# Indexation et recodages avec R et questionr

Julien Barnier

28 novembre 2013

# Au menu


PIZZA
TASTE SPECIAL
SEAFOOD SUPREME
VEGETARIAN
CHICKEN
TROPICANO
MEAT LOVERS
CLASSIC
SUPREMO
GARLIC PIZZA BREAD
BURGERS
HAMBURGER
$5.00
CHEESEBURGER
$5.50
BACON & EGG BURGER
$7.00
FISH BURGER
$7.00
HAWAIIAN BURGER
$7.50
TASTE BURGER
$9.00
CHICKEN SCHNITZEL BURGER
$8.00
VEGETARIAN BURGER
$7.50
FISH & CHIPS
$2.00
4.50
4.50
3.80
2.50
1.80
1.50
1.20
2.50
2.50
3.00
1.50
1.50
1.20
.50
3.00
1.50
2.00
2.50
Fresh Fish of day:

# Au menu

- Manipulation de données : indexation et recodages

# Au menu

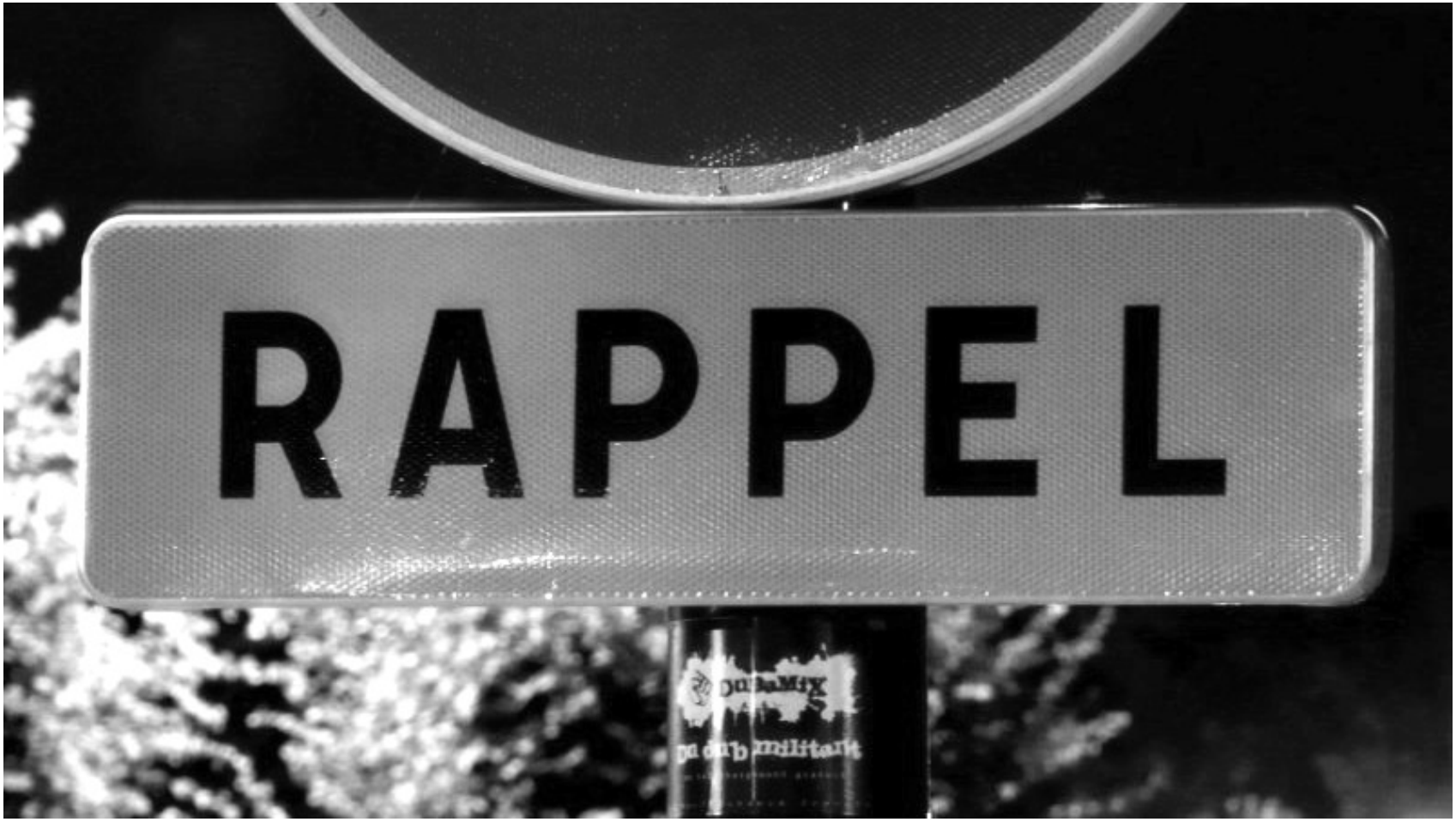
- Manipulation de données : indexation et recodages
- Le document *Introduction à R* :
  `http://alea.fr.eu.org/pages/intro-R`

# Au menu

- Manipulation de données : indexation et recodages
- Le document *Introduction à R* : `http://alea.fr.eu.org/pages/intro-R`
- L'extension `questionr`

```
R> install.packages("questionr")
```

# Rappels



# Rappels

Écrire un script R, c'est appliquer des *fonctions* à des *objets*.

```
R> x <- c(150, 182, 197, 171)
R> mean(x)
```

```
[1] 175
```

# Les objets

Les objets en R servent à stocker des données. Ils peuvent être de différents types :

# Les objets

Les objets en R servent à stocker des données. Ils peuvent être de différents types :

- Nombre

```
R> x <- 3
```

# Les objets

Les objets en R servent à stocker des données. Ils peuvent être de différents types :

- Nombre

```
R> x <- 3
```

- Chaîne de caractère

```
R> x <- "Chihuahua"
```

# Les objets

Les objets en R servent à stocker des données. Ils peuvent être de différents types :

- Nombre

```
R> x <- 3
```

- Chaîne de caractère

```
R> x <- "Chihuahua"
```

- Vecteur

```
R> x <- c(150, 182, 197, 171)
R> y <- c("rouge", "vert", "rouge")
```

# Les objets

Ils peuvent aussi stocker des données beaucoup plus complexes :

- Un tableau de données (`data.frame`)

```
R> df <- read.csv("fichier.csv", stringsAsFactors = FALSE)
```

- Le résultat d'un tri croisé ou d'un test

```
R> tab <- table(df$sexe, df$revenu)
R> test <- chisq.test(tab)
```

- Un graphique
- etc. etc.

# Les fonctions

Elles prennent une suite d'arguments entre parenthèses, et renvoient un résultat :

```
R> c(150, 182, 197, 171)
```

```
[1] 150 182 197 171
```

```
R> mean(x)
```

```
[1] 175
```

# Les fonctions

Elles prennent une suite d'arguments entre parenthèses, et renvoient un résultat :

```
R> c(150, 182, 197, 171)
```

```
[1] 150 182 197 171
```

```
R> mean(x)
```

```
[1] 175
```

Certains arguments sont nommés :

```
R> mean(x, na.rm = TRUE)
```

# Travail avec des données

# Indexation

# Indexation

- Indexer, c'est extraire une partie des valeurs d'un objet.
- L'opérateur d'indexation est le crochet `[]`.
- Il existe plusieurs types d'indexation, selon ce qu'on met entre les crochets.

# Indexation par position

Si on utilise des nombres entiers, on indique les positions des éléments à sélectionner.

# Indexation par position

Si on utilise des nombres entiers, on indique les positions des éléments à sélectionner.

```
R> x <- c(2, 5, 8, 11)
```

# Indexation par position

Si on utilise des nombres entiers, on indique les positions des éléments à sélectionner.

```
R> x <- c(2, 5, 8, 11)

R> x[1]


[1] 2
```

# Indexation par position

Si on utilise des nombres entiers, on indique les positions des éléments à sélectionner.

```
R> x <- c(2, 5, 8, 11)

R> x[1]

[1] 2

R> x[3]

[1] 8
```

# Indexation par position

On peut sélectionner plusieurs éléments en indexant avec un vecteur.

# Indexation par position

On peut sélectionner plusieurs éléments en indexant avec un vecteur.

```
R> x <- c(2, 5, 8, 11)
```

# Indexation par position

On peut sélectionner plusieurs éléments en indexant avec un vecteur.

```
R> x <- c(2, 5, 8, 11)

R> x[c(1, 4)]

[1]  2 11
```

# Indexation par position

On peut sélectionner plusieurs éléments en indexant avec un vecteur.

```
R> x <- c(2, 5, 8, 11)

R> x[c(1, 4)]

[1]  2 11

R> x[c(2, 2, 3, 1)]

[1] 5 5 8 2
```

# Indexation par position

Pour l'instant on a indexé seulement un vecteur

```
R> x[positions]
```

Cette indexation peut également se faire sur un *data frame*.

```
R> df[lignes, colonnes]
```

# Indexation par position

```
   sexe age
1 homme  25
2 femme  37
3 femme  21
```

# Indexation par position

```
   sexe age
1 homme  25
2 femme  37
3 femme  21

R> df[2, 1]


[1] "femme"
```

# Indexation par position

```
   sexe age
1 homme  25
2 femme  37
3 femme  21
```

```
R> df[2, 1]
```

```
[1] "femme"
```

```
R> df[c(1, 3), 2]
```

```
[1] 25 21
```

# Indexation par position

Cas particulier : si on n'indique rien, on sélectionne tout.

# Indexation par position

Cas particulier : si on n'indique rien, on sélectionne tout.

C'est vrai pour un vecteur.

```
R> x <- c(2, 5, 8, 11)
R> x[]
```

```
[1]  2  5  8 11
```

# Indexation par position

C'est vrai aussi, et c'est plus intéressant, pour un *data frame*.

```
   sexe age
1 homme  25
2 femme  37
3 femme  21
```

# Indexation par position

C'est vrai aussi, et c'est plus intéressant, pour un *data frame*.

```
   sexe age
1 homme  25
2 femme  37
3 femme  21
```

```
R> df[1, ]
```

```
   sexe age
1 homme  25
```

# Indexation par position

C'est vrai aussi, et c'est plus intéressant, pour un *data frame*.

```
   sexe age
1 homme  25
2 femme  37
3 femme  21
```

```
R> df[1, ]
```

```
   sexe age
1 homme  25
```

```
R> df[, 2]
```

```
[1] 25 37 21
```

# Exemple sur des données

# Indexation par nom

Si les éléments du vecteur sont nommés, on peut les sélectionner en utilisant leurs noms.

# Indexation par nom

Si les éléments du vecteur sont nommés, on peut les sélectionner en utilisant leurs noms.

```
R> xn <- c(a = 2, b = 57, c = 34)
```

# Indexation par nom

Si les éléments du vecteur sont nommés, on peut les sélectionner en utilisant leurs noms.

```
R> xn <- c(a = 2, b = 57, c = 34)

R> xn["a"]


a
2

R> xn[c("a", "a", "c")]


 a  a  c
 2  2 34
```

# Indexation par nom

On peut également indexer par nom pour les *data frame*, notamment sur les colonnes.

```
   sexe age
1 homme  25
2 femme  37
3 femme  21
```

# Indexation par nom

On peut également indexer par nom pour les *data frame*, notamment sur les colonnes.

```
   sexe age
1 homme  25
2 femme  37
3 femme  21
```

```
R> df[, "sexe"]
```

```
[1] "homme" "femme" "femme"
```

ou l'équivalent :

```
R> df$sexe
```

```
[1] "homme" "femme" "femme"
```

# Exemple sur des données

# Indexation logique

Troisième type d'indexation : l'indexation par un vecteur de valeurs logiques.

# Indexation logique

Troisième type d'indexation : l'indexation par un vecteur de valeurs logiques.

```
R> x <- c(2, 5, 8, 11)
R> x[c(TRUE, TRUE, FALSE, FALSE)]
```

```
[1] 2 5
```

# Indexation logique

Troisième type d'indexation : l'indexation par un vecteur de valeurs logiques.

```
R> x <- c(2, 5, 8, 11)
R> x[c(TRUE, TRUE, FALSE, FALSE)]
```

```
[1] 2 5
```

Le vecteur de valeur logique peut être recyclé.

```
R> y <- 1:20
R> y[c(TRUE, FALSE)]
```

```
 [1]  1  3  5  7  9 11 13 15 17 19
```

# Indexation logique

Là aussi, ça fonctionne sur les *data frame*.

```
   sexe age
1 homme  25
2 femme  37
3 femme  21
```

# Indexation logique

Là aussi, ça fonctionne sur les *data frame*.

```
   sexe age
1 homme  25
2 femme  37
3 femme  21
```

```
R> df[c(TRUE, TRUE, FALSE), c(FALSE, TRUE)]
```

```
[1] 25 37
```

# Tests

# Tests

Un test est une opération logique qui renvoie vrai (TRUE) ou faux (FALSE).

```
R> 1 == 2
```

```
[1] FALSE
```

# Tests

Un test est une opération logique qui renvoie vrai (TRUE) ou faux (FALSE).

```
R> 1 == 2
```

```
[1] FALSE
```

```
R> "Yorkshire" == "Chihuahua"
```

```
[1] FALSE
```

# Tests

Un test est une opération logique qui renvoie vrai (TRUE) ou faux (FALSE).

```
R> 1 == 2
```

```
[1] FALSE
```

```
R> "Yorkshire" == "Chihuahua"
```

```
[1] FALSE
```

```
R> 3 < 5
```

```
[1] TRUE
```

# Tests

Un test est une opération logique qui renvoie vrai (TRUE) ou faux (FALSE).

```
R> 1 == 2
```

```
[1] FALSE
```

```
R> "Yorkshire" == "Chihuahua"
```

```
[1] FALSE
```

```
R> 3 < 5
```

```
[1] TRUE
```

```
R> 1.5 >= mean(c(3, 4, 5))
```

```
[1] FALSE
```

# Tests

On peut appliquer un test directement à un vecteur.

# Tests

On peut appliquer un test directement à un vecteur.

```
R> x <- c(2, 5, 8, 11)
R> y <- c("Rouge", "Vert", "Rouge", "Bleu")
```

# Tests

On peut appliquer un test directement à un vecteur.

```
R> x <- c(2, 5, 8, 11)
R> y <- c("Rouge", "Vert", "Rouge", "Bleu")

R> x == 5
```

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

# Tests

On peut appliquer un test directement à un vecteur.

```
R> x <- c(2, 5, 8, 11)
R> y <- c("Rouge", "Vert", "Rouge", "Bleu")

R> x == 5
```

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

```
R> x > 6
```

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

# Tests

On peut appliquer un test directement à un vecteur.

```
R> x <- c(2, 5, 8, 11)
R> y <- c("Rouge", "Vert", "Rouge", "Bleu")

R> x == 5

[1] FALSE  TRUE FALSE FALSE

R> x > 6

[1] FALSE FALSE  TRUE  TRUE

R> y == "Rouge"

[1]  TRUE FALSE  TRUE FALSE
```

# Indexation par test

On peut indexer avec un vecteur logique

**et**

Un test renvoie un vecteur logique

**donc**

On peut indexer avec un test

# Indexation par test

```
R> x <- c(2, 5, 8, 11)
```

# Indexation par test

```
R> x <- c(2, 5, 8, 11)

R> x == 5

[1] FALSE  TRUE FALSE FALSE
```

# Indexation par test

```
R> x <- c(2, 5, 8, 11)

R> x == 5

[1] FALSE  TRUE FALSE FALSE

R> x[x == 5]

[1] 5
```

# Indexation par test

```
R> x <- c(2, 5, 8, 11)

R> x == 5

[1] FALSE  TRUE FALSE FALSE

R> x[x == 5]

[1] 5

R> x[x > 7]

[1]  8 11
```

# Indexation par test

Le test peut tout à fait porter sur un autre vecteur.

# Indexation par test

Le test peut tout à fait porter sur un autre vecteur.

```
R> taille <- c(151, 187, 178, 169)
R> sexe <- c("Homme", "Femme", "Homme", "Femme")
```

# Indexation par test

Le test peut tout à fait porter sur un autre vecteur.

```
R> taille <- c(151, 187, 178, 169)
R> sexe <- c("Homme", "Femme", "Homme", "Femme")

R> taille[sexe == "Homme"]
```

```
[1] 151 178
```

# Indexation par test

Le test peut tout à fait porter sur un autre vecteur.

```
R> taille <- c(151, 187, 178, 169)
R> sexe <- c("Homme", "Femme", "Homme", "Femme")

R> taille[sexe == "Homme"]
```

```
[1] 151 178
```

```
R> sexe[taille <= 180]
```

```
[1] "Homme" "Homme" "Femme"
```

# Indexation par test

Ça marche évidemment très bien sur les *data frame*.

```
   sexe age
1 homme  25
2 femme  37
3 femme  21
```

# Indexation par test

Ça marche évidemment très bien sur les *data frame*.

```
   sexe age
1 homme  25
2 femme  37
3 femme  21
```

```
R> df[df$age < 30, ]
```

```
   sexe age
1 homme  25
3 femme  21
```

# Indexation par test

Ça marche évidemment très bien sur les *data frame*.

```
   sexe age
1 homme  25
2 femme  37
3 femme  21
```

```
R> df[df$age < 30, ]
```

```
   sexe age
1 homme  25
3 femme  21
```

```
R> df[df$sexe == "homme", "age"]
```

```
[1] 25
```

# Exemples sur les données

# Recodage

# Recodage

On a utilisé l'indexation pour extraire des données. Mais on peut l'utiliser pour remplacer les données sélectionnées.

# Recodage

On a utilisé l'indexation pour extraire des données. Mais on peut l'utiliser pour remplacer les données sélectionnées.

```
R> x <- c(2, 5, 8, 11)

R> x[1]


[1] 2
```

# Recodage

On a utilisé l'indexation pour extraire des données. Mais on peut l'utiliser pour remplacer les données sélectionnées.

```
R> x <- c(2, 5, 8, 11)

R> x[1]


[1] 2

R> x[1] <- 4
R> x


[1]  4  5  8 11
```

# Recodage

On peut remplacer plusieurs valeurs d'un coup.

```
R> x <- c(2, 5, 8, 11)
```

# Recodage

On peut remplacer plusieurs valeurs d'un coup.

```
R> x <- c(2, 5, 8, 11)

R> x[c(2, 3)] <- c(100, 200)
R> x

[1]   2 100 200  11
```

# Recodage

On peut remplacer plusieurs valeurs d'un coup.

```
R> x <- c(2, 5, 8, 11)

R> x[c(2, 3)] <- c(100, 200)
R> x
```

```
[1]   2 100 200  11
```

```
R> x[1:3] <- 0
R> x
```

```
[1]  0  0  0 11
```

# Recodage

C'est encore plus intéressant avec l'indexation par test.

```
R> x <- c(2, 5, 8, 11)
```

# Recodage

C'est encore plus intéressant avec l'indexation par test.

```
R> x <- c(2, 5, 8, 11)

R> x[x == 5] <- 66
R> x

[1]  2 66  8 11
```

# Recodage

C'est encore plus intéressant avec l'indexation par test.

```
R> x <- c(2, 5, 8, 11)

R> x[x == 5] <- 66
R> x
```

```
[1]  2 66  8 11
```

```
R> x[x < 10] <- 0
R> x
```

```
[1]  0 66  0 11
```

# Recodage

Ça s'applique aux *data frame*.

```
   sexe age
1 homme  25
2 femme  37
3 femme  21
```

# Recodage

Ça s'applique aux *data frame*.

```
   sexe age
1 homme  25
2 femme  37
3 femme  21
```

```
R> df$sexe[df$sexe == "femme"] <- "f"
```

# Recodage

Ça s'applique aux *data frame*.

```
   sexe age
1 homme  25
2 femme  37
3 femme  21
```

```
R> df$sexe[df$sexe == "femme"] <- "f"
```

```
R> df
```

```
   sexe age
1 homme  25
2     f  37
3     f  21
```

# Exemples sur les données

# Recodage

On sait désormais recoder une variable qualitative.

**Problèmes** :

- la syntaxe n'est pas forcément intuitive au début
- difficile de s'en souvenir si on ne l'utilise que ponctuellement
- sensible aux fautes de frappe

# Recodage

On sait désormais recoder une variable qualitative.

**Problèmes** :

- la syntaxe n'est pas forcément intuitive au début
- difficile de s'en souvenir si on ne l'utilise que ponctuellement
- sensible aux fautes de frappe

Fonction `irec`

# irec

# Facteurs

# Facteurs

Un facteur (`factor`) est une variable comportant une liste définie de valeurs possibles.

```
R> x <- c("Rouge", "Vert", "Rouge", "Bleu")
```

# Facteurs

Un facteur (`factor`) est une variable comportant une liste définie de valeurs possibles.

```
R> x <- c("Rouge", "Vert", "Rouge", "Bleu")

R> f <- factor(x)
R> f

[1] Rouge Vert  Rouge Bleu
Levels: Bleu Rouge Vert
```

# Facteurs

Un facteur (`factor`) est une variable comportant une liste définie de valeurs possibles.

```
R> x <- c("Rouge", "Vert", "Rouge", "Bleu")

R> f <- factor(x)
R> f


[1] Rouge Vert  Rouge Bleu
Levels: Bleu Rouge Vert

R> f[1] <- "Chihuahua"


Warning: invalid factor level, NA generated
```

# Facteurs

Par défaut les variables qualitatives importées dans R sont souvent converties en facteurs.

C'est notamment le cas des fichiers importés avec `read.table`, `read.csv`...

# Facteurs

Par défaut les variables qualitatives importées dans R sont souvent converties en facteurs.

C'est notamment le cas des fichiers importés avec `read.table`, `read.csv`...

Solutions :

- utiliser l'option `StringsAsFactors=FALSE` de `read.csv`
- convertir le facteur avec `as.character`
- `irec` gère ce cas de figure

# Exemple avec les données

# Facteurs

Un intérêt : les facteurs permettent d'ordonner les niveaux dans les tableaux (tri à plat, tri croisé).

# Facteurs

Un intérêt : les facteurs permettent d'ordonner les niveaux dans les tableaux (tri à plat, tri croisé).

Par défaut les niveaux sont ordonnés par ordre alphabétique.

```
R> dipl <- c("Collège", "Bac", "Sup", "Bac", "Collège")
R> freq(dipl)
```

```
         n  %
Bac      2 40
Collège 2 40
Sup      1 20
NA       0  0
```

# Facteurs

`factor` permet d'ordonner les niveaux.

```
R> diplf <- factor(dipl, levels = c("Collège", "Bac",
+     "Sup"))
R> freq(diplf)
```

```
         n  %
Collège  2 40
Bac      2 40
Sup      1 20
NA       0  0
```

# Facteurs

Même problèmes que pour les recodages :

- la syntaxe n'est pas forcément intuitive au début
- difficile de s'en souvenir si on ne l'utilise que ponctuellement
- sensible aux fautes de frappe

# Facteurs

Même problèmes que pour les recodages :

- la syntaxe n'est pas forcément intuitive au début
- difficile de s'en souvenir si on ne l'utilise que ponctuellement
- sensible aux fautes de frappe

Fonction `iorder`

# Découpage d'une variable numérique en classes

# Découpage d'une variable numérique

On peut convertir une variable quantitative en variable qualitative avec la fonction `cut`.

```
R> cut(var, breaks, include.lowest = FALSE, right = TRUE)
```

# Découpage d'une variable numérique

Même problèmes que pour les recodages :

- la syntaxe n'est pas forcément intuitive au début
- difficile de s'en souvenir si on ne l'utilise que ponctuellement
- sensible aux fautes de frappe

# Découpage d'une variable numérique

Même problèmes que pour les recodages :

- la syntaxe n'est pas forcément intuitive au début
- difficile de s'en souvenir si on ne l'utilise que ponctuellement
- sensible aux fautes de frappe

Fonction `icut`

# C'est fini !