TELECOM ParisTech

# Introduction à R

Mai/Juin 2011

# Plan

## Manipulations élémentaires

- Vecteurs et tableaux de données
- Graphisme

## Structures de contrôle

- Exécution conditionnelle
- Boucles
- Fonctions

## Algèbre

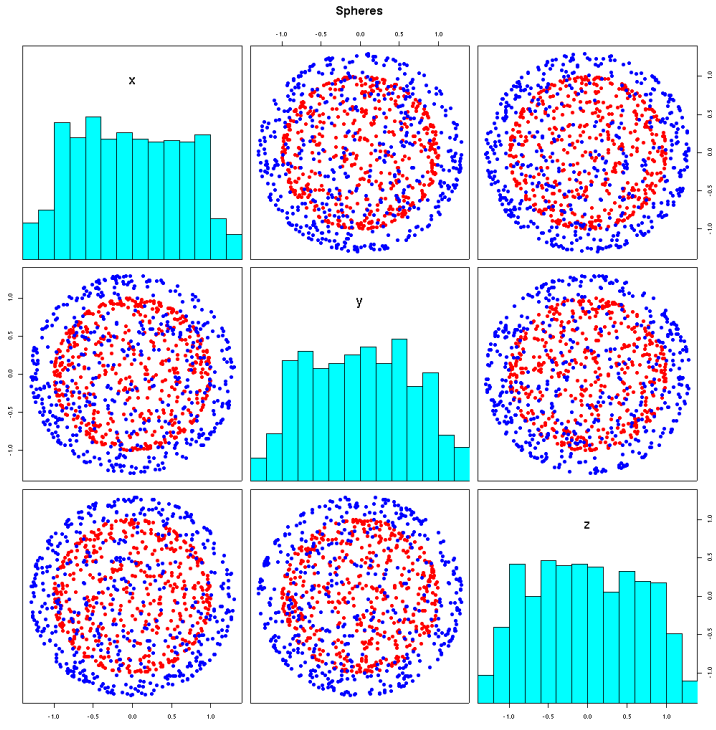
- Vecteurs
- Matrices

# Plan

# Logiciel R

http://www.r-project.org/

- logiciel de statistique et de visualisation
- langage de programmation (proche de S)
- logiciel libre multi-plateformes (Linux, MacOS X, Windows)
- extensible par *packages*



# Principes

- console interactive (lancée avec la commande `R`)
- manipulation d'**objets** :
  - vecteur
  - matrices
  - tableaux de données
  - listes
  - etc.
- liste des objets : `ls()`
- exécution d'un ensemble de commande : `source("script.R")`
- sauvegarde des affichages : `sink("sauvegarde.Rout")`
- aide : `?truc` affiche l'aide sur `truc`

# ?ls

```
ls                      package:base                      R Documentation

List Objects

Description:

     'ls' and 'objects' return a vector of character strings giving
     the names of the objects in the specified environment. When
     invoked with no argument at the top level prompt, 'ls' shows
     what data sets and functions a user has defined. When invoked
     with no argument inside a function, 'ls' returns the names of
     the functions local variables. This is useful in conjunction
     with 'browser'.

Usage:

     ls(name, pos = -1, envir = as.environment(pos),
        all.names = FALSE, pattern)
     objects(name, pos= -1, envir = as.environment(pos),
             all.names = FALSE, pattern)
```

# Vecteur

- objet le plus simple

```
>x=c(1,2)
>ls()
[1] "x"
> x
[1] 1 2
> class(x)
[1] "numeric"
```

- = : affectation (aussi <- qui est plus lisible)
- c : concaténation
- modifications

```
>x[1] <- 3
x
[1] 3 2
```

# Tableau de données

- *Data Frame*

```
>data(iris)
>ls()
[1] "iris"
>class(iris)
[1] "data.frame"
```

- dimensions

```
>dim(iris)
[1] 150   5
```

150 observations pour 5 variables

- chaque colonne (variable) est nommée

```
> names(iris)
[1] "Sepal.Length" "Sepal.Width" "Petal.Length"
[4] "Petal.Width"  "Species"
```

# Tableau de données

- accès au contenu

```
> class(iris$Sepal.Length)
[1] "numeric"
> length(iris$Sepal.Length)
[1] 150
> iris$Sepal.Length[10]
[1] 4.9
```

valeur de la variable *Sepal Length* pour la dixième observation

- une observation

```
> iris[10,]
   Sepal.Length Sepal.Width Petal.Length    Petal.Width Species
10          4.9         3.1          1.5            0.1  setosa
```

ou une valeur

```
> iris[10,3]
[1] 1.5
```

# Variables nominales

- *Factor*

```
> class(iris$Species)
[1] "factor"
> levels(iris$Species)
[1] "setosa"     "versicolor" "virginica"
> class(levels(iris$Species))
[1] "character"
```

  variable nominale à trois modalités

- une valeur

```
> iris$Species[2]
[1] setosa
Levels: setosa versicolor virginica
```

- distribution des modalités

```
> summary(iris$Species)
    setosa versicolor  virginica
        50         50         50
```

- moyenne `mean()` et médiane `median()`

```
> mean(iris$Sepal.Width)
[1] 3.057333
> median(iris$Petal.Length)
[1] 4.35
```

- plusieurs variables simultanément

```
> mean(iris[1:3])
Sepal.Length  Sepal.Width Petal.Length
    5.843333     3.057333     3.758000
```

- remarque : `a:b` : tous les entiers entre `a` et `b`

```
>-3:6
 [1] -3 -2 -1  0  1  2  3  4  5  6
```

- écart type `sd()` et variance `var()`

```
> sd(iris$Petal.Width)
[1] 0.7622377
```

# Calcul vectoriel

- opérateurs « terme à terme »

```
> x <- c(3,4)
> y <- c(-2,5)
> x+y
[1] 1 9
> x*y
[1] -6 20
```

- réutilisation

```
>x-1
[1] 2 3
> z <- 1:3
> z
[1] 1 2 3
> z-x
[1] -2 -2  0
Warning message:
la longueur de l'objet le plus long  n'est pas un multiple de
   la longueur de l'objet le plus court in: z - x
```

# Calcul vectoriel et tableaux de données

- sous-tableau

```
> class(iris[1])
[1] "data.frame"
```

- colonne

```
> class(iris[[1]])
[1] "numeric"
```

- application : calcul de l'écart type

```
> sqrt(sum((iris[[1]]-mean(iris[[1]]))^2)/(length(iris[[1]])-1))
[1] 0.8280661
> sd(iris[[1]])
[1] 0.8280661
```

- corrélation

```
> cor(iris[1:2])
             Sepal.Length Sepal.Width
Sepal.Length    1.0000000  -0.1175698
Sepal.Width    -0.1175698   1.0000000
```

- exercice : calcul de la corrélation

# Statistiques

## Résumé

- résumé d'un tableau de donnée

```
> summary(iris)
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
 Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
 Median :5.800   Median :3.000   Median :4.350   Median :1.300
 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
       Species
 setosa    :50
 versicolor:50
 virginica :50
```

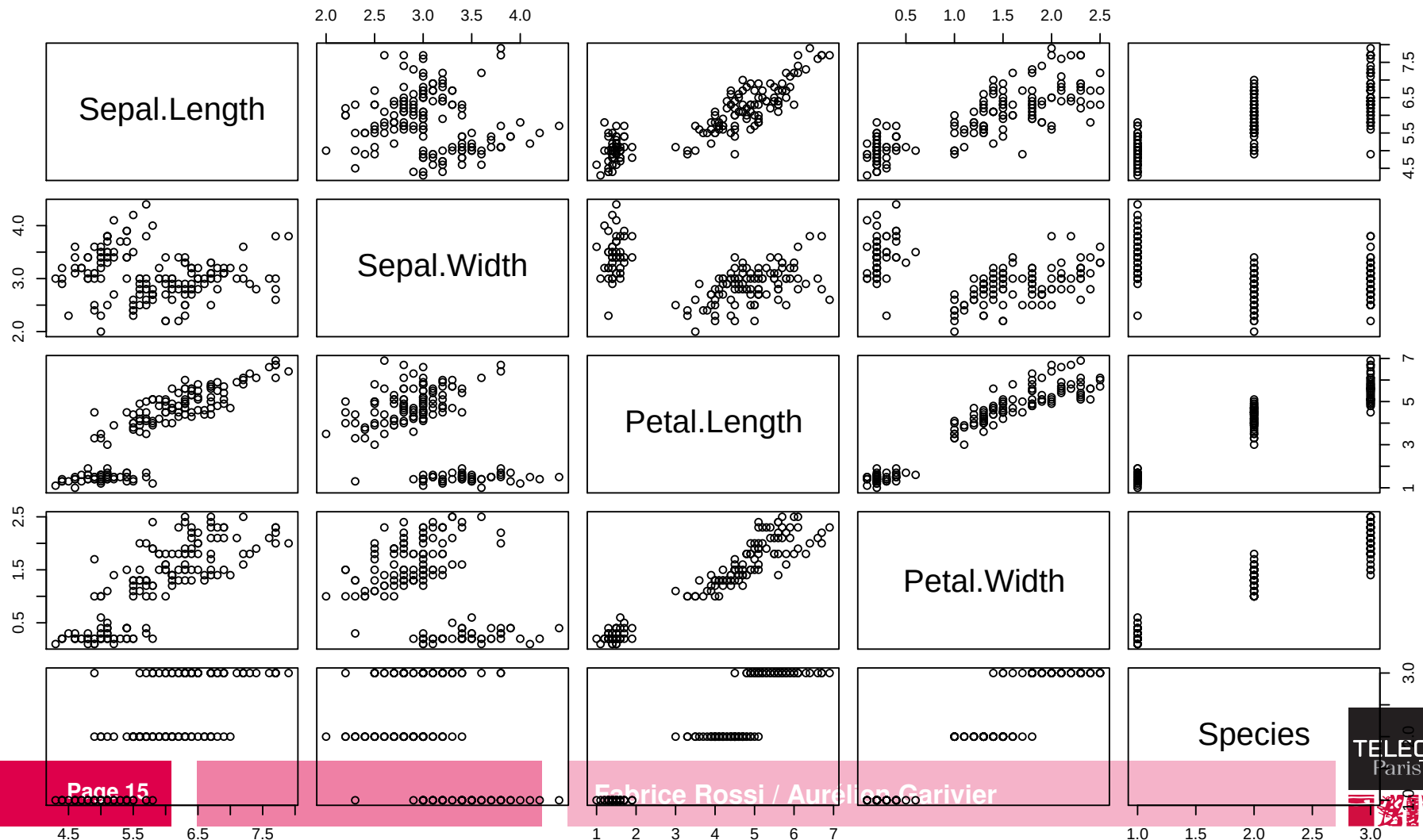
- ou d'une seule variable

```
> summary(iris$Petal.Length)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   1.600   4.350   3.758   5.100   6.900
```

# Diagramme de dispersion

## Affichage d'un tableau de données

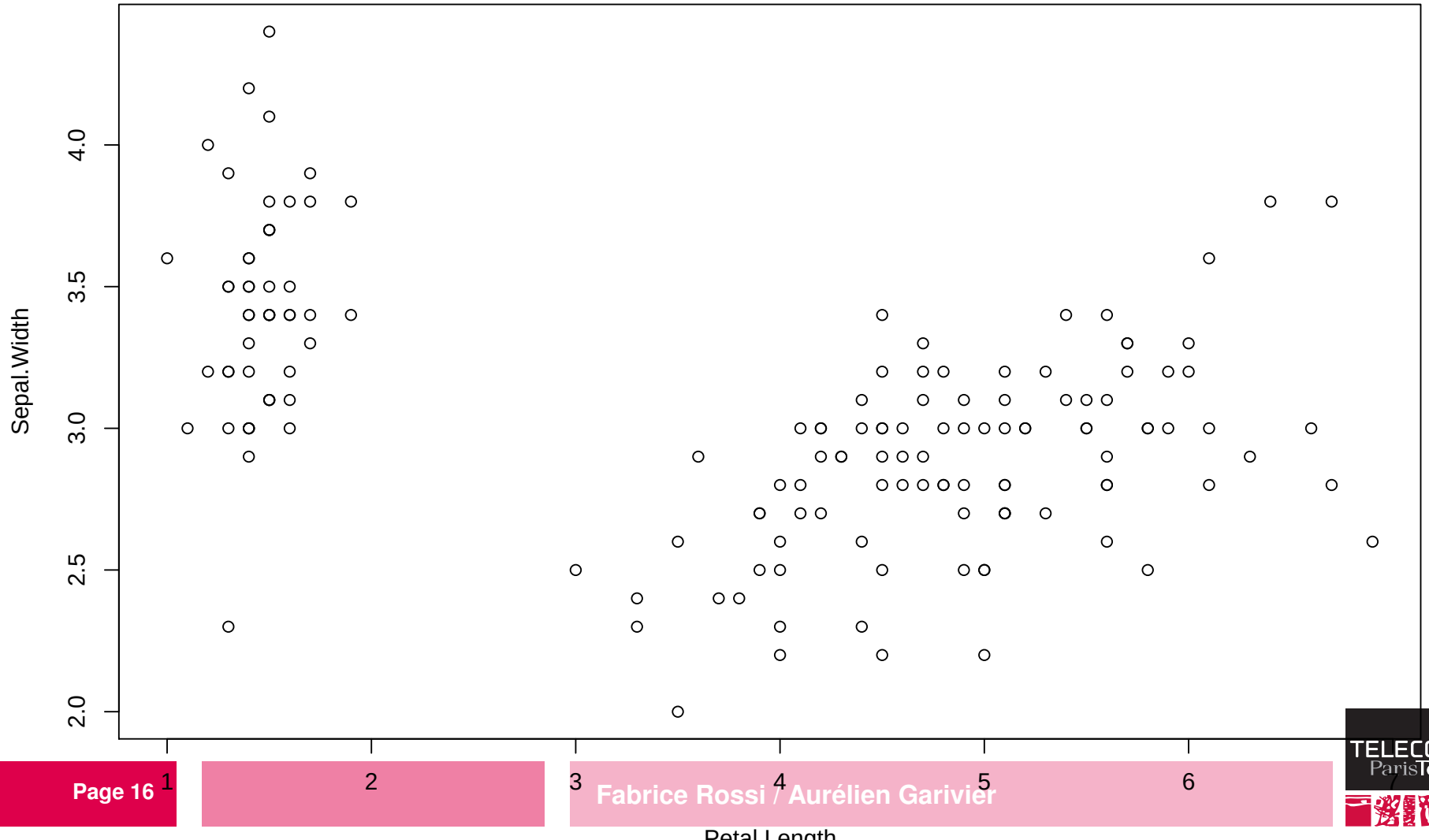
```
>plot(iris)
```



# Diagramme de dispersion

## Deux variables seulement

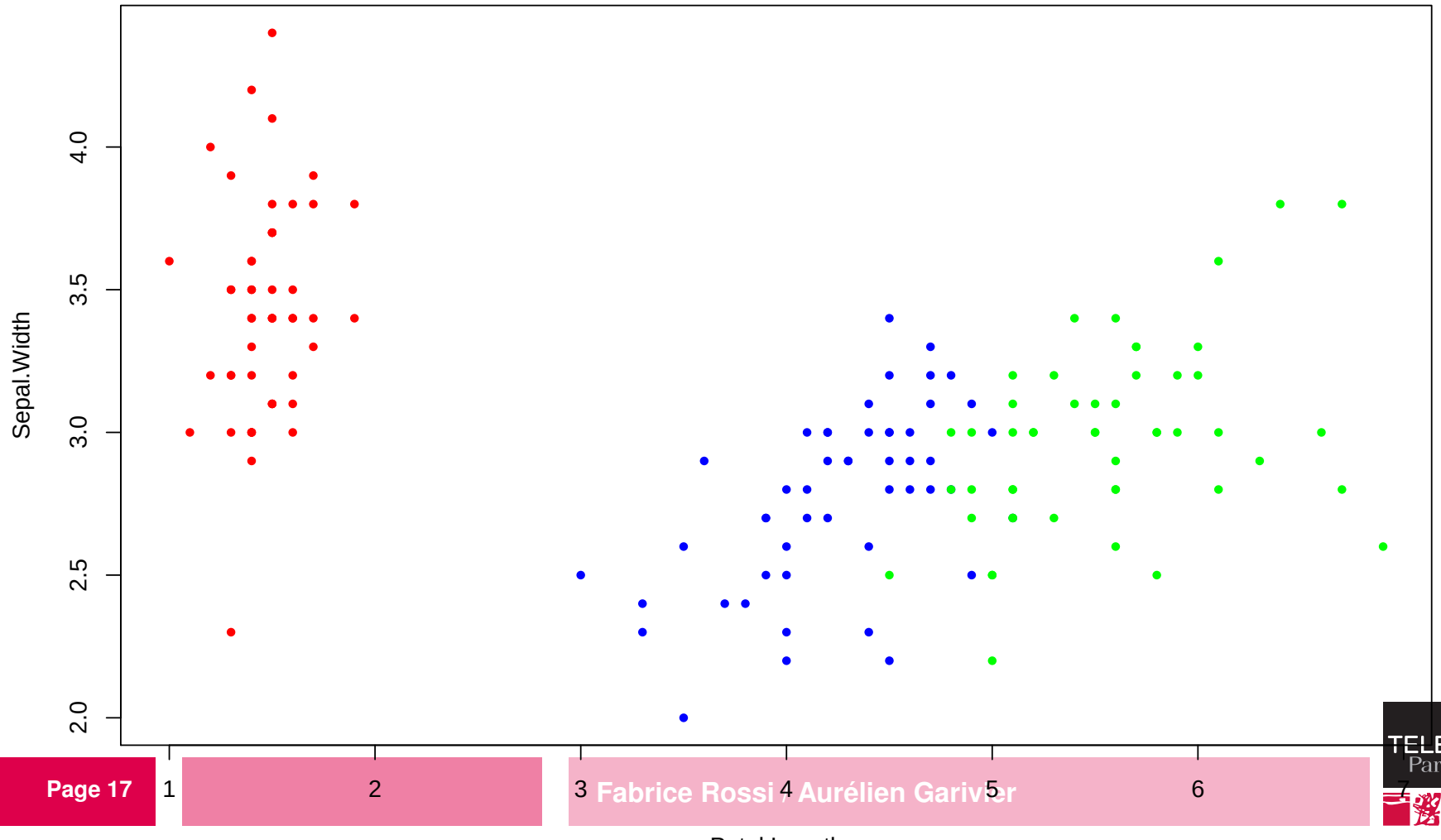
```
>plot(iris[c(3,2)])
```



# Diagramme de dispersion

## Un peu de couleur

```
>plot(iris[c(3,2)],pch=20,
      col=c("red","blue", "green")[unclass(iris$Species)])
```



# Sauvegarde d'un graphique

- R permet très simplement la sauvegarde de ses sorties graphiques pour qu'ils soient intégrables dans vos documents / présentations
- pour sauvegarder le graphe créé par la commande `plot(...)` dans le fichier "monFichier.pdf", il suffit de faire :

```
> pdf("monFichier.pdf")
> plot(...)
> dev.off();
```

- d'autres formats de sauvegarde sont également disponibles : png, xfig, etc... : cf `help(Devices)`

# Indexation

- `x[y]` : contenu de `x` réordonné selon `y`

```
> x <- 10:15
> x
[1] 10 11 12 13 14 15
> y <- c(1,3,2)
> y
[1] 1 3 2
> x[y]
[1] 10 12 11
> x[c(4,4)]
[1] 13 13
```

- `x[-y]` : éléments de `x` dont les indices ne sont pas dans `y`

```
> x[-y]
[1] 13 14 15
```

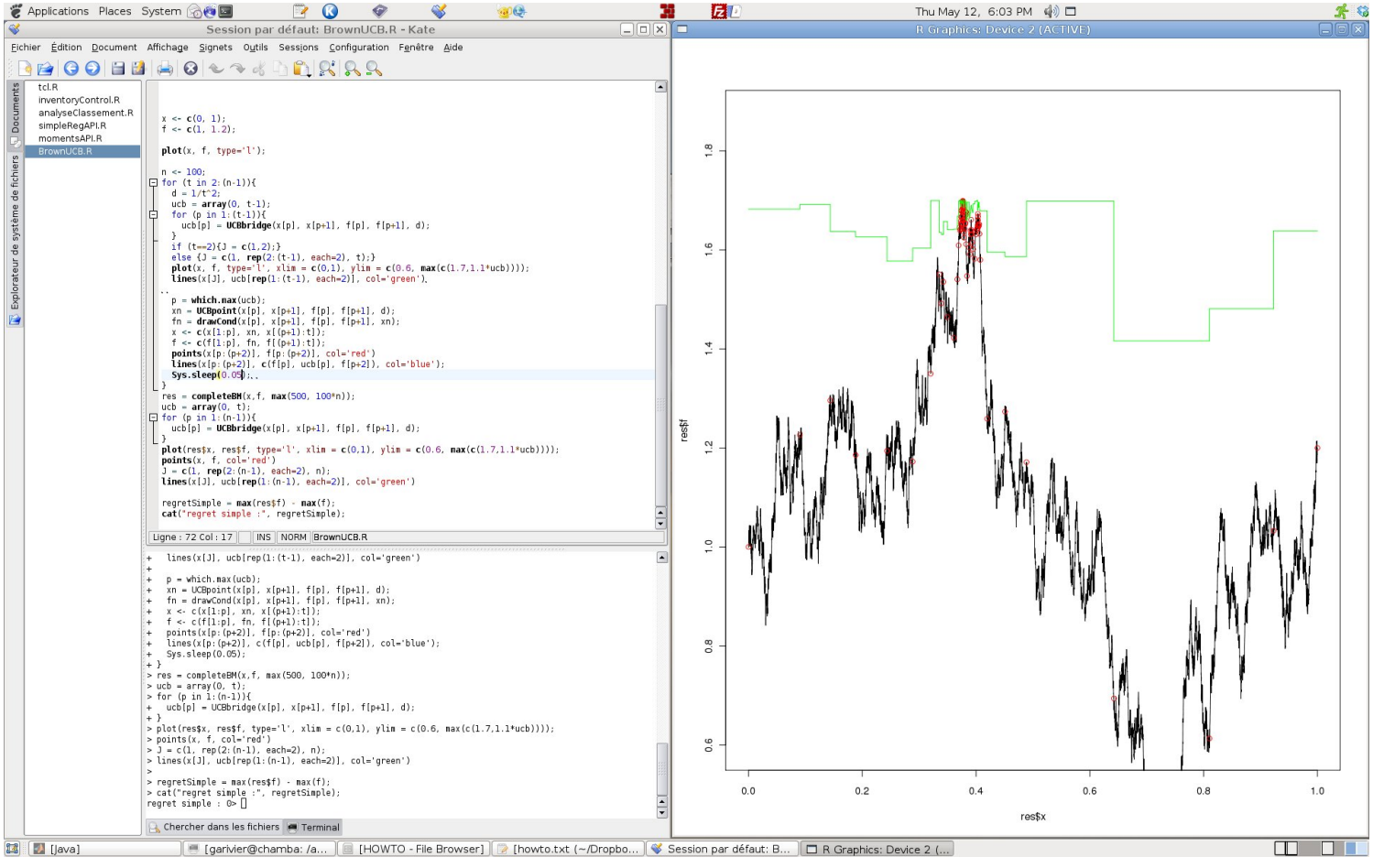
- `x[t]` : éléments de `x` pour lesquels `t` est `TRUE`

```
> x>12
[1] FALSE FALSE FALSE  TRUE  TRUE  TRUE
> x[x>12]
[1] 13 14 15
```

# Exercices

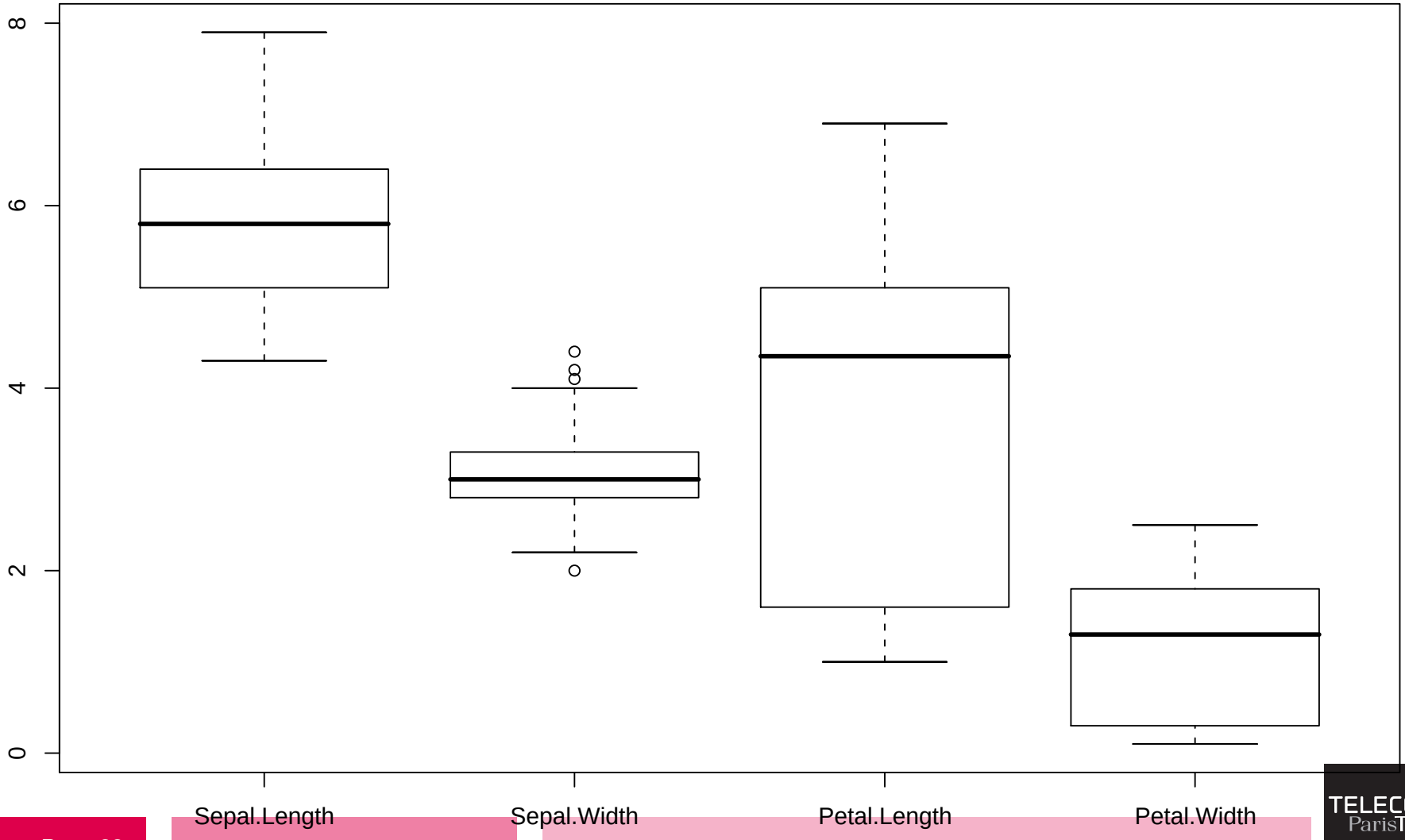
- afficher seulement les classes `"setosa"` et `"virginica"`
- fusionner les classes `"versicolor"` et `"virginica"` dans l'affichage en couleur
- afficher les points dont une coordonnée est à plus d'un écart-type de la moyenne correspondante
- éléments utiles :
  - opérateurs logiques : `|` ou, `&` et
  - comparaison à une valeur nominale : `x=="setosa"` ou `x !="setosa"`

# Utilisation d'un éditeur



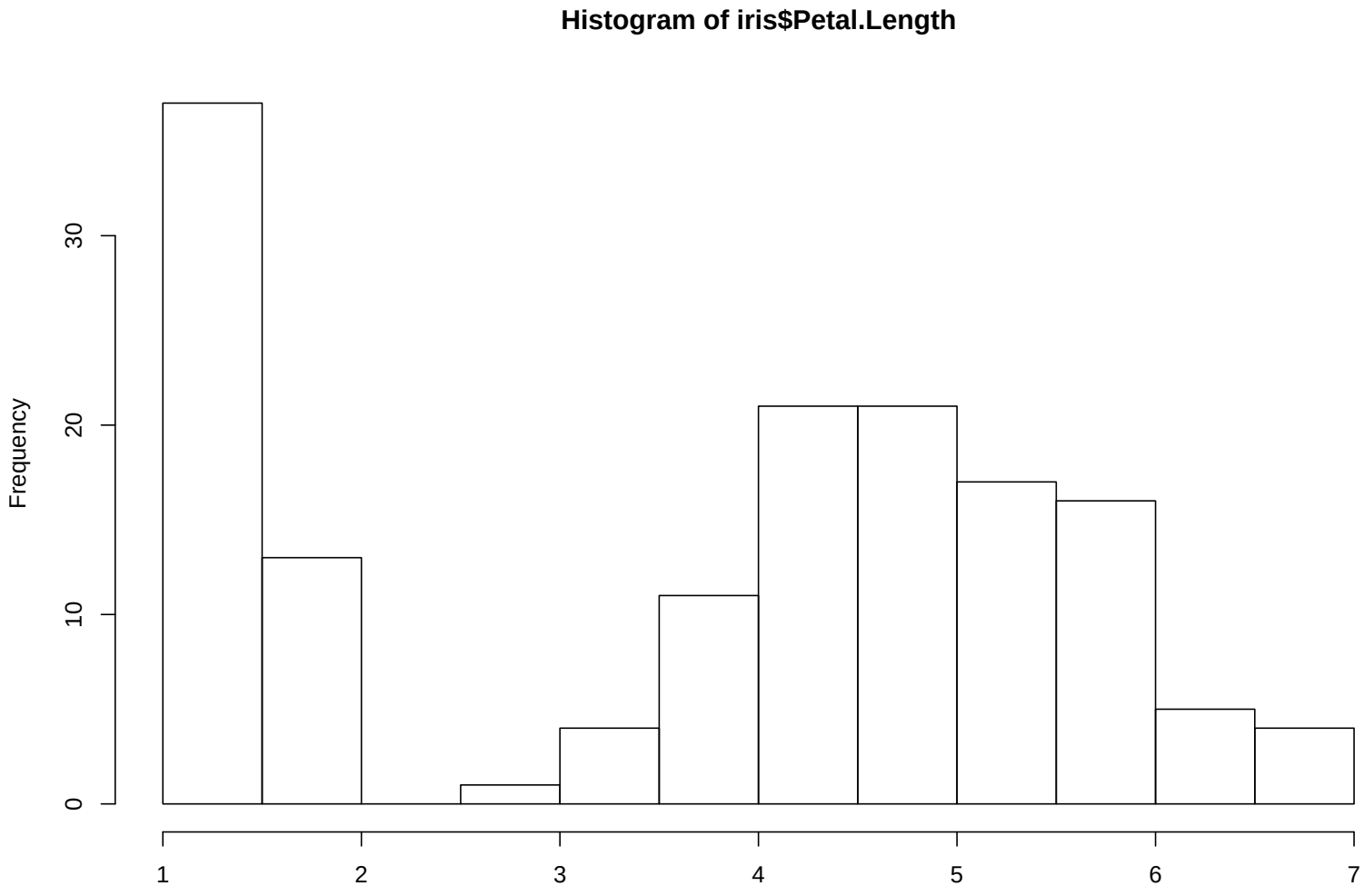
# Boîte à moustaches

```
>boxplot(iris[1:4])
```



# Histogramme

```
>hist(iris$Petal.Length)
```



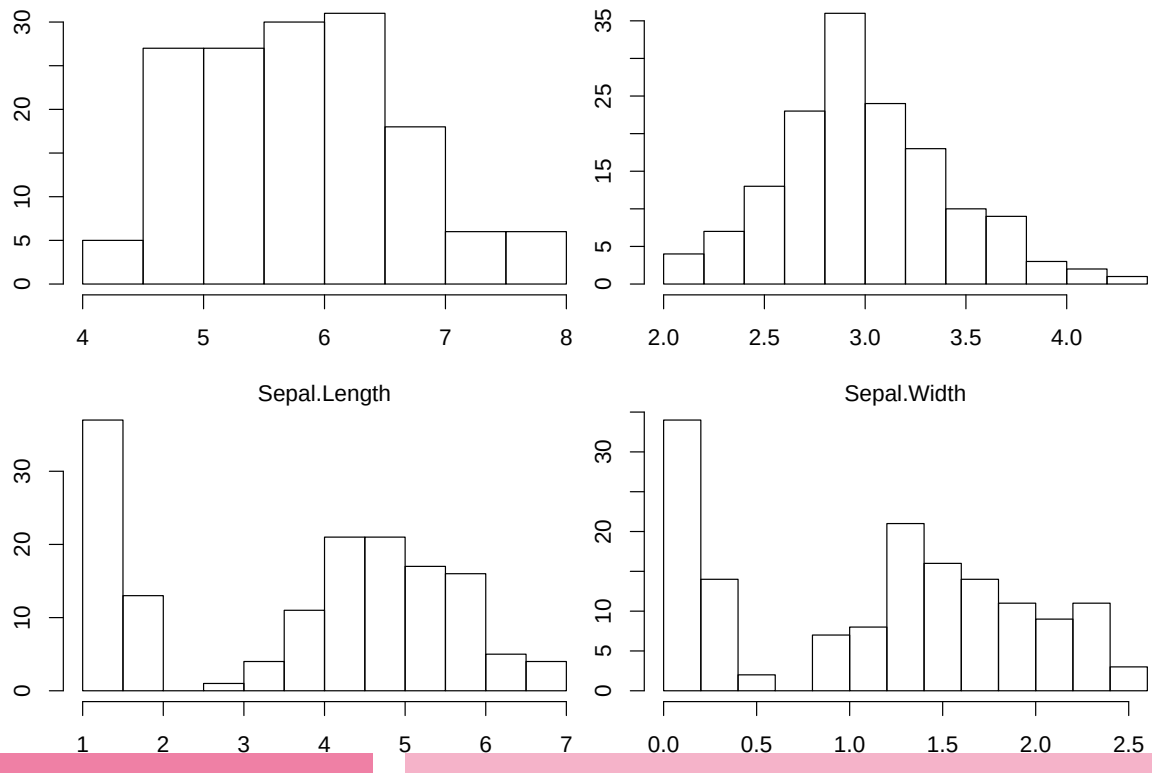
# Histogrammes

- combinaison de plusieurs graphiques

  `>par(mfrow=c(2,2))`

  deux lignes et deux colonnes
- boucle

  `> for(i in 1:4) { hist(iris[[i]],xlab=names(iris)[i],main="") }`



# Exercice

- `data()` : donne la liste des données disponibles
- exercice :
  - charger un tableau de données
  - comprendre sa structure et son contenu

# Plan

Manipulations élémentaires
- Vecteurs et tableaux de données
- Graphisme

Structures de contrôle
- Exécution conditionnelle
- Boucles
- Fonctions

Algèbre
- Vecteurs
- Matrices

# Exécution conditionnelle

- if

```
>x <- c(1,3)
>if(x[1]<x[2]) print(x[2])
[1] 3
```

- if else

```
>x <- c(1,3)
>if(x[1]>x[2]) print(x[2]) else print(x[1])
[1] 1
```

- ifelse

```
> x <- 1:4
> x
[1] 1 2 3 4
> y <- 5:8
> y
[1] 5 6 7 8
> t <- c(T,T,F,T)
> t
[1]  TRUE  TRUE FALSE  TRUE
> ifelse(t,x,y)
[1] 1 2 7 4
```

# Boucles

## for

```
> for(i in 1:4) { print(i) }
[1] 1
[1] 2
[1] 3
[1] 4
> for(u in c(3,-2,5)) { print(u) }
[1] 3
[1] -2
[1] 5
> for(species in levels(iris$Species)) {
    print(paste(species,sum(iris$Species==species)))
  }
[1] "setosa 50"
[1] "versicolor 50"
[1] "virginica 50"
```

# Boucles

- while

```
> i <- 10
> while(i<15) { print(i) ; i <- i+1}
[1] 10
[1] 11
[1] 12
[1] 13
[1] 14
```

- repeat

```
> i <- 15
> repeat { print(i) ; i <- i%/%2 ; if (i<=0) break }
[1] 15
[1] 7
[1] 3
[1] 1
```

# Exercices

- calculer la somme du contenu d'un tableau par une boucle
- calculer l'élément le plus petit d'un tableau par une boucle
- programmer l'équivalent de `ifelse` grâce à une boucle

# Fonctions

- les fonctions sont des objets

```
> f <- function(x) { x^2}
> class(f)
[1] "function"
> f(2)
[1] 4
```

- paramètres nommés (pas de type)

```
> g <- function(a,b) { a+b }
> g(1,2)
[1] 3
> g(1:2,3:4)
[1] 4 6
```

- le résultat est la dernière valeur calculée

```
> h <- function(a) { 2*a ; a-1}
> h(1:2)
[1] 0 1
```

# Plan

Manipulations élémentaires
- Vecteurs et tableaux de données
- Graphisme

Structures de contrôle
- Exécution conditionnelle
- Boucles
- Fonctions

Algèbre
- Vecteurs
- Matrices

# Calcul vectoriel

- combinaison linéaire

```
> x <- 1:3
> y <- c(-1,2,4)
> x+0.5*y
[1] 0.5 3.0 5.0
```

- produit scalaire :

```
> z <- x%*%y
> z
     [,1]
[1,]   15
> class(z)
[1] "matrix"
> dim(z)
[1] 1 1
>sum(x*y)
[1] 15
```

- norme :

```
> sqrt(sum(x^2))
[1] 3.741657
```

# Matrices

- création

```
> A <- matrix(1:6,ncol=2,nrow=3)
> A
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

le stockage se fait colonne par colonne

- informations

```
> class(A)
[1] "matrix"
> dim(A)
[1] 3 2
```

- accès au contenu

```
   > A[1,]
[1] 1 4
> A[1,2]
[1] 4
> A[1,]
[1] 1 4
> A[,2]
[1] 4 5 6
```

indexation similaire à celle des tableaux

# Calcul matriciel

- combinaison linéaire

```
> 2*A
     [,1] [,2]
[1,]    2    8
[2,]    4   10
[3,]    6   12
```

- produit

```
> u <- c(-1,0.5)
> A%*%u
     [,1]
[1,]  1.0
[2,]  0.5
[3,]  0.0
```

- attention aux opérations « terme à terme »

```
> A*A
     [,1] [,2]
[1,]    1   16
[2,]    4   25
[3,]    9   36
```

# Calcul matriciel

- transposition

```
> t(A)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

- résolution d'un système

```
> B <- A[1:2,1:2]
> B
     [,1] [,2]
[1,]    1    4
[2,]    2    5
> v <- solve(B,u)
> v
[1]  2.3333333 -0.8333333
> B%*%v
     [,1]
[1,] -1.0
[2,]  0.5
```

# Matrices et tableaux de données

- `as.matrix(T)` transforme un tableau de données `T` en une matrice
- délicat en raison du type des variables

```
> M <- as.matrix(iris)
> dim(M)
[1] 150   5
> class(M)
[1] "matrix"
> M[1,]
Sepal.Length  Sepal.Width Petal.Length  Petal.Width      Species
       "5.1"        "3.5"        "1.4"        "0.2"     "setosa"
> class(M[1,])
[1] "character"
```

  la conversion est uniforme : ici en texte

- solution possible : bloc par bloc

```
> M <- as.matrix(iris[1:4])
> M[1,]
Sepal.Length  Sepal.Width Petal.Length  Petal.Width
         5.1          3.5          1.4          0.2
> class(M[1,])
[1] "numeric"
```

# Exercices

Toujours écrire la solution sous forme d'une ou plusieurs fonctions

- programmer une régression linéaire simple
- programmer une régression linéaire pour plusieurs variables explicatives en utilisant le fait qu'une solution de

$$X^T X u = X v$$

s'obtient en `R` par

```
> X.qr <- qr(X)
> u <- qr.coeff(X.qr,v)
```