Boucles dans R

Introduction

Au cœur de la programmation se trouve le concept de répéter une tâche plusieurs fois. Une boucle `for` est une des manières fondamentales de le faire. Les boucles permettent une répétition efficace, économisant temps et effort.

Maîtriser ce concept est essentiel pour écrire du code R intelligent et efficace.

Plongeons et améliorez vos compétences en codage !

Objectifs d'apprentissage

À la fin de cette leçon, vous serez capable de :

- Expliquer la syntaxe et la structure d'une boucle `for` basique dans R
- Utiliser des variables d'index pour itérer à travers de multiples vecteurs simultanément dans une boucle
- Intégrer des instructions conditionnelles `if/else` dans une boucle
- Stocker les résultats des boucles dans des vecteurs et des listes
- Appliquer des boucles à des tâches comme l'analyse de multiples jeux de données et la génération de multiples graphiques
- Déboguer des boucles en isolant et en testant des itérations uniques

Packages

Cette leçon nécessitera l'installation et le chargement des packages suivants :

```
# Charger les packages
if(!require(pacman)) install.packages("pacman")
pacman::p_load(tidyverse, here, openxlsx, tools, outbreaks, medicaldata)
```

Introduction aux boucles `for`

Commençons par un exemple simple. Supposons que nous avons un vecteur d'âges d'enfants en années, et nous voulons convertir ces âges en mois :

```
ages <- c(7, 8, 9) # Vecteur d'âges en années
```

Nous pouvons faire cela facilement avec l'opération * dans R :

```
ages * 12
```

```
## [1]  84  96 108
```

Mais voyons en détails la manière dont nous pourrions accomplir cela en utilisant une boucle `for` à la place, car c'est (conceptuellement) ce que R fait en arrière plan.

```
for (age in ages) print(age * 12)
```

```
## [1] 84
## [1] 96
## [1] 108
```

Dans cette boucle, `age` est une variable temporaire qui prend la valeur de chaque élément dans `ages` à chaque itération. D'abord, `age` est 7, ensuite 8, puis 9.

Vous pouvez choisir n'importe quel nom pour cette variable :

```
for (nom_aleatoire in ages) print(nom_aleatoire * 12)
```

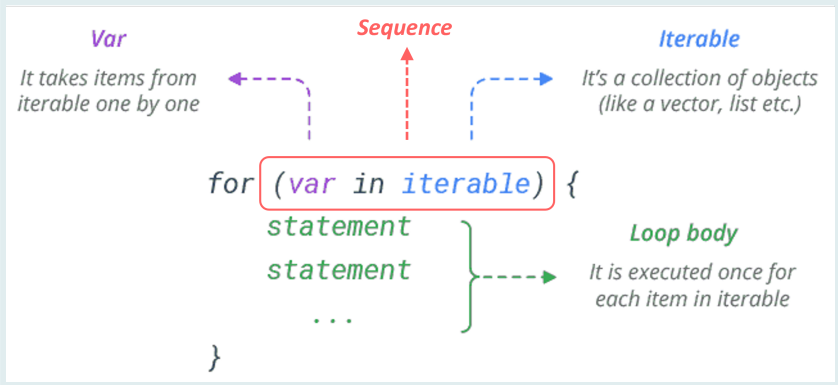
```
## [1] 84
## [1] 96
## [1] 108
```

Si le contenu de la boucle est plus d'une ligne, vous devez utiliser des accolades {} pour indiquer le corps de la boucle.

```
for (age in ages) {
  age_en_mois = age * 12
  print(age_en_mois)
}
```

```
## [1] 84
## [1] 96
## [1] 108
```

La structure générale de n'importe quelle boucle `for` est illustrée dans le diagramme ci-dessous :



Boucle de base pour convertir des heures en minutes

Essayez de convertir des heures en minutes en utilisant une boucle `for`. Commencez avec ce vecteur d'heures :

```
hours <- c(3, 4, 5) # Vecteur d'heures
# Votre code ici

for ___
_____ # convertissez les heures en minutes et affichez
```

Les boucles peuvent être imbriquées les unes dans les autres. Par exemple :

```
for (i in 1:2) {
  for (j in 1:2) {
    print(i * j)
  }
}
```

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

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

Cela crée une combinaison de valeurs `i` et `j` comme le montre ce tableau :

i	j	i * j
1	1	1
1	2	2
2	1	2
2	2	4

Les boucles imbriquées sont cependant moins courantes et ont souvent des alternatives plus efficaces.

Les boucles `for` sont-elles utiles en R ?

Bien que les boucles `for` soient fondamentales dans de nombreux langages de programmation, leur utilisation en R est quelque peu moins fréquente. Cela est dû au fait que R gère intrinsèquement les opérations *vectorisées*, en appliquant automatiquement une fonction à chaque élément d'un vecteur.

Par exemple, notre conversion initiale d'âge pourrait être réalisée sans boucle :

```
ages * 12
```

```
## [1]  84  96 108
```

De plus, R traite généralement avec des jeux de données plutôt qu'avec des vecteurs bruts. Pour les jeux de données, nous utilisons souvent des fonctions du package `tidyverse` pour appliquer des opérations sur les colonnes :

```
ages_df <- tibble(age = ages)
ages_df %>%
  mutate(age_months = age * 12)
```

```
## # A tibble: 3 × 2
##     age age_months
##   <dbl>      <dbl>
## 1     7         84
## 2     8         96
## 3     9        108
```

Cependant, il existe des situations où les boucles sont utiles, en particulier lorsqu'on travaille avec plusieurs jeux de données ou des objets non-dataframe (parfois appelés objets *non rectangulaires*).

Nous explorerons ces cas plus loin dans la leçon, mais d'abord nous allons passer plus de temps à nous familiariser avec les boucles en utilisant des exemples de simulation.

Boucles vs cartographie des fonctions

Il est important de noter que les boucles peuvent souvent être remplacées par des fonctions personnalisées qui sont ensuite appliquées à travers un vecteur ou un jeu de données.

Nous enseignons néanmoins les boucles parce qu'elles sont assez faciles à apprendre, à comprendre et à déboguer, même pour les débutants.

Boucler avec un Index

Il est souvent utile de parcourir un vecteur en utilisant un index, qui est un compteur qui suit l'itération en cours.

Regardons à nouveau notre vecteur `ages` que nous voulons convertir en mois :

```
ages <- c(7, 8, 9) # Vecteur d'âges en années
```

Pour utiliser des index dans une boucle, nous créons d'abord une séquence qui représente chaque position dans le vecteur :

```
1:length(ages) # Crée une séquence d'index de la même longueur que ages
```

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

```
index <- 1:length(ages)
```

Maintenant, `index` a les valeurs 1, 2, 3, correspondant aux positions dans `ages`. Nous utilisons ceci dans une boucle `for` comme suit :

```
for (i in index) {
  print(ages[i] * 12)
}
```

```
## [1] 84
## [1] 96
```

```
## [1] 108
```

Dans ce code, `ages[i]` fait référence au `i`ème élément de notre liste `ages`.

Le nom de la variable `i` est arbitraire. Nous aurions pu utiliser `j` ou `index` ou `position` ou tout autre nom.

```
for (position in index) {
  print(ages[position] * 12)
}
```

```
## [1] 84
## [1] 96
## [1] 108
```

Souvent, nous n'avons pas besoin de créer une variable séparée pour les index. Nous pouvons simplement

Des boucles basées sur les index sont utiles pour travailler avec plusieurs vecteurs en même temps. Nous verrons cela dans la section suivante.

Boucle indexée d'heures en minutes

Réécrivez votre boucle de la dernière question en utilisant des index :

```
heures <- c(3, 4, 5) # Vecteur d'heures

# Votre code ici

for ___ {
  ___
}
```

La fonction `seq_along()` est un raccourci pour créer une séquence d'index. Elle est équivalente à `1:length()` :

```
# Ces deux sont équivalents :
seq_along(ages)
```

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

```
1:length(ages)
```

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

Faire des boucles sur plusieurs vecteurs

Faire des boucles avec des index nous permet de travailler avec plusieurs vecteurs simultanément. Supposons que nous ayons des vecteurs pour les âges et les tailles :

```
ages <- c(7, 8, 9) # âges en années
tailles <- c(120, 130, 140) # tailles en cm
```

Nous pouvons faire des boucles sur les deux en utilisant la méthode d'index :

```
for(i in 1:length(ages)) {
  age <- ages[i]
  taille <- tailles[i]

  print(paste("Âge :", age, "Taille :", taille))
}
```

```
## [1] "Âge : 7 Taille : 120"
## [1] "Âge : 8 Taille : 130"
## [1] "Âge : 9 Taille : 140"
```

À chaque itération : - `i` est l'index - Nous extrayons le i-ème élément de chaque vecteur

et l'imprimons.

Alternativement, nous pouvons sauter l'assignation de variable et utiliser les index directement dans la fonction `print()` :

```
for(i in 1:length(ages)) {
  print(paste("Âge :", ages[i], "Taille :", tailles[i]))
}
```

```
## [1] "Âge : 7 Taille : 120"
## [1] "Âge : 8 Taille : 130"
## [1] "Âge : 9 Taille : 140"
```

Boucle de calcul de l'IMC

En utilisant une boucle for, calculez l'Indice de Masse Corporelle (IMC) des trois individus ci-dessous. La formule de l'IMC est `IMC = poids / (taille ^ 2)`.

```r
poids <- c(30, 32, 35) # Poids en kg
tailles <- c(1.2, 1.3, 1.4) # Tailles en mètres

for(i in ____________________) {


  ___________________________

  print(paste("Poids :", ____,
              "Taille :", ____,
              "IMC :", ____,
              ))

}
```

Stocker les résultats d'une boucle

Dans la plupart des cas, vous voudrez stocker les résultats d'une boucle plutôt que de les imprimer comme nous l'avons fait ci-dessus. Voyons comment faire cela.

Considérons notre exemple de conversion d'âge en mois :

```r
ages <- c(7, 8, 9)

for (age in ages) {
  print(paste(age * 12, "mois"))
}
```

```
## [1] "84 mois"
## [1] "96 mois"
## [1] "108 mois"
```

Pour stocker ces âges convertis, nous créons d'abord un vecteur vide :

```r
ages_mois <- vector(mode = "numeric", length = length(ages))
# Cela peut aussi être écrit comme :
ages_mois <- vector("numeric", length(ages))

ages_mois # Affiche le vecteur vide
```

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

Cela crée un vecteur numérique de la même longueur que ages, initialement rempli de zéros. Pour stocker une valeur dans le vecteur, nous faisons ce qui suit :

```r
ages_mois[1] <- 99 # Stocker 99 dans le premier élément de ages_mois
ages_mois[2] <- 100 # Stocker 100 dans le deuxième élément de ages_mois
ages_mois
```

```
## [1]  99 100   0
```

Maintenant, exécutons la boucle, en stockant les résultats dans `ages_mois` :

```r
ages_mois <- vector("numeric", length(ages))

for (i in 1:length(ages)) {
  ages_mois[i] <- ages[i] * 12
}
ages_mois
```

```
## [1]  84  96 108
```

Dans cette boucle :

- Lors de la première itération, `i` est 1. Nous multiplions le premier élément de `ages` par 12 et le stockons dans le premier élément de `ages_mois`.
- Ensuite `i` est 2, puis 3. À chaque itération, nous multiplions l'élément correspondant de `ages` par 12 et le stockons dans l'élément correspondant de `ages_mois`.

Conversion de la taille de cm en m

Utilisez une boucle for pour convertir les mesures de taille de cm en m. Stockez les résultats dans un vecteur appelé `height_meters`.

```r
height_cm <- c(180, 170, 190, 160, 150) # Tailles en cm

height_m <- vector(_______________) # vecteur numérique de même longueur que
height_cm

for ___ {
  height_m[i] <- ___________________________
}
```

```
## Error: <text>:3:21: unexpected input
## 2:
## 3: height_m <- vector(__
##                       ^
```

Afin de sauvegarder les résultats de votre itération, vous devez créer votre objet vide **à l'extérieur** de la boucle. Sinon, vous ne sauvegarderez que le résultat de la dernière itération.

C'est une erreur commune. Considérez l'exemple ci-dessous :

```
ages <- c(7, 8, 9)

for (i in 1:length(ages)) {
  ages_mois <- vector("numeric", length(ages))
  ages_mois[i] <- ages[i] * 12
}
ages_mois
```

```
## [1]   0   0 108
```

Voyez-vous le problème ?

::: note latérale Si vous êtes pressé, vous pouvez sauter l'utilisation de la fonction `vector()` et initialiser votre vecteur avec `c()` à la place, puis le remplir progressivement avec des valeurs par index :

```
ages_mois <- c()

for (i in 1:length(ages)) {
  ages_mois[i] <- ages[i] * 12
}
ages_mois
```

```
## [1]  84  96 108
```

Et vous pouvez également sauter l'index et utiliser `c()` pour ajouter des valeurs à la fin du vecteur :

```
ages_mois <- c()

for (age in ages) {
  ages_mois <- c(ages_mois, age * 12)
}
ages_mois
```

```
## [1]  84  96 108
```

Cependant, dans les deux cas, R ne connaît pas la longueur finale du vecteur lorsqu'il passe par les itérations, il doit donc réallouer la mémoire à chaque itération. Cela peut entraîner des performances lentes si vous travaillez avec de grands vecteurs. :::

Instructions conditionnelles dans les boucles

Tout comme les instructions `if` peuvent être utilisées dans les fonctions, elles peuvent être intégrées dans les boucles.

Considérez cet exemple :

```
age_vec <- c(2, 12, 17, 24, 60) # Vecteur d'âges

for (age in age_vec) {
  if (age < 18) print(paste("Enfant, Âge", age ))
}
```

```
## [1] "Enfant, Âge 2"
## [1] "Enfant, Âge 12"
## [1] "Enfant, Âge 17"
```

Il est souvent plus clair d'utiliser des accolades pour indiquer le corps de l'instruction `if`. Cela nous permet également d'ajouter plus de lignes de code au corps de l'instruction `if` :

```
for (age in age_vec) {
  if (age < 18) {
    print("Traitement :")
    print(paste("Enfant, Âge", age ))
  }
}
```

```
## [1] "Traitement :"
## [1] "Enfant, Âge 2"
## [1] "Traitement :"
## [1] "Enfant, Âge 12"
## [1] "Traitement :"
## [1] "Enfant, Âge 17"
```

Ajoutons une autre condition pour classer comme 'Enfant' ou 'Adolescent' :

```
for (age in age_vec) {
  if (age < 13) {
    print(paste("Enfant, Âge", age))
  } else if (age >= 13 && age < 18) {
    print(paste("Adolescent, Âge", age))
  }
}
```

```
## [1] "Enfant, Âge 2"
## [1] "Enfant, Âge 12"
```

```
## [1] "Adolescent, Âge 17"
```

Nous pouvons inclure une seule instruction `else` à la fin pour prendre en compte toutes les autres tranches d'âge :

```
for (age in age_vec) {
  if (age < 13) {
    print(paste("Enfant, Âge", age))
  } else if (age >= 13 && age < 18) {
    print(paste("Adolescent, Âge", age))
  } else {
    print(paste("Adulte, Âge", age))
  }
}
```

```
## [1] "Enfant, Âge 2"
## [1] "Enfant, Âge 12"
## [1] "Adolescent, Âge 17"
## [1] "Adulte, Âge 24"
## [1] "Adulte, Âge 60"
```

Pour stocker ces classifications, nous pouvons créer un vecteur vide et utiliser une boucle basée sur l'index pour stocker les résultats :

```
age_class <- vector("character", length(age_vec)) # Créer un vecteur vide
for (i in 1:length(age_vec)) {
  if (age_vec[i] < 13) {
    age_class[i] <- "Enfant" # Enfant
  } else if (age_vec[i] >= 13 && age_vec[i] < 18) {
    age_class[i] <- "Adolescent" # Adolescent
  } else {
    age_class[i] <- "Adulte" # Adulte
  }
}
age_class
```

```
## [1] "Enfant"     "Enfant"     "Adolescent" "Adulte"     "Adulte"
```

Classification de la température

Vous disposez d'un vecteur de températures corporelles en Celsius. Classez chaque température comme 'Hypothermie', 'Normale' ou 'Fièvre' en utilisant une boucle `for` combinée avec des instructions `if` et `else`.

Utilisez ces règles :

- En dessous de 36,5 °C : 'Hypothermie' (Hypothermie)
- Entre 36,5 °C et 37,5 °C : 'Normale' (Normale)

- Au-dessus de 37,5 °C : 'Fièvre' (Fièvre)

```
body_temps <- c(35, 36.5, 37, 38, 39.5) # Températures corporelles en Celsius
classif_vec <- vector(_____________________) # vec caractère, longueur de
body_temps
for (i in 1:length(________)) {
    # Ajoutez votre logique if-else ici
  if (tempe_corporelle[i] < 36.5) {
    out <- "Hypothermie" # Hypothermie
  } ## ajoutez d'autres conditions


    # Dernière instruction d'impression
    classif_vec[i] <- paste(tempe_corporelle[i], "°C est", out)
}
classif_vec
```

```
## Error: <text>:2:24: unexpected input
## 1: body_temps <- c(35, 36.5, 37, 38, 39.5) # Températures corporelles en
Celsius
## 2: classif_vec <- vector(__
##                          ^
```

Un résultat attendu est ci-dessous

```
35°C est Hypothermie
36.5°C est Normale
37°C est Normale
38°C est Fièvre
39.5°C est Fièvre
```

Techniques rapides pour déboguer des boucles `for`

L'édition et le débogage efficaces sont cruciaux lorsqu'on travaille avec des boucles `for` en R. Il existe de nombreuses approches pour cela, mais pour le moment, nous allons montrer deux des plus simples :

- Isoler et exécuter une seule itération de la boucle
- Ajouter des instructions `print()` à la boucle pour imprimer les valeurs des variables à chaque itération

Isoler et exécuter une seule itération

Considérez cette boucle que nous avons vue précédemment :

```r
age_vec <- c(2, 12, 17, 24, 60) # Vecteur d'âges
age_class <- vector("character", length(age_vec))

for (i in 1:length(age_vec)) {
  if (age_vec[i] < 18) {
    age_class[i] <- "Enfant" # Enfant
  } else {
    age_class[i] <- "Adulte" # Adulte
  }
}
age_class
```

```
## [1] "Enfant" "Enfant" "Enfant" "Adulte" "Adulte"
```

Voyons un exemple d'erreur que nous pourrions rencontrer en utilisant la boucle :

```r
# Vecteur d'âge issu du jeu de données fluH7N9_china_2013
flu_dat <- outbreaks::fluH7N9_china_2013
head(flu_dat)
```

```
##   case_id date_of_onset date_of_hospitalisation date_of_outcome outcome
gender age province
## 1       1    2013-02-19                   <NA>      2013-03-04   Death
m  87 Shanghai
## 2       2    2013-02-27             2013-03-03      2013-03-10   Death
m  27 Shanghai
## 3       3    2013-03-09             2013-03-19      2013-04-09   Death
f  35    Anhui
## 4       4    2013-03-19             2013-03-27            <NA>    <NA>
f  45  Jiangsu
## 5       5    2013-03-19             2013-03-30      2013-05-15 Recover
f  48  Jiangsu
## 6       6    2013-03-21             2013-03-28      2013-04-26   Death
f  32  Jiangsu
```

```r
flu_dat_age <- flu_dat$age
age_class <- vector("character", length(flu_dat_age))
for (i in 1:length(flu_dat_age)) {
  if (flu_dat_age[i] < 18) {
    age_class[i] <- "Enfant" # Enfant
  } else {
    age_class[i] <- "Adulte" # Adulte
  }
}
```

```
## Warning in Ops.factor(flu_dat_age[i], 18): '<' not meaningful for factors
```

```
## Error in if (flu_dat_age[i] < 18) {: missing value where TRUE/FALSE needed
```

Nous obtenons cette erreur :

```
Erreur dans if (flu_dat_age[i] < 18) { :
  valeur manquante là où TRUE / FALSE est requis
De plus : Message d'avis :
Dans Ops.factor(flu_dat_age[i], 18) :
  ‘<’ n'est pas pertinent pour des variables facteurs
```

Vous savez peut-être déjà ce que signifie cette erreur, mais supposons que vous ne le sachiez pas.

Nous pouvons entrer dans la boucle et parcourir manuellement la première itération pour voir ce qui se passe :

```
for (i in 1:length(flu_dat_age)) {

  # ▶ Exécutez à partir de cette ligne
  i <- 1 # Définir manuellement i à 1

  # Ensuite, surlignez `flu_dat_age[i]` et appuyez sur Ctrl + Entrée pour
exécuter juste ce code
  # Après cela, surlignez et exécutez `flu_dat_age[i] < 18`

  if (flu_dat_age[i] < 18) {
    age_class[i] <- "Enfant" # Enfant
  } else {
    age_class[i] <- "Adulte" # Adulte
  }

}
```

```
## Warning in Ops.factor(flu_dat_age[i], 18): '<' not meaningful for factors
```

```
## Error in if (flu_dat_age[i] < 18) {: missing value where TRUE/FALSE needed
```

En suivant le processus ci-dessus, nous pouvons voir que `flu_dat_age` est un facteur, et non un vecteur numérique. Nous pouvons manuellement changer cela, en plein milieu du processus de débogage. C'est une bonne idée de convertir d'abord le facteur en vecteur de caractères, puis en vecteur numérique. Sinon, nous pourrions obtenir des résultats inattendus.

Considérez :

```
flu_dat_age[75]
```

```
## [1] ?
## 61 Levels: ? 15 2 21 25 26 27 31 32 34 35 36 37 38 4 41 43 45 47 48 49 50
51 52 53 54 55 56 57 58 59 6 60 ... 91
```

```r
as.numeric(flu_dat_age[75])
```

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

```r
# `?`, qui signifie manquant dans ce cas, est converti en 1, car il est le
premier niveau du facteur

# Nous avons donc besoin :
as.numeric(as.character(flu_dat_age[75]))
```

```
## Warning: NAs introduced by coercion
```

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

Maintenant essayons de corriger la boucle, et exécutons juste la première itération à nouveau :

```r
for (i in 1:length(flu_dat_age)) {

  # ▶ Exécuter à partir de cette ligne
  i <- 1 # Fixer manuellement i à 1

  age_num <- as.numeric(as.character(flu_dat_age[i]))

  # Ensuite, surligner `age_num < 18` et appuyer sur Ctrl + Entrée
  if (age_num < 18) {
    age_class[i] <- "Enfant"
  } else {
    age_class[i] <- "Adulte"
  }

}
```

Maintenant, la première itération fonctionne, mais voyons ce qui se passe lorsque nous exécutons la boucle entière :

```
age_class <- vector("character", length(flu_dat_age))

for (i in 1:length(flu_dat_age)) {
  age_num <- as.numeric(as.character(flu_dat_age[i]))

  if (age_num < 18) {
    age_class[i] <- "Enfant"
  } else {
    age_class[i] <- "Adulte"
  }
}
head(age_class)
```

```
 Erreur dans if (age_num < 18) { :
   valeur manquante là où TRUE / FALSE est requis
 De plus : Message d'avis :
 Dans as.numeric(as.character(flu_dat_age[i])) :
   NAs introduits lors de la conversion automatique
```

Encore une fois, vous savez peut-être déjà ce que signifie cette erreur, mais supposons que non. Nous allons essayer notre prochaine technique de débogage.

Ajouter des Instructions d'Impression à la Boucle

Dans la dernière section, nous avons vu que la boucle fonctionne bien pour la première itération, mais semble échouer lors d'une itération ultérieure.

Pour détecter sur quelle itération elle échoue, nous pouvons ajouter des instructions `print()` à la boucle :

```
for (i in 1:length(flu_dat_age)) {

  print(i) # Imprimer le numéro de l'itération
  age_num <- as.numeric(as.character(flu_dat_age[i]))

  print(age_num) # Imprimer la valeur de age_num

  if (age_num < 18) {
    age_class[i] <- "Enfant"
  } else {
    age_class[i] <- "Adulte"
  }

  print(age_class[i]) # Imprimer la valeur de la sortie
}
head(age_class)
```

Maintenant, en inspectant la sortie, nous pouvons voir que la boucle échoue à la 74ème itération :

```
[1] 73 👈 73ème itération
[1] 43 👈 Valeur de age_num
[1] "Adulte, Âge 43" 👈 Valeur de la sortie à la 73ème itération
[1] 74 👈 74ème itération
[1] NA 👈 Valeur de age_num
```

Cela se produit parce que la 74ème valeur de `flu_dat_age` est NA (à cause de notre conversion de facteur en numérique), donc R ne peut pas évaluer si elle est inférieure à 18.

Nous pouvons corriger cela en ajoutant une instruction `if` pour vérifier les valeurs NA :

```
for (i in 1:length(flu_dat_age)) {

  age_num <- as.numeric(as.character(flu_dat_age[i]))

  if (is.na(age_num)) {
    age_class[i] <- "NA"
  } else if (age_num < 18) {
    age_class[i] <- "Enfant"
  } else {
    age_class[i] <- "Adulte"
  }
}
```

```
## Warning: NAs introduced by coercion

## Warning: NAs introduced by coercion
```

```
# Vérifier la 74ème valeur de age_class
age_class[74]
```

```
## [1] "NA"
```

Super ! Maintenant, nous avons corrigé l'erreur.

Comme vous pouvez le voir, même avec notre boucle "jouet", le débogage peut être un processus long. Comme le disait votre mère : "La programmation, c'est 98 % de débogage et 2 % d'écriture de code."

R offre plusieurs autres techniques pour diagnostiquer et gérer les erreurs :

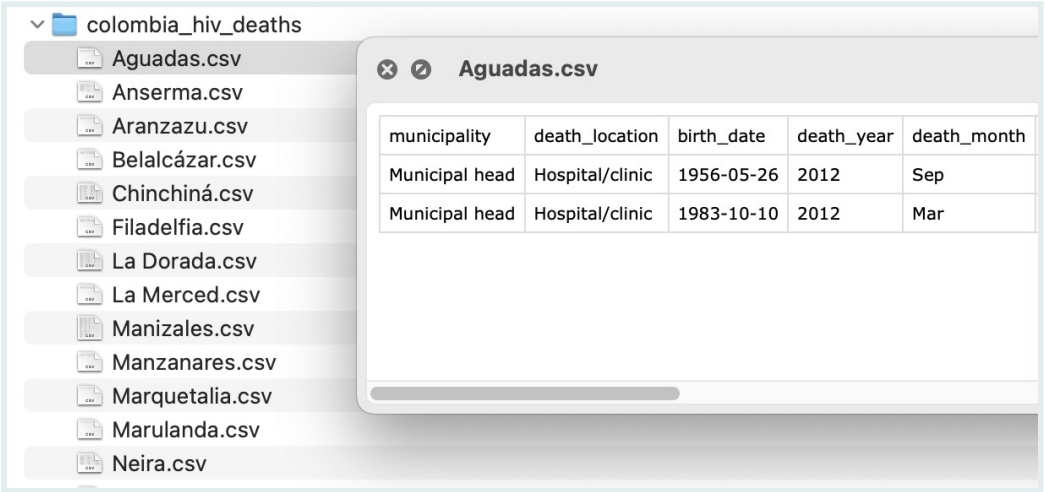
- Les fonctions `try()` et `tryCatch()` permettent de capturer les erreurs tout en continuant l'exécution de la boucle.
- La fonction `browser()` met la boucle en pause à un point désigné, permettant une exécution pas à pas.

Ce sont des méthodes plus avancées, et bien quelles ne soient pas couvertes ici, vous pouvez vous référer à la documentation de R pour des conseils supplémentaires lorsque cela est nécessaire. Ou consultez le livre Advanced R de Hadley Wickham.

Application Réelle des Boucles 1 : Analyser Plusieurs Jeux de Données

Maintenant que vous avez une solide compréhension des boucles `for`, appliquons nos connaissances à une tâche de boucle plus réaliste : travailler avec plusieurs jeux de données.

Nous avons un dossier de fichiers CSV contenant des données sur les décès liés au VIH pour les municipalités de Colombie.



Imaginez que nous devions compiler un seul tableau comprenant les informations suivantes pour chaque jeu de données : le nombre de lignes (nombre de décès), le nombre de colonnes et les noms de toutes les colonnes.

Nous pourrions faire cela un par un, mais cela serait fastidieux et source d'erreurs. Au lieu de cela, nous pouvons utiliser une boucle pour automatiser le processus.

Tout d'abord, listons les fichiers dans le dossier :

```
chemins_donnees_colom <- list.files(here("data/colombia_hiv_deaths"),
                                    full.names = TRUE)
head(chemins_donnees_colom) # Montrer les 6 premiers chemins de fichiers
```

```
## [1]
"/Users/kendavidn/Dropbox/tgc_github_projects/fdar_staging/FDAR_FR_loops/data/c-
```

```
olombia_hiv_deaths/Aguadas.csv"
## [2]
"/Users/kendavidn/Dropbox/tgc_github_projects/fdar_staging/FDAR_FR_loops/data/colombia_hi
## [3]
"/Users/kendavidn/Dropbox/tgc_github_projects/fdar_staging/FDAR_FR_loops/data/colombia_hi
## [4]
"/Users/kendavidn/Dropbox/tgc_github_projects/fdar_staging/FDAR_FR_loops/data/colombia_hi
## [5]
"/Users/kendavidn/Dropbox/tgc_github_projects/fdar_staging/FDAR_FR_loops/data/colombia_hi
## [6]
"/Users/kendavidn/Dropbox/tgc_github_projects/fdar_staging/FDAR_FR_loops/data/colombia_hi
```

Maintenant, importons un jeu de données comme exemple pour démontrer ce que nous voulons atteindre. Une fois cela fait, nous pourrons appliquer le même processus à tous les jeux de données.

```
donnees_colom <- read_csv(chemins_donnees_colom[1]) # Importer le premier jeu
de données
```

```
## Rows: 2 Columns: 15
## ── Column specification
──────────────────────────────────────────────────────────────────────────────
## Delimiter: ","
## chr  (9): municipality, death_location, death_month, municipality_code,
primary_cause_death_description, prim...
## dbl  (2): death_year, death_day
## lgl  (3): tertiary_cause_death_description,
quaternary_cause_death_description, quaternary_cause_death_code
## date (1): birth_date
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this
message.
```

```
donnees_colom
```

```
## # A tibble: 2 × 15
##   municipality   death_location  birth_date death_year
##   <chr>          <chr>           <date>          <dbl>
## 1 Municipal head Hospital/clinic 1956-05-26       2012
## 2 Municipal head Hospital/clinic 1983-10-10       2012
## # ℹ 11 more variables: death_month <chr>, death_day <dbl>,
## #   municipality_code <chr>, …
```

Ensuite, nous appliquons une série de fonctions R pour recueillir les informations que nous voulons de chaque jeu de données :

```
file_path_sans_ext(basename(chemins_donnees_colom[1]))  # Nom du jeu de
données/municipalité
```

```
## [1] "Aguadas"
```

```
nrow(donnees_colom) # Nombre de lignes, ce qui équivaut au nombre de décès
```

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

```
ncol(donnees_colom) # Nombre de colonnes
```

```
## [1] 15
```

```
paste(names(donnees_colom), collapse = ", ") # Noms de toutes les colonnes
```

```
## [1] "municipality, death_location, birth_date, death_year, death_month,
death_day, municipality_code, primary_cause_death_description,
primary_cause_death_code, secondary_cause_death_description,
secondary_cause_death_code, tertiary_cause_death_description,
tertiary_cause_death_code, quaternary_cause_death_description,
quaternary_cause_death_code"
```

basename : extrait le nom de fichier d'un chemin de fichier.

```
chemins_donnees_colom[1]
```

```
## [1]
"/Users/kendavidn/Dropbox/tgc_github_projects/fdar_staging/FDAR_FR_loops/data/colombia_hi
```

```
basename(chemins_donnees_colom[1])
```

```
## [1] "Aguadas.csv"
```

Et `file_path_sans_ext` du package {tools} supprime l'extension de fichier des noms de fichiers. Nous l'utilisons avec basename pour obtenir le nom de la municipalité.

```
file_path_sans_ext(basename(chemins_donnees_colom[1]))
```

```
## [1] "Aguadas"
```

Maintenant, nous devons créer un dataframe avec ces informations. Nous pouvons utiliser la fonction `tibble` pour faire cela :

```
ligne_unique <-
  tibble(jeu_de_donnees = basename(chemins_donnees_colom[1]),
         n_deces = nrow(donnees_colom),
         n_colonnes = ncol(donnees_colom),
         noms_colonnes = paste(names(donnees_colom), collapse = ", "))
ligne_unique
```

```
## # A tibble: 1 × 4
##   jeu_de_donnees n_deces n_colonnes noms_colonnes
##   <chr>            <int>      <int> <chr>
## 1 Aguadas.csv          2         15 municipality, death_loc…
```

Nous allons donc devoir répéter ce processus pour chaque jeu de données. Dans la boucle, nous stockerons chaque dataframe à une seule ligne dans une liste, puis les combinerons à la fin. Rappelons que les listes sont des objets R qui peuvent contenir d'autres objets R, y compris des dataframes.

Initialisons cette liste vide maintenant :

```
liste_dataframes <- vector("list", length(chemins_donnees_colom))
head(liste_dataframes) # Montrer les 6 premiers éléments
```

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

Ajoutons la première dataframe à une seule ligne à la liste :

```
liste_dataframes[[1]] <- ligne_unique
```

Maintenant, si nous regardons la liste, nous voyons que le premier élément est la dataframe à une seule ligne :

```
head(liste_dataframes)
```

```
## [[1]]
## # A tibble: 1 × 4
##   jeu_de_donnees n_deces n_colonnes noms_colonnes
##   <chr>            <int>      <int> <chr>
## 1 Aguadas.csv          2         15 municipality, death_loc…
##
## [[2]]
## NULL
##
## [[3]]
## NULL
##
## [[4]]
## NULL
##
## [[5]]
## NULL
##
## [[6]]
## NULL
```

Et nous pouvons accéder à la dataframe en sous-ensemble de la liste :

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

```
## # A tibble: 1 × 4
##   jeu_de_donnees n_deces n_colonnes noms_colonnes
##   <chr>            <int>      <int> <chr>
## 1 Aguadas.csv          2         15 municipality, death_loc…
```

Notez l'utilisation de doubles crochets pour accéder aux éléments de la liste.

Nous avons maintenant toutes les pièces dont nous avons besoin pour créer une boucle qui traitera chaque jeu de données et stockera les résultats dans une liste. Allons-y !

```
for (i in 1:length(chemins_donnees_colom)) {
    chemin <- chemins_donnees_colom[i]

    # Importation
    donnees_colom <- read_csv(chemin)

    # Récupération des infos
    n_deces <- nrow(donnees_colom)
    n_colonnes <- ncol(donnees_colom)
    noms_colonnes <- paste(names(donnees_colom), collapse = ", ")

    # Création du dataframe pour cet ensemble de données
    ligne_donnees_vih <- tibble(jjeu_de_donnees =
file_path_sans_ext(basename(chemin)),
                            n_deces = n_deces,
                            n_colonnes = n_colonnes,
                            noms_colonnes = noms_colonnes)

    # Stockage dans la liste
    liste_dataframes[[i]] <- ligne_donnees_vih
}
```

Vérifions la liste :

```
head(liste_dataframes, 2) # Montrer les 2 premiers éléments
```

```
## [[1]]
## # A tibble: 1 × 4
##   jjeu_de_donnees n_deces n_colonnes noms_colonnes
##   <chr>              <int>      <int> <chr>
## 1 Aguadas                2         15 municipality, death_lo…
##
## [[2]]
## # A tibble: 1 × 4
##   jjeu_de_donnees n_deces n_colonnes noms_colonnes
##   <chr>              <int>      <int> <chr>
## 1 Anserma               15         16 municipality, death_lo…
```

Et maintenant, nous pouvons combiner tous les dataframes de la liste en un seul dataframe final. Cela peut être fait avec la fonction `bind_rows` du package {dplyr} :

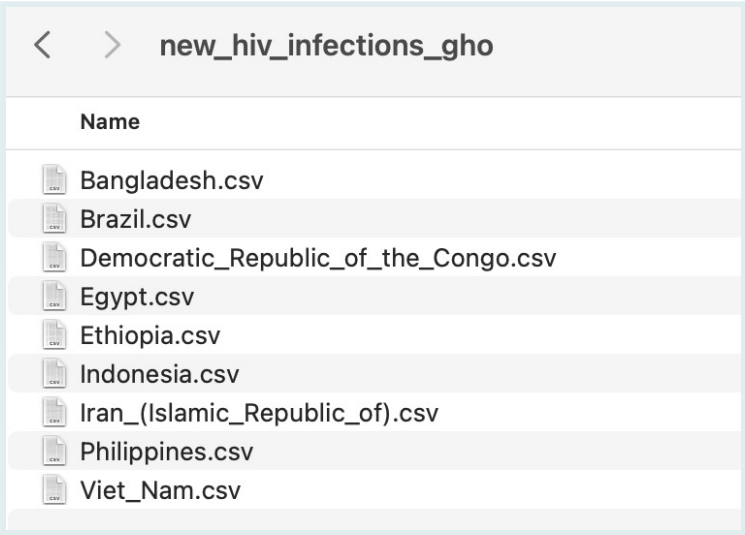
```
donnees_colom_finales <- bind_rows(liste_dataframes)
donnees_colom_finales
```

```
## # A tibble: 25 × 4
##    jjeu_de_donnees n_deces n_colonnes noms_colonnes
##    <chr>              <int>      <int> <chr>
##  1 Aguadas                2         15 municipality, death_l…
##  2 Anserma               15         16 municipality, death_l…
##  3 Aranzazu               2         16 municipality, death_l…
##  4 Belalcázar             4         14 municipality, death_l…
```

```
##  5 Chinchiná            62         17 municipality, death_l…
##  6 Filadelfia            5         15 municipality, death_l…
##  7 La Dorada            46         16 municipality, death_l…
##  8 La Merced             3         17 municipality, death_l…
##  9 Manizales           199         17 municipality, death_l…
## 10 Manzanares            3         14 municipality, death_l…
## # ℹ 15 more rows
```

Propriétés du fichier

Vous avez un dossier contenant des fichiers CSV avec des données sur les cas de VIH, provenant de l'OMS.



En utilisant les principes appris, vous écrirez une boucle qui extrait les informations suivantes de chaque jeu de données et les stocke dans un seul dataframe :

- Le nom du jeu de données (c.-à-d. le pays)
- La taille du jeu de données en octets
- La date de dernière modification du jeu de données

Vous pouvez utiliser les fonctions `file.size()` et `file.mtime()` pour obtenir les deux dernières informations. Par exemple :

```
file.size(here("data/new_hiv_infections_gho/Bangladesh.csv"))
```

```
## [1] 6042
```

```
file.mtime(here("data/new_hiv_infections_gho/Bangladesh.csv"))
```

```
## [1] "2023-12-11 17:34:28 GMT"
```

Notez que vous n'avez pas besoin d'importer les CSV pour obtenir ces informations.

```
# Lister les fichiers
csv_files <- list.files(path = "data/new_hiv_infections_gho",
                        _____________________________________________)

for (i in _________________) {

  path <- csv_files[i]

  # Obtenir le nom du pays. Conseil : utilisez file_path_sans_ext et basename
  nom_pays <- _____________________________________

  # Obtenir la taille du fichier et la date de modification
  taille <- _____________________________________________
  date <- _____________________________________________

  # Dataframe pour cette itération. Conseil : utilisez tibble() pour combiner
les objets ci-dessus
  vih_ligne_df <- _____________________________________________

  # Stocker dans la liste. Conseil : utilisez des doubles crochets et l'index
i
  liste_dataframes_____ <- vih_ligne_df
}

# Combiner en un seul dataframe
infos_fichiers_vih_final <- bind_rows(liste_dataframes)
```

Boucle de Filtrage des Données

Vous travaillerez à nouveau avec le dossier des jeux de données sur le VIH de la question précédente. Voici un exemple d'un des jeux de données par pays de ce dossier :

```
donnees_bangla <- read_csv(here("data/new_hiv_infections_gho/Bangladesh.csv"))
donnees_bangla
```

```
## # A tibble: 89 × 5
##    Continent       Country     Year Sex
##    <chr>           <chr>      <dbl> <chr>
##  1 South-East Asia Bangladesh  2022 Female
##  2 South-East Asia Bangladesh  2022 Both sexes
##  3 South-East Asia Bangladesh  2022 Male
##  4 South-East Asia Bangladesh  2021 Female
##  5 South-East Asia Bangladesh  2021 Both sexes
##  6 South-East Asia Bangladesh  2021 Male
##  7 South-East Asia Bangladesh  2020 Female
##  8 South-East Asia Bangladesh  2020 Both sexes
##  9 South-East Asia Bangladesh  2020 Male
## 10 South-East Asia Bangladesh  2019 Female
```

```
## # ℹ 79 more rows
## # ℹ 1 more variable: NewHIVCases <chr>
```

Votre tâche est de compléter le modèle de boucle ci-dessous afin qu'il : - Importe chaque CSV du dossier - Filtre les données uniquement au sexe "Féminin" - Sauvegarde chaque jeu de données filtré comme un CSV dans votre dossier `sorties`

Notez que dans ce cas, vous n'avez pas besoin de stocker les sorties dans une liste, car vous importez, modifiez puis exportez directement chaque jeu de données.

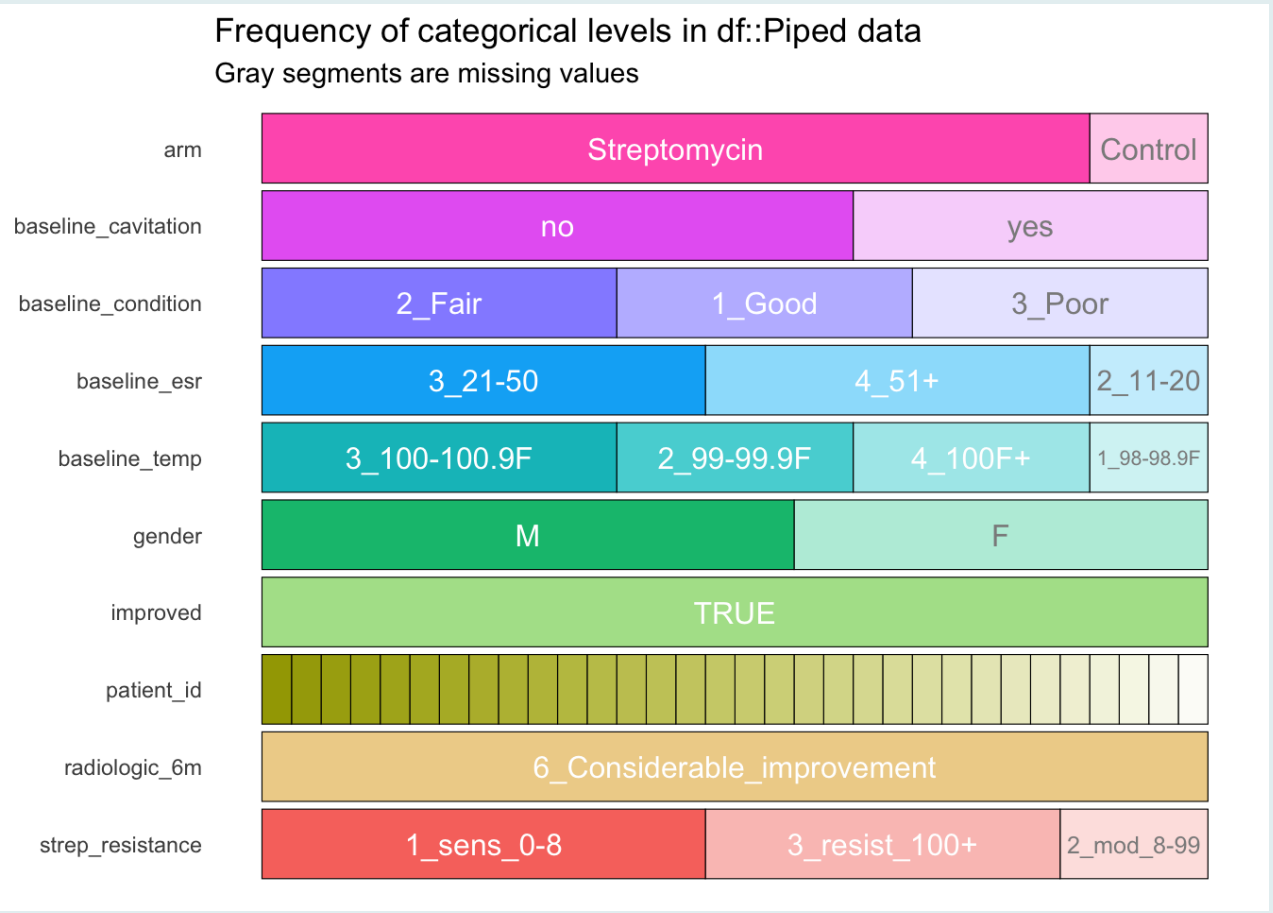
```
# Lister les fichiers
csv_files <- list.files(path = "data/new_hiv_infections_gho",
                        pattern = "*.csv", full.names = TRUE)

for (fichier in _____________) {

  # Importer les données. Conseil : utilisez read_csv avec la variable
`fichier` comme chemin
  donnees_vih ____________________________________________

  # Filtrer. Conseil : utilisez filter() et la variable `Sexe`
  donnees_vih_filtrees <- ____________________________________________

  # Nommer le fichier de sortie
  # Cette ligne est faite pour vous, mais assurez-vous de la comprendre
  nom_fichier_sortie <-  paste0(here(), "sorties/", "Feminin_",
basename(fichier))

  # Exporter.
  write_csv(donnees_vih_filtrees, nom_fichier_sortie)
}
```

~~Application~~n Réelle des Boucles 2 : Génération de Plusieurs Graphiques

Une autre application courante des boucles est la génération de multiples graphiques pour différents groupes au sein d'un jeu de données. Nous utiliserons le jeu de données `strep_tb` du package `medicaldata` pour illustrer cela. Notre objectif est de créer des graphiques d'inspection des catégories pour chaque groupe d'amélioration radiologique à 6 mois.

Commençons par créer un graphique pour l'un des groupes. Nous utiliserons `inspectdf::inspect_cat()` pour générer un graphique d'inspection des catégories :

```r
cat_plot <-
  medicaldata::strep_tb %>%
  filter(radiologic_6m == "6_Considerable_improvement") %>%
  inspectdf::inspect_cat() %>%
  inspectdf::show_plot()
cat_plot
```



Ce graphique nous offre un moyen rapide de visualiser la distribution des catégories dans notre jeu de données.

Maintenant, nous voulons créer des graphiques similaires pour chaque groupe d'amélioration radiologique dans le jeu de données. D'abord, identifions tous les groupes uniques en utilisant la fonction `unique` :

```r
niveaux_radiologiques_6m <- medicaldata::strep_tb$radiologic_6m %>% unique()
niveaux_radiologiques_6m
```

```
## [1] 6_Considerable_improvement   5_Moderate_improvement       4_No_change
## [4] 3_Moderate_deterioration     2_Considerable_deterioration 1_Death
## 6 Levels: 6_Considerable_improvement 5_Moderate_improvement 4_No_change
... 1_Death
```

Ensuite, initialisons un objet liste vide où nous stockerons les graphiques.

```r
liste_graphiques_cat <- vector("list", length(niveaux_radiologiques_6m))
liste_graphiques_cat
```

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

Nous allons également définir les noms des éléments de la liste pour les groupes d'amélioration radiologique. C'est une étape facultative, mais cela facilite l'accès aux graphiques spécifiques plus tard.

```r
names(liste_graphiques_cat) <- niveaux_radiologiques_6m
liste_graphiques_cat
```

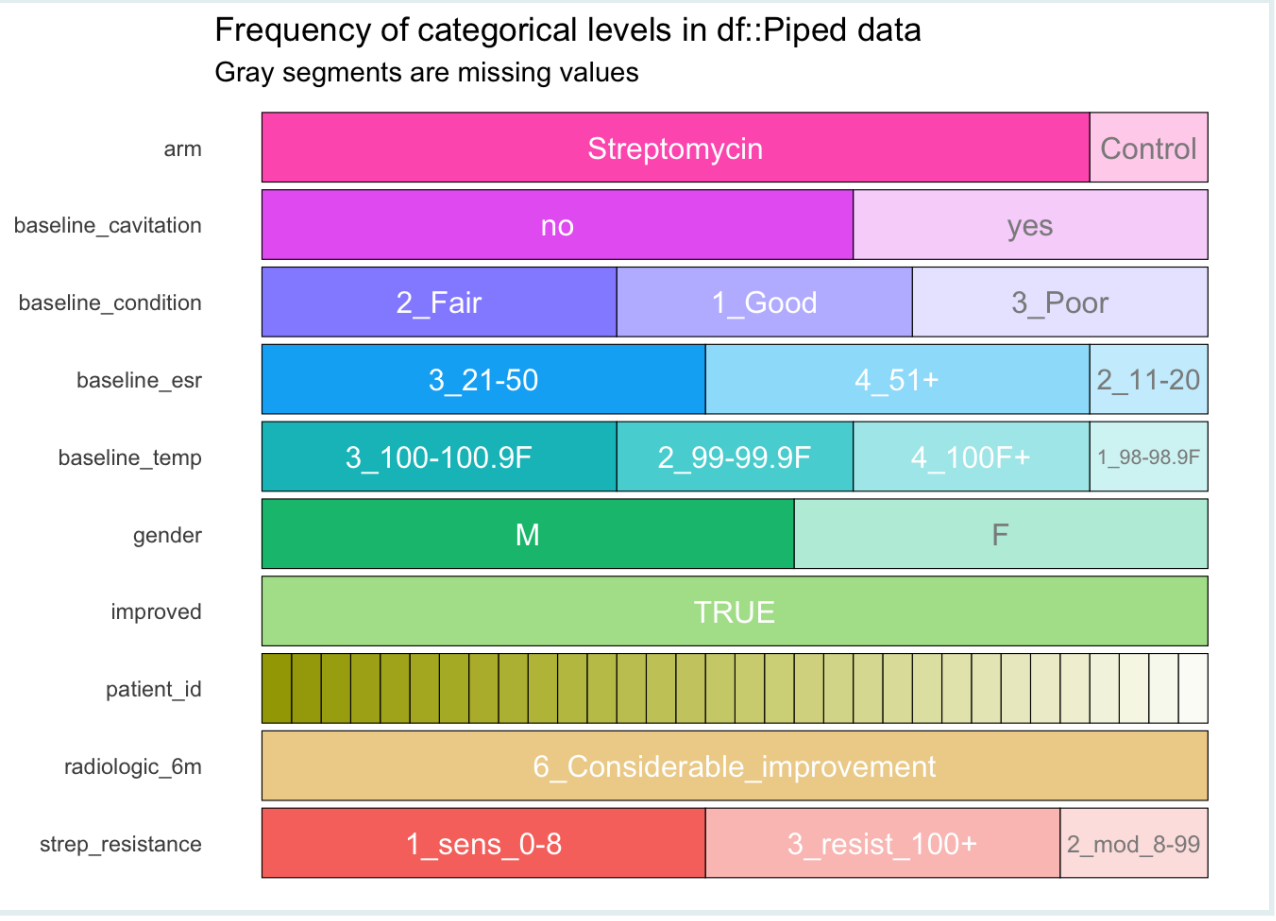
```
## $`6_Considerable_improvement`
## NULL
##
## $`5_Moderate_improvement`
## NULL
##
## $`4_No_change`
## NULL
##
## $`3_Moderate_deterioration`
## NULL
##
## $`2_Considerable_deterioration`
## NULL
##
## $`1_Death`
## NULL
```

Finalement, nous utiliserons une boucle pour générer un graphique pour chaque groupe et le stocker dans la liste :

```r
for (niveau in niveaux_radiologiques_6m) {

  # Générer le graphique pour chaque niveau
  cat_plot <-
    medicaldata::strep_tb %>%
    filter(radiologic_6m == niveau) %>%
    inspectdf::inspect_cat() %>%
    inspectdf::show_plot()

  # Ajouter à la liste
  liste_graphiques_cat[[niveau]] <- cat_plot
}
```

Pour accéder à un graphique spécifique, nous pouvons utiliser la syntaxe à double crochet :

```r
liste_graphiques_cat[["6_Considerable_improvement"]]
```



Notez que dans ce cas, les éléments de la liste sont *nommés*, plutôt que simplement numérotés. Cela est dû au fait que nous avons utilisé la variable `niveau` comme index dans la boucle.

Pour afficher tous les graphiques à la fois, nous appelons simplement la liste entière.

```
liste_graphiques_cat
```

```
## $`6_Considerable_improvement`
```

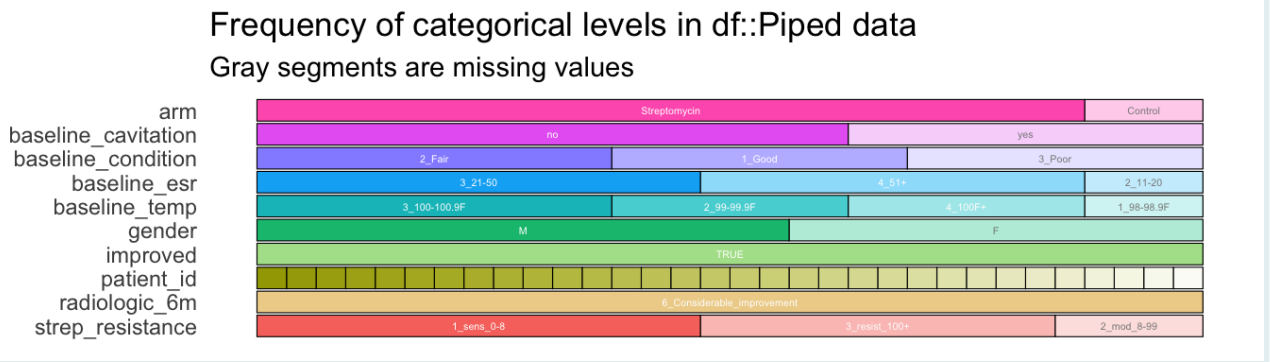
```
##
## $`5_Moderate_improvement`
```

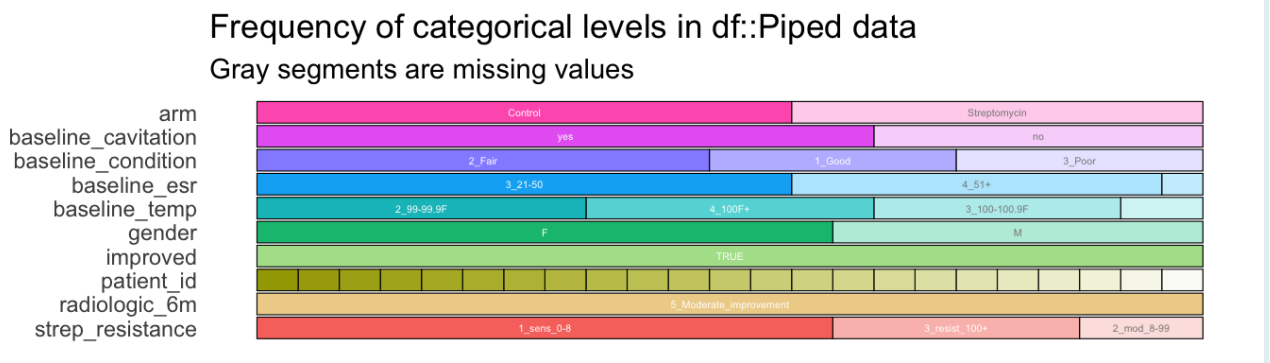
```
##
## $`4_No_change`
```

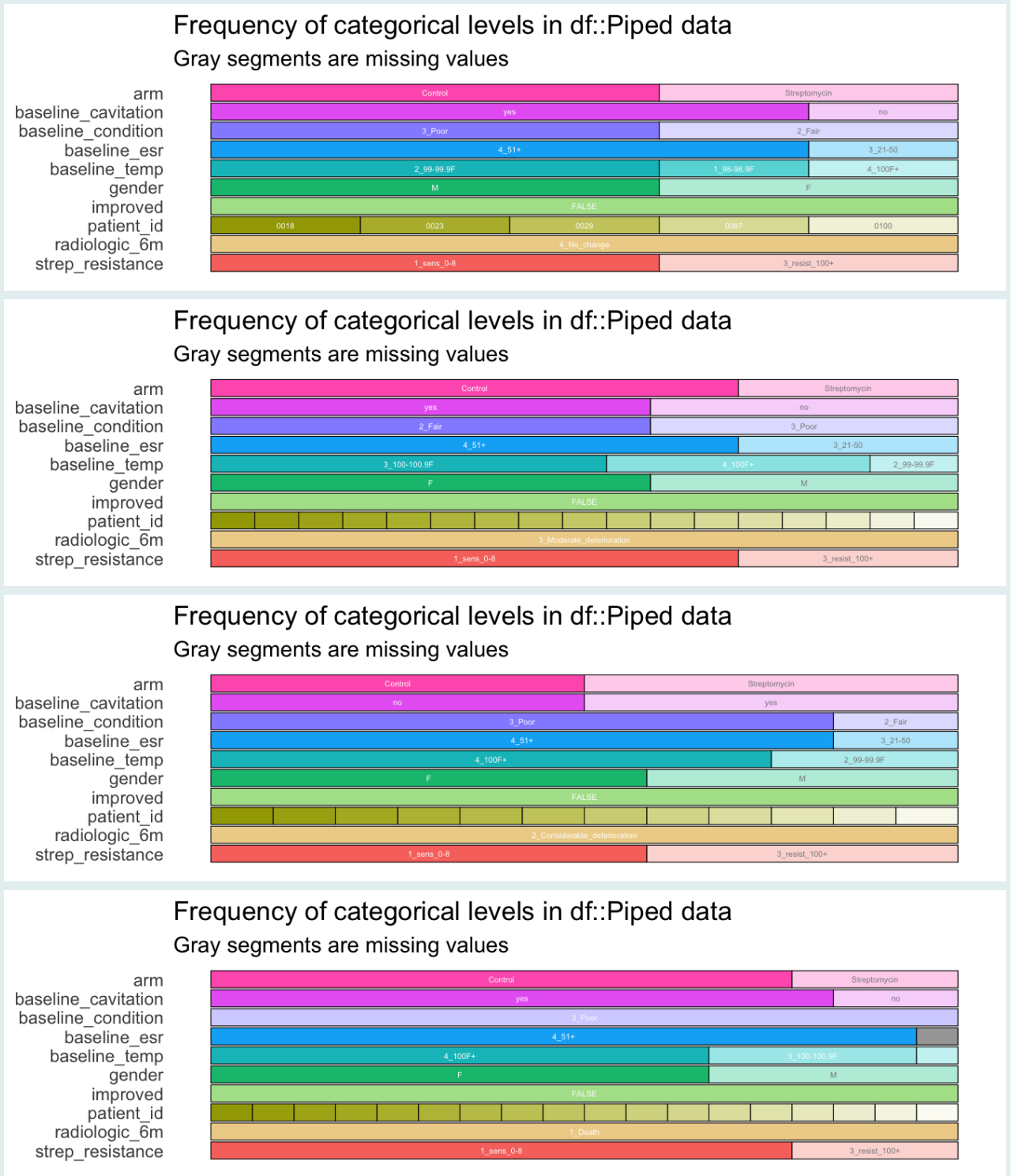
```
##
## $`3_Moderate_deterioration`
```

```
##
## $`2_Considerable_deterioration`
```

```
##
## $`1_Death`
```







Visualisation des cas de tuberculose

Dans cet exercice, vous utiliserez des données de l'OMS du package `tidyr` pour créer des graphiques en lignes montrant le nombre de nouveaux cas de tuberculose chez les enfants au fil des années dans les pays d'Amérique du Sud.

D'abord, nous préparerons les données :

```r
cas_tb_enfants <- tidyr::who2 %>%
  transmute(country, year,
            tb_cases_children = sp_m_014 + sp_f_014 + sn_m_014 + sn_f_014) %>%
  filter(country %in% c("Brazil", "Colombia", "Argentina",
                        "Uruguay", "Chile", "Guyana"))  %>%
  filter(year >= 2006)

cas_tb_enfants
```

```
## # A tibble: 48 × 3
##    country    year tb_cases_children
##    <chr>     <dbl>             <dbl>
##  1 Argentina  2006               880
##  2 Argentina  2007              1162
##  3 Argentina  2008               961
##  4 Argentina  2009               593
##  5 Argentina  2010               491
##  6 Argentina  2011               867
##  7 Argentina  2012               745
##  8 Argentina  2013                NA
##  9 Brazil     2006              2254
## 10 Brazil     2007              2237
## # ℹ 38 more rows
```

Maintenant, remplissez les blancs dans le modèle ci-dessous pour créer un graphique en lignes pour chaque pays en utilisant une boucle `for` :

```r
# Obtenez la liste des pays. Indice : Utilisez unique() sur la colonne pays
pays <- ____________________________________________

# Créez une liste pour stocker les graphiques. Indice : Initialisez une liste vide
graphiques_cas_tb_enfants <- vector("list", _________________)
names(graphiques_cas_tb_enfants) <- pays # Définissez les noms des éléments de la liste

# Boucle à travers les pays
for (pays in _____________) {

  # Filtrer les données pour chaque pays
  cas_tb_enfants_filtrés <- ____________________________________________

  # Créer le graphique
  graphique_cas_tb_enfants <- ____________________________________________

  #Ajouter à la liste. Indice : Utilisez des doubles crochets
  graphiques_cas_tb_enfants[[pays]] <- graphique_cas_tb_enfants
}

graphiques_cas_tb_enfants
```

```
## Error: <text>:2:10: unexpected input
## 1: # Obtenez la liste des pays. Indice : Utilisez unique() sur la colonne
pays
## 2: pays <- __
##             ^
```

Conclusion !

Dans cette leçon, nous nous sommes plongés dans les boucles `for` en R, démontrant leur utilité pour des tâches simples jusqu'à des analyses de données complexes impliquant plusieurs jeux de données et la génération de graphiques. Malgré la préférence de R pour les opérations vectorisées, les boucles for sont indispensables dans certains scénarios. Espérons que cette leçon vous a équipé des compétences nécessaires pour mettre en œuvre avec confiance les boucles `for` dans divers contextes de traitement de données.

Corrigé

Boucle Basique pour Convertir des Heures en Minutes

```r
heures <- c(3, 4, 5) # Vecteur d'heures

for (heure in heures) {
  minutes <- heure * 60
  print(minutes)
}
```

```
## [1] 180
## [1] 240
## [1] 300
```

Boucle Indexée pour Convertir des Heures en Minutes

```r
heures <- c(3, 4, 5) # Vecteur d'heures

for (i in 1:length(heures)) {
  minutes <- heures[i] * 60
  print(minutes)
}
```

```
## [1] 180
## [1] 240
## [1] 300
```

Boucle de Calcul de l'IMC

```
poids <- c(30, 32, 35) # Poids en kg
tailles <- c(1.2, 1.3, 1.4) # Tailles en mètres

for(i in 1:length(poids)) {
  imc <- poids[i] / (tailles[i] ^ 2)

  print(paste("Poids :", poids[i],
              "Taille :", tailles[i],
              "IMC :", imc))
}
```

```
## [1] "Poids : 30 Taille : 1.2 IMC : 20.8333333333333"
## [1] "Poids : 32 Taille : 1.3 IMC : 18.9349112426035"
## [1] "Poids : 35 Taille : 1.4 IMC : 17.8571428571429"
```

Conversion de la Taille de cm en m

```
taille_cm <- c(180, 170, 190, 160, 150) # Tailles en cm

taille_m <- vector("numeric", length = length(taille_cm))

for (i in 1:length(taille_cm)) {
  taille_m[i] <- taille_cm[i] / 100
}
taille_m
```

```
## [1] 1.8 1.7 1.9 1.6 1.5
```

Classification de la Température

```
temp_corporelle <- c(35, 36.5, 37, 38, 39.5) # Températures corporelles en
Celsius
vect_classif <- vector("character", length = length(temp_corps)) # vecteur de
caractères
```

```
## Error in vector("character", length = length(temp_corps)): object
'temp_corps' not found
```

```r
for (i in 1:length(temp_corps)) {
    # Ajoutez votre logique if-else ici
    if (temp_corporelle[i] < 36.5) {
        sortie <- "Hypothermie"
    } else if (temp_corporelle[i] <= 37.5) {
        sortie <- "Normal"
    } else {
        sortie <- "Fièvre"
    }

    # Instruction d'impression finale
    vect_classif[i] <- paste(temp_corporelle[i], "°C est", sortie)
}
```

```
## Error in eval(expr, envir, enclos): object 'temp_corps' not found
```

```r
vect_classif
```

```
## Error in eval(expr, envir, enclos): object 'vect_classif' not found
```

Propriétés des Fichiers

```r
# En supposant que le chemin et la structure des fichiers sont corrects
fichiers_csv <- list.files(path = "data/new_hiv_infections_gho",
                           pattern = "\\.csv$", full.names = TRUE)

liste_data_frames <- vector("list", length = length(fichiers_csv))

for (i in 1:length(fichiers_csv)) {

  chemin <- fichiers_csv[i]
  nom_pays <- tools::file_path_sans_ext(basename(chemin))

  taille <- file.size(chemin)
  date <- file.mtime(chemin)

  ligne_data_hiv <- tibble(pays = nom_pays, taille = taille, date = date)

  liste_data_frames[[i]] <- ligne_data_hiv
}

info_fichiers_hiv_final <- bind_rows(liste_data_frames)
info_fichiers_hiv_final
```

```
## # A tibble: 9 × 3
##   pays                          taille date
##   <chr>                          <dbl> <dttm>
## 1 Bangladesh                      6042 2023-12-11 17:34:28
```

```
## 2 Brazil                                5946 2023-12-11 17:34:28
## 3 Democratic_Republic_of_the_Con…       8028 2023-12-11 17:34:28
## 4 Egypt                                 6181 2023-12-11 17:34:28
## 5 Ethiopia                              5754 2023-12-11 17:34:28
## 6 Indonesia                             6621 2023-12-11 17:34:28
## 7 Iran_(Islamic_Republic_of)            8037 2023-12-11 17:34:28
## 8 Philippines                           6321 2023-12-11 17:34:28
## 9 Viet_Nam                              6230 2023-12-11 17:34:28
```

Filtrage des Données en Boucle

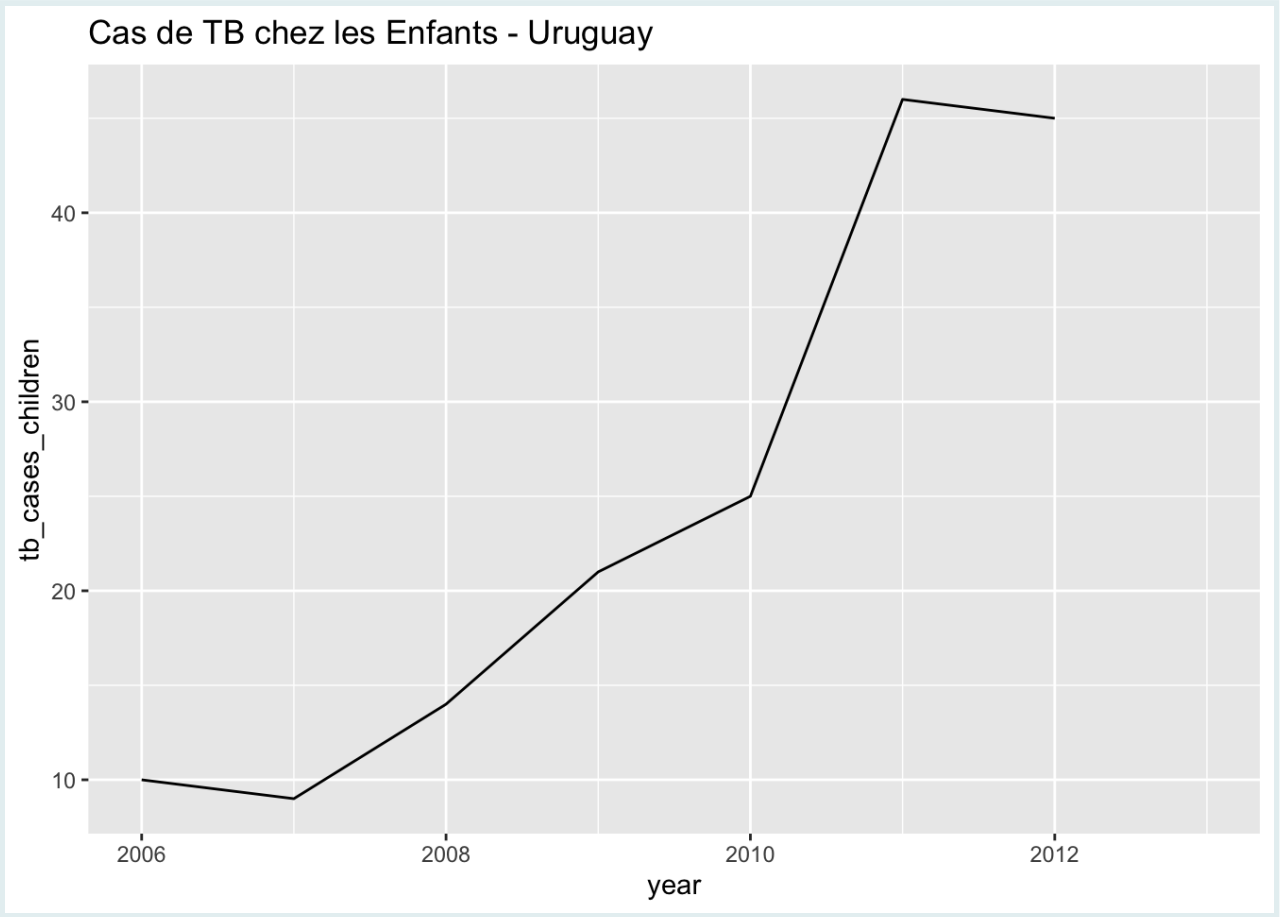
```
fichiers_csv <- list.files(path = "data/new_hiv_infections_gho",
                           pattern = "*.csv", full.names = TRUE)

for (fichier in fichiers_csv) {
  donnees_vih <- read_csv(fichier)

  donnees_vih_filtrees <- donnees_vih %>% filter(Sex == "Female")

  nom_fichier_sortie <- paste0(here(), "/outputs/", "Female_",
basename(fichier))

  write_csv(donnees_vih_filtrees, nom_fichier_sortie)
}
```

```
## Rows: 89 Columns: 5
## ── Column specification ────────────────────────────────────────────────────────
## Delimiter: ","
## chr (4): Continent, Country, Sex, NewHIVCases
## dbl (1): Year
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this
message.


## Error: Cannot open file for writing:
## *
'/Users/kendavidn/Dropbox/tgc_github_projects/fdar_staging/FDAR_FR_loops/outputs/Female_E
```

Visualisation des Cas de Tuberculose

```
# En supposant que tb_child_cases est un dataframe avec les colonnes
nécessaires
pays <- unique(cas_tb_enfants$country)

# Crée une liste pour stocker les graphiques
graphiques_cas_tb_enfants <- vector("list", length(pays))
names(graphiques_cas_tb_enfants) <- pays

# Boucle à travers les pays
for (nom_pays in pays) {

  # Filtre les données pour chaque pays
  cas_tb_enfants_filtres <- filter(cas_tb_enfants, country == nom_pays)

  # Crée le graphique
  graphique_cas_tb_enfant <- ggplot(cas_tb_enfants_filtres, aes(x = year, y =
tb_cases_children)) +
    geom_line() +
    ggtitle(paste("Cas de TB chez les Enfants -", nom_pays))

  # Ajoute au liste
  graphiques_cas_tb_enfants[[nom_pays]] <- graphique_cas_tb_enfant
}

graphiques_cas_tb_enfants[["Uruguay"]]
```

```
## Warning: Removed 1 row containing missing values (`geom_line()`).
```



Contributeurs

Les membres suivants de l'équipe ont contribué à cette leçon :

SABINA RODRIGUEZ VELÁSQUEZ

Project Manager and Scientific Collaborator, The GRAPH Network
Infectiously enthusiastic about microbes and Global Health

KENE DAVID NWOSU

Data analyst, the GRAPH Network
Passionate about world improvement

GUY WAFEU

R Instructor and Public Health Physician
Committed to improving the quality of data analysis

Références

Du matériel dans cette leçon a été adapté des sources suivantes :

- Barnier, Julien. “Introduction à R et au tidyverse.” https://juba.github.io/tidyverse
- Wickham, Hadley; Grolemund, Garrett. “R for Data Science.” https://r4ds.had.co.nz/
- Wickham, Hadley; Grolemund, Garrett. “R for Data Science (2e).” https://r4ds.hadley.nz/