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

Packages

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

```r
# 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 :

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

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

```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.

```r
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 :

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

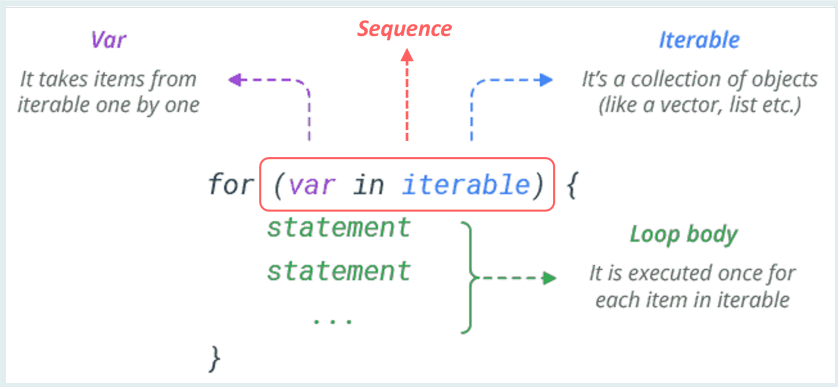
```
## [1] 84
## [1] 96
## [1] 108
```

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
```

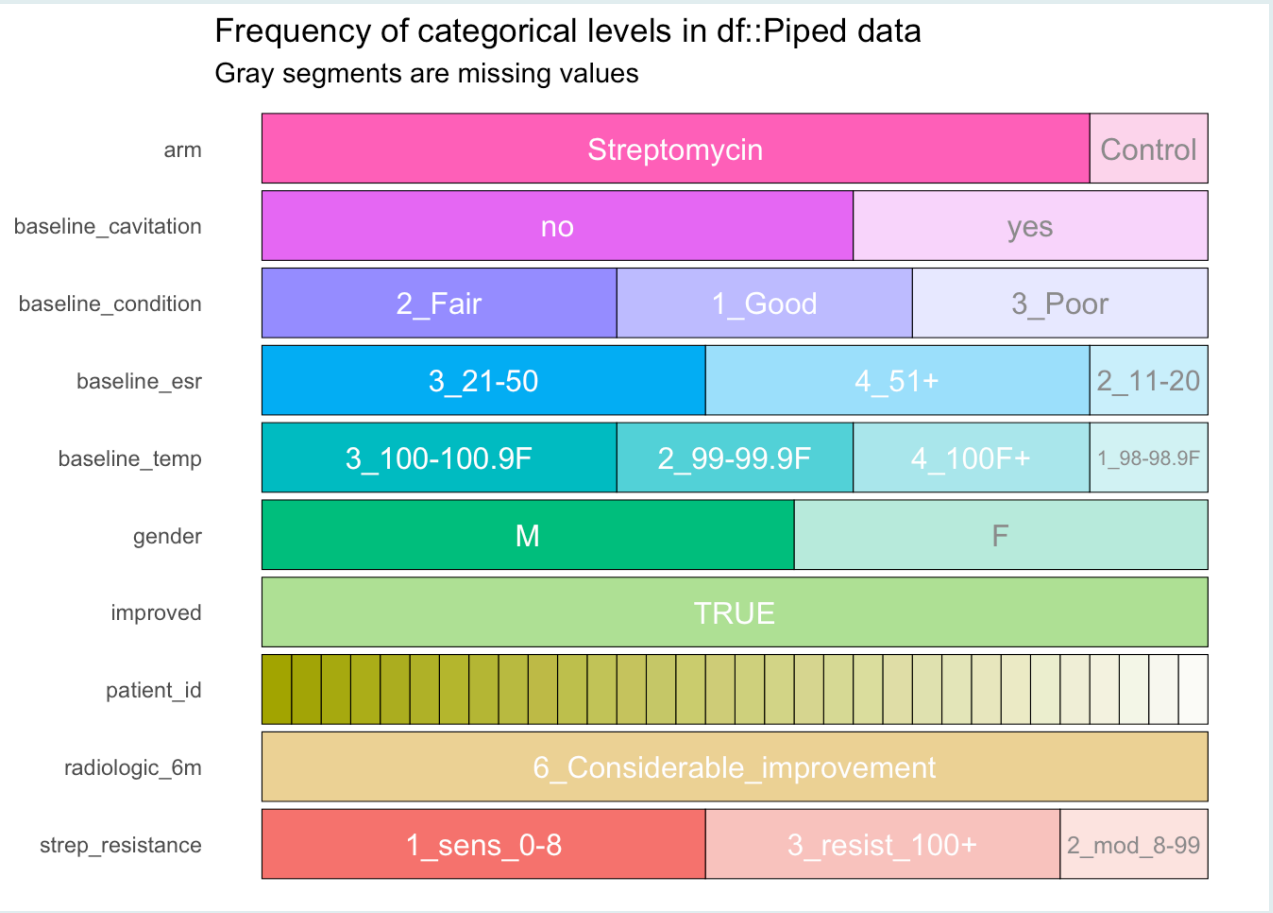
Application Réelle des Boucles: Génération de Plusieurs Graphiques

Maintenant que vous avez une bonne compréhension des boucles, nous allons explorer une application réelle des boucles dans l'analyse de données: la génération de plusieurs graphiques.

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

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

```
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.

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

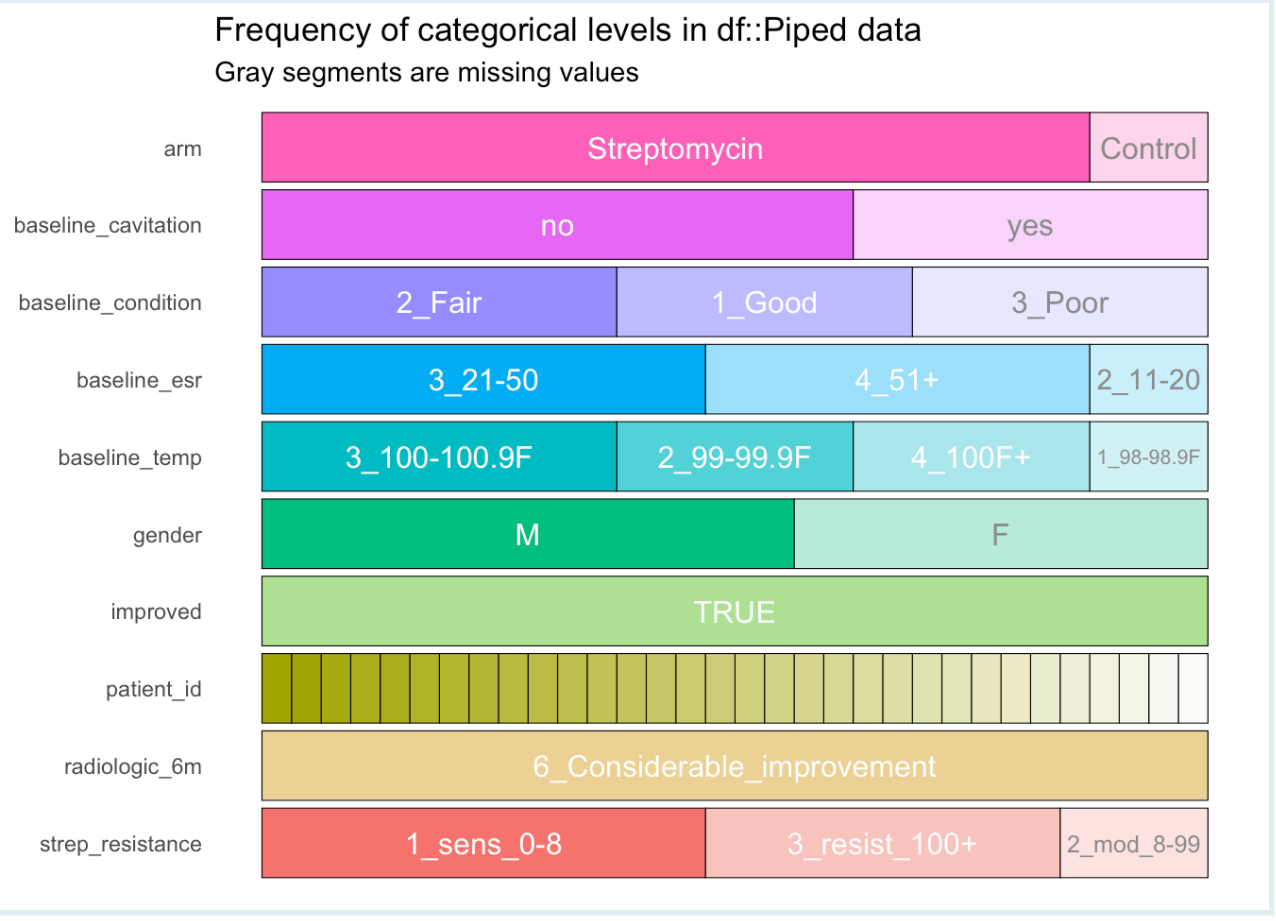
```
## $`6_Considerable_improvement`
## NULL
##
## $`5_Moderate_improvement`
## NULL
##
## $`4_No_change`
## NULL
##
## $`3_Moderate_deterioration`
## NULL
##
## $`2_Considerable_deterioration`
## NULL
##
## $`1_Death`
## NULL
```

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

```
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 :

```
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`
```

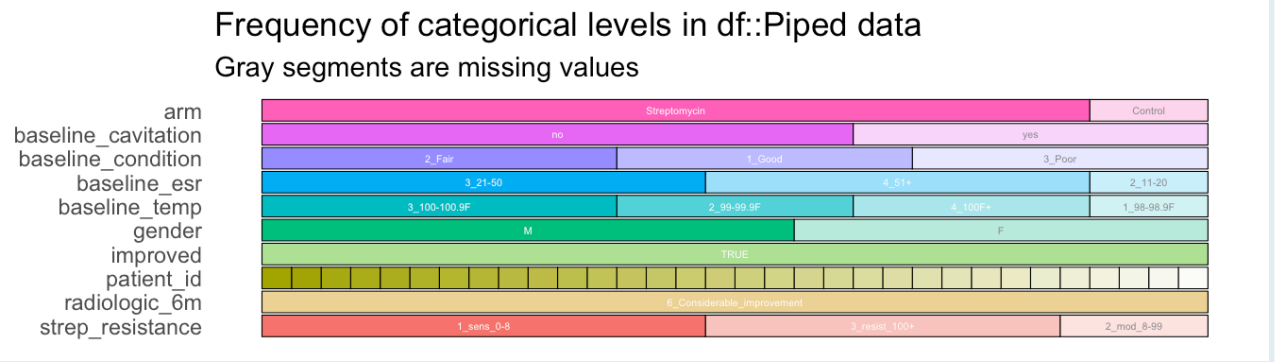
```
##
## $`5_Moderate_improvement`
```

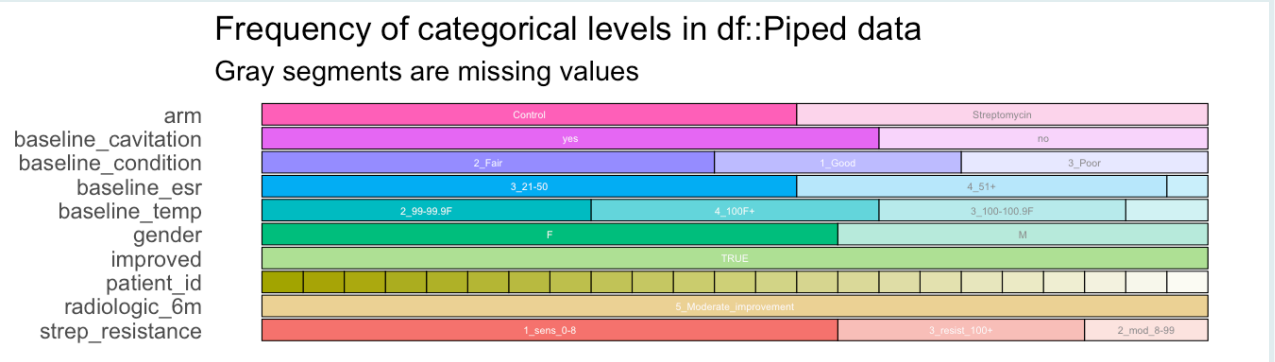
```
##
## $`4_No_change`
```

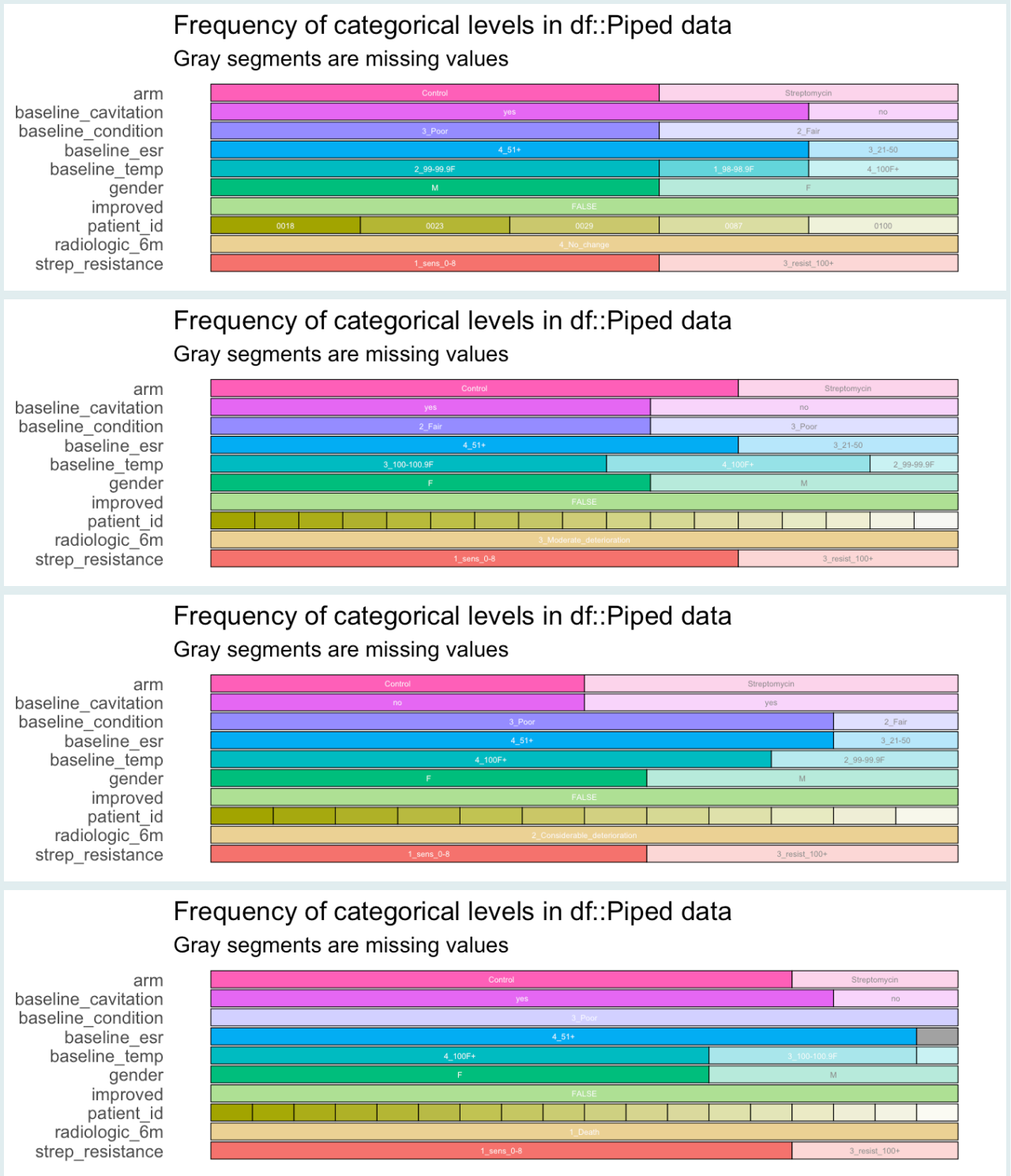
```
##
## $`3_Moderate_deterioration`
```

```
##
## $`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
```

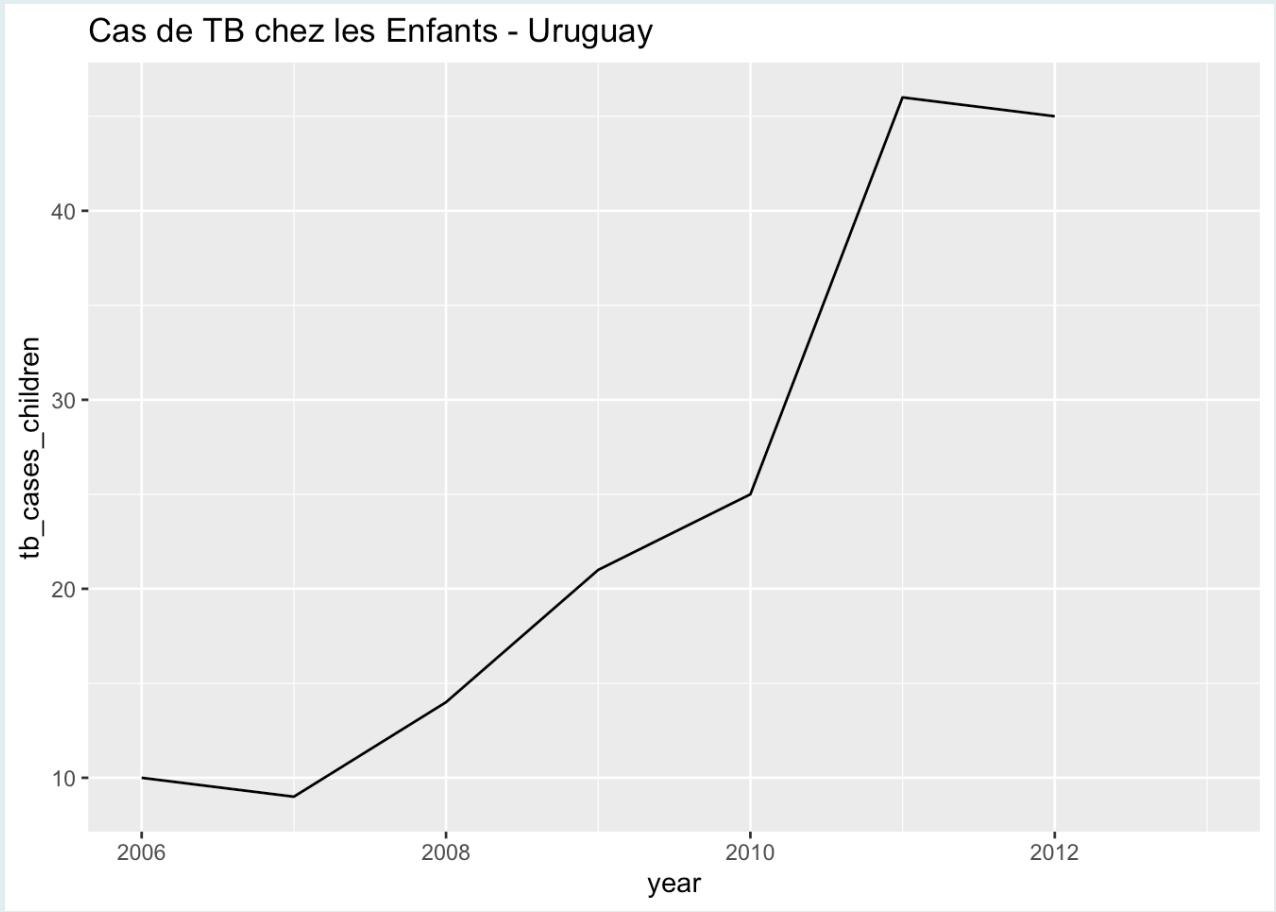
```r
vect_classif
```

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

Visualisation des Cas de Tuberculose

```r
# 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/