UNIVERSITÉ CÔTE D'AZUR

Data Visualization with R

Aline Menin

Maîtresse de Conférences, Université Côte d'Azur

aline.menin@univ-cotedazur.fr

2024/2025

Visualizing data with ggplot2

ggplot2 : create graphics by describing their syntax through a grammar of atomic elements:

- The data
- Geometric objects (point, lines, rectangles, bars, etc) (geom_*)
  - Mapping between data and geometric aesthetics (aes())
- Scales to define the extent and transformations of the graph (scales_)
- A coordinate system to define the link between scales (coord_)
- The decoration components: labels (labs()) and legends (guides())
- Possibly, components to decompose graphs (facets)

These elements are combined as additional properties of graphics via the operator +

```
More information at https://ggplot2.tidyverse.org/
```

The data

```
round_2 <- read_csv('data/results_pres_elections_dept_2017_round_2.csv')
round_2 <- round_2 %>% gather(candidate, votes, c(`LE PEN`, MACRON))

plot_df <- round_2 %>% group_by(region_code, region_name, candidate) %>%
  summarise(votes = sum(votes))
```

```
## # A tibble: 5 x 4
## # Groups:   region_code, region_name [3]
##   region_code region_name             candidate   votes
##         <dbl> <chr>                   <chr>       <dbl>
## 1          11 Île-de-France           LE PEN    1033686
## 2          11 Île-de-France           MACRON    3825279
## 3          24 Centre-Val de Loire     LE PEN     450750
## 4          24 Centre-Val de Loire     MACRON     777973
## 5          27 Bourgogne-Franche-Comté LE PEN     532935
```

Plotting the data

Create a variable that holds the plot

```
plot <- ggplot(plot_df)
```

Note: the data must follow a tidy structure.

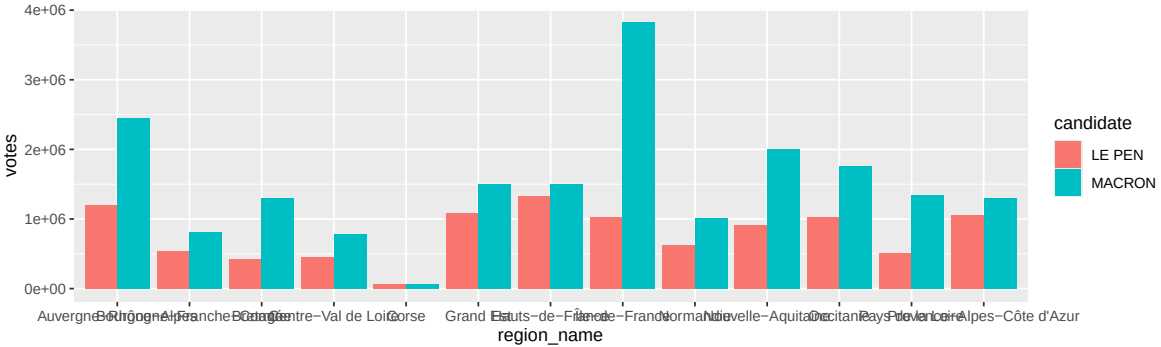
The layers and geometric components

A ggplot2 graphic consists of layers that combine data, aesthetic mapping, a geometric object (geom), a statistical transformation (stat), and a position adjustment.

The aesthetics

The variables are mapped to generic (x, y, color) and specific (linetype, shape, xmin) properties of the geom component.
e.g. the geom_col() requires x and y and can be customized with specific properties (alpha,color, fill, position)

```
plot <- plot + geom_col(aes(x = region_name, y = votes, fill = candidate), position = 'dodge')
```



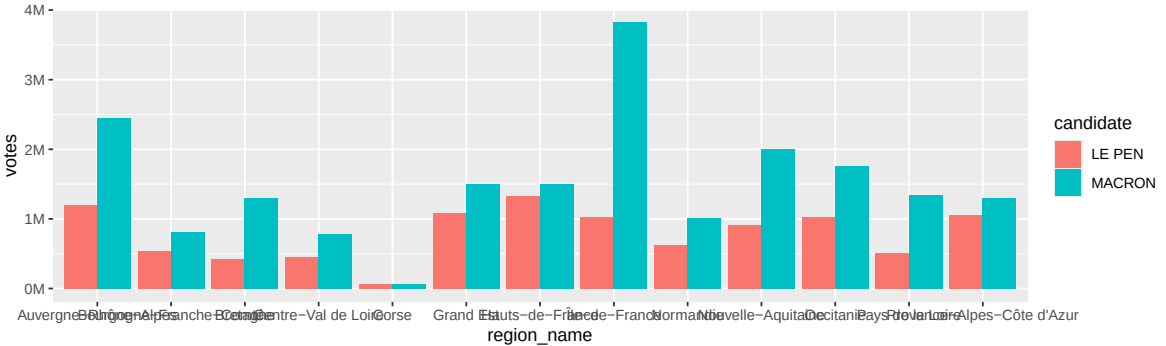
Note: the ggplot2 package provides the function ggplot (without the 2)

The scales

The scales package provides methods to set the breaks and labels of axes and legends.

```
install.packages("scales")
```

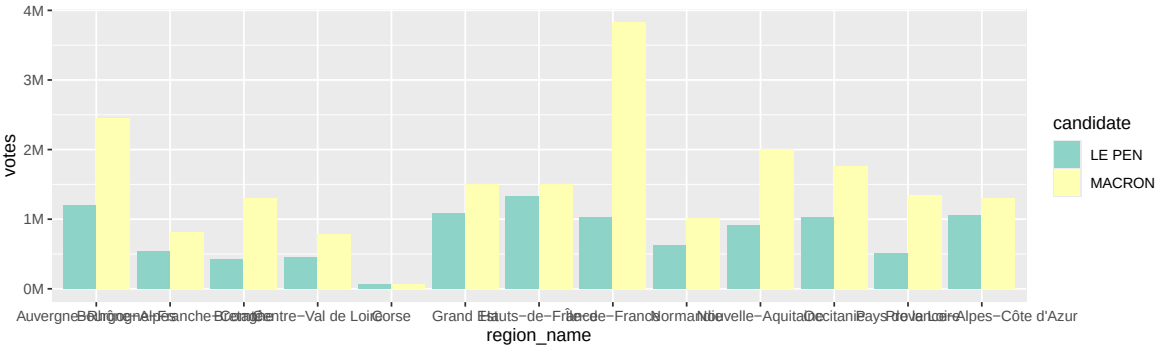
```
library(scales)

plot <- plot + scale_y_continuous(labels = number_format(scale = 1/1000000, suffix = 'M'))
```



The scales: predefined color palettes

Color palettes from https://colorbrewer2.org/ created by Cynthia Brewer

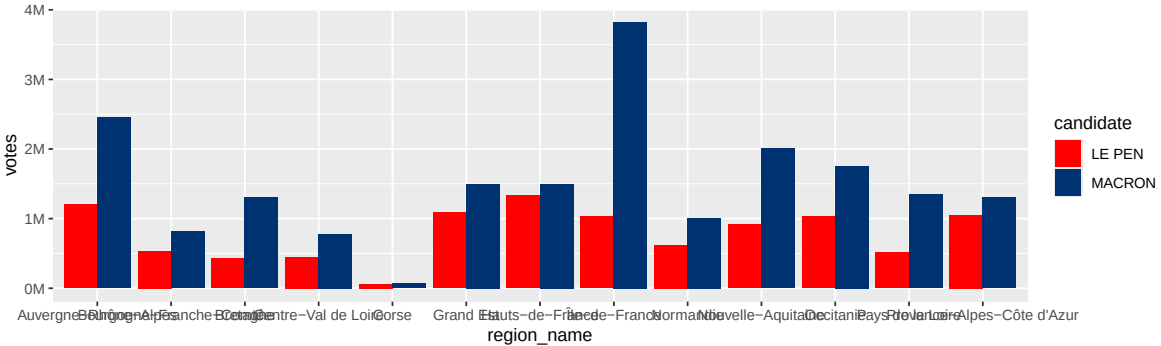
```
plot + scale_fill_brewer(palette = 'Set3')
```




```
More information at https://rdrr.io/cran/RColorBrewer/man/ColorBrewer.html
```

The scales: manual color scale

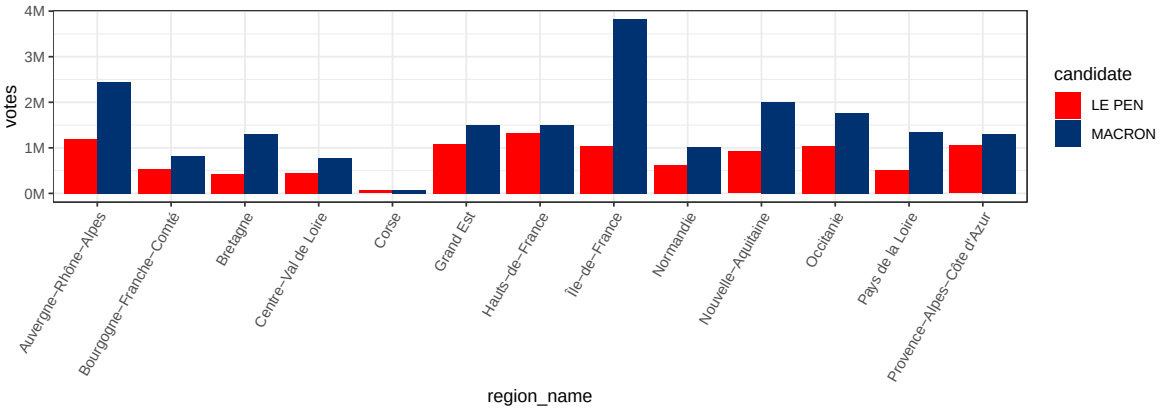
```
plot <- plot +
  scale_fill_manual(values = c('#FF0000', '#003171'))
```



Decoration components: theme

Predefined theme elements: theme_gray(), theme_bw(), theme_dark(), etc. Use the theme() element to customize single components.

```
plot <- plot + theme_bw() + theme(axis.text.x = element_text(angle = 60, hjust = 1))
```

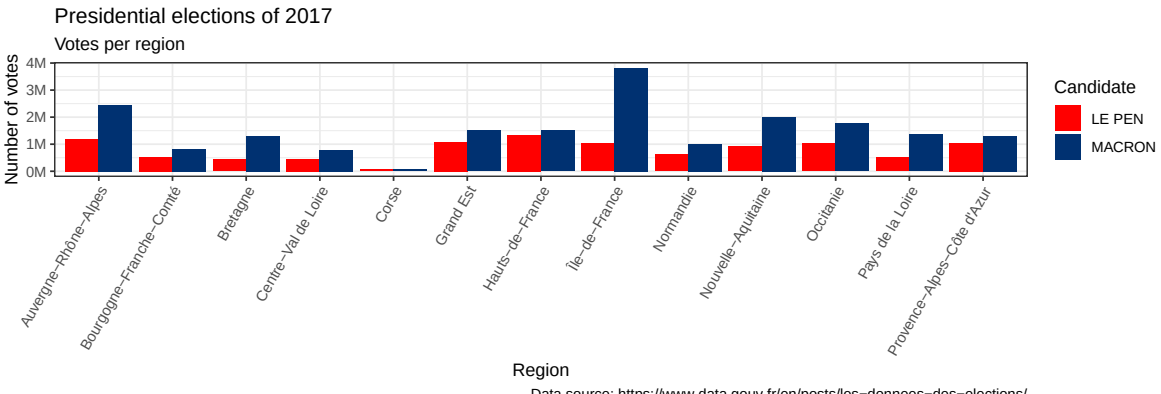



```
List of available themes: https://ggplot2.tidyverse.org/reference/ggtheme.html
```

Decoration components: title and legends

labs(): include information such as title, subtitle, and caption and axes' labels.

```
plot <- plot + labs(title = "Presidential elections of 2017", subtitle = "Votes per region",
          caption = "Data source: https://www.data.gouv.fr/en/posts/les-donnees-des-elections/",
          y = "Number of votes", x = "Region") +
      guides(fill = guide_legend(title = 'Candidate'))
```



Complete code of the bar chart

```r
plot <- ggplot(plot_df) +
  geom_col(aes(x = region_name, y = votes, fill = candidate), position = 'dodge') + # geometric object
  scale_y_continuous(labels = number_format(scale = 1/1000000, suffix = 'M')) +  # y axis format
  scale_fill_manual(values = c('#FF0000', '#003171')) +  # fill colors
  theme_bw() +  # theme
  theme(axis.text.x = element_text(angle = 45, hjust = 1),
        legend.position = 'bottom') +
  labs(title = "Presidential elections of 2017",         # title and labels
       subtitle = "Votes per region and candidate",
       caption = "Data source: https://www.data.gouv.fr/
                  en/posts/les-donnees-des-elections/",
       y = "Number of votes", x = "Region") +
   guides(fill = guide_legend(title = 'Candidate'))             # legend
```

Combining geometric objects → represent multiple variables

Prepare the data that we want to visualize

```
missing_votes <- round_2 %>%
    distinct(region_code, dept_code, .keep_all = TRUE) %>% # keep only one observation per department
    group_by(region_code, region_name) %>%
    summarise(blank_ballot = sum(blank_ballot), null_ballot = sum(null_ballot),
              absent_voters = sum(absent_voters)) %>% gather(category, value, c(3:5))
```

```
## # A tibble: 5 x 4
## # Groups:   region_code [2]
##   region_code region_name         category        value
##         <dbl> <chr>               <chr>           <dbl>
## 1          11 Île-de-France       blank_ballot   406994
## 2          11 Île-de-France       null_ballot    108753
## 3          11 Île-de-France       absent_voters 1820241
## 4          24 Centre-Val de Loire blank_ballot   129941
## 5          24 Centre-Val de Loire null_ballot     43065
```

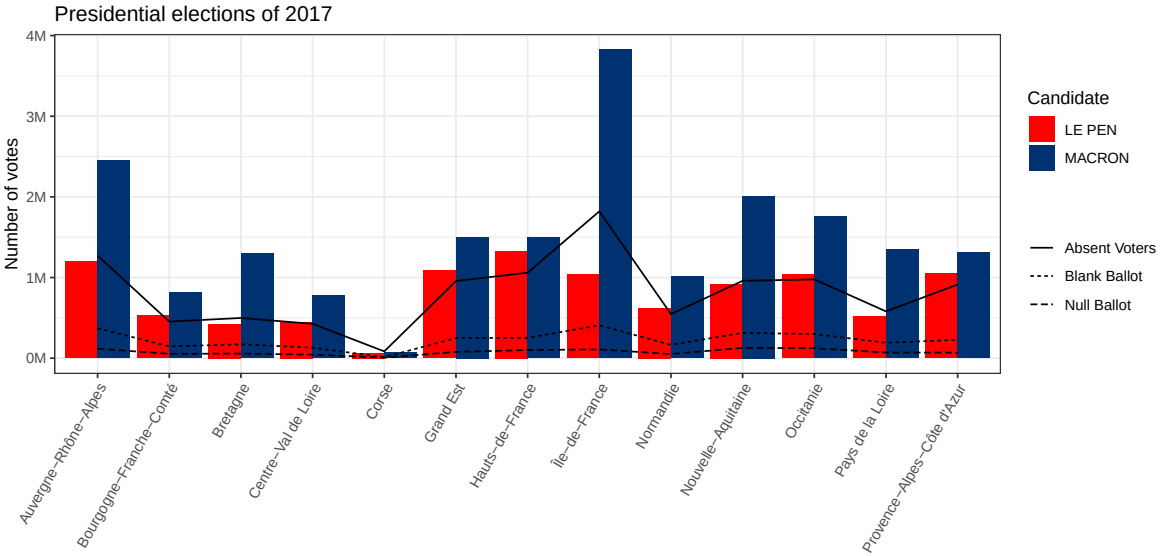
Note that the `distinct()` function modifies the tibble, keeping only the variable(s) given as argument. Use the option `.keep_all = TRUE` to also keep the remaining variables.

Combining geometric objects → represent multiple variables

Combine two or more geometric objects (e.g. geom_col() and geom_line())

```
ggplot(plot_df, aes(x = region_name)) + # common aesthetics
  geom_col(aes(y = votes, fill = candidate), position = 'dodge') +
  # geom_line object for a second variable
  geom_line(data = missing_votes, # new data
            aes(y = value, linetype = category, group = category)) + # aesthetics
  scale_y_continuous(labels = number_format(scale = 1/1000000, suffix = 'M')) +
  scale_fill_manual(values = c('#FF0000', '#003171')) +
  theme_bw() + theme(axis.text.x = element_text(angle = 60, hjust = 1),
                     legend.position = 'right') +
  labs(title = "Presidential elections of 2017", y = "Number of votes", x = "Region") +
  guides(fill = guide_legend(title = 'Candidate'),
         linetype = guide_legend(title = '')) + # title of linetype legend
  scale_linetype_discrete(labels = c("Absent Voters", "Blank Ballot",
                                     "Null Ballot")) # labels for each linetype
```

Combining geometric objects → represent multiple variables



Decomposition components: facets

Facets split up the data by one or more variables and plot the subsets of data together.

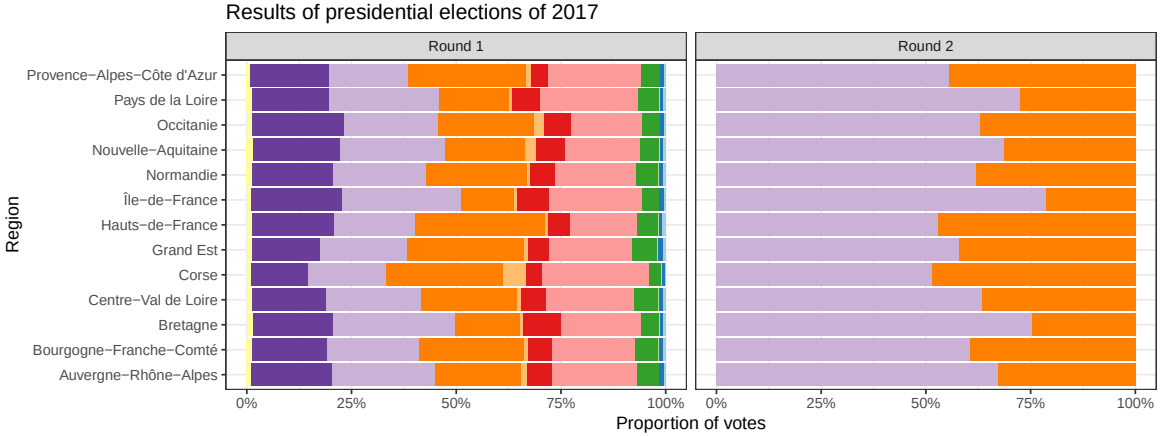
```
round_1 <- read_csv('data/results_pres_elections_dept_2017_round_1.csv')
round_2 <- read_csv('data/results_pres_elections_dept_2017_round_2.csv')

results <- round_1 %>% mutate(round = "Round 1") %>%
    bind_rows(round_2 %>% mutate(round = "Round 2"))

votes_per_round <- results %>%
  gather(candidate, votes, c(ARTHAUD:POUTOU)) %>%
  group_by(region_code, region_name, candidate, round) %>%
  summarise(votes = sum(votes))
```

```
## # A tibble: 286 x 5
## # Groups:   region_code, region_name, candidate [143]
##   region_code region_name   candidate  round   votes
##         <dbl> <chr>         <chr>      <chr>   <dbl>
## 1          11 Île-de-France ARTHAUD    Round 1 23582
## 2          11 Île-de-France ARTHAUD    Round 2    NA
## 3          11 Île-de-France ASSELINEAU Round 1 64391
## 4          11 Île-de-France ASSELINEAU Round 2    NA
## 5          11 Île-de-France CHEMINADE  Round 1  9795
## # i 281 more rows
```

Code: chart with two skacked bar charts side-by-side

```
ggplot(votes_per_round, aes(x = region_name)) +
   geom_col(aes(y = votes, fill = candidate),
            position = 'fill') + # to generate stacked bars
   scale_y_continuous(labels = percent_format()) + # y axis format as percent
   scale_fill_brewer(palette = 'Paired') +
   theme_bw() + theme(legend.position = 'bottom') +
   labs(title = "Results of presidential elections of 2017",
                  y = "Proportion of votes", x = "Region") +
   guides(fill = guide_legend(title = 'Candidate'),
          linetype = guide_legend(title = '')) +
   scale_linetype_discrete(labels = c("Absent Voters", "Blank Ballot", "Null Ballot")) +
   # define cols as the number of different values for the variable "round"
   facet_grid(cols = vars(round)) +
   coord_flip() # flip coordinate system
```

Chart: comparing results of Round 1 & Round 2



Geospatial Data

What is geospatial data?

Geospatial data describe objects, events, or phenomena that have a location on the surface of the earth (Stock and Guesgen, 2016). It consists of:

- Location information (usually coordinates on the earth) static in the short-term (e.g., the locations of a road, an earthquake event, etc) dynamic (e.g., a moving vehicle or pedestrian, the spread of an infectious disease)
- Attribute information (the characteristics of the object, event or phenomenon concerned)
- Temporal information (the time or life span at which the location and attributes exist)

Geospatial data format

The data follows a standard encoding of geographical information into a computer file: the GIS (Geographical Information System) file format.

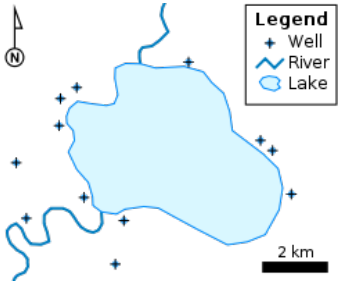
One of the most popular GIS vector file format is the Shapefile. It is a nontopological format for storing the geometric location and attribute information of geographical features. It stores information via a collection of at least three files:

- .shp — shape format: the feature geometry itself
- .shx — shape index format: a positional index of the feature geometry to allow seeking forwards and backwards quickly
- .dbf — attribute format columnar attributes for each shape

Geographical features

Geographical features are often expressed as vectors, by considering those features as geometric shapes. Their geometry varies according to the feature type:

- Points are used to represent geographical features that can be expressed by a single point of reference, e.g. wells, peaks, etc.
- Lines are used to represent linear features such as rivers, roads, etc.
- Polygons are used to represent geographical features that cover a particular area of the earth's surface, e.g. lakes, buildings, etc.



Geospatial Data in R

The package sf

R uses the Simple Features standard, which specifies a common storage and access model of geographic features.

The package sf provides simple features access for R. It represents simple features as records in a tibble with a geometry list-column.

```
install.packages('sf')
```

```
library(sf)
```

```
Installation tutorial:  https://r-spatial.github.io/sf/
```

Loading geospatial data with st_read()

```
regions_sf <- st_read('data/shapefile/contours-geographiques-des-regions-2019.shp')
```

```
## Simple feature collection with 18 features and 5 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -61.80984 ymin: -21.38963 xmax: 55.83663 ymax: 51.08899
## Geodetic CRS:  WGS 84
## First 5 features:
##                         id                region0 insee_reg
## 1 REG00000000000000000001                MAYOTTE        06
## 2 REG00000000000000000001             LA REUNION        04
## 3 REG00000000000000000001             MARTINIQUE        02
## 4 REG00000000000000000010         HAUTS-DE-FRANCE        32
## 5 REG00000000000000000008 BOURGOGNE-FRANCHE-COMTE        27
##                    region     siren                       geometry
## 1                 Mayotte      <NA> MULTIPOLYGON (((45.13822 -1...
## 2              La Réunion      <NA> MULTIPOLYGON (((55.22411 -2...
## 3              Martinique 200055507 MULTIPOLYGON (((-60.86705 1...
## 4         Hauts-de-France 200053742 MULTIPOLYGON (((2.809197 49...
## 5 Bourgogne-Franche-Comté 200053726 MULTIPOLYGON (((5.405612 47...
```

Geospatial data manipulation

```r
regions_sf <- regions_sf %>% mutate(insee_reg = as.numeric(insee_reg)) %>%
  filter(!(insee_reg %in% c(1:6)))

data_sf <- regions_sf %>% left_join(plot_df, by = c('insee_reg'='region_code'))

as_tibble(data_sf) # print sf objects in a nice format
```

```
## # A tibble: 26 x 9
##    id                region0 insee_reg region siren region_name candidate  votes
##    <chr>             <chr>       <dbl> <chr>  <chr> <chr>       <chr>      <dbl>
##  1 REG00000000000000~ HAUTS~~        32 Hauts~ 2000~ Hauts-de-F~ LE PEN    1.33e6
##  2 REG00000000000000~ HAUTS~~        32 Hauts~ 2000~ Hauts-de-F~ MACRON    1.50e6
##  3 REG00000000000000~ BOURGO~        27 Bourg~ 2000~ Bourgogne-~ LE PEN    5.33e5
##  4 REG00000000000000~ BOURGO~        27 Bourg~ 2000~ Bourgogne-~ MACRON    8.16e5
##  5 REG00000000000000~ OCCITA~        76 Occit~ 2000~ Occitanie   LE PEN    1.03e6
##  6 REG00000000000000~ OCCITA~        76 Occit~ 2000~ Occitanie   MACRON    1.76e6
##  7 REG00000000000000~ ILE-DE~        11 Île-d~ <NA>  Île-de-Fra~ LE PEN    1.03e6
##  8 REG00000000000000~ ILE-DE~        11 Île-d~ <NA>  Île-de-Fra~ MACRON    3.83e6
##  9 REG00000000000000~ AUVERG~        84 Auver~ 2000~ Auvergne-R~ LE PEN    1.20e6
## 10 REG00000000000000~ AUVERG~        84 Auver~ 2000~ Auvergne-R~ MACRON    2.45e6
## # i 16 more rows
## # i 1 more variable: geometry <MULTIPOLYGON [°]>
```

Static thematic maps with ggplot2

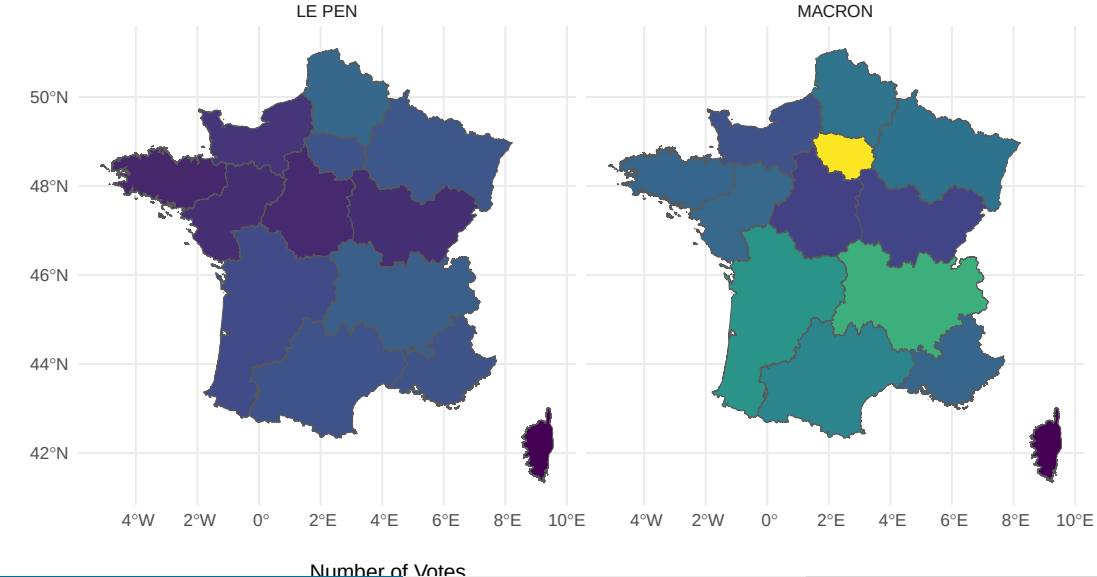
ggplot2 provides a set of geom (geom_sf()), stat (stat_sf()), and coord (coord_sf()) components to visualize sf objects.

The geom_sf() will draw different geometric objects depending on what simple features are present in the data: you can get points, lines, or polygons.

```
ggplot(data_sf) +
  geom_sf(aes(fill = votes)) +
  facet_grid(cols = vars(candidate)) +
  scale_fill_viridis_c(name = 'Number of Votes',
                       labels = number_format(scale = 1/1000000, suffix = 'M')) +
  guides(fill = guide_colourbar(title.position = 'top')) +
  theme_minimal() +
  theme(legend.position = "bottom", legend.key.width = unit(2, 'cm'))
```

The viridis package : color palettes perceptually uniform in both color and black-and-white

Static thematic maps with ggplot2



Dynamic thematic maps with `leaflet`

Leaflet: open-source JavaScript library for interactive maps

```
install.packages('leaflet')
```

```
library(leaflet)
```

Thematic map: data preparation

```
plot_df <- round_2 %>% distinct(region_code, dept_code, .keep_all = TRUE) %>%
    group_by(region_code, region_name) %>%
    summarise(present_voters = sum(present_voters), registered_voters = sum(registered_voters),
              voting_rate = present_voters/registered_voters, .groups = "drop")

plot_sf <- regions_sf %>%  left_join(plot_df, by = c('insee_reg'='region_code'))
```

```
## Simple feature collection with 5 features and 11 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -0.3271761 ymin: 42.33292 xmax: 7.185565 ymax: 51.08899
## Geodetic CRS:  WGS 84
##                          id                region0 insee_reg
## 1 REG00000000000000000010         HAUTS-DE-FRANCE        32
## 2 REG00000000000000000008 BOURGOGNE-FRANCHE-COMTE        27
## 3 REG00000000000000000001               OCCITANIE        76
## 4 REG00000000000000000013           ILE-DE-FRANCE        11
## 5 REG00000000000000000007    AUVERGNE-RHONE-ALPES        84
##                     region     siren             region_name present_voters
## 1         Hauts-de-France 200053742         Hauts-de-France        3181359
## 2 Bourgogne-Franche-Comté 200053726 Bourgogne-Franche-Comté        1548103
## 3               Occitanie 200053791               Occitanie        3216862
## 4          Île-de-France       <NA>          Île-de-France        5374712
```

Define the mapping functions

Define a function to determine the quantiles of the voting_rate values

```
quants <- quantile(x = plot_sf$voting_rate,
                   probs = seq(from = 0, to = 1, by = 0.2))
```

```
##        0%       20%       40%       60%       80%      100%
## 0.6398154 0.7482259 0.7625725 0.7680192 0.7765257 0.7968782
```

Define a function to color the regions according to each quantile

```
color_scale <- colorBin(palette = "YlOrRd",
                        domain = plot_sf$voting_rate, bins = quants)
```

Chunk code: creating a map with `leaflet`

Similar to ggplot2, combine `leaflet` methods to draw the different elements of the map

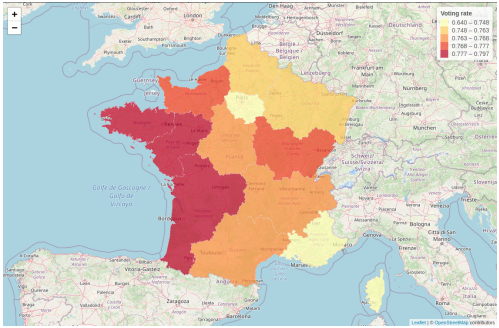
```
map_leaflet <- leaflet(data = plot_sf) %>%
  # draw map tiles (from Open Street Map)
  addProviderTiles(providers$OpenStreetMap) %>%
  # draw regions from .shp file
  addPolygons(fillColor = ~color_scale(voting_rate), fillOpacity = 0.7,
    color = "white", weight = .5, opacity = 1, dashArray = "3") %>%
  addLegend(pal = color_scale, values = ~voting_rate, opacity = 0.7,
            title = "Voting rate", position = "topright") %>%
  # add a tooltip with the actual voting_rate value per region
  addMarkers(~longitude, ~latitude, label = mapply(
                      function(x) { sprintf("~%s %%", trunc(x * 100)) },
                      plot_sf$voting_rate, SIMPLIFY = F))
```

The htmlwidgets package

Save the graphic as an HTML page

```
install.packages('htmlwidgets')
```

```
library(htmlwidgets)
saveWidget(map_leaflet, "leaflet_map.html")
```



Interactive Visualization Dashboards

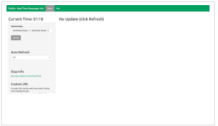
Shiny from RStudio (R Shiny)

Shiny is a R package for building interactive web apps directly from R.

Voronoys - Understanding voters' profile in Brazilian elections

Crime Watch

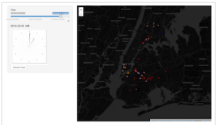
Pasture Potential Tool for improving dairy farm profitability and environmental impact



Dublin Transport Info

Locating Blood Banks in India

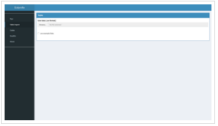
Utah Lake Water Quality Profile Dashboard



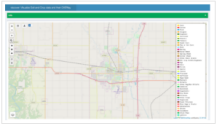
Animated NYC metro traffic

New Zealand Trade Intelligence Dashboard

ScotPHO Profiles Tool



Visualisation of soil profiles



VISCOVER

City Cycle Race (with STRAVA) - Compare the cycling speed of cities

Why should we use R Shiny?

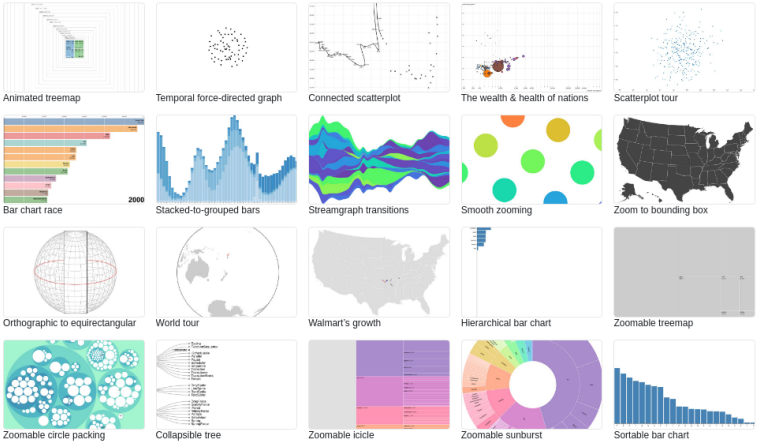
- If you know R, R Shiny is easy to pick up.
- R Shiny provides built-in basic visualizations, mostly using the ggplot2 library.
  - The ggplot2 objects can be integrated with the Plotly or bokeh libraries to allow interactivity
  - JavaScript can be also integrated (i.e. the d3.js library) for producing more powerful visualizations
- RStudio provides a number of options for hosting Shiny apps (only one is free/open-source)
- Beyond hosting costs, R Shiny is completely free.
- Since Shiny is totally integrated with R, the dashboard can easily re-run analyses based on user input
  - Possibly the main advantage of using R Shiny

However

- if you are not a relatively experienced R user, learning how to use R Shiny can be difficult.

JavaScript-based Visualizations (d3.js)

D3 (Data Driven Documents) is a JavaScript library developed by Mike Bostock and huge community of volunteer developers for manipulating documents based on data.



Why should we use d3.js?

- To go beyond static graphics and create powerful and fully interactive visualizations
- d3 relies completely on you, the developer, to specify what you want to see You're likely to reach the limitations of your imagination and sanity long before you stump JavaScript.
- Steep learning curve. But once we understand how d3 creates, selects, and manipulates SVG objects, the more advanced d3 actions are simply extensions of those principles
- If you can host a website, you can host d3
- d3 is an open-source library and, therefore, free for everyone

However

- Even the most basic dashboard can take hundreds of lines of code

  - Tons of reusable code for basic charts

- JavaScript is not very easy for beginners

Cheatsheets