

# Visualisation avec R

---

Laurent Rouvière

2022-01-12

# Présentation

- **Prérequis** : niveau avancé en **R** - bases en statistique et programmation
- **Objectifs** :
  - Comprendre l'importance de la visualisation en **science des données**
  - Visualiser des **données**, des **modèles** et des **résultats**
  - Découvrir quelques packages de visualisation en **R**

# Présentation

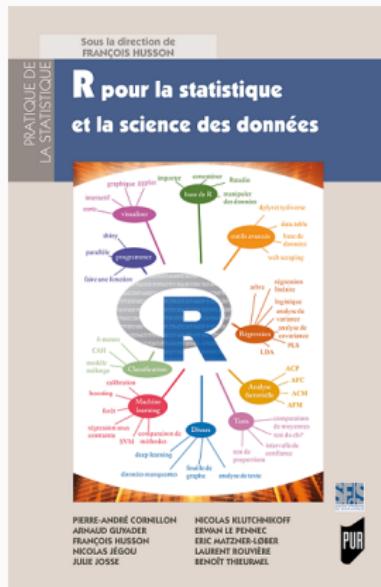
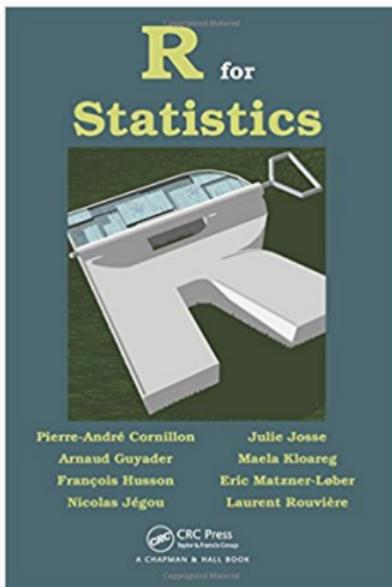
- **Prérequis** : niveau avancé en **R** - bases en statistique et programmation
- **Objectifs** :
  - Comprendre l'importance de la visualisation en **science des données**
  - Visualiser des **données**, des **modèles** et des **résultats**
  - Découvrir quelques packages de visualisation en **R**
- **Enseignant** : Laurent Rouvière, [laurent.rouviere@univ-rennes2.fr](mailto:laurent.rouviere@univ-rennes2.fr)
  - **Recherche** : statistique non paramétrique, apprentissage statistique
  - **Enseignement** : probabilités et statistique (Université, écodes, formation continue)
  - **Consulting**: énergie (ERDF), finance, marketing, sport

# Ressources

- Slides et tutoriel (compléments de cours + exercices) disponibles à <https://lrouvriere.github.io/VISU/>
- Le web

## Ressources

- Slides et tutoriel (compléments de cours + exercices) disponibles à <https://lrouviere.github.io/VISU/>
  - Le web
  - Livres



# Pourquoi un cours de visualisation ?

- **Données** de plus en plus **complexes**
- **Modèles** de plus en plus **complexes**
- **Interprétations** des résultats de plus en plus **complexes.**

# Pourquoi un cours de visualisation ?

- **Données** de plus en plus **complexes**
- **Modèles** de plus en plus **complexes**
- **Interprétations** des résultats de plus en plus **complexes**.
- Besoin de visualiser pour :
  - **décrire** les données
  - **calibrer** les modèles
  - **présenter** les résultats de l'étude.

## Conséquence

- La visualisation se révèle **cruciale** tout au long d'une étude statistique.
- De plus en plus de packages R sont dédiés à la **visualisation**.

# Plan

- (au moins) 2 façons d'appréhender la **visualisation** :
  1. **Méthodes/modèles statistiques** : PCA, LDA, arbres...
  2. **Outils informatique** : packages R.

# Plan

- (au moins) 2 façons d'appréhender la **visualisation** :
  1. **Méthodes/modèles statistiques** : PCA, LDA, arbres...
  2. **Outils informatique** : packages R.
- Dans ce cours, on va présenter quelques **outils R** :
  1. **ggplot2** : un package R pour visualiser les données  $\Rightarrow$  3-4h.
  2. **Cartes** avec **ggmap**, **sf** et **leaflet**  $\Rightarrow$  3-4h.
  3. **Visualisation dynamique/intéractive**
    - données avec **rAmCharts** et **Plotly**  $\Rightarrow$  1h.
    - tableaux de bord avec **flexdashboard**  $\Rightarrow$  1h.
    - application web avec **shiny**  $\Rightarrow$  3-4h.

## Compléments

Workshop shiny en février.

## Boxplot sur les iris

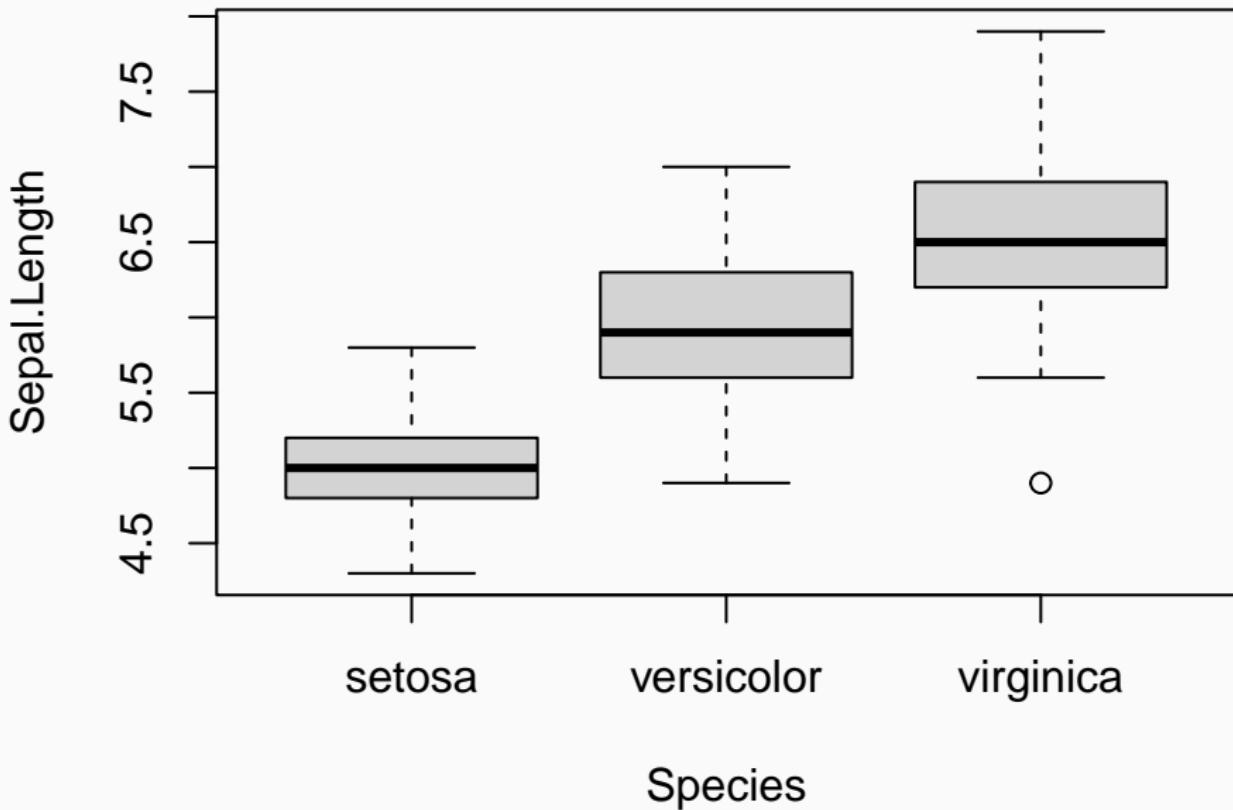
```
> data(iris)
> summary(iris)

  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  
```

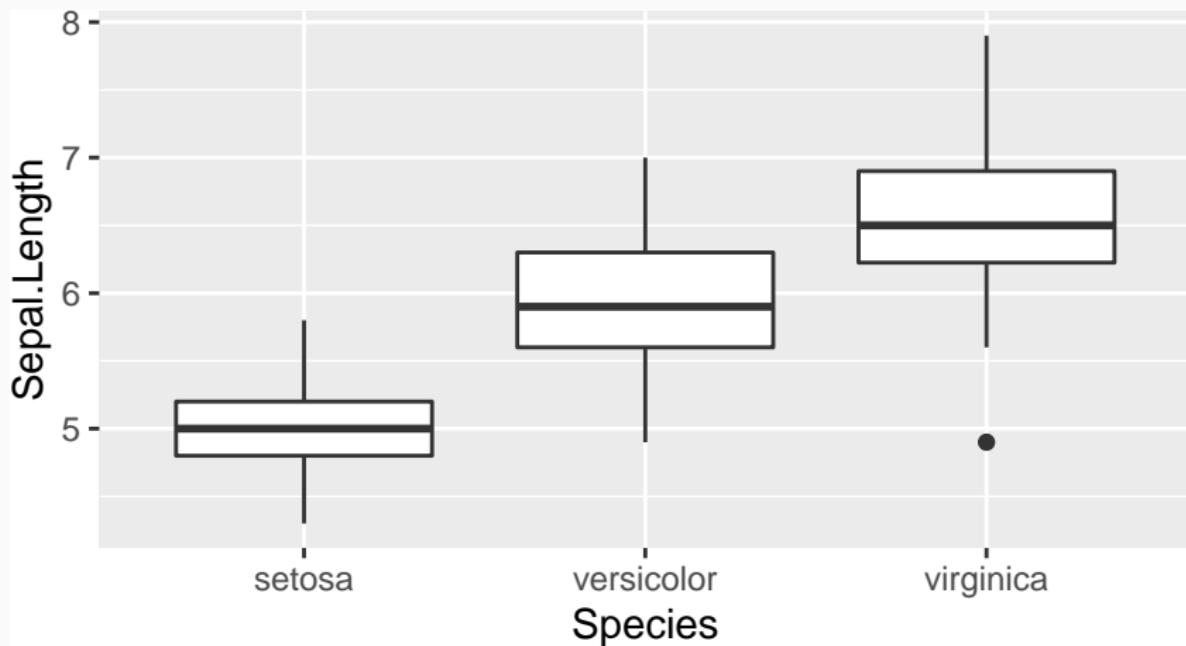
## Fonctions conventionnelles

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

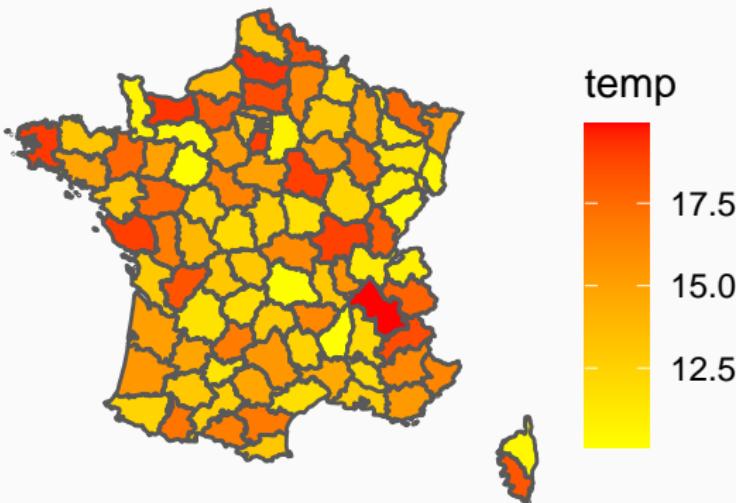


# Grammaire ggplot

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



# Une carte des températures



## Diverses informations

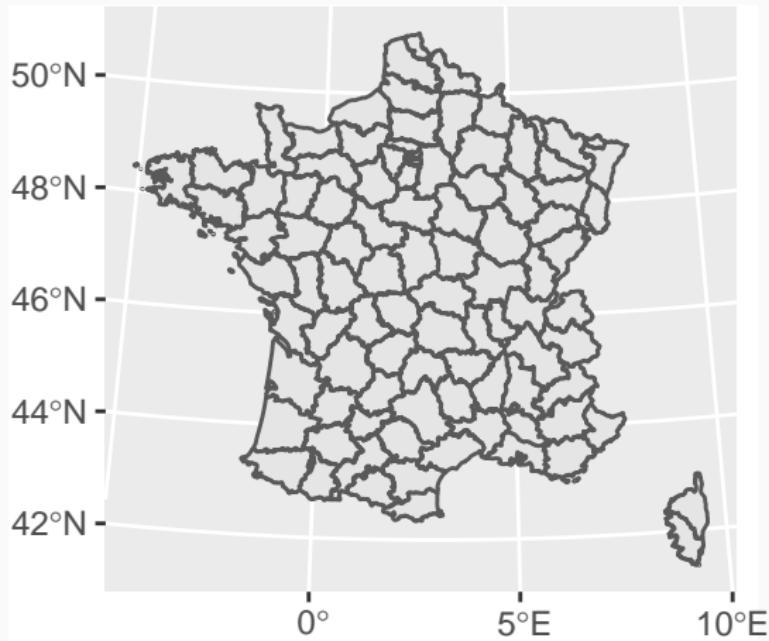
- Fond de cartes avec les frontières des départements ;
- Températures observées dans les départements (site web de météo france).

# Carte shapefile

```
> library(sf)
> dpt <- read_sf("./DATA/dpt")
> dpt %>% select(NOM_DEPT,geometry) %>% head()
Simple feature collection with 6 features and 1 field
Geometry type: MULTIPOLYGON
Dimension:      XY
Bounding box:  xmin: 644570 ymin: 6272482 xmax: 1077507 ymax: 6997000
Projected CRS: RGF93 / Lambert-93
# A tibble: 6 x 2
  NOM_DEPT                         geometry
  <chr>                               <MULTIPOLYGON [m]>
1 AIN        (((919195 6541470, 918932 6541203, 918628 6~)
2 AISNE      (((735603 6861428, 735234 6861392, 734504 6~)
3 ALLIER     (((753769 6537043, 753554 6537318, 752879 6~)
4 ALPES-DE-HAUTE-PROVENCE (((992638 6305621, 992263 6305688, 991610 6~)
5 HAUTES-ALPES   (((1012913 6402904, 1012577 6402759, 101085~)
6 ALPES-MARITIMES (((1018256 6272482, 1017888 6272559, 101677~
```

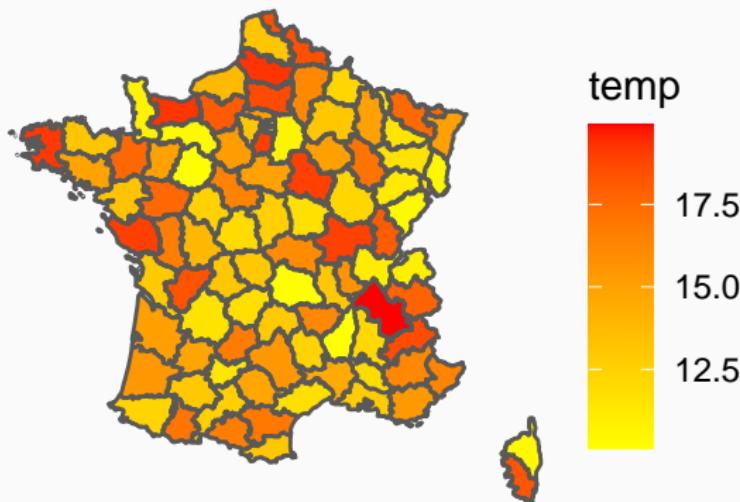
## Fond de carte

```
> ggplot(dpt)+geom_sf()
```



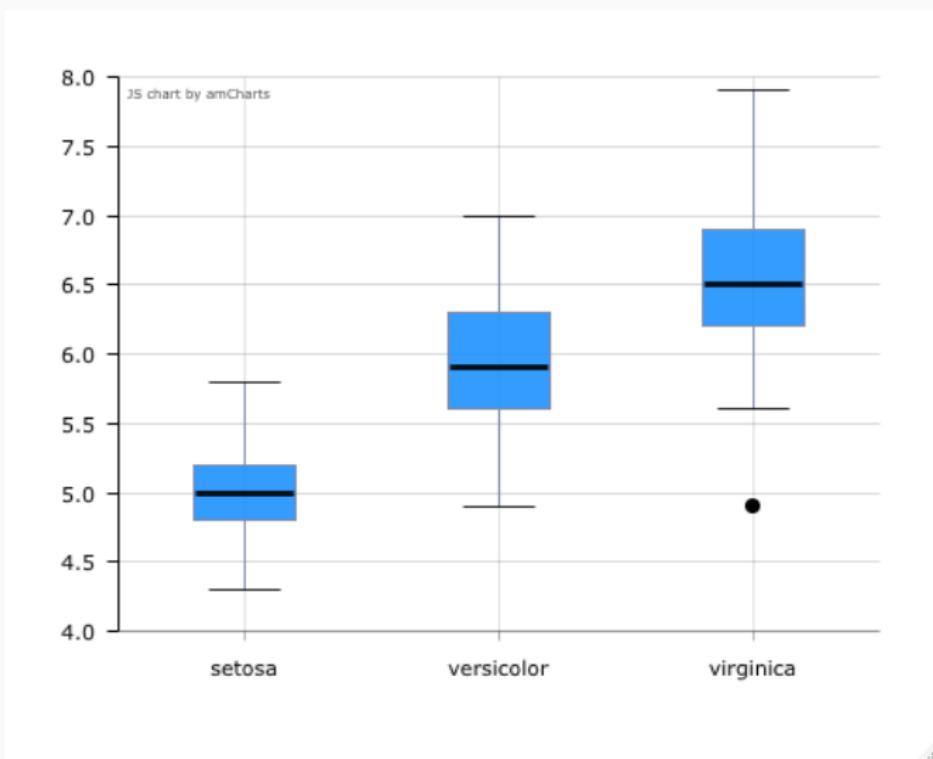
## Ajout des températures

```
> ggplot(dpt) + geom_sf(aes(fill=temp)) +
+   scale_fill_continuous(low="yellow",high="red")+
+   theme_void()
```



# Graphes interactifs avec rAmCharts

```
> library(rAmCharts)  
> amBoxplot(Sepal.Length~Species,data=iris)
```



# Tableaux de bord

- utile pour **publier** des synthèses d'**outils de visualisation** (données, graphes, modèles simples...)
- Package **flexdashboard**:  
<https://rmarkdown.rstudio.com/flexdashboard/index.html>

## Tableaux de bord

- utile pour publier des synthèses d'outils de visualisation (données, graphes, modèles simples...)
- Package flexdashboard:  
<https://rmarkdown.rstudio.com/flexdashboard/index.html>
- Basé sur la syntaxe Rmarkdown
- Exemple: <https://lrouviere.shinyapps.io/dashboard/>

# Applications web avec shiny

- Shiny est un package R qui permet de construire des applications web avec R (uniquement).
- Exemples:
  - overfitting en machine learning:  
[https://lrouviere.shinyapps.io/overfitting\\_app/](https://lrouviere.shinyapps.io/overfitting_app/)
  - stations velib à Rennes: <https://lrouviere.shinyapps.io/velib/>

## En résumé

- 12 (+5) heures pour 3 ou 4 thèmes.
- 1 thème = quelques slides + tutoriel (compléments à lire + exercices).
- Nécessite un **investissement personnel** ⇒ les heures en séance ne sont pas suffisantes pour tout faire !

# Plan

## 1. Visualisation avec ggplot2

Graphiques R conventionnels (rappel)

La grammaire ggplot2

## 2. Cartes

ggmap

Contours shapefile contours avec sf

Cartes interactives avec leaflet

## 3. Quelques outils de visualisation dynamiques

rAmCharts et plotly

Graphes avec visNetwork

Tableau de bord avec flexdasboard

## 4. Évaluation

## Visualisation avec ggplot2

---

- Visualiser des données à l'aide de **graphes** est souvent le **point de départ** d'un projet science des données.
- On peut bien entendu **créer différents types de graphes** avec **R**.
- On commence par un (court) rappel des **fonctions conventionnelles**.
- suivi d'une présentation de la **construction de graphes** avec le package **ggplot2**.

## Visualisation avec ggplot2

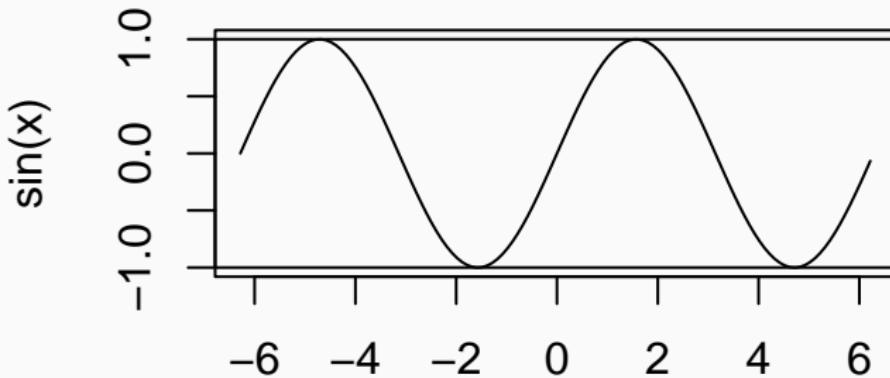
---

Graphiques R conventionnels (rappel)

## 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 barres pour une variable **qualitative**.
- Nuage de points pour 2 variables continues.
- Boxplot pour une distribution continue.

# Graphes classiques pour visualiser des variables

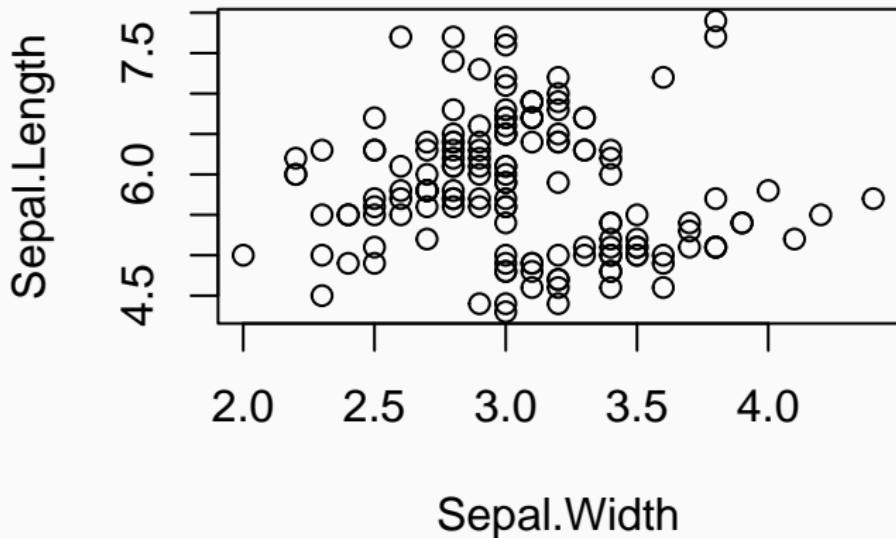
- Histogramme pour une variable continue, diagramme en barres pour une variable qualitative.
- Nuage de points pour 2 variables continues.
- Boxplot pour une distribution continue.

## Constat (positif)

Il existe une fonction R pour toutes les représentations.

## Nuage de points sur un jeu de données

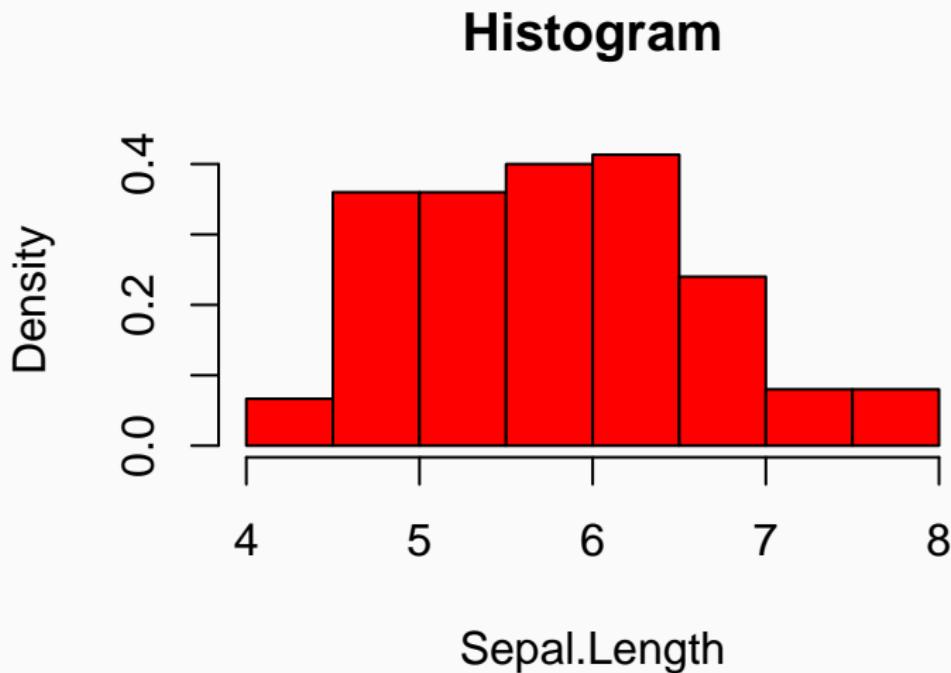
```
> plot(Sepal.Length~Sepal.Width, data=iris)
```



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

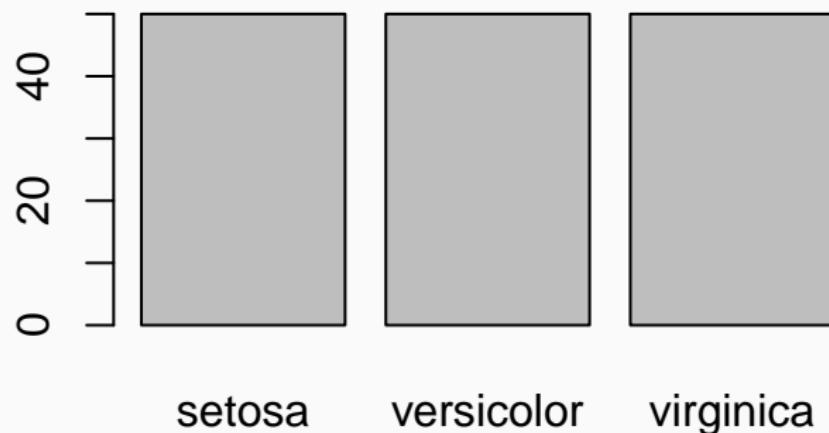
## Histogramme (variable continue)

```
> hist(iris$Sepal.Length, probability=TRUE,  
+       col="red", xlab="Sepal.Length", main="Histogram")
```



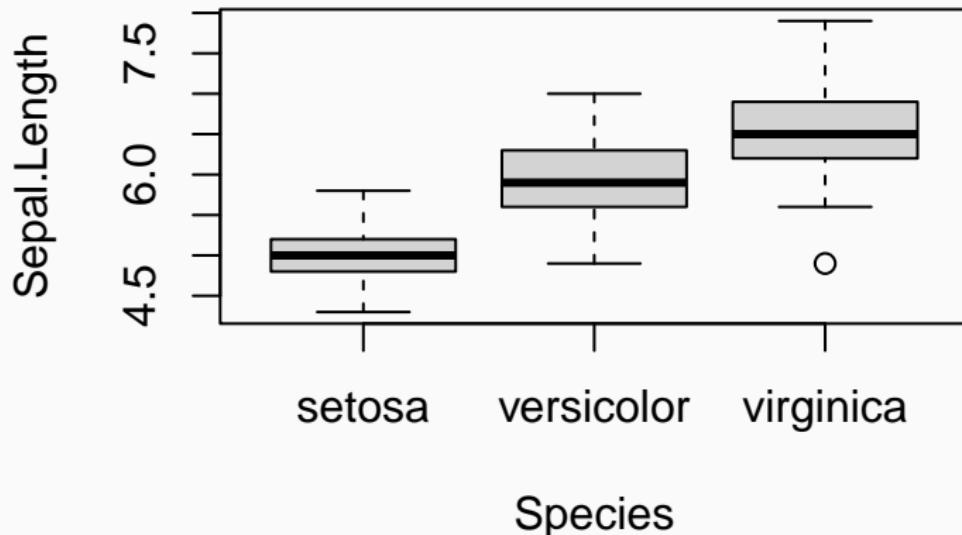
## Diagramme en barres (variable qualitative)

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



## Boxplot (distribution)

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



# Visualisation avec ggplot2

---

## La grammaire ggplot2

- `ggplot2` permet de faire des graphes R en s'appuyant sur une **grammaire des graphiques** (équivalent de `dplyr` pour manipuler les données).
- L'objectif est d'utiliser une **syntaxe claire** pour construire des graphes : **graphes "complexes"** avec une **syntaxe courte et lisible**.
- Les graphes produits sont **agréables à regarder** (pas toujours le cas avec les graphes conventionnels).
- Documents : [tutoriel](#), [livre](#)

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...).

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

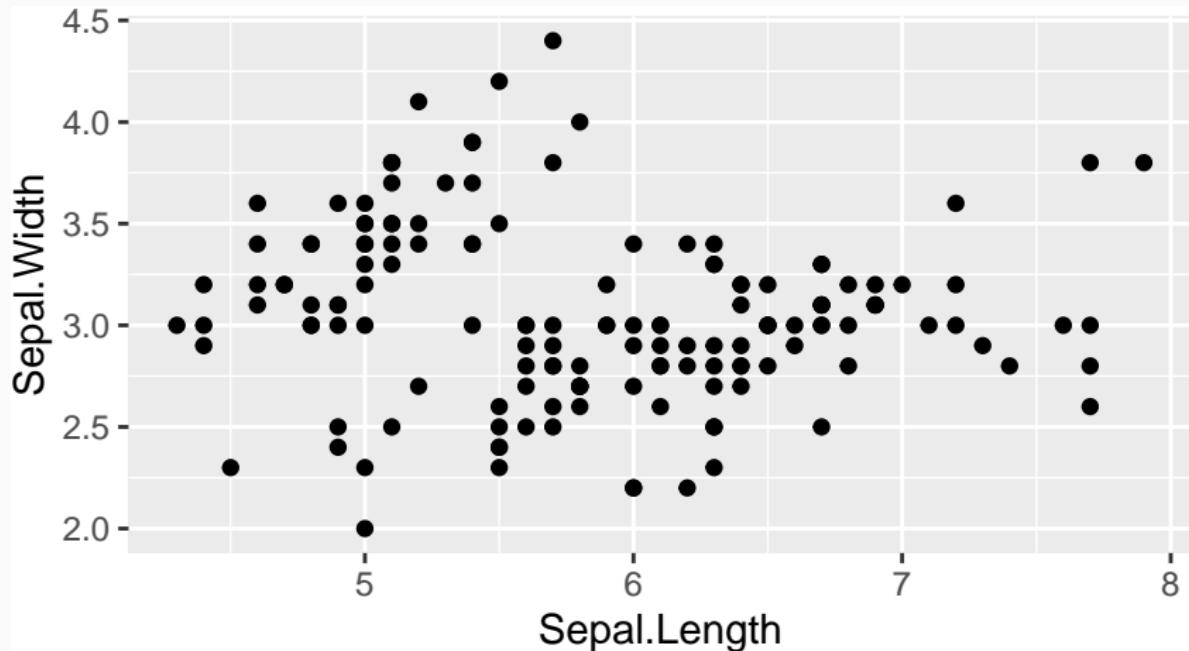
Les principaux **verbes** sont

- **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 +**.

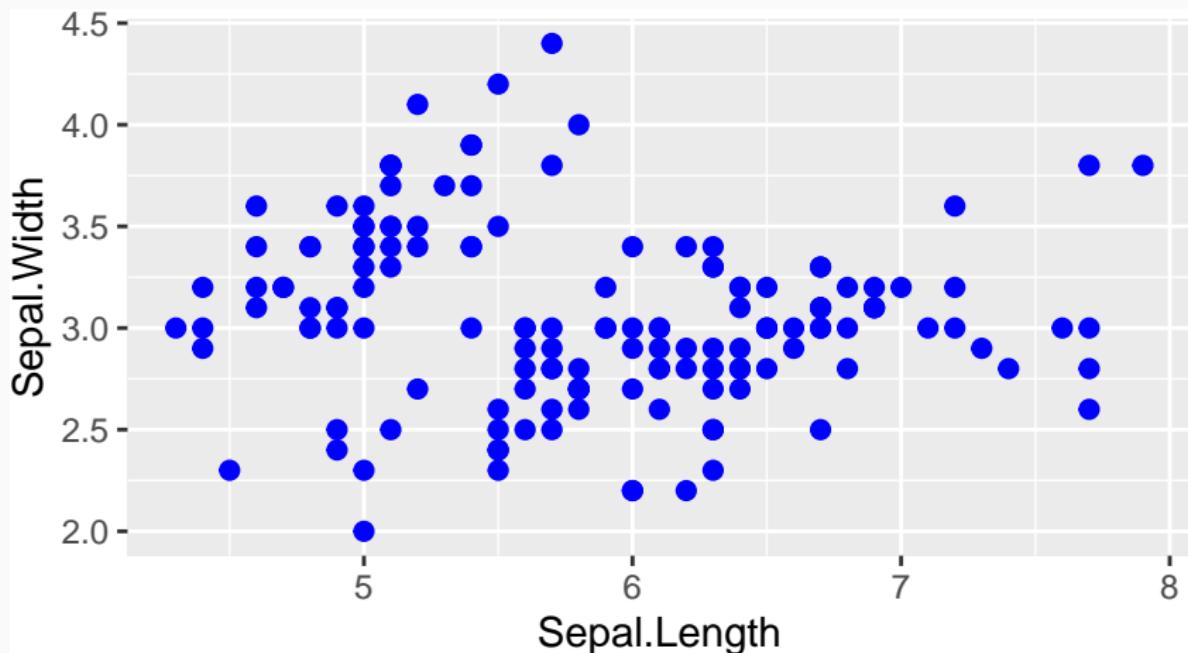
## Un exemple

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



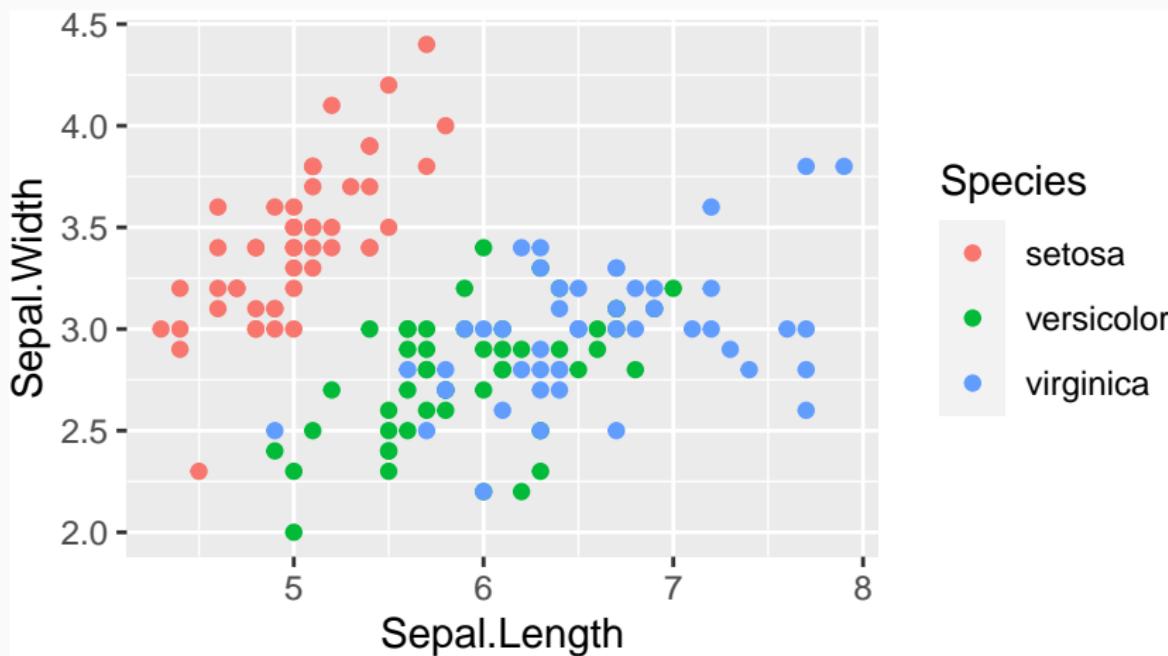
## Couleur et taille

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



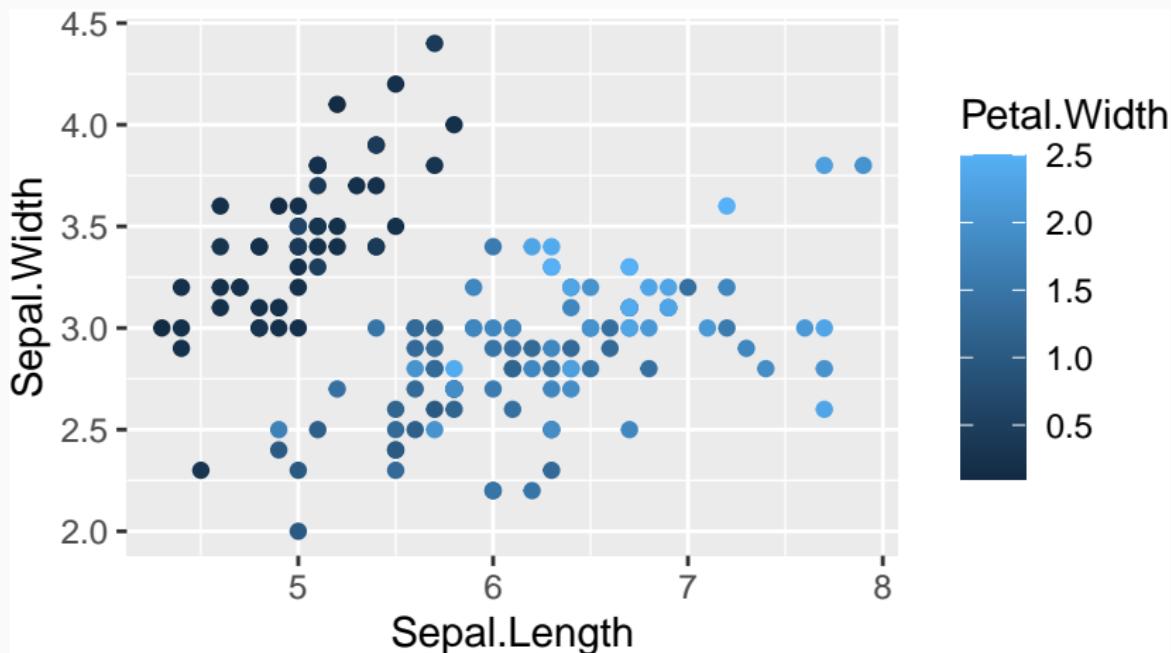
## Couleur avec une variable qualitative

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



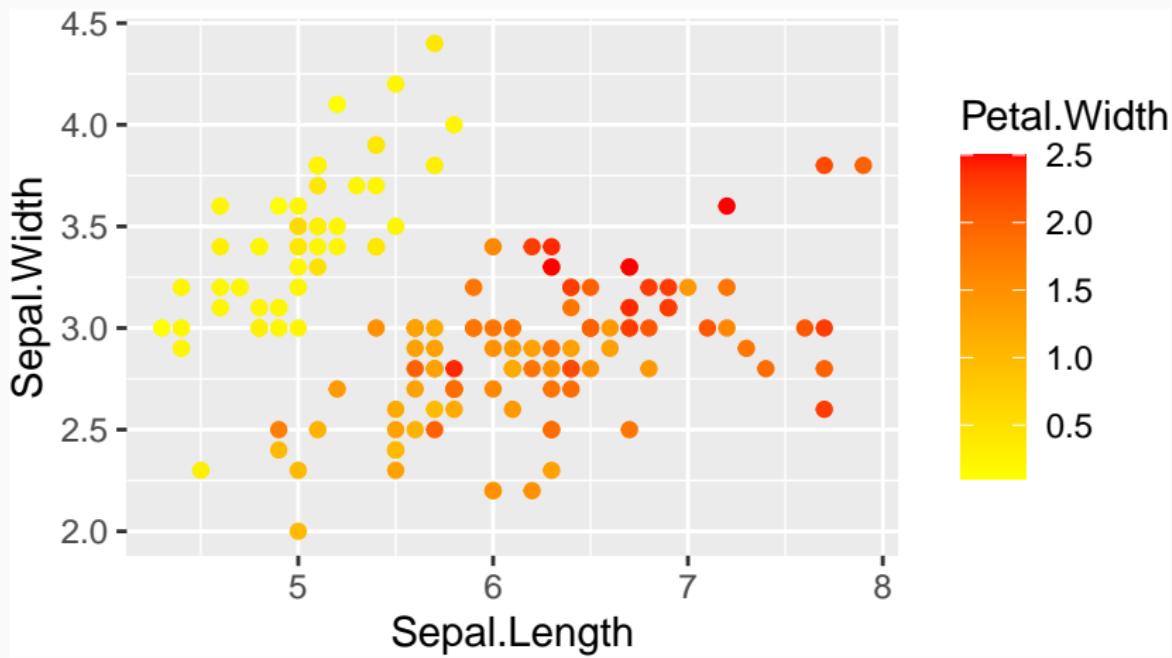
## Couleur avec une variable continue

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



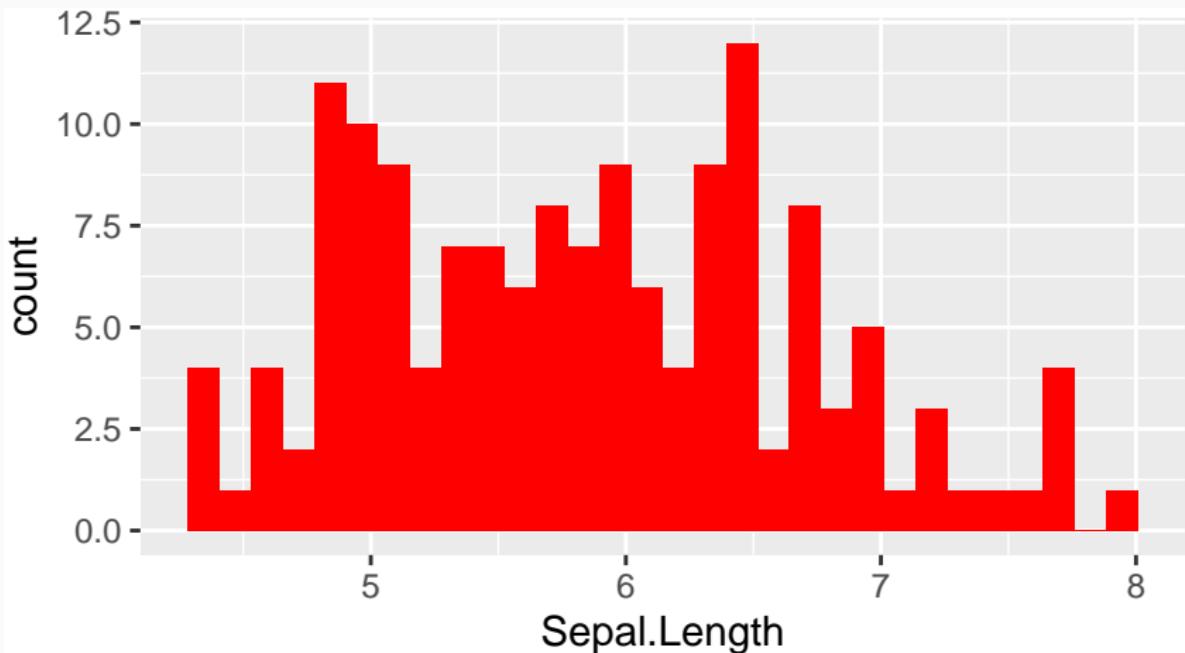
## Changer la couleur

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



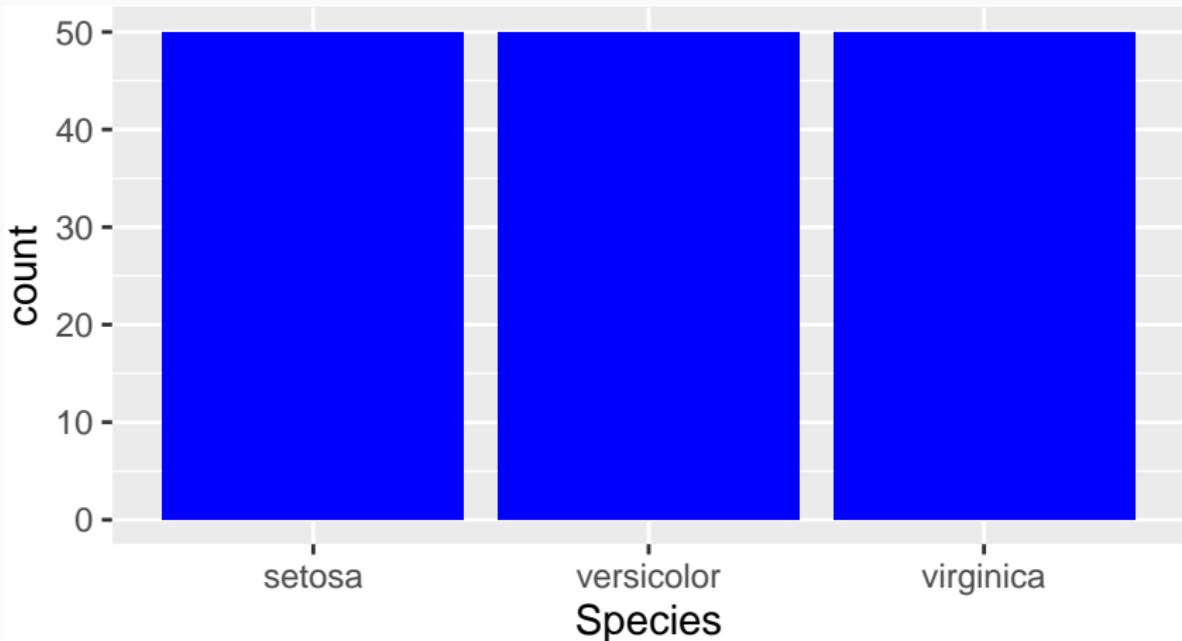
# Histogramme

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



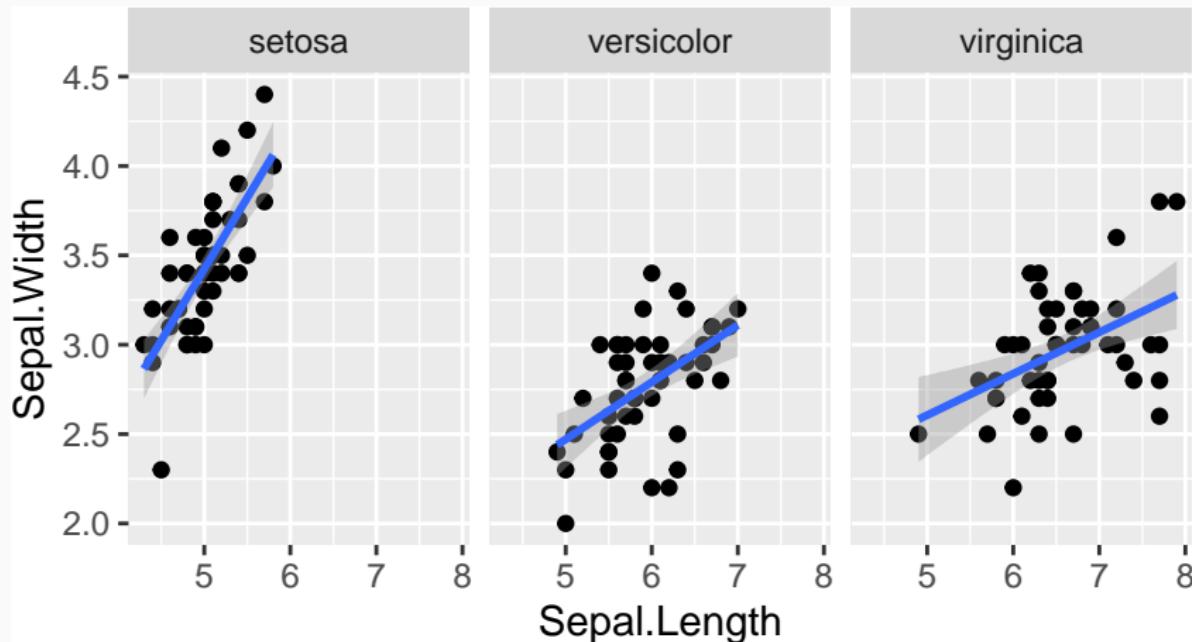
## Diagramme en barres

```
> 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)
```

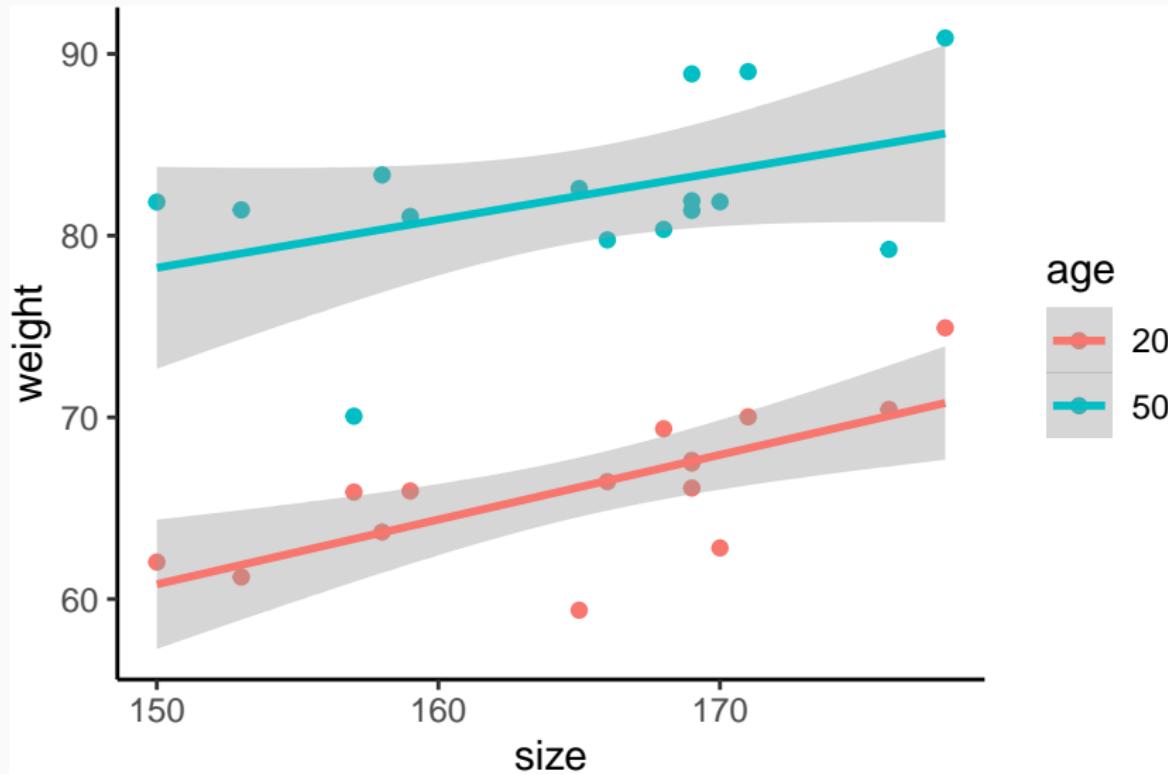


# Combiner ggplot et dplyr

- 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



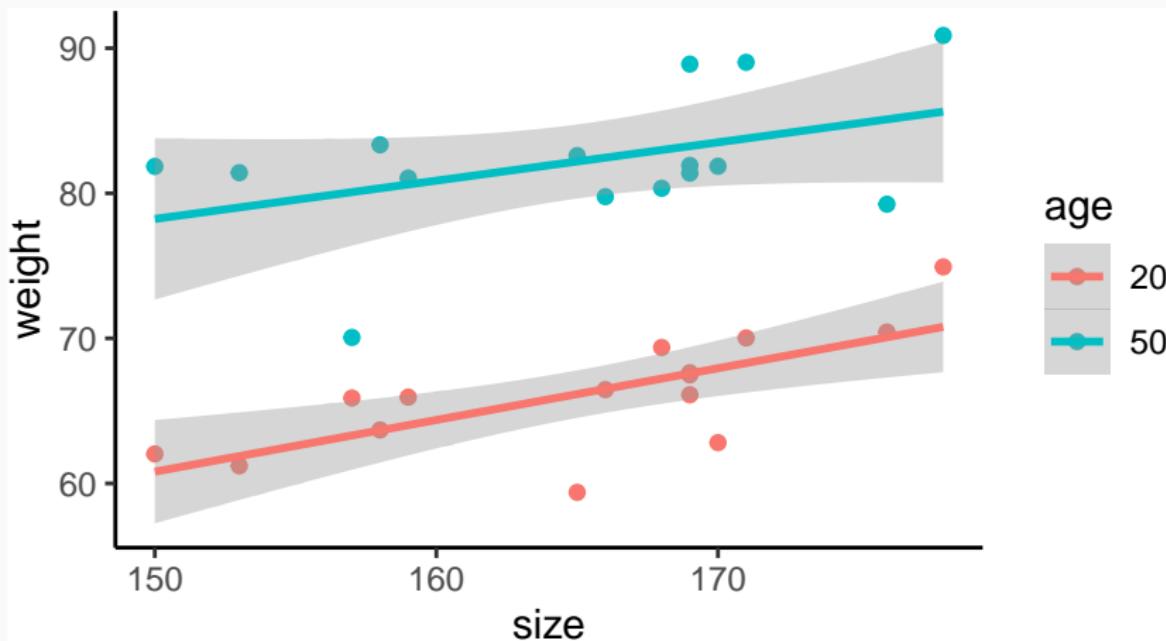
## Etape dplyr

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

```
> 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)
```

## Cartes

---

# Introduction

- 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 : **ggmap**, **sf** (cartes **statiques**) et **leaflet** (cartes **dynamiques**).

# Cartes

---

ggmap

# Syntaxe

- Proche de `ggplot`...

# Syntaxe

- Proche de `ggplot`...
- Au lieu de

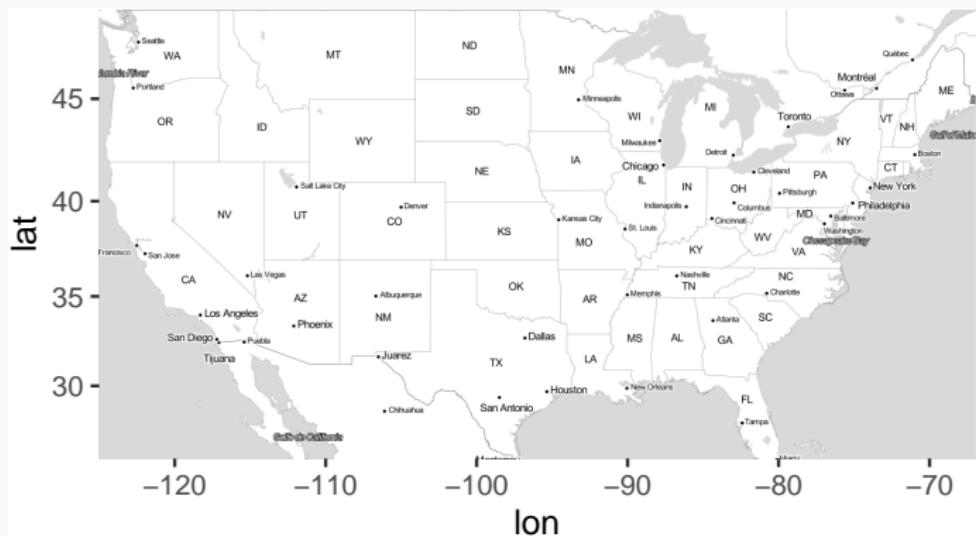
```
> ggplot(data)+...
```

- on utilise

```
> ggmap(backgroundmap)+...
```

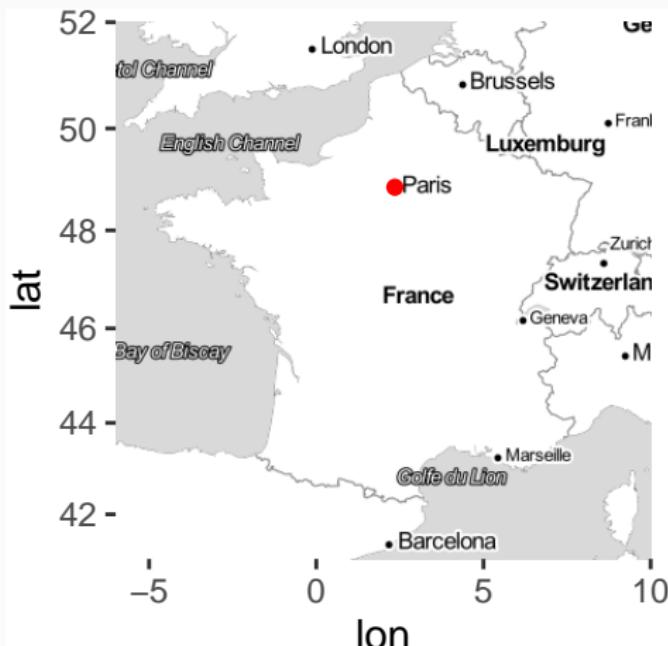
# Fonds de carte ggmap

```
> library(ggmap)  
> us <- c(left = -125, bottom = 25.75, right = -67, top = 49)  
> map <- get_stamenmap(us, zoom = 5, maptype = "toner-lite")  
> ggmap(map)
```



## Ajouts avec ggplot

```
> fr <- c(left = -6, bottom = 41, right = 10, top = 52)
> fond <- get_stamenmap(fr, zoom = 5,"toner-lite")
> Paris <- data.frame(lon=2.351499,lat=48.85661)
> ggmap(fond)+geom_point(data=Paris,aes(x=lon,y=lat),color="red")
```



# Cartes

---

Contours shapefile contours avec sf

## Le package sf

- **Ggmap** : bien pour des cartes “simples” (fond et quelques points).
- **Pas suffisant** pour des **représentations plus complexes** (colorier des pays à partir de variables).

## Le package sf

- **Ggmap** : bien pour des cartes “simples” (fond et quelques points).
- **Pas suffisant** pour des **représentations plus complexes** (colorier des pays à partir de variables).
- **sf** permet de gérer des **objets spécifiques à la cartographie** : notamment les différents **systèmes de coordonnées** et **leurs projections en 2d** (latitudes-longitudes, World Geodesic System 84...)
- Fonds de carte au format **shapefile** (**contours = polygones**)
- Compatible avec **ggplot** (verbe **geom\_sf**).

# Références

- <https://statnmap.com/fr/2018-07-14-initiation-a-la-cartographie-avec-sf-et-compagnie/>
- **Vignettes** sur le cran :  
<https://cran.r-project.org/web/packages/sf/index.html>
- **Un tutoriel** très complet (un peu technique) :  
<https://r-spatial.github.io/sf/articles/>

## Exemple

```
> library(sf)
> dpt <- read_sf("dpt")
> dpt[1:5,3]
Simple feature collection with 5 features and 1 field
Geometry type: MULTIPOLYGON
Dimension:      XY
Bounding box:   xmin: 644570 ymin: 6290136 xmax: 1022851 ymax: 6997000
Projected CRS: RGF93 / Lambert-93
# A tibble: 5 x 2
  NOM_DEPT                      geometry
  <chr>                         <MULTIPOLYGON [m]>
1 AIN                           (((919195 6541470, 918932 6541203, 918628 6...
2 AISNE                          (((735603 6861428, 735234 6861392, 734504 6...
3 ALLIER                         (((753769 6537043, 753554 6537318, 752879 6...
4 ALPES-DE-HAUTE-PROVENCE       (((992638 6305621, 992263 6305688, 991610 6...
5 HAUTES-ALPES                   (((1012913 6402904, 1012577 6402759, 101085...
```

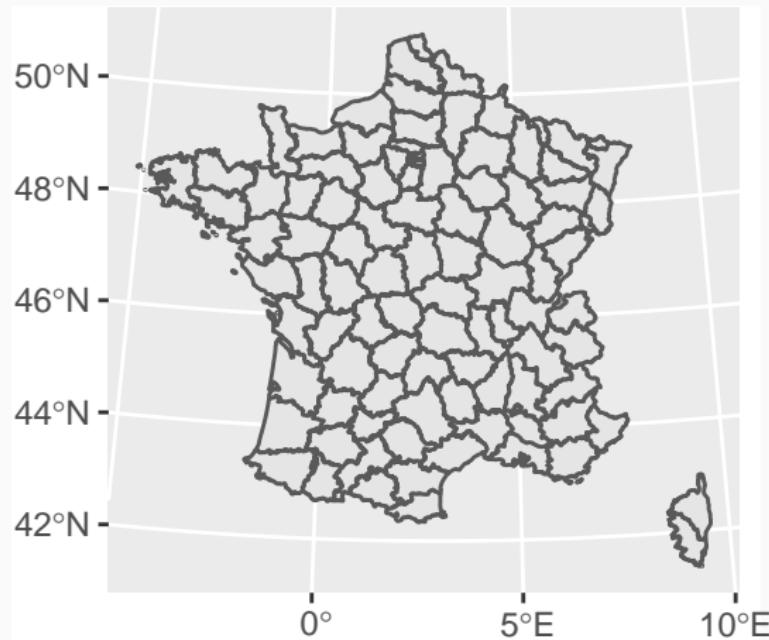
## Visualisation avec plot

```
> plot(st_geometry(dpt))
```



# Visualisation ggplot

```
> ggplot(dpt)+geom_sf()
```



## Ajouter des points sur le graphe

- Définir des coordonnées avec `st_point`

```
> point <- st_sfc(st_point(c(2.351462,48.85670)),  
+                   st_point(c(4.832011,45.75781)),  
+                   st_point(c(5.369953,43.29617)))
```

# Ajouter des points sur le graphe

- Définir des coordonnées avec `st_point`

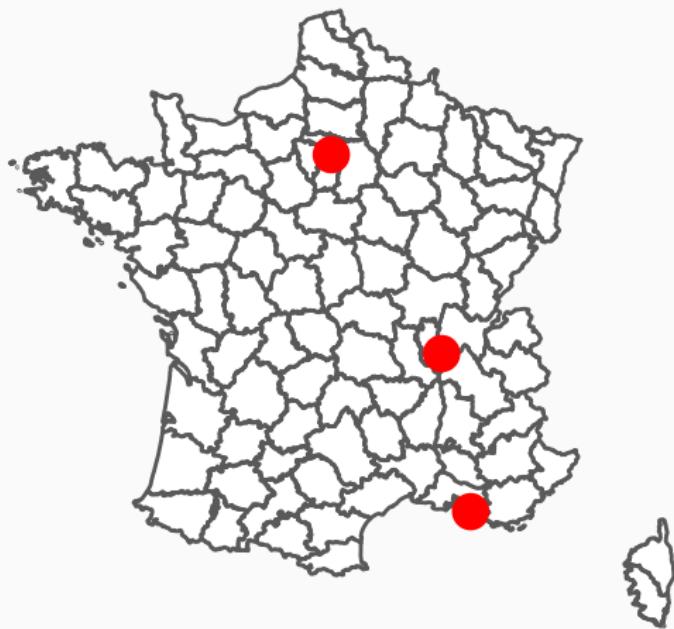
```
> point <- st_sfc(st_point(c(2.351462,48.85670)),  
+                   st_point(c(4.832011,45.75781)),  
+                   st_point(c(5.369953,43.29617)))
```

- Spécifier le `système de coordonnées` (4326 pour lat-lon)

```
> st_crs(point) <- 4326 #coord sont des long/lat  
> point  
Geometry set for 3 features  
Geometry type: POINT  
Dimension:      XY  
Bounding box:  xmin: 2.351462 ymin: 43.29617 xmax: 5.369953 ymax: 48.85  
Geodetic CRS:  WGS 84  
POINT (2.351462 48.8567)  
POINT (4.832011 45.75781)  
POINT (5.369953 43.29617)
```

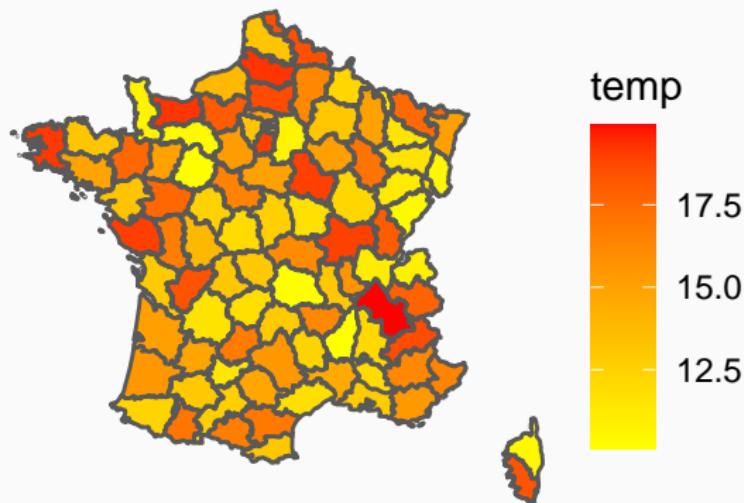
## Étape ggplot

```
> ggplot(dpt) + geom_sf(fill="white")+
+   geom_sf(data=point,color="red",size=4)+theme_void()
```



## Colorier des polygones

```
> set.seed(1234)
> dpt1 <- dpt %>% mutate(temp=runif(96,10,20))
> ggplot(dpt1) + geom_sf(aes(fill=temp)) +
+   scale_fill_continuous(low="yellow",high="red")+
+   theme_void()
```



## Compléments : la classe geometry

- Une des forces de `sf` est la classe `geometry` qu'il propose.

## Compléments : la classe geometry

- Une des forces de `sf` est la classe `geometry` qu'il propose.
- C'est cette classe qui conduit la représentation avec `plot` ou `geom_sf` :
  - `point` ou `multipoint`  $\implies$  points pour localiser un lieu
  - `polygon` ou `multipolygon`  $\implies$  contours pour représenter des frontières.

## Compléments : la classe geometry

- Une des forces de **sf** est la classe **geometry** qu'il propose.
- C'est cette classe qui conduit la représentation avec **plot** ou **geom\_sf** :
  - **point** ou **multipoint**  $\implies$  points pour localiser un lieu
  - **polygon** ou **multipolygon**  $\implies$  contours pour représenter des frontières.
- Quelques fonctions utiles :
  - **st\_point** et **st\_multipoint** : créer des points ou suite de points
  - **st\_sfc** : créer une liste d'objets **sf**
  - **st\_geometry** : extraire, modifier, remplacer, créer le geometry d'un objet
  - **st\_crs** : spécifier le système de coordonnées d'un geometry
  - **st\_cast** : transformer le type de geometry (passer d'un **MULTIPOINTS** à plusieurs **POINTS** par exemple)
  - ...

- Création d'un objet **sf** (simple feature)

```
> b1 <- st_point(c(3,4))
> b1
POINT (3 4)
> class(b1)
[1] "XY"    "POINT" "sfg"
```

- Création d'un objet **sfc** ("liste" d'objets sf)

```
> b2 <- st_sfc(st_point(c(1,2)),st_point(c(3,4)))
> b2
Geometry set for 2 features
Geometry type: POINT
Dimension:      XY
Bounding box:   xmin: 1 ymin: 2 xmax: 3 ymax: 4
CRS:            NA
POINT (1 2)
POINT (3 4)
> class(b2)
[1] "sfc_POINT" "sfc"
```

- Extraction, ajout, remplacement d'un **geometry**

```
> class(dpt)
[1] "sf"          "tbl_df"       "tbl"          "data.frame"
> b3 <- st_geometry(dpt)
> b3
Geometry set for 96 features
Geometry type: MULTIPOLYGON
Dimension:      XY
Bounding box:  xmin: 99226 ymin: 6049647 xmax: 1242375 ymax: 7110524
Projected CRS: RGF93 / Lambert-93
First 5 geometries:
MULTIPOLYGON (((919195 6541470, 918932 6541203, ...
MULTIPOLYGON (((735603 6861428, 735234 6861392, ...
MULTIPOLYGON (((753769 6537043, 753554 6537318, ...
MULTIPOLYGON (((992638 6305621, 992263 6305688, ...
MULTIPOLYGON (((1012913 6402904, 1012577 640275...
> class(b3)
[1] "sfc_MULTIPOLYGON" "sfc"
```

# Cartes

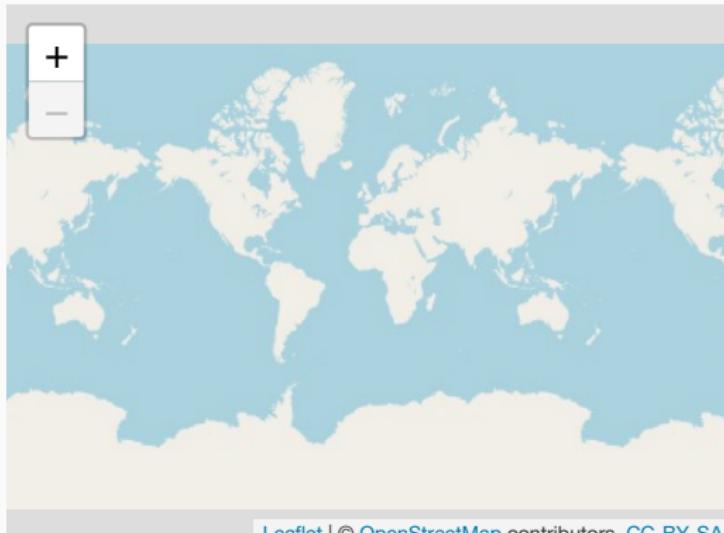
---

Cartes interactives avec leaflet

## Fonds de carte

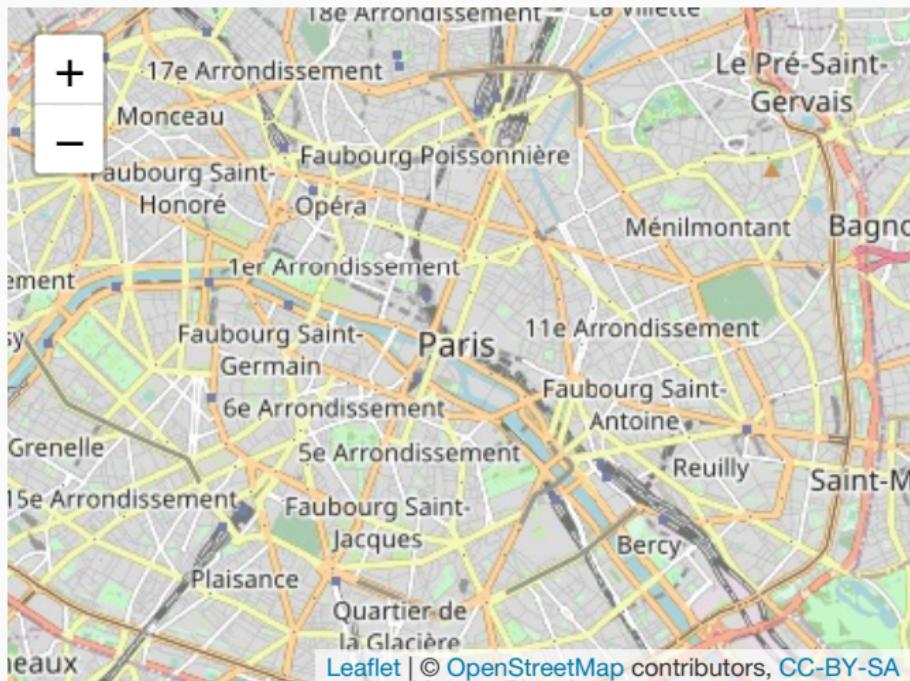
- Leaflet est une des librairies open-source JavaScript les plus populaires pour faire des cartes interactives.
- Documentation: [here](#)

```
> 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)
```



# Avec des données

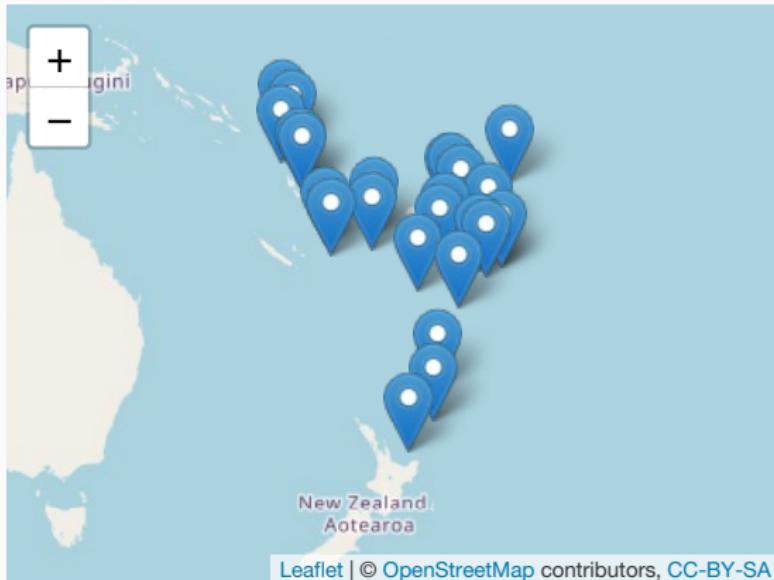
- Localiser 1000 séismes près des Fiji

```
> 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.

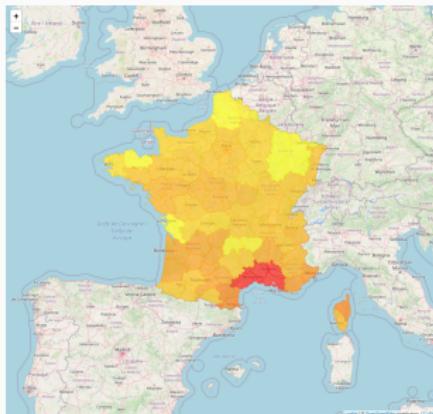
## addCircleMarkers

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



# Colorier polygones en combinant leaflet et sf

```
> leaflet() %>% addTiles() %>%  
+   addPolygons(data = dpt2,color=~pal1(t_prev),fillOpacity = 0.6,  
+                 stroke = TRUE,weight=1,  
+                 popup=~paste(as.character(NOM_DEPT),  
+                               as.character(t_prev),sep=" : "))
```



## Quelques outils de visualisation dynamiques

---

# Des packages R

- Graphiques classiques avec `rAmCharts` et `plotly`.
- Graphes avec `visNetwork`.
- Tableaux de bord avec `flexdashboard`.

## Quelques outils de visualisation dynamiques

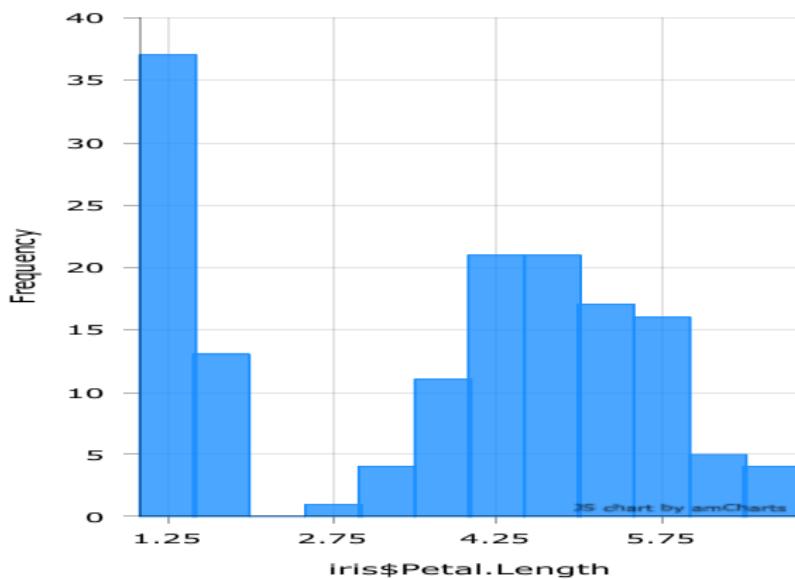
---

rAmCharts et plotly

- **User-friendly** pour des graphes standards (nuages de points, séries chronologiques, histogrammes. . . ).
- Il suffit d'utiliser la fonction **R** classique avec le préfixe **prefix am**.
- **Exemples** : **amPlot**, **amHist**, **amBoxplot**.
- **Références**: [https://datastorm-open.github.io/introduction\\_ramcharts/](https://datastorm-open.github.io/introduction_ramcharts/)

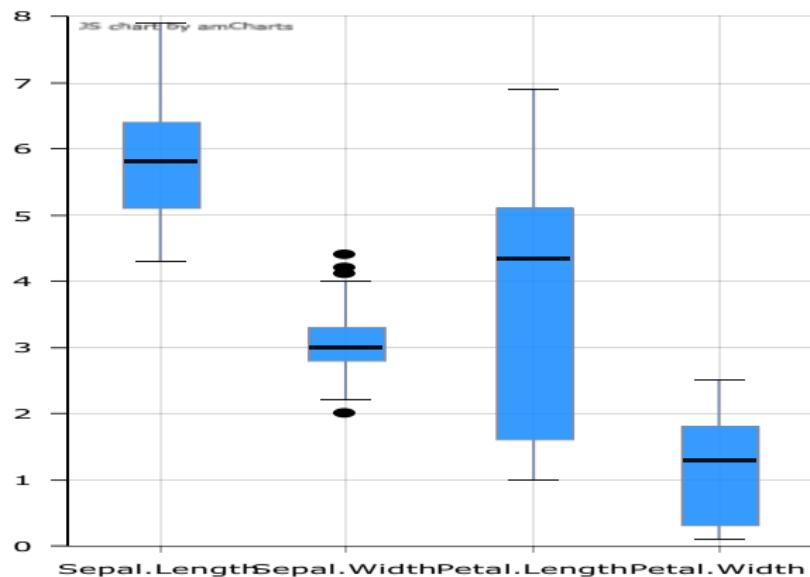
# rAmCharts Histogramme

```
> library(rAmCharts)
> amHist(iris$Petal.Length)
```



# rAmcharts Boxplot

```
> amBoxplot(iris)
```

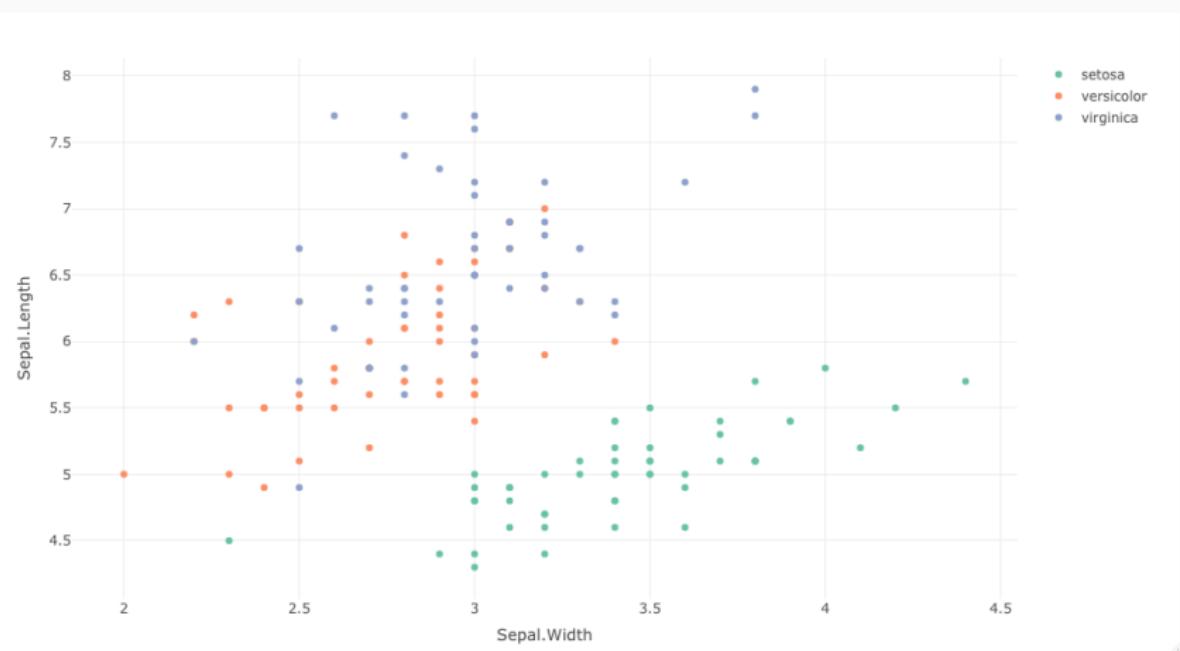


# Plotly

- Package **R** pour créer des **graphes interactifs** à partir de la librairie open source **Javascript plotly.js**.
- La syntaxe se décompose en **3 parties** :
  - données et variables (**plot\_ly**) ;
  - type de représentation (**add\_trace**, **add\_markers**... ) ;
  - options (axes, titres...) (**layout**).
- Références: <https://plot.ly/r/reference/>

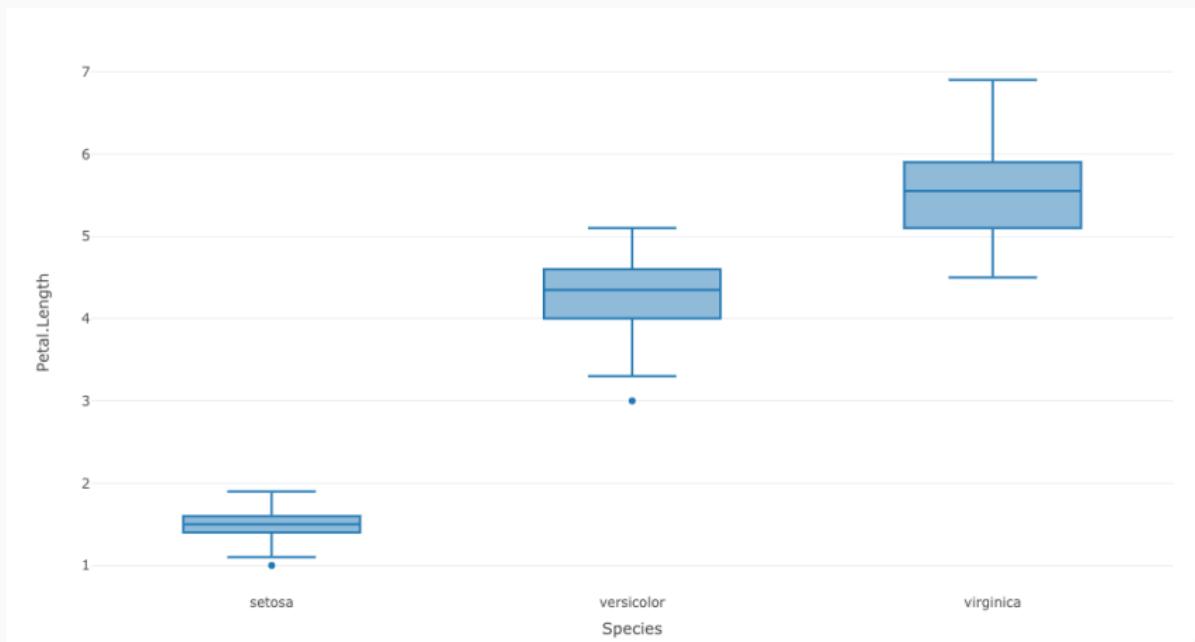
# Nuage de points

```
> library(plotly)  
> iris %>% plot_ly(x=~Sepal.Width,y=~Sepal.Length,color=~Species) %>%  
+   add_markers(type="scatter")
```



# Plotly boxplot

```
> iris %>% plot_ly(x=~Species,y=~Petal.Length) %>% add_boxplot()
```



## Quelques outils de visualisation dynamiques

---

Graphes avec visNetwork

## Connexions entre individus

- De nombreux jeux de données peuvent être visualisés avec des **graphes**, notamment lorsque l'on souhaite étudier des **connexions** entre individus (génétique, réseaux sociaux, système de recommandation...)

## Connexions entre individus

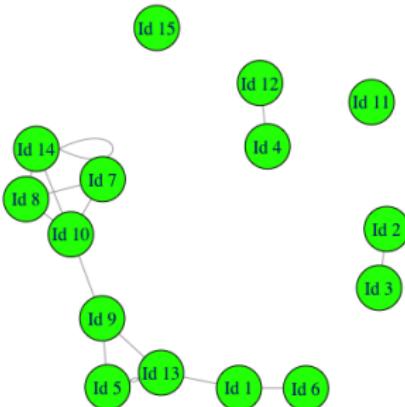
- De nombreux jeux de données peuvent être visualisés avec des **graphes**, notamment lorsque l'on souhaite étudier des **connexions** entre individus (génétique, réseaux sociaux, système de recommandation...)
- Un individu = **un nœud** et une connexion = **une arête**.

```
> set.seed(123)
> nodes <- data.frame(id = 1:15, label = paste("Id", 1:15))
> edges <- data.frame(from = trunc(runif(15)*(15-1))+1,
+                       to = trunc(runif(15)*(15-1))+1)
> head(edges)
   from to
1     5 13
2    12  4
3     6  1
4    13  5
5    14 14
6     1 13
```

# Graphe statique : le package igraph

- Références: <http://igraph.org/r/>, <http://kateto.net/networks-r-igraph>

```
> library(igraph)
> net <- graph_from_data_frame(d=edges, vertices=nodes, directed=F)
> plot(net,vertex.color="green",vertex.size=25)
```

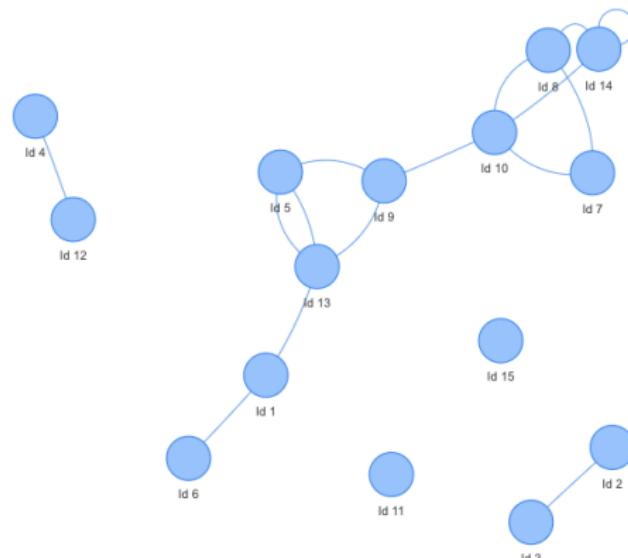


# Graph dynamique : le package visNetwork

- Référence:

<https://datastorm-open.github.io/visNetwork/interaction.html>

```
> library(visNetwork)
> visNetwork(nodes,edges)
```



## Quelques outils de visualisation dynamiques

---

Tableau de bord avec flexdasboard

- Juste un outil... mais **un outil important** en science des données
- Permet **d'assembler des messages importants** sur des données et/ou modèles

- Juste un outil... mais **un outil important** en science des données
- Permet **d'assembler des messages importants** sur des données et/ou modèles
- **Package** : flexdashboard
- **Syntaxe** : simple... juste du **Rmarkdown**
- Référence : <https://rmarkdown.rstudio.com/flexdashboard/>

# Header

```
---
```

```
title: "My title"
output:
  flexdashboard::flex_dashboard:
    orientation: columns
    vertical_layout: fill
    theme: default
---
```

- Le thème par défaut peut être remplacé par d'**autres thèmes** (cosmo, bootstrap, cerulean...) (voir [ici](#)). Il suffit d'ajouter

```
theme: yeti
```

# Flexdashboard | code

```
Descriptive statistics
```

```
=====
```

```
Column {data-width=650}
```

---

```
### Dataset
```

```
```{r}
```

```
DT::datatable(df, options = list(pageLength = 25))
```

```
```
```

```
Column {data-width=350}
```

---

```
### Correlation matrix
```

```
```{r}
```

```
cc <- cor(df[,1:11])
```

```
mat.cor <- corrplot::corrplot(cc)
```

```
```
```

```
### Histogram
```

```
```{r}
```

```
amHist(df$max03)
```

```
```
```

# Flexdashboard | dashboard

Linear models to predict ozone concentrations

Descriptive statistics Full linear model Selecting a simple linear model Selecting a linear model

Show 25 entries Search:

|          | maxO3 | T9   | T12  | T15  | Ne9 | Ne12 | Ne15 | Vx9     | Vx12    | Vx15    | maxO3v | vent  | plate |
|----------|-------|------|------|------|-----|------|------|---------|---------|---------|--------|-------|-------|
| 20010601 | 87    | 15.6 | 18.5 | 18.4 | 4   | 4    | 8    | 0.6946  | -1.7101 | -0.6946 | 84     | Nord  | Sec   |
| 20010602 | 82    | 17   | 18.4 | 17.7 | 5   | 5    | 7    | -4.3301 | -4      | -3      | 87     | Ouest | Sec   |
| 20010603 | 92    | 15.3 | 17.6 | 19.5 | 2   | 5    | 4    | 2.9544  | 1.8794  | 0.5209  | 82     | Est   | Sec   |
| 20010604 | 114   | 16.2 | 19.7 | 22.5 | 1   | 1    | 0    | 0.9848  | 0.3473  | -0.1736 | 92     | Nord  | Sec   |
| 20010605 | 94    | 17.4 | 20.5 | 20.4 | 8   | 8    | 7    | -0.5    | -2.9544 | -4.3301 | 114    | Ouest | Sec   |
| 20010606 | 89    | 17.7 | 19.8 | 18.3 | 6   | 6    | 7    | -5.6382 | -5      | -6      | 94     | Ouest | Plie  |
| 20010607 | 79    | 16.8 | 15.6 | 14.9 | 7   | 8    | 8    | -4.3301 | -1.8794 | -3.7588 | 80     | Ouest | Sec   |
| 20010610 | 79    | 14.9 | 17.5 | 18.9 | 5   | 5    | 4    | 0       | -1.0419 | -1.3802 | 99     | Nord  | Sec   |
| 20010611 | 101   | 16.1 | 19.6 | 21.4 | 2   | 4    | 4    | -0.766  | -1.0261 | -2.2981 | 79     | Nord  | Sec   |
| 20010612 | 106   | 18.3 | 21.9 | 22.9 | 5   | 6    | 8    | 1.2856  | -2.2981 | -3.9392 | 101    | Ouest | Sec   |
| 20010613 | 101   | 17.3 | 19.3 | 20.2 | 7   | 7    | 3    | -1.5    | -1.5    | -0.8662 | 106    | Nord  | Sec   |
| 20010614 | 90    | 17.6 | 20.3 | 17.4 | 7   | 6    | 8    | 0.6946  | -1.0419 | -0.6946 | 101    | Sud   | Sec   |
| 20010615 | 72    | 18.3 | 19.6 | 19.4 | 7   | 5    | 6    | -0.6662 | -2.7362 | -6.8944 | 90     | Sud   | Sec   |
| 20010616 | 70    | 17.1 | 18.2 | 18   | 7   | 7    | 7    | -4.3301 | -7.8785 | -5.1962 | 72     | Ouest | Plie  |
| 20010617 | 83    | 15.4 | 17.4 | 18.6 | 8   | 7    | 7    | -4.3301 | -2.0521 | -3      | 70     | Nord  | Sec   |
| 20010618 | 88    | 15.9 | 19.1 | 21.5 | 6   | 5    | 4    | 0.5209  | -2.9544 | -1.0261 | 83     | Ouest | Sec   |
| 20010620 | 145   | 21   | 24.6 | 28.9 | 0   | 1    | 1    | -0.342  | -1.5321 | -0.6884 | 121    | Ouest | Sec   |
| 20010621 | 81    | 18.2 | 22.4 | 23.4 | 8   | 3    | 1    | 0       | 0.3473  | -2.5712 | 145    | Nord  | Sec   |
| 20010622 | 121   | 18.7 | 24.2 | 26.9 | 2   | 1    | 0    | 1.5321  | 1.7321  | 2       | 81     | Est   | Sec   |
| 20010623 | 146   | 23.6 | 28.6 | 28.4 | 1   | 1    | 2    | 1       | -1.9284 | -1.2159 | 121    | Sud   | Sec   |
| 20010624 | 123   | 20.4 | 25.2 | 27.7 | 1   | 0    | 0    | 0       | -0.5209 | 1.0261  | 146    | Nord  | Sec   |
| 20010625 | 146   | 27   | 32.7 | 33.7 | 0   | 0    | 0    | 2.9544  | 6.5778  | 4.3301  | 121    | Est   | Sec   |
| 20010626 | 108   | 24   | 23.5 | 25.1 | 4   | 4    | 0    | -2.5712 | -3.8567 | -4.6985 | 146    | Sud   | Sec   |
| 20010627 | 83    | 19.7 | 22.9 | 24.8 | 7   | 6    | 6    | -2.5681 | -3.9392 | -4.924  | 108    | Ouest | Sec   |
| 20010628 | 57    | 20.1 | 22.4 | 22.8 | 7   | 6    | 7    | -5.6382 | -3.8302 | -4.5963 | 83     | Ouest | Plie  |

Showing 1 to 25 of 112 entries

Previous 1 2 3 4 5 Next

Correlation matrix

Histogram

# Évaluation

---

# Projet visualisation

- Groupe (3 membres)
- Trouver un **problème de visualisation** (par exemple un jeu de données avec un problème statistique classification, regression... )
- Construire une application shiny pour répondre au problème
- Ne pas hésiter à utiliser les outils présentés dans le cours (possibilité de proposer de nouveaux outils)
- Déployer l'application sur le net avec **shinyapps**, voir  
<https://docs.rstudio.com/shinyapps.io/index.html>
- Produire un document markdown (4 ou 5 pages) qui présente votre travail, utiliser une sortie pdf ou html.

# Protocole

Vous devrez déposer dans cursus

- le document markdown (html ou pdf) qui présente votre travail ainsi que l'url de l'application
- les fichiers shiny (app.R ou ui.R et server.R, global, css, img...)
- le jeu de données (ou l'url où on peut le récupérer facilement)
- tous les fichiers nécessaires au lancement de l'application

avant le lundi 21 mars, 18h.

## Quelques exemples (promo précédente)

- <https://kabdallah.shinyapps.io/shinyapp/>
- [https://quentincarric.shinyapps.io/FooTooL\\_by\\_CARRIC\\_DIALLO\\_MAHE/](https://quentincarric.shinyapps.io/FooTooL_by_CARRIC_DIALLO_MAHE/)
- <https://pierrelepagnol.shinyapps.io/Reporting/>
- <https://hersantmarc.shinyapps.io/projet/>
- [https://euvrardq.shinyapps.io/etude\\_tendances\\_youtube/](https://euvrardq.shinyapps.io/etude_tendances_youtube/)