

Introduction à R

Laurent Rouvière

Septembre 2020

Présentation du cours

- **Prérequis** : bases en programmation, probabilités et statistique.

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.

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
 - **Recherche** : statistique non paramétrique, apprentissage statistique.
 - **Enseignement** : statistique et probabilités (Université, école d'ingénieur, formation continue).
 - **Consulting** : énergie (ERDF), finance, marketing.

- Slides disponibles à l'url https://lrouviere.github.io/intro_R/

- **Slides** disponibles à l'url https://lrouviere.github.io/intro_R/
- **Tutoriel** : compléments de cours et exercices disponible à https://lrouviere.github.io/TUTO_R/

Ressources

- Le **net** : de nombreux tutoriels
- 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 :

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)

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.

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>.

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).

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.

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
```

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

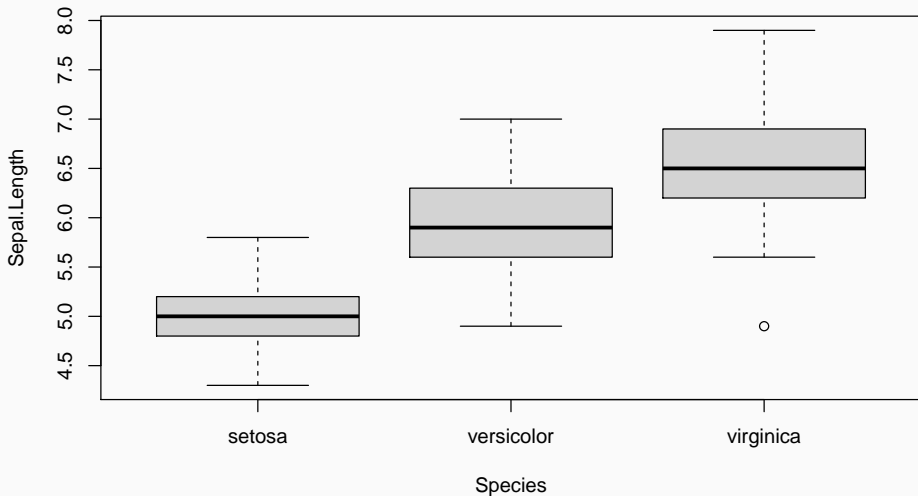
- **dplyr** est un package de **tidyverse** qui permet de faciliter la manipulation des données, notamment en terme de **syntaxe**.

```
> 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**.

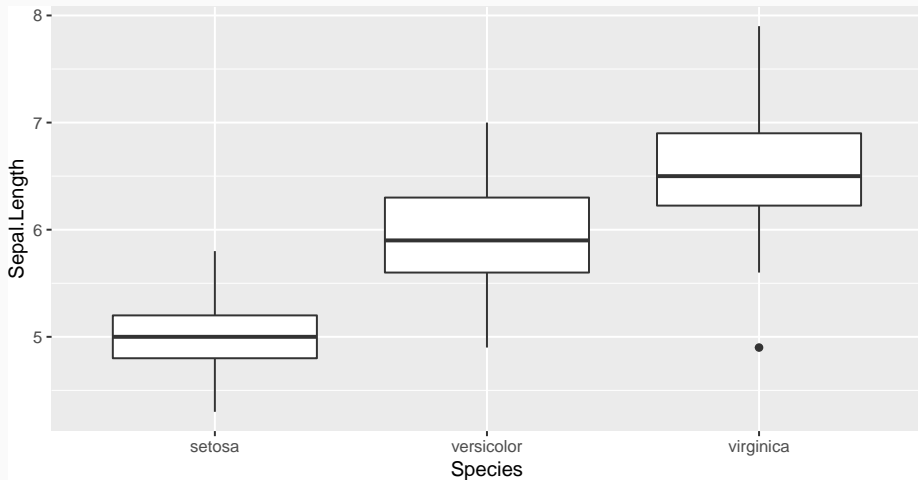
Visualisation

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



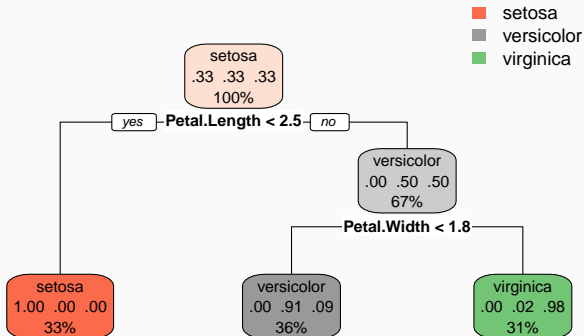
Visualisation avec ggplot2

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



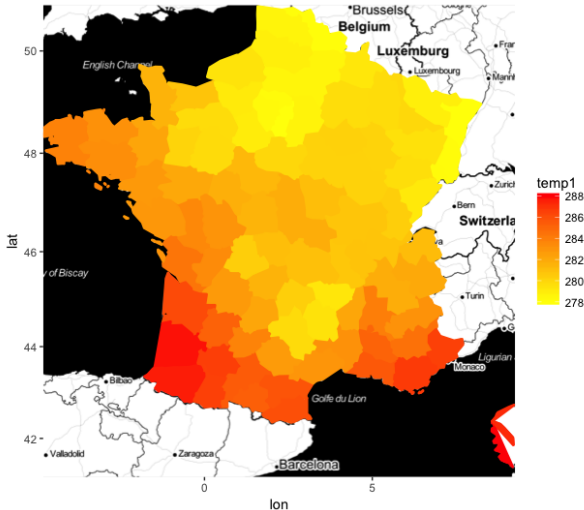
Un modèle d'arbre

```
> 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.

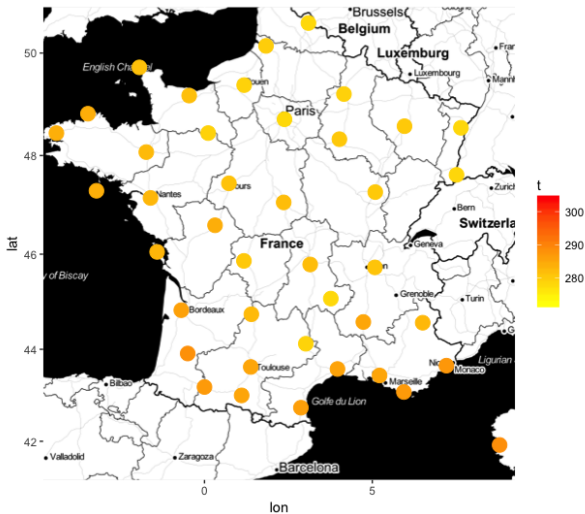


Chargement des données + fond de carte

- Données téléchargées sur le site de meteofrance (températures d'à peu près 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

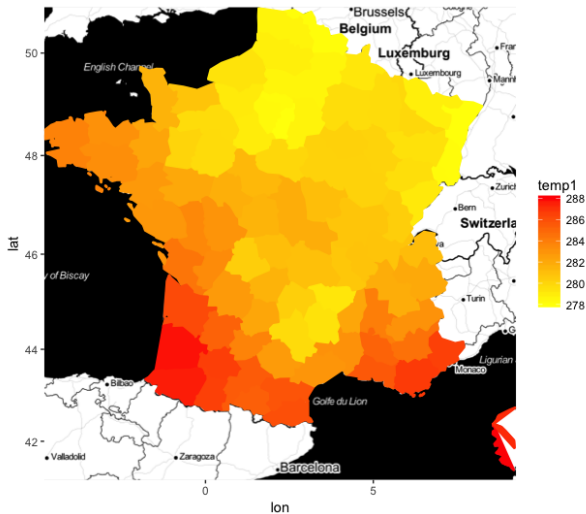
- Algorithme de plus proche voisins pour estimer la température sur toutes les longitudes et latitudes du territoire.

```
> 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



- **Shiny** est un package R qui permet la création de **pages web interactives**.
- **Exemple** : graphiques standards pour un jeu de données :
 - Graphiques descriptifs pour un jeu de données :
https://lrouviere.shinyapps.io/DESC_APP/
 - Visualisation des **stations velib** à Rennes :
<https://lrouviere.shinyapps.io/velib/>

Dans cette partie

- 10 heures pour 4 thèmes :
 - Rstudio et Rmarkdown
 - Objets R
 - Importation et manipulation de données avec dplyr
 - Visualisation de données avec ggplot

Dans cette partie

- 10 heures pour 4 thèmes :
 - Rstudio et Rmarkdown
 - Objets R
 - Importation et manipulation de données avec dplyr
 - Visualisation de données avec ggplot
- 1 thème = slides + Tutoriel (complément de cours + exercices)

Rstudio, Rmarkdown et packages R

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

- **RStudio** est une **interface** facilitant l'utilisation de R.
- Également **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** sous différentes formes (documents, diaporama, etc...).
- Ce diaporama est du **Rmarkdwon**.

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** sous 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 16 000 packages actuellement.
- Contribue au succès de R (toujours à jour).

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 16 000 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).

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 16 000 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.

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 16 000 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.

⇒ Chapitre 1 du tuto.

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"
```

- 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
```

- Vrai ou Faux

```
> 1<2
```

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

```
> 1==2
```

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

```
> 1!=2
```

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

- Vrai ou Faux

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

- Souvent utile pour sélectionner des composantes d'un vecteur

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

```
> size <- runif(5,150,190) #5 tailles générées aléatoirement entre  
> #150 and 190  
> size  
## [1] 178.8362 185.0309 180.4393 185.4450 168.2592
```

Problème

Sélectionner les tailles plus grandes que 174.

```
> size <- runif(5,150,190) #5 tailles générées aléatoirement entre
>                               #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
```


- Pour représenter 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 définie

- 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 |

Variable mal définie

- 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** \implies cela peut générer des **problèmes** dans l'étude statistique.

Variable mal définie

- 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** \implies 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
```

■ Création

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

■ Extraction

```
> m[1,2]
> m[1,]  #Première ligne
> m[,2]  #Seconde colonne
```

Liste

- Permet de regrouper **plusieurs objets** de **différents types** dans un même objet :

```
> 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 |
|----|------------------|-------------|---------------|
| ## | Length:5 | Min. :0.0 | Min. :165.0 |
| ## | Class :character | 1st Qu.:0.0 | 1st Qu.:168.0 |
| ## | Mode :character | 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 |


```
> summary(data)
```

| ## | name | sex | size |
|----|------------------|-------------|---------------|
| ## | Length:5 | Min. :0.0 | Min. :165.0 |
| ## | Class :character | 1st Qu.:0.0 | 1st Qu.:168.0 |
| ## | Mode :character | 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 |

Problème 1

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

Solution

Il faut la convertir en **facteur**.

```
> data$sex <- as.factor(data$sex)
> levels(data$sex) <- c("man", "woman")
> summary(data)
```

| ## | name | sex | size |
|----|------------------|---------|---------------|
| ## | Length:5 | man :3 | Min. :165.0 |
| ## | Class :character | woman:2 | 1st Qu.:168.0 |
| ## | Mode :character | | Median :170.0 |
| ## | | | Mean :171.6 |
| ## | | | 3rd Qu.:175.0 |
| ## | | | Max. :180.0 |

Solution

Il faut la convertir en **facteur**.

```
> data$sex <- as.factor(data$sex)
> levels(data$sex) <- c("man", "woman")
> summary(data)
```

| ## | name | sex | size |
|----|------------------|---------|---------------|
| ## | Length:5 | man :3 | Min. :165.0 |
| ## | Class :character | woman:2 | 1st Qu.:168.0 |
| ## | Mode :character | | Median :170.0 |
| ## | | | Mean :171.6 |
| ## | | | 3rd Qu.:175.0 |
| ## | | | Max. :180.0 |

Problème 2

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
```

| <i>##</i> | <i>sex</i> | <i>size</i> |
|---------------------|--------------|-------------|
| <i>## Paul</i> | <i>man</i> | <i>180</i> |
| <i>## Mary</i> | <i>woman</i> | <i>165</i> |
| <i>## Steven</i> | <i>man</i> | <i>168</i> |
| <i>## Charlotte</i> | <i>woman</i> | <i>170</i> |
| <i>## Peter</i> | <i>man</i> | <i>175</i> |

Conclusion

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

- 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  Length:5
##  1st Qu.:0.0   1st Qu.:168.0   Class :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
```

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).

⇒ Chapitre 2 du tuto.

Gérer des données

Gérer des données

Importer des données

- Les données sont généralement contenues dans des **fichiers** avec les individus en ligne et les variables en colonnes.

- Les données sont généralement contenues dans des **fichiers** 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**.

- Les données sont généralement contenues dans des **fichiers** 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**.

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

- Les données sont généralement contenues dans des **fichiers** 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**.

```
> 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...)

- Les données sont généralement contenues dans des **fichiers** 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**.

```
> 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...)
- Fichiers **.xls** : on pourra les **convertir** en **.csv** ou utiliser des packages spécifiques.

Indiquer le chemin

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

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

Indiquer le chemin

- Le **fichier des données** doit être placé dans le **répertoire de travail**. Sinon, il faut indiquer le **chemin** à **read.table**.
- **Exemple**: 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",...)
```


Indiquer le chemin

- Le **fichier des données** doit être placé dans le **répertoire de travail**. Sinon, il faut indiquer le **chemin** à **read.table**.
- **Exemple**: 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...)

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...)

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

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)

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 identifier les **données manquantes**.
- ...

Exemple

- Fichier `data_imp.txt`

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

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
```

```
##  Length:4
```

```
##  Class :character
```

```
##  Mode  :character
```

Un premier essai

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

```
> df <- read.table(path)
```

```
> summary(df)
```

```
##          V1
```

```
##  Length:4
```

```
##  Class :character
```

```
##  Mode  :character
```

Problème

R lit 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 |

- 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

- L'information utile pour une analyse provient (souvent) de **plusieurs tableaux de données**.

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.

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
```

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
```

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
```

⇒ **Mauvais** 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
```

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.

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.

⇒ Partie 3.1 du tuto.

Gérer des données

Manipuler les données avec Dplyr

- **dplyr** est un package efficace pour **transformer et résumer** des tableaux de données.
- Il propose une **syntaxe claire** (basée sur une **grammaire**) permettant de manipuler les données.

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

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

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

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

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

- La même chose en **dplyr** s'obtient avec

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

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

- **select()** : sélectionner des colonnes (variables)
- **filter()** : filtrer 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
```


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))
```

| | <i>Petal.Length</i> | <i>Petal.Width</i> | <i>diff_petal</i> |
|------|---------------------|--------------------|-------------------|
| ## 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
```

Summarise_all et summarise_at

On peut également calculer des résumés pour des groupes de variables :

- `summarize_all` : toutes les variables du tibble

```
> iris1 <- select(iris, -Species)
> summarise_all(iris1, mean)
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1      5.843333    3.057333         3.758      1.199333
```

Summarise_all et summarise_at

On peut également calculer des résumés pour des groupes de variables :

- `summarize_all` : toutes les variables du tibble

```
> iris1 <- select(iris, -Species)
> summarise_all(iris1, mean)
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1      5.843333    3.057333         3.758      1.199333
```

- `summarize_at` : choisir les variables du tibble

```
> summarise_at(iris, 1:3, mean)
##   Sepal.Length Sepal.Width Petal.Length
## 1      5.843333    3.057333         3.758
```

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'enchâîner les commandes pour une syntaxe plus claire.
- 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
```


ou (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
```

ou (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
```

Pas satisfaisant

Création de deux objets **dataframe** (inutiles) pour un calcul "simple".

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

1. Les données

```
> iris
```

- Avec le **pipe**, on **décompose** et **enchaîne** 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)
```

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,

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
```


Reformater les données

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

```
> df <- iris %>% group_by(Species) %>%  
+   summarize_all(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
```

pivot_longer

- **Assembler** des colonnes en lignes avec `\alert{pivot_longer}` (anciennement `gather`) :

```
> df1 <- df %>% pivot_longer(-Species,names_to="variable",values_to="va.  
> head(df1)  
## # A tibble: 6 x 3  
##   Species      variable      valeur  
##   <fct>      <chr>      <dbl>  
## 1 setosa     Sepal.Length  5.01  
## 2 setosa     Sepal.Width   3.43  
## 3 setosa     Petal.Length  1.46  
## 4 setosa     Petal.Width   0.246  
## 5 versicolor Sepal.Length  5.94  
## 6 versicolor Sepal.Width   2.77
```

Remarque

Même information avec un format différent.

- Décomposer une ligne en plusieurs colonnes avec `pivot_wider` (anciennement `spread`).

```
> df1 %>% pivot_wider(names_from=variable, values_from=valeur)
## # 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
```

- 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))
```

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
```

- 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
```

- 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
```

⇒ Partie 3.2 du tuto.

Visualiser des données

Visualiser des données

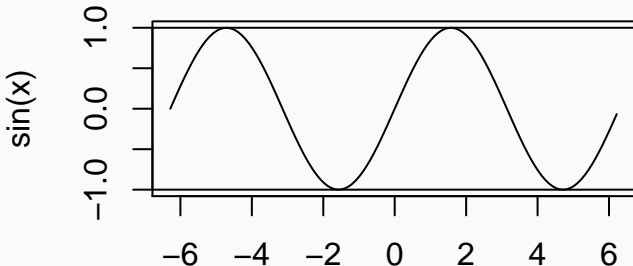
Graphes conventionnels

- **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) présentation 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.

Graphes classiques pour visualiser des variables

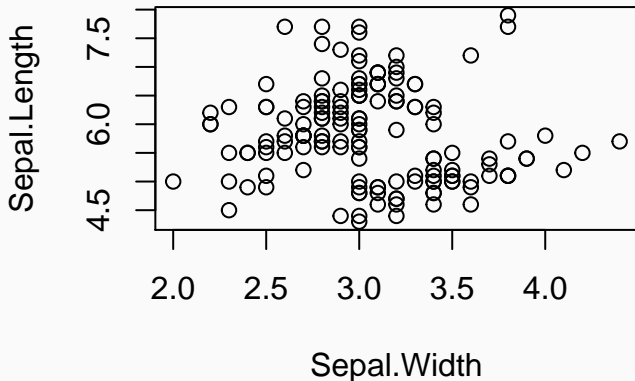
- **Histogramme** pour une variable **continue**, **diagramme en barre** pour une variable **qualitative**.
- **Nuage de points** pour 2 variables continues.
- **Boxplot** pour une distribution continue.

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)
```



```
> #pareil que
```

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

Histogramme (variable continue)

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

Histogram of iris\$Sepal.Length

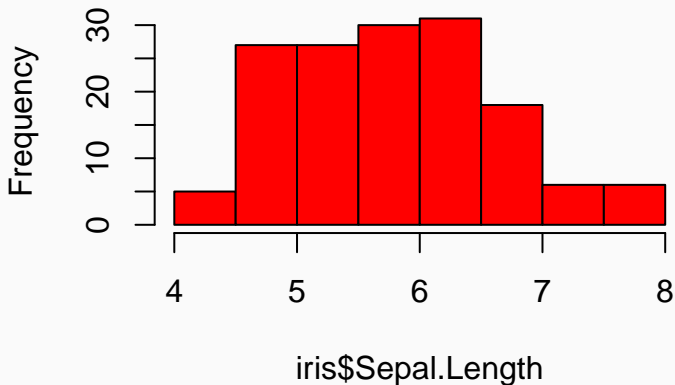
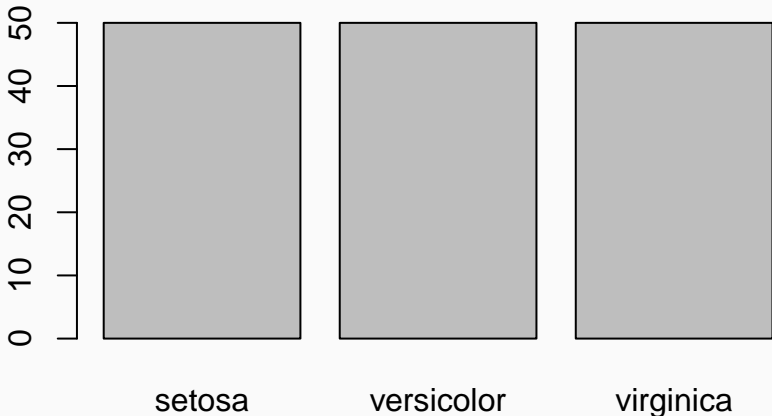


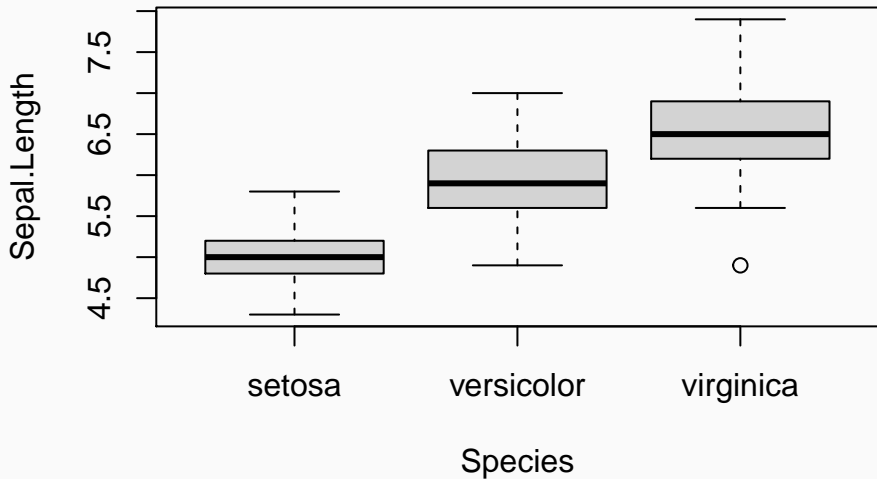
Diagramme en barres (variable qualitative)

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



Boxplot (distribution)

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



Visualiser des données

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 **de très bonnes qualités** (pas toujours le cas avec les graphes conventionnels).
- La **grammaire ggplot** permet d'obtenir des **graphes "complexes"** avec une **syntaxe claire et lisible**.

Assembler des couches

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. . .).

Assembler des couches

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_**

Les principaux **verbes** sont

- **Data (ggplot)** : les **données**, un dataframe ou un tibble.

La grammaire

Les principaux **verbes** sont

- **Data (ggplot)** : les **données**, un dataframe ou un tibble.
- **Aesthetics (aes)** : façon dont les **variables** doivent être représentées.

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.

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.

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...).

La grammaire

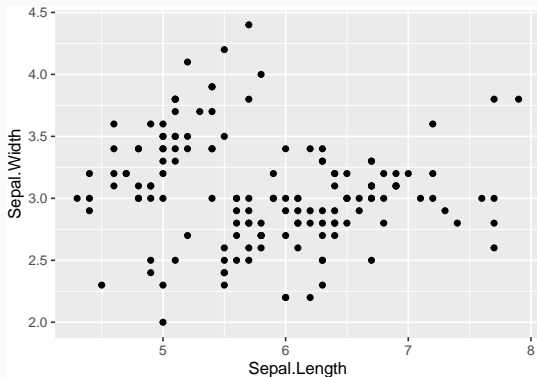
Les principaux **verbes** sont

- **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 +**.

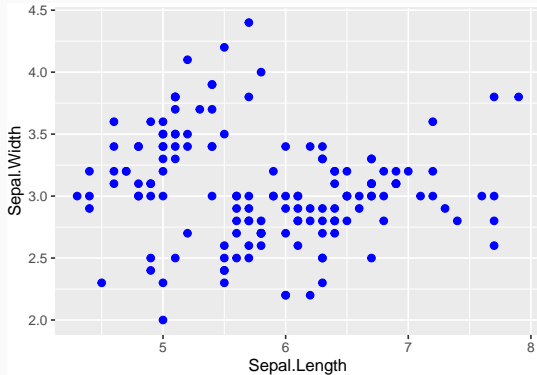
Un premier exemple

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



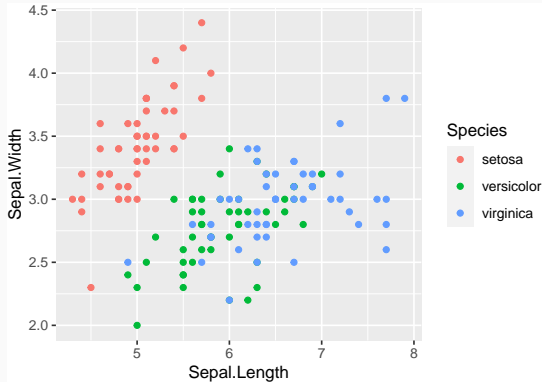
Couleur et taille

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



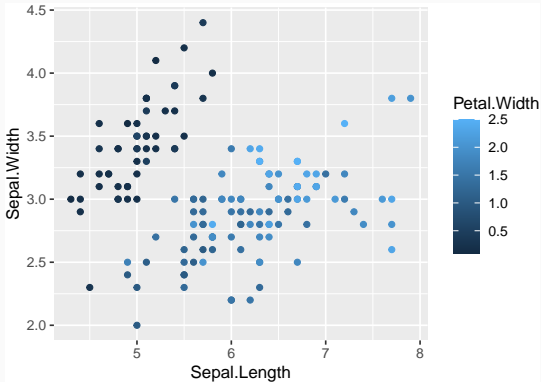
Couleur avec une variable qualitative

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



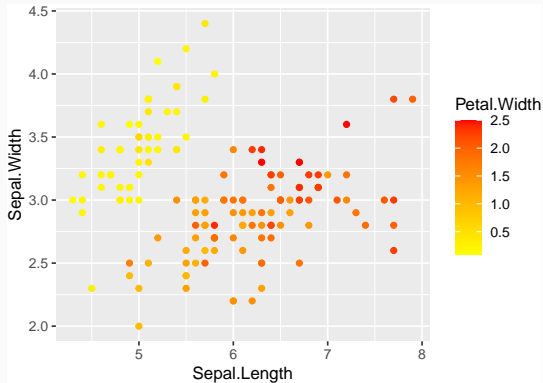
Couleur avec une variable continue

```
> 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")
```

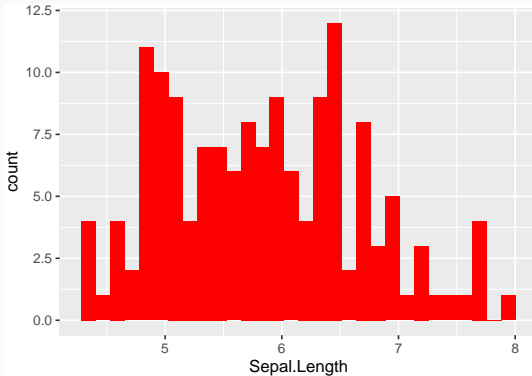
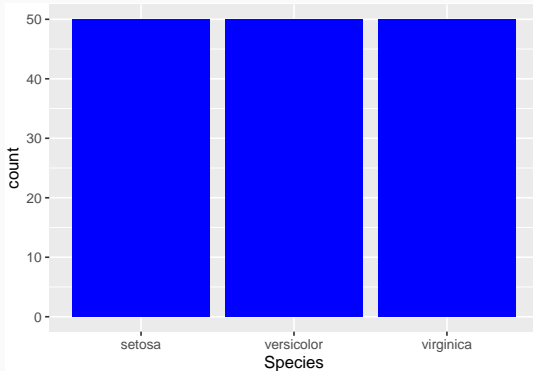


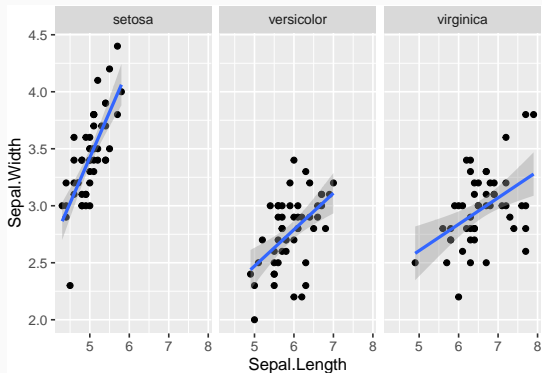
Diagramme en barres

```
> 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)
```

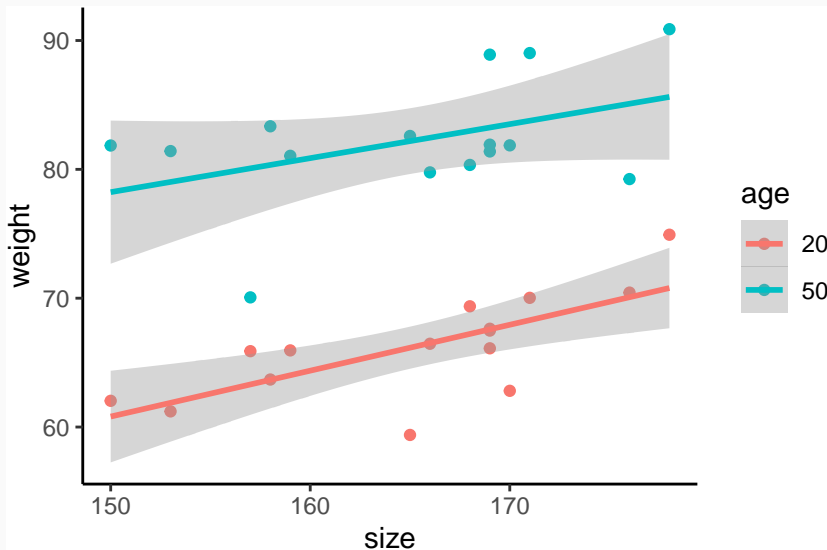


Combiner ggplot et dplyr

- 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



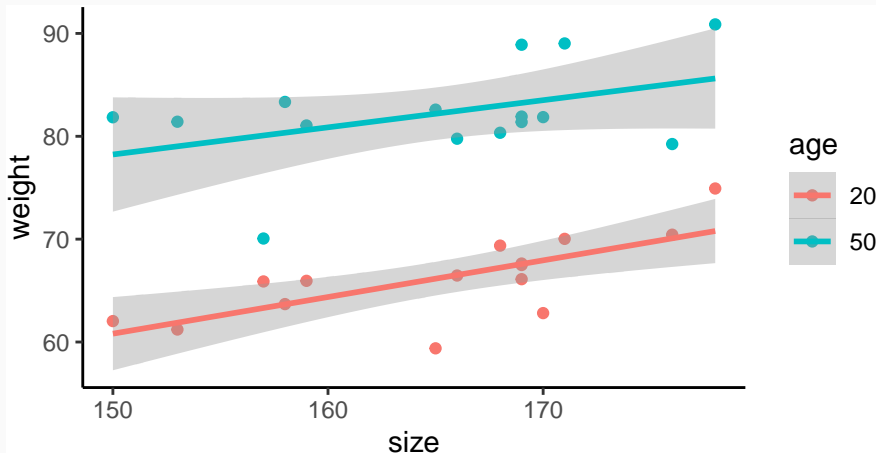
Etape dplyr

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

```
> df1 <- df %>% pivot_longer(-size,names_to="age",values_to="weight")
> df1 %>% head()
## # A tibble: 6 x 3
##   size age      weight
##   <dbl> <chr>    <dbl>
## 1   153 weight.20    61.2
## 2   153 weight.50    81.4
## 3   169 weight.20    67.5
## 4   169 weight.50    81.4
## 5   168 weight.20    69.4
## 6   168 weight.50    80.3
> 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()
```



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)
```


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)
```

⇒ Chapitre 4 du tuto.

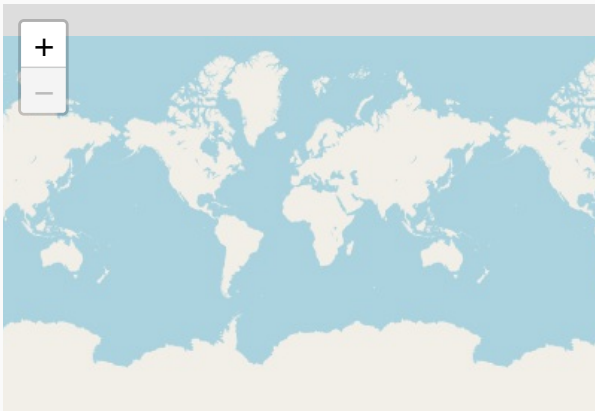
Cartes leaflet

- De nombreuses applications nécessitent des **cartes** pour **visualiser** des **données** ou les résultats d'un **modèle**.
- De **nombreux packages R** : ggmap, RgoogleMaps, maps. . .
- Dans cette partie : **leaflet**.

Fond de carte

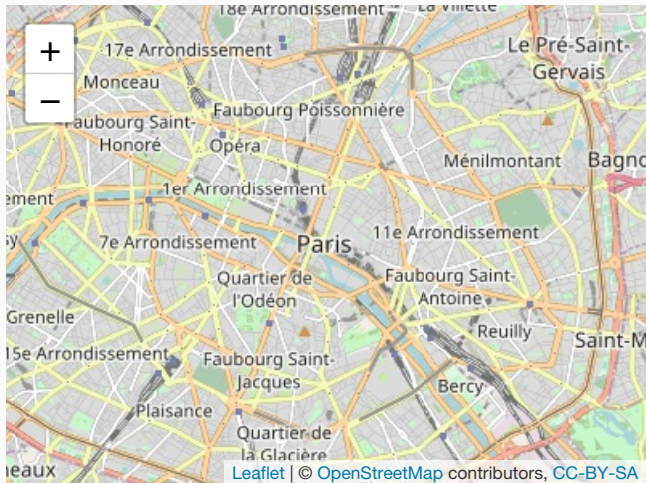
- Leaflet est une des librairies open-source JavaScript les plus populaires pour faire des cartes interactives.
- Documentation: [here](#)

```
> library(leaflet)  
> leaflet() %>% addTiles()
```



Différents styles de fonds de carte

```
> Paris <- c(2.35222,48.856614)
> leaflet() %>% addTiles() %>%
+   setView(lng = Paris[1], lat = Paris[2],zoom=12)
```



```
> leaflet() %>% addProviderTiles("Stamen.Toner") %>%  
+   setView(lng = Paris[1], lat = Paris[2], zoom = 12)
```



- Localiser 1000 séismes près des Fiji

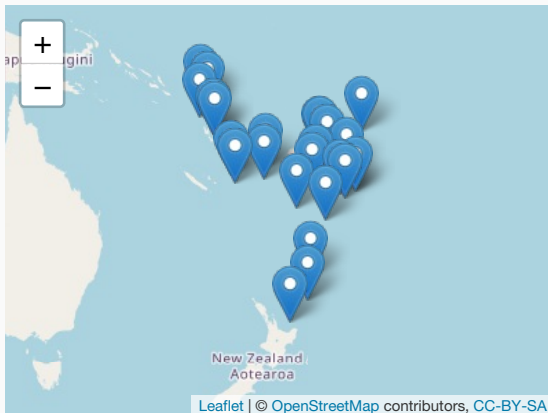
```
> data(quakes)
```

```
> head(quakes)
```

| ## | lat | long | depth | mag | stations |
|------|--------|--------|-------|-----|----------|
| ## 1 | -20.42 | 181.62 | 562 | 4.8 | 41 |
| ## 2 | -20.62 | 181.03 | 650 | 4.2 | 15 |
| ## 3 | -26.00 | 184.10 | 42 | 5.4 | 43 |
| ## 4 | -17.97 | 181.66 | 626 | 4.1 | 19 |
| ## 5 | -20.42 | 181.96 | 649 | 4.0 | 11 |
| ## 6 | -19.68 | 184.31 | 195 | 4.0 | 12 |

Séismes avec une magnitude plus grande que 5.5

```
> quakes1 <- quakes %>% filter(mag>5.5)
> leaflet(data = quakes1) %>% addTiles() %>%
+   addMarkers(~long, ~lat, popup = ~as.character(mag))
```

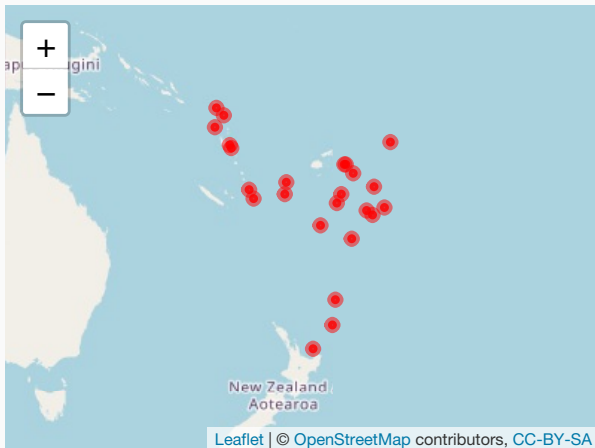


Remarque

La magnitude apparaît lorsqu'on clique sur un marker.

addCircleMarkers

```
> leaflet(data = quakes1) %>% addTiles() %>%  
+   addCircleMarkers(~long, ~lat, popup=~as.character(mag),  
+                   radius=3,fillOpacity = 0.8,color="red")
```



Modèle de régression avec R

| Y | X_1 | X_2 | \dots | X_p |
|----------|-----------|-----------|----------|-----------|
| y_1 | $x_{1,1}$ | $x_{1,2}$ | \dots | $x_{1,p}$ |
| \vdots | \vdots | \vdots | \vdots | \vdots |
| \vdots | \vdots | \vdots | \vdots | \vdots |
| y_n | $x_{n,1}$ | $x_{n,2}$ | \dots | $x_{n,p}$ |

But

Expliquer ou prédire la **sortie** Y par les **entrées** X_1, \dots, X_p .

Exemple : ozone

```
> ozone <- read.table("ozone.txt")
> head(ozone %>% select(1:5))
```

| ## | maxO3 | T9 | T12 | T15 | Ne9 |
|-------------|-------|------|------|------|-----|
| ## 20010601 | 87 | 15.6 | 18.5 | 18.4 | 4 |
| ## 20010602 | 82 | 17.0 | 18.4 | 17.7 | 5 |
| ## 20010603 | 92 | 15.3 | 17.6 | 19.5 | 2 |
| ## 20010604 | 114 | 16.2 | 19.7 | 22.5 | 1 |
| ## 20010605 | 94 | 17.4 | 20.5 | 20.4 | 8 |
| ## 20010606 | 80 | 17.7 | 19.8 | 18.3 | 6 |

But

Expliquer ou prédire la **concentration maximale quotidienne en O3** (colonne maxO3) par les autres variables.

- Il existe une fonction **inconnue** $m : \mathbb{R}^p \rightarrow \mathbb{R}$ telle que

$$Y = m(X_1, \dots, X_p) + \varepsilon.$$

- ε : termes d'erreur (petits).

- Il existe une fonction **inconnue** $m : \mathbb{R}^p \rightarrow \mathbb{R}$ telle que

$$Y = m(X_1, \dots, X_p) + \varepsilon.$$

- ε : termes d'erreur (petits).
- **Job du statisticien** : trouver un bon estimateur \hat{m} de m à partir des données $(x_1, y_1), \dots, (x_n, y_n)$ où $x_i \in \mathbb{R}^p$ et $y_i \in \mathbb{R}$.

Modèle statistique

Permet de construire des estimateurs.

Un exemple : le modèle linéaire

- **Hypothèse** : la fonction inconnue m est **linéaire**

$$Y = \beta_0 + \beta_1 X_1 + \dots + \beta_p X_p + \varepsilon,$$

$\beta = (\beta_0, \beta_1, \dots, \beta_p)$ sont les paramètres **inconnus**.

Un exemple : le modèle linéaire

- **Hypothèse** : la fonction inconnue m est **linéaire**

$$Y = \beta_0 + \beta_1 X_1 + \dots + \beta_p X_p + \varepsilon,$$

$\beta = (\beta_0, \beta_1, \dots, \beta_p)$ sont les paramètres **inconnus**.

- **Moindres carrés** :

$$\hat{\beta} = (X^t X)^{-1} X^t Y.$$

- **Estimateur** de m :

$$\hat{m}(x) = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \dots + \hat{\beta}_p x_p.$$

- Les modèles sur **R** sont souvent entraînés de la même façon :

```
> method(formula,data=...,options)
```

avec

- **method** : nom de la méthode ;
- **formula** : sortie Y et les entrées X_j ;
- **data** : jeu de données ;
- **options** : options en fonction de la méthode.

La méthode (ou le modèle)

Remarque

Chaque modèle correspond à un **fonction R**.

| fonction R | algorithmes | Package | Problème |
|----------------|--------------------------------|---------|---------------|
| lm | modèle linéaire | | Reg |
| glm | modèle logistique | | Class |
| lda | analyse discriminante linéaire | MASS | Class |
| svm | Support Vector Machine | e1071 | Class |
| knn.reg | plus proches voisins | FNN | Reg |
| knn | plus proches voisins | class | Class |
| rpart | arbres | rpart | Reg and Class |
| glmnet | ridge et lasso | glmnet | Reg and Class |

Remarque

Pour spécifier les entrées et la sortie.

```
> lm(Y~X1+X3,data=df)
```

Remarque

Pour spécifier les entrées et la sortie.

```
> lm(Y~X1+X3,data=df)
```

$$\Rightarrow Y = \beta_0 + \beta_1 X_1 + \beta_3 X_3 + \varepsilon$$

Remarque

Pour spécifier les entrées et la sortie.

```
> lm(Y~X1+X3,data=df)
```

$$\Rightarrow Y = \beta_0 + \beta_1 X_1 + \beta_3 X_3 + \varepsilon$$

```
> lm(Y~X1+I(X3)^2,data=df)
```

Remarque

Pour spécifier les entrées et la sortie.

```
> lm(Y~X1+X3,data=df)
```

$$\Rightarrow Y = \beta_0 + \beta_1 X_1 + \beta_3 X_3 + \varepsilon$$

```
> lm(Y~X1+I(X3)^2,data=df)
```

$$\Rightarrow Y = \beta_0 + \beta_1 X_1 + \beta_3 X_3^2 + \varepsilon$$

Remarque

Pour spécifier les entrées et la sortie.

```
> lm(Y~X1+X3,data=df)
```

$$\Rightarrow Y = \beta_0 + \beta_1 X_1 + \beta_3 X_3 + \varepsilon$$

```
> lm(Y~X1+I(X3)^2,data=df)
```

$$\Rightarrow Y = \beta_0 + \beta_1 X_1 + \beta_3 X_3^2 + \varepsilon$$

```
> lm(Y~.,data=df)
```

Remarque

Pour spécifier les entrées et la sortie.

```
> lm(Y~X1+X3,data=df)
```

$$\Rightarrow Y = \beta_0 + \beta_1 X_1 + \beta_3 X_3 + \varepsilon$$

```
> lm(Y~X1+I(X3)^2,data=df)
```

$$\Rightarrow Y = \beta_0 + \beta_1 X_1 + \beta_3 X_3^2 + \varepsilon$$

```
> lm(Y~.,data=df)
```

$$\Rightarrow Y = \beta_0 + \beta_1 X_1 + \dots + \beta_p X_p + \varepsilon$$

Example

```
> mod.lin <- lm(maxO3~T12+Ne9,data=ozone)
> mod.lin
##
## Call:
## lm(formula = maxO3 ~ T12 + Ne9, data = ozone)
##
## Coefficients:
## (Intercept)          T12           Ne9
##          7.638         4.457        -2.696
```

Exemple

```
> mod.lin <- lm(maxO3~T12+Ne9,data=ozone)
> mod.lin
##
## Call:
## lm(formula = maxO3 ~ T12 + Ne9, data = ozone)
##
## Coefficients:
## (Intercept)          T12           Ne9
##      7.638       4.457      -2.696
```

- **Modèle** : $\max O3 = \beta_0 + \beta_1 T12 + \beta_2 Ne9 + \varepsilon$.
- **Estimateurs** : $\hat{\beta}_0 = 7.638, \hat{\beta}_1 = 4.457, \hat{\beta}_2 = -2.696$.

Exemple

```
> mod.lin <- lm(maxO3~T12+Ne9,data=ozone)
> mod.lin
##
## Call:
## lm(formula = maxO3 ~ T12 + Ne9, data = ozone)
##
## Coefficients:
## (Intercept)          T12           Ne9
##      7.638      4.457     -2.696
```

- **Modèle** : $\max O3 = \beta_0 + \beta_1 T12 + \beta_2 Ne9 + \varepsilon$.
- **Estimateurs** : $\hat{\beta}_0 = 7.638, \hat{\beta}_1 = 4.457, \hat{\beta}_2 = -2.696$.

Estimateur de m

$$\hat{m}(x) = 7.638 + 4.457 T12 - 2.696 Ne9.$$

- Une fois le modèle ajusté, on peut l'utiliser pour **prédire**.

Exemple

- Météofrance prédit pour demain: $T_{12}=20$ et $Ne_9=4.9$.
- Concentration en ozone prédite par le modèle pour demain ?

Faire des prévisions

- Une fois le modèle ajusté, on peut l'utiliser pour **prédire**.

Exemple

- Météofrance prédit pour demain: T12=20 et Ne9=4.9.
- Concentration en ozone prédite par le modèle pour demain ?

- **Réponse :**

$$\hat{m}(T12 = 20, Ne9 = 4.9) = 7.638 + 4.457 * 20 - 2.696 * 4.9 = 83.5676$$

.

Fonction predict

- `predict` est une fonction **générique** : on peut l'utiliser pour n'importe quel modèle de régression (linéaire, logistique, arbre...)

```
> predict(model.name, newdata=newdataset, ...)
```

Fonction predict

- `predict` est une fonction **générique** : on peut l'utiliser pour n'importe quel modèle de régression (linéaire, logistique, arbre...)

```
> predict(model.name, newdata=newdataset, ...)
```

- **Exemple**

```
> new.df <- data.frame(T12=20, Ne9=4.9)
> predict(mod.lin, newdata=new.df)
##           1
## 83.57509
```

Très important

Utiliser la **même structure** pour les 2 data-frames.

Estimer l'erreur quadratique de prédiction

- La performance d'un estimateur \hat{m} est souvent mesurée par son **erreur quadratique moyenne**:

$$MSE(\hat{m}) = E[(Y - \hat{m}(X))^2].$$

Estimer l'erreur quadratique de prédiction

- La performance d'un estimateur \hat{m} est souvent mesurée par son **erreur quadratique moyenne**:

$$MSE(\hat{m}) = E[(Y - \hat{m}(X))^2].$$

- Cette erreur (**inconnue**) peut être calculée par **validation hold out** :
 - Séparer les données en un échantillon d'**apprentissage** et un échantillon **test**.
 - Entraîner le modèle sur les données d'apprentissage $\Rightarrow \hat{m}$.
 - Calculer la MSE

$$\frac{1}{n_{test}} \sum_{i \in test} (y_i - \hat{m}(x_i))^2.$$

Un exemple

- Data splitting

```
> library(caret)
> set.seed(12345)
> index.train <- createDataPartition(1:nrow(ozone),p=2/3)
> train <- ozone %>% slice(index.train$Resample1)
> test <- ozone %>% slice(-index.train$Resample1)
```

Un exemple

- Data splitting

```
> library(caret)
> set.seed(12345)
> index.train <- createDataPartition(1:nrow(ozone),p=2/3)
> train <- ozone %>% slice(index.train$Resample1)
> test <- ozone %>% slice(-index.train$Resample1)
```

- Ajustement du modèle

```
> mod <- lm(maxO3~.,data=train)
```

Un exemple

- Data splitting

```
> library(caret)
> set.seed(12345)
> index.train <- createDataPartition(1:nrow(ozone),p=2/3)
> train <- ozone %>% slice(index.train$Resample1)
> test <- ozone %>% slice(-index.train$Resample1)
```

- Ajustement du modèle

```
> mod <- lm(maxO3~.,data=train)
```

- Calcul de la MSE

```
> pred <- predict(mod,newdata=test)
> df <- data.frame(pred=pred,obs=test$maxO3)
> df %>% summarize(MSE=mean((pred-obs)^2))
##           MSE
## 1 387.5472
```

- Très utile pour choisir un modèle.
- **Exemple** : plusieurs modèles (linéaire, arbre, forêt aléatoire. . .)

Méthode

1. Estimer la MSE pour tous les algorithmes ;
2. Choisir celui avec la plus petite MSE.

- Très utile pour choisir un modèle.
- **Exemple** : plusieurs modèles (linéaire, arbre, forêt aléatoire. . .)

Méthode

1. Estimer la MSE pour tous les algorithmes ;
2. Choisir celui avec la plus petite MSE.

⇒ **fiche 6.**

Merci