

Visualisation

2IA - Data science

Nicolas Sutton-Charani



Why data visualization in 2IA department ?

- ▶ Data are more and more complex
- ▶ Models are more and more complex
- ▶ Conclusions are more and more complex.
- ▶ We need visualization tools to :
 - ▶ describe data
 - ▶ calibrate models
 - ▶ present results and conclusions of the study.

Consequence

Visualization reveals crucial throughout a statistical study.

1. Nice plots with ggplot
 - 1.1 Conventional R plots
 - 1.2 ggplot2 grammar
2. Dynamic plots
 - 2.1 rAmCharts
 - 2.2 plotly
3. Mapping
 - 3.1 ggmap
 - 3.2 Shapefile contours with sf
 - 3.3 Interactive maps with leaflet
4. Networks plots
5. Dashboards and shiny apps
 - 5.1 flexdashboards
 - 5.2 shiny apps
6. Project

Plan

1. Nice plots with ggplot

1.1 Conventional R plots

1.2 ggplot2 grammar

2. Dynamic plots

2.1 rAmCharts

2.2 plotly

3. Mapping

3.1 ggmap

3.2 Shapefile contours with sf

3.3 Interactive maps with leaflet

4. Networks plots

5. Dashboards and shiny apps

5.1 flexdashboards

5.2 shiny apps

6. Project

- ▶ Graphs are often the starting point for statistical analysis.
- ▶ One of the main advantages of R is how easy it is for the user to create many different kinds of graphs.
- ▶ We begin by a (short) review on conventional graphs,
- ▶ followed by an examination of some more complex representations, especially with ggplot2 package.

Visualisation

- └ Nice plots with ggplot
- └ Conventional R plots

Plan

1. Nice plots with ggplot

1.1 Conventional R plots

1.2 ggplot2 grammar

2. Dynamic plots

2.1 rAmCharts

2.2 plotly

3. Mapping

3.1 ggmap

3.2 Shapefile contours with sf

3.3 Interactive maps with leaflet

4. Networks plots

5. Dashboards and shiny apps

5.1 flexdashboards

5.2 shiny apps

6. Project

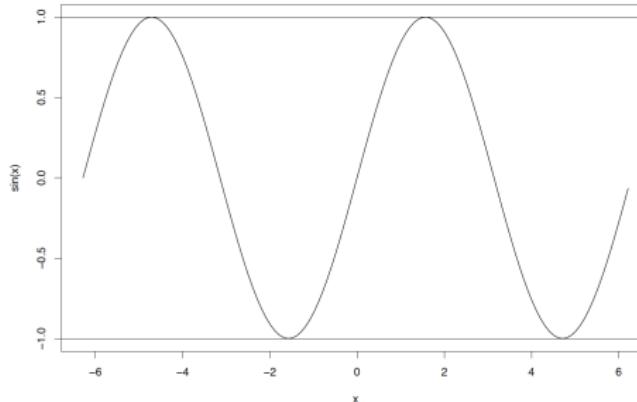
Visualisation

- └ Nice plots with ggplot
- └ Conventional R plots

The plot function

- ▶ It is a **generic** function to represent **all kind of data**.
- ▶ For a **scatter plot**, we have to specify a vector for the x-axis and a vector for the y-axis.

```
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))
```



Graphs for datasets

- ▶ Many kind of representations are needed according to the variables we want to visualize.
- ▶ Histogram for continuous variables, barplot for categorical variables.
- ▶ Scatterplot for 2 continuous variables.
- ▶ Boxplot to visualize distributions.

Fortunately

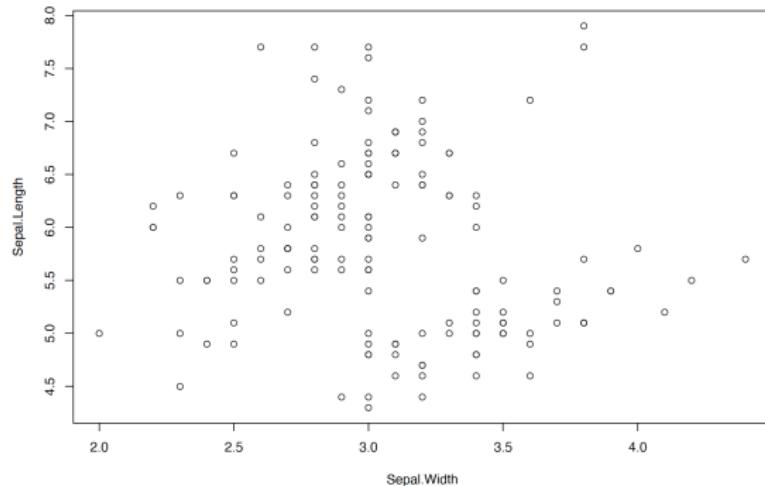
There is a R function for all representations.

Visualisation

- └ Nice plots with ggplot
- └ Conventional R plots

Scatterplot with dataset

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



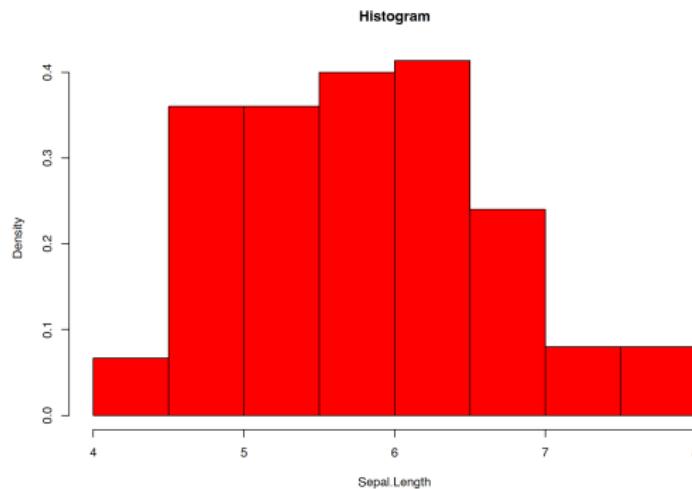
```
# same as  
plot(iris$Sepal.Width, iris$Sepal.Length)
```

Visualisation

- └ Nice plots with ggplot
- └ Conventional R plots

Histogram for continuous variable

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

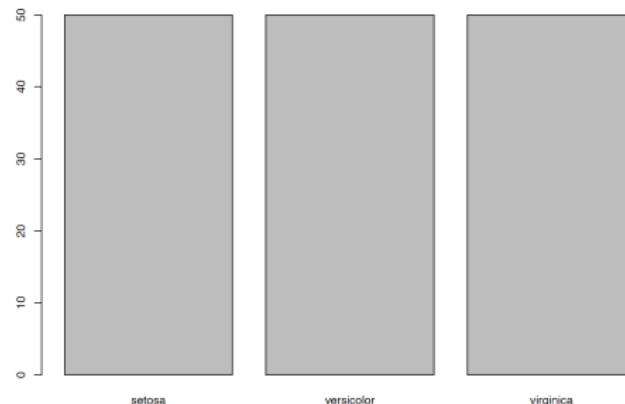


Visualisation

- └ Nice plots with ggplot
- └ Conventional R plots

Barplot for categorical variables

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

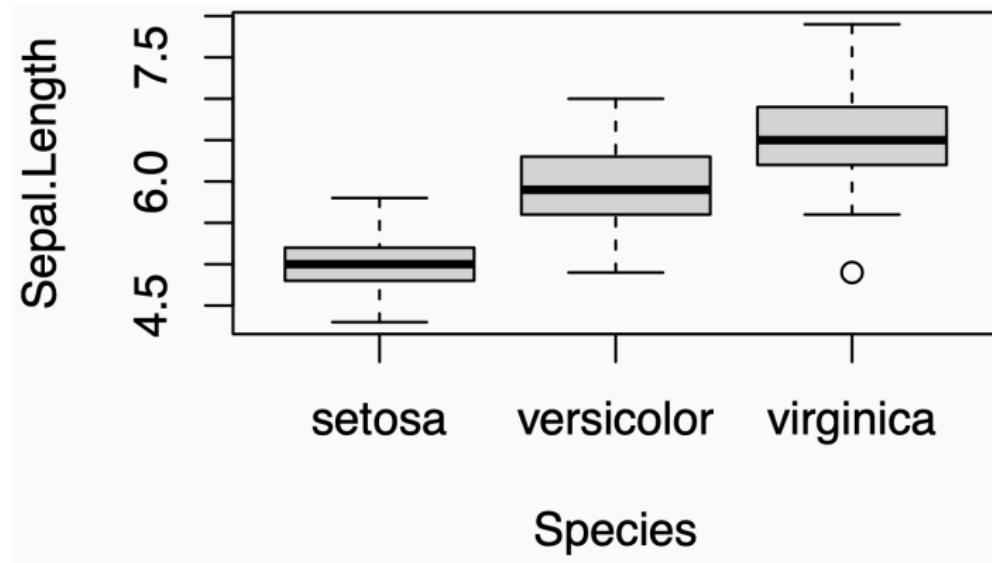


Visualisation

- └ Nice plots with ggplot
- └ Conventional R plots

Boxplots

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



Visualisation

- └ Nice plots with ggplot
 - └ ggplot2 grammar

Plan

1. Nice plots with ggplot
 - 1.1 Conventional R plots
 - 1.2 ggplot2 grammar
2. Dynamic plots
 - 2.1 rAmCharts
 - 2.2 plotly
3. Mapping
 - 3.1 ggmap
 - 3.2 Shapefile contours with sf
 - 3.3 Interactive maps with leaflet
4. Networks plots
5. Dashboards and shiny apps
 - 5.1 flexdashboards
 - 5.2 shiny apps
6. Project

Visualisation

- └ Nice plots with ggplot
 - └ ggplot2 grammar

- ▶ **ggplot2** is a plotting system for R based on the **grammar of graphics** (as dplyr to manipulate data).
- ▶ The goal is to provide a **clear syntax** for an **efficient visualization**.
- ▶ Ggplot provides "**nice**" **graphs** (nor always the case for conventional R graphs).
- ▶ Documentation : **tutorial**, **book**

Visualisation

- └ Nice plots with ggplot
 - └ ggplot2 grammar

For a given dataset, a graph is defined from many layers. We have to specify :

- ▶ the `data`
- ▶ the `variables` we want to plot
- ▶ the `type of representation` (scatterplot, boxplot. . .).

Ggplot graphs are defined from these layers. We indicate

- ▶ the data with `ggplot`
- ▶ the variables with `aes` (aesthetics)
- ▶ the kind of representation with `geom` . . .

The grammar

Main elements of the grammar are :

- ▶ **Data** : the `dataset`, it should be a dataframe or a tibble.
- ▶ **Aesthetics (aes)** : to describe the way that `variables` in the data are mapped. All the variables used in the graph should be specified in aes.
- ▶ **Geometrics (geom...)** : to control the `type` of plot.
- ▶ **Scales (scale...)** : to `control the mapping` from data to aesthetic attributes (change colors, size. . .).

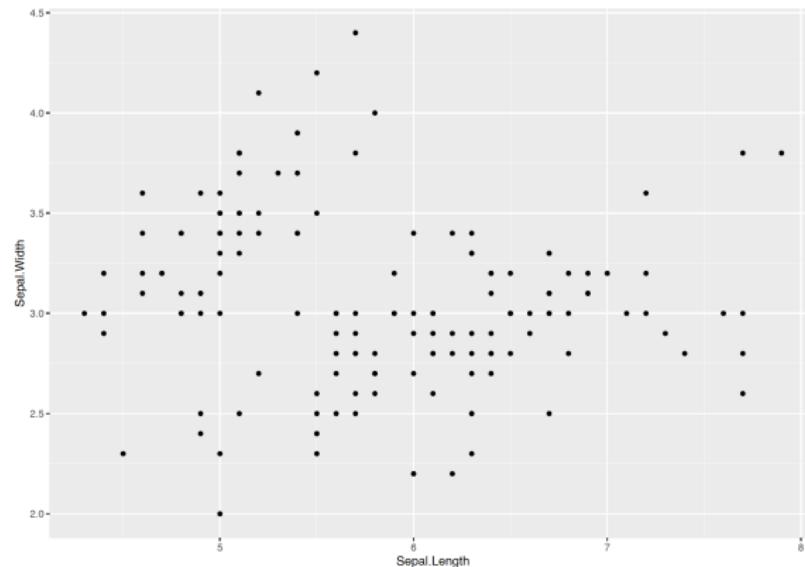
All these elements are gathered with the operator +.

Visualisation

- └ Nice plots with ggplot
 - └ ggplot2 grammar

An example

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

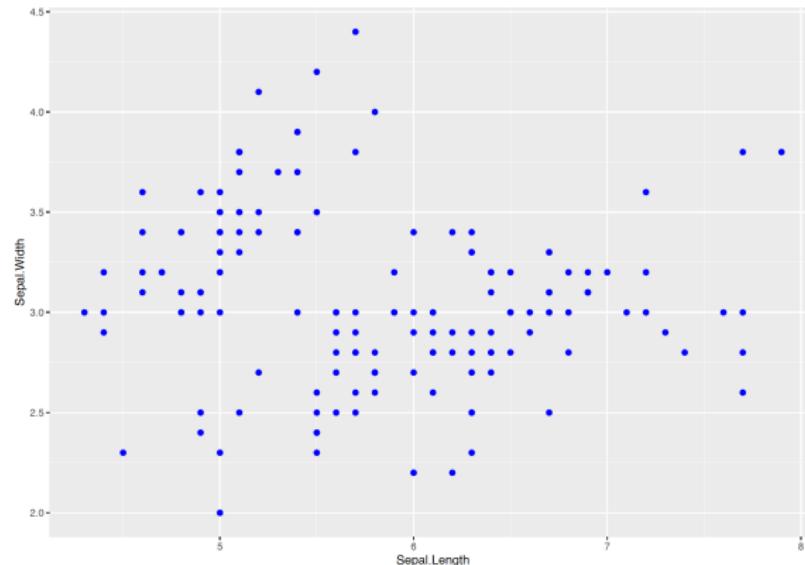


Visualisation

- └ Nice plots with ggplot
 - └ ggplot2 grammar

Color and size

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

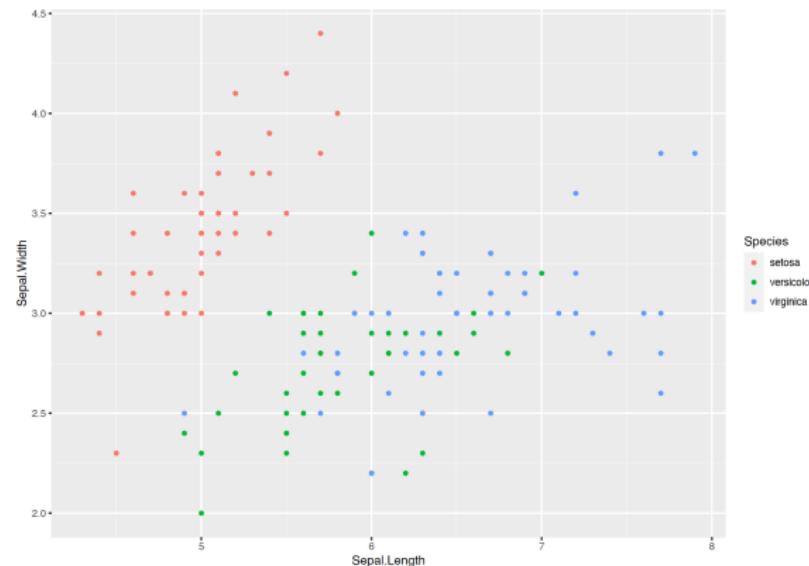


Visualisation

- └ Nice plots with ggplot
- └ ggplot2 grammar

Color by (categorical) variable

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

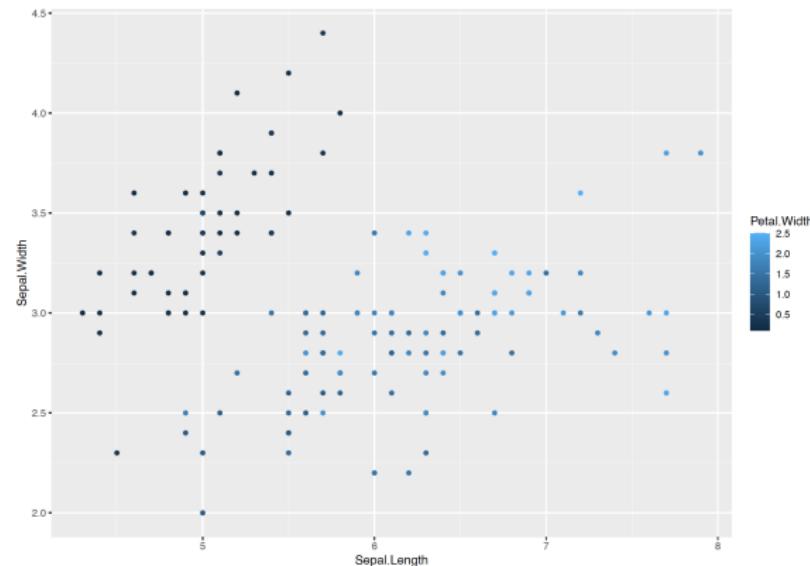


Visualisation

- └ Nice plots with ggplot
- └ ggplot2 grammar

Color by (continuous) variable

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

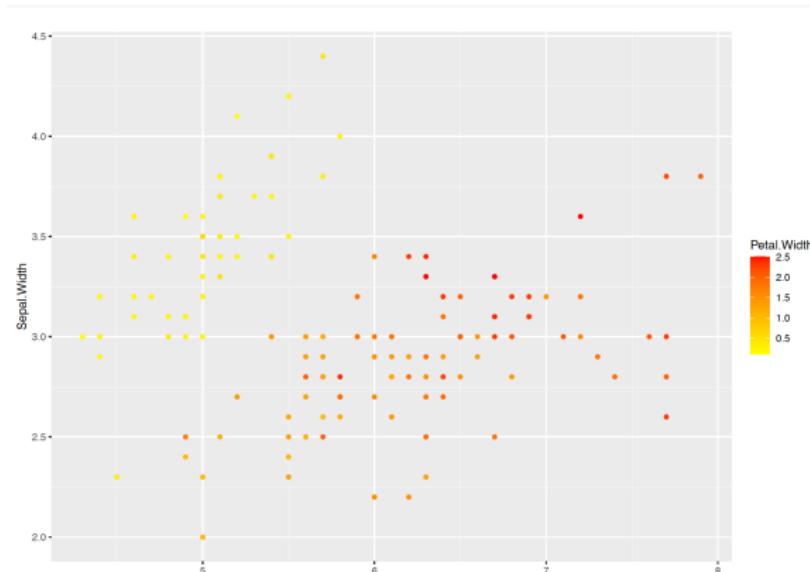


Visualisation

- └ Nice plots with ggplot
- └ ggplot2 grammar

Color by (continuous) variable

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

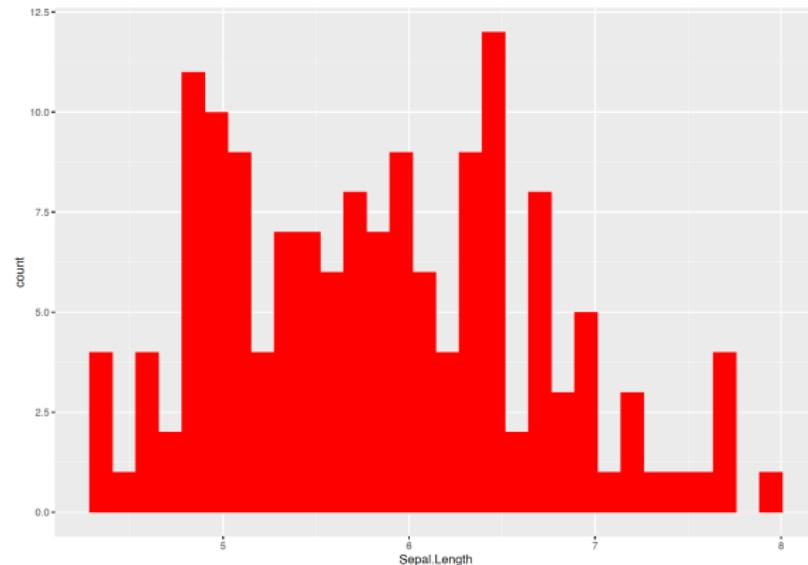


Visualisation

- └ Nice plots with ggplot
 - └ ggplot2 grammar

Histograms

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

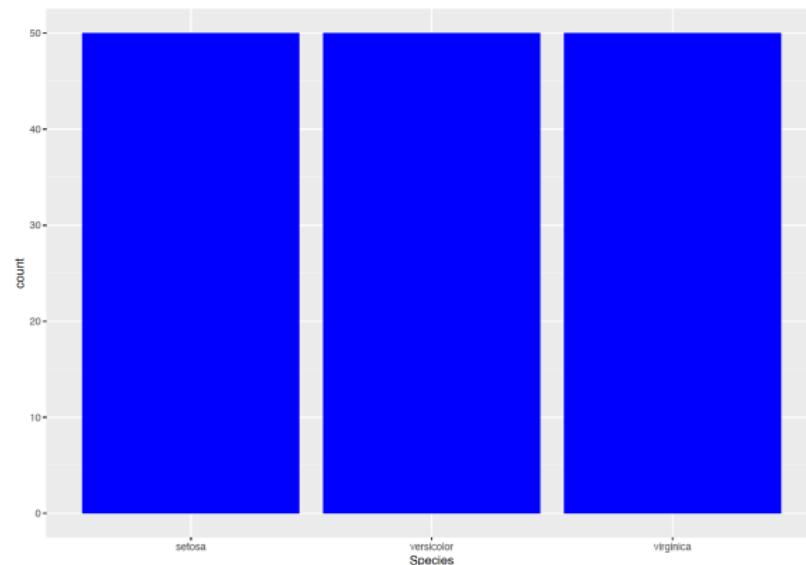


Visualisation

- └ Nice plots with ggplot
 - └ ggplot2 grammar

Barplots

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

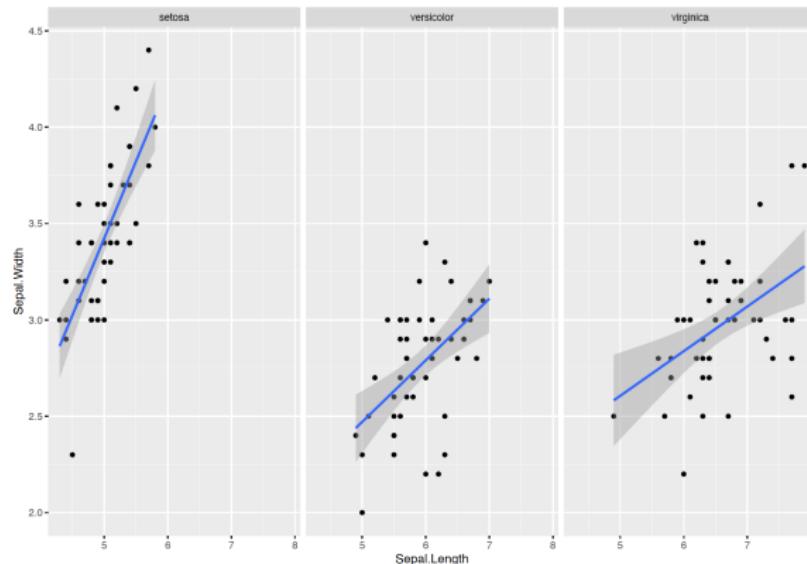


Visualisation

- └ Nice plots with ggplot
- └ ggplot2 grammar

Facetting (more "complex")

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

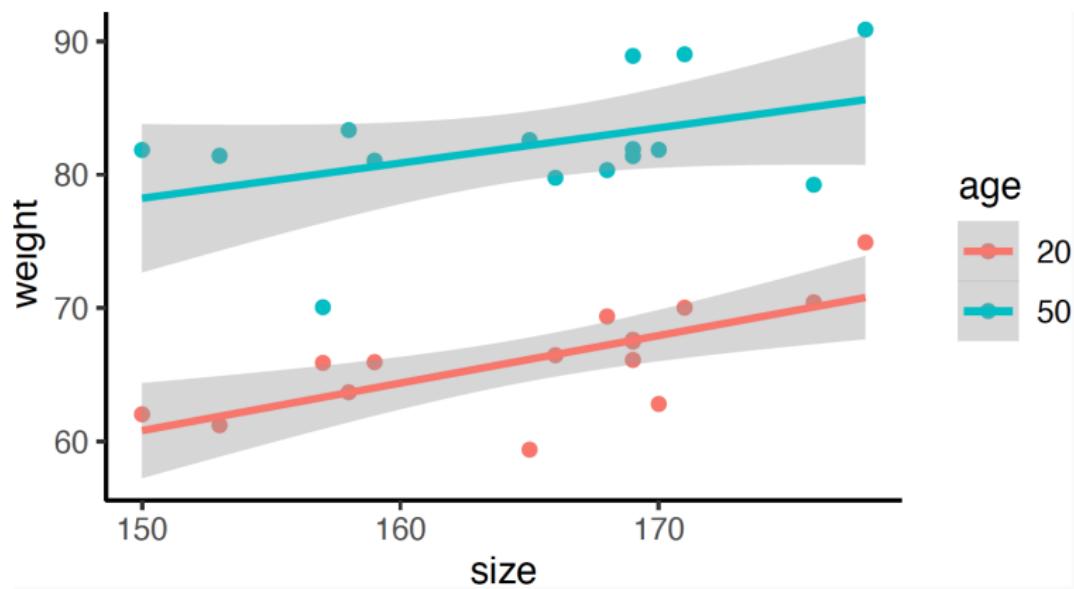


Combining ggplot with dplyr

- ▶ We often have to **work on the dataframe to obtain an efficient ggplot syntax.**
- ▶ For instance

```
> 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
```

Goal



Visualisation

- └ Nice plots with ggplot
 - └ ggplot2 grammar

dplyr step

Gather column `weight.M` and `weight.W` into one column `weight` with `pivot_longer` :

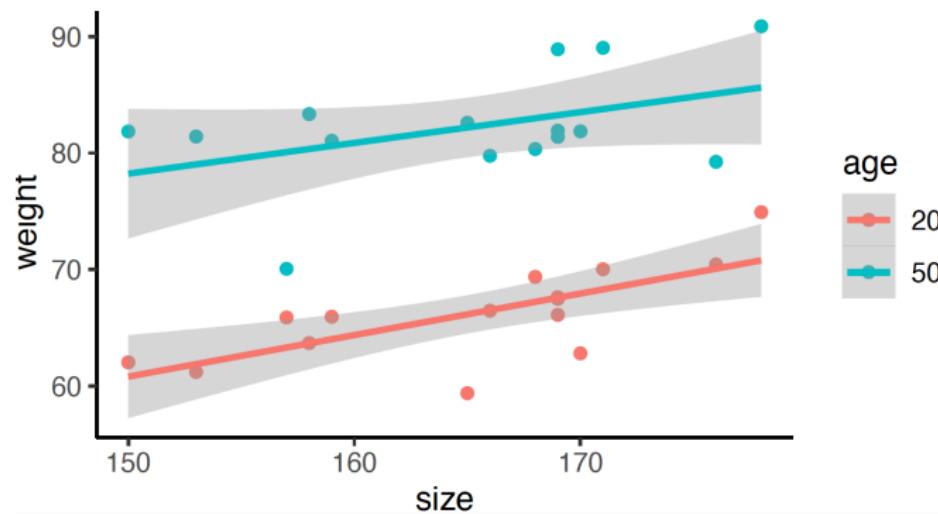
```
> df1 <- df %>% pivot_longer(-size, names_to="age", values_to="weight")
> df1 %>% head()
# A tibble: 6 x 3
  size     age      weight
  <dbl> <chr>    <dbl>
1   153 weight.20    61.2
2   153 weight.50    81.4
3   169 weight.20    67.5
4   169 weight.50    81.4
5   168 weight.20    69.4
6   168 weight.50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

Visualisation

- └ Nice plots with ggplot
- └ ggplot2 grammar

ggplot step

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



Complement : some demos

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

→ Work on **this part** of the tutorial.

Plan

1. Nice plots with ggplot
 - 1.1 Conventional R plots
 - 1.2 ggplot2 grammar
2. Dynamic plots
 - 2.1 rAmCharts
 - 2.2 plotly
3. Mapping
 - 3.1 ggmap
 - 3.2 Shapefile contours with sf
 - 3.3 Interactive maps with leaflet
4. Networks plots
5. Dashboards and shiny apps
 - 5.1 flexdashboards
 - 5.2 shiny apps
6. Project

Some R tools for dynamic visualization

- ▶ Classical charts with `rAmCharts` and `plotly`.
- ▶ Graphs with `visNetwork`.
- ▶ Dashboard with `flexdashboard`.

Plan

1. Nice plots with ggplot

1.1 Conventional R plots

1.2 ggplot2 grammar

2. Dynamic plots

2.1 rAmCharts

2.2 plotly

3. Mapping

3.1 ggmap

3.2 Shapefile contours with sf

3.3 Interactive maps with leaflet

4. Networks plots

5. Dashboards and shiny apps

5.1 flexdashboards

5.2 shiny apps

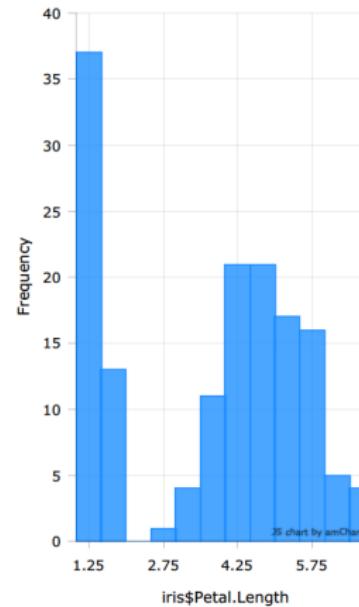
6. Project

rAmCharts

- ▶ **user-friendly** for standard graphs (scatterplot, times series, histogram. . .).
- ▶ We just have to use classical R functions with the **prefix am**.
- ▶ Examples : **amPlot**, **amHist**, **amBoxplot**.
- ▶ References :
https://datastorm-open.github.io/introduction_ramcharts/

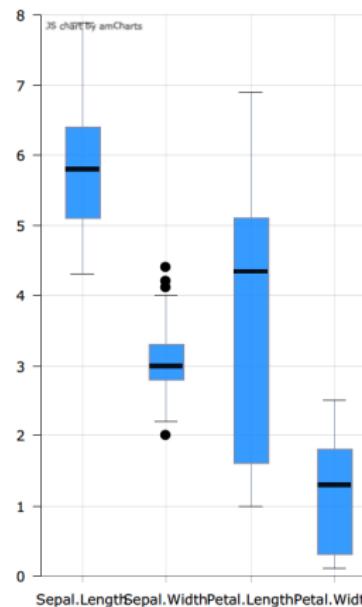
rAmCharts Histogram

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



rAmCharts Boxplot

```
amBoxplot(iris)
```



Plan

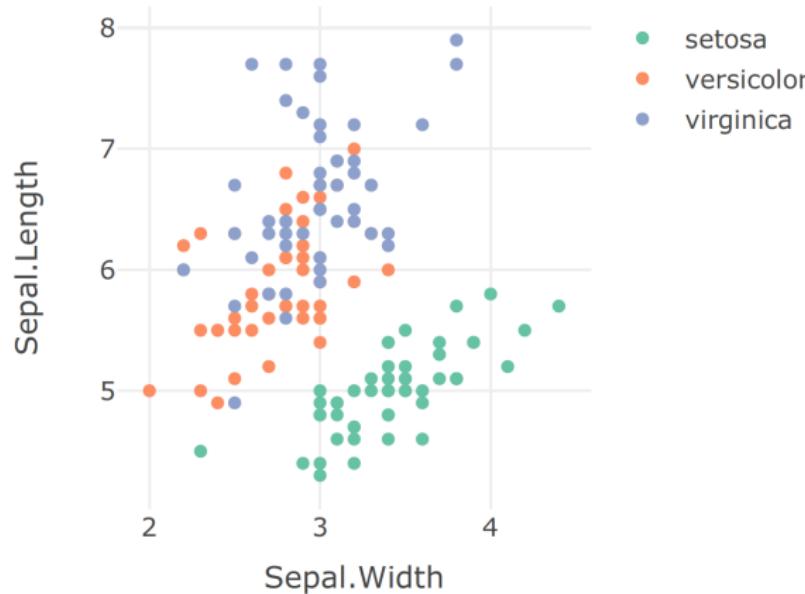
1. Nice plots with ggplot
 - 1.1 Conventional R plots
 - 1.2 ggplot2 grammar
2. Dynamic plots
 - 2.1 rAmCharts
 - 2.2 plotly
3. Mapping
 - 3.1 ggmap
 - 3.2 Shapefile contours with sf
 - 3.3 Interactive maps with leaflet
4. Networks plots
5. Dashboards and shiny apps
 - 5.1 flexdashboards
 - 5.2 shiny apps
6. Project

Plotly

- ▶ R package for creating **interactive web-based** graphs via the open source JavaScript graphing library **plotly.js**.
- ▶ Commands are expanded into **3 parts** :
 - dataset and variables (**plot_ly**)
 - madditional representations (**add_trace**, **add_markers**. . .)
 - options (axis, titles. . .) (**layout**).
- ▶ References : <https://plot.ly/r/reference/>

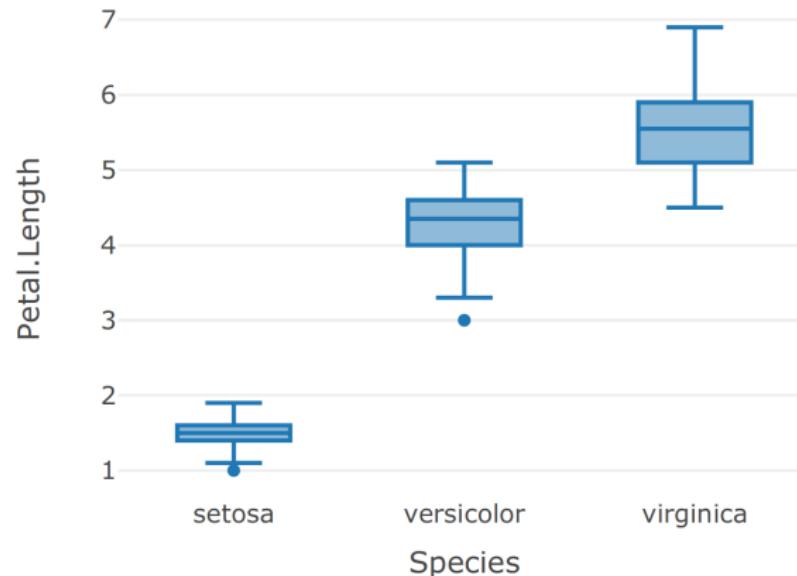
Scatter plot

```
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()
```



Plan

1. Nice plots with ggplot
 - 1.1 Conventional R plots
 - 1.2 ggplot2 grammar
2. Dynamic plots
 - 2.1 rAmCharts
 - 2.2 plotly
3. Mapping
 - 3.1 ggmap
 - 3.2 Shapefile contours with sf
 - 3.3 Interactive maps with leaflet
4. Networks plots
5. Dashboards and shiny apps
 - 5.1 flexdashboards
 - 5.2 shiny apps
6. Project

Introduction to mapping in R

- ▶ Many applications require `maps` to visualize data or results of a model
- ▶ Many `R packages` : `ggmap`, `RgoogleMaps`, `maps`, . . .
- ▶ In this part : `ggmap`, `sf` (**static** mapping) and `leaflet` (**interactive** mapping).

Plan

1. Nice plots with ggplot
 - 1.1 Conventional R plots
 - 1.2 ggplot2 grammar
2. Dynamic plots
 - 2.1 rAmCharts
 - 2.2 plotly
3. Mapping
 - 3.1 ggmap
 - 3.2 Shapefile contours with sf
 - 3.3 Interactive maps with leaflet
4. Networks plots
5. Dashboards and shiny apps
 - 5.1 flexdashboards
 - 5.2 shiny apps
6. Project

Syntax

- ▶ Similar to `ggplot` . . .

- ▶ Instead of

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

- ▶ use

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

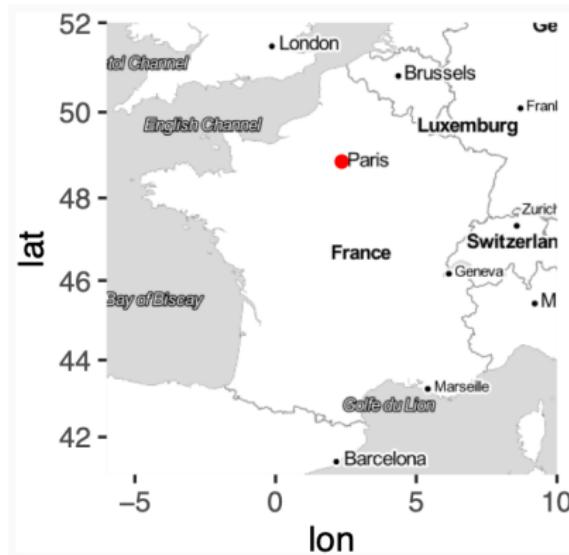
Background map

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



Adding informations with 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")
```



Plan

1. Nice plots with ggplot

1.1 Conventional R plots

1.2 ggplot2 grammar

2. Dynamic plots

2.1 rAmCharts

2.2 plotly

3. Mapping

3.1 ggmap

3.2 Shapefile contours with sf

3.3 Interactive maps with leaflet

4. Networks plots

5. Dashboards and shiny apps

5.1 flexdashboards

5.2 shiny apps

6. Project

sf package

- ▶ **Ggmap** : ok for easy maps (background with some points).
- ▶ Not sufficient for **more complex representations** (color countries according to variables).
- ▶ **sf** allows to manage **specific tools for mapping** : **boundaries** for countries or department, coordinate systems (latitudes-longitudes, World Geodesic System 84. . .)
- ▶ Background map with format **shapefile** (**contours = polygons**)
- ▶ Compatible with **ggplot** (**geom_sf** verb).

References

- ▶ [https://statnmap.com/fr/
2018-07-14-initiation-a-la-cartographie-avec-sf-et-compagnie/](https://statnmap.com/fr/2018-07-14-initiation-a-la-cartographie-avec-sf-et-compagnie/)
- ▶ **Vignettes** on the cran :
[https://cran.r-project.org/web/packages/sf/index.html.](https://cran.r-project.org/web/packages/sf/index.html)

Visualisation

└ Mapping

└ Shapefile contours with sf

Example

```
> 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 / 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

└ Mapping

└ Shapefile contours with sf

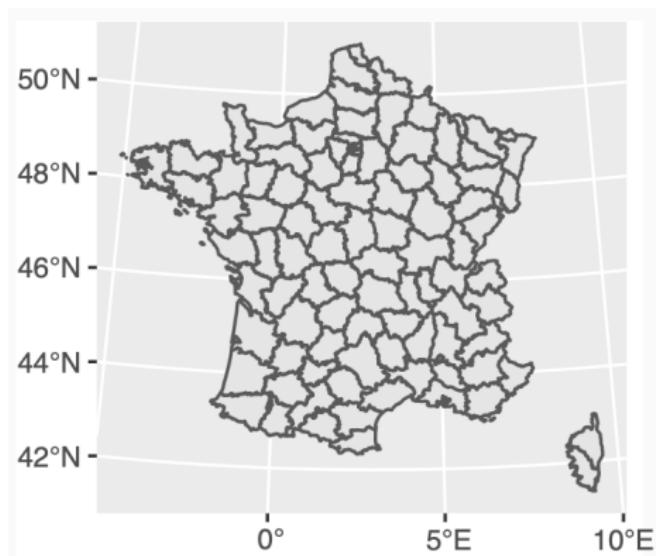
Visualize with plot

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



Visualize with ggplot

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



Adding points on the map

- ▶ Define coordinates with `st_point`

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

- ▶ Specify the `coordinate system` (4326 for lat-lon)

```
> st_crs(point) <- 4326 #lat-lon  
> point  
Geometry set for 3 features  
Geometry type: POINT  
Dimension:      XY  
Bounding box:   xmin: 2.351462 ymin: 43.29617 xmax: 5.369953 ymax: 48.85670  
Geodetic CRS:   WGS 84  
POINT (2.351462 48.8567)  
POINT (4.832011 45.75781)  
POINT (5.369953 43.29617)
```

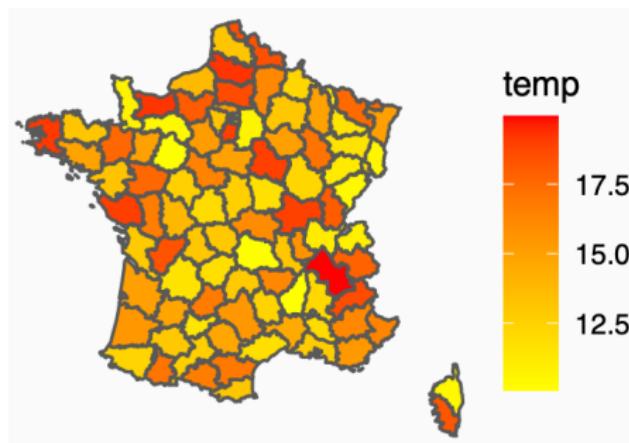
ggplot step

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



Coloring polygons

```
> 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()
```



Supplement : geometry class

- ▶ One of the main advantage of `sf` is the **geometry** class.
- ▶ It allows to conduct the representation with **plot** or **geom_sf** :
 - `point` or `multipoint` → points to locate a place ;
 - `polygon` or `multipolygon` → contours to visualize boundaries.
- ▶ Some useful functions :
 - **st_point** and **st_multipoint** : create points or sequence of points
 - **st_sfc** : create a list of **sf objects**
 - **st_crs** : specify coordinate reference system
 - **st_cast** : cast geometry to another type (convert a multipoint object to many point objects)
 - ...

► Creation of a **sf** object

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

► Creation of a **sfc** object (list of **sf** objects)

```
> 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"
```

► Extract **geometry** in a **sf** object

```
> 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"
```

→ Work on **this part** of the tutorial.

Plan

1. Nice plots with ggplot

1.1 Conventional R plots

1.2 ggplot2 grammar

2. Dynamic plots

2.1 rAmCharts

2.2 plotly

3. Mapping

3.1 ggmap

3.2 Shapefile contours with sf

3.3 Interactive maps with leaflet

4. Networks plots

5. Dashboards and shiny apps

5.1 flexdashboards

5.2 shiny apps

6. Project

Background map

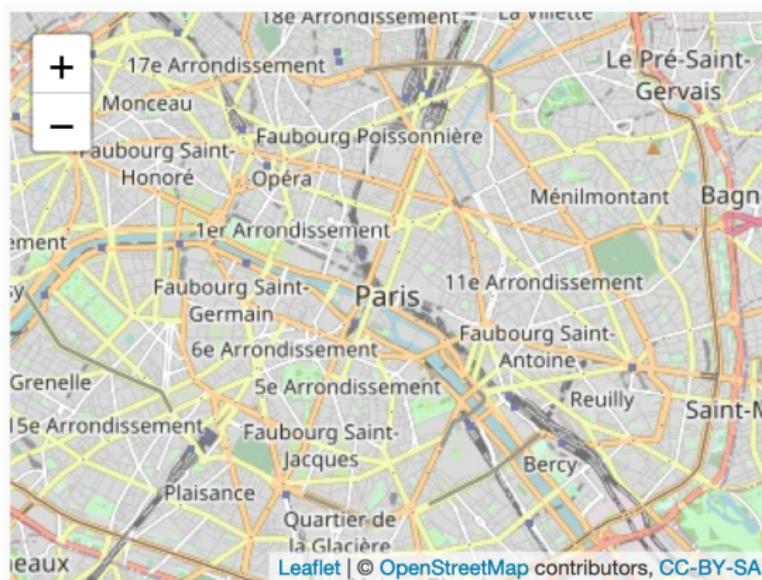
- ▶ Leaflet is one of the most popular open-source JavaScript libraries for **interactive maps**.
- ▶ Documentation : [here](#)

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



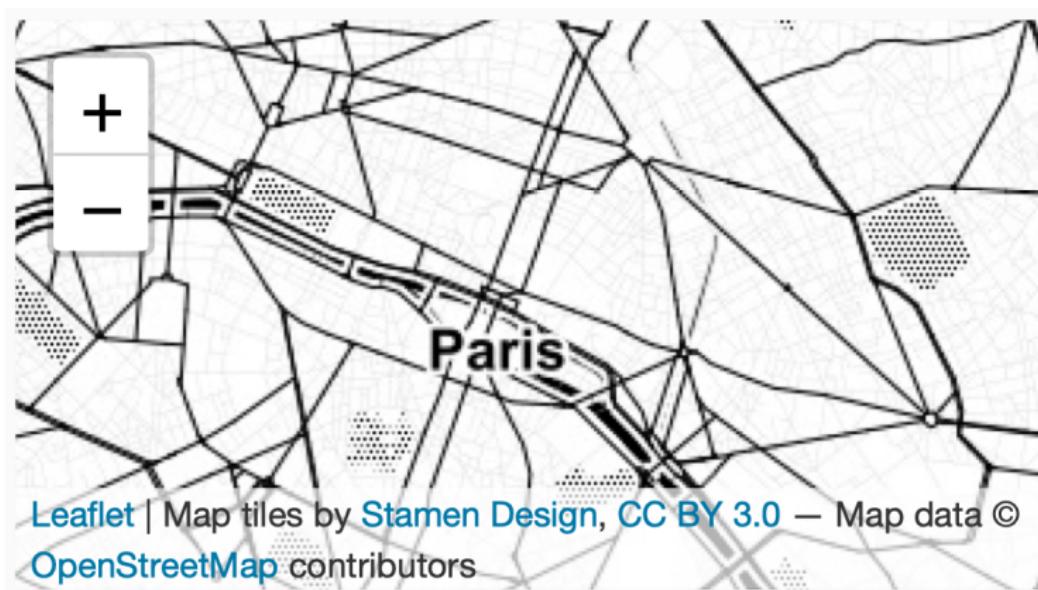
Many background styles

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



Leaflet | © OpenStreetMap contributors, CC-BY-SA

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



Leaflet with data

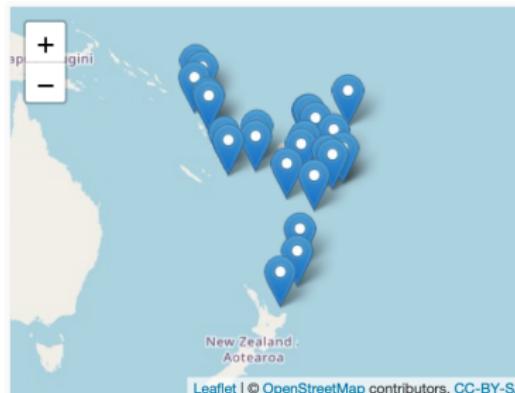
- ▶ Location of 1000 seismic events near Fiji

```
> 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
```

Visualize seismics with magnitude more than 5.5

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

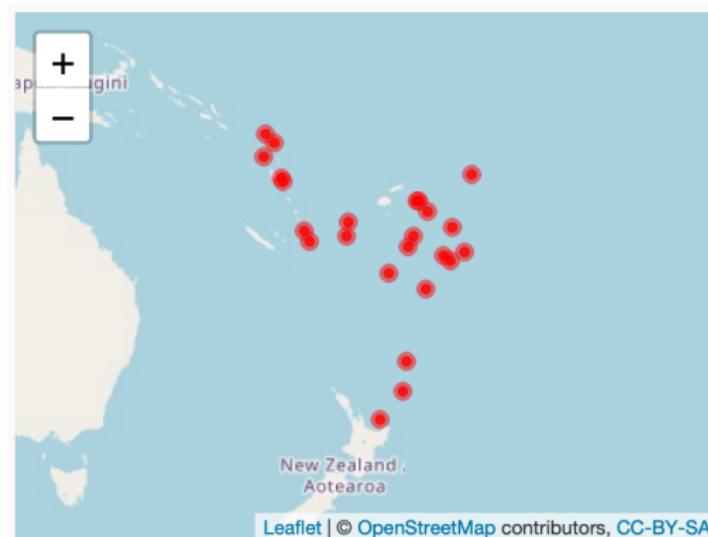


Remark

When you click on a marker, the magnitude appears.

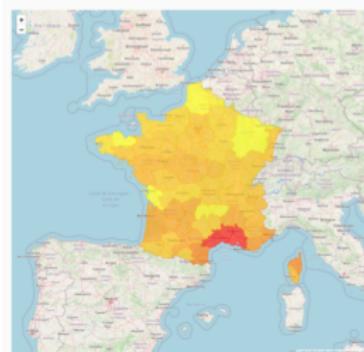
addCircleMarkers

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



Color polygon (combining leaflet and 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 = " :"))
```



→ Work on **this part** of the tutorial.

Plan

1. Nice plots with ggplot
 - 1.1 Conventional R plots
 - 1.2 ggplot2 grammar
2. Dynamic plots
 - 2.1 rAmCharts
 - 2.2 plotly
3. Mapping
 - 3.1 ggmap
 - 3.2 Shapefile contours with sf
 - 3.3 Interactive maps with leaflet
4. Networks plots
5. Dashboards and shiny apps
 - 5.1 flexdashboards
 - 5.2 shiny apps
6. Project

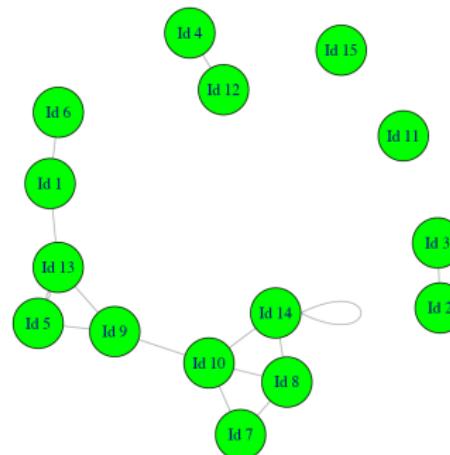
Connections between individuals

- ▶ Many datasets can be visualized with **graphs**, especially when one has to study **connections** between individuals (genomic, social network. . .).
- ▶ One individual = **one node** and one connection = **one edge**.

```
> 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
```

Static graph : igraph package

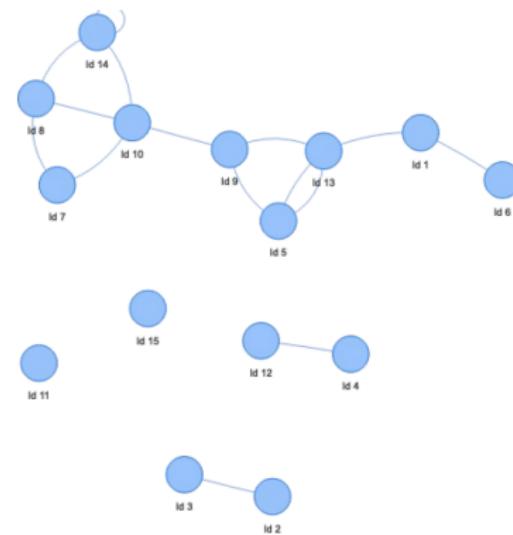
```
library(igraph)
net <- graph_from_data_frame(d=edges, vertices=nodes, directed=F)
plot(net, vertex.color="green", vertex.size=25)
```



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

Dynamic graph : visNetwork Package

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



► References :

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

Plan

1. Nice plots with ggplot
 - 1.1 Conventional R plots
 - 1.2 ggplot2 grammar
2. Dynamic plots
 - 2.1 rAmCharts
 - 2.2 plotly
3. Mapping
 - 3.1 ggmap
 - 3.2 Shapefile contours with sf
 - 3.3 Interactive maps with leaflet
4. Networks plots
5. Dashboards and shiny apps
 - 5.1 flexdashboards
 - 5.2 shiny apps
6. Project

Plan

1. Nice plots with ggplot
 - 1.1 Conventional R plots
 - 1.2 ggplot2 grammar
2. Dynamic plots
 - 2.1 rAmCharts
 - 2.2 plotly
3. Mapping
 - 3.1 ggmap
 - 3.2 Shapefile contours with sf
 - 3.3 Interactive maps with leaflet
4. Networks plots
5. Dashboards and shiny apps
 - 5.1 flexdashboards
 - 5.2 shiny apps
6. Project

- ▶ Just a tool. . . but an **important visualization tool** in datascience
- ▶ Allow to **gather important messages** on datasets and/or models
- ▶ Package : flexdasboard
- ▶ Syntax : simple. . . only **Rmarkdown**
- ▶ Reference : <https://rmarkdown.rstudio.com/flexdashboard/>

Header

```
---
```

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

```
---
```

- ▶ Default theme could be replaced by **other themes** (cosmo, bootstrap, cerulean. . .) (see [here](#)). You just have to add

```
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)
```

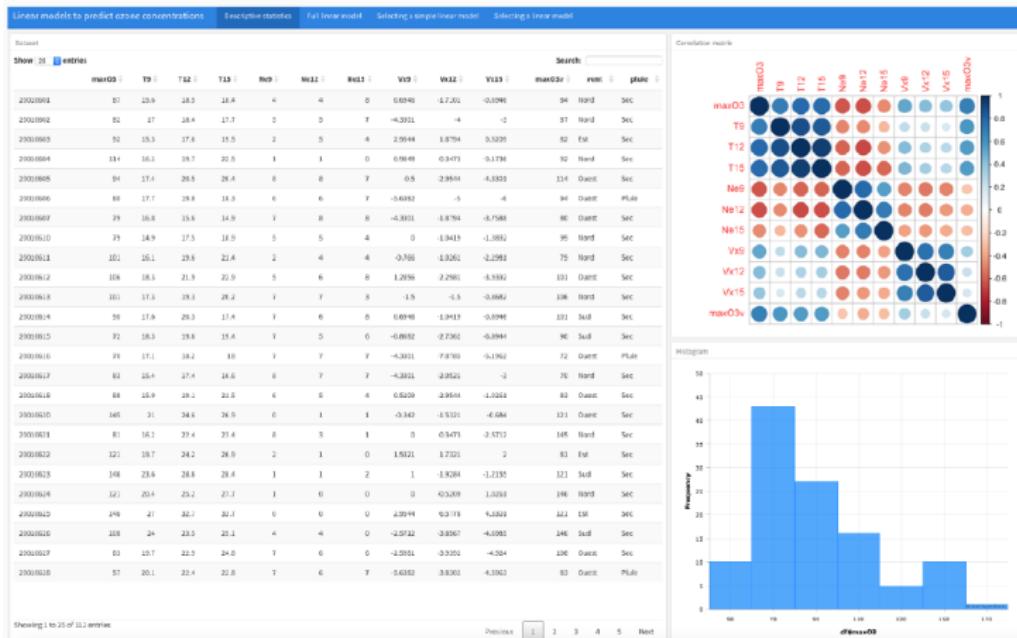
### Histogram
```{r}
amHist(df$max03)
```

```

Visualisation

- └ Dashboards and shiny apps
- └ flexdashboards

Flexdashboard || dashboard



→ Work on **this part** of the tutorial.

Plan

1. Nice plots with ggplot
 - 1.1 Conventional R plots
 - 1.2 ggplot2 grammar
2. Dynamic plots
 - 2.1 rAmCharts
 - 2.2 plotly
3. Mapping
 - 3.1 ggmap
 - 3.2 Shapefile contours with sf
 - 3.3 Interactive maps with leaflet
4. Networks plots
5. Dashboards and shiny apps
 - 5.1 flexdashboards
 - 5.2 shiny apps
6. Project

Shiny

Shiny is a **R package** that makes it easy to build **interactive web applications** with R

- ▶ does not require web expertise
- ▶ combine datascience power of R with **web interactivity**
- ▶ create **local** applications
- ▶ or **deploy** applications for other users : **shiny-server**,
shinyapps.io, **shinyproxy**

References

[https://lrouvriere.github.io/VISU/pres_shiny.pdf](https://lrouviere.github.io/VISU/pres_shiny.pdf)

<http://shiny.rstudio.com>

<http://www.shinyapps.io/>

<https://www.shinyproxy.io/>

<https://www.rstudio.com/products/shiny/shiny-server/>

Shiny app

Structure = 2 scripts

- ▶ ui.R and
- ▶ server.R



Hybrid approach : flexdashboard + shiny

- ▶ it is possible to take advantage of the **interactivity** of shiny within a dashboard
- ▶ call shiny in flexdashboard **headers**

```
1 ▼ ---  
2   title: "Reting dashboard"  
3   output:  
4     flexdashboard::flex_dashboard:  
5       orientation: rows  
6       source_code: embed  
7       theme: spacelab  
8   runtime: shiny  
9 ▲ ---
```

Plan

1. Nice plots with ggplot
 - 1.1 Conventional R plots
 - 1.2 ggplot2 grammar
2. Dynamic plots
 - 2.1 rAmCharts
 - 2.2 plotly
3. Mapping
 - 3.1 ggmap
 - 3.2 Shapefile contours with sf
 - 3.3 Interactive maps with leaflet
4. Networks plots
5. Dashboards and shiny apps
 - 5.1 flexdashboards
 - 5.2 shiny apps
6. Project

Visualization project

- ▶ Group (2 members)
- ▶ Among **the past "data collection and storage" projects** (click [here](#) to see them), choose 1 subject-dataset
- ▶ Build a **dashboard** to visualize important informations on the problem
- ▶ Don't hesitate to use tools presented in the lecture (you can also use other tools)
- ▶ **Deploy** the dashboard on the web with shinyapps, see <https://docs.rstudio.com/shinyapps.io/index.html>
- ▶ Provide a tab in the dashboard presenting the subject, the available visualisations and how the dashboard works

Create and deploy a first dashboard

- ▶ Install the flexdashboard package
- ▶ Register on shinyapp.io (not necessary if you have a gmail account)
- ▶ Open a new Rmdardown file
- ▶ Click on "Knit"
- ▶ click on the "deploy" button and follow the instructions