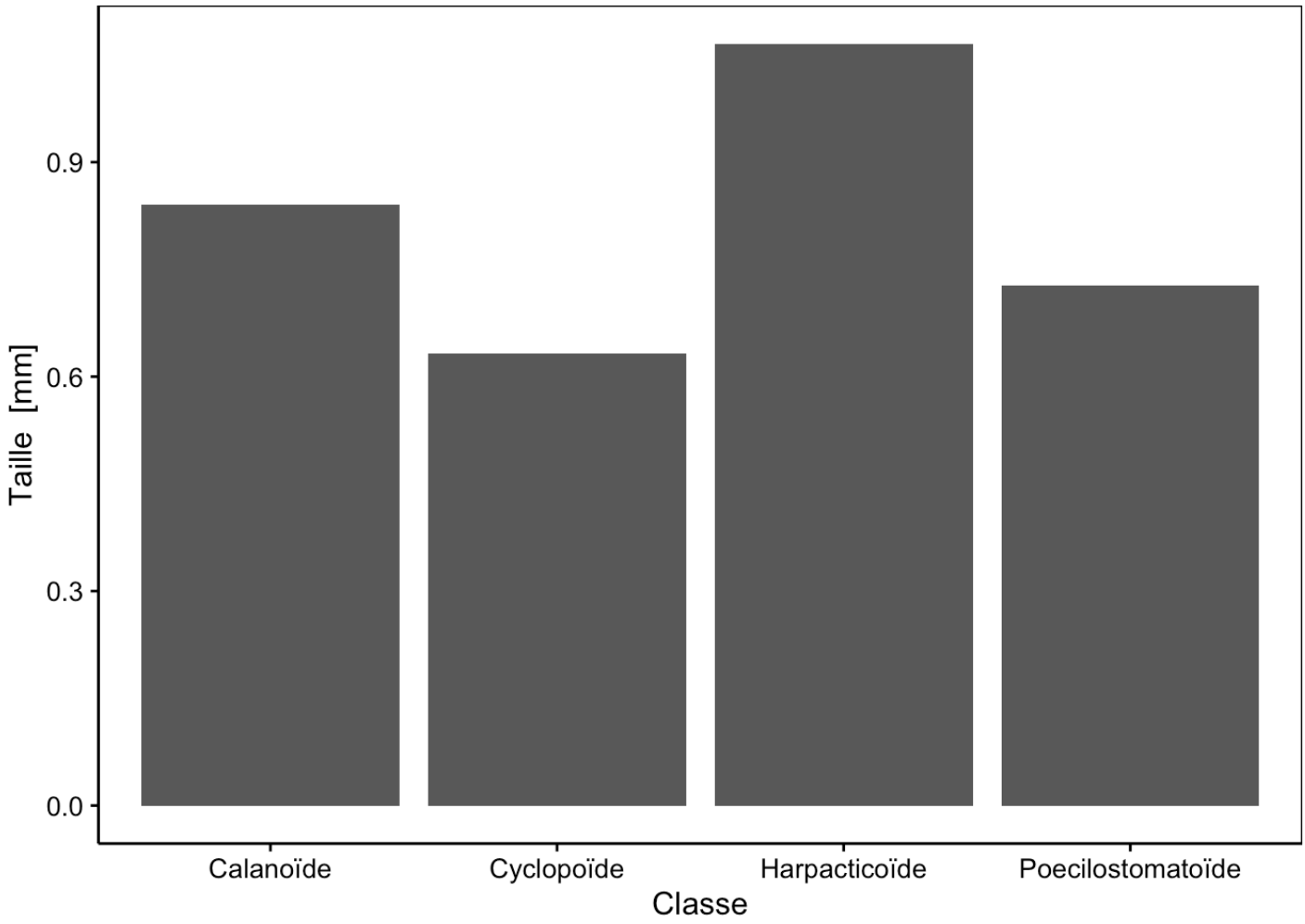
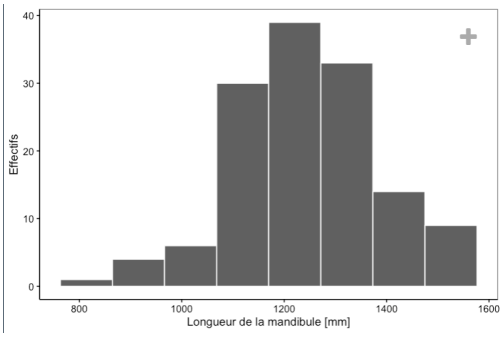


Figure 3.9: Exemple de graphique en barres représentant les moyennes de tailles par groupe zooplanctonique.

Ici, nous faisons appel à une autre famille de fonctions : celles qui effectuent des calculs sur les données avant de les représenter graphiquement. Leurs noms commencent toujours par `stat_` .

Le graphe en barres pour représenter les moyennes est très répandu dans le domaine scientifique malgré le grand nombre d'arguments en sa défaveur et que vous pouvez lire dans la section **pour en savoir plus** ci-dessous. L'un des arguments les plus importants est la faible information qu'il véhicule puisque l'ensemble des données n'est plus représenté que par une valeur (la moyenne) pour chaque niveau de la variable facteur. Pour un petit nombre d'observations, il vaut mieux toutes les représenter à l'aide d'un nuage de points. Si le nombre d'observations devient très grand (dizaines ou plus), le graphique en boites à moustaches est plus indiqué (voir plus loin dans ce module). Le graphique en violon (cf. module précédent) est également utilisable s'il y a encore plus de données.



Ce graphique concerne la longueur de la mandibule du kangourou. Quel type de graphique est-ce ? Faites particulièrement attention aux étiquettes, aux unités et aux axes.

- ○ Il s'agit d'un histogramme, probablement réalisé avec la fonction `geom_histogram()`, qui permet de dénombrer le nombre d'individus par classe d'une variable numérique, ici la longueur de la mandibule.
- ○ Il s'agit d'un graphique en barres, probablement réalisé avec la fonction `geom_bar()`, qui permet de dénombrer le nombre d'individus par niveau d'une variable facteur, ici la longueur de la mandibule.
- ○ Il s'agit d'un graphique en barres, probablement réalisé avec la fonction `geom_col()`, qui permet d'indiquer la valeur moyenne par niveau d'une variable facteur, ici la longueur de la mandibule.

Pour en savoir plus

- Beware of dynamite. Démonstration de l'impact d'un graphe en barres pour représenter la moyenne (et l'écart type) = graphique en "dynamite".
- Dynamite plots: unmitigated evil? Une autre comparaison du graphe en dynamite avec des représentations alternatives qui montre que le premier peut avoir quand même quelques avantages dans des situations particulières.

3.2 Graphique en camembert

Le graphique en camembert (ou en parts de tarte, ou encore appelé diagramme circulaire, *pie chart* en anglais) vous permettra de visualiser un dénombrement d'observations par facteur, tout comme le graphique en barres.

```
chart(data = copepoda, ~ factor(0) %fill=% class) +
  geom_bar(width = 1) +
  coord_polar("y", start = 0) +
  theme_void() +
  scale_fill_viridis_d()
```

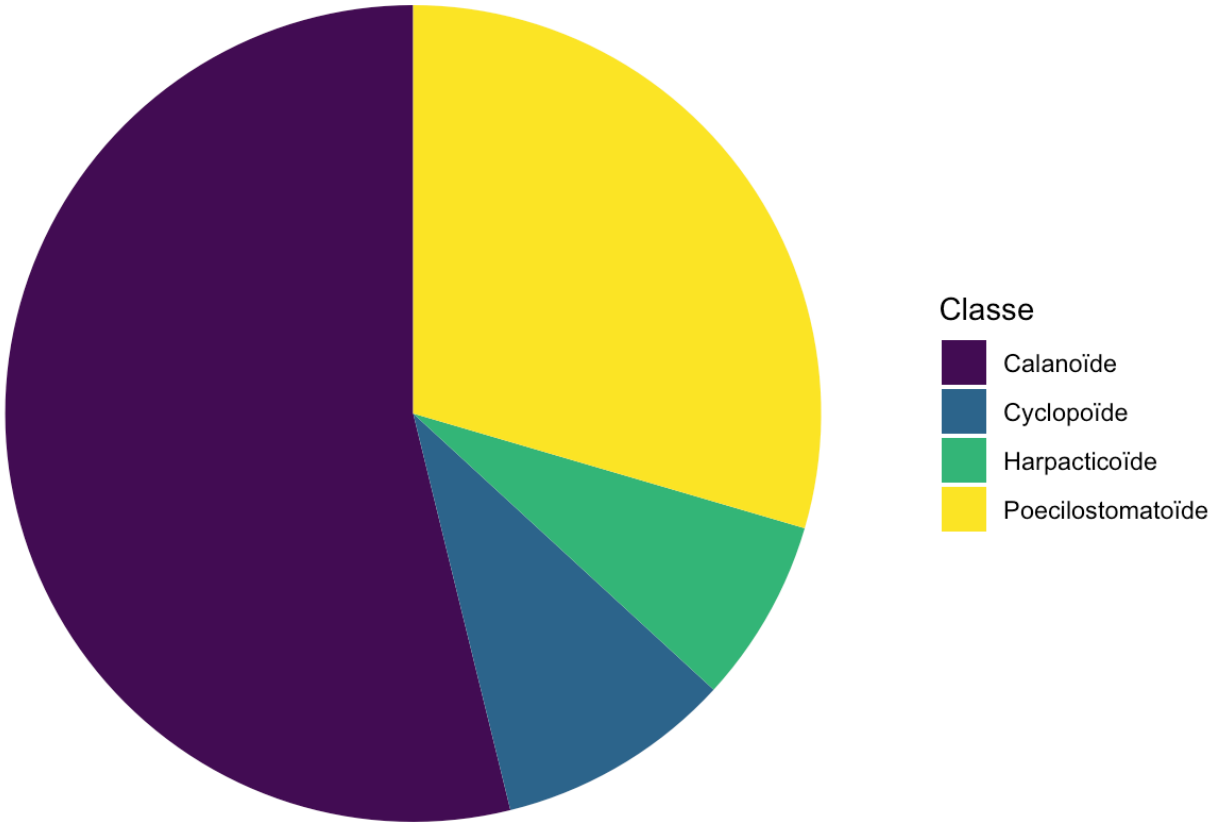


Figure 3.10: Exemple de graphique en camembert montrant les effectifs des niveaux d'une variable facteur.

Ce graphique est plus difficile à réaliser à l'aide de `chart()` ou `ggplot()`. En fait, il faut ruser ici, et l'auteur du package {ggplot2} n'avait en fait pas l'intention d'ajouter ce type de graphique dans la panoplie proposée. En effet, il faut savoir que l'œil humain est nettement moins bon pour repérer des angles que pour comparer des longueurs. **Donc, le diagramme en barres est souvent meilleur pour comparer des effectifs par classes.** Mais d'une part, le graphique en camembert est (malheureusement) un graphique très répandu et il faut savoir l'interpréter, et d'autre part, il peut s'avérer quand même utile dans certaines situations. Notez l'utilisation des fonctions `coord_polar()` qui crée des coordonnées polaires et la fonction `theme_void()` qui crée un graphique sans axes. En fait, un graphique en camembert peut aussi se concevoir comme un graphique en barres représenté en coordonnées polaires.

Pièges et astuces

Partons d'un exemple fictif pour vous convaincre qu'un graphique en barres est souvent plus lisible qu'un graphique en camembert. Combien d'observations comptez-vous pour la lettre **H** ?

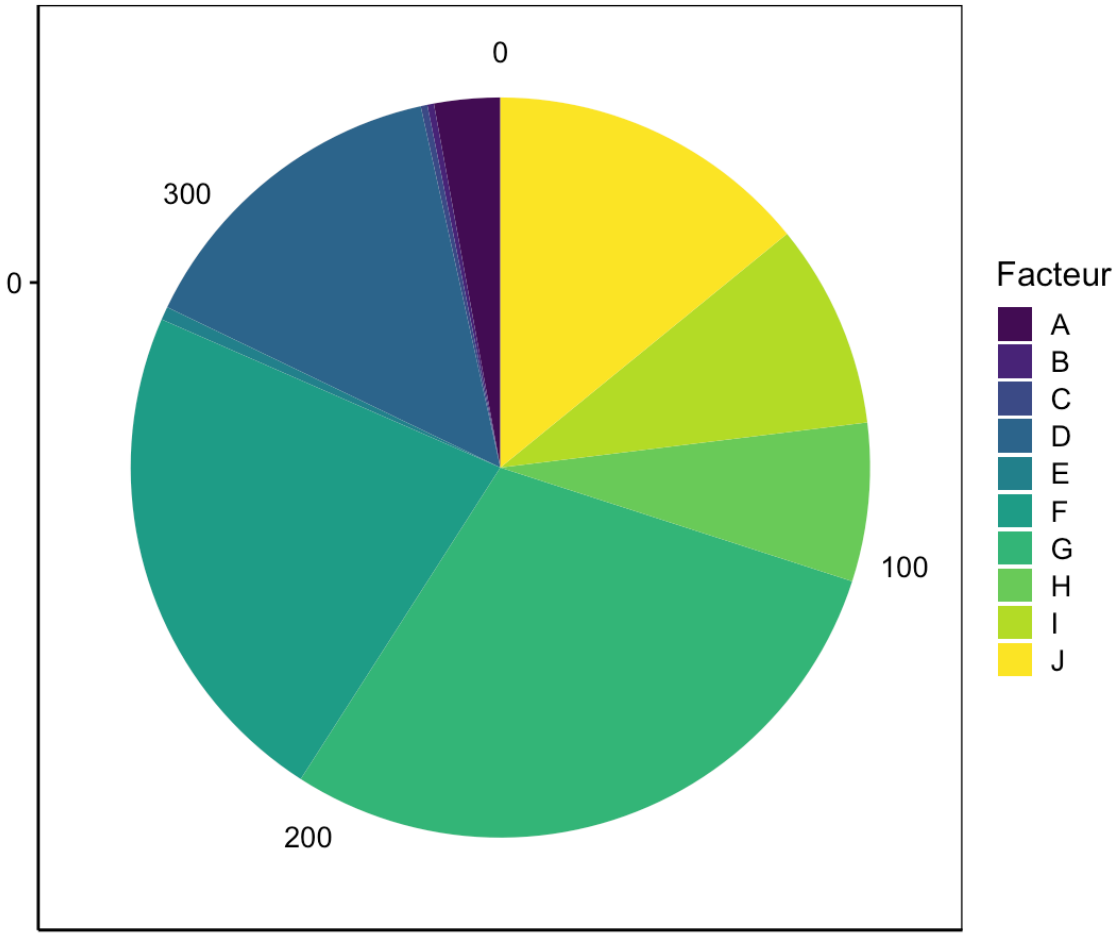


Figure 3.11: Arrivez-vous à lire facilement des valeurs sur un graphique en camenbert (une échelle y est ajoutée de manière exceptionnelle pour vous y aider).

Maintenant, effectuez le même exercice sur base d'un graphique en barres, combien d'observations pour la lettre **H** ?

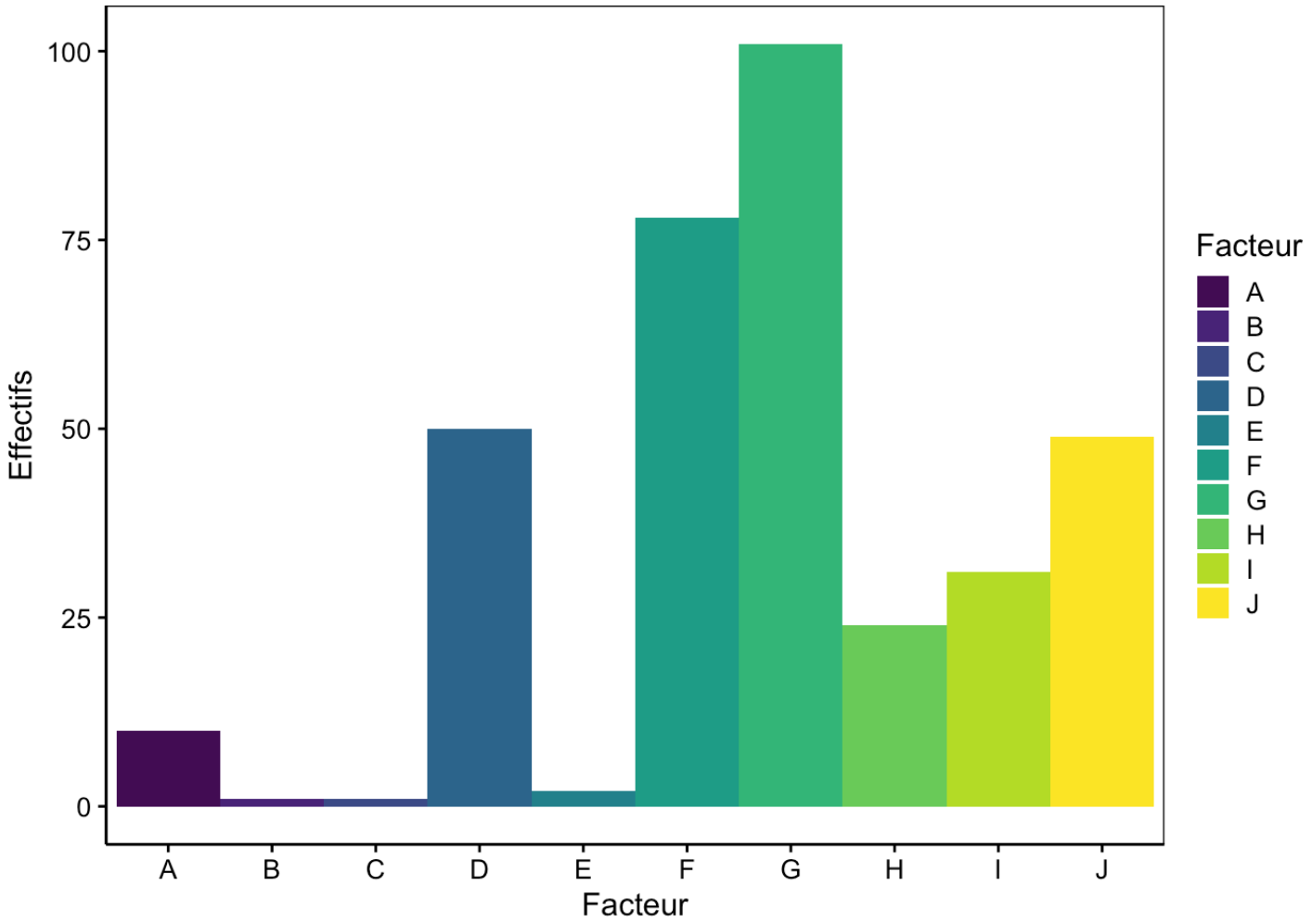


Figure 3.12: Dénombrement des niveaux d'une variable facteur sur un graphique en barres.

Dans ce dernier cas, c'est bien plus facile : il y a 24 observations relatives à la lettre **H** (vous ne voyez peut-être pas que l'effectif de *H* est exactement 24, mais vous pouvez voir sans problème qu'il est d'*environ* 25, alors que sur le graphique en camembert, nous le voyons beaucoup moins bien).

À vous de jouer !

Effectuez maintenant les exercices du tutoriel **A03La_barplot (Graphiques en barres et camembert)**.

```
BioDataScience1::run("A03La_barplot")
```

Pour en savoir plus

- Graphique en camembert à l'aide de la fonction `ggplot()` . Explications en français des différentes étapes pour passer d'un graphique en barres à un graphique en camembert avec **ggplot2**.
- Autre explication en français, également accompagnée d'informations sur les bonnes pratiques en matière de graphique en camembert.
- Save the pies for dessert est une démonstration détaillée des méfaits du graphique en camembert (le graphique en camembert, un graphique puant ? Pourrait-on peut-être titrer en français).
- Les côtés positifs du graphe en camembert sont mis en évidence dans ce document (en anglais).

3.3 Boite à moustaches

Vous souhaitez représenter graphiquement cette fois *un résumé* d'une variable numérique mesurée sur un nombre (relativement) important d'individus, soit depuis une dizaine jusqu'à plusieurs millions. Vous souhaitez également conserver de l'information sur la distribution des données, et voulez éventuellement comparer plusieurs distributions entre elles : soit différentes variables, soit différents niveaux d'une variable facteur. Nous avons déjà vu au module 2 les diagrammes en violon et en lignes de crêtes pour cet usage. Nous allons étudier ici les **boites à moustaches** (encore appelées boites de dispersion) comme option alternative intéressante. La boite à moustaches représentera graphiquement cinq descripteurs communément appelés les **cinq nombres**.

Considérez l'échantillon suivant :

```
1, 71, 55, 68, 78, 60, 83, 120, 82 ,53, 26
```

Ordonnons-le de la plus petite à la plus grande valeur :

```
# Créer du vecteur
x <- c(1, 71, 55, 68, 78, 60, 83, 120, 82, 53, 26)
# Ordonner le vecteur par ordre croissant
sort(x)
```

```
#  [1]   1  26  53  55  60  68  71  78  82  83 120
```

Le premier descripteur des cinq nombres est la **médiane** qui est la valeur se situant à la moitié des observations, donc, avec autant d'observations plus petites et d'observations plus grandes qu'elle. La médiane sépare l'échantillon en deux.

```
median(x)
```

```
# [1] 68
```

En effet, nous voyons sur le vecteur ordonné que cinq valeurs sont plus petites que 68 et cinq valeurs sont plus grandes. Les quartiles séparent l'échantillon en quatre. Le **premier quartile** (Q1) sera la valeur pour laquelle 25% des observations seront plus petites. Elle se situe donc entre la valeur minimale et la médiane. Cette médiane est égale au second quartile (50% des observations plus petites). Le **troisième quartile** (Q3) est la valeur pour laquelle 75% des observations de l'échantillon sont plus petites[13]. Enfin, la valeur **minimale** et la valeur **maximale** observées dans l'échantillon complètent ces cinq nombres qui décrivent de manière synthétique la *position* et l'*étendue* des observations.

H5P

Complétez le blanc.

Quelle est la valeur médiane de la série suivante de chiffres : 9, 4, 3, 5 ? La valeur médiane est de [ ].

Vérifier

> Les **cinq nombres** sont : la **valeur minimale**, le **premier quartile**, la **médiane** (ou deuxième quartile), le **troisième quartile** et la **valeur maximale**.

Voici comment on les calcule facilement dans R :

```
fivenum(x)
```

```
# [1]   1  54  68  80 120
```

La boite à moustaches est une représentation graphique codifiée de ces cinq nombres. La représentation de x sous forme de nuage de points n'est ni très esthétique, ni très lisible, surtout si nous avons affaire à des milliers ou des millions d'observations qui se chevauchent sur le graphique[14].

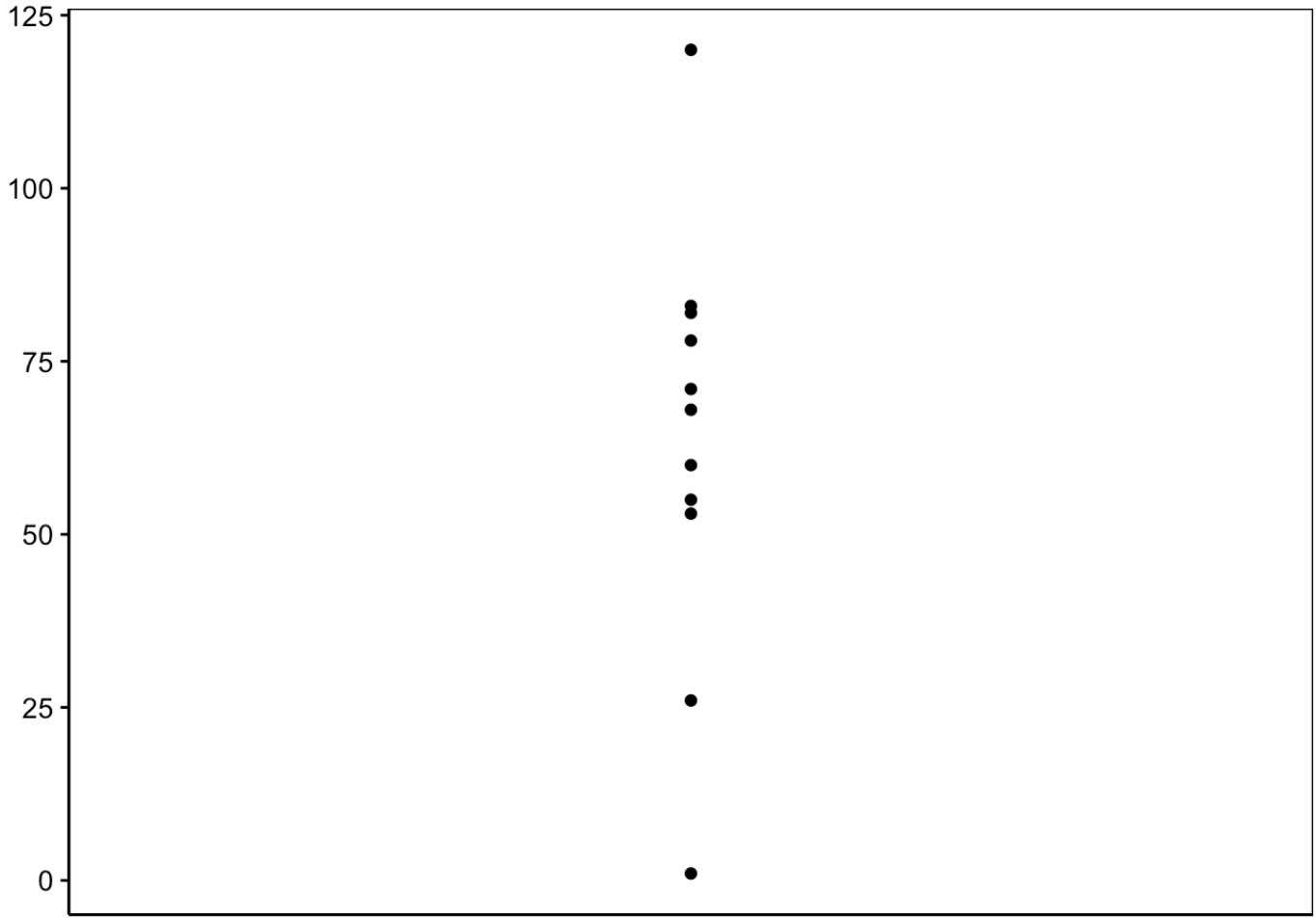


Figure 3.13: Nuage de points univarié.

La boite à moustaches va remplacer cette représentation peu lisible par un objet géométrique qui représente les cinq nombres.

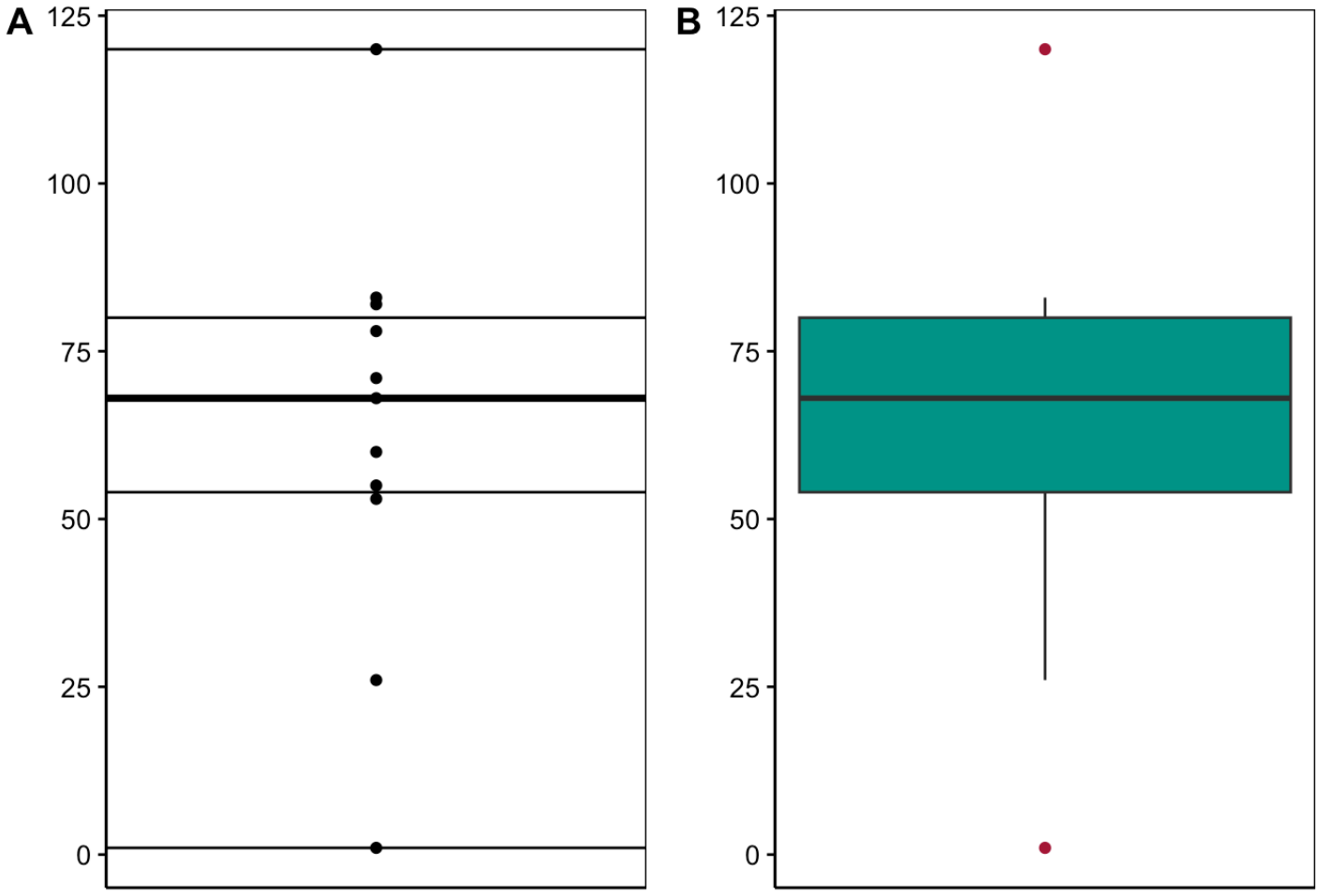


Figure 3.14: A) Nuage de points annoté avec les cinq nombres représentés par des traits horizontaux. B) Boite à moustaches obtenue pour les mêmes données que A.

Vous observez à la Fig. 3.14 que certaines valeurs minimales et maximales ne sont pas reliées à la boite à moustaches, il s'agit de **valeurs extrêmes**.

> **Règle pour déterminer s'il y a des valeurs extrêmes avec une boite à moustaches :** une valeur est considérée comme extrême si son écart par rapport à la boite est supérieur à une fois et demie la hauteur de la boite (encore appelée **espace inter-quartile** IQR correspondant à Q3 - Q1). Les tiges (ou "moustaches") qui prolongent la boite à moustaches s'arrêtent donc aux dernières valeurs les plus petites et plus grandes, mais qui rentrent encore dans une fois et demie l'IQR. Les valeurs extrêmes sont ensuite représentées individuellement par un point au-dessus et en dessous.

La boite à moustaches finale ainsi que sa description sont représentées à la Fig. 3.15 ci-dessous.

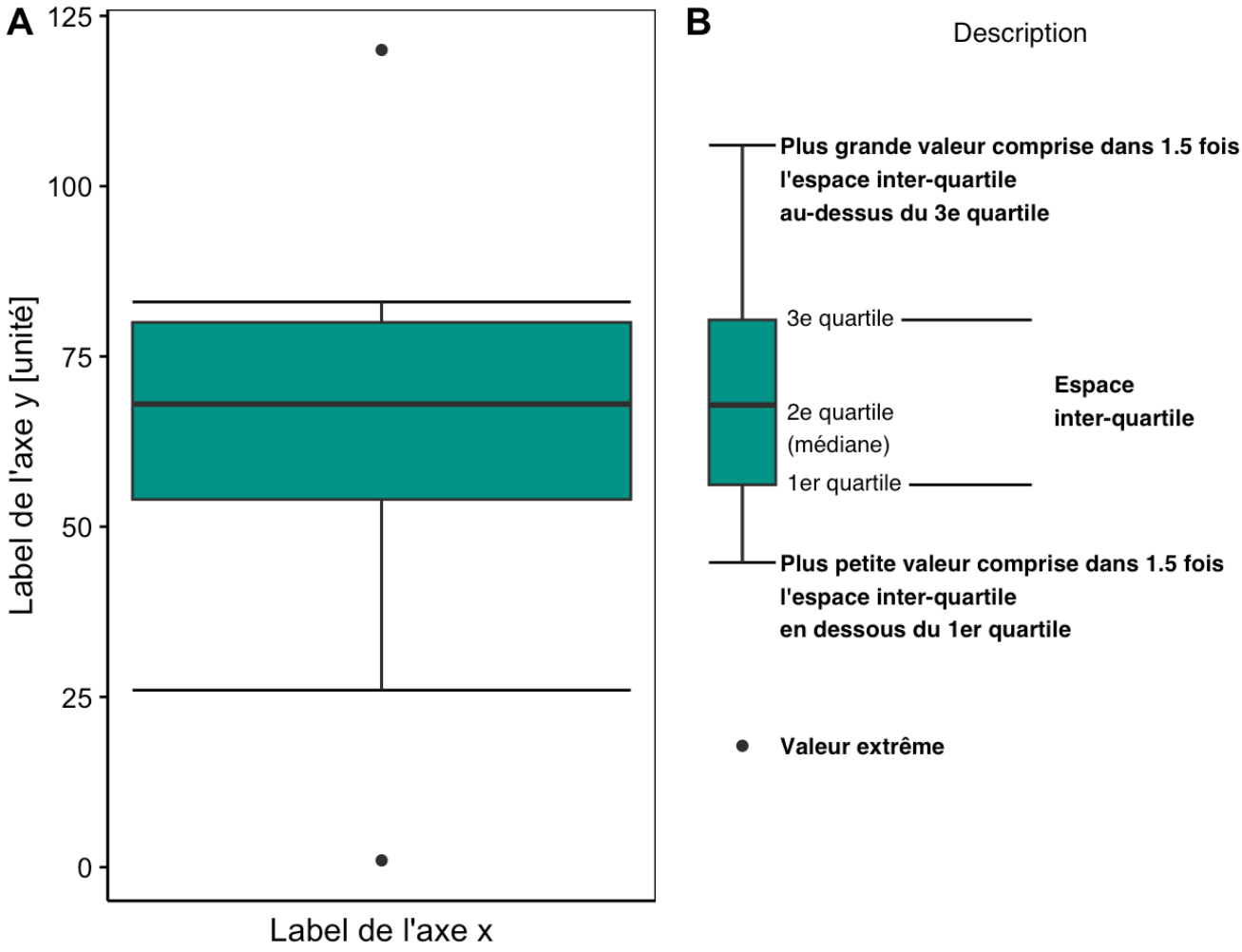


Figure 3.15: A) Boite à moustaches pour x et B) description des différents éléments constitutifs.

Dans une boîte à moustaches, une valeur est considérée comme extrême si elle est supérieure à la médiane plus 1,5 fois l'écart interquartile, ou inférieure à la médiane moins 1,5 fois l'écart interquartile.

○ Vrai ○ Faux

Vérifier

Un des gros avantages de la boite à moustaches est de mettre en évidence de manière synthétique la distribution des données sur l'axe. La **boite à moustaches parallèle** place *plusieurs* boites à moustaches côte à côte en face d'un même axe. C'est un excellent moyen de *comparer* la dispersion de données numériques en fonction des niveaux d'une variable **factor**. Les instructions dans R pour produire un graphique en boites à moustaches parallèles sont :

```
chart(data = copepoda, size ~ class) +
  geom_boxplot()
```

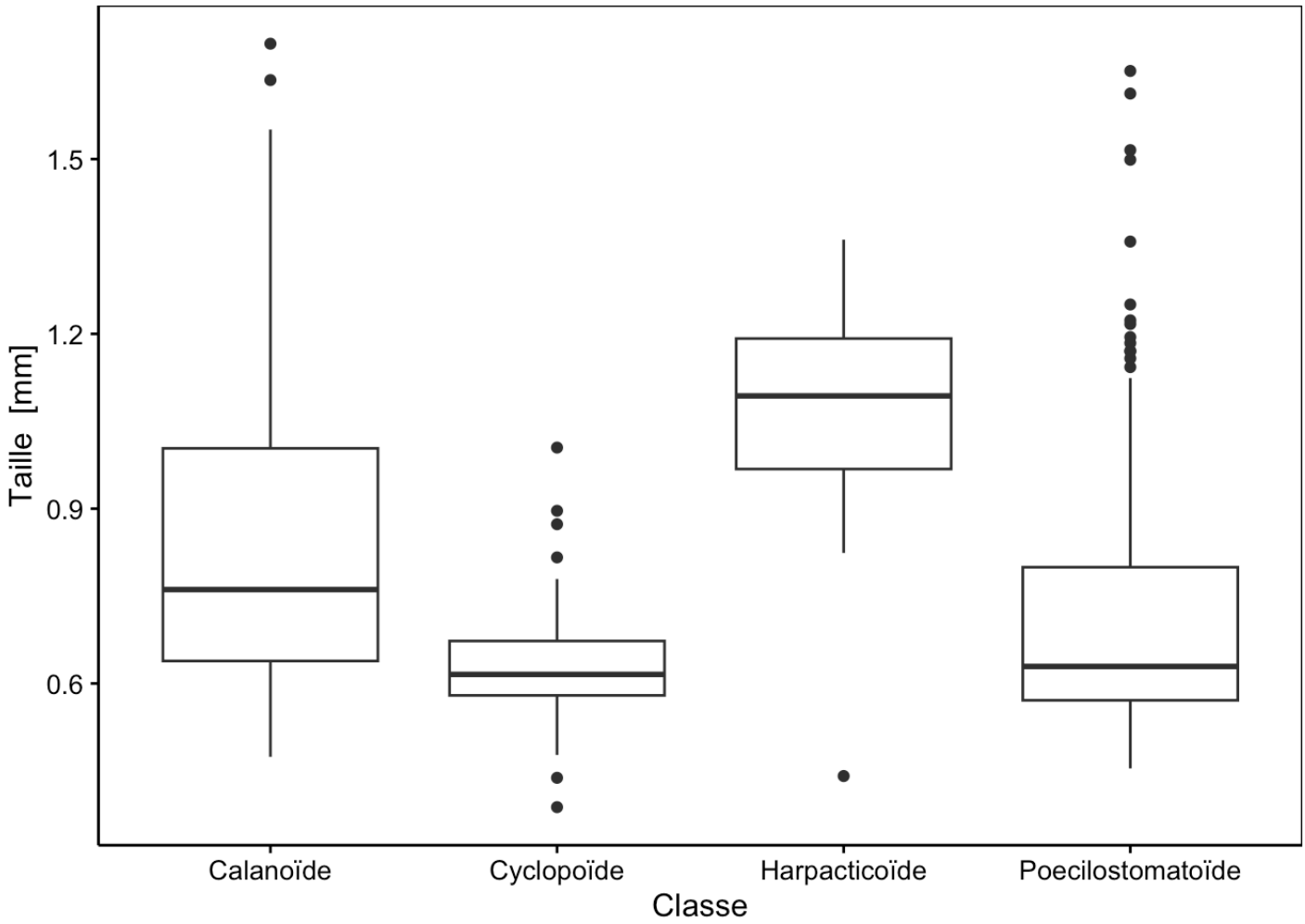


Figure 3.16: Distribution des tailles par groupes taxonomiques pour le zooplancton.

La formule à employer est `YNUM (size) ~ XFACTOR (class)` . Ensuite, pour réaliser une boite à moustaches, vous devez ajouter la fonction `geom_boxplot()` .

3.3.1 Taille de l'échantillon

Lors de la réalisation de boites à moustaches, vous devez être vigilant au nombre d'observations qui se cachent sous chacune d'elles. En effet, réaliser une boite à moustaches à partir d'échantillons ne comportant que cinq valeurs ou moins n'a *aucun* sens !

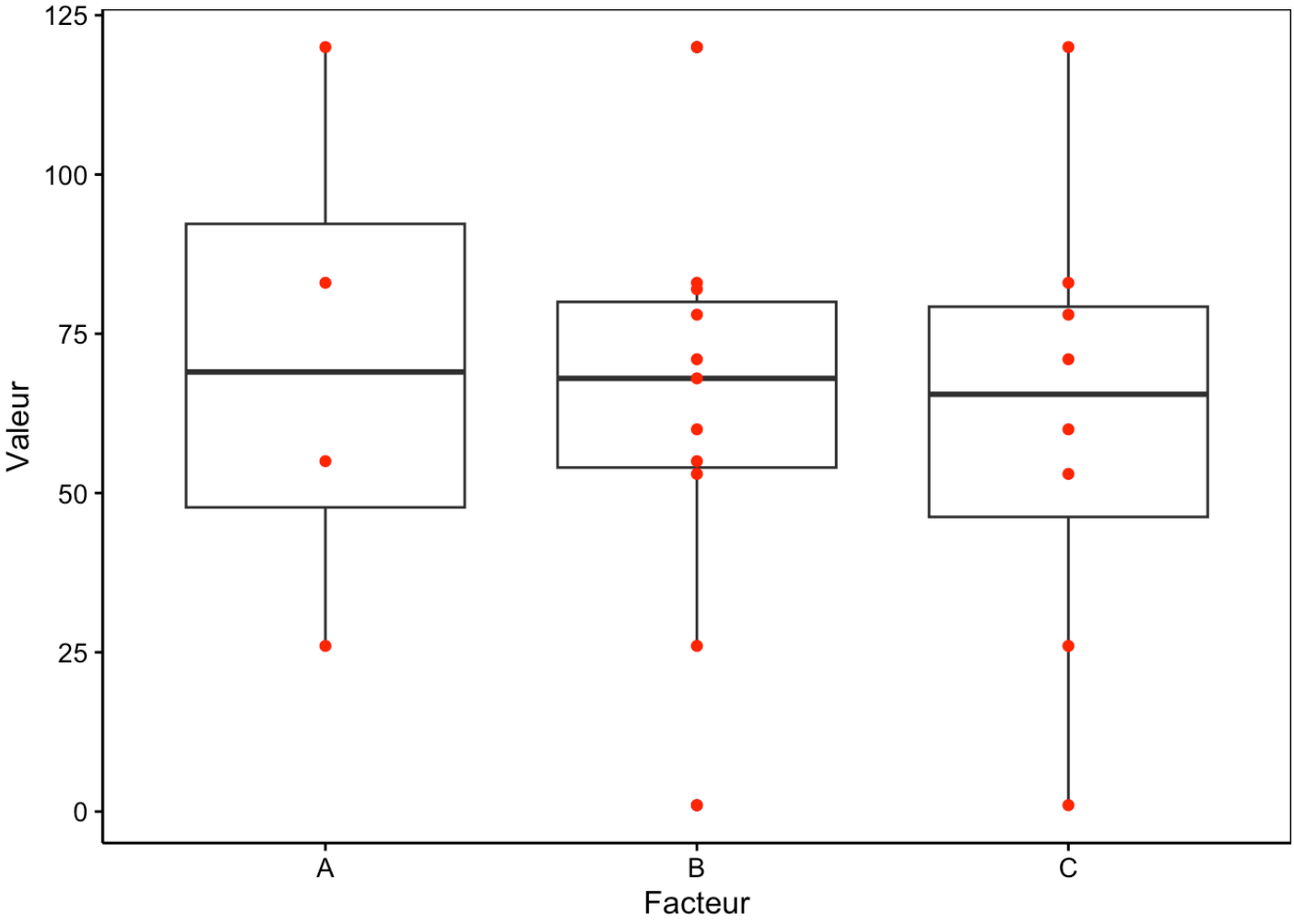


Figure 3.17: Piège des boites à moustaches : trop peu d'observations disponibles pour a .

La boite à moustaches pour le niveau `"A"` est calculée à partir de seulement quatre observations. C'est trop peu. Comme les points représentant les observations ne sont habituellement pas superposés à la boite, cela peut passer inaperçu et tromper le lecteur ! Une bonne pratique consiste à ajouter *n*, le nombre d'observations au-dessus de chaque boite. Cela peut se faire facilement avec les fonctions `give_n()` et `stat_summary()` ci-dessous[15].

```
give_n <- function(x)
  c(y = max(x) * 1.1, label = length(x))

chart(data = copepoda, size ~ class) +
  geom_boxplot() +
  stat_summary(fun.data = give_n, geom = "text", hjust = 0.5)
```

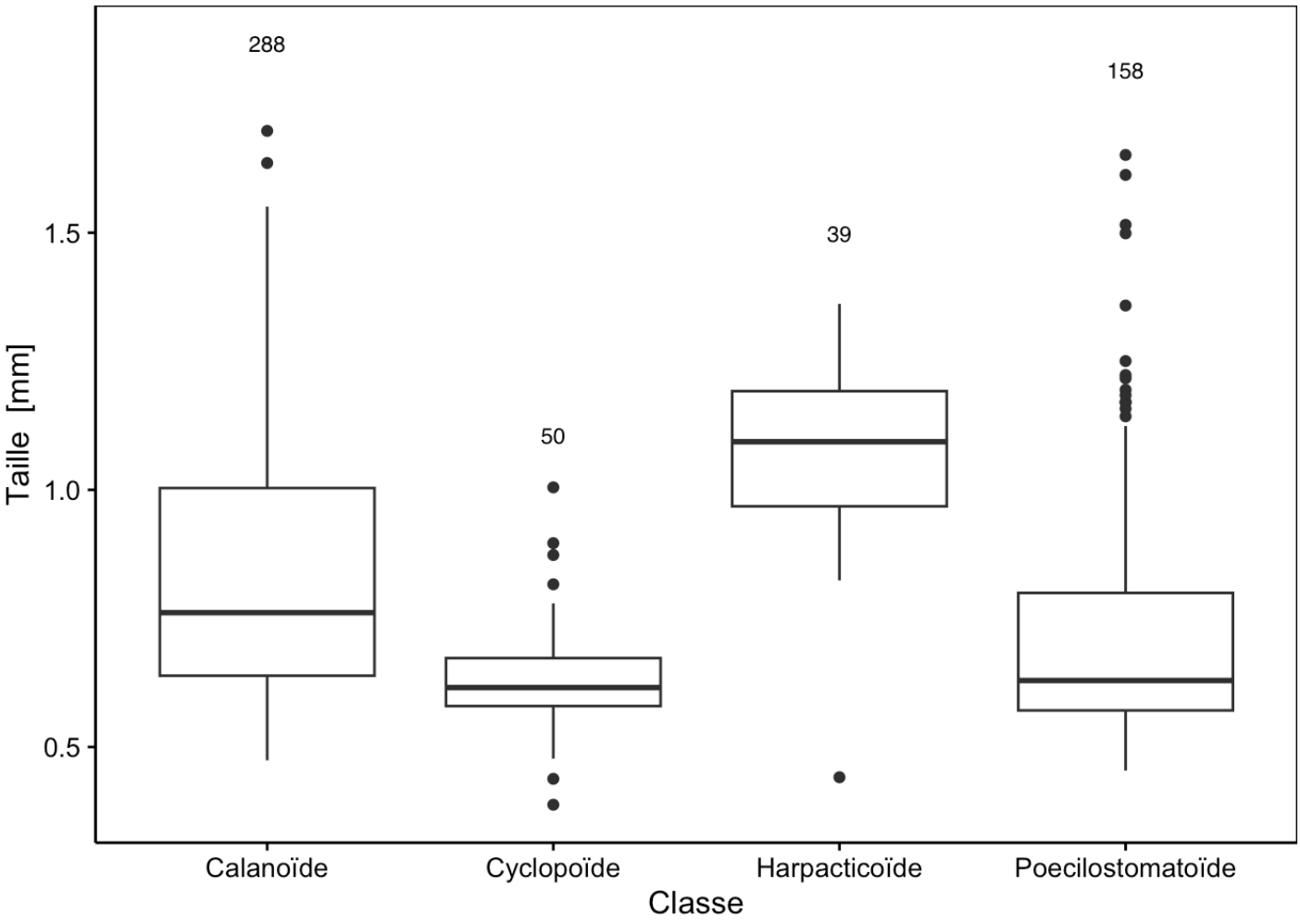


Figure 3.18: Taille de copépodes pour différents groupes taxonomiques (le nombre d'observations est indiqué au dessus de chaque boite).

La fonction `stat_summary()` ajoute des éléments à un graphique sur base d'un *calcul*. Ici, nous rajoutons du texte `geom = "text"`, sur base du calcul effectué avec notre fonction `give_n()` définie plus haut. L'argument `hjust = 0.5` indique que le texte doit être justifié horizontalement à 0.5 (= centré, car 0 = justification à gauche, et 1 = justification à droite).

3.3.2 En fonction de deux facteurs

La Fig. 3.19 présente un graphique en boites à moustaches parallèles qui combine l'usage de *deux* variables facteurs différentes. Nous allons utiliser un autre jeu de données qui a besoin d'être retravaillé pour effectuer ce graphique.

```
# Importation du jeu de données ToothGrowth
(tooth_growth <- read("ToothGrowth", package = "datasets", lang = "fr"))
```

```
# # A data.trame: [60 × 3]
#      len supp   dose
#    <dbl> <fct> <dbl>
#  1   4.2 VC      0.5
#  2  11.5 VC      0.5
#  3   7.3 VC      0.5
#  4   5.8 VC      0.5
#  5   6.4 VC      0.5
#  6  10   VC      0.5
#  7  11.2 VC      0.5
#  8  11.2 VC      0.5
#  9   5.2 VC      0.5
# 10   7   VC      0.5
# # ℹ 50 more rows
```

```r
# Remaniement et labelisation du jeu de données
tooth_growth$dose <- as.ordered(tooth_growth$dose)
tooth_growth <- labelise(tooth_growth, self = FALSE,
  label = list(
    len = "Longueur des dents",
    supp = "Supplémentation",
    dose = "Dose"
  ),
  units = list(
    len = "mm",
    supp = NA,
    dose = "mg/J"
  )
)
```

Petits commentaires sur ce code :

- La fonction `labelise()` appliquée au tableau tout entier et avec l'argument `self = FALSE` s'applique aux **colonnes** du tableau, c'est-à-dire aux variables. Ensuite, les arguments `label =` et `units =` reçoivent une `list()` nommée pour en modifier les attributs (`nom = "valeur"`). C'est une manière pratique et efficace de changer tous les labels et unités des variables en une seule étape (il n'est pas indispensable de reprendre *toutes* les variables, on peut indiquer seulement celles que l'on veut modifier).
- Nous avons utilisé `as.ordered()` à la place de `as.factor()` . Les objets "facteurs ordonnés" dans R (ou **ordered**) sont identiques aux facteurs à ceci près que l'ordre de niveaux a aussi un sens du plus petit au plus grand. Ainsi, des niveaux de `supp` : soit `"VC"` pour vitamine C ou `"OJ"` pour vitamine C dans du jus d'orange n'ont pas d'ordre précis. Nous utilisons un objet **factor**. Par contre, les doses de vitamines C `0.5 < 1 < 2` exprimées en mg/J ont un ordre. Dans ce cas, nous préférons les objets **ordered**, qui s'utilisent en pratique comme les objets **factor** dans R (mais notez bien l'indication de l'ordre des niveaux de la variable à l'aide de `<` dans `Levels:` , c'est ce qui distingue un objet **ordered** d'un objet **factor**).

```r
head(tooth_growth)$dose


# [1] 0.5 0.5 0.5 0.5 0.5 0.5
# attr(,"label")
# [1] Dose
# attr(,"units")
# [1] mg/J
# Levels: 0.5 < 1 < 2


# Réalisation du graphique (nous réutilisons give_n() ici!)
chart(data = tooth_growth, len ~ supp %fill=% dose) +
  geom_boxplot() +
  stat_summary(fun.data = give_n, geom = "text", hjust = 0.5,
    position = position_dodge(0.75))
```

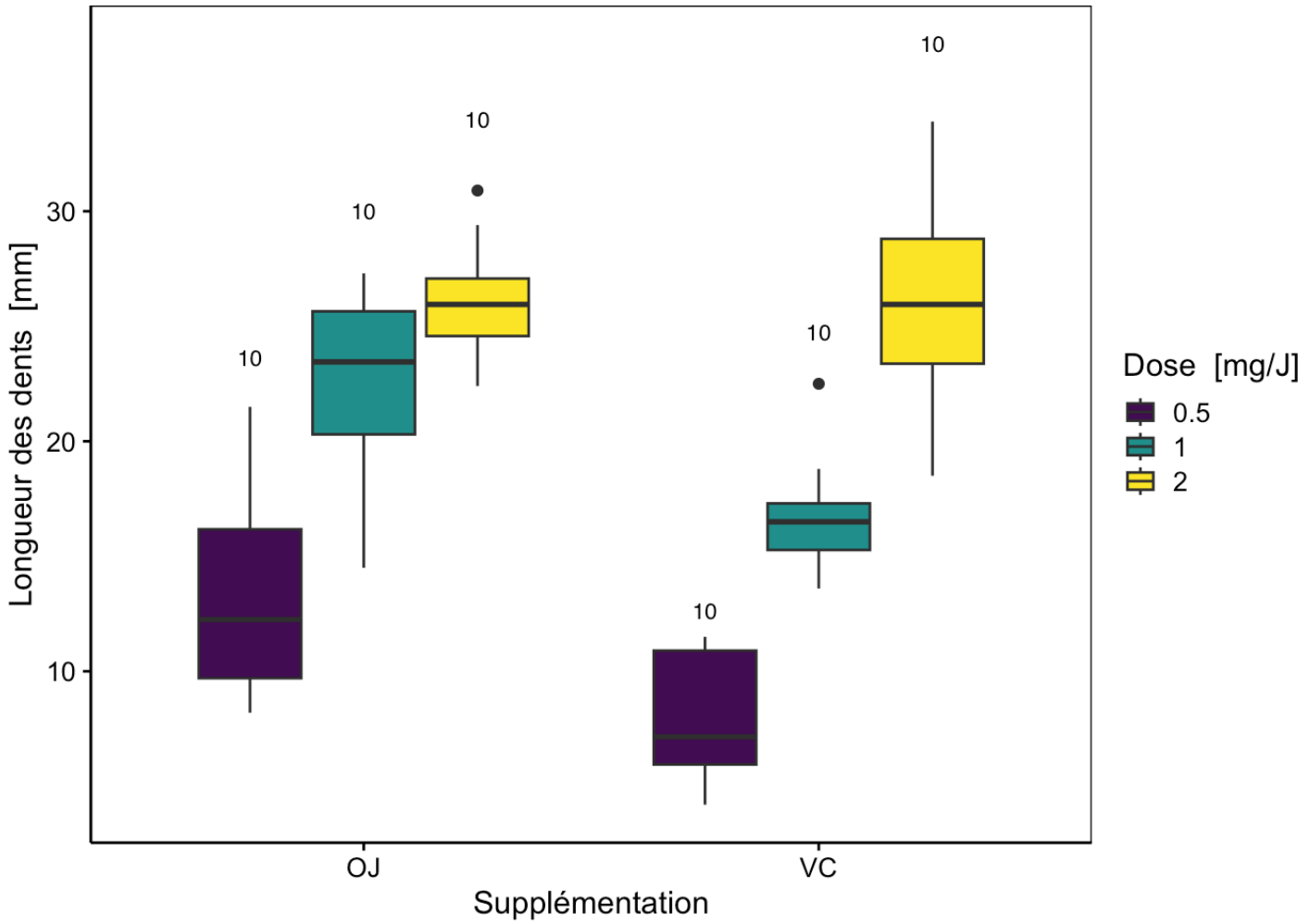


Figure 3.19: Croissance de dents de cochons d'Inde en fonction de la supplémentation (OJ = jus d'orange, VC = vitamine C) et de la dose administrée (nombre d'observations *n* indiqué au dessus de chaque boite).

À vous de jouer !

Effectuez maintenant les exercices du tutoriel **A03Lb_boxplot (Boites à moustaches)**.

```
BioDataScience1::run("A03Lb_boxplot")
```

Pour en savoir plus

- Un tutoriel boites à moustaches à l'aide de `ggplot()` présentant encore bien d'autres variantes possibles, en français.
- Box plots in `ggplot2`. Autre explication en anglais avec sortie utilisant {plotly}.
- Grouped box plots.
- Explication plus détaillée sur les cinq nombres, en anglais.

13. Notez que, lorsque la coupure tombe entre deux observations, une valeur intermédiaire est utilisée. Ici par exemple, le premier quartile est entre 53 et 55, donc, il vaut 54. Le troisième quartile se situe entre 78 et 82. Il vaut donc 80.↩
14. Il est possible de modifier la transparence des points en utilisant l'argument `alpha =` et/ou de les déplacer légèrement vers la gauche ou vers la droite de manière aléatoire pour résoudre le problème de leur chevauchement sur un graphique en nuage de points univarié en replaçant `geom_point()` par `geom_jitter()`.↩
15. La fonction `give_n()` est une **fonction personnalisée** que nous avons écrite nous-mêmes. Elle positionne du texte `y =` 10% plus haut que le `max(x)`, et ce texte est `length(x)`, la longueur du vecteur qui correspond au nombre d'observations pour `x` (`x` étant utilisé en interne par le moteur graphique). Il est possible, et même assez facile, dans R d'écrire ses *propres* fonctions. Néanmoins cela dépasse du cadre du cours pour l'instant. Pour utiliser `give_n()` dans vos documents Quarto ou R Markdown, copiez simplement sa définition dans un chunk avant de l'utiliser comme c'est fait ici. Elle est aussi réutilisable plus loin dans le même Quarto ou R Markdown, une fois qu'elle est définie.↩

3.4 Figures composées

Il arrive fréquemment de vouloir combiner plusieurs graphiques dans une même figure. Plusieurs fonctions sont à votre disposition pour cela. Il faut tout d'abord distinguer deux types de figures multigraphiques :

1. Soit il s'agit d'un seul graphique que vous souhaitez subdiviser par rapport à une ou plusieurs variables facteurs.
2. Soit il s'agit de graphiques indépendants que vous souhaitez assembler dans une même figure parce que les données ont un lien entre elles, ou parce que ces graphiques sont complémentaires pour comprendre les données.

Dans le premier cas, les fonctions `facet_XXX()` comme `facet_grid()` peuvent être employées (ou l'opérateur `|` dans une formule). Dans le second cas, la fonction `combine_charts()` est l'une des alternatives possibles.

3.4.1 Facettes

L'une des règles les plus importantes que vous devez impérativement garder à l'esprit lors de la réalisation de vos graphiques est *la simplicité*. Plus votre graphique contiendra d'information, plus il sera compliqué à décoder par vos lecteurs.

```
# Importation de données relative à la croissance de poulets
(chick_weight <- read("ChickWeight", package = "datasets", lang = "fr"))
```

```
# # A data.trame: [578 × 4]
#    weight  time chick diet
#     <dbl> <dbl> <ord> <fct>
#  1     42     0 1     1
#  2     51     2 1     1
#  3     59     4 1     1
#  4     64     6 1     1
#  5     76     8 1     1
#  6     93    10 1     1
#  7    106    12 1     1
#  8    125    14 1     1
#  9    149    16 1     1
# 10    171    18 1     1
# # ℹ 568 more rows
```

```
# Réalisation du graphique (points semi-transparents)
chart(data = chick_weight, weight ~ time %col=% diet) +
  geom_point(alpha = 0.5)
```

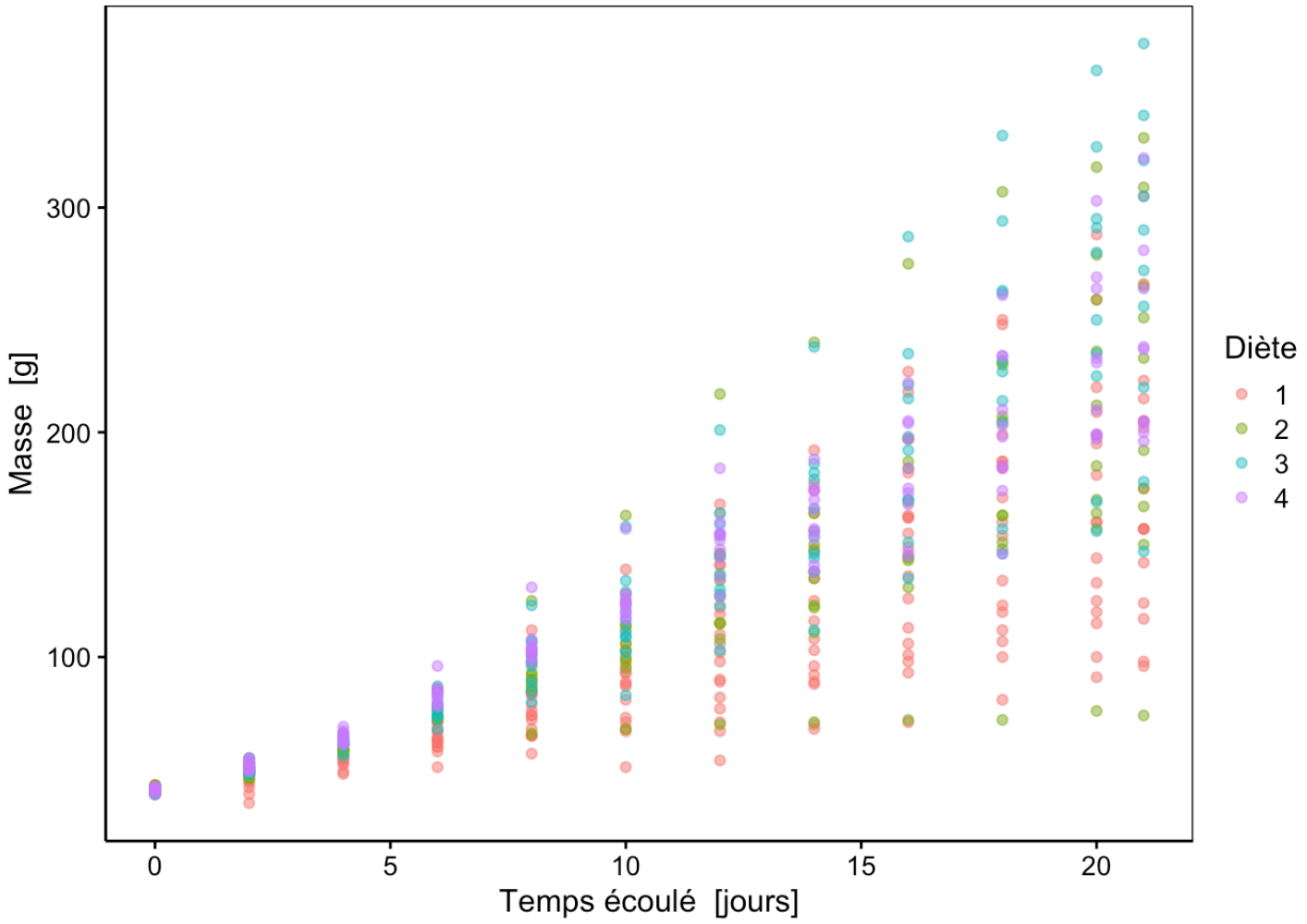


Figure 3.20: Croissance de poulets en utilisant quatre aliments différents.

Le graphique à la Fig. 3.20 est mal adapté pour montrer les différences entre les quatre aliments : tous les points sont entremêlés. Il peut typiquement être simplifié en utilisant des facettes pour représenter les résultats relatifs aux différents régimes alimentaires sur des graphiques séparés. L'information est la même, mais la lecture est beaucoup plus aisée.

```
chart(data = chick_weight, weight ~ time | diet) +
  geom_point(alpha = 0.5)
```

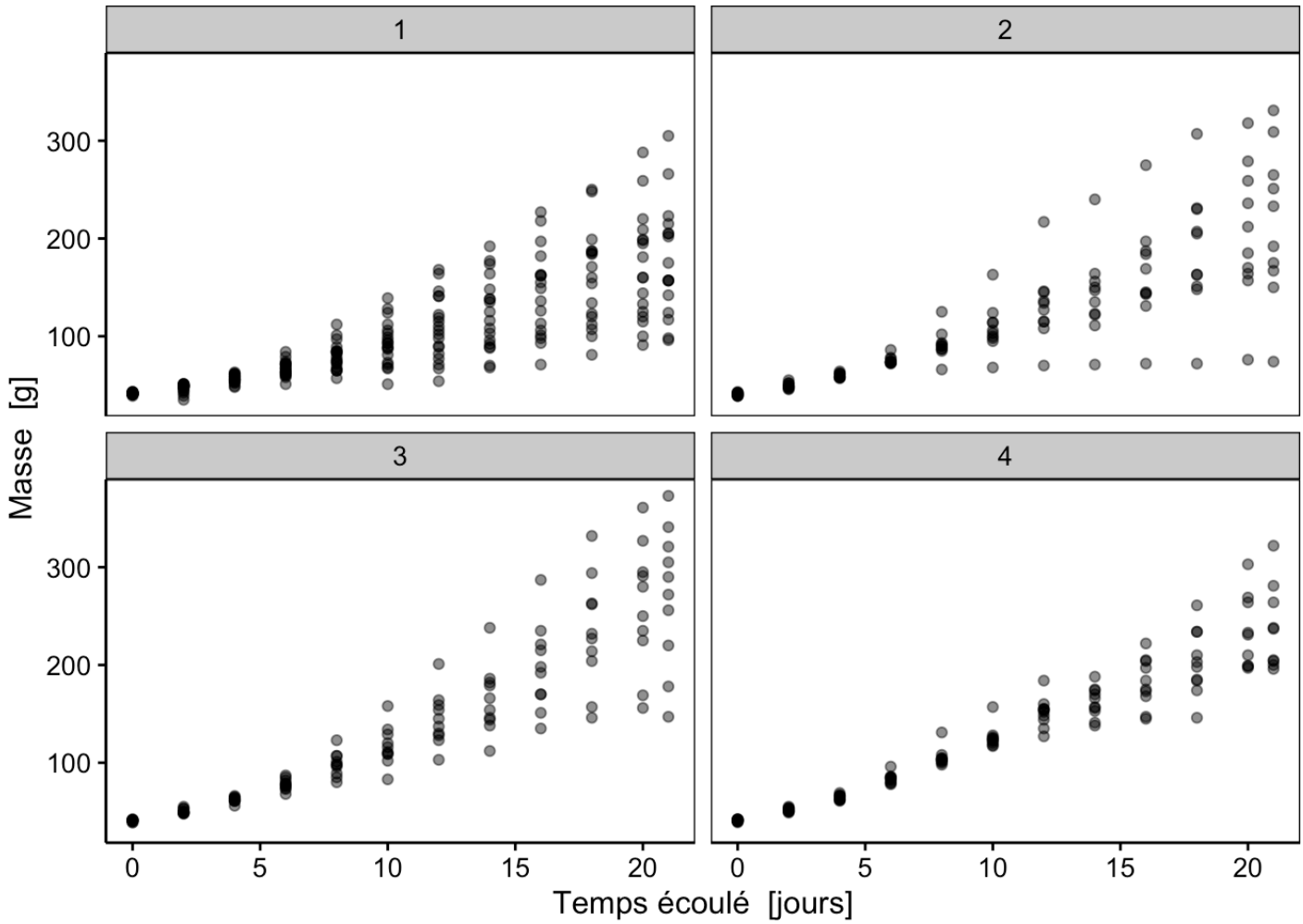


Figure 3.21: Croissance de poulets en utilisant quatre aliments différents (1-4).

Vous observez que les échelles en abscisse et en ordonnée sont similaires sur tous les graphiques. Cela permet une meilleure comparaison. Notez toutefois que, plus le nombre de facettes augmente, plus chaque graphique individuel devient petit. Faites attention à ne pas finir avec des graphiques individuels tellement petits qu'ils en deviennent illisibles !

3.4.2 Graphiques assemblés

La fonction `combine_charts()` permet de combiner plusieurs graphiques dans une figure unique. Nous l'avons déjà utilisée à plusieurs reprises. Cette fonction attend une liste (`list()`) de graphiques de type `chart()` à assembler, et il est possible d'en combiner les légendes à l'aide de `common.legend = TRUE` .

```
# Importation des données
urchin <- read("urchin_bio", package = "data.io", lang = "FR")
# Réalisation des graphiques
a <- chart(data = urchin, weight ~ height %col=% origin) +
  geom_point()

b <- chart(data = urchin, weight ~ solid_parts %col=% origin) +
  geom_point()

# Combinaison des graphiques dans une même figure
combine_charts(list(a, b), common.legend = TRUE)
```

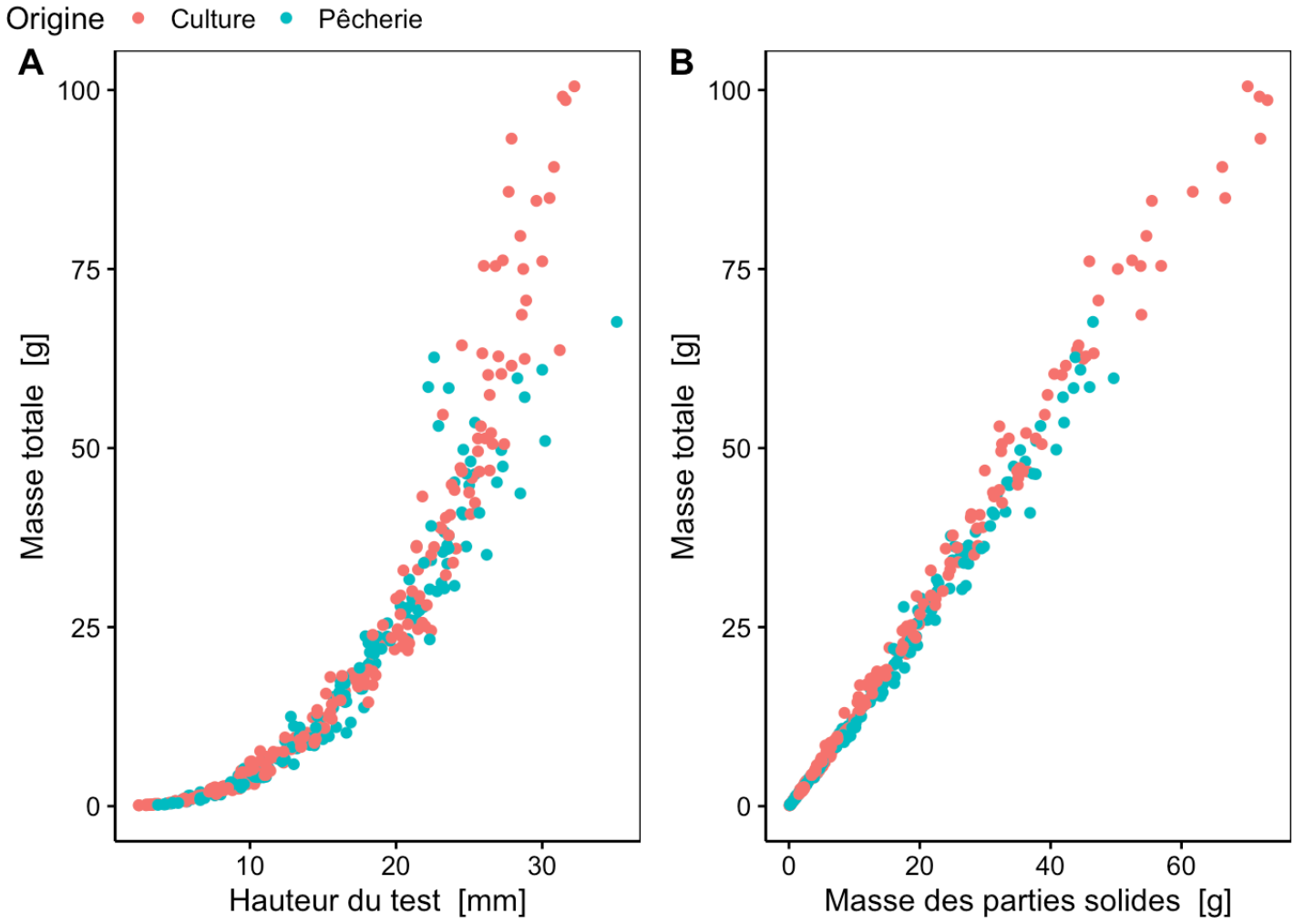
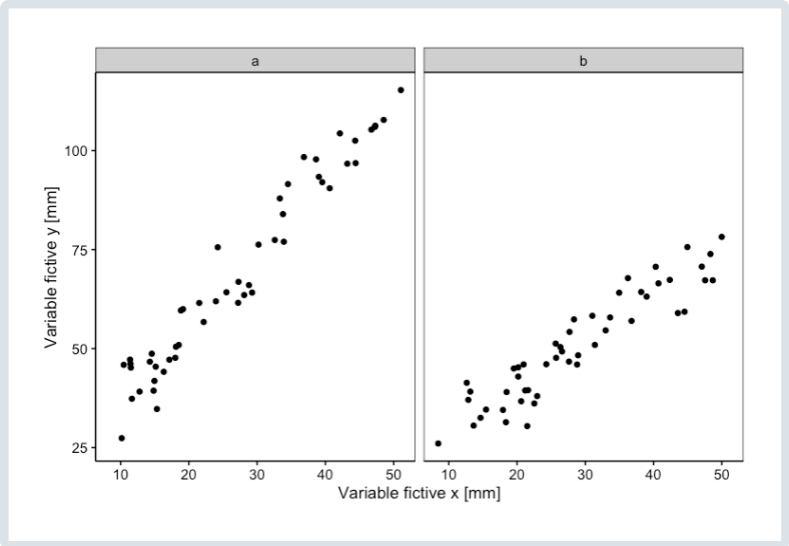


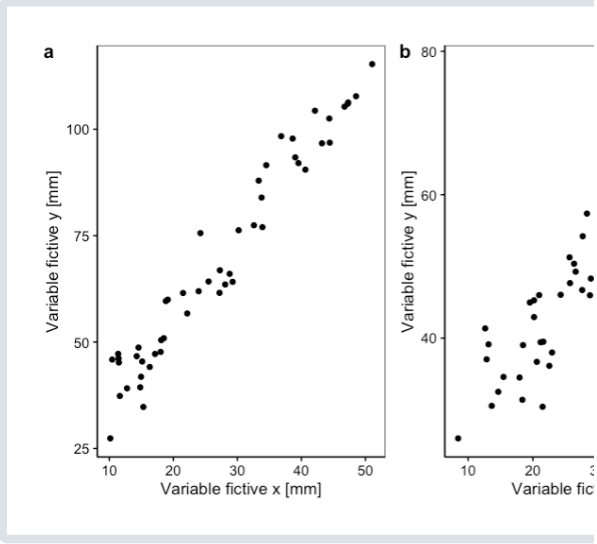
Figure 3.22: A) Masse d'oursins en fonction de leur taille et de leur origine. B) Masse totale en fonction de la masse des parties solides de ces mêmes oursins.

Il existe d'autres fonctions permettant de combiner plusieurs graphiques comme `cowplot::plot_grid()` ou encore, les fonctions du package {patchwork}, mais avec `combine_charts()` vous pourrez déjà faire beaucoup. De plus, un libellé sous forme d'une lettre majuscule est automatiquement associé à chaque sous-région de la figure composée. Cela permet d'y faire plus facilement référence dans le texte et/ou dans la légende.

À vous de jouer !

Vous êtes intéressé par la variation de la variable `y` en fonction de la variable `x` dans deux populations fictives ("a", "b"). Choisissez la paire de graphiques la plus pertinente pour permettre la comparaison entre le groupe `a` et le groupe `b`.





Check

À vous de jouer !

Effectuez maintenant les exercices du tutoriel **A03Lc_comp_fig (Graphiques composites)**.

```
BioDataScience1::run("A03Lc_comp_fig")
```

Réalisez le travail **A03la_graphe_avance**.

Travail individuel pour les étudiants inscrits au cours de Science des Données Biologiques I : visualisation à l'UMONS à terminer avant le 2025-11-17 23:59:59.

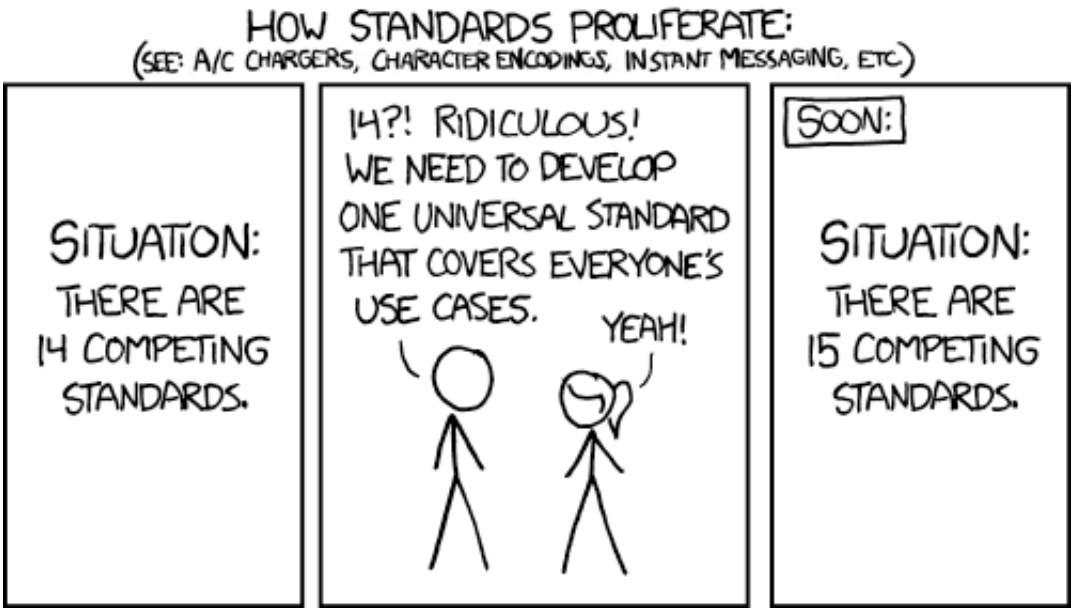
Initiez votre projet GitHub Classroom

Voyez les explications dans le fichier `README.md` .

Pour en savoir plus

- Partitionnement des graphiques en facettes. Différentes options sont présentées ici.
- Figures composées à l'aide de `cowplot::plot_grid()` avec les différentes options, aussi disponibles avec `combine_charts()` .
- Seconde possibilité pour des figures composées à l'aide de `ggpubr::ggarrange()` . `combine_charts()` fait la même chose, mais avec des valeurs par défaut légèrement différentes (`labels = "auto"` par défaut pour ce dernier, mais `labels = NULL` pour `ggpubr::ggarrange()`).
- Site du package {patchwork}. Une façon puissante et intuitive de réaliser des graphiques composites dans R.

3.5 Différents moteurs graphiques



Prolifération des standards d'après xkcd.

Depuis le début, l'ensemble des graphiques que nous vous avons réalisés utilise la fonction `chart()` du package {chart}. Cependant, il ne s'agit pas de la seule fonction permettant de réaliser des graphiques dans R, loin de là. En fait {chart} est tout récent et a été développé pour homogénéiser autant que possible les graphiques issus de trois moteurs graphiques différents : {ggplot2}, {lattice} et les graphiques de base. La fonction `chart()` a d'autres avantages également :

- Un thème par défaut qui est le plus proche possible d'un rendu typique d'une publication scientifique.
- La possibilité d'utiliser l'interface formule avec {ggplot2}.
- La cohérence des objets graphiques obtenus qui peuvent tous être combinés en une figure composite, même si ils sont produits avec des moteurs graphiques différents.
- Un libellé automatique des axes et autres éléments du graphique en fonction des attributs `label` et `units` des variables (pour l'instant, seulement les graphiques de type {ggplot2}).

```
# Importation des données
(urchin <- read("urchin_bio", package = "data.io", lang = "FR"))
```

```
# # A data.trame: [421 × 19]
#    origin   diameter1 diameter2 height buoyant_weight weight solid_parts
#    <fct>        <dbl>     <dbl>  <dbl>          <dbl>  <dbl>       <dbl>
#  1 Pêcherie       9.9      10.2    5               NA  0.522       0.478
#  2 Pêcherie      10.5      10.6    5.7             NA  0.642       0.589
#  3 Pêcherie      10.8      10.8    5.2             NA  0.734       0.677
#  4 Pêcherie       9.6       9.3    4.6             NA  0.370       0.344
#  5 Pêcherie      10.4      10.7    4.8             NA  0.610       0.559
#  6 Pêcherie      10.5      11.1    5               NA  0.610       0.551
#  7 Pêcherie      11        11      5.2             NA  0.672       0.605
#  8 Pêcherie      11.1      11.2    5.7             NA  0.703       0.628
#  9 Pêcherie       9.4       9.2    4.6             NA  0.413       0.375
# 10 Pêcherie      10.1       9.5    4.7             NA  0.449       0.398
# # ℹ 411 more rows
# # ℹ 12 more variables: integuments <dbl>, dry_integuments <dbl>,
# #   digestive_tract <dbl>, dry_digestive_tract <dbl>, gonads <dbl>,
# #   dry_gonads <dbl>, skeleton <dbl>, lantern <dbl>, test <dbl>, spines <dbl
# #   maturity <int>, sex <fct>
```

```
# Réalisation du graphique
chart(data = urchin, height ~ weight %col=% origin) +
  geom_point()
```

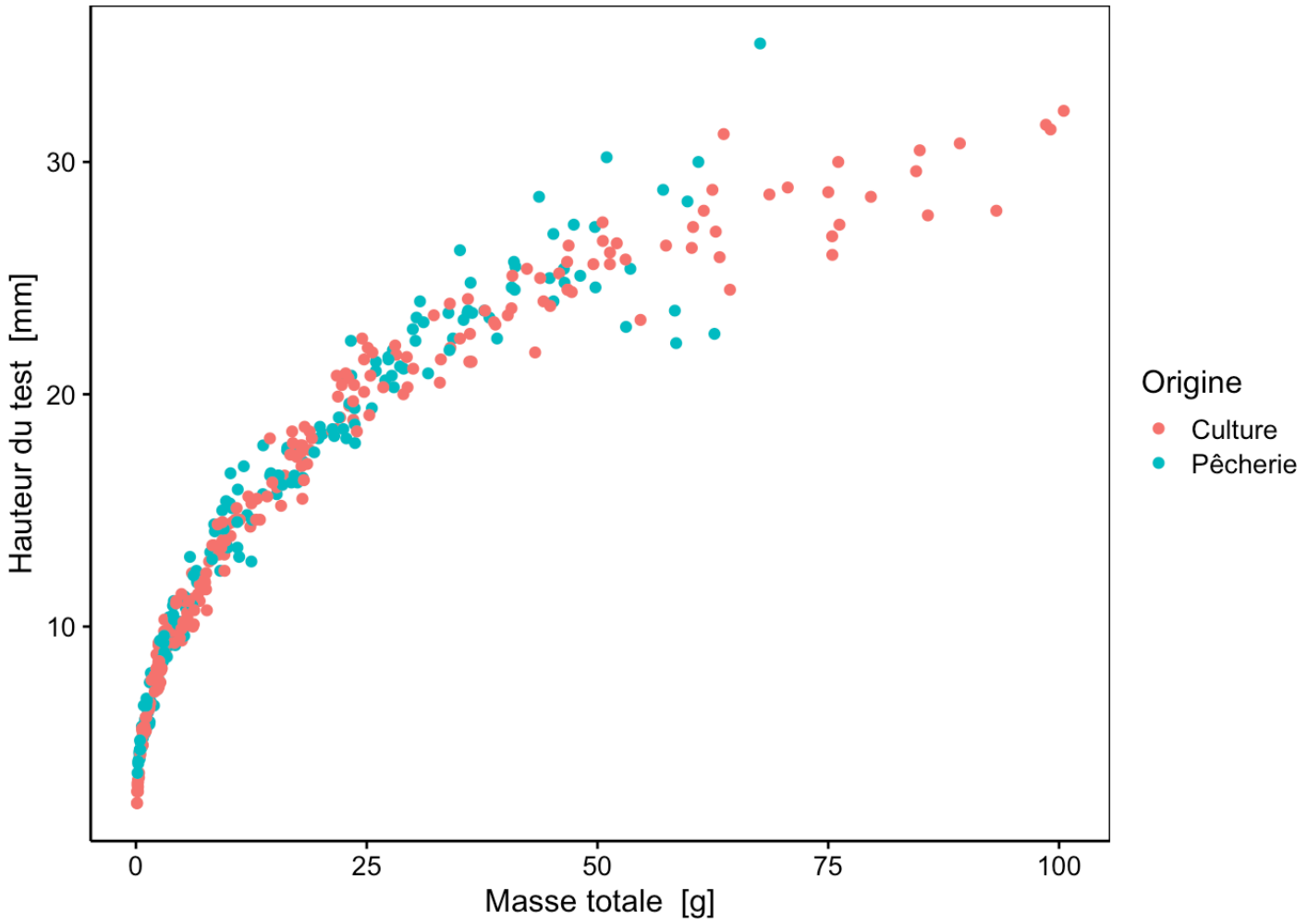


Figure 3.23: Graphique typique obtenu avec `chart()` : rendu par défaut publiable tel quel, et libellé automatique des axes avec les unités.

3.5.1 {ggplot2}

Le moteur graphique {ggplot2} est écrit pas Hadley Wickham, un personnage emblématique de la “révolution tidyverse” (même si ggplot est antérieur et n’utilise pas la syntaxe commune aux packages tidyverse) qui propose une surcouche moderne au-dessus de R. {ggplot2} implémente une “grammaire graphique” particulièrement puissante et flexible, imaginée et popularisée par le statisticien Leland Wilkinson. Par défaut, `chart()` crée en réalité un graphique {ggplot2} adapté. Voici la version {ggplot2} standard du même graphique représenté à la Fig. 3.23 :

```
ggplot(data = urchin, mapping = aes(x = weight, y = height, col = origin)) +
  geom_point()
```

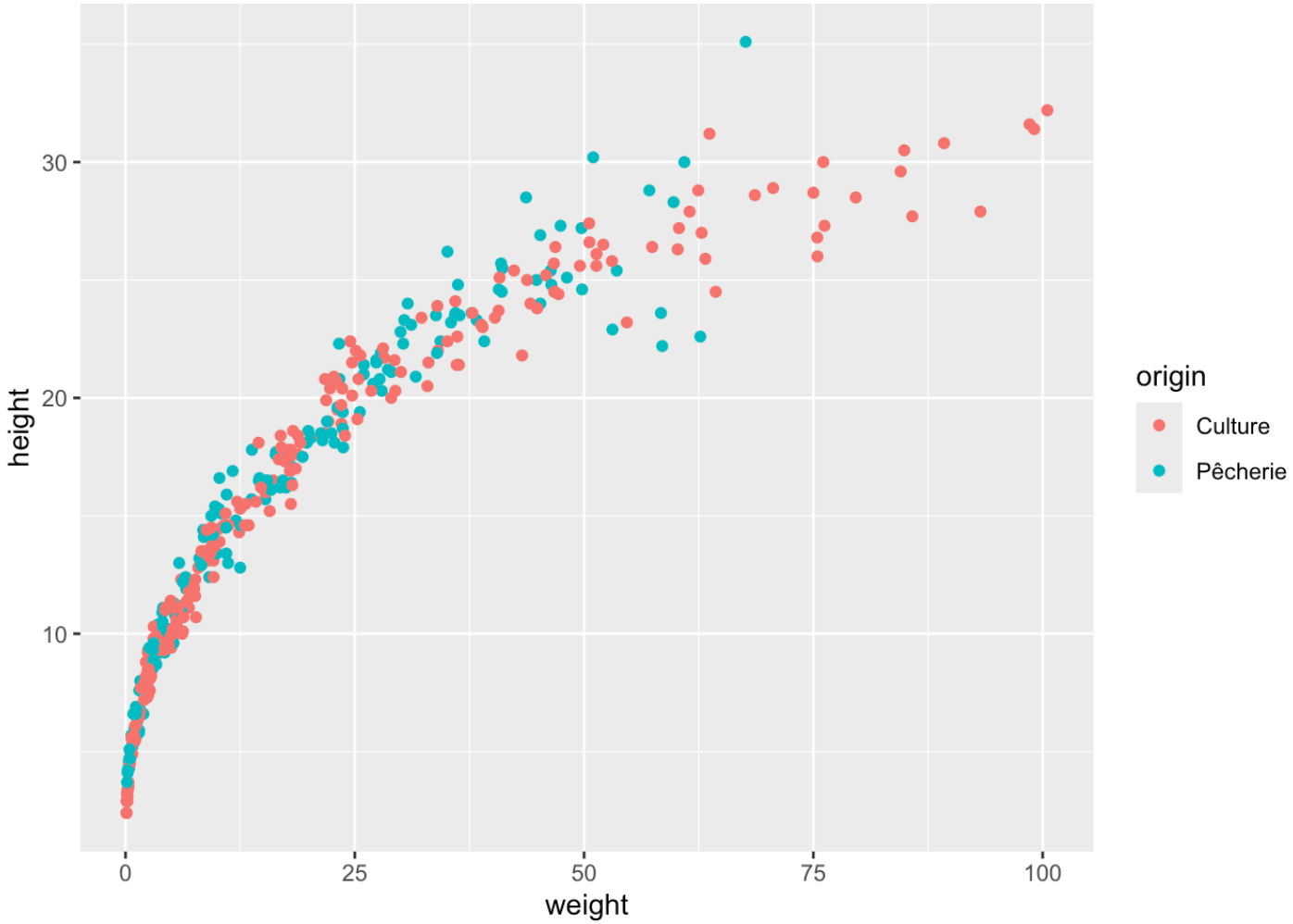


Figure 3.24: Graphique typique obtenu avec `ggplot()` (moteur graphique {ggplot2}).

En comparant les Figs. 3.23 et 3.24 (en faisant abstraction des instructions R utilisées pour l'instant), plusieurs éléments sautent immédiatement aux yeux :

- Le thème par défaut de {ggplot2} est très reconnaissable avec un quadrillage blanc sur fond gris clair. On aime ou on n'aime pas, mais il est évident que (1) ce n'est pas une présentation "standard" d'un graphique scientifique, et (2) le thème tord un peu le cou à une règle importante pour réaliser un graphique de qualité : **minimiser la quantité d'"encre" nécessaire pour représenter un graphique**, autrement dit, plus le graphique est simple et sobre, mieux c'est. Le thème par défaut de `chart()` respecte mieux tout ceci[16].

- La taille des caractères est légèrement plus grande dans la Fig. 3.23 réalisée avec `chart()` (surtout pour les nombres sur les axes). Le manque de lisibilité des parties textuelles dans un graphique est un défaut fréquent, dépendant de la résolution et de la taille de reproduction du graphique dans le document final. Le choix de `chart()` recule un peu ce risque.

- `chart()` est capable d'aller lire les métadonnées (libellés en français et unités des variables) et les utilise automatiquement pour proposer des libellés corrects et complets des axes par défaut. `ggplot()` ne peut pas le faire, et il faut utiliser la fonction `labs()` pour l'indiquer manuellement.

> De manière générale, par rapport à `ggplot()`, `chart()` a été conçu pour produire le graphique le plus proche d'un rendu final impeccable avec tous les paramètres par défaut.

Quelques règles simples vous permettent de passer des **instructions** `ggplot()` à `chart()` et *vice versa*[17] :

1. On peut toujours remplacer `ggplot()` par `chart()` dans les instructions R (à condition que le package {chart} soit chargé bien sûr, par exemple via `SciViews::R`). Dans ce cas, le thème par défaut diffère, et le libellé automatique des axes (non disponible avec `ggplot()`) est activé.

2. Avec `chart()` on peut utiliser `aes()` pour spécifier les "esthétiques" (éléments à visualiser sur le graphique) comme pour `ggplot()`, mais on peut *aussi* utiliser une interface formule plus compacte. Cette interface formule rapproche la version `chart()` des graphiques {ggplot2} d'un autre moteur de graphique dans R : {lattice}.

3. Outre les esthétiques classiques `x` et `y`, l'interface formule de `chart()` permet d'en inclure d'autres directement dans la formule à l'aide d'opérateurs spécifiques `%<esth>%=`. Par exemple, `aes(x = weight, y = height, col = origin)` dans la Fig. 3.24 se traduit en la formule plus concise `height ~ weight %col=% origin` avec `chart()` (notez la position *inversée* de `x` et `y` dans la formule puisqu'on a `y ~ x`). **Tous** les esthétiques de {ggplot2} sont supportés de cette manière.

4. Partout où `aes()` est utilisé pour les instructions {ggplot2}, on peut utiliser à la place `f_aes()` et y spécifier plutôt une formule de type `chart()`.

5. Avec `ggplot()` les facettes doivent être spécifiées à l'aide de `facet_XXX()`. À condition d'utiliser `chart()`, il est possible d'inclure les spécifications des facettes les plus utilisées directement dans la formule en utilisant l'opérateur `|`. Cette façon de procéder est identique à ce qui se fait dans {lattice} (voir plus loin).

Le point (5) mérite une petite démonstration pour comparaison :

```
a <- chart(data = urchin, height ~ weight | origin) +
  geom_point()

b <- ggplot(data = urchin, mapping = aes(x = weight, y = height)) +
  geom_point() +
  facet_grid( ~ origin)

combine_charts(list(a, b))
```

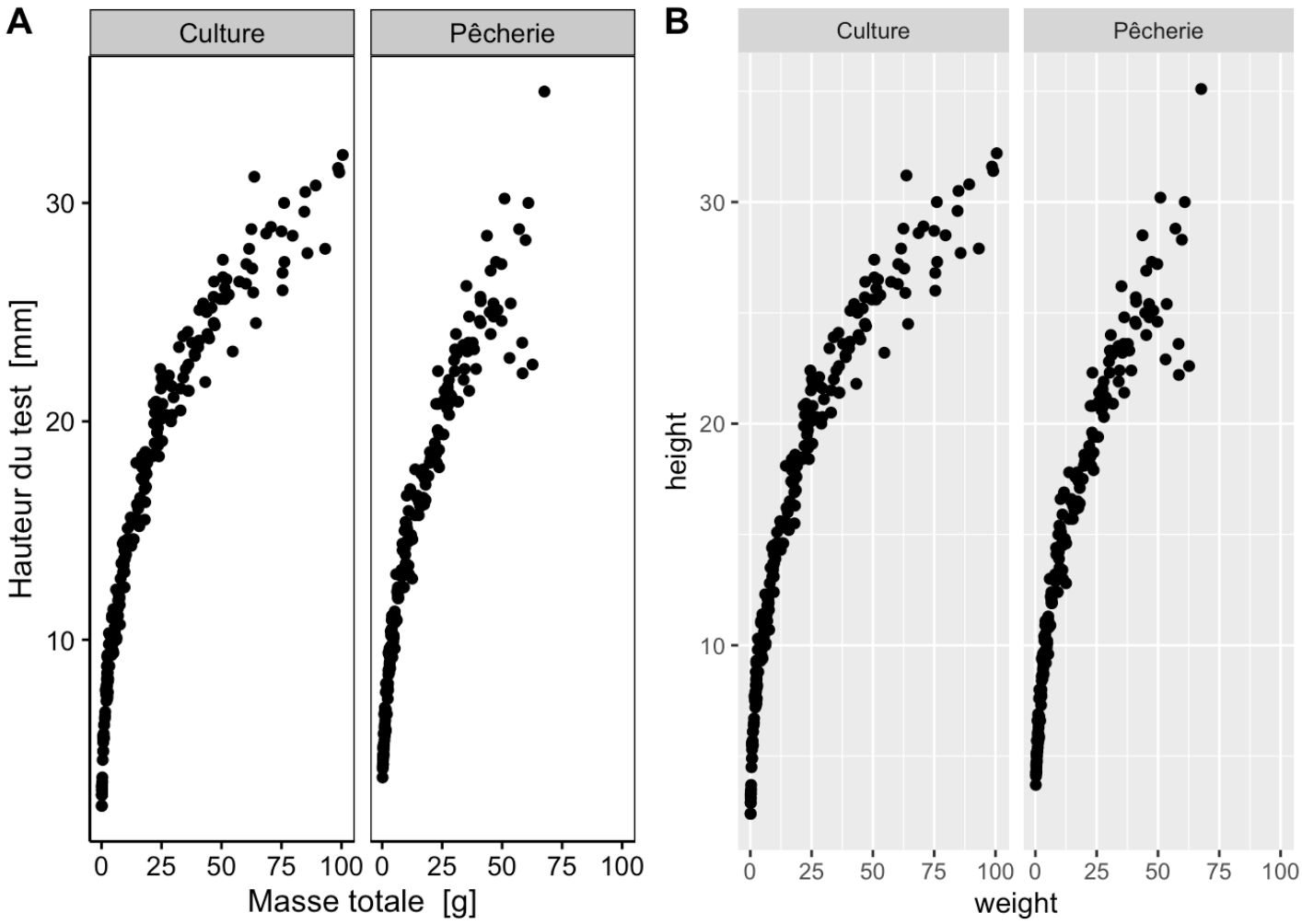


Figure 3.25: Graphique à facettes. A. version `chart()`, B. version `ggplot()`.

Enfin, pour ceux qui n'aiment pas la notation utilisant le `+` (car elle fait trop penser à une addition, ce qu'elle ne fait pas ici), vous pouvez aussi utiliser la fonction `Sgg()` du package {chart}. Cette fonction transforme toutes les commandes {ggplot2} / {chart} qui doivent être combinées à l'aide de `+` en commandes pouvant être combinées à l'aide de l'opérateur de pipe de R de base `|>`. Ce dernier opérateur est typiquement utilisé pour composer une instruction complexe à l'aide de plusieurs instructions simples (les blocs de construction). Il est donc plus logique ici. De l'aveu même de son auteur, {ggplot2} aurait utilisé `|>` à la place de `+` si cet opérateur existait à l'époque où ggplot a été conçu. Donc, `Sgg()` vient en quelque sorte corriger le tir. Pour utiliser `Sgg()`, vous remplacez le `+` par `|>`, et vous débutez le nom de votre fonction {ggplot2} par `Sgg$` (seule exception: `ggtitle()` ne devient pas `Sgg$ggtitle()`, mais `Sgg$title()`). C'est tout. Donc, vous écrirez :

```
chart(data = urchin, height ~ weight) |>
  Sgg$geom_point()
```

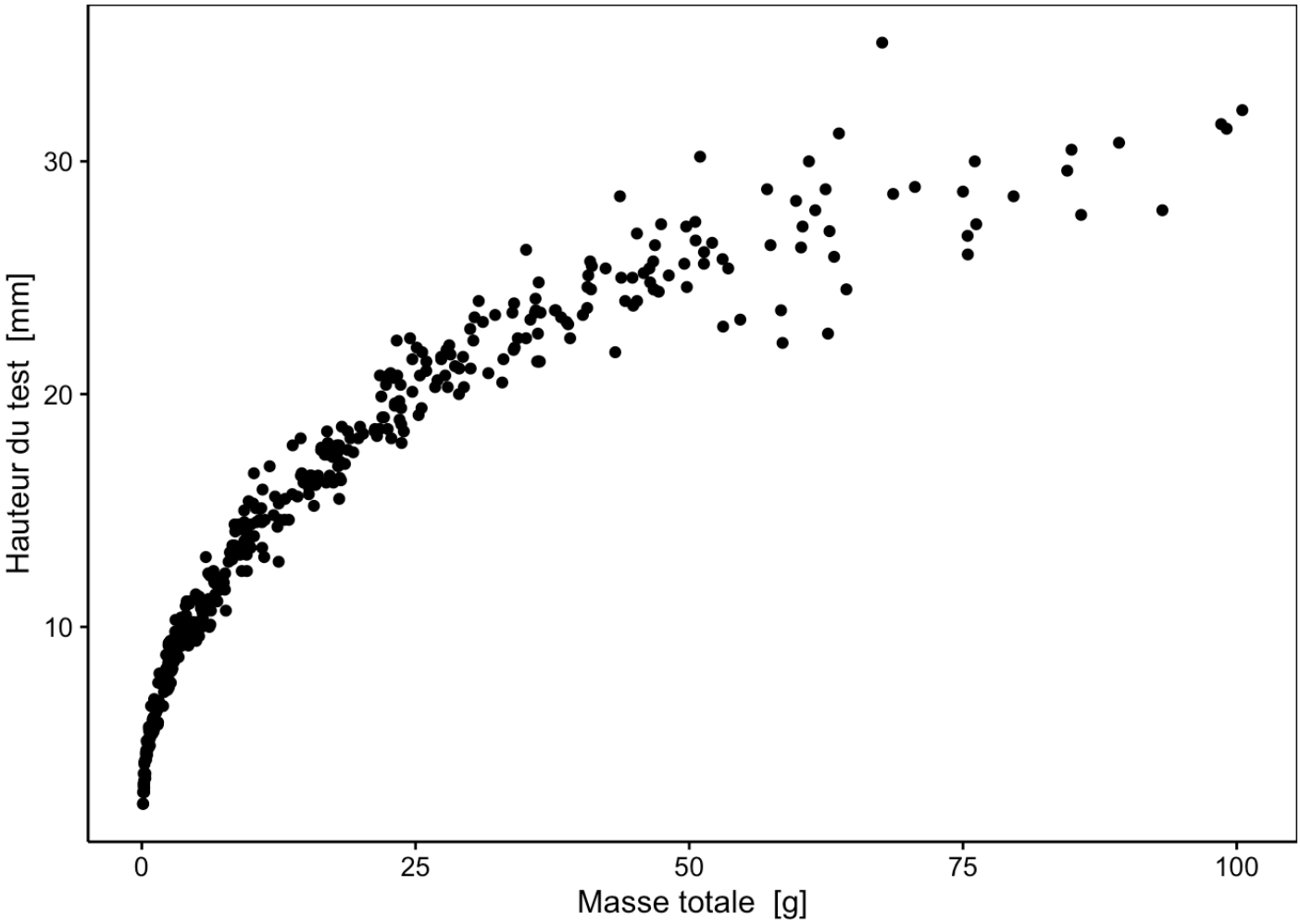


Figure 3.26: Graphique réalisé en utilisant `gg$` .

De plus, vous bénéficiez pleinement de l'aide via la complétion de RStudio avec `Sgg()` . En effet, lorsque vous avez entré `Sgg$` dans l'éditeur ou à la fenêtre console de RStudio, une liste apparaît avec toutes les options possibles.

Exercez-vous

> Pour vous exercer à réaliser des graphiques `chart() / ggplot()` , lancez votre machine virtuelle dans Saturn Cloud. Fermer le projet s'il y en a un d'ouvert. Créer un script R. Chargez les données selon votre envie et réalisez

trois graphiques inédits (qui n'ont pas été vu jusqu'ici). Consultez les liens suivants pour vous inspirer :

- https://www.r-graph-gallery.com
- http://r-statistics.co/Top50-Ggplot2-Visualizations-MasterList-R-Code.html

À vous de jouer !

Complétez votre projet de groupe en réalisant des graphiques de distribution pertinents.

Réalisez en groupe le travail **A02Ga_analysis, partie II**.

Travail en groupe de 4 pour les étudiants inscrits au cours de Science des Données Biologiques I : visualisation à l'UMONS à terminer avant le 2025-12-16 23:59:59.

Initiez votre projet GitHub Classroom

Voyez les explications dans le fichier `README.md` *, partie II.*

Pour en savoir plus

- Chapitre Data visualisation de R for Data Science qui utilise `ggplot()` .
- Site rassemblant des extensions pour ggplot2

La suite de cette section est facultative : elle est importante pour comprendre les différents types de graphiques que vous allez rencontrer avec R. Cependant, si vous vous cantonnez aux graphiques `chart()` / `ggplot()` vous pouvez déjà réaliser énormément de visualisations différentes sans forcément connaitre les autres moteurs graphiques existants dans R.

3.5.2 {lattice}

Autant {ggplot2} est complètement modulable en ajoutant littéralement à l'aide de l'opérateur `+` des couches successives sur le graphique, autant {lattice} vise à réaliser les graphiques **en une seule instruction**. {lattice} utilise également abondamment l'interface formule pour spécifier les variables à utiliser dans le graphique. La version {lattice} du graphique d'exemple est présentée à la Fig. 3.27.

```
xyplot(height ~ weight, data = urchin, groups = origin, auto.key = TRUE)
```

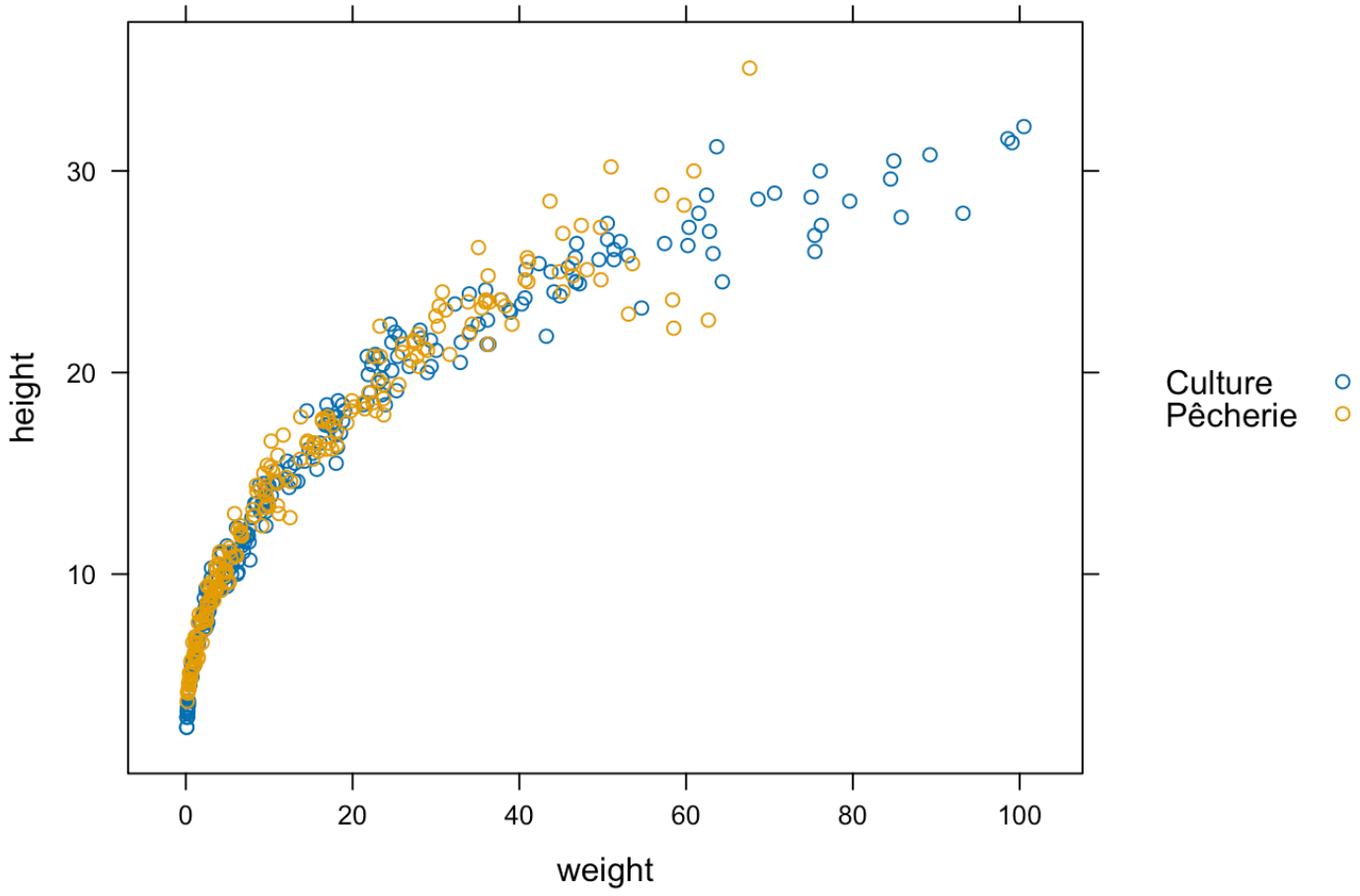


Figure 3.27: Graphique exemple réalisé avec **lattice**.

Et voici la version `chart()` utilisant le moteur {lattice}. Notez la façon d'appeler la fonction `xyplot()` de {lattice} via `chart$xyplot()` :

```
theme_sciviews_lattice(n = 2)
```

```
# Warning in seq.default(h[1], h[2], length = n): partial argument match of
# 'length' to 'length.out'
```

```
a <- chart$xyplot(height ~ weight, data = urchin, groups = origin,
  auto.key = list(space = "right", title = "Origine", cex.title = 1, columns =
  ylab = "Hauteur du test [mm]", xlab = "Masse totale [g]",
  par.settings = list(superpose.symbol = list(col = scales::hue_pal()(2))))
```

```
# Warning in seq.default(h[1], h[2], length = n): partial argument match of
# 'length' to 'length.out'
# Warning in seq.default(h[1], h[2], length = n): partial argument match of
# 'length' to 'length.out'
```

```
# Warning in rep(tck, length = length(lab)): partial argument match of 'length
# to 'length.out'
# Warning in rep(tck, length = length(lab)): partial argument match of 'length
# to 'length.out'
```

```
b <- chart(data = urchin, height ~ weight %col=% origin) +
  geom_point()

combine_charts(list(a, b))
```

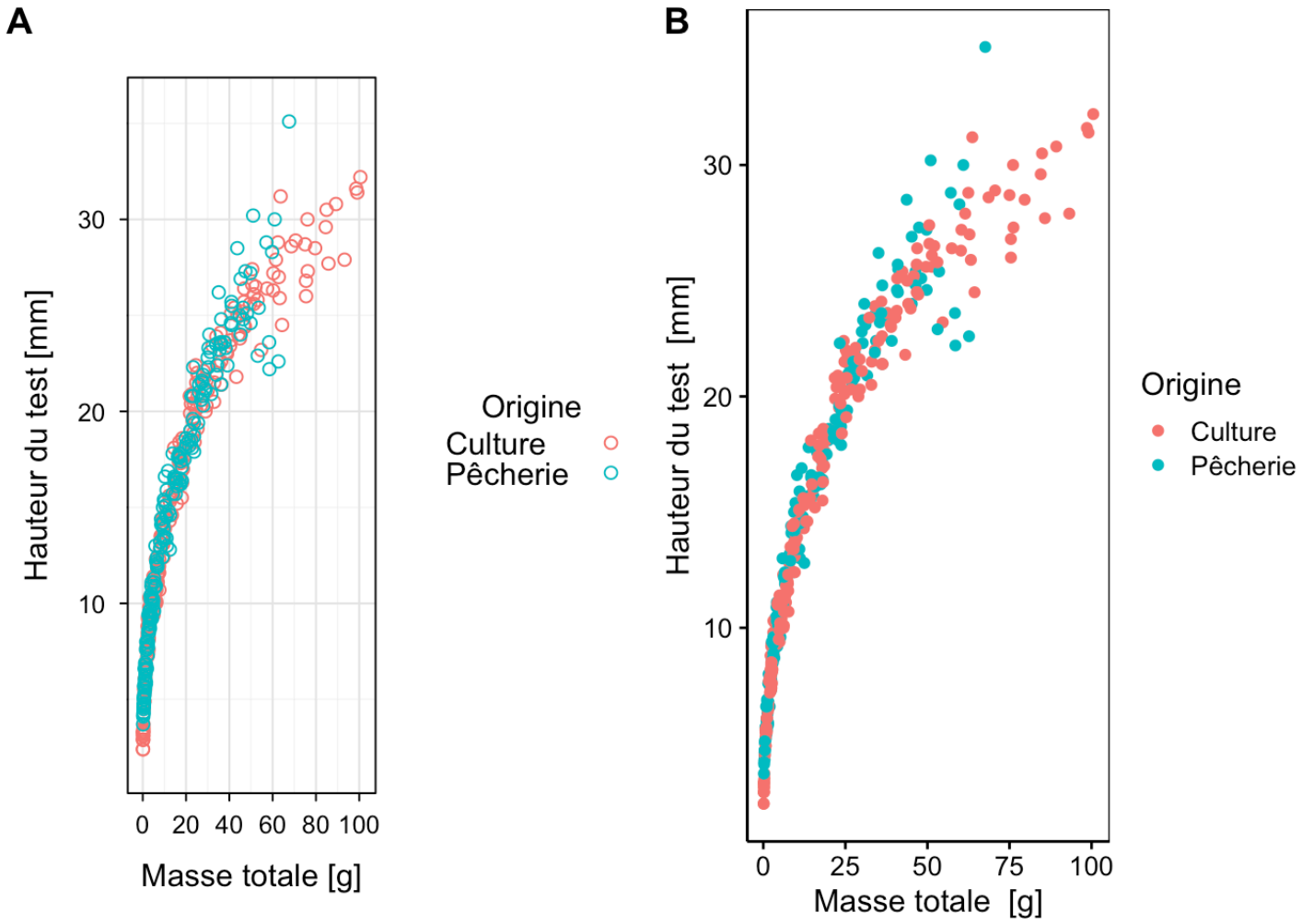


Figure 3.28: Graphique exemple réalisé avec `chart()` A. avec le moteur **lattice**, B. avec le moteur **ggplot2**.

La quantité d'instructions nécessaires pour rendre la version {lattice} proche de la version {ggplot2} devrait disparaître dans les prochaines versions de `chart()` . Un autre objectif est aussi de gommer le plus possible les différences entre les rendus des différents moteurs de graphiques R, et en particulier entre {ggplot2} et {lattice}. Comparez la Fig. 3.28A avec la Fig. 3.27 pour apprécier le gain déjà obtenu en matière d'homogénéisation.

> Par rapport à {ggplot2}, les graphiques {lattice} sont moins flexibles du fait qu'ils doivent être spécifiés en une seule instruction. Cependant, ils sont parfois plus rapides à générer (appréciable quand il y a beaucoup de points à tracer) ! {lattice} offre également quelques types de graphiques non supportés par {ggplot2} comme les graphiques en 3D à facettes, par exemple.

Voici un graphique à facettes réalisé avec `chart()` et le moteur {lattice}. Notez que la formule utilisée est *identique* à celle employée pour la version {ggplot2} avec `chart()`.

```
chart$xyplot(data = urchin, height ~ weight | origin,
  scales = list(alternating = 1),
  xlab = "Masse totale [g]", ylab = "Hauteur du test [mm]")
```

```
# Warning in seq.default(h[1], h[2], length = n): partial argument match of
# 'length' to 'length.out'
# Warning in seq.default(h[1], h[2], length = n): partial argument match of
# 'length' to 'length.out'
```

```
# Warning in rep(tck, length = length(lab)): partial argument match of 'length
# to 'length.out'
# Warning in rep(tck, length = length(lab)): partial argument match of 'length
# to 'length.out'
# Warning in rep(tck, length = length(lab)): partial argument match of 'length
# to 'length.out'
# Warning in rep(tck, length = length(lab)): partial argument match of 'length
# to 'length.out'
```

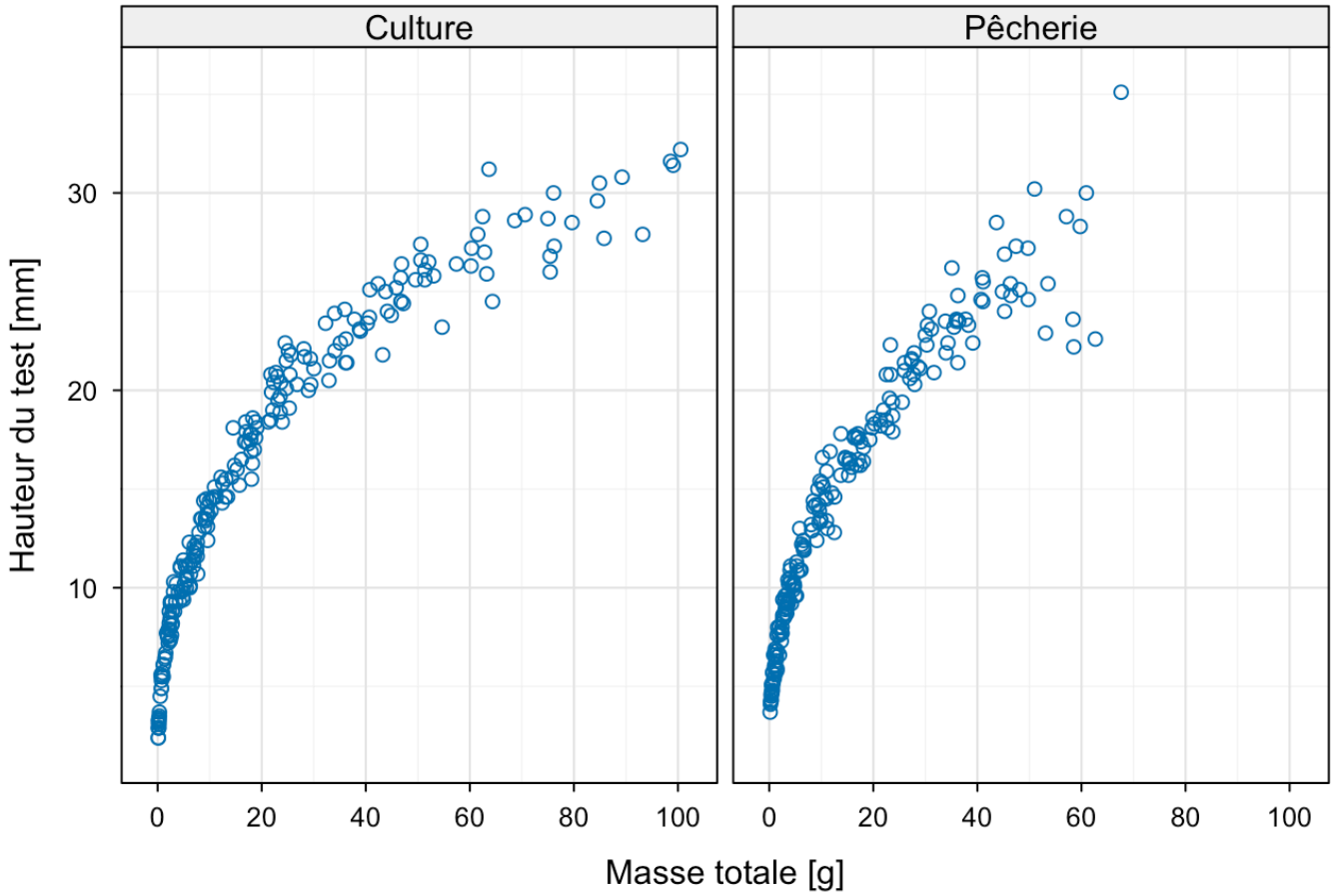


Figure 3.29: Graphique à facettes, avec `chart()` version {lattice}.

Mis à part les instructions additionnelles encore nécessaires dans cette version de `chart()`, l'appel et le rendu sont très similaires par rapport à la version {ggplot2} du même graphique avec `chart()` :

```
chart$xyplot(data = urchin, height ~ weight | origin,
  scales = list(alternating = 1),
  ylab = "Hauteur du test [mm]", xlab = "Masse totale [g]")
```

```
# Warning in seq.default(h[1], h[2], length = n): partial argument match of
# 'length' to 'length.out'
# Warning in seq.default(h[1], h[2], length = n): partial argument match of
# 'length' to 'length.out'
```

```
# Warning in rep(tck, length = length(lab)): partial argument match of 'length
# to 'length.out'
# Warning in rep(tck, length = length(lab)): partial argument match of 'length
# to 'length.out'
# Warning in rep(tck, length = length(lab)): partial argument match of 'length
# to 'length.out'
# Warning in rep(tck, length = length(lab)): partial argument match of 'length
# to 'length.out'
```

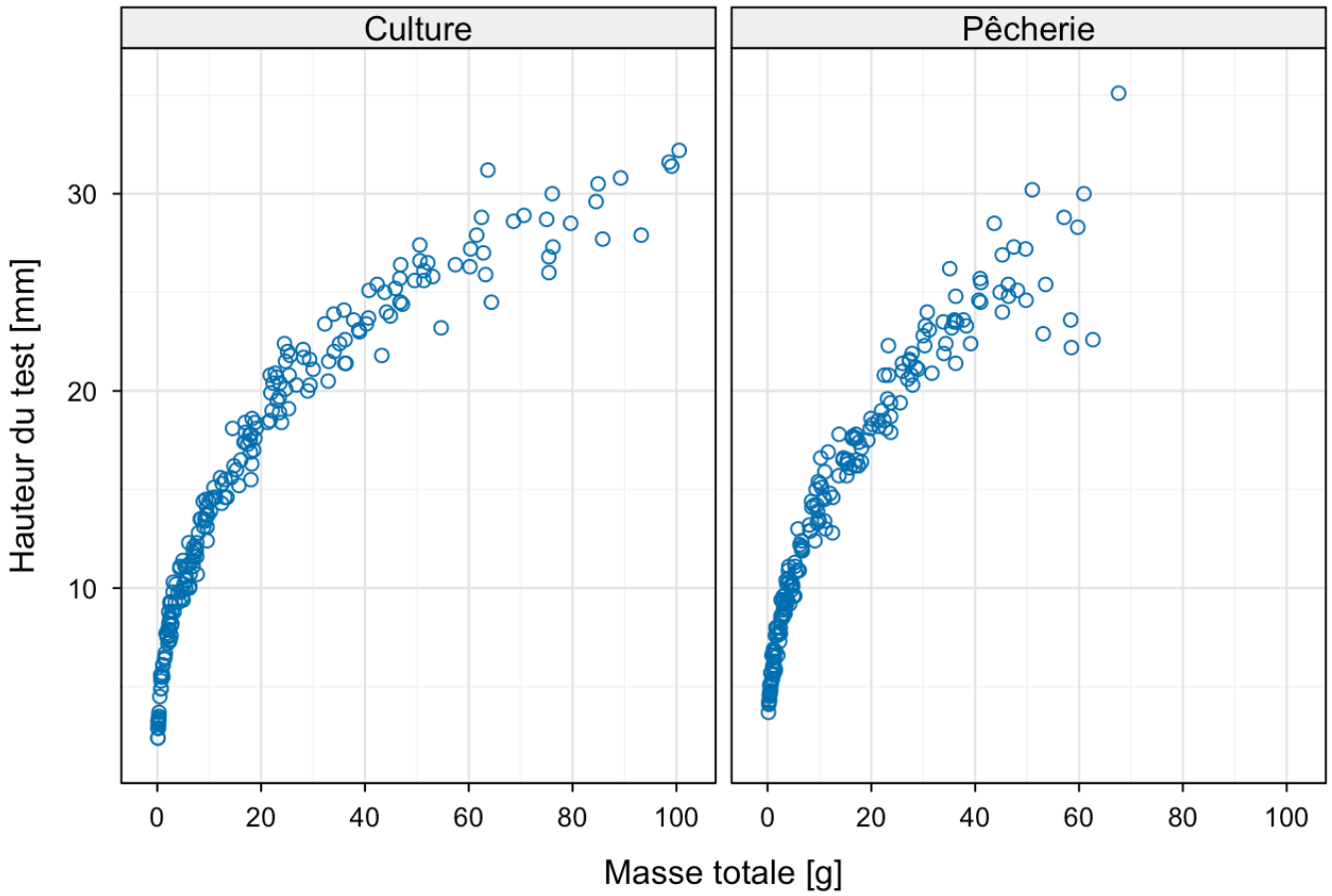


Figure 3.30: Graphique à facettes, avec `chart()` version {lattice}.

Vous noterez que pour réaliser un graphique {lattice} à l'aide de `chart()`, vous devez rajouter `$nom_de_function_lattice` après `chart$`. Aussi, le thème spécifique SciViews et les labels et unités automatiques ne sont pas encore supportés. Les graphiques {lattice} ne supportant pas l'opérateur `+` pour ajouter des couches comme pour {ggplot2}, vous

devez spécifier l'ensemble des options comme arguments de la fonction `chart$fun()`. Cela peut devenir pénible s'il y en a beaucoup. Toutefois {lattice} peut rendre d'énormes services pour des graphiques très compliqués, grâce à sa vitesse nettement supérieure à {ggplot2}.

3.5.3 Graphiques de base

Comme son nom le suggère, le moteur graphique de base est celui qui est implémenté de manière native dans R. Il est donc utilisé un peu partout. Il est vieillissant et est plus difficile à manipuler que {ggplot2} certainement, et même que {lattice}. Néanmoins, il est très flexible et rapide, et encore très utilisé… mais son rendu par défaut n'est plus vraiment au goût du jour. Voici notre graphique d'exemple rendu avec le moteur graphique R de base :

```
plot(urchin$weight, urchin$height,
  col = c("red", "darkgreen")[urchin$origin], pch = 1)
legend(x = 80, y = 10, legend = c("Culture", "Pêcherie"),
  col = c("red", "darkgreen"), pch = 1)
```

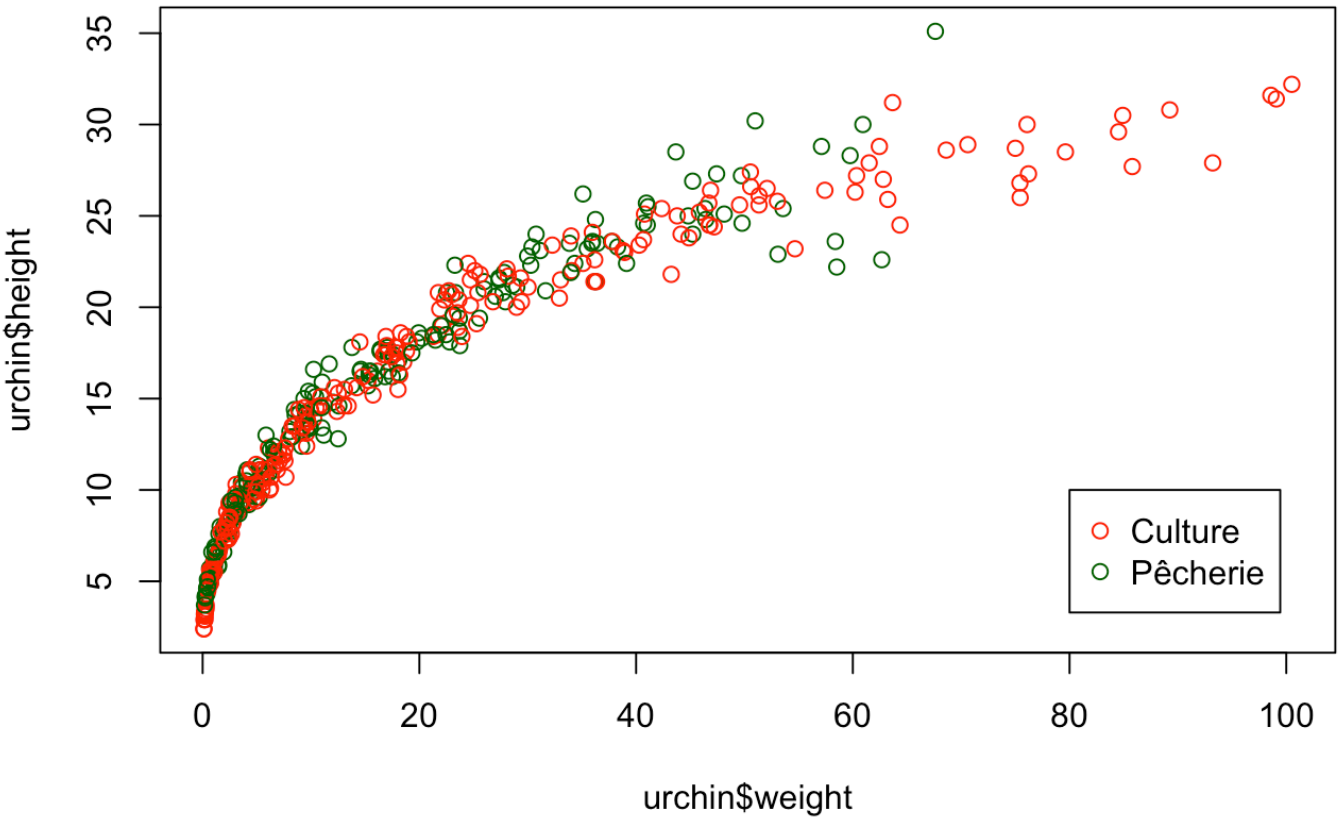


Figure 3.31: Graphique exemple réalisé avec le moteur graphique R de base.

Vous rencontrerez très fréquemment la fonction `plot()`. C'est une fonction dite **générique** dont le comportement change en fonction de l'objet fourni en premier argument. Ainsi, elle réalise le graphique le plus pertinent à chaque fois en fonction du contexte. Notez tout de suite les instructions un peu confuses nécessaires pour spécifier la couleur souhaitée en fonction de l'origine des oursins. Le moteur graphique de base ne gère **pas** automatiquement des aspects plus complexes du graphique, tels que le positionnement d'une légende. Donc, à moins d'avoir prévu la place suffisante *avant* de tracer le graphique, nous ne pouvons que l'inclure à l'intérieur du cadre du graphique dans un second temps à l'aide de la fonction `legend()`. Comme cette dernière ne comprend rien à ce qui a été réalisé jusqu'ici, il faut lui respécifier les couleurs, formes et tailles de points utilisés ! C'est un des aspects pénibles du moteur graphique R de base.

Voici maintenant une version `chart()` de ce graphique de base :

```
chart$base({
  par(mar = c(5.1, 4.1, 4.1, 6.1))
  plot(urchin$weight, urchin$height,
    col = scales::hue_pal()(2)[urchin$origin], pch = 19, cex = 0.8,
    xlab = "Masse totale [g]", ylab = "Hauteur du test [mm]")
  legend(x = 105, y = 20, legend = c("Culture", "Pêcherie"), title = "Origine"
    col = scales::hue_pal()(2), pch = 19, bty = "n", cex = 0.8, y.intersp = 2)
})
```

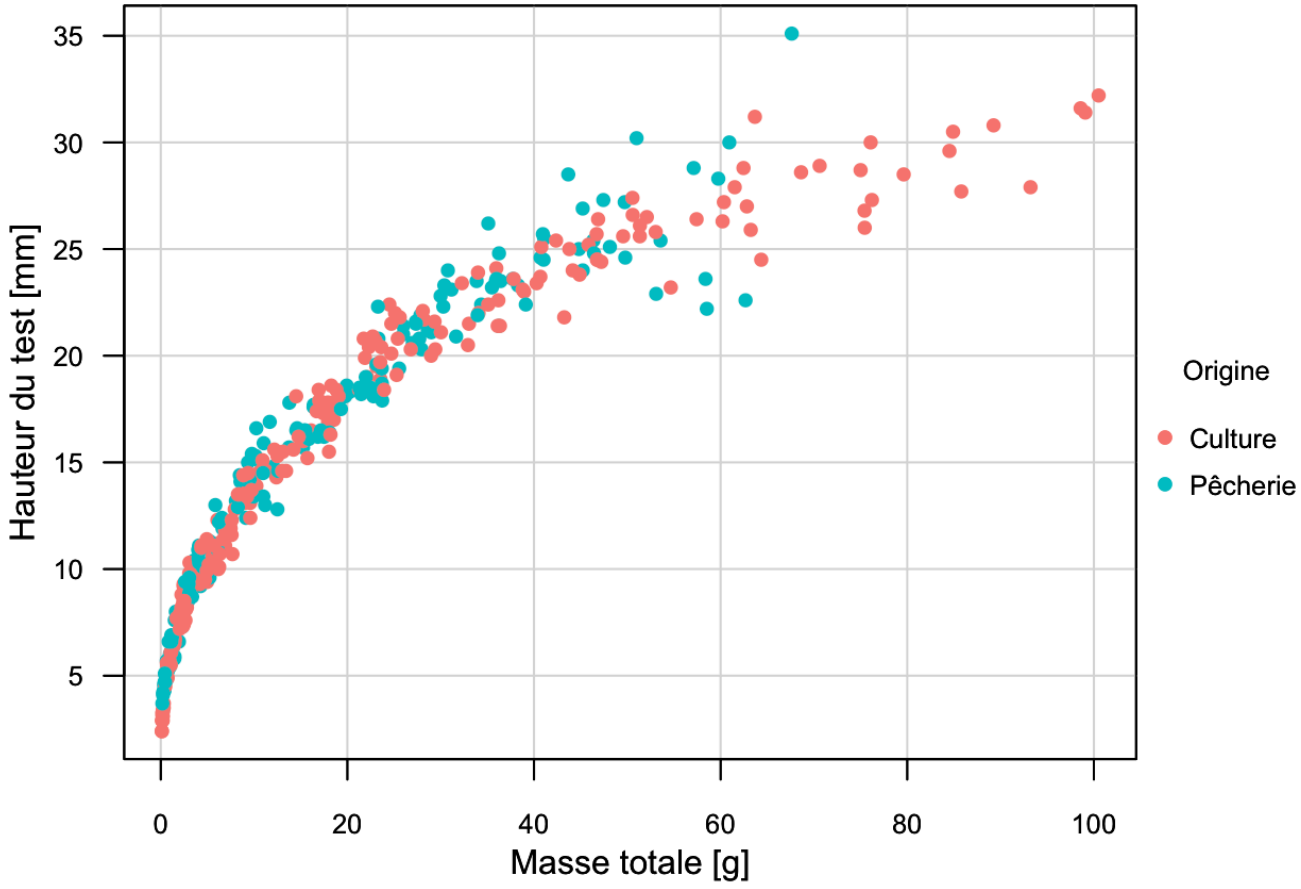


Figure 3.32: Graphique exemple réalisé avec le moteur graphique de base et la fonction `chart()`.

> Vous ne le voyez pas dans le bookdown, mais vous le réaliserez si vous utilisez ce genre de code dans vos propres documents Quarto ou R Markdown : le graphique est en réalité généré deux fois : une première fois dans un format propre aux graphiques R de base, et ensuite, il est traduit en une forme compatible avec les autres graphiques {ggplot2} et {lattice} (et au passage, il gagne la grille en traits grisés). Dans le chunck, nous devons spécifier `fig.keep = 2` si nous voulons éviter d'imprimer la première version dans le rapport lorsqu'on utilise `chart$base()` .

Pour l'instant, le seul avantage de `chart()` avec les graphiques de base est qu'il les convertit en une forme combinable avec les autres graphiques dans une figure composite (sinon, ce n'est pas possible). À part cela, il faut fournir à `chart$base()` tout le code nécessaire pour tracer et personnaliser le graphique. Comme on peut le voir sur cet exemple, cela demande une quantité considérable de code. C'est aussi un autre aspect pénible de ce moteur graphique : il est très flexible, mais l'interface n'est pas optimale. Pour finir, les graphiques de base ont plus de mal avec les facettes, mais ils peuvent quand même générer les versions les plus simples, par exemple à l'aide de la fonction `coplot()` qui accepte une formule très similaire à ce que nous avons employé jusqu'ici, mais avec un rendu différent :

```
coplot(data = urchin, height ~ weight | origin)
```

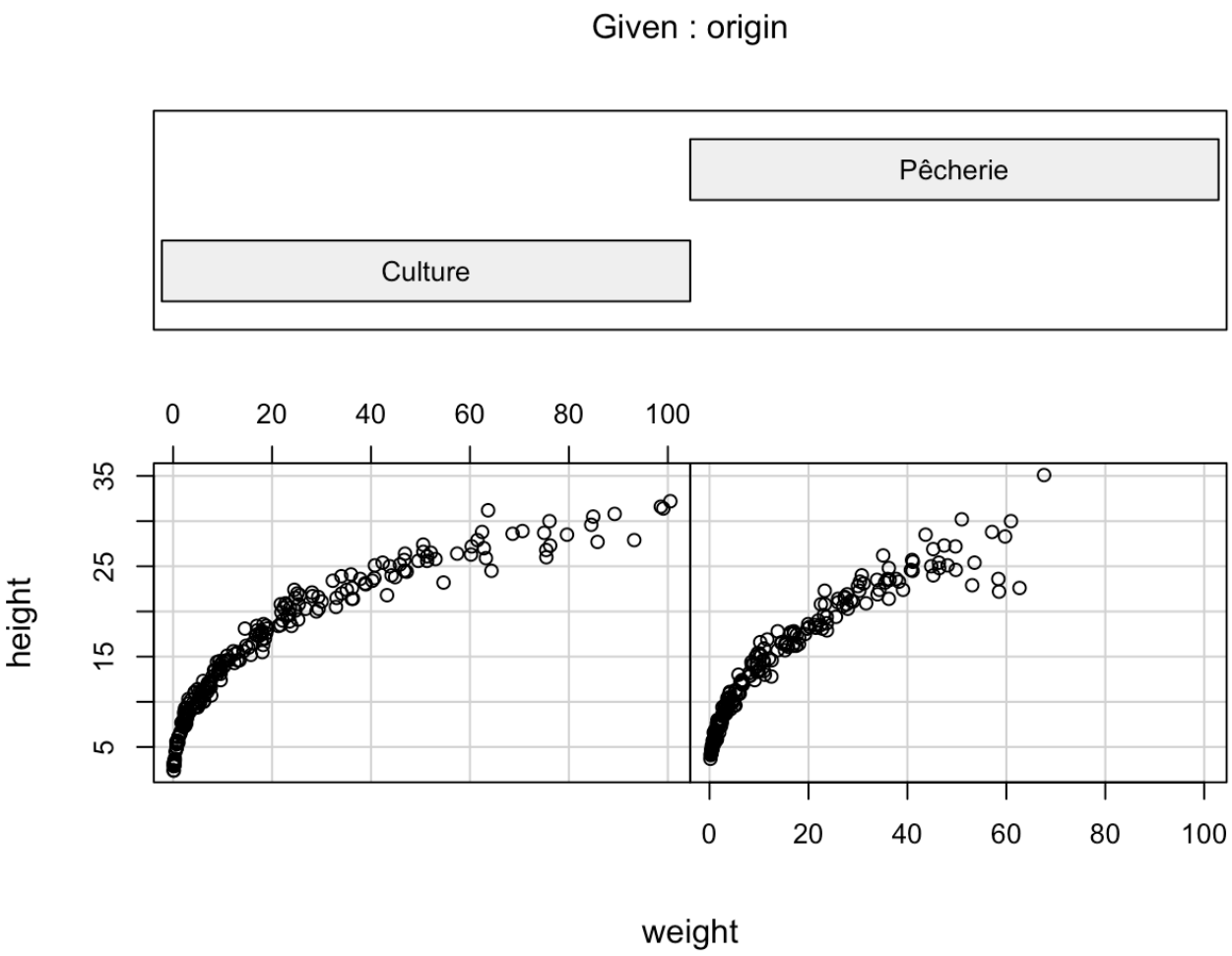


Figure 3.33: Graphique à facettes avec le moteur graphique de base.

À l'issue de cette comparaison, vous pourrez décider du moteur graphique que vous préférerez utiliser. Dans le cadre de ce cours, nous n'utiliserons en tous cas que quasi exclusivement des graphiques {ggplot2} créés à l'aide la fonction `chart()` .

Pour en savoir plus

- Introduction rapide à lattice
- Variantes de graphiques avec lattice
- Comparaison de lattice et ggplot2. Cette page fait aussi référence à un ensemble de graphiques différents générés en {lattice} et en {ggplot2} pour comparaison (en anglais).
- Chapitre d'un libre sur R présentant les graphiques de base

- ggplot2 comparé aux graphiques R de base. Un point de vue différent d'un utilisateur habitué aux graphiques R de base (en anglais).

16. Notez que plusieurs thèmes existent dans **ggplot2**. Il est facile d'en changer et de les personnaliser… mais c'est toujours appréciable d'avoir un rendu impeccable dès le premier essai.↩

17. Étant donné l'abondante littérature écrite sur {ggplot2}, il est utile de pouvoir convertir des exemples {ggplot2} en graphiques `chart()`, si vous êtes convaincu par cette nouvelle interface.↩

3.6 Travail collaboratif II

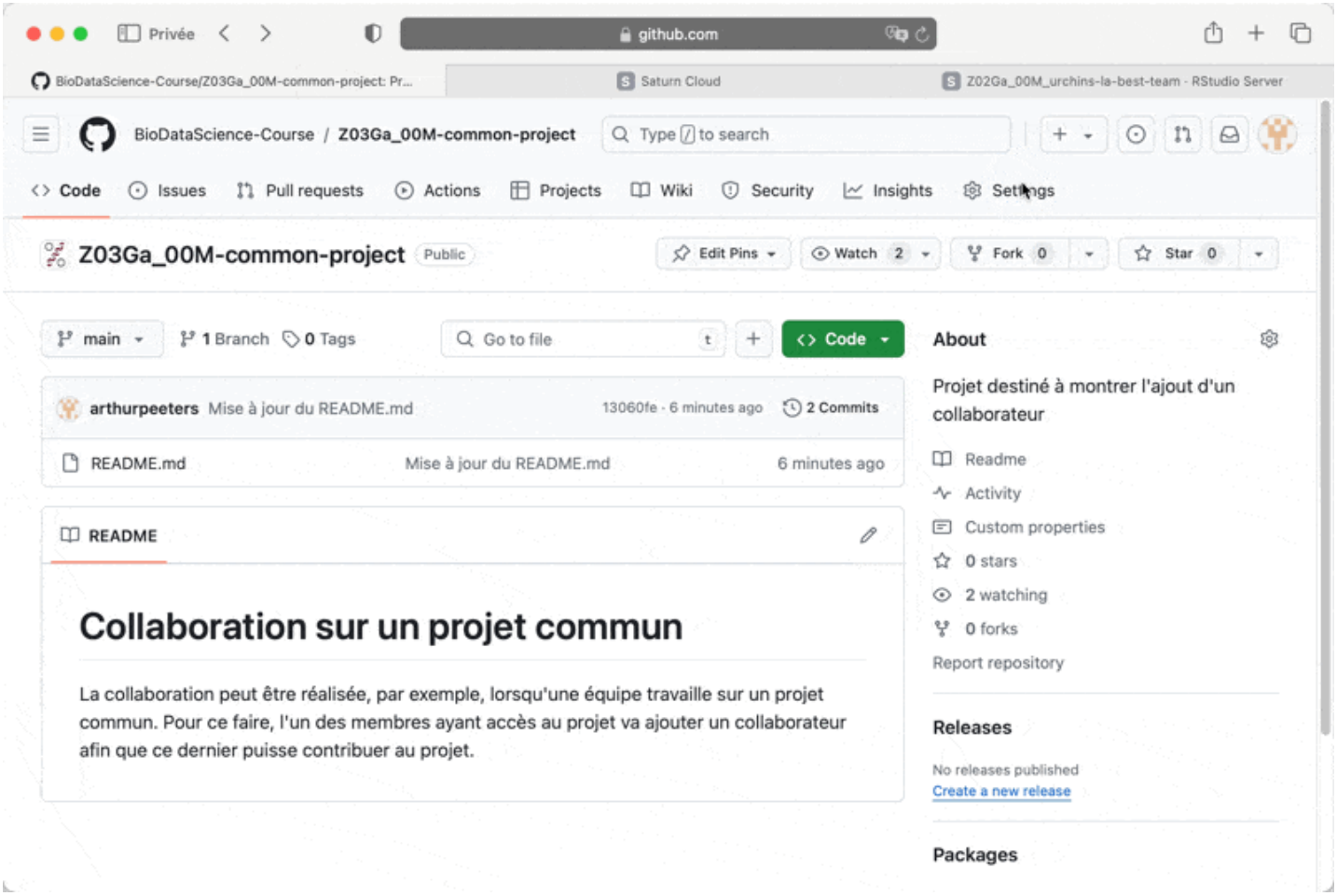
GitHub est une plateforme collaborative. C'est dans ce sens que GitHub a été développé. L'un de ses objectifs est de fournir tous les outils pour permettre le travail en collaboration GitHub comprend plus de 100 millions de dépôts. Vous vous doutez bien que vous n'avez pas la possibilité de cloner un dépôt, de l'éditer et d'y ajouter vos modifications via un push sur un projet de sociétés comme Meta, Microsoft ou encore Google. Il en est de même pour une grande partie des dépôts hébergés sur notre organisation BioDataScience-Course.

Deux possibilités s'offrent principalement à vous. Première option, vous intégrez une équipe pour travailler sur un projet commun. Il faut dans ce cas ajouter l'utilisateur au projet. Néanmoins, être ajouté à un projet ne signifie pas que l'on peut faire ce que l'on veut. Il y a cinq rôles allant de l'accès en lecture, ce qui signifie que l'on peut voir la progression d'un projet et interagir dans les issues, en passant par le droit en écriture, et pour finir, le niveau le plus élevé est l'administrateur de projet qui peut aller jusqu'à supprimer le projet de GitHub. Par exemple, vous n'avez pas le droit administrateur sur vos dépôts d'exercice. Cela nous assure que, par mégarde, vous ne supprimiez vos projets.

La seconde possibilité est de contribuer à un projet en partant d'une copie de ce dernier. Pour dupliquer un dépôt existant sur GitHub, on va réaliser un **"fork"**. Ce fork peut soit être réalisé directement dans votre compte, soit au sein d'une organisation à laquelle vous avez accès. Vous pouvez aussi renommer le projet si vous le souhaitez. Vous avez tous les droits sur votre copie. Par exemple, lorsque vous réalisez un exercice N3 ou N4, GitHub Classroom réalise un fork pour vous. Il part d'un dépôt de référence sur lequel vous n'avez pas le droit en écriture et réalise une copie hébergée dans l'organisation BioDataScience-Course.

3.6.1 Projet commun

Utilisons le projet Z03Ga_00M_commun_project sur lequel arthurpeeters travaille. Suite à des échanges avec oliviamaes, ils vont collaborer dans un projet commun. Pour ce faire, Arthur doit donner accès à Olivia. Pour cela, il se rend dans son dépôt, puis va dans `Settings` puis `Collaborators and teams`. Il clique sur `Add people` et sélectionne ou entre le login GitHub de sa collaboratrice (oliviamaes). Il définit le rôle d'Olivia dans ce dépôt par la même occasion.

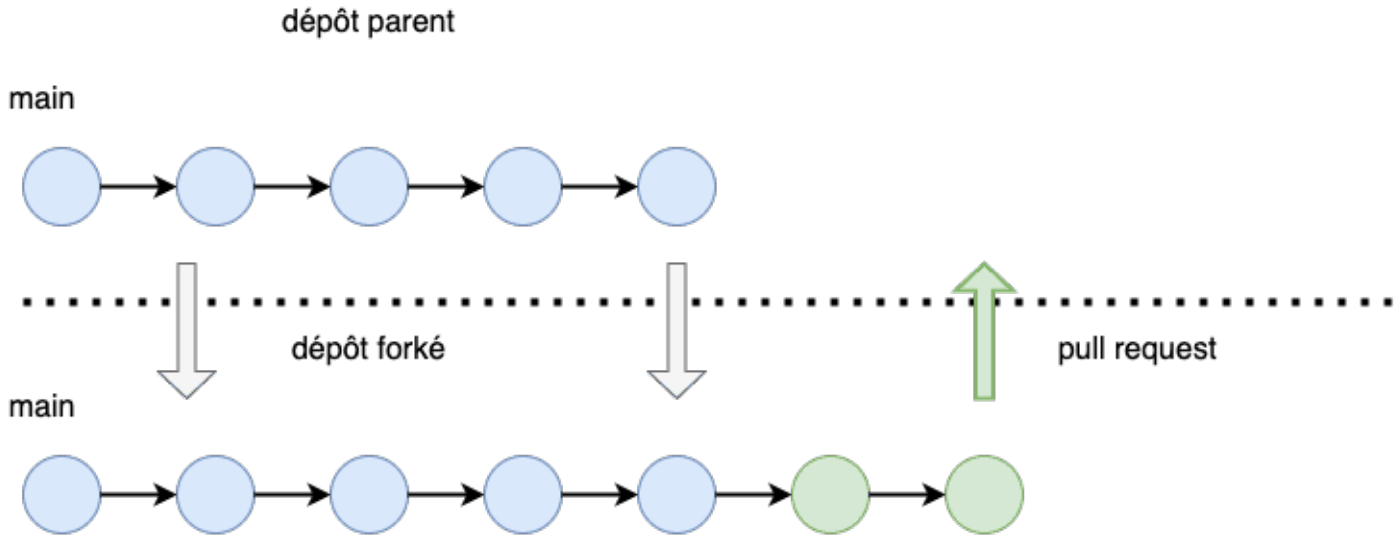


Olivia a à présent la possibilité de réaliser des push sur le dépôt. Olivia peut cloner le projet en local sur son ordinateur pour y travailler en parallèle avec Arthur. Afin de limiter les conflits, il est néanmoins conseillé d'organiser la répartition du travail comme présenté dans le module précédent.

> Si, lors de votre premier push vous avez un message d'erreur indiquant que vous n'avez pas la permission en écriture sur ce dépôt, il y a des chances pour que vous ayez cloné un dépôt qui n'est pas le vôtre, ou pour lequel vous n'êtes pas collaborateur (par exemple, un template de BioDataScience-Course). Faites toujours attention avant de cloner un dépôt et vérifiez vos droits dessus. Sinon, vous pouvez toujours forker un dépôt public appartenant à quelqu'un d'autre, et cloner ensuite votre fork…

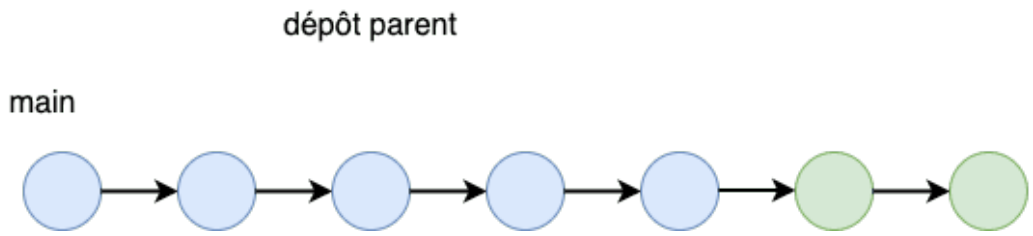
3.6.2 Projet extérieur

Comme mentionné précédemment, il est possible de contribuer à des projets publics même si vous n'avez pas le droit de réaliser des modifications directement dans ces derniers. Pour ce faire, il faut réaliser un **fork** du projet (flèches grises). Un fork est une copie sur GitHub d'un projet également hébergé sur GitHub. Ce nouveau projet peut être renommé comme vous le souhaitez ou garder le même nom. Il s'agit de votre projet. Vous avez par conséquent bien plus de droits sur ce dernier. Vous pouvez, par exemple, réaliser des commits, des pulls et des pushs.



Enfin, il est possible de réaliser une **pull request** (flèche verte), c'est-à-dire de proposer d'intégrer vos modifications au dépôt parent. La fusion de vos ajouts avec le projet parent est réalisée par l'auteur du dépôt parent. Cette fusion est un **merge**. Cette manière de

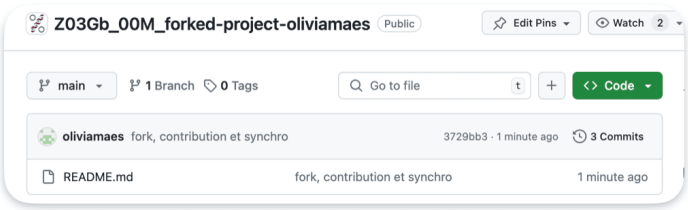
travailler est saine. Les commits sont attribués à chaque auteur. Il est possible de retrouver l'auteur de chaque commit.



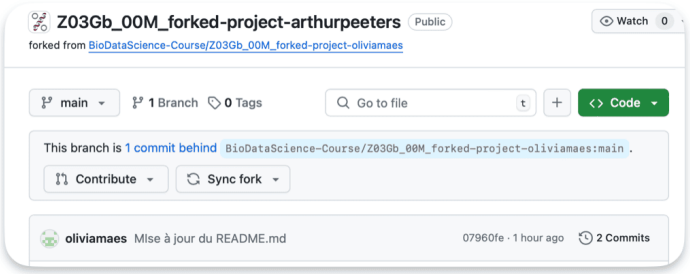
Ce nouveau projet garde un lien avec le projet parent. Cela permet de synchroniser votre dépôt avec le dépôt parent afin de bénéficier de toutes les mises à jour de ce dernier.

Un projet parent et un projet forké sont présentés pour illustrer le processus. Ils sont disponibles ici : Z03Gb_00M_forked-project-oliviamaes et Z03Gb_00M_forked-project-arthurpeeters.

Projet parent



Projet forké



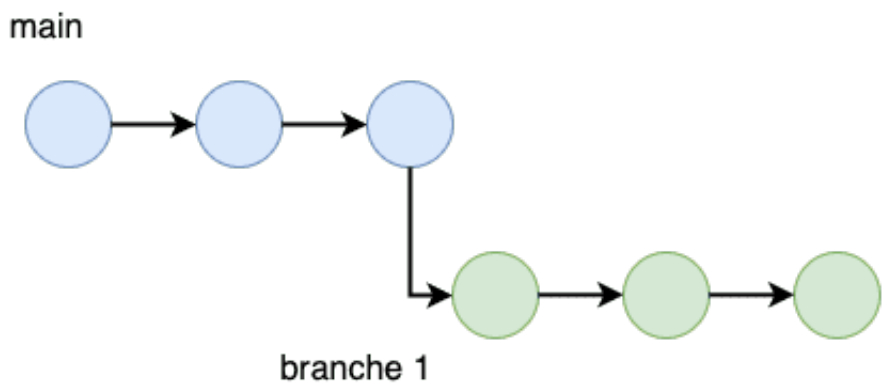
On observe que le projet a été forké lorsque le projet n'avait que 2 commits, alors qu'à présent il en a plus.

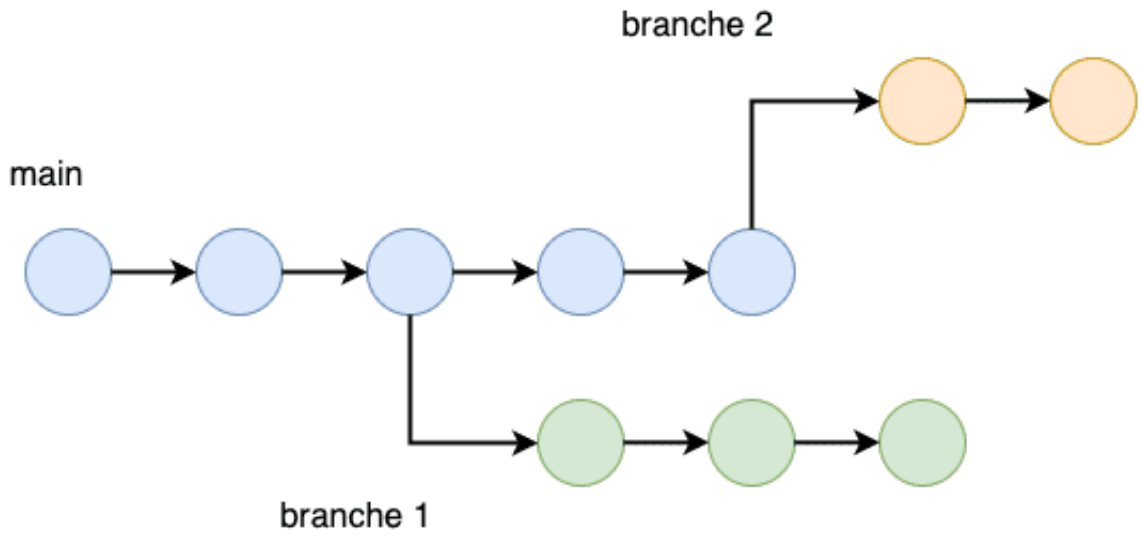
3.6.3 Branches de git

Le gestionnaire de version git peut être envisagé comme une arborescence de branches. Par défaut, votre dépôt à une branche qui se nomme `main` (avant 2020, les branches par défaut se nommaient `master`). Dans le cadre de nos cours, nous travaillerons dans la branche `main` uniquement.

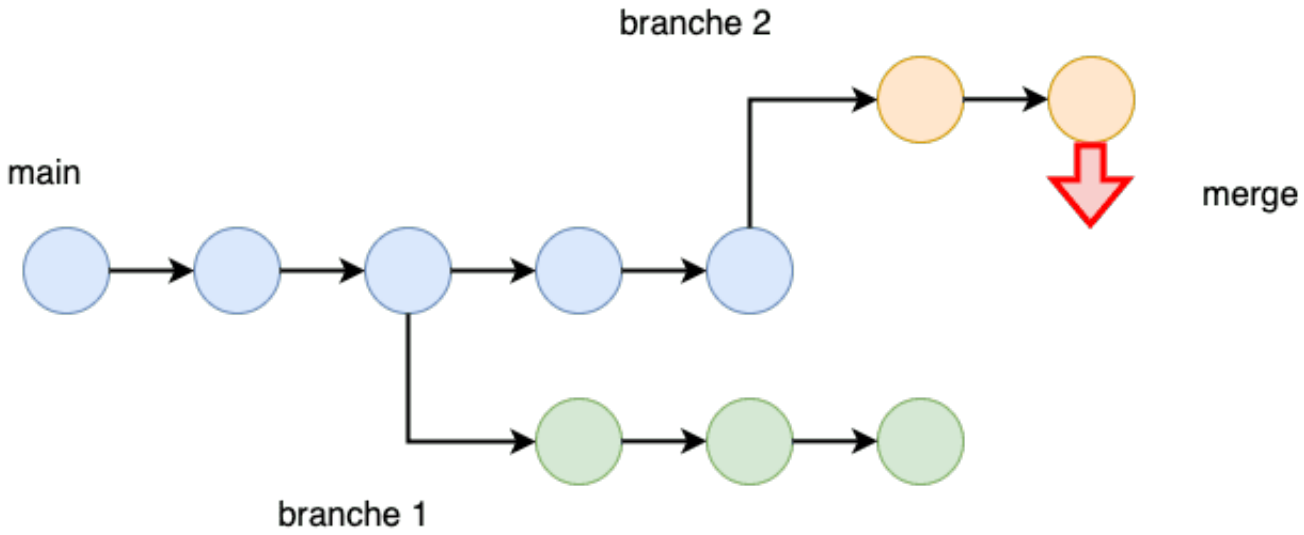
Les branches comme `branche 1` ou `branche 2` sur la figure ci-dessous ont pour objectif de réaliser des essais. Tous les commits en vert ou en orange ne sont pas réalisé dans la branche principale qui est `main` . Cela signifie que la branche main n'est pas affectée.

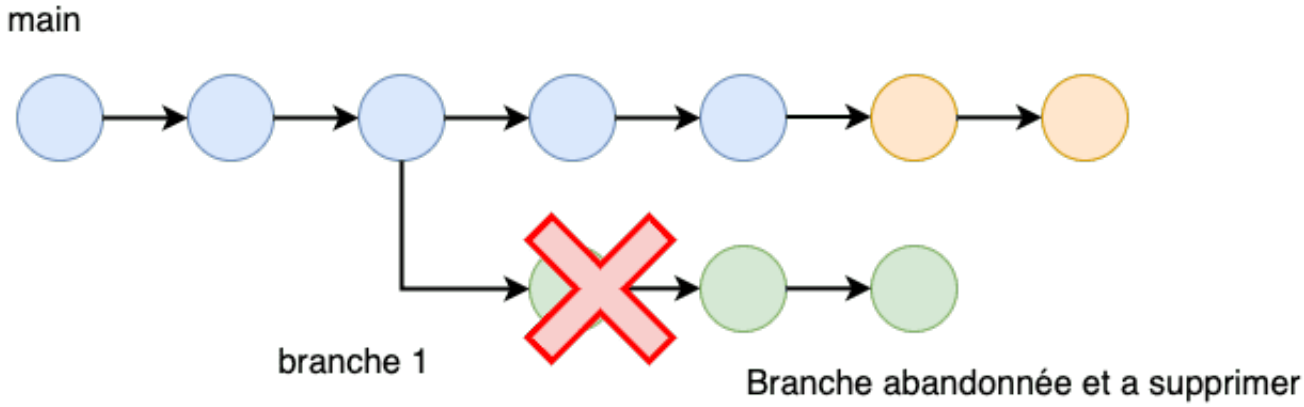
Vous pouvez observer que l'on part d'un commit d'une branche dans notre cas la branche `main`





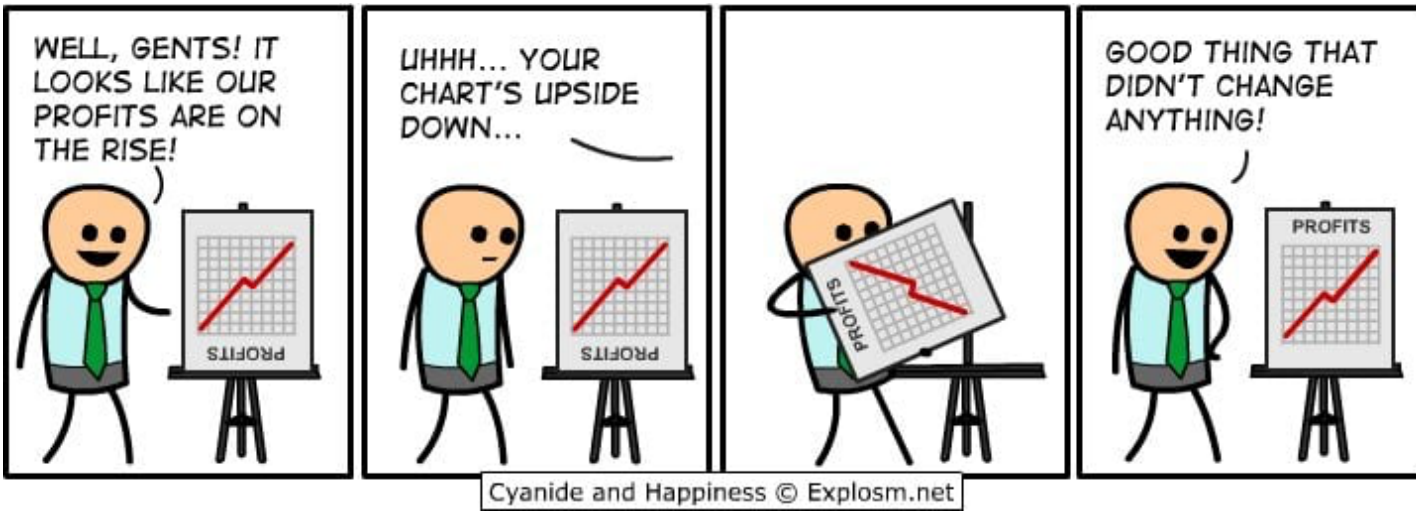
Une branche peut être abandonnée comme la branche 1 ou encore fusionnée avec la branche principale comme la branche 2.


branche 2
main
merge
branche 1


main
branche 1
Branche abandonnée et a supprimer

3.7 Critique graphique

La réalisation de graphiques n'est pas une tâche aisée. En effet, un "mauvais" graphique peut mener à une interprétation erronée. Quelquefois, il s'agit de triche manifeste, mais la plupart du temps, c'est par ignorance. Développer un **esprit critique** est important pour pouvoir démasquer ces diverses situations et ne pas tomber soi-même dans les pièges les plus grossiers. Plus loin dans ce cours, nous reparlerons de l'esprit critique avec toute une série d'erreurs liées à la mauvaise interprétation statistique.



Le développement de votre esprit critique va être long. Pour chaque graphique que vous analyserez, posez-vous au minimum les questions suivantes :

- Les graphiques sont-ils adéquats par rapport à ce qui doit être montré ?
- Les axes sont-ils placés correctement, et sont-ils bien libellés ?
- Le graphique respecte-t-il les conventions ?
- Les unités sont-elles correctes ?
- Les grandeurs observées sont-elles plausibles ? Vous pouvez vous rapporter à des éléments connus et comparer. Par exemple, si l'on vous dit qu'une souris adulte pèse 1g, est-ce plausible ou non ? Faites une recherche sur le Web, ou un raisonnement du genre : une souris est constituée principalement d'eau. Un gramme d'eau occupe un

volume de 1 cm^3. Le volume de la souris adulte est-il supérieur, égal ou inférieur à un cube de 1 cm de côté ?

3.8 Challenge

Tout au long des trois premiers modules, vous avez appris à utiliser R et RStudio, à écrire des documents en R Markdown, à utiliser git et GitHub… Vous avez également appris à réaliser des graphiques clairs qui présentent visuellement l'information contenue dans des jeux de données. Il est maintenant temps de faire un bilan de tout cela sous une forme ludique : un **challenge de rapidité pour la réalisation de graphiques.** Vous vous confronterez entre vous de manière individuelle pour déterminer celui ou celle qui arrivera à reproduire fidèlement le plus de graphiques possible.

À vous de jouer !

> Relevez le challenge **A03Ca_charts**.
>
> **Challenge individuel pour les étudiants inscrits au cours de Science des Données Biologiques I : visualisation à l'UMONS à terminer avant le 2025-11-14 11:30:00.**
>
> **Initiez le projet GitHub Classroom de ce challenge**
>
> *Voyez les explications dans le fichier* `README.md` .

Vous pouvez soumettre vos résultats pendant une période de temps précise via l'application ci-dessous :

3.9 Récapitulatif des exercices

Ce module 3 vous a permis de réaliser et de découvrir de nouveaux graphiques. Vous savez maintenant réaliser des graphiques en barres, des graphiques en camembert et des boites à moustaches. Vous êtes aussi capable de réaliser des graphiques composés. Pour évaluer votre compréhension de cette matière, vous aviez les exercices suivants à réaliser

- A03Ha_position - Graphiques en barres
- A03Hb_bars - Quel graphique est-ce ?
- A03La_barplot - Graphiques en barres et camembert
- A03Hc_median - Médiane
- A03Hd_iqr - Boite à moustaches et valeurs extrêmes
- A03Lb_boxplot - Boites à moustaches
- A03He_facet - Paires de graphiques
- A03Lc_comp_fig - Graphiques composites
- A03Ia_graphe_avance - Graphiques avancés (barres, boites à moustaches et composites)
- A02Ga_analysis - Analyse de données (partie II)
- A03Ca_charts - Challenge graphiques

Progression

Module 4 Traitement des données I

Objectifs

- Pouvoir importer des données depuis différents formats et différentes sources en utilisant la fonction `read()` de `SciViews::R` .
- Appréhender les types de variables et l'importance de les encoder convenablement dans R.
- Être capable de convertir des variables d'un type à l'autre, y compris par l'utilisation du découpage en classes pour passer de variable quantitative à qualitative.
- Comprendre comment remanier des variables, filtrer un tableau et le résumer pour en extraire l'information importante.

Prérequis

Le contenu du module 1 doit être parfaitement maîtrisé. Il est également souhaitable, mais pas indispensable, de comprendre comment réaliser des graphiques dans R pour pouvoir comprendre le contenu de ce module.

4.1 Importation des données

Il est possible d'encoder de très petits jeux de données directement dans R. La fonction `dtx_rows()` de `SciViews::R` [18] permet de le faire facilement. Notez que les noms des colonnes du tableau sont à rentrer sous forme de **formules** (`~var`), que chaque entrée est séparée par une virgule, et que les chaînes de caractères sont entourées de guillemets Les espaces sont optionnels et peuvent être utilisés pour aligner les données afin que le tout soit plus lisible. Des commentaires peuvent être utilisés éventuellement en fin de ligne (un "hashtag" aussi appelé dièse `#` suivi du commentaire).

```
SciViews::R
small_dataset <- dtx_rows(
  ~treatment, ~dose, ~response,
   "control",   0.5,     18.35,
   "control",   1.0,     26.43, # This value needs to be double-checked
   "control",   2.0,     51.08,
   "test"   ,   0.5,     10.29,
   "test"   ,   1.0,     19.92,
   "test"   ,   2.0,     41.06)
# Print the table
small_dataset
```

```
# # A data.trame: [6 × 3]
#   treatment  dose response
#   <chr>     <dbl>    <dbl>
# 1 control     0.5     18.4
# 2 control     1       26.4
# 3 control     2       51.1
# 4 test        0.5     10.3
# 5 test        1       19.9
# 6 test        2       41.1
```

Dans la plupart des cas, vous utiliserez ou collecterez des données stockées dans des formats divers : feuilles Excel, fichiers CSV (“comma-separated-values”, un format standard d’encodage d’un tableau de données sous forme textuelle), formats spécifiques à divers logiciels statistiques comme SAS, Stata, Systat… Ces données peuvent être sur un disque local ou disponibles depuis un lien URL sur le net[19]. De nombreuses fonctions existent dans R pour importer toutes ces données. La fonction `read()` du package {data.io}[20] est l’une des plus simples et conviviales d’entre-elles. Vous l’avez déjà utilisée, mais reprenons un exemple pour en discuter les détails.

```
biometry <- read("biometry", package = "BioDataScience", lang = "fr")
tabularise$headtail(biometry)
```

Genre	Date de naissance	Masse [kg]	Taille [cm]	Tour de poignet [mm]	Année de la mesure	Age [année]
M	1995-03-11	69.0	182	15.0	2 013	18
M	1998-04-03	74.0	190	16.0	2 013	15
M	1967-04-04	83.0	185	17.5	2 013	46
M	1994-02-10	60.0	175	15.0	2 013	19
W	1990-12-02	48.0	167	14.0	2 013	23
…		…	…	…	…	…
M	1998-08-12	80.0	177	18.0	2 017	19
W	1999-08-12	67.3	173	15.0	2 017	18

M	1968-08-18	84.0	187	18.0	2 017	49
W	1971-08-18	64.9	166	15.0	2 017	46
M	1999-08-12	68.7	182	15.0	2 017	18

First and last 5 rows of a total of 395

Le jeu de données `biometry` est disponible dans le package R {BioDataScience} indiqué dans l'argument `package =` . Dans ce cas, il ne faut pas spécifier de chemin d'accès au fichier : R sait où le trouver tout seul. Il est également indiqué ici que la langue souhaitée est le français avec l'argument `lang = "fr"` . Le résultat de l'importation est assigné à la variable `biometry` (mais elle pourrait tout aussi bien porter un autre nom). Pour finir, nous visualisons le résultat de l'importation, ici en faisant un tableau des premières et dernières lignes avec `tabularise$headtail()` .

> **Visualisez toujours votre tableau de données juste après l'importation pour vérifier que l'opération s'est bien déroulée.** Vérifiez que les différentes colonnes ont été importées au bon format. Vous pouvez ouvrir l'onglet **Environnement** de RStudio et cliquer sur la petite icône bleue devant le nom de l'objet pour voir de quoi il est composé. *En particulier*,
>
> - Les données numériques sont-elle bien comprises par R comme des nombres (`<num>` , `<dbl>` ou `<int>`) ?
> - Les variables qualitatives ou semi-quantitatives sont parfois importées comme chaînes de caractères (`<chr>`) et doivent éventuellement être converties en variables de type **facteur** à l'aide de `as.factor()` ou **facteur ordonné** avec `as.ordered()` , voir plus loin.
>
> L'impression du tableau de données est une façon de voir cela, mais il y en a bien d'autres : essayez `View(biometry)` , `glimpse(biometry)` , `str(biometry)` , `skimr::skim(biometry)` .

Avant d'importer vos données dans R, vous devez vous poser les deux questions suivantes :

- Où ces données sont-elles stockées ?

Vous venez d'importer des données depuis un package R. Vous pouvez également les lire depuis un fichier sur le disque ou en utilisant une URL pour les récupérer depuis le Web. Tous ces cas sont gérés par `read()` qui unifie donc de manière simple vos accès aux données.

- Quel est le format de vos données ?

Souvent ce format est renseigné par l'**extension** du fichier. Par exemple **.xlsx** pour un Microsoft Excel ou **.csv** pour du "comma-separated-value". Attention ! L'extension du fichier est cachée sous Windows, et parfois sous macOS. Visualisez vos fichiers dans l'onglet **Files** de RStudio pour voir leurs noms complets avec les extensions. Pour l'instant, `read()` supporte 44 formats de fichiers différents, mais cette liste est amenée à s'agrandir à l'avenir. Pour découvrir les formats supportés, et les fonctions d'importation spécifiques appelées à chaque fois, utilisez :

```
getOption("read_write")
```

```
# # A tibble: 44 × 5
#    type     read_fun          read_header         write_fun              com
#    <chr>    <chr>             <chr>               <chr>                  <ch
#  1 csv      data.table::fread data.io::hread_text data.table::fwrite     com
#  2 csv_alt  readr::read_csv   data.io::hread_text readr::write_csv       com
#  3 csv2     data.table::fread data.io::hread_text data.table::fwrite     sem
#  4 csv2_alt readr::read_csv2  data.io::hread_text <NA>                   sem
#  5 xlcsv    readr::read_csv   data.io::hread_text readr::write_excel_csv wri
#  6 tsv      data.table::fread data.io::hread_text data.table::fwrite     tab
#  7 tsv_alt  readr::read_tsv   data.io::hread_text readr::write_tsv       tab
#  8 fwf      readr::read_fwf   data.io::hread_text <NA>                   fix
#  9 log      readr::read_log   <NA>                <NA>                   sta
# 10 rds      readr::read_rds   <NA>                readr::write_rds       R d
# # ℹ 34 more rows
```

Par la suite, vous allez apprendre à importer vos données depuis différentes sources.

4.1.1 Données sur le disque

Lorsque l'extension du fichier reflète le format des données, il vous suffit juste d'indiquer le chemin d'accès au fichier dans `read()`, en utilisant préférentiellement un chemin relatif. La plupart du temps, cela suffit pour importer correctement les données.

> **Très important :** le chemin d'accès à votre fichier peut s'écrire de manière absolue (chemin partant de la racine d'un disque sous Windows ou de la hiérarchie de dossiers sous macOS ou Linux) ou bien de manière relative (par rapport au dossier courant, la plupart du temps, le dossier du projet dans RStudio). **Vous devez autant que possible employer des chemins relatifs** pour que votre projet soit **portable**. Si vous avez du mal à déterminer le chemin relatif par rapport à vos données, le snippet `filerelchoose` vous sera très utile :
>
> 1. Assurez-vous que le chemin actif dans l'onglet **Console** est (A) le dossier racine de votre projet RStudio, ou (B) si vous travaillez dans un fichier Quarto ou R Markdown, le même que le répertoire contenant le fichier édité. Pour cela, utilisez l'entrée de menu RStudio `Session -> Définir le répertoire de travail -> Vers le répertoire du projet` (situation A) ou `Session -> Définir le répertoire de travail -> Vers l'emplacement du fichier source` (situation B).
> 2. Utilisez le snippet `filerelchoose` que vous activez dans une zone de code R (dans un script R, ou à l'intérieur d'un chunk dans un document Quarto/R Markdown). Entrez `filerel` pus appuyez sur la touche `Tabulation`. Une boite de dialogue de sélection de fichier apparaît. Sélectionnez le fichier qui vous intéresse et … `filerel` est **remplacé par le chemin relatif vers votre fichier** dans votre instruction R.

Les explications détaillées concernant l'organisation de vos projets dans RStudio pour qu'ils soient portables, ainsi que la gestion des chemins d'accès aux fichiers et les chemins relatifs sont détaillés dans l'annexe B, à la section B.1.1. **C'est le moment de**

vérifier que vous avez bien compris et assimilé son contenu.

À vous de jouer !

Déplacez les textes aux emplacements correspondants pour compléter les fonctions ci-dessous. Vous êtes dans le projet RStudio.

../data/mydata.csv

data/mydata.csv

```
/home
  /jovyan
    /workspace
      /project_test        # Répertoire de base du projet
        /docs          # Dossier pour les analyses
           report.Rmd     # Un rapport au format Rmd
        /data               # Dossier avec les données
            mydata.csv   # Fichier du jeu de données
        /R                  # Dossier avec les scripts R
           script.R        # Un script R
```

Si je suis dans un chunk de report.Rmd, je dois utiliser l'instruction ci-dessous pour importer mes données

```
my_data <- read("                              ")
```

Si je suis dans le fichier script.R, je dois utiliser l'instruction ci-dessous pour importer mes données

```
my_data <- read("                              ")
```

Vérifier

Pièges et astuces

- Si l'extension est incorrecte, vous pouvez forcer un format de fichier particulier à l'importation en l'indiquant dans l'appel à `read()` comme `read$<ext>()` . Par exemple, pour forcer l'importation d'un fichier de type "comma-separated-values" pour un fichier qui se nommerait `my_data.txt` et qui se situe dans le répertoire courant, vous écrirez `read$csv("my_data.txt")` .
- Si les données ne sont pas importées correctement, cela signifie que les arguments d'importation par défaut ne sont pas adaptés. Les arguments à spécifier sont différents d'un format à l'autre. Voyez d'abord la fonction appelée en interne par `read()` dans le tableau obtenu grâce à `getOption("read_write")` . Par exemple, pour un fichier `xlsx` , il s'agit de la fonction `readxl::read_excel()` qui est utilisée. Ensuite, voyez l'aide de cette dernière fonction pour en découvrir les différents arguments (`?readxl::read_excel`). Là, vous pourrez découvrir l'argument `sheet =` qui indique la feuille à importer depuis le fichier (première feuille par défaut), ou `range =` qui indique la plage de données dans la feuille à utiliser (par défaut, depuis la cellule A1 en haut à gauche jusqu'à la fin du tableau). Donc, si votre fichier `my_data.xlsx` contient les feuilles `sheet1` , `sheet2` et `sheet3` , et que les données qui vous intéressent sont dans la plage `C5:E34` de `sheet2` , vous pourrez écrire : `read("my_data.xlsx", sheet = "sheet2", range = "C5:E34")` .

4.1.2 Données depuis Internet

Il existe différents logiciels qui permettent d'éditer des jeux de données en ligne et de les partager sur le Net. Google Sheets est l'un d'entre eux, tout comme Excel Online. Des stockages spécifiques pour les données scientifiques existent aussi comme Figshare ou Zenodo. Ces sites permettent de partager facilement des jeux de données sur Internet.

> La science est de plus en plus ouverte, et les pratiques de **données ouvertes** (*Open Data*) de plus en plus fréquentes et même imposées par des programmes de recherche comme les programmes européens ou le FNRS en Belgique. Vous serez donc certainement amenés à accéder à des données depuis des dépôts spécialisés sur Internet.

> Concentrez-vous sur les outils spécifiques à la gestion de ce type de données. il s'agit, en effet, d'une compétence clé qu'un bon scientifique des données se doit de maîtriser parfaitement. En recherchant à chaque fois la meilleure façon d'accéder à des données sur Internet, vous développerez cette compétence progressivement par la pratique… et **vous pourrez faire valoir un atout encore rare mais apprécié lors d'un entretien d'embauche plus tard.**

Voici un exemple de feuille de données Google Sheets : https://docs.google.com/spreadsheets/d/1iEuGrMk4IcCkq7gMNzy04DkSaPeWH35Psb0E56KEQMw. Il est possible d'importer ce genre de données **directement** dans R, mais il faut d'abord déterminer l'URL à utiliser pour obtenir les données dans un format reconnu. Dans le cas de Google Sheets, il suffit d'indiquer que l'on souhaite exporter les données au format CSV en rajoutant `/export?format=csv` à la fin de l'URL.

Cette URL est très longue. Elle est peu pratique et par ailleurs, elle a toujours la même structure : `"https://docs.google.com/spreadsheets/d/{id}/export?format=csv"` avec `{id}` qui est l'identifiant unique de la feuille Google Sheets (ici `1iEuGrMk4IcCkq7gMNzy04DkSaPeWH35Psb0E56KEQMw`). Vous pouvez indiquer explicitement ceci dans votre code et profiter des capacités de remplacement de texte dans des chaînes de caractères de la fonction `glue::glue()` pour effectuer un travail impeccable.

```
googlesheets_as_csv <- "https://docs.google.com/spreadsheets/d/{id}/export?for
coral_data_id <- "1iEuGrMk4IcCkq7gMNzy04DkSaPeWH35Psb0E56KEQMw"
(coral_url <- glue::glue(googlesheets_as_csv, id = coral_data_id))
```

```
# https://docs.google.com/spreadsheets/d/1iEuGrMk4IcCkq7gMNzy04DkSaPeWH35Psb0E
```

Vous n'aurez alors plus qu'à lire les données depuis cette URL. N'oubliez pas non plus de spécifier à `read()` que les données sont à lire au format CSV en utilisant `read$csv()` :

```
coral <- read$csv(coral_url)
tabularise$headtail(coral)
```

localisation	species	id	salinity	temperature	date	time	gain	gain_std
A0	s.hystrix	1	34.7	24.5	2018-04-24 09:10:00	7	0.08	0.05897017
A0	s.hystrix	2	34.7	24.5	2018-04-24 09:10:00	7	0.06	0.03669477
A0	s.hystrix	3	34.7	24.5	2018-04-24 09:10:00	7	0.03	0.01904207
A0	s.hystrix	4	34.7	24.5	2018-04-24 09:10:00	7	0.11	0.04722930
A0	s.hystrix	5	34.7	24.5	2018-04-24 09:10:00	7	0.12	0.06810833
...	...	...	...	...		...	...	...
B0	s.hystrix	96	35.0	25.2	2018-03-19 13:03:00	7	0.07	0.05942559
B0	s.hystrix	97	35.0	25.2	2018-03-19 13:03:00	7	0.05	0.04447699
B0	s.hystrix	98	35.0	25.2	2018-03-19 13:03:00	7	0.09	0.05969835
B0	s.hystrix	99	35.0	25.2	2018-03-19 13:03:00	7	0.06	0.05176493
B0	s.hystrix	100	35.0	25.2	2018-03-19 13:03:00	7	0.10	0.07131212

First and last 5 rows of a total of 98

Lorsque vous travaillez sur des données issues d'une source externe, et donc susceptibles d'être modifiées ou même pire, de disparaître. Il est préférable d'enregistrer une **copie locale** de ces données dans votre projet (dans le sous-dossier `data` de préférence). Si vous voulez enregistrer une copie locale du fichier **non modifié** téléchargé depuis Internet, vous l'indiquez dans l'argument `cache_file =` de `read()` (pensez à utiliser autant que possible un chemin relatif). Ainsi, le code suivant ne téléchargera le fichier depuis Internet qu'une seule fois et en sauvera automatiquement une copie dans `data/coral.csv`. Ensuite, les exécutions suivantes du même code liront les données depuis la version du jeu de données sauvé en local dans `data/coral.csv`.

```
(coral <- read$csv(coral_url, cache_file = "data/coral.csv"))
```

Faites bien attention que le dossier dans lequel vous souhaitez stocker une copie de vos données doit être présent sur votre disque. Dans le cas contraire, un message d'erreur sera généré et vous ne pourrez pas importer les données. Pour vous assurer que le sous-dossier `data` existe bien, vous pouvez faire `dir.create("data")` avant d'appeler `read()`.

Vous pouvez aussi vous passer de cette fonctionnalité, mais importer vos données dans un script à part qui remanie aussi ces données à la suite. Vous pouvez alors terminer ce script en écrivant les données remaniées dans un dossier local. Si vous travaillez exclusivement avec R, l'un des meilleurs formats est RDS, un format natif qui conservera toutes les caractéristiques de votre objet, y compris sa classe, et d'éventuels attributs[21]. Par défaut, les données seront stockées non compressées, mais vous pourrez aussi décider de compresser avec les algorithmes `"gz"` (plus rapide et répandu), `"bz2"` (intermédiaire), ou `"xz"` (le plus efficace en taux de compression, mais aussi le plus lent et gourmand en ressources CPU). Par exemple, pour enregistrer les données avec compression `"gz"`, dans le sous-dossier `data` de votre projet, vous écrirez (le dossier `data` doit être préexistant) :

```
write$rds(coral, file = "data/coral.rds", compress = "gz")
```

Cette instruction est valable si elle est exécutée depuis un script R dans la fenêtre **Console** où le répertoire actif est le dossier racine du projet. Pour une instruction dans un document Quarto ou R Markdown présent dans un sous-dossier du projet, vous écrirez `file = "../data/coral.rds"`. Dans les deux cas, vous pouvez aussi utiliser `file = here:here("data", "coral.rds")` qui fonctionnera dans tous les cas de figure (conseillé, donc). Cette forme est également utilisable pour `cache_file =` dans `read()`. Ensuite, vous pourrez simplement charger ces données plus loin depuis la version locale dans votre Quarto ou R Markdown comme ceci (cas d'un document Quarto dans un sous-dossier du projet, par exemple `docs`) :

```
coral <- read("../data/coral.rds")
```

Attention, ne supprimez jamais l'instruction permettant de retrouver vos données sur Internet sous prétexte que vous avez maintenant une copie locale à disposition. C'est le lien, le fil conducteur vers les données originales. Vous pouvez soit mettre l'instruction en commentaire en ajoutant un `#` devant, soit soustraire le chunk de l'évaluation en indiquant `eval=FALSE` dans son entête. Faites-en de même avec l'instruction `write()`. Ainsi, le traitement de vos données commencera à l'instruction `read()` et vous partirez de la copie locale. Si jamais vous voulez effectuer une mise à jour depuis la source initiale, il sera toujours possible de dé-commenter les instructions, ou de passer le chunk à `eval=TRUE` temporairement (ou encore plus simplement, forcez l'exécution du chunk dans l'éditeur en cliquant sur la petite flèche verte en haut à gauche de ce chunk).

Pièges et astuces

- Comme il s'agit seulement d'une *copie* des données originelles, vous pouvez choisir de ne pas inclure le fichier `.rds` dans le système de gestion de version de Git. Il suffit d'ajouter une entrée `.rds` dans le fichier `.gitignore` à la racine de votre dépôt, et tous les fichiers avec cette extension seront ignorés. Notez toutefois que, si vous partagez votre projet sur GitHub, **les données locales n'y apparaîtront pas non plus.** D'une part, cela décharge le système de gestion de version, et d'autre part, les gros fichiers de données n'ont pas vraiment leur place dans GitHub. Cependant, soyez conscient que quelqu'un qui réalise un clone ou un fork de votre dépôt devra

d'abord réimporter lui aussi localement les données avant de pouvoir travailler, ce qui implique de bien comprendre le mécanisme que vous avez mis en place. Documentez-le correctement, avec une note explicite dans le fichier `README.md`, par exemple.

- Les données originales ne sont peut-être pas présentées de la façon qui vous convient. Cela peut nécessiter un travail important de **préparation du tableau de données.** Au fur et à mesure que le ou les chunks d'importation/préparation des données augmentent en taille, ils deviennent de plus en plus gênants dans un document consacré à l'**analyse** de ces données. Si c'est le cas, vous avez deux options possibles :

  1. Séparer votre Quarto/R Markdown en deux. Un premier document dédié à l'importation/préparation des données et un second qui se concentre sur l'analyse. Une bonne pratique consiste à numéroter les fichiers en tête pour qu'ils apparaissent par ordre logique lorsqu'ils sont listés par ordre alphabétique (`01import.qmd`, `02analysis.qmd`).
  2. Effectuer le travail d'importation/préparation du tableau de données dans un script R. Dans le R Markdown, vous pouvez ajouter l'instruction (commentée ou placée dans un chunk `eval=FALSE`) pour "sourcer" ce script R afin de réimporter/retraiter vos données (ici, cas d'un fichier Quarto qui se situerait dans un sous-dossier docs) :

```
#source("../R/data-import.R")
```

Si le travail de préparation des données est lourd (et donc, prends beaucoup de temps) il peut être avantageux d'enregistrer localement la version **nettoyée** de vos données plutôt que la version originale. Mais alors, indiquez-le explicitement.

> Faites toujours la distinction entre **données brutes** et **données nettoyées**. Ne les mélangez jamais et documentez toujours de manière reproductible le processus qui mène des unes aux autres ! C'est tout aussi important que de garder un lien vers la source originale des données dans votre code et d'utiliser toujours des chemins relatifs vers vos fichiers pour une analyse portable et reproductible.

Nous vous conseillons de placer la version brute des données dans le sous-dossier `data/raw` et la version nettoyée dans le sous-dossier `data` de votre projet RStudio. Ainsi, la structure des dossiers est elle-même très suggestive.

À vous de jouer !

H5P

Quel est le rôle du fichier .gitignore dans un projet RStudio (ou un dépôt git en général) ?

- [ ] Il permet de sélectionner les fichier que l'on souhaite intégrer au système de gestion de version git.
- [ ] Il indique les fichiers et les dossiers que l'on ne souhaite pas intégrer au système de gestion de version git.
- [ ] Ce fichier peut être ignoré. Il est là juste pour nous indiquer que nous sommes dans un dossier géré par git.

4.1.3 Données depuis un package

Les packages R comme {data.io}, {chart} ou encore {svFlow}, fournissent une série de fonctions supplémentaires. Certains d'entre eux proposent également des jeux de données. Ici aussi, `read()` permet de les récupérer, même si c'est la fonction `data()` qui est souvent utilisée à cet effet dans R de base. Comparons `read()` et `data()` dans le cas des données issues de packages R. Avec `data()`, vous n'assignez pas le jeu de données à un nom. Ce nom vous est **imposé** comme le nom initial du jeu de données :

```
data("urchin_bio", package = "data.io")
```

Le jeu de données `urchin_bio` n'est pas véritablement chargé dans l'environnement utilisateur avec `data()`. Seulement une "promesse" de chargement (`Promise`) est enregistrée. Voyez dans l'onglet **Environnement** ce qui apparaît. Ce n'est qu'à la première

utilisation du jeu de données que le tableau est véritablement chargé. Par exemple :

```
head(urchin_bio)
```

```
#    origin diameter1 diameter2 height buoyant_weight weight solid_parts
# 1 Fishery       9.9      10.2    5.0             NA 0.5215      0.4777
# 2 Fishery      10.5      10.6    5.7             NA 0.6418      0.5891
# 3 Fishery      10.8      10.8    5.2             NA 0.7336      0.6770
# 4 Fishery       9.6       9.3    4.6             NA 0.3697      0.3438
# 5 Fishery      10.4      10.7    4.8             NA 0.6097      0.5587
# 6 Fishery      10.5      11.1    5.0             NA 0.6096      0.5509
#   integuments dry_integuments digestive_tract dry_digestive_tract gonads
# 1      0.3658              NA          0.0525              0.0079      0
# 2      0.4447              NA          0.0482              0.0090      0
# 3      0.5326              NA          0.0758              0.0134      0
# 4      0.2661              NA          0.0442              0.0064      0
# 5      0.4058              NA          0.0743              0.0117      0
# 6      0.4269              NA          0.0492              0.0097      0
#   dry_gonads skeleton lantern   test spines maturity  sex
# 1          0   0.1793  0.0211 0.0587 0.0995        0 <NA>
# 2          0   0.1880  0.0205 0.0622 0.1053        0 <NA>
# 3          0   0.2354  0.0254 0.0836 0.1263        0 <NA>
# 4          0   0.0630  0.0167 0.0180 0.0283        0 <NA>
# 5          0       NA      NA     NA     NA        0 <NA>
# 6          0       NA      NA     NA     NA        0 <NA>
```

Regardez à nouveau dans l'onglet **Environnement**. Ce coup-ci `urchin_bio` apparaît bien dans la section **Data** et l'icône en forme de petit tableau à la droite qui permet de le visualiser est enfin accessible.

La fonction `read()` permet de choisir librement le nom que nous souhaitons donner à notre jeu de données. Si nous voulons l'appeler `urchin` au lieu de `urchin_bio` , pas de problèmes. De plus, il est directement chargé et accessible dans l'onglet **Environnement**

(en effet, si on utilise une instruction qui charge un jeu de données, c'est *très vraisemblablement* parce que l'on souhaite ensuite le manipuler depuis R, non ?)

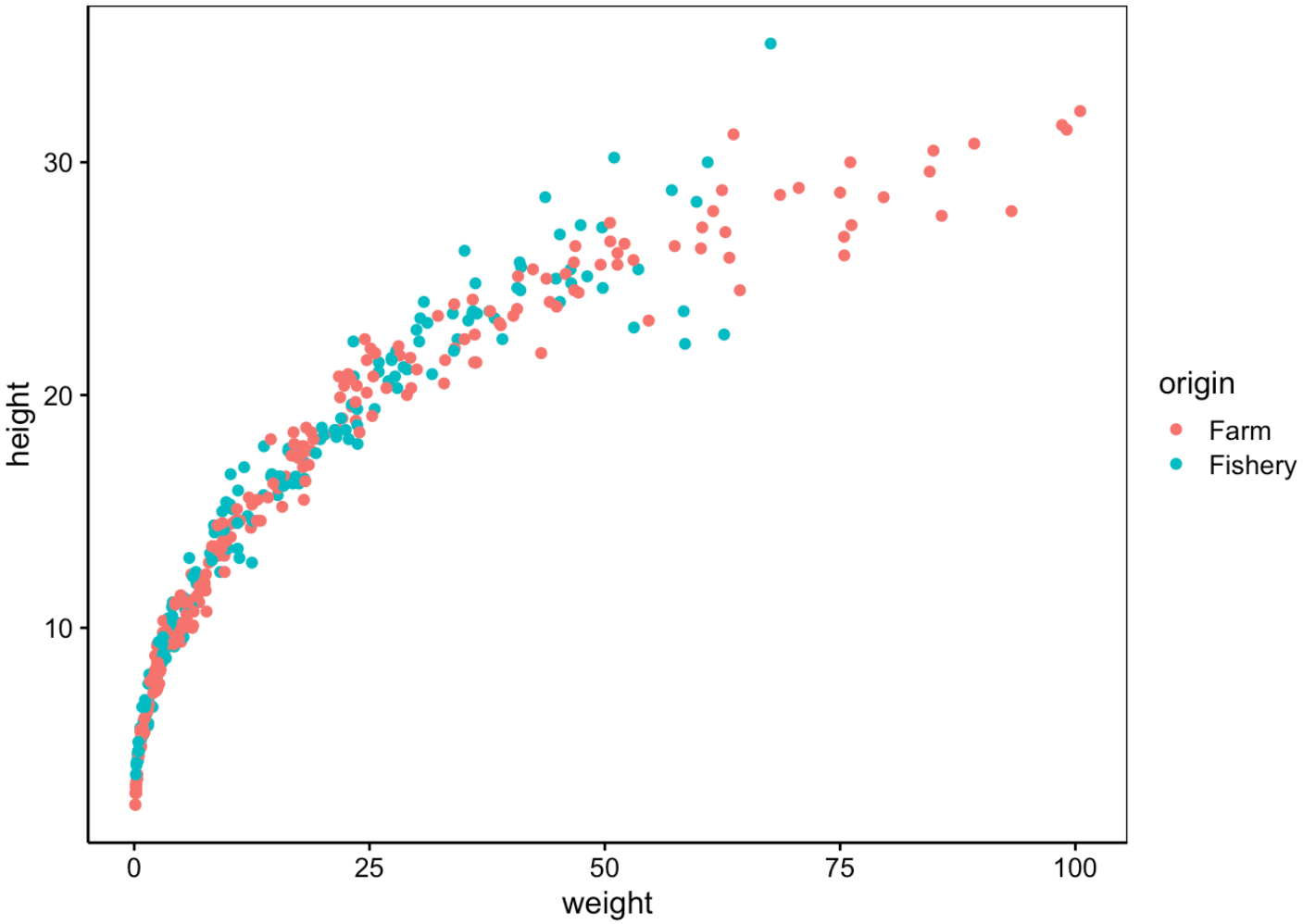
```
urchin <- read("urchin_bio", package = "data.io")
```

Nous avons déjà vu que `read()` donne accès également dans certains cas à des métadonnées (par exemple le label et les unités des jeux de données) dans différentes langues, ce que ne permet pas `data()`. Enfin, la syntaxe et la fonction utilisée sont pratiquement identiques pour charger des données depuis un fichier, depuis Internet ou depuis un package avec `read()`. C'est logique et facile à retenir. `data()` ne permet *que* de récupérer des données liées à un package R, et c'est tout ! Concernant l'importation des données depuis un fichier CSV, `read()` utilise du code rapide, là où `read.csv()` de R de base peine et prends énormément de temps avec des fichiers moyennement volumineux, et déclare forfait avec des gros jeux de données. Pour les données provenant d'Internet, seul `read()` offre le mécanisme de cache avec l'argument `cache_file=`. **Pour toutes ces raisons, nous préférons utiliser ici `read()` de `SciViews::R` au lieu des fonctions R de base telles que `data()` ou `read.csv()`.**

4.1.3.1 Langue du jeu de données

La fonction `read()` est également capable de lire un fichier annexe permettant de rajouter des **métadonnées** (données complémentaires) à notre tableau, comme les **labels** et les **unités** des variables en différentes langues. Lorsque le jeu de données est importé avec la fonction `data()`, ces métadonnées ne sont pas employées.

```
data("urchin_bio", package = "data.io")
# Visualisation des données
chart(urchin_bio, height ~ weight %col=% origin) +
  geom_point()
```

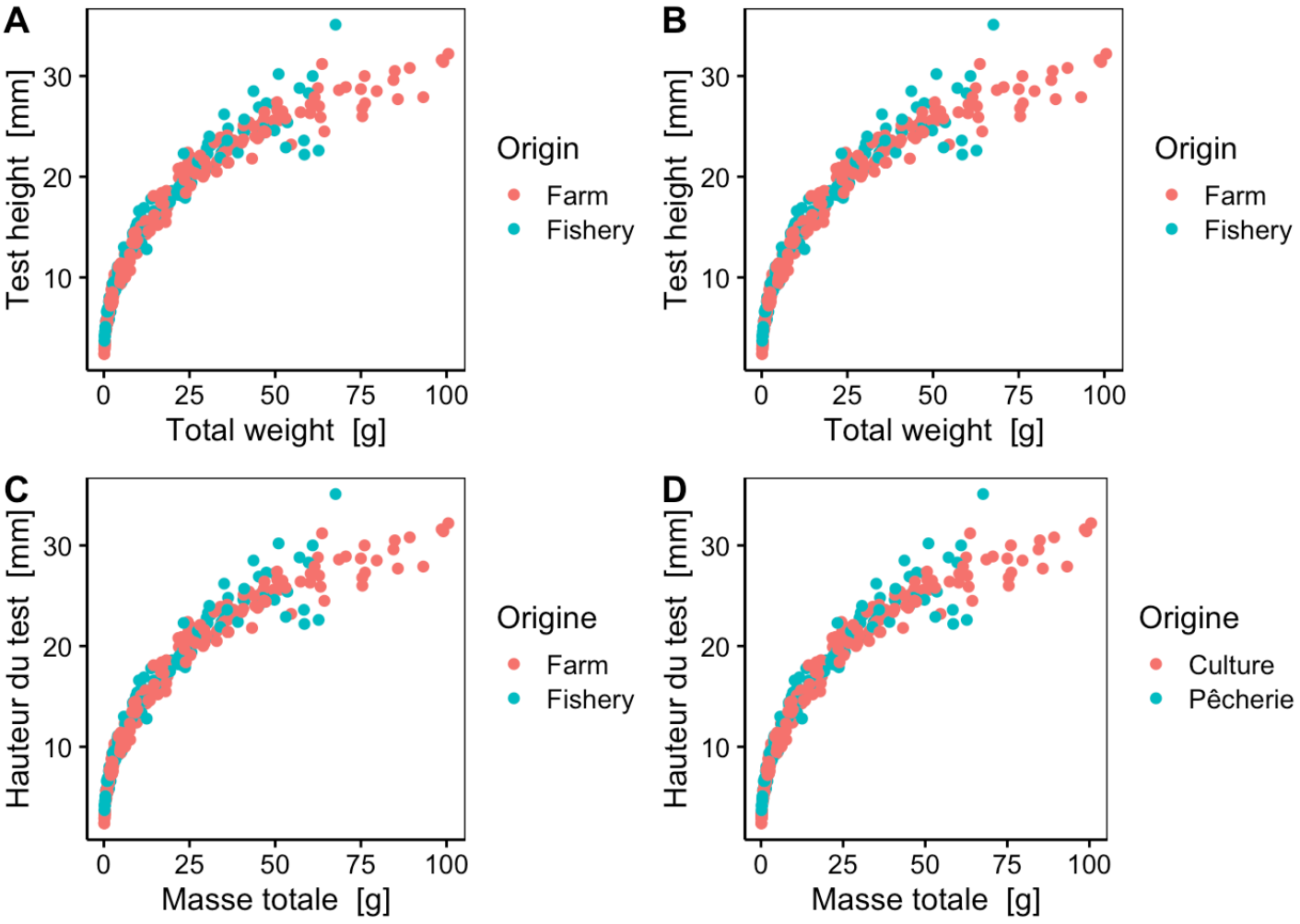


Comparez ceci avec le même graphique, mais obtenu à partir de différentes versions du jeu de données `urchin_bio` importé à l'aide de `read()` avec des valeurs différentes pour l'argument `lang = .`

```
urchin    <- read("urchin_bio", package = "data.io") # langage par défaut défi
urchin_en <- read("urchin_bio", package = "data.io", lang = "en")
urchin_fr <- read("urchin_bio", package = "data.io", lang = "fr")
urchin_FR <- read("urchin_bio", package = "data.io", lang = "FR")
```

Les différences dans les labels sont observables sur le graphique ci-dessous.

```
a <- chart(urchin,    height ~ weight %col=% origin) +
  geom_point()
b <- chart(urchin_en, height ~ weight %col=% origin) +
  geom_point()
c <- chart(urchin_fr, height ~ weight %col=% origin) +
  geom_point()
d <- chart(urchin_FR, height ~ weight %col=% origin) +
  geom_point()

combine_charts(list(a, b, c, d))
```



- `a` & `b` : l'argument `lang =` par défaut est `lang = "en"` . Il utilise les labels et unités en anglais avec les unités dans le système international.
- `c` : l'argument `lang = "fr"` utilise les labels et unités en français. Il laisse cependant les niveaux des variables facteurs en anglais (`Farm` et `Fishery`) afin

d'éviter de devoir changer les instructions de manipulation des données qui feraient référence à ces niveaux.

- `d` : l'argument `lang = "FR"` ajoute les labels et unités en français. De plus, il traduit également les niveaux des variables facteurs (`Culture` et `Pêcherie`).

Il vous est conseillé d'employer l'argument `lang = "fr"` lors de vos différents travaux. La langue internationale en science est l'anglais et vous serez très certainement amené dans votre carrière scientifique à produire des documents en français et en anglais. L'utilisation de `lang = "fr"` rend le **même** code réutilisable sur la version française ou anglaise, contrairement à `lang = "FR"` . Observez les exemples ci-dessous.

```
urchin_en %>.%
  filter_(., ~origin == "Farm") %>.%
  head(.)
```

```
# # A data.trame: [6 × 19]
#   origin diameter1 diameter2 height buoyant_weight weight solid_parts
#   <fct>      <dbl>     <dbl>  <dbl>          <dbl>  <dbl>       <dbl>
# 1 Farm        53.1      54.5   26.3           9.57   60.2        41.7
# 2 Farm        52.7      52.7   25.9          10.8    63.2        46.6
# 3 Farm        54        54.2   24.5          10.7    64.4        44.3
# 4 Farm        51.1      51.3   28.8          11.2    62.4        45.0
# 5 Farm        52.1      53.6   31.2          11.1    63.7        44.0
# 6 Farm        52.3      51.4   28.6          12.4    68.6        53.9
# # ℹ 12 more variables: integuments <dbl>, dry_integuments <dbl>,
# #   digestive_tract <dbl>, dry_digestive_tract <dbl>, gonads <dbl>,
# #   dry_gonads <dbl>, skeleton <dbl>, lantern <dbl>, test <dbl>, spines <dbl
# #   maturity <int>, sex <fct>
```

```
urchin_fr %>.%
  filter_(., ~origin == "Farm") %>.%
  head(.)
```

```
# # A data.trame: [6 × 19]
#   origin diameter1 diameter2 height buoyant_weight weight solid_parts
#   <fct>      <dbl>     <dbl>  <dbl>          <dbl>  <dbl>       <dbl>
# 1 Farm        53.1      54.5   26.3           9.57   60.2        41.7
# 2 Farm        52.7      52.7   25.9          10.8    63.2        46.6
# 3 Farm        54        54.2   24.5          10.7    64.4        44.3
# 4 Farm        51.1      51.3   28.8          11.2    62.4        45.0
# 5 Farm        52.1      53.6   31.2          11.1    63.7        44.0
# 6 Farm        52.3      51.4   28.6          12.4    68.6        53.9
# # ℹ 12 more variables: integuments <dbl>, dry_integuments <dbl>,
# #   digestive_tract <dbl>, dry_digestive_tract <dbl>, gonads <dbl>,
# #   dry_gonads <dbl>, skeleton <dbl>, lantern <dbl>, test <dbl>, spines <dbl
# #   maturity <int>, sex <fct>
```

Pas d'adaptation nécessaire du code pour passer de `urchin_en` à `urchin_fr` .

```
urchin_FR %>.%
  filter_(., ~origin == "Pêcherie") %>.%
  head(.)
```

```
# # A data.trame: [6 × 19]
#   origin   diameter1 diameter2 height buoyant_weight weight solid_parts
#   <fct>        <dbl>     <dbl>  <dbl>          <dbl>  <dbl>       <dbl>
# 1 Pêcherie       9.9      10.2    5               NA  0.522       0.478
# 2 Pêcherie      10.5      10.6    5.7             NA  0.642       0.589
# 3 Pêcherie      10.8      10.8    5.2             NA  0.734       0.677
# 4 Pêcherie       9.6       9.3    4.6             NA  0.370       0.344
# 5 Pêcherie      10.4      10.7    4.8             NA  0.610       0.559
# 6 Pêcherie      10.5      11.1    5               NA  0.610       0.551
# # ℹ 12 more variables: integuments <dbl>, dry_integuments <dbl>,
# #   digestive_tract <dbl>, dry_digestive_tract <dbl>, gonads <dbl>,
# #   dry_gonads <dbl>, skeleton <dbl>, lantern <dbl>, test <dbl>, spines <dbl
# #   maturity <int>, sex <fct>
```

Le code a dû être modifié dans l'instruction `filter_()` lors du passage à `urchin_FR` (nom du niveau `Farm -> Pêcherie`). Bien évidemment, pour un rapport *plus formel* en français, **tout** doit être traduit en français et l'option `lang = "FR"` accompagnée d'une vérification et une adaptation éventuelle du code est à préférer dans ce cas précis.

À vous de jouer !

Complétez les blancs dans la fonction ci-dessous pour importer le jeu de données **biometry** qui se trouve dans le package **BioDataScience1**. Assurez-vous que les niveaux des variables, les labels et les unités soient en français.

bio <- read("", package = "", lang = "")

Vérifier

18. Si vous utilisez le Tidyverse, vous pouvez utiliser la fonction `tibble::tribble()` à la place, avec la même syntaxe.↩

19. R permet également d'interroger des bases de données spécialisées, mais nous n'aborderons ce sujet spécifique qu'au cours de Science des Données Biologiques II en dernière année de Bachelier.↩

20. Le package {data.io}, et donc la fonction `read()` sont disponibles par défaut dans `SciViews::R`. Si vous travaillez avec un R de base, vous pouvez l'installer facilement avec l'instruction suivante : `install.packages('data.io', repos = c('https://sciviews.r-universe.dev', 'https://cloud.r-project.org'))`. Ensuite, vous faites `library(data.io)` pour rendre la fonction `read()` disponible ou vous l'appelez par `data.io::read()`.↩

21. Si vous devez aussi accéder à vos données à partir d'autres langages comme Python, Java ou C++, utilisez un format commun reconnu par les différents logiciels. Le CSV fonctionne généralement bien, mais des formats binaires plus performants sont également disponibles. Parmi ces formats "inter-langages", gardez un œil sur Apache Arrow qui offre des possibilités très intéressantes d'intercompatibilité et de performance.↩

4.2 Types de variables

Lors de la réalisation de graphiques dans les modules précédents, vous avez compris que toutes les variables ne sont pas équivalentes. Certains graphiques sont plutôt destinés à des variables **qualitatives** (par exemple, graphique en barres), alors que d'autres représentent des données **quantitatives** comme le nuage de points.

```
(biometry <- read("biometry", package = "BioDataScience", lang = "fr"))
```

```
# # A data.trame: [395 × 7]
#    gender day_birth  weight height wrist year_measure   age
#    <fct>  <date>      <dbl>  <dbl> <dbl>        <dbl> <dbl>
#  1 M      1995-03-11     69    182  15           2013    18
#  2 M      1998-04-03     74    190  16           2013    15
#  3 M      1967-04-04     83    185  17.5         2013    46
#  4 M      1994-02-10     60    175  15           2013    19
#  5 W      1990-12-02     48    167  14           2013    23
#  6 W      1994-07-15     52    179  14           2013    19
#  7 W      1971-03-03     72    167  15.5         2013    42
#  8 W      1997-06-24     74    180  16           2013    16
#  9 M      1972-10-26    110    189  19           2013    41
# 10 M      1945-03-15     82    160  18           2013    68
# # ℹ 385 more rows
```

La Figure 4.1 montre deux boites à moustaches parallèles différentes. Laquelle de ces deux représentations est incorrecte et pourquoi ?

```
a <- chart(biometry, height ~ gender %fill=% gender) +
  geom_boxplot()

b <- chart(biometry, height ~ weight %fill=% gender) +
  geom_boxplot()

combine_charts(list(a, b), common.legend = TRUE)
```

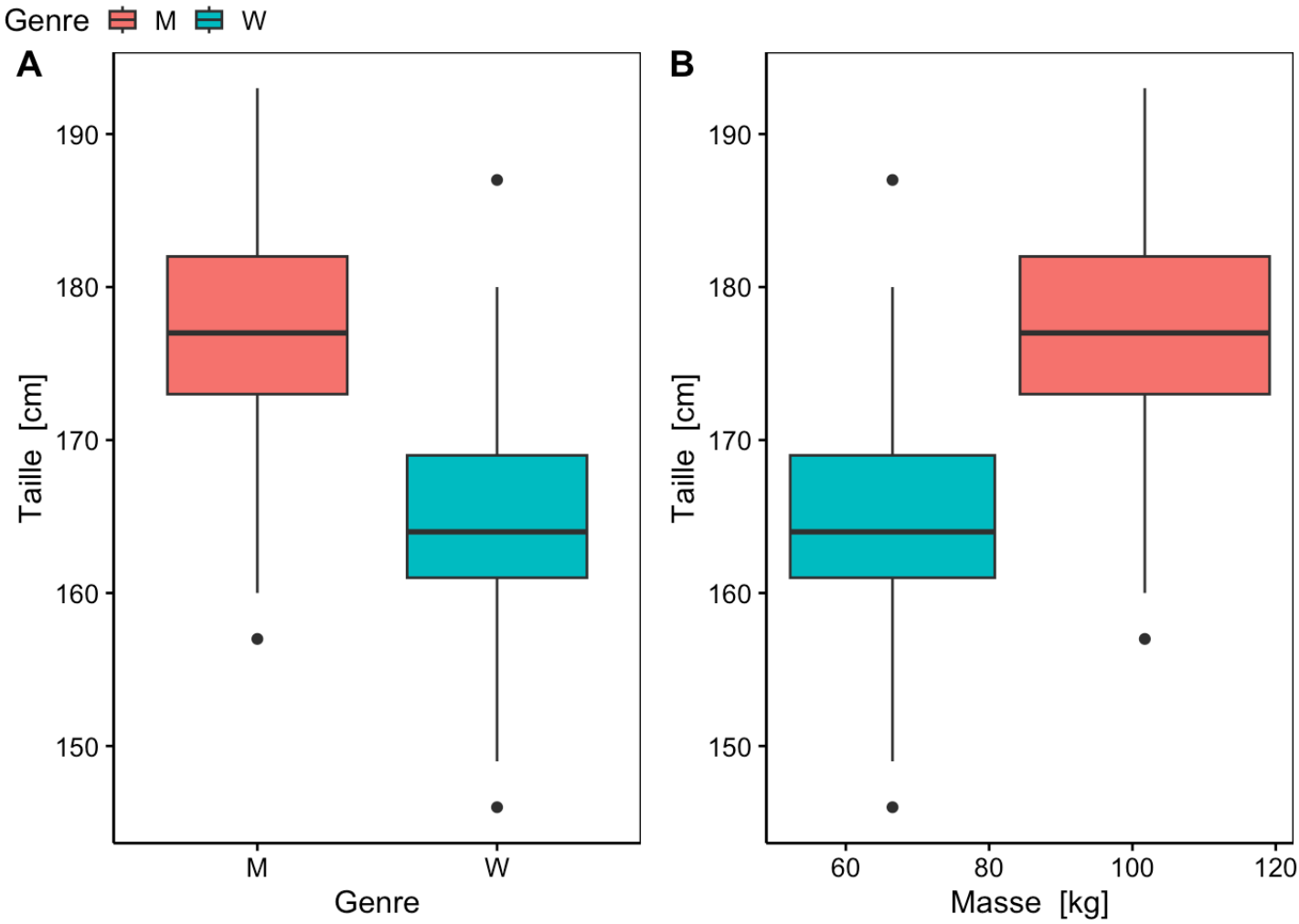


Figure 4.1: Boites à moustaches parallèles de la taille (`height`) en fonction de A. une variable qualitative (`gender`) et B. une variable quantitative (`weight`) et couleur en fonction de gender`

C'est la Figure 4.1B qui est incorrecte car elle tente de représenter une variable quantitative numérique `height` sous forme de boites à moustaches parallèles en fonction d'une variable de découpage en sous-ensemble (`weight`) qui est elle-même une **variable**

quantitative, … alors qu'une variable qualitative telle que `gender` aurait dû être utilisée pour ce découpage (comme dans la Fig. 4.1A). Dans le cas présent, R a bien voulu réaliser le graphique, mais comment l'interpréter ? Dans d'autres situations, il vous renverra purement et simplement un message d'erreur.

Les jeux de données, lorsqu'ils sont bien encodés (**tableaux "cas par variables"**, en anglais on parlera de tidy data), sont en fait un ensemble de variables en colonnes mesurées sur un ensemble d'individus en lignes. Vous avez à votre disposition plusieurs *types* de variables pour personnaliser le jeu de données. Deux catégories principales de variables existent, chacune avec deux sous-catégories :

- Les variables **quantitatives** sont issues de mesures quantitatives ou de dénombrements
  - Les variables quantitatives **continues** sont représentées par des valeurs réelles (**double** dans R)
  - Les variables quantitatives **discrètes** sont typiquement représentées par des entiers (**integer** dans R)
- Les variables **qualitatives** sont constituées d'un petit nombre de valeurs possibles (on parle des niveaux de la variables ou de leurs modalités)
  - Les variables qualitatives **ordonnées** ont des niveaux qui peuvent être classés dans un ordre du plus petit au plus grand. elles sont typiquement représentées dans R par des objets **ordered**.
  - Les variables qualitatives **non ordonnées** ont des niveaux qui ne peuvent être rangés et sont typiquement représentées par des objets **factor** en R

Il existe naturellement encore d'autres types de variables. Les dates sont représentées, par exemple, par des objets **Date**, les nombres complexes par **complex**, les données binaires par **raw**, etc.

La fonction `skimr::skim()` permet de visualiser la classe de la variable et bien plus encore. Elle fournit un résumé différent en fonction du type de la variable et propose, par exemple, un histogramme stylisé pour les variables numériques comme le montre le tableau ci-dessous.

```
skimr::skim(biometry)
```

Tableau 4.1: Data summary

Name	biometry
Number of rows	395
Number of columns	7
_____	
Column type frequency:	
Date	1
factor	1
numeric	5
_____	
Group variables	None

Variable type: Date

skim_variable	n_missing	complete_rate	min	max	median	n_unique
day_birth	0	1	1927-08-29	2000-08-11	1988-10-05	210

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
gender	0	1	FALSE	2	M: 198, W: 197

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25
weight	0	1.00	71.20	15.45	41.5	59.0
height	0	1.00	170.71	9.07	146.0	164.0
wrist	2	0.99	16.65	1.67	10.0	15.5
year_measure	0	1.00	2015.32	1.61	2013.0	2014.0
age	0	1.00	35.34	17.32	15.0	19.0

Avec une seule instruction, on obtient une quantité d'information sur notre jeu de données comme le nombre d'observations, le nombre de variables et un traitement spécifique pour chaque type de variable. Cette instruction permet de visualiser et d'appréhender le jeu de données mais ne doit généralement pas figurer tel quel dans un rapport final d'analyse (par contre il a toute sa place dans un bloc-notes, par exemple, où vous consignez vos différentes explorations comme dans un cahier de laboratoire).

À vous de jouer !

H5P

Quel est le format de la variable ci-dessous ? Vous avez à votre disposition les 10 premières valeurs de cette variable.

-> enfant, adulte, adulte, senior, enfant, senior, senior, enfant, enfant, senior

- ○ C'est une variable quantitative discrète
- ○ C'est une variable qualitative ordonnée
- ○ C'est une variable qualitative non ordonnée
- ○ C'est une variable quantitative continue

Afficher la réponse

4.3 Conversion de variables

Il est possible de convertir les variables seulement dans un sens : du plus détaillé au moins détaillé, c'est-à-dire, quantitatif continu -> quantitatif discret -> qualitatif ordonné -> qualitatif non ordonné.

4.3.1 Quantitatif continu à discret

R essaye de gommer autant que possible la distinction entre nombres **integer** et **double**, tous deux rassemblés en **numeric**. Si besoin, la conversion se fait automatiquement. En pratique, concentrez-vous essentiellement sur les objets **numeric** pour tout ce qui est quantitatif. Un nombre tel que `1` est considéré par R comme un **double** par défaut. Si vous vouliez expressément spécifier que c'est un entier, vous pouvez le faire en ajoutant un `L` majuscule derrière le nombre. Ainsi, `1L` est compris par R comme l'**entier** 1. Encore une fois, cette distinction *explicite* est rarement nécessaire dans R.

Si vous voulez arrondir des nombres, vous pouvez utiliser la fonction `round()` avec son argument `digits =` qui indique le nombre de chiffres à conserver derrière la virgule (0 par défaut, pour arrondir à l'entier le plus proche). Pour arrondir vers l'entier le plus proche vers le bas, utilisez `floor()` (plancher en anglais) et pour le plus proche vers le haut, employez `ceiling()` (plafond en anglais).

```
(x <- seq(-1, 1, by = 0.1) + 0.01)
```

```
#  [1] -0.99 -0.89 -0.79 -0.69 -0.59 -0.49 -0.39 -0.29 -0.19 -0.09  0.01  0.1
# [13]  0.21  0.31  0.41  0.51  0.61  0.71  0.81  0.91  1.01
```

```
round(x)
```

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

```
round(x, digits = 1)
```

```
#  [1] -1.0 -0.9 -0.8 -0.7 -0.6 -0.5 -0.4 -0.3 -0.2 -0.1  0.0  0.1  0.2  0.3
# [16]  0.5  0.6  0.7  0.8  0.9  1.0
```

```
ceiling(x)
```

```
#  [1] 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 2
```

```
floor(x)
```

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

À vous de jouer !

H5P

Complétez les blancs afin d'arrondir le vecteur x avec 2 décimales

x <- c(0.00001, 3.50821, 5.15687)

round([ ] , [ ])

Vérifier

4.3.2 Quantitatif à qualitatif

Le traitement diffère selon le nombre de valeurs différentes rencontrées dans le jeu de données. Si une variable numérique contient en réalité un petit nombre de valeurs différentes, il suffit de convertir la *classe* de l'objet de **numeric** vers **factor** ou **ordered** pour obtenir une variable qualitative. Un exemple concret l'illustre ci-dessous. Si, par contre, le nombre de valeurs différentes est important (dizaines ou plus) alors il va falloir créer des regroupements. C'est le **découpage en classes** abordé plus loin.

Voici un jeu de données qui étudie l'allongement des dents chez le cobaye (cochon d'Inde) en fonction de la supplémentation alimentaire en acide ascorbique.

```
tooth <- read("ToothGrowth", package = "datasets", lang = "fr")
skimr::skim(tooth)
```

Tableau 4.2: Data summary

Name	tooth
Number of rows	60
Number of columns	3
_____	
Column type frequency:	
factor	1
numeric	2
_____	
Group variables	None

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
supp	0	1	FALSE	2	OJ: 30, VC: 30

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50
len	0	1	18.81	7.65	4.2	13.07	19.25
dose	0	1	1.17	0.63	0.5	0.50	1.00

Le jeu de données contient 60 observations effectuées sur des cochons d'Inde. Ces derniers reçoivent deux types de suppléments alimentaires : soit du jus d'orange (`OJ`), soit de la vitamine C (`VC`). Des lots différents reçoivent des doses différentes d'acide ascorbique par ces suppléments, soit 0.5, 1, ou 2 mg/j.

La variable dose est encodée sous forme numérique alors que cette dernière ne contient que trois niveaux différents et devra être le plus souvent traitée comme une **variable qualitative ordonnée à trois niveaux. Vous devrez donc probablement réencoder cette variable en variable facteur (ordonné).**

> Ce n'est pas le caractère quantitatif ou qualitatif du mécanisme sous-jacent mesuré qui détermine si la variable est quantitative ou qualitative, mais d'autres critères comme la précision avec laquelle la mesure a été effectuée. Par exemple, un anémomètre mesure la vitesse du vent sous forme de variable **quantitative** alors qu'une échelle approximative de type `vent nul` , `vent faible` , `vent moyen` , `vent fort` ou `tempête` basée sur l'observation des rides ou des vagues à la surface de la mer pourrait éventuellement convenir pour mesurer le même phénomène si une grande précision n'est pas nécessaire. Mais dans ce cas, la variable devra être traitée comme une variable **qualitative** ordonnée.
>
> De même, un plan expérimental qui réduit volontairement les valeurs fixées dans une expérience, comme ici les doses journalières d'acide ascorbique, fera

aussi basculer la variable en **qualitative**, et ce, quelle que soit la précision avec laquelle les valeurs sont mesurées par ailleurs. Un découpage en classes (voir ci-dessous) aura aussi le même effet de transformer une variable quantitative en variable qualitative ordonnée.

Indiquons à présent explicitement à R que la variable `dose` doit être considérée comme qualitative :

```
tooth$dose <- as.factor(tooth$dose)
# Visualisation des données
skimr::skim(tooth)
```

Tableau 4.3: Data summary

Name	tooth
Number of rows	60
Number of columns	3
_____	
Column type frequency:	
factor	2
numeric	1
_____	
Group variables	None

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
supp	0	1	FALSE	2	OJ: 30, VC: 30
dose	0	1	FALSE	3	0.5: 20, 1: 20, 2: 20

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50
len	0	1	18.81	7.65	4.2	13.07	19.25

Vous pouvez (et devez !) cependant aller encore plus loin, car la variable est en réalité qualitative **ordonnée**, et doit être représentée par un objet “facteur ordonné” (**ordered**) plutôt que **factor**. Il y a en effet, une progression dans les doses administrées. Lors de la conversion, R considère les différents niveaux par **ordre alphabétique** par défaut. Ici cela convient, mais ce n’est pas toujours le cas. Il vaut donc mieux spécifier explicitement l’*ordre* des niveaux dans l’argument optionnel `levels =` . Cela donne :

```
tooth$dose <- ordered(tooth$dose, levels = c(0.5, 1, 2))
# Visualisation des données
skimr::skim(tooth)
```

Tableau 4.4: Data summary

Name	tooth
Number of rows	60
Number of columns	3
_____	
Column type frequency:	
factor	2
numeric	1
_____	
Group variables	None

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
supp	0	1	FALSE	2	OJ: 30, VC: 30
dose	0	1	TRUE	3	0.5: 20, 1: 20, 2: 20

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50
len	0	1	18.81	7.65	4.2	13.07	19.25

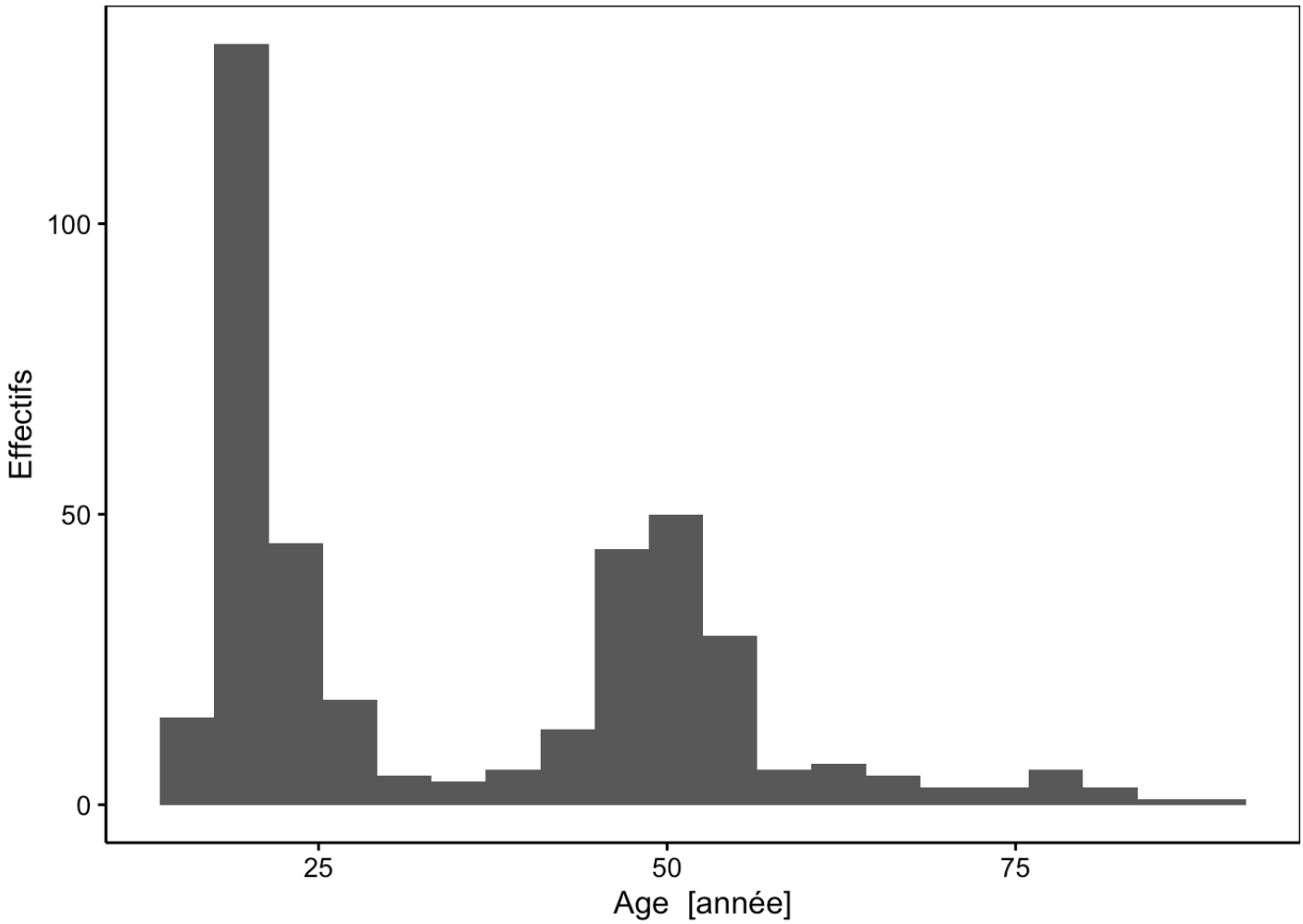
Les fonctions `as.factor()` ou `factor()` et `as.ordered()` ou `ordered()` effectuent cette conversion de **character** ou **numeric** vers des objets **factor** ou **ordered**. Une variable facteur ordonnée sera alors reconnue comme telle par un ensemble de fonction

dans R. Elle ne sera, de ce fait, pas traitée de la même manière qu'une variable facteur non ordonnée, ni même qu'une variable numérique. Soyez bien attentif à l'encodage correct des données dans R avant d'effectuer vos graphiques et vos analyses.

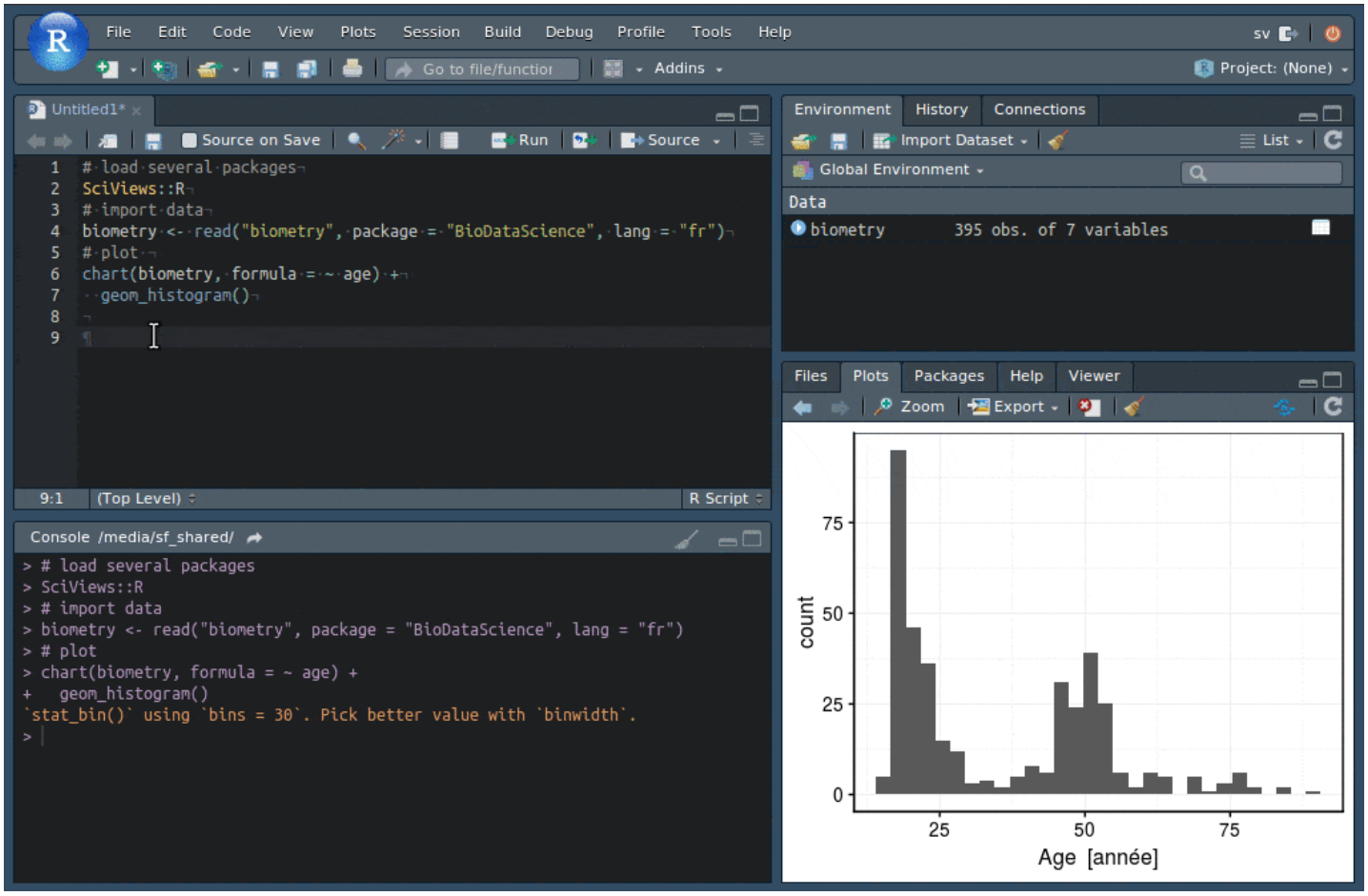
4.3.3 Découpage en classes

La conversion d'une variable quantitative en une variable qualitative doit souvent passer par une réduction des niveaux en rassemblant les valeurs proches dans des **classes**. Vous avez déjà utilisé de manière implicite le découpage en classes lorsque vous avez réalisé des histogrammes. Si les histogrammes sont bi- ou multimodaux, un découpage se justifie. Par exemple, le jeu de données portant sur la biométrie humaine est typique d'un cas de distribution bimodale. En fait, ce sont des étudiants (ayant tous une vingtaine d'années) qui ont réalisé ces mesures. La plupart ont choisi de s'inclure dans l'échantillon, d'où un premier mode vers une vingtaine d'années. Ensuite, ils ont pu mesurer d'autres personnes, éventuellement dans leur entourage. Beaucoup ont demandé à leurs parents, ce qui résulte en un second mode vers la cinquantaine[22]. Donc, la distribution bimodale résulte plus de l'échantillonnage en lui-même que d'une réalité démographique ! Cela ne change cependant rien pour l'exercice.

```
biometry <- read("biometry", package = "BioDataScience", lang = "fr")
chart(data = biometry, ~ age) +
  geom_histogram(bins = 20) +
  ylab("Effectifs")
```

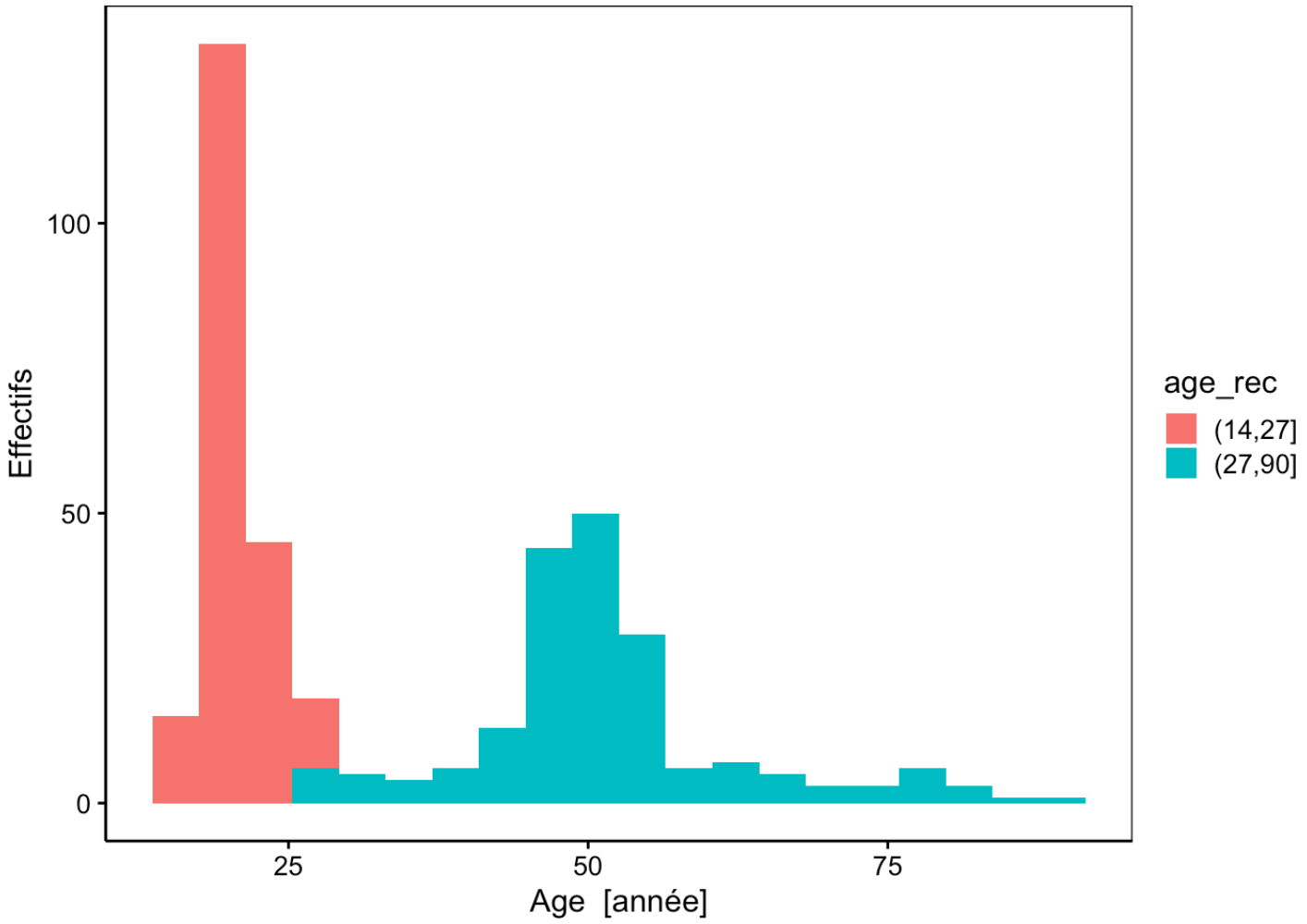


Les **addins** de RStudio vont vous permettre de réaliser facilement un découpage du jeu de données en fonction de classes d'âges (bouton `Extensions -> QUESTIONR -> Numeric range dividing`)[23].



Vous spécifiez le découpage voulu dans une boite de dialogue sur base de l'histogramme et lorsque vous cliquez sur le bouton `Done`, le code R qui effectue ce découpage est inséré dans l'éditeur RStudio à l'endroit du curseur. La nouvelle variable facteur `age_rec` basée sur le découpage en classes sera ensuite utile pour faire ressortir de l'information supplémentaire en contrastant les individus plus jeunes et ceux plus âgés.

```
# Instructions obtenues à partir de l'addins
biometry$age_rec <- cut(biometry$age, include.lowest = FALSE, right = TRUE,
  breaks = c(14, 27, 90))
# Visualisation de la variable facteur obtenue
chart(biometry, formula = ~ age %fill=% age_rec) +
  geom_histogram(bins = 20) +
  ylab("Effectifs")
```



4.3.4 Qualitatif ordonné ou non

Les données qualitatives sont souvent représentées par du texte (nom d'une couleur par exemple) et importées sous forme de chaînes de caractère (**character**) par défaut dans R à partir de la fonction `read()`. Vous devez les convertir de manière explicite à l'aide de `as.factor()`, `factor()`, `as.ordered()` ou `ordered()` par la suite. Voici un exemple :

```
df <- dtx(
  color     = c("blue", "green", "blue", "red", "green"),
  intensity = c("low",  "low",   "high", "mid", "high"))
df
```

```
# # A data.trame: [5 × 2]
#   color intensity
#   <chr> <chr>
# 1 blue  low
# 2 green low
# 3 blue  high
# 4 red   mid
# 5 green high


# Conversion en factor (color) et ordered (intensity)
df$color <- factor(df$color,
  levels = c("red", "green", "blue"))
df$intensity <- ordered(df$intensity,
  levels = c("low", "mid", "high"))
df


# # A data.trame: [5 × 2]
#   color intensity
#   <fct> <ord>
# 1 blue  low
# 2 green low
# 3 blue  high
# 4 red   mid
# 5 green high


# Information plus détaillée
str(df)
```

```
# dtrm [5 × 2] (S3: data.trame/data.frame)
#  $ color    : Factor w/ 3 levels "red","green",..: 3 2 3 1 2
#  $ intensity: Ord.factor w/ 3 levels "low"<"mid"<"high": 1 1 3 2 3
#  - attr(*, ".internal.selfref")=<externalptr>
```

```
skimr::skim(df)
```

Tableau 4.5: Data summary

Name	df
Number of rows	5
Number of columns	2
_____	
Column type frequency:	
factor	2
_____	
Group variables	None

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
color	0	1	FALSE	3	gre: 2, blu: 2, red: 1
intensity	0	1	TRUE	3	low: 2, hig: 2, mid: 1

> Les différents niveaux des variables **factor** ou **ordered** sont et doivent rester entièrement de votre responsabilité. Certains aspects anciens de R essayent de

> gérer cela pour vous, mais ces fonctions ou options (`StringsAsFactor =` par exemple) tendent heureusement à être remplacées par des versions moins assertives. De même, les niveaux ne sont **pas** réduits lorsque vous filtrez un tableau pour ne retenir que certains niveaux. Vous devez indiquer explicitement ensuite que vous voulez éliminer les niveaux vides du tableau avec la fonction `droplevels()` .

Le jeu de données `iris` contient des données relatives à trois espèces différentes (`table()` permet de compter le nombre d'observations pour chaque niveau d'une variable qualitative **factor** ou **ordered**) :

```
iris <- read("iris", package = "datasets", lang = "fr")
table(iris$species)


#
#     setosa versicolor  virginica
#         50         50         50
```

Si nous restreignons le tableau aux 20 premiers individus, cela donne :

```
iris20 <- iris[1:20, ]
table(iris20$species)


#
#     setosa versicolor  virginica
#         20          0          0
```

Nous voyons que le tableau réduit `iris20` ne contient des données que d'une seule espèce. Pourtant `table()` continue de lister les autres niveaux de la variable. Les niveaux connus sont aussi imprimés avec `levels()` :

```
levels(iris20$species)
```

```
# [1] "setosa"     "versicolor" "virginica"
```

Dans le cas ici, nous souhaitons uniquement nous focaliser sur l'espèce *Iris setosa*. Dès lors, l'utilisation de la fonction `droplevels()` permet de faire disparaître les autres niveaux de la variable `species`.

```
iris20$species <- droplevels(iris20$species)
levels(iris20$species)
```

```
# [1] "setosa"
```

```
table(iris20$species)
```

```
#
# setosa
#     20
```

22. Notez que ceci ne constitue **pas** un échantillonnage correct par rapport à la population générale du Hainaut pour plusieurs raisons. (1) toutes les tranches d'âges ne sont échantillonnées de manière équivalente pour les raisons évoquées, (2) des liens génétiques existent au sein des familles, ce qui résulte en une **non indépendance** des observations entre elles, et (3) seule une sous-population constituée de personnes fréquentant l'université et de leur entourage a été échantillonnée. Cependant, dans le cadre de l'exercice, nous accepterons ces biais, tout en étant conscients qu'ils existent.↩

23. Si vous travaillez avec un R de base hors SciViews Box, vous devez d'abord installer le package {questionr} à l'aide de l'instruction `install.packages("questionr")` pour avoir accès à cet addin.↩

4.4 Remaniement des données

Dans le module 3, vous avez réalisé vos premiers remaniements de données dans le cadre des graphiques en barres. Nous ne nous sommes pas étendus sur les fonctions utilisées à cette occasion. Le **remaniement des données est une étape cruciale en analyse des données** et il faut en maîtriser au moins les principaux outils. Heureusement, il est déjà possible d'aller loin en combinant une petite dizaine d'outils simples. Les cinq principaux (les plus utilisés) dans l'approche Tidyverse utilisée ici sont :

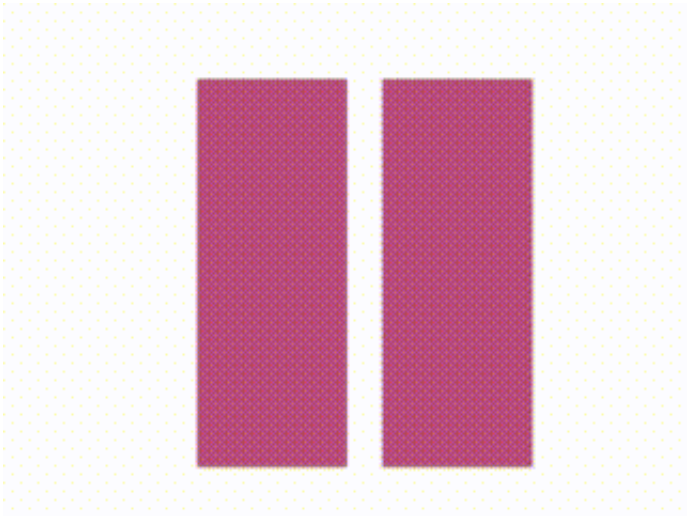
- sélectionner des colonnes au sein d'un jeu de données avec `select() / select_()`
- filtrer des lignes dans un jeu de données avec `filter() / filter_()`
- calculer de nouvelles variables dans un jeu de données avec `mutate() / mutate_()`
- regrouper les données au sein d'un tableau avec `group_by() / group_by_()`
- résumer les variables d'un jeu de données avec `summarise() / summarise_()`

Ces outils provenant du package {dplyr} et de {collapse} et {svTidy}[24] pour les versions suffixées d'un `_` sont décrits en détails dans le chapitre 4 de "R for Data Science (2e)". Nous allons nous familiariser avec ces outils en adoptant une approche pratique sur base de résolution d'exemples concrets.

```
urchin <- read("urchin_bio", package = "data.io", lang = "fr")
urchin
```

```
# # A data.trame: [421 × 19]
#    origin  diameter1 diameter2 height buoyant_weight weight solid_parts
#    <fct>       <dbl>     <dbl>  <dbl>          <dbl>  <dbl>       <dbl>
#  1 Fishery       9.9      10.2    5               NA  0.522       0.478
#  2 Fishery      10.5      10.6    5.7             NA  0.642       0.589
#  3 Fishery      10.8      10.8    5.2             NA  0.734       0.677
#  4 Fishery       9.6       9.3    4.6             NA  0.370       0.344
#  5 Fishery      10.4      10.7    4.8             NA  0.610       0.559
#  6 Fishery      10.5      11.1    5               NA  0.610       0.551
#  7 Fishery      11        11      5.2             NA  0.672       0.605
#  8 Fishery      11.1      11.2    5.7             NA  0.703       0.628
#  9 Fishery       9.4       9.2    4.6             NA  0.413       0.375
# 10 Fishery      10.1       9.5    4.7             NA  0.449       0.398
# # ℹ 411 more rows
# # ℹ 12 more variables: integuments <dbl>, dry_integuments <dbl>,
# #   digestive_tract <dbl>, dry_digestive_tract <dbl>, gonads <dbl>,
# #   dry_gonads <dbl>, skeleton <dbl>, lantern <dbl>, test <dbl>, spines <dbl
# #   maturity <int>, sex <fct>
```

4.4.1 select() / select_()



Lors de l'utilisation de vos jeux de données, vous serez amené à réduire vos données en sous-tableau ne reprenant qu'un sous-ensemble des variables initiales. `select()` et `select_()` effectuent cette opération[25] :

La fonction `select()` est une fonction du "tidyverse" qui a un comportement particulier dans R. Une fonction relativement équivalente, mais "svTidy" est `select_()`. Le suffixe `_` est là pour indiquer que c'est une fonction légèrement différente (du point de vue de sa syntaxe, et aussi, de sa vitesse d'exécution qui est meilleure dans bien des cas). Les fonctions tidyverse, ont cependant d'autres qualités. Notamment, elles fonctionneront aussi en grande partie sur des bases de données. Ces deux fonctions sont souvent interchangeables, mais pas toujours. **Dans `SciViews::R`, il est très important de faire la distinction entre les deux types de fonctions et leurs particularités.**

4.4.1.1 Fonctions "svTidy"

Elles sont l'équivalent fonctionnel des fonctions du même nom, mais sans suffixe `_` qui sont définies dans le tidyverse. Rappelez-vous de **ne pas mélanger les fonctions svTidy et tidyverse dans une même instruction R.**

Par exemple, si vous voulez utiliser `select_` pour extraire du tableau `urchin` un sous-tableau qui ne reprend que les variables (= colonnes) `origin`, `height` et `skeleton`, vous utiliserez :

```
urchin2 <- select_(urchin, "origin", "height", "skeleton")
head(urchin2)
```

```
# # A data.trame: [6 × 3]
#   origin  height skeleton
#   <fct>    <dbl>    <dbl>
# 1 Fishery    5      0.179
# 2 Fishery    5.7    0.188
# 3 Fishery    5.2    0.235
# 4 Fishery    4.6    0.063
# 5 Fishery    4.8   NA
# 6 Fishery    5     NA
```

Vous voyez que vous obtenez ici un `data.trame` qui ne contient plus que trois colonnes.

Vous pouvez aussi utiliser des **formules** pour sélectionner les variables. Un formule dans R est un objet défini par l'opérateur **tilde** `~` . Il existe des formules à un seul membre (à droite du tilde), comme `~x` , et des formules à deux membres, un à gauche et l'autre à droite du tilde, comme `y ~ x` . Vous avez déjà utilisé les formules avec vos graphiques `chart()` . Avec `select_()` , vous pouvez également écrire :

```
urchin2 <- select_(urchin, ~origin, ~height, ~skeleton)
head(urchin2)
```

```
# # A data.trame: [6 × 3]
#   origin  height skeleton
#   <fct>    <dbl>    <dbl>
# 1 Fishery    5      0.179
# 2 Fishery    5.7    0.188
# 3 Fishery    5.2    0.235
# 4 Fishery    4.6    0.063
# 5 Fishery    4.8   NA
# 6 Fishery    5     NA
```

Avec la fonction tidyverse `select()`, vous n'utilisez **pas** de formule, mais vous pouvez indiquer le nom des variables avec ou sans les guillements :

```
urchin2 <- select(urchin, origin, height, skeleton)
head(urchin2)
```

```
# # A data.trame: [6 × 3]
#   origin  height skeleton
#   <fct>    <dbl>    <dbl>
# 1 Fishery    5      0.179
# 2 Fishery    5.7    0.188
# 3 Fishery    5.2    0.235
# 4 Fishery    4.6    0.063
# 5 Fishery    4.8   NA
# 6 Fishery    5     NA
```

Tant avec `select()` que `select_()`, vous pouvez pouvez *éliminer* des variables en indiquant un signe moins `-` devant leur nom, mais seulement en utilisant la version formule pour `select_()`.

```
urchin2 <- select_(urchin, ~ -origin, ~ -height, ~ -skeleton)
head(urchin2)
```

```
# # A data.trame: [6 × 16]
#   diameter1 diameter2 buoyant_weight weight solid_parts integuments
#       <dbl>     <dbl>          <dbl>  <dbl>       <dbl>       <dbl>
# 1       9.9      10.2             NA  0.522       0.478       0.366
# 2      10.5      10.6             NA  0.642       0.589       0.445
# 3      10.8      10.8             NA  0.734       0.677       0.533
# 4       9.6       9.3             NA  0.370       0.344       0.266
# 5      10.4      10.7             NA  0.610       0.559       0.406
# 6      10.5      11.1             NA  0.610       0.551       0.427
# # ℹ 10 more variables: dry_integuments <dbl>, digestive_tract <dbl>,
# #   dry_digestive_tract <dbl>, gonads <dbl>, dry_gonads <dbl>, lantern <dbl>
# #   test <dbl>, spines <dbl>, maturity <int>, sex <fct>
```

Enfin, vous pouvez aussi indiquer une plage de variable (de la première à la dernière) en utilisant `première_var:dernière_var` (dans une formule pour `select_()`).

```
urchin2 <- select_(urchin, ~ origin:height)
head(urchin2)
```

```
# # A data.trame: [6 × 4]
#   origin  diameter1 diameter2 height
#   <fct>       <dbl>     <dbl>  <dbl>
# 1 Fishery       9.9      10.2    5
# 2 Fishery      10.5      10.6    5.7
# 3 Fishery      10.8      10.8    5.2
# 4 Fishery       9.6       9.3    4.6
# 5 Fishery      10.4      10.7    4.8
# 6 Fishery      10.5      11.1    5
```

À noter que la façon de sélectionner des colonnes dans un **data.frame** en R de base se fait différemment. R de base utilise l'opérateur `[` dans lequel vous spécifiez les lignes à conserver d'abord, puis, séparé par une virgule, les colonnes à conserver. Si vous n'indiquez rien devant la virgule vous conservez toutes les lignes, et si vous n'indiquez rien derrière la virgule vous conservez toutes les colonnes. Donc `urchin[ , ]` conservera le tableau entier. La sélection se fait à l'aide de nombre entiers qui indiquent l'**index** des lignes ou des colonnes à conserver. Si ce sont des nombres négatifs, ces lignes ou colonnes sont enlevées. Par exemple pour conserver les colonnes 1 et 3 de `urchin`, on écrira en R de base :

```
urchin2 <- urchin[, c(1, 3)]
head(urchin2)
```

```
# # A data.trame: [6 × 2]
#   origin  diameter2
#   <fct>       <dbl>
# 1 Fishery      10.2
# 2 Fishery      10.6
# 3 Fishery      10.8
# 4 Fishery       9.3
# 5 Fishery      10.7
# 6 Fishery      11.1
```

Au contraire, pour conserver tout sauf les colonnes 1 et 3, on écrira :

```
urchin2 <- urchin[, c(-1,-3)] # ou -c(1, 3)
head(urchin2)
```

```
# # A data.trame: [6 × 17]
#   diameter1 height buoyant_weight weight solid_parts integuments dry_integum
#       <dbl>  <dbl>          <dbl>  <dbl>       <dbl>       <dbl>           <
# 1       9.9    5               NA  0.522       0.478       0.366
# 2      10.5    5.7             NA  0.642       0.589       0.445
# 3      10.8    5.2             NA  0.734       0.677       0.533
# 4       9.6    4.6             NA  0.370       0.344       0.266
# 5      10.4    4.8             NA  0.610       0.559       0.406
# 6      10.5    5               NA  0.610       0.551       0.427
# # ℹ 10 more variables: digestive_tract <dbl>, dry_digestive_tract <dbl>,
# #   gonads <dbl>, dry_gonads <dbl>, skeleton <dbl>, lantern <dbl>, test <dbl
# #   spines <dbl>, maturity <int>, sex <fct>
```

Enfin, pour être complet, vous pouvez aussi utiliser un vecteur de valeurs logiques pour sélectionner les lignes ou les colonnes à conserver en R de base.

4.4.1.2 Tidy-select

Tidy-select est un mode de fonctionnement particulier pour définir un argument de fonction. Il permet d'indiquer le nom des variables à sélectionner, ou à éliminer si elles sont précédées d'un signe moins, ou d'une plage de variables en utilisant `première_var:dernière_var`, des fonctionnalités déjà vues ci-dessus. Mais elle permet également de faire beaucoup plus. Une série de fonctions annexes facilitent la sélection des variables, voir `?dplyr::select`.

Les fonctions `select()` et `select_()` permettent aussi de sélectionner des colonnes par leur position dans le tableau :

```
urchin2 <- select_(urchin, c(1, 4, 14))
head(urchin2)
```

```
# # A data.trame: [6 × 3]
#   origin  height skeleton
#   <fct>    <dbl>    <dbl>
# 1 Fishery    5      0.179
# 2 Fishery    5.7    0.188
# 3 Fishery    5.2    0.235
# 4 Fishery    4.6    0.063
# 5 Fishery    4.8   NA
# 6 Fishery    5     NA
```

La fonction `contains()` est utile pour sélectionner les variables dont le nom contient une chaîne de caractères. Utilisez-la dans une formule avec `select_()` .

```
urchin3 <- select_(urchin, ~origin, ~contains("weight"))
head(urchin3)
```

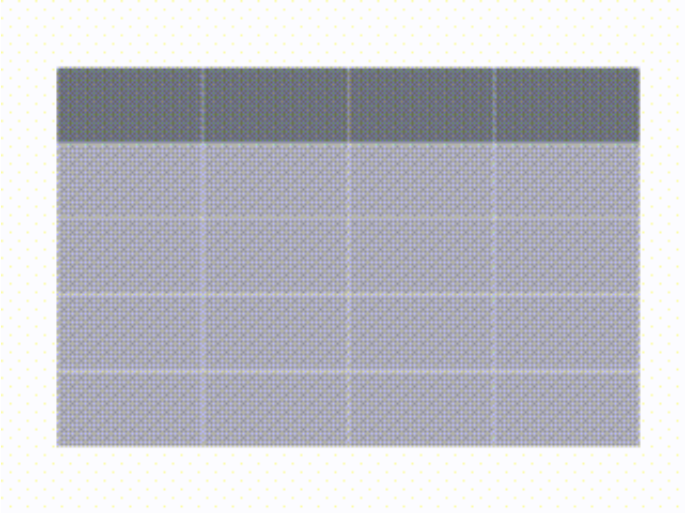
```
# # A data.trame: [6 × 3]
#   origin  buoyant_weight weight
#   <fct>            <dbl>  <dbl>
# 1 Fishery             NA  0.522
# 2 Fishery             NA  0.642
# 3 Fishery             NA  0.734
# 4 Fishery             NA  0.370
# 5 Fishery             NA  0.610
# 6 Fishery             NA  0.610
```

Vous pouvez aussi utiliser `starts_with()` ou `ends_with()` et d'autres fonctions du genre rassemblés dans la page d'aide `?dplyr::select_helpers` :

```
urchin4 <- select_(urchin, ~ends_with("ht"))
head(urchin4)
```

```
# # A data.trame: [6 × 3]
#   height buoyant_weight weight
#    <dbl>          <dbl>  <dbl>
# 1    5               NA  0.522
# 2    5.7             NA  0.642
# 3    5.2             NA  0.734
# 4    4.6             NA  0.370
# 5    4.8             NA  0.610
# 6    5               NA  0.610
```

4.4.2 filter() / filter_()



De même que toutes les colonnes d'un tableau ne sont pas forcément utiles, il est souvent nécessaire de sélectionner les lignes en fonction de critères particuliers pour restreindre l'analyse à une sous-population données, ou pour éliminer les cas qui ne correspondent pas à ce que vous voulez. `filter()` est une fonction tidyverse qui effectue ce travail. L'équivalent svTidy est `filter_()` . Repartons du jeu de données `urchin` simplifié à trois variables (`urchin2`).

```
urchin2 <- select_(urchin, ~origin, ~height, ~skeleton)
head(urchin2)
```

```
# # A data.trame: [6 × 3]
#   origin  height skeleton
#   <fct>    <dbl>    <dbl>
# 1 Fishery    5      0.179
# 2 Fishery    5.7    0.188
# 3 Fishery    5.2    0.235
# 4 Fishery    4.6    0.063
# 5 Fishery    4.8   NA
# 6 Fishery    5     NA
```

Si vous voulez sélectionner uniquement un niveau `"lvl"` d'une variable facteur `fact`, vous pouvez utiliser une **instruction de comparaison** "égal à" (`==`) : `fact == "lvl"`. Notez bien le **double** signe égal ici, et n'oubliez pas d'indiquer le niveau entre guillemets. De même, vous pouvez sélectionner tout **sauf** ce niveau avec l'opérateur "différent de" (`!=`). Les opérateurs "plus petit que" (`<`) ou "plus grand que" (`>`) fonctionnent sur les chaines de caractères selon une logique d'ordre alphabétique, donc, `"a" < "b"` [26].

Comparaison	Opérateur	Exemple
Égal à	`==`	`fact == "lvl"`
Différent de	`!=`	`fact != "lvl"`
Plus grand que	`>`	`fact > "lvl"`
Plus grand ou égal à	`>=`	`fact >= "lvl"`
Plus petit que	`<`	`fact < "lvl"`
Plus petit ou égale à	`<=`	`fact <= "lvl"`

Bien entendu, les comparaisons sont aussi possibles entre données numériques. Dans ce cas, vous **n'utilisez pas de guillemets**. Par exemple pour indiquer une condition telle qu'une variable numérique `x` doit être supérieure à 15, vous indiquerez `x > 15` et surtout pas `x > "15"` dans ce cas.

En version svTidy, vous utiliserez `filter_()` pour indiquer les *lignes* de votre jeu de données à conserver, la plupart du temps grâce à un test de condition dans une formule (donc, commençant par un tilde `~`. Ainsi pour conserver tous les oursins qui ne sont pas issus de la pêche, vous indiquerez :

```
# Tous les oursins sauf ceux issus de la pêche
urchin_sub1 <- filter_(urchin2, ~origin != "Fishery")
urchin_sub1
```

```
# # A data.trame: [218 × 3]
#    origin height skeleton
#    <fct>   <dbl>    <dbl>
#  1 Farm     26.3  14.6
#  2 Farm     25.9  16.1
#  3 Farm     24.5  15.7
#  4 Farm     28.8  16.6
#  5 Farm     31.2  NA
#  6 Farm     28.6  NA
#  7 Farm     27.2  NA
#  8 Farm      2.9   0.0605
#  9 Farm      3.5   0.101
# 10 Farm      3.1   0.065
# # ℹ 208 more rows
```

Vous pouvez aussi utiliser une variable numérique pour filtrer les données. Les comparaisons précédentes sont toujours applicables, sauf que cette fois vous faites porter la comparaison par rapport à une constante (ou par rapport à une autre variable numérique) et comme nous l'avons déjà expliqué, vous ne l'indiquez **pas** entre guillemets.

```
# Oursins plus hauts que 20mm
urchin_sub2 <- filter_(urchin2, ~height > 20)
urchin_sub2
```

```
# # A data.trame: [133 × 3]
#    origin  height skeleton
#    <fct>    <dbl>    <dbl>
#  1 Fishery   21.6     9.31
#  2 Fishery   22.8    10.4
#  3 Fishery   21.2    10.3
#  4 Fishery   23.1    10.6
#  5 Fishery   23.3    11.6
#  6 Fishery   20.3     9.54
#  7 Fishery   20.6     9.4
#  8 Fishery   23.2    13.0
#  9 Fishery   24.5    13.3
# 10 Fishery   23.6    11.0
# # ℹ 123 more rows
```

Vous pouvez combiner différentes comparaisons avec les opérateurs “et” (`&`) et “ou” (`|`) :

```
# Oursins plus hauts que 20 mm ET issus d'élevage ("Farm")
urchin_sub3 <- filter_(urchin2, ~height > 20 & origin == "Farm")
urchin_sub3
```

```
# # A data.trame: [77 × 3]
#    origin height skeleton
#    <fct>   <dbl>    <dbl>
#  1 Farm     26.3    14.6
#  2 Farm     25.9    16.1
#  3 Farm     24.5    15.7
#  4 Farm     28.8    16.6
#  5 Farm     31.2    NA
#  6 Farm     28.6    NA
#  7 Farm     27.2    NA
#  8 Farm     20.9     6.83
#  9 Farm     20.8     7.59
# 10 Farm     21.5    NA
# # ℹ 67 more rows
```

Avec des variables facteurs composées de nombreux niveaux comme on peut en retrouver dans le jeu de données `zooplankton` du package {data.io}, vous pouvez être amené à sélectionner plusieurs niveaux. L'opérateur `%in%` permet d'indiquer que nous souhaitons garder tous les niveaux qui sont dans une liste. Il n'existe pas d'opérateur `%not_in%`, mais il suffit d'inverser le résultat en précédent l'instruction de `!` pour obtenir cet effet. Par exemple, `!letters %in% c("a", "d", "f")` conserve toutes les lettres *sauf* a, d et f. L'opérateur `!` est d'ailleurs utilisable avec toutes les comparaisons pour en inverser les effets. Ainsi, `!x == 1` est équivalent à `x != 1`.

```
zooplankton <- read("zooplankton", package = "data.io", lang  = "FR")
# Garde uniquement les copépodes (correspondant à 4 groupes distincts)
copepoda <- filter_(zooplankton,
  ~class %in% c("Calanoïde", "Cyclopoïde",  "Harpacticoïde", "Poecilostomatoïd
select_(copepoda, ~ecd:perimeter, ~class)
```

```
# # A data.trame: [535 × 4]
#      ecd  area perimeter class
#    <dbl> <dbl>     <dbl> <fct>
#  1 0.770 0.465      4.45 Poecilostomatoïde
#  2 0.815 0.521      4.15 Calanoïde
#  3 0.785 0.484      4.44 Poecilostomatoïde
#  4 0.361 0.103      1.71 Harpacticoïde
#  5 0.832 0.544      5.27 Poecilostomatoïde
#  6 1.23  1.20      15.7  Calanoïde
#  7 0.620 0.302      3.98 Poecilostomatoïde
#  8 1.19  1.12      15.3  Calanoïde
#  9 1.04  0.856      7.60 Calanoïde
# 10 0.725 0.412      7.14 Calanoïde
# # ℹ 525 more rows
```

Enfin, la détection et l'élimination de lignes contenant des valeurs manquantes (encodées comme `NA`) est spéciale. En effet, vous ne pouvez pas écrire quelque chose comme `x == NA` car ceci se lit comme "x est égale à … je ne sais pas quoi", ce qui renvoie à son tour `NA` pour toutes les comparaisons quelles qu'elles soient. Vous pouvez utiliser la fonction spécialement prévue pour ce test `is.na()`. Ainsi, `is.na(x)` effectue en réalité ce que vous voulez. Il peut être utilisé à l'intérieur de `filter()` ou `filter_()`. Par exemple pour conserver toutes les lignes n'ayant pas de valeurs manquantes pour `skeleton`, vous écrirez `filter_(urchin2, ~!is.na(skeleton))`. Vous noterez tout de même que ce n'est pas des plus lisibles. Il existe une fonction spécialement prévue pour débarrasser les tableaux des lignes contenant des valeurs manquantes : `drop_na()` ou `drop_na_()`. Si vous spécifier des noms de colonnes (facultatifs), la fonction ira rechercher les valeurs manquantes uniquement dans ces colonnes-là, sinon, elle scrutera tout le tableau (mais faites très attention à ne pas utiliser inconsidérément cette fonction sans spécifier les colonnes : éliminer des individus sur base de valeurs manquantes dans des colonnes que vous n'utilisez pas ensuite est idiot).

```
urchin_sub4 <- drop_na_(urchin, ~skeleton, ~sex)
nrow(urchin)
```

```
# [1] 421
```

```
nrow(urchin_sub4) # 287 lignes éliminées tout de même !
```

```
# [1] 134
```

Pour terminer le filtrage des lignes de vos tableaux, il est utile d'être aussi capable de le faire en R de base. Nous avons déjà étudié l'opérateur `[` qui extrait un sous-ensemble d'un **data.frame** et nous avons dit que son premier argument sélectionne les lignes. Nous pouvons y indiquer aussi un vecteur de valeurs logiques issues d'une comparaison. Ainsi, l'équivalent en R de base de `urchin_sub2 <- filter_(urchin2, ~height > 20)` est :

```
urchin_sub2 <- urchin2[urchin2$height > 20, ]
head(urchin_sub2)
```

```
# # A data.trame: [6 × 3]
#   origin  height skeleton
#   <fct>    <dbl>    <dbl>
# 1 Fishery   21.6     9.31
# 2 Fishery   22.8    10.4
# 3 Fishery   21.2    10.3
# 4 Fishery   23.1    10.6
# 5 Fishery   23.3    11.6
# 6 Fishery   20.3     9.54
```

Notez la différence importante : en R de base, les variables du jeu de données `urchin2` ne sont pas accessibles juste avec leur nom. Il faut spécifier de manière complète “la variable `height` du jeu de données `urchin2` ” qui s’écrit `urchin2$height` , sans quoi R vous dira qu’il ne connait pas l’objet `height` . Avec `filter()` ou `filter_()` , vous pouvez utiliser directement le nom de la variable, sans spécifier le nom du jeu de données qui est connu car c’est le premier argument de la fonction.

Enfin, l’opérateur `[` permet de sélectionner des lignes **et** des colonnes en une seule opération, là ou les fonctions tidyverse ou svTidy nécessitent un appel de `filter()` / `filter_()` suivi d’un appel de `select()` / `select_()` . Donc, à partir du tableau complet `urchin` , pour sélectionner les six première lignes et les trois premières colonnes vous ferez en R de base :

```
urchin_sub5 <- urchin[1:6, 1:3]
urchin_sub5
```

```
# # A data.trame: [6 × 3]
#   origin  diameter1 diameter2
#   <fct>       <dbl>     <dbl>
# 1 Fishery       9.9      10.2
# 2 Fishery      10.5      10.6
# 3 Fishery      10.8      10.8
# 4 Fishery       9.6       9.3
# 5 Fishery      10.4      10.7
# 6 Fishery      10.5      11.1
```

Dans ce cas-ci, R de base est bien plus concis que les fonctions tidyverse ou svTidy. Pour obtenir la même chose, vous devez écrire :

```
filter_(urchin, 1:6) %>.%
  select_(., 1:3) ->
  urchin_sub5
urchin_sub5
```

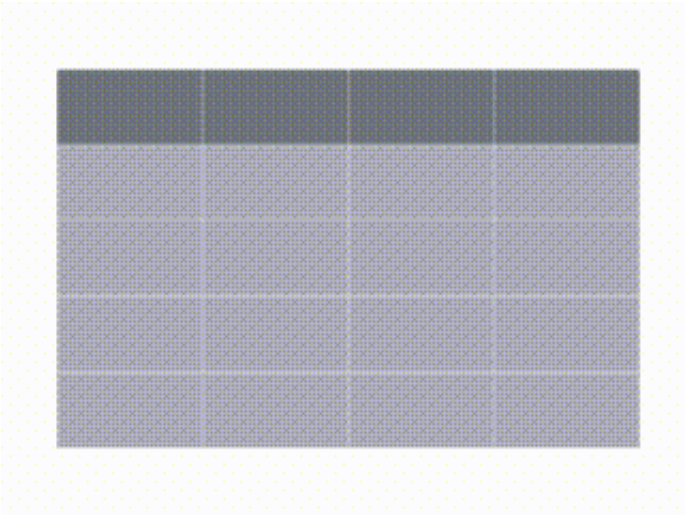
```
# # A data.trame: [6 × 3]
#   origin  diameter1 diameter2
#   <fct>       <dbl>     <dbl>
# 1 Fishery       9.9      10.2
# 2 Fishery      10.5      10.6
# 3 Fishery      10.8      10.8
# 4 Fishery       9.6       9.3
# 5 Fishery      10.4      10.7
# 6 Fishery      10.5      11.1
```

À vous de jouer !

Complétez les blancs dans cette fonction afin de garder les oursins de ferme (cette information est disponible dans la variable `origin` à deux niveaux : "Farm", "Fishery") présentant un diamètre (`diameter1`) supérieur ou égal à 40mm. Nommez le nouveau jeu de données `urchin_diam`.

[ ] <- filter_(urchin, ~ [ ] == [ ] , ~ diameter1 [ ] [ ])

4.4.3 `mutate()` / `mutate_()`



La fonction tidyverse `mutate()` permet de calculer de nouvelles variables (si le nom fourni n'existe pas encore dans le jeu de donnée) ou écrase les variables existantes de même nom. La fonction svTidy équivalente est `mutate_()`. Comme pour les autres fonctions, vous utiliserez une formule avec la version svTidy et une expression R classique (donc sans le tilde `~`) avec la version tidyverse. Repartons du jeu de données `urchin`. Pour calculer de nouvelles variables, vous pouvez employer :

- les opérateurs arithmétiques :
  - addition : `+`
  - soustraction : `-`
  - multiplication : `*`
  - division : `/`
  - exposant : `^`
  - modulo (reste lors d'une division entière) : `%%`
  - division entière : `%/%`

```
urchin <- mutate_(urchin,
  sum_skel  =  ~lantern + spines + test,
  ratio     = ~sum_skel / skeleton,
  skeleton2 = ~skeleton^2)
select_(urchin, ~skeleton:spines, ~sum_skel:skeleton2)
```

```
# # A data.trame: [421 × 7]
#    skeleton lantern    test  spines sum_skel ratio skeleton2
#       <dbl>   <dbl>   <dbl>   <dbl>    <dbl> <dbl>     <dbl>
#  1    0.179  0.0211  0.0587  0.0995    0.179  1.00   0.0321
#  2    0.188  0.0205  0.0622  0.105     0.188  1.00   0.0353
#  3    0.235  0.0254  0.0836  0.126     0.235  1.00   0.0554
#  4    0.063  0.0167  0.018   0.0283    0.063  1      0.00397
#  5   NA     NA      NA      NA        NA     NA     NA
#  6   NA     NA      NA      NA        NA     NA     NA
#  7   NA     NA      NA      NA        NA     NA     NA
#  8   NA     NA      NA      NA        NA     NA     NA
#  9   NA     NA      NA      NA        NA     NA     NA
# 10   NA     NA      NA      NA        NA     NA     NA
# # ℹ 411 more rows
```

- les fonctions mathématiques :
  - `ln()` ou `log()` (logarithme népérien), `lg()` ou `log10()` (logarithme en base 10)
  - `ln1p()` ou `log1p()` (logarithme népérien de x + 1), ou `lg1p()` (logarithme en base 10 de x + 1)
  - `exp()` (exponentielle, e^x) et `expm1()` (e^x - 1)
  - `sqrt()` (racine carrée)
  - `sin()` , `cos()` , `tan()`
  - ...

```
urchin <- mutate_(urchin,
  skeleton_log  = ~log(skeleton),
  skeleton_sqrt = ~sqrt(skeleton))
select_(urchin, ~skeleton, ~skeleton_log, ~skeleton_sqrt)
```

```
# # A data.trame: [421 × 3]
#    skeleton skeleton_log skeleton_sqrt
#       <dbl>        <dbl>         <dbl>
#  1    0.179        -1.72         0.423
#  2    0.188        -1.67         0.434
#  3    0.235        -1.45         0.485
#  4    0.063        -2.76         0.251
#  5   NA           NA            NA
#  6   NA           NA            NA
#  7   NA           NA            NA
#  8   NA           NA            NA
#  9   NA           NA            NA
# 10   NA           NA            NA
# # ℹ 411 more rows
```

La fonction `transmute()` ou `transmute_()` effectue la même opération, mais en plus, elle laisse tomber les variables d'origine pour ne garder *que* les nouvelles variables calculées.

En R de base, le calcul d'une nouvelle variable se fait comme ceci :

```
urchin$log_skeleton <- log(urchin$skeleton)
select_(urchin, "skeleton", "log_skeleton")
```

```
# # A data.trame: [421 × 2]
#    skeleton log_skeleton
#       <dbl>        <dbl>
#  1    0.179        -1.72
#  2    0.188        -1.67
#  3    0.235        -1.45
#  4    0.063        -2.76
#  5   NA            NA
#  6   NA            NA
#  7   NA            NA
#  8   NA            NA
#  9   NA            NA
# 10   NA            NA
# # ℹ 411 more rows
```

Remarquez encore une fois que vous devez qualifier complètement le nom de la variable à calculer, en indiquant le nom du jeu de données et le nom de la variable comme `urchin$skeleton`. Si vous voulez éliminer une variable du jeu de données en R de base, vous lui assignez `NULL` :

```
urchin$log_skeleton <- NULL # Efface la variable log_skeleton dans urchin
```

Selon le contexte, la version tidyverse/svTidy ou la version R de base sera la plus pratique.

À vous de jouer !

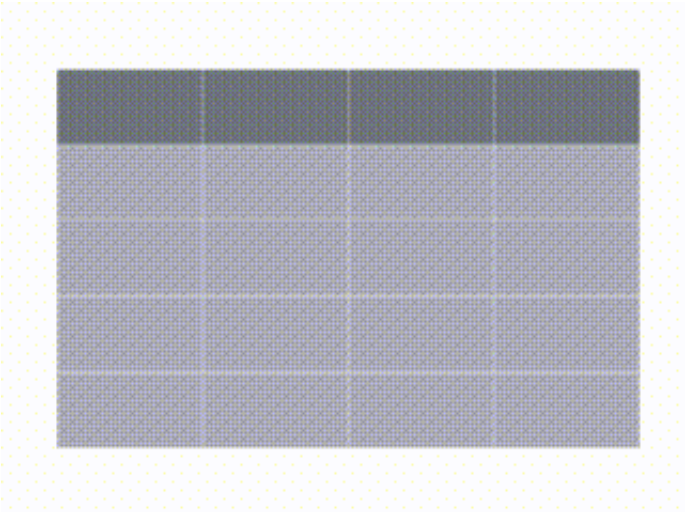
H5P

Complétez les blancs dans cette instruction R pour calculer la variable `diameter2` dans le jeu de données `urchin` dont l'expression est `diameter^2`

urchin <- mutate_(, ~ =)

Vérifier

4.4.4 group_by() / group_by_()



La fonction tidyverse `group_by()` ou la fonction svTidy `group_by_()` ne change rien dans le tableau lui-même, mais ajoute une annotation qui indique que les calculs ultérieurs devront être effectués sur des sous-ensembles du tableau en parallèle. Ceci est surtout utile avec `summarise() / summarise_()` (voir ci-dessous), mais aussi avec `mutate() / mutate_()` pour faire des transformations par groupes ou `filter() / filter_()` pour des sélections par groupe. Pour annuler le regroupement, il suffit d'utiliser `ungroup() / ungroup_()` .

À noter que toutes les fonctions qui collectent les résultats, à savoir `as_dtx()` , `collect_dtx()` et les assignations alternatives `%<-%` et `%->%` dégroupent également automatiquement les données. Ceci est voulu afin d'éviter que l'on oublie plus tard que l'objet a un regroupement enregistré et que vous ne réalisiez par la suite des calculs non voulus (par groupes alors que vous pensez travailler sur l'ensemble des données en une seule fois). **Il faut toujours préciser les groupes aussi proche que possible de leur utilisation, et dégrouper éventuellement à la fin lorsqu'on n'en a plus besoin.**

```
urchin_by_orig <- group_by_(urchin, "origin")
urchin_by_orig
```

```
# # A data.trame: [421 × 24]
#    origin  diameter1 diameter2 height buoyant_weight weight solid_parts
#    <fct>       <dbl>     <dbl>  <dbl>          <dbl>  <dbl>       <dbl>
#  1 Fishery       9.9      10.2    5               NA  0.522       0.478
#  2 Fishery      10.5      10.6    5.7             NA  0.642       0.589
#  3 Fishery      10.8      10.8    5.2             NA  0.734       0.677
#  4 Fishery       9.6       9.3    4.6             NA  0.370       0.344
#  5 Fishery      10.4      10.7    4.8             NA  0.610       0.559
#  6 Fishery      10.5      11.1    5               NA  0.610       0.551
#  7 Fishery      11        11      5.2             NA  0.672       0.605
#  8 Fishery      11.1      11.2    5.7             NA  0.703       0.628
#  9 Fishery       9.4       9.2    4.6             NA  0.413       0.375
# 10 Fishery      10.1       9.5    4.7             NA  0.449       0.398
# # ℹ 411 more rows
# # ℹ 17 more variables: integuments <dbl>, dry_integuments <dbl>,
# #   digestive_tract <dbl>, dry_digestive_tract <dbl>, gonads <dbl>,
# #   dry_gonads <dbl>, skeleton <dbl>, lantern <dbl>, test <dbl>, spines <dbl
# #   maturity <int>, sex <fct>, sum_skel <dbl>, ratio <dbl>, skeleton2 <dbl>,
# #   skeleton_log <dbl>, skeleton_sqrt <dbl>
#
# Grouped by:  origin  [2 | 210 (10.6) 203–218]
```

Noter la ligne ci-dessus : `Grouped by: origin` qui indique quels regroupements sont actifs dans le tableau. Il est aussi possible d'utiliser l'interface formule ici avec `~origin` :

```
urchin_by_orig <- group_by_(urchin, ~origin)
urchin_by_orig
```

```
# # A data.trame: [421 × 24]
#    origin  diameter1 diameter2 height buoyant_weight weight solid_parts
#    <fct>       <dbl>     <dbl>  <dbl>          <dbl>  <dbl>       <dbl>
#  1 Fishery       9.9      10.2    5               NA  0.522       0.478
#  2 Fishery      10.5      10.6    5.7             NA  0.642       0.589
#  3 Fishery      10.8      10.8    5.2             NA  0.734       0.677
#  4 Fishery       9.6       9.3    4.6             NA  0.370       0.344
#  5 Fishery      10.4      10.7    4.8             NA  0.610       0.559
#  6 Fishery      10.5      11.1    5               NA  0.610       0.551
#  7 Fishery      11        11      5.2             NA  0.672       0.605
#  8 Fishery      11.1      11.2    5.7             NA  0.703       0.628
#  9 Fishery       9.4       9.2    4.6             NA  0.413       0.375
# 10 Fishery      10.1       9.5    4.7             NA  0.449       0.398
# # ℹ 411 more rows
# # ℹ 17 more variables: integuments <dbl>, dry_integuments <dbl>,
# #   digestive_tract <dbl>, dry_digestive_tract <dbl>, gonads <dbl>,
# #   dry_gonads <dbl>, skeleton <dbl>, lantern <dbl>, test <dbl>, spines <dbl
# #   maturity <int>, sex <fct>, sum_skel <dbl>, ratio <dbl>, skeleton2 <dbl>,
# #   skeleton_log <dbl>, skeleton_sqrt <dbl>
#
# Grouped by:  origin  [2 | 210 (10.6) 203–218]
```

Le dégroupement est automatique avec certaines fonctions (sinon, utilisez `ungroup()` / `ungroup_()` pour le forcer) :

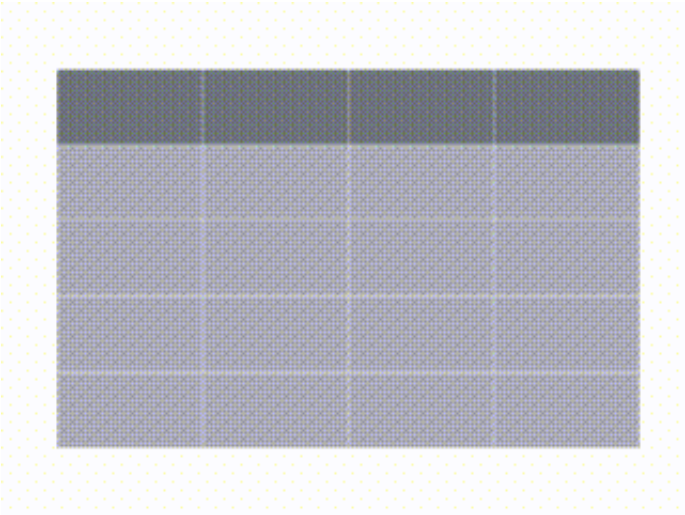
```
# collect_dtx() ou %<-% / %->% dégroupent automatiquement
urchin_collected %<-% urchin_by_orig
urchin_collected
```

```
# # A data.trame: [421 × 24]
#    origin  diameter1 diameter2 height buoyant_weight weight solid_parts
#    <fct>       <dbl>     <dbl>  <dbl>          <dbl>  <dbl>       <dbl>
#  1 Fishery       9.9      10.2    5               NA  0.522       0.478
#  2 Fishery      10.5      10.6    5.7             NA  0.642       0.589
#  3 Fishery      10.8      10.8    5.2             NA  0.734       0.677
#  4 Fishery       9.6       9.3    4.6             NA  0.370       0.344
#  5 Fishery      10.4      10.7    4.8             NA  0.610       0.559
#  6 Fishery      10.5      11.1    5               NA  0.610       0.551
#  7 Fishery      11        11      5.2             NA  0.672       0.605
#  8 Fishery      11.1      11.2    5.7             NA  0.703       0.628
#  9 Fishery       9.4       9.2    4.6             NA  0.413       0.375
# 10 Fishery      10.1       9.5    4.7             NA  0.449       0.398
# # ℹ 411 more rows
# # ℹ 17 more variables: integuments <dbl>, dry_integuments <dbl>,
# #   digestive_tract <dbl>, dry_digestive_tract <dbl>, gonads <dbl>,
# #   dry_gonads <dbl>, skeleton <dbl>, lantern <dbl>, test <dbl>, spines <dbl
# #   maturity <int>, sex <fct>, sum_skel <dbl>, ratio <dbl>, skeleton2 <dbl>,
# #   skeleton_log <dbl>, skeleton_sqrt <dbl>
```

```
# Excepté pour les commentaires, ces deux objets sont identiques
comment(urchin_collected) <- comment(urchin)
identical(urchin_collected, urchin)
```

```
# [1] TRUE
```

4.4.5 summarise() / summarise_()



Si vous voulez résumer vos données (calcul de la moyenne, médiane, etc.), vous pouvez réaliser ceci sur une variable en particulier avec les fonctions dédiées. Par exemple `mean(urchin$skeleton)` renvoie la masse moyenne de squelette pour tous les oursins (ce calcul donne `NA` dès qu'il y a des valeurs manquantes, mais l'argument `na.rm = TRUE` permet d'obtenir un résultat en ne tenant pas compte de ces données manquantes : `mean(urchin$skeleton, na.rm = TRUE)`). Cela devient vite laborieux s'il faut réitérer ce genre de calcul sur plusieurs variables du jeu de données, et assembler ensuite les résultats dans un petit tableau synthétique. D'autant plus, s'il faut séparer d'abord le jeu de données en sous-groupes pour faire ces calculs. La fonction tidyverse `summarise()` , ou son équivalent svTidy `summarise_()` reporte automatiquement ces calculs, en tenant compte des regroupements indiqués auparavant à l'aide de `group_by()` / `group_by_()` .

Si vous utilisez les fonctions svTidy, il est conseillé d'utiliser les fonction "fstat" (commençant par un "f") pour les calculs des moyennes, médianes, écarts types… Donc `fmean()` , `fmedian()` , `fsd()` … La combinaison des deux permet des accélérations substantielles des calculs. Cela ne vous apparaîtra pas avec les petits jeux de données utilisés au cours, mais lorsque vous traiterez des gros tableaux, le gain de vitesse peut être de dix fois, voire plus encore.

```
tooth <- read("ToothGrowth", package = "datasets", lang = "fr")
tooth_summary <- summarise_(tooth,
  "Allongement moyen"   = ~fmean(len),
  "Allongement minimal" = ~fmin(len),
  "Allongement médian"  = ~fmedian(len),
  "Allongement maximal" = ~fmax(len))
# Utiliser soit kable(), soit tabularise() pour afficher un tableau
#knitr::kable(tooth_summary, digits = 2,
#  caption = "Allongement des dents chez des cochons d'Inde recevant de l'acid
tabularise(tooth_summary)
```

Allongement des dents	Allongement des dents	Allongement des dents	Allongement des dents
18.8	4.2	19.2	33.9

Vous noterez que toutes le colonnes du tableau s'appellent "Allongement des dents". C'est le label de la variable `len`. Ce n'est pas très pratique. Vous pouvez éviter ce problème en enlevant le label avec `unlabelise()` avant de faire le `summarise()` :

```
tooth_summary <- summarise_(unlabelise(tooth),
  "Allongement moyen"   = ~fmean(len),
  "Allongement minimal" = ~fmin(len),
  "Allongement médian"  = ~fmedian(len),
  "Allongement maximal" = ~fmax(len))
tabularise(tooth_summary)
```

Allongement moyen	Allongement minimal	Allongement médian	Allongement maximal
18.8	4.2	19.2	33.9

Voici les mêmes calculs, mais effectués séparément pour les deux types de supplémentations alimentaires. Pour ce faire, nous allons procéder en deux étapes : un `group_by_()` suivi d'un `summarise_()`. Pour ces étapes successives, nous utiliserons

aussi une présentation particulière, dite “liste à puce”, ou *“bullet point”* en anglais, avec le pseudo-opérateur `.=` que nous détaillerons plus bas :

```
tooth_summary2 <- {
  .= unlabelise(tooth, "len") # Enlever seulement le label pour len
  .= group_by_(., ~supp)
  .= summarise_(
    "Allongement moyen"   = ~fmean(len),
    "Allongement minimal" = ~fmin(len),
    "Allongement médian"  = ~fmedian(len),
    "Allongement maximal" = ~fmax(len))
}
tabularise(tooth_summary2)
```

Supplément	Allongement moyen	Allongement minimal	Allongement médian	Allongement maximal
OJ	20.7	8.2	22.7	30.9
VC	17.0	4.2	16.5	33.9

Pièges et astuces

- Tout comme lors de réalisation d’une boite à moustaches, vous devez être particulièrement vigilant au nombre d’observation par sous-groupe. Pensez toujours à ajoutez à chaque tableau de résumé des données, le nombre d’observations par sous-groupe grâce à la fonction `fnobs()` qui comptabilise le nombre d’observations non manquantes.

```
tooth_summary2 <- {
  .= unlabelise(tooth, "len") # Enlever seulement le label pour len
  .= group_by_(., ~supp)
  .= summarise_(
    "Allongement moyen"     = ~fmean(len),
    "Allongement minimal"   = ~fmin(len),
    "Allongement médian"    = ~fmedian(len),
    "Allongement maximal"   = ~fmax(len),
    "Nombre d'observations" = ~fnobs(len))
}
tabularise(tooth_summary2)
```

Supplément	Allongement moyen	Allongement minimal	Allongement médian	Allongement maximal	Nombre d'observations
OJ	20.7	8.2	22.7	30.9	30
VC	17.0	4.2	16.5	33.9	30

Si vous souhaitez connaitre le nombre d'observations **totales** dans votre jeu de données, utilisez `fn()` au lieu de `fnobs()`. Vous pouvez naturellement utiliser les deux simultanément.

```
urchin_skeleton_summary <- {
  .= unlabelise(urchin, "skeleton")
  .= group_by_(., ~origin)
  .= summarise_(
    "Masse squelettique moyenne"  = ~fmean(skeleton),
    "Observations totales"        = ~fn(skeleton),
    "Observations non manquantes" = ~fnobs(skeleton))
}
tabularise(urchin_skeleton_summary)
```

Origine	Masse squelettique moyenne	Observations totales	Observations non manquantes

Farm	6.88	218	122
Fishery	7.21	203	136

Dans le cas présent, le nombre d'observations utilisable est nettement inférieur au nombre total de cas dans le tableau. Cette distinction entre `fn()` et `fnobs()` est donc très importante. **Ne la perdez pas de vue.** De plus, si vous n'utilisez pas `na.rm = TRUE` dans les fonctions comme `mean()` / `fmean()`, `median()` / `fmedian()` ... vous aurez des `NA` partout dans votre tableau résumé. D'un autre côté, soyez bien conscient qu'avec `na.rm = TRUE`, c'est le nombre d'observations hors valeurs manquantes qui est utilisé pour ces calculs !

Les fonctions `fn()` et `fnobs()` ne sont pas définies dans le tidyverse. Ce sont des fonctions d'une autre famille appelée dans `SciViews::R` des fonctions "fstat". Dans tidyverse, on vous fera utiliser la fonction `n()` sans argument qui est l'équivalent de `fn()`. Il n'y a pas d'équivalent direct de `fnobs()` et il faut ruser avec du code comme `sum(!is.na(skeleton))` pour obtenir ce résultat, ce qui n'est pas pratique. **La fonction `n()` n'est pas utilisable avec les fonctions svTidy, et d'une manière générale, préférez-lui `fn()` qui est plus logique et qui fonctionne partout.**

- `summarise()` calcule ses variables dans le même environnement que le tableau de départ. Donc, si vous utiliser des noms de colonnes qui existent déjà, elles seraient écrasées par le résultat du calcul. Avec un **data.trame**, ceci n'est pas permis (le code fonctionnera pourtant dans un autre contexte avec un **data.frame**, c'est pourquoi nous convertissons d'abord `tooth` avec `as_dtf()`). Voici un exemple concret :

```
tooth_summary3 <- summarise(unlabelise(tooth),
  "len" = mean(len), # Notez le même nom à gauche et à droite du = (len)
  "len_sd" = sd(len))
knitr::kable(tooth_summary3, digits = 2,
  caption = "Exemple de résumé des données erroné à cause de l'écrasement de l
```

Tableau 4.6: Exemple de résumé des données erroné à cause de l'écrasement de la variable `len`.

len	len_sd
18.81	NA

```
#tabularise(tooth_summary3, auto.labs = FALSE)
```

L'écart type est… `NA` ??? Pour comprendre ce qui s'est passé, il faut lire la transformation réalisée par `summarise()` ligne après ligne :

- la moyenne de la variable `len` est placée dans … `len`. Donc ici, **nous écrasons la variable initiale** de 60 observations par un nombre unique : la moyenne,
- l'écart type de `len` est ensuite calculé. Attendez une minute, de quel `len` s'agit-il ici ? Et bien la dernière calculée, soit celle qui contient **une seule valeur**, la moyenne. Or, `sd()` nécessite au moins **deux** valeurs pour que l'écart type puisse être calculé, sinon, `NA` est renvoyé. C'est encore heureux ici, car nous aurions pu faire un calcul qui renvoie un résultat, … mais qui n'est pas celui qu'on espère !

Conclusion : ne nommez **jamais** vos variables créées avec `summarise()` exactement comme les variables de votre tableau en entrée.

Toutefois, `summarise_()` fonctionne différemment et n'a pas de problèmes dans ce cas-là :

```
tooth_summary4 <- summarise_(unlabelise(tooth),
  "len" = ~fmean(len), # Notez le même nom à gauche et à droite du = (len)
  "len_sd" = ~fsd(len))
knitr::kable(tooth_summary4, digits = 2,
  caption = "Exemple de résumé des données correct avec `summarise_()`.")
```

Tableau 4.7: Exemple de résumé des données correct avec `summarise_()`.

len	len_sd
18.81	7.65

Faites bien attention : les couples de fonctions comme `select()` / `select_()`, `mutate()` / `mutate_()` ou `summarise()` / `summarise_()` se ressemblent très fort au niveau de leur usage. On pourrait donc être tenté de croire qu'elles sont parfaitement interchangeables. La plupart du temps, c'est le cas. Cependant, elles fonctionnement radicalement différemment en interne. Des différences existent, sont connues, et ne sont *pas* des bugs. Vérifiez toujours votre code !

4.4.5.1 Résumé avec les fonctions "fstat"

R propose de base une série de fonctions qui calculent des descripteurs statistiques classiques tels que la moyenne `mean()`, la médiane `median()`, la variance `var()`, etc. Toutes ces fonctions ont un argument `na.rm` qui prend la valeur `FALSE` par défaut. S'il y a une valeur manquante ou plus, le calcul renverra alors `NA`.

```
vec <- c(5, 3, 8, NA, 4)
mean(vec)
```

```
# [1] NA
```

Vous devez indiquer `na.rm = TRUE` si vous voulez quand même calculer la moyenne sur ce vecteur `vec` en utilisant uniquement les observations (non manquantes) :

```
mean(vec, na.rm = TRUE)
```

```
# [1] 5
```

Ces fonctions ne sont pas prévues pour travailler sur des **tableaux entiers**.

```
mean(iris)


# Warning in mean.default(iris): argument is not numeric or logical: returning


# [1] NA
```

La fonction `mean()` (et les autres fonctions équivalentes) ne sont pas prévues pour travailler sur ce genre d'objet qui contient plusieurs colonnes. C'est pour cela que vous devez utiliser une construction plus complexe avec `summarise()` ou `summarise_()`. Si vous voulez résumer les quatre variables numériques par la moyenne en fonction de `species`, vous devez faire (rappelez-vous que vous devez aussi utiliser d'autres noms que les noms de variables dans le tableau de départ) :

```
summarise_(group_by_(iris, ~species),
  mean_sepal_length = ~mean(sepal_length, na.rm = TRUE),
  mean_sepal_width  = ~mean(sepal_width, na.rm = TRUE),
  mean_petal_length = ~mean(petal_length, na.rm = TRUE),
  mean_petal_width  = ~mean(petal_width, na.rm = TRUE))


# # A data.trame: [3 × 5]
#   species  mean_sepal_length mean_sepal_width mean_petal_length mean_petal_w
#   <fct>                <dbl>            <dbl>             <dbl>           <
# 1 setosa                5.01             3.43              1.46           0
# 2 versico…              5.94             2.77              4.26           1
# 3 virgini…              6.59             2.97              5.55           2
```

Tidyverse offre une astuce pour éviter de se répéter. Si vous devez appliquer la *même* fonction sur plusieurs variables, vous pouvez l'indiquer avec `across()` (aussi avec les fonctions svTidy) comme ceci :

```
summarise_(group_by_(iris, ~species),
  ~across(sepal_length:petal_width, fmean, na.rm = TRUE))


# # A data.trame: [3 × 5]
#   species    sepal_length sepal_width petal_length petal_width
#   <fct>             <dbl>       <dbl>        <dbl>       <dbl>
# 1 setosa             5.01        3.43         1.46       0.246
# 2 versicolor         5.94        2.77         4.26       1.33
# 3 virginica          6.59        2.97         5.55       2.03
```

C'est déjà mieux, même si c'est moins lisible quant à ce que l'on veut obtenir comme tableau final. Dans le package {collapse}, et dans `SciViews::R` qui utilise ce package, il y a des fonctions de remplacement de `mean()`, `median()` ... qui offrent plusieurs avantages. Ce sont les fonctions "fstat" que nous avons déjà utilisées. Nous allons maintenant détailler un peu plus ces fonctions.

Souvent, vous pourrez utiliser `fmean()` à la place de `mean()`, mais son comportement et ses possibilités sont très différentes :

- elle est plus rapide (appréciable pour les gros jeux de données), en fait "f", c'est pour fast !
- la valeur par défaut pour son argument `na.rm` est `TRUE`. Vous ne devez donc pas préciser `na.rm = TRUE` à tout bout de champ (mais restez bien attentif aux valeurs manquantes dans vos données !)
- elle fonctionne aussi sur des tableaux de données ne contenant que des variables numériques
- elle peut utiliser les regroupements effectués à l'aide de `group_by_()` ou `group_by()` et en tient compte à condition que toutes les autres variables non concernées par le regroupement soient numériques

Illustrons tout ceci :

```
fmean(vec) # Pas besoin de préciser na.rm = TRUE, c'est la valeur pas défaut
```

```
# [1] 5
```

Application directe sur un tableau, même avec regroupement :

```
fmean(group_by_(iris, ~species))
```

```
# # A data.trame: [3 × 5]
#   species    sepal_length sepal_width petal_length petal_width
#   <fct>             <dbl>       <dbl>        <dbl>       <dbl>
# 1 setosa             5.01        3.43         1.46       0.246
# 2 versicolor         5.94        2.77         4.26       1.33
# 3 virginica          6.59        2.97         5.55       2.03
```

Le tableau obtenu est très similaire à celui que nous avions calculé à l'aide de `summarise_()` plus haut (à part que le nom des variables est conservé) et identique à celui obtenu en utilisant `across()`. Le calcul est toutefois bien plus rapide (vous ne devez pas comprendre le code ci-dessous, juste considérer que l'on détermine les performances de trois versions du même calcul : "tidyverse", "svTidy" et "fstat") :

```
iris2 <- datasets::iris
names(iris2) <- c("sepal_length", "sepal_width", "petal_length", "petal_width"
bench::mark(check = FALSE,
  tidyverse = summarise(group_by(iris2, species),
    across(sepal_length:petal_width, mean, na.rm = TRUE)),
  svTidy    = summarise_(group_by_(iris2, ~species),
    ~across(sepal_length:petal_width, fmean, na.rm = TRUE)),
  fstat     = fmean(group_by_(iris2, ~species)))
```

```
# Warning: There was 1 warning in `summarise()`.
# ℹ In argument: `across(sepal_length:petal_width, mean, na.rm = TRUE)`.
# ℹ In group 1: `species = setosa`.
# Caused by warning:
# ! The `...` argument of `across()` is deprecated as of dplyr 1.1.0.
# Supply arguments directly to `.fns` through an anonymous function instead.
#
#   # Previously
#   across(a:b, mean, na.rm = TRUE)
#
#   # Now
#   across(a:b, \(x) mean(x, na.rm = TRUE))


# # A tibble: 3 × 6
#   expression      min   median `itr/sec` mem_alloc `gc/sec`
#   <bch:expr> <bch:tm> <bch:tm>     <dbl> <bch:byt>    <dbl>
# 1 tidyverse     2.1ms    2.2ms      450.    2.94MB     20.1
# 2 svTidy       56.7µs     65µs    15343.    1.27KB     20.9
# 3 fstat        29.2µs   33.7µs    29896.    1.27KB     20.9
```

Examinez les colonnes **median** qui est le temps median et **mem_alloc** qui est la mémoire vive nécessaires pour faire ces calculs.

- Il a fallu 68 fois moins de temps à `fmean()` qu'à la version tidyverse avec `summarise()` et deux fois moins de temps que la version svTidy avec `summarise_()`.
- Concernant l'utilisation de la mémoire vive, il a fallu 27 fois moins de mémoire vive à `fmean()` et `summarise_()` qu'à `summarise`.

Par rapport aux petits jeux de données que nous utilisons dans le cadre de ce cours, la différence n'est pas très visible. Mais si vous travaillez plus tard avec des bien plus gros jeux de données, pensez à utiliser les fonctions svTidy, ou mieux les fonctions

fstat directement si vous le pouvez.

Enfin, vous pouvez lister toutes les fonctions fstat disponibles comme ceci :

```
list_fstat_functions()
```

```
#  [1] "ffirst"     "flast"      "fmax"       "fmean"      "fmedian"
#  [6] "fmin"       "fmode"      "fndistinct" "fnobs"      "fn"
# [11] "fna"        "fnth"       "fprod"      "fsd"        "fsum"
# [16] "fvar"
```

Notez bien aussi les fonction `fn()` et `fnobs()` que nous avons vues pour énumérer les cas et les observations effectivement réalisées hors valeurs manquantes. `fna` compte les valeurs manquantes et est donc complémentaire à `fnobs()` (`fna(x) + fnobs(x) == fn(x)`) `fndistinct()` peut aussi être utile pour énumérer le nombre de valeurs différentes rencontrées (par exemple pour décider si cela est raisonnable de convertir directement une variable `numeric` ou `character` en `factor`). Vous devriez pouvoir déduire le rôle des autres fonctions via leur nom. `fvar()` calcule la variance alors que `fsd()` calcule l'écart type (*standard deviation* en anglais, d'où l'abréviation "sd") qui est la racine carrée de la variance.

Pour en savoir plus

- Le chapitre consacré à la transformation des données de **R for Data Science seconde edition** présente le remaniement d'un tableau de données à l'aide des fonctions tidyverse différemment et propose des exemples et exercices complémentaires très utiles.
- La meilleure façon de se familiariser avec les "verbes" du **tidyverse** est de réaliser des transformations de données par soi-même. En cas de blocage, le site https://stackoverflow.com ou https://phind.com permet de chercher des solutions. Pour une recherche ciblée sur le langage R, précédez vos mots clés par "[R]" (R entre crochets). Par exemple, pour explorer diverses utilisations de la fonction `mutate()` ,

vous entrerez le texte de recherche suivant: “[R] mutate”. Ne cherchez pas les fonctions svTidy ou fstat, vous ne trouverez pas grand chose par contre. Adaptez les exemples trouvés avec les fonctions tidyverse, si nécessaire.

- N’oubliez pas les aide-mémoires de {dplyr} et de {tidyr} qui forment aussi une source d’inspiration utile pour vous guider vers les fonction (les “verbes”) adéquats. Ensuite, allez voir l’aide en ligne de la fonction avec `?ma_fonction` .

24. Si vous utilisez un R de base, hors SciViews Box, vous devez utiliser l’instruction `install.packages("dplyr")` pour installer le package {dplyr}. Pour avoir les fonctions suffixées d’un `_` , il vous faut le package {svTidy} de SciViews, vous entrez : `install.packages('svTidy', repos = c('https://sciviews.r-universe.dev', 'https://cloud.r-project.org'))` . Ensuite, vous rendez les fonctions accessibles avec `library(dplyr)` ou `library(svTidy)` . Faites bien attention que les fonctions suffixées d’un `_` sont bien celles du package {svTidy}, et pas celles, dépréciées, de {dplyr} !↩
25. Voyez `?tidyselect::select_helpers` pour une panoplie de fonctions supplémentaires qui permettent une sélection “intelligente” des variables utilisables avec `select()` .↩
26. L’ordre alphabétique qui fait également intervenir les caractères accentués diffère en fonction de la configuration du système (langue). L’état du système tel que vu par R pour le tri alphabétique est obtenu par `Sys.getlocale("LC_COLLATE")` . Dans la SciViews Box, ceci est **toujours** `"en_US.UTF-8"` , ceci afin de rendre le traitement reproductible d’un PC à l’autre, qu’il soit en anglais, français, espagnol, chinois, ou n’importe quelle autre langue.↩

4.5 Chaînage des instructions

Le chaînage (ou *“pipe”* en anglais, prononcez “païpe”) permet de combiner une suite d’instructions R. Il permet une représentation facilement lisible et compréhensible d’un traitement décomposé en plusieurs étapes simples de remaniement des données.

Différents opérateurs de chaînage existent dans R. Le Tidyverse a introduit un opérateur de chaînage `%>%` issu du package {magrittr}. Si nous sommes sensibles au clin d’œil fait ici à un artiste belge bien connu (“ceci n’est pas un pipe”), nous n’adhérons pas à ce choix pour des raisons multiples et plutôt techniques qui n’ont pas leur place dans ce document[27]. Nous vous présentons ici l’un des opérateurs de chaînage du package {svFlow}[28] : `%>.%` . Le jeu de données sur la biométrie humaine est employé pour cette démonstration qui va comparer le remaniement d’un tableau de données avec et sans l’utilisation du chaînage.

```
biometry <- read("biometry", package = "BioDataScience", lang = "fr")
```

Vous vous intéressez à l’indice de masse corporelle ou IMC (*BMI* en anglais) des individus de moins de 25 ans. Vous souhaitez représenter la moyenne, la médiane et le nombre d’observations de manière séparée pour les hommes et les femmes. Pour obtenir ces résultats, vous devez :

- calculer le BMI,
- filtrer le tableau pour ne retenir que les individus de moins de 25 ans,
- résumer les données afin d’obtenir la moyenne et la médiane par genre,
- afficher un tableau de données avec ces résultats.

Il est très clair ici que le traitement peut être décomposé en étapes plus simples. Cela apparaît naturellement rien que dans la description de ce qui doit être fait. Sans l’utilisation de l’opérateur de chaînage, deux approches sont possibles :

- Imbriquer les instructions les unes dans les autres (très difficile à lire et à déboguer) :

```
knitr::kable(
  summarise_(
    group_by_(
      filter_(
        mutate_(biometry, bmi = ~weight / (height/100)^2),
        ~age <= 25),
      ~gender),
    mean   = ~fmean(bmi),
    median = ~fmedian(bmi),
    number = ~fn(bmi)),
  rows = NULL,  digits = 1,
  col.names = c("Genre", "Moyenne", "Médiane", "Observations"),
  caption = "IMC d'hommes (M) et femmes (W) de 25 ans maximum."
)
```

Tableau 4.8: IMC d'hommes (M) et femmes (W) de 25 ans maximum.

Genre	Moyenne	Médiane	Observations
M	22.3	22.1	97
W	21.8	21.0	94

- Passer par des variables intermédiaires (`biometry_25` et `biometry_tab`). Les instructions sont plus lisibles, mais les variables intermédiaires "polluent" inutilement l'environnement de travail (en tout cas, si elles ne servent plus par après) :

```
biometry <- mutate_(biometry, bmi = ~weight / (height/100)^2)
biometry_25 <- filter_(biometry, ~age <= 25)
biometry_25 <- group_by_(biometry_25, ~gender)
biometry_tab <- summarise_(biometry_25,
  mean   = ~fmean(bmi),
  median = ~fmedian(bmi),
  number = ~fn(bmi))
knitr::kable(biometry_tab, rows = NULL, digits = 1,
  col.names = c("Genre", "Moyenne", "Médiane", "Observations"),
  caption = "IMC d'hommes (M) et femmes (W) de 25 ans maximum.")
```

Tableau 4.9: IMC d'hommes (M) et femmes (W) de 25 ans maximum.

Genre	Moyenne	Médiane	Observations
M	22.3	22.1	97
W	21.8	21.0	94

- La version ci-dessous avec chaînage des opérations est plus lisible[29].

```
biometry %>.%
  mutate_(., bmi = ~weight / (height/100)^2) %>.%
  filter_(., ~age <= 25) %>.%
  group_by_(., ~gender) %>.%
  summarise_(.,
    mean   = ~fmean(bmi),
    median = ~fmedian(bmi),
    number = ~fn(bmi)) ->
  biometry_tab
  knitr::kable(biometry_tab, rows = NULL, digits = 1,
    col.names = c("Genre", "Moyenne", "Médiane", "Observations"),
    caption = "IMC d'hommes (M) et femmes (W) de 25 ans maximum.")
```

Tableau 4.10: IMC d'hommes (M) et femmes (W) de 25 ans maximum.

Genre	Moyenne	Médiane	Observations
M	22.3	22.1	97
W	21.8	21.0	94

Le pipe `%>.%` injecte le résultat précédent dans l'instruction suivante à travers l'objet `.` Ainsi, en seconde ligne `mutate_(.)`, `.` se réfère à `biometry`. A la ligne suivante, `filter_(.)`, le `.` se réfère au résultat issu de l'opération `mutate_()`, et ainsi de suite. La logique d'enchaînement des opérations sur le résultat, à chaque fois, du calcul précédent est donc le fondement de cet opérateur "pipe".

Le pipe permet d'éviter de répéter le nom des objets (version avec variables intermédiaires), ce qui alourdit inutilement le code et le rend moins agréable à la lecture. L'imbrication des fonctions dans la première version est catastrophique pour la compréhension du code car les arguments des fonctions de plus haut niveau sont repoussés loin. Par exemple, l'argument de l'appel à `group_by_()` (`gender`) se retrouve quatre lignes plus loin. Et encore, nous avons pris soin d'indenter le code pour repérer sur un plan vertical qui appartient à qui, mais imaginez ce que cela donne si l'instruction est mise à plat sur une seule ligne !

Nous vous proposons une quatrième façon de présenter votre code à l'aide de la forme "liste à puces" (*bullet point* en anglais). Le principe consiste à enchaîner les étapes en les précédant d'un pseudo-opérateur liste à puce `.=`. Cela donne ceci :

```
biometry_tab <- {
  .= biometry
  .= mutate_(., bmi = ~weight / (height/100)^2)
  .= filter_(., ~age <= 25)
  .= group_by_(., ~gender)
  .= summarise_(.,
      mean   = ~fmean(bmi),
      median = ~fmedian(bmi),
      number = ~fn(bmi))
}
knitr::kable(biometry_tab, rows = NULL, digits = 1,
  col.names = c("Genre", "Moyenne", "Médiane", "Observations"),
  caption = "IMC d'hommes (M) et femmes (W) de 25 ans maximum.")
```

Tableau 4.11: IMC d'hommes (M) et femmes (W) de 25 ans maximum.

Genre	Moyenne	Médiane	Observations
M	22.3	22.1	97
W	21.8	21.0	94

Cette présentation du code donne l'impression d'une liste à puces (les puces étant les `.=`), encadrées par des accolades `{ }`. Chaque item de la liste est indépendante des autres, contrairement à l'opérateur pipe qui lie les instructions les unes aux autres. Cela facilite le débogage du code.

Le code le plus clair à la lecture est celui avec chaînage des opérations, ou avec liste à puce (selon que vous soyez habitué à l'un ou à l'autre). Or, un code plus lisible est plus compréhensible… et donc, moins bogué.

Vous pouvez aussi assigner le résultat de la dernière étape de votre pipeline avec `%->%` ou le résultat de la liste à puce avec l'opérateur `%<-%`. Ces opérateurs d'assignation alternatifs à `->` ou `<-` vont dégrouper les données (s'il y avait un `group_by()` dans le pipeline ou la liste à puce). Cela évite des problèmes plus tard, si on oublie que le regroupement est là ! Il va aussi collecter les résultats du pipeline, c'est-à-dire effectuer

l'équivalent de `collect_dtx()`, en un tableau dans sa forme par défaut (classe **data.frame**, **data.trame** ou **tbl_bf**). Si vous n'avez pas modifié les options de `SciViews::R`, cette classe sera un **data.trame**. Enfin, indiquez bien le nom de l'objet final comme ici `biometry_tab` à la ligne pour le mettre en évidence et le placer comme dernière étape du pipeline ou première ligne de la liste à puce.

4.5.1 Opérateur pipe de base ou léger `|>`

L'opérateur de base de R à partir de sa version 4.1 est `|>`. Il est plus limité que `%>.%` et se contente d'injecter l'expression de gauche comme premier argument dans l'expression de droite. Donc, `x |> log(base = 3)` est strictement équivalent à `log(x, base = 3)`. Cet opérateur est donc cosmétique pour présenter du code plus complexe de manière plus lisible. Notez bien que contrairement à `%>.%`, vous ne devez pas et même **ne pouvez pas** préciser où l'expression de gauche est injectée à l'aide du point `.`. Ainsi, `x |> log(., base = 3)` sera une erreur.

En `SciViews::R` nous utilisons `%>.%` pour relier les différentes étapes d'un pipeline complexe et bien indiquer qu'il s'agit d'une étape suivante. Le `%>.%` peut se lire à haute voix "… et ensuite…". L'opérateur pipe léger `|>` s'utilise éventuellement au sein d'un même étape pour réarranger le code de manière plus lisible. Ceci est également vrai à l'intérieur d'une liste à puce.

À vous de jouer !

Effectuez maintenant les exercices du tutoriel **A04La_wrangling (Traitement des données)**.

```
BioDataScience1::run("A04La_wrangling")
```

Réalisez le travail **A04la_transformation**.

Initiez votre projet GitHub Classroom

Voyez les explications dans le fichier `README.md` *.*

Vous allez maintenant planifier la récolte et collecter des données relatives à l'étude de l'obésité afin de préparer le travail du module 5.

Réalisez en groupe le travail **A04Ga_biometry, partie I**.

Initiez votre projet GitHub Classroom

Voyez les explications dans le fichier `README.md` *, partie I.*

Effectuez également des remaniements de données dans votre projet de groupe récurrent.

Réalisez en groupe le travail **A02Ga_analysis, partie III**.

Initiez votre projet GitHub Classroom

Voyez les explications dans le fichier `README.md` *, partie III.*

Pour en savoir plus

- Présentation en détail du "dot-pipe" assez proche fonctionnellement de `%>.%` du package {svFlow}.
- Section sur le pipe dans "R for Data Science second edition" expliquant l'utilisation du pipe de R de base et de {magrittr}.

27. Le lecteur intéressé pourra lire les différents articles suivants : more pipes in R, y compris les liens qui s'y trouvent, permet de se faire une idée de la diversité des opérateurs de chaînage dans R et de leur historique. Dot pipe présente l'opérateur `%.>%` du package {wrapr} très proche du nôtre et in praise of syntactic sugar explique ses avantages. Nous partageons l'idée que le "pipe de base" ne devrait pas

modifier l'instruction de droite contrairement à ce que fait `%>%` de {magrittr}, et notre opérateur `%>.%` va en outre plus loin encore que `%.>%` dans la facilité de débogage du code chaîne.↩

28. Le pakage {svFlow} est préinstallé dans la SciViews Box. Si vous travaillez avec un R installé autrement, vous pouvez facilement ajouter le package {svFlow} avec l'instruction suivante : `install.packages('svFlow', repos = c('https://sciviews.r-universe.dev', 'https://cloud.r-project.org'))` . Ensuite, vous faites `library(svFlow)` pour rendre ses fonctions disponibles.↩

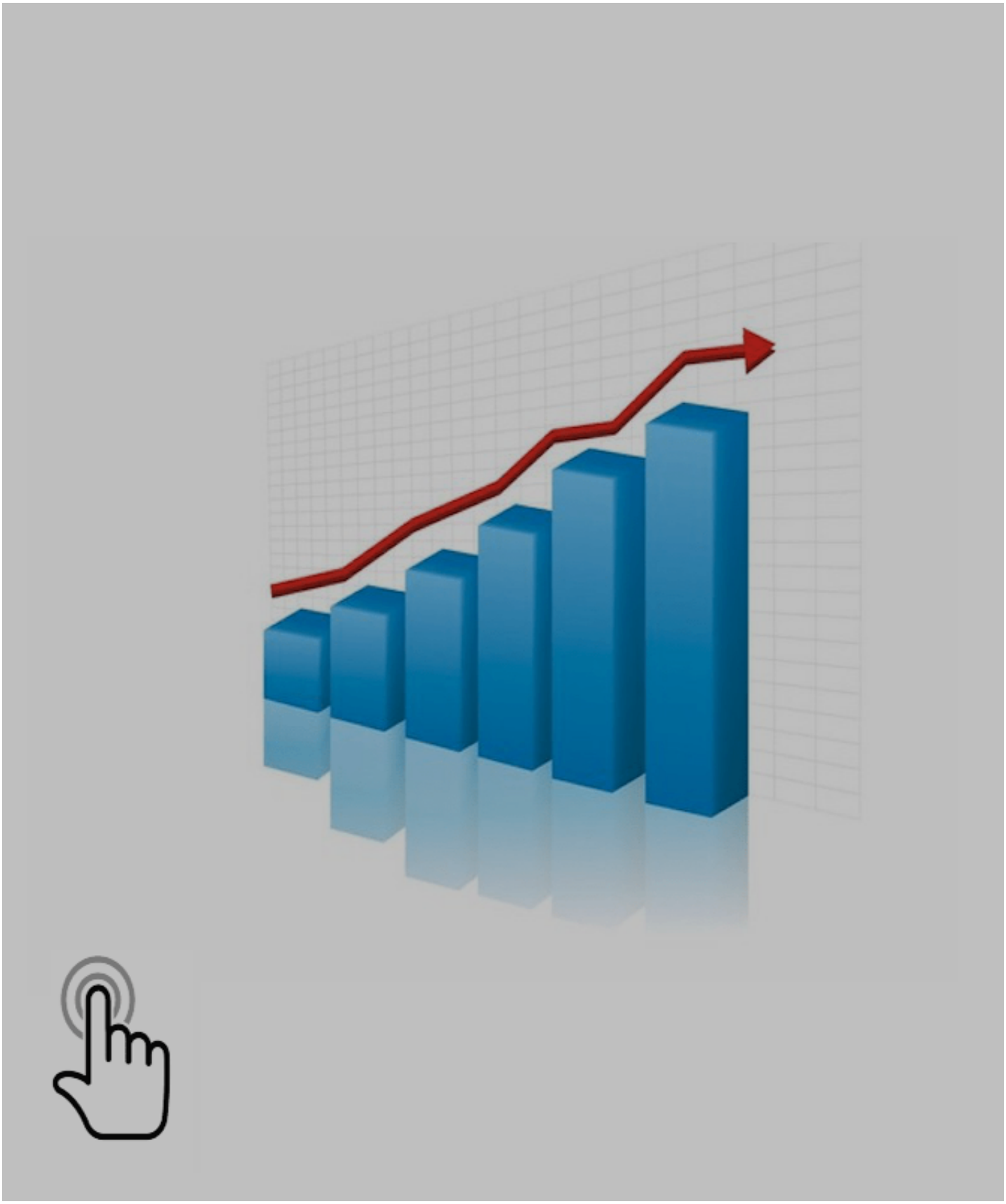
29. Le chaînage n'est cependant pas forcément plus facile à déboguer que la version avec variables intermédiaires. Le package {svFlow} propose la fonction `debug_flow()` à appeler directement après un plantage pour inspecter la dernière instruction qui a causé l'erreur, voir `?debug_flow` .↩

4.6 Récapitulatif des exercices

Le module 4 que vous venez de terminer vous a permis d'apprendre à importer des données depuis différents formats et sources grâce à la fonction `read()` et de transformer un tableau de données ou le résumez à l'aide des fonctions tidyverse, svTidy et fstat. Pour évaluer votre compréhension de cette matière vous aviez les exercices suivants à réaliser :

- A04Ha_acces - Chemins relatifs pour les fichiers
- A04Hb_gitignore - Rôle de .gitignore
- A04Hc_readpackage - Importation de données depuis un package
- A04Hd_variable - Types de variables
- A04He_round - Arrondir des nombres
- A04Hf_filter - Filtrer un tableau
- A04Hg_mutate - Calculer une nouvelle variable
- A04La_wrangling - Traitement des données
- A04Ia_transformation - Importation et transformation de données
- A04Ga_biometry - Collecte de données relative à l'obésité
- A02Ga_analysis - Analyse de données (partie III)

Progression



Cliquez pour visualiser le rapport de progression.

Module 5 Traitement des données II

Objectifs

- Comprendre les principaux tableaux de données utilisés en science des données
- Pouvoir réaliser des tableaux de contingence
- Acquérir des données et les encoder correctement et de manière à ce que les analyses soient reproductibles
- Être capable de remanier des tableaux de données de long en large ou inversement et de fusionner plusieurs tableaux

Prérequis

Ce module est la continuation du module 4 dont le contenu doit être bien compris et maîtrisé avant de poursuivre ici.

5.1 Tableaux de données

Les tableaux de données sont principalement représentés sous deux formes : les tableaux **cas par variables** et les tableaux de **contingence**.

5.1.1 Tableaux cas par variables

Chaque individu est représenté en ligne et chaque variable en colonne par convention. En anglais, on parlera de tidy data.

Nous nous efforcerons de toujours créer un tableau de ce type pour nos données. La question à se poser est la suivante : est-ce que j'ai un seul et même individu statistique[30] représenté à *chaque* ligne du tableau ? Si la réponse est non, le tableau de données n'est **pas** correctement encodé.

À vous de jouer !

Considérez les données suivantes concernant la taille adulte mesurée en cm pour deux espèces de petits rongeurs :

hamster	cobaye
15,4	21,7
18,2	22,0
17,6	24,3
14,2	23,9
16,8	

Est-ce un tableau cas par variable correctement encodé ?

○ Vrai ○ Faux

Vérifier

Considérez ce second tableau de données concernant également la taille adulte mesurée en cm de deux espèces de petits rongeurs :

espèce	taille
hamster	15,4
hamster	18,2
hamster	17,6
hamster	14,2
hamster	16,8
cobaye	21,7
cobaye	22,0
cobaye	24,3
cobaye	23,9

Est-ce un tableau cas par variables correctement encodé ?

○ Vrai ○ Faux

Vérifier

Considérez maintenant ce troisième exemple fictif avec le résultat d'un test enregistré avant et après un traitement censé améliorer le résultat du test :

individu	résultat
individu 1 avant	10,3
individu 1 après	12,5
individu 2 avant	11,6
individu 2 après	12,4
individu 3 avant	10,8
individu 3 après	10,6

Est-ce un tableau cas par variables correctement encodé ?

○ Vrai ○ Faux

Vérifier

avant	après
10,3	12,5
11,6	12,4
10,8	10,6

Est-ce un tableau cas par variables correctement encodé ?

○ Vrai ○ Faux

Vérifier

> Avant toute analyse, vérifiez le type de tableau de données que vous avez à disposition. Le point de départ le plus courant et le plus sûr est le **tableau cas par variables**. Rappelez-vous bien que la question à se poser pour contrôler si le tableau est correctement présenté est : “a-t-on une et une seule ligne dans le tableau pour chaque individu statistique et chaque colonne correspond-t-elle à une et une seule variable ?” Si ce n’est pas le cas, il faut remanier le tableau avant de commencer son analyse.

Les tableaux de données que vous avez traités jusqu’à présent étaient tous des tableaux **cas par variables**. Chaque ligne représentait un *individu statistique* sur qui une ou plusieurs *variables* (en colonnes) étaient mesurées.

```
SciViews::R
biometry <- read("biometry", package = "BioDataScience", lang = "fr")
head(biometry)
```

```
# # A data.trame: [6 × 7]
#   gender day_birth  weight height wrist year_measure   age
#   <fct>  <date>      <dbl>  <dbl> <dbl>        <dbl> <dbl>
# 1 M      1995-03-11     69    182  15           2013    18
# 2 M      1998-04-03     74    190  16           2013    15
# 3 M      1967-04-04     83    185  17.5         2013    46
# 4 M      1994-02-10     60    175  15           2013    19
# 5 W      1990-12-02     48    167  14           2013    23
# 6 W      1994-07-15     52    179  14           2013    19
```

L'encodage d'un petit tableau cas par variables directement dans R est facile. Cela peut se faire de plusieurs façons différentes. En voici deux utilisant les fonctions SciViews::R `dtx()` (spécification colonne par colonne) et `dtx_rows()` (spécification ligne par ligne)[31] :

```
# Spécification colonne par colonne avec dtx()
(df1 <- dtx(
  x = c(1, 2),
  y = c(3, 4)
))
```

```
# # A data.trame: [2 × 2]
#       x     y
#   <dbl> <dbl>
# 1     1     3
# 2     2     4
```

```
# Spécification ligne par ligne avec dtx_rows()
(df2 <- dtx_rows(
  ~x, ~y,
   1,  3,
   2,  4
))


# # A data.trame: [2 × 2]
#       x     y
#   <dbl> <dbl>
# 1     1     3
# 2     2     4
```

La seconde approche est plus naturelle, mais la première permet d'utiliser diverses fonctions de R pour faciliter l'encodage, par exemple :

- Séquence d'entiers successifs :

```
1:10


#  [1]  1  2  3  4  5  6  7  8  9 10
```

- Répétition d'un vecteur cinq fois (le "L" après 5 indique que c'est un entier, mais il est ici facultatif) :

```
rep(c("a", "b", "c"), 5L)


#  [1] "a" "b" "c" "a" "b" "c" "a" "b" "c" "a" "b" "c" "a" "b" "c"
```

- Répétition de chaque item d'un vecteur cinq fois :

```r
rep(c("a", "b", "c"), each = 5L)
```

```
#  [1] "a" "a" "a" "a" "a" "b" "b" "b" "b" "b" "c" "c" "c" "c" "c"
```

Pour de plus gros tableaux, il vaut mieux utiliser un tableur tel qu'Excel ou LibreOffice Calc pour l'encodage. Les tableurs en ligne comme Google Sheets ou Excel Online conviennent très bien également et facilitent un travail collaboratif ainsi que la mise à disposition sur Internet, comme nous avons vu au module 4. **Évitez les formules dans vos tableaux, encodez à partir de la cellule A1 en haut à gauche avec la première ligne qui contient le nom des colonnes. Soyez aussi attentif à ce qu'Excel ne modifie pas de manière inadéquate vos entrées**. Par exemple, Excel a tendance à essayer d'interpréter certaines entrées comme des dates et les modifie automatiquement sans demander votre avis ! Pour cette raison, l'utilisation du format CSV est fortement recommandé à la place du format Excel. Un éditeur CSV spécifique est alors utile, par exemple, ModernCSV qui offre une version gratuite du logiciel suffisante pour l'encodage des données, y compris pour des très gros tableaux que les tableurs ne peuvent pas gérer.

5.1.2 Tableau de contingence

Le tableau cas par variables n'est toutefois pas la seule représentation (correcte) possible des données. Un **tableau de contingence** représente de manière bien plus compacte le dénombrement de l'occurrence de chaque niveau d'une (tableau à une entrée) ou de deux variables **qualitatives** (tableau à double entrée).

À vous de jouer !

Le tableau de contingence permet de représenter de manière compacte le dénombrement des occurrences d'une variable quantitative.

○ Vrai ○ Faux

Vérifier

La fonction `table()` crée ces deux types de tableaux de contingence à partir de données présentées en tableau cas par variables :

```
# Variable age_rec qualitative
biometry$age_rec <- cut(biometry$age, include.lowest = FALSE, right = TRUE,
  breaks = c(14, 27, 90))
# Tableau de contingence genre versus age_rec
(bio_tab <- table(biometry$gender, biometry$age_rec))
```

```
#
#     (14,27] (27,90]
#   M     106      92
#   W      97     100
```

À vous de jouer !

Placez les mots dans les emplacements correspondants pour calculer un tableau de contingence de la variable 'origin' du jeu de données 'urchin'.

urchin <- read("urchin_bio", package = "data.io")

($)

origin
table
urchin

Vérifier

Le tableau de contingence peut toujours être calculé à partir d'un tableau cas par variable, mais il peut également être encodé directement si nécessaire. Voici un petit tableau de contingence à simple entrée encodé directement comme tel (vecteur nommé transformé en objet `table` à l'aide de la fonction `as.table()`) :

```
# Tableau de contingence encodé directement
anthirrhinum <- as.table(c(
  "fleur rouge"   = 54,
  "fleur rose"    = 122,
  "fleur blanche" = 58)
)
anthirrhinum


#   fleur rouge    fleur rose fleur blanche
#            54           122            58
```

Une troisième possibilité est d'utiliser un tableau indiquant les **fréquences d'occurrences** dans une colonne (`freq` ci-dessous). Ce n'est **pas** un tableau cas par variables, mais une forme bien plus concise et pratique pour préencoder les données qui devront être ensuite transformées en tableau de contingence en utilisant cette fois-ci la fonction `xtabs()`. Voici un exemple pour un tableau de contingence à double entrée. Notez que le tableau cas par variables correspondant devrait contenir 44 + 116 + 19 + 128 = 307 lignes et serait plus fastidieux à construire et à manipuler (même en utilisant la fonction `rep()`).

```
# Tableau avec fréquences
timolol <- dtx(
  traitement = c("timolol", "timolol", "placebo", "placebo"),
  patient    = c("sain",    "malade",  "sain",    "malade"),
  freq       = c(44,        116,       19,        128)
)
# Tableau de contingence patient versus traitement
(timolol_table <- xtabs(data = timolol, freq ~ patient + traitement))
```

```
#         traitement
# patient  placebo timolol
#   malade     128     116
#   sain        19      44
```

La sortie par défaut d'un tableau de contingence n'est pas très esthétique, mais plusieurs options existent pour le formater d'une façon agréable. En voici trois exemples :

```
pander::pander(timolol_table,
  caption = "Exemple de table de contingence à double entrée.")
```

Exemple de table de contingence à double entrée.

	placebo	timolol
malade	128	116
sain	19	44

```
knitr::kable(timolol_table,
  caption = "Exemple de table de contingence à double entrée.")
```

Tableau 5.1: Exemple de table de contingence à double entrée.

	placebo	timolol
malade	128	116
sain	19	44

```
tabularise(timolol_table)
```

Tableau 5.2: Exemple de table de contingence à double entrée.

	traitement	

patient		placebo	timolol	Total
malade	Count	128 (41.7%)	116 (37.8%)	244 (79.5%)
	Mar. pct (1)	87.1% ; 52.5%	72.5% ; 47.5%	
sain	Count	19 (6.2%)	44 (14.3%)	63 (20.5%)
	Mar. pct	12.9% ; 30.2%	27.5% ; 69.8%	
Total	Count	147 (47.9%)	160 (52.1%)	307 (100.0%)

(1) Columns and rows percentages

La dernière version avec `tabularise()` ajoute des informations utiles comme les totaux par ligne, par colonne et général, ainsi que divers pourcentages. C'est ce type de sortie que nous privilégierons dans `SciViews::R` .

Il est aussi possible de représenter *graphiquement* un tableau de contingence pour l'inclure dans une figure composée, éventuellement en le mélangeant à des graphiques[32].

```
tab1 <- ggpubr::ggtexttable(head(biometry), rows = NULL)
tab2 <- ggpubr::ggtexttable(table(biometry$gender, biometry$age_rec))

combine_charts(list(tab1, tab2), nrow = 2)
```

A

gender	day_birth	weight	height	wrist	year_measure	age	age_rec
M	1995-03-11	69	182	15.0	2013	18	(14,27]
M	1998-04-03	74	190	16.0	2013	15	(14,27]
M	1967-04-04	83	185	17.5	2013	46	(27,90]
M	1994-02-10	60	175	15.0	2013	19	(14,27]
W	1990-12-02	48	167	14.0	2013	23	(14,27]
W	1994-07-15	52	179	14.0	2013	19	(14,27]

B

	(14,27]	(27,90]
M	106	92
W	97	100

Différentes fonctions dans R existent également pour convertir un tableau de contingence en tableau cas par variables (ou en tous cas, en un tableau similaire). Par exemple, `as_dtx()` renvoie un tableau indiquant les fréquences d'occurrences dans une colonne nommée `N` :

```
(timolol2 <- as_dtx(timolol_table))
```

```
# # A data.trame: [4 × 3]
#   patient traitement     N
#   <chr>   <chr>      <dbl>
# 1 malade  placebo      128
# 2 sain    placebo       19
# 3 malade  timolol      116
# 4 sain    timolol       44
```

Si vous insistez, vous pouvez aussi obtenir un tableau cas par variables (mais celui-ci est très long et peu pratique à manipuler) à l'aide de la fonction "svTidy" `uncount_()` (ou la fonction "tidyverse" `uncount()`)[33] :

```
uncount_(timolol2, "N") # On peut aussi écrire ~N
```

```
# # A data.trame: [307 × 2]
#    patient traitement
#    <chr>   <chr>
#  1 malade  placebo
#  2 malade  placebo
#  3 malade  placebo
#  4 malade  placebo
#  5 malade  placebo
#  6 malade  placebo
#  7 malade  placebo
#  8 malade  placebo
#  9 malade  placebo
# 10 malade  placebo
# # ℹ 297 more rows
```

5.1.3 Métadonnées

Les données dans un tableau doivent **impérativement** être associées à un ensemble de métadonnées. Les métadonnées (*metadata* en anglais) apportent des informations complémentaires nécessaires pour une interprétation correcte des données. Elles permettent donc de replacer les données dans leur contexte et de spécifier des caractéristiques liées aux mesures réalisées comme les unités de mesure entre autres.

> Données de qualité = tableau de données + métadonnées

Les données correctement qualifiées et documentées sont les seules qui peuvent être utilisées par un collaborateur externe. C'est-à-dire qu'une personne externe ne peut interpréter le tableau de données que si les métadonnées sont complètes et suffisamment explicites.

Exemple de métadonnées :

- Unités de mesure (exemple : 3,5 mL, 21,2 °C)
- Précision de la mesure (21,2 ± 0,2 dans le cas d'un thermomètre gradué tous les 0,2 °C)
- Méthode de mesure utilisée (thermomètre à mercure, ou électronique, ou …)
- Type d'instrument employé (marque et modèle du thermomètre)
- Date et lieu de la mesure (si pas encodés directement dans le tableau)
- Nom du projet lié à la prise de mesure
- Nom de l'opérateur en charge de la mesure
- …

Vous avez pu vous apercevoir que la fonction `read()` permet d'ajouter certaines métadonnées comme les unités aux variables d'un jeu de données. Cependant, il n'est pas toujours possible de rajouter les métadonnées dans un tableau sous forme électronique, mais il faut toujours les consigner dans un **cahier de laboratoire**, et ensuite les **retranscrire dans le rapport**. La fonction `labelise()` vous permet de rajouter le **label** et les **unités** de mesure pour vos différentes variables directement dans le tableau. Par exemple, voici l'encodage direct d'un petit jeu de données qui mesure la distance du saut (`jump`) en cm de grenouilles-taureaux *Lithobates catesbeianus* (Shaw 1802) en fonction de leur masse (`weight`) en g pour cinq individus différents (`ind`). Vous pouvez annoter ce tableau de la façon suivante :

```r
frog <- dtx_rows(
  ~ind, ~jump, ~weight,
   "1",    71,     204,
   "2",    70,     240,
   "3",   100,     296,
   "4",   120,     303,
   "5",   103,     422
)
# Ajout des labels et des unités
frog <- labelise(frog,
  label = list(
    ind    = "Individu",
    jump   = "Distance du saut",
    weight = "Masse"),
  units = list(# Si pas d'unité, juste ne rien indiquer
    jump   = "cm",
    weight = "g")
  )
# Affichage synthétique des données et métadonnées associées
str(frog)


# dtrm [5 × 3] (S3: data.trame/data.frame)
#  $ ind   : chr [1:5] "1" "2" "3" "4" ...
#   ..- attr(*, "label")= chr "Individu"
#  $ jump  : num [1:5] 71 70 100 120 103
#   ..- attr(*, "label")= chr "Distance du saut"
#   ..- attr(*, "units")= chr "cm"
#  $ weight: num [1:5] 204 240 296 303 422
#   ..- attr(*, "label")= chr "Masse"
#   ..- attr(*, "units")= chr "g"
#  - attr(*, ".internal.selfref")=<externalptr>
```

```r
# Affichage des labels
label(frog)
```

```
#                ind               jump             weight
#         "Individu" "Distance du saut"           "Masse"
```

Les métadonnées sont enregistrées dans des **attributs** en R (`attr`). De même, `comment()` permet d'associer ou de récupérer un attribut commentaire (notez comment assigner une chaîne de caractères à `comment()` ajoute ou modifie le commentaire de l'objet, tableau entier ou variable du tableau) :

```r
# Ajout d'un commentaire concernant le jeu de données lui-même
comment(frog) <- "Saut de grenouilles taureaux"
# Ajout d'un commentaire sur une variable
comment(frog$jump) <- "Premier saut mesuré après stimulation de l'animal"
# Affichage synthétique
str(frog)
```

```
# dtrm [5 × 3] (S3: data.trame/data.frame)
#  $ ind   : chr [1:5] "1" "2" "3" "4" ...
#   ..- attr(*, "label")= chr "Individu"
#  $ jump  : num [1:5] 71 70 100 120 103
#   ..- attr(*, "label")= chr "Distance du saut"
#   ..- attr(*, "units")= chr "cm"
#   ..- attr(*, "comment")= chr "Premier saut mesuré après stimulation de l'an
#  $ weight: num [1:5] 204 240 296 303 422
#   ..- attr(*, "label")= chr "Masse"
#   ..- attr(*, "units")= chr "g"
#  - attr(*, ".internal.selfref")=<externalptr>
#  - attr(*, "comment")= chr "Saut de grenouilles taureaux"
```

```r
# Récupération des commentaires
comment(frog)


# [1] "Saut de grenouilles taureaux"


comment(frog$jump)


# [1] "Premier saut mesuré après stimulation de l'animal"


comment(frog$weight) # Rien!


# NULL
```

5.1.4 Dictionnaire des données

Le dictionnaire des données est un élément important de la constitution d'une base de données. Il s'agit d'un tableau annexe qui reprend au minimum le **nom** de chaque variable, son **label** (nom plus long et explicite), ses **unités** éventuelles, son **type** (numérique, facteur, facteur ordonné, date …), les **niveaux** des variables qualitatives, la façon dont les **valeurs manquantes** sont encodées, et un **commentaire** éventuel. Cela donne donc un tableau du genre :

Variable	Label	Unités	Type	Niveaux	Val. manquantes	Commentaire
date	Date	-	Date		NA	Date de mesure
age	Âge	année	numeric		-1	
diameter	Diamètre du test	mm	numeric		NA	Moyenne de deux diamètres perpendiculaires
origin	Origine	-	factor	Fishery, Farm	unknown	“Fishery” = oursins sauvages, “Farm” = oursins d’élevage

Ce tableau peut-être encodé sous forme textuelle (CSV par exemple) et placé dans le même dossier que le jeu de données lui-même. Il peut aussi être encodé comme feuille supplémentaire dans un fichier Excel.

> Le dictionnaire des données est un outil important pour comprendre ce que contient le tableau de données, et donc, son interprétation. *Ne le négligez pas !*

À vous de jouer !

Sélectionnez la phrase correcte. ✔ Progression : 0/5

Le tableau de contingence est le type de tableau de données à utiliser de préférence lors de l'encodage des données d'une expérience.

Le tableau cas par variables est le type de tableau de données à utiliser de préférence lors de l'encodage des données d'une expérience.

30. Un individu statistique n'est pas forcément la même chose qu'un individu biologique. En statistique, le cas ou l'individu est l'entité sur laquelle nous travaillons. Il peut s'agir d'un organe (partie d'un individu biologique) comme d'un ensemble d'organismes (une culture bactérienne ou la portée d'une souris par exemple, donc, plusieurs individus biologiques).↩

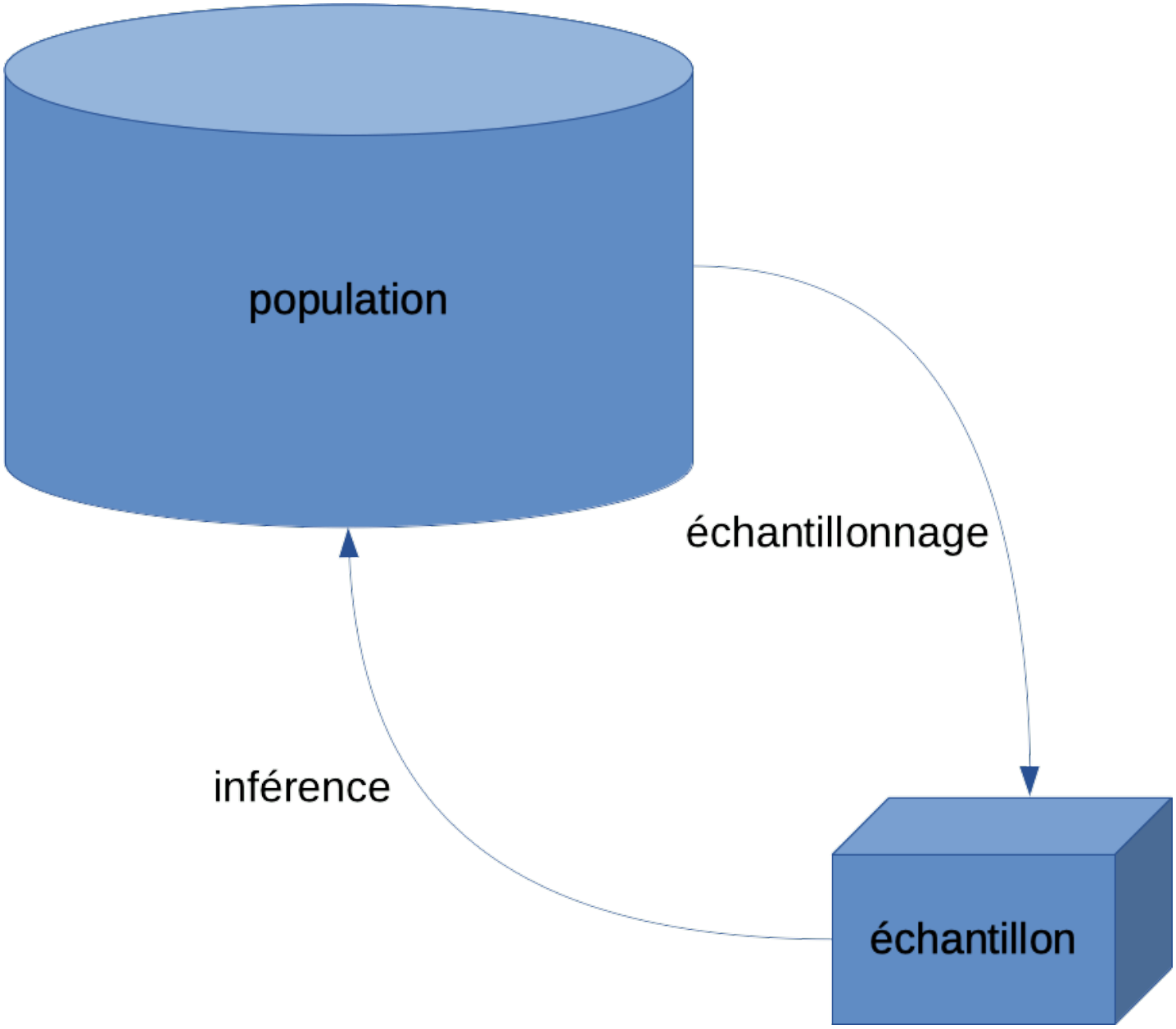
31. L'équivalent en R de base de `dtx()` est `data.frame()` et vous pouvez aussi utiliser `data.table()` du package de même nom, ou pour le Tidyverse, `tibble()` du package {tibble}. Pour `dtx_rows()`, seul un équivalent Tidyverse existe avec `tibble::tribble()` que `dtx_rows()` utilise d'ailleurs en interne. Les fonctions `dtx_...()` ont pour avantage qu'elles vous permettent de choisir le type de data frame que vous voulez utiliser de manière centralisée avec `options(SciViews.as_dtx = as_dtrm)` pour un **data.trame** de SciViews::R. Substituez `as_dtrm` par `as_dtf` pour un **data.frame** de R de base, `as_dtt` pour une **data.table** du Fastverse et `as.dtbl` pour un **tibble** du Tidyverse.↩

32. Utilisez cette option avec parcimonie. Il vaut toujours mieux représenter un tableau comme … un tableau plutôt que comme une figure !↩

33. Notez également que passer d'un tableau cas par variables à un tableau des fréquences d'occurrences se fait à l'aide de `count_()` ou `count()`.↩

5.2 Population et échantillonnage

Nos analyses statistiques seraient bien évidemment beaucoup plus simples si nous pouvions toujours mesurer tous les individus statistiques concernés par nos études. En fait, c'est presque toujours impossible, car les **populations** concernées sont souvent très larges, voire infinies. Nous devons donc mesurer un petit sous-ensemble de la population, ce que nous appelons un **échantillon** (*sample* en anglais) de taille finie déterminée (on parle de la **taille de l'échantillon** ou *sample size* en anglais). Le processus qui mène à la sélection des individus dans l'échantillon s'appelle l'**échantillonnage** (*sampling* en anglais). De cet échantillon, nous souhaitons malgré tout retirer un maximum d'information concernant la population d'origine. Cela s'appelle faire de l'**inférence** sur la population à partir de l'échantillon (*to infer* en anglais).



Population et échantillon.

> Faites attention à la signification des termes selon les disciplines. Par exemple, le terme **population** ne signifie pas la même chose pour un biologiste (ensemble des individus d'une même espèce pouvant se reproduire entre eux) et pour un statisticien (ensemble des valeurs que peut prendre une variable). Ainsi, la définition statistique du terme se réfère à un bien plus grand sous-ensemble en général. Par exemple, si nous étudions la souris *Mus musculus* Linné 1758, nous considérerons bien évidemment une population (ou une souche donnée pour les souris de laboratoire) en qualité de biologistes. Les statisticiens considéreront l'ensemble des souris qui existent, ont existé et existeront à l'avenir comme **la** population de souris.

> De même, nous l'avons déjà expliqué, un **individu** statistique est une entité sur laquelle nous effectuons nos mesures (ce qui donne lieu à une **observation** pour chaque **variable**). Ainsi, un individu statistique peut correspondre ou non à un individu biologique selon ce que l'on considère. Par exemple, une section dans un organe peut constituer un individu statistique lors de l'étude de l'hétérogénéité à l'intérieur de cet organe. Nous aurons donc autant d'individus statistiques que de sites de prélèvement sur l'organe… même si ce dernier provient au final du même individu biologique (même animal ou végétal) !

La **variabilité** tant au niveau de la population que de l'échantillon provient essentiellement de deux facteurs :

1. La **variabilité individuelle** inhérente (nous n'avons pas tous la même taille ni la même masse, par exemple),
2. Les **erreurs de mesure**.

Ces deux sources de variabilité se cumulent (généralement) pour contribuer à disperser les valeurs d'un individu à l'autre. Nous n'avons que peu de prise sur la variabilité individuelle, mais nous pouvons parfois réduire les erreurs de mesure si cela s'avère souhaitable en utilisant un appareil de mesure plus précis. Quoi qu'il en soit, plus la variabilité est importante, plus la taille de l'échantillon devra également être grande pour conserver une bonne **"représentativité"** de la population statistique.

Du point de vue de la notation mathématique, nous utiliserons une lettre latine majuscule en italique pour représenter une variable, par exemple, X. La taille de l'échantillon est souvent notée n. Les observations individuelles de la variable X pour les n individus de l'échantillon seront alors notées avec une lettre minuscule en italique assortie d'un indice compris entre 1 et n, donc, x_i de manière générale avec l'indice i variant de 1 à n. Ainsi, x_5 représente la cinquième observation pour la variable X dans notre échantillon, et x_n est la dernière observation. Quelquefois, l'indice i est omis et vous pourrez aussi lire x simplement.

5.2.1 Échantillonnage aléatoire

La démarche du statisticien et du scientifique des données est de retirer un maximum d'information d'un échantillon afin de faire des inférences fiables sur la population tout entière (c'est-à-dire, tenter de tirer des conclusions les plus générales possibles à partir de l'étude d'un petit échantillon seulement). Ceci n'est faisable que si l'échantillon est "conforme" à la population. En langage statistique on dit que l'**échantillon est représentatif**.

Un échantillon représentatif n'est pas facile à obtenir et il faut respecter scrupuleusement certaines règles pour l'obtenir. *A contrario* un échantillon **non** représentatif s'obtient sans précautions particulières, mais ce dernier est quasi toujours totalement inutile (on ne peut tirer de conclusions *que* sur cet échantillon particulier). Imaginez un soudage qui ne représente pas la population sondée… quel est son intérêt ? Il est nul !

La meilleure façon d'obtenir un échantillon représentatif est de réaliser un **échantillonnage aléatoire** (*random sampling* en anglais). Dans ce cas, chaque individu de la population doit avoir la même chance d'être sélectionné dans l'échantillon. Nous parlons aussi d'échantillonnage au hasard. Vous devez bien réfléchir au **processus** qui va mener à la sélection des individus dans l'échantillon. Celui-ci doit comporter une étape qui fait intervenir le hasard.

> "Choisir au hasard" ses individus en les prélevant au petit bonheur la chance en fonction de son humeur n'est **pas** une bonne approche. En effet, notre main qui saisira les individus à inclure dans l'échantillon aura tendance à prélever ceux qui sont plus facile à attraper ou plus proches, par exemple. Il peut s'agir d'individus moins vigoureux, moins réactifs, ou au contraire, moins peureux… Du coup, nous n'étudierions qu'une fraction de la population qui correspond à une caractéristique particulière (ceux qui sont faibles et peu réactifs, par exemple).
>
> Une bonne sélection aléatoire **doit** faire intervenir le hasard (tirage au sort dans une urne, pile ou face, jet de dés, ou génération de nombres dits "pseudo-aléatoires" à l'aide d'un ordinateur). Par exemple, si vous avez un élevage de souris dans votre laboratoire, et que vous considérez cet élevage comme votre population, vous pouvez réaliser deux groupes (témoin et traitement) de cinq souris chacun de plusieurs façons :

1. Prendre dix souris dans les cages “au hasard”, et les répartir toujours “au hasard” entre les deux groupes. Ce type d’échantillonnage est **incorrect**. En effet, votre choix sera (inconsciemment) conditionné.
2. Donner un identifiant numérique unique à chacune de vos souris. Ensuite tirer deux fois cinq identifiants à partir d’une urne, ou effectuer ce même traitement virtuellement à l’aide d’un ordinateur. Dans ce cas, l’échantillonnage sera réellement aléatoire et correctement réalisé !

En pratique, nous pourrons utiliser la fonction `sample()` dans R. Elle permet de simuler facilement et rapidement le processus de tirage au hasard depuis une urne. Dans le cas de nos deux groupes de cinq souris à partir d’un élevage qui contient 213 animaux numérotés de 1 à 213 (identifiants uniques), nous ferons :

```
mice <- 1:213
# Échantillon de 5 souris témoins
(control <- sample(mice, size = 5))
```

```
# [1]  47 201  36  92 212
```

```
# Échantillon de cinq autres souris pour le traitement (pas déjà incluses dans
(treatment <- sample(mice[!mice %in% control], size = 5))
```

```
# [1] 140  20  22 106  83
```

Ici, la sélection aléatoire (en réalité, on parle de “pseudo-aléatoire” pour ces nombres générés par un algorithme, mais qui ont des propriétés très similaires au hasard pur) nous indique que nous devrons aller chercher les souris 47, 201, 36, 92, 212 dans notre élevage pour notre groupe témoin, et les souris 140, 20, 22, 106, 83 pour le groupe traitement.

Dans le cas où un échantillonnage doit se faire **avec remplacement** (l'équivalent de replacer une boule dans l'urne à chaque tirage au sort), nous pouvons indiquer l'argument `replace = TRUE` dans `sample()`. Donc, quelque chose comme `sample(0:9, size = 50, replace = TRUE)` pour échantillon les chiffres 0 à 9 au hasard cinquante fois avec remise (à chaque tirage chaque chiffre a même chance d'être tiré).

L'utilisation de `sample()` est pratique, mais cela rend le code **non reproductible**. En effet, à chaque fois, la fonction génère une série différente au hasard (c'est son rôle !) Toutefois, comme la série est pseudo-aléatoire, il est possible de régénérer la même série une seconde fois si on part du même point dans l'algorithme qui la calcule. Cela peut être réalisé à l'aide de `set.seed()`. Vous *devez* indiquer comme argument à cette dernière fonction un nombre aléatoire. Ce nombre représentera une position dans la séquence générée par l'algorithme de sorte que la série suivante obtenue à l'aide de `sample()` sera toujours la même. Voici comment cela fonctionne :

```
# Utiliser un nombre différent à chaque appel de set.seed() !
set.seed(2563)
sample(1:10, size = 6)
```

```
# [1] 9 8 5 1 6 3
```

Naturellement, chaque fois que vous insérez `set.seed()` dans votre code, vous devez indiquer un nombre différent, lui-même choisi au hasard.

5.2.2 Échantillonnage stratifié

L'échantillonnage aléatoire n'est pas la seule stratégie correcte. L'**échantillonnage stratifié** consiste à diviser la population en sous-populations identifiables facilement (par exemple, séparer la population en fonction du sexe, ou de l'âge, voire des deux simultanément). Ensuite, un échantillonnage aléatoire est réalisé à l'intérieur de chaque

sous-population pour un nombre déterminé d'individus, souvent le même. Cette approche plus complexe est utile si les sous-populations sont **très mal balancées** (une sous-population possède bien plus d'individus qu'une autre).

Par exemple, vous voulez comparer des prélèvements sanguins faits sur une population d'Européens hospitalisés dans le but de déterminer les critères qui permettent de diagnostiquer une infection nosocomiale[34] rare. Les individus sont suivis en hôpital par définition et nous admettrons que l'on sait *a priori* s'ils sont atteints de la maladie ou non grâce à d'autres tests plus lourds et coûteux. La maladie est rare, heureusement. Sa **prévalence** n'est que de 1 cas sur 10.000.

Dans ce cas, si vous effectuez un échantillonnage aléatoire de 100 patients ou même de 1.000 patients hospitalisés, vous avez toutes les chances de ne même pas inclure un seul patient atteint de l'infection dans votre échantillon ! Ici l'échantillonnage stratifié est utile. Il consiste à séparer les patients en deux sous-populations : ceux qui sont atteints et les autres. Ensuite, vous décidez, par exemple, d'analyser 50 prélèvements sanguins dans chacune des deux sous-populations. Vous échantillonnez alors 50 patients aléatoirement comme nous l'avons fait plus haut pour nos souris, mais à l'intérieur de chaque sous-population. Au final, votre échantillon contiendra alors 50 patients infectés et 50 autres qui ne le sont pas. Cet échantillon est également représentatif (sauf, bien sûr, concernant la prévalence de l'infection que nous ne pouvons plus calculer à partir de ce type d'échantillon).

34. Une infection nosocomiale est une infection contractée dans un hôpital. ↩

5.3 Acquisition de données

Dans le module 4, vous avez pris connaissance des types de variables et venez d'apprendre comment encoder différents types de tableaux de données et leur associer les indispensables métadonnées. Cependant, la première étape avant d'acquérir des données est de planifier correctement son expérience. La science des données est intimement liée à la démarche scientifique et intervient dans toutes les étapes depuis la caractérisation de la question et le planning de l'expérience jusqu'à la diffusion des résultats. Plus en détail, cela correspond à :

- Définir une question (objectif)
- Réaliser une recherche bibliographique sur la thématique
- Définir le protocole de l'expérience à partir de l'objectif
    - Définir la population étudiée et l'échantillonnage
    - Définir les variables à mesurer
        - Définir les unités des mesures
        - Définir la précision des mesures
        - Sélectionner les instruments de mesure nécessaires
- Définir les conventions d'encodage
    - Codifier l'identification des individus
    - Définir les niveaux des variables facteurs et leurs labels
- Acquérir et encoder les données
- Traiter les données
    - Importer des données
    - Remanier des données
    - Visualiser et décrire les données
    - Analyser les données (traitements statistiques, modélisation…).
- Produire des supports de présentation répondant à la question de départ et diffuser l'information dans la communauté scientifique

Nous traitons ici des premières étapes qui visent à **acquérir les données**.

5.3.1 Précision et exactitude

Les erreurs de mesures sont inévitables lors de l'acquisition de nos données. Cependant, il est possible de les minimiser en choisissant un instrument plus précis (*precise* en anglais) et plus exact (*accurate* en anglais). Ces deux notions sont différentes, mais complémentaires.

À vous de jouer !

H5P
A05Hh_precision

Exact et i

Inexact e

Exact et

Inexact et

Vérifier

5.3.2 Codification des données

Afin d'éviter que divers collaborateurs n'encodent différemment une information similaire, vous allez devoir préciser très clairement comment encoder les différentes variables de votre jeu de données. Par exemple pour une variable `genre`, est-ce que vous indiquez `homme / femme`, ou `h / f`, ou encore `H / F` ?

De même, vous allez devoir attribuer un code **unique** à chaque individu mesuré. Enfin, vous devez vous assurer que toutes les mesures sont réalisées de la même manière et avec des instruments qui, s'ils sont différents, seront cependant **intercalibrés** (procédure visant à vérifier que les différents appareils produisent des mesures similaires en comparant leurs réponses à partir des *mêmes* échantillons). Comment faire ? Réfléchissez à cette question sur base d'une mesure de la masse des individus à l'aide de pèse-personnes différents !

5.3.3 Respect de la vie privée

Lors d'expérience sur des personnes, le respect de la vie privée **doit** être pris en compte[35]. Le nom et le prénom, ou toute autre information permettant de retrouver les individus étudiés (adresse mail, numéro de sécurité sociale, etc.) **ne peuvent pas apparaître** dans la base de données consolidée. En outre, il vous faudra un **accord explicite** des personnes que vous voulez mesurer, et il faudra leur expliquer ce que vous faites, et **comment** les données seront ensuite utilisées. Une question se pose : comment pouvoir revenir vers les enregistrements liés à un individu en particulier (en cas d'erreur d'encodage par exemple) si les informations relatives directement à ces individus ne sont pas consignées dans le tableau final ? Réfléchissez à la façon dont vous vous y prendriez avant de lire la suite…

Voici un petit tableau qui correspond à ce que vous ne pourrez **pas faire** (nom et prénom explicitement mentionnés dans le tableau) :

```
(biometry_marvel <- dtx_rows(
  ~id,                ~gender, ~weight, ~height,
   "Banner Bruce",        "M",      95,    1.91,
   "Stark Tonny",         "M",      80,    1.79,
   "Fury Nicholas",       "M",      82,    1.93,
   "Romanoff Natasha",    "F",      53,    1.70
))


# # A data.trame: [4 × 4]
#   id              gender weight height
#   <chr>           <chr>   <dbl>  <dbl>
# 1 Banner Bruce    M          95   1.91
# 2 Stark Tonny     M          80   1.79
# 3 Fury Nicholas   M          82   1.93
# 4 Romanoff Natasha F          53   1.7
```

Vous devez fournir un code permettant de garder l'anonymat des sondés à l'ensemble des personnes étudiées vis-à-vis des analystes qui vont utiliser ces données. Cependant, le code doit permettre au chercheur ayant pris ces mesures de les retrouver dans son cahier de laboratoire, si besoin. Une façon de procéder consiste à attribuer un numéro au hasard par tirage dans une urne à chacune des personnes chargées des mesures. Ensuite, chaque expérimentateur attribue lui-même un second numéro aux différentes personnes qu'il mesure. Prenons par exemple le scientifique n°24 (seul lui sait qu'il porte ce numéro). Il attribue un code de 1 à *n* sous forme d'une lettre de l'alphabet à chaque personne étudiée. En combinant le code secret de l'expérimentateur et le code individu, cela donne un identifiant unique de la forme `24_A`, `24_B`, etc. Il pourra alors encoder sa partie comme suit :

```
(biometry_marvel1 <- dtx_rows(
    ~id, ~gender, ~weight, ~height,
 "24_A",     "M",      95,    1.91,
 "24_B",     "M",      80,    1.79,
 "24_C",     "M",      82,    1.93,
 "24_D",     "F",      53,    1.70
))
```

```
# # A data.trame: [4 × 4]
#   id    gender weight height
#   <chr> <chr>   <dbl>  <dbl>
# 1 24_A  M          95   1.91
# 2 24_B  M          80   1.79
# 3 24_C  M          82   1.93
# 4 24_D  F          53   1.7
```

Il garde néanmoins les correspondances dans son carnet de laboratoire, au cas où il faudrait faire des vérifications ou revenir à la donnée originale (l'information ci-dessous n'est connue que de cet expérimentateur).

```
(biometrie_correspondance <- dtx(
  name = biometry_marvel$id,
  id   = biometry_marvel1$id
))
```

```
# # A data.trame: [4 × 2]
#   name             id
#   <chr>            <chr>
# 1 Banner Bruce     24_A
# 2 Stark Tonny      24_B
# 3 Fury Nicholas    24_C
# 4 Romanoff Natasha 24_D
```

À partir des données du tableau général consolidé, personne à part lui ne peut revenir sur ces données d'origine et mettre un nom sur les individus mesurés. Et lui-même n'a pas la possibilité de déterminer *qui* se cache derrière les autres identifiants tels `03_A`, `12_C`, `21_B`, etc.

Vous allez maintenant encoder et analyser vos données relatives à l'étude de l'obésité.

> Réalisez en groupe le travail **A04Ga_biometry, partie II**.
>
> **Initiez votre projet GitHub Classroom**
>
> *Voyez les explications dans le fichier `README.md`, partie II.*

35. En Europe, les données numériques concernant les personnes sont soumises à des règles strictes édictées dans le Règlement Général pour la Protection des Données ou **RGPD** en abrégé, en vigueur depuis le 25 mai 2018. Vous devez vous assurer de respecter ce règlement lors de la collecte et de l'utilisation de données relatives à des personnes. Pour les autres types de données, le droit d'auteur, une licence particulière ou des copyrights peuvent aussi limiter votre champ d'action. Renseignez-vous !↩

5.4 Recombinaison de tableaux

5.4.1 Formats long et large

Le **format long** d'un tableau de données correspond à un encodage en un minimum de colonnes, les données étant réparties sur un plus grand nombre de lignes en comparaison du **format large** qui regroupe les données dans plusieurs colonnes successives. Voici un exemple fictif d'un jeu de données au format long :

```
# # A data.trame: [6 × 3]
#   sex   treatment value
#   <chr> <chr>     <dbl>
# 1 m     control     1.2
# 2 f     control     3.4
# 3 m     test1       4.8
# 4 f     test1       3.1
# 5 m     test2       0.9
# 6 f     test2       1.2
```

Voici maintenant le même jeu de données présenté dans le format large :

```
# # A data.trame: [2 × 4]
#   sex   control test1 test2
#   <chr>   <dbl> <dbl> <dbl>
# 1 m         1.2   4.8   0.9
# 2 f         3.4   3.1   1.2
```

Dans le format large, les différents niveaux de la variable facteur `treatment` deviennent autant de colonnes (donc de variables) séparées, et la variable d'origine n'existe plus de manière explicite. Ces deux tableaux contiennent la même information. Bien évidemment, un seul de ces formats est un *tableau cas par variables correct*. Le format long sera le bon si toutes les mesures sont réalisées sur des individus différents. Par contre, le format large sera correct si les différentes mesures ont été faites à chaque fois sur les *mêmes* individus (dans le cas présent, *un seul* mâle et *une seule* femelle auraient alors été mesurés dans les trois situations).

> C'est la règle qui veut qu'**une ligne corresponde à un et un seul individu statistique** dans un tableau cas par variables qui permet de décider si le format long ou le format large est celui qui est correctement encodé.

Encoder correctement un tableau de données n'est pas une chose simple. Il peut y avoir plusieurs manières de le représenter. De plus, beaucoup de scientifiques ignorent ou oublient l'importance de bien encoder un tableau sous forme cas par variables. Lorsque vous souhaitez effectuer une représentation graphique, un format peut convenir mieux qu'un autre également, en fonction de ce que vous souhaitez visualiser sur le graphique. Il est donc important de connaître les fonctions permettant de recombiner un tableau de données d'une forme vers l'autre : `pivot_longer_()` / `pivot_longer()` et `pivot_wider_()` / `pivot_wider()` .

> L'aide-mémoire Data tidying with tidyr est un outil pratique pour vous aider à retrouver les fonctions. Les explications relatives à cette partie s'y trouvent dans la section **Reshape Data**.
>
> L'utilisation des fonction `pivot_longer()` et `pivot_wider()` du package {tidyr} est également décrite en détails dans R for Data Science (2e).

Prenons l'exemple d'un jeu de données provenant de l'article scientifique suivant : Paleomicrobiology to investigate copper resistance in bacteria: isolation and description of *Cupriavidus necator* B9 in the soil of a medieval foundry. L'article est basé sur l'analyse

métagénomique de type “shotgun” de quatre communautés microbiennes (notées `c1`, `c4`, `c7` et `c10`, respectivement)[36]. Il en résulte une longue liste de séquences que l’on peut attribuer à des règnes.

```
shotgun_wide <- dtx(
  kingdom = c("Archaea", "Bacteria", "Eukaryota", "Viruses",
              "other sequences", "unassigned", "unclassified sequences"),
  c1      = c( 98379, 6665903, 81593, 1245, 757, 1320419, 15508),
  c4      = c( 217985, 9739134, 101834, 4867, 1406, 2311326, 21572),
  c7      = c( 143314, 7103244, 71111, 5181, 907, 1600886, 14423),
  c10     = c(272541, 15966053, 150918, 15303, 2688, 3268646, 35024))

shotgun_wide
```

```
# # A data.trame: [7 × 5]
#   kingdom                    c1      c4      c7      c10
#   <chr>                   <dbl>   <dbl>   <dbl>    <dbl>
# 1 Archaea                 98379  217985  143314   272541
# 2 Bacteria              6665903 9739134 7103244 15966053
# 3 Eukaryota               81593  101834   71111   150918
# 4 Viruses                  1245    4867    5181    15303
# 5 other sequences           757    1406     907     2688
# 6 unassigned             1320419 2311326 1600886  3268646
# 7 unclassified sequences   15508   21572   14423    35024
```

Ce tableau est clair et lisible… seulement, est-il correctement encodé en cas par variables d’après vous ? Quelle que soit la réponse à cette question, il est toujours possible de passer de ce format large à un format long dans R de la façon suivante :

```
shotgun_long <- pivot_longer_(shotgun_wide, cols = c("c1", "c4", "c7", "c10"),
  names_to = "batch", values_to = "sequences")

DT::datatable(shotgun_long)
```

Show 10 entries Search:

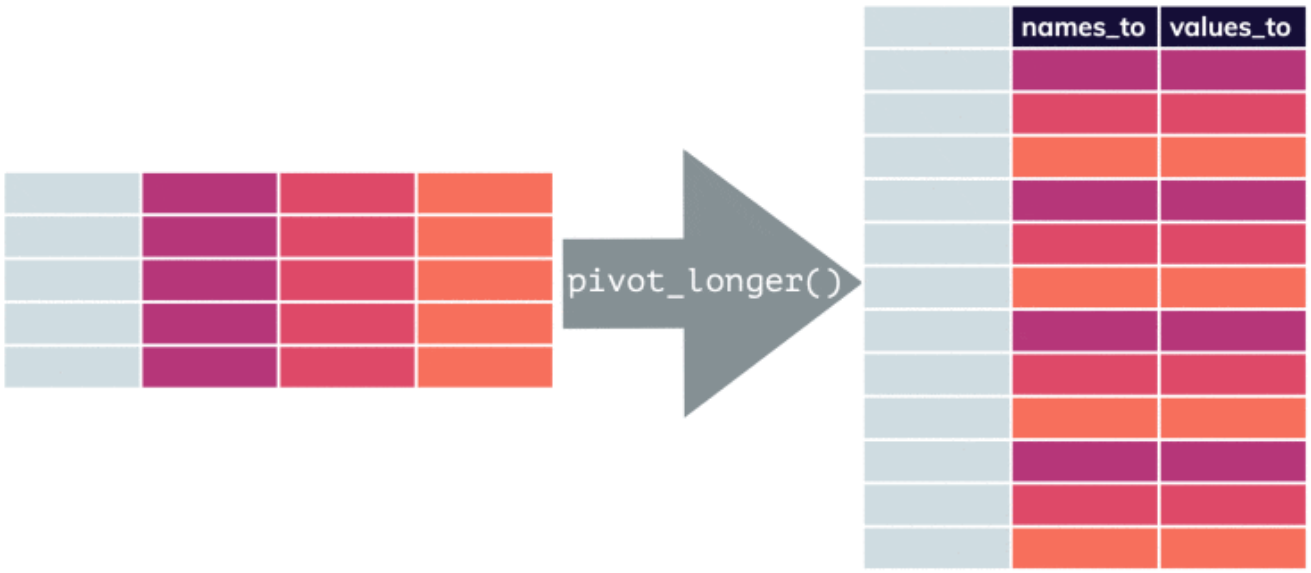
	kingdom	batch	sequences
1	Archaea	c1	98379
2	Archaea	c4	217985
3	Archaea	c7	143314
4	Archaea	c10	272541
5	Bacteria	c1	6665903
6	Bacteria	c4	9739134
7	Bacteria	c7	7103244
8	Bacteria	c10	15966053
9	Eukaryota	c1	81593
10	Eukaryota	c4	101834

Showing 1 to 10 of 28 entries Previous 1 2 3 Next

L'argument `cols =` est assez flexible. Toutes les formes ci-dessous permettent d'obtenir le même résultat.

```
shotgun_long <- pivot_longer_(shotgun_wide, cols = ~c1:c10, # Plage de variabl
  names_to = "batch", values_to = "sequences")
shotgun_long <- pivot_longer_(shotgun_wide, cols = 2:5, # Numero des colonnes
  names_to = "batch", values_to = "sequences")
shotgun_long <- pivot_longer_(shotgun_wide, cols = ~starts_with("c"), # Sélect
  names_to = "batch", values_to = "sequences")
```

Les fonctions spéciales de sélection de variables sont : `starts_with()`, `ends_with()`, `contains()`, `matches()`, et `num_range()`, voir `?tidyselect::starts_with`. Voici la logique derrière `pivot_longer()` du tidyverse, similaire à `pivot_longer_()` de `SciViews::R`, présentée sous forme d'une animation :

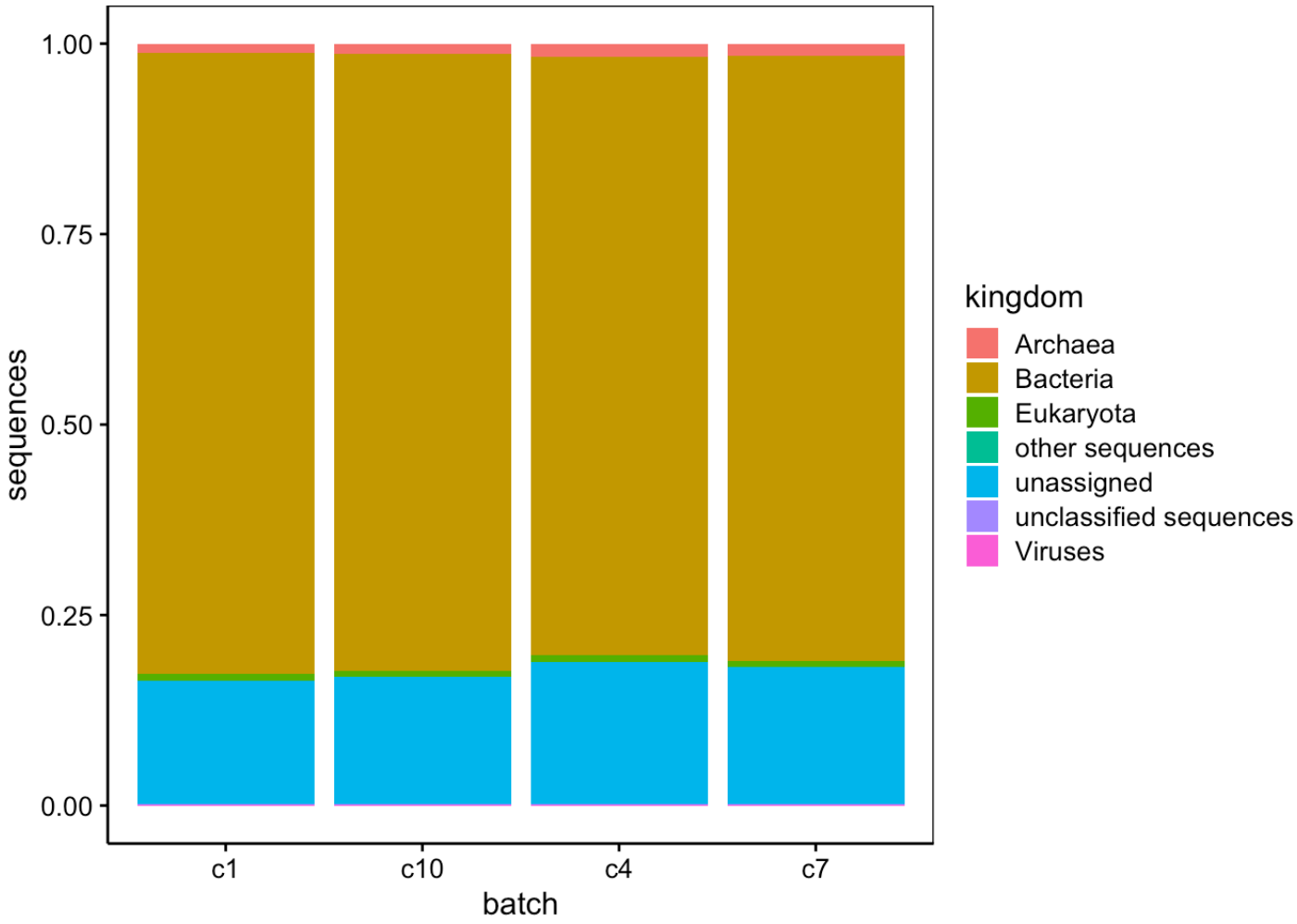


`pivot_longer()` par apreshill.

Vous conviendrez que le tableau nommé `shotgun_long` est moins compact et moins aisé à lire comparé à `shotgun_wide`. C'est une raison qui fait que beaucoup de scientifiques sont tentés d'utiliser le format large alors qu'ici il ne correspond **pas** à un tableau cas par variables correct, puisqu'il est impossible que les *mêmes* individus soient présents dans

les différents lots (il s'agit de communautés microbiennes *indépendantes* les unes des autres). De plus, seul le format `shotgun_long` permet de produire des graphiques pertinents[37].

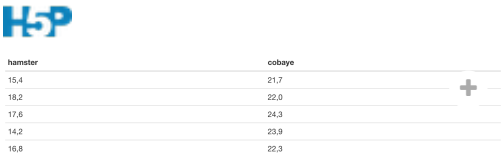
```
chart(data = shotgun_long, sequences ~ batch %fill=% kingdom) +
  geom_col(position = "fill")
```



Essayez de réaliser ce type de graphique en partant de `shotgun_wide` … Bonne chance !

> Très souvent, lorsqu'il est impossible de réaliser un graphique avec `chart()` ou `ggplot()` parce que les données se présentent mal, c'est parce que le jeu de données est encodé de manière incorrecte ! Si les données sont, par contre, correctement encodées, demandez-vous alors si le graphique que vous voulez faire est pertinent.

À vous de jouer !



hamster	cobaye
15,4	21,7
18,2	22,0
17,6	24,3
14,2	23,9
16,8	22,3

Complétez les blancs dans cette instruction R afin de convertir le tableau ci-dessus en un tableau long. Ce nouveau tableau doit contenir une variable **rongeur** pour regrouper les espèces et une variable **taille** pour représenter la taille des individus. Utilisez les fonctions svTidy terminées par un '_'.

rongeurs_long <- [ ](rongeurs, [ ] = c(hamster, cobaye), [ ] = "rongeur", [ ] = "taille")

Vérifier

Pour passer du format long au format large (traitement inverse à `pivot_longer_()` / `pivot_longer()`), il faut utiliser la fonction svTidy `pivot_wider_()` ou la fonction tidyverse `pivot_wider()` . Ainsi pour retrouver le tableau d'origine (ou quelque chose de très semblable) à partir de `shotgun_long` nous utiliserons :

```
shotgun_wide2 <- pivot_wider_(shotgun_long,
  names_from  = "batch", values_from = "sequences")

DT::datatable(shotgun_wide2)
```

Show 10 entries Search:

	kingdom	c1	c4	c7	c10
1	Archaea	98379	217985	143314	272541
2	Bacteria	6665903	9739134	7103244	15966053
3	Eukaryota	81593	101834	71111	150918
4	Viruses	1245	4867	5181	15303
5	other sequences	757	1406	907	2688
6	unassigned	1320419	2311326	1600886	3268646
7	unclassified sequences	15508	21572	14423	35024

Showing 1 to 7 of 7 entries Previous 1 Next

5.4.2 Recombinaison de variables

Parfois, ce sont les variables qui sont encodées de manière inappropriée par rapport aux analyses que vous souhaitez faire. Les fonctions svTidy `separate_()` et `unite_()` ou les fonctions tidyverse `separate()` et `unite()` permettent de séparer une colonne en plusieurs, ou inversement, de rassembler plusieurs colonnes en une seule.

> L'aide-mémoire Data tidying with tidyr vous rappelle ces fonctions dans sa section **Split Cells**. Elles sont également décrites en détail dans R for Data Science.

Partons, par exemple, du jeu de données sur la biométrie des crabes du package {MASS} :

```
crabs <- read("crabs", package = "MASS", lang = "fr")
DT::datatable(crabs)
```

Show 10 entries Search:

	species	sex	index	front	rear	length	width	depth
1	B	M	1	8.1	6.7	16.1	19	7
2	B	M	2	8.8	7.7	18.1	20.8	7.4
3	B	M	3	9.2	7.8	19	22.4	7.7
4	B	M	4	9.6	7.9	20.1	23.1	8.2
5	B	M	5	9.8	8	20.3	23	8.2
6	B	M	6	10.8	9	23	26.5	9.8
7	B	M	7	11.1	9.9	23.8	27.1	9.8
8	B	M	8	11.6	9.1	24.5	28.4	10.4
9	B	M	9	11.8	9.6	24.2	27.8	9.7
10	B	M	10	11.8	10.5	25.2	29.3	10.3

Showing 1 to 10 of 200 entries

Previous 1 2 3 4 5 … 20 Next

Les fonctions `unite_()` ou `unite()` permettent de combiner facilement les colonnes `sex` et `species` comme montré dans l'exemple ci-dessous. N'hésitez pas à faire appel à la page d'aide de la fonction via `?unite` pour vous guider.

```
crabs2 <- unite_(crabs, col = "sp_sex", "species", "sex", sep = "_")
DT::datatable(crabs2)
```

Show 10 entries Search:

	sp_sex	index	front	rear	length	width	depth
1	B_M	1	8.1	6.7	16.1	19	7
2	B_M	2	8.8	7.7	18.1	20.8	7.4
3	B_M	3	9.2	7.8	19	22.4	7.7
4	B_M	4	9.6	7.9	20.1	23.1	8.2
5	B_M	5	9.8	8	20.3	23	8.2
6	B_M	6	10.8	9	23	26.5	9.8
7	B_M	7	11.1	9.9	23.8	27.1	9.8
8	B_M	8	11.6	9.1	24.5	28.4	10.4
9	B_M	9	11.8	9.6	24.2	27.8	9.7
10	B_M	10	11.8	10.5	25.2	29.3	10.3

Showing 1 to 10 of 200 entries

Previous 1 2 3 4 5 … 20 Next

Les fonctions complémentaires à `unite_()` / `unite()` sont `separate_()` et `separate()`. Elles permettent de séparer une variable en deux ou plusieurs colonnes. Donc, pour retrouver un tableau similaire à celui d'origine, nous pourrons faire :

```
crabs3 <- separate_(crabs2, col = "sp_sex", into = c("species", "sex"), sep =
DT::datatable(crabs3)
```

Show 10 entries Search:

	species	sex	index	front	rear	length	width	depth
1	B	M	1	8.1	6.7	16.1	19	7
2	B	M	2	8.8	7.7	18.1	20.8	7.4
3	B	M	3	9.2	7.8	19	22.4	7.7
4	B	M	4	9.6	7.9	20.1	23.1	8.2
5	B	M	5	9.8	8	20.3	23	8.2
6	B	M	6	10.8	9	23	26.5	9.8
7	B	M	7	11.1	9.9	23.8	27.1	9.8
8	B	M	8	11.6	9.1	24.5	28.4	10.4
9	B	M	9	11.8	9.6	24.2	27.8	9.7
10	B	M	10	11.8	10.5	25.2	29.3	10.3

Showing 1 to 10 of 200 entries

Previous 1 2 3 4 5 … 20 Next

À vous de jouer !

Effectuez maintenant les exercices du tutoriel **A05La_recombination (Recombinaison de tableaux)**.

```
BioDataScience1::run("A05La_recombination")
```

36. Les analyses métagénomiques coûtent très cher. Il est souvent impossible de faire des réplicas. Un seul échantillon d'ADN a donc été séquencé ici pour chaque communauté.↩

37. Notez malgré tout que, à condition de bien en comprendre les implications, le format complémentaire peut se justifier dans une publication pour y présenter un tableau le plus lisible possible, ce qui est le cas ici. Mais pour les analyses, c'est le format qui correspond à un tableau cas par variables qui **doit** être utilisé.↩

5.5 Traitements multi-tableaux

Dans vos analyses, vous serez confronté à devoir gérer plusieurs tableaux que vous allez vouloir rassembler en un seul. Selon le travail à réaliser, il s'agit de coller les tableaux l'un au-dessus de l'autre, l'un à côté de l'autre, ou d'effectuer un travail de fusion plus complexe. Nous allons maintenant voir ces différents cas successivement.

> L'aide-mémoire Data Transformation with dplyr vous rappelle les différentes fonctions à utiliser dans sa section **Combine Tables**. Leur utilisation est également décrite en détails dans R for Data Science (2e).

5.5.1 Empilement vers le bas

Pour empiler des tableaux l'un au-dessus de l'autre, la fonction la plus simple est `bind_rows_()` (svTidy) ou `bind_rows()` (tidyverse). Partons de données mesurées dans des mésoscosmes récifaux. Les différentes variables mesurées sont les suivantes :

- les données **physico-chimiques** : la température, le pH, la salinité, l'oxygène dissous à l'aide, respectivement, d'un pH-mètre, d'un conductimètre et d'un oxymètre
- la concentration en **nutriments** : orthophosphates (PO_4^{3-}) et nitrates (NO_3^-) dissous dans l'eau par analyse colorimétrique

Pour la première série de mesures, des étudiants ont encodé deux tableaux qu'ils devront par la suite rassembler en un seul. Voici le premier tableau :

```
physico1 <- dtx(
  sample      = c("A0", "B0", "A0", "B0", "A0", "B0", "A0", "B0"),
  student     = c("st1", "st1", "st2", "st2", "st3", "st3", "st4", "st4"),
  ph          = c(7.94, 7.94, 7.94, 7.99, 7.94, 7.99, 7.94, 7.99),
  salinity    = c(34.0, 35.3, 33.9, 35.1, 34.0, 35.2, 33.9, 35.1),
  oxygen      = c(7.98, 8.00, 7.98, 7.98, 7.99, 7.86, 7.89, 7.98),
  temperature = c(24.6, 24.4, 25.1, 24.7, 24.9, 24.7, 25.0, 24.6))
DT::datatable(physico1)
```

Show 10 entries Search:

	sample	student	ph	salinity	oxygen	temperature
1	A0	st1	7.94	34	7.98	24.6
2	B0	st1	7.94	35.3	8	24.4
3	A0	st2	7.94	33.9	7.98	25.1
4	B0	st2	7.99	35.1	7.98	24.7
5	A0	st3	7.94	34	7.99	24.9
6	B0	st3	7.99	35.2	7.86	24.7
7	A0	st4	7.94	33.9	7.89	25
8	B0	st4	7.99	35.1	7.98	24.6

Showing 1 to 8 of 8 entries Previous 1 Next

Voici le second tableau :

```
physico2 <- dtx(
  sample      = c("A0", "B0", "A0", "B0"),
  student     = c( "st5", "st5", "st6", "st6"),
  ph          = c(7.94, 7.99, 7.93, 7.99),
  salinity    = c(33.8, 35.0, 33.9, 35.1),
  oxygen      = c(7.96, 8.01, 7.90, 8.00),
  temperature = c(25.0, 24.6, 24.0, 24.0))
DT::datatable(physico2)
```

Show 10 entries Search:

	sample	student	ph	salinity	oxygen	temperature
1	A0	st5	7.94	33.8	7.96	25
2	B0	st5	7.99	35	8.01	24.6
3	A0	st6	7.93	33.9	7.9	24
4	B0	st6	7.99	35.1	8	24

Showing 1 to 4 of 4 entries Previous 1 Next

L'empilement des deux tableaux de données en un seul se fait en utilisant `bind_rows_()` ou `bind_rows()` lorsque les tableaux contiennent les mêmes variables présentées exactement dans le même ordre comme ici :

```
physico <- bind_rows_(physico1, physico2)
DT::datatable(physico)
```

Show 10 entries Search:

	sample	student	ph	salinity	oxygen	temperature
1	A0	st1	7.94	34	7.98	24.6
2	B0	st1	7.94	35.3	8	24.4
3	A0	st2	7.94	33.9	7.98	25.1
4	B0	st2	7.99	35.1	7.98	24.7
5	A0	st3	7.94	34	7.99	24.9
6	B0	st3	7.99	35.2	7.86	24.7
7	A0	st4	7.94	33.9	7.89	25
8	B0	st4	7.99	35.1	7.98	24.6
9	A0	st5	7.94	33.8	7.96	25
10	B0	st5	7.99	35	8.01	24.6

Showing 1 to 10 of 12 entries Previous 1 2 Next

5.5.2 Empilement à droite

Pour combiner des tableaux de données par les colonnes, de gauche à droite, la fonction la plus simple à utiliser est la fonction svTidy `bind_cols_()` ou la fonction tidyverse `bind_cols()`. Les étudiants ont également réalisé des prélèvements d'eaux qui ont été dosés par colorimétrie avec un analyseur automatique. Les échantillons des deux groupes ont été mesurés dans la même série par l'appareil, ce qui donne le tableau suivant pour les nutriments :

```r
nutrients <- dtx(
  sample  = rep(c("A0", "B0"), times = 6),
  student = rep(c("st4", "st6", "st5", "st2", "st1", "st3"), each = 2),
  po4     = c(2.445, 0.374, 2.446, 0.394, 2.433, 0.361,
              2.441, 0.372, 2.438, 0.388, 2.445, 0.390),
  no3     = c(1.145, 0.104, 0.447, 0.066, 0.439, 0.093,
              0.477, 0.167, 0.443, 0.593, 0.450, 0.125))
DT::datatable(nutrients)
```

Show 10 entries Search:

	sample	student	po4	no3
1	A0	st4	2.445	1.145
2	B0	st4	0.374	0.104
3	A0	st6	2.446	0.447
4	B0	st6	0.394	0.066
5	A0	st5	2.433	0.439
6	B0	st5	0.361	0.093
7	A0	st2	2.441	0.477
8	B0	st2	0.372	0.167
9	A0	st1	2.438	0.443
10	B0	st1	0.388	0.593

Showing 1 to 10 of 12 entries Previous 1 2 Next

Vous devez être très vigilant lors de l'utilisation de, `bind_cols_() / bind_cols()` car ces fonctions combinent vos tableaux sans s'assurer que vos lignes soient alignées convenablement !

```
oceano <- bind_cols_(nutrients, physico)


# New names:
# • `sample` -> `sample...1`
# • `student` -> `student...2`
# • `sample` -> `sample...5`
# • `student` -> `student...6`


DT::datatable(oceano)
```

Show 10 entries Search:

	sample...1	student...2	po4	no3	sample...5	student...6	ph	s
1	A0	st4	2.445	1.145	A0	st1	7.94	
2	B0	st4	0.374	0.104	B0	st1	7.94	
3	A0	st6	2.446	0.447	A0	st2	7.94	
4	B0	st6	0.394	0.066	B0	st2	7.99	
5	A0	st5	2.433	0.439	A0	st3	7.94	
6	B0	st5	0.361	0.093	B0	st3	7.99	
7	A0	st2	2.441	0.477	A0	st4	7.94	
8	B0	st2	0.372	0.167	B0	st4	7.99	
9	A0	st1	2.438	0.443	A0	st5	7.94	
10	B0	st1	0.388	0.593	B0	st5	7.99	

Showing 1 to 10 of 12 entries Previous 1 2 Next

Qu’observez-vous (outre que la fonction a remanié les noms des colonnes dupliquées) ?

Effectivement nos deux tableaux de données n'ont pas les lignes dans le même ordre pour "student". Il faut être vigilant lors de ce genre de combinaison de tableaux. Il est préférable d'employer des fonctions de fusion de tableaux plus complexes comme `full_join_()` / `full_join()` (ci-dessous). Pour utiliser correctement `bind_cols_()` / `bind_cols()`, il faut vous assurer que les lignes des deux tableaux correspondent exactement, par exemple, en utilisant la fonction `arrange_()` ou `arrange()` pour trier les données par ordre alphabétique pour une ou plusieurs colonnes avant la fusion :

```
nutrients2 <- arrange_(nutrients, "student", "sample")
DT::datatable(nutrients2)
```

Show 10 entries Search:

	sample	student	po4	no3
1	A0	st1	2.438	0.443
2	B0	st1	0.388	0.593
3	A0	st2	2.441	0.477
4	B0	st2	0.372	0.167
5	A0	st3	2.445	0.45
6	B0	st3	0.39	0.125
7	A0	st4	2.445	1.145
8	B0	st4	0.374	0.104
9	A0	st5	2.433	0.439
10	B0	st5	0.361	0.093

Showing 1 to 10 of 12 entries Previous 1 2 Next

Le tableau `nutrients2` a maintenant les données présentées dans le même ordre (en lignes) que le tableau `physico`. Nous pouvons donc rassembler ces deux tableaux à l'aide de `bind_cols_()` :

```
oceano <- bind_cols_(nutrients2, physico)


# New names:
# • `sample` -> `sample...1`
# • `student` -> `student...2`
# • `sample` -> `sample...5`
# • `student` -> `student...6`


DT::datatable(oceano)
```

Show 10 entries Search:

	sample...1	student...2	po4	no3	sample...5	student...6	ph	s
1	A0	st1	2.438	0.443	A0	st1	7.94	
2	B0	st1	0.388	0.593	B0	st1	7.94	
3	A0	st2	2.441	0.477	A0	st2	7.94	
4	B0	st2	0.372	0.167	B0	st2	7.99	
5	A0	st3	2.445	0.45	A0	st3	7.94	
6	B0	st3	0.39	0.125	B0	st3	7.99	
7	A0	st4	2.445	1.145	A0	st4	7.94	
8	B0	st4	0.374	0.104	B0	st4	7.99	
9	A0	st5	2.433	0.439	A0	st5	7.94	
10	B0	st5	0.361	0.093	B0	st5	7.99	

Showing 1 to 10 of 12 entries Previous 1 2 Next

Après vérification de l'adéquation des lignes communes (`sample` et `student`), nous n'aurons plus besoin des colonnes `sample...5` et `student...6` . La vérification automatique à l'aide de code R et l'élimination de ces variables du tableau `oceano` vous sont laissées comme exercice…

5.5.3 Fusion de tableaux

La fusion fait intervenir une ou plusieurs colonnes communes des deux tableaux pour déterminer quelles lignes du premier correspondent aux lignes du second. Ainsi, la fusion des tableaux est assurée d'être réalisée correctement, quel que soit l'ordre des lignes dans les deux tableaux d'origine. Utilisons la fonction svTidy `full_join_()` en joignant les lignes en fonction des valeurs de `student` et `sample` :

```
oceano <- full_join_(nutrients, physico, by = c("student", "sample"))
DT::datatable(oceano)
```

Show 10 entries Search:

	sample	student	po4	no3	ph	salinity	oxygen	temperature
1	A0	st4	2.445	1.145	7.94	33.9	7.89	25
2	B0	st4	0.374	0.104	7.99	35.1	7.98	24.6
3	A0	st6	2.446	0.447	7.93	33.9	7.9	24
4	B0	st6	0.394	0.066	7.99	35.1	8	24
5	A0	st5	2.433	0.439	7.94	33.8	7.96	25
6	B0	st5	0.361	0.093	7.99	35	8.01	24.6
7	A0	st2	2.441	0.477	7.94	33.9	7.98	25.1
8	B0	st2	0.372	0.167	7.99	35.1	7.98	24.7
9	A0	st1	2.438	0.443	7.94	34	7.98	24.6
10	B0	st1	0.388	0.593	7.94	35.3	8	24.4

Showing 1 to 10 of 12 entries Previous 1 2 Next

Observez bien ce dernier tableau. Les données initiales ont été *retriées* avant d'être fusionnées pour que les données correspondent. Comparez au tableau `physico` d'origine réimprimé ci-dessous : ses lignes ont été triées pour correspondre à celles du premier tableau `nutrients` .

```
DT::datatable(physico)
```

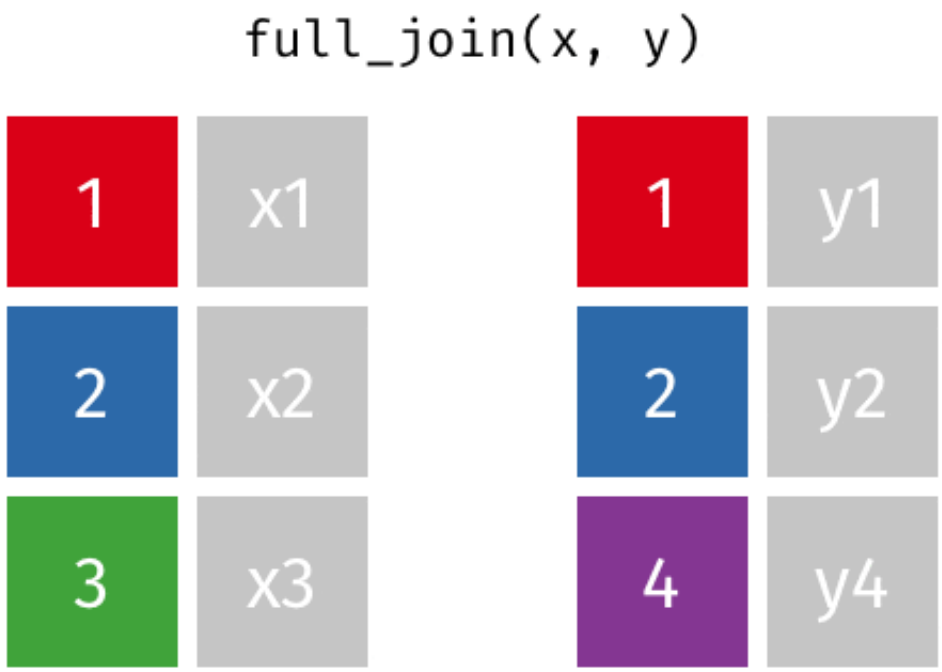
Show 10 entries Search:

	sample	student	ph	salinity	oxygen	temperature
1	A0	st1	7.94	34	7.98	24.6
2	B0	st1	7.94	35.3	8	24.4
3	A0	st2	7.94	33.9	7.98	25.1
4	B0	st2	7.99	35.1	7.98	24.7
5	A0	st3	7.94	34	7.99	24.9
6	B0	st3	7.99	35.2	7.86	24.7
7	A0	st4	7.94	33.9	7.89	25
8	B0	st4	7.99	35.1	7.98	24.6
9	A0	st5	7.94	33.8	7.96	25
10	B0	st5	7.99	35	8.01	24.6

Showing 1 to 10 of 12 entries Previous 1 2 Next

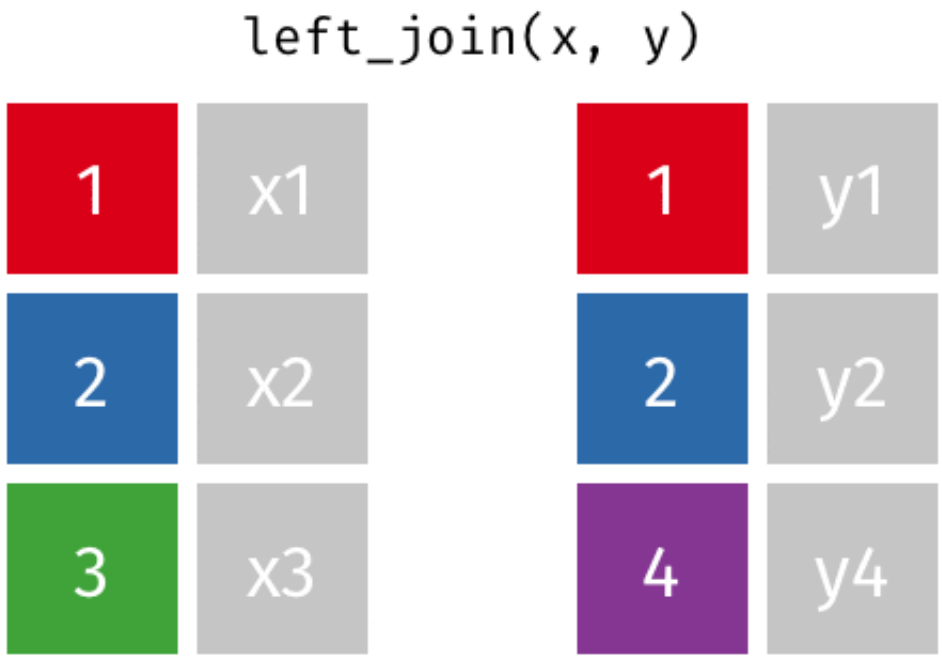
Il existe, en fait, plusieurs versions pour la fusion de tableaux, représentées par une série de fonctions tidyverse `XXX_join()` ou svTidy `XXX_join_()` . Lorsque les différentes lignes apparaissent exactement une fois dans chaque tableau comme dans l'exemple traité ici, les différentes variantes ont le même effet. Mais lorsque des lignes diffèrent ou manquent d'un côté ou de l'autre, les variantes ont leur importance :

- `full_join_()` / `full_join()` garde toutes les lignes,



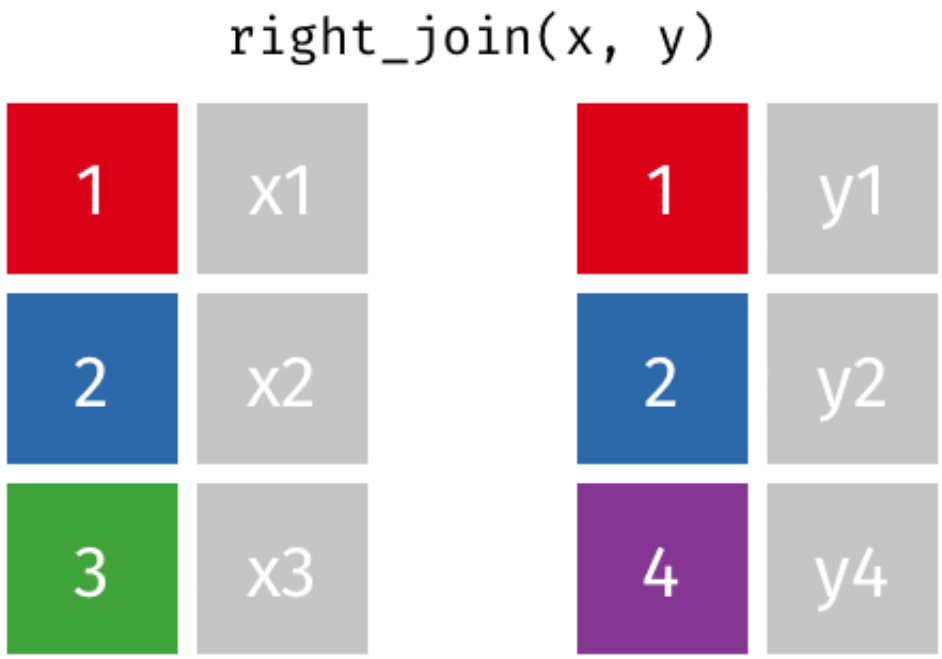
`full_join()` par gadenbuie.

- `left_join_() / left_join()` ne garde que les lignes uniques du tableau de gauche en plus des lignes communes,



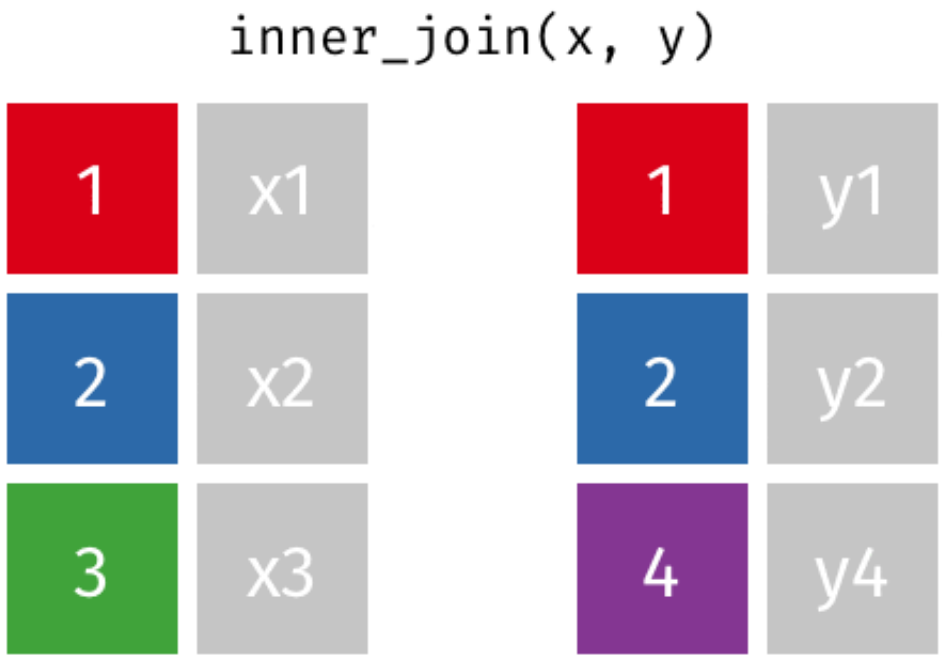
`left_join()` par gadenbuie.

- `right_join_()` / `right_join()` ne garde que les lignes uniques du tableau de droite en plus des lignes communes,



`right_join()` par gadenbuie.

- `inner_join_() / inner_join()` garde uniquement les lignes communes aux deux tableaux,



`inner_join()` par gadenbuie.

À vous de jouer !

Effectuez maintenant les exercices du tutoriel **A05Lb_multi_table (Traitements multi-tableaux)**.

```
BioDataScience1::run("A05Lb_multi_table")
```

Réalisez le travail **A05la_belgium_demo**.

Initiez votre projet GitHub Classroom

Voyez les explications dans le fichier `README.md` .

Effectuez également des remaniements de données multi-tableaux dans votre projet de groupe récurrent.

> Réalisez en groupe le travail **A02Ga_analysis, partie IV**.
>
> **Initiez votre projet GitHub Classroom**
>
> *Voyez les explications dans le fichier `README.md`, partie IV.*

Pour en savoir plus

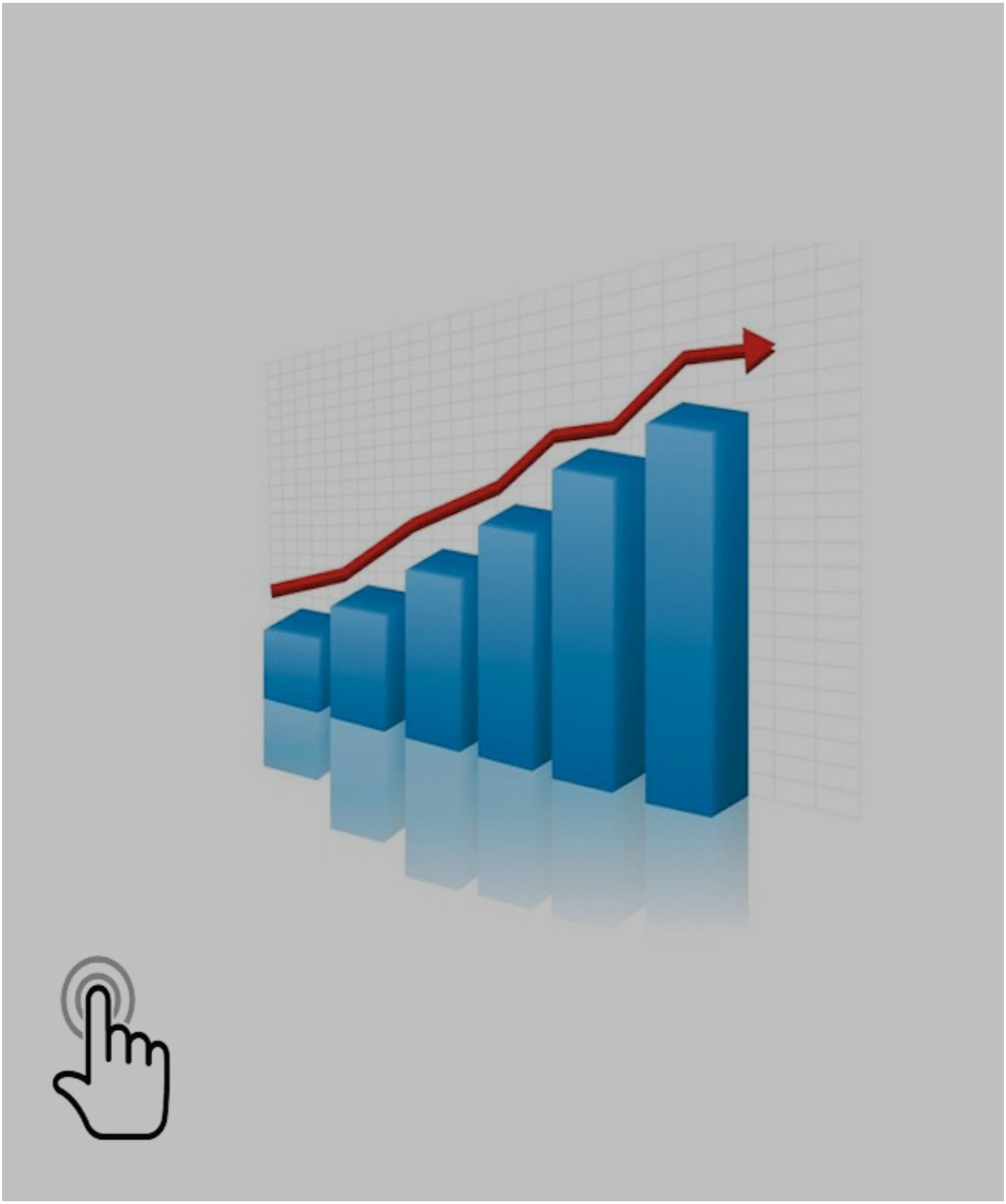
- Un aide-mémoire général en science des données avec lien vers les sites Web importants et les autres aide-mémoires. À partir de là, vous pouvez retrouver les packages, puis les fonctions qui réalisent telle ou telle transformation des données. Allez ensuite voir la page d'aide de la fonction avec `?ma_fonction`. Vous pouvez aussi rechercher sur https://stackoverflow.com, en précédent le nom de la fonction par "[R]" (R entre crochets) pour restreindre la recherche uniquement dans les pages relatives au langage R.
- Le chapitre relatif aux jointures de **R for Data Science (2e)** contient divers exemples de transformation des données à partir d'un ou plusieurs tableaux et présente aussi d'autres formes de jointures.

5.6 Récapitulatif des exercices

Vous venez de terminer le module 5 qui clôture la première partie du cours. Ce module vous a permis d'apprendre à présenter correctement des données (cas par variables ou tableau de contingence), à planifier vos mesures et préparer votre échantillonnage, à recombiner, remodeler et fusionner des tableaux. Pour évaluer votre compréhension de cette matière, vous aviez les exercices suivants à réaliser :

- A05Ha_cas_variables1 - Tableau cas par variables 1
- A05Hb_cas_variables2 - Tableau cas par variables 2
- A05Hc_cas_variables3 - Tableau cas par variables 3
- A05Hd_cas_variables4 - Tableau cas par variables 4
- A05He_contingence - Tableau de contingence
- A05Hf_table - Tableau de contingence dans R
- A05Hg_tidy_data - Données et métadonnées
- A05Hh_precision - Précision et exactitude
- A04Ga_biometry - Collecte de données relative à l'obésité
- A05Hi_pivot_longer - Transformation de long vers large
- A05La_recombination - Recombinaison de tableaux
- A05Lb_multi_table - Traitements multi-tableaux
- A05la_belgium_demo - Démographie en Belgique
- A02Ga_analysis - Analyse de données (partie IV)

Progression



Cliquez pour visualiser le rapport de progression.

Appendice A Installation de la SciViews Box

À partir de l'année académique 2022-2023, la SciViews Box s'utilise sur le Cloud avec SaturnCloud uniquement. Les instructions d'installation sont accessibles depuis la page qui s'affiche lorsque vous cliquez sur le bouton bleu RStudio en haut à droite des pages du site. Cet appendice n'a donc, pour l'instant, plus de raison d'être, mais il est conservé uniquement "pour mémoire".

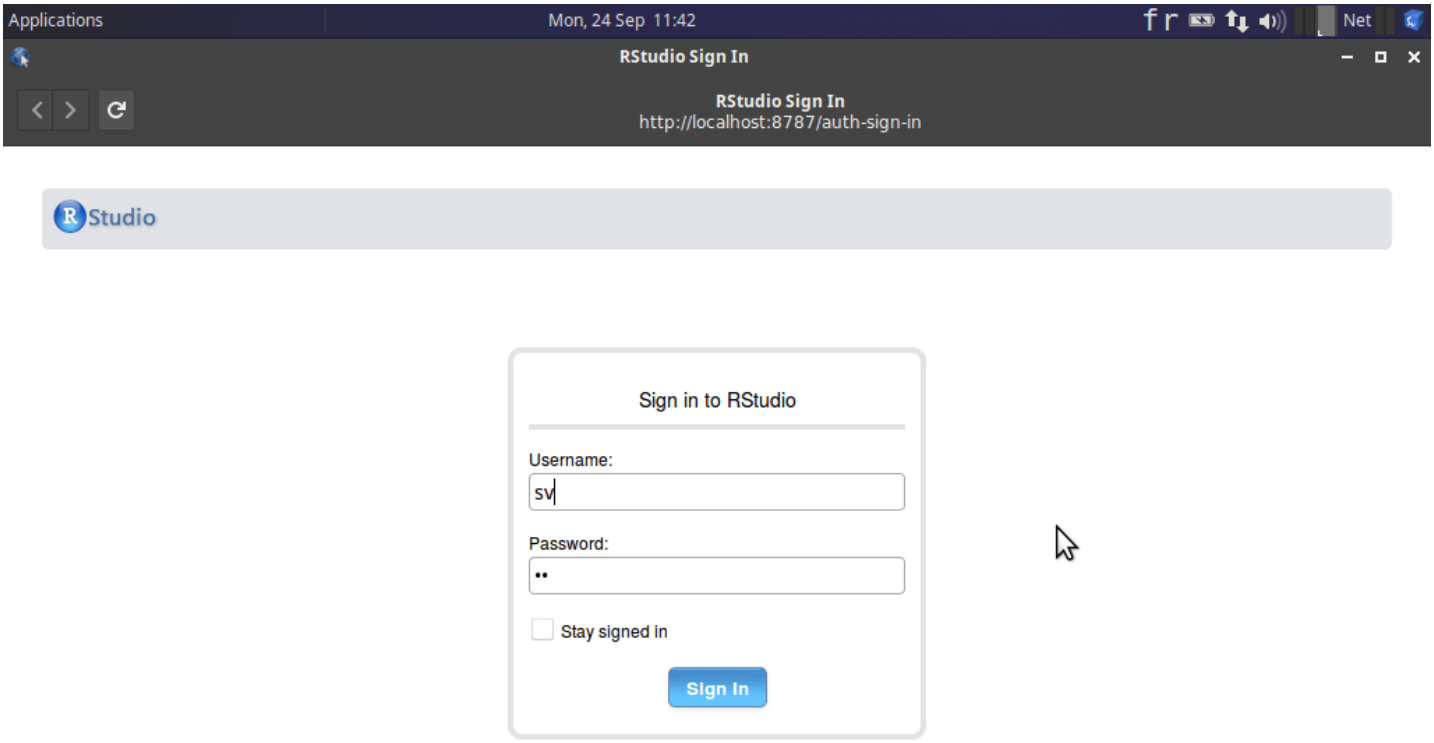
Appendice B Prise en main

Cette annexe comprend une description détaillée des différents outils utilisés dans le cadre de cette formation.

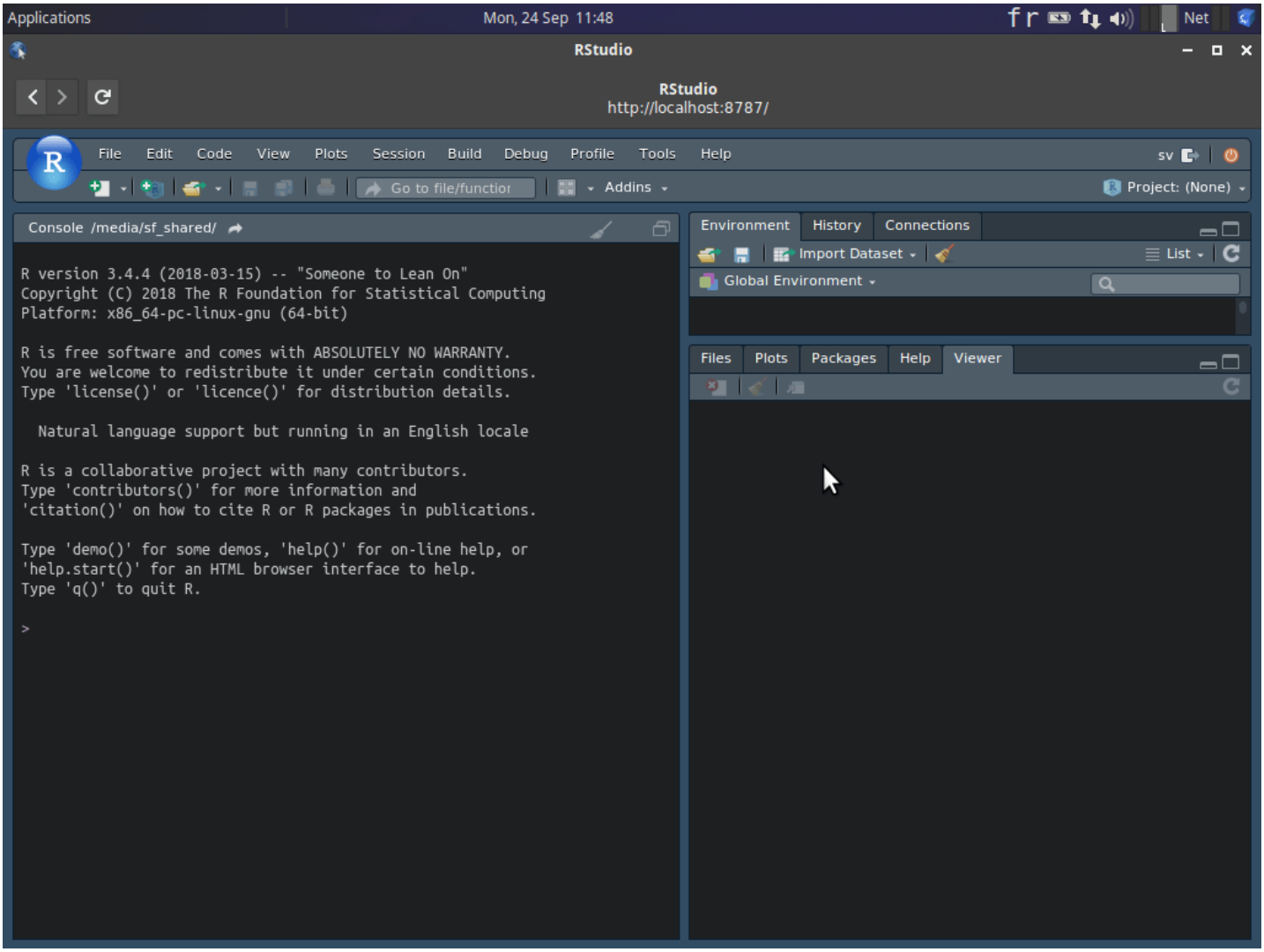
B.1 RStudio

Une fois que vous avez démarré la SciViews Box, vous pourrez accéder à **RStudio** qui est une interface conviviale pour manipuler des analyses statistiques dans le logiciel **R**. On parle de “IDE” pour *Integrated Development Environment* en anglais, ou *Environnement de développement intégré*. RStudio[65] permet de piloter vos analyses statistiques dans le logiciel **R**. En fait, **R** est un second logiciel, lancé en arrière-plan. C’est lui qui réalise effectivement les calculs demandés dans l’interface de RStudio. Si cela parait compliqué, ne vous inquiétez pas : cela se fait de manière totalement transparente. Si cela vous est demandé –dans la version **Server** de RStudio, mais pas dans la machine SaturnCloud– vous entrez votre login et mot de passe pour lancer votre session de travail. Dans la SciViews Box, ces informations sont :

- Username : **sv**
- Password : **sv**
- Cochez éventuellement **Stay signed in** pour éviter de devoir rentrer ces informations continuellement.

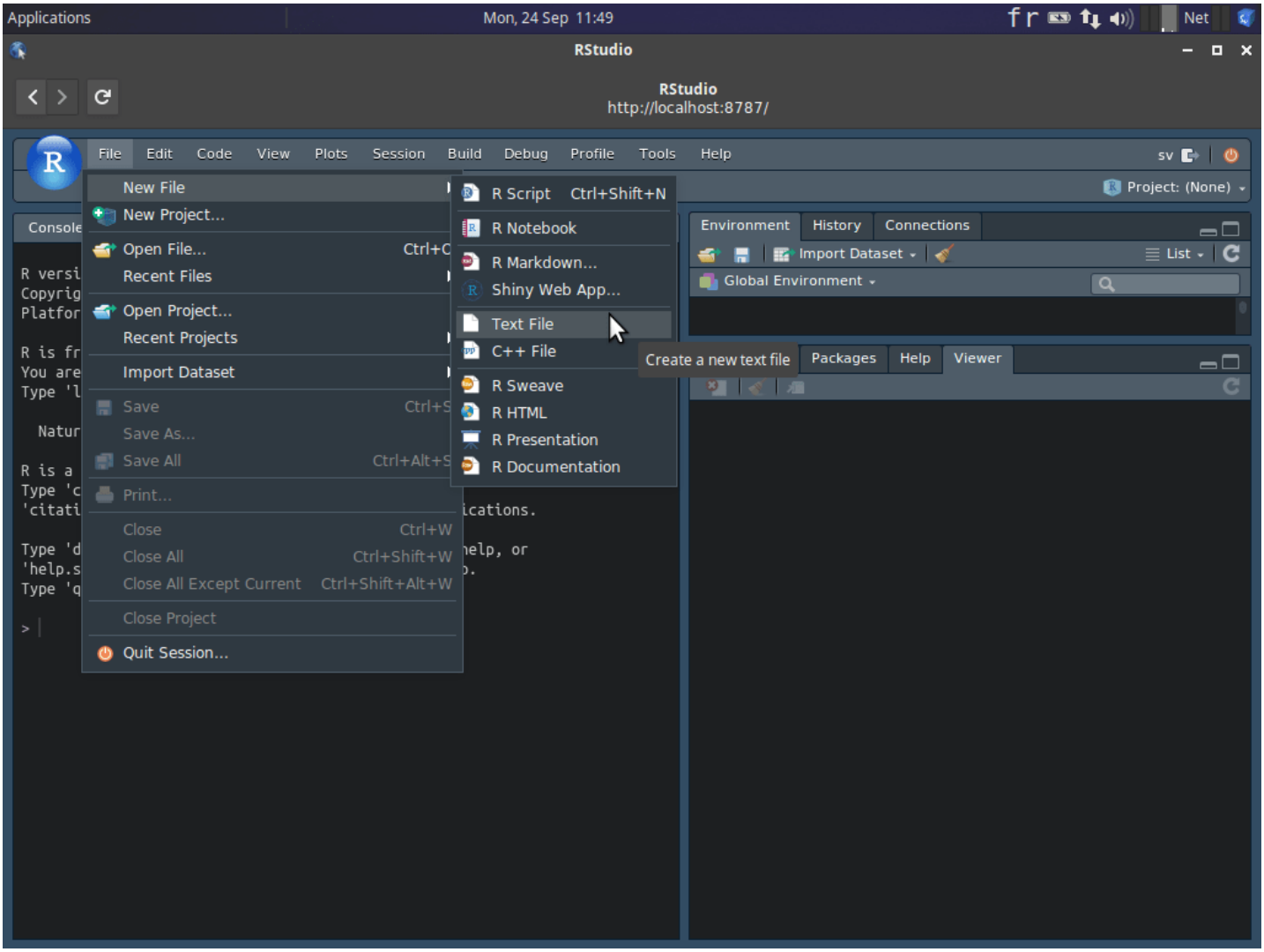


RStudio s’ouvre. La fenêtre principale comporte différents éléments :

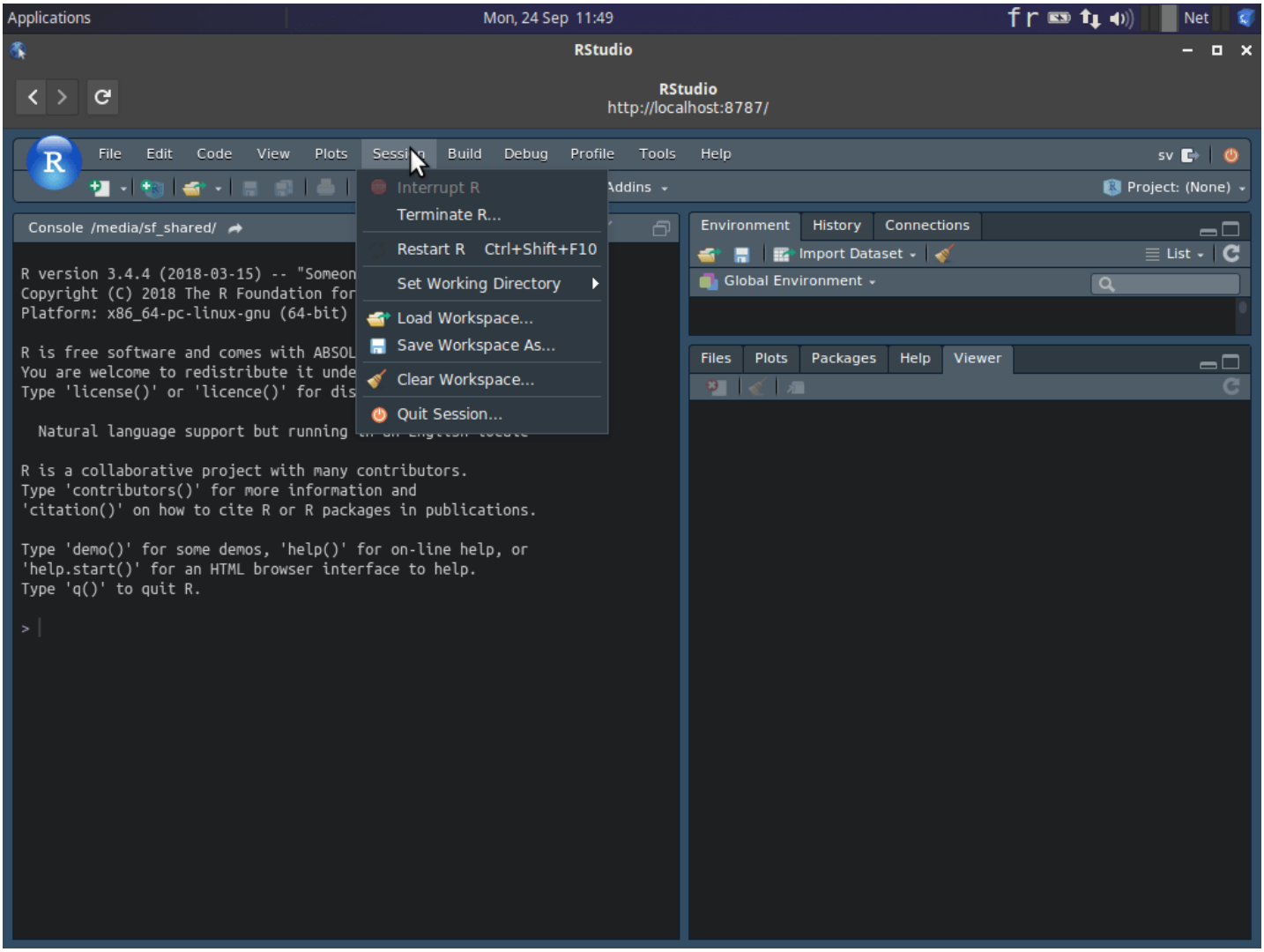


- Une *barre de menu* et une *barre d'outils générale* en haut
- Un *panneau* à gauche intitulé **Console** où vous pouvez entrer des instructions qui sont envoyées à **R** pour manipuler vos données
- Un panneau à droite en haut qui comprend plusieurs *onglets*, dont **Environment** (ou **Environnement** si vous utilisez RStudio en français, noté dorénavant *version anglaise* || *version française* comme **Environment || Environnement**) qui vous indique les différents items –on parle d'**objets**– chargés en mémoire dans **R**. Mais pour l'instant, il n'y a encore rien dans cet onglet.
- Un panneau en bas à droite comportant lui aussi plusieurs onglets. Vous devriez voir le contenu de **Files || Fichiers** au démarrage, un explorateur de fichiers simplifié.

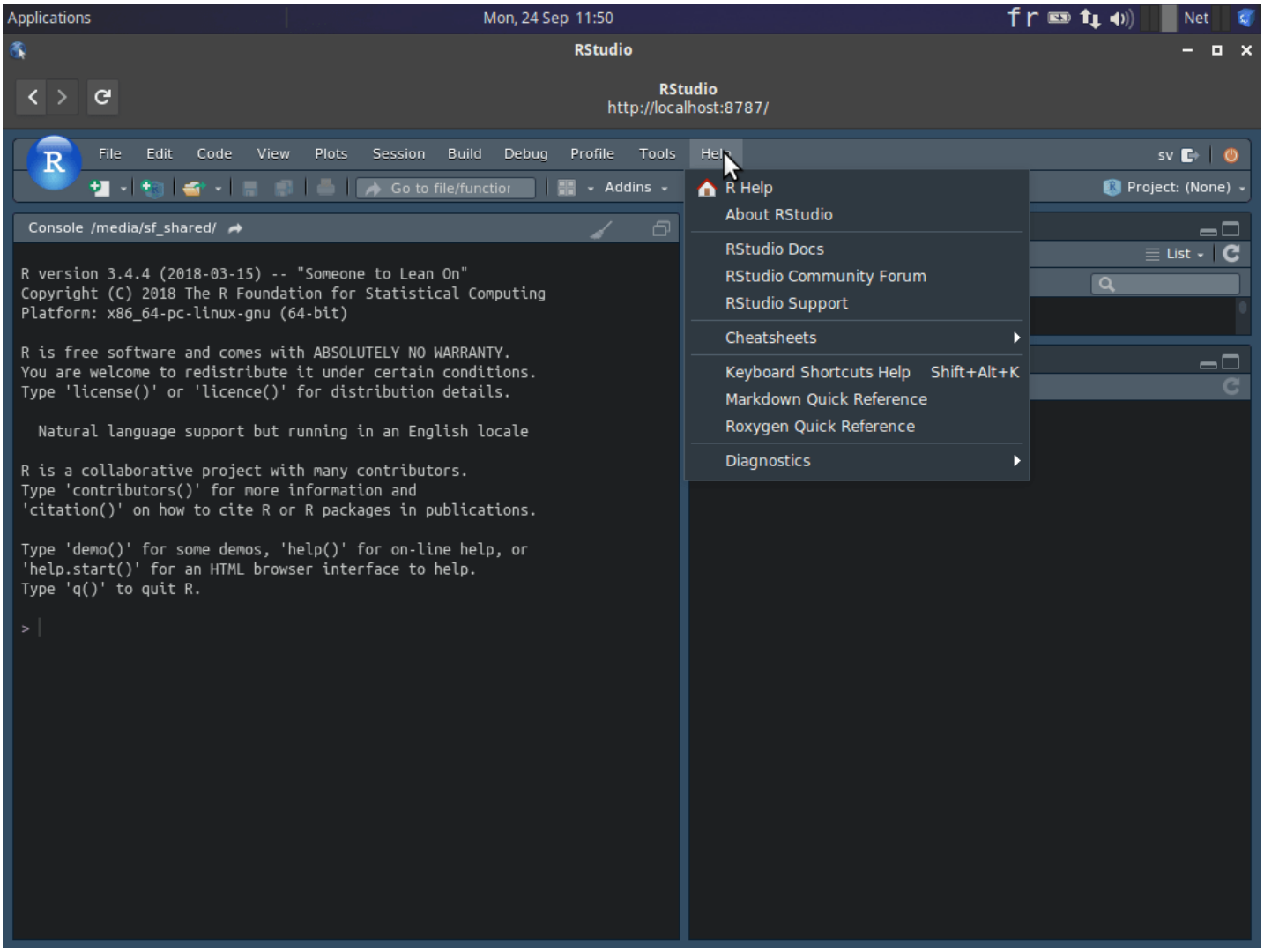
Pour l'instant, aucun document de travail n'est encore ouvert. Pour en créer un, ou ouvrir un document existant, vous utilisez le menu `Files || Fichiers` , ou encore, le premier bouton de la barre d'outils générale :



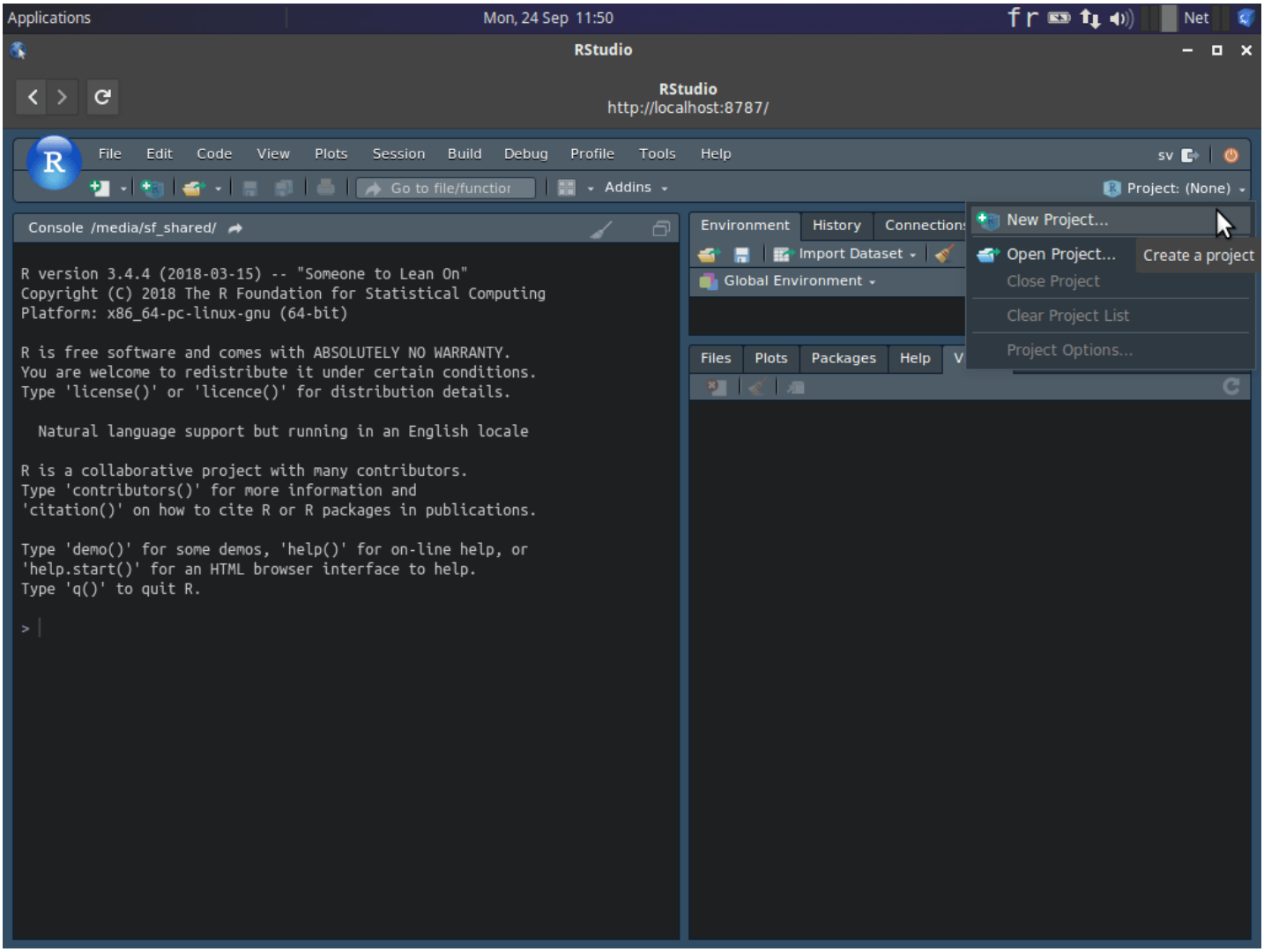
Le menu `Session` permet d'interagir directement avec **R**. Par exemple, il est possible de relancer **R** à partir d'une entrée de ce menu (`Session -> Restart R || Session -> Redémarrer R`) :



Le menu `Help || Aide` expose différentes entrées permettant d'accéder à la documentation de **R** ou de **RStudio**. Les **aide-mémoires** (`Help -> Cheat Sheets || Aide -> Cheat Sheets`) sont très pratiques lors de l'apprentissage. Nous conseillons de les consulter régulièrement et éventuellement, de les imprimer pour s'y référer plus facilement sans encombrer votre espace de travail dans votre ordinateur.



Le dernier item de la barre d'outils générale, intitulé **Project || Projet** permet d'ouvrir, fermer, et gérer les **projets** RStudio, c'est-à-dire, des espaces de travail organisés autour d'une analyse particulière, et qui peuvent contenir plusieurs fichiers et sous-dossiers (ce sera détaillé ci-dessous).

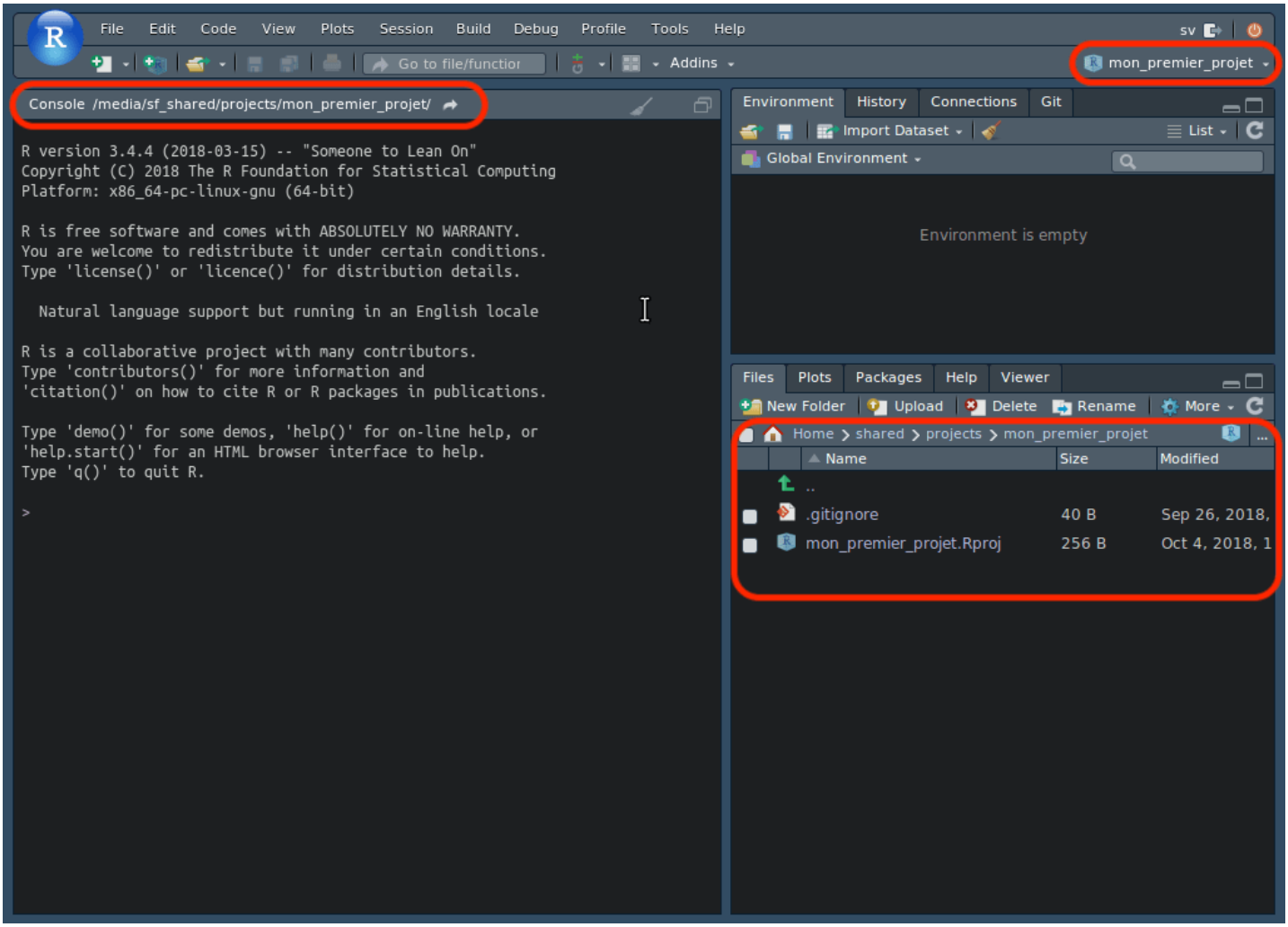


Vous avez maintenant repéré les éléments les plus importants de l'interface de RStudio.

À ce stade, vous pouvez vous familiariser avec l'aide-mémoire relatif à l'IDE RStudio. Vous verrez qu'il y a beaucoup de fonctionnalités accessibles à partir de sa fenêtre principale. Ne vous laissez pas intimider : vous les apprendrez progressivement au fur et à mesure de votre utilisation du logiciel.

B.1.1 Projet dans RStudio

Un **projet** sert, dans RStudio, à organiser son travail. Il va regrouper l'ensemble des **jeux de données**, des **rapports**, des **présentations**, des **scripts** (fichiers composés d'une série d'instructions réalisant un traitement particulier) d'une analyse, généralement en relation avec une ou plusieurs expériences ou observations réalisées sur le terrain ou en laboratoire. Voici à quoi ressemble l'interface de RStudio lorsque vous ouvrez un projet :



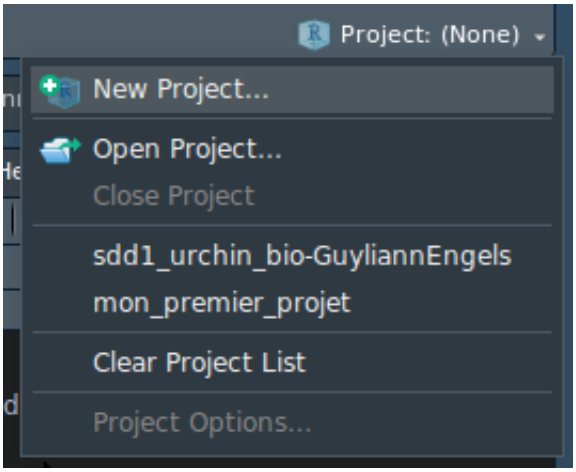
Notez que le **nom du projet** est mentionné en haut à droite. Notez également que le répertoire de base de votre projet est le répertoire actif dans l'onglet **Console** (`/media/shared/projects/mon_premier_projet/` dans l'exemple, mais dans SaturnCloud ce sera probablement plutôt `~/workspace/mon_premier_projet`). Remarquez aussi que l'onglet **Files || Fichiers** affiche son contenu. Un fichier `mon_premier_projet.Rproj` y est créé automatiquement par RStudio. Ce fichier contient les paramètres de configuration propres à ce projet[66]. C'est aussi une excellente façon de repérer qu'un répertoire est la base d'un projet RStudio, en repérant ce fameux fichier `.Rproj` .

B.1.1.1 Création d'un projet

Créez votre premier projet en suivant les quatre étapes suivantes :

- **Étape 1 :** dans RStudio, cliquez sur l'item en haut à droite dans la barre d'outils générale de RStudio qui ouvre un menu relatif aux projets. Sélectionnez-y l'entrée `New Project... || Nouveau projet.. .`
- **Étape 2 :** une boite de dialogue s'ouvre. Sélectionnez `New Directory || Nouveau répertoire` pour créer votre projet dans un nouveau dossier. Il est également possible d'employer un dossier existant comme point de départ `Existing Directory || Répertoire existant`).
- **Étape 3 :** sélectionnez `New Project || Nouveau projet` tout en haut dans la boite de dialogue suivante qui vous propose également des projets particuliers (que nous n'utiliserons pas pour l'instant).
- **Étape 4 :** RStudio vous demande quelques informations pour préconfigurer votre projet.
    - Nommez le projet dans `Directory name || Nom du répertoire` . Indiquez ici `project_test`
    - Indiquez où vous voulez le placer dans `Create project as subdirectory of || Créer un projet en tant que sous-répertoire de` . Sélectionnez le sous-dossier `workspace` dans Saturn Cloud ou le sous-dossier `projects` dans le dossier `shared` partagé entre la SciViews Box et la machine hôte si vous travaillez dans VirtualBox.
    - Cochez l'option `Create a git repository || Créer un dépôt git` .
    - Assurez-vous que l'option `Use renv with this project || Utiliser renv avec ce projet` soit bien **décochée** (**il est très important de ne pas sélectionner renv, sous peine de réinstaller et dupliquer de nombreux dossiers et fichiers –appelés "packages" ou "paquets" R– dans votre projet**)

Vous utilisez le même menu déroulant `Project || Projet` en haut à droite de la barre d'outils générale pour réouvrir un projet existant (`Open Project... || Ouvrir le projet...`) ou fermer le projet actuel (`Close Project || Fermer le projet`). Vous remarquez également que les derniers projets visités sont listés en dessous, ce qui permet d'y accéder plus rapidement.



> **Un projet ne doit bien sûr être créé qu'une seule fois !** Une fois les étapes ci-dessus effectuées, vous retournez simplement à votre projet en ouvrant le menu projets en haut à droite et en sélectionnant votre projet dans la liste. S'il n'y apparaît pas, choisissez `Open Project… || Ouvrir le projet…` et sélectionnez le fichier `.Rproj` relatif à votre projet. Ne créez bien évidemment **jamais** de projet à l'intérieur des dossiers d'un autre projet, surtout si vous utilisez Git. Sinon, RStudio va s'emmêler les pinceaux !

B.1.1.2 Organisation d'un projet

Le dossier `workspace` (Saturn Cloud) ou `projects` (SciViews Box dans VirtualBox) contient maintenant un projet RStudio intitulé `project_test` . Depuis la SciViews Box, il se situe dans (version VirtualBox représentée ci-dessous) :

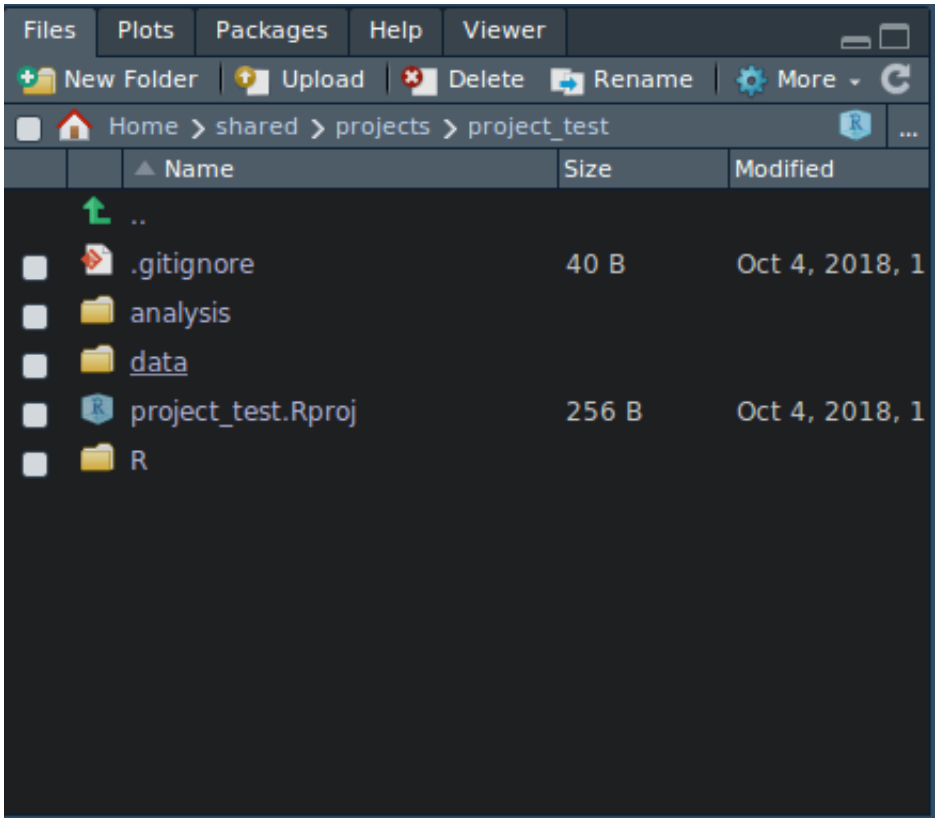
```
/home
  /sv
    /shared
      /projects
        /project_test          # Répertoire de base du projet
          project_test.Rproj   # Fichier de configuration du projet RStudio
          .gitignore           # Fichier relatif à la gestion de version
```

Vous devez maintenant structurer votre projet afin d'avoir différents sous-dossiers pour organiser au mieux votre travail. Ceci concerne à la fois les données et les rapports d'analyse en lien avec ce projet.

Cliquez sur le bouton `New Folder || Nouveau dossier` dans la barre d'outils de l'onglet **`Files || Fichiers`** et appelez ce nouveau dossier `data`. Ajoutez également le dossier `R` de la même manière.

```
/home
  /sv
    /shared
      /projects
        /project_test           # Répertoire de base du projet
           data                 # Dossier contenant les données
           project_test.Rproj   # Fichier de configuration du projet RStudio
           .gitignore           # Fichier relatif à la gestion de version
           R                    # Dossier contenant les scripts d'analyse
```

Vous obtenez donc un projet configuré de la manière suivante :



L'organisation cohérente d'un projet est indispensable pour le bon fonctionnement et la clarté de vos analyses de données.

B.1.1.3 Chemins relatifs dans un projet

L'utilisation d'un projet permet de structurer de manière cohérente son travail. Vous allez maintenant devoir rendre votre projet **portable**.

Un projet RStudio pourra être qualifié de *portable* s'il est possible de déplacer son répertoire de base et tout ce qu'il contient (ou le renommer) sans que les analyses qu'il réalise n'en soient affectées. Ceci est utile pour copier, par exemple, le projet d'un ordinateur à un autre, ou si vous décidez de restructurer vos fichiers sur le disque dur.

- La première règle est de placer tous les fichiers nécessaires dans le dossier du projet ou dans un sous-dossier. C'est ce que nous venons de faire plus haut.
- La seconde règle est de référencer les différents fichiers au sein du projet avec des **chemins relatifs**. Nous allons maintenant apprendre à faire cela. Partons d'un projet qui contient les dossiers et fichiers suivants :

```
/home
  /sv
    /shared
      /projects
        /project_test             # Répertoire de base du projet
           analysis               # Dossier contenant les analyses
                rapport_test.Rmd  # Rapport d'analyse
           data                   # Dossier contenant les données
                dataset.csv       # Jeu de données
           project_test.Rproj     # Fichier de configuration du projet RStudio
           .gitignore             # Fichier relatif à la gestion de version
           R                      # Dossier contenant les scripts d'analyse
```

Les différents systèmes d'exploitation (Windows, macOS, Linux) utilisent des conventions différentes pour les chemins d'accès aux fichiers. Dans notre cas, la machine virtuelle dans SaturnCloud ou VirtualBox utilise un système d'exploitation **Linux**. La barre oblique (`/` dite "slash" en anglais) sépare les différents dossiers imbriqués sous Linux et sous macOS. Le système d'exploitation Windows utilise, pour sa part, la barre oblique inverse (`\` , dite "backslash" en anglais, mais dans R et RStudio, vous pourrez également utiliser le slash `/` , ce que nous vous conseillons de faire toujours pour un maximum de compatibilité entre systèmes). Par exemple, votre fichier `dataset.csv` qui contient les données du projet hypothétique d'exemple se référence comme suit dans VirtualBox, donc sous Linux :

```
/home/sv/shared/projects/project_test/data/dataset.csv
```

Ce chemin d'accès est le plus détaillé. Il est dit **chemin d'accès absolu** au fichier. Vous noterez qu'il est totalement dépendant de la structure actuelle des dossiers sur le disque. Si vous renommez `project_test` ou si vous le déplacez ailleurs, la référence au fichier sera cassée ! Ainsi, si vous partagez votre projet, généré avec une SciViews Box fonctionnant dans VirtualBox avec un collaborateur qui utilise SaturnCloud, il installera le projet probablement dans `/home/jovyan/workspace` [67]. Par conséquent, le chemin d'accès aux données devra être adapté en `/home/jovyan/workspace/project_test/data/dataset.csv` , sans quoi l'analyse ne pourra plus s'exécuter correctement chez lui.

Décodons ce chemin d'accès :

- `/` , racine du système
- `/home/sv/` , notre dossier personnel comme utilisateur `sv`
- `/home/sv/shared/` , le dossier partagé entre la SciViews Box dans VirtualBox et notre ordinateur hôte
- `/home/sv/shared/projects/project_test/` , le dossier de base de notre projet
- `/home/sv/shared/projects/project_test/data/` , le répertoire qui contient le fichier `dataset.csv` .

Le répertoire utilisateur `/home/<user>` est différent sous macOS (il s'appelle `/Users/<user>`) et sous Windows (il se nomme généralement `C:\Users\<user>`). Comme c'est un répertoire clé, et qu'il est impossible d'écrire un chemin absolu qui soit le même partout, il existe un raccourci : le "tilde" (`~`) qui signifie "mon répertoire utilisateur". Ainsi, vous pouvez aussi accéder à votre jeu de données `dataset.csv` comme ceci dans VirtualBox :

```
~/shared/projects/project_test/data/datasets.csv
```

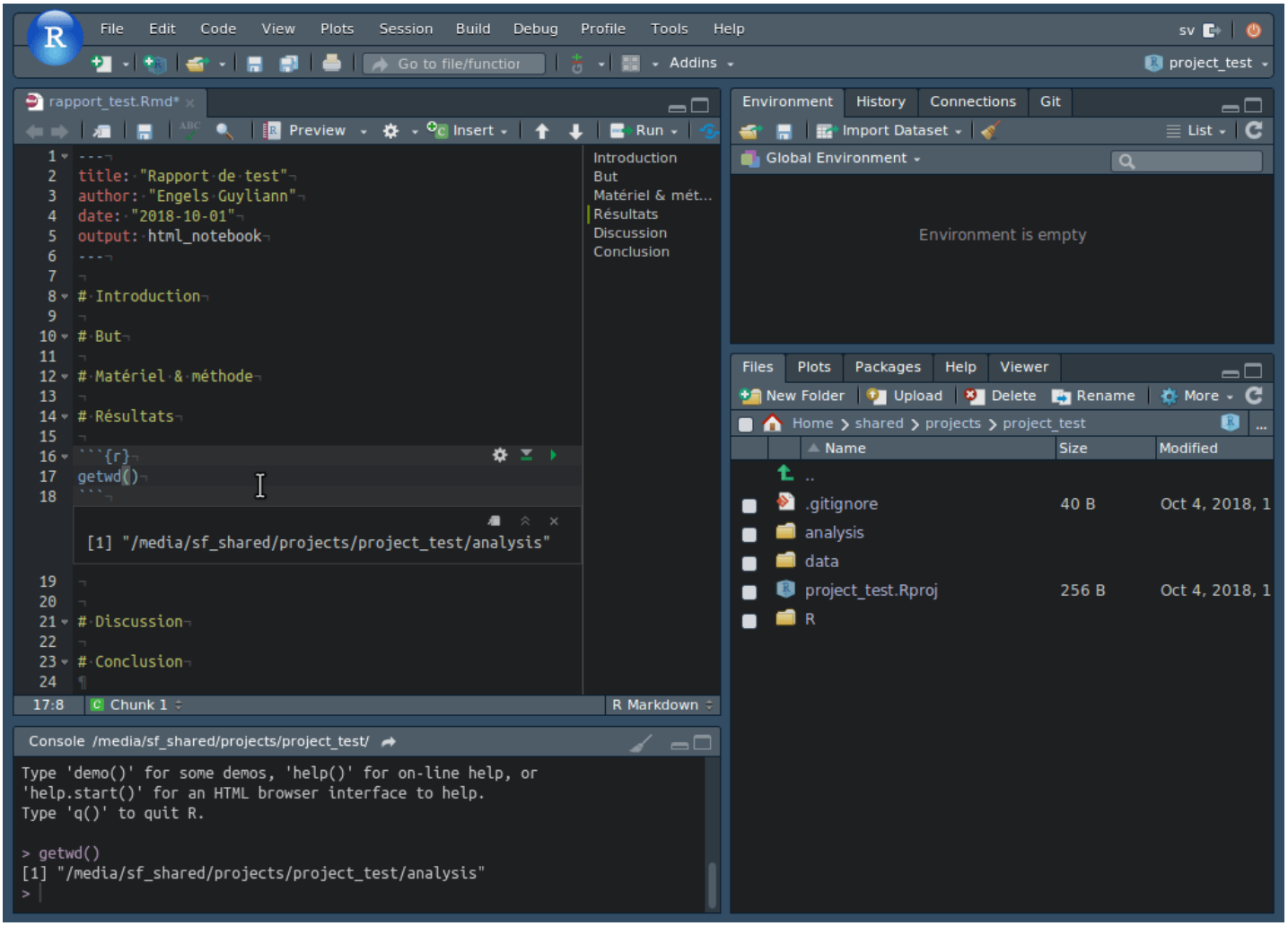
Ce chemin d'accès est déjà plus "portable" d'un système à l'autre et d'un utilisateur à l'autre. Il est donc à préférer[68]. Si cette façon d'écrire le chemin d'accès est compatible entre les trois systèmes d'exploitation, elle ne permet toujours pas de déplacer ou de renommer notre projet.

L'utilisation d'un **chemin relatif** permet de définir la position d'un fichier par rapport à un autre dossier qui est dit **"répertoire actif"**. A titre d'exemple, nous voulons faire référence au jeu de données `dataset.csv` depuis notre rapport `rapport_test.Rmd` .

Demandez-vous d'abord quel est le **répertoire actif**. Pour un fichier R Markdown ou R Notebook, c'est facile, c'est le dossier qui contient ce fichier. Dans la console R, cela peut varier selon le contexte. Si vous avez ouvert un projet, c'est le répertoire de base du projet par défaut, mais cela peut être modifié. **Le répertoire actif pour la console R est toujours indiqué en première ligne à côté de l'icône de R et de son numéro de version.** Vous pouvez aussi interroger R à l'aide de l'instruction `getwd()` qui vous indiquera alors quel est le répertoire actif :

```
getwd()
```

Vous apprendrez que vous pouvez réaliser cela dans un **script R** (fichier regroupant une succession d'instructions), ou dans un **chunk R** (zone spéciale contenant des instructions à exécuter dans votre document) dans votre document R Markdown :



Une fois que vous connaissez le répertoire actif, vous naviguez *à partir de* celui-ci. Il existe une convention pour reculer d'un dossier dans la hiérarchie. Pour cela vous indiquez `..` à la place d'un nom de dossier. Voici ce que cela donne :

```
../data/dataset.csv
```

C'est un **chemin relatif**. Comment le lit-on ? Tout d'abord, notez –c'est très important– que le chemin d'accès ne commence pas par `/` (Linux ou macOS), ou `C:\` (ou toute autre lettre, sous Windows), ni par `~`. C'est le signe que l'on ne part pas de la racine du système de fichier ou du dossier de l'utilisateur, mais du **répertoire actif**. Ensuite, les différents éléments se décryptent comme suit :

- `~/shared/projects/project_test/analysis`, répertoire actif au départ pour le document R Notebook
- `..` retourne en arrière d'un niveau. On est donc dans

`~/shared/projects/project_test`

- `/data` navigue dans le sous-dossier `data`. On est donc maintenant dans `~/shared/projects/project_test/data`. C'est le répertoire qui contient le fichier qui nous intéresse
- `/datasets.csv` pointe bien vers le fichier qui nous intéresse.

À noter que si le fichier se trouve déjà dans le répertoire actif, le chemin relatif se résume au nom du fichier directement. C'est donc très simple dans ce cas.

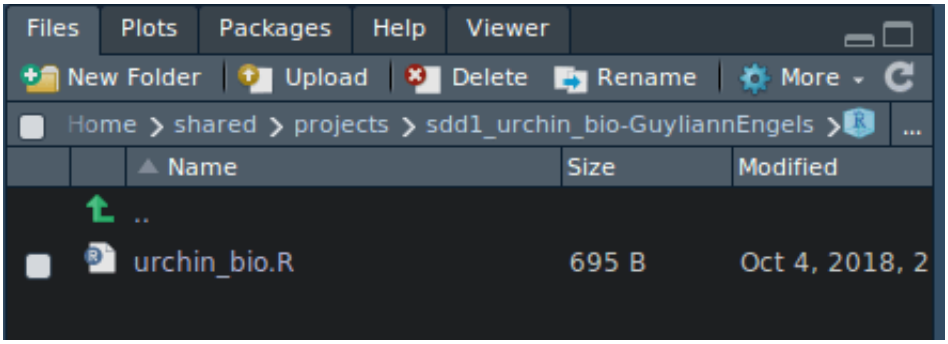
Nulle part dans ce chemin relatif n'apparaît le nom du dossier de base du projet, ni aucun autre dossier parent. Ainsi, il est possible de renommer ou déplacer le projet sans casser la référence relative à n'importe quel fichier à l'intérieur de ce projet. Donc, en utilisant uniquement des références relatives, **le projet reste parfaitement portable**, y compris entre SaturnCloud et VirtualBox, par exemple.

B.1.1.4 Chemins portables avec here::here()

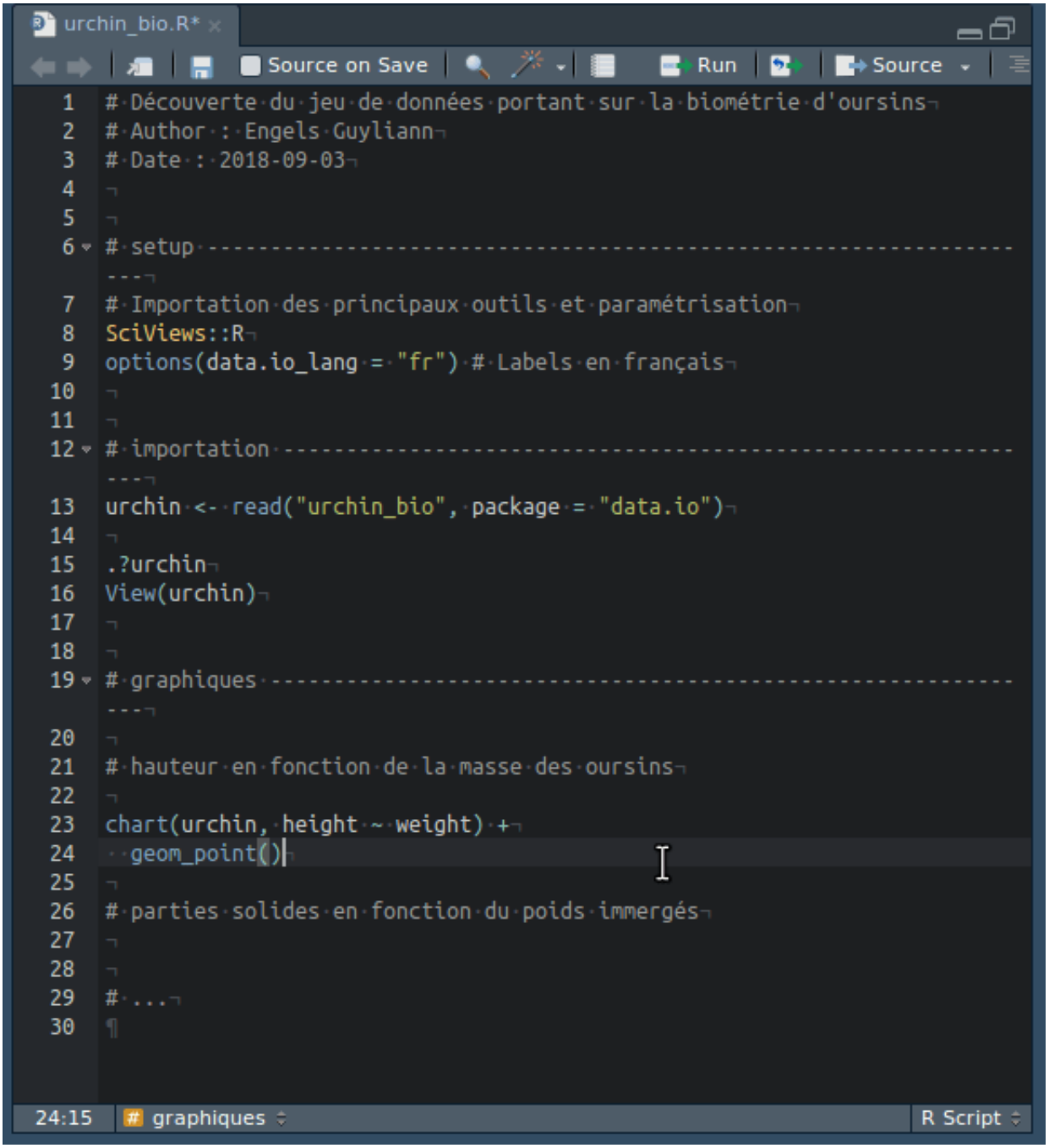
Il existe un autre mécanisme qui permet de référencer vos fichiers à l'intérieur de votre projet tout en assurant sa portabilité : la fonction `here::here()`. Quel que soit l'endroit où vous vous situez dans le projet, ou dans n'importe lequel de ses sous-dossiers, cette fonction pointera toujours vers le dossier de base du projet (projet en cours s'il y en a un ouvert dans RStudio, ou projet recherché en amont dans la hiérarchie de fichiers à partir du répertoire actif dans le cas contraire), donc dans notre exemple, `/home/sv/shared/projects/project_test`. La fonction accepte un ou plusieurs arguments sous forme de chaînes de caractères entre guillemets et séparés par une virgule qui complètent le chemin d'accès depuis ce répertoire de base. Donc, vous pouvez écrire `here::here("data", "datasets.csv)`. Cela produira `/home/sv/shared/projects/project_test/data/datasets.csv` qui est bien le chemin d'accès souhaité. Comme ce chemin d'accès est construit à la demande, la première partie correspondant au répertoire de base sera toujours bien calculée, même si le projet est déplacé ou renommé. Notez que vous auriez très bien pu aussi indiquer `here::here("data/datasets.csv)`, cela fonctionnera aussi.

B.1.2 Scripts R dans RStudio

Un script R est une suite d'instructions qui peuvent être interprétées pour effectuer nos analyses. Ce script est stocké dans un fichier dont l'extension est `.R` (attention : "R" majuscule), et que l'on placera de préférence dans le sous-dossier `R` de notre projet.



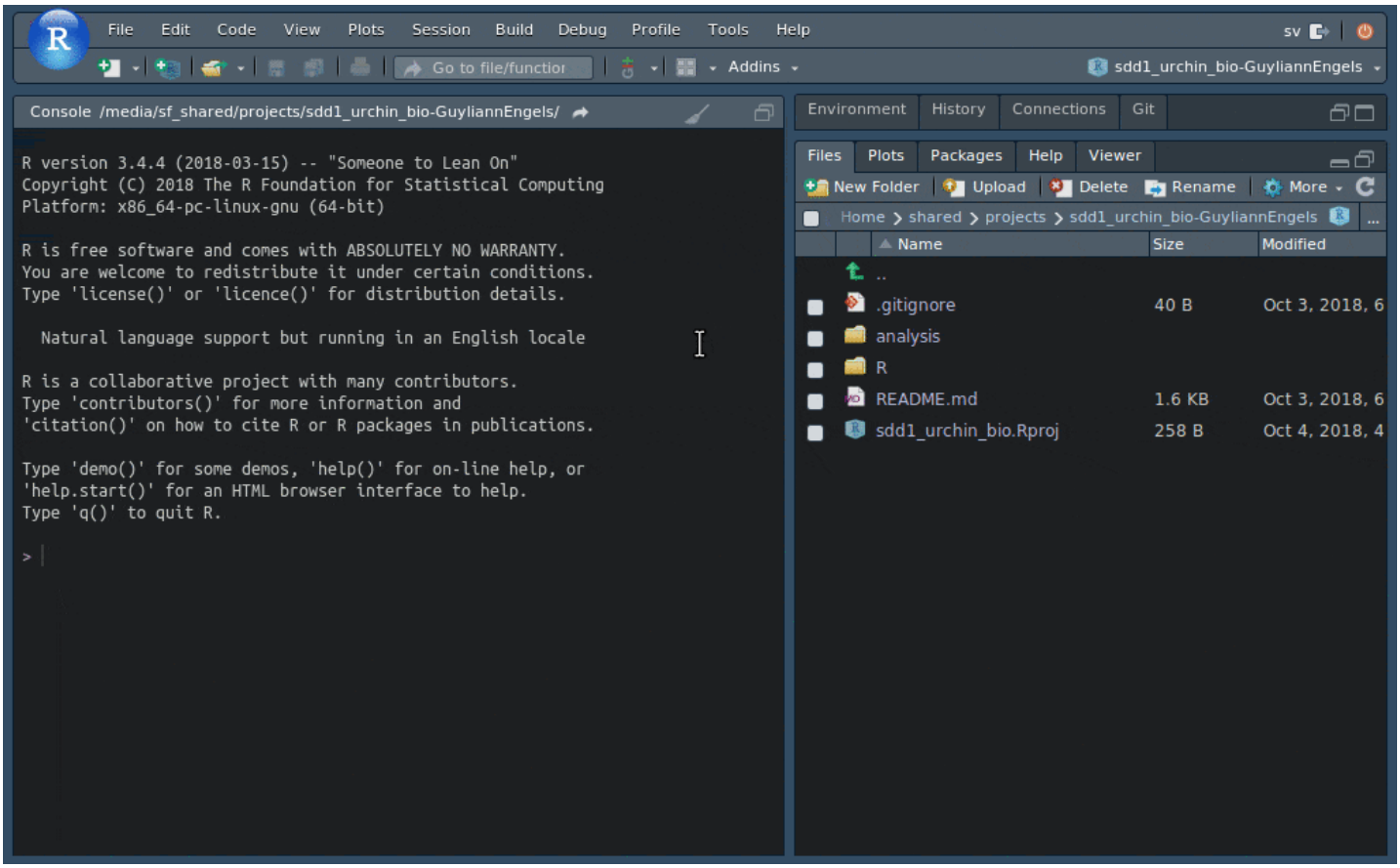
Un script R s'ouvre dans la fenêtre d'édition de RStudio. Les parties de texte précédées d'un dièse (`#`) sont des commentaires. Ils ne sont jamais exécutés, mais ils permettent de structurer et d'expliquer le contenu du document (ou bien d'empêcher temporairement l'exécution d'instructions). Voici un exemple de script R :



Pour bien documenter vos scripts, commencez-les toujours par quelques lignes de commentaires qui contiennent un titre, le nom du ou des auteurs, la date, un copyright éventuel, et une description courte de ce qu'il fait … L'utilisation de sections comme à la ligne 6 ci-dessus est vivement conseillée. Ces sections sont créées à l'aide de l'entrée de menu `Code -> Insert Section... || Code -> Insérer une section...` dans RStudio. Elles sont reprises dans un menu déroulant depuis le bas de la fenêtre d'édition pour une navigation rapide dans le script.

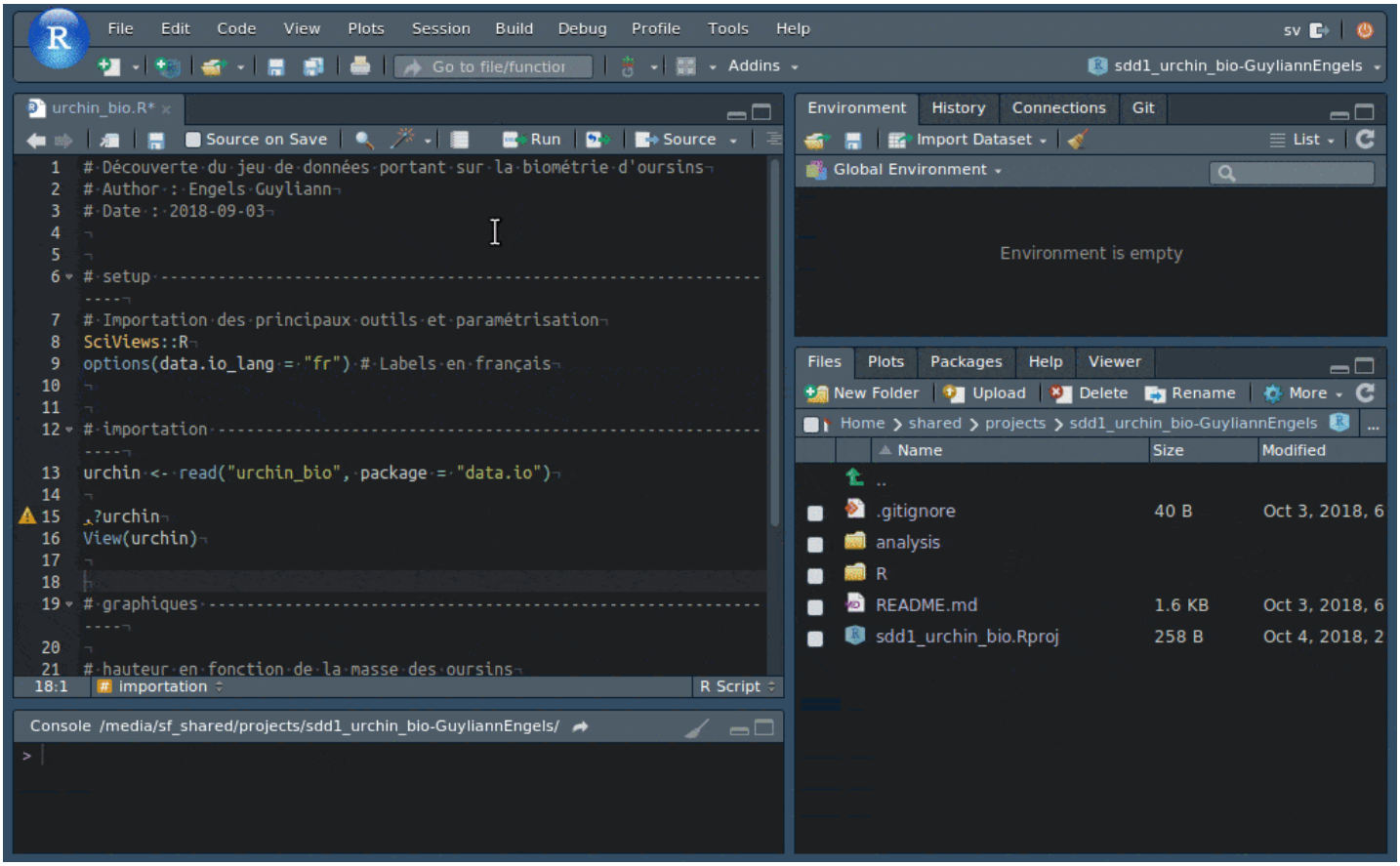
B.1.2.1 Création d'un script R

Vous avez à votre disposition plusieurs méthodes pour créer un nouveau script R dans RStudio, dont deux vous sont montrées dans l'animation ci-dessous.



B.1.2.2 Utilisation d'un script R

Un script R est un document au format natif de R. R va interpréter les instructions qui composent le script et qui ne sont pas précédées d'un dièse (cliquez sur `Run` || `Exécuter` dans la barre d'outils de la fenêtre d'édition, ou utilisez le raccourci clavier `Ctrl+Entrée` (ou `Cmd+Entrée` dans macOS) pour exécuter les instructions les unes après les autres.



Un script R doit être organisé de manière cohérente afin d'être exécutable de haut en bas. Dans l'exemple ci-dessus, on commence par :

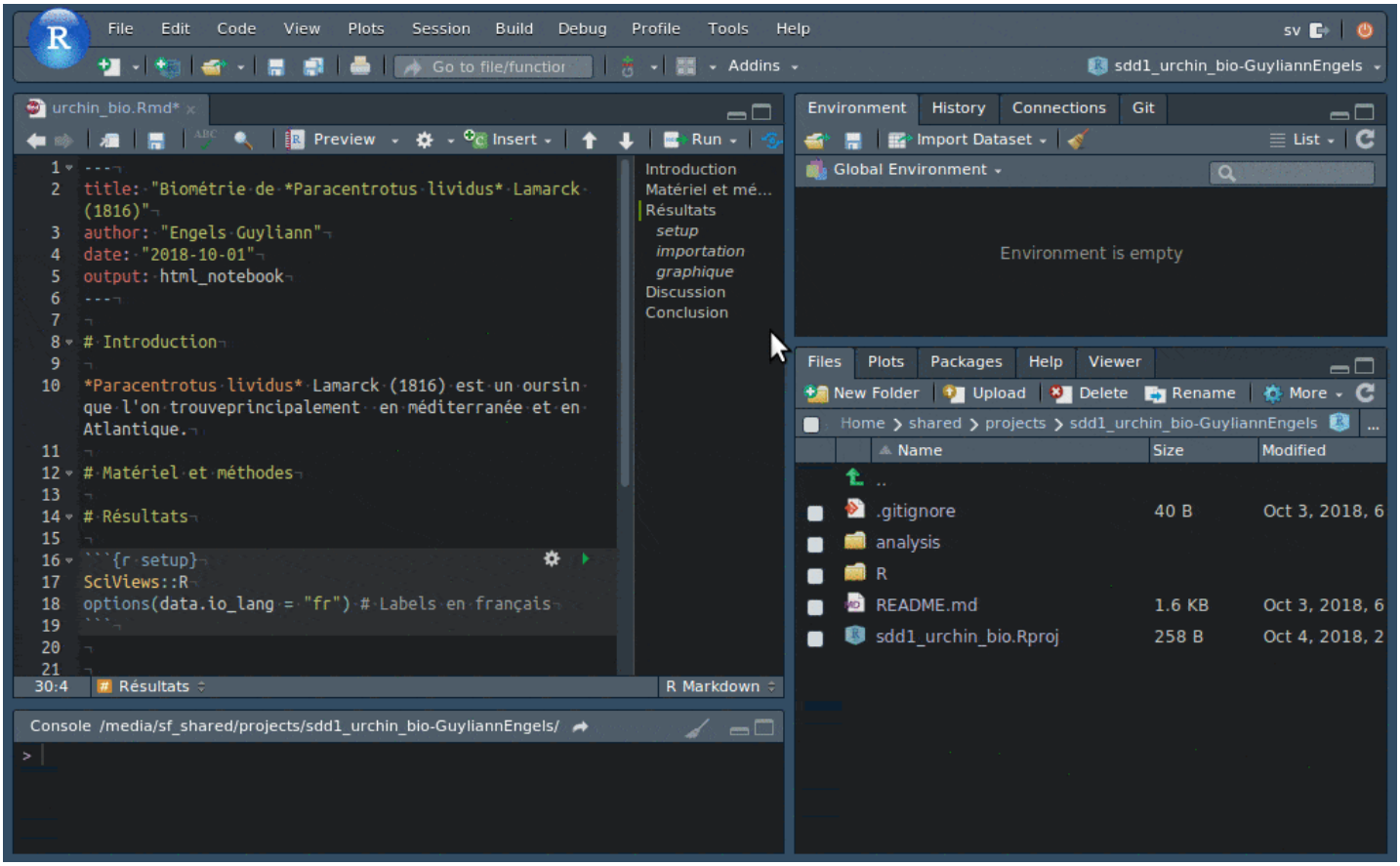
- **Étape 1 :** configuration de R en "mode SciViews" avec l'instruction `SciViews::R`.
- **Étape 2 :** instruction `urchin <- read("urchin_bio", package = "data.io")` pour lire le jeu de données `urchin_bio` provenant du package {data.io} (un "package" R est une sorte d'"addin" qui peut contenir de nouvelles fonctions, des jeux de données et de la documentation) et l'"assigner" ou "affecter" à `urchin` (le nom sous lequel nous avons choisi de nous référer à l'objet créé dans R). On retrouve à présent `urchin` dans l'environnement global (`Global Environment` dans l'onglet **Environment || Environnement** dans la fenêtre en haut à droite) de RStudio. Cet onglet liste, en effet, tous les objets en mémoire dans la session actuelle de R.
- **Étape 3 :** `.?urchin` et `View(urchin)` entrés à la console R (et validées en appuyant sur la touche `Entrée`) donnent des renseignements sur le jeu de données en renvoyant vers la page d'aide du jeu de données ou en ouvrant ce jeu de données dans une fenêtre de visualisation.

- **Étape 4 :** réaliser des graphiques avec une fonction de R nommée `chart()`. Une fonction est un petit programme qui réalise une tâche particulière sur base d'éléments fournis en entrée entre parenthèses, et que l'on nomme **arguments de la fonction**. À l'issue du traitement, la fonction renvoie un résultat que l'on peut envoyer vers une autre fonction ou "assigner" à un nom pour le réutiliser plus tard.

Notez que les instructions exécutées dans le script sont envoyées dans la fenêtre `Console` en bas à gauche et que le résultat de leur exécution est affiché directement en dessous dans la même fenêtre.

B.1.3 Quarto/R Markdown/R Notebook

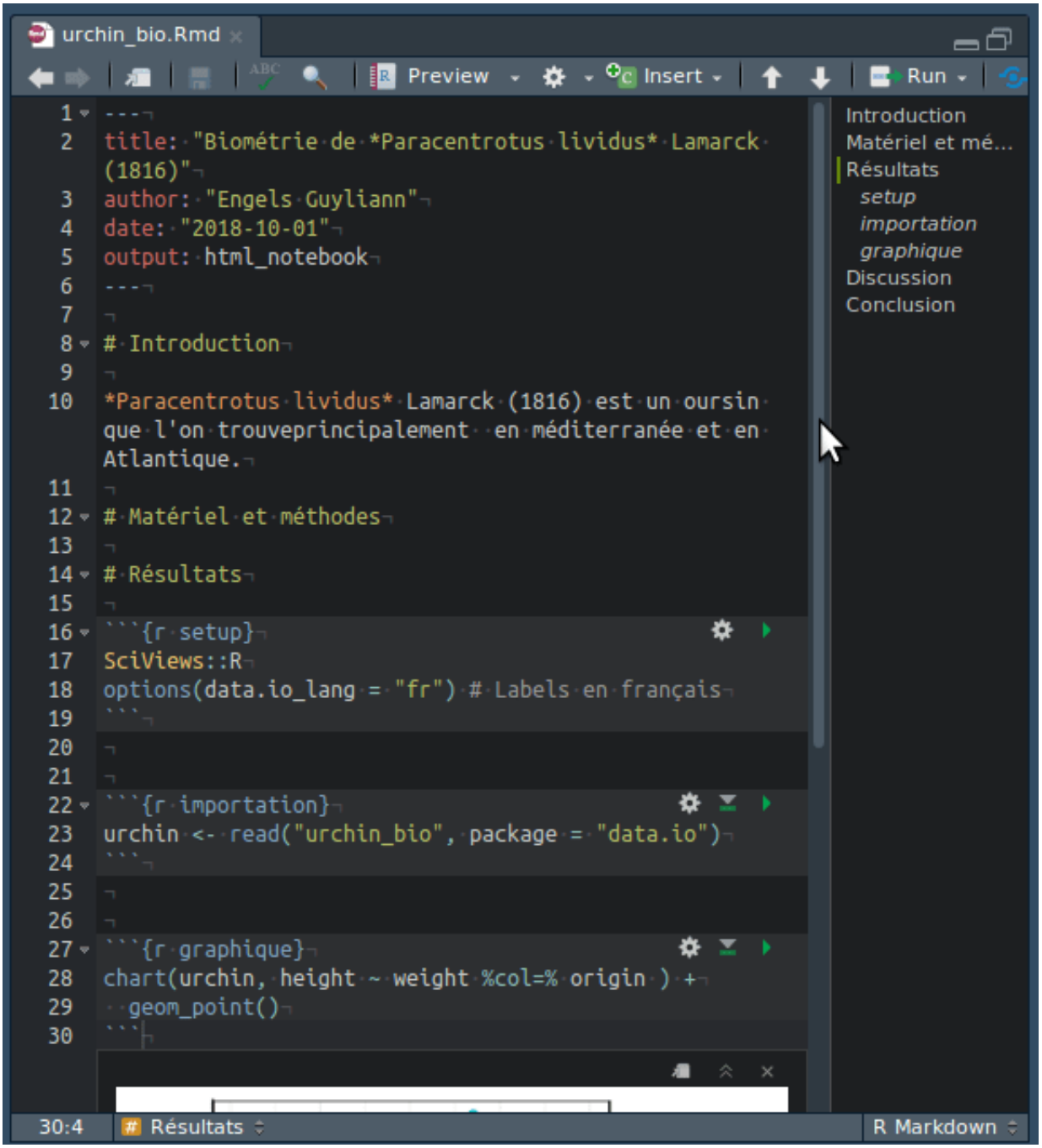
Un document Quarto est un fichier dont l'extension est `.qmd`. Un document R Markdown a, quant à lui, une extension `.Rmd`. Ces deux formats sont très similaires. Ils combinent à la fois des instructions R (pour les analyses) et le langage Markdown (un système de balisage de texte pour le formater, par exemple, pour définir un titre, une partie de texte en gras…), ce que l'on appelle le **"R Markdwon"**. Le R Markdown ne vous permet pas de visualiser directement le résultat final d'un rapport d'analyse[69]



Tout comme dans un script R, les instructions doivent également être exécutées lors de la réalisation du rapport. Une forme spéciale de document R Markdown est le **R Notebook**. Ce dernier est un peu un intermédiaire entre un script R et un document R Markdown. Il se présente de manière très similaire à ce dernier, mais vous pouvez devez exécuter le code qu'il contient ligne par ligne comme dans un script R pour obtenir le rendu, alors que le R Markdown ou le Quarto intégrera toujours l'exécution de code R à la compilation du document final. Si cela ne vous parait pas clair, pas d'inquiétude : vous comprendrez plus tard par la pratique.

Un document Quarto / R Markdown / R Notebook se structure de la manière suivante :

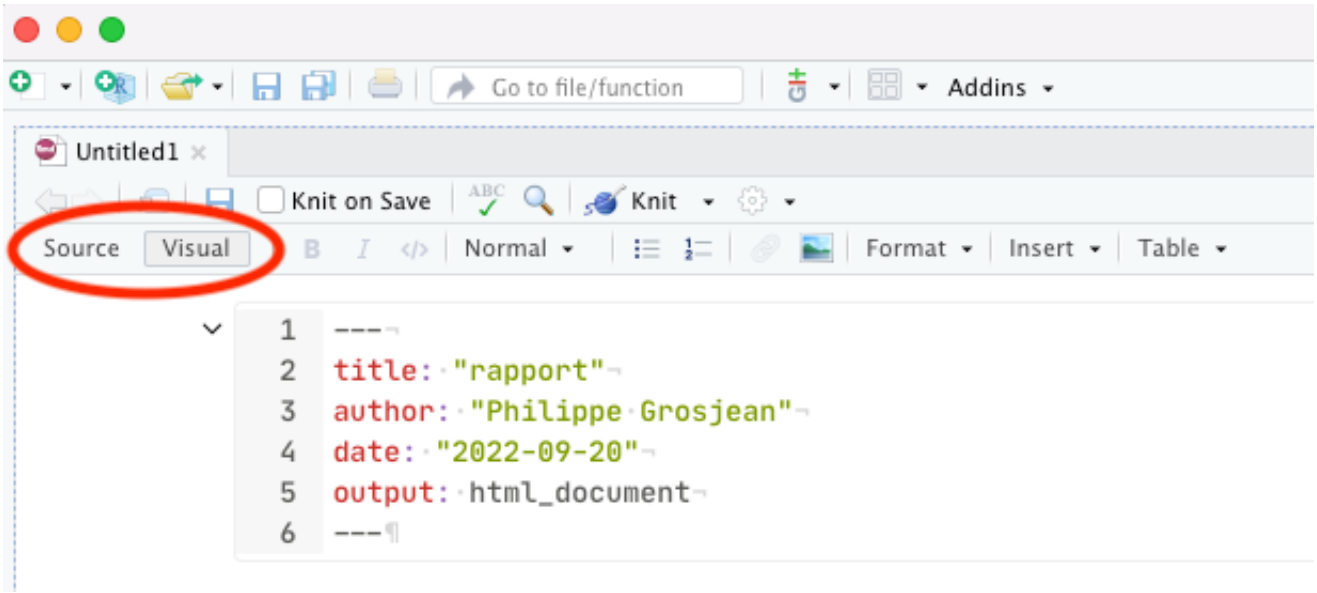
- Un **préambule**, encore appelé **entête YAML** ("YAML" est le nom du langage utilisé pour y encoder des informations)
- Des zones d'édition de texte (le langage employé est ici Markdown)
- Des zones de code appelées des **chunks** (aussi appelés "morceaux" dans RStudio en français)



Le **préambule** est nécessairement situé au tout début du document et est délimité à l'aide de trois tirets `---` sans rien d'autre sur une ligne au début et à la fin. Le préambule comporte un ensemble d'entrées de type `nom: valeur` qui configurent le document ou la façon dont il sera compilé en document final (rapport, présentation, publication…). Nous pouvons y indiquer le titre principal, le ou les auteurs, la date …

Le reste du document Quarto / R Markdown est subdivisé en zones successives et contrastées sur des fonds de couleurs différentes qui alternent zones d'édition de texte et chunks. Ces chunks seront interprétés pour réaliser un calcul, un graphique, un tableau, etc. Le résultat de ce traitement sera placé au même endroit dans le document final. La

présentation diffère profondément selon que le document est édité en mode “source” ou en mode “visuel”. On passe d’un mode à l’autre grâce aux deux boutons `Source` et `Visual` à la gauche de la barre d’édition du document.



En mode “visuel”, le document présente un rendu qui suggère la présentation finale, un peu comme Microsoft Word, mais gardez bien à l’esprit que le rendu final réel dépendra du style que vous appliquerez et du format de sortie choisi (dissociation entre contenu et rendu, contrairement à Word). Dans le mode visuel, vous effectuez la mise en forme de votre texte en utilisant les outils de la barre d’outils d’édition juste après le bouton `Visual` . Par exemple pour créer un titre de niveau deux, vous cliquerez sur le quatrième item qui indique le format actuel (probablement `Normal` , et vous sélectionnerez `Header 2 || Titre 2` . Le paragraphe où se trouve le curseur est alors transformé en titre de niveau deux. Pour transformer un paragraphe en liste, vous irez dans `Format -> Bullets & Numbering -> Bulletted List || Liste à puces` , et ainsi de suite. Les différents items parlent d’eux-mêmes pour la majorité. Explorez-les et expérimentez-les sur un document de test pour vous les approprier.

Le mode “source” est intéressant, car il expose la structure et le formatage du document en “brut de décoffrage”. Cela vous permet de voir comment le formatage Markdown se matérialise sous forme de **“balises”** dans le texte. Si vous basculez le document dans ce mode, vous découvrirez, par exemple, qu’un titre de niveau deux commence par deux dièses suivis d’un espace (`## titre niveau 2`) et qu’une liste à points commence par un tiret suivi de trois espaces (`-  item de liste à points`). Ce mode permet aussi un

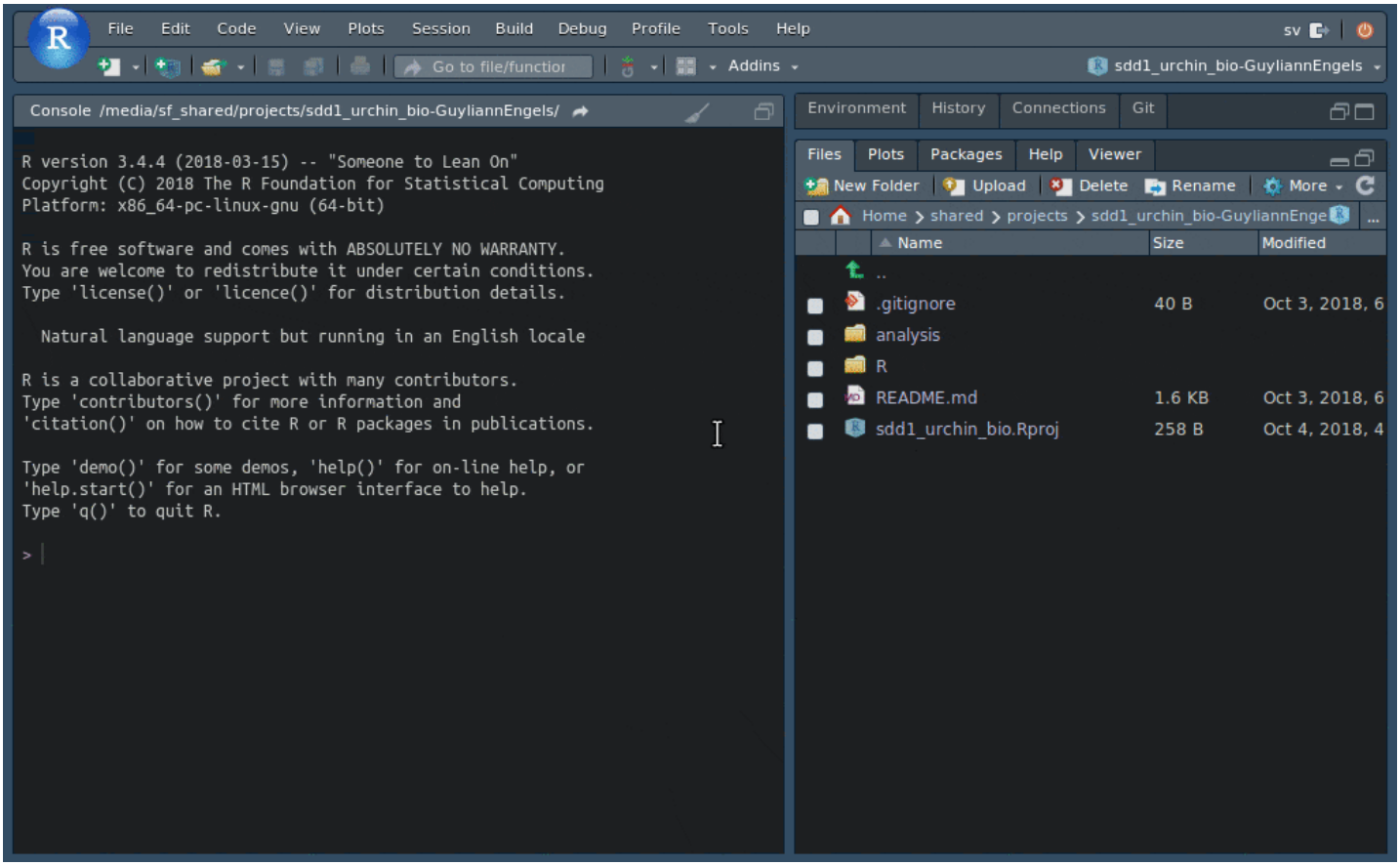
contrôle plus fin sur le contenu de votre document, mais comme il nécessite de connaitre la signification des balises Markdown, vous préférerez certainement le mode visuel au début. En cours d'édition, vous pouvez basculer d'un mode à l'autre librement quand vous le souhaitez.

Toujours en mode source, les chunks sont balisés en entrée par trois apostrophes inverses suivies d'accolades contenant des instructions relatives au programme à utiliser sur une seule ligne, par exemple, ```` ```{r} ```` pour des chunks faisant appel au logiciel **R**, et sont terminés par trois apostrophes inverses (```` ``` ````), également sur une ligne sans rien d'autre. Entre les deux, les instructions R et les commentaires avec un dièse devant se présentent exactement comme dans un script R.

Structurez le contenu narratif de votre document Markdown à l'aide de titres de niveaux appropriés. Vous pouvez utiliser jusqu'à six niveaux de titres préfixés à l'aide d'un à six dièses et d'une espace. Utilisez donc un titre de niveau un pour le titre **Introduction** et des titres de niveau deux pour les sous-parties de votre introduction, et ainsi de suite.

B.1.3.1 Création d'un Quarto/R Markdown/R Notebook

Vous avez à votre disposition deux méthodes pour créer un nouveau document Quarto/R Markdown/R Notebook dans RStudio et l'ouvrir pour édition. Voyez l'animation ci-dessous.

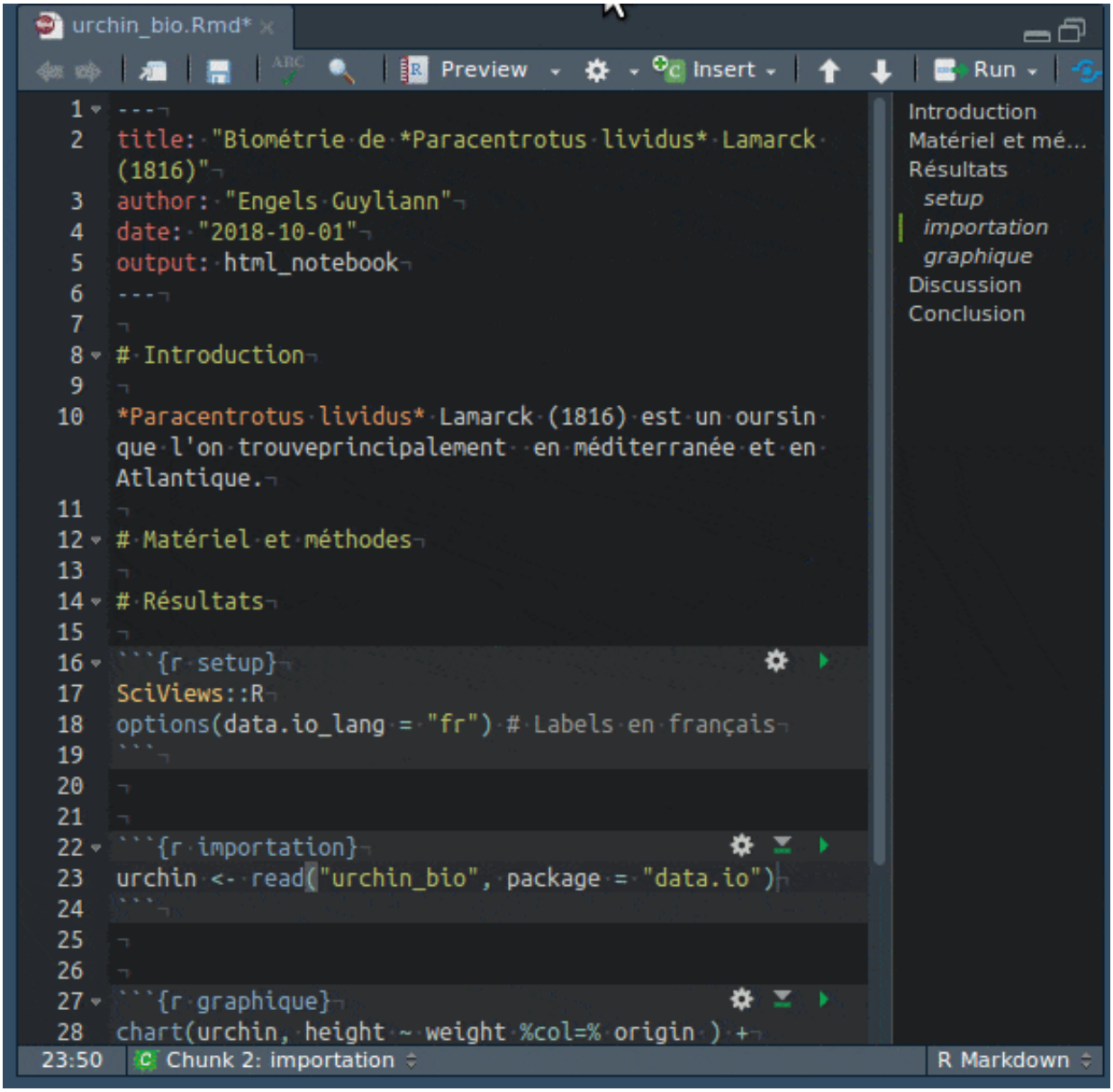


B.1.3.2 Utilisation d'un Quarto/R Markdown/R Notebook

Afin de visualiser les résultats des chunks dans votre rapport final, vous devez veiller à exécuter chaque chunks dans l'ordre dans un R Notebook. Ceci n'est pas nécessaire dans un Quarto ou un R Markdown, mais dans ce cas, tous les chunks sont systématiquement réexécutés les uns après les autres à chaque génération de rapport, ce qui peut être pénible si les calculs sont longs.

Pour exécuter un chunk, vous pouvez :

- cliquer sur le bouton **Play**, sous forme d'une flèche verte pointant vers la droite, située en haut à droite du chunk
- cliquer sur `Run || Exécuter` et sélectionner `Run Current Chunk || Exécuter le Chunk actuel` dans le menu déroulant qui apparaît
- Employer le raccourci clavier `Ctrl+Shift+Entrée` (ou `Cmd+Shift+Entrée` sous macOS)

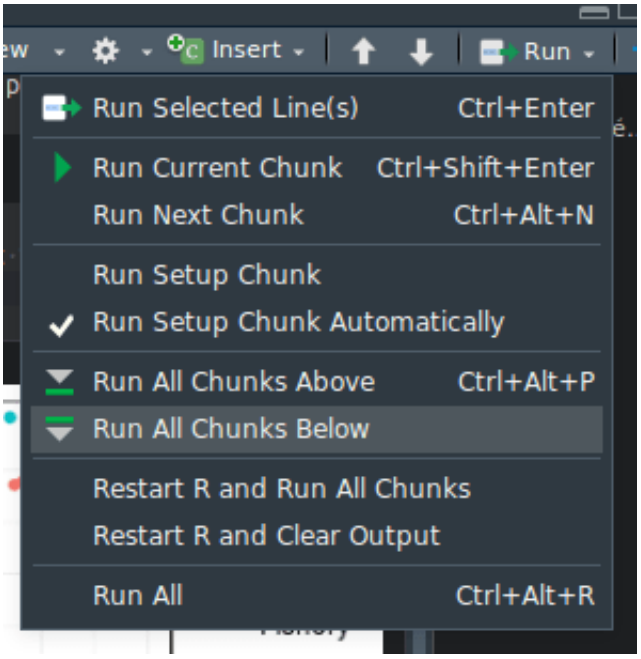


L'item `Run || Exécuter` ouvre un menu qui expose plusieurs autres actions intéressantes, entre autres :

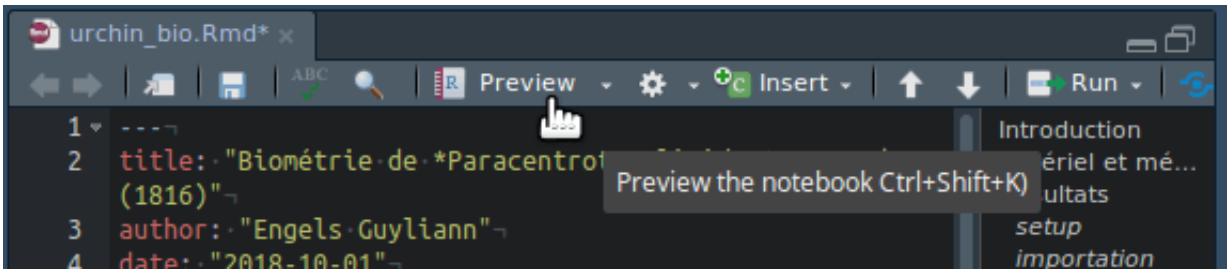
- Exécuter la/les ligne(s) d'instruction sélectionnée(s) : `Run Selected Line(s) || Exécuter la ou les Lignes sélectionnées` , raccourci `Ctrl+Entrée` ou `Cmd+Entrée`
- Exécuter le chunk dans lequel le curseur se trouve en entier : `Run Current Chunk || Exécuter le Chunk actuel` que nous avons déjà mentionné
- Exécuter tous les chunk précédents : `Run All Chunk Above | Exécuter tous les chunks au-dessus`
- Redémarrer R et exécuter tous les chunks dans la foulée : `Restart R and Run All Chunks || Redémarrer R et exécuter tous les chunks` . **Cette action est**

particulièrement intéressante pour s'assurer que le document est réellement reproductible !

- …



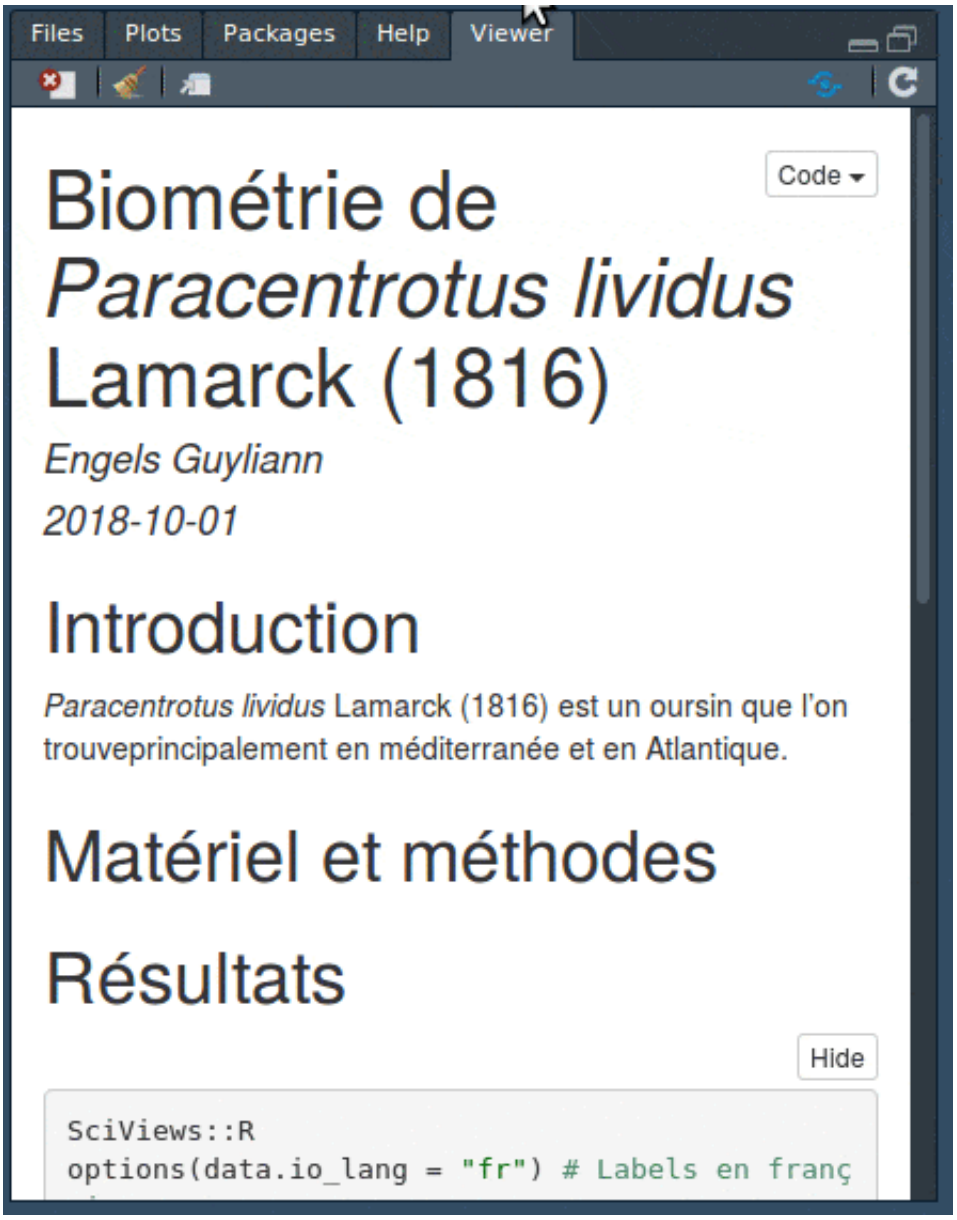
Après la phase d'édition du texte et des instructions dans les chunks, vous pouvez visualiser votre document final en cliquant sur le bouton `Preview || Prévisualiser` (R Notebook), `Knit || Tricoter` (R Markdown) ou `Render || Rendu` (Quarto). Dans le cas du R Notebook, vous devrez obligatoirement avoir exécuté tous les chunks auparavant. Pour le Quarto ou le R Markdown, ce n'est pas nécessaire puisque leur exécution fait partie intrinsèque de la compilation du document final.



Le document final est généré avec un rendu simple et professionnel. Par défaut, ce document final présente le texte que vous avez écrit avec les résultats que vous avez choisi de générer via R, mais également les instructions que vous avez employées pour obtenir ces résultats. Ceci permet de mieux comprendre, directement dans le rapport, comment tout cela a été calculé. Il est possible de cacher le code (dans un document généré depuis un R Notebook), ou d'indiquer une directive de compilation dans les chunks

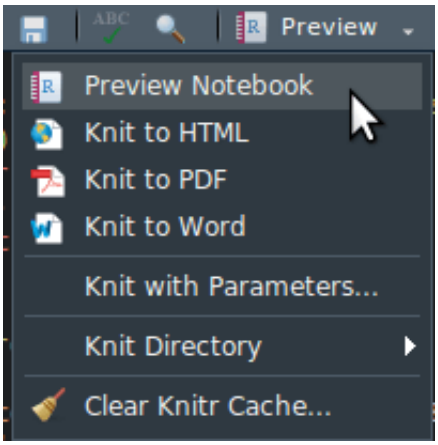
pour éviter que le code ne s'imprime dans le document final. Voyez les options en cliquant sur le petit engrenage à côté de la flèche verte en haut à droite du chunk. Consultez l'aide-mémoire de R Markdown accessible à partir du menu RStudio `Help -> Cheat Sheets -> Markdown Reference Guide || Aide -> Cheat Sheets -> Guide de référence de R Markdown` , voir **chunk options** p. 2-3 pour plus d'informations sur les nombreuses options disponibles.

Par exemple, en ajoutant la directive `echo=FALSE` dans la balise d'entrée d'un chunk (```` ```{r, echo=FALSE} ````), on empêche d'imprimer le code de ce chunk dans le document final.



Notez que sur la droite du bouton `Preview || Prévisualiser` (R Notebook) `Knit | Tricoter` (R Markdown) ou `Render || Rendu` (Quarto), vous avez un autre bouton représenté par un petit engrenage blanc. Il donne accès à un menu déroulant qui vous permet de modifier la façon de générer vos rapports. L'entrée tout en bas `Output Options... || Options de sortie...` permet de paramétrer la présentation du document final.

Si vous cliquez sur la petite flèche noire pointant vers le bas juste après `Preview || Prévisualiser` ou `Knit || Tricoter` , vous avez un autre menu déroulant qui donne accès aux différents formats possibles : HTML, PDF, Word, etc. Essayez les différentes options pour visualiser comment votre document final se présente dans les différents formats.



65. En réalité, il existe deux versions de RStudio : **RStudio IDE** qui fonctionne en mode local, comme la plupart des logiciels que vous utilisez sur votre ordinateur, et **RStudio Server** qui offre quasiment la même interface à l'utilisateur, mais qui fonctionne sur une machine distante (dans SaturnCloud, par exemple). Pour des raisons de simplicité, lorsque nous indiquons "RStudio" dans le texte, cela se réfère indifféremment à ces deux moutures, sauf indication particulière et pour les fonctionnalités spécifiques à l'une ou à l'autre.↩

66. N'éditer **jamais** à la main un fichier `.Rproj` . Laisser RStudio s'en occuper !↩

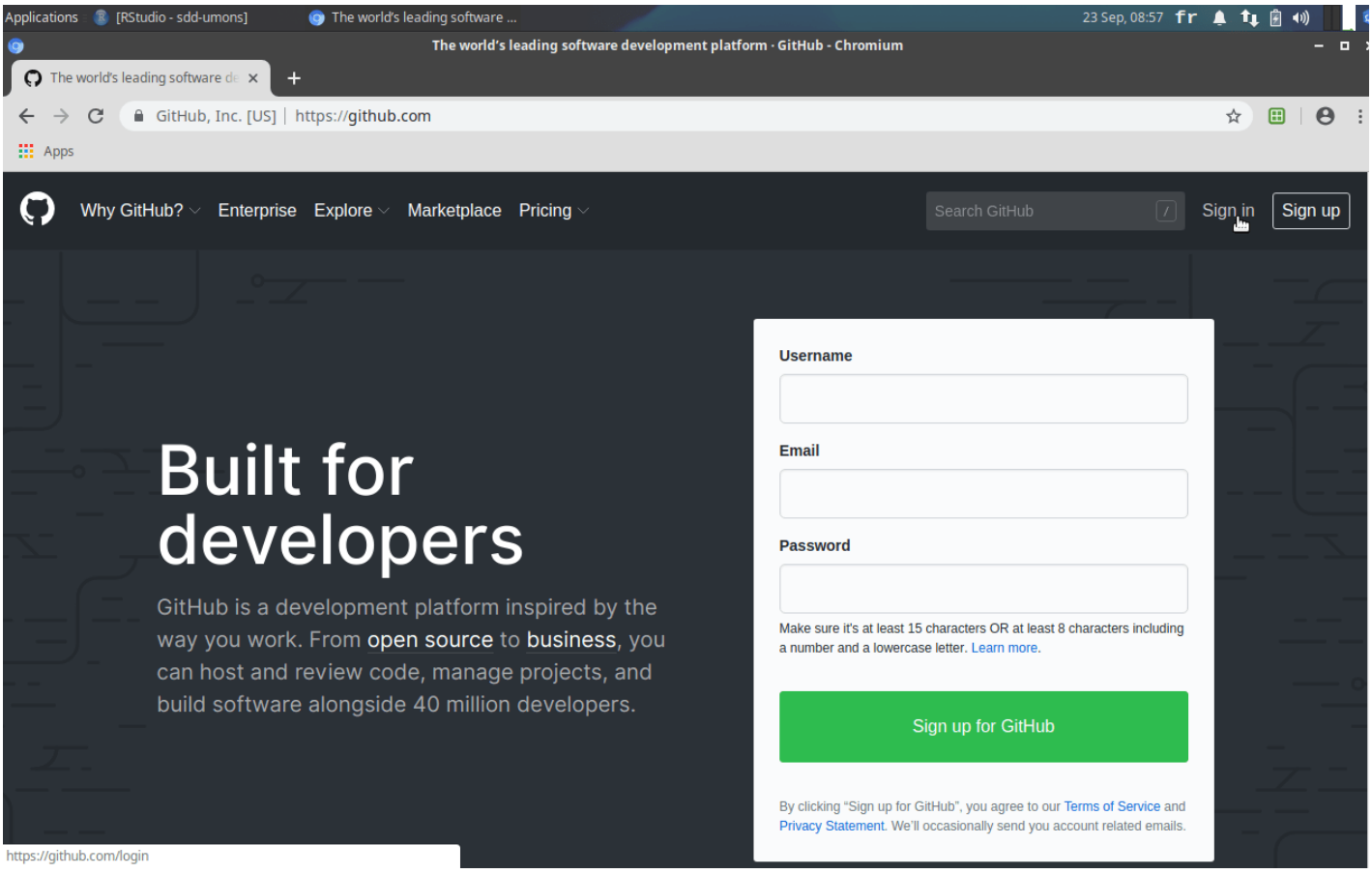
67. Vous noterez que RStudio indique `HOME || Accueil` dans l'onglet **Files || Fichiers** pour représenter le dossier de l'utilisateur qui est en réalité `/home/sv` dans VirtualBox ou / `home/jovyan` dans SaturnCloud.↩

68. Dans R sous Windows, si vous utilisez les backslashs, vous devez les doubler (`~\\Documents\\...`). Ce n'est ni très esthétique, ni compatible avec les deux autres systèmes. Heureusement, nous avons déjà noté que R comprend aussi le slash comme séparateur sous Windows, de sorte que la même syntaxe peut être utilisée partout ! **Nous vous conseillons donc d'utiliser également les slashs sous Windows dans R ou RStudio.**↩

69. Les systèmes d'édition professionnels dissocient en effet le fond de la forme : vous rédiger d'abord le contenu, et ensuite, vous indiquer le style à lui appliquer.↩

B.2 GitHub

Un réseau social a été conçu autour de Git pour sauvegarder vos projets sur le “cloud”, les partager et collaborer avec d’autres personnes. Ce système se nomme GitHub (tout comme Facebook ou LinkedIn). GitHub rassemble donc “Git”, la gestion de version et “Hub” relatif au réseau. D’autres réseaux équivalents existent comme Gitlab ou Bitbucket.



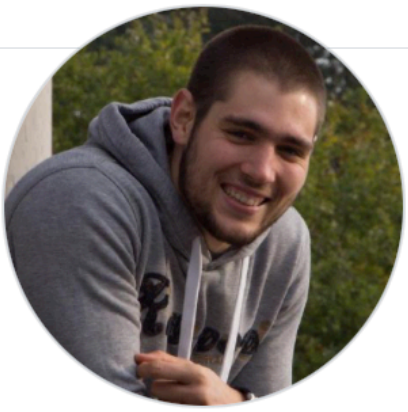
B.2.1 Votre activité et profil

Pour vous montrer différentes sections sur GitHub, nous utiliserons le compte de GuyliannEngels. Une fois enregistré dans GitHub, nous nous trouvons dans une page qui nous montre notre activité. Au milieu de la page, nous pouvons observer les dépôts

épinglés (section “Pinned”) que vous considérez comme les plus importants.

Search or jump to... / Pull requests Issues Marketplace Explore

Overview Repositories 5 Projects Packages Stars 20



Guyliann Engels
GuyliannEngels

Unfollow

R developper

10 followers · **4** following

University of Mons
Mons, Belgium
guyliann.engels@umons.ac.be
@EngelsGuyliann

Achievements

Beta Send feedback

Organizations

Block or Report

GuyliannEngels / README.md

GuyliannEngels

I'm Guyliann from Belgium. I'm research and teaching assistant at Numerical Ecology of Aquatic systems laboratory, University of Mons (UMONS).

My languages of choice are R.

- I'm currently working on
  - the Data Science courses for biologists
  - the diversity of marine plankton : identification guide to mesoplankton

Recent GitHub Activity

1. Commented on #3 in BioDataScience-Course/BioDataScience
2. Commented on #3 in BioDataScience-Course/BioDataScience

GitHub Stats ⭐

Guyliann Engels' GitHub Stats

Total Stars Earned:	0
Total Commits (2022):	168
Total PRs:	8
Total Issues:	49
Contributed to:	20

A+

Pinned

BioDataScience-Course/teaching_data_science_in_biology
Public
R ☆ 2

EcoNum/zooimage_mesozooplankton_guide3
Public
Guide d'identification du mésozooplancton en Mer Ligurienne
TeX

covid19_belgium Public
Shinydashboard on COVID-19
R

BioDataScience-Course/BioDataScience
Public
A Series of Learnr Documents to study Biological Data Science
R ☆ 6 10

Plus bas dans la page, il y a un graphique qui représente vos contributions au cours du temps. Il indique de manière globale votre travail, c'est-à-dire vos contributions dans les différents projets.

Dans notre exemple, nous pouvons observer 983 contributions pendant l'année écoulée.

B.2.2 Vos projets

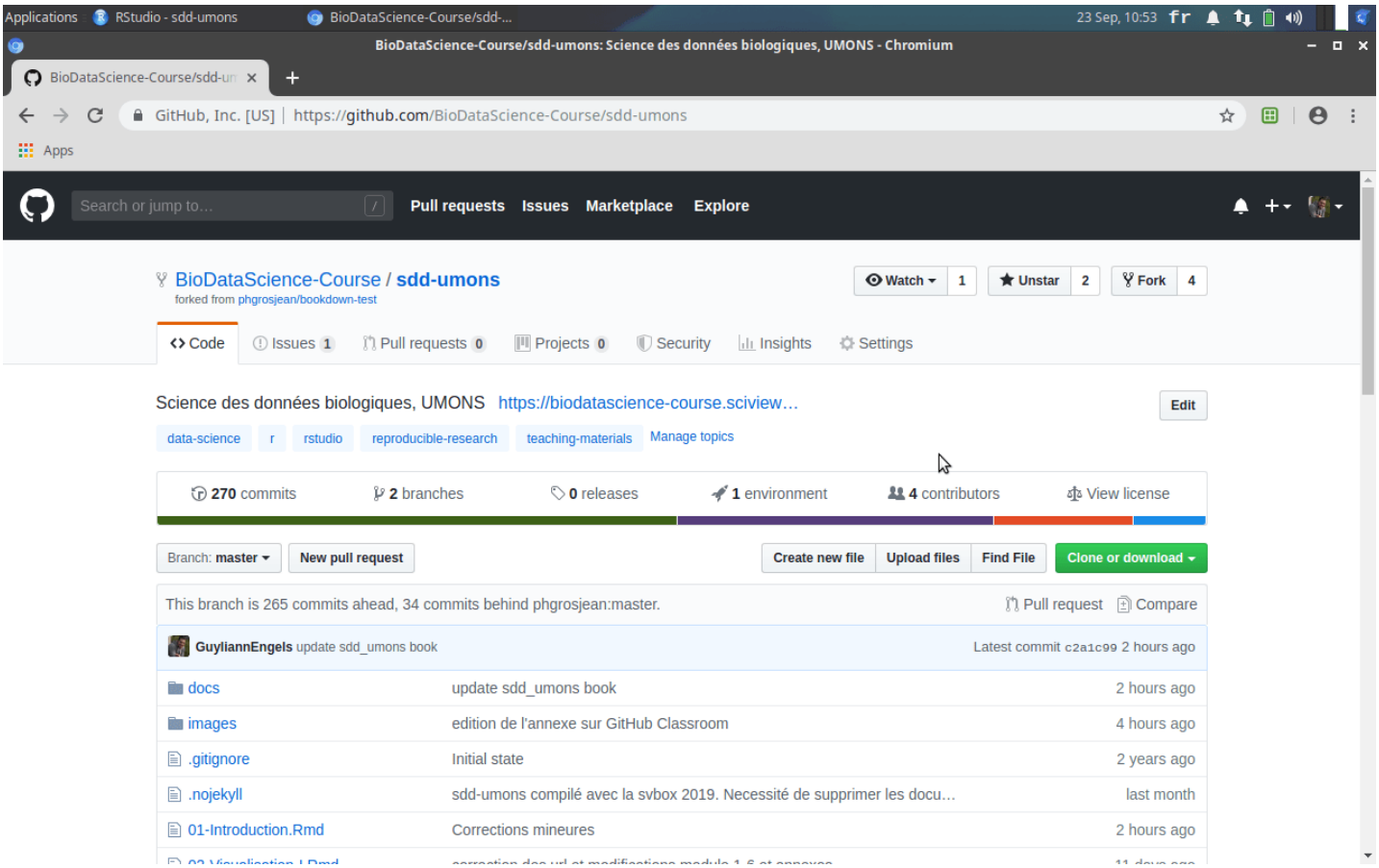
GitHub vous sert à héberger vos projets dans des **dépôts** (qui se nomment des *repositories* en anglais). Notre exemple se base sur le dépôt sdd-umons-2024, que vous pouvez librement consulter et qui contient les sources d'une version antérieure du présent ouvrage en ligne. Il s'agit en effet d'un dépôt public. Vous avez la possibilité d'avoir des projets publics ou privés (vos exercices dans le cadre du cours seront, eux, dans des dépôts privés visibles uniquement de vous et de vos enseignants).

Les projets publics sont visibles par tous. La collaboration est la pierre angulaire de GitHub. Donc, tout le monde peut y apporter des modifications dans des copies de vos dépôts et puis vous les proposer d'une manière qui vous permet d'examiner les améliorations suggérées, de les discuter, et enfin de les incorporer ou de les rejeter. Cela s'appelle un **Pull request**, mais nous y reviendrons plus tard. Pour des projets plus sensibles, vous avez la possibilité de les héberger en mode privé et de ne les partager qu'avec un nombre restreint et prédéterminé d'utilisateurs.

La référence à un dépôt sur GitHub est composée de deux éléments. Le premier est le nom de la personne ou de l'organisation auquel le dépôt appartient. Le second élément est le nom du dépôt lui-même. Dans notre exemple, nous aurons donc `BioDataScience-`

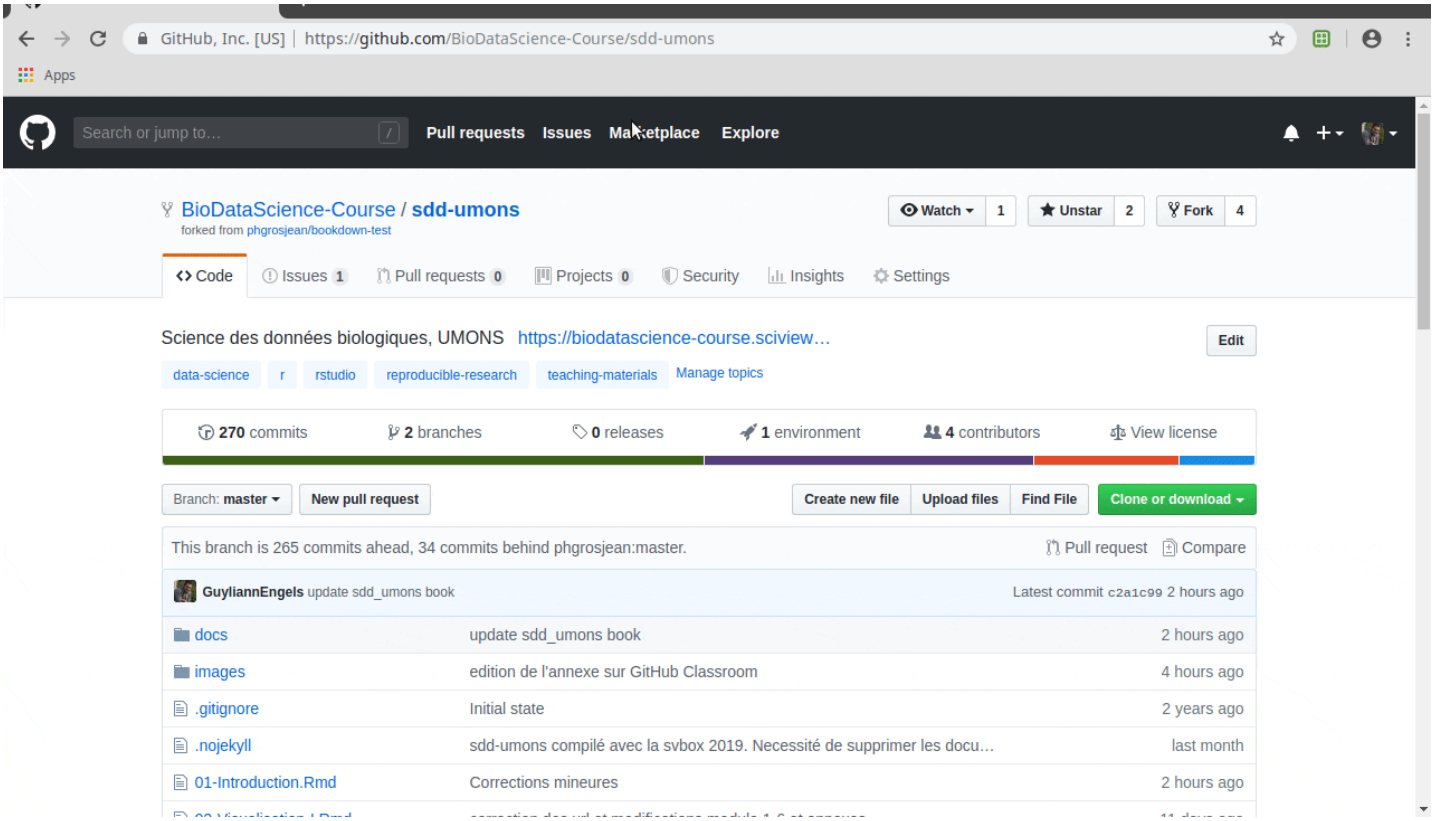
`Course/sdd-umons-2024` . Tous les projets relatifs au cours de sciences des données biologiques à l'UMONS sont hébergés dans l'organisation BioDataScience-Course (Il en sera de même pour tous les travaux que vous réaliserez dans le cadre de vos cours).

Vous pouvez observer une première barre d'outils comprenant les sections `Code` , `Issues` , `Pull requests` , `Actions` , `Projects` , `Wiki` , `Security` , `Insights` et `Settings` (par la suite, nous ne détaillerons que les sections importantes dans le cadre de votre cours de science des données).

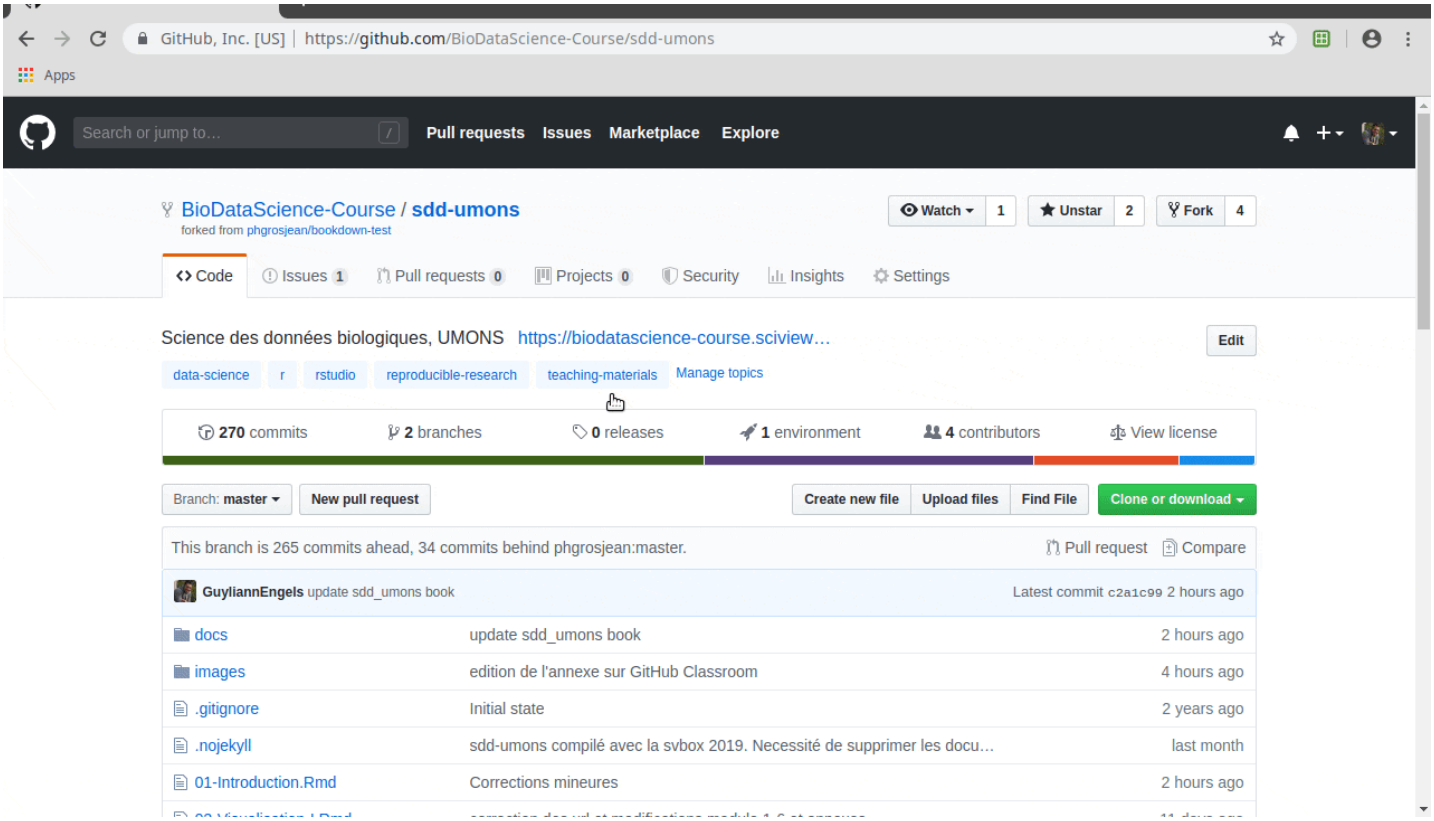


B.2.2.1 Code

Dans cette section, vous pouvez visualiser le contenu des différents fichiers qui se trouvent dans le dépôt. Vous naviguez dans les sous-dossiers simplement en cliquant dessus. La visualisation des fichiers se fait, éventuellement, selon plusieurs présentations différentes, dont le mode "Raw" pour voir le texte brut. C'est l'équivalent du mode "Source" dans l'éditeur de documents R Markdown de RStudio. Le bouton vert `Code` dans cette section est important. C'est grâce à lui que vous pourrez récupérer le contenu du dépôt. On parle de **cloner** un dépôt.

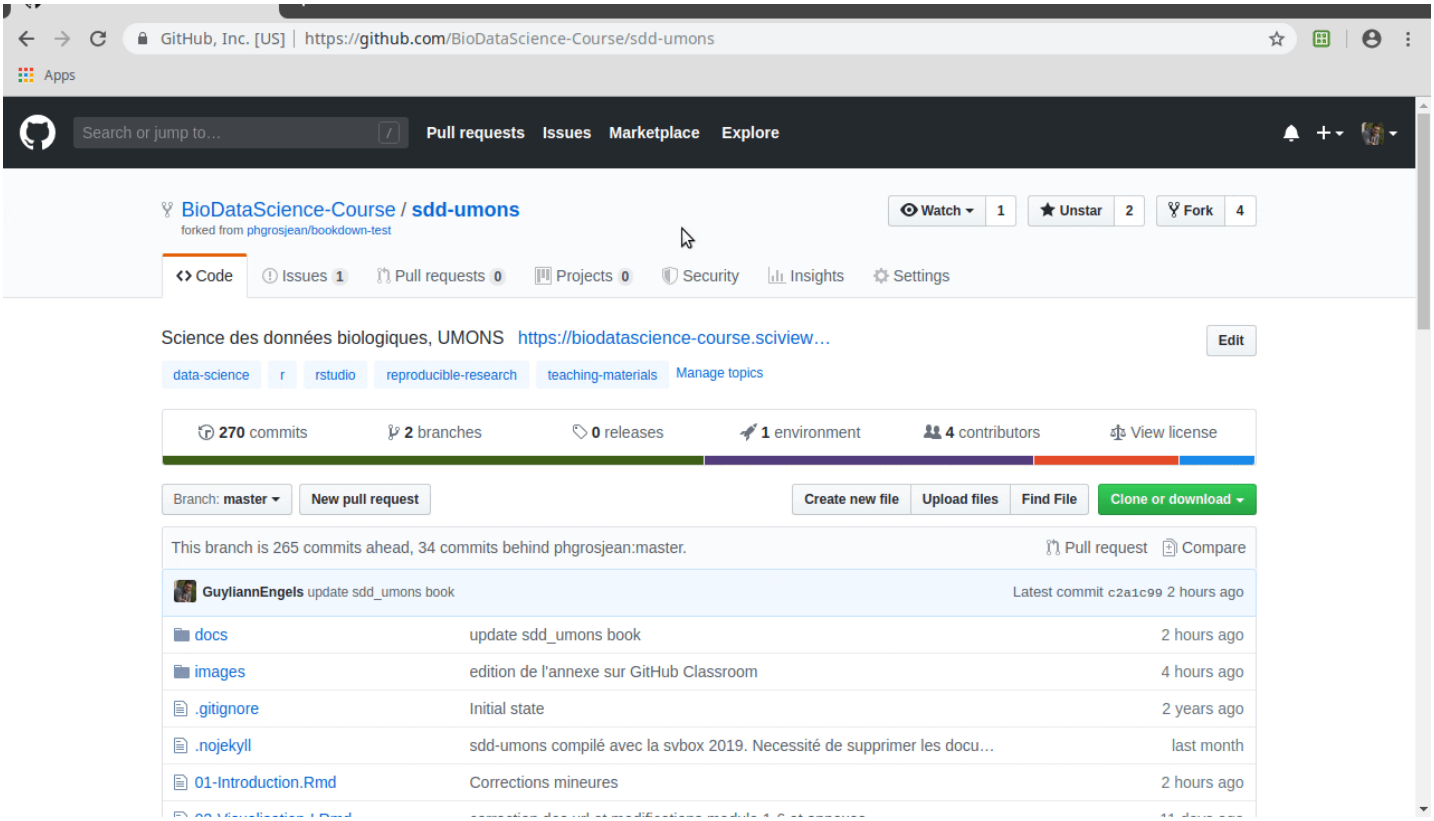


Comme il s'agit d'un système de gestion de versions, toutes les versions successives du dépôts sont stockées et accessibles, mais au départ, vous ne voyer que la dernière version. Chaque fois que vous enregistrez une version, vous réaliser ce que l'on appelle dans le jargon Git un **commit**. Le nombre de commits et des informations à son propos sont présentées dans la barre grisée sous les quelques boutons.



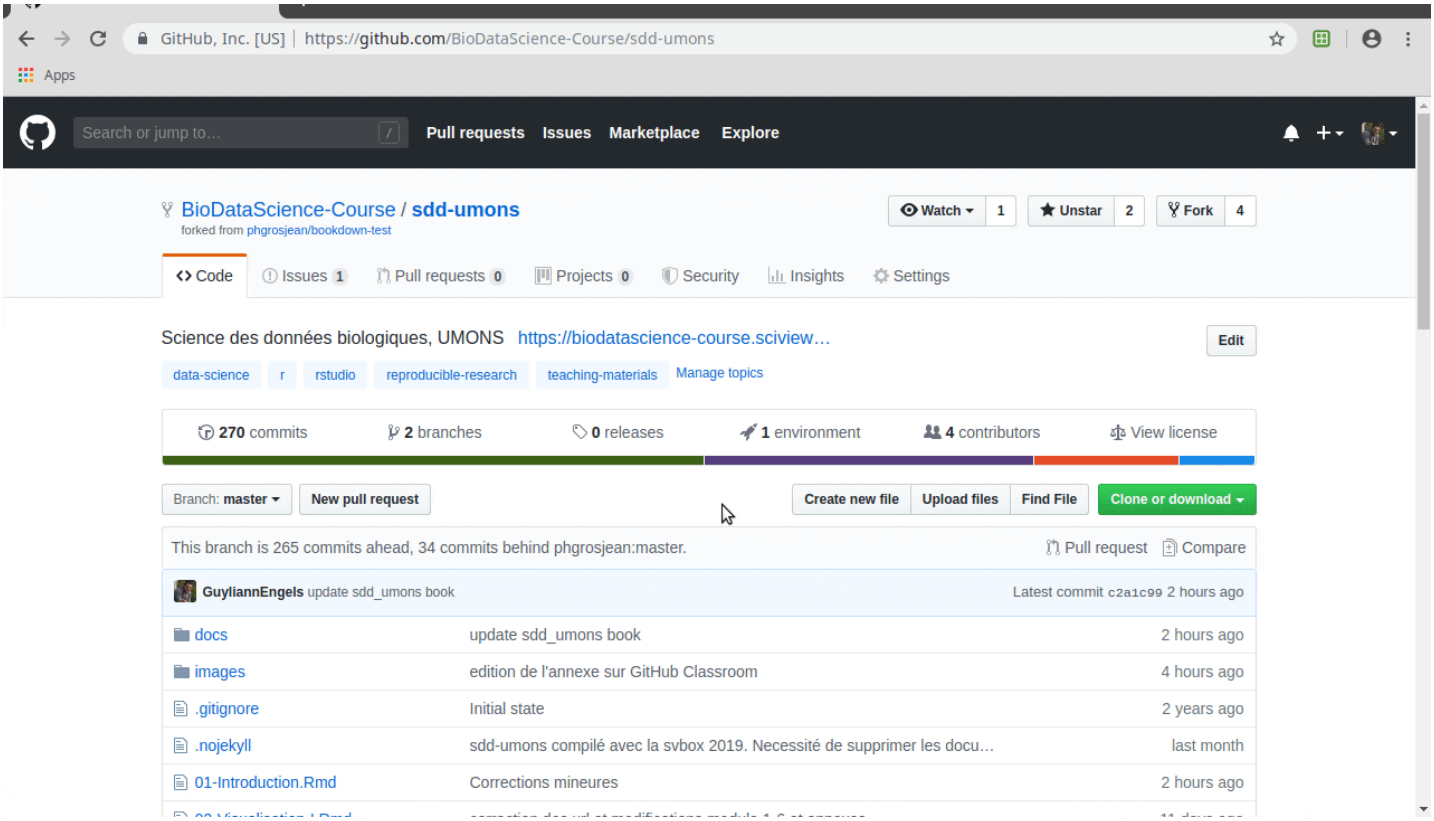
B.2.2.2 Issues

Cette section est prévue pour discuter des problèmes ou des idées à développer. C'est une sorte de petit forum de discussion. Vous utiliserez ces "issues" pour poser vos questions à vos enseignants et pour interagir entre vous dans les projets de groupe.



B.2.2.3 Insights

La section `Insights` nous renseigne sur l'activité du projet. On peut y voir, par exemple, les contributeurs du projet, le nombre de commits réalisés par chacun, etc.



Concernant vos projets dans le cadre du cours, ces informations sont employées pour évaluer la contribution de chacun dans les travaux de groupes et ajuster la note en fonction.

B.2.3 Permettre à RStudio de communiquer avec GitHub

Dans le cours de Science des Données vous utiliserez RStudio pour éditer vos documents et travailler avec R. Mais d'autre part, vous partagerez vos œuvres via GitHub. C'est d'ailleurs un schéma classique que nous vous conseillons d'adopter aussi en dehors du cours ! Les professionnels travaillent tous comme cela. Nous devons donc trouver un moyen de permettre à RStudio de communiquer avec GitHub. Les fonctionnalités existent, dans l'onglet **Git** (qui n'apparaît que si vous êtes dans un projet géré par Git). Encore faut-il indiquer à GitHub que vous autoriser votre RStudio à accéder à votre compte et à vos dépôts. Pour cela, deux solution très différentes existent : le code d'accès personnel **PAT**, un code généré par GitHub et que vous incluez dans votre SciViews Box, ou la **clé SSH**, une clé privée/publique générée au contraire dans votre machine et dont vous renseignez

la partie publique à GitHub pour qu'il puisse l'utiliser. Nous allons voir ces deux méthodes successivement, sachant que la version VirtualBox de la SciViews Box utilise le PAT, alors que c'est plus facile d'utiliser la clé SSH dans SaturnCloud.

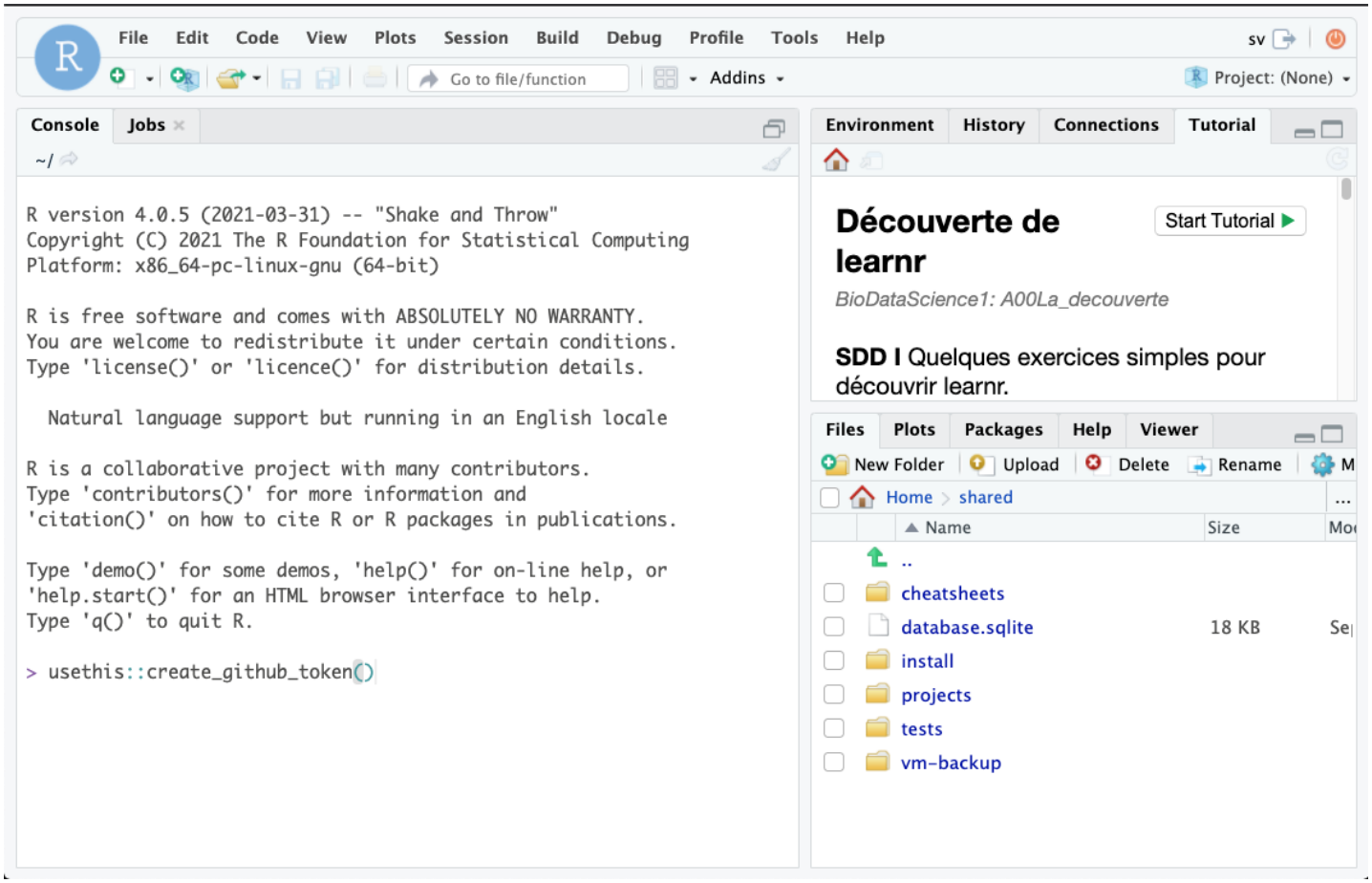
B.2.3.1 Personal Access Token

Note : dans SaturnCloud, une autre technique basée sur la clé SSH est utilisée. Dans ce cas, voyez la section suivante.

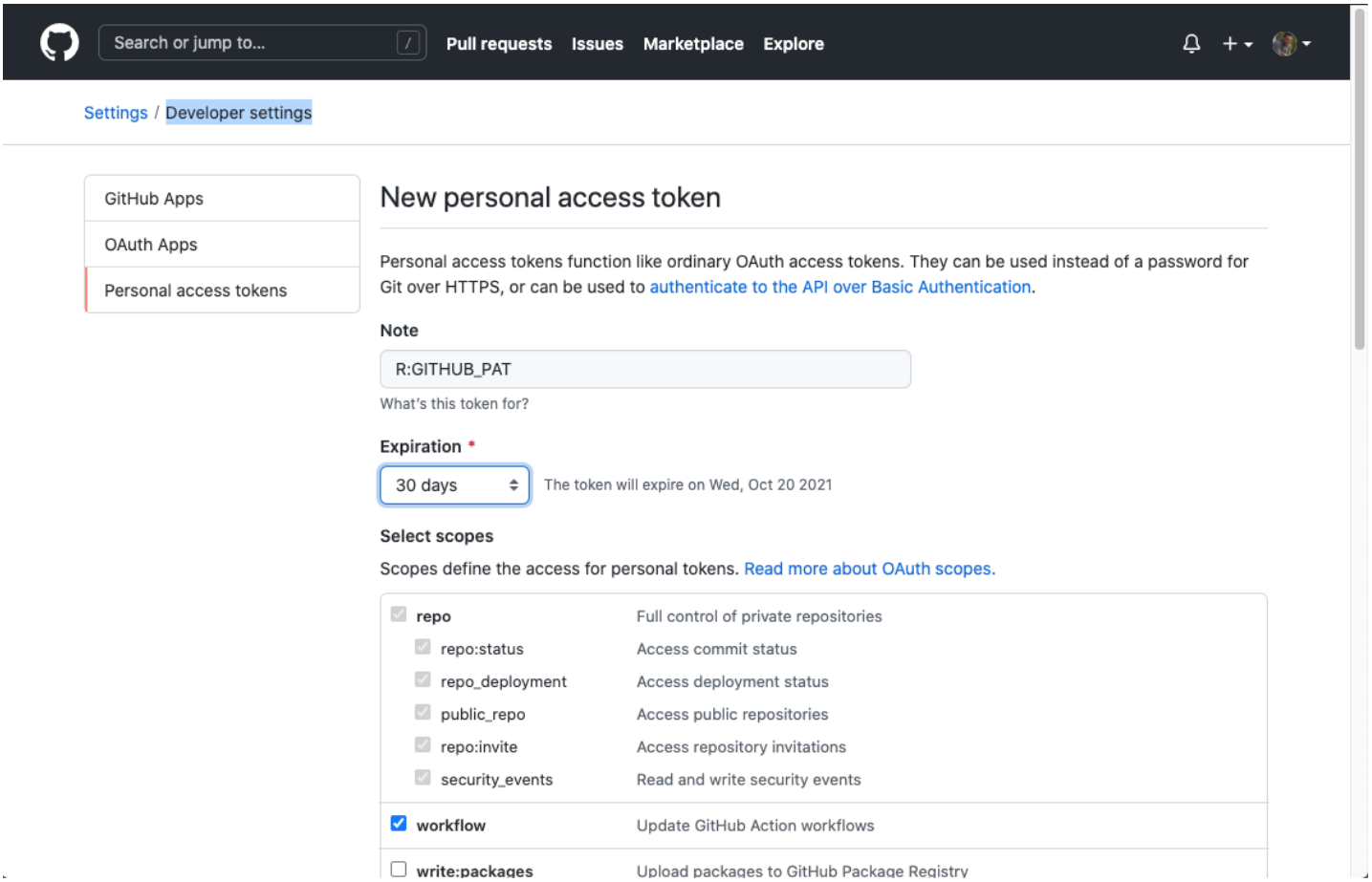
L'authentification dans GitHub peut se faire via un *Personal Access Token* (PAT). Il s'agit d'une sorte de mot de passe long et compliqué (donc bien sécurisé) que GitHub génère pour vous et que vous devez renseigner à tout logiciel souhaitant accéder à votre compte. Ce système d'authentification ne doit être réalisé qu'une seule fois dans RStudio. La procédure est la suivante.

1. Dans RStudio (que vous aurez lancé au préalable via la machine virtuelle VirtualBox, ou en local), entrez la commande suivante dans votre console et appuyez sur `Entrée`.

   ```
   usethis::create_github_token()
   ```



Vous êtes automatiquement redirigé vers GitHub dans votre navigateur Web (il faut peut-être s'y identifier). Vous êtes face à une page qui s'appelle "New personal access token". Les champs sont déjà préremplis.



2. Modifiez la date d'expiration

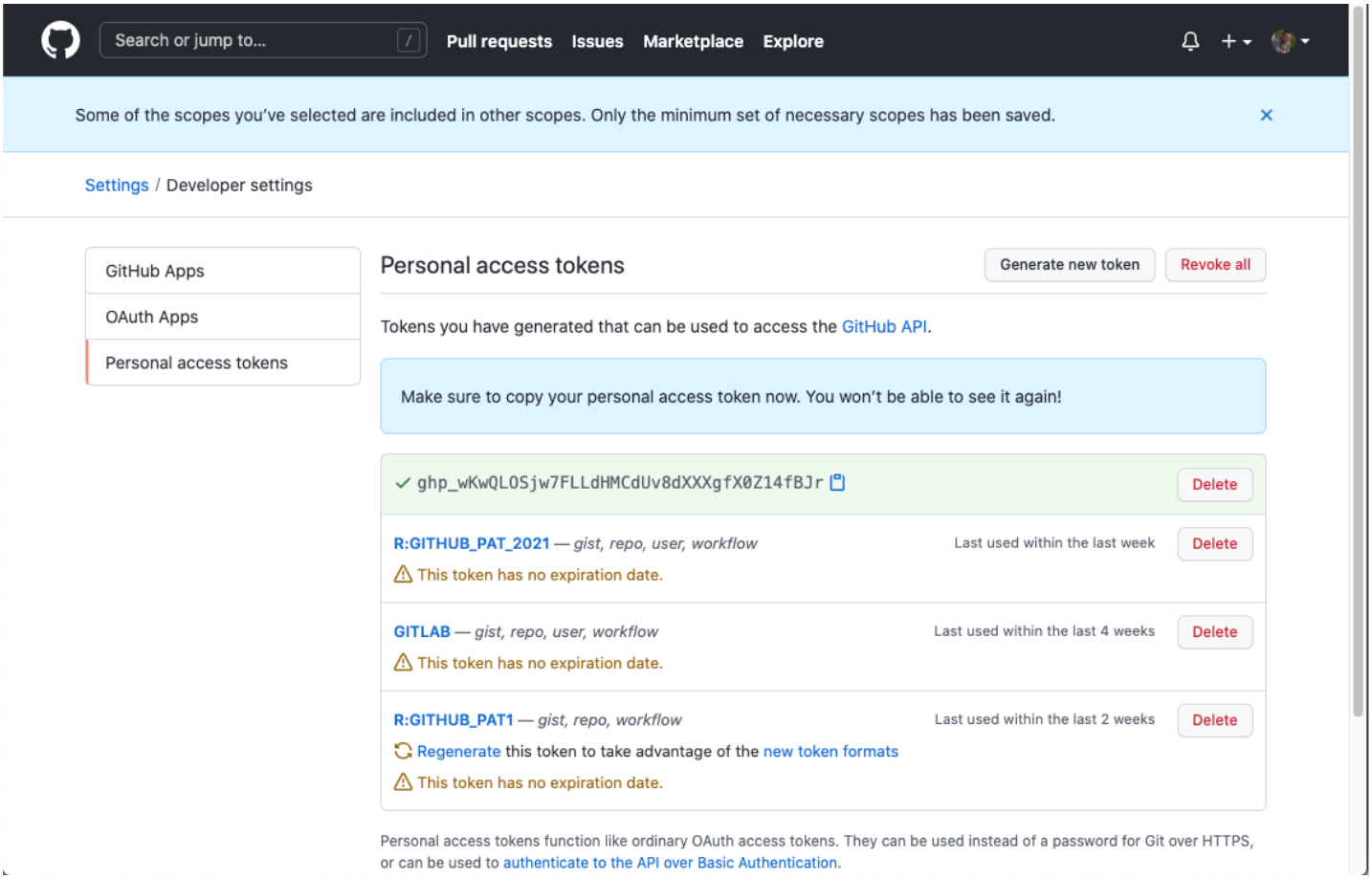
Vous pouvez conserver les options telles quelles. Cependant, votre clé d'authentification a une durée de validité limitée à 30 jours par défaut. C'est une bonne pratique de renouveler ses mots de passe régulièrement, mais chaque mois… c'est un peu court. Dans le champ `Expiration`, nous vous conseillons de changer la valeur pour couvrir toute votre année académique.

3. Générez votre "token" en cliquant sur le bouton vert "Generate token" tout en bas de la page.

Vous vous retrouver à présent dans une nouvelle page GitHub qui liste tous vos PATs (vous n'en aurez probablement qu'un seul à ce stade). Votre token est affiché dans un encadré en vert avec une petite icône de presse-papier à sa droite. Cliquez sur cette icône presse-

papier pour le copier. **Attention : une fois sorti de la page, vous ne pourrez plus jamais voir ce token (mais vous pourrez en recréer d'autres si nécessaire, pas de panique).**

> À ce propos, **n'enregistrer jamais ce token dans votre ordinateur, tout comme d'ailleurs n'importe quel mot de passe, à moins d'utiliser un logiciel coffre-fort comme Keeweb et ne les placez pas non plus dans un email ou un fil sur une messagerie dans l'espoir d'en conserver une trace**. Dans le cas du token, il n'est pas nécessaire du tout d'en conserver la trace parce que vous pouvez toujours en créer un nouveau si nécessaire. D'ailleurs, aucun de vos mots de passe ne doit se trouver dans vos notes, vos fichiers Word ou Excel, votre messagerie, etc. ni dans votre ordinateur, ni dans votre smartphone. **C'est une faute grave et une brèche de sécurité énorme que vous offrez aux hackers mal intentionnés que de stocker en clair vos mots de passe sous forme électronique.**



4. Revenez dans RStudio dans l'onglet **Console**. Comme indiqué dans les instructions, exécutez maintenant la commande suivante (puis appuyez sur `Entrée`):

   gitcreds::gitcreds_set()

5. Vous avez un message qui vous demande votre token. Collez-le dans la Console à l'aide du raccourci `Ctrl+V` (ou `Cmd+V` dans macOS) ou clic bouton droit dans l'onglet **Console** de RStudio et sélection de l'entrée de menu contextuel `Paste || Coller`. Votre token est collé dans la Console. Validez ensuite avec la touche `Entrée`.

```
> usethis::create_github_token()
● Call `gitcreds::gitcreds_set()` to register this token in the local
Git credential store
  It is also a great idea to store this token in any password-managem
ent software that you use
✓ Opening URL 'https://github.com/settings/tokens/new?scopes=repo,use
r,gist,workflow&description=R:GITHUB_PAT'
> gitcreds::gitcreds_set()


? Enter password or token: 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
```

Votre token doit à présent être enregistré dans la SciViews Box.

```
> usethis::create_github_token()
● Call `gitcreds::gitcreds_set()` to register this token in the local
Git credential store
  It is also a great idea to store this token in any password-managem
ent software that you use
✓ Opening URL 'https://github.com/settings/tokens/new?scopes=repo,use
r,gist,workflow&description=R:GITHUB_PAT'
> gitcreds::gitcreds_set()


? Enter password or token: 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
-> Adding new credentials...
-> Removing credetials from cache...
-> Done.
>
```

Une fois cela réalisé, vous pourrez travailler dans GitHub depuis RStudio (clone de dépôts, push/pull, …). Si plus tard, vous perdez ce lien et que RStudio se plaint de ne pas pouvoir communiquer avec GitHub et vous demande un mot de passe, vous devez alors régénérer un autre token et suivre la même procédure à nouveau.

B.2.3.2 Clé SSH dans SaturnCloud

Le principe de la clé SSH est assez différent. Vous créez dans votre ordinateur une paire de clés. L'une est publique et sert à encoder les données à un bout de la chaîne. L'autre est privée et ne réside *que* dans votre ordinateur. Elle est nécessaire pour pouvoir décoder les données. Donc, avec la clé publique, on peut encoder, mais pas décoder. La procédure de création des clés privée/publique est un rien plus compliquée que la génération du PAT. Mais SaturnCloud nous propose de le faire pour nous, et en plus, de générer une seule paire de clés pour l'ensemble de nos ressources. C'est donc bien pratique. *La procédure relative à la clé SSH entre SaturnCloud et GitHub est intégrée dans la page du site accessible depuis le bouton bleu **RStudio** en haut à droite. Cependant, au cas où vous devriez refaire l'opération, elle est brièvement rappelée ci-dessous.*

Dans SatunCloud, vous vous identifiez dans votre compte. Ensuite, vous cliquez sur **USER** et choisissez **Manage** . Là, vous avez une page nommée **Access Keys**. C'est la section du milieu intitulée **Git SSH Key** qui nous intéresse. Vous allez créer cette clé (en demandant à SaturnCloud de le faire pour vous). Ensuite, vous verrez une boite de dialogue qui expose la clé publique ainsi générée. Cliquez sur l'icône en forme de presse-papier pour la copier.

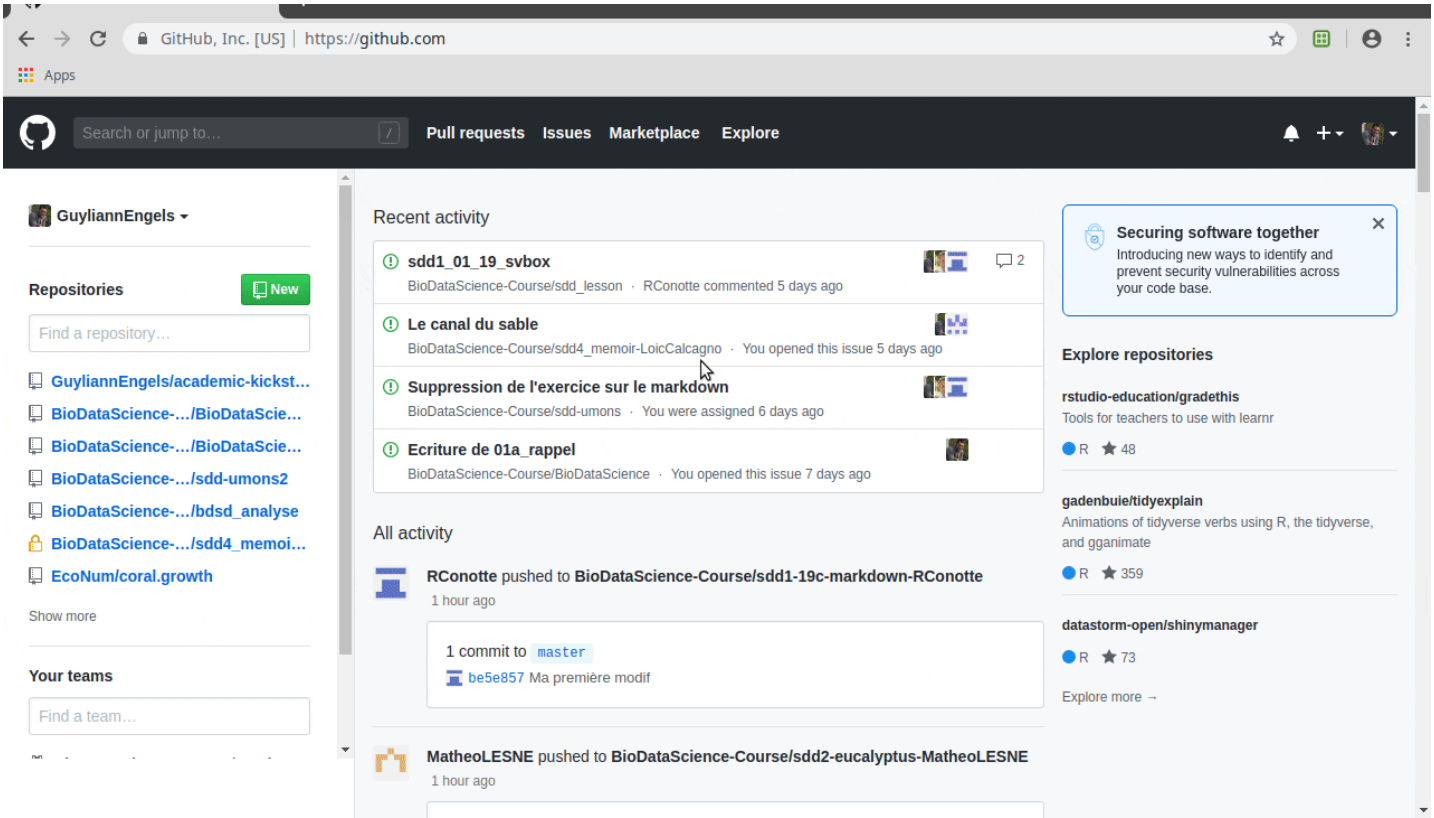
Rendez-vous dans un second temps dans GitHub.com où vous vous identifiez et vous allez dans les paramètres de votre compte. Pour cela, vous cliquez sur votre avatar en haut à droite de la page du site et vous sélectionnez **Settings** dans le menu déroulant. Vous allez dans la section **SSH and GPG keys** à gauche. Ensuite, dans **SSH keys** vous cliquez sur le bouton **New SSH key**. Vous indiquez un titre qui ne soit pas déjà utilisé si vous avez déjà d'autres clés enregistrées (nous vous conseillons `SaturnCloud`). Enfin, vous collez la clé publique depuis le presse-papier dans le champ **Key**. Vous terminez l'opération en cliquant sur le bouton **Add SSH key**. En principe, la communication entre RStudio et

GitHub est effective à présent. Cette clé n'expire pas et ne doit pas être recréée. Mais si vous suspectez un problème, vous pouvez toujours en recréer un autre dans SaturnCloud. Ensuite, dans GitHub vous effacez l'ancienne et ajouter la nouvelle de la même façon.

> Que ce soit via un PAT ou via la clé SSH, RStudio ne doit vous demander aucun mot de passe et ne doit pas afficher de message d'erreur de connexion lors d'opérations entre Rstudio et GitHub (clone, push, pull…). Lorsque vous clonerez vos dépôts depuis GitHub, vous devrez par contre penser à utiliser la référence **HTTPS** si vous utilisez un PAT, et la référence **SSH** si vous utilisez une clé SSH. **Si vous inversez les références, cela ne fonctionnera pas !**

B.2.4 Créer un dépôt

Lorsque nous souhaitons créer un nouveau dépôt GitHub, nous devons l'initialiser comme suit :



Pour créer un nouveau dépôt (`Create a new repository`), nous devons fournir les informations suivantes :

- `Repository template`

Nous devons décider d'utiliser ou non un "template" existant parmi la liste des templates que nous avons à disposition.

- `Owner`

Nous devons décider du responsable du dépôt, soit une organisation, soit une personne.

- `Repository name`

Nous devons choisir un nom court, mais pertinent pour notre dépôt et qui ne soit pas déjà utilisé dans l'organisation ou l'espace de l'utilisateur sélectionné.

- `Description`

Nous pouvons indiquer ici une courte description de notre dépôt.

- `Public` ou `Private`

Nous devons décider si notre projet est public ou privé.

- `README`

Nous pouvons éditer un fichier de présentation qui se nomme `README` . Ce dernier va présenter succinctement notre projet. On peut l'éditer depuis GitHub directement. Si ce fichier est au format Markdown (nettement conseillé), il se nommera alors `README.md` . Ainsi, vous pourrez formater les titres et le texte dans ce fichier.

- `.gitignore`

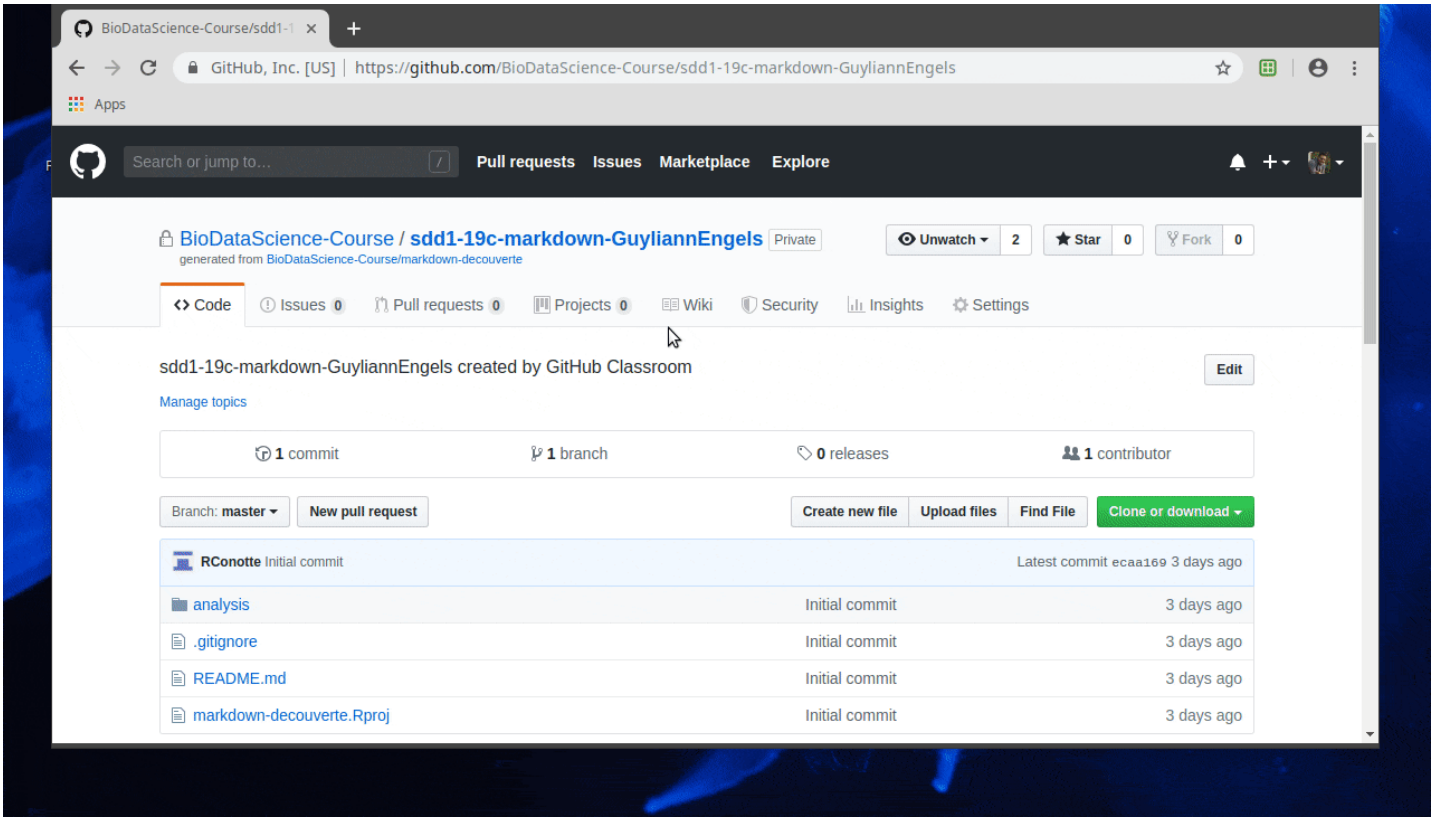
Il est intéressant de configurer le dépôt avec un fichier `.gitignore` orienté sur l'utilisation de **R**. GitHub peut en effet héberger des projets dans des langages informatiques très variés.

- `license`

Nous pouvons adjoindre à notre projet une licence. Il en existe plusieurs qui définissent précisément ce que l'on a le droit de faire ou non avec votre dépôt. Le site https://choosealicense.com peut vous aider à choisir la licence la plus appropriée en fonction du contenu que vous souhaitez y héberger. Une fois votre dépôt configuré, il ne vous reste plus qu'à le **cloner** comme expliqué dans la section B.2.5.

B.2.5 Cloner un dépôt existant via RStudio

Lorsque nous souhaitons travailler sur un de nos projets hébergés dans un dépôt GitHub, il faut commencer par le cloner pour avoir une copie en local de ce dernier. Notez que "local" signifie ici dans la machine sur laquelle nous travaillons. Si nous utilisons une ressource SaturnCloud ou tout autre service dans le Cloud comme Posit Cloud par exemple, le contenu du dépôt sera recopié dans cette machine-là dans le Cloud. Notre contenu sera donc "local" de manière relative au logiciel qui s'y exécute.



Pour commencer, vous devez copier le lien menant à votre dépôt GitHub. Il vous suffit de cliquer sur le bouton vert `Code` dans la section du même nom et de copier l'URL proposée dans le presse-papier (vous avez d'ailleurs un bouton avec une icône suggérant la copie pour le faire). **Mais attention, avant de faire cela, il faut déterminer quel format**

doit être copié. Si vous vous identifiez via un PAT, il faut récupérer la version **HTTPS**. Si vous utilisez une clé SSH, vous devez utiliser la version **SSH**. Donc pour simplifier, si vous utilisez la SciViews Box dans VirtualBox choisissez **HTTPS**. Si au contraire, vous utilisez la SciViews Box dans SaturnCloud, choisissez la version **SSH**.

Ensuite, vous devez vous rendre dans RStudio et cliquer sur `Project || Projet` en haut à droite, suivi de `New Project... || Nouveau projet...` (Si les projets restent encore un peu flous pour vous, rendez-vous dans la section @ref(#rs-projet)). Une nouvelle fenêtre s'ouvre. Vous devez sélectionner `Version Control || Contrôle de version`, puis `Git` dans les préférences successives.

Pour finir, vous devez :

- renseigner l'URL précédemment copiée depuis GitHub,
- choisir un nom à votre dépôt (une bonne pratique est de nommer votre projet du même nom que votre dépôt, sans l'extension `.git` à la fin),
- choisir un dossier pour cloner votre dépôt (le sous-dossier `projects`, du dossier `shared` est dédié à cela en version VirtualBox, ou le dossier `workspace` dans la version SaturnCloud) et
- créer une copie en local de votre projet en cliquant sur `Create Project || Créer un projet`.

Vous êtes à présent prêt à éditer votre projet. Vous réaliserez des **commits** (enregistrement d'une version de votre dépôt), des **pulls** (récupération en local des modifications de votre dépôt enregistrées dans GitHub) et des **pushes** (envoi de vos modifications locales dans le dépôt GitHub) pour enregistrer des versions successives de votre travail et pour les synchroniser avec le dépôt GitHub. Vous apprenez à faire cela dans le module 1 du cours.

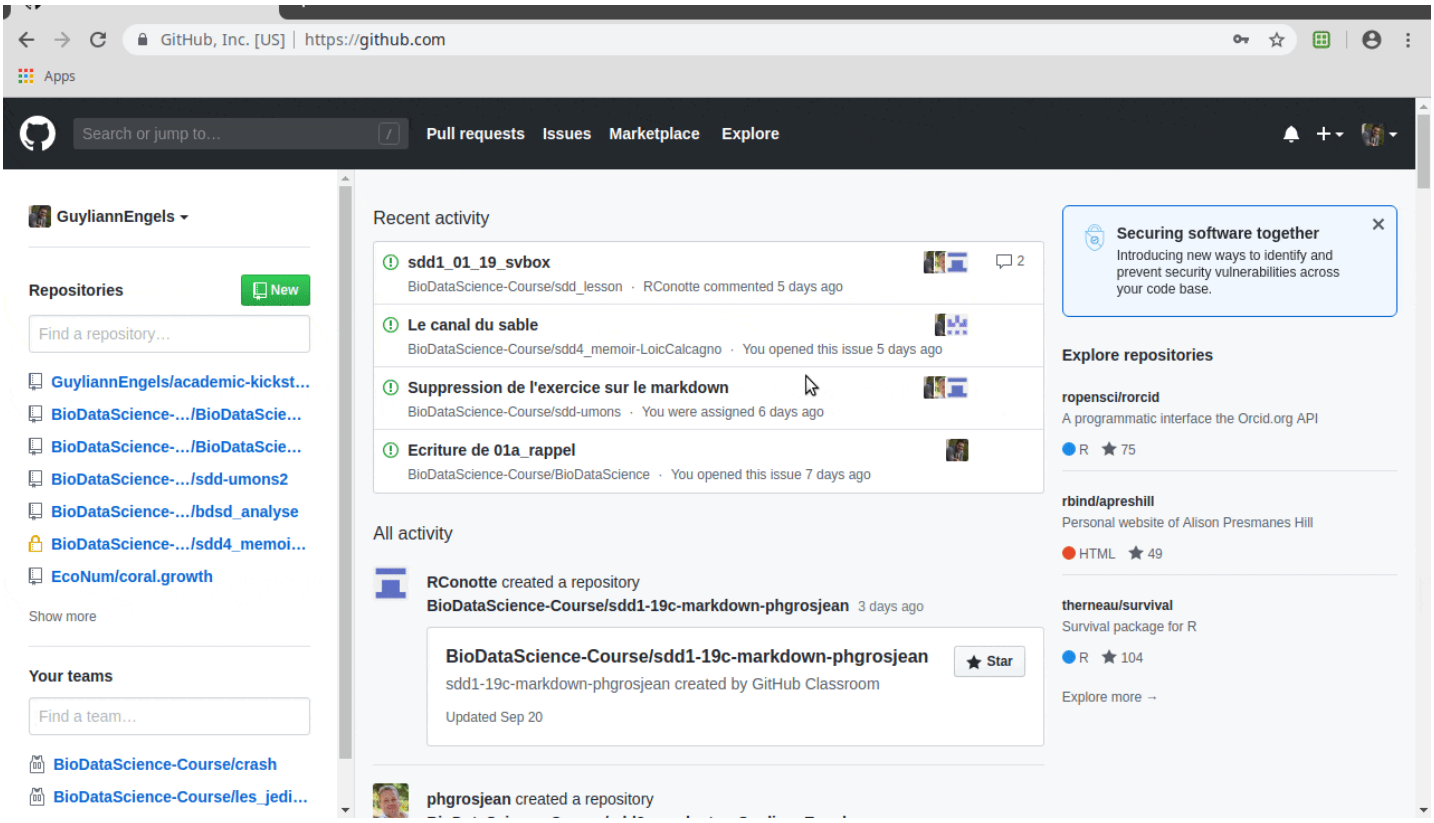
B.2.6 Déposer un projet déjà créé

Nous avons créé un projet dans RStudio et l'avons configuré avec le gestionnaire de version Git comme présenté dans l'annexe B.1.1.1. Cependant, après avoir progressé dans ce projet (et réalisé plusieurs **commits**), nous souhaitons le partager à présent via

GitHub. Rassurez-vous, il ne faut pas tout recommencer. Il aurait été plus simple de réfléchir dès le début du projet à cette éventualité et de créer le dépôt GitHub en premier lieu, mais cela reste parfaitement faisable à ce stade de transformer un projet local RStudio en un dépôt GitHub.

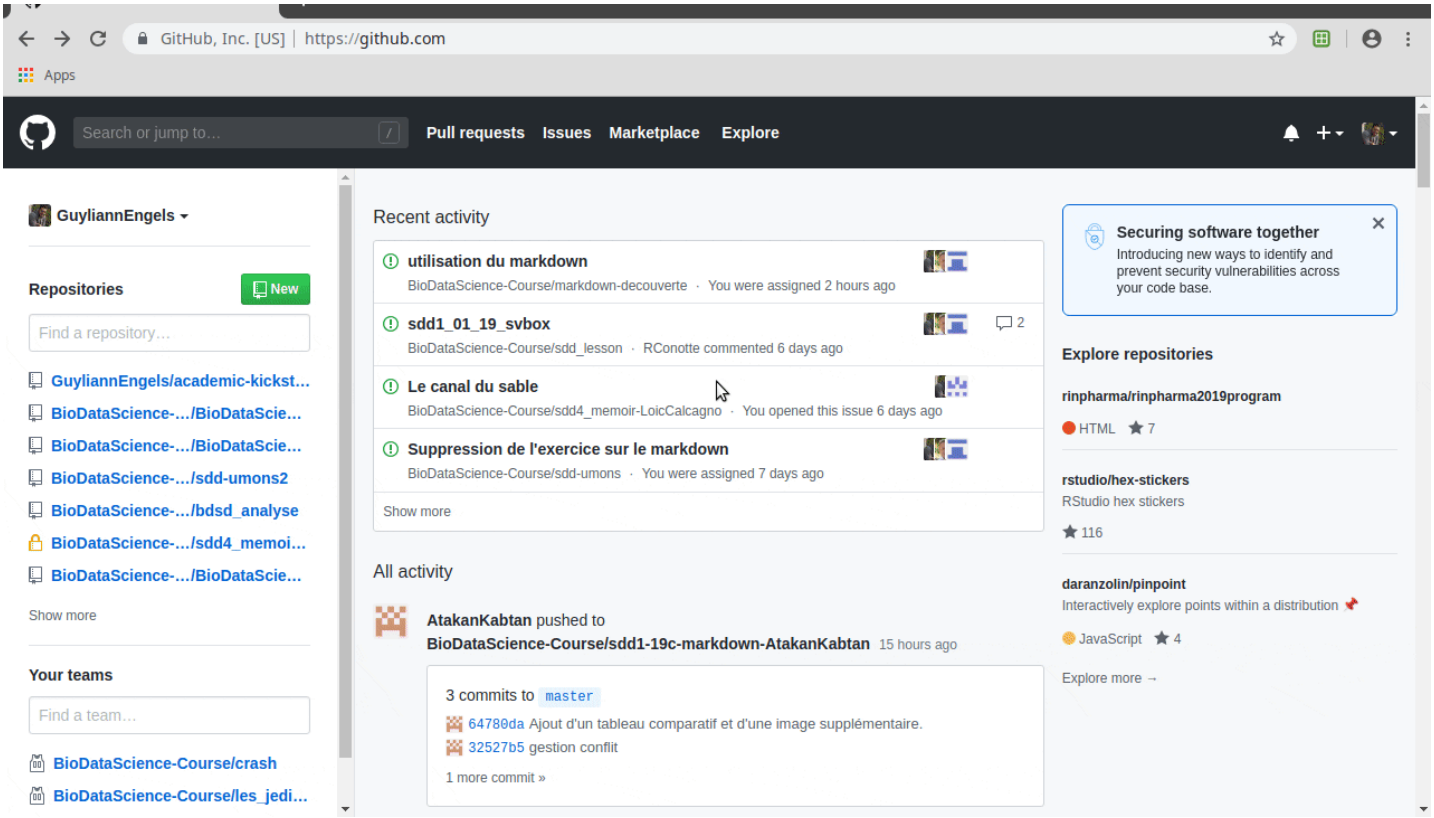
> Une bonne pratique avant de vous lancer dans un nouveau projet et de se poser et de réfléchir aux objectifs du projet et aux moyens à mettre en œuvre pour atteindre ces objectifs.

Pour les explications, nous partons d'un projet RStudio d'exemple fictif qui se nomme `repos-example`. Comme vous pouvez le voir, trois **commits** ont déjà été réalisés dans ce projet, mais nous ne pouvons pas faire de **pull** ou **push**, puisque notre projet n'est pas lié à un dépôt distant GitHub ou autre.

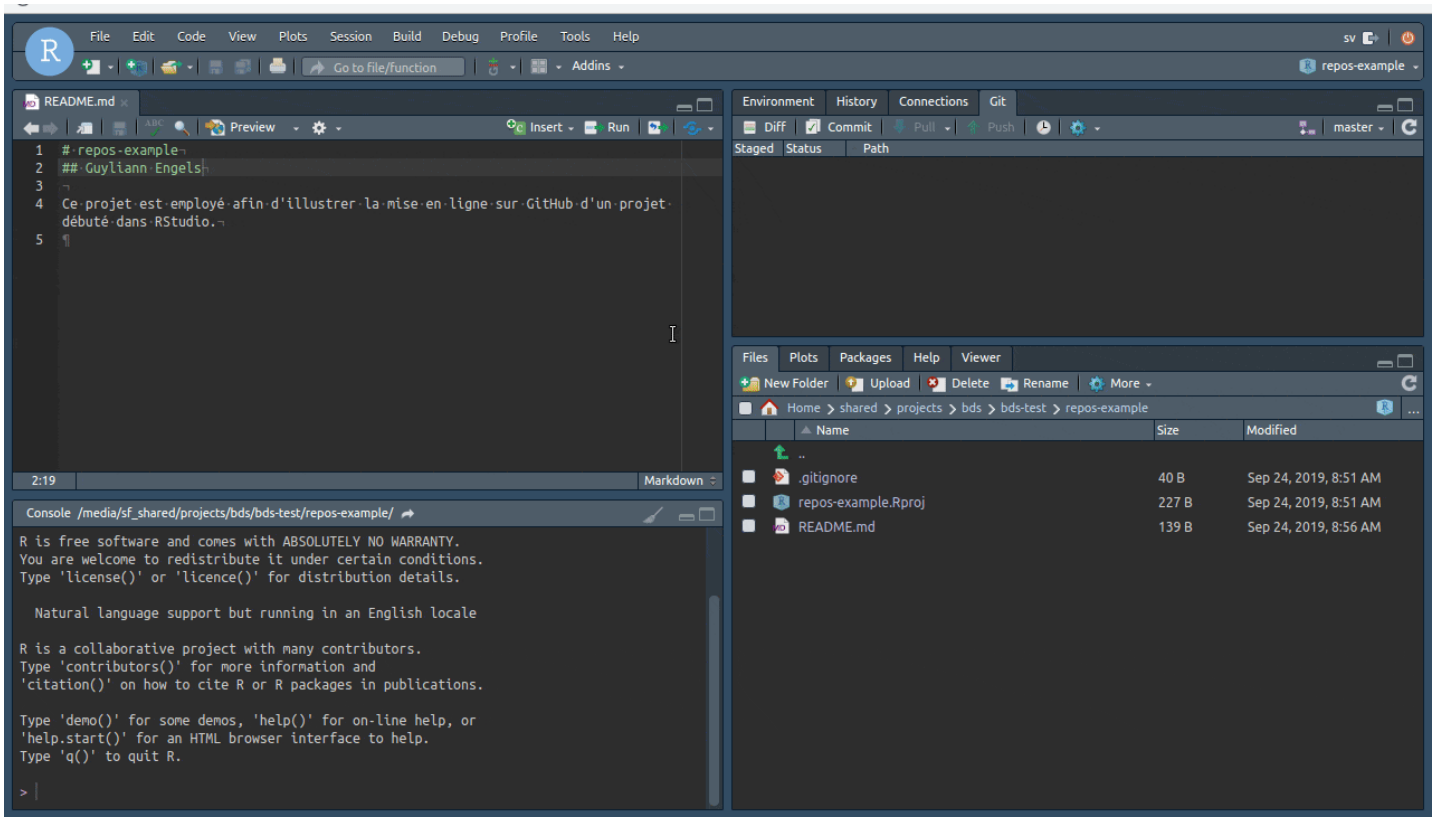


Pour déposer un projet RStudio existant dans GitHub, vous devez commencer par créer un nouveau dépôt dans Github qui ressemble très fortement à l'annexe B.2.4. Avec une particularité que vous ne devez pas configurer le `README`, le `.gitignore` et la `license`. Comme le dépôt est vide, GitHub vous propose plusieurs options pour le

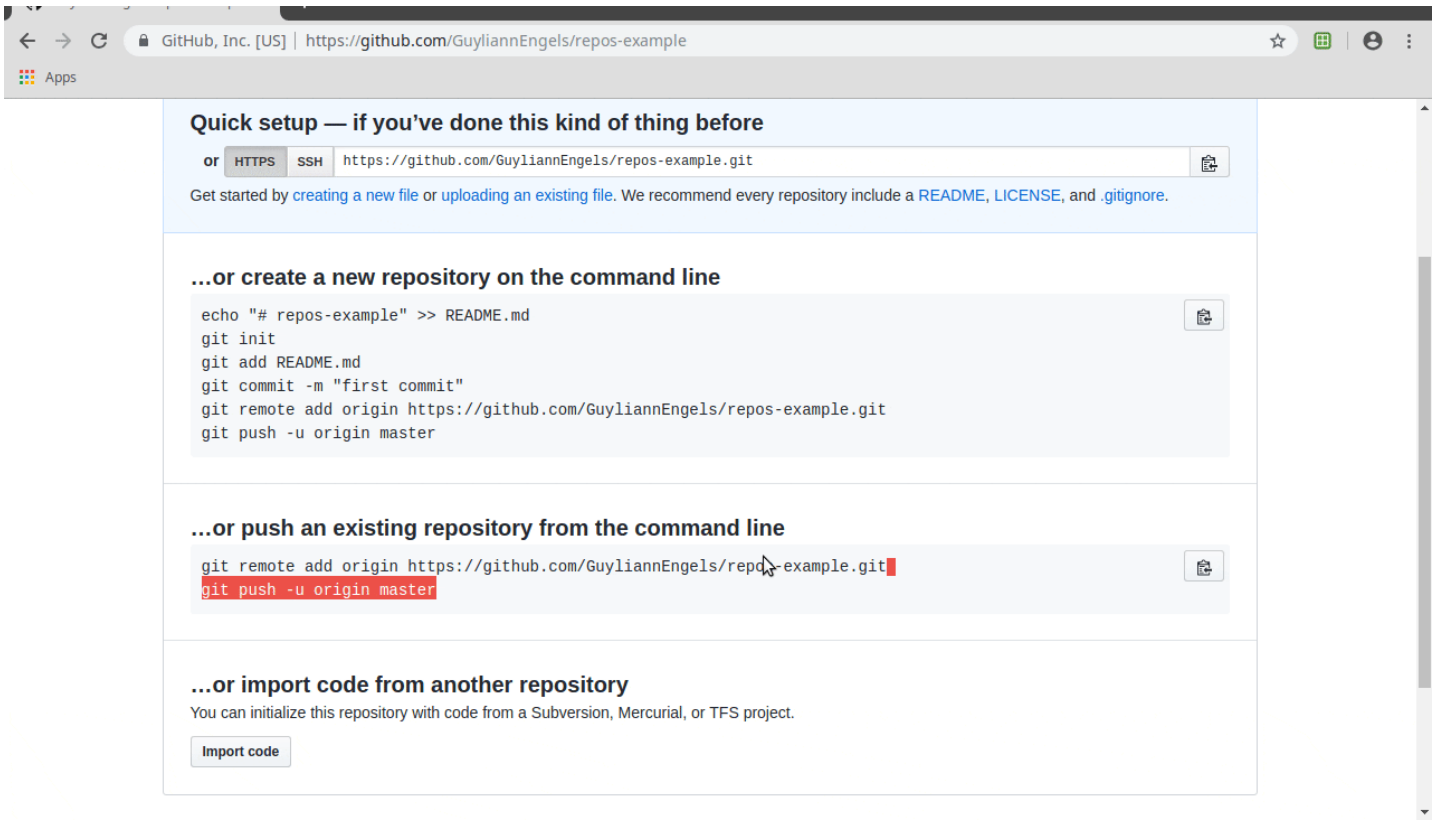
remplir, dont `…or push an existing repository from the command line` . Il s'agit donc de mettre en ligne (**push**) un projet existant sous forme de dépôt local Git. Copier les trois lignes de code proposées à cet endroit dans le presse-papier.



Dans votre projet RStudio, sélectionnez le menu `Tools || Plus` puis `Shell... || Nouveau terminal` . Un onglet **Terminal** vient de s'ouvrir à côté de l'onglet **Console** en bas à gauche. Il vous suffit ensuite d'y coller les trois instructions qui se trouvent dans le presse-papier et de taper sur la touche `Entrée` pour transformer votre projet git en un dépôt GitHub.



Pour vérifier que votre projet RStudio est correctement mis en ligne dans GitHub, vous pouvez recharger votre page dans https://github.com.

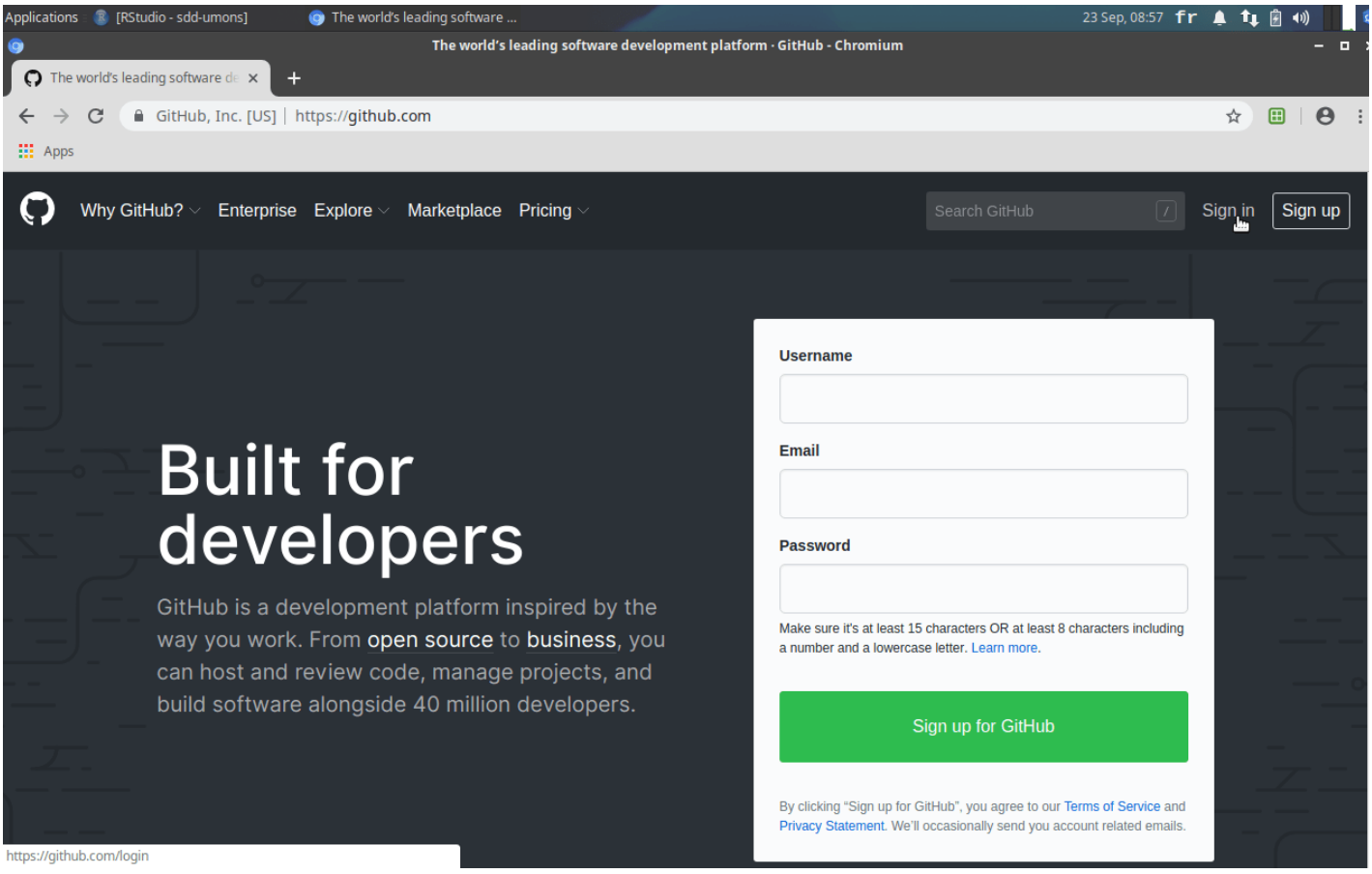


B.2.7 Copier un dépôt

Lorsque vous souhaitez apporter votre aide à un projet qui n'est pas le vôtre, vous devez réaliser un **fork** puis un **clone**. N'oubliez pas que la base de GitHub est de faciliter la collaboration. Pour soumettre vos modifications au projet de départ, il faut faire un **pull request**. Cette étape permet de proposer vos modifications au responsable du projet original sous une forme qui lui permet de visualiser et de discuter les changements proposés. Il pourra alors, en connaissance de cause, accepter ou refuser vos modifications.

B.2 GitHub

Un réseau social a été conçu autour de Git pour sauvegarder vos projets sur le “cloud”, les partager et collaborer avec d’autres personnes. Ce système se nomme GitHub (tout comme Facebook ou LinkedIn). GitHub rassemble donc “Git”, la gestion de version et “Hub” relatif au réseau. D’autres réseaux équivalents existent comme Gitlab ou Bitbucket.



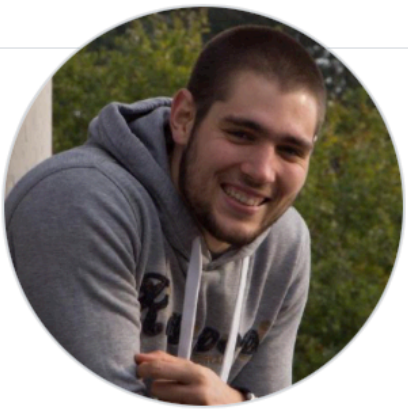
B.2.1 Votre activité et profil

Pour vous montrer différentes sections sur GitHub, nous utiliserons le compte de GuyliannEngels. Une fois enregistré dans GitHub, nous nous trouvons dans une page qui nous montre notre activité. Au milieu de la page, nous pouvons observer les dépôts

épinglés (section “Pinned”) que vous considérez comme les plus importants.

Search or jump to... / Pull requests Issues Marketplace Explore

Overview Repositories 5 Projects Packages Stars 20



Guyliann Engels
GuyliannEngels

Unfollow

R developper

10 followers · **4** following

University of Mons
Mons, Belgium
guyliann.engels@umons.ac.be
@EngelsGuyliann

Achievements

Beta Send feedback

Organizations

Block or Report

GuyliannEngels / README.md

GuyliannEngels

I'm Guyliann from Belgium. I'm research and teaching assistant at Numerical Ecology of Aquatic systems laboratory, University of Mons (UMONS).

My languages of choice are R.

- I'm currently working on
  - the Data Science courses for biologists
  - the diversity of marine plankton : identification guide to mesoplankton

Recent GitHub Activity

1. Commented on #3 in BioDataScience-Course/BioDataScience
2. Commented on #3 in BioDataScience-Course/BioDataScience

GitHub Stats ⭐

Guyliann Engels' GitHub Stats

Total Stars Earned:	0
Total Commits (2022):	168
Total PRs:	8
Total Issues:	49
Contributed to:	20

A+

Pinned

BioDataScience-Course/teaching_data_science_in_biology
Public
R ☆ 2

EcoNum/zooimage_mesozooplankton_guide3
Public
Guide d'identification du mésozooplancton en Mer Ligurienne
TeX

covid19_belgium Public
Shinydashboard on COVID-19
R

BioDataScience-Course/BioDataScience
Public
A Series of Learnr Documents to study Biological Data Science
R ☆ 6 10

Plus bas dans la page, il y a un graphique qui représente vos contributions au cours du temps. Il indique de manière globale votre travail, c'est-à-dire vos contributions dans les différents projets.

Dans notre exemple, nous pouvons observer 983 contributions pendant l'année écoulée.

B.2.2 Vos projets

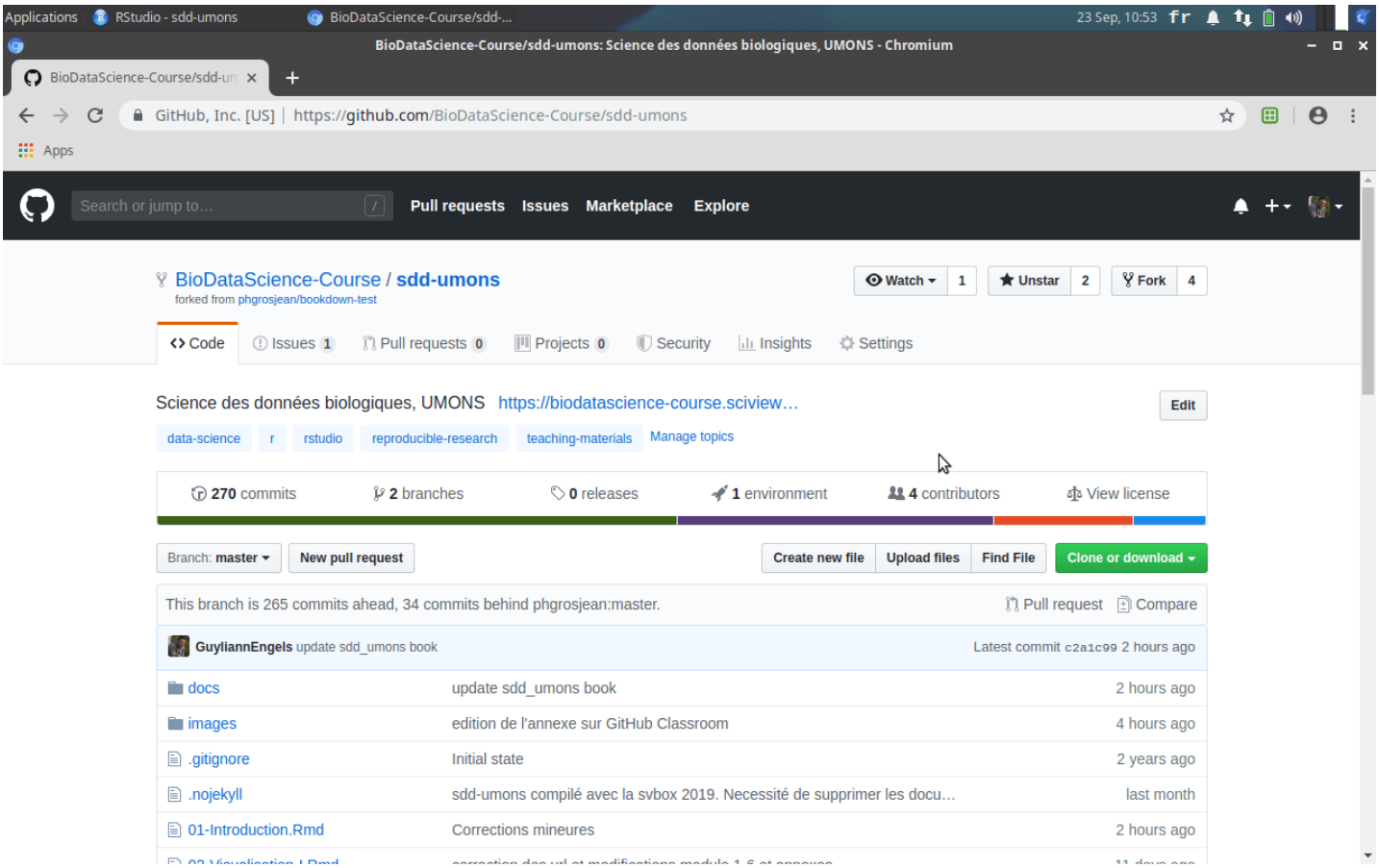
GitHub vous sert à héberger vos projets dans des **dépôts** (qui se nomment des *repositories* en anglais). Notre exemple se base sur le dépôt sdd-umons-2024, que vous pouvez librement consulter et qui contient les sources d'une version antérieure du présent ouvrage en ligne. Il s'agit en effet d'un dépôt public. Vous avez la possibilité d'avoir des projets publics ou privés (vos exercices dans le cadre du cours seront, eux, dans des dépôts privés visibles uniquement de vous et de vos enseignants).

Les projets publics sont visibles par tous. La collaboration est la pierre angulaire de GitHub. Donc, tout le monde peut y apporter des modifications dans des copies de vos dépôts et puis vous les proposer d'une manière qui vous permet d'examiner les améliorations suggérées, de les discuter, et enfin de les incorporer ou de les rejeter. Cela s'appelle un **Pull request**, mais nous y reviendrons plus tard. Pour des projets plus sensibles, vous avez la possibilité de les héberger en mode privé et de ne les partager qu'avec un nombre restreint et prédéterminé d'utilisateurs.

La référence à un dépôt sur GitHub est composée de deux éléments. Le premier est le nom de la personne ou de l'organisation auquel le dépôt appartient. Le second élément est le nom du dépôt lui-même. Dans notre exemple, nous aurons donc `BioDataScience-`

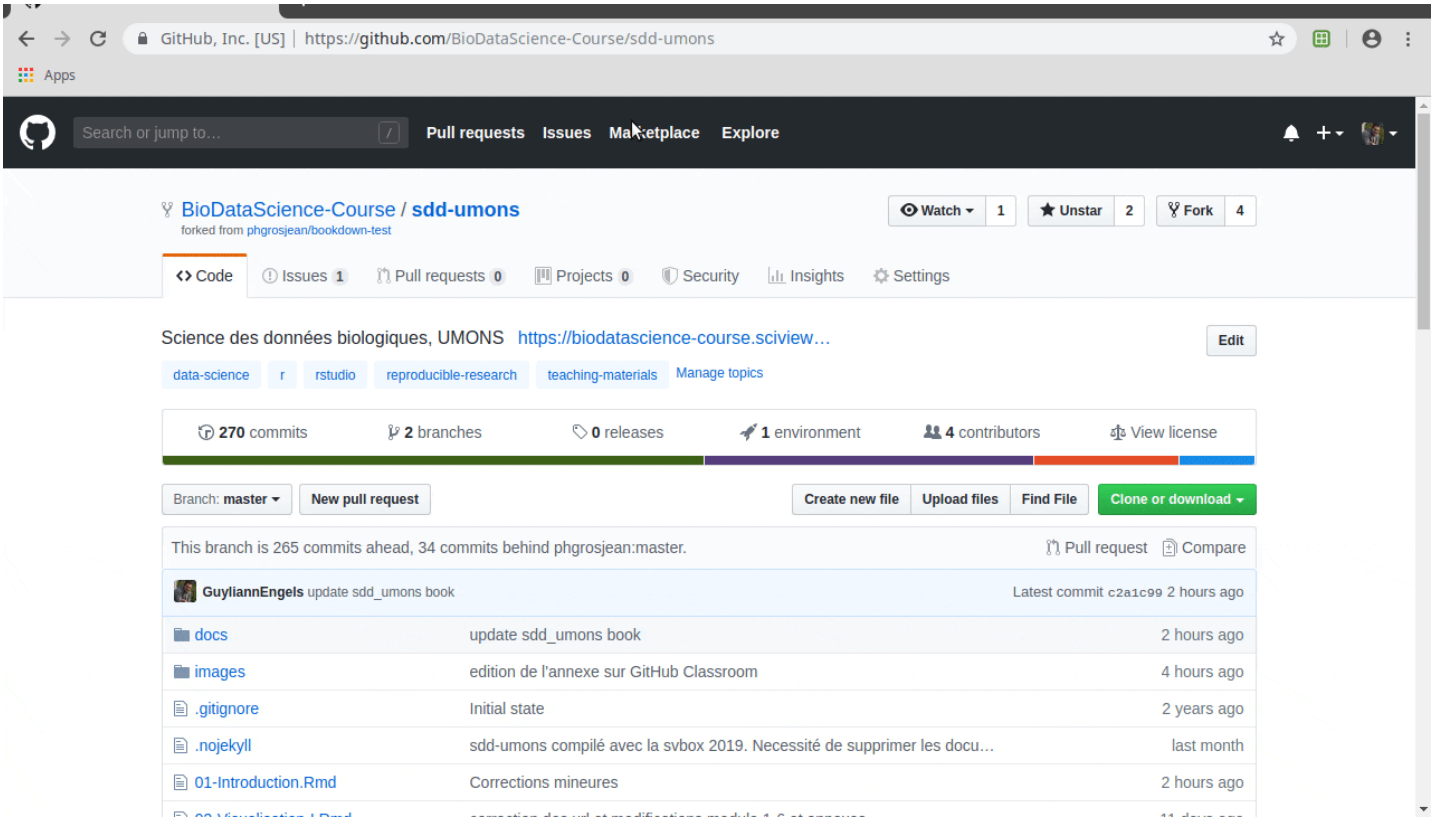
`Course/sdd-umons-2024` . Tous les projets relatifs au cours de sciences des données biologiques à l'UMONS sont hébergés dans l'organisation BioDataScience-Course (Il en sera de même pour tous les travaux que vous réaliserez dans le cadre de vos cours).

Vous pouvez observer une première barre d'outils comprenant les sections `Code` , `Issues` , `Pull requests` , `Actions` , `Projects` , `Wiki` , `Security` , `Insights` et `Settings` (par la suite, nous ne détaillerons que les sections importantes dans le cadre de votre cours de science des données).

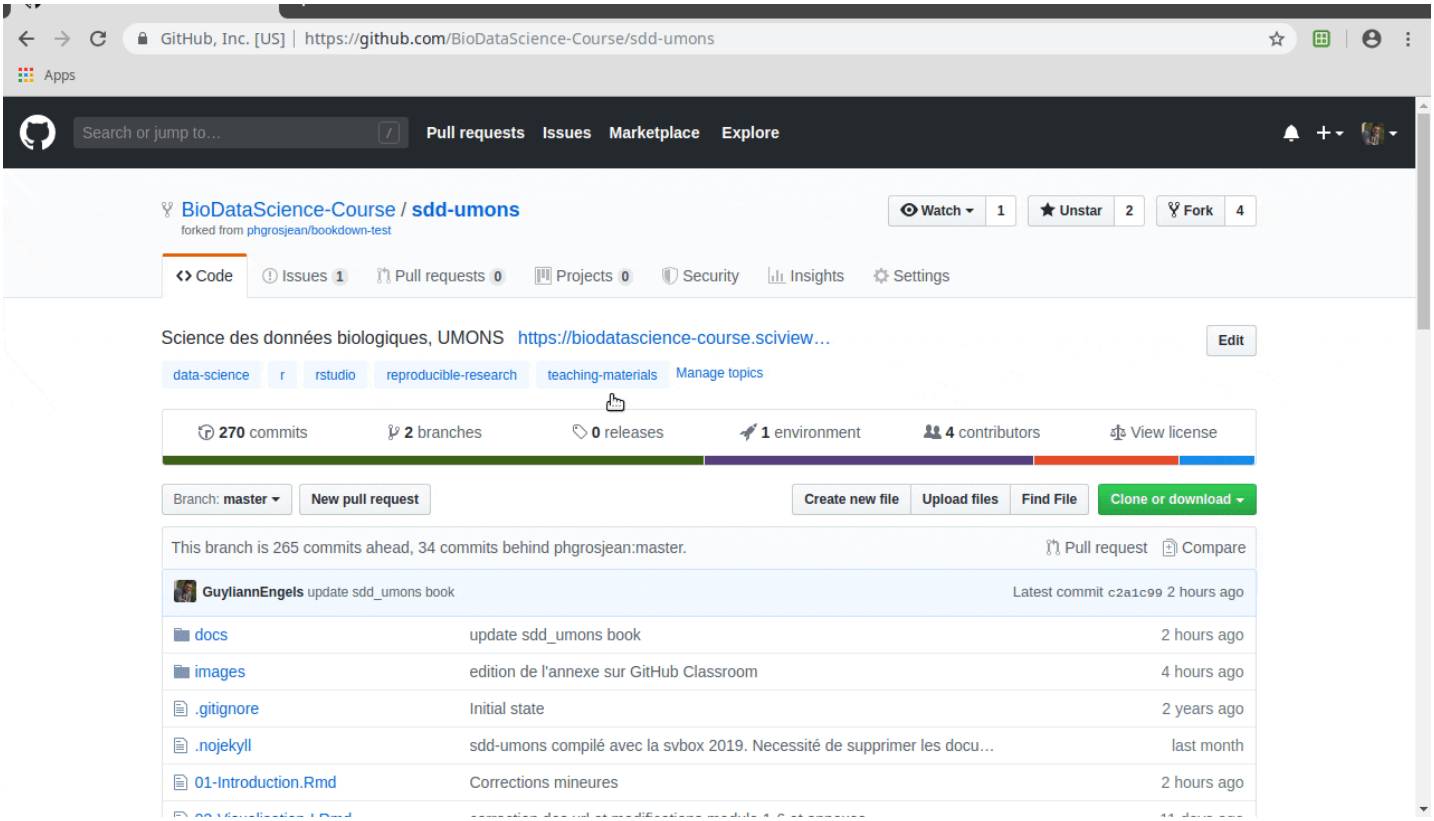


B.2.2.1 Code

Dans cette section, vous pouvez visualiser le contenu des différents fichiers qui se trouvent dans le dépôt. Vous naviguez dans les sous-dossiers simplement en cliquant dessus. La visualisation des fichiers se fait, éventuellement, selon plusieurs présentations différentes, dont le mode "Raw" pour voir le texte brut. C'est l'équivalent du mode "Source" dans l'éditeur de documents R Markdown de RStudio. Le bouton vert `Code` dans cette section est important. C'est grâce à lui que vous pourrez récupérer le contenu du dépôt. On parle de **cloner** un dépôt.

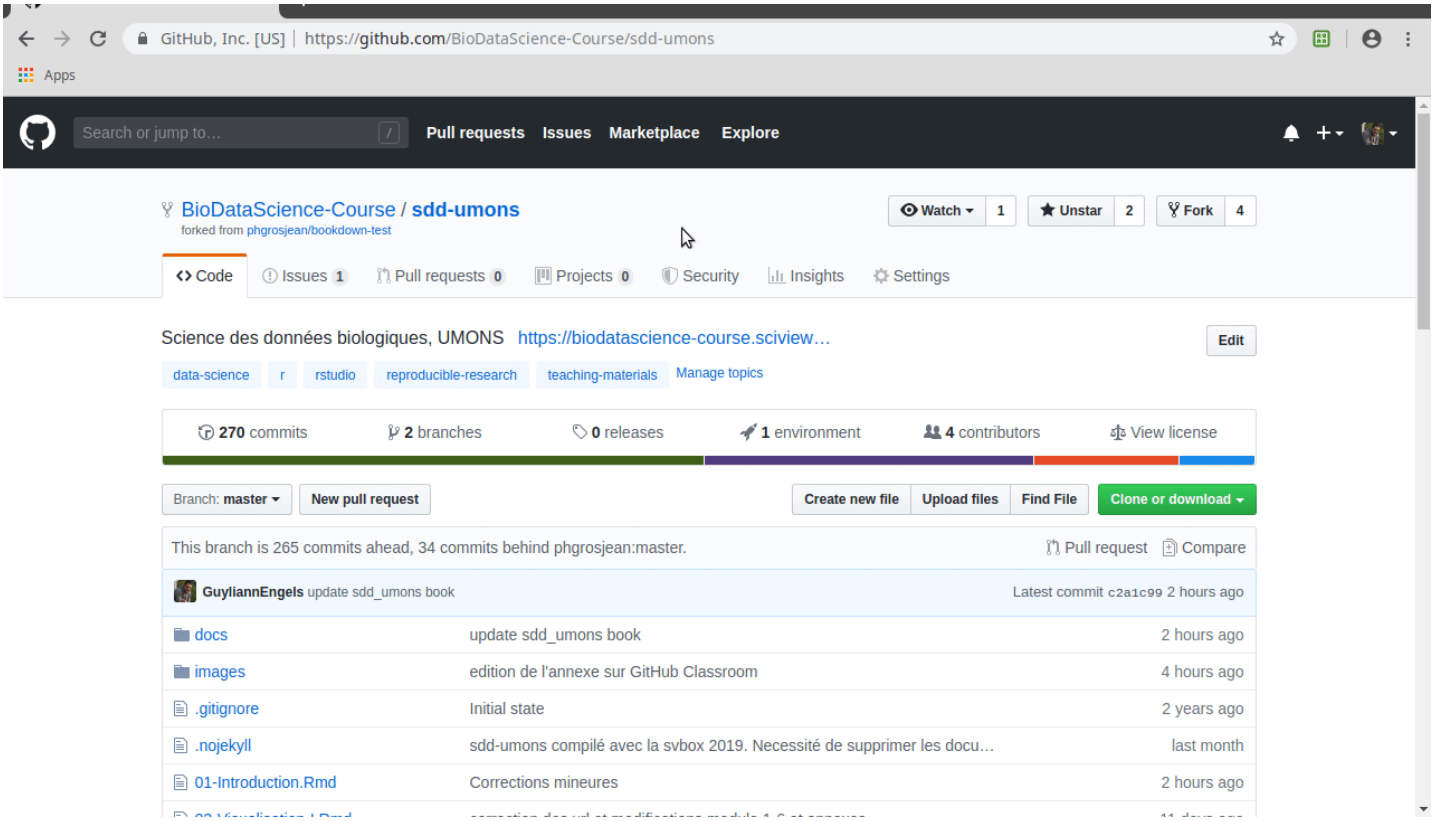


Comme il s'agit d'un système de gestion de versions, toutes les versions successives du dépôts sont stockées et accessibles, mais au départ, vous ne voyer que la dernière version. Chaque fois que vous enregistrez une version, vous réaliser ce que l'on appelle dans le jargon Git un **commit**. Le nombre de commits et des informations à son propos sont présentées dans la barre grisée sous les quelques boutons.



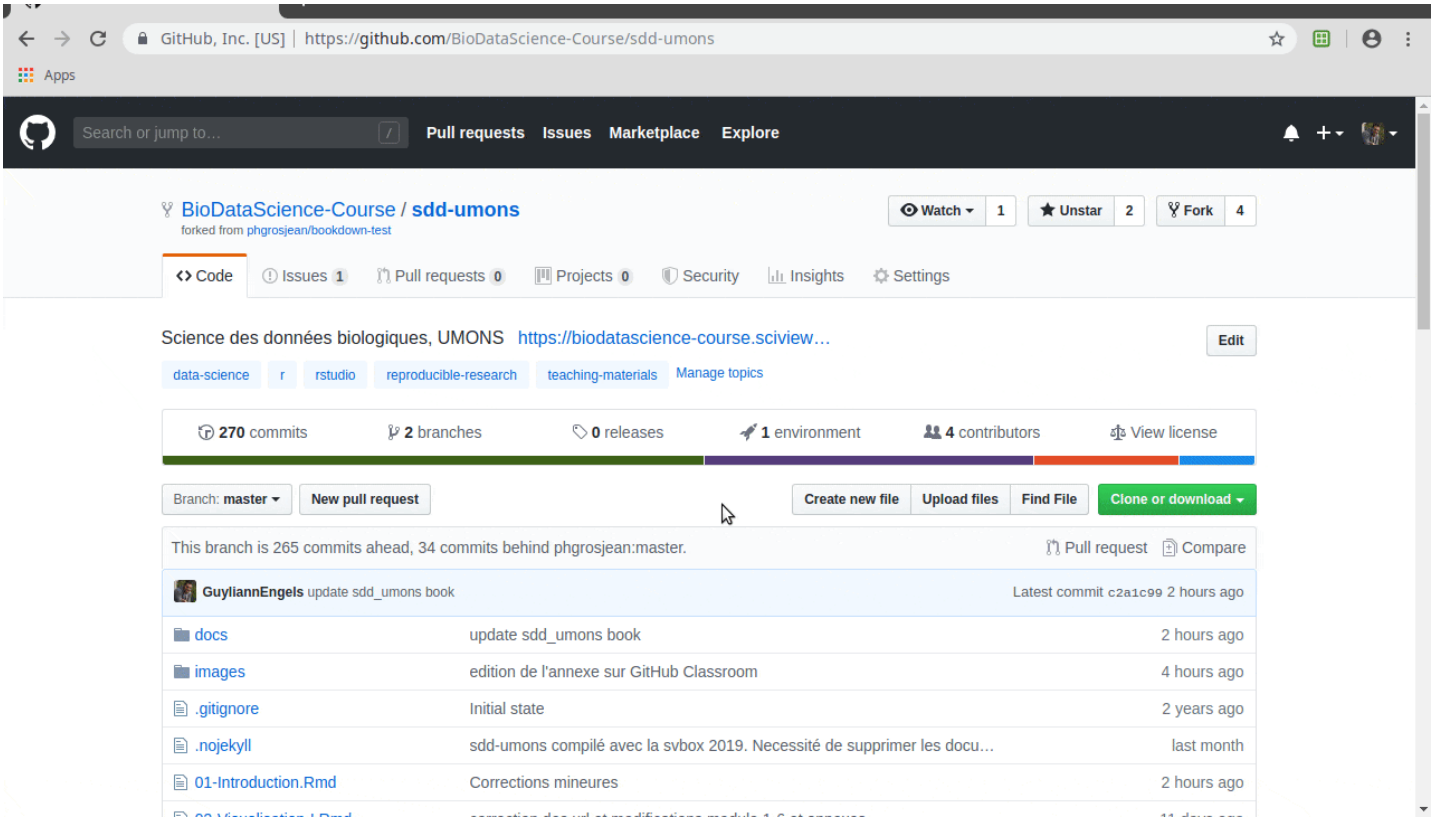
B.2.2.2 Issues

Cette section est prévue pour discuter des problèmes ou des idées à développer. C'est une sorte de petit forum de discussion. Vous utiliserez ces "issues" pour poser vos questions à vos enseignants et pour interagir entre vous dans les projets de groupe.



B.2.2.3 Insights

La section `Insights` nous renseigne sur l'activité du projet. On peut y voir, par exemple, les contributeurs du projet, le nombre de commits réalisés par chacun, etc.



Concernant vos projets dans le cadre du cours, ces informations sont employées pour évaluer la contribution de chacun dans les travaux de groupes et ajuster la note en fonction.

B.2.3 Permettre à RStudio de communiquer avec GitHub

Dans le cours de Science des Données vous utiliserez RStudio pour éditer vos documents et travailler avec R. Mais d'autre part, vous partagerez vos œuvres via GitHub. C'est d'ailleurs un schéma classique que nous vous conseillons d'adopter aussi en dehors du cours ! Les professionnels travaillent tous comme cela. Nous devons donc trouver un moyen de permettre à RStudio de communiquer avec GitHub. Les fonctionnalités existent, dans l'onglet **Git** (qui n'apparaît que si vous êtes dans un projet géré par Git). Encore faut-il indiquer à GitHub que vous autoriser votre RStudio à accéder à votre compte et à vos dépôts. Pour cela, deux solution très différentes existent : le code d'accès personnel **PAT**, un code généré par GitHub et que vous incluez dans votre SciViews Box, ou la **clé SSH**, une clé privée/publique générée au contraire dans votre machine et dont vous renseignez

la partie publique à GitHub pour qu'il puisse l'utiliser. Nous allons voir ces deux méthodes successivement, sachant que la version VirtualBox de la SciViews Box utilise le PAT, alors que c'est plus facile d'utiliser la clé SSH dans SaturnCloud.

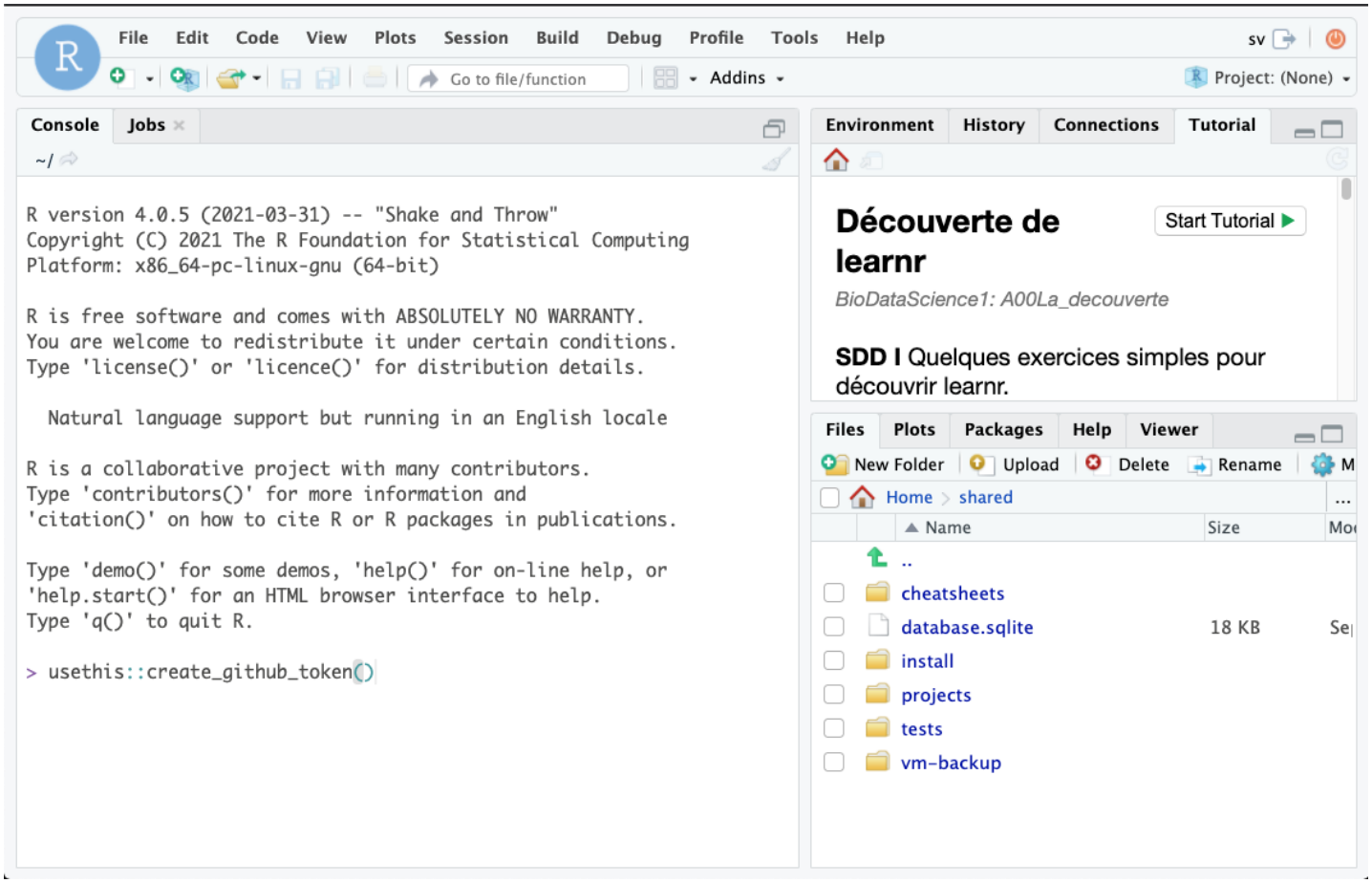
B.2.3.1 Personal Access Token

Note : dans SaturnCloud, une autre technique basée sur la clé SSH est utilisée. Dans ce cas, voyez la section suivante.

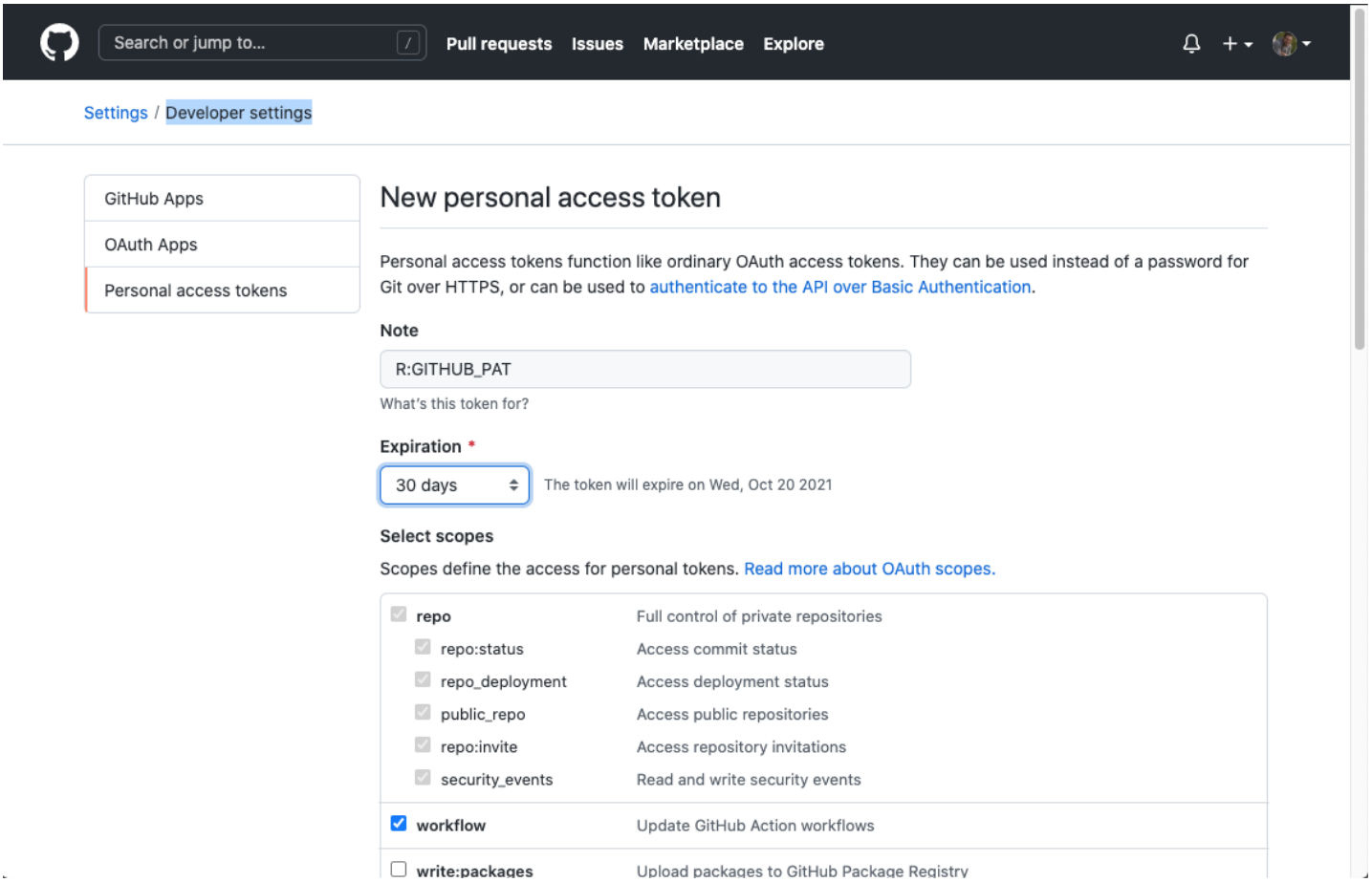
L'authentification dans GitHub peut se faire via un *Personal Access Token* (PAT). Il s'agit d'une sorte de mot de passe long et compliqué (donc bien sécurisé) que GitHub génère pour vous et que vous devez renseigner à tout logiciel souhaitant accéder à votre compte. Ce système d'authentification ne doit être réalisé qu'une seule fois dans RStudio. La procédure est la suivante.

1. Dans RStudio (que vous aurez lancé au préalable via la machine virtuelle VirtualBox, ou en local), entrez la commande suivante dans votre console et appuyez sur `Entrée`.

   ```
   usethis::create_github_token()
   ```



Vous êtes automatiquement redirigé vers GitHub dans votre navigateur Web (il faut peut-être s'y identifier). Vous êtes face à une page qui s'appelle "New personal access token". Les champs sont déjà préremplis.



2. Modifiez la date d'expiration

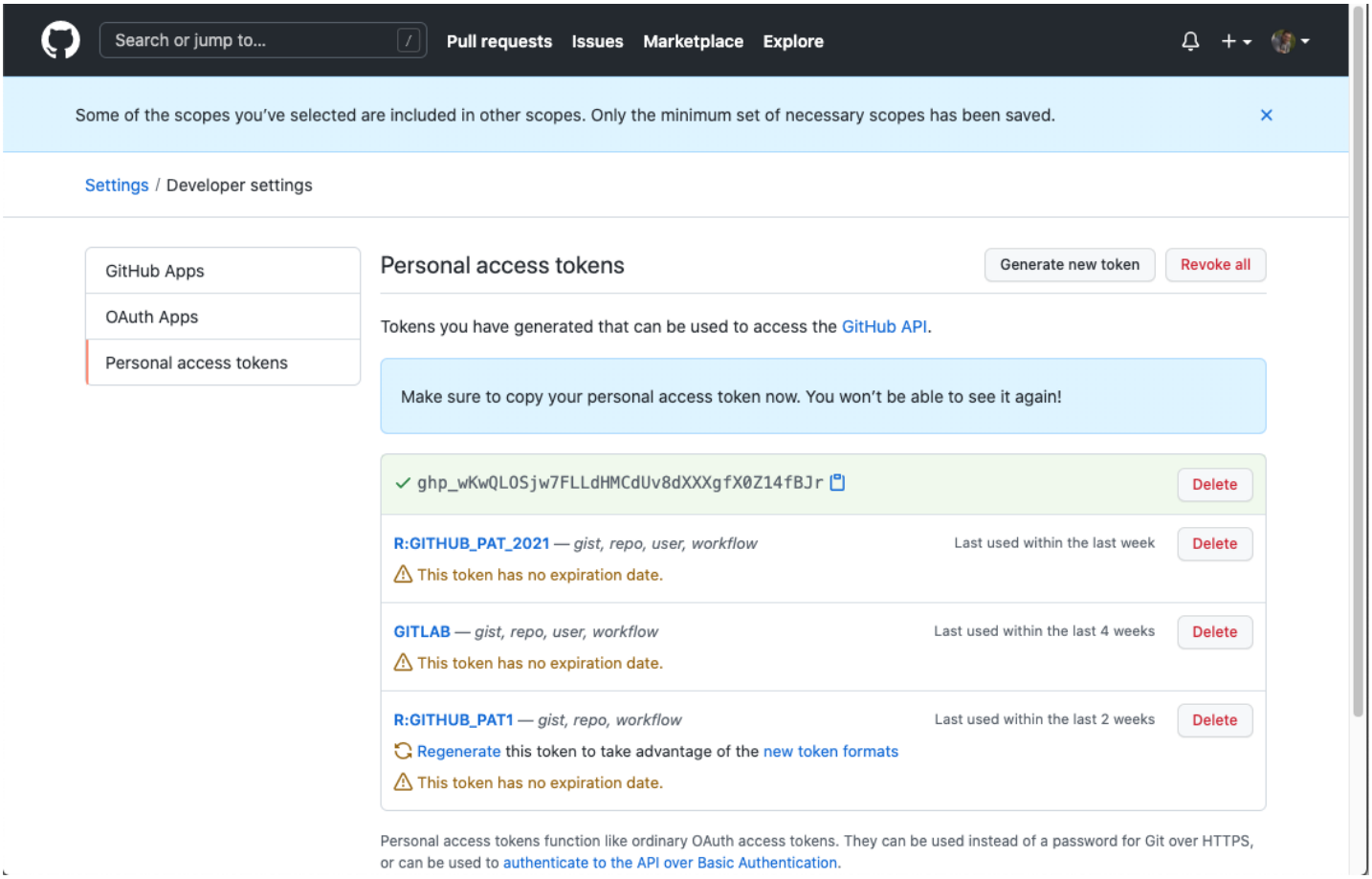
Vous pouvez conserver les options telles quelles. Cependant, votre clé d'authentification a une durée de validité limitée à 30 jours par défaut. C'est une bonne pratique de renouveler ses mots de passe régulièrement, mais chaque mois… c'est un peu court. Dans le champ `Expiration`, nous vous conseillons de changer la valeur pour couvrir toute votre année académique.

3. Générez votre "token" en cliquant sur le bouton vert "Generate token" tout en bas de la page.

Vous vous retrouver à présent dans une nouvelle page GitHub qui liste tous vos PATs (vous n'en aurez probablement qu'un seul à ce stade). Votre token est affiché dans un encadré en vert avec une petite icône de presse-papier à sa droite. Cliquez sur cette icône presse-

papier pour le copier. **Attention : une fois sorti de la page, vous ne pourrez plus jamais voir ce token (mais vous pourrez en recréer d'autres si nécessaire, pas de panique).**

> À ce propos, **n'enregistrer jamais ce token dans votre ordinateur, tout comme d'ailleurs n'importe quel mot de passe, à moins d'utiliser un logiciel coffre-fort comme Keeweb et ne les placez pas non plus dans un email ou un fil sur une messagerie dans l'espoir d'en conserver une trace**. Dans le cas du token, il n'est pas nécessaire du tout d'en conserver la trace parce que vous pouvez toujours en créer un nouveau si nécessaire. D'ailleurs, aucun de vos mots de passe ne doit se trouver dans vos notes, vos fichiers Word ou Excel, votre messagerie, etc. ni dans votre ordinateur, ni dans votre smartphone. **C'est une faute grave et une brèche de sécurité énorme que vous offrez aux hackers mal intentionnés que de stocker en clair vos mots de passe sous forme électronique.**



4. Revenez dans RStudio dans l'onglet **Console**. Comme indiqué dans les instructions, exécutez maintenant la commande suivante (puis appuyez sur `Entrée`):

   gitcreds::gitcreds_set()

5. Vous avez un message qui vous demande votre token. Collez-le dans la Console à l'aide du raccourci `Ctrl+V` (ou `Cmd+V` dans macOS) ou clic bouton droit dans l'onglet **Console** de RStudio et sélection de l'entrée de menu contextuel `Paste || Coller`. Votre token est collé dans la Console. Validez ensuite avec la touche `Entrée`.

```
> usethis::create_github_token()
● Call `gitcreds::gitcreds_set()` to register this token in the local
Git credential store
  It is also a great idea to store this token in any password-managem
ent software that you use
✓ Opening URL 'https://github.com/settings/tokens/new?scopes=repo,use
r,gist,workflow&description=R:GITHUB_PAT'
> gitcreds::gitcreds_set()


? Enter password or token: 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
```

Votre token doit à présent être enregistré dans la SciViews Box.

```
> usethis::create_github_token()
● Call `gitcreds::gitcreds_set()` to register this token in the local
Git credential store
  It is also a great idea to store this token in any password-managem
ent software that you use
✓ Opening URL 'https://github.com/settings/tokens/new?scopes=repo,use
r,gist,workflow&description=R:GITHUB_PAT'
> gitcreds::gitcreds_set()


? Enter password or token: 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
-> Adding new credentials...
-> Removing credetials from cache...
-> Done.
>
```

Une fois cela réalisé, vous pourrez travailler dans GitHub depuis RStudio (clone de dépôts, push/pull, …). Si plus tard, vous perdez ce lien et que RStudio se plaint de ne pas pouvoir communiquer avec GitHub et vous demande un mot de passe, vous devez alors régénérer un autre token et suivre la même procédure à nouveau.

B.2.3.2 Clé SSH dans SaturnCloud

Le principe de la clé SSH est assez différent. Vous créez dans votre ordinateur une paire de clés. L'une est publique et sert à encoder les données à un bout de la chaîne. L'autre est privée et ne réside *que* dans votre ordinateur. Elle est nécessaire pour pouvoir décoder les données. Donc, avec la clé publique, on peut encoder, mais pas décoder. La procédure de création des clés privée/publique est un rien plus compliquée que la génération du PAT. Mais SaturnCloud nous propose de le faire pour nous, et en plus, de générer une seule paire de clés pour l'ensemble de nos ressources. C'est donc bien pratique. *La procédure relative à la clé SSH entre SaturnCloud et GitHub est intégrée dans la page du site accessible depuis le bouton bleu **RStudio** en haut à droite. Cependant, au cas où vous devriez refaire l'opération, elle est brièvement rappelée ci-dessous.*

Dans SatunCloud, vous vous identifiez dans votre compte. Ensuite, vous cliquez sur **USER** et choisissez **Manage** . Là, vous avez une page nommée **Access Keys**. C'est la section du milieu intitulée **Git SSH Key** qui nous intéresse. Vous allez créer cette clé (en demandant à SaturnCloud de le faire pour vous). Ensuite, vous verrez une boite de dialogue qui expose la clé publique ainsi générée. Cliquez sur l'icône en forme de presse-papier pour la copier.

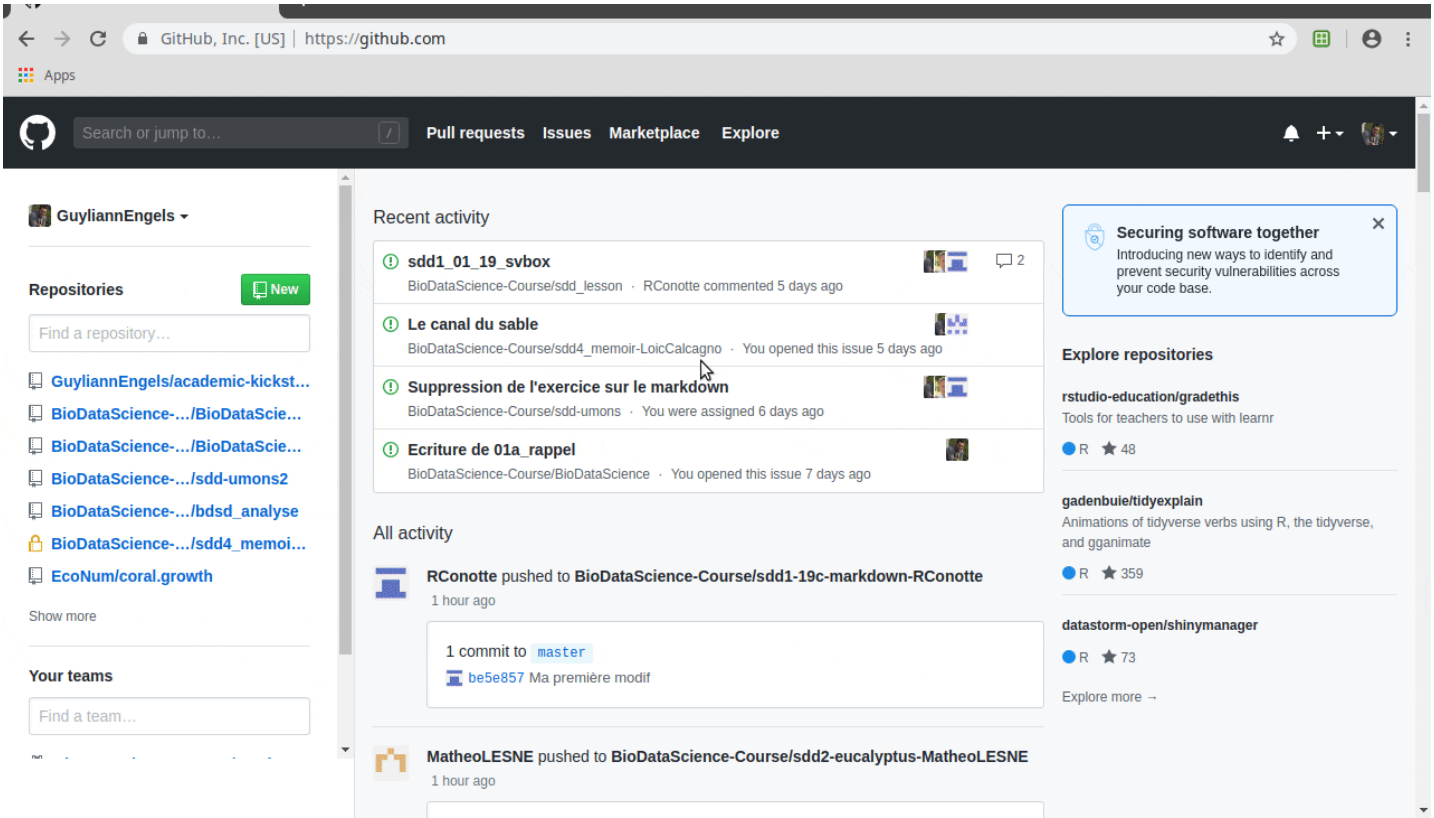
Rendez-vous dans un second temps dans GitHub.com où vous vous identifiez et vous allez dans les paramètres de votre compte. Pour cela, vous cliquez sur votre avatar en haut à droite de la page du site et vous sélectionnez **Settings** dans le menu déroulant. Vous allez dans la section **SSH and GPG keys** à gauche. Ensuite, dans **SSH keys** vous cliquez sur le bouton **New SSH key**. Vous indiquez un titre qui ne soit pas déjà utilisé si vous avez déjà d'autres clés enregistrées (nous vous conseillons `SaturnCloud`). Enfin, vous collez la clé publique depuis le presse-papier dans le champ **Key**. Vous terminez l'opération en cliquant sur le bouton **Add SSH key**. En principe, la communication entre RStudio et

GitHub est effective à présent. Cette clé n'expire pas et ne doit pas être recréée. Mais si vous suspectez un problème, vous pouvez toujours en recréer un autre dans SaturnCloud. Ensuite, dans GitHub vous effacez l'ancienne et ajouter la nouvelle de la même façon.

> Que ce soit via un PAT ou via la clé SSH, RStudio ne doit vous demander aucun mot de passe et ne doit pas afficher de message d'erreur de connexion lors d'opérations entre Rstudio et GitHub (clone, push, pull…). Lorsque vous clonerez vos dépôts depuis GitHub, vous devrez par contre penser à utiliser la référence **HTTPS** si vous utilisez un PAT, et la référence **SSH** si vous utilisez une clé SSH. **Si vous inversez les références, cela ne fonctionnera pas !**

B.2.4 Créer un dépôt

Lorsque nous souhaitons créer un nouveau dépôt GitHub, nous devons l'initialiser comme suit :



Pour créer un nouveau dépôt (`Create a new repository`), nous devons fournir les informations suivantes :

- `Repository template`

Nous devons décider d'utiliser ou non un "template" existant parmi la liste des templates que nous avons à disposition.

- `Owner`

Nous devons décider du responsable du dépôt, soit une organisation, soit une personne.

- `Repository name`

Nous devons choisir un nom court, mais pertinent pour notre dépôt et qui ne soit pas déjà utilisé dans l'organisation ou l'espace de l'utilisateur sélectionné.

- `Description`

Nous pouvons indiquer ici une courte description de notre dépôt.

- `Public` ou `Private`

Nous devons décider si notre projet est public ou privé.

- `README`

Nous pouvons éditer un fichier de présentation qui se nomme `README` . Ce dernier va présenter succinctement notre projet. On peut l'éditer depuis GitHub directement. Si ce fichier est au format Markdown (nettement conseillé), il se nommera alors `README.md` . Ainsi, vous pourrez formater les titres et le texte dans ce fichier.

- `.gitignore`

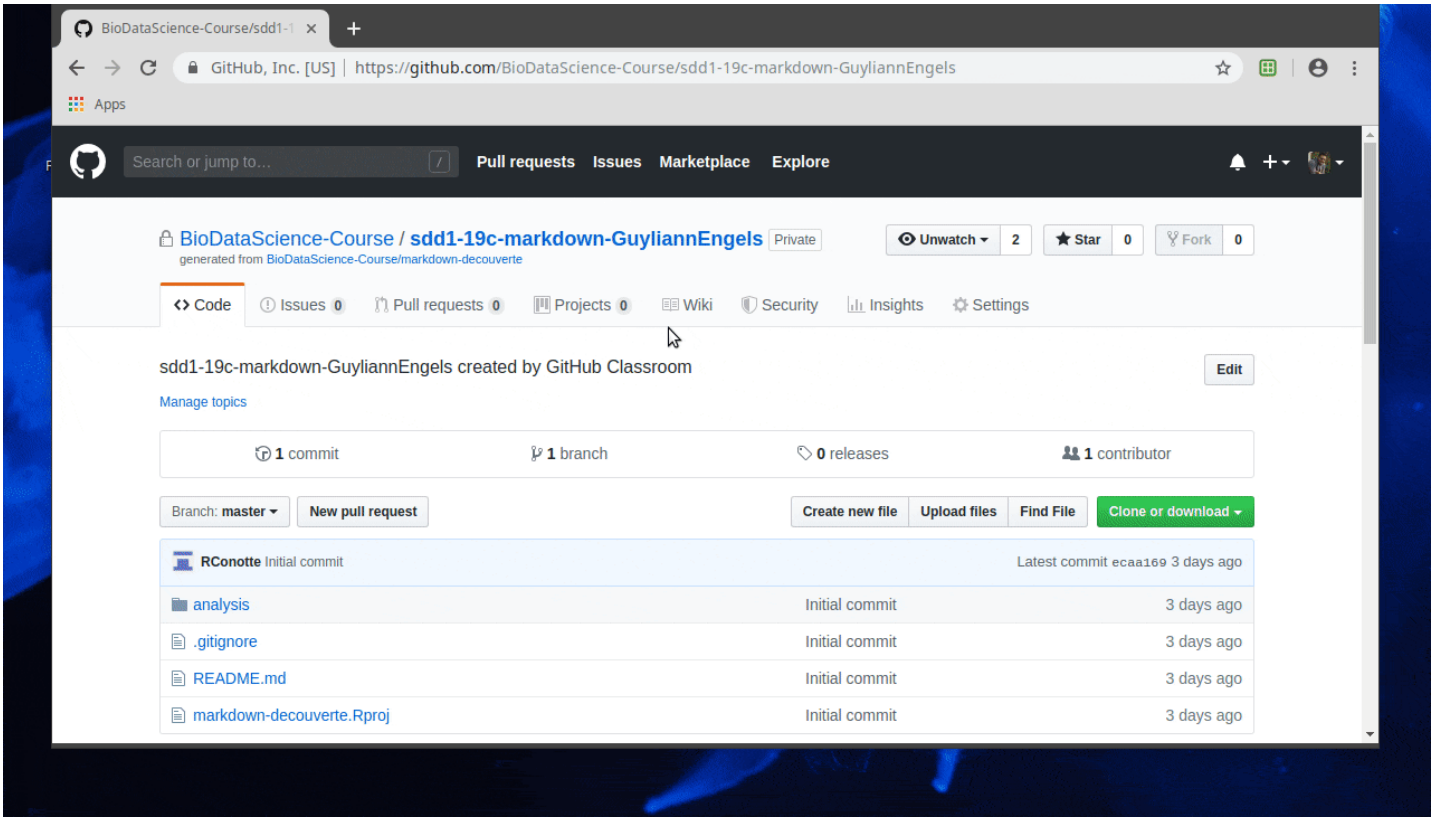
Il est intéressant de configurer le dépôt avec un fichier `.gitignore` orienté sur l'utilisation de **R**. GitHub peut en effet héberger des projets dans des langages informatiques très variés.

- `license`

Nous pouvons adjoindre à notre projet une licence. Il en existe plusieurs qui définissent précisément ce que l'on a le droit de faire ou non avec votre dépôt. Le site https://choosealicense.com peut vous aider à choisir la licence la plus appropriée en fonction du contenu que vous souhaitez y héberger. Une fois votre dépôt configuré, il ne vous reste plus qu'à le **cloner** comme expliqué dans la section B.2.5.

B.2.5 Cloner un dépôt existant via RStudio

Lorsque nous souhaitons travailler sur un de nos projets hébergés dans un dépôt GitHub, il faut commencer par le cloner pour avoir une copie en local de ce dernier. Notez que "local" signifie ici dans la machine sur laquelle nous travaillons. Si nous utilisons une ressource SaturnCloud ou tout autre service dans le Cloud comme Posit Cloud par exemple, le contenu du dépôt sera recopié dans cette machine-là dans le Cloud. Notre contenu sera donc "local" de manière relative au logiciel qui s'y exécute.



Pour commencer, vous devez copier le lien menant à votre dépôt GitHub. Il vous suffit de cliquer sur le bouton vert `Code` dans la section du même nom et de copier l'URL proposée dans le presse-papier (vous avez d'ailleurs un bouton avec une icône suggérant la copie pour le faire). **Mais attention, avant de faire cela, il faut déterminer quel format**

doit être copié. Si vous vous identifiez via un PAT, il faut récupérer la version **HTTPS**. Si vous utilisez une clé SSH, vous devez utiliser la version **SSH**. Donc pour simplifier, si vous utilisez la SciViews Box dans VirtualBox choisissez **HTTPS**. Si au contraire, vous utilisez la SciViews Box dans SaturnCloud, choisissez la version **SSH**.

Ensuite, vous devez vous rendre dans RStudio et cliquer sur `Project || Projet` en haut à droite, suivi de `New Project... || Nouveau projet...` (Si les projets restent encore un peu flous pour vous, rendez-vous dans la section @ref(#rs-projet)). Une nouvelle fenêtre s'ouvre. Vous devez sélectionner `Version Control || Contrôle de version`, puis `Git` dans les préférences successives.

Pour finir, vous devez :

- renseigner l'URL précédemment copiée depuis GitHub,
- choisir un nom à votre dépôt (une bonne pratique est de nommer votre projet du même nom que votre dépôt, sans l'extension `.git` à la fin),
- choisir un dossier pour cloner votre dépôt (le sous-dossier `projects`, du dossier `shared` est dédié à cela en version VirtualBox, ou le dossier `workspace` dans la version SaturnCloud) et
- créer une copie en local de votre projet en cliquant sur `Create Project || Créer un projet`.

Vous êtes à présent prêt à éditer votre projet. Vous réaliserez des **commits** (enregistrement d'une version de votre dépôt), des **pulls** (récupération en local des modifications de votre dépôt enregistrées dans GitHub) et des **pushes** (envoi de vos modifications locales dans le dépôt GitHub) pour enregistrer des versions successives de votre travail et pour les synchroniser avec le dépôt GitHub. Vous apprenez à faire cela dans le module 1 du cours.

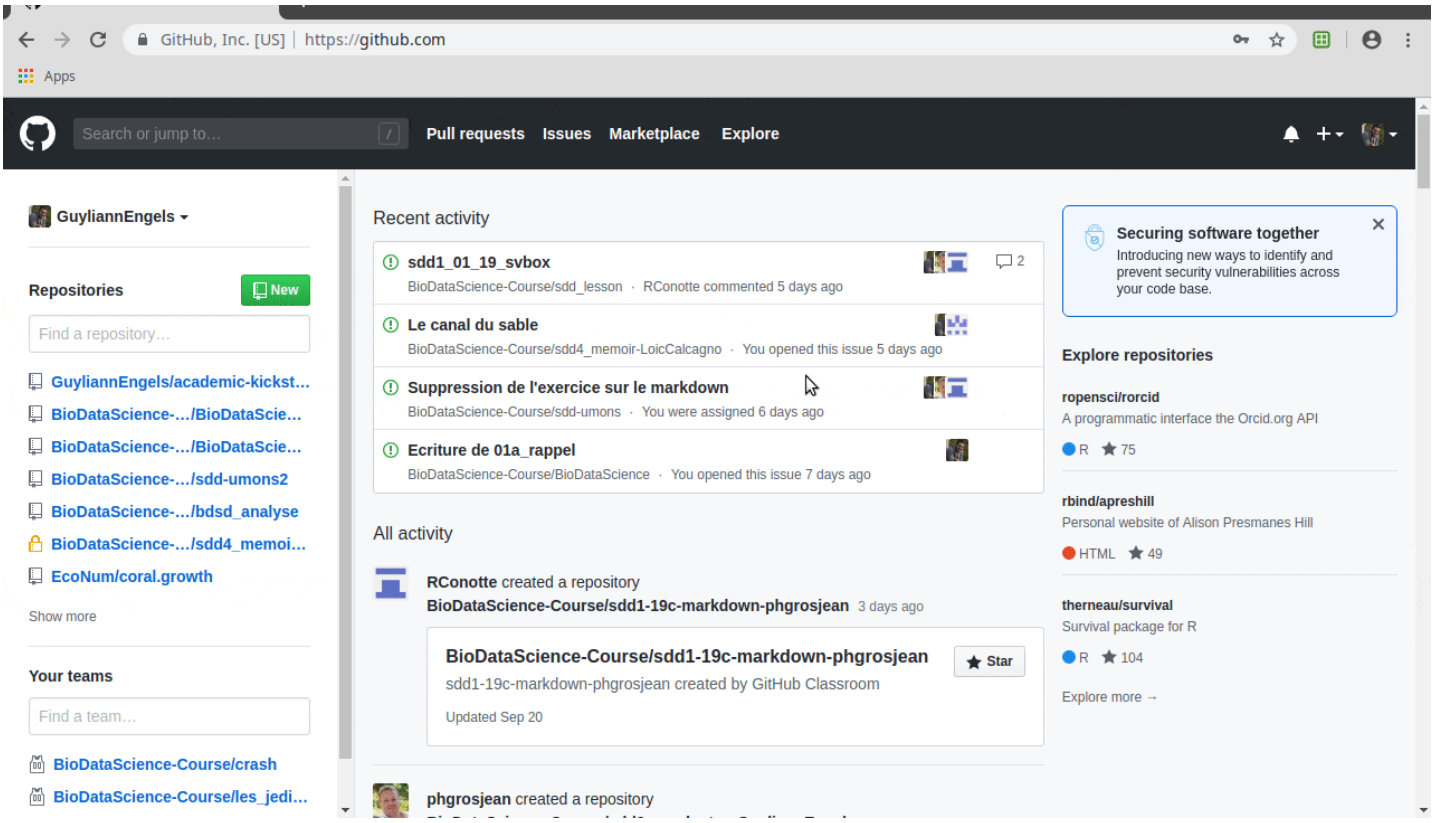
B.2.6 Déposer un projet déjà créé

Nous avons créé un projet dans RStudio et l'avons configuré avec le gestionnaire de version Git comme présenté dans l'annexe B.1.1.1. Cependant, après avoir progressé dans ce projet (et réalisé plusieurs **commits**), nous souhaitons le partager à présent via

GitHub. Rassurez-vous, il ne faut pas tout recommencer. Il aurait été plus simple de réfléchir dès le début du projet à cette éventualité et de créer le dépôt GitHub en premier lieu, mais cela reste parfaitement faisable à ce stade de transformer un projet local RStudio en un dépôt GitHub.

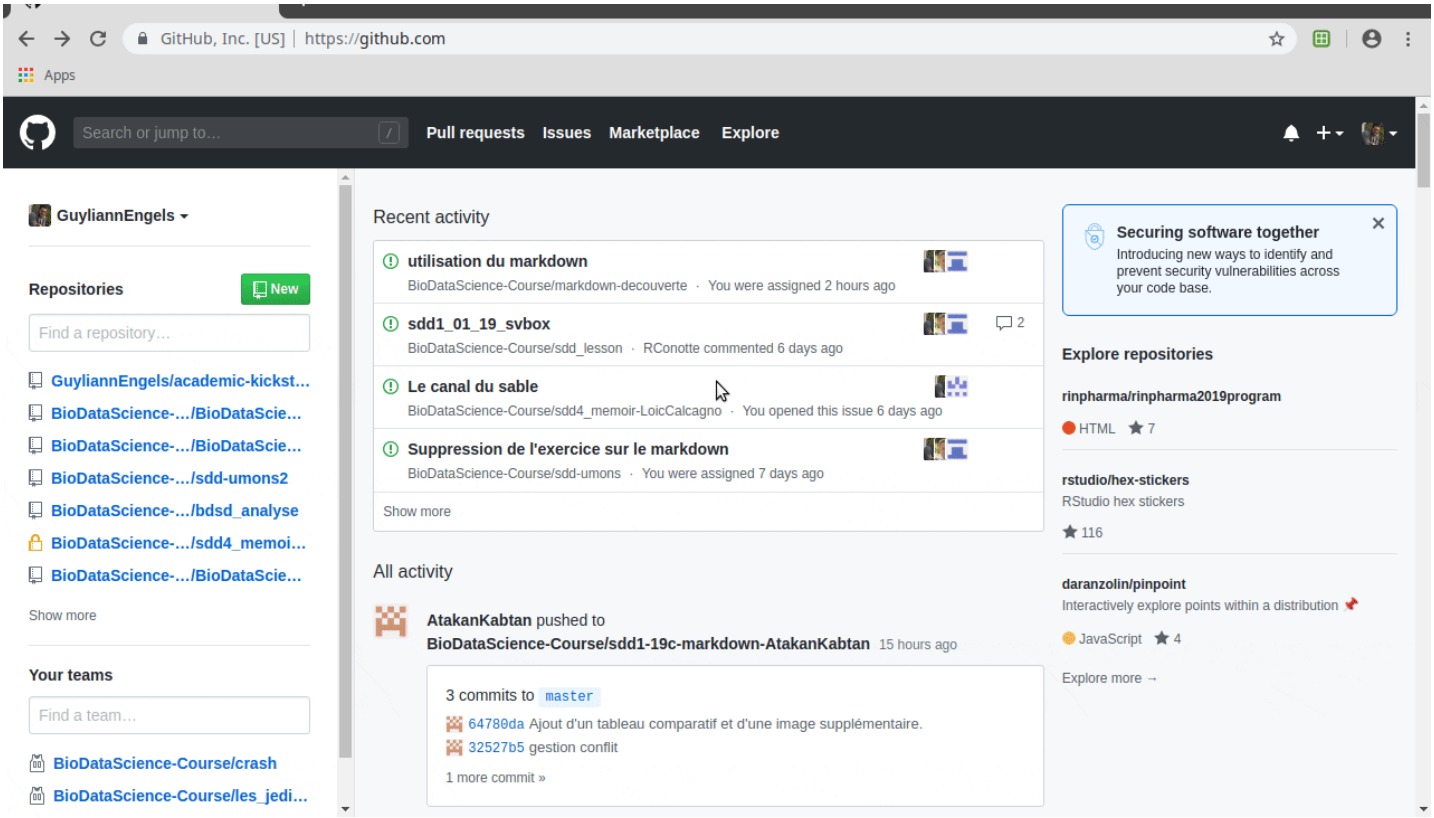
> Une bonne pratique avant de vous lancer dans un nouveau projet et de se poser et de réfléchir aux objectifs du projet et aux moyens à mettre en œuvre pour atteindre ces objectifs.

Pour les explications, nous partons d'un projet RStudio d'exemple fictif qui se nomme `repos-example`. Comme vous pouvez le voir, trois **commits** ont déjà été réalisés dans ce projet, mais nous ne pouvons pas faire de **pull** ou **push**, puisque notre projet n'est pas lié à un dépôt distant GitHub ou autre.

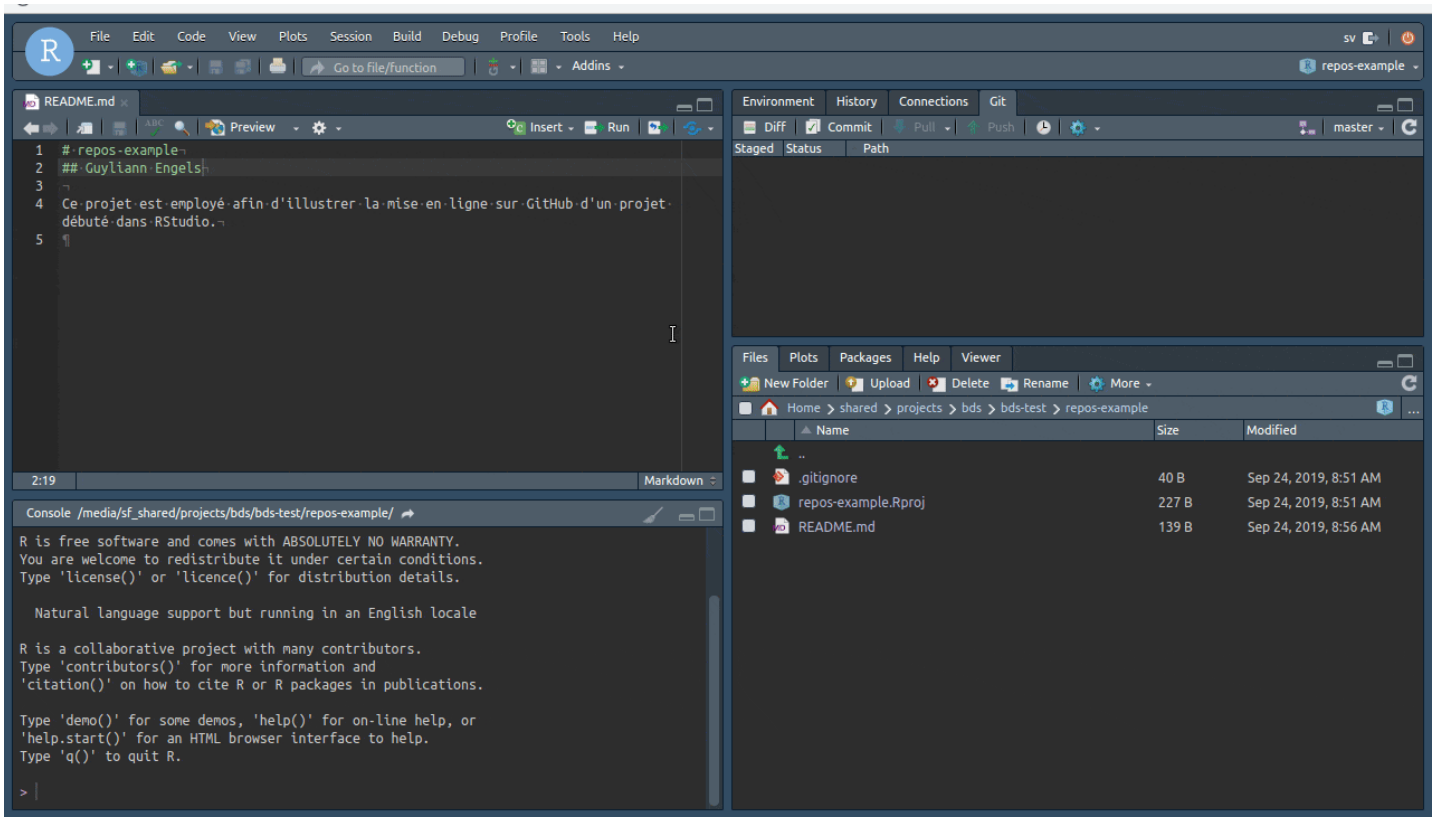


Pour déposer un projet RStudio existant dans GitHub, vous devez commencer par créer un nouveau dépôt dans Github qui ressemble très fortement à l'annexe B.2.4. Avec une particularité que vous ne devez pas configurer le `README`, le `.gitignore` et la `license`. Comme le dépôt est vide, GitHub vous propose plusieurs options pour le

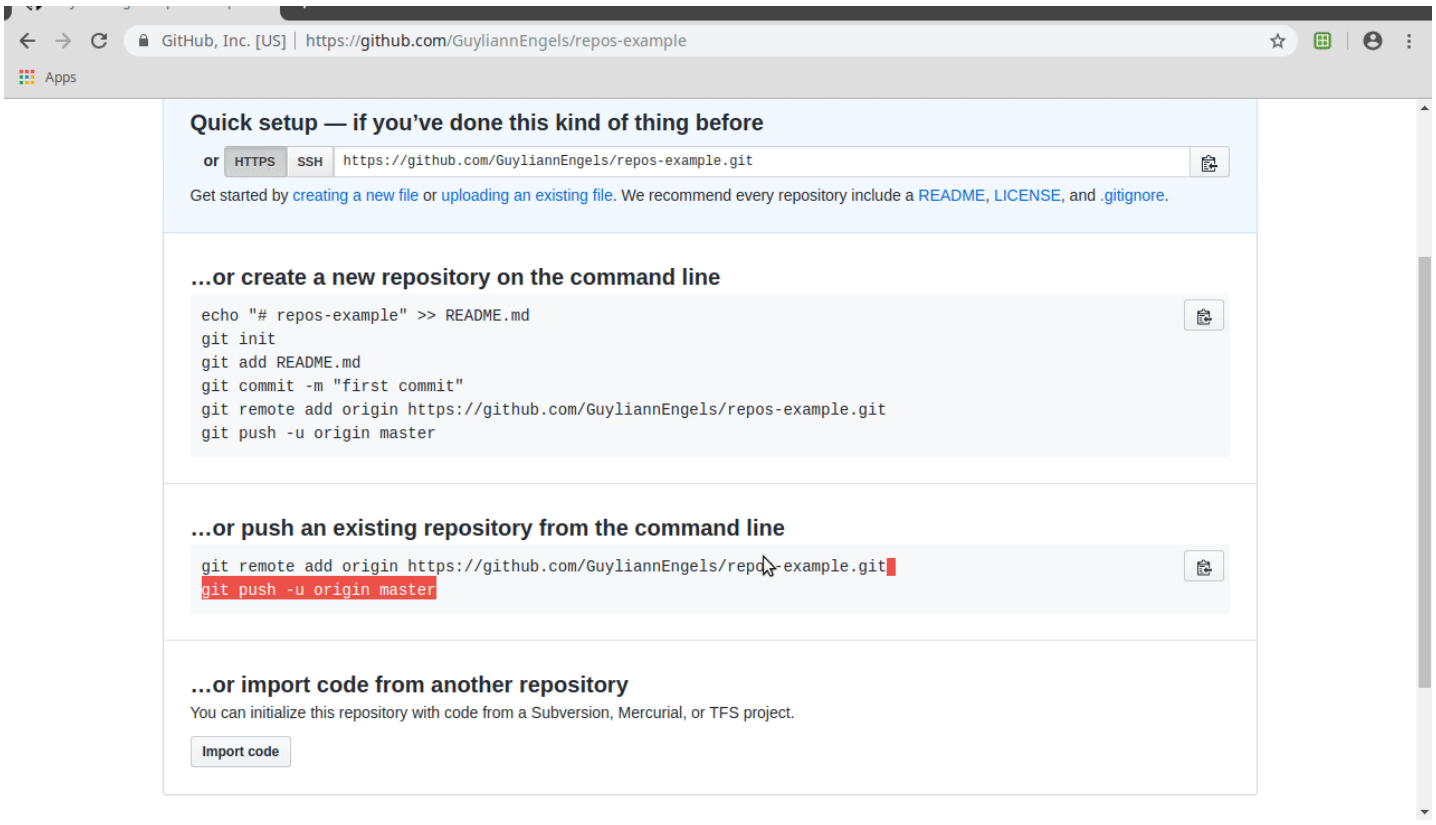
remplir, dont `…or push an existing repository from the command line` . Il s'agit donc de mettre en ligne (**push**) un projet existant sous forme de dépôt local Git. Copier les trois lignes de code proposées à cet endroit dans le presse-papier.



Dans votre projet RStudio, sélectionnez le menu `Tools || Plus` puis `Shell... || Nouveau terminal` . Un onglet **Terminal** vient de s'ouvrir à côté de l'onglet **Console** en bas à gauche. Il vous suffit ensuite d'y coller les trois instructions qui se trouvent dans le presse-papier et de taper sur la touche `Entrée` pour transformer votre projet git en un dépôt GitHub.



Pour vérifier que votre projet RStudio est correctement mis en ligne dans GitHub, vous pouvez recharger votre page dans https://github.com.



B.2.7 Copier un dépôt

Lorsque vous souhaitez apporter votre aide à un projet qui n'est pas le vôtre, vous devez réaliser un **fork** puis un **clone**. N'oubliez pas que la base de GitHub est de faciliter la collaboration. Pour soumettre vos modifications au projet de départ, il faut faire un **pull request**. Cette étape permet de proposer vos modifications au responsable du projet original sous une forme qui lui permet de visualiser et de discuter les changements proposés. Il pourra alors, en connaissance de cause, accepter ou refuser vos modifications.

Appendice C Tutoriels “learnr”

En complément de ce cours en ligne, vous allez utiliser également des tutoriels interactifs construits avec learnr. Si ce n’est déjà fait, commencez par installer ces tutoriels dans RStudio.

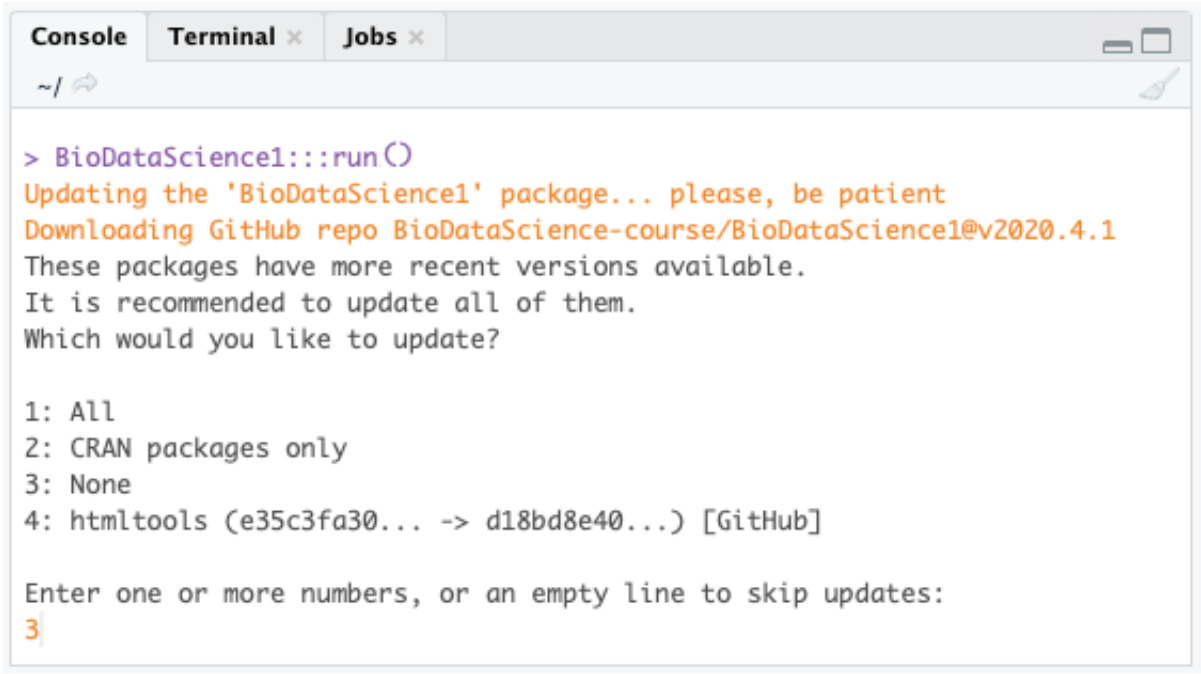
Allez dans le bouton **Addins** au milieu de la barre d’outils générale de RStudio et sélectionnez `Run SDD I tutorial or app` dans la section **BioDataScience1**, ou entrez l’instruction ci-dessous dans la fenêtre **Console** de RStudio suivie de la touche `Entrée` :

```
BioDataScience1::run()
```

Si la section n’apparaît pas dans les addins, ou si l’instruction génère une erreur qui indique qu’il n’y a pas de package BioDataScience1, vous devez installer les deux packages {BioDataScience} et {BioDataScience1} du cours[70]. **Cela ne doit pas se produire si vous êtes étudiant·e UMONS car ces packages sont déjà installés dans votre SciViews Box et vous ne devez donc pas exécuter le code juste ci-dessous !** Dans la Console, entrez les lignes de code suivantes pour effectuer cette installation :

```
remotes::install_github("BioDataScience-Course/BioDataScience")
remotes::install_github("BioDataScience-Course/BioDataScience1")
```

Un fois l’addin lancé, si une mise à jour du package est disponible, elle sera installée. Si d’autres packages R peuvent également être mis à jour, cela vous sera demandé. Dans ce cas, **refusez la mise à jour des autres packages R** en indiquant `3` puis la touche `Entrée` pour l’option `None` (à part le package {BioDataScience1} nous voulons conserver les versions de tous les autres packages R identiques à la version installée avec la SciViews Box par souci de compatibilité et reproductibilité de nos analyses).



Après la mise à jour éventuelle du package, la liste des tutoriels vous est proposée :

```
1: A00La_discovery
2: A01La_tools
3: A01Lb_base
...
```

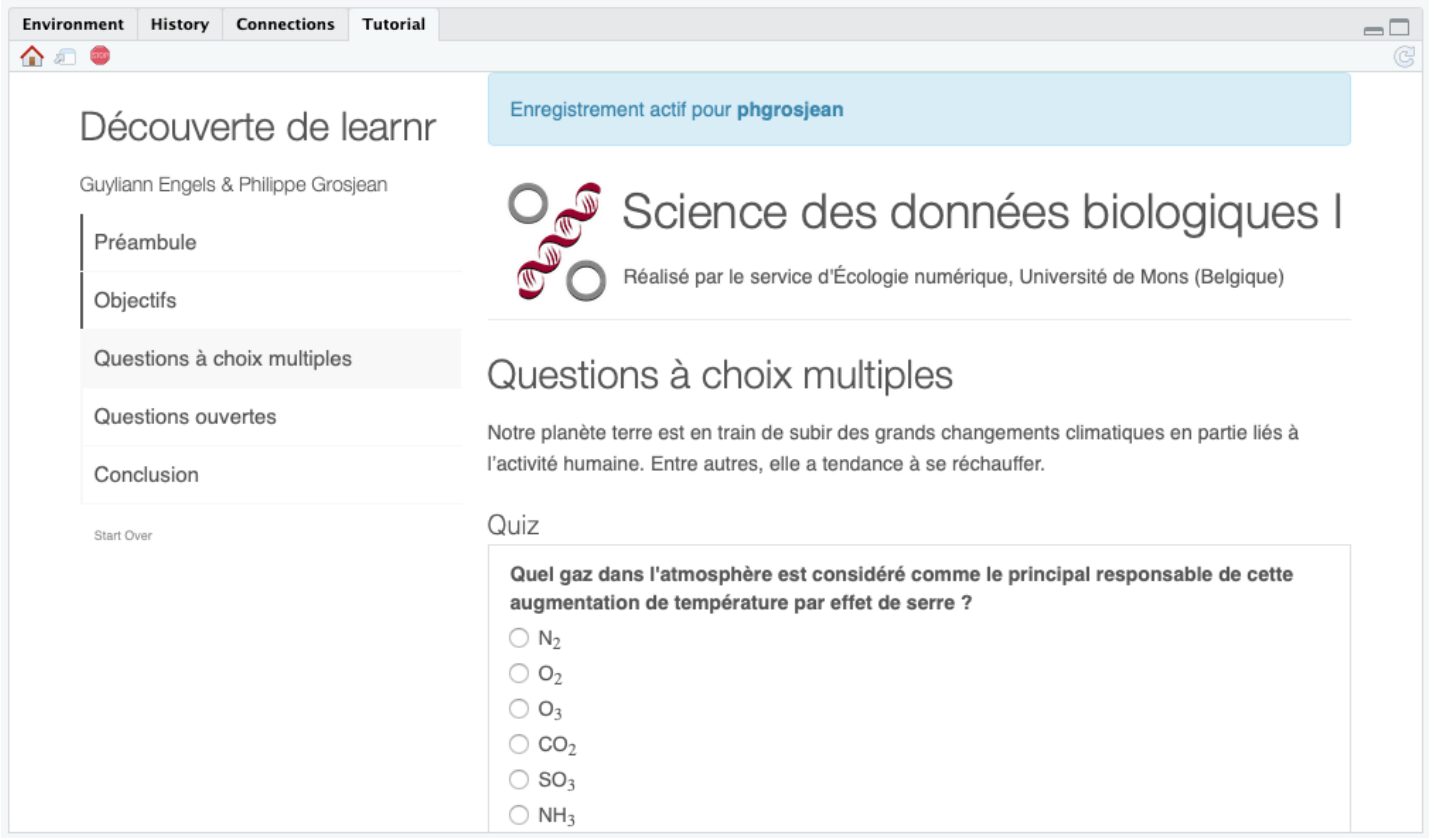
Sélectionnez le tutoriel désiré dans la boite de dialogue (si lancé à partir de l’addin) ou entrez le numéro correspondant au tutoriel que vous voulez exécuter dans la console. Un document learnr interactif apparaît. Par exemple, en entrant `1` suivi de la touche `Entrée`, vous êtes redirigé vers le tutoriel concernant la découverte de R et intitulé `A00La_discovery`.

> L’onglet **Tutorial || Tutoriel** donne également accès aux différents tutoriels, mais la liste n’est **pas** mise-à-jour automatiquement. **Vous devez donc vous assurer que cette liste correspond bien aux versions les plus récentes des exercices en ayant exécuté au moins une fois `BioDataScience1::run()` au cours de la session actuelle.** Ensuite, si une mise-à-jour a été installée, il faut

aussi rafraîchir la liste des tutoriels présentés dans l’onglet **Tutorial || Tutoriel** en allant dans le menu de RStudio `Session -> Restart R || Session -> Redémarrer R .`

Dans certains cas, le lancement des tutoriels à partir de l’addin ou à partir de `BioDataScience1::run()` lance la compilation du tutoriel dans l’onglet **Jobs || Travaux**, mais ce tutoriel ne s’affiche *pas* dans **Tutorial || Tutoriel** ensuite. C’est un bug dans RStudio. Si c’est le cas, vous pouvez alors relancer le tutoriel à partir de l’onglet **Tutorial || Tutoriel** en cliquant sur le bouton **Start Tutorial**, et normalement, il doit apparaître maintenant.

La première chose à vérifier à l’ouverture du tutoriel interactif est le nom d’utilisateur (équivalent à votre **username** dans Github) qui sera utilisé pour l’enregistrement de votre activité si vous êtes étudiant UMONS. En effet, votre progression devra être enregistrée, mais cela ne peut se faire que si vous renseignez ces données correctement.



Si vous êtes correctement enregistré, un bandeau bleu l’indiquera avec votre login GitHub en gras comme ci-dessus. Dans le cas contraire, un message sur fond jaune ou rouge (selon le contexte) indiquera le contraire. De même, si vous voyez également un message

sur fond rouge qui indique que la base de données est inaccessible, cela signifie que vous n’êtes pas correctement connecté à l’Internet ou que votre connexion Internet ne permet *pas* d’accéder à la base de données qui enregistre votre activité. Essayez un autre réseau, ou même, essayez en redémarrant complètement votre ordinateur (ou la machine virtuelle sur le cloud). Si la base de données est actuellement inaccessible, vous pouvez toujours effectuer les exercices, mais votre activité sera temporairement enregistrée localement dans votre ordinateur. Dès la prochaine connexion fructueuse, et dès la soumission d’un exercice learnr, l’ensemble de votre activité sera alors enregistrée d’un coup dans la base de données.

70. Pour le cours 2, vous installerez aussi le package {BioDataScience2} et pour le cours 3 ou 4, {BioDataScience3} selon la même technique.↩

C.1 Organisation d'un learnr

Le learnr est un outil pédagogique mis au point pour proposer des tutoriels interactifs comprenant des illustrations, des questions à choix multiples, des exercices R… Les learn qui vous seront proposés tout au long de votre formation en Science des Données biologiques seront composés de la manière suivante :

- Objectifs
- Introduction
- Une série d'exercices
- Conclusion

Vous retrouvez d'ailleurs cette structure en haut à gauche dans la page de celui-ci. Chaque page du tutoriel est importante et nécessite votre attention.

C.1.1 Objectifs

Cette section détaille l'ensemble des notions que vous allez apprendre à maîtriser dans le tutoriel. Par exemple, pour le premier tutoriel, l'objectif est de découvrir comment fonctionne un "learnr" à l'aide de différentes questions de culture générale.

C.1.2 Introduction

Cette section facultative vous place dans le contexte du tutoriel interactif avec un rappel succinct des notions théoriques indispensables pour répondre à la série d'exercices, et éventuellement aussi, le contexte du jeu de données traité. **Cette section ne remplace pas les autres matériels pédagogiques qui vous sont proposés.** Vous devez donc travailler et étudier la section du cours correspondant *avant* de vous lancer dans le tutoriel dans un second temps… sinon, il ne servira à rien.

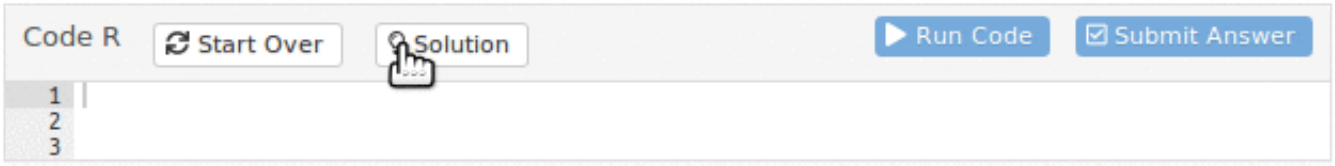
C.1.3 Divers exercices

Cette ou ces sections peuvent être de longueur variable en fonction de la difficulté et des notions à appréhender. Les questions sont présentées sous forme de quiz (questions fermées) ou de zones de code R (questions ouvertes).

Pour les quiz, vous devez cliquer sur le bouton bleu `Submit Answer`. En cas d'erreur, vous avez la possibilité de réessayer dans les questions formatives (mais pas dans les questions d'évaluation). Pour les étudiants de l'UMONS, les points sont calculés en fonction du nombre d'essais nécessaires pour obtenir la bonne réponse (10% déduits par essai infructueux, ce qui est finalement une pénalité très faible qui vous permet plusieurs essais incorrects, tout en conservant une excellente note à l'exercice… mais n'en abusez pas tout de même).

Des zones de code R vous sont proposées dans certains exercices. Elles vous permettent d'expérimenter directement des instructions dans R depuis le document learnr. Pour exécuter ces instructions, il faut cliquer sur le bouton `Run Code`. **Vous pouvez le faire autant de fois que vous le voulez, et ce, sans perte de points.** Modifiez le code, cliquez `Run Code`, analysez le résultat, modifiez votre code, recliquez `Run Code`, etc. jusqu'à ce que vous soyez satisfait du résultat. Finissez l'exercice et soumettez votre réponse en cliquant sur le bouton `Submit Answer`. Tout comme pour les quiz, 10% sont déduits à chaque soumission de réponse incorrecte. Voici un exemple de question nécessitant l'écriture de code R :

- Multipliez les nombres `15` et `23` (un encadré **Code R** est une zone où vous pouvez vous-même entrer des instructions R et/ou les modifier. Les numéros à gauche sont les numéros de lignes. Ils ne font pas partie des instructions. Utilisez le bouton **Run Code** pour tester, et ensuite **Submit Answer** quand vous êtes satisfait de ce votre réponse)



Vous pouvez également utiliser des fonctions mathématiques directement implementées dans R comme le logarithme népérien avec la **fonction** `log()`. Ce language a été mis au point pour suivre les conventions des mathématiques et des statistiques comme le respect de l'ordre des opérations.

Selon le cas, vous pouvez alors avoir un encart bleu qui indique simplement que votre réponse est enregistrée, ou alors, si l'exercice est assorti d'une correction automatique, vous verrez un encart vert en cas de bonne réponse ou un encart rouge qui vous donne un indice sur ce qui est incorrect (**attention : ces messages d'aide sont élaborés automatiquement par l'ordinateur et ne sont pas toujours totalement fiables ; concentrez-vous surtout sur le fait que la réponse n'est pas celle attendue et tentez de déterminer ce qui ne va pas**).

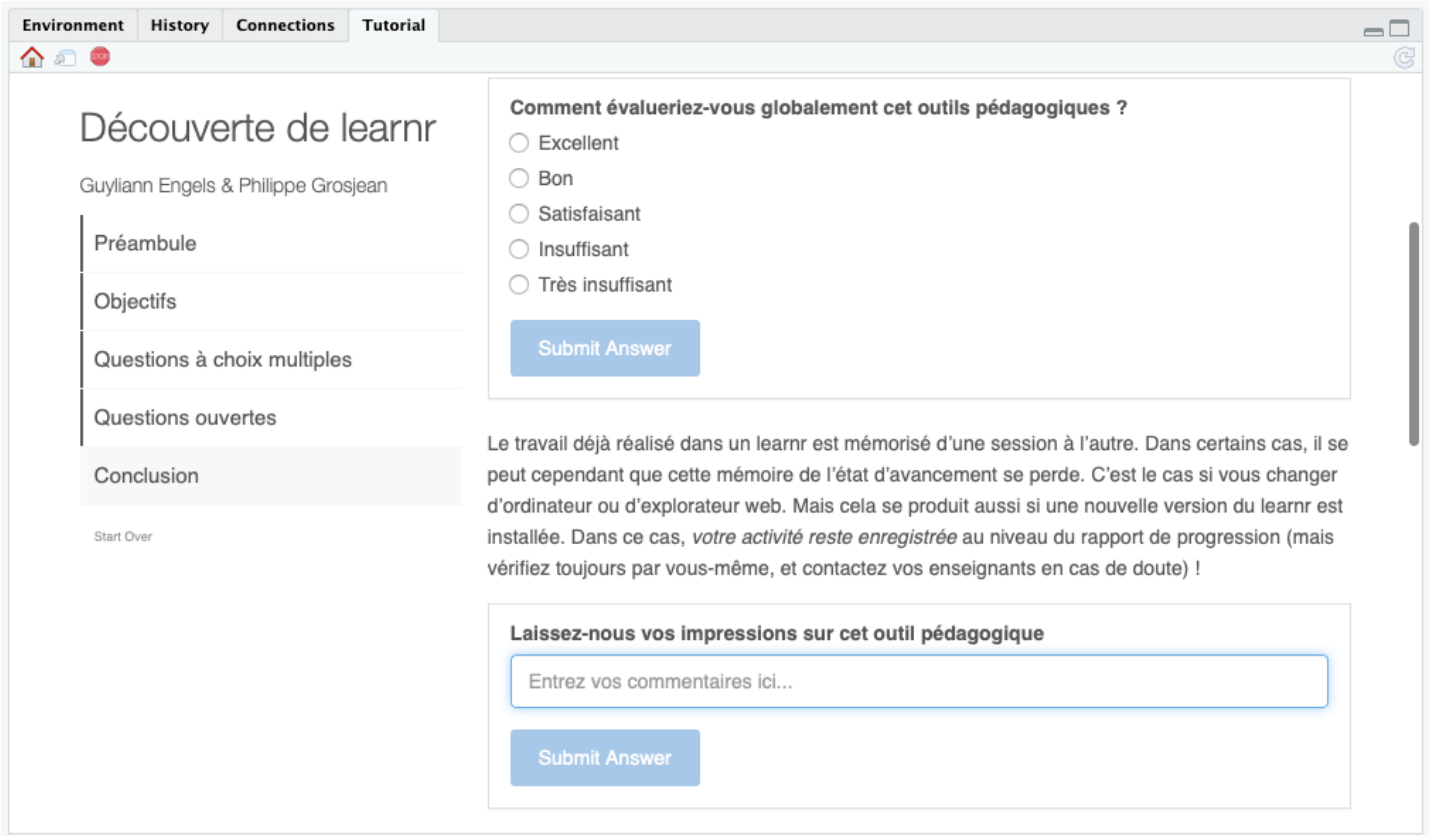
Un bouton `Hint`, lorsqu'il est présent, vous propose une aide si vous êtes bloqué sur la question. Un bouton `Solution` ou le dernier `Hint` affiché après celui qui indique que le suivant sera la solution à la question… montrent ce qu'il fallait entrer. **N'allez pas voir directement la solution. Essayez d'abord par vous-même !** Ces aides supplémentaires sont, en effet, à utiliser en dernier recours seulement si vous êtes bloqué, moyennant une pénalité de 10% à chaque "hint" vu (et si vous avez lu la solution, vous n'aurez plus que la moitié des points pour cet exercice).

Quelle que soit la façon dont vous répondez à une question, vous obtenez donc au moins la moitié des points. Si vous avez été voir la solution avant de chercher la réponse par vous-même, nous considérons que c'est parce que vous n'arrivez pas à résoudre cet exercice seul. Dans le cadre d'une auto-évaluation, nous vous faisons confiance pour

analyser la réponse et essayer de comprendre où sont vos lacunes pour progresser. *Si vous ne le faites pas, vous n'apprendrez rien, et vous serez incapables de résoudre les exercices dans les projets GitHub ensuite.* Au moins vous sollicitez les aides pour obtenir la réponse, au plus vous obtiendrez une bonne note pour cet exercice. Mais ne perdez pas de temps sur les exercices qui vous semblent difficiles et utilisez les `Hint`s à bon escient.

C.1.4 Conclusion

Cette section termine le tutoriel et propose de laisser, de manière optionnelle, un commentaire (soyez constructifs, si vous pensez que le tutoriel pourrait être amélioré d'une quelconque façon : nous tiendrons compte de vos remarques pour les versions des années suivantes). Fermez le tutoriel lorsqu'il est terminé en cliquant sur le bouton rouge en forme de panneau stop dans la barre d'outils de l'onglet **Tutorial || Tutoriel**.



Appendice D Rédaction scientifique

La rédaction de textes scientifiques doit respecter un certain canevas et différentes règles qui sont résumés dans cette annexe.

Pour en savoir plus

- Pour les étudiants de l'UMONS, vous avez accès à Recherche documentaire et aide à la création (ReDAC), un cours en ligne qui rassemble un maximum de renseignements sur la rédaction de rapports scientifiques.

D.1 Organisation

Un rapport scientifique respecte généralement le schéma suivant :

1. Table des matières (facultatif)
2. Introduction
3. But
4. Matériel et méthodes
5. Résultats
6. Discussion
7. Conclusion
8. Bibliographie
9. Annexe(s) (si nécessaire)

Pour des travaux de plus grande ampleur comme les travaux de fin d'études, le schéma ci dessus est adapté, et éventuellement divisé en chapitres, en y ajoutant généralement une partie remerciements en début de manuscrit, ainsi qu'une liste des figures, des tables, des abréviations utilisées, voire un index en fin d'ouvrage.

D.2 Contenu

Le rapport sert à restituer de façon synthétique les résultats d'une étude scientifique et les interprétations. Le tout est remis dans le contexte de la bibliographie existante en la synthétisant dans l'introduction et en comparant les résultats avec d'autres études connexes dans la discussion. Il faut garder à l'esprit qu'un lecteur doit comprendre l'intégralité du rapport avec un minimum de connaissances *a priori* sur l'étude réalisée, mais avec des connaissances générales dans la spécialité. Donc, un rapport sur un sujet biologique est adressé à un lecteur biologiste pour lequel il ne faut pas rappeler les concepts de base dans sa discipline. Par contre, il faut expliquer avec suffisamment de détails comment l'étude a été réalisée dans la section "matériel et méthodes".

En général, les phrases sont simples, directes, courtes et précises (veillez à utiliser le vocabulaire adéquat et précis). Les explications sont, autant que possible, linéaires. Évitez les renvois dans différentes autres parties du rapport, si ce n'est pour rappeler un élément évoqué plus haut, ou pour se référer à une figure ou une table. À ce sujet, les figures (dont les images, photos, schémas et graphiques) sont numérotées (Figure 1, Figure 2, …) et accompagnées d'une légende en dessous d'elles. La figure et sa légende doivent être compréhensibles telles quelles. Dans le texte, vous pourrez alors vous référer à la figure, par exemple : "Tel phénomène est observable (voir Fig. 3)", ou "La Figure 4 montre …". Idem pour les tableaux qui sont également numérotés (Table 1, Table 2, …) et légendés, mais *au dessus* du tableau. Les règles de lisibilité du tableau + légende et de renvoi vers les tableaux sont identiques que pour les figures. Les équations peuvent aussi être numérotées et des renvois de type (eq. 5) peuvent être alors utilisés. Enfin, toute affirmation doit être soit démontrée dans le rapport, soit complétée d'une citation vers un autre document scientifique qui la démontre. La partie bibliographie regroupe la liste de tous les documents qui sont ainsi cités à la fin du rapport.

Veillez à respecter les notations propres au système métrique international, les abréviations usuelles dans la discipline, et le droit d'auteur et les licences si vous voulez citer un passage ou reprendre une illustration provenant d'un autre auteur (sans omettre d'indiquer qui en est l'auteur). Enfin, en vue de rendre le document parfaitement reproductible, vous pouvez indiquer dans les annexes où trouver la source de votre rapport (le document `.Rmd`) et les données analysées. Vous pouvez également terminer avec un "chunk" qui renseigne de l'état du système R utilisé, y compris l'ensemble des packages employés. Ce chunk, présenté en annexe, contiendra l'instruction `utils::sessionInfo()`, ou mieux : `sessioninfo::session_info()`. Cela donnera quelque chose du genre :

```
sessioninfo::session_info()
```

```
─ Session info ─────────────────────────────────────────────────────────────
 setting  value
 version  R version 4.4.3 (2025-02-28)
 os       macOS Sequoia 15.6.1
 system   aarch64, darwin20
 ui       X11
 language (EN)
 collate  en_US.UTF-8
 ctype    en_US.UTF-8
 tz       Europe/Brussels
 date     2025-12-06
 pandoc   3.7.0.2 @ /opt/homebrew/bin/ (via rmarkdown)
 quarto   1.5.45 @ /usr/local/bin/quarto

─ Packages ─────────────────────────────────────────────────────────────────
 package           * version date (UTC) lib source
 abind               1.4-8   2024-09-12 [2] RSPM (R 4.4.0)
 anytime             0.3.11  2024-12-19 [2] RSPM (R 4.4.0)
 askpass             1.2.1   2024-10-04 [2] RSPM (R 4.4.0)
 assertthat          0.2.1   2019-03-21 [2] RSPM (R 4.4.0)
```

```
backports           1.5.0   2024-05-23 [2] RSPM (R 4.4.0)
base64enc           0.1-3   2015-07-28 [2] RSPM (R 4.4.0)
bit                 4.6.0   2025-03-06 [2] RSPM (R 4.4.0)
bit64               4.6.0-1 2025-01-16 [2] RSPM (R 4.4.0)
bookdown            0.42    2025-01-07 [2] RSPM (R 4.4.0)
broom             * 1.0.8   2025-03-28 [2] RSPM (R 4.4.0)
bslib               0.9.0   2025-01-30 [2] RSPM (R 4.4.0)
cachem              1.1.0   2024-05-16 [2] RSPM (R 4.4.0)
car                 3.1-3   2024-09-27 [2] RSPM (R 4.4.0)
carData             3.0-5   2022-01-06 [2] RSPM (R 4.4.0)
chart             * 1.5.4   2025-08-29 [2] https://sciviews.r-universe.dev (R
checkmate           2.3.2   2024-07-29 [2] RSPM (R 4.4.0)
cli                 3.6.4   2025-02-13 [2] RSPM (R 4.4.0)
cluster             2.1.8.1 2025-03-12 [3] RSPM (R 4.4.0)
codetools           0.2-20  2024-03-31 [3] CRAN (R 4.4.3)
collapse          * 2.1.0   2025-03-10 [2] RSPM (R 4.4.3)
colorspace          2.1-1   2024-07-26 [2] RSPM (R 4.4.0)
cowplot             1.1.3   2024-01-22 [2] RSPM (R 4.4.0)
crayon              1.5.3   2024-06-20 [2] RSPM (R 4.4.0)
curl                6.2.2   2025-03-24 [2] RSPM (R 4.4.0)
data.io           * 1.7.0   2025-08-29 [2] https://sciviews.r-universe.dev (R
data.table          1.17.0  2025-02-22 [2] RSPM (R 4.4.3)
data.trame        * 0.9.0   2025-08-29 [2] https://sciviews.r-universe.dev (R
deldir              2.0-4   2024-02-28 [2] RSPM (R 4.4.0)
digest              0.6.37  2024-08-19 [2] RSPM (R 4.4.0)
dplyr             * 1.1.4   2023-11-17 [2] RSPM (R 4.4.0)
dtplyr            * 1.3.1   2023-03-22 [2] RSPM (R 4.4.0)
ellipse             0.5.0   2023-07-20 [2] RSPM (R 4.4.0)
ellipsis            0.3.2   2021-04-29 [2] RSPM (R 4.4.0)
equatags            0.2.2   2025-09-14 [2] Github (ardata-fr/equatags@462e77b
equatiomatic      * 0.4.5   2025-09-21 [2] Github (datalorax/equatiomatic@572
evaluate            1.0.3   2025-01-10 [2] RSPM (R 4.4.0)
fastmap             1.2.0   2024-05-15 [2] RSPM (R 4.4.0)
flextable           0.9.7   2024-10-27 [2] RSPM (R 4.4.3)
```

```
fontBitstreamVera   0.1.1   2017-02-01 [2] RSPM (R 4.4.0)
fontLiberation      0.1.0   2016-10-15 [2] RSPM (R 4.4.0)
fontquiver          0.2.1   2017-02-01 [2] RSPM (R 4.4.0)
forcats           * 1.0.0   2023-01-29 [2] RSPM (R 4.4.0)
foreign             0.8-90  2025-03-31 [3] RSPM (R 4.4.0)
Formula             1.2-5   2023-02-24 [2] RSPM (R 4.4.0)
fs                * 1.6.5   2024-10-30 [2] RSPM (R 4.4.0)
gdtools             0.4.2   2025-03-27 [2] CRAN (R 4.4.1)
generics            0.1.3   2022-07-05 [2] RSPM (R 4.4.0)
getPass             0.2-4   2023-12-10 [2] RSPM (R 4.4.0)
ggplot2           * 3.5.2   2025-04-09 [2] RSPM (R 4.4.0)
ggplotify           0.1.2   2023-08-09 [2] RSPM (R 4.4.0)
ggpubr              0.6.0   2023-02-10 [2] RSPM (R 4.4.0)
ggsignif            0.6.4   2022-10-13 [2] RSPM (R 4.4.0)
glue                1.8.0   2024-09-30 [2] RSPM (R 4.4.0)
gridExtra           2.3     2017-09-09 [2] RSPM (R 4.4.0)
gridGraphics        0.5-1   2020-12-13 [2] RSPM (R 4.4.0)
gtable              0.3.6   2024-10-25 [2] RSPM (R 4.4.0)
helpai            * 0.1.1   2025-08-31 [2] https://sciviews.r-universe.dev (R
Hmisc               5.2-3   2025-03-16 [2] RSPM (R 4.4.0)
hms                 1.1.3   2023-03-21 [2] RSPM (R 4.4.0)
htmlTable           2.4.3   2024-07-21 [2] RSPM (R 4.4.0)
htmltools           0.5.8.1 2024-04-04 [2] RSPM (R 4.4.0)
htmlwidgets         1.6.4   2023-12-06 [2] RSPM (R 4.4.0)
httpuv              1.6.15  2024-03-26 [2] RSPM (R 4.4.0)
httr                1.4.7   2023-08-15 [2] RSPM (R 4.4.0)
httr2               1.1.2   2025-03-26 [2] RSPM (R 4.4.0)
igraph              2.1.4   2025-01-23 [2] RSPM (R 4.4.0)
interp              1.1-6   2024-01-26 [2] RSPM (R 4.4.0)
jpeg                0.1-11  2025-03-21 [2] RSPM (R 4.4.0)
jquerylib           0.1.4   2021-04-26 [2] RSPM (R 4.4.0)
jsonlite            2.0.0   2025-03-27 [2] RSPM (R 4.4.0)
katex               1.5.0   2024-09-29 [2] RSPM (R 4.4.0)
keyring             1.3.2   2023-12-11 [2] RSPM (R 4.4.0)
```

```
knitr              1.50    2025-03-16 [2] RSPM (R 4.4.0)
later              1.4.2   2025-04-08 [2] RSPM (R 4.4.0)
lattice          * 0.22-7  2025-04-02 [2] RSPM (R 4.4.0)
latticeExtra       0.6-30  2022-07-04 [2] RSPM (R 4.4.0)
learnitdown        1.9.1   2025-10-21 [1] Github (learnitr/learnitdown@9510f
learnr             0.11.5  2023-09-28 [2] RSPM (R 4.4.0)
lifecycle          1.0.4   2023-11-07 [2] RSPM (R 4.4.0)
lubridate          1.9.4   2024-12-08 [2] RSPM (R 4.4.0)
magick             2.8.6   2025-03-23 [2] RSPM (R 4.4.0)
magrittr           2.0.3   2022-03-30 [2] RSPM (R 4.4.0)
MASS             * 7.3-65  2025-02-28 [3] RSPM (R 4.4.0)
mime               0.13    2025-03-17 [2] RSPM (R 4.4.0)
mongolite          4.0.0   2025-04-01 [2] RSPM (R 4.4.0)
munsell            0.5.1   2024-04-01 [2] RSPM (R 4.4.0)
nanotime           0.3.12  2025-04-02 [2] RSPM (R 4.4.0)
nnet               7.3-20  2025-01-01 [3] CRAN (R 4.4.3)
officer            0.6.8   2025-03-23 [2] RSPM (R 4.4.0)
openssl            2.3.2   2025-02-03 [2] RSPM (R 4.4.0)
pillar             1.10.2  2025-04-05 [2] RSPM (R 4.4.0)
pkgconfig          2.0.3   2019-09-22 [2] RSPM (R 4.4.0)
PKI                0.1-14  2024-06-15 [2] RSPM (R 4.4.0)
png                0.1-8   2022-11-29 [2] RSPM (R 4.4.0)
promises           1.3.2   2024-11-28 [2] RSPM (R 4.4.0)
proto              1.0.0   2016-10-29 [2] RSPM (R 4.4.0)
pryr               0.1.6   2023-01-17 [2] RSPM (R 4.4.0)
purrr              1.0.4   2025-02-05 [2] RSPM (R 4.4.0)
R.methodsS3        1.8.2   2022-06-13 [2] RSPM (R 4.4.0)
R.oo               1.27.0  2024-11-01 [2] RSPM (R 4.4.0)
R.utils            2.13.0  2025-02-24 [2] RSPM (R 4.4.0)
R6                 2.6.1   2025-02-15 [2] RSPM (R 4.4.0)
ragg               1.3.3   2024-09-11 [2] RSPM (R 4.4.0)
rappdirs           0.3.3   2021-01-31 [2] RSPM (R 4.4.0)
RColorBrewer       1.1-3   2022-04-03 [2] RSPM (R 4.4.0)
Rcpp               1.0.14  2025-01-12 [2] RSPM (R 4.4.0)
```

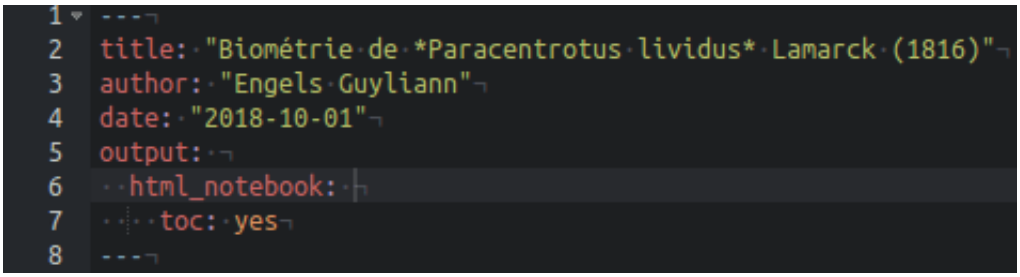
```
RcppCCTZ            0.2.13  2024-12-12 [2] RSPM (R 4.4.0)
RcppParallel        5.1.10  2025-01-24 [2] RSPM (R 4.4.0)
readr               2.1.5   2024-01-10 [2] RSPM (R 4.4.0)
remotes             2.5.0   2024-03-17 [2] RSPM (R 4.4.0)
rlang             * 1.1.5   2025-01-17 [2] RSPM (R 4.4.0)
rmarkdown           2.29    2024-11-04 [2] RSPM (R 4.4.0)
roxygen2            7.3.2   2024-06-28 [2] RSPM (R 4.4.0)
rpart               4.1.24  2025-01-07 [2] RSPM (R 4.4.0)
rprojroot           2.0.4   2023-11-05 [2] RSPM (R 4.4.0)
rstatix             0.7.2   2023-02-01 [2] RSPM (R 4.4.0)
rstudioapi          0.17.1  2024-10-22 [2] RSPM (R 4.4.0)
sass                0.4.9   2024-03-15 [2] RSPM (R 4.4.0)
scales              1.3.0   2023-11-28 [2] RSPM (R 4.4.0)
SciViews          * 1.9.2   2025-09-02 [2] https://sciviews.r-universe.dev (R
sessioninfo         1.2.3   2025-02-05 [2] RSPM (R 4.4.0)
shiny               1.10.0  2024-12-14 [2] RSPM (R 4.4.0)
shinylogs           0.2.1   2022-04-18 [2] RSPM (R 4.4.0)
shinytoastr         2.2.0   2023-08-30 [2] RSPM (R 4.4.0)
stringi             1.8.7   2025-03-27 [2] RSPM (R 4.4.0)
stringr             1.5.1   2023-11-14 [2] RSPM (R 4.4.0)
svBase            * 1.7.2   2025-11-22 [1] Github (SciViews/svBase@9411e72)
svFast            * 0.1.0   2025-08-29 [2] https://sciviews.r-universe.dev (R
svFlow            * 1.2.1   2025-08-29 [2] https://sciviews.r-universe.dev (R
svMisc            * 1.6.0   2025-08-29 [2] https://sciviews.r-universe.dev (R
svTidy            * 0.1.1   2025-12-05 [1] Github (SciViews/svTidy@81c91fd)
systemfonts         1.2.2   2025-04-04 [2] RSPM (R 4.4.0)
tabularise        * 0.7.0   2025-08-29 [2] https://sciviews.r-universe.dev (R
textshaping         1.0.0   2025-01-20 [2] RSPM (R 4.4.0)
tibble            * 3.2.1   2023-03-20 [2] RSPM (R 4.4.0)
tidyr             * 1.3.1   2024-01-24 [2] RSPM (R 4.4.0)
tidyselect          1.2.1   2024-03-11 [2] RSPM (R 4.4.0)
timechange          0.3.0   2024-01-18 [2] RSPM (R 4.4.0)
tinytable           0.8.0   2025-03-23 [2] RSPM (R 4.4.0)
tsibble             1.1.6   2025-01-30 [2] RSPM (R 4.4.0)
```

```
tzdb                0.5.0   2025-03-15 [2] RSPM (R 4.4.0)
uuid                1.2-1   2024-07-29 [2] RSPM (R 4.4.0)
V8                  6.0.3   2025-03-26 [2] RSPM (R 4.4.0)
vctrs               0.6.5   2023-12-01 [2] RSPM (R 4.4.0)
viridis             0.6.5   2024-01-29 [2] RSPM (R 4.4.0)
viridisLite         0.4.2   2023-05-02 [2] RSPM (R 4.4.0)
webshot             0.5.5   2023-06-26 [2] RSPM (R 4.4.0)
withr               3.0.2   2024-10-28 [2] RSPM (R 4.4.0)
xfun                0.52    2025-04-02 [2] RSPM (R 4.4.0)
xml2                1.3.8   2025-03-14 [2] RSPM (R 4.4.0)
xslt                1.5.1   2025-02-27 [2] RSPM (R 4.4.0)
xtable              1.8-4   2019-04-21 [2] RSPM (R 4.4.0)
yaml                2.3.10  2024-07-26 [2] RSPM (R 4.4.0)
yulab.utils         0.2.0   2025-01-29 [2] RSPM (R 4.4.0)
zeallot             0.1.0   2018-01-28 [2] RSPM (R 4.4.0)
zip                 2.3.2   2025-02-01 [2] RSPM (R 4.4.0)
zoo                 1.8-13  2025-02-22 [2] RSPM (R 4.4.0)

[1] /Users/phgrosjean/Library/R/arm64/4.4/svuser-library
[2] /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/sciviews-lib
[3] /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/library
* ── Packages attached to the search path.
```

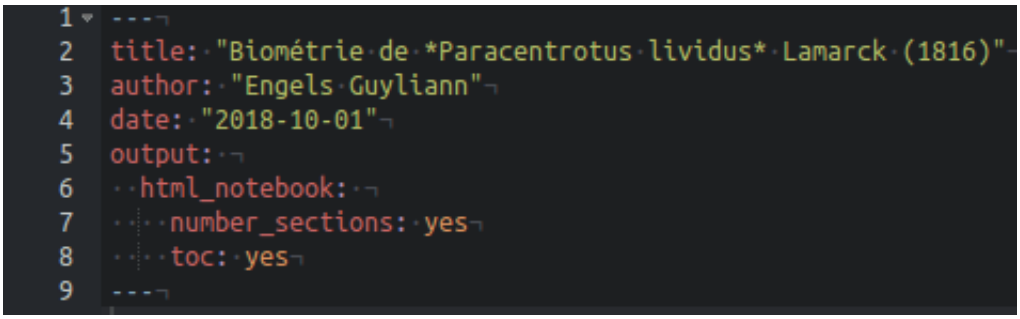
D.2.1 Table des matières

La table des matières est d'une importance capitale pour un long document (mais facultative pour un plus court rapport) afin de présenter la structure de votre œuvre aux lecteurs. Heureusement, il n'est pas nécessaire de l'écrire manuellement. La table des matières est générée automatiquement dans un rapport R Markdown. L'instruction à ajouter dans le préambule du document pour obtenir une table des matières est `toc:`

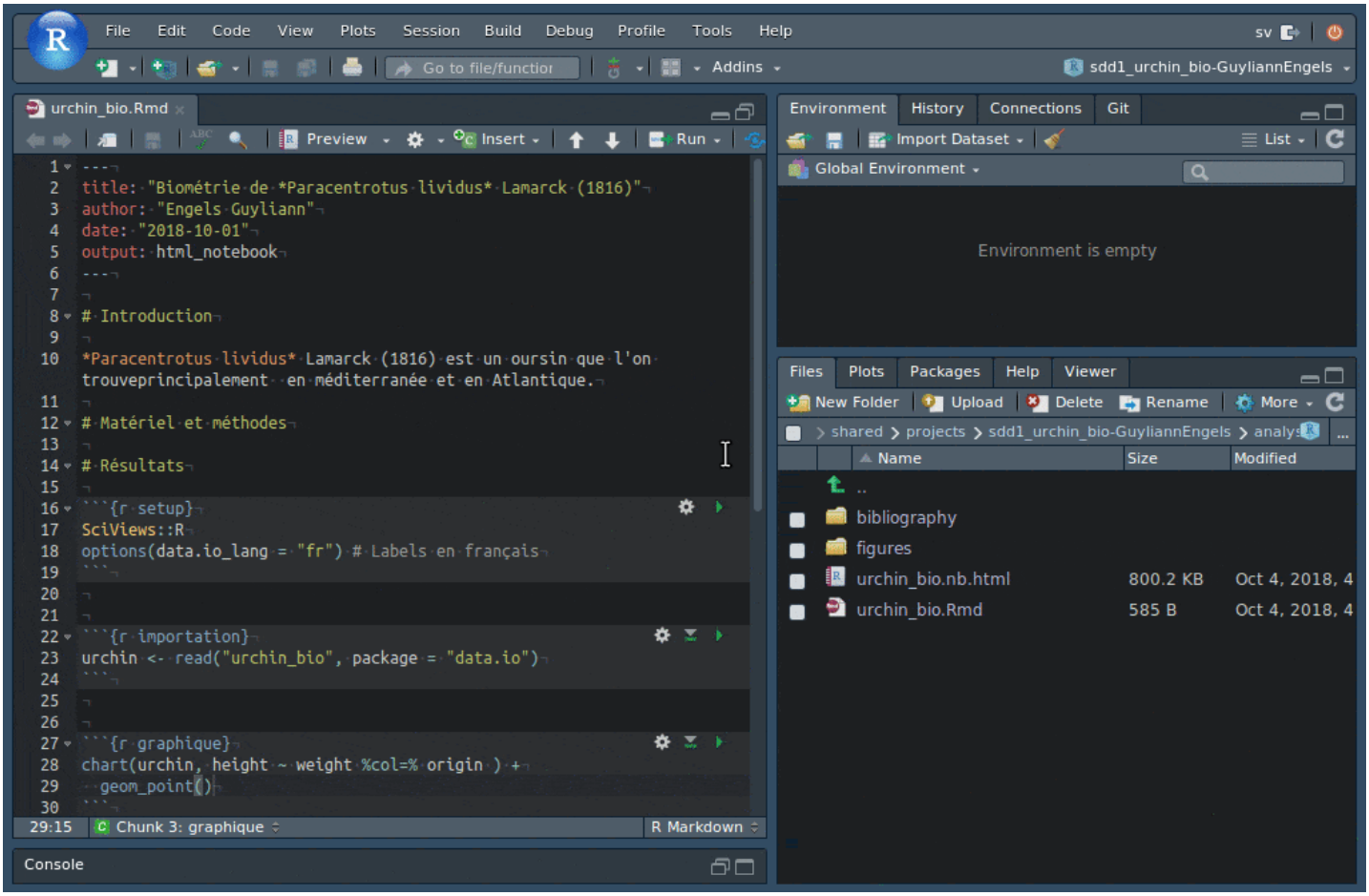
`yes` ou `toc: true` (ne l'encodez pas directement, mais sélectionnez l'option `Include table of contents` dans les options de formatage du document accessibles à partir du bouton engrenage à droite de `Preview || Prévisualiser` ou `Knit || Tricoter` et ensuite `Output Options... || Options de sortie...` pour les documents R Notebook ou R Markdown, mais pour Quarto, vous devrez rentrer le paramétrage manuellement dans l'entête). Lorsque vous fermerez cette boite de dialogue de configuration, l'entrée *ad hoc* sera ajoutée pour vous dans le préambule.



Vous pouvez aussi choisir de numéroter vos titres automatiquement. L'instruction à ajouter en plus de `toc: yes` dans le préambule pour avoir des titres numérotés est `number_sections: yes` (`number-sections: true` pour un document Quarto). Encore une fois, passez par la boite de dialogue de configuration, et cochez-y l'entrée `Number section headings` .



Voyez l'animation ci-dessous pour accéder à la boite de dialogue de configuration du document R Markdown/R Notebook.



D.2.2 Introduction

L'introduction d'un travail scientifique a pour principal objectif de replacer l'étude scientifique réalisée dans son contexte. La règle la plus importante est qu'**un lecteur n'ayant jamais entendu parler de cette étude doit comprendre l'intégralité du rapport.** L'introduction doit donc permettre de :

- Remettre l'expérience dans son contexte,
- Décrire le ou les organismes étudiés : par exemple, caractéristiques générales, distribution géographique, biotope…

Notez que l'ajout d'images ou d'une carte de distribution est un plus dans l'introduction.

D.2.3 But

Le but permet de synthétiser la question posée en fonction du contexte de l'expérience expliqué dans l'introduction. Il s'agit donc généralement d'un seul paragraphe court.

D.2.4 Matériel & méthodes

La section matériel & méthodes permet de décrire les aspects techniques de l'étude comme le matériel employé et les méthodes mises en œuvre (protocoles des manipulations et des mesures effectuées) pour acquérir les données. Cette section est également le lieu de description des techniques statistiques utilisées pour analyser les données, des programmes informatiques employés (pensez toujours à citer un logiciel en indiquant son nom **et son numéro de version** ainsi qu'un lien vers où le trouver)…

> Citer **RStudio** comme étant le logiciel qui a servi à faire les calculs statistiques est une erreur fréquence. En effet, c'est bien dans RStudio que vous travaillez. Mais en fait, les calculs sont réalisés en arrière plan dans le logiciel **R**. C'est ce dernier qu'il faut donc citer. Pour connaitre la version de R utilisez, regardez en haut de l'onglet **Console** de RStudio. Vous pouvez aussi utiliser l'instruction R `R.version.string` pour obtenir la version de R utilisée. Vous pouvez inclure cette instruction dans un "chunk en ligne" dans le texte qui s'écrit comme ceci : `r r R.version.string`. Pour citer R lui-même dans les références, vous pouvez utiliser l'instruction `citation()` dans la Console de RStudio (une version bibtex de la citation est même proposée).

D.2.5 Résultats

Les résultats vont généralement contenir deux parties :

- La description des données, via l'exploration des données récoltées (avec graphiques et/ou estimateurs statistiques)
- L'application des outils statistiques pertinents pour répondre à la question posée

D.2.6 Discussion

Cette section reprend l'interprétation biologique des résultats et la remise dans un contexte plus général, notamment en comparant ces résultats à des observations connexes réalisées par d'autres auteurs. Il est crucial d'avoir un regard critique sur les résultats obtenus. Cette mise en contexte aide en ce sens.

D.2.7 Conclusion(s)

La section conclusion(s) (et perspectives) va résumer les principales implications à retenir de notre étude et éventuellement, proposer des perspectives afin de poursuivre la recherche dans cette thématique.

D.2.8 Bibliographie (ou références)

La rédaction de travaux s'appuie toujours sur une recherche bibliographique au préalable. Il faut documenter convenablement les sources bibliographiques au sein de cette section pour éviter le **plagiat** volontaire ou involontaire. Une multitude de programmes existent pour faciliter la gestion de votre base de données bibliographique comme Zotero, Mendeley, ou encore Endnote.

- Pour générer correctement ses références bibliographiques dans un document R Markdown/R Notebook, consulter ceci. Pour un document Quarto, lisez ceci. Il s'agit de manuels en anglais qui expliquent comment encoder sa bibliographie et comment la citer dans le texte à l'aide de balises Markdown spécifiques.

D.3 Nom des espèces

Le nom complet d'une espèce en biologie suit une convention particulière, propre à la nomenclature binomiale de Linné que vous devez utiliser dans tous vos travaux. Partons de l'exemple de l'oursin violet. Il s'agit ici du **nom vernaculaire** en français. Mais ce nom n'est pas assez précis pour être utilisé seul dans un travail scientifique. En effet, le nom vernaculaire d'une espèce change d'une langue à l'autre. Il peut aussi varier d'une région géographique à l'autre, ou pire, il peut désigner des espèces différentes selon les endroits. Seul le **nom latin** fait référence ! Une espèce est classée de la manière suivante (les niveaux de classification les plus importants sont mis en gras) :

- **Règne** : Animalia
- **Embranchement** : Echinodermata
- Sous-Embranchement : Echinozoa
- **Classe** : Echinoidea
- Sous-classe : Euechinoidea
- Super-ordre : Echinacea
- **Ordre** : Camarodonta
- Infra-ordre : Echinidae
- **Famille** : Parachinidae
- **Genre** : *Paracentrotus*
- **Espèce** : *lividus*

Pour former le nom binomial de l'oursin violet, on utilise le genre et l'espèce de la classification ci-dessus :

- *Paracentrotus lividus*

En toute rigueur, il faut aussi associer le **nom du naturaliste** qui a nommé et décrit l'espèce (on parle de son "inventeur") et l'année de la publication de la description (on parle de diagnose en biologie), et ce, uniquement la première fois qu'on cite cette espèce dans notre rapport.

- *Paracentrotus lividus* Lamarck 1816

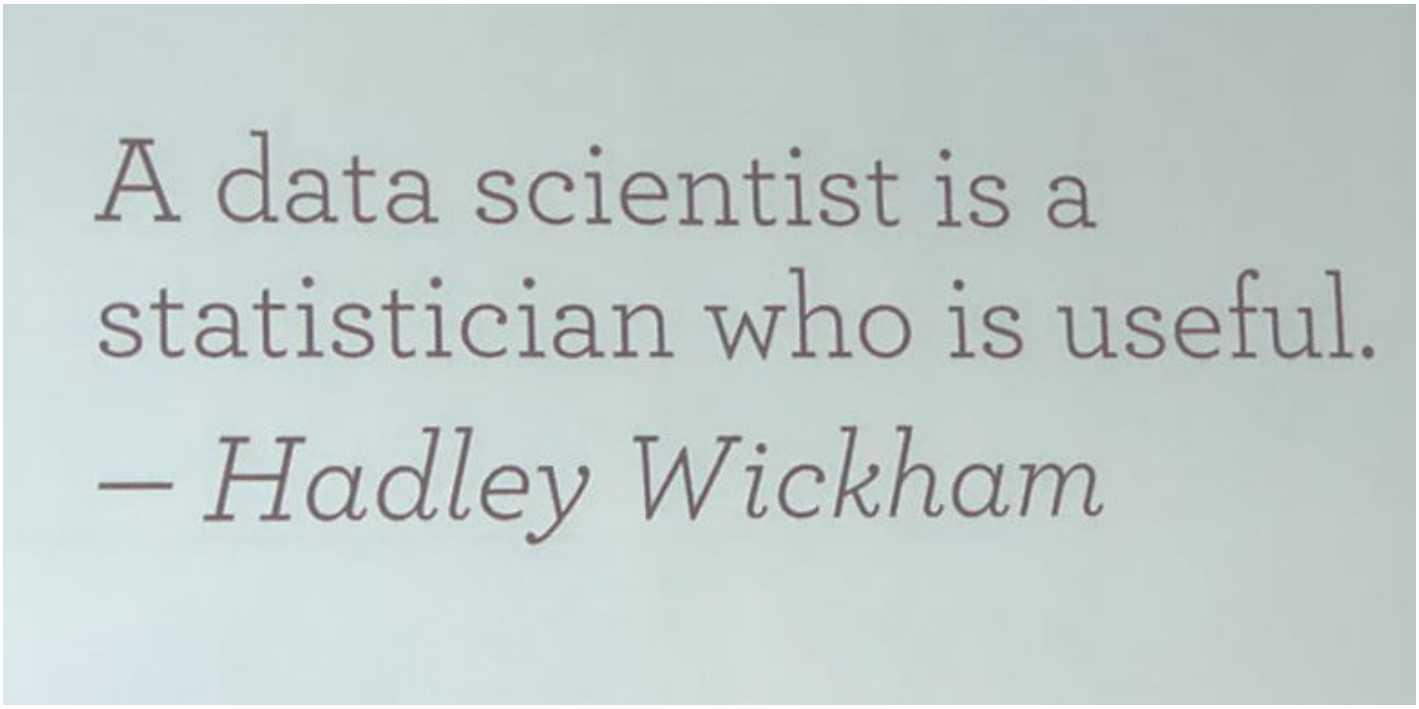
Le nom de l'inventeur et l'année sont placés entre parenthèses si le nom du genre ou de l'espèce a changé depuis la première description. Lors de la première citation d'une espèce, et certainement dans le titre ou le résumé, il est indispensable de spécifier le nom latin complet de l'espèce (genre espèce) qui pourra être éventuellement abrégé par la suite en indiquant la première lettre du genre, à condition que cela ne prête pas à confusion dans le contexte. Dans l'exemple cité, on pourra écrire ensuite *P. lividus* plus loin dans le texte (pour autant que cela ne prête pas à confusion, bien sûr). Notez aussi que les noms latins s'écrivent toujours en italique dans le texte.

Appendice E Communication & esprit critique

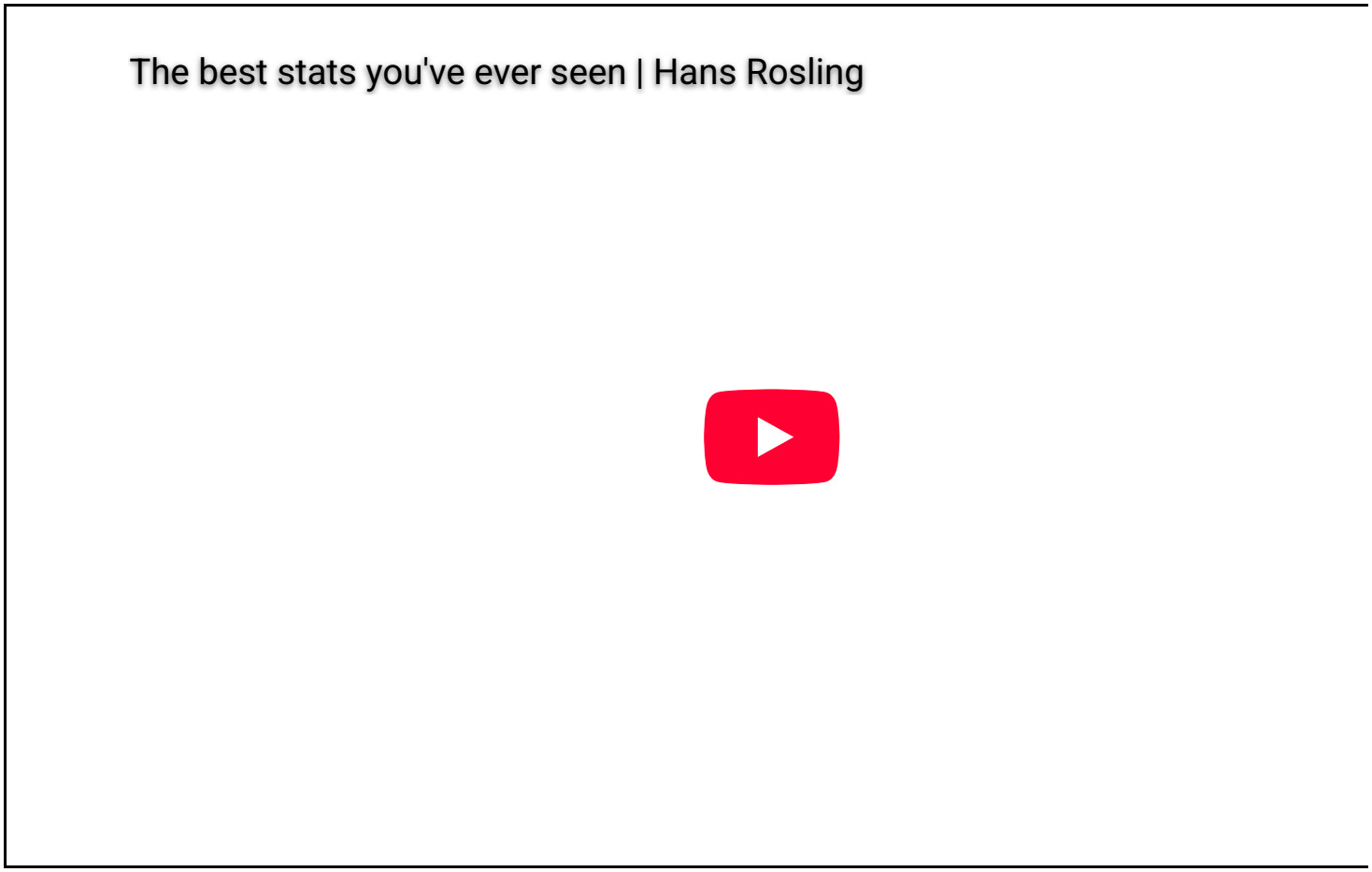
Bien que cela semble être des compétences “douces” (soft skills), la communication et l’esprit critique sont **indispensables** pour un scientifique des données. En effet, il doit être capable de communiquer clairement ses résultats à des non-initiés, et aussi de critiquer les résultats d’autres scientifiques. Ces deux compétences sont essentielles pour réussir dans ce domaine.

E.1 Communication

Être capable de bien communiquer ses résultats est vital en science des données. Ce n'es pas si facile, car il faut pouvoir simplifier les analyses et utiliser au mieux les visuels (c'est-à-dire, les graphiques) pour raconter une histoire qui soit à la fois captivante et compréhensible. **Communiquer le fruit de ses recherches de la meilleure façon qui soit pour que les non-initiés puissent le comprendre fait partie du bagage indispensable du scientifique des données**.



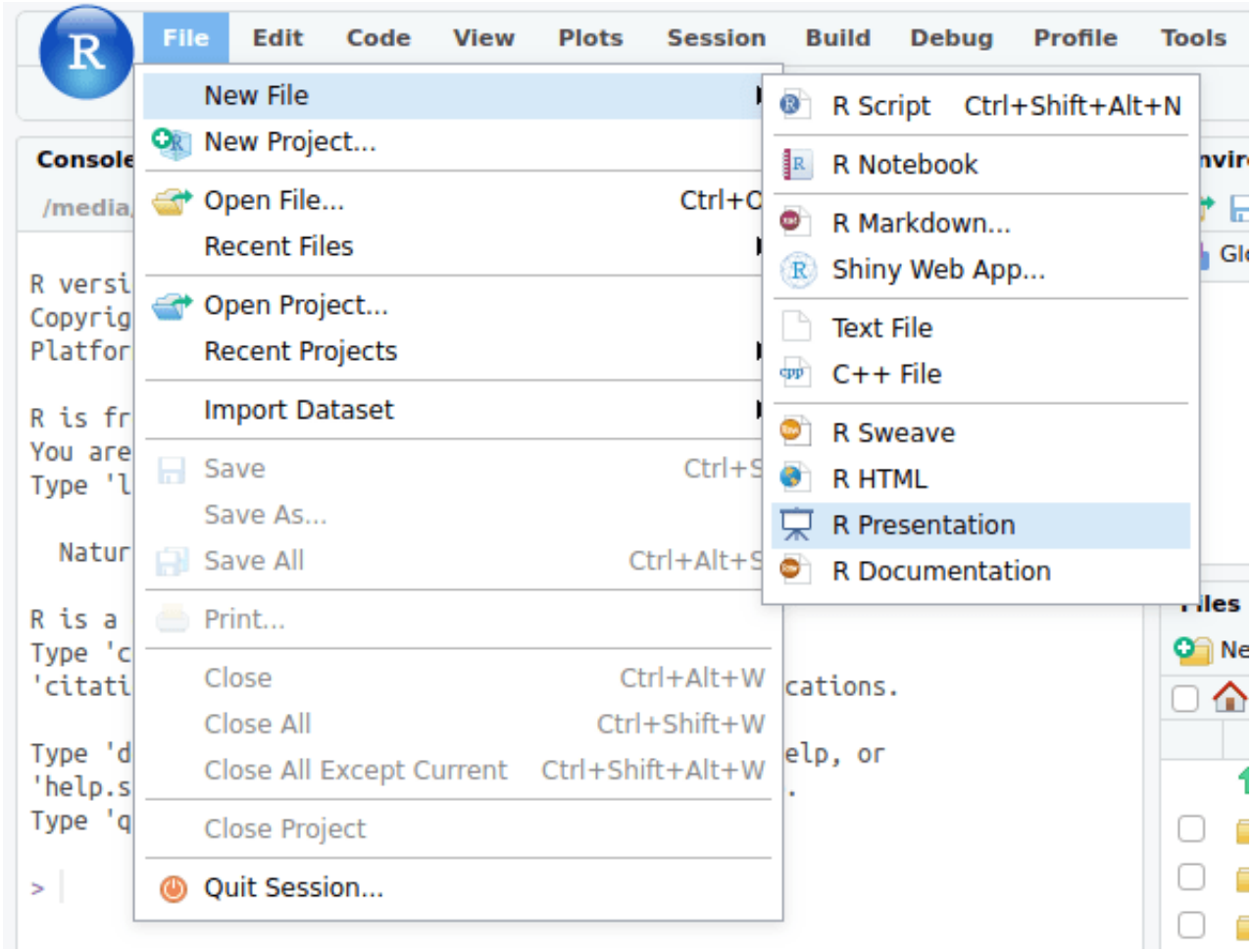
Hans Rosling était sans nul doute l'un des plus doués pour communiquer des résultats statistiques. La vidéo suivante est un peu longue (20min) et en anglais, mais elle en vaut vraiment la peine[71]. De plus, il explique à quel point il est important de partager et de rassembler les données dans de grandes bases de données, et ensuite d'en tirer des études *utiles* pour l'humanité. C'est l'avenir des sciences des données, y compris en biologie, qu'il est en train de prédire là à l'époque où il a tourné cette vidéo.



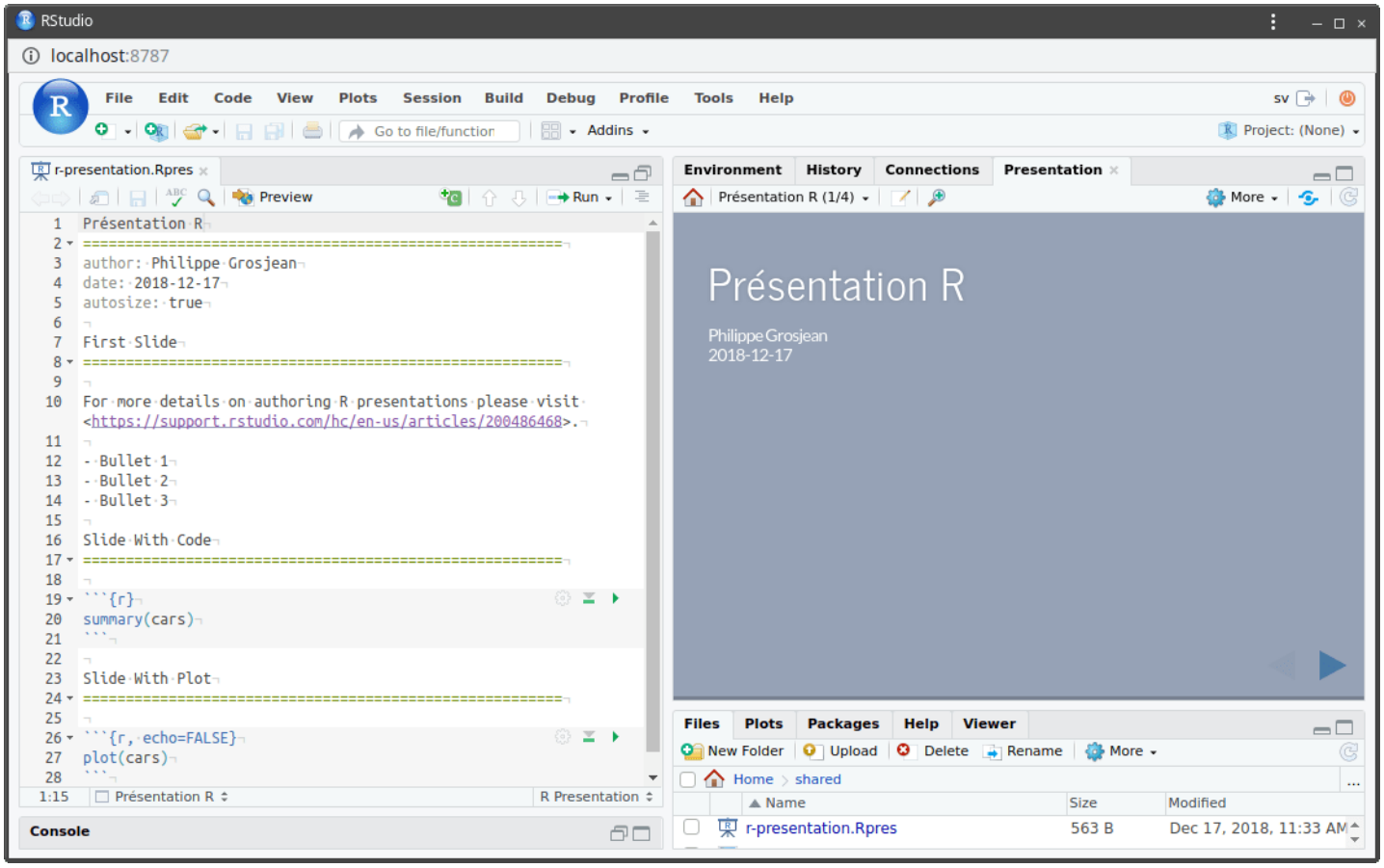
E.1.1 Présentations R Markdown et Quarto

Parmi tous les formats R Markdown ou Quarto, il en existe plusieurs adaptés aux présentations de type “PowerPoint”. Le format **R Presentation** est très simple et parfaitement intégré dans RStudio. Divers formats compilés depuis R Markdown sont également disponibles. Les formats **ioslides** ou **Slidy**, ou **xaringan** sont adaptés aux explorateurs Web. Le format **Beamer** permet de créer une présentation en PDF. Enfin, il est possible de compiler sa présentation au format **PowerPoint**.

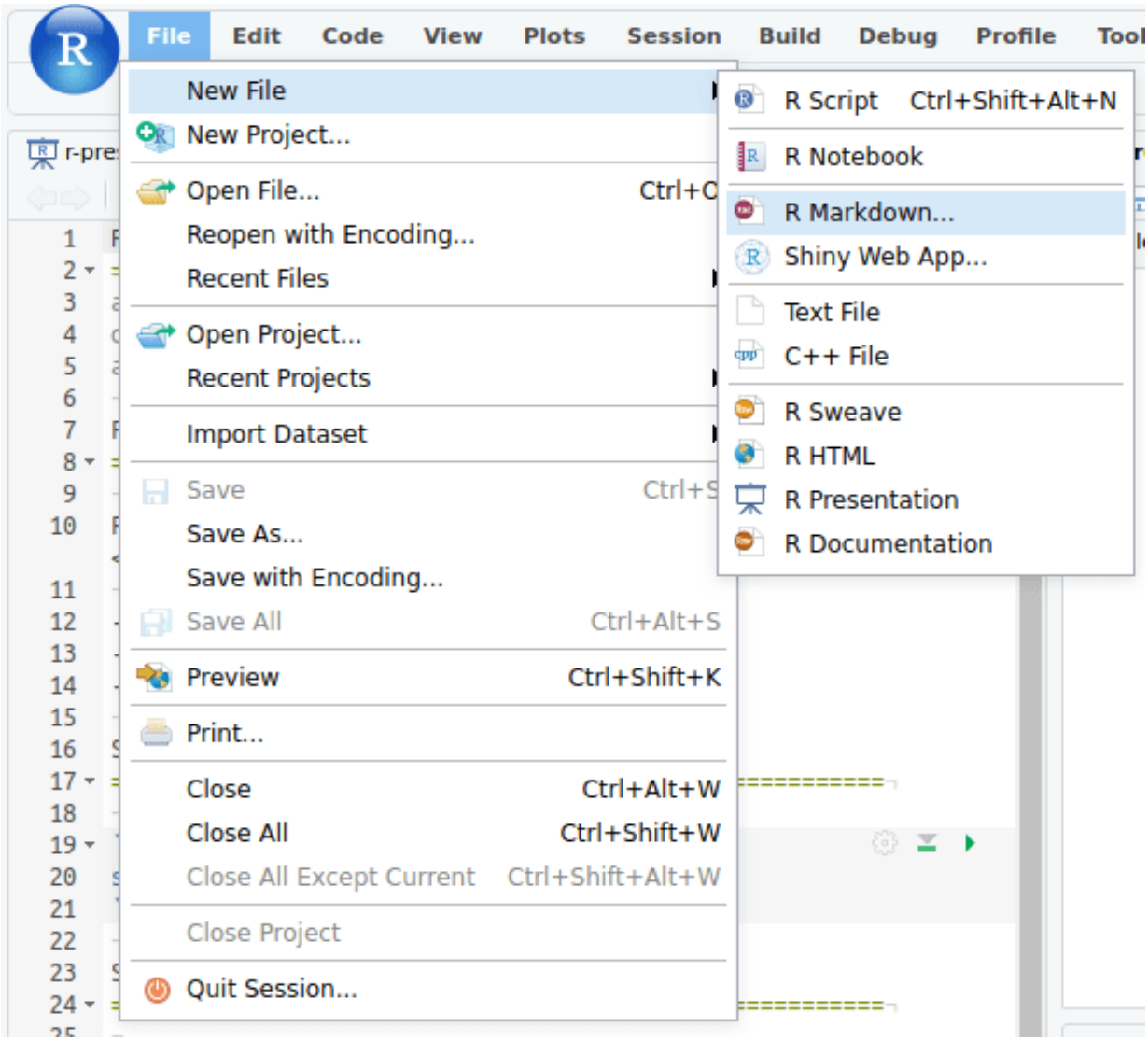
Le format **R Presentations** (menu `File -> New file -> R Presentation` dans RStudio) est très simple pour créer des diapositives de présentation directement depuis Markdown sous forme HTML, et pouvant s’exécuter dans RStudio ou dans un explorateur Web.

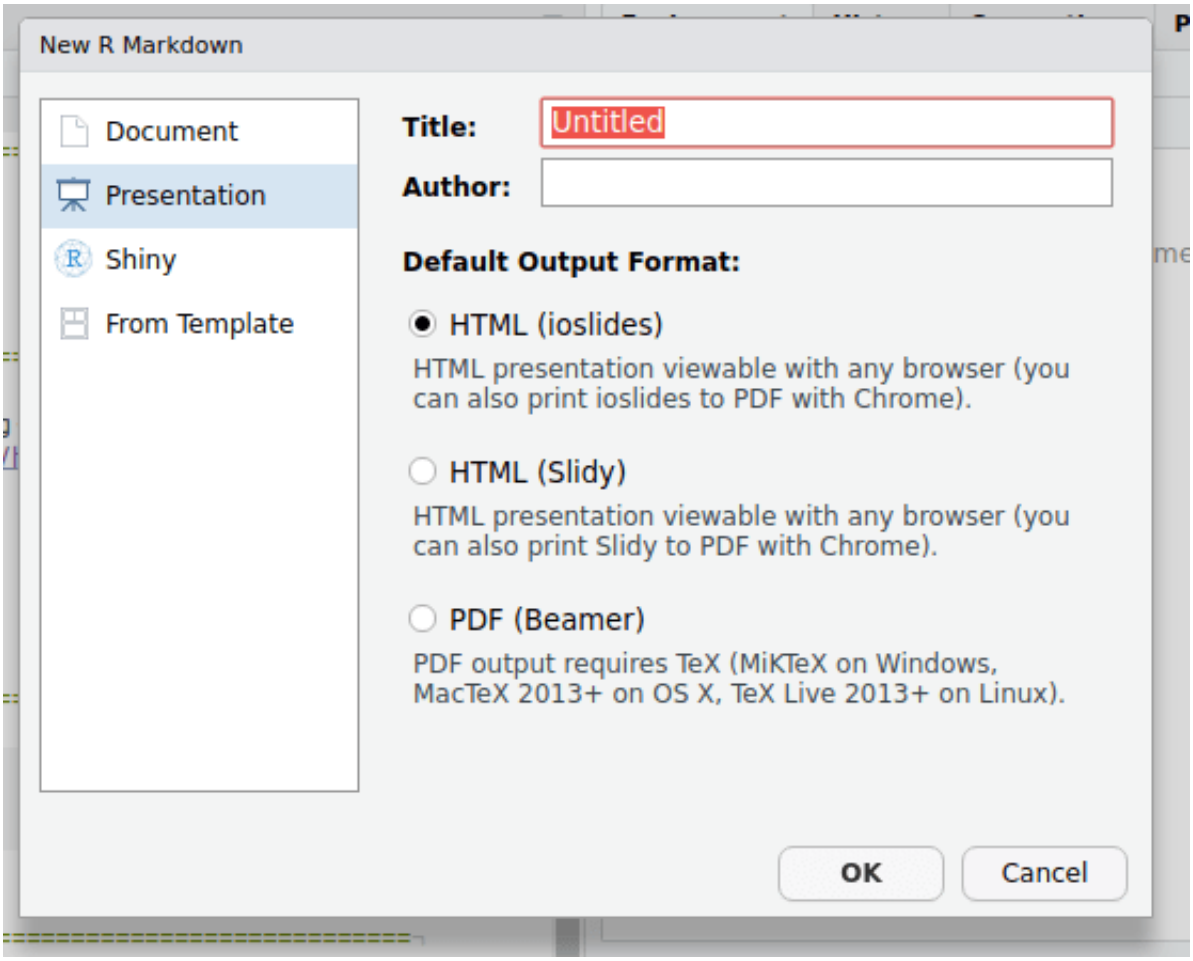


Ce type de présentation est très bien intégré dans RStudio. Outre l'édition au format R Markdown la présentation elle-même apparaît dans un onglet spécial **Presentation** en haut à droite qui n'interfère pas avec les autres onglets. Par contre, les possibilités de personnalisation sont plutôt limitées.

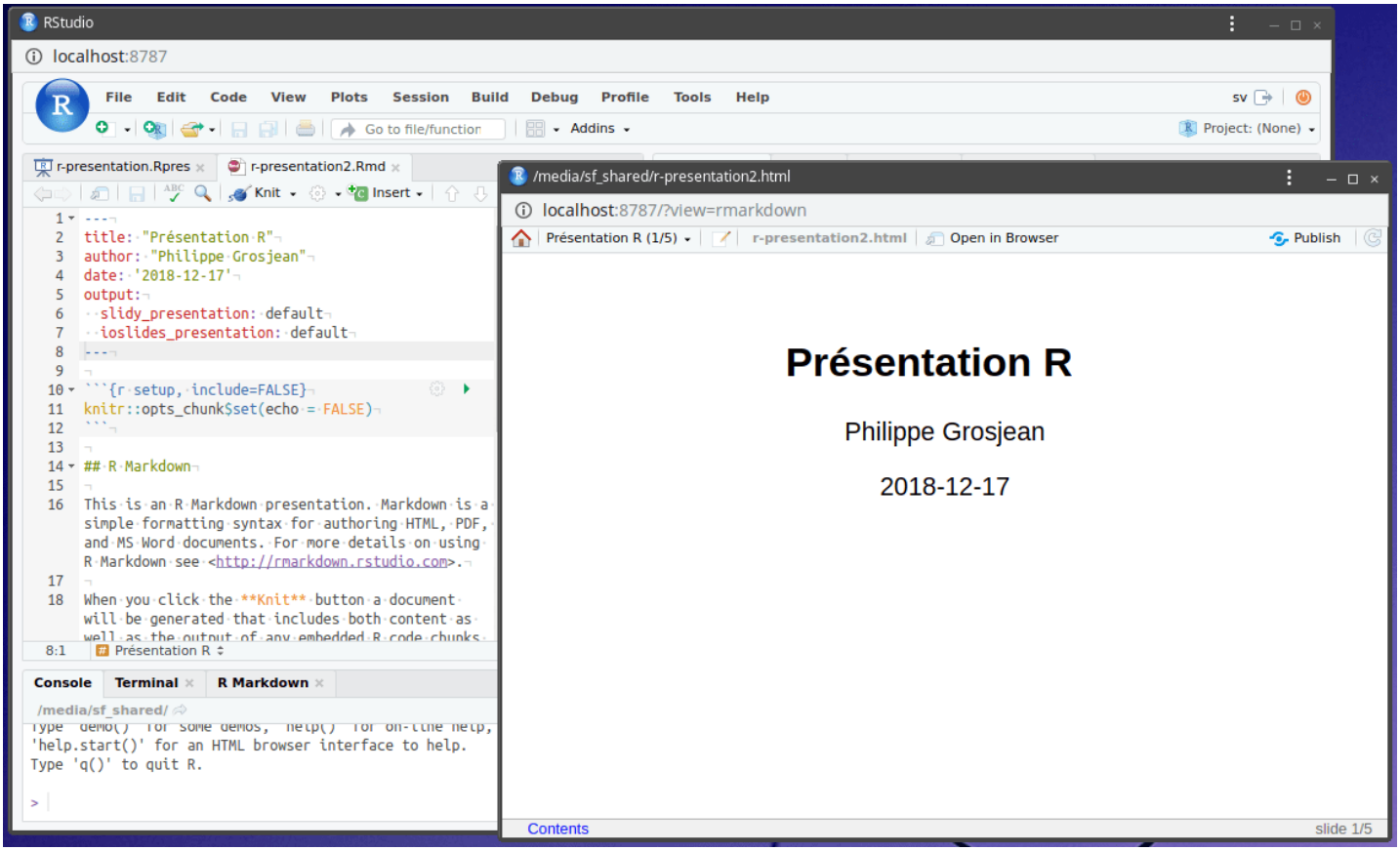


Deux formats basés sur le HTML et JavaScript (comprenez, des formats qui s'exécutent dans un explorateur Web) sont aussi proposés par défaut via les types de documents R Markdown ou Quarto (menu `File -> New File -> R Markdown...` ou `File -> New File -> R Quarto Document` , puis `Presentation`).


File
Edit
Code
View
Plots
Session
Build
Debug
Profile
New File
New Project...
Open File... Ctrl+O
Reopen with Encoding...
Recent Files
Open Project...
Recent Projects
Import Dataset
Save Ctrl+S
Save As...
Save with Encoding...
Save All Ctrl+Alt+S
Preview Ctrl+Shift+K
Print...
Close Ctrl+Alt+W
Close All Ctrl+Shift+W
Close All Except Current Ctrl+Shift+Alt+W
Close Project
Quit Session...
R Script Ctrl+Shift+Alt+N
R Notebook
R Markdown...
Shiny Web App...
Text File
C++ File
R Sweave
R HTML
R Presentation
R Documentation

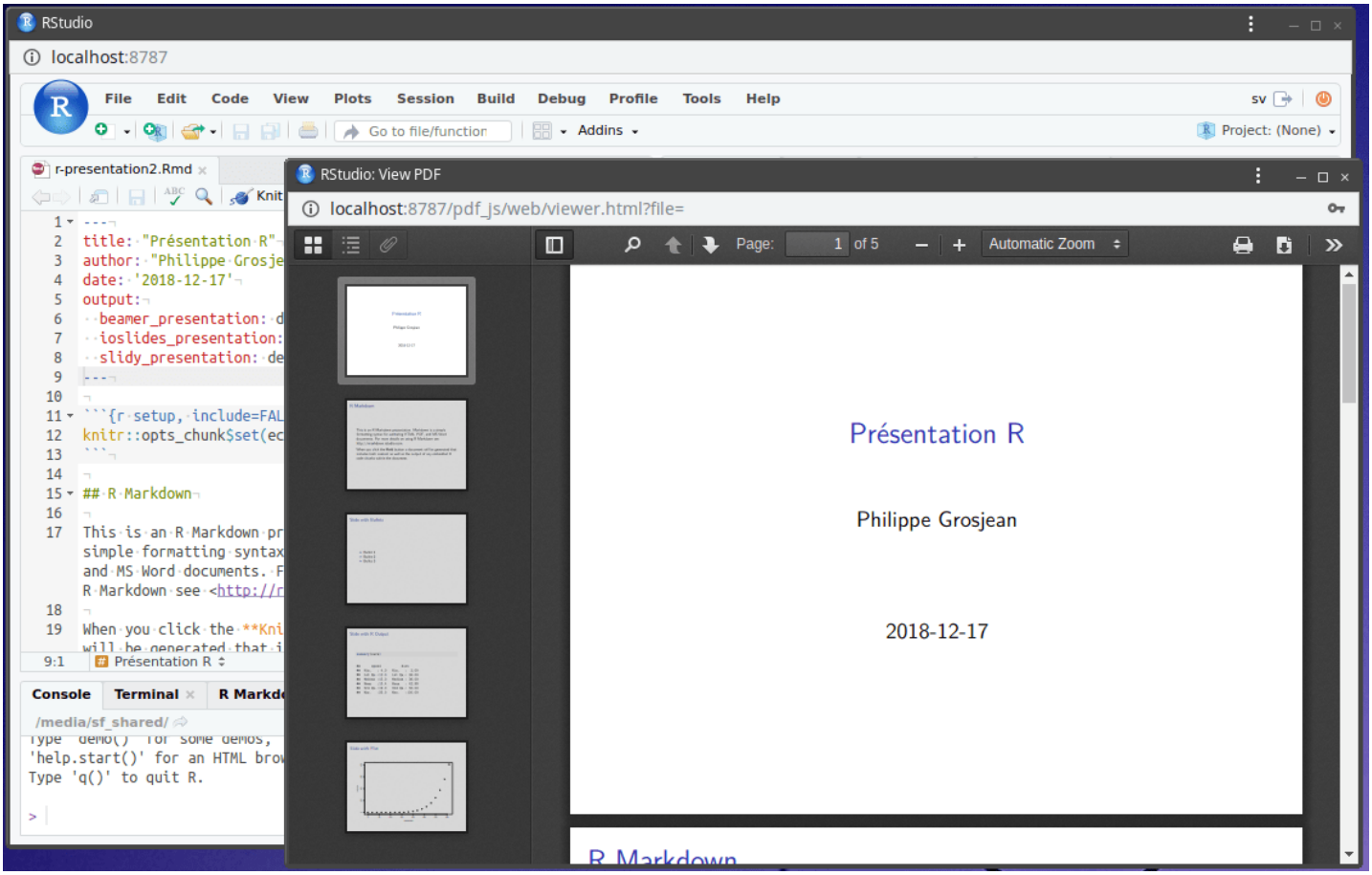


Les deux formats Web par défaut sont **ioslides** et **Slidy** pour R Markdown ou le format est **Reveal JS** pour Quarto. L'apparence et les fonctionnalités des différents systèmes diffèrent quelque peu. Le mieux est de tester les différentes versions et de choisir celui qu'on préfère. La présentation apparaît dans une fenêtre séparée. Les possibilités de personnalisation sont plus poussées, mais elles se font à l'aide de feuilles de styles au format CSS. Cela impose de comprendre leur logique.

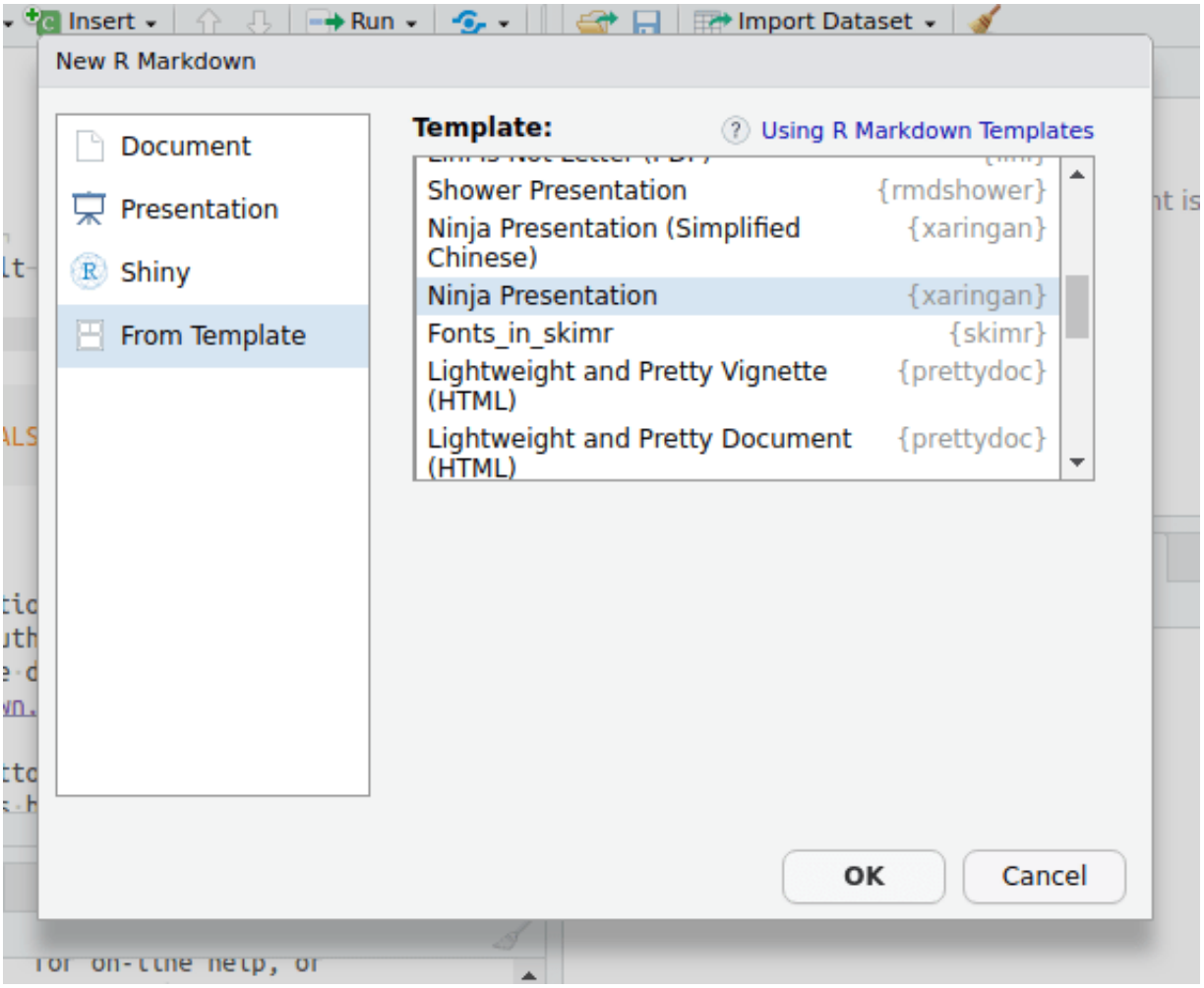


Une troisième option est de compiler un document PDF de présentation en passant par le package **Beamer** sous LaTeX. L'avantage est que c'est lisible partout. L'inconvénient : seul du *contenu statique* est accepté (pas de gifs animés, pas de vidéos ou difficilement directement dans la présentation). Sinon, les possibilités de personnalisation sont immenses[72].

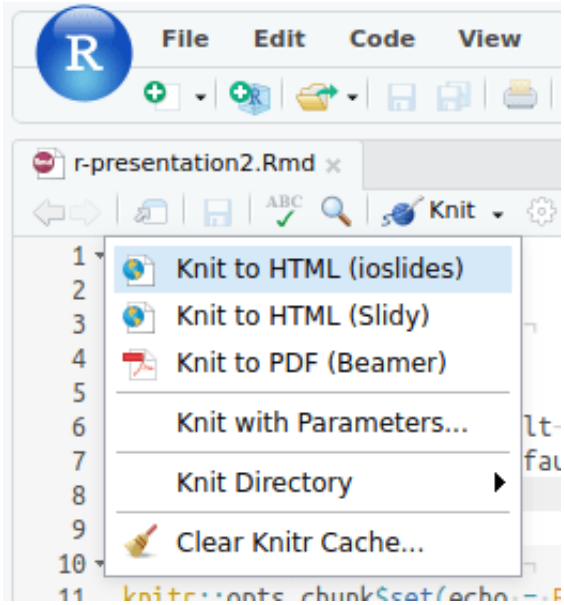
Avec le format **Beamer**, vous voyez le PDF résultant dans une fenêtre séparée. Cette fenêtre ne permet pas de lancer la présentation. Il vous faut l'ouvrir dans un lecteur PDF séparé qui offre cette fonctionnalité (Acrobat Reader, Mac Preview, SumatraPDF sous Windows, eVince sous Linux …) pour visionner votre présentation confortablement.



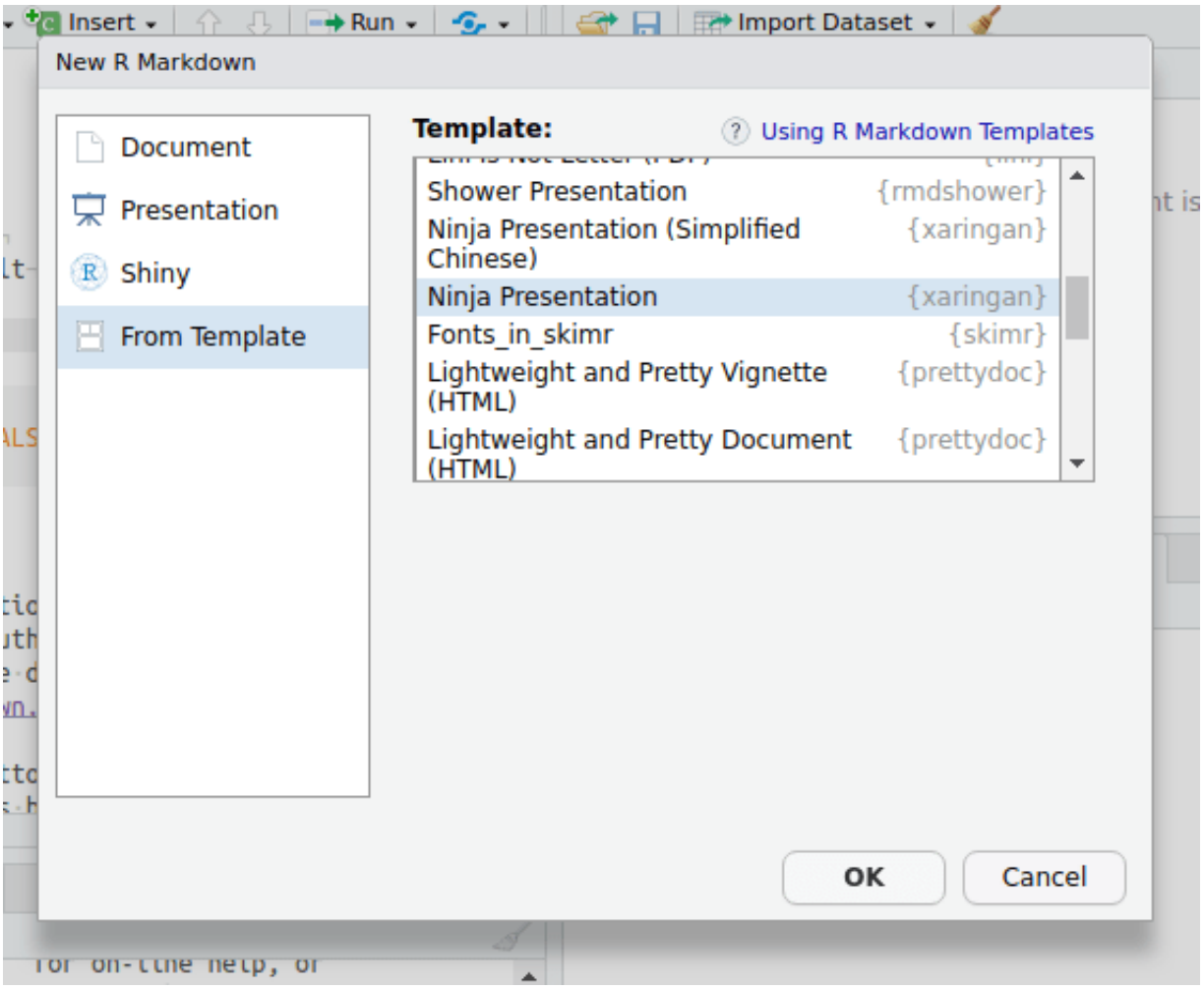
La quatrième option est de compiler un document au format **PowerPoint**. Il n'est pas possible d'avoir une prévisualisation avec ce format. Il vous faut l'ouvrir dans un lecteur de fichier **PowerPoint**. Il est possible de personnaliser ce format à l'aide d'un template comme présenter dans la section sur les présentations PowerPoint du livre R Markdown: The definitive Guide.



Les cinq formats **ioslides**, **Slidy**, **Reveal JS**, **Beamer** et **PowerPoint** sont parfaitement intégrés à RStudio, avec un menu contextuel dans le bouton `Knit` (`Tricoter`) ou `Render` (`Rendu`) qui permet de passer facilement de l'un à l'autre pour autant que vous n'utilisez pas des balises spécifiques à l'un de ces quatre formats.



Enfin, quelques packages R additionnels proposent d'autres formats de présentation. Dans la SciViews Box, vous avez {xaringan} qui propose un moteur particulièrement sophistiqué et flexible. Ces outils sont plus spécialisés, mais aussi plus puissants pour créer des présentations au format HTML à visionner dans un explorateur Web.

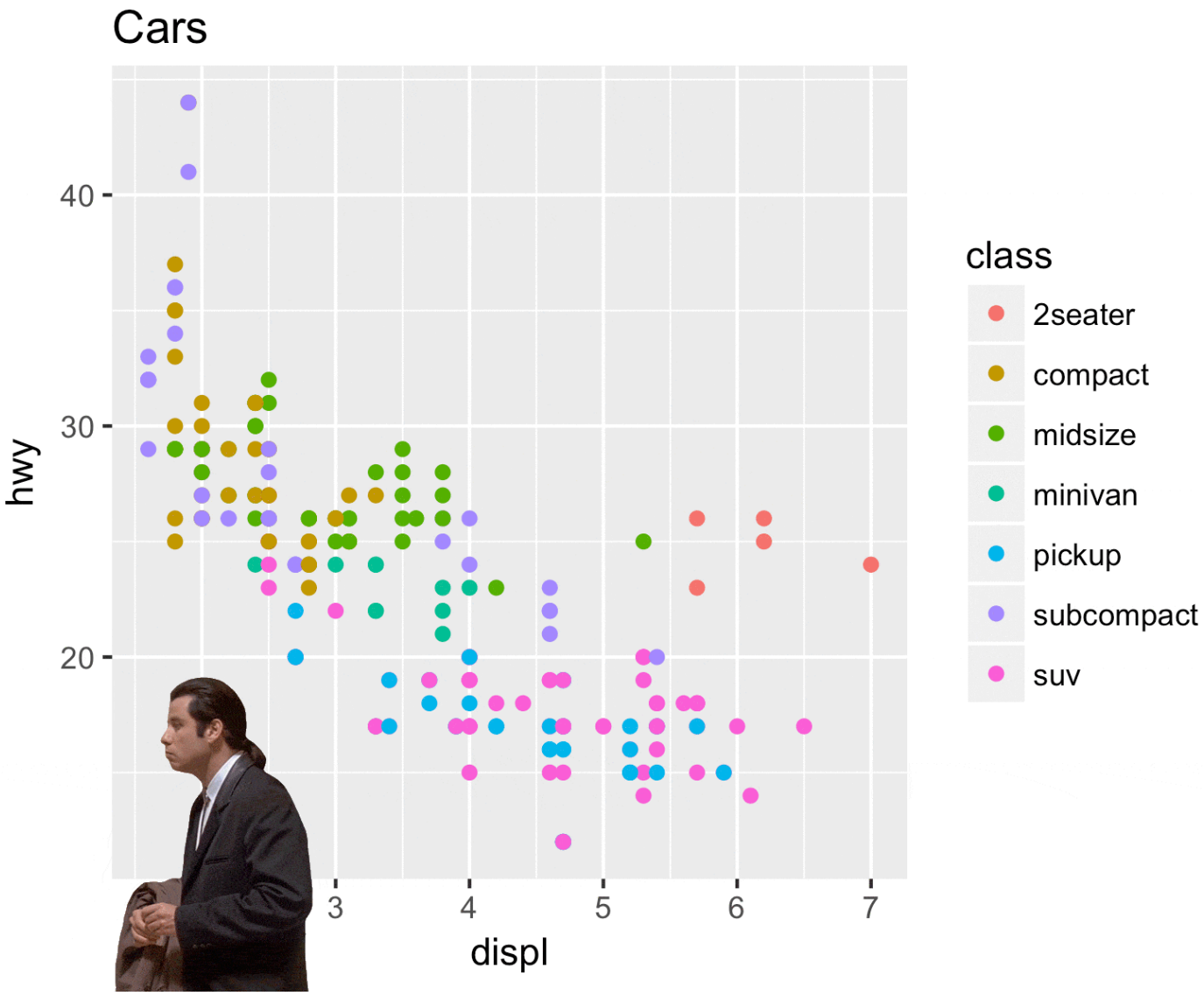


> Quel type de présentation R Markdown ou Quarto choisir au final ? Toute cette panoplie d'options ne facilite pas notre choix. En fait, c'est plus une question de goût personnel. Essayez les différentes options par vous-même. Le choix principal est, au final, entre un format HTML, PDF ou PowerPoint. Le format PDF est, par définition, plus portable. Cependant, il ne permet que du contenu statique. Si vous avez des gifs animés, des graphiques interactifs, ou des vidéos, alors orientez-vous plutôt vers un moteur HTML/Javascript.

71. Vous pouvez activer les sous-titres en anglais via la barre de boutons en bas de la vidéo.↩

72. Toutes nos présentations dans le cadre du cours sont au format R Markdown/Beamer avec un template UMONS/SDD fortement personnalisé.↩

E.2 Critique statistique

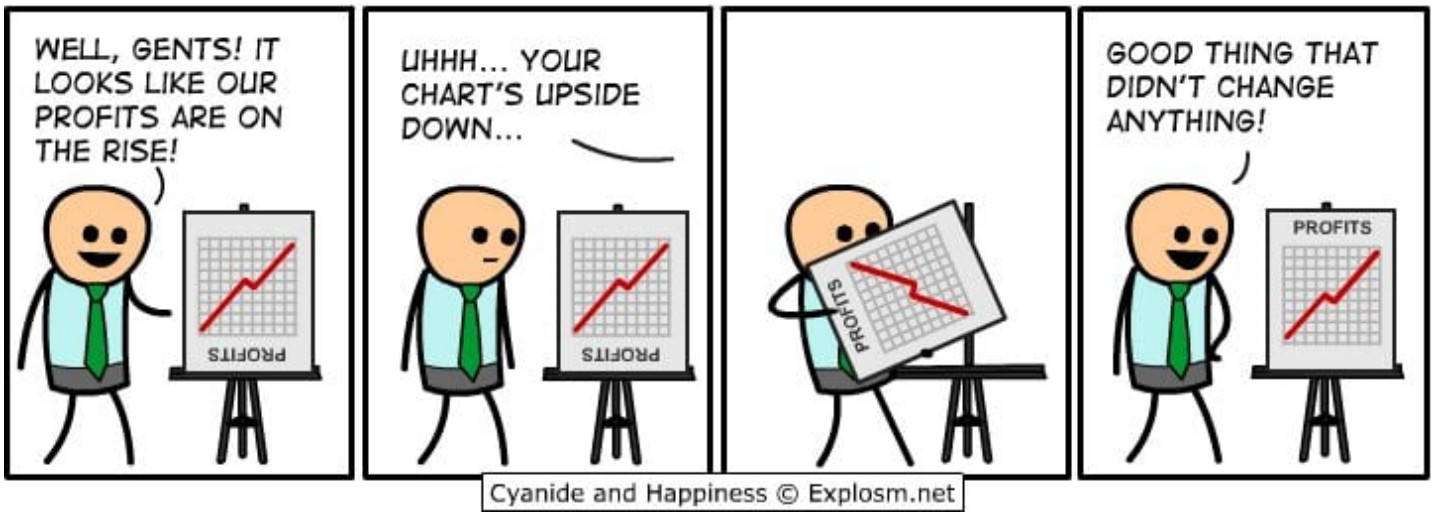


Les statistiques ont mauvaise presse auprès de certaines personnes qui pensent qu'on peut leur faire dire tout et son contraire. Cela a même donné lieu à des ouvrages comme "Attention, statistiques ! Comment en déjouer les pièges" ou "How to lie with statistics". Derrière des titres provocateurs, ces ouvrages présentent, en fait, de manière sérieuse les

pièges principaux et les moyens de les déjouer. Car, en réalité, ce n'est pas l'usage des statistiques qui est en cause ici, mais son **mauvais** usage. Voir aussi "Statistical reasoning for everyday life".

> Dans la littérature scientifique et tout autour de nous, nous pouvons trouver des exemples de mauvais usages des statistiques (application *erronée* de méthodes statistiques). Quelquefois, il s'agit de triche manifeste, mais la plupart du temps

c’est par ignorance. Développer un **esprit critique** statistique est important pour pouvoir démasquer ces diverses situations et ne pas tomber soi-même dans les pièges les plus grossiers.



Voici quelques conseils qui vous aideront à développer votre esprit critique statistique.

- La formulation statistique est-elle en adéquation avec la question biologique posée ?
- Y a-t-il des biais dans les techniques d’échantillonnage et/ou de mesure ?
- Les graphiques sont-ils adéquats par rapport à ce qui doit être montré ?
- Les axes sont-ils placés correctement, et sont-ils bien libellés ?
- Le graphique respecte-t-il les conventions ?
- Les unités sont-elles correctes ?
- Les calculs sont-ils corrects ?
- Les variables sont-elles du type correct pour l’analyse (qualitative ordonnée ou non, ou alors, quantitative discrète ou continue) ?
- Les conditions d’application des tests statistiques sont-elles respectées ?
- La taille de l’échantillon est-elle suffisante ?
- N’y a-t-il pas pseudo-réplication (plusieurs mesures issues d’un même individu considérées comme des observations indépendantes) ?
- Les grandeurs observées sont-elles plausibles ? Vous pouvez vous rapporter à des éléments connus et comparer. Par exemple, si l’on vous dit qu’une souris adulte pèse 1g, est-ce plausible ou non ? Faites une recherche sur le Web, ou un raisonnement du genre : une souris est constituée principalement d’eau. Un gramme d’eau occupe un volume de 1 cm^3. Le volume de la souris adulte est-il supérieur, égal ou inférieur à un

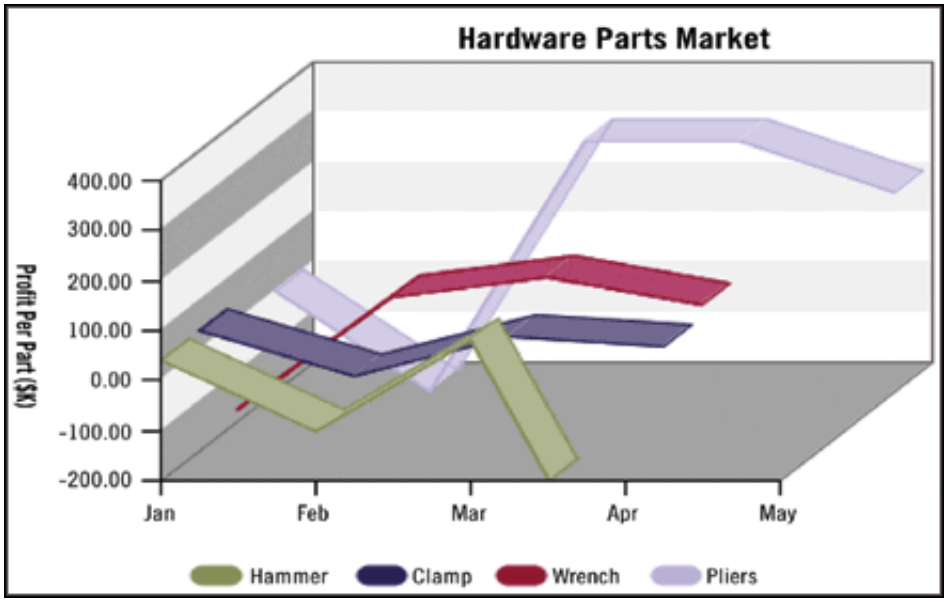
cube de 1 cm de côté ?

- Les mêmes données ne peuvent être utilisées deux fois. Si elles sont utilisées pour découvrir un effet, et en même temps pour le vérifier, c'est incorrect.
- Une corrélation ou un effet fortuit n'est-il retiré d'une grande quantité de tests non significatifs ? Soyez attentifs aux tests multiples réalisés sans ajustement du seuil α. Cela s'appelle du *"p hacking"*.
- Un test d'hypothèse est-il significatif après avoir ajouté de manière itérative de nouveaux individus et mesures à l'échantillon jusqu'à obtenir le résultat désiré (autre forme de *"p hacking"*).
- Les conclusions sont-elles en adéquation avec ce qui est observé dans les données ? Les conclusions répondent-elles à la ou les questions posées initialement ?

Pour terminer ce module, nous vous proposons quelques situations (soit des problèmes, soit des graphiques) qui ont toutes en commun d'être erronées. À vous de trouver ce qui ne va pas. Pour ne pas fausser la donne, les réponses ne **sont pas** fournies dans ce document, mais seront discutées en classes tous ensemble.

Graphe en rubans

Que pensez-vous du graphique suivant ?

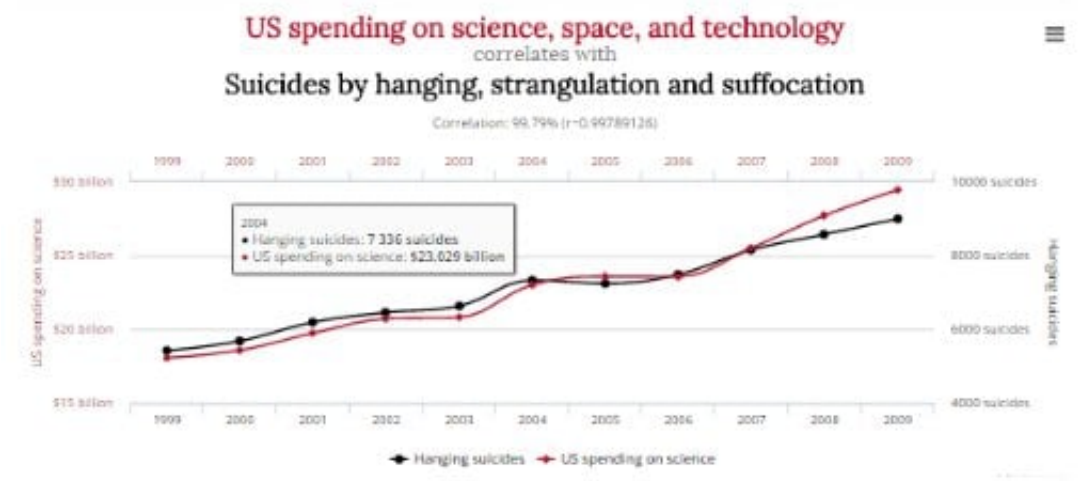


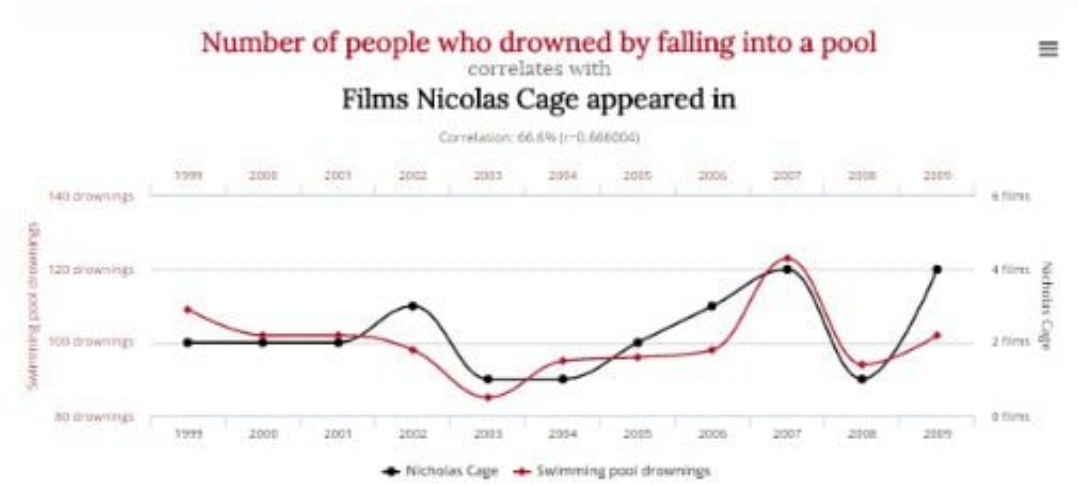
Longévité

Un chercheur compile les statistiques de longévité de diverses professions. Pour ce faire, il encode les données des certificats de décès (nom, âge au moment du décès et profession). Il calcule ensuite l'âge moyen de décès par profession. Il constate que la valeur minimale est observée **chez les étudiants**, avec une valeur moyenne de seulement 20,7 ans (Wainer, Palmer & Bradlow, A selection of selection anomalies, *Chance*, vol. 11, n°2).

La « profession » d'étudiant est-elle réellement plus dangereuse que celle de policier, chauffeur de taxi, ou cascadeur ? Expliquez…

Corrélations

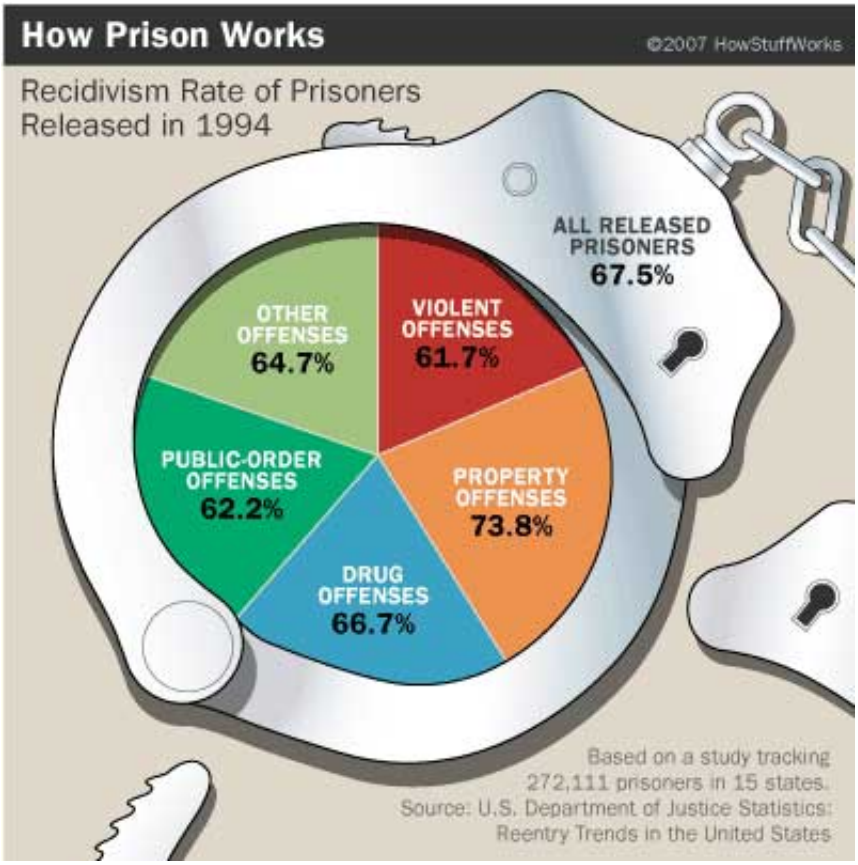




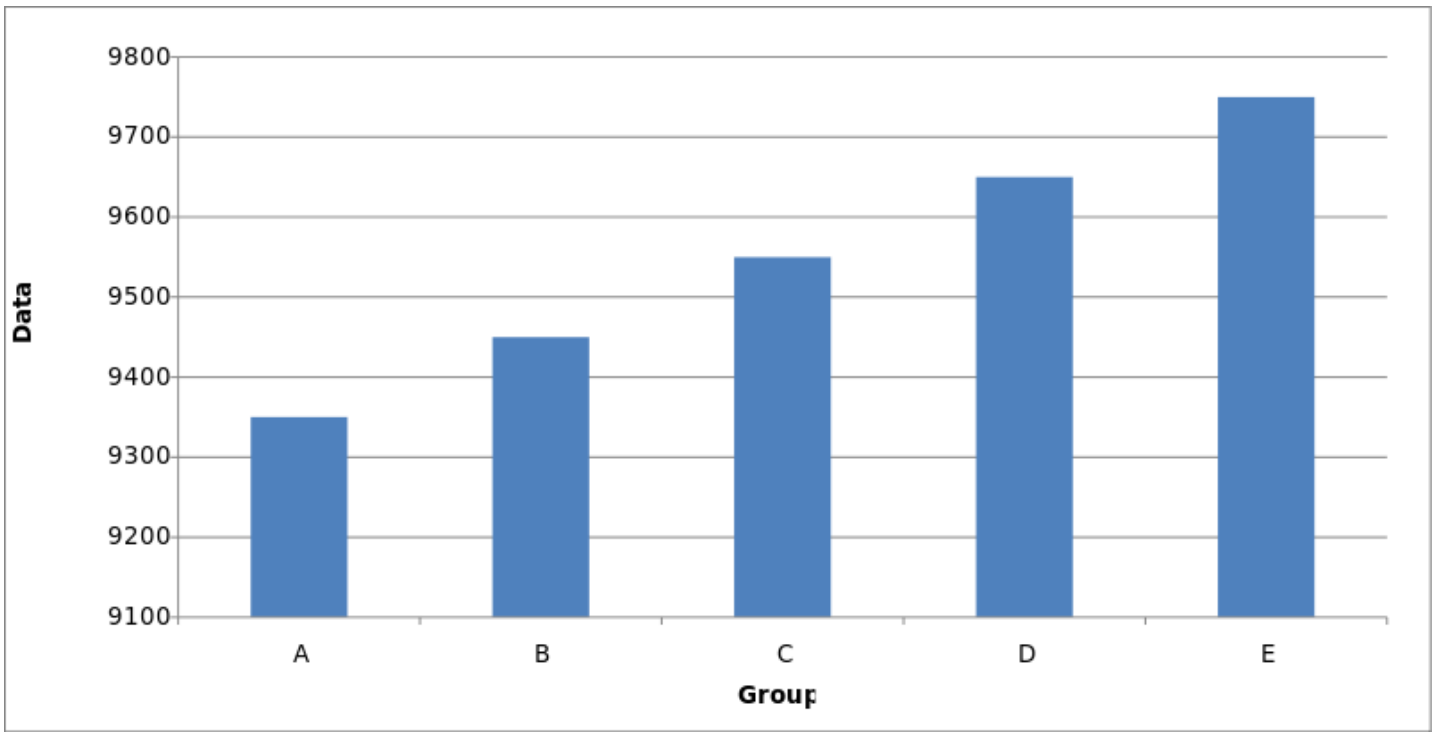
Vous en pensez quoi ?

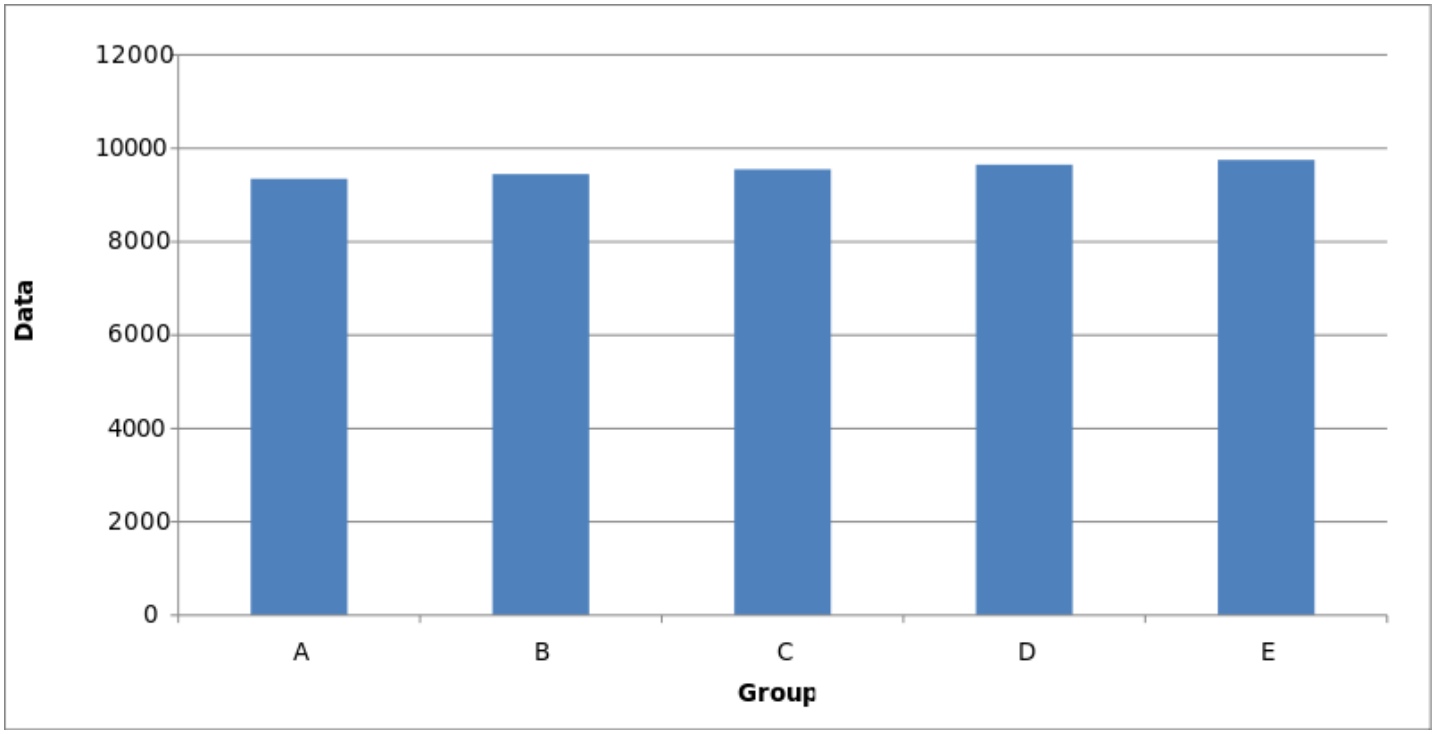
Prison

Qu'est-ce qui ne va pas dans la figure suivante ?

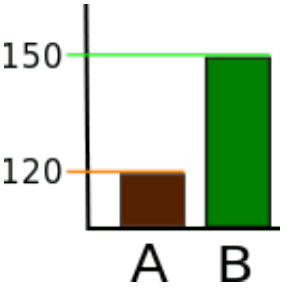


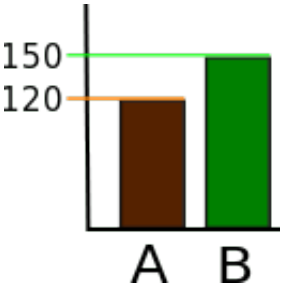
Étendue des axes

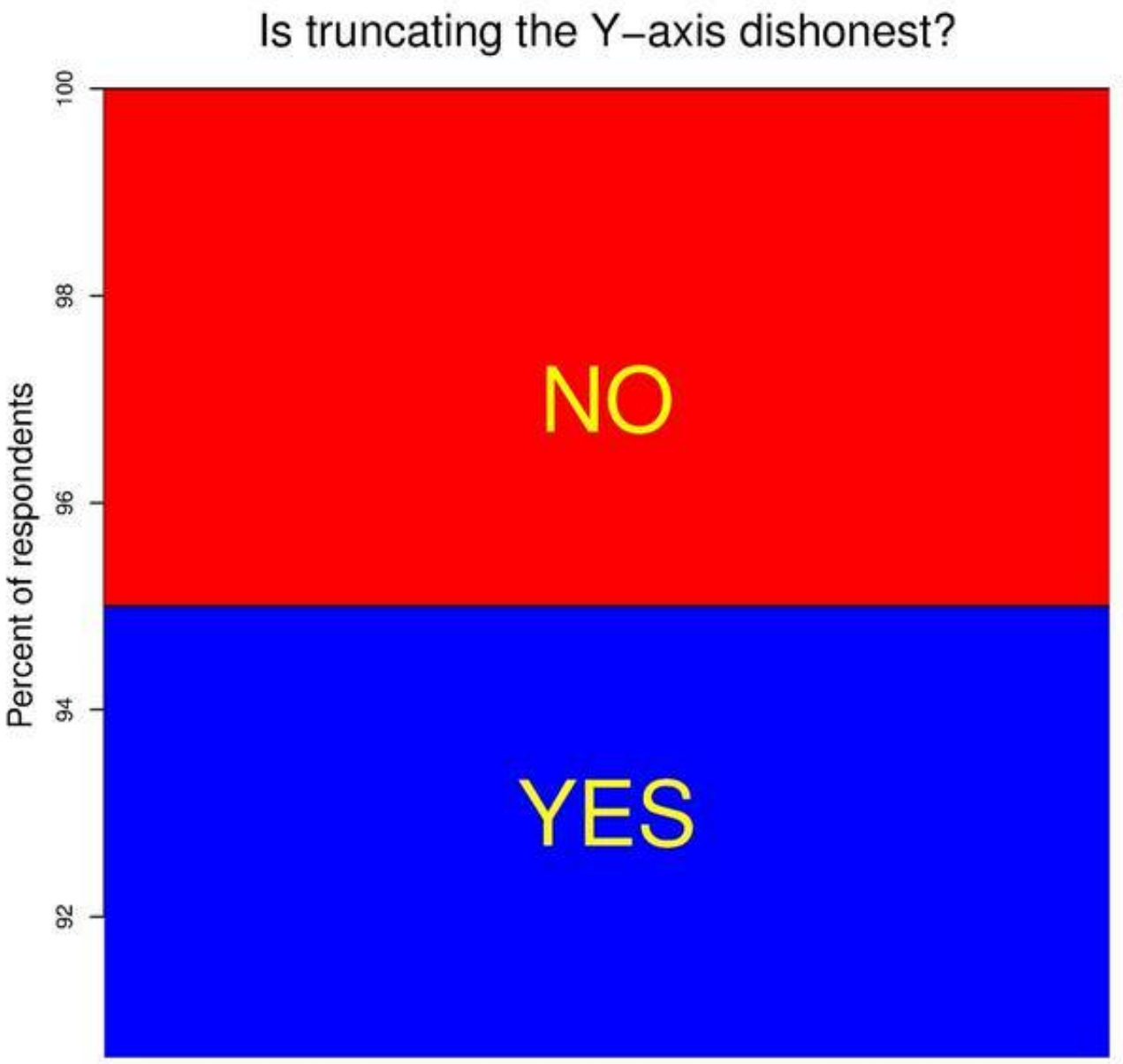




Comparez de manière critique les deux graphiques précédents. Aidez-vous des schémas ci-dessous pour étayer votre explication.







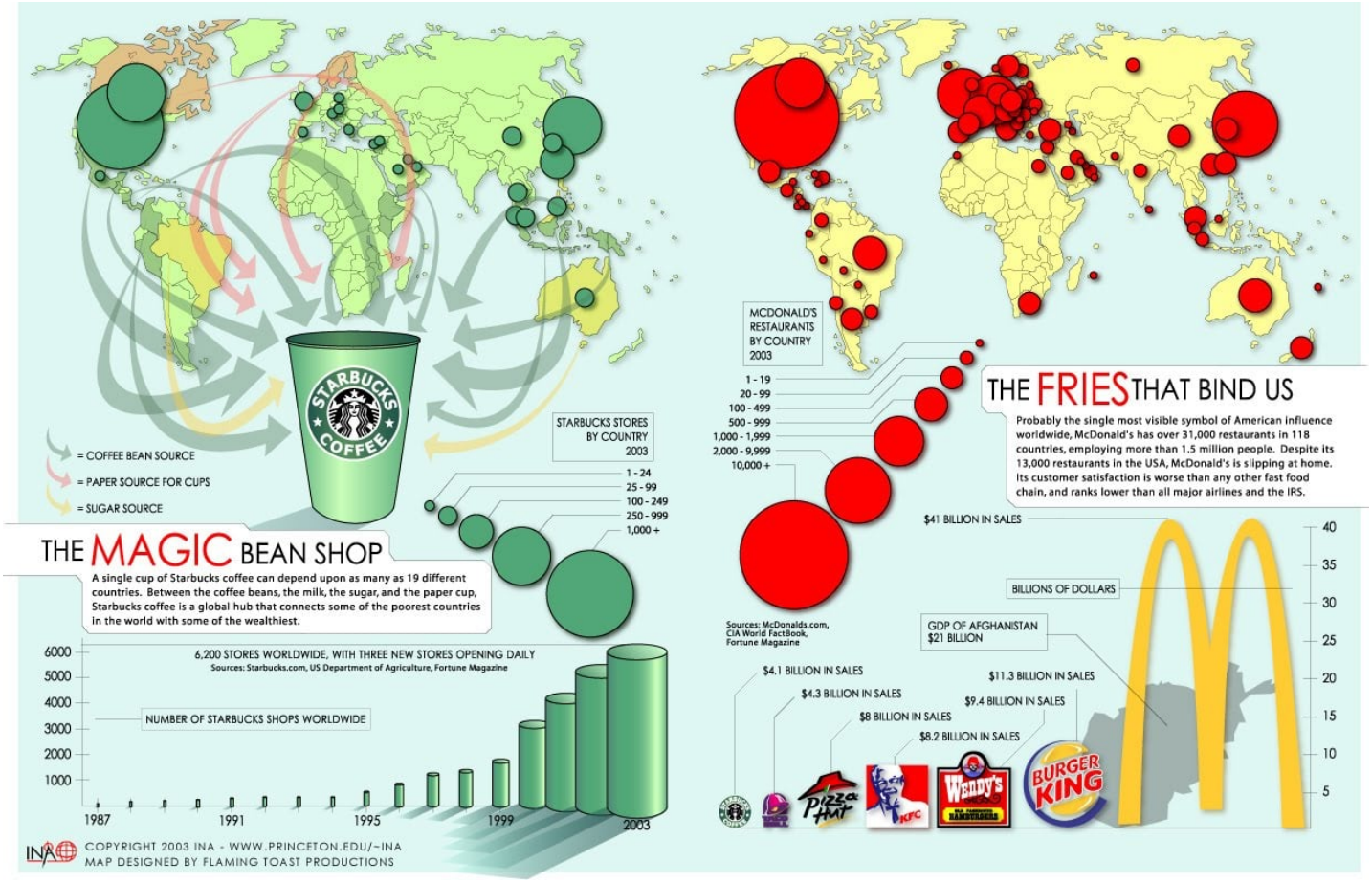
Travaux d'artistes ?

Que pensez-vous des trois figures suivantes ?


HOMICIDES
PRISONERS IN CUSTODY
HOMICIDES[1]
PER 100,000 POPULATION
Chile
India
Indonesia
Japan
Mexico
South Korea
United Kingdom
West Germany
United States
INCARCERATIONS[2]
PER 100,000 POPULATION
Canada
France
India
Japan
Malaysia
Sweden
United Kingdom
Venezuela
United States
Crime
(PER 100,000 POPULATION)
HOMICIDES
PRISONERS IN CUSTODY[3]
The U.S. has proportionately more prisoners than other countries—and a high murder rate.
[1]Latest available figures.
[2]Admits to federal or state custody with sentences of one year or more.
[3]Number of defendants sentenced to any term during the year. Latest available figures.
Sources: Bureau of Justice Statistics; National Center for Health Statistics; Interpol; The Sentencing Project; Rawlins Corp.
12
10
8
6
4
2
0
300
250
200
150
100
50
0
1900
1910
1920
1930
1940
1950
1960
1970
1980
1991
308
Forbes ■ September 14, 1992

I.C.-WIPO, Geneva: EUROPAIMPERATOR.com;
Trading Desk-Futures, kingofamerica.com
c/o SMALAND LEGATIO
#266 611 Pennsylvania Ave SE
Washington, DC 20003-4303
1.2020.777.2642.2453

L.T. Stoc & RSI, etc., Bullish Divergence shows ultimate trend change if not at first Intermediate term challenge. Shorter terms indicatore confirm present wedge predicament/workout mode while the Parabolic also implies its own turn signaled at I.T. high test/breakout while on its S.T. change I.T. support numbers will be generated for next low cycle to resumption of Long term trend establishment, specifically, Greenback/"Dollar..."



Chauve-souris

Un biologiste étudie une chauve-souris insectivore naine. Il trouve dans la littérature que la biomasse totale de cette chauve-souris varie de 0,23 à 1,95 kg/ha dans les forêts recensées. Afin de calculer l'abondance de ces populations de chauve-souris, il détermine le poids moyen d'un individu comme étant (moyenne ± écart type) 55 ± 13 mg (n = 45). Il utilise ces données pour comparer les populations de chauve-souris aux autres animaux présents dans cette forêt. Il en conclut que la population de chauve-souris dans ces forêts est très nettement supérieure à celle des oiseaux et équivalente à celle des insectes. Ce résultat est inattendu et permet de considérer cette chauve-souris comme espèce clé dans la chaîne trophique, alors que son effet a toujours été négligé auparavant, tant elle est discrète et passe inaperçu la plupart du temps.

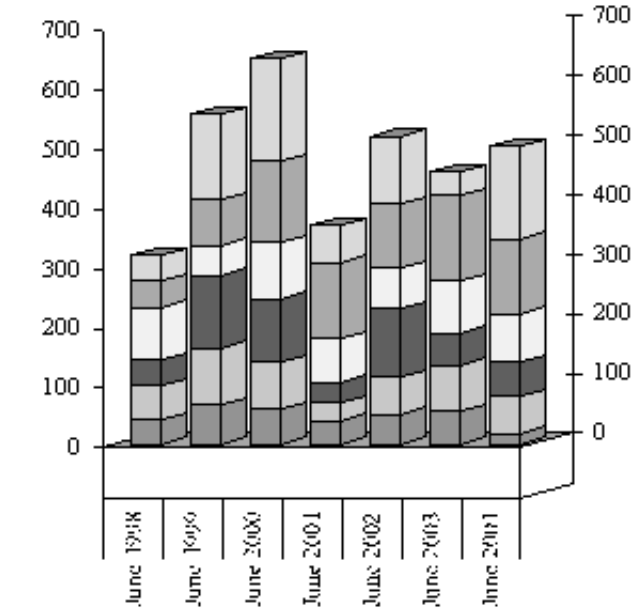
Vous travaillez aussi sur les chaînes trophiques de ces mêmes forêts. Comment réagissez-vous à la lecture de ce rapport ? Que faites-vous ensuite ?

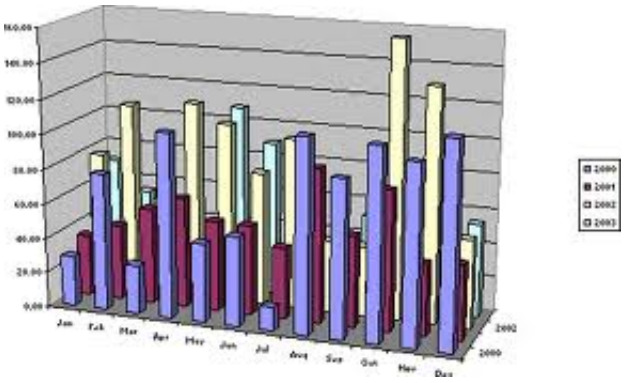
Patinage

Que se passe-t-il si tout le mode respecte cette consigne (considérant qu'il est impossible que tous les patineurs aient exactement la même vitesse) ?

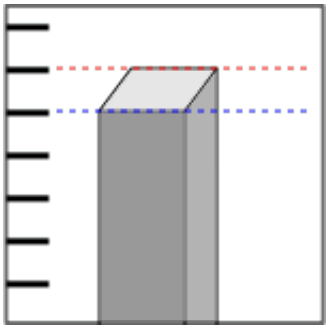
Pseudo-perspective

Que pensez-vous de ces graphiques ?





Aidez-vous du schéma suivant pour expliquer ce qui ne va pas…

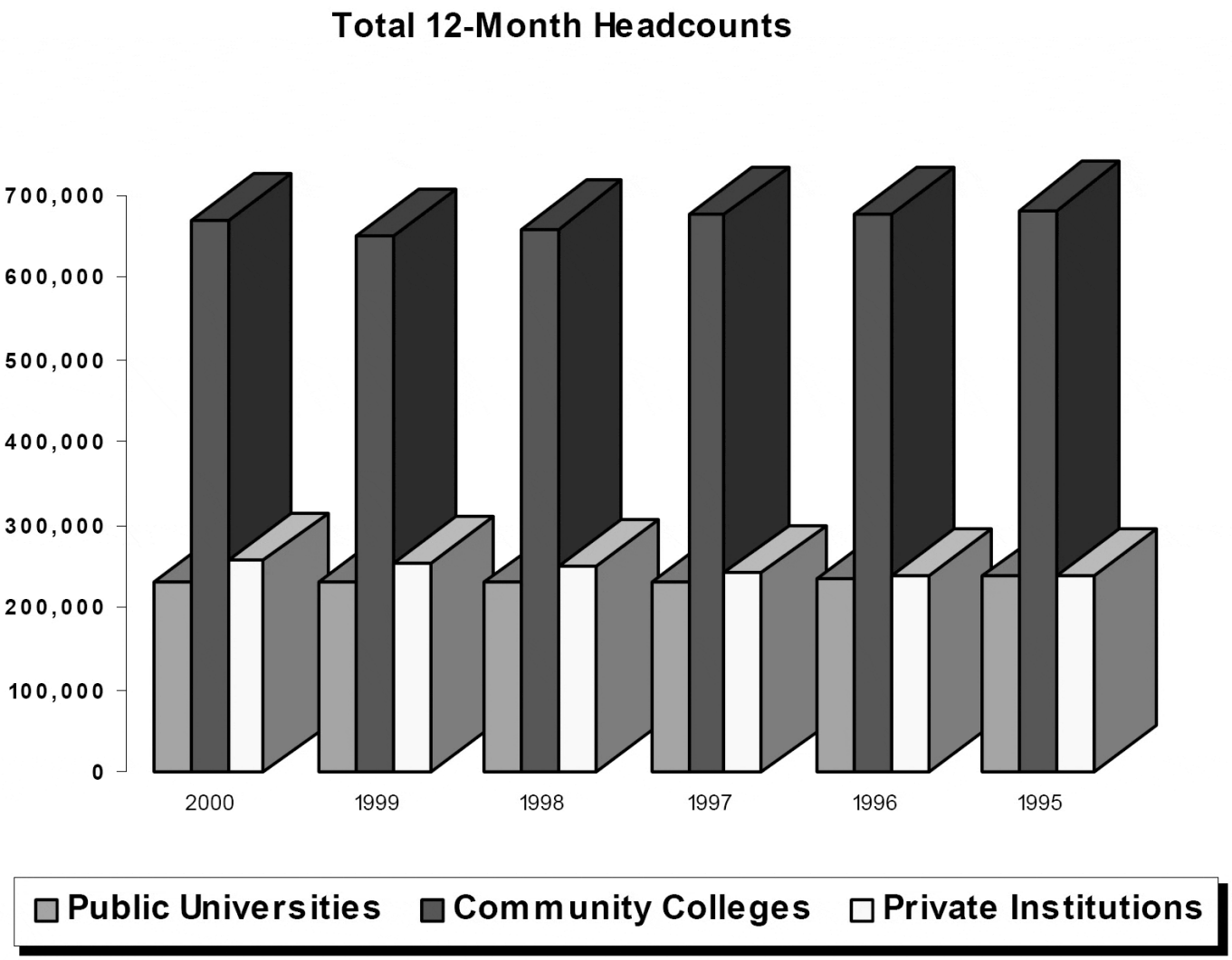


Homme moyen

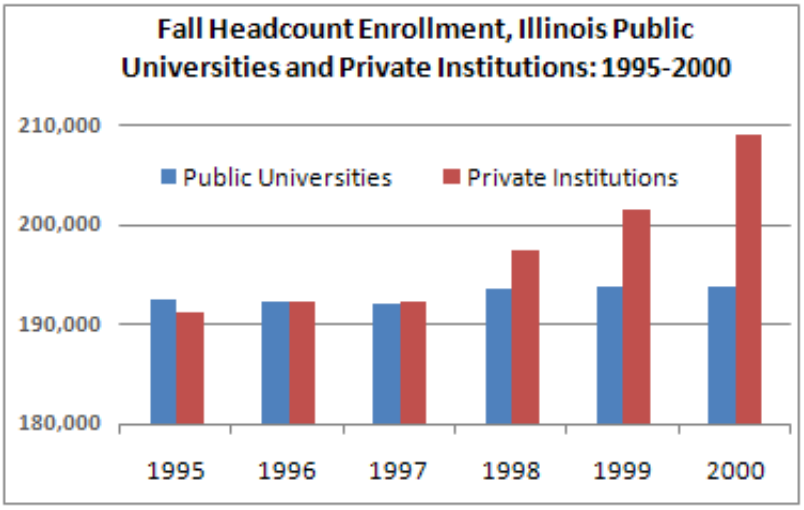
Le magazine “Men’s Health” a publié des statistiques qui décrivent l’“homme moyen”. Celui-ci a 34,4 ans, pèse 79,4kg, mesure 177,8cm, dort 6,9 heures chaque nuit, boit 3,3 tasses de café par jour et consomme 1,2 boisson alcoolique quotidiennement.

Sachant que toutes les distributions sont unimodales, donc que les valeurs moyennes correspondent toutes à des observations effectivement mesurées en grands nombres (identiques ou très proches) sur des hommes réels, ce portrait-robot de l'"homme moyen" décrit-il effectivement un grand nombre d'individus réellement existants ? Justifiez. Qu'en serait-il de l'"homme médian" ?

Public ou privé ?

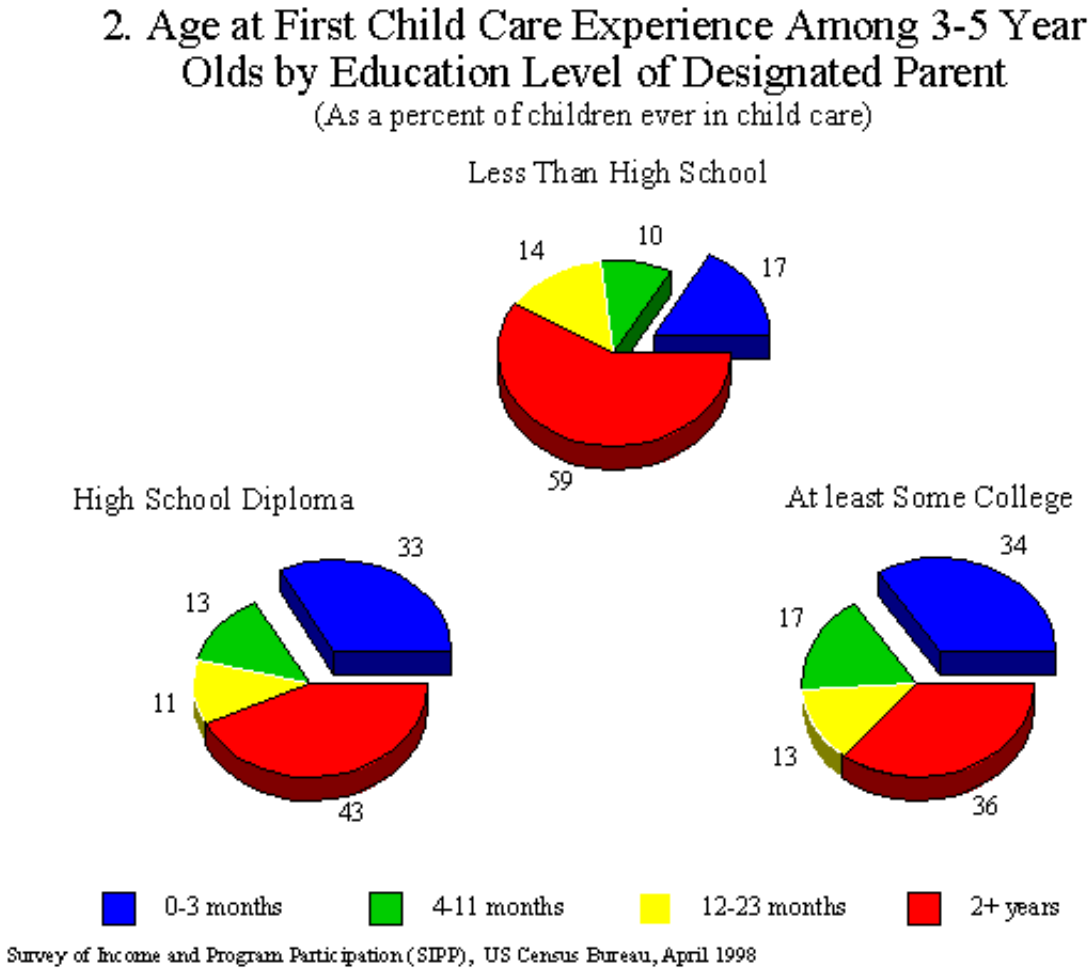


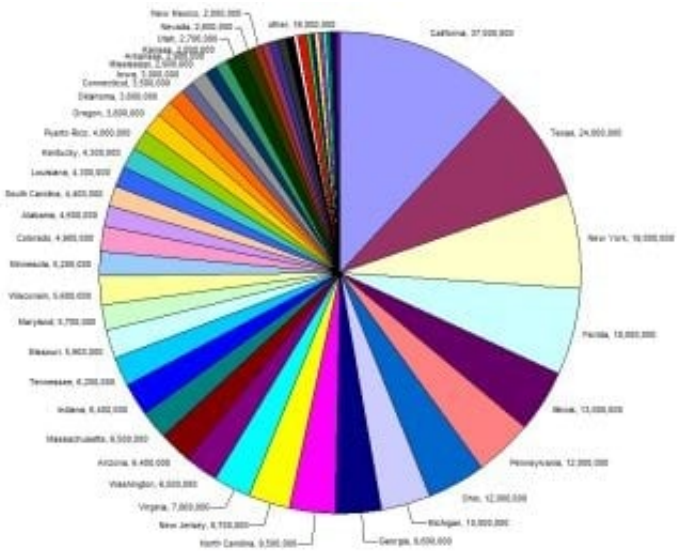
Observez bien le graphique ci-dessus... Ensuite, regardez celui ci-dessous qui est réalisé à l'aide des mêmes données. Commentez...

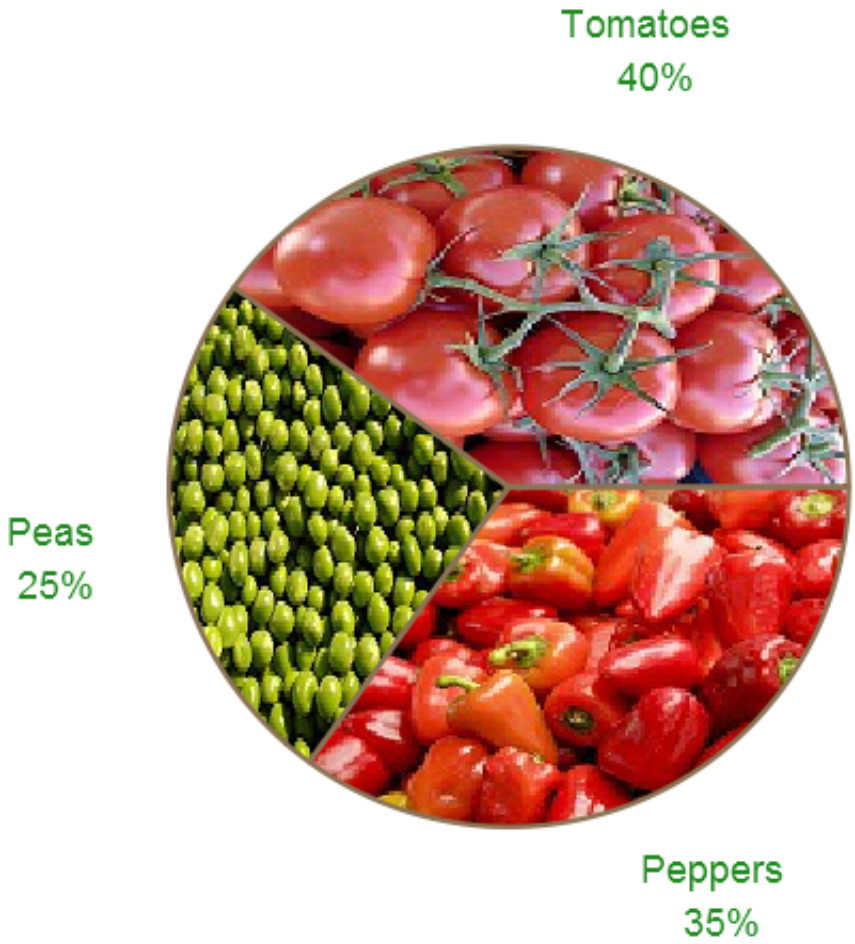


Camemberts, tartes et cie

Que pensez-vous de ces graphiques ?






Tomatoes
40%
Peas
25%
Peppers
35%

Espérance de vie

L'espérance de vie est une donnée statistique qui permet de connaître la durée de vie moyenne qu'on peut espérer atteindre à un moment donné pour une nation donnée. Cette statistique est calculée et publiée par de nombreux organismes, incluant l'OMS. Les statistiques indiquent que l'espérance de vie des hommes dans nos pays est de 75,5 ans, et des femmes de 83,5 ans.

Calculez le temps que vous pouvez espérer encore vivre en fonction de votre âge. Que pensez-vous de ce calcul ?

Femmes au travail

Considérez les deux graphiques suivants qui sont censés représenter la même information (les mêmes données sont utilisées). Comparez-les de manière critique.

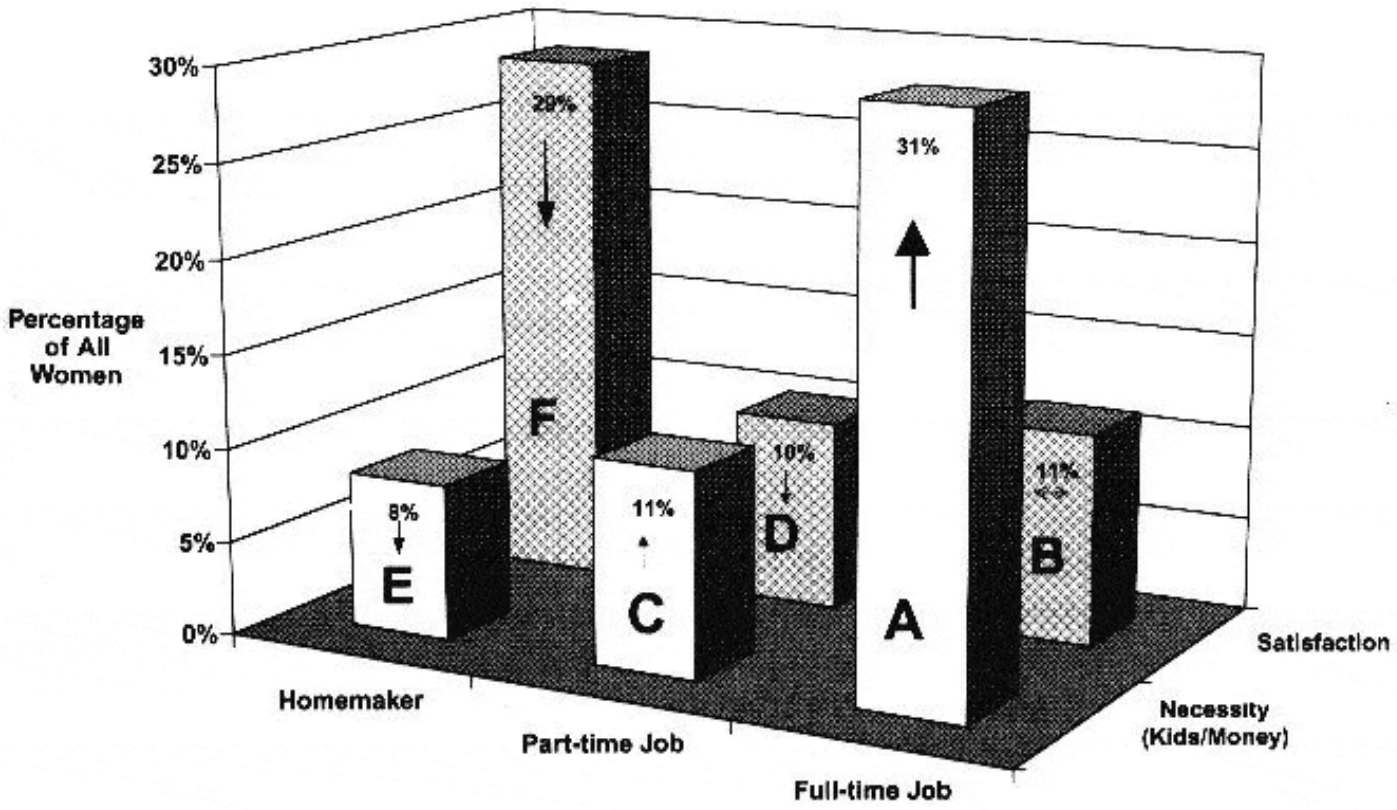
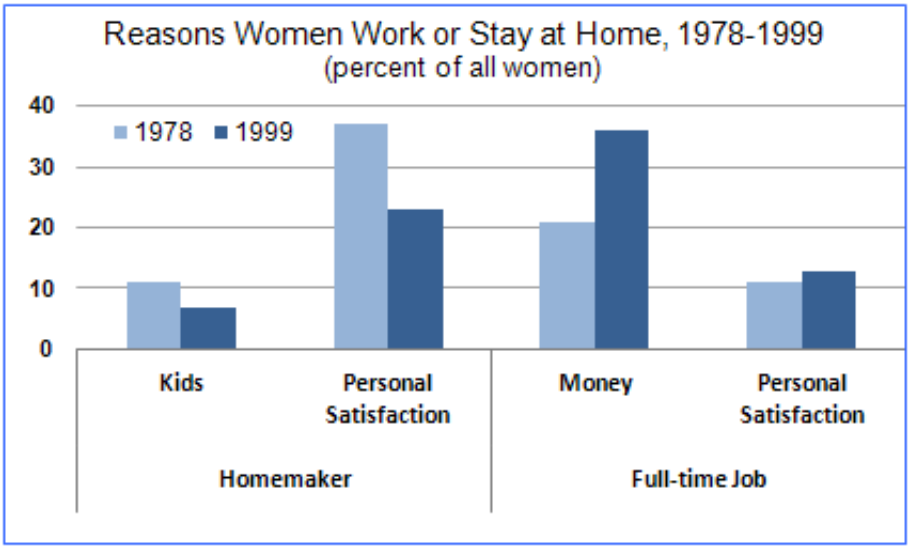


Figure 47: Working by Choice and by Necessity Among American Women, 1978–1999



Moules

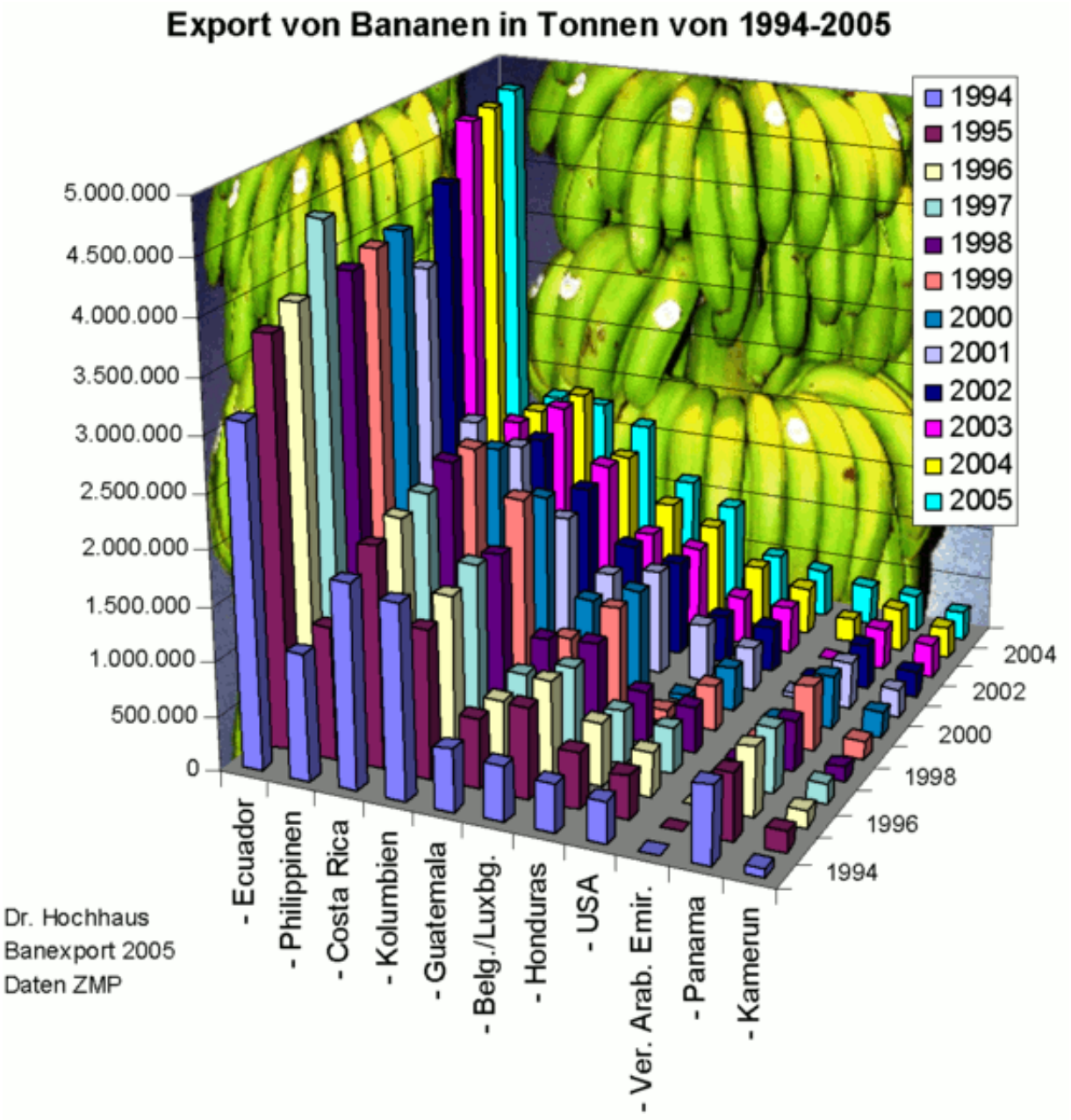
Un scientifique mesure la stabilité de la membrane lysosomale (indice de stress des cellules utilisé en écotoxicologie : on sait que les polluants étudiés tendent à déstabiliser la membrane des lysosomes) chez la moule *Mytilus edulis* en Mer du Nord. Deux régions sont

comparées : la pleine mer (A), et l'embouchure de l'Escaut dans sa partie considérée comme la plus polluée (B). Cinq moules sont prélevées aléatoirement sur les deux sites, et dix mesures sont réalisées sur chaque individu. Le scientifique conclut à une stabilité lysosomale significative plus faible au seuil alpha de 5% dans le site B (test *t* de Student non apparié et unilatéral à gauche, t = -6,5, ddl = 49, valeur $p << 0.001$).

Que pensez-vous de cette étude ?

République bananière ?

Que pensez-vous du graphique suivant ?

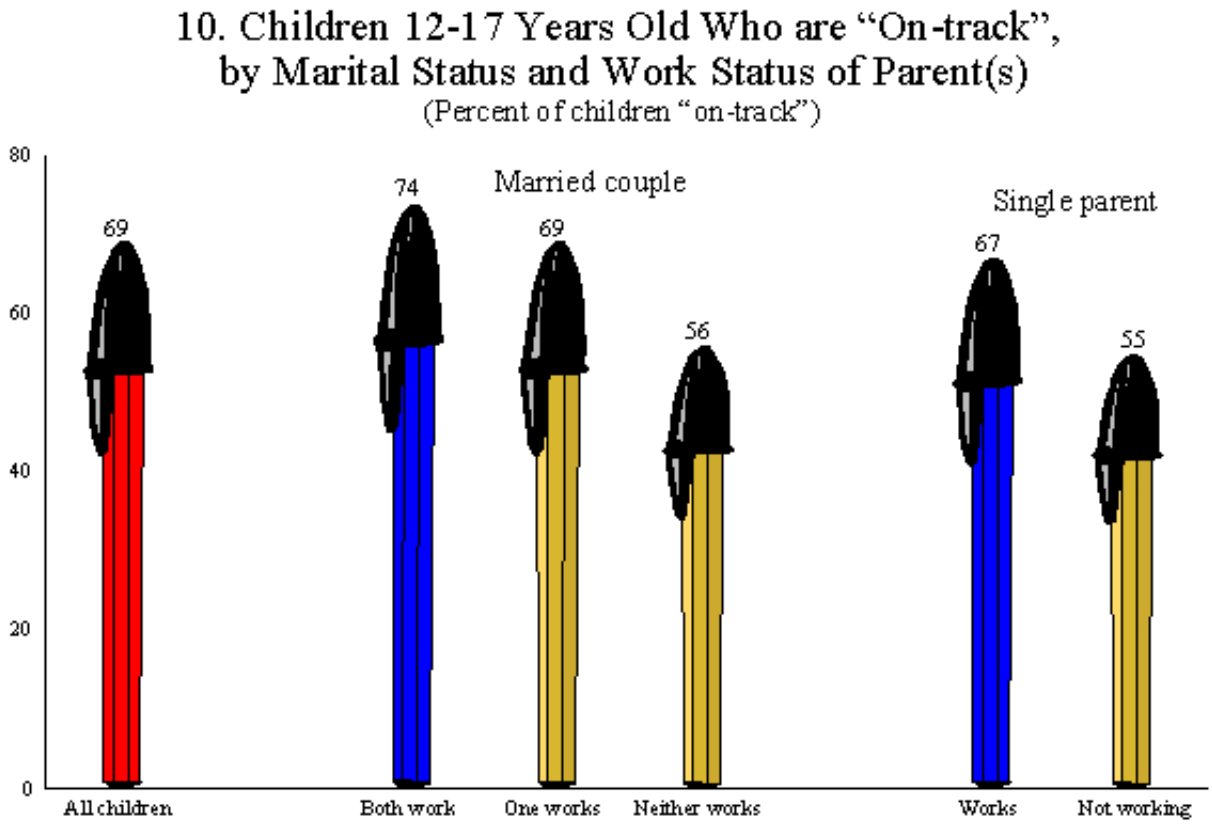


Euro manquant

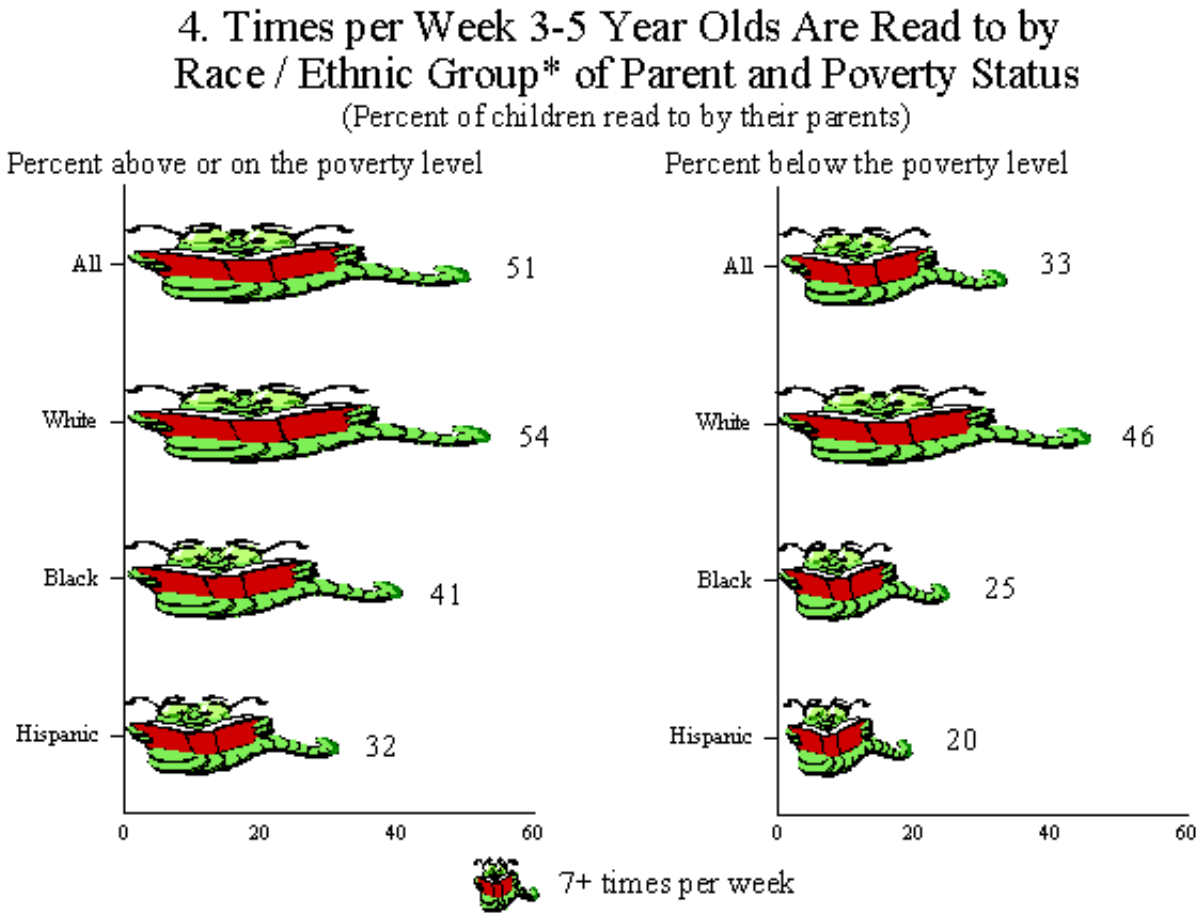
Trois clients dans un restaurant payent leur repas : 30€ (10€ par personne). Le serveur se rend compte qu'en fait leur repas n'a coûté que 25€ en tout. Comme il ne pourra diviser les 5€ à rendre en trois facilement, il décide de garder 2€ dans sa poche et rend 1€ à chaque client. Donc, chaque client a payé 10 – 1 = 9€, soit un total de 27€. Avec les 2€ que le serveur a gardés dans sa poche, cela fait 29€. Alors, où est passé l'euro manquant par rapport aux 30€ payés initialement ?

Réfléchissez et dénoncez l'erreur de raisonnement dans le récit précédent.

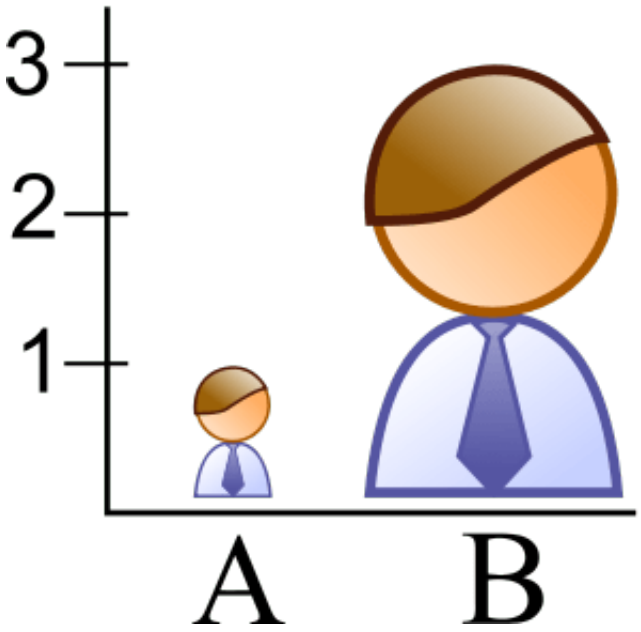
Stylos et vers verts ?

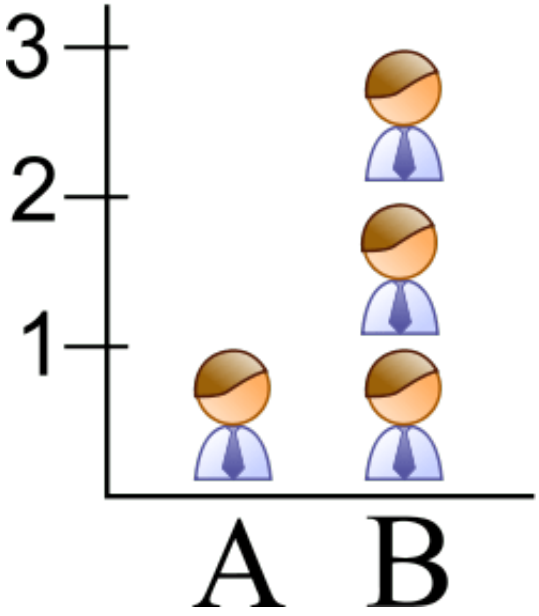


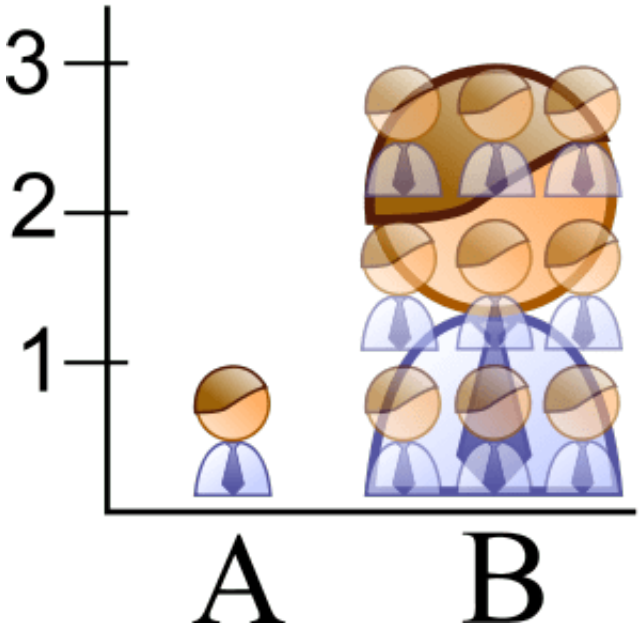
Survey of Income and Program Participation (SIPP), US Census Bureau, April 1998



Que pensez-vous de ces graphiques ? Vous pouvez vous aider des schémas suivants pour étayer votre réponse.





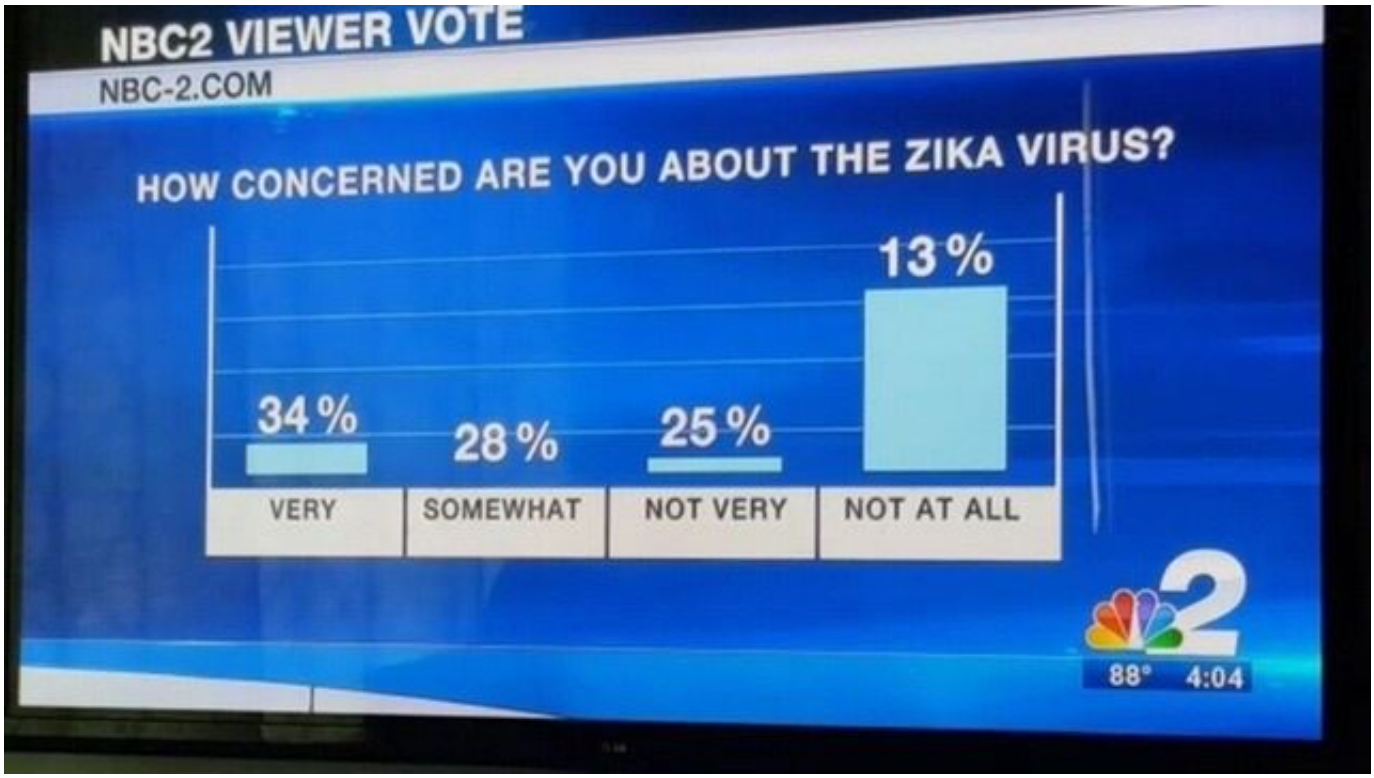


Insecticides

Un chercheur dans une industrie chimique s'intéresse à l'effet d'un nouvel insecticide à effet progressif. Il teste son produit sur des drosophiles et observe une mortalité de 10% par jour, et ce, quel que soit le moment où il effectue les mesures après avoir mis les mouches en contact avec l'insecticide. Il en conclut qu'il faut 10 jours pour tuer toutes les mouches. Ce résultat est meilleur que le produit du concurrent, car ce dernier tue 80% des mouches sur la même durée de 10 jours.

Que pensez-vous de la façon dont cette expérience a été menée et de ses conclusions ?

Virus Zika



Cela ne s'invente pas !

Lotto

Par le plus grand des hasards, le numéro 8 est sorti 6 fois en 7 tirages successifs du lotto. Sachant qu'une vérification de ce que ce numéro n'a pas plus de chances que les autres d'être tiré au sort, vous ne manquerez pas de constater en bon statisticien(ne) que le numéro 8 est très nettement surreprésenté dans les tirages.

La prochaine fois que vous remplirez votre grille de lotto, jouerez-vous le numéro 8 ? Pourquoi ?

Vous est-il arrivé de jouer la suite 1, 2, 3, 4, 5, 6, 7, 8 au lotto (ou rempliriez-vous une grille avec ces nombres si vous deviez y jouer) ? Pourquoi ?

Références

Allaire, JJ, Yihui Xie, Christophe Dervieux, Jonathan McPherson, Javier Luraschi, Kevin Ushey, Aron Atkins, et al. 2024. *Rmarkdown: Dynamic Documents for r*. https://github.com/rstudio/rmarkdown.

R Core Team. 2025. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.