Logiciel R et programmation

Données

Ewen Gallic
Université de Rennes 1, 2016 - 2017

Types de données

Source : http://dilbert.com/strips/comic/2010-04-02/

Types de données

- R gère des objets ;
- Ces objets ont quatre caractéristiques :
  - un **nom**,
  - un **mode**,
  - une **longueur**,
  - un **contenu**.

Mode

- Mode = nature, type des éléments qui composent l'objet ;
- Accès par `mode()` ou `typeof()`.

```
a  <- 2
mode(a)
```

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

Il existe quatre principaux modes :

- `numeric` ;
- `character` :
- `logical` ;
- `complex`.

Mode : numeric

- Deux types de `numeric` :
  - `integers` (entiers),
  - `double` ou `real` (réels).

```
a <- 2.0
typeof(a)
```

```
## [1] "double"
```

```
is.integer(a) # a est un réel, pas un entier.
```

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

```
b <- 2
```

Mode : numeric

```
typeof(b)
```

```
## [1] "double"
```

```
c <- as.integer(b)
typeof(c)
```

```
## [1] "integer"
```

```
is.numeric(c) # c est bien un numérique.
```

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

Mode : character

- Placées entre guillemets simples `'` ou doubles `"`.

```
a <- "Hello world!"
a
```

```
## [1] "Hello world!"
```

```
typeof(a)
```

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

Mode : logical

- Deux valeurs : `TRUE` ou `FALSE`.

```
a <- 1 ; b <- 2
a < b
```

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

```
a == 1 # Test d'égalité
```

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

```
a != 1 # Test d'inégalité
```

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

Mode : logical

```
is.character(a)
```

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

```
(a <- TRUE)
```

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

```
(a <- T)
```

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

Mode : logical

Remarque

`TRUE` est converti en `1` et `FALSE` en `0`

```
TRUE + TRUE + FALSE + TRUE*TRUE
```

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

Mode : complex

- Partie réelle : `Re()` ;
- Partie imaginaire : `Im()` ;
- Création à l'aide de la lettre `i`.

```
1i
```

```
## [1] 0+1i
```

```
z <- 2+3i
```

Mode : complex

```
Re(z) # Partie réelle de z
```

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

```
Im(z) # Partie imaginaire de z
```

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

```
Mod(z) # Module de z
```

```
## [1] 3.605551
```

```
Arg(z) # Argument de z
```

```
## [1] 0.9827937
```

Longueur

- Nombre d'éléments que contient un objet ;
- `length()`.

```
a <- 1
length(a)
```

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

Données manquantes

- `NA` : ;
- type `logical`.

```
x <- NA
typeof(x)
```

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

```
is.na(x)
```

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

L'objet vide

- `NULL` : représente le vide ;
- Mode `NULL` ;
- Longueur `0` ;
- Différent de l'objet vide !

```
x <- NULL
typeof(x)
```

```
## [1] "NULL"
```

```
length(x)
```

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

L'objet vide

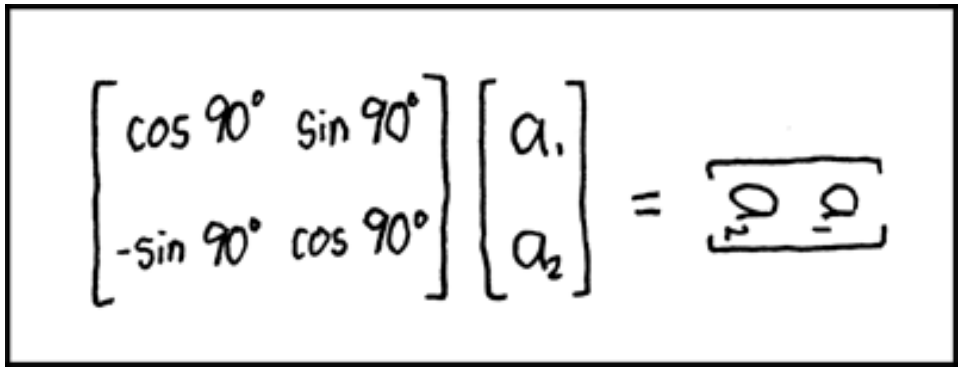
```
is.null(x)
```

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

```
is.null(list()) # La liste vide n'est pas NULL
```

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

Structure des données



Source : http://xkcd.com/184/

- Les structures servent à **organiser** les données ;
- Accès par la fonction `class()` ;
- Quelques structures principales :
  - vecteurs,
  - facteurs,
  - matrices, tableaux,
  - tableaux de données.

Vecteurs

- Objets les plus communs en R ;
- Tous les éléments d'un vecteur doivent être **de même type** ;
- Création (par exemple) par la fonction `c()`.

```
c(1,2,3)
```

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

On peut attribuer un nom aux éléments d'un vecteur, soit , soit .

Vecteurs

```
a <- c(nom = "Piketty", prenom = "Thomas", annee_naissance = "1971")
a
```

```
##             nom          prenom annee_naissance
##       "Piketty"        "Thomas"          "1971"
```

```
b <- c("Piketty", "Thomas", "1971")
b
```

```
## [1] "Piketty" "Thomas"  "1971"
```

```
names(b) <- c("nom", "prenom", "annee_naissance")
b
```

```
##             nom          prenom annee_naissance
##       "Piketty"        "Thomas"          "1971"
```

Vecteurs

- En cas de mélange de types, R convertit automatiquement dans le type le plus général.

```
c("deux", 1, TRUE)
```

```
## [1] "deux" "1"    "TRUE"
```

- L'objet est-il un vecteur ? Combien possède-t-il d'éléments ?

```
a <- c(2,1)
is.vector(a)
```

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

```
length(a)
```

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

Facteurs

- Pour les variables **qualitatives** ;
- Construction avec `factor()`

```
pays <- factor(c("France", "France", "Chine", "Espagne", "Chine"))
pays
```

```
## [1] France  France  Chine   Espagne Chine
## Levels: Chine Espagne France
```

```
class(pays)
```

```
## [1] "factor"
```

Facteurs

- Les **niveaux** s'obtiennent avec la fonction `levels()`.

```
levels(pays)
```

```
## [1] "Chine"   "Espagne" "France"
```

- Le changement de la référence s'effectue avec `relevel()`.

```
pays <- relevel(pays, ref = "Espagne")
pays
```

```
## [1] France  France  Chine   Espagne Chine
## Levels: Espagne Chine France
```

Facteurs

Avec des modalités **ordonnées**.

```
revenus <- ordered(c("<1500", ">2000", ">2000", "1500-2000",
                     ">2000", "<1500"),
                   levels = c("<1500", "1500-2000", ">2000"))
revenus
```

```
## [1] <1500     >2000     >2000     1500-2000 >2000     <1500
## Levels: <1500 < 1500-2000 < >2000
```

Matrices et tableaux

- Vecteurs avec un attribut de dimension (`dim`) valant 2 ;
- Création avec `matrix()` ou `array()` ;
- Remplissage par **colonnes** successives.
- Les données doivent être de même type.

```
(X <- matrix(1:6, ncol = 2, nrow = 3))
```

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

Matrices

Pour forcer le remplissage par **lignes** successives :

```
(Y <- matrix(1:6, ncol = 2, nrow = 3, byrow = TRUE))
```

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

Tableaux

```
(Z <- array(1:24, dim = c(2, 4, 3)))
```

```
## , , 1
##
##      [,1] [,2] [,3] [,4]
## [1,]    1    3    5    7
## [2,]    2    4    6    8
##
## , , 2
##
##      [,1] [,2] [,3] [,4]
## [1,]    9   11   13   15
## [2,]   10   12   14   16
##
## , , 3
##
##      [,1] [,2] [,3] [,4]
## [1,]   17   19   21   23
## [2,]   18   20   22   24
```

Listes

- Structures moins rigide ;
- Les **modes** des éléments peuvent être **différents**.

```
(personne <- list("Piketty", "Thomas", "1971"))
```

```
## [[1]]
## [1] "Piketty"
##
## [[2]]
## [1] "Thomas"
##
## [[3]]
## [1] "1971"
```

```
class(personne)
```

```
## [1] "list"
```

Listes

- Les éléments peuvent être **nommés** ;
- Si le nom contient une espace, il faut utiliser des guillemets.

```
personne <- list("nom de famille" = "Piketty", prenom = "Thomas", annee = 1971)
personne
```

```
## $`nom de famille`
## [1] "Piketty"
##
## $prenom
## [1] "Thomas"
##
## $annee
## [1] 1971
```

Listes

- Pour **aplatir** une liste, on utilise la fonction `unlist()` ;
- Le retour est un **vecteur** : attention aux conversions !

```
unlist(personne)
```

```
## nom de famille         prenom          annee
##      "Piketty"       "Thomas"         "1971"
```

Tableaux de données

- Structure largement utilisée par de nombreuses fonctions ;
- Il s'agit d'une liste composée de vecteurs ;
- Le mode des vecteurs peut varier ;
- Création avec `data.frame()`.

```
femmes <- data.frame(height = c(58, 59, 60, 61, 62, 63, 64, 65,
                                66, 67, 68, 69, 70, 71, 72),
                     weight = c(115, 117, 120, 123, 126, 129, 132,
                                135, 139, 142, 146, 150, 154, 159, 164))
```

Tableaux de données

```
head(femmes, 4)
```

```
##   height weight
## 1     58    115
## 2     59    117
## 3     60    120
## 4     61    123
```

```
is.data.frame(femmes)
```

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

```
class(femmes)
```

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

Tableaux de données

```
dim(femmes) # Dimensions
```

```
## [1] 15  2
```

```
nrow(femmes) # Nombre de lignes
```

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

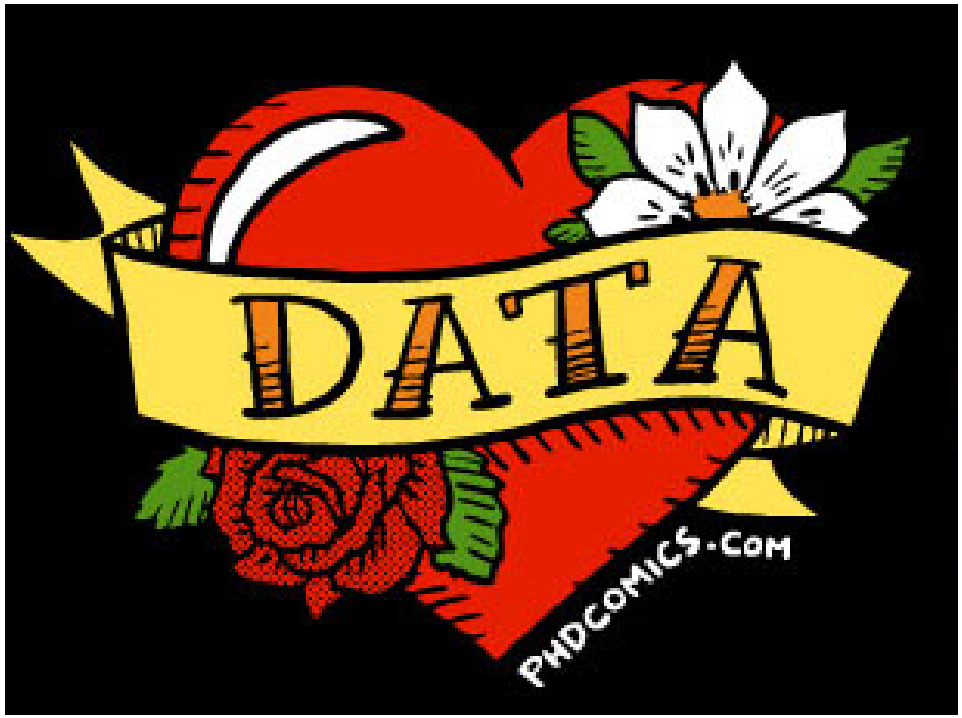
```
ncol(femmes) # Nombre de colonnes
```

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

Exercices

1. Que permet une liste par rapport à un vecteur ?
2. Quel sera le retour de `c("deux", 1, TRUE)` ? de `c(0, FALSE)` ? de `c(0, "FALSE")` ?
3. Créer un vecteur de facteurs, de taille 5, dont les modalités sont les suivantes : "Google", "Tumblr", "Twitter". La modalité de référence doit être "Google". Toutes les modalités doivent être présentes au moins une fois ;
4. Créer la matrice suivante sous `R` : $\begin{bmatrix} 39 & 66 & 13 \\ 66 & 168 & 28 \\ 13 & 28 & 5 \end{bmatrix}$;
5. En utilisant la matrice de la question 4, la transformer en ;
6. Créer la matrice de la fonction 4 à l'aide de la fonction `data.frame()`.

Importation, exportation et création de données



Source : http://phdcomics.com/comics.php

Un mot sur l'espace de travail

- Travail dans un **environnement de travail** ;
- Lecture et écriture par défaut dans le répertoire courant ;
- Le connaître avec `getwd()` ;
- Le changer avec `setwd()` ;
- Utilisateurs de Windows : attentions avec le caractère "`/`".

```
getwd() # Affichage de l'environnement de travail
setwd("C:\\Mes Documents\R") # Changement de cet environnement
```

Importation

- Avec des fichiers textes (ASCII) :
  - `read.table()`,
  - `scan()`
- Avec des formats propriétaires, d'autres doivent être chargés.

La fonction read.table()

- Pratique avec des données déjà organisées sous forme de tableau ;
- Range les données dans un ;
- Quelques paramètres principaux de la fonction :

PARAMÈTRE	DESCRIPTION
`file`	Nom de fichier ou chemin complet vers le fichier (peut être une URL)
`header`	Noms de variables en 1ère ligne ? (`FALSE` par défaut)
`sep`	Champ de séparation (caractère blanc par défaut)
`dec`	Caractère pour les décimales ("`.`" par défaut)
`row.names`	Vecteur des noms de lignes (numéro des lignes par défaut)
`na.strings`	Vecteur de chaînes de caractères indiquant les valeurs manquantes (`NA` par défaut)
`colClasses`	Vecteur de caractères indiquant les modes des colonnes (`R` fait au mieux par défaut)

La fonction read.table() : exemple

```
url <- "http://data.princeton.edu/wws509/datasets/phbirths.dat"
phbirths <- read.table(url)
```

```
head(phbirths)
```

```
##   black educ smoke gestate grams
## 1 FALSE    0  TRUE      40  2898
## 2  TRUE    0  TRUE      26   994
## 3 FALSE    2 FALSE      38  3977
## 4 FALSE    2  TRUE      37  3040
## 5 FALSE    2 FALSE      38  3523
## 6 FALSE    5  TRUE      40  3100
```

Exercices

1. Télécharger le fichier phbirths.dat, et le placer dans le répertoire de travail ;
2. Lire le fichier téléchargé à l'aide de la fonction `read.table()`.

La fonction read.table() : variantes

- Il existe des variantes de `read.table()` :

FONCTION	SÉPARATEUR DE CHAMPS	SÉPARATEUR DÉCIMAL
`read.csv()`	`","`	`"."`
`read.csv2()`	`";"`	`","`
`read.delim()`	`"\t"`	`"."`
`read.delim2()`	`"\t"`	`","`

La fonction scan()

- Plus souple que `read.table()` ;
- Quand les données ne sont pas organisées en tableau ;
- La nature des variables peut être spécifiée par `what`.
- Quelques paramètres principaux de la fonction :

PARAMÈTRE	DESCRIPTION
`file`	Nom de fichier ou chemin complet vers le fichier (peut être une URL)
`what`	Type des données lues
`nmax`	Nombre de données maximum à lire
`n`	Nombre de données à lire (pas de limite par défaut)
`sep`	Champ de séparation (caractère blanc par défaut)
`dec`	Caractère pour les décimales ("`.`" par défaut)
`skio`	Nombre de lignes à omettre avant de débuter l'importation
`na.strings`	Vecteur de chaînes de caractères indiquant les valeurs manquantes (`NA` par défaut)
`flush`	Si `TRUE`, commence l'importation de la ligne suivante après le dernier champs requis (permet d'avoir des commentaires après le dernier champ)

La fonction scan() : exemple

```
url <- "http://data.princeton.edu/wws509/datasets/phbirths.dat"
phbirths <- read.scan(url, what = "character")
```

```
head(phbirths)
```

```
## [1] "black"   "educ"    "smoke"   "gestate" "grams"   "1"
```

Exercices

1. À partir du fichier **phbirths.dat** téléchargé précédemment, utiliser la fonction `scan()` pour l'importer dans R.

La fonction scan() : remarque

- Appel à `scan()` sans paramètre file : données à entrer au clavier ;
- Touche `ENTREEE` pour valider chaque entrée ;
- Touche `ENTREE` sans saisie, ou `ESC` pour terminer.
- Fin de saisie en

```
tmp <- scan()
```

La fonction read.fwf()

- Fichiers dans un format à largeur fixée ( ) ;
- Mêmes paramètres que `read.table()` ;
- Paramètre supplémentaire : `widths` pour indiquer les largeur des colonnes.

```
url <- "http://editerna.free.fr/Enseignement/R/Data/data_fwf.txt"
phbirths <- read.fwf(url, widths = c(6,2,4))
```

```
head(phbirths, 4)
```

```
##       V1 V2   V3
## 1 FRANCE FR 14.0
## 2 FRANCE FR 23.0
## 3 FRANCE FR 32.9
## 4 ITALIE IT 15.9
```

Importation depuis Excel

- Fichiers `.xls` ou `.xlsx` ;
- Une fonction parmi d'autres : `read.xls()` (`gdata`) ;
- Choix optionnel de la feuille avec le paramètre `sheet` : par numéro ou par nom.

```
library("gdata")
# Récupérer le lien vers le fichier iris.xls
xlsfile <- file.path(path.package('gdata'),'xls','iris.xls')
iris <- read.xls(xlsfile) # Créé un fichier csv temporaire
head(iris, 3)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
```

Importation depuis d'autres formats

- `foreign` pour fichiers `DBF`, `STATA`, `STATA`, `SPSS`, etc. ;
- `sas7bdat`, fonction `read.sas7bdat()` pour fichiers `SAS`.

Exportation

- Fonction `write.table()` ;
- De nombreux paramètres modifiables (cf `?write.table`) ;
- Uniquement pour des `data.frame`, `vector` ou `matrix` :

```
write.table(donnees, file = "nom_fichier.txt", sep = ";")
```

- Sauvegarder n'importe quel type d'objets : `save()` ;
- Les importer dans `R` : `load()` :

```
save(d1, d2, d3, file = "nom_fichier.RData")
load(load("nom_fichier.RData"))
```

- Sauvegarder la session : `save.image()`.

Génération

- Génération de données :
  - séquences régulières,
  - séquences pseudo-aléatoires.

Séquences régulières : seq()

- Nombres également espacés : `seq()` en fournissant une valeur initiale et finale ;
- Incrément défini par `by` (1 par défaut);
- Longueur souhaitée : `length.out` (définit automatiquement l'incrément) ;

```
seq(1,3)
```

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

```
seq(1,3, by = 0.5)
```

```
## [1] 1.0 1.5 2.0 2.5 3.0
```

```
seq(1,3, length.out = 10)
```

```
##  [1] 1.000000 1.222222 1.444444 1.666667 1.888889 2.111111 2.333333 2.555556
##  [9] 2.777778 3.000000
```

Séquences régulières : ":"

- Pour des séquences dont l'incrément vaut 1, on peut utiliser l'opérateur "`:`" :

```
1:10
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
1.5:10
```

```
## [1] 1.5 2.5 3.5 4.5 5.5 6.5 7.5 8.5 9.5
```

Séquences régulières : rep()

- Séquence de valeur avec duplication du premier paramètre : `rep()`.

```
rep(1, 3) # Répète trois fois la valeur 1
```

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

```
rep(1:2, 3) # Répète trois fois la séquence 1:2
```

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

```
rep(1:2, each = 3) # Répète chaque élément de la séquence 1:2 trois fois
```

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

Séquences régulières : rep()

```
# Répète deux fois la séquence dans laquelle
# les éléments de la séquence 1:2 sont répétés trois fois
rep(1:2, 2, each = 3)
```

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

```
# Répète la séquence 1:2 jusqu'à ce que
# la longueur du résultat soit égale à 3 (le résultat peut être tronqué)
rep(1:2, length.out = 3)
```

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

Séquences régulières : seq_len()

- Pour créer une séquence allant de 1 au nombre passé au paramètre : `seq_len()`.

```
seq_len(4)
```

```
## [1] 1 2 3 4
```

Séquences régulières : sequence()

- Suite de nombre un peu spéciale ;
- Pour **chaque élément** du vecteur passé au paramètre `nvec`, une **séquence** allant de 1 à la valeur de l'élément est créée ;
- Les séquences créées sont ensuite **concaténées**.

```
sequence(2:4)
```

```
## [1] 1 2 1 2 3 1 2 3 4
```

```
sequence(c(3,5))
```

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

Séquences régulières : gl()

- Générer des séries de facteurs : `gl()` ;
- Deux paramètres requis :
  - `n` : nombre de niveaux souhaités,
  - `k` : nombre de réplications voulu ;
- Paramètres optionnels :
  - `labels` : étiquettes des modalités,
  - `ordered` : logique indiquant si le facteur est ordonné,
  - `length` : longueur du résultat souhaitée.

Séquences régulières : gl()

```
gl(2, 4)
```

```
## [1] 1 1 1 1 2 2 2 2
## Levels: 1 2
```

```
gl(2, 4, length = 10)
```

```
##  [1] 1 1 1 1 2 2 2 2 1 1
## Levels: 1 2
```

```
gl(2, 4, labels = c("Oui", "Non"))
```

```
## [1] Oui Oui Oui Oui Non Non Non Non
## Levels: Oui Non
```

Séquences régulières : expand.grid()

- Pour générer toutes les combinaisons possibles des vecteurs donnés en paramètre.

```
expand.grid(age = seq(18, 20), sexe = c("Femme", "Homme"), fumeur = c("Oui", "Non"))
```

```
##    age  sexe fumeur
## 1   18 Femme    Oui
## 2   19 Femme    Oui
## 3   20 Femme    Oui
## 4   18 Homme    Oui
## 5   19 Homme    Oui
## 6   20 Homme    Oui
## 7   18 Femme    Non
## 8   19 Femme    Non
## 9   20 Femme    Non
## 10  18 Homme    Non
## 11  19 Homme    Non
## 12  20 Homme    Non
```

Séquences pseudo-aléatoires

- Génération de données suivant une distribution de probabilité donnée ;
- Quatre fonctions pour chaque distribution, qui commencent par la lettre :
  - `r` : génération aléatoire,
  - `d` : densité (ou fonction de masse),
  - `p` : densité cumulée (ou fonction de répartition),
  - `q` : valeur de quantiles.

Séquences pseudo-aléatoires : exemple avec la loi Normale

```
rnorm(n = 1, mean = 0, sd = 1)
```

```
## [1] 0.5609951
```

```
dnorm(1) == 1/sqrt(2*pi) * exp(-1/2)
```

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

```
pnorm(1.96)
```

```
## [1] 0.9750021
```

```
qnorm(0.025) ; qnorm(0.975)
```

```
## [1] -1.959964
```

```
## [1] 1.959964
```

Séquences pseudo-aléatoires : lois de probabilités discrètes

- Quelques fonctions pour les lois de probabilité discrètes :

LOI	FONCTION
Binomiale	`rbinom(n, size, prob)`
Poisson	`rpois(n, lambda)`
Géométrique	`rgeom(n, prob)`
Hyper-géométrique	`rhyper(nn, m, n, k)`
Binomiale négative	`rnbinom(n, size, prob, mu)`

Séquences pseudo-aléatoires : lois de probabilités continues

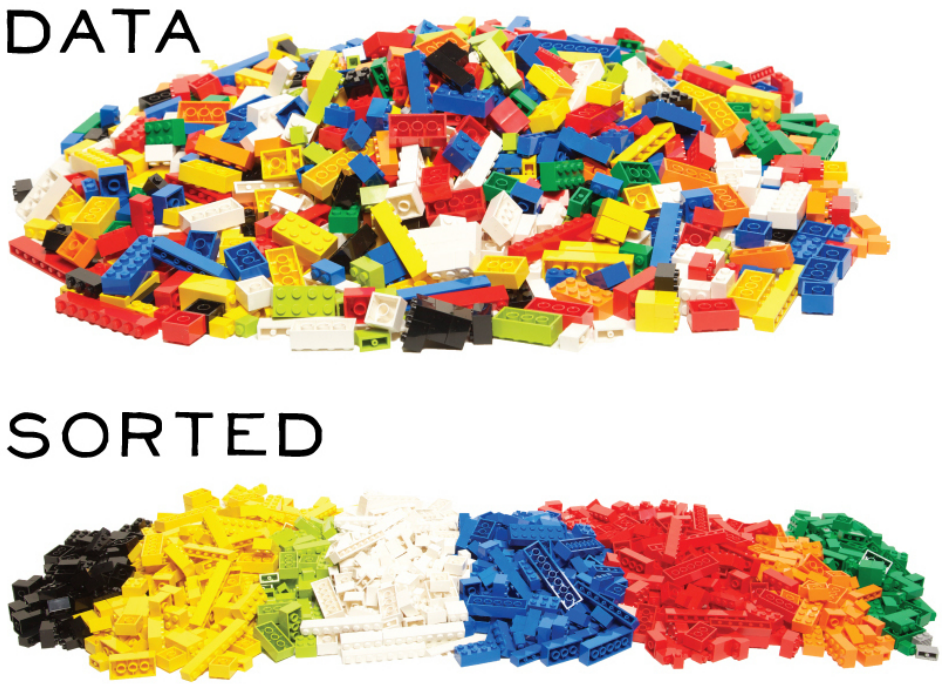
- Quelques fonctions pour les lois de probabilité continues :

LOI	FONCTION
Normale	`rnorm(n, mean = 0, sd = 1)`
Student	`rt(n, df, ncp)`
Khi-deux	`rchisq(n, df, ncp = 0)`
Fisher	`rf(n, df1, df2, ncp)`
Exponentielle	`rexp(n, rate = 1)`
Uniforme	`runif(n, min = 0, max = 1)`
Beta	`rbeta(n, shape1, shape2, ncp = 0)`
Logistique	`rlogis(n, location = 0, scale = 1)`
Log-Normale	`rlnorm(n, meanlog = 0, sdlog = 1)`
Gamma	`rgamma(n, shape, rate = 1, scale = 1/rate)`
Weibull	`rweibull(n, shape, scale = 1)`

Exercices

1. Quelle fonction utiliser pour importer un fichier `.csv`, dont les champs sont séparés par une tabulation, et dont la première ligne contient les noms de variables ?
2. Quelle fonction utiliser pour sauvregarder un en `.csv`, en séparant les champs par un point-virgule ?
3. Que retourne l'expression suivante : `c(FALSE, rep(1, 4))` ?
4. Proposer une solution pour créer un vecteur de nombres pairs allant de 0 à 10 ;
5. Générer une séquence de 10 nombres issus d'une loi normale centrée réduite ;
6. Trouver la valeur du quantile d'ordre 0.95 de la loi de Fisher à (4,26) degrés de liberté,
7. Calculer la probabilité d'obtenir une valeur plus grande que 2 lors d'un tirage selon une loi de Fisher de paramètres (4,26).

Manipulation de données



Source : http://www.hotbutterstudio.com/#/alps/

Opérateurs

- Il existe trois sortes d'opérateurs en `R` :
  - **arithmétiques**,
  - **de comparaison**,
  - **logiques**.

Opérateurs arithmétiques

- Les opérateurs **arithmétiques** agissent sur des **vecteurs** ou des **matrices** ;
- Ils opèrent sur des objets de type **numérique**, **complèxe** ou **logique** ;
- Les voici :
  - `+` : addition,
  - `-` : soustraction,
  - `*` : multiplication,
  - `/` : division,
  - `^` ou `**` : puissance,
  - `%%` : modulo,
  - `%/%` : division entière.

Opérateurs arithmétiques

```
x <- c(1, 2, 3, 4, 5) ; y <- c(2, 5, 2, 8, 1)
x + y  # Addition
```

```
## [1]  3  7  5 12  6
```

```
x - y  # Soustraction
```

```
## [1] -1 -3  1 -4  4
```

```
x * y  # Multiplication
```

```
## [1]  2 10  6 32  5
```

Opérateurs arithmétiques

```
x / y   # Division
```

```
## [1] 0.5 0.4 1.5 0.5 5.0
```

```
x^y  # Puissance
```

```
## [1]     1    32     9 65536     5
```

```
x %% y  # Modulo
```

```
## [1] 1 2 1 4 0
```

```
x %/% y # Division entière
```

```
## [1] 0 0 1 0 5
```

Opérateurs de comparaison

- Les opérateurs de **comparaison** agissent sur des **vecteurs**, des **tableaux de données** et des **listes** ;
- Le type de données n'est pas restreint comme pour les opérateurs arithmétiques ;
- Les voici :
  - `<` : inférieur à ;
  - `<=` : inféfieur ou égal à ;
  - `>` : supérieur à ;
  - `>=` : supérieur ou égal à ;
  - `==` : égal à;
  - `!=` : différent de ;

Opérateurs de comparaison

```
x <- seq_len(5)
x < 2  # Inférieur à
```

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

```
x <= 2  # Inférieur ou égal à
```

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

```
x > 2   # Supérieur à
```

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

Opérateurs de comparaison

```
x >= 2  # Supérieur ou égal à
```

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

```
x == 2  # Égal à
```

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

```
x != 2  # Différent de
```

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

Opérateurs de comparaison : attention !

Attention avec l'opérateur de comparaison == ! Deux objets égaux pour un humain ne le sont pas forcément pour l'ordinateur, à cause de la représentation des nombres et des approximations. Voici un exemple tiré de "R pour les débutants" d'Emmanuel Paradis (2002)

```
0.9 == (1.1 - 0.2)
```

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

```
all.equal(0.9, 1.1-0.2)
```

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

```
identical(0.9, 1.1-0.2)
```

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

Opérateurs logiques

- Les opérateurs **logiques** agissent sur un ou deux objets de type **logique** ;
- Les opérateurs **"ET"** et **"OU"** existent sous deux formes :
  - la forme simple (`&` et `|`) : opération sur chaque élément, résultat pour chaque élément,
  - la forme double (`&&` et `||`) : opération uniquement sur le premier élément des objets.

Opérateurs logiques

```
x <- c(TRUE, TRUE, FALSE, FALSE) ; y <- c(TRUE, FALSE, TRUE, FALSE)
!x  # 'Non' logique
```

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

```
x&y  # 'Et' logique
```

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

```
x&&y  # 'Et' logique (revient à faire x[1] & y[1])
```

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

Opérateurs logiques

```
x|y  # 'Ou' logique
```

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

```
x||y    # 'Ou' logique (revient à faire x[1] | y[1])
```

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

```
xor(x, y)   # 'Ou' exlusif
```

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

Opérateurs logiques : deux fonctions intéressantes

- Pour les vecteurs de types logiques, voici deux fonctions très utiles :
  - `any()` : au moins un des éléments du vecteur vaut `TRUE`,
  - `all()` : tous les éléments du vecteur valent `TRUE`.

```
x <- c(TRUE, FALSE)
any(x)
```

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

```
y <- c(FALSE, FALSE)
all(!y)
```

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

Attention au recyclage

- Avec une opération entre deux vecteurs de **tailles différentes**, `R` effectue un **recyclage** :
  - le vecteur de plus petite taille est **complété** par ses propres valeurs,
  - au final, les deux vecteurs ont la même taille.

```
x <- c(1, 2, 3)
y <- c(1, 2, 3, 4, 5, 6, 7)
x + y
```

```
## Warning in x + y: la taille d'un objet plus long n'est pas multiple de la taille
## d'un objet plus court
```

```
## [1] 2 4 6 5 7 9 8
```

```
# R a ajouté le vecteur c(1, 2, 3, 1, 2, 3, 1) à y
```

Exercices

Soient les observations `x <- c(2, 5, 6, 3, 2)`.

1. Multiplier `x` par 10 ;
2. Normaliser les données (utiliser les fonctions `min()` et `max()`) : $x_i = \frac{x_i - min(x)}{max(x) - min(x)}$;
3. Convertir les valeurs de `x` pour obtenir leur proportions (utiliser la fonction `sum()`) ;
4. Tester si les valeurs présentes appartiennent à l'ensemble $\{2, 3, 4\}$;
5. Tester si les valeurs sont supérieures ou égales à 2 ;
6. Tester si les valeurs sont inférieures ou égales à 5 ;
7. Tester si les valeurs sont comprises entre 2 et 5 inclus.

Accès aux valeurs, modifications

- R propose deux types d'accès aux données des différentes structures :
  - par **indices**,
  - par **noms**.

Accès par indices (vecteurs)

- Utiliation de l'indexation :
  - **numérique**,
  - **logique** ;
- Avec la fonction `"["()` ;
- Les paramètres sont les suivants :
  - le vecteur sur lequel on souhaite réaliser l'extraction,
  - un vecteur d'indices d'éléments à extraire, ou ne pas extraire, ou bien un vecteur d'éléments de type logique.

Accès par indices (vecteurs)

```
x <- c(4, 7, 3, 5, 0)
"["(x, 2)  # Extraire le second élément de x
```

```
## [1] 7
```

```
x[2]  # Une écriture plus commode pour extraire le second élément de x
```

```
## [1] 7
```

```
x[-2]   # Tous les éléments de x sauf le second
```

```
## [1] 4 3 5 0
```

```
x[3:4]  # Les troisième et quatrième éléments de x
```

```
## [1] 3 5
```

Accès par indices (vecteurs)

```
i <- 3:4
x[i] # On peut utiliser une variable contenant un vecteur d'indices
```

```
## [1] 3 5
```

```
x[c(F, T, F, F, F)]  # Le second élément de x
```

```
## [1] 7
```

```
x[x<1]  # Les éléments de x inférieurs à 1
```

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

```
x<1 # Il s'agit bien d'un vecteur de logiques
```

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

Accès par indices (vecteurs) : remarque

- Positions des éléments `TRUE` d'un vecteur logique : `which()` ;
- Position du (premier) **minimum** : `which.min()` d'un vecteur logique ou numérique ;
- Position du (premier) **maximum** : `which.max()` d'un vecteur logique ou numérique.

```
x <- c(2, 4, 5, 1, 7, 6)
which(x < 7 & x > 2)
```

```
## [1] 2 3 6
```

```
which.min(x)
```

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

Accès par indices (vecteurs) : remarque

```
which.max(x)
```

```
## [1] 5
```

```
x[which.max(x)]
```

```
## [1] 7
```

Accès par indices (vecteur) : remplacement

- Utilisation de la flèche d'assignation "`->`" ;
- Plusieurs éléments peuvent être modifiés en une seule instruction.

```
x <- seq_len(5)
x[2] <- 3
x
```

```
## [1] 1 3 3 4 5
```

```
x[2] <- x[3] <- 0
```

Accès par indices (vecteurs) : remplacement

```
x
```

```
## [1] 1 0 0 4 5
```

```
x[which(x == 0)] <- 10
x
```

```
## [1]  1 10 10  4  5
```

Accès par indices (matrices et tableaux de données)

- Utilisation de la même fonction : `"["()`, deux paramètres : `i` (lignes) et `j` (colonnes) ;
- Syntaxe : `x[i, j]` ;
- Omettre `i` retourne toutes les lignes pour les colonnes indicées `j`;
- L'omission de `j` retourne toutes les colonnes pour les lignes indicées `i`.

```
(x <- matrix(1:9, ncol = 3, nrow = 3))
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

```
x[1, 2]  # Élément de la ligne 1 et de la colonne 2
```

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

Accès par indices (matrices et tableaux de données)

```
i <- c(1,3) ; j <- 3
x[i,j]  # Éléments des lignes 1 et 3 de la troisième colonne
```

```
## [1] 7 9
```

```
x[, 2]  # Éléments de la seconde colonne
```

```
## [1] 4 5 6
```

```
x[1 ,]  # Éléments de la première ligne
```

```
## [1] 1 4 7
```

Accès par indices (matrices et tableaux de données)

```
x[, -c(1,3)]  # x sans les colonnes 1 et 3
```

```
## [1] 4 5 6
```

- Dans ce dernier exemple, un **vecteur** est retourné ;
- Si on souhaite conserver une structure de **matrice** : `drop = FALSE`.

```
x[, -c(1,3), drop = FALSE]
```

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

Accès par indices (matrices et tableaux de données)

- Les matrices sont des vecteurs empliés : on peut utiliser un seul paramètre à `"["()`.

```
x[c(1,3,7)]
```

```
## [1] 1 3 7
```

Accès par indices (matrices et tableaux de données) : remplacement

- Utilisation de la flèche d'assignation : "`->`".

```
x <- matrix(1:9, ncol = 3, nrow = 3)
x[1,2] <- 0
x
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    7
## [2,]    2    5    8
## [3,]    3    6    9
```

Accès par indices (matrices et tableaux de données) : remarque

- `which()` sur une matrice retourne les indices sous forme de **couples**.

```
which(x > 5, arr.ind = TRUE)
```

```
##      row col
## [1,]   3   2
## [2,]   1   3
## [3,]   2   3
## [4,]   3   3
```

Accès par indices (vecteurs logiques)

- Au lieu des indices, on peut utiliser des **vecteurs logiques**.

```
(x_logique <- matrix(c(TRUE, FALSE), ncol = 3, nrow = 3))
```

```
## Warning in matrix(c(TRUE, FALSE), ncol = 3, nrow = 3): la longueur des données
## [2] n'est pas un diviseur ni un multiple du nombre de lignes [3]
```

```
##       [,1]  [,2]  [,3]
## [1,]  TRUE FALSE  TRUE
## [2,] FALSE  TRUE FALSE
## [3,]  TRUE FALSE  TRUE
```

```
x[x_logique]
```

```
## [1] 1 3 5 7 9
```

Accès par indices (tableaux)

- Autant de paramètres à `"["()` que de dimensions dans le tableau.

```
z <-array(1:24, dim =c(2, 4, 3))
z[2,4,3]  # Élément de la ligne 2, colonne 4 de la dimension 3
```

```
## [1] 24
```

Accès par indices (listes)

- À nouveau la fonction `"["()` ;
- Le résultat est une liste.

```
personne <- list("Piketty", "Thomas", "1971")
personne[1]
```

```
## [[1]]
## [1] "Piketty"
```

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

```
## [[1]]
## [1] "Piketty"
##
## [[2]]
## [1] "1971"
```

Accès par indices (listes)

- L'accès au **contenu** d'un ou plusieurs éléments d'une liste se fait avec `"[["()`.

```
"[["(personne, 1)  # Premier élément de la liste
```

```
## [1] "Piketty"
```

```
personne[[1]]  # idem
```

```
## [1] "Piketty"
```

```
personne[[1,2]] # Ne fonctionne pas
```

```
## Error in personne[[1, 2]]: nombre d'indices incorrect
```

Accès par indices (listes)

```
personne[[c(1,2)]]  # Ne fonctionne pas non plus ici
```

```
## Error in personne[[c(1, 2)]]: indice hors limites
```

```
personne[1,2] # Ne fonctionne pas
```

```
## Error in personne[1, 2]: nombre de dimensions incorrect
```

Accès par indices (listes)

```
personne[c(1,2)]  # Retourne une liste de dim 2 contenant :
```

```
## [[1]]
## [1] "Piketty"
##
## [[2]]
## [1] "Thomas"
```

```
  # - le premier élément de la liste
    # - le second élément de la liste
```

Accès par indices (listes)

- Indexation récursive.

```
(l <- list("foo_1", "foo_2", c("foo_3", "foo_4")))
```

```
## [[1]]
## [1] "foo_1"
##
## [[2]]
## [1] "foo_2"
##
## [[3]]
## [1] "foo_3" "foo_4"
```

```
l[[3]]  # Troisième élément de la liste
```

```
## [1] "foo_3" "foo_4"
```

Accès par indices (listes)

```
l[[c(3,1)]]  # Premier élément du troisième élément
```

```
## [1] "foo_3"
```

```
l[[3]][1]   # Premier élément du troisième élément
```

```
## [1] "foo_3"
```

Accès par indices (listes) : remplacement

- À nouveau avec la flèche d'assignation "`->`".

```
l <- list(1, TRUE, "foo", list(matrix(1:4, ncol = 2), "foo_2"))
l[[4]] <- 2
l  # La liste en position 4 a été remplacée par un vecteur de longueur 1
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] TRUE
##
## [[3]]
## [1] "foo"
##
## [[4]]
## [1] 2
```

Accès par noms (vecteurs)

- Lorsque les éléments d'un vecteur sont nommés ;
- Avec la fonction `"["`.

```
personne <- c(nom = "Piketty", prenom = "Thomas", "annee de naissance" = "1971")
personne["nom"]
```

```
##       nom
## "Piketty"
```

```
names(personne)  # Accès aux noms des éléments du vecteur
```

```
## [1] "nom"                "prenom"             "annee de naissance"
```

Accès par noms (vecteurs)

```
names(personne) <- c("nom", "prenom", "naissance")  # Modification des noms
personne
```

```
##       nom    prenom naissance
## "Piketty"  "Thomas"    "1971"
```

```
names(personne) <- NULL  # Suppression des noms
personne
```

```
## [1] "Piketty" "Thomas"  "1971"
```

Accès par noms (listes, tableaux de données)

- Utilisation de `"["()` (et `"[["()` pour les listes) possible ;
- Utilisation de "`$`" également.

```
personne_liste <- list('nom de famille' = "Piketty", prenom = "Thomas",
                       annee = 1971)
personne_liste[["nom de famille"]]
```

```
## [1] "Piketty"
```

Accès par noms (listes, tableaux de données)

```
personne_liste$"nom de famille"  # Le nom contenant au moins une espace,
```

```
## [1] "Piketty"
```

```
# Il est nécessaire d'employer des guillemets
personne_liste$prenom
```

```
## [1] "Thomas"
```

```
names(personne_liste)  # Accès aux noms des éléments de la liste
```

```
## [1] "nom de famille" "prenom"         "annee"
```

Accès par noms (matrices, tableaux de données)

- **Nommer** les lignes et les colonnes : `rownames()` et `colnames()` ;
- La fonction `dimnames()` retourne une liste dont les éléments sont :
  - une liste contenant le vecteur des noms de lignes,
  - une liste contenant le vecteur des noms de colonnes.

```
femmes <-data.frame(height =c(58, 59, 60, 61, 62, 63, 64,
                              65, 66, 67, 68,69, 70, 71, 72),
                    weight =c(115, 117, 120, 123, 126, 129, 132, 135,
                              139,142, 146, 150, 154, 159, 164))
colnames(femmes)
```

```
## [1] "height" "weight"
```

Accès par noms (matrices, tableaux de données)

```
rownames(femmes)
```

```
##  [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10" "11" "12" "13" "14" "15"
```

```
dimnames(femmes)
```

```
## [[1]]
##  [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10" "11" "12" "13" "14" "15"
##
## [[2]]
## [1] "height" "weight"
```

Accès par noms (matrices, tableaux de données) : remplacement

- Encore une fois, on utilise la flèche d'assignation "`->`" ;
- Un message d'erreur apparaît (et le remplacement n'a pas lieu) si l'objet de remplacement n'est pas de la bonne longueur (pour les tableaux de données).

```
df <- data.frame(x = seq_len(3), y = rep(2,3), z = c(1,4,3))
df$x <- rep(NA, 3)
df
```

```
##    x y z
## 1 NA 2 1
## 2 NA 2 4
## 3 NA 2 3
```

Accès par noms (matrices, tableaux de données) : remplacement

```
df$x <- rep(1, 4)
```

```
## Error in `$<-.data.frame`(`*tmp*`, "x", value = c(1, 1, 1, 1)): replacement has 4 rows, dat
```

```
df  # La modification n'a pas été faite
```

```
##    x y z
## 1 NA 2 1
## 2 NA 2 4
## 3 NA 2 3
```

Accès par noms (matrices, tableaux de données) : remplacement

```
# Attention
is.list(df$x)
```

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

```
df$x <- list(4,3,2)
df$x  # La colonne "x" a changé de structure !
```

```
## [[1]]
## [1] 4
##
## [[2]]
## [1] 3
##
## [[3]]
## [1] 2
```

Exercices

1. Extraire le troisième élément du vecteur `x <-c(3, 5, 6, 9)` ;
2. Extraire les éléments supérieurs à 5 ;
3. Extraire les éléments qui ne sont pas dans l'intervalle $[1, 3]$;
4. Déterminer le maximum du vecteur `x` et donner sa position ;
5. Remplacer ce maximum par le minimum ;
6. Remplacer par la valeur `NA` les éléments négatifs de la matrice suivante :
   - $\begin{bmatrix} 1 & -1 & 5 \\ 5 & 7 & -9 \end{bmatrix}$;
7. Extraire la deuxième colonne de la matrice obtenue à l'issue de la question 6 ;
8. Extraire le 3e élément de la liste suivante : `L <- list(rnorm(10), LETTERS[1:26], month.abb)` ;
9. Extraire tous les éléments de la liste `L` sauf le second.

Exercices (suite)

1. Extraire le troisième élément de la liste `L`, en utilisant le nom de l'élément.
   - `L <- list(nombre = rnorm(10), lettre = LETTERS[1:26], mois = month.abb)` ;
2. Toujours en accédant par le nom, ajouter la valeur `10` aux valeurs du premier élément de `L`. Le résultat doit altérer `L` ;
3. Afficher les valeurs de la variable `y`du `df` :
   - `df <- data.frame(x = 1:10, y = letters[1:10], z = rev(letters[1:10]))` ;
4. Afficher uniquement les colonnes `x` et `z` du `df`.

Manipulation sur les matrices et tableaux de données

Fusion

- Juxtaposition : `cbind()` ;
- Superposition : `rbind()`.

```
(A <- matrix(c(1, -1, 0, 3), ncol = 2))
```

```
##      [,1] [,2]
## [1,]    1    0
## [2,]   -1    3
```

```
(B <- matrix(c(3, 2, 1, 1), ncol = 2))
```

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

Fusion

```
(C <- matrix(c(0, 3), ncol = 1))
```

```
##      [,1]
## [1,]    0
## [2,]    3
```

```
rbind(A,B)
```

```
##      [,1] [,2]
## [1,]    1    0
## [2,]   -1    3
## [3,]    3    1
## [4,]    2    1
```

Fusion

```
cbind(A,B)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    0    3    1
## [2,]   -1    3    2    1
```

```
cbind(A,B,C)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    0    3    1    0
## [2,]   -1    3    2    1    3
```

Fusion (data frame)

- `cbind()` pas optimal si colonne identique ;
- `merge()` plus adapté.

```
(pib <- data.frame(annee = 2010:2013, pib = c(1998.5, 2059.3, 2091.1, 2113.7)))
```

```
##   annee    pib
## 1  2010 1998.5
## 2  2011 2059.3
## 3  2012 2091.1
## 4  2013 2113.7
```

Fusion (data frame)

```
(importations <- data.frame(annee = 2010:2013,
                            importations = c(558.1, 625.3, 628.5, 629.1)))
```

```
##   annee importations
## 1  2010        558.1
## 2  2011        625.3
## 3  2012        628.5
## 4  2013        629.1
```

Fusion (data frame)

```
cbind(pib, importations)
```

```
##   annee    pib annee importations
## 1  2010 1998.5  2010        558.1
## 2  2011 2059.3  2011        625.3
## 3  2012 2091.1  2012        628.5
## 4  2013 2113.7  2013        629.1
```

```
merge(pib, importations)
```

```
##   annee    pib importations
## 1  2010 1998.5        558.1
## 2  2011 2059.3        625.3
## 3  2012 2091.1        628.5
## 4  2013 2113.7        629.1
```

Fusion (data frame)

- Pour fusionner en fonction de colonnes aux noms différents : `by.x` et `by.y` ;
- Si `all` (`all.x`) [`all.x`] vaut `TRUE` :
  - faire figurer toutes les valeurs présentes dans les deux (dans le premier) [dans le second],
  - les valeurs manquantes sont compélétes par `NA`.

```
(exportations <- data.frame(year = 2011:2013, exportations = c(572.6, 587.3, 597.8)))
```

```
##   year exportations
## 1 2011        572.6
## 2 2012        587.3
## 3 2013        597.8
```

Fusion (data frame)

```
merge(pib, exportations, by.x = c("annee"), by.y = c("year"))
```

```
##   annee    pib exportations
## 1  2011 2059.3        572.6
## 2  2012 2091.1        587.3
## 3  2013 2113.7        597.8
```

```
merge(pib, exportations, by.x = c("annee"), by.y = c("year"), all.x = TRUE)
```

```
##   annee    pib exportations
## 1  2010 1998.5           NA
## 2  2011 2059.3        572.6
## 3  2012 2091.1        587.3
## 4  2013 2113.7        597.8
```

Fusion (data frame)

- Rajout d'une colonne à un en fonction des valeurs d'une colonne de référence : `match()` et "`<-`".

```
df <- pib
match(df$annee, exportations$year)
```

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

```
df$exportations <- exportations[match(df$annee,
                                      exportations$year), "exportations"]
df
```

```
##   annee    pib exportations
## 1  2010 1998.5           NA
## 2  2011 2059.3        572.6
## 3  2012 2091.1        587.3
## 4  2013 2113.7        597.8
```

Exercices : données

```
(X <- data.frame(mois = month.name[c(1,6,1,6)], annee = c(2014, 2014, 2015, 2015),
                 val_1 = 1:4, val_2 = 5:8))
```

```
##      mois annee val_1 val_2
## 1 January  2014     1     5
## 2    June  2014     2     6
## 3 January  2015     3     7
## 4    June  2015     4     8
```

```
(Y <- data.frame(mois = rev(month.name[c(1,6,1,6)]), annee = c(2014, 2014, 2015, 2015),
                 val_3 = 9:12, val_2 = 13:16))
```

```
##      mois annee val_3 val_2
## 1    June  2014     9    13
## 2 January  2014    10    14
## 3    June  2015    11    15
## 4 January  2015    12    16
```

Exercices : questions

1. Créer une matrice `Z` qui contient `X` et `Y` côte-à-côte ;
2. Que se passe-t-il si on évalue l'instruction suivante : `rbind(X,Y)` ;
3. En utilisant la fonction `merge()`, fusionner les `X` et `Y` par année et par mois ;
4. À l'aide de la fonction `match()`, ajouter à `X` les valeurs de la colonne `var_5` du `Z`, en effectuant l'appariement sur la colonne des années :

```
(Z <- data.frame(year = c(2013, 2014), val_5 = c("A", "B")))
```

```
##   year val_5
## 1 2013     A
## 2 2014     B
```

Transformation sur les colonnes

- Pénible de devoir réécrire à chaque fois le nom du :

```
chomeurs <- data.frame(annee = 2012:2008,
                       nb_chomeurs = c(2.811, 2.604, 2.635, 2.573, 2.064),
                       pop_active = c(28.328, 28.147, 28.157, 28.074, 27.813))
chomeurs$taux_chomage_0 <- chomeurs$nb_chomeurs / chomeurs$pop_active * 100
chomeurs
```

```
##   annee nb_chomeurs pop_active taux_chomage_0
## 1  2012       2.811     28.328       9.923044
## 2  2011       2.604     28.147       9.251430
## 3  2010       2.635     28.157       9.358241
## 4  2009       2.573     28.074       9.165064
## 5  2008       2.064     27.813       7.420990
```

Transformation sur les colonnes

- Existence de la fonction `attach()` ;
- Elle rend visible les colonnes d'un dans l'espace de travail ;
- La fonction `detach()` les cache ;
- Mais ces fonctions apportent souvent plus de problèmes que de solutions...

```
data(quakes)
quakes <- quakes[1:4,] # On ne prend que quelques observations pour l'exemple
quakes
```

```
##      lat   long depth mag stations
## 1 -20.42 181.62   562 4.8       41
## 2 -20.62 181.03   650 4.2       15
## 3 -26.00 184.10    42 5.4       43
## 4 -17.97 181.66   626 4.1       19
```

Transformation sur les colonnes

```
lat  # lat n'est pas dans l'espace de travail
```

```
## Error in eval(expr, envir, enclos): objet 'lat' introuvable
```

```
attach(quakes)
head(lat)  # maintenant il l'est
```

```
## [1] -20.42 -20.62 -26.00 -17.97 -20.42 -19.68
```

```
detach(quakes)
lat # il est à nouveau masqué
```

```
## Error in eval(expr, envir, enclos): objet 'lat' introuvable
```

Transformation sur les colonnes

- Trois fonctions à peu près similaires :
  - `with()` (syntaxe un peu moin lisible),
  - `within()`,
  - `transform()`.

```
# Changements simples
chomeurs$tx_chomage_1 <- with(chomeurs, nb_chomeurs/pop_active*100)
chomeurs <- within(chomeurs, tx_chomage_2 <- nb_chomeurs/pop_active*100)
chomeurs <- transform(chomeurs, tx_chomage_3 = nb_chomeurs/pop_active*100)
```

Transformation sur les colonnes

```
# Changements multiples
chomeurs[c("log_nb_chomeurs", "log_pop_active")] <- with(chomeurs, list(
  log(nb_chomeurs), log(pop_active)
  ))
chomeurs <- within(chomeurs,{
  log_nb_chomeurs_2 <- log(nb_chomeurs)
  log_pop_active_2 <- log(pop_active)
  })
chomeurs <- transform(chomeurs,
                      log_nb_chomeurs_3 = log(nb_chomeurs),
                      log_pop_active_3 = log(pop_active)
                      )
```

Transformation sur les colonnes

```
head(chomeurs, 3)
```

```
##   annee nb_chomeurs pop_active taux_chomage_0 tx_chomage_1 tx_chomage_2
## 1  2012       2.811     28.328       9.923044     9.923044     9.923044
## 2  2011       2.604     28.147       9.251430     9.251430     9.251430
## 3  2010       2.635     28.157       9.358241     9.358241     9.358241
##   tx_chomage_3 log_nb_chomeurs log_pop_active log_pop_active_2
## 1     9.923044       1.0335403       3.343851         3.343851
## 2     9.251430       0.9570487       3.337441         3.337441
## 3     9.358241       0.9688832       3.337796         3.337796
##   log_nb_chomeurs_2 log_nb_chomeurs_3 log_pop_active_3
## 1         1.0335403         1.0335403         3.343851
## 2         0.9570487         0.9570487         3.337441
## 3         0.9688832         0.9688832         3.337796
```

Agrégation

- Fonction `aggregate()` ;
- Découpe un en fonction d'un ou plusieurs **facteurs** ;
- `by` : la ou les colonnes servant à la décomposition ;
- `FUN` : indique la modification à apporter aux sous population.

```
chomage <- data.frame(region = rep(c(rep("Bretagne", 4), rep("Corse", 2)), 2),
                      departement = rep(c("Cotes-d'Armor", "Finistere",
                                          "Ille-et-Vilaine", "Morbihan",
                                          "Corse-du-Sud", "Haute-Corse"), 2),
                      annee = rep(c(2011, 2010), each = 6),
                      ouvriers = c(8738, 12701, 11390, 10228, 975, 1297,
                                   8113, 12258, 10897, 9617, 936, 1220),
                      ingenieurs = c(1420, 2530, 3986, 2025, 259, 254,
                                     1334, 2401, 3776, 1979, 253, 241))
```

Agrégation

```
chomage
```

```
##      region     departement annee ouvriers ingenieurs
## 1  Bretagne   Cotes-d'Armor  2011     8738        1420
## 2  Bretagne       Finistere  2011    12701        2530
## 3  Bretagne Ille-et-Vilaine  2011    11390        3986
## 4  Bretagne        Morbihan  2011    10228        2025
## 5     Corse    Corse-du-Sud  2011      975         259
## 6     Corse     Haute-Corse  2011     1297         254
## 7  Bretagne   Cotes-d'Armor  2010     8113        1334
## 8  Bretagne       Finistere  2010    12258        2401
## 9  Bretagne Ille-et-Vilaine  2010    10897        3776
## 10 Bretagne        Morbihan  2010     9617        1979
## 11    Corse    Corse-du-Sud  2010      936         253
## 12    Corse     Haute-Corse  2010     1220         241
```

Agrégation

```
# Agrégation par année
aggregate(chomage[, c("ouvriers", "ingenieurs")],
          by = list(annee = chomage$annee), FUN = sum)
```

```
##   annee ouvriers ingenieurs
## 1  2010    43041       9984
## 2  2011    45329      10474
```

```
# Agrégation par année et par région
aggregate(chomage[, c("ouvriers", "ingenieurs")],
          by = list(annee = chomage$annee, region = chomage$region), FUN = sum)
```

```
##   annee   region ouvriers ingenieurs
## 1  2010 Bretagne    40885       9490
## 2  2011 Bretagne    43057       9961
## 3  2010    Corse     2156        494
## 4  2011    Corse     2272        513
```

Tri

- Tri par odre de valeurs croissantes ou décroissantes : `order()` ;
- Le résultat est un vecteur de rangs ;
- Tri selon l'ordre alphanumérique ;

```
df <- data.frame(nom = c("Durand", "Martin", "Martin", "Martin", "Durand"),
                 prenom = c("Sonia", "Serge", "Julien-Yacine", "Victor", "Emma"),
                 note = c(23, 18, 17, 17, 19))
df
```

```
##      nom         prenom note
## 1 Durand          Sonia   23
## 2 Martin          Serge   18
## 3 Martin Julien-Yacine   17
## 4 Martin         Victor   17
## 5 Durand           Emma   19
```

Tri

```
# Ordonner par notes décroissantes
order(df$note, decreasing = TRUE)
```

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

```
df[order(df$note), ]
```

```
##      nom        prenom note
## 3 Martin Julien-Yacine   17
## 4 Martin        Victor   17
## 2 Martin         Serge   18
## 5 Durand          Emma   19
## 1 Durand         Sonia   23
```

Tri

```
# Ordonner par ordre alphabétique des noms puis des prénoms
df[with(df, order(nom, prenom)), ]
```

```
##      nom         prenom note
## 5 Durand          Emma   19
## 1 Durand         Sonia   23
## 3 Martin Julien-Yacine   17
## 2 Martin         Serge   18
## 4 Martin        Victor   17
```

Tri : remarque

- `sort()` retourne un vecteur trié par valeurs croissantes ou décroissantes ;
- `order()` retourne les rangs du classement.

Tri : `plyr`

- La fonction `arrange()` du `plyr` est pratique ;
- Il n'y a pas de paramètre `decreasing` : on fait appel à la fonction `desc()`.

```
library(plyr)
# Ordonner par notes décroissantes
arrange(df, desc(note))
```

```
##      nom         prenom note
## 1 Durand          Sonia   23
## 2 Durand           Emma   19
## 3 Martin          Serge   18
## 4 Martin Julien-Yacine   17
## 5 Martin         Victor   17
```

Tri : plyr

```
# Ordonner par ordre alphabétique des noms puis inverse des prénoms
arrange(df, nom, desc(prenom))
```

```
##      nom         prenom note
## 1 Durand          Sonia   23
## 2 Durand           Emma   19
## 3 Martin         Victor   17
## 4 Martin          Serge   18
## 5 Martin Julien-Yacine   17
```

Stacking et unstacking

- Empiler les contenus des colonnes d'un tableau dans un vecteur : `stack()` ;
- Opération inverse : `unstack()`.

```
patients <- data.frame(traitement_1 = c(4, 2, 7),
                       traitement_2 = c(8, 5, 0),
                       traitement_3 = c(0, 5, 6))
patients
```

```
##   traitement_1 traitement_2 traitement_3
## 1            4            8            0
## 2            2            5            5
## 3            7            0            6
```

Stacking et unstacking

```
stack(patients)
```

```
##   values          ind
## 1      4 traitement_1
## 2      2 traitement_1
## 3      7 traitement_1
## 4      8 traitement_2
## 5      5 traitement_2
## 6      0 traitement_2
## 7      0 traitement_3
## 8      5 traitement_3
## 9      6 traitement_3
```

Stacking et unstacking

```
unstack(stack(patients))
```

```
##   traitement_1 traitement_2 traitement_3
## 1            4            8            0
## 2            2            5            5
## 3            7            0            6
```

Tableaux longs, tableaux larges

- Pour passer d'un tableau long à un tableau large, et vice-versa : `reshape2` ;
- Large vers long : `melt()` ;
- Long vers large : `dcast()`.

```
pop <- data.frame(ville = c("Paris", "Paris", "Lyon", "Lyon"),
                  arrondissement = c(1, 2, 1, 2),
                  pop_municipale = c(17443, 22927, 28932, 30575),
                  pop_totale = c(17620, 23102, 29874, 31131))
pop
```

```
##   ville arrondissement pop_municipale pop_totale
## 1 Paris              1          17443      17620
## 2 Paris              2          22927      23102
## 3  Lyon              1          28932      29874
## 4  Lyon              2          30575      31131
```

Tableaux longs, tableaux larges : `melt()`

- `id.vars` : vecteur contenant les noms des variables à conserver ;
- `value.name` : renommer la colonne du retourné dans laquelle les valeurs sont empilées ;
- `variable.name` : renommer la colonne du retourné contenant les modalités.

```
library(reshape2)
pop_long <- melt(pop, id.vars=c("ville", "arrondissement"),
                 value.name = "population",
                 variable.name="type_population")
```

Tableaux longs, tableaux larges : `melt()`

```
pop_long
```

```
##   ville arrondissement type_population population
## 1 Paris              1  pop_municipale      17443
## 2 Paris              2  pop_municipale      22927
## 3  Lyon              1  pop_municipale      28932
## 4  Lyon              2  pop_municipale      30575
## 5 Paris              1      pop_totale      17620
## 6 Paris              2      pop_totale      23102
## 7  Lyon              1      pop_totale      29874
## 8  Lyon              2      pop_totale      31131
```

Tableaux longs, tableaux larges : `dcast()`

- `formula` : objet de classe formule (de type `x ~ y`) précisant les colonnes à conserver en fonction de celles contenant les noms des nouvelles colonnes dans lesquelles on souhaite placer les mesures ;
- `value.var` : renommer la colonne dans laquelle se trouvent les valeurs.

```
dcast(pop_long, formula = ville + arrondissement ~ type_population,
      value.var="population")
```

```
##   ville arrondissement pop_municipale pop_totale
## 1  Lyon              1          28932      29874
## 2  Lyon              2          30575      31131
## 3 Paris              1          17443      17620
## 4 Paris              2          22927      23102
```

Exercices : données

```
df <- data.frame(region = rep(c(rep("Bretage",4), rep("Corse", 2)), 2),
                 dep = rep(c("Cotes-d'Armor", "Finistere",
                             "Ille-et-Vilaine", "Morbihan",
                             "Corse-du-Sud", "Haute-Corse"), 2),
                 annee = rep(2012:2013, each = 6),
                 ensoleillement = c(1586, 1545, 1762, 1792, 2933, 2735,
                                    1639, 1661, 1729, 1770, 2715, 2606),
                 precipitations  = c(733, 1311, 788, 920, 876, 676,
                                     679, 1166, 736, 788, 871, 729))
```

Exercices : questions

1. Creer les variables `ensoleillement_c` et `precipitations_c` dans le `df`, en utilisant la fonction `within()`. Ces deux variables doivent représenter l'écart à la moyenne de la variable `ensoleillement` et `precipitations` respectivement ;
2. Calculer la moyenne de précipitations pour chaque département et chaque année ;
3. Trier le résultat de la question précédente par année décroissante et ordre alphabétique des régions ;
4. À partir de `df`, créer un en long, où chaque ligne correspond à la durée d'ensoleillement ou les précipitations par région, département et année. Chaque ligne doit indiquer si la valeur concerne l'ensoleillement ou les précipitations.

Calculs matriciels

Considérons trois matrices A, B et C de même type, ainsi qu'un scalaire a.

```
(A <- matrix(c(1, 3, 2, 2, 2, 1, 3, 1, 3), ncol = 3))
```

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

```
(B <- matrix(c(4, 6, 4, 5, 5, 6, 6, 4, 5), ncol = 3))
```

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

Calculs matriciels

```
(C <- matrix(c(0, 3, 1), ncol = 1))
```

```
##      [,1]
## [1,]    0
## [2,]    3
## [3,]    1
```

```
a <- 2
```

Calculs matriciels : addition

```
A+a # Avec un scalaire
```

```
##      [,1] [,2] [,3]
## [1,]    3    4    5
## [2,]    5    4    3
## [3,]    4    3    5
```

```
A+B # De deux matrices
```

```
##      [,1] [,2] [,3]
## [1,]    5    7    9
## [2,]    9    7    5
## [3,]    6    7    8
```

Calculs matriciels : soustraction

```
A-a # Avec un scalaire
```

```
##      [,1] [,2] [,3]
## [1,]   -1    0    1
## [2,]    1    0   -1
## [3,]    0   -1    1
```

```
A-B # De deux matrices
```

```
##      [,1] [,2] [,3]
## [1,]   -3   -3   -3
## [2,]   -3   -3   -3
## [3,]   -2   -5   -2
```

Calculs matriciels : multiplication par un scalaire

```
a*A
```

```
##      [,1] [,2] [,3]
## [1,]    2    4    6
## [2,]    6    4    2
## [3,]    4    2    6
```

Calculs matriciels : division par un scalaire

```
A/a
```

```
##      [,1] [,2] [,3]
## [1,]  0.5  1.0  1.5
## [2,]  1.5  1.0  0.5
## [3,]  1.0  0.5  1.5
```

Calculs matriciels : transposée

```
t(A)
```

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

Calculs matriciels : conjuguée

```
Conj(A)
```

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

Calculs matriciels : multiplication de deux matrices

```
A%*%C
```

```
##      [,1]
## [1,]    9
## [2,]    7
## [3,]    6
```

Attention, `*` effectue une multiplication termes à termes, si les deux matrices sont de même dimension.

```
A*B
```

```
##      [,1] [,2] [,3]
## [1,]    4   10   18
## [2,]   18   10    4
## [3,]    8    6   15
```

Calculs matriciels : inversion

```
solve(A)
```

```
##             [,1]  [,2]       [,3]
## [1,] -0.41666667  0.25  0.3333333
## [2,]  0.58333333  0.25 -0.6666667
## [3,]  0.08333333 -0.25  0.3333333
```

Calculs matriciels : division matricielle

```
B %/% solve(A)
```

```
##      [,1] [,2] [,3]
## [1,]  -10   19   18
## [2,]   10   20   -7
## [3,]   48  -25   15
```

Calculs matriciels : produit avec la transposition

```
crossprod(A,B)
```

```
##      [,1] [,2] [,3]
## [1,]   30   32   28
## [2,]   24   26   25
## [3,]   30   38   37
```

```
t(A) %*% B # Moins rapide
```

```
##      [,1] [,2] [,3]
## [1,]   30   32   28
## [2,]   24   26   25
## [3,]   30   38   37
```

Calculs matriciels : déterminant, diagonale et trace

```
det(A)
```

```
## [1] -12
```

```
diag(A)
```

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

```
sum(diag(A))
```

```
## [1] 6
```

Exercices : données

Considérons le modèle suivant :

$$\mathrm{mpg} = \beta_0 + \beta_1 \mathrm{hp} + \beta_2 \mathrm{wt} + \varepsilon,$$

où mpg est un vecteur de valeurs de consommation de carburant (miles par gallons), hp et wt des vecteurs de puissance (nombre de chevaux) et de masse du véhicule (en millième de livres) respectivement. Le vecteur ε est un terme d'erreurs.

On cherche à estimer les paramètres β_0, β_1 et β_3 par la méthode des moindres carrés ordinaires.

Les données sont contenues dans le `mtcars`.

Exercices : question

1. À partir de `mtcars`, créer le vecteur `y` contenant les valeurs de consommation (`mpg`) ;
2. Créer également la matrice `X` qui doit contenir 3 colonnes : une pour la constante, une autre pour la puissance (`ht`) et une dernière pour la masse (`wt`) ;
3. Calculer la transposée de X : X^T ;
4. Calculer le produit matriciel : $X^T X$ (de deux fa\c cons);
5. Calculer l'inverse de ce produit matriciel : $(X^T X)^{-1}$;
6. Calculer le produit matriciel $X^T y$;
7. Donner les estimations des paramètres : $(X^T X)^{-1} X^T y$.

Manipulation des data.table

- Le `data.table` propose une structure de données ressemblant au `data.table` :
- Pourquoi ?
  - réduire la longueur du code,
  - proposer une syntaxe plus facile à lire et écrire,
  - réduire les temps de calculs ;
- Syntaxe proche du SQL.

```
library("data.table")
```

Création, conversion

- Création : comme un , mais avec la fonction `data.table()` ;
- Conversion : idem, avec `data.table()`.

```
chomage <- data.table(region = rep(c(rep("Bretagne", 4), rep("Corse", 2)), 2),
                      departement = rep(c("Cotes-d'Armor", "Finistere",
                                          "Ille-et-Vilaine", "Morbihan",
                                          "Corse-du-Sud", "Haute-Corse"), 2),
                      annee = rep(c(2011, 2010), each = 6),
                      ouvriers = c(8738, 12701, 11390, 10228, 975, 1297,
                                   8113, 12258, 10897, 9617, 936, 1220),
                      ingenieurs = c(1420, 2530, 3986, 2025, 259, 254,
                                     1334, 2401, 3776, 1979, 253, 241))
```

Création, conversion

```
chomage
```

```
##       region     departement annee ouvriers ingenieurs
##  1: Bretagne   Cotes-d'Armor  2011     8738       1420
##  2: Bretagne       Finistere  2011    12701       2530
##  3: Bretagne Ille-et-Vilaine  2011    11390       3986
##  4: Bretagne        Morbihan  2011    10228       2025
##  5:    Corse    Corse-du-Sud  2011      975        259
##  6:    Corse     Haute-Corse  2011     1297        254
##  7: Bretagne   Cotes-d'Armor  2010     8113       1334
##  8: Bretagne       Finistere  2010    12258       2401
##  9: Bretagne Ille-et-Vilaine  2010    10897       3776
## 10: Bretagne        Morbihan  2010     9617       1979
## 11:    Corse    Corse-du-Sud  2010      936        253
## 12:    Corse     Haute-Corse  2010     1220        241
```

Création, conversion

```
pop <- data.table(region = rep(c("Bretagne", "Alsace"), each = 2),
                  annee = rep(c(2010, 2011), 2),
                  population = c(3199066, 3217767, 1845687, 1852325))

population <- data.table(pop)
population
```

```
##      region annee population
## 1: Bretagne  2010    3199066
## 2: Bretagne  2011    3217767
## 3:   Alsace  2010    1845687
## 4:   Alsace  2011    1852325
```

Création, conversion

- Aperçu des `data.table` en mémoire : tables().

```
tables()
```

```
##       NAME        NROW MB COLS                                                KEY
## [1,] chomage       12 1  region,departement,annee,ouvriers,ingenieurs
## [2,] pop            4 1  region,annee,population
## [3,] population     4 1  region,annee,population
## Total: 3MB
```

Création, conversion

- Conversion possible d'un `data.table` en `data.frame`.

```
pop <- data.frame(population)
pop
```

```
##     region annee population
## 1 Bretagne  2010    3199066
## 2 Bretagne  2011    3217767
## 3   Alsace  2010    1845687
## 4   Alsace  2011    1852325
```

Accès aux données

- Accès par numéro de ligne avec la fonction `"["()` (voir `?"[".data.table()`) ;

```
chomage[1,]
```

```
##      region   departement annee ouvriers ingenieurs
## 1: Bretagne Cotes-d'Armor  2011     8738       1420
```

Accès aux données

- Attention, pour les colonnes, `"["()` ne fonctionne pas comme quand on l'utilise sur un
  .

```
# Retourne un résultat pas forcément attendu
chomage[,1]
```

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

```
chomage[1,1]
```

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

Accès aux données

- L'accès aux données en mentionnant la colonne se fait à l'aide du **nom** de la colonne, **sans guillemets** ;
- Pour plusieurs colonnes, fournir les noms sous forme de liste.

```
chomage[1,ouvriers]
```

```
## [1] 8738
```

```
chomage[1, list(ouvriers, ingenieurs)]
```

```
##    ouvriers ingenieurs
## 1:     8738       1420
```

Accès aux données : extraction de sous parties

- Un moyen simple, mais pas optimal : à l'aide des opérateurs logiques.

```
chomage[annee == 2010 & departement %in% c("Finistere", "Morbihan"),
        list(departement, ouvriers, annee)]
```

```
##    departement ouvriers annee
## 1:   Finistere    12258  2010
## 2:    Morbihan     9617  2010
```

Exercices

Après avoir chargé le `car`, créer l'objet `mroz_dt` qui doit contenir les données du `Mroz`. La structure de `mroz_dt` doit être : .

```
library(car)
head(Mroz, 2)
```

```
##   lfp k5 k618 age wc hc       lwg   inc
## 1 yes  1    0  32 no no 1.2101647 10.91
## 2 yes  0    2  30 no no 0.3285041 19.50
```

Exercices

1. Extraire la première ligne de `mroz_dt`, puis la première et la troisième ;
2. Extraire les valeurs de la colonne `lwg` uniquement ;
3. Extraire les valeurs des colonnes `lwg` et `inc` ;
4. Afficher un aperçu des s en mémoire ;
5. Extraire les valeurs de `mroz_dt` pour lesquelles `age` vaut moins de 34 :
6. Extraire les valeurs de `mroz_dt` pour lesquelles `age` vaut moins de 34, et `lfp` vaut `yes`.

Clés

- Clés ( ) **primaires** uniquement ;
- Une clé est composée d'**une ou plusieurs variables** du `data.table`, qui peuvent être de mode `factor`, `numeric`, `integer`, `character`, ... ;
- Le `data.table` est **ordonné** en fonction de la clé ;
- Les clés dupliquées sont autorisées ;
- La **définition** d'une clé se fait à l'aide de la fonction : `setkey()`

Clés

```
setkey(chomage, departement)
tables()
```

```
##       NAME        NROW MB COLS
## [1,] chomage       12 1  region,departement,annee,ouvriers,ingenieurs
## [2,] pop            4 1  region,annee,population
## [3,] population     4 1  region,annee,population
##       KEY
## [1,] departement
## [2,]
## [3,]
## Total: 3MB
```

Clés

- Le `data.table` est bien ordonné selon sa clé.

```
chomage
```

```
##       region     departement annee ouvriers ingenieurs
##  1:    Corse    Corse-du-Sud  2011      975         259
##  2:    Corse    Corse-du-Sud  2010      936         253
##  3: Bretagne   Cotes-d'Armor  2011     8738        1420
##  4: Bretagne   Cotes-d'Armor  2010     8113        1334
##  5: Bretagne       Finistere  2011    12701        2530
##  6: Bretagne       Finistere  2010    12258        2401
##  7:    Corse     Haute-Corse  2011     1297         254
##  8:    Corse     Haute-Corse  2010     1220         241
##  9: Bretagne Ille-et-Vilaine  2011    11390        3986
## 10: Bretagne Ille-et-Vilaine  2010    10897        3776
## 11: Bretagne        Morbihan  2011    10228        2025
## 12: Bretagne        Morbihan  2010     9617        1979
```

Clés : accès

- On peut faire référence à une valeur de la clé pour réaliser une **extraction**.

```
chomage["Finistere",]
```

```
##    departement   region annee ouvriers ingenieurs
## 1:   Finistere Bretagne  2011    12701       2530
## 2:   Finistere Bretagne  2010    12258       2401
```

- On n'est pas obligé de mettre la virgule.

```
chomage["Finistere"]
```

```
##    departement   region annee ouvriers ingenieurs
## 1:   Finistere Bretagne  2011    12701       2530
## 2:   Finistere Bretagne  2010    12258       2401
```

Clés : première ou dernière observation en cas de duplication

- La clé `departement` est dupliquée pour la valeur `Finistere` ;
- Deux observations sont retournées ;
- Extraire la première uniquement : `mult = "first"` ;
- Extraire la dernière uniquement : `mult = "last"`

```
chomage["Finistere", mult = "first"]
```

```
##    departement   region annee ouvriers ingenieurs
## 1:   Finistere Bretagne  2011    12701       2530
```

```
chomage["Finistere", mult = "last"]
```

```
##    departement   region annee ouvriers ingenieurs
## 1:   Finistere Bretagne  2010    12258       2401
```

Clés : plusieurs variables

- Utiliser la fonction `J()`.

```
##       region departement annee ouvriers ingenieurs
## 1: Finistere          NA    NA       NA         NA
```

```
setkey(chomage, region, departement)
tables()
```

```
##      NAME        NROW MB COLS
## [1,] chomage       12 1  region,departement,annee,ouvriers,ingenieurs
## [2,] pop            4 1  region,annee,population
## [3,] population     4 1  region,annee,population
##      KEY
## [1,] region,departement
## [2,]
## [3,]
## Total: 3MB
```

Clés : plusieurs variables

```
chomage["Corse"]
```

```
##    region  departement annee ouvriers ingenieurs
## 1:  Corse Corse-du-Sud  2011      975         259
## 2:  Corse Corse-du-Sud  2010      936         253
## 3:  Corse  Haute-Corse  2011     1297         254
## 4:  Corse  Haute-Corse  2010     1220         241
```

```
chomage[J("Bretagne", "Finistere")]
```

```
##      region departement annee ouvriers ingenieurs
## 1: Bretagne   Finistere  2011    12701       2530
## 2: Bretagne   Finistere  2010    12258       2401
```

Clés : recherche par clé secondaire

- Pas encore développé ;
- Contournement de problème : nouvelle définition de clé.

```
# Ne fonctionne pas
chomage["Finistere"]
```

```
##       region departement annee ouvriers ingenieurs
## 1: Finistere          NA    NA       NA         NA
```

```
setkey(chomage, departement)
chomage["Finistere"]
```

```
##    departement   region annee ouvriers ingenieurs
## 1:   Finistere Bretagne  2011    12701       2530
## 2:   Finistere Bretagne  2010    12258       2401
```

Clés : recherche par clé secondaire

```
# Retour à la clé précédente
setkey(chomage, region, departement)
```

Exercices

1. En reprenant le `mroz_dt` de l'exercice précédent, définir la variable `lfp` comme clé ;
2. En s'appuyant sur cette clé, extraire les observations pour lesquelles `lfp` vaut `yes` ;
3. Définir une nouvelle clé, qui permet d'identifier les couples uniques de `lfp` et `age` ;
4. Extraire les observations des femmes de 34 ans (`age` = 34) et participant au marché du travail (`lfp` = `"yes"`).

Transformation sur les colonnes : ajout de variable

- Le deuxième paramètre de `"["()` peut une ou plusieurs expression, dont les paramètres sont les noms des variables du `data.table` (sans les guillemets) ;
- L'ajout d'une variable se fait avec "`:=`".

```
head(chomage[, a := letters[1:12]])
```

```
##      region   departement annee ouvriers ingenieurs a
## 1:    Corse  Corse-du-Sud  2011      975         259 a
## 2:    Corse  Corse-du-Sud  2010      936         253 b
## 3: Bretagne Cotes-d'Armor  2011     8738        1420 c
## 4: Bretagne Cotes-d'Armor  2010     8113        1334 d
## 5: Bretagne     Finistere  2011    12701        2530 e
## 6: Bretagne     Finistere  2010    12258        2401 f
```

```
# Equivalent de
# chomage$temp <- letters[1:12]
```

Transformation sur les colonnes : ajout de variable

```
chomage[, b := ouvriers + ingenieurs]
```

```
##       region     departement annee ouvriers ingenieurs a     b
##  1:    Corse    Corse-du-Sud  2011      975        259 a  1234
##  2:    Corse    Corse-du-Sud  2010      936        253 b  1189
##  3: Bretagne   Cotes-d'Armor  2011     8738       1420 c 10158
##  4: Bretagne   Cotes-d'Armor  2010     8113       1334 d  9447
##  5: Bretagne       Finistere  2011    12701       2530 e 15231
##  6: Bretagne       Finistere  2010    12258       2401 f 14659
##  7:    Corse     Haute-Corse  2011     1297        254 g  1551
##  8:    Corse     Haute-Corse  2010     1220        241 h  1461
##  9: Bretagne Ille-et-Vilaine  2011    11390       3986 i 15376
## 10: Bretagne Ille-et-Vilaine  2010    10897       3776 j 14673
## 11: Bretagne        Morbihan  2011    10228       2025 k 12253
## 12: Bretagne        Morbihan  2010     9617       1979 l 11596
```

Transformation sur les colonnes : ajout de plusieurs variables

- Syntaxe un peu lourde.

```
chomage[, ":=" (c = ouvriers/100, d = paste0(a,b))]
```

```
##       region     departement annee ouvriers ingenieurs a     b      c      d
##  1:    Corse    Corse-du-Sud  2011      975         259 a  1234   9.75  a1234
##  2:    Corse    Corse-du-Sud  2010      936         253 b  1189   9.36  b1189
##  3: Bretagne   Cotes-d'Armor  2011     8738        1420 c 10158  87.38 c10158
##  4: Bretagne   Cotes-d'Armor  2010     8113        1334 d  9447  81.13  d9447
##  5: Bretagne       Finistere  2011    12701        2530 e 15231 127.01 e15231
##  6: Bretagne       Finistere  2010    12258        2401 f 14659 122.58 f14659
##  7:    Corse     Haute-Corse  2011     1297         254 g  1551  12.97  g1551
##  8:    Corse     Haute-Corse  2010     1220         241 h  1461  12.20  h1461
##  9: Bretagne Ille-et-Vilaine  2011    11390        3986 i 15376 113.90 i15376
## 10: Bretagne Ille-et-Vilaine  2010    10897        3776 j 14673 108.97 j14673
## 11: Bretagne        Morbihan  2011    10228        2025 k 12253 102.28 k12253
## 12: Bretagne        Morbihan  2010     9617        1979 l 11596  96.17 l11596
```

Transformation sur les colonnes : ajout de plusieurs variables

- Réutilisation d'une variable fraîchement créée.

```
head(chomage[, c("e", "f") := list(tmp <- ouvriers + ingenieurs,
                                   round(sqrt(tmp)))], 4)
```

```
##      region   departement annee ouvriers ingenieurs a     b     c      d     e
## 1:    Corse  Corse-du-Sud  2011      975        259 a  1234  9.75  a1234  1234
## 2:    Corse  Corse-du-Sud  2010      936        253 b  1189  9.36  b1189  1189
## 3: Bretagne Cotes-d'Armor  2011     8738       1420 c 10158 87.38 c10158 10158
## 4: Bretagne Cotes-d'Armor  2010     8113       1334 d  9447 81.13  d9447  9447
##      f
## 1:  35
## 2:  34
## 3: 101
## 4:  97
```

Transformation sur les colonnes : suppression

- Avec `NULL` ;
- Nom des variables entre guillemets, dans un vecteur.

```
# Suppression de plusieurs colonnes
head(chomage[, c("a", "b", "c", "d", "e") := NULL], 4)
```

```
##      region   departement annee ouvriers ingenieurs   f
## 1:    Corse  Corse-du-Sud  2011      975        259  35
## 2:    Corse  Corse-du-Sud  2010      936        253  34
## 3: Bretagne Cotes-d'Armor  2011     8738       1420 101
## 4: Bretagne Cotes-d'Armor  2010     8113       1334  97
```

Transformation sur les colonnes : suppression

```
# Suppression d'une seule colonne
head(chomage[, f := NULL], 4)
```

```
##      region   departement annee ouvriers ingenieurs
## 1:    Corse  Corse-du-Sud  2011      975         259
## 2:    Corse  Corse-du-Sud  2010      936         253
## 3: Bretagne Cotes-d'Armor  2011     8738        1420
## 4: Bretagne Cotes-d'Armor  2010     8113        1334
```

Agrégation

- Pour appliquer une fonction à une colonne : travailler avec le second paramètre de `"["()`.

```
chomage[, mean(ouvriers)]
```

```
## [1] 7364.167
```

Agrégation

- Pour effectuer un groupement préalable à l'application de la fonction : `by`.

```
chomage[, mean(ouvriers), by = region]
```

```
##      region       V1
## 1:    Corse  1107.00
## 2: Bretagne 10492.75
```

Agrégation

```
chomage[, mean(ouvriers), by = "region,departement"]
```

```
##      region     departement      V1
## 1:    Corse    Corse-du-Sud   955.5
## 2: Bretagne   Cotes-d'Armor  8425.5
## 3: Bretagne       Finistere 12479.5
## 4:    Corse     Haute-Corse  1258.5
## 5: Bretagne Ille-et-Vilaine 11143.5
## 6: Bretagne        Morbihan  9922.5
```

```
# Fournit le même résultat
# chomage[, mean(ouvriers), by = list(region, departement)]
```

Agrégation : calculs sur plusieurs colonnes

- Fournir la liste des opérations à effectuer en second paramètre de `"["()`.

```
chomage[, list(mean(ouvriers), sd(ouvriers)), by = region]
```

```
##      region       V1        V2
## 1:    Corse  1107.00  178.4507
## 2: Bretagne 10492.75 1627.4581
```

```
# Ne permet pas d'attribuer le nom "moyenne" au calcul
chomage[, moyenne = mean(ouvriers), by = region]
```

```
## Error: argument inutilisé (moyenne = mean(ouvriers))
```

Agrégation : calculs sur plusieurs colonnes

```
# Il faut être dans une liste
chomage[, list(moyenne = mean(ouvriers)), by = region]
```

```
##      region  moyenne
## 1:    Corse  1107.00
## 2: Bretagne 10492.75
```

Tri

- Soit changer de clé ;
- Soit utiliser la fonction `order()`

```
head(chomage[order(annee)])
```

```
##      region     departement annee ouvriers ingenieurs
## 1:    Corse    Corse-du-Sud  2010      936         253
## 2: Bretagne   Cotes-d'Armor  2010     8113        1334
## 3: Bretagne       Finistere  2010    12258        2401
## 4:    Corse     Haute-Corse  2010     1220         241
## 5: Bretagne Ille-et-Vilaine  2010    10897        3776
## 6: Bretagne        Morbihan  2010     9617        1979
```

Exercices

1. Créer la variable `kids` dans `mroz_dt`, qui est la somme de `k5` et `k618`, et dans la même instruction, créer la variable `inc_2` qui doit être le carré de `inc` ;
2. Calculer la moyenne du revenu familial (`inc`), pour chaque âge (`age`) et par participation ou non au marché du travail (`lfp`) ;
3. Supprimer les variables `kids` et `inc_2`.

Copie de data.table

- Utiliser la fonction `copy()` ;
- La flèche d'assignation "`<-`" ne fait que pointer dans le cas d'un `data.frame` !

```
dt <- data.table(x = letters[1:3], y = 4:6)
dt
```

```
##    x y
## 1: a 4
## 2: b 5
## 3: c 6
```

```
# Référence
dt_ref <- dt

# Copie
dt_copie <- copy(dt)
```

Copie de data.table

```
# Ajoutons une variable
dt[, a := rep(1,3)]
```

```
##    x y a
## 1: a 4 1
## 2: b 5 1
## 3: c 6 1
```

```
dt_ref[, z := paste0(x,y)]
```

```
##    x y a  z
## 1: a 4 1 a4
## 2: b 5 1 b5
## 3: c 6 1 c6
```

Copie de data.table

```
dt_copie[, t := paste0(y,x)]
```

```
##    x y  t
## 1: a 4 4a
## 2: b 5 5b
## 3: c 6 6c
```

Copie de data.table

- Les variables "`a`" et "`z`" ont été ajoutées à `dt` et `dt_ref`;

```
dt
```

```
##    x y a  z
## 1: a 4 1 a4
## 2: b 5 1 b5
## 3: c 6 1 c6
```

```
dt_ref
```

```
##    x y a  z
## 1: a 4 1 a4
## 2: b 5 1 b5
## 3: c 6 1 c6
```

Copie de data.table

- `dt_copie` n'a pas été impactée

```
dt_copie
```

```
##    x y  t
## 1: a 4 4a
## 2: b 5 5b
## 3: c 6 6c
```

Copie de data.table

- Par contre, la suppression de `dt_ref` ne supprime pas `dt`.

```
rm(dt_ref)
dt
```

```
##    x y a
## 1: a 4 1
## 2: b 5 1
## 3: c 6 1
```

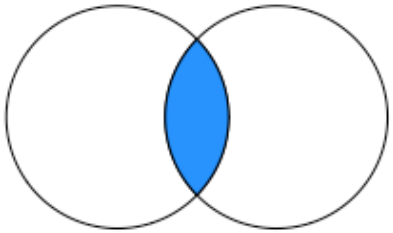
Jointure

- Jointure possible avec `merge()` ;
- Plus puissant en utilisant `"["()` !

```
setkey(chomage, region, annee)
setkey(population, region, annee)
chomage[population]
```

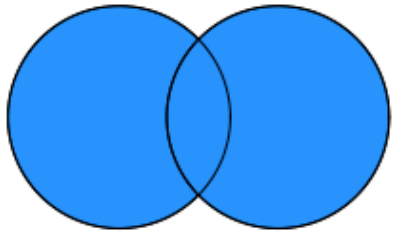
```
##       region annee     departement ouvriers ingenieurs population
##  1:   Alsace  2010              NA       NA         NA    1845687
##  2:   Alsace  2011              NA       NA         NA    1852325
##  3: Bretagne  2010   Cotes-d'Armor     8113       1334    3199066
##  4: Bretagne  2010       Finistere    12258       2401    3199066
##  5: Bretagne  2010 Ille-et-Vilaine    10897       3776    3199066
##  6: Bretagne  2010        Morbihan     9617       1979    3199066
##  7: Bretagne  2011   Cotes-d'Armor     8738       1420    3217767
##  8: Bretagne  2011       Finistere    12701       2530    3217767
##  9: Bretagne  2011 Ille-et-Vilaine    11390       3986    3217767
## 10: Bretagne  2011        Morbihan    10228       2025    3217767
```

Jointure : les différentes sortes



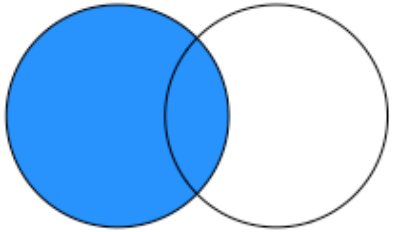
population[chomage, nonmatch = 0]

Full Outer Join



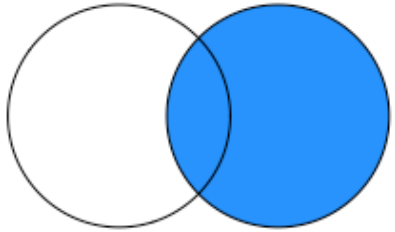
merge(chomage, population, all = TRUE)

Left Outer Join



population[chomage]

Inner Join



chomage[population]

Right Outer Join

Retirer les valeurs dupliquées

- La fonction `unique()` fonctionne aussi bien sur les que sur les `data.table`.

```
# Les observations 1 et 7, ainsi que 3 et 6 sont dupliquées
dt <- data.table(x = letters[c(1,2,3,1,4,3,1)],
                 y = letters[c(1,5,2,4,3,2,1)])

# setkey(dt, NULL)
unique(dt)
```

```
##    x y
## 1: a a
## 2: b e
## 3: c b
## 4: a d
## 5: d c
```

Importation

- Fonction encore en cours de développement : `fread()` ;
- Similaire à `read.table()` ;
- Lit mieux les gros volumes de données ;
- Séparateurs de classes automatiquement détectés (parfois !).

Voici un exemple fictif, avec un fichier `.csv` :

```
dt <- fread("monfichier.csv")
```

Manipulation des chaînes de caractères

La fonction `cat()`

- Concaténer et afficher à l'écran : `cat()` ;
- Les paramètres fournis sont convertis en vecteurs de chaînes de caractères ;
- Ces vecteurs sont ensuite concaténés en un seul, avec une espace entre chaque ;
- L'espace peut être remplacée par ce que l'utilisateur souhaite, en jouant avec `sep`.

```
cat("Hello", "World", "!")
```

```
## Hello World !
```

La fonction `cat()`

- En ajoutant un passage à la ligne entre chaque élément

```
cat("Hello", "World", "!", sep = "\n")
```

```
## Hello
## World
## !
```

- Avec une matrice :

```
cat(matrix(1:6))
```

```
## 1 2 3 4 5 6
```

La fonction `cat()` : remarque

- Le caractère `\` sert d'échappement en `R` ;
- Il permet d'afficher certains caractères :

CARACTÈRE	DESCRIPTION	CARACTÈRE	DESCRIPTION
`\n`	Nouvelle ligne	`\r`	Retour à la ligne
`\t`	Tabulation	`\b`	Retour arrière
`\\`	Barre olbique inversée	`\'`	Apostrophe
`\"`	Apostrophe double	`\‘`	Accent grave

La fonction `str_c()`

- Nécessité de charger le `stringr`
- La fonction `str_c()` convertit les éléments passés en caractères ;
- Elle les concatène ;
- Le résultat est une chaîne de caractères ;
- Le résultat peut être sotcké avec la flèche d'assignation "`<-`" ;
- Une chaîne de séparation peut être définie à l'aide du paramètre `sep`.

```
library(stringr)
x <- str_c("Hello", "World", "!", sep = " ")
x
```

```
## [1] "Hello World !"
```

La fonction `str_c()`

- Le paramètre `collapse` permet de joindre les éléments d'un vecteur dans une même chaîne.

```
str_c(c("Rennes", "Bretagne"))
```

```
## [1] "Rennes"   "Bretagne"
```

```
# Il n'y a qu'un paramètre, donc le séparateur est inutile !
str_c(c("Rennes", "Bretagne"), sep = ", ")
```

```
## [1] "Rennes"   "Bretagne"
```

```
# En ajoutant le paramètre collapse
str_c(c("Rennes", "Bretagne"), collapse = ", ")
```

```
## [1] "Rennes, Bretagne"
```

La fonction `str_c()`

- Les règles de recyclage s'appliquent.

```
str_c("Hello", c("Julien-Yacine", "Sonia", "Victor"))
```

```
## [1] "HelloJulien-Yacine" "HelloSonia"         "HelloVictor"
```

- Les valeurs `NA` sont converties en chaîne `"NA"`...

```
str_c("Hello", NA)
```

```
## [1] "HelloNA"
```

La fonction `str_c()` : remarque

- La fonction `get()` retourne le contenu d'une variable dont on lui indique le nom sous forme de chaîne de caractères.

```
variable_1 <- 5
# Affiche la chaîne "variable_1"
str_c("variable_", 1)
```

```
## [1] "variable_1"
```

```
# Affiche le contenu de la variable nommée "variable_1"
get(str_c("variable_", 1))
```

```
## [1] 5
```

Conversion en majuscules ou minuscules

- Majuscule : `toupper()`;
- Minuscule : `tolower()` ;

```
x <- "Bonjour !"
toupper(x)
```

```
## [1] "BONJOUR !"
```

```
tolower(x)
```

```
## [1] "bonjour !"
```

Conversion en majuscules ou minuscules

- La fonction `casefold()` fait les deux : on lui indique un logique pour son paramètre `upper`.

```
casefold(x)
```

```
## [1] "bonjour !"
```

```
casefold(x, upper = TRUE)
```

```
## [1] "BONJOUR !"
```

Exercices

1. Créer le vecteur de chaînes commençant par `joueur_` et se terminant par un entier `i` allant de `1 à 10` ;
2. Créer la chaîne composée des deux sous-chaînes `Criquette Rockwell` et `Brett Montgomery`, séparées par un retour à la ligne. Afficher le résultat dans la console avec la fonction `cat()` ;
3. Créer la chaîne `J'aime les pommes - "Ryuk"`, et stocker le résultat dans l'objet `light` ;
4. Passer la chaîne de la question 3 (`light`) en majuscules ;
5. Afficher le contenu de `light` en faisant appel à la fonction `get()`.

Compter le nombre de caractères d'une chaîne

- La fonction `str_length()` retourne le nombre de caractères d'une chaîne.

```
str_length("Bonjour")
```

```
## [1] 7
```

```
str_length(c("Bonjour", "Hello"))
```

```
## [1] 7 5
```

Extraction et remplacement de sous-chaines

- **Extraction** en précisant la position de début et de fin : `str_sub()` ;
- Attention, le premier caractère d'une chaîne est en position 1 en `R`.

```
x <- "Debt is one person's liability, but another person's asset."
str_sub(x, 1, 4)
```

```
## [1] "Debt"
```

- Le **remplacement** se fait avec la flèche d'assignation "`<-`" ;
- Le remplacement d'une chaîne par une autre plus longue ne pose pas de problème.

```
str_sub(x, 1, 4) <- "Remplacement"
x
```

```
## [1] "Remplacement is one person's liability, but another person's asset."
```

Extraction et remplacement de sous-chaines : remarques

- Le paramètre `x` de `str_sub()` peut être un vecteur ;
- La fonction est alors appliquée à chaque élément du vecteur.

```
str_sub(c("Rouge", "Vert", "Bleu"), 2, 3)
```

```
## [1] "ou" "er" "le"
```

- Les règles de recyclage s'appliquent.

```
x <- c("Rouge", "Vert", "Bleu")
str_sub(x, 2, 3) <- c("!!", "@@")
x
```

```
## [1] "R!!ge" "V@@ge" "B!!ge"
```

Extraction et remplacement de sous-chaines : remarques

- Si le paramètre `start` (`end`) est négatif, le décompte des positions se fait à partir du dernier caractère de la liste.

```
texte <- "le train de tes injures roule sur le rail de mon indifférence"
str_sub(string = texte, start = -12, end = -1)
```

```
## [1] "indifférence"
```

Recherche de chaînes de caractères

- Recherche d'un **motif** ( ) dans un vecteur de chaînes de caractères : `str_detect()` ;
- Retourne une chaîne de booléens de la longueur du vecteur de chaînes fourni en paramètre.

```
str_detect(string = c("Pomme", "Poire", "Ananas"), pattern = "o")
```

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

Recherche de chaînes de caractères

- Le remplacement par une autre chaîne de la **première** occurence du motif trouvé se fait avec `str_replace()` ;
- La chaîne de remplacement n'est pas tenue d'avoir la même longueur que le motif.

```
str_replace(string = c("Pomme", "Poire", "Ananas"), pattern = "a", replacement = "@@")
```

```
## [1] "Pomme"   "Poire"   "An@@nas"
```

- Pour remplacer **toutes** les occurences trouvées : `str_replace_all()`.

```
str_replace_all(string = c("Pomme", "Poire", "Ananas"), pattern = "a", replacement = "@@")
```

```
## [1] "Pomme"    "Poire"    "An@@n@@s"
```

Recherche de chaînes de caractères

- L'utilisation de `ignore.case()` sur le paramètre `pattern` permet dignorer la casse.

```
str_detect(string = c("Obi-Wan Kenobi", "Darth Vader"), pattern = "w")
```

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

```
str_detect(string = c("Obi-Wan Kenobi", "Darth Vader"), pattern = ignore.case("w"))
```

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

Recherche de chaînes de caractères

- On peut utiliser le symbole `|` à l'intérieur de la chaîne de caractère passée au paramètre `pattern` pour détecter la présence d'un motif ou d'un autre dans la chaîne de caractères.

```
str_detect(string = c("Obi-Wan Kenobi", "Darth Vader"), pattern = "w|D")
```

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

Recherche de chaînes de caractères

- La **découpe** d'une chaîne en fonction d'un motif se fait avec `str_split()`.

```
x = "Criquette ! Vous, ici ? Dans votre propre salle de bain ? Quelle surprise !"
str_split(string = x, pattern = " ")
```

```
## [[1]]
##  [1] "Criquette" "!"         "Vous,"     "ici"       "?"         "Dans"
##  [7] "votre"     "propre"    "salle"     "de"        "bain"      "?"
## [13] "Quelle"    "surprise"  "!"
```

Exercices

1. Quelle fonction utiliser pour connaître le nombre de caractères d'une chaîne ?
2. Extraire `c("LON-1.6794", "LAT48.1147")` les valeurs numériques de longitue et latitude ;
3. Dans la chaîne `LON-1.6794`, remplacer `LON` par `LONG` ;
4. Repérer les éléments du vecteur `c("Finistere:I. d'Ouessant", "Finistere:I. de Batz", "Gard")` qui contiennent la sous-chaîne `finistere`, en ne tenant pas compte de la casse ;
5. Remplacer toutes les occurences de tirets (`-`) par une espace dans les éléments du vecteur `c("02-23-23-35-35", "02-23-23-35-45")` ;
6. Découper les numéros de téléphone `c("02-23-23-35-35", "02/23/23/35/45")` en fonction du tiret ou de la barre oblique.

La fonction : `str_dup()`

- Duplication de texte : `str_dup()`.

```
texte <- c("bla", "ah", "eh")
str_dup(texte, 2)
```

```
## [1] "blabla" "ahah"   "eheh"
```

```
str_dup(texte, 1:3)
```

```
## [1] "bla"    "ahah"   "eheheh"
```

La fonction : `str_pad()`

- Complétion d'une chaîne de caractères pour obtenir une chaîne avec une longueur donnée ;
- La longueur désirée se fixe avec le paramètre `width` ;
- Choix du côté pour la complétion : avec le paramètre `side` qui prend les valeurs `left`, `right` ou `both` ;

```
coords <- c(lat = "48.11", long = "-1.6794")
str_pad(string = coords, width = 7, side = "left", pad = " ")
```

```
## [1] "  48.11" "-1.6794"
```

- Les chaînes plus longues que la valeur fournie à width sont inchangées.

```
str_pad(c("Gauthier", "Pascaline"), 3)
```

```
## [1] "Gauthier"  "Pascaline"
```

La fonction : `str_trim()`

- Retrait des caractères blancs (espaces, sauts de ligne, retours à la ligne, quadratins, etc.) : `str_trim()` ;
- On peut choisir le côté sur lequel faire le retrait, avec le paramètre `side` (`left`, `right` ou `both`).

```
texte <- c("\n\nPardon, du sucre ?", "Oui, seize   \n ", "...\t\t...\t")
str_trim(texte, side = "both")
```

```
## [1] "Pardon, du sucre ?" "Oui, seize"         "...\t\t..."
```

La fonction : `word()`

- Extraction de mots dans une phrase : `word()` ;
- La séparation entre les mots est l'espace par défaut ;
- Le paramètre `sep` permet de changer le caractère unissant les mots ;
- Le paramètre `sart` attend un entier indiquant la position du premier mot à extraire (1 par défaut) ;
- Leparamètre `end` attend un entier indiquant la position du dernier mot à extraire.

```
phrase <- c("Mademoiselle Deray, il est interdit de manger de la choucroute ici.",
            "Oh si si, prenez un chewing-gum, Émile.")
```

La fonction : `word()`

```
word(phrase) # Extraction du premier mot
```

```
## [1] "Mademoiselle" "Oh"
```

```
word(phrase, 2, -1) # Extraction du second au dernier mot
```

```
## [1] "Deray, il est interdit de manger de la choucroute ici."
## [2] "si si, prenez un chewing-gum, Émile."
```

- Si `start` reçoit une valeur négative, le décompte s'effectue de la droite vers la gauche ;

```
word(phrase, -1) # Extraction du dernier mot
```

```
## [1] "ici."   "Émile."
```

La fonction : `word()`

```
# Du premier au dernier mot, du second au dernier, et du troisième au dernier
# pour le premier élément de phrase
word(phrase[1], 1:3, -1)
```

```
## [1] "Mademoiselle Deray, il est interdit de manger de la choucroute ici."
## [2] "Deray, il est interdit de manger de la choucroute ici."
## [3] "il est interdit de manger de la choucroute ici."
```

```
# Premier mot, Premier et second mot, Premier et troisième mot
# pour le second élément de phrase
word(phrase[2], 1, 1:3)
```

```
## [1] "Oh"        "Oh si"     "Oh si si,"
```

Manipulation des dates

Les classes de dates

- Trois classes de dates :
  1. `Date` : pour les dates sans informations sur les heures (`1977-05-27`),
  2. `POSIXct` et `POSIXlt` : pour les dates avec des heures (`1977-05-27 20:00:00`).

Les classes de dates : `date`

- Stockage en `R` : **nombre de jours** depuis `1970-01-01`, en valeurs négatives pour les dates antérieures ;
- On peut le voir avec la fonction `unclass()` ;
- Le format standard est : `%Y-%m-%d` ou `%Y/%m/%d` (`%Y` : année sur 4 chiffres, `%m` : mois sur deux chiffres, `%d` : jour sur deux chiffres).

```
(d <- as.Date("2015-10-21"))
```

```
## [1] "2015-10-21"
```

```
unclass(d)
```

```
## [1] 16729
```

Les classes de dates : `date`

- Le paramètre `format` permet de préciser des formats non standards ;
- Le paramètre `origin` permet de préciser une date d'origine différente de `1970-01-01`.

```
(d <- as.Date("2015 21 10", format = ("%Y %d %m")))
```

```
## [1] "2015-10-21"
```

```
(d <- as.Date("21 Octobre, 2015", format = ("%d %B, %Y")))
```

```
## [1] "2015-10-21"
```

Les classes de dates : `formats`

- Les codes des formats sont accessibles sur `?strptime()` (cf poly) ;

CODE	DESCRIPTION	EXEMPLE
%a	Abréviation du jour de la semaine (dépend du lieu)	Mer
%A	Jour de la semaine complet (dépend du lieu)	Mercredi
%b	Abréviation du mois (dépend du lieu)	oct
%B	Nom du mois complet (dépend du lieu)	octobre
%d	Jour du mois (01-31)	21
%D	Date au format %m/%d/%y	10/21/15
%H	Heure (00-24)	00
%m	Mois (01-12)	10
%M	Minute (00-59)	00
%S	Seconde (00-61)	00
%Y	Année (en input, uniquement de 0 à 9999)	2015
%z	offset en heures et minutes par rapport au temps UTC	+0000
%Z	Abréviation du fuseau horraire (en output seulement)	UTC

Les classes de dates : `POSIXct` et `POSIXlt`

- `POSIXct` : dates stockées en **secondes** depuis `1970-01-01 01:00:00` ;

```
d <- as.POSIXct("2015-10-21 13:55:44")
unclass(d)
```

```
## [1] 1445428544
## attr(,"tzone")
## [1] ""
```

```
unclass(as.POSIXct("1970-01-01 01:00:00"))
```

```
## [1] 0
## attr(,"tzone")
## [1] ""
```

Les classes de dates : `POSIXct` et `POSIXlt`

- La fonction `Sys.time()` retourne un objet `POSIXct` indiquant la date et l'heure actuelle.

```
Sys.time()
```

```
## [1] "2014-11-20 16:15:13 CET"
```

Les classes de dates : `POSIXct` et `POSIXlt`

- `POSIXlt` : dates sous formes de **listes** dont les éléments sont (cf `?DateTimeClasses`) :

CODE	DESCRIPTION
sec	secondes (0-61)
min	minutes (0-59)
hour	heures (0-23)
mday	jour du mois (1-31)
mon	mois après le premier de l'année (0-11)
year	années depuis 1900
wday	jour de la semaine (0-6), début de semaine le dimanche
yday	jour de l'année (0-365)
isdst	indicateur d'heure d'été (positif si appliquable, zéro sinon; négatif si in- connu)
zone	Abréviation du fuseau horraire local ("" si inconnu, ou UTC)
gmtoff	décalage en minutes par rapport au temps GMT (NA ou 0 si inconnu)

Les classes de dates : `POSIXct` et `POSIXlt`

- Voici un exemple de liste retournée par la fonction `POSIXlt` (la fonction `unlist()` est appliquée pour permettre l'affichage sur la diapositive) ;

```
d <- as.POSIXlt("2015-10-21 13:55:44")
unlist(unclass(d))
```

```
##    sec    min   hour   mday    mon   year   wday   yday  isdst   zone gmtoff
##   "44"   "55"   "13"   "21"    "9"  "115"    "3"  "293"    "1" "CEST"     NA
```

- Récupération aisée des éléments.

```
d$hour
```

```
## [1] 13
```

Les classes de dates : soucis de langue

- Il arrive de récupérer des dates dans lesquelles le nom du jour ou du mois est en anglais ;
- Avec les paramètres locaux français, R ne parvient pas à interpréter ce genre de dates.

```
d_char_fr <- "Mer 04 Fév 2015"
d_char <- "Wed 04 Feb 2015"
as.Date(d_char_fr, format = c("%a %d %b %Y"))
```

```
## [1] "2015-02-04"
```

```
as.Date(d_char, format = c("%a %d %b %Y"))
```

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

Les classes de dates : soucis de langue

- Avec les paramètres locaux français, `R` considère les noms de la semaine et des mois en français :

```
weekdays(Sys.Date()+0:6)
```

```
## [1] "Jeudi"    "Vendredi" "Samedi"   "Dimanche" "Lundi"    "Mardi"    "Mercredi"
```

```
month.name
```

```
##  [1] "January"   "February"  "March"     "April"     "May"       "June"
##  [7] "July"      "August"    "September" "October"   "November"  "December"
```

```
month.abb
```

```
##  [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"
```

Les classes de dates : soucis de langue

- Il faut alors changer les paramètres locaux, avec la fonction `Sys.setlocale()`.

```
# Les paramètres locaux pour les unités de temps
(old <- Sys.getlocale("LC_TIME"))
```

```
## [1] "fr_FR.UTF-8"
```

```
# Indiquer que les dates doivent être au format américain
Sys.setlocale('LC_TIME','en_us')
```

```
## [1] "en_us"
```

Les classes de dates : soucis de langue

```
as.Date(d_char, format = c("%a %d %b %Y"))
```

```
## [1] "2015-02-04"
```

```
as.Date(d_char_fr, format = c("%a %d %b %Y"))
```

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

Les classes de dates : soucis de langue

```
# Retourner aux paramètres précédents
Sys.setlocale('LC_TIME', old)
```

```
## [1] "fr_FR.UTF-8"
```

```
as.Date(d_char, format = c("%a %d %b %Y"))
```

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

```
as.Date(d_char_fr, format = c("%a %d %b %Y"))
```

```
## [1] "2015-02-04"
```

Exercices

1. Avec la fonction `as.Date()`, créer un objet contenant la date du jour de l'an 2015 ;
2. Convertir la chaîne de caractères `x <- "15/09/2014"` en objet de type `Date` ;
3. Convertir l'objet de la question 2 en `POSIXct`.

Opérations

- Pour les exemples, créons des objets de chaque type de dates.

```
d_date_1 <- as.Date("2014-09-01")
d_date_2 <- as.Date("2015-10-21")

d_posix_ct_1 <- as.POSIXct("2014-09-01 18:32:28")
d_posix_ct_2 <- as.POSIXct("2015-10-21 13:55:44")

d_posix_lt_1 <- as.POSIXlt("2014-09-01 18:32:28")
d_posix_lt_2 <- as.POSIXlt("2015-10-21 13:55:44")
```

Opérations : différence entre deux dates

- La différence entre deux dates s'obtient à l'aide du symbole "-" ;

```
d_date_2 - d_date_1
```

```
## Time difference of 415 days
```

```
d_posix_ct_2 - d_posix_ct_1
```

```
## Time difference of 414.8078 days
```

```
d_posix_lt_2 - d_posix_lt_1
```

```
## Time difference of 414.8078 days
```

Opérations : différence avec `difftime()`

- Pour les `POSIXct` et `POSIXlt`, on peut également utiliser `difftime()` ;
- Deux dates à fournir en paramètres ;
- Éventuellement un l'unité temporelle souhaitée pour le résultat avec le paramètre `units` ;

CODE	DESCRIPTION
auto	le choix de l'unité pour que celle-ci soit la plus large possible (à l'exception de weeks)
secs	secondes
mins	minutes
hours	heures
days	jours (périodes de 24 heures)
weeks	semaines

- On peut utiliser les éléments du groupe `Math` sur les résultats (`round()`, `signif()`, `floor()`, `ceiling()`, `trunc()`, `abs()` et `sign()`, etc.)

Opérations : différence avec `difftime()`

```
# Avec des POSIXct
difftime(d_posix_ct_2, d_posix_ct_1, units = "hours")
```

```
## Time difference of 9955.388 hours
```

```
round(difftime(d_posix_ct_2, d_posix_ct_1, units = "hours"))
```

```
## Time difference of 9955 hours
```

```
# Avec des POSIXlt
difftime(d_posix_lt_2, d_posix_lt_1, units = "week")
```

```
## Time difference of 59.25826 weeks
```

Opérations : ajout d'une durée à une date

- Il faut se rappeler l'unité de stockage selon le format :
  - `date` : jours,
  - `POSIXct` et `POSIXlt` : secondes.

```
d_date_2
```

```
## [1] "2015-10-21"
```

```
d_date_2 + 10
```

```
## [1] "2015-10-31"
```

Opérations : ajout d'une durée à une date

```
d_posix_ct_2
```

```
## [1] "2015-10-21 13:55:44 CEST"
```

```
d_posix_ct_2 + 10
```

```
## [1] "2015-10-21 13:55:54 CEST"
```

Opérations : comparaison de deux dates

- Il est possible d'utiliser les opérateurs de comparaison entre deux dates.

```
d_date_1 > d_date_2
```

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

```
d_posix_ct_2 > d_posix_ct_1
```

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

```
d_posix_lt_2 == d_posix_lt_1
```

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

Séquence de dates

- La création de séquences de dates se fait avec la fonction `seq()` ;

```
# Avec des objets de class date
seq(as.Date("2014-09-01"), length = 4, by = "day")
```

```
## [1] "2014-09-01" "2014-09-02" "2014-09-03" "2014-09-04"
```

```
# Tous les deux jours
seq(as.Date("2014-09-01"), length = 4, by = "2 day")
```

```
## [1] "2014-09-01" "2014-09-03" "2014-09-05" "2014-09-07"
```

```
# En spécifiant le début et la date maximum
seq(as.Date("2014-09-01"), as.Date("2014-09-08") , by = "2 day")
```

```
## [1] "2014-09-01" "2014-09-03" "2014-09-05" "2014-09-07"
```

Séquence de dates

```
# Avec des objets de classe POSIXct et POSIXlt
seq(d_posix_ct_1, by = "9 month", length = 2)
```

```
## [1] "2014-09-01 18:32:28 CEST" "2015-06-01 18:32:28 CEST"
```

```
seq(d_posix_lt_1, by = "9 month", length = 2)
```

```
## [1] "2014-09-01 18:32:28 CEST" "2015-06-01 18:32:28 CEST"
```

Exercices

1. Calculer le nombre de jours séparant "2015-01-01" et "2015-01-14" ;
2. Créer une séquence de dates allant de "1975-01-01" à "2010-01-01", par pas de 6 mois.

Le `lubridate`

- Le `lubridate` s'appuie sur les fonctions présentées avant ;
- Il permet de manipuler plus facilement des dates.

```
library(lubridate)
```

Le `lubridate` : conversion

- Pour passer d'une chaîne de caractères à un `POSIXct` :
  - `ymd()`,
  - `mdy()`,
  - `dmy()` ;
- Les lettres de ces fonctions font référence à l'année (`y`), le mois (`m`) et le jour (`d`) ;
- Leur ordre dans le nom de la fonction correspond à celui dans la chaîne de caractères à convertir.

Le `lubridate` : conversion

```
ymd("2015-10-21")
```

```
## [1] "2015-10-21 UTC"
```

```
mdy("10.21-15")
```

```
## [1] "2015-10-21 UTC"
```

```
dmy("21 oct 2015")
```

```
## [1] "2015-10-21 UTC"
```

Le `lubridate` : conversion

- Le fuseau horraire peut être précisé par le paramètre `tz` (s'il est reconnu par le système d'exploitation) ;
- La liste des fuseaux horaires s'obtient avec la fonction `OlsonNames()`.

```
ymd("2015-10-21", tz = "Pacific/Auckland")
```

```
## [1] "2015-10-21 NZDT"
```

Le `lubridate` : conversion et langue différente

- Le paramètre `locale` permet d'indiquer le lieu ;
- De fait, plus de problèmes de langue !

```
dmy("Wed 04 Feb 2015")
```

```
## Warning: All formats failed to parse. No formats found.
```

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

```
dmy("Wed 04 Feb 2015", locale = "en_us")
```

```
## [1] "2015-02-04 UTC"
```

Le `lubridate` : conversion de dates avec heures

- Avec des heures dans les dates, rajouter au nom de la fonction le suffixe :
  - `_h` : présence de l'heure,
  - `_hm` : présence de l'heure et des minutes,
  - `_hms` : présence de l'heure, des minutes et des secondes.

```
ymd_hms("2015-10-21 10:00:50")
```

```
## [1] "2015-10-21 10:00:50 UTC"
```

```
ymd_hms("2015 Feb 21 10:00:50", locale = "en_us")
```

```
## [1] "2015-02-21 10:00:50 UTC"
```

Le `lubridate` : extraction d'éléments de dates

FONCTION	EXTRACTION
`second()`	secondes (0-59)
`minute()`	minutes (0-59)
`hour()`	heures (0-23)
`day()`, `mday()`	jour du mois (1-31)
`wday()`	jour de la semaine (1-7), le dimanche étant le 1
`yday()`	jour de l'année (1-366)
`week()`	numéro de la semaine dans l'année. Les semaines sont définies comme les pé- riodes complètes de 7 jours s'étant déroulées depuis le premier janvier, plus 1
`isoweek()`	Semaine de l'année (00-53). Si la semaine (qui commence un lundi) qui contient le 1er janvier a 4 jours ou plus dans la nouvelle année, alors elle est considérée comme la semaine 1. Sinon, elle est considérée comme la dernière de l'année précédente, et la suivante est considérée comme semaine 1
`month()`	mois (1-12)
`year()`	année, uniquement celles après l'an 1
`tz()`	fuseau horraire

Le `lubridate` : extraction d'éléments de dates

- L'ajout du paramètre `label` évalué à `TRUE` aux fonctions `wday()` et `month()` permet d'afficher le nom du jour de la semaine et du mois respectivement (en anglais).

```
(d <- ymd_hms("2015-10-30 23:59:59"))
```

```
## [1] "2015-10-30 23:59:59 UTC"
```

```
wday(d, label = TRUE)
```

```
## [1] Fri
## Levels: Sun < Mon < Tues < Wed < Thurs < Fri < Sat
```

```
month(d, label = TRUE)
```

```
## [1] Oct
## 12 Levels: Jan < Feb < Mar < Apr < May < Jun < Jul < Aug < Sep < ... < Dec
```

Le `lubridate` : extraction d'éléments de dates

- Si en plus on fournit `abbr = TRUE`, cela retourne l'abréviation.

```
wday(d, label = TRUE, abbr = TRUE)
```

```
## [1] Fri
## Levels: Sun < Mon < Tues < Wed < Thurs < Fri < Sat
```

```
month(d, label = TRUE, abbr = TRUE)
```

```
## [1] Oct
## 12 Levels: Jan < Feb < Mar < Apr < May < Jun < Jul < Aug < Sep < ... < Dec
```

Le `lubridate` : conversion de fuseaux horaires

- Obtenir la date et l'heure dans un autre fuseau horaire : `with_tz()`

```
# Une heure à Paris
(d <- ymd_hms("2014-10-30 20:00:00", tz = "Europe/Paris"))
```

```
## [1] "2014-10-30 20:00:00 CET"
```

```
# L'heure équivalente à New York City
with_tz(d, "America/New_York")
```

```
## [1] "2014-10-30 15:00:00 EDT"
```

Le `lubridate` : conversion de fuseaux horaires

- Pour remplacer le fuseau horaire d'une date (pas de conversion) : `force_tz()`.

```
(d <- ymd_hms("2014-10-30 20:00:00", tz = "Europe/Paris"))
```

```
## [1] "2014-10-30 20:00:00 CET"
```

```
force_tz(d, "America/New_York")
```

```
## [1] "2014-10-30 20:00:00 EDT"
```

Exercices

1. Créer un objet de classe `POSIXct` contenant la date d'aujourd'hui à l'aide de la fonction `ymd()` ;
2. Même démarche, mais en rajoutant les informations sur l'heure ;
3. Extraire la valeurs des minutes de l'objet créé en question 2 ;
4. À partir de cette date sous forme de chaîne de caractères, `"Fri Sep 21 23:40:54 +0000 2012"`, la stocker sous forme de `POSIXct` ;
5. À partir de la date obtenue en question 4, obtenir l'heure équivalente à New York City ;

Le `lubridate` : intervalle de temps

- Intervalle de temps entre deux dates : `new_interval()` (ou `%--%`).

```
debut <- ymd_hms("2014-10-01 07:00:00", tz = "Australia/Perth")
fin <- ymd_hms("2014-10-01 23:00:00", tz = "Australia/Perth")

(intervalle_1 <- interval(debut, fin))
```

```
## [1] 2014-10-01 07:00:00 AWST--2014-10-01 23:00:00 AWST
```

```
debut %--% fin
```

```
## [1] 2014-10-01 07:00:00 AWST--2014-10-01 23:00:00 AWST
```

Le `lubridate` : intervalle de temps

- La durée de l'intervalle de temps s'obtient avec la fonction : `int_length()` ;
- Le résultat est exprimé en secondes.

```
int_length(intervalle_1)
```

```
## [1] 57600
```

Le `lubridate` : intervalle de temps

- Chercher si deux intervalles se chevauchent : `int_overlaps()`.

```
debut_2 <- ymd_hms("2014-10-01 08:00:00", tz = "Australia/Perth")
fin_2 <- ymd_hms("2014-10-01 10:00:00", tz = "Australia/Perth")

debut_3 <- ymd_hms("2014-10-02 09:00:00", tz = "Australia/Perth")
fin_3 <- ymd_hms("2014-10-02 10:00:00", tz = "Australia/Perth")

(intervalle_2 <- new_interval(debut_2, fin_2))
```

```
## [1] 2014-10-01 08:00:00 AWST--2014-10-01 10:00:00 AWST
```

```
(intervalle_3 <- new_interval(debut_3, fin_3))
```

```
## [1] 2014-10-02 09:00:00 AWST--2014-10-02 10:00:00 AWST
```

Le `lubridate` : intervalle de temps

```
int_overlaps(intervalle_1, intervalle_2)
```

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

```
int_overlaps(intervalle_1, intervalle_3)
```

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

Le `lubridate` : intervalle de temps et opérations ensemblistes

- Les fonctions ensemblistes fonctionnent sur les intervalles de temps :
  - `union()` : union,
  - `intersect()` : intersection,
  - `setdiff()` : différence,
  - `setequal()` : égalité,
  - `%within%` : appartenance à l'intervalle.

```
union(intervalle_1, intervalle_2)
```

```
## [1] 2014-10-01 07:00:00 AWST--2014-10-01 23:00:00 AWST
```

```
intersect(intervalle_1, intervalle_3)
```

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

Le `lubridate` : intervalle de temps et opérations ensemblistes

```
setdiff(intervalle_1, intervalle_2)
```

```
## [1] 2014-10-01 10:00:00 AWST--2014-10-01 08:00:00 AWST
```

```
setequal(intervalle_1, intervalle_1)
```

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

```
debut_2 %within% intervalle_1
```

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

Le `lubridate` : intervalle de temps et bornes

- L'extraction des bornes d'un intervalle de temps : `int_start()` et `int_end()`.

```
(intervalle <- setdiff(intervalle_1, intervalle_2))
```

```
## [1] 2014-10-01 10:00:00 AWST--2014-10-01 08:00:00 AWST
```

```
int_start(intervalle)
```

```
## [1] "2014-10-01 10:00:00 AWST"
```

```
int_end(intervalle)
```

```
## [1] "2014-10-01 08:00:00 AWST"
```

Le `lubridate` : intervalle de temps et bornes

- Échanger les bornes de l'intervalle `int_flip()` ;
- Le résultat a la même longueur, mais une direction opposée.

```
int_flip(intervalle_1)
```

```
## [1] 2014-10-01 23:00:00 AWST--2014-10-01 07:00:00 AWST
```

Le `lubridate` : intervalle de temps et décalage

- Pour décaler un intervalle de temps : `int_shift()` ;
- Préciser la durée du décalage : `new_duration()` :
  - `seconds`,
  - `minutes`,
  - `hours`,
  - `days`,
  - `weeks`.

```
int_shift(intervalle_1, new_duration(days = 5))
```

```
## [1] 2014-10-06 07:00:00 AWST--2014-10-06 23:00:00 AWST
```

Le `lubridate` : intervalle de temps et bornes communes

- Tester si deux intervalles ont une borne commune (peu importe la direction) : `int_aligns()`.

```
int_1 <- new_interval(ymd("2014-10-01"), ymd("2014-10-05"))
int_2 <- new_interval(ymd("2014-10-04"), ymd("2014-10-01"))
int_3 <- new_interval(ymd("2015-10-01"), ymd("2015-10-05"))
int_aligns(int_1, int_2)
```

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

```
int_aligns(int_1, int_3)
```

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

Le `lubridate` : laps de temps

- Deux classes de laps de temps :
  - `duration` : durées,
  - `period` : époques ;
- Nom des fonctions :
  - préfixe `d` suivi de l'unité de la durée, au pluriel pour les `duration`,
  - unité de la durée au pluriel pour les `period`.

```
minutes(10)
```

```
## [1] "10M 0S"
```

```
dminutes(10)
```

```
## [1] "600s (~10 minutes)"
```

Le `lubridate` : laps de temps

- Les `duration` ne tiennent pas compte des fluctuations de la ligne du temps (e.g. les années bissextiles) ;
- Les `period` si.
- Exemple avec l'année 2000 qui est bissextile (la fonction `leap_year(2000)` retourne `TRUE`).

```
# Ajout d'une durée d'un an, soit 365 jours
ymd("2000-01-01") + dyears(1)
```

```
## [1] "2000-12-31"
```

```
# Ajout d'une période d'un an, soit 366 jours dans ce cas
ymd("2000-01-01") + years(1)
```

```
## [1] "2001-01-01"
```

Le `lubridate` : durée d'un intervalle

- Pour obtenir la durée d'un intervalle dans l'unité désirée : division par une durée.

```
intervalle <- new_interval(ymd_hms("2014-10-01 10:00:00"),
                           ymd_hms("2014-10-06 08:00:00"))
# Nombre de jours
intervalle / ddays(1)
```

```
## [1] 4.916667
```

```
# Nombre de durées de 2 jours
intervalle / ddays(2)
```

```
## [1] 2.458333
```

Le `lubridate` : durée d'un intervalle

```
# Nombre d'heures
intervalle / dhours(1)
```

```
## [1] 118
```

Le `lubridate` : durée d'un intervalle

- On peut effectuer une division entière ou extraire le reste de cette division (modulo).

```
intervalle %% days(1)
```

```
## [1] 2014-10-05 10:00:00 UTC--2014-10-06 08:00:00 UTC
```

```
as.period(intervalle %% days(1))
```

```
## [1] "22H 0M 0S"
```

Le `lubridate` : durée d'un intervalle

```
intervalle %/% days(1)
```

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

```
as.period(intervalle %/% days(1))
```

```
## [1] "4S"
```

Exercices

1. À partir de `x`, trouver la date une semaine auparavant (en termes de durées, et en termes d'époques).

```
x <- "Fri Mar 02 23:40:54 +0000 2012"
```

Quelques fonctions utiles

FONCTION	DESCRIPTION
`x%%y`	Modulo
`x%/%y`	Division entière
`ceiling(x)`	Plus petits entier supérieur ou égal à `x`
`floor(x)`	Plus petits entier inférieur ou égal à `x`
`round(x, digits)`	Arrondi de `x` à digits décimales près
`signif(x, digits)`	Arrondi de `x` à digits chiffres significatifs
`trunc(x)`	Partie entière de `x`

Quelques fonctions utiles

FONCTION	DESCRIPTION
`abs(x)`	Valeur absolue de `x`
`cor(x)`	Matrice de corrélations, si `x` est une matrice ou une data frame
`cor(x, y)`	Corrélation linéaire entre `x` et `y`, ou matrice de corrélations si `x` et `y` sont des matrices ou des data frames
`cummax(x)`	Vecteur dont le ie élément est le maximum des éléments de `x[1]` à `x[i]`
`cummin(x)`	Vecteur dont le ie élément est le minimum des éléments de `x[1]` à `x[i]`
`cumprod(x)`	Vecteur dont le ie élément est le produit des éléments `x[1]` à `x[i]`
`cumsum(x)`	Vecteur dont le ie élément est la somme des éléments de `x[1]` à `x[i]`
`exp(x)`	Exponentielle de `x`
`log(x, base)`	Logarithme de `x`, avec `base=10` par défaut
`max(..., na.rm)`	Maximum du premier paramètre (peut être de type `numeric`, `logical` ou `character`. Si `na.rm = TRUE`, omet les valeurs non disponibles
`mean(x, na.rm, trim)`	Moyenne de `x`. Si `na.rm = TRUE`, omet les valeurs non disponibles. Le paramètre `trim`, compris entre 0 (par défaut) et 0.5 indique la fraction d'observations à retirer de chaque côté de `x` avant de calculer la moyenne des observations restantes

Quelques fonctions utiles

FONCTION	DESCRIPTION
`median(x, na.rm)`	Médiane de `x`. Si `na.rm = TRUE`, omet les valeurs non disponibles
`min(..., na.rm)`	Minimum du premier paramètre (peut être de type `numeric`, `logical` ou `character`. Si `na.rm = TRUE`, omet les valeurs non disponibles
`prod(..., na.rm)`	Produit des éléments du premier paramètre (peut être de type `numeric`, `logical` ou `complex`. Si `na.rm = TRUE`, omet les valeurs non disponibles
`quantile(x, probs, na.rm)`	Fractiles empiriques de `x` d'ordre `probs`. Si `na.rm = TRUE`, omet les valeurs non disponibles.
`range(..., na.rm, finite)`	Étendue du premier paramètre (peut être de type `numeric`, `logical` ou `character`. Si `na.rm = TRUE`, omet les valeurs non disponibles. Si `finite = TRUE`, les éléments non-finis sont omis.
`sd(x, na.rm)`	Écart-type de `x`. Si `na.rm = TRUE`, omet les valeurs non disponibles
`sign(x)`	Signe de `x`

Quelques fonctions utiles

FONCTION	DESCRIPTION
`sqrt(x)`	Racine carrée de `x`
`sum(..., na.rm)`	Somme du premier paramètre (peut être de type `numeric`, `logical` ou `complex`. Si `na.rm = TRUE`, omet les valeurs non disponibles.
`var(x, na.rm)` ou `cov(x)`	Variance corrigée de `x` (division par `n - 1` et non `n`). Si `na.rm = TRUE`, omet les valeurs non disponibles. Si `x` est une matrice ou un data frame, le résultat est la matrice de variance-covariance
`var(x, y, na.rm)` ou `cov(x, y)`	Covariance entre `x` et `y`, ou, dans le cas où `x` et `y` sont des matrices ou des data frames, entre les colonnes de `x` et `y`. Si `na.rm = TRUE`, omet les valeurs non disponibles

Quelques fonctions utiles

FONCTION	DESCRIPTION
`sin(x)`	Sinus de `x`
`cos(x)`	Cosinus de `x`
`tan(x)`	Tangente de `x`
`asin(x)`	Arc-sinus de `x`
`acos(x)`	Arc-cosinus de `x`
`atan(x)`	Arc-tangente de `x`
`sinh(x)`	Sinus hyperbolique de `x`
`cosh(x)`	Cosinus hyperbolique de `x`
`tanh(x)`	Tangente hyperbolique de `x`
`asinh(x)`	Arc-sinus hyperbolique de `x`
`acosh(x)`	Arc-cosinus hyperbolique de `x`
`atanh(x)`	Arc-tangente hyperbolique de `x`

Quelques fonctions utiles

FONCTION	DESCRIPTION
`factorial(x)`	Factorielle de `x`
`choose(n, k)`	Coeffcient binomial C_n^k

Quelques fonctions utiles

FONCTION	DESCRIPTION
`any(x, na.rm)`	Indique si au moins un élément de `x` vaut `TRUE`. Si `na.rm = TRUE`, omet les valeurs non disponibles
`duplicated(x, fromLast)`	Indique les éléments de `x` qui commencent à être dupliqués. Si `fromLast = TRUE`, le calcul est effectué de droite à gauche
head(x, n)`	Vecteur des `n` premières valeurs de `x`
`order(x, decreasing)`	Vecteur des rangs de classement des éléments de `x`. Si ex æquo, les rangs sont classés par ordre croissant
`rev(x)`	Retourne `x` avec ses éléments dans l'ordre inversé
`table(...)`	Tableau de contingence
`sample(x, size, replace, prob)`	Ré-échantillonnage pseudo-aléatoire (avec remise si `replace = TRUE`) de `x` de taille `size`. Le paramètre optionnel `prob` permet d'attribuer des poids aux éléments

Quelques fonctions utiles

FONCTION	DESCRIPTION
`summary(object)`	Retourne une table indiquant pour object : le minimum, le premier quartile, la médiane, la moyenne, le troisième quartile, le maximum et, s'il y en a, le nombre de valeurs manquantes. Les calculs sont effectués en omettant les valeurs manquantes.
`sort(x, decreasing)`	`x` trié par ordre de valeurs croissantes (décroissantes si `decreasing = TRUE`)
`tail(x, n)`	Vecteur des `n` dernières valeurs de `x`
`unique(x, fromLast)`	Conserve uniquement les valeurs uniques de `x`, en retirant les doublons. Si `fromLast = TRUE`, le calcul indiquant si l'élément est dupliqué est effectué de droite vers la gauche
`which.min(x)`	Position du (premier) minimum de `x`
`which.max(x)`	Position du (premier) maximum de `x`

Quelques fonctions utiles

FONCTION	DESCRIPTION
`"["()`	Fonction d'extraction ou de remplacement de parties d'un objet
`"[["()`	Fonction d'extraction ou de remplacement de parties d'un objet. Les noms sont perdus.
`dim(x)`	Liste des tailles de la matrice ou data frame `x`
`nrow(x)`	Nombre de lignes de `x`
`ncol(x)`	Nombre de colonnes de `x`
`dimnames(x)`	Liste des noms des lignes et des colonnes de `x`
`names(x)`	Vecteur du (des) nom(s) de `x`
`colnames(x)`	Nom des colonnes de `x`
`rownames(x)`	Nom des lignes de `x`
`c()`	Concaténation
`cbind(x, y, ...)`	Fusion des matrices ou data frames `x` et `y` par colonnes
`merge(x, y)`	Fusion de deux data frames par noms de colonnes communs
`rbind()`	Fusion des matrices ou data frames `x` et `y` par lignes