

Introduction aux méthodes quantitatives en sciences sociales avec R

Philippe Apparicio et Jérémy Gelb

2021-08-30

Table des matières

Préface	9
Comment lire ce livre	9
Pourquoi faut-il programmer en sciences sociales?	9
Structure du livre	10
Remerciements	10
À propos des auteurs	13
Première partie Découverte de R	15
1 Prise en main de R	17
1.1 Histoire et philosophie de R	17
1.2 Environnement de travail	20
1.2.1 Installer R	20
1.2.2 L'environnement RStudio	20
1.2.3 Installer et charger un <i>package</i>	24
1.2.4 Obtenir de l'aide	25
1.3 Les bases du langage R	26
1.3.1 Hello World!	26
1.3.2 Objets et expressions	26
1.3.3 Fonctions et arguments	28
1.3.4 Principaux types de données	29
1.3.5 Opérateurs	30
1.3.6 Structures de données	32
1.4 Manipuler des données	39
1.4.1 Charger un <i>DataFrame</i> depuis un fichier	39
1.4.2 Manipuler un <i>DataFrame</i>	42
1.5 Bien structurer un code R	60
1.6 Conclusion et ressources pertinentes	62
Deuxième partie Analyses univariées	65
2 Statistiques descriptives univariées	67
2.1 Notion de variable	67
2.1.1 La variable : l'opérationnalisation d'un concept	67
2.1.2 Les types de variables	69
2.2 Les types de données	71

2.2.1	Données secondaires <i>versus</i> données primaires	71
2.2.2	Données transversales <i>versus</i> données longitudinales	72
2.2.3	Données spatiales <i>versus</i> données aspatiales	72
2.2.4	Données individuelles <i>versus</i> données agrégées	72
2.3	Statistique descriptive et statistique inférentielle	74
2.3.1	Population, échantillon et inférence	74
2.3.2	Deux grandes familles de méthodes statistiques	74
2.4	La notion de distribution	75
2.4.1	Définitions générales	75
2.4.2	Anatomie d'une distribution	77
2.4.3	Principales distributions	79
2.4.4	Conclusion sur les distributions	94
2.5	Statistiques descriptives sur des variables quantitatives	94
2.5.1	Les paramètres de tendance centrale	94
2.5.2	Les paramètres de position	95
2.5.3	Les paramètres de dispersion	96
2.5.4	Les paramètres de forme	99
2.5.5	La transformation des variables	115
2.5.6	Mise en œuvre dans R	118
2.6	Statistiques descriptives sur des variables qualitatives et semi-qualitatives	129
2.6.1	Les fréquences	129
2.6.2	Mise en œuvre dans R	130
2.7	Pour aller un peu plus loin : les statistiques descriptives pondérées	131

Liste des tableaux

1.1	Opérateurs mathématiques	30
1.2	Opérateurs relationnels	31
1.3	Opérateurs logiques	31
1.4	Un premier DataFrame	37
1.5	Temps nécessaire pour lire les données en fonction du type de fichiers	43
1.6	Avantages et inconvénients du tidyverse	43
1.7	Ressources pertinente pour en apprendre plus sur R	63
2.1	Revenus moyens et médians des ménages en dollars, municipalités de l'île de Montréal, 2015	95
2.2	Stastistiques descriptives de l'exposition au bruit des cyclistes par minute dans trois villes (dB(A), Laeq 1min)	96
2.3	Calcul des mesures de dispersion sur des données fictives	99
2.4	Illustration de la sensibilité des mesures de dispersion à l'unité de mesure et aux valeurs extrêmes	100
2.5	Résumé de la sensibilité de la moyenne et des mesures de dispersion	100
2.6	Les différents tests d'hypothèse pour la normalité	106
2.7	Calculs des tests de normalité pour différentes distributions	106
2.8	Comparaison des LogLikekelihood des trois distributions	111
2.9	Illustration des trois tranformations	117
2.10	Les différents types de fréquences sur une variable qualitative ou semi-qualitative	129
2.11	Les différents types de fréquences sur une variable semi-qualitative	130
2.12	Calcul de la moyenne pondérée	132
2.13	Statistiques de l'aire de jeux la plus proche par secteur de recensement pondérées par la population de moins de 10 ans	132

Table des figures

1	Cargo, le plus beau	11
1.1	Lieu de pélerinage de R	18
1.2	Nombre d'articles trouvés sur Google Scholar (Source : Robert A. Muenchen)	18
1.3	Métaphore sur les langages et programmes d'analyse statistique	20
1.4	Icône des encadrés dédiés aux packages	20
1.5	La console de base de R	21
1.6	Environnement de base de RStudio	21
1.7	RStudio avec le style pastel on dark	22
1.8	Les quatre fenêtres de RStudio	23
1.9	Les quatre fenêtres de RStudio	23
1.10	Description des packages	25
1.11	Arguments de la fonction round	28
1.12	Du vecteur à la matrice	35
1.13	Un array avec trois dimension	36
1.14	De la donnée au DataFrame	37
1.15	Répartition temporelle des accidents à vélo	52
1.16	Fusion de DataFrames	57
1.17	Jointure de DataFrames	58
2.1	Les types de variables	70
2.2	Exemple d'agrégation de données individuelles	73
2.3	Distribution théorique d'un lancé de dé	76
2.4	Distribution empirique d'un lancé de dé (n=10)	76
2.5	Distribution empirique d'un lancé de dé (n=10)	76
2.6	Distributions uniformes continues	78
2.7	18 distributions essentielles, design inspiré de ?	79
2.8	La distribution binomiale	81
2.9	La distribution géométrique	82
2.10	La distribution binomiale négative	83
2.11	La distribution de poisson	84
2.12	La distribution de poisson avec excès de zéros	85
2.13	La distribution Gaussienne	85
2.14	La distribution skew-Gaussienne	86
2.15	La distribution log-gaussienne	87
2.16	La distribution de Student	88
2.17	La distribution de Cauchy	88
2.18	La distribution du Chi ²	89
2.19	La distribution exponentielle	90

2.20 La distribution de Gamma	91
2.21 La distribution de Beta	91
2.22 La distribution de Weibull	92
2.23 La distribution de Pareto	93
2.24 Exemples de cartographie avec une discréétisation selon les quantiles	96
2.25 Graphique en violon, boîte à moustaches et intervalle interquartile	97
2.26 Formes d'une distribution et les coefficients d'asymétrie et d'aplatissement	101
2.27 Asymétrie d'une distribution	102
2.28 Aplatissement d'une distribution	103
2.29 Distributions et courbe normale	104
2.30 Diagrammes quantile-quantile	105
2.31 Distribution empirique des temps de retard des bus à Toronto en janvier 2019	110
2.32 Comparaison des distributions ajustées aux données de retard des bus	111
2.33 Distribution empirique du nombre d'accidents par intersection impliquant un cycliste à Montréal en 2017 dans les quartiers centraux	113
2.34 Ajustement des distributions de poisson et poisson avec excès de zéros	114
2.35 Histogramme avec courbe normale	121
2.36 Histogramme des transformations	128
2.37 Différents graphiques pour représenter les fréquences absolues et relatives	130
2.38 Accessibilité aux aires de jeux par secteur de recensement, Communauté métropolitaine de Montréal, 2016	133

Préface

Comment lire ce livre

Si vous googlez l'expression « comment lire un livre ? », vous trouverez une multitude de conseils et astuces. Pour ce livre, nous conseillons de le lire de gauche à droite et page par page. Plus sérieusement, il comprend plusieurs types de blocs de texte qui, on l'espère, faciliteront la lecture.



Bloc packages : habituellement localisé en début du chapitre, il comprend la liste des *packages R* utilisés pour un chapitre.



Bloc objectif : comprend une description des objectifs d'une section.



Bloc notes : comprend une information secondaire sur une notion, un élément, une idée abordée dans une section.



Bloc pour aller plus loin : peut comprendre des références ou des extensions d'une méthode statistique abordée dans une section.



Bloc astuce : décrit un élément qui vous facilera la vie : une propriété statistique, un *package*, une fonction, une syntaxe R.



Bloc attention : comprend une notion ou un élément important à bien maîtriser.

Pourquoi faut-il programmer en sciences sociales ?

Il est vrai que la programmation n'est pas la compétence qui vient tout de suite à l'esprit lorsque l'on pense aux sciences sociales. Pourtant, cette compétence est de plus en plus importante, et ce pour plusieurs raisons :

- Une part toujours plus grande des phénomènes sociaux se produisent ou peuvent s'observer au travers d'environnements numériques. Être capable d'exploiter efficacement ces outils permet d'extraire des données riches sur des phénomènes complexes, tels qu'en témoigne des études récentes sur la propagation de la désinformation sur les réseaux sociaux (?), la migration des personnes (?), la propagation et les risques de contamination de la COVID19 (?), etc. Le plus souvent, les interfaces (API par exemple) permettant d'accéder à ces données nécessitent une base en programmation.

- La quantité de données numériques ouvertes et accessibles en ligne croît chaque année sur des sujets très divers. La plupart des villes et des gouvernements ont maintenant leurs portails de données ouvertes auxquels s'ajoutent les données produites par des projets collaboratifs comme OpenStreetMap¹ ou NoisePlanet². Récupérer ces données et les structurer pour les utiliser à des fins de recherche nécessite le plus souvent des compétences en programmation.
- Les méthodes d'analyse quantitative connaissent également un développement très important. Les logiciels propriétaires peinent à suivre la cadence de ce développement contrairement aux logiciels à code source ouvert qui permettent d'avoir accès aux dernières méthodes. Il est souvent long et coûteux de développer une interface graphique pour un logiciel, ce qui explique que la plupart de ces programmes en sont dépourvus et nécessitent alors de savoir programmer pour les utiliser.
- Savoir programmer donne une liberté considérable en recherche. Cette compétence permet notamment de ne plus être limité aux fonctionnalités proposées par des logiciels spécifiques. Il devient possible d'innover tant en matière de structuration, d'analyse que de représentation des résultats en écrivant vos propres fonctions. Cette flexibilité contribue directement à la production d'une recherche de meilleure qualité et plus diversifiée.
- Programmer permet également d'automatiser des tâches qui autrement seraient extrêmement répétitives. Déplacer et renommer une centaine de fichiers ? Retirer les lignes inutiles dans un ensemble de fichiers CSV et les compiler dans une seule base de données ? Tester parmi des milliers d'adresses lesquelles sont valides ? Récupérer chaque jour les messages postés sur un forum ? Autant de tâches faciles à automatiser si l'on sait programmer.
- Dans un logiciel avec une interface graphique, il est compliqué de conserver un historique des opérations effectuées. Programmer permet au contraire de garder une trace de l'ensemble des actions effectuées au cours d'un projet de recherche. En effet, le code utilisé reste disponible et permet de reproduire la méthode et les résultats obtenus ce qui est essentiel dans le monde de la recherche. À cela s'ajoute le fait que chaque ligne de code que vous écrivez vient s'ajouter à un capital de code que vous possédez, car elles pourront être réutilisées dans d'autres projets !

Structure du livre

À écrire plus tard.

Remerciements

Note au beau Cargo (chien Mira) qui nous supporte dans l'écriture du livre !

¹<https://www.openstreetmap.org>

²https://noise-planet.org/map_noisecapture/index.html



FIG. 1 : Cargo, le plus beau

À propos des auteurs

Philippe Apparicio (<http://www.ucs.inrs.ca/philippe-apparicio>) est professeur titulaire au Centre Urbanisation Culture Société de l'INRS (<http://www.ucs.inrs.ca/>). Il enseigne au programme de maîtrise en études urbaines (<http://www.ucs.inrs.ca/ucs/etudier/programmes/etudes-urbaines>) les cours *méthodes quantitatives appliquées aux études urbaines* et *analyses spatiales appliquées aux études urbaines*. Il a aussi créé et enseigné, il y a plusieurs années, le cours *systèmes d'information géographique appliqués aux études urbaines*. Durant les dernières années, il a offert plusieurs formations aux Écoles d'été du Centre interuniversitaire québécois de statistiques sociales (CIQSS, <https://www.ciqss.org/>). Titulaire de la Chaire de recherche du Canada (niveau 2) sur l'équité environnementale et la ville, il est le directeur du **laboratoire d'équité environnementale** (<http://laeq.ucs.inrs.ca>). Géographe de formation, ses intérêts de recherche actuels incluent la justice et l'équité environnementale, la pollution atmosphérique, le bruit et le vélo en ville. Il a publié une centaine d'articles scientifiques dans différents domaines des études urbaines et de la géographie.

Jérémy Gelb est candidat au doctorat en études urbaines à l'INRS (sous la supervision de Philippe Apparicio) et membre du **laboratoire d'équité environnementale** (<http://laeq.ucs.inrs.ca>). Son sujet de thèse porte sur l'exposition des cyclistes aux pollutions atmosphériques et sonores en milieu urbain. Il utilise quotidiennement des systèmes d'information géographique (SIG) et est tombé dans la marmite de l'*open source* avec le triptyque QGIS, R et Python au début de sa maîtrise. Il a récemment développé deux packages R : **geocmeans** et **spNetwork**, permettant respectivement d'effectuer des analyses de classification floue non-supervisée pondérée spatialement et des estimations de densité par kernel sur réseau.

Philippe et Jérémy travaillent étroitement ensemble depuis déjà plusieurs années. Avec d'autres collègues, ils ont copublié plusieurs articles (?;?;?;?;?;?;?;?;?;?;?;?;?). Tous deux s'intéressent à l'exposition des cyclistes à la pollution atmosphérique et sonore dans plusieurs villes à travers le monde : Philippe ayant une préférence pour les collectes dans les villes des Suds (notamment indiennes, africaines et latino-américaines) et Jérémy dans les villes du Nord (européennes et nord-américaines).

Première partie

Découverte de R

Chapitre 1

Prise en main de R

Dans ce chapitre, nous reviendrons brièvement sur l'histoire de R et la philosophie qui entoure le logiciel. Nous donnerons quelques conseils pour son installation et la mise en place d'un environnement de développement. Nous présenterons les principaux objets qui sous-tendent le travail effectué avec R (*dataframe*, vecteur, matrice, etc.) et comment les manipuler avec des exemples appliqués. Enfin, nous terminerons cette section avec un tour d'horizon des capacités graphiques de R. Si vous maîtrisez déjà R, nullement besoin de lire ce chapitre !



Dans ce chapitre, nous utiliserons principalement les *packages* suivants :

- Pour importer des fichiers externes :
 - * **foreign** pour entre autres les fichiers *dbase* et ceux des logiciels SPSS et Stata
 - * **sas7bdat** pour les fichiers du logiciel SAS
 - * **xlsx** pour les fichiers Excel
- Pour manipuler des chaînes de caractères et des dates :
 - * **stringr** pour les chaînes de caractères
 - * **lubridate** pour les dates
- Pour manipuler des données :
 - * **dplyr** du **tidyverse** propose une grammaire pour manipuler et structurer des données.

1.1 Histoire et philosophie de R

R est à la fois un langage de programmation et un logiciel libre (sous la licence publique générale GNU) dédié à l'analyse statistique et soutenu par une fondation : *R foundation for Statistical computing*. Il est principalement écrit en C et Fortran.

R a été créé par Ross Ihaka et Robert Gentleman à l'Université d'Auckland en Nouvelle-Zélande. Si vous avez un jour l'occasion de passer dans le coin, une plaque est affichée dans le département de statistique de l'université, ça mérite le détour (figure 1.1). Une version expérimentale a été publiée en 1996, mais la première version stable ne date que de 2000, il s'agit donc d'un logiciel relativement récent si on le compare à ses concurrents SPSS (1968) , SAS (1976) et Stata (1984).

R a cependant réussi à s'imposer tant dans la milieu de la recherche que dans le secteur privé. Pour s'en convaincre, il suffit de lire l'excellent article concernant la popularité des logiciels d'analyse de données tiré du site r4stats.com¹ (figure 1.2).

¹<http://r4stats.com/articles/popularity>

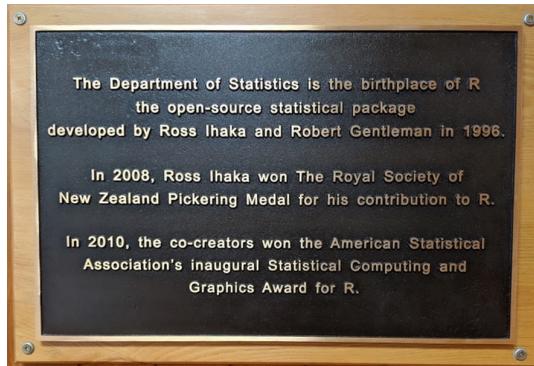


FIG. 1.1 : Lieu de pélerinage de R

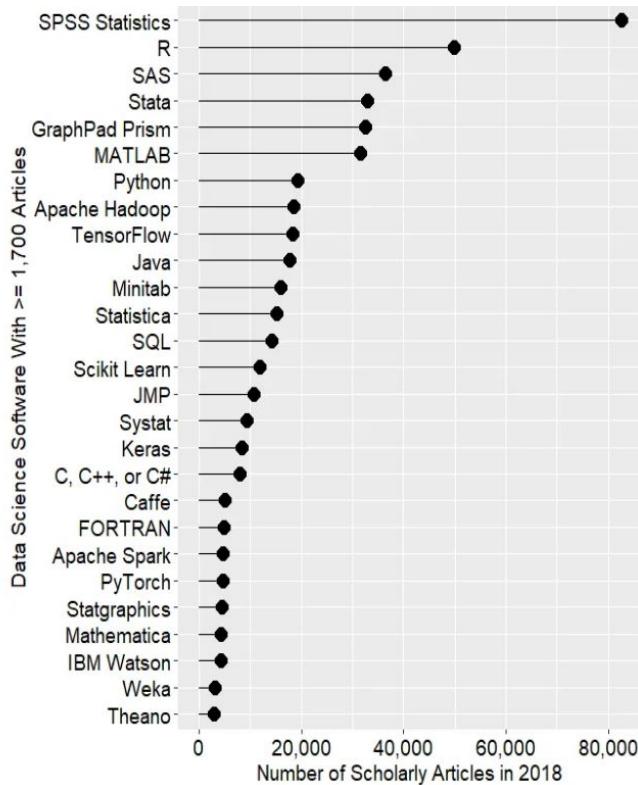


FIG. 1.2 : Nombre d'articles trouvés sur Google Scholar (Source : Robert A. Muenchen)

Les nombreux atouts de R justifient largement sa popularité sans cesse croissante :

- R est un logiciel à code source ouvert (*open source*) et ainsi accessible à tous gratuitement.
- Le développement du langage R est centralisé, mais la communauté peut créer et partager facilement des *packages*. Les nouvelles méthodes sont ainsi rapidement implémentées comparativement aux logiciels propriétaires.
- R est un logiciel multi-plateforme, fonctionnant sur Linux, Unix, Windows et Mac.
- Comparativement à ses concurrents, R dispose d'excellentes solutions pour manipuler des données et réaliser des graphiques.
- R dispose de nombreuses interfaces lui permettant de communiquer, notamment avec des systèmes de bases de données SQL et non SQL (MySQL, PostgreSQL, MongoDB, etc.), avec des systèmes de *big data* (Spark, Hadoop), avec des systèmes d'information géographique (QGIS, ArcGIS) et même avec des services en ligne comme Microsoft Azure ou Amazon AWS.

- R est un langage de programmation à part entière, ce qui lui donne plus de flexibilité que ses concurrents dans le domaine privé (SPSS, SAS, STATA). Avec R, vous pouvez accomplir des tâches aussi variées que : monter un site web, créer un robot collectant des données en ligne, combiner des fichiers PDF, composer des diapositives pour une présentation ou même éditer un livre (comme celui-ci), mais aussi et surtout réaliser des analyses statistiques.

Un des principaux attrait de R est la quantité astronomique de *packages* actuellement disponibles. **Un package est un ensemble de nouvelles fonctionnalités développées par un·e ou plusieurs utilisateurs·trices de R et mises à disposition de l'ensemble de la communauté.** Par exemple, le *package ggplot2* est dédié à la réalisation de graphiques ; les *packages data.table* et *plyr* permettent de manipuler des tableaux de données ; le *package car* apporte de nombreux outils pour faciliter l'analyse de modèles de régressions, etc. Ce partage des *packages* rend accessible à tous des méthodes d'analyses complexes et récentes et favorise grandement la reproductibilité de la recherche. Cependant, ce fonctionnement implique quelques désavantages :

- il existe généralement plusieurs *packages* pour effectuer le même type d'analyse, ce qui peut devenir une source de confusion ;
- certains *packages* cessent d'être mis à jour au fil des années, ce qui nécessite de leur trouver d'autres alternatives (et ainsi apprendre la syntaxe des nouveaux *packages*) ;
- il est impératif de s'assurer de la fiabilité des *packages* que vous souhaitez utiliser, car n'importe qui peut proposer un *package*.

Il nous semble important de relativiser d'emblée la portée du dernier point. Il est rarement nécessaire de lire et analyser le code source d'un *package* pour s'assurer de sa fiabilité. Nous ne sommes pas des spécialistes de tous les sujets et il peut être extrêmement ardu de comprendre la logique d'un code écrit par une autre personne. Nous vous recommandons donc de privilégier l'utilisation de *packages* qui :

- ont fait l'objet d'une publication dans une revue à comité de lecture ou qui ont déjà été cités dans des études ayant fait l'objet d'une publication revue par les pairs ;
- font partie de projets comme ROpenSci² prônant la vérification par les pairs ou subventionnés par des organisations comme R Consortium³ .
- sont disponibles sur l'un des deux principaux répertoires de *packages* R, soit CRAN⁴ et Bioconductor⁵ .

Toujours pour nuancer notre propos, il convient de distinguer *package* de *package!* Certains d'entre eux sont des ensembles très complexes de fonctions permettant de réaliser des analyses poussées alors que d'autres sont des projets plus modestes dont l'objectif principal est de simplifier le travail des utilisateurs·trices. Ces derniers ressemblent à des petites boîtes à outils et font généralement moins l'objet d'une vérification intensive.

Pour conclure cette section, l'illustration partagée sur Twitter par Darren L Dahly résume avec humour la force du logiciel R et de sa communauté (figure 1.3) : R apparaît clairement comme une communauté hétéroclète, mais diversifiée et adaptable.

Dans ce livre, nous détaillerons les **packages** utilisés dans chaque section avec un encadré spécifique, accompagné de l'icône présenté à la figure 1.4.

²<https://ropensci.github.io/reproducibility-guide/sections/introduction/>

³<https://www.r-consortium.org/>

⁴<https://cran.r-project.org/>

⁵<https://www.bioconductor.org/>

If statistics programs/languages were cars...

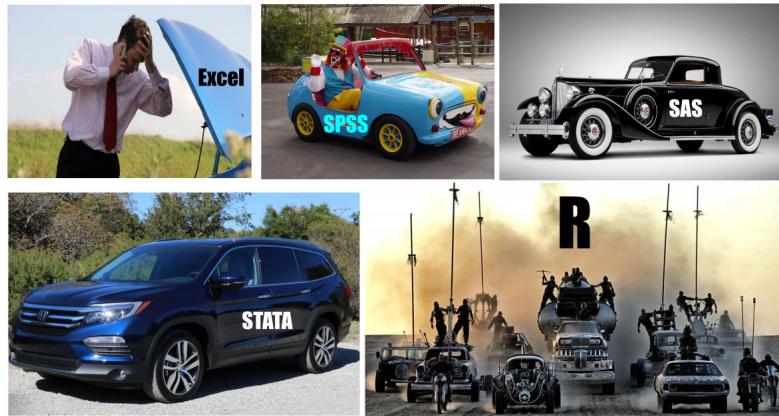


FIG. 1.3 : Métaphore sur les langages et programmes d'analyse statistique



FIG. 1.4 : Icône des encadrés dédiés aux packages

1.2 Environnement de travail

Dans cette section, nous vous proposons une visite de l'environnement de travail classique R.

1.2.1 Installer R

La première étape pour travailler avec R est bien sûr de l'installer. Pour ce faire, il suffit de visiter le site web de CRAN⁶ et de télécharger la dernière version de R en fonction de votre système d'exploitation : Windows, Linux ou Mac. Une fois installé, si vous démarrez R immédiatement, vous aurez alors accès à une console, plutôt rudimentaire, attendant sagement vos instructions (figure 1.5).

Notez que vous pouvez aussi télécharger des versions plus anciennes de R en allant sur ce lien⁷. Ceci peut être intéressant lorsque vous voulez reproduire des résultats d'une autre étude ou que certains *packages* ne sont plus disponibles dans les nouvelles versions.

1.2.2 L'environnement RStudio

Rares sont les utilisateurs·trices de R qui préfèrent travailler directement avec la console classique. Nous vous recommandons vivement d'utiliser RStudio, soit un environnement de développement dédié à R, offrant une intégration très intéressante d'une console, d'un éditeur de texte, d'une fenêtre de visualisation des données, d'une autre pour les graphiques, d'un accès à la documentation, etc. En d'autres termes, si R est un vélo minimaliste, RStudio permet d'y rajouter des freins, des vitesses, un porte-bagage, des gardes-boues et une selle confortable. Vous pouvez télécharger⁸ et installer RStudio sur Windows, Li-

⁶<https://cran.r-project.org/>

⁷<https://cran.r-project.org/bin/windows/base.old/>

⁸<https://rstudio.com/products/rstudio/download>

```
R version 4.0.2 (2020-06-22) -- "Taking Off Again"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R est un logiciel libre livré sans AUCUNE GARANTIE.
Vous pouvez le redistribuer sous certaines conditions.
Tapez 'license()' ou 'licence()' pour plus de détails.

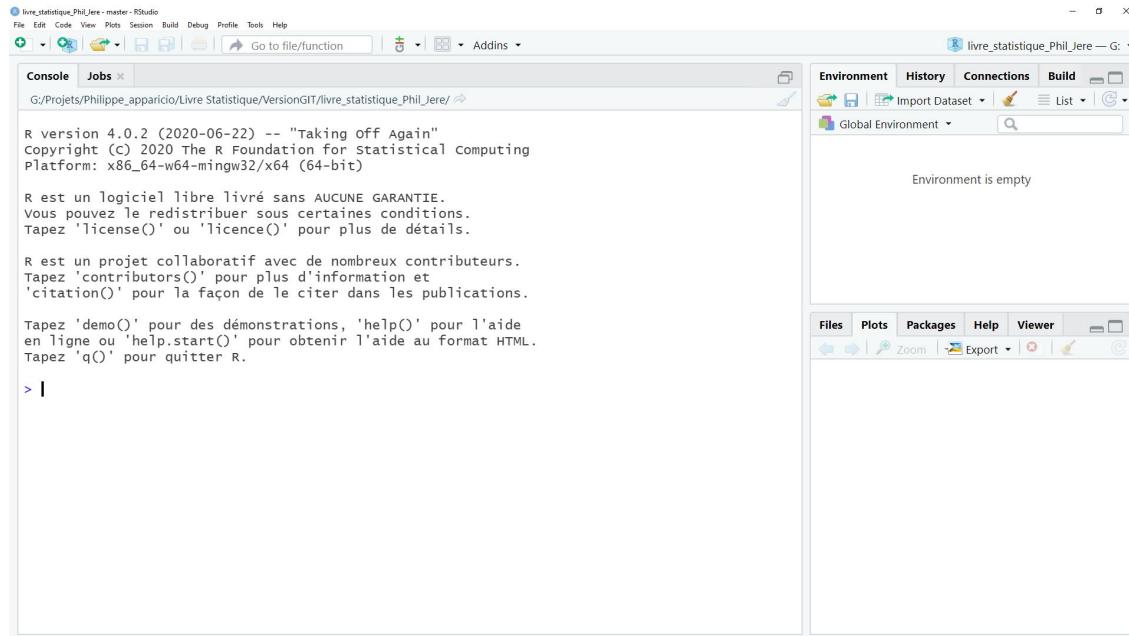
R est un projet collaboratif avec de nombreux contributeurs.
Tapez 'contributors()' pour plus d'information et
'citation()' pour la façon de le citer dans les publications.

Tapez 'demo()' pour des démonstrations, 'help()' pour l'aide
en ligne ou 'help.start()' pour obtenir l'aide au format HTML.
Tapez 'q()' pour quitter R.

> 4+4
[1] 8
> -
```

FIG. 1.5 : La console de base de R

nux et Mac. La version de base est gratuite, mais l'entreprise qui développe ce logiciel propose aussi des versions commerciales du logiciel qui assurent essentiellement un support technique. Il existe d'autres environnements de développement pour travailler avec R (VisualStudio, Jupyter, Tinn-R, Radiant, RIDE, etc.), mais RStudio offre à ce jour la meilleure option en terme de facilité d'installation, de prise en main et de fonctionnalités proposées (voir l'interface de RStudio à la figure 1.6).

**FIG. 1.6 :** Environnement de base de RStudio

Avant d'aller plus loin, notez que :

- La console actuellement ouverte dans RStudio vous informe de la version de R que vous utilisez. Vous pouvez en effet avoir plusieurs versions de R installées sur votre ordinateur et passer de l'une à l'autre avec RStudio. Pour cela, naviguez dans l'onglet *Tools / Global Options* et dans le volet *General*,

vous pouvez sélectionner la version de R que vous souhaitez utiliser.

- L'aspect de RStudio peut être modifié en navigant dans l'onglet *Tools / Global Options* et dans le volet *Appearance*. Nous avons une préférence pour le mode sombre avec le style *pastel on dark*, mais libre à chacun de choisir le style qui lui convient.

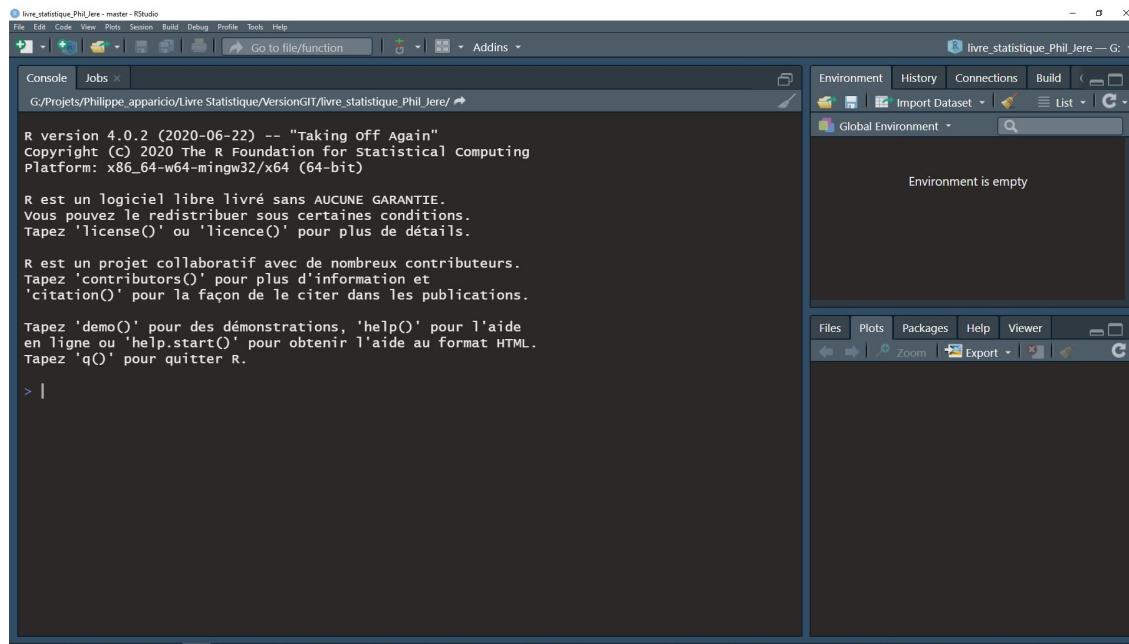


FIG. 1.7 : RStudio avec le style pastel on dark

Une fois ces détails réglés, vous pouvez ouvrir votre première feuille de code en allant dans l'onglet *File / New File / R Script*. Votre environnement est maintenant découpé en quatre fenêtres (figure 1.8) :

1. L'éditeur de code, vous permettant d'écrire le script que vous voulez exécuter et permettant de garder une trace de votre travail. Ce script peut être enregistré sur votre ordinateur avec l'extension **.R**, mais ce n'est qu'un simple fichier texte.
2. La console vous permettant d'exécuter votre code R et de voir les résultats s'afficher au fur et à mesure.
3. La fenêtre d'environnement vous montrant les objets, fonctions et jeux de données actuellement disponibles dans votre session (chargés dans la mémoire vive).
4. La fenêtre de l'aide, des graphiques et de l'explorateur de fichiers. Vous pouvez accéder ici à la documentation de R et des *packages* que vous utilisez, aux sorties graphiques que vous produisez et aux dossiers de votre environnement de travail.

Prenons un bref exemple, tapez la syntaxe suivante dans l'éditeur de code (fenêtre 1 à la figure 1.8) :

```
ma_somme <- 4+4
```

Sélectionnez ensuite cette syntaxe (mettre en surbrillance avec la souris), quand vous utilisez le raccourci *Ctrl+Enter* ou cliquez sur le bouton *Run* (avec la flèche verte), cette syntaxe est envoyée à la console qui l'exécute immédiatement. Notez que rien ne se passe tant que le code n'est pas envoyé à la console. Il s'agit donc de deux étapes distinctes : écrire son code, puis l'envoyer à la console. Vous constaterez également qu'un objet *ma_somme* est apparu dans votre environnement et que sa valeur est bien 8. Votre console se "souvient" de cette valeur, elle est actuellement stockée dans votre mémoire vive sous le nom de *ma_somme* (figure 1.9).

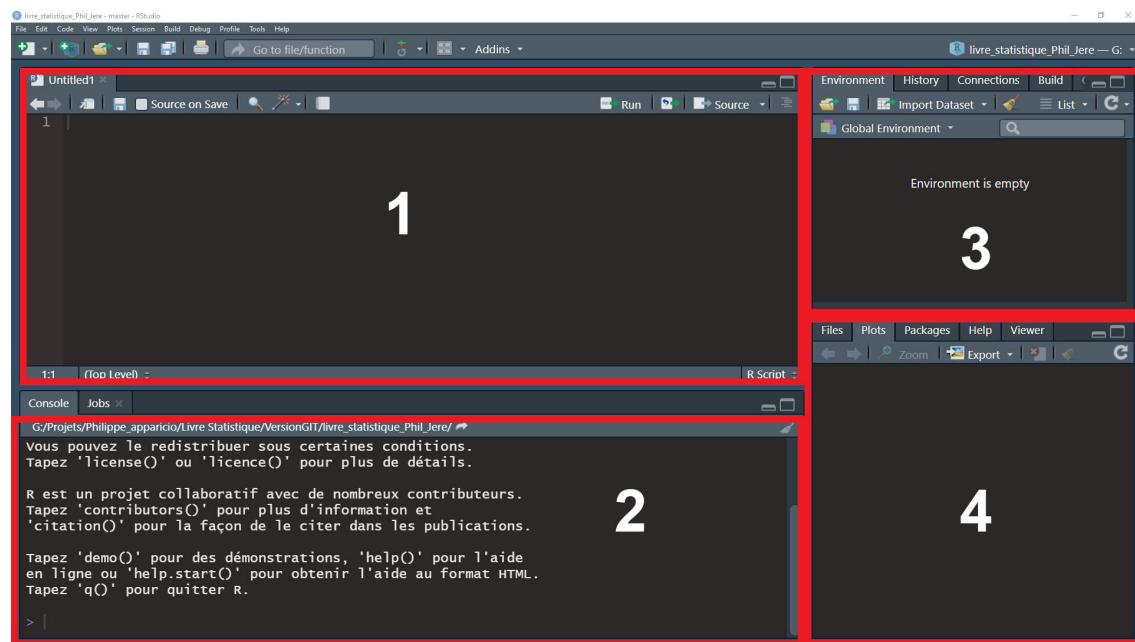


FIG. 1.8 : Les quatre fenêtres de RStudio

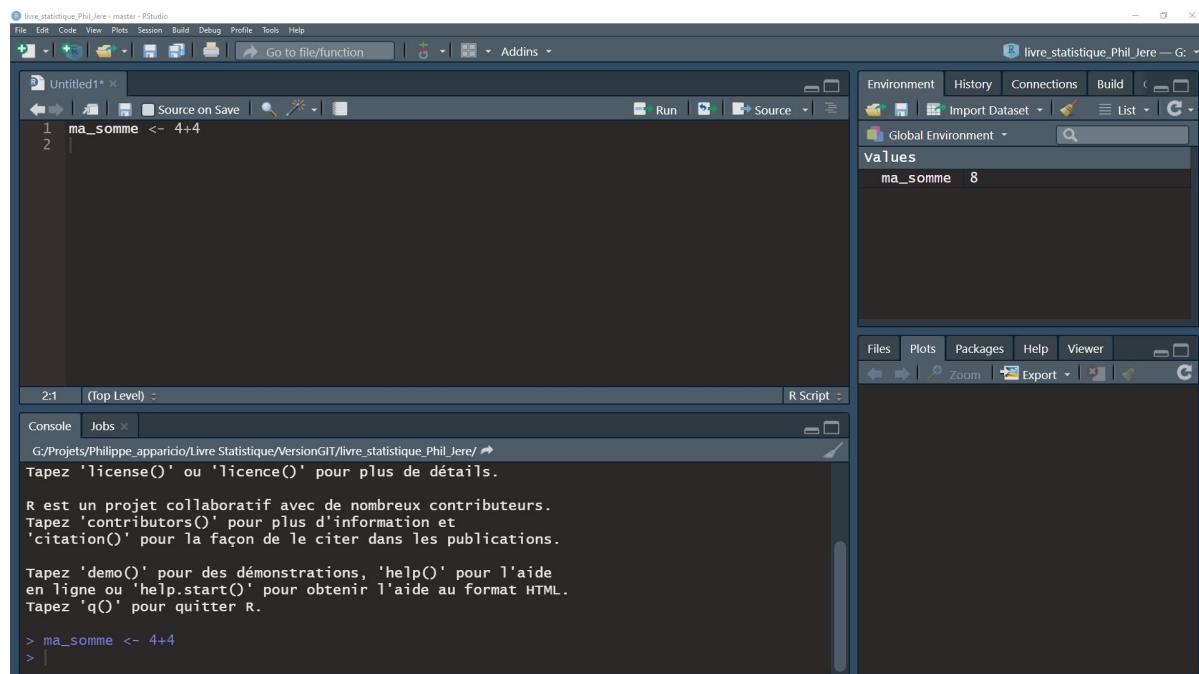


FIG. 1.9 : Les quatre fenêtres de RStudio

Pour conclure cette section, nous vous invitons à enregistrer votre première syntaxe R (*File / Save As*) dans un fichier **.R** que vous pouvez appeler par exemple “*mon_premier_script.R*”. Fermez ensuite RStudio, redémarrez le et ouvrez (*File / Open File*) votre fichier “*mon_premier_script.R*”. Vous pouvez constater que votre code est toujours présent, mais que votre environnement est vide tant que vous n'exécutez pas votre syntaxe. En effet, lorsque vous fermez RStudio, l'environnement est vidé pour libérer de la mémoire vive. Ceci peut poser problème lorsque certains codes sont très longs à exécuter, nous verrons donc plus tard comment enregistrer l'environnement en cours pour le recharger par la suite.

1.2.3 Installer et charger un *package*

Dans la section sur la Philosophie de R, nous avons souligné la place centrale jouée par les *packages*. Notez que les termes **paquet** et plus rarement *librairie* sont parfois utilisés en français. Voyons ensemble comment installer un *package* intitulé **lubridate**, qui nous permettra plus tard de manipuler des données temporelles.

1.2.3.1 Installer un *package* depuis CRAN

Pour installer un *package*, vous devez être connecté à Internet puisque R va accéder au répertoire de *packages CRAN* pour télécharger le *package* et l'installer sur votre machine. Cette opération est réalisée avec la fonction `install.packages`.

```
install.packages("lubridate")
```

Notez qu'une fois que le *package* est installé, vous n'aurez plus besoin de le refaire. Le *package* est disponible localement sur votre ordinateur, à moins de le désinstaller explicitement avec la fonction `remove.packages`.

1.2.3.2 Installer un *package* depuis GitHub

CRAN est le répertoire officiel des *packages* de R. Vous pouvez cependant télécharger des *packages* provenant d'autres sources. Très souvent, les *packages* sont disponibles sur le site web GitHub⁹ et l'on peut même y trouver des versions en développement avec des fonctionnalités encore non intégrées dans la version sur CRAN. Reprenons le cas de **lubridate**, sur GitHub, il est disponible à la page suivante¹⁰. Pour l'installer nous devons d'abord installer un autre *package* appelé **remotes** (depuis CRAN).

```
install.packages("remotes")
```

Maintenant que nous disposons de **remotes**, nous pouvons utiliser la fonction d'installation `remotes::install_github` pour directement télécharger **lubridate** depuis GitHub.

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

1.2.3.3 Charger un *package*

Maintenant que **lubridate** est installé, nous pouvons le charger dans notre session actuelle de R et accéder aux fonctions qu'il propose. Pour cela, suffit d'utiliser la fonction `library`. Conventionnellement, l'appel

⁹<https://github.com/>

¹⁰<https://github.com/tidyverse/lubridate>

des *packages* se fait au tout début du script que vous rédigez. Rien ne vous empêche de le faire au fur et à mesure de votre code, mais ce dernier perd alors en lisibilité. Notez qu'à chaque nouvelle session (redémarrage de R), il faudra recharger les *packages* dont vous avez besoin.

```
library(lubridate)
```

Si vous obtenez un message d'erreur du type :

Error in library(mon_package) : aucun package nommé 'mon_package' n'est trouvé.

Cela signifie que le *package* que vous tentez de charger n'est pas encore installé sur votre ordinateur. Dans ce cas, réessayer de l'installer avec la fonction `install.packages`. Si le problème persiste, vérifiez que vous n'avez pas fait une faute de frappe dans le nom du *package*. Vous pouvez également redémarrer RStudio et réessayer d'installer le *package*.

1.2.4 Obtenir de l'aide

Lorsque vous installez des *packages* dans R, vous téléchargez aussi leur documentation. Tous les *packages* de CRAN disposent d'une documentation, ce n'est pas forcément vrai pour GitHub. Dans RStudio, vous pouvez accéder à la documentation des *packages* dans l'onglet **Packages** (figure 1.10). Vous pouvez utiliser la barre de recherche pour retrouver rapidement un *package* installé. Si vous cliquez sur le nom du *package*, vous accédez directement à sa documentation dans cette fenêtre.

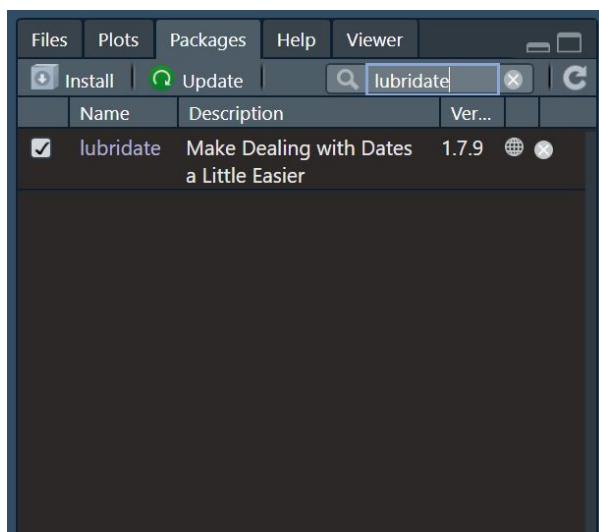


FIG. 1.10 : Description des packages

Vous pouvez également accéder à ces informations en utilisant la syntaxe suivante dans votre console :

```
help(package = 'lubridate')
```

Souvent, vous aurez besoin d'accéder à la documentation d'une fonction spécifique d'un *package*. Affichons la documentation de la fonction `now` de **lubridate** :

```
help(now, package = 'lubridate')
```

ou plus simplement :

```
?lubridate::now
```

Vous pouvez aussi utiliser le raccourci suivant :

```
?now
```

Si vous connaissez le nom d'une fonction, mais vous ne vous souvenez plus à quel *package* elle appartient, lancez une recherche en utilisant un double point d'interrogation :

```
??now
```

Vous découvrirez ainsi que la fonction `now` n'existe pas que dans **lubridate**, ce qui souligne l'importance de bien connaître les *packages* que l'on installe et que l'on charge dans notre session !

Maintenant que nous avons fait le tour de l'environnement de travail, nous pouvons passer aux choses sérieuses, soit les bases du langage R.

1.3 Les bases du langage R

R est un langage de programmation. Il vous permet de communiquer avec votre ordinateur pour lui donner des tâches à accomplir. Dans cette section, nous aborderons les bases du langage. Ce type de section introductory à R est présente dans tous les manuels sur R; elle est donc incontournable. À la première lecture, elle vous semblera probablement aride, et ce, d'autant plus que nous ne réalisons pas d'analyse à proprement parler. Gardez en tête que l'analyse de données requiert au préalable une phase de structuration de ces dernières, opération qui nécessite la maîtrise des notions abordées dans cette section. Nous vous recommandons une première lecture de ce chapitre pour comprendre quelles manipulations que vous pouvez effectuer avec R, la lecture des chapitres suivants dédiés aux statistiques, puis de consulter à nouveau cette section au besoin. Notez aussi que la maîtrise des différents objets et opérations de base de R ne s'acquiert qu'en pratiquant. Vous gagnerez cette expertise au fil de vos prochains codes R, période durant laquelle vous pourrez consulter ce chapitre tel un guide de référence des différents objets et notions fondamentales de R.

1.3.1 Hello World!

Une introduction à un langage de programmation se doit de commencer par le rite de passage **Hello World**. Il s'agit d'une forme de tradition consistant à montrer aux nouveaux utilisateurs·trices comment afficher le message "Hello World" à l'écran avec le langage en question.

```
print("Hello World")
```

```
## [1] "Hello World"
```

Bravo! Vous venez officiellement de faire votre premier pas dans R!

1.3.2 Objets et expressions

Dans R, nous passons notre temps à manipuler des **objets** à l'aide d'**expressions**. Prenons un exemple concret, si vous tapez la syntaxe `4 + 3`, vous manipulez deux objets (4 et 3) au travers d'une expression indiquant que vous souhaitez obtenir la somme des deux objets.

```
4 + 3
```

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

Cette expression est correcte, R comprend vos indications et effectue le calcul.

Il est possible d'enregistrer le résultat d'une expression et de la conserver dans un nouvel objet. On appelle cette opération déclarer une variable.

```
ma_somme <- 4 + 3
```

Concrètement, nous venons de demander à R d'enregistrer le résultat de $4 + 3$ dans un espace spécifique de notre mémoire vive. Si vous regardez dans votre fenêtre **Environment**, vous verrez en effet qu'un objet appelé `ma_somme` est actuellement en mémoire et a pour valeur 7.

Notez ici que le nom des variables ne peut être composé que de lettres, de chiffres, de points(.) et de tiret bas (_) et doit commencer par une lettre. R est sensible à la casse, en d'autres termes, les variables `Ma_somme`, `ma_sommE`, `ma_SOMME`, et `MA_SOMME` renvoient toutes à un objet différent. Attention donc aux fautes de frappe. Si vous déclarez une variable en utilisant le nom d'une variable existante, la première est écrasée par la seconde :

```
age <- 35
age
```

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

```
age <- 45
age
```

```
## [1] 45
```

Portez alors attention aux noms de variables que vous utilisez et réutilisez. Réutilisons notre objet `ma_somme` dans une nouvelle expression :

```
ma_somme2 <- ma_somme + ma_somme
```

Avec cette nouvelle expression, nous indiquons à R que nous souhaitons déclarer une nouvelle variable appelée `ma_somme2`, et que cette variable aura pour valeur `ma_somme + ma_somme`, soit $7 + 7$. Sans surprise, `ma_somme2` a pour valeur 14.

Notez que la mémoire vive (l'environnement) est vidée lorsque vous fermez R. En d'autres termes, R perd complètement la mémoire lorsque vous le fermez. Vous pouvez bien sûr recréer vos objets en relançant les mêmes syntaxes. C'est pourquoi vous devez conserver vos feuilles de codes et ne pas seulement travailler dans la console. La console ne garde aucune trace de votre travail. Pensez donc à bien enregistrer votre code!

Nous verrons dans un autre chapitre comment sauvegarder des objets et les recharger dans une session ultérieure de R (LIEN SECTION). Ce type d'opération est pertinent quand le temps de calcul nécessaire à la production de certains objets est très long.

1.3.3 Fonctions et arguments

Dans R, nous manipulons le plus souvent nos objets avec des **fonctions**. Une fonction est elle-même un objet, mais qui a la particularité de pouvoir effectuer des opérations sur d'autres objets. Par exemple, déclarons l'objet `taille` avec une valeur de 175.897 :

```
taille <- 175.897
```

Nous allons utiliser la fonction `round` dont l'objectif est d'arrondir un nombre à virgule pour obtenir un nombre entier.

```
round(taille)
```

```
## [1] 176
```

Pour effectuer leurs opérations, les fonctions ont généralement besoin d'**arguments**. Ici, `taille` est un argument passé à la fonction `round`. Si nous regardons la documentation de `round` avec `help(round)`, nous constatons que cette fonction prend en réalité deux arguments : `x` et `digits`. Le premier est le nombre que nous souhaitons arrondir et le second le nombre de décimales à conserver. On peut lire dans la documentation que la valeur par défaut de `digits` est 0, ce qui explique que `round(taille)` a produit le résultat de 176.

The screenshot shows the R help documentation for the `round` function. It includes the function signature `round(x, digits = 0)` and `signif(x, digits = 6)`, followed by a detailed description of the arguments:

- x**: a numeric vector. Or, for `round` and `signif`, a complex vector.
- digits**: integer indicating the number of decimal places (`round`) or significant digits (`signif`) to be used. Negative values are allowed (see 'Details').

FIG. 1.11 : Arguments de la fonction `round`

Réutilisons maintenant la fonction `round` mais en gardant une décimale :

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

```
## [1] 175.9
```

Il est aussi possible que certaines fonctions ne requièrent pas d'arguments. Par exemple, la fonction `now` va indiquer la date précise (avec l'heure) et n'a besoin d'aucun argument pour le faire :

```
now()
```

```
## [1] "2021-08-30 16:02:25 EDT"
```

Par contre, si nous essayons de lancer la fonction `round` sans argument, nous obtiendrons une erreur :

```
round()
```

Erreur : 0 arguments passed to ‘round’ which requires 1 or 2 arguments

Le message est très clair, `round` a besoin d'au moins un argument pour fonctionner. Si au lieu d'un nombre, nous avions donné du texte à la fonction `round`, nous aurions aussi obtenu une erreur :

```
round("Hello World")
```

Error in `round("Hello World")` : non-numeric argument to mathematical function

À nouveau le message est très explicite : nous avons passé un argument non-numérique à une fonction mathématique. Lisez toujours vos messages d'erreurs qui vous permettront de repérer les coquilles et de corriger votre code !

Une fonction essentielle est la fonction `print` qui permet d'afficher la valeur d'une variable.

```
print(ma_somme)
```

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

1.3.4 Principaux types de données

Depuis le début de ce chapitre, nous avons déclaré plusieurs variables et essentiellement des données numériques. Dans R, il existe trois principaux types de données de base :

- Les données numériques, qui peuvent être des nombres entiers (appelés *integers*), ou des nombres décimaux (appelés *floats*), par exemple `15` et `15.3`.
- Les données de type texte, qui sont des chaînes de caractères (appelées *strings*) et déclarées entre guillemets `"abcdefg"`.
- Les données booléennes (*booleans*) avec deux valeurs, soit vrai (`TRUE`) ou faux (`FALSE`).

Déclarons une variable pour chacun de ces types :

```
age <- 35
taille <- 175.5
adresse <- '4225 rue de la gauchetiere'
proprietaire <- TRUE
```

Si vous avez un doute sur le type de données stockées dans une variable, vous pouvez utiliser la fonction `typeof`. Par exemple, cela permet de repérer si des données qui sont supposées être numériques sont en fait stockées sous forme de texte comme dans l'exemple ci-dessous.

```
typeof(age)
```

```
## [1] "double"
```

```
typeof(taille)
```

```
## [1] "double"
```

```
tailletxt <- "175.5"
typeof(tailletxt)
```

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

Notez également qu'il existe des types pour représenter l'absence de données :

- pour représenter un objet vide, on utilisera l'objet `NULL`,
- pour représenter une donnée manquante, on utilisera l'objet `NA`,
- pour représenter un texte vide, on utilisera une chaîne de caractère de longueur 0, soit `""`.

```
age2 <- NULL
taille2 <- NA
adresse2 <- ''
```

1.3.5 Opérateurs

Nous avons vu que les fonctions permettent de manipuler des objets. Nous pouvons également effectuer un grand nombre d'opérations avec des opérateurs.

1.3.5.1 Opérateurs mathématiques

Les opérateurs mathématiques permettent d'effectuer du calcul avec des données de type numérique.

1.3.5.2 Opérateurs relationnels

Les opérateurs relationnels permettent de vérifier des conditions dans R. Ils renvoient un booléen, `TRUE` si la condition est vérifiée et `FALSE` si ce n'est pas le cas.

1.3.5.3 Opérateurs logiques

Les opérateurs logiques permettent de combiner plusieurs conditions :

- L'opérateur **ET** permet de vérifier que deux conditions (l'une ET l'autre) sont `TRUE`. Si l'une des deux est `FALSE`, il renvoie `FALSE`.
- L'opérateur **OU** permet de vérifier que l'une des deux conditions est `TRUE` (l'une OU l'autre). Si les deux sont `FALSE`, alors il renvoie `FALSE`.

TAB. 1.1 : Opérateurs mathématiques

Opérateur	Description	Syntaxe	Résultat
<code>+</code>	Addition	<code>4 + 4</code>	8,0
<code>-</code>	Soustraction	<code>4 - 3</code>	1,0
<code>*</code>	Multiplication	<code>4 * 3</code>	12,0
<code>/</code>	Division	<code>12 / 4</code>	3,0
<code>^</code>	Exponentiel	<code>4 ^ 3</code>	64,0
<code>**</code>	Exponentiel	<code>4 ** 3</code>	64,0
<code>%%</code>	Reste de division	<code>15.5 %% 2</code>	1,5
<code>/%%</code>	Division entière	<code>15.5 %/% 2</code>	7,0

TAB. 1.2 : Opérateurs relationnels

Opérateur	Description	Syntaxe	Résultat
<code>==</code>	Égalité	<code>4 == 4</code>	TRUE
<code>!=</code>	Différence	<code>4 != 4</code>	FALSE
<code>></code>	Est supérieur	<code>5 > 4</code>	TRUE
<code><</code>	Est inférieur	<code>5 < 4</code>	FALSE
<code>>=</code>	Est supérieur ou égal	<code>5 >= 4</code>	TRUE
<code><=</code>	Est inférieur ou égal	<code>5 <= 4</code>	FALSE

- L'opérateur NOT permet d'inverser une condition. Ainsi NOT TRUE est FALSE et NOT FALSE est TRUE.

Prenons le temps pour un rapide exemple :

```
A <- 4
B <- 10
C <- -5

# produit TRUE car A est bien plus petit que B et C est bien plus petit que A
A < B & C < A
```

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

```
# produit FALSE car si A est bien plus petit que B,
# B est en revanche plus grand que c
A < B & B < C
```

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

```
# produit TRUE car la seconde condition est inversée
A < B & ! B < C
```

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

```
# produit TRUE car au moins une des deux conditions est juste
A < B | B < C
```

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

Notez que l'opérateur ET est prioritaire sur l'opérateur OU et que les parenthèses sont prioritaires sur tous les opérateurs :

TAB. 1.3 : Opérateurs logiques

Opérateur	Description	Syntaxe	Résultat
<code>&</code>	ET	TRUE & FALSE	FALSE
<code> </code>	OU	TRUE FALSE	TRUE
<code>!</code>	NOT	! TRUE	FALSE

```
# produit TRUE car on commence par tester A < B puis B < C ce qui donne FALSE
# on obtient ensuite
# FALSE | A > C
# enfin, A est bien supérieur à C, donc l'une des deux conditions est vraie
A < B & B < C | A > C
```

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

Notez qu'en arrière-plan, les opérateurs sont en réalité des fonctions déguisées. Il est donc possible de définir de nouveau comportements pour les opérateurs. Il est par exemple possible d'additionner ou comparer des objets spéciaux comme des dates, des géométries, des graphes, etc.

1.3.6 Structures de données

Jusqu'à présent, nous avons utilisé des objets ne comprenant qu'une seule valeur. Or, des analyses statistiques nécessitent de travailler à des volumes de données bien plus grands. Pour stocker plusieurs valeurs, nous allons travailler avec plusieurs structures de données : les vecteurs, les matrices, les tableaux de données et les listes.

1.3.6.1 Vecteurs

Les vecteurs sont la brique élémentaire de R. Ils permettent de stocker une série de valeurs du même type dans une seule variable. Pour déclarer un vecteur, on utilise la fonction `c()` :

```
ages <- c(35,45,72,56,62)
tailles <- c(175.5,180.3,168.2,172.8,167.6)
adresses <- c('4225 rue de la gauchetiere',
             '4223 rue de la gauchetiere',
             '4221 rue de la gauchetiere',
             '4219 rue de la gauchetiere',
             '4217 rue de la gauchetiere')
proprietaires <- c(TRUE,TRUE,FALSE,TRUE,TRUE)
```

Nous venons ainsi de déclarer quatre nouvelles variables étant chacune un vecteur de longueur cinq (comprenant chacun cinq valeurs). Ces vecteurs représentent, par exemple, les réponses de plusieurs répondants à un questionnaire.



Il existe dans R une subtilité à l'origine de nombreux malentendus : la distinction entre un vecteur de type texte et un vecteur de type facteur. Dans l'exemple précédent, le vecteur `adresses` est un vecteur de type texte. Chaque nouvelle valeur ajoutée dans le vecteur peut être n'importe quelle nouvelle adresse. Déclarons un nouveau vecteur qui contiendrait cette fois-ci la couleur des yeux de personnes ayant répondu au questionnaire.

```
couleurs_yeux <- c('marron','marron','bleu','bleu','marron','vert')
```

Contrairement aux adresses, il y a un nombre limité de couleurs que nous pouvons mettre dans ce vecteur. Il serait intéressant de fixer les valeurs possibles du vecteur pour s'assurer que de nouvelles ne soient pas ajoutées par erreur. Pour cela, nous pouvons convertir ce vecteur texte en vecteur de type facteur avec la fonction `as.factor`.

```
couleurs_yeux_facteur <- as.factor(couleurs_yeux)
```

Notez qu'à présent, nous pouvons ajouter une nouvelle couleur dans le premier vecteur, mais pas dans le second.

```
couleurs_yeux[7] <- "rouge"
couleurs_yeux_facteur[7] <- "rouge"
```

```
## Warning in `[<-.factor`(`*tmp*`, 7, value = "rouge"): invalid factor level, NA
## generated
```

Le message d'erreur nous informe que nous avons tenté d'introduire une valeur invalide dans le facteur.

Les facteurs peuvent sembler restrictifs et très régulièrement, on préfère travailler avec de simples vecteurs de type texte plutôt que des facteurs. Cependant, de nombreuses fonctions d'analyse nécessitent d'utiliser des facteurs car ils assurent une certaine cohérence dans les données. Il est donc essentiel de savoir passer du texte au facteur avec la fonction **as.factor**. À l'inverse, il est parfois nécessaire de revenir à une variable de type texte avec la fonction **as.character**.

Notez que des vecteurs numériques peuvent aussi être convertis en facteurs :

```
tailles_facteur <- as.factor(tailles)
```

Cependant, si vous souhaitez reconvertis ce facteur en format numérique, il faudra passer dans un premier temps par le format texte :

```
as.numeric(tailles_facteur)
```

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

Comme vous pouvez le voir, convertir un facteur en valeur numérique renvoie des nombres entiers. Ceci est dû au fait que les valeurs dans un facteur sont recodées sous forme de nombres entiers, chaque nombre correspondant à une des valeurs originales (appelées niveaux). Si on convertit un facteur en valeurs numériques, on obtient donc ces nombres entiers.

```
as.numeric(as.character(tailles_facteur))
```

```
## [1] 175.5 180.3 168.2 172.8 167.6
```

Moralité de l'histoire, ne confondez pas les données de type texte et de type facteur. Dans le doute, vous pouvez demander à R quel est le type d'un vecteur avec la fonction **class**.

```
class(tailles)
```

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

```
class(tailles_facteur)
```

```
## [1] "factor"
```

```
class(couleurs_yeux)
```

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

```
class(couleurs_yeux_facteur)
## [1] "factor"
```

Quasiment toutes les fonctions utilisent des vecteurs. Par exemple, on pourrait calculer la moyenne du vecteur *ages* en utilisant la fonction *mean* présente de base dans R.

```
mean(ages)
```

```
## [1] 54
```

Cela démontre bien que le vecteur est la brique élémentaire de R ! Toutes les variables que nous avons déclarées dans les sections précédentes sont aussi des vecteurs, mais de longueur 1 !

1.3.6.2 Matrices

Il est possible de combiner des vecteurs pour former des matrices. Une matrice est un tableau en deux dimensions (colonnes et lignes) généralement utilisé pour représenter certaines structures de données comme des images (pixels), effectuer du calcul matriciel ou plus simplement présenter des matrices de corrélations. Vous aurez rarement à travailler directement avec des matrices, mais il est bon de savoir ce qu'elles sont. Créons deux matrices à partir de nos précédents vecteurs.

```
matrice1 <- cbind(ages,tailles)
# afficher la matrice 1
print(matrice1)

##      ages tailles
## [1,]    35   175.5
## [2,]    45   180.3
## [3,]    72   168.2
## [4,]    56   172.8
## [5,]    62   167.6

# afficher les dimensions de la matrice 1 (1er chiffre : lignes; 2e chiffre : colonnes)
print(dim(matrice1))

## [1] 5 2

matrice2 <- rbind(ages, tailles)
# afficher la matrice 2
print(matrice2)

##      [,1]  [,2]  [,3]  [,4]  [,5]
## ages    35.0  45.0  72.0  56.0  62.0
## tailles 175.5 180.3 168.2 172.8 167.6

# afficher les dimensions de la matrice 2
print(dim(matrice2))
```

```
## [1] 2 5
```

Comme vous pouvez le constater, la fonction `cbind` permet de concaténer des vecteurs comme s'ils étaient les colonnes d'une matrice, alors que `rbind` les combine comme s'ils étaient des lignes d'une matrice. La figure 1.12 présente graphiquement le passage du vecteur à la matrice.

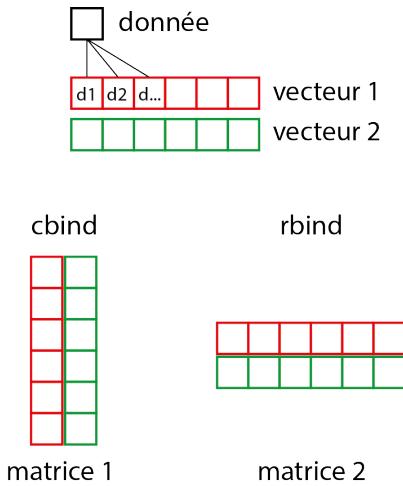


FIG. 1.12 : Du vecteur à la matrice

Notez que vous pouvez transposer une matrice avec la fonction `t`. Si nous essayons maintenant de comparer la matrice 1 et la matrice 2 nous allons avoir une erreur car elles n'ont pas les mêmes dimensions.

```
matrice1 == matrice2
```

Error in matrice1 == matrice2 : non-conformable arrays

En revanche, on pourrait transposer la matrice 1 et refaire cette comparaison :

```
t(matrice1) == matrice2
```

```
##      [,1] [,2] [,3] [,4] [,5]
## ages   TRUE TRUE TRUE TRUE TRUE
## tailles TRUE TRUE TRUE TRUE TRUE
```

Le résultat souligne bien que l'on a les mêmes valeurs dans les deux matrices. Il est aussi possible de construire des matrices directement avec la fonction `matrix`, ce que nous montrons dans la prochaine section.

1.3.6.3 Arrays

S'il est rare de travailler avec des matrices, il est encore plus rare de manipuler des *arrays*. Un *array* est une matrice spéciale qui peut avoir plus que deux dimensions. Un cas simple serait un *array* en trois dimensions : lignes, colonnes, profondeur, que l'on pourrait se représenter comme un cube divisé en sous cubes. Au delà de trois dimensions, il devient difficile de se les représenter. Cette structure de données peut être utilisée pour représenter les différentes bandes spectrales d'une image satellitaire. Les lignes et les colonnes délimiteraient les pixels de l'image, la profondeur quant à elle délimiterait les différents bandes composant l'image (figure 1.12).

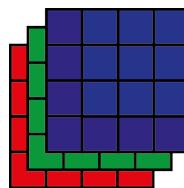


FIG. 1.13 : Un array avec trois dimension

Créons un array en combinant trois matrices avec la fonction `array`. Chacune de ces matrices sera composée respectivement de 1, de 2 et de 3 et aura une dimension de 5×5 . L'array final aura donc des dimensions de $5 \times 5 \times 3$.

```
mat1 <- matrix(1, nrow = 5, ncol = 5)
mat2 <- matrix(2, nrow = 5, ncol = 5)
mat3 <- matrix(3, nrow = 5, ncol = 5)

mon_array <- array(c(mat1, mat2, mat3), dim = c(5,5,3))

print(mon_array)
```

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

1.3.6.4 DataFrames

S'il est rare de manipuler des matrices et des arrays, le *DataFrame* (tableau de données en français) est la structure de données la plus souvent utilisée. Dans cette structure, chaque ligne du tableau représente un individu et chaque colonne représente une caractéristique de ces individus. Ces colonnes ont des noms, ce qui permet facilement d'accéder à leurs valeurs. Créons un *DataFrame* à partir de nos quatres vecteurs et de la fonction `data.frame`.

```
df <- data.frame(
  "age" = ages,
  "taille" = tailles,
  "adresse" = addresses,
  "proprietaire" = proprietaires
)
```

Dans Rstudio, vous pouvez visualiser votre tableau de données avec la fonction `View(df)`. Comme vous pouvez le constater, chaque vecteur est devenu une colonne de votre tableau de données *df*. La figure 1.14 résume ce passage d'une simple donnée à un DataFrame en passant par un vecteur.

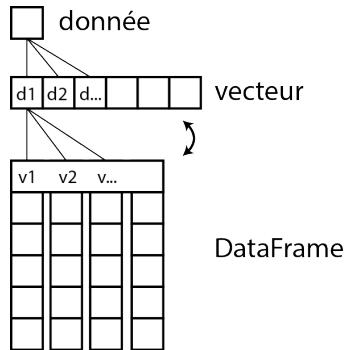


FIG. 1.14 : De la donnée au DataFrame

Plusieurs fonctions de base de R fournissent des informations importantes sur un *DataFrame* :

- `names` renvoie les noms des colonnes du DataFrame;
- `nrow` renvoie le nombre de lignes;
- `ncol` renvoie le nombre de colonnes.

```
names(df)
```

```
## [1] "age"          "taille"        "adresse"       "proprietaire"
```

TAB. 1.4 : Un premier DataFrame

age	taille	adresse	proprietaire
35	175,5	4225 rue de la gauchetiere	TRUE
45	180,3	4223 rue de la gauchetiere	TRUE
72	168,2	4221 rue de la gauchetiere	FALSE
56	172,8	4219 rue de la gauchetiere	TRUE
62	167,6	4217 rue de la gauchetiere	TRUE

```
nrow(df)
```

```
## [1] 5
```

```
ncol(df)
```

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

Vous pouvez accéder à chaque colonne de *df* en utilisant le symbole \$ ou `["nom_de_la_colonne"]`. Recalculons ainsi la moyenne des âges :

```
mean(df$age)
```

```
## [1] 54
```

```
mean(df[["age"]])
```

```
## [1] 54
```

1.3.6.5 Listes

La dernière structure de données à connaître est la liste. Elle ressemble à un vecteur, au sens où elle permet de stocker un ensemble d'objets les uns à la suite des autres. Cependant, une liste peut contenir n'importe quel type d'objets. Vous pouvez ainsi construire des listes de matrices, des listes d'*arrays*, des listes mixant des vecteurs, des graphiques, des *DataFrames*, des listes de listes...

Créons ensemble une liste qui va contenir des vecteurs et des matrices à l'aide de la fonction `list`.

```
ma_liste <- list(c(1,2,3,4),
                  matrix(1, ncol = 3, nrow = 5),
                  matrix(5, ncol = 3, nrow = 7),
                  'A'
                 )
```

Il est possible d'accéder aux éléments de la liste par leur position dans cette dernière en utilisant les doubles crochets `[[]]` :

```
print(ma_liste[[1]])
```

```
## [1] 1 2 3 4
```

```
print(ma_liste[[4]])
```

```
## [1] "A"
```

Il est aussi possible de donner des noms aux éléments de la liste et d'utiliser le symbole \$ pour y accéder. Créons une nouvelle liste de vecteurs et donnons leurs des noms avec la fonction `names`.

```

liste2 <- list(c(35,45,72,56,62),
               c(175.5,180.3,168.2,172.8,167.6),
               c(TRUE,TRUE,FALSE,TRUE,TRUE))
)
names(liste2) <- c("age",'taille','proprietaire')

print(liste2$age)

```

```
## [1] 35 45 72 56 62
```

Si vous avez bien suivi, vous devez avoir compris qu'un *DataFrame* n'est en fait rien d'autre qu'une liste de vecteurs avec des noms!

Bravo! Vous venez de faire le tour des bases du langage R. Vous allez apprendre désormais à manipuler des données dans des *DataFrames*!

1.4 Manipuler des données

Dans cette section, vous apprendrez à charger et manipuler des *DataFrames* en vue d'effectuer des opérations classiques de gestion de données.

1.4.1 Charger un *DataFrame* depuis un fichier

Il sera rarement nécessaire de créer vos *DataFrames* manuellement comme réalisé dans la section précédente. Le plus souvent, vous disposerez de fichiers contenant vos données et utiliserez des fonctions pour les importer dans R sous forme d'un *DataFrame*. Les formats à importer les plus répandus sont :

- *.csv*, soit un fichier texte dont chaque ligne représente une ligne du tableau de données et dont les colonnes sont séparées par un délimiteur (généralement une virgule ou un point-virgule).
- *.dbf*, ou fichier *dBase*, souvent associés à des fichiers d'information géographique au format *Shape-File*.
- *.xls* et *.xlsx*, soit des fichiers générés par Excel.
- *.json*, soit un fichier texte utilisant la norme d'écriture propre au langage JavaScript.

Plus rarement, il se peut que vous ayez à charger des fichiers provenant de logiciels propriétaires :

- *.sas7bdat* (SAS),
- *.sav* (SPSS) et
- *.dta* (STATA).

Pour lire la plupart de ces fichiers, nous allons utiliser le package **foreign** dédié à l'importation d'une multitude de formats. Commencez donc par l'installer (`install.packages("foreign")`). Nous allons charger cinq fois le même jeu de données enregistré dans des formats différents (*csv*, *dbf*, *dta*, *sas7bdat* et *xlsx*). Aussi, nous mesurerons le temps nécessaire pour importer chacun de ces fichiers avec la fonction `system.time`.

1.4.1.1 Lire un fichier *csv*

Pour le format *csv*, il n'y a pas besoin d'utiliser un package puisque R dispose d'une fonction de base pour lire ce format.

```
t1 <- Sys.time()
df1 <- read.csv("data/priseenmain/SR_MTL_2016.csv",
                 header = TRUE, sep = ",", dec = ".",
                 stringsAsFactors = FALSE)
t2 <- Sys.time()
d1 <- as.numeric(difftime(t2,t1,units="secs"))

cat("le dataframe df1 a ",nrow(df1),' observations',
    'et ',ncol(df1),"colonnes\n")
```

le dataframe df1 a 951 observations et 48 colonnes

Rien de bien compliqué! Notez tout de même que :

- Lorsque vous chargez un fichier *csv*, vous devez connaître le **séparateur**, soit le caractère utilisé pour délimiter les colonnes. Dans le cas présent, il s'agit d'une virgule (spécifiez avec l'argument *sep* = ","), mais il pourrait tout aussi bien être un point virgule (*sep* = ";") une tabulation (*sep* = " "), etc.
- Vous devez également spécifier le caractère utilisé comme séparateur de décimales. Le plus souvent, ce sera le point (*dec* = "."), mais certains logiciels avec des paramètres régionaux de langue française (notamment Excel) exportent des fichiers *csv* avec des virgules comme séparateur de décimales (utilisez alors *dec* = ",").
- L'argument *header* indique si la première ligne (l'entête) du fichier comprend ou non les noms des colonnes du jeu de données (avec les valeurs **TRUE** ou **FALSE**). Il arrive que certains fichiers *csv* soient fournis sans entête et que les noms et descriptions des colonnes soient fournis dans un autre fichier.
- L'argument *stringsAsFactors* permet d'indiquer à R que les colonnes comportant du texte doivent être chargées comme des vecteurs de type texte et nom de type facteur.

1.4.1.2 Lire un fichier *dbase*

Pour lire un fichier *dbase* (.dbf), nous utilisons la fonction *read.dbf* du package **foreign** installé précédemment :

```
library(foreign)

t1 <- Sys.time()
df2 <- read.dbf("data/priseenmain/SR_MTL_2016.dbf")
t2 <- Sys.time()
d2 <- as.numeric(difftime(t2,t1,units="secs"))

cat("le dataframe df2 a ",nrow(df2)," observations",
    "et ",ncol(df2),"colonnes\n")
```

le dataframe df2 a 951 observations et 48 colonnes

Comme vous pouvez le constater, nous obtenons les mêmes résultats qu'avec le fichier *csv*.

1.4.1.3 Lire un fichier *dta* (Stata)

Si vous travaillez avec des collègues utilisant le logiciel Stata, il se peut que ces derniers vous partagent des fichiers *dta*. Toujours en utilisant le package **foreign**, vous serez en mesure de les charger directement dans R.

```
t1 <- Sys.time()
df3 <- read.dta("data/priseenmain/SR_MTL_2016.dta")
t2 <- Sys.time()
d3 <- as.numeric(difftime(t2,t1,units="secs"))

cat("le dataframe df3 a ",nrow(df3)," observations",
"et ",ncol(df3),"colonnes\n", sep = "")

## le dataframe df3 a 951 observationset 48colonnes
```

1.4.1.4 Lire un fichier *sav* (SPSS)

Pour importer un fichier *sav* provenant du logiciel statistique SPSS, utilisez la fonction `read.spss` du package **foreign**.

```
t1 <- Sys.time()
df4 <- as.data.frame(read.spss("data/priseenmain/SR_MTL_2016.sav"))
t2 <- Sys.time()
d4 <- as.numeric(difftime(t2,t1,units="secs"))

cat("le dataframe df4 a ",nrow(df4)," observations",
"et ",ncol(df4),"colonnes\n", sep = "")

## le dataframe df4 a 951 observationset 48colonnes
```

1.4.1.5 Lire un fichier *sas7bdat* (SAS)

Pour importer un fichier *sas7bdat* provenant du logiciel statistique SAS, utilisez la fonction `read.sas7bdat` du package **sas7bdat**. Installez préalablement le package (`install.packages("sas7bdat")`) et chargez le (`library(sas7bdat)`).

```
library(sas7bdat)

t1 <- Sys.time()
df5 <- read.sas7bdat("data/priseenmain/SR_MTL_2016.sas7bdat")
t2 <- Sys.time()
d5 <- as.numeric(difftime(t2,t1,units="secs"))

cat("le dataframe df5 a ",nrow(df5)," observations",
"et ",ncol(df5),"colonnes\n", sep = "")

## le dataframe df5 a 951 observationset 48colonnes
```

1.4.1.6 Lire un fichier *xlsx* (Excel)

Lire un fichier Excel dans R n'est pas toujours une tâche facile. Généralement, nous recommandons d'exporter les fichiers en question au format *csv* dans un premier temps, puis de le lire avec la fonction `read.csv` dans un second temps (section 1.4.1.1). Il est néanmoins possible de lire directement un fichier *xlsx* avec le package **xlsx**. Ce dernier requiert que le logiciel JAVA soit installé sur votre ordinateur (Windows, Mac ou Linux). Si vous utilisez la version 64 bit de R, vous devrez télécharger et installer la

version 64 bit de JAVA. Une fois que ce logiciel tiers est installé, il ne vous restera plus qu'à installer (`install.packages("xlsx")`) et charger (`library(xlsx)`) le package `xlsx`.

```
library(xlsx)

t1 <- Sys.time()
df6 <- read.xlsx(file="data/priseenmain/SR_MTL_2016.xlsx",
                  sheetIndex = 1,
                  as.data.frame = TRUE)
t2 <- Sys.time()
d6 <- as.numeric(difftime(t2,t1,units="secs"))

cat("le dataframe df6 a ",nrow(df6)," observations",
    "et ",ncol(df6)," colonnes\n", sep = "")

## le dataframe df6 a 951 observationset 48colonnes
```

Il est possible d'accélérer significativement la vitesse de lecture d'un fichier `xlsx` en utilisant la fonction `read.xlsx2`. Il faut cependant indiquer à cette dernière le type de données de chaque colonne. Dans le cas présent, les cinq premières colonnes contiennent des données de type texte (`character`), alors que les 43 autres sont des données numériques (`numeric`). Nous utilisons la fonction `rep` afin de ne pas avoir à écrire plusieurs fois `character` et `numeric`.

```
library(xlsx)

t1 <- Sys.time()
df7 <- read.xlsx2(file="data/priseenmain/SR_MTL_2016.xlsx",
                  sheetIndex = 1,
                  as.data.frame = TRUE,
                  colClasses = c(rep("character",5),rep("numeric",43)))
)
t2 <- Sys.time()
d7 <- as.numeric(difftime(t2,t1,units="secs"))

cat("le dataframe df6 a ",nrow(df7)," observations",
    "et ",ncol(df7)," colonnes\n", sep = "")

## le dataframe df6 a 951 observationset 48colonnes
```

Si l'on compare les temps d'exécution (tableau 1.5), on constate que la lecture des fichiers `xlsx` peut être extrêmement longue si l'on ne spécifie pas le type des colonnes, ce qui peut devenir problématique pour des fichiers volumineux. Notez également que la lecture des fichiers `csv` devient de plus en plus laborieuse à mesure que la taille du fichier `csv` augmente. Si vous devez un jour charger des fichiers `csv` de plusieurs gigaoctets, nous vous recommandons vivement d'utiliser la fonction `fread` du package `data.table` qui est beaucoup plus rapide.

1.4.2 Manipuler un *DataFrame*

Une fois le *DataFrame* chargé, voyons comment il est possible de le manipuler.

TAB. 1.5 : Temps nécessaire pour lire les données en fonction du type de fichiers

Durée (s)	fonction
0,06	read.csv
0,06	read.dbf
0,02	read.spss
0,03	read.dta
0,86	read.sas7bdat
20,09	read.xlsx
0,53	read.xlsx2

1.4.2.1 Un petit mot sur le tidyverse

Tidyverse est un ensemble de *packages* conçus pour faciliter la structuration et la manipulation des données dans R. Avant d'aller plus loin, il est important d'aborder brièvement un débat actuel dans la Communauté R. Entre 2010 et 2020, l'utilisation du **tidyverse** s'est peu à peu répandue. Développé et maintenu par Hadley Wickham, **tidyverse** introduit une philosophie et une grammaire spécifiques qui diffèrent du langage R traditionnel. Une partie de la communauté a pour ainsi dire complètement embrassé le **tidyverse** et de nombreux *packages* en dehors du **tidyverse** ont adopté sa grammaire et sa philosophie. À l'inverse, une autre partie de la communauté est contre cette évolution (voir l'article du blogue suivant¹¹). Les arguments pour et contre **tidyverse** sont résumés dans le tableau suivant.

Le dernier point est probablement le plus problématique. Dans sa volonté d'évoluer au mieux et sans restriction, le *package* **tidyverse** n'offre aucune garantie de rétro-compatibilité. En d'autre termes, des changements importants peuvent être introduits d'une version à l'autre rendant potentiellement obsolète votre propre code. Nous n'avons pas d'opinion tranchée sur le sujet : **tidyverse** est un outil très intéressant dans de nombreux cas ; nous évitons simplement de l'utiliser systématiquement et préférons charger directement des sous-packages (comme **dplyr** ou **ggplot2**) du **tidyverse**. Notez que le *package* **data.table** offre une alternative au **tidyverse** dans la manipulation de données. Au prix d'une syntaxe généralement un peu plus complexe, le *package* **data.table** offre une vitesse de calcul bien supérieure au **tidyverse** et assure une bonne rétro-compatibilité.

1.4.2.2 Gérer les colonnes d'un *DataFrame*

Repartons du *DataFrame* que nous avions chargé précédemment en important un fichier *csv*.

¹¹<https://blog.ephorie.de/why-i-dont-use-the-tidyverse>

TAB. 1.6 : Avantages et inconvénients du tidyverse

Avantage du tidyverse	Problème posé par le tidyverse
Simplicité d'écriture et d'apprentissage	Nouvelle syntaxe à apprendre
Ajout de l'opérateur %>% permettant d'enchaîner les traitements	Perte de lisibilité avec l'opérateur ->
La meilleure librairie pour réaliser des graphiques : ggplot2	Certaines fonctions de base sont remplacées par tidyverse lors de son chargement, pouvant créer des erreurs.
Crée un écosystème cohérent	Ajoute une dépendance dans le code
Package en développement et de plus en plus utilisé	Philosophie d'évolution agressive, aucune assurance de rétro-compatibilité

```
df <- read.csv(file="data/priseenmain/SR_MTL_2016.csv",
               header = TRUE, sep = ",", dec = ".",
               stringsAsFactors = FALSE)
```

1.4.2.2.1 Sélectionner une colonne

Pour rappel, il est possible d'accéder aux colonnes dans ce *DataFrame* en utilisant le symbole dollar `$ma_colonne` ou les doubles crochets `["ma_colonne"]`.

```
# Calcul de la superficie totale de l'île de Montréal
sum(df$KM2)
```

```
## [1] 4680.543
```

```
sum(df[["KM2"]])
```

```
## [1] 4680.543
```

1.4.2.2.2 Sélectionner plusieurs colonnes

Il est possible de sélectionner plusieurs colonnes d'un *DataFrame* et filtrer ainsi les colonnes inutiles. Pour cela, on peut utiliser un vecteur contenant soit les positions des colonnes (1 pour la première colonne, 2 pour la seconde et ainsi de suite), soit les noms des colonnes.

```
# Conserver les 5 premières colonnes
df2 <- df[1:5]

# Conserver les colonnes 1, 5, 10 et 15
df3 <- df[c(1,5,10,15)]

# Cela peut aussi être utilisé pour changer l'ordre des champs
df3 <- df[c(10,15,1,5)]

# Conserver les colonnes 1 à 5, 7 à 12, 17 et 22
df4 <- df[c(1:5,7:12,17,22)]

# Conserver les colonnes avec leurs noms
df5 <- df[c("SRIDU","KM2","Pop2016","MaisonIndi","LoyerMed")]
```

1.4.2.2.3 Supprimer des colonnes

Il est parfois plus intéressant et rapide de supprimer directement des colonnes plutôt que de recréer un nouveau *DataFrame*. Pour ce faire, on attribue la valeur `NULL` à ces colonnes.

```
# Supprimer les colonnes 2, 3 et 5
df3[c(2,3,5)] <- list(NULL)

# Supprimer une colonne avec son nom
df4$OID <- NULL
```

```
# Supprimer plusieurs colonnes par leur nom
df5[c("SRIDU", "LoyerMed")] <- list(NULL)
```

Notez que si vous supprimez une colonne, vous ne pouvez pas revenir en arrière. Il faudra recharger votre jeu de données ou éventuellement relancer les calculs qui avaient produit cette colonne.

1.4.2.2.4 Renommer des colonnes

Il est possible de changer le nom d'une colonne. Cette opération est importante pour faciliter la lecture du *DataFrame* ou encore s'assurer que l'exportation du *DataFrame* dans un format ne posera pas de problème.

```
# Voici les noms des colonnes
names(df5)

## [1] "KM2"          "Pop2016"       "MaisonIndi"

# Renommer toutes les colonnes
names(df5) <- c('superficie_km2', 'population_2016', 'maison_individuelle_prt')
names(df5)

## [1] "superficie_km2"      "population_2016"
## [3] "maison_individuelle_prt"

# Renommer avec dplyr
library(dplyr)
df4 <- rename(df4, "population_2016" = "Pop2016",
               "prs_moins_14ans_prt" = "A014",
               "prs_15_64_ans_prt" = "A1564",
               "prs_65plus_ans_prt" = "A65plus"
               )
```

1.4.2.3 Calculer de nouvelles variables

Il est possible d'utiliser les colonnes de type numérique pour calculer de nouvelles colonnes en utilisant les opérateurs mathématiques vus dans la section 1.3.5. Prenons un exemple concret : calculons la densité de population par secteur de recensement dans notre *DataFrame*, puis affichons un résumé de cette nouvelle variable.

```
# Calcul de la densité
df$pop_density_2016 <- df$Pop2016 / df$KM2

# Statistiques descriptives
summary(df$pop_density_2016)

##      Min.   1st Qu.   Median   Mean   3rd Qu.   Max.
## 17.45  1946.96  3700.50  5465.03  7918.39 48811.79
```

Nous pouvons aussi calculer le ratio entre le nombre de maisons et le nombre d'appartements.

```
# Calcul du ratio
df$total_maison <- (df$MaisonIndi + df$MaisJumule + df$MaisRangee + df$AutreMais)
df$total_apt <- (df$AppDuplex + df$App5Moins + df$App5Plus)
df$ratio_maison_apt <- df$total_maison / df$total_apt
```

Retenez ici que R va appliquer le calcul à chaque ligne de votre jeu de données et stocker le résultat dans une nouvelle colonne. Cette opération est du calcul vectoriel : toute la colonne est calculée en une seule fois. R est d'ailleurs optimisé pour le calcul vectoriel.

1.4.2.4 Fonctions mathématiques

R propose un ensemble de fonctions de base pour effectuer du calcul. Voici une liste non-exhaustive des principales fonctions :

- `abs` calcule les valeurs absolues des valeurs d'un vecteur
- `sqrt` calcule les racines carrées des valeurs d'un vecteur
- `log` calcule les logarithmes des valeurs d'un vecteur
- `exp` calcule les exponentielles des valeurs d'un vecteur
- `factorial` calcule la factorielle des valeurs d'un vecteur
- `round` arrondit les valeurs d'un vecteur
- `ceiling, floor` arrondit à l'unité supérieure ou inférieure les valeurs d'un vecteur
- `sin, asin, cos, acos, tan, atan` sont des fonctions de trigonométrie
- `cumsum` calcule la somme cumulative des valeurs d'un vecteur.

Ces fonctions sont des fonctions vectorielles puisqu'elles s'appliquent à tous les éléments d'un vecteur. Si votre vecteur en entrée comprend cinq valeurs, le vecteur en sortie comprendra aussi cinq valeurs.

À l'inverse, les fonctions suivantes s'appliquent directement à l'ensemble d'un vecteur et ne vont renvoyer qu'une seule valeur :

- `sum` calcule la somme des valeurs d'un vecteur
- `prod` calcule le produit des valeurs d'un vecteur
- `min, max` renvoient les valeurs maximale et minimale d'un vecteur
- `mean, median` renvoient la moyenne et la médiane d'un vecteur
- `quantile` renvoie les percentiles d'un vecteur.

1.4.2.5 Fonctions pour manipuler des chaînes de caractères

Outre les données numériques, vous aurez à travailler avec des données de type texte (`string`). Le `tidyverse` avec le package `stringr` offre des fonctions très intéressantes pour manipuler ce type de données. Pour un aperçu de toutes les fonctions offertes par `stringr`, référez-vous à sa *Cheat Sheet*¹². Commençons avec un `DataFrame` assez simple comprenant des adresses et des noms de personnes.

```
library(stringr)

df <- data.frame(
  noms = c("Jérémie Toutanplace", "constant Tinople", "dino Resto", "Luce tancil"),
  adresses = c('15 rue Levy', '413 Blvd Saint-Laurent', '3606 rue Duké', '2457 route St Marys')
)
```

¹²<https://github.com/rstudio/cheatsheets/blob/master/strings.pdf>

1.4.2.5.1 Majuscules et minuscules

Pour harmoniser ce *dataframe*, nous allons dans un premier temps mettre des majuscules au premier caractère des prénoms et noms des individus avec la fonction `str_to_title`.

```
df$noms_corr <- str_to_title(df$noms)
print(df$noms_corr)

## [1] "Jérémie Toutanplace" "Constant Tinople"    "Dino Resto"
## [4] "Luce Tancil"
```

On pourrait également tout mettre en minuscules ou tout en majuscules.

```
df$noms_min <- tolower(df$noms)
df$noms_maj <- toupper(df$noms)
print(df$noms_min)

## [1] "jérémie toutanplace" "constant tinople"    "dino resto"
## [4] "luce tancil"

print(df$noms_maj)

## [1] "JÉRÉMIE TOUTANPLACE" "CONSTANT TINOPLE"    "DINO RESTO"
## [4] "LUCE TANCIL"
```

1.4.2.5.2 Remplacer du texte

Les adresses comprennent des caractères accentués. Ce type de caractères pose régulièrement des problèmes d'encodage. Nous pourrions alors décider de les remplacer par des caractères simples avec la fonction `str_replace_all`.

```
df$adresses_1 <- str_replace_all(df$adresses, 'é', 'e')
print(df$adresses_1)

## [1] "15 rue Levy"           "413 Blvd Saint-Laurent" "3606 rue Duke"
## [4] "2457 route St Marys"
```

La même fonction peut être utilisée pour remplacer les *St* par *Saint* et les *Blvd* par *Boulevard*.

```
df$adresses_2 <- str_replace_all(df$adresses_1, ' St ', ' Saint ')
df$adresses_3 <- str_replace_all(df$adresses_2, ' Blvd ', ' Boulevard ')
print(df$adresses_3)

## [1] "15 rue Levy"           "413 Boulevard Saint-Laurent"
## [3] "3606 rue Duke"         "2457 route Saint Marys"
```

1.4.2.5.3 Découper du texte

Il est parfois nécessaire de découper du texte pour en extraire des éléments. On doit alors choisir un caractère de découpage. Dans notre exemple, on pourrait vouloir extraire les numéros civiques des adresses, en utilisant le premier espace comme caractère de découpage, en utilisant la fonction `str_split_fixed`.

```
df$num_civique <- str_split_fixed(df$adresses_3, ' ', n=2) [,1]
print(df$num_civique)

## [1] "15"   "413"  "3606" "2457"
```

Pour être exact, sachez que pour notre exemple, la fonction `str_split_fixed` renvoie deux colonnes de texte : une avec le texte avant le premier espace, soit le numéro civique, et une avec le reste du texte. Le nombre de colonnes est contrôlé par l'argument `n`. Si `n = 1`, la fonction ne fait aucun découpage, avec `n = 2` la fonction va découper en deux parties le texte avec la première occurrence du délimiteur, et ainsi de suite. En ajoutant `[,1]` à la fin, nous indiquons que l'on souhaite garder seulement la première des deux colonnes.

Il est également possible d'extraire des parties de texte et de ne garder par exemple que les N premiers caractères ou les N derniers caractères :

```
# ne garder que les 5 premiers caractères
substr(df$adresses_3, start = 1, stop = 5)

## [1] "15 ru" "413 B" "3606" "2457"

# ne garder que les 5 derniers caractères
n_caract <- nchar(df$adresses_3)
substr(df$adresses_3, start = n_caract-4, stop = n_caract)

## [1] " Levy" "urent" " Duke" "Marys"
```

Notez que les paramètres `start` et `stop` de la fonction `substr` peuvent accepter un vecteur de valeurs. Il est ainsi possible d'appliquer une sélection de texte différente à chaque chaîne de caractères dans notre vecteur en entrée. On pourrait par exemple vouloir récupérer tout le texte avant le second espace pour garder uniquement le numéro civique et le type de rue.

```
# étape 1 : récupérer les positions des espaces pour chaque adresses
positions <- str_locate_all(df$adresses_3, " ")

# étape 2 : récupérer les positions des seconds espaces
sec_positions <- sapply(positions, function(i){
  i[2,1]
})

# étape 3 : appliquer le découpage
substr(df$adresses_3, start = 1, stop = sec_positions-1)

## [1] "15 rue"        "413 Boulevard"  "3606 rue"      "2457 route"
```

1.4.2.5.4 Coller du texte

À l'inverse du découpage, il est parfois nécessaire de concaténer des éléments de texte, ce qu'il est possible de réaliser avec la fonction `paste`.

```
df$texte_complet <- paste(df$noms_corr, df$adresses_3, sep = " : ")
print(df$texte_complet)
```

```
## [1] "Jérémie Toutanplace : 15 rue Levy"
## [2] "Constant Tinople : 413 Boulevard Saint-Laurent"
## [3] "Dino Resto : 3606 rue Duke"
## [4] "Luce Tancil : 2457 route Saint Marys"
```

Le paramètre `sep` permet d'indiquer le ou les caractères à intercaler entre les éléments à concaténer. Notez qu'il est possible de concaténer plus que deux éléments.

```
df$ville <- c('Montreal','Montreal','Montreal','Montreal')
paste(df$noms_corr, df$adresses_3, df$ville, sep = ", ")
```

```
## [1] "Jérémie Toutanplace, 15 rue Levy, Montreal"
## [2] "Constant Tinople, 413 Boulevard Saint-Laurent, Montreal"
## [3] "Dino Resto, 3606 rue Duke, Montreal"
## [4] "Luce Tancil, 2457 route Saint Marys, Montreal"
```

1.4.2.6 Manipuler des colonnes de type date

Nous avons vu que les principaux types de données dans R sont le numérique, le texte, le booléen et le facteur. Il existe d'autres types introduits par différents *packages*. Nous abordons ici les types date et heure (*date and time*). Pour les manipuler, nous privilégions l'utilisation du *package lubridate* du *tidyverse*. Pour illustrer le tout, nous l'appliquerons avec un jeu de données ouvertes de la Ville de Montréal représentant les collisions routières impliquant au moins un cycliste survenues après le 1^{er} janvier 2017.

```
accidents_df <- read.csv(file="data/priseenmain/accidents.csv", sep = ", ")
names(accidents_df)
```

```
## [1] "HEURE_ACCDN"          "DT_ACCDN"           "NB_VICTIMES_TOTAL"
```

Nous disposons de trois colonnes représentant respectivement l'heure, la date et le nombre de victimes impliquées dans la collision.

1.4.2.6.1 Du texte à la date

Actuellement, les colonnes *HEURE_ACCDN* et *DT_ACCDN* sont au format texte. Nous pouvons afficher quelques lignes du jeu de données avec la fonction `head` pour visualiser comment elles ont été saisies.

```
head(accidents_df, n = 5)
```

```
##          HEURE_ACCDN    DT_ACCDN NB_VICTIMES_TOTAL
## 1 16:00:00-16:59:00 2017/11/02            0
## 2 06:00:00-06:59:00 2017/01/16            1
## 3 18:00:00-18:59:00 2017/04/18            0
## 4 11:00:00-11:59:00 2017/05/28            1
## 5 15:00:00-15:59:00 2017/05/28            1
```

Un peu de ménage s'impose : les heures sont indiquées comme des périodes d'une heure. Nous utilisons la fonction `str_split_fixed` du *package stringr* pour ne garder que la première partie de l'heure (avant le tiret). Nous allons ensuite concaténer l'heure et la date avec la fonction `paste`, puis nous convertirons ce résultat en un objet *date-time*.

```

library(lubridate)

# Étape 1 : découper la colonne Heure_ACCDN
accidents_df$heure <- str_split_fixed(accidents_df$HEURE_ACCDN, "-", n=2) [,1]

# Étape 2 : concaténer l'heure et la date
accidents_df$date_heure <- paste(accidents_df$DT_ACCDN,
                                    accidents_df$heure,
                                    sep = ' ')

# Étape 3 : convertir au format datetime
accidents_df$datetime <- as_datetime(accidents_df$date_heure,
                                       format = "%Y/%m/%d %H:%M:%S")

```

Pour effectuer la conversion, nous avons utilisé la fonction `as_datetime` du package **lubridate**. Elle prend comme paramètre un vecteur de texte et une indication du format de ce vecteur de texte. Il existe de nombreuses façons de spécifier une date et une heure et l'argument `format` permet d'indiquer celle à utiliser. Dans cet exemple, la date est structurée comme suit : année/mois/jour heure:minute:seconde, ce qui se traduit par le format `%Y/%m/%d %H:%M:%S`.

- `%Y` signifie une année indiquée avec quatre caractères : 2017
- `%m` signifie un mois, indiqué avec deux caractères : 01, 02, 03, ... 12
- `%d` signifie un jour, indiqué avec deux caractères : 01, 02, 03, ... 31
- `%H` signifie une heure, au format 24 heures avec deux caractères : 00, 02, ... 23
- `%M` signifie des minutes indiquées avec deux caractères : 00, 02, ... 59
- `%S` signifie des secondes, indiquées avec deux caractères : 00, 02, ... 59

Notez que les caractères séparant les années, jours, heures, etc. sont aussi à indiquer dans le format. Dans notre exemple, nous utilisons des / pour séparer les éléments de la date et des : pour l'heure, et un espace pour séparer la date et l'heure.

Il existe d'autres nomenclatures pour spécifier un format `datetime` : par exemple, des mois renseignés par leur nom, l'indication AM-PM, etc. Vous pouvez vous référer à la documentation de la fonction `strptime` (`help(strptime)`) pour explorer les différentes nomenclatures et choisir celle qui vous convient. Bien évidemment, il est **nécessaire** que toutes les dates de votre colonne soient renseignées dans le même format. Sinon, la fonction renverra des valeurs NA aux endroits où elle a échoué à lire le format. Après toutes ces opérations, rejettons un oeil à notre `DataFrame`.

```
head(accidents_df, n = 5)
```

```

##           HEURE_ACCDN    DT_ACCDN NB_VICTIMES_TOTAL      heure      date_heure
## 1 16:00:00-16:59:00 2017/11/02          0 16:00:00 2017/11/02 16:00:00
## 2 06:00:00-06:59:00 2017/01/16          1 06:00:00 2017/01/16 06:00:00
## 3 18:00:00-18:59:00 2017/04/18          0 18:00:00 2017/04/18 18:00:00
## 4 11:00:00-11:59:00 2017/05/28          1 11:00:00 2017/05/28 11:00:00
## 5 15:00:00-15:59:00 2017/05/28          1 15:00:00 2017/05/28 15:00:00
##           datetime
## 1 2017-11-02 16:00:00
## 2 2017-01-16 06:00:00
## 3 2017-04-18 18:00:00
## 4 2017-05-28 11:00:00
## 5 2017-05-28 15:00:00

```

1.4.2.6.2 Extraire des informations d'une date

À partir de la nouvelle colonne `datetime`, nous sommes en mesure d'extraire des informations intéressantes comme :

- le nom du jour de la semaine avec la fonction `weekdays`

```
accidents_df$jour <- weekdays(accidents_df$datetime)
```

- la période de la journée avec les fonctions `am` et `pm`

```
accidents_df$AM <- am(accidents_df$datetime)
accidents_df$PM <- pm(accidents_df$datetime)
head(accidents_df[c("jour", "AM", "PM")], n=5)
```

```
##      jour     AM     PM
## 1 jeudi FALSE  TRUE
## 2 lundi  TRUE FALSE
## 3 mardi FALSE  TRUE
## 4 dimanche  TRUE FALSE
## 5 dimanche FALSE  TRUE
```

Il est aussi possible d'accéder aux sous-éléments d'un `datetime` comme l'année, le mois, le jour, l'heure, la minute, la seconde avec les fonctions `year()`, `month()`, `day()`, `hour()`, `minute()` et `second()`.

1.4.2.6.3 Calculer une durée entre deux `datetime`

Une autre utilisation intéressante du format `datetime` est de calculer des différences de temps. Par exemple, nous pourrions utiliser le nombre de minutes écoulées depuis 7h00 le matin comme une variable dans une analyse visant à déterminer le moment critique des collisions routières durant l'heure de pointe du matin. Pour cela, nous devons créer un `datetime` de référence en concaténant la date de chaque observation, et le temps `07:00:00` qui sera notre point de départ.

```
accidents_df$date_heure_07 <- paste(accidents_df$DT_ACCDN,
                                         '07:00:00',
                                         sep = ' ')
accidents_df$ref_datetime <- as_datetime(accidents_df$date_heure_07,
                                           format = "%Y/%m/%d %H:%M:%S")
```

Il ne nous reste plus qu'à calculer la différence de temps entre la colonne `datetime` et notre temps de référence `ref_datetime`.

```
accidents_df$diff_time <- difftime(accidents_df$datetime,
                                         accidents_df$ref_datetime,
                                         units = 'min')
```

Notez qu'ici la colonne `diff_time` est d'un type spécial : une différence temporelle (`difftime`). Il faut encore la convertir au format numérique pour pouvoir l'utiliser avec la fonction `as.numeric`. Par curiosité, réalisons rapidement un histogramme avec la fonction `hist` pour analyser rapidement cette variable d'écart de temps !

```
accidents_df$diff_time_num <- as.numeric(accidents_df$diff_time)
hist(accidents_df$diff_time_num, breaks = 50)
```

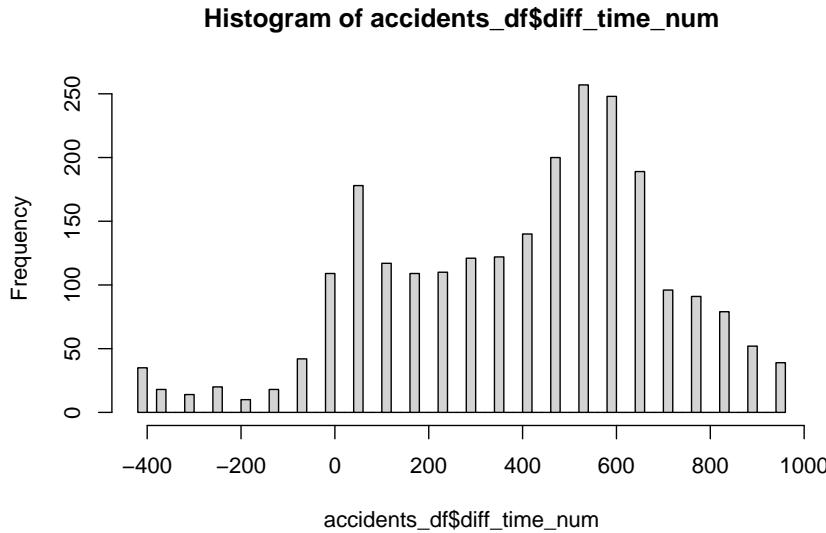


FIG. 1.15 : Répartition temporelle des accidents à vélo

On observe clairement deux pics, un premier entre 0 et 100 (entre 07h00 08h30 environ) et un second plus important entre 550 et 650 (entre 16h00 et 17h30 environ), ce qui correspond sans surprise aux heures de pointe. Il est intéressant de noter que plus d'accidents se produisent à l'heure de pointe du soir qu'à celle du matin.

1.4.2.6.4 Tenir compte du fuseau horaire

Lorsque l'on travaille avec des données provenant de différents endroits dans le monde ou que l'on doive tenir compte des heures d'été et d'hiver, il convient de tenir compte du fuseau horaire. Pour créer une date avec un fuseau horaire, il est possible d'utiliser le paramètre `tz` dans la fonction `as_datetime` et d'utiliser l'identifiant du fuseau approprié. Dans notre cas, les données d'accident ont été collectées à Montréal qui a un décalage de -5 heures par rapport au temps de référence UTC (+1 heure en été). Le code spécifique de ce fuseau horaire est *EDT*, il est facile de trouver ces codes avec le site web [timeanddate.com](https://www.timeanddate.com/time/map/)¹³.

```
accidents_df$datetime <- as_datetime(accidents_df$date_heure,
                                         format = "%Y/%m/%d %H:%M:%S",
                                         tz = "EDT")
```

1.4.2.7 Recoder des variables

Recoder des variables signifie changer la valeur d'une variable selon une condition afin d'obtenir une nouvelle variable. Si nous reprenons le jeu de données précédent sur les accidents à vélo, nous pourrions vouloir créer une nouvelle colonne nous indiquant si la collision a eu lieu en heures de pointe ou non. On obtiendrait ainsi une nouvelle variable avec seulement deux catégories plutôt que la variable numérique

¹³ <https://www.timeanddate.com/time/map/>

originale. Nous pourrions aussi définir quatre catégories avec l'heure de pointe du matin, l'heure de pointe du soir, le reste de la journée et la nuit.

1.4.2.7.1 Le cas binaire avec `ifelse`

Si l'on ne souhaite créer que deux catégories, le plus simple est d'utiliser la fonction `ifelse`. Cette fonction va évaluer une condition (section 1.3.5) pour chaque ligne d'un *DataFrame* et produire un nouveau vecteur. Créons donc une variable binaire indiquant si une collision a eu lieu durant les heures de pointe ou hors heures de pointe. Nous devons alors évaluer les conditions suivantes :

Est-ce que l'accident a eu lieu entre 07h00 (0) ET 09h00 (120), OU est ce que la collision a eu lieu entre 16h30 (570) ET 18h30 (690)?

```
table(is.na(accidents_df$diff_time_num))
```

```
##  
## FALSE TRUE  
## 2414    40
```

Notons dans un premier temps que nous avons 40 observations sans valeurs pour la colonne `diff_time_num`. Il s'agit d'observations pour lesquelles nous ne disposons pas de dates au départ.

```
Cond1 <- accidents_df$diff_time_num >= 0 & accidents_df$diff_time_num <= 120  
Cond2 <- accidents_df$diff_time_num >= 570 & accidents_df$diff_time_num <= 690  
  
accidents_df$moment_bin <- ifelse(Cond1 | Cond2,  
                                     "en heures de pointe",  
                                     "hors heures de pointe")
```

Comme vous pouvez le constater, la fonction `ifelse` nécessite trois arguments :

- Une condition, pouvant être `TRUE` ou `FALSE`,
- La valeur à renvoyer si la condition est `FALSE`

Avec la fonction `table`, nous pouvons rapidement visualisuer les effectifs des deux catégories ainsi créées :

```
table(accidents_df$moment_bin)
```

```
##  
## en heures de pointe hors heures de pointe  
##                 841                  1573
```

```
# vérifier si on a toujours seulement 40 NA  
table(is.na(accidents_df$moment_bin))
```

```
##  
## FALSE TRUE  
## 2414    40
```

Les heures de pointe représentent quatre heures de la journée, ce qui nous laisse neuf heures hors heures de pointe entre 07h00 et 20h00.

```
# Ratio de collisions routières en heures de pointe
(841 / 2414) / (4 / 13)
```

```
## [1] 1.132249
```

```
# Ratio de collisions routières hors heure de pointe
(1573 / 2414) / (9 / 13)
```

```
## [1] 0.9412225
```

En rapportant les collisions aux durées des deux périodes, on observe une nette surreprésentation des collisions impliquant un vélo pendant les heures de pointe d'environ 13% comparativement à la période hors des heures de pointe.

1.4.2.7.2 Le cas multiple avec la fonction `case_when`

Lorsque l'on souhaite créer plus que deux catégories, il est possible soit d'enchaîner plusieurs fonctions `ifelse` (ce qui produit un code plus long et moins lisible), soit d'utiliser la fonction `case_when` du package `dplyr` du `tidyverse`. Reprenons notre exemple et créons quatre catégories :

- En heures de pointe du matin
- En heures de pointe du soir
- Le reste de la journée (entre 07 :00 et 20 :00)
- La nuit (entre 21 :00 et 07 :00)

```
library(dplyr)

accidents_df$moment_multi <- case_when(
  accidents_df$diff_time_num >= 0 & accidents_df$diff_time_num <= 120 ~ "pointe matin",
  accidents_df$diff_time_num >= 570 & accidents_df$diff_time_num <= 690 ~ "pointe soir",
  accidents_df$diff_time_num > 690 & accidents_df$diff_time_num < 780 ~ "journée",
  accidents_df$diff_time_num > 120 & accidents_df$diff_time_num < 570 ~ "journée",
  accidents_df$diff_time_num < 0 | accidents_df$diff_time_num >= 780 ~ "nuit"
)
```

```
table(accidents_df$moment_multi)
```

```
##
##      journée        nuit pointe matin  pointe soir
##      1155            418       404        437
```

```
# vérifions encore les NA
table(is.na(accidents_df$moment_multi))
```

```
##
## FALSE   TRUE
## 2414     40
```

La syntaxe de cette fonction est un peu particulière. Elle accepte un nombre illimité d'arguments. Chaque argument est composé d'une condition et d'une valeur à renvoyer si la condition est vraie; ces deux éléments étant reliés par le symbole `~`. Notez que toutes les évaluations sont effectuées dans l'ordre des

arguments. En d'autres termes, la fonction va d'abord tester la première condition et assigner ces valeurs, puis recommencer pour les prochaines conditions. Ainsi, si une observation (ligne du tableau de données) obtient `TRUE` à plusieurs conditions, elle obtiendra la valeur de la dernière condition qu'elle a validée.

1.4.2.8 Sous-sélection d'un `DataFrame`

Dans cette section, nous verrons comment extraire des sous-parties d'un `DataFrame`. Il est possible de sous-sélectionner des lignes et des colonnes en se basant sur des conditions ou leurs index. Pour cela, nous allons utiliser un jeu de données fourni avec R : le jeu de données `iris` décrivant des fleurs du même nom.

```
data("iris")

# Nombre de lignes et de colonnes
dim(iris)

## [1] 150    5
```

1.4.2.8.1 Sous-sélection des lignes

Sous-sélectionner des lignes par index est relativement simple. Admettons que nous souhaitons sélectionner les lignes 1 à 5, 10 à 25, 37 et 58.

```
sub_iris <- iris[c(1:5, 10:25, 37, 58),]
nrow(sub_iris)
```

```
## [1] 23
```

Sous-sélectionner des lignes avec une condition peut être effectué soit avec une syntaxe similaire, soit en utilisant la fonction `subset`. Sélectionnons toutes les fleurs de l'espèce Virginica.

```
iris_virginica1 <- iris[iris$Species == "virginica",]
iris_virginica2 <- subset(iris, iris$Species == "virginica")

# Vérifions que les deux dataframes ont le même nombre de lignes
nrow(iris_virginica1) == nrow(iris_virginica2)
```

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

Vous pouvez utiliser dans les deux cas tous les opérateurs vus dans les sections 1.3.5.2 et 1.3.5.3. L'enjeu est d'arriver à un vecteur booléen final permettant d'identifier les observations à conserver.

1.4.2.8.2 Sous-sélectionner des colonnes

Nous avons déjà vu comment sélectionner des colonnes en utilisant leur nom ou leur index dans la section 1.4.2.2.1. Ajoutons ici un cas particulier où nous souhaiterions sélectionner des colonnes selon une condition. Par exemple, nous pourrions vouloir conserver que les colonnes comprenant le mot *Length*. Pour cela, nous utiliserons la fonction `grep`, permettant de déterminer si des caractères sont présents dans une chaîne de caractères.

```

nom_cols <- names(iris)
print(nom_cols)

## [1] "Sepal.Length" "Sepal.Width"   "Petal.Length" "Petal.Width"   "Species"

test_nom <- grep("Length", nom_cols, fixed = TRUE)
ok_nom <- nom_cols[test_nom]

iris_2 <- iris(ok_nom)
print(names(iris_2))

## [1] "Sepal.Length" "Petal.Length"

```

Il est possible d'obtenir ce résultat en une seule ligne de code, mais elle est un peu moins lisible.

```
iris2 <- iris[names(iris)[grep("Length", names(iris), fixed = TRUE)]]
```

1.4.2.8.3 Sélectionner des colonnes et des lignes

Nous avons vu qu'avec les crochets, nous pouvons extraire les colonnes et les lignes d'un *DataFrame*. Il est possible de combiner les deux opérations simultanément. Pour ce faire, il faut indiquer en premier les indices ou la condition permettant de sélectionner une ligne, puis les indices ou la condition pour sélectionner les colonnes : [index_lignes , index_colonnes]. Sélectionnons cinq premières lignes et les trois premières colonnes du jeu de données iris :

```

iris_5x3 <- iris[c(1,2,3,4,5),c(1,2,3)]
print(iris_5x3)

```

```

##   Sepal.Length Sepal.Width Petal.Length
## 1          5.1        3.5       1.4
## 2          4.9        3.0       1.4
## 3          4.7        3.2       1.3
## 4          4.6        3.1       1.5
## 5          5.0        3.6       1.4

```

Combinons nos deux exemples précédents pour sélectionner uniquement les lignes avec des fleurs de l'espèce virginica, et les colonnes avec le mot Length.

```

iris_virginica3 <- iris[iris$Species == "virginica",
                        names(iris)[grep("Length", names(iris), fixed = TRUE)]]
head(iris_virginica3, n=5)

```

```

##   Sepal.Length Petal.Length
## 101         6.3        6.0
## 102         5.8        5.1
## 103         7.1        5.9
## 104         6.3        5.6
## 105         6.5        5.8

```

1.4.2.9 Fusionner des *DataFrames*

Terminons cette section avec la fusion de *DataFrames* qu'il est possible de réaliser de deux façons, soit par ajout, soit par jointure.

1.4.2.9.1 Fusionner des *DataFrame* par ajout

Ajouter deux *DataFrames* peut se faire en fonction de leurs colonnes, ou en fonction de leurs lignes. Dans ces deux cas, on utilisera respectivement les fonctions `cbind` et `rbind`. La figure 1.16 résume graphiquement le fonctionnement des deux fonctions.

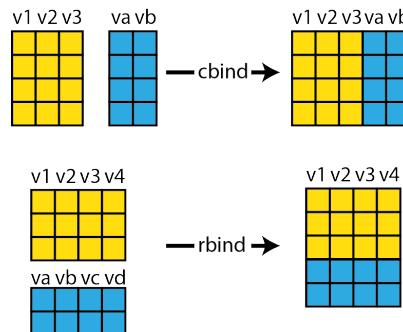


FIG. 1.16 : Fusion de DataFrames

Pour que `cbind` fonctionne, il faut que les deux *DataFrames* aient le même nombre de lignes. Pour `rbind`, les deux *DataFrames* doivent avoir le même nombre de colonnes. Prenons à nouveau comme exemple le jeu de données iris. Nous allons commencer par le séparer en trois sous-jeux de données comprenant chacun une espèce d'iris. Puis, nous fusionnerons deux d'entre eux avec la fonction `rbind`.

```
iris1 <- subset(iris, iris$Species == "virginica")
iris2 <- subset(iris, iris$Species == "versicolor")
iris3 <- subset(iris, iris$Species == "setosa")

iris_comb <- rbind(iris2,iris3)
```

Nous pourrions aussi extraire dans les deux *DataFrames* les colonnes comprenant le mot *Length* et le mot *Width*, puis les fusionner.

```
iris_l <- iris[names(iris)[grepl("Length",names(iris), fixed = TRUE)]]
iris_w <- iris[names(iris)[grepl("Width",names(iris), fixed = TRUE)]]

iris_comb <- cbind(iris_l,iris_w)
names(iris_comb)

## [1] "Sepal.Length" "Petal.Length" "Sepal.Width"   "Petal.Width"
```

1.4.2.9.2 Joindre des *DataFrame*

Une jointure est une opération un peu plus complexe qu'un simple ajout. L'idée est d'associer des informations de plusieurs *DataFrames* en utilisant une colonne (appelée une clef) présente dans les deux jeux de données. On distingue plusieurs types de jointure :

- Les jointures internes permettant de combiner les éléments communs entre deux *DataFrames* A et B
- La jointure complète permettant de combiner les éléments présents dans A ou B
- La jointure à gauche, permettant de ne conserver que les éléments présents dans A même s'ils ne trouvent pas leur correspondance dans B.

Ces trois jointures sont présentées à la figure 1.16; pour ces trois cas, la colonne commune se nomme *id*.

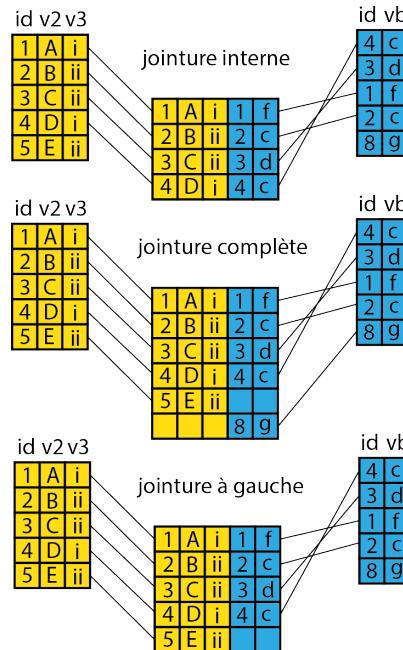


FIG. 1.17 : Jointure de DataFrames

Vous retiendrez que les deux dernières jointures peuvent produire des valeurs manquantes. Pour réaliser ces opérations, on utilise la fonction `merge`. Prenons un exemple simple à partir d'un petit jeu de données.

```
auteurs <- data.frame(
  name = c("Tukey", "Venables", "Tierney", "Ripley", "McNeil", "Apparicio"),
  nationality = c("US", "Australia", "US", "UK", "Australia", "Canada"),
  retired = c("yes", rep("no", 5)))
livres <- data.frame(
  aut = c("Tukey", "Venables", "Tierney", "Ripley", "Ripley", "McNeil", "Wickham"),
  title = c("Exploratory Data Analysis",
            "Modern Applied Statistics ...",
            "LISP-STAT",
            "Spatial Statistics", "Stochastic Simulation",
            "Interactive Data Analysis", "R for Data Science"))
```

Nous avons donc deux *DataFrames*, le premier décrivant des auteurs et le second des livres. Effectuons une première jointure interne afin de savoir pour chaque livre la nationalité de son auteur et si ce dernier est à la retraite.

```
df1 <- merge(livres, auteurs, #les deux DataFrames
              by.x = "aut", by.y = 'name', #les noms des colonnes de jointures
              all.x = FALSE, all.y = FALSE)
```

```
print(df1)

##          aut              title nationality retired
## 1  McNeil  Interactive Data Analysis  Australia     no
## 2  Ripley      Spatial Statistics        UK     no
## 3  Ripley  Stochastic Simulation        UK     no
## 4 Tierney        LISP-STAT           US     no
## 5  Tukey  Exploratory Data Analysis       US    yes
## 6 Venables Modern Applied Statistics ...  Australia     no
```

Cette jointure est interne car les deux paramètres `all.x` et `all.y` ont pour valeur `FALSE`. Ainsi, nous indiquons à la fonction que nous ne souhaitons ni garder tous les éléments du premier *DataFrame* ni tous les éléments du second, mais uniquement les éléments présents dans les deux. Vous noterez ainsi que le livre “R for Data Science” n’est pas présent dans le jeu de données final car son auteur “Wickham” ne fait pas partie du *DataFrame* auteurs. De même, l’auteur “Apparicio” n’apparaît pas dans la jointure, car aucun livre dans le *DataFrame* books n’a été écrit par cet auteur.

Pour conserver tous les livres, nous pouvons effectuer une jointure à gauche en renseignant `all.x = TRUE`. Nous allons ainsi forcer la fonction à garder tous les livres et mettre des valeurs vides aux informations manquantes des auteurs.

```
df2 <- merge(livres, auteurs, #les deux DataFrames
             by.x = "aut", by.y = 'name', #les noms des colonnes de jointures
             all.x = TRUE, all.y = FALSE)
```

```
print(df2)
```

```
##          aut              title nationality retired
## 1  McNeil  Interactive Data Analysis  Australia     no
## 2  Ripley      Spatial Statistics        UK     no
## 3  Ripley  Stochastic Simulation        UK     no
## 4 Tierney        LISP-STAT           US     no
## 5  Tukey  Exploratory Data Analysis       US    yes
## 6 Venables Modern Applied Statistics ...  Australia     no
## 7 Wickham        R for Data Science      <NA>     <NA>
```

Et pour garder tous les livres et tous les auteurs, nous pouvons faire une jointure complète en indiquant `all.x = TRUE` et `all.y = TRUE`.

```
df3 <- merge(livres, auteurs, #les deux DataFrames
             by.x = "aut", by.y = 'name', #les noms des colonnes de jointures
             all.x = TRUE, all.y = TRUE)
```

```
print(df3)
```

```
##          aut              title nationality retired
## 1 Apparicio                  <NA>      Canada     no
## 2  McNeil  Interactive Data Analysis  Australia     no
## 3  Ripley      Spatial Statistics        UK     no
## 4  Ripley  Stochastic Simulation        UK     no
```

```

## 5 Tierney           LISP-STAT      US    no
## 6 Tukey            Exploratory Data Analysis   US    yes
## 7 Venables Modern Applied Statistics ... Australia  no
## 8 Wickham          R for Data Science        <NA>  <NA>

```

1.5 Bien structurer un code R

Terminons ici avec quelques conseils sur la rédaction d'un code R. Bien rédiger son code est essentiel pour trois raisons :

1. pouvoir relire et réutiliser son code dans le futur.
2. permettre à d'autres personnes de réutiliser ou relire son code.
3. minimiser les risques d'erreurs.

Ne négligez pas l'importance d'un code bien rédigé et bien documenté, vous vous éviterez ainsi des migraines lorsque vous devrez exhumer du code écrit il y a plusieurs mois.

Voici quelques lignes directrices peu contraignantes, mais qui devraient vous être utiles :

1. **Privilégier la clarté à la concision** : il vaut mieux parfois scinder une ligne de code en plusieurs sous-étapes afin de faciliter la lecture de l'opération réalisée. Par exemple, si l'on reprend une ligne de code d'une section précédente où nous sélectionnions l'ensemble des colonnes du jeu de données `iris` comprenant le mot `Length` :

```
iris_l <- iris[names(iris)[grep("Length", names(iris), fixed = TRUE)]]
```

Il serait possible de simplifier la lecture de ce code en détaillant les différentes étapes comme suit :

```

noms_cols <- names(iris)
sel_noms <- noms_cols[grep("Length", noms_cols, fixed = TRUE)]
iris_l <- iris[sel_noms]

```

2. **Documenter et commenter son code le plus possible** : il est possible de rajouter du texte dans un code R qui ne sera pas exécuté, ce qu'on appelle des commentaires. Typiquement, une ligne commençant par un `#` ne sera pas interprétée par le logiciel. Utilisez des commentaires le plus souvent possible pour décrire les actions que vous souhaitez effectuer avec votre code. Il sera ainsi plus facile de le relire, de naviguer dans votre code, mais également de repérer d'éventuelles erreurs. Si l'on reprend l'exemple précédent :

```

# récupération du nom des colonnes dans le dataframe iris
noms_cols <- names(iris)

# sélection des colonnes avec les caractères "Length"
sel_noms <- noms_cols[grep("Length", noms_cols, fixed = TRUE)]

# extraction des colonnes sélectionnées dans un nouveau dataframe
iris_l <- iris[sel_noms]

```

3. **Éviter le code à rallonge...** : typiquement, essayez de vous limiter à des lignes de code d'une longueur maximale de 80 caractères. Au-delà de ce seuil, il est judicieux de découper votre code en plusieurs lignes.

4. **Adopter une convention d'écriture** : une convention d'écriture est un ensemble de règles strictes définissant comment un code doit être rédigé. À titre d'exemple, il est parfois recommandé d'utiliser le lowerCamelCase, le UpperCamelCase, ou encore de séparer les mots par des tirets bas upper_camel_case. Un mélange de ces différentes conventions peut être utilisé pour distinguer les variables, les fonctions et les classes. Il peut être difficile de réellement arrêter une telle convention, car les différents packages dans R utilisent des conventions différentes. Dans vos propres codes, il est surtout important d'avoir une certaine cohérence et ne pas changer de convention au fil de votre code.
5. **Indenter son code** : l'indentation du code permet de le rendre beaucoup plus lisible. Indenter son code signifie insérer au début de chaque ligne de code un certain nombre d'espaces permettant d'indiquer à quel niveau de profondeur on se situe. Typiquement, lorsque des accolades ou des parenthèses sont ouvertes dans une fonction, une boucle ou une condition, on rajoute deux ou quatre espaces en début de ligne. Prenons un exemple très concret, admettons que nous écrivons une fonction affichant un résumé statistique à chaque colonne d'un jeu de données si cette colonne est de type numérique. L'indentation dans cette fonction va jouer un rôle crucial dans sa lisibilité. Sans indentation et sans respecter la règle des 80 caractères, on obtient ceci :

```
summary_all_num_cols <- function(dataset){for(col in names(dataset)){if(class(dataset[[col]] == "numeric")){print(su
```

Avec de l'indentation et des commentaires, la syntaxe est beaucoup plus lisible puisqu'elle permet de repérer facilement trois niveaux / paliers dans le code :

```
#définition d'une fonction
summary_all_num_cols <- function(dataset){
  #Itération sur chaque colonne de la fonction
  for(col in names(dataset)){
    # A chaque itération, testons si la colonne est de type numérique
    if(class(dataset[[col]] == "numeric")){
      # Si oui, on affiche un résumé statistique pour cette colonne
      print(summary(dataset[[col]]))
    } # ici on sort de la condition (niveau 3)
  } # ici on sort de la boucle (niveau 2)
}# ici on sort de la fonction (niveau 1)
```

6. **Adopter une structure globale pour vos scripts** : un code R peut être comparé à une recette de cuisine. Si tous les éléments sont dans le désordre et sans structure globale, la recette risque d'être très difficile à suivre. Cette structure risque de changer quelque peu en fonction de la recette ou de l'auteur·e, mais les principaux éléments restent les mêmes. Dans un code R, on peut distinguer plusieurs éléments récurrents que nous vous recommandons d'organiser de la façon suivante :
7. Charger les différents packages **utilisés** par le script. Cela permet dès le début du code de savoir quelles sont les fonctions et méthodes qui seront employées dans le script. On limite aussi les risques d'oublier des packages qui seraient chargés plus loin dans le code.
8. Définir les fonctions dont vous aurez besoin en plus de celles présentes dans les packages. Idem, placer ses fonctions en début de code évite d'oublier de les charger ou de les chercher quand on en a besoin.
9. Définir le répertoire de travail avec la fonction `setwd` et charger les données nécessaires.
10. Effectuer au besoin les opérations de manipulations sur les données.
11. Effectuer les analyses nécessaires en scindant si possibles les différentes étapes.

Notez également que l'étape de définition des fonctions complémentaires peut être effectuée dans une feuille de code séparée, et l'ensemble de ces fonctions chargées à l'aide de la fonction `source`. De même,

si la manipulation des données est conséquente, il est recommandé de l'effectuer avec un code à part, d'enregistrer les données structurées, puis de le charger directement au début de votre code dédié à l'analyse.

7. **Exploiter les commentaires délimitant les sections dans Rstudio** : Dans RStudio, il est possible d'écrire des commentaires d'une certaine façon pour que l'IDE les détecte comme des délimiteurs de sections. L'intérêt principal est que l'on peut ensuite facilement naviguer entre ces sections en utilisant RStudio comme montré à la figure @ref(fig :sections_rstudio), mais aussi masquer des sections afin de faciliter la lecture du reste du code. Pour délimiter une section, il suffit d'ajouter une ligne de commentaire comprenant quatre fois les caractères -, = ou # à la suite :

```
# Voici ma section 1 -----
# Voici ma section 2 =====
# Voici ma section 3 #####
```

Autre exemple pour mieux marquer la rupture dans un code :

```
##### Titre de ma section 4 #####
#####
```

8. **Adopter une structure globale pour vos projets** : au-delà du script, il est nécessaire de bien structurer vos projets. Le plus important étant d'utiliser une structure commune à chaque projet pour vous faciliter le travail. Nous proposons ci-dessous un exemple de structure assez général pouvant être utilisé dans la plupart des cas. Elle sépare notamment les données originales des données structurées, ainsi que les fonctions complémentaires et la structuration des données du principal bloc d'analyse.

Ne négligez jamais l'importance d'un code bien écrit et documenté!

1.6 Conclusion et ressources pertinentes

Voilà qui conclut ce chapitre sur les bases du langage R. Vous avez maintenant les connaissances nécessaires pour commencer à travailler. N'hésitez pas à revenir sur les différentes sous-sections au besoin! Pour aller plus loin dans l'apprentissage du langage, vous pouvez également vous plonger dans le chapitre R AVANCÉ. Cependant, nous vous recommandons de faire vos premiers pas avec cette base avant de vous lancer dans cette partie davantage orientée programmation. Quelques ressources pertinentes qui pourraient vous être utiles sont aussi reportées au tableau ci-dessous.

TAB. 1.7 : Ressources pertinente pour en apprendre plus sur R

Ressource	Description	Url
Rbloggers	Un recueil de nombreux blogues sur R : parfait pour être tenu au courant des nouveautés et faire des découvertes.	https://www.r-bloggers.com
CRAN packages by date	Les derniers packages publiés sur CRAN : cela permet de garder un oeil sur les nouvelles fonctionnalités de ses packages préférés.	https://cran.r-project.org/web/packages
Introduction à R et au TidyVerse	Une excellente ressource en français pour en apprendre plus sur le tidyverse.	https://juba.github.io/tidyverse
Numyard	Une chaîne YouTube pour revoir les bases de R en vidéo.	https://www.youtube.com/user/TheLearnR
Cheatsheets	Des feuilles de triche résumant les fonctionnalités de nombreux packages.	https://rstudio.com/resources/cheatsheets

Deuxième partie

Analyses univariées

Chapitre 2

Statistiques descriptives univariées

Dans ce chapitre, nous décrirons la notion de variable, permettant l'opérationnalisation d'un concept. Comprendre les différents types de variables est essentiel en statistiques. En effet, en fonction du type de variable à l'étude, les tests d'hypothèse et les méthodes de statistique inférentielle que l'on pourra appliquer seront différents. Nous distinguerons ainsi cinq types de variables : nominale, ordinaire, discrète, continue et semi-quantitative. Aussi, nous abordons un concept central de la statistique : les distributions. Nous présenterons ensuite les différentes statistiques descriptives univariées qui peuvent s'appliquer à ces types de variables.



Dans ce chapitre, nous utiliserons principalement les packages suivants (À MODIFIER PLUS TARD) :

- Pour créer des graphiques :
 - * **ggplot2**, le seul, l'unique
 - * **ggbpubr** pour combiner des graphiques et réaliser des diagrammes
- Pour créer des distributions :
 - * **fitdistrplus** pour générer différentes distributions
 - * **actuar** pour la fonction de densité de Pareto
 - * **gamlss.dist** pour des distributions de Poisson
- Pour les statistiques descriptives :
 - * **stats** pour les statistiques descriptives
 - * **nortest** pour le test de Kolmogorov-Smirnov
 - * **DescTools** pour les tests de Lilliefors, Shapiro-Wilk, Anderson-Darling et Jarque-Bera
- Autres packages :
 - * **Hmisc** et **Weighted.Desc.Stat** pour les statistiques descriptives pondérées
 - * **foreign** pour importer des fichiers externes

2.1 Notion de variable

2.1.1 La variable : l'opérationnalisation d'un concept

Une variable permet d'opérationnaliser un concept, soit une « idée générale et abstraite que se fait l'esprit humain d'un objet de pensée concret ou abstrait, et qui lui permet de rattacher à ce même objet les diverses perceptions qu'il en a, et d'en organiser les connaissances » (Larousse¹). Pour valider un modèle théorique, il convient alors d'opérationnaliser ses différentes concepts et d'établir les relations qu'ils partagent. L'opérationnalisation d'un concept nécessite soit de mesurer (dans un intervalle de valeurs,

¹<https://www.larousse.fr/dictionnaires/francais/concept/17875?q=concept#17749>

c'est-à-dire de manière quantitative), soit de qualifier (avec plusieurs catégories, c'est-à-dire de manière qualitative) un phénomène.

Selon Statistique Canada², « une variable est une caractéristique d'une unité statistique que l'on observe et pour laquelle une valeur numérique ou une catégorie d'une classification peut être attribuée ». Il convient alors de bien saisir à quelle unité statistique (ou unité d'observation) s'applique les valeurs d'une variable : des personnes, des ménages, des municipalités, etc.

Prenons deux exemples concrets tirées du Recensement de 2016 de Statistique Canada :

- Le concept **famille de recensement** est défini comme étant « un couple marié et les enfants, le cas échéant, du couple et/ou de l'un ou l'autre des conjoints ; un couple en union libre et les enfants, le cas échéant, du couple et/ou de l'un ou l'autre des partenaires ; ou un parent seul, peu importe son état matrimonial, habitant avec au moins un enfant dans le même logement et cet ou ces enfants. Tous les membres d'une famille de recensement particulière habitent le même logement. Un couple peut être de sexe opposé ou de même sexe. Les enfants peuvent être des enfants naturels, par le mariage, par l'union libre ou par adoption, peu importe leur âge ou leur état matrimonial, du moment qu'ils habitent dans le logement sans leur propre conjoint marié, partenaire en union libre ou enfant. Les petits-enfants habitant avec leurs grands-parents, alors qu'aucun des parents n'est présent, constituent également une famille de recensement » (Statistique Canada³). À partir de cette définition, les familles de recensement peuvent être qualifiées selon plusieurs modalités : couples mariés sans enfant, couples mariés avec enfants, couples en union libre sans enfant, couples en union libre avec enfant, famille monoparentale (avec un parent de sexe féminin), famille monoparentale (avec un parent de sexe masculin).
- Le concept de **revenu d'emploi** est défini comme étant « tous les revenus reçus sous forme de traitements, salaires et commissions d'un travail rémunéré ou le revenu net d'un travail autonome dans une entreprise agricole ou non agricole non constituée en société et/ou dans l'exercice d'une profession au cours de la période de référence. Pour le Recensement de 2016, la période de référence est l'année civile 2015 pour toutes les variables de revenu » (Statistique Canada⁴). Il est donc mesurée en dollars pour chaque individu de 15 ans et plus. Pour l'ensemble de la population de 15 ans et plus, il peut ensuite être classé en déciles de revenu d'emploi, soit en dix groupes (Statistique Canada⁵).

Maîtriser la définition des variables que vous utilisez : un enjeu crucial !

Nous avons vu qu'une variable est l'opérationnalisation d'un concept. Par conséquent, ne pas maîtriser la définition d'une variable revient à ne pas bien saisir le concept sous-jacent qu'elle tente de mesurer. Si vous exploitez des données secondaires – par exemple, issues d'un recensement de population ou d'une enquête longitudinale ou transversale –, il faut impérativement lire les définitions des variables que vous souhaiteriez utiliser. Ne pas le faire risque d'aboutir à :

- Une mauvaise opérationnalisation de votre modèle théorique, même si votre analyse est bien menée statistiquement parlant. Autrement dit, vous risquez de ne pas sélectionner les bonnes variables. Prenons un exemple concret. Vous avez construit un modèle théorique dans lequel vous souhaitez inclure un concept sur la langue des personnes. Dans le recensement canadien de 2016, plusieurs variables relatives à la langue sont disponibles : connaissance des langues officielles⁶, langue parlée à la maison⁷, langue maternelle⁸, première langue officielle parlée⁹, connaissance des langues non officielles¹⁰ et langue de travail¹¹ (Statistique Canada, 2019¹²). La sélection de l'une de ces variables doit être faite de manière rigoureuse, c'est-à-dire en lien avec votre cadre théorique et suite à une bonne compréhension

²<https://www.statcan.gc.ca/fra/concepts/variable>

³<https://www12.statcan.gc.ca/census-recensement/2016/ref/dict/fam004-fra.cfm>

⁴<https://www12.statcan.gc.ca/census-recensement/2016/ref/dict/pop027-fra.cfm>

⁵<https://www12.statcan.gc.ca/census-recensement/2016/ref/dict/pop204-fra.cfm%22%7D>

des définitions des variables. Dans une étude sur le marché du travail, on sélectionnerait probablement la variable *sur la connaissance des langues officielles du Canada*, afin d'évaluer son effet sur l'employabilité, toutes choses étant égales par ailleurs. Dans une autre étude portant sur la réussite ou la performance scolaire, il est probable qu'on utilise plutôt la *langue maternelle*.

- Une mauvaise interprétation et discussion de vos résultats en lien avec votre cadre théorique.
- Une mauvaise identification des pistes de recherche.

Finalement, la définition d'une variable peut évoluer à travers plusieurs recensements de population : la société évolue, les variables aussi ! Par conséquent, si vous comptez utiliser plusieurs années de recensement dans une même étude, assurez-vous que les définitions des variables soient similaires d'un jeu de données à l'autre et qu'elles mesurent ainsi la même chose.

Comprendre les variables utilisées dans un article scientifique : un exercice indispensable dans l'élaboration d'une revue de littérature

Une lecture rigoureuse d'un article scientifique suppose, entre autres, de bien comprendre les concepts et variables mobilisés. Il convient alors de lire attentivement la section méthodologique (pas uniquement la section des résultats ou pire le résumé), sans quoi vous risquez d'aboutir à une revue de littérature approximative. Ayez aussi un **regard critique** sur les variables visant à opérationnaliser les concepts clés de l'étude. Certains concepts sont très difficiles à traduire en variables ; leurs opérationnalisations (mesures) peuvent ainsi faire l'objet de vifs débats parmi les chercheurs. Très succinctement, c'est notamment le cas du concept de capital social. D'une part, les définitions et ancrages sont biens différents selon Bourdieu (sociologue, ancrage au niveau des individus) et Putman (politologue, ancrage au niveau des collectivités) ; d'autre part, aucun consensus ne semble clairement se dégager quant à la définition de variables permettant de le mesurer efficacement (de manière quantitative).

Variable de substitution (*proxy variable* en anglais)

On fait la moins pire des recherches ! En effet, les données disponibles sont parfois imparfaites pour répondre avec précision à une question de recherche ; on peut toujours les exploiter, tout en signalant honnêtement leurs faiblesses et limites, et ce, tant pour les données que les variables utilisées.

- Des bases de données peuvent être en effet imparfaites. Par exemple, en criminologie, des chercheur·e·s exploitant des données policières signalent habituellement la limite du **chiffre noir** : les données policières comprennent uniquement les crimes et délits découverts par la police et occultent ainsi les crimes non-découverts ; ils ne peuvent ainsi refléter la criminalité réelle sur un territoire donné.
- Des variables peuvent aussi être imparfaites. Dans un jeu de données, il est fréquent qu'une variable opérationnalisant un concept précis ne soit pas disponible ou qu'elle n'ait tout simplement pas été mesurée. On cherchera alors une variable de substitution (*proxy*) pour la remplacer. Prenons un exemple concret portant sur l'exposition des cyclistes à la pollution atmosphérique ou au bruit environnemental. L'un des principaux facteurs d'exposition à ces pollutions est le trafic routier : plus ce dernier est élevé, plus les cyclistes risquent de rouler dans un environnement bruyant et pollué. Toutefois, il est rare de disposer de mesures du trafic en temps réel qui nécessitent des comptages de véhicules pendant le trajet des cyclistes (par exemple, à partir de vidéos captées par une caméra fixée sur le guidon). Pour pallier à l'absence de mesures directes, plusieurs auteurs utilisent des variables de substitution de la densité du trafic, comme la typologie des types d'axes (primaire, secondaire, tertiaire, rue locale, etc.), supposant ainsi qu'un axe primaire supporte un volume de véhicules supérieur à un axe secondaire.

2.1.2 Les types de variables

On distingue habituellement les variables qualitatives (nominale ou ordinale) des variables quantitatives (discrète ou continue). Tel qu'illustré à la figure 2.1, l'opérationnalisation du concept en variable est réalisée par différents mécanismes visant à qualifier, classer, compter ou mesurer afin de caractériser les unités statistiques (observations) d'une population ou d'un échantillon.

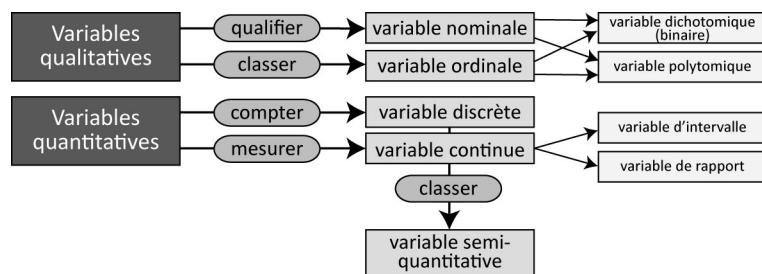


FIG. 2.1 : Les types de variables

2.1.2.1 Les variables qualitatives

Une **variable nominale** permet de **qualifier** des observations (individus) à partir de plusieurs catégories dénommées modalités. Par exemple, la variable *couleur des yeux* pourrait comprendre les modalités *bleu*, *marron*, *vert*, *noir* tandis que les *types de familles* compendrait les modalités *couple marié*, *couple en union libre* et *famille monoparentale*.

Une **variable ordinaire** permet de **classer** des observations à partir de plusieurs modalités hiérarchisées. L'exemple le plus connu est certainement l'échelle de Likert, très utilisée dans les sondages évaluant le degré d'accord d'une personne à une affirmation avec les modalités suivantes : *tout à fait d'accord*, *d'accord*, *ni en désaccord ni d'accord*, *pas d'accord* et *pas du tout d'accord*. Une multitude de variantes sont toutefois possibles pour classer la fréquence d'un phénomène (*Très souvent*, *souvent*, *parfois*, *rarement*, *jamais*), l'importance accordée à un phénomène (*Pas du tout important*, *peu important*, *plus ou moins important*, *important*, *très important*) ou la proximité perçue d'un lieu (*très éloigné*, *loin*, *plus ou moins proche*, *proche*, *très proche*).

En fonction du nombre de modalités qu'elle comprend, une variable qualitative (nominale ou ordinaire) est soit **dichotomique (binnaire)** (deux modalités), soit **polytomique** (plus de deux modalités). Par exemple, dans le recensement canadien, le *sexe* est une variable binaire (avec les modalités *sexe masculin*, *sexe féminin*), tandis que le *genre* est une variable polytomique (avec les modalités *genre masculin*, *genre féminin* et *diverses identités de genre*).



Les variables nominales et ordinaires sont habituellement encodées avec des valeurs numériques entières (par exemple, 1 pour *couple marié*, 2 pour *couple en union libre* et 3 pour *famille monoparentale*). Toutefois, aucune opération arithmétique (moyenne ou écart-type par exemple) n'est possible sur ces valeurs. Dans R, on utilisera un facteur pour attribuer un intitulé à chacune des valeurs numériques de la variable qualitative :

```
df$Famille <- factor(df$Famille, c(1,2,3), labels = c("couple marié", "couple en union libre", "famille monoparentale"))
```

On calculera toutefois les fréquences des différentes modalités pour une variable nominale ou ordinaire. Il est aussi possible de calculer la médiane sur une variable ordinaire.

2.1.2.2 Les variables quantitatives

Une **variable discrète** permet de **compter** un phénomène dans un ensemble fini de valeurs, comme le nombre d'accidents impliquant un-e cycliste à une intersection sur une période de cinq ans ou encore le nombre de vélos en libre service disponibles à une station. Il existe ainsi une variable binaire sous-jacente : la présence ou non d'un accident à l'intersection ou d'un vélo ou non à la station pour laquelle on opère un comptage. Habituellement, une variable discrète ne peut prendre que des valeurs entières (sans décimales), comme le nombre de personnes fréquentant un parc.

Une **variable continue** permet de **mesurer** un phénomène avec un nombre infini de valeurs réelles (avec

décimales) dans un intervalle donné. Par exemple, une variable relative à la distance de dépassement d'un·e cycliste par un véhicule motorisé pourrait varier de 0 à 5 mètres ($X \in [0, 5]$); toutefois cette distance peut être de 0,759421 ou de 4,785612 mètres. Le nombre de décimales de la valeur réelle dépendra de la précision et de la fiabilité de la mesure. Pour un capteur de distance de dépassement, le nombre de décimales dépendra de la précision du lidar ou du sonar de l'appareil; aussi, l'utilisation de trois décimales – soit une précision au millimètre – est largement suffisant pour mesurer la distance de dépassement. Une variable continue est soit une variable d'intervalle, soit une variable de rapport. Les **variables d'intervalle** ont une échelle relative, c'est-à-dire que les intervalles entre les valeurs de la variables ne sont pas constants; elles n'ont pas de vrai zéro. Ces valeurs peuvent être manipulées uniquement par addition et soustraction et non par multiplication et division. La variable d'intervalle la plus connue est certainement celle de la température. S'il fait 10 degrés Celsius à Montréal et 30°C à Mumbai (soit 50 et 86 degrés en Fahrenheit), on peut affirmer qu'il y a 20°C ou 36°F d'écart entre les deux villes, mais on ne peut pas affirmer qu'il fait trois fois plus chaud à Mumbai. Presque toutes les mesures statistiques sur une variable d'intervalle peuvent être calculées, exceptés le coefficient de variation et la moyenne géométrique puisqu'il n'y a pas de vrai zéro et d'intervalles constants entre les valeurs. À l'inverse, les **variables de rapport** ont une échelle absolue, c'est-à-dire que les intervalles entre les valeurs sont constants et elles ont un vrai zéro. Elles peuvent ainsi être manipulées par addition, soustraction, multiplication et division. Par exemple, le prix d'un produit exprimé dans une unité monétaire ou la distance exprimée dans le système métrique sont des variables de rapport. Un vélo dont le prix affiché est de 1000\$ est bien deux fois plus cher qu'un autre à 500\$, une piste cyclable hors rue à 25 mètres du tronçon routier le plus proche est bien quatre fois plus proche qu'une autre à 100 mètres.

Une variable semi-quantitative, appelée aussi variable quantitative ordonnée, est une variable discrète ou continue dont les valeurs ont été regroupées en classes hiérarchisées. Par exemple, l'âge est une variable continue pouvant être transformée avec les groupes d'âge ordonnés suivants : *moins 25 ans, 25 à 44 ans, 45 à 64 ans et 65 ans et plus*.

2.2 Les types de données

Différents types de données sont utilisés en sciences sociales. L'objectif ici n'est pas de les décrire en détail, mais plutôt de donner quelques courtes définitions. En fonction de votre question de recherche et des bases des données disponibles ou non, il s'agira de sélectionner le ou les types de données les plus appropriés à votre sujet.

2.2.1 Données secondaires *versus* données primaires

Les **données secondaires** sont des données qui existent déjà au début de votre projet de recherche : pas besoin de les collecter, il suffit de les exploiter! Une multitude de données de recensements ou d'enquêtes de Statistique Canada sont disponibles et largement exploitées en sciences sociales (par exemple, l'enquête nationale auprès des ménages – ENM, l'enquête sur la dynamique du marché du travail et du revenu – EDTR, l'enquête longitudinale auprès des immigrants – ELIC, etc.).



Au Canada, les chercheurs (étudiants et professeurs) ont accès aux microdonnées des enquêtes de Statistique Canada dans les Centres de données de recherche (CDR). Vous pouvez consulter le moteur de recherche du (RCCDR¹³) afin d'explorer les différentes enquêtes disponibles.

Au Québec, l'accès à ces enquêtes est possible dans les différentes antennes du Centre interuniversitaire québécois de statistiques sociales de Statistique Canada (CIQSS¹⁴).

Par opposition, les **données primaires** n'existent pas quand vous démarrez votre projet : vous devez les

collecter spécifiquement pour votre étude ! Par exemple, un·e chercheur·e souhaitant analyser l'exposition des cyclistes au bruit et à la pollution dans une ville donnée devra réaliser une collecte de données avec idéalement plusieurs participants (équipés de différents capteurs), et ce, sur plusieurs jours. Une collecte de données primaires peut aussi être réalisée avec une enquête par sondage. Brièvement, réaliser une collecte de données primaires nécessite différentes phases complexes comme la définition de la méthode de collecte, de la population à l'étude, l'estimation de la taille de l'échantillon, la validation des outils de collecte avec une phase de test, la réalisation de la collecte, la structuration, la gestion et l'exploitation de données collectées. Finalement, dans le milieu académique, une collecte de données primaires auprès d'individus doit être approuvée par le comité d'éthique de la recherche de l'université à laquelle est affilié·e le ou la responsable du projet de recherche (qu'il soit professeur·e, chercheur·e ou étudiant·e).

2.2.2 Données transversales *versus* données longitudinales

Les **données transversales** sont des mesures pour une période relativement courte. L'exemple classique est un jeu de données constitué des variables extraites d'un recensement de population pour une année donnée (comme celui 2016 de Statistique Canada).

Les **données longitudinales**, appelées aussi données par panel, sont des mesures répétées pour plusieurs observations au cours du temps (N observations pour T dates). Par exemple, des observations pourraient être des pays, les dates pourraient être différentes années (de 1990 à 2019) pour lesquelles différentes variables seraient disponibles (population totale, taux d'urbanisation, produit intérieur brut par habitant, émissions de gaz à effet de serre par habitant, etc.).

2.2.3 Données spatiales *versus* données aspatiales

Les observations des **données spatiales** sont des unités spatiales géoréférencées (points, lignes, polygones ou encore pixels d'une image). Elles peuvent être par exemple :

- des points (x,y) ou (*lat-long*) représentant des entreprises avec plusieurs variables (adresse, date de création, nombre d'employés, secteurs d'activité, etc.);
- les lignes représentant des tronçons de rues pour lesquels plusieurs variables sont disponibles (types d'axe, longueur en mètres, nombre de voies, débit journalier moyen annuel, etc.);
- des polygones délimitant des régions ou des arrondissements pour lesquels une multitude de variables sociodémographiques et socioéconomiques sont disponibles.

À l'inverse, aucune information spatiale n'est disponible pour des **données aspatiales**.

2.2.4 Données individuelles *versus* données agrégées

Comme son nom l'indique, pour des **données individuelles**, chaque observation correspond à un individu. Les microdonnées de recensement ou d'enquêtes, par exemple, sont des données individuelles pour lesquelles toute une série de variables est disponible. Une étude analysant les caractéristiques de chaque arbre d'un quartier nécessite aussi des données individuelles : l'information doit être disponible pour chaque arbre. Pour les microdonnées des recensements canadiens, «chaque enregistrement au niveau de la personne comprend des identifiants (comme les identifiants du ménage et de la famille), des variables géographiques et des variables directes et dérivées tirées du questionnaire» (Statistique Canada¹⁵). Comme signalé plus haut, ces microdonnées de recensement ou d'enquêtes sont uniquement accessibles dans les Centres de données de recherche (CDR).

¹⁵ <https://www150.statcan.gc.ca/n1/pub/12-002-x/2012001/article/11642-fra.htm>

Les données individuelles peuvent être **agrégées** à un niveau supérieur. Prenons le cas de microdonnées d'un recensement. Les informations disponibles pour chaque individu sont agrégées par territoire géographique (province, région économique, division de recensement, subdivision de recensement, région et agglomération de recensement, secteurs de recensement, aires de diffusion, etc.) en fonction du lieu de résidence des individus. Des sommaires statistiques – basés sur la moyenne, la médiane, la somme ou la proportion de chacune des variables mesurées au niveau individuel (âge, sexe, situation familiale, revenu, etc.) – sont alors construits pour ces différents découpages géographiques (Statistique Canada¹⁶).

L'agrégation n'est pas nécessairement géographique. En éducation, il est fréquent de travailler avec des données concernant les élèves, mais agrégées au niveau des écoles. La figure 2.2 donne un exemple simple d'agrégation de données individuelles.

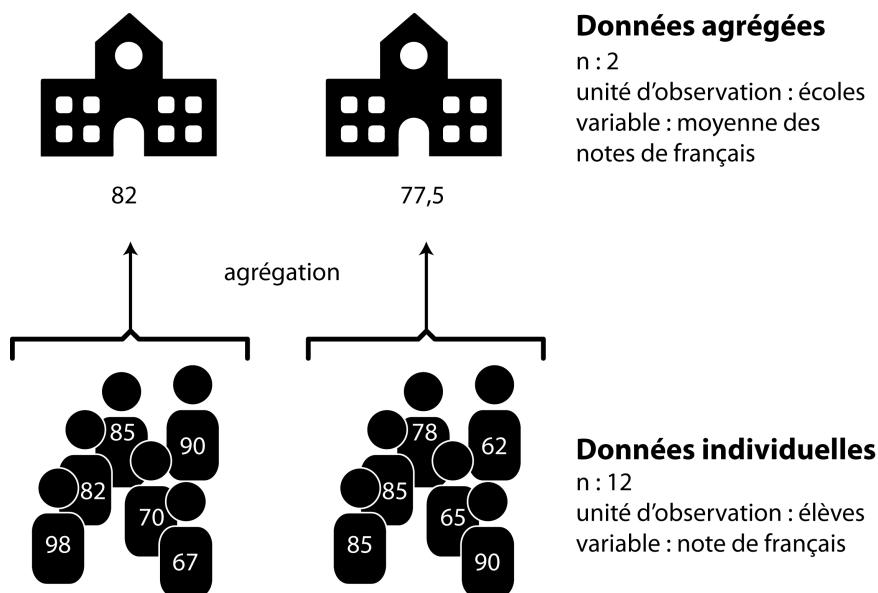


FIG. 2.2 : Exemple d'agrégation de données individuelles

Pour le cas de l'agrégation géographique, il convient alors de bien comprendre la hiérarchie des régions géographiques délimitées par l'organisme ou l'agence ayant la responsabilité de produire, gérer et diffuser les données des recensements et des enquêtes, puis de sélectionner le découpage géographique qui répond le mieux à votre question de recherche.



Pour le recensement de 2016 de Statistique Canada vous pourrez consulter :

- la hiérarchie des régions géographiques normalisées pour la diffusion¹⁷
- le glossaire illustré¹⁸ des régions géographiques
- les différents profils du recensement de 2016¹⁹ à télécharger pour les différentes régions géographiques.



Bien entendu, les différents types de données abordés ci-dessus ne sont pas exclusifs. Par exemple, des données pour des régions administratives extraites de plusieurs recensements sont en fait des données secondaires, spatiales, agrégées et longitudinales.

Une collecte de données sur la pollution atmosphérique et sonore réalisée à vélo (avec différents capteurs et un GPS) sont des données spatiales primaires.

¹⁶ <https://www.statcan.gc.ca/fra/idd/trousse/section5#a4>

2.3 Statistique descriptive et statistique inférentielle

2.3.1 Population, échantillon et inférence

Les notions de **population** et d'**échantillon** sont essentielles en statistique puisqu'elles sont le socle de l'inférence statistique. Un échantillon est un **sous-ensemble représentatif** d'une population donnée. Prenons un exemple concret. Une chercheure veut comprendre la mobilité des étudiants d'une université. Bien entendu, elle ne pourra interroger l'ensemble des étudiants de son université. Elle devra alors s'assurer d'obtenir un échantillon de taille suffisante et représentatif de la population étudiante. Une fois les données collectées (avec un sondage par exemple), elle pourra utiliser des techniques inférentielles pour analyser la mobilité des étudiants interrogés. Si son échantillon est représentatif, les résultats obtenus pourront être inférés – c'est-à-dire généralisés, extrapolés – à l'ensemble de la population.



Les méthodes d'échantillonnage

Nous n'abordons pas ici les méthodes d'échantillonnage. Sachez toutefois qu'il existe plusieurs méthodes probabilistes pour constituer un échantillon, notamment de manière aléatoire, systématique, stratifiée, par grappes (voir par exemple cette publique de Statistique Canada²⁰).

Autre exemple, une autre chercheure souhaite comprendre les facteurs influençant le sentiment de sécurité des cyclistes dans un quartier. De nouveau, elle ne pourra pas enquêter tous les cyclistes du quartier et devra constituer un échantillon représentatif. Par la suite, la mise en œuvre de techniques inférentielles lui permettra d'identifier les caractéristiques individuelles (âge, sexe, habiletés à vélo, etc.) et de l'environnement urbain (types de voies empruntés, niveaux de trafic, de pollution, de bruit, etc.) ayant des effets significatifs sur le sentiment de sécurité. Si l'échantillon est représentatif, les résultats pourront être généralisés à l'ensemble des cyclistes du quartier.

2.3.2 Deux grandes familles de méthodes statistiques

On distingue deux grandes familles de méthodes statistiques :

- « **La statistique descriptive et exploratoire** : elle permet, par des résumés et des graphiques plus ou moins élaborés, de décrire des ensembles de données statistiques, d'établir des relations entre les variables sans faire jouer de rôle privilégié à une variable particulière. Les conclusions ne portent dans cette phase de travail que sur les données étudiées, sans être inférées à une population plus large. L'analyse exploratoire s'appuie essentiellement sur des notions élémentaires telles que des indicateurs de moyenne et de dispersion, sur des représentations graphiques. [...]]
- **La statistique inférentielle et confirmatoire** : elle permet de valider ou d'infirmer, à partir de tests statistiques ou de modèles probabilistes, des hypothèses formulées a priori (ou après une phase exploratoire), et d'extrapoler, c'est-à-dire d'étendre certaines propriétés d'un échantillon à une population plus large. Les conclusions obtenues à partir des données vont au-delà de ces données. La statistique confirmatoire fait surtout appel aux méthodes dites explicatives et prévisionnelles, destinées comme leurs noms l'indiquent, à expliquer puis à prévoir, suivant des règles de décision, une variable privilégiée à l'aide d'une ou plusieurs variables explicatives (régressions multiples et logistiques, analyse de variance, analyse discriminante, segmentation, etc.) » (? , p. 209).

2.4 La notion de distribution



Dans cette section, nous abordons un concept central de la statistique : les distributions. Prenez le temps de lire cette section à tête reposée et assurez-vous de bien comprendre chaque idée avant de passer à la suivante. N'hésitez pas à y revenir plusieurs fois si nécessaire, car la compréhension de ces concepts est essentielle pour utiliser adéquatement les méthodes que nous abordons dans ce livre.

2.4.1 Définitions générales

En statistique, on s'intéresse aux résultats d'expériences. Lancer un dé, mesurer la pollution atmosphérique, compter le nombre de collisions à une intersection, demander à une personne d'évaluer son sentiment de sécurité sur une échelle de 1 à 10 sont autant d'expériences pouvant produire des résultats.

Une distribution est une fonction permettant d'associer pour chaque résultat possible d'une expérience la probabilité d'obtenir ce résultat. En d'autres termes, il s'agit d'une fonction indiquant par exemple que pour l'expérience : «mesurer la concentration d'ozone à Montréal à 13h en été», la probabilité de mesurer une valeur inférieure à $15 \mu\text{g}/\text{m}^3$ est de seulement 2%.

Les distributions sont toujours définies dans un intervalle en dehors duquel elles sont indéfinies ; les valeurs dans cet intervalle sont appelées **l'espace d'échantillonnage**. Il s'agit donc des valeurs possibles que peut produire l'expérience. La somme des probabilités de l'ensemble des valeurs de l'espace d'échantillonnage est 1 (100%). Intuitivement, cela signifie que si l'on réalise l'expérience, on est obligé d'obtenir un résultat, et que cette probabilité totale est répartie entre tous les résultats possibles de l'expérience. En langage mathématique, on dit que l'intégrale des fonctions de distribution est 1 dans leur intervalle de définition.

Prenons un exemple concret avec l'expérience suivante : tirer à pile ou face avec une pièce de monnaie non truquée. Si l'on souhaite décrire la probabilité d'obtenir pile ou face, on peut utiliser une distribution qui aura comme espace d'échantillonnage [pile ; face] et ces deux valeurs auront chacune comme probabilité 0,5. Il est facile d'étendre cet exemple au cas d'un dé à six faces. La distribution de probabilité décrivant l'expérience «lancer le dé» a pour espace d'échantillonnage [1,2,3,4,5,6], chacune de ces valeurs étant associée à la probabilité 1/6.

Les deux distributions précédentes appartiennent à la famille des distributions **discrètes**. Elles servent à décrire des expériences dont le nombre de valeurs possibles est fini. Par opposition, la seconde famille de distributions regroupe les distributions **continues**, décrivant des expériences dont le nombre de résultats possibles est infini. Par exemple, mesurer la taille d'une personne adulte sélectionnée au hasard peut produire un nombre infini de valeurs comprises entre 50 cm et 280 cm. Les distributions sont utiles pour décrire les résultats attendus d'une expérience. Reprenons notre exemple du dé. Nous savons que chaque face a une chance sur six d'être tirée au hasard. Nous pouvons représenter cette distribution avec un graphique (figure 2.3).

Nous avons donc sous les yeux un modèle statistique décrivant le comportement attendu d'un dé, nous l'appelons la distribution **théorique**. Cependant, si nous effectuons l'expérience 10 fois (nous collectons donc un échantillon), nous obtiendrons une distribution différente de cette distribution théorique (figure 2.4).

Nous appelons cette distribution la distribution **empirique**. Chaque échantillon aura sa propre distribution empirique. Cependant, comme le prédit la loi des grands nombres (ou théorème de Bernoulli) : si une expérience est répétée un grand nombre de fois, la probabilité empirique d'un résultat se rapproche de la probabilité théorique à mesure que le nombre de répétitions augmente. Pour nous en convaincre, collectons trois échantillons de lancer de dé de respectivement 30, 100 et 1000 observations (figure 2.5).

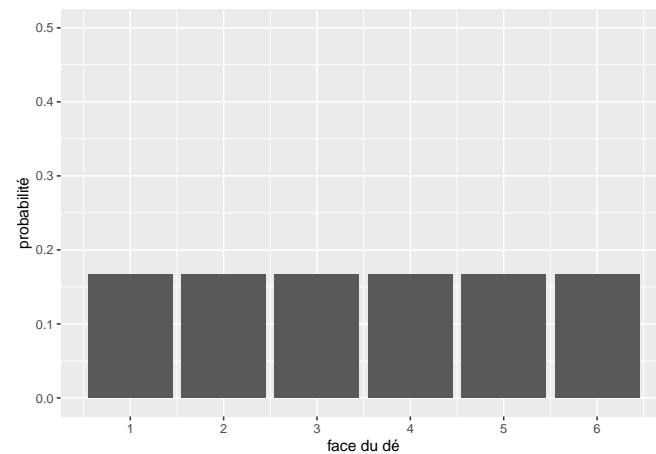


FIG. 2.3 : Distribution théorique d'un lancé de dé

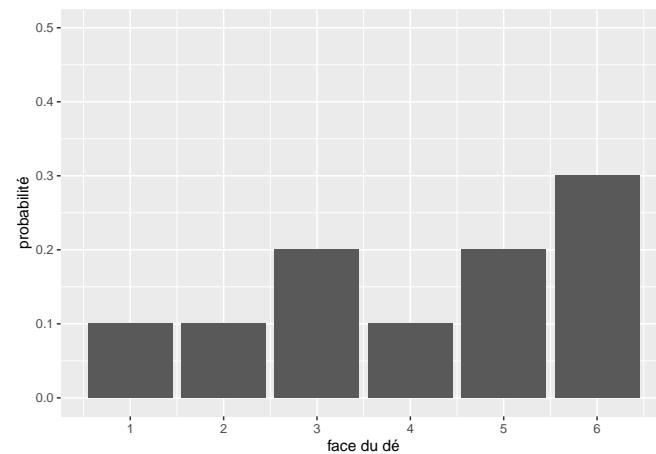


FIG. 2.4 : Distribution empirique d'un lancé de dé ($n=10$)

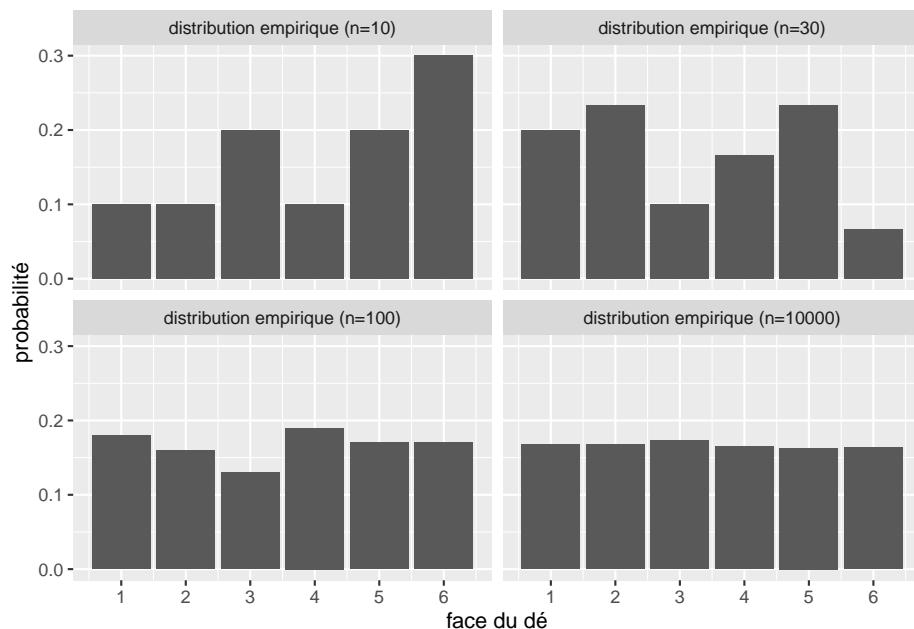


FIG. 2.5 : Distribution empirique d'un lancé de dé ($n=10$)

On constate bien qu'au fur et à mesure que la taille de l'échantillon augmente, on tend vers la distribution théorique. Ces dernières sont donc utilisées pour modéliser des phénomènes réels et sont à la base de presque tous les tests statistiques d'inférence fréquentiste ou bayésienne.

En pratique, la question que l'on se pose le plus souvent est : quelle distribution théorique peut le mieux décrire le phénomène empirique à l'étude ? Pour répondre à cette question, deux approches sont possibles :

- Considérant la littérature existante sur le sujet, les connaissances accumulées et la nature de la variable étudiée, il est possible de sélectionner des distributions théoriques pouvant vraisemblablement correspondre à la variable.
- Comparer visuellement ou à l'aide de tests statistiques la distribution empirique de la variable et diverses distributions théoriques pour trouver la plus adaptée.

Idéalement, le choix d'une distribution théorique devrait reposer sur ces deux méthodes combinées.

2.4.2 Anatomie d'une distribution

Puisqu'une distribution est une fonction, il est possible de la représenter à l'aide d'une formule mathématique (appelée **fonction de masse** pour les distributions discrètes et **fonction de densité** pour les distributions continues). Prenons un premier exemple concret avec la distribution théorique associée au lancer de pièce de monnaie : la distribution de **Bernoulli**. Sa formule est la suivante :

$$f(x; p) = \begin{cases} q = 1 - p & \text{si } x = 0 \\ p & \text{si } x = 1 \end{cases} \quad (2.1)$$

avec p la probabilité d'obtenir $x = 1$ (pile), et $1-p$ la probabilité d'avoir $x = 0$ (face). La distribution de Bernoulli ne dépend que d'un paramètre : p . Avec différentes valeurs de p , on peut obtenir différentes formes pour la distribution de Bernoulli. Si $p = 1/2$, la distribution de Bernoulli décrit parfaitement l'expérience : obtenir pile à un lancer de pièce de monnaie. Si $p = 1/6$, elle décrit alors l'expérience : obtenir 4 (tout comme n'importe quelle valeur de 1 à 6) à un lancer de dé. Pour un exemple plus appliqué, la distribution de Bernoulli est utilisée en analyse spatiale pour étudier la concentration d'accidents de la route ou de crimes en milieu urbain. En chaque endroit du territoire, il est possible de calculer la probabilité qu'un tel événement ait lieu ou non en se basant sur les données observées et cette distribution. La distribution continue la plus simple à décrire est certainement la distribution **uniforme**. Il s'agit d'une distribution un peu spéciale puisqu'elle attribue la même probabilité à toutes ses valeurs dans son espace d'échantillonnage. Elle est définie sur l'intervalle $[-\infty; +\infty]$ et a la fonction de densité suivante :

$$f(x; a; b) = \begin{cases} \frac{1}{b-a} & \text{si } a \leq x \leq b \\ 0 & \text{sinon} \end{cases} \quad (2.2)$$

La fonction uniforme a donc deux paramètres, a et b , représentant respectivement les valeurs maximale et minimale au-delà desquelles les valeurs ont une probabilité 0 d'être obtenues. Pour avoir une meilleure intuition de ce que décrit une fonction de densité, il est intéressant de la représenter avec un graphique (figure 2.6). Notez que sur ce graphique, l'axe des ordonnées n'indique pas précisément la probabilité associée à chaque valeur car celle-ci serait infinidécimale. Il sert uniquement à représenter la densité de la fonction de distribution.

On observe clairement que toutes les valeurs de x entre a et b ont la même probabilité pour chacune de trois distributions uniformes présentées dans le graphique. Plus l'étendue est grande ($a - b$), plus l'espace d'échantillonnage est grand et plus la probabilité totale est répartie dans cet espace. Cette distribution serait donc idéale pour décrire un phénomène pour lequel chaque valeur a autant de chance de se produire qu'une autre. Prenons pour exemple un cas fictif avec un jeu de hasard qui vous proposerait la situation

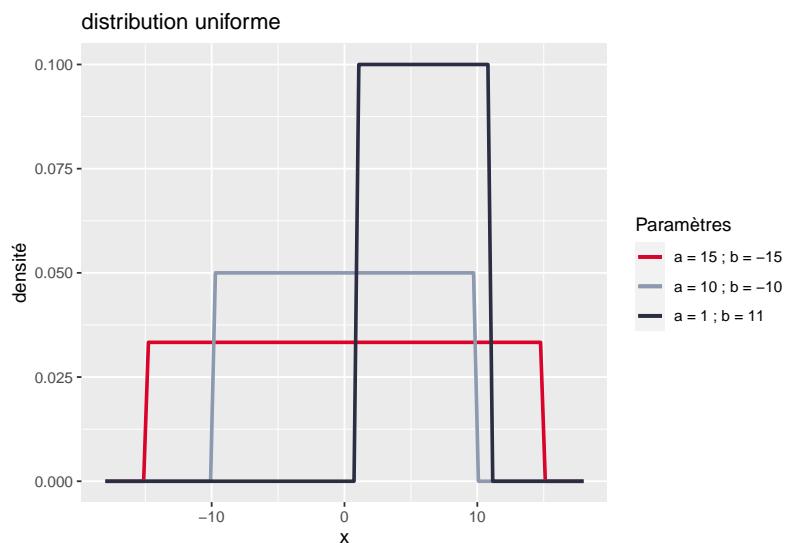


FIG. 2.6 : Distributions uniformes continues

suivante : en tirant sur la manette d'une machine à sous, un nombre est tiré aléatoirement entre -60 et +50. Si le nombre est négatif, vous perdez de l'argent et inversement si le nombre est positif. Nous pouvons représenter cette situation avec une distribution uniforme continue et l'utiliser pour calculer quelques informations essentielles :

1. Selon cette distribution, quelle est la probabilité de gagner de l'argent lors d'un tirage ($x > 0$)?
2. Quelle est la probabilité de perdre de l'argent? ($x < 0$)?
3. Si je perds moins de 30\$ au premier tirage, quelle est la probabilité que ai-je d'au moins récupérer ma mise au second tirage ($x > 30$)?

Il est assez facile de calculer ces probabilités en utilisant la fonction `punif` dans R. Concrètement, cela permet de calculer l'intégrale de la fonction de masse sur un intervalle donné.

```
# Probabilité d'obtenir une valeur supérieure ou égale à 0
punif(0,min = -60, max = 50)
```

```
## [1] 0.5454545
```

```
# Probabilité d'obtenir une valeur inférieure à 0
punif(0,min = -60, max = 50, lower.tail = F)
```

```
## [1] 0.4545455
```

```
# Probabilité d'obtenir une valeur supérieure à 30
punif(30, min = -60, max = 50,lower.tail = F)
```

```
## [1] 0.1818182
```

Les paramètres permettent donc d'ajuster la fonction de masse ou de densité d'une distribution afin de lui permettre de prendre des formes différentes. Certains paramètres vont changer la localisation de la distribution (la déplacer vers la droite ou la gauche de l'axe des X), d'autres son degré de dispersion (distribution pointue ou aplatie) ou encore sa forme (symétrie). Les différents paramètres d'une distribution

correspondent donc à sa carte d'identité et donnent une idée précise sur sa nature.

2.4.3 Principales distributions

Il existe un très grand nombre de distributions théoriques et parmi elles, de nombreuses sont en fait des cas spéciaux d'autres distributions. Pour un petit aperçu du bestiaire, vous pouvez faire un saut à la page Univariate Distribution Relationships²¹, qui liste près de 80 distributions.

Nous nous concentrons ici sur une sélection de 18 distributions très répandues en sciences sociales. La figure ?? présente graphiquement leurs fonctions de masse et de densité présentées dans cette section. Notez que ces graphiques correspondent tous à une forme possible de chaque distribution. En modifiant leurs paramètres, il serait possible de produire une figure très différente. Les distributions discrètes sont représentées avec des graphiques en barres, et les distributions continues avec des graphiques de densité.

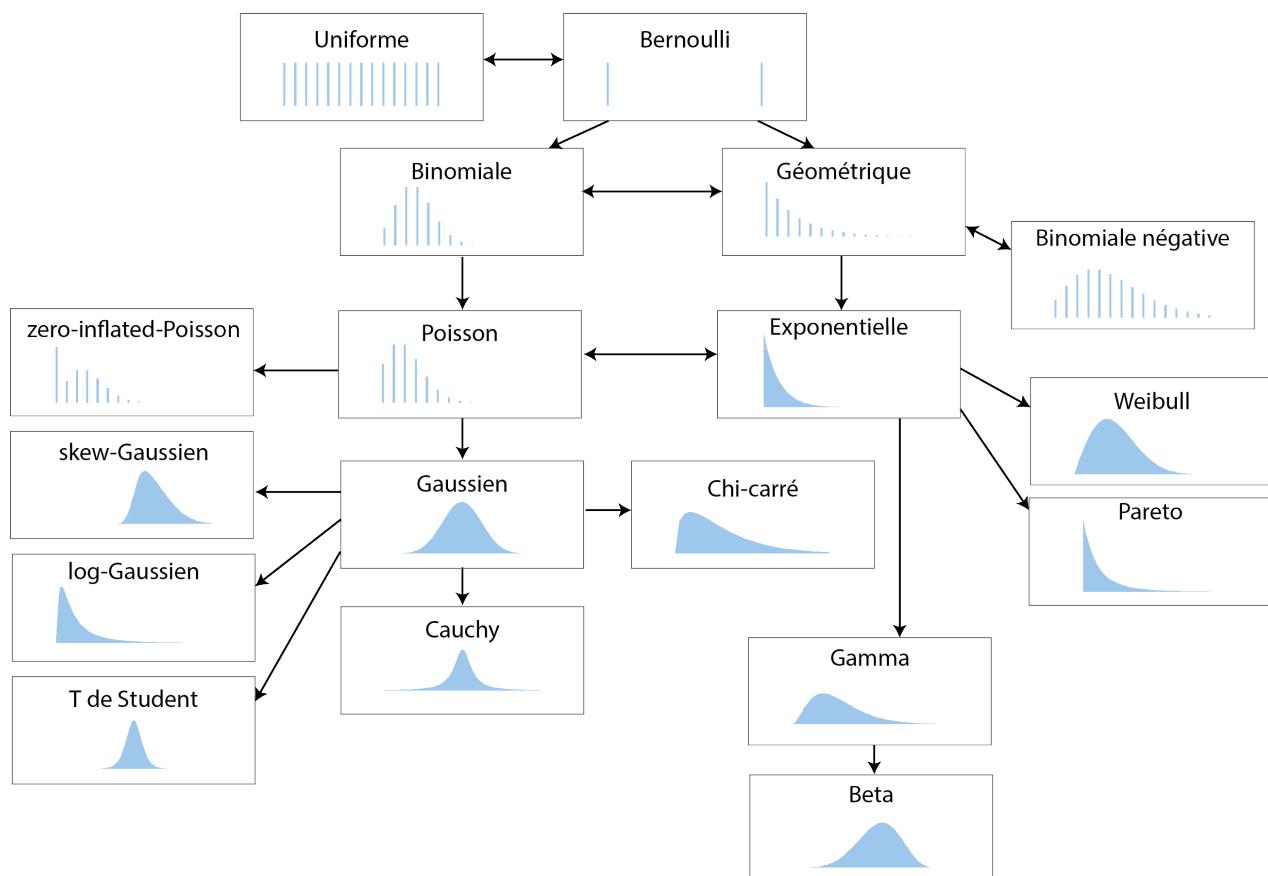


FIG. 2.7 : 18 distributions essentielles, design inspiré de?

2.4.3.1 La distribution uniforme discrète

Nous avons déjà abordé cette distribution dans les exemples précédents. Elle permet de décrire un phénomène dont tous les résultats possibles ont exactement la même probabilité de se produire. L'exemple classique est bien sûr un lancer de dé.

²¹ <http://www.math.wm.edu/~leemis/chart/UDR/UDR.html>

2.4.3.2 La distribution de Bernoulli

La distribution de Bernoulli permet de décrire une expérience pour laquelle deux résultats sont possibles. Son espace d'échantillonnage est donc $[0; 1]$. Sa fonction de masse est la suivante :

$$f(x; p) = \begin{cases} q = 1 - p & \text{si } x = 0 \\ p & \text{si } x = 1 \end{cases} \quad (2.3)$$

avec p , la probabilité d'obtenir $x = 1$ (réussite) et donc $1-p$, la probabilité d'avoir $x = 0$ (échec). La distribution de Bernoulli ne dépend que d'un paramètre : p contrôlant la probabilité de réussite de l'expérience. Notez que si $p = 1/2$, alors la distribution de Bernoulli est également une distribution uniforme. Un exemple d'application de la distribution de Bernoulli en études urbaines serait la modélisation de la survie d'un·e cycliste (1 pour survie, 0 pour décès) lors d'une collision avec une voiture selon une vitesse donnée.

2.4.3.3 La distribution binomiale

La distribution binomiale est utilisée pour caractériser une somme de distributions de Bernoulli. Un exemple simple serait l'accumulation des lancers d'une pièce de monnaie. Si l'on compte le nombre de fois où l'on fait pile, cette expérience est décrite par une distribution binomiale. Son espace d'échantillonnage est donc $[0; +\infty[$ (limité aux nombres entiers). Sa fonction de masse est la suivante :

$$f(x; n) = \binom{n}{x} p^x (1 - p)^{n-x} \quad (2.4)$$

avec x le nombre de tirages réussis sur n essais avec une probabilité p de réussite à chaque tirage. Pour reprendre l'exemple précédent concernant les accidents de la route, une distribution binomiale permettrait de représenter la distribution du nombre de cyclistes survivant·e·s sur dix accidents impliquant une voiture à une intersection.

2.4.3.4 La distribution géométrique

La distribution géométrique permet de représenter le nombre de tirages nécessaires avec une distribution de Bernoulli avant d'obtenir une réussite. Par exemple, avec un lancer de dé, l'idée serait de compter le nombre de lancers nécessaires avant de tomber sur un 6. Son espace d'échantillonnage est donc $[1; +\infty[$ (limité aux nombres entiers). Sa distribution de masse est la suivante :

$$f(x; p) = (1 - p)^x p \quad (2.5)$$

avec x le nombre de tentatives avant d'obtenir une réussite, $f(x)$ la probabilité que le premier succès n'arrive qu'après x tentatives et p la probabilité de réussite à chaque tentative. Cette distribution est notamment utilisée en marketing pour modéliser le nombre d'appels nécessaires avant de réussir une vente.

2.4.3.5 La distribution binomiale négative

La distribution binomiale négative est proche de la distribution géométrique. Elle permet de représenter le nombre de tentatives nécessaires afin d'obtenir un nombre n de réussites $[1; +\infty[$ (limité aux nombres entiers positifs). Sa formule est la suivante :

$$f(x; n; p) = \binom{x + n - 1}{n} p^n (1 - p)^x \quad (2.6)$$

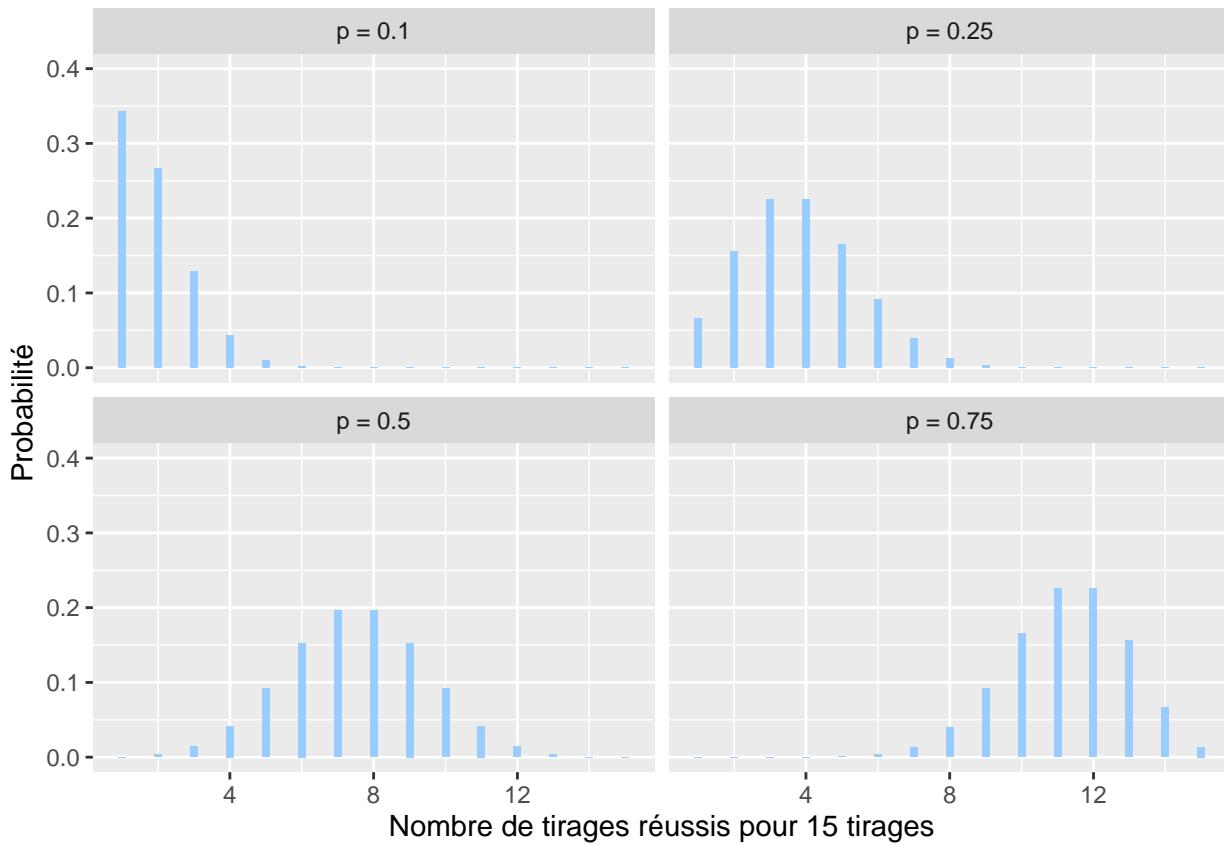


FIG. 2.8 : La distribution binomiale

avec x le nombre de tentatives avant d'obtenir n réussites et p la probabilité d'obtenir une réussite à chaque tentative. Cette distribution pourrait être utilisée pour modéliser le nombre de questionnaires x à envoyer pour une enquête si l'on espère au moins n réponses, sachant que la probabilité d'une réponse est p .

2.4.3.6 La distribution de poisson

La distribution de poisson est utilisée pour modéliser des comptages. Son espace d'échantillonnage est donc $[0; +\infty[$ (limité aux nombres entiers positifs). Par exemple, il est possible de compter à une intersection le nombre de collisions entre des automobilistes et des cyclistes sur une période donnée. Cet exemple devrait vous faire penser à la distribution binomiale vue plus haut. En effet, il serait possible de noter chaque rencontre entre une voiture et un cycliste et de considérer que leur collision est une « réussite » (0 : pas d'accidents, 1 : accident). Cependant, ce type de données serait fastidieux à collecter comparativement au simple comptage des accidents. La distribution de poisson a une fonction de densité avec un seul paramètre λ (lambda) et est décrite par la formule suivante :

$$f(x; \lambda) = \frac{\lambda^x}{x!} e^{-\lambda} \quad (2.7)$$

avec x le nombre de cas, $f(x)$ la probabilité d'obtenir x sachant λ . λ peut être vue comme le taux moyen d'occurrences (nombre d'événements divisé par la durée totale de l'expérience). Il permet à la fois de caractériser le centre et la dispersion de la distribution. Notez également que plus le paramètre λ augmente, plus la distribution de poisson tend vers une distribution normale.

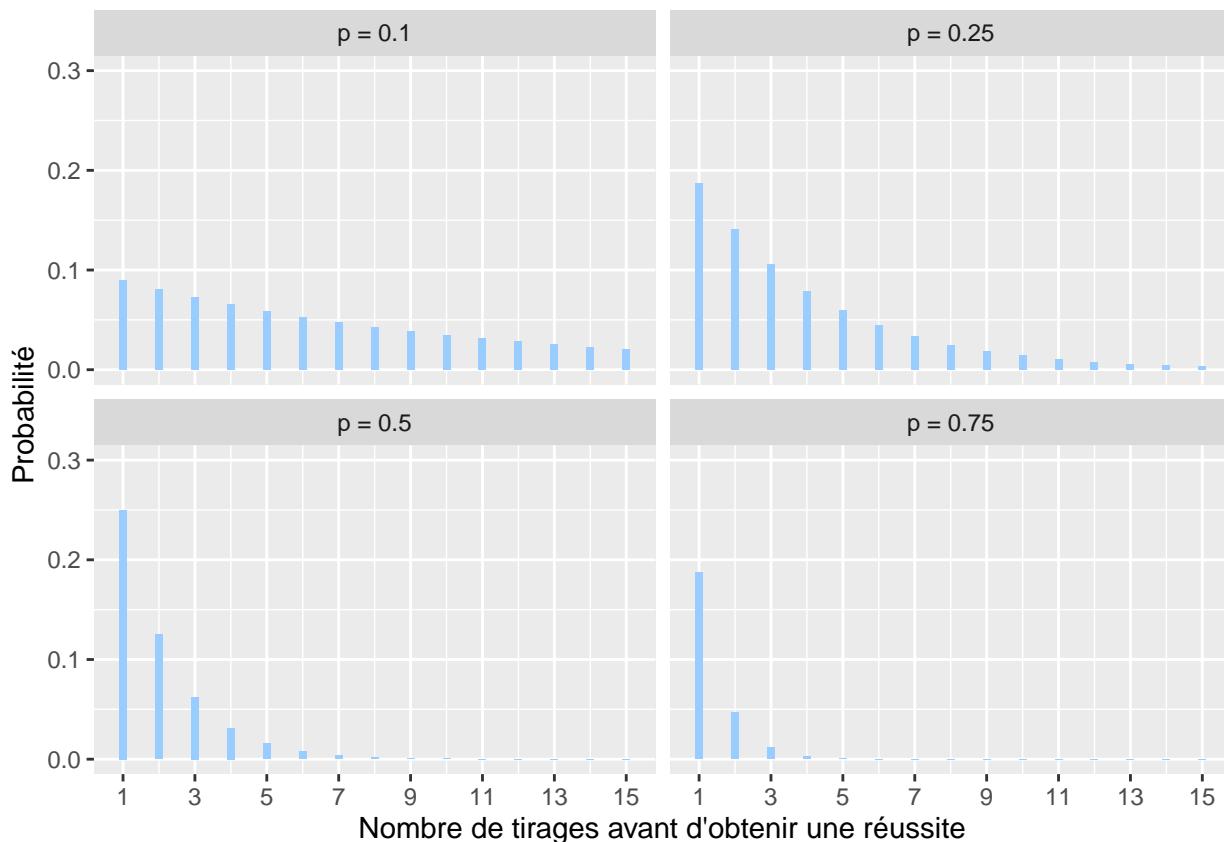


FIG. 2.9 : La distribution géométrique

2.4.3.7 La distribution de poisson avec excès de zéros

Il arrive régulièrement qu'une variable de comptage mesurée produise un très grand nombre de zéros. Prenons pour exemple le nombre de seringues de drogue injectable par tronçon de rue ramassées sur une période d'un mois. À l'échelle de toute une ville, un très grand nombre de tronçons n'auront tout simplement aucune seringue et dans ce contexte, la distribution classique de poisson n'est pas adaptée. On lui préfère alors sa version avec une inflation de zéros qui inclut un paramètre contrôlant la forte présence de zéros. Sa fonction de densité est la suivante :

$$f(x; \lambda; p) = (1 - p) \frac{\lambda^x}{x!} e^{-\lambda} \quad (2.8)$$

Plus exactement, la distribution de poisson avec excès de zéro (zero-inflated en anglais) est une combinaison de deux processus générant des zéros. En effet, un zéro peut être produit par la distribution de poisson originale (aussi appelé vrai zéro) ou alors par le processus menant à la surreprésentation des 0 dans le jeu de données, capturée par la probabilité p (faux zéro). p est donc le paramètre contrôlant la probabilité d'obtenir un zéro, indépendamment du phénomène étudié.

```
## Loading required package: stats4
## Loading required package: splines
##
## Attaching package: 'VGAM'
## The following object is masked from 'package:tidyverse':
##
```

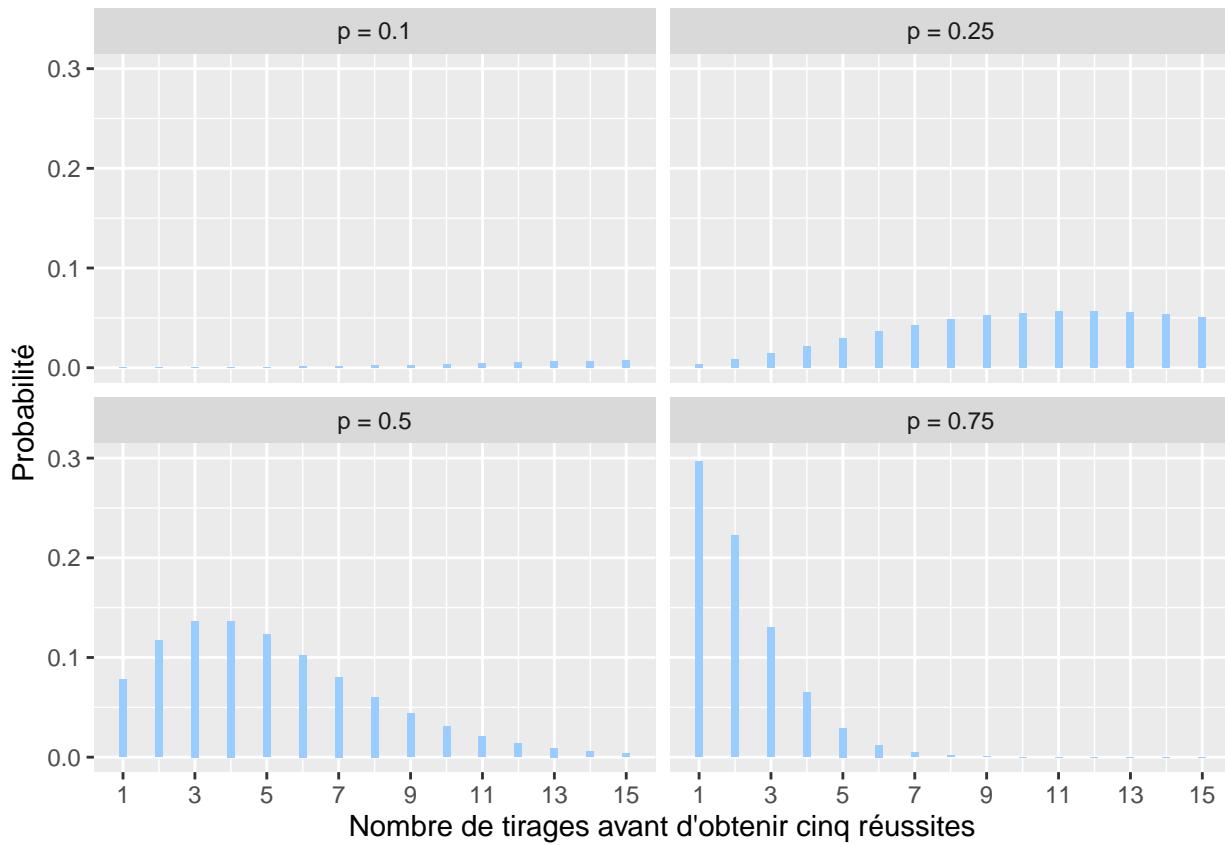


FIG. 2.10 : La distribution binomiale négative

```
##  
##      fill
```

2.4.3.8 La distribution gaussienne

Plus communément appelée la distribution normale, la distribution gaussienne est utilisée pour représenter des variables continues centrées sur leur moyenne. Son espace d'échantillonnage est $]-\infty; +\infty[$. Cette distribution joue un rôle central en statistique. Le théorème central limite stipule que la somme d'un grand nombre de distributions tend généralement vers une distribution normale. Autrement dit, lorsque nous répétons une même expérience et que nous conservons les résultats de ces expériences, la distribution du résultat de ces expériences tend vers la normalité. Ceci s'explique par le fait qu'en moyenne, chaque répétition de l'expérience produit le même résultat, mais qu'un ensemble de petits facteurs aléatoires viennent rajouter de la variabilité dans les données collectées. Prenons un exemple concret, si l'on plante une centaine d'arbres simultanément dans un parc avec un degré d'ensoleillement identique et qu'on leur apporte les mêmes soins pendant dix ans, la distribution de leurs tailles suivra une distribution normale. Un ensemble de facteurs aléatoires (composition du sol, exposition au vent, aléas génétiques, passage de nuages, etc.) auront affecté différemment chaque arbre, ajoutant ainsi un peu de hasard dans leurs tailles finales. Ces dernières seront cependant davantage affectées par des paramètres centraux (espèces, ensoleillement, arrosage, etc.), et seront donc centrées autour d'une moyenne. La fonction de densité de la distribution normale est la suivante :

$$f(x; \mu; \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} \quad (2.9)$$

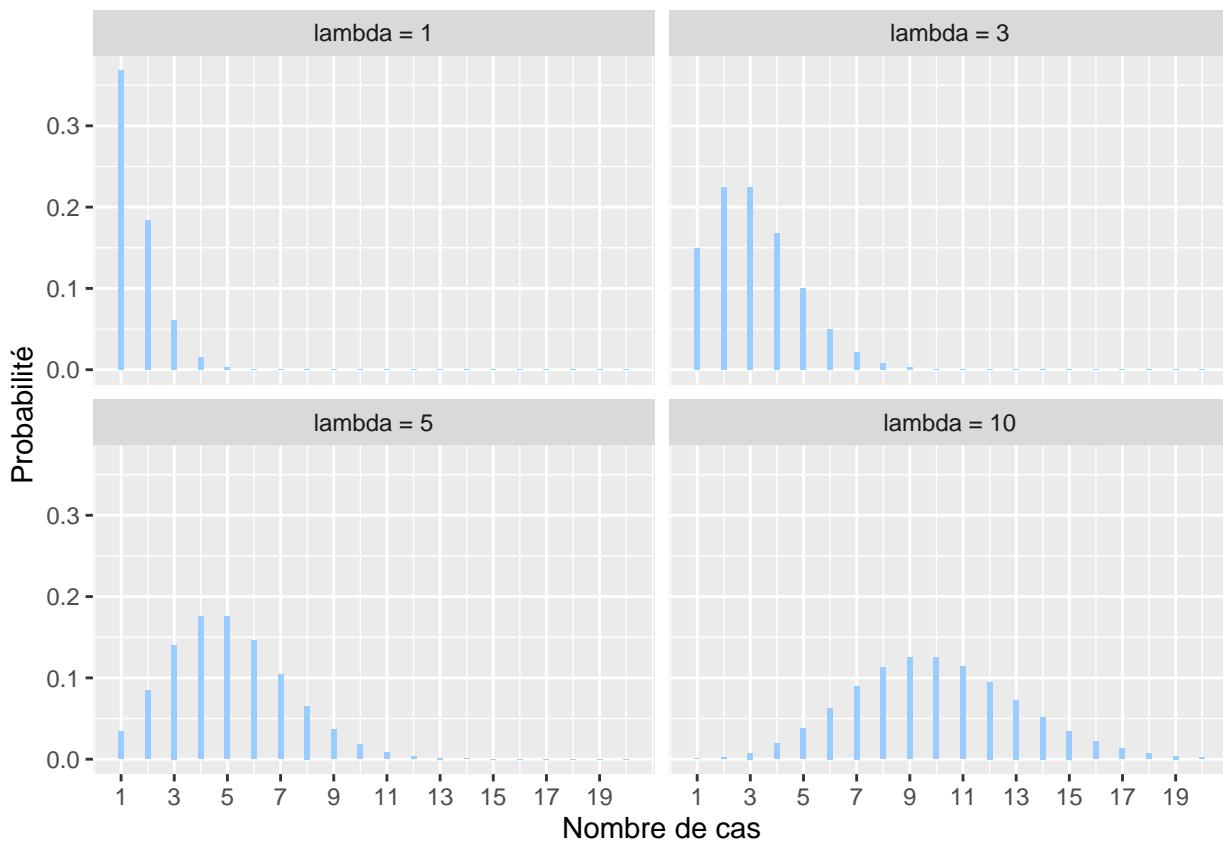


FIG. 2.11 : La distribution de poisson

avec x une valeur dont on souhaite connaître la probabilité, $f(x)$ sa probabilité, μ (mu) la moyenne de la distribution normale (paramètre de localisation) et σ (sigma) son écart-type (paramètre de dispersion). La courbe normale suit une forme de cloche. Notez que :

- 68,2% de la masse de la distribution normale est comprise dans l'intervalle $[\mu - \sigma \leq x \leq \mu + \sigma]$
- 95,4% dans l'intervalle $[\mu - 2\sigma \leq x \leq \mu + 2\sigma]$
- 99,7% dans l'intervalle $[\mu - 3\sigma \leq x \leq \mu + 3\sigma]$

Autrement dit, dans le cas d'une distribution normale, il est très invraisemblable d'observer des données situées à plus de trois écarts types de la moyenne. Notez ici que lorsque $\mu = 0$ et $\sigma = 0$, on obtient la loi normale générale (ou centrée-réduite) (section 2.5.5.2).

2.4.3.9 La distribution gaussienne asymétrique

La distribution normale asymétrique (skew-normal) est une extension de la distribution gaussienne permettant de modifier la forme de la distribution normale pour qu'elle ne soit plus symétrique. Son espace d'échantillonnage est donc $]-\infty; +\infty[$. Sa fonction de densité est la suivante :

$$f(x; \xi; \omega; \alpha) = \frac{2}{\omega\sqrt{2\pi}} e^{-\frac{(x-\xi)^2}{2\omega^2}} \int_{-\infty}^{\alpha} \frac{1}{\sqrt{2\pi}} e^{-\frac{t^2}{2}} dt \quad (2.10)$$

avec ξ (xi) le paramètre de localisation, ω (omega) le paramètre de dispersion (ou d'échelle) et α (alpha) le paramètre de forme (contrôlant le degré de symétrie). Si $\alpha = 0$, alors la distribution skew-normal est une simple distribution normale. Ce type de distribution est très utile lorsque que l'on souhaite modéliser

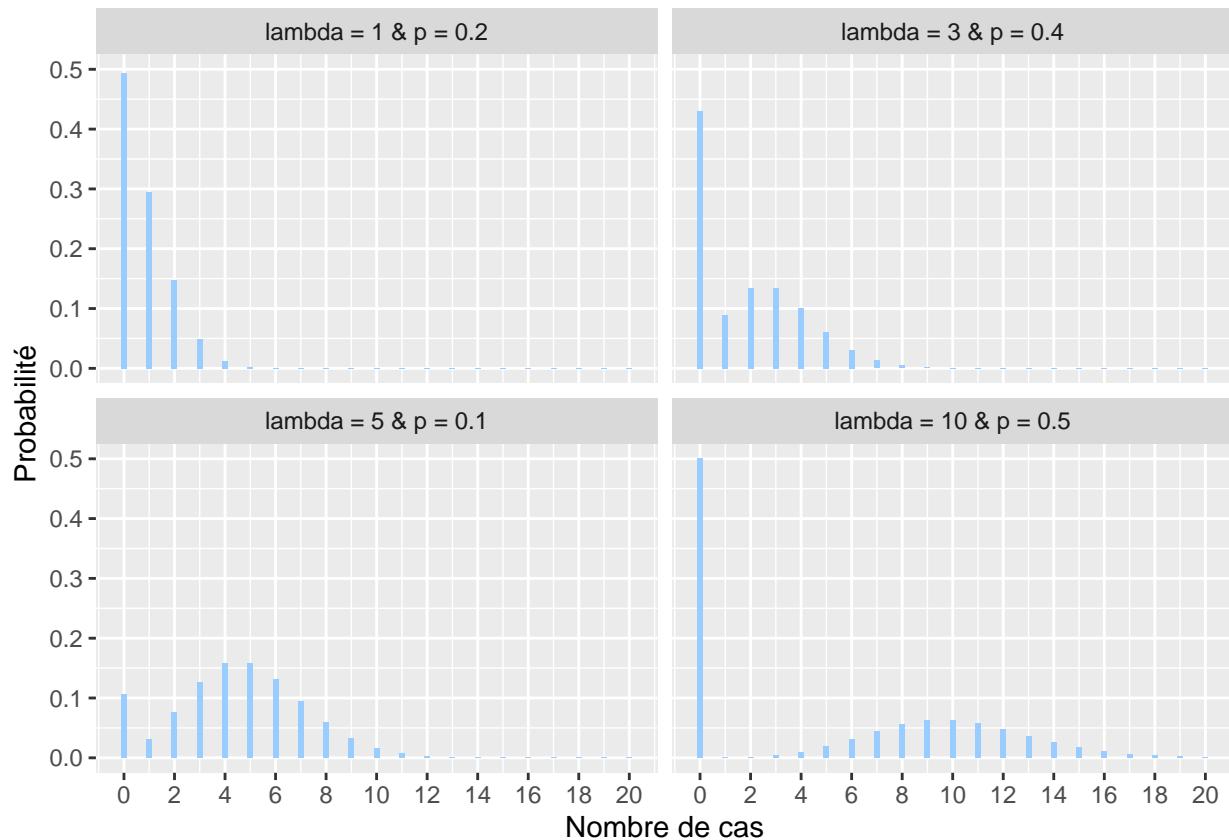


FIG. 2.12 : La distribution de poisson avec excès de zéros

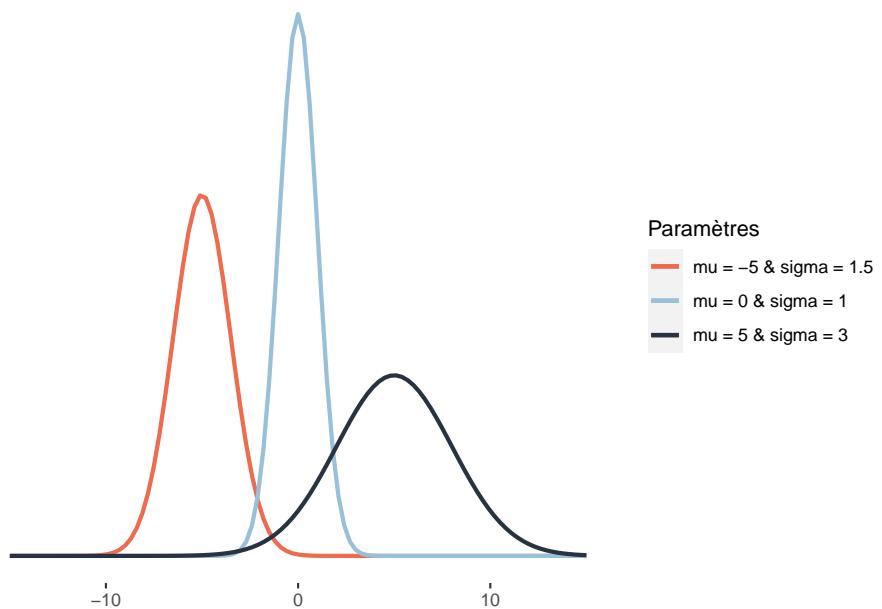


FIG. 2.13 : La distribution Gaussienne

une variable pour laquelle on sait que des valeurs plus extrêmes s'observeront d'un côté ou de l'autre de la distribution. Les revenus totaux annuels des personnes ou des ménages sont de très bons exemples puisqu'ils sont distribués généralement avec une asymétrie positive : bien qu'une moyenne existe, il y a généralement plus de personnes ou de ménages avec des revenus très faibles, que de personnes ou de ménages avec des revenus très élevés.

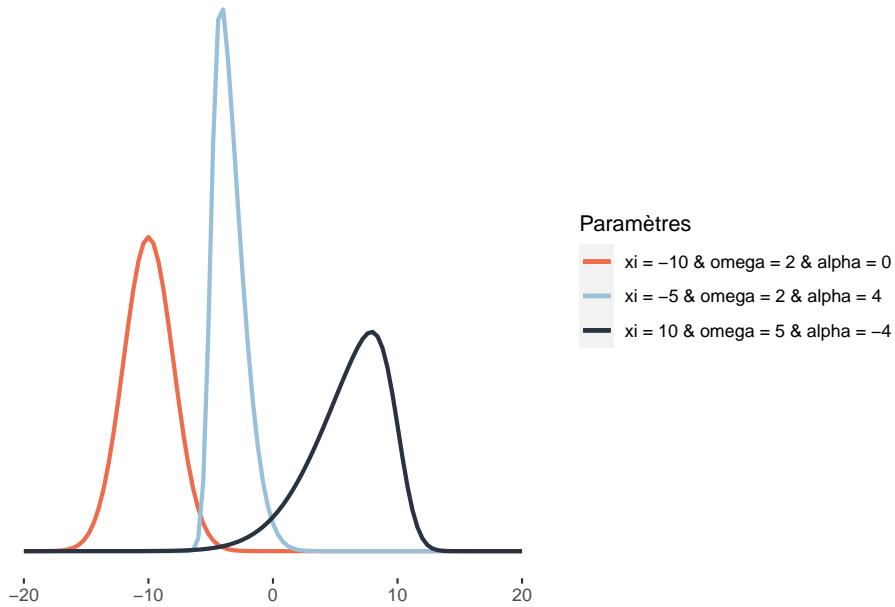


FIG. 2.14 : La distribution skew-Gaussienne

2.4.3.10 La distribution log-normale

Au même titre que la distribution skew-normal, la distribution log-normal est une version asymétrique de la distribution normale. Son espace d'échantillonnage est $]0; +\infty[$. Cela signifie que cette distribution ne peut décrire que des données continues et positives. Sa fonction de densité est la suivante :

$$f(x; \mu, \sigma) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-\left(\frac{(\ln x - \mu)^2}{2\sigma^2}\right)} \quad (2.11)$$

À la différence la distribution skew-normal, la distribution log-normal ne peut avoir qu'une asymétrie positive (étirée vers la droite). Elle est cependant intéressante puisqu'elle ne compte que deux paramètres (μ et σ) ce qui la rend plus facile à ajuster. À nouveau, une distribution log-normal pourrait être utilisée pour décrire les revenus totaux annuels des individus ou des ménages ou les revenus d'emploi. Elle est aussi utilisée en économie sur les marchés financiers pour représenter les cours des actions et des biens (ces derniers ne pouvant pas être inférieurs à 0).

2.4.3.11 La distribution de Student

La distribution de Student joue un rôle important en statistique, elle est par exemple utilisée lors du test t pour calculer le degré de significativité du test. Comme la distribution gaussienne, la distribution de Student a une forme de cloche, est centrée sur sa moyenne et définie sur $]-\infty; +\infty[$. Elle a cependant des « queues plus lourdes » (*heavy tails* en anglais). Entendez par-là que les valeurs extrêmes ont une plus grande probabilité d'occurrence dans une distribution de Student que dans une distribution gaussienne. Sa fonction de densité est la suivante :

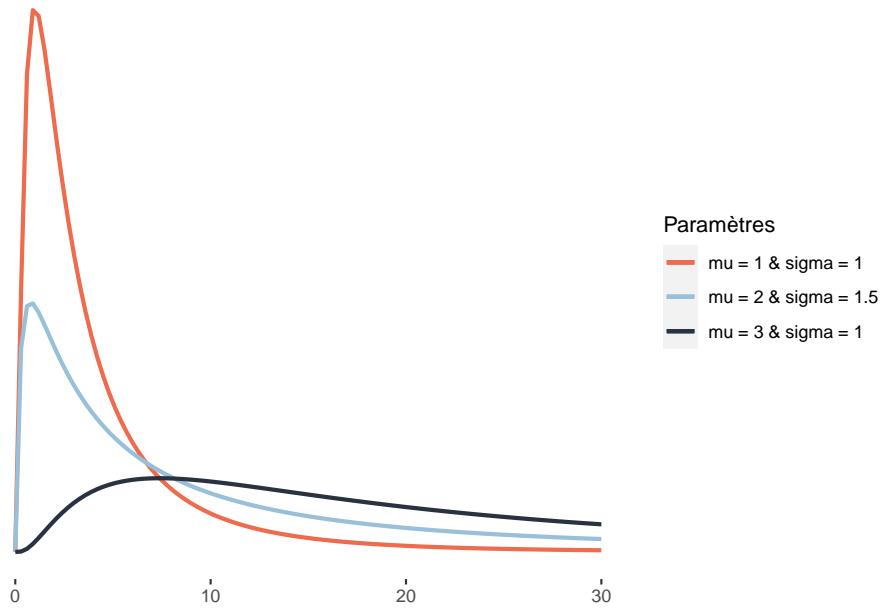


FIG. 2.15 : La distribution log-gaussienne

$$p(x; \nu; \hat{\mu}; \hat{\sigma}) = \frac{\Gamma(\frac{\nu+1}{2})}{\Gamma(\frac{\nu}{2}) \sqrt{\pi\nu} \hat{\sigma}} \left(1 + \frac{1}{\nu} \left(\frac{x - \hat{\mu}}{\hat{\sigma}} \right)^2 \right)^{-\frac{\nu+1}{2}} \quad (2.12)$$

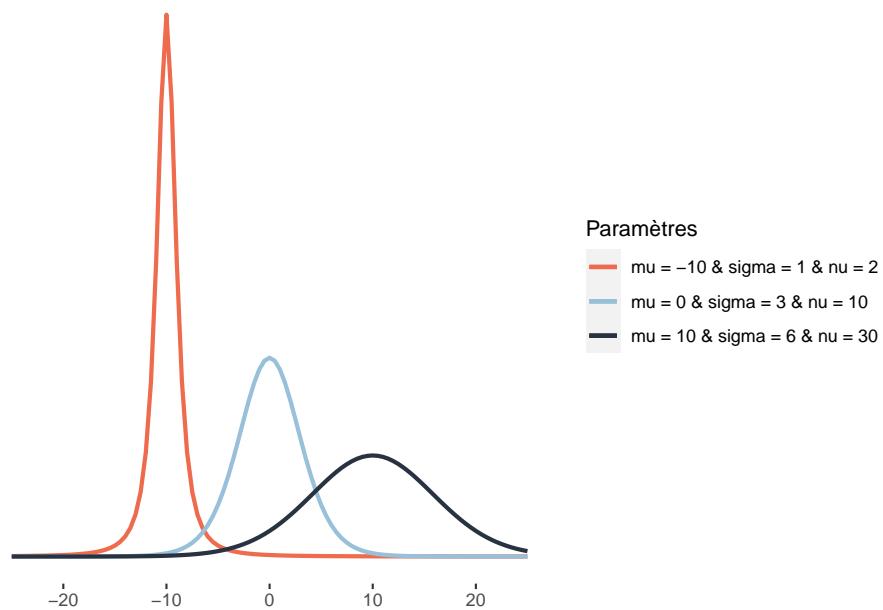
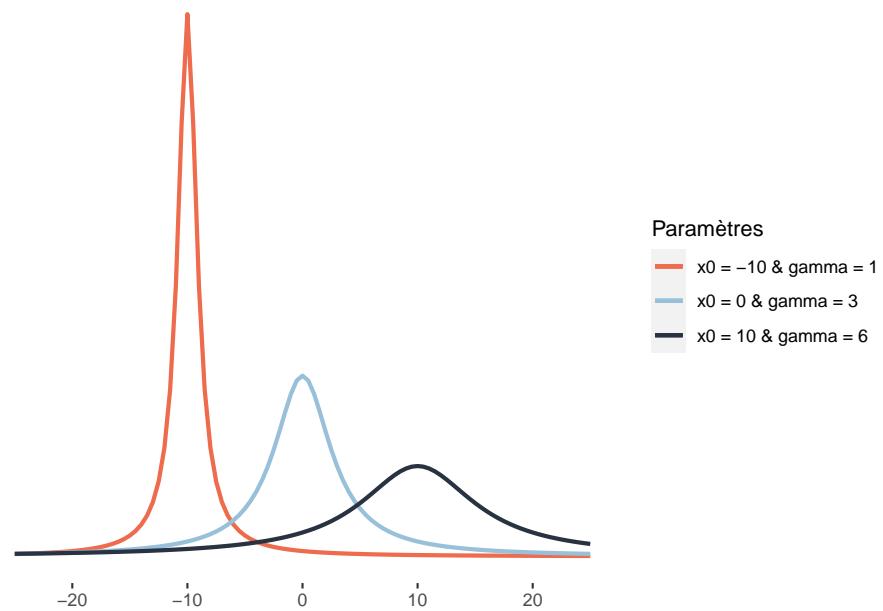
avec μ le paramètre de localisation, σ le paramètre de dispersion (qui n'est cependant pas un écart-type comme pour la distribution normale) et ν le nombre de degré de liberté. Plus ν est grand, plus la distribution de Student tend vers une distribution normale. Γ représente la fonction mathématique gamma (à ne pas confondre avec la distribution de Gamma). Un exemple d'application en études urbaines serait l'exposition au bruit environnemental de cyclistes. Cette distribution s'approcherait certainement d'une distribution normale, mais les cyclistes croisent régulièrement des secteurs peu bruyants (parcs, rues résidentielles, etc.) et des secteurs très bruyants (artères majeures, zones industrielles, etc.), ce qui conduit vers une distribution de Student.

2.4.3.12 La distribution de Cauchy

La distribution de Cauchy est également une distribution symétrique définie sur l'intervalle $]-\infty; +\infty[$. Elle a comme particularité d'avoir des queues potentiellement plus lourdes que la distribution de Student. Elle est notamment utilisée pour modéliser des phénomènes extrêmes comme les précipitations maximales annuelles, les niveaux d'inondations maximaux annuels ou les *values at risk* pour les portefeuilles financiers. Il est également intéressant de noter que le quotient de deux variables indépendantes normalement distribuées suit une distribution de Cauchy. Sa fonction de densité est la suivante :

$$\frac{1}{\pi\gamma} \left[\frac{\gamma^2}{(x - x_0)^2 + \gamma^2} \right] \quad (2.13)$$

Elle dépend donc de deux paramètres : x_0 , le paramètre de localisation indiquant le pic de la distribution et γ , un paramètre de dispersion.

**FIG. 2.16 :** La distribution de Student**FIG. 2.17 :** La distribution de Cauchy

2.4.3.13 La distribution du Chi-carré

La distribution du Chi² est utilisée dans de nombreux tests statistiques. Spécifiquement, le test du Chi² de Pearson est utilisé pour comparer les écarts au carré entre des fréquences attendues et observées de deux variables qualitatives. La distribution du Chi² décrit donc les sommes des carrés d'un nombre k de variables indépendantes normalement distribuées. Il est assez rare de modéliser un phénomène à l'aide d'une distribution du Chi², mais son omniprésence dans les tests statistiques justifie qu'elle soit mentionnée ici. Cette distribution est définie sur l'intervalle $[0; +\infty[$ et a pour fonction de densité :

$$f(x; k) = \frac{1}{2^{k/2}\Gamma(k/2)}x^{k/2-1}e^{-x/2} \quad (2.14)$$

Cette fonction n'a qu'un paramètre k , représentant donc le nombre de variables au carré sommées pour obtenir la distribution du Chi²

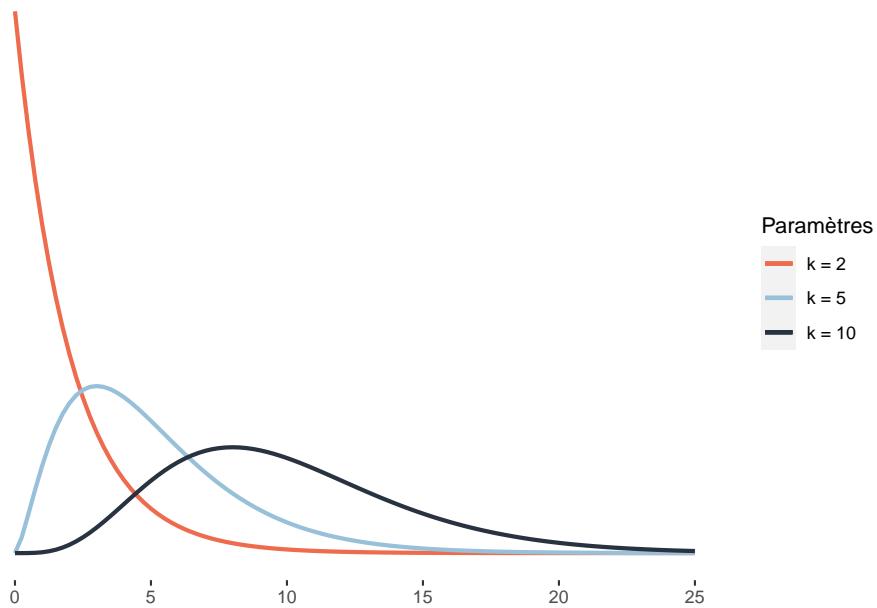


FIG. 2.18 : La distribution du Chi²

2.4.3.14 La distribution exponentielle

La distribution exponentielle est une version continue de la distribution géométrique. Pour cette dernière, on s'intéresserait au nombre de tentatives nécessaires pour obtenir un résultat positif, soit une dimension discrète. Pour la distribution exponentielle, cette dimension discrète est remplacée par une dimension continue. L'exemple le plus intuitif est sûrement le cas du temps. Dans ce cas, la distribution exponentielle servirait à décrire le temps d'attente nécessaire pour qu'un évènement se produise. Il pourrait aussi s'agir d'une force que l'on applique jusqu'à ce qu'un matériau cède. Cette distribution est donc définie sur l'intervalle $[0; +\infty[$ et a pour fonction de densité :

$$f(x; \lambda) = \lambda e^{-\lambda x} \quad (2.15)$$

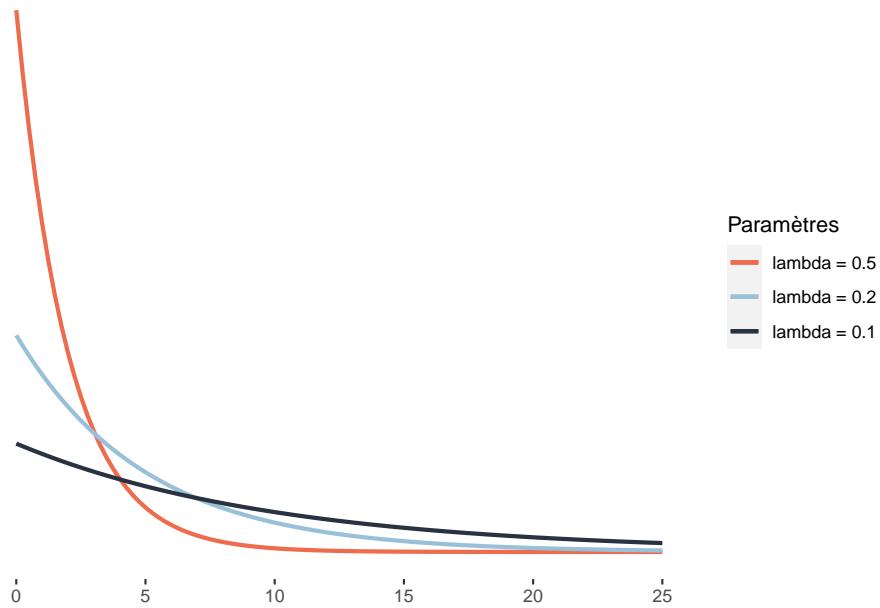


FIG. 2.19 : La distribution exponentielle

2.4.3.15 La distribution de Gamma

La distribution de Gamma est une généralisation d'un grand nombre de distributions. Elle regroupe ainsi la distribution exponentielle et du Chi2. En d'autres termes, les distributions du chi2 et exponentielles sont des cas particuliers de la distribution de Gamma. Cette distribution est définie sur l'intervalle $]0; +\infty[$ (notez que le 0 est exclu) et sa fonction de densité est la suivante :

$$f(x; \alpha; \beta) = \frac{\beta^\alpha x^{\alpha-1} e^{-\beta x}}{\Gamma(\alpha)} \quad (2.16)$$

Elle comprend donc deux paramètres : α et β . Le premier est le paramètre de forme et le second un paramètre d'échelle (à l'inverse d'un paramètre de dispersion, plus sa valeur est petite, plus la distribution sera dispersée). Notez que cette distribution ne dispose pas d'un paramètre de localisation. Du fait de sa flexibilité, cette distribution est largement utilisée, que ce soit dans la modélisation des temps d'attente avant un évènement, la taille des réclamations d'assurance, les quantités de précipitations, etc.

2.4.3.16 La distribution de Beta

La distribution de Beta est définie sur l'intervalle $[0; 1]$, elle est donc énormément utilisée pour représenter des variables étant des proportions ou des probabilités. Elle a aussi une utilité pratique en statistique, car en combinaison avec d'autres distributions, elle permet de modéliser leurs paramètres de probabilité (distribution beta-binomial, beta-negative-binomial, etc.). Un autre usage plus rare, mais intéressant est la modélisation de la fraction du temps représentée par une tâche dans le temps nécessaire à la réalisation de deux tâches de façon séquentielle. Ceci est dû au fait que la distribution d'une distribution gamma $g1$ divisée par la somme de $g1$ et d'une autre distribution gamma $g2$, suit une distribution beta. Un exemple concret serait par exemple la fraction du temps effectué à pied dans un déplacement multimodal. La distribution de beta a la fonction de densité suivante :

$$f(x; \alpha; \beta) = \frac{1}{B(\alpha, \beta)} x^{\alpha-1} (1-x)^{\beta-1} \quad (2.17)$$

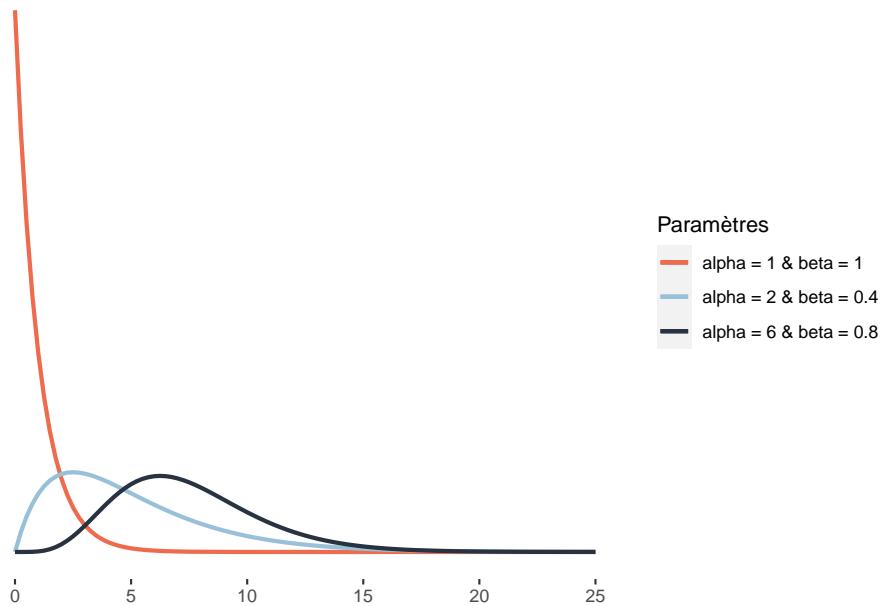


FIG. 2.20 : La distribution de Gamma

Elle a donc deux paramètres α et β contrôlant tous les deux la forme de la distribution. Cette caractéristique lui permet d'avoir une très grande flexibilité et même d'adopter des formes bimodales. B correspondant à la fonction mathématique Beta, à ne pas confondre avec la distribution de Beta et le paramètre Beta (β) de cette même distribution.

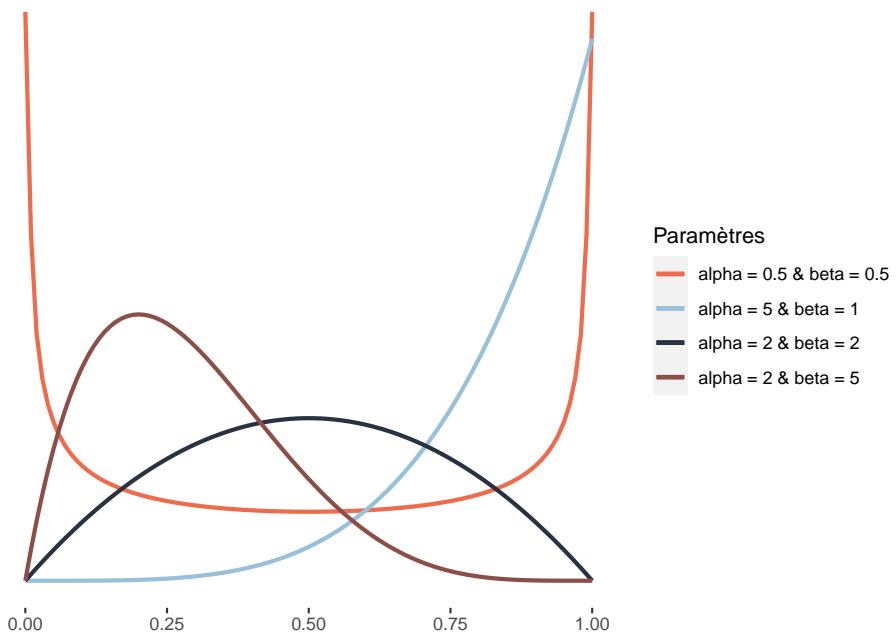


FIG. 2.21 : La distribution de Beta

2.4.3.17 La distribution de Weibull

La distribution de Weibull est directement liée à la distribution exponentielle, cette dernière étant en fait un cas particulier de distribution de Weibull. Elle sert donc à représenter une quantité x (souvent le temps) à accumuler pour qu'un évènement se produise. La distribution de Weibull est définie sur l'intervalle $[0; +\infty[$ et a la fonction de densité suivante :

$$f(x; \lambda) = \frac{k}{\lambda} \left(\frac{x}{\lambda}\right)^{k-1} e^{-(\frac{x}{\lambda})^k} \quad (2.18)$$

λ est le paramètre de dispersion (analogue à celui d'une distribution exponentielle classique) et k le paramètre de forme. Pour bien comprendre le rôle de k , prenons un exemple : la propagation d'un champignon d'un arbre à son voisin. Si $k < 1$, cela signifie que la probabilité que l'évènement modélisé se produise diminue avec le temps. En d'autres termes, dans de nombreux cas la contamination se fait rapidement. Si $k = 1$, alors la probabilité que l'évènement se produise reste stable dans le temps. Si $k > 1$, alors la probabilité que l'évènement se produise augmente avec le temps, ce qui signifie une augmentation des risques de contamination à mesure que les deux arbres restent à proximité. La distribution de Weibull est très utilisée en analyse de survie, en météorologie, en ingénierie des matériaux et dans la théorie des valeurs extrêmes.

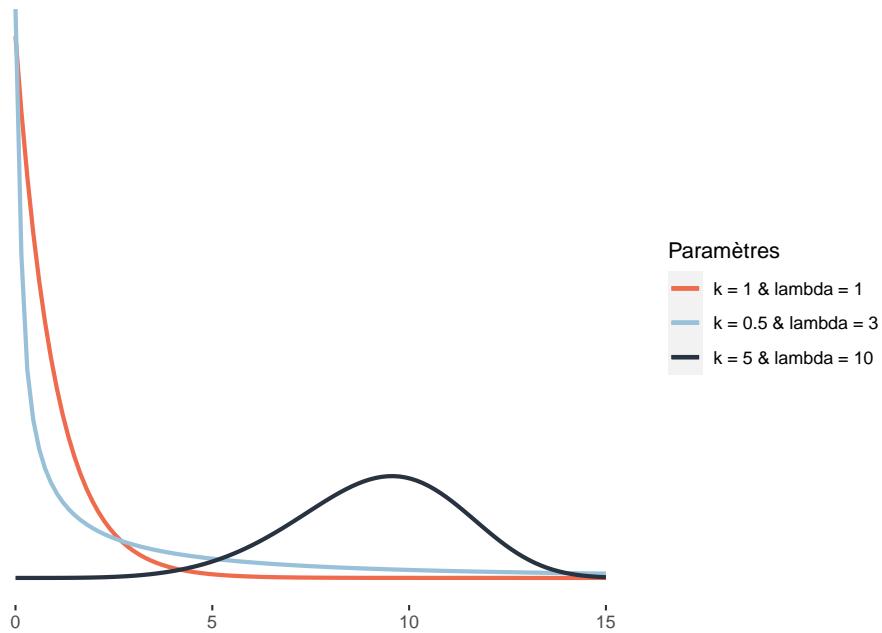


FIG. 2.22 : La distribution de Weibull

2.4.3.18 La distribution de Pareto

La distribution de Pareto est à la distribution exponentielle ce que la distribution log-normal est à la distribution gaussienne : la distribution de l'exponentiel (e) de cette distribution originale. Elle est définie sur l'intervalle $[x_m; +\infty[$ avec la fonction de densité suivante :

$$f(x; x_m; k) = \left(\frac{x_m}{x}\right)^k \quad (2.19)$$

Elle comprend donc deux paramètres, x_m étant un paramètre de localisation (décalant la distribution vers la droite ou vers la gauche) et k un paramètre de forme. Plus k augmente, plus la probabilité prédictive

par la distribution décroît rapidement.

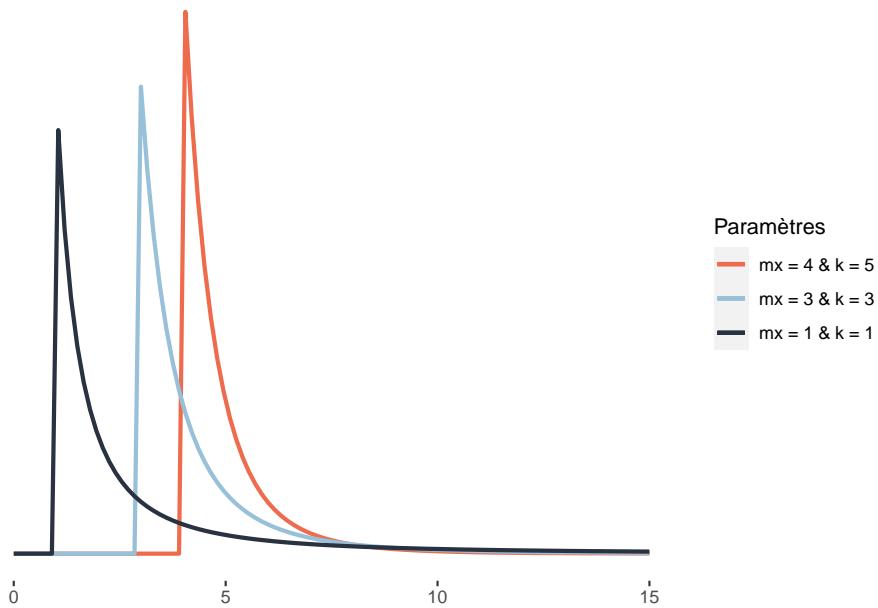


FIG. 2.23 : La distribution de Pareto

Originalement, le mathématicien Pareto a utilisé cette fonction pour décrire la répartition du capital parmi la population puisqu'une large partie du capital est détenue par une petite fraction de la population. Elle peut également être utilisée pour décrire la répartition de la taille des villes (?), la popularité des hommes sur tinder²² ou la taille des fichiers échangés sur internet (?). Pour ces trois exemples, nous avons une situation avec : de nombreuses petites villes, profils peu attractifs, petits fichiers échangés et à l'inverse très peu de grandes villes, profils très attractifs, gros fichiers échangés.

2.4.3.19 Cas particuliers

Sachez également qu'il existe des formes «plus exotiques» de distributions que nous n'abordons pas ici, mais auxquelles vous pourriez être confrontés un jour :

- Les distributions sphériques, servant à décrire des données dont le 0 est équivalent à la valeur maximale. Par exemple, des angles puisque 0 et 360 degrés sont identiques.
- Les mixtures de distributions, décrivant des combinaisons de distributions. Par exemple, la distribution de la taille de tous les humains est en réalité un mixte entre deux distributions gaussiennes, une pour chaque sexe, puisque ces deux sous-distributions n'ont pas la même moyenne ni le même écart-type.
- Les distributions multivariées, permettant de décrire des phénomènes multidimensionnels. Par exemple, la réussite des élèves en français et en mathématique pourrait être modélisée comme une distribution gaussienne bivariée plutôt que deux distributions distinctes.
- Les distributions censurées décrivant des variables pour lesquels des valeurs sont possibles au-delà d'une certaine limite mais que l'on est incapable de mesurer. Un bon exemple serait la mesure de la pollution sonore avec un capteur incapable de détecter des niveaux sonores en dessous de 55 décibels. Il arrive parfois en ville que les niveaux sonores soient si faibles, mais les données collectées ne le montrent pas. Dans ce contexte, il est important d'utiliser des versions censurées

²²<https://medium.com/@worstonlinedater/tinder-experiments-ii-guys-unless-you-are-really-hot-you-are-probably-better-off-not-wasting-your-2ddf370a6e9a>

des distributions présentées précédemment. Les observations au-delà de la limite sont conservées dans l'analyse, mais nous ne disposons que d'une information partielle à leur égard.

- Les distributions tronquées, souvent confondues avec les distributions censurées, décrivent des situations ou des données qui au-delà d'une certaine limite sont retirées simplement de l'analyse.

2.4.4 Conclusion sur les distributions

Voilà qui conclut cette exploration des principales distributions à connaître. L'idée n'est bien sûr pas de toutes les retenir par cœur (et encore moins les formules mathématiques), mais plutôt de se rappeler dans quels contextes elles peuvent être utiles ; et de revenir au besoin sur ce chapitre. Vous aurez certainement besoin de le relire avant d'aborder le chapitre portant sur les modèles linéaires généralisés (GLM). Wikipedia dispose d'informations très détaillées sur chaque distribution si vous avez besoin d'informations complémentaires. Pour un tour d'horizon plus exhaustif des distributions, vous pouvez aussi faire un tour sur les projets probonto²³ et the ultimate probability distribution explorer²⁴.

2.5 Statistiques descriptives sur des variables quantitatives

2.5.1 Les paramètres de tendance centrale

Trois mesures de tendance centrale permettent de résumer rapidement une variable quantitative :

- la **moyenne arithmétique** est simplement la somme des données d'une variable divisée par le nombre d'observations (n), soit $\frac{\sum_{i=1}^n x_i}{n}$ notée μ (prononcez *mu*) pour des données pour une population et \bar{x} (prononcez *x barre*) pour un échantillon.
- la **médiane** est la valeur qui coupe la distribution d'une variable d'une population ou d'un échantillon en deux parties égales. Autrement dit, 50% des valeurs des observations lui sont supérieures et 50% lui sont inférieures.
- le **mode** est la valeur la plus fréquente parmi un ensemble d'observations pour une variable. Il s'applique ainsi à des variables discrètes (avec un nombre fini de valeurs discrètes dans un intervalle donné) et non à des variables continues (avec un nombre infini de valeurs réelles dans un intervalle donné). Prenons deux variables, l'une discrète relative au nombre d'accidents par intersection (avec $X \in [0, 20]$) et l'autre continue relative à la distance de dépassement (en mètres) d'un-e cycliste par un véhicule motorisé (avec $X \in [0, 5]$). Pour la première, le mode – la valeur la plus fréquente – est certainement 0. Pour la seconde, identifier le mode n'est pas pertinent puisqu'il peut y avoir un nombre infini de valeurs entre 0 et 5 mètres.

Il convient de ne pas confondre moyenne et médiane ! Dans le tableau 2.1, nous avons reporté les valeurs moyennes et médianes des revenus des ménages pour les municipalités de l'île de Montréal en 2015. Par exemple, les 8685 ménages résidant à Westmount disposaient en moyenne d'un revenu de 295099\$; la moitié de ces 8685 ménages avaient un revenu inférieur à 100153\$ et l'autre moitié un revenu supérieur à cette valeur (médiane). Cela démontre clairement que la moyenne peut être grandement affectée par des valeurs extrêmes (faibles ou fortes) ; autrement dit, plus l'écart entre les valeurs de la moyenne et la médiane est importante, plus les données de la variable sont inégalement réparties. À Westmount, soit la municipalité la plus nantie de l'île de Montréal, les valeurs extrêmes sont des ménages avec des revenus très élevés tirant fortement la moyenne vers le haut. À l'inverse, le faible écart entre les valeurs moyenne et médiane dans la municipalité de Montréal-Est (58594\$ versus 50318\$) soulignent que les revenus des ménages sont plus également répartis. Cela explique que pour comparer les revenus totaux ou d'emploi

²³<https://sites.google.com/site/probonto/screenshots>

²⁴<https://blog.wolfram.com/2013/02/01/the-ultimate-univariate-probability-distribution-explorer/>

entre différents groupes (selon le sexe, le groupe d'âge, le niveau d'éducation, la municipalité ou région métropolitaine, etc.), on prévilegio habituellement l'utilisation des revenus médians.

2.5.2 Les paramètres de position

Les paramètres de position permettent de diviser une distribution en n parties égales.

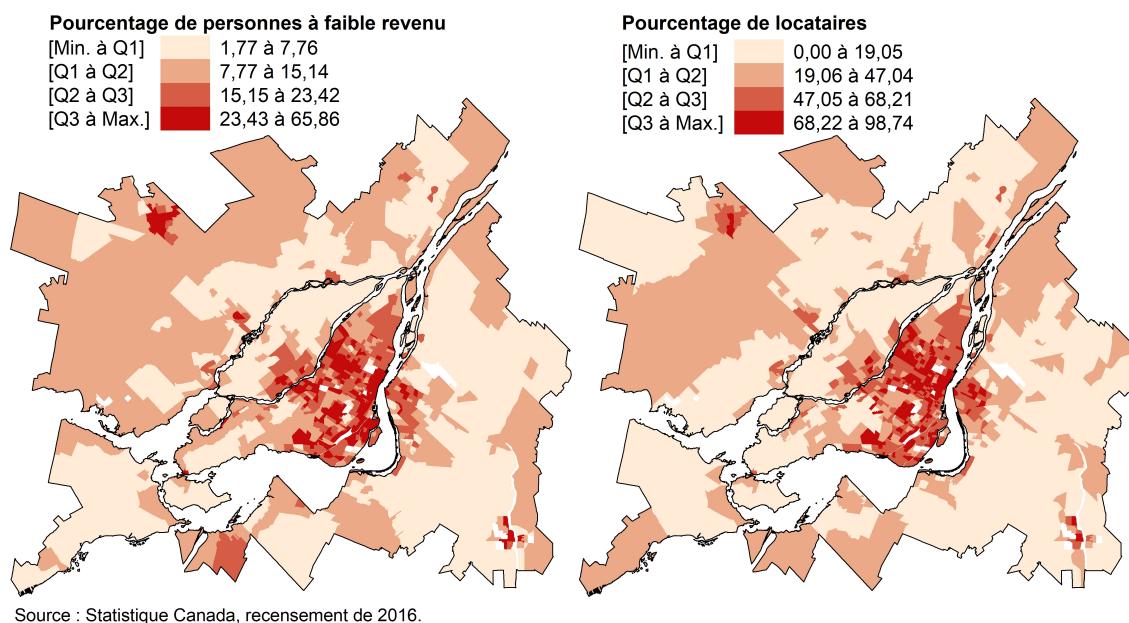
- Les **quartiles** qui divisent une distribution en quatre parties (25%) :
 - Q1 (25%), soit le quartile inférieur ou premier quartile ;
 - Q2 (50%), soit la médiane ;
 - Q3 (75%), soit le quartile supérieur ou troisième quartile.
- Les **quintiles** qui divisent une distribution en cinq parties égales (20%).
- Les **déciles** (de D1 à D9) qui divisent une distribution en dix parties égales (10%).
- Les **centiles** (de C1 à C99) qui divisent une distribution en cent parties égales (1%).

En cartographie, les quartiles et les quintiles sont souvent utilisés pour discréteriser une variable quantitative (continue ou discrète) en quatre ou cinq classes et plus rarement, en huit ou dix classes. Avec les quartiles, les bornes des classes qui comprendront chacune 25% des unités spatiales seront ainsi définies comme suit : [Min à Q1], [Q1 à Q2], [Q2 à Q3] et [Q3 à Max]. La méthode de discréterisation selon les quartiles ou quintiles permet alors de repérer, en un coup d'œil, à quelle tranche de 25% ou 20% des données appartient chacune des unités spatiales. Cette méthode de discréterisation est aussi utile pour comparer plusieurs cartes et vérifier si deux phénomènes sont ou non colocalisés (?). En guise d'exemple, les pourcentages de personnes à faible revenu et de locataires par secteur de recensement ont clairement des distributions spatiales très semblables dans la région métropolitaine de Montréal en 2016 (figure 2.24).

Une lecture attentive des valeurs des centiles permet de repérer la présence de valeurs extrêmes voire aberrantes dans un jeu de données. Il n'est donc pas rare de les voir reportées dans un tableau de statistiques descriptives d'un article scientifique, et ce, afin de décrire succinctement les variables à l'étude. Par exemple, dans une étude récente comparant les niveaux d'exposition au bruit des cyclistes dans trois villes (?), les auteurs reportent à la fois les valeurs moyennes et celles de plusieurs centiles. Globalement, la lecture des valeurs moyennes permet de constater que, sur la base des données collectées, les cyclistes sont plus exposés au bruit à Paris qu'à Montréal et Copenhague (73,4 dB(A) contre 70,7 et 68,4, tableau 2.2). Compte tenu de l'échelle logarithmique du bruit, la différence de 5 dB(A) entre les valeurs moyennes du bruit de Copenhague et de Paris peut être considérée comme une multiplication de l'énergie sonore

TAB. 2.1 : Revenus moyens et médians des ménages en dollars, municipalités de l'île de Montréal, 2015

Municipalité	Nombre de ménages	Revenu moyen	Revenu médian
Baie-D'Urfé	1 330	171 390	118 784
Beaconsfield	6 660	187 173	123 392
Côte-Saint-Luc	13 490	94 570	58 935
Dollard-Des Ormeaux	17 210	102 104	78 981
Dorval	8 390	89 952	64 689
Hampstead	2 470	250 497	122 496
Kirkland	6 685	144 676	115 381
Montréal	779 805	69 047	50 227
Montréal-Est	1 730	58 594	50 318
Montréal-Ouest	1 850	159 374	115 029
Mont-Royal	7 370	205 309	109 540
Pointe-Claire	12 380	100 294	80 242
Sainte-Anne-de-Bellevue	1 960	102 969	67 200
Senneville	345	203 790	116 224
Westmount	8 685	295 099	100 153



Source : Statistique Canada, recensement de 2016.

FIG. 2.24 : Exemples de cartographie avec une discréétisation selon les quantiles

par plus de 3. Pour Paris, l'analyse des quartiles montre que durant 25% du temps des trajets à vélo (plus de 63 heures de collecte), les participants ont été exposés à des niveaux de bruit soit inférieurs à 69,1 dB(A) (premier quartile), soit supérieurs à 74 dB(A). Quant à l'analyse des centiles, elle permet de constater que durant 5% et 10% du temps, les participants étaient exposés à des niveaux de bruit très élevés, dépassant 77 dB(A) (C90=76 et C90=77,2).

2.5.3 Les paramètres de dispersion

Cinq principales mesures de dispersion permettent d'évaluer la variabilité des valeurs d'une variable quantitative : l'étendue, l'écart interquartile, la variance, l'écart-type et le coefficient de variation. Notez d'emblée que cette dernière mesure ne s'applique pas à des variables d'intervalle (section 2.1.2.2).

- **L'étendue** est la différence entre les valeurs minimale et maximale d'une variable, soit l'intervalle des valeurs dans lequel elle a été mesurée. Il convient d'analyser avec prudence cette mesure puis-

TAB. 2.2 : Stastistiques descriptives de l'exposition au bruit des cyclistes par minute dans trois villes (dB(A), Laeq 1min)

Statistiques	Copenhague	Montréal	Paris
N	6 212,0	4 723,0	3 793,0
Moyenne de bruit	68,4	70,7	73,4
Centiles			
1	57,5	59,2	62,3
5	59,1	61,1	65,0
10	60,3	62,3	66,5
25 (premier quartile)	62,7	64,5	69,1
50 (médiane)	66,0	67,7	71,6
75 (troisième quartile)	69,2	71,0	74,0
90	71,9	73,7	76,0
95	73,3	75,2	77,2
99	76,5	78,9	81,0

qu'elle inclut dans son calcul des valeurs potentiellement extrêmes voire aberrantes (faibles ou fortes).

- **L'intervalle ou écart interquartile** est la différence entre les troisième et premier quartiles ($Q3 - Q1$). Il représente ainsi une mesure de la dispersion des valeurs de 50% des observations centrales de la distribution. Plus la valeur de l'écart interquartile est élevée, plus la dispersion des 50% des observations centrales est forte. Contrairement à l'étendue, cette mesure élimine l'influence des valeurs extrêmes puisqu'elle ne tient pas compte des 25% des observations les plus faibles [Min à $Q1$] et des 25% des observations les plus fortes [$Q3$ à Max]. Graphiquement, l'intervalle interquartile est représenté à l'aide d'une boîte à moustaches (*boxplot* en anglais) : plus l'intervalle interquartile sera grand, plus la boîte sera allongée (figure 2.25)

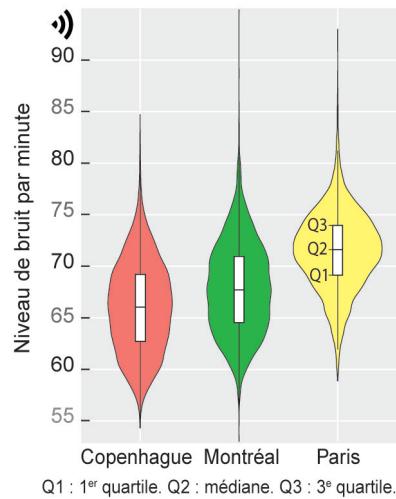


FIG. 2.25 : Graphique en violon, boîte à moustaches et intervalle interquartile

- **La variance** est la somme des déviations à la moyenne au carré (numérateur) divisée par le nombre d'observations pour une population (σ^2) ou divisée par le nombre d'observations moins une (s^2) pour un échantillon (eq. (2.20)). Puisque les déviations à la moyenne sont mises au carré, la valeur de la variance (tout comme celle de l'écart-type) sera toujours positive. Plus sa valeur est élevée, plus les observations sont dispersées autour de la moyenne. La variance représente ainsi l'écart au carré moyen des observations à la moyenne.

$$\sigma^2 = \frac{\sum_{i=1}^n (x_i - \mu)^2}{n} \text{ ou } s^2 = \frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n-1} \quad (2.20)$$

- **L'écart-type** est la racine carrée de la variance (eq. (2.21)). Rappelez-vous que la variance est calculée à partir des déviations à la moyenne mises au carré. Étant donné que l'écart-type est la racine carrée de la variance, il est donc évalué dans les mêmes unités que la variable, contrairement à la variance. Bien entendu, comme pour la variance, plus la valeur de l'écart-type est élevée, plus la distribution des observations autour de la moyenne est dispersée.

$$\sigma = \sqrt{\sigma^2} = \sqrt{\frac{\sum_{i=1}^n (x_i - \mu)^2}{n}} \text{ ou } s = \sqrt{s^2} = \sqrt{\frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n-1}} \quad (2.21)$$



Les formules des variances et des écart-types pour une population et un échantillon sont très similaires : seul le dénominateur change avec n versus $n - 1$ observations. Par conséquent, plus le nombre d'observations de

votre jeu de données sera important, plus l'écart entre ces deux mesures de dispersion pour une population et un échantillon sera minime.

Comme dans la plupart des logiciels de statistique, les fonctions de base `var` et `sd` de R calculent la variance et l'écart-type pour un échantillon ($n - 1$ au dénominateur). Si vous souhaitez les calculer pour une population, adaptez la syntaxe ci-dessous dans laquelle `df$var1` représente la variable intitulée `var1` présente dans un *dataframe* nommé `df`.

```
var.p <- mean((df$var1 - mean(df$var1))^2)
sd.p <- sqrt(mean((df$var1 - mean(df$var1))^2))
```

- **Le coefficient de variation (CV)** est le rapport entre l'écart-type et la moyenne, représentant ainsi une standardisation de l'écart-type ou, en d'autres termes, une mesure de dispersion relative (eq. (2.22)). L'écart-type étant exprimé dans l'unité de mesure de la variable, il ne peut pas être utilisé pour comparer les dispersions de variables exprimées des unités de mesure différentes (par exemple, en pourcentage, en kilomètres, en dollars, etc.). Pour y remédier, on utilisera le coefficient de variation : une variable est plus dispersée qu'une autre si la valeur de son CV est plus élevée. Certains préféreront multiplier la valeur du CV par 100 : l'écart-type est alors exprimé en pourcentage de la moyenne.

$$CV = \frac{\sigma}{\mu} \text{ ou } CV = \frac{s^2}{\bar{x}} \quad (2.22)$$

Illustrons comment calculer les cinq mesures de dispersion précédemment décrites à partir de valeurs fictives pour huit observations (colonne intitulée x_i au tableau 2.3). Les différentes statistiques reportées dans ce tableau sont calculées comme suit :

- La **moyenne** est la somme divisée par le nombre d'observations, soit $248/8 = 31$.
- L'**étendue** est la différence entre les valeurs maximale et minimale, soit $40 - 22 = 30$.
- Les quartiles coupent la distribution en quatre parties égales. Avec huit observations triées par ordre croissant, le **premier quartile** est égale à la valeur de la 2^e observation (soit 25), la **médiane** à celle de la 4^e (30), le **troisième quartile** à celle de la 6^e (35).
- L'**écart interquartile** est la différence entre Q3 et Q1, soit $35 - 25 = 10$.
- La seconde colonne du tableau est l'écart à la moyenne ($x_i - \bar{x}$), soit $22 - 31 = -9$ pour l'observation 1 ; la somme de ces écarts est toujours égale à 0. La troisième colonne est cette déviation mise au carré ($(x_i - \bar{x})^2$), soit $-9^2 = 81$, toujours pour l'observation 1. La somme de ces déviations à la moyenne au carré (268) représente le numérateur de la variance (eq. (2.20)). En divisant cette somme par le nombre d'observations, on obtient la **variance pour une population** ($268/8 = 33,5$) tandis que la **variance d'un échantillon** est égale à $268/(8 - 1) = 38,29$.
- L'**écart-type** est la racine carrée de la variance (eq. (2.21)), soit $\sigma = \sqrt{33,5} = 5,79$ et $s = \sqrt{38,29} = 6,19$.
- Finalement, les valeurs des coefficients de variation (eq. (2.22)) sont de $5,79/31 = 0,19$ pour une population et $6,19/31 = 0,20$ pour un échantillon.

Le tableau 2.4 vise à démontrer à partir de trois variables comment certaines mesures de dispersion sont sensibles à l'unité de mesure et/ou aux valeurs extrêmes.

Concernant l'**unité de mesure**, nous avons créé deux variables *A* et *B*, avec *B* étant simplement *A* multiplié par 10. Pour *A*, les valeurs de la moyenne, l'étendue et l'intervalle interquartile sont respectivement de

TAB. 2.3 : Calcul des mesures de dispersion sur des données fictives

Observation	x_i	$x_i - \bar{x}$	$(x_i - \bar{x})^2$
1	22,00	-9	81,0
2	25,00	-6	36,0
3	27,00	-4	16,0
4	30,00	-1	1,0
5	32,00	1	1,0
6	35,00	4	16,0
7	37,00	6	36,0
8	40,00	9	81,0
Statistique			
N	8,00		
Somme	248,00	0	268,0
Moyenne (\bar{x} ou μ)	31,00	0	33,5
Étendue	18,00		
Premier quartile	25,00		
Troisième quartile	35,00		
Intervalle interquartile	10,00		
Variance (population, σ^2)	33,50		
Écart-type (population, σ)	5,79		
Variance (échantillon, s^2)	38,29		
Écart-type (échantillon, s)	6,19		
Coefficient de variation (σ/μ)	0,19		
Coefficient de variation (s/\bar{x})	0,20		

31, 18 et 10. Sans surprise, celles de B sont multipliées par 10 (310, 180, 100). La variance étant la moyenne des déviations à la moyenne au carré, elle est égale à 33,50 pour A et donc à $33,50 \times 10^2 = 3350$ pour B ; l'écart-type de B est égal à celui de A multiplié par 10. Cela démontre que l'étendue, l'intervalle interquartile, la variance et l'écart-type sont des mesures de dispersion dépendantes de l'unité de mesure. Par contre, le coefficient de variation (CV) étant le rapport de l'écart-type avec la moyenne, il a la même valeur pour A et B, ce qui démontre que CV est bien une mesure de dispersion relative permettant de comparer des variables exprimées dans des unités de mesure différentes.

Concernant la **sensibilité aux valeurs extrêmes**, nous avons créé la variable C pour laquelle seule la huitième observation a une valeur différente (40 pour A et 105 pour B). Cette valeur de 105 pourrait être soit une valeur extrême positive mesurée, soit une valeur aberrante (par exemple, si l'unité de mesure était un pourcentage variant de 0 à 100%). Cette valeur a un impact important sur la moyenne (31 contre 39,12) et l'étendue (18 contre 83) et corollairement sur la variance (33,50 contre 641,86), l'écart-type (5,79 contre 25,33) et le coefficient de variation (0,19 contre 0,65). Par contre, l'intervalle interquartile étant calculé sur 50% des observations centrales ($Q3 - Q1$), il n'est pas affecté par cette valeur extrême.

2.5.4 Les paramètres de forme

2.5.4.1 Vérifier la normalité d'une variable quantitative



De nombreuses méthodes statistiques qui seront abordées dans les chapitres suivants – entre autres, la corrélation de Pearson, les test t et l'analyse de variance, les régressions simple et multiple – requièrent que la variable quantitative suive une **distribution normale** (nommée aussi **distribution gaussienne**).

Dans cette sous-section, nous décrirons trois démarches pour vérifier si la distribution d'une variable est normale : les coefficients d'asymétrie et d'aplatissement (*skewness* et *kurtosis* en anglais), les graphiques (histogramme avec courbe normale, diagramme quantile-quantile), les tests de normalité (tests de Shapiro-

TAB. 2.4 : Illustration de la sensibilité des mesures de dispersion à l’unité de mesure et aux valeurs extrêmes

Observation	A	B	C
1	22,00	220,00	22,00
2	25,00	250,00	25,00
3	27,00	270,00	27,00
4	30,00	300,00	30,00
5	32,00	320,00	32,00
6	35,00	350,00	35,00
7	37,00	370,00	37,00
8	40,00	400,00	105,00
Statistique			
Moyenne (μ)	31,00	310,00	39,12
Étendue	18,00	180,00	83,00
Intervalle interquartile	10,00	100,00	10,00
Variance (population, σ^2)	33,50	3 350,00	641,86
Écart-type (population, σ)	5,79	57,88	25,33
Coefficient de variation (σ/μ)	0,19	0,19	0,65

TAB. 2.5 : Résumé de la sensibilité de la moyenne et des mesures de dispersion

Statistique	Unité de mesure	Valeurs extrêmes
Moyenne	X	X
Étendue	X	X
Intervalle interquartile	X	
Variance	X	X
Écart-type	X	X
Coefficient de variation		X

Wilk, Kolmogorov-Smirnov, Lilliefors, Anderson-Darling et Jarque-Bera).

Il est vivement recommandé de réaliser les trois démarches !

Une distribution est normale quand elle est symétrique et mésokurtique (figure 2.26).

2.5.4.1.1 Vérifier la normalité avec les coefficients d’asymétrie et d’aplatissement

Une distribution est dite symétrique quand la moyenne arithmétique est au centre de la distribution, c'est-à-dire que les observations sont bien réparties de part et d'autre de la moyenne qui sera alors égale à la médiane et au mode (on utilisera uniquement le mode pour une variable discrète et non pour une variable continue). Pour évaluer l'asymétrie, on utilise habituellement le coefficient d'asymétrie (*skewness* en anglais).

Sachez toutefois qu'il existe trois façons (formules) pour le calculer (?) : g_1 est la formule classique (eq. (2.23), disponible dans R avec la fonction `skewness` du package **moments**), G_1 est une version ajustée (eq. (2.24), utilisée dans les logiciels SAS et SPSS notamment) et b_1 est une autre version ajustée (eq. (2.25), utilisée par les logiciels MINITAB et BMDP). Nous verrons qu'avec les packages **DescTools** ou **e1071**, il est possible de calculer ces trois méthodes. Aussi, pour des grands échantillons ($n > 100$), il y a très peu de différences entre les résultats produits par ces trois formules (?). Quelle que soit la formule utilisée, le coefficient d'assymétrie s'interprète comme suit (figure 2.27) :

- quand la valeur du *skewness* est négative, la distribution est asymétrique négative. La distribution

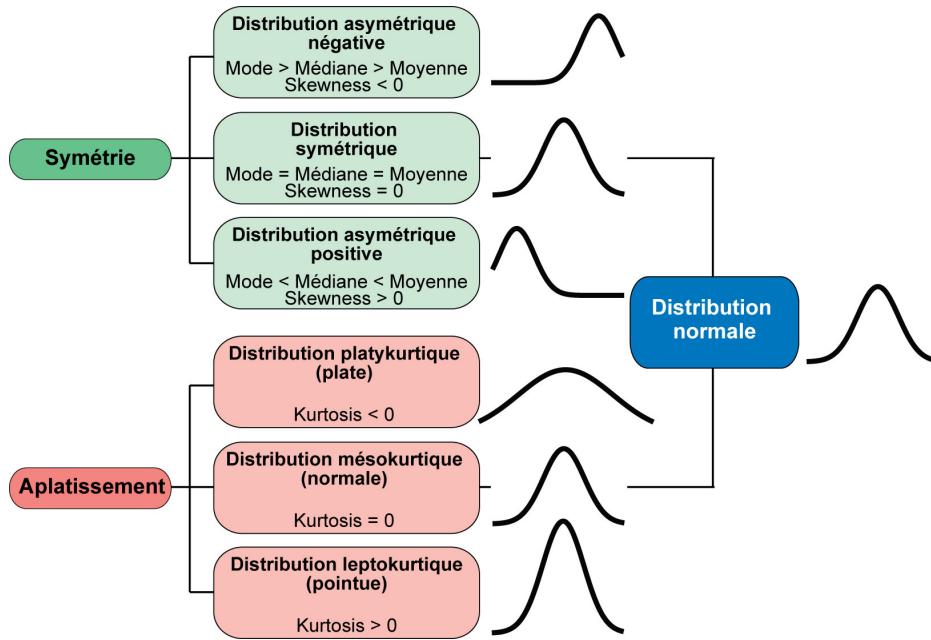


FIG. 2.26 : Formes d'une distribution et les coefficients d'asymétrie et d'aplatissement

est alors tirée à gauche par des valeurs extrêmes faibles, mais peu nombreuses. On emploie souvent l'expression *la queue de distribution* est étirée vers la gauche. La moyenne est alors inférieure à la médiane.

- quand la valeur du *skewness* est égale à 0, la **distribution est symétrique** (la médiane sera égale à la moyenne). Pour une variable discrète, les valeurs du mode, de la moyenne et de la médiane seront égales.
- quand la valeur du *skewness* est positive, la **distribution est symétrique positive**. La distribution est alors tirée à droite par des valeurs extrêmes fortes, mais peu nombreuses. La queue de distribution est alors étirée vers la droite. La moyenne est alors supérieure à la médiane. En sciences sociales, les variables de revenu (totaux ou d'emploi, des individus ou des ménages) ont souvent des distributions asymétriques positives : la moyenne est affectée par quelques observations avec des valeurs de revenu très élevées et est ainsi supérieure à la médiane. En études urbaines, la densité de population pour des unités géographiques d'une métropole donnée (secteur de recensement par exemple) a aussi souvent une distribution asymétrique positive : quelques secteurs de recensement au centre de la métropole sont caractérisés par des valeurs de densité très élevées qui tirent la distribution vers la droite.

$$g_1 = \frac{\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^3}{\left[\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2 \right]^{\frac{3}{2}}} \quad (2.23)$$

$$G_1 = \frac{\sqrt{n(n-1)}}{n-2} g_1 \quad (2.24)$$

$$b_1 = \left(\frac{n-1}{n} \right)^{\frac{3}{2}} g_1 \quad (2.25)$$

Pour évaluer l'aplatissement d'une distribution, on utilisera le coefficient d'aplatissement (*kurtosis* en anglais). Là encore, il existe trois formules pour le calculer (eq. (2.26), (2.27), (2.28)) qui renverront des

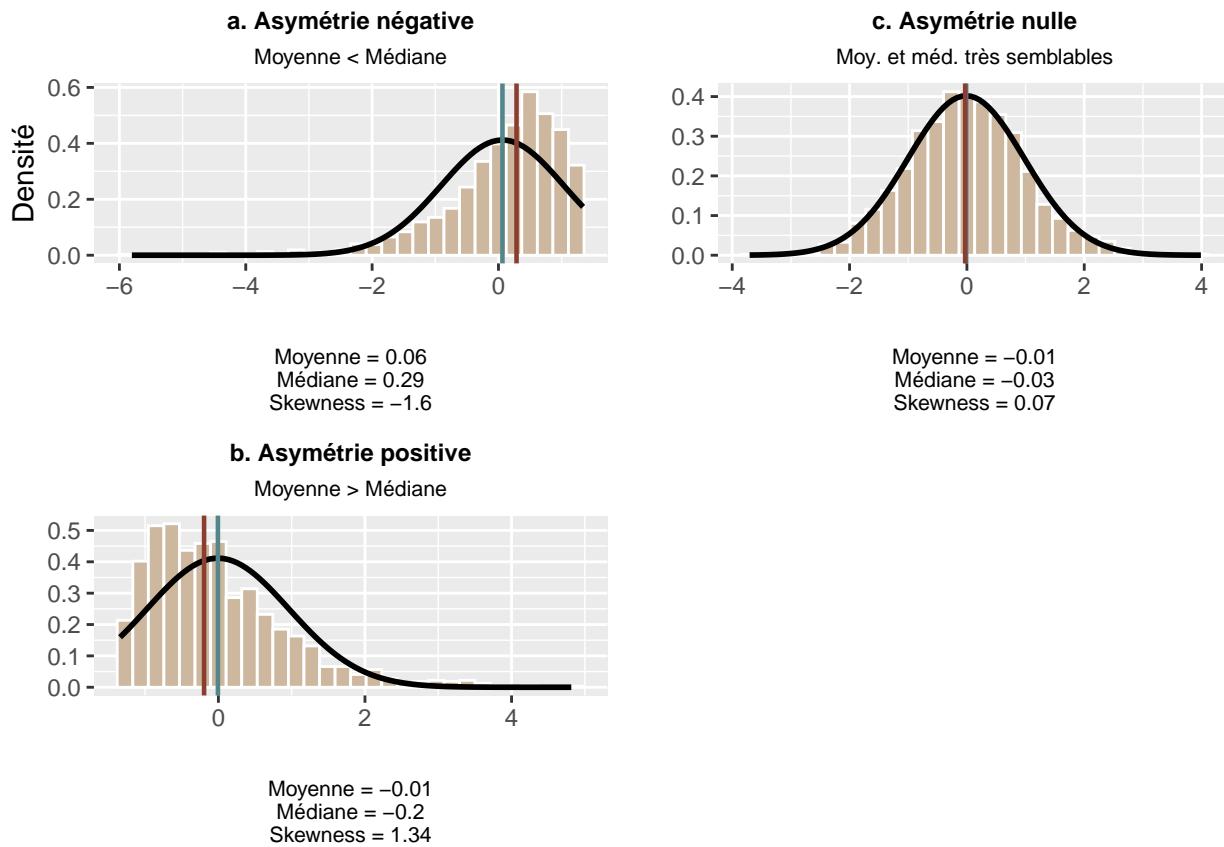


FIG. 2.27 : Asymétrie d'une distribution

valeurs très semblables pour de grands échantillons (?). Cette mesure s'interprète comme suit (figure 2.27) :

- quand la valeur du *kurtosis* est négative, la **distribution est platikurtique**. La distribution est dite plate, c'est-à-dire que la valeur de l'écart-type est importante (comparativement à une distribution normale), signalant une grande dispersion des valeurs de part et d'autre la moyenne.
- quand la valeur du *kurtosis* est égale à 0, la **distribution est mésokurtique**, ce qui est typique d'une distribution normale.
- quand la valeur du *kurtosis* est positive, la **distribution est leptokurtique**, signalant que l'écart-type (la dispersion des valeurs) est plutôt faible. Autrement dit, la dispersion des valeurs autour de la moyenne est faible.

$$g_2 = \frac{\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^4}{\left(\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2 \right)^2} - 3 \quad (2.26)$$

$$G_2 = \frac{n-1}{(n-2)(n-3)} \{(n+1)g_2 + 6\} \quad (2.27)$$

$$b_2 = (g_2 + 3)(1 - 1/n)^2 - 3 \quad (2.28)$$



Regardez attentivement les équations (2.26), (2.27), (2.28); vous remarquez que pour g_2 et b_2 , il y a une soustraction de -3 et une addition $+6$ pour G_2 . On parle alors de *kurtosis normalisé* (*excess kurtosis* en anglais).

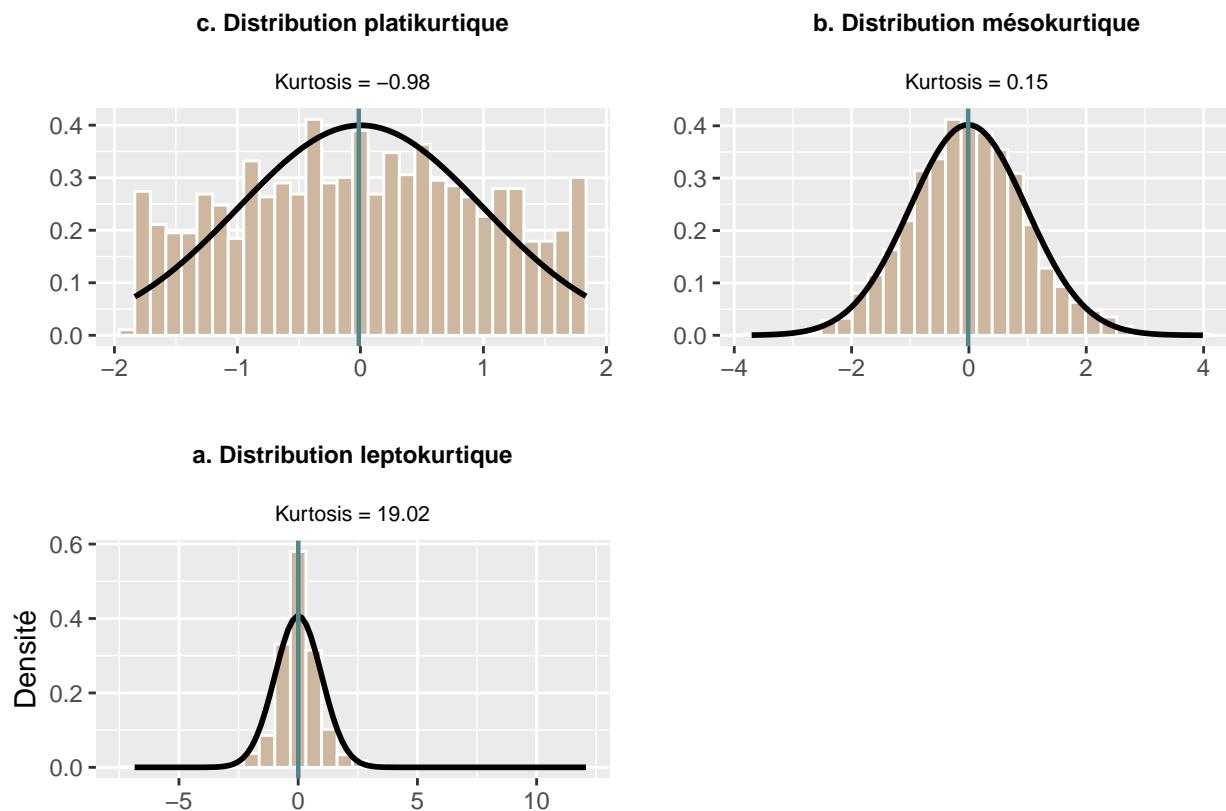


FIG. 2.28 : Applatissement d'une distribution

Pour une distribution normale, il prendra la valeur de 0, comparativement à la valeur de 3 pour un *kurtosis* non normalisé. Par conséquent, avant de calculer du *kurtosis*, il convient de s'assurer que la fonction que vous utilisez implémente une méthode de calcul normalisée (donnant une valeur de 0 pour une distribution normale). Par exemple, la fonction *Kurt* du package **DescTools** calcule les trois formules normalisées tandis que la fonction *kurtosis* du package **moments** renvoie un *kurtosis* non normalisé.

```
library(DescTools)
library(moments)
#Générer une variable normalement distribuée avec 1000 observations
Normale <- rnorm(1500,0,1)
round(DescTools:::Kurt(Normale),3)

## [1] -0.097

round(moments:::kurtosis(Normale),3)

## [1] 2.907
```

2.5.4.1.2 Vérifier la normalité avec des graphiques

Les graphiques sont un excellent moyen de vérifier visuellement si une distribution est normale ou pas. Bien entendu, les histogrammes, que nous avons déjà largement utilisés, sont un incontournable ; à titre

de rappel, ils permettent de représenter la forme de la distribution des données (figure 2.29). Un autre type de graphique intéressant est le **diagramme quantile-quantile** (*Q-Q plot* en anglais) qui permet de comparer la distribution d'une variable avec une distribution gaussienne (normale). Trois éléments composent ce graphique tel qu'illustré à la figure 2.30 :

- les points, représentant les observations de la variable
- la distribution gaussienne (normale), représentée par une ligne
- l'intervalle de confiance à 5% de la distribution normale (en orange sur la figure).

Quand la variable est normale distribuée, les points seront situés le long de la ligne. Plus les points localisés en dehors de l'intervalle de confiance (bande orange) seront nombreux, plus la variable sera alors anormalement distribuée.

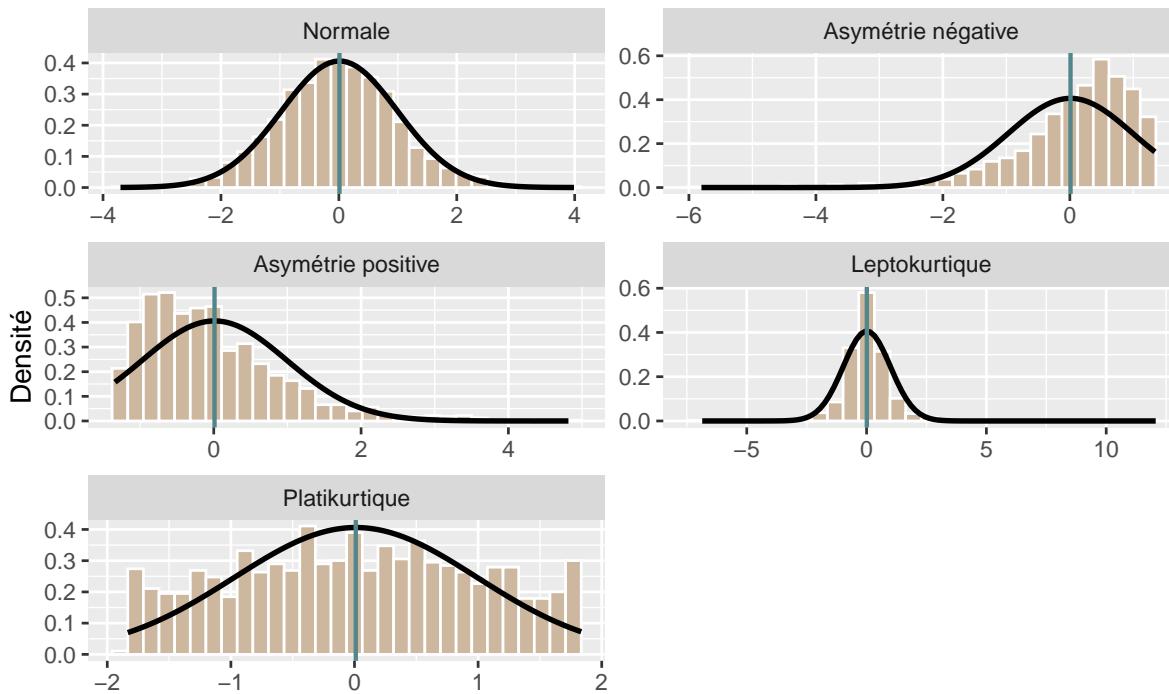


FIG. 2.29 : Distributions et courbe normale

2.5.4.1.3 Vérifier la normalité avec des tests de normalité

Cinq principaux tests d'hypothèse permettent de vérifier la normalité d'une variable : les tests de **Kolmogorov-Smirnov** (KS), **Lilliefors** (LF), **Shapiro-Wilk** (SW), **Anderson-Darling**, et de **Jarque-Bera** (JB); sachez toutefois qu'il y en a d'autres non discutés ici (tests de D'Agostino-Pearson, Cramer-von Mises, de Ryan-Joiner, Shapiro-Francia, etc.). Pour les formules et une description détaillée de ces tests, vous pouvez consulter Razali et al. (?) ou Yap et Sim (?). **Quel test choisir?** Plusieurs auteurs ont comparé ces différents tests à partir de plusieurs échantillons, et ce, en faisant varier la forme de la distribution et le nombre d'observations (??). Selon Razali et al. (?), le meilleur test semble être celui de Shapiro-Wilk, puis ceux de Anderson-Darling, Lilliefors et Kolmogorov-Smirnov. Yap et Sim (?) concluent aussi que le Shapiro-Wilk semble être le plus performant.

Quoi qu'il en soit, ces cinq tests postulent que la variable suit une distribution gaussienne (hypothèse

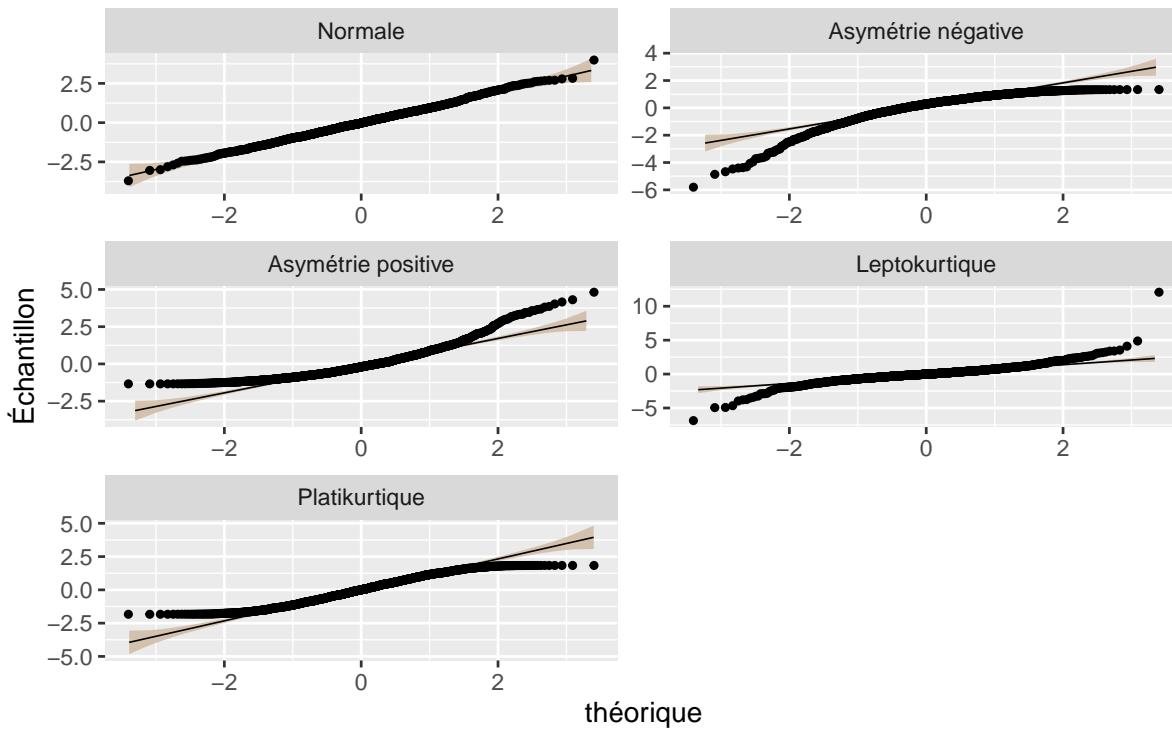


FIG. 2.30 : Diagrammes quantile-quantile

nulle, H_0). Cela signifie que si la valeur de P associée à la valeur de chacun des tests est supérieure au seuil alpha choisi (habituellement $\alpha = 0,05$), la distribution est normale. À l'inverse, si $P < 0,05$, on choisit l'hypothèse alternative (H_1), c'est-à-dire que la distribution est anormale.

Dans le tableau ci-dessous sont reportées les valeurs des différents tests pour les cinq types de distribution générées à la figure 2.29. Sans surprise, pour l'ensemble des tests, la valeur de P est inférieur à 0,05 pour la distribution normale.



Attention! La plupart des auteurs s'entendent sur le fait que ces tests sont très restrictifs : plus la taille de votre échantillon (n) est importante, plus les tests risquent de vous signaler que vos distributions sont anormales (à la lecture des valeurs de P).

Certains conseillent même de ne pas les utiliser quand $n > 200$ et de vous fier uniquement aux graphiques (histogramme et diagramme Q-Q) !



Bref, vérifier la normalité d'une variable n'est pas une tâche si simple. De nouveau, nous vous conseillons vivement de :

- construire les graphiques pour analyser visuellement la forme de la distribution (histogramme avec courbe normale et diagramme Q-Q)
- calculer le *skewness* et le *kurtosis*,
- calculer plusieurs tests (minimamente Shapiro-Wilk et Kolmogorov-Smirnov)
- accorder une importance particulière aux graphiques lorsque vous traitez des grands échantillons ($n > 200$).

TAB. 2.6 : Les différents tests d'hypothèse pour la normalité

Test	Propriétés et interprétation	Fonction R
Kolmogorov-Smirnov	Plus sa valeur est proche de zéro, plus la distribution est normale. L'avantage de ce test est qu'il peut être utilisé pour vérifier si une variable suit la distribution de n'importe quelle loi (autre que la loi normale).	ks.test du package stats
Lilliefors	Ce test est une adaptation du test de Kolmogorov-Smirnov. Plus sa valeur est proche de zéro, plus la distribution est normale.	lillie.test du package nortest
Shapiro-Wilk	Si la valeur de la statistique de Shapiro-Wilk est proche de 1, alors la distribution est normale; et anormale quand elle est inférieure à 1.	shapiro.test du package stats
Anderson-Darling	Ce test est une modification du test de Cramer-von Mises (CVM). Il peut être aussi utilisé pour tester d'autres distributions (uniforme, log-normale, exponentielle, Weibull, distribution de pareto généralisée, logistique, etc.).	ad.test du package stats
Jarque-Bera	Basé sur un test du type multiplicateur de Lagrange, il utilise dans son calcul les valeurs du <i>Skewness</i> et du <i>Kurtosis</i> . Plus sa valeur s'approche de 0, plus la distribution est normale. Ce test est surtout utilisé pour vérifier si les résidus d'un modèle de régression linéaire sont normalement distribués, nous y reviendrons dans le chapitre sur la régression multiple. Il s'écrit $JB = \frac{1}{6} \left(g_1^2 + \frac{g_2^2}{4} \right)$ avec g_1 et g_2 qui sont respectivement les valeurs du <i>skewness</i> et du <i>kurtosis</i> de la variable (voir plus haut les équations 2.23 et 2.26).	JarqueBeraTest du package DescTools

TAB. 2.7 : Calculs des tests de normalité pour différentes distributions

	Normale	Asymétrie négative	Asymétrie positive	Leptokurtique	Platikurtique
Skewness	0.012	1.35	-1.331	-0.571	0.035
Kurtosis	-0.074	2.327	2.988	4.424	-0.966
Kolmogorov-Smirnov (KS)	0.018	0.093	0.086	0.074	0.047
Lilliefors (LF)	0.018	0.093	0.086	0.074	0.047
Shapiro-Wilk (SW)	0.999	0.9	0.91	0.935	0.972
Anderson-Darling (AD)	0.131	11.086	7.854	6.623	2.633
Jarque-Bera (JB)	0.036	348.162	347.491	850.708	13.97
KS (valeur p)	0.998	0	0.001	0.009	0.226
LF (valeur p)	0.966	0	0	0	0.011
SW (valeur p)	0.967	0	0	0	0
AD (valeur p)	0.982	0	0	0	0
JB (valeur p)	0.982	0	0	0	0.001

2.5.4.2 Vérifier d'autres formes de distributions

Comme nous l'avons vu, la distribution normale n'est que l'une des multiples distributions existantes. Dans de nombreuses situations, elle ne sera pas adaptée pour décrire vos variables. La démarche à adopter pour trouver une distribution adaptée est la suivante :

1. Définissez la nature de votre variable, identifier si elle est discrète ou continue et l'intervalle dans lequel elle est définie. Une variable dont les valeurs sont positives ou négatives ne pourra pas être décrite avec une distribution Gamma par exemple (à moins de la décaler).
2. Explorez votre variable, affichez son histogramme et son graphique de densité pour avoir une vue générale de sa morphologie.
3. Présélectionnez un ensemble de distributions candidates compte tenu des observations précédentes. Vous pouvez également vous reporter à la littérature existante sur votre sujet d'étude pour inclure d'autres distributions. Soyez flexible ! Une variable strictement positive pourrait tout de même avoir une forme normale. De même, une variable décrivant des comptages suffisamment grands pourrait être mieux décrite par une distribution normale qu'une distribution de poisson.
4. Tentez d'ajuster chacune des distributions retenues à vos données et comparez les qualités d'ajustements pour retenir la plus adaptée.

Pour ajuster une distribution à un jeu de données, il faut trouver les valeurs des paramètres de cette distribution qui lui permettront d'adopter une forme la plus proche possible des données. On appelle cette opération **ajuster un modèle**, puisque la distribution théorique est utilisée pour modéliser les données. L'ajustement des paramètres est un problème d'optimisation que plusieurs algorithmes sont capables de résoudre (*gradient descent, Newton-Raphson method, Fisher scoring, etc.*). Dans R, le package **fitdistrplus** permet d'ajuster pratiquement n'importe quelle distribution à des données en offrant plusieurs stratégies d'optimisation grâce à la fonction **fitdist**. Il suffit de disposer d'une fonction représentant la distribution de densité ou de masse de la distribution en question, généralement noté **dnomadeladistribution** (**dnorm**, **dgamma**, **dpoisson**, etc.) dans R. Notez que certains *packages* comme **VGAM** ou **gamlss.dist** ajoutent un grand nombre de fonctions de densité et de masse à celles déjà disponibles de base dans R.

Pour comparer l'ajustement de plusieurs distributions théoriques à des données, trois approches doivent être combinées :

- Observer graphiquement l'ajustement de la courbe théorique à l'histogramme des données. Cela permet d'éliminer au premier coup d'œil les distributions qui ne correspondent pas.
- Comparer les *loglikelihood*. Le *loglikelihood* est un score d'ajustement des distributions aux données. Pour faire simple, plus le *loglikelihood* est grand, plus la distribution théorique est proche des données. Référez-vous à l'encadré suivant pour une description plus en profondeur du *loglikelihood*.
- Utiliser le test de Kolmogorov-Smirnov pour déterminer si une distribution particulière est mieux ajustée pour les données.



Qu'est-ce-que le loglikelihood ?

Le *loglikelihood* est une mesure de l'ajustement d'un modèle à des données. Il est utilisé à peu près partout en statistique. Comprendre sa signification est donc un exercice important pour développer une meilleure intuition du fonctionnement général de nombreuses méthodes. Si les concepts de fonction de densité et de fonction de masse vous semblent encore flous, reportez-vous à la section 2.4 sur les distributions dans un premier temps.

Admettons que nous disposons d'une variable continue v que nous avons tenté de modéliser avec une distribution d (il peut s'agir de n'importe quelle distribution). d a une fonction de densité avec laquelle il est possible de calculer pour chacune des valeurs de v sa probabilité d'être observée selon le modèle d .

Prenons un exemple concret dans R. Admettons que nous avons une variable comprenant 10 valeurs (oui,

c'est un petit échantillon, mais c'est pour faire un exemple simple).

```
v <- c(5,8,7,8,10,4,7,6,9,7)
moyenne <- mean(v)
ecart_type <- sd(v)
```

En calculant sa moyenne et son écart type, nous obtenons les paramètres d'une distribution normale que nous pouvons utiliser pour représenter les données observées. En utilisant la fonction `dnorm` (la fonction de densité de la distribution normale), nous pouvons calculer la probabilité d'observer chacune des valeurs de v selon cette distribution normale.

```
probas <- dnorm(v, moyenne, ecart_type)
df <- data.frame(valeur = v,
                  proba = probas)
print(df)

##     valeur      proba
## 1      5 0.11203710
## 2      8 0.19624888
## 3      7 0.22228296
## 4      8 0.19624888
## 5     10 0.06009897
## 6      4 0.04985613
## 7      7 0.22228296
## 8      6 0.18439864
## 9      9 0.12689976
## 10     7 0.22228296
```

On observe ainsi que les valeurs 7 et 8 sont très probables selon le modèle alors que la valeur 10 est très improbable.

Le *likelihood* est simplement le produit de toutes ces probabilités. Il s'agit donc de **la probabilité conjointe** d'avoir observé toutes les valeurs de v **sous l'hypothèse** que d est la distribution produisant ces valeurs. Si d décrit efficacement v , alors le *likelihood* est plus grand que si d ne décrit pas efficacement v . Il s'agit d'une forme de raisonnement par l'absurde : après avoir observé v , on calcule la probabilité d'avoir observé v (*likelihood*) si notre modèle d était vrai. Si cette probabilité est très basse, alors c'est que notre modèle est mauvais puisqu'on a bien observé v .

```
likelihood_norm <- prod(probas)
print(likelihood_norm)
```

```
## [1] 3.322759e-09
```

Cependant, multiplier un grand nombre de valeurs inférieures à zéro tend à produire des chiffres infiniment petits et donc à complexifier grandement le calcul. On préfère donc utiliser le *loglikelihood*. L'idée étant transformer les probabilités obtenues avec la fonction \log puis d'additionner leurs résultats, puisque $\log(xy) = \log(x) + \log(y)$.

```
loglikelihood_norm <- sum(log(probas))
print(loglikelihood_norm)
```

```
## [1] -19.52247
```

Comparons ce *loglikelihood* à celui d'un second modèle dans lequel nous utilisons toujours la distribution normale, mais avec une moyenne différente (faussée en rajoutant +3) :

```
probas2 <- dnorm(v, moyenne+3, ecart_type)
loglikelihood_norm2 <- sum(log(probas2))
print(loglikelihood_norm2)
```

```
## [1] -33.53631
```

Ce second *loglikelihood* est plus faible, indiquant clairement que le premier modèle est plus adapté aux données.

Passons à la pratique avec deux exemples.

2.5.4.2.1 Temps de retard des bus de la ville de Toronto

Analysons les temps de retard pris par les bus de la ville de Toronto lorsqu'un évènement perturbe la circulation. Ce jeu de données est disponible sur le site de l'Open Data²⁵ de la ville de Toronto. Compte tenu de la grande quantité d'observations, nous avons fait le choix de nous concentrer sur les évènements ayant eu lieu durant le mois de janvier 2019. Puisque la variable étudiée est une durée exprimée en minutes, elle est strictement positive (supérieure à 0), car un bus avec zéro minute de retard est à l'heure. Nous considérons également qu'un bus ayant plus de 150 minutes de retard (2h30) n'est tout simplement pas passé (personne ne risque d'attendre 2h30 pour prendre son bus). Commençons par charger les données et observer leur distribution empirique.

```
library(ggplot2)
# charger le jeu de données
data_trt_bus <- read.csv('data/univariee/bus-delay-2019_janv.csv', sep =';')
# retirer les observations aberrantes
data_trt_bus <- subset(data_trt_bus, data_trt_bus$Min.Delay > 0 &
                        data_trt_bus$Min.Delay < 150)
# représenter la distribution empirique du jeu de données
ggplot(data = data_trt_bus) +
  geom_histogram(aes(x=Min.Delay, y = ..density..), bins = 40) +
  geom_density(aes(x=Min.Delay), color = 'blue', bw = 2, size = 0.8) +
  labs(x = 'temps de retard (min)',
       y = '')
```

Compte tenu de la forme de la distribution empirique et de sa nature, quatre distributions sont envisageables :

- La distribution Gamma, strictement positive et asymétrique, elle est aussi une généralisation de la distribution exponentielle utilisée pour modéliser des temps d'attente. Pour des raisons similaires, on peut aussi retenir la distribution de Weibull et la distribution log-normale. Nous écartons ici la distribution skew-normale puisque le jeu de données n'a clairement pas une forme normale au départ.
- La distribution de Pareto, strictement positive et permettant de représenter ici le fait que la plupart des retards durent moins de 10 minutes, mais que quelques retards sont également beaucoup plus longs.

Commençons par ajuster les quatre distributions avec la fonction `fitdist` du package **fitdistrplus** et représentons-les graphiquement pour éliminer les moins bons candidats. Nous utilisons également le package **actuar** pour la fonction de densité de Pareto (`dpareto`).

²⁵ <https://open.toronto.ca/catalogue/?search=bus%20delay&sort=score%20desc>

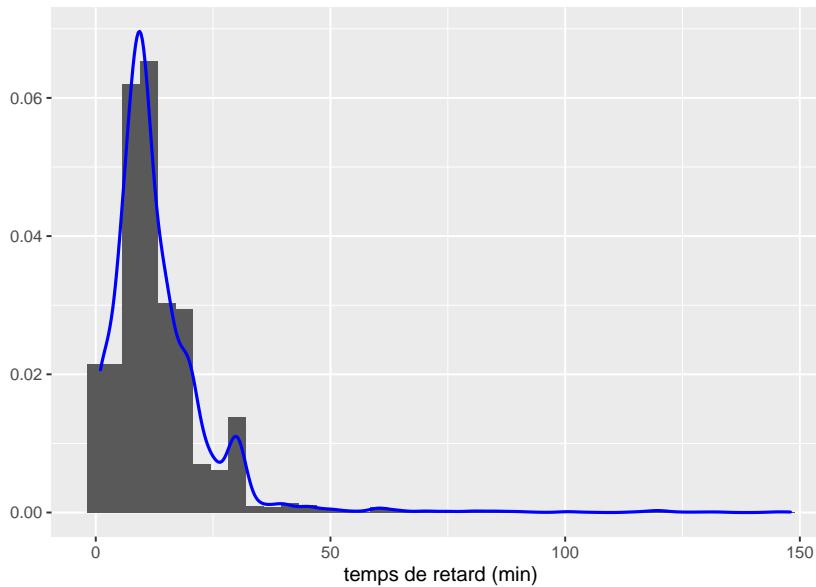


FIG. 2.31 : Distribution empirique des temps de retard des bus à Toronto en janvier 2019

```

library(fitdistrplus)
library(actuar)
library(ggpubr)
# ajustement des modèles
model_gamma <- fitdist(data_trt_bus$Min.Delay, distr = "gamma")
model_weibull <- fitdist(data_trt_bus$Min.Delay, distr = "weibull")
model_lognorm <- fitdist(data_trt_bus$Min.Delay, distr = "lnorm")
model_pareto <- fitdist(data_trt_bus$Min.Delay, distr = "pareto",
                         start = list(shape = 1, scale = 1),
                         method = "mse") # différentes méthodes d'optimisations
# réalisation des graphiques
plot1 <- ggplot(data = data_trt_bus) +
  geom_histogram(aes(x=Min.Delay, y = ..density..), bins = 40) +
  stat_function(fun = dgamma, color = 'red', size = 0.8,
                args = as.list(model_gamma$estimate)) +
  labs(x = 'temps de retard (min)',
       y = '',
       subtitle = "modèle Gamma")
plot2 <- ggplot(data = data_trt_bus) +
  geom_histogram(aes(x=Min.Delay, y = ..density..), bins = 40) +
  stat_function(fun = dweibull, color = 'red', size = 0.8,
                args = as.list(model_weibull$estimate)) +
  labs(x = 'temps de retard (min)',
       y = '',
       subtitle = "modèle Weibull")
plot3 <- ggplot(data = data_trt_bus) +
  geom_histogram(aes(x=Min.Delay, y = ..density..), bins = 40) +
  stat_function(fun = dlnorm, color = 'red', size = 0.8,
                args = as.list(model_lognorm$estimate)) +
  labs(x = 'temps de retard (min)',
       y = '',
       subtitle = "modèle log-normal")

```

```

plot4 <- ggplot(data = data_trt_bus) +
  geom_histogram(aes(x=Min.Delay, y = ..density..), bins = 40) +
  stat_function(fun = dpareto, color = 'red', size = 0.8,
                args = as.list(model_pareto$estimate)) +
  labs(x = 'temps de retard (min)',
       y = '',
       subtitle = "Modèle Pareto")
ggarrange(plotlist = list(plot1, plot2, plot3, plot4),
          ncol = 2, nrow = 2)

```

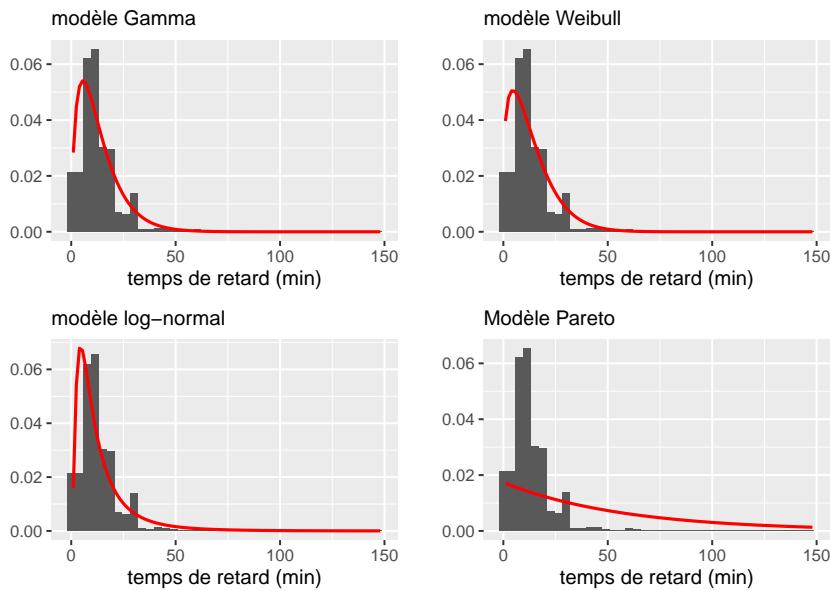


FIG. 2.32 : Comparaison des distributions ajustées aux données de retard des bus

Visuellement, on constate que la distribution de Pareto est un mauvais choix. Pour les trois autres distributions, la comparaison des *loglikelihood* s'impose.

```

df <- data.frame(model = c("Gamma", "Weibull",
                           "log-normal"),
                  loglikelihood = c(model_gamma$loglik,
                                    model_weibull$loglik,
                                    model_lognorm$loglik))

show_table(df,
           col.names = c("Distributon", "LogLikelihood"),
           caption = 'Comparaison des LogLikekelihood des trois distributions',
           )

```

Le plus grand *logLikelihood* est obtenu par la distribution de Gamma qui s'ajuste donc le mieux à nos don-

TAB. 2.8 : Comparaison des LogLikekelihood des trois distributions

Distributon	LogLikelihood
Gamma	-23 062,56
Weibull	-23 195,54
log-normal	-23 375,74

nées. Pour finir, nous pouvons tester formellement avec le test de Kolmogorov-Smirnov si nos données proviennent bien de cette distribution de Gamma.

```
params <- as.list(model_gamma$estimate)
ks.test(data_trt_bus$Min.Delay,
        y = pgamma, shape = params$shape, rate = params$rate)
```

```
## 
## One-sample Kolmogorov-Smirnov test
##
## data: data_trt_bus$Min.Delay
## D = 0.099912, p-value < 2.2e-16
## alternative hypothesis: two-sided
```

La valeur de p est inférieure à 0,05, on ne peut donc pas accepter l'hypothèse que notre jeu de données suit effectivement un loi de Gamma. Considérant le nombre d'observations et le fait que de nombreux temps d'attente sont identiques (ce à quoi le test est très sensible), ce résultat n'est pas surprenant. La distribution de Gamma reste cependant la distribution qui représente le mieux nos données. Nous pouvons estimer grâce à cette distribution la probabilité qu'un bus ait un retard de plus de 10 minutes de la façon suivante :

```
pgamma(10, shape = params$shape, rate = params$rate, lower.tail = F)
```

```
## [1] 0.5409424
```

ce qui correspond à 54% de chance.

Pour moins de 10 minutes :

```
pgamma(10, shape = params$shape, rate = params$rate, lower.tail = T)
```

```
## [1] 0.4590576
```

soit 46%.

Un dernier exemple avec la probabilité qu'un retard dépasse 45 minutes :

```
pgamma(45, shape = params$shape, rate = params$rate, lower.tail = F)
```

```
## [1] 0.01348194
```

Soit seulement 1,3%.

Par conséquent, si un matin à Toronto votre bus a plus de 45 minutes de retard, bravo vous êtes tombé sur une des très rares occasions où un tel retard se produit

2.5.4.2.2 Les accidents de vélo à Montréal

Le second jeu de données représente le nombre d'accidents de la route impliquant un vélo sur les intersections dans les quartiers centraux de Montréal. Le jeu de données complet est disponible sur le site des données ouvertes²⁶ de la ville de Montréal. Puisque ces données correspondent à des comptages, la première distribution à envisager est la distribution de poisson. Cependant, puisque nous aurons également

²⁶ <http://donnees.ville.montreal.qc.ca/dataset/collisions-routieres>

un grand nombre d'intersections sans accident, il serait judicieux de tester la distribution de poisson avec excès de zéro.

```
library(ggplot2)
# charger le jeu de données
data_accidents <- read.csv('data/univariee/accidents_mtl.csv', sep = ',', )
counts <- data.frame(table(data_accidents$nb_accident))
names(counts) <- c("nb_accident", "fréquence")
counts$nb_accident <- as.numeric(as.character(counts$nb_accident))
counts$prop <- counts$fréquence / sum(counts$fréquence)
# représenter la distribution empirique du jeu de donnée
ggplot(data = counts) +
  geom_bar(aes(x=nb_accident, weight = fréquence), width = 0.5) +
  labs(x = "nombre d'accidents",
       y = 'fréquence')
```

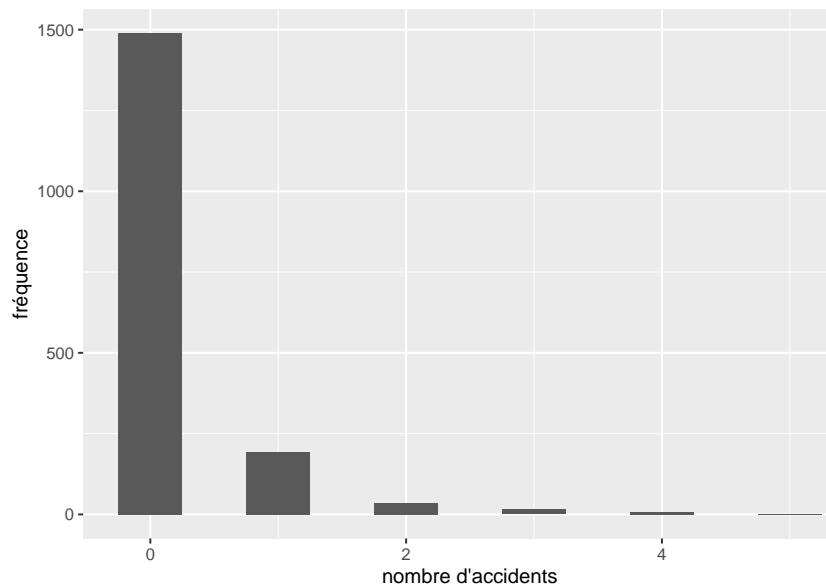


FIG. 2.33 : Distribution empirique du nombre d'accidents par intersection impliquant un cycliste à Montréal en 2017 dans les quartiers centraux

Nous avons effectivement de nombreux zéros ici, essayons d'ajuster nos deux distributions à ce jeu de données. Dans le graphique suivant, les barres grises représentent la distribution empirique du jeu de données et les barres rouges les distributions théoriques ajustées. Nous utilisons ici le package **gamlss.dist** pour avoir la fonction de masse d'une distribution de poisson avec excès de zéros.

```
library(gamlss.dist)
#ajuster le modèle de poisson
model_poisson <- fitdist(data_accidents$nb_accident, distr = "pois")
#ajuster le modèle de poisson avec excès de zéros
model_poissonzi <- fitdist(data_accidents$nb_accident, "ZIP",
  start = list(mu = 4, sigma = 0.15), # valeurs pour faciliter la convergence
  optim.method = "L-BFGS-B", # méthode d'optimisation recommandée dans la doc
  lower = c(0.00001, 0.00001),# valeurs minimales des deux paramètres
  upper = c(Inf, 1)# valeurs maximales des deux paramètres
  )
```

```

dfpoisson <- data.frame(x=c(0:10),
                         y=dpois(0:10, model_poisson$estimate)
                         )
plot1 <- ggplot() +
  geom_bar(aes(x=nb_accident, weight = prop), width = 0.6, data = counts) +
  geom_bar(aes(x=x, weight = y), width = 0.15, data = dfpoisson, fill = "red") +
  scale_x_continuous(limits = c(-0.5,7), breaks = c(0:7)) +
  labs(subtitle = "modèle poisson",
       x = "nombre d'accidents",
       y = "")
dfpoissonzi <- data.frame(x=c(0:10),
                           y=dZIP(0:10, model_poissonzi$estimate[[1]],
                                   model_poissonzi$estimate[[2]])
                           )
plot2 <- ggplot() +
  geom_bar(aes(x=nb_accident, weight = prop), width = 0.6, data = counts) +
  geom_bar(aes(x=x, weight = y), width = 0.15, data = dfpoissonzi, fill = "red") +
  scale_x_continuous(limits = c(-0.5,7), breaks = c(0:7)) +
  labs(subtitle = "modèle poisson avec excès de zéro",
       x = "nombre d'accident",
       y = "")
ggarrange(plotlist = list(plot1,plot2), ncol = 2)

```

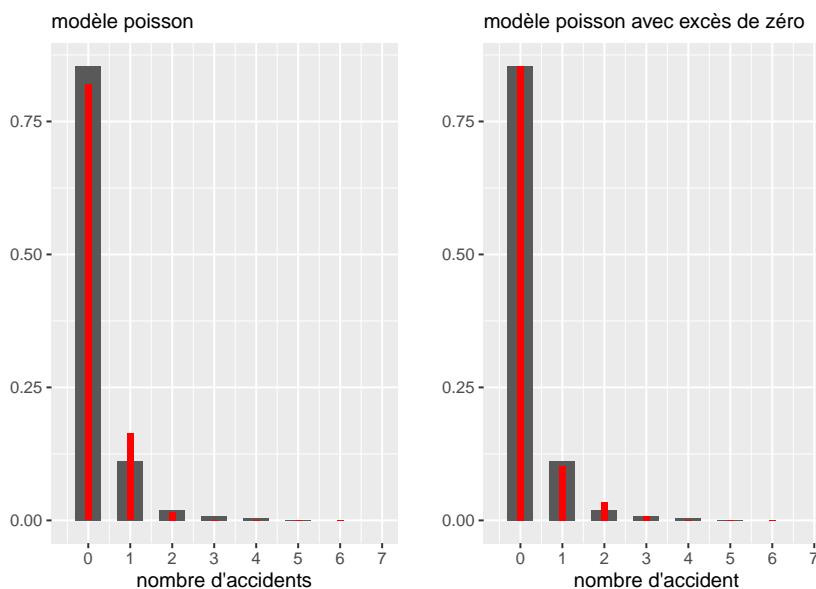


FIG. 2.34 : Ajustement des distributions de poisson et poisson avec excès de zéros

Visuellement, le modèle avec excès de zéro semble s'imposer. Nous pouvons vérifier cette impression avec la comparaison des *loglikelihood*.

```
print(model_poisson$loglik)
```

```
## [1] -989.83
```

```

print(model_poissonzi$loglik)

## [1] -931.8778

#afficher les paramètres ajustés
model_poissonzi$estimate

##          mu      sigma
## 0.6690301 0.7022605

```

Nous avons donc la confirmation que le modèle de poisson avec excès de zéros est mieux ajusté. Nous apprenons donc que 70% ($\sigma = 0,70$) des intersections sont en fait exclues du phénomène étudié (probablement parce que très peu de cyclistes les utilisent ou parce qu'elles sont très peu accidentogènes) et que pour les autres, le taux d'accidents par année en 2017 était de 0,67 ($\mu = 0,669$, μ signifiant λ pour le package `gamlss`). À nouveau, nous pouvons effectuer un test formel avec la fonction `ks.test`.

```

params <- as.list(model_poissonzi$estimate)
ks.test(data_accidents$nb_accident,
        y = pZIP, mu = params$mu, sigma = params$sigma)

##
## One-sample Kolmogorov-Smirnov test
##
## data: data_accidents$nb_accident
## D = 0.85476, p-value < 2.2e-16
## alternative hypothesis: two-sided

```

Encore une fois, on doit rejeter l'hypothèse selon laquelle le test suit une distribution de poisson avec excès de zéros. Ces deux exemples montrent à quel point ce test est restrictif.

2.5.5 La transformation des variables

2.5.5.1 Les transformations visant à atteindre la normalité

Comme énoncé au début de cette section, plusieurs méthodes statistiques nécessitent que la variable quantitative soit normalement distribuée. C'est notamment le cas de l'analyse de variance et des tests t (abordés dans les chapitres suivants) qui fourniront des résultats plus robustes lorsque la variable est normalement distribuée. Plusieurs transformations sont possibles, les plus courantes étant la racine carrée, le logarithme et l'inverse de la variable. Selon plusieurs auteurs (notamment, Tabacknick et *et al.* (?), p. 89)), en fonction du type (positive ou négative) et du degré d'asymétrie, les transformations suivantes sont possibles afin d'améliorer la normalité de la variable :

- Asymétrie positive modérée : la racine carrée de la variable X avec la fonction `sqrt(df$x)`.
- Asymétrie positive importante : le logarithme de la variable avec `log10(df$x)`
- Asymétrie positive sévère : l'inverse de la variable avec `1/(df$x)`



Attention, pour une valeur égale ou inférieure à 0, on ne peut pas calculer une racine carrée ou un logarithme. Par conséquent, il convient de décaler simplement la distribution vers la droite afin de s'assurer qu'il n'y ait plus de valeurs négative ou égale à 0 :

– `sqrt(df$x - min(df$x+1))` avec pour une asymétrie positive avec des valeurs négatives ou égales à 0

- $\log(df$x - \min(df$x+1))$ pour une asymétrie positive avec des valeurs négatives ou égales à 0

Par exemple, si la valeur minimale de la variable est égale à -10, la valeur minimale de variable décalée sera ainsi de 11.

- Asymétrie négative modérée : $\sqrt{\max(df$x+1) - df$x}$.
- Asymétrie négative importante : $\log(\max(df$x+1) - df$x)$
- Asymétrie négative sévère : $1/(\max(df$x+1) - df$x)$

Transformation des variables pour atteindre la normalité : ce n'est pas toujours la panacée !

La transformation des données fait et fera encore longtemps débat à la fois parmi les statisticiens, les débutants et utilisateurs avancés des méthodes quantitatives. Field et al. (? , pp. 193) résument le tout avec humour : « To transform or not transform, that is the question ».

Avantages de la transformation

- L'obtention de *résultats plus robustes*.
- Dans une régression linéaire multiple, la transformation de la variable dépendante peut *remédier au non-respect des hypothèses de base liées à la régression* (linéarité et homoscédasticité des erreurs, absence des valeurs aberrantes, etc.).

Inconvénients de la transformation

- *Une variable transformée est plus difficile à interpréter* puisque cela change l'unité de mesure de la variable. Prenons un exemple concret : vous souhaitez comparer les moyennes de revenu de deux groupes A et B. Vous obtenez une différence de 15000\$, soit une valeur facile à interpréter. Par contre, si la variable a été préalablement transformée en logarithme, il est possible que vous obteniez une différence de 9, ce qui est beaucoup moins parlant. Aussi, en transformant la variable en *log*, vous ne comparez plus les moyennes arithmétiques des deux groupes, mais plutôt leurs moyennes géométriques (? , pp. 193).
- *Pourquoi perdre la forme initiale de la distribution du phénomène à expliquer ?* Il est possible pour de nombreuses méthodes de choisir la distribution que l'on souhaite utiliser, il n'est donc pas nécessaire de toujours se limiter à la distribution normale. Par exemple, dans les modèles de régression généralisés (GLM), on pourrait indiquer que notre variable indépendante suit une distribution de *Student* plutôt que de vouloir à tout prix la rendre normale. De même, certains tests non-paramétriques permettent d'analyser des variables ne suivant pas une distribution normale.

Démarche à suivre avant et après la transformation

- *La transformation est-elle nécessaire ?* Ne transformez jamais une variable sans avoir analyser rigoureusement sa forme (histogramme avec courbe normale, *skewness* et *kurtosis*, tests de normalité).
- *D'autres options à la transformation d'une variable dépendante (VD) sont-elles envisageables ?* Identifiez la forme de la distribution de la VD et utilisez au besoin un modèle GLM adapté à cette distribution. Autrement dit, ne transformez pas automatiquement votre VD pour simplement pouvoir l'introduire dans une régression linéaire multiple.
- *La transformation a-t-elle un apport significatif ?* Premièrement, vérifiez si la transformation utilisée (logarithme, racine carrée, inverse, etc.) améliore la normalité de la variable. Ce n'est toujours le cas, pourquoi c'est pire ! Prenez soin de comparer les histogrammes, les valeurs de *skewness*, *kurtosis* et des différents tests de normalité avant et après la transformation. Deuxièmement, comparez les résultats de vos analyses statistiques sans et avec transformation, et ce, dans une démarche coût-avantage. Vos résultats sont-ils bien plus robustes ? Par exemple, un R^2 qui passe de 0,597 à 0,602 avant et après la transformation des variables avec des associations significatives similaires, mais plus difficiles à interpréter (du fait des transformations), n'est pas forcément un gain significatif. La modélisation en sciences sociales ne vise pas à prédire la trajectoire d'un satellite ou l'atterrissement d'un engin sur Mars ! La précision à la quatrième décimale n'est pas une condition ! Par conséquent, un modèle un peu moins robuste, mais plus facile à interpréter est parfois préférable.

2.5.5.2 Autres types de transformations

Les trois transformations les plus couramment utilisées sont :

- **La côte z** (*z score* en anglais) qui consiste à soustraire à chaque valeur sa moyenne (soit un centrage), puis à la diviser par son écart-type (soit une réduction) (eq. (2.29)). Par conséquent, on parle aussi de variable centrée-réduite qui a comme propriétés intéressantes une moyenne égale à 0 et un écart-type égale à 1 (la variance est aussi égale à 1 puisque $1^2 = 1$). Nous verrons que cette transformation est largement utilisée dans les méthodes de classification (chapitre ??) et les méthodes factorielles (chapitre ??).

$$z = \frac{x_i - \mu}{\sigma} \quad (2.29)$$

- **La transformation en rangs** qui consiste simplement à trier une variable en ordre croissant, puis à affecter le rang de chaque observation de 1 à n . Cette transformation est très utilisée quand la variable est très anormalement distribuée, notamment pour calculer le coefficient de corrélation de Spearman (section ??) et certains tests non-paramétriques (sections ?? et ??).
- **La transformation sur une échelle de 0 à 1** (ou de 0 à 100) qui consiste à soustraite à chaque observation la valeur minimale et à diviser le tout par l'étendue (eq. (2.30)).

$$X_{\in[0-1]} = \frac{x_i - \max}{\max - \min} \text{ ou } X_{\in[0-100]} = \frac{x_i - \min}{\max - \min} \times 100 \quad (2.30)$$

Pour un *dataframe* nommé *df* comprenant une variable *x*, la syntaxe ci-dessous illustre comment obtenir quatre transformations (côte z , rangs, 0 à 1 et 0 à 100).

```
df2 <- data.frame(x = c(22, 27, 25, 30, 37, 32, 35, 40))

# Transformation centrée-réduite : côte Z
df2$zx <- (df2$x - mean(df2$x)) / sd(df2$x)

# Transformation en rangs avec la fonction rank
df2$rz <- rank(df2$x)

# Transformation en rangs de 0 à 1
```

TAB. 2.9 : Illustration des trois transformations

Observation	x_i	Côte z	Rang	0 à 1
1	22,00	-1,45	1	0,00
2	27,00	-0,65	3	0,28
3	25,00	-0,97	2	0,17
4	30,00	-0,16	4	0,44
5	37,00	0,97	7	0,83
6	32,00	0,16	5	0,56
7	35,00	0,65	6	0,72
8	40,00	1,45	8	1,00
Moyenne	31,00	0,00		
Écart-type	6,19	1,00		

```
df2$x01 <- (df2$x-min(df2$x))/(max(df2$x)-min(df2$x))

# Transformation en rangs de 0 à 100
df2$x0100 <- (df2$x-min(df2$x))/(max(df2$x)-min(df2$x))*100
```



Ces trois transformations sont parfois utilisées pour générer un indice composite à partir de plusieurs variables ou encore dans une analyse de sensibilité avec les indices de Sobol (?).

2.5.6 Mise en œuvre dans R

Il existe une multitude de *packages* dédiés au calcul des statistiques descriptives univariées. Par parcimonie, nous en utiliserons uniquement trois : `DescTools`, `nortest` et `stats`. Libre à vous de faire vos recherches sur Internet pour utiliser d'autres *packages* au besoin. Les principales fonctions que nous utilisons ici sont :

- `summary` : pour obtenir un résumé sommaire des statistiques descriptives (minimum, Q1, Q2 Q3, Maximum)
- `mean` : moyenne
- `min` : minimum
- `max` : maximum
- `range` : minimum et maximum
- `quantile` : quartiles
- `quantile((x, probs = seq(.0, 1, by = .2))` : quintiles
- `quantile((x, probs = seq(.0, 1, by = .1))` : déciles
- `var` : variance
- `sd` : écart-type
- Skew du *package DescTools* : coefficient d'asymétrie
- Kurt du *package DescTools* : coefficient d'aplatissement
- `ks.test(x, "pnorm", mean=mean(x), sd=sd(x))` du *package nortest* : test de Kolmogorov-Smirnov
- `shapiro.test` du *package DescTools* : test de Shapiro-Wilk
- `lillie.test` du *package DescTools* : du package `nortest` : test de Lilliefors
- `ad.test` du *package DescTools* : test d'Anderson-Darling
- `JarqueBeraTest` du *package DescTools* : test de Jarque-Bera

2.5.6.1 Application à une seule variable

Admettons que vous voulez obtenir des statistiques pour une seule variable présente dans un *dataframe* (`dataMTL$PctFRev`) :

```
library(DescTools)
library(stats)
library(nortest)

# Importation du fichier csv dans un dataframe
dataMTL <- read.csv("data/univariee/DataSR2016.csv")
# Tableau sommaire pour la variable PctFRev
summary(dataMTL$PctFRev)
```

```
##      Min. 1st Qu. Median Mean 3rd Qu. Max.
## 1.846 11.242 15.471 16.822 20.229 68.927
```

```

# PARAMÈTRES DE TENDANCE CENTRALE
mean(dataMTL$PctFRev)    # Moyenne

## [1] 16.82247

median(dataMTL$PctFRev)   # Médiane

## [1] 15.471

# PARAMÈTRES DE POSITION
# Quartiles
quantile(dataMTL$PctFRev)

##      0%     25%     50%     75%    100%
## 1.8460 11.2420 15.4710 20.2285 68.9270

# Quintiles
quantile(dataMTL$PctFRev, probs = seq(.0, 1, by = .2))

##      0%     20%     40%     60%     80%    100%
## 1.846 10.294 13.626 16.918 21.756 68.927

# Déciles
quantile(dataMTL$PctFRev, probs = seq(.0, 1, by = .1))

##      0%     10%     20%     30%     40%     50%     60%     70%     80%     90%    100%
## 1.846  8.402 10.294 12.172 13.626 15.471 16.918 18.868 21.756 26.854 68.927

# Percentiles personnalisés avec apply
quantile(dataMTL$PctFRev, probs = c(0.01,.05,0.10,.25,.50,.75,.90,.95,.99))

##      1%      5%     10%     25%     50%     75%     90%     95%     99%
## 5.2290  7.1470  8.4020 11.2420 15.4710 20.2285 26.8540 31.7530 45.6010

# PARAMÈTRES DE DISPERSION
range(dataMTL$PctFRev)  # Min et Max

## [1] 1.846 68.927

# Étendue
max(dataMTL$PctFRev)-min(dataMTL$PctFRev)

## [1] 67.081

# Écart interquartile
quantile(dataMTL$PctFRev)[4]-quantile(dataMTL$PctFRev)[2]

```

```

##      75%
## 8.9865

var(dataMTL$PctFRev) # Variance

## [1] 66.62482

sd(dataMTL$PctFRev) # Écart-type

## [1] 8.162403

sd(dataMTL$PctFRev) / mean(dataMTL$PctFRev) # CV

## [1] 0.4852083

# PARAMÈTRES DE FORME
Skew(dataMTL$PctFRev) # Skewness

## [1] 1.67367

Kurt(dataMTL$PctFRev) # Kurtosis

## [1] 4.858815

# TESTS D'HYPOTHÈSE SUR LA NORMALITÉ
# K-Smirnov
ks.test(dataMTL$PctFRev, "pnorm", mean=mean(dataMTL$PctFRev), sd=sd(dataMTL$PctFRev))

## 
## One-sample Kolmogorov-Smirnov test
##
## data: dataMTL$PctFRev
## D = 0.10487, p-value = 1.646e-09
## alternative hypothesis: two-sided

shapiro.test(dataMTL$PctFRev)

##
## Shapiro-Wilk normality test
##
## data: dataMTL$PctFRev
## W = 0.88748, p-value < 2.2e-16

lillie.test(dataMTL$PctFRev)

##
## Lilliefors (Kolmogorov-Smirnov) normality test
##

```

```
## data: dataMTL$PctFRev
## D = 0.10487, p-value < 2.2e-16
```

```
ad.test(dataMTL$PctFRev)
```

```
##
## Anderson-Darling normality test
##
## data: dataMTL$PctFRev
## A = 21.072, p-value < 2.2e-16
```

```
JarqueBeraTest(dataMTL$PctFRev)
```

```
##
## Robust Jarque Bera Test
##
## data: dataMTL$PctFRev
## X-squared = 2173.1, df = 2, p-value < 2.2e-16
```

Pour construire un histogramme avec la courbe normale, vous pourrez consulter la section ?? ou la syntaxe ci-dessous.

```
moyenne <- mean(dataMTL$PctFRev)
ecart_type <- sd(dataMTL$PctFRev)

ggplot(data = dataMTL) +
  geom_histogram(aes(x = PctFRev, y = ..density..),
                 bins = 30, color = "#343a40", fill = "#a8dadc") +
  labs(y = "densité") +
  stat_function(fun = dnorm, args = list(mean = moyenne, sd = ecart_type),
                color = "#e63946", size = 1.2, linetype = "dashed")
```

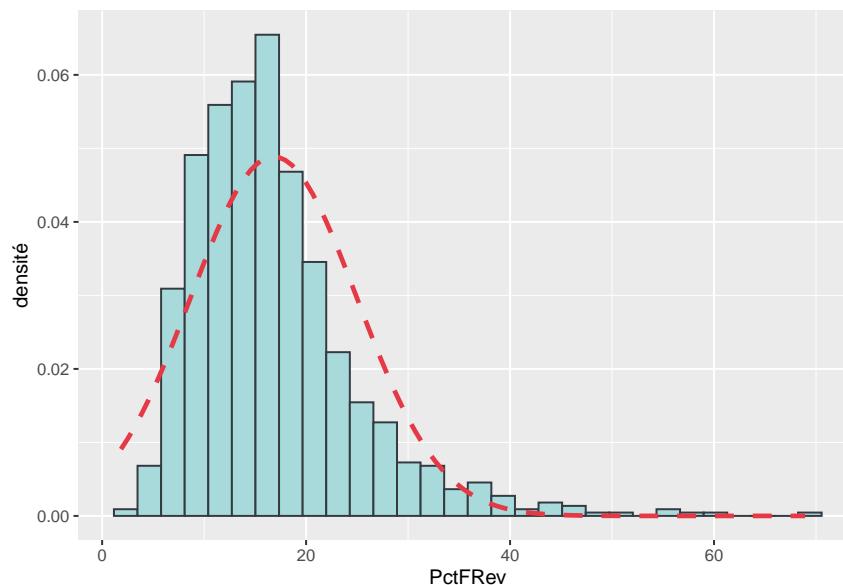


Fig. 2.35 : Histogramme avec courbe normale

2.5.6.2 Application à plusieurs variables

Pour obtenir des sorties de statistiques descriptives pour plusieurs variables, nous vous conseillons :

- de créer un vecteur avec les noms de variables (*VarsSelect* dans la syntaxe ci-dessous)
- d'utiliser ensuite les fonctions *sapply* et *apply*.

```
# Noms des variables du dataframe
names(dataMTL)

## [1] "CTNAME"          "PopTotal"        "HabKm2"
## [4] "PctFRev"         "TxChomage"       "PctImmigrant"
## [7] "PctImgRecent"    "PctMenage1pers"  "PctFamilleMono"
## [10] "PctLangueMaternelleFR" "PctLangueMaternelleAN" "PctLangueMaternelleAU"

# Vecteur pour trois variables
VarsSelect <- c("HabKm2", "TxChomage", "PctFRev" )

# Tableau sommaire pour les 3 variables
summary(dataMTL[VarsSelect])

##      HabKm2      TxChomage      PctFRev
##  Min.   : 18   Min.   : 1.942   Min.   : 1.846
##  1st Qu.: 1980  1st Qu.: 5.482   1st Qu.:11.242
##  Median : 3773  Median : 7.130   Median :15.471
##  Mean   : 5513  Mean   : 7.743   Mean   :16.822
##  3rd Qu.: 7916  3rd Qu.: 9.391   3rd Qu.:20.229
##  Max.   :50282  Max.   :26.882   Max.   :68.927

# PARAMÈTRES DE TENDANCE CENTRALE
sapply(dataMTL[VarsSelect], mean) # Moyenne

##      HabKm2      TxChomage      PctFRev
##  5512.830705  7.743329  16.822470

sapply(dataMTL[VarsSelect], median) # Médiane

##      HabKm2      TxChomage      PctFRev
##  3773.000     7.130      15.471

# PARAMÈTRES DE POSITION
# Quartiles
sapply(dataMTL[VarsSelect], quantile)

##      0%      25%      50%      75%     100%
##  18.0    1980.5   3773.0   7915.5  50282.0
##          1.9420  5.4825  7.1300  9.3910 26.8820
##          11.2420 15.4710 20.2285 68.9270
```

```
# Quintiles
apply(dataMTL[VarsSelect], 2, function(x) quantile(x, probs = seq(.0, 1, by = .2)))
```

```
##      HabKm2 TxChomage PctFRev
## 0%       18     1.942   1.846
## 20%     1525     5.116 10.294
## 40%     2953     6.422 13.626
## 60%     4971     7.973 16.918
## 80%     9509    10.000 21.756
## 100%    50282    26.882 68.927
```

```
# Déciles
apply(dataMTL[VarsSelect], 2, function(x) quantile(x, probs = seq(.0, 1, by = .1)))
```

```
##      HabKm2 TxChomage PctFRev
## 0%       18     1.942   1.846
## 10%      455     4.369   8.402
## 20%     1525     5.116 10.294
## 30%     2298     5.780 12.172
## 40%     2953     6.422 13.626
## 50%     3773     7.130 15.471
## 60%     4971     7.973 16.918
## 70%     6918     8.909 18.868
## 80%     9509    10.000 21.756
## 90%    13055    11.749 26.854
## 100%    50282    26.882 68.927
```

```
# Percentiles personnalisés avec apply
apply(dataMTL[VarsSelect], 2,
      function(x) quantile(x, probs = c(0.01,.05,0.10,.25,.50,.75,.90,.95,.99)))
```

```
##      HabKm2 TxChomage PctFRev
## 1%       58.5    2.9665  5.2290
## 5%      178.0    3.8980  7.1470
## 10%     455.0    4.3690  8.4020
## 25%    1980.5    5.4825 11.2420
## 50%    3773.0    7.1300 15.4710
## 75%    7915.5    9.3910 20.2285
## 90%   13055.0   11.7490 26.8540
## 95%   15355.0   13.8400 31.7530
## 99%  18578.5   17.1920 45.6010
```

```
# PARAMÈTRES DE DISPERSION
sapply(dataMTL[VarsSelect], range) # Min et Max
```

```
##      HabKm2 TxChomage PctFRev
## [1,]     18     1.942   1.846
## [2,]    50282    26.882 68.927
```

```

# Étendue
sapply(dataMTL[VarsSelect], max) - sapply(dataMTL[VarsSelect], min)

##      HabKm2 TxChomage   PctFRev
## 50264.000    24.940     67.081

# Écart interquartile
sapply(dataMTL[VarsSelect], quantile)[4,] - sapply(dataMTL[VarsSelect], quantile)[2,]

##      HabKm2 TxChomage   PctFRev
## 5935.0000    3.9085     8.9865

sapply(dataMTL[VarsSelect], var)      # Variance

##      HabKm2      TxChomage       PctFRev
## 2.633462e+07 9.880932e+00 6.662482e+01

sapply(dataMTL[VarsSelect], sd)      # Écart-type

##      HabKm2      TxChomage       PctFRev
## 5131.726785    3.143395     8.162403

# Coefficient de variation
sapply(dataMTL[VarsSelect], sd) / sapply(dataMTL[VarsSelect], mean)

##      HabKm2 TxChomage   PctFRev
## 0.9308696  0.4059488  0.4852083

# PARAMÈTRES DE FORME
sapply(dataMTL[VarsSelect], Skew)    # Skewness

##      HabKm2 TxChomage   PctFRev
## 1.967468  1.280216   1.673670

sapply(dataMTL[VarsSelect], Kurt)    # Kurtosis

##      HabKm2 TxChomage   PctFRev
## 8.546403  2.892443   4.858815

# TESTS D'HYPOTHÈSE POUR LA NORMALITÉ
# K-Smirnov
apply(dataMTL[VarsSelect], 2, function(x) ks.test(x, "pnorm", mean=mean(x), sd=sd(x)))

## $HabKm2
##
## One-sample Kolmogorov-Smirnov test
##

```

```

## data:  x
## D = 0.14899, p-value < 2.2e-16
## alternative hypothesis: two-sided
##
## 
## $TxChomage
##
## One-sample Kolmogorov-Smirnov test
##
## data:  x
## D = 0.080183, p-value = 9.778e-06
## alternative hypothesis: two-sided
##
## 
## $PctFRev
##
## One-sample Kolmogorov-Smirnov test
##
## data:  x
## D = 0.10487, p-value = 1.646e-09
## alternative hypothesis: two-sided

```

```
sapply(dataMTL[VarsSelect], shapiro.test)      # Shapiro-Wilk
```

```

##          HabKm2                  TxChomage
## statistic 0.8385086            0.9235146
## p.value   5.648795e-30         1.451222e-21
## method    "Shapiro-Wilk normality test" "Shapiro-Wilk normality test"
## data.name "X[[i]]"                "X[[i]]"
##          PctFRev
## statistic 0.8874803
## p.value   1.00278e-25
## method    "Shapiro-Wilk normality test"
## data.name "X[[i]]"

```

```
sapply(dataMTL[VarsSelect], lillie.test)      # Lilliefors
```

```

##          HabKm2
## statistic 0.148988
## p.value   5.689619e-58
## method    "Lilliefors (Kolmogorov-Smirnov) normality test"
## data.name "X[[i]]"
##          TxChomage
## statistic 0.0801829
## p.value   7.758887e-16
## method    "Lilliefors (Kolmogorov-Smirnov) normality test"
## data.name "X[[i]]"
##          PctFRev
## statistic 0.1048704

```

```

## p.value    7.43257e-28
## method     "Lilliefors (Kolmogorov-Smirnov) normality test"
## data.name  "X[[i]]"

sapply(dataMTL[VarsSelect], ad.test)          # Anderson-Darling

##             HabKm2                  TxChomage
## statistic 36.40276                 14.9237
## p.value   3.7e-24                  3.7e-24
## method    "Anderson-Darling normality test" "Anderson-Darling normality test"
## data.name "X[[i]]"                  "X[[i]]"
##             PctFRev
## statistic 21.07194
## p.value   3.7e-24
## method    "Anderson-Darling normality test"
## data.name "X[[i]]"

sapply(dataMTL[VarsSelect], JarqueBeraTest)    # Jarque-Bera

##             HabKm2                  TxChomage
## statistic 4270.113                639.2741
## parameter 2                      2
## p.value   0                      0
## method    "Robust Jarque Bera Test" "Robust Jarque Bera Test"
## data.name "X[[i]]"                  "X[[i]]"
##             PctFRev
## statistic 2173.082
## parameter 2
## p.value   0
## method    "Robust Jarque Bera Test"
## data.name "X[[i]]"

```

2.5.6.3 Transformer une variable dans R

La syntaxe ci-dessous illustre trois exemples de transformation (logarithme, racine carrée et inverse de la variable). Rappelez-vous qu'il faut comparer les valeurs de forme (*skewness* et *kurtosis*) et de forme (tests de Shapiro-Wilk) avant et après les transformations pour identifier celle qui est la plus efficace.

```

library(ggpubr)

# Importation du fichier csv dans un dataframe
dataMTL <- read.csv("data/univariee/DataSR2016.csv")

# Noms des variables du dataframe
names(dataMTL)

## [1] "CTNAME"           "PopTotal"          "HabKm2"
## [4] "PctFRev"          "TxChomage"         "PctImmigrant"
## [7] "PctImgRecent"     "PctMenage1pers"    "PctFamilleMono"
## [10] "PctLangueMaternelleFR" "PctLangueMaternelleAN" "PctLangueMaternelleAU"

```

```

# Transformations
dataMTL$HabKm2_log <- log10(dataMTL$HabKm2)
dataMTL$HabKm2_sqrt <- sqrt(dataMTL$HabKm2)
dataMTL$HabKm2_inv <- 1/dataMTL$HabKm2

# Vecteur pour la variable et les trois transformations
VarsSelect <- c("HabKm2", "HabKm2_log", "HabKm2_sqrt", "HabKm2_inv")

# paramètres de forme
sapply(dataMTL[VarsSelect], Skew)      # Skewness

##      HabKm2   HabKm2_log   HabKm2_sqrt   HabKm2_inv
## 1.9674683 -1.2071326    0.4179037    8.2536901

sapply(dataMTL[VarsSelect], Kurt)       # Kurtosis

##      HabKm2   HabKm2_log   HabKm2_sqrt   HabKm2_inv
## 8.54640302 1.55670769  0.04563433 82.85604898

# TESTS D'HYPOTHÈSE SUR LA NORMALITÉ
sapply(dataMTL[VarsSelect], shapiro.test)

##          statistic        p.value      method      data.name
## 1 0.8385086 5.648795e-30 Shapiro-Wilk normality test
## 2 "Shapiro-Wilk normality test" "X[[i]]"
## 3 0.9771699 4.638049e-11 Shapiro-Wilk normality test
## 4 "Shapiro-Wilk normality test" "X[[i]]"

# Histogrammes avec courbe normale
Graph1 <- ggplot(data = dataMTL) +
  geom_histogram(aes(x = HabKm2, y = ..density..),
                 bins = 30, color = "#343a40", fill = "#a8dadc") +
  labs(x="Habitants au km2", y = "densité")+
  stat_function(fun = dnorm,
                args = list(mean = mean(dataMTL$HabKm2),
                            sd = sd(dataMTL$HabKm2)),
                color = "#e63946", size = 1.2)

Graph2 <- ggplot(data = dataMTL) +
  geom_histogram(aes(x = HabKm2_log, y = ..density..),
                 bins = 30, color = "#343a40", fill = "#a8dadc") +
  labs(x="habitants au km2 (logarithme)", y = "densité")+
  stat_function(fun = dnorm,
                args = list(mean = mean(dataMTL$HabKm2_log),
                            sd = sd(dataMTL$HabKm2_log)),
                color = "#e63946", size = 1.2)

```

```

Graph3 <- ggplot(data = dataMTL) +
  geom_histogram(aes(x = HabKm2_sqrt, y = ..density..),
                 bins = 30, color = "#343a40", fill = "#a8dadc") +
  labs(x="habitants au km2 (racine carrée)", y = "densité")+
  stat_function(fun = dnorm,
                args = list(mean = mean(dataMTL$HabKm2_sqrt),
                            sd = sd(dataMTL$HabKm2_sqrt)),
                color = "#e63946", size = 1.2)

Graph4 <- ggplot(data = dataMTL) +
  geom_histogram(aes(x = HabKm2_inv, y = ..density..),
                 bins = 30, color = "#343a40", fill = "#a8dadc") +
  labs(x="habitants au km2 (inverse)", y = "densité")+
  stat_function(fun = dnorm,
                args = list(mean = mean(dataMTL$HabKm2_inv), sd = sd(dataMTL$HabKm2_inv)),
                color = "#e63946", size = 1.2)

ggarrange(plotlist = list(Graph1, Graph2, Graph3, Graph4), ncol = 2, nrow=2)

```

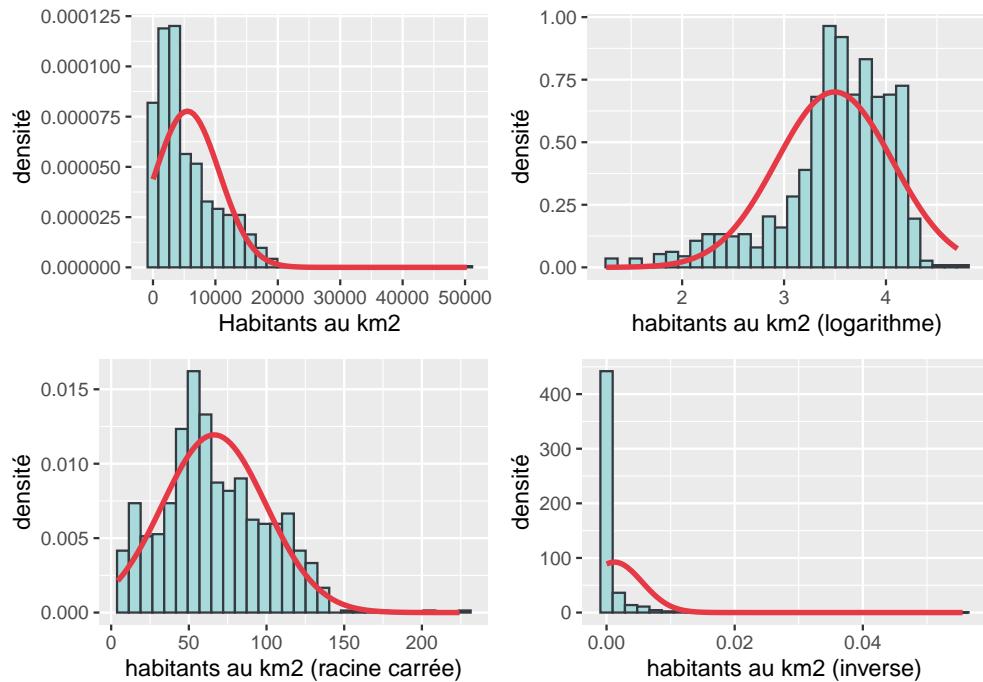


FIG. 2.36 : Histogramme des transformations

La variable *HabKm2* est asymétrique positive et leptokurtique. Tant les valeurs des statistiques de forme, du test de Shapiro-Wilk que les histogrammes semblent démontrer que la transformation la plus efficace est la racine carrée. Si la variable originale est asymétrique positive, sa transformation logarithme est par contre asymétrique négative. Cela démontre que la transformation logarithmique n'est pas toujours la panacée.

2.6 Statistiques descriptives sur des variables qualitatives et semi-qualitatives

2.6.1 Les fréquences

En guise de rappel, les variables nominales, ordinaires et semi-quantitatives comprennent plusieurs modalités pour lesquelles plusieurs types de fréquences sont généralement calculées. Pour illustrer le tout, nous avons extrait du recensement de 2016 de Statistique Canada les effectifs des modalités de la variable sur le principal mode de transport utilisé pour les déplacements domicile-travail, et ce, pour la subdivision de recensement (MRC) de l'île de Montréal (tableau 2.10). Les différents types de fréquences sont les suivantes :

- les fréquences absolues simples (**FAS**) ou fréquences observées représentent le nombre d'observations pour chacune des modalités. Par exemple, sur 857 540 navetteurs domicile-travail (ligne totale), seulement 30 645 optent pour le vélo, alors que 427 530 conduisent un véhicule motorisé (automobile, camion ou fourgonnette) comme principal mode de transport.
- les fréquences relatives simples (**FRS**) sont les proportions de chaque modalité sur le total ($30645/857540 = 0,036$); leur somme est égale à 1. Elles peuvent bien entendu être exprimées en pourcentage ($30645/857540 \times 100 = 3,57$); leur somme est alors égale à 100%. Par exemple, 3,7% des navetteurs utilisent le vélo comme mode de transport principal.
- les fréquences absolues cumulées (**FAC**) représentent la fréquence observée (FAS) de la modalité auxquelles sont additionnées celles qui la précèdent. La valeur de la FAC pour la dernière est donc égale au total.
- À partir des fréquences absolues cumulées (FAC), il est alors possible de calculer les fréquences relatives cumulées (**FRC**) en proportion ($453930/857540 = 0,529$) et en pourcentage ($453930/857540 \times 100 = 52,93$). Par exemple, plus de la moitié des navetteurs utilisent l'automobile comme mode de transport principal (passager ou conducteur).



Les fréquences cumulées : peu pertinentes pour les variables nominales

Le calcul et l'analyse des fréquences cumulées (absolues et relatives) sont très souvent inutiles pour les variables nominales.

Par exemple, au tableau 2.10, la fréquence cumulée relative (en %) est de 87,43% pour la troisième ligne. Cela signifie que 87,43% des navetteurs se déplacent en véhicule motorisé (conducteur ou passager) ou en transport en commun. Par contre, si la troisième modalité avait été *à pied*, le pourcentage aurait été de 61,02 ($52,93 + 8,09$). Si vous souhaitez calculer les fréquences cumulées sur une variable nominale, assurez-vous que l'ordre des modalités vous convient et de le modifier au besoin. Sinon, abstenez-vous de les calculer!

Les fréquences cumulées : très utiles pour l'analyse pour des variables ordinaires ou semi-quantitatives

Pour des modalités hiérarchisées (variable ordinaire ou semi-quantitative), l'analyse des fréquences cumulées

TAB. 2.10 : Les différents types de fréquences sur une variable qualitative ou semi-qualitative

Mode de transport	FAS	FRS	FRS (%)	FAC	FRC	FRC (%)
Véhicule motorisé (conducteur)	427 530	0,499	49,86	427 530	0,499	49,86
Véhicule motorisé (passager)	26 400	0,031	3,08	453 930	0,529	52,93
Transport en commun	295 860	0,345	34,50	749 790	0,874	87,43
À pied	69 410	0,081	8,09	819 200	0,955	95,53
Bicyclette	30 645	0,036	3,57	849 845	0,991	99,10
Autre moyen	7 695	0,009	0,90	857 540	1,000	100,00
Total	857 540	1,000	100,00			

(absolues et relatives) est par contre très intéressante. Par exemple, au tableau 2.11, elle permet de constater rapidement que sur l'île de Montréal, un peu moins du tiers de la population à moins de 25 ans (35,95%) et 83,33% moins de 65 ans.

Différents graphiques peuvent être construits pour illustrer la répartition des observations : les graphiques en barres (verticales et horizontales) avec les fréquences absolues, les diagrammes circulaires ou en anneau pour les fréquences relatives (figure 2.37). Ces graphiques seront présentés plus en détails dans le chapitre suivant.

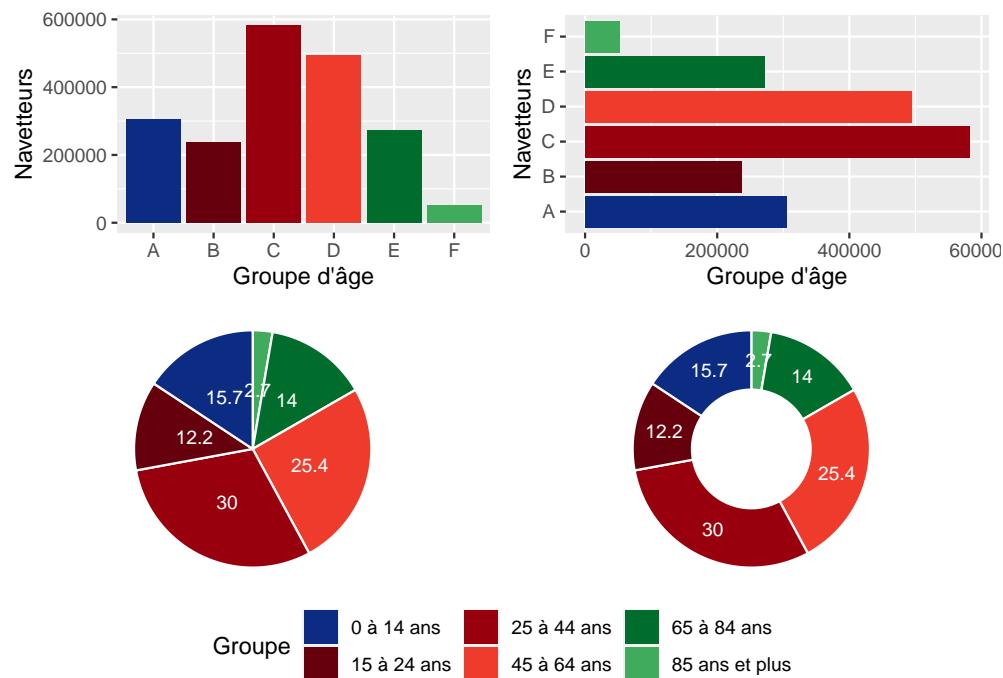


FIG. 2.37 : Différents graphiques pour représenter les fréquences absolues et relatives

2.6.2 Mise en œuvre dans R

La syntaxe ci-dessous permet de calculer les différentes fréquences présentées au tableau 2.11. Notez que pour les fréquences cumulées, nous utilisons la fonction `cumsum`.

```
# Vecteur pour les noms des modalités
Modalite <- c("0 à 14 ans",
```

TAB. 2.11 : Les différents types de fréquences sur une variable semi-qualitative

Groupes d'âge	FAS	FRS	FRS (%)	FAC	FRC	FRC (%)
0 à 14 ans	304 470	0,157	15,68	304 470	0,157	15,68
15 à 24 ans	237 555	0,122	12,23	542 025	0,279	27,91
25 à 44 ans	582 150	0,300	29,98	1 124 175	0,579	57,89
45 à 64 ans	494 205	0,254	25,45	1 618 380	0,833	83,33
65 à 84 ans	271 560	0,140	13,98	1 889 940	0,973	97,32
85 ans et plus	52 100	0,027	2,68	1 942 040	1,000	100,00
Total	1 942 040	1,000	100,00			

```

    "15 à 24 ans",
    "25 à 44 ans",
    "45 à 64 ans",
    "65 à 84 ans",
    "85 ans et plus")
# Vecteur pour les fréquences absolues simples (FAS)
Navetteurs <- c(304470,237555,582150,494205,271560,52100)
# Somme des FAS
sumFAS <- sum(Navetteurs)
# Construction du dataframe avec les deux vecteurs
df <- data.frame(
  GroupeAge = Modalite,
  FAS = Navetteurs,
  FRS = Navetteurs / sumFAS,
  FRSpct = Navetteurs / sumFAS * 100,
  FAC = cumsum(Navetteurs),
  FRC = cumsum(Navetteurs) / sumFAS,
  FRCpct = cumsum(Navetteurs) / sumFAS * 100
)
df

```

##	GroupeAge	FAS	FRS	FRSpct	FAC	FRC	FRCpct
## 1	0 à 14 ans	304470	0.15677844	15.677844	304470	0.1567784	15.67784
## 2	15 à 24 ans	237555	0.12232240	12.232240	542025	0.2791008	27.91008
## 3	25 à 44 ans	582150	0.29976211	29.976211	1124175	0.5788629	57.88629
## 4	45 à 64 ans	494205	0.25447725	25.447725	1618380	0.8333402	83.33402
## 5	65 à 84 ans	271560	0.13983234	13.983234	1889940	0.9731725	97.31725
## 6	85 ans et plus	52100	0.02682746	2.682746	1942040	1.0000000	100.00000

2.7 Pour aller un peu plus loin : les statistiques descriptives pondérées

Dans la section 2.5, les différentes statistiques descriptives sur des variables quantitatives – paramètres de tendance centrale, de position, de dispersion et de forme – ont été largement abordées. Il est possible de calculer ces différentes statistiques en tenant compte d'une pondération. La statistique descriptive pondérée la plus connue est certainement la moyenne arithmétique pondérée. Son calcul est très simple ; pour chaque observation, deux valeurs sont disponibles :

- x_i , soit la valeur de la variable X pour l'observation i
- w_i , soit la valeur de la pondération pour i .

Prenez soin de comparer les deux équations ci-dessous (à gauche, la moyenne arithmétique ; à droite, la moyenne arithmétique pondérée). Vous constaterez rapidement qu'il suffit simplement de multiplier chaque observation par sa pondération (numérateur) et de diviser ce produit par la somme des pondérations (dénominateur ; et non par n , soit le nombre d'observations comme pour la moyenne arithmétique non pondérée).

$$\bar{x} = \frac{\sum_{i=1}^n x_i}{n} \text{ versus } \bar{m} = \frac{\sum_{i=1}^n w_i x_i}{\sum_{i=1}^n w_i} \quad (2.31)$$

TAB. 2.12 : Calcul de la moyenne pondérée

Observation	x_i	w_i	$x_i \times w_i$
1	200	20	4 000
2	225	80	18 000
3	275	50	13 750
4	300	200	60 000
Somme	1 000	350	95 750
Moyenne	250		
Moyenne pondérée			274

NOTE : Calcul d'autres statistiques descriptives pondérées

Nous n'allons pas reporter ici les formules des versions pondérées de toutes les statistiques descriptives. Retenez toutefois le principe suivant permettant de les calculer à partir de l'exemple du tableau 2.12. Pour la variable X , dupliquons respectivement 20, 80, 50, 200 fois les observations 1 à 4. Si nous calculons la moyenne arithmétique sur ces valeurs dupliquées, alors cette valeur sera identique à la celle de la moyenne arithmétique pondérée. Le même principe reposant sur la duplication des valeurs s'applique à l'ensemble des statistiques descriptives.

Dans un article récent, Alvarenga et al. (?) évaluent l'accessibilité aux aires de jeux dans les parcs de la Communauté métropolitaine de Montréal (CMM). Pour les 881 secteurs de recensement de la CMM, ils ont calculé la distance à l'aire de jeux la plus proche à travers le réseau de rues. Ce résultat, cartographié à la figure 2.38, permet d'avancer le constat suivant : «la quasi-totalité des secteurs de recensement de l'agglomération de Montréal présente des distances de l'aire de jeux la plus proche inférieures à 500 m, alors que les secteurs situés à plus d'un kilomètre d'une aire de jeux sont très majoritairement localisés dans les couronnes nord et sud de la CMM» (? , p. 238).

Pour chaque secteur de recensement, Alvarenga et al. (?) disposent des données suivantes :

- x_i , soit la distance à l'aire de jeux la plus proche pour le secteur de recensement i et
- w_i , la pondération, soit le nombre d'enfants de moins de dix ans.

Il est alors possible de calculer les statistiques descriptives de la proximité à l'aire de jeux la plus proche en tenant compte du nombre d'enfants résidant dans chaque secteur de recensement (tableau 2.13). Cet exercice permet de conclure que : « [...] globalement, les enfants ont une bonne accessibilité aux aires de jeux sur le territoire de la CMM. [...] Les enfants sont en moyenne à un peu plus de 500 m de l'aire de jeux la plus proche (moyenne = 559 ; médiane = 512). Toutefois, les valeurs percentiles extrêmes signalent que respectivement 10% et 5% des enfants résident à près de 800 m et à plus de 1000 m de l'aire de jeux la plus proche » (? , p. 236).

De nombreux *packages* sont disponibles pour calculer des statistiques pondérées, dont notamment `Weighted.Desc.Stat` et `Hmisc` utilisés dans la syntaxe ci-dessous.

TAB. 2.13 : Statistiques de l'aire de jeux la plus proche par secteur de recensement pondérées par la population de moins de 10 ans

N	Moyenne	P5	P10	Q1	Médiane	Q3	P90	P95
881	559	282	327	408	512	640	799	1 006

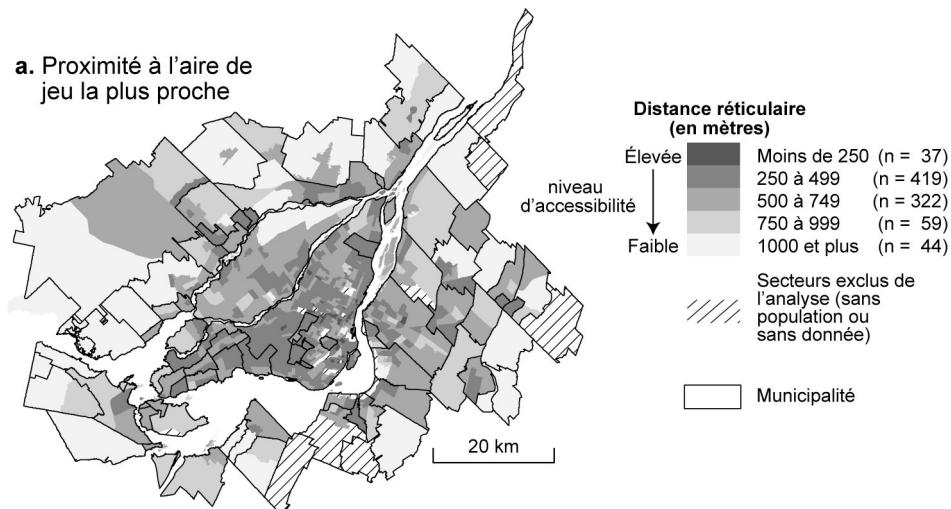


FIG. 2.38 : Accessibilité aux aires de jeux par secteur de recensement, Communauté métropolitaine de Montréal, 2016

```
library(foreign)
library(Hmisc)
library(Weighted.Desc.Stat)

df <- read.dbf("data/bivariee/SR_AireJeux_PopMoins10.dbf")

head(df, n = 5)
```

```
##      SRNOM PopMoins10 AireJeux
## 1 0659.06     380 600.1921
## 2 0410.02     390 324.4396
## 3 0863.01     325 524.3323
## 4 0734.05     875 574.6682
## 5 0073.00     100 352.9505
```

```
# xi (variable) et wi (pondération)
x <- df$AireJeux
w <- df$PopMoins10

# Calcul des paramètres de position
# Moyenne
Hmisc::wtd.mean(x, w)
```

```
## [1] 559.8026
```

```
Weighted.Desc.Stat::w.mean(x, w)
```

```
## [1] 559.8026
```

```
# Quartiles et percentile
Hmisc::wtd.quantile(x, weights=w, probs=c(.05, .10, .25, .50, .75, .90, .95))
```

```
##      5%    10%    25%    50%    75%    90%    95%
## 281.3623 327.3056 406.0759 511.5880 639.4813 798.6559 1011.5493
```

```
# Paramètres de dispersion avec le package Weighted.Desc.Stat
# Variance, écart-type et coefficient de variation
w.var(x,w)
```

```
## [1] 82818.18
```

```
w.sd(x,w)
```

```
## [1] 287.7815
```

```
w.cv(x,w)
```

```
## [1] 0.5140767
```

```
# Paramètres de forme avec le package Weighted.Desc.Stat
# Skewness et kurtosis
w.skewness(x, w)
```

```
## [1] 4.735351
```

```
w.kurtosis(x, w)
```

```
## [1] 41.17146
```

Bibliographie

- Allcott, H. and Gentzkow, M. (2017). Social media and fake news in the 2016 election. *Journal of economic perspectives*, 31(2) :211–36.
- Apparicio, P., Carrier, M., Gelb, J., Séguin, A.-M., and Kingham, S. (2016). Cyclists' exposure to air pollution and road traffic noise in central city neighbourhoods of montreal. *Journal of Transport Geography*, 57 :63–69.
- Apparicio, P. and Gelb, J. (2020). Cyclists' exposure to road traffic noise : A comparison of three north american and european cities. *Acoustics*, 2(1) :73–86.
- Apparicio, P., Gelb, J., Carrier, M., Mathieu, M.-È., and Kingham, S. (2018). Exposure to noise and air pollution by mode of transportation during rush hours in montreal. *Journal of Transport Geography*, 70 :182–192.
- Apparicio, P., Gelb, J., Dubé, A.-S., Kingham, S., Gauvin, L., and Robitaille, É. (2017). The approaches to measuring the potential spatial access to urban health services revisited : distance types and aggregation-error issues. *International journal of health geographics*, 16(1) :32.
- Apparicio, P., Gelb, J., Jarry, V., and Lesage-Mann, É. (2021). Cycling in one of the most polluted cities in the world : Exposure to noise and air pollution and potential adverse health impacts in delhi. *International journal of health geographics*, 20(1) :1–16.
- Apparicio, P., Gelb, J., and Mathieu, M.-È. (2019). Un atlas-web pour comparer l'exposition individuelle aux pollutions atmosphérique et sonore selon le mode de transport. *Cybergeo : European Journal of Geography*, (903).
- Apparicio, P., Maignan, D., and Gelb, J. Vifeco : An open-source software for counting features on a video. *Journal of Open Research Software*, 9(1).
- Boulos, M. N. K. and Geraghty, E. M. (2020). Geographical tracking and mapping of coronavirus disease covid-19/severe acute respiratory syndrome coronavirus 2 (sars-cov-2) epidemic and associated events around the world : how 21st century gis technologies are supporting the global fight against outbreaks and epidemics.
- Buregeya, J. M., Apparicio, P., and Gelb, J. (2020). Short-term impact of traffic-related particulate matter and noise exposure on cardiac function. *International Journal of Environmental Research and Public Health*, 17(4) :1220.
- De Alvarenga, B., Apparicio, P., and Séguin, A.-M. (2018). L'accessibilité aux aires de jeux dans les parcs de la communauté métropolitaine de montréal. *Cahiers de géographie du Québec*, 62(176) :229–246.
- Delaunay, D., Apparicio, P., Séguin, A.-M., Gelb, J., and Carrier, M. (2019). L'identification des zones calmes et un diagnostic d'équité environnementale à montréal. *The Canadian Geographer/Le Géographe canadien*, 63(2) :184–197.

- Field, A. P., Miles, J., and Field, Z. (2012). Discovering statistics using r.
- Gelb, J. and Apparicio, P. (2019). Noise exposure of cyclists in ho chi minh city : A spatio-temporal analysis using non-linear models. *Applied Acoustics*, 148 :332–343.
- Gelb, J. and Apparicio, P. (2020). Modelling cyclists' multi-exposure to air and noise pollution with low-cost sensors—the case of paris. *Atmosphere*, 11(4) :422.
- Gelb, J. and Apparicio, P. (2021a). Apport de la classification floue c-means spatiale en géographie : essai de taxinomie socio-résidentielle et environnementale à lyon. *Cybergeo : European Journal of Geography*.
- Gelb, J. and Apparicio, P. (2021b). Cyclists' exposure to atmospheric and noise pollution : a systematic literature review. *Transport Reviews*, pages 1–24.
- Joanes, D. and Gill, C. (1998). Comparing measures of sample skewness and kurtosis. *Journal of the Royal Statistical Society : Series D (The Statistician)*, 47(1) :183–189.
- Lebart, L., Morineau, A., and Piron, M. (1995). *Statistique exploratoire multidimensionnelle*, volume 3. Dunod Paris.
- Pumain, D. and Béguin, M. (1994). *La représentation des données géographiques : statistique et cartographie*. Armand Colin.
- Razali, N. M., Wah, Y. B., et al. (2011). Power comparisons of shapiro-wilk, kolmogorov-smirnov, lilliefors and anderson-darling tests. *Journal of statistical modeling and analytics*, 2(1) :21–33.
- Reed, W. J. (2002). On the rank-size distribution for human settlements. *Journal of Regional Science*, 42(1) :1–17.
- Reed, W. J. and Jorgensen, M. (2004). The double pareto-lognormal distribution : A new parametric model for size distributions. *Communications in Statistics - Theory and Methods*, 33(8) :1733–1753.
- Sean, O. (2018). Common probability distributions.
- Sobol, I. (1993). Sensitivity estimates for nonlinear mathematical models. *Mathematics and Computers in Simulation*, 1(4) :407–414.
- Spyratos, S., Vespe, M., Natale, F., Weber, I., Zagheni, E., and Rango, M. (2019). Quantifying international human mobility patterns using facebook network data. *PloS one*, 14(10) :e0224134.
- Tabachnick, B. G., Fidell, L. S., and Ullman, J. B. (2007). *Using multivariate statistics*, volume 5. Pearson Boston, MA.
- Yap, B. W. and Sim, C. H. (2011). Comparisons of various types of normality tests. *Journal of Statistical Computation and Simulation*, 81(12) :2141–2155.