Gestion de données

Jean-Michel Marin

Février 2023

CNRS – IMAG (Montpellier, France)

Supports réalisés à partir de l'ouvrage R pour la statistique et la science des données

Introduction

Introduction

- Les origines de R remontent à la création du langage S au milieu des années 1970.
- À cette époque, la volumétrie et l'organisation des données étaient radicalement différentes de ce que l'on connaît aujourd'hui
- Deux packages majeurs se partagent la manipulation des données :
  - data.table, qui vise avant tout l'efficacité en étant très optimisé
  - dplyr qui réalise un compromis entre efficacité et facilité d'utilisation
- On ne traite ici que du package data.table ; pour dplyr voir ouvrage **R pour la statistique et la science des données**

Le package data.table

Le package data.table

- Le package data.table est conçu pour traiter de manière efficace, tant du point de vue du temps d'exécution que de l'empreinte mémoire utilisée, des data.frame très volumineux
- Le package définit une nouvelle syntaxe de manipulation des données

Importation avec fread

La création d'un data.table est similaire à celle d'un data.frame

```
library(data.table)
set.seed(1234)
dt <- data.table(group1 = c("A", "B"),
group2 = rep(c("C", "D"), each = 5000),
value = rnorm(10000),
weight = sample(1:10, 10000, replace = TRUE))
```

Importation avec fread

```
dt
```

```
##         group1 group2       value weight
##      1:      A      C -1.20706575      5
##      2:      B      C  0.27742924      7
##      3:      A      C  1.08444118      1
##      4:      B      C -2.34569770      4
##      5:      A      C  0.42912469      9
##     ---
##   9996:      B      D  0.01973902      7
##   9997:      A      D -2.12674529      6
##   9998:      B      D -0.05022201      9
##   9999:      A      D -0.23817408      7
##  10000:      B      D  0.77640531      7
```

Importation avec fread

```
write.table(dt,"dt.csv",sep=",",row.names=FALSE,quote=FALSE)
cat("File size (MB):",round(file.info("dt.csv")$size/1024^2),"\n")
```

```
## File size (MB): 0
```

Importation avec fread

```
system.time(df <- read.table("dt.csv", sep=",", header=TRUE,
stringsAsFactors=FALSE))
```

```
## utilisateur     système      écoulé
##       0.012       0.000       0.013
```

```
system.time(dt <- fread("dt.csv"))
```

```
## utilisateur     système      écoulé
##       0.004       0.000       0.004
```

```
all.equal(as.data.table(df),dt)
```

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

Importation avec fread

```
set.seed(1234)
n <- 1e6
dt2 <- data.table(a=sample(1:1000,n,replace=TRUE),
b=runif(n),
c=rnorm(n),
d=sample(c("A","B","C","D"),n,replace=TRUE))
write.table(dt2,"dt2.csv",sep=",",row.names=FALSE,quote=FALSE)
cat("Taille en (MB):", round(file.info("dt2.csv")$size/1024^2),"\n")
```

```
## Taille en (MB): 40
```

Importation avec fread

```
system.time(df2 <- read.table("dt2.csv", sep=",", header=TRUE,
stringsAsFactors=FALSE))
```

```
## utilisateur     système      écoulé
##       4.324       0.089       4.417
```

```
system.time(dt2 <- fread("dt2.csv"))
```

```
## utilisateur     système      écoulé
##       0.110       0.011       0.121
```

Importation avec fread

- Il y a un facteur 20 en gain de temps
- Si l'on reprend cette procédure avec 10 millions de lignes, ce qui donne un fichier d'environ 400 Mb,
  - la lecture avec read.table laisse le temps de prendre un café
  - avec fread, le gain de temps est cette fois d'un facteur 30

Syntaxe

- La syntaxe pour manipuler un data.table peut sembler proche, au premier abord, de celle utilisée pour les data.frame car elle utilise des crochets [...]
- Elle est néanmoins très différente en pratique

De façon générique la manipulation d'un data.table revêt la forme suivante

```
data[i, j, by]
```

où i définit une sous-sélection des lignes, j un calcul (opération, modification, sélection) sur les colonnes et by un groupement des lignes

Syntaxe

```
dt[group1 == "A", mean(value), by = group2]
```

```
##    group2           V1
## 1:      C 0.0008501418
## 2:      D 0.0539426361
```

On constate ici l'appel aux différentes colonnes sans avoir besoin de les préfixer par dt$ comme c'est le cas en R de base

Sélection

- Sur les lignes : i
- Un data.table ne dispose pas de rownames ; la sélection peut se faire de différentesmanières
- Par exemple, en utilisant les indices des lignes

```
dt[1:2, ]
```

```
##    group1 group2      value weight
## 1:      A      C -1.2070657      5
## 2:      B      C  0.2774292      7
```

Sélection

- Sur les lignes : i

```
dt[c(1,5)] # virgule optionnelle
```

```
##    group1 group2      value weight
## 1:      A      C -1.2070657      5
## 2:      A      C  0.4291247      9
```

Sélection

- Sur les lignes : i

```
dt[order(value), ]
```

```
##         group1 group2     value weight
##      1:      B      C -3.396064      3
##      2:      B      D -3.335108      9
##      3:      A      C -3.233152      2
##      4:      A      D -3.200732      3
##      5:      B      D -3.185662      3
##     ---
##   9996:      B      D  3.357021     10
##   9997:      B      D  3.456972      9
##   9998:      A      D  3.560775      4
##   9999:      B      D  3.606510     10
##  10000:      B      D  3.618107      2
```

Sélection

- Sur les lignes : i

```
dt[weight > 9, ] # pas besoin de guillemets
```

```
##      group1 group2      value weight
##   1:      A      C -0.7762539     10
##   2:      B      C -1.4482049     10
##   3:      A      C  0.5747557     10
##   4:      A      C -0.9943401     10
##   5:      B      C -1.2519859     10
##  ---
## 992:      B      D -1.5507432     10
## 993:      A      D -0.1802639     10
## 994:      A      D  1.0293445     10
## 995:      B      D  0.7022495     10
## 996:      A      D  1.0235271     10
```

Sélection

- Sur les lignes : i

```
dt[weight > 9 & group2 == "C", ]
```

```
##      group1 group2      value weight
##   1:      A      C -0.7762539     10
##   2:      B      C -1.4482049     10
##   3:      A      C  0.5747557     10
##   4:      A      C -0.9943401     10
##   5:      B      C -1.2519859     10
##  ---
## 491:      B      C -0.9043558     10
## 492:      A      C -0.1230837     10
## 493:      B      C -1.1952174     10
## 494:      B      C -0.2176025     10
## 495:      A      C -1.2130983     10
```

Selection

- Sur les colonnes : j

```
dt[, 1]
```

```
##         group1
##      1:      A
##      2:      B
##      3:      A
##      4:      B
##      5:      A
##     ---
##   9996:      B
##   9997:      A
##   9998:      B
##   9999:      A
##  10000:      B
```

Selection

- Sur les colonnes : j

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

```
##        group1       value
##     1:      A -1.20706575
##     2:      B  0.27742924
##     3:      A  1.08444118
##     4:      B -2.34569770
##     5:      A  0.42912469
##    ---
##  9996:      B  0.01973902
##  9997:      A -2.12674529
##  9998:      B -0.05022201
##  9999:      A -0.23817408
## 10000:      B  0.77640531
```

Selection

- Sur les colonnes : j

```
dt[, "group1"]
```

```
##        group1
##     1:      A
##     2:      B
##     3:      A
##     4:      B
##     5:      A
##    ---
##  9996:      B
##  9997:      A
##  9998:      B
##  9999:      A
## 10000:      B
```

Selection

- Sur les colonnes : j

```
dt[, c("group1", "value")]
```

```
##        group1       value
##     1:      A -1.20706575
##     2:      B  0.27742924
##     3:      A  1.08444118
##     4:      B -2.34569770
##     5:      A  0.42912469
##    ---
##  9996:      B  0.01973902
##  9997:      A -2.12674529
##  9998:      B -0.05022201
##  9999:      A -0.23817408
## 10000:      B  0.77640531
```

Selection

- Sur les colonnes : j

```
dt[, list(group1, value)]
```

```
##        group1       value
##     1:      A -1.20706575
##     2:      B  0.27742924
##     3:      A  1.08444118
##     4:      B -2.34569770
##     5:      A  0.42912469
##    ---
##  9996:      B  0.01973902
##  9997:      A -2.12674529
##  9998:      B -0.05022201
##  9999:      A -0.23817408
## 10000:      B  0.77640531
```

Selection

- Sur les colonnes : j

```
dt[, .(group1, value)]
```

```
##        group1       value
##     1:      A -1.20706575
##     2:      B  0.27742924
##     3:      A  1.08444118
##     4:      B -2.34569770
##     5:      A  0.42912469
##    ---
##  9996:      B  0.01973902
##  9997:      A -2.12674529
##  9998:      B -0.05022201
##  9999:      A -0.23817408
## 10000:      B  0.77640531
```

Modification des colonnes

```
dt[, tval := trunc(value)][1:2]
```

```
##    group1 group2      value weight tval
## 1:      A      C -1.2070657      5   -1
## 2:      B      C  0.2774292      7    0
```

- Contrairement aux data.frame, il n'est pas nécessaire de créer un nouvel objet qui corresponde à l'ancien de jeu de données concaténé avec la nouvelle colonne
- L'objet initial est modifié. Ici, la nouvelle colonne tval fait maintenant partie du data.table dt

Modification des colonnes

Il est possible de créer plusieurs colonnes de deux façons : par référence

```
dt[, c("tvalue", "rvalue") := list(trunc(value), round(value, 2))]
dt[, ':=' (tvalue = trunc(value), rvalue = round(value ,2))]
```

Modification des colonnes

La modification d'une colonne existante est très facile, quant à sa suppression elle se fait en affectant la valeur NULL à la colonne

```
dt[, tvalue := tvalue + 10] # modification
dt[, rvalue := NULL] # suppression
```

Modification des colonnes

En combinant les deux paramètres i et j, il est donc possible de modifier directement des colonnes pour une sous-population

```
dt[ group1 %in% "A", weight := weight * 10L][1:2]
```

```
##    group1 group2      value weight tvalue
## 1:      A      C -1.2070657     50      9
## 2:      B      C  0.2774292      7     10
```

Dans la première ligne de code, on utilise L pour indiquer que c'est un entier et non un réel (double), sans quoi on reçoit un avertissement

Modification des colonnes

Avec cette syntaxe, on peut par exemple remplacer facilement l'ensemble des valeurs manquantes d'une colonne par sa moyenne

```
dt[is.na(value), value := mean(value, na.rm = TRUE)]
```

Calculs et agrégations avec by

Le calcul d'agrégats est facilité avec la troisième partie by

Agrégats sur une seule variable

```
dt[, sum(value), by = group1]
```

```
##    group1        V1
## 1:      A 136.98194
## 2:      B -75.82302
```

```
dt[, sum(value), by = "group1"]
```

```
##    group1        V1
## 1:      A 136.98194
## 2:      B -75.82302
```

Calculs et agrégations avec by

Agrégats sur plusieurs variables

```
dt[, list(somme = sum(value)), by = list(group1, group2)]
```

```
##    group1 group2      somme
## 1:      A      C   2.125355
## 2:      B      C -33.898836
## 3:      A      D 134.856590
## 4:      B      D -41.924180
```

```
dt[, list(somme = sum(value)), by = c("group1", "group2")]
```

```
##    group1 group2      somme
## 1:      A      C   2.125355
## 2:      B      C -33.898836
## 3:      A      D 134.856590
## 4:      B      D -41.924180
```

Calculs et agrégations avec by

Nous pouvons renommer les variables utilisées et nous servir d'expressions R

```
dt[, list(somme = sum(value)), by = list(pop = group1, poids = weight>5)]
```

```
##    pop poids     somme
## 1:   A FALSE  58.74680
## 2:   B  TRUE -38.23590
## 3:   B FALSE -37.58711
## 4:   A  TRUE  78.23514
```

dplyr

dplyr - Introduction

dplyr est l'un des packages les plus connus du tidyverse

Il facilite le traitement et la manipulation des tableaux de données (qu'il s'agisse de data frame ou de tibble)

Il propose une syntaxe claire et cohérente, sous formes de verbes correspondant à des fonctions

dplyr part du principe que les données sont tidy. Les verbes de dplyr prennent en entrée un tableau de données (data frame ou tibble) et renvoient systématiquement un tibble

```
library(dplyr)
```

dplyr - Introduction

Dans ce qui suit on va utiliser le jeu de données nycflights13, contenu dans l'extension du même nom

Celui-ci correspond aux données de tous les vols au départ d'un des trois aéroports de New-York en 2013. Il a la particularité d'être réparti en trois tables

- nycflights13::flights contient des informations sur les vols : date, départ, destination, horaires, retard...
- nycflights13::airports contient des informations sur les aéroports
- nycflights13::airlines contient des données sur les compagnies aériennes

On va charger les trois tables du jeu de données

dplyr - Introduction

```r
library(dplyr)
library(nycflights13)
data(flights)
data(airports)
data(airlines)
```

Opérations sur les lignes

dplyr::filter() sélectionne des lignes d'un tableau de données selon une condition. On lui passe en paramètre un test, et seules les lignes pour lesquelles ce test renvoi TRUE (vrai) sont conservées

Opérations sur les lignes

```r
filter(flights, month == 1)
```

```
## # A tibble: 27,004 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      517            515         2      830            819
##  2  2013     1     1      533            529         4      850            830
##  3  2013     1     1      542            540         2      923            850
##  4  2013     1     1      544            545        -1     1004           1022
##  5  2013     1     1      554            600        -6      812            837
##  6  2013     1     1      554            558        -4      740            728
##  7  2013     1     1      555            600        -5      913            854
##  8  2013     1     1      557            600        -3      709            723
##  9  2013     1     1      557            600        -3      838            846
## 10  2013     1     1      558            600        -2      753            745
## # i 26,994 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

Opérations sur les lignes

```
flights |> filter(month == 1)
```

```
## # A tibble: 27,004 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      517            515         2      830            819
##  2  2013     1     1      533            529         4      850            830
##  3  2013     1     1      542            540         2      923            850
##  4  2013     1     1      544            545        -1     1004           1022
##  5  2013     1     1      554            600        -6      812            837
##  6  2013     1     1      554            558        -4      740            728
##  7  2013     1     1      555            600        -5      913            854
##  8  2013     1     1      557            600        -3      709            723
##  9  2013     1     1      557            600        -3      838            846
## 10  2013     1     1      558            600        -2      753            745
## # i 26,994 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

Opérations sur les lignes

Si l'on veut uniquement les vols avec un retard au départ (variable dep_delay) compris entre 10 et 15 minutes

```
flights |>
  filter(dep_delay >= 10 & dep_delay <= 15)
```

```
## # A tibble: 14,919 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      611            600        11      945            931
##  2  2013     1     1      623            610        13      920            915
##  3  2013     1     1      743            730        13     1107           1100
##  4  2013     1     1      743            730        13     1059           1056
##  5  2013     1     1      851            840        11     1215           1206
##  6  2013     1     1      912            900        12     1241           1220
##  7  2013     1     1      914            900        14     1058           1043
##  8  2013     1     1      920            905        15     1039           1025
##  9  2013     1     1     1011           1001        10     1133           1128
## 10  2013     1     1     1112           1100        12     1440           1438
## # i 14,909 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

Opérations sur les lignes

Si l'on passe plusieurs arguments à dplyr::filter(), celui-ci rajoute automatiquement une condition ET

```
flights |>
  filter(dep_delay >= 10, dep_delay <= 15)
```

```
## # A tibble: 14,919 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      611            600        11      945            931
##  2  2013     1     1      623            610        13      920            915
##  3  2013     1     1      743            730        13     1107           1100
##  4  2013     1     1      743            730        13     1059           1056
##  5  2013     1     1      851            840        11     1215           1206
##  6  2013     1     1      912            900        12     1241           1220
##  7  2013     1     1      914            900        14     1058           1043
##  8  2013     1     1      920            905        15     1039           1025
##  9  2013     1     1     1011           1001        10     1133           1128
## 10  2013     1     1     1112           1100        12     1440           1438
## # i 14,909 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

Opérations sur les lignes

Enfin, on peut également placer des fonctions dans les tests, qui nous permettent par exemple de sélectionner les vols avec la plus grande distance

```
flights |>
  filter(distance == max(distance))
```

```
## # A tibble: 342 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      857            900        -3     1516           1530
##  2  2013     1     2      909            900         9     1525           1530
##  3  2013     1     3      914            900        14     1504           1530
##  4  2013     1     4      900            900         0     1516           1530
##  5  2013     1     5      858            900        -2     1519           1530
##  6  2013     1     6     1019            900        79     1558           1530
##  7  2013     1     7     1042            900       102     1620           1530
##  8  2013     1     8      901            900         1     1504           1530
##  9  2013     1     9      641            900      1301     1242           1530
## 10  2013     1    10      859            900        -1     1449           1530
## # i 332 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

Opérations sur les lignes

Évaluation contextuelle

Il est important de noter que dplyr procède à une évaluation contextuelle des expressions qui lui sont passées

Ainsi, on peut indiquer directement le nom d'une variable et dplyr l'interprétera dans le contexte du tableau de données, c'est-à-dire regardera s'il existe une colonne portant ce nom dans le tableau

Dans l'expression flights |> filter(month == 1), month est interprété comme la colonne month du tableau flights, à savoir flights$month

Opérations sur les lignes

Il est également possible d'indiquer des objets extérieurs au tableau

```
m <- 2
flights |>
  filter(month == m)
```

```
## # A tibble: 24,951 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     2     1      456            500        -4      652            648
##  2  2013     2     1      520            525        -5      816            820
##  3  2013     2     1      527            530        -3      837            829
##  4  2013     2     1      532            540        -8     1007           1017
##  5  2013     2     1      540            540         0      859            850
##  6  2013     2     1      552            600        -8      714            715
##  7  2013     2     1      552            600        -8      919            910
##  8  2013     2     1      552            600        -8      655            709
##  9  2013     2     1      553            600        -7      833            815
## 10  2013     2     1      553            600        -7      821            825
## # i 24,941 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

Opérations sur les lignes

Par contre, si une colonne existe dans le tableau, elle aura priorité sur les objets du même nom dans l'environnement

```
month <- 3
flights |>
  filter(month == month)
```

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      517            515         2      830            819
##  2  2013     1     1      533            529         4      850            830
##  3  2013     1     1      542            540         2      923            850
##  4  2013     1     1      544            545        -1     1004           1022
##  5  2013     1     1      554            600        -6      812            837
##  6  2013     1     1      554            558        -4      740            728
##  7  2013     1     1      555            600        -5      913            854
##  8  2013     1     1      557            600        -3      709            723
##  9  2013     1     1      557            600        -3      838            846
## 10  2013     1     1      558            600        -2      753            745
## # i 336,766 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

Opérations sur les lignes

Afin de distinguer ce qui correspond à une colonne du tableau et à un objet de l'environnement, on pourra avoir recours à .data et .env (voir help(".env", package = "rlang"))

Opérations sur les lignes

```
month <- 3
flights |>
  filter(.data$month == .env$month)
```

```
## # A tibble: 28,834 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     3     1        4           2159       125      318             56
##  2  2013     3     1       50           2358        52      526            438
##  3  2013     3     1      117           2245       152      223           2354
##  4  2013     3     1      454            500        -6      633            648
##  5  2013     3     1      505            515       -10      746            810
##  6  2013     3     1      521            530        -9      813            827
##  7  2013     3     1      537            540        -3      856            850
##  8  2013     3     1      541            545        -4     1014           1023
##  9  2013     3     1      549            600       -11      639            703
## 10  2013     3     1      550            600       -10      747            801
## # i 28,824 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

Opérations sur les lignes

Le verbe dplyr::slice() sélectionne des lignes du tableau selon leur position. On lui passe un chiffre ou un vecteur de chiffres.

Si l'on souhaite sélectionner la 345e ligne du tableau airports

```
airports |>
  slice(345)
```

```
## # A tibble: 1 x 8
##   faa   name                lat   lon   alt    tz dst   tzone
##   <chr> <chr>             <dbl> <dbl> <dbl> <dbl> <chr> <chr>
## 1 CYF   Chefornak Airport  60.1 -164.    40    -9 A     America/Anchorage
```

Opérations sur les lignes

Si l'on veut sélectionner les 5 premières lignes

```
airports |>
  slice(1:5)
```

```
## # A tibble: 5 x 8
##   faa   name                            lat   lon   alt    tz dst   tzone
##   <chr> <chr>                         <dbl> <dbl> <dbl> <dbl> <chr> <chr>
## 1 04G   Lansdowne Airport              41.1 -80.6  1044    -5 A     America/New~
## 2 06A   Moton Field Municipal Airport  32.5 -85.7   264    -6 A     America/Chi~
## 3 06C   Schaumburg Regional            42.0 -88.1   801    -6 A     America/Chi~
## 4 06N   Randall Airport                41.4 -74.4   523    -5 A     America/New~
## 5 09J   Jekyll Island Airport          31.1 -81.4    11    -5 A     America/New~
```

Opérations sur les lignes

dplyr::arrange() réordonne les lignes d'un tableau selon une ou plusieurs colonnes. Ainsi, si l'on veut trier le tableau flights selon le retard au départ, dans l'ordre croissant

```
flights |>
  arrange(dep_delay)
```

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013    12     7     2040           2123       -43       40           2352
##  2  2013     2     3     2022           2055       -33     2240           2338
##  3  2013    11    10     1408           1440       -32     1549           1559
##  4  2013     1    11     1900           1930       -30     2233           2243
##  5  2013     1    29     1703           1730       -27     1947           1957
##  6  2013     8     9      729            755       -26     1002            955
##  7  2013    10    23     1907           1932       -25     2143           2143
##  8  2013     3    30     2030           2055       -25     2213           2250
##  9  2013     3     2     1431           1455       -24     1601           1631
## 10  2013     5     5      934            958       -24     1225           1309
## # i 336,766 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

Opérations sur les lignes

On peut trier selon plusieurs colonnes. Par exemple selon le mois, puis selon le retard au départ

```
flights |>
  arrange(month, dep_delay)
```

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1    11     1900           1930       -30     2233           2243
##  2  2013     1    29     1703           1730       -27     1947           1957
##  3  2013     1    12     1354           1416       -22     1606           1650
##  4  2013     1    21     2137           2159       -22     2232           2316
##  5  2013     1    20      704            725       -21     1025           1035
##  6  2013     1    12     2050           2110       -20     2310           2355
##  7  2013     1    12     2134           2154       -20        4             50
##  8  2013     1    14     2050           2110       -20     2329           2355
##  9  2013     1     4     2140           2159       -19     2241           2316
## 10  2013     1    11     1947           2005       -18     2209           2230
## # i 336,766 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

Opérations sur les lignes

Si l'on veut trier selon une colonne par ordre décroissant, on lui applique la fonction dplyr::desc()

```
flights |>
  arrange(desc(dep_delay))
```

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     9      641            900      1301     1242           1530
##  2  2013     6    15     1432           1935      1137     1607           2120
##  3  2013     1    10     1121           1635      1126     1239           1810
##  4  2013     9    20     1139           1845      1014     1457           2210
##  5  2013     7    22      845           1600      1005     1044           1815
##  6  2013     4    10     1100           1900       960     1342           2211
##  7  2013     3    17     2321            810       911      135           1020
##  8  2013     6    27      959           1900       899     1236           2226
##  9  2013     7    22     2257            759       898      121           1026
## 10  2013    12     5      756           1700       896     1058           2020
## # i 336,766 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

Opérations sur les lignes

Combiné avec dplyr::slice(), dplyr::arrange() permet par exemple de sélectionner les trois vols ayant eu le plus de retard

```
flights |>
  arrange(desc(dep_delay)) |>
  slice(1:3)
```

```
## # A tibble: 3 x 19
##    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
## 1  2013     1     9      641            900      1301     1242           1530
## 2  2013     6    15     1432           1935      1137     1607           2120
## 3  2013     1    10     1121           1635      1126     1239           1810
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

Opérations sur les lignes

dplyr::slice_sample() permet de sélectionner aléatoirement un nombre de lignes ou une fraction des lignes d'un tableau. Ainsi si l'on veut choisir 5 lignes au hasard dans le tableau airports

```
airports |>
  slice_sample(n = 5)
```

```
## # A tibble: 5 x 8
##   faa   name                          lat    lon   alt    tz dst   tzone
##   <chr> <chr>                       <dbl>  <dbl> <dbl> <dbl> <chr> <chr>
## 1 AKN   King Salmon                  58.7 -157.     68    -9 A     America/Anchor~
## 2 FXE   Fort Lauderdale Executive   26.2  -80.2    13    -5 A     America/New_Yo~
## 3 FAF   Felker Aaf                  37.1  -76.6    12    -5 A     America/New_Yo~
## 4 AOH   Lima Allen County Airport   40.7  -84.0   975    -5 A     America/New_Yo~
## 5 DAW   Skyhaven Airport            43.3  -70.9   321    -5 A     America/New_Yo~
```

Opérations sur les lignes

Si l'on veut tirer au hasard 10% des lignes de flights

```
flights |>
  slice_sample(prop = .1)
```

```
## # A tibble: 33,677 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     2    11       47           2150       177      137           2256
##  2  2013     3     9      844            830        14     1152           1150
##  3  2013     5     9     1226           1205        21     1448           1425
##  4  2013     2    21     2120           2015        65     2236           2130
##  5  2013    12     1     1927           1910        17     2050           2054
##  6  2013     9     6     1829           1829         0     2140           2143
##  7  2013     2     1      705            700         5     1030           1019
##  8  2013     9    15      556            604        -8      850            906
##  9  2013     1     5      755            800        -5     1106           1106
## 10  2013    12    29     1444           1450        -6     1702           1649
## # i 33,667 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

Opérations sur les lignes

Ces fonctions sont utiles notamment pour faire de l'échantillonnage en tirant au hasard un certain nombre d'observations du tableau

Opérations sur les lignes

dplyr::distinct() filtre les lignes du tableau pour ne conserver que les lignes distinctes, en supprimant toutes les lignes en double

```
flights |>
  select(day, month) |>
  distinct()
```

```
## # A tibble: 365 x 2
##      day month
##    <int> <int>
##  1     1     1
##  2     2     1
##  3     3     1
##  4     4     1
##  5     5     1
##  6     6     1
##  7     7     1
##  8     8     1
##  9     9     1
## 10    10     1
## # i 355 more rows
```

Opérations sur les lignes

On peut lui spécifier une liste de variables : dans ce cas, pour toutes les observations ayant des valeurs identiques pour les variables en question, dplyr::distinct() ne conservera que la première d'entre elles

```
flights |>
  distinct(month, day)
```

```
## # A tibble: 365 x 2
##    month   day
##    <int> <int>
##  1     1     1
##  2     1     2
##  3     1     3
##  4     1     4
##  5     1     5
##  6     1     6
##  7     1     7
##  8     1     8
##  9     1     9
## 10     1    10
## # i 355 more rows
```

Opérations sur les colonnes

dplyr::select() permet de sélectionner des colonnes d'un tableau de données. Ainsi, si l'on veut extraire les colonnes lat et lon du tableau airports

```
airports |>
  select(lat, lon)
```

```
## # A tibble: 1,458 x 2
##      lat    lon
##    <dbl>  <dbl>
##  1  41.1  -80.6
##  2  32.5  -85.7
##  3  42.0  -88.1
##  4  41.4  -74.4
##  5  31.1  -81.4
##  6  36.4  -82.2
##  7  41.5  -84.5
##  8  42.9  -76.8
##  9  39.8  -76.6
## 10  48.1 -123.
## # i 1,448 more rows
```

Opérations sur les colonnes

Si on fait précéder le nom d'un -, la colonne est éliminée plutôt que sélectionnée

```
airports |>
  select(-lat, -lon)
```

```
## # A tibble: 1,458 x 6
##    faa   name                            alt    tz dst   tzone
##    <chr> <chr>                         <dbl> <dbl> <chr> <chr>
##  1 04G   Lansdowne Airport               1044    -5 A     America/New_York
##  2 06A   Moton Field Municipal Airport    264    -6 A     America/Chicago
##  3 06C   Schaumburg Regional              801    -6 A     America/Chicago
##  4 06N   Randall Airport                  523    -5 A     America/New_York
##  5 09J   Jekyll Island Airport             11    -5 A     America/New_York
##  6 0A9   Elizabethton Municipal Airport  1593    -5 A     America/New_York
##  7 0G6   Williams County Airport          730    -5 A     America/New_York
##  8 0G7   Finger Lakes Regional Airport    492    -5 A     America/New_York
##  9 0P2   Shoestring Aviation Airfield    1000    -5 U     America/New_York
## 10 0S9   Jefferson County Intl            108    -8 A     America/Los_Angeles
## # i 1,448 more rows
```

Opérations sur les colonnes

dplyr::select() comprend toute une série de fonctions facilitant la sélection de multiples colonnes. Par exemple, dplyr::starts_with(), dplyr::ends_width(), dplyr::contains() ou dplyr::matches() permettent d'exprimer des conditions sur les noms de variables

```
flights |>
  select(starts_with("dep_"))
```

```
## # A tibble: 336,776 x 2
##    dep_time dep_delay
##       <int>     <dbl>
##  1      517         2
##  2      533         4
##  3      542         2
##  4      544        -1
##  5      554        -6
##  6      554        -4
##  7      555        -5
##  8      557        -3
##  9      557        -3
## 10      558        -2
## # i 336,766 more rows
```

Opérations sur les colonnes

La syntaxe colonne1:colonne2 permet de sélectionner toutes les colonnes situées entre colonne1 et colonne2 incluses

```
flights |>
  select(year:day)
```

```
## # A tibble: 336,776 x 3
##     year month   day
##    <int> <int> <int>
##  1  2013     1     1
##  2  2013     1     1
##  3  2013     1     1
##  4  2013     1     1
##  5  2013     1     1
##  6  2013     1     1
##  7  2013     1     1
##  8  2013     1     1
##  9  2013     1     1
## 10  2013     1     1
## # i 336,766 more rows
```

Opérations sur les colonnes

dplyr::all_of() et dplyr::any_of() permettent de fournir une liste de variables à extraire sous forme de vecteur textuel. Alors que dplyr::all_of() renverra une erreur si une variable n'est pas trouvée dans le tableau de départ, dplyr::any_of() sera moins stricte

```
flights |>
  select(all_of(c("year", "month", "day")))
```

```
## # A tibble: 336,776 x 3
##     year month   day
##    <int> <int> <int>
##  1  2013     1     1
##  2  2013     1     1
##  3  2013     1     1
##  4  2013     1     1
##  5  2013     1     1
##  6  2013     1     1
##  7  2013     1     1
##  8  2013     1     1
##  9  2013     1     1
## 10  2013     1     1
## # i 336,766 more rows
```

Opérations sur les colonnes

dplyr::where() permets de sélectionner des variables à partir d'une fonction qui renvoie une valeur logique. Par exemple, pour sélectionner seulement les variables textuelles

```
flights |>
  select(where(is.character))
```

```
## # A tibble: 336,776 x 4
##    carrier tailnum origin dest
##    <chr>   <chr>   <chr>  <chr>
##  1 UA      N14228  EWR    IAH
##  2 UA      N24211  LGA    IAH
##  3 AA      N619AA  JFK    MIA
##  4 B6      N804JB  JFK    BQN
##  5 DL      N668DN  LGA    ATL
##  6 UA      N39463  EWR    ORD
##  7 B6      N516JB  EWR    FLL
##  8 EV      N829AS  LGA    IAD
##  9 B6      N593JB  JFK    MCO
## 10 AA      N3ALAA  LGA    ORD
## # i 336,766 more rows
```

Opérations sur les colonnes

dplyr::select() peut être utilisée pour réordonner les colonnes d'une table en utilisant la fonction dplyr::everything(), qui sélectionne l'ensemble des colonnes non encore sélectionnées

```
airports |>
  select(name, everything())
```

```
## # A tibble: 1,458 x 8
##    name                           faa     lat    lon   alt    tz dst   tzone
##    <chr>                          <chr> <dbl>  <dbl> <dbl> <dbl> <chr> <chr>
##  1 Lansdowne Airport              04G    41.1  -80.6  1044    -5 A     America/~
##  2 Moton Field Municipal Airport  06A    32.5  -85.7   264    -6 A     America/~
##  3 Schaumburg Regional            06C    42.0  -88.1   801    -6 A     America/~
##  4 Randall Airport                06N    41.4  -74.4   523    -5 A     America/~
##  5 Jekyll Island Airport          09J    31.1  -81.4    11    -5 A     America/~
##  6 Elizabethton Municipal Airport 0A9    36.4  -82.2  1593    -5 A     America/~
##  7 Williams County Airport        0G6    41.5  -84.5   730    -5 A     America/~
##  8 Finger Lakes Regional Airport  0G7    42.9  -76.8   492    -5 A     America/~
##  9 Shoestring Aviation Airfield   0P2    39.8  -76.6  1000    -5 U     America/~
## 10 Jefferson County Intl          0S9    48.1 -123.    108    -8 A     America/~
## # i 1,448 more rows
```

Opérations sur les colonnes

Pour réordonner des colonnes, on pourra aussi avoir recours à dplyr::relocate() en indiquant les premières variables. Il n'est pas nécessaire d'ajouter everything() car avec dplyr::relocate() toutes les variables sont conservées.

```
airports |>
  relocate(lon, lat, name)
```

```
## # A tibble: 1,458 x 8
##       lon   lat name                           faa     alt    tz dst   tzone
##     <dbl> <dbl> <chr>                          <chr> <dbl> <dbl> <chr> <chr>
##  1  -80.6  41.1 Lansdowne Airport              04G    1044    -5 A     America/~
##  2  -85.7  32.5 Moton Field Municipal Airport  06A     264    -6 A     America/~
##  3  -88.1  42.0 Schaumburg Regional            06C     801    -6 A     America/~
##  4  -74.4  41.4 Randall Airport                06N     523    -5 A     America/~
##  5  -81.4  31.1 Jekyll Island Airport          09J      11    -5 A     America/~
##  6  -82.2  36.4 Elizabethton Municipal Airport 0A9    1593    -5 A     America/~
##  7  -84.5  41.5 Williams County Airport        0G6     730    -5 A     America/~
##  8  -76.8  42.9 Finger Lakes Regional Airport  0G7     492    -5 A     America/~
##  9  -76.6  39.8 Shoestring Aviation Airfield   0P2    1000    -5 U     America/~
## 10 -123.   48.1 Jefferson County Intl          0S9     108    -8 A     America/~
## # i 1,448 more rows
```

Opérations sur les colonnes

Une variante de dplyr::select() est dplyr::rename()4, qui permet de renommer facilement des colonnes. On l'utilise en lui passant des paramètres de la forme nouveau_nom = ancien_nom.

```
airports |>
  rename(longitude = lon, latitude = lat)
```

```
## # A tibble: 1,458 x 8
##    faa   name                          latitude longitude   alt    tz dst   tzone
##    <chr> <chr>                            <dbl>     <dbl> <dbl> <dbl> <chr> <chr>
##  1 04G   Lansdowne Airport                 41.1     -80.6  1044    -5 A     Amer~
##  2 06A   Moton Field Municipal Airpo~     32.5     -85.7   264    -6 A     Amer~
##  3 06C   Schaumburg Regional              42.0     -88.1   801    -6 A     Amer~
##  4 06N   Randall Airport                  41.4     -74.4   523    -5 A     Amer~
##  5 09J   Jekyll Island Airport            31.1     -81.4    11    -5 A     Amer~
##  6 0A9   Elizabethton Municipal Airp~     36.4     -82.2  1593    -5 A     Amer~
##  7 0G6   Williams County Airport          41.5     -84.5   730    -5 A     Amer~
##  8 0G7   Finger Lakes Regional Airpo~     42.9     -76.8   492    -5 A     Amer~
##  9 0P2   Shoestring Aviation Airfield     39.8     -76.6  1000    -5 U     Amer~
## 10 0S9   Jefferson County Intl            48.1    -123.    108    -8 A     Amer~
## # i 1,448 more rows
```

Opérations sur les colonnes

Si les noms de colonnes comportent des espaces ou des caractères spéciaux, on peut les entourer de guillemets (“) ou de quotes inverses (‘) :

```
flights |>
  rename(
    "retard départ" = dep_delay,
    "retard arrivée" = arr_delay
  ) |>
  select(`retard départ`, `retard arrivée`)
```

```
## # A tibble: 336,776 x 2
##    `retard départ` `retard arrivée`
##              <dbl>            <dbl>
##  1               2               11
##  2               4               20
##  3               2               33
##  4              -1              -18
##  5              -6              -25
##  6              -4               12
##  7              -5               19
##  8              -3              -14
##  9              -3               -8
## 10              -2                8
## # i 336,766 more rows
```

Opérations sur les colonnes

La fonction dplyr::rename_with() permets de renommer plusieurs colonnes d'un coup en transmettant une fonction, par exemple toupper() qui passe tous les caractères en majuscule

```
airports |>
  rename_with(toupper)
```

```
## # A tibble: 1,458 x 8
##    FAA   NAME                            LAT    LON   ALT    TZ DST   TZONE
##    <chr> <chr>                         <dbl>  <dbl> <dbl> <dbl> <chr> <chr>
##  1 04G   Lansdowne Airport              41.1  -80.6  1044    -5 A     America/~
##  2 06A   Moton Field Municipal Airport  32.5  -85.7   264    -6 A     America/~
##  3 06C   Schaumburg Regional            42.0  -88.1   801    -6 A     America/~
##  4 06N   Randall Airport                41.4  -74.4   523    -5 A     America/~
##  5 09J   Jekyll Island Airport          31.1  -81.4    11    -5 A     America/~
##  6 0A9   Elizabethton Municipal Airport 36.4  -82.2  1593    -5 A     America/~
##  7 0G6   Williams County Airport        41.5  -84.5   730    -5 A     America/~
##  8 0G7   Finger Lakes Regional Airport  42.9  -76.8   492    -5 A     America/~
##  9 0P2   Shoestring Aviation Airfield   39.8  -76.6  1000    -5 U     America/~
## 10 0S9   Jefferson County Intl          48.1 -123.    108    -8 A     America/~
## # i 1,448 more rows
```

Opérations sur les colonnes

La fonction dplyr::pull() permet d'accéder au contenu d'une variable. C'est un équivalent aux opérateurs $ ou [[]]. On peut lui passer un nom de variable ou bien sa position

```
airports |>
  pull(alt) |>
  mean()
```

```
## [1] 1001.416
```

Opérations sur les colonnes

dplyr::mutate() permet de créer de nouvelles colonnes dans le tableau de données, en général à partir de variables existantes

```
airports <-
  airports |>
  mutate(alt_m = alt / 3.2808)
```

Opérations sur les colonnes

On peut créer plusieurs nouvelles colonnes en une seule fois, et les expressions successives peuvent prendre en compte les résultats des calculs précédents

```
flights <-
  flights |>
  mutate(
    distance_km = distance / 0.62137,
    vitesse = distance_km / air_time * 60
)
```

Opérations groupées

Un élément très important de dplyr est la fonction dplyr::group_by(). Elle permet de définir des groupes de lignes à partir des valeurs d'une ou plusieurs colonnes

```
flights |>
  group_by(month)
```

Opérations groupées

Par défaut ceci ne fait rien de visible, à part l'apparition d'une mention Groups dans l'affichage du résultat. Mais à partir du moment où des groupes ont été définis, les verbes comme dplyr::slice() ou dplyr::mutate() vont en tenir compte lors de leurs opérations

```
flights |>
  group_by(month) |>
  slice(1)
```

Opérations groupées

```
flights |>
  group_by(month) |>
  mutate(mean_delay_month = mean(dep_delay, na.rm = TRUE))
```

Opérations sur les colonnes

dplyr::group_by() peut aussi être utile avec dplyr::filter()

```
flights |>
  group_by(month) |>
  filter(dep_delay == max(dep_delay, na.rm = TRUE))
```

summarise()

dplyr::summarise() permet d'agréger les lignes du tableau en effectuant une opération résumée sur une ou plusieurs colonnes

```
flights |>
  summarise(
    retard_dep = mean(dep_delay, na.rm=TRUE),
    retard_arr = mean(arr_delay, na.rm=TRUE)
  )
```

summarise()

Cette fonction est en général utilisée avec dplyr::group_by()

```
flights |>
  group_by(month) |>
  summarise(
    max_delay = max(dep_delay, na.rm=TRUE),
    min_delay = min(dep_delay, na.rm=TRUE),
    mean_delay = mean(dep_delay, na.rm=TRUE)
  )
```

summarise()

dplyr::summarise() dispose d'une fonction spéciale dplyr::n(), qui retourne le nombre de lignes du groupe

```
flights |>
  group_by(dest) |>
  summarise(n = n())
```

count()

À noter que quand l'on veut compter le nombre de lignes par groupe, on peut utiliser directement la fonction dplyr::count()

```
flights |>
  count(dest)
```

Grouper selon plusieurs variables

On peut grouper selon plusieurs variables à la fois

```
flights |>
  group_by(month, dest) |>
  summarise(nb = n()) |>
  arrange(desc(nb))
```

Grouper selon plusieurs variables

On peut également compter selon plusieurs variables :

```
flights |>
  count(origin, dest) |>
  arrange(desc(n))
```

Grouper selon plusieurs variables

Lorsqu'on effectue un dplyr::group_by() suivi d'un dplyr::summarise(), le tableau résultat est automatiquement dégroupé de la dernière variable de regroupement

```
flights |>
  group_by(month, origin, dest) |>
  summarise(nb = n())
```

Grouper selon plusieurs variables

On peut à tout moment dégrouper un tableau à l'aide de dplyr::ungroup()

```
flights |>
  group_by(month, dest) |>
  summarise(nb = n()) |>
  ungroup() |>
  mutate(pourcentage = nb / sum(nb) * 100)
```