

Manipulation de données avec R

1 - Fondamentaux

Robin Cura & Lise Vaudor
d'après L. Vaudor : [Formation startR \(2018\)](#).

15/10/2018

École Thématique GeoViz 2018

Sommaire

Environnement R

- Organisation de RStudio
- Assignation de variables
- Afficher un objet
- Les classes d'objets
- Les fonctions
- Les packages

Les opérateurs

- Opérateurs arithmétiques
- Opérateurs de comparaison
- Opérateurs logiques
- Indicateurs statistiques

Manipuler des vecteurs

- Créer un vecteur
- Créer un facteur
- Conversion de vecteurs
- Les valeurs manquantes

Manipuler des tableaux

- Créer des tableaux
- Indexation en R
- Les structures de données
- Afficher un tableau

Pourquoi utiliser le logiciel R?

Le langage R est un **langage de programmation** et un **environnement mathématique** utilisé pour le **traitement de données** et l'**analyse statistique**.

Il est en outre d'utilisation **libre** et **gratuite** et peut être téléchargé par exemple à l'adresse suivante:
<http://cran.r-project.org/>

L'installation prend au plus quelques minutes.

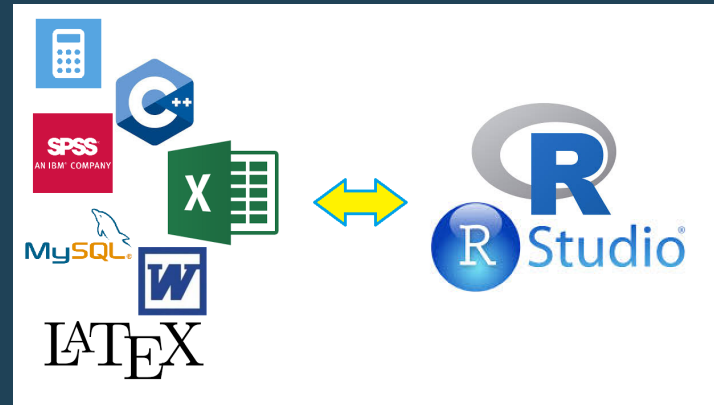
Il permet de réaliser

- des **calculs arithmétiques**
- de la **manipulation de données**
- une très large variété de **graphiques**
- des **scripts** (automatisation de traitements)
- de la **modélisation** et des **simulations numériques**
- une très large variété de **traitements statistiques** (c'est ce pour quoi il est le plus reconnu)
- des **rapports, pages web, applications interactives** et **diaporamas**

Pourquoi utiliser le logiciel R?

Il peut donc remplir les fonctions

- d'une **calculatrice**,
- d'un **tableur**,
- d'un **langage de programmation**,
- d'un **logiciel de statistiques**,
- d'un **logiciel de dessin**
- d'un **éditeur de rapports et de présentations...**

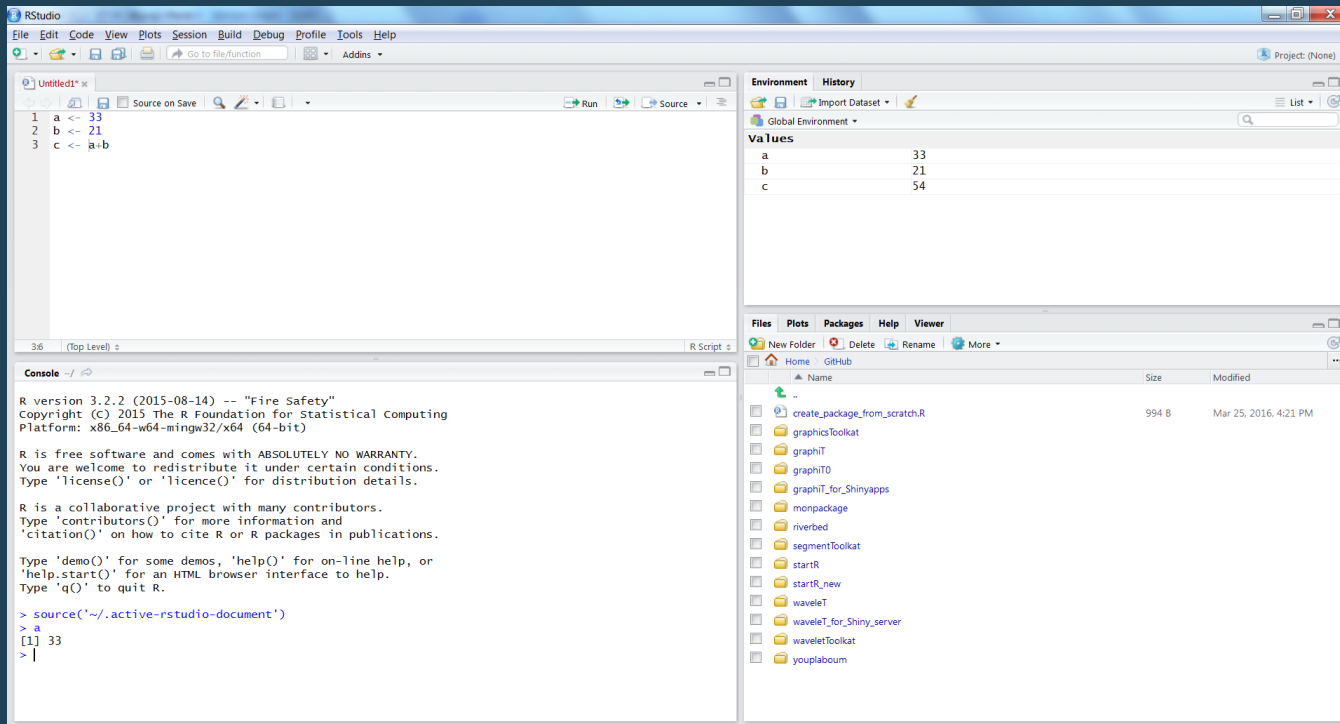


En contrepartie de sa polyvalence et de sa flexibilité, R peut être un peu déroutant au premier abord, car il ne s'agit pas d'un logiciel "clic-boutons": on exécute les différentes opérations à travers l'exécution de **lignes de commande**.

- Pour simplifier l'usage de ce langage, on le mobilise au sein d'un **IDE** adapté : **RStudio**

RStudio

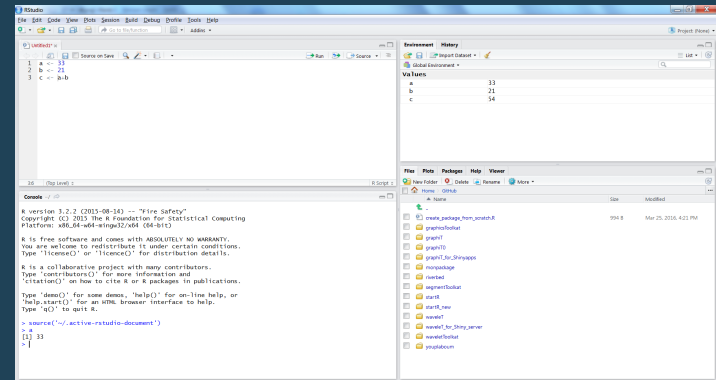
- Nous allons travailler sur un éditeur de script (ou plus précisément un IDE, pour Integrated Development Environment) le logiciel **RStudio**.
- Il est lui aussi **libre et gratuit** et peut être téléchargé à l'adresse suivante: <http://www.rstudio.com/ide/>.



RStudio: Fonctionnalités

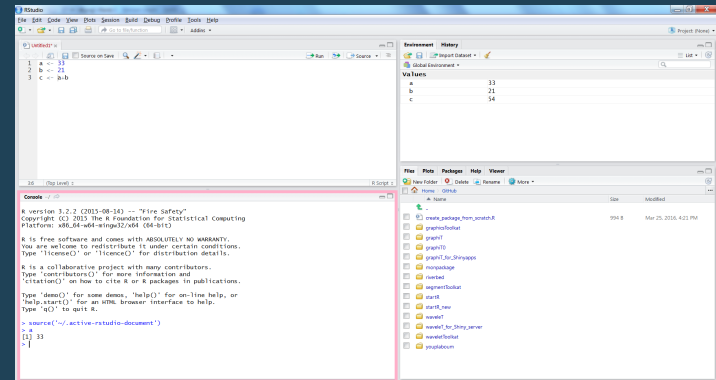
Dans RStudio, quatre zones apparaissent:

- **Source** en haut à gauche,
- **Console** en bas à gauche,
- **Environnement** en haut à droite
- **Plots** en bas à droite



RStudio: zone Console

La zone Console de RStudio correspond en fait à l'**interpréteur R de base**... C'est cette console qui s'ouvre quand vous lancez R (sans RStudio)... Simplement ici, la console est "enrobée" de différents outils pour vous aider à travailler...

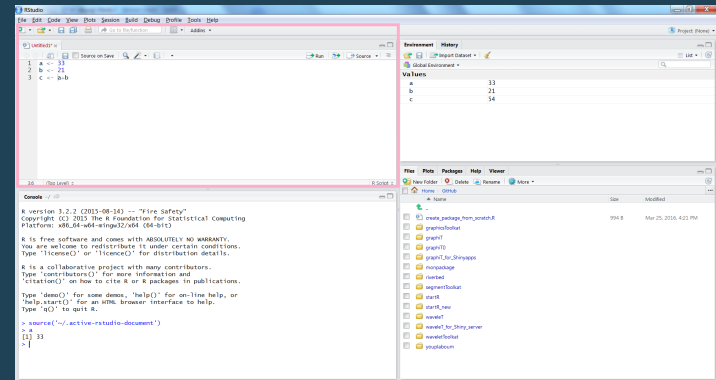


RStudio: zone Source

La zone **Source** constitue l'éditeur de code à proprement parler. C'est dans cette zone que vous allez écrire vos scripts.

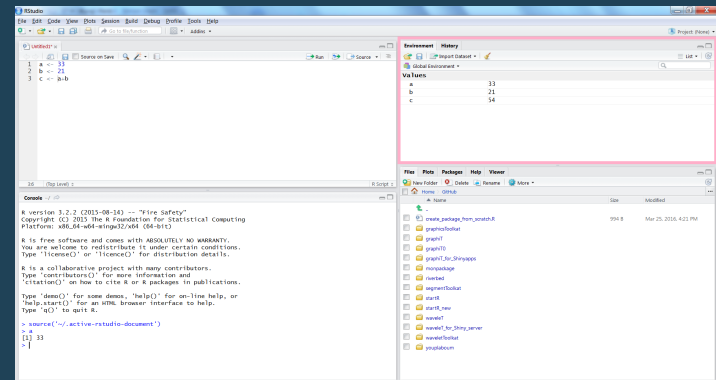
Les calculs sont exécutés dans la zone **Console**. On peut envoyer les codes de la zone "Source" vers la zone "Console"

- grâce au bouton **Run** (qui exécute la ou les lignes de commande sélectionnée(s))
- grâce au bouton **Source** (qui exécute l'ensemble des lignes de commande du script).



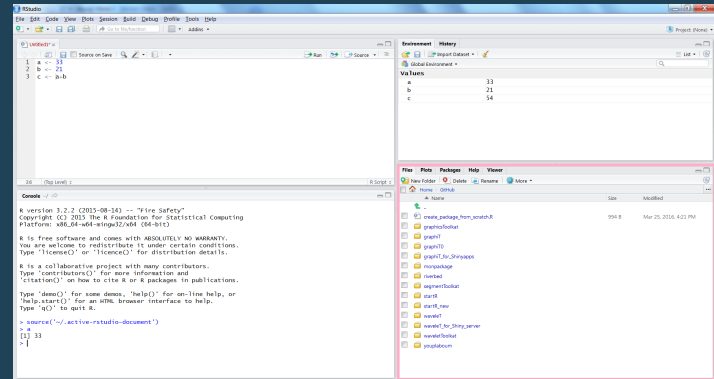
RStudio: zone Environment/History

- onglet **Environment**: il vous permet de consulter l'ensemble des objets de votre environnement
- onglet **History**: il vous permet de consulter l'historique de vos commandes (i.e. l'ensemble des commandes que vous avez exécutées depuis le lancement de votre session).



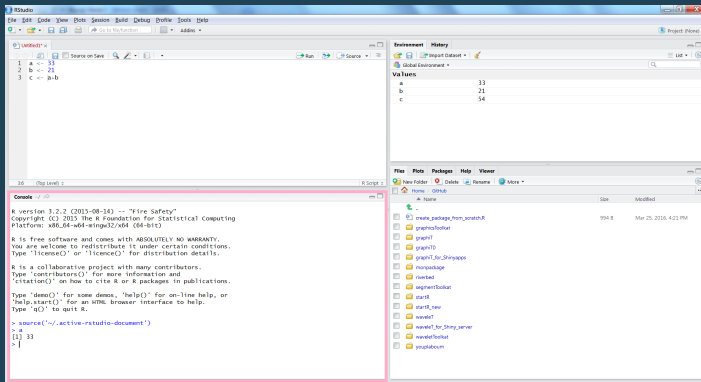
RStudio: zone Files/Plots/Packages/Help

- onglet **Files**: il vous permet de naviguer dans vos dossiers et d'ouvrir/renommer/supprimer vos fichiers.
- onglet **Plots**: c'est là que s'afficheront (par défaut) les graphiques produits. Il vous permet donc de vérifier d'un coup d'oeil vos sorties graphiques...
- onglet **Packages**: vous montre l'ensemble des packages installés et chargés pour la session actuelle.
- onglet **Help**: vous pouvez y consulter les fichiers d'aide associés aux différentes fonctions de R.



Console, commandes

Au lancement de RStudio, une fenêtre (la **console**) apparaît: le symbole ">" indique que R est prêt à exécuter toute ligne de commande que nous allons lui fournir.



Un exemple de **ligne de commande**:

```
2+2
```

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

Taper entrée pour exécuter la commande. R exécute la commande et nous affiche le résultat.

Commentaires, historique des commandes

Commentaires

```
32.7*59.6 # multiplication
```

```
## [1] 1948.92
```

Les indications précédées du symbole **#** sont des **commentaires**. Ils sont ignorés par R mais vous seront très utiles pour **annoter vos scripts**.

Historique

Si l'on exécute plusieurs lignes de commandes dans la console, on peut "récupérer" les **lignes de commandes précédemment exécutées** avec la flèche ↑ ou au contraire en récupérer de plus récentes avec ↓.

Assignment

Lorsque vous exécutez une commande, vous pouvez en observer le résultat directement dans la console:

```
32.7*59.6
```

```
## [1] 1948.92
```

```
53/59
```

```
## [1] 0.8983051
```

Vous pouvez également choisir d'attribuer ce résultat à un **objet**, qui sera référencé comme une **variable**

```
a <- 32.7*59.6  
b <- 53/59
```

On dit qu'on **assigne une valeur à une variable**. On a ainsi créé les objets **a** et **b**.

Assignment

N.B. : En R, l'assignation peut se faire avec le symbole `<-` ou `=`, qui sont équivalents.

```
a = 32.7*59.6  
a <- 32.7*59.6
```

On préférera l'opérateur `<-` qui est plus spécifique et donc non ambigu.

- En revanche, R est **sensible à la casse** donc les deux commandes suivantes créeront deux objets distincts!

```
a <- 32.7*59.6  
A <- 32.7*59.6
```

Environnement

```
a <- 32.7*59.6  
b <- 53/59
```

Lorsque vous exécutez les commandes ci-dessus, **rien ne s'affiche dans la console**. Cela ne signifie pas pour autant que rien ne s'est passé... Vous avez **créé les objets a et b**, qui font désormais partie de votre environnement de travail...

Rappelez-vous, ces objets apparaissent dans la zone **Environnement** de RStudio. Vous pouvez également afficher la liste des objets dans l'environnement global de la manière suivante:

```
ls()
```

```
## [1] "a" "A" "b"
```

Affichage des objets

Pour afficher la valeur des objets dans la console, plusieurs possibilités:

```
a
```

```
## [1] 1948.92
```

```
print(a)
```

```
## [1] 1948.92
```

Vous pourrez par la suite manipuler les objets de différentes façon... Par exemple, ici on peut les utiliser pour de simples opérations arithmétiques:

```
a+b # calcul puis affichage
```

```
## [1] 1949.818
```

```
c <- a+b # calcul et creation d'  
print(c) # affichage
```

```
## [1] 1949.818
```


Création d'objets

Nous avons vu dans la section précédente **comment créer des objets très simples** et comment les **assigner à un environnement**.

Il y a en fait une multitude de types d'objets possibles dans R. Ici nous allons aborder

- les **vecteurs**
- les **facteurs**
- les **tableaux de données**

Création de vecteurs

On appelle **vecteur** toute **séquence d'éléments** de même type, par exemple:

$v1 = \{2.3, 3.6, 1.1, 2.4\}$

$v2 = \{\text{"Paris"}, \text{"Lyon"}, \text{"Rennes"}\}$

ou $v3 = \{\text{TRUE}, \text{FALSE}, \text{FALSE}\}$

En R, ces vecteurs s'écrivent:

```
v1 <- c(2.3, 3.6, 1.1, 2.4)
v1
```

```
## [1] 2.3 3.6 1.1 2.4
```

```
v2 <- c("Paris", "Lyon", "Rennes")
v2
```

```
## [1] "Paris" "Lyon" "Rennes"
```

```
v3 <- c(TRUE, FALSE, FALSE)
v3
```

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

Création de vecteurs

On peut également créer des vecteurs correspondant à:

- des séquences de valeurs régulièrement espacées
- des séquences de valeurs répétées

```
# valeurs de 0 à 6  
# par pas de 2  
v4 <- seq(from = 0,  
          to = 6,  
          by = 2)  
  
v4
```

```
## [1] 0 2 4 6
```

```
# nombres entiers de 0 a 3  
v5 <- 0:3  
v5
```

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

```
# repetition de "date1" :  
# 2 fois  
v6<-rep("date1", 2)  
v6
```

```
## [1] "date1" "date1"
```

```
# repetition du vecteur v6 :  
# 2 fois  
v7<-rep(v6, 2)  
v7
```

```
## [1] "date1" "date1" "date1" "date1"
```

Classes des objets

Les vecteurs peuvent être de classes différentes selon le type de valeurs qu'ils contiennent (mais toutes les valeurs d'un vecteur doivent être d'un même type).

Ils peuvent par exemple être de mode

- **numérique**
- **caractère**
- **logique/booléen**, c'est à dire que les valeurs qu'ils contiennent sont de type vrai / faux (**TRUE** / **FALSE**).

Par exemple, pour v1, v2, et v3:

```
v1 <- c(2.3, 3.6, 1.1, 2.4, 2.5, 10.2)
class(v1)
```

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

```
v2 <- c("Paris", "Lyon", "Marseille")
class(v2)
```

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

```
v3 <- c(TRUE, FALSE, FALSE, TRUE, TRUE)
class(v3)
```

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

Création de vecteurs

Remarquez que l'on peut aussi utiliser `c()` pour combiner plusieurs vecteurs:

```
v4
```

```
## [1] 0 2 4 6
```

```
v5
```

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

```
vglobal <- c(v4,v5)  
vglobal
```

```
## [1] 0 2 4 6 0 1 2 3
```

Création de vecteurs

Si l'on tente quelque chose comme

```
v5
```

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

```
v6
```

```
## [1] "date1" "date1"
```

```
vessai <- c(v5,v6)  
vessai
```

```
## [1] "0" "1" "2" "3" "date1" "date1"
```

R ne renvoie pas de message d'erreur, mais fait en sorte que toutes les valeurs de `vessai` soient d'un même type (des chaînes de caractère ici: voyez les guillemets autour des valeurs de `v5`).

Valeurs manquantes

Il peut arriver que certaines valeurs d'un objet soient **non renseignées**. En R, ces valeurs s'écrivent **NA** (pour **N**ot **A**vailable).

Par exemple:

```
v8 <- c(3.2, NA, 8.9, 42.3, 59.2, NA)
```

Création de facteurs

Les facteurs ressemblent généralement à des vecteurs de mode caractère, à la nuance près qu'ils comprennent généralement plusieurs **niveaux** (**levels**), c'est-à-dire un ensemble fini de modalités possibles :

```
f1 <- factor(c("val1", "val2", "val3", "val2", "val2", "val3"))  
f1
```

```
## [1] val1 val2 val3 val2 val2 val3  
## Levels: val1 val2 val3
```

```
levels(f1)
```

```
## [1] "val1" "val2" "val3"
```

La nuance entre vecteurs et facteurs est importante pour un certain nombre de fonctions implémentées dans R, il est donc assez souvent nécessaire de convertir les vecteurs en facteurs et inversement.

Création d'objets: tableaux de données

Un tableau de données, ou **data . frame**, compte **plusieurs lignes et colonnes**. Les colonnes (ou **variables**) d'un tableau de données **peuvent être de types différents**, et sont **nommées**. Au sein d'une colonne, toutes les valeurs doivent être de même type.

| Espece | Nom | Date | Parle |
|---------|----------|------|-------|
| Chien | Lassie | 1940 | Non |
| Dauphin | Flipper | 1964 | Non |
| Chat | Garfield | 1978 | Oui |
| Eponge | Bob | 1999 | Oui |

]

Création d'objets: tableaux de données

Voici comment créer un tableau de données (`data.frame`) sous R, en assemblant plusieurs vecteurs de même taille:

```
Espece <- c("Chien", "Dauphin", "Chat", "Eponge")
Nom <- c("Lassie", "Flipper", "Garfield", "Bob")
Date <- c(1940, 1964, 1978, 1999)
Parle <- c(FALSE, FALSE, TRUE, TRUE)
t1 <- data.frame(Espece, Nom, Date, Parle)
t1
```

```
##      Espece      Nom Date Parle
## 1   Chien   Lassie 1940 FALSE
## 2 Dauphin Flipper 1964 FALSE
## 3    Chat Garfield 1978  TRUE
## 4  Eponge      Bob 1999  TRUE
```

Création d'objets, conversion d'objets

Pour interroger R quant au type (vecteur, facteur, tableau, matrice, etc.) ou au mode (numérique, caractère, logique, etc.) d'un objet, on utilise les fonctions de type `is.____`.

Par exemple:

```
v6
```

```
## [1] "date1" "date1"
```

```
is.factor(v6)
```

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

```
is.character(v6)
```

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

Création d'objets, conversion d'objets

On peut convertir un objet d'un type ou mode à un autre en utilisant les fonctions de type `as.____`. Par exemple,

```
v6f <- as.factor(v6)
v6f
```

```
## [1] date1 date1
## Levels: date1
```

convertit le vecteur `v6` en facteur pour créer `v6f`.

Indexation d'un vecteur ou d'un facteur

On peut s'intéresser à une partie d'un objet, par exemple un ou plusieurs éléments d'un vecteur ou d'un facteur.

On a accès au i^{eme} élément d'un vecteur/facteur par la commande: `v[i]`

Par exemple:

```
v2
```

```
## [1] "Paris"      "Lyon"        "Marseille"   "Rennes"     "Montpellier"
```

```
v2[4]
```

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

Indexation d'un vecteur ou d'un facteur

On peut s'intéresser à une partie d'un objet, par exemple un ou plusieurs éléments d'un vecteur ou d'un facteur.

On a accès au i^{eme} élément d'un vecteur/facteur par la commande: `v[i]`

Par exemple:

```
v2[1:3] # les trois premières valeurs
```

```
## [1] "Paris"      "Lyon"         "Marseille"
```

```
v2[c(2,4,5)] # les valeurs 2, 4 et 5
```

```
## [1] "Lyon"          "Rennes"       "Montpellier"
```

Attention ! : dans R,
l'indexation commence à 1 et
non pas à 0.

Indexation d'un vecteur ou d'un facteur

Pour un facteur:

```
f1
```

```
## [1] val1 val2 val3 val2 val2 val3  
## Levels: val1 val2 val3
```

```
f1[3:4]
```

```
## [1] val3 val2  
## Levels: val1 val2 val3
```

Remarquez que dans les éléments n° 3 et 4 du facteur **f1**, il n'y a que la valeur "date1". Cependant, "date2" fait toujours partie des niveaux possibles de ce facteur!

Indexation d'un tableau

Si l'on s'intéresse à l'élément d'un `data.frame` "df" qui se situe sur la i^{eme} ligne et sur la j^{eme} colonne, on y accède par: `df[i,j]`

```
df <- data.frame(X = c(1:3),  
                 Y = c(10:12),  
                 Z = c("a", "b", "c"),  
                 stringsAsFactors = FALSE)
```

```
df
```

```
##   X  Y Z  
##  1  1 10 a  
##  2  2 11 b  
##  3  3 12 c
```

Indexation d'un tableau

Si l'on s'intéresse à l'élément d'un `data.frame` "df" qui se situe sur la i^{eme} ligne et sur la j^{eme} colonne, on y accède par: `df[i,j]`

```
df[1,3] # la valeur sur la ligne 1 et la colonne 3
```

```
## [1] "a"
```

```
df[,3] # toutes les valeurs sur la colonne 3
```

```
## [1] "a" "b" "c"
```

```
df[2,] # toutes les valeurs de la ligne 2
```

```
##   X Y Z  
## 2 2 11 b
```

```
df[2,1:2] # les deux premières valeurs de la ligne 2
```

```
##   X Y  
## 2 2 11
```

Indexation d'un tableau

Pour accéder à une colonne par son nom, on peut aussi utiliser l'opérateur `$` suivi du nom de la colonne :

```
df
```

```
##   X  Y Z  
##  1  1 10 a  
##  2  2 11 b  
##  3  3 12 c
```

```
df$X
```

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

Indexation d'un tableau

Ou encore, on peut utiliser des double crochets avec l'index ou bien le nom de la variable à laquelle on s'intéresse:

```
df[[3]]
```

```
## [1] "a" "b" "c"
```

```
i <- 3  
df[[i]]
```

```
## [1] "a" "b" "c"
```

```
df[["Z"]]
```

```
## [1] "a" "b" "c"
```

```
df$Z
```

```
## [1] "a" "b" "c"
```

Opérateurs arithmétiques

Ils permettent d'effectuer des opérations arithmétiques simples, comme des additions, des multiplications, etc.

```
v1
```

```
## [1] 2.3 3.6 1.1 2.4 2.5 10.2 5.1 2.0
```

```
v1 + 4 # addition
```

```
## [1] 6.3 7.6 5.1 6.4 6.5 14.2 9.1 6.0
```

```
v1 - 3 # soustraction
```

```
## [1] -0.7 0.6 -1.9 -0.6 -0.5 7.2 2.1 -1.0
```

```
v1 * 5 # multiplication
```

```
## [1] 11.5 18.0 5.5 12.0 12.5 51.0 25.5 10.0
```

```
v1 / 4 # division
```

```
## [1] 0.575 0.900 0.275 0.600 0.625 2.550 1.275 0.500
```

```
v1^2 # puissance
```

```
## [1] 5.29 12.96 1.21 5.76 6.25 104.04 26.01 4.00
```

Opérateurs de comparaison

Ils permettent de **comparer** des vecteurs entre eux.

```
v0 <- v1[1:3]
v0

## [1] 2.3 3.6 1.1

v0 == 3.6 # égal à

## [1] FALSE TRUE FALSE

v0 != 2.3 # différent de

## [1] FALSE TRUE TRUE
```

```
v0 < 4 # plus petit

## [1] TRUE TRUE TRUE

v0 > 10 # plus grand

## [1] FALSE FALSE FALSE

v0 <= 5 # plus petit ou égal

## [1] TRUE TRUE TRUE

v0 >= 3 # plus grand ou égal

## [1] FALSE TRUE FALSE
```

Opérateurs logiques

Ils permettent de **vérifier si une proposition est vraie** ou non.

```
v0
```

```
## [1] 2.3 3.6 1.1
```

```
!(v0 > 3) # NON logique
```

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

```
v0 < 2 & v0 > 5 # ET logique
```

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

```
v0 < 3 | v0 > 5 # OU logique
```

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

Noter également l'existence de la fonction **is.na()** qui permet d'évaluer si les éléments d'un vecteur sont vides ou non

```
v9 <- c(3.2, NA, 59.2, NA)  
is.na(v9)
```

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

Description des variables: Structure des données

Dans R, un ensemble de fonctions permet d'afficher la **structure**, le **type**, le **contenu** ou encore les **dimensions** d'un objet

```
v9 <- c(3.2, NA, 59.2, NA)
v9
```

```
## [1] 3.2 NA 59.2 NA
```

```
print(v9)
```

```
## [1] 3.2 NA 59.2 NA
```

```
str(v9)
```

```
## num [1:4] 3.2 NA 59.2 NA
```

```
class(v9)
```

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

```
length(v9)
```

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

```
df <- data.frame(X = c(1:4), Y = c(10:13),
                 Z = c("a", "b", "c", "a"),
                 stringsAsFactors = FALSE)
str(df)
```

```
## 'data.frame': 4 obs. of 3 variables:
## $ X: int 1 2 3 4
## $ Y: int 10 11 12 13
## $ Z: chr "a" "b" "c" "a"
```

```
class(df)
```

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

```
dim(df)
```

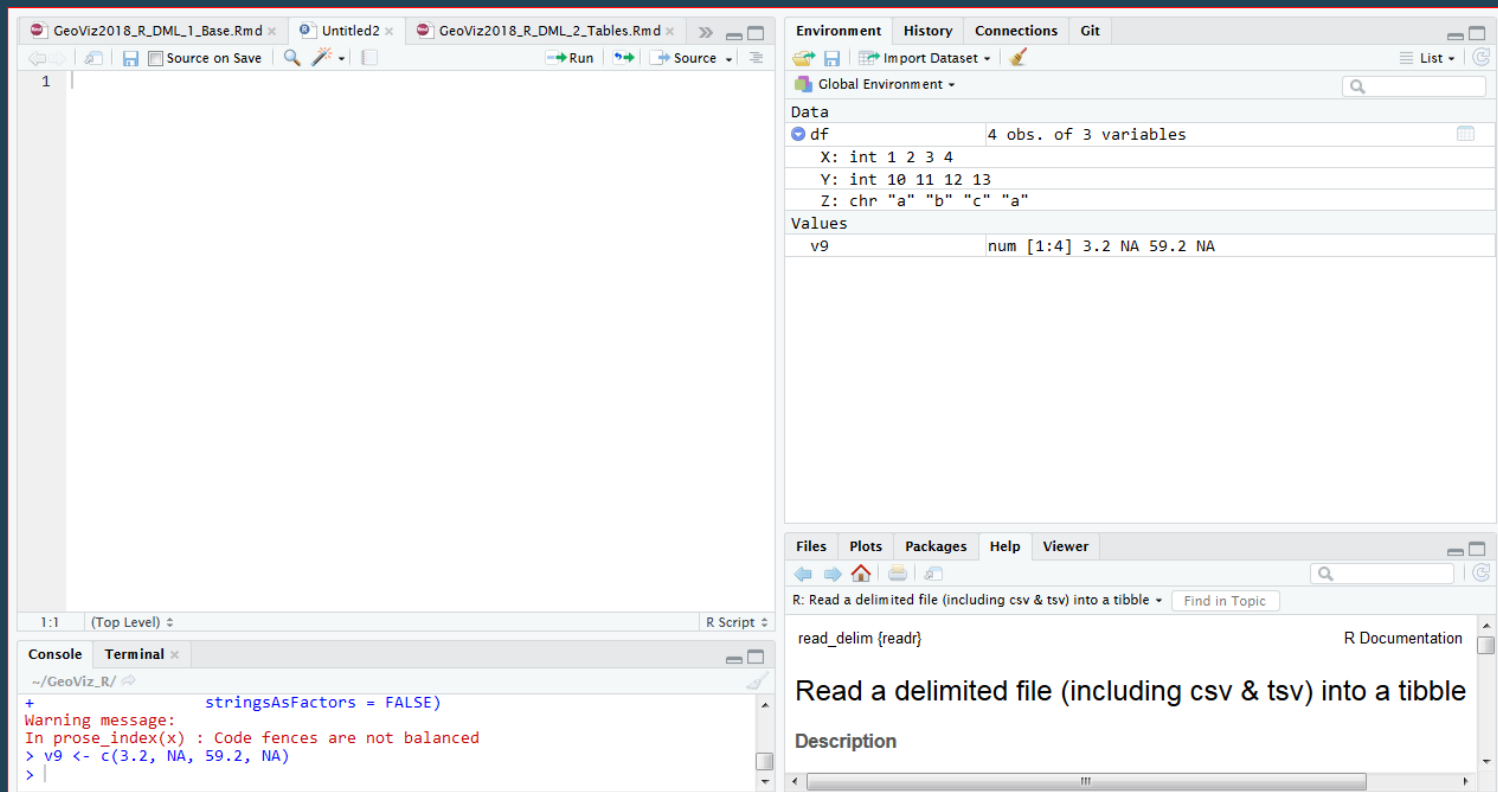
```
## [1] 4 3
```

```
unique(df$Z)
```

```
## [1] "a" "b" "c"
```


Description des variables: Structure des données

- On peut aussi se référer à l'interface graphique de RStudio :



The screenshot displays the RStudio interface. The top pane shows a script editor with a single line of code: `1`. The bottom-left pane is the console, showing the following output:

```
~/GeoViz_R/
+ stringsAsFactors = FALSE)
Warning message:
In prose_index(x) : Code fences are not balanced
> v9 <- c(3.2, NA, 59.2, NA)
>
```

The right-hand side of the interface is divided into two panes. The top pane is the Environment pane, showing the Global Environment with a search bar and a 'List' button. Below this, the 'Data' section shows a data frame named 'df' with 4 observations and 3 variables. The variables are: X (int 1 2 3 4), Y (int 10 11 12 13), and Z (chr "a" "b" "c" "a"). The 'Values' section shows a variable 'v9' with a numeric value of 3.2, NA, 59.2, and NA.

The bottom pane is the Viewer pane, showing the R documentation for the `read_delim` function. The title is 'Read a delimited file (including csv & tsv) into a tibble'. The description is 'Read a delimited file (including csv & tsv) into a tibble'.

Description des variables: Structure des données

- Et pour les `data.frame`, à la fonction `View()`, qui affiche un tableau interactif dans l'interface :

View(df)

The screenshot shows the RStudio interface. On the left, a data frame 'df' is displayed in a table view with 4 rows and 3 columns (X, Y, Z). The data is as follows:

| | X | Y | Z |
|---|---|----|---|
| 1 | 1 | 10 | a |
| 2 | 2 | 11 | b |
| 3 | 3 | 12 | c |
| 4 | 4 | 13 | a |

On the right, the R Documentation for `read_delim()` is shown, including a description and usage examples. The console at the bottom shows the command `View(df)` being executed.

Description des variables: Indicateurs statistiques

Soit x une variable aléatoire suivant une loi normale (`rnorm`) d'espérance (`mean`) 10 et d'écart-type (`sd`) 3

```
x <- round(rnorm(n = 15, mean = 10, sd = 3), digits = 1)
```

```
mean(x)
```

```
## [1] 9.913333
```

```
median(x)
```

```
## [1] 10.4
```

```
min(x)
```

```
## [1] 5.1
```

```
max(x)
```

```
## [1] 13.5
```

```
var(x)
```

```
## [1] 6.375524
```

```
sd(x)
```

```
## [1] 2.52498
```

```
IQR(x)
```

```
## [1] 3.6
```

```
summary(x)
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      5.100   8.350   10.400   9.913   11.950   13.500
```

Description des variables: Indicateurs statistiques

N.B. : Ces fonctions sont aussi valables pour les colonnes d'un `data.frame` :

```
df <- data.frame(X = c(1:4),  
                Y = c(10:13),  
                Z = c("a", "b", "c", "a"),  
                stringsAsFactors = FALSE)
```

```
mean(df$X)
```

```
## [1] 2.5
```

```
median(df$Y)
```

```
## [1] 11.5
```

```
min(df[,c(1,2)])
```

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

```
max(df[,c(1,2)])
```

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

```
summary(df$X)
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.   
##      1.00   1.75   2.50   2.50   3.25   4.00
```

```
summary(df)
```

```
##           X           Y           Z   
##  Min.    :1.00  Min.    :10.00  Length:4   
##  1st Qu.:1.75  1st Qu.:10.75  Class :character   
##  Median :2.50  Median :11.50  Mode  :character   
##  Mean    :2.50  Mean    :11.50   
##  3rd Qu.:3.25  3rd Qu.:12.25   
##  Max.    :4.00  Max.    :13.00
```

Description des variables: Indicateurs statistiques

- Pour décrire un `data.frame`, des fonctions spécifiques sont disponibles

```
df <- data.frame(X = c(1:10),  
                 Y = c(30:21),  
                 Z = rep(c("a", "b", "c", "a", "c"), 2),  
                 stringsAsFactors = FALSE)
```

```
nrow(df) # Nombre de lignes (row)
```

```
## [1] 10
```

```
ncol(df) # Nombre de colonnes (col)
```

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

```
colnames(df) # Nom des colonnes
```

```
## [1] "X" "Y" "Z"
```

```
head(df) # 5 premières lignes
```

```
##   X  Y Z  
## 1 1 30 a  
## 2 2 29 b  
## 3 3 28 c  
## 4 4 27 a  
## 5 5 26 c  
## 6 6 25 a
```

```
tail(df) # 5 dernières lignes
```

```
##   X  Y Z  
## 5  5 26 c  
## 6  6 25 a  
## 7  7 24 b  
## 8  8 23 c  
## 9  9 22 a  
## 10 10 21 c
```

Fonctions: remarques

Nous avons d'ores et déjà utilisé un certain nombre de fonctions, comme

- `c()`,
- `seq()`,
- `rep()`,
- `data.frame()`,
- `mean()`, etc.

Toutes les fonctions que nous avons utilisées jusqu'à présent sont définies sur le **package de base** de R.

Les fonctions sont des **objets** qui s'écrivent avec des **parenthèses**, dans lesquelles l'utilisateur précise la **valeur des arguments** si besoin est. Ces arguments peuvent être **obligatoires** ou **optionnels** :

```
quantile(x = x, probs = 0.1)
```

```
## 10%  
## 6.54
```

L'argument **x** est obligatoire, et l'argument **probs** est optionnel. On peut ainsi ne passer que l'argument **x** à la fonction:

```
quantile(x = x)
```

```
## 0% 25% 50% 75% 100%  
## 5.10 8.35 10.40 11.95 13.50
```

Fonctions: remarques

Si l'on passe les arguments à la fonction **dans le bon ordre**, on n'a pas besoin de préciser le nom des arguments. Ainsi, il est possible d'appeler la fonction **quantile** des deux manières suivantes:

```
quantile(x = x, probs = 0.1)
```

```
## 10%  
## 6.54
```

```
quantile(x, 0.1)
```

```
## 10%  
## 6.54
```

En revanche, l'appel suivant produira une erreur:

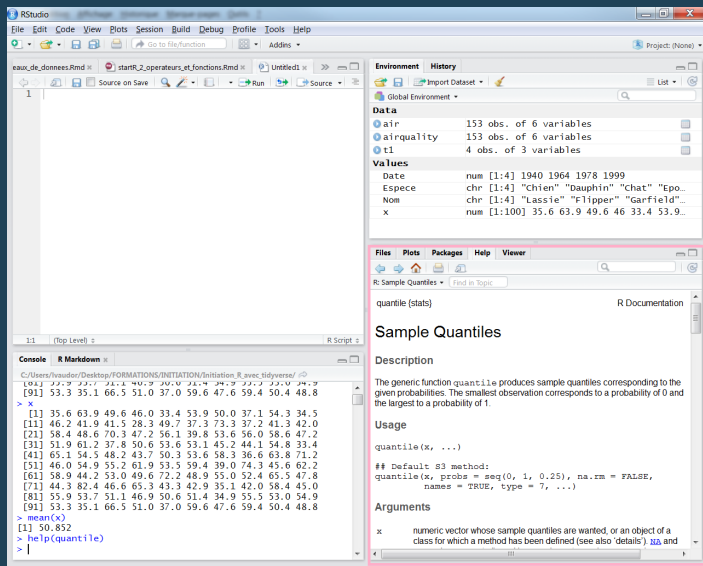
```
quantile(0.1,x)
```

```
## Error in quantile.default(0.1, x): 'probs' outside [0,1]
```

Fonctions: remarques

Pour accéder aux **informations** quant aux **arguments** d'une fonction, on peut consulter l'aide associée des deux façons suivantes:

```
help(quantile)  
?quantile
```



The screenshot shows the RStudio interface. The console on the left displays the execution of `mean(x)` and `help(quantile)`. The Environment pane on the right shows the global environment with data objects like `air`, `airquality`, and `t1`. The Files pane on the right shows the help page for `quantile`, including a description and usage instructions.

```
mean(x)  
[1] 50.852  
> help(quantile)  
>
```

Sample Quantiles

Description

The generic function `quantile` produces sample quantiles corresponding to the given probabilities. The smallest observation corresponds to a probability of 0 and the largest to a probability of 1.

Usage

```
quantile(x, ...)
```

Default S3 method:

```
quantile(x, probs = seq(0, 1, 0.25), na.rm = FALSE,  
        names = TRUE, type = 7, ...)
```

Arguments

- `x` numeric vector whose sample quantiles are wanted, or an object of a class for which a method has been defined (see also 'details'). `NA` and `NaN` values are ignored.

Packages

Les packages sont des **paquets de fonctions** visant à réaliser des tâches un peu particulières. L'installation de base de R vous installe, par défaut, un certain nombre de packages (**base**, **methods**, **stats**, **graphics**, etc.)

Dans la suite de ce cours, nous serons amenés à utiliser le package **dplyr** qui sert à manipuler des tableaux de données.

Pour être en mesure d'utiliser les fonctions de ce package, il faut:

- **Installer le package**: les fonctions du package sont alors téléchargées depuis un serveur distant et enregistrées sur le disque dur de votre ordinateur:

```
install.packages("dplyr")
```

- **Charger le package** (les fonctions du package sont chargées dans l'environnement R pour la session en cours)

```
library(dplyr)
```