

Visualisation des données avec R

Laurent Rouvière

janvier 2024

Table des matières

1	Présentation du cours	1
2	Visualiser des données	9
2.1	Graphes conventionnels	9
2.2	Visualisation avec ggplot2	12
3	Cartes	23
3.1	ggplot2	24
3.2	Contours shapefile contours avec sf	27
3.3	Cartes interactives avec leaflet	34
3.4	Autres packages carto	40
4	Quelques outils de visualisation dynamiques	43
4.1	rAmCharts, plotly et ggiraph	44
4.2	Graphes avec visNetwork	50
4.3	Tableau de bord avec flexdashboard	53

1 Présentation du cours

Présentation

- *Enseignant* : Laurent Rouvière, laurent.rouviere@univ-rennes2.fr
 - *Recherche* : statistique non paramétrique, apprentissage statistique.
 - *Enseignement* : statistique et probabilités (Université, école d'ingénieur, formation continue).

- Co-responsable du **Master MAS** et responsable du parcours **SDD-IA**.
- **Consulting** : énergie (ERDF), finance, marketing.
- **Prérequis** : niveau avancé en **R** - bases en statistique et programmation
- **Objectifs** :
 - Comprendre l'importance de la visualisation en **science des données**
 - Visualiser des **données**, approches statique, dynamique et interactive
 - Découvrir quelques packages de visualisation en **R**
 - Créer des applications web

Documents de cours

- **Slides** disponibles à l'url https://lrouviere.github.io/page_perso/cours/visualisationR.html#MAS
- **Tutoriel** : compléments de cours et exercices disponibles à l'url https://lrouviere.github.io/TUTO_VISU_R/

Ressources

- Le **net** : de nombreux tutoriels
- Livre : **R pour la statistique et la science des données**, PUR



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**.
- Dans ce cours, on va présenter quelques *outils R* :
 1. **ggplot2** : un package R pour visualiser les données 3h.
 2. **Cartes** avec **ggplot2**, **sf** et **leaflet** 3-4h.
 3. **Visualisation dynamique/interactive**
 - données avec **rAmCharts**, **Plotly** et **ggiraph** 1h.
 - tableaux de bord avec **flexdashboard** 1h.
 - application web avec **shiny** 3-4h.

Compléments

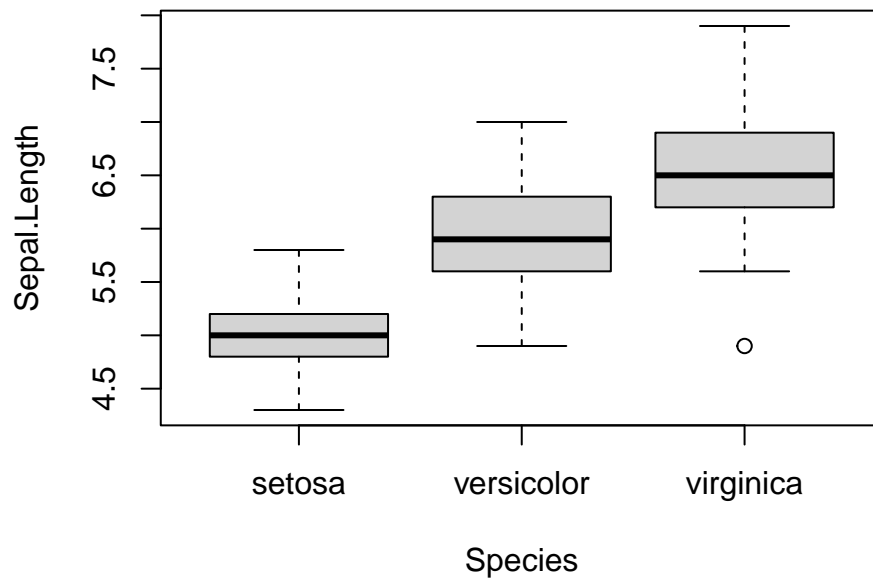
Workshop shiny à venir.

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
```

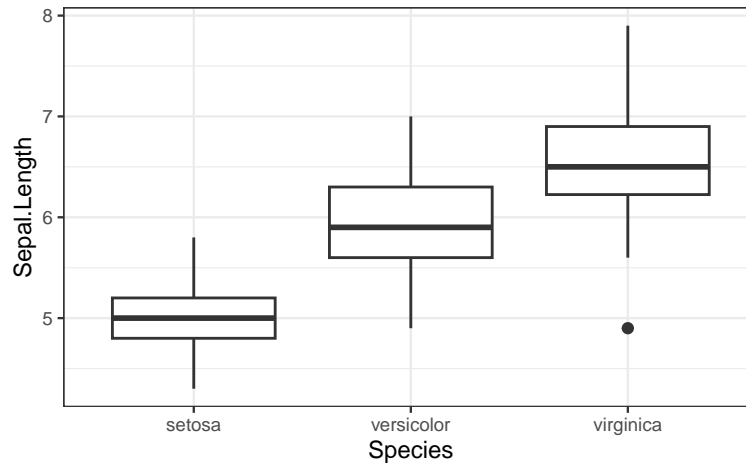
Fonctions conventionnelles

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

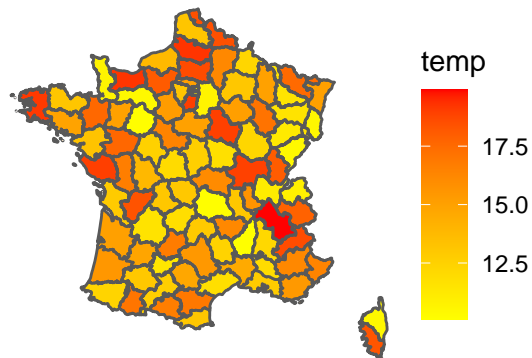


Grammaire ggplot

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



Une carte des températures



Diverses informations

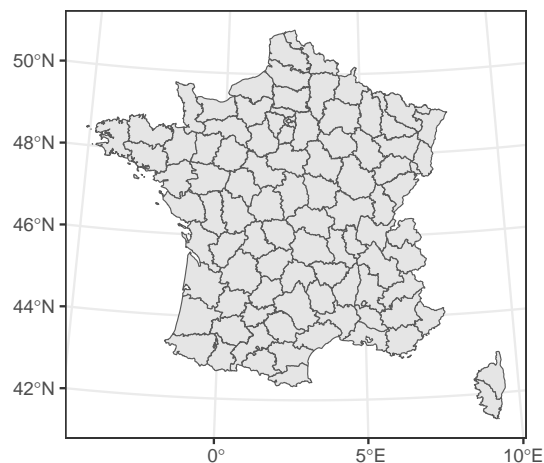
- **Fond de cartes** avec les frontières des départements ;
- **Températures** observées dans les départements (site web de météo france).

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 v1 / 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~
```

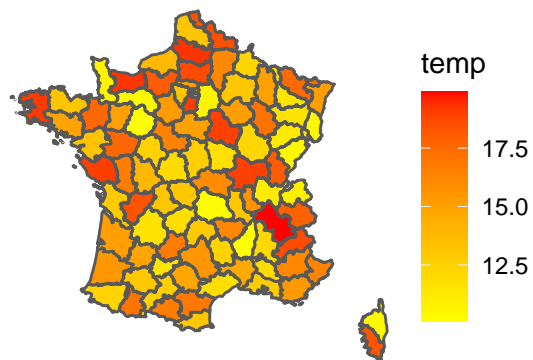
Fond de carte

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



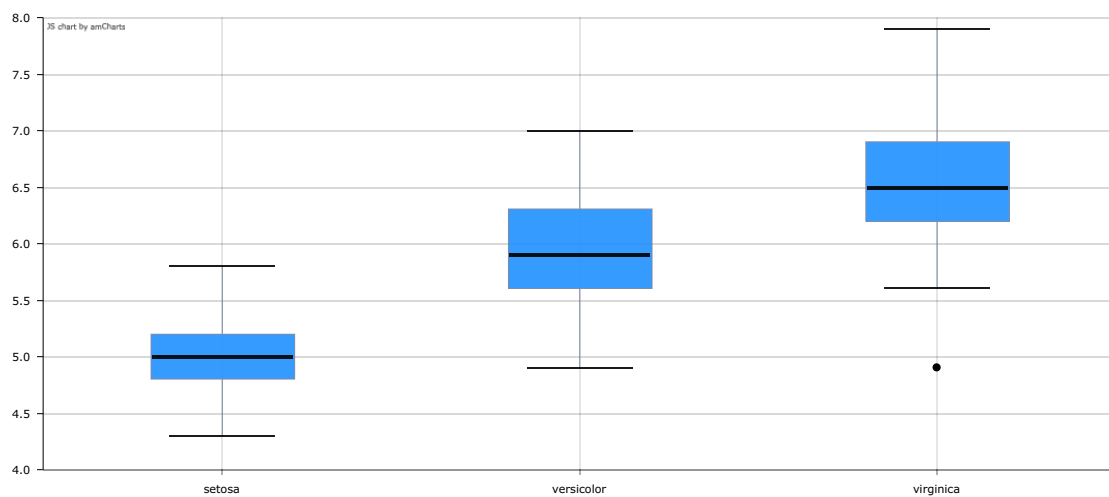
Ajout des températures

```
> 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>
- 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/
 - 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

Table des matières

2 Visualiser des données

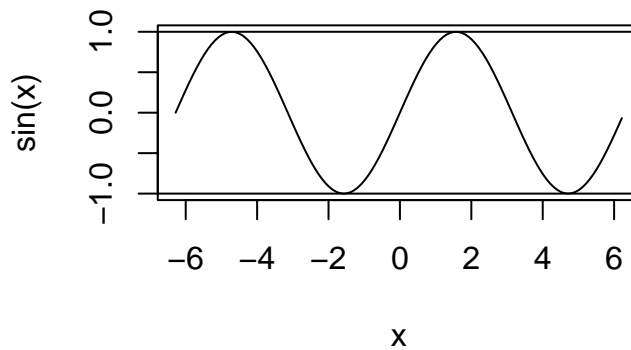
2.1 Graphes conventionnels

- *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 `ggplot2`.

La fonction `plot`

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

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



Graphes classiques pour visualiser des variables

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

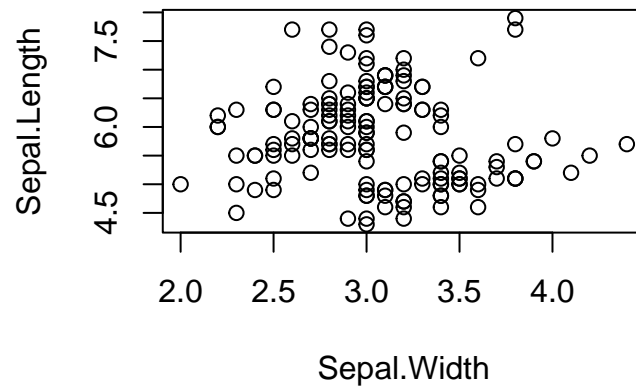
- **Boxplot** pour une distribution continue.

Constat (positif)

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

Nuage de points sur un jeu de données

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



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

Histogramme (variable continue)

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

Histogram of iris\$Sepal.Length

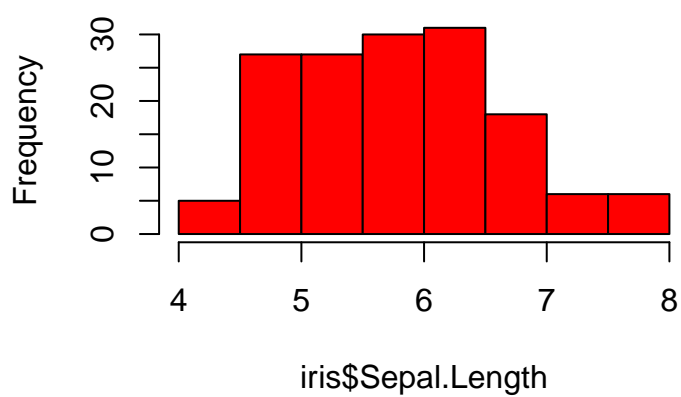
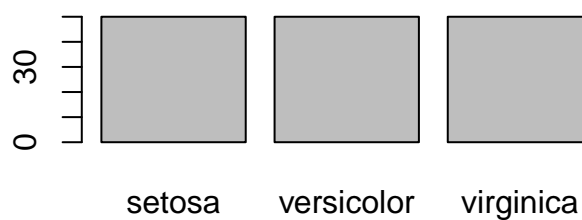


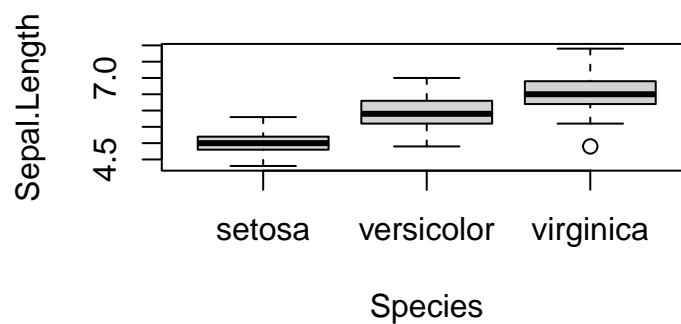
Diagramme en barres (variable qualitative)

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



Boxplot (distribution)

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



2.2 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** `ggplot2` permet d'obtenir des graphes "complexes" avec une **syntaxe claire et lisible**.

Remarque

Aujourd'hui la plupart des **graphes statiques** faits dans les tutoriels, livres, applications... sont faits avec `ggplot2`.

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_`

La grammaire

Les principaux *verbes* sont

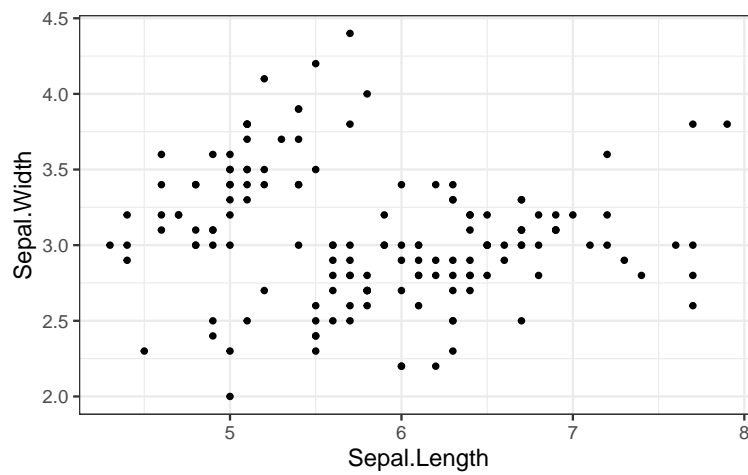
- **Data** (`ggplot`) : les *données*, un **dataframe** ou un **tibble**.
- **Aesthetics** (`aes`) : façon dont les *variables* doivent être représentées.
- **Geometrics** (`geom_...`) : *type* de représentation.

- **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 +*.

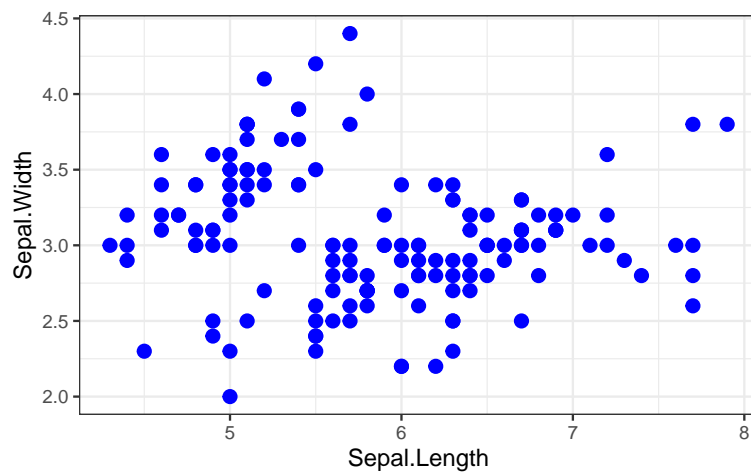
Un premier exemple

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



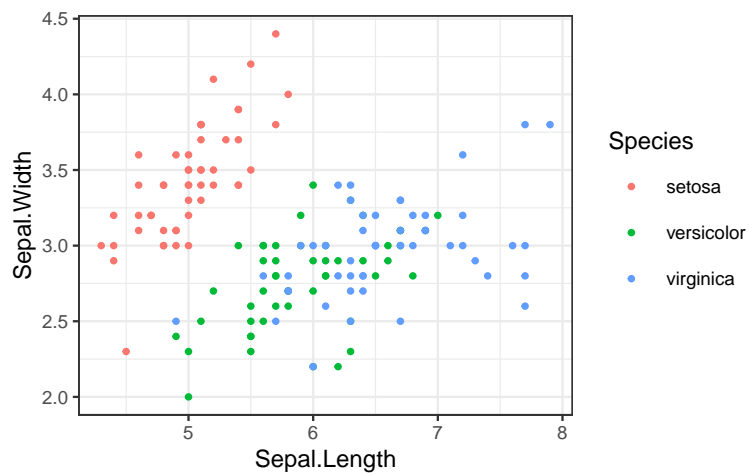
Couleur et taille

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



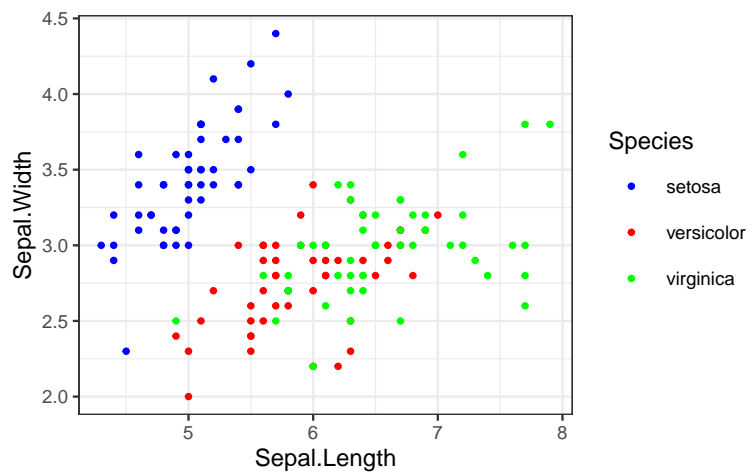
Couleur avec une variable qualitative

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



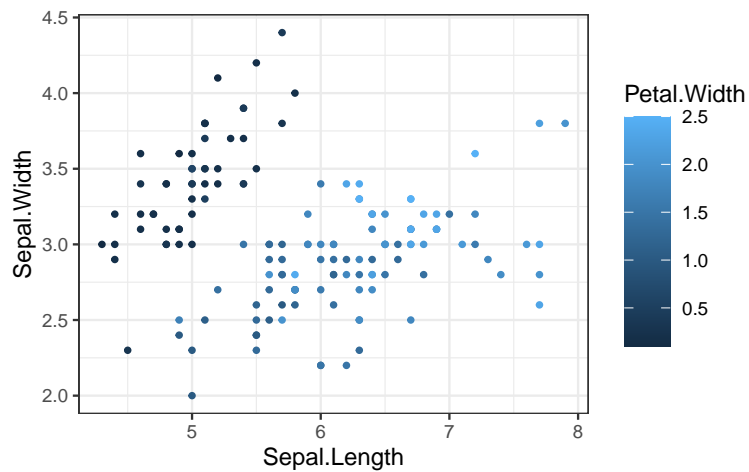
Changer la couleur

```
> ggplot(iris)+aes(x=Sepal.Length,y=Sepal.Width,
+ color=Species)+geom_point()+
+ scale_color_manual(values=c("setosa"="blue", "virginica"="green",
+ "versicolor"="red"))
```



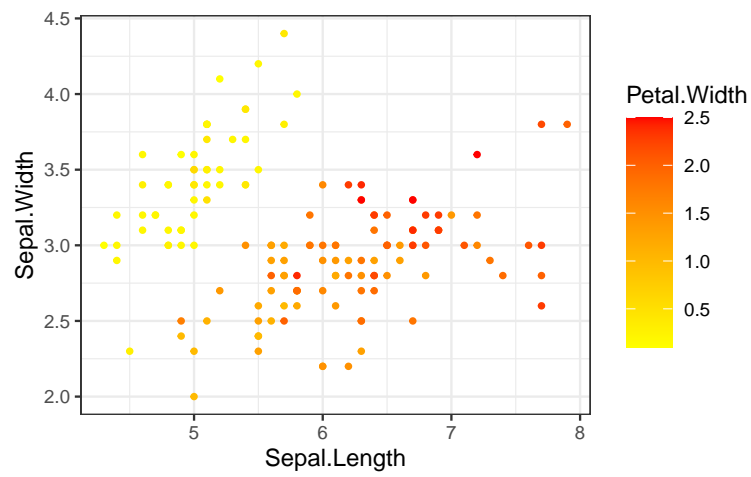
Couleur avec une variable continue

```
> 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")
```

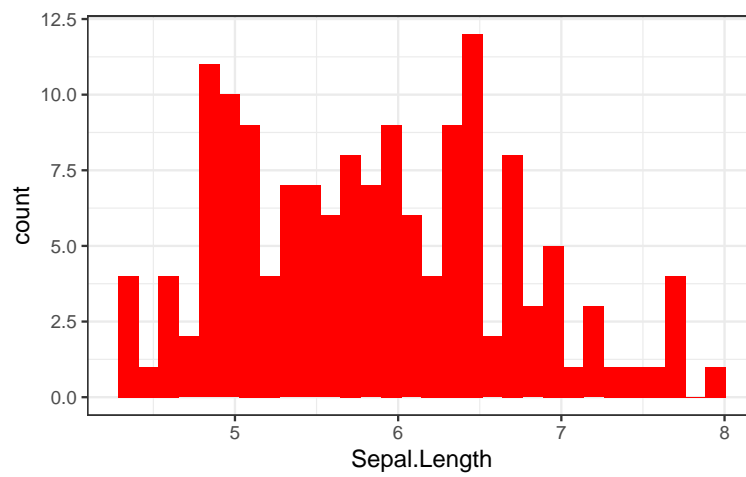
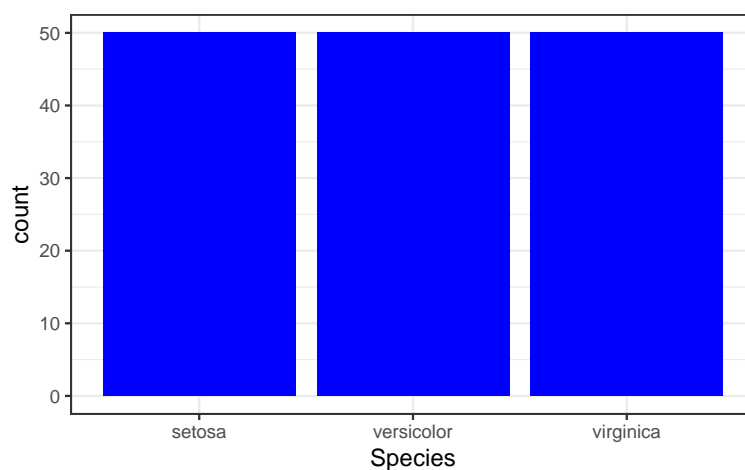


Diagramme en barres

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

Exemples de geom

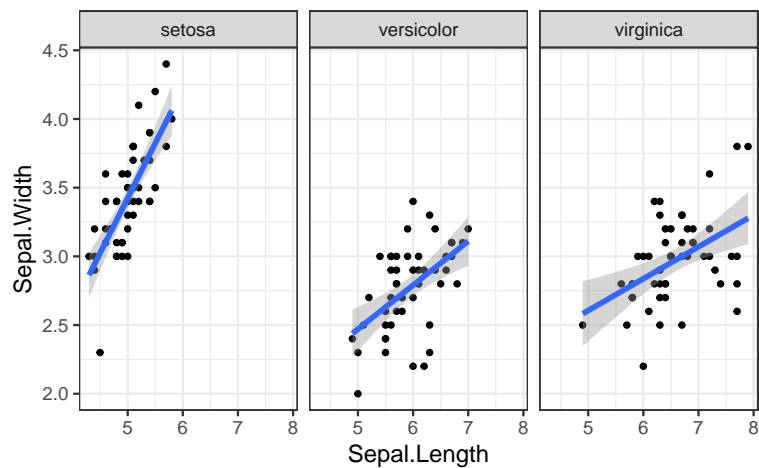
Geom	Description	Aesthetics
<code>geom_point()</code>	nuage de points	x, y, shape, fill
<code>geom_line()</code>	Ligne (ordonnée selon x)	x, y, linetype
<code>geom_abline()</code>	Ligne	slope, intercept
<code>geom_path()</code>	Ligne (ordonnée par l'index)	x, y, linetype
<code>geom_text()</code>	Texte	x, y, label, hjust, vjust
<code>geom_rect()</code>	Rectangle	xmin, xmax, ymin, ymax, fill, linetype
<code>geom_polygon()</code>	Polygone	x, y, fill, linetype
<code>geom_segment()</code>	Segment	x, y, xend, yend, fill, linetype

Geom	Description	Aesthetics
<code>geom_bar()</code>	Diagramme en barres	x, fill, linetype, weight
<code>geom_histogram()</code>	Histogramme	x, fill, linetype, weight
<code>geom_boxplot()</code>	Boxplot	x, fill, weight
<code>geom_density()</code>	Densité	x, y, fill, linetype
<code>geom_contour()</code>	Lignes de contour	x, y, fill, linetype
<code>geom_smooth()</code>	Lisseur (linéaire ou non linéaire)	x, y, fill, linetype

Geom	Description	Aesthetics
Tous		color, size, group

Facetting (très pertinent)

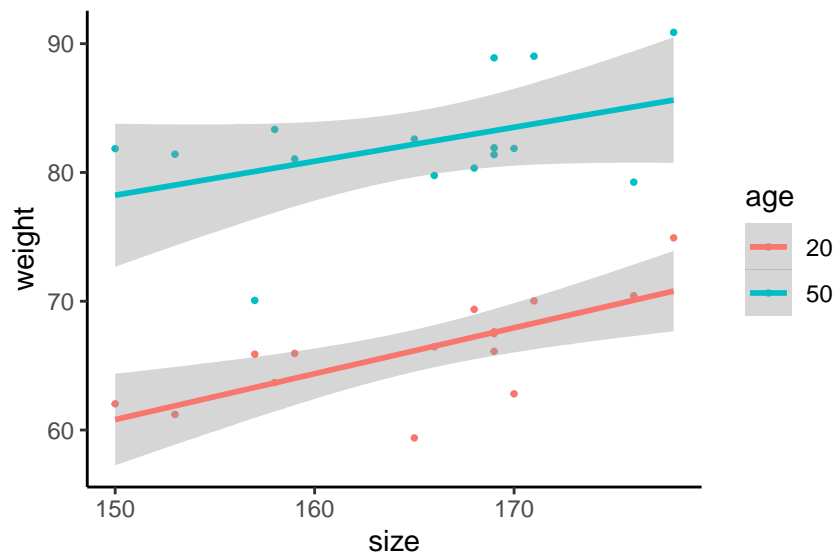
```
> ggplot(iris)+aes(x=Sepal.Length,y=Sepal.Width)+geom_point()+
+   geom_smooth(method="lm")+facet_wrap(~Species)
```



Combiner ggplot2 et dplyr/tidyr

- 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

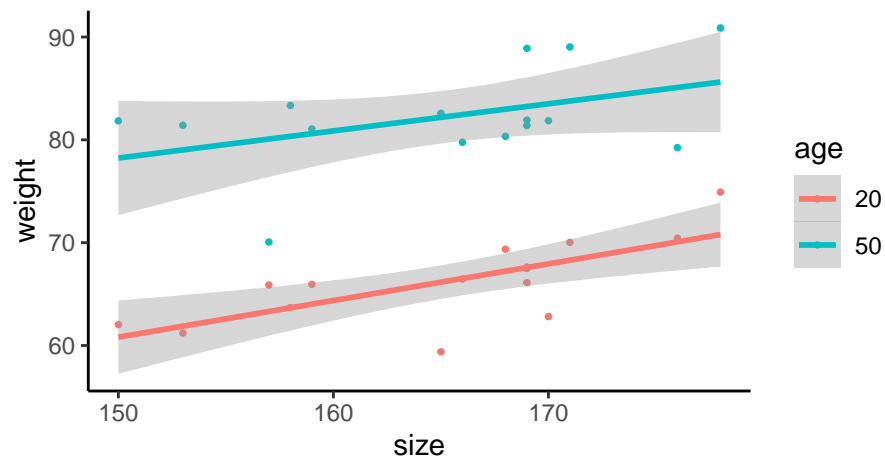
Etape **tidyr**

- 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 **ggplot2**

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



Statistics

- Certains graphes nécessitent de calculer des *indicateurs* à partir des données.
- **Exemple de l'histogramme** : compter le nombre d'observations (ou la densité) dans chaque classe.

Conséquence

geom_histogram fait appel à la fonction stat_bin pour calculer ces indicateurs.

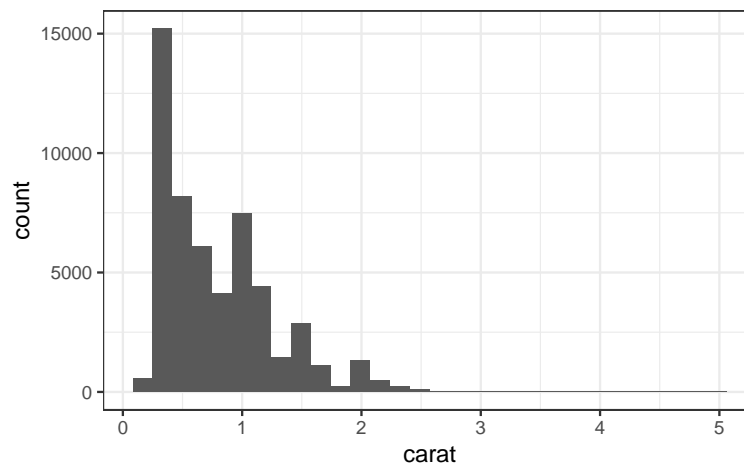
```
> geom_histogram(...,stat = "bin",...)
```

```
help(stat_bin)
Computed variables
count
number of points in bin

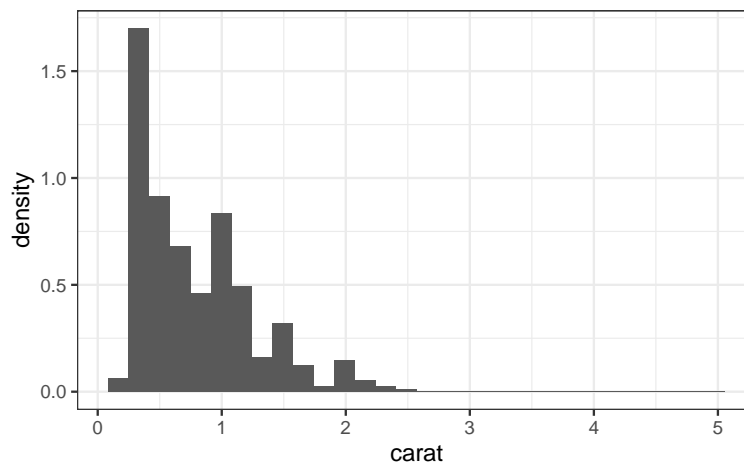
density
density of points in bin, scaled to integrate to 1
...
```

Visualiser une autre statistique

```
> ggplot(diamonds)+aes(x=carat)+
+   geom_histogram()
```



```
> ggplot(diamonds)+
+   aes(x=carat,y=after_stat(density))+
+   geom_histogram()
```

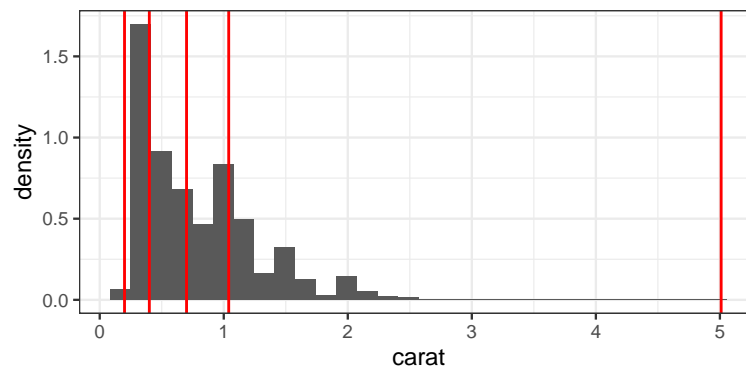


```
> #ou
> #ggplot(diamonds)+aes(x=carat,
> #                       y=..density..)+
> #   geom_histogram()
```

stat_summary

- D'une façon générale, `stat_summary` permet de calculer *n'importe quel indicateur* nécessaire au graphe.

```
> ggplot(diamonds)+aes(x=carat)+
+   geom_histogram(aes(y=after_stat(density)))+
+   stat_summary(aes(y=0,xintercept=after_stat(x)),
+     fun="quantile",geom="vline",
+     orientation = "y",color="red")
```

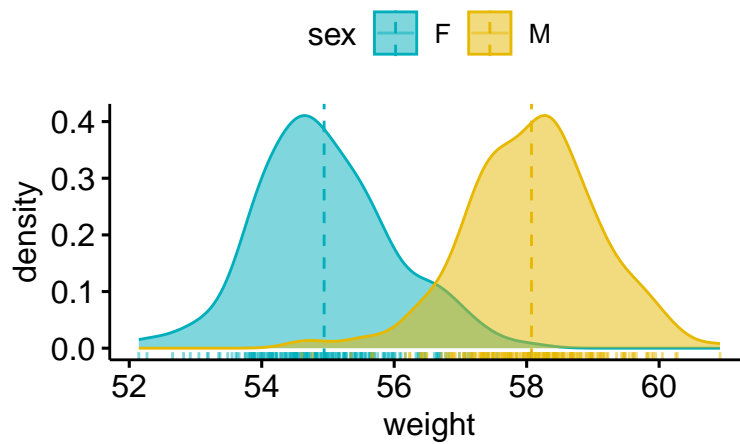


Compléments **ggpubr**

- Permet de faire des graphes ggplot relativement simples avec une *syntaxe simplifiée* (notamment sans l'utilisation de `aes`).
- Voir <https://rpkgs.datanovia.com/ggpubr/>

```
> head(wdata, 4)
  sex weight
1  F 53.79293
2  F 55.27743
3  F 56.08444
4  F 52.65430
```

```
> library(ggpubr)
> ggdensity(wdata, x = "weight",
+   add = "mean", rug = TRUE,
+   color = "sex", fill = "sex",
+   palette = c("#00AFBB", "#E7B800"))
```



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)
```

3 Cartes

Introduction

- De nombreuses applications nécessitent des *cartes* pour *visualiser* des *données* ou les résultats d'un *modèle*.
- De *nombreux packages R* : *ggplot2*, *RgoogleMaps*, *maps*...
- Dans cette partie : *ggplot2*, *sf* (cartes *statiques*) et *leaflet* (cartes *dynamiques*).

3.1 ggplot2

Syntaxe

- `ggplot2` permet de récupérer des fonds de carte avec `map_data`

```
> fond <- map_data(...)
```

- La syntaxe reste similaire par la suite :

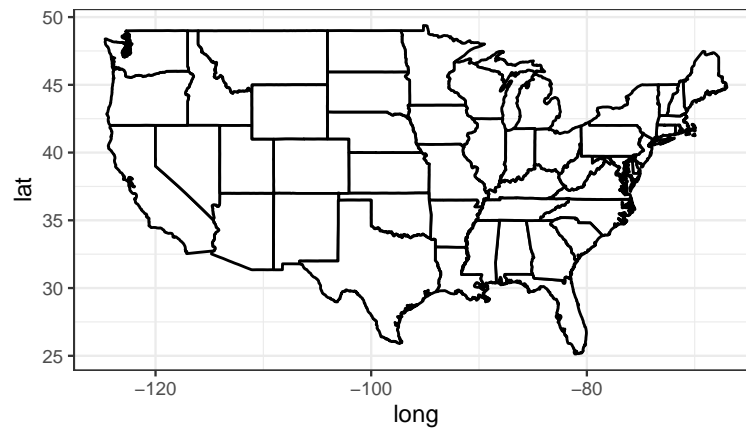
```
> ggplot(fond)+aes(...)
```

- On pourra consulter <https://ggplot2-book.org/maps#sec-polygonmaps>

Fonds de carte `map_data`

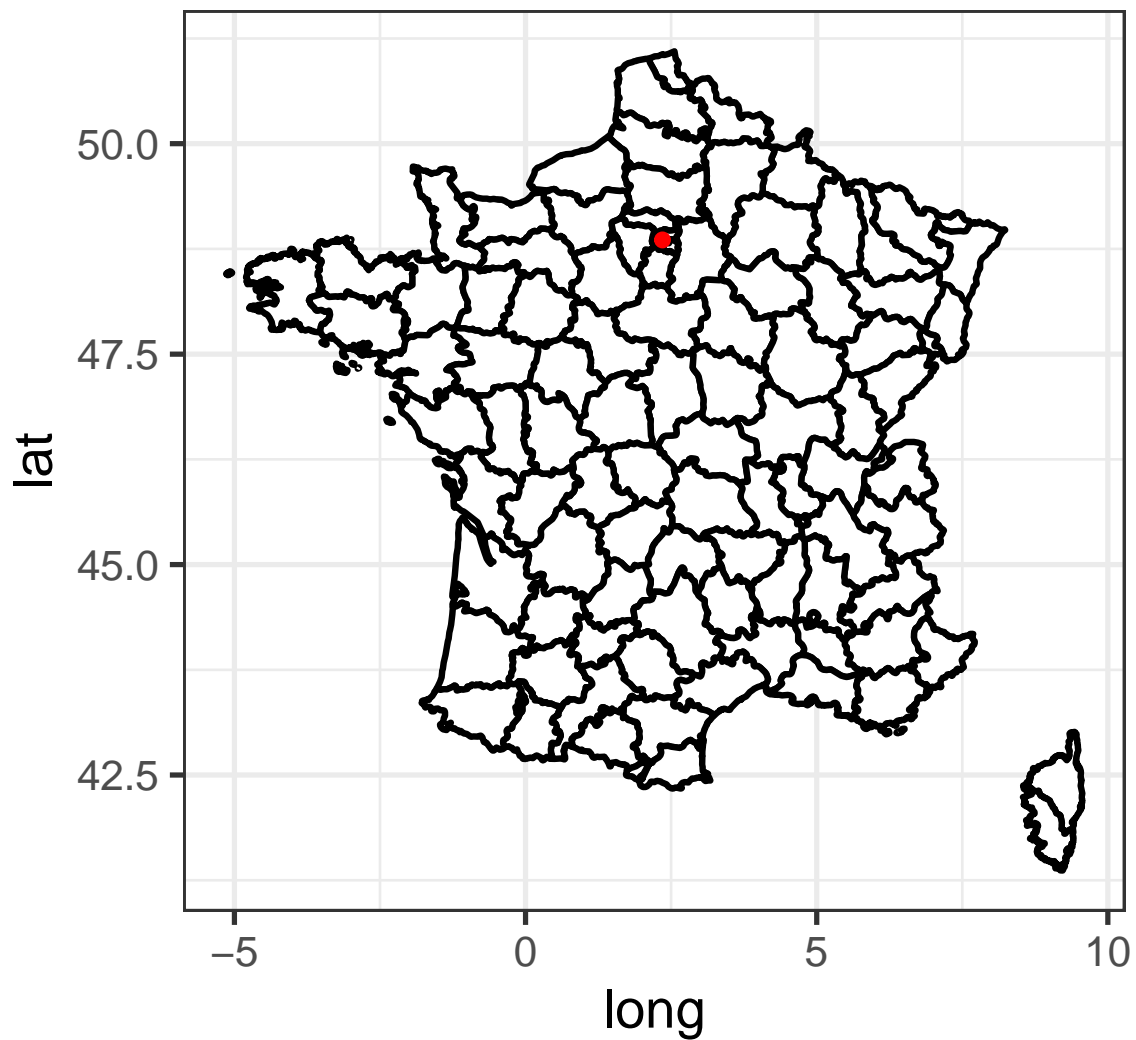
```
> us <- map_data("state")
> head(us)
      long      lat group order  region subregion
1 -87.46201 30.38968    1     1 alabama    <NA>
2 -87.48493 30.37249    1     2 alabama    <NA>
3 -87.52503 30.37249    1     3 alabama    <NA>
4 -87.53076 30.33239    1     4 alabama    <NA>
5 -87.57087 30.32665    1     5 alabama    <NA>
6 -87.58806 30.32665    1     6 alabama    <NA>
```

```
> ggplot(us)+aes(x=long,y=lat,group=group)+
+   geom_polygon(fill="white",color="black")+coord_quickmap()
```

Ajouts de points

```
> fr <- map_data("france")
> Paris <- tibble(long=2.351499,lat=48.85661)
> ggplot(fr)+aes(x=long,y=lat)+
+   geom_polygon(aes(group=group),fill="white",color="black")+
+   geom_point(data=Paris,color="red")+
+   coord_quickmap()
```



Géolocalisation

- Le package `tidygeocoder` propose de nombreux outils pour *géolocaliser* des lieux.
- Avec notamment la fonction `geocode` :

```
> library(tidygeocoder)
> tibble(ville=c("Paris", "Lyon", "Marseille", "Rennes")) |>
+   geocode(city=ville)
# A tibble: 4 x 3
  ville      lat long
<chr>    <dbl> <dbl>
1 Paris    48.9  2.32
```

2	Lyon	45.8	4.83
3	Marseille	43.3	5.37
4	Rennes	48.1	-1.68

3.2 Contours shapefile contours avec sf

Le package sf







- `ggplot2` : bien pour des cartes “simples” (fond et quelques points).
- *Pas suffisant* pour des **représentations plus complexes** (considérer une région comme un individu statistique).
- `sf` (*Simple Features*) 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 `ggplot2` (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/>
- Le chapitre <https://ggplot2-book.org/maps#sec-sf>

Le format shapefile

- Format de fichiers pour les *systèmes d'information géographiques (SIG)*.
- Permet de stocker la **forme**, la **localisation** et les **attributs** d'entités géographiques.
- Stocker sous la forme d'un *ensemble de fichiers*.

-  **departement.dbf**
-  **departement.lyr**
-  **departement.prj**
-  **departement.shp**
-  **departement.shx**
-  **departement.avl**

Lire des fichiers shapefile

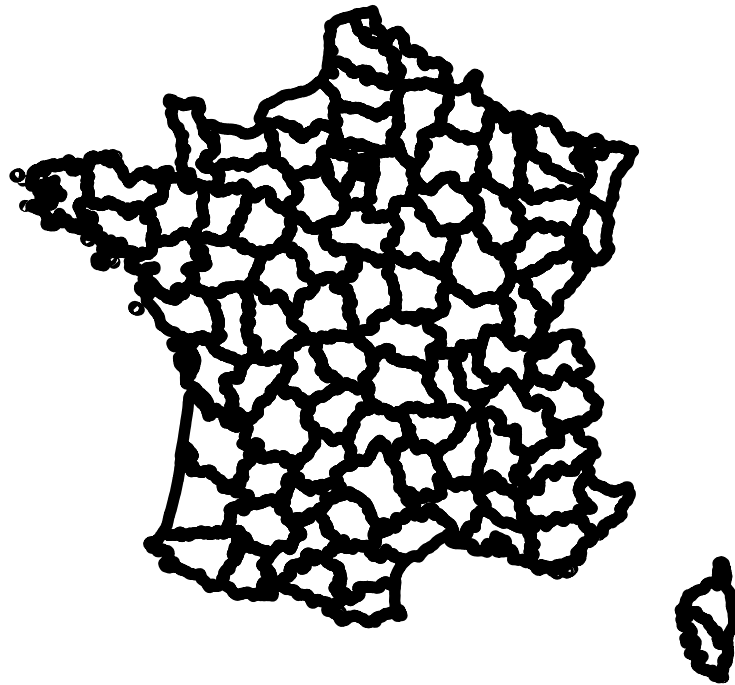
```
> library(sf)
> dpt <- read_sf("./data/dpt")
> dpt[1:5,3]
Simple feature collection with 5 features and 1 field
Geometry type: MULTIPOLYGON
Dimension: XY
Bounding box: xmin: 644570 ymin: 6290136 xmax: 1022851 ymax: 6997000
Projected CRS: RGF93 v1 / Lambert-93
# A tibble: 5 x 2
  NOM_DEPT geometry
  <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~
> class(dpt)
[1] "sf"          "tbl_df"      "tbl"         "data.frame"
```

Créer un objet sf

```
> fr <- map_data("france") |> as_tibble()
> fr1 <- fr |>
+   select(long,lat) |>
+   split(fr$group) |>
+   map(as.matrix) |>
+   st_polygon() |>
+   st_sfc(crs=4326)
> fr1
Geometry set for 1 feature
Geometry type: POLYGON
Dimension: XY
Bounding box: xmin: -5.14209 ymin: 41.366 xmax: 9.562665 ymax: 51.09752
Geodetic CRS: WGS 84
POLYGON ((2.557093 51.09752, 2.579995 51.00298,...
```

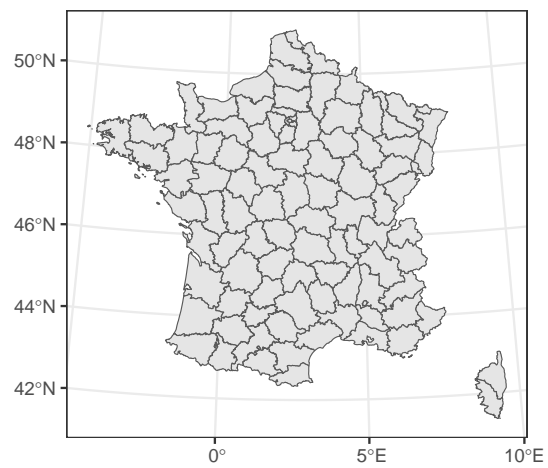
Visualisation avec plot

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



Visualisation ggplot

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



Ajouter des points sur le graphe

- Définir des coordonnées avec `st_point`

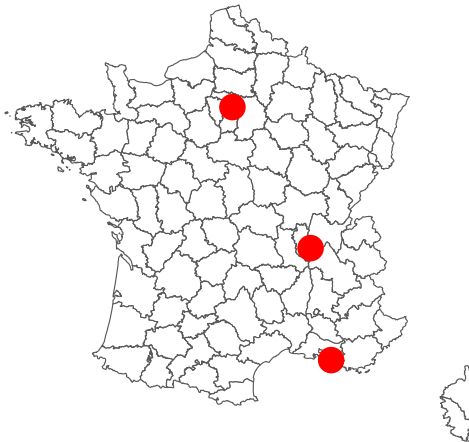
```
> point <- st_sfc(st_point(c(2.351462,48.85670)),  
+                 st_point(c(4.832011,45.75781)),  
+                 st_point(c(5.369953,43.29617)))
```

- 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: 48.8567  
Geodetic CRS: WGS 84  
POINT (2.351462 48.8567)  
POINT (4.832011 45.75781)  
POINT (5.369953 43.29617)
```

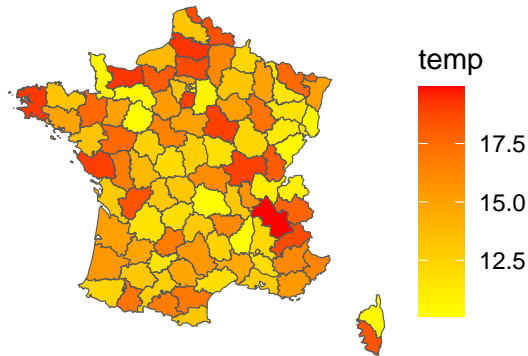
Étape ggplot

```
> 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.
- C'est cette classe qui conduit la représentation avec `plot` ou `geom_sf` :
 - `point` ou `multipoint` points pour localiser un lieu
 - `polygon` ou `multipolygon` contours pour représenter des frontières.
 - `linestring` ou `multilinestring` lignes pour représenter des fleuves, des routes...
- 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 `point`


```

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

```

- Création d'un objet **sfc** (liste avec des caractéristiques géométriques)

```

> 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 v1 / 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"

```

3.3 Cartes interactives avec leaflet

Fonds de carte

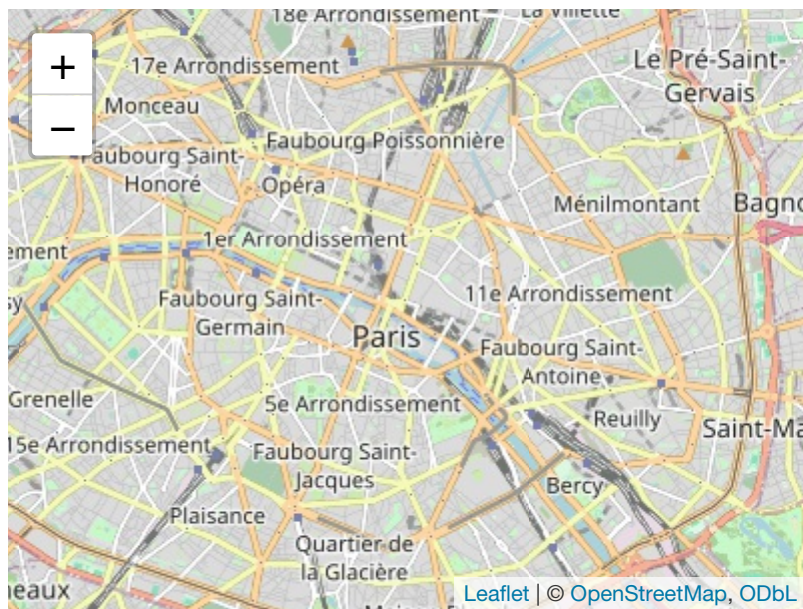
- **Leaflet** est une des librairies open-source JavaScript les plus populaires pour faire des **cartes interactives**.
- *Documentation*: <https://rstudio.github.io/leaflet/>

```
> 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(providers$Esri.NatGeoWorldMap) |>  
+   setView(lng = Paris[1], lat = Paris[2], zoom = 12)
```



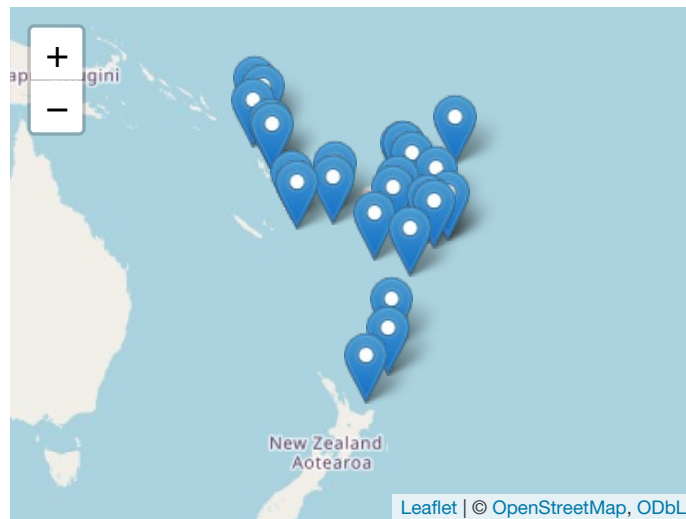
Avec des données

- 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))
```

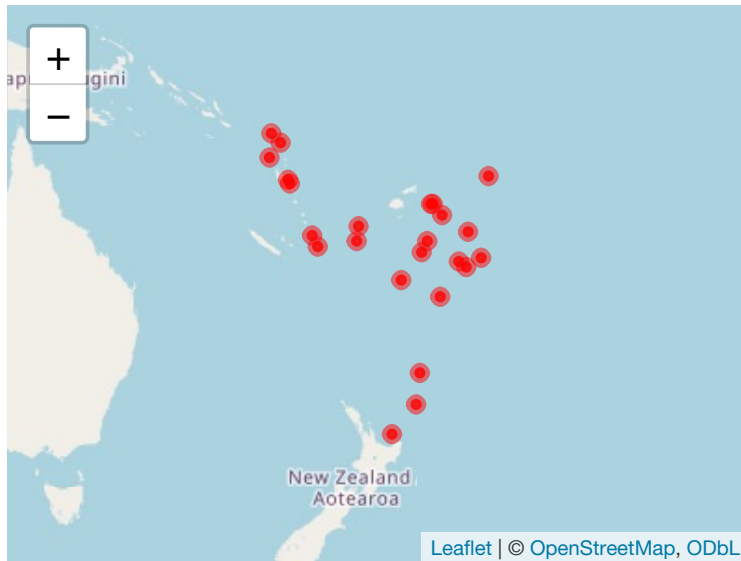


Remarque

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")
```



Des polygones utilisant sf

```
> states <- spData::us_states
> states
Simple feature collection with 49 features and 6 fields
Geometry type: MULTIPOLYGON
Dimension: XY
Bounding box: xmin: -124.7042 ymin: 24.55868 xmax: -66.9824 ymax: 49.38436
Geodetic CRS: NAD83
First 10 features:
```

	GEOID	NAME	REGION	AREA	total_pop_10
1	01	Alabama	South	133709.27 [km ²]	4712651
2	04	Arizona	West	295281.25 [km ²]	6246816
3	08	Colorado	West	269573.06 [km ²]	4887061
4	09	Connecticut	Northeast	12976.59 [km ²]	3545837
5	12	Florida	South	151052.01 [km ²]	18511620
6	13	Georgia	South	152725.21 [km ²]	9468815
7	16	Idaho	West	216512.66 [km ²]	1526797
8	18	Indiana	Midwest	93648.40 [km ²]	6417398
9	20	Kansas	Midwest	213037.08 [km ²]	2809329
10	22	Louisiana	South	122345.76 [km ²]	4429940

```
total_pop_15 geometry
1 4830620 MULTIPOLYGON (((-88.20006 3...
2 6641928 MULTIPOLYGON (((-114.7196 3...
3 5278906 MULTIPOLYGON (((-109.0501 4...
4 3593222 MULTIPOLYGON (((-73.48731 4...
5 19645772 MULTIPOLYGON (((-81.81169 2...
```

```

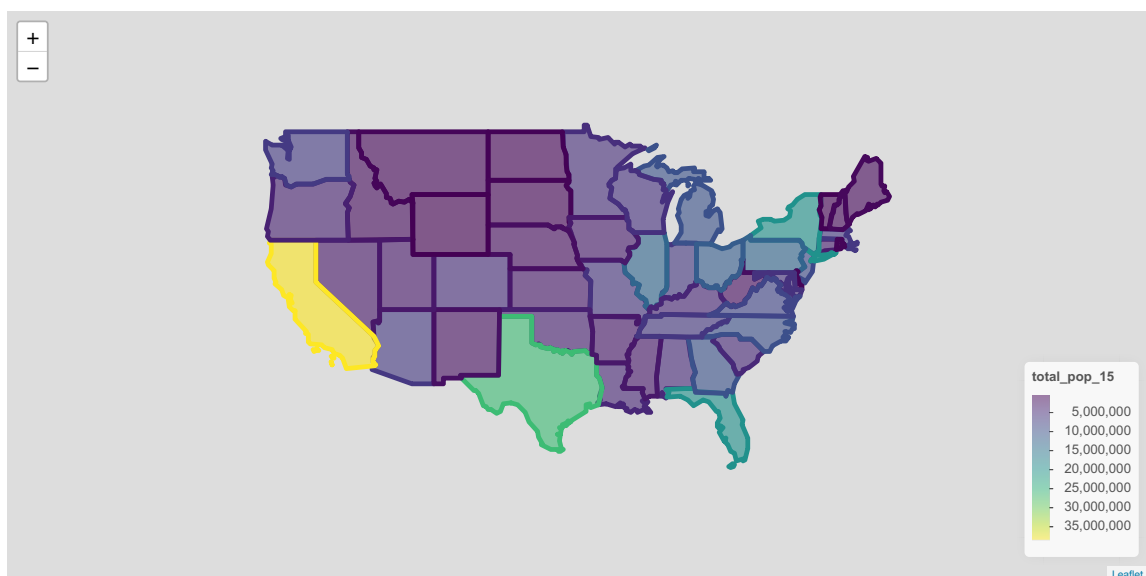
6      10006693 MULTIPOLYGON (((-85.60516 3...
7      1616547 MULTIPOLYGON (((-116.916 45...
8      6568645 MULTIPOLYGON (((-87.52404 4...
9      2892987 MULTIPOLYGON (((-102.0517 4...
10     4625253 MULTIPOLYGON (((-92.01783 2...

```

```

> pal1 <- colorNumeric(palette = c("viridis"),domain = states$total_pop_15)
> leaflet(states) |>
+   addPolygons(color=~pal1(total_pop_15),
+               popup=~str_c(as.character(NAME),
+                             as.character(total_pop_15),sep=" : "),
+               fillOpacity = 0.6,
+               opacity = 1) |>
+   addLegend(pal=pal1,value=~total_pop_15,position="bottomright")

```



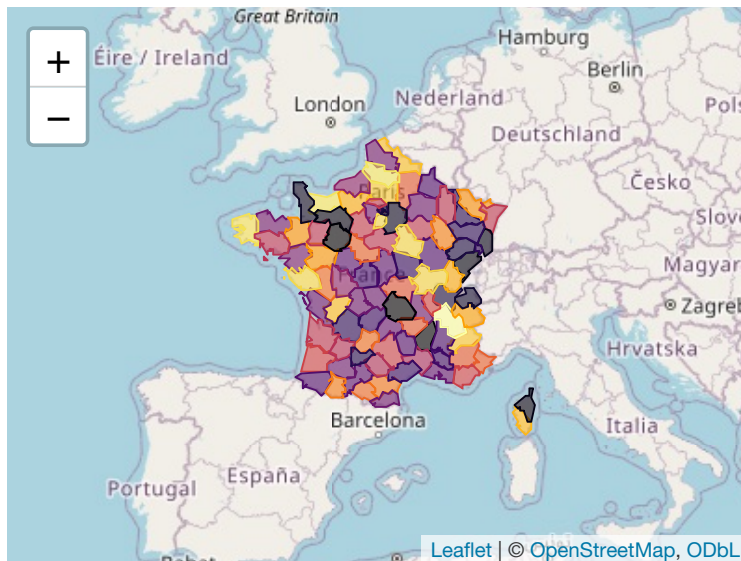
L'exemple des températures

```

> dpt2 <- st_transform(dpt1,crs=4326)
> pal2 <- colorNumeric(palette = c("inferno"),domain = dpt2$temp)
> leaflet() |> addTiles() |>

```

```
+ addPolygons(data = dpt2,color=~pal2(temp),fillOpacity = 0.6,
+             stroke = TRUE,weight=1,
+             popup=~paste(as.character(NOM_DEPT),
+                           as.character(temp),sep=" : "),
+             opacity = 1)
```



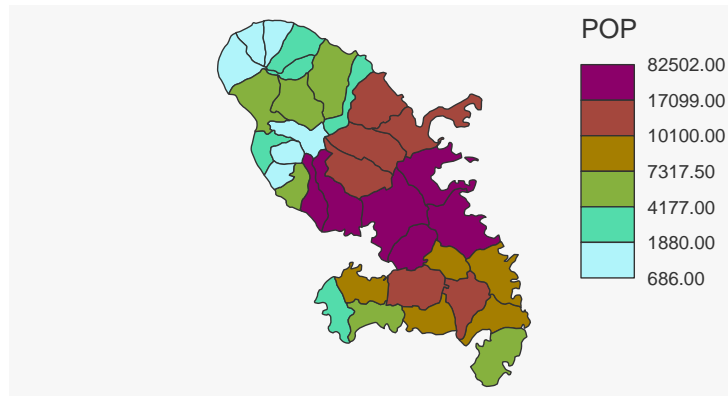
3.4 Autres packages carto

mapsf

```
> library(mapsf)
> mtq <- mf_get_mtg()
> mtq |> select(3,4,8) |> head()
Simple feature collection with 6 features and 2 fields
Geometry type: MULTIPOLYGON
Dimension: XY
Bounding box: xmin: 695444 ymin: 1598818 xmax: 717731 ymax: 1645182
Projected CRS: WGS 84 / UTM zone 20N
  LIBGEO POP geom
1 L'Ajoupa-Bouillon 1902 MULTIPOLYGON (((699261 1637...
2 Les Anses-d'Arlet 3737 MULTIPOLYGON (((709840 1599...
3 Basse-Pointe 3357 MULTIPOLYGON (((697602 1638...
4 Le Carbet 3683 MULTIPOLYGON (((702229 1628...
5 Case-Pilote 4458 MULTIPOLYGON (((698805 1621...
6 Le Diamant 5976 MULTIPOLYGON (((709840 1599...
```



```
> #hcl.pals(type="sequential")
> mf_map(x=mtq,var="POP",type="choro",pal="Hawaii")
```

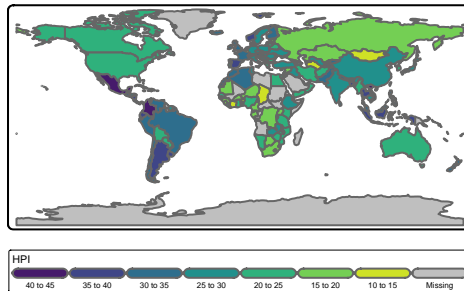


tmap

```
> library(tmap)
> #tmap_mode("view")
> data("World")
> World |> select(2,15,16) |> head()
Simple feature collection with 6 features and 2 fields
Geometry type: MULTIPOLYGON
Dimension: XY
Bounding box: xmin: -73.41544 ymin: -55.25 xmax: 75.15803 ymax: 42.68825
Geodetic CRS: WGS 84
```

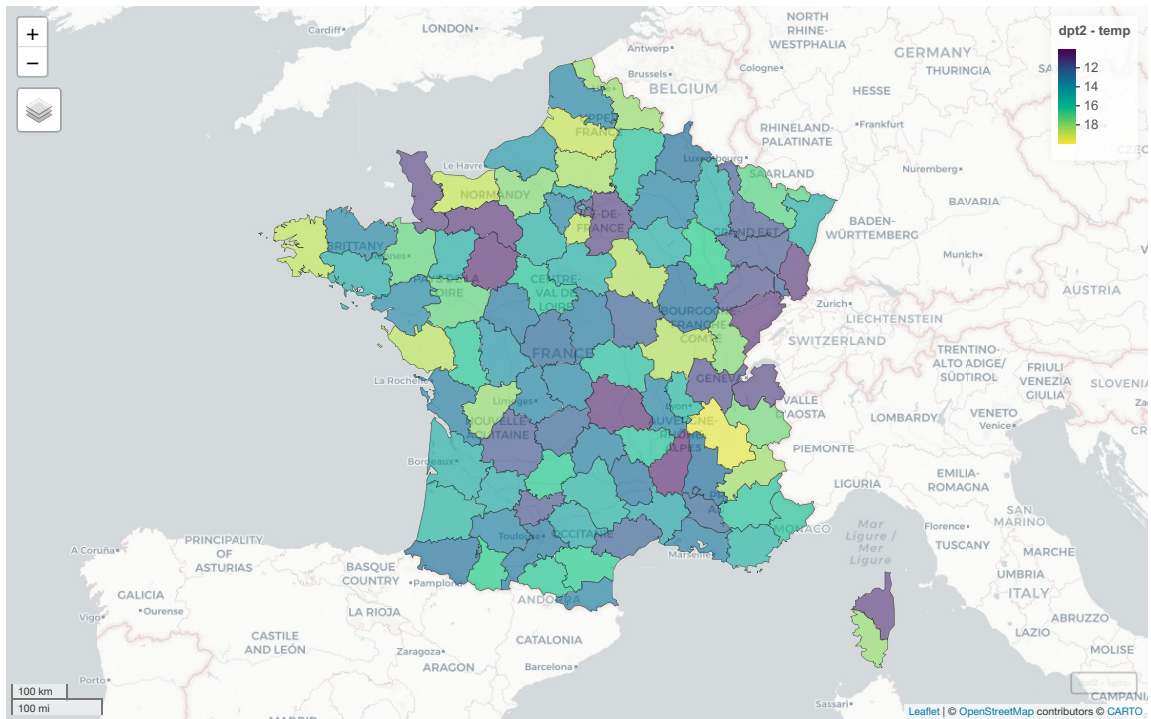
	name	HPI	geometry
1	Afghanistan	20.22535	MULTIPOLYGON (((61.21082 35...
2	Angola	NA	MULTIPOLYGON (((16.32653 -5...
3	Albania	36.76687	MULTIPOLYGON (((20.59025 41...
4	United Arab Emirates	NA	MULTIPOLYGON (((51.57952 24...
5	Argentina	35.19024	MULTIPOLYGON (((-65.5 -55.2...
6	Armenia	25.66642	MULTIPOLYGON (((43.58275 41...

```
> tm_shape(World) +
+   tm_polygons("HPI",
+               palette="viridis",
+               legend.is.portrait=FALSE,
+               legend.reverse=TRUE)
```



mapview

```
> library(mapview)
> mapview(dpt2,zcol="temp",
+         popup=leaflet::popupTable(dpt2, zcol = c("NOM_DEPT", "temp")))
```



Références

- **mapsf** : <https://cran.r-project.org/web/packages/mapsf/vignettes/mapsf.html>
- **tmap** : <https://cran.r-project.org/web/packages/tmap/vignettes/tmap-getstarted.html>
- *Tutoriel thinkr* :
 - <https://thinkr.fr/cartographie-interactive-comment-visualiser-mes-donnees-spatiales-de-maniere-dynamique-avec-leaflet/>
 - <https://thinkr.fr/cartographie-interactive-avec-r-la-suite/>

4 Quelques outils de visualisation dynamiques

Des packages R

- Graphiques classiques avec **rAmCharts**, **plotly** et **ggiraph**.

- Graphes avec `visNetwork`.
- Tableaux de bord avec `flexdashboard`.

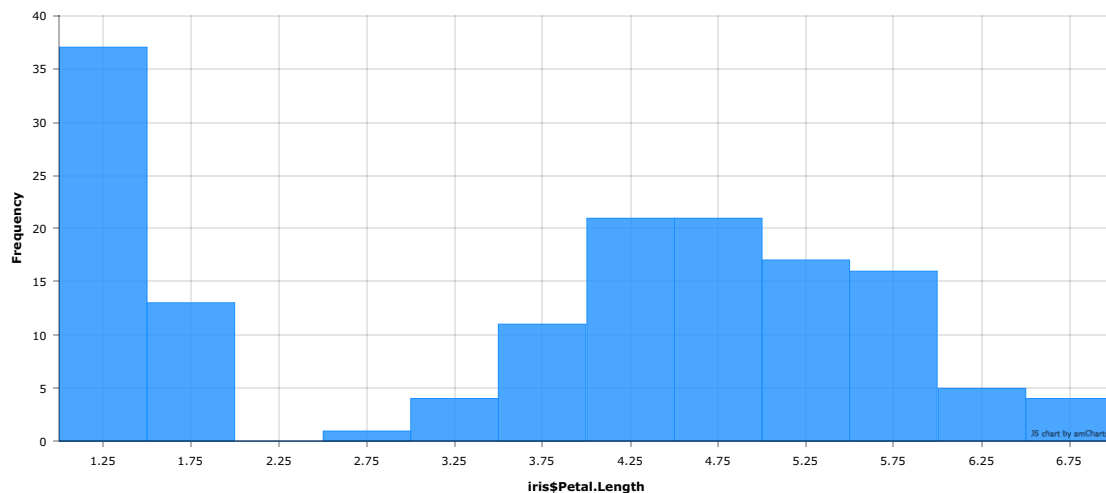
4.1 rAmCharts, plotly et ggiraph

rAmCharts

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

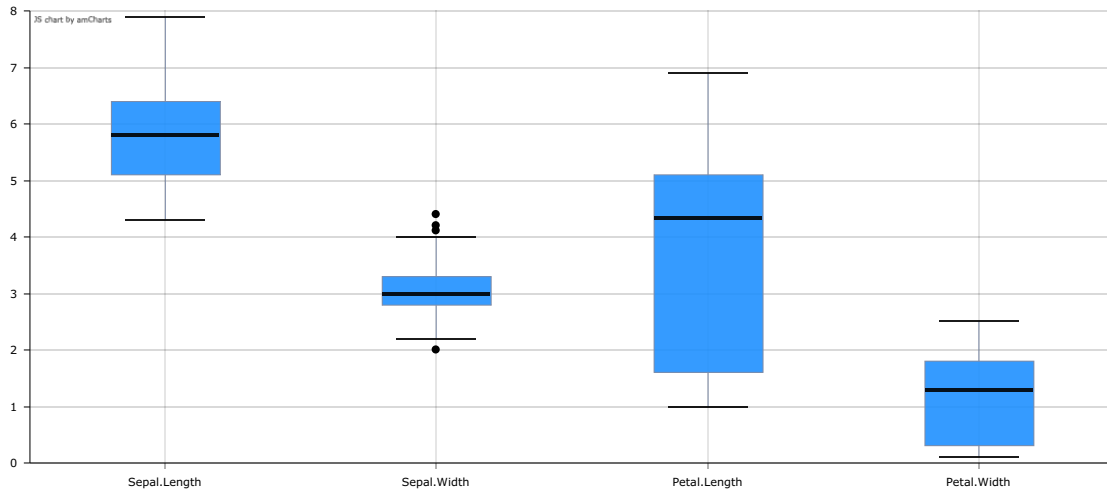
rAmCharts Histogramme

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



rAmcharts Boxplot

```
> amBoxplot(iris)
```

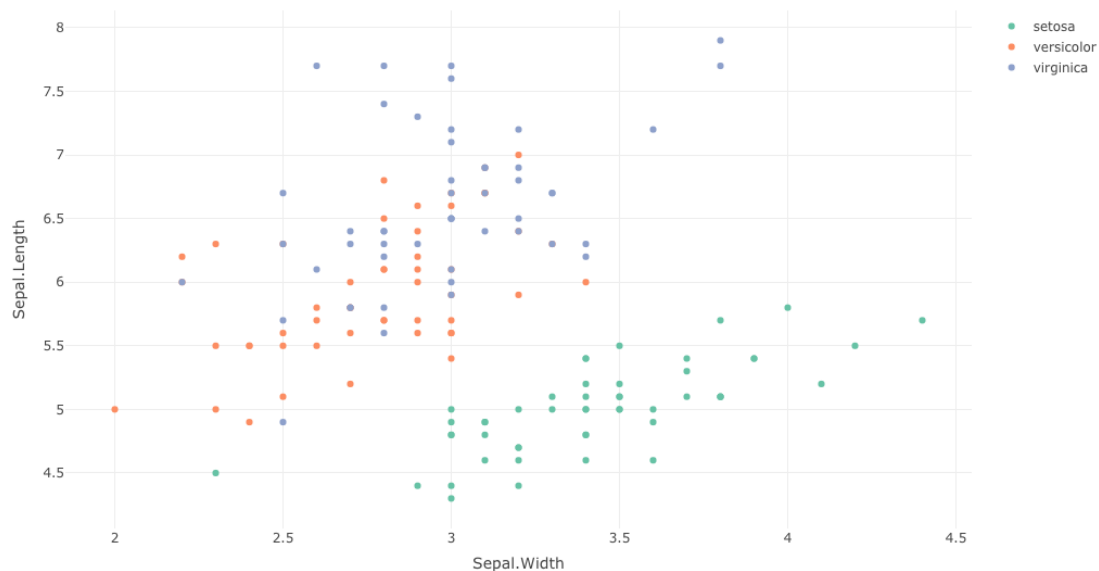


Plotly

- Package **R** pour créer des *graphes dynamiques* 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()
```

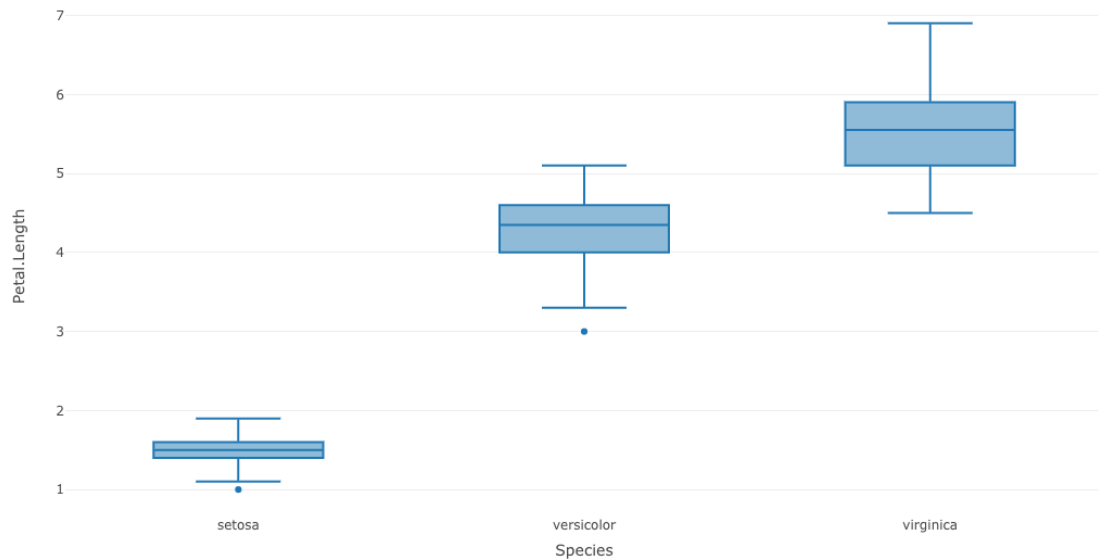
ggiraph

- Extension de `ggplot2` pour des graphes *dynamiques* et *interactifs*.
- Basé sur les fonctions `ggplot2` avec ajout du suffixe `_interactive`.
- *Documentation* : <https://www.ardata.fr/ggiraph-book/>

Le principe

- Création du graphe :

```
> library(ggiraph)
> p <- ggplot(data=iris) +
+   aes(x=Sepal.Length,y=Sepal.Width,
+       tooltip=Petal.Width,data_id=Species)+
+   geom_point_interactive(size=1,hover_nearest=TRUE)
```

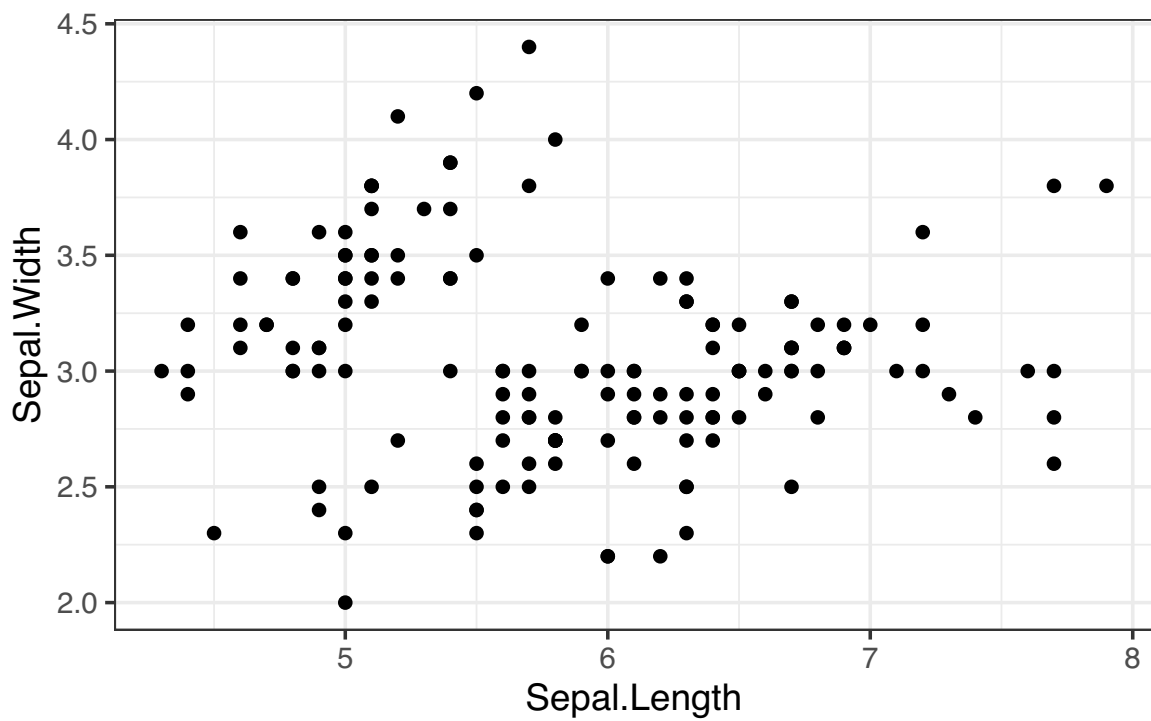


Interprétation

- `data_id=Species` : identifie les points de la même espèce que celui où se trouve la souris.
- `tooltip=Petal.Width` : affiche la largeur de pétale du point où se trouve la souris.

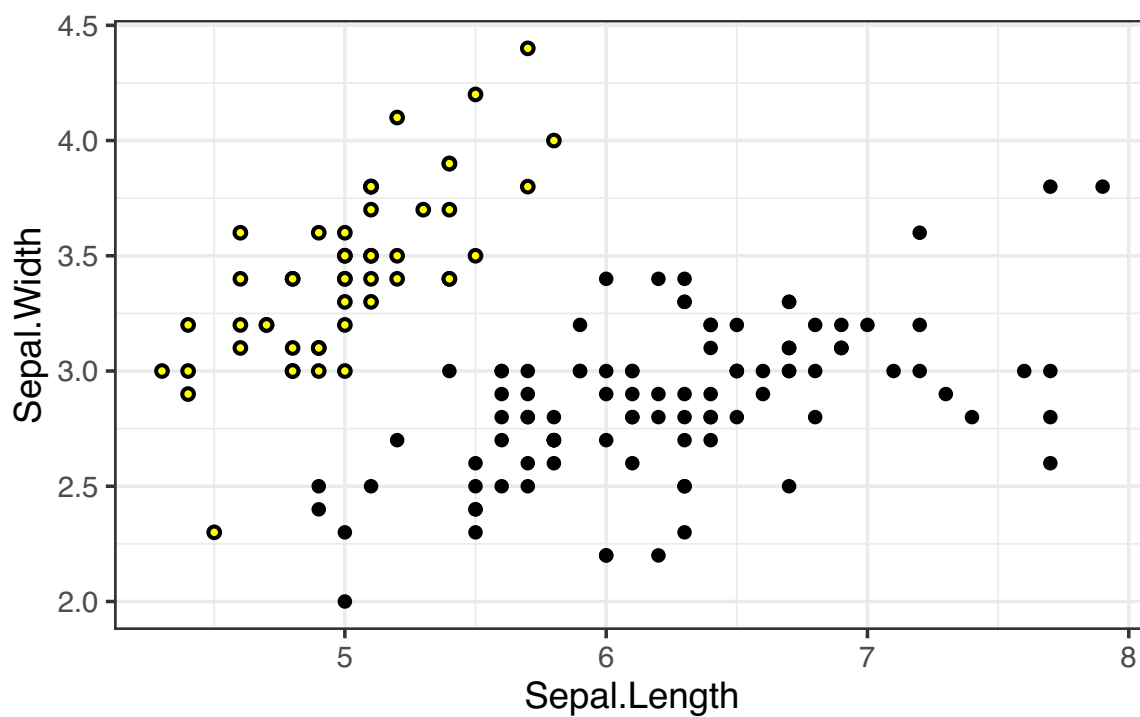
- Visualisation avec la fonction `girafe`

```
> girafe(ggobj=p)
```



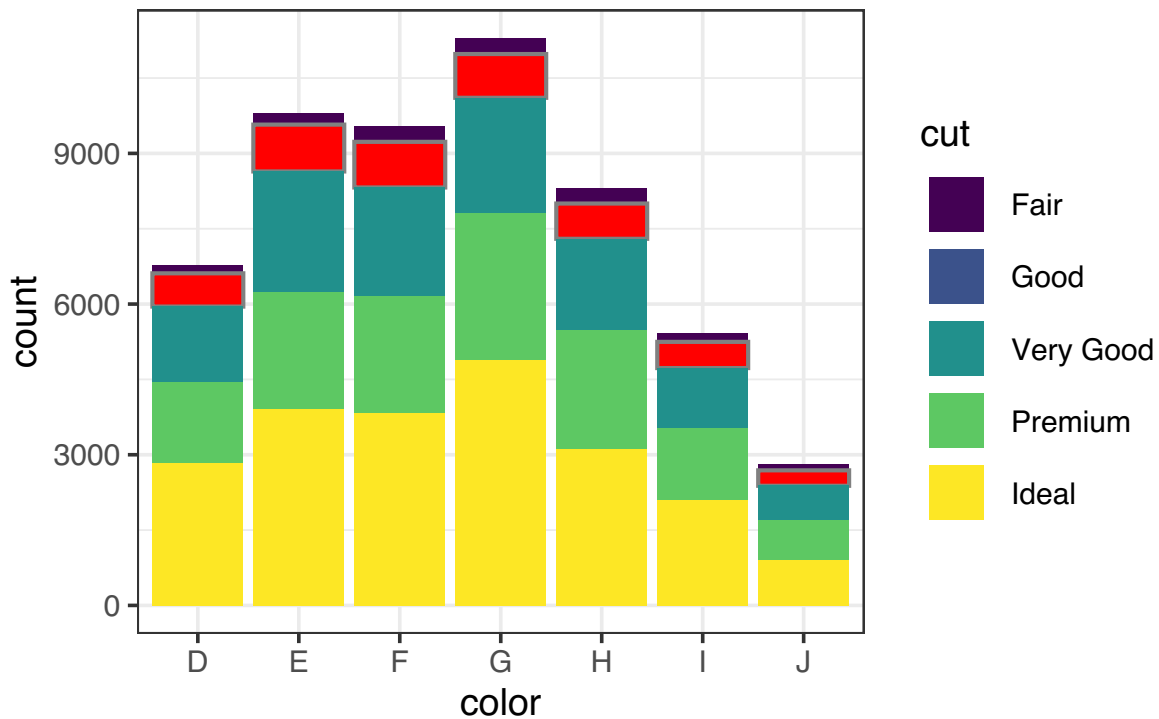
Améliorer avec du css

```
> preselection <- iris$Species[1]
> girafe(ggobj = p,
+       options=list(
+         opts_hover(css="fill:blue;stroke:black;stroke-width:2px;"),
+         opts_selection(
+           css="fill:yellow;stroke:black;stroke-width:1px;",
+           selected=preselection,type="multiple",only_shiny = FALSE)))
```

Autre exemple

```
> p1 <- ggplot(
+   data = diamonds,
+   mapping = aes(x = color, fill = cut, data_id = cut)
+ ) +
+   geom_bar_interactive(
+     aes(tooltip = sprintf("%s: %.0f", fill, after_stat(count))),
+     size = 3
+   )
> girafe(ggobj = p1,
+   options=list(
+     opts_selection(
+       selected="Good",
+       only_shiny = FALSE
+     )
+   )
+ )
```



4.2 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...)
- Un individu = *un nœud* et une connexion = *une arête*.

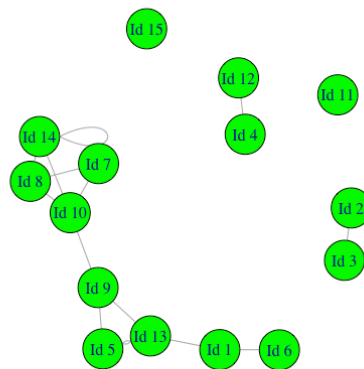
```
> set.seed(123)
> nodes <- data.frame(id = 1:15, label = paste("Id", 1:15))
> edges <- data.frame(from = trunc(runif(15)*(15-1))+1,
+                     to = trunc(runif(15)*(15-1))+1)
> head(edges)
  from to
1    5 13
2   12  4
3    6  1
4   13  5
```

```
5  14 14
6   1 13
```

Graphe statique : le package **igraph**

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

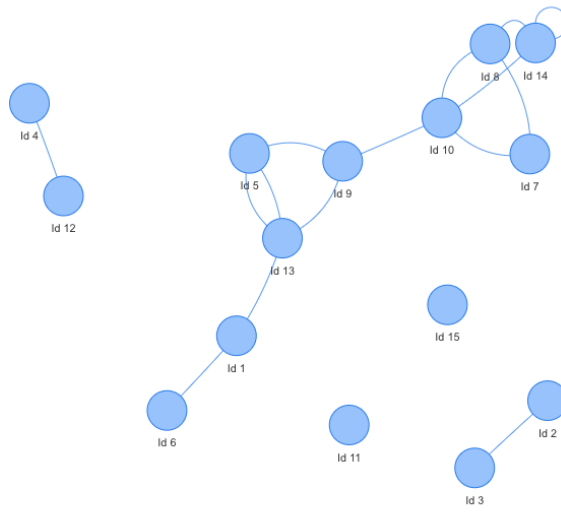
```
> library(igraph)
> net <- graph_from_data_frame(d=edges, vertices=nodes, directed=F)
> plot(net,vertex.color="green",vertex.size=25)
```



Graph dynamique : le package **visNetwork**

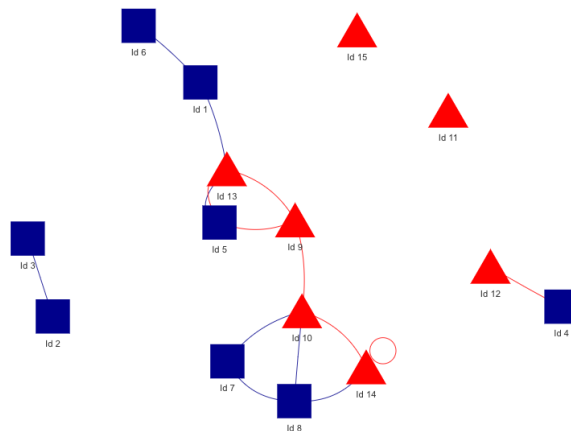
- *Référence* : <https://datastorm-open.github.io/visNetwork/interaction.html>

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



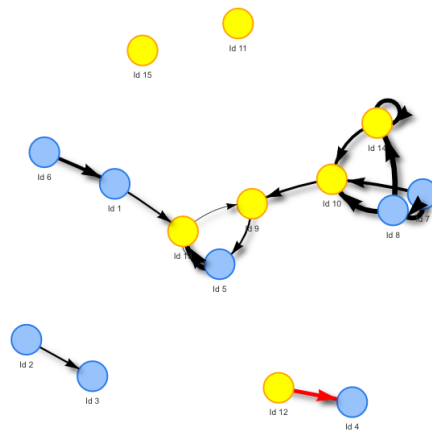
Nodes color

```
> nodes$group <- c(rep("A",8),rep("B",7))
> visNetwork(nodes,edges) |>
+   visGroups(groupname = "A", color = "darkblue",
+             shape = "square") |>
+   visGroups(groupname = "B", color = "red",
+             shape = "triangle")
```



Edeges width

```
> edges$width <- round(runif(nrow(edges),1,10))
> visNetwork(nodes,edges) |>
+   visEdges(shadow = TRUE,
+     arrows =list(to = list(enabled = TRUE)),
+     color = list(color = "black", highlight = "red"))
```



4.3 Tableau de bord avec flexdashboard

- 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
---
```

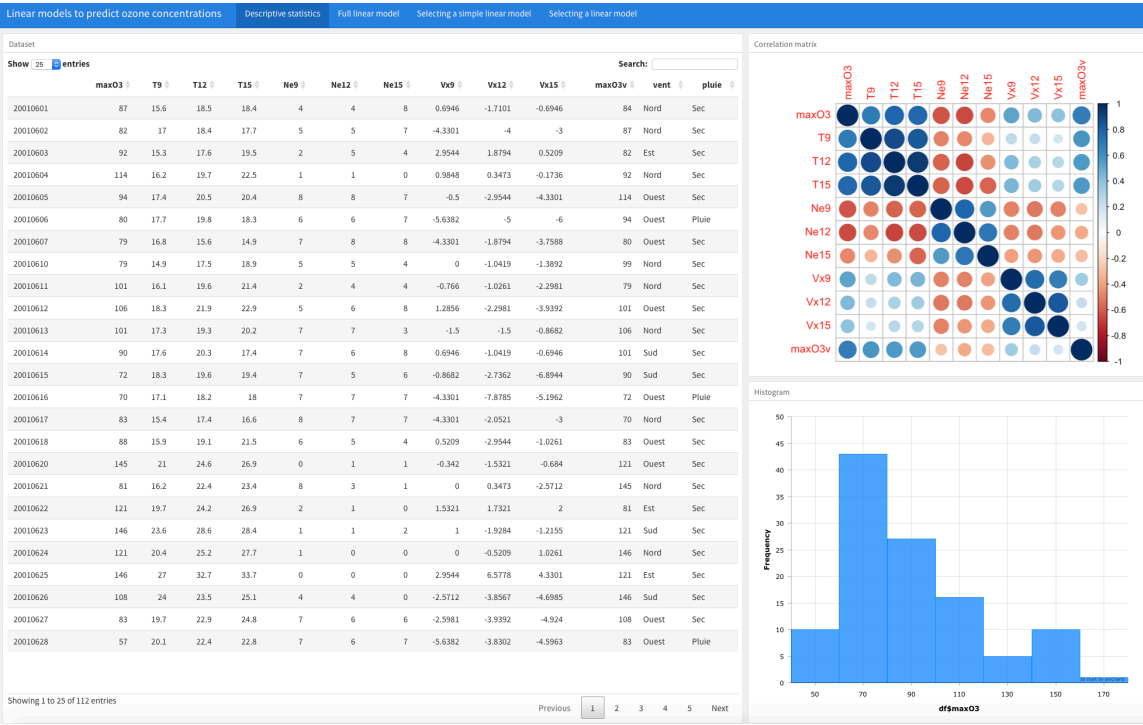
- Le thème par défaut peut être remplacé par d'*autres thèmes* (cosmo, bootstrap, cerulean...) (voir [ici](#)). Il suffit d'ajouter

```
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



Correlation matrix

Histogram