

Séance 2: Introduction au langage R

<https://economuds.github.io/BIO109/cours2/>

Dominique Gravel
Laboratoire d'écologie intégrative



Séance 2

- ✓ Ces diapositives sont disponibles en version HTML (Web) et en **PDF**.
- ✓ L'ensemble du matériel de cours est disponible sur la page du portail **moodle**.

Note: Il est préférable d'ouvrir le document HTML (Web) avec Firefox ou Google Chrome.

Contenu du cours

1. Interagir avec R
2. Lire et écrire des objets
3. Manipuler des objets

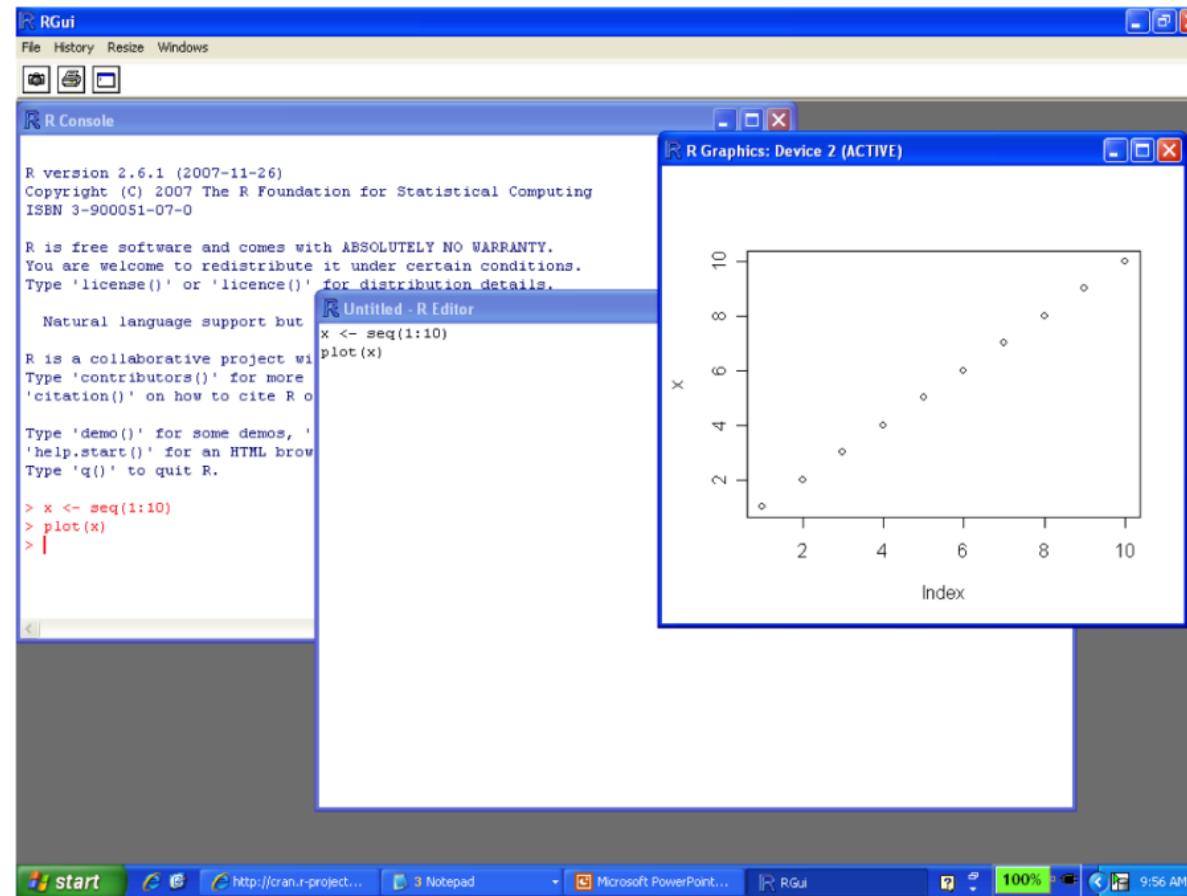
5 raisons pour utiliser R

R est un langage de programmation en source libre conçu pour l'analyse statistique, l'analyse de données et la visualisation. R n'est cependant pas optimisé pour la performance, mais néanmoins il est fort utile pour l'apprentissage de la programmation en raison de sa convivialité (!) et de sa facilité de **débuggage** (!!)

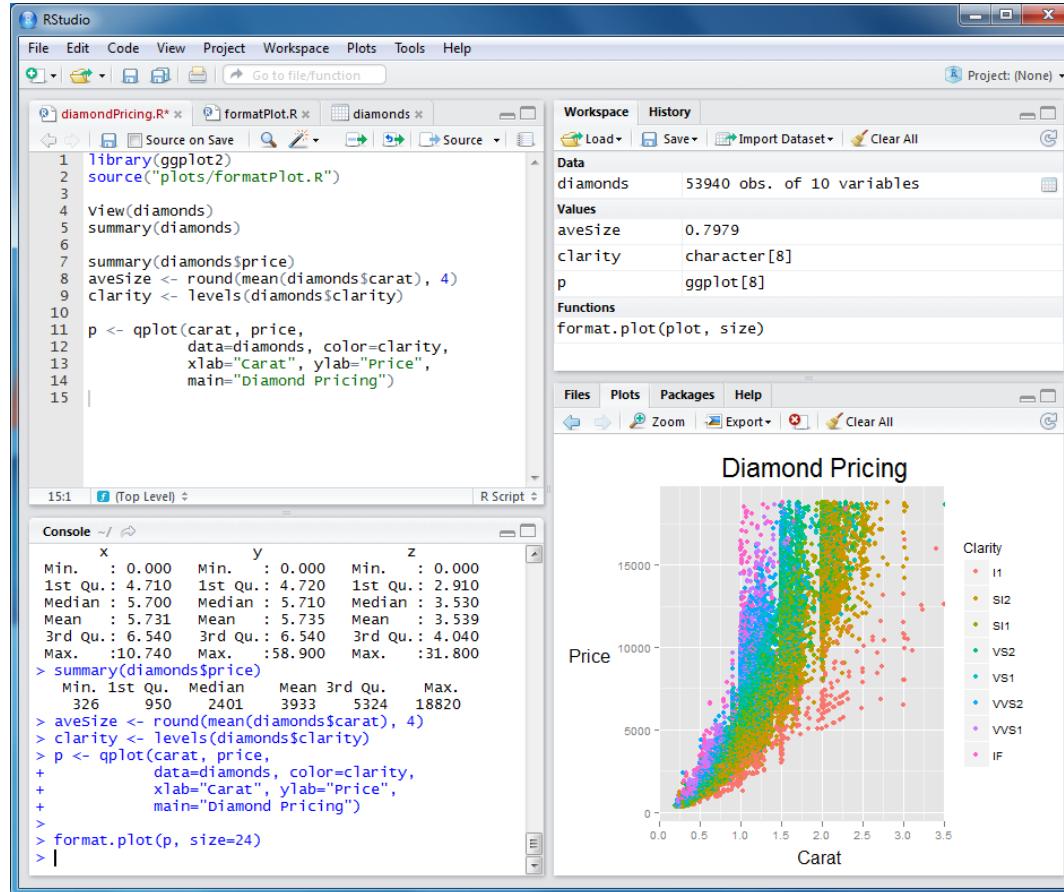
Les arguments en faveur de R:

1. Gratuit et ouvert
2. Une façon de communiquer avec la communauté scientifique
3. Diversité d'utilisateurs et de contributions
4. En phase avec les développements les plus récents en écologie
5. Intégration du langage de programmation avec les outils de visualisation et de manipulation des données

Interagir avec R sur la console



Utiliser R Studio



Exercice : installer et lancer R Studio



Exercice : interaction avec la ligne de commande

Le plus simple est d'utiliser R comme une calculatrice.

Que donne l'opération suivante ?

```
2 + 16 * 24 - 56
```

Répertoire de travail

Savoir où on est

```
getwd()
```

```
## [1] "/Users/SteveVissault/Documents/Git/BIO109/cours2/pres"
```

Répertoire de travail

Changer le répertoire de travail

- ✓ Pour les utilisateurs Mac et Linux:

```
setwd("/home/DominiqueGravel/BIO109/cours2/donnees/")
```

- ✓ Pour les utilisateurs Windows:

```
setwd("c:/DominiqueGravel/BIO109/cours2/donnees/")
```

Le **c:** correspond à la partition du disque.

Environnement de travail

Obtenir le contenu du dossier dans le répertoire de travail

```
dir()
```

```
## [1] "assets"      "donnees"       "index.html"    "index.md"  
## [5] "index.Rmd"    "libraries"     "MaListe.Rdata"
```

Obtenir la liste des objets en mémoire

```
ls()
```

```
## character(0)
```

Environnement de travail

Effacer un objet en mémoire:

```
test <- 1  
ls()
```

```
## [1] "test"
```

```
rm(test)  
ls()
```

```
## character(0)
```

Le concept d'objet

Un objet peut contenir de l'information de toute nature: un tableau de données, les résultats d'une analyse, une figure, une fonction

La commande pour créer un objet **a** est composé de trois parties:

1. une valeur d'intérêt
2. un identifiant (soit un nom d'objet)
3. l'opérateur d'assignation

```
a <- c(1,2)  
a
```

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

Le concept d'objet

- ✓ Notons que les opérateurs `<-` et `=` sont équivalents pour attribuer une valeur à un objet.
- ✓ L'opérateur `<-` peut être difficile à lire ou confondu avec le signe `-`.

La recommandation est donc de limiter le signe `=` à des opérations mathématiques et d'utiliser `<-` pour l'assignation de valeurs.

Les types de valeurs assignable à un objet

Assigner une valeur de type **numérique**

Les données numériques peuvent être déclarées de différentes manières

```
a <- 1  
a
```

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

```
b <- 1.1  
b
```

```
## [1] 1.1
```

```
c <- 1e-6  
c
```

```
## [1] 1e-06
```

Assigner une valeur de type **caractère**

On peut également déclarer des séries de caractères

```
test <- "test"  
test
```

```
## [1] "test"
```

Important - Les valeurs de type **caractère** doivent être placé entre guillemets. Cette nomenclature permet de différencier une valeur du nom d'un objet.

Assigner une valeur de type **caractère**

Finalement, des séries de caractères peuvent être collées entre elles pour créer des chaînes de caractères.

```
collage <- paste("a", "b", sep = "")  
collage
```

```
## [1] "ab"
```

Assigner une valeur de type **booléen**

Le type **booléen** (logical) permet de représenter les états **TRUE** (1) et **FALSE** (0) et de faire des opérations mathématiques sur ces objets.

```
vrai <- TRUE  
faux <- FALSE  
vrai
```

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

```
vrai + faux
```

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

Assigner une valeur de type **facteur**

R est d'abord un langage utilisé pour les statistiques, et par conséquent on y retrouve un type de données utilisé pour la réalisation de tests d'hypothèse, qui n'est pas standard à tous les langages. Les facteurs sont des données de catégories comme des traitements expérimentaux.

```
MesFacteurs <- factor(c("trait1","trait2","trait3","trait2","trait3","trait3","trait1","trait2"))
MesFacteurs
```

```
## [1] trait1 trait2 trait3 trait2 trait3 trait3 trait1 trait2
## Levels: trait1 trait2 trait3
```

Conversion entre type de valeur

Conversion entre type de valeur

R permet de convertir des objets en différents types de données lorsque le contenu le permet.

Conversion vers les types **numérique** et **caractère**:

```
as.numeric(c("4", "6"))
```

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

```
as.character(c(4, 6))
```

```
## [1] "4" "6"
```

Conversion entre type de valeur

Conversion vers les types **facteurs** et nombre **entier**:

```
as.integer(2.6)
```

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

```
as.factor(c("4", "6"))
```

```
## [1] 4 6  
## Levels: 4 6
```

Conversion entre type de valeur

Conversion vers le type **boléen**:

```
as.logical(0)
```

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

```
as.logical(1)
```

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

```
as.logical(2)
```

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

Les types d'objet

Les types d'objet

Depuis le début du cours, nous avons assigné une seule valeur à un objet. Lorsqu'un objet contient une seule valeur, on dit que c'est un objet **scalaire**.

```
a <- 1  
a
```

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

Cependant, un objet peut cependant contenir plusieurs valeurs. Il sera défini alors comme un **vecteur**.

Type d'objet: les **vecteurs**

Le vecteur est un des objets les plus importants de R.

La façon la plus simple de déclarer un vecteur est:

```
MonPremierVecteur <- c(1,2,3,4,5)  
MonPremierVecteur
```

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

Note - D'autres méthodes seront présentées plus loin.

Type d'objet: les **vecteurs**

La force de R repose dans sa capacité à réaliser des opérations **vectoriel**.

Il est ainsi possible d'additionner simplement deux vecteurs d'une même dimension de la façon suivante:

```
MonPremierVecteur <- c(1,2,3,4,5)
MonDeuxiemeVecteur <- c(11,12,13,15,16)
MonPremierVecteur + MonDeuxiemeVecteur
```

```
## [1] 12 14 16 19 21
```

Exercice

Créez un vecteur contenant les valeurs **[-1, 2, 5, 9]**. Vous pouvez calculer la racine carrée de ces nombres au moyen de la fonction **sqrt()**

Types d'objets: les **vecteurs**

On peut accéder à une position sur le vecteur au moyen d'un **index**, indiqué par les **[]**

```
MonPremierVecteur <- c(1,2,3,4,5)  
MonPremierVecteur[3]
```

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

On obtient un **NA** si on tente d'accéder à une position qui n'existe pas:

```
MonPremierVecteur <- c(1,2,3,4,5)  
MonPremierVecteur[6]
```

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

Exercice

Au moyen du vecteur créé précédemment, calculer le produit des valeurs aux positions **2** et **4**.

Types d'objets: les **vecteurs**

On obtient la dimension d'un vecteur ainsi:

```
length(MonPremierVecteur)
```

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

Types d'objets: les matrices

L'extension naturelle d'un vecteur est une **matrice**, soit une collection de vecteurs. R est également optimisé pour réaliser des opérations mathématiques et de manipulation de données sur ce type d'objet.

La commande de base pour créer une matrice est **matrix()**

```
MaMatrice <- matrix(c(1,2,3,4,5,6), nrow = 2, ncol = 3)
MaMatrice
```

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

Types d'objets: les matrices

On accède à la ligne **i** et la colonne **j** au moyen de la commande suivante

```
MaMatrice[1,2]
```

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

Types d'objets: les **matrices**

On peut aussi accéder à des lignes ou des colonnes entières

```
MaMatrice[1, ]
```

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

```
MaMatrice[,1]
```

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

Types d'objets: les **matrices**

On obtient la dimension de la matrice ainsi

```
dim(MaMatrice)
```

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

Types d'objets: les **matrices**

Ou encore

```
nrow(MaMatrice)
```

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

```
ncol(MaMatrice)
```

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

Types d'objets: les **matrices**

Les noms de colonnes et de lignes peuvent être modifiées

```
colnames(MaMatrice) = c("A", "B", "C")
rownames(MaMatrice) = c("X", "Y")
MaMatrice
```

```
##   A B C
## X 1 3 5
## Y 2 4 6
```

Exercice

Créez une matrice de 2 lignes et 5 colonnes remplie de chiffres tirés au hasard. Pour ce faire, vous pouvez utiliser la fonctions **runif()**. Calculez la somme de la première colonne au moyen de la fonction **sum()**

Astuce : pour obtenir de l'aide sur une fonction, essayez

```
?runif()
```

Type d'objets: les **listes**

R peut organiser également des collections d'objets sous forme de liste. Ces collections peuvent être très hétérogènes et rassembler par exemple, des vecteurs et matrices, mais aussi des objets plus complexes et hiérarchiques comme des résultats d'analyses statistiques.

La création et l'indexation des listes est très similaire à celle des vecteurs et matrices

```
MaListe = list()
MaListe[[1]] = c(1,2)
MaListe[[2]] = matrix(c("A","B","C","D"), nrow = 2, ncol = 2)
```

MaListe[[1]] permet d'obtenir l'objet contenu par le premier niveau de la liste.

Type d'objets: les **listes**

Et de même on peut nommer les items d'une liste

```
names(MaListe) <- c("Element1", "Element2")  
MaListe
```

```
## $Element1  
## [1] 1 2  
##  
## $Element2  
##      [,1] [,2]  
## [1,] "A"  "C"  
## [2,] "B"  "D"
```

Type d'objets: les `data.frame`

Le `data.frame` se décrit comme un tableau de données, avec les rangées **i** et les colonnes **j**, ainsi que des noms de colonnes.

```
df <- data.frame(a = c(1:3), b = c(11:13))
df
```

```
##   a   b
## 1 1 11
## 2 2 12
## 3 3 13
```

Il n'est cependant pas un objet mathématique sur lequel on peut effectuer des opérations mathématiques comme les matrices.

Type d'objets: les **data.frame**

Un **data.frame** peut avoir différents types de valeur pour chaque colonne

```
df2 <- data.frame(a = c(1:3), traitement = factor(c("trait1","trait2","trait3")))
df2
```

```
##   a   traitement
## 1 1   trait1
## 2 2   trait2
## 3 3   trait3
```

Cette capacité à entreposer plusieurs types de valeur est ce qui distingue un **data.frame** d'une matrice.

Type d'objets: les **data.frame**

On peut néanmoins transformer un **data.frame** en matrice

```
as.matrix(df)
```

```
##      a   b  
## [1,] 1 11  
## [2,] 2 12  
## [3,] 3 13
```

```
as.matrix(df2)
```

```
##      a    traitement  
## [1,] "1"  "trait1"  
## [2,] "2"  "trait2"  
## [3,] "3"  "trait3"
```

Type d'objets: **data.frame**

Ou inversement transformer une matrice en **data.frame**

```
mat <- matrix(c(1:6), nrow = 3, ncol = 2)
as.data.frame(mat)
```

```
##   V1 V2
## 1  1  4
## 2  2  5
## 3  3  6
```

Interagir avec ses fichiers

Lire un fichier

La lecture de fichiers est souvent l'étape la plus frustrante lorsque l'on travail avec R.

Pour ce faire, nous utiliserons les fichiers sous format **CSV** (*Comma Separated values*) qui sont des fichiers de texte brut dont chaque ligne dispose d'un séparateur de colonne.

Lire un fichier

Si j'ouvre le fichier **arbres.csv** avec un bloc note, j'obtiens les 5 premières lignes suivantes:

```
id_bor;borx;bory;arbre;esp;multi;mort;dhp  
0-0;0;0;34501;acpe;FAUX;FAUX;82  
0-0;0;0;34502;acpe;VRAI;FAUX;26  
0-0;0;0;34502;acpe;VRAI;FAUX;98  
0-0;0;0;34503;acpe;FAUX;FAUX;73
```

On voit ici le point virgule comme séparateur de colonne. La première ligne corresponds au nom des colonnes.

Note: Garder à l'esprit que connaitre la structure du fichier facilite grandement l'importation des données dans R.

Lire un fichier Excel

Les fichiers **.xlsx** peuvent être lus au moyen de librairies spécialisées, mais il est préférable pour R d'utiliser un format multi-plateforme ouvrable sans logiciels propriétaires.

Ouvrir un fichier .csv dans R

Nous allons utiliser le fichier **arbres.csv** pour cet exemple.

On peut lire le fichier au moyen de la commande

```
arbres <- read.table(file = './donnees/arbres.csv', header = T, dec = ".", sep = ";")  
head(arbres)
```

```
##   id_bor borx bory arbre esp multi mort dhp  
## 1     0-0     0     0 34501 acpe FAUX FAUX  82  
## 2     0-0     0     0 34502 acpe VRAI FAUX  26  
## 3     0-0     0     0 34502 acpe VRAI FAUX  98  
## 4     0-0     0     0 34503 acpe FAUX FAUX  73  
## 5     0-0     0     0 34504 acpe FAUX VRAI  28  
## 6     0-0     0     0 34506 fagr FAUX FAUX  26
```

Lire un fichier .csv: anatomie des arguments

- ✓ **file** : nom du fichier à lire
- ✓ **header** : indique s'il y a un entête avec les noms de colonnes
- ✓ **dec** : caractère utilisé pour délimiter les décimales
- ✓ **sep** : caractère utilisé pour séparer les colonnes
- ✓ **quote** : spécifie si les chaînes de caractère sont entouré par des guillemets.

Note : l'objet retourné est par défaut un **data.frame**. Il peut être ensuite converti, au besoin.

Lire un fichier: les erreurs courantes

- ✓ Mauvais nom de fichier
- ✓ Mauvais répertoire de travail
- ✓ Guillemets
- ✓ Type de séparateur de colonnes
- ✓ Présence de points
- ✓ Nom des colonnes
- ✓ Entrées vides
- ✓ Présence de caractères invisibles
- ✓ Mauvais encodage des données
- ✓ Type de données et d'objets

Écrire des fichiers CSV

Il y a une grande diversité de façons d'enregistrer sur le disque des objets provenant de R. La façon la plus simple, versatile et qui permet les échanges entre différents logiciels et systèmes d'exploitation est d'écrire sous format **CSV**.

La syntaxe pour écrire une **matrice** ou un **data.frame** est aussi simple que pour lire un fichier **CSV**.

```
mat2 <- matrix(runif(n = 10,min = 0,max = 1),nrow = 5,ncol = 10)
write.csv2(mat2, file = "mat.csv")
```

On privilie l'utilisation de la fonction **write.csv2()** qui utilise le point-virgule comme séparateur par défaut (**write.csv()**, la virgule).

Écrire des fichiers .Rdata avec `save()`

Parfois les objets que l'on souhaite enregistrer ont une structure plus complexe qu'un tableau de données. R permet d'enregistrer ces objets dans un format qui lui est unique, le **.Rdata**. Ces objets sont compressés pour minimiser l'espace disque et ne peuvent être lus uniquement par R.

```
MaListe <- list()
MaListe[[1]] <- 1
MaListe[[2]] <- c(1:10)
save(MaListe, file = "MaListe.Rdata")
```

NOTE - `save()` permet également de sauvegarder plusieurs objets.

```
save(MaListe,MonDataFrame, file = "MaListe.Rdata")
```

Lire des fichiers **.Rdata** avec **load()**

Puisque le fichier **.Rdata** est spécifique à R, il s'agit peut-être du format le plus facile à lire puisque R prend en charge la mise en forme de l'objet, les noms et les types de données

```
load("./MaListe.Rdata")
ls()
```

```
## [1] "a"                  "arbres"             "b"
## [4] "c"                  "collage"            "df"
## [7] "df2"                "faux"               "MaListe"
## [10] "MaMatrice"          "mat"                "MesFacteurs"
## [13] "MonDeuxiemeVecteur" "MonPremierVecteur" "test"
## [16] "vrai"
```

Lire des fichiers: `load()` et `.Rdata`

Faites attention, si le nom de l'objet contenu dans le fichier `.Rdata` est le même qu'un objet en mémoire, il va écraser ce premier objet.

```
MaListe <- list("Hello", "World")
load("./MaListe.Rdata")
MaListe
```

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

Pour éviter cette erreur, il faut utiliser les fonctions `saveRDS()` et `readRDS()`.

Quelques commandes utiles

Il est important de vérifier la structure de l'objet après son importation dans l'environnement R.

```
head(arbres, n = 3)
```

```
##   id_bor borx bory arbre esp multi mort dhp
## 1     0-0     0     0 34501 acpe  FAUX FAUX  82
## 2     0-0     0     0 34502 acpe  VRAI FAUX  26
## 3     0-0     0     0 34502 acpe  VRAI FAUX  98
```

```
tail(arbres, n = 3)
```

```
##           id_bor borx bory arbre esp multi mort dhp
## 17633 180-980   180   980 17271 pиру  FAUX FAUX 125
## 17634 180-980   180   980 17272 pиру  FAUX FAUX 190
## 17635 180-980   180   980 17273 бел  FAUX FAUX 210
```

Quelques commandes utiles

Il est important de vérifier la structure de l'objet après son importation dans l'environnement R.

```
str(arbres)
```

```
## 'data.frame':    17635 obs. of  8 variables:
## $ id_bor: Factor w/ 499 levels "0-0","0-100",...: 1 1 1 1 1 1 1 1 1 ...
## $ borx  : int  0 0 0 0 0 0 0 0 0 ...
## $ bory  : int  0 0 0 0 0 0 0 0 0 ...
## $ arbre : int  34501 34502 34502 34503 34504 34506 34507 34509 34510 34511 ...
## $ esp   : Factor w/ 7 levels "abba","acpe",...: 2 2 2 2 2 6 2 2 2 2 ...
## $ multi : Factor w/ 2 levels "FAUX","VRAI": 1 2 2 1 1 1 1 1 1 ...
## $ mort  : Factor w/ 2 levels "FAUX","VRAI": 1 1 1 1 2 1 1 1 1 ...
## $ dhp   : int  82 26 98 73 28 26 29 18 24 34 ...
```

Quelques commandes utiles

summary() permet d'obtenir un résumé du contenu de chaque colonne

```
summary(arbres)
```

```
##      id_bor          borx          bory          arbre
## 80-80 : 227  Min.   : 0.00  Min.   : 0.0  Min.   :    1
## 100-20 : 224  1st Qu.: 60.00  1st Qu.:160.0  1st Qu.: 4358
## 100-60 : 204  Median : 80.00  Median :400.0  Median : 8777
## 80-260 : 196  Mean    : 88.99  Mean    :437.9  Mean    : 9074
## 80-240 : 189  3rd Qu.:120.00  3rd Qu.:700.0  3rd Qu.:13186
## 80-20  : 184  Max.   :180.00  Max.   :980.0  Max.   :64508
## (Other):16411
##      esp        multi        mort          dhp
## abba:2596  FAUX:16920  FAUX:15564  Min.   : 0.0
## acpe:1864  VRAI:  715  VRAI: 2071  1st Qu.:105.0
## acsa:3326                               Median :165.0
## beal:3995                               Mean   :183.7
## bepa:2080                               3rd Qu.:254.0
## fagr:2785                               Max.   :3015.0
## pиру: 989
```

Exercice de manipulation des données

- ✓ Ouvrir le fichier **arbres.csv** au moyen de Excel ou d'un éditeur de texte
- ✓ Calculer l'abondance moyenne pour chaque espèce sur l'ensemble des quadrats
- ✓ Ouvrir le même fichier au moyen de R
- ✓ Vérifier que le fichier a le format approprié
- ✓ Créer un vecteur **AbondanceMoyenne** d'une longueur correspondant au nombre d'espèces dans les données quadrats

Exercice de manipulation des données (suite)

- ✓ Calculer l'abondance moyenne de chaque espèce au moyen de la fonction 'mean' et l'inscrire dans chaque position de **AbondanceMoyenne**
- ✓ Convertir AbondanceMoyenne en **data.frame**
- ✓ Attribuer les noms des espèces pour chaque entrée de **AbondanceMoyenne**
- ✓ Enregistrer **AbondanceMoyenne** sous forme de fichier **CSV** (séparateur point-virgule)
- ✓ Ouvrir le fichier **CSV** et le comparer au calcul fait au moyen de Excel

Note: vous pouvez explorer les données avec **summary** et pour les avancés. Les plus avancés peuvent explorer la commande **apply...**

Complément: le script

Un script est un fichier **.R** contenant une série d'instructions et de commentaires pour réaliser des opérations sur R. Le script est utilisé pour conserver l'historique des opérations et les répéter au besoin.

On peut exécuter un script avec **source()**:

```
rm(list = ls())
source("./MonScript.R")
```

Manipulation des données

Différents outils pour générer des données

```
seq(from = 1, to = 10, by = 0.5)
```

```
## [1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5
## [15] 8.0 8.5 9.0 9.5 10.0
```

```
rep(c(1:3), times = 3)
```

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

Différents outils pour générer des données

```
v1 <- c(1:3)
v2 <- c("A", "B", "C")
expand.grid(v1, v2)
```

```
##   Var1 Var2
## 1    1    A
## 2    2    A
## 3    3    A
## 4    1    B
## 5    2    B
## 6    3    B
## 7    1    C
## 8    2    C
## 9    3    C
```

Combiner des objets

```
rbind(v1,v2)
```

```
##      [,1] [,2] [,3]
## v1 "1"  "2"  "3"
## v2 "A"  "B"  "C"
```

```
cbind(v1,v2)
```

```
##      v1  v2
## [1,] "1" "A"
## [2,] "2" "B"
## [3,] "3" "C"
```

Fonctions utiles: trier des objets

```
tri <- runif(n = 10, min = 0, max = 100)  
tri
```

```
## [1] 48.52697 78.75877 93.96197 89.33506 83.52468 22.82650 55.80610  
## [8] 71.62585 87.45704 34.92941
```

```
sort(tri)
```

```
## [1] 22.82650 34.92941 48.52697 55.80610 71.62585 78.75877 83.52468  
## [8] 87.45704 89.33506 93.96197
```

Fonctions utiles: obtenir des rangs

```
rang <- runif(n = 10, min = 0, max = 100)  
rang
```

```
## [1] 74.8870835 46.4749101 74.4229886 91.1670190 61.6912239 17.0348118  
## [7] 67.8081483 33.1892844 0.6241306 37.1237869
```

```
rank(rang)
```

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

Fonctions utiles: échantillonner les valeurs uniques

```
uq <- c(1,2,5,7,4,3,2,1,10,5,8)  
uq
```

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

```
unique(uq)
```

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

Sous-échantillonner des objets

Parfois, on souhaite avoir seulement une partie des données contenues dans un objet. La fonction **subset()** est fort pratique pour réaliser cette opération.

Ici par exemple, dans l'exemple de Sutton, on souhaite étudier seulement les quadrats possédants des érables à sucre:

```
quadrats <- read.csv2(file=".//donnees/quadrats.csv", header=TRUE, stringsAsFactors=FALSE)
sub_quadrats <- subset(quadrats, quadrats$acsas > 0)
head(sub_quadrats, n=5)
```

```
##      X abba acpe acsa beal bepa fagr piru
## 1   0-0    1   55   11    7    0   92    0
## 2 0-100   0    5    4    3    0    6    0
## 3 0-120   2    7   12    4    1    7    0
## 4 0-140   4    5    4    8    1    2    1
## 5 0-160   2    2   11    8    1    6    1
```

Faire des tableaux sommaires (enfin !)

Très souvent, on souhaite réaliser un sommaire de nos données. La fonction `summary()` est fort utile, mais parfois on souhaite avoir d'autres informations que la moyenne (e.g. la variance). Dans ce cas, la fonction `table()` est recommandée.

Exercice de fin de séance

Le fichier **quadrats.csv** est un sommaire de données individuelles, où la présence de chaque espèce est mesurée. Ces données se trouvent dans **arbres.csv**. Dans le cadre de ce projet, on s'intéresse à la distribution de l'érable à sucre et des autres espèces tout au long du gradient d'élévation de la parcelle. Pour cet exercice, on vous demande de:

1. Charger les données "arbres"
2. Délimiter cinq zones au sein du gradient d'élévation : 0-200m, 201-400m, 401-600m, 601-800m, 801-1000m
3. Pour chacune de ces zones, calculer le nombre de tiges de chaque espèce
4. Enregistrer les résultats dans un tableau avec 5 rangées et 5 colonnes

On vous demande de rédiger un script qui réalisera l'ensemble de ces étapes, de la lecture des données à l'enregistrement du tableau final.