Introduction à R

Visseho Adjiwanou, PhD.

SICSS-Montréal

09 June 2023

Remerciements

Aoudou Mounchingam, Robert Djogbenou, Georges Ngalé, Nima Zahedinameghi ont aidé à développer les labos de cette école.

Cette ère est finie . . .

Au programme

1. Présentation générale de R

- C'est quoi R?
- Avantages
- Ressources

2. Introduction à R

Présentation générale de R

Pourquoi R?

- Logiciel supportant l'enseignement des méthodes quantitatives
- R est une implémentation populaire de S et S-PLUS (commercial) utilisés depuis des décennies
- Le système de base est disponible sur le Comprehensive R Archive Network (CRAN): http://cran.r-project.org

Pourquoi R?

- R est gratuit
- R est un logiciel de très grande qualité
- R est très bien conçu, tant du point de vue de l'analyste de données que de celui de l'informaticien
- Le système de base de R (R base) est complété par plusieurs "packages":
  - "Packages recommandés" qui font partie de la distribution de R standard/base
  - Plus de 400 "packages contribués" sont disponibles gratuitement
  - Ce système (R) est sans doute la ressource la plus complète actuellement disponible pour l'analyse statistique

Pourquoi R?

- R s'exécute sur toutes les plates-formes informatiques couramment utilisées - y compris les systèmes Windows, Unix / Linux et Macintosh - et s'installe normalement sur chaque plate-forme.
  - (S-PLUS, en revanche, n'a pas de version Macintosh.)
- Contrairement aux progiciels statistiques tels que SPSS, qui fournissent des interfaces utilisateur graphiques point-à-clic à la plupart de leurs capacités statistiques, R est orienté commande.

Avantages du logiciel orienté commande

- Il est plus facile de corriger, de modifier et de répliquer des analyses dans un système piloté par des commandes.
- Il est plus simple et naturel de créer un dossier permanent sur son travail, y compris les entrées et les sorties.
- Les interfaces graphiques vers des systèmes statistiques complexes sont incomplètes (nécessitant l'utilisation de commandes pour des analyses plus avancées ou non standard) ou byzantines (obligeant l'utilisateur à parcourir un labyrinthe complexe de menus, de boîtes de dialogue et de contrôles graphiques) - ou les deux.

Avantages du logiciel orienté commande

"Apprendre à utiliser R, par conséquent, est du temps bien dépensé par le sérieux spécialiste des sciences sociales quantitatives"

"Learning to use R, therefore, is time well spent by the serious quantitative social scientist."

(Fox J. & Andersen, R. 2005. Using the R statistical computing environment to teach Social Statistics Courses)

Présentation générale de R

Installation de R et de RStudio

Installation de R

- Pour installer R, aller à https://cran.r-project.org/ (The Comprehensive R Archive Network or CRAN)
- Choisissez votre système d'opération
- Suivez la procédure
- Principal coût de R d'un point de vue pratique est qu'il doit être appris en tant que *langage de programmation*
  - Maîtriser différentes syntaxes et règles de base de la programmation informatique
  - Demande beaucoup de pratique et de patience
  - Processus d'apprentissage frustrant

Utilisation de RStudio

- Pour faciliter l'apprentissage, nous allons utiliser RStudio
- RStudio est un programme open source (libre) et gratuit qui facilite grandement l'utilisation de R.
- Pour obtenir RStudio, visitez le site http://www.rstudio.com/ et suivez les instructions de téléchargement et d'installation

RStudio

RStudio offre aux utilisateurs :

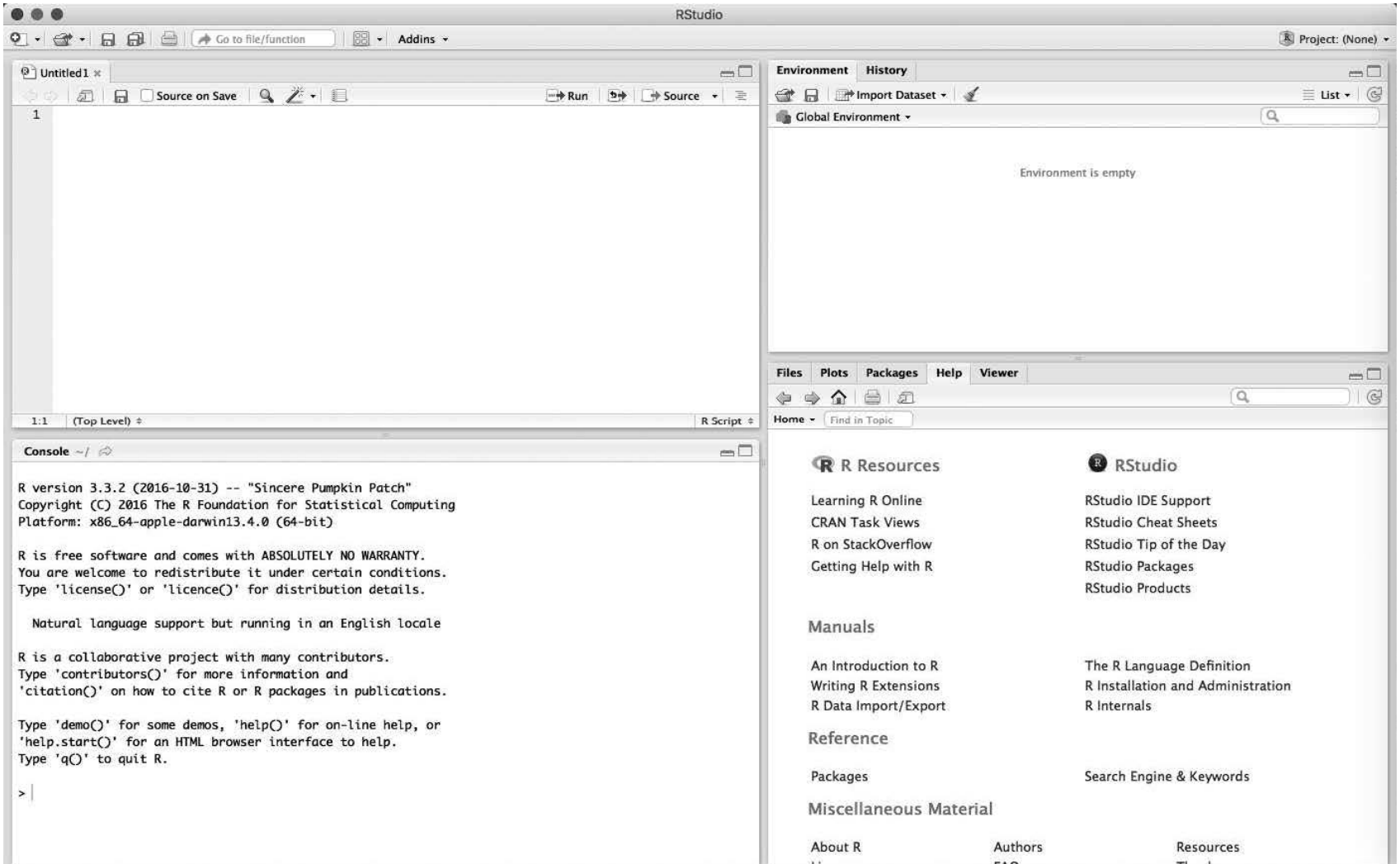
1. Un éditeur de texte pour écrire des programmes,
   - Console –> Calcul interactif, vérification rapide de commandes
   - Script –> Écriture de programme à éxécuter
   - RMarkDown –> Écriture de programme + Texte, moyen de replicabilité
2. Un visualiseur de graphiques qui affiche les graphiques que nous créons,
3. La console R où les programmes sont exécutés,
4. Une section aide, et
5. De nombreuses autres fonctionnalités.

Remarque:

- Cela peut sembler compliqué au premier abord, mais RStudio

RStudio

La figure 1.1 montre une capture d'écran de RStudio.



RStudio

- Description de RStudio:
  https://www.rstudio.com/products/RStudio/
- Pour un résumé des composantes de RStudio:
  https://www.rstudio.com/resources/cheatsheets/

Opérations arithmétiques

Opérations arithmétiques avec la console

- Lancer R Studio *File* > *New File* > *R Script*
- Dans la console, saisissez:

```
5 + 3
```

```
## [1] 8
```

- R ignore les espaces et 5+3 retournera le même résultat.
- Cependant, nous avons ajouté un espace avant et après l'opérateur + pour faciliter la lecture.
- [1] indique que la sortie est le premier élément d'un vecteur de longueur 1

Opérations arithmétiques avec le console

- Votre tour d'essayer: rappelez-vous que nous ne pouvons apprendre la programmation qu'en essayant!
- Autres exemples

```
(5 - 2) * 3
```

```
## [1] 9
```

Opérations arithmétiques avec le console

```
sqrt(4)
```

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

- L'expression finale est un exemple de **fonction**, qui prend une entrée (ou plusieurs entrées) et produit une sortie. Ici, la fonction **sqrt ()** prend un nombre non négatif et renvoie sa racine carrée.
- R possède de nombreuses autres fonctions que nous rencontrerons tout au long de ce cours.
- Vous apprendrez dans ce cours à créer votre propre fonction.

Objets de R

Objets

- R peut stocker des informations en tant qu'objet avec un nom de notre choix.
- Une fois que nous avons créé un objet, nous le référons simplement par son nom.
- En d'autres termes, nous utilisons des objets comme "raccourcis" pour certaines informations ou données.
- Pour cette raison, il est important d'utiliser un **nom intuitif et informatif**.

Objets

- Le nom de notre objet doit respecter certaines restrictions:
  - Ne peut pas commencer par un nombre (mais il peut contenir des nombres).
  - Ne doit pas non plus contenir d'espaces.
  - Doit éviter les caractères spéciaux tels que **%** et **$**, qui ont des significations spécifiques dans R.
- Nous utilisons l'opérateur d'affectation <- pour attribuer une valeur à un objet.
- Dans RStudio, dans la fenêtre supérieure droite, appelée **Environnement**, nous verrons les objets que nous avons créés.

Objets

```
resultat <- (5 - 2) * 3
resultat
```

```
## [1] 9
```

```
print(resultat)
```

```
## [1] 9
```

- Le calcul ci-dessus est stocké en tant qu'objet nommé **resultat**
- Nous pouvons accéder à la valeur en nous référant au nom de l'objet.
- Par défaut, R imprimera la valeur de l'objet sur la console si nous entrons simplement le nom de l'objet et que nous cliquons sur Entrée.
- Alternativement, nous pouvons l'imprimer explicitement en utilisant la fonction **print ()**.

Objets

```
resultat <- 7 - 2
resultat
```

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

- Notez que si nous affectons une valeur différente au même nom d'objet, la valeur de l'objet sera modifiée.
- Par conséquent, nous devons veiller à ne pas écraser les informations précédemment attribuées que nous prévoyons utiliser ultérieurement.

Objets

- Une autre chose à prendre en compte est que les noms d'objet sont sensibles à la casse.
- Par exemple, Bonjour n'est pas le même que bonjour ou bon jour.
- En conséquence, nous recevons une erreur dans la console R lorsque nous tapons **Resultat** plutôt que **resultat** que nous avons défini ci-dessus.

  Resultat

  Error: object 'Resultat' not found

- **Les erreurs de programmation** ou **bogues** font partie du processus d'apprentissage.
- La partie délicate consiste à déterminer comment les réparer. Vous devez faire attention au message d'erreur.
- Ici, le message d'erreur nous dit que l'objet **Resultat** n'existe pas.

Objets

- Nous pouvons voir la liste des objets existants dans l'onglet Environnement.
- Il est également possible d'obtenir la même liste en utilisant la fonction **ls ()**.

```
ls()
```

```
## [1] "resultat"
```

Objets

- Jusqu'à présent, nous n'avons attribué que des chiffres à un objet.
- R peut représenter divers autres types de valeurs en tant qu'objets.
- Par exemple, nous pouvons stocker une chaîne de caractères en utilisant des guillemets.

```
nom <- "Vissého"
nom
```

```
## [1] "Vissého"
```

```
nom <- "Visseho Adjiwanou"
nom
```

```
## [1] "Visseho Adjiwanou"
```

Objets

- Dans les chaînes de caractères, l'espacement est autorisé.
- Notez que R traite les nombres comme des caractères quand on lui dit de le faire.
- Cependant, les opérations arithmétiques telles que l'addition et la soustraction ne peuvent pas être utilisées pour les chaînes de caractères.
- Par exemple, tenter de diviser ou de prendre une racine carrée d'une chaîne de caractères entraînera une erreur.

```
resultat <- "5"
resultat
```

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

Objets

- R reconnaît différents types d'objets en affectant chaque objet à **une classe**.
- La séparation d'objets en classes permet à R d'effectuer des opérations appropriées en fonction de la classe des objets.
- Par exemple, un nombre est stocké en tant qu'objet numérique alors qu'une chaîne de caractères est reconnue en tant qu'objet de caractère.
- Dans RStudio, la fenêtre Environnement affiche la classe d'un objet ainsi que son nom.
- La fonction (qui par ailleurs est une autre classe) **class ()** nous indique à quelle classe appartient un objet.

Objets

```
class(resultat)
```

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

```
class(nom)
```

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

```
class(sqrt)
```

```
## [1] "function"
```

Objet de R

Autres objets:

- Numérique
- Caractères
- Fonction
- Logique

Mode des objets

objet	modes	plusieurs modes possibles dans le même objet ?
vecteur	numérique, caractère, complexe ou logique	Non
facteur	numérique ou caractère	Non
tableau	numérique, caractère, complexe ou logique	Non
matrice	numérique, caractère, complexe ou logique	Non
tableau de données	numérique, caractère, complexe ou logique	Oui
ts	numérique, caractère, complexe ou logique	Non
liste	numérique, caractère, complexe, logique, fonction, expression…	Oui

Vecteurs

Vecteurs

- Vous savez que vous n'êtes pas dans cette classe pour utiliser R comme une calculatrice :)
- Mais, pour analyser les données.
- Voici une première manière mais **non efficiente** d'entrer les données dans R et de les traiter

Vecteurs

Annee	Population Mondiale (en millier)
1950	2,525,779
1960	3,026,003
1970	3,691,173
1980	4,449,049
1990	5,320,817
2000	6,127,700
2010	6,916,183

- Le tableau contient des estimations de la population mondiale (en milliers) au cours des dernières décennies

Vecteurs

- Nous pouvons entrer ces données dans R en tant qu'objet numérique.
- Un vecteur ou un tableau à une dimension représente simplement une collection d'informations stockées dans un ordre spécifique.
- Nous utilisons la fonction **c()**, qui signifie **"concaténer"**, pour entrer un vecteur de données contenant plusieurs valeurs avec des virgules séparant les différents éléments du vecteur que nous créons.
- Par exemple, nous pouvons entrer les estimations de la population mondiale en tant qu'éléments d'un seul vecteur.

```
population <- c(2525779, 3026003, 3691173, 4449049, 5320817

population
```

Vecteurs

- Notons également que la fonction c() peut être utilisée pour combiner plusieurs vecteurs.

```
pop1 <- c(2525779, 3026003, 3691173)
pop2 <- c(4449049, 5320817, 6127700, 6916183)

pop_totale <- c(pop1, pop2)

pop_totale
```

```
## [1] 2525779 3026003 3691173 4449049 5320817 6127700 6916
```

Vecteurs - Information provenant du vecteur

- Pour accéder à des éléments spécifiques d'un vecteur, nous utilisons des crochets **[ ]**.
- Ceci s'appelle **l'indexation**.
- Plusieurs éléments peuvent être extraits via un vecteur d'indices entre crochets.
- Également entre crochets, **le tiret,** -, supprime l'élément correspondant d'un vecteur.
- Notez qu'aucune de ces opérations **ne** modifie le vecteur d'origine.

```
population
```

```
## [1] 2525779 3026003 3691173 4449049 5320817 6127700 6916
```

```
population[2]
```

```
## [1] 3026003
```

Vecteurs - Information provenant du vecteur

```
population[c(1,3)]
```

```
## [1] 2525779 3691173
```

```
population[-2]
```

```
## [1] 2525779 3691173 4449049 5320817 6127700 6916183
```

Vecteurs - Opération sur vecteurs

- Comme chaque élément de ce vecteur est une valeur numérique, nous pouvons lui appliquer des opérations arithmétiques.
- Les opérations seront **répétées** pour chaque élément du vecteur.
- Donnons les estimations de population en millions au lieu de milliers en divisant chaque élément du vecteur par 1000.
- Nous pouvons également exprimer chaque estimation de population en proportion de l'estimation de population de 1950.
  - Rappelons que l'estimation de 1950 est le premier élément du vecteur **population**.

Vecteurs - Opération sur les vecteurs

```
pop_million <- population / 1000
pop_million
```

```
## [1] 2525.779 3026.003 3691.173 4449.049 5320.817 6127.70
```

```
rate <- population / population[1]
rate
```

```
## [1] 1.000000 1.198047 1.461400 1.761456 2.106604 2.42606
```

- Comment interprétez-vous ce résultat?

Vecteurs - Opération sur les vecteurs

```
round(rate, 2)
```

```
## [1] 1.00 1.20 1.46 1.76 2.11 2.43 2.74
```

- Qu'est-ce que la commande **round** permet de faire?

Matrice

Matrices

- Mis en commun de plusieurs vecteurs
- Deux dimensions
  - Nombre de colonnes
  - Nombre de lignes

Matrices

```
population
```

```
## [1] 2525779 3026003 3691173 4449049 5320817 6127700 6916
```

```
annee <- c(1950, 1960, 1970, 1980, 1990, 2000, 2010)
# pib en million de $
pib <- c(5336101, 9424203, 16059180, 19367892, 24639012, 33

Mat_an_pop <- cbind(annee, population, pib)
Mat_an_pop
```

```
##       annee population       pib
## [1,]  1950    2525779   5336101
## [2,]  1960    3026003   9424203
## [3,]  1970    3691173  16059180
## [4,]  1980    4449049  19367892
## [5,]  1990    5320817  24639012
```

Calcul sur une matrice

```r
# Element d'une matrice
Mat_an_pop[3, 2]
```

```
## population
##    3691173
```

```r
# Element d'une ligne
Mat_an_pop[2, ]
```

```
##      annee population        pib
##       1960    3026003    9424203
```

```r
# Élément d'une colonne
Mat_an_pop[, 3]
```

```
## [1]   5336101   9424203  16059180  19367892  24639012 33
```

```r
Mat_an_pop[, "population"]
```

Créer un objet à partir d'une matrice: objet hors de la matrice

```
# Objet hors de la matrice

pop_totale_1 <- Mat_an_pop[1, "population"] + Mat_an_pop[2,
pop_totale_1
```

```
## population
##   32056704
```

Créer un objet à partir d'une matrice: objet hors de la matrice

```
# De manière plus élégante
pop_totale <- sum(Mat_an_pop[ , "population"])
pop_totale
```

```
## [1] 32056704
```

Créer un objet à partir d'une matrice: objet dans la matrice

```
# pib par tête

pib_tete <- Mat_an_pop[, "pib"]/Mat_an_pop[, "population"]*
pib_tete
```

```
## [1]  2112.656  3114.406  4350.698  4353.266  4630.682 54
```

Alors, on a créé un autre vecteur qui donne le pib/tête de chaque décennie. Il va falloir maintenant inclure cela dans la base de données existantes.

```
Mat_an_pop <- cbind(Mat_an_pop, pib_tete)
Mat_an_pop
```

```
##       annee population       pib  pib_tete
## [1,]  1950    2525779   5336101  2112.656
```

Question

Dans quel cas faut-il créer l'objet à l'intérieur et dans quel cas, il faut la créer à l'extérieur?

Exercices

2 EXERCICE 1

En utilisant les informations du tableau sur la population mondiale, calculer le pourcentage d'augmentation de la population pour chaque décennie, défini comme l'augmentation au cours de la décennie divisée par la population de départ. Par exemple, supposons que la population atteignait 100 000 habitants en un an et atteignait 120 000 l'année suivante. Dans ce cas, nous disons «la population a augmenté de 20% = 120000/100000*100».

Réponse 1

```
# A partir d'un vecteur

population
```

```
## [1] 2525779 3026003 3691173 4449049 5320817 6127700 6916
```

```
croissane <- population[-1] / population[-7]
croissane
```

```
## [1] 1.198047 1.219818 1.205321 1.195945 1.151646 1.12867
```

Réponse 2

```
# A partir d'une matrice
Mat_an_pop
```

```
##       annee population       pib  pib_tete
## [1,]  1950    2525779   5336101  2112.656
## [2,]  1960    3026003   9424203  3114.406
## [3,]  1970    3691173  16059180  4350.698
## [4,]  1980    4449049  19367892  4353.266
## [5,]  1990    5320817  24639012  4630.682
## [6,]  2000    6127700 333725635 54461.810
## [7,]  2010    6916183  74004249 10700.158
```

```
croissance1 <- Mat_an_pop[2:7, "population"] / Mat_an_pop[1
croissance1
```

```
## [1] 1.198047 1.219818 1.205321 1.195945 1.151646 1.1286
```

Fichier/base de données

Fichier de données - Type

- Jusqu'à présent, les seules données utilisées ont été entrées manuellement dans R.
- Mais, la plupart du temps, nous chargerons les données à partir d'un fichier externe.
- Différents types de fichiers:
  - Les fichiers **CSV** ou les valeurs séparées par des virgules représentent des données tabulaires. Ce concept est similaire à un tableur contenant des valeurs de données telles que celles générées par Microsoft Excel ou Google Spreadsheet.
  - Chaque observation est séparée par des sauts de ligne et chaque champ de l'observation est séparé par une virgule, une tabulation ou un autre caractère ou une chaîne. Ces fichiers ont l'extention **.csv.**

Fichier de données - Type

- Les fichiers **RData** représentent une collection d'objets R, y compris des ensembles de données.
- Ceux-ci peuvent contenir plusieurs objets R de différents types.
- Ils sont utiles pour enregistrer les résultats intermédiaires de notre code R ainsi que les fichiers de données. Ces fichiers ont l'extention **.RData.**

Fichier de données - Type

- Finalement, vos données peuvent provenir d'autres logiciels statistiques comme SPSS (.sav) ou Stata (.dta).
- Pour travailler avec ces données, vous devez utiliser des **"packages"** spécialisés pour pouvoir les ouvrir.
- Je reviendrai sur l'importance des "packages" à la fin du cours.

Fichier/base de données

- Avant d'interagir avec des fichiers de données, nous devons nous assurer qu'ils résident dans le `répertoire de travail`, par lequel R chargera les données et les enregistrera par défaut.
- Il existe différentes manières de modifier le répertoire de travail.
  - Dans RStudio, le répertoire de travail par défaut est affiché dans la fenêtre en bas à droite sous l'onglet `Files`.
  - Souvent, cependant, le répertoire par défaut n'est pas le répertoire que nous voulons utiliser.
  - Pour modifier le répertoire de travail, nous pouvons utiliser le menu déroulant RStudio **`Session > Set Working Directory > Choose Directory ...`** et choisir le dossier à partir duquel nous souhaitons travailler.

Fichier/base de données - Opération sur les fichiers de données

Les fichiers de données utilisés dans R sont accessibles de trois manières:

1. **A partir du package**

- Tout d'abord, les fichiers de données peuvent être distribués avec les packages R. Ce sont souvent des fichiers de données plus petits, utilisés dans des exemples et des didacticiels dans des packages.
- Ceux-ci sont chargés avec la fonction **data ()**.
- Par exemple, vous pouvez charger des données UN sur des statistiques démographiques à partir du package **qss**, qui distribue les fichiers de données utilisés dans le manuel QSS de Kosuke Imai.
- La fonction **data ()** appelée sans aucun argument listera tous

Fichier/base de données - Opération sur les fichiers de données

```
getwd()
```

```
## [1] "/Users/visseho/Library/CloudStorage/OneDrive-UQAM/S
```

```
library(qss)

# liste de tous les fichiers de données du package "qss"
data(package = "qss")

# Chargement d'un fichier particulier
data("UNpop", package = "qss")
```

Fichier/base de données - Opération sur les fichiers de données

```
head(UNpop)
```

```
##   year world.pop
## 1 1950   2525779
## 2 1960   3026003
## 3 1970   3691173
## 4 1980   4449049
## 5 1990   5320817
## 6 2000   6127700
```

Fichier/base de données - Opération sur les fichiers de données

2 En les entrant manuellement

- Nous avons vu comment créer des vecteurs
- Ces vecteurs peuvent être combinés dans une base de données à partir de la fonction **data_frame**

```
age <- c(24, 25, 37)
nom <- c("Jules", "Olivia", "John")

base <- data.frame(age, nom)
```

- Pourquoi la fonction **cbind** ne fonctionnerait pas?

Fichier/base de données - Opération sur les fichiers de données

2 En les entrant manuellement

- La fonction **rep** est souvent utilisée pour repéter les saisies:

```
age1 <- rep(c(12, 25, 18), 4)
age1
```

```
##  [1] 12 25 18 12 25 18 12 25 18 12 25 18
```

```
age2 <- rep(c(12, 25, 18), each = 4)
age2
```

```
##  [1] 12 12 12 12 25 25 25 25 18 18 18 18
```

- Quelle est la différence?

Fichier/base de données - Opération sur les fichiers de données

2 En les entrant manuellement

- Quand une variable caractère est mise dans une base de données, elle est transformée en facteur, ce qui est toujours plus approprié pour les analyses.

Fichier/base de données - Opération sur les fichiers de données

3. A **partir d'un emplacement donné**

- Deuxièmement, les fichiers de données peuvent être chargés à partir de fichiers externes, y compris des objets R stockés (.RData, .rda) et d'autres formats (.csv, .dta, .sav).
- Pour lire ces fichiers, vous devez connaître leurs types.

Fichier/base de données - Opération sur les fichiers de données

3. A **partir d'un emplacement donné**

- Lecture de fichiers "plain-text"
  - Les fichiers de données en texte brut (plain-text) peuvent être lus à la fois par des personnes et par des programmes informatiques tels que R ou Excel
  - Ils peuvent être ouverts et modifiés dans un éditeur de texte brut, tel que l'éditeur de RStudio

Fichier/base de données - Opération sur les fichiers de données

3. A **partir d'un emplacement donné**

- Nous considérons ici les deux formats les plus usuels:
  - white-space-delimited values (.txt)
  - comma-separated values (.csv)
- Considérez les fichiers de texte brut comme le plus petit dénominateur commun pour stocker des ensembles de données rectangulaires, car la plupart des logiciels statistiques, tableurs et logiciels de base de données peuvent à la fois lire et écrire des fichiers de données texte.

Fichier/base de données - Opération sur les fichiers de données

3. A **partir d'un emplacement donné**

- Fichier txt
  - Le fichier ressemble à une base de données sauf que les données ne sont pas bien arangées
  - La première ligne peut contenir le nom des variables.
  - Chaque ligne subséquent peut définir le nom de la ligne
  - la fonction **read.table** permet de lire ces fichiers

Fichier/base de données - Opération sur les fichiers de données

3. A **partir d'un emplacement donné**

- Fichier txt

3. A **partir d'un emplacement donné**: fichier txt

- Problèmes fréquents lors de la lecture d'un fichier txt
  - les modalités d'une variable caractère ne doivent pas contenir de l'espace
  - les donnés manquantes doivent être spécifiés avec un "NA"
  - Les noms de colonne doivent être des noms valides pour R
  - les nombres ne doivent pas contenir des virgules (,) ou les signes dollars ($) ou pourcentages (%)
  - Confusion entre l (el) et 1 (un) ou entre le o (oh) et 0 (zéro) peuvent changer le format de vos données

Fichier/base de données - Opération sur les fichiers de données

3. A **partir d'un emplacement donné**

- Pour lire un fichier csv dans R, utilisez la fonction read.csv.

```
UNpop_URL <- "https://raw.githubusercontent.com/kosukeimai/
UNpop1 <- read.csv(UNpop_URL)
```

3. A **partir d'un emplacement donné**: autres formats

- Fichier .dta
- Fichier .sas
- Fichier .sps

Chemin pour ouvrir vos fichiers

Exporter et sauvegarder les fichiers de données

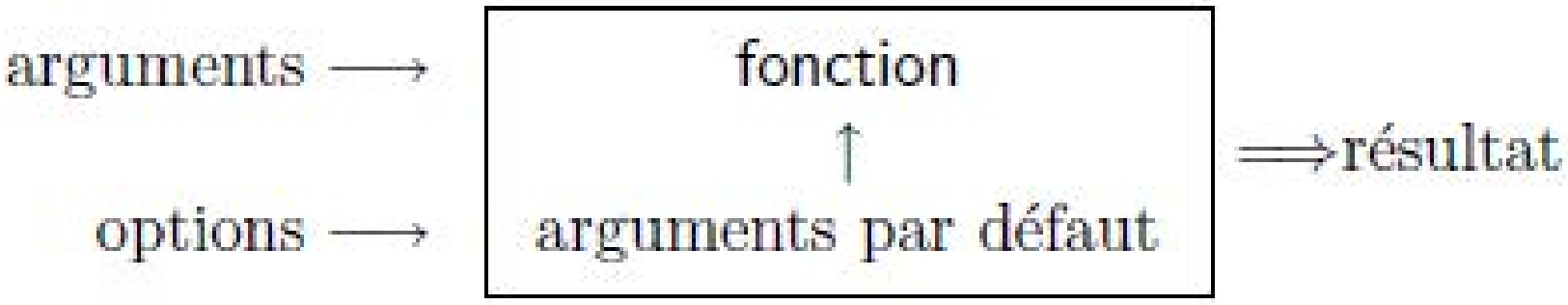
List

Fonctions

Fonctions - Définitions

- Les fonctions sont des objets importants dans R et effectuent un large éventail de tâches.
- Une fonction prend souvent plusieurs objets d'entrée et renvoie un objet de sortie.
- Nous avons déjà vu plusieurs fonctions: **sqrt ()**, **print ()**, **class ()** et **c ()**.
- Dans R, une fonction s'exécute généralement comme **funcname (input)**
  - funcname est le nom de la fonction et
  - input est l'objet d'entrée.
  - En programmation (et en math), nous appelons ces entrées des **arguments**.
- Par exemple, dans la syntaxe **sqrt (4)**, `sqrt` est le nom de la fonction et 4 est l'argument ou l'objet d'entrée.

Fonctions - Définitions



- Les arguments peuvent être des objets (« données », formules, expressions. . .)
- Certains arguments peuvent être définis par défaut dans la fonction ; ces valeurs par défaut peuvent être modifiées par l'utilisateur avec les options.
- Une fonction de R peut ne nécessiter aucun argument de la part de l'utilisateur : soit tous les arguments sont définis par défaut (et peuvent être changés avec les options), ou soit aucun argument n'est défini.

Fonctions - Définitions

```
population
```

```
## [1] 2525779 3026003 3691173 4449049 5320817 6127700 6916
```

```
# Nombre d'éléments dans le vecteur
length(population)
```

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

```
# Le chiffre le plus petit de la série
min(population)
```

```
## [1] 2525779
```

Fonctions - autres exemples

```
# Le chiffre le plus grand de la série
max(population)
```

```
## [1] 6916183
```

```
# L'étendue
range(population)
```

```
## [1] 2525779 6916183
```

```
# Somme
sum(population)
```

```
## [1] 32056704
```

Fonctions - autres exemples

```
# Moyenne
mean(population)
```

```
## [1] 4579529
```

```
# Une autre manière de calculer la moyenne
sum(population) / length(population)
```

```
## [1] 4579529
```

Fonctions - Autres fonctions intéressantes

```
c(1:10)
```

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

```
seq(1:10)
```

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

```
seq(1, 10, 0.5)
```

```
##  [1]  1.0  1.5  2.0  2.5  3.0  3.5  4.0  4.5  5.0  5.5
## [16]  8.5  9.0  9.5 10.0
```

Fonctions - Autres fonctions intéressantes

```
seq(length = 9, from = 1, to = 5)
```

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

```
rep(1, 7)
```

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

Fonctions - Autres fonctions intéressantes

```
sequence(4:7)
```

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

```
gl(2, 5)       # Le premier chiffre est le nombre de niveau
```

```
##  [1] 1 1 1 1 1 2 2 2 2 2
## Levels: 1 2
```

```
gl(2, 6, label=c("Male", "Female"))
```

```
##  [1] Male   Male   Male   Male   Male   Male   Female Fe
## [11] Female Female
## Levels: Male Female
```

- gl pour generate levels

Fonctions - Autres fonctions intéressantes

```
gl(2, 1)
```

```
## [1] 1 2
## Levels: 1 2
```

```
gl(2, 1, length=20)
```

```
##  [1] 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
## Levels: 1 2
```

Fonctions - Créer notre propre fonction

1. Créer une fonction qui donne le carré d'un nombre

```
ca <- function(a){
  carre <- a*a
  print(carre)
}

ca(3)
```

```
## [1] 9
```

```
ca(438)
```

```
## [1] 191844
```

Fonctions - Créer notre propre fonction

2. Créer une fonction qui donne le rapport de deux nombres multiplié par 24

```
r24 <- function(a, b){
  a/b*24
}

r24(4,2)
```

```
## [1] 48
```

```
r24(4,0)
```

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

Les fonctions APPLY

https://bit.ly/2NSpxLI

Remarque importante

- Chaque fonction prend des arguments, mais a presque toujours aussi des options.
- Les options permettent d'aller au-délà de ce qui est fait par défaut.
- **help(nom_fonction)** ou **?nom_fonction** vous permet de connaître ces options et leurs utilisations

```
help(rep)
?rep
```

Les packages

Les packages

- Vous ne pouvez pas créer des fonctions pour chaque code que vous voulez écrire.
- Ces fonctions pour la plupart sont déja écrites par d'autres et stockées dans ce qu'on appelle des "packages"
- Les packages R sont une collection de fonctions R, de code conforme et d'exemples de données.
- Par défaut, R installe un ensemble de packages lors de l'installation.
- D'autres packages sont ajoutés plus tard, lorsqu'ils sont nécessaires à des fins spécifiques: c'est le cas de Tidyverse et de Summarytools
- Il existe un package pour presque tout

Les packages

- Devenir compétent en R demande de savoir écrire ses fonctions mais aussi et surtout de pouvoir utiliser les packages existants.
- Une manière de commencer par travailler facilement avec les nouveaux packages, c'est d'utiliser leur feuille de résumé s'il en existe.
- Ce lien vous renvoie à ces résumés : https://rstudio.com/resources/cheatsheets/

Packages les plus importants

- tidyverse: ensemble de packages (en même temps, une philosophie d'écriture de code)
- summarytools (pour les statistiques descriptives)
- haven, pour la conversion de fichiers . . .

Quels packages utiliser?

- Il y a plus de 10000 packages enregistrés sur CRAN et bien d'autres sur le net.
- Comment choisir le package qui satisfait ses besoins?
- Une recherche dans google avec des mots clés doit vous y guider.
- Je conseille aussi de terminer ces mots clés avec le mot tidyverse pour que la recherche vous sorte des packages qui sont compatibles avec tidyverse
- Une autre solution: utiliser le CRAN task views qui rassemble les packages selon les sujets

CRAN task views

- https://cran.r-project.org/web/views/
- https://codehorizons.com/resources/r-tutorials/