

Manipulation des ensembles de données sous R

Le type data frame

Ricco Rakotomalala

http://eric.univ-lyon2.fr/~ricco/cours/cours_programmation_R.html

Le type **data.frame** est un type spécifique dédié à la manipulation d'ensemble de données de type « individus x variables » (lignes x colonnes).

On peut le voir comme une **liste** de **vecteurs de même longueur** (numérique, factor, etc.). Avec des fonctionnalités spécifiques.

On peut aussi le voir comme une **matrice**. Cette spécificité peut être exploitée intensivement lorsqu'il s'agit d'accéder aux valeurs.

Liste. Un objet essentiel de R.

CRÉATION ET MANIPULATION DES LISTES

Liste = vecteur hétérogène indicé

Une liste R est un vecteur permettant de stocker des objets hétérogènes. L'accès indicé est possible. Les « cases » vides correspondent à la valeur NULL (pointeur nul).

#création d'une liste hétérogène

```
lst <- list("toto",10,TRUE,c(14,15,7),c("toto","titi"))
```

```
print(class(lst))
```

```
print(lst)
```

#accès indicé

```
print(lst[[2]])
```

#nombre d'éléments

```
print(length(lst))
```

```
> #création d'une liste hétérogène
> lst <- list("toto",10,TRUE,c(14,15,7),c("toto","titi"))
> print(class(lst))
[1] "list"
> print(lst)
[[1]]
[1] "toto"

[[2]]
[1] 10

[[3]]
[1] TRUE

[[4]]
[1] 14 15 7

[[5]]
[1] "toto" "titi"

>
> #accès indicé
> print(lst[[2]])
[1] 10
> #nombre d'éléments
> print(length(lst))
[1] 5
>
```

#modification implicite de taille

```
lst[[8]] <- c(11.4,17.3)
```

```
print(lst)
```

Les cases n°6 et 7 ne
« pointent » sur rien

```
> #modification de taille
> lst[[8]] <- c(11.4,17.3)
> print(lst)
[[1]]
[1] "toto"

[[2]]
[1] 10

[[3]]
[1] TRUE

[[4]]
[1] 14 15 7

[[5]]
[1] "toto" "titi"

[[6]]
NULL

[[7]]
NULL

[[8]]
[1] 11.4 17.3
```

Liste = type structuré

```
#nommer chaque champ
usain <- list(nom="Bolt",naiss=1986,records=c(9.58,19.19,36.84))
print(usain)

#accès aux champs avec $
names(usain$records) <- c("100m","200m","4x100m")
print(usain)

#autres exemples
print(usain$naiss) #1986
print(usain$records[2]) #19.19
```

Une liste R peut être vue comme un type structuré (un enregistrement). Chaque élément est nommé.

```
> usain <- list(nom="Bolt",naiss=1986,records=c(9.58,19.19,36.84))
> print(usain)
$nom
[1] "Bolt"

$naiss
[1] 1986

$records
[1] 9.58 19.19 36.84

> #nommer chaque champ
> usain <- list(nom="Bolt",naiss=1986,records=c(9.58,19.19,36.84))
> print(usain)
$nom
[1] "Bolt"

$naiss
[1] 1986

$records
[1] 9.58 19.19 36.84

>
> #accès aux champs
> names(usain$records) <- c("100m","200m","4x100m")
> print(usain)
$nom
[1] "Bolt"

$naiss
[1] 1986

$records
 100m   200m 4x100m
 9.58  19.19 36.84
```

Manipulation d'ensemble de données

LE TYPE DATA FRAME

Data frame est une liste

Data frame

- liste de vecteurs de données
- tous les vecteurs sont de même longueur
- certains sont numériques, d'autres des factor, des étiquettes, etc.
- les vecteurs peuvent être nommés = nom des variables
- les lignes peuvent être nommés = étiquettes des observations
- on peut la manipuler comme une matrice, avec 2 indices [n°ligne , n°colonne]

#création à partir de 3 vecteurs de valeurs

```
v1 <- c(15,8.2,14)
```

```
v2 <- c(TRUE,FALSE,TRUE)
```

```
v3 <- factor(c("M","F","M"))
```

#liste

```
liste <- list(v1,v2,v3)
```

#data.frame

```
donnees <- data.frame(liste)
```

#nommer les colonnes et les lignes

```
colnames(donnees) <- c("x1","x2","x3")
```

```
rownames(donnees) <- c("pierre","paul","jacques")
```

#affichage

```
print(class(donnees))
```

```
print(summary(donnees))
```

```
print(donnees)
```

```
> print(class(donnees))
[1] "data.frame"
> print(summary(donnees))
```

	x1	x2	x3
Min.	: 8.2	Mode :logical	F:1
1st Qu.:	11.1	FALSE:1	M:2
Median :	14.0	TRUE :2	
Mean	:12.4	NA's :0	
3rd Qu.:	14.5		
Max.	:15.0		

```
> print(donnees)
```

	x1	x2	x3
pierre	15.0	TRUE	M
paul	8.2	FALSE	F
jacques	14.0	TRUE	M

#accès 1ere variable

#par nom

donnees\$x1

#accès indicé

#par num.colonne

donnees[1]

#format liste

donnees[[1]]

#format matrice

donnees[,1]

#accès par nom

donnees["x1"]

donnees[["x1"]]

donnees[, "x1"]



```
> #accès à la première variable
> #par nom
> donnees$x1
[1] 15.0  8.2 14.0
> #accès indicé
> #par num.colonne
> donnees[1]
      x1
pierre 15.0
paul    8.2
jacques 14.0
> #format liste
> donnees[[1]]
[1] 15.0  8.2 14.0
> #format matrice
> donnees[,1]
[1] 15.0  8.2 14.0
> #accès par nom
> donnees["x1"]
      x1
pierre 15.0
paul    8.2
jacques 14.0
> donnees[["x1"]]
[1] 15.0  8.2 14.0
> donnees[, "x1"]
[1] 15.0  8.2 14.0
```

Selon le mode d'accès, R affiche les informations de manière différente.

Accès indicé, accès par nom, combinaisons.

#accès individus x variables

#1er individu

donnees[1,]

#1er et 3e individus

donnees[c(1,3),]

#accès par nom

donnees["paul",]

#indiv. (1,3) et var. (1,2)

donnees[c(1,3),1:2]

#accès par nom ind (1,3) x var (1,3)

donnees[c("pierre","jacques"),c("x1","x3")]

```
> #accès individus x variables
> #1er individu
> donnees[1,]
      x1  x2 x3
pierre 15 TRUE M
> #1er et 3e individus
> donnees[c(1,3),]
      x1  x2 x3
pierre 15 TRUE M
jacques 14 TRUE M
> #accès par nom
> donnees["paul",]
      x1  x2 x3
paul 8.2 FALSE F
> #indiv. (1,3) et var. (1,2)
> donnees[c(1,3),1:2]
      x1  x2
pierre 15 TRUE
jacques 14 TRUE
> donnees[c("pierre","jacques"),c("x1","x3")]
      x1 x3
pierre 15 M
jacques 14 M
```

```
> print(donnees)
```

	x1	x2	x3
pierre	15.0	TRUE	M
paul	8.2	FALSE	F
jacques	14.0	TRUE	M


#restrictions

```
donnees[donnees$x1>10,]  
donnees[donnees$x3=="M",]  
donnees[donnees$x1>=15 | donnees$x3=="F",]  
donnees[donnees$x1>10,c(1,3)]
```

#condition = vecteur de booléens

```
b <- (donnees$x1>=15 | donnees$x3=="F")  
print(b)
```

Les conditions génèrent en réalité un vecteur de booléens qui permet définir les lignes à afficher.



```
> #restrictions  
> donnees[donnees$x1>10,]  
      x1  x2 x3  
pierre 15 TRUE M  
jacques 14 TRUE M  
> donnees[donnees$x3=="M",]  
      x1  x2 x3  
pierre 15 TRUE M  
jacques 14 TRUE M  
> donnees[donnees$x1>=15 | donnees$x3=="F",]  
      x1  x2 x3  
pierre 15.0 TRUE M  
paul    8.2 FALSE F  
> donnees[donnees$x1>10,c(1,3)]  
      x1 x3  
pierre 15 M  
jacques 14 M  
>  
> b <- (donnees$x1>=15 | donnees$x3=="F")  
> print(b)  
[1] TRUE TRUE FALSE
```

Lecture des fichiers aux format texte et Excel (xls, xlsx)

IMPORTATION D'UN FICHIER DE DONNÉES

Importation d'un fichier texte dans un data frame

demo_reglog.txt - Bloc-notes

Fichier	Edition	Format	Affichage ?
age	taux	engine	coeur
50	126.0	1	presence
49	126.0	0	presence
46	144.0	0	presence
49	139.0	0	presence
62	154.0	1	presence
35	156.0	1	presence
67	160.0	0	absence
65	140.0	0	absence
47	143.0	0	absence
58	165.0	0	absence
57	115.0	1	absence
59	145.0	0	absence
44	175.0	0	absence
41	153.0	0	absence
54	152.0	0	absence
52	169.0	0	absence
57	168.0	1	absence
50	158.0	0	absence
44	170.0	0	absence
49	171.0	0	absence

Nom du data.frame

Séparateur de colonnes

1^{ère} ligne = nom des variables

Nom du fichier

Point décimal

```
R Console
> heart <- read.table(file="demo_reglog.txt", sep="\t", dec=".", header=T)
> class(heart)
[1] "data.frame"
> summary(heart)
```

age	taux	engine	coeur
Min. :35.00	Min. :115.0	Min. :0.00	absence :14
1st Qu.:46.75	1st Qu.:142.2	1st Qu.:0.00	presence: 6
Median :50.00	Median :153.5	Median :0.00	
Mean :51.75	Mean :151.4	Mean :0.25	
3rd Qu.:57.25	3rd Qu.:165.8	3rd Qu.:0.25	
Max. :67.00	Max. :175.0	Max. :1.00	

« âge », « taux » et « engine » sont considérées comme quantitatives.

« cœur » est une variable qualitative.

Fichier texte, séparateur tabulation.

summary() fournit une vision globale des données.

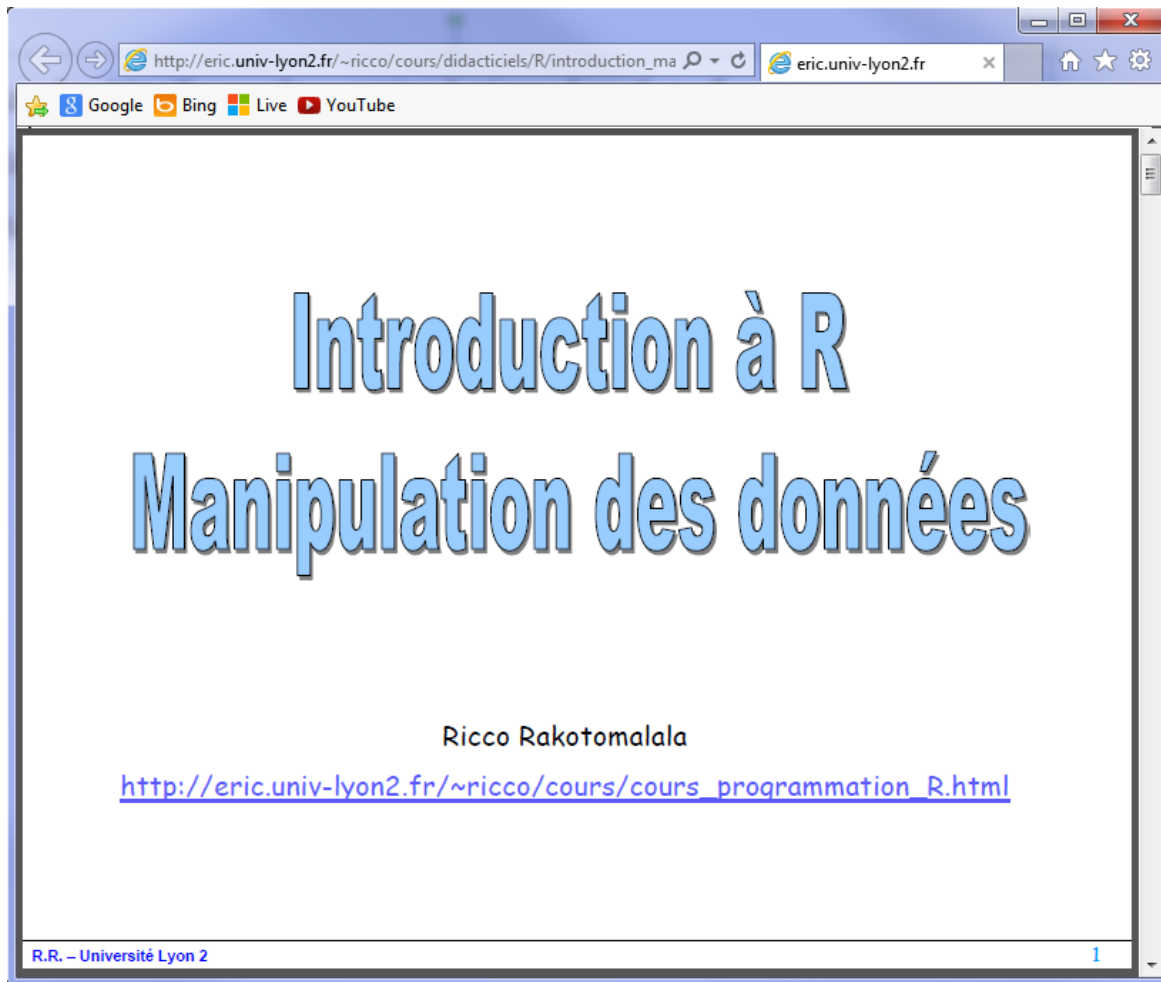
Les variables sont typées. Les plus utilisées sont « numeric / integer » (variables quantitatives) et « factor » (variables qualitatives)

Remarquer toujours le rôle de \$

Ne pas oublier de faire **setwd()** pour changer le répertoire courant. Ou bien spécifier le chemin d'accès complet dans le paramètre « file ».

```
R Console
> class(heart$age)
[1] "integer"
> class(heart$taux)
[1] "numeric"
> class(heart$engine)
[1] "integer"
> class(heart$coeur)
[1] "factor"
> mean(heart$age)
[1] 51.75
> |
```

Pour plus de détails sur la manipulation des data frame (accès, requêtes, calculs sur les variables, croisement, graphiques élémentaires, etc.), et notamment l'importation des **fichiers Excel (xls,xlsx)** via des packages spécifiques, voir « [Introduction à R – Manipulation des données](#) ».



`read.table()`
`write.table()`

`read.xlsx()`
`write.xlsx()`

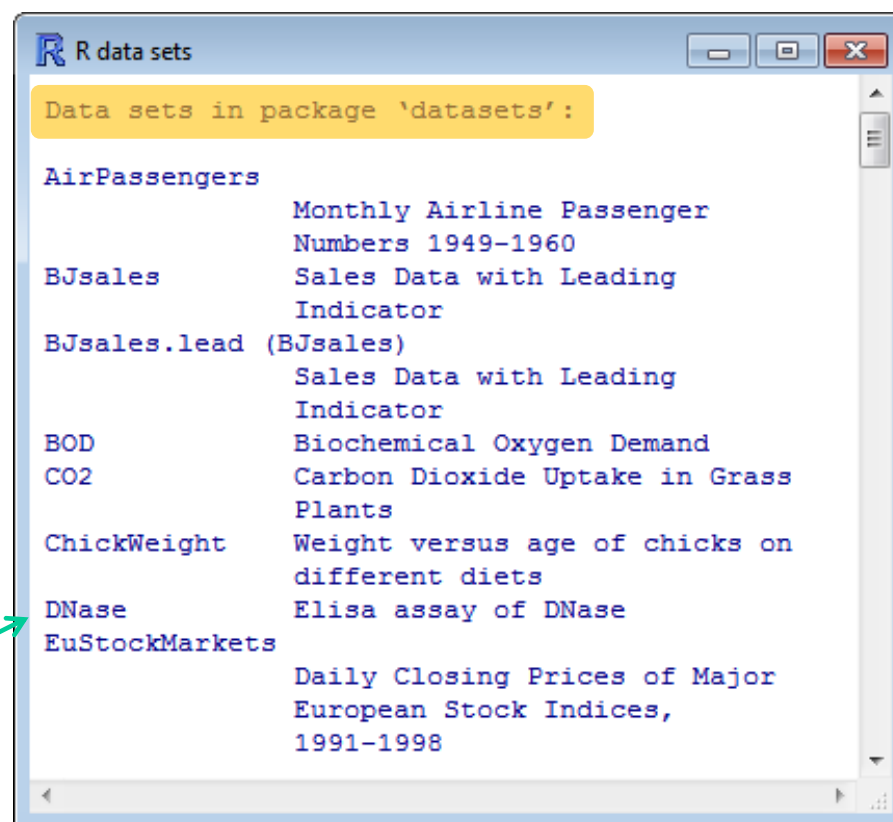
`nrow()`
`ncol()`

Des données peuvent être intégrées dans des packages (pour les démonstrations, les exemples d'utilisation, etc.).

Au démarrage de R, un certain nombre de packages « de base » sont automatiquement chargés, entres autres le package « **datasets** ».

De fait, plusieurs jeux de données sont directement accessibles dans R. On peut en obtenir la liste avec la commande **data()**

```
> which(installed.packages()[, "Priority"] == "base")
      base  compiler  datasets  graphics  grDevices
      58      63      64      66      67
      grid  methods  parallel  splines    stats
      68      73      77      80      81
      stats4  tcltk    tools      utils
      82      84      85      87
```



Exemple : les données « mtcars »

```
#mtcars
data(mtcars)
#les 5 premières lignes
print(head(mtcars,n=5))
#stat.desc.
print(summary(mtcars))
```

```
> #mtcars
> data(mtcars)
> #les 5 premières lignes
> print(head(mtcars,n=5))
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360	175	3.15	3.440	17.02	0	0	3	2

```
> #stat.desc.
> print(summary(mtcars))
```

mpg		cyl		disp		hp		drat		wt	
Min.	:10.40	Min.	:4.000	Min.	: 71.1	Min.	: 52.0	Min.	:2.760	Min.	:1.513
1st Qu.:	15.43	1st Qu.:	4.000	1st Qu.:	120.8	1st Qu.:	96.5	1st Qu.:	3.080	1st Qu.:	2.581
Median	:19.20	Median	:6.000	Median	:196.3	Median	:123.0	Median	:3.695	Median	:3.325
Mean	:20.09	Mean	:6.188	Mean	:230.7	Mean	:146.7	Mean	:3.597	Mean	:3.217
3rd Qu.:	22.80	3rd Qu.:	8.000	3rd Qu.:	326.0	3rd Qu.:	180.0	3rd Qu.:	3.920	3rd Qu.:	3.610
Max.	:33.90	Max.	:8.000	Max.	:472.0	Max.	:335.0	Max.	:4.930	Max.	:5.424

qsec		vs		am		gear		carb	
Min.	:14.50	Min.	:0.0000	Min.	:0.0000	Min.	:3.000	Min.	:1.000
1st Qu.:	16.89	1st Qu.:	0.0000	1st Qu.:	0.0000	1st Qu.:	3.000	1st Qu.:	2.000
Median	:17.71	Median	:0.0000	Median	:0.0000	Median	:4.000	Median	:2.000
Mean	:17.85	Mean	:0.4375	Mean	:0.4062	Mean	:3.688	Mean	:2.812
3rd Qu.:	18.90	3rd Qu.:	1.0000	3rd Qu.:	1.0000	3rd Qu.:	4.000	3rd Qu.:	4.000
Max.	:22.90	Max.	:1.0000	Max.	:1.0000	Max.	:5.000	Max.	:8.000

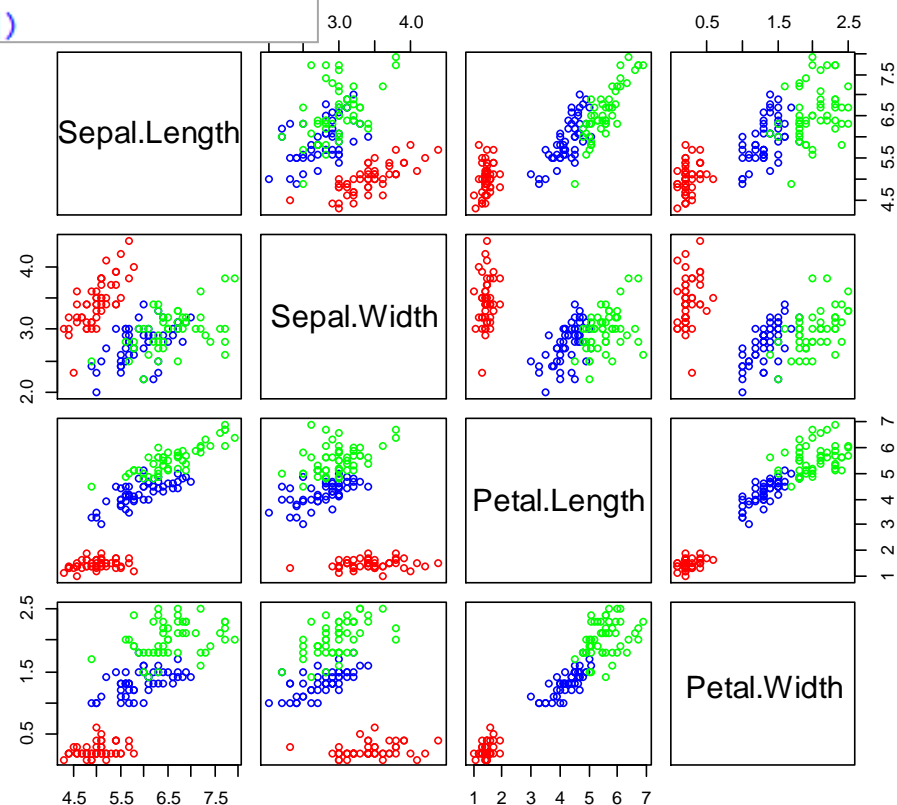
Exemple : les données « iris »

```
> #iris
> data(iris)
> print(head(iris,n=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

> print(summary(iris))
  Sepal.Length      Sepal.Width      Petal.Length      Petal.Width      Species
Min.   :4.300    Min.   :2.000    Min.   :1.000    Min.   :0.100    setosa   :50
1st Qu.:5.100    1st Qu.:2.800    1st Qu.:1.600    1st Qu.:0.300    versicolor:50
Median :5.800    Median :3.000    Median :4.350    Median :1.300    virginica :50
Mean   :5.843    Mean   :3.057    Mean   :3.758    Mean   :1.199
3rd Qu.:6.400    3rd Qu.:3.300    3rd Qu.:5.100    3rd Qu.:1.800
Max.   :7.900    Max.   :4.400    Max.   :6.900    Max.   :2.500

> pairs(iris[1:4],col=c("red","blue","green")[iris$Species])
```

On commence à
percevoir les capacités
graphiques de R.




Effectuer des calculs à partir d'un ensemble de données

PROGRAMMER AVEC LES DATA FRAME

Les calculs usuels définis pour les vecteurs s'appliquent bien évidemment aux vecteurs de data frame : univariés (moyenne...), bivariés (corrélation, moyennes conditionnelles, tableau de contingence...), multivariés (matrice de corrélations...)...

```
#moyenne
print(mean(mtcars$mpg))
#corrélation
print(cor(mtcars[1:4]))
#moyennes conditionnelles
print(tapply(iris$Petal.Width,iris$Species,mean))
#tableau croisé
print(table(iris$Species,iris$Species))
```



```
> print(mean(mtcars$mpg))
[1] 20.09062
> #corrélation
> print(cor(mtcars[1:4]))
```

	mpg	cyl	disp	hp
mpg	1.0000000	-0.8521620	-0.8475514	-0.7761684
cyl	-0.8521620	1.0000000	0.9020329	0.8324475
disp	-0.8475514	0.9020329	1.0000000	0.7909486
hp	-0.7761684	0.8324475	0.7909486	1.0000000

```
> #moyennes conditionnelles
> print(tapply(iris$Petal.Width,iris$Species,mean))
```

	setosa	versicolor	virginica
	0.246	1.326	2.026

```
> #tableau croisé
> print(table(iris$Species,iris$Species))
```

	setosa	versicolor	virginica
setosa	50	0	0
versicolor	0	50	0
virginica	0	0	50

sapply() permet d'appliquer une fonction sur chaque colonne d'un data frame, chaque fonction renvoie une valeur (un vecteur), le résultat est stockée dans un vecteur (une matrice)

```
#mediane par variable  
v <- sapply(mtcars, median)  
print(v)
```

Utilisation d'une
fonction prédéfinie



```
> print(v)  
      mpg      cyl      disp      hp      drat      wt      qsec      vs      am      gear      carb  
19.200   6.000 196.300 123.000   3.695   3.325 17.710   0.000   0.000   4.000   2.000
```

```
#fonction "etendue"  
etendue <- function(x){  
  e <- max(x) - min(x)  
  return(e)  
}  
#appel sur chaque variable  
v <- sapply(mtcars, etendue)  
print(v)
```

Utilisation d'une
fonction créée par
l'utilisateur



```
> print(v)  
      mpg      cyl      disp      hp      drat      wt      qsec      vs      am      gear      carb  
23.500   4.000 400.900 283.000   2.170   3.911   8.400   1.000   1.000   2.000   7.000
```

supply() (suite)

#fonction "etendue" avec ratio

```
etendue.bis <- function(x,ratio){  
  e <- ifelse(ratio>0,(max(x) - min(x))/ratio,NA)  
  return(e)  
}
```

#appel sur chaque variable

```
v <- supply(mtcars,etendue.bis,ratio=5)  
print(v)
```

Utilisation d'une
fonction paramétrée

```
> print(v)  
      mpg      cyl    disp      hp      drat      wt  
4.7000  0.8000 80.1800 56.6000  0.4340  0.7822  
      qsec      vs      am      gear      carb  
1.6800  0.2000  0.2000  0.4000  1.4000
```

#bornes des quantiles

```
bornes <- function(x,p.inf,p.sup){  
  #renvoie un vecteur de 2 valeurs  
  return(quantile(x,probs=c(p.inf,p.sup)))  
}
```

#appel : intervalle inter-quartile

```
m <- supply(mtcars,bornes,p.inf=0.25,p.sup=0.75)  
print(class(m))  
print(m)
```

Fonction renvoyant
un vecteur

```
> print(class(m))  
[1] "matrix"  
> print(m)  
      mpg cyl  disp  hp drat    wt  qsec vs am gear carb  
25% 15.425  4 120.825 96.5 3.08 2.58125 16.8925 0 0  3  2  
75% 22.800  8 326.000 180.0 3.92 3.61000 18.9000 1 1  4  4
```

```
#division des valeurs par ratio
#uniquement si numérique
divise <- function(x,ratio){
  if (is.factor(x)==TRUE){
    return(x)
  } else
  {
    return(ifelse(ratio>0,x/ratio,NA))
  }
}
```

```
#appel sur iris
lst <- lapply(iris,divise,ratio=2)
print(class(lst))
```

```
#transformation en data frame
iris.bis <- data.frame(lst)
print(summary(iris.bis))
```

```
#on aurait pu faire directement
iris.ter <- data.frame(lapply(iris,divise,ratio=2))
```

lapply(), même démarche que **sapply()** sauf qu'elle renvoie une liste, qu'on peut transtyper en data frame (si nécessaire et si c'est possible)

```
> lst <- lapply(iris,divise,ratio=2)
> print(class(lst))
[1] "list"
> #transformation en data frame
> iris.bis <- data.frame(lst)
> print(summary(iris.bis))
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
Min. :2.55	Min. :1.75	Min. :0.7	Min. :0.1	setosa :50
1st Qu.:2.55	1st Qu.:1.75	1st Qu.:0.7	1st Qu.:0.1	versicolor:50
Median :2.55	Median :1.75	Median :0.7	Median :0.1	virginica :50
Mean :2.55	Mean :1.75	Mean :0.7	Mean :0.1	
3rd Qu.:2.55	3rd Qu.:1.75	3rd Qu.:0.7	3rd Qu.:0.1	
Max. :2.55	Max. :1.75	Max. :0.7	Max. :0.1	

#génération des données

#n lignes et K colonnes

```
set.seed(1)
```

```
n <- 100
```

```
K <- 100000
```

```
base.test <- data.frame(lapply(1:K,function(i){runif(n)}))
```





```
colnames(base.test) <- paste("v",1:K,sep="")
```


Génération d'une base avec $n = 100$
individus x $K = 100.000$ variables.




Workspace

History

 Import Dataset ▾



Data

base.test100 obs. of 100000 variables

Values

K1e+05

n100

Functions

calc.boucle(dataset)

calc.sapply(dataset)

```
#calculer les moyennes
#en utilisant une boucle
calc.boucle <- function(dataset){
  p <- ncol(dataset)
  m <- double(p)
  for (k in 1:p){
    m[k] <- mean(dataset[,k])
  }
  return(mean(m))
}
system.time(print(calc.boucle(base.test)))
```

Temps CPU calculs

```
> system.time(print(calc.boucle(base.test)))
[1] 0.4999733
utilisateur      système      écoulé
      53.58         1.59        56.84
```

```
#calc moyenne
#en utilisant sapply()
calc.sapply <- function(dataset){
  m <- sapply(dataset,mean)
  return(mean(m))
}
system.time(print(calc.sapply(base.test)))
```

```
> system.time(print(calc.sapply(base.test)))
[1] 0.4999733
utilisateur      système      écoulé
      1.94         0.00        1.93
```


De la documentation à profusion (n'achetez jamais des livres sur R)

Site du cours

http://eric.univ-lyon2.fr/~ricco/cours/cours_programmation_R.html

Programmation R

<http://www.duclert.org/>

Quick-R

<http://www.statmethods.net/>

POLLS (Kdnuggets)

Data Mining / Analytics Tools Used

(R, 2nd ou 1^{er} depuis 2010)

What languages you used for data mining / data analysis?

<http://www.kdnuggets.com/polls/2013/languages-analytics-data-mining-data-science.html>

(Août 2013, langage R en 1^{ère} position)

Article New York Times (Janvier 2009)

“Data Analysts Captivated by R’s Power” - http://www.nytimes.com/2009/01/07/technology/business-computing/07program.html?_r=1