Cours 2 : Les boucles

Exercices

Pour ces exemples on chargera le package ggplot2 car il contient economics, les données qu'on utilisera en exemple.

library(ggplot2)  
head(economics)

## date pce pop psavert uempmed unemploy  
## 1 1967-07-01 507.4 198712 12.5 4.5 2944  
## 2 1967-08-01 510.5 198911 12.5 4.7 2945  
## 3 1967-09-01 516.3 199113 11.7 4.6 2958  
## 4 1967-10-01 512.9 199311 12.5 4.9 3143  
## 5 1967-11-01 518.1 199498 12.5 4.7 3066  
## 6 1967-12-01 525.8 199657 12.1 4.8 3018

Cette table contient des données macroéconomiques régulièrement mesurées dans le temps.

On prépare les données en créant des variables année et mois à partir de la date avec le package lubridate.

library(lubridate)  
tab <- economics  
tab$annee <- year(tab$date)  
tab$mois <- month(tab$date)  
  
head(tab[c("date", "annee", "mois")])

## date annee mois  
## 1 1967-07-01 1967 7  
## 2 1967-08-01 1967 8  
## 3 1967-09-01 1967 9  
## 4 1967-10-01 1967 10  
## 5 1967-11-01 1967 11  
## 6 1967-12-01 1967 12

Si on souhaite obtenir des données à partir d'une année précise, une approche est de travailler sur un subset d'une année. Par exemple pour pce, personal consumption expenditures, in billions of dollars :

temp <- tab$pce[tab$annee == 1967]  
  
mean(temp)

## [1] 515.1667

Mais effectuer cette opération année par année est laborieux. On peut donc utiliser donc une boucle.

# Les boucles for

Une boucle for est construite de la façon suivante :

vecteur <- c(1, 3, 5)  
  
for (variable in vecteur) {  
 print(variable)  
   
 autre\_variable <- variable \* 10  
 print(autre\_variable)  
}

## [1] 1  
## [1] 10  
## [1] 3  
## [1] 30  
## [1] 5  
## [1] 50

* Le code contenu dans la boucle est répété autant de fois qu'il y a de valeurs dans vecteur ;
* variable prend successivement toutes les valeurs de vecteur ;
* Il est possible d'exectuer n'importe quel type de code dans la boucle.

Exercices :

1. Afficher la moyenne de pce par année avec une boucle for.
2. Stocker la moyenne de pce par année dans un vecteur.
3. Découper la table par an et par mois et sauvegarder chaque mois dans un fichier csv différent.

## Exercice 1

annees <- 1967:1980  
  
for(i in annees){  
 temp <- tab$pce[tab$annee == i]   
 mean(temp)  
}

Toutefois cette boucle n'affiche rien. C'est parceque le contenu de la boucle n'est pas automatiquement "printé".[[1]](#footnote-23) Cette nuance est très importante, surtout pour des objets dont on oublie qu'ils ont besoin de passer par print pour être affichés (en particulier les graphiques de ggplot2).

for(i in annees){  
 temp <- tab$pce[tab$annee == i]   
 print(mean(temp))  
}

## [1] 515.1667  
## [1] 557.4583  
## [1] 604.4833  
## [1] 647.6917  
## [1] 701  
## [1] 769.4333  
## [1] 851.1417  
## [1] 932  
## [1] 1032.767  
## [1] 1150.167  
## [1] 1276.675  
## [1] 1426.175  
## [1] 1589.483  
## [1] 1754.625

## Exercice 2

On peut vouloir stocker le résultat au lieu de juste l'afficher dans la console.

output <- numeric()  
for(i in annees){  
 temp <- tab$pce[tab$annee == i]   
 output <- c(output, mean(temp))  
}  
  
output

## [1] 515.1667 557.4583 604.4833 647.6917 701.0000 769.4333 851.1417  
## [8] 932.0000 1032.7667 1150.1667 1276.6750 1426.1750 1589.4833 1754.6250

## Exercice 3

Si on souhaite découper le jeu de données en tables mensuelles on peut utiliser des doubles boucles : d'abord par année, puis par mois.

# dossier pour les tables crées  
dir.create("./exercice\_temp")  
  
for(i in 1970:1971){  
 yr <- tab[tab$annee == i, ]  
   
 for(j in 1:12){  
 temp <- yr[yr$mois == j, ]  
   
 file\_name <- paste0(  
 "./exercice\_temp/donnees\_",  
 i,  
 "\_",  
 month.name[j],  
 ".csv")  
   
 write.csv2(x = temp, file = file\_name, row.names = FALSE)  
 }  
}

# Les lapply

Un lapply est construit de la façon suivante :

vecteur <- c(.5, 1, 2)  
lapply(X = vecteur, FUN = exp)

## [[1]]  
## [1] 1.648721  
##   
## [[2]]  
## [1] 2.718282  
##   
## [[3]]  
## [1] 7.389056

* lapply prend deux arguments, un vecteur et une fonction ;
* La fonction est appliquées succesivement à tous les éléments du vecteur ;
* Le résultat est retourné dans une liste.

Exercice :

1. Lire les tables qu'on vient de créer et les agréger en une seule table.

Si on souhaite faire l'opération inverse il faut donc lire une série de table, puis les concaténer. Le problème est où stocker la série de table ? Il n'est pas très pratique de créer une série d'objets dans l'environnement global.

C'est là que les listes deviennent pratiques : on peut tout mettre dans une liste de tables. C'est possible de faire ça dans une boucle for, mais lapply est encore plus pratique.

fichiers <- list.files("./exercice\_temp/", full.names = TRUE)  
  
list\_tables <- lapply(fichiers, read.csv2)  
head(list\_tables)

## [[1]]  
## date pce pop psavert uempmed unemploy annee mois  
## 1 1970-04-01 637 204401 13.3 4.1 3797 1970 4  
##   
## [[2]]  
## date pce pop psavert uempmed unemploy annee mois  
## 1 1970-08-01 653.9 205295 13.1 5.4 4256 1970 8  
##   
## [[3]]  
## date pce pop psavert uempmed unemploy annee mois  
## 1 1970-12-01 666.6 206238 12.9 5.9 5076 1970 12  
##   
## [[4]]  
## date pce pop psavert uempmed unemploy annee mois  
## 1 1970-02-01 634.9 204008 11.6 4.5 3453 1970 2  
##   
## [[5]]  
## date pce pop psavert uempmed unemploy annee mois  
## 1 1970-01-01 629.6 203849 11.7 4.6 3201 1970 1  
##   
## [[6]]  
## date pce pop psavert uempmed unemploy annee mois  
## 1 1970-07-01 649.5 205052 13.2 5.1 4175 1970 7

for (i in 1:length(list\_tables)) {  
 if (i == 1) {  
 result <- list\_tables[[i]]  
 } else {  
 result <- rbind(  
 result,  
 list\_tables[[i]]  
 )  
 }  
}  
  
head(result)

## date pce pop psavert uempmed unemploy annee mois  
## 1 1970-04-01 637.0 204401 13.3 4.1 3797 1970 4  
## 2 1970-08-01 653.9 205295 13.1 5.4 4256 1970 8  
## 3 1970-12-01 666.6 206238 12.9 5.9 5076 1970 12  
## 4 1970-02-01 634.9 204008 11.6 4.5 3453 1970 2  
## 5 1970-01-01 629.6 203849 11.7 4.6 3201 1970 1  
## 6 1970-07-01 649.5 205052 13.2 5.1 4175 1970 7

# Les apply

Pour appliquer ligne à ligne une fonction sur une table (ou colonne par colonne). Un apply est construit de la façon suivante :

test <- data.frame(  
 x = 1:2,  
 y = 3:4  
)  
  
test

## x y  
## 1 1 3  
## 2 2 4

apply(X = test, MARGIN = 1, FUN = sum)

## [1] 4 6

apply(X = test, MARGIN = 2, FUN = sum)

## x y   
## 3 7

* apply est fait pour travailler sur des tables ;
* La fonction qu'on lui passe peut être appliquée ligne par ligne (MARGIN = 1) ou colonne par colonne (MARGIN = 2).

x <- data.frame(  
 x1 = c(rep(3, 7), NA),  
 x2 = c(4:2, NA, 2:4, NA)  
)  
x

## x1 x2  
## 1 3 4  
## 2 3 3  
## 3 3 2  
## 4 3 NA  
## 5 3 2  
## 6 3 3  
## 7 3 4  
## 8 NA NA

Exercices :

1. A partir de la table x, calculer la moyenne par ligne.
2. Calculer le nombre de données manquantes par colonne. Faire un LOCF.

## Exercice 1

Par exemple pour calculer une moyenne par ligne. On peut passer tous les arguments qu'on veut à mean par l'intermédiaire de la fonction apply, par exemple ici na.rm = TRUE.

apply(x, 1, mean)

## [1] 3.5 3.0 2.5 NA 2.5 3.0 3.5 NA

apply(x, 1, mean, na.rm = TRUE)

## [1] 3.5 3.0 2.5 3.0 2.5 3.0 3.5 NaN

## Exercice 2

Il est possible d'utiliser avec apply une fonction qu'on a écrit. [[2]](#footnote-30) Par exemple pour compter les données manquantes par variables :

n\_na <- function(x) {  
 return(sum(is.na(x)))  
}  
apply(x, 2, n\_na)

## x1 x2   
## 1 2

Ou faire un LOCF :

locf <- function(x) {  
 x\_ok <- na.omit(x)  
 if (length(x\_ok) > 0) {  
 return(x\_ok[length(x\_ok)])  
 } else {  
 return(NA)  
 }  
}  
  
x$x3 <- apply(x, 1, locf)  
x

## x1 x2 x3  
## 1 3 4 4  
## 2 3 3 3  
## 3 3 2 2  
## 4 3 NA 3  
## 5 3 2 2  
## 6 3 3 3  
## 7 3 4 4  
## 8 NA NA NA

# Pour aller plus loin

## for

Dans une boucle for on peut boucler comme dans l'exemple sur un vecteur numérique, mais aussi bien entendu sur un vecteur d'un autre type (character par exemple), ou même sur une liste. Par exemple :

for (tab in list\_tables[1:3]) {  
 print(summary(tab))  
}

## date pce pop psavert   
## 1970-04-01:1 Min. :637 Min. :204401 Min. :13.3   
## 1st Qu.:637 1st Qu.:204401 1st Qu.:13.3   
## Median :637 Median :204401 Median :13.3   
## Mean :637 Mean :204401 Mean :13.3   
## 3rd Qu.:637 3rd Qu.:204401 3rd Qu.:13.3   
## Max. :637 Max. :204401 Max. :13.3   
## uempmed unemploy annee mois   
## Min. :4.1 Min. :3797 Min. :1970 Min. :4   
## 1st Qu.:4.1 1st Qu.:3797 1st Qu.:1970 1st Qu.:4   
## Median :4.1 Median :3797 Median :1970 Median :4   
## Mean :4.1 Mean :3797 Mean :1970 Mean :4   
## 3rd Qu.:4.1 3rd Qu.:3797 3rd Qu.:1970 3rd Qu.:4   
## Max. :4.1 Max. :3797 Max. :1970 Max. :4   
## date pce pop psavert   
## 1970-08-01:1 Min. :653.9 Min. :205295 Min. :13.1   
## 1st Qu.:653.9 1st Qu.:205295 1st Qu.:13.1   
## Median :653.9 Median :205295 Median :13.1   
## Mean :653.9 Mean :205295 Mean :13.1   
## 3rd Qu.:653.9 3rd Qu.:205295 3rd Qu.:13.1   
## Max. :653.9 Max. :205295 Max. :13.1   
## uempmed unemploy annee mois   
## Min. :5.4 Min. :4256 Min. :1970 Min. :8   
## 1st Qu.:5.4 1st Qu.:4256 1st Qu.:1970 1st Qu.:8   
## Median :5.4 Median :4256 Median :1970 Median :8   
## Mean :5.4 Mean :4256 Mean :1970 Mean :8   
## 3rd Qu.:5.4 3rd Qu.:4256 3rd Qu.:1970 3rd Qu.:8   
## Max. :5.4 Max. :4256 Max. :1970 Max. :8   
## date pce pop psavert   
## 1970-12-01:1 Min. :666.6 Min. :206238 Min. :12.9   
## 1st Qu.:666.6 1st Qu.:206238 1st Qu.:12.9   
## Median :666.6 Median :206238 Median :12.9   
## Mean :666.6 Mean :206238 Mean :12.9   
## 3rd Qu.:666.6 3rd Qu.:206238 3rd Qu.:12.9   
## Max. :666.6 Max. :206238 Max. :12.9   
## uempmed unemploy annee mois   
## Min. :5.9 Min. :5076 Min. :1970 Min. :12   
## 1st Qu.:5.9 1st Qu.:5076 1st Qu.:1970 1st Qu.:12   
## Median :5.9 Median :5076 Median :1970 Median :12   
## Mean :5.9 Mean :5076 Mean :1970 Mean :12   
## 3rd Qu.:5.9 3rd Qu.:5076 3rd Qu.:1970 3rd Qu.:12   
## Max. :5.9 Max. :5076 Max. :1970 Max. :12

## Listes

Dans une liste [] sélectionne une autre liste alors que [[]] sélectionne le *contenu* d'un élément de la liste. Si une liste est un train composé de wagons qui contiennent des vaches[[3]](#footnote-34) alors [] sélectionne un ou plusieurs wagons et retourne donc un autre train alors que [[]] sélectionne les vaches contenues dans un wagon particulier.

# un train avec 3 wagons  
exemple\_liste <- list(  
 c(1, 3, 5), # des vaches  
 c("a, b"), # d'autres vaches  
 c(TRUE, FALSE) # encore d'autres vaches  
)  
exemple\_liste

## [[1]]  
## [1] 1 3 5  
##   
## [[2]]  
## [1] "a, b"  
##   
## [[3]]  
## [1] TRUE FALSE

# un train avec 2 wagons  
exemple\_liste[1:2]

## [[1]]  
## [1] 1 3 5  
##   
## [[2]]  
## [1] "a, b"

# un train avec 1 wagon  
exemple\_liste[2]

## [[1]]  
## [1] "a, b"

# le contenu du wagon 2  
exemple\_liste[[2]]

## [1] "a, b"

# donc ca ne marche pas car c'est une liste  
exemple\_liste[1] \* 2

## Error in exemple\_liste[1] \* 2: non-numeric argument to binary operator

# mais ca marche  
exemple\_liste[[1]] \* 2

## [1] 2 6 10

# de meme on ne peut selectionner que le contenu  
# d'un seul wagon a la fois  
exemple\_liste[[2:3]]

## Error in exemple\_liste[[2:3]]: subscript out of bounds

## Mmmmmmagic

Dans l'exemple avec lapply on peut éviter l'appel à la boucle for avec cette opération magique :

result <- Reduce(rbind, list\_tables)  
head(result)

## date pce pop psavert uempmed unemploy annee mois  
## 1 1970-04-01 637.0 204401 13.3 4.1 3797 1970 4  
## 2 1970-08-01 653.9 205295 13.1 5.4 4256 1970 8  
## 3 1970-12-01 666.6 206238 12.9 5.9 5076 1970 12  
## 4 1970-02-01 634.9 204008 11.6 4.5 3453 1970 2  
## 5 1970-01-01 629.6 203849 11.7 4.6 3201 1970 1  
## 6 1970-07-01 649.5 205052 13.2 5.1 4175 1970 7

?Reduce pour les plus curieux<

1. Appeler print automatiquement est le comportement par défaut de R dans la console, et les utilisateurs pourraient penser que c'est habituel. En réalité c'est une exception, la règle est de ne pas appeler print. [↑](#footnote-ref-23)
2. Avec lapply aussi d'ailleurs. [↑](#footnote-ref-30)
3. Fallait pas confier cette partie du cours à un Lorrain. [↑](#footnote-ref-34)