

# Introduction à R

Laurent Rouvière

Juin 2019

## Plan

## Table des matières

Présentation de la formation	1
Rstudio, Rmarkdown et packages R	8
Objets R	8
Importer des données	12
Manipuler des données avec Dplyr	15
Visualiser des données	19
Visualisation avec ggplot2	22

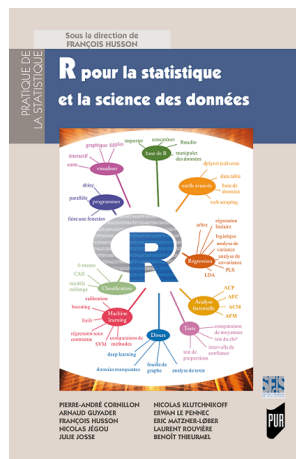
## Présentation de la formation

### Présentation

- *Prérequis* : bases en programmation, probabilités et statistique.
- *Objectifs* : **comprendre et utiliser les outils R classiques** en datascience :
  - importer et assembler des tables, manipuler des individus et des variables.
  - visualiser des données.
  - outils classiques et tidyverse.
- *Enseignant* : Laurent Rouvière, [laurent.rouviere@univ-rennes2.fr](mailto:laurent.rouviere@univ-rennes2.fr)
  - **Recherche** : statistique non paramétrique, apprentissage statistique.
  - **Enseignement** : statistique et probabilités (Université, école d'ingénieur, formation continue).
  - **Consulting** : énergie (ERDF), finance, marketing.

### Ressources

- Diapo et fiches d'exercices (1 fiche=1 ou 2 notions+exercices)
- Le net
- Livre : *R pour la statistique et la science des données*, PUR



## Pourquoi R ?

- De plus en plus de *données*, dans de plus en plus de **domaines** (énergie, santé, sport, économie. . .)
- La *science des données* contient tous les outils qui permettent d'**extraire de l'information** à partir de données. Elle comprend :
  - l'importation de données
  - la manipulation
  - la visualisation
  - le choix et l'entraînement de modèles
  - la visualisation de modèles (ils sont de plus en plus complexes. . .)
  - la restitution et la visualisation des résultats (applications web)

### Remarque importante

- **Toutes** ces notions peuvent être réalisées avec **R**.
- **R** (data scientists) et **Python** (informaticiens) font partie des outils les plus utilisés en sciences des données.

## Quelques mots sur R

- **R** est un *logiciel libre et gratuit*.
- Il est distribué par le CRAN (Comprehensive R Archive Network) à l'url suivante : <https://www.r-project.org>.
- Tous les statisticiens (notamment) *peuvent contribuer* en créant des fonctions et en les distribuant à la communauté (**packages**).

### Conséquence

- Le logiciel est **toujours à jour**.
- Une des principales raisons de son succès.

## Exemple : Les Iris de Fisher

```
> data(iris)
> 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
##
##
##
```

## Objectifs

### La problématique

Expliquer *species* par les autres variables.

- Species est *variable qualitative*.
- Confronté à un problème de *classification supervisée*.

## Manipulation des données

```
> apply(iris[,1:4],2,mean)
## Sepal.Length Sepal.Width Petal.Length Petal.Width
## 5.843333 3.057333 3.758000 1.199333
> apply(iris[,1:4],2,var)
## Sepal.Length Sepal.Width Petal.Length Petal.Width
## 0.6856935 0.1899794 3.1162779 0.5810063
```

### Remarque

Non informatif pour le problème (expliquer Species).

## Manipulation avec dplyr

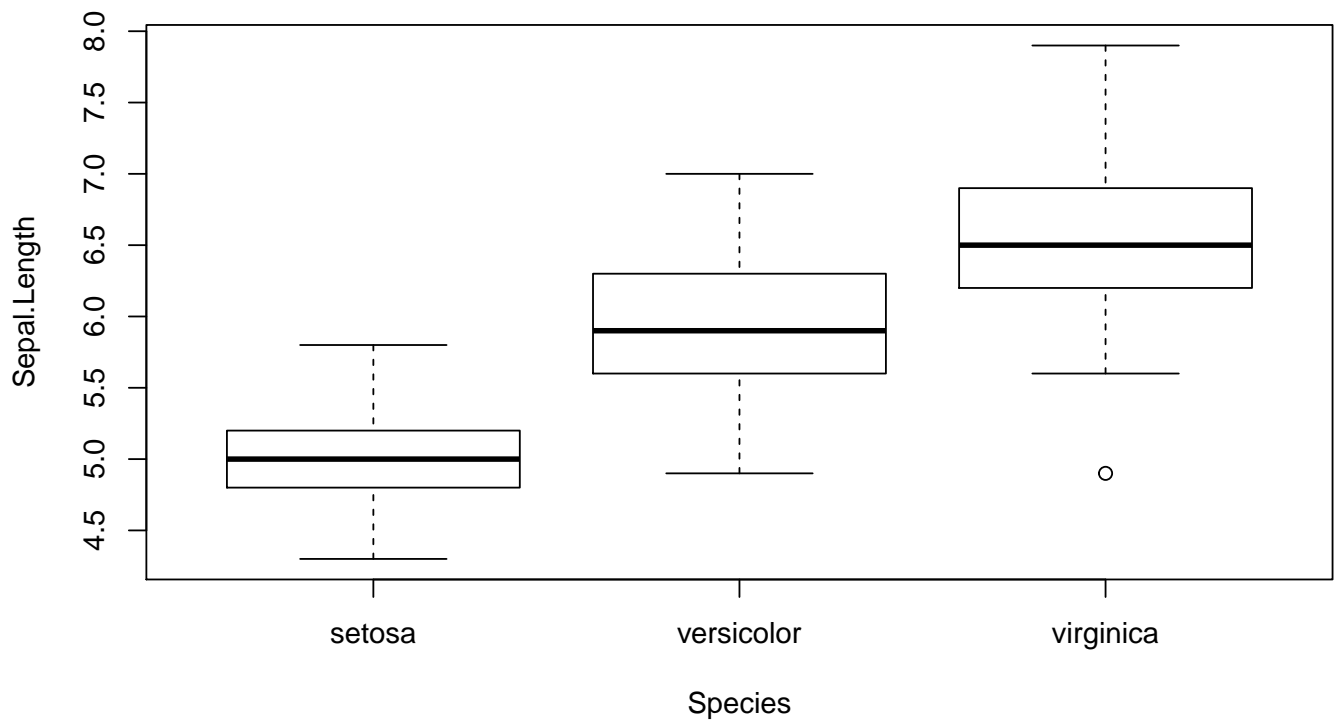
- *dplyr* est un package de **tidyverse** qui permet de faciliter la manipulation des données, notamment en terme de **syntaxe**.

```
> library(dplyr)
> iris %>% group_by(Species) %>% summarise_all(mean)
## # A tibble: 3 x 5
##   Species Sepal.Length Sepal.Width Petal.Length Petal.Width
##   <fct>      <dbl>      <dbl>      <dbl>      <dbl>
## 1 setosa      5.01        3.43        1.46        0.246
## 2 versicolor 5.94        2.77        4.26        1.33
## 3 virginica  6.59        2.97        5.55        2.03
```

- *Plus intéressant* : nous obtenons les moyennes pour **chaque espèce**.

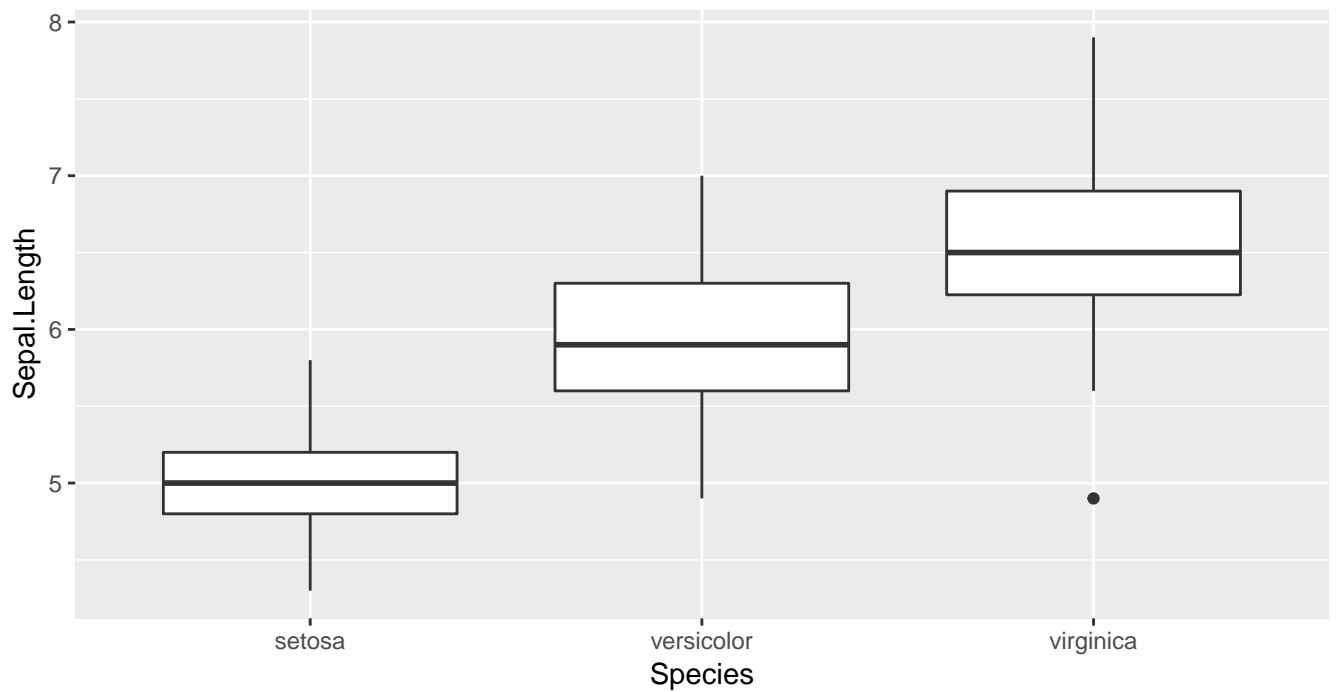
## Visualisation

```
> boxplot(Sepal.Length~Species,data=iris)
```



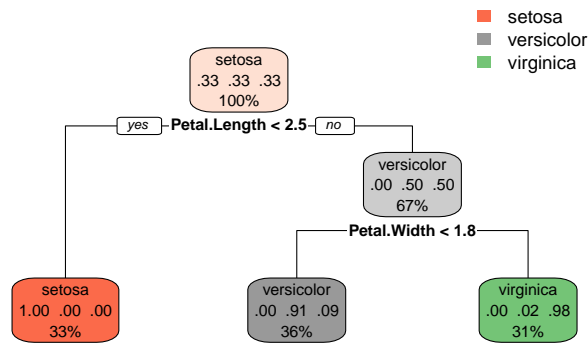
### Visualization avec ggplot2

```
> library(ggplot2)
> ggplot(iris)+aes(x=Species,y=Sepal.Length)+geom_boxplot()
```



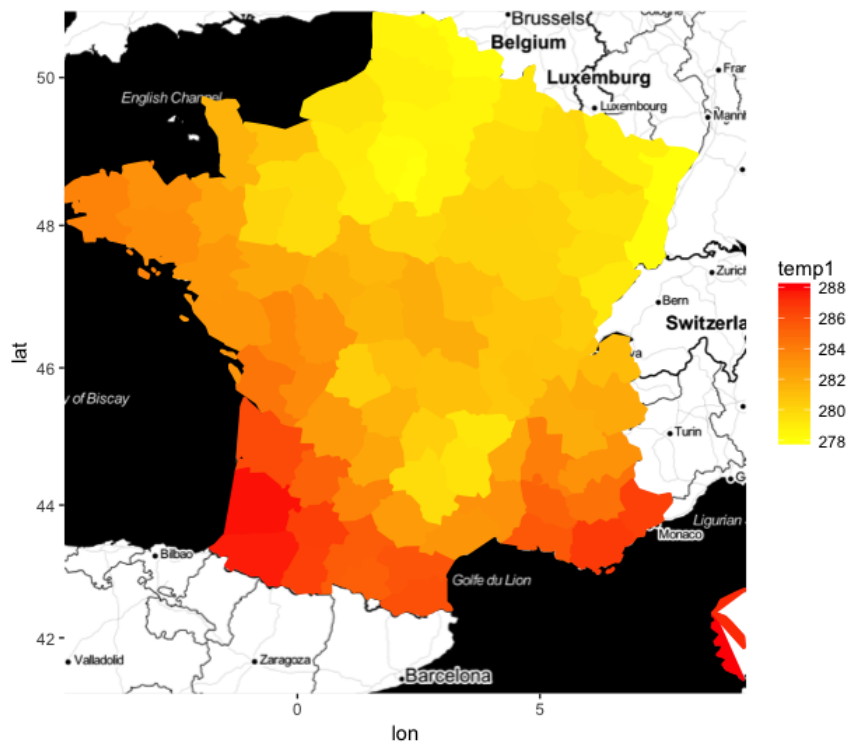
### Un modèle d'arbre

```
> library(rpart)
> tree <- rpart(Species~.,data=iris)
> library(rpart.plot)
> rpart.plot(tree)
```



## Carte avec ggmap

— *Objectif* : visualiser les températures en france pour une date donnée.



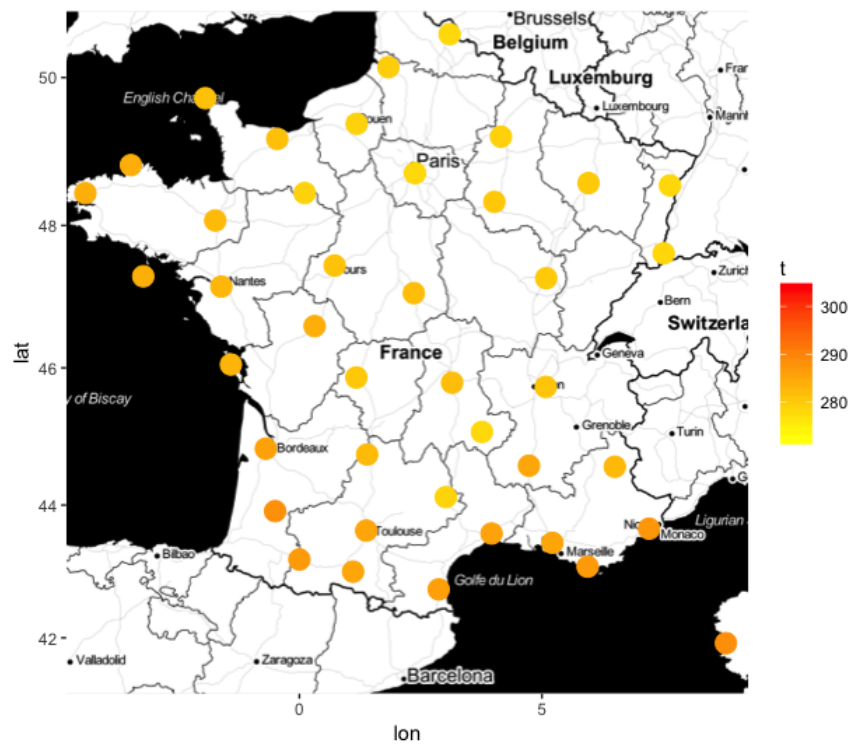
## Chargement des données + fond de carte

— Données téléchargées sur le site de meteofrance (températures for about 60 stations).

```

> donnees <- fread("https://donneespubliques.meteofrance.fr/
+ donnees_libres/Txt/Synop/synop.2017082815.csv")
> station <- fread("https://donneespubliques.meteofrance.fr/
+ donnees_libres/Txt/Synop/postesSynop.csv")
> fond <- get_map("France",maptype="toner",zoom=6)
> ggmap(fond)+geom_point(data=D,
+ aes(y=Latitude,x=Longitude,color=t),size=5)+
+ scale_color_continuous(low="yellow",high="red")
  
```

## Une première carte



## Modèle de prévision

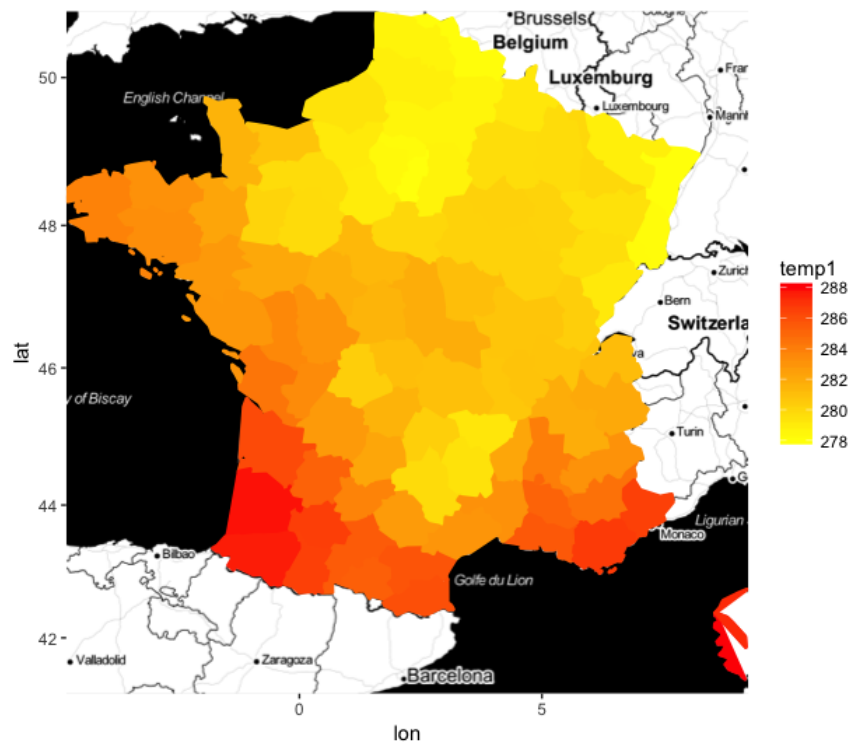
- **Algorithme** de *plus proche voisins* pour estimer la température sur toutes les longitudes et latitudes du territoire.

```
> library(FNN)
> mod <- knn.reg(train=D[,.(Latitude,Longitude)],y=D[,t],
+               test=Test1[,.(Latitude,Longitude)],k=1)$pred
```

- Visualisation avec *ggmap*.

```
> library(ggmap)
> ggmap(fond)+geom_polygon(data=Test5,
+   aes(y=Latitude,x=Longitude,
+       fill=temp1,color=temp1,group=dept),size=1)+
+   scale_fill_continuous(low="yellow",high="red")+
+   scale_color_continuous(low="yellow",high="red")
```

## La carte finale



## Application web avec shiny

- *Shiny* est un package R qui permet la création de **pages web interactives**.
- **Exemple** : graphiques standards pour un jeu de données.

```
> library(shiny)
> runApp('desc_app.R')
```

## Dans cette formation

- 18 heures pour 5 (ou 6) thèmes
- 1 thème = **slides** + **fiche** (notebook) à compléter (ajouter des commentaires et faire les exercices)

## Notebook R

- reporting qui combine des **commandes R** et leurs **sorties** avec du **texte**.
- le code peut être **exécuté de façon interactive**, la **sortie est visible** immédiatement sous la code.
- très approprié pour des rapports de très bonne qualité.

## Plan

### Thèmes

1. Rstudio - Rmarkdown (notebook et slides) (J1)
2. Objets R (vecteurs, matrices, dates, data.frames...) (J1)
3. Importation de données et manipulation "standard" (J1)
4. Manipulation avec dplyr (J2)
5. Visualisation avec ggplot2 (J2-3)
6. Carte ? Modèles ?

# Rstudio, Rmarkdown et packages R

## Rstudio

- **RStudio** est une *interface* facilitant l'utilisation de R.
- Egalement *libre et gratuit* : <https://www.rstudio.com>.

L'écran est divisé en 4 parties :

- *Console* : pour entrer les commandes et visualiser les sorties.
- *Workspace and History* : visualiser l'historique des objets créés.
- *Files Plots...* : voir les répertoires et fichiers dans l'environnement de travail, les graphes de sortie, installer les packages...
- *Script* : éditeur pour entrer les commandes R et les commentaires. Penser à *régulièrement sauvegarder ce fichier*!

## Rmarkdown

### Fichier Rmarkdown

- Un fichier **Rmarkdown (.Rmd)** permet de produire un document de travail.
- Il contient le **code**, les **sorties** et des **commentaires** sur le travail réalisé.
- Il produit des **rapports de grande qualité** de différentes formes (documents, diaporama, etc...).
- Ce diaporama est du *Rmarkdwon*.
- *Recherche Reproductible* : en cliquant sur un bouton, on peut ré-exécuter tout le code du fichier et **exporter les résultats sous un format rapport**.
- *Documents dynamiques* : possibilité d'**exporter le rapport final dans différents formats** : html, pdf, rtf, slides, notebook...

## Packages

- *Ensemble de programmes R* qui complètent et améliorent les fonctions de **R**.
- Un package est généralement dédié à des *méthodes ou domaines d'application spécifiques*.
- Plus de *13 000* packages actuellement.
- Contribue au *succès* de R (**toujours à jour**).

### 2 phases

- Installation : **install.packages(package.name)** (une seule fois).
- Chargement : **library(package.name)** (chaque fois).
- On peut aussi utiliser le bouton **package** dans **Rstudio**.

⇒ *Fiche 1.*

## Objets R

### Numérique et caractères

- Numérique (facile)

```
> x <- pi
> x
## [1] 3.141593
> is.numeric(x)
## [1] TRUE
```

- Caractères



```
> b <- "x"
> paste(b,1:5,sep="")
## [1] "X1" "X2" "X3" "X4" "X5"
```

## Vecteurs

— *Création : c, seq, rep*

```
> x1 <- c(1,3,4)
> x2 <- 1:5
> x3 <- seq(0,10,by=2)
> x4 <- rep(x1,3)
> x5 <- rep(x1,3,each=3)
```

— *Extraction :*

```
> x3[c(1,3,4)] # pareil que x3[x1]
## [1] 0 4 6
```

## Logique

```
> 1<2
## [1] TRUE
> 1==2
## [1] FALSE
> 1!=2
## [1] TRUE
```

```
> x <- 1:3
> test <- c(TRUE,FALSE,TRUE)
> x[test]
## [1] 1 3
```

```
> size <- runif(5,150,190) #5 sizes randomly generated between
>                               #150 and 190
> size
## [1] 178.8362 185.0309 180.4393 185.4450 168.2592
```

## Problème

Sélectionner les tailles plus grandes que 174.

```
> size>174
## [1] TRUE TRUE TRUE TRUE FALSE
> size[size>174]
## [1] 178.8362 185.0309 180.4393 185.4450
```

## Facteurs

— Pour les *variables qualitatives* :

```
> x1 <- factor(c("a","b","b","a","a"))
> x1
## [1] a b b a a
## Levels: a b
> levels(x1)
## [1] "a" "b"
```

## Variable mal configurée

— On suppose que les données sont codées : 0=homme, 1=femme

```
> X <- c(1,1,0,0,1)
> summary(X)
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       0.0     0.0     1.0     0.6     1.0     1.0
```

— *Problème* : **R** interprète  $X$  comme un vecteur **continu**  $\Rightarrow$  cela peut générer des problèmes dans l'étude statistique.

— **Solution** :

```
> X <- as.factor(X)
> levels(X) <- c("man", "woman")
> X
## [1] woman woman man   man   woman
## Levels: man woman
> summary(X)
##      man woman
##       2     3
```

## Matrice

— Création

```
> m <- matrix(1:4,nrow=2,byrow=TRUE)
> m
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
```

— Extraction

```
> m[1,2]
> m[1,] #First row
> m[,2] #Second column
```

## Liste

— Permet de considérer *plusieurs objets* :

```
> mylist <- list(vector=1:5,mat=matrix(1:8,nrow=2))
> mylist
## $vector
## [1] 1 2 3 4 5
##
## $mat
##      [,1] [,2] [,3] [,4]
## [1,]    1    3    5    7
## [2,]    2    4    6    8
```

— Extraction :

```
> mylist[[1]]
> mylist$vector
> mylist[["vector"]]
```

## Dataframe

— Objets pour représenter des *données* dans **R**

```
> name <- c("Paul", "Mary", "Steven", "Charlotte", "Peter")
> sex <- c(0,1,0,1,0)
> size <- c(180,165,168,170,175)
> data <- data.frame(name,sex,size)
> data
##      name sex size
## 1    Paul   0  180
## 2    Mary   1  165
## 3  Steven   0  168
## 4 Charlotte 1  170
## 5    Peter   0  175
```

```
> summary(data)
##      name      sex      size
## Charlotte:1 Min.   :0.0 Min.   :165.0
## Mary      :1 1st Qu.:0.0 1st Qu.:168.0
## Paul      :1 Median :0.0 Median :170.0
## Peter     :1 Mean   :0.4 Mean   :171.6
## Steven    :1 3rd Qu.:1.0 3rd Qu.:175.0
##           Max.   :1.0 Max.   :180.0
```

## Problème

`sex` est interprété comme une *variable continue*. C'est une **variable qualitative**.

```
> data$sex <- as.factor(data$sex)
> levels(data$sex) <- c("man", "woman")
> summary(data)
##      name      sex      size
## Charlotte:1 man   :3 Min.   :165.0
## Mary      :1 woman:2 1st Qu.:168.0
## Paul      :1      Median :170.0
## Peter     :1      Mean   :171.6
## Steven    :1      3rd Qu.:175.0
##           Max.   :180.0
```

## Problème

`name` est interprété comme une *variable*. C'est plutôt un *identifiant*.

```
> row.names(data) <- data$name
> data <- data[,-1] #suppression de la colonne name
> data
##      sex size
## Paul   man  180
## Mary   woman 165
## Steven man  168
## Charlotte woman 170
## Peter  man  175
```

## Conclusion

Il est **crucial** de toujours vérifier que les données sont **correctement interprétées** par **R** (avec `summary` ou `mode` par exemple).

## Tibbles

- Un *tibble* est une version **moderne** du dataframe, qui conserve les avantages et supprime les inconvénients (selon les créateurs du tibble).
- C'est la version dataframe du *tidyverse* (nécessité de charger ce package).
- Deux différences notables :
  - les variables **qualitatives** sont par défaut des **caractères** (et non des facteurs);
  - pas de **rownames**.

## Exemple : data frame

```

> name <- c("Paul","Mary","Steven","Charlotte","Peter")
> sex <- c(0,1,0,1,0)
> size <- c(180,165,168,170,175)
> age <- c("old","young","young","old","old")
> data <- data.frame(sex,size,age)
> rownames(data) <- name
> summary(data)
##      sex      size      age
##  Min.   :0.0   Min.   :165.0   old   :3
##  1st Qu.:0.0   1st Qu.:168.0   young:2
##  Median :0.0   Median :170.0
##  Mean   :0.4   Mean   :171.6
##  3rd Qu.:1.0   3rd Qu.:175.0
##  Max.   :1.0   Max.   :180.0

```

## Example : tibble

```

> library(tidyverse)
> data1 <- tibble(name,sex,size,age)
> #data1 <- column_to_rownames(data1,var="name")
> summary(data1)
##      name      sex      size      age
##  Length:5      Min.   :0.0   Min.   :165.0   Length:5
##  Class :character 1st Qu.:0.0   1st Qu.:168.0   Class :character
##  Mode  :character Median :0.0   Median :170.0   Mode  :character
##                Mean  :0.4   Mean   :171.6
##                3rd Qu.:1.0   3rd Qu.:175.0
##                Max.   :1.0   Max.   :180.0

```

## dataframe vs tibbles

Principale différence : **pas de facteur** dans les tibbles (par défaut).

⇒ *fiche 2.*

## Importer des données

- Les données sont généralement contenues dans un *fichier* avec les individus en ligne et les variables en colonnes.
- Les fonctions **read.table** et **read.csv** permettent d'**importer des données** à partir de fichiers *.txt* et *.csv*.
- Pour les fichiers *.xls*, on peut les *convertir* en *.csv* ou utiliser des packages spécifiques.

```

> data <- read.table("file",...)
> data <- read.csv("file",...)

```

- ... correspondent à un tas d'**options** souvent très *importantes* car les fichiers de données contiennent **toujours des spécificités** (données manquantes, noms de variables...)

## Indiquer le chemin

- Le **fichier des données** doit être placé dans le **répertoire de travail**. Sinon, il faut indiquer le *chemin* à **read.table**.
- *Example* : Importer le fichier *data.csv* enregistré dans */lectureR/Part1* :
- Changement du répertoire de travail

```

> setwd("~/lectureR/Part1")
> df <- read.csv("data.csv",...)

```

- Spécification du chemin dans **read.csv**

```

> df <- read.csv("~/lecture_R/Part1/data.csv",...)

```

- Utilisation de la fonction **file.path**

```
> path <- file.path("~/lecture_R/Part1/", "data.csv")
> df <- read.csv(path,...)
```

## Quelques options importantes

Il y a plusieurs *options* importantes dans **read.table** et **read.csv** :

- **sep** : le caractère de séparation (espace, virgule...)
- **dec** : le caractère pour le séparateur décimal (virgule, point...)
- **header** : logique pour indiquer si le nom des variables est spécifié à la première ligne du fichier
- **row.names** : vecteurs des identifiants (si besoin)
- **na.strings** : vecteur de caractères pour repérer les données manquantes.
- ...

## Exemple

- Fichier *data\_imp.txt*

```
name;size;age
John;174;32
Peter;?;28
Mary;165.5;NA
```

## Caractéristiques

- 3 variables (ou plutôt 2...)
- Première ligne = nom des variables
- Données manquantes = NA, ?

## Un premier essai

```
> path <- file.path("~/COURS/EDHEC/R/SLIDES/", "data_imp.txt")
```

```
> df <- read.table(path)
> summary(df)
##           V1
## John;174;32 :1
## Mary;165.5;NA:1
## name;size;age:1
## Peter;?;28   :1
```

## Problème

R considère quatre lignes et **une** colonne !

## Solution

```
> df <- read.table(path,header=TRUE,sep=";",dec=".",
+                 na.strings = c("NA","?"),row.names = 1)
> df
##      size age
## John 174.0 32
## Peter  NA  28
## Mary 165.5 NA
> summary(df)
##      size      age
## Min.   :165.5   Min.   :28
## 1st Qu.:167.6   1st Qu.:29
## Median :169.8   Median :30
## Mean   :169.8   Mean   :30
## 3rd Qu.:171.9   3rd Qu.:31
## Max.   :174.0   Max.   :32
## NA's   :1       NA's   :1
```

## Package readr

- Version *tidyverse* pour l'importation.
- Il contient `read_table` et `read_csv` à la place de `read.table` et `read.csv` (underscores à la place des points).
- Dans *Rstudio*, on peut lire des données avec `readr` en cliquant sur **Import Dataset** (pas toujours efficace pour des données complexes).

## Autres outils importations

- `readxl` : fichier au format Excel.
- `sas7bdat` : importation depuis SAS.
- `foreign` : formats SPSS ou STATA
- `jsonlite` : format JSON
- `rvest` : webscrapping

## Concaténer des données

- L'information utile pour une analyse provient (souvent) de *plusieurs tableaux de données*.
- Besoin de *correctement assembler ces tables* avant l'étude statistique.
- *Fonctions R standard* : `rbind`, `cbind`, `cbind.data.frame`, `merge`...
- *Fonctions R tidyverse* : `bind_rows`, `bind_cols`, `left_join`, `inner_join`.

## Un exemple avec 2 tables

```
> df1
## # A tibble: 4 x 2
##   name nation
##   <chr> <chr>
## 1 Peter USA
## 2 Mary  GB
## 3 John  Aus
## 4 Linda USA
> df2
## # A tibble: 3 x 2
##   name age
##   <chr> <dbl>
## 1 John  35
## 2 Mary  41
## 3 Fred  28
```

## Objectif

Un tableau de données avec 3 colonnes : name, nation et age.

## bind\_rows

```
> bind_rows(df1,df2)
## # A tibble: 7 x 3
##   name nation age
##   <chr> <chr> <dbl>
## 1 Peter USA    NA
## 2 Mary  GB      NA
## 3 John  Aus      NA
## 4 Linda USA    NA
## 5 John  <NA>    35
## 6 Mary  <NA>    41
## 7 Fred  <NA>    28
```

⇒ Pas un bon choix ici (2 lignes pour certains individus).

## full\_join

```
> full_join(df1,df2)
## # A tibble: 5 x 3
##   name nation age
##   <chr> <chr> <dbl>
## 1 Peter USA    NA
## 2 Mary  GB     41
## 3 John  Aus     35
## 4 Linda USA     NA
## 5 Fred  <NA>    28
```

⇒ tous les individus sont conservés (NA sont ajoutés pour les quantités non mesurées.)

## left\_join

```
> left_join(df1,df2)
## # A tibble: 4 x 3
##   name nation age
##   <chr> <chr> <dbl>
## 1 Peter USA    NA
## 2 Mary  GB     41
## 3 John  Aus     35
## 4 Linda USA     NA
```

⇒ seuls les individus du premier tableau (gauche) sont conservés.

## inner\_join

```
> inner_join(df1,df2)
## # A tibble: 2 x 3
##   name nation age
##   <chr> <chr> <dbl>
## 1 Mary  GB     41
## 2 John  Aus     35
```

⇒ on garde les individus pour lesquels **nation** et **age** sont mesurés.

## Conclusion

- Plusieurs possibilités pour assembler des données.
- Important de faire le bon choix en fonction du contexte.

⇒ *Fiche 3 - Partie 1*

## Manipuler des données avec Dplyr

- *dplyr* est un package puissant qui permet de *transformer et résumer* des tableaux de données.
- Il propose une **syntaxe claire** (basée sur une *grammaire*) pour manipuler les données.
- Par exemple, pour calculer le moyenne de *Sepal.Length* pour l'espèce *setosa*, utilise généralement

```
> mean(iris[iris$Species=="setosa",]$Sepal.Length)
## [1] 5.006
```

- On peut faire la même chose avec **dplyr**

```
> library(dplyr)
> iris %>% filter(Species=="setosa") %>%
+   summarise(mean(Sepal.Length))
##   mean(Sepal.Length)
## 1                5.006
```

## Grammaire dplyr

**dplyr** propose une *grammaire* dont les principaux **verbes** sont :

- **select()** : sélectionner des colonnes (variables)
- **filter()** : filter des lignes (individus)
- **arrange()** : ordonner des lignes
- **mutate()** : créer des nouvelles colonnes (nouvelles variables)
- **summarise()** : calculer des résumés numériques (ou résumés statistiques)
- **group\_by()** : effectuer des opérations pour des groupes d'individus

Penser à consulter la **cheat sheet**

## Select

### *But*

Sélectionner des **variables**.

```
> df <- select(iris,Sepal.Length,Petal.Length)
> head(df)
##   Sepal.Length Petal.Length
## 1         5.1         1.4
## 2         4.9         1.4
## 3         4.7         1.3
## 4         4.6         1.5
## 5         5.0         1.4
## 6         5.4         1.7
```

## Filter

### *But*

Filtrer des **individus**.

```
> df <- filter(iris,Species!="versicolor")
> head(df)
##   Sepal.Length Sepal.Width Petal.Length Petal.Width  Species
## 1         7.0         3.2         4.7         1.4 versicolor
## 2         6.4         3.2         4.5         1.5 versicolor
## 3         6.9         3.1         4.9         1.5 versicolor
## 4         5.5         2.3         4.0         1.3 versicolor
## 5         6.5         2.8         4.6         1.5 versicolor
## 6         5.7         2.8         4.5         1.3 versicolor
```

## Arrange

### *But*

Ordonner des **individus** en fonction d'une variable.

```
> df <- arrange(iris,Sepal.Length)
> head(df)
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1         4.3         3.0         1.1         0.1  setosa
## 2         4.4         2.9         1.4         0.2  setosa
## 3         4.4         3.0         1.3         0.2  setosa
## 4         4.4         3.2         1.3         0.2  setosa
## 5         4.5         2.3         1.3         0.3  setosa
## 6         4.6         3.1         1.5         0.2  setosa
```

## Mutate

### *But*

Définir des **nouvelles variables** dans le jeu de données.



```
> df <- mutate(iris,diff_petal=Petal.Length-Petal.Width)
> head(select(df,Petal.Length,Petal.Width,diff_petal))
##   Petal.Length Petal.Width diff_petal
## 1         1.4         0.2         1.2
## 2         1.4         0.2         1.2
## 3         1.3         0.2         1.1
## 4         1.5         0.2         1.3
## 5         1.4         0.2         1.2
## 6         1.7         0.4         1.3
```

## Summarise

### But

Calculer des **résumés statistiques**.

```
> summarise(iris,mean=mean(Petal.Length),var=var(Petal.Length))
##   mean      var
## 1 3.758 3.116278
```

## group\_by

### But

Faire des opérations pour des **groupes de données**.

```
> summarise(group_by(iris,Species),mean(Petal.Length))
## # A tibble: 3 x 2
##   Species      'mean(Petal.Length)'
##   <fct>          <dbl>
## 1 setosa          1.46
## 2 versicolor     4.26
## 3 virginica       5.55
```

## L'opérateur pipe %>%

- L'opérateur pipe **%>%** permet d'organiser les commandes *étape par étape*.
- Par exemple,

```
> mean(iris[iris$Species=="setosa","Sepal.Length"])
## [1] 5.006
```

ou (un peu plus lisible)

```
> df1 <- iris[iris$Species=="setosa",]
> df2 <- df1$Sepal.Length
> mean(df2)
## [1] 5.006
```

or (un peu plus lisible avec **dplyr**)

```
> df1 <- filter(iris,Species=="setosa")
> df2 <- select(df1,Sepal.Length)
> summarize(df2,mean(Sepal.Length))
##   mean(Sepal.Length)
## 1          5.006
```

- Avec le *pipe*, on **décompose** les opérations :

1. Les données

```
> iris
```

2. On filtre les individus **setosa**

```
> iris %>% filter(Species=="setosa")
```

3. On garde la variable d'intérêt

```
> iris %>% filter(Species=="setosa") %>% select(Sepal.Length)
```

4. On calcule la moyenne

```
> iris %>% filter(Species=="setosa") %>%  
+   select(Sepal.Length)%>% summarize_all(mean)  
##   Sepal.Length  
## 1          5.006
```

## Plus généralement

- L'opérateur pipe `%>%` applique l'*objet de droite* en considérant que le premier argument est l'*objet de gauche* (non symétrique).

```
> X <- as.numeric(c(1:10,"NA"))  
> mean(X,na.rm = TRUE)  
## [1] 5.5
```

ou, de façon équivalente,

```
> X %>% mean(na.rm=TRUE)  
## [1] 5.5
```

## Reshaping data

- Certaines analyses statistiques nécessitent un *format particulier* pour les données.
- Un exemple jouet

```
> df <- iris %>% group_by(Species) %>%  
+   summarize_all(funs(mean))  
> head(df)  
## # A tibble: 3 x 5  
##   Species    Sepal.Length Sepal.Width Petal.Length Petal.Width  
##   <fct>          <dbl>         <dbl>         <dbl>         <dbl>  
## 1 setosa          5.01           3.43           1.46           0.246  
## 2 versicolor      5.94           2.77           4.26           1.33  
## 3 virginica       6.59           2.97           5.55           2.03
```

## gather

- **Assembler** des colonnes en lignes avec *gather* :

```
> df1 <- df %>% gather(key=variable,value=value,-Species)  
> head(df1)  
## # A tibble: 6 x 3  
##   Species    variable    value  
##   <fct>      <chr>      <dbl>  
## 1 setosa    Sepal.Length  5.01  
## 2 versicolor Sepal.Length  5.94  
## 3 virginica Sepal.Length  6.59  
## 4 setosa    Sepal.Width   3.43  
## 5 versicolor Sepal.Width   2.77  
## 6 virginica Sepal.Width   2.97
```

## Remarque

Même information avec un *format différent*.

## Spread

- Décomposer une ligne en plusieurs colonnes avec *spread*.

```
> df1 %>% spread(variable,value)
## # A tibble: 3 x 5
##   Species      Petal.Length Petal.Width Sepal.Length Sepal.Width
##   <fct>          <dbl>          <dbl>          <dbl>          <dbl>
## 1 setosa         1.46           0.246          5.01           3.43
## 2 versicolor     4.26           1.33           5.94           2.77
## 3 virginica      5.55           2.03           6.59           2.97
```

## Separate

- *Séparer* une colonne en plusieurs.

```
> df <- tibble(date=as.Date(c("01/03/2015","05/18/2017",
+ "09/14/2018"),"%m/%d/%Y"),temp=c(18,21,15))
```

```
> df1 <- df %>% separate(date,into = c("year","month","day"))
> df1
## # A tibble: 3 x 4
##   year month day    temp
##   <chr> <chr> <chr> <dbl>
## 1 2015   01    03     18
## 2 2017   05    18     21
## 3 2018   09    14     15
```

## Unite

- *Assembler* des colonnes.

```
> df1 %>% unite(date,year,month,day,sep="/")
## # A tibble: 3 x 2
##   date      temp
##   <chr>    <dbl>
## 1 2015/01/03    18
## 2 2017/05/18    21
## 3 2018/09/14    15
```

⇒ *Fiche 3, partie 2.*

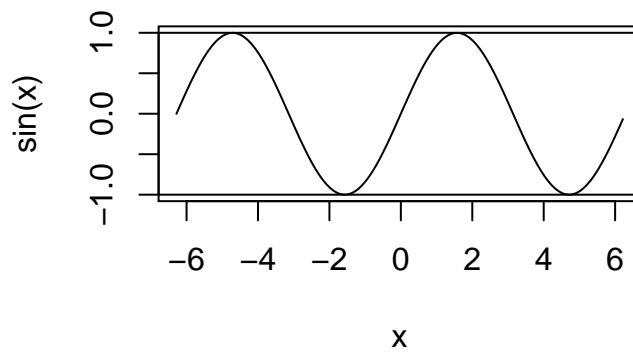
## Visualiser des données

- *Visualisation* : cruciale à **toutes les étapes** d'une étude statistique.
- **R** Permet de créer un **très grand nombre** de type de graphes.
- On propose une (courte) présentations des *graphes classiques*,
- suivie par les graphes **ggplot**.

## La fonction plot

- Fonction *générique* pour représenter (presque) **tous les types de données**.
- Pour un *nuage de points*, il suffit de renseigner un **vecteur** pour l'axe des **x**, et un autre vecteur pour celui des **y**.

```
> x <- seq(-2*pi,2*pi,by=0.1)
> plot(x,sin(x),type="l",xlab="x",ylab="sin(x)")
> abline(h=c(-1,1))
```



### Graphes classiques pour visualiser des variables

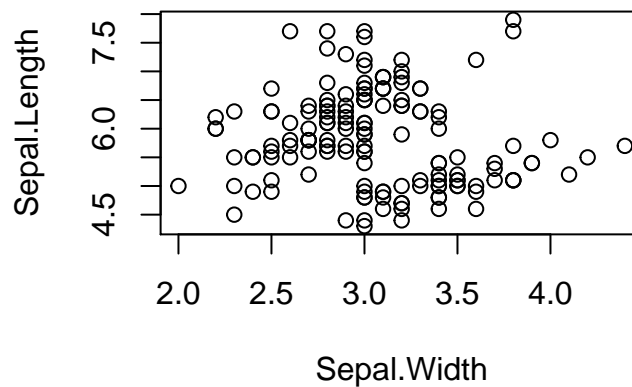
- **Histogramme** pour une variable *continue*, **diagramme en barre** pour une variable *qualitative*.
- **Nuage de points** pour 2 variables continues.
- **Boxplot** pour une distribution continue.

### *Constat (positif)*

Il existe une **fonction R** pour toutes les représentations.

### Nuage de points sur un jeu de données

```
> plot(Sepal.Length~Sepal.Width,data=iris)
```

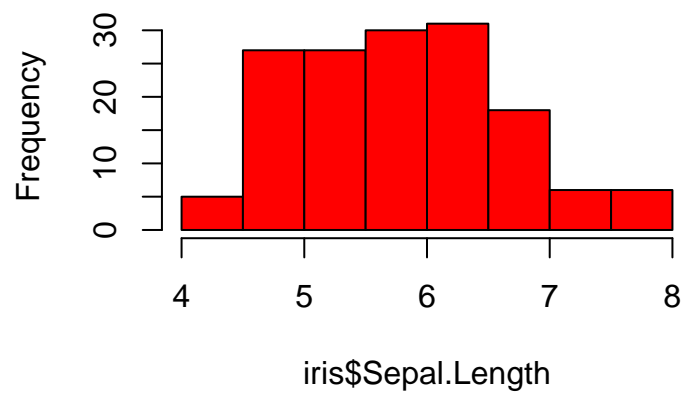


```
> #same as
> plot(iris$Sepal.Width,iris$Sepal.Length)
```

### Histogramme (variable continue)

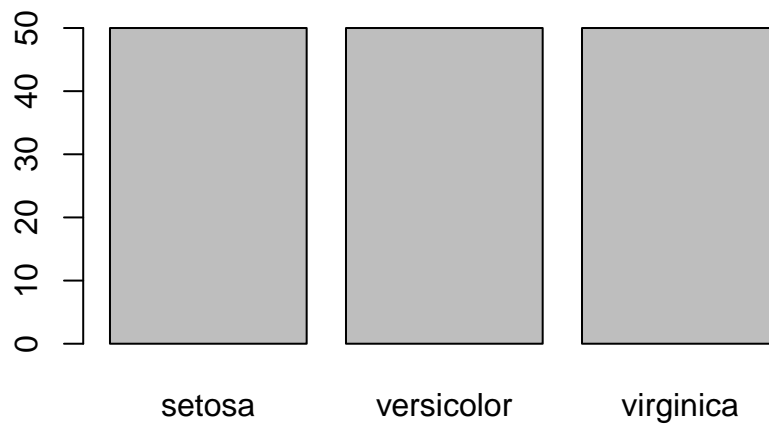
```
> hist(iris$Sepal.Length,col="red")
```

## Histogram of iris\$Sepal.Length



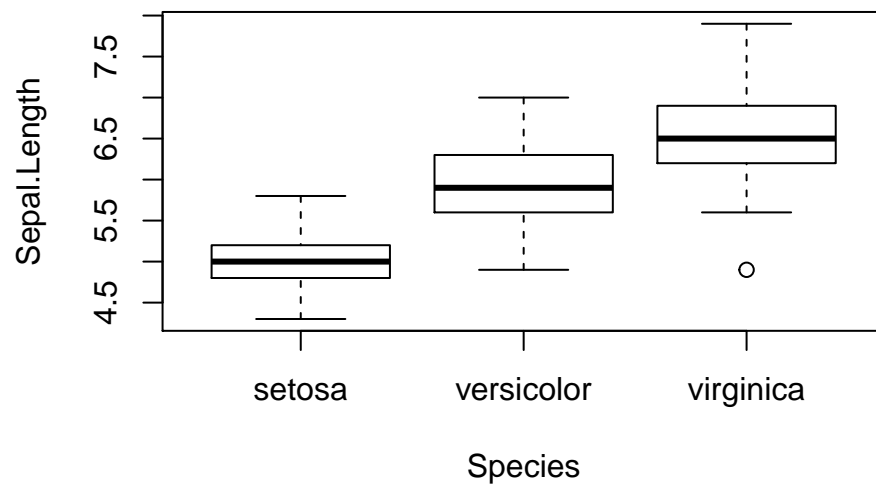
## Diagramme en barres (variable qualitative)

```
> barplot(table(iris$Species))
```



## Boxplot (distribution)

```
> boxplot(Sepal.Length~Species,data=iris)
```



## Visualisation avec ggplot2

- *ggplot2* permet de faire des graphes R en s'appuyant sur une **grammaire des graphiques** (équivalent de **dplyr** pour manipuler les données).
- Les graphes produits sont *agréables à regarder* (pas toujours le cas avec les graphes conventionnels).
- La *grammaire ggplot* permet d'obtenir des **graphes "complexes"** avec une **syntaxe claire et lisible**.

Pour un tableau de données fixé, un graphe est défini comme une succession de **couches**. Il faut toujours spécifier :

- les *données*
- les *variables* à représenter
- le *type de représentation* (nuage de points, boxplot...).

Les graphes ggplot sont construits à partir de ces couches. On indique

- les données avec **ggplot**
- les variables avec **aes** (aesthetics)
- le type de représentation avec **geom\_**

### La grammaire

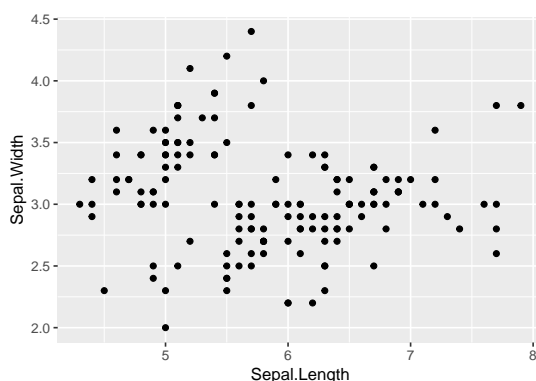
Les principaux *verbes* sont

- **Data (ggplot)** : les *données*, un dataframe ou un tibble.
- **Aesthetics (aes)** : façon dont les *variables* doivent être représentées.
- **Geometrics (geom\_...)** : *type* de représentation.
- **Statistics (stat\_...)** : spécifier les *transformations* des données.
- **Scales (scale\_...)** : modifier certains *paramètres du graphe* (changer de couleurs, de taille...).

Tous ces éléments sont **séparés par un +**.

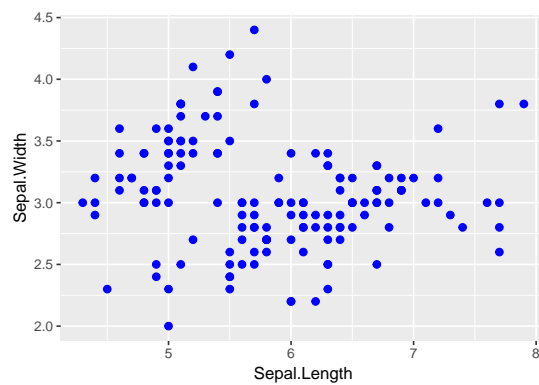
### Un exemple

```
> ggplot(iris)+aes(x=Sepal.Length,y=Sepal.Width)+geom_point()
```



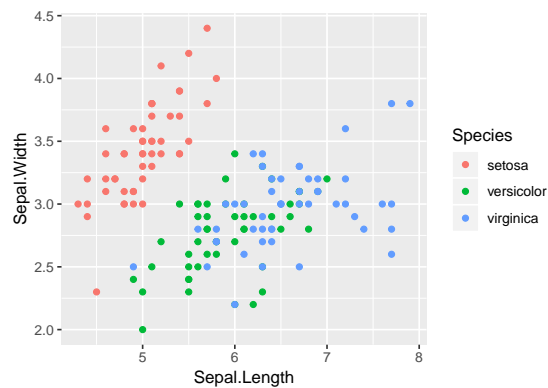
### Couleur et taille

```
> ggplot(iris)+aes(x=Sepal.Length,y=Sepal.Width)+
+   geom_point(color="blue",size=2)
```



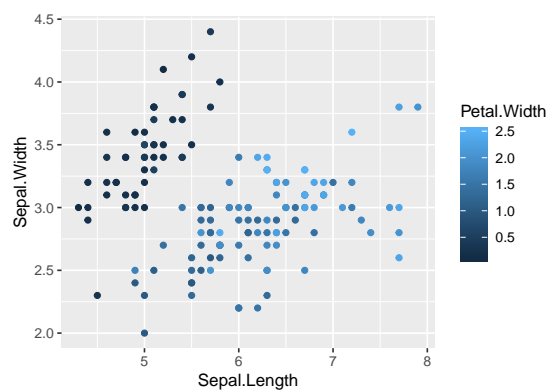
## Couleur avec une variable qualitative

```
> ggplot(iris)+aes(x=Sepal.Length,y=Sepal.Width,
+   color=Species)+geom_point()
```



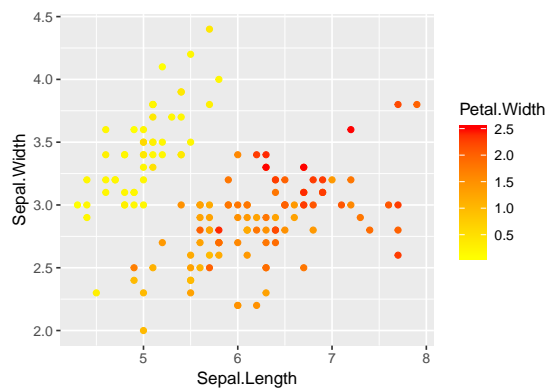
## Couleur avec une variable continue

```
> ggplot(iris)+aes(x=Sepal.Length,y=Sepal.Width,
+   color=Petal.Width)+geom_point()
```



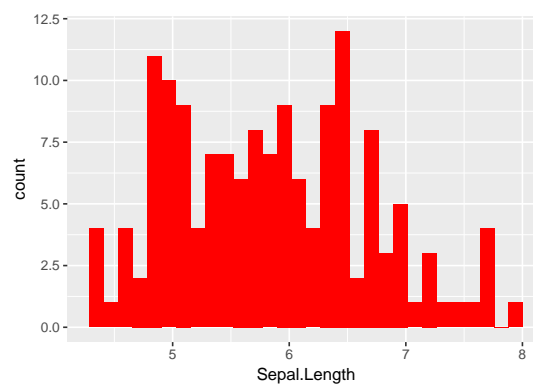
## Changer la couleur

```
> ggplot(iris)+aes(x=Sepal.Length,y=Sepal.Width,
+   color=Petal.Width)+geom_point()+
+   scale_color_continuous(low="yellow",high="red")
```



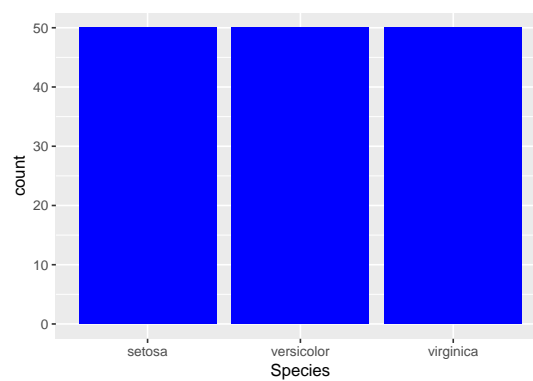
## Histogramme

```
> ggplot(iris)+aes(x=Sepal.Length)+geom_histogram(fill="red")
```



## Diagramme en barres

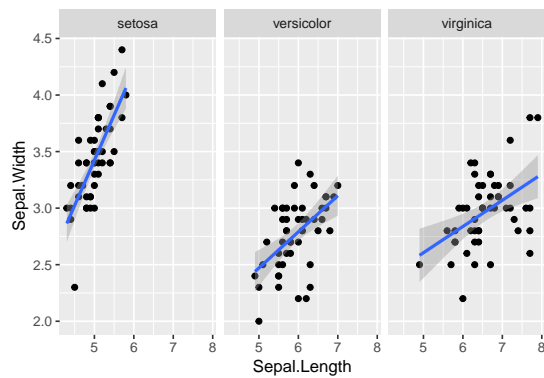
```
> ggplot(iris)+aes(x=Species)+geom_bar(fill="blue")
```



## Facetting (plus compliqué)

```
> ggplot(iris)+aes(x=Sepal.Length,y=Sepal.Width)+geom_point()+
+   geom_smooth(method="lm")+facet_wrap(~Species)
```



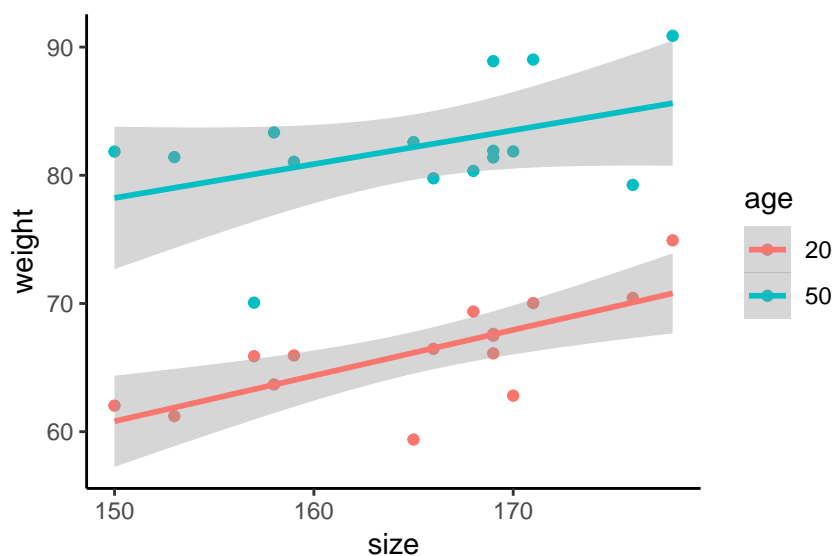


## Combiner ggplot et dplyr

- Souvent important de *construire un bon jeu de données* pour obtenir **un bon graphe**.
- Par exemple

```
> head(df)
## # A tibble: 6 x 3
##   size weight.20 weight.50
##   <dbl>   <dbl>   <dbl>
## 1  153    61.2    81.4
## 2  169    67.5    81.4
## 3  168    69.4    80.3
## 4  169    66.1    81.9
## 5  176    70.4    79.2
## 6  169    67.6    88.9
```

## Objectif



## Etape dplyr

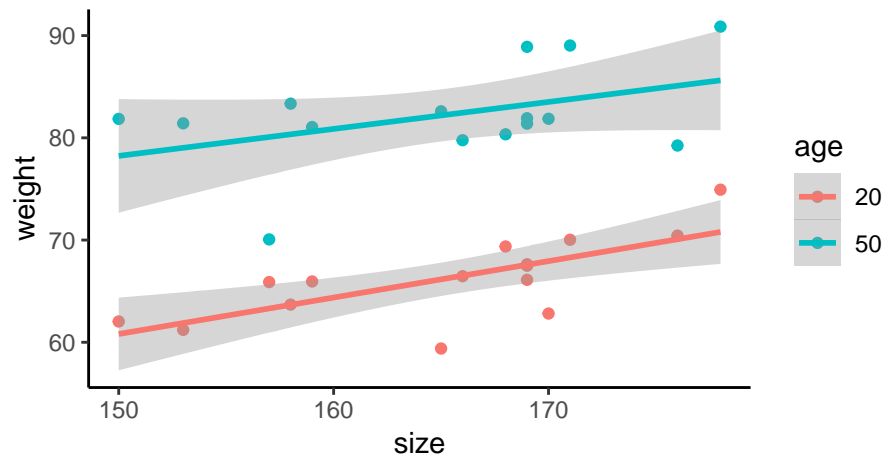
- Assembler les colonnes **weight.M** et **weight.W** en une colonne *weight* :

```
> df1 <- df %>% gather(key=age,value=weight,-size)
> df1 %>% head()
## # A tibble: 6 x 3
##   size age      weight
##   <dbl> <chr>   <dbl>
## 1  153 weight.20  61.2
## 2  169 weight.20  67.5
## 3  168 weight.20  69.4
## 4  169 weight.20  66.1
```

```
## 5 176 weight.20 70.4
## 6 169 weight.20 67.6
> df1 <- df1 %>% mutate(age=recode(age,
+   "weight.20"="20", "weight.50"="50"))
```

## Etape ggplot

```
> ggplot(df1)+aes(x=size,y=weight,color=age)+
+   geom_point()+geom_smooth(method="lm")+theme_classic()
```



⇒ Fiche 4.

## Compléments : quelques démos

```
> demo(image)
> example(contour)
> demo(persp)
> library("lattice");demo(lattice)
> example(wireframe)
> library("rgl");demo(rgl)
> example(persp3d)
> demo(plotmath);demo(Hershey)
```