

Sur le chemin de l'enf-R

Une introduction au monde merveilleux de R

Julien Martin



*Faites ce que vous pensez être intéressant,
faites quelque chose que vous pensez être utile et amusant,
parce que sinon, vous ne le ferez pas bien de toute façon.*

—Brian W. Kernighan

Table des matières

Préface	ii
L'objectif de ce livre	ii
Livre multilingue	ii
Comment utiliser ce livre	iii
Qui sommes-nous ?	iv
Contribution au livre	iv
Remerciements	v
Crédits d'image	v
Licence	vi
Citer le livre	vi
Lecture associée au cours	vii
Collant Hex	ix
I. Utiliser R	1
1. Pour commencer	2
Quelques conseils sur R	3
1.1. Installation	3
1.1.1. Installation de R	3
1.1.2. Installation d'un IDE	4
1.2. Orientation dans l'IDE	6
1.2.1. RStudio	6
1.2.2. VSCode	11
1.3. Répertoires de travail	13
1.4. Structure du répertoire	15
1.5. Organisation des projets	17
1.5.1. RStudio	18
1.5.2. VSCode	20
1.6. Nom des fichiers	21
1.7. Documentation du script	22
1.8. Guide de style R	24
1.9. Sauvegarde des projets	26
1.10. Citer R et les paquets R	26
2. Quelques notions de base sur R	29
2.1. Considérations importantes	29
2.2. Première étape dans la console	30
2.3. Objets en R	32
2.3.1. Création d'objets	32
2.3.2. Nommer les objets	35
2.4. Utilisation de fonctions dans R	36

2.5.	Travailler avec des vecteurs	40
2.5.1.	Extraction d'éléments	40
2.5.2.	Remplacement d'éléments	43
2.5.3.	Ordonner les éléments	44
2.5.4.	Vectorisation	46
2.5.5.	Données manquantes	48
2.6.	Obtenir de l'aide	49
2.6.1.	Aide R	49
2.6.2.	Autres sources d'aide	53
2.7.	Sauvegarder des données dans R	54
2.8.	Paquets R	55
2.8.1.	Utilisation des paquets	56
2.8.2.	Installation des paquets R	57
3.	Données	59
3.1.	Types de données	59
3.2.	Structures de données	62
3.2.1.	Scalaires et vecteurs	62
3.2.2.	Matrices et tableaux (array)	63
3.2.3.	Listes	66
3.2.4.	Jeu de données	68
3.3.	Importer des données	71
3.3.1.	Enregistrement de fichiers à importer	72
3.3.2.	Fonctions d'importation	73
3.3.3.	Frustrations courantes liées à l'importation	77
3.3.4.	Autres options d'importation	78
3.4.	Manipuler des jeux de données	79
3.4.1.	Index positionnels	81
3.4.2.	Index logiques	85
3.4.3.	Ordonner un jeu de données	91
3.4.4.	Ajout de colonnes et de lignes	95
3.4.5.	Fusionner des jeux de données	99
3.4.6.	Remodeler des jeux de données	103
3.5.	Introduction au tidyverse (<i>L'univers ordonné</i>)	107
3.6.	Résumer des jeux de données	107
3.7.	Exporter des données	115
3.7.1.	Fonctions d'exportation	116
3.7.2.	Autres fonctions d'exportation	117
4.	Figures	119
4.1.	Graphiques de base simples en R	121
4.2.	ggplot2 	125
4.3.	Graphiques simples	127
4.3.1.	Diagrammes de dispersion (nuages de points, Scatter plots)	127
4.3.2.	Histogrammes	128
4.3.3.	Boîtes à moustache (Box plots)	133
4.3.4.	Diagrammes en violon (Violin plots)	139
4.3.5.	Diagramme en pointillés (Dot charts)	141
4.3.6.	Diagrammes en paires	144
4.3.7.	Graphique de conditionnement (Coplots)	147

4.3.8. Résumé de la fonction du graphique	150
4.4. Graphiques multiples	151
4.4.1. R de base	151
4.4.2. ggplot2 	152
4.5. Personnalisation de ggplots	155
4.6. Exportation des parcelles	155
5. Programmation	158
5.1. Regarder derrière le rideau	158
5.2. Fonctions en R	161
5.3. Structures conditionnelles	168
5.4. Combinaison d'opérateurs logiques	173
5.5. Boucles	175
5.5.1. Boucle “For” (pour)	175
5.5.2. Boucle “While” (tant que)	180
5.5.3. Quand utiliser une boucle ?	181
5.5.4. Si on n'utilise pas une boucle, alors quoi ?	182
6. Rapports reproductibles avec Quarto	185
6.1. Qu'est-ce que R markdown / Quarto ?	185
6.1.1. R Markdown	185
6.1.2. Quarto ?	186
6.2. Pourquoi utiliser Quarto ?	186
6.3. Commencer avec Quarto	189
6.3.1. Installation de la solution	189
6.3.2. Créez un document Quarto, .qmd	189
6.4. Anatomie du document Quarto (.qmd)	193
6.4.1. En-tête YAML	193
6.4.2. Texte formaté	194
6.4.1. Morceaux de code	200
6.4.2. Code R en ligne	206
6.4.3. Images et photos	207
6.4.4. Les chiffres	209
6.4.5. Tableaux	213
6.4.6. Références croisées	218
6.4.7. Citations et bibliographie	222
6.5. Quelques conseils et astuces	224
6.6. Informations complémentaires	226
6.7. Pratique	226
6.7.1. Le contexte	227
6.7.2. Questions	227
Exemple de résultats	231
7. Contrôle de version avec Git et GitHub	233
7.1. Qu'est-ce que le contrôle de version ?	234
7.2. Pourquoi utiliser le contrôle de version ?	234
7.3. Qu'est-ce que Git et GitHub ?	234
7.4. Pour commencer	237
7.4.1. Installer Git	237
7.4.2. Configurer Git	238

7.4.3. Configurer RStudio	238
7.4.4. Configurer VSCode	239
7.4.5. Créer un compte GitHub	239
7.5. Mise en place d'un projet	240
7.5.1. dans RStudio	240
7.5.2. Option 1 - GitHub d'abord	240
7.5.3. Option 2 - RStudio d'abord	243
7.5.4. dans VSCode	250
7.6. Utiliser Git avec RStudio	250
7.6.1. Suivi des modifications	256
7.6.2. Historique des engagements	260
7.6.3. Annulation des modifications	262
7.7. Utiliser Git avec VSCode	271
Suivi des modifications	273
Historique des modifications	273
Annulation des modifications	273
7.8. Collaborer avec Git	273
7.9. Conseils Git	274
7.10. Autres ressources	275
7.11. Pratique	276
7.11.1. Contexte	276
7.11.2. Informations sur les données	276
7.11.3. Questions	278
7.11.4. Solution	279
II. Principes de statistique	281
8. Analyse de puissance avec R et G*Power	282
8.1. La théorie	282
8.1.1. Qu'est-ce que la puissance?	282
8.1.2. Pourquoi faire une analyse de puissance?	282
8.1.3. Facteurs qui affectent la puissance	283
8.2. Qu'est ce que G*Power?	284
8.3. Comment utiliser G*Power	284
8.3.1. Principe général	284
8.3.2. Types d'analyses de puissance disponibles	284
8.3.3. Comment calculer la taille d'effet ?	285
8.4. Puissance pour un test de t comparant deux moyennes	286
8.4.1. Analyse post-hoc - Calculer la puissance du test	287
8.4.2. Analyse à priori - Calculer la taille de l'effectif à échantillonner	295
8.4.3. Analyse de sensibilité - Calculer la taille d'effet détectable	297
8.5. Points à retenir	300
III. Modèles linéaires	301
9. Corrélation et régression linéaire simple	302
9.1. Paquets R et données requises	302
9.2. Diagrammes de dispersion	303

9.3. Transformations de données et coefficient de corrélation	306
9.4. Matrices de corrélations et correction de Bonferroni	309
9.5. Corrélations non paramétriques: r de Spearman et τ de Kendall	310
9.6. Régression linéaire simple	312
9.6.1. Vérifier les conditions d'application de la régression	315
9.6.2. Tests formels des conditions d'application pour la régression	318
9.7. Transformation des données en régression	320
9.8. Traitement des valeurs extrêmes	324
9.9. Quantifier la taille d'effet et analyse de puissance en régression	327
9.9.1. Puissance de détecter une pente donnée	328
9.9.2. Effectif requis pour atteindre une puissance désirée (test A-priori)	331
9.10. Bootstrap en régression simple avec R	333
10. Comparaison de deux échantillons	337
10.1. Paquets R et données requises	337
10.2. Examen visuel des données	338
10.3. Comparer les moyennes de deux échantillons indépendants	341
10.4. Bootstrap et tests de permutation pour comparer deux moyennes	345
10.4.1. Bootstrap	345
10.4.2. Permutation	347
10.5. Comparer les moyennes de deux échantillons appariés	349
10.6. Références	355
11. ANOVA à un critère de classification	356
11.1. Paquets et données requises pour le labo	356
11.2. ANOVA à un critère de classification et comparaisons multiples	357
11.2.1. Visualiser les données	357
11.2.2. Vérifier les conditions d'application de l'ANOVA paramétrique	360
11.2.3. Faire l'ANOVA	362
11.2.4. Les comparaisons multiples	363
11.3. Transformations de données et ANOVA non-paramétrique	369
11.4. Examen des valeurs extrêmes	371
11.5. Test de permutation	372
12. ANOVA à critères multiples : plans factoriels et hiérarchiques	375
12.1. Paquets et données requises pour le labo	375
12.2. Plan factoriel à deux facteurs de classification et réplication	376
12.2.1. ANOVA à effets fixes (Modèle I)	376
12.2.2. ANOVA à effets mixtes (Modèle III)	385
12.3. Plan factoriel à 2 facteurs de classification sans réplication	387
12.4. Plans hiérarchiques	390
12.5. ANOVA non paramétrique avec 2 facteurs de classification	394
12.6. Comparaisons multiples	397
12.7. Test de permutation pour l'ANOVA à 2 facteurs de classification	403
12.8. Bootstrap pour l'ANOVA à 2 facteurs de classification	406
13. Régression multiple	408
13.1. Paquets et données requises pour le labo	408
13.2. Conseils généraux	409
13.3. Premières régressions multiples	409
13.4. Régression multiple pas-à-pas (stepwise)	417

13.5. Déetecter la multicolinéarité	421
13.6. Régression polynomiale	422
13.7. Vérifier les conditions d'application de modèles de régression multiple	428
13.8. Visualiser la taille d'effet	430
13.9. Tester la présence d'interactions	432
13.10 Recherche du meilleur modèle fondée sur la théorie de l'information	438
13.11 Bootstrap et régression multiple	441
13.12 Test de permutation	444
14. ANCOVA et modèle linéaire général	446
14.1. Paquets et données requises pour le labo	446
14.2. Modèle linéaire général	447
14.3. ANCOVA	447
14.4. Homogénéité des pentes	447
14.4.1. Cas 1 - La taille en fonction de l'âge (exemple avec pente commune)	448
14.4.2. Cas 2 - Taille en fonction de l'âge (exemple avec des pentes différentes)	452
14.5. Le modèle d'ANCOVA	456
14.6. Comparer l'ajustement de modèles	461
14.7. Bootstrap	463
14.8. Test de Permutation	464
IV. Modèles linéaires généralisés	466
15. Analyse de données de fréquence: Tableaux de contingence, modèles log-linéaires et régression de Poisson	467
15.1. Paquets et données requises pour le labo	467
15.2. Organisation des données: 3 formats	468
15.3. Visualiser graphiquement les tableaux de contingence et test d'indépendance	471
15.4. Régression de Poisson: une alternative au test de Chi carré pour les tableaux de contingence	475
15.5. Tester une hypothèse extrinsèque	478
15.6. Régression de Poisson pour l'analyse de tableaux de contingence à plusieurs critères	479
15.7. Exercice	481
V. Modèles mixtes	488
VI. Modèles additifs généralisés	489
VII. Analyses multivariées	490
VIII Approche Bayesienne	491
Références	492
Paquets R	492
Bibliographie	493

Annexes	497
A. Données utilisées dans le livre	497
A.1. Fichier compressé tout-inclus	497
A.2. Tous les fichiers séparés	497
A.3. Code R et fonctions utilisées dans les diapositives	498

Préface

Avertissement

Première version en cours de traduction. Pas encore revue par un humain

Certaine section non encore développée

L'objectif de ce livre

L'objectif de ce livre est double :

1. Vous présenter R, un environnement interactif puissant et flexible pour le calcul et la recherche statistiques.
2. Vous présenter (ou vous familiariser à nouveau) avec l'analyse statistique effectuée dans R.

R n'est pas difficile à apprendre en soi, mais comme pour toute nouvelle langue (parlée ou informatique), la courbe d'apprentissage initiale peut être raide et quelque peu intimidante. L'objectif n'est pas de tout couvrir (ni avec R, ni avec les statistiques), mais simplement de vous aider à franchir le cap (potentiellement plus rapidement) et à vous fournir les compétences de base (et la confiance !) nécessaires pour commencer votre propre voyage avec R et avec des analyses spécifiques.

Livre multilingue

Le livre est fourni comme un livre multilingue qui brise la barrière de la langue et permet potentiellement de faciliter l'apprentissage de R et de son environnement principalement anglophone. Nous sommes toujours à la recherche de bénévoles pour nous aider à développer le livre et à ajouter d'autres langues à la liste qui ne cesse de s'allonger . N'hésite pas à Nous contacter si tu veux nous aider

Sur la page web, tu peux changer de langue via le  dans la barre de navigation. Après avoir changer de langue, tu peux télécharger le document en pdf ou epub pour cet langue .

Liste des langues :

- anglais (publié mais à peaufiner)
- français (en développement, en attendant que l'anglais soit peaufiné)
- espagnol (un jour peut-être...)
- ... des volontaires pour plus ??

Comment utiliser ce livre

Pour une meilleure expérience, nous te recommandons de lire la version web de ce livre que tu peux trouver à <https://biostats-uottawa.github.io/enfR>.

La version web inclut une barre de navigation incluant des options pour faciliter la lecture , de recherche , pour changer la couleur  et pour suggérer des modifications ou reporter des problèmes . Tu peux aussi télécharger le document  au format pdf ou epub.

Nous utilisons quelques conventions typographiques tout au long de ce livre.

Le code R et la sortie qui en résulte sont présentés dans des blocs de code dans notre livre.

2 + 2

[1] 4

Les fonctions dans le texte sont présentées avec des parenthèses à la fin en utilisant la police de code, c'est-à-dire `mean()` ou `sd()` etc.

Les objets sont représentés à l'aide de la police de code sans les parenthèses, c'est-à-dire `obj1`, `obj2` etc.

Les paquets R dans le texte sont indiqués en utilisant la police de code et suivis de l'icone , exemple `tidyverse` .

Une série d'actions nécessaires pour accéder aux commandes de menu dans RStudio ou VSCode sont identifiées comme suit `File -> New File -> R Script` ce qui se traduit par “clique sur le menu Fichier, puis clique sur Nouveau fichier et sélectionne R Script”.

Lorsque nous faisons référence à **IDE** (**I**ntegrated **D**evelopment **E**nvironment : Logiciel d’Environnement de Développement Intégré) dans la suite du texte, il s’agit de RStudio ou de VScode.

Lorsque nous parlons de **.[Rq]md**, nous entendons par là les documents R markdown (.Rmd) ou Quarto (.qmd) et nous parlerons généralement des documents R markdown en faisant référence à l'un ou l'autre des fichiers .Rmd ou .qmd.

Le manuel tente de mettre en évidence certaines parties du texte à l'aide des encadrés et icônes suivants.

 Exercices

Des choses à faire pour toi

 Solutions

Code R et explications

 Avertissement

Avertissements

 Important

Points importants

 Note

Notes

Qui sommes-nous ?



Julien Martin est professeur à l'Université d'Ottawa en Écologie évolutive. Il a découvert le merveilleux monde R avec la version 1.8.1 et l'enseigne depuis R v2.4.0.

-  uOttawa <https://www.uottawa.ca/faculty-science/professors/julien-martin/>, page du labo <https://juliengamartin.github.io>
-  <https://twitter.com/jgamartin>
-  <https://github.com/juliengamartin>

Contribution au livre

JM: développement et écriture de la première version des chapitres en anglais, mise en page, traduction, maintenance



Augustin Birot est étudiant au doctorat à l'Université d'Ottawa en Écologie évolutive.

- 📌: page du labo <https://juliengamartin.github.io/people/people/>

AB: traduction en français et clarification de tous les chapitres.

Remerciements

La première partie du livre sur l'utilisation de R a commencé comme un fork sur github à partir de l'excellent livre en anglais [An introduction to R](#) de Douglas, Roos, Mancini, Couto et Lusseau (Douglas 2023). Il a été forké le 23 avril 2023 à partir de [Alexd106 github repository](#) puis modifié et mis à jour en fonction de mes propres besoins et de ma perspective d'enseignement de R. Le contenu n'a pas été revu ni approuvé par les développeurs précédents.

Plusieurs parties du livre sont basées sur des manuels de laboratoire pour les cours de biostatistique à l'Université d'Ottawa écrits par Martin, Findlay, Morin et Rundle.

Sites ayant fourni de nombreuses informations pour le livre :

- [dplyr introduction](#)
- [Introduction to gam](#)
- [Intoduction to gams by Noam Ross](#)

Crédits d'image

Les photos, images et captures d'écran sont de Julien Martin sauf lorsque indiqué dans la légende.

L'image de couverture a été générée par [Nightcafe Ai Art generator](#). Le Favicon et l'autocollant hexagonal ont été créés à partir de l'image de couverture.

i Note

plusieurs captures d'écran sont actuellement réalisées par Alex Douglas et sont en train d'être refaites pour pour se conformer à la déclaration précédente

Licence

Je partage cette version modifiée du [livre original](#) sous la licence [Licence Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International](#).



Figure 1.: Licence Creative Commons

Si tu enseignes R, n'hésite pas à utiliser tout ou partie du contenu de ce livre pour aider tes propres élèves. La seule chose que je te demande, c'est de citer la source originale et les auteurs. Si tu trouves ce livre utile ou si tu as des commentaires ou des suggestions, j'aimerais beaucoup que tu me les fasses parvenir (contact info).

Citer le livre

Julien Martin. (2024). Sur le chemin de l'enf-R. Un livre multilingue d'introduction à R. Version: 0.6.0 (2024-12-10).

DOI: [10.5281/zenodo.1380126](https://doi.org/10.5281/zenodo.1380126)

Lecture associée au cours

Table 1.: Course associated reading for biostatistical course at uOttawa

	BioXx58	Bio8940
Chapter		
Utiliser R		
1.-4.	✓✓	😊
5. Programmation		✓✓
6. Rapports reproductibles	✓	✓✓
7. Contrôle de version		✓✓
Principes de statistiques		
tous les chapitres	✓✓	😊
Modèles linéaires		
tous les chapitres	✓✓	😊
Modèles linéaires généralisés		
tous les chapitres	✓	✓✓
Modèles mixtes		
tous les chapitres		✓✓
Modèles additifs généralisés		
tous les chapitres		✓
Analyses multivariées		
tous les chapitres		✓
Approche Bayésienne		
tous les chapitres		✓✓

Suggéré ✓ ; Obligatoire ✓✓ ; connaissances attendues (pourraient avoir besoin d'une remise à niveau)



Collant Hex



partie I.

Utiliser R

Chapitre 1

Pour commencer

Bien que R ne soit pas nouveau, sa popularité s'est rapidement accrue au cours des dix dernières années environ (voir [ici](#)). Il a été créé et développé à l'origine par Ross Ihaka et Robert Gentleman dans les années 1990 et la première version stable a été publiée en 2000. Aujourd'hui, R est maintenu par la [R Development Core Team](#). Pourquoi R est-il devenu si populaire et pourquoi devriez-vous apprendre à l'utiliser ? Voici quelques raisons :

- Le logiciel est open source et disponible gratuitement.
- Disponible pour les systèmes d'exploitation Windows, Mac et Linux.
- Un ensemble complet et cohérent d'outils d'analyse statistique.
- Un outil graphique étendu et très flexible, capable de produire des figures de qualité professionnelle.
- Un ensemble de “paquets”  disponibles gratuitement pour étendre les capacités de R.
- Un vaste réseau d'assistance avec de nombreux documents en ligne et disponibles gratuitement.

Toutes les raisons ci-dessus sont d'excellentes raisons d'utiliser R. Cependant, à notre avis, la plus grande raison d'utiliser R est qu'il facilite des pratiques de recherche robustes et reproductibles. Contrairement aux logiciels plus traditionnels de type “pointer et cliquer”, l'écriture de code garantit un enregistrement permanent et précis de toutes les méthodes que vous avez utilisées (et des décisions que vous avez prises) pour l'analyse de vos données. Vous pouvez ensuite partager ce code (et vos données) avec d'autres chercheurs, collègues ou évaluateurs qui pourront reproduire votre analyse à l'identique. C'est l'un des principes de la [science ouverte](#). Nous aborderons d'autres sujets pour faciliter la science ouverte tout au long de ce livre, notamment la création de rapports reproductibles (Chapitre 6) et le contrôle des versions (Chapitre 7).

Dans ce chapitre, nous aborderons les points suivants :

- Comment télécharger et installer R et un IDE sur votre ordinateur
- Vous donner une brève orientation sur les 2 IDE les plus courants utilisés avec R

- Quelques bonnes habitudes à prendre lorsque vous travaillez sur des projets
- Et enfin quelques conseils sur la documentation de votre flux de travail et l'écriture d'un code R lisible.

Quelques conseils sur R

- Utilisez R souvent et régulièrement. Cela permettra de créer et maintenir une dynamique importante.
- Apprendre R n'est pas un test de mémoire. L'un des avantages d'un langage de script est que vous aurez toujours votre code (bien annoté) auquel vous pourrez vous référer lorsque vous oubliez comment faire quelque chose.
- Il n'est pas nécessaire de tout savoir sur R pour être productif.
- Si vous êtes bloqué, faites des recherches en ligne, ce n'est pas de la triche et écrire une bonne requête de recherche est une compétence en soi.
- Si vous vous retrouvez à fixer un code pendant des heures en essayant de comprendre pourquoi il ne fonctionne pas, éloignez-vous quelques minutes.
- En R, il existe de nombreuses façons d'aborder un problème particulier. Si votre code fait ce que vous voulez qu'il fasse dans un temps raisonnable et de manière robuste, ne vous inquiétez pas.
- R n'est qu'un outil pour vous aider à répondre à vos questions intéressantes. Ne perdez pas de vue ce qui est important : vos questions de recherche et vos données. Aucune compétence en matière d'utilisation de R ne sera utile si votre collecte de données est fondamentalement défectueuse ou si votre question est vague.
- Soyez conscient qu'il y aura des moments où les choses deviendront un peu difficiles ou frustrantes. Essayez d'accepter ces périodes comme faisant partie du processus naturel d'apprentissage d'une nouvelle compétence (nous sommes tous passés par là) et rappelez-vous que le temps et l'énergie que vous investissez maintenant seront largement remboursés dans un avenir pas trop lointain.

Bonne chance et n'oubliez pas de vous amuser.

1.1. Installation

1.1.1. Installation de R

Pour démarrer, la première chose à faire est d'installer R. R est disponible gratuitement pour les systèmes d'exploitation Windows, Mac et Linux à partir du [site web du Comprehensive R Archive Network \(CRAN\)](#). Pour les utilisateurs de

Windows et de Mac, nous vous suggérons de télécharger et d'installer les versions binaires précompilées. Il existe des instructions assez complètes pour installer R pour chaque système d'exploitation ([Windows](#), [Mac](#) ou [linux](#)).

Quel que soit le système d'exploitation que vous utilisez, une fois que vous avez installé R, vous devez vérifier qu'il fonctionne correctement. La manière la plus simple de le faire est de lancer R en double-cliquant sur l'icône R (Windows ou Mac) ou en tapant R dans la console (Linux). La console R devrait s'afficher et vous devriez pouvoir taper des commandes R dans la console après l'invite de commande >. Essayez de taper le code R suivant et appuyez sur la touche Entrée :

```
plot(1)
```

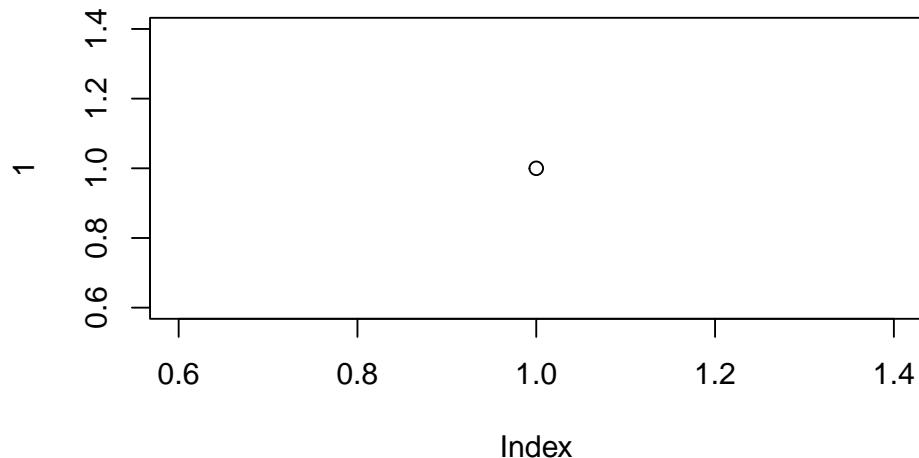


Figure 1.1.: Le meilleur graph du monde, sert juste à tester R

Un graphique avec un seul point au centre devrait apparaître. Si c'est le cas, vous pouvez commencer. Si ce n'est pas le cas, nous vous suggérons de noter toutes les erreurs produites et d'utiliser ensuite votre moteur de recherche préféré pour résoudre le problème.

1.1.2. Installation d'un IDE

Il est fortement recommandé d'utiliser un logiciel d'**E**nvironnement de **D**éveloppement **I**ntégré (IDE) pour travailler avec R. Un IDE simple et extrêmement populaire est [RStudio](#). Une alternative à RStudio est Visual Studio Code, ou VSCode. Un IDE peut être considéré comme un complément à R qui fournit une interface plus conviviale, intégrant

la console R, un éditeur de scripts et d'autres fonctionnalités utiles (comme R markdown et l'intégration de Git Hub).

🔥 Mise en garde

Vous devez installer R avant d'installer un IDE (voir Section 1.1.1 pour plus de détails).

ℹ Note

Lorsque l'on se réfère à un **IDE** dans la suite du texte, il s'agit de RStudio ou de VScode.

1.1.2.1. RStudio

RStudio est disponible gratuitement pour les systèmes d'exploitation Windows, Mac et Linux et peut être téléchargé à partir du [site RStudio](#). Vous devez sélectionner la version 'RStudio Desktop'.

1.1.2.2. VSCode

VSCode est disponible gratuitement pour les systèmes d'exploitation Windows, Mac et Linux et peut être téléchargé à partir du site VS Code .

En outre, vous devez installer l'extension [R pour VSCode](#). Pour faire de VSCode une véritable centrale pour travailler avec R, nous vous recommandons fortement d'installer également :

- [radian](#) : Une console R moderne qui corrige de nombreuses limitations du terminal R officiel et prend en charge de nombreuses fonctionnalités telles que la coloration syntaxique et l'autocomplétion.
- [VSCode-R-Debugger](#) : Une extension de VS Code pour supporter les capacités de débogage de R.
- [httpgd](#) : Un paquet R  pour fournir un dispositif graphique qui sert de manière asynchrone des graphiques SVG via HTTP et WebSockets.

1.1.2.3. Alternatives à RStudio et VSCode

Plutôt que d'utiliser un IDE "tout-en-un", de nombreuses personnes choisissent d'utiliser R et un éditeur de script séparé pour écrire et exécuter du code R. Si vous ne savez pas ce qu'est un éditeur de script, vous pouvez voir ça comme un logiciel de traitement de texte, mais spécialement conçu pour écrire du code. Par chance, de nombreux éditeurs de scripts sont disponibles gratuitement. N'hésitez donc pas à les télécharger et à les tester jusqu'à ce que vous en trouviez un qui vous convienne. Certains éditeurs de scripts ne sont disponibles que pour certains systèmes

d'exploitation et tous ne sont pas spécifiques à R. Vous trouverez ci-dessous des suggestions d'éditeurs de scripts. C'est à vous de choisir celui qui vous convient le mieux : l'une des grandes qualités de R est que *VOUS* choisissez comment vous voulez utiliser R.

1.1.2.3.1. Éditeurs de texte avancés Un moyen léger mais efficace de travailler avec R est d'utiliser des éditeurs de texte avancés tels que :

- [Atom](#) (tous les systèmes d'exploitation)
- [BBedit](#) (Mac OS)
- [gedit](#) (Linux ; livré avec la plupart des distributions Linux)
- [MacVim](#) (Mac OS)
- [Nano](#) (Linux)
- [Notepad++](#) (en anglais) (Windows)
- [Sublime Text](#) (tous les systèmes d'exploitation)
- [vim](#) et son extension [NVim-R](#) (Linux)

1.1.2.3.2. Environnements de développement intégrés Ces environnements sont plus puissants que de simples éditeurs de texte et sont similaires à RStudio :

- [Emacs](#) et son extension [Statistiques de Emacs Speaks](#) (tous les systèmes d'exploitation)
- [RKWard](#) (Linux)
- [Tinn-R](#) (Windows)

1.2. Orientation dans l'IDE

1.2.1. RStudio

Lorsque vous ouvrez RStudio pour la première fois, vous devriez voir la présentation suivante (elle peut être légèrement différente sur un ordinateur Windows).

La grande fenêtre (ou volet) de gauche est la **Console**. La fenêtre en haut à droite est le volet **Environnement / Historique / Connexions** et la fenêtre en bas à droite est le volet **Fichiers / Graphiques / Paquets / Aide / Visualiseur**. Nous aborderons chacun de ces volets dans les paragraphes qui suivent. Vous pouvez personnaliser l'emplacement de chaque volet en cliquant sur le menu “Outils”, puis en sélectionnant Options globales → Disposition

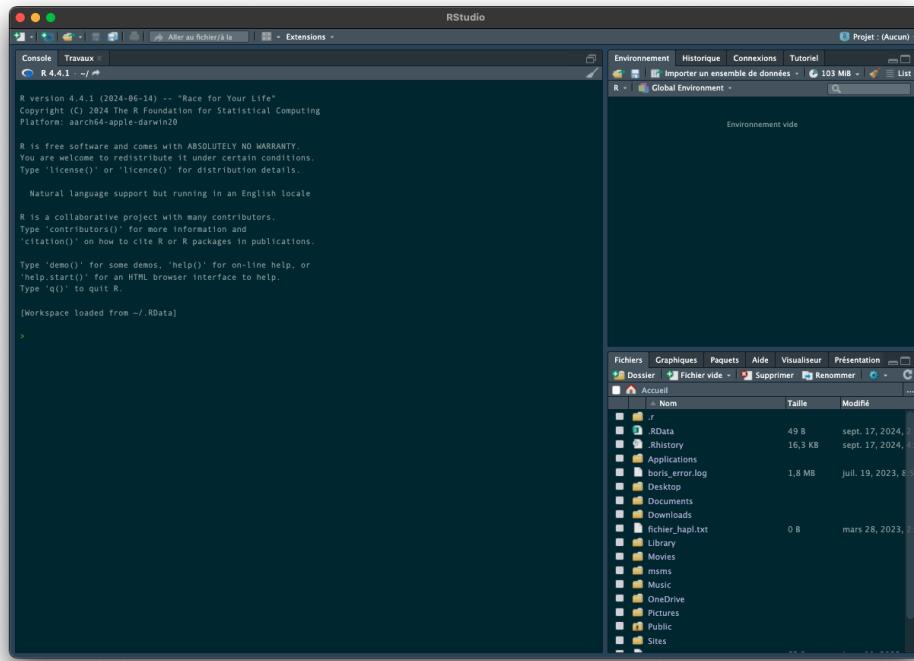


Figure 1.2.: Fenêtre principale R studio

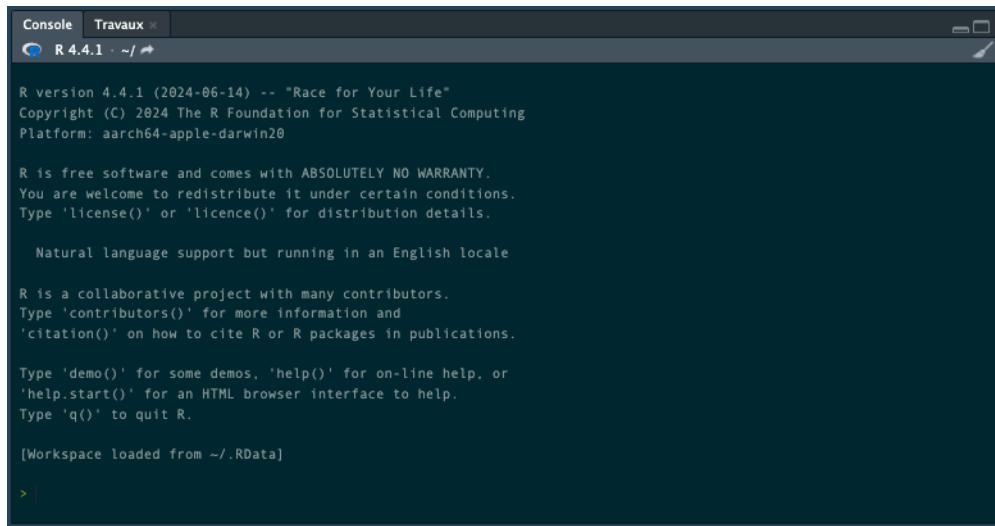
des volets. Vous pouvez redimensionner les volets en cliquant sur le milieu des bords de la fenêtre et en le faisant glisser dans la direction souhaitée. Il existe une multitude d'autres façons de [personnaliser](#) RStudio.

1.2.1.1. Console

La console est le cheval de bataille de R. C'est là que R évalue tout le code que vous écrivez. Vous pouvez taper du code R directement dans la console à l'invite de la ligne de commande, >. Par exemple, si vous tapez `2 + 2` dans la console, vous devriez obtenir la réponse `4` (avec un peu de chance). Ne vous préoccupez pas du [1] au début de la ligne pour l'instant.

Cependant, dès que vous commencez à écrire plus de code R, cela devient plutôt encombrant. Au lieu de taper le code R directement dans la console, il est préférable de créer un script R. Un script R est un simple fichier texte portant l'extension .R qui contient vos lignes de code R. Ces lignes de code sont ensuite introduites dans la console R, ligne par ligne. Pour créer un nouveau script R, cliquez sur le menu “Fichier”, puis sélectionnez Nouveau fichier → Script R.

Vous remarquerez qu'une nouvelle fenêtre (appelée volet Source) apparaît en haut à gauche de RStudio et que la console se trouve désormais en bas à gauche. La nouvelle fenêtre est un éditeur de script et c'est là que vous écrirez votre code.



```
R version 4.4.1 (2024-06-14) -- "Race for Your Life"
Copyright (C) 2024 The R Foundation for Statistical Computing
Platform: aarch64-apple-darwin20

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

Natural language support but running in an English locale

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

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

[Workspace loaded from ~/.RData]

> |
```

Figure 1.3.: Vue de la console R studio



Figure 1.4.: Créer un nouveau fichier script sur R studio

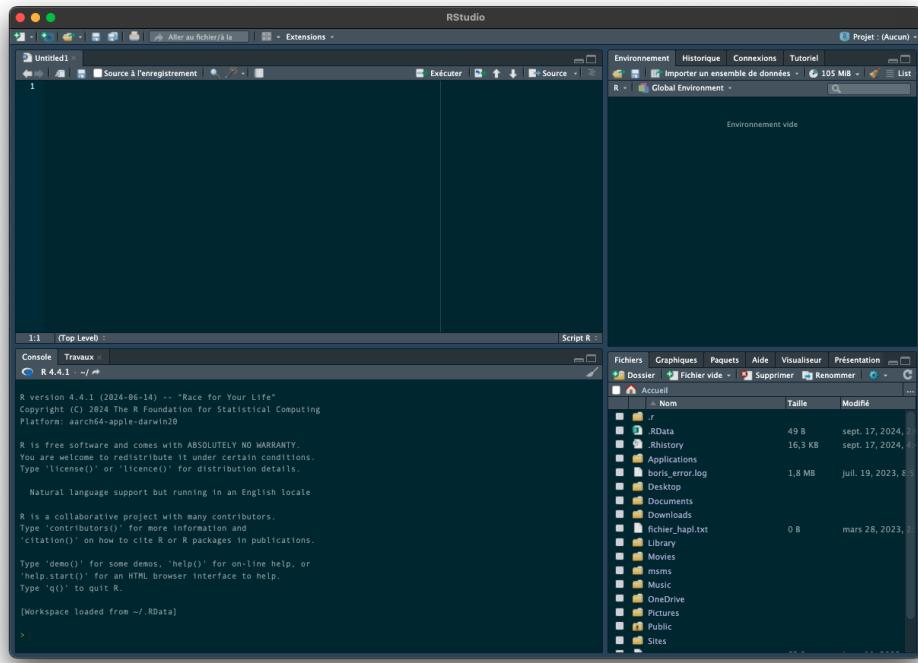


Figure 1.5.: Fenêtre principale avec un nouveau script sur R studio

Pour transférer votre code de votre éditeur de script à la console, placez simplement votre curseur sur la ligne de code, puis cliquez sur le bouton “Exécuter” en haut à droite de la fenêtre de l’éditeur de script.

Vous devriez voir le résultat dans la fenêtre de la console. Si cliquer sur le bouton “Exécuter” devient fastidieux, vous pouvez utiliser le raccourci clavier “ctrl + entrée” (sous Windows et Linux) ou “cmd + entrée” (sous Mac). Vous pouvez enregistrer vos scripts R sous la forme d’un fichier .R en sélectionnant le menu “Fichier” et en cliquant sur “Enregistrer”. Notez que le nom du fichier dans l’onglet devient rouge pour vous rappeler que des modifications n’ont pas été enregistrées. Pour ouvrir votre script R dans RStudio, sélectionnez le menu “Fichier”, puis “Ouvrir le fichier...”. Enfin, il convient de noter que, bien que les scripts R soient enregistrés avec un nom de fichier .R il s’agit en fait de fichiers texte simples qui peuvent être ouverts avec n’importe quel éditeur de texte.

1.2.1.2. Environnement/Histoire/Connexions

La fenêtre Environnement / Historique / Connexions contient de nombreuses informations utiles. Vous pouvez accéder à chaque composant en cliquant sur l’onglet approprié dans le volet.

- L’onglet “Environnement” affiche tous les objets que vous avez créés dans l’environnement actuel (global). Ces objets peuvent être des données que vous avez importées ou des fonctions que vous avez écrites. Les objets peuvent être affichés sous forme de liste ou de grille en sélectionnant l’option dans le menu déroulant

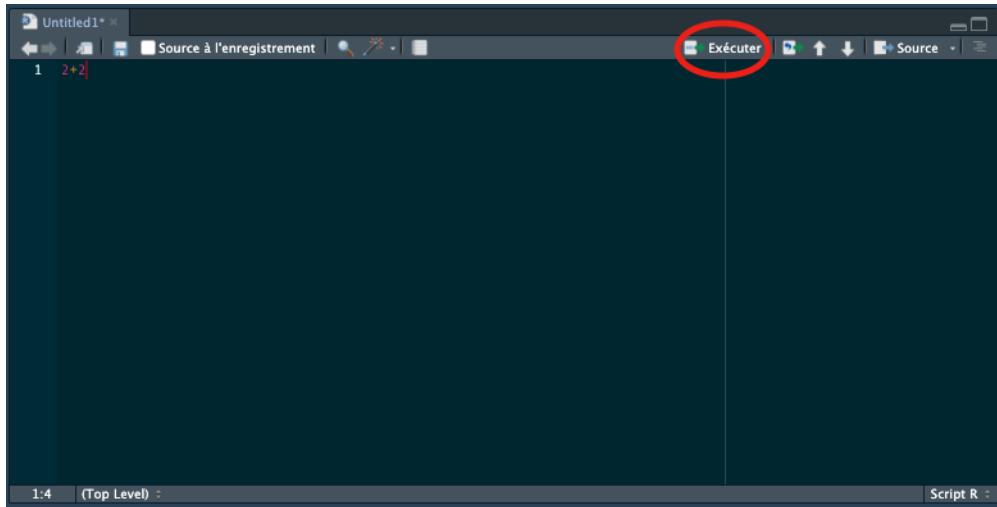


Figure 1.6.: Bouton “Exécuter” sur R studio

situé en haut à droite de la fenêtre. Si vous utilisez le format Grille, vous pouvez supprimer des objets de l'environnement en cochant la case vide située à côté du nom de l'objet, puis en cliquant sur l'icône de balai. Il existe également un bouton “Importer un ensemble de données” qui permet d'importer des données sauvegardées dans différents formats de fichiers. Cependant, nous vous conseillons de ne pas utiliser cette approche pour importer vos données car elle n'est pas reproductible et donc pas robuste (voir Chapitre 3 pour plus de détails).

- L'onglet “Historique” contient une liste de toutes les commandes que vous avez saisies dans la console R. Vous pouvez rechercher dans votre historique la ligne de code que vous avez oubliée, renvoyer le code sélectionné dans la Console ou dans la fenêtre Source. En général, nous n'utilisons jamais cette fonction car nous nous référerons toujours à notre script R.
- L'onglet “Connexions” vous permet de vous connecter à diverses sources de données telles que des bases de données externes.

1.2.1.3. Fichiers/Graphiques/Paquets/Aide/Visualiseur

- L'onglet “Fichiers” répertorie tous les fichiers et répertoires externes dans le répertoire de travail actuel de votre ordinateur. Il fonctionne comme l'explorateur de fichiers (Windows) ou le Finder (Mac). Vous pouvez ouvrir, copier, renommer, déplacer et supprimer les fichiers répertoriés dans la fenêtre.
- L'onglet “Graphiques” est l'endroit où tous les graphiques que vous créez dans R sont affichés (sauf indication contraire de votre part). Vous pouvez “zoomer” sur les graphiques pour les agrandir à l'aide du bouton loupe et faire défiler les graphiques créés précédemment à l'aide des boutons flèches. Il est également possible

d'exporter les graphiques vers un fichier externe à l'aide du menu déroulant “Exportation”. Les graphiques peuvent être exportés dans différents formats de fichiers tels que jpeg, png, pdf, tiff ou copiés dans le presse-papiers (bien qu'il soit probablement préférable d'utiliser les fonctions R appropriées pour ce faire - voir Chapitre 4 pour plus de détails).

- L'onglet “Paquets” répertorie tous les paquets que vous avez installés sur votre ordinateur. Vous pouvez également installer de nouveaux paquets et mettre à jour les paquets existants en cliquant respectivement sur les boutons “Installer” et “Mettre à jour”.
- L'onglet “Aide” affiche la documentation d'aide R pour chaque fonction. Nous verrons comment consulter les fichiers d'aide et comment rechercher de l'aide dans le Chapitre 2).
- L'onglet “Visualiseur” affiche le contenu web local tel que les graphiques web générés par certains packages.

1.2.2. VSCode

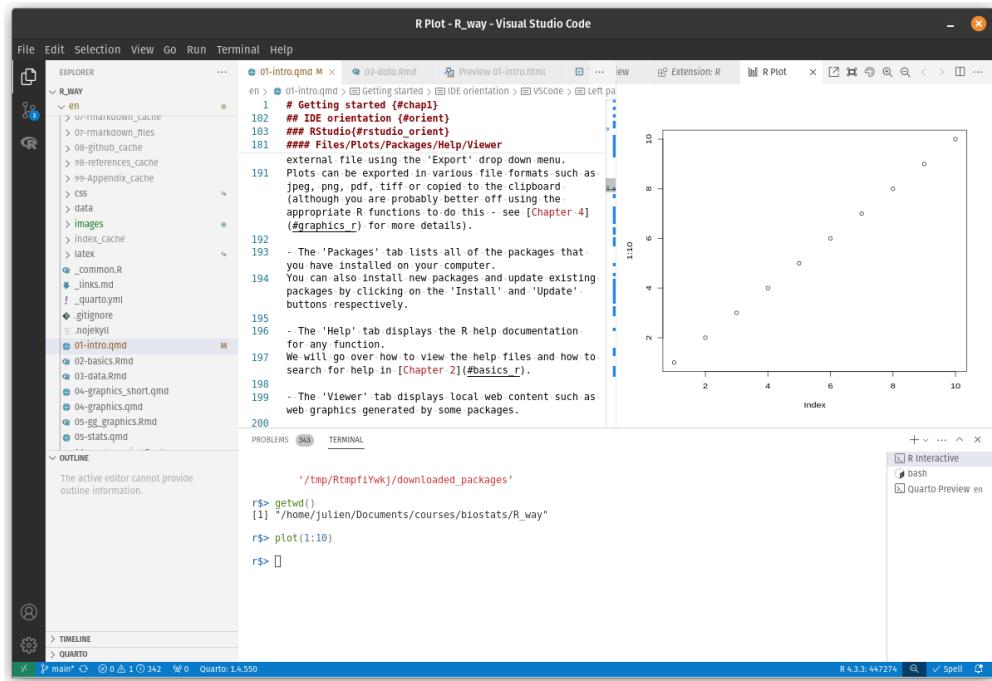


Figure 1.7.: Aperçu de la fenêtre VSCode

1.2.2.1. Panneau gauche

Contient :

- Gestionnaire de fichiers et aperçu des fichiers

- Support R incluant l'environnement R / la recherche R / l'aide R / l'installation de paquets
- Interaction avec Github

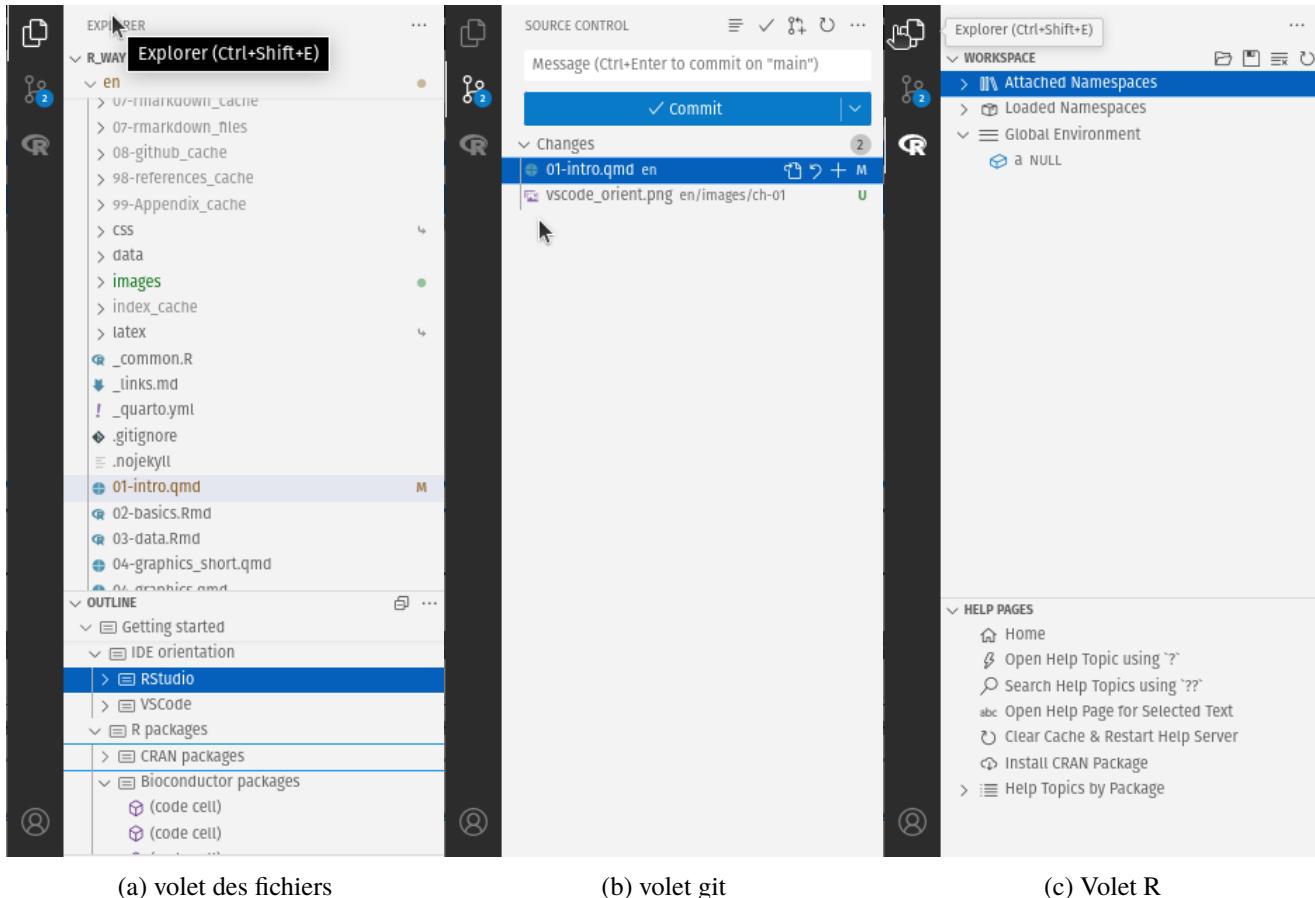


Figure 1.8.: Panneau de gauche VS Code

1.2.2.2. Onglets de l'éditeur

Comprend :

- Un onglet de graphiques (avec historique et navigation)
- Un éditeur de scripts
- Un panneaux de prévisualisation

1.2.2.3. Fenêtre du terminal

Contient :

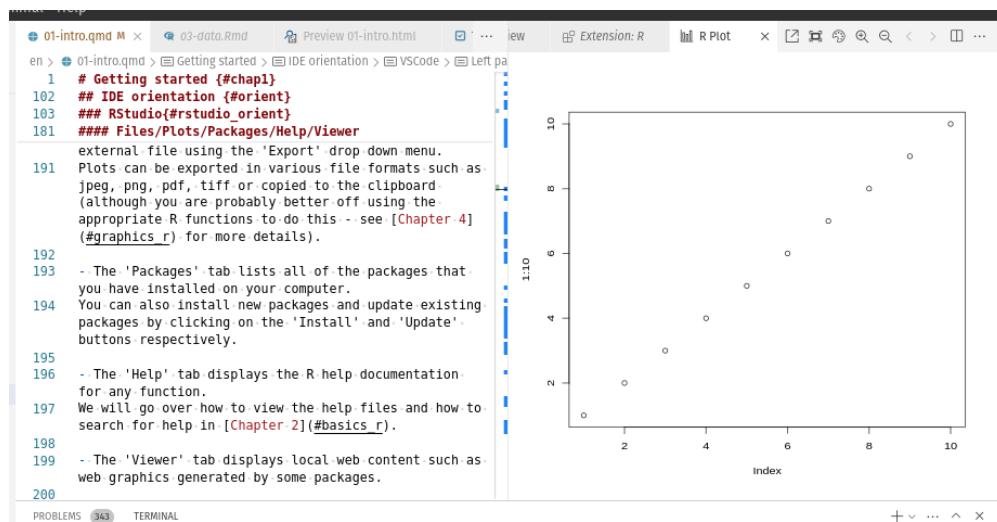


Figure 1.9.: Onglet d'édition et panneaux de prévisualisation VSCode

- Le terminal permettant d'avoir une session R ou tout autre type de terminal nécessaire (bash/tmux/). Il peut être divisé et exécuter plusieurs sessions en même temps.
- Un panneau de problèmes mettant en évidence les problèmes de grammaire et de codage



Figure 1.10.: Fenêtre du terminal VSCode

1.3. Répertoires de travail

Le répertoire de travail est l'emplacement par défaut où R cherchera les fichiers que vous souhaitez charger et où il placera tous les fichiers que vous enregistrez. L'un des avantages de l'utilisation des projets RStudio est que lorsque vous ouvrez un projet, il définit automatiquement votre répertoire de travail à l'emplacement approprié. Vous pouvez vérifier le chemin d'accès de votre répertoire de travail en utilisant l'une des méthodes suivantes `getwd()` ou `here()` fonctions.

```
getwd()
```

```
[1] "/home/julien/Documents/courses/biostats/livre/enfR"
```

Dans l'exemple ci-dessus, le répertoire de travail est un dossier appelé “R_way” qui est un sous-dossier de “biostats” dans le dossier “courses” qui lui-même se trouve dans un dossier “Documents” situé dans le dossier “julien” qui lui-même se trouve dans le dossier “home”. Sur un ordinateur fonctionnant sous Windows, notre répertoire de travail comprendrait également une lettre de lecteur (c.-à-d. C:\home\julien\Documents\courses\biostats\R_way).

Si vous n'utilisez pas d'IDE, vous devez définir votre répertoire de travail à l'aide de la commande `setwd()` au début de chaque script R (ce que nous avons fait pendant de nombreuses années).

```
setwd("/home/julien/Documents/courses/biostats/R_way/")
```

Cependant, le problème avec `setwd()` est qu'il utilise un chemin d'accès *absolu* spécifique à l'ordinateur sur lequel vous travaillez. Si vous souhaitez envoyer votre script à quelqu'un d'autre (ou si vous travaillez sur un autre ordinateur), ce chemin d'accès absolu ne fonctionnera pas sur l'ordinateur de votre ami/collègue, car la configuration de ses répertoires sera différente (il est peu probable que vous ayez une structure de répertoires /home/julien/Documents/courses/biostats/ sur votre ordinateur). Il en résulte un projet qui n'est pas autonome et qui n'est pas facilement transportable. Les IDE résolvent ce problème en vous permettant d'utiliser des fichiers *relatif* qui sont relatifs au chemin d'accès au fichier *racine* du projet. Le fichier racine du projet est simplement le répertoire qui contient le fichier `.Rproj` dans Rstudio (`first_project.Rproj` dans notre cas) ou le dossier de base de votre espace de travail dans VScode. Si vous souhaitez partager vos analyses avec quelqu'un d'autre, il vous suffit de copier le répertoire complet du projet et de l'envoyer à votre collaborateur. Il lui suffira alors d'ouvrir le fichier du projet et tous les scripts R qui contiennent des références à des chemins d'accès relatifs fonctionneront. Par exemple, disons que vous avez créé un sous-répertoire appelé `data` dans votre répertoire de projet racine, qui contient un fichier délimité au format csv appelé `mydata.csv` (nous aborderons les structures de répertoire plus loin dans Section 1.4). Pour importer ce fichier dans un projet RStudio à l'aide de la commande `read.csv()` (ne vous en préoccupez pas pour l'instant, nous aborderons cette question plus en détail dans Chapitre 3), tout ce que vous devez inclure dans votre script R est

```
dat <- read.csv("data/mydata.csv")
```

Parce que le chemin du fichier `data/mydata.csv` est relatif au répertoire du projet, peu importe où votre collaborateur enregistre le répertoire du projet sur son ordinateur, cela fonctionnera toujours.

Si vous n'utilisez pas un projet RStudio ou un espace de travail VScode, vous devrez soit définir le répertoire de travail en fournissant le chemin complet de votre répertoire, soit spécifier le chemin complet du fichier de données. Aucune de ces deux options n'est reproductible sur d'autres ordinateurs.

```
setwd("/home/julien/Documents/courses/biostats/R_way")
dat <- read.csv("data/mydata.csv")
```

ou

```
dat <- read.csv("/home/julien/Documents/courses/biostats/R_way/data/mydata.csv")
```

Pour ceux d'entre vous qui souhaitent pousser plus loin la notion de chemins d'accès relatifs aux fichiers, jetez un coup d'œil à la fonction `here()` du paquet `here` . La fonction `here()` vous permet de créer des chemins d'accès pour n'importe quel fichier par rapport au répertoire racine du projet, qui ne dépendent pas du système d'exploitation (fonctionne sur une machine Mac, Windows ou Linux). Par exemple, pour importer notre `mydata.csv` à partir du répertoire `data` il suffit d'utiliser :

```
library(here) # il se peut que vous deviez d'abord installer le paquet 'here'
dat <- read.csv(here("data", "mydata.csv"))
```

1.4. Structure du répertoire

Outre l'utilisation de RStudio Projects, il est également conseillé de structurer votre répertoire de travail de manière cohérente et logique afin de vous aider, vous et vos collaborateurs. Nous utilisons fréquemment la structure de répertoire suivante dans nos projets basés sur R.

Dans notre répertoire de travail, nous avons les répertoires suivants :

- **Racine** - Il s'agit du répertoire de votre projet contenant votre fichier `.Rproj`. Nous avons tendance à garder tous les scripts R ou [Rq]md nécessaires à l'analyse/au rapport dans ce dossier racine ou dans le dossier `scripts` s'il y en a trop.
- **données** - Nous stockons toutes nos données dans ce répertoire. Le sous-répertoire appelé `données` contient des fichiers de données brutes et uniquement des fichiers de données brutes. Ces fichiers doivent être traités comme s'ils étaient **en lecture seule** et ne doivent en aucun cas être modifiés. Si vous devez traiter/nettoyer/modifier

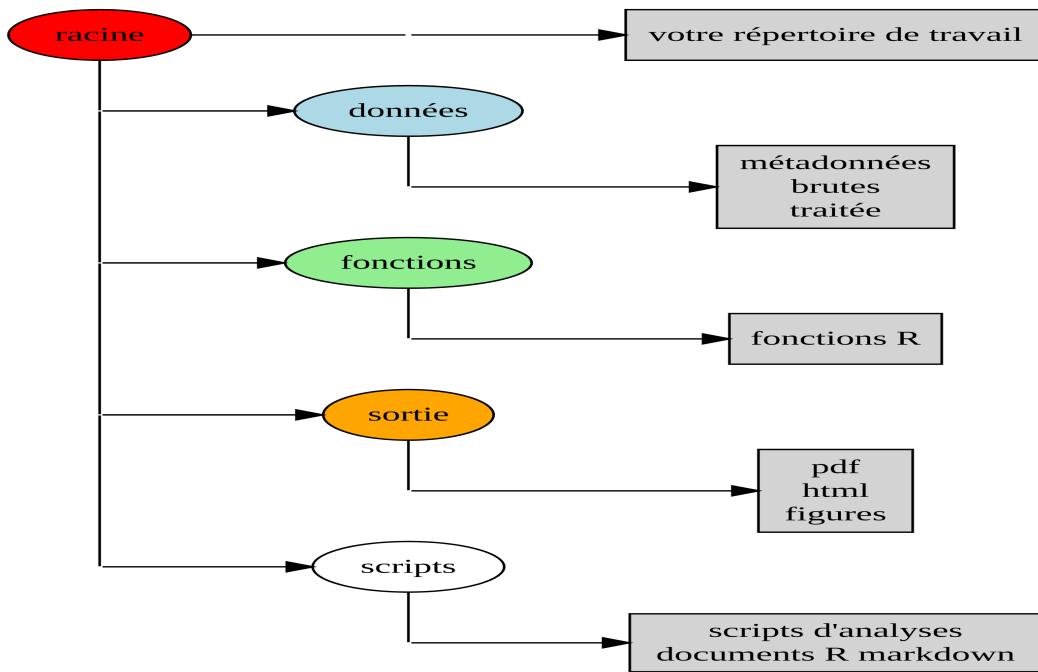


Figure 1.11.: Structure de répertoire recommandée pour des analyses sur R

vos données, faites-le dans R (et non dans MS Excel), car vous pourrez documenter (et justifier) toutes les modifications apportées. Toutes les données traitées doivent être sauvegardées dans un fichier séparé et stockées dans le fichier `données_traitées`. Les informations sur les méthodes de collecte des données, les détails du téléchargement des données et toute autre métadonnée utile doivent être sauvegardés dans un document texte (voir les fichiers README ci-dessous) dans le sous-répertoire `meta_données`.

- **fonctions** - Il s'agit d'un répertoire facultatif dans lequel nous enregistrons toutes les fonctions R personnalisées que nous avons écrites pour l'analyse en cours. Celles-ci peuvent ensuite être importées dans R à l'aide de la fonction `source()`.
- **scripts** - Un répertoire optionnel où nous enregistrons nos documents R markdown et/ou les principaux scripts R que nous avons écrits pour le projet en cours. Ces documents ne sont pas enregistrés dans le dossier racine.
- **sortie** - Les résultats de nos scripts R, tels que les graphiques, les fichiers HTML et les résumés de données, sont enregistrés dans ce répertoire. Cela nous aide, ainsi que nos collaborateurs, à distinguer les fichiers qui sont des sorties de ceux qui sont des fichiers sources.

Bien entendu, la structure décrite ci-dessus est celle qui nous convient le mieux la plupart du temps et doit être considérée comme un point de départ pour vos propres besoins. Nous avons tendance à avoir une structure de répertoires assez cohérente dans tous nos projets, car cela nous permet de nous orienter rapidement lorsque nous revenons à un projet après un certain temps. Cela dit, les besoins varient d'un projet à l'autre et nous ajoutons ou

supprimons des répertoires en fonction des besoins.

Vous pouvez créer votre structure de répertoire à l'aide de l'explorateur Windows (ou Finder sur Mac) ou dans votre IDE en cliquant sur le bouton “Nouveau dossier” dans le panneau “Fichiers”.

Une autre approche est d'utiliser la fonction `dir.create()` dans la console R.

```
# créer un répertoire appelé 'données'  
dir.create("données")
```

1.5. Organisation des projets

Comme pour la plupart des choses de la vie, lorsqu'il s'agit de traiter des données et de les analyser, les choses sont tellement plus simples si vous êtes organisé. Une organisation claire du projet permet à la fois à vous (et surtout au futur vous) et à vos collaborateurs de donner un sens à ce que vous avez fait. Il n'y a rien de plus frustrant que de revenir à un projet des mois (parfois des années) plus tard et de devoir passer des jours (ou des semaines) à comprendre où tout se trouve, ce que vous avez fait et pourquoi vous l'avez fait. Un projet bien documenté, doté d'une structure cohérente et logique, augmente les chances de pouvoir reprendre le projet là où il s'est arrêté sans trop de difficultés, quel que soit le temps qui s'est écoulé. En outre, il est beaucoup plus facile d'écrire du code pour automatiser des tâches lorsque les fichiers sont bien organisés et portent des noms judicieux. Cela est d'autant plus vrai maintenant qu'il n'a jamais été aussi facile de collecter de grandes quantités de données qui peuvent être sauvegardées dans des milliers, voire des centaines de milliers de fichiers de données distincts. Enfin, un projet bien organisé réduit le risque d'introduire des bogues ou des erreurs dans votre flux de travail et, s'ils se produisent (ce qui est inévitable à un moment ou à un autre), il est plus facile de retrouver ces erreurs et de les traiter efficacement.

Il existe également quelques mesures simples que vous pouvez prendre dès le début d'un projet pour aider à maintenir les choses en bon état.

Un bon moyen de garder les choses organisées est d'utiliser les projets RStudio ou les espaces de travail VSCode, désignés sous le nom de `projet`. Un `projet` conserve tous vos scripts R, vos documents R markdown, vos fonctions R et vos données en un seul endroit. Ce qu'il y a de bien avec un `projet` est que chacun a son propre répertoire, son propre historique et ses propres documents sources, de sorte que les différentes analyses sur lesquelles vous travaillez sont complètement séparées les unes des autres. Cela signifie que vous pouvez très facilement passer d'un `projet` à l'autre sans craindre qu'ils n'interfèrent l'un avec l'autre.

1.5.1. RStudio

Pour créer un projet, ouvrez RStudio et sélectionnez Fichier → Nouveau projet... dans le menu. Vous pouvez créer un projet entièrement nouveau, un projet à partir d'un répertoire existant ou un projet à version contrôlée (voir le Chapitre 7 pour plus de détails à ce sujet). Dans ce chapitre, nous allons créer un projet dans un nouveau répertoire.

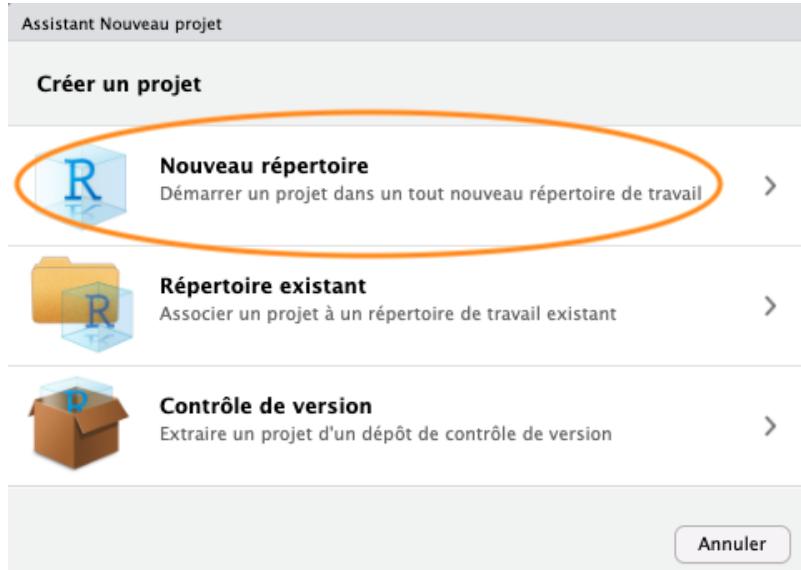


Figure 1.12.: Crée un Projet R Studio étape 1

Vous pouvez également créer un nouveau projet en cliquant sur le bouton “Projet” en haut à droite de RStudio et en sélectionnant “Nouveau projet...”

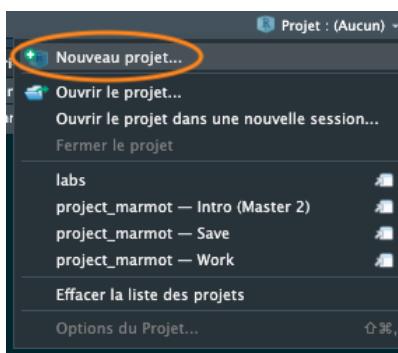


Figure 1.13.: Crée un Projet R Studio étape 2

Dans la fenêtre suivante, sélectionnez “Nouveau projet”.

Saisissez le nom du répertoire que vous souhaitez créer dans le champ “Nom du répertoire :” (nous l'appellerons `premier_projet` pour ce chapitre). Si vous souhaitez modifier l'emplacement du répertoire sur votre ordinateur, cliquez sur le bouton “Parcourir...” et naviguez jusqu'à l'endroit où vous souhaitez créer le répertoire. Nous cochons

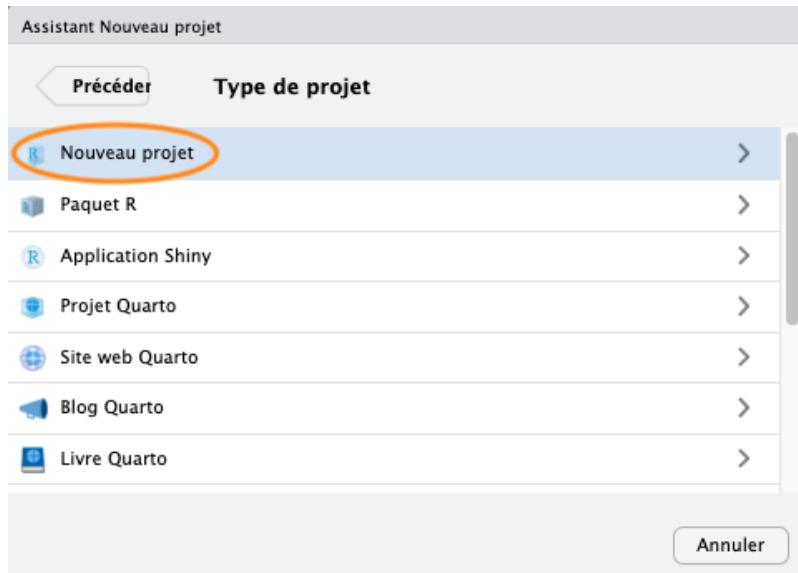


Figure 1.14.: Crée un Projet R Studio étape 3

toujours la case “Ouvrir dans une nouvelle session”. Enfin, cliquez sur le bouton “Créer un projet” pour créer le nouveau projet.

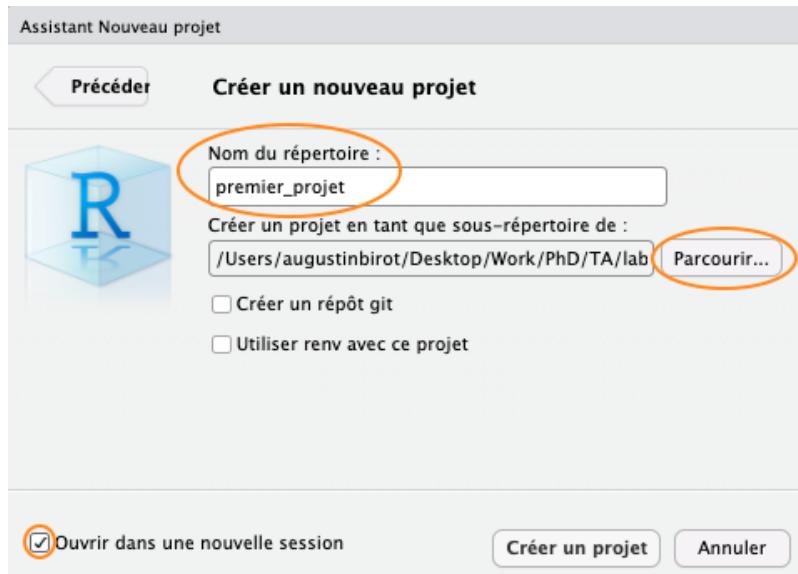


Figure 1.15.: Crée un Projet R Studio étape 4

Une fois votre nouveau projet créé, vous disposerez d'un nouveau dossier sur votre ordinateur contenant un fichier de projet RStudio appelé `premier_projet.Rproj`. Ce projet `.Rproj` contient diverses options de projet (mais vous ne devriez pas vraiment interagir avec lui) et peut également être utilisé comme raccourci pour ouvrir le projet directement à partir du système de fichiers (il suffit de double-cliquer dessus). Vous pouvez le vérifier dans l'onglet “Fichiers” de RStudio (ou dans Finder si vous êtes sur Mac ou dans l'Explorateur de fichiers sous Windows).

La dernière chose que nous vous suggérons de faire est de sélectionner Outils → Options du Project... dans

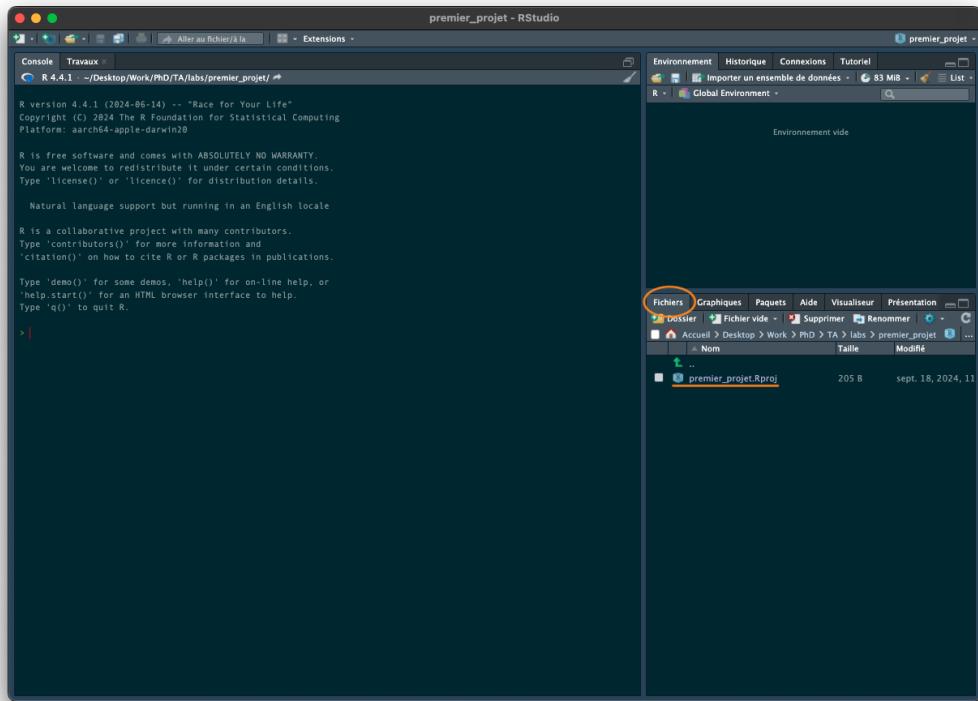


Figure 1.16.: Crée un Projet R Studio étape finale

le menu. Cliquez sur l’onglet “Général” sur le côté gauche et modifiez les valeurs pour “Restaurer .RData dans l’espace de travail au démarrage” et “Sauvegarder l’espace de travail dans .RData à la sortie” de “Par défaut” à “Non”. Cela garantit que chaque fois que vous ouvrez votre projet, vous démarrez avec une session R propre. Vous n’êtes pas obligé de faire cela (beaucoup de gens ne le font pas), mais nous préférions commencer avec un espace de travail complètement propre chaque fois que nous ouvrons nos projets pour éviter tout conflit potentiel avec des choses que nous avons faites dans des sessions précédentes (ce qui conduit parfois à des résultats surprenants et à des maux de tête pour résoudre le problème). L’inconvénient est que vous devrez réexécuter votre code R à chaque fois que vous ouvrirez votre projet.

Maintenant que vous avez mis en place un projet RStudio, vous pouvez commencer à créer des scripts R (ou des documents R markdown /Quarto, voir Chapitre 6) ou tout ce dont vous avez besoin pour compléter votre projet. Tous les scripts R seront désormais contenus dans le projet RStudio et enregistrés dans le dossier du projet.

1.5.2. VSCode

Les espaces de travail sont similaires aux projets RStudio. Vous devez cependant créer un nouveau dossier avec un fichier R (ou un fichier texte) et l’enregistrer en tant qu’espace de travail.

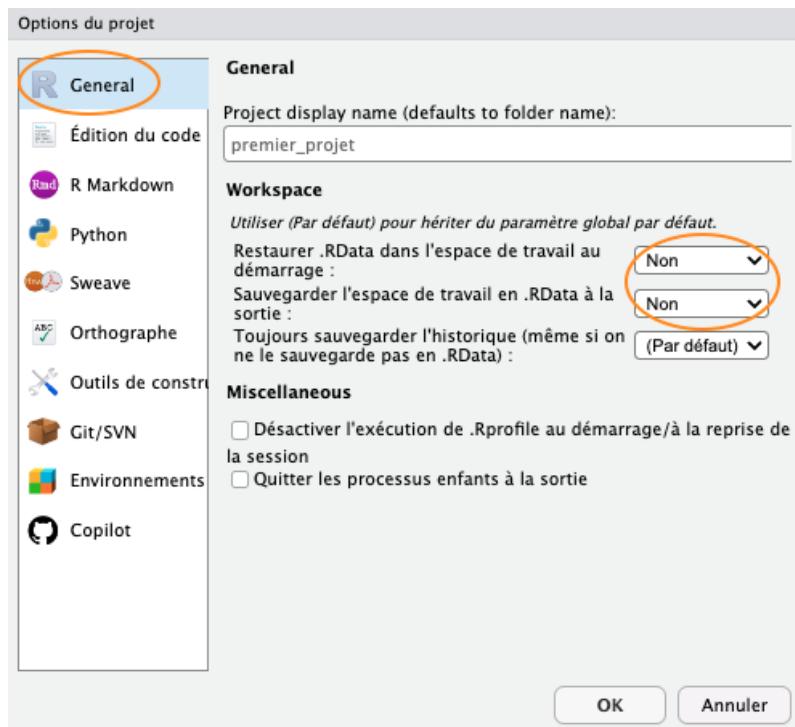


Figure 1.17.: Créer un Projet R Studio changement d'options

1.6. Nom des fichiers

Le nom que vous donnez à vos fichiers a plus d’importance que vous ne le pensez. Nommer les fichiers est également plus difficile que vous ne le pensez. Pour qu’un nom de fichier soit “bon”, il faut qu’il soit informatif et relativement court. Ce n’est pas toujours un compromis facile et il faut souvent y réfléchir. L’idéal est d’éviter les éléments suivants !

Bien qu’il n’y ait pas vraiment d’approche standard reconnue pour nommer les fichiers (en fait [il y en a une](#) mais tout le monde ne l’utilise pas), il y a quelques points à garder à l’esprit.

- Évitez d’utiliser des espaces dans les noms de fichiers en les remplaçant par des traits de soulignement ou des tirets. Pourquoi cela est-il important ? L’une des raisons est que certains logiciels de ligne de commande (en particulier de nombreux outils bioinformatiques) ne reconnaissent pas un nom de fichier comportant un espace et que vous devez vous livrer à toutes sortes de manigances en utilisant des caractères d’échappement pour vous assurer que les espaces sont traités correctement. Même si vous ne pensez pas utiliser un jour un logiciel de ligne de commande, il se peut que vous le fassiez indirectement. Prenez R markdown par exemple, si vous voulez rendre un document R markdown au format pdf en utilisant le paquet `rmarkdown`  vous utiliserez en fait une ligne de commande `LATEX` sous le capot. Une autre bonne raison de ne pas utiliser d’espaces dans les noms de fichiers est que cela rend la recherche de noms de fichiers (ou de parties de noms de fichiers) à

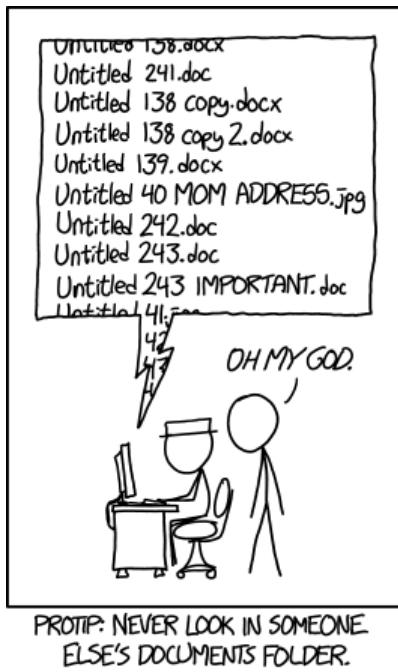


Figure 1.18.: File renaming song (source:<https://xkcd.com/1459/>)

l'aide d'[expressions régulières](#) dans R (ou tout autre langage) beaucoup plus difficile.

- Évitez d'utiliser des caractères spéciaux (par exemple @£\$%^&*(:/)) dans vos noms de fichiers.
- Si vous créez des versions de vos fichiers à l'aide de nombres séquentiels (par ex. fichier1, fichier2, fichier3 ...). Si vous prévoyez d'avoir plus de 9 fichiers, vous devez utiliser 01, 02, 03, ..., 10 afin de garantir que les fichiers soient listés dans le bon ordre. Si vous prévoyez d'avoir plus de 99 fichiers, utilisez 001, 002, 003,
- Pour les dates, utilisez le format ISO 8601 AAAA-MM-JJ (ou AAAAMMMJJ) afin de vous assurer que vos fichiers sont listés dans un ordre chronologique correct.
- N'utilisez jamais le mot *final* dans un nom de fichier - il est extrêmement rare qu'il le soit !

Quelle que soit la convention de dénomination des fichiers que vous décidez d'utiliser, essayez de l'adopter rapidement, de vous y tenir et d'être cohérent.

1.7. Documentation du script

Un petit mot sur l'écriture du code R et la création de scripts R. À moins que vous ne fassiez quelque chose de vraiment rapide et sale, nous vous suggérons de toujours écrire votre code R sous la forme d'un script R. Les scripts R sont ce qui permet à R d'être plus efficace. Les scripts R sont ce qui rend R si utile. Non seulement vous disposez d'un enregistrement complet de votre analyse, de la manipulation des données à la visualisation et à l'analyse

statistique, mais vous pouvez également partager ce code (et ces données) avec vos amis, vos collègues et, surtout, lorsque vous soumettez et publiez votre recherche dans une revue. Dans cette optique, veillez à inclure dans votre script R toutes les informations nécessaires pour rendre votre travail reproductible (noms des auteurs, dates, plan d'échantillonnage, etc.). Ces informations peuvent être incluses sous la forme d'une série de commentaires # ou, mieux encore, en mélangeant code exécutable et narration dans un document R markdown (Chapitre 6). C'est aussi une bonne pratique d'inclure la sortie de la fonction `sessionInfo()` à la fin de n'importe quel script qui affiche la version de R, les détails du système d'exploitation et les paquets chargés. Une très bonne alternative est d'utiliser la fonction `session_info()` du paquet `xfun` pour un résumé plus concis de notre environnement de session.

Voici un exemple d'inclusion de méta-information au début d'un script R

```
# Titre: Analyse de séries temporelles de consommation de lasagnes

# Objectif : Ce script effectue une analyse de série temporelle sur
#             les plats de lasagnes que les enfants veulent manger chaque semaine.
#             Les données sont des dénominations de plats de lasagnes (révés) par semaine
#             collectés à partir de 24 enfants à l'école "Repas-de-rêve"
#             entre 2042 et 2056.

# fichier de données: lasagna_dreams.csv

# Auteur: Un. Estomac
# Coordonnées de contact: un.estomac@univ.repas.com

# Date de création du script: Fri Mar 29 17:06:44 2010 -----
# Date de dernière modification du script: Wed Sep 18 12:14:18 2024 ----

# paquets chargés
library(tidyverse)
library(ggplot2)

print("écrivez votre merveilleux code R ici")

# bonne pratique pour inclure des informations sur la session
```

```
xfun::session_info()
```

Il ne s'agit que d'un exemple et il n'y a pas de règles strictes, alors n'hésitez pas à développer un système qui vous convienne. Un raccourci très utile dans RStudio est d'inclure automatiquement un horodatage dans votre script R. Pour ce faire, écrivez `ts` à l'endroit où vous souhaitez insérer votre horodatage dans votre script R, puis appuyez sur les touches ‘shift + tab’. RStudio convertira `ts` en date et heure actuelles et commenterà automatiquement cette ligne avec un `#`. Un autre raccourci très utile de RStudio consiste à commenter plusieurs lignes de votre script avec le symbole `#`. Pour ce faire, sélectionnez les lignes de texte que vous souhaitez commenter et appuyez sur ‘ctrl + shift + c’ (ou ‘cmd + shift + c’ sur un mac). Pour décommenter les lignes, utilisez à nouveau ‘ctrl + shift + c’.

En plus d'inclure des métadonnées dans vos scripts R, il est également courant de créer un fichier texte séparé pour enregistrer les informations importantes. Par convention, ces fichiers texte sont nommés README. Nous incluons souvent un fichier README dans le répertoire où nous conservons nos données brutes. Dans ce fichier, nous indiquons la date à laquelle les données ont été collectées (ou téléchargées), la manière dont elles ont été collectées, des informations sur l'équipement spécialisé, les méthodes de conservation, le type et la version des machines utilisées (c'est-à-dire l'équipement de séquençage), etc. Vous pouvez créer un fichier README pour votre projet dans RStudio en cliquant sur le menu `Fichier → Nouveau fichier → Fichier texte`.

1.8. Guide de style R

La façon dont vous écrivez votre code dépend plus ou moins de vous, bien que votre objectif soit de le rendre aussi facile à lire que possible (pour vous et pour les autres). Bien qu'il n'y ait pas de règles (ni de police du code), nous vous encourageons à prendre l'habitude d'écrire un code R lisible en adoptant un style particulier. Nous vous suggérons de suivre le [R style guide](#) de Google dans la mesure du possible. Ce guide de style vous aidera à décider où utiliser les espaces, comment indenter le code et comment utiliser les crochets `[]` et les parenthèses bouclées `{ }`, entre autres choses.

Pour vous aider à formater le code :

- VSCode il y a un formateur intégré dans l'extension R pour VSCode. Il vous suffit d'utiliser le raccourci clavier pour reformater le code de manière agréable et automatique.
- Pour RStudio, vous pouvez installer le paquet `styler`  qui inclut une extension RStudio vous permettant de reformater automatiquement le code sélectionné (ou des fichiers et projets entiers) d'un simple clic de souris. Vous pouvez trouver plus d'informations sur le paquet `styler` , y compris comment l'installer [ici](#). Une

fois installé, vous pouvez sélectionner le code que vous souhaitez remodeler, cliquer sur le bouton “Extension” en haut de RStudio et sélectionner l’option “Style Selection”. Voici un exemple de code R mal formaté

```

1 for(i in 1:20){ snouter.means[i] <- mean(snouter$snouter.count[i]/12+cm,na.rm=TRUE)}
2
3 v if(y==0){
4 ^ log(x) else (y^x)

```

Figure 1.19.: Code mal formaté

Mettez maintenant le code en surbrillance et utilisez le paquet **styler** 📦 pour reformater

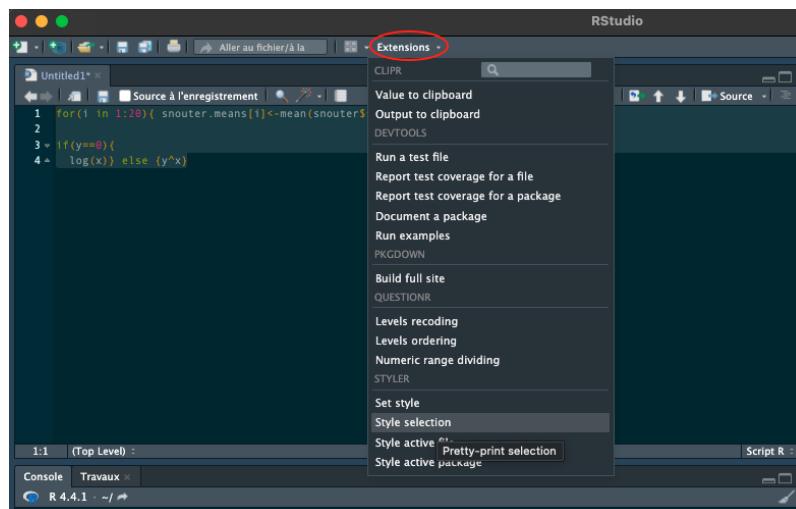


Figure 1.20.: Structurer le code avec styler

Pour produire un code joliment formaté

```

1 v for (i in 1:20) {
2   snouter.means[i] <- mean(snouter$snouter.count[i] / 12 + cm, na.rm = TRUE)
3 ^
4
5 v if (y == 0) {
6   log(x)
7 v } else {
8   y^x
9 ^

```

Figure 1.21 : Code bien structuré

1.9. Sauvegarde des projets

Ne soyez pas la personne qui perd des données et des analyses durement acquises (et souvent coûteuses). Ne soyez pas cette personne qui pense que cela ne m'arrivera jamais - ça vous arrivera ! Pensez toujours au pire scénario, à quelque chose qui vous donnera des sueurs froides la nuit, et faites tout ce qui est en votre pouvoir pour que cela n'arrive jamais. Pour être clair, si vous comptez copier vos précieux fichiers sur un disque dur externe ou une clé USB, ce n'est **PAS** une stratégie de sauvegarde efficace. Ces objets tombent souvent en panne lorsque vous les glissez dans votre sac à dos ou votre "sac de tous les jours" et que vous les trimballez entre votre bureau et votre domicile. Même si vous les laissez branchés sur votre ordinateur, que se passe-t-il lorsque le bâtiment brûle (nous avons bien dit le pire des cas !)?

Idéalement, vos sauvegardes devraient être hors site et incrémentielles. Heureusement, il existe de nombreuses options pour sauvegarder vos fichiers. La première chose à faire est de chercher dans votre propre institut. La plupart (toutes ?) des universités disposent d'une forme de stockage en réseau qui devrait être facilement accessible et qui est également étayée par un plan complet de reprise après sinistre. D'autres options incluent des services basés sur le cloud tels que Google Drive et Dropbox (pour n'en citer que quelques-uns), mais assurez-vous que vous ne stockez pas de données sensibles sur ces services et que vous êtes à l'aise avec les politiques de confidentialité souvent exorbitantes.

Si ces services sont très efficaces pour stocker des fichiers, ils ne sont pas vraiment utiles pour les sauvegardes incrémentielles. Pour retrouver les versions précédentes des fichiers, il faut souvent passer un temps fou à parcourir plusieurs fichiers nommés '*final.doc*', '*final_v2.doc*' et '*final_utilisecluila.doc*' etc. jusqu'à ce que vous trouviez celui que vous cherchiez. Le meilleur moyen que nous connaissons pour sauvegarder des fichiers et gérer différentes versions de fichiers est d'utiliser Git et GitHub. Pour en savoir plus sur la façon dont vous pouvez utiliser RStudio, Git et GitHub ensemble, consultez Chapitre 7.

1.10. Citer R et les paquets R

De nombreuses personnes ont investi énormément de temps et d'énergie pour faire de R l'excellent logiciel que vous utilisez aujourd'hui. Si vous utilisez R dans votre travail (et nous espérons que vous le ferez), n'oubliez pas de citer non seulement R, mais aussi tous les paquets que vous avez utilisés. Pour obtenir la citation la plus récente pour R, vous pouvez utiliser la fonction `citation()`.

```
citation()
```

To cite R in publications use:

R Core Team (2024). *_R: A Language and Environment for Statistical Computing_*. R Foundation for Statistical Computing, Vienna, Austria.
<https://www.R-project.org/>.

A BibTeX entry for LaTeX users is

```
@Manual{,
  title = {R: A Language and Environment for Statistical Computing},
  author = {{R Core Team}},
  organization = {R Foundation for Statistical Computing},
  address = {Vienna, Austria},
  year = {2024},
  url = {https://www.R-project.org/},
}
```

We have invested a lot of time and effort in creating R, please cite it when using it for data analysis. See also 'citation("pkgname")' for citing R packages.

Si vous souhaitez citer un package particulier que vous avez utilisé pour votre analyse de données, vous pouvez également utiliser la fonction `citation()` pour obtenir l'information.

```
citation(package = "here")
```

To cite package 'here' in publications use:

Müller K (2020). *_here: A Simpler Way to Find Your Files_*. R package version 1.0.1, <https://CRAN.R-project.org/package=here>.

A BibTeX entry for LaTeX users is

```
@Manual{,  
  title = {here: A Simpler Way to Find Your Files},  
  author = {Kirill Müller},  
  year = {2020},  
  note = {R package version 1.0.1},  
  url = {https://CRAN.R-project.org/package=here},  
}
```

Selon nous, l'outil le plus utile pour la citation est le paquet `grateful`  qui vous permet de générer les informations de citation dans un fichier, ainsi que de créer une phrase ou un tableau citant tous les paquets utilisés. Cela devrait devenir la norme dans tout manuscrit honnêtement. Voir Table 15.10 pour un exemple de résultat produit avec `grateful`.

Chapitre 2

Quelques notions de base sur R

Objectif de ce chapitre :

- Créer des objets et leur attribuer des valeurs.
- Explorer les différents types d'objets et effectuer des opérations courantes sur ces objets.
- Obtenir de l'aide en R et présenter certaines ressources pour vous aider à apprendre R.
- Sauvegarder votre travail.
- Utiliser et installer des paquets  pour étendre les capacités de base de R.

2.1. Considérations importantes

Les captures d'écrans présentées proviennent de RStudio mais tout est très similaire sur VSCode.

Avant de poursuivre, quelques points à garder à l'esprit tout au long de ce chapitre :

- R est sensible à la casse, c'est-à-dire que A n'est pas la même chose que a et anova, ce n'est pas Anova.
- Tout ce qui suit un # est interprété comme un commentaire et ignoré par R. Ces *commentaires* doivent être utilisés librement dans votre code, à la fois pour votre propre information et pour aider vos collaborateurs. L'écriture de commentaires est un peu un [art](#) que vous maîtriserez de mieux en mieux avec l'expérience.
- Dans R, les commandes sont généralement séparées par une nouvelle ligne. Vous pouvez également utiliser un point-virgule ; pour séparer vos commandes, mais nous vous le déconseillons fortement (rend le code très difficilement lisible).

- Si une invite de continuation + apparaît dans la console après l'exécution de votre code, cela signifie que vous n'avez pas terminé votre code correctement. Cela se produit souvent lorsque vous oubliez de fermer une parenthèse, ce qui est particulièrement fréquent lors que l'on utilise des parenthèses imbriquées (((commande quelconque))). Terminez simplement la commande sur la nouvelle ligne ou appuyez sur la touche “escape” de votre clavier (voir le point ci-dessous) et corrigez la faute de frappe.
- En général, R est assez tolérant vis-à-vis des espaces supplémentaires insérés dans votre code, en fait l'utilisation d'espaces est activement encouragée. Cependant, les espaces ne doivent pas être insérés dans les opérateurs, c'est-à-dire <- ne peut pas s'écrire < - (notez l'espace). Voir le [guide de style](#) pour savoir où placer les espaces afin de rendre votre code plus lisible.
- Si votre console se bloque et ne répond plus après l'exécution d'une commande, vous pouvez souvent vous sortir d'affaire en appuyant sur la touche “escape” (esc) de votre clavier ou en cliquant sur l'icône d'arrêt/stop en haut à droite de votre console. Cela mettra fin à la plupart des opérations en cours.

2.2. Première étape dans la console

Dans le Chapitre 1, nous avons appris ce qu'était la console R la création de scripts et de Projets. Nous avons également vu comment écrire votre code R dans un script, puis comment insérer ce code dans la console pour qu'il s'exécute (si vous avez oublié comment faire, revenez à la section sur la console (1.2.1.1) pour vous rafraîchir la mémoire). Le fait d'écrire votre code dans un script signifie que vous aurez un enregistrement permanent de tout ce que vous avez fait (à condition de sauvegarder votre script) et vous permet également de faire de nombreux commentaires pour vous rappeler ce que vous aviez fait (ou voulu faire) quand vous retournerez si votre code à l'avenir. Ainsi, pendant que vous travaillez sur ce chapitre, nous vous suggérons de créer un nouveau script (ou un Projet Rstudio) pour écrire votre code au fur et à mesure.

Comme nous l'avons vu au Chapitre 1, nous pouvons utiliser R de la même manière qu'une calculatrice. Nous pouvons saisir une expression arithmétique dans notre script, puis l'envoyer dans la console et recevoir un résultat. Par exemple, si nous tapons l'expression 1 + 1 et que l'on exécute cette ligne de code dans la console, on obtient la réponse 2 (😊 !)

```
1 + 1
```

```
[1] 2
```

Le [1] devant le résultat indique que l'observation au début de la ligne est la première. Cela n'est pas très utile dans cet exemple, mais peut l'être lors de l'impression de résultats sur plusieurs lignes (nous en verrons un exemple ci-dessous). Les autres opérateurs arithmétiques évidents sont `-`, `*`, `/` pour la soustraction, la multiplication et la division respectivement. Pour la multiplication de la matrice, l'opérateur est `%*%`.

R suit la convention mathématique habituelle de l'[ordre des opérations](#). Par exemple, l'expression `2 + 3 * 4` est interprétée comme ayant la valeur $2 + (3 * 4) = 14$ et non $(2 + 3) * 4 = 20$. Il existe un grand nombre de fonctions mathématiques dans R, dont les plus utiles sont les suivantes : `log()`, `log10()`, `exp()`, `sqrt()`.

```
log(1) # logarithme en base e
```

```
[1] 0
```

```
log10(1) # logarithme en base 10
```

```
[1] 0
```

```
exp(1) # antilog naturel, fonction exponentielle
```

```
[1] 2.718282
```

```
sqrt(4) # racine carrée
```

```
[1] 2
```

```
4^2 # 4 puissance 2
```

```
[1] 16
```

```
pi # pas une fonction mais utile
```

```
[1] 3.141593
```

Il est important de comprendre que lorsque vous exécutez un code comme nous l'avons fait ci-dessus, le résultat du code (ou **valeur**) n'est affiché que dans la console. Bien que cela puisse parfois être utile, il est généralement beaucoup plus pratique de stocker la ou les valeurs dans un **objet**.

2.3. Objets en R

Au cœur de presque tout ce que vous ferez (ou ferez probablement) en R se trouve le concept selon lequel tout en R est un **objet**. Ces objets peuvent être pratiquement n’importe quoi, d’un simple nombre ou d’une chaîne de caractères (comme un mot) à des structures très complexes comme la sortie d’un graphique, un résumé de votre analyse statistique ou un ensemble de commandes R effectuant une tâche spécifique. Pour comprendre R, il est essentiel de savoir comment créer des objets et leur attribuer des valeurs.

2.3.1. Création d’objets

Pour créer un objet, il suffit de lui donner un nom. Nous pouvons ensuite attribuer une valeur à cet objet à l’aide d’un *opérateur d’affectation* `<-` (parfois appelé *opérateur d’obtention*). L’opérateur d’affectation est un symbole composite composé d’un symbole “moins que” `<` et d’un trait d’union `-`

raccourci clavier : “option” + “-” sur Mac ; “alt” + “-” sur Windows.

```
mon_obj <- 32
```

Dans le code ci-dessus, nous avons créé un objet appelé `mon_obj` et lui avons attribué la valeur numérique 32 à l’aide de l’opérateur d’affectation (dans notre tête, nous lisons toujours cela comme ‘*mon_obj est 32*’). Vous pouvez également utiliser `=` à la place de `<-` pour assigner des valeurs, mais c’est une mauvaise pratique car cela peut entraîner des confusions plus tard quand vous programmerez en R (voir Chapitre 5) donc nous vous déconseillons d’utiliser cette notation.

Pour afficher la valeur de l’objet, il suffit de taper son nom.

```
mon_obj
```

```
[1] 32
```

Maintenant que nous avons créé cet objet, R le connaît et en gardera la trace pendant la session R en cours. Tous les objets que vous créez sont stockés dans l’espace de travail actuel et vous pouvez visualiser tous les objets de votre espace de travail dans RStudio en cliquant sur l’onglet “Environnement” dans le volet supérieur droit.

Si vous cliquez sur la flèche vers le bas de l’icône “List” (Liste) dans le même volet et que vous passez à l’affichage “Grid” (Grille), RStudio vous présentera un résumé des objets, y compris le type (“numeric” (numérique) - c’est

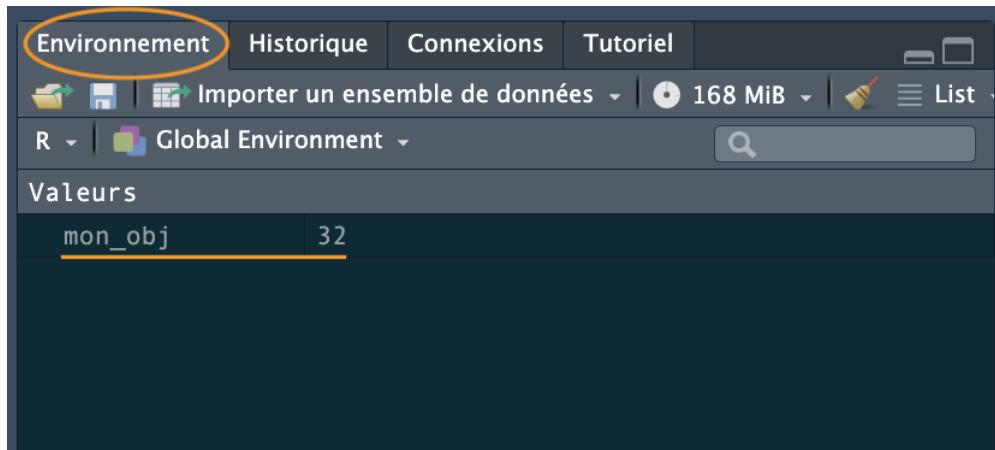


Figure 2.1.: Onglet Environnement RStudio

un nombre), la longueur (une seule valeur dans cet objet), sa taille “physique” et sa valeur (32 dans ce cas). Dans VSCode, allez sur le panneau d’extension R et vous obtiendrez les mêmes informations.

The screenshot shows the RStudio interface with the 'Environnement' tab selected. In the 'Valeurs' section, there is one entry: 'mon_obj' with a value of '32'. The 'Grid' button in the top right corner is highlighted with a yellow oval. The table has columns: Nom, Type, Longueur, Taille, and Valeur. The row for 'mon_obj' is highlighted with a yellow underline.

	Nom	Type	Longueur	Taille	Valeur
■	mon_obj	numeric	1	56 B	32

Figure 2.2.: Onglet Environnement RStudio au format grille

Il existe de nombreux types de valeurs que vous pouvez attribuer à un objet. Par exemple

```
mon_obj2 <- "R c'est trop bien"
```

Nous avons créé un objet appelé `mon_obj2` et lui avons attribué la valeur `R c'est trop bien` qui est une chaîne de caractères. Remarquez que nous avons mis la chaîne de caractères entre guillemets. Si vous oubliez d’utiliser les guillemets, vous recevrez un message d’erreur.

Notre espace de travail contient maintenant les deux objets que nous avons créés jusqu’à présent avec `mon_obj2` de type “character” (caractère).

Pour modifier la valeur d’un objet existant, il suffit de lui réattribuer une nouvelle valeur. Par exemple, pour modifier la valeur de `mon_obj2` de `"R c'est trop bien"` au nombre 1024

Nom	Type	Longueur	Taille	Valeur
mon_obj	numeric	1	56 B	32
mon_obj2	character	1	136 B	"R c'est trop bien"

Figure 2.3.: Onglet Environnement RStudio avec mon_obj2 de type caractère

```
mon_obj2 <- 1024
```

Remarquez que le type est devenu numérique et que la valeur est passée à 1024 dans l'environnement.

Nom	Type	Longueur	Taille	Valeur
mon_obj	numeric	1	56 B	32
mon_obj2	numeric	1	56 B	1024

Figure 2.4.: Onglet Environnement RStudio avec mon_obj2 mis-à-jours en numérique

Une fois que nous avons créé plusieurs objets, nous pouvons faire des choses avec. Par exemple, le code suivant crée un nouvel objet mon_obj3 et lui assigne la valeur de mon_obj ajouté à mon_obj2 soit 1056 ($32 + 1024 = 1056$).

```
mon_obj3 <- mon_obj + mon_obj2
mon_obj3
```

```
[1] 1056
```

Remarquez que pour afficher la valeur de mon_obj3 nous devons également écrire le nom de l'objet. Le code ci-dessus fonctionne parce que les valeurs de mon_obj et mon_obj2 sont numériques (donc des nombres). Si vous essayez de faire ça avec des objets dont les valeurs sont des caractères (**classe character**), vous recevrez une erreur

```
char_obj <- "hello"
char_obj2 <- "world!"
char_obj3 <- char_obj + char_obj2
# Error in char_obj+char_obj2:non-numeric argument to binary operator
```

Le message d'erreur vous indique que l'un ou les deux objets `char_obj` et `char_obj2` n'est pas un nombre et ne peut donc pas être additionné.

Lorsque vous commencez à apprendre R, la gestion des erreurs et des avertissements peut être frustrante car ils sont souvent difficiles à comprendre (qu'est-ce qu'un *argument* ? qu'est-ce qu'un *opérateur binaire* ?). Une façon de trouver plus d'informations sur une erreur particulière est de rechercher une version généralisée du message d'erreur. Pour l'erreur ci-dessus, essayez de rechercher '*non-numeric argument to binary operator error + r*' ou même '*common r error messages*'.

Un autre message d'erreur que vous obtiendrez assez souvent lorsque vous commencerez à utiliser R est `Error` : `object 'XXX' not found` (erreur : objet 'XXX' non trouvé). A titre d'exemple, regardez le code ci-dessous

```
mon_obj <- 48
mon_obj4 <- mon_obj + no_obj
# Error: object 'no_obj' not found
```

R renvoie un message d'erreur parce que nous n'avons pas encore créé (défini) l'objet `no_obj`. Un autre indice qu'il y a un problème avec ce code est que, si vous vérifiez votre environnement, vous verrez que l'objet `mon_obj4` n'a pas été créé.

2.3.2. Nommer les objets

Nommer vos objets est l'une des choses les plus difficiles que vous ferez dans R. Idéalement, les noms de vos objets devraient être courts et informatifs, ce qui n'est pas toujours facile. Si vous devez créer des objets avec plusieurs mots dans leur nom, utilisez un trait de soulignement `_` ou un point `.` entre les mots ou mettez les différents mots en majuscules. Nous préférons le format souligné `_` et n'incluons jamais de majuscules dans les noms (appelé *snake_case*).

```
resume_sortie <- "mon analyse" # recommandé #
resume.sortie <- "mon analyse"
resumeSortie <- "mon analyse"
```

Il y a également quelques limitations lorsqu'il s'agit de donner des noms aux objets. Un nom d'objet ne peut pas commencer par un chiffre ou un point suivi d'un chiffre (ex. `2ma_variable` ou `.2ma_variable`). Vous devez également éviter d'utiliser des caractères non alphanumériques ou des accents dans vos noms d'objets (i.e. `&`, `^`, `/`, `!`, `é`, `è`, etc). De plus, assurez-vous de ne pas nommer vos objets avec des mots réservés (i.e. `TRUE`, `NA`) et ce n'est

jamais une bonne idée de donner à votre objet le même nom qu'une fonction intégrée. Une fonction qui revient plus souvent qu'on ne peut s'en souvenir est :

```
data <- read.table("monfichierdedonnees", header = TRUE)
```

Oui, `data()` est une fonction de R qui permet de charger ou de lister les ensembles de données disponibles dans les paquets.

2.4. Utilisation de fonctions dans R

Jusqu'à présent, nous avons créé des objets simples en assignant directement une valeur unique à un objet. Il est très probable que vous souhaitiez bientôt créer des objets plus compliqués au fur et à mesure que vous aurez de l'expérience sur R et que la complexité de vos tâches augmente. Heureusement, R dispose d'une multitude de fonctions pour vous aider à le faire. Vous pouvez considérer une fonction comme *un objet qui contient une série d'instructions pour effectuer une tâche spécifique*. L'installation de base de R est livrée avec de nombreuses fonctions déjà définies ou vous pouvez augmenter la puissance de R en installant l'un des 10 000 **paquets** actuellement disponibles. Une fois que vous aurez acquis un peu plus d'expérience dans l'utilisation de R, vous voudrez peut-être définir vos propres fonctions pour effectuer des tâches spécifiques à vos objectifs (plus d'informations à ce sujet dans Chapitre 5).

La première fonction que nous allons découvrir est la fonction `c()`. La fonction `c()` est l'abréviation de concaténer et nous l'utilisons pour joindre une série de valeurs et les stocker dans une structure de données appelée **vecteur** (plus d'informations sur les vecteurs dans Chapitre 3).

```
mon_vec <- c(2, 3, 1, 6, 4, 3, 3, 7)
```

Dans le code ci-dessus, nous avons créé un objet appelé `mon_vec` et lui avons assigné une valeur en utilisant la fonction `c()`. Il y a quelques points très importants à noter ici. Premièrement, lorsque vous utilisez une fonction dans R, le nom de la fonction est **toujours** suivi d'une paire de parenthèses `()`, même s'il n'y a rien entre les parenthèses. Deuxièmement, les arguments d'une fonction sont placés à l'intérieur des parenthèses `()` et sont séparés par des virgules `,`. Vous pouvez considérer un argument comme un moyen de personnaliser l'utilisation ou le comportement d'une fonction. Dans l'exemple ci-dessus, les arguments sont les nombres que nous voulons concaténer. Enfin, l'une des choses les plus délicates lorsque vous commencez à utiliser R est de savoir quelle fonction utiliser pour une tâche particulière et comment l'utiliser. Heureusement, chaque fonction est toujours associée à un

document d'aide qui explique comment utiliser la fonction (plus d'informations à ce sujet plus tard Section 2.6) et une recherche rapide sur le web peut également vous aider.

Pour examiner la valeur de notre nouvel objet, nous pouvons simplement taper le nom de l'objet comme nous l'avons fait précédemment

```
mon_vec
```

```
[1] 2 3 1 6 4 3 3 7
```

Maintenant que nous avons créé un vecteur, nous pouvons utiliser d'autres fonctions pour faire des choses utiles avec cet objet. Par exemple, nous pouvons calculer la moyenne, la variance, l'écart-type et le nombre d'éléments de notre vecteur en utilisant les fonctions `mean()`, `var()`, `sd()` et `length()`.

```
mean(mon_vec) # renvoie la moyenne de mon_vec
```

```
[1] 3.625
```

```
var(mon_vec) # renvoie la variance de mon_vec
```

```
[1] 3.982143
```

```
sd(mon_vec) # renvoie l'écart-type de mon_vec
```

```
[1] 1.995531
```

```
length(mon_vec) # renvoie le nombre d'éléments dans mon_vec
```

```
[1] 8
```

Si nous voulons utiliser l'une de ces valeurs plus tard dans notre analyse, il nous suffit d'affecter la valeur obtenue à un autre objet.

```
moyenne_vec <- mean(mon_vec) # renvoie la moyenne de mon_vec  
moyenne_vec
```

```
[1] 3.625
```

Il peut parfois être utile de créer un vecteur contenant une séquence régulière de valeurs par pas de un. Dans ce cas, nous pouvons utiliser un raccourci en utilisant le symbole `:`.

```
ma_seq <- 1:10 # créer une séquence régulière  
ma_seq
```

```
[1] 1 2 3 4 5 6 7 8 9 10
```

```
ma_seq2 <- 10:1 # en ordre décroissant  
ma_seq2
```

```
[1] 10 9 8 7 6 5 4 3 2 1
```

D'autres fonctions utiles pour générer des vecteurs de séquences sont `seq()` et `rep()`. Par exemple, pour générer une séquence de 1 à 5 par pas de 0,5 :

```
ma_seq2 <- seq(from = 1, to = 5, by = 0.5)  
ma_seq2
```

```
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

Ici, nous avons utilisé les arguments `from =` et `to =` pour définir les limites de la séquence et l'argument `by =` pour spécifier l'incrément (les pas) de la séquence. Jouez avec d'autres valeurs pour ces arguments afin de voir leur effet.

L'argument `rep()` vous permet de répliquer (répéter) des valeurs un certain nombre de fois. Pour répéter la valeur '2', 10 fois :

```
ma_seq3 <- rep(2, times = 10) # répète '2', 10 fois
ma_seq3
```

```
[1] 2 2 2 2 2 2 2 2 2 2
```

Vous pouvez également répéter des valeurs non numériques :

```
ma_seq4 <- rep("abc", times = 3) # répète 'abc' 3 fois
ma_seq4
```

```
[1] "abc" "abc" "abc"
```

ou chaque élément d'une série :

```
ma_seq5 <- rep(1:5, times = 3) # répète la série de '1' à '5', 3 fois
ma_seq5
```

```
[1] 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
```

ou des éléments d'une série :

```
ma_seq6 <- rep(1:5, each = 3) # répète chaque élément de la série 3 fois
ma_seq6
```

```
[1] 1 1 1 2 2 2 3 3 3 4 4 4 5 5 5
```

On peut aussi répéter une série non séquentielle :

```
ma_seq7 <- rep(c(3, 1, 10, 7), each = 3) # répète chaque élément de la série 3 fois
ma_seq7
```

```
[1] 3 3 3 1 1 1 10 10 10 7 7 7
```

Notez dans le code ci-dessus comment nous avons utilisé la fonction `c()` à l'intérieur de la fonction `rep()`. L'imbrication de fonctions nous permet de construire des commandes assez complexes à l'intérieur d'une seule ligne de code et est une pratique très courante dans l'utilisation de R. Cependant, il faut faire attention car trop de fonctions imbriquées peuvent rendre votre code difficile à comprendre pour les autres (et pour vous-même dans le futur !). Nous pourrions réécrire le code ci-dessus pour séparer explicitement les deux étapes de la génération de notre vecteur. L'une ou l'autre approche donnera le même résultat, il vous suffit d'utiliser votre propre jugement pour déterminer laquelle est la plus lisible.

```
vec_int <- c(3, 1, 10, 7)
ma_seq7 <- rep(vec_int, each = 3) # répète chaque élément de la série, 3 fois
ma_seq7
```

```
[1] 3 3 3 1 1 1 10 10 10 7 7 7
```

2.5. Travailler avec des vecteurs

Manipuler, résumer et trier des données à l'aide de R est une compétence importante à maîtriser, mais que de nombreuses personnes trouvent un peu déroutante au début. Nous allons voir ici quelques exemples simples utilisant des vecteurs pour illustrer certains concepts importants, mais nous développerons cela plus en détail dans Chapitre 3 où nous verrons des structures de données plus compliquées (et plus utiles).

2.5.1. Extraction d'éléments

Pour extraire (ou indexer ou souscrire) une ou plusieurs valeurs (plus généralement appelées éléments) d'un vecteur, nous utilisons les crochets `[]`. L'approche générale consiste à nommer l'objet à extraire, puis écrire l'indice de l'élément à extraire dans les crochets. Cet indice peut être une position ou le résultat d'un test logique.

Indice de position

Pour extraire des éléments en fonction de leur position, il suffit d'écrire la position à l'intérieur des crochets `[]`. Par exemple, pour extraire la 3e valeur de `mon_vec` :

```
mon_vec # rappelons-nous à quoi mon_vec ressemble
```

```
[1] 2 3 1 6 4 3 3 7
```

```
mon_vec[3] # extrait la 3e valeur
```

```
[1] 1
```

```
# si vous voulez stocker cette valeur dans un autre objet
val_3 <- mon_vec[3]
val_3
```

```
[1] 1
```

Notez que l'indice de position commence à 1 et non à 0 comme dans d'autres langages de programmation (i.e. Python).

Nous pouvons également extraire plusieurs valeurs en utilisant la fonction `c()` à l'intérieur des crochets. Ici, nous extrayons le 1^{er}, le 5^e, le 6^e et le 8^e élément de l'objet `mon_vec` :

```
mon_vec[c(1, 5, 6, 8)]
```

```
[1] 2 4 3 7
```

Nous pouvons également extraire une plage de valeurs à l'aide de la fonction `:`. Pour extraire du 3^e au 8^e élément :

```
mon_vec[3:8]
```

```
[1] 1 6 4 3 3 7
```

2.5.1.1. Indice logique

Une autre façon très utile d'extraire des données d'un vecteur est d'utiliser une expression logique comme indice. Par exemple, pour extraire tous les éléments dont la **valeur** est supérieure à 4 dans le vecteur `mon_vec` :

```
mon_vec[mon_vec > 4]
```

```
[1] 6 7
```

Ici, l'expression logique est `mon_vec > 4` et R n'extraira que les éléments qui satisfont à cette condition logique. Comment cela fonctionne-t-il réellement ? Si nous regardons la sortie de l'expression logique sans les crochets, vous pouvez voir que R renvoie un vecteur contenant soit TRUE soit FALSE qui indique si la condition logique est remplie pour chaque élément. Dans ce cas, seuls les éléments en 4^e et 8^e position renvoient un TRUE car leur valeur est supérieure à 4.

```
mon_vec > 4
```

```
[1] FALSE FALSE FALSE TRUE FALSE FALSE FALSE TRUE
```

Ainsi, ce que R fait en réalité sous le capot est équivalent à :

```
mon_vec[c(FALSE, FALSE, FALSE, TRUE, FALSE, FALSE, FALSE, TRUE)]
```

```
[1] 6 7
```

et seuls les éléments qui sont TRUE seront extraits.

En plus de `<` et `>` vous pouvez également utiliser des opérateurs composites pour augmenter la complexité de vos expressions. Par exemple, l'expression “supérieur ou égal à” est : `>=`. Pour vérifier si une valeur est égale à une autre, nous devons utiliser un double symbole égal `==` et pour vérifier si une valeur est différente de, nous utilisons le symbole `!=` (le symbole `!` signifie “pas”).

```
mon_vec[mon_vec >= 4] # valeurs supérieures ou égales à 4
```

```
[1] 6 4 7
```

```
mon_vec[mon_vec < 4] # valeurs inférieures à 4
```

```
[1] 2 3 1 3 3
```

```
mon_vec[mon_vec <= 4] # valeurs inférieures ou égales à 4
```

[1] 2 3 1 4 3 3

```
mon_vec[mon_vec == 4] # valeurs égales à 4
```

[1] 4

```
mon_vec[mon_vec != 4] # valeurs pas égales à 4
```

[1] 2 3 1 6 3 3 7

Nous pouvons également combiner plusieurs expressions logiques à l'aide d'[expressions booléennes](#). Dans R, l'élément & signifie ET et le symbole | signifie OU. Par exemple, pour extraire des valeurs dans `mon_vec` qui sont inférieures à 6 ET supérieures à 2 :

```
val26 <- mon_vec[mon_vec < 6 & mon_vec > 2]
val26
```

[1] 3 4 3 3

ou extraire des valeurs dans `mon_vec` qui sont supérieures à 6 OU inférieures à 3 :

```
val63 <- mon_vec[mon_vec > 6 | mon_vec < 3]
val63
```

[1] 2 1 7

2.5.2. Remplacement d'éléments

Nous pouvons modifier les valeurs de certains éléments d'un vecteur à l'aide des crochets [] en combinaison avec l'opérateur d'affectation <- . Par exemple, pour remplacer la 4^e valeur de `mon_vec` de 6 à 500 :

```
mon_vec[4] <- 500  
mon_vec
```

```
[1] 2 3 1 500 4 3 3 7
```

Nous pouvons également remplacer plusieurs valeurs ou même remplacer des valeurs sur la base d'une expression logique :

```
# remplacer les 6e et 7e éléments par 100  
mon_vec[c(6, 7)] <- 100  
mon_vec
```

```
[1] 2 3 1 500 4 100 100 7
```

```
# remplacer les éléments inférieurs ou égaux à 4 par 1000  
mon_vec[mon_vec <= 4] <- 1000  
mon_vec
```

```
[1] 1000 1000 1000 500 1000 100 100 7
```

2.5.3. Ordonner les éléments

Outre l'extraction d'éléments particuliers d'un vecteur, il est également possible d'ordonner les valeurs contenues dans un vecteur. Pour trier les valeurs de la plus petite à la plus grande, nous pouvons utiliser la fonction `sort()` :

```
vec_trie <- sort(mon_vec)  
vec_trie
```

```
[1] 7 100 100 500 1000 1000 1000 1000
```

Pour inverser le tri, du plus élevé au plus bas, nous pouvons soit inclure l'option `decreasing = TRUE` lors de l'utilisation de la fonction `sort()` :

```
vec_trie2 <- sort(mon_vec, decreasing = TRUE)
vec_trie2
```

```
[1] 1000 1000 1000 1000 500 100 100 7
```

soit trier d'abord le vecteur à l'aide de la fonction `sort()` puis l'inverser à l'aide de la fonction `rev()`. Il s'agit là d'un autre exemple d'imbrication d'une fonction dans une autre fonction :

```
vec_trie3 <- rev(sort(mon_vec))
vec_trie3
```

```
[1] 1000 1000 1000 1000 500 100 100 7
```

Bien qu'il soit amusant de trier un seul vecteur, il serait peut-être plus utile de trier un vecteur en fonction des valeurs d'un autre vecteur. Pour ce faire, nous devons utiliser la fonction `order()` en combinaison avec `[]`. Pour le démontrer, créons un vecteur appelé `taille` contenant la taille de 5 personnes différentes et un autre vecteur appelé `p_noms` contenant les noms de ces personnes (Joanna mesure 180 cm, Charlotte mesure 155 cm, etc.)

```
taille <- c(180, 155, 160, 167, 181)
taille
```

```
[1] 180 155 160 167 181
```

```
p_noms <- c("Joanna", "Charlotte", "Helen", "Karen", "Amy")
p_noms
```

```
[1] "Joanna"      "Charlotte"   "Helen"       "Karen"       "Amy"
```

Notre objectif est de classer les personnes dans `p_noms` dans l'ordre croissant de leur `taille`. La première chose que nous allons faire est d'utiliser la fonction `order()` avec le vecteur `taille` pour créer un vecteur appelé `taille_ord`

```
taille_ord <- order(taille)
taille_ord
```

```
[1] 2 3 4 1 5
```

OK, que se passe-t-il ici ? La première valeur, 2(n'oubliez pas d'ignorer [1]) doit être lue comme “la plus petite valeur de `taille` est le deuxième élément du vecteur `taille`”. Si nous le vérifions en regardant le vecteur `taille` ci-dessus, nous pouvons voir que le 2^e élément a une valeur de 155, ce qui est la plus petite valeur. La deuxième valeur la plus petite du vecteur `taille` est la 3^e ce qui, après vérification, donne 160 et ainsi de suite. La plus grande valeur de `taille` est la 5^e qui vaut 181. Maintenant que nous avons le vecteur des indices de position des tailles par ordre croissant (`taille_ord`), nous pouvons extraire ces valeurs de notre vecteur `p_noms` dans cet ordre

```
noms_ord <- p_noms[taille_ord]
noms_ord
```

```
[1] "Charlotte" "Helen"      "Karen"       "Joanna"      "Amy"
```

Vous vous demandez probablement à quoi cela peut bien servir. Imaginons que vous disposiez d'un jeu de données contenant deux colonnes et que vous souhaitiez trier chacune d'entre elles. Si vous utilisez simplement `sort()` pour trier chaque colonne séparément, les valeurs de chaque colonne seront dissociées les unes des autres. En utilisant `order()` sur une colonne, un vecteur d'indices de position est créé à partir des valeurs de la colonne dans l'ordre croissant. Ce vecteur peut être utilisé sur la deuxième colonne, en tant qu'indice d'éléments qui renverront un vecteur de valeurs basé sur la première colonne. En toute honnêteté, lorsque vous avez plusieurs vecteurs liés, vous devez utiliser un objet de type `data.frame` (voir Chapitre 3) au lieu de plusieurs vecteurs indépendants.

2.5.4. Vectorisation

L'un des avantages des fonctions R est que la plupart d'entre elles sont vectorisées. Cela signifie que la fonction opère sur tous les éléments d'un vecteur sans qu'il soit nécessaire d'appliquer la fonction à chaque élément séparément. Par exemple, pour multiplier chaque élément d'un vecteur par 5, il suffit d'utiliser la fonction :

```
# créer un vecteur
mon_vec2 <- c(3, 5, 7, 1, 9, 20)
```

```
# multiplier chaque élément par 5  
mon_vec2 * 5
```

```
[1] 15 25 35 5 45 100
```

Ou nous pouvons additionner les éléments de deux vecteurs ou plus :

```
# créer un deuxième vecteur  
mon_vec3 <- c(17, 15, 13, 19, 11, 0)  
  
# additionner les 2 vecteurs  
mon_vec2 + mon_vec3
```

```
[1] 20 20 20 20 20 20
```

```
# multiplier les 2 vecteurs  
mon_vec2 * mon_vec3
```

```
[1] 51 75 91 19 99 0
```

Cependant, vous devez faire attention lorsque vous utilisez la vectorisation avec des vecteurs de longueurs différentes, car R recyclera tranquillement les éléments du vecteur le plus court plutôt que de signaler une erreur.

```
# créer un troisième vecteur  
mon_vec4 <- c(1, 2)  
  
# additionner les 2 vecteurs - recyclage tranquille!  
mon_vec2 + mon_vec4
```

```
[1] 4 7 8 3 10 22
```

2.5.5. Données manquantes

Dans R, les données manquantes sont généralement représentées par un NA qui signifie “Not Available” (Non disponible). Les données peuvent être manquantes pour toute une série de raisons : votre machine est peut-être tombée en panne, vous êtes peut-être tombé en panne, le temps était peut-être trop mauvais pour collecter des données un jour donné, etc. Les données manquantes peuvent être une véritable plaie, tant du point de vue de R que du point de vue statistique. Du point de vue de R, les données manquantes peuvent être problématiques car différentes fonctions traitent les données manquantes de différentes manières. Par exemple, supposons que nous ayons recueilli des relevés de température de l’air pendant 10 jours, mais que notre thermomètre se soit cassé le deuxième et le neuvième jour, de sorte que nous n’avons pas de données pour ces jours-là :

```
temp <- c(7.2, NA, 7.1, 6.9, 6.5, 5.8, 5.8, 5.5, NA, 5.5)
temp
```

```
[1] 7.2 NA 7.1 6.9 6.5 5.8 5.8 5.5 NA 5.5
```

Nous voulons maintenant calculer la température moyenne sur ces jours à l’aide de la fonction `mean()` :

```
temp_moyenne <- mean(temp)
temp_moyenne
```

```
[1] NA
```

Si un vecteur a une valeur manquante, la seule valeur possible à renvoyer lors du calcul d’une moyenne est NA. R ne sait pas que vous souhaitez peut-être ignorer la valeur NA (R ne peut pas lire dans vos pensées - pour l’instant !). Si nous regardons le fichier d’aide (en utilisant `?mean` - voir la section suivante Section 2.6 pour plus de détails) associé à la fonction `mean()` nous pouvons voir qu’il y a un argument `na.rm =` qui prend la valeur FALSE par défaut.

`na.rm` - une valeur logique indiquant si les valeurs NA doivent être supprimées avant le calcul (“`na.remove`”).

Si nous remplaçons cet argument par `na.rm = TRUE` lorsque nous utilisons la fonction `mean()` cela nous permettra d’ignorer les NA lors du calcul de la moyenne :

```
temp_moyenne <- mean(temp, na.rm = TRUE)
temp_moyenne
```

[1] 6.2875

Il est important de noter que les NA n'ont pas été retirés du vecteur temp (ce serait une mauvaise pratique), mais que l'objet `mean()` les a simplement ignorées. Le but de ce qui précède est de souligner comment nous pouvons modifier le comportement par défaut d'une fonction à l'aide d'un argument approprié. Le problème est que toutes les fonctions n'ont pas d'argument `na.rm` = elles peuvent gérer les NA différemment. Cependant, la bonne nouvelle est que chaque fichier d'aide associé à une fonction vous indiquera **toujours** comment les données manquantes sont traitées par défaut.

2.6. Obtenir de l'aide

Ce livre est conçu comme une introduction relativement brève à R et, en tant que tel, vous utiliserez bientôt des fonctions et des paquets qui dépassent le cadre de ce texte d'introduction. Heureusement, l'une des forces de R est son système d'aide complet et facilement accessible, ainsi que la richesse des ressources en ligne où vous pouvez obtenir de plus amples informations.

2.6.1. Aide R

Pour accéder à l'aide intégrée de R et obtenir des informations sur n'importe quelle fonction, il suffit d'utiliser la fonction `help()`. Par exemple, pour ouvrir la page d'aide de notre amie, la fonction `mean()` :

```
help("mean")
```

ou vous pouvez utiliser le raccourci `? devant la fonction :`

```
?mean
```

la page d'aide est affichée dans l'onglet "Aide"(généralement en bas à droite sur RStudio)

Il est vrai que les fichiers d'aide peuvent sembler tout sauf utiles lorsque vous commencez à utiliser R. Cela est probablement dû au fait qu'ils sont écrits de manière très concise et que le langage utilisé est souvent assez technique

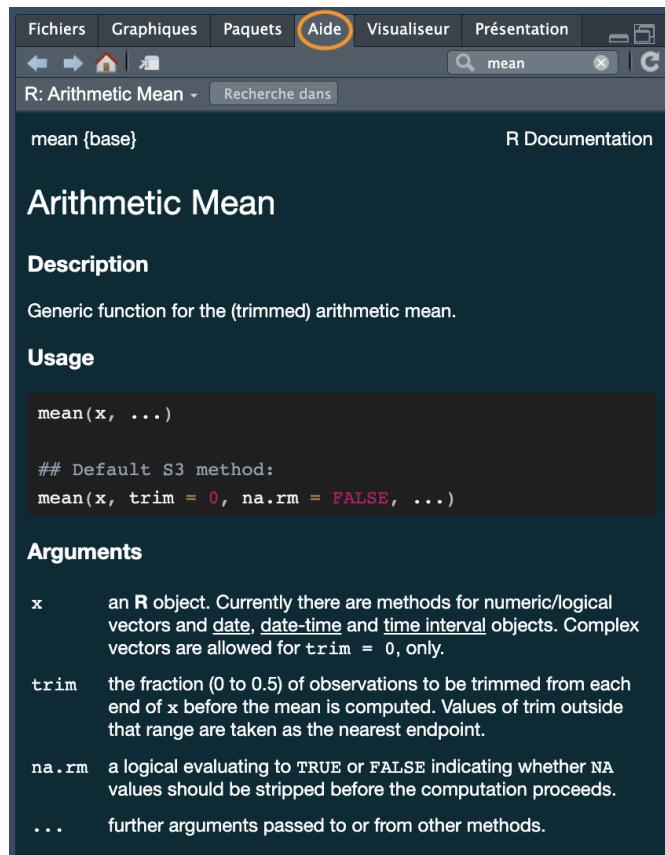


Figure 2.5.: Page d'aide pour la fonction `mean()` dans le panneau Aide sur Rstudio

et plein de jargon. Cela dit, on s'y habitue et, avec le temps, on finit même par apprécier une certaine beauté dans cette brièveté (honnêtement !). L'un des aspects les plus intéressants des fichiers d'aide est qu'ils ont tous une structure très similaire, quelle que soit la fonction. Il est donc facile de naviguer dans le fichier pour trouver exactement ce dont vous avez besoin.

La première ligne du document d'aide contient des informations telles que le nom de la fonction et le paquet d'où elle provient (entre les accolades {}, ici {base} signifie que la fonction `mean()` fait partie des fonctions de base de R). D'autres rubriques fournissent des informations plus spécifiques, telles que

Rubriques	Description de la rubrique
Description :	donne une brève description de la fonction et de ce qu'elle fait.
Usage :	donne le nom des arguments associés à la fonction et les éventuelles valeurs par défaut.
Arguments :	fournit plus de détails sur chaque argument et sur ce qu'il fait.
Details :	donne des détails supplémentaires sur la fonction si nécessaire.
Value :	le cas échéant, indique le type et la structure de l'objet renvoyé par la fonction ou l'opérateur.
See also :	fournit des informations sur d'autres pages d'aide au contenu similaire ou connexe.
Examples :	donne quelques exemples d'utilisation de la fonction.

Les **Examples** (Exemples d'application) sont très utiles, il suffit de les copier et de les coller dans la console pour voir ce qui se passe. Vous pouvez également accéder aux exemples à tout moment en utilisant la fonction `example()` (c'est-à-dire `example("mean")`)

La fonction `help()` est utile si vous connaissez le nom de la fonction. Si vous n'êtes pas sûr du nom, mais que vous vous souvenez d'un mot clé, vous pouvez faire une recherche dans le système d'aide de R à l'aide de la fonction `help.search()`.

```
help.search("mean")
```

ou vous pouvez utiliser le raccourci équivalent ?? :

```
??mean
```

Les résultats de la recherche seront affichés dans RStudio sous l'onglet “Aide” comme précédemment. `help.search()` recherche dans la documentation d'aide, les démonstrations de code et les vignettes de paquet et affiche les résultats sous forme de liens cliquables pour une exploration plus approfondie.

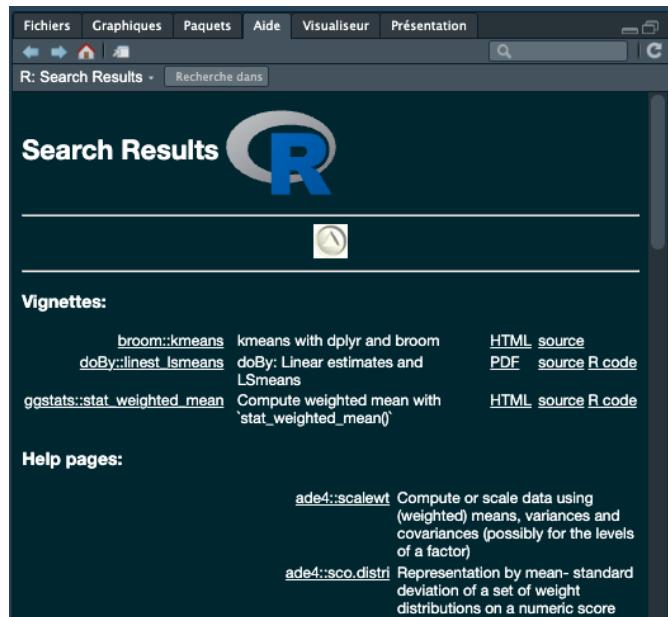


Figure 2.6.: Sortie de la fonction `help.search()` dans Rstudio

Une autre fonction utile est `apropos()`. Cette fonction peut être utilisée pour dresser la liste de toutes les fonctions contenant une chaîne de caractères spécifiée. Par exemple, pour trouver toutes les fonctions avec `mean` dans leur nom :

```
apropos("mean")
```

```
[1] ".colMeans"      ".rowMeans"      "colMeans"       "kmeans"
[5] "mean"           "mean.Date"      "mean.default"   "mean.difftime"
[9] "mean.POSIXct"   "mean.POSIXlt"   "rowMeans"       "weighted.mean"
```

Vous pouvez alors afficher le fichier d'aide de la fonction concernée.

```
help("kmeans")
```

Une autre fonction est `RSiteSearch()` qui vous permet de rechercher des mots-clés et des phrases dans les pages d'aide des fonctions et les vignettes de tous les paquets CRAN. Cette fonction vous permet d'accéder au moteur de recherche du site web de R <https://www.r-project.org/search.html> directement à partir de la console et d'afficher les résultats dans votre navigateur web.

```
RSiteSearch("regression")
```

2.6.2. Autres sources d'aide

Il n'y a jamais eu de meilleur moment pour commencer à apprendre R. Il existe pléthore de ressources en ligne disponibles gratuitement, allant de cours complets à des tutoriels et des listes de diffusion spécifiques à un sujet. Il existe également de nombreuses options payantes si c'est votre truc, mais à moins que vous n'ayez de l'argent à brûler, il n'est vraiment pas nécessaire de dépenser votre argent durement gagné. Vous trouverez ci-dessous quelques ressources que nous avons trouvées utiles.

2.6.2.1. Ressources générales sur les R

- [Projet R](#): Documentation fournie par l'utilisateur
- [Le journal R](#): Journal du projet R pour le calcul statistique
- [Tourbillon](#): Un paquet R qui vous enseigne R de l'intérieur
- [Les antisèches imprimables de RStudio](#)
- [Rseek](#) Une recherche Google personnalisée pour les sites liés à R

2.6.2.2. Obtenir de l'aide

- Cherchez sur internet: utiliser votre moteur de recherche préféré pour chercher les messages d'erreur que vous obtenez. Ce n'est pas de la triche et tout le monde le fait ! Vous serez surpris du nombre de personnes qui ont probablement rencontré le même problème et l'ont résolu.
- [Stack Overflow](#): Il y a plusieurs milliers de questions relatives à R sur Stack Overflow. [Ici](#) sont les plus populaires, classées par vote. Veillez à rechercher des questions similaires avant de poser la vôtre et à inclure un [exemple reproductible](#) afin d'obtenir les conseils les plus utiles. Un exemple reproductible est un exemple minimal qui permet aux personnes qui essaient de vous aider de voir l'erreur elles-mêmes.

2.6.2.3. Ressources R markdown

- [Référence de base pour markdown et markdown R](#)
- [Une bonne référence en markdown](#)
- [Un bon tutoriel de 10 minutes sur le markdown](#)
- [Feuille de contrôle de RStudio sur le format R markdown](#)

- Feuille de référence pour R markdown
- La documentation R markdown incluant un [guide de démarrage](#), a [galerie de démonstrations](#) et plusieurs [articles](#) pour une utilisation plus avancée.
- Le site web de knitr contient de nombreux documents de référence utiles sur le fonctionnement de knitr.

2.6.2.4. Ressources Git et GitHub

- [Happy Git](#): Excellente ressource pour l'utilisation de Git et GitHub
- [Contrôle de version avec RStudio](#): Document RStudio pour l'utilisation du contrôle de version
- [Utiliser Git depuis RStudio](#): Un bon guide en 10 minutes
- [La classe R](#): Guide approfondi de l'utilisation de Git et GitHub avec RStudio

2.6.2.5. Programmation R

- [Programmation R pour la science des données](#): Guide approfondi de la programmation R
- [R pour la science des données](#): Livre fantastique, orienté tidyverse

2.7. Sauvegarder des données dans R

Votre approche de l'enregistrement du travail dans R et RStudio dépend de ce que vous voulez enregistrer. La plupart du temps, la seule chose que vous devrez sauvegarder est le code R de vos scripts. N'oubliez pas que votre script est un enregistrement reproductible de tout ce que vous avez fait. Il vous suffit donc d'ouvrir votre script dans une nouvelle session RStudio et de l'exécuter dans la console R pour revenir à l'endroit où vous vous étiez arrêté.

À moins que vous n'ayez suivi notre suggestion de modifier les paramètres par défaut des projets RStudio (voir Section 1.5), il vous sera demandé si vous souhaitez sauvegarder l'image de votre espace de travail à chaque fois que vous quitterez RStudio. Nous pensons que dans 99,9 % des cas, vous ne souhaitez pas le faire. En commençant avec une session RStudio propre chaque fois que nous revenons à notre analyse, nous pouvons être sûrs d'éviter tout conflit potentiel avec les choses que nous avons faites dans les sessions précédentes.

Cependant, il est parfois utile de sauvegarder les objets que vous avez créés dans R. Par exemple, imaginons que vous créez un objet dont la génération nécessite des heures (voire des jours) de temps de calcul. Il serait extrêmement gênant de devoir attendre tout ce temps à chaque fois que vous revenez sur votre analyse. Cependant, dans ce cas, nous pouvons enregistrer cet objet en tant que fichier externe, .RData que nous pourrons charger dans RStudio la

prochaine fois que nous voudrons l'utiliser. Pour enregistrer un objet dans un fichier .RData vous pouvez utiliser la fonction `save()` (remarquez que nous n'avons pas besoin d'utiliser l'opérateur d'affectation ici) :

```
save(nomDelObjet, file = "nom_du_fichier.RData")
```

ou si vous souhaitez sauvegarder tous les objets de votre espace de travail dans un seul fichier .RData utilisez la fonction `save.image()` :

```
save.image(file = "nom_du_fichier.RData")
```

Pour charger votre .RData dans RStudio, utilisez la fonction `load()` :

```
load(file = "nom_du_fichier.RData")
```

2.8. Paquets R

L'installation de base de R est livrée avec de nombreux paquets utiles. Ces paquets contiennent de nombreuses fonctions que vous utiliserez quotidiennement. Cependant, lorsque vous commencerez à utiliser R pour des projets plus variés (et que votre propre utilisation de R évoluera), vous constaterez qu'il y a un moment où vous aurez besoin d'étendre les capacités de R. Heureusement, des milliers d'utilisateurs de R ont développé du code utile et l'ont partagé sous forme de paquets installables. Vous pouvez considérer un paquet comme une collection de fonctions, de données et de fichiers d'aide rassemblés dans une structure standard bien définie que vous pouvez télécharger et installer dans R. Ces paquets peuvent être téléchargés à partir de diverses sources, mais les plus populaires sont les suivantes [CRAN](#), [Bioconductor](#) et [GitHub](#). Actuellement, le CRAN héberge plus de 15 000 paquets et est le dépôt officiel des paquets R fournis par les utilisateurs. Bioconductor fournit des logiciels libres orientés vers la bioinformatique et héberge plus de 1800 paquets R. GitHub est un site web qui héberge des dépôts git pour toutes sortes de logiciels et de projets (pas seulement R). Souvent, les versions de développement de pointe des paquets R sont hébergées sur GitHub, donc si vous avez besoin de toutes les nouvelles fonctionnalités, cela peut être une option. Cependant, l'inconvénient potentiel de l'utilisation de la version de développement d'un paquet R est qu'elle peut ne pas être aussi stable que la version hébergée sur CRAN (elle est en cours de développement !) et que la mise à jour des paquets ne sera pas automatique.

2.8.1. Utilisation des paquets

Une fois que vous avez installé un paquet sur votre ordinateur, vous ne pouvez pas l'utiliser immédiatement. Pour utiliser un paquet, vous devez d'abord le charger à l'aide de la fonction `library()`. Par exemple, pour charger le paquet `remotes` 📦 que vous avez installé précédemment :

```
library(remotes)
```

La fonction `library()` chargera également tous les paquets supplémentaires nécessaires et pourra afficher des informations supplémentaires sur les paquets dans la console. Il est important de savoir que chaque fois que vous démarrez une nouvelle session R (ou que vous restaurez une session précédemment sauvegardée), vous devez charger les paquets que vous utiliserez. Nous avons tendance à mettre tous nos `library()` nécessaires à notre analyse en tête de nos scripts R afin de les rendre facilement accessibles et de pouvoir les compléter au fur et à mesure du développement de notre code. Si vous essayez d'utiliser une fonction sans avoir préalablement chargé le paquet R correspondant, vous recevrez un message d'erreur indiquant que R n'a pas pu trouver la fonction. Par exemple, si vous essayez d'utiliser la fonction `install_github()` sans charger le paquet `remotes` 📦 en premier lieu, vous obtiendrez l'erreur suivante :

```
install_github("tidyverse/dplyr")  
  
# Error in install_github("tidyverse/dplyr") :  
#   could not find function "install_github"
```

Il peut parfois être utile d'utiliser une fonction sans utiliser au préalable la fonction `library()`. Si, par exemple, vous n'utilisez qu'une ou deux fonctions dans votre script et que vous ne souhaitez pas charger toutes les autres fonctions d'un paquet, vous pouvez accéder directement à la fonction en spécifiant le nom du paquet, suivi de deux points (2 fois) ::, puis du nom de la fonction :

```
remotes::install_github("tidyverse/dplyr")
```

C'est ainsi que nous avons pu utiliser la fonction `install()` et `install_github()` ci-dessous sans charger les paquets au préalable `BiocManager` 📦 et `remotes` 📦. La plupart du temps, nous recommandons d'utiliser la fonction `library()`.

2.8.2. Installation des paquets R

2.8.2.1. Paquets CRAN

Pour installer un paquet à partir du CRAN, vous pouvez utiliser la fonction `install.packages()`. Par exemple, si vous voulez installer le paquet `remotes`  entrez le code suivant dans la Console (note : vous aurez besoin d'une connexion internet fonctionnelle pour effectuer cette opération) :

```
install.packages("remotes", dependencies = TRUE)
```

Il vous sera peut-être demandé de choisir un miroir CRAN, sélectionnez simplement ‘0-cloud’ ou un miroir proche de votre localisation. L'argument `dependencies = TRUE` permet de s'assurer que les paquets supplémentaires nécessaires seront également installés.

Il est conseillé de mettre régulièrement à jour les paquets déjà installés afin de bénéficier des nouvelles fonctionnalités et des corrections de bogues. Pour mettre à jour les paquets CRAN, vous pouvez utiliser la commande `update.packages()` (vous aurez besoin d'une connexion internet pour cela) :

```
update.packages(ask = FALSE)
```

L'argument `ask = FALSE` évite d'avoir à confirmer chaque téléchargement de paquet, ce qui peut être fastidieux si de nombreux paquets sont installés.

2.8.2.2. Paquets Bioconductor

Pour installer des paquets de Bioconductor, le processus est [un peu différent](#). Vous devez d'abord installer le paquet `BiocManager` . Vous ne devez le faire qu'une seule fois, sauf si vous réinstallez ou mettez à jour R.

```
install.packages("BiocManager", dependencies = TRUE)
```

Une fois que `BiocManager` a été installé, vous pouvez soit installer tous les paquets “de base” de Bioconductor avec la commande :

```
BiocManager::install()
```

ou installer des paquets spécifiques tels que le `GenomicRanges`  et `edgeR`  :

```
BiocManager::install(c("GenomicRanges", "edgeR"))
```

Pour mettre à jour les paquets de Bioconductor, il suffit d'utiliser la commande `BiocManager::install()` à nouveau :

```
BiocManager::install(ask = FALSE)
```

Là encore, vous pouvez utiliser l'argument `ask = FALSE` pour éviter d'avoir à confirmer chaque téléchargement de paquet.

2.8.2.3. Paquets GitHub

Il existe plusieurs options pour installer les paquets hébergés sur GitHub. La méthode la plus efficace est sans doute d'utiliser la fonction `install_github()` du paquet `remotes` (vous avez installé ce paquet précédemment, Section 2.8.2.1). Avant d'utiliser la fonction, vous devez connaître le nom d'utilisateur GitHub du propriétaire du répertoire ainsi que le nom du répertoire. Par exemple, la version de développement de `dplyr` de Hadley Wickham est hébergée sur le compte GitHub de tidyverse et porte le nom de répertoire “`dplyr`” (recherchez simplement “github `dplyr`”). Pour installer cette version depuis GitHub, utilisez :

```
remotes::install_github("tidyverse/dplyr")
```

Le moyen le plus sûr (à notre connaissance) de mettre à jour un paquet installé depuis GitHub est de le réinstaller en utilisant la commande ci-dessus.

Chapitre 3

Données

Jusqu'à présent, vous avez créé des données relativement simples dans R et les avez stockées sous forme de vecteur (Section 2.4). Cependant, la plupart d'entre vous (si ce n'est tous) disposeront d'ensembles de données beaucoup plus complexes provenant de vos diverses expériences et enquêtes, qui vont bien au-delà de ce qu'un vecteur peut gérer. Apprendre comment R traite les différents types de données et structures de données, comment importer vos données dans R et comment manipuler et résumer vos données sont quelques-unes des compétences les plus importantes que vous devrez maîtriser.

Dans ce chapitre, nous passerons en revue les principaux types de données dans R et nous nous concentrerons sur certaines des structures de données les plus courantes. Nous verrons également comment importer des données dans R à partir d'un fichier externe, comment manipuler et résumer des données et enfin comment exporter des données de R vers un fichier externe.

3.1. Types de données

Il est important de comprendre les différents types de données et la manière dont R traite ces données. La tentation est grande d'ignorer ces détails techniques, mais attention, cela peut se retourner contre vous très vite si vous ne faites pas attention. Nous avons déjà vu un exemple (Section 2.3.1) de cela lorsque nous avons essayé (et échoué) d'ajouter deux objets de type caractères ensemble en utilisant l'opérateur +.

R dispose de six types de données de base : numérique (`numeric`), entier (`integer`), logique (`logical`), complexe (`complex`) et caractère (`character`). Les plus attentifs d'entre vous remarqueront que nous n'avons listé ici que cinq types de données, le dernier étant le type de données brut (`raw`), que nous n'aborderons pas car il n'est pas utile dans 99,99 % des cas. Nous n'aborderons pas non plus les nombres complexes, mais nous vous laisserons **imaginer** cette partie !

- **Numérique** sont des nombres contenant une décimale. En fait, il peut également s'agir de nombres entiers, mais nous n'y reviendrons pas.
- **Entiers (integer)** sont des nombres entiers (sans virgule).
- **Logique** prennent la valeur de TRUE ou FALSE. Il existe également un autre type de logique appelé NA pour représenter les valeurs manquantes.
- **Caractère** sont des chaînes de caractères comme un (ou plusieurs) mot(s). Un type particulier de chaîne de caractères est le *facteur* qui a des attributs supplémentaires (comme des niveaux ou un ordre). Nous reviendrons sur les facteurs plus tard.

R est (généralement) capable de distinguer automatiquement les différentes classes de données en fonction de leur nature et du contexte dans lequel elles sont utilisées, bien que vous deviez garder à l'esprit que R ne peut pas vraiment lire dans vos pensées et que vous devrez peut-être lui indiquer explicitement comment vous souhaitez traiter un type de données. Vous pouvez connaître le type (ou la classe) de n'importe quel objet en utilisant la fonction `class()` pour connaître le type (ou la classe) d'un objet.

```
num <- 2.2  
class(num)
```

```
[1] "numeric"
```

```
char <- "hello"  
class(char)
```

```
[1] "character"
```

```
logi <- TRUE  
class(logi)
```

```
[1] "logical"
```

Vous pouvez également demander si un objet appartient à une classe spécifique à l'aide d'un test logique. Le test `is.[classOfData]()` renvoie soit un TRUE ou un FALSE.

```
is.numeric(num)
```

```
[1] TRUE
```

```
is.character(num)
```

```
[1] FALSE
```

```
is.character(char)
```

```
[1] TRUE
```

```
is.logical(logi)
```

```
[1] TRUE
```

Il peut parfois être utile de pouvoir changer la classe d'une variable à l'aide de la fonction `as.[className]()` bien qu'il faille être prudent car vous pourriez obtenir des résultats inattendus (voir ce qui se passe ci-dessous lorsque nous essayons de convertir une chaîne de caractères en une chaîne numérique).

```
# convertir un objet numérique en caractère  
class(num)
```

```
[1] "numeric"
```

```
num_char <- as.character(num)  
num_char
```

```
[1] "2.2"
```

```
class(num_char)
```

```
[1] "character"
```

```
# convertir un objet caractère en numérique !
class(char)
```

```
[1] "character"
```

```
char_num <- as.numeric(char)
```

Warning: NAs introduced by coercion

Voici un tableau récapitulatif de certaines des fonctions de test logique et de coercion à votre disposition.

Type de test	Test logique	Coercition
Caractère	<code>is.character</code>	<code>as.character</code>
Numérique	<code>is.numeric</code>	<code>as.numeric</code>
Logique	<code>is.logical</code>	<code>as.logical</code>
Facteur	<code>is.factor</code>	<code>as.factor</code>
Complexe	<code>is.complex</code>	<code>as.complex</code>

3.2. Structures de données

Maintenant que vous connaissez certaines des classes de données les plus importantes de R, examinons quelques-unes des principales structures dont nous disposons pour stocker ces données.

3.2.1. Scalaires et vecteurs

Le type de structure de données le plus simple est sans doute le vecteur. Vous avez déjà été initié aux vecteurs dans la Section 2.4, certains des vecteurs que vous avez créés ne contenaient qu'une seule valeur (longueur 1) on appelle ça un scalaires. Les vecteurs peuvent contenir des nombres, des caractères, des facteurs ou des logiques, mais la chose essentielle à retenir est que tous les éléments à l'intérieur d'un vecteur doivent être de la même classe. En d'autres termes, les vecteurs peuvent contenir des nombres, des caractères ou des logiques, mais pas des mélanges de ces types de données. Il existe une exception importante à cette règle : vous pouvez inclure des NA (rappelez-vous qu'il s'agit d'un type spécial de logique) pour indiquer les données manquantes dans les vecteurs contenant d'autres types de données.

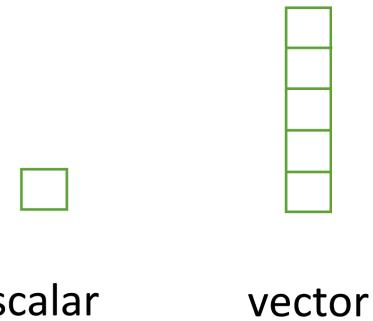


Figure 3.1.: Structure de données scalaires et vecteurs

3.2.2. Matrices et tableaux (array)

La matrice est une autre structure de données utile, utilisée dans de nombreuses disciplines telles que l’écologie des populations et les statistiques théoriques et appliquées. Une matrice est simplement un vecteur doté d’attributs supplémentaires appelés dimensions. Les tableaux (array) ne sont que des matrices multidimensionnelles. Là encore, les matrices et les tableaux doivent contenir des éléments appartenant tous à la même classe de données.

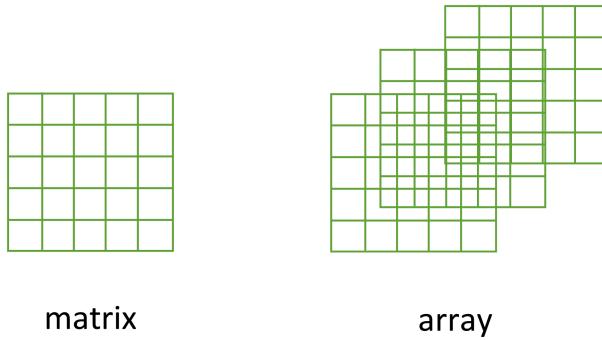


Figure 3.2.: Structure de données matrices et tableaux (array)

Un moyen pratique de créer une matrice ou un tableau est d’utiliser la fonction `matrix()` et `array()` respectivement. Ci-dessous, nous allons créer une matrice à partir d’une séquence de 1 à 16 en quatre lignes (`nrow = 4`) et la remplir par rangée (`byrow = TRUE`) plutôt que par colonne, comme c’est le cas par défaut. Lors de l’utilisation de l’option `array()` il faut définir les dimensions à l’aide de la fonction `dim =` dans notre cas, *2 lignes, 4 colonnes* dans 2 *matrices* différentes :

```
ma_mat <- matrix(1:16, nrow = 4, byrow = TRUE)  
ma_mat
```

```
[,1] [,2] [,3] [,4]  
[1,] 1 2 3 4  
[2,] 5 6 7 8  
[3,] 9 10 11 12  
[4,] 13 14 15 16
```

```
mon_tabl <- array(1:16, dim = c(2, 4, 2))  
mon_tabl
```

, , 1

```
[,1] [,2] [,3] [,4]  
[1,] 1 3 5 7  
[2,] 2 4 6 8
```

, , 2

```
[,1] [,2] [,3] [,4]  
[1,] 9 11 13 15  
[2,] 10 12 14 16
```

Il est parfois utile de définir les noms des lignes et des colonnes de votre matrice, mais ce n'est pas obligatoire. Pour ce faire, utilisez la fonction `rownames()` et `colnames()` :

```
rownames(ma_mat) <- c("A", "B", "C", "D")  
colnames(ma_mat) <- c("a", "b", "c", "d")  
ma_mat
```

```
a b c d  
A 1 2 3 4  
B 5 6 7 8
```

```
C 9 10 11 12
D 13 14 15 16
```

Une fois que vous avez créé vos matrices, vous pouvez faire des choses utiles avec elles et, comme vous pouvez vous y attendre, R dispose de nombreuses fonctions intégrées pour effectuer des opérations sur les matrices. Certaines des plus courantes sont présentées ci-dessous. Par exemple, pour transposer une matrice, nous utilisons la fonction de transposition `t()` :

```
ma_mat_t <- t(ma_mat)
ma_mat_t
```

```
A B C D
a 1 5 9 13
b 2 6 10 14
c 3 7 11 15
d 4 8 12 16
```

Pour extraire les éléments diagonaux d'une matrice et les stocker sous forme de vecteur, nous pouvons utiliser la fonction `diag()` :

```
ma_mat_diag <- diag(ma_mat)
ma_mat_diag
```

```
[1] 1 6 11 16
```

Les opérations habituelles d'addition, de multiplication, etc. de matrices peuvent être effectuées. Notez l'utilisation de la fonction `%*%` pour effectuer une multiplication matricielle.

```
mat.1 <- matrix(c(2, 0, 1, 1), nrow = 2)
mat.1 # Notez que la matrice a été remplie par colonne par défaut
```

```
[,1] [,2]
[1,]    2    1
[2,]    0    1
```

```
mat.2 <- matrix(c(1, 1, 0, 2), nrow = 2)  
mat.2
```

```
[,1] [,2]  
[1,] 1 0  
[2,] 1 2
```

```
mat.1 + mat.2 # Addition de matrices
```

```
[,1] [,2]  
[1,] 3 1  
[2,] 1 3
```

```
mat.1 * mat.2 # Produit élément par élément
```

```
[,1] [,2]  
[1,] 2 0  
[2,] 0 2
```

```
mat.1 %*% mat.2 # Multiplication de matrice
```

```
[,1] [,2]  
[1,] 3 2  
[2,] 1 2
```

3.2.3. Listes

La prochaine structure de données que nous examinerons rapidement est la liste. Alors que les vecteurs et les matrices sont contraints de contenir des données du même type, les listes peuvent stocker des mélanges de types de données. En fait, nous pouvons même stocker d'autres structures de données telles que des vecteurs et des tableaux à l'intérieur d'une liste ou même avoir une liste de liste. Il s'agit donc d'une structure de données très flexible, idéale pour stocker des données irrégulières ou non rectangulaires (voir Chapitre 5 pour un exemple).

Pour créer une liste, nous pouvons utiliser la fonction `list()`. Notez que les trois éléments de la liste sont de classes différentes (caractère, logique et numérique) et de longueurs différentes.

```
list_1 <- list(
  c("black", "yellow", "orange"),
  c(TRUE, TRUE, FALSE, TRUE, FALSE, FALSE),
  matrix(1:6, nrow = 3)
)
```

```
list_1
```

```
[[1]]
```

```
[1] "black" "yellow" "orange"
```

```
[[2]]
```

```
[1] TRUE TRUE FALSE TRUE FALSE FALSE
```

```
[[3]]
```

```
[,1] [,2]
```

```
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

Les éléments de la liste peuvent être nommés lors de la construction de la liste :

```
list_2 <- list(
  couleurs = c("black", "yellow", "orange"),
  evaluation = c(TRUE, TRUE, FALSE, TRUE, FALSE, FALSE),
  temps = matrix(1:6, nrow = 3)
)
```

```
list_2
```

```
$couleurs
```

```
[1] "black" "yellow" "orange"
```

```
$evaluation
```

```
[1] TRUE TRUE FALSE TRUE FALSE FALSE
```

```
$temps
[,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

ou après la création de la liste à l'aide de la fonction `names()` :

```
names(list_1) <- c("couleurs", "evaluation", "temps")
list_1
```

```
$couleurs
[1] "black"  "yellow" "orange"

$evaluation
[1] TRUE  TRUE FALSE TRUE FALSE FALSE

$temps
[,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

3.2.4. Jeu de données

De loin la structure de données la plus couramment utilisée : le jeu de données (`data frame`). Un jeu de données est un objet bidimensionnel puissant composé de **lignes** et de **colonnes** qui ressemble superficiellement à une matrice. Toutefois, alors que les matrices ne peuvent contenir que des données du même type, les jeux de données peuvent contenir un mélange de différents types de données. En règle générale, dans un jeu de données, chaque ligne correspond à une observation individuelle et chaque colonne correspond à une variable mesurée ou enregistrée différente. Cette configuration est peut-être familière à ceux d'entre vous qui utilisent LibreOffice Calc ou Microsoft Excel pour gérer et stocker leurs données. Il peut être utile de penser que les jeux de données sont essentiellement constitués d'un ensemble de vecteurs (colonnes), chaque vecteur contenant son propre type de données, mais le type de données peut être différent d'un vecteur à l'autre.

Par exemple, le jeu de données ci-dessous contient les résultats d'une expérience visant à déterminer l'effet des soins parentaux (avec ou sans) chez les licornes (*Unicornus magnificens*) sur la croissance des petits sous trois régimes de disponibilité alimentaire différents. Le jeu de données contient 8 variables (colonnes) et chaque ligne représente une licorne individuelle. Les variables `p_care` et `food` sont des facteurs (variables [catégoriques](#)). La variable `p_care` a 2 niveaux (`care` et `no_care`) et la variable `food` a 3 niveaux (`low`, `medium` et `high`). Les variables `height`, `weight`, `mane_size` et `fluffyness` sont numériques et la variable `horn_rings` est un nombre entier représentant le nombre d'anneaux sur la corne. Bien que la variable `block` possède des valeurs numériques, celles-ci n'ont pas vraiment d'ordre et pourraient également être traitées comme un facteur (i.e. elles auraient également pu être appelées A et B).

Table 3.2.: Données importées sur les licornes

<code>p_care</code>	<code>food</code>	<code>block</code>	<code>height</code>	<code>weight</code>	<code>mane_size</code>	<code>fluffyness</code>	<code>horn_rings</code>
care	medium	1	7.5	7.62	11.7	31.9	1
care	medium	1	10.7	12.14	14.1	46.0	10
care	medium	1	11.2	12.76	7.1	66.7	10
care	medium	1	10.4	8.78	11.9	20.3	1
care	medium	1	10.4	13.58	14.5	26.9	4
care	medium	1	9.8	10.08	12.2	72.7	9
no_care	low	2	3.7	8.10	10.5	60.5	6
no_care	low	2	3.2	7.45	14.1	38.1	4
no_care	low	2	3.9	9.19	12.4	52.6	9
no_care	low	2	3.3	8.92	11.6	55.2	6
no_care	low	2	5.5	8.44	13.5	77.6	9
no_care	low	2	4.4	10.60	16.2	63.3	6

Il y a deux choses importantes à garder à l'esprit à propos des jeux de données. Ces types d'objets sont connus sous le nom de données rectangulaires (ou données ordonnées), car chaque colonne doit comporter le même nombre d'observations. Aussi, toute donnée manquante doit être enregistrée sous la forme d'un NA comme nous l'avons fait pour nos vecteurs.

Nous pouvons construire un jeu de données à partir d'objets de données existants, tels que des vecteurs, à l'aide de la fonction `data.frame()`. À titre d'exemple, créons trois vecteurs `p_taille`, `p_poids` et `p_noms` et incluons tous ces vecteurs dans un jeu de données appelé `dataaf`.

```
p_taille <- c(180, 155, 160, 167, 181)
p_poids <- c(65, 50, 52, 58, 70)
p_noms <- c("Joanna", "Charlotte", "Helen", "Karen", "Amy")

dataaf <- data.frame(taille = p_taille,
                     poids = p_poids,
                     noms = p_noms)

dataaf
```

	taille	poids	noms
	180	65	Joanna
	155	50	Charlotte
	160	52	Helen
	167	58	Karen
	181	70	Amy

Vous remarquerez que chacune des colonnes est nommée avec le nom de la variable que nous avons fourni lorsque nous avons utilisé la fonction `data.frame()` lorsque nous avons utilisé la fonction. Il semble également que la première colonne du cadre de données soit une série de nombres allant de 1 à 5. En fait, il ne s'agit pas vraiment d'une colonne, mais du nom de chaque ligne. Nous pouvons le vérifier en demandant à R de renvoyer les dimensions du jeu de données `dataaf` à l'aide de la fonction `dim()`. Nous constatons qu'il y a 5 lignes et 3 colonnes.

```
dim(dataaf) # 5 lignes et 3 colonnes
```

```
[1] 5 3
```

Une autre fonction très utile que nous utilisons en permanence est `str()` qui renvoie un résumé compact de la structure de l'objet data frame (ou de tout autre objet).

```
str(dataaf)
```

```
'data.frame': 5 obs. of 3 variables:
 $ taille: num 180 155 160 167 181
```

```
$ poids : num  65 50 52 58 70
$ noms   : chr  "Joanna" "Charlotte" "Helen" "Karen" ...
```

La fonction `str()` nous donne les dimensions du jeu de données et nous rappelle que `dataf` est un `data.frame`. Il énumère également toutes les variables (colonnes) contenues dans le jeu de données, nous indique le type de données qu'elles contiennent et leurs 5 premières valeurs. Nous copions souvent ce résumé pour le mettre dans le scripts R avec des commentaires au début de chaque ligne afin de pouvoir nous y référer facilement lors de l'écriture du code.

Nous vous avons montré comment commenter les blocs dans RStudio dans la Section 1.7.

Notez également que R a automatiquement décidé que notre `p_noms` doit être un caractère (`chr`) lors de la création du jeu de données. La question de savoir si c'est une bonne idée ou non dépendra de la manière dont vous souhaitez utiliser cette variable dans des analyses ultérieures. Si nous décidons que ce n'est pas une bonne idée, nous pouvons modifier le comportement par défaut du `data.frame()` en incluant l'argument `stringsAsFactors = TRUE`. Nos chaînes de caractères seront maintenant automatiquement converties en facteurs.

```
p_taille <- c(180, 155, 160, 167, 181)
p_poids <- c(65, 50, 52, 58, 70)
p_noms <- c("Joanna", "Charlotte", "Helen", "Karen", "Amy")

dataf <- data.frame(
  taille = p_taille, poids = p_poids, noms = p_noms,
  stringsAsFactors = TRUE
)
str(dataf)
```

```
'data.frame': 5 obs. of 3 variables:
 $ taille: num 180 155 160 167 181
 $ poids : num 65 50 52 58 70
 $ noms  : Factor w/ 5 levels "Amy","Charlotte",...: 4 2 3 5 1
```

3.3. Importer des données

Bien que la création de jeux de données à partir de structures de données existantes soit extrêmement utile, l'approche de loin la plus courante consiste à créer un jeu de données en important des données à partir d'un fichier externe.

Pour ce faire, vos données doivent être correctement formatées et enregistrées dans un format de fichier que R est capable de reconnaître. Heureusement pour nous, R est capable de reconnaître une grande variété de formats de fichiers, même si, en réalité, vous n'en utiliserez probablement que deux ou trois régulièrement.

3.3.1. Enregistrement de fichiers à importer

La méthode la plus simple pour créer un fichier de données à importer dans R consiste à saisir vos données dans une feuille de calcul à l'aide de Microsoft Excel ou LibreOffice Calc et à l'enregistrer sous la forme d'un fichier délimité par des virgules. Nous préférons LibreOffice Calc car c'est un logiciel libre, indépendant de plate-forme et gratuit, mais MS Excel convient également (mais voir [ici](#) pour quelques problèmes). Voici les données de l'expérience sur les licornes dont nous avons parlé précédemment, affichées dans LibreOffice. Si vous souhaitez suivre en même temps que le livre, vous pouvez télécharger le fichier de données ('*unicorn.xlsx*') à partir de l'Annexe A.

	A	B	C	D	E	F	G	H	I	J	K	L
1	p_care	food	block	height	weight	mane_size	fluffyness	horn_rings				
2	tip	medium	1	7.5	7.62	11.7	31.9	1				
3	tip	medium	1	10.7	12.14	14.1	46	10				
4	tip	medium	1	11.2	12.76	17.1	66.7	10				
5	tip	medium	1	10.4	8.76	11.9	20.3	1				
6	tip	medium	1	10.4	13.58	14.5	26.9	4				
7	tip	medium	1	9.8	10.08	12.2	72.7	9				
8	tip	medium	1	6.9	10.11	13.2	43.1	7				
9	tip	medium	1	9.4	10.28	14	28.5	6				
10	tip	medium	2	10.4	10.48	10.6	57.8	5				
11	tip	medium	2	12.3	13.48	16.1	36.9	8				
12	tip	medium	2	10.4	13.18	11.1	56.8	12				
13	tip	medium	2	11	11.56	12.6	31.3	6				
14	tip	medium	2	7.1	8.16	29.6	9.7	2				
15	tip	medium	2	6	11.22	13	16.4	3				
16	tip	medium	2	9	10.2	10.8	90.1	6				
17	tip	medium	2	4.5	12.55	13.4	14.4	6				
18	tip	high	1	12.6	18.66	18.6	54	9				
19	tip	high	1	10	18.07	16.9	90.5	3				
20	tip	high	1	10	13.29	15.8	142.7	12				
21	tip	high	1	8.5	14.33	13.2	91.4	5				
22	tip	high	1	14.1	19.12	13.1	113.2	13				
23	tip	high	1	10.1	15.49	12.6	77.2	12				

Figure 3.3.: Données Licornes (Unicorn) dans LibreOffice Calc

Pour ceux d'entre vous qui ne connaissent pas le format de fichier délimité par des **virgules**, cela signifie simplement que les données des différentes colonnes sont séparées par le caractère “,” et sont généralement enregistrées dans un fichier portant l'extension “.csv” (*Comma-Separated Values*).

Pour enregistrer une feuille de calcul en tant que fichier délimité par des virgules (CSV) dans LibreOffice Calc, sélectionnez Fichier -> Enregistrer sous... dans le menu principal. Vous devrez spécifier l'emplacement où vous souhaitez enregistrer votre fichier dans l'option “Enregistrer dans le dossier” et le nom du fichier dans l'option “Nom”. Dans le menu déroulant situé au-dessus du bouton “Enregistrer”, remplacez l'option par défaut “Tous les formats” par “Texte CSV (.csv)”.

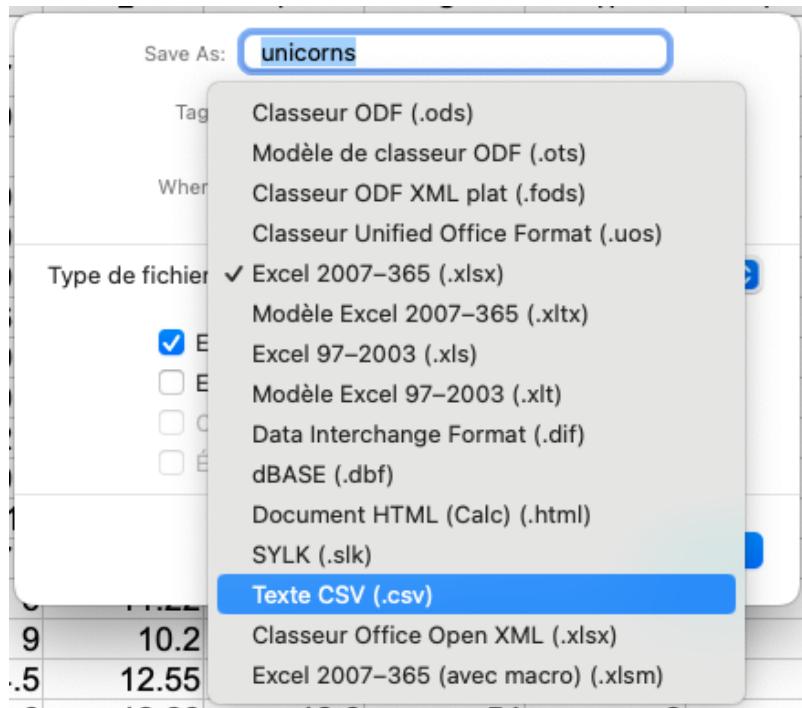


Figure 3.4.: Choisir le format csv lors de l'enregistrement avec LibreOffice Calc

Cliquez sur le bouton Enregistrer, puis sélectionnez l’option “Utiliser le format CSV texte”. Cliquez sur OK pour enregistrer le fichier.

Il y a quelques points à prendre en compte lors de l’enregistrement des fichiers à importer dans R, qui vous faciliteront la vie à long terme. Les titres de vos colonnes (si vous en avez) doivent être courts et informatifs. Évitez également les espaces dans vos titres de colonnes en les remplaçant par un trait de soulignement _ ou un point . (c'est-à-dire remplacer `taille de la criniere` par `taille_de_la_criniere` ou `taille.de.la.criniere`) et d'éviter d'utiliser des caractères spéciaux (par ex. `aire (mm^2)`), des accents (par ex. écrire `criniere` au lieu de `crinière`) ou des majuscules pour vous simplifier la vie. N'oubliez pas que si vous avez des données manquantes dans votre cadre de données (cellules vides), vous devez utiliser un champ NA pour les représenter. Cela permettra de conserver un cadre de données ordonné.

3.3.2. Fonctions d’importation

Une fois que vous avez enregistré votre fichier de données dans un format approprié, nous pouvons maintenant lire ce fichier dans R. La fonction la plus utilisée pour importer des données dans R est la fonction `read.table()` (nous examinerons d’autres solutions plus loin dans ce chapitre). La fonction `read.table()` est une fonction très flexible qui dispose d’un grand nombre d’arguments (voir `?read.table`), mais elle est assez simple à utiliser. Importons le fichier délimité par des virgules appelé `unicorns.csv` qui contient les données que nous avons vues précédemment

dans ce chapitre (Section 3.2.4) que l'on va assigner à un objet appelé `licornes`. Le fichier est situé dans un dossier `data` (“données”) qui est lui-même situé dans notre répertoire racine (Section 1.4). La première ligne des données contient les noms des variables (colonnes). Pour utiliser les `read.table()` pour importer ce fichier

```
licornes <- read.table(  
  file = "data/unicorns.csv", header = TRUE, sep = ",", dec = ".",
  stringsAsFactors = TRUE  
)
```

Il y a quelques points à noter à propos de la commande ci-dessus. Tout d'abord, le chemin d'accès au fichier et le nom du fichier (y compris l'extension) doivent être placés entre guillemets simples ou doubles (c-à-d. le `data/unicorns.csv`), car la fonction `read.table()` attend une chaîne de caractères. Si votre répertoire de travail est déjà le répertoire qui contient le fichier, il n'est pas nécessaire d'inclure le chemin d'accès complet au fichier, mais seulement le nom du fichier. Dans l'exemple ci-dessus, le chemin d'accès au fichier est séparé par une simple barre oblique `/`. Cela fonctionne quel que soit le système d'exploitation que vous utilisez et nous vous recommandons de vous en tenir à cela. Cependant, les utilisateurs de Windows peuvent être plus familiers avec la notation de la barre oblique inverse simple et si vous voulez continuer à l'utiliser, vous devrez l'inclure en tant que barre oblique inverse double.

Avertissement

Notez cependant que la notation de la double barre oblique inverse fonctionnera **pas** sur les ordinateurs utilisant les systèmes d'exploitation Mac OSX ou Linux. Nous le déconseillons donc fortement car il n'est pas reproductible

Les `header = TRUE` spécifie que la première ligne de vos données contient les noms des variables (c.-à-d. `food`, `block` etc.) Si ce n'est pas le cas, vous pouvez spécifier `header = FALSE` (en fait, c'est la valeur par défaut, vous pouvez donc omettre complètement cet argument). L'argument `sep = ","` indique à R quel est le délimiteur de fichier.

D'autres arguments utiles sont `dec =` et `na.strings =`. Les `dec =` permet de modifier le caractère par défaut `(.)` utilisé par défaut pour le point décimal. Ceci est utile si vous êtes dans un pays où les décimales sont généralement représentées par une virgule (c.-à-d. `dec = ", "`). L'argument `na.strings =` vous permet d'importer des données où les valeurs manquantes sont représentées par un symbole autre que `NA`. Cela peut être assez courant si vous importez des données à partir d'un autre logiciel statistique tel que Minitab, qui représente les valeurs manquantes avec le symbole `*` (`na.strings = "*"`).

Une série de fonctions prédéfinies sont disponibles dans `read.table()` pour définir des options spécifiques au format. Mais nous pouvons aussi simplement utiliser `read.csv()` pour lire un fichier csv, avec la séparation “;” et “.” pour les décimales. Dans les pays où “;” est utilisé pour les décimales, les fichiers csv utilisent “;” comme séparateur. Dans ce cas, l'utilisation de `read.csv2()` serait nécessaire. Lorsque l'on travaille avec des fichiers délimités par des tabulations, les fonctions `read.delim()` et `read.delim2()` peuvent être utilisées avec “.” et “;” comme décimales respectivement.

Après avoir importé nos données dans R, pour voir le contenu du jeu de données, il suffit de taper le nom de l'objet comme nous l'avons fait précédemment. **MAIS** avant de faire cela, réfléchissez à la raison pour laquelle vous faites cela. Si votre jeu de données est autre chose que minuscule, tout ce que vous allez faire, c'est remplir votre Console de données. Ce n'est pas comme si vous pouviez facilement vérifier s'il y a des erreurs ou si vos données ont été importées correctement. Une bien meilleure solution consiste à utiliser notre vieil ami, la fonction `str()` pour obtenir un résumé compact et informatif de votre base de données.

```
str(licornes)
```

```
'data.frame': 96 obs. of 8 variables:
 $ p_care    : Factor w/ 2 levels "care","no_care": 1 1 1 1 1 1 1 1 1 1 ...
 $ food       : Factor w/ 3 levels "high","low","medium": 3 3 3 3 3 3 3 3 3 3 ...
 $ block      : int  1 1 1 1 1 1 1 2 2 ...
 $ height     : num  7.5 10.7 11.2 10.4 10.4 9.8 6.9 9.4 10.4 12.3 ...
 $ weight     : num  7.62 12.14 12.76 8.78 13.58 ...
 $ mane_size  : num  11.7 14.1 7.1 11.9 14.5 12.2 13.2 14 10.5 16.1 ...
 $ fluffyness: num  31.9 46 66.7 20.3 26.9 72.7 43.1 28.5 57.8 36.9 ...
 $ horn_rings: int  1 10 10 1 4 9 7 6 5 8 ...
```

Ici, nous voyons que `licornes` est un objet “`data.frame`” qui contient 96 lignes et 8 variables (colonnes). Chacune des variables est répertoriée avec sa classe de données et les 10 premières valeurs. Comme nous l'avons mentionné précédemment dans ce chapitre, il peut être très pratique de copier/coller ces données dans votre script R sous la forme d'un bloc de commentaires afin de pouvoir s'y référer ultérieurement.

Notez également que vos variables de type chaîne de caractères (`care` et `food`) ont été importées en tant que facteurs parce que nous avons utilisé l'argument `stringsAsFactors = TRUE`. Si ce n'est pas ce que vous voulez, vous pouvez utiliser l'argument `stringsAsFactors = FALSE` ou à partir de la version 4.0.0 de R, vous pouvez simplement ne pas utiliser cet argument car `stringsAsFactors = FALSE` est la valeur par défaut.

Nous pouvons donc aussi extraire le jeu de données depuis un fichier texte (.txt) grâce à la fonction `read.delim()` :

```
licornes <- read.delim(file = "data/unicorns.txt")
str(licornes)
```



```
'data.frame': 96 obs. of 8 variables:
 $ p_care    : chr  "care" "care" "care" "care" ...
 $ food       : chr  "medium" "medium" "medium" "medium" ...
 $ block      : int  1 1 1 1 1 1 1 2 2 ...
 $ height     : num  7.5 10.7 11.2 10.4 10.4 9.8 6.9 9.4 10.4 12.3 ...
 $ weight     : num  7.62 12.14 12.76 8.78 13.58 ...
 $ mane_size  : num  11.7 14.1 7.1 11.9 14.5 12.2 13.2 14 10.5 16.1 ...
 $ fluffyness: num  31.9 46 66.7 20.3 26.9 72.7 43.1 28.5 57.8 36.9 ...
 $ horn_rings: int  1 10 10 1 4 9 7 6 5 8 ...
```

Si nous voulons simplement voir les noms de nos variables (colonnes) dans le jeu de données, nous pouvons utiliser l'argument `names()` qui renverra un vecteur de caractères contenant les noms des variables.

```
names(licornes)
```



```
[1] "p_care"      "food"        "block"        "height"       "weight"
[6] "mane_size"   "fluffyness"  "horn_rings"
```

Vous pouvez même importer des feuilles de calcul de MS Excel ou d'autres logiciels de statistiques directement dans R, mais nous vous conseillons d'éviter cette méthode dans la mesure du possible, car elle ne fait qu'ajouter une couche d'incertitude entre vous et vos données. À notre avis, il est presque toujours préférable d'exporter vos feuilles de calcul sous forme de fichiers délimités par des tabulations (.txt) ou des virgules (.csv), puis de les importer dans R à l'aide de l'un des fonctions dérivées de `read.table()` (c-à-d. `read.delim()`, `read.csv()`, etc.). Si vous tenez absolument à importer directement des données à partir d'un autre logiciel, vous devrez installer le paquet `foreign` qui contient des fonctions permettant d'importer des fichiers Minitab, SPSS, Stata et SAS. Pour les feuilles de calcul MS Excel et LO Calc, quelques paquets peuvent être utilisés.

3.3.3. Frustrations courantes liées à l'importation

Il est assez courant d'obtenir un tas de messages d'erreur vraiment frustrants lorsque l'on commence à importer des données dans R. Le plus courant est sans doute

```
Error in file(file, "rt") : cannot open the connection
In addition: Warning message:
In file(file, "rt") :
  cannot open file 'unicorns.txt': No such file or directory
```

Ce message d'erreur vous indique que R ne peut pas trouver le fichier que vous essayez d'importer. Il apparaît généralement pour l'une ou l'autre des raisons suivantes (ou pour toutes !). La première est que vous avez fait une erreur dans l'orthographe du nom de fichier ou du chemin d'accès au fichier. Une autre erreur fréquente est d'avoir oublié d'inclure l'extension du fichier dans le nom du fichier (par ex. .txt). Enfin, le fichier ne se trouve pas à l'endroit indiqué ou vous avez utilisé un chemin d'accès incorrect. L'utilisation de projets RStudio (Section 1.5) et d'une structure de répertoire logique (Section 1.4) permet d'éviter (limiter) ce type d'erreurs.

Une autre erreur très fréquente est d'oublier d'inclure l'élément `header = TRUE` lorsque la première ligne des données contient des noms de variables. Par exemple, si nous omettons cet argument lorsque nous importons notre fichier `unicorns.txt` tout semble correct au début (pas de message d'erreur au moins).

```
licornes_erreur <- read.table(file = "data/unicorns.txt", sep = "\t")
```

mais lorsque nous jetons un coup d'œil à notre jeu de données en utilisant `str()`

```
str(licornes_erreur)

'data.frame':   97 obs. of  8 variables:
 $ V1: chr  "p_care" "care" "care" "care" ...
 $ V2: chr  "food"   "medium" "medium" "medium" ...
 $ V3: chr  "block"  "1"    "1"    "1"    ...
 $ V4: chr  "height" "7.5"  "10.7" "11.2" ...
 $ V5: chr  "weight" "7.62" "12.14" "12.76" ...
 $ V6: chr  "mane_size" "11.7" "14.1" "7.1" ...
 $ V7: chr  "fluffyness" "31.9" "46"   "66.7" ...
 $ V8: chr  "horn_rings" "1"   "10"   "10"   ...
```

Nous constatons un problème évident, toutes nos variables ont été importées en tant que facteurs et sont nommées V1, V2, V3 ... V8. Le problème vient du fait que nous n'avons pas dit au `read.table()` que la première ligne contient les noms des variables et donc elle les traite comme des données. Dès que nous avons une chaîne de caractères dans l'une de nos variable, R les traite comme des données de type caractère (rappelez-vous que tous les éléments d'un vecteur doivent contenir le même type de données (Section 3.2.1)).

Ce n'est qu'un argument de plus pour utiliser `read.csv()` ou `read.delim()` avec des valeurs par défaut appropriées pour les arguments.

3.3.4. Autres options d'importation

Il existe de nombreuses autres fonctions permettant d'importer des données à partir d'une variété de sources et de formats. La plupart de ces fonctions sont contenues dans des paquets que vous devez installer avant de les utiliser. Vous trouverez ci-dessous une liste des paquets et des fonctions les plus utiles.

Les paquets `fread()` du paquet `data.table` 📦 est idéale pour importer rapidement et efficacement de grands fichiers de données (beaucoup plus rapidement que la fonction `read.table()`). L'un des aspects les plus intéressants de la fonction `fread()` est qu'elle détecte automatiquement la plupart des arguments que vous devriez normalement spécifier (comme `sep = etc.`) Une choses à prendre en compte cependant est que la fonction `fread()` renverra un objet de type `data.table` et non `data.frame` comme ce serait le cas avec la fonction `read.table()`. Ce n'est généralement pas un problème, car vous pouvez passer un objet `data.table` à n'importe quelle fonction n'acceptant normalement que des objets `data.frame`. Pour en savoir plus sur les différences entre `data.table` et `data.frame` voir [ici](#).

```
library(data.table)
toutes_donnees <- fread(file = "data/unicorns.txt")
```

Diverses fonctions du paquet `readr` 📦 sont également très efficaces pour lire des fichiers de données volumineux. Le paquet `readr` 📦 fait partie de la collection de paquet ‘tidyverse’ et fournit de nombreuses fonctions équivalentes à celles de la version de base de R pour l'importation de données. Les fonctions de `readr` sont utilisées de la même manière que les fonctions `read.table()` ou `read.csv()` et la plupart des arguments sont les mêmes (voir `?readr::read_table` pour plus de détails). Il existe cependant quelques différences. Par exemple, lors de l'utilisation de l'option `read_table()` l'argument `header = TRUE` est remplacé par `col_names = TRUE` et la fonction renvoie un objet de classe `tibble` qui est l'équivalent dans le tidyverse d'un objet de classe `data.frame` (voir [ici](#) pour les différences).

Avertissement

Attention ! certain.e.s de vos modèles ou fonctions, etc. peuvent ne pas fonctionner ou fonctionner différemment à cause de ça (vérifier/reconvertir votre jeu de données au format `data.frame` peut vous économiser du temps de débogage).

```
library(readr)

# importation de fichiers délimités par des espaces blancs
toutes_donnees <- read_table(file = "data/unicorns.txt", col_names = TRUE)

# importation de fichiers délimités par des virgules
toutes_donnees <- read_csv(file = "data/unicorns.csv")

# importation de fichiers délimités par des tabulations
toutes_donnees <- read_delim(file = "data/unicorns.txt", delim = "\t")

# ou utilisez
toutes_donnees <- read_tsv(file = "data/unicorns.txt")
```

Si votre fichier de données est énorme, les paquets `ff` et `bigmemory` peuvent être utiles car ils contiennent tous deux des fonctions d'importation capables de stocker des données volumineuses de manière efficace en termes de mémoire. Pour en savoir plus sur ces fonctions [ici](#) et [ici](#).

3.4. Manipuler des jeux de données

Maintenant que vous avez réussi à importer vos données dans R depuis un fichier externe, la prochaine étape est de faire quelque chose d'utile avec nos données. La manipulation de données est une compétence fondamentale que vous devrez développer et avec laquelle vous devrez vous sentir à l'aise, car vous en ferez probablement beaucoup au cours de n'importe quel projet. La bonne nouvelle, c'est que R est particulièrement efficace pour manipuler, résumer et visualiser les données. La manipulation de données (souvent connue sous le nom de “data wrangling” ou “munging”) en R peut sembler un peu intimidante au début pour un nouvel utilisateur, mais si vous suivez quelques règles logiques simples, vous prendrez rapidement le coup de main, surtout avec un peu de pratique.

Rappelons la structure du jeu de données `licornes` que nous avons importée dans la section précédente :

```
licornes <- read.table(file = "data/unicorns.txt", header = TRUE, sep = "\t")
str(licornes)
```

```
'data.frame': 96 obs. of 8 variables:
 $ p_care     : chr  "care" "care" "care" "care" ...
 $ food        : chr  "medium" "medium" "medium" "medium" ...
 $ block       : int  1 1 1 1 1 1 1 1 2 2 ...
 $ height      : num  7.5 10.7 11.2 10.4 10.4 9.8 6.9 9.4 10.4 12.3 ...
 $ weight      : num  7.62 12.14 12.76 8.78 13.58 ...
 $ mane_size   : num  11.7 14.1 7.1 11.9 14.5 12.2 13.2 14 10.5 16.1 ...
 $ fluffyness: num  31.9 46 66.7 20.3 26.9 72.7 43.1 28.5 57.8 36.9 ...
 $ horn_rings: int  1 10 10 1 4 9 7 6 5 8 ...
```

Pour accéder aux données de n'importe quelle variable (colonne) de notre jeu de données, nous pouvons utiliser la notation \$. Par exemple, pour accéder à la variable `height` dans le jeu de données `licornes` on écrit `licornes$height`. Cela indique à R que la variable `height` est contenue dans le jeu de données `licornes`.

```
licornes$height
```

```
[1] 7.5 10.7 11.2 10.4 10.4 9.8 6.9 9.4 10.4 12.3 10.4 11.0 7.1 6.0 9.0
[16] 4.5 12.6 10.0 10.0 8.5 14.1 10.1 8.5 6.5 11.5 7.7 6.4 8.8 9.2 6.2
[31] 6.3 17.2 8.0 8.0 6.4 7.6 9.7 12.3 9.1 8.9 7.4 3.1 7.9 8.8 8.5
[46] 5.6 11.5 5.8 5.6 5.3 7.5 4.1 3.5 8.5 4.9 2.5 5.4 3.9 5.8 4.5
[61] 8.0 1.8 2.2 3.9 8.5 8.5 6.4 1.2 2.6 10.9 7.2 2.1 4.7 5.0 6.5
[76] 2.6 6.0 9.3 4.6 5.2 3.9 2.3 5.2 2.2 4.5 1.8 3.0 3.7 2.4 5.7
[91] 3.7 3.2 3.9 3.3 5.5 4.4
```

Cette opération renvoie un vecteur des données de la variable `height`. Si nous le souhaitons, nous pouvons assigner ce vecteur à un autre objet et faire d'autres choses avec, comme calculer une moyenne ou obtenir un résumé de la variable à l'aide de la fonction `summary()` :

```
f_taille <- licornes$height
mean(f_taille)
```

```
[1] 6.839583
```

```
summary(f_taille)
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
1.200	4.475	6.450	6.840	9.025	17.200

Ou si nous ne voulons pas créer un objet supplémentaire, nous pouvons utiliser des fonctions “à la volée” pour afficher uniquement la valeur dans la console.

```
mean(licornes$height)
```

```
[1] 6.839583
```

```
summary(licornes$height)
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
1.200	4.475	6.450	6.840	9.025	17.200

Tout comme nous l'avons fait avec les vecteurs (Section 2.5), nous pouvons également accéder aux données contenues dans le jeu de données en utilisant les crochets []. Cependant, au lieu d'utiliser un seul index, nous devons maintenant utiliser deux index, l'un pour spécifier les lignes et l'autre pour les colonnes. Pour ce faire, nous pouvons utiliser la notation `mes_donnees[ligne, colonne]` où `ligne` et `colonne` sont des indices et `mes_donnees` est le nom du jeu de données. Une fois encore, comme pour nos vecteurs, nos index peuvent être positionnels ou résulter d'un test logique.

3.4.1. Index positionnels

Pour utiliser les index positionnels, il suffit d'écrire la position des lignes et des colonnes que l'on veut extraire à l'intérieur des crochets []. Par exemple, si, pour une raison quelconque, nous voulons extraire la première valeur (1^{ère} ligne) de la variable `height` (4^e colonne) :

```
licornes[1, 4]
```

```
[1] 7.5
```

```
# on peut obtenir le même résultat avec cette notation :  
licornes$height[1]
```

```
[1] 7.5
```

Nous pouvons également extraire les valeurs de plusieurs lignes ou colonnes en spécifiant ces index sous forme de vecteurs à l'intérieur des crochets []. Pour extraire les 10 premières lignes et les 4 premières colonnes, il suffit de fournir un vecteur contenant une séquence de 1 à 10 pour l'index des lignes (1:10) et un vecteur de 1 à 4 pour l'index des colonnes (1:4) :

```
licornes[1:10, 1:4]
```

p_care	food	block	height
care	medium	1	7.5
care	medium	1	10.7
care	medium	1	11.2
care	medium	1	10.4
care	medium	1	10.4
care	medium	1	9.8
care	medium	1	6.9
care	medium	1	9.4
care	medium	2	10.4
care	medium	2	12.3

Si les lignes et les colonnes ne sont pas séquentielles, nous pouvons fournir des vecteurs de positions à l'aide de la fonction `c()`. Pour extraire les 1^{ère}, 5^e, 12^e et 30^e lignes des 1^{ère}, 3^e, 6^e et 8^e colonnes :

```
licornes[c(1, 5, 12, 30), c(1, 3, 6, 8)]
```

	p_care	block	mane_size	horn_rings
1	care	1	11.7	1
5	care	1	14.5	4
12	care	2	12.6	6

	p_care	block	mane_size	horn_rings
30	care	2	11.6	5

Tout ce que nous faisons dans les deux exemples ci-dessus est de créer des vecteurs de positions pour les lignes et les colonnes que nous voulons extraire. Pour ce faire, nous avons utilisé les compétences que nous avons développées dans la Section 2.4 lorsque nous avons généré des vecteurs à l'aide de la fonction `c()` ou en utilisant la notation `:`.

Mais qu'en est-il si nous voulons extraire toutes les lignes ou toutes les colonnes ? Il serait extrêmement fastidieux de devoir générer des vecteurs pour toutes les lignes ou pour toutes les colonnes. Heureusement, R dispose d'un raccourci. Si vous ne spécifiez pas d'index de ligne ou de colonne dans les crochets `[]` R interprète cela comme signifiant que vous voulez toutes les lignes ou toutes les colonnes. Par exemple, pour extraire les 4 premières lignes et toutes les colonnes du jeu de données `licornes` :

```
licornes[1:4, ]
```

p_care	food	block	height	weight	mane_size	fluffyness	horn_rings
care	medium	1	7.5	7.62	11.7	31.9	1
care	medium	1	10.7	12.14	14.1	46.0	10
care	medium	1	11.2	12.76	7.1	66.7	10
care	medium	1	10.4	8.78	11.9	20.3	1

ou toutes les lignes et les 3 premières colonnes ¹.

```
unicorns[, 1:3]
```

	p_care	food	block
1	care	medium	1
2	care	medium	1
3	care	medium	1
4	care	medium	1
5	care	medium	1

¹Par exemple le modèle de Ricker peut, en partie, être construit à l'aide de boucles.

	p_care	food	block
92	no_care	low	2
93	no_care	low	2
94	no_care	low	2
95	no_care	low	2
96	no_care	low	2

Nous pouvons même utiliser des index de position négatifs pour exclure certaines lignes et colonnes. Par exemple, extrayons toutes les lignes à l'exception des 85 premières et toutes les colonnes à l'exception de la 4^e, 7^e et 8^e. Remarquez que nous devons utiliser `-()` lorsque nous générions nos vecteurs de position de ligne. Si nous avions simplement utilisé `-1:85` cela générerait en fait une séquence régulière de -1 à 85, ce qui n'est pas ce que nous voulons (nous pouvons bien sûr utiliser `-1:-85`).

```
licornes[-(1:85), -c(4, 7, 8)]
```

	p_care	food	block	weight	mane_size
86	no_care	low	1	6.01	17.6
87	no_care	low	1	9.93	12.0
88	no_care	low	1	7.03	7.9
89	no_care	low	2	9.10	14.5
90	no_care	low	2	9.05	9.6
91	no_care	low	2	8.10	10.5
92	no_care	low	2	7.45	14.1
93	no_care	low	2	9.19	12.4
94	no_care	low	2	8.92	11.6
95	no_care	low	2	8.44	13.5
96	no_care	low	2	10.60	16.2

Outre l'utilisation d'un index de position pour extraire des colonnes (variables) particulières, nous pouvons également nommer les variables directement en utilisant les crochets []. Par exemple, extrayons les 5 premières lignes des variables `care`, `food` et `mane_size`. Au lieu d'utiliser `licornes[1:5, c(1, 2, 6)]` nous pouvons utiliser :

```
licornes[1:5, c("p_care", "food", "mane_size")]
```

	p_care	food	mane_size
	care	medium	11.7
	care	medium	14.1
	care	medium	7.1
	care	medium	11.9
	care	medium	14.5

En général, on préférera utiliser cette méthode plutôt que l’index positionnel pour sélectionner les colonnes, car elle nous donnera toujours ce que nous voulons même si nous avons changé l’ordre des colonnes dans notre jeu de données pour une raison quelconque.

3.4.2. Index logiques

Tout comme nous l’avons fait avec les vecteurs, nous pouvons également extraire des données de notre jeu de données sur la base d’un test logique. Nous pouvons utiliser tous les opérateurs logiques que nous avons utilisés dans les exemples des vecteurs. Si ceux-ci vous ont échappé, jetez un coup d’œil à la Section 2.5.1.1 pour vous rafraîchir la mémoire. Extrayons toutes les lignes pour lesquelles la variable `height` a une valeur supérieure à 12 et extrayons toutes les colonnes par défaut (rappelez-vous, si vous n’incluez pas d’index de colonne après la virgule, cela signifie toutes les colonnes).

```
grandes_cornes <- licornes[licornes$height > 12, ]
grandes_cornes
```

	p_care	food	block	height	weight	mane_size	fluffyness	horn_rings
10	care	medium	2	12.3	13.48	16.1	36.9	8
17	care	high	1	12.6	18.66	18.6	54.0	9
21	care	high	1	14.1	19.12	13.1	113.2	13
32	care	high	2	17.2	19.20	10.9	89.9	14
38	care	low	1	12.3	11.27	13.7	28.7	5

Remarquez dans le code ci-dessus qu'il faut utiliser l'élément `licornes$height` pour le test logique. Si nous nommions simplement la variable `height` sans le nom du jeu de données, nous recevrions une erreur nous indiquant que R ne peut pas trouver la variable `height`. La raison en est que la variable `height` n'existe qu'à l'intérieur du jeu de données `licornes` vous devez donc indiquer à R où elle se trouve exactement.

```
grandes_cornes <- licornes[height > 12, ]  
Error in ` [.data.frame` (licornes, height > 12, ) :  
  object 'height' not found
```

Comment cela fonctionne-t-il ? Le test logique est le suivant `licornes$height > 12` et R n'extraira que les lignes qui satisfont à cette condition logique. Si nous regardons la sortie de la seule condition logique, vous pouvez voir qu'elle renvoie un vecteur contenant `TRUE` si `height` est supérieur à 12 et `FALSE` si `height` n'est pas supérieur à 12.

```
licornes$height > 12
```

```
[1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE  
[13] FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE TRUE FALSE FALSE FALSE  
[25] FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE  
[37] FALSE TRUE FALSE  
[49] FALSE  
[61] FALSE  
[73] FALSE  
[85] FALSE FALSE
```

Notre indice de ligne est donc un vecteur contenant soit `TRUE` ou `FALSE` et seulement les lignes qui sont `TRUE` sont sélectionnées.

D'autres opérateurs couramment utilisés sont présentés ci-dessous :

```
licornes[licornes$height >= 6, ] # valeurs supérieures ou égales à 6  
  
licornes[licornes$height <= 6, ] # valeurs inférieures ou égales à 6  
  
licornes[licornes$height == 8, ] # valeurs égales à 8
```

```
licornes[licornes$height != 8, ] # valeurs différentes de 8
```

Nous pouvons également extraire des lignes en fonction de la valeur d'une chaîne de caractères ou d'un niveau de facteur. Extrayons toutes les lignes pour lesquelles la valeur de la variable `food` est égal à `high` (là encore, nous extrairons toutes les colonnes). Remarquez que la double égalité `==` doit être utilisé pour un test logique et que la chaîne de caractères doit être placée entre guillemets simples ou doubles (c.-à-d. `"high"`).

```
nourriture_bcp <- licornes[licornes$food == "high", ]
rbind(head(nourriture_bcp, n = 10), tail(nourriture_bcp, n = 10))
```

	p_care	food	block	height	weight	mane_size	fluffyness	horn_rings
17	care	high	1	12.6	18.66	18.6	54.0	9
18	care	high	1	10.0	18.07	16.9	90.5	3
19	care	high	1	10.0	13.29	15.8	142.7	12
20	care	high	1	8.5	14.33	13.2	91.4	5
21	care	high	1	14.1	19.12	13.1	113.2	13
22	care	high	1	10.1	15.49	12.6	77.2	12
23	care	high	1	8.5	17.82	20.5	54.4	3
24	care	high	1	6.5	17.13	24.1	147.4	6
25	care	high	2	11.5	23.89	14.3	101.5	12
26	care	high	2	7.7	14.77	17.2	104.5	4
71	no_care	high	1	7.2	15.21	15.9	135.0	14
72	no_care	high	1	2.1	19.15	15.6	176.7	6
73	no_care	high	2	4.7	13.42	19.8	124.7	5
74	no_care	high	2	5.0	16.82	17.3	182.5	15
75	no_care	high	2	6.5	14.00	10.1	126.5	7
76	no_care	high	2	2.6	18.88	16.4	181.5	14
77	no_care	high	2	6.0	13.68	16.2	133.7	2
78	no_care	high	2	9.3	18.75	18.4	181.1	16
79	no_care	high	2	4.6	14.65	16.7	91.7	11
80	no_care	high	2	5.2	17.70	19.1	181.1	8

Où nous pouvons extraire toutes les lignes où `food` est différent de `medium` (en utilisant `!=`) et ne sélectionner que les colonnes 1 à 4.

```
nourriture_pas_moyenne <- licornes[licornes$food != "medium", 1:4]  
rbind(head(nourriture_pas_moyenne, n = 10), tail(nourriture_pas_moyenne, n = 10))
```

	p_care	food	block	height
17	care	high	1	12.6
18	care	high	1	10.0
19	care	high	1	10.0
20	care	high	1	8.5
21	care	high	1	14.1
22	care	high	1	10.1
23	care	high	1	8.5
24	care	high	1	6.5
25	care	high	2	11.5
26	care	high	2	7.7
87	no_care	low	1	3.0
88	no_care	low	1	3.7
89	no_care	low	2	2.4
90	no_care	low	2	5.7
91	no_care	low	2	3.7
92	no_care	low	2	3.2
93	no_care	low	2	3.9
94	no_care	low	2	3.3
95	no_care	low	2	5.5
96	no_care	low	2	4.4

Nous pouvons accroître la complexité de nos tests logiques en les combinant avec des [des expressions booléennes](#) comme nous l'avons fait pour les objets vectoriels. Par exemple, pour extraire toutes les lignes où la taille (`height`) est supérieure ou égale à 6 ET `food` est égal à `medium` ET `care` est égal à `no_care` nous combinons une série d'expressions logiques avec le symbole `&` :

```
faible_soins_non_taille6 <- licornes[licornes$height >= 6 &
                                         licornes$food == "medium" &
                                         licornes$p_care == "no_care", ]
```

`faible_soins_non_taille6`

	p_care	food	block	height	weight	mane_size	fluffyness	horn_rings
51	no_care	medium	1	7.5	13.60	13.6	122.2	11
54	no_care	medium	1	8.5	10.04	12.3	113.6	4
61	no_care	medium	2	8.0	11.43	12.6	43.2	14

Pour extraire des lignes sur la base d'une expression booléenne “OR” (“OU”), nous pouvons utiliser le symbole `|`. Extrayons toutes les lignes où la taille (`height`) est supérieure à 12,3 OU inférieure à 2,2.

```
taille2.2_12.3 <- licornes[licornes$height > 12.3 | licornes$height < 2.2, ]
```

`taille2.2_12.3`

	p_care	food	block	height	weight	mane_size	fluffyness	horn_rings
17	care	high	1	12.6	18.66	18.6	54.0	9
21	care	high	1	14.1	19.12	13.1	113.2	13
32	care	high	2	17.2	19.20	10.9	89.9	14
62	no_care	medium	2	1.8	10.47	11.8	120.8	9
68	no_care	high	1	1.2	18.24	16.6	148.1	7
72	no_care	high	1	2.1	19.15	15.6	176.7	6
86	no_care	low	1	1.8	6.01	17.6	46.2	4

Une autre méthode pour sélectionner des parties d'un jeu de données sur la base d'une expression logique consiste à utiliser la fonction `subset()` au lieu des crochets `[]`. L'avantage d'utiliser `subset()` est qu'il n'est plus nécessaire d'utiliser le symbole `$` pour spécifier des variables à l'intérieur du jeu de données, car le premier argument de la fonction est le nom du jeu de données à subdiviser. L'inconvénient est que `subset()` est moins flexible que la notation `[]`.

```
soins_moy_2 <- subset(licornes, p_care == "care" & food == "medium" & block == 2)
soins_moy_2
```

	p_care	food	block	height	weight	mane_size	fluffyness	horn_rings
9	care	medium	2	10.4	10.48	10.5	57.8	5
10	care	medium	2	12.3	13.48	16.1	36.9	8
11	care	medium	2	10.4	13.18	11.1	56.8	12
12	care	medium	2	11.0	11.56	12.6	31.3	6
13	care	medium	2	7.1	8.16	29.6	9.7	2
14	care	medium	2	6.0	11.22	13.0	16.4	3
15	care	medium	2	9.0	10.20	10.8	90.1	6
16	care	medium	2	4.5	12.55	13.4	14.4	6

Et si vous ne voulez que certaines colonnes, vous pouvez utiliser l'argument `select = :`

```
uni_p_care <- subset(licornes, p_care == "care" & food == "medium" & block == 2,
  select = c("p_care", "food", "mane_size"))
)
uni_p_care
```

	p_care	food	mane_size
9	care	medium	10.5
10	care	medium	16.1
11	care	medium	11.1
12	care	medium	12.6
13	care	medium	29.6
14	care	medium	13.0
15	care	medium	10.8
16	care	medium	13.4

3.4.3. Ordonner un jeu de données

Rappelez-vous lorsque nous avons utilisé la fonction `order()` pour ordonner un vecteur en fonction de l'ordre d'un autre vecteur (dans la Section 2.5.3). Cette fonction est très utile si vous souhaitez réorganiser les lignes de votre jeu de données. Par exemple, si nous voulons que toutes les lignes du jeu de données `licornes` soient classées par ordre croissant en fonction de la variable `height` et éditer toutes les colonnes par défaut :

```
taille_ord <- licornes[order(licornes$height), ]
head(taille_ord, n = 10)
```

	p_care	food	block	height	weight	mane_size	fluffyness	horn_rings
68	no_care	high	1	1.2	18.24	16.6	148.1	7
62	no_care	medium	2	1.8	10.47	11.8	120.8	9
86	no_care	low	1	1.8	6.01	17.6	46.2	4
72	no_care	high	1	2.1	19.15	15.6	176.7	6
63	no_care	medium	2	2.2	10.70	15.3	97.1	7
84	no_care	low	1	2.2	9.97	9.6	63.1	2
82	no_care	low	1	2.3	7.28	13.8	32.8	6
89	no_care	low	2	2.4	9.10	14.5	78.7	8
56	no_care	medium	1	2.5	14.85	17.5	77.8	10
69	no_care	high	1	2.6	16.57	17.1	141.1	3

Nous pouvons également ordonner par ordre décroissant d'une variable (i.e. `mane_size`) en utilisant l'argument `decreasing = TRUE` :

```
taille_criniere_ord <- licornes[order(licornes$mane_size, decreasing = TRUE), ]
head(taille_criniere_ord, n = 10)
```

	p_care	food	block	height	weight	mane_size	fluffyness	horn_rings
70	no_care	high	1	10.9	17.22	49.2	189.6	17
13	care	medium	2	7.1	8.16	29.6	9.7	2
24	care	high	1	6.5	17.13	24.1	147.4	6
65	no_care	high	1	8.5	22.53	20.8	166.9	16

	p_care	food	block	height	weight	mane_size	fluffyness	horn_rings
23	care	high	1	8.5	17.82	20.5	54.4	3
66	no_care	high	1	8.5	17.33	19.8	184.4	12
73	no_care	high	2	4.7	13.42	19.8	124.7	5
80	no_care	high	2	5.2	17.70	19.1	181.1	8
17	care	high	1	12.6	18.66	18.6	54.0	9
49	no_care	medium	1	5.6	11.03	18.6	49.9	8

Nous pouvons même ordonner des jeux de données sur la base de plusieurs variables. Par exemple, pour ordonner le jeu de données `licornes` dans l'ordre croissant des deux variables `block` et `height` :

```
bloc_taille_ord <- licornes[order(licornes$block, licornes$height), ]
head(bloc_taille_ord, n = 10)
```

	p_care	food	block	height	weight	mane_size	fluffyness	horn_rings
68	no_care	high	1	1.2	18.24	16.6	148.1	7
86	no_care	low	1	1.8	6.01	17.6	46.2	4
72	no_care	high	1	2.1	19.15	15.6	176.7	6
84	no_care	low	1	2.2	9.97	9.6	63.1	2
82	no_care	low	1	2.3	7.28	13.8	32.8	6
56	no_care	medium	1	2.5	14.85	17.5	77.8	10
69	no_care	high	1	2.6	16.57	17.1	141.1	3
87	no_care	low	1	3.0	9.93	12.0	56.6	6
53	no_care	medium	1	3.5	12.93	16.6	109.3	3
88	no_care	low	1	3.7	7.03	7.9	36.7	5

Et si nous voulions ordonner `licornes` par ordre croissant selon la variable `block` mais par ordre décroissant de `height`? Nous pouvons utiliser une astuce simple en ajoutant un `-` avant l'argument `licornes$height` lorsque nous utilisons la fonction `order()`. Cela transformera essentiellement toutes les valeurs de `height` en négatives, ce qui aura pour effet d'inverser l'ordre. Notez que cette astuce ne fonctionne qu'avec des variables numériques.

```
bloc_invtaille_ord <- licornes[order(licornes$block, -licornes$height), ]
rbind(head(bloc_invtaille_ord, n = 10), tail(bloc_invtaille_ord, n = 10))
```

	p_care	food	block	height	weight	mane_size	fluffyness	horn_rings
21	care	high	1	14.1	19.12	13.1	113.2	13
17	care	high	1	12.6	18.66	18.6	54.0	9
38	care	low	1	12.3	11.27	13.7	28.7	5
3	care	medium	1	11.2	12.76	7.1	66.7	10
70	no_care	high	1	10.9	17.22	49.2	189.6	17
2	care	medium	1	10.7	12.14	14.1	46.0	10
4	care	medium	1	10.4	8.78	11.9	20.3	1
5	care	medium	1	10.4	13.58	14.5	26.9	4
22	care	high	1	10.1	15.49	12.6	77.2	12
18	care	high	1	10.0	18.07	16.9	90.5	3
64	no_care	medium	2	3.9	12.97	17.0	97.5	5
93	no_care	low	2	3.9	9.19	12.4	52.6	9
91	no_care	low	2	3.7	8.10	10.5	60.5	6
94	no_care	low	2	3.3	8.92	11.6	55.2	6
92	no_care	low	2	3.2	7.45	14.1	38.1	4
42	care	low	2	3.1	8.74	16.1	39.1	3
76	no_care	high	2	2.6	18.88	16.4	181.5	14
89	no_care	low	2	2.4	9.10	14.5	78.7	8
63	no_care	medium	2	2.2	10.70	15.3	97.1	7
62	no_care	medium	2	1.8	10.47	11.8	120.8	9

Si nous voulons faire la même chose avec une variable de type facteur (ou caractère) comme `food` il faut utiliser la fonction `xtfrm()` sur cette variable, à l'intérieur de notre fonction `order()` :

```
invnourriture_taille_ord <- licornes[order(-xtfrm(licornes$food), licornes$height), ]
rbind(head(invnrirture_taille_ord, n = 10), tail(invnrirture_taille_ord, n = 10))
```

	p_care	food	block	height	weight	mane_size	fluffyness	horn_rings
62	no_care	medium	2	1.8	10.47	11.8	120.8	9
63	no_care	medium	2	2.2	10.70	15.3	97.1	7
56	no_care	medium	1	2.5	14.85	17.5	77.8	10
53	no_care	medium	1	3.5	12.93	16.6	109.3	3
58	no_care	medium	2	3.9	9.07	9.6	90.4	7
64	no_care	medium	2	3.9	12.97	17.0	97.5	5
52	no_care	medium	1	4.1	12.58	13.9	136.6	11
16	care	medium	2	4.5	12.55	13.4	14.4	6
60	no_care	medium	2	4.5	13.68	14.8	125.5	9
55	no_care	medium	1	4.9	6.89	8.2	52.9	3
29	care	high	2	9.2	13.26	11.3	108.0	9
78	no_care	high	2	9.3	18.75	18.4	181.1	16
18	care	high	1	10.0	18.07	16.9	90.5	3
19	care	high	1	10.0	13.29	15.8	142.7	12
22	care	high	1	10.1	15.49	12.6	77.2	12
70	no_care	high	1	10.9	17.22	49.2	189.6	17
25	care	high	2	11.5	23.89	14.3	101.5	12
17	care	high	1	12.6	18.66	18.6	54.0	9
21	care	high	1	14.1	19.12	13.1	113.2	13
32	care	high	2	17.2	19.20	10.9	89.9	14

Il est à noter que la variable `food` a été classée dans l'ordre alphabétique inverse et que la variable `height` a été ordonnée par valeurs croissantes, à l'intérieur de chaque niveau de `food`.

Si nous voulions ordonner le jeu de données selon les niveau de la variable `food`, c.-à-d. `low` → `medium` → `high` au lieu de l'ordre alphabétique par défaut (`high`, `low`, `medium`), nous devons d'abord modifier l'ordre des niveaux de `food` dans le jeu de données à l'aide de la fonction `factor()`. Une fois que nous avons fait cela, nous pouvons utiliser la fonction `order()` comme d'habitude. Remarque : si vous lisez la version pdf de ce livre, la sortie a été tronquée pour économiser de l'espace.

```
licornes$food <- factor(licornes$food,
  levels = c("low", "medium", "high"))
```

```
)
nourriture_ord <- licornes[order(licornes$food), ]
rbind(head(nourriture_ord, n = 10), tail(nourriture_ord, n = 10))
```

	p_care	food	block	height	weight	mane_size	fluffyness	horn_rings
33	care	low	1	8.0	6.88	9.3	16.1	4
34	care	low	1	8.0	10.23	11.9	88.1	4
35	care	low	1	6.4	5.97	8.7	7.3	2
36	care	low	1	7.6	13.05	7.2	47.2	8
37	care	low	1	9.7	6.49	8.1	18.0	3
38	care	low	1	12.3	11.27	13.7	28.7	5
39	care	low	1	9.1	8.96	9.7	23.8	3
40	care	low	1	8.9	11.48	11.1	39.4	7
41	care	low	2	7.4	10.89	13.3	9.5	5
42	care	low	2	3.1	8.74	16.1	39.1	3
71	no_care	high	1	7.2	15.21	15.9	135.0	14
72	no_care	high	1	2.1	19.15	15.6	176.7	6
73	no_care	high	2	4.7	13.42	19.8	124.7	5
74	no_care	high	2	5.0	16.82	17.3	182.5	15
75	no_care	high	2	6.5	14.00	10.1	126.5	7
76	no_care	high	2	2.6	18.88	16.4	181.5	14
77	no_care	high	2	6.0	13.68	16.2	133.7	2
78	no_care	high	2	9.3	18.75	18.4	181.1	16
79	no_care	high	2	4.6	14.65	16.7	91.7	11
80	no_care	high	2	5.2	17.70	19.1	181.1	8

3.4.4. Ajout de colonnes et de lignes

Il est parfois utile de pouvoir ajouter des lignes et des colonnes de données supplémentaires à nos jeux de données. Il existe plusieurs façons d'y parvenir (comme toujours en R !) en fonction des circonstances. Pour ajouter simplement des lignes supplémentaires à un jeu de données existant, nous pouvons utiliser la fonction `rbind()` (`row`/'ligne' - `bind`/'liaison') et pour ajouter des colonnes, la fonction `cbind()` (`columns`/'colonnes' - `bind`/'liaison'). Créons quelques jeux de données test pour voir cela en action en utilisant notre vieille amie, la fonction `data.frame()` :

```
# rbind pour les lignes ('rows')
df1 <- data.frame(
  id = 1:4, taille = c(120, 150, 132, 122),
  poids = c(44, 56, 49, 45)
)
df1
```

	id	taille	poids
	1	120	44
	2	150	56
	3	132	49
	4	122	45

```
df2 <- data.frame(
  id = 5:6, taille = c(119, 110),
  poids = c(39, 35)
)
df2
```

	id	taille	poids
	5	119	39
	6	110	35

```
df3 <- data.frame(
  id = 1:4, taille = c(120, 150, 132, 122),
  poids = c(44, 56, 49, 45)
)
df3
```

	id	taille	poids
	1	120	44
	2	150	56

id	taille	poids
3	132	49
4	122	45

```
df4 <- data.frame(localisation = c("UK", "CZ", "CZ", "UK"))
df4
```

localisation
UK
CZ
CZ
UK

Nous pouvons utiliser la fonction `rbind()` pour ajouter les lignes du jeu de données `df2` aux lignes de `df1` et créer le nouveau jeu de données à `df_lcomb` :

```
df_lcomb <- rbind(df1, df2)
df_lcomb
```

id	taille	poids
1	120	44
2	150	56
3	132	49
4	122	45
5	119	39
6	110	35

Et `cbind` pour ajouter la colonne de `df4` à la colonne `df3` et l'assigner à `df_ccomb` :

```
df_ccomb <- cbind(df3, df4)
df_ccomb
```

	id	taille	poids	localisation
	1	120	44	UK
	2	150	56	CZ
	3	132	49	CZ
	4	122	45	UK

Une autre situation dans laquelle l’ajout d’une nouvelle colonne à un jeu de données est utile est lorsque vous souhaitez effectuer une sorte de transformation sur une variable existante. Par exemple, supposons que nous voulions appliquer une transformation logarithmique, \log_{10} à une variable existante, la variable `taille` dans le jeu de données `df_rcomb` que nous avons créée ci-dessus. Nous pourrions simplement créer une variable distincte contenant ces valeurs, mais il est préférable de créer cette variable directement en tant que nouvelle colonne dans notre jeu de données existant afin de conserver toutes nos données ensemble. Appelons cette nouvelle variable `taille_log10`.

```
# transformation log10
df_lcomb$taille_log10 <- log10(df_lcomb$taille)
df_lcomb
```

	id	taille	poids	taille_log10
	1	120	44	2.079181
	2	150	56	2.176091
	3	132	49	2.120574
	4	122	45	2.086360
	5	119	39	2.075547
	6	110	35	2.041393

Cette situation survient également lorsque nous voulons changer le type d’une variable existante dans un jeu de données. Par exemple, la variable `id` dans la base de données `df_lcomb` est une donnée de type numérique (utilisez la fonction `str()` ou `class()` pour le vérifier). Si nous voulions convertir `id` en un facteur à utiliser plus tard dans notre analyse, nous pouvons créer une nouvelle variable appelée `id_f` dans notre jeu de données et utiliser la fonction `factor()` pour convertir la variable `id`.

```
# conversion en un facteur
df_lcomb$id_f <- factor(df_lcomb$id)
df_lcomb
```

	id	taille	poids	taille_log10	id_f
	1	120	44	2.079181	1
	2	150	56	2.176091	2
	3	132	49	2.120574	3
	4	122	45	2.086360	4
	5	119	39	2.075547	5
	6	110	35	2.041393	6

```
str(df_lcomb)
```

```
'data.frame': 6 obs. of 5 variables:
 $ id       : int 1 2 3 4 5 6
 $ taille    : num 120 150 132 122 119 110
 $ poids     : num 44 56 49 45 39 35
 $ taille_log10: num 2.08 2.18 2.12 2.09 2.08 ...
 $ id_f      : Factor w/ 6 levels "1","2","3","4",..: 1 2 3 4 5 6
```

3.4.5. Fusionner des jeux de données

Au lieu d'ajouter simplement des lignes ou des colonnes à un jeu de données, nous pouvons également fusionner deux jeux de données. Supposons que nous ayons un jeu de données contenant des informations taxonomiques sur certains invertébrés communs des côtes rocheuses du Royaume-Uni (appelé `taxa`) et un autre jeu de données qui contient des informations sur l'endroit où ils se trouvent habituellement sur le littoral rocheux (appelé `zone`). Nous pouvons fusionner ces jeux cadres de données pour produire un seul jeu de données contenant à la fois des informations taxonomiques et des informations sur l'emplacement. Commençons par créer ces deux jeux de données (en réalité, il vous suffira probablement d'importer vos différents ensembles de données).

```

taxa <- data.frame(
  GENUS = c("Patella", "Littorina", "Halichondria", "Semibalanus"),
  espece = c("vulgata", "littoria", "panacea", "balanoides"),
  famille = c("patellidae", "Littorinidae", "Halichondriidae", "Archaeobalanidae")
)
taxa

```

GENUS	espece	famille
Patella	vulgata	patellidae
Littorina	littoria	Littorinidae
Halichondria	panacea	Halichondriidae
Semibalanus	balanoides	Archaeobalanidae

```

zone <- data.frame(
  genus = c(
    "Laminaria", "Halichondria", "Xanthoria", "Littorina",
    "Semibalanus", "Fucus"
  ),
  espece = c(
    "digitata", "panacea", "parietina", "littoria",
    "balanoides", "serratus"
  ),
  zone = c("v_bas", "bas", "v_haut", "bas_moy", "haut", "bas_moy")
)
zone

```

genus	espece	zone
Laminaria	digitata	v_bas
Halichondria	panacea	bas
Xanthoria	parietina	v_haut
Littorina	littoria	bas_moy
Semibalanus	balanoides	haut

genus	espece	zone
Fucus	serratus	bas_moy

Puisque nos deux jeux de données contiennent au moins une variable en commun (`espece` dans notre cas), nous pouvons simplement utiliser la fonction `merge()` ('fusionner') pour créer un nouveau jeu de données appelé `taxa_zone`.

```
taxa_zone <- merge(x = taxa, y = zone)
taxa_zone
```

espece	GENUS	famille	genus	zone
balanoides	Semibalanus	Archaeobalanidae	Semibalanus	haut
littoria	Littorina	Littorinidae	Littorina	bas_moy
panacea	Halichondria	Halichondriidae	Halichondria	bas

Remarquez que le jeu de données fusionné ne contient que les lignes ayant des espèces **dans les 2** jeux de données. Il existe également deux colonnes appelées `GENUS` et `genus` car la fonction `merge()` les traite comme deux variables différentes provenant des deux jeux de données.

Si nous voulons inclure toutes les données des deux jeux de données, nous devrons utiliser l'argument `all = TRUE` dans la fonction `merge()`. Les valeurs manquantes seront incluses en tant que `NA` :

```
taxa_zone <- merge(x = taxa, y = zone, all = TRUE)
taxa_zone
```

espece	GENUS	famille	genus	zone
balanoides	Semibalanus	Archaeobalanidae	Semibalanus	haut
digitata	NA	NA	Laminaria	v_bas
littoria	Littorina	Littorinidae	Littorina	bas_moy
panacea	Halichondria	Halichondriidae	Halichondria	bas
parietina	NA	NA	Xanthoria	v_haut
serratus	NA	NA	Fucus	bas_moy

espece	GENUS	famille	genus	zone
vulgata	Patella	patellidae	NA	NA

Si les noms des variables sur lesquelles vous souhaitez baser la fusion sont différents dans chaque jeux de données (par exemple GENUS et genus), vous pouvez spécifier les noms dans le premier jeu de données (appelé x) et dans le second jeu de données (appelé y) à l'aide des arguments by.x = et by.y =.

```
taxa_zone <- merge(x = taxa, y = zone, by.x = "GENUS", by.y = "genus", all = TRUE)
taxa_zone
```

GENUS	espece.x	famille	espece.y	zone
Fucus	NA	NA	serratus	bas_moy
Halichondria	panacea	Halichondriidae	panacea	bas
Laminaria	NA	NA	digitata	v_bas
Littorina	littoria	Littorinidae	littoria	bas_moy
Patella	vulgata	patellidae	NA	NA
Semibalanus	balanoides	Archaeobalanidae	balanoides	haut
Xanthoria	NA	NA	parietina	v_haut

Ou utiliser plusieurs noms de variables :

```
taxa_zone <- merge(
  x = taxa, y = zone, by.x = c("espece", "GENUS"),
  by.y = c("espece", "genus"), all = TRUE
)
taxa_zone
```

espece	GENUS	famille	zone
balanoides	Semibalanus	Archaeobalanidae	haut
digitata	Laminaria	NA	v_bas
littoria	Littorina	Littorinidae	bas_moy
panacea	Halichondria	Halichondriidae	bas

espece	GENUS	famille	zone
parietina	Xanthoria	NA	v_haut
serratus	Fucus	NA	bas_moy
vulgata	Patella	patellidae	NA

3.4.6. Remodeler des jeux de données

Le remodelage des données dans différents formats est une tâche courante. Avec des données de type rectangulaire (les jeux de données ont le même nombre de lignes dans chaque colonne), vous rencontrerez deux formes principales de jeu de données : le format “long” (parfois appelé “empilé”) et le format “large”. Un exemple de jeu de données au format “long” est donné ci-dessous. Nous pouvons voir que chaque ligne représente une observation unique d'un sujet individuel et que chaque sujet peut avoir plusieurs lignes. Il en résulte une seule colonne de notre `mesures`.

```
donnees_longue <- data.frame(
  sujet = rep(c("A", "B", "C", "D"), each = 3),
  sexe = rep(c("M", "F", "F", "M"), each = 3),
  condition = rep(c("controle", "cond1", "cond2"), times = 4),
  mesures = c(
    12.9, 14.2, 8.7, 5.2, 12.6, 10.1, 8.9,
    12.1, 14.2, 10.5, 12.9, 11.9
  )
)
donnees_longue
```

sujet	sexe	condition	mesures
A	M	controle	12.9
A	M	cond1	14.2
A	M	cond2	8.7
B	F	controle	5.2
B	F	cond1	12.6
B	F	cond2	10.1
C	F	controle	8.9

sujet	sexe	condition	mesures
C	F	cond1	12.1
C	F	cond2	14.2
D	M	controle	10.5
D	M	cond1	12.9
D	M	cond2	11.9

Nous pouvons également formater les mêmes données au format “large”, comme indiqué ci-dessous. Dans ce format, nous avons plusieurs observations de chaque sujet dans une seule ligne avec des mesures dans différentes colonnes (`controle`, `cond1` et `cond2`). Il s’agit d’un format courant lorsqu’il s’agit de mesures répétées à partir d’unités d’échantillonnage.

```
donnees_large <- data.frame(
  sujet = c("A", "B", "C", "D"),
  sexe = c("M", "F", "F", "M"),
  controle = c(12.9, 5.2, 8.9, 10.5),
  cond1 = c(14.2, 12.6, 12.1, 12.9),
  cond2 = c(8.7, 10.1, 14.2, 11.9)
)
donnees_large
```

sujet	sexe	controle	cond1	cond2
A	M	12.9	14.2	8.7
B	F	5.2	12.6	10.1
C	F	8.9	12.1	14.2
D	M	10.5	12.9	11.9

Bien qu’il n’y ait pas de problème inhérent à l’un ou l’autre de ces formats, il est parfois nécessaire d’effectuer une conversion entre les deux, car certaines fonctions requièrent un format spécifique pour fonctionner. Le format le plus courant est le format “long”.

Il existe de nombreuses façons de convertir ces deux formats, mais nous utiliserons les fonctions `melt()` et `dcast()` du paquet `reshape2`  (vous devez d’abord l’installer). Les fonctions `melt()` est utilisée pour convertir les formats

larges en formats longs. Le premier argument de la fonction `melt()` est la base de données que nous voulons faire fondre (dans notre cas `wide_data`). La fonction `id.vars = c("subject", "sex")` est un vecteur des variables que vous souhaitez empiler, l'argument `measured.vars = c("control", "cond1", "cond2")` identifie les colonnes des mesures dans les différentes conditions, l'argument `variable.name = "condition"` spécifie ce que vous voulez appeler la colonne empilée de vos différentes conditions dans votre cadre de données de sortie et l'argument `value.name = "measurement"` est le nom de la colonne de vos mesures empilées dans votre cadre de données de sortie.

```
library(reshape2)
donnees_large # rappelons-nous à quoi ressemble le format "large"
```

	sujet	sexe	controle	cond1	cond2
A	M		12.9	14.2	8.7
B	F		5.2	12.6	10.1
C	F		8.9	12.1	14.2
D	M		10.5	12.9	11.9

```
# conversion du "large" en "long"
mon_df_long <- melt(
  data = donnees_large, id.vars = c("sujet", "sexe"),
  vars.measures = c("controle", "cond1", "cond2"),
  nom.variable = "condition", value.name = "mesures"
)
mon_df_long
```

	sujet	sexe	variable	mesures
A	M		controle	12.9
B	F		controle	5.2
C	F		controle	8.9
D	M		controle	10.5
A	M		cond1	14.2
B	F		cond1	12.6
C	F		cond1	12.1

	sujet	sexe	variable	mesures
	D	M	cond1	12.9
	A	M	cond2	8.7
	B	F	cond2	10.1
	C	F	cond2	14.2
	D	M	cond2	11.9

La fonction `dcast()` est utilisée pour convertir un jeu de données au format “long” en un jeu de données au format “large”. Le premier argument est à nouveau le jeu de données que nous voulons convertir (`donnees_longue` pour cet exemple). Le deuxième argument est la syntaxe de la formule. L’argument `sujet + sexe` de la formule signifie que nous voulons garder ces colonnes séparées, et l’élément `~ condition` est la colonne qui contient les étiquettes que nous voulons diviser en nouvelles colonnes dans notre nouveau jeu de données. La colonne `var.valeur = "mesures"` est la colonne qui contient les données mesurées.

```
donnees_longue # rappelons-nous à quoi ressemble le format "long"
```

	sujet	sexe	condition	mesures
	A	M	controle	12.9
	A	M	cond1	14.2
	A	M	cond2	8.7
	B	F	controle	5.2
	B	F	cond1	12.6
	B	F	cond2	10.1
	C	F	controle	8.9
	C	F	cond1	12.1
	C	F	cond2	14.2
	D	M	controle	10.5
	D	M	cond1	12.9
	D	M	cond2	11.9

```
# conversion du "long" en "large"
mon_df_large <- dcast(
```

```

data = donnees_longue, sujet + sexe ~ condition,
var.valeur = "mesures"
)
mon_df_large

```

sujet	sexe	cond1	cond2	controle
A	M	14.2	8.7	12.9
B	F	12.6	10.1	5.2
C	F	12.1	14.2	8.9
D	M	12.9	11.9	10.5

3.5. Introduction au tidyverse (*L'univers ordonné*)

il semble que ce ne soit pas très ordonné ici et que nous devons améliorer ça.

3.6. Résumer des jeux de données

Maintenant que nous sommes en mesure de manipuler et extraire des données de nos jeux de données, notre prochaine tâche est de commencer à explorer et connaître nos données. Dans cette section, nous commencerons à produire des tableaux de statistiques récapitulatives utiles sur les variables de notre jeu de données et, dans les deux chapitres suivants, nous aborderons la visualisation de nos données à l'aide de graphiques R de base et l'utilisation du paquet `ggplot2` 📈.

Un point de départ très utile consiste à produire des statistiques récapitulatives simples pour toutes les variables de notre jeu de données licornes à l'aide de la fonction `summary()` :

```
summary(licornes)
```

p_care	food	block	height	weight
Length:96	low :32	Min. :1.0	Min. : 1.200	Min. : 5.790
Class :character	medium:32	1st Qu.:1.0	1st Qu.: 4.475	1st Qu.: 9.027
Mode :character	high :32	Median :1.5	Median : 6.450	Median :11.395
		Mean :1.5	Mean : 6.840	Mean :12.155

```
            3rd Qu.:2.0    3rd Qu.: 9.025   3rd Qu.:14.537  
            Max.     :2.0      Max.     :17.200   Max.     :23.890  
  
mane_size      fluffyness      horn_rings  
Min.    : 5.80    Min.    : 5.80    Min.    : 1.000  
1st Qu.:11.07   1st Qu.: 39.05   1st Qu.: 4.000  
Median  :13.45   Median  : 70.05   Median  : 6.000  
Mean    :14.05   Mean    : 79.78   Mean    : 7.062  
3rd Qu.:16.45   3rd Qu.:113.28  3rd Qu.: 9.000  
Max.    :49.20   Max.    :189.60   Max.    :17.000
```

Pour les variables numériques (c'est-à-dire `height`, `weight` etc.), la moyenne, le minimum, le maximum, la médiane, le premier quartile (inférieur) et le troisième quartile (supérieur) sont présentés. Pour les variables factorielles (i.e. `care` et `food`), le nombre d'observations dans chacun des niveaux est indiqué. Si une variable contient des données manquantes, le nombre de NA est également indiqué.

Si nous voulons résumer un sous-ensemble plus petit de variables dans notre jeu de données, nous pouvons utiliser nos compétences en matière d'indexation en combinaison avec la fonction `summary()`. Par exemple, pour résumer uniquement les variables `height`, `weight`, `mane_size` et `fluffyness` nous pouvons inclure les index de colonne appropriés avec les crochets `[]`. Remarquez que nous incluons toutes les lignes en ne spécifiant pas d'index de ligne :

```
summary(licornes[, 4:7])
```

```
height        weight       mane_size      fluffyness  
Min.    : 1.200    Min.    : 5.790    Min.    : 5.80    Min.    : 5.80  
1st Qu.: 4.475   1st Qu.: 9.027   1st Qu.:11.07   1st Qu.: 39.05  
Median  : 6.450   Median  :11.395   Median  :13.45   Median  : 70.05  
Mean    : 6.840   Mean    :12.155   Mean    :14.05   Mean    : 79.78  
3rd Qu.: 9.025   3rd Qu.:14.537   3rd Qu.:16.45   3rd Qu.:113.28  
Max.    :17.200   Max.    :23.890   Max.    :49.20   Max.    :189.60
```

```
# ou de manière équivalente
```

```
# summary(licornes[, c("height", "weight", "mane_size", "fluffyness")])
```

Et pour résumer une seule variable :

```
summary(licornes$mane_size)

Min. 1st Qu. Median Mean 3rd Qu. Max.
5.80 11.07 13.45 14.05 16.45 49.20
```

```
# ou de manière équivalente
# summary(licornes[, 6])
```

Comme vous l'avez vu plus haut, le `summary()` indique le nombre d'observations dans chaque niveau de nos variables factorielles. Une autre fonction utile pour générer des tableaux de comptage est la fonction `table()`. La fonction `table()` peut être utilisée pour construire des tables de contingence de différentes combinaisons de niveaux de facteurs. Par exemple, pour compter le nombre d'observations pour chaque niveau de `food` :

```
table(licornes$food)
```

	low	medium	high
32	32	32	

Nous pouvons aller plus loin en produisant un tableau des effectifs pour chaque combinaison des niveau de `food` et `care` :

```
table(licornes$food, licornes$p_care)
```

	care	no_care
low	16	16
medium	16	16
high	16	16

Une version plus souple de la fonction `table()` est la fonction `xtabs()`. La fonction `xtabs()` utilise une notation de formule (~) pour construire des tables de contingence avec les variables de classification croisée séparées par un + à droite de la formule. `xtabs()` contient également un argument utile, `data =`, pour ne pas avoir à inclure le nom du jeu de données lors de la spécification de chaque variable :

```
xtabs(~ food + p_care, data = licornes)
```

```
p_care  
food      care no_care  
low       16    16  
medium    16    16  
high      16    16
```

Nous pouvons même construire des tables de contingence plus compliquées en utilisant davantage de variables. Notez que dans l'exemple ci-dessous, la variable `xtabs()` a discrètement contraint notre `block` à être un facteur.

```
xtabs(~ food + p_care + block, data = licornes)
```

```
, , block = 1  
  
p_care  
food      care no_care  
low       8     8  
medium    8     8  
high      8     8
```

```
, , block = 2  
  
p_care
```

```
food      care no_care  
low       8     8  
medium    8     8  
high      8     8
```

Et pour un tableau mieux formaté, nous pouvons imbriquer la fonction `xtabs()` à l'intérieur de la fonction `ftable()` pour “aplatisir” le tableau.

```
ftable(xtabs(~ food + p_care + block, data = licornes))
```

		block 1 2	
food	p_care		
low	care	8	8
	no_care	8	8
medium	care	8	8
	no_care	8	8
high	care	8	8
	no_care	8	8

Nous pouvons également résumer nos données pour chaque niveau d'une variable factorielle. Supposons que nous voulions calculer la valeur moyenne de `height` pour chacun de nos niveaux `low`, `meedium` et `high` de la variable `food`. Pour ce faire, nous utilisons la fonction `mean()` que nous appliquons à la variable `height` pour chaque niveau de `food` en utilisant la fonction `tapply()`.

```
tapply(licornes$height, licornes$food, mean)
```

low	medium	high
5.853125	7.012500	7.653125

La fonction `tapply()` n'est pas limitée au calcul des valeurs moyennes, vous pouvez l'utiliser pour appliquer de nombreuses fonctions fournies avec R ou même des fonctions que vous avez écrites vous-même (voir le Chapitre 5 pour plus de détails). Par exemple, nous pouvons appliquer la fonction `sd()` pour calculer l'écart-type pour chaque niveau de `food` ou même la fonction `summary()`.

```
tapply(licornes$height, licornes$food, sd)
```

low	medium	high
2.828425	3.005345	3.483323

```
tapply(licornes$height, licornes$food, summary)
```

```
$low
  Min. 1st Qu. Median   Mean 3rd Qu.   Max.
1.800  3.600  5.550  5.853  8.000 12.300
```

```
$medium
  Min. 1st Qu. Median   Mean 3rd Qu.   Max.
1.800  4.500  7.000  7.013  9.950 12.300
```

```
$high
  Min. 1st Qu. Median   Mean 3rd Qu.   Max.
1.200  5.800  7.450  7.653  9.475 17.200
```

Remarque : si la variable que vous souhaitez résumer contient des valeurs manquantes (NA), vous devrez également inclure un argument spécifiant comment vous souhaitez que la fonction traite les valeurs manquantes (NA). Nous en avons vu un exemple dans la Section 2.5.5 où la fonction `mean()` a renvoyé une valeur NA en cas de données manquantes. Pour inclure les NA, il suffit d'ajouter l'argument `na.rm = TRUE` lors de l'utilisation de `tapply()`.

```
tapply(licornes$height, licornes$food, mean, na.rm = TRUE)
```

```
low    medium      high
5.853125 7.012500 7.653125
```

Nous pouvons également utiliser `tapply()` pour appliquer des fonctions à plus d'un seul facteur. La seule chose à retenir est que les facteurs doivent être fournis à la fonction `tapply()` sous la forme d'une liste à l'aide de la fonction `list()`. Pour calculer la moyenne de `height` pour chaque combinaison de `food` et `care` nous pouvons utiliser la notation `list(licornes$food, licornes$p_care)`.

```
tapply(licornes$height, list(licornes$food, licornes$p_care), mean)
```

```
care no_care
low     8.0375 3.66875
medium  9.1875 4.83750
high    9.6000 5.70625
```

Et si vous en avez un peu marre d'avoir à écrire `licornes$` pour chaque variable, vous pouvez imbriquer la notation `tapply()` à l'intérieur de la fonction `with()`. La fonction `with()` permet à R d'évaluer une expression par rapport à un objet de données nommé (dans ce cas, le jeu de données `licornes`).

```
with(licornes, tapply(height, list(food, p_care), mean))
```

	care	no_care
low	8.0375	3.66875
medium	9.1875	4.83750
high	9.6000	5.70625

La fonction `with()` fonctionne également avec de nombreuses autres fonctions et peut vous faire économiser beaucoup de temps de frappe !

Une autre fonction très utile pour résumer des données est la fonction `aggregate()`. La fonction `aggregate()` fonctionne de manière très similaire à la fonction `tapply()` mais est un peu plus flexible.

Par exemple, pour calculer la moyenne des variables `height`, `weight`, `mane_size` et `fluffyness` pour chaque niveau de `food` :

```
aggregate(licornes[, 4:7], by = list(food = licornes$food), FUN = mean)
```

food	height	weight	mane_size	fluffyness
low	5.853125	8.652812	11.14375	45.1000
medium	7.012500	11.164062	13.83125	67.5625
high	7.653125	16.646875	17.18125	126.6875

Dans le code ci-dessus, nous avons indexé les colonnes que nous voulons résumer dans le jeu de données `licornes` à l'aide de la notation `licornes[, 4:7]`. Les `by =` ("par =") spécifie une liste de facteurs (`list(food = licornes$food)`) et l'argument `FUN =` désigne la fonction à appliquer (`mean` dans cet exemple).

Comme dans le cas de la fonction `tapply()` nous pouvons inclure plus d'un facteur sur lequel appliquer une fonction. Ici, nous calculons les valeurs moyennes pour chaque combinaison de `food` et `care` :

```
aggregate(licornes[, 4:7], by = list(
  food = licornes$food,
  p_care = licornes$p_care
), FUN = mean)
```

food	p_care	height	weight	mane_size	fluffyness
low	care	8.03750	9.016250	9.96250	30.30625
medium	care	9.18750	11.011250	13.48750	40.59375
high	care	9.60000	16.689375	15.54375	98.05625
low	no_care	3.66875	8.289375	12.32500	59.89375
medium	no_care	4.83750	11.316875	14.17500	94.53125
high	no_care	5.70625	16.604375	18.81875	155.31875

Nous pouvons également utiliser la fonction `aggregate()` d'une manière différente en utilisant la méthode de la formule (comme nous l'avons fait avec la fonction `xtabs()`). Dans la partie gauche de la formule (~), nous spécifions la variable à laquelle nous voulons appliquer la fonction `mean()` et, dans la partie droite, nos facteurs séparés par un symbole +. La méthode de la formule permet également d'utiliser l'argument `data =` pour des raisons de commodité.

```
aggregate(height ~ food + p_care, FUN = mean, data = licornes)
```

food	p_care	height
low	care	8.03750
medium	care	9.18750
high	care	9.60000
low	no_care	3.66875
medium	no_care	4.83750
high	no_care	5.70625

L'un des avantages de la méthode de la formule est qu'elle permet également d'utiliser l'argument `subset =` pour appliquer la fonction à des sous-ensembles des données originales. Par exemple, pour calculer la moyenne de `height` pour chaque combinaison de `food` et `care` mais seulement pour les licornes qui ont moins de 7 anneaux sur leur corne (`horn_rings`).

```
aggregate(height ~ food + p_care, FUN = mean, subset = horn_rings < 7, data = licornes)
```

food	p_care	height
low	care	8.176923
medium	care	8.570000
high	care	7.900000
low	no_care	3.533333
medium	no_care	5.316667
high	no_care	3.850000

Ou seulement pour les licornes du block 1.

```
aggregate(height ~ food + p_care, FUN = mean, subset = block == "1", data = licornes)
```

food	p_care	height
low	care	8.7500
medium	care	9.5375
high	care	10.0375
low	no_care	3.3250
medium	no_care	5.2375
high	no_care	5.9250

3.7. Exporter des données

Nous espérons que vous avez maintenant une idée de la puissance et de l'utilité de R pour manipuler et résumer des données (et nous n'avons fait qu'effleurer la surface). L'un des grands avantages de l'utilisation de R pour manipuler vos données est que vous disposez d'un enregistrement permanent de tout ce que vous avez fait avec vos données. Fini le temps des modifications non documentées dans Excel ou Calc ! En traitant vos données en “lecture seule” et en documentant toutes vos décisions dans R, vous aurez fait un grand pas en avant pour rendre votre analyse plus reproductible et plus transparente pour les autres. Il est toutefois important de savoir que les modifications apportées à votre base de données dans R ne changeront pas le fichier de données original que vous avez importé dans R (et c'est une bonne chose). Heureusement, il est facile d'exporter des cadres de données vers des fichiers externes dans une grande variété de formats.

3.7.1. Fonctions d'exportation

La principale fonction d'exportation de jeux de données est `write.table()`. Comme pour la fonction `read.table()` la fonction `write.table()` est très flexible et dispose de nombreux arguments permettant de personnaliser son comportement. Prenons l'exemple de notre jeu de données d'origine licornes et exportons ces modifications vers un fichier externe.

De la même manière que pour `read.table()`, `write.table()` possède une série de fonctions avec des valeurs par défaut spécifiques au format, telles que `write.csv()` et `write.delim()` qui utilisent respectivement “,” et les tabulations comme délimiteurs, et incluent les noms de colonnes par défaut.

Ordonnons les lignes du jeu de données par ordre croissant de la variable `height` à l'intérieur de chaque niveau de `food`. Nous appliquerons également une transformation racine carrée à la variable du nombre d'anneaux de la corne (`horn_rings`) et une transformation logarithmique, `log10` sur la variable `height` et enregistrons ça en tant que colonnes supplémentaires dans notre jeu de données (nous espérons que cela vous est familier maintenant !).

```
licornes_df2 <- licornes[order(licornes$food, licornes$height), ]  
licornes_df2$horn_rings_sqrt <- sqrt(licornes_df2$horn_rings)  
licornes_df2$log10_height <- log10(licornes_df2$height)  
str(licornes_df2)  
  
'data.frame': 96 obs. of 10 variables:  
 $ p_care       : chr "no_care" "no_care" "no_care" "no_care" ...  
 $ food         : Factor w/ 3 levels "low","medium",...: 1 1 1 1 1 1 1 1 1 1 ...  
 $ block        : int 1 1 1 2 1 2 2 2 1 2 ...  
 $ height       : num 1.8 2.2 2.3 2.4 3 3.1 3.2 3.3 3.7 3.7 ...  
 $ weight       : num 6.01 9.97 7.28 9.1 9.93 8.74 7.45 8.92 7.03 8.1 ...  
 $ mane_size    : num 17.6 9.6 13.8 14.5 12 16.1 14.1 11.6 7.9 10.5 ...  
 $ fluffyness   : num 46.2 63.1 32.8 78.7 56.6 39.1 38.1 55.2 36.7 60.5 ...  
 $ horn_rings   : int 4 2 6 8 6 3 4 6 5 6 ...  
 $ horn_rings_sqrt: num 2 1.41 2.45 2.83 2.45 ...  
 $ log10_height : num 0.255 0.342 0.362 0.38 0.477 ...
```

Nous pouvons maintenant exporter notre nouveau jeu de données `licornes_df2` à l'aide de la fonction `write.table()`. Le premier argument est le jeu de données que vous voulez exporter (`licornes_df2` dans notre exemple). Nous donnons ensuite le nom du fichier que nous voulons créer (avec son extension) et le chemin d'accès

au fichier entre guillemets simples ou doubles en utilisant l’argument `file =`. Dans cet exemple, nous exportons le jeu de données vers un fichier appelé `licornes_transformee.csv` dans le répertoire `donnees`. L’argument `row.names = FALSE` empêche R d’inclure les noms des lignes dans la première colonne du fichier.

```
write.csv(licornes_transformee,
  file = "donnees/licornes_transformee.csv",
  row.names = FALSE
)
```

Comme nous avons enregistré le fichier en tant que fichier texte délimité par des virgules (CSV), nous pouvons ouvrir ce fichier dans n’importe quel éditeur de texte.

Nous pouvons bien sûr exporter nos fichiers dans une variété d’autres formats.

3.7.2. Autres fonctions d’exportation

Comme pour l’importation de fichiers de données dans R, il existe également de nombreuses fonctions alternatives pour exporter des données vers des fichiers externes, en plus de la fonction `write.table()`. Si vous avez suivi la rubrique “Autres fonctions d’importation” Section 3.3.4 de ce chapitre, vous aurez déjà installé les paquets nécessaires.

La fonction `fwrite()` du paquet `data.table`  est très efficace pour exporter des objets de données volumineux et est beaucoup plus rapide que la fonction `write.table()`. Il est également assez simple à utiliser car il possède la plupart des arguments de la fonction `write.table()`. Pour exporter un fichier texte délimité par des tabulations, il suffit de spécifier le nom du jeu de données, le nom et le chemin du fichier de sortie et le séparateur entre les colonnes :

```
library(data.table)
fwrite(licornes_df2, file = "donnees/licornes_04_12.txt", sep = "\t")
```

Pour exporter un fichier csv délimité, c’est encore plus facile car nous n’avons même pas besoin d’inclure l’option `sep = :`

```
library(data.table)
fwrite(licornes_df2, file = "donnees/licornes_04_12.csv")
```

Le paquet `readr` 📦 est également fourni avec deux fonctions utiles pour exporter rapidement des données dans des fichiers externes : la fonction `write_tsv()` pour écrire des fichiers délimités par des tabulations et la fonction `write_csv()` pour enregistrer des fichiers de valeurs séparées par des virgules (csv).

```
library(readr)  
  
write_tsv(licornes_df2, path = "donnees/licornes_04_12.txt")  
  
write_csv(licornes_df2, path = "donnees/licornes_04_12.csv")
```

Chapitre 4

Figures

La synthèse de vos données, sous forme numérique ou graphique, est un élément important (bien que souvent négligé) de toute analyse de données. Par chance, R dispose d'excellentes capacités graphiques et peut être utilisé pour produire des graphiques pour l'exploration initiale des données, la validation de modèles ou des figures très complexes destinées à la publication.

Il existe trois systèmes principaux pour produire des graphiques dans R :

- Les graphiques de base de R
- `lattice` 
- `ggplot2` 

Le système graphique R de base est le système de traçage original qui existe (et qui a évolué) depuis les premiers jours de R. Lors de la création de graphiques avec base R, nous avons tendance à utiliser des fonctions de haut niveau (comme la fonction `plot()`) pour créer notre graphique, puis utiliser une ou plusieurs fonctions de bas niveau (comme les fonctions `lines()`, `text()`, etc.) pour ajouter des informations supplémentaires à ces graphiques.

Cela peut sembler un peu bizarre (et prendre du temps) lorsque vous commencez à créer des graphiques fantaisistes dans R, mais cela vous permet de personnaliser presque tous les aspects de votre graphique et de le complexifier avec des couches successives. Le revers de cette flexibilité est que vous devrez souvent prendre de nombreuses décisions sur la manière dont vous souhaitez que votre graphique se présente plutôt que de compter sur le logiciel pour prendre ces décisions à votre place. Cela dit, il est généralement très rapide et facile de générer des graphiques exploratoires simples à l'aide de graphiques R de base.

Le système `lattice` est implémenté dans le paquet `lattice`  qui est pré installé avec l'installation standard de R. Cependant, il n'est pas chargé par défaut, vous devez donc d'abord utiliser l'appeler `library(lattice)` pour

accéder à toutes les fonctions de graphiques. Contrairement aux graphiques de base de R, les graphiques lattices sont généralement générés en une seule fois à l'aide d'une seule fonction. Il n'est donc pas nécessaire d'utiliser des fonctions de traçage de haut et bas niveau pour personnaliser l'aspect d'un graphique.

Les graphiques lattices prennent également plus de décisions pour vous en ce qui concerne l'aspect des graphiques, mais cela a un coût, car la personnalisation des graphiques lattices pour qu'ils aient exactement l'aspect que vous souhaitez peut s'avérer plus complexe. Les graphiques lattices se prêtent particulièrement bien au traçage de données multidimensionnelles complexes à l'aide de graphiques en panel (également appelés graphiques en treillis). Nous verrons quelques exemples de ces types de graphiques plus loin dans ce chapitre.

`ggplot2`  est basé sur le livre *Grammar of Graphics* (la grammaire de graphiques) de Wilkinson (2005). (Vous trouverez un résumé intéressant du livre [ici](#)).

La *Grammaire des graphiques* décompose les figures en leurs différents éléments (par ex. les statistiques sous-jacentes, l'arrangement géométrique, le thème, voir Figure 4.1).

Les utilisateurs sont ainsi en mesure de manipuler chacun de ces composants (i.e. couches) et de produire une figure sur mesure répondant à leurs besoins spécifiques.

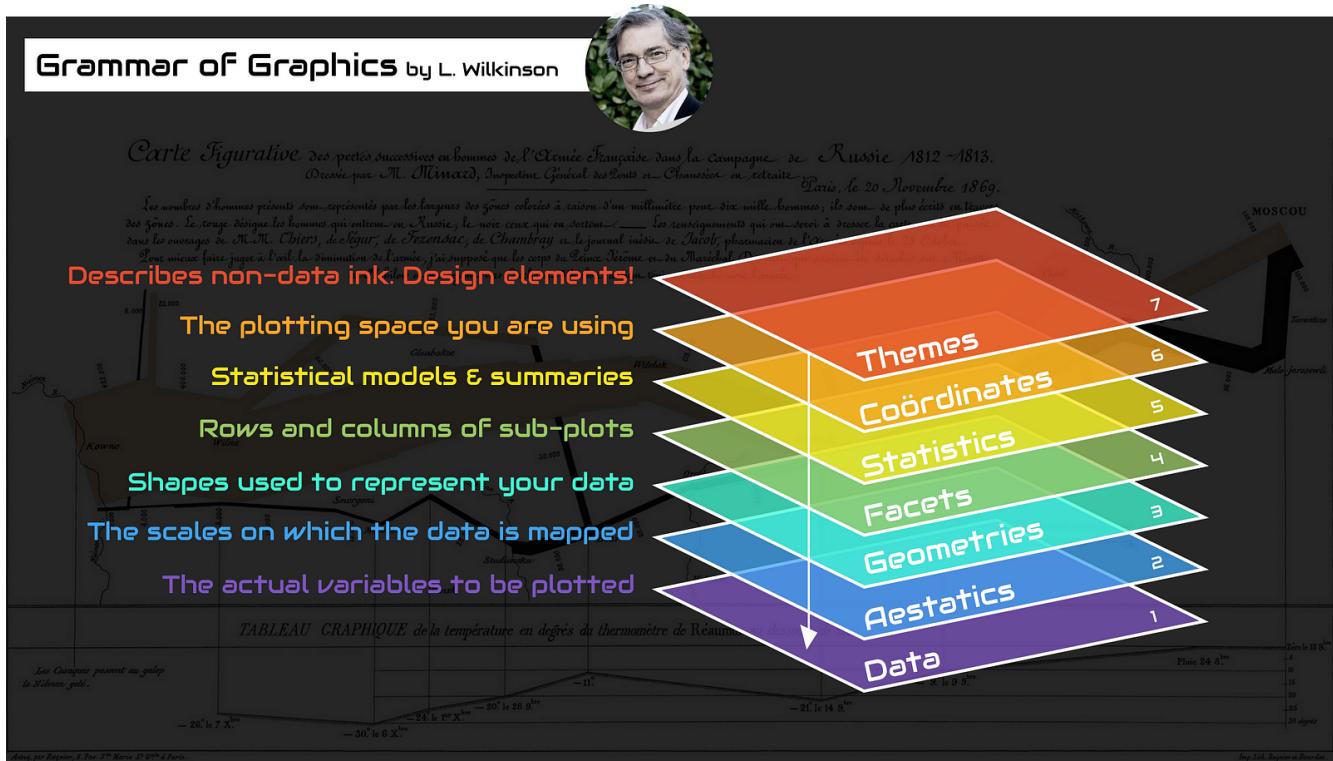


Figure 4.1.: La grammaire des graphique. Visuel de Thomas de Beus

Chacun de ces systèmes a ses forces et ses faiblesses et nous les utilisons souvent de manière interchangeable. Dans ce chapitre, nous vous présenterons les graphiques de base de R et les fonction de base de `ggplot2`  . Il est important

de noter que `ggplot2` 📦 n'est pas **obligatoire** pour faire des figures "fantaisistes" et informatives dans R. Si vous préférez utiliser les graphiques de base de R, n'hésitez pas à continuer, car presque tous les graphiques de `ggplot2` 📦 peuvent être créées avec les graphiques de base de R (nous utilisons souvent l'une ou l'autre approche en fonction de ce que nous faisons). La différence entre `ggplot2` 📦 et la base R est la façon dont vous *obtenez* le produit final plutôt que des différences substantielles dans le produit final lui-même. Il s'agit néanmoins d'une croyance répandue, probablement due au fait qu'il est (à notre avis du moins) plus facile d'obtenir une silhouette modérément attrayante avec `ggplot2` 📦 car de nombreuses décisions esthétiques sont prises pour l'utilisateur, sans qu'il sache nécessairement qu'une décision a été prise !

Dans cet esprit, commençons à créer quelques figures.

4.1. Graphiques de base simples en R

Il existe de nombreuses fonctions dans R pour produire des graphiques, des plus simples aux plus complexes. Il est impossible de couvrir tous les aspects de la production de graphiques en R dans ce livre. Nous vous présenterons donc la plupart des méthodes courantes de représentation graphique des données et nous décrirons comment personnaliser vos graphiques plus tard dans la Section 4.5.

La fonction de haut niveau la plus couramment utilisée pour produire des graphiques en R est (sans surprise) la fonction `plot()` fonction. Par exemple, représentons le poids (`weight`) des licornes de notre jeu de données `licornes` ("unicorns.csv") que nous avons importé dans la Section 3.3.2.

```
licornes <- read.csv(file = "data/unicorns.csv")

plot(licornes$weight)
```

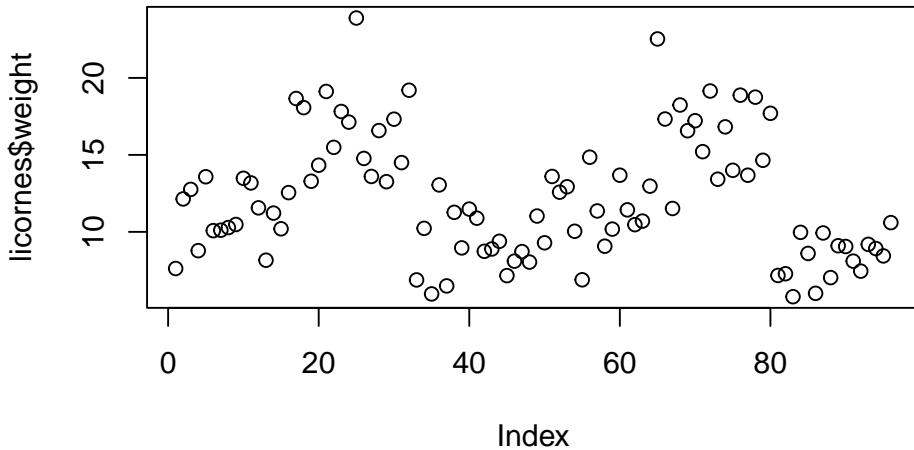


Figure 4.2.: Dispersion des données de poids des licornes.

R a tracé les valeurs de `weight` (sur l'axe des y) en fonction d'un indice puisque nous ne traçons qu'une seule variable. L'indice est simplement l'ordre des valeurs de `weight` dans le jeu de données (allant de la valeur 1 à 97). Le nom de variable `weight` a été automatiquement inclus comme nom de l'axe des y et les échelles des axes ont été automatiquement définies.

Si nous n'avions inclus que la variable `weight` plutôt que `licornes$weight`, la fonction `plot()` affichera une erreur car la variable `weight` n'existe que dans le jeu de données `licornes`.

```
plot(weight)
## Error in plot(weight) : object 'weight' not found (Erreur dans plot(weight) : objet 'weight' n'
```

Comme de nombreuses fonctions de graphique de base de R n'ont pas d'argument `data =` pour spécifier directement le nom du jeu de données, nous pouvons utiliser la fonction `with()` en combinaison avec la fonction `plot()` comme raccourci.

```
with(licornes, plot(weight))
```

Pour tracer un nuage de points d'une variable numérique par rapport à une autre variable numérique, il suffit d'inclure les deux variables en tant qu'arguments lors de l'utilisation de la fonction `plot()`. Par exemple, pour tracer `fluffyness` sur l'axe des y et `weight` sur l'axe des x.

```
plot(x = licornes$weight, y = licornes$fluffyness)
```

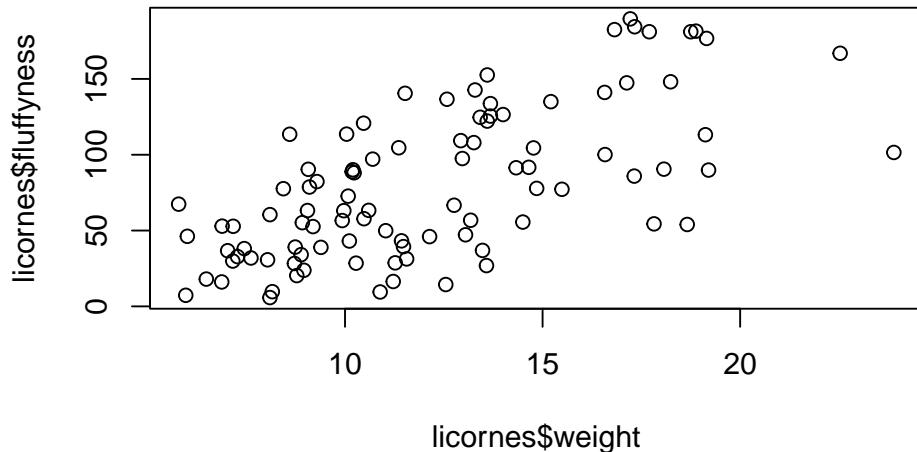


Figure 4.3.: Dispersion des données de fluffyness en fonction du poids des licornes.

Il existe une approche équivalente pour ces types de graphiques, ce qui est souvent source de confusion au début. Vous pouvez également utiliser la notation de formule lors de l'utilisation de la fonction `plot()`. Cependant, contrairement à la méthode précédente, la méthode de la formule exige que vous spécifiez d'abord la variable de l'axe des y, puis un `~` puis la variable de l'axe des x.

```
plot(fluffyness ~ weight, data = licornes)
```

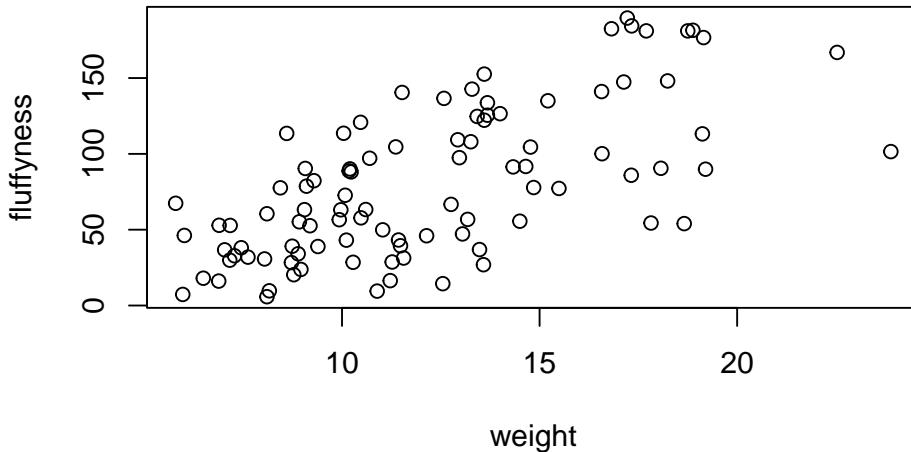


Figure 4.4.: Dispersion des données de fluffyness en fonction du poids des licornes (avec la notation de formule).

Ces deux approches étant équivalentes, nous vous suggérons de choisir celle que vous préférez et de l'appliquer.

Vous pouvez également spécifier le type de graphique que vous souhaitez tracer en utilisant l'argument `type =`. Vous pouvez tracer seulement les points (`type = "p"` c'est l'option par défaut), seulement les lignes (`type = "l"`), les points et les lignes connectés (`type = "b"`), les points et les lignes avec les lignes passant par les points (`type = "o"`) et les points vides reliés par des lignes (`type = "c"`). Par exemple, utilisons nos connaissances acquises grâce à la Section 2.4 pour générer deux vecteurs de nombres (`mon_x` et `mon_y`), puis traçons l'un par rapport à l'autre en utilisant différents `type =` pour voir quels types de graphiques on obtient. Ne vous préoccupez pas de la ligne de code `par(mfrow = c(2, 2))` pour l'instant. Nous l'utilisons simplement pour diviser fenêtre des graphiques afin de pouvoir placer les quatre graphiques sur la même fenêtre pour gagner de la place. Voir Section 4.4 dans le chapitre pour plus de détails à ce sujet. Le graphique en haut à gauche est de `type = "l"`, le graphe en haut à droite de `type = "b"` en bas à gauche de `type = "o"` et en bas à droite est de `type = "c"`.

```
mon_x <- 1:10
mon_y <- seq(from = 1, to = 20, by = 2)

par(mfrow = c(2, 2))
plot(mon_x, mon_y, type = "l")
plot(mon_x, mon_y, type = "b")
plot(mon_x, mon_y, type = "o")
plot(mon_x, mon_y, type = "c")
```

```
plot(mon_x, mon_y, type = "c")
```

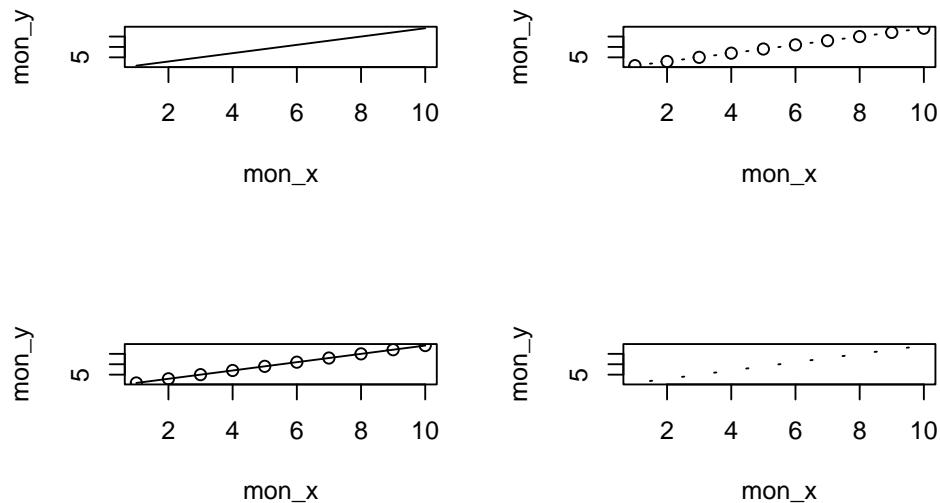


Figure 4.5.: Présentation des 4 types principaux de représentation avec la fonction `plot()`.

Il est vrai que les graphiques que nous avons produites jusqu'à présent n'ont rien d'extraordinaire. Cependant, la fonction `plot()` est incroyablement polyvalente et peut générer un large éventail de graphiques que vous pouvez personnaliser à votre guise. Nous verrons comment personnaliser les ggplots dans la Section 4.5. Pour l'anecdote, la fonction `plot()` est également ce que l'on appelle une fonction générique, ce qui signifie qu'elle peut modifier son comportement par défaut en fonction du type d'objet utilisé comme argument. Vous en verrez un exemple dans la Section 9.6 où nous utilisons la fonction `plot()` pour générer des graphiques de diagnostic des résidus d'un objet de type modèle linéaire (je parie que vous êtes impatient !).

4.2. *ggplot2*

Comme nous l'avons déjà mentionné, la grammaire de `ggplot2`  nécessite plusieurs éléments pour produire un graphique (Figure 4.1) et au moins 3 éléments sont nécessaire :

- un jeu de données (“`data =`”)
- un système de cartographie définissant x et y (“`aes()`”)
- une couche géométrique (“`geom_...()`”)

Les données et la cartographie sont fournies dans le cadre de l'appel de la fonction `ggplot()` à l'aide des arguments `data` et `mapping`. La couche géométrique est ajoutée à l'aide de fonctions spécifiques.

En fait, toutes les couches sont nécessaires, mais les valeurs simples par défaut des autres couches sont automatiquement fournies.

Pour refaire la Figure 4.4, qui ne contient qu'un nuage de points, nous pouvons utiliser la fonction `geom_point()`.

```
ggplot(  
  data = licornes,  
  mapping = aes(x = weight, y = fluffyness)  
) +  
  geom_point()
```

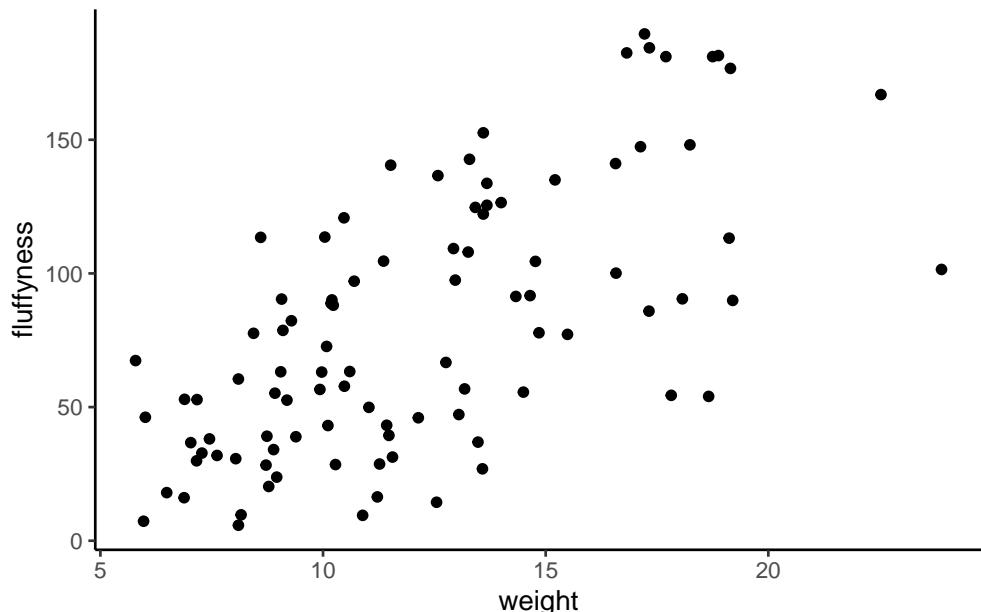


Figure 4.6.: Dispersion des données de fluffyness en fonction du poids des licornes (avec ggplot2 📦).

Maintenant que nous avons une compréhension de base de `ggplot()` nous pouvons explorer quelques graphiques en utilisant à la fois le code de base de R et le code ggplot2 📦

4.3. Graphiques simples

4.3.1. Diagrammes de dispersion (nuages de points, Scatter plots)

Type de graphique simple très utile pour étudier la relation entre deux variables, par exemple. Voici le code pour le faire en utilisant les fonctions R de base (Figure 4.4)

```
plot(fluffyness ~ weight, data = licornes)
```

ou ggplot2  (Figure 4.6)

```
ggplot(
  data = licornes,
  mapping = aes(x = weight, y = fluffyness)
) +
  geom_point()
```

Un grand avantage de `ggplot()` pour les nuages de points simples est la facilité avec laquelle nous pouvons ajouter une régression, une ligne plus lisse (loess ou gam) au graphique en utilisant `geom_smooth()` pour ajouter une couche statistique au graphique.

```
ggplot(
  data = licornes,
  mapping = aes(x = weight, y = fluffyness)
) +
  geom_point() +
  geom_smooth()
```

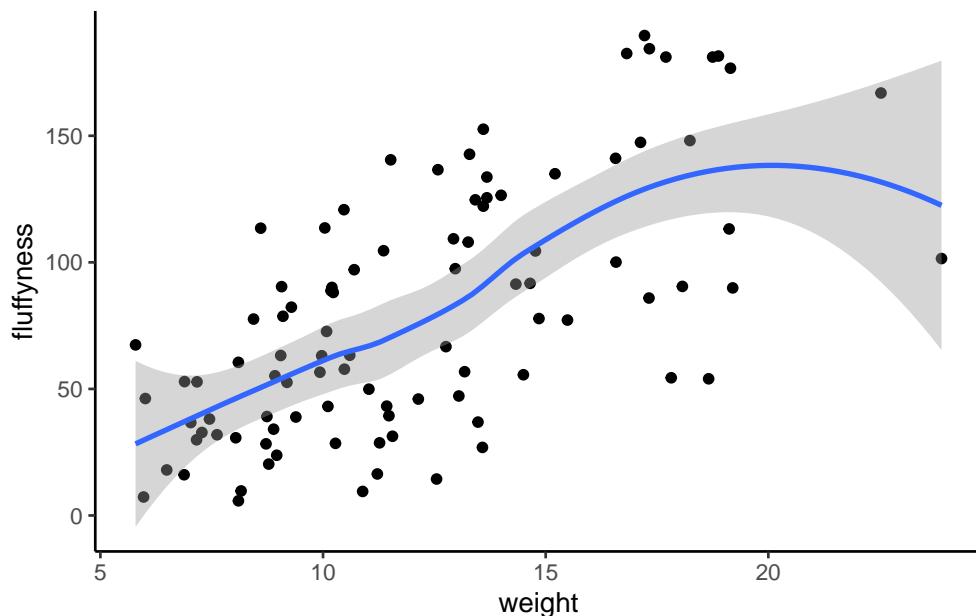


Figure 4.7.: Nuage de point avec une régression LOESS.

4.3.2. Histogrammes

Les histogrammes de fréquence sont utiles pour se faire une idée de la distribution des valeurs d'une variable numérique. En utilisant la base de R, la fonction `hist()` prend un tableau numérique comme argument principal. Dans `ggplot2` 📦, il faut utiliser `geom_histogram()`. Générons un histogramme des valeurs de taille (`height` dans le jeu de données).

Avec la base de R :

```
hist(licornes$height)
```



Figure 4.8.: Histogramme des valeurs de taille avec la base de R.

avec ggplot2 :

```
ggplot(licornes, aes(x = height)) +
  geom_histogram()
```

`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.

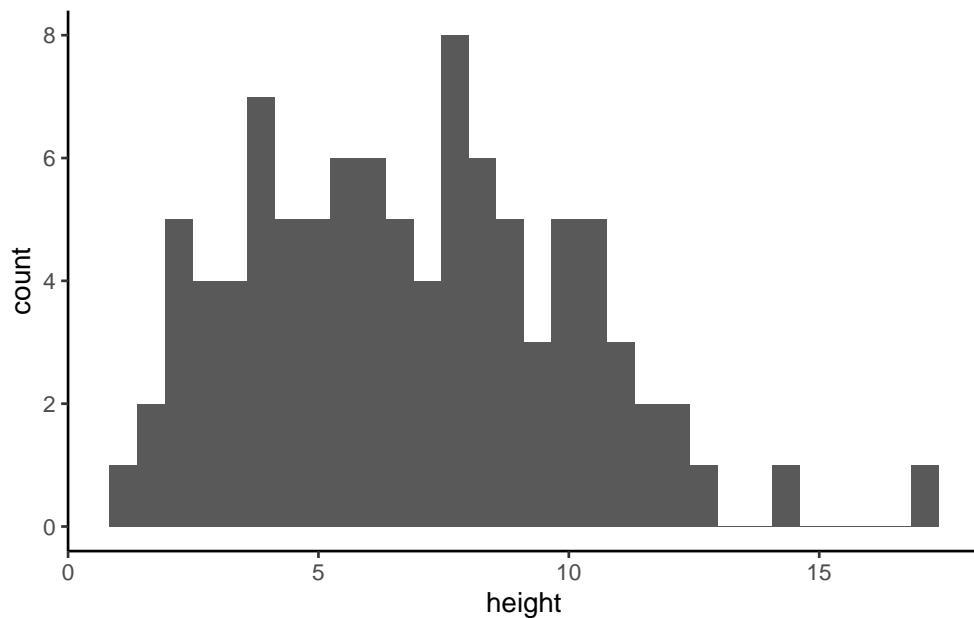


Figure 4.9.: Histogramme des valeurs de taille avec ggplot2

Les fonctions `hist()` et `geom_histogram()` créent automatiquement les points de rupture (ou “bins”) dans l’histogramme, à moins que vous n’indiquiez le contraire à l’aide de l’argument `breaks` = pour spécifier le contraire. Par exemple, disons que nous voulons tracer notre histogramme avec des points de rupture tous les 1 cm de taille des licornes. Nous générerons d’abord une séquence allant de zéro à la valeur maximale de `height` (18 arrondi vers le haut) par pas de 1 à l’aide de la fonction `seq()`. Nous pouvons ensuite utiliser cette séquence avec l’argument `breaks` =. Tant qu’on y est, remplaçons également le titre moche par quelque chose d’un peu mieux en utilisant l’argument `main` =.

```
brk <- seq(from = 0, to = 18, by = 1)
hist(licornes$height, breaks = brk, main = "Taille des licornes")
```

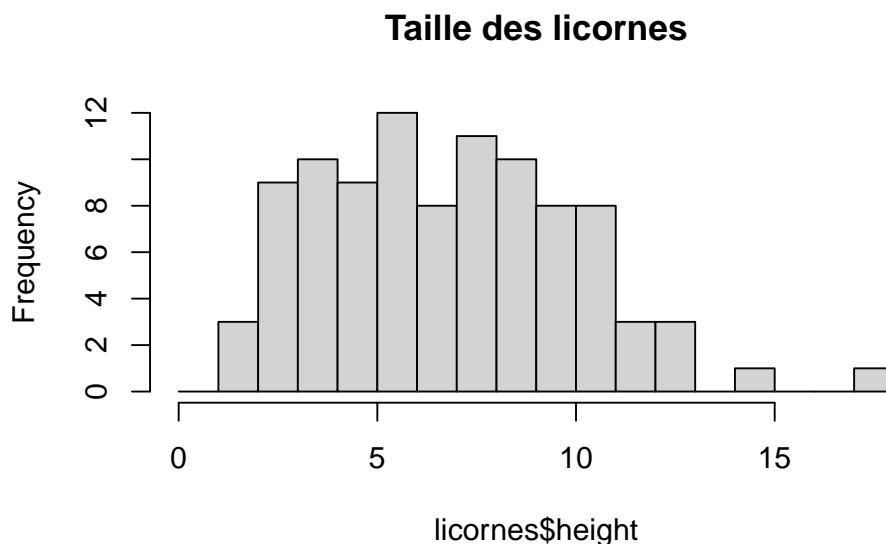


Figure 4.10.: Histogramme des valeurs de taille avec des points de ruptures à chaque cm (R de base).

```
brk <- seq(from = 0, to = 18, by = 1)
ggplot(licornes, aes(x = height)) +
  geom_histogram(breaks = brk) +
  ggtitle("Taille des licornes")
```



Figure 4.11.: Histogramme des valeurs de taille avec des points de ruptures à chaque cm (ggplot2).

Vous pouvez aussi afficher l'histogramme sous forme de proportion plutôt que de fréquence en utilisant l'argument `freq = FALSE` dans la fonction `hist()` ou en indiquant `aes(y = after_stat(density))` dans `geom_histogram()`.

```
brk <- seq(from = 0, to = 18, by = 1)

hist(licornes$height,
  breaks = brk, main = "Taille des licornes",
  freq = FALSE
)
ggplot(licornes, aes(x = height)) +
  geom_histogram(aes(y = after_stat(density)), breaks = brk) +
  ggtitle("Taille des licornes")
```

Une alternative au tracé d'un simple histogramme est d'ajouter une **densité du noyau ("Kernel density")** au graphique. Dans la version de base de R, vous devez d'abord calculer les estimations de la densité du noyau à l'aide de la fonction `density()` puis ajouter les estimations à un graphique sous forme de ligne à l'aide de la fonction `lines()`.

```
dens <- density(licornes$height)
hist(licornes$height,
  breaks = brk, main = "Taille des licornes",
```

```

freq = FALSE
)
lines(dens)

```

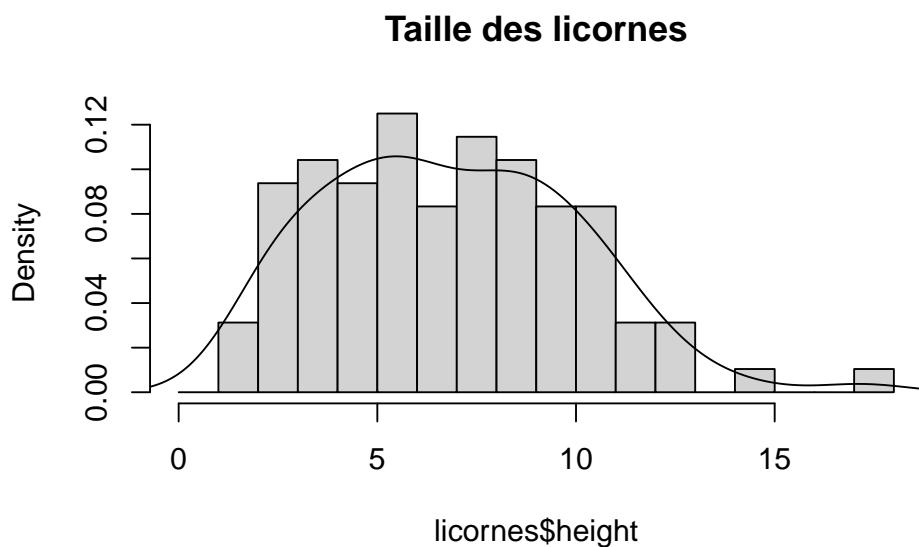


Figure 4.12.: Histogramme des valeurs de taille avec la courbe de densité (avec R de base).

Avec `ggplot2` 📈, vous pouvez simplement ajouter la fonction `geom_density()` au graphique :

```

ggplot(licornes, aes(x = height)) +
  geom_histogram(aes(y = after_stat(density)), breaks = brk) +
  geom_density() +
  ggtitle("Taille des licornes")

```

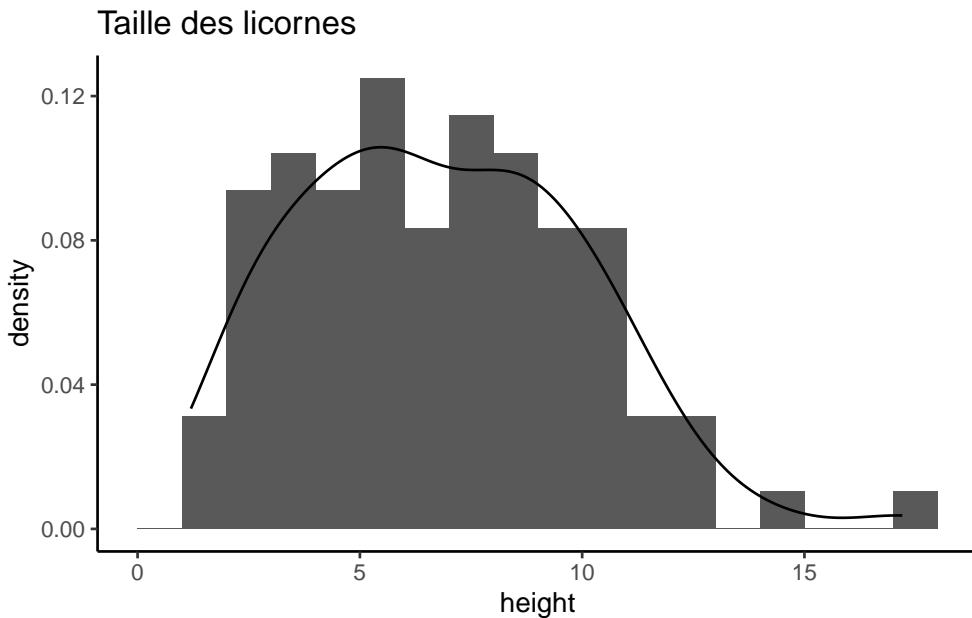


Figure 4.13.: Histogramme des valeurs de taille avec la courbe de densité (avec ggplot2 ).

4.3.3. Boîtes à moustache (Box plots)

D'accord, nous allons le dire franchement, nous adorons les boîtes à moustache et leur relation étroite avec le diagramme en violon. Les “boxplots” (ou “box-and-whisker” plots pour leur nom complet) sont très utiles pour résumer graphiquement la distribution d'une variable, identifier d'éventuelles valeurs inhabituelles et comparer les distributions entre différents groupes. La raison pour laquelle nous les aimons est leur facilité d'interprétation, leur transparence et leur rapport données-encre relativement élevé (c'est-à-dire qu'ils transmettent efficacement une grande quantité d'informations). Nous vous suggérons d'utiliser les “boxplots” autant que possible lorsque vous explorez vos données et de résister à la tentation d'utiliser les diagrammes en bâtons (“barplots”) plus omniprésent (même avec des barres d'erreur standard ou d'intervalle de confiance à 95 %). Le problème des diagrammes en bâtons (ou diagrammes en dynamite) est qu'ils cachent au lecteur des informations importantes telles que la distribution des données et qu'ils supposent que les barres d'erreur (ou les intervalles de confiance) sont symétriques par rapport à la moyenne. Bien sûr, c'est à vous de décider ce que vous faites, mais si vous êtes tenté d'utiliser des diagrammes en bâtons, cherchez “dynamite plots are evil” (les diagrammes de dynamite sont diaboliques) ou voyez [ici](#) ou [ici](#) pour une discussion plus complète.

Pour créer un “boxplot” dans R, nous utilisons la fonction `boxplot()`. Par exemple, créons un “boxplot” de la variable `weight` du jeu de données `licornes`. Nous pouvons également inclure un nom pour l'axe des y en utilisant l'argument `ylabel =`.

```
boxplot(licornes$weight, ylab = "Poids (g)")
```

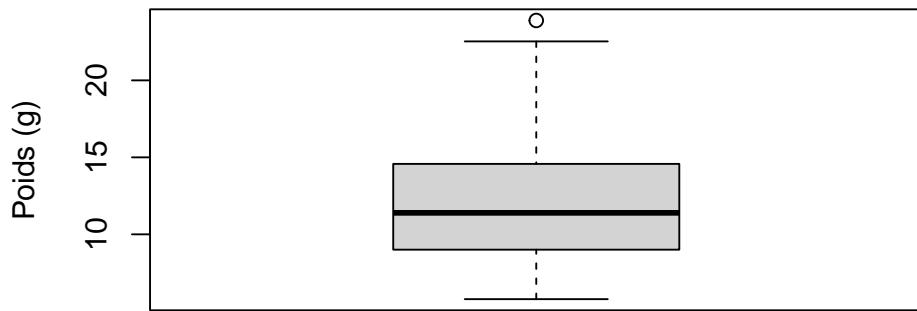


Figure 4.14.: Boîte à moustache du poids des licornes (avec R de base).

```
ggplot(licornes, aes(y = weight)) +  
  geom_boxplot() +  
  labs(y = "Poids (g)")
```

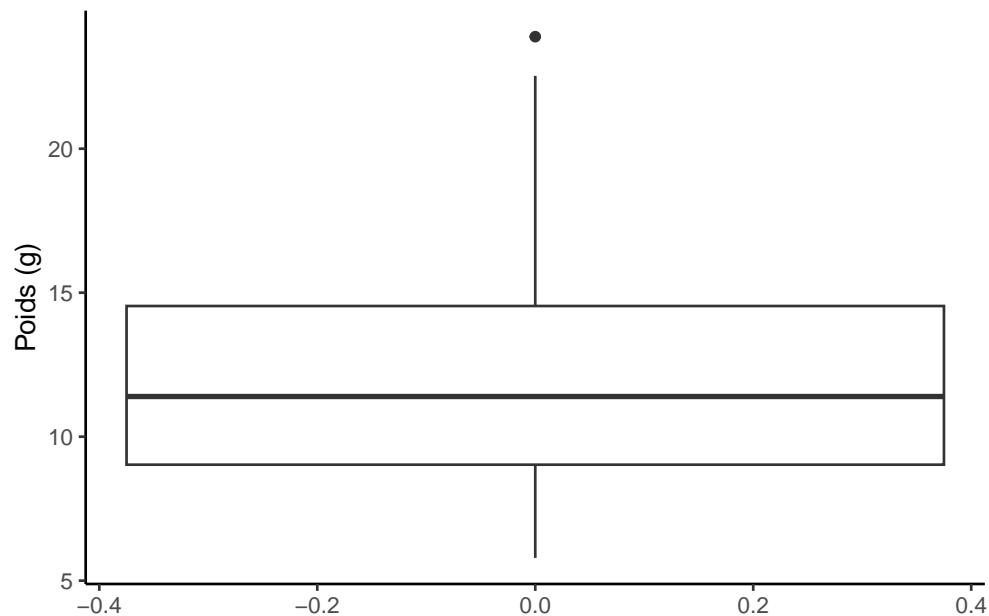


Figure 4.15.: Boîte à moustache du poids des licornes (avec “ggplot2” ).

La ligne horizontale épaisse au milieu de la boîte est la médiane de la variable `weight` (environ 11 g). La ligne supérieure de la boîte est le quartile supérieur (75^e percentile) et la ligne inférieure est le quartile inférieur (25^e percentile). La distance entre les quartiles supérieur et inférieur est appelée l'intervalle interquartile et représente les valeurs de `weight` pour 50 % des données. Les lignes verticales en pointillés sont appelées moustaches et leur longueur est égale à 1,5 x l'intervalle interquartile. Les points tracés en dehors des moustaches représentent des observations inhabituelles potentielles. Cela ne signifie pas qu'ils sont inhabituels, mais simplement qu'ils méritent un examen plus approfondi. Nous recommandons d'utiliser les “boxplots” en combinaison avec les “dotplots” de Cleveland pour identifier les observations potentiellement inhabituelles (voir la Section 4.3.5 pour plus de détails). Ce qui est intéressant avec les “boxplots”, c'est qu'ils ne fournissent pas seulement une mesure de la tendance centrale (médiane), mais qu'ils vous donnent également une idée de la distribution des données. Si la ligne médiane se trouve plus ou moins au milieu de la boîte (entre les quartiles supérieur et inférieur) et que les moustaches sont plus ou moins de la même longueur, vous pouvez être raisonnablement sûr que la distribution de vos données est symétrique.

Si nous voulons examiner comment la distribution d'une variable change entre différents niveaux d'un facteur, il faut utiliser la notation de formule avec la fonction `boxplot()`. Par exemple, traçons notre variable `weight` mais cette fois-ci, voyons comment elle évolue entre chaque niveau de la variable `food`. Lorsque nous utilisons la notation de formule avec `boxplot()` nous pouvons utiliser l'argument `data =` afin d'économiser de la frappe. Nous introduirons également un nom pour l'axe des x à l'aide de l'argument `xlab =`.

```
boxplot(weight ~ food,
        data = licornes,
        ylab = "Poids (g)", xlab = "Niveau de nutrition"
)
```

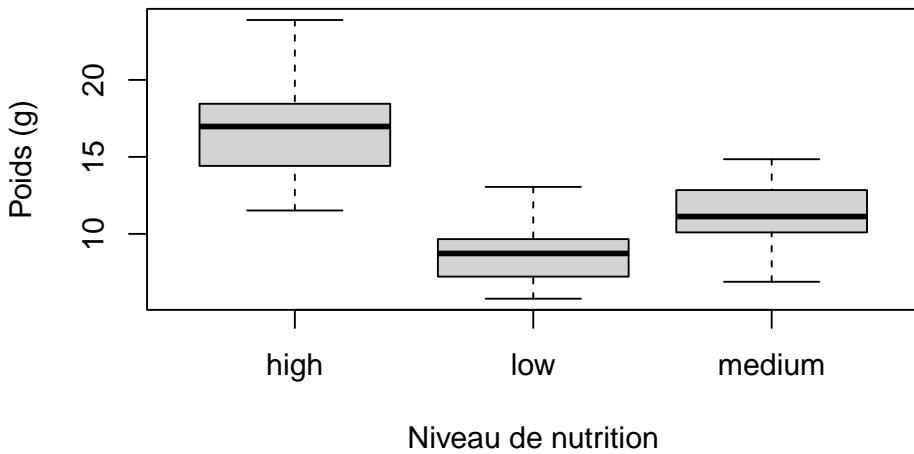


Figure 4.16.: Comparaison du poids des licornes en fonction du niveau de nutrition (avec R de base).

```
ggplot(licornes, aes(y = weight, x = food)) +
  geom_boxplot() +
  labs(y = "Poids (g)", x = "Niveau de nutrition")
```

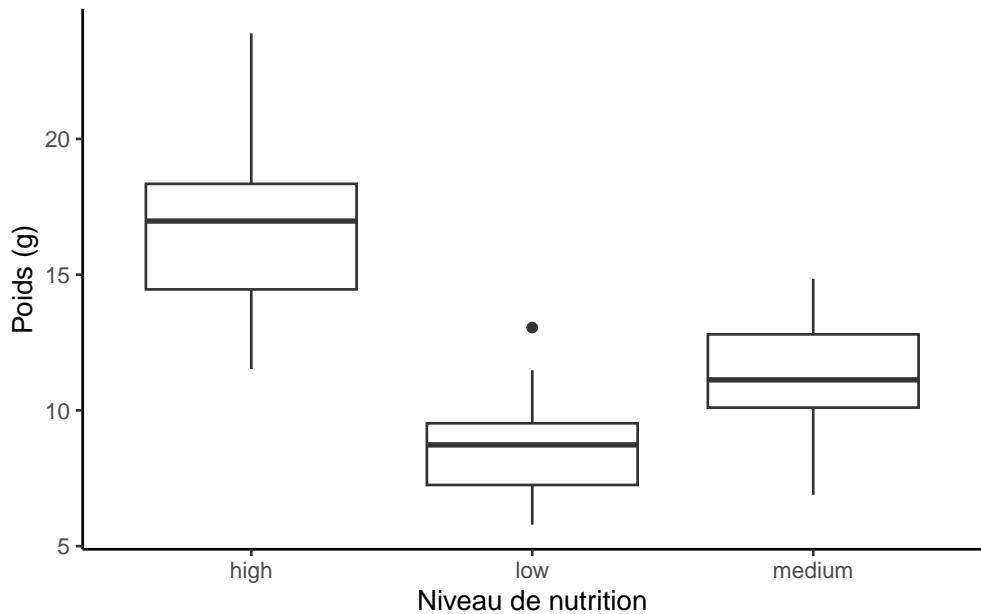


Figure 4.17.: Comparaison du poids des licornes en fonction du niveau de nutrition (avec ggplot2 📦).

Les niveaux des facteurs sont représentés dans l'ordre défini par notre variable factorielle `food` (souvent par ordre alphabétique). Pour modifier l'ordre, il faut changer l'ordre des niveaux du facteur `food` dans le jeu de données à

l'aide de la fonction `factor()` puis redessiner le graphique. Traçons notre boîte à moustache avec nos niveaux de facteurs allant de `low` à `high`.

```
licornes$food <- factor(licornes$food,
  levels = c("low", "medium", "high"))

ggplot(licornes, aes(y = weight, x = food)) +
  geom_boxplot() +
  labs(y = "Poids (g)", x = "Niveau de nutrition")
```

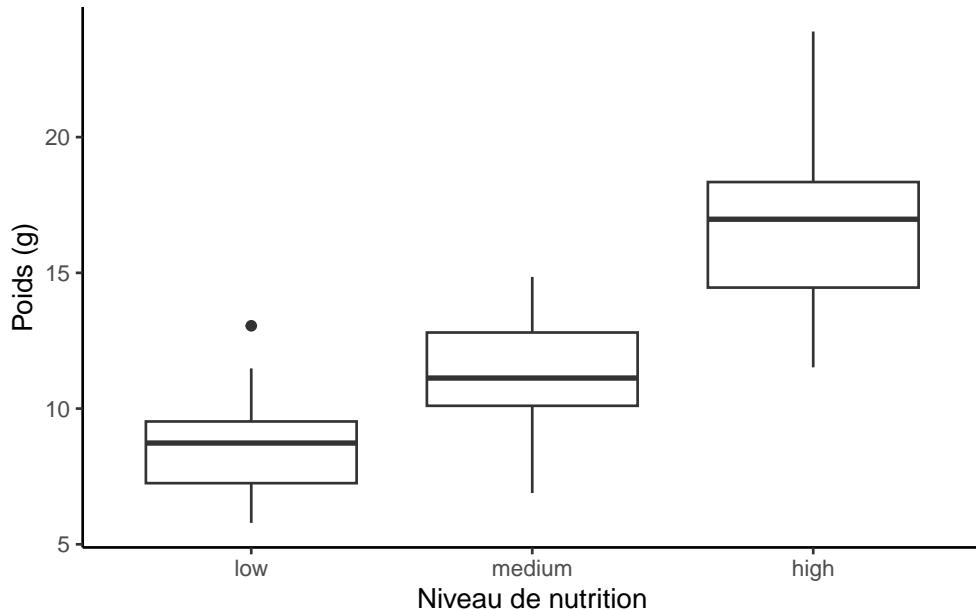


Figure 4.18.: Comparaison du poids des licornes en fonction du niveau (trié) de nutrition (avec `ggplot2` 📦).

Nous pouvons également regrouper nos variables selon deux facteurs dans le même graphique. Traçons notre `weight` mais, cette fois, traçons une boîte séparée pour chaque niveau `food` et le traitement des soins parentaux (`p_care`).

```
boxplot(weight ~ food * p_care,
  data = licornes,
  ylab = "Poids (g)", xlab = "Niveau de nutrition"
)
```

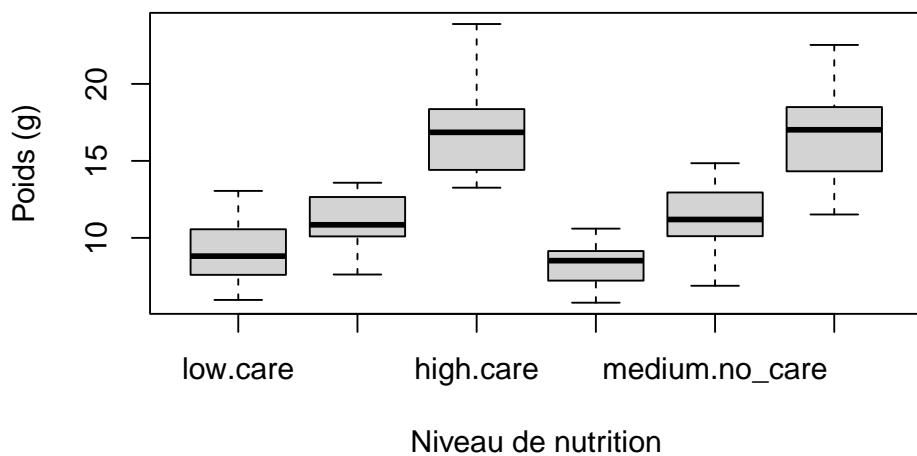


Figure 4.19.: Comparaison du poids des licornes en fonction du niveau de nutrition et de soins parentaux (avec R de base).

```
ggplot(licornes, aes(y = weight, x = food)) +  
  geom_boxplot() +  
  labs(y = "Poids (g)", x = "Niveau de nutrition") +  
  facet_grid(.  
 ~ p_care)
```

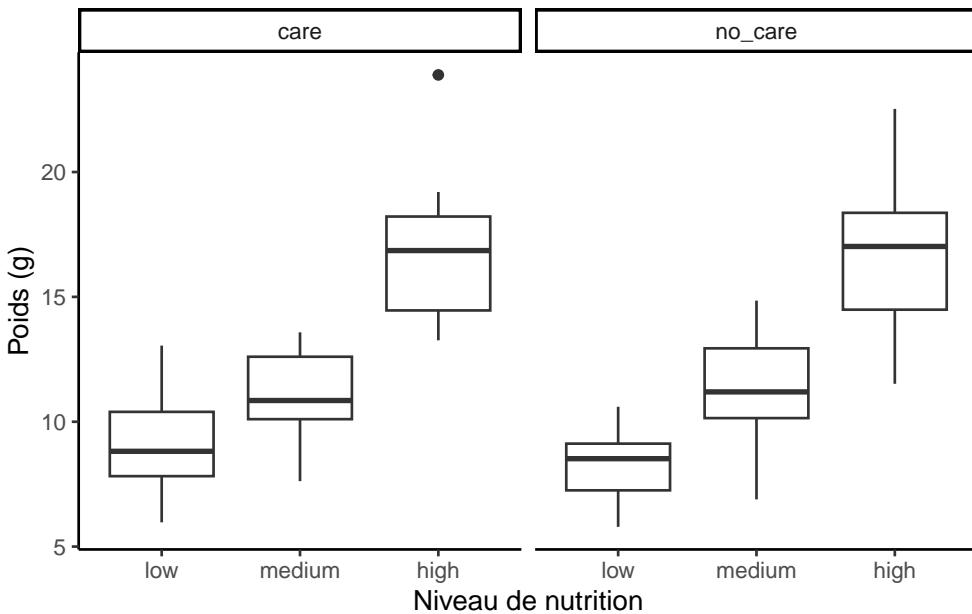


Figure 4.20.: Comparaison du poids des licornes en fonction du niveau de nutrition et de soins parentaux (avec ggplot2 📦).

Ce graphique est beaucoup plus intéressant dans ggplot2 📦, l'utilisation de `facet_grid()` permettant de réaliser des graphiques similaires en fonction d'une troisième (ou même d'une quatrième) variable.

4.3.4. Diagrammes en violon (Violin plots)

Les diagrammes en violon, ou “violin plots”, sont une combinaison d'un “boxplot” et d'un diagramme de densité de noyau (vous avez vu un exemple de diagramme de densité de noyau dans la section histogramme ci-dessus), le tout en une seule figure. Nous pouvons créer un diagramme en violon dans R à l'aide de la fonction `vioplot()` paquet `vioplot` 📦. Vous devrez d'abord installer ce paquet en utilisant la commande `install.packages('vioplot')` (comme d'habitude). L'avantage de l'option `vioplot()` est qu'elle s'utilise à peu près de la même manière que la fonction `boxplot()`. Nous utiliserons également l'argument `col = "lightblue"` pour changer la couleur de remplissage en bleu clair.

```
library(vioplot)
vioplot(weight ~ food,
        data = licornes,
        ylab = "Poids (g)", xlab = "Niveau de nutrition",
        col = "lightblue")
```

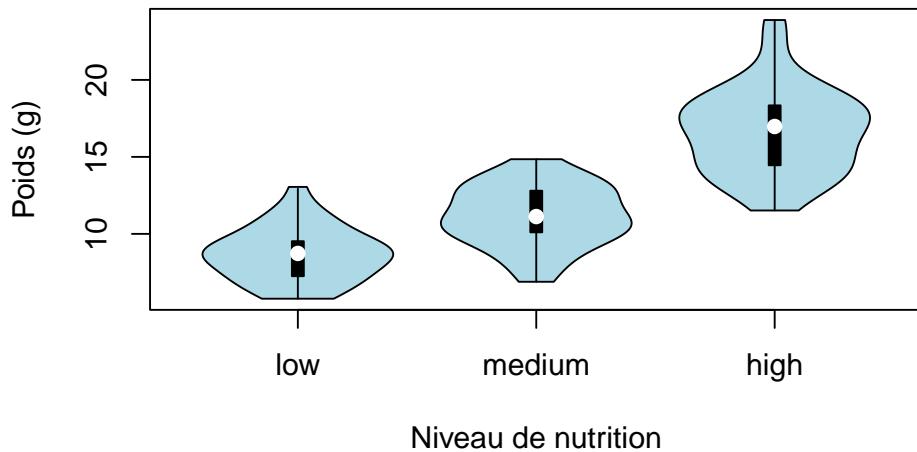


Figure 4.21.: Diagramme en violon comparant le niveau poids des licornes en fonction de leurs niveau de nutrition (avec R de base).

Dans le diagramme en violon ci-dessus, nous avons des boîtes familiaires, comme dans un “boxplot”, pour chaque niveaux de `food` mais cette fois, la valeur médiane est représentée par un cercle blanc. Autour de chaque boîte figure le diagramme de densité de noyau qui représente la distribution des données pour chaque niveau d’alimentation.

```
ggplot(licornes, aes(y = weight, x = food)) +
  geom_violin() +
  geom_boxplot(width = 0.1) +
  labs(y = "Poids (g)", x = "Niveau de nutrition")
```

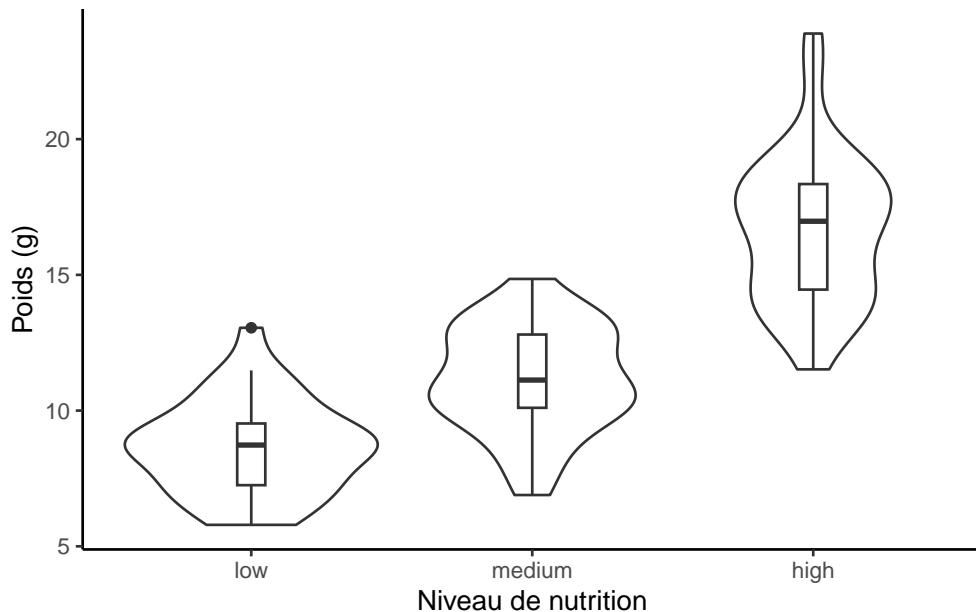


Figure 4.22.: Diagramme en violon comparant le niveau poids des licornes en fonction de leurs niveau de nutrition (avec ggplot2).

4.3.5. Diagramme en pointillés (Dot charts)

Il est extrêmement important d'identifier les observations inhabituelles (appelées “valeurs aberrantes”) dans une variable numérique, car elles peuvent influencer les estimations des paramètres de votre modèle statistique ou indiquer une erreur dans vos données. Un graphique très utile (bien que sous-estimé) pour aider à identifier les valeurs aberrantes est le diagramme en pointillés de Cleveland (“Cleveland’s dotplot”). Vous pouvez produire un diagramme en pointillés en R très simplement en utilisant la fonction `dotchart()`.

```
dotchart(licornes$height)
```

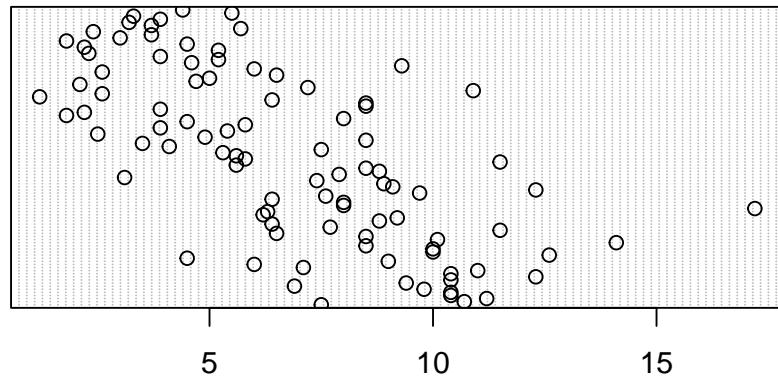


Figure 4.23.: Diagramme en pointillés de la taille des licornes (avec R de base).

Dans le diagramme en pointillés ci-dessus, les données de taille (`height`) sont représentées le long de l'axe des x et les données sont représentées dans l'ordre dans lequel elles apparaissent dans le jeu de données `licornes` sur l'axe des y (les valeurs situées en haut de l'axe des y apparaissent plus tard dans le jeu de données et celles situées plus bas apparaissent au début du jeu de données). Sur ce graphique, une seule valeur s'étend vers la droite à environ 17 cm, mais elle ne semble pas particulièrement importante par rapport aux autres. Un exemple de diagramme à points avec une observation inhabituelle est donné ci-dessous :

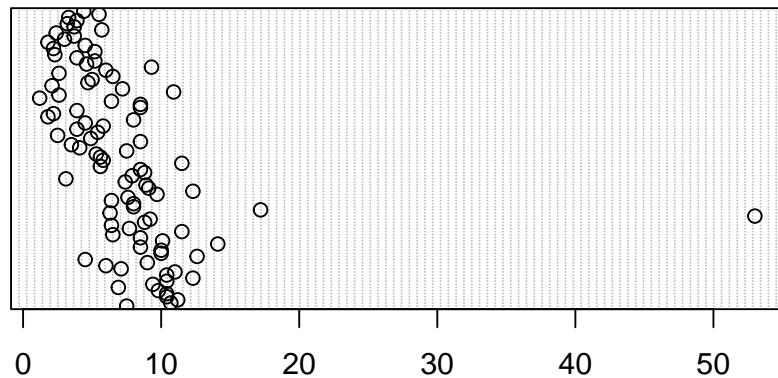


Figure 4.24.: Diagramme en pointillés de la taille des licornes avec une valeur aberrante (avec R de base).

Nous pouvons également regrouper les valeurs dans notre variable `height` selon une variable factorielle telle que `food` en utilisant l'argument `groups =`. Ceci est utile pour identifier des observations inhabituelles au sein d'un niveau de facteur qui pourraient être masquées lorsque l'on examine toutes les données ensemble.

```
dotchart(licornes$height, groups = licornes$food)
```

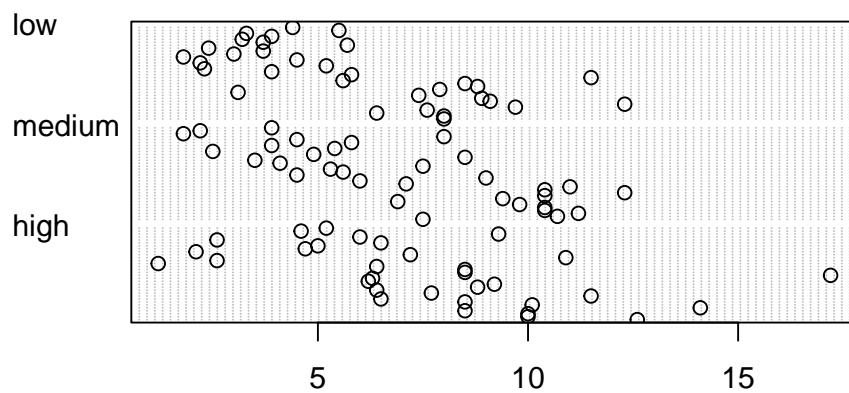


Figure 4.25.: Diagramme en pointillés de la taille des licornes en fonction des niveaux de nutrition (avec R de base).

```
ggdotchart(data = licornes, x = "height", y = "food")
```

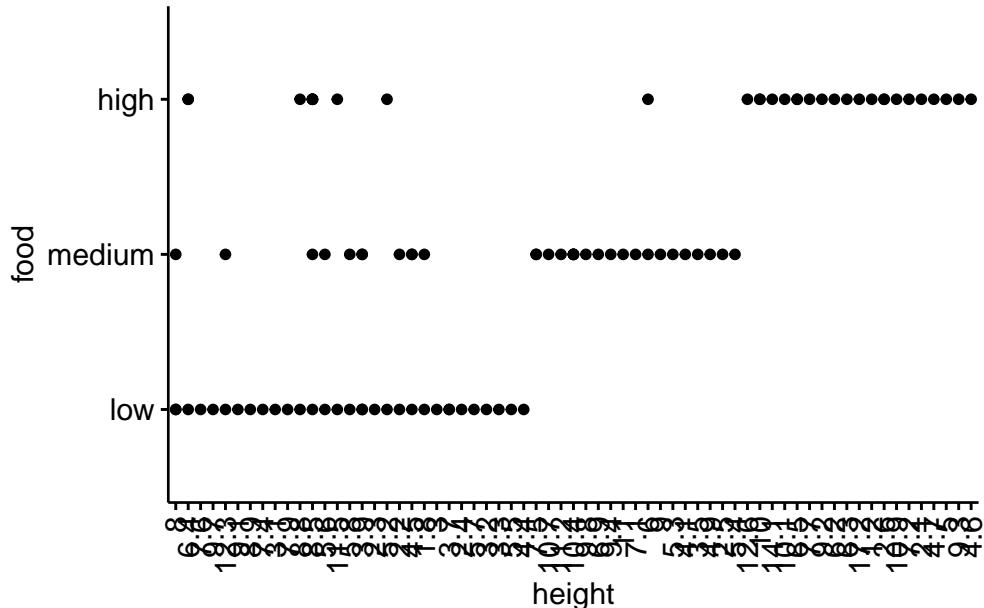


Figure 4.26.: Diagramme en pointillés de la taille des licornes en fonction des niveaux de nutrition (avec `ggpubr`).

4.3.6. Diagrammes en paires

Dans ce chapitre, nous avons déjà utilisé la fonction `plot()` pour créer un nuage de points afin d'explorer la relation entre deux variables numériques. Dans le cas de jeux de données contenant de nombreuses variables numériques, il est souvent utile de créer plusieurs diagrammes de dispersion pour visualiser les relations entre toutes ces variables. Nous pouvons utiliser la fonction `plot()` pour créer chacun de ces graphiques individuellement, mais il est beaucoup plus facile d'utiliser la fonction `pairs()`. La fonction `pairs()` crée un nuage de points multi-panneaux (parfois appelé matrice de nuage de points) qui représente toutes les combinaisons de variables. Créons un nuage de points multi-panneaux de toutes les variables numériques de notre jeu de données `licornes`. Notez que vous devrez peut-être cliquer sur le bouton “Zoom” dans RStudio pour afficher clairement le graphique.

```
pairs(licornes[, c(
  "height", "weight", "mane_size",
  "fluffyness", "horn_rings"
)])
```

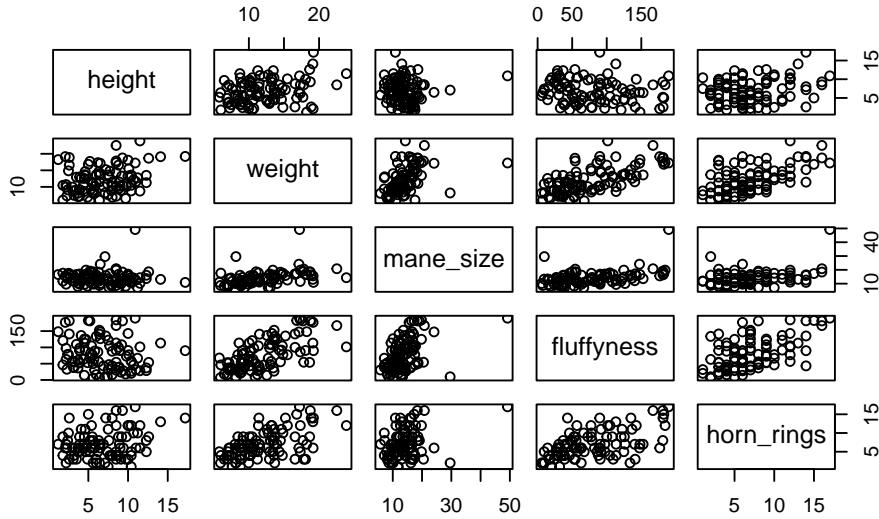


Figure 4.27.: Nuage de points multi-panneaux représentants les relations entre toutes les combinaisons de variables du jeu de données licornes (avec R de base).

```
# Ou, de manière équivalente, vous pouvez utiliser :
# pairs(licornes[, 4:8])
```

Il faut un peu de temps pour s'habituer à l'interprétation d'un diagramme en paires. Les panneaux sur la diagonale indiquent les noms des variables. La première rangée de graphiques affiche les `height` sur l'axe des y et les variables `weight`, `mane_size`, `fluffyness` et `horn_rings` sur l'axe des x pour chacun des quatre graphiques respectivement. La rangée suivante de graphiques comporte `weight` sur l'axe des y et `height`, `mane_size`, `fluffyness` et `horn_rings` sur l'axe des x. Nous interprétons les autres lignes de la même manière, la dernière ligne affichant la variable `horn_size` sur l'axe des y et les autres variables sur l'axe des x. Nous espérons que vous remarquerez que les graphiques situés sous la diagonale sont les mêmes que ceux situés au-dessus de la diagonale, mais avec les axes inversés.

Pour réaliser des diagrammes en paires avec `ggplot2` 📦, vous avez besoin de la fonction `ggpairs()` du paquet `GGally` 📦. La sortie est assez similaire mais vous n'avez que la partie inférieure de la matrice des graphiques, vous obtenez un graphique de densité sur la diagonale et les corrélations sur la partie supérieure du graphique.

```
ggpairs(licornes[, c(
  "height", "weight", "mane_size",
  "fluffyness", "horn_rings"
)])
```

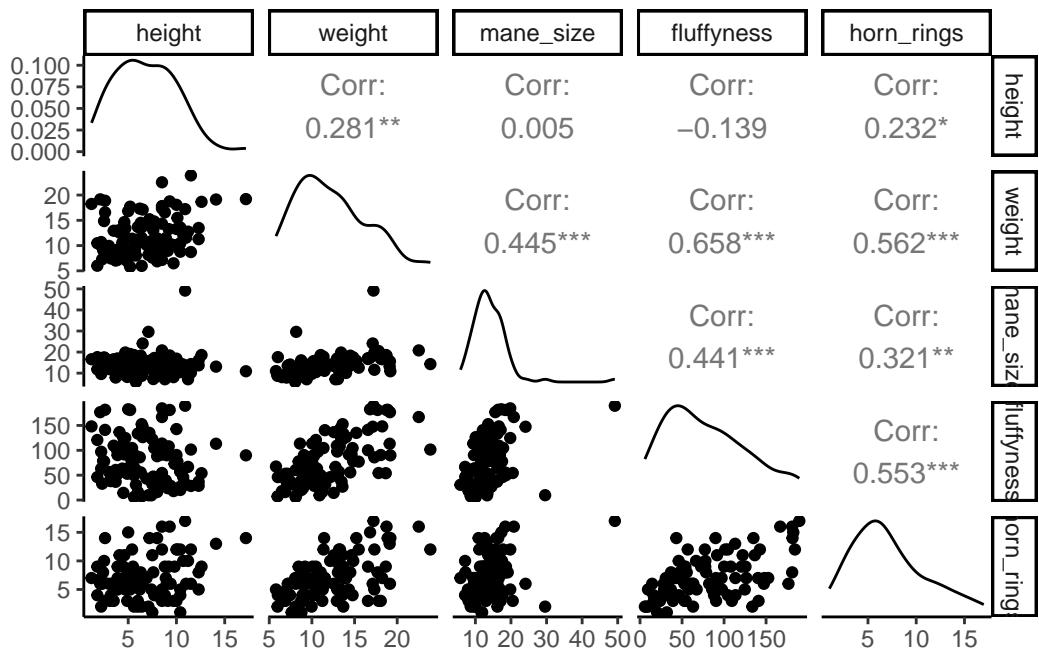


Figure 4.28.: Nuage de points multi-panneaux représentants les relations entre toutes les combinaisons de variables du jeu de données licornes(avec GGally ).

La fonction `pairs()` peut être modifiée pour faire des choses similaires et plus, mais elle est plus complexe. Jetez un coup d'œil à l'excellent fichier d'aide de la fonction `pairs()` (`?pairs`), qui fournit tous les détails permettant de faire quelque chose comme la figure ci-dessous.

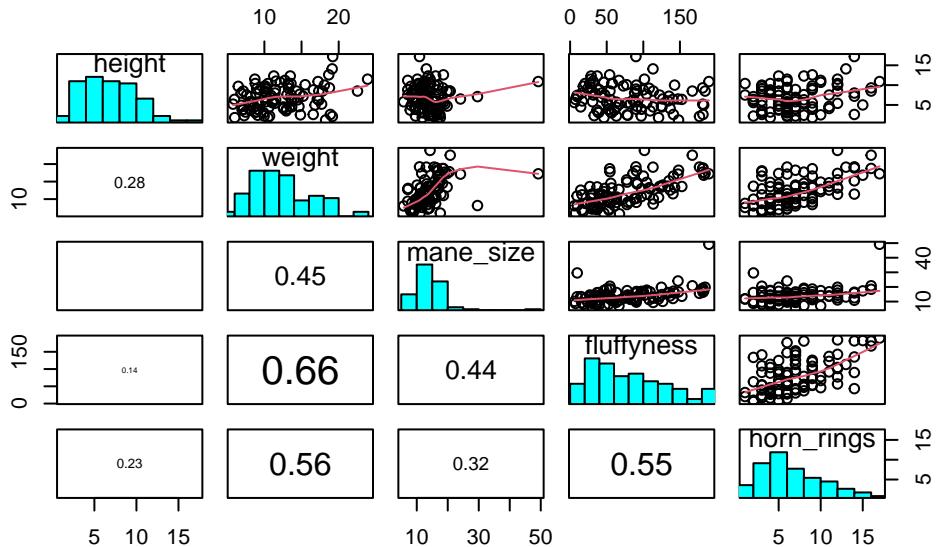


Figure 4.29.: Nuage de points multi-panneaux représentants les relations entre toutes les combinaisons de variables du jeu de données licornes(avec R de base mais en plus classe).

4.3.7. Graphique de conditionnement (Coplots)

Lorsque l'on examine la relation entre deux variables numériques, il est souvent utile de pouvoir déterminer si une troisième variable obscurcit ou modifie la relation. Un graphique très pratique à utiliser dans ce genre de situations est le graphique de conditionnement (également connu sous le nom de graphique de dispersion conditionnel, ou encore “coplots”) que nous pouvons créer dans R à l'aide de la fonction `coplot()`. La fonction `coplot()` trace les graphiques de deux variables, mais chaque graphique est conditionné (`|`) par une troisième variable. Cette troisième variable peut être soit numérique, soit factorielle. À titre d'exemple, voyons comment la relation entre le nombre d'anneaux sur la corne (`horn_ring`) et le poids (`weight`) des licornes change en fonction de la taille de la crinière (`mane_size`). Notez que la fonction `coplot()` a un argument `data` = il n'est donc pas nécessaire d'utiliser la notation `$`.

```
coplot(horn_rings ~ weight | mane_size, data = licornes)
```

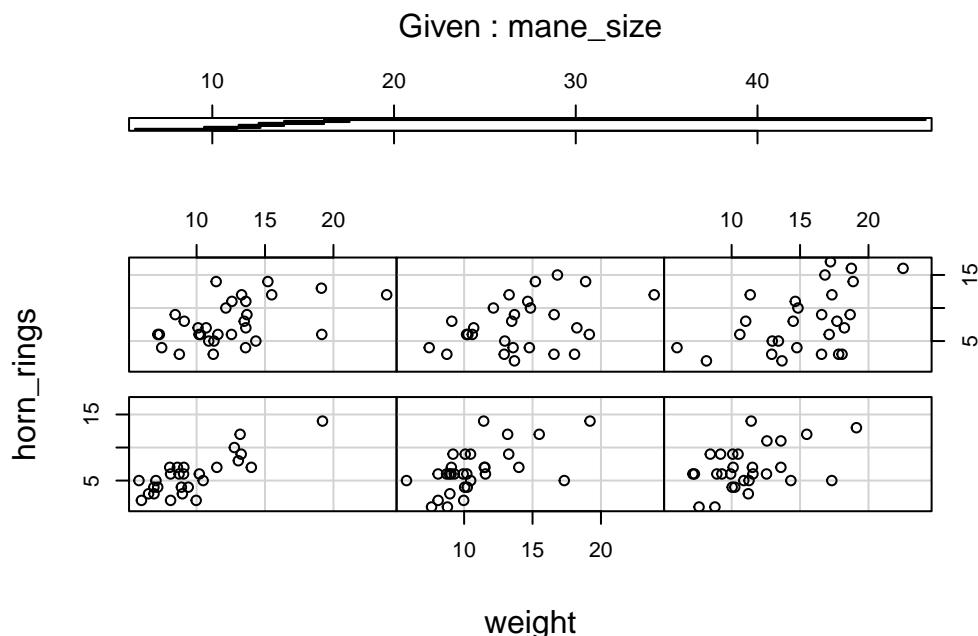


Figure 4.30.: Évolution de la relation entre le nombre d'anneaux sur la corne et le poids en fonction de la taille de la crinière chez les licornes (avec R de base).

Pour avoir le même type de graphiques avec `ggplot2` 📈, on peut utiliser la fonction `gg_coplot()` du paquet `ggleveland` 📦 :

```
gg_coplot(licornes,
  x = weight, y = horn_rings,
  facetting = mane_size
)

`geom_smooth()` using formula = 'y ~ x'
```

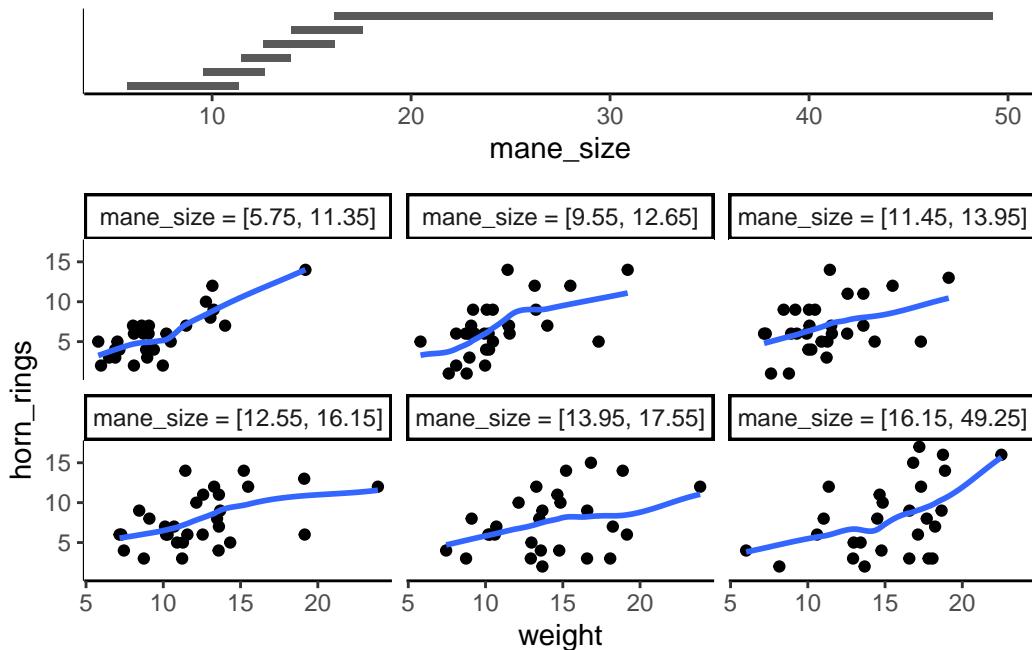


Figure 4.31.: Évolution de la relation entre le nombre d’anneaux sur la corne et le poids en fonction de la taille de la crinière chez les licornes (avec `ggcleveland`).

Il faut un peu de pratique pour interpréter les “coplots”. Le nombre d’anneaux sur la corne est représenté sur l’axe des y et le poids des licornes sur l’axe des x. Les six graphiques montrent la relation entre ces deux variables pour différents intervalles de taille de crinière. Le diagramme en barres en haut indique la plage de valeurs de tailles de crinières pour chacun des graphiques. Les graphiques sont lus de haut à gauche (en premier) jusqu’à bas à droite le long de chaque ligne. Par exemple, le graphique en haut à gauche montre la relation entre le nombre d’anneaux sur la corne et le poids des licornes dont la crinière est la plus courte (environ 5 - 11 cm). Le graphique en bas à droite montre la relation entre le nombre d’anneaux sur la corne et le poids des licornes dont la crinière est comprise entre 16 et 50 cm de long.

Remarquez que la plage de valeurs de la taille de la crinière diffère d’un graphique à l’autre et que les plages se chevauchent d’un graphique à l’autre. La fonction `coplot()` fait de son mieux pour diviser les données afin de s’assurer qu’il y a un nombre adéquat de points de données dans chaque graphique. Si vous ne souhaitez pas produire

des graphiques avec des données qui se chevauchent dans le graphique, vous pouvez définir l'option `overlap` = ("chevauchement") à `overlap = 0`

Vous pouvez également utiliser la fonction `coplot()` avec des variables de conditionnement factorielles.

Avec `gg_coplot()` (`ggcleveland` ) vous devez d'abord définir le facteur comme numérique avant de tracer le graphique et spécifier `overlap=0`.

Par exemple, nous pouvons examiner la relation entre `horn_rings` et `weight` conditionnées par le facteur `food`. Le graphique en haut à gauche représente la relation entre `horn_rings` et `weight` pour les licornes avec un faible (low) niveau de nutrition. Le graphique en bas à droite montre la même relation mais pour les licornes avec un haut (high) niveau de nutrition.

```
coplot(horn_rings ~ weight | food, data = licornes)
```

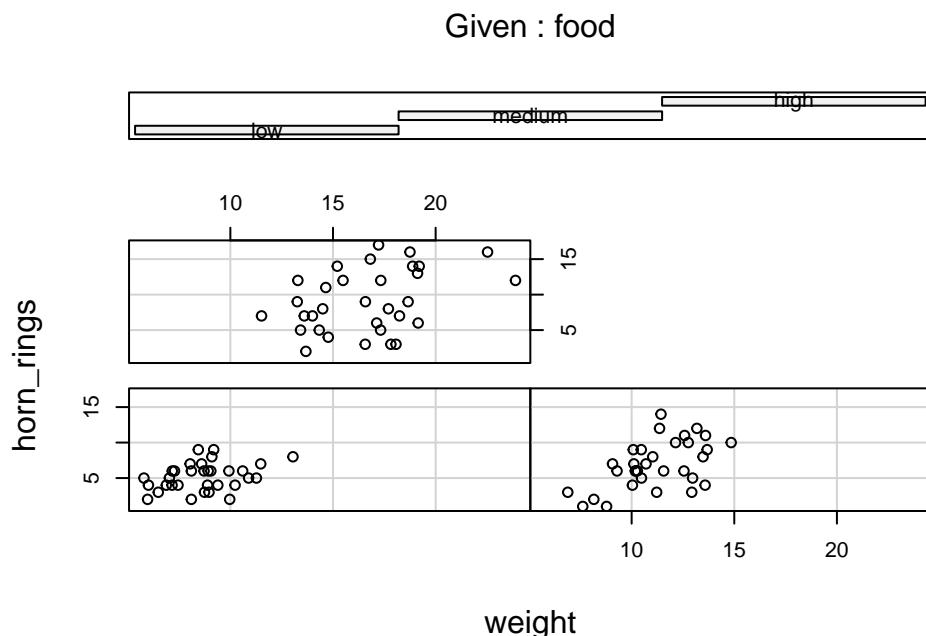


Figure 4.32.: Évolution de la relation entre le nombre d'anneaux sur la corne et le poids en fonction du niveau de nutrition chez les licornes (avec R de base).

```
licornes <- mutate(licornes, food_num = as.numeric(food))

gg_coplot(licornes,
  x = weight, y = horn_rings,
  faceting = food_num, overlap = 0
)
```

```
`geom_smooth()` using formula = 'y ~ x'
```

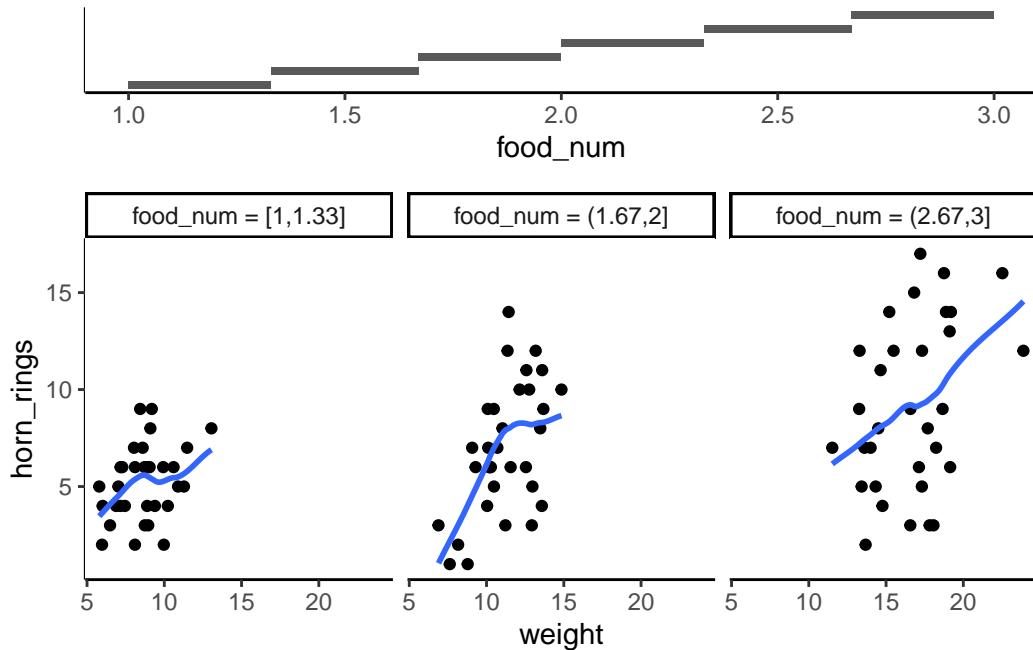


Figure 4.33.: Évolution de la relation entre le nombre d’anneaux sur la corne et le poids en fonction du niveau de nutrition chez les licornes (avec ggcleaveland .

4.3.8. Résumé de la fonction de graphique

Type de graphique	ggplot2	Fonction de base de R
nuage de points	geom_point() plot()	
histogramme de fréquence	geom_histogram()	
boîte à moustache	geom_boxplot() boxplot()	
dotplots de Cleveland	ggdotchart() dotchart()	
diagrammes en paires	ggpairs() pairs()	
graphique de conditionnement	gg_coplot() coplot()	

Nous espérons que vous avez compris qu’il est possible de créer facilement des graphiques exploratoires très instructifs à l'aide des graphiques de base de R ou de ggplot2 .

Le choix de l'un ou l'autre est entièrement libre (c'est ce qui fait l'intérêt de R, vous pouvez choisir) et nous mélangeons volontiers les deux pour répondre à nos besoins. Dans la section suivante, nous verrons comment personnaliser des graphiques de base de R pour leur donner l'aspect que vous souhaitez.

4.4. Graphiques multiples

4.4.1. R de base

Dans la base de R, l'une des méthodes les plus courantes pour tracer plusieurs graphiques consiste à utiliser la fonction graphique principale `par()` pour diviser la fenêtre de graphiques en un certain nombre de sections définies avec l'argument `mfrow =`. Avec cette méthode, il faut d'abord spécifier le nombre de lignes, puis de colonnes de graphiques que vous souhaitez, puis exécuter le code pour chaque graphique. Par exemple, pour tracer deux graphiques côte à côté, nous utiliserions `par(mfrow = c(1, 2))` pour diviser la fenêtre en une ligne et deux colonnes.

```
par(mfrow = c(1, 2))

plot(licornes$weight, licornes$fluffyness,
      xlab = "Poids",
      ylab = "Niveau de douceur",
      main = "a)")

boxplot(fluffyness ~ food, data = licornes, cex.axis = 0.6,
      xlab = "Niveau de nutrition",
      ylab = "Niveau de douceur",
      main = "b)")
```

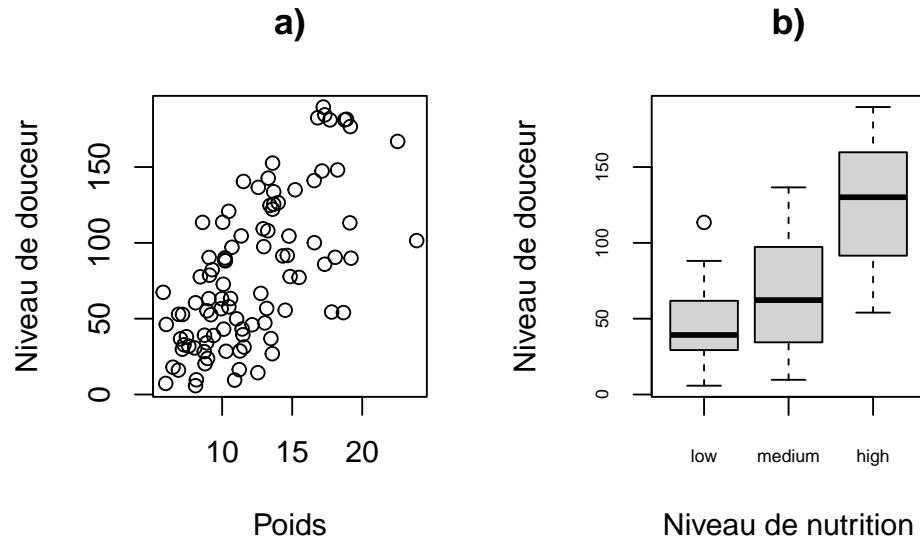


Figure 4.34.: Niveau de douceur des licornes en fonction de leurs a) poids et de leurs b) niveau de nutrition (avec R de base).

Une fois que vous avez terminé vos graphiques, n’oubliez pas de réinitialiser votre fenêtre graphique à la normale avec `par(mfrow = c(1,1))`.

4.4.2. `ggplot2`

En plus des fonctions `facet_grid()` et `facet_wrap()` qui permettent de répéter et d’organiser facilement plusieurs graphiques en fonction de variables spécifiques, il y a beaucoup de manière d’organiser plusieurs graphiques `ggplots2` ensemble. L’approche que nous recommandons est d’utiliser le paquet `patchwork`.

Vous devrez d’abord l’installer (si vous ne l’avez pas encore) et l’appeler (`library(patchwork)`).

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

Une note importante : pour ceux qui ont utilisé la base R pour produire leurs figures et qui sont familiers avec l’utilisation de `par(mfrow = c(2,2))` (qui permet de tracer quatre figures sur deux lignes et deux colonnes), sachez que cela ne fonctionne pas avec les objets de `ggplot2`. Vous devrez utiliser soit le paquet `patchwork` ou d’autres paquets tels que `gridArrange` ou `cowplot` ou convertir le `ggplot` en objets grobs.

Pour tracer les deux graphiques ensemble, il faut assigner chaque figure à un objet distinct, puis utiliser ces objets lorsque l’on utilise `patchwork`.

Nous pouvons donc générer 2 figures et les assigner à des objets. Comme vous pouvez le constater, les figures n'apparaissent pas dans la fenêtre graphique. Elles n'apparaîtront que lorsque vous appellerez l'objet.

```
premiere_figure <- ggplot(
  aes(x = height, y = fluffyness, color = food),
  data = licornes) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
  facet_grid(block ~ p_care)

deuxieme_figure <- ggplot(
  aes(x = weight, y = fluffyness, color = food),
  data = licornes) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
  facet_grid(block ~ p_care)
```

Deux options simples et immédiates s'offrent à nous avec `patchwork` 📦 : disposer les figures les unes sur les autres (spécifiées avec un `/`) ou arranger les figures côte à côté (spécifié soit avec un `+` ou un `|`). Essayons de tracer les deux figures, l'une au-dessus de l'autre.

```
premiere_figure / deuxieme_figure
```

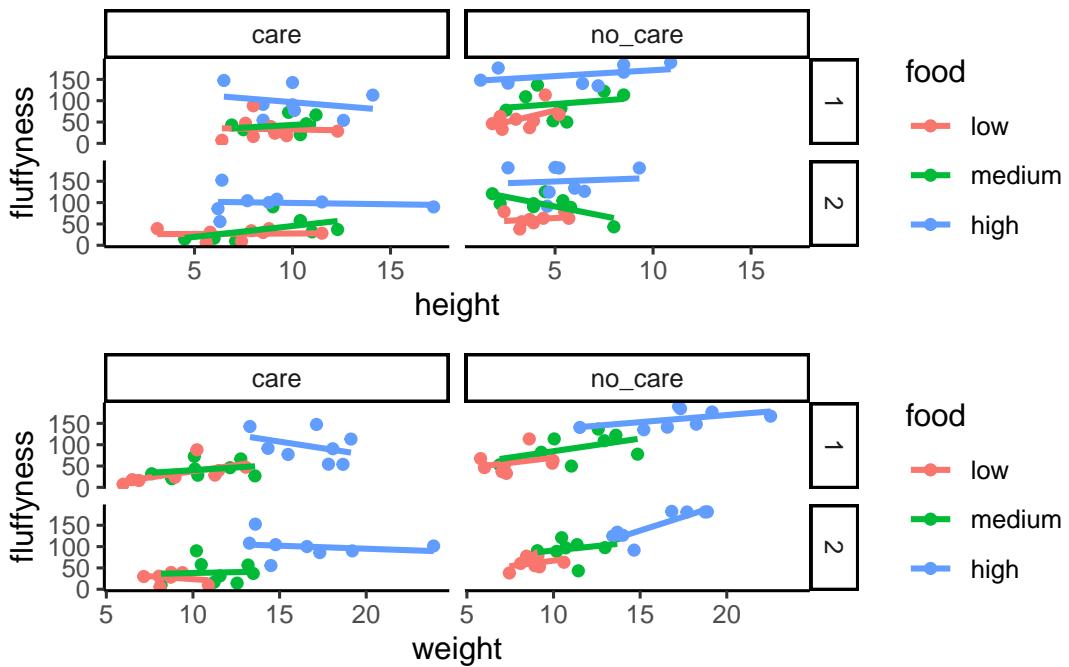


Figure 4.35.: Exemple de disposition de figures l'une sur l'autre avec patchwork 📦.

Jouons un peu: Essayez de créer une version juxtaposée de la figure ci-dessus (indice : essayez les autres opérateurs).

Nous pouvons aller plus loin et assigner des figures imbriquées patchwork 📦 à un objet et l'utiliser à son tour pour créer des étiquettes pour les figures individuelles.

```
figure_imbriquees <- premiere_figure / deuxieme_figure

figure_imbriquees +
  plot_annotation(tag_levels = "A", tag_suffix = ")")
```

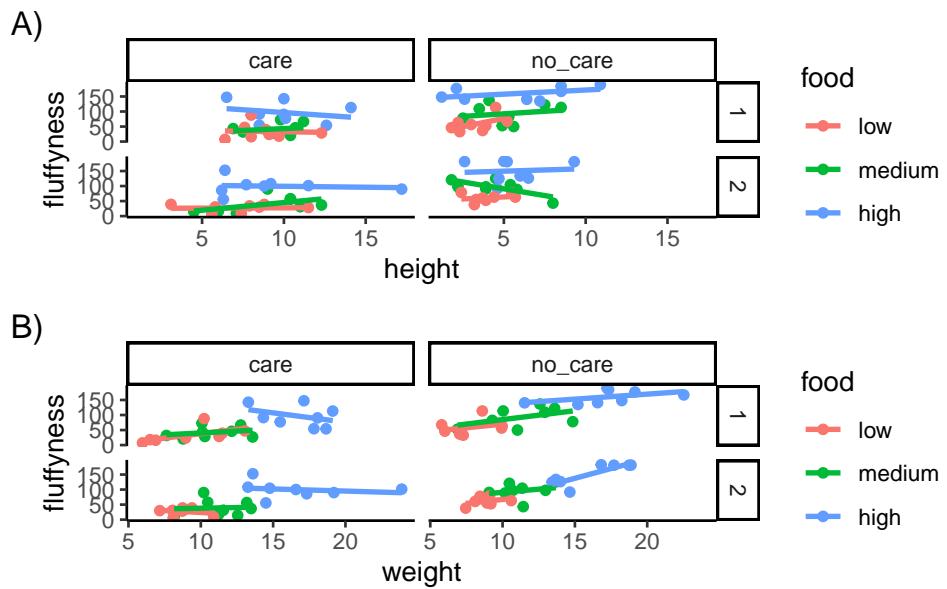


Figure 4.36.: Exemple d'utilisation de figures imbriquées patchowrk 📦.

4.5. Personnalisation de ggplots

Partir se balader pour être édité 🦄

4.6. Exportation des parcelles

Créer des graphiques dans R, c'est bien, mais que faire si vous souhaitez utiliser ces graphiques dans votre thèse, votre rapport ou votre publication ? Une option consiste à cliquer sur le bouton “Exporter” dans l'onglet “Graphiques” de RStudio. Vous pouvez également exporter vos graphiques de R vers un fichier externe en écrivant du code dans votre script R (*option privilégiée*). L'avantage de cette approche est que vous avez un peu plus de contrôle sur le format de sortie et qu'elle vous permet également de générer (ou de mettre à jour) des graphiques automatiquement chaque fois que vous exécutez votre script. Vous pouvez exporter vos figures dans de nombreux formats différents, mais les plus courants sont pdf, png, jpeg et tiff.

Par défaut, R (et donc RStudio) dirige tous les graphiques que vous créez vers la fenêtre de graphiques. Pour enregistrer votre graphique dans un fichier externe, vous devez d'abord rediriger votre graphique vers un périphérique graphique différent. Pour ce faire, vous pouvez utiliser l'une des nombreuses fonctions de périphérique graphique pour démarrer une nouvelle fenêtre graphique. Par exemple, pour enregistrer un graphique au format pdf, nous utiliserons la fonction `pdf()`. Le premier argument de la fonction `pdf()` est le chemin d'accès et le nom du fichier que nous voulons créer

(n'oubliez pas d'inclure l'extension .pdf). Une fois que nous avons utilisé la fonction `pdf()` nous pouvons alors écrire tout le code que nous avons utilisé pour créer notre graphique, y compris les paramètres graphiques tels que le réglage des marges et la division de la fenêtre graphique. Une fois le code exécuté, il faut fermer le dispositif de graphique `pdf` à l'aide de la fonction `dev.off()`.

```
pdf(file = "output/my_plot.pdf")
par(mar = c(4.1, 4.4, 4.1, 1.9), xaxs = "i", yaxs = "i")
plot(licornes$weight, licornes$fluffyness,
      xlab = "weight (g)",
      ylab = expression(paste("shoot area (cm"^(2), ")")),
      xlim = c(0, 30), ylim = c(0, 200), bty = "l",
      las = 1, cex.axis = 0.8, tcl = -0.2,
      pch = 16, col = "dodgerblue1", cex = 0.9
)
text(x = 28, y = 190, label = "A", cex = 2)
dev.off()
```

Si nous voulons sauvegarder ce graphique au format `png`, il nous suffit d'utiliser la fonction `png()`, plus ou moins de la même manière.

```
png("output/my_plot.png")
par(mar = c(4.1, 4.4, 4.1, 1.9), xaxs = "i", yaxs = "i")
plot(licornes$weight, licornes$fluffyness,
      xlab = "weight (g)",
      ylab = expression(paste("shoot area (cm"^(2), ")")),
      xlim = c(0, 30), ylim = c(0, 200), bty = "l",
      las = 1, cex.axis = 0.8, tcl = -0.2,
      pch = 16, col = "dodgerblue1", cex = 0.9
)
text(x = 28, y = 190, label = "A", cex = 2)
dev.off()
```

D'autres fonctions utiles sont : `jpeg()`, `tiff()` et `bmp()`. Des arguments supplémentaires dans ces fonctions vous permettent de modifier la taille, la résolution et la couleur d'arrière-plan de vos images enregistrées. Voir aussi `?png` pour plus de détails.

ggplot2 📦 fournir une fonction très utile : `ggsave()`, qui simplifie grandement la sauvegarde des graphiques, mais ne fonctionne que pour les ggplots.

Après avoir produit un graphique et l'avoir vu dans votre IDE, vous pouvez simplement exécuter `ggsave()` avec l'argument adéquat pour sauvegarder le dernier ggplot produit. Vous pouvez aussi, bien sûr, spécifier quel tracé doit être sauvegardé.

```
ggsave("file.png")
```

Chapitre 5

Programmation

Maintenant que nous avons appris les bases, la prochaine étape importante dans votre aventure sur R est... *la programmation* !

Il existe déjà un grand nombre de paquets R disponibles, ce qui est sûrement plus que suffisant pour couvrir tout ce que vous pourriez vouloir faire ! Alors, *pourquoi alors créer ses propres fonctions R* ? Pourquoi ne pas s'en tenir aux fonctions d'un paquet ?

Eh bien, dans certains cas, vous voudrez personnaliser ces fonctions pré-existantes pour répondre à vos besoins spécifiques. Il se peut aussi que vous souhaitiez mettre en œuvre une nouvelle approche, ce qui signifie qu'il n'y aura pas de paquets déjà développés qui fonctionneront pour vous (Bon, ces deux cas de figure ne sont pas particulièrement courants).

Les fonctions sont principalement utilisées pour faire une chose de manière simple sans avoir à taper le code nécessaire à chaque fois (ce qui n'est pas très intéressant et peut rapidement surcharger votre script). On peut considérer les fonctions comme un raccourci pour copier-coller.

Si vous devez effectuer une tâche similaire quatre fois ou plus, créez une fonction qui exécute cette tâche, et appelez-la simplement quatre fois, ou encore mieux :appelez-la dans une boucle !

5.1. Regarder derrière le rideau

Une bonne façon de commencer à apprendre à programmer en R est de regarder ce que d'autres ont fait avant : Alors, commençons par jeter un bref coup d'œil derrière le rideau !

Pour beaucoup de fonctions en R, si vous voulez jeter un coup d'œil rapide à la machinerie en coulisses, nous pouvons simplement écrire le nom de la fonction, mais sans l'attribut () .

Notez que l'affichage du code source des paquets R de base (ceux qui sont pré-installés avec R) nécessite quelques étapes supplémentaires que nous ne couvrirons pas ici (voir ce [lien](#) si cela vous intéresse), mais pour la plupart des autres paquets que vous installez vous-même, il suffit généralement d'entrer le nom de la fonction sans la mention () pour afficher le code source de la fonction.

Vous pouvez regarder comment la fonction d'ajustement d'un modèle linéaire lm() est construite :

```
lm

function (formula, data, subset, weights, na.action, method = "qr",
model = TRUE, x = FALSE, y = FALSE, qr = TRUE, singular.ok = TRUE,
contrasts = NULL, offset, ...)
{
  ret.x <- x
  ret.y <- y
  cl <- match.call()
  mf <- match.call(expand.dots = FALSE)
  m <- match(c("formula", "data", "subset", "weights", "na.action",
  "offset"), names(mf), 0L)
  mf <- mf[c(1L, m)]
  mf$drop.unused.levels <- TRUE
  mf[[1L]] <- quote(stats::model.frame)
  mf <- eval(mf, parent.frame())
  if (method == "model.frame")
    return(mf)
  else if (method != "qr")
    warning(gettextf("method = '%s' is not supported. Using 'qr'", method),
    domain = NA)
  mt <- attr(mf, "terms")
  y <- model.response(mf, "numeric")
  w <- as.vector(model.weights(mf))
  if (!is.null(w) && !is.numeric(w))
```

```

stop("'weights' must be a numeric vector")

offset <- model.offset(mf)

mlm <- is.matrix(y)

ny <- if (mlm)
       nrow(y)
     else length(y)

if (!is.null(offset)) {

  if (!mlm)
    offset <- as.vector(offset)

  if (NROW(offset) != ny)
    stop(gettextf("number of offsets is %d, should equal %d (number of observations)",
                 NROW(offset), ny), domain = NA)

}

if (is.empty.model(mt)) {

  x <- NULL

  z <- list(coefficients = if (mlm) matrix(NA_real_, 0,
                                              ncol(y)) else numeric(),
            residuals = y, fitted.values = 0 *
              y, weights = w, rank = 0L, df.residual = if (!is.null(w)) sum(w !=
0) else ny)

  if (!is.null(offset)) {

    z$fitted.values <- offset
    z$residuals <- y - offset
  }

}

else {

  x <- model.matrix(mt, mf, contrasts)

  z <- if (is.null(w))

    lm.fit(x, y, offset = offset, singular.ok = singular.ok,
           ...)

  else lm.wfit(x, y, w, offset = offset, singular.ok = singular.ok,
               ...)

}

class(z) <- c(if (mlm) "mlm", "lm")

```

```

z$na.action <- attr(mf, "na.action")
z$offset <- offset
z$contrasts <- attr(x, "contrasts")
z$xlevels <- .getXlevels(mt, mf)
z$call <- cl
z$terms <- mt
if (model)
  z$model <- mf
if (ret.x)
  z$x <- x
if (ret.y)
  z$y <- y
if (!qr)
  z$qr <- NULL
z
}

<bytecode: 0x5f76a57df378>
<environment: namespace:stats>
```

Ce que nous voyons ci-dessus est le code sous-jacent de cette fonction particulière. Nous pourrions le copier et le coller dans notre propre script et y apporter toutes les modifications que nous jugerions nécessaires, mais en faisant preuve de prudence et en testant les changements apportés.

Ne vous inquiétez pas outre mesure si la majeure partie du code contenu dans les fonctions n'a pas de sens dans l'immédiat. C'est parfaitement normale, surtout si vous êtes novice en matière de R, auquel cas cela semble incroyablement intimidant. Honnêtement, cela peut être intimidant même après des années d'expérience avec R.

Pour remédier à cela, nous commencerons par créer nos propres fonctions en R dans la section suivante.

5.2. Fonctions en R

Les fonctions sont le pain et le beurre de R, ce sont les éléments essentiels qui vous permettent de travailler avec R.

Elles sont créées (la plupart du temps) avec le plus grand soin et la plus grande attention, mais peuvent finir par ressembler à un monstre de Frankenstein - avec des membres bizarrement attachés. Mais aussi alambiqués qu'elles

puissent être, ils feront toujours fidèlement la même chose.

Cela signifie que les fonctions peuvent également être très stupides.

Si nous vous demandons d'aller au supermarché pour acheter les ingrédients nécessaires pour faire *un poulet Balmoral* même que vous ne savez pas ce que c'est, vous serez capable de deviner et prendre au moins *quelque articles* (du poulet par exemple). Ou, vous pouvez aussi décider de faire autre chose. Ou vous pouvez demander de l'aide à un chef cuisinier. Ou vous pouvez sortir votre téléphone et chercher sur internet, qu'est ce qu'un *Poulet Balmoral*? Le fait est que, même si nous ne vous avons pas donné suffisamment d'informations pour accomplir la tâche, vous êtes suffisamment intelligent pour, au moins, essayer de trouver une solution.

Si, à la place, nous demandions à une fonction de faire la même chose, elle écouterait attentivement notre demande, puis renverrait simplement une erreur. Elle répéterait cela à chaque fois que nous lui demanderions de faire le travail lorsque la tâche n'est pas claire. Ce qu'il faut retenir ici, c'est que le code et les fonctions ne peuvent pas trouver de solutions de contournement pour pallier des informations mal fournies, ce qui est une excellente chose. C'est à vous de lui dire très explicitement ce qu'elle doit faire, étape par étape.

N'oubliez pas deux choses : l'intelligence du code vient du codeur, pas de l'ordinateur, et les fonctions ont besoin d'instructions exactes pour fonctionner.

Pour éviter que les fonctions ne soient *trop* stupides, vous devez fournir les informations dont la fonction a besoin pour fonctionner. Comme pour l'exemple du *poulet Balmoral* si nous avions fourni une liste d'ingrédients à la fonction, tout se serait bien passé. C'est ce que nous appelons "remplir un argument". La grande majorité des fonctions exigent de l'utilisateur qu'il remplisse au moins un argument.

Ceci est illustré dans le pseudocode ci-dessous. Lorsque l'on crée une fonction, on peut :

- Spécifier les **arguments** que l'utilisateur doit remplir (*p.e. arg1 et arg2*)
- Fournir des **valeurs par défaut** aux arguments (*p.e. arg2 = TRUE*)
- Définir **ce qu'il faut faire** avec ces arguments (**expression**) :

```
ma_fonction <- function(arg1, arg2, ...) {  
  expression  
}
```

La première chose à noter est que nous avons utilisé la fonction `function()` pour créer une nouvelle fonction appelée `ma_fonction`.

Entre les parenthèses (), on spécifie les informations (*i.e.*, arguments) dont la fonction a besoin pour fonctionner (autant ou aussi peu que nécessaire).

Ces arguments sont ensuite transmis à la partie expression de la fonction. L'expression peut être n'importe quelle commande R valide ou n'importe quel ensemble de commandes R et est généralement entre une paire d'accolades {}.

Une fois que vous avez exécuté le code ci-dessus, vous pouvez utiliser votre nouvelle fonction en tapant :

```
ma_fonction(arg1, arg2)
```

Prenons un exemple pour clarifier les choses.

Tout d'abord, on crée un jeu de données appelé repas où les colonnes lasagnes, stovies, poutine et tartiflette sont remplis avec 10 valeurs aléatoires tirées d'un sac (à l'aide de la fonction rnorm() pour tirer des valeurs aléatoires d'une distribution normale avec une moyenne de 0 et un écart type de 1).

Nous incluons également un “problème”, que nous devrons résoudre plus tard, en incluant 3 NA dans la variable poutine (en utilisant rep(NA, 3)).

```
repas <- data.frame(
  lasagnes = rnorm(10),
  stovies = rnorm(10),
  poutine = c(rep(NA, 3), rnorm(7)),
  tartiflette = rnorm(10)
)
```

Supposons que vous souhaitez multiplier les valeurs des variables stovies et lasagnes pour créer un nouvel objet appelé stovies_lasagnes.

Nous pouvons le faire “à la main” :

```
stovies_lasagnes <- repas$stovies * repas$lasagnes
```

Si c'était tout ce que nous avions à faire, on pourrait s'arrêter là.

R fonctionne avec des vecteurs, de sorte qu'effectuer ce type d'opérations dans R est en fait beaucoup plus simple que dans d'autres langages de programmation, où ce type de code peut nécessiter des boucles

(nous disons que R est un langage *vectorisé*). Une chose à garder à l'esprit pour plus tard est que faire ce genre d'opérations avec des boucles peut être beaucoup plus lent que la vectorisation.

Mais que se passe-t-il si nous voulons répéter cette multiplication plusieurs fois ?

Supposons que nous voulions multiplier les colonnes `lasagnes` et `stovies`, `stovies` et `tartiflette` et `poutine` et `tartiflette`. Dans ce cas, nous pouvons copier et coller le code en remplaçant les informations pertinentes.

```
lasagnes_stovies <- repas$lasagnes * repas$stovies
stovies_tartiflette <- repas$stovies * repas$stovies
poutine_tartiflette <- repas$poutine * repas$tartiflette
```

Bien que cette approche fonctionne, il est facile de faire des **erreurs**.

Et en effet, ici, nous avons “oublié” de modifier `stovies` en `tartiflette` dans la deuxième ligne de code lors du copier-coller. C'est là que l'écriture d'une fonction s'avère utile !

Si nous écrivions cela sous forme de fonction, il n'y aurait qu'une seule source d'erreur potentielle (dans la fonction elle-même) au lieu de nombreuses lignes de code copiées-collées.

Astuce

En règle générale, si nous devons faire la même chose (par copier-coller et modifier) 3 fois ou plus, nous créons une fonction pour le faire.

Dans cet exemple, nous avons utilisé un code assez trivial où il est peut-être difficile de faire une véritable erreur. Mais que se passerait-il si nous augmentions la complexité ?

```
repas$lasagnes * repas$stovies / repas$lasagnes + (repas$lasagnes * 10^(repas$stovies))
- repas$stovies - (repas$lasagnes * sqrt(repas$stovies + 10))
```

Imaginez maintenant que vous deviez copier-coller ce code trois fois, et que vous deviez à chaque fois modifier l'élément `lasagnes` et `stovies` (surtout si nous devions le faire plus de trois fois).

Ce que nous pourrions faire à la place, c'est généraliser notre code pour `x` et `y` au lieu de nommer des plats spécifiques. En procédant de la sorte, nous pourrions recycler le code `x * y`. Chaque fois que nous voulions regrouper plusieurs colonnes, nous assignions un plat à `x` ou `y`.

Nous attribuerons la multiplication aux objets `lasagnes_stovies` et `stovies_poutine` afin de pouvoir y revenir plus tard.

```
# Définir les valeurs x et y
x <- repas$lasagnes
y <- repas$stovies

# Utiliser le code de multiplication
lasagnes_stovies <- x * y

# Définir les nouvelles valeurs x et y
x <- repas$stovies
y <- repas$poutine

# Ré-utiliser le code de multiplication
stovies_poutine <- x * y
```

C'est essentiellement ce que fait une fonction.

Appelons notre nouvelle fonction `col_multiplicateur()` et définissons-la avec deux arguments, `x` et `y`.

Une fonction dans R renvoie simplement sa dernière valeur. Toutefois, il est possible de forcer la fonction à renvoyer une valeur antérieure si cela s'avère nécessaire. Pour ce faire, il suffit d'utiliser la fonction `return()`

Ce n'est pas strictement nécessaire dans cet exemple car R retournera automatiquement la valeur de la dernière ligne de code de notre fonction. Nous l'incluons ici pour l'expliciter.

```
col_multiplicateur <- function(x, y) {
  return(x * y)
}
```

Maintenant que nous avons défini notre fonction, nous pouvons l'utiliser, ou “l'appeler”.

Utilisons la fonction pour multiplier les colonnes `lasagnes` et `stovies` en assignant le résultat à un nouvel objet appelé `lasagna_stovies_func`

```
lasagnes_stovies_fonc <- col_multiplicateur(x = repas$lasagnes, y = repas$stovies)
lasagnes_stovies_fonc
```

```
[1] 1.18774448 0.12282565 0.60907643 -0.57424832 -2.03657230 -0.06215484  
[7] -2.01338744 -0.02556164 -1.35888055 0.17026158
```

Si on ne s'intéresse qu'à la multiplication de `repas$lasagnes` par `repas$stovies` ce serait exagéré de créer une fonction pour faire quelque chose une seule fois.

Cependant, l'avantage de créer une fonction est que nous avons maintenant cette fonction ajoutée à notre environnement et que nous pouvons l'utiliser aussi souvent que souhaité.

Nous disposons également du code pour créer la fonction, ce qui signifie que nous pouvons l'utiliser dans des projets entièrement nouveaux, réduisant ainsi la quantité de code à écrire (et à tester) à chaque fois.

Pour s'assurer que la fonction a fonctionné correctement, nous pouvons comparer la variable `lasagnes_stovies` avec notre nouvelle variable `lasagnes_stovies_fonc` à l'aide de la fonction `identical()`.

La fonction `identical()` teste si deux objets sont *exactement* identiques et renvoie un TRUE ou FALSE.

Tapez `?identical` dans la console pour en savoir plus sur cette fonction.

```
identical(lasagnes_stovies, lasagnes_stovies_fonc)
```

```
[1] TRUE
```

Et nous confirmons que la fonction a produit le même résultat que le calcul manuel. Nous vous recommandons de prendre l'habitude de vérifier que la fonction que vous avez créée fonctionne comme vous le pensez.

Utilisons maintenant notre `col_multiplicateur()` pour multiplier les colonnes `stovies` et `poutine`. Remarquez maintenant que l'argument `x` reçoit la valeur `repas$stovies` et `y` la valeur `repas$poutine`.

```
stovies_poutine_fonc <- col_multiplicateur(x = repas$stovies, y = repas$poutine)  
stovies_poutine_fonc
```

```
[1] NA NA NA 0.35252949 1.00574206 0.16454388  
[7] 1.69943141 0.01156064 0.27726558 0.64071363
```

Jusqu'à présent, tout va bien.

Tout ce que nous avons fait, c'est envelopper le code `x * y` dans une fonction, où nous demandons à l'utilisateur de spécifier à quoi correspondent `x` et `y`.

L'utilisation de la fonction est un peu longue car nous devons retaper le nom du jeu de données pour chaque variable. Pour nous amuser un peu, nous pouvons modifier la fonction afin de spécifier le jeu de données en tant qu'argument et les noms des colonnes sans les mettre entre guillemets (comme dans le style `tidyverse` 📦).

```

1 col_multiplicateur <- function(donnees, x, y) {
2   temp_var <- donnees %>%
3     select({{ x }}, {{ y }}) %>%
4     mutate(xy = prod(.)) %>%
5     pull(xy)
6 }
```

Pour cette nouvelle version de la fonction, nous avons ajouté un paramètre `donnees` à la ligne 1.

À la ligne 3, nous sélectionnons les variables `x` et `y` fournies comme arguments.

À la ligne 4, nous créons le produit des 2 colonnes sélectionnées et

À la ligne 5, nous extrayons la colonne que nous venons de créer.

Nous supprimons également la fonction `return()` puisqu'elle n'était pas nécessaire

Notre fonction est maintenant compatible avec la fonction tuyau, ou “pipe” (soit en natif `|>` ou `magrittr` 📦 `%>%`). Toutefois, étant donné que la fonction utilise désormais le pipe de `magrittr` 📦 et `dplyr` 📦, il faut charger le paquet `tidyverse` 📦 pour qu'elle fonctionne.

```

library(tidyverse)
lasagnes_stovies_fonc <- col_multiplicateur(repas, lasagnes, stovies)
lasagnes_stovies_fonc <- repas |> col_multiplicateur(lasagnes, stovies)
```

Ajoutons maintenant un peu plus de complexité.

Si vous regardez la sortie de `poutine_tartiflette` certains des calculs ont produit des valeurs NA. Cela s'explique par le fait qu'il y a des NA dans `poutine` (incluses lorsque nous avons créé le jeu de données `repas`). Malgré ces NA, la fonction semble avoir fonctionné, mais elle ne nous a donné aucune indication quant à l'existence d'un problème. Dans ce cas, nous préférerions qu'elle nous avertisse que quelque chose ne va pas.

Comment pouvons-nous faire en sorte que la fonction nous informe lorsque des NA sont produites ? Voici une solution.

```

1 col_multiplicateur <- function(donnees, x, y) {
2   temp_var <- donnees %>%
3     select({{ x }}, {{ y }}) %>%
4     mutate(xy = {
5       .[1] * .[2]
6     }) %>%
7     pull(xy)
8   if (any(is.na(temp_var))) {
9     warning("La fonction a produit des NA")
10    return(temp_var)
11  } else {
12    return(temp_var)
13  }
14}

```

```
stovies_poutine_fonc <- col_multiplicateur(repas, stovies, poutine)
```

Warning in col_multiplicateur(repas, stovies, poutine): La fonction a produit des NA

```
lasagnes_stovies_fonc <- col_multiplicateur(repas, lasagnes, stovies)
```

Le cœur de notre fonction reste le même, mais nous avons maintenant six lignes de code supplémentaires (lignes 6 à 11).

Nous avons inclus des *structures conditionnelles*, `if` (lignes 6-8) et `else` (lignes 9-11), afin de tester si des NA ont été produits et, si c'est le cas, nous affichons un message d'avertissement à l'intention de l'utilisateur.

La section suivante de ce chapitre explique le fonctionnement et l'utilisation de ces structures conditionnelles.

5.3. Structures conditionnelles

`x * y` n'applique aucune logique. Il prend simplement la valeur de `x` et la multiplie par la valeur de `y`. Les structures conditionnelles permettent d'injecter de la logique dans votre code.

La structure conditionnelle la plus couramment utilisée est **if**. Chaque fois que vous voyez un **if** lisez-le comme “*Si X est VRAI, fait quelque chose*”.

Inclure un **else** permet simplement d’étendre la logique à “*Si X est VRAI, fait quelque chose, sinon fait autre chose*”.

if et **else** vous permettent d’exécuter des sections de code, en fonction d’une condition qui est soit TRUE ou FALSE.

Le pseudo-code ci-dessous vous montre la forme générale.

```
if (condition) {
    Code executed when condition is TRUE
} else {
    Code executed when condition is FALSE
}
```

Pour approfondir la question, nous pouvons utiliser une vieille blague de programmeur pour poser un problème.

Le partenaire d’un.e. programmeu.r.se dit : “S’il-te-plaît, va au magasin et achète une brique de lait, s’ils ont des œufs, prends-en 6”.

Le.la programmeu.r.se revient avec 6 briques de lait.

Lorsque le partenaire s’en aperçoit, il s’exclame : “Pourquoi diable as-tu acheté 6 briques de lait ?”

Le.la programmeu.r.se répond “*Ils avaient des œufs*”

Au risque d’expliquer une blague, l’énoncé conditionnel ici est de savoir si le magasin avait ou non des œufs. Si le codage est conforme à la demande initiale, le.la programmeu.r.se doit apporter 6 briques de lait si le magasin a des œufs (condition = VRAI), ou apporter 1 brique de lait s’il n’y a pas d’œufs (condition = FAUX).

Dans R, cela est codé comme suit :

```
oeufs <- TRUE # Est-ce qu'il y a des œufs au magasin

if (oeufs == TRUE) { # S'il y a des œufs
  n.lait <- 6 # Prend 6 briques de lait
} else { # S'il n'y a pas d'œufs
```

```
n.lait <- 1 # Prend 1 brique de lait
}
```

Nous pouvons alors vérifier n.lait le nombre de briques de lait que le programmeur se a ramenées.

```
n.lait
```

```
[1] 6
```

Et comme dans la blague, notre code R n'a pas compris que la condition était de déterminer s'il fallait ou non acheter des œufs, et non plus du lait (il s'agit en fait d'un exemple libre du [schéma de Winograd](#) conçu pour tester la condition *d'intelligence* d'une intelligence artificielle en fonction de sa capacité à raisonner sur le sens d'une phrase).

Nous pourrions coder exactement la même structure conditionnelle de blague œuf-lait à l'aide de la fonction `ifelse()`.

```
oeufs <- TRUE
n.lait <- ifelse(oeufs == TRUE, yes = 6, no = 1)
```

`ifelse()` fait exactement la même chose que la version plus étouffée de tout à l'heure, mais elle est maintenant condensée en une seule ligne de code.

Elle présente l'avantage supplémentaire de travailler sur des vecteurs plutôt que sur des valeurs individuelles (nous y reviendrons plus tard lorsque nous introduirons les **boucles**). La logique est lue de la même manière : “S'il y a des œufs, assignez une valeur de 6 à n.lait s'il n'y a pas d'œufs, assignez la valeur 1 à n.lait”.

Nous pouvons vérifier à nouveau que la logique renvoie toujours 6 briques de lait :

```
n.lait
```

```
[1] 6
```

Actuellement, il faudrait copier-coller du code pour changer la présence ou l'absence d'œufs dans le magasin. Nous avons appris plus haut comment éviter de nombreux copier-coller en créant une fonction. Comme avec la simple fonction `x * y` de notre précédente fonction `col_multiplicateur()` les déclarations logiques ci-dessus sont simples à coder et se prêtent bien à la transformation en fonction.

Et si nous faisions justement cela et enveloppons cette déclaration logique dans une fonction ?

```

lait <- function(oeufs) {
  if (oeufs == TRUE) {
    6
  } else {
    1
  }
}

```

Nous avons créé une fonction appelée `lait()` dont le seul argument est `oeufs`. L'utilisateur de la fonction spécifie si les œufs sont soit `TRUE` ou `FALSE` et la fonction utilisera alors une structure conditionnelle pour déterminer le nombre de cartons de lait renvoyés.

Essayons rapidement :

```
lait(oeufs = TRUE)
```

```
[1] 6
```

Et la plaisanterie est maintenue.

Remarquez que, dans ce cas, nous avons spécifié que nous remplissons l'argument `oeufs` (`oeufs = TRUE`). Dans certaines fonctions, comme la nôtre ici, lorsqu'une fonction n'a qu'un seul argument, nous pouvons être paresseux et ne pas nommer l'argument que nous remplissons. En réalité, on considère généralement qu'il est préférable d'indiquer explicitement les arguments que l'on remplit afin d'éviter les erreurs potentielles.

OK, revenons à la fonction `col_multiplicateur()` que nous avons créée ci-dessus et expliquons comment nous avons utilisé des structures conditionnelles pour avertir l'utilisateur si des NA sont produites lorsque nous multiplions deux colonnes.

```

col_multiplicateur <- function(donnees, x, y) {
  temp_var <- donnees %>%
    select({{ x }}, {{ y }}) %>%
    mutate(xy = {
      .[1] * .[2]
    }) %>%
    pull(xy)
}

```

```

if (any(is.na(temp_var))) {
  warning("La fonction a produit des NA")
  return(temp_var)
} else {
  return(temp_var)
}
}

```

Dans cette nouvelle version de la fonction, on utilise toujours `x * y`, mais cette fois nous avons assigné les valeurs de ce calcul à un vecteur temporaire appelé `temp_var` afin de pouvoir l'utiliser dans nos structures conditionnelles.

Notez que ce `temp_var` est *locale* à notre fonction et n'existera pas en dehors de la fonction en raison de ce que l'on appelle les [règles de cadrage de R](#).

Nous utilisons ensuite un `if` pour déterminer si notre `temp_var` contient des NA valeurs. Pour ce faire, nous utilisons la fonction `is.na()` pour vérifier si chaque valeur de notre `temp_var` est un NA.

`is.na()` renvoie TRUE si la valeur est un NA et FALSE si la valeur n'est pas un NA.

Nous imbriquons ensuite le `is.na(temp_var)` à l'intérieur de la fonction `any()` pour vérifier si **au moins une** des valeurs retournées par `is.na(temp_var)` est TRUE. Si c'est le cas, `any()` renverra une valeur TRUE.

Ainsi, s'il existe des NA dans notre `temp_var` la condition pour le `if()` sera TRUE alors que s'il n'y a pas de NA, la condition sera FALSE.

Si la condition est TRUE la fonction `warning()` génère un message d'avertissement à l'intention de l'utilisateur et renvoie la valeur de la variable `temp_var`.

Si la condition est FALSE le code sous la condition `else` est exécuté, ce qui renvoie simplement la valeur `temp_var`

Ainsi, si nous exécutons notre `col_multiplicateur()` sur les colonnes `repas$stovies` et `repas$poutine` (qui contient NAs), nous recevrons un message d'avertissement.

```
stovies_poutine_fonc <- col_multiplicateur(repas, stovies, poutine)
```

```
Warning in col_multiplicateur(repas, stovies, poutine): La fonction a produit
des NA
```

En revanche, si nous multiplions deux colonnes qui ne contiennent pas de NA nous ne recevons pas de message d'avertissement

```
lasagnes_stovies_fonc <- col_multiplicateur(repas, lasagnes, stovies)
```

5.4. Combinaison d'opérateurs logiques

Les fonctions que nous avons créées jusqu'à présent sont parfaitement adaptées à nos besoins, bien qu'elles soient assez simplistes. Essayons de créer une fonction un peu plus complexe.

Nous allons créer une fonction permettant de déterminer si la journée d'aujourd'hui sera bonne ou non en fonction de deux critères : le *jour de la semaine* (vendredi ou non) et *est-ce que votre code fonctionne ou non* (VRAI ou FAUX).

Pour ce faire, nous utiliserons les structures conditionnelles `if` et `else`. La complexité vient ici des `if` qui suivent immédiatement les `else`. Nous utiliserons ces instructions conditionnelles quatre fois pour obtenir toutes les combinaisons possibles, qu'il s'agisse d'un vendredi ou non, et que votre code fonctionne ou non.

Nous utilisons également la fonction `cat()` pour produire un texte formaté correctement.

```
bonne.journee <- function(code.fonctionne, jours) {
  if (code.fonctionne == TRUE && jours == "Vendredi") {
    cat(
      "MEILLEURE.
      JOURNÉE.
      DE TOUS LES TEMPS.

      Arrête-toi là tant que ça dure et va au bar !"
    )
  } else if (code.fonctionne == FALSE && jours == "Vendredi") {
    cat("Bon... Au moins c'est vendredi ! Apéro !")
  } else if (code.fonctionne == TRUE && jours != "Vendredi") {
    cat(
      "On était si proche d'une bonne journée...
      Mais c'est pas vendredi"
    )
  }
}
```

```

} else if (code.fonctionne == FALSE && jours != "Vendredi") {
    cat("Le monde est contre moi.")
}
}

```

```
bonne.journee(code.fonctionne = TRUE, jours = "Vendredi")
```

MEILLEURE.

JOURNÉE.

DE TOUS LES TEMPS.

Arrête-toi là tant que ça dure et va au bar !

```
bonne.journee(FALSE, "Tuesday")
```

Le monde est contre moi.

Vous avez remarqué que nous n'avons jamais spécifié ce qu'il fallait faire si le jour n'était pas un "Vendredi" ? C'est parce que, pour cette fonction, la seule chose qui compte est de savoir si c'est un vendredi (==) ou non (!=).

Nous avons également utilisé des opérateurs logiques chaque fois que nous avons utilisé la structure `if`. Les opérateurs logiques sont la dernière pièce du puzzle des conditions logiques. Ils sont résumés dans le tableau ci-dessous. Les deux premiers sont des opérateurs logiques et les six derniers sont des opérateurs relationnels. Vous pouvez utiliser n'importe lequel de ces opérateurs lorsque vous créez vos propres fonctions (ou boucles).

Opérateur	Description technique	Ce que cela signifie	Exemple d'application
<code>&&</code>	ET logique	Les deux conditions doivent être remplies	<code>if(cond1 == test && cond2 == test</code>
<code> </code>	OU logique	L'une ou l'autre des conditions doit être remplies	<code>if(cond1 == test cond2 == test</code>
<code><</code>	Inférieur à	X est inférieur à Y	<code>if(X < Y)</code>
<code>></code>	Supérieur à	X est supérieur à Y	<code>if(X > Y)</code>
<code><=</code>	Inférieur ou égal à	X est inférieur/égal à Y	<code>if(X <= Y)</code>
<code>>=</code>	Supérieur ou égal à	X est supérieur/égal à Y	<code>if(X >= Y)</code>
<code>==</code>	Egal à	X est égal à Y	<code>if(X == Y)</code>

Opérateur	Description technique	Ce que cela signifie	Exemple d'application
<code>!=</code>	N'est pas égal à	X n'est pas égal à Y	<code>if(X != Y)</code>

5.5. Boucles

R est très performant dans l'exécution de tâches répétitives. Si nous voulons qu'un ensemble d'opérations soit répété plusieurs fois, nous utilisons ce que l'on appelle une **boucle**. Lorsque vous créez une boucle, R exécute les instructions qu'elle contient un certain nombre de fois ou jusqu'à ce qu'une condition donnée soit remplie. Il existe trois principaux types de boucles dans R : la boucle **for** ("pour") la boucle **while** ("tant que") et la boucle **repeat** ("répéter").

Les boucles sont l'un des éléments de base de tous les langages de programmation (pas seulement R), et peuvent être un outil puissant (bien qu'à notre avis, elles soient utilisées beaucoup trop souvent lors de l'écriture de code R).

5.5.1. Boucle "For" (pour)

La structure de boucle la plus couramment utilisée lorsque vous souhaitez répéter une tâche un nombre défini de fois est la boucle **for**.

L'exemple le plus simple de boucle **for** est le suivant :

```
for (i in 1:5) {
  print(i)
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

Mais que fait réellement le code ? Il s'agit d'un morceau de code dynamique où un index **i** est remplacé itérativement par chaque valeur du vecteur **1:5**.

Décomposons.

Parce que la première valeur de notre séquence (1:5) est 1 la boucle commence par remplacer `i` par 1 et exécute tout ce qui se trouve entre les accolades {}.

Les boucles utilisent conventionnellement `i` comme compteur (“`i`” pour “itération”), mais vous êtes libre d’utiliser ce que vous voulez, même le nom de votre animal de compagnie, cela n’a pas vraiment d’importance (sauf lorsque vous utilisez des boucles imbriquées, auquel cas les compteurs doivent être appelés différemment, comme `SenorWhiskers` et `HerrFlufferkins`).

Ainsi, si nous devions effectuer manuellement la première itération de la boucle :

```
i <- 1  
print(i)
```

[1] 1

Une fois cette première itération terminée, la *boucle for* revient au début et remplace `i` par la valeur suivante dans notre séquence 1:5 (2 dans ce cas) :

```
i <- 2  
print(i)
```

[1] 2

Ce processus est ensuite répété jusqu’à ce que la boucle atteigne la dernière valeur de la séquence (5 dans cet exemple), après quoi elle s’arrête.

Pour mieux comprendre le fonctionnement de ces boucles `for` et vous présenter une caractéristique importante des boucles en général, nous allons modifier notre compteur à l’intérieur de la boucle. Cela peut être utilisé, par exemple, pour parcourir un vecteur, mais en sélectionnant la ligne suivante (ou toute autre valeur).

Pour ce faire, nous ajouterons simplement 1 à la valeur de notre index à chaque fois que nous itérons notre boucle.

```
for (i in 1:5) {  
  print(i + 1)  
}
```

```
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
```

Comme dans la boucle précédente, la première valeur de notre séquence est 1. La boucle commence par remplacer `i` par 1 mais cette fois, nous + 1 à chaque valeur de `i` dans l'expression. Le résultat est donc `1 + 1`.

```
i <- 1
i + 1
```

```
[1] 2
```

Comme précédemment, une fois l'itération terminée, la boucle passe à la valeur suivante de la séquence et remplace `i` par la valeur suivante (2 dans ce cas), de sorte que `i + 1` devient `2 + 1`.

```
i <- 2
i + 1
```

```
[1] 3
```

Et ainsi de suite. Nous pensons que vous avez l'idée ! Dans les faits, c'est tout ce que fait une boucle `for`, rien d'autre !

Bien que nous ayons utilisé une simple addition dans le corps de la boucle, vous pouvez également combiner des boucles avec des fonctions.

Revenons à notre jeu de données `repas`. Précédemment dans le chapitre, nous avons créé une fonction pour multiplier deux colonnes et l'avons utilisée pour créer les variables `lasagnes_stovies`, `stovies_poutine`, et `poutine_tartiflette`. Nous aurions pu utiliser une boucle pour cela !

Rappelons-nous à quoi ressemblent nos données et le code de la fonction `col_multiplicateur()`.

```
repas <- data.frame(
  lasagnes = rnorm(10),
  stovies = rnorm(10),
  poutine = c(rep(NA, 3), rnorm(7)),
  tartiflette = rnorm(10)
)
```

```
col_multiplicateur <- function(donnees, x, y) {
  temp_var <- donnees %>%
    select({{ x }}, {{ y }}) %>%
    mutate(xy = {
      .[1] * .[2]
    }) %>%
    pull(xy)
  if (any(is.na(temp_var))) {
    warning("La fonction a produit des NA")
    return(temp_var)
  } else {
    return(temp_var)
  }
}
```

Nous allons d'abord créer une liste vide (vous vous souvenez de Section 3.2.3 ?) que nous appelons `temp` (pour temporaire) qui sera utilisée pour stocker les résultats des itérations de la fonction via la boucle `for`.

```
temp <- list()
for (i in 1:(ncol(repas) - 1)) {
  temp[[i]] <- col_multiplicateur(repas, x = colnames(repas)[i], y = colnames(repas)[i + 1])
}
```

```
Warning in col_multiplicateur(repas, x = colnames(repas)[i], y =
  colnames(repas)[i + : La fonction a produit des NA
Warning in col_multiplicateur(repas, x = colnames(repas)[i], y =
  colnames(repas)[i + : La fonction a produit des NA
```

Lorsque nous spécifions notre boucle `for` remarquez que nous avons soustrait 1 à `ncol(repas)`. La boucle `ncol()` renvoie le nombre de colonnes dans notre jeu de données `repas`, donc 4. Ainsi, notre boucle s'exécute de `i = 1` à `i = 4 - 1` c'est-à-dire, `i = 3`.

Ainsi, lors de la première itération de la boucle, `i` prend la valeur `1`. `col_multiplicateur()` multiplie `repas[, 1]` (`lasagnes`) par `repas[, 1 + 1]` (`stovies`) et le stocke en tant que `temp[[1]]`, donc le premier élément de la liste `temp`.

À la deuxième itération de la boucle, `i` prend la valeur `2`. `col_multiplicateur()` multiplie `repas[, 2]` (`stovies`) pas `repas[, 2 + 1]` (`poutine`) et le stocke en tant que `temp[[2]]`, donc le deuxième élément de la liste `temp`.

À la troisième et dernière itération de la boucle, `i` prend la valeur `3`. `col_multiplicateur()` multiplie `repas[, 3]` (`poutine`) par `repas[, 3 + 1]` (`tartiflette`) et le stocke en tant que `temp[[3]]`, donc le troisième élément de la liste `temp`.

Encore une fois, il est bon de vérifier que nous obtenons quelque chose de sensé de notre boucle (rappelez-vous, vérifiez, vérifiez et **vérifiez encore !**).

Pour ce faire, nous pouvons utiliser la fonction `identical()` pour comparer les variables que nous avons créées “`by hand`” avec chaque itération de la boucle manuellement.

```
lasagnes_stovies_fonc <- col_multiplicateur(repas, lasagnes, stovies)
i <- 1
identical(
  col_multiplicateur(repas, colnames(repas)[i], colnames(repas)[i + 1]),
  lasagnes_stovies_fonc
)
```

[1] TRUE

```
stovies_poutine_fonc <- col_multiplicateur(repas, stovies, poutine)
```

Warning in `col_multiplicateur(repas, stovies, poutine)`: La fonction a produit des NA

```
i <- 2
identical(
  col_multiplicateur(repas, colnames(repas)[i], colnames(repas)[i + 1]),
  stovies_poutine_fonc
)
```

Warning in col_multiplicateur(repas, colnames(repas)[i], colnames(repas)[i + 1]) :
La fonction a produit des NA

[1] TRUE

Si vous arrivez à suivre les exemples ci-dessus, vous êtes prêts pour commencer à écrire vos propres boucles `for`.

Mais, il existe d'autres types de boucles.

5.5.2. Boucle “While” (tant que)

La boucle `while` est utilisée lorsque vous voulez faire tourner en boucle jusqu'à ce qu'une certaine condition logique spécifique soit remplie (contrairement à la boucle `for` qui parcourt toujours une séquence entière).

La structure de base d'une boucle `while` est la suivante :

```
while (logical_condition) {
  expression
}
```

Un exemple simple de boucle `while` est :

```
i <- 0
while (i <= 4) {
  i <- i + 1
  print(i)
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

Ici, la boucle continuera seulement à transmettre des valeurs au corps principal de la boucle (l'*expression*) que lorsque *i* est inférieur ou égal à 4 (spécifié à l'aide de l'attribut `<=` dans cet exemple). Une fois que *i* est supérieur à 4, la boucle s'arrête.

Il existe un autre type de boucle, très rarement utilisé : la boucle `repeat`. La boucle `repeat` n'a pas de contrôle conditionnel et peut donc continuer à itérer indéfiniment. Ce qui signifie qu'une pause ("break"), ou "stop here", doit être codée dans la boucle. C'est intéressant de savoir que ça existe, mais pour l'instant ce n'est pas très utile de s'en préoccuper ; les boucles `for` et `while` devraient vous permettre de répondre à la plupart de vos besoins.

5.5.3. Quand utiliser une boucle ?

Les boucles sont assez couramment utilisées, bien que parfois un peu trop à notre avis. Des tâches équivalentes peuvent être effectuées avec des fonctions, qui sont souvent plus efficaces.

La question se pose donc de savoir quand faut-il utiliser une boucle ?

En général, les boucles sont implémentées de manière inefficace dans R et doivent être évitées lorsque de meilleures alternatives existent, en particulier lorsque vous travaillez avec de grands ensembles de données. Cependant, les boucles sont parfois le seul moyen d'obtenir le résultat souhaité.

Voici quelques exemples de cas où l'utilisation de boucles peut s'avérer appropriée :

- Certaines simulations¹
- Relations récursives²
- Problèmes plus complexes³
- Boucles While⁴

¹Par exemple le modèle de Ricker peut, en partie, être construit à l'aide de boucles.

²Une relation qui dépend de la valeur de la relation précédente - "pour comprendre la récursivité, il faut comprendre la récursivité".

³Par exemple, depuis combien de temps le dernier blaireau a-t-il été vu sur le site *j*, sachant qu'une martre des pins a été vue à l'heure *t* au même endroit *j* que le blaireau, lorsque la martre a été détectée au cours d'une période spécifique de 6 heures, mais excluant les blaireaux vus 30 minutes avant l'arrivée de la martre, répétée pour toutes les detections de martres.

⁴Continuez à sauter jusqu'à ce que vous ayez atteint la lune.

5.5.4. Si on n'utilise pas une boucle, alors quoi ?

En bref, utilisez la famille de fonctions **apply** ; `apply()`, `lapply()`, `tapply()`, `sapply()`, `vapply()` et `mapply()`.

Les fonctions `apply` peuvent souvent accomplir les tâches de la plupart des boucles “maison”, parfois plus rapidement (bien que cela ne soit pas vraiment un problème pour la plupart des gens), mais surtout avec un risque d’erreur beaucoup plus faible.

Une stratégie à garder à l’esprit et qui peut s’avérer utile est la suivante : pour chaque boucle que vous faites, essayez de la refaire en utilisant une fonction `apply` (souvent `lapply` ou `sapply` fonctionneront). Si vous le pouvez, utilisez la version applicable.

Il n’y a rien de pire que de se rendre compte qu’il y avait une petite, minuscule, erreur apparemment insignifiante dans une boucle qui, des semaines, des mois ou des années plus tard, s’est transformée en un énorme bazar.

Nous recommandons vivement d’essayer d’utiliser les fonctions `apply` chaque fois que cela est possible.

lapply

La fonction de base sera souvent `lapply()`, du moins au début.

La façon dont les `lapply()` fonctionnent, et la raison pour laquelle elles constituent souvent une bonne alternative aux boucles `for`, est qu’elles passent en revue chaque élément d’une liste et effectuent une tâche (*c’est-à-dire* exécutent une fonction).

Elles présentent l’avantage supplémentaire de produire les résultats sous forme de **liste**, ce que vous devriez autrement coder vous-même dans une boucle.

Une fonction `lapply()` a la structure suivante :

```
lapply(X, FUN)
```

Ici `X` est le vecteur auquel nous voulons faire *quelque chose*. On écrit `FUN` pour décrire ce qualifié ce qu’on est en train de faire là (je plaisante !). C’est aussi l’abréviation de “fonction” (fonction).

Commençons par une démonstration simple : Utilisons la fonction `lapply()` pour créer une séquence de 1 à 5 et ajouter 1 à chaque observation (comme nous l’avons fait avec une boucle `for`) :

```
lapply(0:4, function(a) {
  a + 1
})
```

[[1]]

[1] 1

[[2]]

[1] 2

[[3]]

[1] 3

[[4]]

[1] 4

[[5]]

[1] 5

Remarquez que nous devons spécifier notre séquence en tant que `0:4` pour obtenir la sortie `1 ,2 ,3 ,4 , 5` puisque nous ajoutons 1 à chaque élément de la séquence. *Voyez ce qui se passe si vous utilisez `1:5` à la place.*

De manière équivalente, nous aurions pu définir la fonction d'abord, puis l'utiliser dans `lapply()`

```
fun_ajouter <- function(a) {
  a + 1
}
lapply(0:4, fun_ajouter)
```

[[1]]

[1] 1

[[2]]

[1] 2

```
[[3]]
```

```
[1] 3
```

```
[[4]]
```

```
[1] 4
```

```
[[5]]
```

```
[1] 5
```

Les `sapply()` fait la même chose que `lapply()` mais au lieu de stocker les résultats sous forme de liste, elle les stocke sous forme de **vecteur**.

```
sapply(0:4, function(a) {  
  a + 1  
})
```

```
[1] 1 2 3 4 5
```

Comme vous pouvez le voir, dans les deux cas, nous obtenons exactement les mêmes résultats que lorsque nous avons utilisé la boucle `for`.

Chapitre 6

Rapports reproductibles avec Quarto

Avertissement

les captures d'écran sont encore avec R markdown sera bientôt mis à jour

Ce chapitre vous présente la création de rapports reproductibles en utilisant R markdown / Quarto afin d'encourager les bonnes (ou meilleures) pratiques pour faciliter la science ouverte. Il décrira d'abord ce que sont R markdown et Quarto et pourquoi vous pourriez envisager de les utiliser, puis comment créer un document Quarto à l'aide de RStudio et enfin comment convertir ce document en un rapport au format html ou pdf. Au cours de ce chapitre, vous apprendrez :

- les différents composants d'un document Quarto
- comment formater le texte, les graphiques et les tableaux dans le document
- comment éviter certaines des difficultés les plus courantes lors de l'utilisation de Quarto.

6.1. Qu'est-ce que R markdown / Quarto ?

6.1.1. R Markdown

R markdown est un langage de texte simple et facile à utiliser pour combiner votre code R, les résultats de votre analyse de données (y compris les graphiques et les tableaux) et vos commentaires écrits dans un document unique, bien formaté et reproductible (comme un rapport, une publication, un chapitre de thèse ou une page web comme celle-ci).

Techniquement, R markdown est une combinaison de trois langages, R, Markdown et YAML (un autre langage de balisage). Markdown et YAML sont tous deux un type de langage de balisage. Un langage de balisage permet

simplement de créer un fichier de texte brut facile à lire, qui peut contenir du texte formaté, des images, des en-têtes et des liens vers d'autres documents. Si vous êtes intéressé, vous pouvez trouver plus d'informations sur les langages de balisage [ici][balisage] . En fait, vous êtes exposé à un langage de balisage tous les jours, car la plupart des contenus Internet que vous digérez quotidiennement sont étayés par un langage de balisage appelé HTML (Hypertext Markup Language). Quoi qu'il en soit, le point principal est que R markdown est très facile à apprendre (beaucoup, beaucoup plus facile que HTML) et lorsqu'il est utilisé avec un bon IDE (RStudio ou VS Code), il est ridiculement facile à intégrer dans votre flux de travail pour produire un contenu riche en fonctionnalités (alors pourquoi ne le feriez-vous pas ?!).

6.1.2. Quarto ?

Quarto est une version multilingue de la nouvelle génération de R Markdown de Posit, avec de nombreuses nouvelles fonctionnalités et capacités, compatible non seulement avec R mais aussi avec d'autres langages comme Python et Julia. Comme R Markdown, Quarto utilise knitr  package pour exécuter le code R, et est donc capable de rendre la plupart des codes R existants. **.Rmd** existants sans modification. Cependant, il s'accompagne également d'une pléthore de nouvelles fonctionnalités. Plus important encore, il facilite grandement la création de différents types de sorties puisque le codage est homogénéisé pour un format spécifique sans avoir à se fier à différents paquets r ayant chacun leurs propres spécificités (*Par exemple* bookdown, hugodown, blogdown, thesisdown, rticles, xaringan, ...).

Dans la suite de ce chapitre, nous parlerons de Quarto mais beaucoup de choses peuvent être faites avec R markdown. Quarto utilise .qmd alors que R markdown fonctionne avec des fichiers .Rmd mais Quarto peut rendre .Rmd également.

6.2. Pourquoi utiliser Quarto ?

Au cours des chapitres précédents, nous avons beaucoup parlé de la nécessité de mener vos recherches de manière robuste et reproductible afin de faciliter la science ouverte. En résumé, la science ouverte consiste à faire tout ce qui est en notre pouvoir pour rendre nos données, nos méthodes, nos résultats et nos conclusions transparents et accessibles à tous. Certains des principaux principes de la science ouverte sont décrits [ici](#) et comprennent

- Transparence de la méthodologie expérimentale, de l'observation, de la collecte des données et des méthodes d'analyse.
- Disponibilité publique et réutilisation des données scientifiques

- Accessibilité au public et transparence de la communication scientifique
- Utilisation d'outils en ligne pour faciliter la collaboration scientifique

À l'heure actuelle, vous utilisez tous (avec un peu de chance) R pour explorer et analyser vos données intéressantes. En tant que tel, vous êtes déjà bien avancé dans votre démarche visant à rendre votre analyse plus reproductible, plus transparente et plus facile à partager. Cependant, il se peut que votre flux de travail actuel ressemble à ceci :

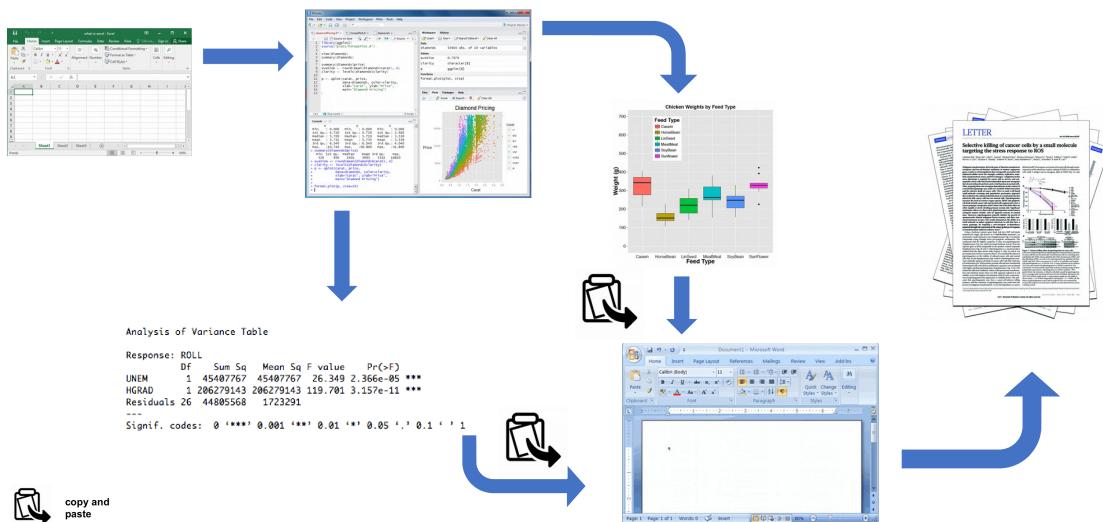


Figure 6.1.: Non-reproducible workflow

Vos données sont importées dans R à partir de votre tableur préféré, vous écrivez votre code R pour explorer et analyser vos données, vous enregistrez les tracés sous forme de fichiers externes, vous copiez les tableaux des résultats d'analyse, puis vous combinez manuellement tous ces éléments et votre prose écrite dans un seul document MS Word (peut-être pour un article ou un chapitre de thèse). Bien qu'il n'y ait rien de particulièrement mauvais dans cette approche (et qu'elle soit certainement meilleure que l'utilisation d'un logiciel “pointer-cliquer” pour analyser vos données), elle présente certaines limites :

- Elle n'est pas particulièrement reproductible. Parce que ce flux de travail sépare votre code R du document final, il y a de multiples occasions de prendre des décisions non documentées (quels graphiques avez-vous utilisés ? quelles analyses avez-vous incluses ou non ? etc.)
- Il est inefficace. Si vous devez revenir en arrière et modifier quelque chose (créer un nouveau graphique ou mettre à jour votre analyse, etc.), vous devrez créer ou modifier plusieurs documents, ce qui augmente le risque que des erreurs se glissent dans votre flux de travail.

- Il est difficile à maintenir. Si votre analyse change, vous devrez à nouveau mettre à jour plusieurs fichiers et documents.
- Il peut être difficile de décider ce qui doit être partagé avec d'autres. Partagez-vous l'ensemble de votre code (exploration initiale des données, validation du modèle, etc.) ou seulement le code spécifique à votre document final ? Il est assez courant (et mauvais !) pour les chercheurs de maintenir deux scripts R, l'un utilisé pour l'analyse proprement dite et l'autre à partager avec le document final ou le chapitre de la thèse. Cela peut prendre du temps et prêter à confusion et devrait être évité.

Un flux de travail plus efficace et plus robuste pourrait ressembler à ceci :

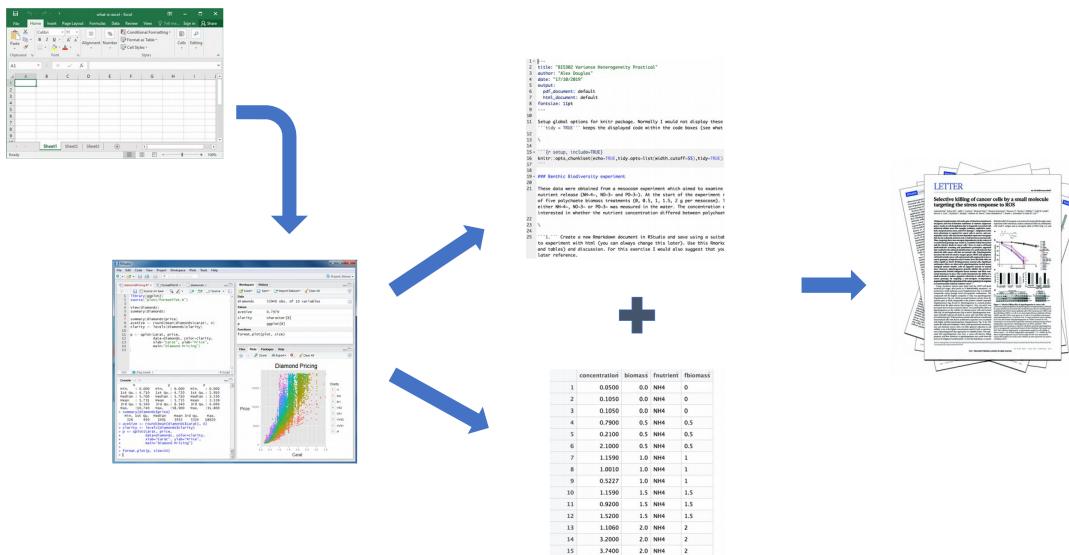


Figure 6.2.: A-reproducible (and more efficient) workflow

Vos données sont importées dans R comme précédemment, mais cette fois-ci, tout le code R que vous avez utilisé pour analyser vos données, produire vos graphiques et votre texte écrit (Introduction, Matériel et Méthodes, Discussion, etc.) est contenu dans un seul document Quarto qui est ensuite utilisé (avec vos données) pour créer automatiquement votre document final. C'est exactement ce que Quarto vous permet de faire.

Voici quelques-uns des avantages de l'utilisation de Quarto :

- Il relie explicitement vos données à votre code R et à vos résultats, créant ainsi un flux de travail entièrement reproductible. **TOUT** du code R utilisé pour explorer, résumer et analyser vos données peut être inclus dans un seul document facile à lire. Vous pouvez décider de ce que vous voulez inclure dans votre document final (comme vous l'apprenez ci-dessous), mais tout votre code R peut être inclus dans le document Quarto.

- Vous pouvez créer une grande variété de formats de sortie (pdf, pages web html, MS Word et bien d’autres) à partir d’un seul document Quarto, ce qui améliore à la fois la collaboration et la communication.
- Améliore la transparence de votre recherche. Vos données et votre fichier Quarto peuvent être joints à votre publication ou au chapitre de votre thèse en tant que matériel supplémentaire ou être hébergés sur un dépôt GitHub (voir Chapitre 7).
- Augmente l’efficacité de votre flux de travail. Si vous avez besoin de modifier ou d’étendre votre analyse actuelle, il vous suffit de mettre à jour votre document Quarto et ces changements seront automatiquement inclus dans votre document final.

6.3. Commencer avec Quarto

Quarto s’intègre très bien avec [R Studio](#) et [VS Code](#) et fournissent à la fois un éditeur de source et un éditeur visuel offrant une expérience proche de votre logiciel d’écriture classique WYSIWYG (ce que vous voyez est ce que vous écrivez) (par exemple Microsoft Word ou LibreOffice writer).

6.3.1. Installation de la solution

Pour utiliser Quarto, vous devez d’abord installer le logiciel Quarto et le logiciel quarto  (avec ses dépendances). Vous trouverez des instructions sur la façon de procéder dans Section 1.1.1 et sur le site web de [Quarto](#). Si vous souhaitez créer des documents PDF (ou des documents MS Word) à partir de votre fichier Quarto, vous devrez également installer une version de {{< latex >}} sur votre ordinateur. Si vous n’avez pas installé {{< latex >}}, nous vous recommandons d’installer [TinyTeX](#). Là encore, des instructions sur la manière de procéder sont disponibles à l’adresse suivante : Section 1.1.1.

6.3.2. Créez un document Quarto, .qmd

Il est temps de créer votre premier document Quarto. Dans RStudio, cliquez sur le menu File -> New File -> Quarto.... Dans la fenêtre qui s’ouvre, donnez un “titre” au document, saisissez les informations relatives à l’ “auteur” (votre nom) et sélectionnez HTML comme format de sortie par défaut. Nous pourrons modifier tout cela ultérieurement, ne vous en préoccupez donc pas pour l’instant.

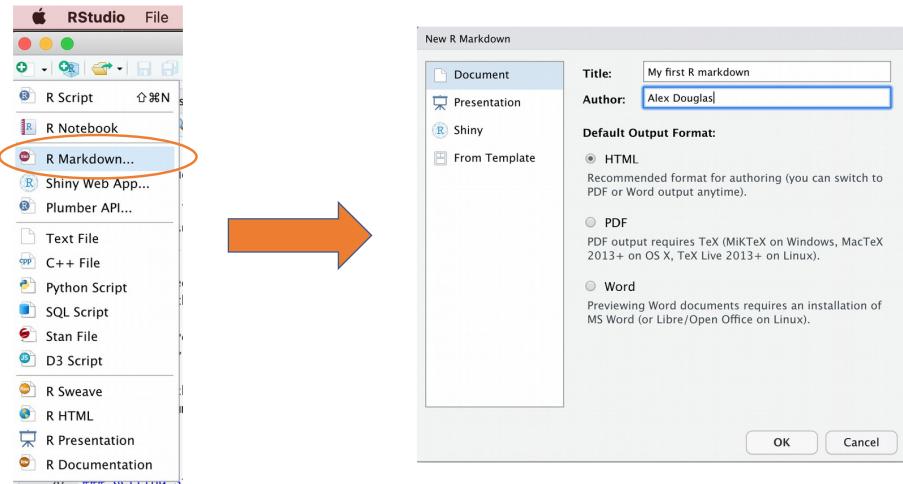


Figure 6.3.: Creating a Quarto document

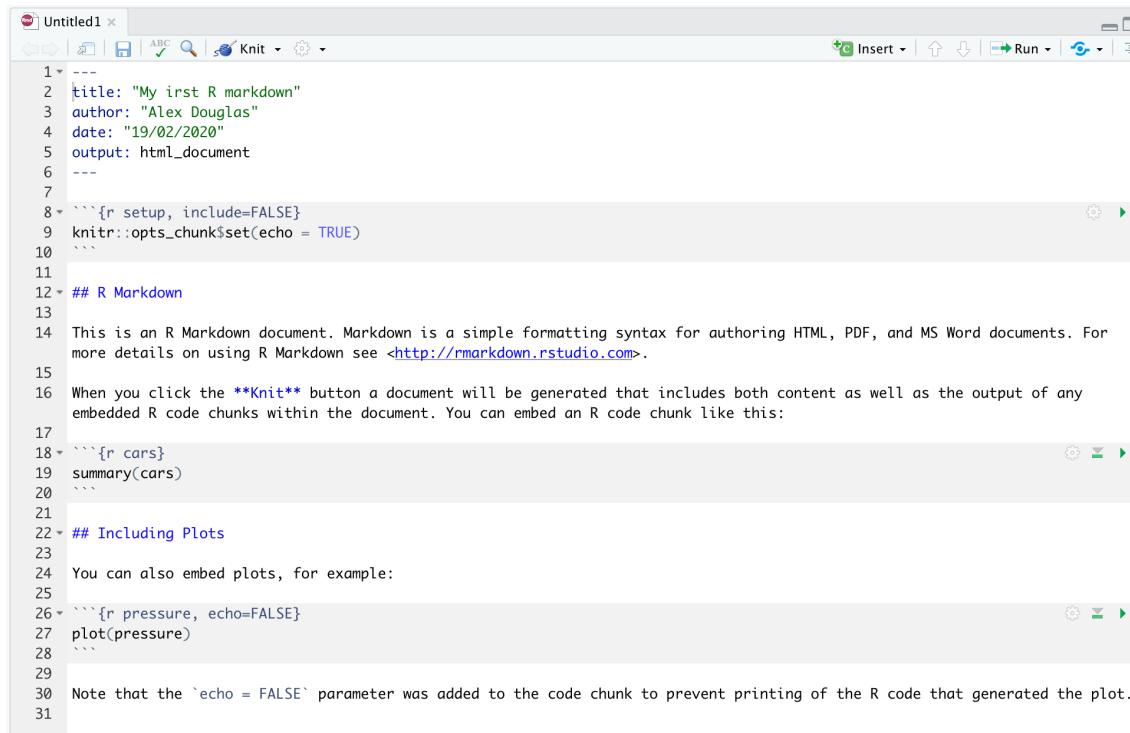
Vous remarquerez que lorsque votre nouveau document Quarto est créé, il comprend un exemple de code Quarto. Normalement, vous devriez surligner et supprimer tout ce qui se trouve dans le document, à l’exception de l’information située en haut, entre les boutons --- (c’est ce qu’on appelle l’en-tête YAML dont nous parlerons dans un instant) et commencer à écrire votre propre code. Cependant, pour l’instant, nous allons utiliser ce document pour nous entraîner à convertir Quarto aux formats html et pdf et vérifier que tout fonctionne.

Une fois que vous avez créé votre document Quarto, c’est une bonne pratique de sauvegarder ce fichier dans un endroit pratique (Section 1.4 et Figure 1.11). Vous pouvez le faire en sélectionnant *File -> Save* dans le menu de RStudio (ou utilisez le raccourci clavier *ctrl + s* sous Windows ou *cmd + s* sur Mac) et entrez un nom de fichier approprié (appelez-le par exemple *my_first_quarto*). Notez que l’extension de votre nouveau fichier Quarto est *.qmd*.

Maintenant, pour convertir votre fichier *.qmd* en document HTML, cliquez sur le petit triangle noir à côté de l’icône *Knit* en haut de la fenêtre source et sélectionnez *knit to HTML*

RStudio va maintenant “tricoter” (ou rendre) votre fichier *.qmd* en un fichier HTML. Remarquez qu’il y a un nouveau Quarto dans votre fenêtre de console, qui vous fournit des informations sur le processus de rendu et affiche également les erreurs si quelque chose ne va pas.

Si tout s’est déroulé sans problème, un nouveau fichier HTML a été créé et enregistré dans le même répertoire que votre fichier *.qmd* (le nôtre s’appellera *my_first_quarto.html*). Pour visualiser ce document, il suffit de double-cliquer sur le fichier pour l’ouvrir dans un navigateur (comme Chrome ou Firefox) et afficher le contenu



A screenshot of a Quarto document titled "Untitled1". The code editor shows R Markdown syntax. The first few lines define the document's metadata: title, author, date, and output type. It then includes an R code chunk to set the echo parameter to TRUE. The document continues with sections on R Markdown basics, including plots, and ends with a note about the echo parameter. The interface includes standard file operations (New, Open, Save) and a "Knit" button.

```

1 ---  

2 title: "My first R markdown"  

3 author: "Alex Douglas"  

4 date: "19/02/2020"  

5 output: html_document  

6 ---  

7  

8 ````{r setup, include=FALSE}  

9 knitr::opts_chunk$set(echo = TRUE)  

10 ````  

11  

12 ## R Markdown  

13  

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

15  

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

17  

18 ````{r cars}  

19 summary(cars)  

20 ````  

21  

22 ## Including Plots  

23  

24 You can also embed plots, for example:  

25  

26 ````{r pressure, echo=FALSE}  

27 plot(pressure)  

28 ````  

29  

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

31

```

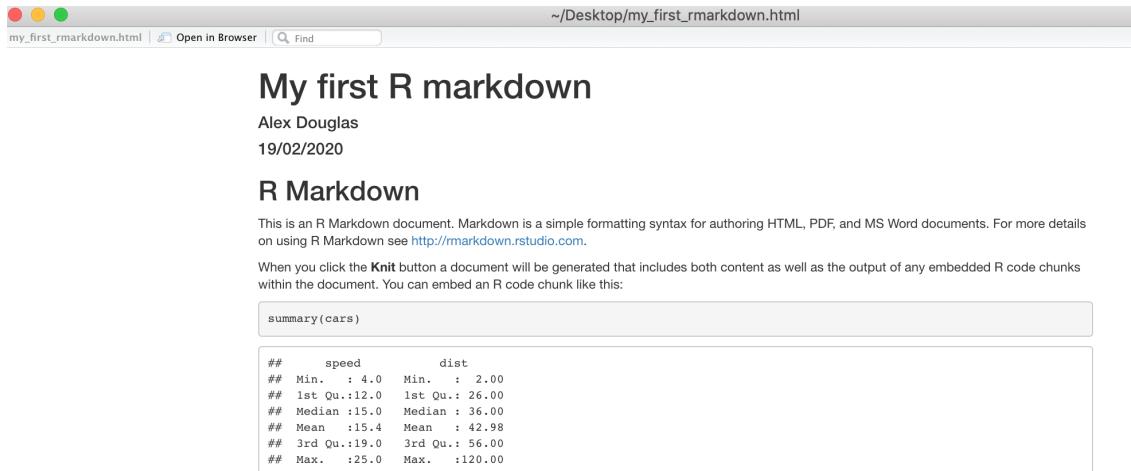
Figure 6.4.: A new Quarto document



A screenshot of the same Quarto document as above, but with the "Knit" button in the toolbar open, revealing a dropdown menu. The menu includes options for knitting to HTML, PDF, or Word, as well as other options like "Knit with Parameters..." and "Knit Directory". The "Knit to HTML" option is highlighted. The rest of the document content is identical to Figure 6.4.

Figure 6.5.: Knitting a Qmd file

rendu. RStudio affichera également un aperçu du fichier rendu dans une nouvelle fenêtre pour que vous puissiez le vérifier (votre fenêtre peut être légèrement différente si vous utilisez un ordinateur Windows).



Including Plots

You can also embed plots, for example:

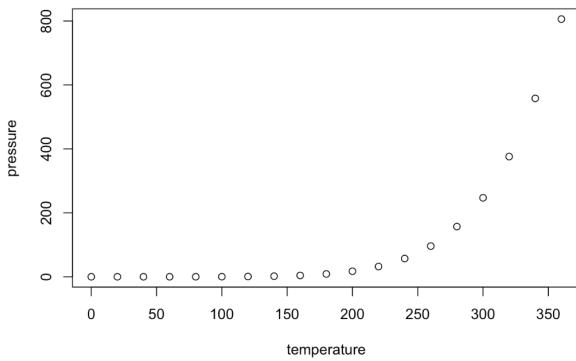


Figure 6.6.: A my first rendered html

Vous venez de rendre votre premier document Quarto. Si vous souhaitez tricoter votre .qmd en un document pdf, il vous suffit de choisir `knit to PDF` au lieu de `knit to HTML` lorsque vous cliquez sur le bouton `knit` l’icône Cela créera un fichier appelé `my_first_quarto.pdf` que vous pouvez ouvrir en double-cliquant dessus. Essayez-le !

Vous pouvez également tricoter un .qmd en utilisant la ligne de commande dans la console plutôt qu’en cliquant sur l’icône de tricotage. Pour ce faire, il suffit d’utiliser la commande `quarto_render()` de la fonction `quarto` 📦 package, comme indiqué ci-dessous. Là encore, vous pouvez modifier le format de sortie à l’aide de la fonction `output_format =` ainsi que de nombreuses autres options.

```
library(quarto)

quarto_render('my_first_quarto.qmd', output_format = 'html_document')
```

```
# alternatively if you don't want to load the quarto package

quarto::quarto_render('my_first_quarto.Rmd', output_format = 'html_document')
```

6.4. Anatomie du document Quarto (.qmd)

Maintenant que vous pouvez rendre un fichier Quarto dans RStudio aux formats HTML et PDF, examinons de plus près les différents composants d'un document Quarto typique. Normalement, chaque document Quarto est composé de 3 éléments principaux :

1. un en-tête YAML
2. texte formaté
3. des morceaux de code.

The screenshot shows an RStudio interface with a code editor containing a qmd file. The code is organized into three main sections:

- YAML header:** Lines 1 through 9, starting with `---` and ending with `---`. It includes metadata like title, author, date, and output format.
- formatted text:** Lines 10 through 16, which are regular text blocks interspersed with code blocks. They describe global options for the knitr package and provide setup instructions.
- code chunk:** Lines 17 through 28, which begin with ````{r}`` and end with `}```. This chunk contains R code for importing data from a file and calculating biomass.

Figure 6.7.: Structure of a qmd file

6.4.1. En-tête YAML

YAML signifie 'YAML Ain't Markup L anguage' (c'est une blague "in") [plaisanterie][blague] !) et ce composant optionnel contient les métadonnées et les options pour l'ensemble du document comme le nom de l'auteur, la date, le format de sortie, etc. L'en-tête YAML est entouré avant et après d'une balise --- sur sa propre ligne. Dans RStudio,

un en-tête YAML minimal est automatiquement créé pour vous lorsque vous créez un nouveau document Quarto comme nous l'avons fait ci-dessus (Section 6.3.2), mais vous pouvez le modifier à tout moment. Un en-tête YAML simple peut ressembler à ceci :

```
---
```

```
title: My first Quarto document
author: Jane Doe
date: March 01, 2020
format: html
---
```

Dans l'en-tête YAML ci-dessus, le format de sortie est défini sur HTML. Si vous souhaitez changer le format de sortie au format pdf, vous pouvez le changer de `format: html` à `format: pdf` (vous pouvez également définir plusieurs formats de sortie si vous le souhaitez). Vous pouvez également modifier la police et la taille de police par défaut pour l'ensemble du document et même inclure des options fantaisistes telles qu'une table des matières, des références en ligne et une bibliographie. Si vous souhaitez explorer la pléthore d'autres options, voir [ici](#). Attention, la plupart des options que vous pouvez spécifier dans l'en-tête YAML fonctionneront avec les documents au format HTML et PDF, mais pas toutes. Si vous avez besoin de plusieurs formats de sortie pour votre document Quarto, vérifiez si vos options YAML sont compatibles avec ces formats. De plus, l'indentation dans l'en-tête YAML a une signification, soyez donc prudent lorsque vous alignez du texte. Par exemple, si vous souhaitez inclure une table des matières, vous devez modifier l'option `output` : dans l'en-tête YAML comme suit

```
---
```

```
title: My first Quarto document
author: Bob Hette
date: March 01, 2020
format:
  html:
    toc: true
---
```

6.4.2. Texte formaté

Comme mentionné ci-dessus, l'un des avantages de Quarto est que vous n'avez pas besoin de votre traitement de texte pour rassembler votre code R, votre analyse et votre écriture. Quarto est capable de restituer (presque) tout le

formatage de texte dont vous pourriez avoir besoin, comme l’italique, le gras, le barré, l’indice supérieur et l’indice inférieur, ainsi que les listes à puces et numérotées, les en-têtes et les pieds de page, les images, les liens vers d’autres documents ou pages web, et aussi les équations. Cependant, contrairement à ce que vous connaissez, le *Ce que vous voyez, c'est ce que vous obtenez* ([WYSIWYG](#)), vous ne voyez pas le texte formaté final dans votre document Quarto (comme vous le feriez avec MS Word), mais vous devez “baliser” le formatage de votre texte pour qu’il soit rendu dans votre document de sortie. À première vue, cela peut sembler une véritable plaie, mais c’est en fait très facile à faire et cela présente de nombreux avantages. [avantages](#) (passez-vous plus de temps à embellir votre texte dans MS Word qu’à rédiger un contenu de qualité ?)

Voici un exemple de marquage du formatage du texte dans un document Quarto

```
#### Tadpole sediment experiment
```

These data were obtained from a mesocosm experiment which aimed to examine the effect of bullfrog tadpoles (**Lithobates catesbeianus**) biomass on sediment nutrient (NH₄[~], NO₃[~] and PO₄[~]) release.

At the start of the experiment 15 replicate mesocosms were filled with 20 cm² of **homogenised** marine sediment and assigned to one of five tadpole biomass treatments.

qui ressemblerait à ceci dans le document rendu final (pouvez-vous repérer les marques ?)

Expérience de sédimentation sur les têtards

Ces données proviennent d’une expérience en mésocosme qui visait à examiner les effets de la sédimentation sur les têtards. l’effet des têtards de grenouille-taureau (*Lithobates catesbeianus*) sur les sédiments nutriments (NH₄ NO₃ et PO₄). Au début de l’expérience, 15 mésocosmes répétés ont été remplis de 20 cm² de **homogénéisé** sédiments marins homogénisés et classés dans l’une des cinq catégories suivantes biomasse de têtards.

L’accent est mis

La syntaxe markdown la plus courante pour mettre en valeur et formater du texte est présentée ci-dessous.

Objectif	Quarto	production
texte en gras	**mytext**	mon texte
texte en italique	*mytext*	<i>mon texte</i>
barré	~~mytext~~	mon texte
exposant	mytext^2^	mon texte ²
indice	mytext~2~	mon texte ₂

Il est intéressant de noter qu'il n'y a pas de soulignement par défaut dans la syntaxe de R markdown, pour des raisons plus ou moins ésoptériques (*par exemple*. un soulignement est considéré comme un élément stylistique (il peut y avoir d'autres raisons). [raisons][souligner]). Quarto a corrigé ce problème, vous pouvez simplement faire [text to underline]{.underline} pour souligner votre texte.

Espaces blancs et sauts de ligne

L'une des choses qui peut être déroutante pour les nouveaux utilisateurs de markdown est l'utilisation des espaces et des retours à la ligne (la touche Entrée de votre clavier). En markdown, les espaces multiples dans le texte sont généralement ignorés, tout comme les retours à la ligne. Par exemple, ce texte en markdown

```
These      data were      obtained from a
mesocosm experiment which      aimed to examine the
effect
of          bullfrog tadpoles (*Lithobates catesbeianus*) biomass.
```

sera rendu sous la forme

Ces données ont été obtenues à partir d'un expérience en mésocosme qui visait à examiner l'impact de l'utilisation de l'eau sur la santé. l'effet des têtards de grenouille-taureau (*Lithobates catesbeianus*) biomasse.

C'est généralement une bonne chose (plus d'espaces multiples aléatoires dans votre texte). Si vous voulez que votre texte commence sur une nouvelle ligne, vous pouvez simplement ajouter deux espaces vides à la fin de la ligne précédente.

Ces données ont été obtenues à partir d'un expérience en mésocosme qui visait à examiner l'impact de l'utilisation de l'eau sur la qualité de l'air. effet des têtards de grenouille-taureau (*Lithobates catesbeianus*).

Si vous voulez vraiment des espaces multiples dans votre texte, vous pouvez utiliser la fonction Nsur **breaking space** tag

```
These &nbsps; &nbsps; &nbsps; data were &nbsps; &nbsps; &nbsps; obtained from a
mesocosm experiment which &nbsps; &nbsps; aimed to examine the
effect &nbsps; &nbsps; &nbsps; bullfrog tadpoles (*Lithobates catesbeianus*) biomass.
```

Ces données ont été obtenues à partir d'un d'une expérience en mésocosme qui visait à examiner l'impact de l'utilisation de l'eau sur la santé. effet des têtards de grenouille-taureau (*Lithobates catesbeianus*).

Rubriques

Vous pouvez ajouter des titres et des sous-titres à votre document Quarto en utilisant la fonction # en début de ligne. Vous pouvez réduire la taille des titres en ajoutant simplement plus de # symboles. Par exemple, il est possible de réduire la taille des titres en ajoutant simplement des symboles supplémentaires.

```
# Header 1
## Header 2
### Header 3
#### Header 4
##### Header 5
###### Header 6
```

permet d'obtenir des titres par ordre de taille décroissante

En-tête 1

En-tête 2

En-tête 3

En-tête 4

En-tête 5

Commentaires

Comme vous pouvez le voir ci-dessus, la signification de la # est différente lorsqu'il s'agit de formater du texte dans un document Quarto par rapport à un script R standard (qui est utilisé pour inclure un commentaire - vous vous souvenez ?!). Vous pouvez cependant utiliser un # pour commenter du code à l'intérieur d'un morceau de code (Section 6.4.1) comme d'habitude (plus d'informations à ce sujet dans un instant). Si vous souhaitez inclure un commentaire dans votre document Quarto en dehors d'un morceau de code qui ne sera pas inclus dans le document rendu final, placez votre commentaire entre les symboles <!-- et -->.

```
<!--  
this is an example of how to format a comment using Quarto.  
-->
```

Listes

Si vous souhaitez créer une liste de texte à puces, vous pouvez mettre en forme une liste non ordonnée avec des sous-éléments. Notez que les sous-éléments doivent être indentés.

```
- item 1
- item 2
  + sub-item 2
  + sub-item 3
- item 3
- item 4
```

- élément 1

- point 2

- sous-poste 2
 - sous-poste 3

- point 3

- point 4

Si vous avez besoin d'une liste ordonnée

1. item 1
1. item 2
 - + sub-item 2
 - + sub-item 3
1. item 3
1. item 4

1. article 1

2. point 2

- sous-poste 2

- sous-poste 3

3. point 3

4. point 4

Liens

Outre les images, vous pouvez également inclure des liens vers des pages web ou d'autres liens dans votre document. Utilisez la syntaxe suivante pour créer un lien cliquable vers une page web existante. Le texte du lien est placé entre les crochets et l'URL de la page web entre les crochets ronds immédiatement après.

You can include a text for your clickable [link] (<https://www.worldwildlife.org>)

qui vous donne :

Vous pouvez inclure un texte pour votre lien cliquable [cliquable](#)

6.4.1. Morceaux de code

Venons-en maintenant au cœur du problème. Pour inclure du code R dans votre document Quarto, il vous suffit de placer votre code dans un “morceau de code”. Tous les morceaux de code commencent et se terminent par trois antisèches ““““. Remarque : ces signes sont également appelés “accents graves” ou “guillemets arrière” et n’ont

rien à voir avec une apostrophe ! Sur la plupart des claviers, vous pouvez [trouver la coche arrière][bâton arrière] sur la même touche que le tilde (~).

```
```{r}
Any valid R code goes here
```
```

Vous pouvez insérer un morceau de code soit en tapant les délimiteurs du morceau ““{r}”” soit en utilisant l'option de votre IDE (barre d'outils de RStudio (bouton Insérer) ou en cliquant sur le menuCode->Insert Chunk'. Dans VS Code, vous pouvez utiliser des extraits de code). Le mieux est peut-être de se familiariser avec les raccourcis clavier de votre IDE ou de vos extraits de code.

Il y a beaucoup de choses que vous pouvez faire avec des morceaux de code : vous pouvez produire du texte à partir de votre analyse, créer des tableaux et des figures et insérer des images, entre autres choses. À l'intérieur du morceau de code, vous pouvez placer des règles et des arguments entre les accolades. {} qui vous permettent de contrôler l'interprétation de votre code et le rendu des résultats. C'est ce qu'on appelle les options du bloc de code. La seule option obligatoire est le premier argument qui spécifie le langage utilisé (r dans notre cas, mais [d'autres][moteurs] langues sont prises en charge). Remarque : les options de blocs peuvent être écrites de deux manières :

1. soit toutes les options de blocs doivent être écrites entre les crochets, sur une seule ligne, sans retour à la ligne.
1. soit elles peuvent être écrites en utilisant une notation YAML à l'intérieur du morceau de code en utilisant #| au début de la ligne.

Nous utilisons la notation YAML pour les options du bloc de code car nous la trouvons beaucoup plus facile à lire lorsque vous avez plusieurs options de longues légendes.

Vous pouvez également spécifier un nom (ou étiquette) facultatif pour le morceau de code, ce qui peut s'avérer utile en cas de problèmes de débogage et de rendu avancé de documents. Dans le bloc suivant, nous nommons le morceau de code **summary-stats**, nous chargeons le paquet **ggplot2**  créer un cadre de données (**dataf**) avec deux variables **x** et **y**, utiliser la méthode **summary()** pour afficher des statistiques sommaires et tracer un nuage de points des données à l'aide de la fonction **ggplot()**. Lorsque nous exécutons le bloc de code, le code R et la sortie résultante sont affichés dans le document final.

```
```{r, summary-stats, echo = TRUE, fig.cap = "Caption for a simple figure but making the chunk options long and hard to read"

library(ggplot)

x <- 1:10 # create an x variable
y <- 10:1 # create a y variable
dataf <- data.frame(x = x, y = y)

summary(dataf)
ggplot(dataf, aes(x = x, y = y)) + geom_point()
````
```

```
```{r}
#| label: summary-stats
#| echo: true
#| fig-cap = "Caption for a simple figure but making the chunk options long and hard to read"

x <- 1:10 # create an x variable
y <- 10:1 # create a y variable
dataf <- data.frame(x = x, y = y)

summary(dataf)
ggplot(dataf, aes(x = x, y = y)) + geom_point()
````
```

Tous deux produiront

```
x <- 1:10      # create an x variable
y <- 10:1      # create a y variable
dataf <- data.frame(x = x, y = y)

summary(dataf)
```

| x | y |
|---------------|---------------|
| Min. : 1.00 | Min. : 1.00 |
| 1st Qu.: 3.25 | 1st Qu.: 3.25 |
| Median : 5.50 | Median : 5.50 |
| Mean : 5.50 | Mean : 5.50 |
| 3rd Qu.: 7.75 | 3rd Qu.: 7.75 |
| Max. : 10.00 | Max. : 10.00 |

```
ggplot(dataf, aes(x = x, y = y)) + geom_point()
```

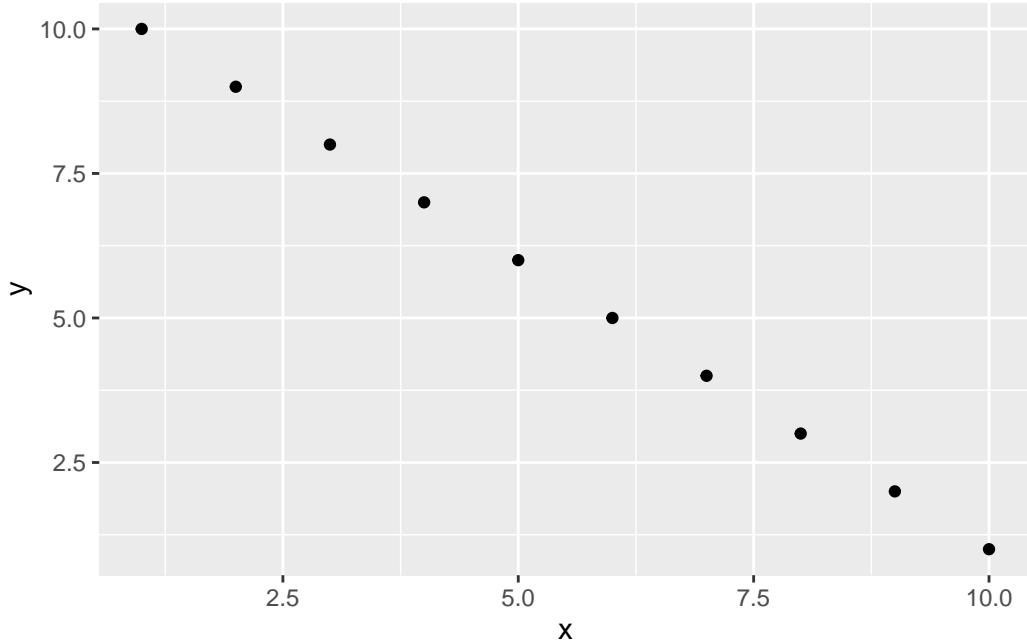


Figure 6.8.: Caption for a simple figure but making the chunk options long and hard to read

Lorsque vous utilisez des noms de morceaux, assurez-vous que vous n'avez pas de noms de morceaux en double dans votre document Quarto et évitez les espaces et les points, car cela posera des problèmes lorsque vous devrez tricoter votre document. – pour séparer les mots dans nos noms de blocs).

Si nous voulons afficher uniquement la sortie de notre code R (juste les statistiques sommaires par exemple) et non le code lui-même dans notre document final, nous pouvons utiliser l'option chunk. `echo=FALSE`

```
```{r}
#| label: summary-stats2
#| echo: false
x <- 1:10 # create an x variable
y <- 10:1 # create a y variable
dataf <- data.frame(x = x, y = y)
summary(dataf)
````
```

| x | y |
|---------------|---------------|
| Min. : 1.00 | Min. : 1.00 |
| 1st Qu.: 3.25 | 1st Qu.: 3.25 |
| Median : 5.50 | Median : 5.50 |
| Mean : 5.50 | Mean : 5.50 |
| 3rd Qu.: 7.75 | 3rd Qu.: 7.75 |
| Max. :10.00 | Max. :10.00 |

Pour afficher le code R mais pas la sortie, utilisez l'option `results='hide'` l'option chunk.

```
```{r}
#| label: summary-stats
#| results: 'hide'
x <- 1:10 # create an x variable
y <- 10:1 # create a y variable
dataf <- data.frame(x = x, y = y)
summary(dataf)
````
```

```
x <- 1:10      # create an x variable
y <- 10:1      # create a y variable
```

```
dataf <- data.frame(x = x, y = y)
summary(dataf)
```

Il peut arriver que vous souhaitiez exécuter un morceau de code sans afficher aucune sortie. Vous pouvez supprimer la totalité de la sortie en utilisant l'option chunk `include: false`.

```
```{r}
#| label: summary-stats4
#| include: false
x <- 1:10 # create an x variable
y <- 10:1 # create a y variable
dataf <- data.frame(x = x, y = y)
summary(dataf)
```

```

Il existe un grand nombre d'options de morceaux documentées [ici](#) avec une version plus condensée [ici](#). Les plus couramment utilisées sont résumées ci-dessous, les valeurs par défaut étant indiquées.

| Option d'assemblage | valeur par | |
|---|-----------------------|--|
| | défaut | Fonction |
| écho | <code>echo:</code> | Si <code>false</code> n'affichera pas le code dans le document final |
| | <code>true</code> | document final |
| résultats | <code>results:</code> | Si “cacher”, les résultats du code ne seront pas affichés dans le document final.
‘markup’ affichés dans le document final. |
| Si “hold”, l'affichage de tous les éléments de sortie sera retardé jusqu'à la fin du morceau. | | |
| Si ‘asis’, les résultats seront affichés sans être reformatés. | | |

`include | include: true | Si false` exécute le bloc mais ne l'inclut pas dans le document final.

eval | eval: true | Si false n'exécutera pas le code contenu dans le morceau de code.

message | message: true | Si false n'affichera pas les messages générés par le code.

avertissement | warning: true | Si false n'affichera pas les messages d'avertissement générés par le code.

6.4.2. Code R en ligne

Jusqu'à présent, nous avons écrit et exécuté notre code R par morceaux. Une autre bonne raison d'utiliser Quarto est que nous pouvons également inclure notre code R directement dans notre texte. C'est ce qu'on appelle le "code en ligne". Pour inclure votre code dans votre texte Quarto, il vous suffit d'écrire `r write your code here`. Cela peut s'avérer très utile lorsque vous souhaitez inclure des statistiques sommaires dans votre texte. Par exemple, nous pourrions décrire le `iris` comme suit :

```
Morphological characteristics (variable names:  
`r names(iris)[1:4]` were measured from  
`r nrow(iris)` *Iris sp.* plants from  
`r length(levels(iris$Species))` different species.  
The mean Sepal length was  
`r round(mean(iris$Sepal.Length), digits = 2)` mm.
```

qui sera rendu par

Caractéristiques morphologiques (noms de variables : Sepal.Length, Sepal.Width, Petal.Length, Petal.Width) étaient mesurés à partir de 150 *l'iris* plantes de 3 différentes espèces. La longueur moyenne des sépales était de 5.84 mm.

L'avantage d'inclure du code R en ligne dans votre texte est que ces valeurs seront automatiquement mises à jour si vos données changent.

6.4.3. Images et photos

La possibilité d'intégrer des images et des liens vers des pages web (ou d'autres documents) dans votre document Quarto est une fonctionnalité utile. Vous pouvez inclure des images dans votre document Quarto de différentes manières. La méthode la plus simple est sans doute d'utiliser le format markdown de Quarto :

```
![Image caption] (path/to/you/image){options}
```

Voici un exemple avec une image occupant 75 % de la largeur et centrée.

```
![Waiting for the eclipse] (images/markdown/eclipse_ready.jpg){fig-align="center" width="75%"} 
```

résultant en :

Une autre façon d'inclure des images dans votre document est d'utiliser la fonction `include_graphics()` de la fonction `knitr` du paquet. Le code suivant produira un résultat similaire.

```
```{r}
#| label: fig-knitr
#| fig-align: center
#| out-width: 75%
#| fig-cap: Waiting for the eclipse
knitr:::include_graphics("images/markdown/eclipse_ready.jpg")
```
```

Le code ci-dessus ne fonctionnera que si le fichier image (`eclipse_ready.jpg`) se trouve au bon endroit par rapport à l'endroit où vous avez enregistré votre fichier `.qmd` fichier. Dans l'exemple, le fichier image se trouve dans un sous-répertoire (dossier) appelé `images/markdown` dans le répertoire où nous avons enregistré notre



Figure 6.9.: En attendant l'éclipse

`my_first_quarto.qmd` fichier. Vous pouvez intégrer des images enregistrées dans de nombreux types de fichiers différents, mais les plus courants sont les suivants `.jpg` et `.png`.

6.4.4. Les chiffres

Par défaut, les figures produites par le code R sont placées immédiatement après le morceau de code à partir duquel elles ont été générées. Par exemple :

```
```{r}
#| label: fig-simple-plot
#| fig-cap: A simple plot
x <- 1:10 # create an x variable
y <- 10:1 # create a y variable
dataf <- data.frame(x = x, y = y)
plot(dataf$x, dataf$y, xlab = "x axis", ylab = "y axis")
```

```

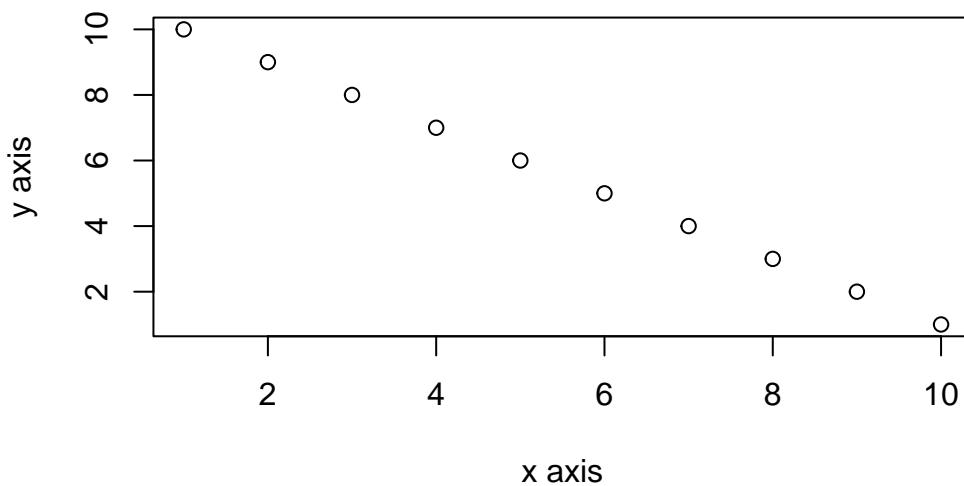


Figure 6.10.: A simple plot

Le fichier **fig-cap**: permet de fournir une légende de figure reconnue par Quarto et utilisée dans la numérotation des figures et les références croisées (Section 6.4.6).

Si vous souhaitez modifier les dimensions de l'intrigue dans le document final, vous pouvez utiliser l'option **fig-width:** et **fig-height:** (en pouces !). Vous pouvez également modifier l'alignement de la figure à l'aide de la fonction **fig-align:** option chunk.

```
```{r}
#| label: fig-simple-plot2
#| fig-cap: A shrinked figure
#| fig-width: 4
#| fig-height: 3
#| fig-align: center
x <- 1:10 # create an x variable
y <- 10:1 # create a y variable
dataf <- data.frame(x = x, y = y)
plot(dataf$x, dataf$y, xlab = "x axis", ylab = "y axis")
````
```

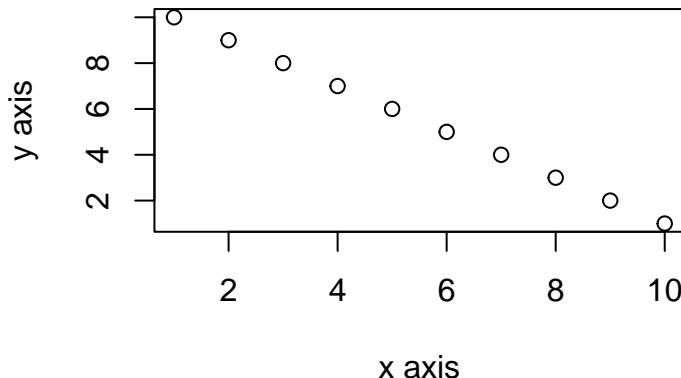


Figure 6.11.: A shrinked figure

Vous pouvez ajouter une légende à la figure à l'aide de la fonction **fig-cap:** option.

```
```{r}
#| label: fig-simple-plot-cap
#| class-source: fold-show
#| fig-cap: A simple plot
#| fig-align: center

x <- 1:10 # create an x variable
y <- 10:1 # create a y variable
dataf <- data.frame(x = x, y = y)
plot(dataf$x, dataf$y, xlab = "x axis", ylab = "y axis")
```

```

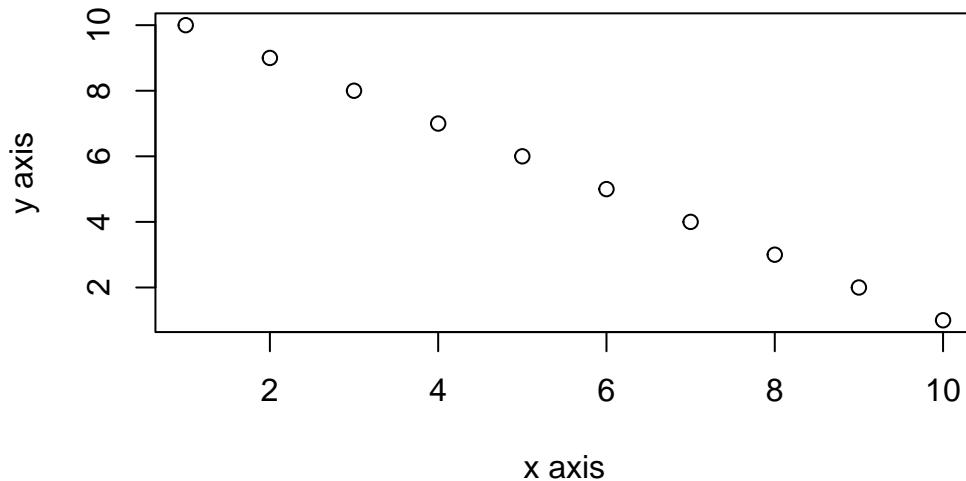


Figure 6.12.: A simple plot

Si vous souhaitez supprimer la figure dans le document final, utilisez l'option `fig-show: 'hide'` pour supprimer la figure dans le document final.

```
```{r}
#| label: fig-simple-plot5
#| fig-show: hide
```

```

```
x <- 1:10      # create an x variable
y <- 10:1      # create a y variable
dataf <- data.frame(x = x, y = y)
plot(dataf$x, dataf$y, xlab = "x axis", ylab = "y axis")
````
```

Si vous utilisez un logiciel comme ggplot2 📈 pour créer vos parcelles, n'oubliez pas que vous devrez rendre le paquet disponible avec l'option `library()` dans le morceau de code (ou dans un morceau de code précédent).

```
```{r}
#| label: fig-simple-ggplot
#| fig-cap: A simple ggplot
x <- 1:10      # create an x variable
y <- 10:1      # create a y variable
dataf <- data.frame(x = x, y = y)

library(ggplot2)
ggplot(dataf, aes(x = x, y = y)) +
  geom_point()
````
```

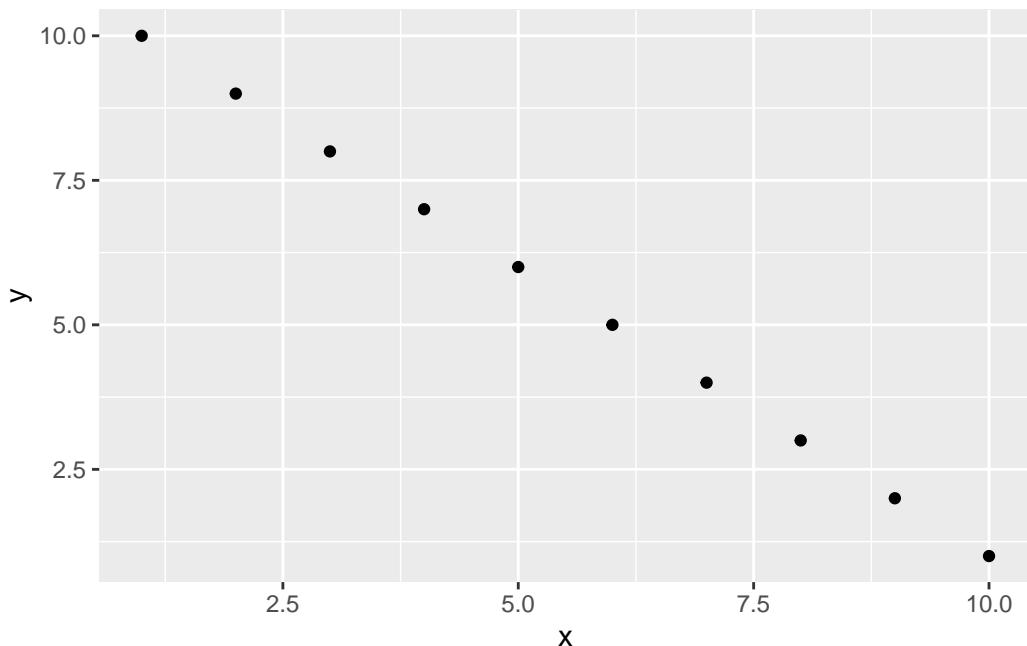


Figure 6.13.: A simple ggplot

Là encore, il existe un grand nombre d'options spécifiques à la production de graphiques et de figures. Voir [ici](#) pour plus de détails.

#### 6.4.5. Tableaux

Dans Quarto, vous pouvez créer des tableaux en utilisant la syntaxe markdown native (il n'est pas nécessaire que ce soit dans un morceau de code).

| x | y |
|---|---|
| 1 | 5 |
| 2 | 4 |
| 3 | 3 |
| 4 | 2 |
| 5 | 1 |

: Caption for a simple markdown table

---

| x | y |
|---|---|
| 1 | 5 |
| 2 | 4 |
| 3 | 3 |
| 4 | 2 |
| 5 | 1 |

---

Caption pour un simple tableau markdown {#tbl-simp-md}

Le :-----: indique à markdown que la ligne du dessus doit être traitée comme un en-tête et les lignes du dessous comme le corps du tableau. L'alignement à l'intérieur du tableau est déterminé par la position de la balise :. Pour centrer l'alignement, utilisez :-----: pour aligner à gauche :----- et aligner à droite -----:. Bien qu'il puisse être amusant (!) de créer des tableaux avec des balises brutes, cela n'est pratique que pour les tableaux très petits et très simples.

La façon la plus simple d'inclure des tableaux dans un document Quarto est d'utiliser la fonction `kable()` de la fonction `knitr`  package. La fonction `kable()` permet de créer des tableaux pour les formats HTML, PDF et Word.

Pour créer un tableau des 2 premières lignes par espèce de l'échantillon `iris` à l'aide de la fonction `kable()` il suffit d'écrire

```
library(knitr)
iris %>%
 group_by(Species) %>%
 slice_head(n = 2) %>%
kable()
```

ou sans charger `knitr`  mais en indiquant où trouver le `kable()` fonction.

```
iris %>%
 group_by(Species) %>%
 slice_head(n = 2) %>%
 knitr::kable()
```

Table 6.4.: A simple kable table

| Sepal.Length | Sepal.Width | Petal.Length | Petal.Width | Species    |
|--------------|-------------|--------------|-------------|------------|
| 5.1          | 3.5         | 1.4          | 0.2         | setosa     |
| 4.9          | 3.0         | 1.4          | 0.2         | setosa     |
| 7.0          | 3.2         | 4.7          | 1.4         | versicolor |
| 6.4          | 3.2         | 4.5          | 1.5         | versicolor |
| 6.3          | 3.3         | 6.0          | 2.5         | virginica  |
| 5.8          | 2.7         | 5.1          | 1.9         | virginica  |

La fonction `kable()` offre de nombreuses options pour modifier la mise en forme du tableau. Par exemple, si nous voulons arrondir les valeurs numériques à une décimale, utilisez la fonction `digits = argument`. Pour centrer le contenu du tableau, utilisez `align = 'c'` et pour fournir des en-têtes de colonne personnalisés, utilisez l'argument `col.names = argument`. Voir `?knitr::kable` pour plus d'informations.

```
iris %>%
 group_by(Species) %>%
 slice_head(n = 2) %>%
 knitr::kable(
 digits=0,
 align = 'c',
 col.names = c(
 'Sepal length', 'Sepal width',
 'Petal length', 'Petal width', 'Species'
```

```
)
)
```

Table 6.5.: A nicer kable table

| Sepal length | Sepal width | Petal length | Petal width | Species    |
|--------------|-------------|--------------|-------------|------------|
| 5            | 4           | 1            | 0           | setosa     |
| 5            | 3           | 1            | 0           | setosa     |
| 7            | 3           | 5            | 1           | versicolor |
| 6            | 3           | 4            | 2           | versicolor |
| 6            | 3           | 6            | 2           | virginica  |
| 6            | 3           | 5            | 2           | virginica  |

Vous pouvez encore améliorer l'aspect de votre `kable` tables en utilisant la fonction `kableExtra`  package (n'oubliez pas d'installer le package au préalable !). Voir [ici](#) pour plus de détails et un tutoriel utile.

Si vous voulez encore plus de contrôle et d'options de personnalisation pour vos tables, jetez un coup d'œil à l'option `gt`  [paquet][gt]. `gt` est un acronyme pour grammar de `tet` est basé sur un principe similaire pour les tableaux qui sont utilisés pour les tracés dans `ggplot`.

```
iris %>%
 group_by(Species) %>%
 slice_head(n = 2) %>%
 rename_with(~ gsub("(\\.)", " ", .x)) %>%
 gt()
```

Table 6.6.: A nice gt table

| Sepal Length | Sepal Width | Petal Length | Petal Width |
|--------------|-------------|--------------|-------------|
| setosa       |             |              |             |
| 5.1          | 3.5         | 1.4          | 0.2         |
| 4.9          | 3.0         | 1.4          | 0.2         |
| versicolor   |             |              |             |
| 7.0          | 3.2         | 4.7          | 1.4         |
| 6.4          | 3.2         | 4.5          | 1.5         |
| virginica    |             |              |             |
| 6.3          | 3.3         | 6.0          | 2.5         |
| 5.8          | 2.7         | 5.1          | 1.9         |

Dans la plupart des paquets R développés pour produire des tableaux, il existe des options permettant d'inclure des légendes de tableaux. Cependant, si vous souhaitez ajouter une légende de tableau, nous vous recommandons d'utiliser l'option code chunk dans Quarto `tbl-cap:` car cela permet des références croisées (Section 6.4.6) et une meilleure intégration dans le document.

```
```{r}
#| label: tbl-gt-table
#| tbl-cap: A nice gt table
#| echo: true

iris %>%
  group_by(Species) %>%
  slice_head(n=2) %>%
  rename_with(~gsub("([._])", " ", .x)) %>%
  gt()
```

```

### 6.4.6. Références croisées

Les références croisées permettent aux lecteurs de naviguer plus facilement dans votre document en fournissant des références numérotées et des liens hypertextes vers diverses entités telles que les figures et les tableaux. Une fois configurée, la numérotation des tableaux et des figures se fait automatiquement, de sorte qu'il n'est pas nécessaire de renommer toutes les figures lorsque vous en ajoutez ou en supprimez une.

Chaque entité pouvant faire l'objet d'une référence croisée nécessite une étiquette (un identifiant unique) précédée d'un type de référence croisée, par exemple `#fig-element`.

Pour plus de détails, voir le site [section sur les références croisées sur le site web de Quarto](#).

#### 6.4.6.1. Sections du document

Vous pouvez faire des références croisées avec d'autres sections du document. Pour ce faire, vous devez

1. définir un identifiant pour la section vers laquelle vous souhaitez établir un lien. L'identifiant doit :

- commencer par `#sec-`
- en minuscules (figure 6.3)
- n'a pas d'espace, en utilisant – à la place

2. utiliser le @ et l'identifiant pour faire référence à la section

```
Cross-referencing sections {#sec-cross-ref-sections}

[...]

As seen before(@sec-cross-ref-sections)
```

#### 6.4.6.2. Images, figures et tableaux

Pour les tableaux, les images et les figures, outre l'identifiant, l'élément doit également être accompagné d'une légende pour que les références croisées fonctionnent.

Le préfixe pour les tableaux est `#tbl-` et `#fig-` pour les images et les figures.

Voici un exemple d'image incluse dans un texte en markdown :

```
! [Rocking the eclipse] (images/markdown/eclipse_ready.jpg){#fig-cute-dog}
```

See `@fig-cute-dog` for an illustration.

Voir Figure 6.14 pour une illustration.

Pour les figures et les tableaux produits avec des morceaux de code R, il suffit de fournir l'identifiant dans le champ `label` et la légende également en tant qu'option de bloc.

Voici le code pour une figure et un tableau.

```
```{r}
#| label: fig-cr-plot
#| fig-cap: A nice figure
x <- 1:10      # create an x variable
y <- 10:1      # create a y variable
dataf <- data.frame(x = x, y = y)

library(ggplot2)
ggplot(dataf, aes(x = x, y = y)) +
  geom_point()
```
```



Figure 6.14.: Le rock de l'éclipse

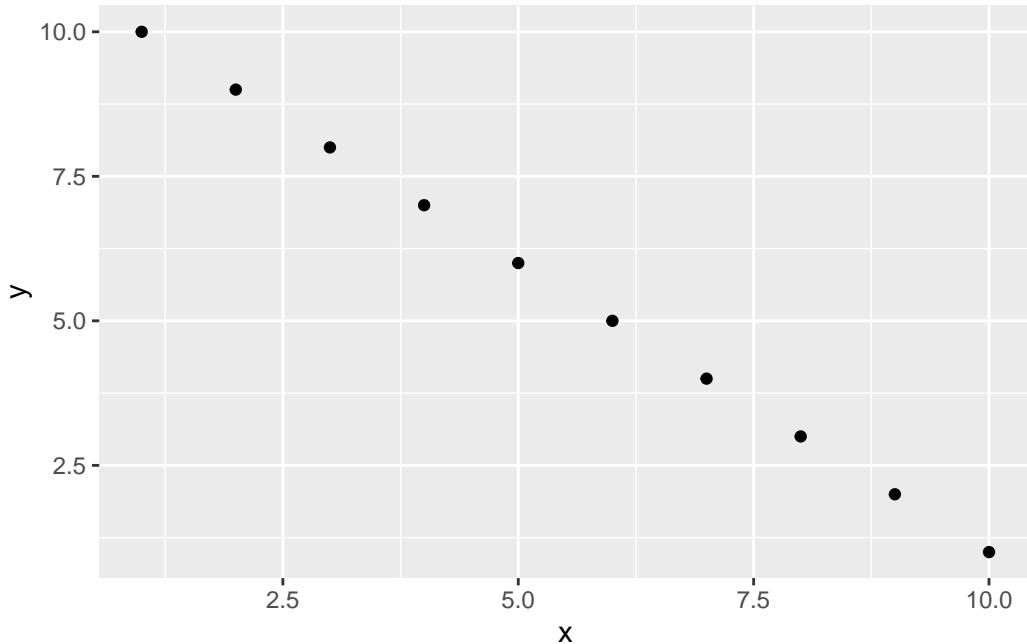


Figure 6.15.: A nice figure

```
```{r}
```

```
#| label: tbl-cr-table
```

```
#| tbl-cap: A nice table
```

```
#| warning: false
```

```
library(knitr)
```

```
kable(iris[1:5], digits=0, align = 'c', col.names = c('sepal length', 'sepal width', 'petal leng
```

Table 6.7.: A nice table

sepal length	sepal width	petal length	petal width	species
5	4	1	0	setosa
5	3	1	0	setosa
5	3	1	0	setosa
5	3	2	0	setosa

Table 6.7.: A nice table

sepal length	sepal width	petal length	petal width	species
5	4	1	0	setosa

En utilisant les références croisées, on peut écrire :

Tel que vu sur @fig-cr-plot et @tbl-cr-table ...

Pour obtenir :

Comme vu sur Figure 6.15 et Table 6.7 ...

6.4.7. Citations et bibliographie

Pour générer des citations et une bibliographie, Quarto a besoin de :

- un document correctement formaté .qmd formaté
- un fichier source bibliographique comprenant toutes les informations pour les citations. Il fonctionne avec une grande variété de formats mais nous suggérons l'utilisation de {{< bibtex >}} le format {.
- (optionnel) un fichier CSL qui spécifie le formatage à utiliser lors de la génération des citations et de la bibliographie.

La source bibliographique et le fichier csl (facultatif) sont spécifiés dans l'en-tête yaml sous la forme :

```
---
```

```
title: "My Document"
```

```
bibliography: references.bib
```

```
csl: ecology.csl
```

```
---
```

6.4.7.1. Citations

Quarto utilise la représentation markdown standard de Pandoc pour les citations (par ex. `[@citation]`) - les citations sont placées entre crochets et séparées par des points-virgules. Chaque citation doit avoir une clé, composée de ‘@’ + l’identifiant de la base de données, et peut optionnellement avoir un préfixe, un localisateur, et un suffixe. La clé de la citation doit commencer par une lettre, un chiffre ou , *et peut contenir des caractères alphanumériques, et des caractères de ponctuation internes.*

Format Markdown	Sortie (default)
Les licornes sont les meilleures [voir @martin1219, pp. 33-35 ; aussi @martin2200, chap. 1]	Les licornes sont les meilleures (voir Martin 1219, pp. 33-35, aussi Martin 2200 chap. 1)
Les licornes sont les meilleures [@martin2200 ; @martin1219]	Les licornes sont les meilleures (Martin 1219, 2200)
Martin dit que les licornes sont les meilleures [-@martin2200]	Martin dit que les licornes sont les meilleures (2200)
@martin1219 dit que les licornes sont les meilleures. @martin1219 [p. 33] dit que les licornes sont ce qu'il y a de mieux.	Martin (1219) dit que les licornes sont les meilleures. La plupart des gens disent que les licornes sont ce qu'il y a de mieux.

6.4.7.2. Créer la bibliographie

Par défaut, la liste des ouvrages cités sera automatiquement générée et placée en fin de document si le style l’exige. Elle sera placée dans une div avec l’id refs s’il y en a une comme

```
### Bibliography
```

```
::: {#refs}
```

```
:::
```

Pour plus de détails, voir le site [page Citation sur le site de Quarto](#).

6.4.7.3. Intégration avec Zotero

Quarto s'intègre très bien avec **Zotero** si vous utilisez l'éditeur visuel de **RStudio** ou **VS Code**.

6.5. Quelques conseils et astuces

Problème :

Lors du rendu de mon document Quarto au format pdf, mon code sort du bord de la page.

Solution :

Ajoutez un argument `global_options` au début de votre fichier `.qmd` dans un morceau de code :

```
```{r}
#| label: global_options
#| include: false
knitr::opts_chunk$set(message=FALSE, tidy.opts=list(width.cutoff=60), tidy=TRUE)
````
```

Ce morceau de code ne sera pas affiché dans le document final en raison de l'argument `global_options. include: false` et vous devez placer le morceau de code immédiatement après l'en-tête YAML pour affecter tout ce qui se trouve en dessous.

`tidy.opts = list(width.cutoff = 60), tidy=TRUE` définit le point de coupure de la marge et fait passer le texte à la ligne suivante. Jouez avec cette valeur pour l'obtenir correctement (60-80 devrait convenir à la plupart des documents).

Avec quarto, vous pouvez également utiliser la fonction globale `knitr` dans un fichier `knitrd` dans l'en-tête YAML (voir [Site web de Quarto](#) pour plus de détails).

```
---
title: "My Document"
format: html
knitr:
  opts_chunk:
    message: false
    tidy.opts: !expr 'list(width.cutoff=60)'
    tidy: true
---
```

Problème :

Lorsque je charge un paquet dans mon document Quarto, le rendu contient tous les messages de démarrage et/ou les avertissements.

Solution :

Vous pouvez charger tous vos paquets au début de votre document Quarto dans un morceau de code en même temps que vous définissez vos options globales.

```
```{r}
#| label: global_options
#| include: false
knitr::opts_chunk$set(
 message = FALSE,
 warning=FALSE,
 tidy.opts=list(width.cutoff=60)
)
suppressPackageStartupMessages(library(ggplot2))
```
```

L'option `message = FALSE` et `warning = FALSE` suppriment les messages et les avertissements. Les `suppressPackageStartupMessages(library(ggplot2))` chargeront le fichier ggplot2  mais supprimera les messages de démarrage.

Le problème est le suivant :

Lors de la conversion de mon document Quarto en PDF, mes tableaux et/ou figures sont répartis sur deux pages.

Solution :

Ajoutez un saut de page à l'aide de la fonction `\pagebreak` avant le tableau ou la figure incriminé(e)

problème :

Le code dans mon document rendu est laid !

Solution :

Ajouter l'argument `tidy: true` à vos arguments globaux. Cependant, cela peut parfois poser des problèmes, notamment en ce qui concerne l'indentation du code. La meilleure solution est d'écrire un code qui a de l'allure (insérer des espaces et utiliser plusieurs lignes)

6.6. Informations complémentaires

Bien que nous ayons couvert plus qu'il n'en faut pour vous permettre d'aller loin avec Quarto, nous n'avons eu le temps que d'effleurer la surface, comme c'est le cas pour la plupart des choses liées aux technologies de l'information et de la communication. Heureusement, il existe une mine d'informations à votre disposition si vous souhaitez approfondir vos connaissances et votre expérience. Un bon point de départ est l'excellent site web de Quarto [ici](#).

Un autre guide de référence Quarto utile et concis peut être trouvé [ici](#)

Une feuille de calcul rapide et facile pour R Markdown [de R Markdown](#)

6.7. Pratique

Nous allons créer un nouveau document Rmarkdown et l'éditer en utilisant les fonctions de base de Rmarkdown. R et Rmarkdown et les fonctions

6.7.1. Le contexte

Nous utiliserons l'awesome `palmerpenguins` jeu de données 🐧 pour explorer et visualiser les données.

Ces données ont été collectées et partagées par [Dr. Kristen Gorman](#) et [Station Palmer, Antarctique LTER](#).

L'ensemble a été conçu par les Drs Allison Horst et Alison Hill. [site officiel](#).

Le paquet `palmerpenguins` comporte deux ensembles de données :

- `penguins_raw` a les données brutes des observations des manchots (voir `?penguins_raw` pour plus d'informations)
- `penguins` est une version simplifiée des données brutes (voir `?penguins` pour plus d'informations)

Pour cet exercice, nous allons utiliser la fonction `penguins` jeu de données.

```
library(palmerpenguins)
head(penguins)
```

| species | island | bill_length_mm | bill_depth_mm | flipper_length_mm | body_mass_g | sex | year |
|---------|-----------|----------------|---------------|-------------------|-------------|--------|------|
| Adelie | Torgersen | 39.1 | 18.7 | 181 | 3750 | male | 2007 |
| Adelie | Torgersen | 39.5 | 17.4 | 186 | 3800 | female | 2007 |
| Adelie | Torgersen | 40.3 | 18.0 | 195 | 3250 | female | 2007 |
| Adelie | Torgersen | NA | NA | NA | NA | NA | 2007 |
| Adelie | Torgersen | 36.7 | 19.3 | 193 | 3450 | female | 2007 |
| Adelie | Torgersen | 39.3 | 20.6 | 190 | 3650 | male | 2007 |

6.7.2. Questions

1) Installer le paquet `palmerpenguins`.

 Solution

```
install.packages("palmerpenguins")
```

2)

- Créez un nouveau document Quarto, nommez-le et enregistrez-le.
- Supprimez tout ce qui se trouve après la ligne 12.
- Ajouter un nouveau titre de section, un texte simple et un texte en caractères gras.
- Compiler (“Tricoter”).

3)

- Ajoutez un morceau dans lequel vous chargez le `palmerpenguins`. La ligne de code correspondante devrait être cachée dans la sortie.
- Chargez également le fichier `tidyverse` de paquets. Modifier les valeurs par défaut pour supprimer tous les messages.

 Solution

```
```{r}
#| echo: false
#| message:false
library(palmerpenguins)
library(tidyverse)
```
```

4) Ajoutez un autre morceau dans lequel vous construisez un tableau avec les 10 premières lignes de l'ensemble de données.

Solution

```
```{r}
penguins %>%
 slice(1:10) %>%
 knitr::kable()
```
```

- 5) Dans une nouvelle section, affichez le nombre d'individus, d'espèces de manchots et d'îles que nous avons dans l'ensemble de données. Cette information doit apparaître directement dans le texte, vous devez utiliser du code en ligne. 😊. Calculer la moyenne des traits (numériques) mesurés sur les manchots.

Solution

```
## Numerical exploration
```

There are `r nrow(penguins)` penguins in the dataset,
 and `r length(unique(penguins\$species))` different species.
 The data were collected in `r length(unique(penguins\$island))`
 islands of the Palmer archipelago in Antarctica.

The mean of all traits that were measured on the penguins are:

```
```{r}
#| echo: false
penguins %>%
 group_by(species) %>%
 summarize(across(where(is.numeric), mean, na.rm = TRUE))
```
```

- 6) Dans une autre section, intitulée “Exploration graphique”, construisez une figure avec 3 histogrammes superposés, chacun correspondant à la masse corporelle d'une espèce.

Solution

```
## Graphical exploration

A histogram of body mass per species:

```{r}
#| fig-cap: Distribution of body mass by species of penguins
ggplot(data = penguins) +
 aes(x = body_mass_g) +
 geom_histogram(aes(fill = species),
 alpha = 0.5,
 position = "identity") +
 scale_fill_manual(values = c("darkorange", "purple", "cyan4")) +
 theme_minimal() +
 labs(x = "Body mass (g)",
 y = "Frequency",
 title = "Penguin body mass")
```

```

7) Dans une autre section, intitulée *Régression linéaire* Ajustez un modèle de la longueur du bec en fonction de la taille du corps (longueur des nageoires), de la masse corporelle et du sexe. Obtenez le résultat et évaluez graphiquement les hypothèses du modèle. Pour rappel, voici comment effectuer une régression linéaire.

```
```{r}
model <- lm(Y ~ X1 + X2, data = data)
summary(model)
plot(model)
```

```

Solution

```
## Linear regression

And here is a nice model with graphical output

```{r}
#| fig-cap: "Checking assumptions of the model"
m1 <- lm(bill_length_mm ~ flipper_length_mm + body_mass_g + sex, data = penguins)
summary(m1)
par(mfrow= c(2,2))
plot(m1)
```

```

8) Ajouter des références manuellement ou à l'aide de `citr` dans RStudio.

1. Choisissez une publication récente du chercheur qui a partagé les données, le Dr Kristen Gorman. Importez cette publication dans votre gestionnaire de références favori (nous utilisons Zotero, sans complexe), et créez une référence bibtex que vous ajouterez au fichier `mabiblio.bib`.
2. Ajouter `bibliography`: `mabiblio.bib` au début de votre document R Markdown (YAML).
3. Citez la référence dans le texte en la tapant manuellement ou en utilisant la commande `citr`. Pour utiliser `citr` installez-le d'abord ; si tout se passe bien, vous devriez le voir apparaître dans le menu déroulant `Addins` . Il suffit ensuite d'utiliser `Insert citations` dans le menu déroulant `Addins`.
4. Compiler.

9) Changez le format de citation par défaut (style Chicago) en format The American Naturalist. Il se trouve ici <https://www.zotero.org/styles>. Pour ce faire, ajoutez `csl: the-american-naturalist.csl` dans le YAML.

10) Créez votre rapport au format html, pdf et docx. 

Exemple de résultats

Vous pouvez voir un exemple de la sortie [fichier source Rmarkdown](#) et [sortie pdf](#)

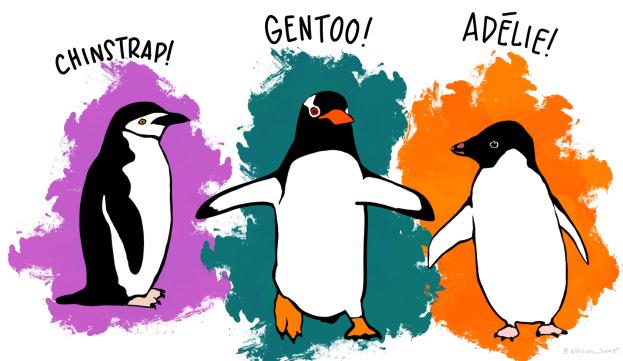


Figure 6.16.: Happy coding

Chapitre 7

Contrôle de version avec Git et GitHub

Ce chapitre vous présente les bases de l'utilisation d'un système de contrôle de version pour garder une trace de tout votre code R important et faciliter la collaboration avec vos collègues et le reste du monde. Ce chapitre se concentre sur l'utilisation du logiciel "Git" en combinaison avec le service d'hébergement en ligne "GitHub". À la fin du chapitre, vous serez en mesure d'installer et de configurer Git et GitHub sur votre ordinateur et de mettre en place et de travailler avec un projet de contrôle de version dans RStudio. Nous n'aborderons pas en détail les sujets plus avancés tels que le branching, le forking et les pull requests, mais nous en donnerons un aperçu plus tard dans Section 7.8.

Juste quelques notes d'avertissement. Dans ce chapitre, nous utiliserons RStudio pour interfaçer avec Git car il offre une interface graphique conviviale qui rend la vie un peu plus facile (et qui ne veut pas ça ?). Cependant, l'un des inconvénients de l'utilisation de RStudio avec Git est que RStudio ne fournit que des fonctionnalités Git assez basiques à travers son système de menus. C'est très bien pour la plupart des choses que nous ferons dans ce chapitre (bien que nous introduirons quelques commandes Git au fur et à mesure), mais si vous voulez vraiment bénéficier de la puissance de Git, vous devrez apprendre quelques commandes Git (voir Section 7.10) et la syntaxe, et changer pour un autre IDE tel que [VSCode](#) qui est bien meilleur en ce qui concerne l'intégration avec GitHub. Git peut devenir un peu déroutant et frustrant quand on commence à l'utiliser. Cela est principalement dû à la terminologie et à l'utilisation abondante du jargon associé à Git, mais il est impossible de cacher le fait qu'il est assez facile de se mettre dans le pétrin avec son dépôt Git. C'est pourquoi nous avons essayé de faire en sorte que les choses soient aussi simples que possible dans ce chapitre et, par conséquent, nous vous montrerons de temps en temps quelques façons de faire très "non Git" (principalement en ce qui concerne le retour à des versions précédentes de documents). N'y voyez pas d'inconvénient, il n'y a pas de honte à utiliser ces solutions peu techniques et si elles fonctionnent, c'est qu'elles fonctionnent. Enfin, GitHub n'a pas été conçu pour héberger de très gros fichiers et vous avertira si vous essayez d'ajouter des fichiers de plus de 50 Mo et vous empêchera d'ajouter des fichiers de plus de 100 Mo.

Si votre projet implique l'utilisation de fichiers de grande taille, il existe quelques [quelques solutions](#) mais nous trouvons que la solution la plus simple est d'héberger ces fichiers ailleurs (Googledrive, Dropbox etc) et de créer un lien vers eux dans un fichier README ou un document R markdown sur Github.

7.1. Qu'est-ce que le contrôle de version ?

A [Système de contrôle des versions](#) (VCS) conserve un enregistrement de toutes les modifications que vous apportez aux fichiers qui composent un projet particulier et vous permet de revenir à des versions antérieures des fichiers si nécessaire. En d'autres termes, si vous vous trompez ou si vous perdez accidentellement des fichiers importants, vous pouvez facilement revenir à une étape antérieure de votre projet pour régler le problème. Le contrôle de version a été conçu à l'origine pour le développement collaboratif de logiciels, mais il est tout aussi utile pour la recherche scientifique et les collaborations (même s'il est vrai que la plupart des termes, du jargon et des fonctionnalités sont axés sur le développement de logiciels). Il existe actuellement de nombreux systèmes de contrôle de version différents, mais nous nous concentrerons sur l'utilisation de *Git* parce qu'il est gratuit et open source et qu'il s'intègre bien à RStudio. Cela signifie qu'il peut facilement faire partie de votre flux de travail habituel avec un minimum de frais supplémentaires.

7.2. Pourquoi utiliser le contrôle de version ?

Pourquoi devriez-vous vous préoccuper du contrôle des versions ? Tout d'abord, il permet d'éviter cette situation (familière ?) lorsque vous travaillez sur un projet qui découle généralement de ce scénario (familier ?).

Le contrôle de version se charge automatiquement de conserver une trace de toutes les versions d'un fichier particulier et vous permet de revenir aux versions précédentes si nécessaire. Le contrôle de version vous aide également (en particulier le futur vous) à garder une trace de tous vos fichiers en un seul endroit et il aide les autres (en particulier les collaborateurs) à revoir, contribuer et réutiliser votre travail via le site web GitHub. Enfin, vos fichiers sont toujours disponibles de n'importe où et sur n'importe quel ordinateur, tout ce dont vous avez besoin, c'est d'une connexion internet.

7.3. Qu'est-ce que Git et GitHub ?

Git est un système de contrôle de version développé à l'origine par [Linus Torvalds](#) qui permet de suivre les modifications apportées à un ensemble de fichiers. Ces fichiers peuvent être de n'importe quel type, y compris la

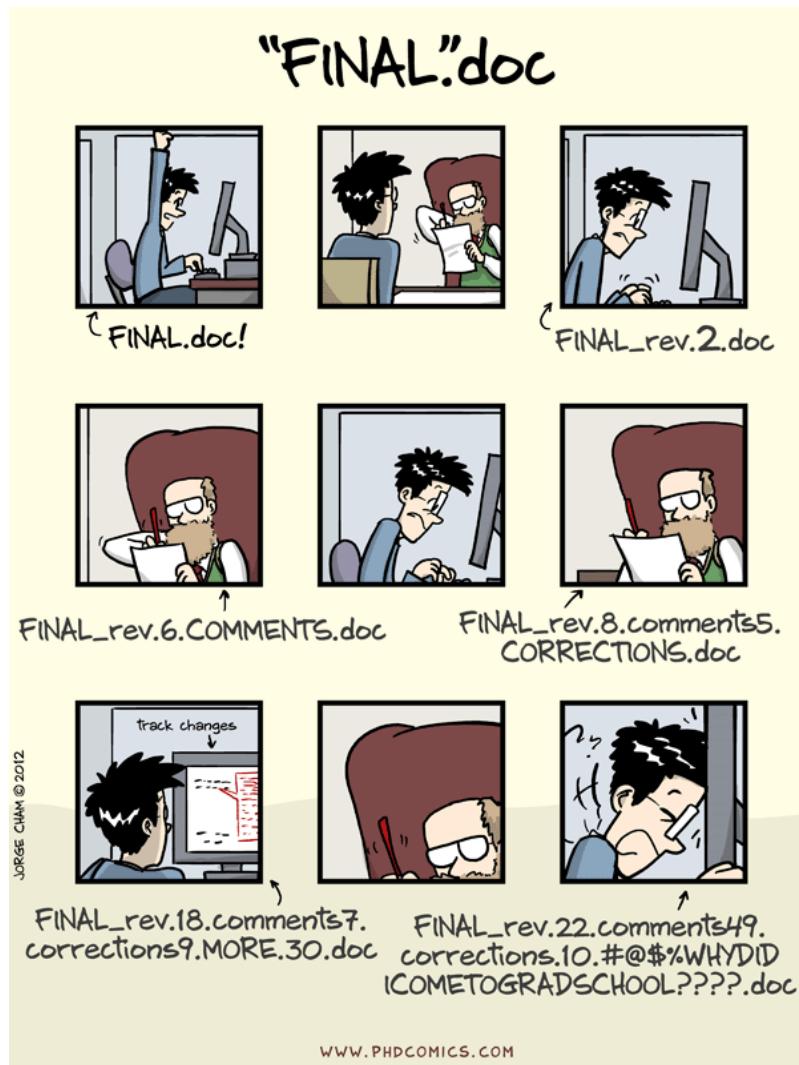


Figure 7.1.: Why you need version control (source: [PhDComics](http://www.phdcomics.com))

ménagerie de fichiers qui composent généralement un projet axé sur les données (.pdf, .Rmd, .docx, .txt, .jpg, etc.), bien que les fichiers texte simples soient ceux qui fonctionnent le mieux. L'ensemble des fichiers qui composent un projet s'appelle un **référentiel** (ou simplement **repo**).

GitHub est un service d'hébergement de dépôts Git basé sur le web qui vous permet de créer une copie distante de votre projet local contrôlé par version. Cette copie peut servir de sauvegarde ou d'archive de votre projet ou vous permettre, ainsi qu'à vos collègues, d'y accéder pour travailler en collaboration.

Au début d'un projet, nous créons généralement (mais pas toujours) un dépôt Git **distant** sur GitHub, puis **cloner** (il s'agit d'une copie) de ce dépôt dans notre base de données **local** local (celui qui se trouve devant vous). Ce clonage est généralement un événement unique et vous ne devriez pas avoir besoin de cloner ce dépôt à nouveau, à moins que vous ne fassiez vraiment n'importe quoi. Une fois que vous avez cloné votre dépôt, vous pouvez travailler localement sur votre projet comme d'habitude, en créant et en sauvegardant des fichiers pour votre analyse de données (scripts, documents R markdown, figures, etc.). En cours de route, vous pouvez prendre des instantanés (appelés **commits**) de ces fichiers après y avoir apporté des modifications importantes. Nous pouvons alors **pousser** ces modifications vers le dépôt GitHub distant pour en faire une sauvegarde ou les mettre à la disposition de nos collaborateurs. Si d'autres personnes travaillent sur le même projet (**dépôt**), ou si vous travaillez sur un autre ordinateur, vous pouvez **tirer** les modifications vers votre dépôt local afin que tout soit synchronisé.

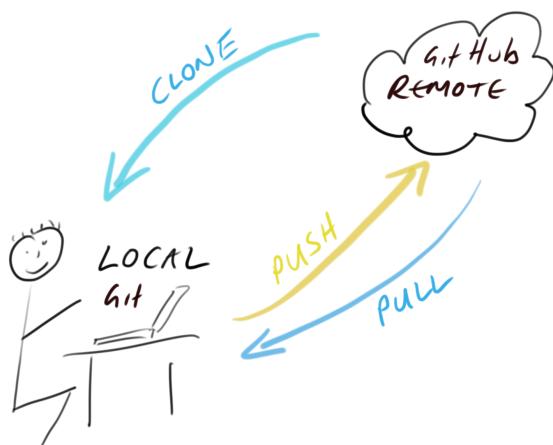


Figure 7.2.: How git works
236

7.4. Pour commencer

Ce chapitre suppose que vous avez déjà installé les dernières versions de R et un IDE (RStudio ou VSCode). Si vous ne l'avez pas encore fait, vous trouverez des instructions dans Section 1.1.1.

7.4.1. Installer Git

Pour commencer, vous devez d'abord installer Git. Si vous avez de la chance, vous avez peut-être déjà installé Git (surtout si vous avez un ordinateur Mac ou Linux). Vous pouvez vérifier si vous avez déjà installé Git en cliquant sur le bouton Terminal tab dans RStudio et en tapant `git --version`. Si vous voyez quelque chose qui ressemble à `git version 2.25.0` (le numéro de version peut être différent sur votre ordinateur), alors vous avez déjà installé Git (jours heureux). Si vous obtenez une erreur (quelque chose comme `git: command not found`), cela signifie que Git n'est pas (encore) installé.

Vous pouvez également faire cette vérification en dehors de RStudio en ouvrant un terminal séparé si vous le souhaitez. Sous Windows, allez dans le menu “Démarrer” et dans la barre de recherche (ou la boîte d'exécution) tapez cmd et appuyez sur la touche Entrée. Sur Mac, allez dans “Applications” dans le Finder, cliquez sur le dossier “Utilitaires”, puis sur le programme “Terminal”. Sur une machine Linux, ouvrez simplement le Terminal (Ctrl+Alt+T fait souvent l'affaire).

Pour installer Git sur un **Windows** nous vous recommandons de télécharger et d'installer Git pour Windows (également connu sous le nom de “Git Bash”). Vous trouverez le fichier de téléchargement et les instructions d'installation [ici](#).

Pour ceux qui utilisent un **Mac** nous vous recommandons de télécharger Git à partir de [ici](#) et de l'installer de la manière habituelle (double-cliquez sur le paquet d'installation une fois téléchargé). Si vous avez déjà installé Xcode sur votre Mac et que vous souhaitez utiliser une version plus récente de Git, vous devrez suivre quelques étapes supplémentaires documentées [ici](#). Si vous n'avez jamais entendu parler de Xcode, ne vous inquiétez pas !

Pour ceux d'entre vous qui ont la chance de travailler sur un **Linux** vous pouvez simplement utiliser le gestionnaire de paquets de votre système d'exploitation pour installer Git à partir du dépôt officiel. Pour Ubuntu Linux (ou ses variantes), ouvrez votre Terminal et tapez

```
sudo apt update
sudo apt install git
```

Vous aurez besoin de priviléges administratifs pour effectuer cette opération. Pour les autres versions de Linux, voir [ici](#) pour de plus amples instructions d'installation.

Quelle que soit la version de Git que vous installez, une fois l'installation terminée, vérifiez que le processus d'installation s'est bien déroulé en exécutant la commande `git --version` dans l'onglet Terminal de RStudio (comme décrit ci-dessus). Sur certaines installations de Git (oui, nous vous regardons MS Windows), cela peut encore produire une erreur car vous devrez également configurer RStudio pour qu'il puisse trouver l'exécutable Git (décrit dans Section 7.4.3).

7.4.2. Configurer Git

Après avoir installé Git, vous devez le configurer pour pouvoir l'utiliser. Cliquez à nouveau sur l'onglet Terminal dans la fenêtre Console et tapez ce qui suit :

```
git config --global user.email 'you@youremail.com'
```

```
git config --global user.name 'Your Name'
```

en remplaçant 'Your Name' votre nom réel et 'you@youremail.com' par votre adresse électronique. Nous vous recommandons d'utiliser l'adresse électronique de votre université (si vous en avez une), car vous l'utiliserez également lors de l'ouverture de votre compte GitHub (voir plus loin).

Si vous avez réussi, vous ne devriez pas voir de message d'erreur dans ces commandes. Pour vérifier que vous avez configuré Git avec succès, tapez ce qui suit dans le Terminal

```
git config --global --list
```

Vous devriez voir vos deux `user.name` et `user.email` configurés.

7.4.3. Configurer RStudio

Comme vous pouvez le voir ci-dessus, Git peut être utilisé à partir de la ligne de commande, mais il s'intègre également bien à RStudio, en fournissant une interface utilisateur graphique conviviale. Si vous souhaitez utiliser l'intégration Git de RStudio (nous vous le recommandons, au moins au début), vous devez vérifier que le chemin d'accès à l'exécutable Git est correctement spécifié. Dans RStudio, allez dans le menu Tools -> Global Options -> Git/SVN et assurez-vous que 'Enable version control interface for RStudio projects' est coché et que le chemin 'Git executable:' est correct pour votre installation. Si ce n'est pas le cas, cliquez sur le bouton Browse... et naviguez jusqu'à l'endroit où vous avez installé git et cliquez sur le fichier exécutable. Vous devrez redémarrer RStudio après cette opération.

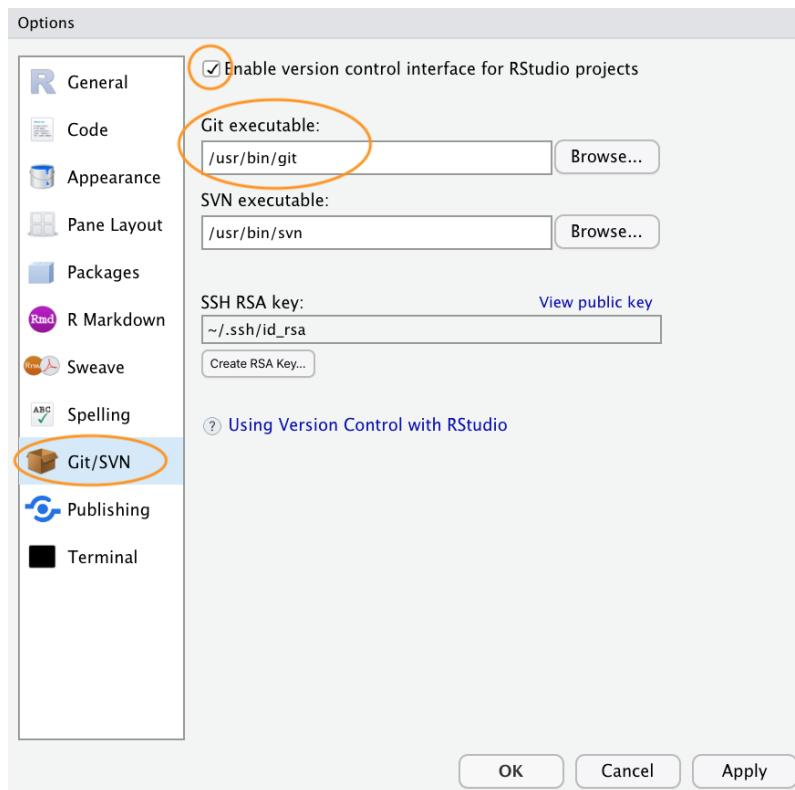


Figure 7.3.: Providing path to git software in RStudio

7.4.4. Configurer VSCode

pour développer

7.4.5. Créer un compte GitHub

Si tout ce que vous voulez, c'est garder une trace des fichiers et de leurs versions sur votre ordinateur local, alors Git est suffisant. En revanche, si vous souhaitez faire une copie hors site de votre projet ou le mettre à la disposition de vos collaborateurs, vous aurez besoin d'un service d'hébergement en ligne pour vos dépôts Git. C'est là que GitHub entre en jeu (il existe également d'autres services tels que [GitLab](#), [Bitbucket](#) et [Savannah](#)). Vous pouvez vous inscrire pour un compte gratuit sur GitHub [ici](#). Vous devrez spécifier un nom d'utilisateur, une adresse email et un mot de passe fort. Nous vous suggérons d'utiliser l'adresse électronique de votre université (si vous en avez une), car elle vous permettra également de demander un compte gratuit d'éducateur ou de chercheur. [gratuit pour les éducateurs ou les chercheurs](#) ce qui vous permettra de bénéficier d'un certain nombre d'avantages [avantages](#) (ne vous en préoccupez pas pour l'instant). Quand il s'agit de choisir un nom d'utilisateur, nous vous suggérons d'y réfléchir. Choisissez un nom d'utilisateur court plutôt que long, utilisez des minuscules et des traits d'union si vous voulez inclure plusieurs mots, trouvez un moyen d'incorporer votre nom réel et, enfin, choisissez un nom d'utilisateur que

vous vous sentirez à l'aise de révéler à votre futur employeur !

Cliquez ensuite sur “Sélectionnez un plan” (il se peut que vous deviez d'abord résoudre une énigme simple pour vérifier que vous êtes un être humain) et choisissez l'option “Plan gratuit”. Github vous enverra un courriel à l'adresse que vous avez fournie pour que vous puissiez vérifier.

Une fois que vous avez terminé toutes ces étapes, vous devriez avoir installé Git et GitHub, prêts à l'emploi (enfin !).

7.5. Mise en place d'un projet

7.5.1. dans RStudio

Maintenant que tout est prêt, créons notre premier projet RStudio à version contrôlée. Pour ce faire, il existe plusieurs approches différentes. Vous pouvez d'abord créer un dépôt GitHub distant, puis connecter un projet RStudio à ce dépôt (c'est ce que nous appellerons l'option 1). Une autre option consiste à créer d'abord un dépôt local, puis à lier un dépôt GitHub distant à ce dépôt (Option 2). Vous pouvez également connecter un projet existant à un dépôt GitHub, mais nous n'aborderons pas cette option ici. Si vous êtes totalement novice en matière de Git et de GitHub, nous vous suggérons d'utiliser l'option 1, car cette approche met en place votre dépôt Git local de manière satisfaisante et vous pouvez **pousser** et **tirer** immédiatement. L'option 2 nécessite un peu plus de travail et offre donc plus de possibilités de se tromper. Nous aborderons ces deux options ci-dessous.

7.5.2. Option 1 - GitHub d'abord

Pour utiliser l'approche GitHub first, vous devez d'abord créer un fichier **dépôt (repo)** sur GitHub. Accédez à votre [page GitHub](#) et connectez-vous si nécessaire. Cliquez sur l'onglet “Dépôts” en haut et ensuite sur le bouton vert “Nouveau” à droite.

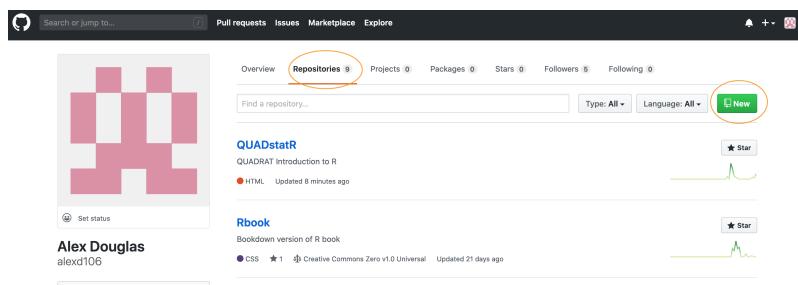


Figure 7.4.: Creating a new repository on Github

Donnez un nom à votre nouveau dépôt (appelons-le `first_repo` pour ce chapitre), sélectionnez ‘Public’, cochez la case ‘Initialize this repository with a README’ (c’est important) et cliquez sur ‘Create repository’ (ignorez les autres options pour l’instant).

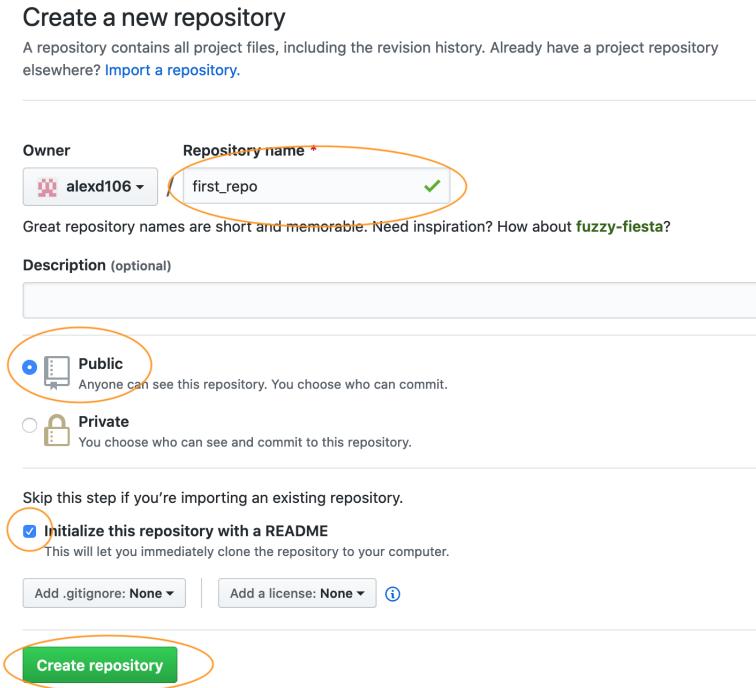


Figure 7.5.: Configuring a new repository on Github

Votre nouveau dépôt GitHub est maintenant créé. Notez que le README a été rendu dans GitHub et qu'il est au format markdown (.md) (voir Chapitre 6 sur R markdown si cela ne vous dit rien). Cliquez ensuite sur le bouton vert “Cloner ou télécharger” et copiez le fichier <https://...> URL qui s'affiche pour plus tard (mettez tout en surbrillance et copiez ou cliquez sur l'icône de copie dans le presse-papiers à droite).

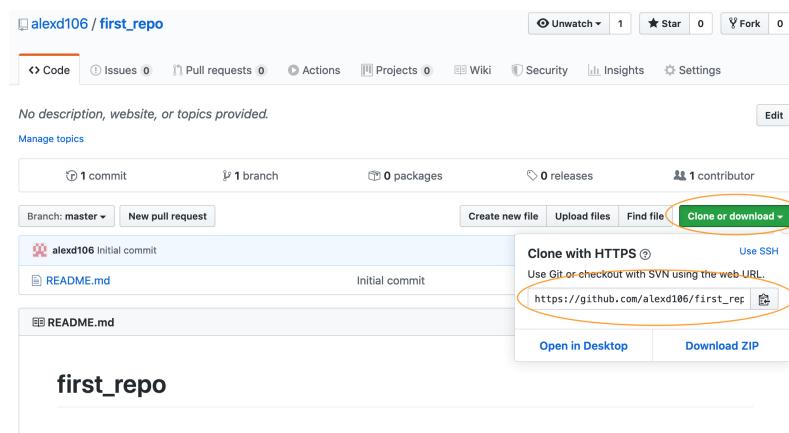


Figure 7.6.: Getting the cloning path for a directory on github

Ok, nous allons maintenant nous intéresser à RStudio. Dans RStudio, cliquez sur le bouton File -> New Project

le menu Dans la fenêtre qui s'ouvre, sélectionnez Version Control.

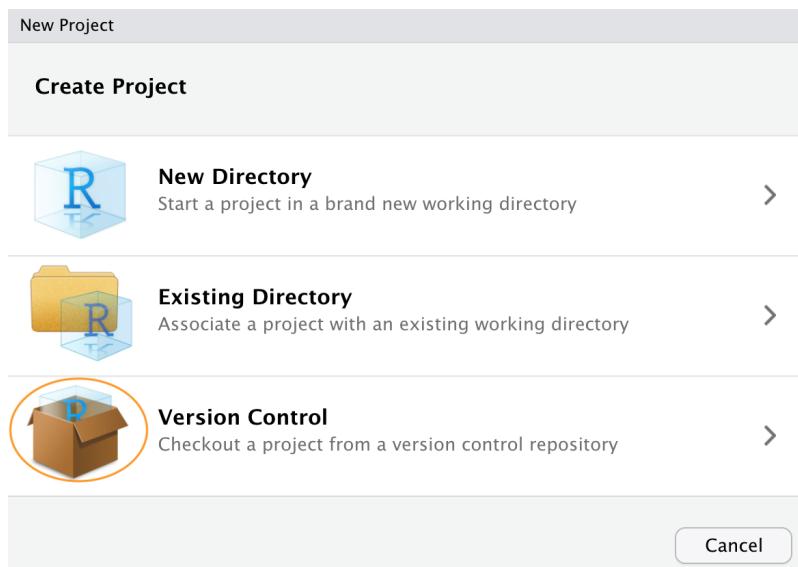


Figure 7.7.: Setting a new Github project in RStudio

Collez maintenant l'URL que vous avez précédemment copiée de GitHub dans le champ Repository URL: dans la boîte de dialogue. Cela devrait remplir automatiquement le champ Project Directory Name: avec le nom correct du dépôt (il est important que le nom de ce répertoire soit le même que celui du dépôt que vous avez créé sur GitHub). Vous pouvez ensuite sélectionner l'endroit où vous souhaitez créer ce répertoire en cliquant sur le bouton Browse en face du bouton Create project as a subdirectory of: en face de l'option Naviguez jusqu'à l'endroit où vous souhaitez créer le répertoire et cliquez sur OK. Nous cochons également l'option Open in new session option.

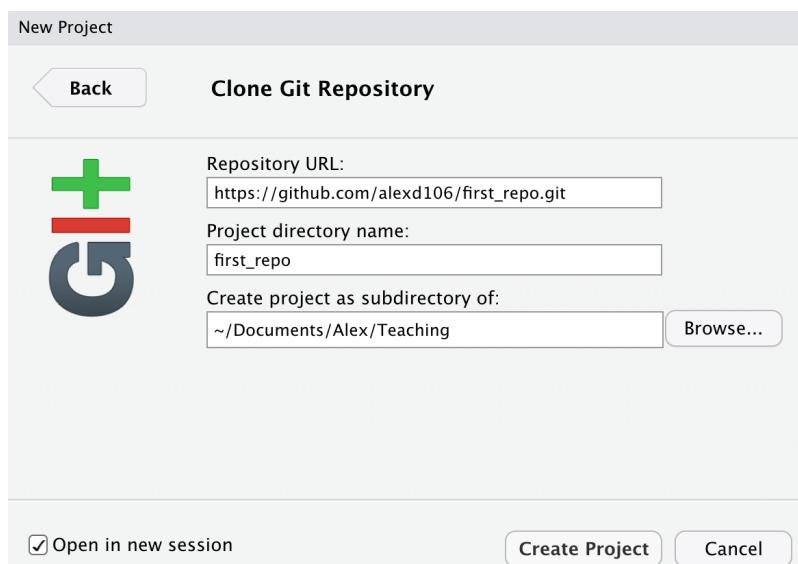


Figure 7.8.

RStudio va maintenant créer un nouveau répertoire portant le même nom que votre référentiel sur votre ordinateur local et va ensuite **cloner** votre référentiel distant dans ce répertoire. Le répertoire contiendra trois nouveaux fichiers ; `first_repo.Rproj` (ou quel que soit le nom que vous avez donné à votre référentiel), `README.md` et `.gitignore`. Vous pouvez vérifier cela dans le **Files** qui se trouve généralement dans le volet inférieur droit de RStudio. Vous disposerez également d'un **Git** dans le volet supérieur droit avec deux fichiers listés (nous y reviendrons plus tard dans le chapitre). C'est tout pour l'option 1, vous avez maintenant un dépôt GitHub distant qui est lié à votre dépôt local géré par RStudio. Toutes les modifications que vous apportez aux fichiers de ce répertoire seront contrôlées par Git.

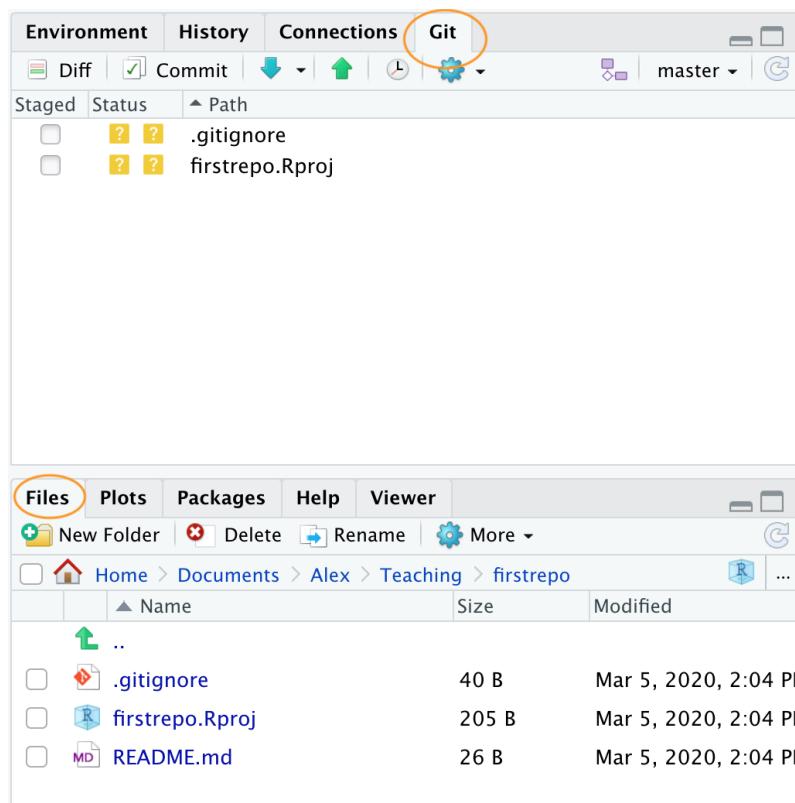


Figure 7.9.

7.5.3. Option 2 - RStudio d'abord

Une autre approche consiste à créer d'abord un projet RStudio local, puis à établir un lien avec un dépôt Github distant. Comme nous l'avons mentionné précédemment, cette option est plus complexe que l'option 1. N'hésitez donc pas à sauter cette étape et à y revenir plus tard si vous êtes intéressé. Cette option est également utile si vous souhaitez simplement créer un projet RStudio local lié à un dépôt Git local (i.e. GitHub n'est pas impliqué). Dans ce cas, suivez les instructions ci-dessous en omettant la partie GitHub.

Dans RStudio, cliquez sur le bouton **File -> New Project** et sélectionnez l'option **New Directory** option.

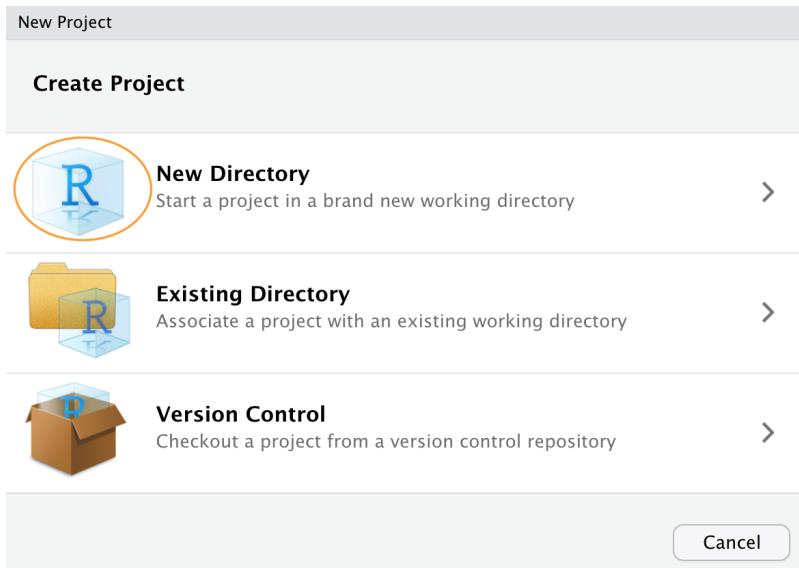


Figure 7.10.

Dans la fenêtre qui s'ouvre, sélectionnez l'option New Project l'option

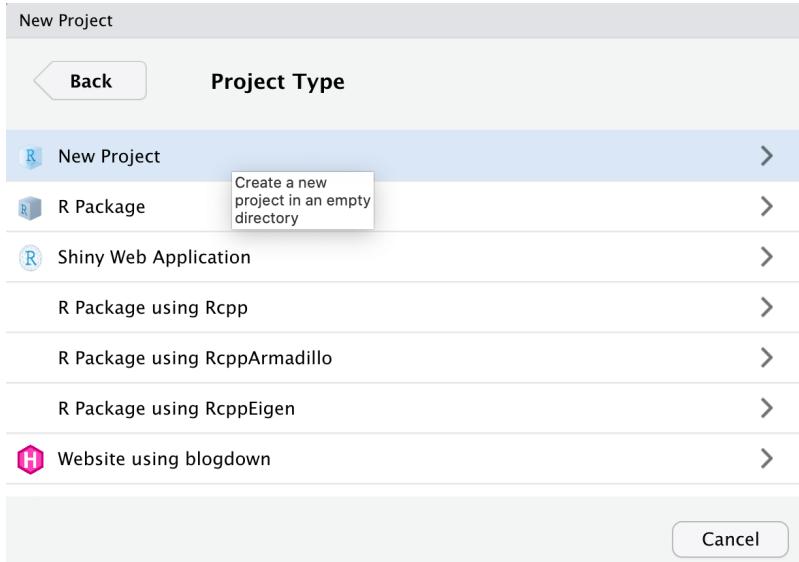


Figure 7.11.

Dans la fenêtre Nouveau projet, spécifiez un `Directory name` (choisissez `second_repo` pour ce chapitre) et sélectionnez l'endroit où vous souhaitez créer ce répertoire sur votre ordinateur (cliquez sur le bouton `Browse` (cliquez sur le bouton `).`). Assurez-vous que le répertoire `Create a git repository` est cochée

Cela créera un répertoire à version contrôlée appelé `second_repo` sur votre ordinateur qui contient deux fichiers, `second_repo.Rproj` et `.gitignore` (il peut également y avoir un fichier `.Rhistory` mais n'en tenez pas compte). Vous pouvez le vérifier en consultant le fichier `Files` dans RStudio (généralement dans le volet inférieur droit).

OK, avant de créer un dépôt sur GitHub, nous devons faire une dernière chose - nous devons placer notre fichier

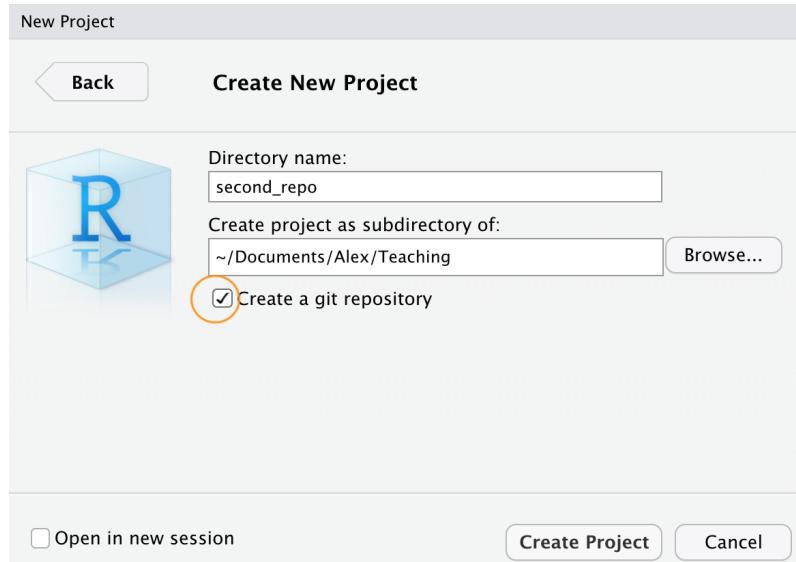


Figure 7.12.

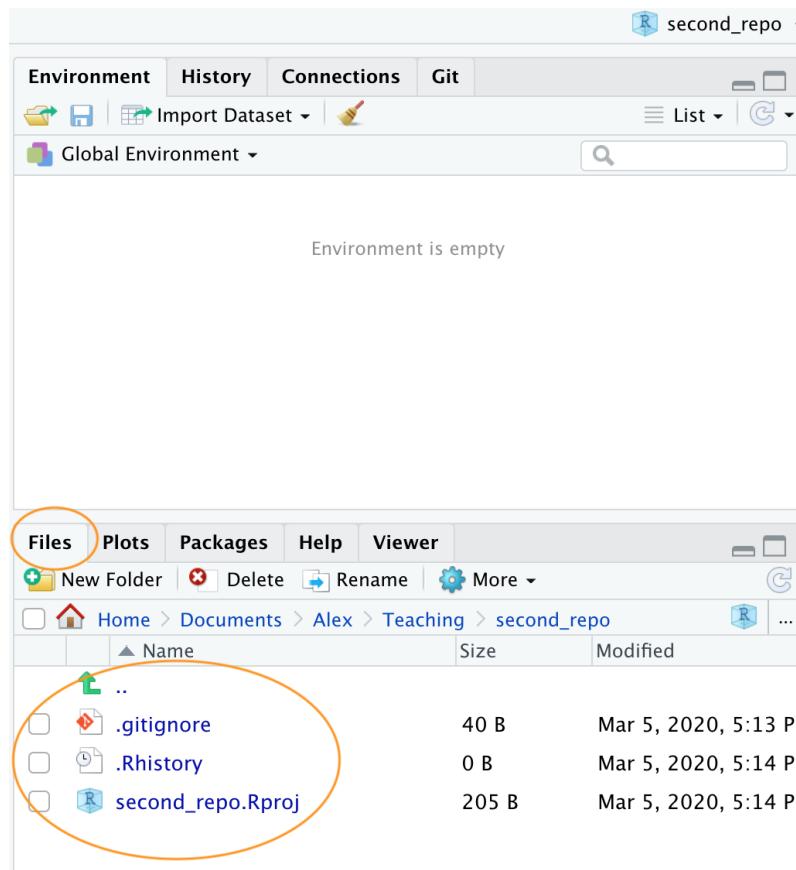


Figure 7.13.

`second_repo.Rproj` et `.gitignore` sous contrôle de version. Malheureusement, nous n'avons pas encore abordé ce sujet en détail, alors suivez les quelques instructions suivantes (à l'aveugle !) et nous y reviendrons dans Section 7.6 de ce chapitre.

Pour placer nos deux fichiers sous contrôle de version, cliquez sur l'onglet “Git” qui se trouve généralement dans le panneau supérieur de RStudio.

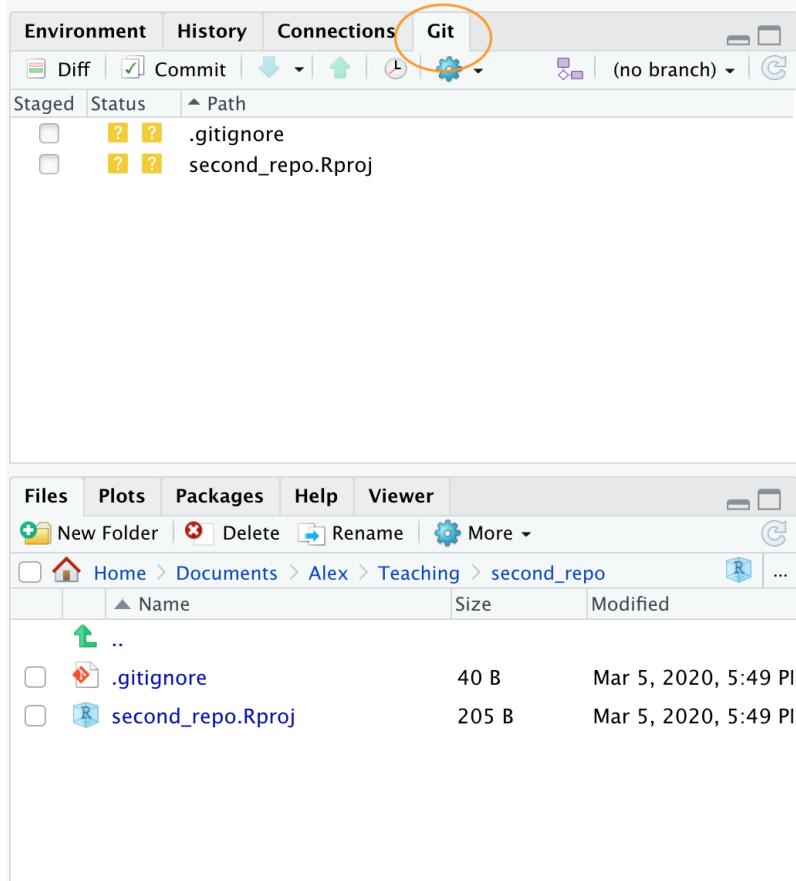


Figure 7.14.

Vous pouvez voir que les deux fichiers sont listés. Ensuite, cochez les cases de la colonne “Staged” pour les deux fichiers et cliquez sur le bouton “Commit”.

Vous accédez alors à la fenêtre “Examiner les modifications”. Saisissez le message de validation “Première validation” dans la fenêtre “Message de validation” et cliquez sur le bouton “Valider”. Une nouvelle fenêtre apparaît avec des messages que vous pouvez ignorer pour l'instant. Cliquez sur “Fermer” pour fermer cette fenêtre ainsi que la fenêtre “Examiner les modifications”. Les deux fichiers devraient maintenant avoir disparu du panneau Git dans RStudio, ce qui indique que la validation a été effectuée avec succès.

OK, ces deux fichiers sont maintenant sous contrôle de version. Nous devons maintenant créer un nouveau dépôt sur GitHub. Dans votre navigateur, allez sur votre [page GitHub](#) et connectez-vous si nécessaire. Cliquez sur l'onglet

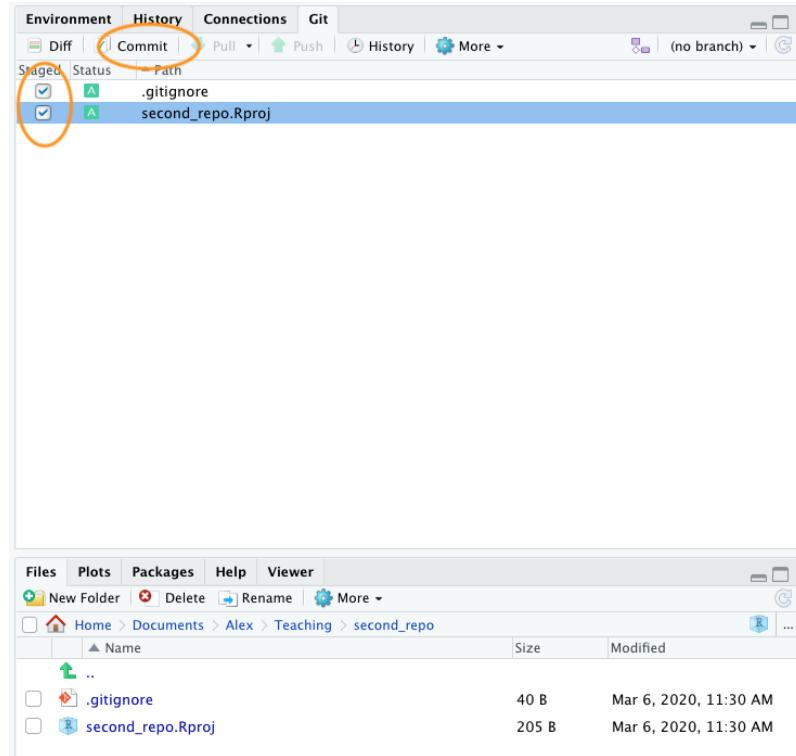


Figure 7.15.

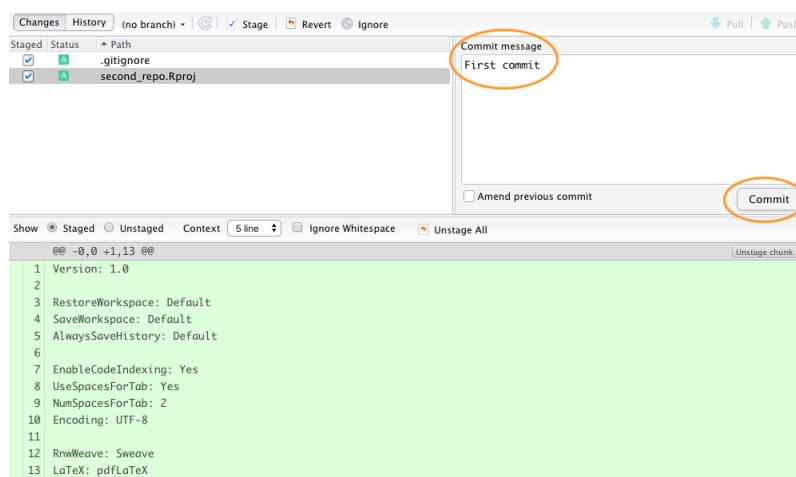


Figure 7.16.

“Dépôts” et cliquez sur le bouton vert “Nouveau” à droite. Donnez à votre nouveau dépôt le nom `second_repo` (identique au nom de votre répertoire de contrôle de version) et sélectionnez “Public”. Cette fois-ci **ne pas** cocher la case ‘Initialize this repository with a README’ (c’est important) et cliquer sur ‘Create repository’.

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository](#).

Owner Repository name *

alexd106 / `second_repo`

Great repository names are short and memorable. Need inspiration? How about [fictional-octo-eureka](#)?

Description (optional)

My second repository

Public Anyone can see this repository. You choose who can commit.

Private You choose who can see and commit to this repository.

Skip this step if you’re importing an existing repository.

Initialize this repository with a README This will let you immediately clone the repository to your computer.

Add .gitignore: None Add a license: None ⓘ

Create repository

Figure 7.17.

Cela vous amènera à une page de configuration rapide qui vous fournira du code pour différentes situations. Le code qui nous intéresse est celui qui se trouve sous `...or push an existing repository from the command line` l’en-tête.

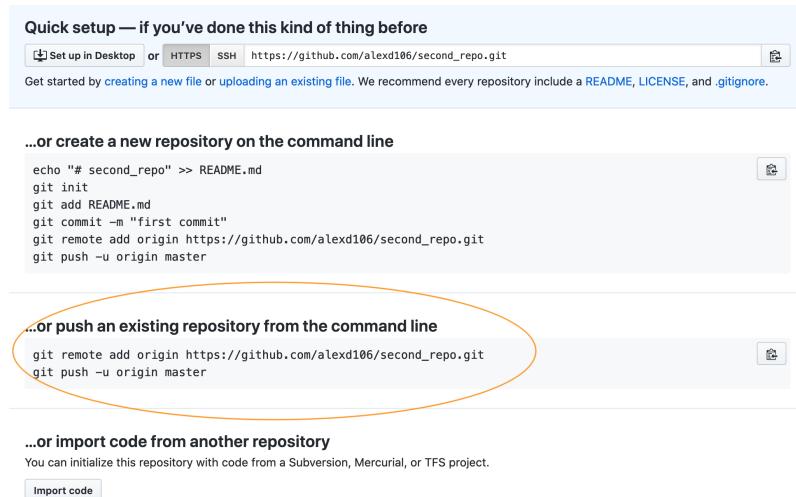


Figure 7.18.

Surlignez et copiez la première ligne de code (note : la vôtre sera légèrement différente car elle inclura votre nom

d'utilisateur GitHub et non le mien).

```
git remote add origin https://github.com/alexd106/second_repo.git
```

Passez à RStudio, cliquez sur l'onglet "Terminal" et collez la commande dans le terminal. Retournez ensuite sur GitHub et copiez la deuxième ligne de code

```
git push -u origin master
```

et collez-la dans le terminal de RStudio. Vous devriez voir quelque chose comme ceci

```
Console Terminal x Jobs x
Terminal 1 ~ /Documents/Alex/Teaching/second_repo
md-051770:second_repo nhy163$ git remote add origin https://github.com/alexd106/second_repo.git
md-051770:second_repo nhy163$ git push -u origin master
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 436 bytes | 218.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0)
To https://github.com/alexd106/second_repo.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
md-051770:second_repo nhy163$
```

Figure 7.19.

Si vous jetez un coup d'œil à votre repo sur GitHub (cliquez sur l'icône /second_repo en haut de la page), vous verrez le `second_repo.Rproj` et `.gitignore` ont été remplacés par les fichiers **poussés** sur GitHub depuis votre dépôt local.

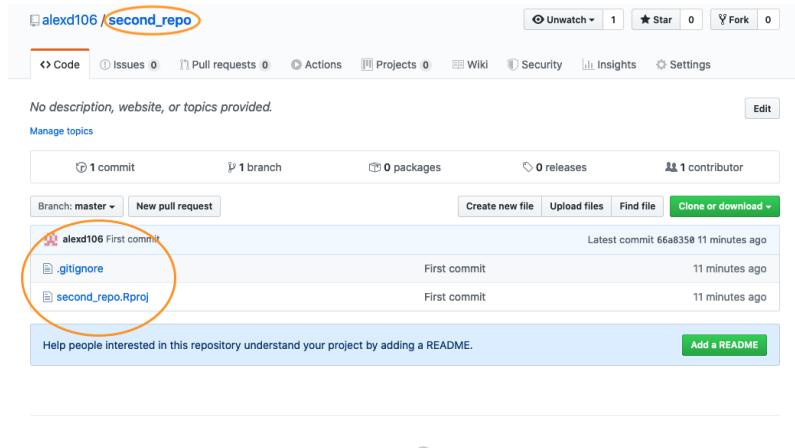


Figure 7.20.

La dernière chose à faire est de créer et d'ajouter un fichier **README** à votre dépôt. Un fichier README décrit votre projet et est écrit en utilisant le même langage Markdown que vous avez appris dans Chapitre 6. Un bon fichier README permet aux autres (ou au futur vous !) d'utiliser votre code et de reproduire votre projet. Vous pouvez créer un fichier README dans RStudio ou dans GitHub. Utilisons la seconde option.

Dans votre dépôt sur GitHub, cliquez sur le bouton vert Add a README vert.

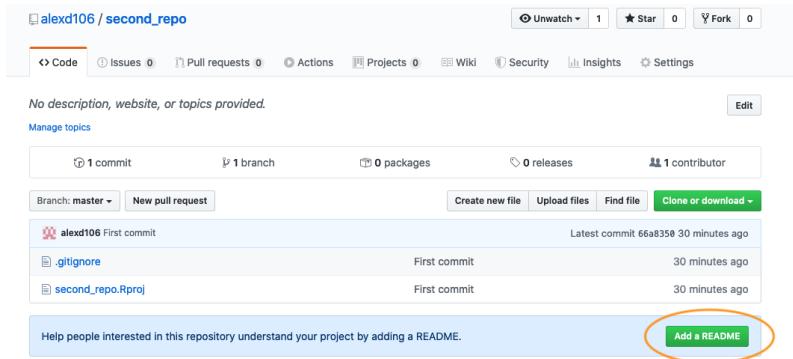


Figure 7.21.

Rédigez maintenant une brève description de votre projet dans la rubrique <> Edit new file puis cliquez sur le bouton vert Commit new file vert.

Vous devriez maintenant voir l'écran README.md dans votre référentiel. Il n'existera pas encore sur votre ordinateur car vous devrez tirer ces changements dans votre dépôt local, mais nous y reviendrons dans la section suivante.

Que vous ayez suivi l'option 1 ou l'option 2 (ou les deux), vous avez maintenant configuré avec succès un projet RStudio à version contrôlée (et un répertoire associé) et l'avez lié à un dépôt GitHub. Git va maintenant surveiller ce répertoire pour toutes les modifications que vous apportez aux fichiers et aussi si vous ajoutez ou supprimez des fichiers. Si les étapes ci-dessus vous semblent un peu difficiles, rappelez-vous que vous n'avez à le faire qu'une seule fois pour chaque projet et que cela devient de plus en plus facile avec le temps.

7.5.4. dans VSCode

pour développer

7.6. Utiliser Git avec RStudio

Maintenant que nous avons mis en place notre projet et nos dépôts (locaux et distants), il est enfin temps d'apprendre à utiliser Git dans votre IDE !

Typiquement, lorsque vous utilisez Git, votre flux de travail se déroule comme suit :

1. Vous créez/supprimez et modifiez les fichiers dans le répertoire de votre projet sur votre ordinateur comme d'habitude (en sauvegardant les modifications au fur et à mesure).

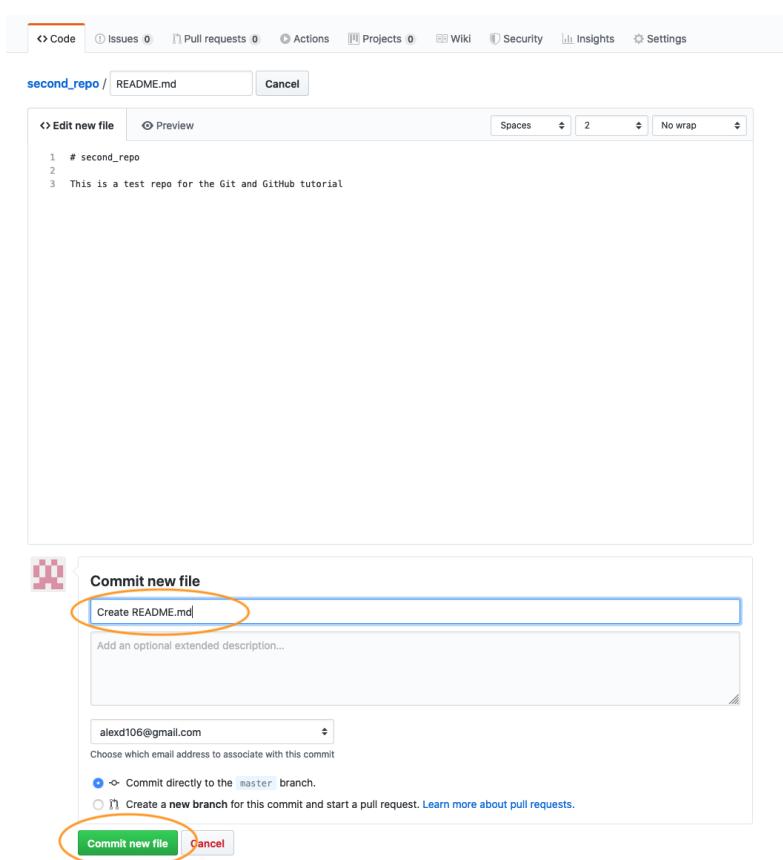


Figure 7.22.

2. Une fois que vous avez atteint un “point d’arrêt” naturel dans votre progression (c.-à-d. vous seriez triste si vous perdiez ce progrès), vous **étape** ces fichiers
3. Ensuite, vous **engager** les modifications que vous avez apportées à ces fichiers mis en scène (avec un message de validation utile), ce qui crée un instantané permanent de ces modifications.
4. Vous continuez ce cycle jusqu’à ce que vous arriviez à un point où vous souhaitez **pousser** ces changements sur GitHub
5. Si vous travaillez avec d’autres personnes sur le même projet, vous pouvez également avoir besoin de **tirer** leurs modifications sur votre ordinateur local

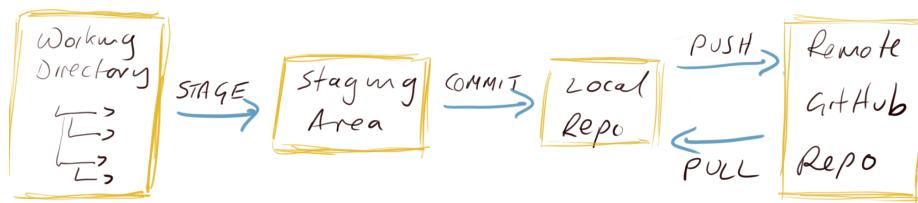


Figure 7.23.

Prenons un exemple pour clarifier ce flux de travail.

Ouvrez le `first_repo.Rproj` que vous avez créé précédemment dans l’option 1. Utilisez soit l’outil File -> Open Project ou cliquer sur l’icône de projet en haut à droite et sélectionner le projet approprié.

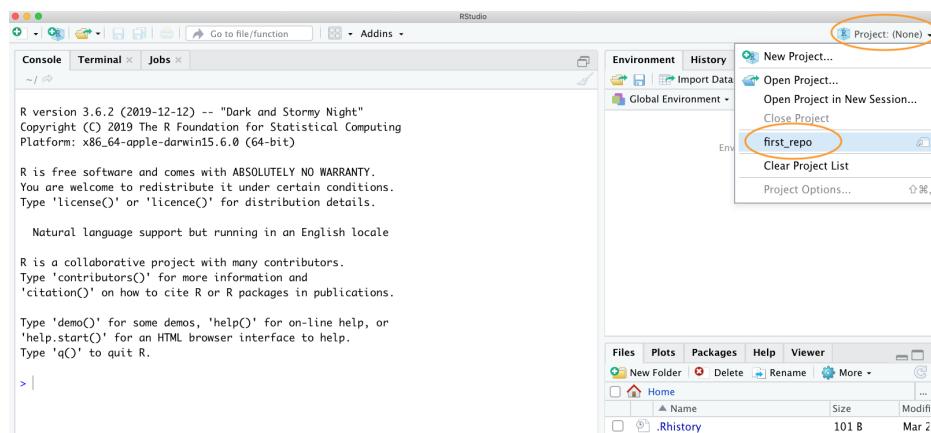
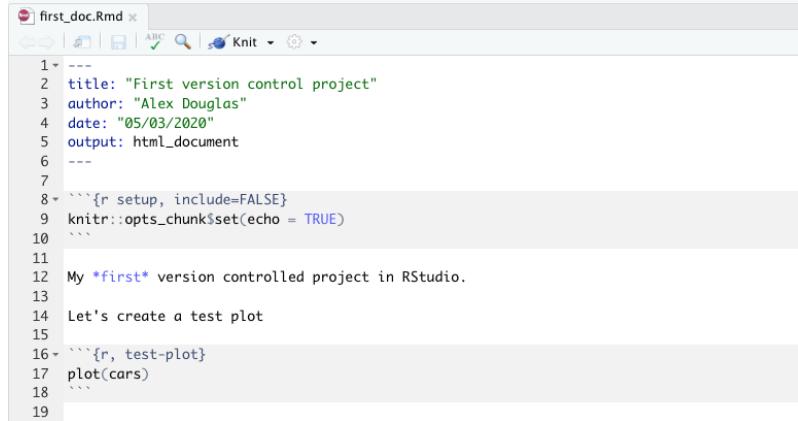


Figure 7.24.

Créez un document R markdown à l’intérieur de ce projet en cliquant sur l’icône File -> New File -> R markdown menu (vous vous souvenez de Chapitre 6 ?).

Une fois créé, nous pouvons supprimer tout le code R markdown de l'exemple (à l'exception de l'en-tête YAML) comme d'habitude et écrire du texte R markdown intéressant et inclure un tracé. Nous utiliserons la fonction `cars` pour ce faire. Enregistrez ce fichier (cmd + s pour Mac ou ctrl + s sous Windows). Votre document R markdown devrait ressembler à ce qui suit (ce n'est pas grave si ce n'est pas exactement la même chose).



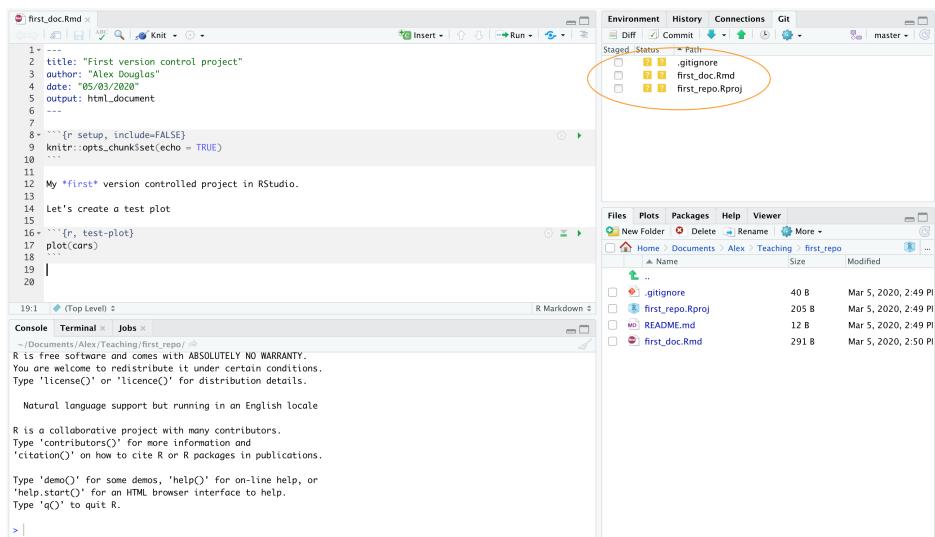
```

1 -> ---
2   title: "First version control project"
3   author: "Alex Douglas"
4   date: "05/03/2020"
5   output: html_document
6 ---
7
8 -> ```{r setup, include=FALSE}
9 knitr::opts_chunk$set(echo = TRUE)
10 ```
11
12 My *first* version controlled project in RStudio.
13
14 Let's create a test plot
15
16 -> ```{r, test-plot}
17 plot(cars)
18 ```
19
20

```

Figure 7.25.

Jetez un coup d'œil à l'onglet ‘Git’ qui devrait contenir votre nouveau document R markdown (`first_doc.Rmd` dans cet exemple) ainsi que `first_repo.Rproj` et `.gitignore` (vous avez créé ces fichiers précédemment en suivant l'option 1).



The Git panel shows the following staged changes:

- `.gitignore`
- `first_doc.Rmd`
- `first_repo.Rproj`

Figure 7.26.

En suivant notre flux de travail, nous devons maintenant **mettre en scène** ces fichiers. Pour ce faire, cochez les cases de la colonne “Etagé” pour tous les fichiers. Notez qu'une icône d'état se trouve à côté de la case, ce qui vous donne une indication sur la façon dont les fichiers ont été modifiés. Dans notre cas, tous les fichiers doivent être ajoutés (A majuscule) car nous venons de les créer.

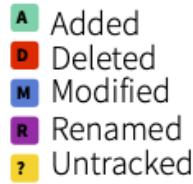


Figure 7.27.

Après avoir mis en scène les fichiers, l'étape suivante consiste à **engager** les fichiers. Pour ce faire, cliquez sur le bouton “Commit”.

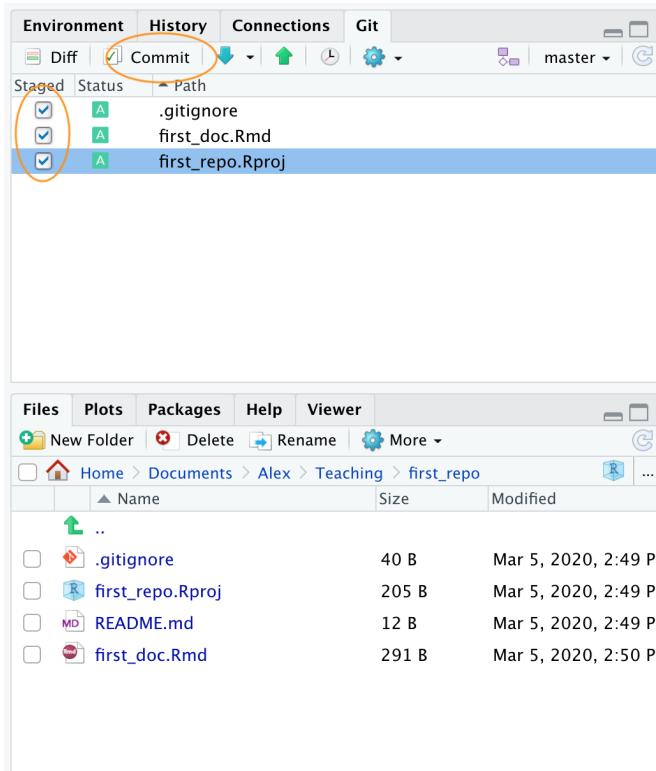


Figure 7.28.

Après avoir cliqué sur le bouton “Valider”, vous accéderez à la fenêtre “Revoir les modifications”. Vous devriez voir les trois fichiers que vous avez mis en scène à l’étape précédente dans le panneau de gauche. Si vous cliquez sur le nom du fichier `first_doc.Rmd` vous verrez les modifications que vous avez apportées à ce fichier en surbrillance dans le volet inférieur. Le contenu que vous avez ajouté est surligné en vert et le contenu supprimé est surligné en rouge. Comme vous venez de créer ce fichier, tout le contenu est surligné en vert. Pour valider ces fichiers (prendre un instantané), saisissez d’abord un message de validation obligatoire dans le champ “Message de validation”. Ce message doit être relativement court et informatif (pour vous et vos collaborateurs) et indiquer pourquoi vous avez effectué les modifications, et non ce que vous avez modifié. Ceci est logique car Git garde une trace de *ce que* a changé et qu’il est donc préférable de ne pas utiliser les messages de livraison à cette fin. Il est traditionnel de saisir

le message “First commit” (ou “Initial commit”) lorsque vous livrez des fichiers pour la première fois. Cliquez maintenant sur le bouton “Commit” pour valider ces modifications.

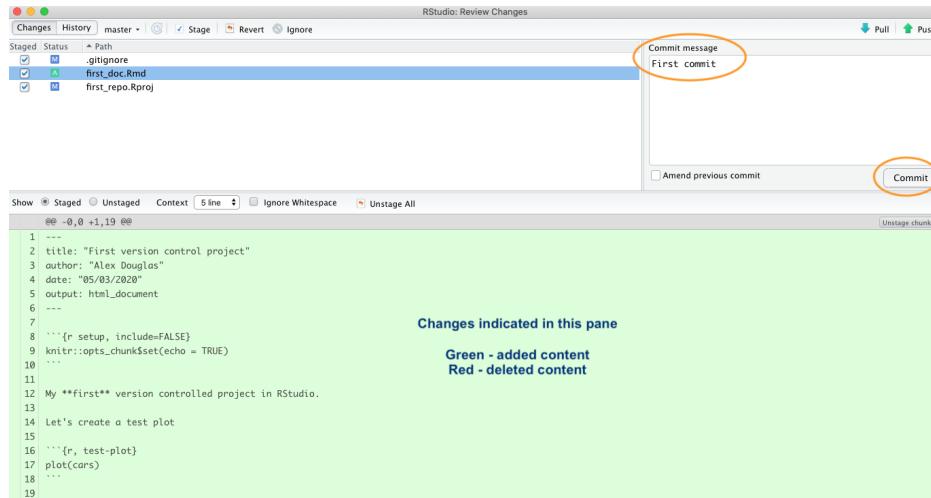


Figure 7.29.

Un résumé de la validation que vous venez d'effectuer s'affiche. Cliquez ensuite sur le bouton “Fermer” pour revenir à la fenêtre “Revoir les modifications”. Notez que les fichiers mis à disposition ont été supprimés.



Figure 7.30.

Maintenant que vous avez validé vos modifications, l'étape suivante consiste à **pousser** ces changements sur GitHub. Avant de pousser vos modifications, il est conseillé de commencer par **tirer** les modifications depuis GitHub. Ceci est particulièrement important si vous et vos collaborateurs travaillez sur les mêmes fichiers, car cela permet de garder votre copie locale à jour et d'éviter tout conflit potentiel. Dans ce cas, votre dépôt sera déjà à jour, mais c'est une bonne habitude à prendre. Pour ce faire, cliquez sur le bouton “Tirer” en haut à droite de la fenêtre “Examiner les modifications”. Une fois que vous avez retiré les modifications, cliquez sur le bouton vert “Pousser” pour transférer vos modifications. Vous verrez un résumé de l'opération que vous venez d'effectuer. Cliquez sur le bouton “Fermer” et fermez la fenêtre “Revoir les modifications”.

Pour confirmer que les modifications que vous avez apportées au projet ont été transférées sur GitHub, ouvrez votre

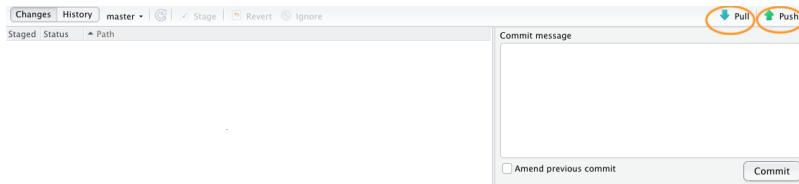


Figure 7.31.

page GitHub, cliquez sur le lien Dépôts, puis sur l’icône `first_repo` dépôt. Vous devriez voir quatre fichiers listés, dont le fichier `first_doc.Rmd` que vous venez de pousser. À côté du nom du fichier, vous verrez votre dernier message de validation (“Première validation” dans ce cas) et la date de la dernière validation.

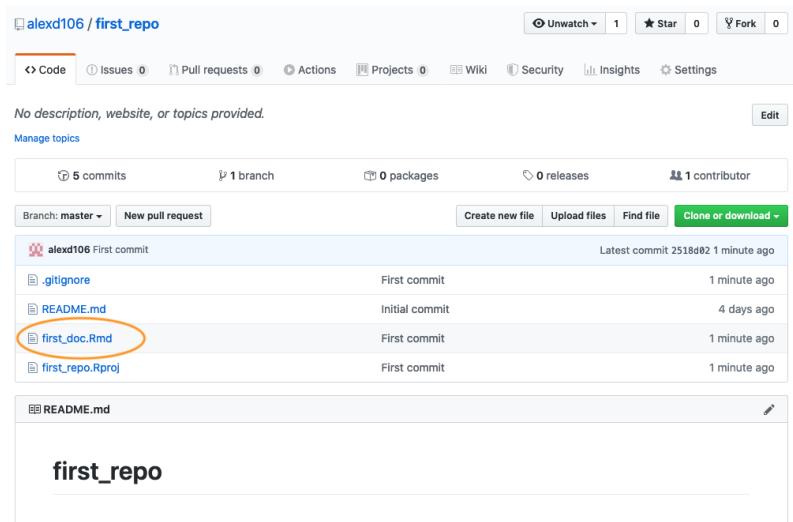


Figure 7.32.

Pour voir le contenu du fichier, cliquez sur le bouton `first_doc.Rmd` nom du fichier.

7.6.1. Suivi des modifications

Après avoir suivi les étapes décrites ci-dessus, vous aurez réussi à modifier un projet RStudio en créant un nouveau document R markdown, à mettre en scène puis à valider ces changements et enfin à pousser les changements vers votre dépôt GitHub. Maintenant, apportons d’autres modifications à votre fichier R markdown et suivons à nouveau le flux de travail, mais cette fois, nous verrons comment identifier les modifications apportées aux fichiers, examiner l’historique des livraisons et comment restaurer une version précédente du document.

Dans RStudio, ouvrez le fichier `first_repo.Rproj` que vous avez créé précédemment (s’il n’est pas déjà ouvert), puis ouvrez le fichier `first_doc.Rmd` (cliquez sur le nom du fichier dans la fenêtre `Files` dans RStudio).

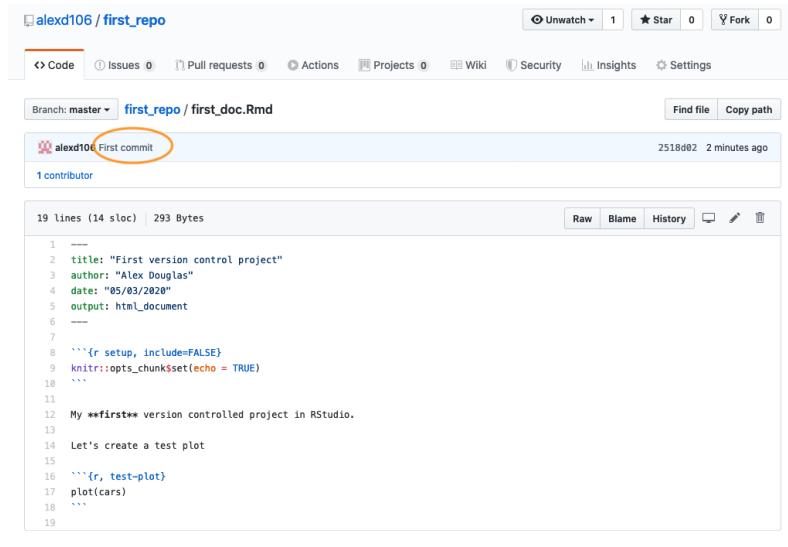


Figure 7.33.

Apportez quelques modifications à ce document. Supprimez la ligne commençant par “Ma première version contrôlée...” et remplacez-la par quelque chose de plus informatif (voir figure ci-dessous). Nous allons également changer les symboles de tracé en rouge et donner des étiquettes aux axes de tracé. Enfin, ajoutons un tableau récapitulatif du cadre de données à l'aide de la commande `kable()` et `summary()` (il se peut que vous ayez besoin d'installer le programme `knitr` si vous ne l'avez pas fait auparavant pour utiliser le paquet `kable()`) et enfin rendre ce document au format pdf en changeant l'option YAML en `output: pdf_document`.

```

1 Go forward to the next source
2 next source | first version control project"
3 location (HTTP) Alex Douglas*
4 date: "05/03/2020"
5 output: pdf_document
6 ---
7
8 ````{r setup, include=FALSE}
9 knitr::opts_chunk$set(echo = TRUE)
10 ````
11
12 This report documents my first attempts of using Git and GitHub to version control an RStudio project. I will be modifying this report, staging and committing changes and then pushing to GitHub.
13
14 Let's create a test plot of distance (miles) and speed (mph).
15
16 ````{r, test-plot}
17 plot(cars, col = "red", xlab = "speed (mph)", ylab = "distance (miles)")
18 ````
19
20 A summary of the data frame is given below
21
22 ````{r, cars-summary}
23 library(knitr)
24 kable(summary(cars))
25 ````
26
27
28
29

```

Figure 7.34.

Sauvegardez ces modifications, puis cliquez sur le bouton `knit` pour effectuer le rendu au format pdf. Un nouveau fichier pdf nommé `first_doc.pdf` sera créé et vous pourrez l'afficher en cliquant sur le nom du fichier dans la fenêtre `Files` dans RStudio.

Notez que ces deux fichiers ont été ajoutés à la base de données Git dans RStudio. Les icônes d'état indiquent

que le fichier `first_doc.Rmd` a été modifié (M majuscule) et que le fichier `first_doc.pdf` n'est pas suivi (point d'interrogation).

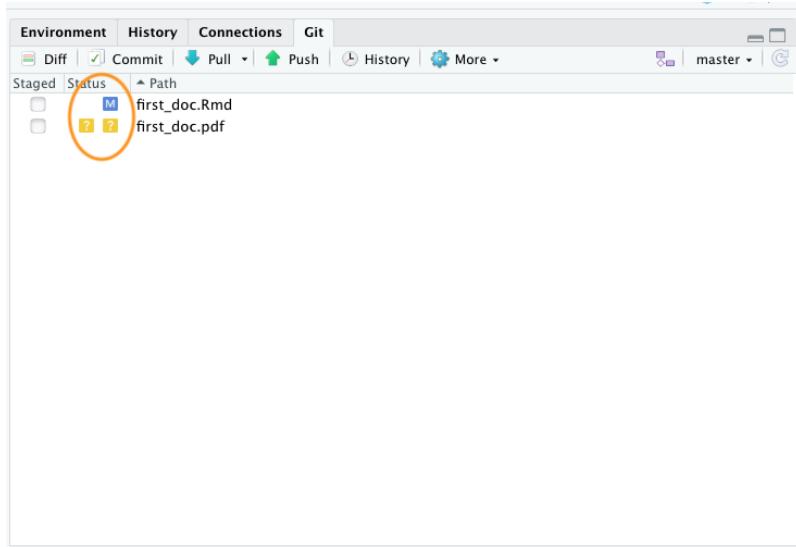


Figure 7.35.

Pour mettre en scène ces fichiers, cochez la case “Mis en scène” pour chaque fichier et cliquez sur le bouton “Valider” pour accéder à la fenêtre “Examiner les modifications”.

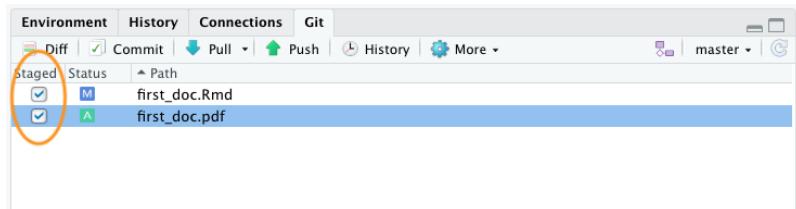


Figure 7.36.

Avant de valider vos modifications, notez l'état des fichiers `first_doc.pdf` est passé de non suivi à ajouté (A). Vous pouvez consulter les modifications que vous avez apportées à l'élément `first_doc.Rmd` en cliquant sur le nom du fichier dans le volet supérieur gauche, ce qui vous donnera un résumé utile des modifications dans le volet inférieur (techniquement appelé diff). Les lignes qui ont été supprimées sont surlignées en rouge et les lignes qui ont été ajoutées sont surlignées en vert (notez que du point de vue de Git, une modification de ligne est en fait deux opérations : la suppression de la ligne d'origine suivie de la création d'une nouvelle ligne). Une fois que vous êtes satisfait, validez ces modifications en rédigeant un message de validation approprié et cliquez sur le bouton “Valider”.

Pour transférer les modifications sur GitHub, cliquez d'abord sur le bouton “Pull” (rappelez-vous qu'il s'agit d'une bonne pratique, même si vous ne collaborez qu'avec vous-même pour l'instant), puis cliquez sur le bouton “Push”.

line numbers

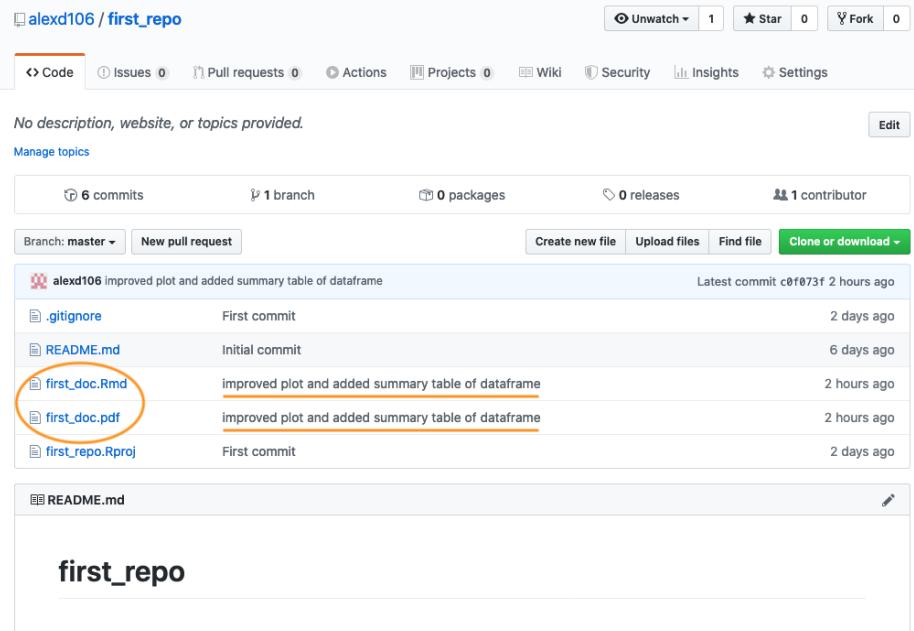
```

@@ -1,19 +1,28 @@
1 1 ---
2 2 title: "First version control project"
3 3 author: "Alex Douglas"
4 4 date: "05/03/2020"
5 5 output: html_document
5 5 output: pdf_document
6 6 ---
7 7
8 8 ```{r setup, include=FALSE}
9 9 knitr::opts_chunk$set(echo = TRUE)
10 10 ...
11 11
12 12 My **first** version controlled project in RStudio.
12 12 This report documents my first attempts of using Git and Github to version control an RStudio project. I will be
modifying this report, staging and committing changes and then pushing to GitHub.
13 13
14 14 Let's create a test plot
14 14 Let's create a test plot of distance (miles) and speed (mph).
15 15
16 16 ```{r, test-plot}
17 17 plot(cars)
17 17 plot(cars, col = "red", xlab = "speed (mph)", ylab = "distance (miles)")
18 18 ...
19 19
20 20 A summary of the data frame is given below
21 21
22 22 ```{r, cars-summary}
23 23 library(knitr)
24 24 kable(summary(cars))
25 25 ...
26 26
27 27
28 28

```

Figure 7.37.

Accédez à votre dépôt GitHub en ligne et vous verrez vos nouveaux commits, y compris le bouton `first_doc.pdf` que vous avez créé lorsque vous avez rendu votre document R markdown.



The screenshot shows a GitHub repository page for 'alex106 / first_repo'. At the top, there are buttons for Unwatch (1), Star (0), Fork (0), and a link to Settings. Below that is a navigation bar with links for Code, Issues (0), Pull requests (0), Actions, Projects (0), Wiki, Security, Insights, and Settings. A message says 'No description, website, or topics provided.' with an 'Edit' button. Below that is a summary bar with '6 commits', '1 branch', '0 packages', '0 releases', and '1 contributor'. A dropdown menu shows 'Branch: master' and a 'New pull request' button. To the right are buttons for 'Create new file', 'Upload files', 'Find file', and a green 'Clone or download' button. The main area lists commits:

| | Author | Message | Date |
|--|------------------|--|-----------------------------------|
| | alex106 | improved plot and added summary table of dataframe | Latest commit c0f073f 2 hours ago |
| | .gitignore | First commit | 2 days ago |
| | README.md | Initial commit | 6 days ago |
| | first_doc.Rmd | improved plot and added summary table of dataframe | 2 hours ago |
| | first_doc.pdf | improved plot and added summary table of dataframe | 2 hours ago |
| | first_repo.Rproj | First commit | 2 days ago |

Below the commits is a preview of the README.md file, which contains the text 'first_repo'.

Figure 7.38.

Pour voir les changements dans `first_doc.Rmd` cliquez sur le nom de ce fichier.

7.6.2. Historique des engagements

L'un des avantages de Git et de GitHub est que vous pouvez consulter l'historique de tous les commits que vous avez effectués, ainsi que les messages de commits associés. Vous pouvez le faire localement en utilisant RStudio (ou la ligne de commande Git) ou si vous avez poussé vos commits sur GitHub, vous pouvez les consulter sur le site web de GitHub.

Pour consulter l'historique des livraisons dans RStudio, cliquez sur le bouton "History" (celui qui ressemble à une horloge) dans le volet Git pour afficher l'historique dans la fenêtre "Review Changes". Vous pouvez également cliquer sur les boutons "Commit" ou "Diff" pour accéder à la même fenêtre (il vous suffit de cliquer en plus sur le bouton "History" dans la fenêtre "Review Changes").

La fenêtre d'historique est divisée en deux parties. Le volet supérieur répertorie toutes les livraisons que vous avez effectuées dans ce dépôt (avec les messages de livraison associés), en commençant par la plus récente en haut et la plus ancienne en bas. Vous pouvez cliquer sur chacune de ces livraisons et le volet inférieur vous montre les modifications que vous avez apportées ainsi qu'un résumé de l'historique. **Date** à laquelle la validation a été effectuée, **l'auteur** du commit et le message du commit (**Sujet**). Il existe également un identifiant unique pour l'engagement

alex106 / **first_repo**

Code Issues 0 Pull requests 0 Actions Projects 0 Wiki Security Insights Settings

Branch: master **first_repo / first_doc.Rmd**

alex106 improved plot and added summary table of dataframe c0f073f 2 hours ago

1 contributor

28 lines (19 sloc) | 642 Bytes

```

1 ---
2   title: "First version control project"
3   author: "Alex Douglas"
4   date: "05/03/2020"
5   output: pdf_document
6 ---
7
8 ```{r setup, include=FALSE}
9 knitr::opts_chunk$set(echo = TRUE)
10 ```
11
12 This report documents my first attempts of using Git and Github to version control an RStudio project. I will be modifying this
13
14 Let's create a test plot of distance (miles) and speed (mph).
15
16 ```{r, test-plot}
17 plot(cars, col = "red", xlab = "speed (mph)", ylab = "distance (miles)")
18 ```
19
20 A summary of the data frame is given below
21
22 ```{r, cars-summary}
23 library(knitr)
24 kable(summary(cars))
25 ```
26
27
28

```

Figure 7.39.

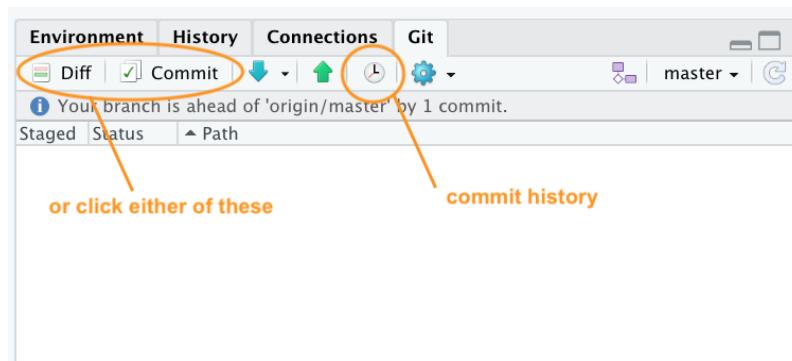


Figure 7.40.

(**SHA** - Secure Hash Algorithm) et un identifiant **Parent** qui identifie la livraison précédente. Ces identifiants SHA sont très importants car vous pouvez les utiliser pour visualiser et revenir à des versions antérieures de fichiers (détails ci-dessous Section 7.6.3). Vous pouvez également consulter le contenu de chaque fichier en cliquant sur le lien “Voir le fichier @ clé SHA” (dans notre cas, “Voir le fichier @ 2b4693d1”).

The screenshot shows the RStudio interface with the 'History' tab selected. It displays three commits:

| SHA | Author | Date | Subject | Parent |
|----------|-----------------------------------|------------------|--|----------|
| 2b4693d1 | alex106 <alexd106@gmail.com> | 2020-03-24 14:18 | improved plot and added summary table of dataframe | d27e79f1 |
| d27e79f1 | alex106 <alexd106@gmail.com> | 2020-03-24 | | c80b0c75 |
| c80b0c75 | Alex Douglas <alexd106@gmail.com> | 2020-03-24 | | |

Below the commits, the file content for 'first_doc.Rmd' is shown. A callout box highlights the commit details (SHA, Author, Date, Subject, Parent) with the label 'summary information'. Another callout box highlights the link 'View file @ 2b4693d1' with the label 'view the file'.

```

@@ -2,17 +2,27 @@
 2 | title: "First version control project"
 3 | author: "Alex Douglas"
 4 | date: "05/03/2020"
 5 | output: html_document
 5 | output: pdf_document
 6 | ---
 7 |
 8 | ````{r setup, include=FALSE}
 9 | knitr::opts_chunk$set(echo = TRUE)
10 | ```
11 |
12 | My **first** version controlled project in RStudio.
13 | This report documents my first attempts at using Git and GitHub to version control an RStudio project. I will be modifying this report, staging and committing changes and then pushing to GitHub.

```

Figure 7.41.

Vous pouvez également consulter l'historique de vos commits sur le site GitHub, mais cette consultation sera limitée aux commits que vous avez déjà transférés sur GitHub. Pour consulter l'historique des livraisons, accédez au dépôt et cliquez sur le lien “livraisons” (dans notre cas, le lien sera intitulé “3 livraisons” car nous avons effectué 3 livraisons).

Vous verrez une liste de tous les commits que vous avez faits, avec les messages de commit, la date du commit et l'identifiant SHA (ce sont les mêmes identifiants SHA que vous avez vus dans l'historique de RStudio). Vous pouvez même parcourir le dépôt à un moment donné en cliquant sur le bouton <> lien. Pour visualiser les modifications apportées aux fichiers associés au commit, il suffit de cliquer sur le lien du commit concerné dans la liste.

Les modifications seront affichées selon le format habituel : vert pour les ajouts et rouge pour les suppressions.

7.6.3. Annulation des modifications

L'un des avantages de l'utilisation de Git est la possibilité de revenir à des versions antérieures des fichiers si vous avez fait une erreur, si vous avez cassé quelque chose ou si vous préférez simplement une approche plus ancienne. La façon de procéder dépend du fait que les modifications que vous souhaitez supprimer ont été mises à disposition,

The screenshot shows the GitHub repository page for `alexd106/first_repo`. At the top, there are buttons for Watch (0), Star (0), and Fork (0). Below the header, a navigation bar includes Code, Issues (0), Pull requests (0), Actions, Projects (0), Wiki, Security, Insights, and Settings. A message states "No description, website, or topics provided." with an "Edit" button. A "Manage topics" link is also present. The main area displays the commit history:

- 3 commits** (circled in orange)
- 1 branch**
- 0 packages**
- 0 releases**
- 1 contributor**

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

| File | Commit Message | Date |
|-------------------------------|--|-----------|
| <code>.gitignore</code> | First commit | yesterday |
| <code>README.md</code> | Initial commit | yesterday |
| <code>first_doc.Rmd</code> | improved plot and added summary table of dataframe | yesterday |
| <code>first_doc.pdf</code> | improved plot and added summary table of dataframe | yesterday |
| <code>first_repo.Rproj</code> | First commit | yesterday |

`README.md`

first_repo

Figure 7.42.

The screenshot shows the GitHub repository page for `alexd106/first_repo`. At the top, there are buttons for Watch (0), Star (0), and Fork (0). Below the header, a navigation bar includes Code, Issues (0), Pull requests (0), Actions, Projects (0), Wiki, Security, Insights, and Settings. A dropdown menu shows "Branch: master". The main area displays the commit history for March 24, 2020:

- Commits on Mar 24, 2020** (circled in orange)

| Commit Message | Author | Date | Actions |
|--|------------------------------|------|----------------------|
| improved plot and added summary table of dataframe | alexd106 committed yesterday | | <code>2b4693d</code> |
| First commit | alexd106 committed yesterday | | <code>d27e79f</code> |
| Initial commit | alexd106 committed yesterday | | <code>c80b0c7</code> |

Newer | Older

Figure 7.43.

```

improved plot and added summary table of dataframe
alex106 committed yesterday
Showing 2 changed files with 15 additions and 5 deletions.
Unified Split

diff --git a/first_doc.Rmd b/first_doc.Rmd
@@ -2,17 +2,27 @@
 2   title: "First version control project"
 3   author: "Alex Douglas"
 4   date: "05/03/2020"
-5   - output: html_document
+5   + output: pdf_document
 6   -----
 7
 8   ```{r setup, include=FALSE}
 9   knitr::opts_chunk$set(echo = TRUE)
10
11
12 - My **first** version controlled project in RStudio.
+ This report documents my first attempts at using Git and GitHub to version control an RStudio project. I will be
  modifying this report, staging and committing changes and then pushing to GitHub.
13
14 - Let's create a test plot
+ Let's create a test plot of distance (miles) and speed (mph)
15
16   ```{r, test-plot}
17 - plot(cars)
18 + plot(cars, col = "red", xlab = "speed (mph)", ylab = "distance (miles)")
+ ```


```

Figure 7.44.

validées ou poussées sur GitHub. Nous allons passer en revue quelques scénarios courants ci-dessous, en utilisant principalement RStudio, mais nous aurons parfois besoin d'utiliser le Terminal (toujours dans RStudio cependant).

Modifications sauvegardées mais non mises à jour, validées ou poussées

Si vous avez enregistré des modifications dans votre ou vos fichiers mais que vous ne les avez pas mis en page, livrés ou poussés sur GitHub, vous pouvez cliquer avec le bouton droit de la souris sur le fichier incriminé dans le panneau Git et sélectionner “Revert ...” (revenir en arrière). Cela ramènera toutes les modifications que vous avez faites au même état que votre dernier commit. Sachez qu'il n'est pas possible d'annuler cette opération, alors utilisez-la avec précaution.

Vous pouvez également annuler les modifications apportées à une partie seulement d'un fichier en ouvrant la fenêtre “Diff” (cliquez sur le bouton “Diff” dans le panneau Git). Sélectionnez la ligne que vous souhaitez annuler en double-cliquant dessus, puis cliquez sur le bouton “Annuler la ligne”. De la même manière, vous pouvez supprimer des morceaux de code en cliquant sur le bouton “Supprimer le morceau”.

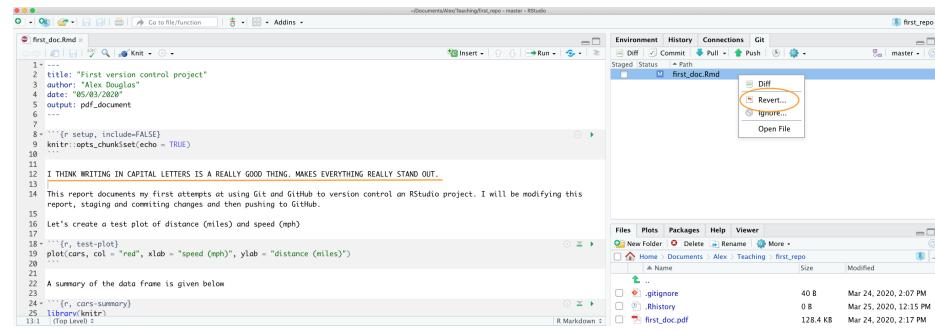


Figure 7.45.

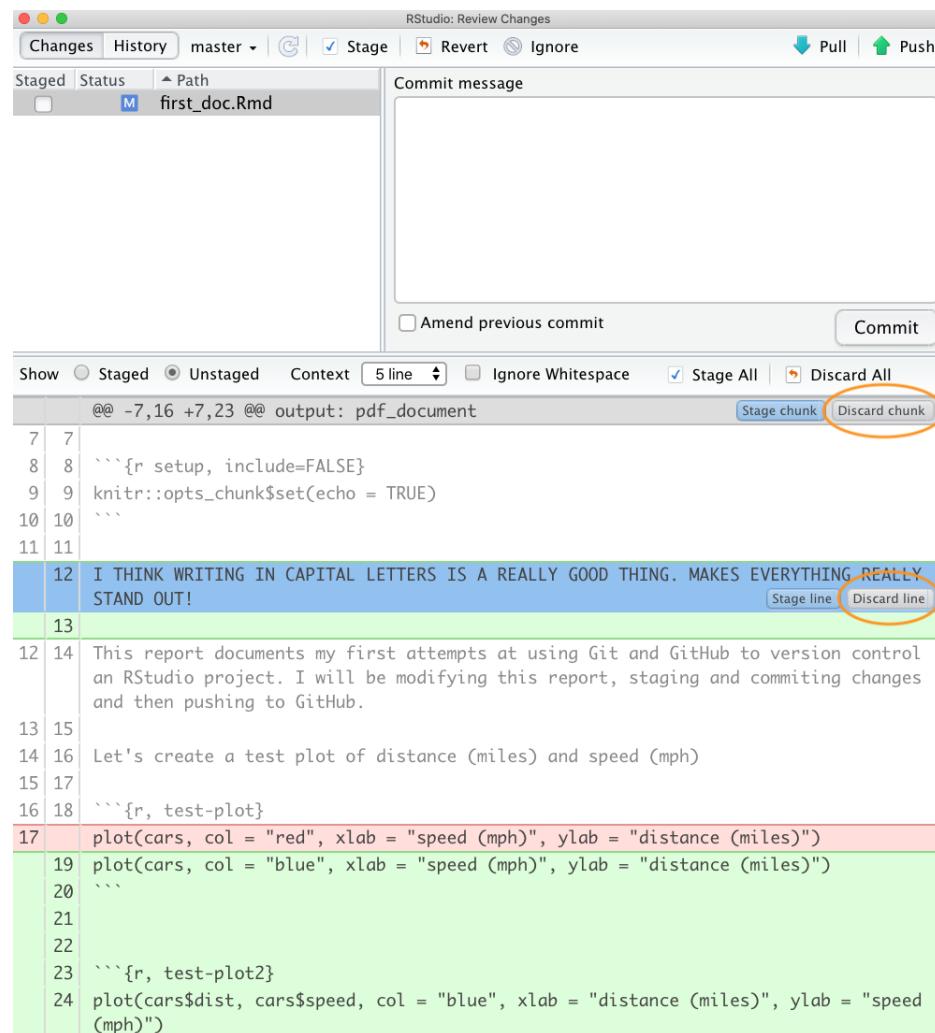


Figure 7.46.

Stagé mais non validé et non poussé

Si vous avez mis en scène vos fichiers, mais que vous ne les avez pas validés, décochez-les simplement en cliquant sur la case “Mis en scène” dans le panneau Git (ou dans la fenêtre “Examiner les modifications”) pour supprimer la coche. Vous pouvez alors revenir sur tout ou partie du fichier comme décrit dans la section ci-dessus.

Stagé et validé mais non poussé

Si vous avez fait une erreur ou avez oublié d’inclure un fichier dans votre dernier commit que vous n’avez pas encore poussé sur GitHub, vous pouvez simplement corriger votre erreur, enregistrer vos modifications, puis modifier votre précédent commit. Vous pouvez le faire en mettant en scène votre fichier, puis en cochant la case “Modifier la livraison précédente” dans la fenêtre “Examiner les modifications” avant de livrer.

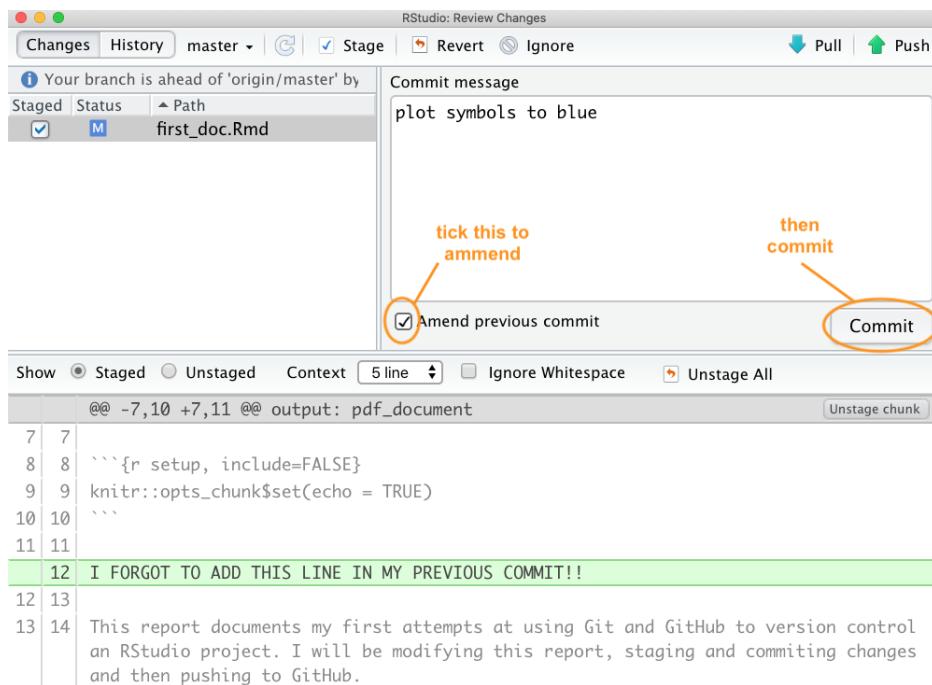


Figure 7.47.

Si nous consultons l'historique des livraisons, nous pouvons voir que notre dernière livraison contient les deux modifications apportées au fichier plutôt que deux livraisons distinctes. Nous utilisons souvent l'approche “amend commit”, mais il est important de comprendre que vous devez **ne pas** faire si vous avez déjà poussé votre dernier commit sur GitHub car vous réécrivez l'histoire et toutes sortes de mauvaises choses peuvent arriver !

Si vous repérez une erreur qui s'est produite plusieurs fois ou si vous souhaitez simplement revenir à une version précédente d'un document, plusieurs options s'offrent à vous.

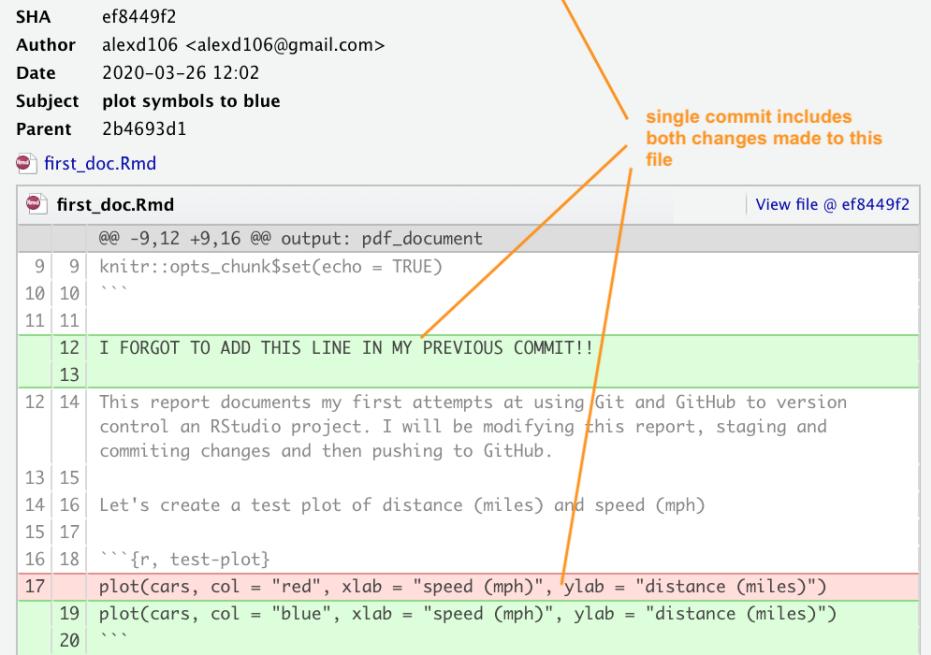
RStudio: Review Changes

Changes History master ▾ (all commits) ▾ C Search Pull

| Subject | Author | Date | SHA |
|--|----------------------------------|------------|----------|
| HEAD -> refs/heads/master plot symbols to blue | alex106 <alex106@gmail.com> | 2020-03-26 | ef8449f2 |
| origin/master origin/HEAD improved plot and adde | alex106 <alex106@gmail.com> | 2020-03-24 | 2b4693d1 |
| First commit | alex106 <alex106@gmail.com> | 2020-03-24 | d27e79f1 |
| Initial commit | Alex Douglas <alex106@gmail.com> | 2020-03-24 | c80b0c75 |

Commits 1-4 of 4

SHA ef8449f2
Author alex106 <alex106@gmail.com>
Date 2020-03-26 12:02
Subject plot symbols to blue
Parent 2b4693d1

A screenshot of the RStudio interface showing a commit history and a detailed view of a specific commit. The commit history shows four commits, with the first one being the selected commit. The commit details show it was authored by 'alex106 <alex106@gmail.com>' on '2020-03-26 12:02' with the subject 'plot symbols to blue'. The commit message includes a note: 'I FORGOT TO ADD THIS LINE IN MY PREVIOUS COMMIT!!'. The code editor shows the R Markdown file 'first_doc.Rmd' with the following content:

```

@@ -9,12 +9,16 @@ output: pdf_document
 9 | 9 knitr::opts_chunk$set(echo = TRUE)
10 | 10 ``
11 | 11
12 | 12 I FORGOT TO ADD THIS LINE IN MY PREVIOUS COMMIT!!
13 | 13
14 | 14 This report documents my first attempts at using Git and GitHub to version
15 | 15 control an RStudio project. I will be modifying this report, staging and
16 | 16 committing changes and then pushing to GitHub.
17 | 17 Let's create a test plot of distance (miles) and speed (mph)
18 | 18 ````{r, test-plot}
19 | 19 plot(cars, col = "red", xlab = "speed (mph)", ylab = "distance (miles)")
20 | 20 ``

```

An orange arrow points from the commit message in the history to the line 'I FORGOT TO ADD THIS LINE IN MY PREVIOUS COMMIT!!' in the code editor. Another orange arrow points from the commit message in the history to the line 'plot(cars, col = "red", xlab = "speed (mph)", ylab = "distance (miles)")' in the code editor. A text annotation 'single commit includes both changes made to this file' is placed near the bottom right of the code editor.

Figure 7.48.

Option 1 - (probablement la plus simple mais très peu Git - mais bon, peu importe !) est de regarder dans votre historique de commit dans RStudio, de trouver le commit sur lequel vous souhaitez revenir et de cliquer sur le bouton ‘View file @’ pour afficher le contenu du fichier.

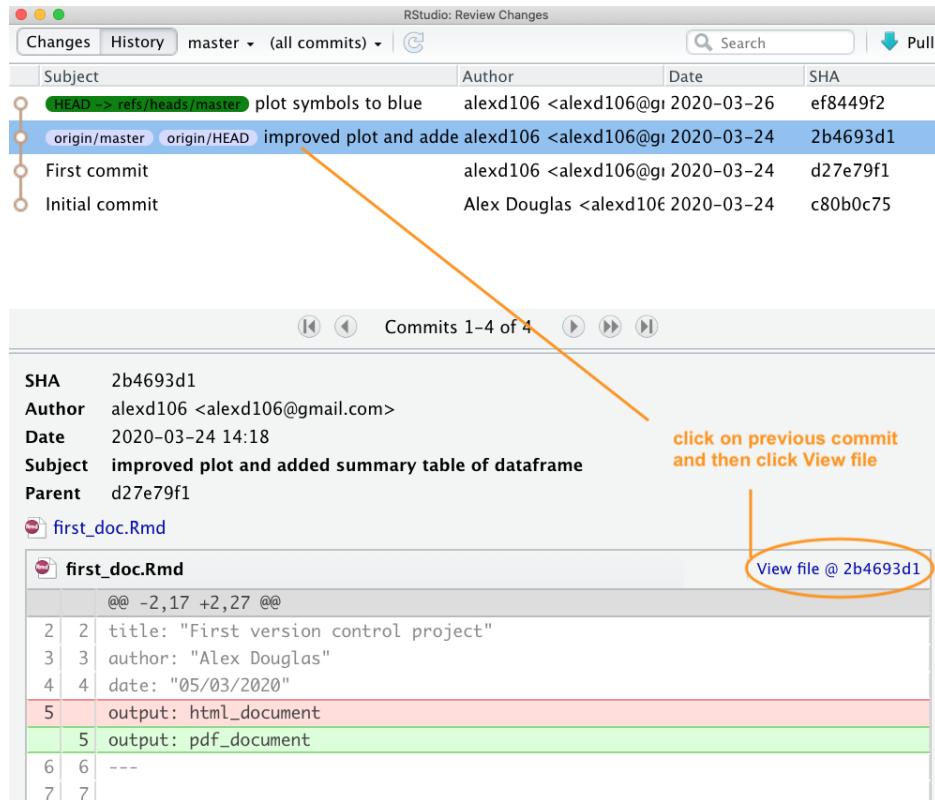


Figure 7.49.

Vous pouvez alors copier le contenu du fichier dans le presse-papiers et le coller dans votre fichier actuel pour remplacer le code ou le texte défectueux. Vous pouvez également cliquer sur le bouton “Enregistrer sous” et enregistrer le fichier sous un autre nom. Une fois que vous avez enregistré votre nouveau fichier, vous pouvez supprimer votre fichier indésirable actuel et continuer à travailler sur votre nouveau fichier. N’oubliez pas de mettre en scène et de valider ce nouveau fichier.

Option 2 - (Git like) Allez dans votre historique Git, trouvez le commit sur lequel vous souhaitez revenir et notez (ou copiez) son identifiant SHA.

Allez maintenant au Terminal dans RStudio et tapez `git checkout <SHA> <filename>`. Dans notre cas, la clé SHA est 2b4693d1 et le nom du fichier est `first_doc.Rmd` notre commande ressemblerait donc à ceci :

```
git checkout 2b4693d1 first_doc.Rmd
```

La commande ci-dessus copiera la version du fichier sélectionné dans le passé et la placera dans le présent. RStudio

```

1 -> 
2   title: "First version control project"
3   author: "Alex Douglas"
4   date: "05/03/2020"
5   output: pdf_document
6   ---
7
8 - ````{r setup, include=FALSE}
9   knitr::opts_chunk$set(echo = TRUE)
10  ```
11
12 This report documents my first attempts at using Git and GitHub to version
13 control an RStudio project. I will be modifying this report, staging and
14 committing changes and then pushing to GitHub.
15
16 ````{r, test-plot}
17   plot(cars, col = "red", xlab = "speed (mph)", ylab = "distance (miles)")
18  ```
19
20 A summary of the data frame is given below
21
22 ````{r, cars-summary}
23   library(knitr)
24   kable(summary(cars))
25  ```
26

```

Figure 7.50.

peut vous demander si vous souhaitez recharger le fichier tel qu'il a été modifié - sélectionnez oui. Vous devrez également mettre en scène et valider le fichier comme d'habitude.

Si vous souhaitez ramener tous vos fichiers au même état qu'une livraison précédente plutôt qu'un seul fichier, vous pouvez utiliser (le seul “point . est important, sinon votre HEAD se détachera) :

```

git rm -r .
git checkout 2b4693d1 .

```

Notez que cela supprimera tous les fichiers que vous avez créés depuis que vous avez effectué ce commit, alors soyez prudent !

Mise en scène, livrée et poussée

Si vous avez déjà transféré vos modifications sur GitHub, vous pouvez utiliser la commande `git checkout` décrite ci-dessus, puis commiter et pousser pour mettre à jour GitHub (bien que cela ne soit pas vraiment considéré comme une “meilleure” pratique). Une autre approche serait d’utiliser `git revert` (Note : pour autant que nous puissions en juger `git revert` n'est pas la même chose que l'option ‘Revert’ dans RStudio). L'option `revert` dans Git crée

The screenshot shows the RStudio interface for reviewing changes. At the top, there's a navigation bar with tabs for 'Changes' (selected), 'History', and 'master'. Below it is a table of commits:

| Subject | Author | Date | SHA |
|--|----------------------------------|------------|----------|
| HEAD -> refs/heads/master rolled back | alex106 <alex106@gmail.com> | 2020-03-26 | 6a4a9a9b |
| plot symbols to blue | alex106 <alex106@gmail.com> | 2020-03-26 | ef8449f2 |
| origin/master origin/HEAD improved plot and adde | alex106 <alex106@gmail.com> | 2020-03-24 | 2b4693d1 |
| First commit | alex106 <alex106@gmail.com> | 2020-03-24 | d27e79f1 |
| Initial commit | Alex Douglas <alex106@gmail.com> | 2020-03-24 | c80b0c75 |

An orange arrow points from the highlighted commit (SHA 2b4693d1) in the list to a detailed view below. The detailed view shows the commit's metadata:

SHA 2b4693d1
Author alex106 <alex106@gmail.com>
Date 2020-03-24 14:18
Subject improved plot and added summary table of dataframe
Parent d27e79f1

Below the metadata is a preview of the file 'first_doc.Rmd' with the commit's content:

```

@@ -2,17 +2,27 @@
 2 | 2 title: "First version control project"
 3 | 3 author: "Alex Douglas"
 4 | 4 date: "05/03/2020"
 5 | output: html_document
 5 | output: pdf_document
 6 | ---
 7 |
 8 | `r setup, include=FALSE}
 9 | knitr::opts_chunk$set(echo = TRUE)
10 | `r
11 |
12 | My **first** version controlled project in RStudio.
12 | This report documents my first attempts at using Git and GitHub to version
| control an RStudio project. I will be modifying this report, staging and
| committing changes and then pushing to GitHub.

```

Figure 7.51.

essentiellement un nouveau commit basé sur un commit précédent et préserve donc tout l'historique des commits. Pour revenir à un état antérieur (commit), vous devez d'abord identifier la ZSD du commit auquel vous souhaitez revenir (comme nous l'avons fait ci-dessus), puis utiliser la commande `revert` dans le terminal. Supposons que nous voulions revenir à notre “Premier commit” qui a un identifiant SHA `d27e79f1`.

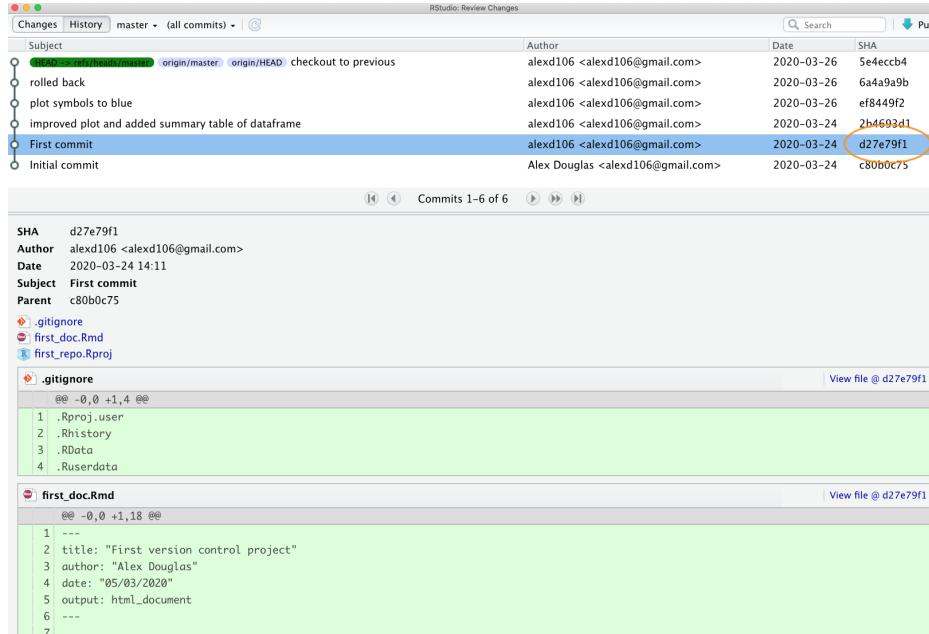


Figure 7.52.

Nous pouvons utiliser le `revert` comme indiqué ci-dessous dans le terminal. La commande `--no-commit` est utilisée pour éviter d'avoir à gérer chaque livraison intermédiaire.

```
git revert --no-commit d27e79f1..HEAD
```

Votre `first_doc.Rmd` va maintenant revenir au même état que celui dans lequel il se trouvait lorsque vous avez effectué votre “premier commit”. Notez également que l’élément `first_doc.pdf` a été supprimé, car il n’était pas présent lorsque nous avons effectué notre première livraison. Vous pouvez maintenant mettre en scène et livrer ces fichiers avec un nouveau message de livraison et enfin les pousser sur GitHub. Remarquez que si nous regardons notre historique des livraisons, toutes les livraisons que nous avons faites sont toujours présentes.

et notre repo sur GitHub reflète également ces changements

7.7. Utiliser Git avec VSCode

Maintenant que nous avons mis en place notre projet et nos dépôts (locaux et distants), il est enfin temps d’apprendre à utiliser Git dans VSCode !

The screenshot shows the RStudio interface with the title "RStudio: Review Changes". The "Changes" tab is selected. A table lists seven commits:

| Subject | Author | Date | SHA | |
|---|-----------------------------------|---|------------|----------|
| HEAD -> refs/heads/master | origin/master | origin/HEAD alexd106 <alexd106@gmail.com> | 2020-03-26 | 550640e2 |
| checkout to previous | | alex106 <alexd106@gmail.com> | 2020-03-26 | 5e4eccb4 |
| rolled back | | alex106 <alexd106@gmail.com> | 2020-03-26 | 6a4a9a9b |
| plot symbols to blue | | alex106 <alexd106@gmail.com> | 2020-03-26 | ef8449f2 |
| improved plot and added summary table of datafr | alex106 <alexd106@gmail.com> | 2020-03-24 | 2b4693d1 | |
| First commit | alex106 <alexd106@gmail.com> | 2020-03-24 | d27e79f1 | |
| Initial commit | Alex Douglas <alexd106@gmail.com> | 2020-03-24 | c80b0c75 | |

Below the table, a message indicates "Commits 1-7 of 7".

The detailed view for the first commit (SHA 550640e2) shows the following information:

- SHA:** 550640e2
- Author:** alexd106 <alexd106@gmail.com>
- Date:** 2020-03-26 16:04
- Subject:** tried revert command
- Parent:** 5e4eccb4

The file content for "first_doc.Rmd" is displayed:

```

@@ -2,27 +2,17 @@
2 | 2 | title: "First version control project"
3 | 3 | author: "Alex Douglas"
4 | 4 | date: "05/03/2020"
5 | 5 | output: pdf_document
6 | 6 | ---
7 |
8 | 8 |   ``{r setup, include=FALSE}
9 | 9 | knitr:::opts_chunk$set(echo = TRUE)
10 | 10 |   ```
11 |
12 | 12 | This report documents my first attempts at using Git and GitHub to version
|    |    control an RStudio project. I will be modifying this report, staging and
|    |    committing changes and then pushing to GitHub.
13 | 13 | My **first** version controlled project in RStudio.

```

Figure 7.53.

The screenshot shows a GitHub repository page for "alexd106 / first_repo".

Repository statistics:

- Code: 7 commits
- Issues: 0
- Pull requests: 0
- Actions: 0
- Projects: 0
- Wiki: 0
- Security: 0
- Insights: 0
- Settings: 0

No description, website, or topics provided.

Manage topics

Branch: master

7 commits

1 branch

0 packages

0 releases

1 contributor

Latest commit 550640e 1 minute ago

alex106 tried revert command

.gitignore

README.md

first_doc.Rmd

first_repo.kproj

README.md

Figure 7.54.

Typiquement, lorsque vous utilisez Git, votre flux de travail se déroule comme suit :

1. Vous créez/supprimez et modifiez les fichiers dans le répertoire de votre projet sur votre ordinateur comme d'habitude (en sauvegardant les modifications au fur et à mesure).
2. Une fois que vous avez atteint un “point d’arrêt” naturel dans votre progression (c.-à-d. vous seriez triste si vous perdiez ce progrès), vous **étape** ces fichiers
3. Ensuite, vous **engager** les modifications que vous avez apportées à ces fichiers mis en scène (avec un message de validation utile), ce qui crée un instantané permanent de ces modifications.
4. Vous continuez ce cycle jusqu'à ce que vous arriviez à un point où vous souhaitez **pousser** ces changements sur GitHub
5. Si vous travaillez avec d’autres personnes sur le même projet, vous pouvez également avoir besoin de **tirer** leurs modifications sur votre ordinateur local

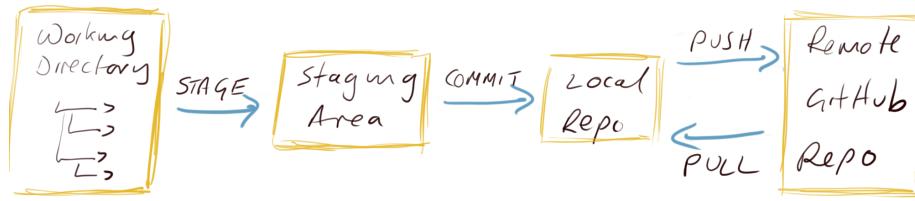


Figure 7.55.

Prenons un exemple pour clarifier ce flux de travail.

Suivi des modifications

Historique des modifications

Annulation des modifications

7.8. Collaborer avec Git

GitHub est un excellent outil de collaboration. Il peut sembler effrayant et compliqué au début, mais cela vaut la peine d’investir un peu de temps pour apprendre comment il fonctionne. Ce qui rend GitHub si bon pour la collaboration, c’est qu’il s’agit d’un système de *système distribué*. Cela signifie que chaque collaborateur travaille sur sa propre

copie du projet et que les modifications sont ensuite fusionnées dans le dépôt distant. Il existe deux façons principales de mettre en place un projet collaboratif sur GitHub. L'une est le flux de travail que nous avons décrit ci-dessus, où chacun connecte son dépôt local au même dépôt distant ; ce système fonctionne bien avec les petits projets où différentes personnes travaillent principalement sur différents aspects du projet, mais il peut rapidement devenir lourd si de nombreuses personnes collaborent et travaillent sur les mêmes fichiers (misère de la fusion !). La seconde approche consiste à ce que chaque collaborateur crée une copie (ou **fork**) du dépôt principal, qui devient leur dépôt distant. Chaque collaborateur doit alors envoyer une demande (une **demande d'extraction**) au propriétaire du référentiel principal afin d'incorporer les modifications dans le référentiel principal, ce qui inclut un processus de révision avant l'intégration des modifications. Plus de détails sur ces sujets peuvent être trouvés dans Section 7.10.

7.9. Conseils Git

D'une manière générale, vous devriez commiter souvent (y compris les commits modifiés) mais pousser beaucoup moins souvent. Cela facilite la collaboration et rend le processus de retour aux versions précédentes des documents beaucoup plus simple. En général, nous n'envoyons des modifications sur GitHub que lorsque nous sommes satisfaits que nos collaborateurs (ou le reste du monde) puissent voir notre travail. Cependant, cela dépend entièrement de vous, du projet (et des personnes avec lesquelles vous travaillez) et de vos priorités dans l'utilisation de Git.

Si vous ne voulez pas suivre un fichier dans votre dépôt (peut-être s'agit-il de fichiers trop volumineux ou transitoires), vous pouvez faire en sorte que Git ignore le fichier en l'ajoutant à la balise `.gitignore` pour qu'il soit ignoré par Git. Sur RStudio, dans le panneau git, vous pouvez faire un clic droit sur le nom du fichier à exclure et sélectionner ‘Ignore...’

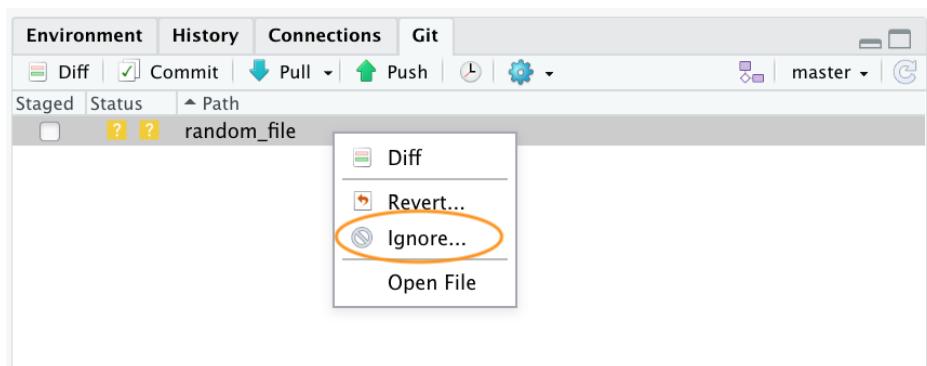


Figure 7.56.

Cela ajoutera le nom du fichier à la base de données `.gitignore` fichier. Si vous souhaitez ignorer plusieurs fichiers ou un type de fichier particulier, vous pouvez également inclure des caractères génériques dans la commande

.gitignore dans le fichier. Par exemple, pour ignorer tous les fichiers png vous pouvez inclure l'expression *.png dans votre .gitignore et enregistrer.

Si tout va de travers et que vous finissez par détruire complètement votre dépôt Git, ne désespérez pas (nous sommes tous passés par là !). Tant que votre dépôt GitHub est bon, tout ce que vous avez à faire est de supprimer le répertoire du projet en question sur votre ordinateur, de créer un nouveau projet RStudio et de le lier à votre dépôt GitHub distant en utilisant l'option 2 (7.5.3). Une fois que vous avez cloné le dépôt distant, vous devriez être prêt à partir.

7.10. Autres ressources

Il existe de nombreux guides en ligne pour en savoir plus sur git et GitHub et, comme pour tout logiciel open source, il existe une vaste communauté qui peut être d'une grande aide :

- Le guide de la British Ecological Society sur [Code reproductive](#)
- Les guides [guides GitHub](#)
- Le laboratoire scientifique de Mozilla [Guide GitHub pour la collaboration sur les projets ouverts](#)
- Jenny Bryan's [Joyeux Git et GitHub](#) . Nous avons emprunté l'idée (mais avec un contenu différent) de RStudio d'abord, RStudio ensuite dans la section "Mise en place d'un projet à version contrôlée dans RStudio".
- L'article de Melanie Frazier [GitHub : Un guide du débutant pour remonter le temps \(et réparer les erreurs\)](#) . Nous avons suivi cette structure (avec des modifications et un contenu différent) dans la section "Revenir sur les modifications".
- Si vous avez fait quelque chose de terriblement mauvais et que vous ne savez pas comment le réparer, essayez [Oh Shit, Git](#) ou si vous êtes facilement offensé [Dangit, Git](#)

Ce ne sont que quelques exemples, il vous suffit de faire une recherche sur "contrôle de version avec git et GitHub" pour voir à quel point la communauté autour de ces projets open source est importante et combien de ressources gratuites sont disponibles pour que vous deveniez un expert en contrôle de version.

7.11. Pratique

7.11.1. Contexte

Nous allons configurer Rstudio pour qu'il fonctionne avec notre compte github, puis nous créerons un nouveau projet et commencerons à utiliser github. Pour avoir des données, je suggère d'utiliser l'outil génial `palmerpenguins` dataset .

7.11.2. Informations sur les données

Ces données ont été collectées et partagées par [Dr. Kristen Gorman](#) et [Station Palmer, Antarctique LTER](#).

L'ensemble a été conçu par les Drs Allison Horst et Alison Hill. [site officiel](#).

Le paquet `palmerpenguins` comporte deux ensembles de données.

```
library(palmerpenguins)
```

L'ensemble de données `penguins` est une version simplifiée des données brutes ; voir `?penguins` pour plus d'informations :

```
head(penguins)
```

| species | island | bill_length_mm | bill_depth_mm | flipper_length_mm | body_mass_g | sex | year |
|---------|-----------|----------------|---------------|-------------------|-------------|--------|------|
| Adelie | Torgersen | 39.1 | 18.7 | 181 | 3750 | male | 2007 |
| Adelie | Torgersen | 39.5 | 17.4 | 186 | 3800 | female | 2007 |
| Adelie | Torgersen | 40.3 | 18.0 | 195 | 3250 | female | 2007 |
| Adelie | Torgersen | NA | NA | NA | NA | NA | 2007 |
| Adelie | Torgersen | 36.7 | 19.3 | 193 | 3450 | female | 2007 |
| Adelie | Torgersen | 39.3 | 20.6 | 190 | 3650 | male | 2007 |

L'autre jeu de données `penguins_raw` contient les données brutes ; voir `?penguins_raw` pour plus d'informations :

```
head(penguins_raw)
```

| stu- | Sample | dy- | Num- | Name | Species | Region | Stage | ID | In- | di- | | | Clutch | | Flip- | | | Delta | |
|---------|----------|-----|------|------|---------|--------|-------|-----|-----------|------|------|------|--------|---------|--------|----------|--------|--------|----------|
| | | | | | | | | | | vi- | Com- | dual | ple- | Date | Length | Depth | Culmen | Culmen | per |
| PAL0708 | Adelie | | | An- | Tor- | Adult, | N1A1 | Yes | 2007-39.1 | 18.7 | 181 | 3750 | MAL | NA | NA | NA | NA | NA | Not |
| | Penguin | | | vers | ger- | 1 | | | 11- | | | | | | | | | | enough |
| | (Pygos- | | | sen | Egg | | | | 11 | | | | | | | | | | blood |
| | celis | | | | Stage | | | | | | | | | | | | | | for iso- |
| | adeliae) | | | | | | | | | | | | | | | | | | topes. |
| PAL0708 | Adelie | | | An- | Tor- | Adult, | N1A2 | Yes | 2007-39.5 | 17.4 | 186 | 3800 | FE- | 8.94956 | - | NA | | | |
| | Penguin | | | vers | ger- | 1 | | | 11- | | | | | MALE | | 24.69454 | | | |
| | (Pygos- | | | sen | Egg | | | | 11 | | | | | | | | | | |
| | celis | | | | Stage | | | | | | | | | | | | | | |
| | adeliae) | | | | | | | | | | | | | | | | | | |
| PAL0708 | Adelie | | | An- | Tor- | Adult, | N2A1 | Yes | 2007-40.3 | 18.0 | 195 | 3250 | FE- | 8.36821 | - | NA | | | |
| | Penguin | | | vers | ger- | 1 | | | 11- | | | | | MALE | | 25.33302 | | | |
| | (Pygos- | | | sen | Egg | | | | 16 | | | | | | | | | | |
| | celis | | | | Stage | | | | | | | | | | | | | | |
| | adeliae) | | | | | | | | | | | | | | | | | | |
| PAL0708 | Adelie | | | An- | Tor- | Adult, | N2A2 | Yes | 2007- | NA | NA | NA | NA | NA | NA | NA | NA | NA | Adult |
| | Penguin | | | vers | ger- | 1 | | | 11- | | | | | | | | | | not |
| | (Pygos- | | | sen | Egg | | | | 16 | | | | | | | | | | sam- |
| | celis | | | | Stage | | | | | | | | | | | | | | pled. |
| | adeliae) | | | | | | | | | | | | | | | | | | |
| PAL0708 | Adelie | | | An- | Tor- | Adult, | N3A1 | Yes | 2007-36.7 | 19.3 | 193 | 3450 | FE- | 8.76651 | - | NA | | | |
| | Penguin | | | vers | ger- | 1 | | | 11- | | | | | MALE | | 25.32426 | | | |
| | (Pygos- | | | sen | Egg | | | | 16 | | | | | | | | | | |
| | celis | | | | Stage | | | | | | | | | | | | | | |
| | adeliae) | | | | | | | | | | | | | | | | | | |

| | | | In- | di- | Clutch | | Flip- | | Delta | | | | | |
|---------|----------|---------|-------|--------|--------|--------|-----------|-------|--------|-------|------------|----------|------------|--------------|
| stu- | Sample | | vi- | Com- | Culmen | Culmen | per | Body | 15 | Delta | | | | |
| dy- | Num- | Re- | Is- | dual | ple- | Date | Length | Depth | Length | Mass | N | 13C | Com- | |
| Name | ber | Species | gion | land | Stage | ID | tion | Egg | (mm) | (mm) | (mm) | (g) | Sex (o/oo) | (o/oo) ments |
| PAL0708 | Adelie | An- | Tor- | Adult, | N3A2 | Yes | 2007-39.3 | 20.6 | 190 | 3650 | MA18E66496 | - | NA | |
| | Penguin | vers | ger- | 1 | | | 11- | | | | | 25.29805 | | |
| | (Pygos- | sen | Egg | | | | 16 | | | | | | | |
| | celis | | Stage | | | | | | | | | | | |
| | adeliae) | | | | | | | | | | | | | |

Pour cet exercice, nous allons utiliser l'outil penguins jeu de données.

7.11.3. Questions

- 1) Créez un compte github si ce n'est pas encore fait.
- 2) Configurez Rstudio avec votre compte github en utilisant l'option usethis paquet.
- 3) Créez et stockez votre jeton d'autorisation personnel GITHUB
- 4) Créez un nouveau projet R Markdown, l'initialiser pour git, et créer un nouveau dépôt git
- 5) Créez un nouveau document Rmarkdown dans votre projet. Sauvegardez ensuite le fichier et mettez-le en scène.
- 6) Créez un nouveau commit incluant le nouveau fichier et le pousser sur github (Vérifier sur github que cela fonctionne).
- 7) Modifiez le fichier. Supprimez tout ce qui se trouve après la ligne 12. Ajoutez un nouveau titre de section, un texte simple et un texte en caractères gras. Puis tricotez et compilez.
- 8) Faire un nouveau commit (avec un message significatif), et pousser sur github.
- 9) Créez une nouvelle branche, et ajoutez une nouvelle section au fichier rmarkdown dans cette branche. Ce que vous voulez. Je suggérerais un graphique des données.
- 10) Créez un commit et le pousser sur la branche.
- 11) Sur github, créer une pull request pour fusionner les 2 branches différentes.

12) Vérifier et accepter la pull request pour fusionner les 2 branches.

Vous avez utilisé avec succès tous les outils essentiels de git 🎉. Vous êtes prêt à explorer 🧑 et découvrir sa puissance 💪



Figure 7.57.: Happy git(hub)-ing

7.11.4. Solution

2)

```
usethis::git_sitrep()
usethis::use_git_config(
  user.name = "your_username",
  user.email = "your_email@address.com"
)
```

3)

```
usethis::create_github_token()
gitcreds::gitcreds_set()
```

4)

```
#create R project
usethis::use_git()

#restart R
```

```
usethis::use_github()  
usethis::git_vaccinate()
```

partie II.

Principes de statistique

Chapitre 8

Analyse de puissance avec R et G*Power

Objectifs de ce chapitre :

- Pouvoir calculer la **puissance** d'un test de t avec R et G*Power
- Pouvoir calculer l'**effectif requis** pour obtenir la puissance désirée avec un test de t
- Pouvoir calculer la **taille d'effet détectable** par un test de t étant donné *l'effectif*, la *puissance* et α
- Comprendre comment la puissance change lorsque l'effectif augmente, la taille d'effet change, ou lorsque α diminue
- Comprendre comment la puissance est affectée lorsque l'on passe d'un test bilatéral à un test unilatéral

8.1. La théorie

8.1.1. Qu'est-ce que la puissance?

La puissance est la probabilité de rejeter l'hypothèse nulle quand elle est fausse.

8.1.2. Pourquoi faire une analyse de puissance?

Évaluer l'évidence

L'analyse de puissance effectuée après avoir accepté une hypothèse nulle permet de calculer la probabilité que l'hypothèse nulle soit rejetée si elle était fausse et que la taille d'effet était d'une valeur donnée. Ce type d'analyse *a posteriori* est très commun.

Planifier de meilleures expériences

L'analyse de puissance effectuée avant de réaliser une expérience (le plus souvent après une expérience préliminaire cependant), permet de déterminer le nombre d'observations nécessaires pour détecter un effet d'une taille donnée à un niveau fixe de probabilité (la puissance). Ce type d'analyse a priori devrait être réalisé plus souvent.

Estimer la “limite de détection” statistique

L'effort d'échantillonnage est souvent déterminé à l'avance (par exemple lorsque vous héritez de données récoltées par quelqu'un d'autre), ou très sévèrement limité (lorsque les contraintes logistiques prévalent). Que ce soit a priori ou a posteriori l'analyse de puissance vous permet d'estimer, pour un effort d'échantillonnage donné et un niveau de puissance fixe, quelle est la taille minimale de l'effet qui peut être détecté (comme étant statistiquement significatif).

8.1.3. Facteurs qui affectent la puissance

Il y a 3 facteurs qui affectent la puissance d'un test statistique.

Le critère de décision

La puissance dépend de α , le seuil de probabilité auquel on rejette l'hypothèse nulle. Si ce seuil est très strict (*i.e.* si α est fixé à une valeur très basse, comme 0.1% ou $p = 0.001$), alors la puissance sera plus faible que si le seuil était moins strict.

La taille de l'échantillon

Plus l'échantillon est grand, plus la puissance est élevée. La capacité d'un test à détecter de petites différences comme étant statistiquement significatives augmente avec une augmentation du nombre d'observations.

La taille d'effet

Plus la taille d'effet est grande, plus un test a de puissance. Pour un échantillon de taille fixe, la capacité d'un test à détecter un effet comme étant statistiquement significatif est plus élevée si l'effet est grand que s'il est petit. La taille d'effet est en fait une mesure du degré de fausseté de l'hypothèse nulle.

8.2. Qu'est ce que G*Power?

G*Power est un programme gratuit, développé par des psychologues de l'Université de Dusseldorf en Allemagne. Le programme existe en version Mac et Windows. Il peut cependant être utilisé sous linux via Wine ou une machine virtuelle.

G*Power vous permettra d'effectuer une analyse de puissance pour la majorité des tests que nous verrons au cours de la session sans avoir à effectuer des calculs complexes ou farfouiller dans des tableaux ou des figures décrivant des distributions ou des courbes de puissance. G*power est vraiment un outil très utile que vous devrez maîtriser.

Il est possible de faire toutes les analyses de G*power avec R, mais cela est un peu plus complexes et nécessite un peu plus de code et donc une meilleure compréhension du processus. Dans les cas les plus simple le code R est aussi fourni.

🔥 Exercice

Téléchargez le programme [ici](#) et installez-le sur votre ordi et votre station de travail au laboratoire (si ce n'est déjà fait).

8.3. Comment utiliser G*Power

8.3.1. Principe général

L'utilisation de G*Power implique généralement 3 étapes:

1. Choisir le test approprié
2. Choisir l'un des 5 types d'analyses de puissance disponibles
3. Incrire les valeurs des paramètres requis et cliquer sur **Calculate**

8.3.2. Types d'analyses de puissance disponibles

A priori

Calcule l'effectif requis pour une valeur de α , β et de taille d'effet donnée. Ce type d'analyse est utile à l'étape de planification des expériences.

Compromis

Calcule α et β pour un rapport β/α donné, un effectif fixe, et une taille d'effet donnée. Ce type d'analyse est plus rarement utilisé (je ne l'ai jamais fait), mais peut être utile lorsque le rapport β/α est d'intérêt, par exemple lorsque le coût d'une erreur de type I et de type II peut être quantifié.

Critère

Calcule α pour β , effectif et taille d'effet donné. En pratique, je vois peu d'utilité pour ce type de calcul. Contactez-moi si vous en voyez une!

Post-hoc

Calcule la puissance ($1 - \beta$) pour α , une taille d'effet et un effectif donné. Très utilisée pour interpréter les résultats d'une analyse statistique non-significative, mais seulement si l'on utilise une taille d'effet biologiquement significative (et non la taille d'effet observée). Peu pertinente lorsque le test est significatif.

Sensibilité

Calcule la taille d'effet détectable pour une valeur d' α , β et un effectif donné. Très utile également au stade de planification des expériences.

8.3.3. Comment calculer la taille d'effet ?

G*Power permet de faire une analyse de puissance pour de nombreux tests statistiques.

L'indice de la taille d'effet qui est utilisé par G*Power pour les calculs dépend du test. Notez que d'autres logiciels peuvent utiliser des indices différents et il est important de vérifier que l'indice que l'on utilise est celui qui convient. G*Power vous facilite la tâche et permet de calculer la taille d'effet en inscrivant seulement les valeurs pertinentes dans la fenêtre de calcul. Le tableau suivant donne les indices utilisés par G*Power pour les différents tests.

| Test | Taille d'effet | Formule |
|---------------------------------|----------------|--|
| test de t sur des moyennes | d | $d = \frac{ \mu_1 - \mu_2 }{\sqrt{(s_1^2 + s_2^2)/2}}$ |
| test de t pour des corrélations | r | |
| autres tests de t | d | $d = \frac{\mu}{\sigma}$ |

| Test | Taille d'effet | Formule |
|----------------|----------------|---|
| test F (ANOVA) | f | $f = \frac{\sqrt{\sum_{i=1}^k (\mu_i - \mu)^2}}{\sigma}$ |
| autres test F | f^2 | $f^2 = \frac{R_p^2}{1-R_p^2}$
R_p est le coefficient de corrélation partiel |
| test Chi-carré | w | $w = \sqrt{\sum_{i=1}^m \frac{(p_{0i} - p_{1i})^2}{p_{0i}}}$
p_{0i} p_{1i} sont les proportions de la catégorie i prédites par l'hypothèse nulle et alternative respectivement |

8.4. Puissance pour un test de t comparant deux moyennes

! Important

L'ensemble des analyses de puissance décrites après peuvent être réalisé avec 2 fonctions dans R.

- `pwr.t.test()`: lorsque les tailles d'échantillons sont identiques
- `pwr.t2n.test()`: lorsque les échantillons ont des tailles différentes

L'objectif de cette séance de laboratoire est de vous familiariser avec G*Power et de vous aider à comprendre comment les quatre paramètres des analyses de puissance (α , β , effectif et taille d'effet) sont reliés entre eux. On examinera seulement l'un des nombreux tests : le *test de t* permettant de comparer deux moyennes indépendantes. C'est le test le plus communément utilisé par les biologistes, vous l'avez tous déjà utilisé, et il conviendra très bien pour les besoins de la cause. Ce que vous apprendrez aujourd'hui s'appliquera à toutes les autres analyses de puissance que vous effectuerez à l'avenir.

Jayne Stephenson a étudié la productivité des ruisseaux de la région d'Ottawa. Elle a, entre autres, quantifié la biomasse des poissons dans 18 ruisseaux sur le Bouclier Canadien d'une part, et dans 18 autres ruisseaux de la vallée de la rivière des Outaouais et de la rivière Rideau d'autre part. Elle a observé une biomasse plus faible dans les ruisseaux de la vallée (2.64 g/m^2 , écart-type=3.28) que dans ceux du Bouclier (3.31 g/m^2 , écart-type=2.79). En faisant un test de t pour éprouver l'hypothèse nulle que la biomasse des poissons est la même dans les deux régions, elle obtient:

Pooled-Variancne Two-Sample t-Test

```
t = -0.5746, df = 34, p-value = 0.5693
```

Elle accepte l'hypothèse nulle (puisque p est plus élevé que 0.05) conclue donc que la biomasse moyenne des poissons est la même dans ces deux régions.

8.4.1. Analyse post-hoc - Calculer la puissance du test

Compte tenu des valeurs des moyennes observées et de leur écart-type, on peut utiliser G*Power pour calculer la puissance du test de t bilatéral pour deux moyennes indépendantes et pour la taille d'effet (i.e. la différence entre la biomasse entre les deux régions, pondérée par les écarts-type) à $\alpha = 0.05$.

Démarrer G*Power.

1. À **Test family**, choisir: t tests
2. À **Statistical test**, choisir: Means: Difference between two independent means (two groups)
3. À **Type of power analysis**, choisir: Post hoc: Compute achieved power - given α , sample size, and effect size
4. Dans **Input Parameters**,
 - à la boîte **Tail(s)**, choisir: Two,
 - vérifier que α **err prob** est égal à 0.05
 - inscrire 18 pour **Sample size group 1** et **2**
 - pour calculer la taille d'effet (Effect size d), cliquer sur le bouton **Determine =>**
5. Dans la fenêtre qui s'ouvre à droite, sélectionner $n1 = n2$
 - entrer les moyennes (**Mean group 1** et **2**)
 - entrer les écarts types (**SD σ group 1** et **2**)
 - cliquer sur le bouton **Calculate and transfer to main window**
6. Cliquer sur le bouton **Calculate** dans la fenêtre principale et vous devriez obtenir ceci:

La même analyse peut être faite en utilisant R. D'abord, il faut calculer la taille d'effet d pour un test de t comparant deux moyennes, puis utiliser la fonction `pwr.t.test` du paquet `pwr`. Le plus simple comme nous allons estimer la taille d'effet d plusieurs fois et de créer une petite fonction qui estime d basé sur les paramètres nécessaires.

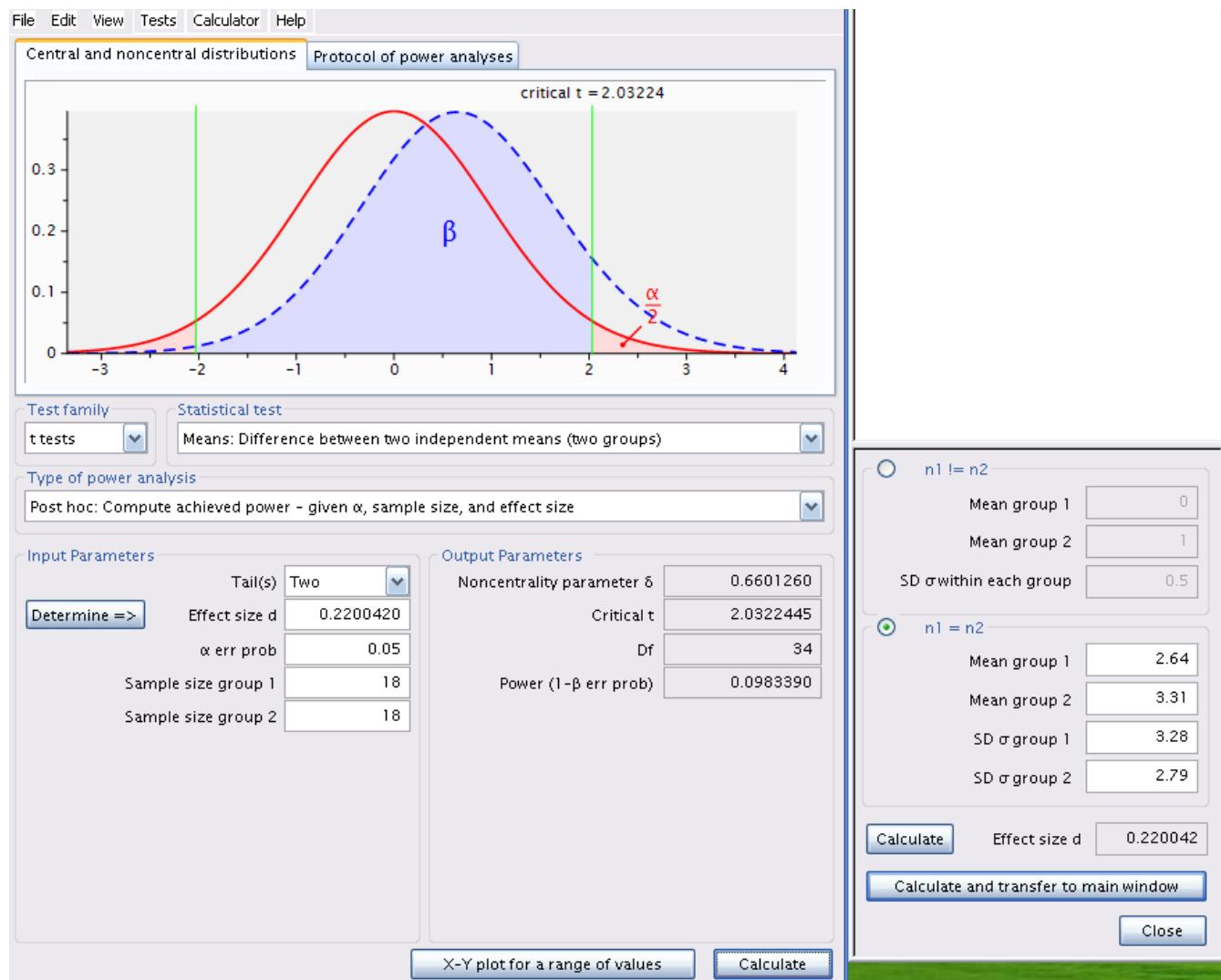


Figure 8.1.: Analyse post-hoc avec la taille d'effet estimée

```
# charger le paquet pwr
library(pwr)

# définir une fonction pour d
d <- function(u1, u2, sd1, sd2) {
  abs(u1 - u2) / sqrt((sd1^2 + sd2^2) / 2)
}

# analyse de puissance

d_cohen <- d(u1 = 2.64, sd1 = 3.28, u2 = 3.31, sd2 = 2.79)

pwr.t.test(
  n = 18, d = d_cohen, sig.level = 0.05,
  type = "two.sample"
)
```

Two-sample t test power calculation

```
n = 18
d = 0.220042
sig.level = 0.05
power = 0.09833902
alternative = two.sided
```

NOTE: n is number in *each* group

```
# graphique comme g*Power
x <- seq(-4, 4, length = 200)
y0 <- dt(x, 34)
y1 <- dt(x, 34, ncp = d_cohen * sqrt(36) / 2)
plot(x, y0, type = "l", col = "red", lwd = 2)
qc <- qt(0.025, 34)
abline(v = qc, col = "green")
```

```

abline(v = -qc, col = "green")
lines(x, y1, type = "l", col = "blue", lwd = 2)

# l'erreur de type 2 correspond à la zone bleue
polygon(
  c(x[x <= -qc], -qc), c(y1[x <= -qc], 0),
  col = rgb(red = 0, green = 0.2, blue = 1, alpha = 0.5)
)
polygon(
  c(-qc, x[x >= -qc]), c(0, y0[x >= -qc]),
  col = rgb(red = 1, green = 0, blue = 0.2, alpha = 0.5)
)
polygon(
  c(x[x <= qc], qc), c(y0[x <= qc], 0),
  col = rgb(red = 1, green = 0, blue = 0.2, alpha = 0.5)
)

```

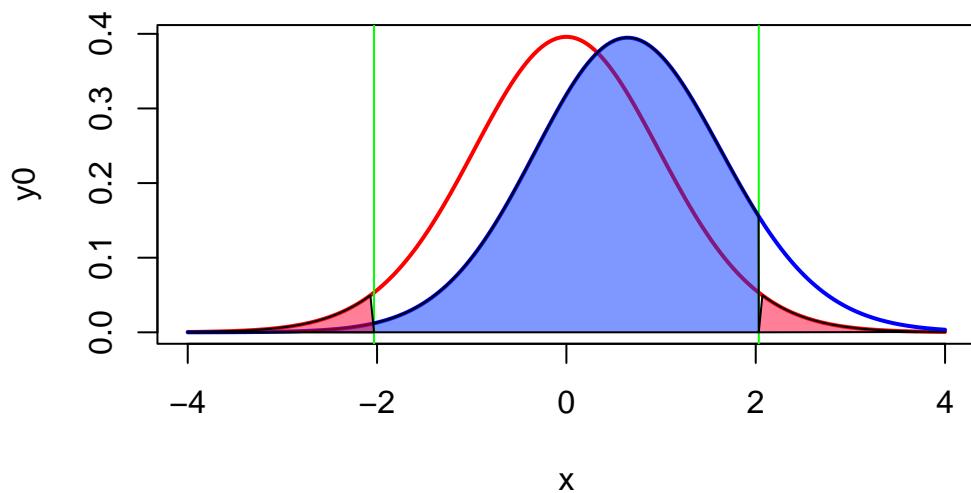


Figure 8.2.: Graphique de l'analyse de puissance dans R

Voici le code pour faire la figure avec ggplot.

```

qc <- qt(0.025, 34)
ncp <- d_cohen * sqrt(36) / 2
dat <- data.frame(
  x = seq(-4, 4, length = 200),
  y0 = dt(x, (n - 1) * 2),
  y1 = dt(x, (n - 1) * 2, ncp = ncp)
) %>%
  mutate(
    beta = ifelse(x <= -qc, y1, 0),
    alpha = ifelse(x <= qc | x >= -qc, y0, 0)
  )

ggplot(dat, aes(x = x)) +
  geom_line(aes(y = y0), color = "red") +
  geom_line(aes(y = y1), color = "blue") +
  geom_vline(xintercept = qcl, color = "green") +
  geom_area(
    aes(x = x, y = beta),
    fill = rgb(red = 0, green = 0.2, blue = 1, alpha = 0.5)
  ) +
  geom_area(
    aes(x = x, y = alpha),
    fill = rgb(red = 1, green = 0, blue = 0.2, alpha = 0.5)
  ) +
  theme_classic() +
  ylab("dt(x)")

```

Étudions un peu la figure Figure 8.1.

- La courbe de gauche, en rouge, correspond à la **distribution de la statistique t si H_0 est vraie** (i.e si les deux moyennes étaient égales) compte tenu de l'effectif (18 dans chaque région) et des écarts- types observés.
- Les lignes verticales correspondent aux **valeurs critiques de t pour $\alpha = 0.05$** et un effectif total de 36 (2×18).
- Les régions ombrées en rose correspondent aux **zones de rejet de H_0 ($\alpha/2$)**. Si Jaynie avait obtenu une valeur

de t en dehors de l'intervalle délimité par les valeurs critiques allant de -2.03224 à 2.03224, alors elle aurait rejeté H_0 , l'hypothèse nulle d'égalité des deux moyennes. En fait, elle a obtenu une valeur de t égale à -0.5746 et conclu que la biomasse est la même dans les deux régions.

- La courbe de droite, en bleu, correspond à la **distribution de la statistique t si H_1 est vraie** (ici H_1 correspond à une différence de biomasse entre les deux régions de $3.33 - 2.64 = 0.69 g/m^2$, compte tenu des écarts-types observés). Cette distribution correspond à ce qu'on devrait s'attendre à observer si H_1 était vraie et que l'on répétait un grand nombre de fois les mesures dans des échantillons aléatoires de 18 ruisseaux des deux régions en calculant la statistique t à chaque fois. En moyenne, on observerait une valeur de t d'environ 0.6.
- Notez que la distribution de droite chevauche considérablement celle de gauche, et une bonne partie de la surface sous la courbe de droite se retrouve à l'intérieur de l'intervalle d'acceptation de H_0 , délimité par les deux lignes vertes et allant de -2.03224 à 2.03224. Cette proportion, correspondant à la partie ombrée en bleu sous la courbe de droite et dénoté par β correspond au risque d'erreur de type II qui est d'accepter H_0 quand H_1 est vraie.
- La **puissance est simplement** $1 - \beta$, et est ici de 0.098339 (*Résultats du test d'estimation de puissance*) . Donc, si la biomasse différait de $0.69 g/m^2$ entre les deux régions, Jaynie n'avait que 9.8% des chances d'être capable de détecter une différence statistiquement significative à $\alpha = 5$ en échantillonnant 18 ruisseaux de chaque région.

Récapitulons: La différence de biomasse entre les deux régions n'est pas statistiquement significative d'après le test de t. C'est donc que cette différence est relativement petite compte tenu de la précision des mesures. Il n'est donc pas très surprenant que la puissance, i.e. *la probabilité de détecter une différence significative*, soit faible. Toute cette analyse ne nous informe pas beaucoup.

Une analyse de puissance post hoc avec la taille d'effet observé n'est pas très utile. On la fera plutôt pour une taille d'effet autre que celle observée quand H_0 est acceptée. *Quelle taille d'effet utiliser?* C'est la biologie du système étudié qui peut nous guider. Par exemple, en ce qui concerne la biomasse des poissons, on pourrait s'attendre à ce qu'une différence de biomasse du simple au double (disons de 2.64 à $5.28 g/m^2$) ait des conséquences écologiques. On voudrait s'assurer que Jaynie avait de bonnes chances de détecter une différence aussi grande que celle-là avant d'accepter ses conclusions que la biomasse est la même entre les deux régions. Quelles étaient les chances de Jaynie de détecter une différence de $2.64 g/m^2$ entre les deux régions? G*Power peut nous le dire.

🔥 Exercice

Changer la moyenne du groupe 2 à 5.28, recalculer la taille d'effet, et cliquer sur **Calculate** pour obtenir Figure 8.3.

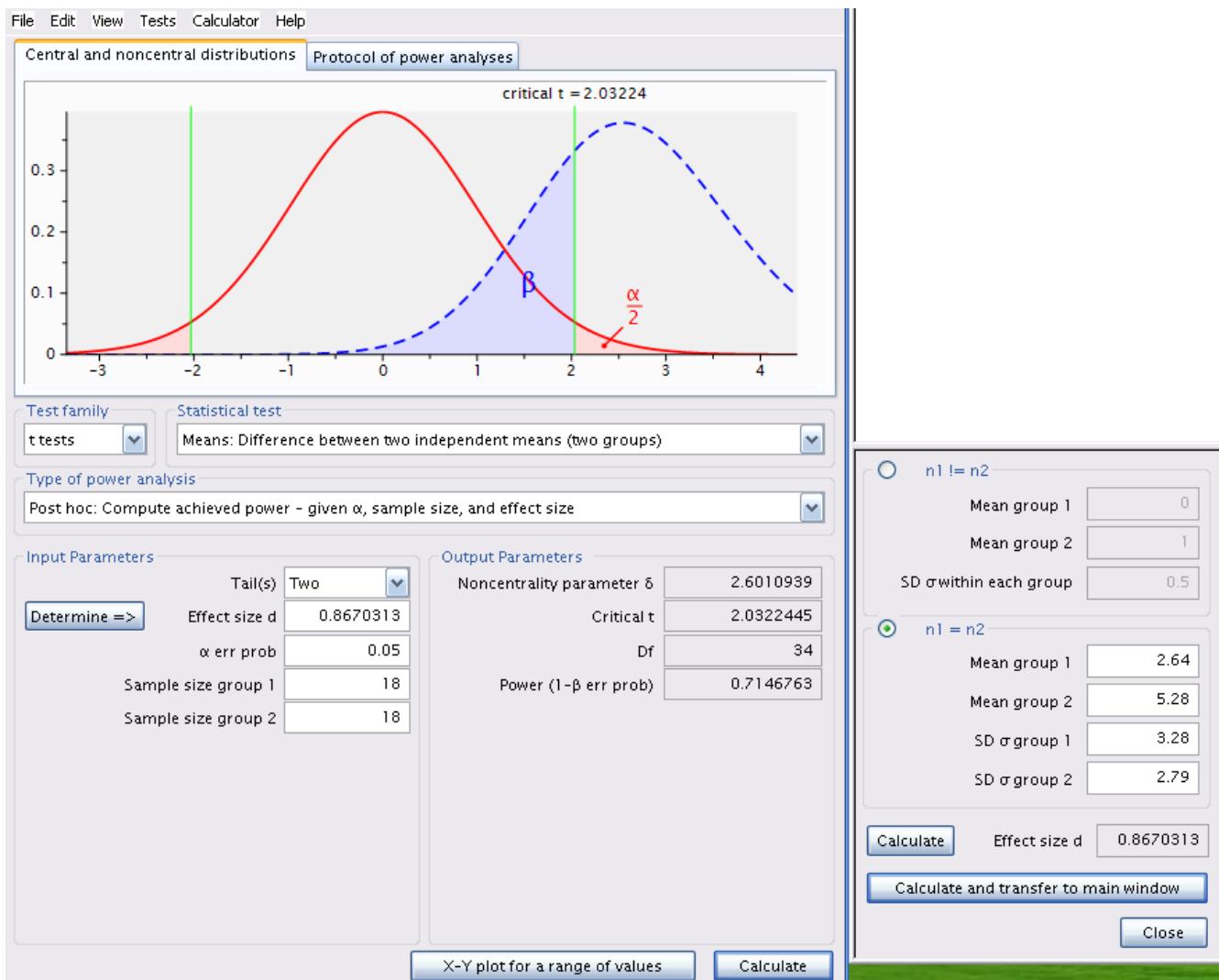


Figure 8.3.: Analyse post-hoc avec une taille d'effet différente

```
pwr.t.test(
  n = 18, d = d(u1 = 2.64, sd1 = 3.28, u2 = 5.28, sd2 = 2.79),
  sig.level = 0.05, type = "two.sample"
)
```

Two-sample t test power calculation

```
n = 18
d = 0.8670313
sig.level = 0.05
power = 0.7146763
alternative = two.sided
```

NOTE: n is number in *each* group

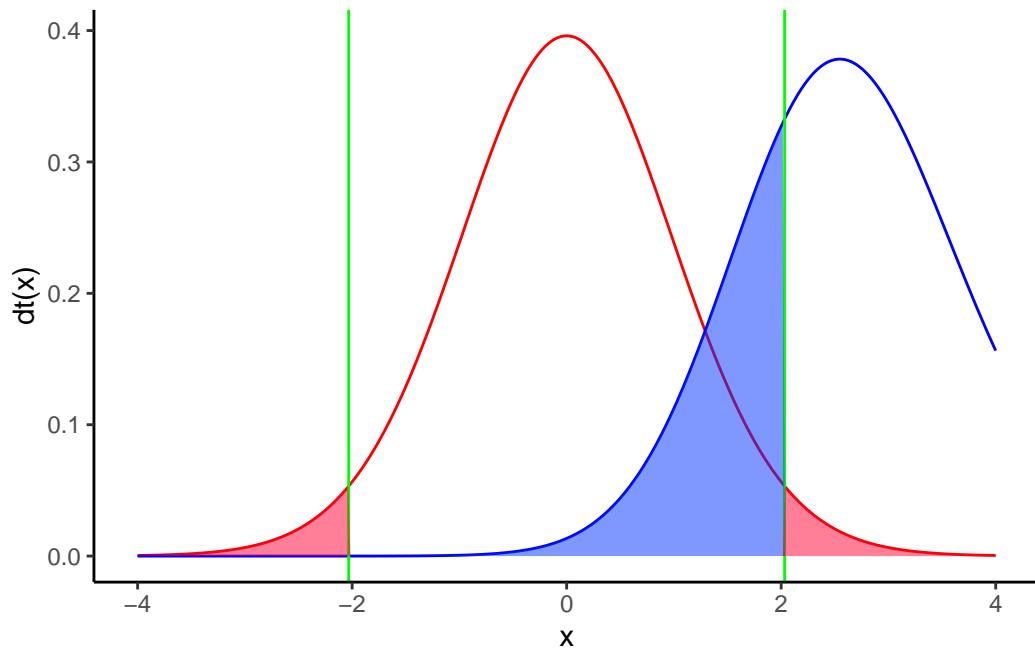


Figure 8.4.

La puissance est de 0.71, donc Jaynie avait une chance raisonnable (71%) de détecter une différence du simple au double avec 18 ruisseaux dans chaque région.

Notez que cette analyse de puissance post hoc pour une taille d'effet jugée biologiquement significative est bien

plus informative que l'analyse précédente pour la taille d'effet observée (qui est celle effectuée par défaut par bien des néophytes et de trop nombreux logiciels qui essaient de penser pour nous). En effet, Jaynie n'a pu détecter de différences significatives entre les deux régions. Cela pourrait être pour deux raisons: soit qu'il n'y a pas de différences entre les régions, ou alors parce que la précision des mesures est si faible et l'effort d'échantillonnage si limité qu'il était très peu probable de détecter même d'énormes différences. La deuxième analyse de puissance permet d'éliminer cette seconde possibilité puisque Jaynie avait 71% des chances de détecter une différence du simple au double.

8.4.2. Analyse à priori - Calculer la taille de l'effectif à échantillonner

Supposons qu'on puisse défendre la position qu'une différence de biomasse observée par Jaynie entre les deux régions, $3.31 - 2.64 = 0.67 g/m^2$, soit écologiquement significative. On devrait donc planifier la prochaine saison d'échantillonnage de manière à avoir de bonnes chances de détecter une différence de cette taille. Combien de ruisseaux Jaynie devrait-elle échantillonner pour avoir 80% des chances de la détecter (compte tenu de la variabilité observée)?

🔥 Exercice

Changer le type d'analyse de puissance dans G*Power à **A priori: Compute sample size - given α , power, and effect size**. Assurez-vous que les valeurs pour les moyennes et les écarts-type soient celles qu'a obtenu Jaynie, recalculez la taille de l'effet, et inscrivez 0.8 pour la puissance Figure 8.5.

```
pwr.t.test(
  power = 0.8, d = d(u1 = 2.64, sd1 = 3.28, u2 = 3.31, sd2 = 2.79),
  sig.level = 0.05, type = "two.sample"
)
```

```
Two-sample t test power calculation
```

```
n = 325.1723
d = 0.220042
sig.level = 0.05
power = 0.8
alternative = two.sided
```

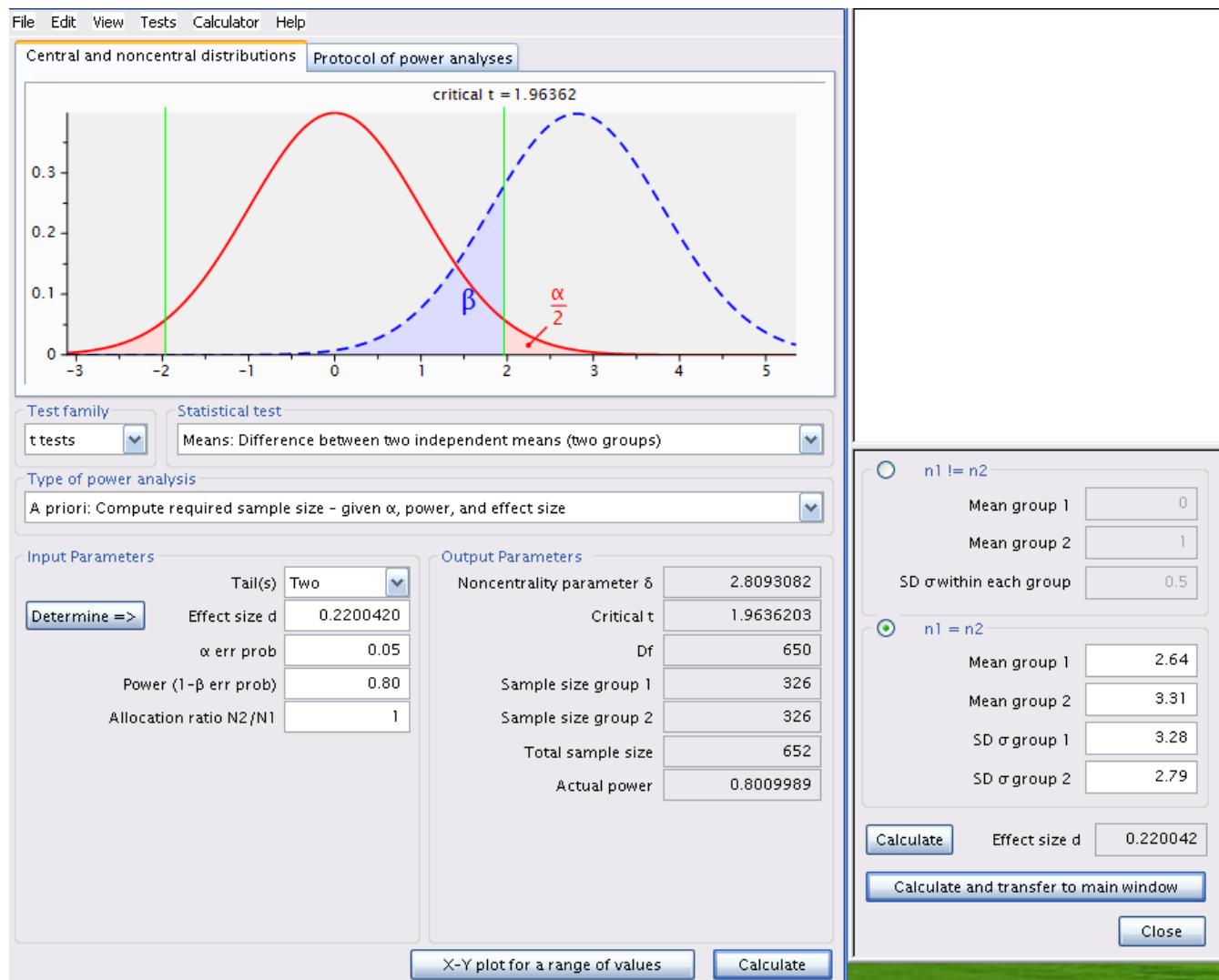


Figure 8.5.: Analyse à priori

NOTE: n is number in *each* group

Ouch! Il faudrait échantillonner 326 ruisseaux dans chaque région! Cela coûterait une fortune et exigerait de nombreuses équipes de travail. Sans cela, on ne pourrait échantillonner que quelques dizaines de ruisseaux, et il serait peu probable que l'on puisse détecter une si faible différence de biomasse entre les deux régions. Ce serait vraisemblablement en vain et pourrait être considéré comme une perte de temps: pourquoi tant d'efforts et de dépenses si les chances de succès sont si faibles.

Si on refait le même calcul pour une puissance de 95%, on obtient 538 ruisseaux par région. Augmenter la puissance ça demande plus d'effort.

```
pwr.t.test(
  power = 0.95, d = d(u1 = 2.64, sd1 = 3.28, u2 = 3.31, sd2 = 2.79),
  sig.level = 0.05, type = "two.sample"
)
```

Two-sample t test power calculation

```
n = 537.7286
d = 0.220042
sig.level = 0.05
power = 0.95
alternative = two.sided
```

NOTE: n is number in *each* group

8.4.3. Analyse de sensibilité - Calculer la taille d'effet détectable

Compte tenu de la variabilité observée, d'un effort d'échantillonnage de 18 ruisseaux par région, et en conservant $\alpha = 0.05$, quelle est la taille d'effet que Jaynie pouvait détecter avec 80% de chances ($\beta = 0.2$) ?

 Exercice

Changez le type d'analyse dans G*Power à **Sensitivity: Compute required effect size - given α , power, and sample size** et assurez-vous que la taille des échantillons est de 18 dans chaque région.

```
pwr.t.test(power = 0.8, n = 18,
            sig.level = 0.05, type = "two.sample")
```

Two-sample t test power calculation

```
n = 18
d = 0.9612854
sig.level = 0.05
power = 0.8
alternative = two.sided
```

NOTE: n is number in *each* group

La taille d'effet détectable pour cette taille d'échantillon, $\alpha = 0.05$ et $\beta = 0.2$ (ou une puissance de 80%) est de 0.961296.

 Avertissement

cette valeur est l'indice d de la taille d'effet et est pondérée par la variabilité des mesures.

Dans ce cas ci, d est approximativement égal à

$$d = \frac{|\bar{X}_1 - \bar{X}_2|}{\sqrt{\frac{s_1^2 + s_2^2}{2}}}$$

Pour convertir cette valeur de d sans unités en une valeur de différence de biomasse détectable (i.e $|\bar{X}_1 - \bar{X}_2|$), il suffit de multiplier d par le dénominateur de l'équation.

$$|\bar{X}_1 - \bar{X}_2| = d * \sqrt{\frac{s_1^2 + s_2^2}{2}}$$

Dans R, on peut estimer cela avec:

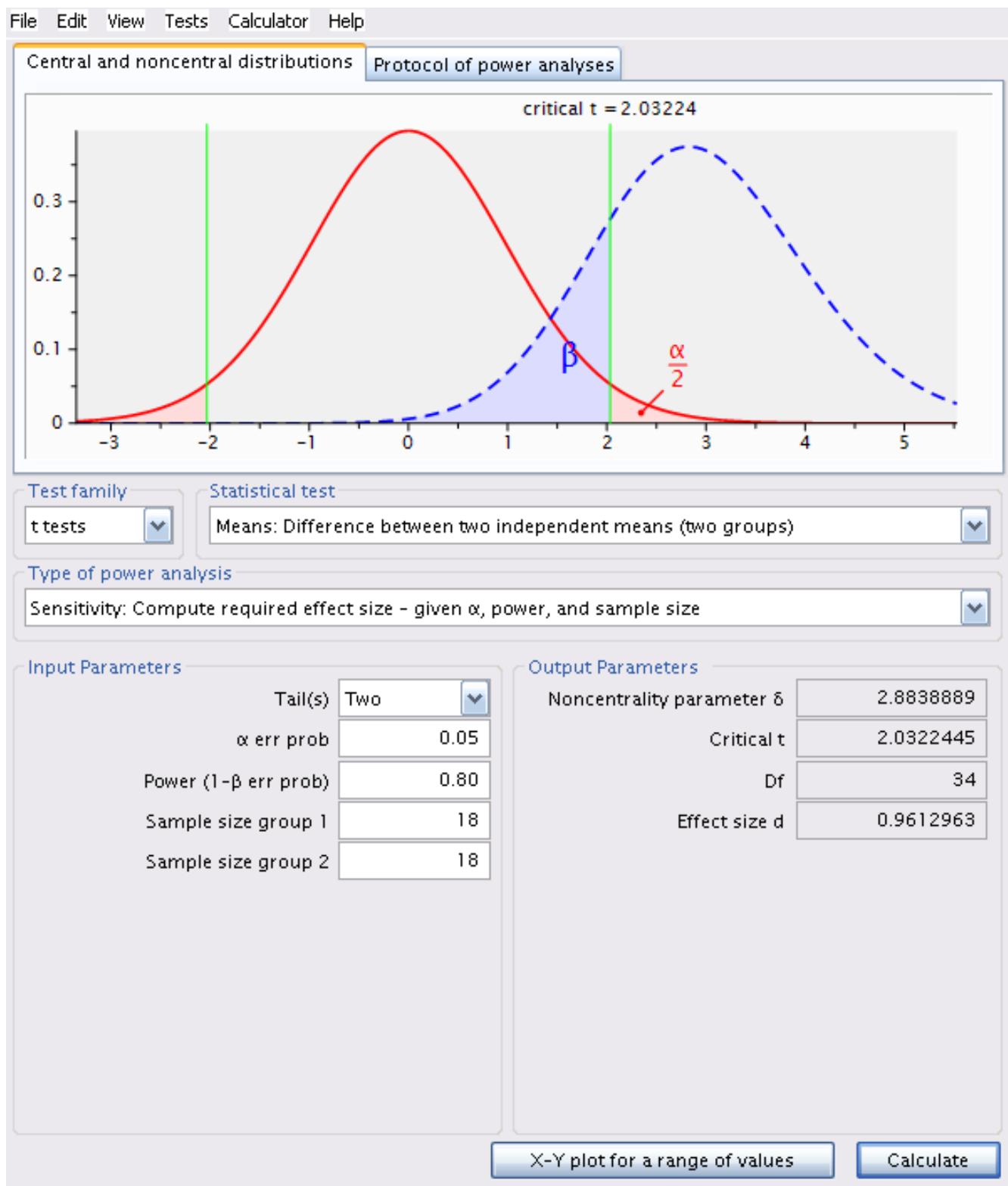


Figure 8.6.: Analyse de sensibilité

```
pwr.t.test(  
  power = 0.8, n = 18, sig.level = 0.05,  
  type = "two.sample")$d * sqrt((3.28^2 + 2.79^2) / 2)
```

```
[1] 2.926992
```

Donc, avec 18 ruisseaux dans chaque région, pour $\alpha = 0.05$ et $\beta = 0.2$ (une puissance de 80%), Jaynie pouvait détecter une différence de biomasse de $2.93g/m^2$ entre les régions, un peu plus que du simple au double.

8.5. Points à retenir

- L'analyse de puissance post hoc n'est pertinente que lorsque l'on a accepté l'hypothèse nulle. Il est en effet impossible de faire une erreur de type II quand on rejette H_0 .
- Avec de très grands échantillons, on a une puissance quasi infinie et on peut détecter statistiquement de très petites différences qui ne sont pas nécessairement biologiquement significatives.
- En utilisant un critère de signification plus strict ($\alpha < 0.05$) on diminue notre puissance.
- En voulant maximiser la puissance, on augmente l'effort requis, à moins d'utiliser une valeur critique plus libérale ($\alpha > 0.05$)
- Le choix de β est quelque peu arbitraire. On considère que $\beta = 0.2$ (puissance de 80%) est relativement élevé.

partie III.

Modèles linéaires

Chapitre 9

Corrélation et régression linéaire simple

Objectifs de ce chapitre :

- Utiliser R pour produire un diagramme de dispersion de la relation entre deux variables.
- Utiliser R pour faire des transformations simples de données.
- Utiliser R pour calculer le coefficient de corrélation de *Pearson* entre deux variables et évaluer sa significativité statistique.
- Utiliser R pour calculer la corrélation entre des paires de variables avec le *r de Spearman* et le *tau* (τ) de Kendall
- Utiliser R pour évaluer la significativité d'une comparaison par paires à partir d'une matrice de corrélation généralisé en utilisant les probabilités ajustées par la *méthode de Bonferroni*.
- Utiliser R pour faire une régression linéaire simple.
- Utiliser R pour évaluer si un ensemble de données remplit les conditions d'application d'une analyse de régression simple.
- Quantifier la taille d'effet d'une régression simple et effectuer une analyse de puissance avec G*Power.

9.1. Paquets R et données requises

Pour ce laboratoire vous aurez besoin de :

- paquets R :

- car 
- lmtest 
- boot 

- pwr 
- ggplot 

- Jeu de données :

- “sturgeon.csv”

Il ne faut pas oublier de charger les paquets avec `library()` (et de les installer au besoin avec `install.packages()`).

Pour les données, il faut les charger et les assigner à un objet R avec la fonction `read.csv()`.

```
library(car)
library(lmtest)
library(boot)
library(ggplot2)
library(pwr)
library(performance)

esturgeon <- read.csv("data/sturgeon.csv")
```

Note

Notez que la ligne de code pour charger les données considère que le fichier de données se trouve dans un dossier `data` au sein de votre répertoire de travail. Si ce n'est pas le cas veuillez ajuster la ligne de commande selon votre répertoire de travail.

9.2. Diagrammes de dispersion

Les analyses de corrélation et de régression devraient toujours commencer par un examen des données; c'est une étape critique qui sert à évaluer si ce type d'analyse est approprié pour un ensemble de données. Supposons que nous voulons évaluer si la longueur d'esturgeons mâles dans la région de *The Pas* covarie avec leur poids. Pour répondre à cette question, regardons d'abord la corrélation entre la longueur et le poids. Souvenez-vous qu'une des conditions d'application de l'analyse de corrélation est que la relation entre les deux variables est linéaire. Pour évaluer cela, commençons par faire un diagramme de dispersion.

- Les données sur les esturgeons sont disponibles dans le fichier `sturgeon.csv`. Après avoir chargé les données en les assignant à un objet `esturgeon`, faites un diagramme de dispersion avec une droite de régression linéaire et une courbe de `Lowess` de la longueur en fonction du poids.

```
esturgeon <- read.csv("data/sturgeon.csv")
str(esturgeon)
```

```
'data.frame': 186 obs. of 9 variables:
 $ fklnngth : num 37 50.2 28.9 50.2 45.6 ...
 $ totlngth: num 40.7 54.1 31.3 53.1 49.5 ...
 $ drlngth : num 23.6 31.5 17.3 32.3 32.1 ...
 $ rdwght   : num 15.95 NA 6.49 NA 29.92 ...
 $ age      : int 11 24 7 23 20 23 20 7 23 19 ...
 $ girth    : num 40.5 53.5 31 52.5 50 54.2 48 28.5 44 39 ...
 $ sex      : chr "MALE" "FEMALE" "MALE" "FEMALE" ...
 $ location: chr "THE_PAS" "THE_PAS" "THE_PAS" "THE_PAS" ...
 $ year     : int 1978 1978 1978 1978 1978 1978 1978 1978 1978 ...
```

```
mongraph <- ggplot(
  data = esturgeon[!is.na(esturgeon$rdwght), ], # origine des données
  aes(x = fklnngth, y = rdwght)
)

# Représentez les données sous forme de Points, Régression linéaire, "Lowess"
mongraph <- mongraph +
  geom_smooth(method = lm, se = FALSE, color = "green") + # Ajoutez une régression linéaire, mais
  geom_smooth(color = "red", se = FALSE) + # Ajoutez la "lowess" (en rouge)
  geom_point() + # Ajoutez les données sous forme de points
  labs(x = "Longueur", y = "Poids") # Modifiez les noms des axes pour rendre le graphique plus lisible

mongraph # Affichez le graphique
```

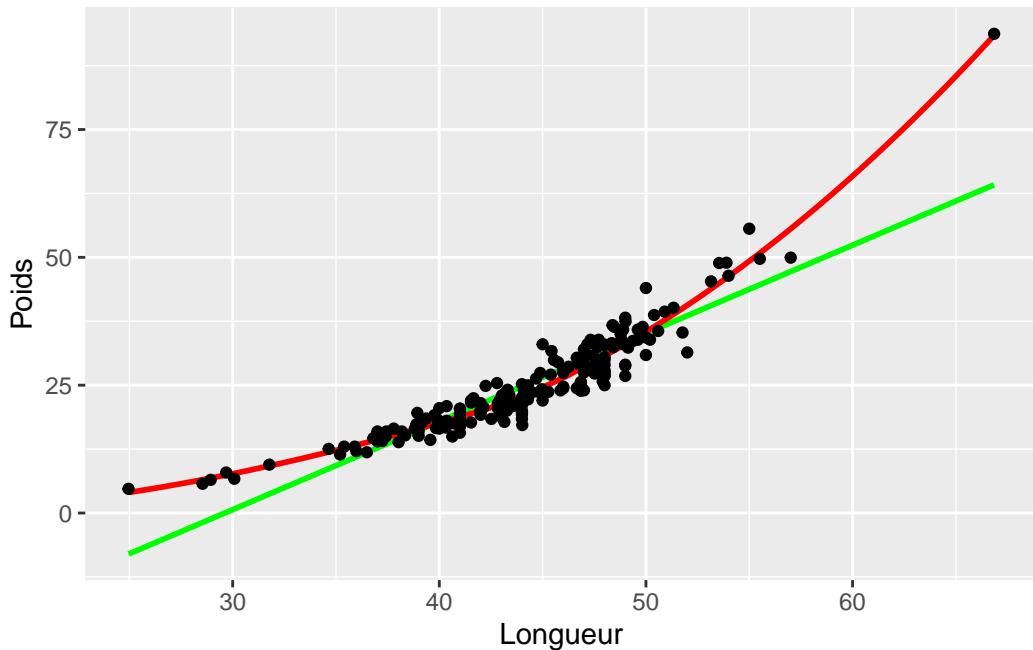


Figure 9.1.: Graphique du poids en fonction de la longueur des esturgeons.

- Est-ce que la dispersion des points suggère une bonne corrélation entre les deux variables? Est-ce que la relation semble linéaire?

Ce graphique suggère une tendance plus curvilinéaire que linéaire. Malgré tout, il semble y avoir une forte corrélation entre les deux variables.

- Refaites le diagramme de dispersion avec des transformations logarithmiques sur les deux axes.

```
# Appliquez une transformation log sur le graphique déjà défini
mongraph + scale_x_log10() + scale_y_log10()
```

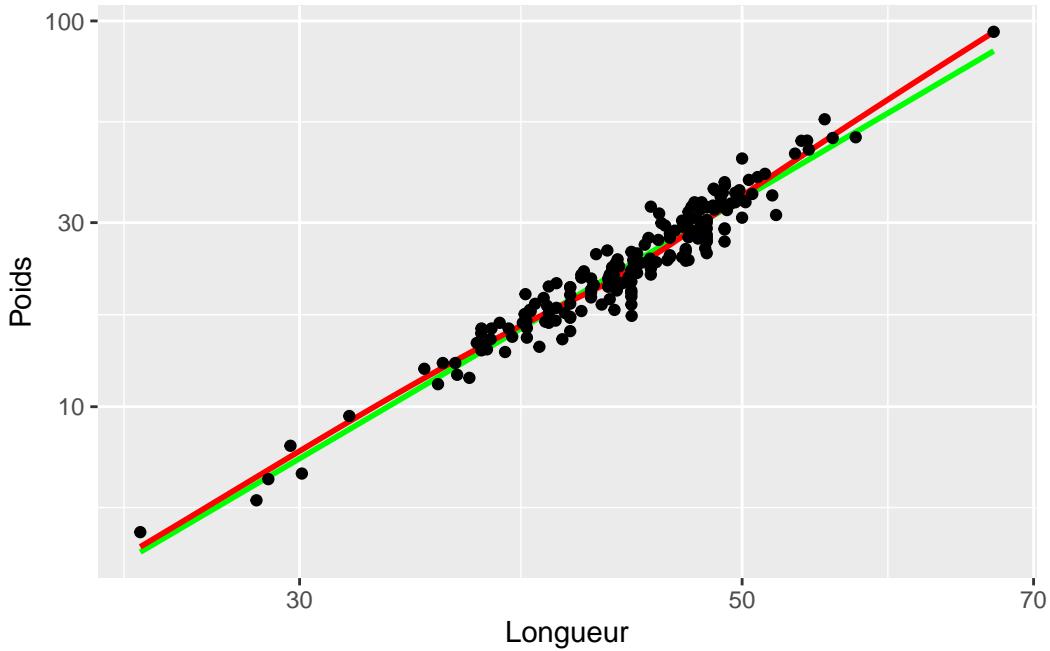


Figure 9.2.: Graphique poids-longueur avec une échelle log.

Comparez les diagrammes de dispersion avant et après transformation (Figure 9.1 et Figure 9.2). Sachant que l’analyse de corrélation presuppose une relation linéaire entre les variables, on devrait donc privilégier l’analyse sur les données log-transformées.

9.3. Transformations de données et coefficient de corrélation

Une autre condition préalable à l’analyse de corrélation est que les deux variables concernées suivent une distribution normale bivariée. On peut aisément vérifier la normalité de chacune des 2 variables séparément tel que décrit dans plus bas dans la section Section 9.6.1. Si les deux variables sont normalement distribuées, on présume généralement que la distribution commune des deux variables est également normale (notez que ce n’est pas toujours le cas cependant).

- Examinez la distribution des quatre variables (les deux variables originales et les variables transformées). Que concluez-vous de l’inspection visuelle de ces graphiques ?

Les figures ci-dessous sont les 4 diagrammes Q(uantile)-Q(uantile) (`qqplot()`). Le code pour produire des graphiques multiples sur une seule page avec 2 lignes et 2 colonnes, comme on voit ci-dessous, est :

```

par(mfrow = c(2, 2)) # divise le graphique en 4 sections
qqnorm(esturgeon$fklnth, ylab = "fklnth")
qqline(esturgeon$fklnth)
qqnorm(log10(esturgeon$fklnth), ylab = "log10(fklnth)")
qqline(log10(esturgeon$fklnth))
qqnorm(esturgeon$rdwght, ylab = "rdwght")
qqline(esturgeon$rdwght)
qqnorm(log10(esturgeon$rdwght), ylab = "log10(rdwght)")
qqline(log10(esturgeon$rdwght))
par(mfrow = c(1, 1)) # redéfinie la zone de graphique par défaut

```

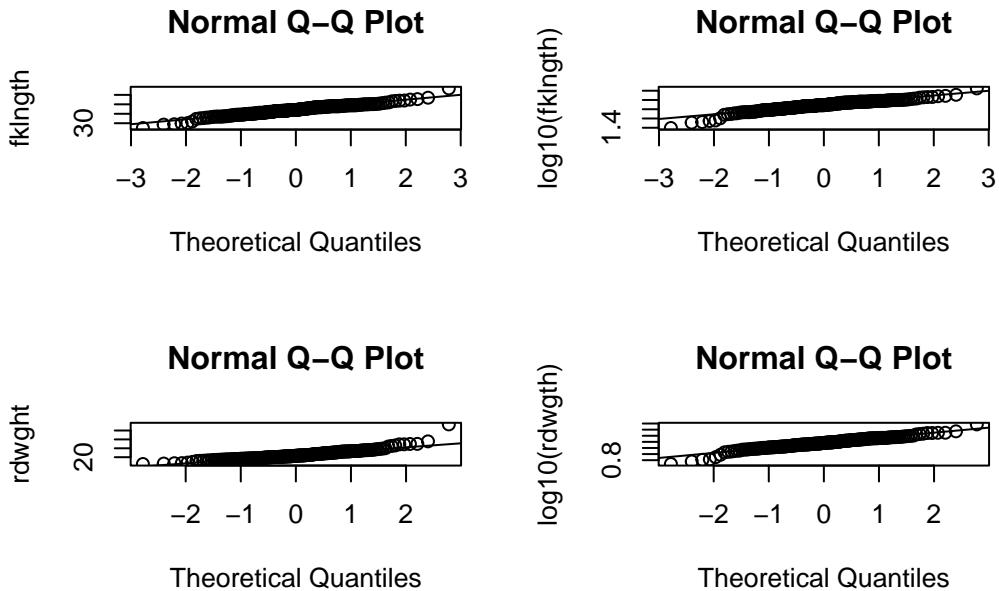


Figure 9.3.: Diagrammes Q-Q des quatres variables étudiées (Poids, Longueur, Originale et Log-transformées).

Il n'y a pas grand-chose à redire: aucune des distributions n'est parfaitement normale, mais les déviations semblent mineures.

- Générez une matrice de graphiques de dispersion de toutes les paires de variables (avec régression linéaires et “lowess”) en utilisant la fonction `scatterplotMatrix` du paquet car .

```

scatterplotMatrix(
  ~ fklnth + log10(fklnth) + rdwght + log10(rdwght),

```

```
data = esturgeon,
smooth = TRUE, diagonal = "density"
)
```

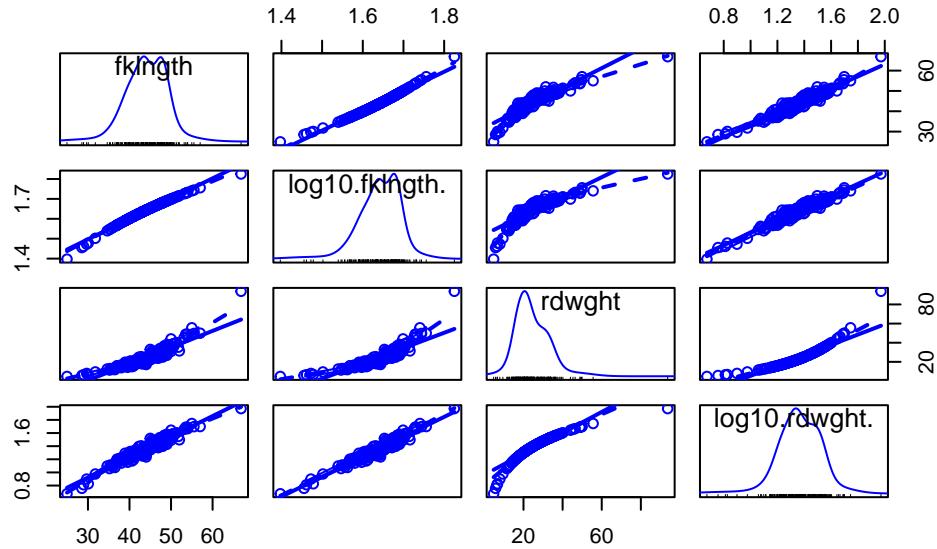


Figure 9.4.: Corrélations entre chaque paires de variables

- Ensuite, calculez le coefficient de corrélation de Pearson entre chaque paire (variables originales et log-transformées) en utilisant la fonction `cor()`. Avant de commencer, on va cependant ajouter les variables transformées au jeu de données `esturgeon` :

```
esturgeon$lfklnth <- with(esturgeon, log10(fklnth))
esturgeon$lrdwght <- log10(esturgeon$rdwght)
```

Vous pouvez ensuite obtenir la matrice de corrélation par :

```
cor(esturgeon[, c("fklnth", "lfklnth", "lrdwght", "rdwght")], use = "complete.obs")
```

Fréquemment, il y a des données manquantes dans un échantillon. En précisant `use="complete.obs"`, toutes les lignes du fichier pour lesquelles les variables ne sont pas toutes mesurées sont éliminées. Dans ce cas, toutes les corrélations seront calculées avec le même nombre de cas. Par contre, en utilisant `use="pairwise.complete.obs"`, R élimine une observation seulement lorsque l'un des deux membres de la paire a une valeur manquante. Dans ce cas, si les données manquantes pour différentes variables se retrouvent dans un groupe différent d'observation, les

corrélations ne seront pas nécessairement calculées sur le même nombre de cas ni sur le même sous-ensemble de cas. En général, vous devriez utiliser l'option `use="complete.obs"` à moins que vous ayez un très grand nombre de données manquantes et que cette façon de procéder élimine la plus grande partie de vos observations.

Pourquoi la corrélation entre les variables originales est-elle plus faible qu'entre les variables transformées ?

```
cor(esturgeon[, c("fklngth", "lfklnth", "lrdwght", "rdwght")], use = "complete.obs")
```

```
fklngth  lfklnth    lrdwght     rdwght
fklngth  1.0000000 0.9921435 0.9645108 0.9175435
lfklnth  0.9921435 1.0000000 0.9670139 0.8756203
lrdwght  0.9645108 0.9670139 1.0000000 0.9265513
rdwght   0.9175435 0.8756203 0.9265513 1.0000000
```

Il y a plusieurs choses à noter ici.

- Premièrement, la corrélation entre la longueur (“fklngth”) et le poids (“rdwght”) est élevée, peu importe la transformation: les poissons lourds ont tendance à être longs.
- Deuxièmement, la corrélation est plus forte pour les données transformées que pour les données brutes.

Pourquoi? Parce que le coefficient de corrélation est inversement proportionnel au bruit autour de la relation linéaire. Si la relation est curvilinéaire (comme dans le cas des données non transformées), le bruit est plus grand que si la relation est parfaitement linéaire. Par conséquent, la corrélation est plus faible.

9.4. Matrices de corrélations et correction de Bonferroni

Une pratique courante est d'examiner la matrice de corrélation à la recherche d'associations significatives. Comme exemple, essayons de tester si la corrélation entre `lfklnth` et `rdwght` est significative (le plus faible coefficient de corrélation de cette matrice).

- Estimez la corrélation entre la longueur (`fklngth`) et le poids (`rdwght`) des esturgeons:

```
cor.test(
  esturgeon$lfklnth, esturgeon$rdwght,
  alternative = "two.sided",
  method = "pearson"
)
```

Pearson's product-moment correlation

```
data: esturgeon$lfklngh and esturgeon$rdwght
t = 24.322, df = 180, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
0.8367345 0.9057199
sample estimates:
cor
0.8756203
```

On voit ici que la corrélation est hautement significative ($p < 2.2e - 16$), ce qui n'est pas surprenant étant donné la valeur du coefficient de corrélation ($0.8756 = 87.6\%$). Il est important de réaliser que si une matrice contient un grand nombre de corrélations, il n'est pas surprenant d'en trouver au moins une qui soit "significative". En effet, on s'attend à en trouver 5% en moyenne lorsqu'il n'y a en fait aucune corrélation entre les paires de moyennes (correspondant au risque d'erreur de type I α). Une façon de corriger pour cette tendance est d'ajuster le niveau α critique auquel on attribue une signification statistique en divisant α par le nombre k de corrélations qui sont examinées :

$$\alpha' = \alpha/k \text{ (ajustement de Bonferroni).}$$

Si initialement $\alpha = 0.05$ et qu'il y a 10 corrélations qui sont examinées, alors $\alpha' = 0.005$. Donc, afin de rejeter l'hypothèse nulle, la valeur de p devra être plus petite que α' , en l'occurrence 0.005. Dans l'exemple qui précède, on devrait donc ajuster α critique en divisant par le nombre total de corrélations dans la matrice (6 dans ce cas, donc $\alpha' = 0.00833$). Cette correction modifie-t-elle votre conclusion quant à la corrélation entre `lfklngh` et `rdwght`?

9.5. Corrélations non paramétriques: r de Spearman et τ de Kendall

L'analyse faite dans la section précédente avec les esturgeons suggère que l'une des conditions préalables à l'analyse de corrélation, soit la distribution normale bivariée de données, pourrait ne pas être remplie pour `fklngth` et `rdwght`, ni pour les paires de variables transformées. La recherche d'une transformation appropriée peut parfois être difficile. Pire encore, pour certaines distributions il n'existe pas de transformation qui va normaliser les données. Dans ces cas-là, la meilleure option est de faire une analyse non paramétrique qui ne présume ni de la normalité ni de la

linéarité. Ces analyses sont basées sur les rangs. Les deux plus communes sont le coefficient de rang de Spearman et le τ (tau) de Kendall.

- Dans R, testez la corrélation entre `fklngth` et `rdwght` en utilisant Spearman et Kendall's .

```
cor.test(  
  esturgeon$lfklnth, esturgeon$rdwght,  
  alternative = "two.sided",  
  method = "spearman"  
)
```

Warning in cor.test.default(esturgeon\$lfklnth, esturgeon\$rdwght, alternative =
"two.sided", : Cannot compute exact p-value with ties

```
Spearman's rank correlation rho  
  
data: esturgeon$lfklnth and esturgeon$rdwght  
S = 47971, p-value < 2.2e-16  
alternative hypothesis: true rho is not equal to 0  
sample estimates:  
rho  
0.9522546
```

```
cor.test(  
  esturgeon$lfklnth, esturgeon$rdwght,  
  alternative = "two.sided",  
  method = "kendall"  
)
```

```
Kendall's rank correlation tau  
  
data: esturgeon$lfklnth and esturgeon$rdwght  
z = 16.358, p-value < 2.2e-16
```

```
alternative hypothesis: true tau is not equal to 0
sample estimates:
tau
0.8208065
```

Comparer les résultats de cette analyse à l'analyse paramétrique. Pourquoi y-a-t'il une différence ?

Calculez les corrélations non paramétriques sur les paires de variables transformées. Vous devriez voir tout de suite que les corrélations des données transformées et non transformées sont identiques puisque dans les deux cas la corrélation est calculée à partir des rangs qui ne sont pas affectés par la transformation.

Notez que les corrélations obtenues avec le τ de Kendall (0.820) sont plus faibles que celles du coefficient de Spearman (0.952). Le τ de Kendall pondère un peu plus les grandes différences entre les rangs alors que le coefficient de Spearman donne le même poids à chaque paire d'observations. En général, on préfère le τ de Kendall lorsqu'il y a plus d'incertitude quant aux rangs qui sont près les uns des autres.

Les esturgeons de cet échantillon ont été capturés à l'aide de filets et d'hameçons d'une taille fixe. Quel impact cette méthode de capture peut-elle avoir eu sur la forme de la distribution de `fklngth` et `rdwght`? Compte tenu de ces circonstances, l'analyse de corrélation est-elle appropriée ?

Rappelez-vous que l'analyse de corrélation présume aussi que chaque variable est échantillonnée aléatoirement. Dans le cas de nos esturgeons, ce n'est pas le cas: les hameçons appâtés et les filets ne capturent pas de petits esturgeons (et c'est pourquoi il n'y en a pas dans l'échantillon). Il faut donc réaliser que les coefficients de corrélation obtenus dans cette analyse ne reflètent pas nécessairement ceux de la population totale des esturgeons.

9.6. Régression linéaire simple

L'analyse de corrélation vise à décrire comment deux variables covariant. L'analyse de régression vise plutôt à produire un modèle permettant de prédire une variable (la variable dépendante) par l'autre (la variable indépendante).

Comme pour l'analyse de corrélation, on devrait commencer en examinant des graphiques. Puisque l'on veut quantifier la relation entre deux variables, un graphique de la variable dépendante (Y) en fonction de la variable indépendante (X) est tout à fait approprié.

- Le fichier `sturgeon.csv` contient les données d'un inventaire d'esturgeons récoltés entre 1978 et 1980 à Cumberland House en Saskatchewan et à The Pas au Manitoba. Faites un diagramme de dispersion de `fklngth` (la variable dépendante, Y) en fonction de l'âge (la variable indépendante, X) pour les esturgeons mâles

uniquement et ajoutez-y une régression linéaire et une “lowess”. Que concluez-vous de ce diagramme de dispersion ?

```
esturgeon.male <- subset(esturgeon, subset = sex == "MALE")
mongraph <- ggplot(
  data = esturgeon.male, # origine des données
  aes(x = age, y = fklngh)
) # aesthetics: y=fklngh, x=rdwght
# Représentez les données sous forme de Points, Régression linéaire, "Lowess"
mongraph <- mongraph +
  geom_smooth(method = lm, se = FALSE, color = "green") + # Ajoutez une régression linéaire, mais
  geom_smooth(color = "red") + # Ajoutez la "lowess" (en rouge)
  geom_point() +# Ajoutez les données sous forme de points
  labs(x = "Age", y = "Longueur") # Modifiez les noms des axes pour rendre le graphique plus lisible
mongraph # Affichez le graphique
```

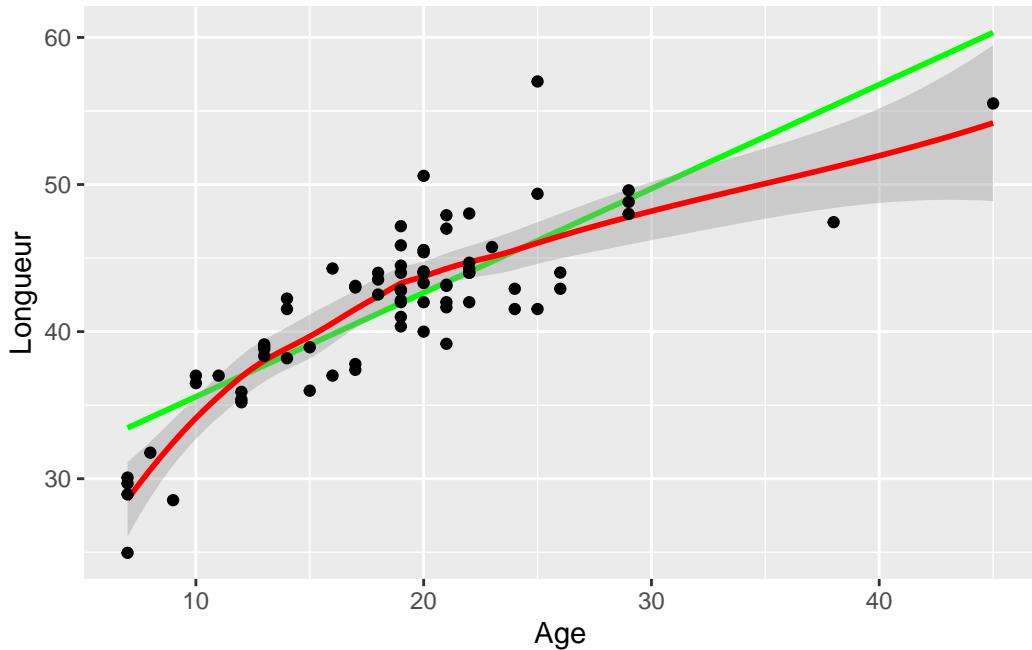


Figure 9.5.: Graphique de la longueur en fonction de l'âge des esturgeons mâles.

Ce graphique suggère que la relation n'est pas linéaire.

Supposons que nous désirions estimer le taux de croissance des esturgeons mâles. Un estimé (peut-être pas terrible...) du taux de croissance peut être obtenu en calculant la pente de la régression de la longueur en fonction de l'âge.

Ajustons d'abord une régression avec la commande `lm()` et sauvons ces résultats dans un objet appelé `RegModel.1`.

```
RegModele.1 <- lm(fklngh ~ age, data = esturgeon.male)
```

Rien n'apparaît à l'écran, c'est normal ne vous inquiétez pas, tout a été sauvegardé en mémoire (nouvel objet visible dans l'onglet environnement). Pour voir les résultats, tapez:

```
summary(RegModele.1)
```

Call:

```
lm(formula = fklngh ~ age, data = esturgeon.male)
```

Residuals:

| Min | 1Q | Median | 3Q | Max |
|---------|---------|--------|--------|---------|
| -8.4936 | -2.2263 | 0.1849 | 1.7526 | 10.8234 |

Coefficients:

| | Estimate | Std. Error | t value | Pr(> t) | | | | | | | |
|----------------|----------|------------|---------|------------|------|-----|------|------|-----|-----|---|
| (Intercept) | 28.50359 | 1.16873 | 24.39 | <2e-16 *** | | | | | | | |
| age | 0.70724 | 0.05888 | 12.01 | <2e-16 *** | | | | | | | |
| --- | | | | | | | | | | | |
| Signif. codes: | 0 | '***' | 0.001 | '**' | 0.01 | '*' | 0.05 | '..' | 0.1 | ' ' | 1 |

Residual standard error: 3.307 on 73 degrees of freedom

(5 observations deleted due to missingness)

Multiple R-squared: 0.664, Adjusted R-squared: 0.6594

F-statistic: 144.3 on 1 and 73 DF, p-value: < 2.2e-16

la sortie R donne:

1. Call: Un petit rappel du modèle qui a été ajusté et des données utilisées.
2. Residuals: Un sommaire statistique des résidus autour du modèle estimé.
3. Coefficients: Valeurs estimées des paramètres du modèle, erreurs-types, statistiques t et probabilités associées.

4. **Residual standard error:** Racine carrée de la variance résiduelle.
5. **Multiple R-squared:** Coefficient de détermination. Il correspond à la proportion de la variabilité de la variable dépendante qui peut être expliquée par la régression.
6. **Adjusted R-squared:** Le R-carré ajusté tient compte du nombre de paramètres du modèle. Si vous voulez comparer différents modèles qui n'ont pas le même nombre de paramètres, c'est ce qu'il faut utiliser.
7. **F-statistic:** C'est le test de signification omnibus du modèle. Dans le cas de la régression simple, il est équivalent au test sur la pente de la régression.

La régression estimée est donc:

$$Fklngh = 28.50359 + 0.70724 * age$$

Étant donné la valeur significative du test de F (ainsi que pour le test de t pour la pente de la droite), on rejette l'hypothèse nulle qu'il n'y a pas de relation entre la taille et l'âge.

9.6.1. Vérifier les conditions d'application de la régression

La régression simple de type I a quatre conditions préalables :

1. il n'y a **pas d'erreur de mesure** sur la variable indépendante (X)
2. la relation entre Y et X est **linéaire**
3. les résidus sont **normalement distribués**
4. la variance des résidus est constante pour toutes les valeurs de la variable indépendante (**homoscedasticité**)

Procédons maintenant à l'examen post-mortem. La première condition est rarement remplie avec des données biologiques ; il y presque toujours de l'erreur sur X et sur Y. Cela veut dire qu'en général les pentes estimées sont biaisées, mais que les valeurs prédites ne le sont pas. Toutefois, si l'erreur de mesure sur X est petite par rapport à l'étendue des valeurs de X, le résultat de l'analyse n'est pas dramatiquement influencé. Par contre, si l'erreur de mesure est relativement grande (toujours par rapport à l'étendue des valeurs de X), la droite de régression obtenue par la régression de modèle I est un piètre estimé de la relation fonctionnelle entre X et Y. Dans ce cas, il est préférable de passer à la régression de modèle II, malheureusement au-delà du contenu de ce cours. Les autres conditions préalables à l'analyse de régression de modèle I peuvent cependant être vérifiées, ou du moins évaluées visuellement. La fonction `plot()` (appliquée directement sur un objet de type `model`) permet de visualiser des graphiques diagnostiques pour des modèles linéaires.

```
par(mfrow = c(2, 2), las = 1)
plot(RegModele.1)
```

La fonction `par()` est utilisée pour dire à R de tracer 2 rangées et 2 colonnes de graphiques par page (il y a quatre graphiques diagnostiques qui sont générés automatiquement pour les modèles linéaires), et l'argument `las = 1` indique à R d'effectuer une rotation des légendes des axes Y pour qu'elles soient perpendiculaires à l'axe (oui. Je sais. Rien de tout ça n'est évident.)

Vous obtiendrez:

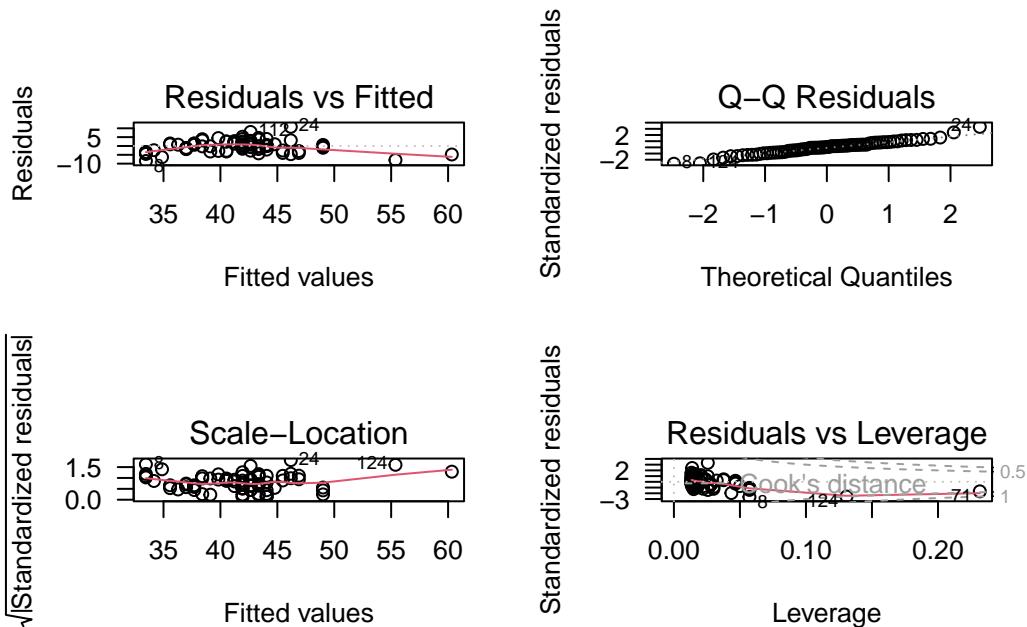


Figure 9.6.: Graphiques diagnostiques du modèle.

1. **Le graphique en haut à gauche**, permet d'évaluer la **linéarité**, la **normalité**, et l'**homoscédasticité** des résidus. Il illustre les déviations autour de la régression en fonction des valeurs prédictes. Rappelez-vous que le graphique de `fklngth` vs `age` suggère que la relation entre la longueur et l'âge n'est pas linéaire. Les très jeunes et très vieux esturgeons sont sous la droite en général, alors que les esturgeons d'âge moyen sont retrouvés généralement au-dessus de la droite de régression. C'est exactement ce que le graphique des résidus en fonction des valeurs prédictes illustre. La ligne en rouge est une trace "lowess" au travers de ce nuage de points. Si la relation était linéaire, la trace "lowess" serait presque plate et près de 0. La dispersion des résidus permet d'évaluer visuellement leur normalité et hétéroscédasticité; mais ce graphique n'est pas optimal pour évaluer ces propriétés. Les deux graphiques suivants sont mieux pour cela.
2. Le graphique en haut à droite permet d'évaluer la **normalité** des résidus. C'est un graphique QQ des résidus

(QQ plot). Des résidus distribués normalement tomberaient exactement sur la diagonale. Ici, on voit que c'est presque le cas, sauf dans les queues de la distribution.

3. **Le graphique en bas à gauche**, intitulé Scale-Location, permet d'évaluer l'**homoscédasticité**. On y retrouve sur l'ordonnée (l'axe des y) la racine carrée de la valeur absolue des résidus standardisés (résidus divisés par l'écart-type des résidus) en fonction des valeurs prédictes. Le graphique aide à déterminer si la variation des résidus est constante ou non. Si les résidus sont homoscédastiques, la valeur moyenne sur l'axe des y ne va pas changer en fonction de la valeur prédictée. Ici, il y a une certaine tendance, mais pas une tendance monotone puisqu'il y a d'abord une baisse puis une hausse.; bref, rien qui soit une forte évidence contre la supposition d'homoscédasticité.
4. **Le graphique en bas à droite**, montre les résidus en fonction du "leverage" et permet de détecter certaines **valeurs extrêmes** qui ont une grande influence sur la régression. Le leverage d'un point mesure sa distance des autres points, mais seulement en ce qui concerne les variables indépendantes. Dans le cas d'une régression simple, cela revient à la distance entre le point sur l'axe des x et la moyenne de tous les points sur cet axe. Vous devriez porter une attention particulière aux observations qui ont un leverage plus grand que $2(k + 1)/n$, où k est le nombre de variables indépendantes (ici, 1) et n est le nombre d'observations. Dans cet exemple, il y a 75 observations et une variable indépendante et donc les points ayant un leverage plus grand que $4/75 = 0.053$ devrait être considérés avec attention. Le graphique indique également comment la régression changerait si on enlevait un point. Ce changement est mesuré par la distance de Cook, illustrée par les bandes en rouge sur le graphique. Un point ayant une distance de Cook supérieure à 1 a une grande influence.

! Avertissement

Notez que R identifie automatiquement les cas les plus extrêmes sur chacun de ces 4 graphiques. Le fait qu'un point soit identifié ne signifie pas nécessairement que c'est une valeur réellement extrême, ou que vous devez vous en préoccuper. Dans tous les ensembles de données il y aura toujours un résidu plus grand que les autres...

Il est possible d'obtenir des graphiques d'évaluations des conditions d'applications, qui sont plus simple à interpréter et plus joli (avec des couleurs). On peut utiliser la fonction `check_model()` du paquet `performance` .

```
check_model(RegModele.1)
```

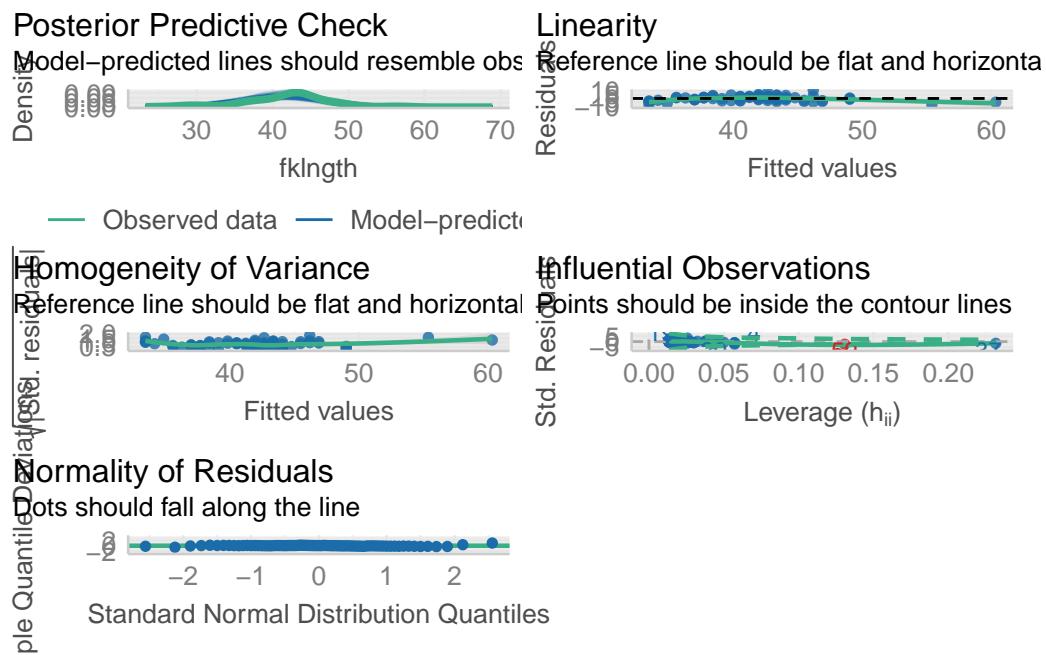


Figure 9.7.: Graphiques diagnostiques du modèle avec la fonction `check_model()`.

Finalement, quel est le verdict concernant la régression linéaire entre `fklength` et `age`? Elle viole la condition de linéarité, possiblement celle de normalité, remplit la condition d'homoscédasticité, et ne semble pas être influencée autre mesure par des valeurs bizarres ou extrêmes.

9.6.2. Tests formels des conditions d'application pour la régression

Personnellement, je n'utilise jamais les tests formels des conditions d'application de la régression et me contente des graphiques des résidus pour guider mes décisions. C'est ce que la plupart des praticiens font. Cependant, lors de mes premières analyses, je n'étais pas toujours certain de bien interpréter les graphiques et j'aurais aimé un indice plus formel ou un test permettant de détecter les violations des conditions d'application de la régression.

Le package `lmtest` (disponible sur CRAN), permet de faire plusieurs tests de linéarité et d'homoscédasticité. Et on peut tester la normalité avec le test Shapiro-Wilk vu précédemment.

Charger le package `lmtest` de CRAN (et installer le si besoin).

```
library(lmtest)
```

Exercice

Exécutez les commandes suivantes

```
bptest(RegModele.1)
```

studentized Breusch-Pagan test

```
data: RegModele.1
BP = 1.1765, df = 1, p-value = 0.2781
```

Le test Breusch-Pagan examine si la variabilité des résidus est constantes lorsque les valeurs prédictives changent. Une faible valeur de p suggère de l'hétérosécédasticité. Ici, la valeur p est élevée et suggère que la condition d'application d'homoscédasticité est remplie avec ces données.

```
dwtest(RegModele.1)
```

Durbin-Watson test

```
data: RegModele.1
DW = 2.242, p-value = 0.8489
alternative hypothesis: true autocorrelation is greater than 0
```

Le test Durbin-Watson permet de détecter l'autocorrélation sérielle des résidus. En l'absence d'autocorrélation (i.e. d'indépendance des résidus) la valeur attendue de la statistique D est 2. Ce test permet d'éprouver l'hypothèse d'indépendance des résidus, mais ne permet de détecter qu'un type particulier de dépendance. Ici, le test ne permet pas de rejeter l'hypothèse d'indépendance.

```
resettest(RegModele.1)
```

RESET test

```
data: RegModele.1
RESET = 14.544, df1 = 2, df2 = 71, p-value = 5.082e-06
```

Le test RESET permet d'éprouver la linéarité. Si la relation est linéaire, alors la statistique RESET sera d'environ 1. Ici, la statistique est beaucoup plus élevée (14.54) et hautement significative. Le test confirme la tendance que nous avons détectée visuellement plus haut: la relation n'est pas linéaire.

```
shapiro.test(residuals(RegModele.1))
```

```
Shapiro-Wilk normality test
```

```
data: residuals(RegModele.1)
W = 0.98037, p-value = 0.2961
```

Le test de normalité Shapiro-Wilk sur les résidus confirme que la déviation par rapport à une distribution normale des résidus n'est pas grande.

9.7. Transformation des données en régression

La relation entre `fklngth` et `age` n'étant pas linéaire, on devrait donc essayer de transformer les données pour tenter de les linéariser :

- Voyons ce qu'une transformation log donne:

```
par(mfrow = c(1, 1), las = 1)
ggplot(
  data = esturgeon.male,
  aes(x = log10(age), y = log10(fklngth))
) +
  geom_smooth(color = "red") +
  geom_smooth(method = "lm", se = FALSE, color = "green") +
  geom_point()
```

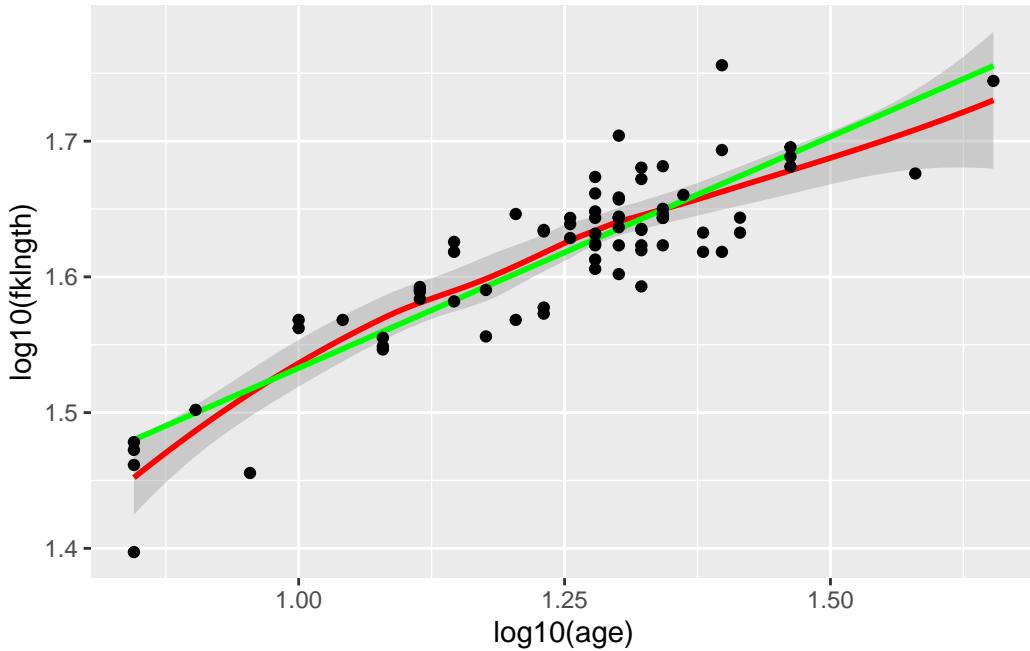


Figure 9.8.: Graphique du log de la longueur en fonction du log de l'âge des esturgeons males

Ajustons maintenant une régression simple sur ces données transformées.

```
RegModele.2 <- lm(log10(fklength) ~ log10(age), data = esturgeon.male)
summary(RegModele.2)
```

Call:

```
lm(formula = log10(fklength) ~ log10(age), data = esturgeon.male)
```

Residuals:

| Min | 1Q | Median | 3Q | Max |
|-----------|-----------|-----------|----------|----------|
| -0.082794 | -0.016837 | -0.000719 | 0.021102 | 0.087446 |

Coefficients:

| | Estimate | Std. Error | t value | Pr(> t) |
|----------------|----------|------------|----------|------------|
| (Intercept) | 1.19199 | 0.02723 | 43.77 | <2e-16 *** |
| log10(age) | 0.34086 | 0.02168 | 15.72 | <2e-16 *** |
| --- | | | | |
| Signif. codes: | 0 '***' | 0.001 '**' | 0.01 '*' | 0.05 '.' |
| | 0.1 ' ' | 1 | | |

Residual standard error: 0.03015 on 73 degrees of freedom

(5 observations deleted due to missingness)

Multiple R-squared: 0.772, Adjusted R-squared: 0.7688

F-statistic: 247.1 on 1 and 73 DF, p-value: < 2.2e-16

Examinons maintenant les graphiques diagnostiques:

```
par(mfrow = c(2, 2), las = 1)
plot(RegModele.2)
```

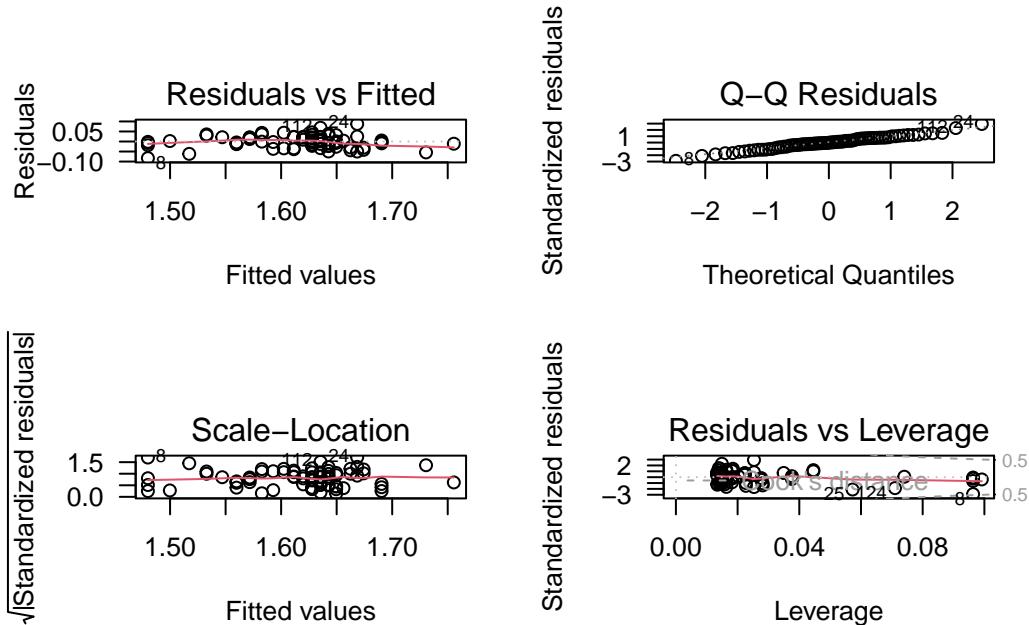


Figure 9.9.: Graphiques diagnostiques du modèle avec les données log-transformées.

Il y a une certaine amélioration, mais ce n'est pas encore parfait (la perfection n'est pas de ce monde....). Le graphique des résidus en fonction des valeurs prédictes suggère encore une certaine non linéarité. Sur le graphique Q-Q les points se retrouvent plus près de la droite diagonale qu'avant, indiquant que les résidus sont encore plus près de la normalité après la transformation log-log. Il n'y a pas d'indice d'hétérosécédasticité. Finalement, même si il reste quelques points avec plus d'influence (leverage) que les autres, aucun n'a une distance de Cook au-delà de 0.5. En résumé, la transformation log a amélioré les choses: la relation est plus linéaire, les résidus sont plus normaux, et il y a moins de points avec une influence relativement élevée. Est-ce que les tests formels supportent cette évaluation ?

```
bptest(RegModele.2)
```

studentized Breusch-Pagan test

```
data: RegModele.2  
BP = 0.14282, df = 1, p-value = 0.7055
```

```
dwtest(RegModele.2)
```

Durbin-Watson test

```
data: RegModele.2  
DW = 2.1777, p-value = 0.6134  
alternative hypothesis: true autocorrelation is greater than 0
```

```
resettest(RegModele.2)
```

RESET test

```
data: RegModele.2  
RESET = 4.4413, df1 = 2, df2 = 71, p-value = 0.01523
```

```
shapiro.test(residuals(RegModele.2))
```

Shapiro-Wilk normality test

```
data: residuals(RegModele.2)  
W = 0.98998, p-value = 0.8246
```

Oui, les conclusions sont les mêmes: les résidus sont encore homoscédastiques (test Breusch-Pagan), ne sont pas autocorrélés (test Durbin-Watson), sont normaux (test Shapiro-Wilk), et sont plus linéaires (la valeur de P du test RESET est maintenant 0.015, au lieu de 0.000005). Donc la linéarité a augmenté, mais cette condition d'application semble encore légèrement violée.

9.8. Traitement des valeurs extrêmes

Dans cet exemple, il n'y a pas de valeur vraiment extrême. Oui, je sais, R a quand même identifié les observations 8, 24, et 112 dans le dernier graphique diagnostique. Mais ces valeurs sont encore dans la fourchette de valeurs que je juge “acceptables”. Mais comment déterminer objectivement ce qui est acceptable ? À quel moment juge t'on qu'une valeur extrême est vraiment trop invraisemblable pour ne pas l'exclure ? Il n'y a malheureusement pas de règle absolue là-dessus. Les opinions varient, mais je penche vers le conservatisme sur cette question.

Ma position est que, à moins que la valeur soit biologiquement impossible ou clairement une erreur d'entrée de données, je n'élimine pas les valeurs extrêmes et j'utilise toutes mes données dans leur analyse. Pourquoi ?

Parce que je veux que mes données reflètent bien la variabilité naturelle ou réelle. C'est d'ailleurs parfois cette variabilité qui est intéressante.

L'approche conservatrice qui consiste à conserver toutes les valeurs extrêmes possibles est probablement la plus honnête, mais elle peut causer certains problèmes. Ces valeurs extrêmes sont souvent la cause des violations des conditions d'application des tests statistiques. La solution suggérée à ce dilemme est de faire l'analyse avec et sans les valeurs extrêmes et de comparer les conclusions. Dans bien des cas, les conclusions seront qualitativement les mêmes et les tailles d'effet ne seront pas très différentes. Toutefois, dans certains cas, la présence des valeurs extrêmes change complètement les conclusions. Dans ces cas, il faut simplement accepter que les conclusions dépendent entièrement de la présence des valeurs extrêmes et sont donc peu concluantes.

Suivant cette approche comparative, refaisons donc l'analyse après avoir enlevé les observations 8, 24, et 112.

```
RegModele.3 <- lm(log10(fklngh) ~ log10(age), data = esturgeon.male, subset = !(rownames(esturge  
summary(RegModele.3)
```

Call:

```
lm(formula = log10(fklngh) ~ log10(age), data = esturgeon.male,  
subset = !(rownames(esturgeon.male) %in% c("8", "24", "112")))
```

Residuals:

| Min | 1Q | Median | 3Q | Max |
|-----------|-----------|----------|----------|----------|
| -0.069163 | -0.017390 | 0.000986 | 0.018590 | 0.047647 |

Coefficients:

| | Estimate | Std. Error | t value | Pr(> t) |
|----------------|----------|------------|----------|------------|
| (Intercept) | 1.22676 | 0.02431 | 50.46 | <2e-16 *** |
| log10(age) | 0.31219 | 0.01932 | 16.16 | <2e-16 *** |
| --- | | | | |
| Signif. codes: | 0 '***' | 0.001 '**' | 0.01 '*' | 0.05 '.' |
| | 0.1 ' ' | 1 | | |

Residual standard error: 0.02554 on 70 degrees of freedom

(5 observations deleted due to missingness)

Multiple R-squared: 0.7885, Adjusted R-squared: 0.7855

F-statistic: 261 on 1 and 70 DF, p-value: < 2.2e-16

L'ordonnée à l'origine (Intercept), la pente, et le R carré sont presque les mêmes, et la valeur de p est encore astronomiquement petite. Enlever les valeurs extrêmes a peu d'effet dans ce cas.

Les graphiques diagnostiques des résidus et les tests formels des conditions d'application sur ce sous-ensemble de données donnent :

```
par(mfrow = c(2, 2))
plot(RegModele.3)
bptest(RegModele.3)
```

studentized Breusch-Pagan test

```
data: RegModele.3
BP = 0.3001, df = 1, p-value = 0.5838
```

```
dwtest(RegModele.3)
```

Durbin-Watson test

```
data: RegModele.3  
DW = 2.0171, p-value = 0.5074  
alternative hypothesis: true autocorrelation is greater than 0
```

```
resettest(RegModele.3)
```

RESET test

```
data: RegModele.3  
RESET = 3.407, df1 = 2, df2 = 68, p-value = 0.0389
```

```
shapiro.test(residuals(RegModele.3))
```

Shapiro-Wilk normality test

```
data: residuals(RegModele.3)  
W = 0.98318, p-value = 0.4502
```

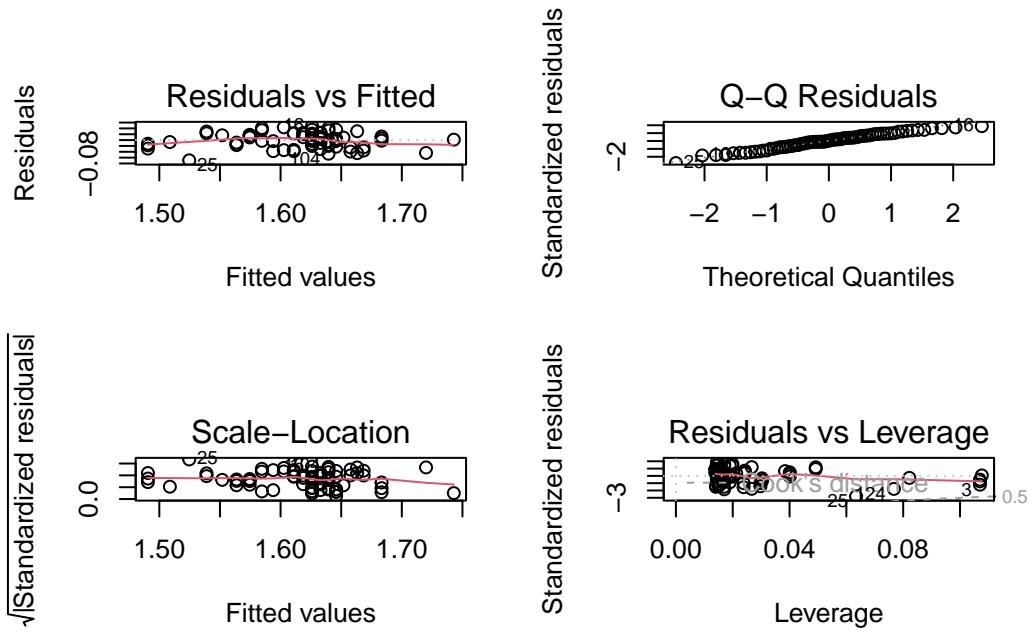


Figure 9.10.: Graphiques diagnostiques du modèle RegModele.3 excluant les données aberrantes.

Il n'y a pas vraiment de différence ici non plus avec l'analyse des données en entier. Bref, tout pointe vers la conclusion que les valeurs les plus extrêmes de cet ensemble de donnée n'influencent pas indûment les résultats statistiques.

9.9. Quantifier la taille d'effet et analyse de puissance en régression

L'interprétation biologique des résultats n'est pas la même chose que l'interprétation statistique. Dans l'analyse qui précède, on conclue statistiquement que la taille augmente avec l'âge (puisque la pente est positive et $p < 0.05$). Mais cette augmentation "statistique" de la taille avec l'âge ne donne pas d'informations sur la différence de taille entre les jeunes et vieux individus. La pente et un graphique sont plus informatifs à ce sujet que la valeur p . La pente (dans l'espace log-log) est de 0.34. Cela veut dire que pour chaque unité d'accroissement de X ($\log_{10}(\text{age})$), il y a une augmentation de 0.34 unités de $\log_{10}(\text{fklngth})$. En d'autres mots, quand l'âge est multiplié par 10, la longueur est multipliée environ par 2 ($10^{0.34} = 2.19$). Donc la longueur des esturgeons augmente plus lentement que leur âge. La valeur de la pente (0.34) est un estimé de la taille de l'effet de l'âge sur la longueur.

Il est aussi important d'estimer l'intervalle de confiance sur la pente pour pouvoir estimer si l'intervalle n'inclus ou non que des valeurs biologiquement importantes. Cela peut être fait simplement avec la fonction `confint()`.

```
confint(RegModele.2)
```

```
2.5 %    97.5 %
(Intercept) 1.1377151 1.246270
log10(age)  0.2976433 0.384068
```

L'intervalle de confiance à 95% de la pente est 0.29-0.38. L'intervalle de confiance est assez étroit et éloigné de zéro.

9.9.1. Puissance de détecter une pente donnée

Pour les calculs de puissance avec G*Power vous devrez cependant utiliser une autre métrique de la taille de l'effet, calculée à partir de la pente, de son erreur-type, et de la taille de l'échantillon (ce qui facilite les calculs pour G*Power, mais malheureusement pas pour vous) La métrique (d) est calculée comme:

$$d = \frac{b}{s_b \sqrt{n - k - 1}}$$

où b est l'estimé de la pente, s_b est l'erreur type de la pente, n est le nombre d'observations, et k est le nombre de variables indépendantes (1 pour la régression linéaire simple).

Vous pouvez calculer approximativement la puissance avec G*Power pour une valeur de pente que vous jugez assez grande pour mériter d'être détectée. Choisissez **Tests: Means: One group: difference from constant**, là, vous devrez remplacer la valeur de b dans l'équation pour la taille d'effet (d) par la pente que vous voudriez détecter, mais utiliser l'erreur type calculée à partir de vos données.

Par exemple, supposons que les ichthyologues considèrent qu'une pente de 0.1 pour la relation entre $\log_{10}(\text{fklngh})$ et $\log_{10}(\text{age})$ est significative biologiquement, et qu'ils désirent estimer la puissance de détecter une telle pente à partir d'un échantillon de 20 esturgeons. Les résultats de la régression log-log nous fournissent ce dont on a besoin:

```
summary(RegModele.2)
```

Call:

```
lm(formula = log10(fklngh) ~ log10(age), data = esturgeon.male)
```

Residuals:

| Min | 1Q | Median | 3Q | Max |
|-----------|-----------|-----------|----------|----------|
| -0.082794 | -0.016837 | -0.000719 | 0.021102 | 0.087446 |

Coefficients:

```
Estimate Std. Error t value Pr(>|t|)  
(Intercept) 1.19199 0.02723 43.77 <2e-16 ***  
log10(age) 0.34086 0.02168 15.72 <2e-16 ***  
---  
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Residual standard error: 0.03015 on 73 degrees of freedom

(5 observations deleted due to missingness)

Multiple R-squared: 0.772, Adjusted R-squared: 0.7688

F-statistic: 247.1 on 1 and 73 DF, p-value: < 2.2e-16

L'erreur-type de la pente est 0.02168. Il y avait 75 poissons (n=75) dans l'échantillon de départ. On peut donc calculer la métrique de taille d'effet pour G*Power

$$d = \frac{b}{s_b \sqrt{n - k - 1}} = \frac{0.1}{0.02168 \sqrt{74 - 1 - 1}} = 0.54$$

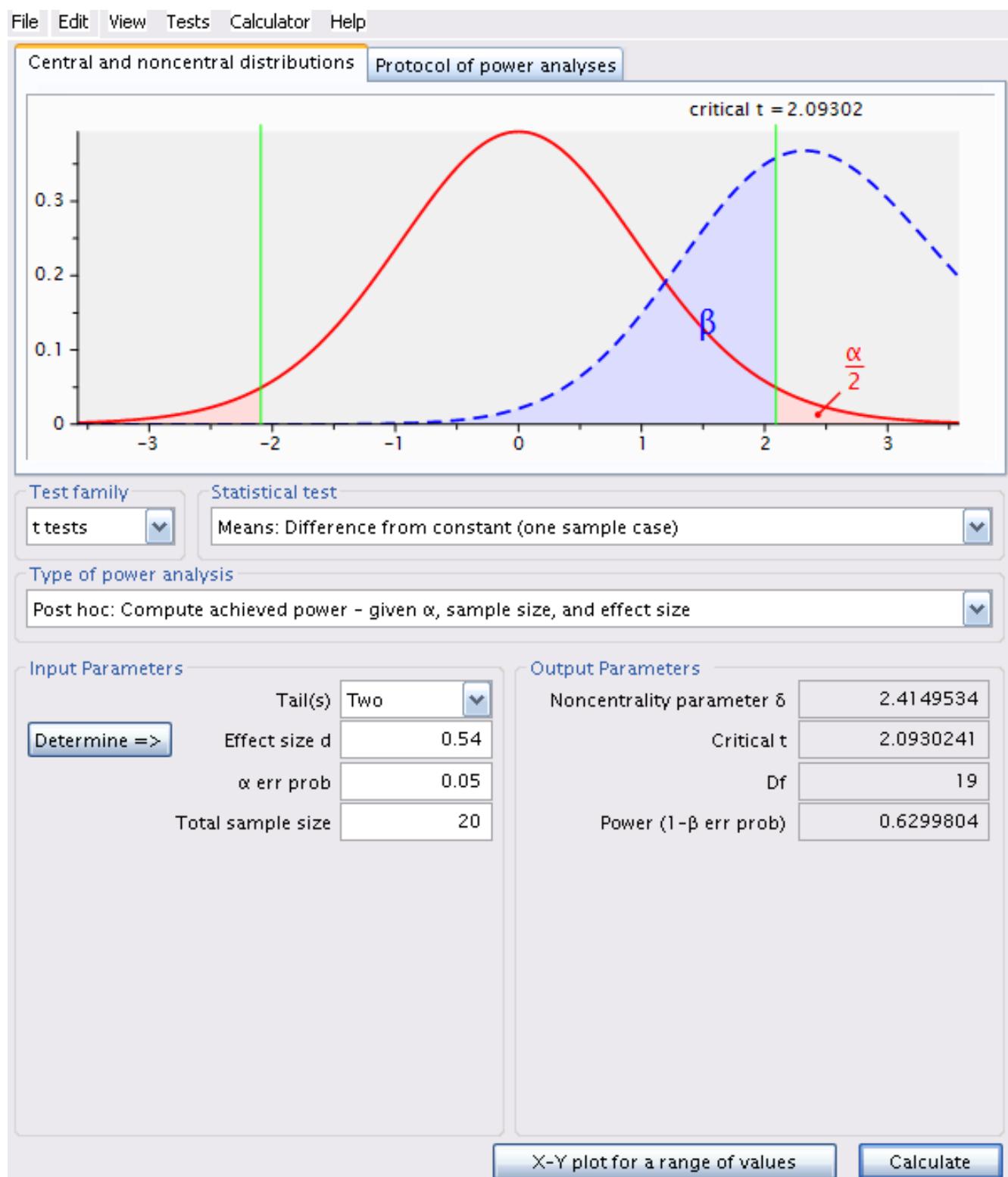
Armés de cette taille d'effet (une pente présumée de 0.1 et une variabilité autour de la régression similaire à la régression de fklngth vs age), choisissez **Tests: Means: One group: difference from constant**, et entrez la valeur calculée de d, alpha, et l'effectif de l'échantillon pour calculer la puissance.

Dans R, il est possible de faire cette analyse avec le code suivant:

```
library(pwr)  
  
# analyse de puissance  
pwr.t.test(n = 20, d = 0.54, sig.level = 0.05, type = "one.sample")
```

One-sample t test power calculation

```
n = 20  
d = 0.54  
sig.level = 0.05
```

Figure 9.11.: Analyse de puissance pour $N = 20$ et pente = 0.1

```
power = 0.6299804
alternative = two.sided
```

La puissance de détecter une pente comme étant statistiquement significative (au niveau alpha), si la pente est 0.1, que la variabilité résiduelle autour de la régression est semblable à celle de notre échantillon (ce qui revient à une taille d'effet de 0.54, pour un échantillon de 20 esturgeons et alpha=0.05) est de 0.629. Seulement environ 2/3 des échantillons de cette taille détecteraient un effet significatif de l'âge sur fklngh.

9.9.2. Effectif requis pour atteindre une puissance désirée (test A-priori)

Pour estimer la taille d'échantillon (effectif) requis pour avoir une puissance de 99% de détecter un effet de l'âge si la pente est 0.1 (sur une échelle log-log), avec alpha=0.05, on utilise la même valeur de d (0.54):

Dans R, il est possible de faire cette analyse avec le code suivant:

```
library(pwr)

# analyse de puissance
pwr.t.test(n = 65, d = 0.54, sig.level = 0.05, type = "one.sample")
```

One-sample t test power calculation

```
n = 65
d = 0.54
sig.level = 0.05
power = 0.9900297
alternative = two.sided
```

En augmentant la taille de l'échantillon à 65, selon le même scénario que précédemment, la puissance augmente à 99%.

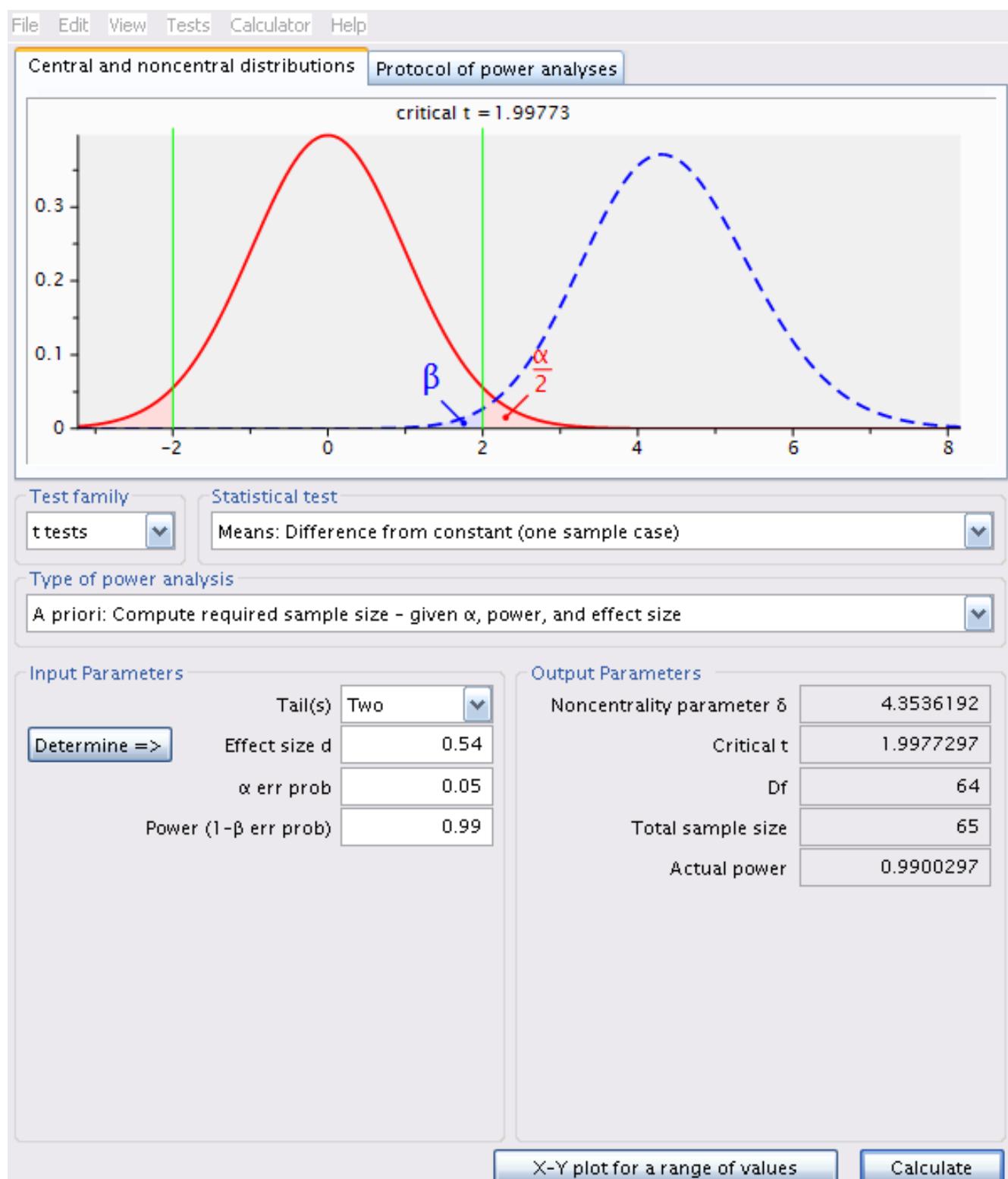


Figure 9.12.: Analyse à priori pour déterminer la taille d'échantillon pour une puissance de 0.99

9.10. Bootstrap en régression simple avec R

Un test non paramétrique pour l'ordonnée à l'origine et la pente d'une régression simple peut être effectué par bootstrap.

```
# charger le paquet boot
library(boot)

# obtenir les poids de régression
bs <- function(formula, data, indices) {
  d <- data[indices, ] # Permet à boot de sélectionner les échantillons
  fit <- lm(formula, data = d)
  return(coef(fit))
}

# bootstrap avec 1000 réplications
results <- boot(
  data = esturgeon.male,
  statistic = bs,
  R = 1000, formula = log10(fklnth) ~ log10(age)
)
# Résultats
results
```

ORDINARY NONPARAMETRIC BOOTSTRAP

Call:

```
boot(data = esturgeon.male, statistic = bs, R = 1000, formula = log10(fklnth) ~
log10(age))
```

Bootstrap Statistics :

| | original | bias | std. error |
|-----|-----------|-------------|------------|
| t1* | 1.1919926 | 0.001539298 | 0.03291399 |

```
t2* 0.3408557 -0.001157070 0.02615716
```

Pour chaque paramètre du modèle (ici l'ordonnée à l'origine est appelée $t1^*$ et la pente de la régression $t2^*$), R imprime :

1. `original` la valeur estimée sur tout l'échantillon
2. `bias` la différence entre la valeur moyenne des estimés par bootstrap et la valeur originale sur tout l'échantillon
3. `std. error` l'erreur-type de l'estimé bootstrap

```
par(mfrow = c(2, 2))
plot(results, index = 1) # intercept
plot(results, index = 2) # log10(age)
```

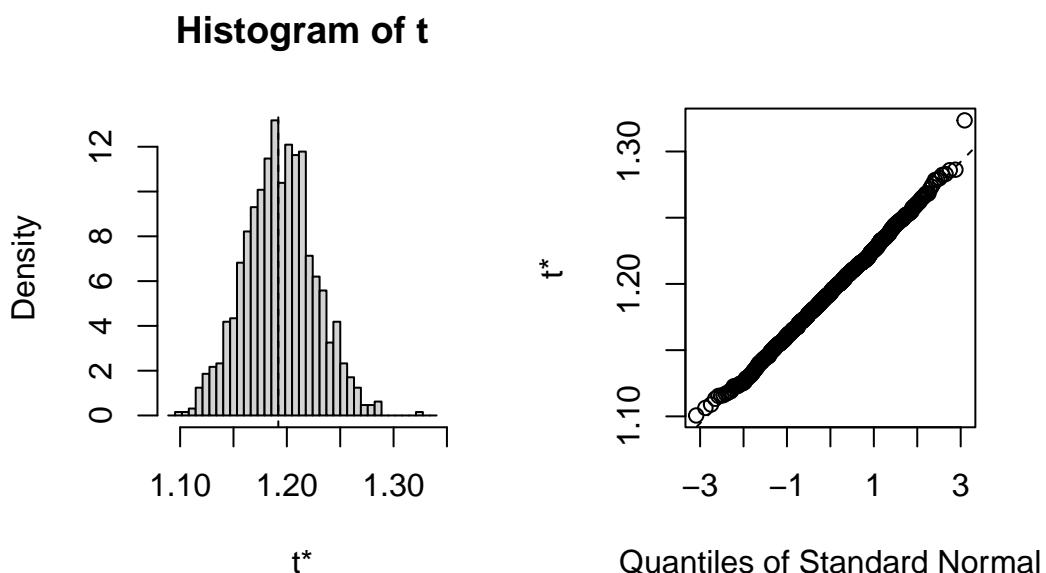


Figure 9.13.: Résultats du test de bootstrap non paramétrique

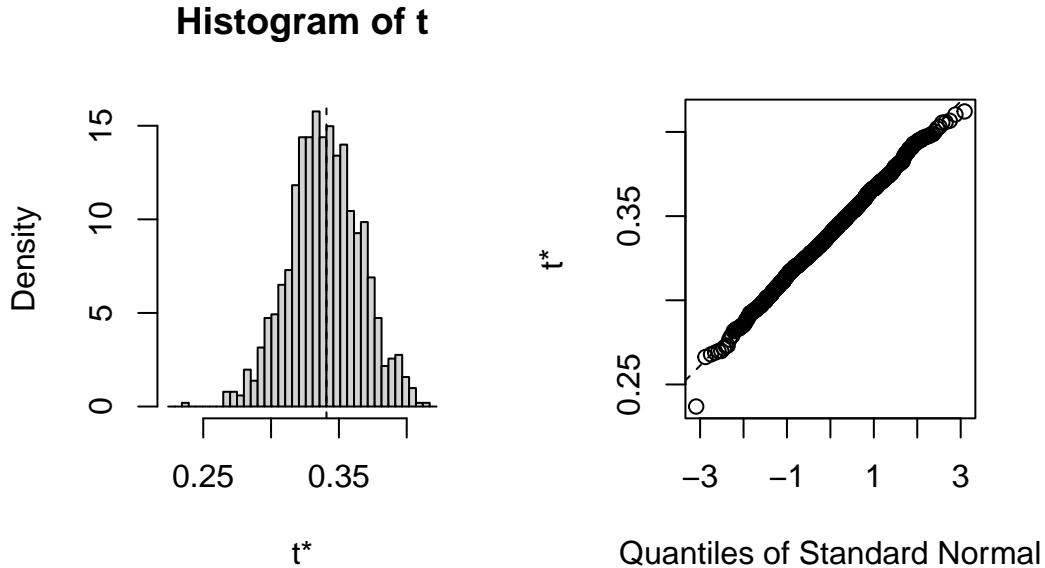


Figure 9.14.: Résultats du test de bootstrap non paramétrique

La distribution des estimés obtenus par bootstrap est assez normale dans cet exemple, avec de petites déviations dans les queue de la distribution (là où ça compte pour les intervalles de confiance...). On pourrait utiliser l'erreur-type des estimés bootstrap pour calculer un intervalle de confiance symétrique (moyenne \pm t E.T.). Cependant, comme R peut facilement calculer des intervalles de confiance qui corrigent pour le biais (BCa: “Bias-Corrected Adjusted”) ou encore des intervalle empiriques à partir des distributions simulées (méthode Percentile) il peut être aussi simple de les calculer selon les 3 méthodes:

```
# interval de confiance pour l'ordonnée à l'origine
boot.ci(results, type = "all", index = 1)
```

```
Warning in boot.ci(results, type = "all", index = 1): bootstrap variances
needed for studentized intervals
```

BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS

Based on 1000 bootstrap replicates

CALL :

```
boot.ci(boot.out = results, type = "all", index = 1)
```

Intervals :

| Level | Normal | Basic |
|-------|------------------|------------------|
| 95% | (1.126, 1.255) | (1.125, 1.256) |

| Level | Percentile | BCa |
|-------|------------------|------------------|
| 95% | (1.128, 1.259) | (1.121, 1.251) |

Calculations and Intervals on Original Scale

```
# intervalle de confiance pour la pente  
boot.ci(results, type = "all", index = 2)
```

Warning in boot.ci(results, type = "all", index = 2): bootstrap variances
needed for studentized intervals

BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS

Based on 1000 bootstrap replicates

CALL :

```
boot.ci(boot.out = results, type = "all", index = 2)
```

Intervals :

| Level | Normal | Basic |
|-------|--------------------|--------------------|
| 95% | (0.2907, 0.3933) | (0.2884, 0.3948) |

| Level | Percentile | BCa |
|-------|--------------------|--------------------|
| 95% | (0.2869, 0.3933) | (0.2951, 0.3978) |

Calculations and Intervals on Original Scale

Ici, les 4 types d'intervalles de confiance que R a calculé sont essentiellement semblables. Si les données avaient violé plus sévèrement les conditions d'application de la régression (normalité, homoscedasticité), alors les différentes méthodes (Normal, Basic, Percentile, et BCa) auraient divergé un peu plus. Lequel choisir alors? BCa est celui qui est préféré de la majorité des praticiens, présentement.

Chapitre 10

Comparaison de deux échantillons

Objectifs de ce chapitre:

- Utiliser R pour examiner visuellement vos données.
- Utiliser R pour comparer les moyennes de deux échantillons tirés de populations à distribution normales.
- Utiliser R pour comparer les moyennes de deux échantillons tirés de populations à distribution non normales.
- Utiliser R pour comparer les moyennes de deux échantillons appariés.

10.1. Paquets R et données requises

Pour ce laboratoire, vous aurez besoin de :

- Paquets R :
 - car 
 - lmtest 
 - boot 
 - lmPerm 
- Jeux de données :
 - pwr 
 - “sturgeon.csv”
 - “skulldat_2020.csv”

```
library(car)
library(lmtest)
library(boot)
library(lmPerm)
library(ggplot2)
library(pwr)
esturgeon <- read.csv("data/sturgeon.csv")
crane <- read.csv("data/skulldat_2020.csv")
```

10.2. Examen visuel des données

Une des premières étapes dans toute analyse de données est l'examen visuel des données par des graphiques et statistiques sommaires pour avoir une idée des distributions sous-jacentes, des valeurs extrêmes et des tendances dans vos données. Cela commence souvent avec des graphiques de vos données (histogrammes, diagrammes de probabilité, boîte à moustache, etc.) qui vous permettent d'évaluer si vos données sont distribuées normalement (c-à-d, suivent une distribution **normale**), si elles sont corrélées les unes aux autres, ou s'il y a des valeurs suspectes dans le jeu de données.

Supposons que l'on veuille comparer la distribution des taille des esturgeons de The Pas et Cumberland House. La variable `fklngth` dans le jeu de données `sturgeon.csv` représente la longueur (en cm) de chaque poisson (mesurée de l'extrémité de la tête à la base de la fourche de la nageoire caudale). Pour commencer, examinons si cette variable est normalement distribuée. On ne va pas tester pour la normalité à ce stade-ci ; la présomption de normalité dans les analyses paramétriques s'applique aux résidus et non aux données brutes. Cependant, si les données brutes ne sont pas normales, vous avez, en général, une très bonne raison de soupçonner que les résidus ne suivront pas non plus une distribution normale.

Une excellente façon de comparer visuellement une distribution à la distribution normale est de superposer un histogramme des données observées à une courbe normale. Pour ce faire, il faut procéder en deux étapes :

1. Indiquer à R que nous voulons créer un histogramme superposé à une courbe normale
2. Spécifier qu'on veut que les graphiques soient faits pour les deux sites
 - En utilisant les données de `sturgeon.csv`, générez les histogrammes et les approximations des distributions normales ajustées aux données de `fklngth` à The Pas et Cumberland House.

```
# Utilisez le jeu de données "sturgeon" pour créer le graphique appelé "mongraph".
# Et définissez l'axe x, correspondant à "fklnghth".

mongraph <- ggplot(data = esturgeon,
                     aes(x = fklnghth)) +
  xlab("Longueur à la fourche (cm)")

# Ajoutez des éléments au graphique (ggplot) "mongraph".

mongraph <- mongraph +
  geom_density() + # Ajoutez la densité des données, lissée.
  geom_rug() + # Ajoutez un "tapis" (barres en bas du graphe).
  geom_histogram(aes(y = ..density..),
                 color = "black", alpha = 0.3) + # Ajoutez un histogramme noir, semi transparent
  stat_function(fun = dnorm,
                args = list(mean = mean(esturgeon$fklnghth),
                            sd = sd(esturgeon$fklnghth)),
                color = "red") # Ajoutez une courbe normale en rouge, à partir de la moyenne et é

mongraph + facet_grid(. ~ location) # Affichez le graphe, par site.
```

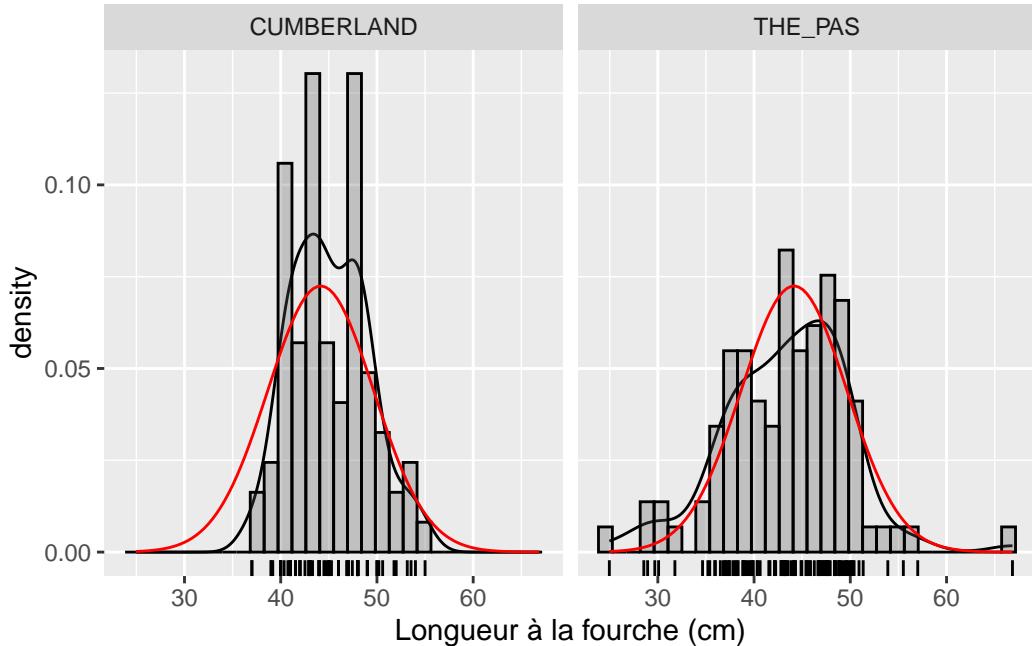


Figure 10.1.: Distribution de la longueur des esturgeons par site.

Examinez ce graphique et essayez de déterminer si ces deux échantillons sont normalement distribués. À mon avis, cette variable est approximativement normalement distribuée dans les deux échantillons.

Puisque ce qui nous intéresse est de comparer la taille des poissons de deux sites différents, c'est probablement une bonne idée de créer un graphique qui compare les deux groupes de données. Une boîte à moustache ("Box plot") convient très bien pour cette tâche.

- Tracez un boxplot de `fklngth` groupé par `location`. Que concluez-vous quant à la différence entre les deux sites?

```
ggplot(data = esturgeon, aes(x = location,
                               y = fklngth)) +
  geom_boxplot(notch = TRUE)
```

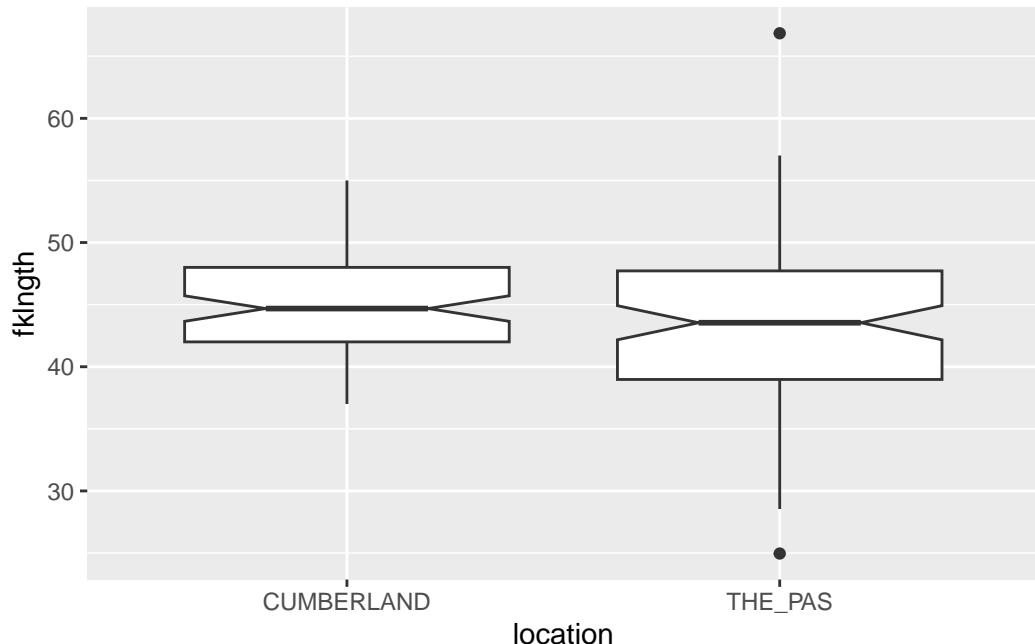


Figure 10.2.: Boxplot de la longueur des esturgeons par site.

Il n'y a pas de grande différence de taille entre les deux sites, mais la taille des poissons à The Pas est plus variable, ayant une plus large étendue de taille et des valeurs extrêmes (définies par les valeurs qui sont $> 1.5 * \text{l'étendue interquartile}$) à chaque bout de la distribution.

10.3. Comparer les moyennes de deux échantillons indépendants

Éprouvez l'hypothèse nulle (H_0) : *La longueur à la fourche n'est pas différente entre The Pas et Cumberland House* de 3 manières différentes :

1. Test paramétriques supposant des variances égales
2. Test paramétriques supposant des variances différentes
3. Test non-paramétrique (pas de conditions d'applications sur la distribution et la variance)

Que concluez-vous?

```
# Test t supposant des variances égales.

t.test(fklngh ~ location,
       data = esturgeon,
       alternative = "two.sided",
       var.equal = TRUE)
```

Two Sample t-test

```
data: fklngh by location
t = 2.1359, df = 184, p-value = 0.03401
alternative hypothesis: true difference in means between group CUMBERLAND and group THE_PAS is not zero
95 percent confidence interval:
0.1308307 3.2982615
sample estimates:
mean in group CUMBERLAND      mean in group THE_PAS
45.08439                      43.36984
```

```
# Test t supposant des variances différentes.

t.test(fklngh ~ location,
       data = esturgeon,
       alternative = "two.sided",
       var.equal = FALSE)
```

```
Welch Two Sample t-test
```

```
data: fklngth by location
t = 2.2201, df = 169.8, p-value = 0.02774
alternative hypothesis: true difference in means between group CUMBERLAND and group THE_PAS is no
95 percent confidence interval:
0.1900117 3.2390804
sample estimates:
mean in group CUMBERLAND      mean in group THE_PAS
45.08439                      43.36984
```

```
# Test non paramétrique.

wilcox.test(fklngth ~ location,
            data = esturgeon,
            alternative = "two.sided")
```

```
Wilcoxon rank sum test with continuity correction
```

```
data: fklngth by location
W = 4973, p-value = 0.06296
alternative hypothesis: true location shift is not equal to 0
```

En se fiant au *test de t*, on rejette donc l'hypothèse nulle. Il y a une différence significative entre les deux moyennes des longueurs à la fourche selon le site.

Notez que si l'on se fie au test de *Wilcoxon*, on ne peut pas rejeter l'hypothèse nulle. Les deux tests mènent donc à des conclusions contradictoires. La différence significative obtenue par le test de t peut provenir en partie d'une violation des conditions d'application du test (normalité et homoscédasticité). D'un autre côté, l'absence de différence significative selon le test de Wilcoxon pourrait être due au fait que, pour un effectif donné, la puissance du test non paramétrique est inférieure à celle du test paramétrique correspondant. Compte tenu 1) des valeurs de p obtenues pour les deux tests, et 2) le fait que pour des grands échantillons (des effectifs de 84 et 101 sont considérés grands) le test de t est considéré robuste, il est raisonnable de rejeter l'hypothèse nulle.

Avant d'accepter les résultats du test de t et de rejeter l'hypothèse nulle qu'il n'y a pas de différences de taille entre les deux sites, il est important de déterminer si les données remplissent les conditions de normalité des résidus et d'égalité des variances. L'examen préliminaire suggérait que les données sont à peu près normales mais qu'il y avait peut-être des problèmes avec les variances (puisque l'étendue des données pour The Pas était beaucoup plus grande que celle pour Cumberland). On peut examiner ces conditions d'application plus en détail en examinant les résidus d'un modèle linéaire et en utilisant les graphiques diagnostiques:

```
m1 <- lm(fklength ~ location, data = esturgeon)
par(mfrow = c(2, 2))
plot(m1)
```

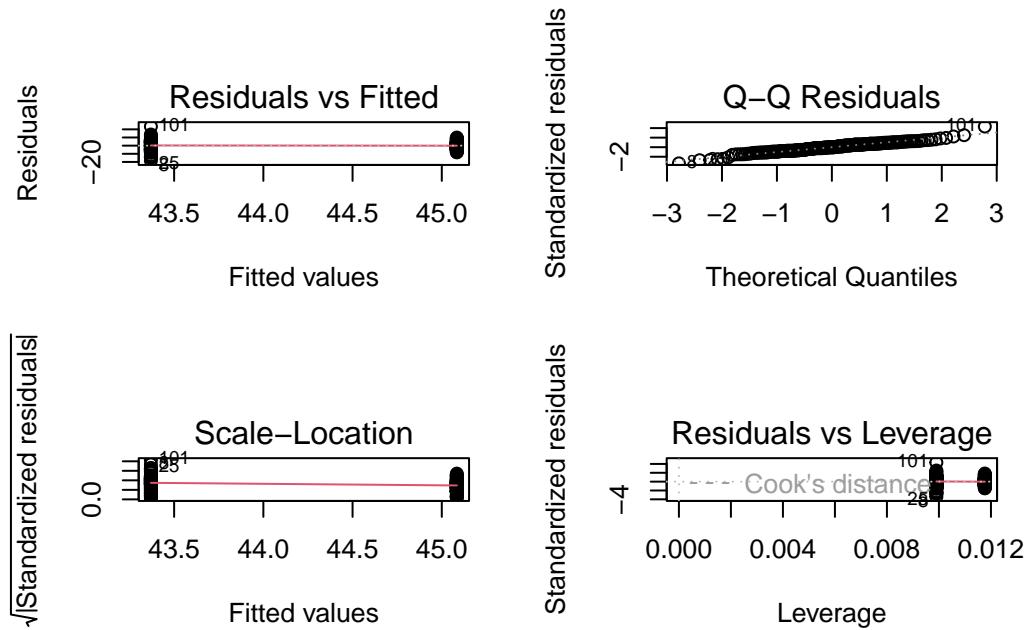


Figure 10.3.: Examen visuel des conditions d'application du modèle linéaire.

Le premier graphique ci-dessus montre comment les résidus se distribuent autour des valeurs prédites (les moyennes) pour chaque site et permet de juger si il semble y avoir un problème de normalité ou d'homoscédasticité. Si les variances étaient égales dans les deux sites, l'étendue verticale des résidus tendrait à être la même. Sur le graphique, on voit que l'étendue des résidus est plus grande à gauche (le site où la taille moyenne est la plus faible), ce qui suggère un possible problème d'homogénéité des variances. On peut tester cela plus formellement en comparant la moyenne de la valeur absolue des résidus.(on y reviendra; c'est le test de Levene).

Le deuxième graphique est un graphique de probabilité (graphique Q–Q) des résidus. Comme ici, les points tombent près de la diagonale, il ne semble pas y avoir de problème important avec la normalité. On peut faire un test formel

de la condition de normalité : le test de Shapiro-Wilk:

```
shapiro.test(residuals(m1))
```

```
Shapiro-Wilk normality test
```

```
data: residuals(m1)
W = 0.97469, p-value = 0.001857
```

Hummm... Ce test indique que les résidus ne sont pas normaux, ce qui contredit notre évaluation visuelle. Cependant, puisque (a) la distribution des résidus ne s'éloigne pas beaucoup de la normalité et (b) le nombre d'observations à chaque site est raisonnablement grand (i.e. >30), nul besoin d'être trop inquiet quant à l'impact de cette violation de normalité sur la fiabilité du test.

Qu'en est-il de l'égalité des variances?

```
library(car)
leveneTest(m1)
```

```
Warning in leveneTest.default(y = y, group = group, ...): group coerced to
factor.
```

| | Df | F value | Pr(>F) |
|-------|-----|----------|-----------|
| group | 1 | 11.51454 | 0.0008456 |
| | 184 | NA | NA |

```
bptest(m1)
```

```
studentized Breusch-Pagan test
```

```
data: m1
BP = 8.8015, df = 1, p-value = 0.00301
```

Les résultats qui précédents proviennent de 2 des tests disponibles en R (dans les package `car` et `lmtest`) qui éprouvent l'hypothèse de l'égalité des variances dans des tests de t ou des modèles linéaires ayant uniquement des variables indépendantes discontinues ou catégoriques. **Il est inutile de faire les 2 tests.** Si ils sont présentés ici, c'est que ces 2 tests sont usuels et qu'il n'y a pas consensus quant au meilleur des deux. Le test de Levene est le plus connu et utilisé. Il compare la moyenne des valeurs absolues des résidus dans les deux groupes. Le test Breusch-Pagan a l'avantage d'être applicable à une plus large gamme de modèles linéaires (il peut être utilisé également avec des variables indépendantes continues, comme en régression). Ici, les deux tests mènent à la même conclusion: la variance diffère entre les deux sites.

Sur la base de ces résultats, on peut conclure qu'il y a des éléments (même si faibles) pour rejeter l'hypothèse nulle qu'il n'y a pas de différence dans la taille de poissons entre les deux sites. On a utilisé une modification du test de t pour tenir compte du fait que les variances ne sont pas égales et nous sommes satisfaits que la condition de normalité des résidus a été remplie. Alors, “fklngh” à Cumberland est plus grande que “fklngh” à The Pas.

10.4. Bootstrap et tests de permutation pour comparer deux moyennes

10.4.1. Bootstrap

Le bootstrap et les tests de permutation peuvent être utilisés pour comparer les moyennes (ou d'autres statistiques). Le principe général est simple et peut être effectué de diverses façons. Ici on utilise certains des outils disponibles et le fait qu'une comparaison de moyenne peut être représentée par un modèle linéaire. On pourra utiliser un programme similaire plus tard quand on ajustera des modèles plus complexes (mais plus amusants !).

```
library(boot)
```

La première section sert à définir une fonction (ici appelée `bs`) qui extraie les coefficients d'un modèle ajusté :

```
# Fonction pour extraire les coefficients d'un modèle pour chaque itérations.
bs <- function(formule, data, indices) {
  d <- data[indices, ]
  fit <- lm(formule, data = d)
  return(coef(fit))
}
```

La deuxième section avec la commande `boot()` fait le gros du travail: on prend les données dans “sturgeon”, on les bootstrap $R = 1000$ fois, et chaque fois on ajuste le modèle `fklngth` vs `location` et on garde les valeurs calculées par la fonction `bs`.

```
# bootstrap avec 1000 réplications.

resultats <- boot(data = esturgeon, statistic = bs, R = 1000,
                    formule = fklngth ~ location)

# Affichez les résultats.

resultats
```

ORDINARY NONPARAMETRIC BOOTSTRAP

Call:

```
boot(data = esturgeon, statistic = bs, R = 1000, formule = fklngth ~
      location)
```

Bootstrap Statistics :

| | original | bias | std. error |
|-----|-----------|-------------|------------|
| t1* | 45.084391 | 0.001731858 | 0.4268881 |
| t2* | -1.714546 | 0.032036414 | 0.7713654 |

On obtient les estimés originaux pour les deux coefficients du modèle: la moyenne pour le premier (alphabétiquement) site soit Cumberland, et la différence entre les deux moyennes à Cumberland et The Pas. C'est ce second paramètre, la différence entre les moyennes, qui nous intéresse.

```
plot(resultats, index = 2)
```

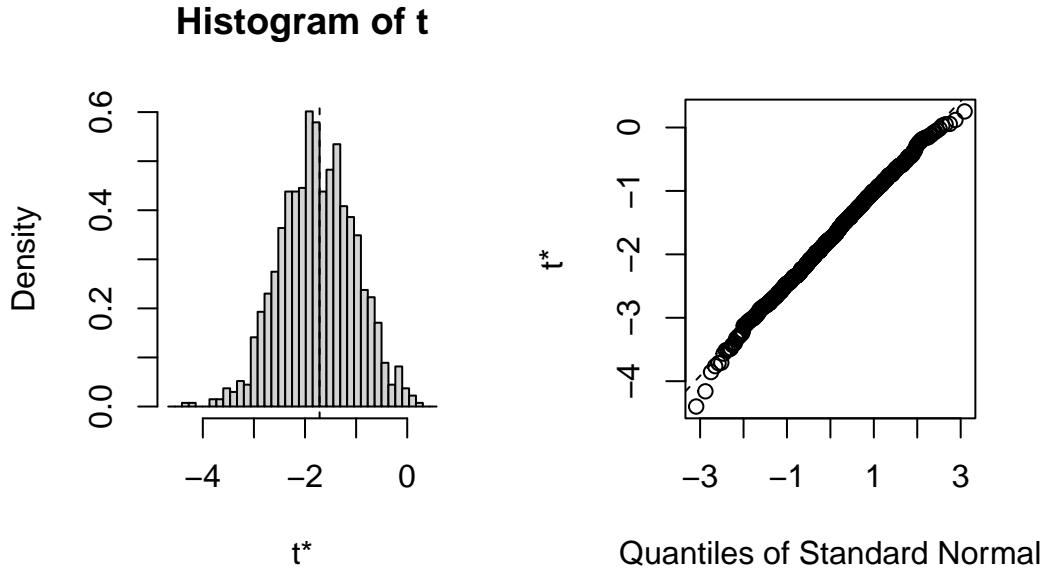


Figure 10.4.: Normalité des estimés de la différence des moyennes par bootstrap.

```
# Calculez l'intervalle de confiance à 95%.
boot.ci(resultats, type = "bca", index = 2)
```

BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS

Based on 1000 bootstrap replicates

CALL :

```
boot.ci(boot.out = resultats, type = "bca", index = 2)
```

Intervals :

Level BCa

95% (-3.278, -0.188)

Calculations and Intervals on Original Scale

Comme l'intervalle de confiance n'inclue pas 0, on conclue que les moyennes ne sont pas les mêmes.

10.4.2. Permutation

Les tests de permutation pour les modèles linéaires peuvent être effectués à l'aide du package `lmPerm` :

```
m1Perm <- lmp(fklnth ~ location,
               data = sturgeon,
               perm = "Prob")
```

```
[1] "Settings: unique SS "
```

La fonction `lmp()` fait tout le travail pour nous. Ici, cette fonction est effectuée avec l'option `perm` pour choisir la règle utilisée pour stopper les calculs. L'option “Prob” arrête les permutations quand la déviation standard estimée pour la p-valeur tombe sous un seuil déterminé. C'est l'une des nombreuses règles qui peuvent possiblement être utilisées pour ne faire les permutations que sur un sous-ensemble des permutations possibles (ce qui prendrait souvent très longtemps).

```
summary(m1Perm)
```

Call:

```
lmp(formula = fklnth ~ location, data = sturgeon, perm = "Prob")
```

Residuals:

| Min | 1Q | Median | 3Q | Max |
|-----------|----------|----------|---------|----------|
| -18.40921 | -3.75370 | -0.08439 | 3.76598 | 23.48055 |

Coefficients:

| | Estimate | Iter | Pr(Prob) | | | |
|----------------|----------|----------|----------|--------|---------|---|
| location1 | 0.8573 | 2702 | 0.0359 * | | | |
| --- | | | | | | |
| Signif. codes: | 0 *** | 0.001 ** | 0.01 * | 0.05 . | 0.1 ' ' | 1 |

Residual standard error: 5.454 on 184 degrees of freedom

Multiple R-Squared: 0.02419, Adjusted R-squared: 0.01889

F-statistic: 4.562 on 1 and 184 DF, p-value: 0.03401

1. Iter: la règle a limité le calcul à 3925 permutations. Notez que ce nombre va varier à chaque fois que vous ferez tourner ce code. Ce sont des résultats obtenus par permutations **aléatoires**, donc vous devez vous attendre à de la variabilité. .

2. **Pr(Prob):** La p-valeur estimée pour H_0 est 0.025. La différence observée pour “fklngh” entre les deux sites était plus grande que les valeurs permutées pour environ ($1 - 0.025 = 97.5\%$) des 3925 permutations. Notez que 3925 permutations ce n'est pas un si grand nombre de permutations que ça, et donc les faibles valeurs de p ne sont pas très précises. Si vous voulez des valeurs précises de p, vous devrez faire plus de permutations.
3. **F-statistic:** Le reste est la sortie standard pour un modèle ajusté à des données, avec le test paramétrique. Ici, la p-valeur, présumant que toutes les conditions d'application sont remplies, est 0.034.

10.5. Comparer les moyennes de deux échantillons appariés

⚠ Avertissement

Pour la section suivante veuillez télécharger le jeu de données `skulldat_2020.csv` qui a été récemment ajouté sur Brightspace.

Dans certaines expériences les mêmes individus sont mesurés deux fois, par exemple avant et après un traitement ou encore à deux moments au cours de leur développement. Les mesures obtenues lors de ces deux événements ne sont donc **pas indépendantes**, et des comparaisons de ces mesures appariées doivent être faites.

Le jeu de données `skulldat_2020.csv` contient des mesures de la partie inférieure du visage de jeunes filles d'Amérique du Nord prises à 5 ans, puis à 6 ans (données de Newman and Meredith, 1956).

- Pour débuter, éprouvons l'hypothèse que la largeur de la figure est la même à 5 ans et à 6 ans en assumant que les mesures viennent d'échantillons indépendants.

```
crane <- read.csv("data/skulldat_2020.csv")
t.test(width ~ age,
       data = crane,
       alternative = "two.sided")
```

Welch Two Sample t-test

```
data: width by age
t = -1.7812, df = 27.93, p-value = 0.08576
```

alternative hypothesis: true difference in means between group 5 and group 6 is not equal to 0
95 percent confidence interval:

-0.43002624 0.03002624

sample estimates:

mean in group 5 mean in group 6

7.461333 7.661333

Jusqu'à maintenant, nous avons spécifié le test de t en utilisant une notation de type `formule` avec $y \sim x$ où y est la variable pour laquelle on souhaite comparer les moyennes et x correspond à une variable définissant les groupes. Cela marche bien lorsque les données ne sont pas appariées et sont présentées dans un format de type *long* où les données prise dans une même catégorie ou sur une même personne sont simplement les unes en-dessous des autres avec des colonnes indiquant l'appartenance des mesures aux différentes catégories (voir la structure de `crane` par exemple) Dans le jeu de données `crane`, il y a 3 colonnes:

- `width`: largeur de la tête
- `age`: age lors de la mesure
- `id`: identité de la personne

```
head(crane)
```

| width | age | id |
|-------|-----|----|
| 7.33 | 5 | 1 |
| 7.53 | 6 | 1 |
| 7.49 | 5 | 2 |
| 7.70 | 6 | 2 |
| 7.27 | 5 | 3 |
| 7.46 | 6 | 3 |

Quand les données sont appariées, il faut indiquer comment elles doivent être associées. Dans notre exemple, elles sont appariées par individu. Le format de données de type *long* indique cet appariement via la colonne `id`. Cependant, la fonction `t.test` ne permet pas de le prendre en compte. Il faut donc transformer les données en format de type *large* ou *horizontale* où il y a une colonne différente pour chaque catégorie. Dans notre exemple, on veut un jeu de données avec une colonne de mesure par `age` et où chaque ligne correspond à une personne différente. On peut modifier le format des données avec le code suivant.

```
crane_h <- data.frame(id = unique(crane$id))
crane_h$width5 <- crane$width[match(crane_h$id, crane$id) & crane$age == 5]
crane_h$width6 <- crane$width[match(crane_h$id, crane$id) & crane$age == 6]
head(crane_h)
```

| id | width5 | width6 |
|----|--------|--------|
| 1 | 7.33 | 7.53 |
| 2 | 7.49 | 7.70 |
| 3 | 7.27 | 7.46 |
| 4 | 7.93 | 8.21 |
| 5 | 7.56 | 7.81 |
| 6 | 7.81 | 8.01 |

Maintenant, effectuons le test apparié qui est approprié: Que conclure? Comment les résultats diffèrent-ils de la première analyse? Pourquoi?

```
t.test(crane_h$width5, crane_h$width6,
       alternative = "two.sided",
       paired = TRUE
      )
```

Paired t-test

```
data: crane_h$width5 and crane_h$width6
t = -19.72, df = 14, p-value = 1.301e-11
alternative hypothesis: true mean difference is not equal to 0
95 percent confidence interval:
-0.2217521 -0.1782479
sample estimates:
mean difference
-0.2
```

La première analyse a comme supposition que les deux échantillons de filles de 5 et 6 ans sont indépendants, alors que la deuxième analyse a comme supposition que la même fille a été mesurée deux fois, une fois à 5 ans, et la deuxième fois à 6 ans.

Notez que, dans le premier cas, on accepte l'hypothèse nulle, mais que le test apparié rejette l'hypothèse nulle. Donc, le test qui est approprié (le test apparié) indique un effet très significatif de l'âge, mais le test inapproprié suggère que l'âge n'importe pas. C'est parce qu'il y a une très forte corrélation entre la largeur du visage à 5 et 6 ans:

```
graphcrane <- ggplot(data = crane_h, aes(x = width5, y = width6)) +
  geom_point() +
  labs(x = "Largeur du visage à 5 ans", y = "Largeur du visage à 6 ans") +
  geom_smooth() +
  scale_fill_continuous(low = "lavenderblush", high = "red")

graphcrane
```

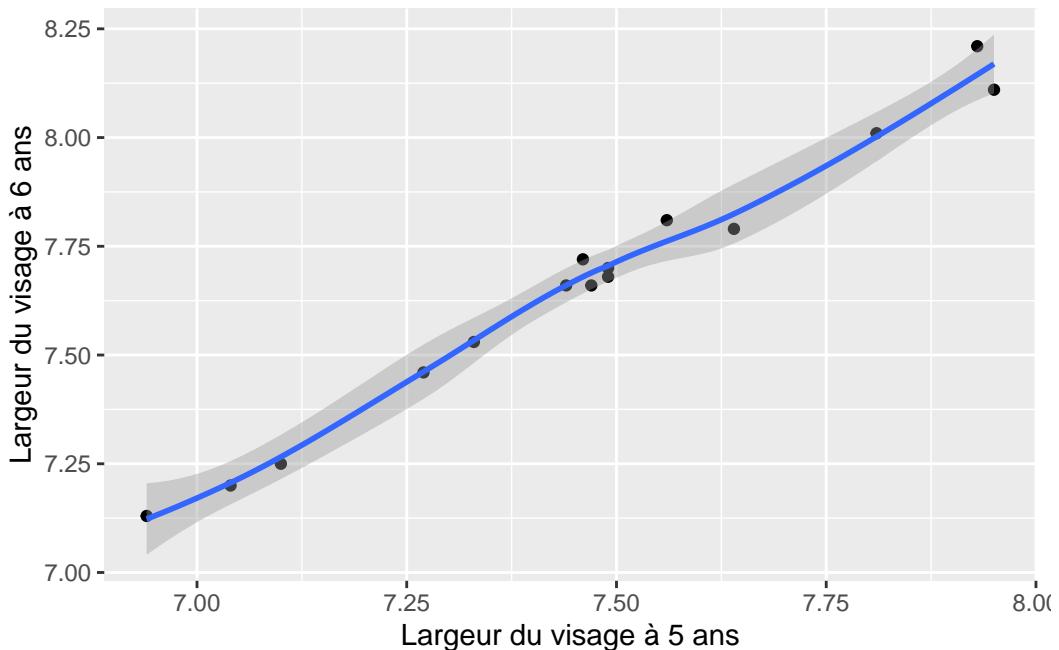


Figure 10.5.: Relation entre la largeur du visage à 5 et 6 ans.

Avec $r = 0.9930841$. En présence d'une si forte corrélation, l'erreur-type de la différence appariée de largeur du visage entre 5 et 6 ans est beaucoup plus petite que l'erreur-type de la différence entre la largeur moyenne à 5 ans et la largeur moyenne à 6 ans. Par conséquent, la statistique t associée est beaucoup plus élevée pour le test apparié, la puissance du test est plus grande, et la valeur de p plus petite.

- Répétez l'analyse en utilisant l'alternative non paramétrique, le test Wilcoxon rang-signés (signed-rank). (Que concluez-vous?)

```
wilcox.test(crane_h$width5, crane_h$width6,
            alternative = "two.sided",
            paired = TRUE
)
```

Warning in wilcox.test.default(crane_h\$width5, crane_h\$width6, alternative =
 "two.sided", : cannot compute exact p-value with ties

Wilcoxon signed rank test with continuity correction

```
data: crane_h$width5 and crane_h$width6
V = 0, p-value = 0.0007193
alternative hypothesis: true location shift is not equal to 0
```

Donc on tire la même conclusion qu'avec le test de t apparié et on conclue qu'il y a des différences significatives entre la taille des crânes de filles âgées de 5 et 6 ans (quelle surprise!).

Mais, attendez une minute ! On a utilisé des tests bilatéraux ici mais, compte tenu des connaissances sur la croissance des enfants, une hypothèse unilatérale serait préférable. Ceci peut être accommodé en modifiant l'option “alternative”. On utilise l'hypothèse alternative pour décider entre “less” ou “greater”. Ici, si il y a une différence, on s'attend à ce que width5 sera inférieur à width6, donc on utiliserait “less”.

```
t.test(crane_h$width5, crane_h$width6,
        alternative = "less",
        paired = TRUE
)
```

Paired t-test

```
data: crane_h$width5 and crane_h$width6
```

```
t = -19.72, df = 14, p-value = 6.507e-12
alternative hypothesis: true mean difference is less than 0
95 percent confidence interval:
-Inf -0.1821371
sample estimates:
mean difference
-0.2
```

```
wilcox.test(crane_h$width5, crane_h$width6,
            alternative = "less",
            paired = TRUE
)
```

```
Warning in wilcox.test.default(crane_h$width5, crane_h$width6, alternative =
"less", : cannot compute exact p-value with ties
```

Wilcoxon signed rank test with continuity correction

```
data: crane_h$width5 and crane_h$width6
V = 0, p-value = 0.0003597
alternative hypothesis: true location shift is less than 0
```

Pour estimer la puissance d'un test de t avec R, il faut utiliser la fonction `pwr.t.test()`. Il faut spécifier l'argument `type = "paired"`, utiliser la moyenne et l'écart-type de la différence pour calculer le `d` de cohen avec la formule `mean(diff) / sd(diff)` pour un test de t pairé.

```
crane_h$diff <- crane_h$width6 - crane_h$width5
pwr.t.test(n = 15,
            d = mean(crane_h$diff) / sd(crane_h$diff),
            type = "paired")
```

Paired t test power calculation

```
n = 15  
d = 5.091751  
sig.level = 0.05  
power = 1  
alternative = two.sided
```

NOTE: n is number of *pairs*

10.6. Références

Bumpus, H.C. (1898) The elimination of the unfit as illustrated by the introduced sparrow, *Passer domesticus*. Biological Lectures, Woods Hole Biology Laboratory, Woods Hole, 11 th Lecture: 209 - 226.

Newman, K.J. and H.V. Meredith. (1956) Individual growth in skeletal bigonial diameter during the childhood period from 5 to 11 years of age. Amer. J. Anat. 99: 157 - 187.

Chapitre 11

ANOVA à un critère de classification

Objectifs de ce chapitre :

- Utiliser R pour effectuer une analyse de variance (ANOVA) paramétrique à un critère de classification, suivie de comparaisons multiples.
- Utiliser R pour vérifier si les conditions d'application de l'ANOVA paramétrique sont remplies.
- Utiliser R pour faire une ANOVA à un critère de classification non-paramétrique.
- Utiliser R pour transformer des données de manière à mieux remplir les conditions d'application de l'ANOVA paramétrique.

11.1. Paquets et données requises pour le labo

Pour ce laboratoire, vous aurez besoin de :

- Paquets R :
 - `ggplot2` 
 - `multcomp` 
 - `car` 
- Jeu de données :
 - “Dam10dat.csv”

```
library(ggplot2)
library(car)
library(multcomp)
```

11.2. ANOVA à un critère de classification et comparaisons multiples

L'ANOVA à un critère de classification est l'analogie du test de t pour des comparaisons de moyennes de plus de deux échantillons. Les conditions d'application du test sont essentiellement les mêmes, et lorsque appliqué à deux échantillons ce test est mathématiquement équivalent au test de t.

En 1961-1962, le barrage Grand Rapids était construit sur la rivière Saskatchewan en amont de Cumberland House. Certains rapports indiqueraient que durant la construction, plusieurs gros esturgeons restèrent prisonniers dans des sections peu profondes et moururent. Des inventaires de la population d'esturgeons furent faits en 1954, 1958, 1965 et 1966. Au cours de ces inventaires, la longueur à la fourche (frklngh) a été mesurée (pas nécessairement sur chaque poisson cependant). Ces données sont contenues dans le fichier Dam10dat.csv.

11.2.1. Visualiser les données

- À partir des données (Dam10dat), il faut d'abord changer le type de donnée de la variable year, pour que R la traite comme un facteur (`as.factor()`) plutôt que comme une variable continue.

```
Dam10dat <- read.csv("data/Dam10dat.csv")
Dam10dat$year <- as.factor(Dam10dat$year)
str(Dam10dat)
```

```
'data.frame': 118 obs. of 21 variables:
 $ year      : Factor w/ 4 levels "1954","1958",...: 1 1 1 1 1 1 1 1 1 ...
 $ fklngh    : num  45 50 39 46 54.5 49 42.5 49 56 54 ...
 $ totlngth: num  49 NA 43 50.5 NA 51.7 45.5 52 60.2 58.5 ...
 $ drlngth   : logi  NA NA NA NA NA ...
 $ drwgght   : num  16 20.5 10 17.5 19.7 21.3 9.5 23.7 31 27.3 ...
 $ rdwgght   : num  24.5 33 15.5 28.5 32.5 35.5 15.3 40.5 51.5 43 ...
 $ sex       : int  1 1 1 2 1 2 1 1 1 ...
 $ age       : int  24 33 17 31 37 44 23 34 33 47 ...
 $ lfkl      : num  1.65 1.7 1.59 1.66 1.74 ...
 $ ltotl     : num  1.69 NA 1.63 1.7 NA ...
 $ ldrl      : logi  NA NA NA NA NA ...
 $ ldrwgght : num  1.2 1.31 1 1.24 1.29 ...
 $ lrdwgght : num  1.39 1.52 1.19 1.45 1.51 ...
```

```
$ lage      : num  1.38 1.52 1.23 1.49 1.57 ...
$ rage      : int  4 6 3 6 7 7 4 6 6 7 ...
$ ryear     : int  1954 1954 1954 1954 1954 1954 1954 1954 1954 1954 ...
$ ryear2    : int  1958 1958 1958 1958 1958 1958 1958 1958 1958 1958 ...
$ ryear3    : int  1966 1966 1966 1966 1966 1966 1966 1966 1966 1966 ...
$ location: int  1 1 1 1 1 1 1 1 1 1 ...
$ girth     : logi  NA NA NA NA NA ...
$ lgirth    : logi  NA NA NA NA NA ...
```

- Ensuite, visualisez les données (de la même manière que dans le chapitre précédent sur les tests de t). Créez un histogramme avec une courbe de densité et un “Box plot” par année. Que vous révèlent ces données ?

```
mongraph <- ggplot(Dam10dat, aes(x = fklnngth)) +
  labs(x = "Longueur à la fourche (cm)",
       y = "Densité") +
  geom_density() +
  geom_rug() +
  geom_histogram(aes(y = after_stat(density)),
                 color = "black",
                 alpha = 0.3
  ) +
  stat_function(
    fun = dnorm,
    args = list(
      mean = mean(Dam10dat$fklnngth),
      sd = sd(Dam10dat$fklnngth)
    ),
    color = "red"
  )

# Affichez le graphe, par année
mongraph + facet_wrap(~year, ncol = 2)
```

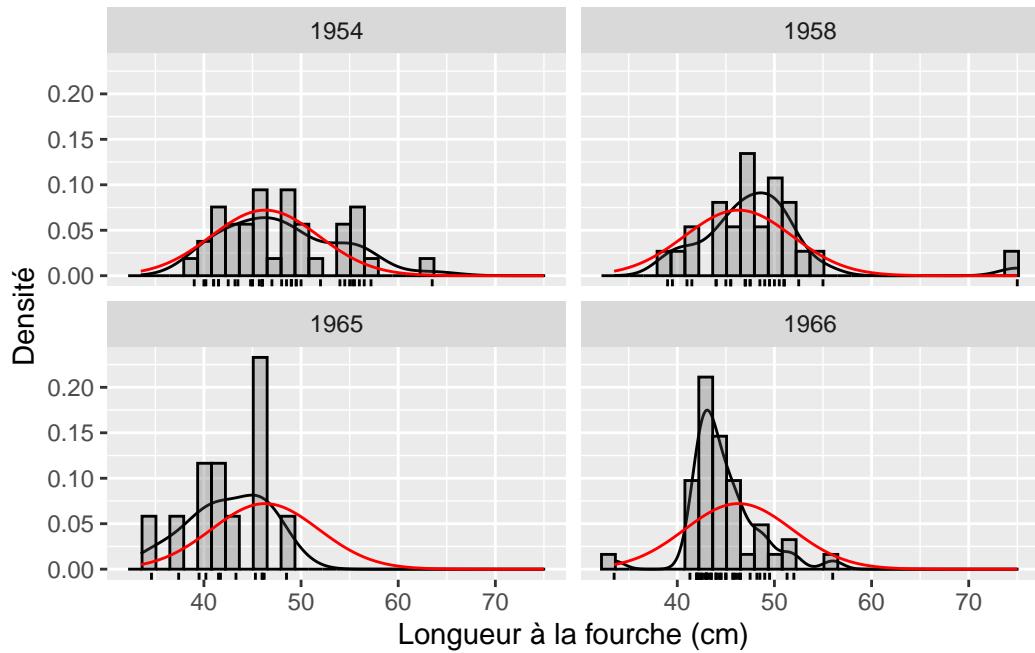


Figure 11.1.: Distribution de la longueur des esturgeons par année.

```
boxplot(fklnghth ~ year, data = Dam10dat,
        xlab = "Années", ylab = "Longueur à la fourche (cm)")
```

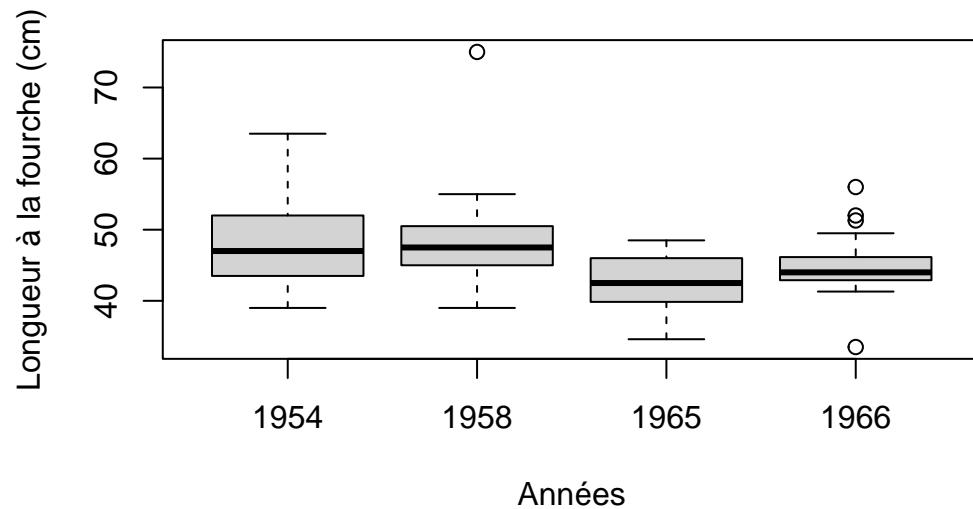


Figure 11.2.: Boxplot de la longueur à la fourche pas année.

Il semble que la taille des esturgeons ait légèrement diminué après la construction du barrage, mais les données sont très variables et les effets ne sont pas parfaitement clairs. Il y a peut-être des problèmes de normalité avec les échantillons de 1954 et 1966, et il y a probablement des valeurs extrêmes dans les échantillons de 1958 et 1966. Testons les conditions d'application de l'ANOVA. Il faut d'abord faire l'analyse et examiner les résidus.

11.2.2. Vérifier les conditions d'application de l'ANOVA paramétrique

L'ANOVA paramétrique a trois principales conditions d'application :

1. Les résidus sont **normalement** distribués,
2. La variance des résidus est égale dans tous les traitements (**homoscédasticité**) et
3. Les résidus sont **indépendants** les uns des autres.

Ces conditions doivent être remplies avant que l'on puisse se fier aux résultats de l'ANOVA paramétrique.

- Faites une ANOVA à un critère de classification sur la longueur (fklngh) par année (year) et produisez les graphiques diagnostiques

```
# Ajustez le modèle anova et affichez les graphiques diagnostiques des résidus
anova.model1 <- lm(fklngh ~ year, data = Dam10dat)
par(mfrow = c(2, 2))
plot(anova.model1)
```

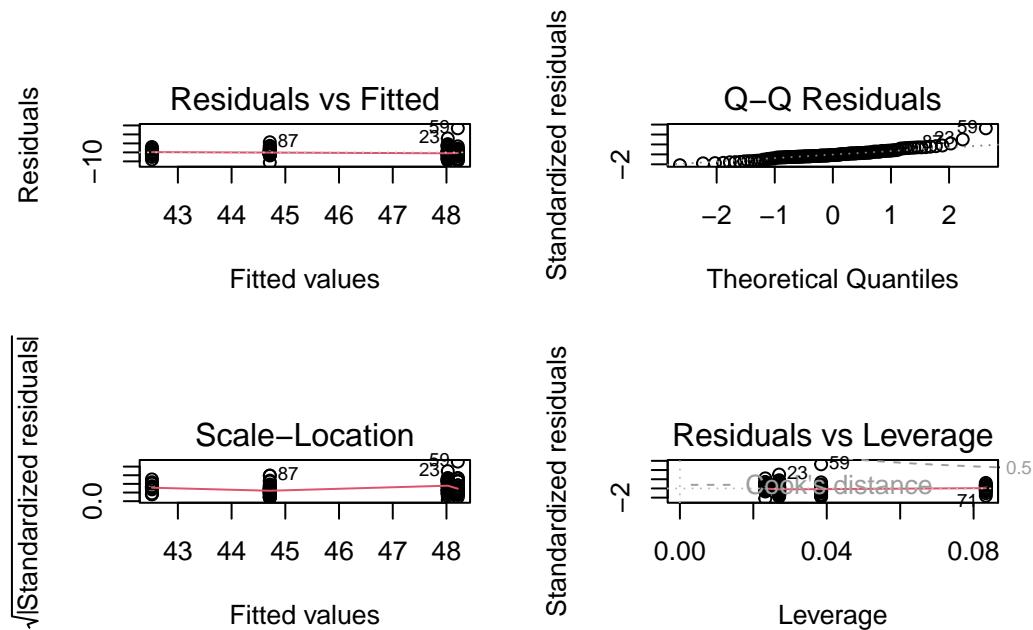


Figure 11.3.: Conditions d'applications de l'ANOVA.

Avertissement

Vérifiez que la variable indépendant est bien un **facteur**. Si la variable indépendante est reconnue comme du texte (**character**) alors vous n'obtiendrez que 3 graphiques et un message d'erreur du type:
'hat values (leverages) are all = 0.1'

and there are no factor predictors; no plot no. 5'

D'après les graphiques, on peut douter de la normalité et de l'homogénéité des variances. Notez qu'il y a un point qui ressort vraiment avec une forte valeur résiduelle (cas numéro 59) et qui ne s'aligne pas bien avec les autres valeurs : c'est la valeur extrême qui avait été détectée plus tôt. Ce point fera sans doute gonfler la variance résiduelle du groupe auquel il appartient.

Des tests formels nous aideront à confirmer ou infirmer les conclusions faites à partir de ces graphiques.

- Test de normalité sur les résidus de l'ANOVA.

```
shapiro.test(residuals(anova.model1))
```

```
Shapiro-Wilk normality test
```

```
data: residuals(anova.model1)
W = 0.91571, p-value = 1.63e-06
```

Ce test confirme nos soupçons: les résidus ne sont pas distribués normalement. Il faut cependant garder à l'esprit que la puissance est grande et que même de petites déviations de la normalité sont suffisantes pour rejeter l'hypothèse nulle.

- Test de l'hypothèse d'égalité des variances (homoscedasticité):

```
leveneTest(fklngh ~ year, data = Dam10dat)
```

| | Df | F value | Pr(>F) |
|-------|-----|---------|-----------|
| group | 3 | 2.8159 | 0.0423438 |
| | 114 | NA | NA |

La valeur de p vous dit que vous pouvez rejeter l'hypothèse nulle qu'il n'y a aucune différence dans les variances entre les années. Alors, nous concluons que les variances ne sont pas homogènes.

11.2.3. Faire l'ANOVA

- Faites une ANOVA de fklnght en choisissant / présumant pour l'instant que les conditions d'application sont suffisamment remplies. Que concluez-vous?

```
summary(anova.model1)
```

Call:

```
lm(formula = fklngh ~ year, data = Dam10dat)
```

Residuals:

| Min | 1Q | Median | 3Q | Max |
|----------|---------|---------|--------|---------|
| -11.2116 | -2.6866 | -0.7116 | 2.2103 | 26.7885 |

Coefficients:

| | Estimate | Std. Error | t value | Pr(> t) |
|----------------|--|------------|---------|-------------|
| (Intercept) | 48.0243 | 0.8566 | 56.061 | < 2e-16 *** |
| year1958 | 0.1872 | 1.3335 | 0.140 | 0.88859 |
| year1965 | -5.5077 | 1.7310 | -3.182 | 0.00189 ** |
| year1966 | -3.3127 | 1.1684 | -2.835 | 0.00542 ** |
| --- | | | | |
| Signif. codes: | 0 *** 0.001 ** 0.01 * 0.05 . 0.1 ' ' 1 | | | |

Residual standard error: 5.211 on 114 degrees of freedom

Multiple R-squared: 0.1355, Adjusted R-squared: 0.1128

F-statistic: 5.957 on 3 and 114 DF, p-value: 0.0008246

- Coefficients: Estimates:** Les 4 coefficients peuvent être utilisés pour obtenir les valeurs prédites par le modèle (i.e., les moyennes de chaque groupe). La longueur à la fourche (fklngh) moyenne de la première année (1954) est 48.0243. Les coefficients pour les 3 autres années sont la différence entre la moyenne de l'année en question et la moyenne de 1954. La moyenne pour 1965 est $48.0243 - 5.5077 = 42.5166$. Pour chaque coefficient, on a également accès à **l'erreur-type**, une **valeur de t** et la **probabilité qui lui est associée** (pour H₀ que le coefficient est 0). Les poissons étaient plus petits après la construction du barrage qu'en 1954. Il faut néanmoins prendre ces p-valeurs avec un grain de sel, car elles ne sont pas corrigées pour les

comparaisons multiples et elles ne constituent qu'un sous-ensemble des comparaisons possibles. En général, je porte peu d'attention à cette partie des résultats imprimés et me concentre sur ce qui suit.

- *Residual standard error*: La racine carrée de la variance des résidus (valeurs *observées* moins valeurs *prédites*) qui correspond à la **variabilité inexpliquée par le modèle** (variation de la taille des poissons capturés la même année).
- *Multiple R-squared*: Le R-carré est la proportion de la variabilité de la variable dépendante qui peut être **expliquée par le modèle**. Ici, le modèle explique 13.5% de la variabilité. Les différences de taille d'une année à l'autre sont relativement petites lorsqu'on les compare à la variation de taille entre les poissons capturés la même année.
- *F-Statistic*: La p-valeur associée au test “omnibus” que toutes les moyennes sont égales. Ici, p est beaucoup plus petit que 0.05 et on rejette H0 pour conclure que fklnghth varie selon les années.

La fonction `anova()` produit le tableau d'ANOVA standard qui contient la plupart de ces informations :

```
anova(anova.model1)
```

| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|-----------|-----|-----------|-----------|---------|-----------|
| year | 3 | 485.2642 | 161.75472 | 5.95744 | 0.0008246 |
| Residuals | 114 | 3095.2955 | 27.15171 | NA | NA |

La variabilité totale de fklnghth, mesurée par la somme du carré des écarts (Sum Sq) est partitionnée en ce qui peut être expliqué par l'année (485.26) et la variabilité résiduelle inexplicée (3095.30). L'année explique bien $\frac{485.26}{3095.30+485.26} = 0.1355$ or 13.55% de la variabilité. Le carré moyen des résidus (Residual Mean Sq) correspond à leur variance.

11.2.4. Les comparaisons multiples

- La fonction `pairwise.t.test()` peut être utilisée pour comparer des moyennes et ajuster (ou non) les probabilités pour le nombre de comparaisons en utilisant l'une des options pour l'argument `p.adj` :

Comparez toutes les moyennes sans ajuster les probabilités :

```
pairwise.t.test(Dam10dat$fklnghth, Dam10dat$year,
  p.adj = "none"
)
```

Pairwise comparisons using t tests with pooled SD

data: Dam10dat\$fklnghth and Dam10dat\$year

| | 1954 | 1958 | 1965 |
|------|--------|--------|--------|
| 1958 | 0.8886 | - | - |
| 1965 | 0.0019 | 0.0022 | - |
| 1966 | 0.0054 | 0.0079 | 0.1996 |

P value adjustment method: none

L'option "bonf" ajuste les p-valeurs avec la correction de Bonferroni. Ici, il y a 6 valeurs p calculées, et la correction de Bonferroni revient à simplement multiplier la p-valeur par 6 (sauf si le résultat est supérieur à 1. Si tel est le cas, la "p-value" ajustée est 1).

```
pairwise.t.test(Dam10dat$fklnghth, Dam10dat$year,  
  p.adj = "bonf"  
)
```

Pairwise comparisons using t tests with pooled SD

data: Dam10dat\$fklnghth and Dam10dat\$year

| | 1954 | 1958 | 1965 |
|------|-------|-------|-------|
| 1958 | 1.000 | - | - |
| 1965 | 0.011 | 0.013 | - |
| 1966 | 0.033 | 0.047 | 1.000 |

P value adjustment method: bonferroni

L'option "holm" correspond à la correction séquentielle de Bonferroni dans laquelle les p-valeurs sont ordonnées de (i=1) la plus faible à (N) la plus grande. La correction pour les p-valeurs est $(N - i + 1)$. Ici, il y a $N = 6$ paires de

moyennes qui sont comparées. La plus petite valeur de p non corrigée est 0.0019 pour 1954 vs 1965. La p-valeur corrigée est donc $0.0019 * (6 - 1 + 1) = 0.011$. La seconde plus petite p-valeur est 0.0022. Sa p-valeur corrigée est $0.0022 * (6 - 2 + 1) = 0.011$. Pour la p-valeur la plus élevée, la correction est $(N - N + 1) = 1$, donc la p-valeur corrigée est égale à la p-valeur brute.

```
pairwise.t.test(Dam10dat$fklnngth, Dam10dat$year,
  p.adj = "holm"
)
```

```
Pairwise comparisons using t tests with pooled SD

data: Dam10dat$fklnngth and Dam10dat$year

  1954   1958   1965
1958  0.889  -
1965  0.011  0.011  -
1966  0.022  0.024  0.399

P value adjustment method: holm
```

L'option "fdr" sert à contrôler le “false discovery rate”.

```
pairwise.t.test(Dam10dat$fklnngth, Dam10dat$year,
  p.adj = "fdr"
)
```

```
Pairwise comparisons using t tests with pooled SD

data: Dam10dat$fklnngth and Dam10dat$year

  1954   1958   1965
1958  0.8886  -      -
```

```
1965 0.0066 0.0066 -
1966 0.0108 0.0119 0.2395
```

P value adjustment method: fdr

Les quatre méthodes mènent ici à la même conclusion: les poissons sont plus gros après la construction du barrage et toutes les comparaisons entre les années 50 et 60 sont significatives alors que les différences entre 54 et 58 ou 65 et 66 ne le sont pas. La conclusion ne dépend pas du choix de méthode.

Dans d'autres situations, vous pourriez obtenir des résultats contradictoires. Alors, quelle méthode choisir ? Les p-valeurs qui ne sont pas corrigées sont certainement suspectes lorsqu'il y a plusieurs comparaisons. D'un autre côté, la correction de *Bonferroni* est conservatrice et le devient encore plus lorsqu'il y a de très nombreuses comparaisons. Des travaux récents suggèrent que la correction **fdr** est un bon compromis lorsqu'il y a beaucoup de comparaisons.

La méthode de *Tukey* est l'une des plus populaires et est facile à utiliser en R (notez cependant qu'il y a un petit bug qui se manifeste quand la variable indépendante peut ressembler à un nombre plutôt qu'un facteur, ce qui explique la petite pirouette avec `paste0()` dans le code pour ajouter la lettre `m` avant le premier chiffre):

```
Dam10dat$myyear <- as.factor(paste0("m", Dam10dat$year))
TukeyHSD(aov(fklngh ~ myyear, data = Dam10dat))
```

```
Tukey multiple comparisons of means
95% family-wise confidence level

Fit: aov(formula = fklngh ~ myyear, data = Dam10dat)

$myyear
      diff        lwr       upr     p adj
m1958-m1954 0.1872141 -3.289570 3.6639986 0.9990071
m1965-m1954 -5.5076577 -10.021034 -0.9942809 0.0100528
m1966-m1954 -3.3126964 -6.359223 -0.2661701 0.0274077
m1965-m1958 -5.6948718 -10.436304 -0.9534397 0.0116943
m1966-m1958 -3.4999106 -6.875104 -0.1247171 0.0390011
m1966-m1965 2.1949612 -2.240630 6.6305526 0.5710111
```

```
plot(TukeyHSD(aov(fklngh ~ myyear, data = Dam10dat)))
```

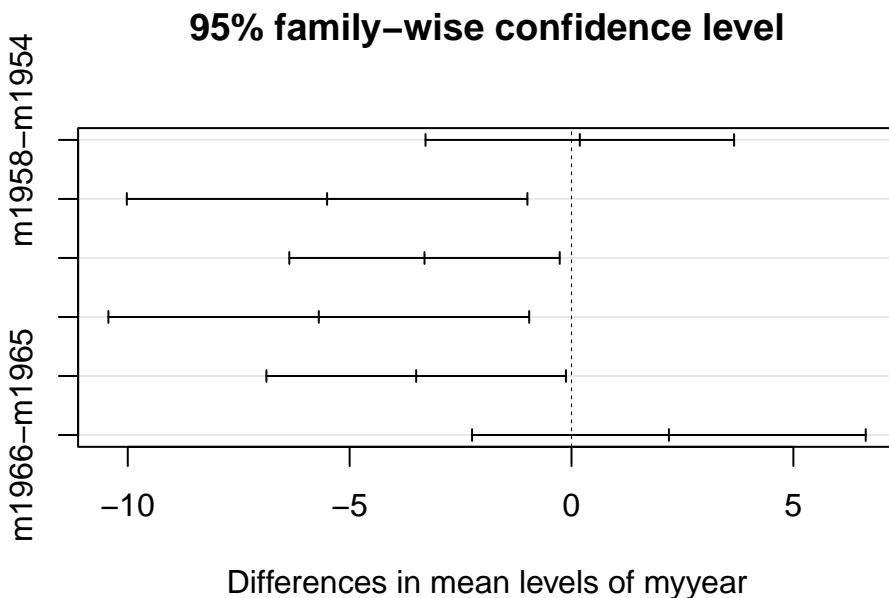


Figure 11.4.: Différence annuelles dans la longueur des esturgeons.

Les intervalles de confiance, corrigés pour les comparaisons multiples par la méthode de Tukey, sont illustrés pour les différences entre années. Malheureusement les légendes ne sont pas complètes, mais l'ordre est le même que dans le tableau précédent.

Le paquet `multcomp`  peut produire de meilleurs graphiques, mais requiert un peu plus de code:

```
# Autre manière de faire une comparaison multiple corrigée par la méthode de Tukey
# Ajustez une ANOVA à un critère de classification
anova.fkl.vs.year <- aov(aov(fklngh ~ myyear, data = Dam10dat))

# Établissez des comparaisons par paires pour le facteur `year`


meandiff <- glht(anova.fkl.vs.year, linfct = mcp(
  myyear =
    "Tukey"
))
confint(meandiff)
```

Simultaneous Confidence Intervals

Multiple Comparisons of Means: Tukey Contrasts

```
Fit: aov(formula = aov(fklngh ~ myyear, data = Dam10dat))
```

```
Quantile = 2.5946
```

```
95% family-wise confidence level
```

Linear Hypotheses:

| | Estimate | lwr | upr |
|--------------------|----------|----------|---------|
| m1958 - m1954 == 0 | 0.1872 | -3.2725 | 3.6470 |
| m1965 - m1954 == 0 | -5.5077 | -9.9989 | -1.0164 |
| m1966 - m1954 == 0 | -3.3127 | -6.3443 | -0.2811 |
| m1965 - m1958 == 0 | -5.6949 | -10.4131 | -0.9767 |
| m1966 - m1958 == 0 | -3.4999 | -6.8586 | -0.1413 |
| m1966 - m1965 == 0 | 2.1950 | -2.2189 | 6.6088 |

```
plot(meandiff)
```

95% family-wise confidence level

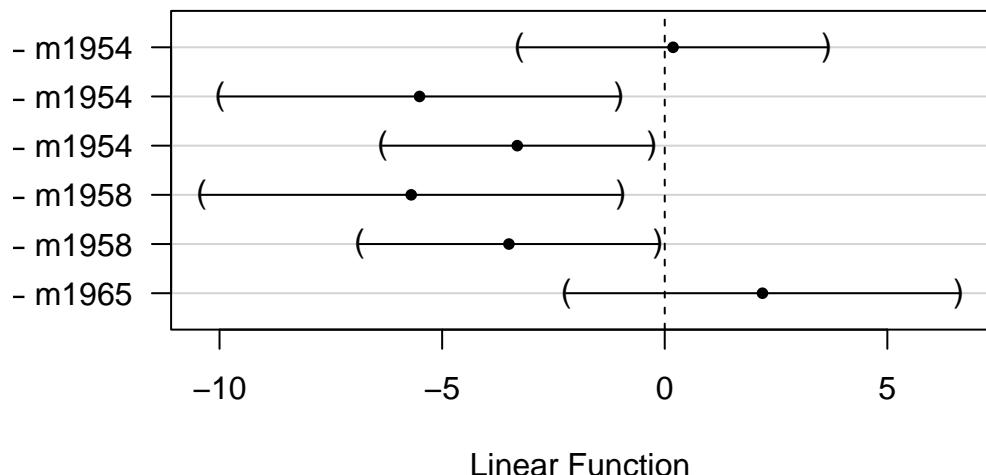


Figure 11.5.: Différence annuelles dans la longueur des esturgeons (avec `multcomp`

C'est un peu mieux, mais ce qui le serait encore plus c'est un graphique des moyennes, avec leurs intervalles de confiance ajustés pour les comparaisons multiples:

```
# Calculez et représentez les moyennes et l'IC de Tukey
means <- glht(
  anova.fkl.vs.year,
  linfct = mcp(myyear = "Tukey")
)
cimeans <- cld(means)

# Utilisez une marge supérieure suffisamment grande
# Graphique
old.par <- par(mai = c(1, 1, 1.25, 1))
plot(cimeans)
```

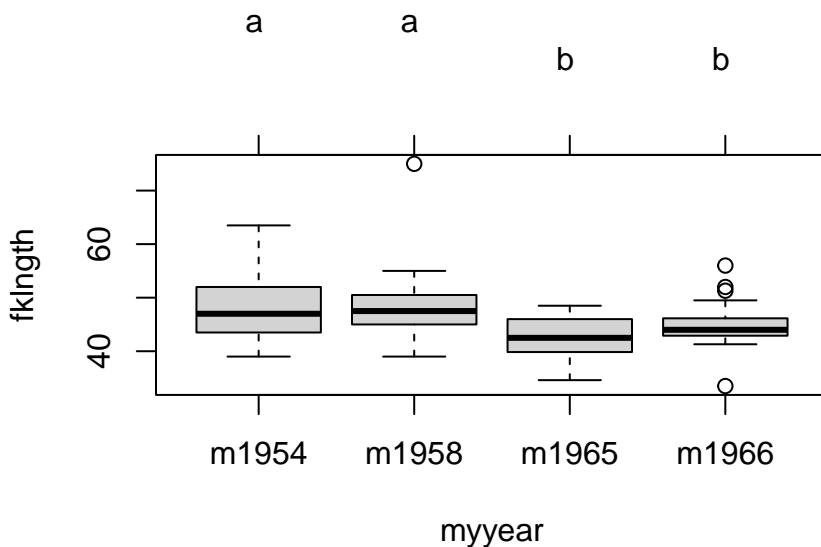


Figure 11.6.: Différence annuelles dans la longueur des esturgeons.

Notez les lettres au dessus du graphique: les années étiquetées avec la même lettre ne diffèrent pas significativement l'une de l'autre.

11.3. Transformations de données et ANOVA non-paramétrique

Dans l'exemple précédent sur les différences annuelles de longueur (variable fklngh), on a noté que les conditions d'application de l'ANOVA n'étaient pas remplies. Si les données ne remplissent pas les conditions de l'ANOVA paramétrique, il y a 3 options :

1. Ne rien faire. Si les effectifs dans chaque groupe sont grands, on peut relaxer les conditions d'application car l'ANOVA est alors assez robuste aux violations de normalité (mais moins aux violations d'homoscedasticité),
 2. Transformer les données
 3. Faire une analyse non-paramétrique.
- Refaites l'ANOVA de la section précédente après avoir transformé `fklngth` en logarithme de base 10. Avec les données transformées, est-ce que les problèmes qui avaient été identifiés disparaissent ?

```
# Ajustez un modèle anova avec une transformation log10 de `fklngth` et tracez les graphiques diag
par(mfrow = c(2, 2))
anova.model2 <- lm(log10(fklngth) ~ year, data = Dam10dat)
plot(anova.model2)
```

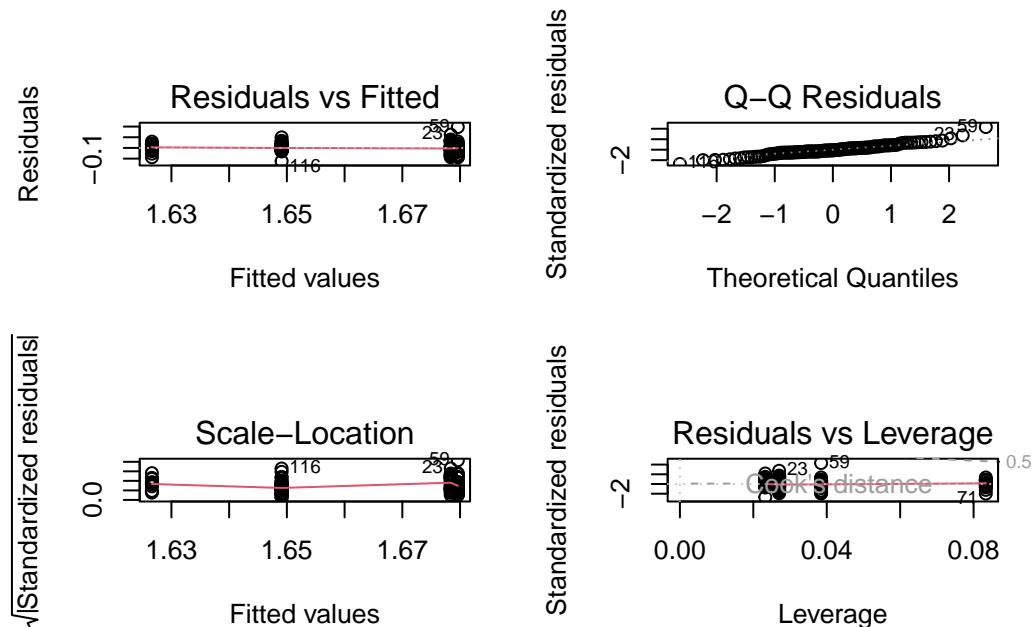


Figure 11.7.: Conditions d'application de l'ANOVA.

Les graphiques sont à peine mieux ici. Si on fait le test Shapiro-Wilk sur les résidus, on obtient:

```
shapiro.test(residuals(anova.model2))
```

```
Shapiro-Wilk normality test

data: residuals(anova.model2)
W = 0.96199, p-value = 0.002048
```

Donc, on a toujours des problèmes avec la normalité et on est juste sur le seuil de décision pour l'égalité des variances. Vous avez le choix à ce point:

1. Essayer de trouver une autre transformation pour mieux rencontrer les conditions d'application.
 2. Assumer que les données sont suffisamment proche des conditions d'application.
 3. Faire une ANOVA non-paramétrique.
- L'analogie non-paramétrique de l'ANOVA à un critère de classification le plus employé est le test de **Kruskall-Wallis**. Faites ce test sur `fklngth` et comparez les résultats à ceux de l'analyse paramétrique. Que concluez-vous?

```
kruskal.test(fklngth ~ year, data = Dam10dat)
```

```
Kruskal-Wallis rank sum test

data: fklngth by year
Kruskal-Wallis chi-squared = 15.731, df = 3, p-value = 0.001288
```

La conclusion est donc la même qu'avec l'ANOVA paramétrique: on rejette l'hypothèse nulle que le rang moyen est le même pour chaque année. Donc, même si les conditions d'application de l'analyse paramétrique n'étaient pas parfaitement rencontrées, les conclusions sont les mêmes, ce qui illustre la robustesse de l'ANOVA paramétrique.

11.4. Examen des valeurs extrêmes

Vous devriez avoir remarqué au cours des analyses précédentes qu'il y avait peut-être des valeurs extrêmes dans les données. Ces points étaient évidents dans le “Box Plot” de `fklngth` par `year` et ont été notés comme les points 59, 23, et 87 dans les diagrammes de probabilité des résidus et dans le diagramme de dispersion des résidus et des valeurs estimées. En général, vous devez avoir de très bonnes raisons pour enlever des valeurs extrêmes de la base de données (i.e. vous savez qu'il y a eu une erreur avec un cas). Cependant, il est quand même toujours valable de voir comment l'analyse change enlevant des valeurs extrêmes de la base de données.

- Répétez l'ANOVA originale sur `fklngth` et `year` mais faites le avec un sous-ensemble de données sans les valeurs extrêmes. Est-ce que les conclusions ont changé?

```
Damsubset <- Dam10dat[-c(23, 59, 87), ] # enlevez les observations 23, 59 et 87
aov.Damsubset <- aov(fklngh ~ as.factor(year), Damsubset)
summary(aov.Damsubset)
```

| | Df | Sum Sq | Mean Sq | F value | Pr(>F) | | | | | | |
|-----------------|-----|--------|---------|---------|--------------|-----|------|------|-----|-----|---|
| as.factor(year) | 3 | 367.5 | 122.50 | 6.894 | 0.000267 *** | | | | | | |
| Residuals | 111 | 1972.4 | 17.77 | | | | | | | | |
| --- | | | | | | | | | | | |
| Signif. codes: | 0 | '***' | 0.001 | '**' | 0.01 | '*' | 0.05 | '. ' | 0.1 | ' ' | 1 |

```
shapiro.test(residuals(aov.Damsubset))
```

Shapiro-Wilk normality test

```
data: residuals(aov.Damsubset)
W = 0.98533, p-value = 0.2448
```

```
leveneTest(fklngh ~ year, Damsubset)
```

| | Df | F value | Pr(>F) |
|-------|-----|----------|-----------|
| group | 3 | 4.623721 | 0.0043666 |
| | 111 | NA | NA |

L'élimination de trois valeurs extrêmes améliore un peu les choses, mais ce n'est pas parfait. On a toujours un problème avec les variances, mais les résidus sont maintenant normaux. Cependant, le fait que la conclusion qu'on tire de l'ANOVA originale ne change pas enlevant les points renforce le fait qu'on n'a pas de raison valable pour enlever ces points.

11.5. Test de permutation

Commande R pour un test de permutation d'une ANOVA à un critère de classification.

```
#####
# Test de permutation pour ANOVA à un critère de classification
# modifié à partir du code écrit par David C. Howell
# http://www.uvm.edu/~dhowell/StatPages/More_Stuff/Permutation%20Anova/PermTestsAnova.html
# Définition du nombre de permutation
nreps <- 500
# Pour simplifier la réutilisation de ce code, assignez le jeu de donnée à utiliser à l'objet "mydata"
mydata <- Dam10dat
# Stockez la formule de votre modèle dans l'objet myformula
myformula <- as.formula("fklngth ~ year")
# Stockez la variable dépendante dans l'objet mydep
mydep <- mydata$fklngth
# Stockez la variable indépendante dans l'objet myindep
myindep <- as.factor(mydata$year)
#####

# Vous ne devriez pas avoir besoin de modifier le code ci-dessous

#####
# Calculez la valeur F observée pour l'échantillon original
mod1 <- lm(myformula, data = mydata) # Anova Standard
ANOVA <- summary(aov(mod1)) # Stockez le résumé dans une variable
observedF <- ANOVA[[1]]$"F value"[1] # Stockez la valeur F observée
# Affichez les résultats de l'ANOVA standard
cat(
  " L'ANOVA standard pour ces données est la suivante ",
  "\n"
)

print(ANOVA, "\n")
cat("\n")
cat("\n")

print("Rééchantillonnage comme dans Manly avec échantillonnage non restreint des observations. ")
```

```

# Commençons le rééchantillonnage

Fboot <- numeric(nreps) # initialisez le vecteur pour qu'il soit permuté

values

Fboot[1] <- observedF

for (i in 2:nreps) {

  newdependent <- sample(mydep, length(mydep)) # Randomiser la variable dépendante

  var

  mod2 <- lm(newdependent ~ myindep) # Réajuste le modèle

  b <- summary(aov(mod2))

  Fboot[i] <- b[[1]]$"F value"[1] # Stock la statistique F

}

permprob <- length(Fboot[Fboot >= observedF]) / nreps

cat(
  " The permutation probability value is: ", permprob,
  "\n"
)

# Fin du bloc de code de permutation.

```

Version lmPerm du test de permutation :

```

## Version lmPerm du test de permutation

library(lmPerm)

# Pour généraliser, assignez votre jeu de donnée à l'objet mydata
# et la formule du modèle à myformula

mydata <- Dam10dat

myformula <- as.formula("fklnghth ~ year")

# Ajustez le modèle souhaité sur le jeu de donnée souhaité

mymodel <- lm(myformula, data = mydata)

# Calcule de la valeur p permuté

anova(lmp(myformula, data = mydata, perm = "Prob", center = FALSE, Ca = 0.001))

```

Chapitre 12

ANOVA à critères multiples : plans factoriels et hiérarchiques

Objectifs de ce chapitre :

- Utiliser R pour faire une ANOVA paramétrique d'un plan factoriel avec deux facteurs de classification et réplication
- Utiliser R pour faire une ANOVA paramétrique d'un plan factoriel avec deux facteurs de classification sans réplication
- Utiliser R pour faire une ANOVA paramétrique d'un plan hiérarchique avec réplication
- Utiliser R pour faire une ANOVA non paramétrique avec deux facteurs de classification
- Utiliser R pour faire des comparaisons multiples

Il existe une très grande variété de plans (designs) d'ANOVA que R peut analyser. Cet exercice n'est qu'une introduction aux plans les plus communs.

12.1. Paquets et données requises pour le labo

Pour ce laboratoire, vous aurez besoin de :

- Paquets R :
 - `multicomp` 
 - `car` 
 - `tidyverse` 

- Jeux de données :

- “Stu2wdat.csv”
- “Stu2mdat.csv”
- “nr2wdat.csv”
- “nestdat.csv”
- “wmcdat2.csv”
- “wmc2dat2.csv”

```
library(multcomp)
library(car)
library(tidyverse)
```

12.2. Plan factoriel à deux facteurs de classification et réPLICATION

Il est fréquent de vouloir analyser l'effet de plusieurs facteurs simultanément. L'ANOVA factorielle à deux critères de classification permet d'examiner deux facteurs à la fois, mais la même approche peut être utilisée pour 3, 4 ou même 5 facteurs quoique l'interprétation des résultats devienne beaucoup plus complexe.

Supposons que vous êtes intéressés par l'effet de deux facteurs : le **site** (location, Cumberland House ou The Pas) et le **sex** (sex, mâle ou femelle) sur la **taille** des esturgeons. Comme l'effectif n'est pas le même pour tous les groupes, c'est un plan qui n'est pas balancé. Notez aussi qu'il y a des valeurs manquantes pour certaines variables, ce qui veut dire que chaque mesure n'a pas été effectuée sur chaque poisson.

12.2.1. ANOVA à effets fixes (Modèle I)

- Examinez d'abord les données avec des “boxplots” de la taille (rdwght) selon le sex et la location des données du fichier Stu2wdat.csv .

Solution

```
Stu2wdat <- read.csv("data/Stu2wdat.csv")

ggplot(Stu2wdat, aes(x = sex, y = rdwght)) +
  geom_boxplot(notch = TRUE) +
  facet_grid(~location)
```

Warning: Removed 4 rows containing non-finite outside the scale range
`stat_boxplot()`).

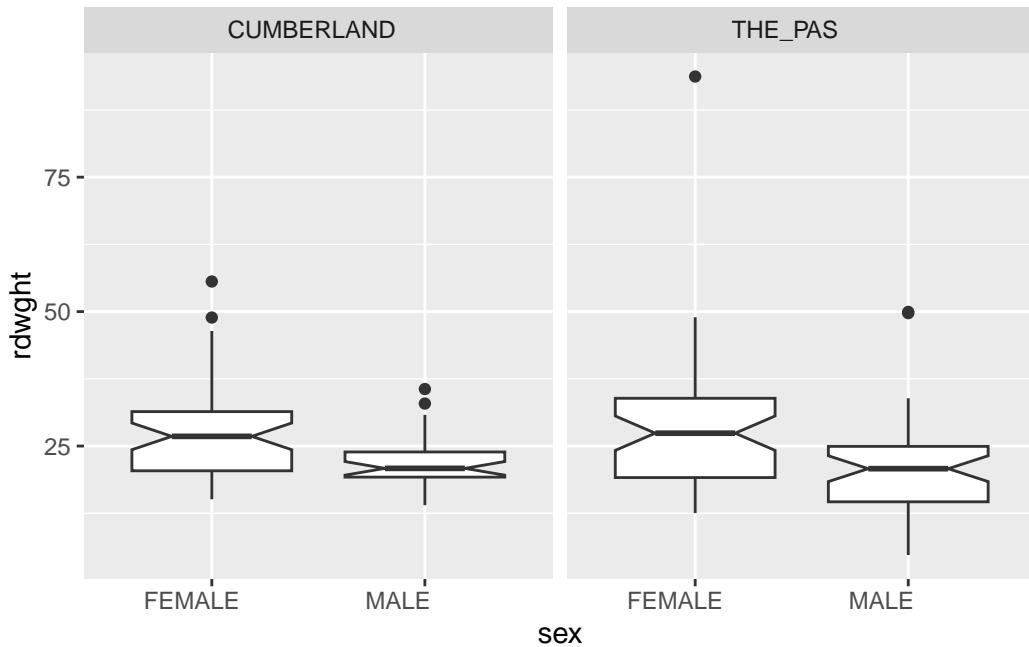


Figure 12.1.

Les graphiques montrent qu'aux deux sites les femelles sont probablement plus grandes que les mâles, mais que les tailles ne varient pas beaucoup d'un site à l'autre. La présence de valeurs extrêmes sur ces graphiques suggère qu'il y aura peut-être des problèmes avec la condition de **normalité** des résidus.

- Résumez les statistiques de taille (rdwght) selon le sex et la location.

Moyenne, écart-type, taille d'échantillon

```
Stu2wdat %>%
  group_by(sex, location) %>%
  summarise(
    mean = mean(rdwght, na.rm = TRUE), sd = sd(rdwght, na.rm = TRUE), n = n()
  )
```

``summarise()`` has grouped output by 'sex'. You can override using the ``.groups`` argument.

| sex | location | mean | sd | n |
|--------|------------|----------|-----------|----|
| FEMALE | CUMBERLAND | 27.37347 | 9.331438 | 51 |
| FEMALE | THE_PAS | 27.97717 | 12.533105 | 55 |
| MALE | CUMBERLAND | 22.14118 | 4.789390 | 34 |
| MALE | THE_PAS | 20.64652 | 9.917066 | 46 |

Ces résultats supportent l'interprétation des “boxplots” : Les femelles sont plus grosses que les mâles, et la différence de taille entre les deux sites n'est pas énorme.

- Faites une ANOVA factorielle à deux critères de classification avec les données de `Stu2wdat.csv` :

```
# Ajuster un modèle (ANOVA) et faites le graphiques des résidues
# Mais d'abord, sauvegardez les paramètres graphiques actuels
opar <- par()

anova.model1 <- lm(rdwght ~ sex + location + sex:location,
  contrasts = list(sex = contr.sum, location = contr.sum),
  data = Stu2wdat
)
anova(anova.model1)
```

| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|--------------|----|-------------|-------------|------------|-----------|
| sex | 1 | 1839.552607 | 1839.552607 | 18.6784735 | 0.0000257 |
| location | 1 | 4.261586 | 4.261586 | 0.0432714 | 0.8354530 |
| sex:location | 1 | 48.692262 | 48.692262 | 0.4944121 | 0.4828844 |

| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|-----------|-----|--------------|-----------|---------|--------|
| Residuals | 178 | 17530.359982 | 98.485168 | NA | NA |

Avertissement

Attention, R imprime les sommes des carrés *séquentielles* (**Type I**) les carrés moyens et probabilités associés. Vous ne pouvez pas vous y fier si votre plan d'expérience n'est *pas parfaitement balancé*. Dans cet exemple, le nombre de poissons capturés change selon le site et le sexe et le plan d'expérience n'est donc pas balancé.

Vous devez extraire les sommes de carrés *partielles* (**Type III**). Le moyen le plus simple est d'utiliser la fonction `Anova()` du paquet car (notez une différence subtile mais fondamentale, `Anova()` n'est pas la même chose que `anova()`, R est impitoyable et distingue les majuscules des minuscules). Malheureusement, ça ne suffit pas. Il faut également spécifier le type de contraste dans le modèle avec l'argument

```
contrasts = list(sex = contr.sum, location = contr.sum)
```

```
library(car)
Anova(anova.model1, type = 3)
```

| | Sum Sq | Df | F value | Pr(>F) |
|--------------|--------------|-----|--------------|------------|
| (Intercept) | 1.065073e+05 | 1 | 1081.4551655 | 0.00000000 |
| sex | 1.745358e+03 | 1 | 17.7220422 | 0.0000405 |
| location | 8.778349e+00 | 1 | 0.0891337 | 0.7656296 |
| sex:location | 4.869226e+01 | 1 | 0.4944121 | 0.4828844 |
| Residuals | 1.753036e+04 | 178 | NA | NA |

Suite à l'ANOVA, on accepte deux hypothèses nulles:

- (1) L'effet du sexe ne varie pas entre les sites (pas d'interaction significative)
- (2) Il n'y a pas de différence de taille des esturgeons (peu importe le sexe) entre les deux sites.

D'un autre coté, on rejette l'hypothèse nulle qu'il n'y a pas de différence de taille entre les esturgeons mâles et les femelles, tel que suggéré par les graphiques.

```
par(mfrow = c(2, 2))
plot(anova.model1)
```

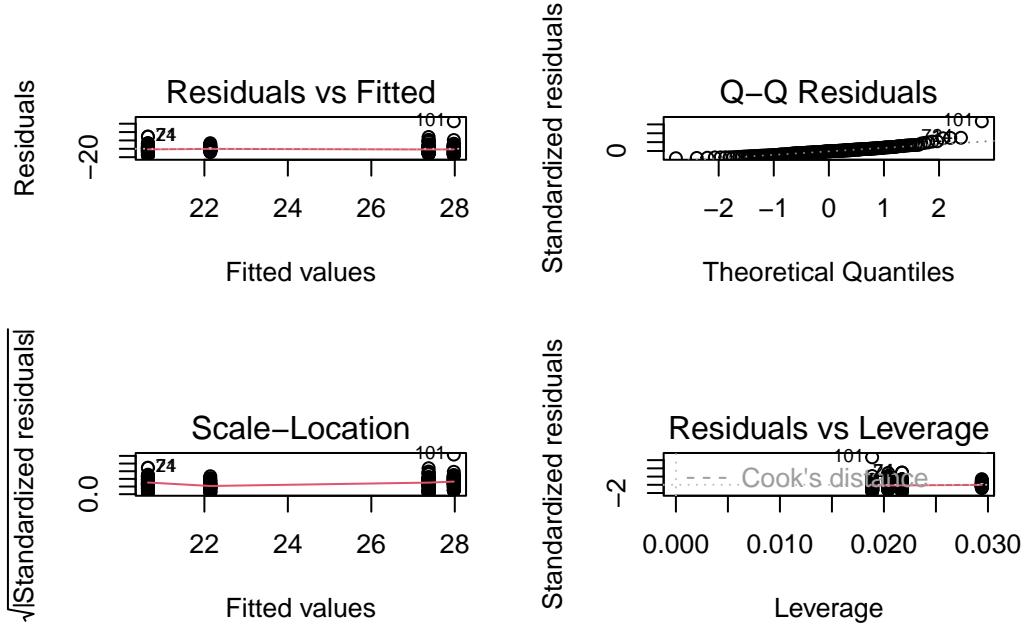


Figure 12.2.: Conditions d'application ANOVA model1

Mais ! Comme d'habitude, on ne peut pas accepter les résultats sans d'abord nous assurer que les conditions d'applications sont respectées.

- **Normalité :** L'examen des graphiques des résidus ci-dessus montre que l'écart à la normalité reste raisonnable, avec seulement 3 potentielles valeurs aberrantes sur le graphique QQ (cas 101, 24 & 71). Cependant leurs distances de Cook ne sont pas importantes (On ne voit même pas la borne 0.5 sur le graphique), il y a donc peu d'indication que ces valeurs sont préoccupantes pour le suite. Si on test la normalité, on obtient:

```
shapiro.test(residuals(anova.model1))
```

Shapiro-Wilk normality test

```
data: residuals(anova.model1)
W = 0.87213, p-value = 2.619e-11
```

Il y a évidence que les résidus ne sont pas distribués normalement ($p < 0.05$).

- **Homoscédasticité** : Le graphique des résidus par rapport à l'ajustement montre que la dispersion des résidus est à peu près égales pour l'ensemble des valeurs ajustées (à l'exception de quelques cas encore un fois). Nous utilisons le test de Levene pour tester cela (de la même façon qu'on a fait pour l'ANOVA à un critère de classification).

```
library(car)
leveneTest(rdwght ~ sex * location, data = Stu2wdat)
```

| | Df | F value | Pr(>F) |
|-------|-----|----------|-----------|
| group | 3 | 3.852621 | 0.0105483 |
| | 178 | NA | NA |

Si les résidus étaient homoscédastiques, on accepterait l'hypothèse nulle que la moyenne des valeurs absolues des résidus ne varie pas entre les sexes et les sites (i.e., `sex:location`). Le tableau d'ANOVA ci-dessus montre que l'hypothèse est rejetée. Il y a donc hétéroscédasticité.

En bref, nous avons **plusieurs conditions d'application non respectées**. La question qui reste est : ces violations sont-elles suffisantes pour invalider nos conclusions ?

🔥 Exercice

Répétez la même analyse avec les données du fichier `stu2mdat.csv`. Que concluez-vous? Supposons que vous vouliez comparer la taille des mâles et des femelles. Comment cette comparaison diffère entre les deux jeux de données ?

💡 Solution

```
Stu2mdat <- read.csv("data/Stu2mdat.csv")
anova.model2 <- lm(
  formula = rdwght ~ sex + location + sex:location,
  contrasts = list(sex = contr.sum, location = contr.sum),
  data = Stu2mdat
)
summary(anova.model2)
Anova(anova.model2, type = 3)
```

Call:

```
lm(formula = rdwght ~ sex + location + sex:location, data = Stu2mdat,  
contrasts = list(sex = contr.sum, location = contr.sum))
```

Residuals:

| Min | 1Q | Median | 3Q | Max |
|---------|--------|--------|-------|--------|
| -15.917 | -6.017 | -0.580 | 4.445 | 65.743 |

Coefficients:

| | Estimate | Std. Error | t value | Pr(> t) | | | | | | | |
|----------------|----------|------------|---------|--------------|------|-----|------|------|-----|-----|---|
| (Intercept) | 24.5346 | 0.7461 | 32.885 | < 2e-16 *** | | | | | | | |
| sex1 | -0.5246 | 0.7461 | -0.703 | 0.483 | | | | | | | |
| location1 | 0.2227 | 0.7461 | 0.299 | 0.766 | | | | | | | |
| sex1:location1 | 3.1407 | 0.7461 | 4.210 | 4.05e-05 *** | | | | | | | |
| --- | | | | | | | | | | | |
| Signif. codes: | 0 | '***' | 0.001 | '**' | 0.01 | '*' | 0.05 | '. ' | 0.1 | ' ' | 1 |

Residual standard error: 9.924 on 178 degrees of freedom

(4 observations deleted due to missingness)

Multiple R-squared: 0.09744, Adjusted R-squared: 0.08223

F-statistic: 6.405 on 3 and 178 DF, p-value: 0.0003817

Notez que cette fois les femelles sont plus grandes que les mâles à Cumberland House, mais que c'est le contraire à The Pas. Quel est le résultat de l'ANOVA (n'oubliez pas qu'il faut des sommes des carrés de **Type III** pour les résultats) ?

| | Sum Sq | Df | F value | Pr(>F) |
|--------------|--------------|-----|--------------|-----------|
| (Intercept) | 1.065073e+05 | 1 | 1081.4551655 | 0.0000000 |
| sex | 4.869226e+01 | 1 | 0.4944121 | 0.4828844 |
| location | 8.778349e+00 | 1 | 0.0891337 | 0.7656296 |
| sex:location | 1.745358e+03 | 1 | 17.7220422 | 0.0000405 |
| Residuals | 1.753036e+04 | 178 | NA | NA |

L'interaction (`sex:location`) est maintenant significative mais les effets principaux ne le sont pas.

- Il est utile ici de créer des graphiques pour les deux jeux de données pour comparer les interactions entre `sex` et `location`. Le graphique d'interaction montre les relations entre les moyennes de chaque combinaison de facteurs (appelées aussi les moyennes des cellules).

Générez un graphique illustrant les interactions en utilisant la fonction `allEffects()` du paquet `effects`  :

```
library(effects)
allEffects(anova.model1)
```

```
model: rdwght ~ sex + location + sex:location
```

```
sex*location effect
```

| | location | |
|--------|------------|----------|
| sex | CUMBERLAND | THE_PAS |
| FEMALE | 27.37347 | 27.97717 |
| MALE | 22.14118 | 20.64652 |

```
plot(allEffects(anova.model1), "sex:location")
```

sex*location effect plot

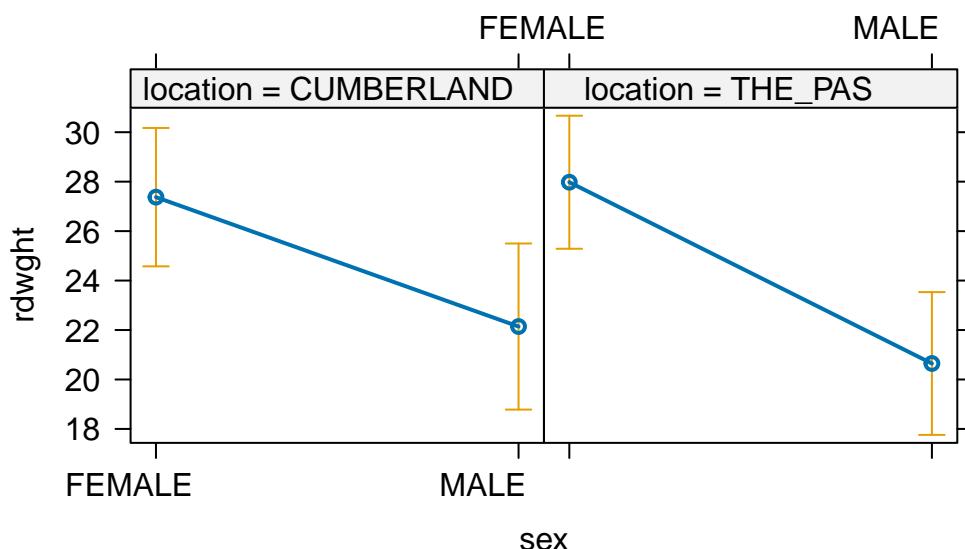


Figure 12.3.: Effet du sexe et du lieu sur le poids des esturgeons (1er modèle)

```
allEffects(anova.model2)
```

```
model: rdwght ~ sex + location + sex:location
```

```
sex*location effect
```

```
location
```

| sex | CUMBERLAND | THE_PAS |
|--------|------------|----------|
| FEMALE | 27.37347 | 20.64652 |
| MALE | 22.14118 | 27.97717 |

```
plot(allEffects(anova.model2), "sex:location")
```

sex*location effect plot

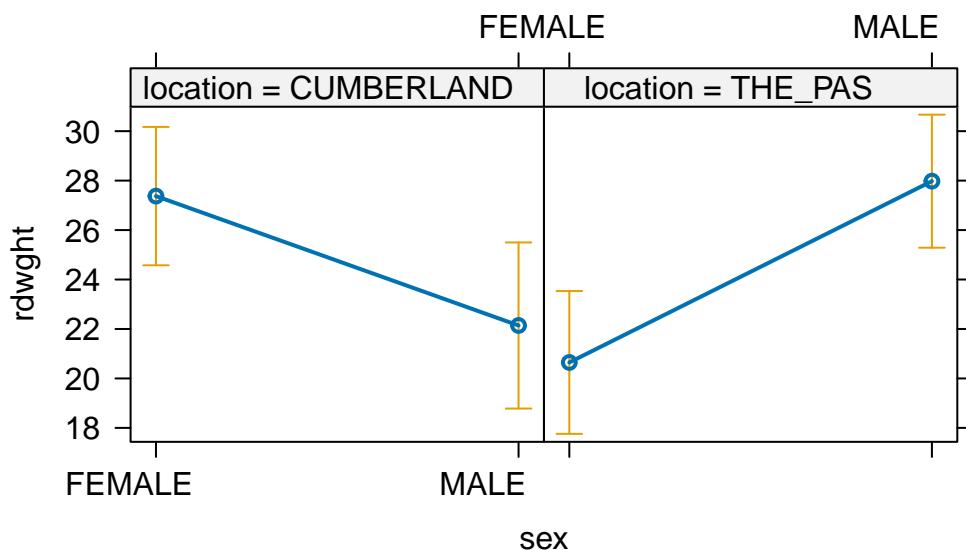


Figure 12.4.: Effet du sexe et du lieu sur le poids des esturgeons (2e modèle)

Il y a une différence importante entre les résultats obtenus avec `stu2wdat` et `stu2mdat`. Dans le premier cas, puisqu'il n'y a pas d'interaction, on peut regrouper les données des deux niveaux d'un facteur (le site, par exemple) pour éprouver l'hypothèse d'un effet de l'autre facteur (le sexe).

En fait, si on fait cela et que l'on calcule une ANOVA à un critère de classification (sex), on obtient:

```
Anova(aov(rdwght ~ sex, data = Stu2wdat), type = 3)
```

| | Sum Sq | Df | F value | Pr(>F) |
|-------------|-----------|-----|-----------|----------|
| (Intercept) | 78191.023 | 1 | 800.43979 | 0.00e+00 |
| sex | 1839.553 | 1 | 18.83146 | 2.38e-05 |
| Residuals | 17583.314 | 180 | NA | NA |

Notez que la somme des carrés des résidus (17,583) est presque égale à celle du modèle complet (17,530) de l'ANOVA factorielle à deux facteurs. C'est parce que dans cette anova factorielle, le terme d'interaction et le terme représentant l'effet du site n'expliquent qu'une partie infime de la variabilité.

D'un autre coté, si on essaie le même truc avec `stu2mdat`, on obtient:

```
Anova(aov(rdwght ~ sex, data = Stu2mdat), type = 3)
```

| | Sum Sq | Df | F value | Pr(>F) |
|-------------|------------|-----|------------|-----------|
| (Intercept) | 55251.2030 | 1 | 515.043549 | 0.0000000 |
| sex | 113.3992 | 1 | 1.057091 | 0.3052593 |
| Residuals | 19309.4672 | 180 | NA | NA |

Ici la somme des carrées des résidus (19,309) est beaucoup plus grande que celle de l'ANOVA factorielle (17,530) parce qu'une partie importante de la variabilité expliquée par le modèle est associée à l'interaction.

Notez que si on n'avait fait que cette analyse, on conclurait que les esturgeons mâles et femelles ont la même taille. Et pourtant, leur taille diffère : seulement la différence est à l'avantage des mâles à un site et à l'avantage des femelles à l'autre. Il est donc délicat d'interpréter l'effet principal (sexe) en présence d'une **interaction significative !**

12.2.2. ANOVA à effets mixtes (Modèle III)

Les analyses qui précèdent négligent un point important, lié au type de modèle ANOVA que l'on veut faire tourner.

Dans cet exemple, le site (`location`) pourrait être traité comme un facteur **aléatoire** et le sexe (`sex`) comme un effet **fixe**. Par conséquent le modèle approprié d'ANOVA est un modèle **mixte**.

Notez que dans toutes les analyses qui précèdent, R a traité tous les termes de cette ANOVA comme des effets fixes , et les termes principaux et d'interaction ont été testés en utilisant le carré moyen des résidus comme dénominateur des tests de F.

Cependant, pour une ANOVA mixte, ces effets devraient être testés en utilisant le carré moyen du terme d’interaction, ou en combinant la somme des carrés de l’erreur et de l’interaction (selon le statisticien consulté!).

En utilisant `Stu2wdat`, refaites un tableau d’ANOVA pour `rdwght` en considérant le site (`location`) comme facteur **aléatoire** et le sexe (`sex`) comme un facteur **fixe**.

Pour ce faire, vous devrez recalculer les valeurs de F pour `sex` et `location` en utilisant le carré moyen de l’interaction `sex:location` au lieu du carré moyen des résidus comme dénominateur.

Le mieux c’est de le faire à la main travaillant avec les sommes des carrés d’ANOVA de Type III.

Solution

```
Anova(anova.model1, type = 3)
```

| | Sum Sq | Df | F value | Pr(>F) |
|--------------|--------------|-----|--------------|-----------|
| (Intercept) | 1.065073e+05 | 1 | 1081.4551655 | 0.0000000 |
| sex | 1.745358e+03 | 1 | 17.7220422 | 0.0000405 |
| location | 8.778349e+00 | 1 | 0.0891337 | 0.7656296 |
| sex:location | 4.869226e+01 | 1 | 0.4944121 | 0.4828844 |
| Residuals | 1.753036e+04 | 178 | NA | NA |

Pour `sex`, la nouvelle valeur de F (le rapport des carrés moyens) est de

$$F = \frac{(1745/1)}{(49/1)} = 35.6$$

Pour obtenir la valeur de p correspondant à cette statistique F, on utilise la fonction de probabilité de la distribution de F :

```
pf(F, df1, df2, lower.tail = FALSE),
```

où F est la valeur de F calculée, et `df1` et `df2` sont les degrés de liberté du numérateur (`sex`) et dénominateur(`sex:location`).

```
pf(35.6, 1, 1, lower.tail = FALSE)
```

[1] 0.1057152

Notez que maintenant la valeur de p pour `sex` n'est plus significative.

C'est parce que le carré moyen de l'erreur dans l'ANOVA initiale est plus petit que celui associé à l'interaction, mais surtout parce que le nombre de degrés de liberté pour le dénominateur du test de F est passé de 178 à 1 seulement.

En général, c'est beaucoup plus difficile d'obtenir des résultats significatifs quand les degrés de liberté pour le dénominateur sont petits.

Note

Les modèles mixtes qui sont une généralisation de l'ANOVA à effets mixtes sont maintenant extrêmement bien développé et sont à favoriser lors d'analyse incluant des effets dit aléatoires.

12.3. Plan factoriel à 2 facteurs de classification sans réplication

Dans certains plans d'expérience il n'y a pas de réplicats pour chaque combinaison de facteurs, par exemple parce qu'il serait trop coûteux de faire plus d'une observation. L'ANOVA à deux critères de classification est quand même possible dans ces circonstances, mais il y a une limitation importante.

Avertissement

Étant donnée qu'il n'y a pas de réplication, on ne peut pas estimer la variance du terme d'erreur : on a simplement la somme des carrées des lignes, une somme des carrés des colonnes et une somme des carré restants. Cela a des implications importantes :

S'il y a une interaction dans un ANOVA de modèle III, seul l'effet fixe peut être testé (sur la somme des carrés restants) ; pour les ANOVA de modèle I, ou pour les effets aléatoires dans les ANOVA de modèle III, il n'est pas approprié de tester les effets principaux sur le reste, à moins d'être sur de l'absence d'interaction.

Un limnologue qui étudie Round Lake dans le Parc Algonquin prend une seule mesure de température (`temp`, °C) à 10 profondeurs différentes (`depth`, m) à quatre dates (`date`) au cours de l'été. Ses données sont dans le fichier `nr2wdat.csv`.

- Effectuez une ANOVA à 2 critères de classification en utilisant `temp` comme variable dépendante et `date` et `depth` comme variables indépendantes (vous devez changer le type de données pour `depth` pour que R traite cette variable comme un facteur et non pas une variable continue).

```
nr2wdat <- read.csv("data/nr2wdat.csv")
nr2wdat$depth <- as.factor(nr2wdat$depth)
anova.model4 <- lm(temp ~ date + depth, data = nr2wdat)
Anova(anova.model4, type = 3)
```

| | Sum Sq | Df | F value | Pr(>F) |
|-------------|-----------|----|------------|---------|
| (Intercept) | 1511.9940 | 1 | 125.565158 | 0.0e+00 |
| date | 591.1467 | 3 | 16.364138 | 2.9e-06 |
| depth | 1082.8202 | 9 | 9.991552 | 1.5e-06 |
| Residuals | 325.1207 | 27 | NA | NA |

Si on suppose que c'est un modèle d'ANOVA mixte (date aléatoire, depth fixe), que concluez vous ?

Indice: faites un graphique d'interaction de la température en fonction de la profondeur et de la date, pour voir ce qui se passe.

```
interaction.plot(nr2wdat$depth, nr2wdat$date, nr2wdat$temp)
```

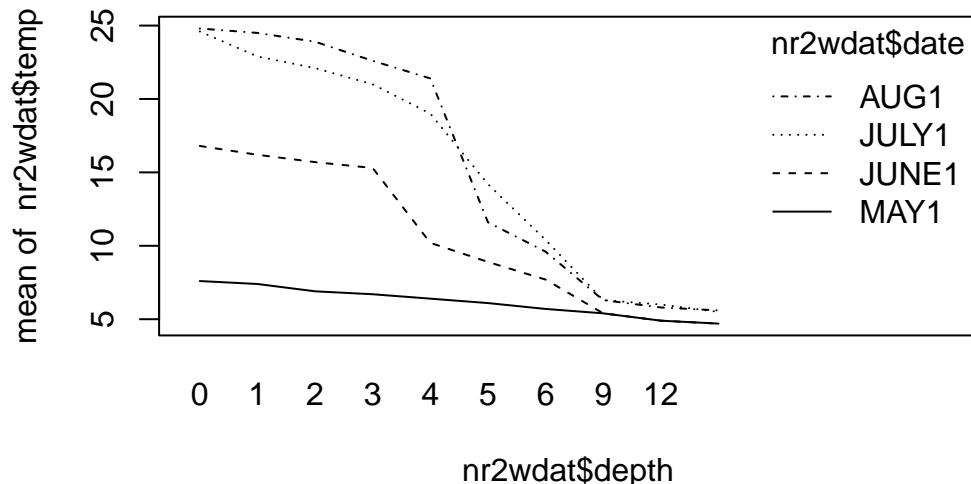


Figure 12.5.: Effet du mois et de la profondeur sur la température.

La température diminue significativement en fonction de la profondeur.

Pour tester l'effet du mois (facteur aléatoire), on doit présumer qu'il n'y a pas d'interaction entre la profondeur et le mois (*donc que l'effet de la profondeur sur la température est le même à chaque mois*).

C'est peu probable: si vous faites un graphique de la température en fonction de la profondeur pour chaque mois, vous observerez que le profil de température change au fur et à mesure du développement de la thermocline. Bref, comme le profil change au cours de l'été, ce modèle ne fait pas de très bonnes prédictions.

Jetez un coup d'oeil aux graphiques des résidus :

```
par(mfrow = c(2, 2))
plot(anova.model4)
```

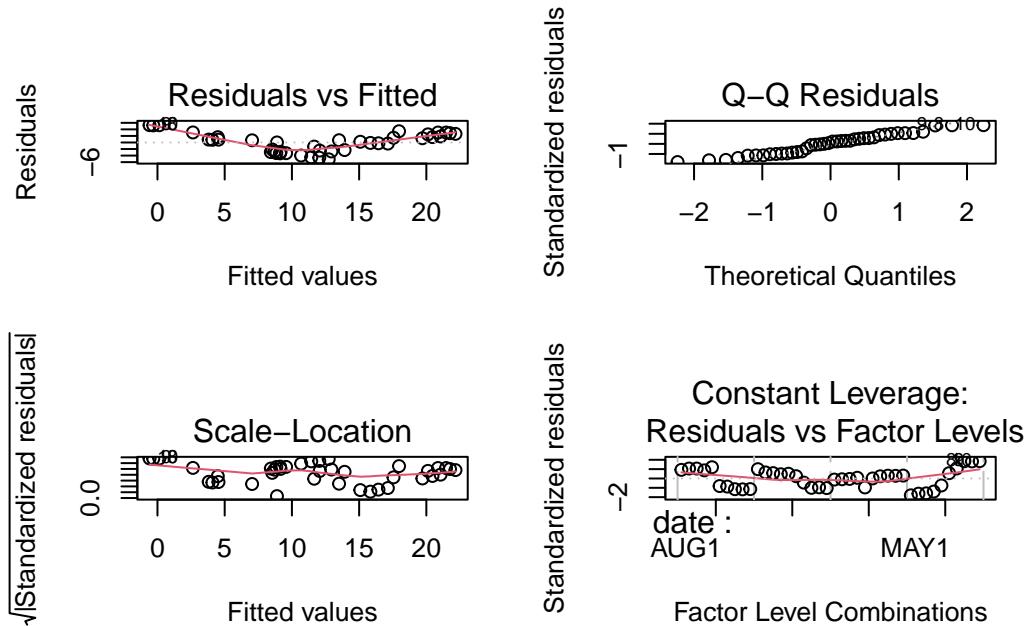


Figure 12.6.: Conditions d'applications du modèle `anova.model4`.

```
shapiro.test(residuals(anova.model4))
```

Shapiro-Wilk normality test

```
data: residuals(anova.model4)
W = 0.95968, p-value = 0.1634
```

Le test de normalité sur les résidus donne $p = 0.16$, donc l'hypothèse de normalité ne semble pas être sérieusement en doute.

Pour l'égalité des variances, on peut seulement comparer entre les mois en utilisant les profondeurs comme réplicats (ou l'inverse). En utilisant les profondeurs comme réplicats, on obtient:

```
leveneTest(temp ~ date, data = nr2wdat)
```

Warning in leveneTest.default(y = y, group = group, ...): group coerced to factor.

| | Df | F value | Pr(>F) |
|-------|----|---------|--------|
| group | 3 | 17.9789 | 3e-07 |
| | 36 | NA | NA |

Il y a donc un problème d'hétérosécédasticité, comme on peut très bien voir dans le graphique des résidus vs les valeurs estimées. Cette analyse n'est donc pas très satisfaisante: il y a des violations des conditions d'application et il semble y avoir une interaction entre depth et date qui pourrait invalider l'analyse.

12.4. Plans hiérarchiques

Un design expérimental fréquent implique la division de chaque groupe du facteur majeur en sous-groupes aléatoires.

Par exemple, une généticienne intéressée par l'effet du génotype sur la résistance à la dessiccation chez la drosophile effectue une expérience.

Pour chaque génotype (facteur principal) elle prépare trois chambres de croissance (sous-groupes) avec une température et humidité contrôlées. Dans chaque chambre de croissance, elle place cinq larves, puis mesure le nombre d'heures pendant lesquelles chaque larve survit.

Les données ont donc une structure hiérarchique. Il y a des observations répétées dans chaque chambre au sein de chaque génotype.

- Le fichier nestdat.csv contient les résultats d'une expérience semblable. Il contient trois variables : genotype, chamber et survival.

Effectuez une ANOVA hiérarchique avec survival comme variable dépendante et genotype et chamber/genotype comme variables indépendantes.

```
nestdat <- read.csv("data/nestdat.csv")
nestdat$chamber <- as.factor(nestdat$chamber)
nestdat$genotype <- as.factor(nestdat$genotype)
anova.nested <- lm(survival ~ genotype / chamber, data = nestdat)
```

Que concluez-vous de cette analyse ? Que devrait être la prochaine étape ?

Indice: si l'effet de `genotype / chamber` n'est pas significatif, vous pouvez augmenter la puissance des comparaisons entre génotypes en regroupant les chambres de chaque génotype, bien que (y compris Dr. Rundle) n'est pas d'accord avec cette mise en commun. Faites-le ! N'oubliez pas de vérifier les conditions d'applications de l'ANOVA!

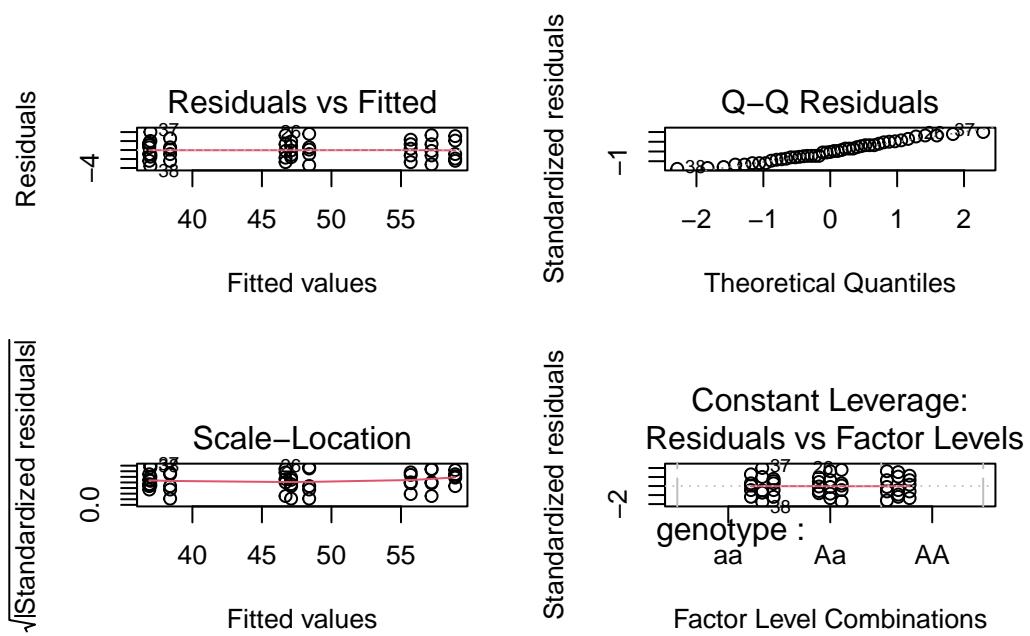
Solution

```
anova(anova.nested)
```

| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|------------------|----|------------|-------------|------------|-----------|
| genotype | 2 | 2952.22044 | 1476.110222 | 292.608079 | 0.0000000 |
| genotype:chamber | 6 | 40.65467 | 6.775778 | 1.343157 | 0.2638893 |
| Residuals | 36 | 181.60800 | 5.044667 | NA | NA |

Figure 12.7.: Conditions d'applications du modèle `anova.nested`

```
par(mfrow = c(2, 2))
plot(anova.nested)
```

Figure 12.8.: Conditions d'applications du modèle `anova.nested`

On conclue de cette analyse que la variation entre les chambres de croissance n'est pas significative, mais qu'on doit rejeter l'hypothèse nulle que tous les génotypes ont la même résistance à la dessiccation.

Comme l'effet hiérarchique `genotype / chamber` n'est pas significatif, on peut regrouper les observations pour augmenter le nombre de degrés de liberté :

```
anova.simple <- lm(survival ~ genotype, data = nestdat)
anova(anova.simple)
```

| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|-----------|----|-----------|-------------|----------|--------|
| genotype | 2 | 2952.2204 | 1476.110222 | 278.9341 | 0 |
| Residuals | 42 | 222.2627 | 5.291968 | NA | NA |

Donc on conclue qu'il y a une variation significative de résistance à la dessiccation entre les trois génotypes.

Le graphique de `survival` en fonction du génotype suggère que la résistance à la dessiccation varie entre chaque génotype. On peut combiner cela avec un test de Tukey.

```
par(mfrow = c(1, 1))
# Calculer et représenter les moyennes et l'IC de Tukey
means <- glht(anova.simple, linfct = mcp(
```

```

genotype =
  "Tukey"
))

cimeans <- cld(means)

# Utiliser des marges suffisantes
old.par <- par(mai = c(1, 1, 1.25, 1))
# Graph
plot(cimeans, las = 1) # Argument `las` pour placer les noms d'axes comme Dieu l'a prévu

```

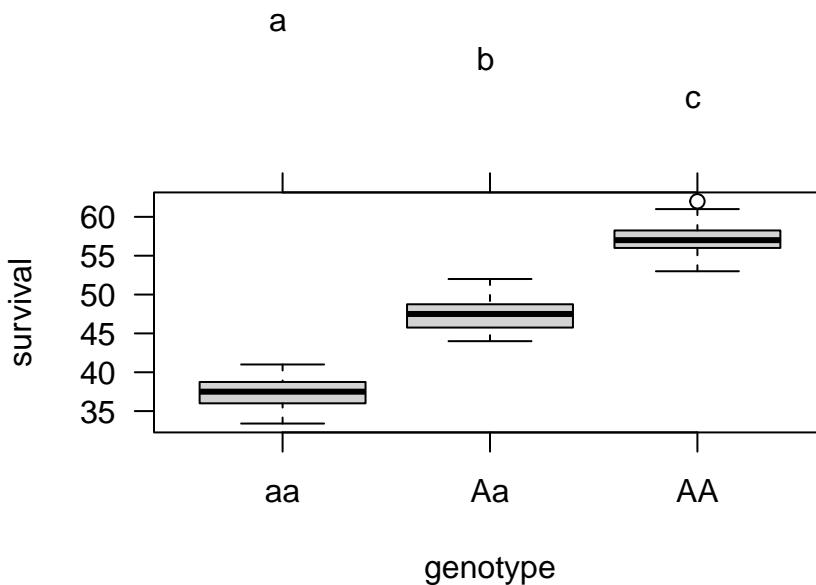


Figure 12.9.: Effet du genotype sur la résistance à la dessication avec un test de Tukey.

On conclue donc que la résistance à la dessiccation (R), telle que mesurée par la survie dans des conditions chaudes et sèches, varie significativement entre les trois génotypes avec $R(AA) > R(Aa) > R(aa)$.

Cependant, avant d'accepter cette conclusion, il faut éprouver les conditions d'application du test.

Voici les diagnostics des résidus pour l'ANOVA à un critère de classification (non hiérarchique):

Solution

```

par(mfrow = c(2, 2))
plot(anova.simple)

```

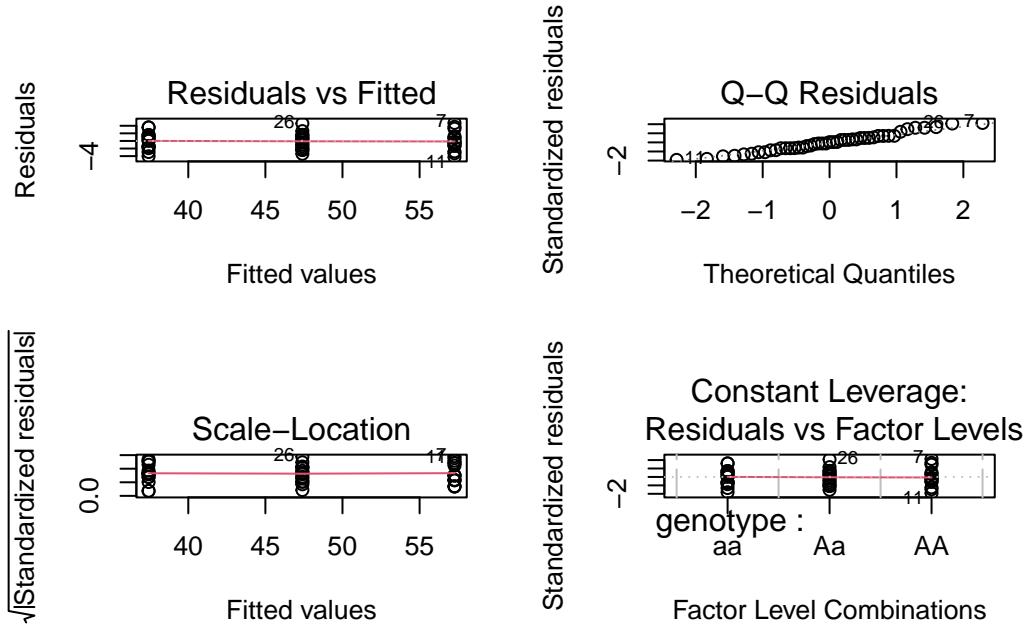


Figure 12.10.: Conditions d'applications du modèle anova.simple

Donc, toutes les conditions d'application semblent être remplies, et on peut donc accepter les conclusions. Notez que si l'on compare le carré moyen des résidus de l'ANOVA hiérarchique et de l'ANOVA à un critère de classification (5.045 vs 5.292), ils sont presque identiques. Cela n'est pas surprenant compte tenu de la faible variabilité associée aux chambres de croissance pour chaque génotype.

12.5. ANOVA non paramétrique avec 2 facteurs de classification

L'ANOVA non paramétrique à 2 critères de classification est une extension de celle à 1 critère de classification vue précédemment. Elle débute par une ANOVA faite sur les données transformées en rangs. Elle peut se faire sur des données avec ou sans réplicats.

À partir du fichier `stu2wdat.csv`, effectuez une ANOVA non paramétrique à 2 facteurs de classification pour examiner l'effet de `sex` et `location` sur `rank(rdwght)`.

```
aov.rank <- aov(
  rank(rdwght) ~ sex * location,
  contrasts = list(
    sex = contr.sum, location = contr.sum
  ),
```

```
data = Stu2wdat
)
```

L'extension de Schreirer-Ray-Hare au test de Kruskall-Wallis se fait en calculant une statistique H donnée par la somme des carrés de l'effet divisé par la somme des carrés totale (qui peut être calculé comme la variance des rangs). On calcule ensuite cette statistique H pour chaque terme, pour ensuite les comparer à une distribution théorique de χ^2 (chi-carré) en utilisant la commande `pchisq(H, df, lower.tail = FALSE)`, où H et df sont les statistiques H calculées et les degrés de libertés, respectivement.

Testez l'effet de `sex` et `location` sur `rdwght`. Que concluez-vous ? Comment ce résultat se compare-t-il à celui obtenu en faisant l'ANOVA paramétrique faite précédemment ?

```
Anova(aov.rank, type = 3)
```

| | Sum Sq | Df | F value | Pr(>F) |
|--------------|-------------|-----|-------------|-----------|
| (Intercept) | 1499862.414 | 1 | 577.8672627 | 0.0000000 |
| sex | 58393.631 | 1 | 22.4979086 | 0.0000042 |
| location | 1128.212 | 1 | 0.4346776 | 0.5105359 |
| sex:location | 1229.819 | 1 | 0.4738251 | 0.4921091 |
| Residuals | 472383.499 | 182 | NA | NA |

Pour calculer l'extension Schreirer-Ray-Hare au test de Kruskall-Wallis, on doit d'abord calculer le carré moyen total (i.e., la variance des données transformées en rang. Ici, on a 186 observations, donc des rangs; 1, 2, 3, ... 186).

La variance de cette série de 186 valeurs peut être calculée simplement par `var(1:186)`.

Donc on peut calculer la statistique H pour chaque terme:

```
Hsex <- 58394 / var(1:186)
Hlocation <- 1128 / var(1:186)
Hsexloc <- 1230 / var(1:186)
```

Et convertir ces statistiques en valeur de p :

```
# sex  
Hsex  
  
[1] 20.14628  
  
pchisq(Hsex, 1, lower.tail = FALSE)  
  
[1] 7.173954e-06  
  
# location  
Hlocation  
  
[1] 0.3891668  
  
pchisq(Hlocation, 1, lower.tail = FALSE)  
  
[1] 0.5327377  
  
# sex:location  
Hsexloc  
  
[1] 0.4243574  
  
pchisq(Hsexloc, 1, lower.tail = FALSE)  
  
[1] 0.5147707
```

Ces résultats sont semblables aux résultats de l'ANOVA non-paramétrique à 2 critères de classification. Malgré la puissance réduite, il y a encore un effet significatif du sexe, mais ni interaction ni effet du site.

Il y a cependant une différence importante. Rappelez-vous que dans l'ANOVA paramétrique d'origine, il y avait un effet significatif du sexe en considérant le problème comme une ANOVA Modèle I (fixe). Mais, si nous le considérons comme un modèle III (mixte), l'effet significatif du sexe pourrait en principe disparaître, car les degrés de liberté (ddl) associés au carré moyen (CM) de l'interaction sont plus faibles que le nombre de ddl du CM de l'erreur du modèle à effet fixes. Dans ce cas ci, cependant, le CM de l'interaction est environ la moitié du CM de l'erreur. Par conséquent, l'effet significatif de `sex` pourrait devenir encore plus significatif si le problème est analysé (comme il se doit) comme une ANOVA mixte. Encore une fois on peut voir l'importance de spécifier le modèle adéquat en ANOVA.

12.6. Comparaisons multiples

Les épreuves d'hypothèses subséquentes en ANOVA à plus d'un critère de classification dépendent des résultats initiaux de l'ANOVA. Si vous êtes intéressés à comparer des effets moyens d'un facteur pour tous les niveaux d'un autre facteur (par exemple l'effet du sexe sur la taille des esturgeons peu importe d'où ils viennent), alors vous pouvez procéder exactement tel que décrit dans la section sur les comparaisons multiples suivant l'ANOVA à un critère de classification. Pour comparer les moyennes des cellules entre elles, il faut spécifier l'interaction comme variable qui représente le groupe.

Le fichier `wmcdat2.csv` contient des mesures de consommation d'oxygène, `o2cons`, de deux espèces, `species`, d'un mollusque (une patelle) à trois concentrations différentes d'eau de mer (donc de salinité), `conc`. Ces données sont présentées à la p. 332 de Sokal et Rohlf 1995.

- Effectuez une ANOVA factorielle à 2 critères de classification sur ces données en utilisant `o2cons` comme variable dépendante et `species` et `conc` comme les facteurs (il va probablement falloir changer le type de données de variable `conc` à facteur). Que concluez-vous ?

Solution

```
wmcdat2 <- read.csv("data/wmcdat2.csv")
wmcdat2$species <- as.factor(wmcdat2$species)
wmcdat2$conc <- as.factor(wmcdat2$conc)
anova.model5 <- lm(o2cons ~ species * conc, data = wmcdat2)
Anova(anova.model5, type = 3)
```

| | Sum Sq | Df | F value | Pr(>F) |
|--------------|-------------|----|-------------|-----------|
| (Intercept) | 1185.601512 | 1 | 124.0164931 | 0.0000000 |
| species | 0.093025 | 1 | 0.0097306 | 0.9218903 |
| conc | 74.896658 | 2 | 3.9171766 | 0.0275505 |
| species:conc | 23.926200 | 2 | 1.2513662 | 0.2965616 |
| Residuals | 401.521300 | 42 | NA | NA |

Comme l'effectif dans chaque cellule est relativement petit, il faudrait idéalement refaire cette analyse avec une ANOVA non-paramétrique. Pour le moment, contentons nous de la version paramétrique.

Examinons les graphiques diagnostiques:

Solution

```
par(mfrow = c(2, 2))
plot(anova.model5)
```

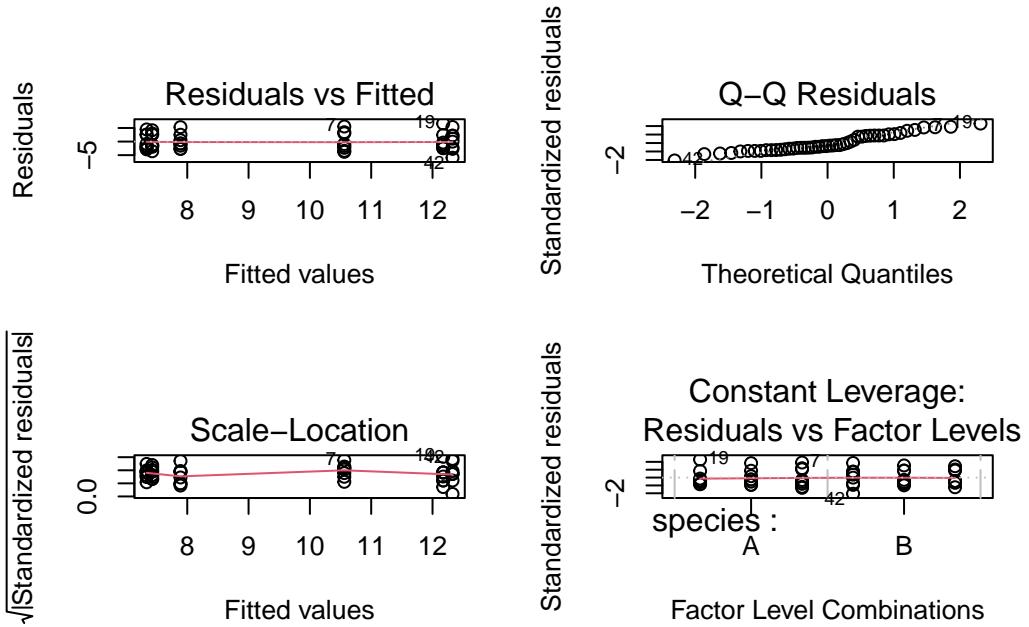


Figure 12.11.

Les variances semblent égales.

Le test de normalité donne:

Solution

```
shapiro.test(residuals(anova.model5))
```

```
Shapiro-Wilk normality test
```

```
data: residuals(anova.model5)
W = 0.93692, p-value = 0.01238
```

Il y a un écart à la normalité, mais à part ça tout semble aller. Comme l'ANOVA est relativement robuste à la non-normalité, on va regarder de l'autre côté (Si vous voulez être plus confiants, vous pouvez tourner une ANOVA

non paramétrique. Vous arriverez aux mêmes conclusions.)

On conclue donc qu'il n'y a pas de différence entre les espèces et que l'effet de la concentration ne dépend pas de l'espèce (il n'y a pas d'interaction). Par conséquent, les seules comparaisons justifiables sont entre les concentrations :

```
# Ajuster un modèle simplifié
anova.model6 <- aov(o2cons ~ conc, data = wmcdat2)
# Faire de comparaisons multiple de Tukey
TukeyHSD(anova.model6)
```

```
Tukey multiple comparisons of means
95% family-wise confidence level

Fit: aov(formula = o2cons ~ conc, data = wmcdat2)

$conc
      diff      lwr      upr      p adj
75-50 -4.63625 -7.321998 -1.9505018 0.0003793
100-50 -3.25500 -5.940748 -0.5692518 0.0141313
100-75  1.38125 -1.304498  4.0669982 0.4325855
```

```
par(mfrow = c(1, 1))
# Graphique de toutes les comparaisons pour `conc`
tuk <- glht(anova.model6, linfct = mcp(conc = "Tukey"))
# Extraire l'information
tuk.cld <- cld(tuk)
# Utiliser des marges suffisantes
old.par <- par(mai = c(1, 1, 1.25, 1))
# Graphe
plot(tuk.cld)
par(old.par)
```

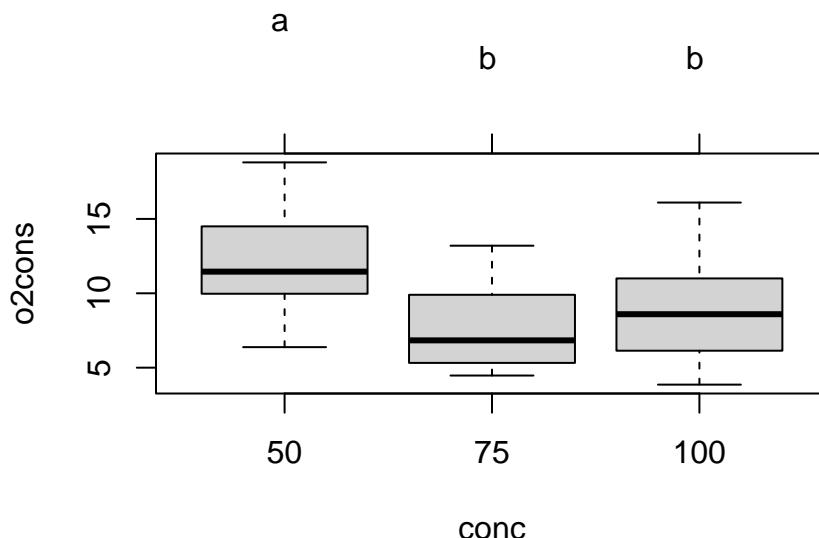


Figure 12.12.: Comparaison de Tukey des moyennes de consommation d'oxygène en fonction de la concentration

Il y a donc une différence de consommation d'oxygène significative lorsque la salinité est réduite de 50%, mais pas à 25% de réduction.

- Répétez les deux analyses précédentes sur les données du fichier wmc2dat2.csv. Comment les résultats se comparent-ils à ceux obtenus précédemment ?

Solution

```
wmc2dat2 <- read.csv("data/wmc2dat2.csv")
wmc2dat2$species <- as.factor(wmc2dat2$species)
wmc2dat2$conc <- as.factor(wmc2dat2$conc)
anova.model7 <- lm(o2cons ~ species * conc, data = wmc2dat2)
```

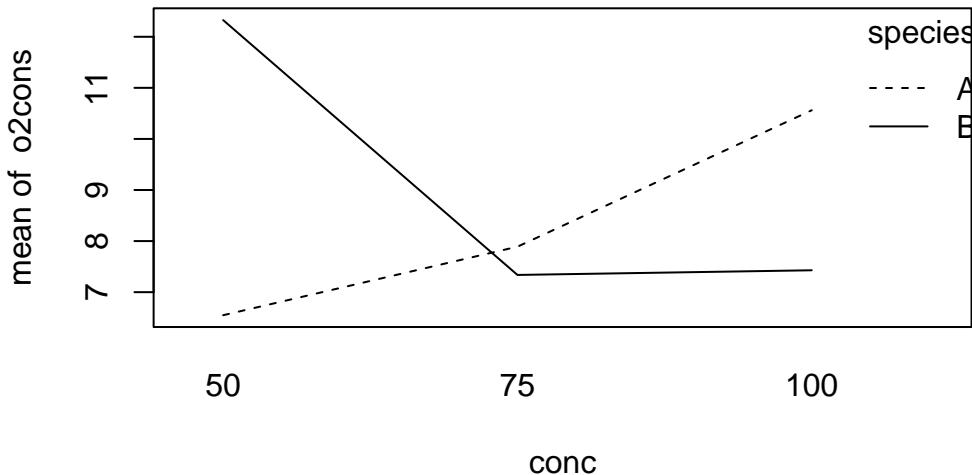
En utilisant wmc2dat2.csv, on obtient:

| | Sum Sq | Df | F value | Pr(>F) |
|--------------|-----------|----|-----------|-----------|
| (Intercept) | 343.08901 | 1 | 36.213216 | 0.0000004 |
| species | 133.51802 | 1 | 14.092894 | 0.0005286 |
| conc | 66.75916 | 2 | 3.523231 | 0.0385011 |
| species:conc | 168.15120 | 2 | 8.874222 | 0.0006101 |
| Residuals | 397.91380 | 42 | NA | NA |

Dans ce cas ci, il y a une interaction significative, et il n'est par conséquent pas approprié de comparer les moyennes

regroupées par espèce ou concentration. Ceci est clairement visualisé par un graphique d'interaction:

```
with(wmc2dat2, interaction.plot(conc, species, o2cons))
```



- Toujours en utilisant les données de `wmc2dat2.csv`, comparez les 6 moyennes avec l'ajustement Bonferroni. Pour ce faire, il sera utile de créer une nouvelle variable qui combine `species` et `conc`:

```
wmc2dat2$species.conc <- as.factor(paste0(wmc2dat2$species, wmc2dat2$conc))
```

ensuite on peut faire les comparaisons de Bonferroni:

```
with(wmc2dat2, pairwise.t.test(o2cons, species.conc, p.adj = "bonf"))
```

Pairwise comparisons using t tests with pooled SD

data: o2cons and species.conc

| | A100 | A50 | A75 | B100 | B50 |
|------|--------|--------|--------|------|-----|
| A50 | 0.1887 | - | - | - | - |
| A75 | 1.0000 | 1.0000 | - | - | - |
| B100 | 0.7223 | 1.0000 | 1.0000 | - | - |

```
B50  1.0000 0.0079 0.0929 0.0412 -
B75  0.6340 1.0000 1.0000 1.0000 0.0350

P value adjustment method: bonferroni
```

Ces comparaisons sont un peu plus difficiles à interpréter, mais l'analyse porte essentiellement sur les différences entre les concentrations d'eau de mer au sein de l'espèce A et sur les différences entre les concentrations au sein de l'espèce B. Nous voyons ici que la consommation d'oxygène à 50% d'eau de mer pour l'espèce B est significativement différente par rapport à 75% et 100% d'eau de mer. Là où il n'y a pas de différences significatives pour l'espèce A selon les concentrations.

Je trouve ces tableaux de résultats peu satisfaisants parce qu'ils indiquent seulement les valeur de p sans indices de la taille de l'effet. On peut obtenir à la fois le résultat des tests de comparaison multiple et un indice de la taille de l'effet à l'aide du code suivant :

```
# Ajuster un ANOVA à 2 facteurs pour toutes les combinaisons espèce.concentration
anova.modelx <- aov(o2cons ~ species.conc, data = wmc2dat2)
tuk2 <- glht(anova.modelx, linfct = mcp(species.conc = "Tukey"))

# Extraire l'information
tuk2.cld <- cld(tuk2)

# Utiliser des marges suffisantes
old.par <- par(mai = c(1, 1, 1.25, 1))

# Graphes
plot(tuk2.cld)
par(old.par)
```

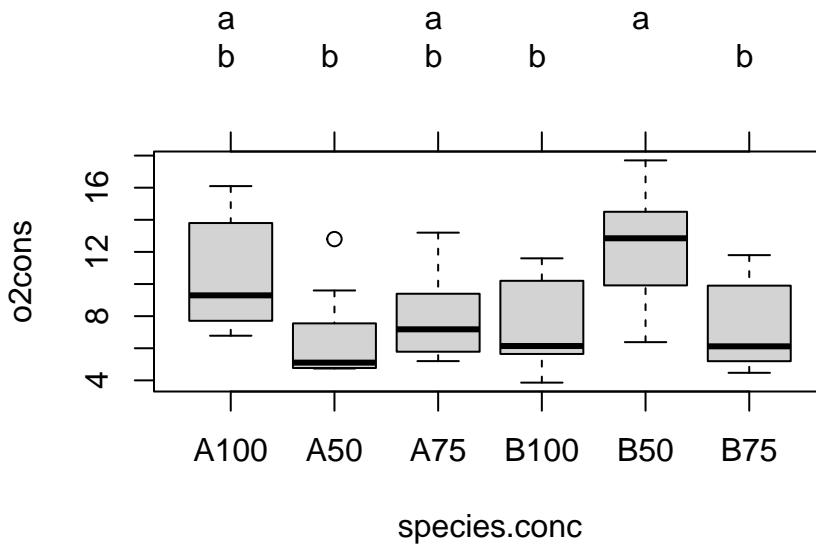


Figure 12.13.

Dans cette analyse on a utilisé le CM = 9.474 du modèle d'ANOVA pour comparer les moyennes. En faisant ça, on présume qu'il s'agit d'une situation d'ANOVA à effet fixes, ce qui n'est peut-être pas le cas (conc est certainement fixe, mais species peut être fixe ou aléatoire).

12.7. Test de permutation pour l'ANOVA à 2 facteurs de classification

Quand les données ne répondent pas aux conditions d'applications d'une ANOVA paramétrique à 2 ou plusieurs facteurs de classification, il est possible, comme alternative à l'ANOVA non-paramétrique, de faire des tests de **permutation** pour calculer les valeurs p.

```
#####
# Test de permutation pour une ANOVA à 2 facteurs
# Ter Braak crée des résidus à partir des moyennes des cellules,
# puis effectue des permutations entre toutes les cellules
# Cela peut être fait en prenant les résidus du modèle complet.
# modifié depuis le code écrit par David C. Howell
# http://www.uvm.edu/~dhowell/StatPages/More_Stuff/Permutation%20Anova/PermTestsAnova.html
nreps <- 500
dependent <- Stu2wdat$rdwght
factor1 <- as.factor(Stu2wdat$sex)
factor2 <- as.factor(Stu2wdat$location)
```

```
my.dataframe <- data.frame(dependent, factor1, factor2)
my.dataframe.noNA <- my.dataframe[complete.cases(my.dataframe), ]
mod <- lm(dependent ~ factor1 + factor2 + factor1:factor2,
           data = my.dataframe.noNA
)
res <- mod$residuals
TBint <- numeric(nreps)
TB1 <- numeric(nreps)
TB2 <- numeric(nreps)
ANOVA <- summary(aov(mod))
cat(
  " L'ANOVA standard pour ces données suit ",
  "\n"
)
F1 <- ANOVA[[1]]$"F value"[1]
F2 <- ANOVA[[1]]$"F value"[2]
Finteract <- ANOVA[[1]]$"F value"[3]
print(ANOVA)
cat("\n")
cat("\n")
TBint[1] <- Finteract
for (i in 2:nreps) {
  newdat <- sample(res, length(res), replace = FALSE)
  modb <- summary(aov(newdat ~ factor1 + factor2 +
    factor1:factor2,
    data = my.dataframe.noNA
  ))
  TBint[i] <- modb[[1]]$"F value"[3]
  TB1[i] <- modb[[1]]$"F value"[1]
  TB2[i] <- modb[[1]]$"F value"[2]
}
probInt <- length(TBint[TBint >= Finteract]) / nreps
prob1 <- length(TB1[TB1 >= F1]) / nreps
```

```

prob2 <- length(TB2[TB1 >= F2]) / nreps
cat("\n")
cat("\n")
print("Rééchantillonnage comme dans ter Braak avec rééchantillonne non restreint des cellules rési-
cat(
  "La probabilité pour l'effet d'Interaction est ",
  probInt, "\n"
)
cat(
  "La probabilité pour l'effet du Facteur 1 est ",
  prob1, "\n"
)
cat(
  "La probabilité pour l'effet du Facteur 2 est ",
  prob2, "\n"
)

```

Avec le paquet `lmPerm` , vous pouvez effectuer le test de permutation beaucoup plus rapidement et facilement :

```

#####
## Test de permutation version `lmPerm`
library(lmPerm)

# Par soucis de généralisation, assignez votre jeu de donnée à l'objet "mesdonnees"
# et la formule du modèle à "maformule"
mesdonnees <- Stu2wdat

maformule <- as.formula("rdwght ~ sex+location+sex:location")

# Ajustez le modèle voulu aux données voulues
monmodele <- lm(maformule, data = mesdonnees)

# Calculez la valeur p permutée
anova(lmp(maformule, data = mesdonnees, perm = "Prob", center = FALSE, Ca = 0.001))

```

12.8. Bootstrap pour l'ANOVA à 2 facteurs de classification

Dans la plupart des cas, les tests de permutation seront plus appropriés que le bootstrap pour les ANOVA. J'ai quand même un bout de code qui pourra servir si vous en avez besoin:

```
#####
#####
# Bootstrap pour une ANOVA à 2 facteurs
# Vous aurez peut être envie de modifier "bootfunction.mod1" pour renvoyer d'autres valeurs
# Ici, ça renvoie les coefficients standard du modèle ajusté
# Requiert le paquet `boot`
#
nreps <- 5000
dependent <- Stu2wdat$rdwght
factor1 <- as.factor(Stu2wdat$sex)
factor2 <- as.factor(Stu2wdat$location)
my.dataframe <- data.frame(dependent, factor1, factor2)
my.dataframe.noNA <- my.dataframe[complete.cases(my.dataframe), ]
library(boot)
# Ajustez le modèle aux données observées
mod1 <- aov(dependent ~ factor1 + factor2 + factor1:factor2,
             data = my.dataframe.noNA
)
# Bootstrap 1000 fois en utilisant la méthode de bootstrapping résiduel pour
# garder le même nombre inégal d'observations pour chaque niveaux de la variable indépendante.
fit <- fitted(mod1)
e <- residuals(mod1)
X <- model.matrix(mod1)
bootfunction.mod1 <- function(data, indices) {
  y <- fit + e[indices]
  bootmod <- lm(y ~ X)
  coefficients(bootmod)
```

```
}

bootresults <- boot(my.dataframe.noNA, bootfunction.mod1,
  R = 1000
)

bootresults

## Calcule l'IC 90% et trace les estimés du bootstrap séparément pour chaque paramètres du modèle

boot.ci(bootresults, conf = 0.9, index = 1)
plot(bootresults, index = 1)

boot.ci(bootresults, conf = 0.9, index = 3)
plot(bootresults, index = 3)

boot.ci(bootresults, conf = 0.9, index = 4)
plot(bootresults, index = 4)

boot.ci(bootresults, conf = 0.9, index = 5)
plot(bootresults, index = 5)
```

Chapitre 13

Régression multiple

Objectifs de ce chapitre :

- Utiliser R pour ajuster une régression multiple et comparer des modèles selon l'approche inférentielle et celle de la théorie de l'information
- Utiliser R pour éprouver des hypothèses sur l'effet des variables indépendantes sur la variable dépendante.
- Utiliser R pour évaluer la multicolinéarité entre les variables indépendantes et en évaluer ses effets
- Utiliser R pour effectuer une régression curvilinéaire (polynomiale).

Avertissement

Je ne supporte pas absolument pas l'approche stepwise (ni par AIC ni par P). Je déteste l'approche par la fonction `dredge()` qui selon moi va à l'encontre de la philosophie des AIC et de la parcimonie. L'approche stepwise ne devrait plus être utilisé. Ce chapitre est en grand besoin d'une réécriture. Je soutiens de développer un modèle complet basé sur des hypothèses biologiques et de reporter ce modèle avec tous les effets significatifs ou non.

13.1. Paquets et données requises pour le labo

Pour ce laboratoire, vous aurez besoin de :

- Paquets R:

- `ggplot2` 
- `car` 
- `lmtest` 

- simpleboot 
- boot 
- MuMIn 
- Jeu de données
 - “Mregdat.csv”

13.2. Conseils généraux

Les variables qui intéressent les biologistes sont généralement influencées par plusieurs facteurs, et une description exacte ou une prédiction de la variable dépendante requiert que plus d'une variable soit incluse dans le modèle. La régression multiple permet de quantifier l'effet de plusieurs variables continues sur la variable dépendante.

Il est important de réaliser que la maîtrise de la régression multiple ne s'acquiert pas instantanément. Les débutants doivent garder à l'esprit plusieurs points importants :

1. Un modèle de régression multiple peut être hautement significatif même si aucun des termes pris isolément ne l'est (ceci est causé par la **multicolinéarité**),
2. Un modèle peut ne pas être significatif alors que l'un ou plusieurs des termes le sont (ceci est un signe d'un modèle trop complexe (“overfitting”)) et,
3. À moins que les variables indépendantes soient parfaitement orthogonales (c'est-à-dire qu'il n'y ait aucune corrélation entre elles et donc pas de multicolinéarité) les diverses approches de sélection des variables indépendantes peuvent mener à des modèles différents.

13.3. Premières régressions multiples

Le fichier Mregdat .csv contient des données de richesse spécifique de quatre groupes d'organismes dans 30 marais de la région Ottawa-Cornwall-Kingston. Les variables sont:

- la richesse spécifique:
 - des oiseaux (bird, et son logarithme base 10 logbird)
 - des mammifères (mammal, logmam)
 - des amphibiens et reptiles (herptile, logherp)
 - des vertébrés (totsp, logtot)

- les coordonnées des sites (lat, long)
- la superficie du marais (logarea)
- le pourcentage du marais inondé toute l'année (swamp)
- le pourcentage des terres couvertes par des forêts dans un rayon de 1km du marais (cpfor2)
- la densité des routes pavées (en m/ha) dans un rayon de 1km du marais (thtden).

Nous allons nous concentrer sur les amphibiens et les reptiles (herptile) pour cet exemple, il est donc avisé d'examiner la distribution de cette variable et les corrélations avec les variables indépendantes potentielles:

```
mydata <- read.csv("data/Mregdat.csv")

scatterplotMatrix(
  ~ logherp + logarea + cpfor2 + thtden + swamp,
  regLine = TRUE, smooth = TRUE, diagonal = TRUE,
  data = mydata
)
```

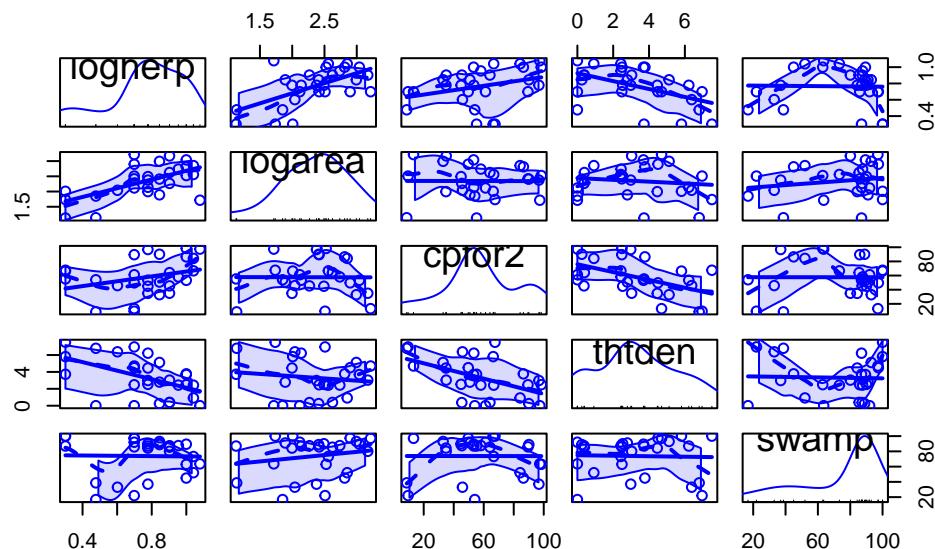


Figure 13.1.: Matrice de relation et densité pour la richesse spécifique des amphibiens et reptiles

- En utilisant les données de ce fichier, faites la régression simple de logherp en fonction de logarea . Que concluez-vous à partir de cette analyse?

```
model.loga <- lm(logherp ~ logarea, data = mydata)
summary(model.loga)
```

Call:

```
lm(formula = logherp ~ logarea, data = mydata)
```

Residuals:

| Min | 1Q | Median | 3Q | Max |
|----------|----------|---------|---------|---------|
| -0.38082 | -0.09265 | 0.00763 | 0.10409 | 0.46977 |

Coefficients:

| | Estimate | Std. Error | t value | Pr(> t) |
|----------------|----------|------------|---------|--------------|
| (Intercept) | 0.18503 | 0.15725 | 1.177 | 0.249996 |
| logarea | 0.24736 | 0.06536 | 3.784 | 0.000818 *** |
| --- | | | | |
| Signif. codes: | 0 *** | 0.001 ** | 0.01 * | 0.05 . |
| | '***' | '**' | '*' | '.' |
| | 1 | 1 | 1 | 1 |

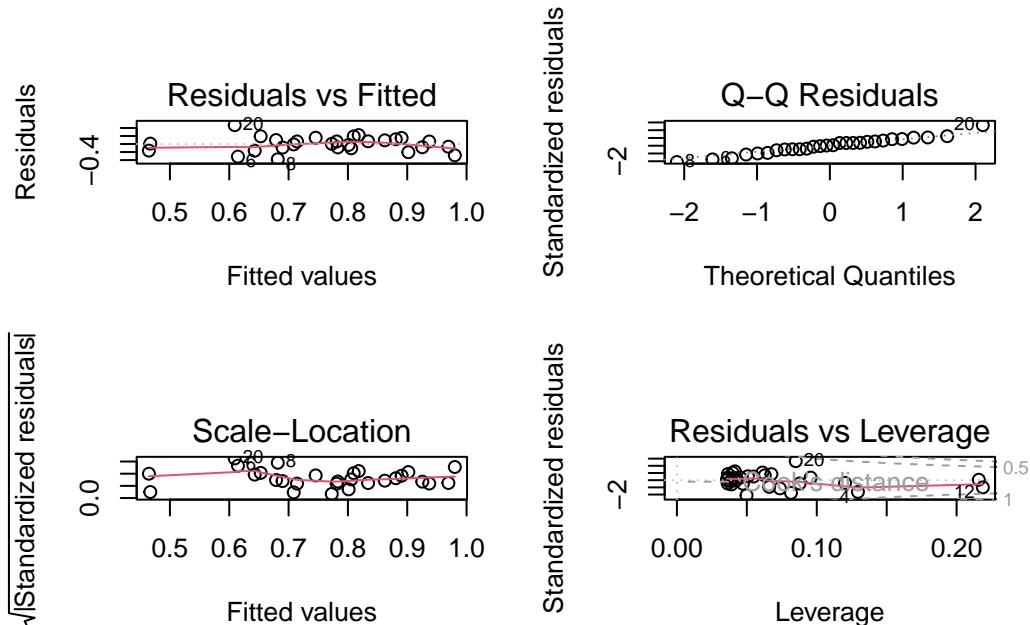
Residual standard error: 0.1856 on 26 degrees of freedom

(2 observations deleted due to missingness)

Multiple R-squared: 0.3552, Adjusted R-squared: 0.3304

F-statistic: 14.32 on 1 and 26 DF, p-value: 0.0008185

```
par(mfrow = c(2, 2))
plot(model.loga)
```

Figure 13.2.: Conditions d'applications de la régression de *logherp* sur *logarea*

Il semble donc y avoir une relation positive entre la richesse spécifique des reptiles et des amphibiens et la surface des marais. La régression n'explique cependant qu'environ le tiers de la variabilité ($R^2 = 0.355$). L'analyse des résidus indique qu'il n'y a pas de problème avec la normalité, l'homoscédasticité, ni l'indépendance.

- Faites ensuite la régression de *logherp* en fonction de *cpfor2*. Que concluez-vous?

Solution

```
model.logcp <- lm(logherp ~ cpfor2, data = mydata)
summary(model.logcp)
```

Call:

```
lm(formula = logherp ~ cpfor2, data = mydata)
```

Residuals:

| Min | 1Q | Median | 3Q | Max |
|----------|----------|---------|---------|---------|
| -0.49095 | -0.10266 | 0.05881 | 0.16027 | 0.25159 |

Coefficients:

| Estimate | Std. Error | t value | Pr(> t) |
|----------|------------|---------|----------|
|----------|------------|---------|----------|

```
(Intercept) 0.609197 0.104233 5.845 3.68e-06 ***
cpfor2      0.002706 0.001658 1.632    0.115
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.2202 on 26 degrees of freedom
(2 observations deleted due to missingness)

Multiple R-squared:  0.09289,   Adjusted R-squared:  0.058
F-statistic: 2.662 on 1 and 26 DF,  p-value: 0.1148
```

Ici, on doit accepter l'hypothèse nulle et conclure qu'il n'y a pas de relation entre la richesse spécifique dans les marais (`logherp`) et la proportion de forêts sur les terres adjacentes (`cpfor2`). Qu'est-ce qui arrive quand on fait une régression avec les 2 variables indépendantes?

- Refaites la régression de `logherp` en fonction de `logarea` et `cpfor2` à la fois. Que concluez-vous?

Solution

```
model.mcp <- lm(logherp ~ logarea + cpfor2, data = mydata)
summary(model.mcp)
```

Call:

```
lm(formula = logherp ~ logarea + cpfor2, data = mydata)
```

Residuals:

| Min | 1Q | Median | 3Q | Max |
|----------|----------|---------|---------|---------|
| -0.40438 | -0.11512 | 0.01774 | 0.08187 | 0.36179 |

Coefficients:

| | Estimate | Std. Error | t value | Pr(> t) |
|-------------|----------|------------|---------|--------------|
| (Intercept) | 0.027058 | 0.166749 | 0.162 | 0.872398 |
| logarea | 0.247789 | 0.061603 | 4.022 | 0.000468 *** |
| cpfor2 | 0.002724 | 0.001318 | 2.067 | 0.049232 * |
| --- | | | | |

```

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.175 on 25 degrees of freedom
(2 observations deleted due to missingness)

Multiple R-squared: 0.4493, Adjusted R-squared: 0.4052
F-statistic: 10.2 on 2 and 25 DF, p-value: 0.0005774

```

On voit donc qu'on peut rejeter les 2 hypothèses nulles que la pente de la régression de logherp sur logarea est zéro et que la pente de la régression de logherp sur cpfor2 est zéro. Pourquoi cpfor2 devient-il un facteur significatif dans la régression multiple alors qu'il n'est pas significatif dans la régression simple? Parce qu'il est parfois nécessaire de contrôler pour l'effet d'une variable pour pouvoir détecter les effets plus subtils d'autres variables. Ici, il y a une relation significative entre logherp et logarea qui masque l'effet de cpfor2 sur logherp . Lorsque le modèle tient compte des deux variables explicatives, il devient possible de détecter l'effet de cpfor2 .

- Ajustez un autre modèle, cette fois en remplaçant cpfor2 par thtden ($\text{logherp} \sim \text{logarea} + \text{thtden}$). Que concluez-vous?

Solution

```

model.mden <- lm(logherp ~ logarea + thtden, data = mydata)
summary(model.mden)

```

Call:

```
lm(formula = logherp ~ logarea + thtden, data = mydata)
```

Residuals:

| Min | 1Q | Median | 3Q | Max |
|----------|----------|---------|---------|---------|
| -0.31583 | -0.12326 | 0.02095 | 0.13201 | 0.31674 |

Coefficients:

| | Estimate | Std. Error | t value | Pr(> t) |
|-------------|----------|------------|---------|--------------|
| (Intercept) | 0.37634 | 0.14926 | 2.521 | 0.018437 * |
| logarea | 0.22504 | 0.05701 | 3.947 | 0.000567 *** |
| thtden | -0.04196 | 0.01345 | -3.118 | 0.004535 ** |

```
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Residual standard error: 0.1606 on 25 degrees of freedom

(2 observations deleted due to missingness)

Multiple R-squared: 0.5358, Adjusted R-squared: 0.4986

F-statistic: 14.43 on 2 and 25 DF, p-value: 6.829e-05

On rejette donc l'hypothèse nulle que la richesse spécifique n'est pas influencée par la taille des marais (`logarea`) ni par la densité des routes (`thtden`). Notez qu'ici il y a une relation négative significative entre la richesse spécifique des amphibiens et reptiles et la densité des routes sur les terres adjacentes, tandis que la relation est positive pour la taille des marais et pour la densité des forêts (`cpfor2` ; résultat de la dernière régression).

Le R^2 de ce modèle est plus élevé que pour le précédent, reflétant une corrélation plus forte entre `logherp` et `thtden` qu'entre `logherp` et `cpfor2`.

La richesse spécifique des reptiles et amphibiens semble donc reliée à la surface de marais (`logarea`), la densité des routes (`thtden`), et possiblement au couvert forestier sur les terres adjacentes aux marais (`cpfor2`). Cependant, les trois variables ne sont peut-être pas nécessaires dans un modèle prédictif. Si deux des trois variables (disons `cpfor2` et `thtden`) sont parfaitement corrélées, alors l'effet de `thtden` ne serait rien de plus que celui de `cpfor2` (et vice-versa) et un modèle incluant l'une des deux variables ferait des prédictions identiques à un modèle incluant ces deux variables (en plus de `logarea`).

- Estimez un modèle de régression avec `logherp` comme variable dépendante et `logarea`, `cpfor2` et `thtden` comme variables indépendantes. Que concluez-vous?

Solution

```
model.mtri <- lm(logherp ~ logarea + cpfor2 + thtden, data = mydata)
summary(model.mtri)
```

Call:

```
lm(formula = logherp ~ logarea + cpfor2 + thtden, data = mydata)
```

Residuals:

```

      Min       1Q    Median       3Q      Max
-0.30729 -0.13779  0.02627  0.11441  0.29582

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 0.284765  0.191420   1.488 0.149867
logarea     0.228490  0.057647   3.964 0.000578 ***
cpfor2      0.001095  0.001414   0.774 0.446516
thtden     -0.035794  0.015726  -2.276 0.032055 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1619 on 24 degrees of freedom
(2 observations deleted due to missingness)
Multiple R-squared:  0.5471,    Adjusted R-squared:  0.4904
F-statistic: 9.662 on 3 and 24 DF,  p-value: 0.0002291

```

Plusieurs choses sont à noter ici: 1. Tel que prévu, le coefficient de régression pour cpfor2 n'est plus significativement différent de 0. Une fois que la variabilité attribuable à logarea et thtden est enlevée, il ne reste qu'une fraction nonsignificative de la variabilité attribuable à cpfor2 . 2. Le R^2 pour ce modèle(0.547) n'est que légèrement supérieur au R^2 du modèle avec seulement logarea et thtden (.536), ce qui confirme que cpfor2 n'explique pas grand-chose de plus.

Notez aussi que même si le coefficient de régression pour thtden n'a pas beaucoup changé par rapport à ce qui avait été estimé lorsque seul thtden et logarea étaient dans le modèle (0-.036 vs -0.042), l'erreur type pour l'estimé du coefficient est plus grand, et ce modèle plus complexe mène à un estimé moins précis. Si la corrélation entre thtden et cpfor2 était plus forte, la décroissance de la précision serait encore plus grande.

On peut comparer les deux derniers modèles (i.e., le modèle incluant les 3 variables et celui avec seulement logarea and thtden) pour décider lequel privilégier.

```
anova(model.mtri, model.mdan)
```

| Res.Df | RSS | Df | Sum of Sq | F | Pr(>F) |
|--------|-----------|----|------------|-----------|-----------|
| 24 | 0.6293717 | NA | NA | NA | NA |
| 25 | 0.6450798 | -1 | -0.0157081 | 0.5990024 | 0.4465163 |

Cette comparaison révèle que le modèle à 3 variables ne fait pas de prédictions significativement meilleures que le modèle avec seulement Logarea et thtden . Ce résultat n'est pas surprenant puisque le test de signification pour cpfor2 dans le modèle complet indique qu'il faut accepter l'hypothèse nulle.

À la suite de cette analyse, on doit conclure que:

1. Le meilleur modèle est celui incluant thtden et logarea .
2. Il y a une relation négative entre la richesse spécifique des amphibiens et reptiles et la densité des routes sur les terres adjacentes.
3. Il y a une relation positive entre la richesse spécifique et la taille des marais.

Notez que le “meilleur” modèle n'est pas nécessairement le modèle parfait, seulement le meilleur n'utilisant que ces trois variables indépendantes. Il est évident qu'il y a d'autres facteurs qui contrôlent la richesse spécifique dans les marais puisque, même le “meilleur” modèle n'explique que la moitié de la variabilité.

13.4. Régression multiple pas-à-pas (stepwise)

⚠ Avertissement

Section encore présente à titre d'information mais à ne jamais utiliser à cause de l'ensemble des problèmes d'inférences et de bias dans l'erreur de type 1 qu'elle génère.

Quand le nombre de variables prédictives est restreint, comme dans l'exemple précédent, il est aisément de comparer manuellement les modèles pour sélectionner le plus adéquat. Cependant, lorsque le nombre de variables indépendantes augmente, cette approche n'est rapidement plus utilisable et il est alors utile d'utiliser une méthode automatisée.

La sélection pas à pas avec R utilise le Critère Informatif de Akaike (Akaike Information Criterion, $AIC = 2 * \ln(RSS) + 2K$ où K le nombre de variables indépendantes, n est le nombre d'observations, et RSS est la somme des carrés des résidus) comme mesure de la qualité d'ajustement des modèles. Cette mesure favorise la précision des prédictions et pénalise la complexité. **Lorsque l'on compare des modèles par AIC, le modèle avec le plus petit AIC est le modèle à préférer.**

- Utiliser la fonction step pour activer la sélection pas à pas des variables indépendantes sur le modèles de régression incluant logarea, cpfor2 et thtden:

```
# Stepwise Regression
step.mtri <- step(model.mtri, direction = "both")
```

Start: AIC=-98.27

logherp ~ logarea + cpfor2 + thtden

| | Df | Sum of Sq | RSS | AIC |
|-----------|----|-----------|---------|---------|
| - cpfor2 | 1 | 0.01571 | 0.64508 | -99.576 |
| <none> | | | 0.62937 | -98.267 |
| - thtden | 1 | 0.13585 | 0.76522 | -94.794 |
| - logarea | 1 | 0.41198 | 1.04135 | -86.167 |

Step: AIC=-99.58

logherp ~ logarea + thtden

| | Df | Sum of Sq | RSS | AIC |
|-----------|----|-----------|---------|---------|
| <none> | | 0.64508 | -99.576 | |
| + cpfor2 | 1 | 0.01571 | 0.62937 | -98.267 |
| - thtden | 1 | 0.25092 | 0.89600 | -92.376 |
| - logarea | 1 | 0.40204 | 1.04712 | -88.013 |

```
step.mtri$anova # display results
```

| Step | Df | Deviance | Resid. Df | Resid. Dev | AIC |
|----------|----|-----------|-----------|------------|-----------|
| | NA | NA | 24 | 0.6293717 | -98.26666 |
| - cpfor2 | 1 | 0.0157081 | 25 | 0.6450798 | -99.57640 |

R nous donne:

1. L'ajustement (mesuré par AIC) du modèle complet en premier lieu.

2. L'AIC des modèles dans lesquels une variable a été enlevée du modèle complet. Notez que c'est seulement en enlevant cpfor2 du modèle qu'on peut réduire l'AIC
3. La valeur de AIC pour les modèles auxquels on enlève ou on ajoute une variable au modèle sélectionné à la première étape.(i.e. logherp ~ logarea + thtden). Notez qu'aucun des modèles n'a un AIC inférieur à ce modèle.

Au lieu de débuter par le modèle complet (saturé) et enlever des termes, on peut commencer par le modèle nul et ajouter des termes:

```
# Forward selection approach

model.null <- lm(logherp ~ 1, data = mydata)
step.f <- step(
  model.null,
  scope = ~ . + logarea + cpfor2 + thtden, direction = "forward"
)
```

Start: AIC=-82.09

logherp ~ 1

| | Df | Sum of Sq | RSS | AIC |
|-----------|----|-----------|--------|---------|
| + logarea | 1 | 0.49352 | 0.8960 | -92.376 |
| + thtden | 1 | 0.34241 | 1.0471 | -88.013 |
| + cpfor2 | 1 | 0.12907 | 1.2605 | -82.820 |
| <none> | | 1.3895 | | -82.091 |

Step: AIC=-92.38

logherp ~ logarea

| | Df | Sum of Sq | RSS | AIC |
|----------|----|-----------|---------|---------|
| + thtden | 1 | 0.25093 | 0.64508 | -99.576 |
| + cpfor2 | 1 | 0.13078 | 0.76522 | -94.794 |
| <none> | | 0.89600 | | -92.376 |

Step: AIC=-99.58

logherp ~ logarea + thtden

```
Df Sum of Sq      RSS      AIC
<none>             0.64508 -99.576
+ cpfor2  1  0.015708 0.62937 -98.267
```

```
step.f$anova # display results
```

| Step | Df | Deviance | Resid. Df | Resid. Dev | AIC |
|-----------|----|-----------|-----------|------------|-----------|
| | NA | NA | 27 | 1.3895281 | -82.09073 |
| + logarea | -1 | 0.4935233 | 26 | 0.8960048 | -92.37639 |
| + thtden | -1 | 0.2509250 | 25 | 0.6450798 | -99.57640 |

Le résultat final est le même, mais la trajectoire est différente. Dans ce cas, R débute avec le modèle le plus simple et ajoute une variable indépendante à chaque étape, sélectionnant la variable minimisant AIC à cette étape. Le modèle de départ a donc seulement une ordonnée à l'origine. Puis, `logarea` est ajouté, suivi de `thtden`. `cpfor2` n'est pas ajouté au modèle, car son addition fait augmenter l'AIC.

Il est recommandé de comparer le résultat final de plusieurs approches. Si le modèle retenu diffère selon l'approche utilisée, c'est un signe que le “meilleur” modèle est possiblement difficile à identifier et que vous devriez être circonspects dans vos inférences. Dans notre exemple, pas de problème: toutes les méthodes convergent sur le même modèle final.

Pour conclure cette section, quelques conseils concernant les méthodes automatisées de sélection des variables indépendantes:

1. Les différentes méthodes de sélection des variables indépendantes peuvent mener à des modèles différents.
Il est souvent utile d'essayer plus d'une méthode et de comparer les résultats. Si les résultats diffèrent, c'est presque toujours à cause de multicolinéarité entre les variables indépendantes.
2. Attention à la régression pas-à-pas. Les auteurs de SYSTAT en disent:

Stepwise regression is probably the most abused computerized statistical technique ever devised. If you think you need automated stepwise regression to solve a particular problem, you probably don't. Professional statisticians rarely use automated stepwise regression because it does not necessarily find the “best” fitting model, the “real” model, or alternative “plausible” models. Furthermore, the order in which variables enter or leave a stepwise program is usually of no theoretical significance. You are always better off thinking about why a model could generate your data and then testing that model.

En bref, on abuse trop souvent de cette technique.

3. Il faut toujours garder à l'esprit que l'existence d'une régression significative n'est pas suffisante pour prouver une relation causale.

13.5. Déetecter la multicolinéarité

La multicolinéarité est la présence de corrélations entre les variables indépendantes. Lorsqu'elle est extrême (multicolinéarité parfaite) elle empêche l'estimation des modèles statistiques. Lorsqu'elle est grande ou modérée, elle réduit la puissance de détection de l'effet des variables indépendantes individuellement, mais elle n'empêche pas le modèle de faire des prédictions.

Un des indices les plus utilisés pour quantifier la multicolinéarité est le facteur d'inflation de la variance (VIF, variance inflation factor). Le fichier d'aide du package HH explique ainsi son calcul:

A simple diagnostic of collinearity is the variance inflation factor, VIF one for each regression coefficient (other than the intercept). Since the condition of collinearity involves the predictors but not the response, this measure is a function of the X's but not of Y. The VIF for predictor i is $1/(1 - R_i^2)$, where R_i^2 is the R^2 from a regression of predictor i against the remaining predictors. If R_i^2 is close to 1, this means that predictor i is well explained by a linear function of the remaining predictors, and, therefore, the presence of predictor i in the model is redundant. Values of VIF exceeding 5 are considered evidence of collinearity: The information carried by a predictor having such a VIF is contained in a subset of the remaining predictors. If, however, all of a model's regression coefficients differ significantly from 0 (p-value < .05), a somewhat larger VIF may be tolerable.

Bref, les VIF indiquent de combien l'incertitude de chaque coefficient de régression est augmentée par la multicolinéarité.

Attrappe. Il y a plusieurs fonctions vif() (j'en connais au moins trois dans les extensions car, HH et DAAG), et je ne sais pas en quoi elles diffèrent.

On peut calculer les VIF avec la fonction vif() de l'extension car:

```
library(car)
vif(model.mtri)
```

```
logarea    cpfor2    thtden
1.022127  1.344455  1.365970
```

Ici, il n'y a pas d'évidence de multicolinéarité car toutes les valeurs de VIF sont près de 1.

13.6. Régression polynomiale

La régression requiert la linéarité de la relation entre les variables dépendante et indépendante(s). Lorsque la relation n'est pas linéaire, il est parfois possible de linéariser la relation en effectuant une transformation sur une ou plusieurs variables. Cependant, dans bien des cas il est impossible de transformer les axes pour rendre la relation linéaire. On doit alors utiliser une forme ou l'autre de régression nonlinéaire. La forme la plus simple de régression non-linéaire est la régression polynomiale dans laquelle les variables indépendantes sont à une puissance plus grande que 1 (Ex : X^2 ou X^3).

- Faites un diagramme de dispersion des résidus (residual) de la régression `logherp ~ logarea` en fonction de `swamp`.

Solution

```
# problème avec les données de manquantes dans logherp
mysub <- subset(mydata, !is.na(logherp))
# ajouter les résidus dans les données
mysub$resloga <- residuals(model.loga)
ggplot(data = mysub, aes(y = resloga, x = swamp)) +
  geom_point() +
  geom_smooth()
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

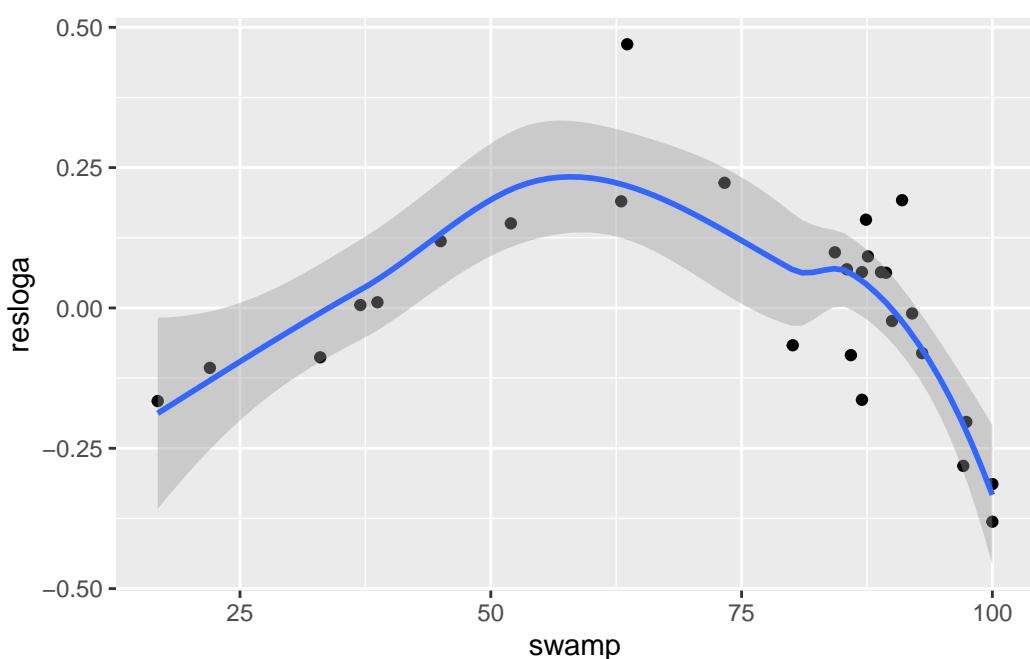


Figure 13.3.: Relation entre swamp et les résidus de la régression entre logherp et logarea

- L'examen de ce graphique suggère qu'il y a une forte relation entre les deux variables, mais qu'elle n'est pas linéaire. Essayez de faire une régression de residual sur swamp . Quelle est votre conclusion?

Solution

```
model.resloga <- lm(resloga ~ swamp, mysub)
summary(model.resloga)
```

Call:

```
lm(formula = resloga ~ swamp, data = mysub)
```

Residuals:

| Min | 1Q | Median | 3Q | Max |
|----------|----------|---------|---------|---------|
| -0.35088 | -0.13819 | 0.00313 | 0.10849 | 0.45802 |

Coefficients:

| | Estimate | Std. Error | t value | Pr(> t) |
|-------------|-----------|------------|---------|----------|
| (Intercept) | 0.084571 | 0.109265 | 0.774 | 0.446 |
| swamp | -0.001145 | 0.001403 | -0.816 | 0.422 |

```
Residual standard error: 0.1833 on 26 degrees of freedom
Multiple R-squared:  0.02498,   Adjusted R-squared:  -0.01252
F-statistic: 0.666 on 1 and 26 DF,  p-value: 0.4219
```

En deux mots, l'ajustement est épouvantable! Malgré le fait que le graphique suggère une relation très forte entre les deux variables. Cependant, cette relation n'est pas linéaire... (ce qui est également apparent si vous examinez les résidus du modèle linéaire).

- Refaites la régression d'en haut, mais cette fois incluez un terme pour représenter $swamp^2$. L'expression devrait apparaître comme: $Y \sim X + I(X^2)$. Que concluez-vous? Qu'est-ce que l'examen des résidus de cette régression multiple révèle?

Solution

```
model.resloga2 <- lm(resloga ~ swamp + I(swamp^2), mysub)
summary(model.resloga2)
```

Call:

```
lm(formula = resloga ~ swamp + I(swamp^2), data = mysub)
```

Residuals:

| Min | 1Q | Median | 3Q | Max |
|-----------|-----------|----------|----------|----------|
| -0.181185 | -0.085350 | 0.007377 | 0.067327 | 0.242455 |

Coefficients:

| | Estimate | Std. Error | t value | Pr(> t) |
|----------------|------------|------------|----------|--------------|
| (Intercept) | -7.804e-01 | 1.569e-01 | -4.975 | 3.97e-05 *** |
| swamp | 3.398e-02 | 5.767e-03 | 5.892 | 3.79e-06 *** |
| I(swamp^2) | -2.852e-04 | 4.624e-05 | -6.166 | 1.90e-06 *** |
| --- | | | | |
| Signif. codes: | 0 '***' | 0.001 '**' | 0.01 '*' | 0.05 '.' |
| | 0.1 | ' ' | 1 | |

Residual standard error: 0.1177 on 25 degrees of freedom

```
Multiple R-squared:  0.6132,    Adjusted R-squared:  0.5823
F-statistic: 19.82 on 2 and 25 DF,  p-value: 6.972e-06
```

Il devient évident que si on corrige la richesse spécifique pour la taille des marais, une fraction importante de la variabilité résiduelle peut être associée à swamp, selon une relation quadratique. Si vous examinez les résidus, vous observerez que l'ajustement est nettement meilleur qu'avec le modèle linéaire.

Solution

```
par(mfrow = c(2, 2))
plot(model.resloga2)
```

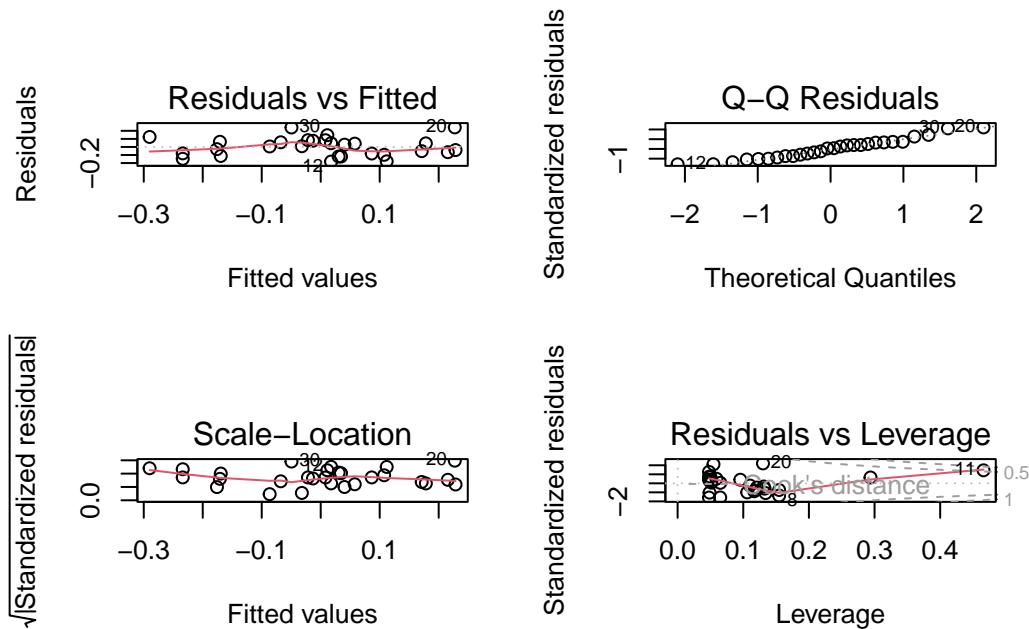


Figure 13.4.: Relation

- En vous basant sur les résultats de la dernière analyse, comment suggérez-vous de modifier le modèle de régression multiple? Quel est, d'après vous, le meilleur modèle? Pourquoi? Ordonnez les différents facteurs en ordre croissant de leur effet sur la richesse spécifique des reptiles.

Suite à ces analyses, il semble opportun d'essayer d'ajuster un modèle incluant logarea, thtden, cpfor2, swamp et swamp^2 :

Solution

```
model.poly1 <- lm(
  logherp ~ logarea + cpfor2 + thtden + swamp + I(swamp^2),
  data = mydata
)
summary(model.poly1)
```

Call:

```
lm(formula = logherp ~ logarea + cpfor2 + thtden + swamp + I(swamp^2),
  data = mydata)
```

Residuals:

| Min | 1Q | Median | 3Q | Max |
|-----------|-----------|-----------|----------|----------|
| -0.201797 | -0.056170 | -0.002072 | 0.051814 | 0.205626 |

Coefficients:

| | Estimate | Std. Error | t value | Pr(> t) |
|----------------|------------|------------|---------|--------------|
| (Intercept) | -3.203e-01 | 1.813e-01 | -1.766 | 0.0912 . |
| logarea | 2.202e-01 | 3.893e-02 | 5.656 | 1.09e-05 *** |
| cpfor2 | -7.864e-04 | 9.955e-04 | -0.790 | 0.4380 |
| thtden | -2.929e-02 | 1.048e-02 | -2.795 | 0.0106 * |
| swamp | 3.113e-02 | 5.898e-03 | 5.277 | 2.70e-05 *** |
| I(swamp^2) | -2.618e-04 | 4.727e-05 | -5.538 | 1.45e-05 *** |
| --- | | | | |
| Signif. codes: | 0 *** | 0.001 ** | 0.01 * | 0.05 . |
| | '' | ' | ' | ' |

Residual standard error: 0.1072 on 22 degrees of freedom

(2 observations deleted due to missingness)

Multiple R-squared: 0.8181, Adjusted R-squared: 0.7767

F-statistic: 19.78 on 5 and 22 DF, p-value: 1.774e-07

Les résultats de cette analyse suggèrent qu'on devrait probablement exclure cpfor2 du modèle:

Solution

```
model.poly2 <- lm(
  logherp ~ logarea + thtden + swamp + I(swamp^2),
  data = mydata
)
summary(model.poly2)
```

Call:

```
lm(formula = logherp ~ logarea + thtden + swamp + I(swamp^2),
  data = mydata)
```

Residuals:

| Min | 1Q | Median | 3Q | Max |
|----------|----------|----------|---------|---------|
| -0.19621 | -0.05444 | -0.01202 | 0.07116 | 0.21295 |

Coefficients:

| | Estimate | Std. Error | t value | Pr(> t) |
|-------------|------------|------------|---------|--------------|
| (Intercept) | -3.461e-01 | 1.769e-01 | -1.957 | 0.0626 . |
| logarea | 2.232e-01 | 3.842e-02 | 5.810 | 6.40e-06 *** |
| thtden | -2.570e-02 | 9.364e-03 | -2.744 | 0.0116 * |
| swamp | 2.956e-02 | 5.510e-03 | 5.365 | 1.89e-05 *** |
| I(swamp^2) | -2.491e-04 | 4.409e-05 | -5.649 | 9.46e-06 *** |

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1063 on 23 degrees of freedom

(2 observations deleted due to missingness)

Multiple R-squared: 0.8129, Adjusted R-squared: 0.7804

F-statistic: 24.98 on 4 and 23 DF, p-value: 4.405e-08

Est-ce qu'il y a possiblement un problème de multicolinéarité?

```
vif(model.poly2)
```

| | logarea | thtden | swamp | I(swamp^2) |
|--|----------|----------|-----------|------------|
| | 1.053193 | 1.123491 | 45.845845 | 45.656453 |

Les valeurs d'inflation de la variance (VIF) pour les deux termes de swamp sont beaucoup plus élevés que le seuil de 5. Cependant, c'est la norme pour les termes polynomiaux et on ne doit pas s'en préoccuper outre mesure, surtout quand les deux termes sont hautement significatifs dans le modèle. Les fortes valeurs de VIF indiquent que les coefficients pour ces deux termes ne sont pas estimés précisément, mais leur utilisation dans le modèle permet tout de même de faire de bonnes prédictions (i.e. ils décrivent la réponse à swamp).

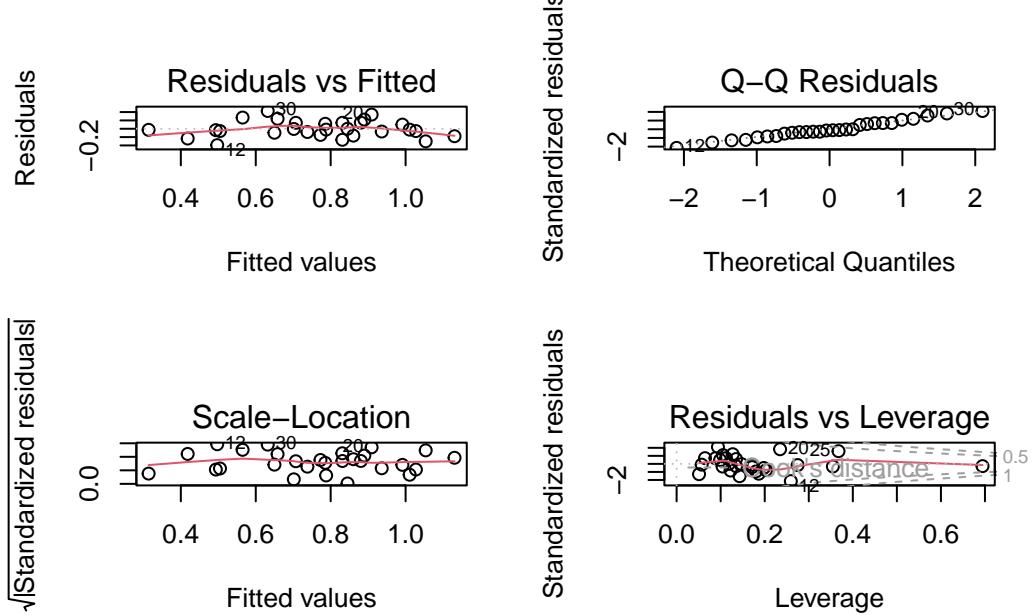
13.7. Vérifier les conditions d'application de modèles de régression multiple

Toutes les techniques de sélection des modèles présument que les conditions d'applications (indépendance, normalité, homoscédasticité, linéarité) sont remplies. Comme il y a un grand nombre de modèles qui peuvent être ajustés, il peut paraître quasi impossible de vérifier si les conditions sont remplies à chaque étape de construction. Cependant, il est souvent suffisant d'examiner les résidus du modèle complet (saturé) puis du modèle final. Les termes qui ne contribuent pas significativement à l'ajustement n'affectent pas beaucoup les résidus et donc les résidus du modèle final sont généralement similaires à ceux du modèle complet.

Examinons donc les graphiques diagnostiques du modèle final:

Solution

```
par(mfrow = c(2, 2))
plot(model.poly2)
```

Figure 13.5.: Conditions d'application du modèle `model.poly2`

Tout semble acceptable dans ce cas. Pour convaincre les sceptiques, on peut faire les tests formels des conditions d'application:

```
shapiro.test(residuals(model.poly2))
```

```
Shapiro-Wilk normality test
```

```
data: residuals(model.poly2)
W = 0.9837, p-value = 0.9278
```

Les résidus ne dévient pas significativement de la normalité. Bien.

```
library(lmtest)
bptest(model.poly2)
```

```
studentized Breusch-Pagan test
```

```
data: model.poly2
BP = 3.8415, df = 4, p-value = 0.4279
```

Pas de déviation d'homoscédasticité non plus. Bien.

```
dwtest(model.poly2)
```

Durbin-Watson test

```
data: model.poly2
DW = 1.725, p-value = 0.2095
alternative hypothesis: true autocorrelation is greater than 0
```

Pas de corrélation sérielle des résidus, donc pas d'évidence de nonindépendance.

```
resettest(model.poly2, type = "regressor", data = mydata)
```

RESET test

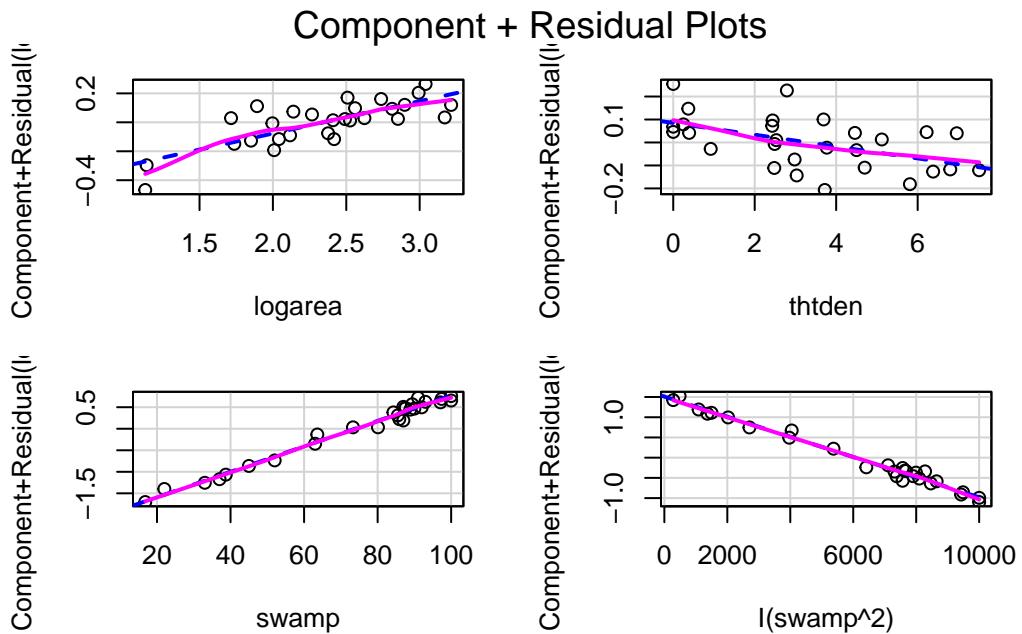
```
data: model.poly2
RESET = 0.9823, df1 = 8, df2 = 15, p-value = 0.4859
```

Et finalement, pas de déviation significative de la linéarité. Donc tout semble acceptable.

13.8. Visualiser la taille d'effet

Les coefficients de la régression multiple peuvent mesurer la taille d'effet, quoiqu'il puisse être nécessaire de les standardiser pour qu'ils ne soient pas influencés par les unités de mesure. Mais un graphique est souvent plus informatif. Dans ce contexte, les graphiques des résidus partiels (appelés components+residual plots dans R) sont particulièrement utiles. Ces graphiques illustrent comment la variable dépendante, corrigée pour l'effet des autres variables dans le modèle, varie avec chacune des variables indépendantes du modèle. Voyons voir:

```
# Evaluate visually linearity and effect size
# component + residual plot
crPlots(model.poly2)
```

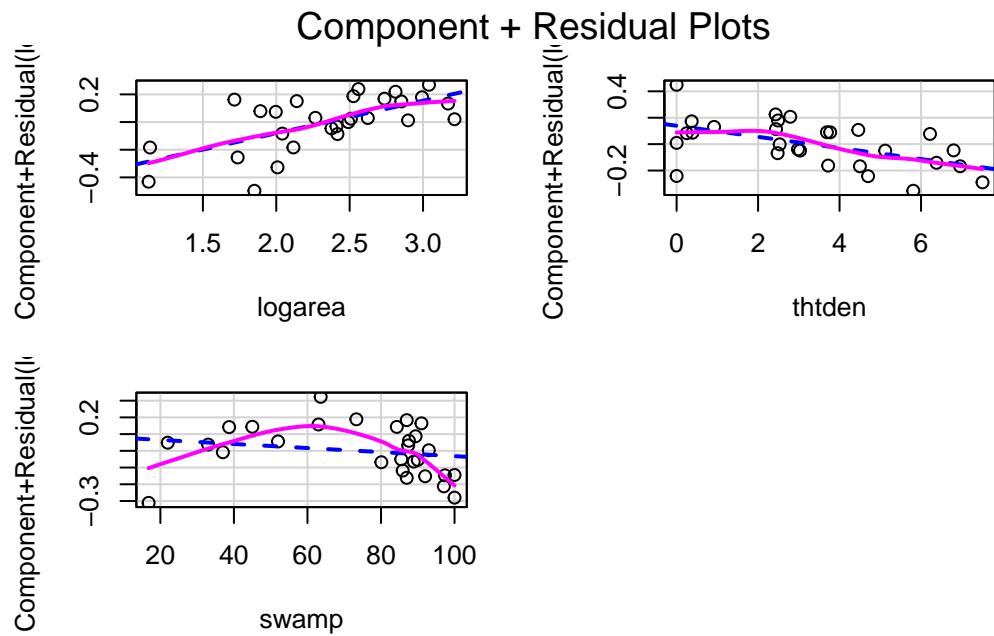
Figure 13.6.: Graphiques de résidus partiels du modèle `model1.poly2`

Notez que l'échelle de l'axe des y varie sur chaque graphique. Pour thtden, la variable dépendante ($\log_{10}(\text{richesse des herpiles})$) varie d'environ 0.4 unités entre la valeur minimum et maximum de thtden. Pour logarea, la variation est d'environ 0.6 unité log. Pour swamp, l'interprétation est plus compliquée parce qu'il y a deux termes qui quantifient son effet, et que ces termes ont des signes opposés (positif pour `swamp` et négatif pour `swamp^2`) ce qui donne une relation curvilinéaire de type parabole. Le graphique ne permet pas de bien visualiser cela. Ceci dit, ces graphique n'indiquent pas vraiment de violation de linéarité.

Pour illustrer ce qui serait visible sur ces graphiques si il y avait une déviation de linéarité, enlevons le terme du second degré pour `swamp`, puis on va refaire ces graphiques et effectuer le test RESET.

Solution

```
model.nopoly <- lm(
  logherp ~ logarea + thtden + swamp,
  data = mydata
)
crPlots(model.nopoly)
```

Figure 13.7.: Graphiques de résidus partiels du modèle `model.nopoly`

La relation non-linéaire avec `swamp` devient évidente. Et le test RESET détecte bien cette non-linéarité:

```
resettest(model.nopoly, type = "regressor")
```

```
RESET test
```

```
data: model.nopoly
RESET = 6.7588, df1 = 6, df2 = 18, p-value = 0.0007066
```

13.9. Tester la présence d'interactions

Lorsqu'il y a plusieurs variables indépendantes, vous devriez toujours garder à l'esprit la possibilité d'interactions. Dans la majorité des situations de régression multiple cela n'est pas évident parce que l'addition de termes d'interaction augmente la multicolinéarité des termes du modèle, et parce qu'il n'y a souvent pas assez d'observations pour éprouver toutes les interactions ou que les observations ne sont pas suffisamment balancées pour faire des tests puissants pour les interactions. Retournons à notre modèle “final” et voyons ce qui se passe si on essaie d'ajuster un modèle saturé avec toutes les interactions:

```
fullmodel.withinteractions <- lm(
  logherp ~ logarea * cpfor2 * thtden * swamp * I(swamp^2),
  data = mydata
)
summary(fullmodel.withinteractions)
```

Call:

```
lm(formula = logherp ~ logarea * cpfor2 * thtden * swamp * I(swamp^2),
  data = mydata)
```

Residuals:

ALL 28 residuals are 0: no residual degrees of freedom!

Coefficients: (4 not defined because of singularities)

| | Estimate | Std. Error | t value | Pr(> t) |
|-----------------------|------------|------------|---------|----------|
| (Intercept) | -5.948e+03 | NaN | NaN | NaN |
| logarea | 3.293e+03 | NaN | NaN | NaN |
| cpfor2 | 7.080e+01 | NaN | NaN | NaN |
| thtden | 9.223e+02 | NaN | NaN | NaN |
| swamp | 1.176e+02 | NaN | NaN | NaN |
| I(swamp^2) | -3.517e-01 | NaN | NaN | NaN |
| logarea:cpfor2 | -3.771e+01 | NaN | NaN | NaN |
| logarea:thtden | -4.781e+02 | NaN | NaN | NaN |
| cpfor2:thtden | -1.115e+01 | NaN | NaN | NaN |
| logarea:swamp | -7.876e+01 | NaN | NaN | NaN |
| cpfor2:swamp | -1.401e+00 | NaN | NaN | NaN |
| thtden:swamp | -1.920e+01 | NaN | NaN | NaN |
| logarea:I(swamp^2) | 5.105e-01 | NaN | NaN | NaN |
| cpfor2:I(swamp^2) | 3.825e-03 | NaN | NaN | NaN |
| thtden:I(swamp^2) | 7.826e-02 | NaN | NaN | NaN |
| swamp:I(swamp^2) | -2.455e-03 | NaN | NaN | NaN |
| logarea:cpfor2:thtden | 5.359e+00 | NaN | NaN | NaN |

| | | | | |
|--|------------|-----|-----|-----|
| logarea:cpfor2:swamp | 8.743e-01 | NaN | NaN | NaN |
| logarea:thtden:swamp | 1.080e+01 | NaN | NaN | NaN |
| cpfor2:thtden:swamp | 2.620e-01 | NaN | NaN | NaN |
| logarea:cpfor2:I(swamp^2) | -5.065e-03 | NaN | NaN | NaN |
| logarea:thtden:I(swamp^2) | -6.125e-02 | NaN | NaN | NaN |
| cpfor2:thtden:I(swamp^2) | -1.551e-03 | NaN | NaN | NaN |
| logarea:swamp:I(swamp^2) | -4.640e-04 | NaN | NaN | NaN |
| cpfor2:swamp:I(swamp^2) | 3.352e-05 | NaN | NaN | NaN |
| thtden:swamp:I(swamp^2) | 2.439e-04 | NaN | NaN | NaN |
| logarea:cpfor2:thtden:swamp | -1.235e-01 | NaN | NaN | NaN |
| logarea:cpfor2:thtden:I(swamp^2) | 7.166e-04 | NaN | NaN | NaN |
| logarea:cpfor2:swamp:I(swamp^2) | NA | NA | NA | NA |
| logarea:thtden:swamp:I(swamp^2) | NA | NA | NA | NA |
| cpfor2:thtden:swamp:I(swamp^2) | NA | NA | NA | NA |
| logarea:cpfor2:thtden:swamp:I(swamp^2) | NA | NA | NA | NA |

Residual standard error: NaN on 0 degrees of freedom

(2 observations deleted due to missingness)

Multiple R-squared: 1, Adjusted R-squared: NaN

F-statistic: NaN on 27 and 0 DF, p-value: NA

Notez les coefficients manquants aux dernières lignes: on ne peut inclure les 32 termes si on a seulement 28 observations. Il manque des observations, le R carré est 1, et le modèle “prédit” parfaitement les données.

Si on essaie une méthode automatique pour identifier le “meilleur” modèle dans ce gâchis, R refuse:

```
step(fullmodel.withinteractions)
```

Error in step(fullmodel.withinteractions): AIC is -infinity for this model, so 'step' cannot proceed

Bon, est-ce qu’on oublie tout ça et qu’on accepte le modèle final sans ce souci des interactions? Non, pas encore. Il y a un compromis possible: comparer notre modèle “final” à un modèle qui contient au moins un sous-ensemble des interactions, par exemple toutes les interactions du second degré, pour éprouver si l’addition de ces interactions améliore beaucoup l’ajustement du modèle.

```
full.model.2ndinteractions <- lm(
  logherp ~ logarea + cpfor2 + thtden + swamp + I(swamp^2)
  + logarea:cpfor2
  + logarea:thtden
  + logarea:swamp
  + cpfor2:thtden
  + cpfor2:swamp
  + thtden:swamp,
  data = mydata
)
summary(full.model.2ndinteractions)
```

Call:

```
lm(formula = logherp ~ logarea + cpfor2 + thtden + swamp + I(swamp^2) +
  logarea:cpfor2 + logarea:thtden + logarea:swamp + cpfor2:thtden +
  cpfor2:swamp + thtden:swamp, data = mydata)
```

Residuals:

| Min | 1Q | Median | 3Q | Max |
|-----------|-----------|----------|----------|----------|
| -0.216880 | -0.036534 | 0.003506 | 0.042990 | 0.175490 |

Coefficients:

| | Estimate | Std. Error | t value | Pr(> t) |
|----------------|------------|------------|---------|--------------|
| (Intercept) | 4.339e-01 | 6.325e-01 | 0.686 | 0.502581 |
| logarea | -1.254e-01 | 2.684e-01 | -0.467 | 0.646654 |
| cpfor2 | -9.344e-03 | 7.205e-03 | -1.297 | 0.213032 |
| thtden | -1.833e-01 | 9.035e-02 | -2.028 | 0.059504 . |
| swamp | 3.569e-02 | 7.861e-03 | 4.540 | 0.000334 *** |
| I(swamp^2) | -3.090e-04 | 7.109e-05 | -4.347 | 0.000500 *** |
| logarea:cpfor2 | 2.582e-03 | 2.577e-03 | 1.002 | 0.331132 |
| logarea:thtden | 7.017e-02 | 3.359e-02 | 2.089 | 0.053036 . |
| logarea:swamp | -5.290e-04 | 2.249e-03 | -0.235 | 0.816981 |

```
cpfor2:thtden -2.095e-04 6.120e-04 -0.342 0.736544
cpfor2:swamp   4.651e-05 5.431e-05  0.856 0.404390
thtden:swamp   2.248e-04 4.764e-04  0.472 0.643336
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Residual standard error: 0.108 on 16 degrees of freedom

(2 observations deleted due to missingness)

Multiple R-squared: 0.8658, Adjusted R-squared: 0.7735

F-statistic: 9.382 on 11 and 16 DF, p-value: 4.829e-05

Ce modèle s'ajuste un peu mieux aux données que les modèle “final” (il explique 86.6% de la variance de logherp, comparé à 81.2% pour le modèle “final” sans interactions), mais il compte deux fois plus de paramètres. De plus, si vous examinez les coefficients, il se passe d’étranges choses: le signe pour logare a changé par exemple. C’est un des symptômes de la multicolinéarité. Allons voir les facteurs d’inflation de la variance:

```
vif(full.model.2ndinteractions)
```

```
there are higher-order terms (interactions) in this model
consider setting type = 'predictor'; see ?vif
```

| | | | | |
|----------------|----------------|---------------|---------------|--------------|
| logarea | cpfor2 | thtden | swamp | I(swamp^2) |
| 49.86060 | 78.49622 | 101.42437 | 90.47389 | 115.08457 |
| logarea:cpfor2 | logarea:thtden | logarea:swamp | cpfor2:thtden | cpfor2:swamp |
| 66.97792 | 71.69894 | 67.27034 | 14.66814 | 29.41422 |
| thtden:swamp | | | | |
| 20.04410 | | | | |

Aie! tous les VIF sont plus grands que 5, pas seulement les termes incluant swamp. Cette forte multicolinéarité empêche de quantifier avec précision l’effet de ces interactions. De plus, ce modèle avec interactions n’est pas plus informatif que le modèle “final” puisque son AIC est plus élevé (souvenez-vous qu’on privilégie le modèle avec la valeur d’AIC la plus basse):

```
AIC(model.poly1)
```

[1] -38.3433

```
AIC(full.model.2ndinteractions)
```

[1] -34.86123

On peut également utiliser la fonction anova() pour comparer l'ajustement des deux modèles et vérifier si l'addition des termes d'intéraction améliore significativement l'ajustement:

```
anova(model.poly1, full.model.2ndinteractions)
```

| Res.Df | RSS | Df | Sum of Sq | F | Pr(>F) |
|--------|-----------|----|-----------|-----------|-----------|
| 22 | 0.2528203 | NA | NA | NA | NA |
| 16 | 0.1865067 | 6 | 0.0663136 | 0.9481497 | 0.4890062 |

Ici, l'addition des termes d'interaction ne réduit pas significativement la variabilité résiduelle du modèle “complet”. Qu'en est-il de la si on compare le modèle avec interaction et notre modèle “final”?

```
anova(model.poly2, full.model.2ndinteractions)
```

| Res.Df | RSS | Df | Sum of Sq | F | Pr(>F) |
|--------|-----------|----|-----------|-----------|-----------|
| 23 | 0.2599923 | NA | NA | NA | NA |
| 16 | 0.1865067 | 7 | 0.0734856 | 0.9005955 | 0.5294411 |

Le test indique que ces deux modèles ont des variances résiduelles comparables, et donc que l'addition des termes d'interaction et de cpfor2 au modèle final n'apporte pas grand chose.

13.10. Recherche du meilleur modèle fondée sur la théorie de l'information

Une des principales critiques des méthodes pas-à-pas (stepwise) est que les valeurs de p ne sont pas strictement interprétables à cause du grand nombre de tests qui sont implicites dans le processus. C'est le problème des comparaisons ou tests multiples: en construisant un modèle linéaire (comme une régression multiple) à partir d'un grand nombre de variables et de leurs interactions, il y a tellement de combinaisons possibles qu'un ajustement de Bonferroni rendrait les tests trop conservateurs.

Une alternative, défendue par Burnham et Anderson (2002, Model selection and multimodel inference: a practical information-theoretic approach. 2nd ed), est d'utiliser l'AIC (ou mieux encore AICc qui est plus approprié quand le nombre d'observations est inférieur à 40 fois le nombre de variables indépendantes) pour ordonner les modèles et identifier un sousensemble de modèles qui sont les meilleurs. On peut ensuite calculer les moyennes des coefficients pondérées par la probabilité que chacun des modèles soit le meilleur pour obtenir des coefficients qui sont plus robustes et moins sensibles à la multicolinéarité.

L'approche de comparaison par *AIC* a d'abord été développé pour comparer un ensemble de modèle préalablement défini basé sur les connaissance du système et les hypothèses biologiques. Cependant, certains ont développé une approche plutôt brutale et sans scrupule de faire tous les modèles possibles et de les comparer par *AIC*. Cette approche a été suivie dans le package MuMIn. Les comparaisons de modèle par *AIC* doivent être faites en utilisant exactement le même jeu de données pour chaque modèle. Il faut donc s'assurer d'enlever les données manquantes et de spécifier dans la fonction `lm` de ne pas marcher si l'y a des données manquantes.

Note

Je ne supporte pas l'approche stepwise ni l'approche par AIC. Je déteste l'approche par la fonction `dredge()` qui selon moi va à l'encontre de la philosophie des AIC et de la parsimonie. Je soutiens de développer un modèle basé sur des hypothèses biologiques et de reporter ce modèle avec tous les effets significatifs ou non.

```
# refaire le modèle en s'assurant qu'il n'y a pas de "NA" et en spécifiant na.action
full.model.2ndinteractions <- update(
  full.model.2ndinteractions,
  . ~ .,
  data = mysub,
  na.action = "na.fail"
)
```

```
library(MuMIn)
dd <- dredge(full.model.2ndinteractions)
```

Fixed term is "(Intercept)"

L'objet dd contient tous les modèles possibles (i.e. ceux qui ont toutes les combinaisons possibles) en utilisant les termes du modèle full.model.2ndinteractions ajusté précédemment. On peut ensuite extraire de l'objet dd le sous-ensemble de modèles qui ont un AICc semblable au meilleur modèle (Burnham et Anderson suggèrent que les modèles qui dévient par plus de 7 unités d'AICc du meilleur modèle ont peu de support empirique).

```
# get models within 2 units of AICc from the best model
top.models.1 <- get.models(dd, subset = delta < 2)
avgmodel1 <- model.avg(top.models.1) # compute average parameters
summary(avgmodel1) # display averaged model
```

Call:

```
model.avg(object = top.models.1)
```

Component model call:

```
lm(formula = logherp ~ <2 unique rhs>, data = mysub, na.action =
na.fail)
```

Component models:

| | df | logLik | AICc | delta | weight |
|-------|----|--------|--------|-------|--------|
| 12345 | 7 | 27.78 | -35.95 | 0.00 | 0.55 |
| 1234 | 6 | 25.78 | -35.56 | 0.39 | 0.45 |

Term codes:

| I(swamp^2) | logarea | swamp | thtden | logarea:thtden |
|------------|---------|-------|--------|----------------|
| 1 | 2 | 3 | 4 | 5 |

Model-averaged coefficients:

(full average)

| | Estimate | Std. Error | Adjusted SE | z value | Pr(> z) |
|----------------|------------|------------|-------------|---------|-----------|
| (Intercept) | -2.145e-01 | 2.308e-01 | 2.406e-01 | 0.891 | 0.373 |
| logarea | 1.356e-01 | 1.089e-01 | 1.119e-01 | 1.213 | 0.225 |
| swamp | 3.180e-02 | 5.971e-03 | 6.273e-03 | 5.070 | 4e-07 *** |
| I(swamp^2) | -2.669e-04 | 4.770e-05 | 5.011e-05 | 5.326 | 1e-07 *** |
| thtden | -6.985e-02 | 5.233e-02 | 5.361e-02 | 1.303 | 0.193 |
| logarea:thtden | 2.131e-02 | 2.487e-02 | 2.545e-02 | 0.837 | 0.403 |

(conditional average)

| | Estimate | Std. Error | Adjusted SE | z value | Pr(> z) |
|----------------|------------|------------|-------------|---------|-----------|
| (Intercept) | -2.145e-01 | 2.308e-01 | 2.406e-01 | 0.891 | 0.3727 |
| logarea | 1.356e-01 | 1.089e-01 | 1.119e-01 | 1.213 | 0.2253 |
| swamp | 3.180e-02 | 5.971e-03 | 6.273e-03 | 5.070 | 4e-07 *** |
| I(swamp^2) | -2.669e-04 | 4.770e-05 | 5.011e-05 | 5.326 | 1e-07 *** |
| thtden | -6.985e-02 | 5.233e-02 | 5.361e-02 | 1.303 | 0.1927 |
| logarea:thtden | 3.882e-02 | 2.114e-02 | 2.237e-02 | 1.735 | 0.0827 . |

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```
confint(avgmodel1) # display CI for averaged coefficients
```

| | 2.5 % | 97.5 % |
|----------------|---------------|--------------|
| (Intercept) | -0.6860022996 | 0.257064603 |
| logarea | -0.0836067896 | 0.354883299 |
| swamp | 0.0195105703 | 0.044099316 |
| I(swamp^2) | -0.0003650809 | -0.000168656 |
| thtden | -0.1749296690 | 0.035236794 |
| logarea:thtden | -0.0050266778 | 0.082666701 |

1. La liste des modèles qui sont à 4 unités ou moins de l'AICc du meilleur modèle. Les variables dans chaque modèle sont codées et on retrouve la clé en dessous du tableau.
2. Pour chaque modèle, en plus de l'AICc, le poids Akaike est calculé. C'est un estimé de la probabilité que ce modèle est le meilleur. Ici on voit que le premier modèle (le meilleur) a seulement 34% des chance d'être vraiment le meilleur.

3. À partir de ce sous-ensemble de modèles, la moyenne pondérée des coefficients (en utilisant les poids Akaike) est calculée, avec un IC à 95%. Notez que les termes absents d'un modèle sont considérés avoir un coefficient de 0 pour ce terme.

13.11. Bootstrap et régression multiple

Quand les données ne rencontrent pas les conditions d'application de normalité et d'homoscédasticité et que les transformations n'arrivent pas à corriger ces violations, le bootstrap peut être utilisé pour calculer des intervalles de confiance pour les coefficients. Si la distribution des coefficients bootstrappés est symétrique et approximativement normale, on peut utiliser les percentiles empiriques pour calculer les limites de confiance.

Le code qui suit, utilisant le package `simpleboot`, a été conçu pour être facilement modifiable et calcule les limites des IC à partir des percentiles empiriques.

```
#####
#####
# Bootstrap analysis the simple way with library simpleboot
# Define model to be bootstrapped and the data source used
mymodel <- lm(logherp ~ logarea + thtden + swamp + I(swamp^2), data = mydata)
# Set the number of bootstrap iterations
nboot <- 1000
library(simpleboot)
# R is the number of bootstrap iterations
# Setting rows to FALSE indicates resampling of residuals
mysimpleboot <- lm.boot(mymodel, R = nboot, rows = FALSE)
# Extract bootstrap coefficients
myresults <- sapply(mysimpleboot$boot.list, function(x) x$coef)
# Transpose matrix so that lines are bootstrap iterations and columns are coefficients
tmyresults <- t(myresults)
```

Vous pouvez ensuite faire des graphiques pour voir les résultats. Lorsque vous tournez ce code, il y aura une pause pour vous permettre d'examiner la distribution pour chaque coefficient du modèle sur des graphiques:

```
# Plot histograms of bootstrapped coefficients
ncoefs <- length(data.frame(tmyresults))
par(mfrow = c(1, 2), mai = c(0.5, 0.5, 0.5, 0.5), ask = TRUE)
for (i in 1:ncoefs) {
  lab <- colnames(tmyresults)[i]
  x <- tmyresults[, i]
  plot(density(x), main = lab, xlab = "")
  abline(v = mymodel$coef[i], col = "red")
  abline(v = quantile(x, c(0.025, 0.975)))
  hist(x, main = lab, xlab = "")
  abline(v = quantile(x, c(0.025, 0.975)))
  abline(v = mymodel$coef[i], col = "red")
}
}
```

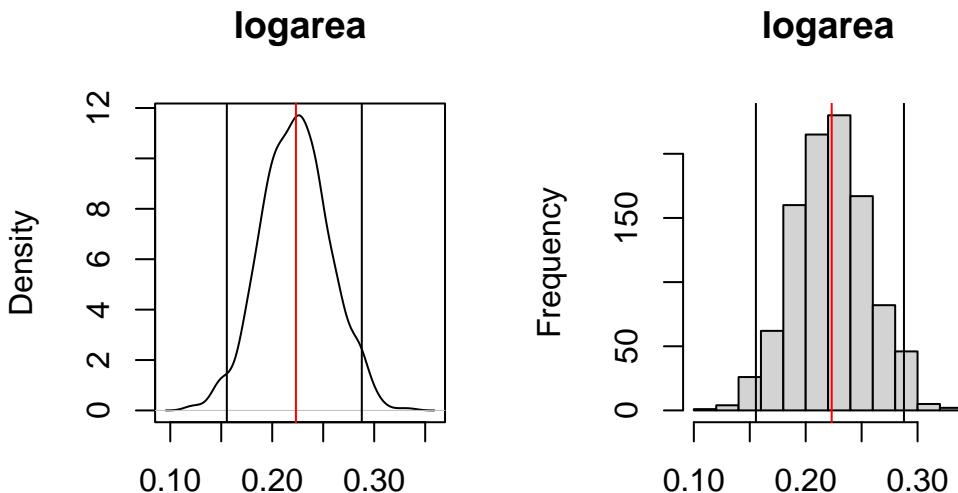


Figure 13.8.: Distribution des estimé par bootstrap pour logarea

Le graphique de droite illustre la densité lissée (kernel density) et celui de gauche est l'histogramme des estimés bootstrap du coefficient. La ligne rouge sur le graphique indique la valeur du coefficient ordinaire (pas bootstrap) et les deux lignes verticales noires marquent les limites de l'intervalle de confiance à 95%. Ici l'IC ne contient pas 0, et donc on peut conclure que l'effet de logarea sur logherp est significativement positif.

Les limites précises peuvent être obtenues par:

```
# Display empirical bootstrap quantiles (not corrected for bias)
p <- c(0.005, 0.01, 0.025, 0.05, 0.95, 0.975, 0.99, 0.995)
apply(tmyresults, 2, quantile, p)
```

| | (Intercept) | logarea | thtden | swamp | I(swamp^2) |
|-------|-------------|-----------|--------------|------------|---------------|
| 0.5% | -0.74102813 | 0.1421013 | -0.045620804 | 0.01584749 | -0.0003638466 |
| 1% | -0.70942482 | 0.1450945 | -0.043045264 | 0.01741514 | -0.0003583716 |
| 2.5% | -0.66630933 | 0.1554787 | -0.040899292 | 0.01886882 | -0.0003317467 |
| 5% | -0.62209885 | 0.1708719 | -0.038708552 | 0.02102114 | -0.0003163083 |
| 95% | -0.07687366 | 0.2804950 | -0.011258495 | 0.03806321 | -0.0001782677 |
| 97.5% | -0.03149582 | 0.2878404 | -0.008243260 | 0.03993189 | -0.0001628157 |
| 99% | 0.01341771 | 0.2980666 | -0.006353121 | 0.04281474 | -0.0001516612 |
| 99.5% | 0.04806317 | 0.3018333 | -0.003387129 | 0.04397008 | -0.0001311139 |

Ces intervalles de confiances ne sont pas fiables si la distribution des estimés bootstrap n'est pas Gaussienne. Dans ce cas, il vaut mieux calculer des coefficients non-biaisés (bias-corrected accelerated confidence limits, BCa):

```
#####
# Bootstrap analysis in multiple regression with BCa confidence intervals
# Preferable when parameter distribution is far from normal
# Bootstrap 95% BCa CI for regression coefficients

library(boot)

# function to obtain regression coefficients for each iteration
bs <- function(formula, data, indices) {
  d <- data[indices, ] # allows boot to select sample
  fit <- lm(formula, data = d)
  return(coef(fit))
}

# bootstrapping with 1000 replications
results <- boot(
  data = mydata, statistic = bs, R = 1000,
  formula = logherp ~ logarea + thtden + swamp + I(swamp^2)
)
# view results
```

Pour obtenir les résultats, le code suivant va produire le graphique standard pour chaque coefficient, et les estimés BCa pour l'intervalle de confiance

```
plot(results, index = 1) # intercept
plot(results, index = 2) # logarea
plot(results, index = 3) # thtden
plot(results, index = 4) # swamp
plot(results, index = 5) # swamp^2

# get 95% confidence intervals
boot.ci(results, type = "bca", index = 1)
boot.ci(results, type = "bca", index = 2)
boot.ci(results, type = "bca", index = 3)
boot.ci(results, type = "bca", index = 4)
boot.ci(results, type = "bca", index = 5)
```

Pour logarea, cela donne:

BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS

Based on 1000 bootstrap replicates

CALL :

```
boot.ci(boot.out = results, type = "bca", index = 2)
```

Intervals :

Level BCa

95% (0.1155, 0.3168)

Calculations and Intervals on Original Scale

Notez que l'intervalle BCa va de 0.12 à 0.32, alors que l'intervalle standard était de 0.16 à 0.29. L'intervalle BCa est ici plus grand du côté inférieur et plus petit du côté supérieur comme il se doit compte tenu de la distribution non-Gaussienne et asymétrique des estimés bootstrap.

13.12. Test de permutation

Les tests de permutations sont plus rarement effectués en régression multiple que le bootstrap. Voici un fragment de code pour le faire tout de même.

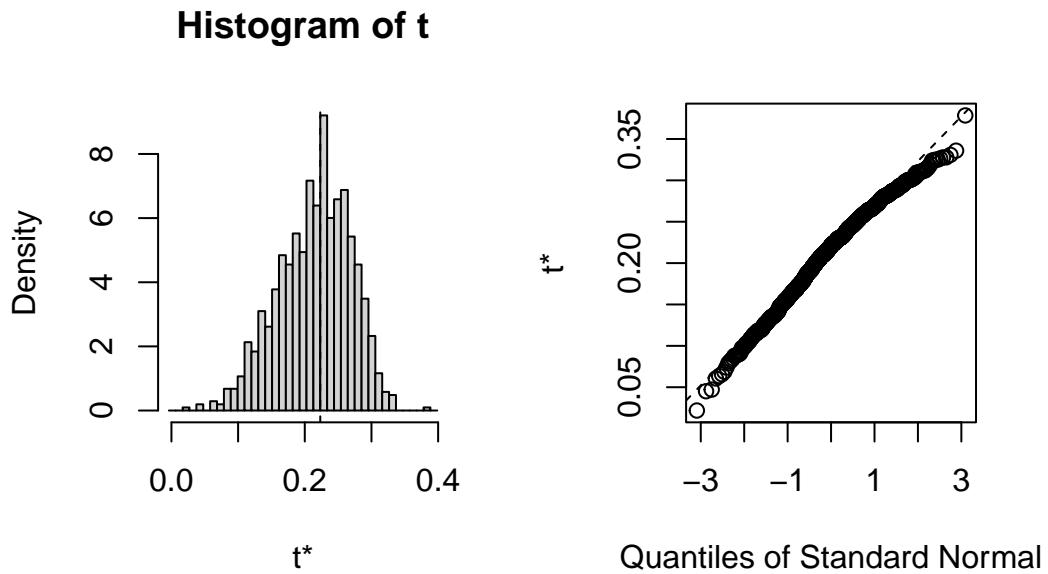


Figure 13.9.

```
#####
#####
# Permutation in multiple regression
#
# using lmperm library
library(lmPerm)
# Fit desired model on the desired dataframe
mymodel <- lm(logherp ~ logarea + thtden + swamp + I(swamp^2),
  data = mydata
)
mymodelProb <- lmp(
  logherp ~ logarea + thtden + swamp + I(swamp^2),
  data = mydata, perm = "Prob"
)
summary(mymodel)
summary(mymodelProb)
```

Chapitre 14

ANCOVA et modèle linéaire général

Objectifs de ce chapitre :

- Utiliser R pour faire une analyse de covariance (ANCOVA) et ajuster des modèles ayant des variables indépendantes continues et discontinues (modèle linéaire général)
- Utiliser R pour vérifier les conditions préalables à ces modèles
- Utiliser R pour comparer l'ajustement de modèles statistiques
- Utiliser R pour faire des tests de bootstrap et de permutation sur des modèles avec des variables indépendantes continues et discontinues.

14.1. Paquets et données requises pour le labo

Pour ce laboratoire, vous aurez besoin de :

- Paquets R :
 - `ggplot2` 
 - `car` 
 - `lmtest` 
- Jeux de données :
 - “anc1dat.csv”
 - “anc3dat.csv”

14.2. Modèle linéaire général

Les modèles linéaires généraux ou “General Linear Model” en anglais sont différent des modèles linéaires généralisés (ou “Generalized Linear Model”, GLM).

Les modèles linéaires généraux sont des modèles statistiques de la forme $Y = B\mathbf{X} + E$, où

- **Y** est un vecteur contenant la variable dépendante *continue*,
- **B** est un vecteur des paramètres estimés,
- **X** et la matrice des différents variables indépendantes et
- **E** est un vecteur de résidus *homoscédastiques* et *normalement distribués*.

Tous les tests que nous avons étudiés jusque là (test de t, régression linéaire simple, ANOVA à un facteur de classification, ANOVA à plusieurs facteurs de classification et régression multiple) sont formulés ainsi. Notez que tous les modèles que nous avons rencontrés à ce jour ne contiennent qu'**un** type de variable indépendante : soit continue (numérique) ou discontinue (catégorique). Dans cet exercice de laboratoire, vous allez ajuster des modèles qui ont les **deux** types de variables indépendantes.

14.3. ANCOVA

ANCOVA est l'abréviation pour l'analyse de covariance. C'est un type de modèle linéaire général dans lequel il y a une (ou plusieurs) variable indépendante continue (parfois appelé la *covariable*) et une (ou plusieurs) variable indépendante discontinue (catégorique). Dans la présentation traditionnelle de l'ANCOVA dans les manuels de biostatistique, le modèle ANCOVA ne contient pas de termes d'interaction entre les variables continues et discontinues. Par conséquent, on doit précéder l'ajustement de ce modèle (réduit parce que sans terme d'interaction), par un test de signification de l'interaction qui correspond à éprouver l'égalité des pentes (coefficients pour la ou les variables continues) entre les différents niveaux de la ou des variables discontinues (i.e un **test d'homogénéité des pentes**).

14.4. Homogénéité des pentes

Pour répondre à de nombreuses questions biologiques, il est nécessaire de déterminer si deux (ou plus) régressions diffèrent significativement. Par exemple, pour comparer l'efficacité de deux insecticides on doit comparer la relation entre leur dose et la mortalité. Ou encore, pour comparer le taux de croissance des mâles et des femelles on doit comparer la relation entre la taille et l'âge des mâles et des femelles.

Comme chaque régression linéaire est décrite par deux paramètres, la **pente** et l'**ordonnée à l'origine**, on doit considérer les deux dans la comparaison. Le modèle d'ANCOVA, à strictement parler, n'éprouve que l'hypothèse d'égalité des ordonnées à l'origine. Cependant, avant d'ajuster ce modèle, il faut éprouver l'hypothèse d'égalité des pentes (homogénéité des pentes).

14.4.1. Cas 1 - La taille en fonction de l'âge (exemple avec pente commune)

🔥 Exercice

En utilisant les données du fichier `anc1dat.csv`, éprouvez l'hypothèse que le taux de croissance des esturgeons mâles et femelles de The Pas est le même (données de 1978-1980). Comme mesure du taux de croissance, nous allons utiliser la pente de la régression du log 10 de la longueur à la fourche, `1fk1`, sur le log 10 de l'`age`.

Commençons par examiner les données. Pour faciliter la comparaison, il serait utile de tracer la droite de régression et la trace "lowess" pour évaluer plus facilement la linéarité. On peut aussi ajouter un peu de code pour obtenir des légendes plus complètes (remarquez l'utilisation de la commande `expression()` pour obtenir des indices) :

```
anc1dat <- read.csv("data/anc1dat.csv")
anc1dat$sex <- as.factor(anc1dat$sex)

myplot <- ggplot(data = anc1dat, aes(x=lage,      y=log10(fklength)))+facet_grid(.~sex)+geom_point()
myplot <- myplot+
  geom_smooth(method = lm, se=FALSE)+
  geom_smooth(se=FALSE, color="red") +
  labs(
    y = expression(log[10]~(Fork-length)),
    x = expression(log[10]~(Age))
  )
myplot
```

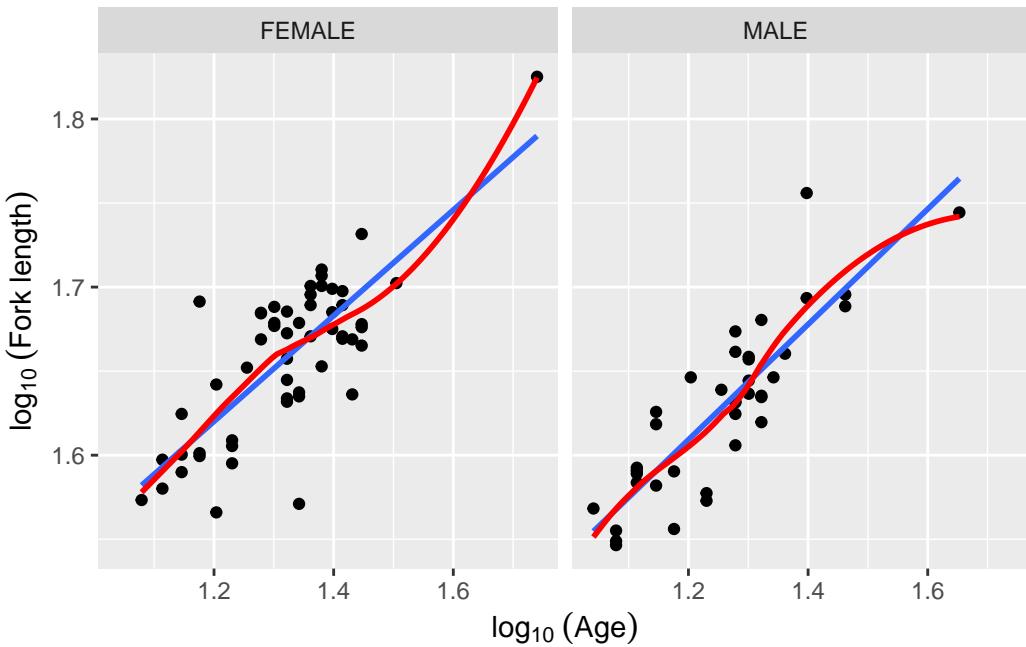


Figure 14.1.: Longueur des esturgeons en fonction de l'âge entre mâles et femelles.

La transformation log-log rend la relation linéaire et, à première vue, il ne semble pas y avoir de problème évident avec les conditions d'application. Ajustons donc le modèle complet avec l'interaction:

```
model.full1<-lm(lfkl ~ sex + lage + sex:lage, data = anc1dat)
Anova(model.full1, type = 3)
```

| | Sum Sq | Df | F value | Pr(>F) |
|-------------|-----------|----|-------------|-----------|
| (Intercept) | 0.6444361 | 1 | 794.8182143 | 0.0000000 |
| sex | 0.0004089 | 1 | 0.5043251 | 0.4794836 |
| lage | 0.0725916 | 1 | 89.5311705 | 0.0000000 |
| sex:lage | 0.0002730 | 1 | 0.3366921 | 0.5632277 |
| Residuals | 0.0713501 | 88 | NA | NA |

Notez que l'on utilise la fonction `Anova()` du paquet `car` avec un “A” majuscule au lieu de la fonction native `anova()` (avec un “a” minuscule) pour obtenir les **sommes de carrés de type III (partiels)**. Ces sommes des carrés des écarts sont calculées comme si la variable était entrée la dernière dans le modèle et correspondent à la différence entre la variance expliquée par le modèle complet et par le modèle sans cette variable.

La fonction native `anova()` donne les **sommes des carrés séquentielles**, calculées au fur et à mesure que chaque variable est ajoutée au modèle nul avec seulement une ordonnée à l'origine. Dans de rares cas, les sommes des carrés de type I et III sont égales (quand le design est parfaitement orthogonal ou balancé). Dans la vaste majorité des cas, les sommes des carrés de type I et III sont différentes et **je vous conseille de toujours utiliser les sommes des carrés de type III dans vos analyses**.

À partir de cette analyse, on devrait accepter les hypothèses nulles (1) d'égalité des pentes pour les deux sexes, et (2) que les ordonnées à l'origine sont les mêmes pour les deux sexes. Mais, avant d'accepter ces conclusions, il faut vérifier si les données rencontrent les conditions d'application, comme d'habitude...

Solution

```
par(mfrow = c(2, 2))
plot(model.full1)
```

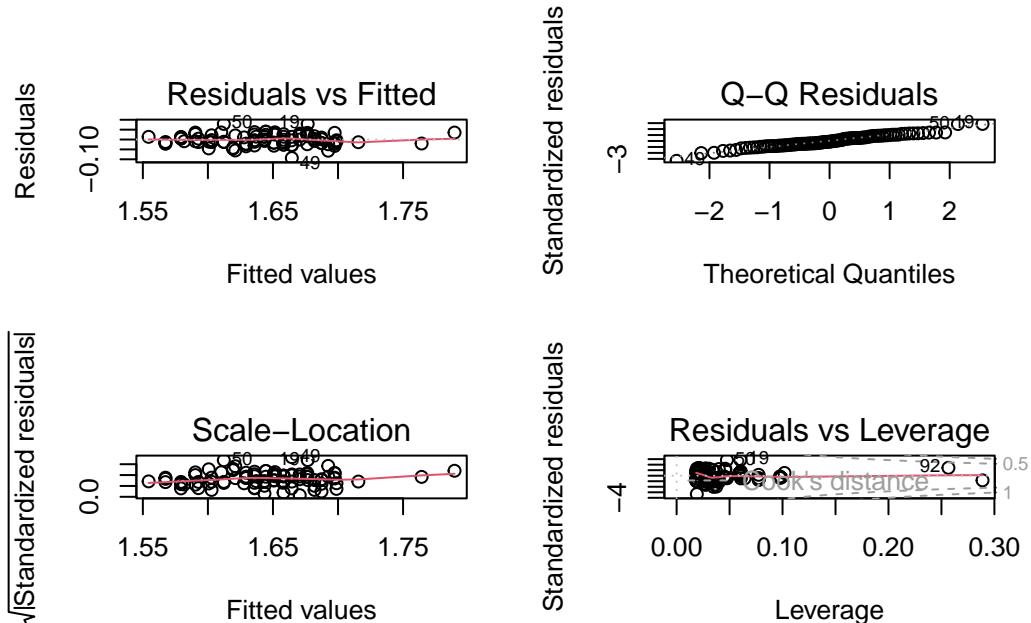


Figure 14.2.: Conditions d'applications du modèle `model.full1`

En ce qui concerne la **normalité**, ça a l'air d'aller quoiqu'il y a quelques points, en haut à droite, qui dévient. Si on effectue le test de Wilk-Shapiro ($W = .9764$, $p = 0.09329$), on confirme que les résidus ne dévient pas significativement de la normalité.

Les résidus semblent **homoscédastiques**, mais si vous voulez vous en assurer, vous pouvez l'éprouver par un des tests formels. Ici j'utilise le test Breusch-Pagan, qui est approprié quand certaines des variables indépendantes sont

continues (Le test de Levene n'est approprié que lorsqu'il n'y a que des variables discontinues).

```
bptest(model.full1)
```

```
studentized Breusch-Pagan test
```

```
data: model.full1
BP = 0.99979, df = 3, p-value = 0.8013
```

Comme l'hypothèse nulle de ce test est que les résidus sont homoscédastiques, et que p est relativement élevé, le test confirme l'évaluation visuelle. De plus, il n'y a pas de tendance évidente dans les résidus, suggérant qu'il n'y a pas de problème de **linéarité**. Ce qui peut également être éprouvé formellement:

```
resettest(model.full1, power = 2:3, type = "regressor", data = anc1dat)
```

```
RESET test
```

```
data: model.full1
RESET = 0.59861, df1 = 2, df2 = 86, p-value = 0.5519
```

La dernière condition d'application est qu'il n'y a **pas d'erreur de mesure** sur la variable indépendante continue. On ne peut pas vraiment éprouver cette condition, mais on sait que des estimés indépendants de l'âge des poissons obtenus par différents chercheurs donnent des âges qui concordent avec moins de 1-2 ans d'écart, ce qui est inférieur au 10% de la fourchette observée des âges et donc acceptable pour des analyses de modèles de type I (*attention ici on ne parle pas des SC de type I, je sais, c'est facile de confondre...*)

Vous noterez qu'il y a une observation qui a un résidu normalisé (Standardized residual) qui est élevé, i.e. une valeur extrême (cas numéro 49). Éliminez-la de l'ensemble de données et refaites l'analyse. Vos conclusions changent-elles?

```
model.full.no49<-lm(lfkl ~ sex + lage + sex:lage, data = anc1dat[c(-49),])
Anova(model.full.no49, type=3)
```

| | Sum Sq | Df | F value | Pr(>F) |
|-------------|-----------|----|-------------|-----------|
| (Intercept) | 0.6425466 | 1 | 895.9393626 | 0.0000000 |
| sex | 0.0003782 | 1 | 0.5273066 | 0.4696905 |
| lage | 0.0737792 | 1 | 102.8745821 | 0.0000000 |
| sex:lage | 0.0002248 | 1 | 0.3134569 | 0.5770053 |
| Residuals | 0.0623943 | 87 | NA | NA |

La conclusion ne change pas après avoir enlevé la valeur extrême. Comme on n'a pas de bonne raison d'éliminer cette valeur, il est probablement mieux de la conserver. Un examen des conditions d'application après avoir enlevé cette valeur révèle qu'elles sont toutes rencontrées.

14.4.2. Cas 2 - Taille en fonction de l'âge (exemple avec des pentes différentes)

🔥 Exercice

Le fichier anc3dat.csv contient des données sur des esturgeons mâles de deux sites (locate) : Lake of the Woods dans le Nord-Ouest de l'Ontario et Chruchill River dans le Nord du Manitoba. En utilisant la même procédure, éprouvez l'hypothèse que la pente de la régression de lfk1 sur lage est la même aux deux sites (alors Locate est la variable catégorique au lieu de sex). Que concluez-vous?

```
anc3dat <- read.csv("data/anc3dat.csv")

myplot <- ggplot(data = anc3dat, aes(x = lage, y = log10(fklnngth))) +
  facet_grid(. ~ locate) +
  geom_point() +
  geom_smooth(method = lm, se = FALSE) +
  geom_smooth(se = FALSE, color = "red") +
  labs(
    y = expression(log[10] ~ (Fork ~ length)),
    x = expression(log[10] ~ (Age)))
  )

myplot

model.full2 <- lm(lfk1 ~ lage + locate + lage:locate, data = anc3dat)
Anova(model.full2, type = 3)
```

| | Sum Sq | Df | F value | Pr(>F) |
|-------------|-----------|----|------------|-----------|
| (Intercept) | 0.6295064 | 1 | 1078.63230 | 0.0000000 |
| lage | 0.0777287 | 1 | 133.18483 | 0.0000000 |
| locate | 0.0096826 | 1 | 16.59072 | 0.0001012 |
| lage:locate | 0.0090901 | 1 | 15.57541 | 0.0001592 |
| Residuals | 0.0513582 | 88 | NA | NA |

Figure 14.3.: Longueur des esturgeons en fonction de l'âge selon le site d'étude d'après anc3dat.csv

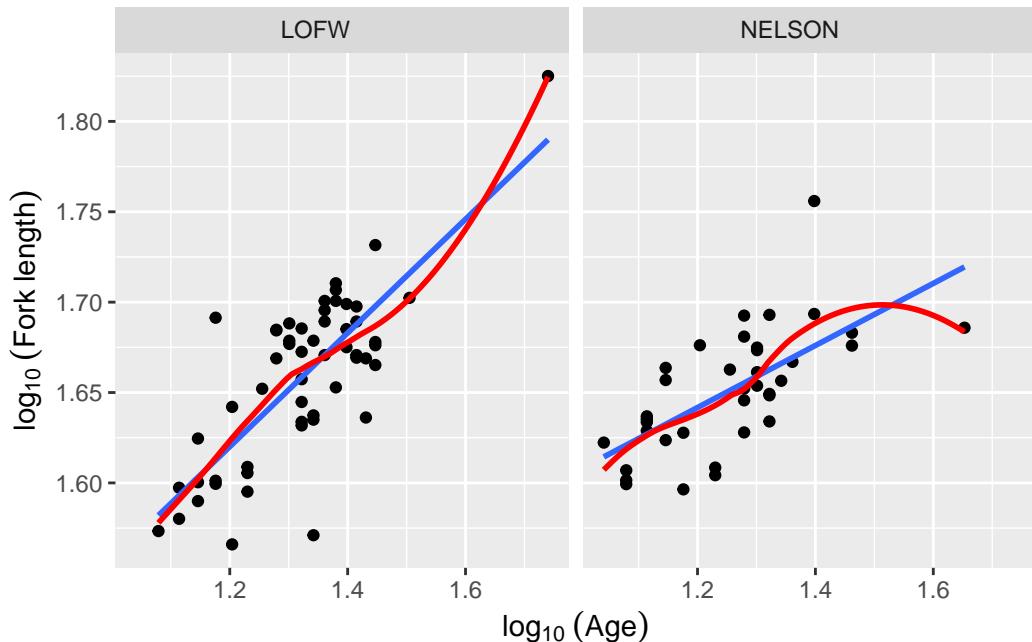


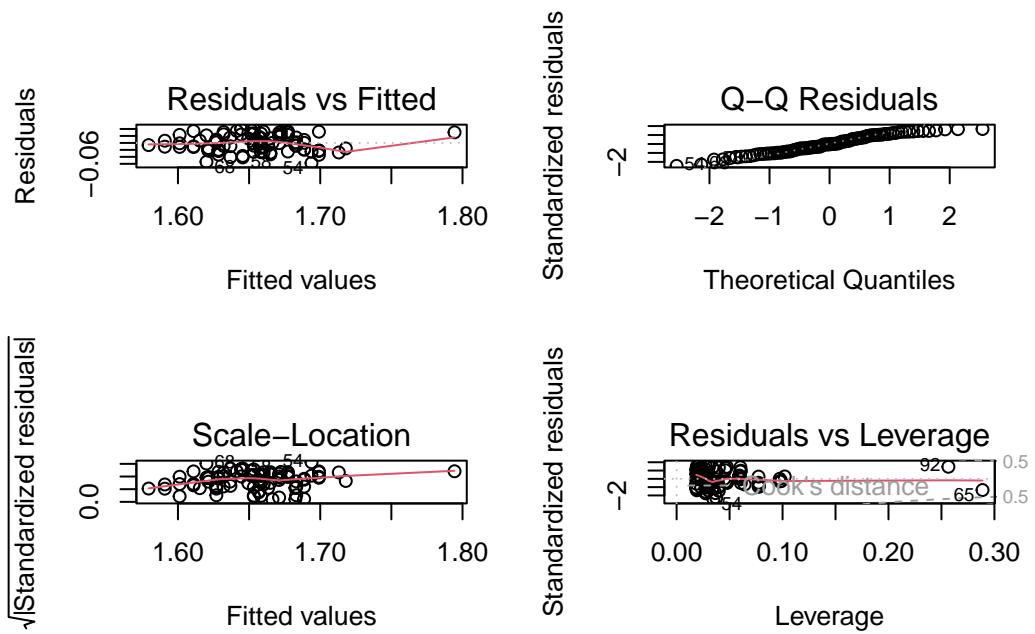
Figure 14.4.: Longueur des esturgeons en fonction de l'âge selon le site d'étude d'après anc3dat.csv

Ici, on rejette les hypothèses nulles (1) que les pentes sont les mêmes dans les deux sites et (2) que les ordonnées à l'origine sont égales. En d'autres mots, si on veut prédire la longueur à la fourche d'un esturgeon à un âge donné précisément, il faut savoir de quel site il provient. Puisque les pentes diffèrent, il faut estimer des régressions séparées.

Mais avant d'accepter ces conclusions, on doit s'assurer que les conditions d'application sont rencontrées :

Solution

```
par(mfrow = c(2,2))
plot(model.full12)
```

Figure 14.5.: Conditions d'applications du modèle `model.full12`

Si on examine les résidus selon les méthodes habituelles, on voit qu'il n'y a pas de problème de linéarité, ni d'homoscédasticité ($BP = 2.8721$, $p = 0.4118$). Cependant, le test de Wilk-Shapiro est significatif ($W=0.97$, $p = 0.03$). Étant donné la taille assez grande de l'échantillon ($N=92$), ce test a beaucoup de puissance, même si la déviation de normalité ne semble pas très grande. Compte-tenu de la robustesse relative des LM, de la taille de l'échantillon, on ne s'inquiète pas trop de cette déviation de normalité.

Donc, comme les conditions des LM sont suffisamment remplies, on peut accepter les résultats donnés par R. Tous les termes sont significatifs (location, lage, interaction). Ce modèle complet est équivalent à ajuster des régressions séparées pour chaque site. Pour obtenir les coefficients, on peut ajuster des régressions simples sur chaque sous-ensemble, ou extraire les coefficients ajustés du modèle complet:

```
model.full12
```

Call:

```
lm(formula = lfkl ~ lage + locate + lage:locate, data = anc3dat)
```

Coefficients:

| (Intercept) | lage | locateNELSON |
|-------------|--------|--------------|
| 1.2284 | 0.3253 | 0.2207 |

```
lage:locateNELSON
-0.1656
```

Par défaut, la variable `locate` est encodée comme 0 pour le site qui vient le premier en ordre alphabétique (LofW) et 1 pour l'autre (Nelson). Les régressions pour chaque site deviennent donc:

Pour LofW:

$$\begin{aligned} lfk \&= 1.2284 + 0.3253 \times \text{lage} + 0.2207 \times 0 - 0.1656 \times 0 \times \text{lage} \\ &= 1.2284 + 0.3253 \times \text{lage} \end{aligned}$$

Pour Nelson:

$$\begin{aligned} lfk \&= 1.2284 + 0.3253 \times \text{lage} + 0.2207 \times 1 - 0.1656 \times 1 \times \text{lage} \\ &= 1.4491 + 0.1597 \times \text{lage} \end{aligned}$$

Vous pouvez vérifier en ajustant séparément les régressions pour chaque site:

```
by(anc3dat, anc3dat$locate,function(x) lm(lfkl~lage, data=x))
```

anc3dat\$locate: LOFW

Call:

```
lm(formula = lfkl ~ lage, data = x)
```

Coefficients:

| (Intercept) | lage |
|-------------|--------|
| 1.2284 | 0.3253 |

anc3dat\$locate: NELSON

Call:

```
lm(formula = lfkl ~ lage, data = x)
```

Coefficients:

| (Intercept) | lage |
|-------------|--------|
| 1.4491 | 0.1597 |

14.5. Le modèle d'ANCOVA

Si le test d'homogénéité des pentes indique qu'elles diffèrent, alors on devrait estimer des régressions individuelles pour chaque niveau des variables discontinues.

Cependant, si on accepte l'hypothèse d'égalité des pentes, l'étape suivante est de **comparer les ordonnées à l'origine**. Selon la méthode traditionnelle i.e., on ajuste un modèle avec la variable catégorique et la variable continue, mais sans interaction (le modèle ANCOVA sensus stricto) et on utilise la somme des carrés des écarts de type III, avec la fonction `Anova()`.

L'autre approche consiste à utiliser les résultats de l'analyse du modèle complet, et tester la signification de chaque terme à partir des sommes des carrés partiels. C'est plus rapide, mais moins puissant. Dans la plupart des cas, cette perte de puissance n'est pas trop préoccupante, sauf lorsque le modèle est très complexe et contient de nombreuses interactions non-significatives. Je vous suggère d'utiliser l'approche simplifiée, et de n'utiliser l'approche traditionnelle que lorsque vous acceptez l'hypothèse d'égalité des ordonnées à l'origine. Pourquoi?

Puisque l'approche simplifiée est moins puissante, si vous rejetez quand même H_0 , alors votre conclusion ne changera pas, mais sera seulement renforcée, en utilisant l'approche traditionnelle.

Ici, je vais comparer l'approche traditionnelle et l'approche simplifiée. Rappelez-vous que vous voulez évaluer l'égalité des ordonnées à l'origine **après avoir déterminé que les pentes étaient égales**. Éprouver l'égalité des ordonnées à l'origine quand les pentes diffèrent (ou, si vous préférez, quand il y a une interaction) est rarement sensé, peut facilement être mal interprété, et ne devrait être effectué que rarement.

De retour aux données de `anc1dat.csv`, en comparant la relation entre `lfkl` et `lage` entre les `sexes`, nous avions obtenu les résultats suivants pour le modèle complet avec interactions :

```
Anova(model.full1, type = 3)
```

| | Sum Sq | Df | F value | Pr(>F) |
|-------------|-----------|----|-------------|-----------|
| (Intercept) | 0.6444361 | 1 | 794.8182143 | 0.0000000 |
| sex | 0.0004089 | 1 | 0.5043251 | 0.4794836 |
| lage | 0.0725916 | 1 | 89.5311705 | 0.0000000 |
| sex:lage | 0.0002730 | 1 | 0.3366921 | 0.5632277 |
| Residuals | 0.0713501 | 88 | NA | NA |

On avait déjà conclu que la pente ne varie pas entre les sexes (i.e. l’interaction n’est pas significative). Notez que la valeur de p associée au sexe (0.4795) n’est pas significative non plus.

De l’autre côté, selon l’approche traditionnelle, l’inférence quand à l’effet du sexe se fait à partir du modèle réduit (le modèle ANCOVA sensus stricto):

```
model.ancova <- lm(lfkl ~ sex + lage, data = anc1dat)
Anova(model.ancova, type = 3)
```

| | Sum Sq | Df | F value | Pr(>F) |
|-------------|-----------|----|-------------|-----------|
| (Intercept) | 1.1348025 | 1 | 1410.123239 | 0.0000000 |
| sex | 0.0014899 | 1 | 1.851324 | 0.1770653 |
| lage | 0.1433772 | 1 | 178.162744 | 0.0000000 |
| Residuals | 0.0716231 | 89 | NA | NA |

```
summary(model.ancova)
```

Call:

```
lm(formula = lfkl ~ sex + lage, data = anc1dat)
```

Residuals:

| Min | 1Q | Median | 3Q | Max |
|-----------|-----------|-----------|----------|----------|
| -0.093992 | -0.018457 | -0.000876 | 0.022491 | 0.081161 |

Coefficients:

| | Estimate | Std. Error | t value | Pr(> t) |
|----------------|-----------|------------|---------|------------|
| (Intercept) | 1.225533 | 0.032636 | 37.552 | <2e-16 *** |
| sexMALE | -0.008473 | 0.006228 | -1.361 | 0.177 |
| lage | 0.327253 | 0.024517 | 13.348 | <2e-16 *** |
| --- | | | | |
| Signif. codes: | 0 *** | 0.001 ** | 0.01 * | 0.05 . |

Residual standard error: 0.02837 on 89 degrees of freedom

```
Multiple R-squared:  0.696, Adjusted R-squared:  0.6892
F-statistic: 101.9 on 2 and 89 DF,  p-value: < 2.2e-16
```

Dans ce modèle, sex n'est pas significatif et on conclue donc que l'ordonnée à l'origine ne diffère pas entre les sexes. Notez que la valeur de p est plus petite (0.1771 vs 0.4795), ce qui reflète la puissance accrue de l'approche traditionnelle. Toutefois, les conclusions sont les mêmes : les ordonnées à l'origine ne diffèrent pas.

🔥 Exercice

En examinant les graphiques diagnostiques, vous noterez qu'il y a trois observations dont la valeur absolue du résidu est grande (cas 19, 49, et 50). Ces observations pourraient avoir un effet disproportionné sur les résultats de l'analyse. Éliminez-les et refaites l'analyse. Les conclusions changent-elles ?

```
model.ancova.nooutliers <- lm(lfkl ~ sex + lage, data = anc1dat[c(-49, -50, -19),])
Anova(model.ancova.nooutliers, type = 3)
```

| | Sum Sq | Df | F value | Pr(>F) |
|-------------|-----------|----|-------------|-----------|
| (Intercept) | 1.0916027 | 1 | 1896.520424 | 0.0000000 |
| sex | 0.0023238 | 1 | 4.037371 | 0.0476388 |
| lage | 0.1399208 | 1 | 243.094594 | 0.0000000 |
| Residuals | 0.0495000 | 86 | NA | NA |

```
summary(model.ancova.nooutliers)
```

Call:

```
lm(formula = lfkl ~ sex + lage, data = anc1dat[c(-49, -50, -19),
])
```

Residuals:

| Min | 1Q | Median | 3Q | Max |
|-----------|-----------|-----------|----------|----------|
| -0.058397 | -0.018469 | -0.000976 | 0.020696 | 0.040288 |

Coefficients:

```

              Estimate Std. Error t value Pr(>|t|)
(Intercept) 1.224000  0.028106 43.549 <2e-16 ***
sexMALE     -0.010823  0.005386 -2.009  0.0476 *
lage        0.328604  0.021076 15.591 <2e-16 ***
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

Residual standard error: 0.02399 on 86 degrees of freedom
 Multiple R-squared: 0.7706, Adjusted R-squared: 0.7653
 F-statistic: 144.4 on 2 and 86 DF, p-value: < 2.2e-16

Ouch! Les résultats changent.

Il faudrait donc rejeter l'hypothèse nulle et conclure que les ordonnées à l'origine diffèrent! Une conclusion qualitativement différente de celle obtenue en considérant toutes les données.

Pourquoi? Il y a deux raisons possibles :

- (1) Les valeurs extrêmes influencent beaucoup les régressions ou,
- (2) L'exclusion des valeurs extrêmes permet d'augmenter la puissance de détection d'une différence.

La première explication est moins plausible parce que les valeurs extrêmes n'avaient pas une grande influence (leverage faible). Alors, la deuxième explication est plus plausible et vous pouvez le vérifier en faisant des régressions pour chaque sexe sans et avec les valeurs extrêmes. Si vous le faites, vous noterez que les ordonnées à l'origine pour chaque sexe ne changent presque pas alors que leurs erreurs-types changent beaucoup.

🔥 Exercice

Ajustez une régression simple entre `lfkl` et `lage` pour l'ensemble complet de données et aussi pour le sous-ensemble sans les 3 valeurs déviante. Comparez ces modèles avec les modèles d'ANCOVA ajustés précédemment. Que concluez-vous ? Quel modèle, d'après vous, a le meilleur ajustement aux données ? Pourquoi ?

Le modèle en excluant les valeurs extrêmes:

```

model.linear.nooutliers<-lm(lfkl ~ lage,data = anc1dat[c(-49, -50, -19),])
summary(model.linear.nooutliers)

```

Call:

```
lm(formula = lfk1 ~ lage, data = anc1dat[c(-49, -50, -19), ])
```

Residuals:

| Min | 1Q | Median | 3Q | Max |
|-----------|-----------|-----------|----------|----------|
| -0.055567 | -0.017809 | -0.002944 | 0.021272 | 0.044972 |

Coefficients:

| | Estimate | Std. Error | t value | Pr(> t) | | | |
|----------------|----------|------------|------------|------------|----------|---------|---|
| (Intercept) | 1.20378 | 0.02670 | 45.09 | <2e-16 *** | | | |
| lage | 0.34075 | 0.02054 | 16.59 | <2e-16 *** | | | |
| --- | | | | | | | |
| Signif. codes: | 0 | '***' | 0.001 '**' | 0.01 '*' | 0.05 '.' | 0.1 ' ' | 1 |

Residual standard error: 0.02441 on 87 degrees of freedom

Multiple R-squared: 0.7598, Adjusted R-squared: 0.7571

F-statistic: 275.2 on 1 and 87 DF, p-value: < 2.2e-16

Pour la régression simple (sans les valeurs extrêmes) on obtient un R^2 de 0.76 et une erreur-type des résidus de 0.02441, En comparant à l'erreur-type des résidus du modèle d'ANCOVA (0.02399) on réalise que la qualité des prédictions est essentiellement la même, même en ajustant des ordonnées à l'origine différentes pour chaque groupe. Par conséquent, les bénéfices de l'inclusion d'un terme pour les différentes ordonnées à l'origine sont faibles alors que le coût, en terme de complexité du modèle, est élevé (33% d'augmentation du nombre de termes pour un très faible amélioration de la qualité d'ajustement). Si vous examinez les résidus de ce modèle, vous trouverez qu'ils sont à peu près OK.

Si on ajuste une régression simple sur toutes les données, on obtient:

```
model.linear<-lm(lfk1 ~ lage, data = anc1dat)
summary(model.linear)
```

Call:

```
lm(formula = lfk1 ~ lage, data = anc1dat)
```

Residuals:

| Min | 1Q | Median | 3Q | Max |
|-----------|-----------|-----------|----------|----------|
| -0.090915 | -0.018975 | -0.002587 | 0.021270 | 0.085273 |

Coefficients:

| | Estimate | Std. Error | t value | Pr(> t) |
|----------------|----------|------------|----------|------------|
| (Intercept) | 1.21064 | 0.03089 | 39.19 | <2e-16 *** |
| lage | 0.33606 | 0.02376 | 14.14 | <2e-16 *** |
| --- | | | | |
| Signif. codes: | 0 '***' | 0.001 '**' | 0.01 '*' | 0.05 '.' |
| | 0.1 ' | 1 | | |

Residual standard error: 0.0285 on 90 degrees of freedom

Multiple R-squared: 0.6897, Adjusted R-squared: 0.6863

F-statistic: 200.1 on 1 and 90 DF, p-value: < 2.2e-16

Encore une fois, l'erreur-type des résidus (0.0285) pour cette régression unique est semblable à la variance du modèle d'ANCOVA (0.02837) et le modèle simplifié prédit presque aussi bien que le modèle plus complexe. Ici encore, toutes les conditions d'application semblent remplies, si ce n'est de la valeur extrême.

Donc, dans les deux cas (avec ou sans les valeurs extrêmes), l'addition d'un terme supplémentaire pour le sexe n'ajoute pas grand-chose. Il semble donc que le meilleur modèle soit celui de la régression simple. Un estimé raisonnablement précis de la taille des esturgeons peut être obtenu de la régression commune sur l'ensemble des résultats.

Note: Il est fréquent que l'élimination de valeurs extrêmes en fasse apparaître d'autres. C'est parce que ces valeurs extrêmes dépendent de la variabilité résiduelle. Si on élimine les valeurs les plus déviantes, la variabilité résiduelle diminue, et certaines observations qui n'étaient pas si déviantes que cela deviennent proportionnellement plus déviantes. Notez aussi qu'en éliminant des valeurs extrêmes, l'effectif diminue et que la puissance décroît. Il faut donc être prudent.

14.6. Comparer l'ajustement de modèles

Comme vous venez de le voir, le processus d'ajustement de modèles est itératif. La plupart du temps il y a plus d'un modèle qui peut être ajusté aux données et c'est à vous de choisir celui qui est le meilleur compromis entre la **qualité**

d'ajustement (qu'on essaie de maximiser) et la **complexité** (qu'on essaie de minimiser). La stratégie de base en ajustant des modèles linéaires (ANOVA, régression, ANCOVA) est de **privilégier le modèle le plus simple si la qualité d'ajustement n'est pas significativement plus mauvaise**. R peut calculer une statistique F vous permettant de comparer l'ajustement de deux modèles. Dans ce cas, l'hypothèse nulle est que la qualité d'ajustement ne diffère pas entre les deux modèles.

🔥 Exercice

En utilisant les données de `anc1dat` comparez l'ajustement du modèle ANCOVA et de la régression commune:

```
anova(model.ancova,model.linear)
```

| Res.Df | RSS | Df | Sum of Sq | F | Pr(>F) |
|--------|-----------|----|------------|----------|-----------|
| 89 | 0.0716231 | NA | NA | NA | NA |
| 90 | 0.0731130 | -1 | -0.0014899 | 1.851324 | 0.1770653 |

La fonction `anova()` utilise la différence entre la somme des carrés des deux modèles et la divise par la différence entre le nombre de degrés de liberté pour obtenir un carré moyen. Ce carré moyen est utilisé au numérateur et est divisé par la variance résiduelle du modèle le plus complexe pour obtenir la statistique F. Dans ce cas-ci, le test de F n'est pas significatif, et on conclut que les deux modèles ont une qualité d'ajustement équivalente, et qu'on devrait donc privilégier le modèle le plus simple, la régression linéaire simple.

🔥 Exercice

Refaites le même processus avec le données de `anc3dat`, ajustez le modèle complet avec interaction (`LFKL~LAGE+LOCATE+LAGE:LOCATE`) et sans interaction (`LFKL~LAGE+LOCATE`), Comparez l'ajustement des deux modèles, que concluez vous?

💡 Solution

```
model.full.anc3dat<-lm(lfkl ~ lage + locate + lage:locate, data = anc3dat)
model.ancova.anc3dat<-lm(lfkl ~ lage + locate, data = anc3dat)
anova(model.full.anc3dat,model.ancova.anc3dat)
```

| Res.Df | RSS | Df | Sum of Sq | F | Pr(>F) |
|--------|-----------|----|------------|----------|-----------|
| 88 | 0.0513582 | NA | NA | NA | NA |
| 89 | 0.0604482 | -1 | -0.0090901 | 15.57541 | 0.0001592 |

Cette fois-ci, le modèle plus complexe s'ajuste significativement mieux aux données (Pas surprenant puisque nous avions précédemment conclu que l'interaction est significative avec ces données).

14.7. Bootstrap

```
#####
#####
# Analyses Bootstrap
# Analyses Bootstrap intervalles de confiance BCa
# Préférable quand il y a un fort écart à la normalité
# Bootstrap 95% BCa IC pour les coefficients de la régression
library(boot)

# Pour simplifier des futures modification du code dans ce fichier,
# assignez les données à un objet mesdonnees
mesdonnees <- anc3dat

# créez une variable maformule contenant la formule du modèle à ajuster
maformule <- as.formula(lfkl ~ lage + locate + lage:locate)

# fonction pour obtenir les coefficients de régression à chaque itération
bs <- function(formula, data, indices) {
  d <- data[indices, ]
  fit <- lm(formula, data = d)
  return(coef(fit))
}
# bootstrap avec 1000 réplications
```

```
resultats <- boot(data = mesdonnees, statistic = bs, R = 1000, formula = maformule)

# afficher les résultats
resultats
boot_res <- summary(resultats)
rownames(boot_res) <- names(resultats$t0)
boot_res

op <- par(ask = TRUE)
for (i in 1:length(resultats$t0)) {
  plot(resultats, index = i)
  title(names(resultats$t0)[i])
}
par(op)

# get 95% confidence intervals
for (i in 1:length(resultats$t0)) {
  cat("\n", names(resultats$t0)[i], "\n")
  print(boot.ci(resultats, type = "bca", index = i))
}
```

14.8. Test de Permutation

```
#####
#####
# Test de Permutation
#
# en utilisant le paquet `lmperm`
# Pour simplifier des futures modification du code dans ce fichier,
# assignez les données à un objet mesdonnees
mesdonnees<-anc3dat
# créez une variable maformule contenant la formule du modèle à ajuster
```

```
maformule<-as.formula(lfkl ~ lage + locate + lage:locate)
require(lmPerm2)
# Ajustez le modèle voulu sur les données voulues
monmodele <- lm(maformule, data = mesdonnees)
# Calculez la valeur de p pour chaque termes à chaque permutation
# Notez que lmp centre les variables numériques par défaut,
# donc pour avoir des résultats consistants avec les modèles standards,
# il faut :
center=FALSE
monmodeleProb <- lmp(maformule, data = mesdonnees, center=FALSE,
perm = "Prob")
summary(monmodele)
summary(monmodeleProb)
```

partie IV.

Modèles linéaires généralisés

Chapitre 15

Analyse de données de fréquence: Tableaux de contingence, modèles log-linéaires et régression de Poisson

Après avoir complété ce laboratoire, vous devriez être en mesure de:

- Créer et manipuler des jeux de données en R pour analyser des données de fréquences.
- Utiliser R pour éprouver une hypothèse extrinsèque à propos de données de fréquence d'une population.
- Utiliser R pour éprouver l'hypothèse d'indépendance pour des tableaux de contingence à 2 dimensions.
- Utiliser R pour ajuster des régressions de Poisson et des modèles log-linéaires à des données de fréquence.

15.1. Paquets et données requises pour le labo

Ce laboratoire nécessite:

- les paquets R:
 - vcd
 - vcdExtra
 - car
- les fichiers de données
 - USPopSurvey.csv
 - loglin.csv
 - sturgdat.csv

15.2. Organisation des données: 3 formats

Les résultats de certaines expériences sont sous forme de fréquences, par exemple le nombre de plantes infectées par un pathogène sous différents régimes d'infection, ou le nombre de tortues mâles et femelles qui éclosent à différentes températures (oui, chez les tortues le sexe dépend de la température!), etc. La question statistique qui se pose généralement est de savoir si la proportion des observations dans chaque catégorie (infecté vs non infecté, mâle vs femelle, etc) diffère significativement entre les traitements (régime d'infection ou température dans les deux exemples). Pour répondre à cette question, on peut organiser les données de manière à refléter comment les observations se retrouvent dans chaque catégorie. Il existe 3 façons d'organiser ces données. Vous devriez être capable de choisir la manière appropriée pour votre analyse, et savoir convertir entre elles avec R.

Le fichier USPopSurvey.csv contient les données de recensement d'une ville du midwest américain en 1980:

```
USPopSurvey <- read.csv("data/USPopSurvey.csv")  
USPopSurvey
```

| ageclass | sex | frequency |
|----------|--------|-----------|
| 0-9 | female | 17619 |
| 10-19 | female | 17947 |
| 20-29 | female | 21344 |
| 30-39 | female | 19138 |
| 40-49 | female | 13135 |
| 50-59 | female | 11617 |
| 60-69 | female | 11053 |
| 70-79 | female | 7712 |
| 80+ | female | 4114 |
| 0-9 | male | 17538 |
| 10-19 | male | 18207 |
| 20-29 | male | 21401 |
| 30-39 | male | 18837 |
| 40-49 | male | 12568 |
| 50-59 | male | 10661 |
| 60-69 | male | 9374 |
| 70-79 | male | 5348 |

| ageclass | sex | frequency |
|----------|------|-----------|
| 80+ | male | 1926 |

Notez qu'il y a 18 lignes et 3 colonnes dans ce fichier. Chaque ligne donne le nombre de personnes (frequency) pour un sexe et une classe d'âge. Il y a 239539 individus qui ont été classifiés selon les 18 catégories (2 sexes x 9 classes d'âge). Cette manière de représenter les données est sous **le format de fréquences** (frequency form). C'est un format compact permettant d'enregistrer les données quand il y a seulement des variables catégoriques à représenter.

Lorsqu'il y a des variables continues, ce format ne peut être utilisé. Les données doivent être enregistrée sous **le format de cas** (case form) dans laquelle chaque observation (individu) est représenté par une ligne dans le fichier, et où chaque variable est représentée par une colonne. Le package `vcdExtra` contient la fonction `expand.dft()` qui permet de convertir de la forme de fréquence à la forme de cas. Par exemple, pour créer un data frame avec 239439 lignes et 2 colonnes (`sex` et `ageclass`) à partir du data frame `USPopSurvey`:

```
USPopSurvey.caseform <- expand.dft(USPopSurvey, freq = "frequency")
head(USPopSurvey.caseform)
```

| ageclass | sex |
|----------|--------|
| 0-9 | female |

```
tail(USPopSurvey.caseform)
```

| | ageclass | sex |
|--------|----------|------|
| 239534 | 80+ | male |
| 239535 | 80+ | male |
| 239536 | 80+ | male |
| 239537 | 80+ | male |

| | ageclass | sex |
|--------|----------|------|
| 239538 | 80+ | male |
| 239539 | 80+ | male |

Ces données peuvent finalement être organisées sous **le format de tableau** (table form) de contingence où chacune des n variables est représentée par une dimension d'un tableau n-dimensionnel (dans notre exemple on a 2 variables, sexe et classe d'âge, et les rangées pourraient représenter les classes d'âge et les colonnes chaque sexe). Les cellules de ce tableau contiennent les fréquences. Le format tableau peut être obtenu du format de fréquence ou de cas par la commande `xtabs()` :

```
# convert case form to table form
xtabs(~ ageclass + sex, USPopSurvey.caseform)
```

```
sex
ageclass female male
0-9      17619 17538
10-19    17947 18207
20-29    21344 21401
30-39    19138 18837
40-49    13135 12568
50-59    11617 10661
60-69    11053  9374
70-79    7712   5348
80+      4114   1926

# convert frequency form to table form
xtabs(frequency ~ ageclass + sex, data = USPopSurvey)
```

```
sex
ageclass female male
0-9      17619 17538
10-19    17947 18207
20-29    21344 21401
```

| 30-39 | 19138 | 18837 |
|-------|-------|-------|
| 40-49 | 13135 | 12568 |
| 50-59 | 11617 | 10661 |
| 60-69 | 11053 | 9374 |
| 70-79 | 7712 | 5348 |
| 80+ | 4114 | 1926 |

Table 15.4.: (#tab:unnamed-chunk-1) Fonctions permettant la conversion de données de fréquences entre les différents formats.

| De (ligne) \ Vers (colonne) | Cas | Fréquence | Tableau |
|-----------------------------|---------------|------------------|----------------------|
| Cas | | xtabs(~ A + B) | table(A, B) |
| Fréquence | expand.dft(X) | | xtabs(count ~ A + B) |
| Tableau | expand.dft(X) | as.data.frame(X) | |

15.3. Visualiser graphiquement les tableaux de contingence et test d'indépendance

Les tableaux de contingence peuvent servir à éprouver l'hypothèse d'indépendance des observations. Ceci équivaut à répondre à la question: est-ce que la classification des observations selon une variable (par exemple sex) indépendante de la classification par une autre variable (par exemple ageclass). En autres mots, est-ce que la proportion des mâles et femelles indépendante de l'âge ou varie avec l'âge?

Le package vcd inclut la fonction `mosaic()` qui permet de représenter graphiquement le contenu d'un tableau de contingence:

```
library(vcd)
USTable <- xtabs(frequency ~ ageclass + sex, data = USPopSurvey) # save the table form as USTable
# Mosaic plot of the contingency table
mosaic(USTable)
```

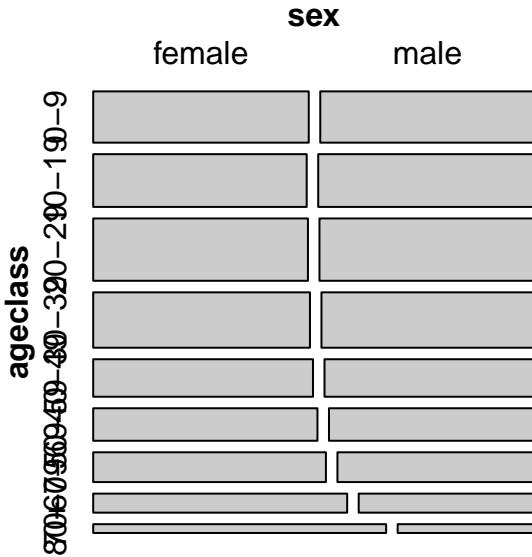


Figure 15.1.: Représentation mosaique de la proportion des sexes par classe d'âge

Cette mosaïque représente la proportion des observations dans chaque combinaison de catégories (ici il y a 18 catégories, 2 sexes x 9 classes d'âge). Les catégories contenant une plus grande proportion d'observations sont représentées par de plus grands rectangles. Visuellement, on peut voir que la proportion des mâles et femelles est approximativement égale chez les jeunes, mais que la proportion des femelles augmente chez les personnes âgées.

Le test de Chi carré permet d'éprouver l'hypothèse nulle que la proportion des mâles et femelles ne change pas avec l'âge (est indépendante de l'âge):

```
# Test of independence  
chisq.test(USTable) # runs chi square test of independence of sex and age class
```

Pearson's Chi-squared test

```
data: USTable  
X-squared = 1162.6, df = 8, p-value < 2.2e-16
```

La valeur p étant très faible, on rejette donc l'hypothèse nulle que âge et sexe sont indépendants. Ces graphiques mosaïques peuvent être colorés pour souligner les catégories qui contribuent le plus à cette dépendance:

```
# Mosaic plot of the contingency table with shading
mosaic(USTable, shade = TRUE)
```

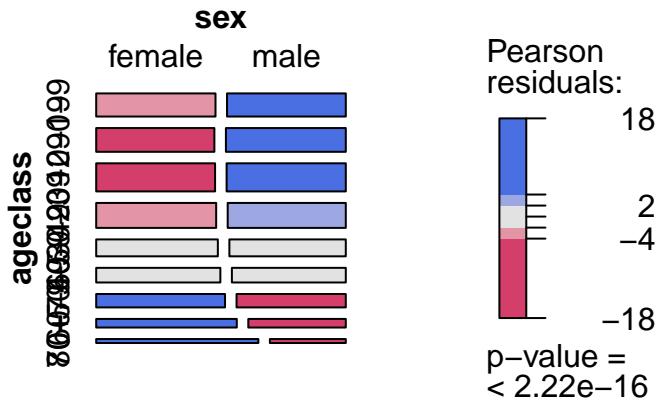


Figure 15.2.: Représentation mosaique de la proportion des sexes par classe d'âge avec échelle de couleur

La couleur de chaque rectangle est proportionnelle à la déviation des fréquences observées de ce qui serait attendu si l'âge et le sexe étaient indépendants. Les classes d'âge 40-49 et 50-59 ont un rapport des sexe approximativement égal à celui de toutes les classes d'âge réunies. Il y a plus de jeunes mâles et de femelles âgées que si le rapport des sexe ne variait pas avec l'âge et ces rectangles sont colorés en bleu. De l'autre côté, il y a moins de jeunes femelles et de mâles âgés que si le rapport des sexe était indépendant de l'âge, et ces rectangles sont en rouge. La valeur p à la droite de la figure est pour le test de Chi carré qui éprouve l'hypothèse nulle d'indépendance pour l'ensemble des observations, toutes classes d'âge confondues.

L'estimation de la valeur p associée à la statistique du Chi carré est approximative lorsque les fréquences attendues sont faibles dans certaines cellules, et ce particulièrement pour les tableaux de contingence 2x2. Deux options permettant des valeurs p plus exactes sont préférées dans ce cas, et le choix dépend du nombre total d'observations. Pour de grands échantillons (comme ici avec plus de 200,000 observations!), une approche par simulation de type Monte Carlo est suggérée et peut être obtenue en ajoutant `simulate.p.value=TRUE` comme argument à la fonction `chisq.test()`:

```
# Monte-carlo estimation of p value (better for small n)
chisq.test(USTable, simulate.p.value = TRUE, B = 10000)
```

```
Pearson's Chi-squared test with simulated p-value (based on 10000
replicates)
```

```
data: USTable
X-squared = 1162.6, df = NA, p-value = 9.999e-05
```

Ici, la simulation a été faite B=10000 fois, et la valeur de Chi carré observée avec les données réelles n'a jamais été observée. Par conséquent, p a été estimé à 1/10001=9.999e-05, qui est beaucoup plus élevé que la valeur p estimée à partir de la distribution théorique de Chi carré ($p < 2.2e-16$). Cette différence est due au moins en partie à un artefacts de la simulation. Pour obtenir des valeurs p de l'ordre de 1e-16, il faut effectuer au moins 10^{16} simulations. Et je ne suis pas aussi patient que ça!

Pour de petits tableaux de contingence avec des fréquences attendues petites, le test exact de Fisher peut servir à estimer la valeur p associée à l'hypothèse d'indépendance. Mais ce test ne peut être effectué avec de grands échantillons, comme ici:

```
# Fisher exact test for contingency tables (small samples and small tables)
fisher.test(USTable) # fails here because too many observations
```

```
Error in fisher.test(USTable): FEXACT error 40.
```

```
Out of workspace.
```

```
fisher.test(USTable, simulate.p.value = TRUE, B = 10000)
```

```
Fisher's Exact Test for Count Data with simulated p-value (based on
10000 replicates)
```

```
data: USTable
p-value = 9.999e-05
alternative hypothesis: two.sided
```

15.4. Régression de Poisson: une alternative au test de Chi carré pour les tableaux de contingence

Rendu à ce stade, vous devriez avoir appris à apprécier la flexibilité et la généralité des modèles linéaires, et réaliser que le test de t est un cas spécial d'un modèle linéaire avec une variable indépendante catégorique. L'analyse des tableaux de contingence par le test du Chi carré peut également être généralisé. Un modèle linéaire généralisé pour une distribution de Poisson peut être utilisé quand la variable dépendante est une fréquence d'observations et les variables indépendantes sont catégorique (comme pour les tableaux de contingence, on parle alors de modèles log-linéaires), continue (régression Poisson), ou une combinaison de variables indépendante continues et catégoriques (aussi appelé régression de Poisson, mais avec des variables catégoriques en plus, analogue à l'ANCOVA sensu largo).

Ces modèles prédisent le logarithme naturel de la fréquence des observations en fonction des variables indépendantes. Comme pour les modèles linéaires qui présument de la normalité des résidus, on peut évaluer la qualité d'ajustement du modèle (par AICc par exemple) et la signification statistique des termes du modèle (par exemple en comparant l'ajustement d'un modèle "complet" et celui d'un modèle qui exclue un terme à tester). On peut également obtenir des estimés des paramètre pour chaque terme dans le modèle, avec des intervalles de confiance et des valeur p pour l'hypothèse nulle que ce terme n'a pas d'influence sur la fréquence.

La fonction `glm()` avec l'option `family=poisson()` permet l'estimation, par la méthode du maximum de vraisemblance, de modèles linéaires pour des fréquences. Comparativement aux modèles linéaires vus précédemment, une des particularité de ces modèles est que seuls les termes d'interaction sont d'intérêt. En partant des données de recensement en forme tableau, on peut ajuster un `glm` aux fréquences observées par sexe et classe d'âge par:

```
mymodel <- glm(frequency ~ sex * ageclass, family = poisson(), data = USPopSurvey)
summary(mymodel)
```

Call:

```
glm(formula = frequency ~ sex * ageclass, family = poisson(),
  data = USPopSurvey)
```

Coefficients:

| | Estimate | Std. Error | z value | Pr(> z) |
|-------------|-----------|------------|----------|-------------|
| (Intercept) | 9.776733 | 0.007534 | 1297.730 | < 2e-16 *** |
| sexmale | -0.004608 | 0.010667 | -0.432 | 0.6657 |

```

ageclass10-19      0.018445  0.010605   1.739  0.0820 .
ageclass20-29      0.191793  0.010179  18.842 < 2e-16 ***
ageclass30-39      0.082698  0.010441   7.921 2.36e-15 ***
ageclass40-49     -0.293697  0.011528 -25.477 < 2e-16 ***
ageclass50-59     -0.416508  0.011951 -34.850 < 2e-16 ***
ageclass60-69     -0.466276  0.012134 -38.428 < 2e-16 ***
ageclass70-79     -0.826200  0.013654 -60.511 < 2e-16 ***
ageclass80+       -1.454582  0.017316 -84.004 < 2e-16 ***
sexmale:ageclass10-19 0.018991  0.014981   1.268  0.2049
sexmale:ageclass20-29 0.007275  0.014400   0.505  0.6134
sexmale:ageclass30-39 -0.011245  0.014803  -0.760  0.4475
sexmale:ageclass40-49 -0.039519  0.016416  -2.407  0.0161 *
sexmale:ageclass50-59 -0.081269  0.017136  -4.742 2.11e-06 ***
sexmale:ageclass60-69 -0.160154  0.017633  -9.083 < 2e-16 ***
sexmale:ageclass70-79 -0.361447  0.020747 -17.422 < 2e-16 ***
sexmale:ageclass80+   -0.754343  0.029598 -25.486 < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

(Dispersion parameter for poisson family taken to be 1)

```

Null deviance: 5.3611e+04  on 17  degrees of freedom
Residual deviance: 6.5463e-12  on  0  degrees of freedom
AIC: 237.31

```

Number of Fisher Scoring iterations: 2

L'ajustement du modèle complet, avec L'interaction triple sex:ageclass interaction, permet à la proportion des mâles et femelles de changer entre les classes d'âge, et donc d'estimer exactement les fréquences observées pour chaque combinaison de sexe et classe d'âge (notez que les résidus (deviance residuals) sont tous 0, et que l'estimé de déviance résiduelle est également approximativement zéro).

Un masochiste peut utiliser le tableau des coefficients pour obtenir la fréquence prédictive pour les différentes catégories. Les fréquences prédictives, comme pour l'ANOVA à critères multiple, sont obtenus en additionnant les coefficients

appropriés. Puisque, en R, le premier niveau d'une variable catégorique (facteur) en ordre alphabétique) est utilisé comme référence, l'ordonnée à l'origine (9.776733) est la valeur prédite pour le logarithme naturel de la fréquence des femelles dans la première classe d'âge (0 to 9). En effet, 9.776733 est approximativement égal à 17619, le nombre observé de femelles dans cette classe d'âge.

Pour les mâles dans la classe d'âge 80+, il faut calculer l'antilog du coefficient pour l'ordonnée à l'origine (pour les femelles dans la première classe d'âge), plus le coefficient pour sexmale (égal à la différence du log de la fréquence entre les femelles et les mâles), plus le coefficient pour la classe d'âge 80+ qui correspond à la différence de fréquence entre cette classe d'âge et la classe d'âge de référence, plus le coefficient pour l'interaction sexmale:ageclass80+ (qui correspond à la différence de proportion de mâles dans cette classe d'âge par rapport à la classe d'âge de référence). Ceci donne: $\ln(\text{frequency}) = 9.776733 - 0.004608 - 1.454582 - 0.754343 = 7.5632$, et la fréquence est égale à $e^{7.5632} = 1926$

Il y a de nombreuses valeurs p dans ce tableau, mais elles ne sont en général pas très utiles. Pour éprouver l'hypothèse que l'effet du sexe sur la fréquence est identique dans chaque classe d'âge (i.e. que sexe et âge sont indépendants), vous devez ajuster un modèle qui exclut cette interaction (sex:ageclass) et déterminer comment l'ajustement du modèle est affecté.

La fonction Anova() du package car permet de prendre un raccourci:

```
Anova(mymodel, type = 3, test = "LR")
```

| | LR Chisq | Df | Pr(>Chisq) |
|--------------|--------------|----|------------|
| sex | 1.866202e-01 | 1 | 0.6657446 |
| ageclass | 2.107464e+04 | 8 | 0.0000000 |
| sex:ageclass | 1.182152e+03 | 8 | 0.0000000 |

Les arguments type=3 and test="LR" font en sorte que le test effectué pour comparer le modèle complet aux modèles réduits est le test de Chi carré sur le rapport de vraisemblance (Likelihood Ratio Chi-Square) à partir de la variance résiduelle, et que c'est un test partiel, et non séquentiel.

Selon ces tests, il n'y a pas d'effet principal de sex ($p=0.667$) mais il y a un effet principal de ageclass et une interaction significative sex:ageclass. L'interaction significative signifie que l'effet du sexe sur la fréquence varie selon les classes d'âge, bref que le rapport des sexes varie avec l'âge. L'effet principal de ageclass signifie que la fréquence des individus varie avec l'âge dans la population recensée (i.e. que certaines classes d'âge sont plus populaires que d'autres). L'absence d'un effet principal du sexe suggère qu'il y a approximativement le même nombre de mâles et femelles

dans l'échantillon (quoique, puisqu'il y a une interaction, vous devez être prudents en faisant cette déclaration. C'est "vrai" au total, mais semble incorrect pour certaines classes d'âge).

15.5. Tester une hypothèse extrinsèque

Le test d'indépendance ci-dessus éprouve une hypothèse intrinsèque parce que les proportions utilisées pour calculer les valeurs attendues et tester l'indépendance sont celles observées (i.e. la proportion des mâles et femelles dans tout l'échantillon, et la proportion des individus dans chaque classe d'âge).

Pour éprouver l'hypothèse (extrinsèque) que le rapport des sexes est 1:1 pour les individus les plus jeunes (ageclass 0-9), on doit produire le tableau 2X2 des fréquences observées et attendues. Les fréquences attendues sont obtenues simplement en divisant le total des mâles et femelles par 2.

Code R pour créer et analyser un tableau de contingence 2X2 et éprouver une hypothèse extrinsèque

```
### Produce a table of obs vs exp for 0-9 age class
Popn0.9 <- rbind(c(17578, 17578), c(17619, 17538))
### Run X2 test on above table
chisq.test(Popn0.9, correct = F) ### X2 without Yates
chisq.test(Popn0.9) ### X2 with Yates
```

🔥 Exercice

Éprouvez l'hypothèse nulle que la proportion de mâles et femelles à la naissance est égale. Que concluez-vous?
Croyez-vous que ces données sont appropriées pour tester cette hypothèse?

💡 Solution

```
chisq.test(Popn0.9, correct = F)
```

Pearson's Chi-squared test

```
data: Popn0.9
X-squared = 0.093309, df = 1, p-value = 0.76
```

```
chisq.test(Popn0.9)
```

```
Pearson's Chi-squared test with Yates' continuity correction

data: Popn0.9
X-squared = 0.088758, df = 1, p-value = 0.7658
```

Notez que pour un tableau 2X2, on devrait utiliser une correction de Yates ou un test de Fisher. Le test de Fisher ne pouvant être utilisé lorsque l'échantillon dépasse 200, on utilise la correction de Yates. Selon cette analyse, on accepte l'hypothèse nulle que le rapport des sexes est 1:1 à la naissance. Ceci dit, ces données ne sont pas très appropriées pour éprouver l'hypothèse car la première classe d'âge est trop grossière. Il est possible que le rapport des sexes à la naissance soit différent de 1:1 mais que la mortalité différentielle des deux sexes compense au cours des 9 premières années (par exemple si il y a plus de mâles à la naissance, mais que les jeunes garçons ont une survie plus faible au cours de leurs 9 premières années). Dans un tel cas, le rapport des sexes n'est PAS de 1:1 à la naissance, mais on accepte l'hypothèse nulle à partir des données dans la classe d'âge 0-9.

15.6. Régression de Poisson pour l'analyse de tableaux de contingence à plusieurs critères

Le principe d'éprouver l'indépendance en examinant les interactions peut être utilisé avec les tableaux de contingence à plusieurs critères. Par exemple, examinons si la température (2 niveaux: base et haute) et l'éclairage (2 niveaux: bas et haut) affectent si des plantes sont infectées (2 niveaux: infecté et non-infecté) par un pathogène. On peut représenter ces données par un tableau de contingence à 3 critères (température, lumière, statut d'infection).

L'ajustement de modèles log-linéaires à des données de fréquence implique que l'on éprouve plusieurs modèles en les comparant au modèle complet (saturé). Une série de modèles contenant tous les termes sauf une des interactions qui nous intéressent est produite, et l'ajustement de chaque modèle est comparé à celui du modèle complet. Si la réduction de la qualité d'ajustement n'est pas significative, cela implique que l'interaction manquante contribue peu à la qualité de l'ajustement. Par contre, si le modèle réduit s'ajuste nettement moins bien aux données, alors l'interaction manquante contribue beaucoup à l'ajustement du modèle complet. Comme pour les tableaux de contingence 2X2, les termes qui nous intéressent le plus sont les interactions, pas les effets principaux, si l'on teste pour l'indépendance des différents facteurs.

Le fichier `loglin.csv` contient les fréquences (frequency) des plantes infectées ou non infectées (`infected`) à basse et haute température (`temperature`) à basse et haute lumière (`light`). Pour visualiser ces données et déterminer si le taux d'infection dépend de la lumière et de la température, on peut faire une figure mosaïque et ajuster un modèle log-linéaire:

```
loglin <- read.csv("data/loglin.csv")  
# Convert from frequency form to table form for mosaic plot  
loglinTable <- xtabs(frequency ~ temperature + light + infected, data = loglin)  
# Create mosaic plot to look at data  
mosaic(loglinTable, shade = TRUE)
```

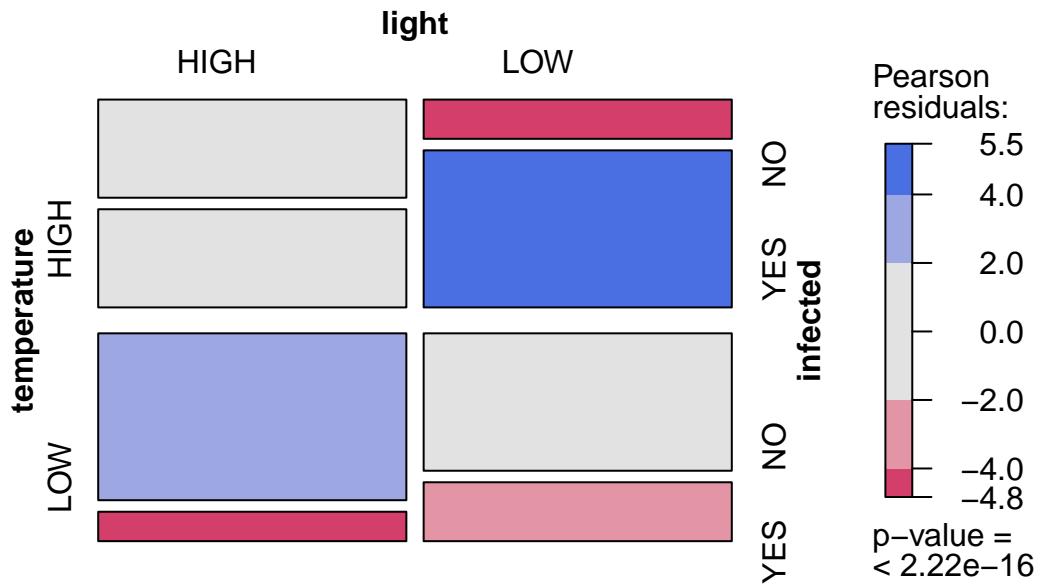


Figure 15.3.: Proportion de plantes infectées en fonction de la température et la lumière

Cette expérience contrôlée avec le même nombre de plantes à chaque niveau de lumière et de température produit une mosaïque où la surface occupée par les observations dans les quatre quadrants est égale. Ce qui nous intéresse, le taux d'infection par le pathogène, semble varier entre les quadrants (i.e. les niveaux de température et de lumière). Le rectangle rouge dans le coin en bas à gauche indique que le nombre de plantes infectées à basse température et haute lumière est plus faible qu'attendu si ces deux facteurs n'influencent pas le taux d'infection. Même chose pour les conditions de basse lumière et de haute température (coin supérieur droit). La valeur p au bas de l'échelle représente un test d'indépendance équivalent à comparer le modèle complet au modèle excluant toutes les interactions et ne contenant que les effets principaux de la température, la lumière, et le statut d'infection sur le logarithme naturel du nombre d'observations.

```
# Fit full model
full.model <- glm(frequency ~ temperature * light * infected, family = poisson(), data = loglin)
# Test partial effect of terms in full model
Anova(full.model, type = 3, test = "LR")
```

| | LR Chisq | Df | Pr(>Chisq) |
|----------------------------|-----------|----|------------|
| temperature | 9.178563 | 1 | 0.0024487 |
| light | 13.282863 | 1 | 0.0002678 |
| infected | 0.000000 | 1 | 0.9999999 |
| temperature:light | 5.675769 | 1 | 0.0172008 |
| temperature:infected | 29.061158 | 1 | 0.0000001 |
| light:infected | 20.268735 | 1 | 0.0000067 |
| temperature:light:infected | 1.083963 | 1 | 0.2978126 |

Les probabilités associées à chaque terme sont ici calculées en comparant l’ajustement du modèle complet à un modèle qui exclue seulement le terme d’intérêt. Plusieurs des termes sont ici sans véritable intérêt puisque les fréquences sont partiellement contrôlées dans notre expérience. Puisque la question biologique porte sur le taux d’infection, les seuls termes d’intérêt sont les termes d’interactions qui incluent le statut d’infection (`temperature:infected`, `light:infected` et `temperature:light:infected`).

- L’interaction significative `temperature:infected` implique que le taux d’infection n’est pas indépendant de la température. D’ailleurs il est apparent dans la mosaïque que le taux d’infection (le nombre relatif de plantes infectées) est supérieur à haute température.
- L’interaction significative `light:infected` implique que le taux d’infection dépend de la lumière. La mosaïque illustre que la proportion des plantes infectées est plus élevée en basse lumière.
- L’interaction `temperature:light:infected` n’est pas significative. Cela implique que l’effet de la température et de la lumière sur le taux d’infection sont indépendants. Autrement dit, l’effet de la lumière sur le taux d’infection ne dépend pas de la température, et vice versa.

15.7. Exercice

Le fichier Sturgdat contient les données qui vous permettront d’éprouver l’hypothèse que le nombre d’esturgeons capturé est indépendants du site, de l’année, et du sexe. Avant de commencer l’analyse, les données devront être

réorganisées pour pouvoir ajuster un modèle log-linéaire:

🔥 Exercice

Ouvrez sturgdat.csv, puis utilisez la fonction table() pour obtenir les fréquence d'individus capturés par sex, location, et year . Sauvegardez ce tableau comme strugdat.table . Faites une figure mosaïque de ces données.

```
sturgdat <- read.csv("data/sturgdat.csv")

# Reorganize data from case form to table form
sturgdat.table <- with(sturgdat, table(sex, year, location))

# display the table
sturgdat.table

, , location = CUMBERLAND

year
sex      1978 1979 1980
FEMALE    10    30    11
MALE      14    14     6

, , location = THE_PAS

year
sex      1978 1979 1980
FEMALE    5    12    38
MALE      16    12    18

# Create data frame while converting from table form to frequency form
sturgdat.freq <- as.data.frame(sturgdat.table)

# display data frame
sturgdat.freq

# Look at the data as mosaic plot
# mosaic using the table created above
mosaic(sturgdat.table, shade = TRUE)
```

| sex | year | location | Freq |
|--------|------|------------|------|
| FEMALE | 1978 | CUMBERLAND | 10 |
| MALE | 1978 | CUMBERLAND | 14 |
| FEMALE | 1979 | CUMBERLAND | 30 |
| MALE | 1979 | CUMBERLAND | 14 |
| FEMALE | 1980 | CUMBERLAND | 11 |
| MALE | 1980 | CUMBERLAND | 6 |
| FEMALE | 1978 | THE_PAS | 5 |
| MALE | 1978 | THE_PAS | 16 |
| FEMALE | 1979 | THE_PAS | 12 |
| MALE | 1979 | THE_PAS | 12 |
| FEMALE | 1980 | THE_PAS | 38 |
| MALE | 1980 | THE_PAS | 18 |

Figure 15.4.: Fréquence de femelles et males en fonction de l'année et du lieu

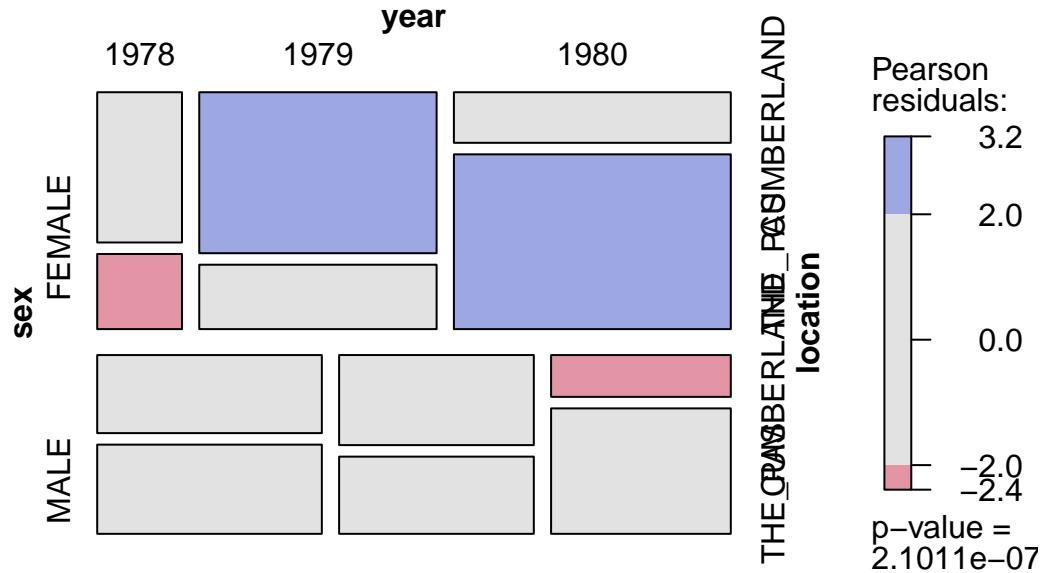


Figure 15.5.: Fréquence de femelles et males en fonction de l'année et du lieu

🔥 Exercice

À partir de ces données en format de fréquence, ajustez le modèle loglinéaire complet et le tableau d'anova avec les statistique de Chi carré pour les termes du modèles. Est-ce que l'interaction triple (location:year:sex) est significative? Est-ce que le rapport des sexes varient entre les sites ou d'une année à l'autre?

```
# Fit full model  
  
full.model <- glm(Freq ~ sex * year * location, data = sturgdat.freq, family = "poisson")  
summary(full.model)
```

Call:

```
glm(formula = Freq ~ sex * year * location, family = "poisson",  
    data = sturgdat.freq)
```

Coefficients:

| | | Estimate | Std. Error | z value |
|-----------------------------------|----------|-------------|------------|---------|
| (Intercept) | | 2.30259 | 0.31623 | 7.281 |
| sexMALE | | 0.33647 | 0.41404 | 0.813 |
| year1979 | | 1.09861 | 0.36515 | 3.009 |
| year1980 | | 0.09531 | 0.43693 | 0.218 |
| locationTHE_PAS | | -0.69315 | 0.54772 | -1.266 |
| sexMALE :year1979 | | -1.09861 | 0.52554 | -2.090 |
| sexMALE :year1980 | | -0.94261 | 0.65498 | -1.439 |
| sexMALE :locationTHE_PAS | | 0.82668 | 0.65873 | 1.255 |
| year1979:locationTHE_PAS | | -0.22314 | 0.64550 | -0.346 |
| year1980:locationTHE_PAS | | 1.93284 | 0.64593 | 2.992 |
| sexMALE :year1979:locationTHE_PAS | | -0.06454 | 0.83986 | -0.077 |
| sexMALE :year1980:locationTHE_PAS | | -0.96776 | 0.87942 | -1.100 |
| | Pr(> z) | | | |
| (Intercept) | | 3.3e-13 *** | | |
| sexMALE | | 0.41641 | | |
| year1979 | | 0.00262 ** | | |
| year1980 | | 0.82732 | | |
| locationTHE_PAS | | 0.20569 | | |
| sexMALE :year1979 | | 0.03658 * | | |
| sexMALE :year1980 | | 0.15011 | | |
| sexMALE :locationTHE_PAS | | 0.20950 | | |
| year1979:locationTHE_PAS | | 0.72957 | | |

```

year1980:locationTHE_PAS           0.00277 ** 
sexMALE      :year1979:locationTHE_PAS    0.93875
sexMALE      :year1980:locationTHE_PAS    0.27114
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

(Dispersion parameter for poisson family taken to be 1)

```

Null deviance: 5.7176e+01 on 11 degrees of freedom
Residual deviance: -2.6645e-15 on 0 degrees of freedom
AIC: 77.28

```

Number of Fisher Scoring iterations: 3

```
Anova(full.model, type = 3)
```

| | LR Chisq | Df | Pr(>Chisq) |
|-------------------|------------|----|------------|
| sex | 0.6697879 | 1 | 0.4131256 |
| year | 13.8894797 | 2 | 0.0009637 |
| location | 1.6989904 | 1 | 0.1924201 |
| sex:year | 4.6930229 | 2 | 0.0957024 |
| sex:location | 1.6322742 | 1 | 0.2013888 |
| year:location | 25.2580075 | 2 | 0.0000033 |
| sex:year:location | 1.6677328 | 2 | 0.4343666 |

Ce tableau a trois critères: sex, location et year . Donc le modèles compelt (saturé) contient 7 termes: trois effets principaux (sex, location et year), trois interactions du second degré (double) (sex:year, sex:location et year: location) et une interaction du troisième degré (triple)(sex:year:location). La déviance nulle est 57.17574, la déviance résiduelle du modèle complet est, sans surprise, 0. La déviance pouvant être attribuée à l'interaction triple est 1.6677, non significative.

Qu'est ce que cela implique? S'il y a des interactions doubles, alors elles ne dépendent pas de la troisième variable. Par exemple, si le rapport des sexe des esturgeons varie d'une année à l'autre (une interaction sex:year), alors cette tendance est la même aux 2 stations.

Puisqu'il n'y a pas d'interaction triple, il est (statistiquement) justifié de combiner les données pour éprouver les interactions du second degré. Par exemple, pour tester l'effet `sex:location`, on peut combiner les années. Pour tester l'effet `sex:year`, on peut combiner les sites. Cette aggrégation a pour effet d'augmenter la puissance, et est analogue à la stratégie en ANOVA à critères multiples. L'approche de la régression de Poisson permet de faire l'équivalent simplement en ajustant le modèle sans l'interaction du troisième degré.

- Ajustez le modèle en excluant l'interaction du troisième degré:

 Solution

```
o2int.model <- glm(Freq ~ sex + year + location + sex:year + sex:location + year:location, data = sturgdat.fr)
Anova(o2int.model, type = 3)
```

| | LR Chisq | Df | Pr(>Chisq) |
|---------------|-----------|----|------------|
| sex | 1.869079 | 1 | 0.1715807 |
| year | 15.128861 | 2 | 0.0005186 |
| location | 1.544449 | 1 | 0.2139568 |
| sex:year | 15.584729 | 2 | 0.0004129 |
| sex:location | 2.176220 | 1 | 0.1401583 |
| year:location | 28.349871 | 2 | 0.0000007 |

L'interaction `sex:location` n'explique pas une portion significative de la déviance, alors que les deux autres sont significatives. Le rapport des sexes ne varie pas entre les sites, mais il varie selon les années. L'interaction `year:location` est aussi significative (voir plus bas pour son interprétation).

Devriez vous tenter de simplifier le modèle encore plus? Les vrais statisticiens sont divisés sur cette question. Tous s'entendent cependant sur le fait que conserver des interactions non significatives dans un modèle peut réduire la puissance. De l'autre côté, le retrait des interactions non significatives peut rendre l'interprétation plus délicate lorsque les observations ne sont pas bien balancées (i.e. il y a de la colinéarité entre les termes du modèle).

- Ajustez le modèle sans l'interaction `sex:location`:

 Solution

```
o2int.model2 <- glm(Freq ~ sex + year + location + sex:year + year:location, data = sturgdat.fr)
Anova(o2int.model2, type = 3)
```

| | LR Chisq | Df | Pr(>Chisq) |
|---------------|------------|----|------------|
| sex | 5.0969711 | 1 | 0.0239677 |
| year | 16.1226325 | 2 | 0.0003155 |
| location | 0.2001484 | 1 | 0.6546011 |
| sex:year | 13.9882526 | 2 | 0.0009173 |
| year:location | 26.7533952 | 2 | 0.0000016 |

Les deux interactions sont significatives et ce modèle semble le meilleur. Ce modèle est:

$$\ln[f_{(ijk)}] = \text{location} + \text{sex} + \text{year} + \text{sex : year} + \text{location : year}$$

Comment ces effets peuvent-ils être interprétés biologiquement? Souvenez vous que, comme dans les test d'indépendance, on n'est pas vraiment intéressé aux effets principaux, seulement par les interactions. Par exemple, l'effet principal de location nous dit que le nombre total d'esturgeons capturé (le total des 2 sexes pendant les 3 années d'échantillonnage) diffère entre les 2 sites. Cela n'est pas vraiment surprenant et peu intéressant en l'absence d'information sur l'effort de pêche. Cependant, l'interaction sex:year nous dit que le rapport des sexes a changé d'une année à l'autre. Et puisque l'interaction du troisième degré n'est pas significative, on sait que ce changement dans le temps est approximativement le même dans les deux sites. Un résultat probablement intéressant. Pourquoi? Comme l'expliquer?

L'interaction `location:year` nous dit que le nombre d'esturgeons n'a pas seulement varié d'une année à l'autre, mais que la tendance dans le temps diffère entre les deux sites. Ceci pourrait refléter une différence d'effort de pêche à un des sites durant l'une des campagnes d'échantillonnage, ou un impact à seulement un des deux sites la dernière année par exemple. Mais cette tendance est la même pour les mâles et les femelles (donc n'a pas affecté le rapport des sexes) puisque l'interaction triple n'est pas significative.

partie V.

Modèles mixtes

partie VI.

Modèles additifs généralisés

partie VII.

Analyses multivariées

partie VIII.

Approche Bayesienne

Références

Paquets R

Ce livre a utilisé les paquets R (excluant leurs dépendances) listé dans le tableau Table 15.10. Comme recommandé par l'équipe de développement de ‘tidyverse’, seul le paquet ‘tidyverse’ est cité et non pas chacun de ses composants.

Table 15.10.: Paquets utilisés dans le livre

| Paquets | Version | Citation |
|-------------|----------|---|
| base | 4.4.1 | R Core Team (2024) |
| boot | 1.3.31 | A. C. Davison et D. V. Hinkley (1997); Angelo Canty et B. D. Ripley (2024) |
| car | 3.1.2 | Fox et Weisberg (2019a) |
| effects | 4.2.2 | Fox (2003); Fox et Hong (2009); Fox et Weisberg (2018); Fox et Weisberg (2019b) |
| emoji | 15.0 | Hvitfeldt (2022) |
| GGally | 2.2.1 | Schloerke et al. (2024) |
| ggcleveland | 0.1.0 | Prunello et Mari (2021) |
| ggpubr | 0.6.0 | Kassambara (2023) |
| grateful | 0.2.10 | Rodriguez-Sanchez et Jackson (2023) |
| gt | 0.11.0 | Iannone et al. (2024) |
| kableExtra | 1.4.0 | Zhu (2024) |
| knitr | 1.48 | Xie (2014); Xie (2015); Xie (2024) |
| lme4 | 1.1.35.5 | Bates et al. (2015) |
| lmPerm | 2.1.0 | Wheeler et Torchiano (2016) |
| lmtest | 0.9.40 | Zeileis et Hothorn (2002) |
| multcomp | 1.4.26 | Hothorn et al. (2008) |

Table 15.10.: Paquets utilisés dans le livre

| Paquets | Version | Citation |
|----------------|---------|---|
| MuMIn | 1.48.4 | Bartoń (2024) |
| palmerpenguins | 0.1.1 | Horst et al. (2020) |
| patchwork | 1.2.0 | Pedersen (2024) |
| performance | 0.12.3 | Lüdecke et al. (2021) |
| pwr | 1.3.0 | Champely (2020) |
| reshape2 | 1.4.4 | Wickham (2007) |
| rmarkdown | 2.28 | Xie et al. (2018); Xie et al. (2020); Allaire et al. (2024) |
| simpleboot | 1.1.8 | Peng (2024) |
| tidyverse | 2.0.0 | Wickham et al. (2019) |
| vcd | 1.4.13 | Meyer et al. (2006); Zeileis et al. (2007); Meyer et al. (2024) |
| vcdExtra | 0.8.5 | Friendly (2023) |
| vioplot | 0.5.0 | Adler et al. (2024) |

Bibliographie

- A. C. Davison, et D. V. Hinkley. 1997. [Bootstrap Methods and Their Applications](#). Cambridge University Press, Cambridge.
- Adler, D., S. T. Kelly, T. Elliott, et J. Adamson. 2024. [vioplot: violin plot](#).
- Allaire, J., Y. Xie, C. Dervieux, J. McPherson, J. Luraschi, K. Ushey, A. Atkins, H. Wickham, J. Cheng, W. Chang, et R. Iannone. 2024. [rmarkdown: Dynamic Documents for R](#).
- Angelo Canty, et B. D. Ripley. 2024. [boot: Bootstrap R \(S-Plus\) Functions](#).
- Bartoń, K. 2024. [MuMIn: Multi-Model Inference](#).
- Bates, D., M. Mächler, B. Bolker, et S. Walker. 2015. [Fitting Linear Mixed-Effects Models Using lme4](#). Journal of Statistical Software 67:1-48.
- Champely, S. 2020. [pwr: Basic Functions for Power Analysis](#).
- Douglas, A. 2023. [An introduction to R](#).
- Fox, J. 2003. [Effect Displays in R for Generalised Linear Models](#). Journal of Statistical Software 8:1-27.
- Fox, J., et J. Hong. 2009. [Effect Displays in R for Multinomial and Proportional-Odds Logit Models: Extensions to the effects Package](#). Journal of Statistical Software 32:1-24.

- Fox, J., et S. Weisberg. 2018. [Visualizing Fit and Lack of Fit in Complex Regression Models with Predictor Effect Plots and Partial Residuals](#). Journal of Statistical Software 87:1-27.
- Fox, J., et S. Weisberg. 2019a. [An R Companion to Applied Regression](#). Third. Sage, Thousand Oaks CA.
- Fox, J., et S. Weisberg. 2019b. [An R Companion to Applied Regression](#). 3rd édition. Sage, Thousand Oaks CA.
- Friendly, M. 2023. [vcdExtra: « vcd » Extensions and Additions](#).
- Horst, A. M., A. P. Hill, et K. B. Gorman. 2020. [palmerpenguins: Palmer Archipelago \(Antarctica\) penguin data](#).
- Hothorn, T., F. Bretz, et P. Westfall. 2008. Simultaneous Inference in General Parametric Models. Biometrical Journal 50:346-363.
- Hvitfeldt, E. 2022. [emoji: Data and Function to Work with Emojis](#).
- Iannone, R., J. Cheng, B. Schloerke, E. Hughes, A. Lauer, J. Seo, K. Brevoort, et O. Roy. 2024. [gt: Easily Create Presentation-Ready Display Tables](#).
- Kassambara, A. 2023. [ggpubr: « ggplot2 » Based Publication Ready Plots](#).
- Lüdecke, D., M. S. Ben-Shachar, I. Patil, P. Waggoner, et D. Makowski. 2021. [performance: An R Package for Assessment, Comparison and Testing of Statistical Models](#). Journal of Open Source Software 6:3139.
- Martin, J. 1219. Another lasagna recipe from Medieval times. Journal of Lasagna 4:1686.
- Martin, J. 2200. A silly example. Chapman; Hall/CRC, Boca Raton, Florida.
- Meyer, D., A. Zeileis, et K. Hornik. 2006. [The Strucplot Framework: Visualizing Multi-Way Contingency Tables with vcd](#). Journal of Statistical Software 17:1-48.
- Meyer, D., A. Zeileis, K. Hornik, et M. Friendly. 2024. [vcd: Visualizing Categorical Data](#).
- Pedersen, T. L. 2024. [patchwork: The Composer of Plots](#).
- Peng, R. D. 2024. [simpleboot: Simple Bootstrap Routines](#).
- Prunello, M., et G. Mari. 2021. [ggcleveland: Implementation of Plots from Cleveland's Visualizing Data Book](#).
- R Core Team. 2024. [R: A Language and Environment for Statistical Computing](#). R Foundation for Statistical Computing, Vienna, Austria.
- Rodriguez-Sanchez, F., et C. P. Jackson. 2023. [grateful: Facilitate citation of R packages](#).
- Schloerke, B., D. Cook, J. Larmarange, F. Briatte, M. Marbach, E. Thoen, A. Elberg, et J. Crowley. 2024. [GGally: Extension to « ggplot2 »](#).
- Wheeler, B., et M. Torchiano. 2016. [lmPerm: Permutation Tests for Linear Models](#).
- Wickham, H. 2007. [Reshaping Data with the reshape Package](#). Journal of Statistical Software 21:1-20.
- Wickham, H., M. Averick, J. Bryan, W. Chang, L. D. McGowan, R. François, G. Grolemund, A. Hayes, L. Henry, J. Hester, M. Kuhn, T. L. Pedersen, E. Miller, S. M. Bache, K. Müller, J. Ooms, D. Robinson, D. P. Seidel, V. Spinu, K. Takahashi, D. Vaughan, C. Wilke, K. Woo, et H. Yutani. 2019. [Welcome to the tidyverse](#). Journal of Open Source Software 4:1686.

- Wilkinson, L. 2005. *The Grammar of Graphics*. Springer Science & Business Media.
- Xie, Y. 2014. knitr: A Comprehensive Tool for Reproducible Research in R. *in* V. Stodden, F. Leisch, et R. D. Peng, éditeurs. *Implementing Reproducible Computational Research*. Chapman; Hall/CRC.
- Xie, Y. 2015. [Dynamic Documents with R and knitr](#). 2nd édition. Chapman; Hall/CRC, Boca Raton, Florida.
- Xie, Y. 2024. [knitr: A General-Purpose Package for Dynamic Report Generation in R](#).
- Xie, Y., J. J. Allaire, et G. Grolemund. 2018. [R Markdown: The Definitive Guide](#). Chapman; Hall/CRC, Boca Raton, Florida.
- Xie, Y., C. Dervieux, et E. Riederer. 2020. [R Markdown Cookbook](#). Chapman; Hall/CRC, Boca Raton, Florida.
- Zeileis, A., et T. Hothorn. 2002. [Diagnostic Checking in Regression Relationships](#). *R News* 2:7-10.
- Zeileis, A., D. Meyer, et K. Hornik. 2007. [Residual-based Shadings for Visualizing \(Conditional\) Independence](#). *Journal of Computational and Graphical Statistics* 16:507-525.
- Zhu, H. 2024. [kableExtra: Construct Complex Table with « kable » and Pipe Syntax](#).

Annexe A

Données utilisées dans le livre

A.1. Fichier compressé tout-inclus

Tous les fichiers de données et de code dans un [fichier zip](#)

A.2. Tous les fichiers séparés

- [age.csv](#)
- [anc1dat.csv](#)
- [anc3dat.csv](#)
- [atmosphere.txt](#)
- [Banta_TotalFruits.csv](#)
- [Biston_pd_1.csv](#)
- [Biston_pd_2.csv](#)
- [Biston_student.csv](#)
- [Biston.postdoc.csv](#)
- [Biston.prof.csv](#)
- [Dam10dat.csv](#)
- [dragons.csv](#)
- [ErablesGatineau.csv](#)
- [gala.txt](#)
- [hypoxia.uottawa.csv](#)
- [JacobsenDangles_1.csv](#)

- JacobsenDangles_2.csv
- loglin.csv
- mouflon.csv
- Mregdat.csv
- nematodes.csv
- nestdat.csv
- nr2wdat.csv
- pollution.txt
- Rway_enfR_data_code.zip
- salmonella.csv
- simulies.csv
- simuliidae.csv
- skulldat_2020.csv
- skulldat-rm.csv
- smoking.txt
- Stu2mdat.csv
- Stu2wdat.csv
- sturgdat.csv
- sturgeon.csv
- unicorns_aggression.csv
- unicorns.csv
- unicorns.txt
- unicorns.xlsx
- USPopSurvey.csv
- wmc2daf2.csv
- wmcdat2.csv

A.3. Code R et fonctions utilisées dans les diapositives

- book-fnc.R
- extra_funs.R
- glmm_simdev.rda