

Introduction à R - partie 1

Aurélie Labbe, Léo Belzile

Nous allons passer en revue dans ce document les principales caractéristiques du langage de programmation statistique **R**. Pour plus de détails, veuillez vous référer aux différents manuels et fichiers d'aide.

Dans ce document, nous allons couvrir les thèmes suivants:

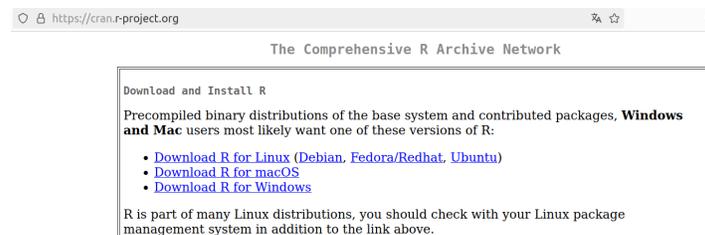
- Installation de **R** et RStudio Desktop
- Introduction à l'environnement de développement intégré RStudio
- Bases de la syntaxe **R**
- Classes et types d'objets en **R**
- Écrire des rapports avec Quarto

Installation

Installer R

Pour commencer, on installe la dernière version de **R** (la machine qui fait les calculs), actuellement 4.4.1 (Race for Your Life).

1. Allez sur le site du *Comprehensive R Archive Network (CRAN)*: <https://cran.r-project.org/>
2. Cliquez sur "Download **R** for **XXX**", où **XXX** est Mac, Linux ou Windows:



Si vous utilisez macOS, faites défiler le menu jusqu'au premier fichier `.pkg` de la liste (selon que votre machine soit Intel ou Apple silicon) et téléchargez-le.

Latest release:

For Apple silicon (M1-3) Macs:
[R-4.4.1-arm64.pkg](#)

SHA1-
hash: 616560b17092bbdd8b814d9ed92d098e52204830
(ca. 94MB, notarized and signed)

For older Intel Macs:
[R-4.4.1-x86_64.pkg](#)

SHA1-
hash: e66eb09244121d7db7f8fb41d3c06a7579fe93b5
(ca. 96MB, notarized and signed)

R 4.4.1 binary for macOS 11 (**Big Sur**) and higher, signed and notarized packages.

Contains R 4.4.1 framework, R.app GUI 1.80, Tcl/Tk 8.6.12 X11 libraries and Texinfo 6.8. The latter two components are optional and can be omitted when choosing "custom install", they are only needed if you want to use the `tcltk` R package or build package documentation from sources.

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

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

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

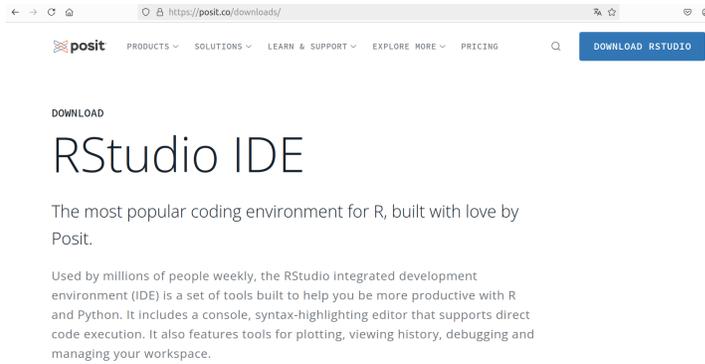
Si vous utilisez Windows, choisissez “base” (ou cliquez sur l’hyperlien en gras “install **R** for the first time”) et téléchargez le programme.

3. Double cliquez sur le fichier de téléchargement. Approuvez toutes les requêtes, comme pour n’importe quel autre programme.
4. Si vous utilisez macOS, [téléchargez et installez XQuartz](#).

Installer RStudio

Une fois **R** installé, nous installerons un environnement de développement intégré, soit une interface graphique, RStudio Desktop, pour interagir avec **R**.

1. Naviguez jusqu’à la section de téléchargements gratuits du site de RStudio: <https://posit.co/download/rstudio-desktop/>
2. Le site internet devrait automatiquement détecter votre système d’exploitation (Linux, macOS ou Windows). Cliquez sur le bouton pour lancer le téléchargement:



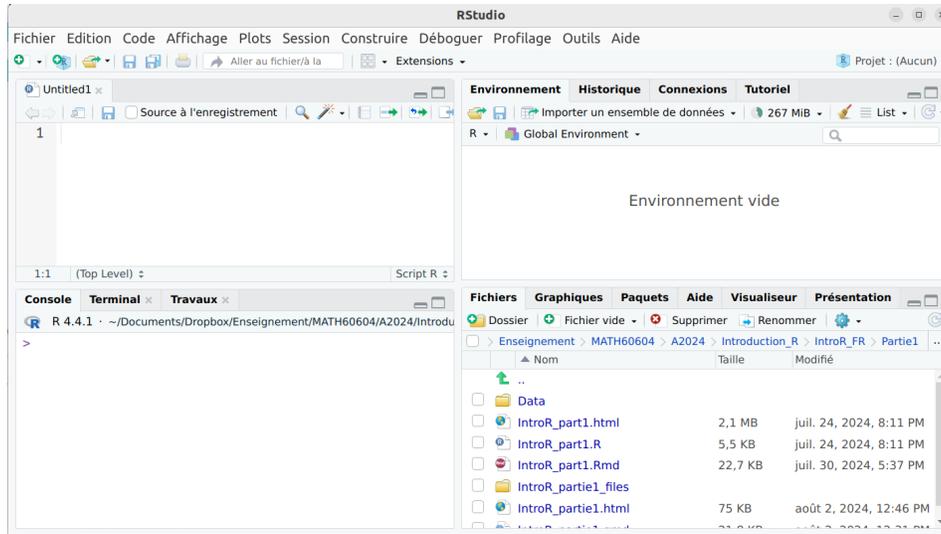
Sinon, naviguez vers le tableau et choisissez la version de RStudio adéquate. 3. Double-cliquez sur le fichier de téléchargement (potentiellement caché dans votre dossier de Téléchargements). Installez comme n'importe lequel autre logiciel. 4. Double-cliquez sur l'icône RStudio pour lancer l'application.

RStudio

RStudio est un environnement spécifiquement conçu pour éditer du code R. Il contient de nombreuses fonctionnalités très utiles telles que (entre autres):

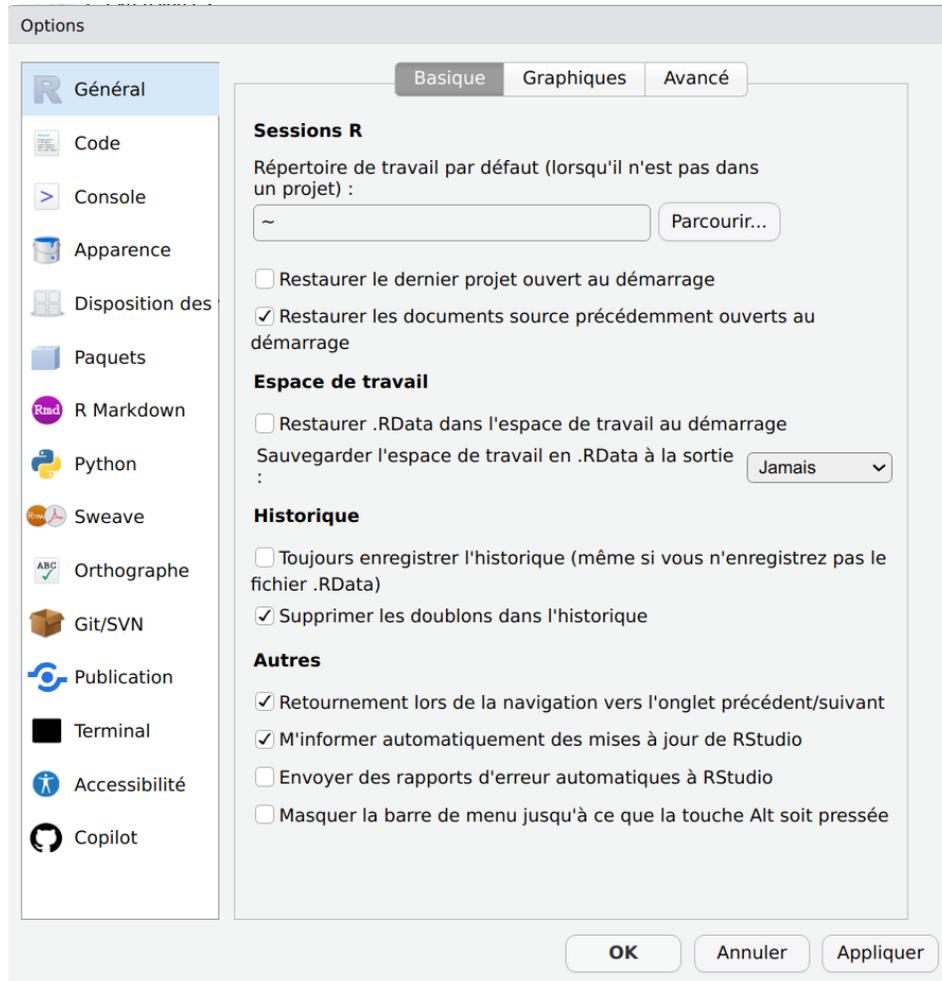
- Utilisation d'un code de couleur pour les différentes composantes d'un programme **R** (par ex: les commentaires sont en vert, certains mots-clé sont en bleu, etc.)
- Possibilité de faire rouler les commandes dans **R** directement à partir du fichier code sans avoir besoin de faire un copier-coller dans la console R
- Outils permettant de gérer les graphiques produits et de gestion de fichiers
- Possibilité de créer de façon automatique des rapports, etc.

RStudio est une plateforme gratuite et peut être installée sur Windows, Mac ou Linux. Lorsque vous l'ouvrez, vous allez voir qu'il contient quatre fenêtres:



Naviguez jusqu'à *Tools > Global Options* (Outils > Options globales en français). Vous pouvez modifier la langue de l'interface vers le français à partir de *General > Advanced* et en sélectionnant le français dans le menu déroulant de *User Interface Language*.

Par défaut, RStudio rechargera la session précédente. À partir du menu Général, modifiez les options pour ne jamais restaurer la session précédente.



Ouvrir un fichier

Les scripts contenant les commandes **R** sont enregistrés sous l'extension `.R`, mais ce sont simplement des fichiers textes. Vous pouvez ouvrir un fichier avec l'extension `.R` déjà existant ou ouvrir un nouveau fichier `.R` à l'aide du menu **Fichier** dans la fenêtre d'éditeur de texte (en haut à gauche). Utilisez de préférence l'encodage UTF-8 (via **Fichier** > **Enregistrer avec l'encodage**) si vous voulez utiliser des accents dans votre texte.

Certains documents contiennent des blocs de texte et du code, mélangeant la syntaxe [Mark-down](#) et des blocs de code R. Les fichiers Rmarkdown et Quarto (extensions `.Rmd` ou `.qmd`) incluent un en-tête qui définit les propriétés du document, suivi de texte et de code. Ils peuvent être compilés pour créer des pages web (fichiers `.html`) ou des documents au format portable (fichiers `.pdf`).

Interface de RStudio

Les quatre quadrants ou panneaux dans RStudio indique le fichier Source, la Console, l'environnement (premier onglet) et les autres options (Fichiers, Graphiques, Paquets, Aide, etc.)

Dirigez votre curseur dans la R et cliquez dans la Console. après le symbole `>`. Votre curseur devrait clignoter, ce qui indique que **R** est en attente de vos instructions. Tapez votre code (disons `2 + 2`, et appuyez sur Retour pour le compiler. Vous devriez voir apparaître le résultat suivant.

```
2 + 2
```

```
[1] 4
```

Ce résultat (et le code qui le génère) sont éphémères. Contrairement à d'autres langages de programmation, comme C, **R** peut compiler des instructions en temps réel, ce qui permet une plus grande interactivité. Pour recompiler, il faudra retaper les instructions. On va donc plutôt enregistrer le code dans un document **R** ou assigner le résultat à une variable pour l'utiliser subséquentement.

Vous pouvez utiliser le panneau Source pour taper vos instructions et enregistrer le tout sous format `.R` ou équivalent. Pour faire rouler un bloc de code d'un fichier de commandes **R**, il suffit de surligner les lignes que l'on veut exécuter dans la fenêtre contenant le code **R** et de cliquer sur le bouton **Exécuter** dans l'éditeur ou appuyer sur la touche "Retour". Vous pouvez aussi copier le code et coller le code dans la fenêtre Console.

Paquets

La popularité de **R** comme gratuitiel est due notamment à son importante archive de paquets, la plupart contribués par des utilisateurs, qui permettent d'obtenir et d'installer des collections de fonctions et de bases de données. Le *Comprehensive R Archive Network (CRAN)* permet d'installer directement dans **R**. La commande `install.packages("nom")`, où "nom" est une chaîne de caractères avec le nom de l'objet. Cela permet d'installer les fonctionnalités sur votre ordinateur, tandis que `library` permet de charger les fonctions dans l'environnement de travail. Certains paquets (`base`, `stat`, etc.) sont installés par défaut avec **R**, tandis que les quelques 20K paquets disponibles peuvent être téléchargés à l'aide des instructions.

Il suffit d'installer une seule fois un paquet (comme si on achetait un livre à la paquets) et de le charger ensuite si l'on veut utiliser les fonctionnalités (chaque fois qu'on veut utiliser, on l'extrait de la bibliothèque).

Charger un paquet ajoute à l’environnement toutes ses fonctions et bases de données, ce qui peut masquer des fonctions homonymes! Parfois, on n’utilisera qu’une fonction, et on peut mettre le nom du paquet, suivi de `::`, pour éviter les conflits et utiliser directement une fonction sans charger l’ensemble du paquet.

```
## Installer un paquet depuis le CRAN
## Effectuer cette action une seule fois
install.packages("remotes")
## Charger à chaque session/script
library(remotes)
library(tidyverse) # noter conflits
## Installation depuis Github
## :: pour utiliser une fonction d'un paquet
remotes::install_github("lbelzile/hecmoostat")
## Charger une base de données d'un paquet
data(distractio, package = "hecmoostat")
```

Exercice 0.1. Compilez les instructions du bloc de code précédent. Ensuite, installez le paquet `this.path` et vérifiez que l’installation a fonctionné. Vous pouvez aussi utiliser l’onglet “Paquets” pour Installer ou charger le paquet. Notez que **R** est sensible la casse.

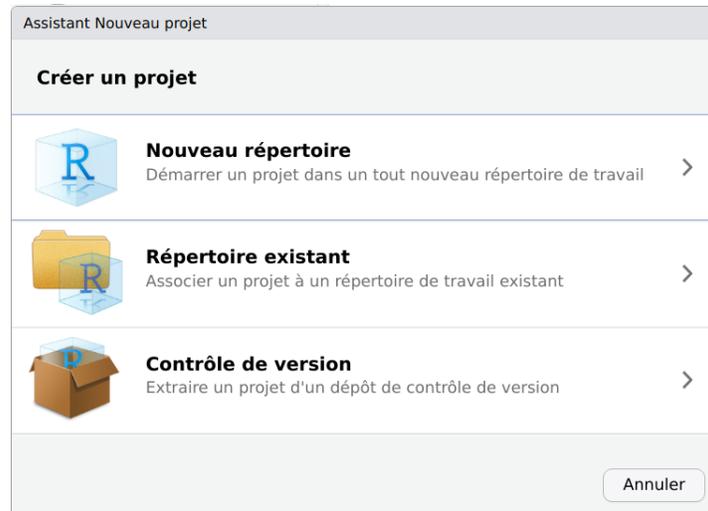
Répertoire de travail et projets

Certaines commandes de **R**, par exemple `load`, `save`, `pdf`, etc., permettent de charger ou de sauvegarder des fichiers. L’onglet “Fichiers” montre le chemin et la liste des dossiers et fichiers courants, tandis que la ligne sous “Console” indique le répertoire actuel (par défaut, `~`). Pour déterminer le répertoire ou le modifier, utilisez les commandes `setwd()` et `getwd()`.

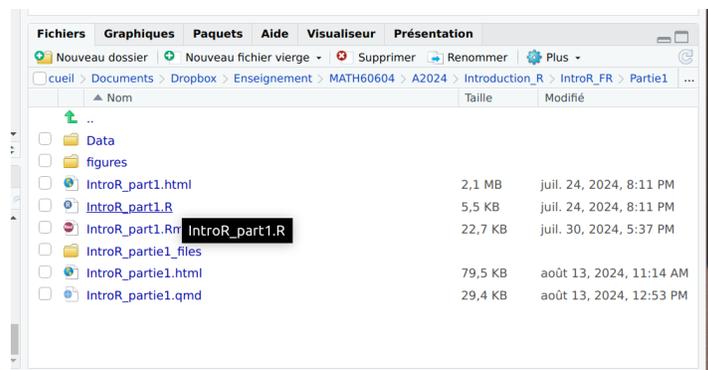
Durant votre session de travail, vous allez probablement devoir importer dans **R** des données d’un fichier stocké sur votre ordinateur, ou bien sauver certains résultats dans des fichiers. Si tous vos fichiers sont stockés au même endroit et que vous voulez éviter à chaque fois d’avoir à spécifier le chemin d’accès des fichiers au complet, vous pouvez spécifier à **R** que vous allez dorénavant travailler à partir d’un répertoire de votre choix. Ceci peut se faire en tapant la commande `setwd()`, avec comme argument l’adresse du répertoire:

Vous pouvez aussi le faire en cliquant dans le menu “Session” de l’éditeur de texte, puis en choisissant l’option “Définir le répertoire de travail” et en cliquant sur “Vers l’emplacement du fichier source” (auquel cas le répertoire est le même que celui où le programme **R** est stocké) ou bien sur “Choisir un répertoire” et en spécifiant le chemin d’accès. Si vous voulez charger des fichiers dans d’autres dossiers, vous devez utiliser le chemin relatif de ces derniers.

Une façon de contrôler l’emplacement des fichiers dans RStudio est de créer un projet, via Fichier > Nouveau Projet. Si vous cliquez par la suite sur le fichier `.Rproj`, cela ouvrira RStudio et pointera directement au répertoire qui contient le projet.



Exercice 0.2. Créez un projet dans le répertoire qui contient les fichiers téléchargés pour IntroR_partie1. Utilisez ensuite l’onglet “Fichier” pour charger le fichier IntroR_partie1.R.



Exercice 0.3. Tapez dans la console la commande `plot(hwy ~ displ, data = mpg)`. Cela devrait créer un graphique dans l’onglet “Graphiques”.

Aide

Pour obtenir de l’aide sur une fonction et sur les arguments, chaque fonction et base de données se trouvant dans un paquet doit être documentée. Vous pouvez utiliser le point d’interrogation, suivi d’un seul mot (ou de manière équivalente, la fonction `help()` ou deux points d’interrogation suivis d’un texte (entre guillemets anglais simples ou doubles) pour de l’aide. Par exemple,

```
?Normal
??"Normal distribution"
```

Si une page d'aide avec l'entrée correspondante existe, elle sera ouverte dans l'onglet "Aide". Si on utilise `??`, une recherche sera effectuée dans l'ensemble des paquets installés avec les mots clés utilisés. Les fichiers d'aide contiennent notamment des détails sur les arguments des fonctions et il y a souvent des exemples à la fin.

Exercice 0.4. Tapez directement dans la console la commande `help(mean)`. Le fichier d'aide de la commande `mean` en **R** devrait s'ouvrir et apparaître dans la fenêtre en bas à droite.

Avant de commencer

Assurez-vous que vous avez bien téléchargé tous les fichiers nécessaires pour la séance et que tout est sauvegardé dans un répertoire de votre ordinateur (fichier .html, fichiers de données et code .R).

Notex que vous pouvez en tout temps écrire des commentaires dans le fichier `.R` en incluant le signe `#` devant votre commentaire comme ceci:

```
# ceci est un commentaire
a = 1 # ceci est un commentaire mais a=1 est une commande R
```

Avant de commencer, voici quelques conseils très généraux sur la façon de gérer les données dans **R**:

- En général, il est plus facile d'importer les données dans **R** à partir d'un fichier `.txt` ou `.csv`, avec les fonctions `read.table` et `read.csv`
- Il est conseillé d'inclure le nom des variables du fichier de données directement dans la première ligne du fichier.
- **N'utilisez pas d'espace ni d'accent dans le nom de vos variables ou de vos dossiers**
- Utilisez des noms qui ont une signification pour le nom de vos variables (et non pas `x1`, `x2`, etc.)
- Évitez les noms de variables qui sont trop longs
- Évitez les formats de date compliqués: essayez de le remplacer par des colonnes jour / mois / année.
- Attention, le langage **R** est sensible à la casse: il faut faire la différence entre les majuscules et les minuscules dans le nom des variables et dans les commandes que vous utilisez.

Données utilisées pendant la séance

Les données électricité

Nous allons illustrer les différentes commandes et concepts à l'aide d'un fichier de données contenant la consommation en électricité d'une maison entre janvier 1991 et décembre 2000. Chaque ligne du fichier représente un mois donné. Les données sont dans les fichiers `electricbill.txt` et `electricbill.csv`. Les variables fournies sont les suivantes:

- `NUM`: identifiant de l'observation
- `YEAR`: année
- `MONTH`: mois
- `BILL`: montant de la facture en Dollars (inclus 5% taxe de vente)
- `TEMP`: température moyenne (en degrés Fahrenheit)
- `HDD`: mesure quantifiant la quantité d'énergie nécessaire pour chauffer le bâtiment
- `CDD`: mesure quantifiant la quantité d'énergie nécessaire pour refroidir le bâtiment
- `SIZE`: nombre de personnes vivant dans la maison
- `METER`: présence d'un nouveau compteur électrique (1=oui, 0=non)
- `PUMP1`: présence d'une nouvelle pompe de chauffage (1=oui, 0=non)
- `PUMP2`: présence d'une nouvelle pompe de chauffage pour la seconde fois (1=oui, 0=non)
- `RIDER TOTAL`: charge totale (par kwh)
- `CONSUMPTION`: consommation (en kwh)

Les données baby-boom

Vous aurez à pratiquer les notions vues en classe à l'aide du jeu de données `babyboom.txt` qui contient l'heure de naissance, le sexe et le poids à la naissance de 44 bébés nés dans une période de 24h à l'hôpital de Brisbane, en Australie. Les variables sont les suivantes:

- `Time`: heure de naissance
- `Sex`: sexe du bébé (1 = fille, 2 = garçon)
- `Weight`: poids à la naissance en grammes
- `Time.midnight`: nombre de minutes après minuit passées au moment de la naissance

Calculs de base et arithmétique

Il est possible d'utiliser **R** comme une simple calculatrice. En l'absence d'instructions sur l'utilisation de la sortie d'une commande, **R** affiche généralement le résultat à l'écran. Pour l'instant, ignorez le `[1]` avant la réponse: nous verrons que cela est utile lorsque **R** génère plusieurs lignes à la fois. Notez que **R** respecte les règles mathématiques standard de la multiplication et de la division, avant l'addition et la soustraction: il divise 2 par 3 avant d'ajouter 100 pour réaliser l'opération $2/3+100$. Voici quelques exemples de code de base:

```
2 + 3
pi
2 + 3 * pi
log(2 + 3 * pi)
log(2, base = 10)
log(2, base = 2)
exp(2.435785)
```

Exercice 0.5. Compilez le code et vérifiez l'arithmétique.

Types de variables

Dans **R**, les différents objets ou vecteur peuvent avoir des types différents, selon que la valeur est logique, entière, numérique, catégorielle ou une chaîne de caractère. Utilisez `typeof` pour obtenir le type de l'objet

- les variables numériques (`double`, 1.2, 0.4),
- les entiers (`integer`, comme les séquences 1:20 ou les nombres suivis d'un L, comme 1L),
- les chaînes de caractères (`character`) et
- les booléens de type `logical` (`TRUE` ou `FALSE`, éviter les abbréviations car seuls ces deux termes sont réservés).

```
typeof(1:5) # entier, avec 2L
```

```
[1] "integer"
```

```
typeof(TRUE) # logique / booléen
```

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

```
typeof(1.234) # numérique
```

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

```
typeof("string") # chaîne de caractères, entre guillemets simples ou doubles
```

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

Il y a peu de différence entre les valeurs numériques et les entiers, sinon le stockage. Les valeurs `TRUE` et `FALSE` sont des mots réservés dans **R**, contrairement aux abbréviations `T` ou `F` dont l'usage n'est pas recommandé! Plusieurs fonctions ont des choix d'arguments prérequis et retourneront un message d'erreur (par exemple, on ne peut calculer la moyenne de texte, et l'opération retournera un message d'erreur).

Un objet peut contenir des valeurs manquantes, soit `NA` pour non disponible ou `NaN` pour "pas un nombre", si par exemple on fait une opération arithmétique invalide comme `log(-3)` dans la sortie qui suit. On peut enlever les valeurs manquantes avec `na.omit`, `is.na` retourne un vecteur logique avec `TRUE` si l'argument est manquant et `FALSE` sinon. Plusieurs fonctions de base ont un argument `na.rm`, voir par exemple la documentation de `?mean`.

```
mean(1:5)
```

```
[1] 3
```

```
mean(c("abc", "def"))
```

```
Warning in mean.default(c("abc", "def")): argument is not numeric or logical:
returning NA
```

```
[1] NA
```

D'autres types spécialisés existent pour encoder des nombres complexes. `NULL` est un pointeur vers un ensemble vide, `Date` permet de stocker des dates et les variables catégorielles sont stockées dans des facteurs (`factor`). Les facteurs sont encodés par des valeurs entières à l'interne, mais possèdent des étiquettes qui permettent l'interprétation des différentes modalités. La plupart des fonctions pour la modélisation traitent ces variables différemment.

Par défaut dans **R**, la catégorie de référence est la première valeur en ordre alphanumérique.

```
facteur <- factor(1:3, # vecteur de valeurs
  levels = 1:4, # valeur des niveaux, par défaut les valeurs uniques
  labels = c("un", "deux", "trois", "quatre") # étiquettes
)
# classe du vecteur
class(facteur)
```

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

```
# Modifier la catégorie de référence
relevel(facteur, ref = "trois")
```

```
[1] un    deux  trois
Levels: trois un deux quatre
```

```
# Convertir un vecteur de caractères en facteur et
# imprimer le décompte de chaque catégorie
table(factor(mpg$drv))
```

```
 4  f  r
103 106 25
```

La commande `ls()` permet de voir tous les objets qui sont actuellement disponibles dans votre environnement de travail (Onglet “Environnement”):

```
ls()
```

```
[1] "a"      "facteur"
```

Vous pouvez effacer un objet avec la commande `rm`. Il est recommandé de ne pas recharger les objets à chaque fois que l’on ouvre **R**.

```
rm(a) # retirer un élément 'a'
rm(ls()) # retirer la liste de toutes les variables
```

Classes d’objets

R est un langage de programmation orienté objet qui permet de créer une variété d’objets tels que des :

- **vecteur**: objet **par défaut**, un ensemble ordonné de même type
- **matrice**: une table de données (toutes du même type ou format) avec deux dimensions (des lignes et des colonnes).
- **listes**: une liste avec des arguments de type et de longueur potentiellement différentes
- **data frame**: une table de données (liste) dans laquelle les colonnes peuvent être de format différents (caractère, numérique, etc.)

La fonction `class` permet d'obtenir la classe de l'objet. La classe d'un objet permet de définir des fonctions génériques (`methods`). Par exemple, pour un modèle de régression linéaire,

```
methods(class = "lm")
```

```
[1] add1          alias          anova          case.names     coerce
[6] confint       cooks.distance deviance       dfbeta         dfbetas
[11] drop1         dummy.coef     effects        extractAIC     family
[16] formula       fortify        hatvalues      influence      initialize
[21] kappa         labels         logLik         model.frame    model.matrix
[26] nobs          plot           predict        print          proj
[31] qr            residuals     rstandard     rstudent      show
[36] simulate      slotsFromS3   summary       variable.names vcov
see '?methods' for accessing help and source code
```

Autrement dit, le résultat dépend de la classe. Les méthodes les plus fréquentes incluent l'impression (`print`, appelée par défaut lorsqu'on imprime le nom d'un objet dans la console), le récapitulatif (`summary`) et le graphique (`plot`).

```
a = 3 # Crée un vecteur de taille 1 (scalaire) nommé 'a', qui prend la valeur 3
a # Imprimer la valeur de a
```

```
[1] 3
```

```
a <- 3 # meme code avec l'opérateur <-, préféré
b <- 5 # Crée une vecteur 'b', qui prend la valeur 5
b - a # Affiche la valeur de b - a
```

```
[1] 2
```

```
# Stocke la valeur de b / a dans une nouvelle variable 'd'
# et imprimer ce qui est entre parenthèse
(d <- b / a)
```

```
[1] 1.666667
```

Vecteurs

Un vecteur est une séquence d'éléments du même type (par exemple: tous numérique ou tous de type chaînes de caractères). Avant de voir comment lire un vecteur directement à partir d'un jeu de données, voyons comment créer un vecteur à la main.

La commande pour créer un vecteur à la main dans **R** est `c` (concaténer) et on peut accéder aux différents éléments du vecteur avec des crochets:

```
# Créer un vecteur x de 10 éléments numériques
x <- c(2, 3, 1, 5, 4, 6, 5, 7, 6, 8)
# Séquence régulière d'entiers
1:10 # entiers de 1 à 10, pas enregistré
```

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

```
seq(1, 10, by = 1)
```

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

```
x # Imprimer valeurs de x
```

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

```
x[3] # Affiche le 3e élément de x
```

```
[1] 1
```

```
a <- x[3] # Sauvegarde le 3e élément de 'x' dans un objet appelé 'a'
a # Affiche la valeur de a
```

```
[1] 1
```

```
length(x) # longueur de x, un attribut des vecteurs
```

```
[1] 10
```

Exercice 0.6. Créer un vecteur appelé `annee` de longueur 7 et le remplir avec les nombres 2010 à 2016. Afficher les valeurs de ce vecteur sur votre écran.

Exercice 0.7. Vous pouvez ajouter / soustraire un nombre à un vecteur (ou multiplier un vecteur par un nombre). Par exemple, essayez de soustraire le nombre 2009 au vecteur `annee` en tapant la commande `annee-2009` et voir ce qui se passe.

Exercice 0.8. Créer une variable appelée `montant.dollars` qui prend les valeurs (55, 70, 100, 20, 15). Ensuite, créer une variable appelée `montant.euros` qui correspond aux mêmes montants, mais en euros (notez que 1 dollars canadien = 0,66 euro)

Il est possible de additionner, soustraire, multiplier et diviser des vecteurs entre eux, élément par élément. Voici un exemple:

```
salaire <- c(36, 54, 42, 81) # Salaire pour 4 employés
# Pourcentage d'augmentation de salaire pour chaque employé
augmentation <- c(0.02, 0.02, 0.03, 0.05)
salaire_final <- salaire * (1 + augmentation)
salaire_final
```

```
[1] 36.72 55.08 43.26 85.05
```

Exercice 0.9. Faites-vous un petit exemple pour illustrer le cas de l'addition / soustraction de vecteurs.

Matrices

Les matrices sont définies comme une table de scalaires (avec des lignes et des colonnes). Toutes les colonnes d'une matrice doivent avoir le même mode (numérique, caractère, etc.) et la même longueur, comme dans un tableau de données. Nous verrons comment stocker les données d'un fichier externe dans une matrice en R, mais voyons tout d'abord comment créer une matrice de données à la main et en extraire certains éléments.

La commande générale pour créer une matrice est `matrix`. Il faut spécifier le nombre de lignes `nrow` ou de colonnes `ncol`. Par défaut, les entrées sont stockées colonne par colonne.

```
z.trans <- matrix(1:9, nrow = 3)
# Remplir plutôt la matrice ligne par ligne
z <- matrix(1:9, nrow = 3, byrow = TRUE)
# Attributs de matrices
dim(z) # dimensions (# lignes, n# colonnes)
```

```
[1] 3 3
```

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

```
[1] 3
```

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

```
[1] 3
```

```
length(z) # nombre d'entrées
```

```
[1] 9
```

Voici maintenant comment extraire ou accéder à des éléments spécifiques de la matrice:

```
z[2,3] # Élément de la ligne 2 et colonne 3
```

```
[1] 6
```

```
z[,1] # 1e colonne
```

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

```
z[1,] # 1e ligne
```

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

Voici un autre exemple:

```
z <- matrix(0, nrow = 5, ncol = 2)
# Remplir une matrice 5 par 2 de zéros,
# l'argument 0 est recyclé
# Créer deux vecteurs
amount.dollars <- c(55, 70, 100, 20, 15)
amount.euros <- 0.66 * amount.dollars
# Remplir les 2 colonnes de la matrice avec chacun des vecteurs
z[,1] <- amount.dollars
z[,2] <- amount.euros
z
```

```
      [,1] [,2]
[1,]   55 36.3
[2,]   70 46.2
[3,]  100 66.0
[4,]   20 13.2
[5,]   15  9.9
```

Listes

Une liste est une collection d'objets de type et de longueur potentiellement différents. C'est particulièrement utile pour enregistrer la sortie d'une procédure.

```
# Liste avec arguments nommés
liste <- list(a = 1:3, b = letters[1:10])
# Extraire un élément spécifique par son nom
liste$a # premier élément
```

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

```
liste[[2]] # deuxième élément
```

```
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"
```

```
# Ajuster un modèle linéaire - le résultat est une liste avec des attributs supplémentaires
linmod <- lm(formula = hwy ~ displ, data = mpg)
typeof(linmod)
```

```
[1] "list"
```

```
# Noms des éléments de la liste
names(linmod)
```

```
[1] "coefficients" "residuals"      "effects"         "rank"
[5] "fitted.values" "assign"         "qr"              "df.residual"
[9] "xlevels"       "call"          "terms"           "model"
```

Bases de données

La base de données (`data.frame`) est la classe d'objet le plus couramment utilisé pour stocker les tables de données dans **R**: il s'agit d'une liste avec des éléments de même longueur utilisé pour le stockage de base de données. Avant d'examiner en détail ce type de structure, voyons tout d'abord comment nous pouvons lire un fichier et stocker les données dans un objet `data.frame`.

La fonction **R** `read.table` qui permet de lire des données stockées dans un fichier. Consultez l'aide pour plus de détails

Les options principales de cette fonction (parmi un grand nombre) sont les suivantes:

- **file**: le nom du fichier (incluant le chemin d'accès relatif) qui contient les données à lire. Chaque ligne du fichier va devenir une ligne de la table dans **R**. Si le chemin absolu du fichier n'est pas spécifié, **R** va le chercher à partir du répertoire courant de travail.
- **header**: une valeur logique (`TRUE` ou `FALSE`) qui indique si la première ligne du fichier contient le nom des variables.
- **sep**: le caractère de séparation des entrées dans le fichier. Si `sep = " "` le séparateur est un espace blanc (valeur par défaut). Si `sep="\t"`, le séparateur est une tabulation. Les choix des types de séparateurs sont multiples (virgule, point-virgule, etc.)
- **dec**: le type de caractère utilisé pour les décimales (par défaut, le point).
- **row.names**: un vecteur des noms de lignes, si il y en a. Si `row.names` n'est pas spécifié, les lignes sont numérotées.
- **col.names**: vecteur de noms (optionnel) des variables. Si `header = TRUE`, la première ligne sert de vecteur, il faut donc éviter les espaces.
- **na.strings**: les caractères dans le fichier qui doivent être interprétés comme valeur manquantes. Les espaces blancs sont par défaut considérés comme des valeurs manquantes.
- **skip**: le nombre de lignes à sauter dans le fichier avant de commencer à lire les données.

Par exemple, voici le code qui permet de lire les données du fichier `electricbill.txt` (la sortie **R** n'est pas affichée ici pour sauver de l'espace). Il faut assigner la base de données à une variable pour pouvoir l'utiliser subséquemment.

Si vous voulez avoir seulement un aperçu de l'objet `electric` et afficher les premières lignes au lieu d'afficher les données au complet à l'écran, vous pouvez utiliser les fonctions `str` ou `head` qui sont très pratiques:

```
electric <- read.table(file = "Data/electricbill.txt",
                      header = TRUE)
head(electric, n = 5L) # cinq premières lignes
```

	NUM	YEAR	MONTH	BILL	TEMP	HDD	CDD	SIZE	METER	PUMP1	PUMP2	RIDER
1	1	1991	Jan	162.10	29.1	1229	0	4	0	0	0	-0.002288

```

2  2 1991  Feb 256.90 31.5  999  0  4  0  0  0 -0.002288
3  3 1991  Mar 151.15 41.9  734  0  4  0  0  0 -0.002288
4  4 1991  Apr 118.76 53.4  373  0  4  0  0  0 -0.000600
5  5 1991  May 100.71 63.7  162 94  4  0  0  0 -0.000634

```

CONSUMPTION

```

1      5600
2      9463
3      5154
4      3576
5      2894

```

```
str(electric)
```

```

'data.frame':  120 obs. of  13 variables:
 $ NUM      : int  1 2 3 4 5 6 7 8 9 10 ...
 $ YEAR     : int  1991 1991 1991 1991 1991 1991 1991 1991 1991 1991 ...
 $ MONTH    : chr  "Jan" "Feb" "Mar" "Apr" ...
 $ BILL     : num  162 257 151 119 101 ...
 $ TEMP     : num  29.1 31.5 41.9 53.4 63.7 72.9 76.8 75 68.4 56.8 ...
 $ HDD      : int  1229 999 734 373 162 4 0 0 52 326 ...
 $ CDD      : int  0 0 0 0 94 211 328 261 110 18 ...
 $ SIZE     : int  4 4 4 4 4 4 4 4 4 4 ...
 $ METER    : int  0 0 0 0 0 0 0 0 0 0 ...
 $ PUMP1    : int  0 0 0 0 0 0 0 0 0 0 ...
 $ PUMP2    : int  0 0 0 0 0 0 0 0 0 0 ...
 $ RIDER    : num  -0.002288 -0.002288 -0.002288 -0.0006 -0.000634 ...
 $ CONSUMPTION: int  5600 9463 5154 3576 2894 2257 2826 2986 1774 2713 ...

```

Exercice 0.10. À l’aide de la commande `read.csv`, lire les données du fichier `electricbill.csv`, qui utilise un `;` comme séparateur d’entrées. Vous devriez obtenir les mêmes résultats que ci-dessus.

Notez qu’il y a la possibilité dans **R** de lire des fichiers de données dans des formats spécifiques à certains logiciels comme par exemple `.xlsx` ou `csv` (Excel), `.sav` (SPSS), `.sas7bdat` (SAS), etc. Il existe donc des fonctions très similaires à `read.table` qui sont dans des paquets spécifiques à installer comme par exemple `read.sas7bdat` (paquets `sas7bdat`), `read.csv`, `read.spss` (paquets `foreign`) ou `read.xlsx` (paquets `readxl`). Nous ne les détaillons pas ici, mais vous pouvez trouver toute la documentation nécessaire dans les fichiers d’aide correspondant.

Exercice 0.11. Vérifier que l’objet `electric` apparaît bien dans la fenêtre “Environnement” de RStudio (en haut à droite). Dans cette fenêtre, vous pouvez aussi voir le nombre de lignes et colonnes de cet objet `electric`.

Il y a d'autres fonctions qui peuvent vous donner de l'information sur une base de données, telles que `nrow` (nombre de lignes) ou `ncol` (nombre de colonnes):

```
nrow(electric)
```

```
[1] 120
```

```
ncol(electric)
```

```
[1] 13
```

On peut accéder à une entrée particulière (pour une ligne/ colonne données) d'une base de données en utilisant le symbole `[]`, comme nous l'avons vu pour les matrices:

```
# Affiche l'élément [2,3] (ligne #2 et colonne #3) de la base de données  
electric[2,3]
```

```
[1] "Feb"
```

```
# Affiche la 1re ligne  
electric[1,]
```

```
      NUM YEAR MONTH  BILL TEMP  HDD CDD SIZE METER PUMP1 PUMP2  RIDER  
1     1 1991  Jan 162.1 29.1 1229  0   4     0     0     0 -0.002288  
      CONSUMPTION  
1           5600
```

```
# Affiche la 3e colonne (les cinq premiers éléments seulement)  
head(electric[,3], n = 5L)
```

```
[1] "Jan" "Feb" "Mar" "Apr" "May"
```

Étant donné que les colonnes ont des noms, on peut l'indiquer directement.

```
# 2e colonne (YEAR)  
head(electric$YEAR)
```

```
[1] 1991 1991 1991 1991 1991 1991
```

Jusqu'ici, nous avons seulement affiché les valeurs des données à l'écran, mais ces valeurs n'ont pas été sauvegardées dans **R**. Si on voulait sauvegarder certaines valeurs pour une utilisation future, il faudrait les assigner à un nouvel objet comme suit:

```
# Sauvegarder la 5eme colonne de electric dans l'objet appelé elec5
elec5 <- electric[, 5]
# Affiche le contenu de elec5 (5 premiers elements)
head(elec5, n = 5L)
```

```
[1] 29.1 31.5 41.9 53.4 63.7
```

Il est aussi possible de stocker chaque variable d'un jeu de données dans un objet avec le nom correspondant (par exemple, sauver la variable YEAR dans un objet appelé YEAR). Ceci peut se faire de façon automatique avec la fonction `attach` (pas recommandé, une mauvaise pratique).

Les objets YEAR, BILL, etc. contiennent une série de valeurs. Ces objets sont stockés en tant que vecteurs dans **R**. Nous pouvons donc utiliser toutes les fonctionnalités des vecteurs que nous avons vues plus tôt comme par exemple:

```
# Créer une nouvelle variable YEAR2 qui compte les années de YEAR de 1 à 10
YEAR2 <- electric$YEAR - 1990
head(YEAR2)
```

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

```
# Convertir la température des Fahrenheit en Celsius
TEMP.CELSIUS <- (electric$TEMP - 32) / 1.8
head(TEMP.CELSIUS)
```

```
[1] -1.6111111 -0.2777778 5.5000000 11.8888889 17.6111111 22.7222222
```

Finalement, il est possible de modifier une base de données ou de créer un nouveau base de données à partir d'une autre déjà existante. Voici des exemples:

```
# Créer une base de données appelée "electric" qui contient seulement
# les colonnes YEAR, BILL and CONSUMPTION
electric2 <- with(electric,
  data.frame(YEAR = YEAR,
             BILL = BILL,
             CONSUMPTION = CONSUMPTION)
)
head(electric2)
```

```

YEAR  BILL CONSUMPTION
1 1991 162.10      5600
2 1991 256.90      9463
3 1991 151.15      5154
4 1991 118.76      3576
5 1991 100.71      2894
6 1991  83.97      2257

```

```

# Modifier une base de données existante et y ajouter la colonne
# TEMP.CELSIUS créée plus haut
electric3 <- data.frame(electric,
                        TEMP.CELSIUS = TEMP.CELSIUS)
head(electric3)

```

```

NUM YEAR MONTH  BILL TEMP  HDD CDD SIZE METER PUMP1 PUMP2  RIDER
1   1 1991  Jan 162.10 29.1 1229  0   4    0    0    0 -0.002288
2   2 1991  Feb 256.90 31.5  999  0   4    0    0    0 -0.002288
3   3 1991  Mar 151.15 41.9  734  0   4    0    0    0 -0.002288
4   4 1991  Apr 118.76 53.4  373  0   4    0    0    0 -0.000600
5   5 1991  May 100.71 63.7  162  94   4    0    0    0 -0.000634
6   6 1991  Jun  83.97 72.9    4 211   4    0    0    0 -0.000634
CONSUMPTION TEMP.CELSIUS
1          5600  -1.6111111
2          9463  -0.2777778
3          5154   5.5000000
4          3576  11.8888889
5          2894  17.6111111
6          2257  22.7222222

```

Écrire des rapports avec Quarto

Il existe plusieurs outils permettant d'écrire des rapports en **R** qui incluent à la fois du texte, du code **R**, des sorties **R** et des graphiques (comme le présent document que vous êtes en train de consulter). L'avantage principal de ces outils est que le texte et code **R** sont dans un seul et unique fichier qui peut être compilé dans RStudio, et que les sorties **R** sont générées automatiquement et incluses automatiquement au rapport final. Ceci permet donc d'assurer la **reproductibilité de vos résultats**.

Les deux outils principaux disponibles avec RStudio sont Quarto (plus moderne, compatible avec **R** et Python) et RMarkdown. Voici un exemple de l'interface Quarto, avec l'en-tête, le texte formaté avec Markdown, les blocs de code et la sortie **R**:

The screenshot shows the Quarto editor interface. On the left, the source code for a .qmd file is visible, including a title, format, editor, and R code to load packages and create a plot. On the right, the rendered document is shown, featuring a title, a section 'Meet Quarto', and a section 'Meet the penguins'. The 'Meet the penguins' section includes text about the palmerpenguins package, an illustration of three penguins (Chinstrap, Gentoo, Adelie), and a scatter plot titled 'Flipper and bill length' showing the relationship between flipper and bill lengths for these species.

L'en-tête utilise le langage YAML pour spécifier les options de compilation du document et le type de sortie. Le formatage Markdown permet de définir les sections, etc., mais vous pouvez utiliser le mode d'affichage **Visual** pour obtenir un bloc WYSIWYG. Les blocs de code **R** sont délimités par trois accents graves, suivis de crochets avec la lettre r minuscule.

Pour utiliser Quarto, vous devez [télécharger Quarto](#) depuis le site de Posit.

Nous installerons ensuite quelques paquets nécessaires à la compilation du rapport.

```
# Installer les paquets - ne faire qu'une fois
# (analogie: acheter un livre)
install.packages(c("knitr", "quarto", "rmarkdown"))
# Charger les paquets dans R
# (prendre le livre dans sa bibliothèque)
library(knitr)
```

Vous pouvez trouver sur ZoneCours le fichier .qmd qui a servi à générer le document que vous lisez. Vous pouvez vous baser sur ces exemples de code pour créer vos propres documents. Voici les étapes à suivre:

```

---
title: "Hello, Quarto"
format: html
editor: visual
---

```{r}
#| label: load-packages
#| include: false

library(tidyverse)
library(palmerpenguins)
```

## Meet Quarto

Quarto enables you to weave together content and executable code into a finished document. To learn more about Quarto see <https://quarto.org>.

```

Meet the penguins

The 'penguins' data from the `palmerpenguins` package (https://allisonhorst.github.io/palmerpenguins "palmerpenguins R package") contains size measurements for `r nrow(penguins)` penguins from three species observed on three islands in the Palmer Archipelago, Antarctica.

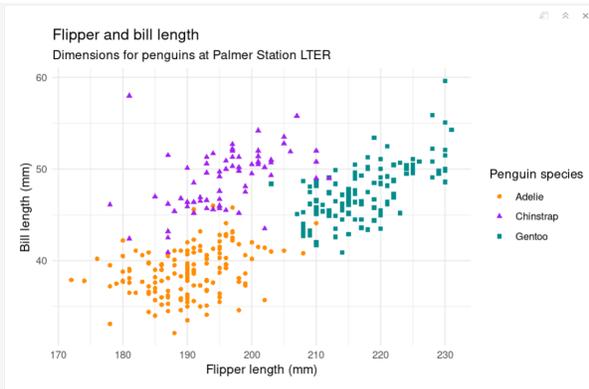
The plot below shows the relationship between flipper and bill lengths of these penguins.

```

```{r}
#| label: plot-penguins
#| warning: false
#| echo: false

ggplot(penguins,
 aes(x = flipper_length_mm, y = bill_length_mm)) +
 geom_point(aes(color = species, shape = species)) +
 scale_color_manual(values = c("darkorange", "purple", "cyan4")) +
 labs(
 title = "Flipper and bill length",
 subtitle = "Dimensions for penguins at Palmer Station LTER",
 x = "Flipper length (mm)", y = "Bill length (mm)",
 color = "Penguin species", shape = "Penguin species"
) +
 theme_minimal()
```

```



quarto Overview Get Started Guide Extensions Reference Gallery Blog Help

Welcome to Quarto

An open-source scientific and technical publishing system

- Author using Jupyter notebooks or with plain text markdown in your favorite editor.
- Create dynamic content with Python, R, Julia, and Observable.
- Publish reproducible, production quality articles, presentations, dashboards, websites, blogs, and books in HTML, PDF, MS Word, ePub, and more.
- Share knowledge and insights organization-wide by publishing to Posit Connect, Confluence, or other publishing systems.
- Write using Pandoc markdown, including equations, citations, crossrefs, figure panels, callouts, advanced layout, and more.

Analyze. Share. Reproduce. You have a story to tell with data—tell it with Quarto.

quarto Overview Get Started Guide Extensions Reference Gallery Blog Help

Get Started

Install Quarto, then check out the tutorials to learn the basics.

Step 1
Install Quarto

Find your operating system in the table below

| Platform | Download | Size | SHA-256 |
|---------------------------|--|-----------|---------|
| Ubuntu 18+/
Debian 10+ | quarto-1.5.56-linux-amd64.deb | 119.21 MB | 75d19bc |
| Linux x86 Tarball | quarto-1.5.56-linux-amd64.tar.gz | 120.55 MB | b2aeff2 |
| Linux Arm64 | quarto-1.5.56-linux-arm64.deb | 122.87 MB | bc4a341 |
| Linux Arm64
Tarball | quarto-1.5.56-linux-arm64.tar.gz | 124.03 MB | 176df3d |

- Ouvrir le fichier `IntroR_part1.qmd` dans RStudio (dans la même fenêtre où vous ouvrez généralement les fichiers `.R`)
- Cliquer sur l'icône **Render** qui est dans la barre d'outils
- Le fichier résultat devrait s'ouvrir (pour les fichiers `html`, cliquez sur `Open in browser` pour voir le rapport final)

Exercice 0.12. Essayez de compiler ce document.

À vous de jouer

À l'aide de **R**, répondez aux questions suivantes:

1. Lire les données `babyboom` dans R.
2. Afficher le nombre de colonnes / lignes des données.
3. Afficher les 6 premières lignes des données.
4. Afficher les deux premières colonnes du jeu de données.
5. Afficher les données du 5e bébé.
6. Créer une nouvelle base de données dans le format `data.frame` qui contient seulement les variables `Time`, `Sex` et `Weight` .
7. Créer une nouvelle variable qui contient le poids en livres du bébé à la naissance, et incorporez cette variable à la base de données