Notes de leçon | Structures de données

GRAPH Network & OMS, soutenu par le Fonds Mondial

January 2024

Ce cours a été créé par le Réseau GRAPH, une organisation à but non lucratif basée à l'Institut de santé globale de l'Université de Genève, en collaboration avec l'Organisation mondiale de la Santé, dans le cadre d'une subvention du Fonds mondial pour créer des cours afin de renforcer les capacités nationales en matière d'analyse épidémiologique.

Intro

Dans cette leçon, nous allons jeter un bref coup d'œil aux structures de données dans R. Comprendre les structures de données est crucial pour la manipulation et l'analyse des données. Nous commencerons par explorer les vecteurs, la structure de données de base dans R. Ensuite, nous apprendrons à combiner des vecteurs dans des data frames, la structure la plus courante pour organiser et analyser des données.

Objectifs d'apprentissage

1. Vous pouvez créer des vecteurs avec la fonction `c()`.
2. Vous pouvez combiner des vecteurs dans des data frames.
3. Vous comprenez la différence entre un tibble et un data frame.

Packages

Veuillez charger les packages nécessaires pour cette leçon avec le code ci-dessous :

```
ıire(pacman)) install.packages("pacman")
p_load(tidyverse)
```

Introduction aux vecteurs

Les structures de données les plus élémentaires dans R sont les vecteurs. Les vecteurs sont une collection de valeurs qui partagent toutes la même classe (par exemple, toutes les valeurs numériques ou tous les caractères). Il peut être utile de penser à un vecteur comme une colonne dans une feuille de calcul Excel.

Création de vecteurs

Les vecteurs peuvent être créés en utilisant la fonction `c()`, avec les composantes du vecteur étant séparées par des virgules. Par exemple, le code `c(1, 2, 3)` définit un vecteur avec les éléments `1`, `2` et `3`.

Dans votre script, définissez les vecteurs suivants :

```
(18, 25, 46)
 c('M', 'F', 'F')
sitif <- c(T, T, F)
 3 # le deux points créent une séquence de nombres
```

Vous pouvez également vérifier les classes de ces vecteurs :

```
ge)
```

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

```
exe)
```

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

```
est_positif)
```

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

Chaque ligne de code ci-dessous tente de définir un vecteur avec trois éléments mais contient une erreur. Corrigez les erreurs et effectuez l'affectation.

```
_1 <- (1,2,3)
_2 <- c("Obi", "Chika" "Nonso")
```

Les valeurs individuelles au sein d'un vecteur sont appelées *composantes* ou éléments. Donc le vecteur `c(1, 2, 3)` a trois composantes/éléments.

Manipulation de vecteurs

La plupart des fonctions et opérations que vous avez déjà rencontrées jusqu'à présent peuvent être appliquées à des vecteurs.

Par exemple, nous pouvons multiplier notre objet `age` par 2 :

```

```

```
## [1] 18 25 46
```

```

```

```
## [1] 36 50 92
```

Remarquez que chaque élément du vecteur a été multiplié par 2.

Ou, ci-dessous nous prenons la racine carrée de `age` :

```

```

```
## [1] 18 25 46
```

```
e)
```

```
## [1] 4.242641 5.000000 6.782330
```

Vous pouvez également additionner (numériquement) des vecteurs :

```
i
```

```
## [1] 19 27 49
```

Remarquez que le premier élément de `age` est ajouté au premier élément de `id` et le deuxième élément de `age` est ajouté au deuxième élément de `id`, et ainsi de suite.

Des vecteurs aux data frames

Maintenant que nous maîtrisons la création de vecteurs, passons à l'objet le plus couramment utilisé dans R : les data frames. Un data frame est juste une collection de vecteurs de même longueur avec des métadonnées utiles. Nous pouvons en créer un en utilisant la fonction `data.frame()`.

Nous avons précédemment créé des variables de type vecteur (id, age, sexe et test_positif) pour trois individus :

Nous pouvons maintenant utiliser la fonction `data.frame()` pour les combiner dans une seule structure tabulaire :

```
epi <- data.frame(id, age, sexe, test_positif)
epi
```

```
##   id age sexe test_positif
## 1  1  18    M         TRUE
## 2  2  25    F         TRUE
## 3  3  46    F        FALSE
```

Remarquez qu'au lieu de créer chaque vecteur séparément, vous pouvez créer votre data frame en définissant chacun des vecteurs à l'intérieur de la fonction `data.frame()`.

```
epi_2 <- data.frame(age = c(18, 25, 46),
                    sexe = c('M', 'F', 'F'))

epi_2
```

```
##   age sexe
## 1  18    M
## 2  25    F
## 3  46    F
```

SIDE NOTE

SIDE NOTE 

La plupart du temps lorsque vous travaillez avec des données dans R, vous les importerez à partir de contextes externes. Mais il est parfois utile de créer des jeux de données *dans* R lui-même. C'est dans de tels cas que la fonction `data.frame()` sera utile.

Pour extraire les vecteurs du data frame, utilisez la syntaxe `$`. Exécutez les lignes de code suivantes dans votre console pour observer cela.

```
_epi$age
)r(donnees_epi$age) # vérifie que cette colonne est bien un vecteur
)nnees_epi$age) # vérifie la classe du vecteur
```

Combinez les vecteurs ci-dessous dans un data frame, avec les noms de colonnes suivants : "nom" pour le vecteur de caractères, "nb_enfants" pour le vecteur numérique et "est_marie" pour le vecteur logique.

```
_caracteres <- c("Bob", "Jane", "Joe")
_numerique <- c(1, 2, 3)
_logique <- c(T, F, F)
```

Utilisez la fonction `data.frame()` pour définir dans R un data frame qui ressemble au tableau suivant :

salle	nb_fenetres
salle à manger	3
cuisine	2
chambre	5

Tibbles

La version par défaut des données tabulaires dans R s'appelle un data frame, mais il existe une autre représentation des données tabulaires fournie par le package *tidyverse*. Elle s'appelle un `tibble`, et c'est une version améliorée du data frame.

Vous pouvez convertir un data frame en tibble avec la fonction `as_tibble()` :

```
_epi
```

```
##   id age sexe test_positif
## 1  1  18    M         TRUE
## 2  2  25    F         TRUE
## 3  3  46    F        FALSE
```

```
epi <- as_tibble(donnees_epi)
epi
```

```
## # A tibble: 3 × 4
##      id   age sexe  test_positif
##   <int> <dbl> <chr> <lgl>
## 1     1    18 M     TRUE
## 2     2    25 F     TRUE
## 3     3    46 F     FALSE
```

Remarquez que le tibble donne les dimensions des données dans la première ligne :

```
👉# A tibble: 3 × 4👈
     id   age sexe   test_positif
  <int> <dbl> <chr> <lgl>
1     1    18 M     VRAI
2     2    25 F     VRAI
3     3    46 F     FAUX
```

Et indique également les types de données, en haut de chaque colonne :

```
# A tibble: 3 × 4
     id   age sexe   test_positif
👉   <int> <dbl> <chr> <lgl> 👈
1     1    18 M     VRAI
2     2    25 F     VRAI
3     3    46 F     FAUX
```

Ici, “int” signifie entier, “dbl” signifie double (qui est une sorte de classe numérique), “chr” signifie caractère et “lgl” signifie logique.

L’autre avantage des tibbles est qu’ils évitent d’inonder votre console lorsque vous imprimez un grand tableau.

Considérez par exemple la sortie console des lignes ci-dessous :

```
ne le data frame infert (un jeu de données R intégré)
# Impression très longue
le(infert) # impression plus gérable
```

Pour la plupart de vos besoins d’analyse de données, vous devriez préférer les tibbles aux data frames classiques.

`read_csv()` crée des tibbles

Lorsque vous importez des données avec la fonction `read_csv()` de {readr}, vous obtenez un tibble :

```
.b <- read_csv("https://tinyurl.com/ebola-data-sample") # Nécessite internet
 pour fonctionner
)ola_tib)
```

```
## [1] "spec_tbl_df" "tbl_df"      "tbl"         "data.frame"
```

Mais lorsque vous importez des données avec la fonction `read.csv()` de base, vous obtenez un data.frame :

```
f <- read.csv("https://tinyurl.com/ebola-data-sample") # Nécessite internet pour
 fonctionner
)ola_df)
```

```
## [1] "data.frame"
```

Essayez d'imprimer `ebola_tib` et `ebola_df` dans votre console pour observer le comportement d'impression différent des tibbles et des data frames.

C'est une des raisons pour lesquelles nous recommandons d'utiliser `read_csv()` plutôt que `read.csv()`.

Conclusion

Avec votre compréhension des classes et des structures de données, vous êtes désormais bien équipé pour effectuer des tâches de manipulation de données dans R. Dans les prochaines leçons, nous explorerons les puissantes capacités de transformation de données du package dplyr, qui amélioreront encore vos compétences en analyse de données.

Félicitations d'être arrivé jusque-là ! Vous avez beaucoup appris et vous pouvez être fiers de vous.

Solutions

Solution au premier bloc r-practice :

```
_1 <- c(1,2,3) # Utilisez la fonction 'c' pour créer un vecteur
_2 <- c("Obi", "Chika", "Nonso") # Séparez chaque chaîne par une virgule
```

Solution au deuxième bloc r-practice :

```
ata.frame(nom = vecteur_caracteres,
          nb_enfants = vecteur_numerique,
          est_marie = vecteur_logique)
```

Solution au troisième bloc r-practice :

```
ion au troisième bloc r-practice
<- data.frame(piece = c("salle à manger", "cuisine", "chambre"),
              nb_fenetres = c(3, 2, 5))
```

Contributeurs

L'équipe suivante a contribué à cette leçon :

DANIEL CAMARA

Data Scientist at the GRAPH Network and fellowship as Public Health researcher at Fiocruz, Brazil
Passionate about lots of things, especially when it involves people leading lives with more equality and freedom

EDUARDO ARAUJO

Student at Universidade Tecnologica Federal do Parana
Passionate about reproducible science and education

LAURE VANCAUWENBERGHE

Data analyst, the GRAPH Network
A firm believer in science for good, striving to ally programming, health and education

KENE DAVID NWOSU

Data analyst, the GRAPH Network
Passionate about world improvement

Références

Certains éléments de cette leçon ont été adaptés des sources suivantes :

- Wickham, H., & Grolemund, G. (s.d.). *R for data science*. 15 Factors | R for Data Science. Consulté le 26 octobre 2022. https://r4ds.had.co.nz/factors.html.