# Data Manipulation and Visualization with R

Aline Menin

I3S, Inria

October 8, 2021

# Overview of R and RStudio

# What is R?

R is a free software environment for statistical computing and graphics

We can download it at https://www.r-project.org/

R for Beginners: https://cran.r-project.org/doc/contrib/Paradis-rdebuts_en.pdf

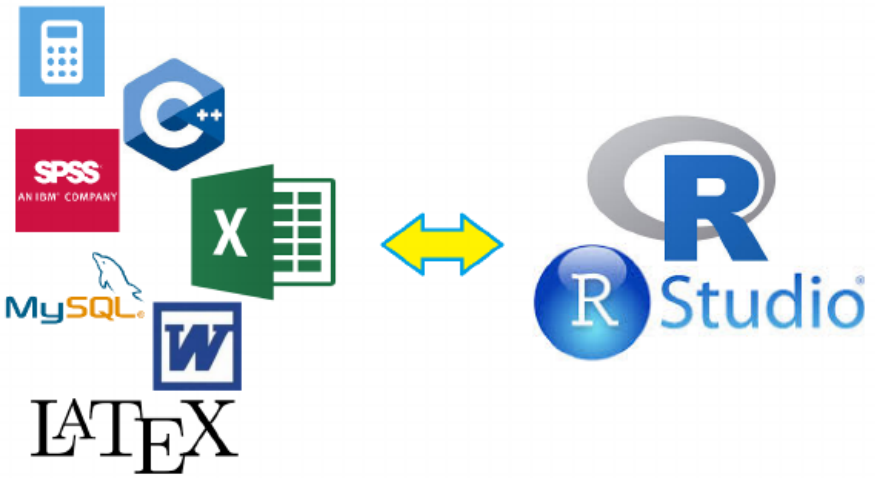
It allows us to do

- arithmetic calculations;
- data manipulation;
- a large variety of graphics;
- scripts to automatize data treating;
- numeric modeling and simulation;
- a large variety of statistical treatments;
- reports, web pages, interactive applications, and slideshows.

# Why should we use R?

It can replace the functions of

- a calculator;
- a spreadsheet;
- a programming language;
- a statistical software;
- a reports and presentation editor.



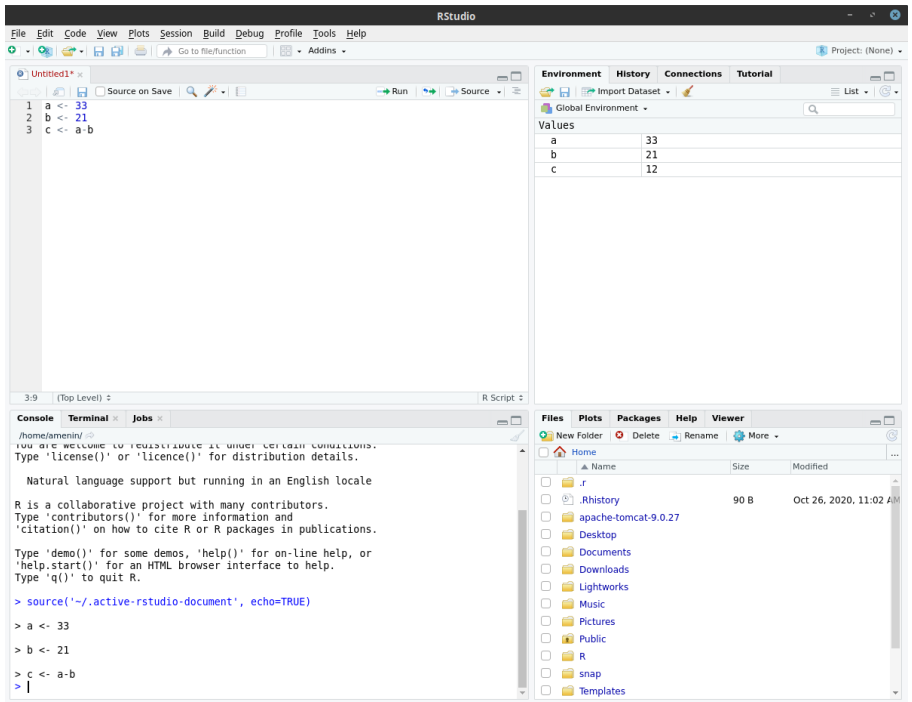

Although R is quite cumbersome at first, it quickly reveals itself as an incredible asset compared to spreadsheets.

# How to use R?

To simplify the usage of R, we normally use it via an adapted editor: RStudio (Download at https://rstudio.com/products/rstudio/download/)

RStudio contains four windows:

- the Source (top left)
- the Console (bottom left)
- the Environment, History, ... (top right)
- the Files, Plots, Packages, Help, ... (bottom right)



# Packages

R packages are collections of functions and datasets developed by the community. They increase the power of R by improving existing base R functionalities or by adding new ones. To install a package in R, we run the following command:

```
install.packages("package_name")
```

Then, we must include the package into our script by using:

```
library(package_name)
```

# Packages

We start by using the tidyverse package, which is a collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures. It contains

- the readr package for reading/writing data tables, which is based on
- the tibble package for establishing the data format;
- the dplyr package for computing aggregate statistics in datasets;
- the tidyr package for reshaping datasets according to our needs;
- the stringr package for manipulating strings; and
- the ggplot2 package for producing nice looking and configurable graphics.

```
install.packages("tidyverse")
library(tidyverse)
```

# Data Manipulation

# Data import: `readr`

The package provides different methods according to the data format:

- read_csv(): comma separated (CSV) files; uses comma (,) as field separator and dot (.) as decimal point (English format).
- read_csv2(): comma separated (CSV) files; uses semi-colon (;) as field separator and comma (,) as decimal point (French format).
- read_delim(): generic and customizable method for reading files with specific field separators (e.g., tabs).
- write_csv(): save a dataframe to a csv file.

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

# The data

The results of the presidential election of 2017 in France, round 2.

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

```
##
## -- Column specification --------------------------------------------------
## cols(
##   region_code = col_double(),
##   region_name = col_character(),
##   dept_code = col_character(),
##   dept_name = col_character(),
##   registered_voters = col_double(),
##   absent_voters = col_double(),
##   present_voters = col_double(),
##   blank_ballot = col_double(),
##   null_ballot = col_double(),
##   votes_cast = col_double(),
##   `LE PEN` = col_double(),
##   MACRON = col_double()
## )
```

Note that names of variables containing white spaces are put in between backticks, which must be used to access it later.

# Data format: data.frame

A data frame is a table or a two-dimensional array-like structure in which each column contains values of one variable and each row contains one set of values from each column.

- The column names should be non-empty.
- The row names should be unique.
- The data stored in a data frame can be of numeric, factor or character type.
- Each column should contain same number of data items.

| | region_code | region_name | dept_code | dept_name |
|---|---|---|---|---|
| 1 | 84 | Auvergne-Rhône-Alpes | 01 | Ain |
| 2 | 32 | Hauts-de-France | 02 | Aisne |
| 3 | 84 | Auvergne-Rhône-Alpes | 03 | Allier |
| 4 | 93 | Provence-Alpes-Côte d'Azur | 04 | Alpes-de-Haute-Provence |
| 5 | 93 | Provence-Alpes-Côte d'Azur | 05 | Hautes-Alpes |
| 6 | 93 | Provence-Alpes-Côte d'Azur | 06 | Alpes-Maritimes |
| 7 | 84 | Auvergne-Rhône-Alpes | 07 | Ardèche |
| 8 | 44 | Grand Est | 08 | Ardennes |
| 9 | 76 | Occitanie | 09 | Ariège |
| 10 | 44 | Grand Est | 10 | Aube |
| 11 | 76 | Occitanie | 11 | Aude |
| 12 | 76 | Occitanie | 12 | Aveyron |
| 13 | 93 | Provence-Alpes-Côte d'Azur | 13 | Bouches-du-Rhône |
| 14 | 28 | Normandie | 14 | Calvados |
| 15 | 84 | Auvergne-Rhône-Alpes | 15 | Cantal |
| 16 | 75 | Nouvelle-Aquitaine | 16 | Charente |
| 17 | 75 | Nouvelle-Aquitaine | 17 | Charente-Maritime |
| 18 | 24 | Centre-Val de Loire | 18 | Cher |
| 19 | 75 | Nouvelle-Aquitaine | 19 | Corrèze |
| 20 | 27 | Bourgogne-Franche-Comté | 21 | Côte-d'Or |
| 21 | 53 | Bretagne | 22 | Côtes-d'Armor |
| 22 | 75 | Nouvelle-Aquitaine | 23 | Creuse |

# Data format: tibble

```
class(round_2)
```

```
## [1] "spec_tbl_df" "tbl_df"      "tbl"         "data.frame"
```

The readr package stores data in an object of type tibble (tbl) that is a modern definition of the data.frame.

```
## # A tibble: 96 x 12
##    region_code region_name dept_code dept_name registered_vote~ absent_voters
##          <dbl> <chr>       <chr>     <chr>                <dbl>         <dbl>
##  1          84 Auvergne-R~ 01        Ain                 415950         93130
##  2          32 Hauts-de-F~ 02        Aisne               375791         90745
##  3          84 Auvergne-R~ 03        Allier              253479         59294
##  4          93 Provence-A~ 04        Alpes-de~           126459         29255
##  5          93 Provence-A~ 05        Hautes-A~           109892         24895
##  6          93 Provence-A~ 06        Alpes-Ma~           761780        198631
##  7          84 Auvergne-R~ 07        Ardèche             249216         54487
##  8          44 Grand Est   08        Ardennes            194349         49886
##  9          76 Occitanie   09        Ariège              117406         26698
## 10          44 Grand Est   10        Aube                203800         45670
## # ... with 86 more rows, and 6 more variables: present_voters <dbl>,
## #   blank_ballot <dbl>, null_ballot <dbl>, votes_cast <dbl>, `LE PEN` <dbl>,
## #   MACRON <dbl>
```

```
round_2$region_name # use the $ symbol to access variables
```

```
##  [1] "Auvergne-Rhône-Alpes"       "Hauts-de-France"
##  [3] "Auvergne-Rhône-Alpes"       "Provence-Alpes-Côte d'Azur"
##  [5] "Provence-Alpes-Côte d'Azur" "Provence-Alpes-Côte d'Azur"
##  [7] "Auvergne-Rhône-Alpes"       "Grand Est"
##  [9] "Occitanie"                  "Grand Est"
## [11] "Occitanie"                  "Occitanie"
```

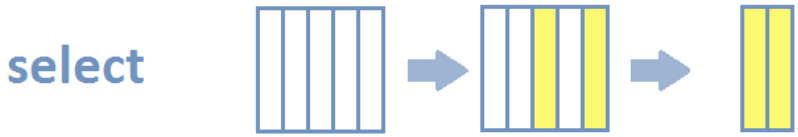
# Manipulating data with dplyr

The dplyr package provides useful functions to manipulate the content of a tibble:

- select(): picks variables (columns) from a tibble
- filter(): picks rows from a tibble based on their values
- arrange(): changes the ordering of the rows according to one or several variables
- mutate(): adds new variables that are functions of existing variables
- summarise(): reduces multiple values down to a single summary (e.g., sum, mean)
- {inner, left, right, full, semi, anti, nest}_join(): join two tibbles by mutating, filtering or nesting them
- bind_{rows, cols}: combine tibbles by binding their rows or columns

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

# dplyr::select()

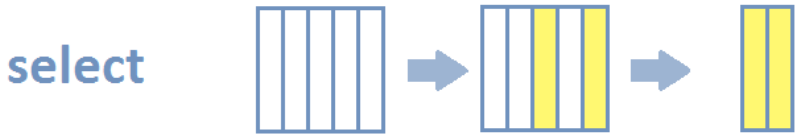



We can select variables based on their names

```
round_2 %>% select(region_name, `LE PEN`, MACRON)
```

```
## # A tibble: 96 x 3
##    region_name                `LE PEN` MACRON
##    <chr>                         <dbl>  <dbl>
##  1 Auvergne-Rhône-Alpes         111421 173809
##  2 Hauts-de-France              133939 119202
##  3 Auvergne-Rhône-Alpes          60207 106579
##  4 Provence-Alpes-Côte d'Azur    34817  48994
##  5 Provence-Alpes-Côte d'Azur    26417  47211
##  6 Provence-Alpes-Côte d'Azur   224544 278407
##  7 Auvergne-Rhône-Alpes          63109 104599
##  8 Grand Est                     62571  64424
##  9 Occitanie                     28074  47983
## 10 Grand Est                     64180  75810
## # ... with 86 more rows
```

# dplyr::select()



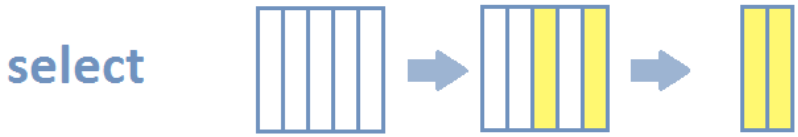

We can select variables based on their names, positions

```
round_2 %>% select(1:5)
```

```
## # A tibble: 96 x 5
##    region_code region_name          dept_code dept_name         registered_vote~
##          <dbl> <chr>                <chr>     <chr>                        <dbl>
##  1          84 Auvergne-Rhône-Alpes 01        Ain                         415950
##  2          32 Hauts-de-France      02        Aisne                       375791
##  3          84 Auvergne-Rhône-Alpes 03        Allier                      253479
##  4          93 Provence-Alpes-Côte~ 04        Alpes-de-Haute-P~           126459
##  5          93 Provence-Alpes-Côte~ 05        Hautes-Alpes                109892
##  6          93 Provence-Alpes-Côte~ 06        Alpes-Maritimes             761780
##  7          84 Auvergne-Rhône-Alpes 07        Ardèche                     249216
##  8          44 Grand Est            08        Ardennes                    194349
##  9          76 Occitanie            09        Ariège                      117406
## 10          44 Grand Est            10        Aube                        203800
## # ... with 86 more rows
```

Note that in R, the first index is 1, not 0 as for the remaining programming languages

# dplyr::select()




We can select variables based on their names, positions, by excluding variables

```
round_2 %>% select(-c(3:7), -region_code)
```

```
## # A tibble: 96 x 6
##    region_name               blank_ballot null_ballot votes_cast `LE PEN` MACRON
##    <chr>                            <dbl>       <dbl>      <dbl>    <dbl>  <dbl>
##  1 Auvergne-Rhône-Alpes             28852        8738     285230   111421 173809
##  2 Hauts-de-France                  22838        9067     253141   133939 119202
##  3 Auvergne-Rhône-Alpes             18877        8522     166786    60207 106579
##  4 Provence-Alpes-Côte d'Az~         9671        3722      83811    34817  48994
##  5 Provence-Alpes-Côte d'Az~         8670        2699      73628    26417  47211
##  6 Provence-Alpes-Côte d'Az~        47784       12414     502951   224544 278407
##  7 Auvergne-Rhône-Alpes             19800        7221     167708    63109 104599
##  8 Grand Est                        12787        4681     126995    62571  64424
##  9 Occitanie                        10133        4518      76057    28074  47983
## 10 Grand Est                        13662        4478     139990    64180  75810
## # ... with 86 more rows
```

# dplyr::select()

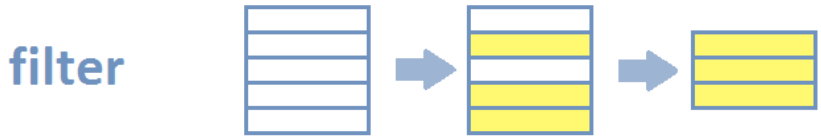
Further, these helpers select variables by matching patterns in their names:

- starts_with(): starts with a prefix
- ends_with(): ends with a suffix
- contains(): contains a literal string

```
round_2 %>% select(contains("vote"))
```

```
## # A tibble: 96 x 4
##    registered_voters absent_voters present_voters votes_cast
##                <dbl>         <dbl>          <dbl>      <dbl>
##  1            415950         93130         322820     285230
##  2            375791         90745         285046     253141
##  3            253479         59294         194185     166786
##  4            126459         29255          97204      83811
##  5            109892         24895          84997      73628
##  6            761780        198631         563149     502951
##  7            249216         54487         194729     167708
##  8            194349         49886         144463     126995
##  9            117406         26698          90708      76057
## 10            203800         45670         158130     139990
## # ... with 86 more rows
```

# dplyr::filter()




The filter() function is used to subset a data frame, retaining all rows that satisfy the given conditions.

Useful functions and operators to build filtering expressions:

- Arithmetic operators: ==, >, >=, etc.
- Logical operators: &, |, !, xor()
- Missing Values: is.na()
- Range: between()
- Comparison (with tolerance): near()

```
Note that in R, missing values are identified by the logical constant NA (Not Available).
```

# dplyr::filter()

## Filtering by one criterion

```
round_2 %>% filter(region_name == "Provence-Alpes-Côte d'Azur")
```

```
## # A tibble: 6 x 12
##   region_code region_name dept_code dept_name registered_vote~ absent_voters
##         <dbl> <chr>       <chr>     <chr>                <dbl>         <dbl>
## 1          93 Provence-A~ 04        Alpes-de~           126459         29255
## 2          93 Provence-A~ 05        Hautes-A~           109892         24895
## 3          93 Provence-A~ 06        Alpes-Ma~           761780        198631
## 4          93 Provence-A~ 13        Bouches-~          1370057        364020
## 5          93 Provence-A~ 83        Var                 794665        202023
## 6          93 Provence-A~ 84        Vaucluse            402307         94108
## # ... with 6 more variables: present_voters <dbl>, blank_ballot <dbl>,
## #   null_ballot <dbl>, votes_cast <dbl>, `LE PEN` <dbl>, MACRON <dbl>
```

Note that R is case-sensitive: it differentiate between capital and lower-case letters.

# dplyr::filter()

## Filtering by multiple criteria within a single logical expression

```
round_2 %>% filter(registered_voters > 100000 & present_voters > 100000)
```

```
## # A tibble: 87 x 12
##    region_code region_name dept_code dept_name registered_vote~ absent_voters
##          <dbl> <chr>       <chr>     <chr>                <dbl>         <dbl>
##  1          84 Auvergne-R~ 01        Ain                 415950         93130
##  2          32 Hauts-de-F~ 02        Aisne               375791         90745
##  3          84 Auvergne-R~ 03        Allier              253479         59294
##  4          93 Provence-A~ 06        Alpes-Ma~           761780        198631
##  5          84 Auvergne-R~ 07        Ardèche             249216         54487
##  6          44 Grand Est   08        Ardennes            194349         49886
##  7          44 Grand Est   10        Aube                203800         45670
##  8          76 Occitanie   11        Aude                272643         61830
##  9          76 Occitanie   12        Aveyron             218097         43851
## 10          93 Provence-A~ 13        Bouches-~          1370057        364020
## # ... with 77 more rows, and 6 more variables: present_voters <dbl>,
## #   blank_ballot <dbl>, null_ballot <dbl>, votes_cast <dbl>, `LE PEN` <dbl>,
## #   MACRON <dbl>
```

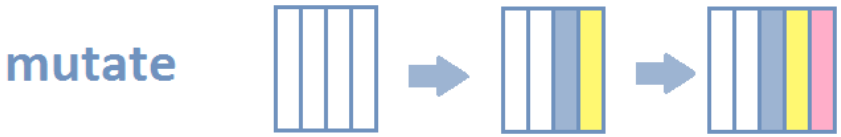
# dplyr::arrange()


```
round_2 %>%
  arrange(registered_voters)
```

```
## # A tibble: 96 x 12
##    region_code region_name dept_code dept_name registered_vote~ absent_voters
##          <dbl> <chr>       <chr>     <chr>                <dbl>         <dbl>
##  1          76 Occitanie   48        Lozère               59496         11884
##  2          75 Nouvelle-A~ 23        Creuse               93122         21738
##  3          27 Bourgogne-~ 90        Territoi~            95258         23179
##  4          94 Corse       2A        Corse-du~           108760         38512
##  5          93 Provence-A~ 05        Hautes-A~           109892         24895
##  6          84 Auvergne-R~ 15        Cantal              117394         25279
##  7          76 Occitanie   09        Ariège              117406         26698
##  8          94 Corse       2B        Haute-Co~           124873         45639
##  9          93 Provence-A~ 04        Alpes-de~           126459         29255
## 10          44 Grand Est   52        Haute-Ma~           134119         32081
## # ... with 86 more rows, and 6 more variables: present_voters <dbl>,
## #   blank_ballot <dbl>, null_ballot <dbl>, votes_cast <dbl>, `LE PEN` <dbl>,
## #   MACRON <dbl>
```

Note that the default behavior is to arrange values in an ascending order. For descending order use the function desc()

# dplyr::mutate()




The mutate() function adds new variables and preserves existing ones.

Useful functions:

- Mathematical operations: +, -, `log()`
- `lead()`, `lag()`: recovers the value in the next and previous rows, respectively
- `dense_rank()`, `min_rank()`, `percent_rank()`, `row_number()`, `cume_dist()`, `ntile()`: ranking functions
- `cumsum()`, `cummean()`, `cummin()`, `cummax()`, `cumany()`, `cumall()`: cumulative sums, products, and extremes
- `na_if()`, `coalesce()`: transform value into NA and find the first non-NA value of a set of rows, respectively
- `if_else()`, `recode()`, `case_when()`: replace values with condition

```
Note that new variables overwrite existing variables of same name.
```

# dplyr::mutate()

Creating a new variable that gives the voting rate per department

```
round_2 %>%
  mutate(voting_rate = present_voters/registered_voters) %>%
  select(c(1:4), voting_rate, everything())
```

```
## # A tibble: 96 x 13
##    region_code region_name dept_code dept_name voting_rate registered_vote~
##          <dbl> <chr>       <chr>     <chr>           <dbl>            <dbl>
##  1          84 Auvergne-R~ 01        Ain             0.776           415950
##  2          32 Hauts-de-F~ 02        Aisne           0.759           375791
##  3          84 Auvergne-R~ 03        Allier          0.766           253479
##  4          93 Provence-A~ 04        Alpes-de~       0.769           126459
##  5          93 Provence-A~ 05        Hautes-A~       0.773           109892
##  6          93 Provence-A~ 06        Alpes-Ma~       0.739           761780
##  7          84 Auvergne-R~ 07        Ardèche         0.781           249216
##  8          44 Grand Est   08        Ardennes        0.743           194349
##  9          76 Occitanie   09        Ariège          0.773           117406
## 10          44 Grand Est   10        Aube            0.776           203800
## # ... with 86 more rows, and 7 more variables: absent_voters <dbl>,
## #   present_voters <dbl>, blank_ballot <dbl>, null_ballot <dbl>,
## #   votes_cast <dbl>, `LE PEN` <dbl>, MACRON <dbl>
```

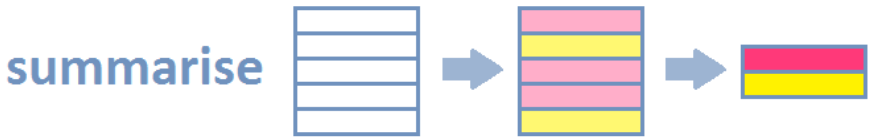
# dplyr::mutate()

Creating a new variable that gives the rank of department according to the number of votes for Emmanuel Macron

```
round_2 %>%
  mutate(rank = min_rank(desc(MACRON))) %>%
  select(dept_name, MACRON, rank) %>%
  arrange(rank)
```

```
## # A tibble: 96 x 3
##    dept_name        MACRON  rank
##    <chr>             <dbl> <int>
##  1 Paris            849257     1
##  2 Nord             669806     2
##  3 Hauts-de-Seine   590963     3
##  4 Rhône            572015     4
##  5 Loire-Atlantique 525200     5
##  6 Bouches-du-Rhône 519335     6
##  7 Gironde          515491     7
##  8 Yvelines         503661     8
##  9 Haute-Garonne    436665     9
## 10 Val-de-Marne     419145    10
## # ... with 86 more rows
```

# dplyr::summarise()




The summarise() function creates a new dataset. It will have one or more rows for each combination of grouping variables; if there are no grouping variables, the output will have a single row summarizing all observations in the input dataset.

Useful functions:

- Center: `mean()`, `median()`
- Spread: `sd()`, `IQR()`, `mad()`
- Range: `min()`, `max()`, `quantile()`
- Position: `first()`, `last()`, `nth()`
- Count: `n()`, `n_distinct()`
- Logical: `any()`, `all()`

# dplyr::summarise()

## Recovering the total number of votes over the country

```
  round_2 %>%
    summarise(total_votes = sum(votes_cast))
```

```
## # A tibble: 1 x 1
##   total_votes
##         <dbl>
## 1    29941125
```

## Total number of votes per region

```
  round_2 %>% group_by(region_name) %>%
    summarise(total_votes = sum(votes_cast))
```

```
## # A tibble: 13 x 2
##    region_name              total_votes
##  * <chr>                          <dbl>
##  1 Auvergne-Rhône-Alpes         3652917
##  2 Bourgogne-Franche-Comté      1348644
##  3 Bretagne                     1726688
##  4 Centre-Val de Loire          1228723
##  5 Corse                         129801
##  6 Grand Est                    2590008
##  7 Hauts-de-France              2828570
##  8 Île-de-France                4858965
##  9 Normandie                    1633534
## 10 Nouvelle-Aquitaine           2929305
## # ... with 3 more rows
```

The group_by() function takes an existing tibble and converts it into a grouped tibble where operations are performed "by group". The ungroup() function removes grouping.

# dplyr::left_join()

The left_join() function joins tibbles x and y by returning all rows from x, and all columns from x and y

```
  geo_data <- read_csv("data/coordinates_regions_2016.csv")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   insee_reg = col_double(),
##   latitude = col_double(),
##   longitude = col_double()
## )
```

```
  round_2 %>% left_join(geo_data, by=c("region_code"="insee_reg")) %>%
    select(region_code, region_name, latitude, longitude, everything())
```

```
## # A tibble: 96 x 14
##    region_code region_name latitude longitude dept_code dept_name
##          <dbl> <chr>          <dbl>     <dbl> <chr>     <chr>
##  1          84 Auvergne-R~     45.5      4.54 01        Ain
##  2          32 Hauts-de-F~     50.0      2.77 02        Aisne
##  3          84 Auvergne-R~     45.5      4.54 03        Allier
##  4          93 Provence-A~     44.0      6.06 04        Alpes-de~
##  5          93 Provence-A~     44.0      6.06 05        Hautes-A~
##  6          93 Provence-A~     44.0      6.06 06        Alpes-Ma~
##  7          84 Auvergne-R~     45.5      4.54 07        Ardèche
##  8          44 Grand Est       48.7      5.61 08        Ardennes
##  9          76 Occitanie       43.7      2.14 09        Ariège
## 10          44 Grand Est       48.7      5.61 10        Aube
## # ... with 86 more rows, and 8 more variables: registered_voters <dbl>,
## #   absent_voters <dbl>, present_voters <dbl>, blank_ballot <dbl>,
## #   null_ballot <dbl>, votes_cast <dbl>, `LE PEN` <dbl>, MACRON <dbl>
```

# dplyr::bind_rows()

Using `dplyr::bind_rows()` function, we combine combine two tibbles to obtain a single tibble with results from both rounds of the presidential election.

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

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

```
## # A tibble: 192 x 22
##    region_code region_name round ARTHAUD ASSELINEAU CHEMINADE `DUPONT-AIGNAN`
##          <dbl> <chr>       <chr>   <dbl>      <dbl>     <dbl>           <dbl>
##  1          11 Île-de-Fra~ Roun~    2897       8337      1472           17997
##  2          11 Île-de-Fra~ Roun~    3706       8195      1247           41505
##  3          11 Île-de-Fra~ Roun~    2872       8148      1358           32906
##  4          11 Île-de-Fra~ Roun~    2924       7514      1241           44793
##  5          11 Île-de-Fra~ Roun~    2447       8453      1345           21359
##  6          11 Île-de-Fra~ Roun~    3235       8739      1088           16601
##  7          11 Île-de-Fra~ Roun~    2749       7303      1066           26252
##  8          11 Île-de-Fra~ Roun~    2752       7702       978           24790
##  9          11 Île-de-Fra~ Roun~      NA         NA        NA              NA
## 10          11 Île-de-Fra~ Roun~      NA         NA        NA              NA
## # ... with 182 more rows, and 15 more variables: FILLON <dbl>, HAMON <dbl>,
## #   LASSALLE <dbl>, `LE PEN` <dbl>, MACRON <dbl>, MÉLENCHON <dbl>,
## #   POUTOU <dbl>, dept_code <chr>, dept_name <chr>, registered_voters <dbl>,
## #   absent_voters <dbl>, present_voters <dbl>, blank_ballot <dbl>,
## #   null_ballot <dbl>, votes_cast <dbl>
```

Note that the `bind_rows()` function match tibbles' columns by name, and any missing columns will be filled with NA.

# Tidying messy data with `tidyr`

In a tidy dataset

- Every column is a variable
- Every row is an observation
- Every cell is a single value

There are five main categories of tidyr functions:

- Pivotting functions convert between long and wide formats: `gather()`, `spread()`
- Rectangling functions turns deeply nested lists (as from JSON) into tidy tibbles: `unnest_longer()`, `unnest_wider()`
- Nesting functions convert grouped data into a form where each group becomes a single row containing a nested data frame: `nest()`, `unnest()`
- Splitting and combining character columns: `separate()`, `extract()`, `unite()`
- Treating missing values: `complete()`, `drop_na()`, `fill()`, `replace_na()`

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

# tidyr::gather()

```
round_2 %>% gather(candidate, votes, c(`LE PEN`, MACRON)) %>%
  arrange(region_name, dept_name) %>%
  select(region_name, candidate, votes, everything())
```

```
## # A tibble: 192 x 12
##    region_name candidate  votes region_code dept_code dept_name registered_vote~
##    <chr>       <chr>      <dbl>       <dbl> <chr>     <chr>                 <dbl>
##  1 Auvergne-R~ LE PEN    111421          84 01        Ain                  415950
##  2 Auvergne-R~ MACRON    173809          84 01        Ain                  415950
##  3 Auvergne-R~ LE PEN     60207          84 03        Allier               253479
##  4 Auvergne-R~ MACRON    106579          84 03        Allier               253479
##  5 Auvergne-R~ LE PEN     63109          84 07        Ardèche              249216
##  6 Auvergne-R~ MACRON    104599          84 07        Ardèche              249216
##  7 Auvergne-R~ LE PEN     23938          84 15        Cantal               117394
##  8 Auvergne-R~ MACRON     55411          84 15        Cantal               117394
##  9 Auvergne-R~ LE PEN     94312          84 26        Drôme                369462
## 10 Auvergne-R~ MACRON    157992          84 26        Drôme                369462
## # ... with 182 more rows, and 5 more variables: absent_voters <dbl>,
## #   present_voters <dbl>, blank_ballot <dbl>, null_ballot <dbl>,
## #   votes_cast <dbl>
```

# Why should we use tidyr?

Example 1. Calculating the number of votes per candidate and department

Using the input data format

```
round_2 %>% group_by(region_name) %>%
  summarise(votesLePen = sum(`LE PEN`),
            votesMacron = sum(MACRON),
            .groups='drop')
```

```
## # A tibble: 13 x 3
##    region_name              votesLePen votesMacron
##  * <chr>                         <dbl>       <dbl>
##  1 Auvergne-Rhône-Alpes        1200726     2452191
##  2 Bourgogne-Franche-Comté      532935      815709
##  3 Bretagne                     425462     1301226
##  4 Centre-Val de Loire          450750      777973
##  5 Corse                         62982       66819
##  6 Grand Est                   1089356     1500652
##  7 Hauts-de-France             1331169     1497401
##  8 Île-de-France               1033686     3825279
##  9 Normandie                    621472     1012062
## 10 Nouvelle-Aquitaine           918237     2011068
## # ... with 3 more rows
```

Using the data format after applying `tidyr::gather()`

```
round_2 %>% group_by(region_name, candidate) %>%
  summarise(votes = sum(votes),
            .groups='drop')
```

```
## # A tibble: 26 x 3
##    region_name              candidate   votes
##  * <chr>                    <chr>       <dbl>
##  1 Auvergne-Rhône-Alpes     LE PEN    1200726
##  2 Auvergne-Rhône-Alpes     MACRON    2452191
##  3 Bourgogne-Franche-Comté  LE PEN     532935
##  4 Bourgogne-Franche-Comté  MACRON     815709
##  5 Bretagne                 LE PEN     425462
##  6 Bretagne                 MACRON    1301226
##  7 Centre-Val de Loire      LE PEN     450750
##  8 Centre-Val de Loire      MACRON     777973
##  9 Corse                    LE PEN      62982
## 10 Corse                    MACRON      66819
## # ... with 16 more rows
```

# Why should we use tidyr?

Example 2. Identifying the winner candidate per department

```
round_2 %>%
  group_by(dept_name) %>%
  mutate(rank = min_rank(desc(votes))) %>%
  arrange(dept_name, rank) %>%
  mutate(winner = if_else(rank == 1, TRUE, FALSE)) %>%
  select(dept_name, candidate, votes, rank, winner)
```

```
## # A tibble: 192 x 5
## # Groups:   dept_name [96]
##    dept_name                candidate  votes  rank winner
##    <chr>                    <chr>      <dbl> <int> <lgl>
##  1 Ain                      MACRON    173809     1 TRUE
##  2 Ain                      LE PEN    111421     2 FALSE
##  3 Aisne                    LE PEN    133939     1 TRUE
##  4 Aisne                    MACRON    119202     2 FALSE
##  5 Allier                   MACRON    106579     1 TRUE
##  6 Allier                   LE PEN     60207     2 FALSE
##  7 Alpes-de-Haute-Provence MACRON      48994     1 TRUE
##  8 Alpes-de-Haute-Provence LE PEN      34817     2 FALSE
##  9 Alpes-Maritimes          MACRON    278407     1 TRUE
## 10 Alpes-Maritimes          LE PEN    224544     2 FALSE
## # ... with 182 more rows
```

# tidyr::spread()

```
round_2 %>% spread(candidate, votes) %>%
  select(region_name, `LE PEN`, MACRON, everything())
```

```
## # A tibble: 96 x 12
##    region_name `LE PEN` MACRON region_code dept_code dept_name registered_vote~
##    <chr>          <dbl>  <dbl>       <dbl> <chr>     <chr>                 <dbl>
##  1 Auvergne-R~   111421 173809          84 01        Ain                  415950
##  2 Auvergne-R~    60207 106579          84 03        Allier               253479
##  3 Auvergne-R~    63109 104599          84 07        Ardèche              249216
##  4 Auvergne-R~    23938  55411          84 15        Cantal               117394
##  5 Auvergne-R~    94312 157992          84 26        Drôme                369462
##  6 Auvergne-R~   199097 383197          84 38        Isère                856330
##  7 Auvergne-R~   123714 218603          84 42        Loire                510903
##  8 Auvergne-R~    44112  76233          84 43        Haute-Lo~            177116
##  9 Auvergne-R~    88155 219437          84 63        Puy-de-D~            460919
## 10 Auvergne-R~   205317 572015          84 69        Rhône               1155506
## # ... with 86 more rows, and 5 more variables: absent_voters <dbl>,
## #   present_voters <dbl>, blank_ballot <dbl>, null_ballot <dbl>,
## #   votes_cast <dbl>
```

# Manipulating strings with `stringr`

The stringr package provides a set of functions designed to ease the process of manipulating strings, which play a big role in many data cleaning and presentation tasks.

The package provides six main categories of functions:

- Detect matches with a given pattern: `str_detect()`, detects the presence or absence of a pattern in a string; `str_which()`, find positions matching a pattern; `str_count()`, count the number of matches in a string.
- Subset strings: `str_sub()`, extract and replace substrings from a character vector; `str_subset()` and `str_extract()`, keep and extract matching patterns from a string, respectively; `str_match()`, extract matched groups.
- Manage lenghts of strings: `str_length()`, the length of a string; `str_pad()`, pad a string; `str_trunc()`, truncate a character string; `str_trim()`, remove extra white spaces from a string; `str_wrap()`, wrap strings into nicely formatted paragraphs.
- Mutate strings: `str_replace()`, replace matched patterns in a string.
- Join and split: `str_split()`, split up a string into pieces; `str_glue()`, interpolate strings.
- Order strings: `str_order()`, order or sort a character vector.

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

# Abstract Data Visualization

# Visualizing data with ggplot2

ggplot2 is a system for creating 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

The data must follow a tidy structure, which means:

- 1 row = 1 observation
- 1 column = 1 homogeneous and consistent variable

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

```
## `summarise()` has grouped output by 'region_code', 'region_name'. You can override using the `.groups` argum
```

```
## # A tibble: 26 x 4
## # Groups:   region_code, region_name [13]
##    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
##  6          27 Bourgogne-Franche-Comté MACRON     815709
##  7          28 Normandie               LE PEN     621472
##  8          28 Normandie               MACRON    1012062
##  9          32 Hauts-de-France         LE PEN    1331169
## 10          32 Hauts-de-France         MACRON    1497401
## # ... with 16 more rows
```

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

# 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.

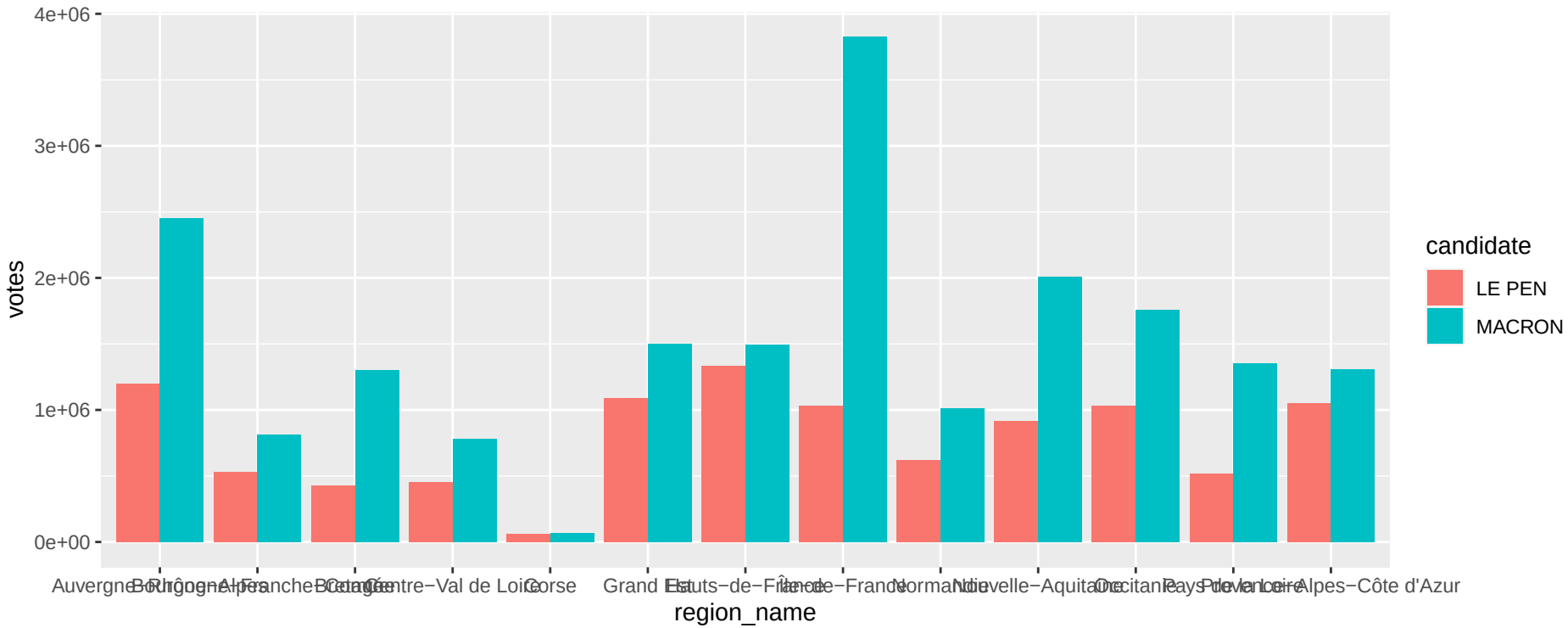

```
A list of available geometric components:  https://ggplot2.tidyverse.org/reference/index.html#section-layer-geoms
```

# 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. Further, it can be customized with specific properties (alpha,color, fill, position)

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

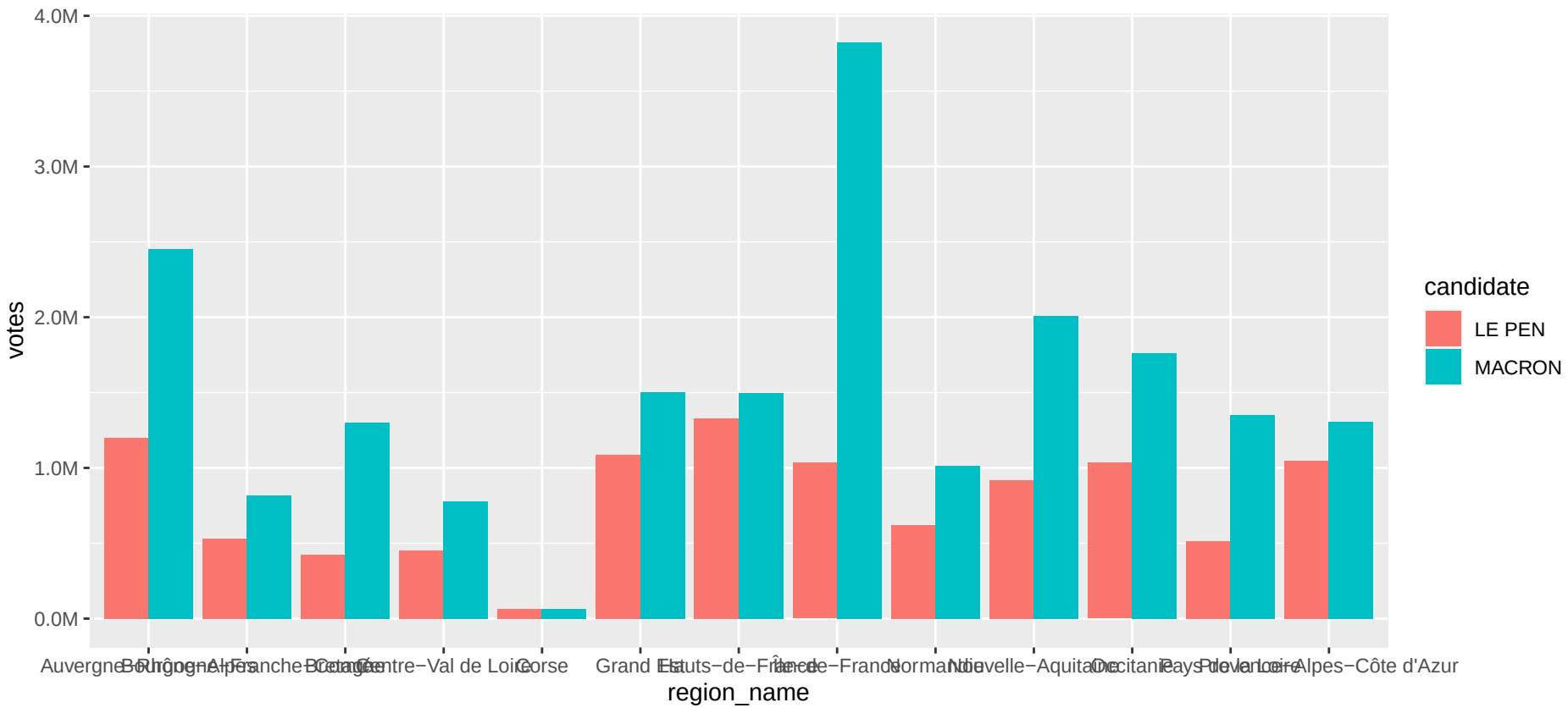


# The scales

The scales package provides methods for automatically determining breaks and labels for axes and legends.

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

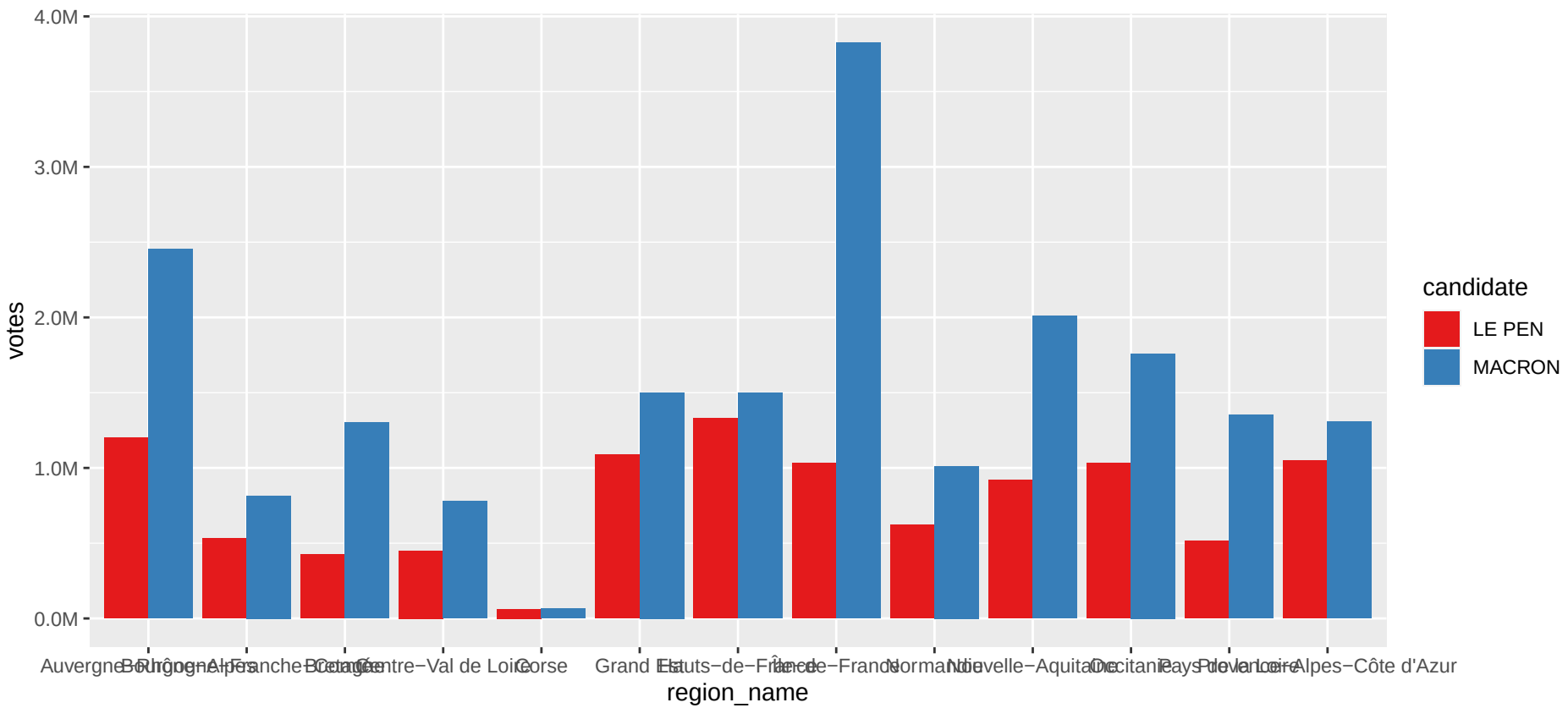
```
library(scales)

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



# The scales: color

You can use predefined sequential, diverging, and qualitative color scales based on https://colorbrewer2.org/ created by Cynthia Brewer

```
plot + scale_fill_brewer(palette = 'Set1')
```

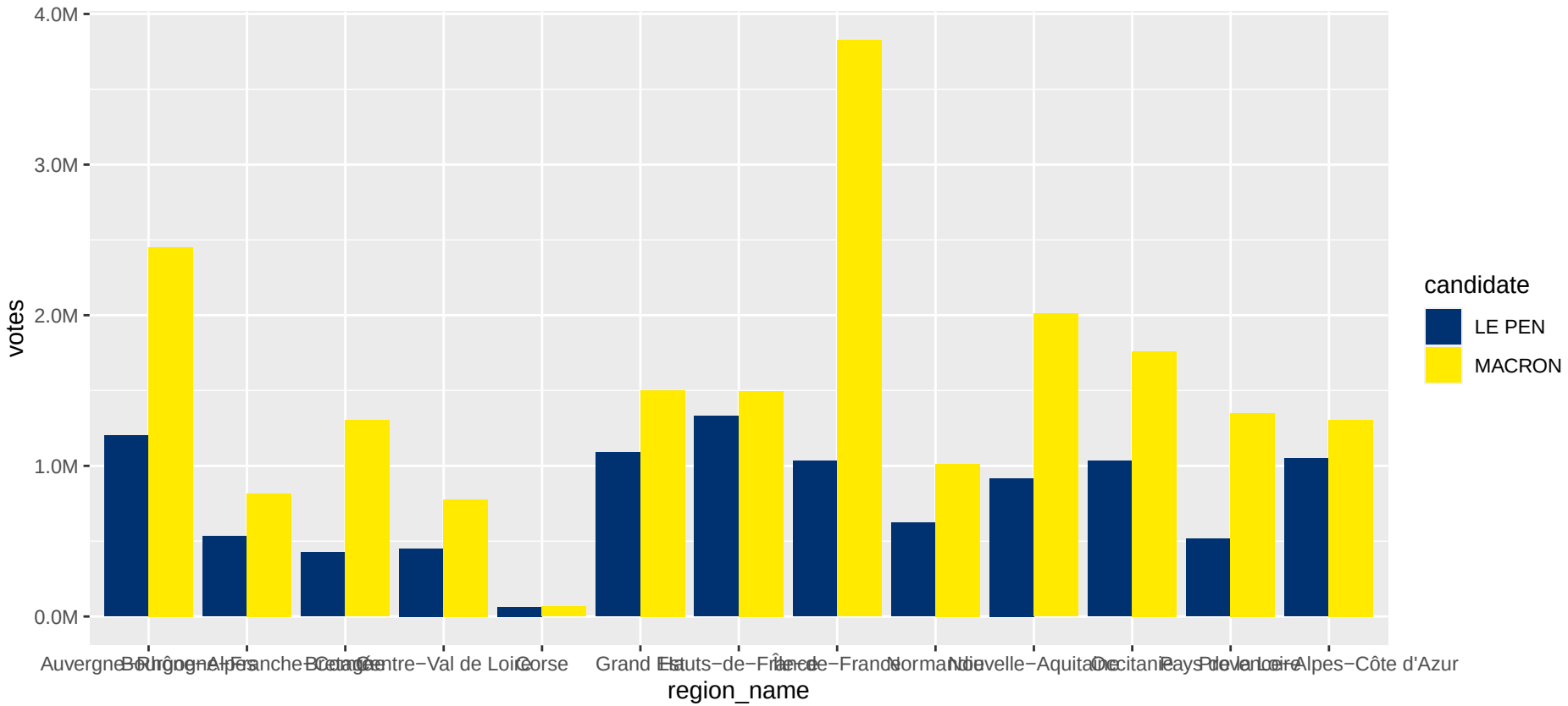



More information at https://rdrr.io/cran/RColorBrewer/man/ColorBrewer.html

# The scales: color

Or you can use a manual color scale

```
plot <- plot +
  scale_fill_manual(values = c('#003171', '#ffea00'))
```

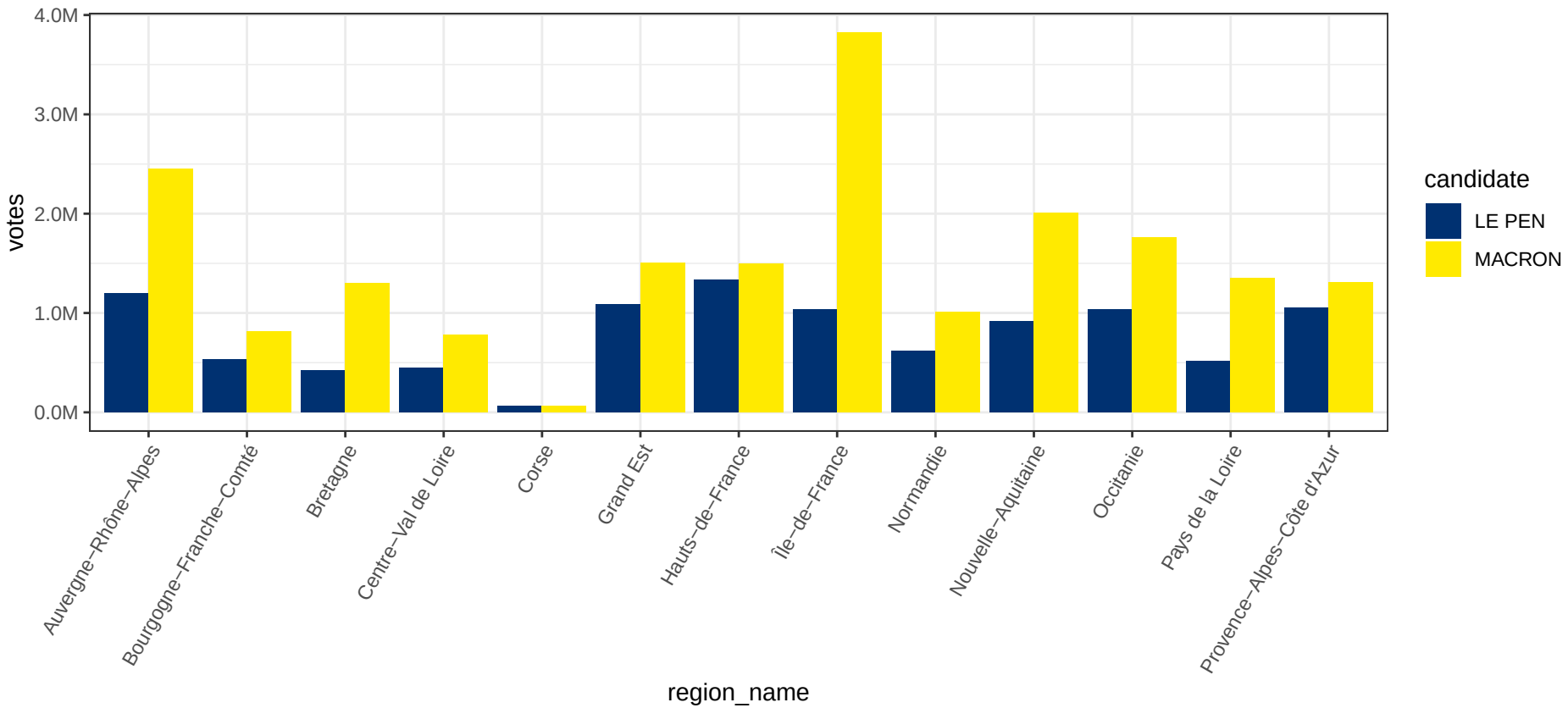


# Decoration components: theme

Themes are a powerful way to customize the non-data components of your plots.

You can include a predefined theme element (theme_gray(), theme_bw(), theme_dark(), etc), and use the theme() element to customize single components.

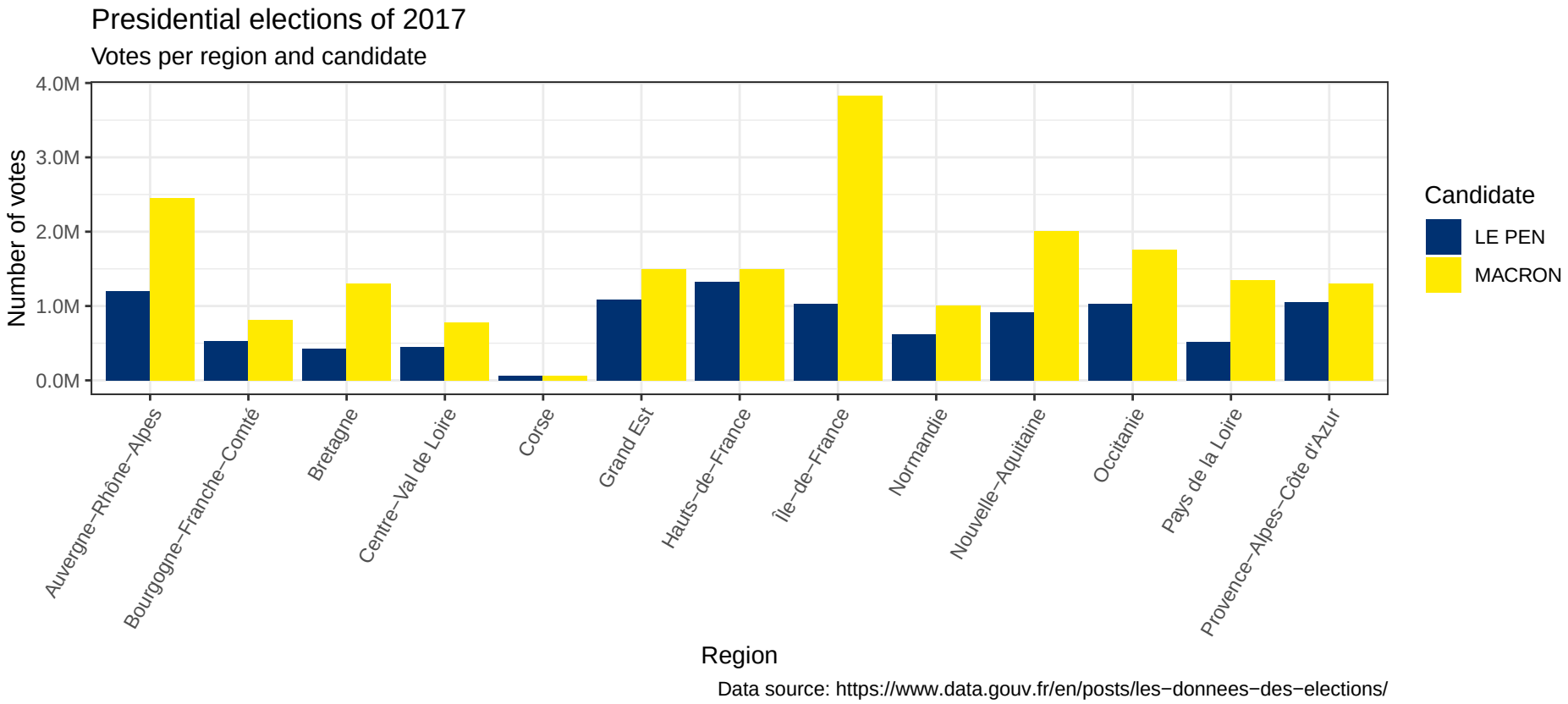
```
plot <- plot + theme_bw() +
  theme(axis.text.x = element_text(angle = 60, hjust = 1))
```



# Decoration components: title and legends

The labs() function allows to include information such as the title, subtitle, and caption of a graphic, and axis' labels.

```
plot <- plot + labs(title = "Presidential elections of 2017",
          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'))
```



# Summarized chunk code of the bar chart

```
ggplot(plot_df) +
  geom_col(aes(x = region_name, y = votes, fill = candidate), # geometric object
           position = 'dodge') +
  scale_y_continuous(labels = number_format(scale = 1/1000000, # y axis format
                                            suffix = 'M')) +
  scale_fill_manual(values = c('#003171', '#ffea00')) +         # 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

We can combine several geometric objects to represent different variables.

1. We 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))
```

```
## `summarise()` has grouped output by 'region_code'. You can override using the `.groups` argument.
```

```
## # A tibble: 39 x 4
## # Groups:   region_code [13]
##    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
##  6          24 Centre-Val de Loire     absent_voters  425870
##  7          27 Bourgogne-Franche-Comté blank_ballot   145595
##  8          27 Bourgogne-Franche-Comté null_ballot     53864
##  9          27 Bourgogne-Franche-Comté absent_voters  453202
## 10          28 Normandie               blank_ballot   165820
## # ... with 29 more rows
```

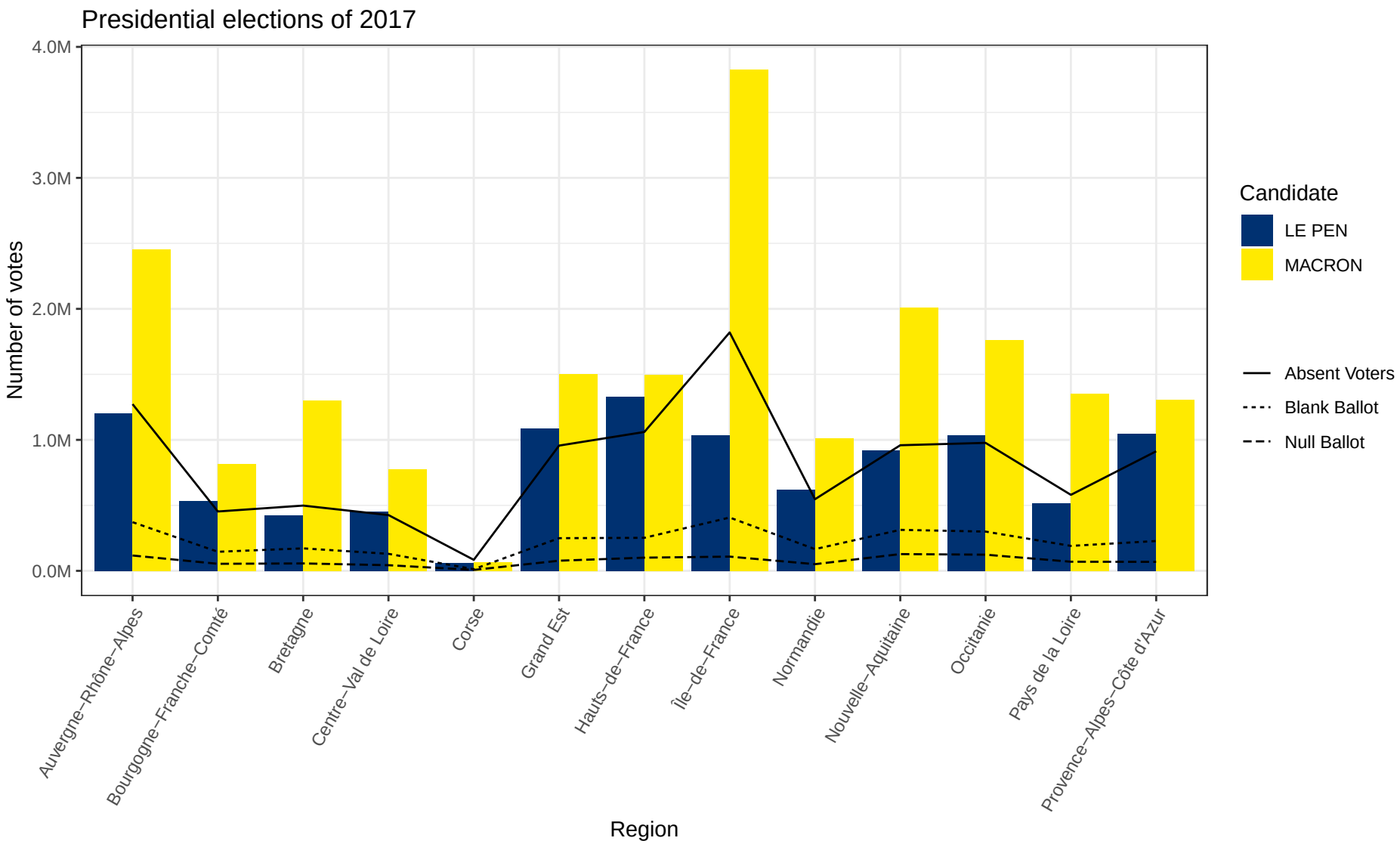
Note that the distinct() function modifies the tibble, keeping only the variable(s) given as argument. Use the option .keep_all = TRUE to also keep the remaining variables.

# Combining geometric objects

2. We 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('#003171', '#ffea00')) +
  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



# Decomposition components: facets

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

```
## # A tibble: 192 x 22
##   region_code region_name round ARTHAUD ASSELINEAU CHEMINADE `DUPONT-AIGNAN`
##         <dbl> <chr>       <chr>   <dbl>      <dbl>     <dbl>           <dbl>
## 1          11 Île-de-Fra~ Roun~    2897       8337      1472           17997
## 2          11 Île-de-Fra~ Roun~    3706       8195      1247           41505
## 3          11 Île-de-Fra~ Roun~    2872       8148      1358           32906
## 4          11 Île-de-Fra~ Roun~    2924       7514      1241           44793
## 5          11 Île-de-Fra~ Roun~    2447       8453      1345           21359
## # ... with 187 more rows, and 15 more variables: FILLON <dbl>, HAMON <dbl>,
## #   LASSALLE <dbl>, `LE PEN` <dbl>, MACRON <dbl>, MÉLENCHON <dbl>,
## #   POUTOU <dbl>, dept_code <chr>, dept_name <chr>, registered_voters <dbl>,
## #   absent_voters <dbl>, present_voters <dbl>, blank_ballot <dbl>,
## #   null_ballot <dbl>, votes_cast <dbl>
```

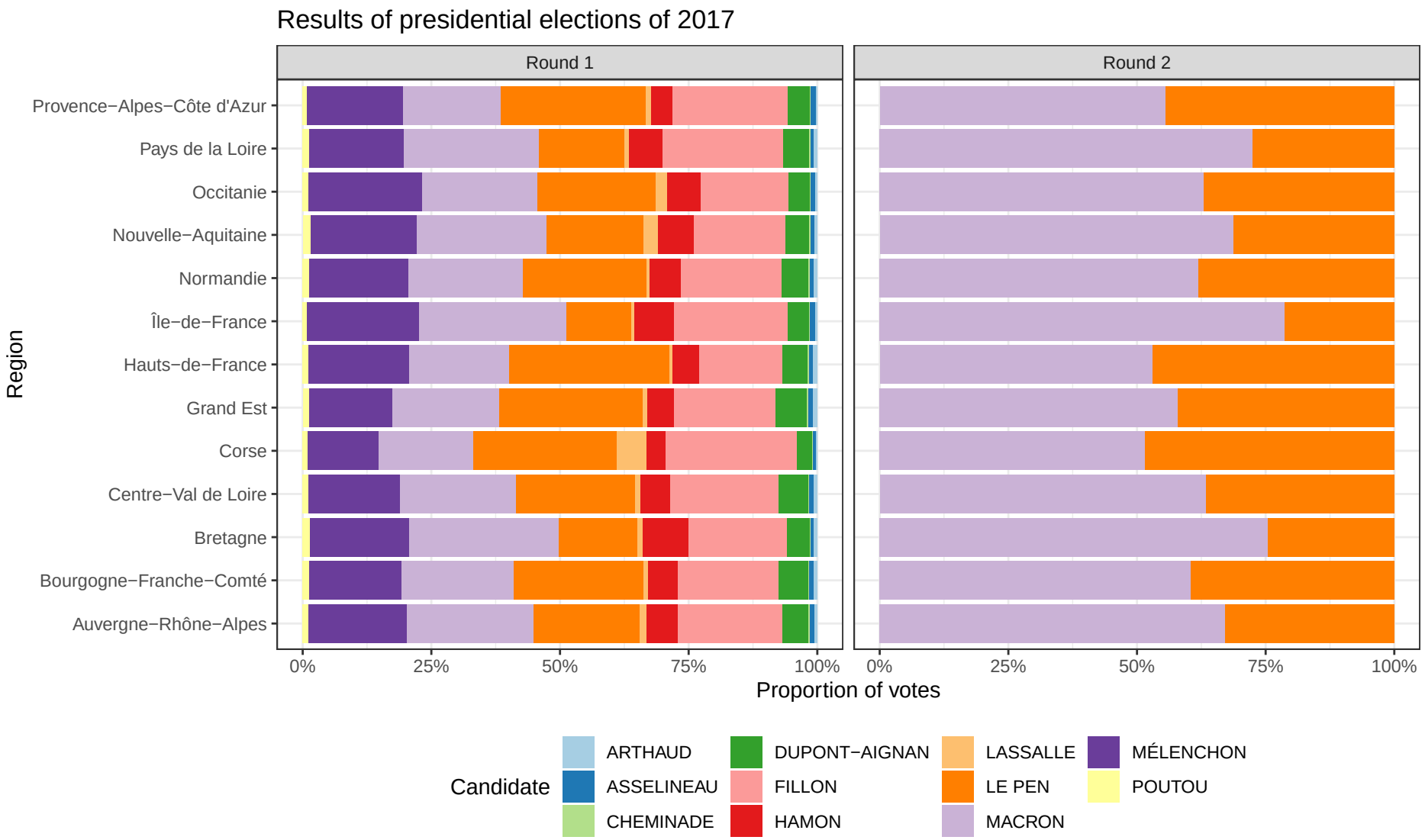
```
results <- results %>%
  gather(candidate, votes, c(ARTHAUD:POUTOU)) %>%
  group_by(region_code, region_name, candidate, round) %>%
  summarise(votes = sum(votes), .groups='drop')
```

```
## # A tibble: 286 x 5
##   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
## # ... with 281 more rows
```

# Decomposition components: facets

```
ggplot(results, 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
```

# Decomposition components: facets



# 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.

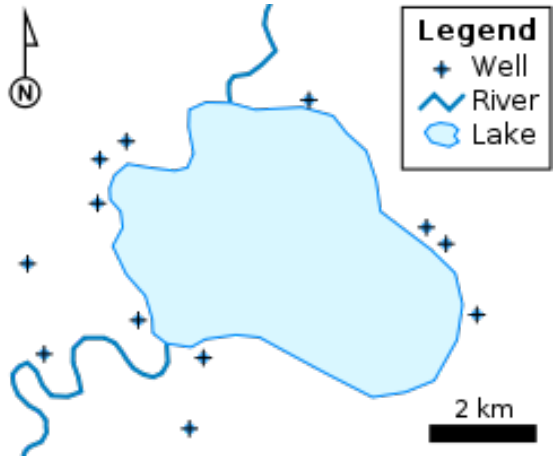
One of the most popular GIS vector file format is the Shapefile. It is a nontopological format for storing the geometric location and attribute information of geographical features. It stores information via a collection of at least three files:

- .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)
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
## bbox:           xmin: -61.80984 ymin: -21.38963 xmax: 55.83663 ymax: 51.08899
## geographic 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

```
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 REG0~ HAUTS-~        32 Hauts~ 2000~ Hauts-de-F~ LE PEN    1.33e6
##  2 REG0~ HAUTS-~        32 Hauts~ 2000~ Hauts-de-F~ MACRON    1.50e6
##  3 REG0~ BOURGO~        27 Bourg~ 2000~ Bourgogne-~ LE PEN    5.33e5
##  4 REG0~ BOURGO~        27 Bourg~ 2000~ Bourgogne-~ MACRON    8.16e5
##  5 REG0~ OCCITA~        76 Occit~ 2000~ Occitanie   LE PEN    1.03e6
##  6 REG0~ OCCITA~        76 Occit~ 2000~ Occitanie   MACRON    1.76e6
##  7 REG0~ ILE-DE~        11 Île-d~ <NA>  Île-de-Fra~ LE PEN    1.03e6
##  8 REG0~ ILE-DE~        11 Île-d~ <NA>  Île-de-Fra~ MACRON    3.83e6
##  9 REG0~ AUVERG~        84 Auver~ 2000~ Auvergne-R~ LE PEN    1.20e6
## 10 REG0~ AUVERG~        84 Auver~ 2000~ Auvergne-R~ MACRON    2.45e6
## # ... with 16 more rows, and 1 more variable: geometry <MULTIPOLYGON [°]>
```

Note that every aforementioned package for data manipulation can also be used for geospatial datasets

# Static thematic maps with ggplot2

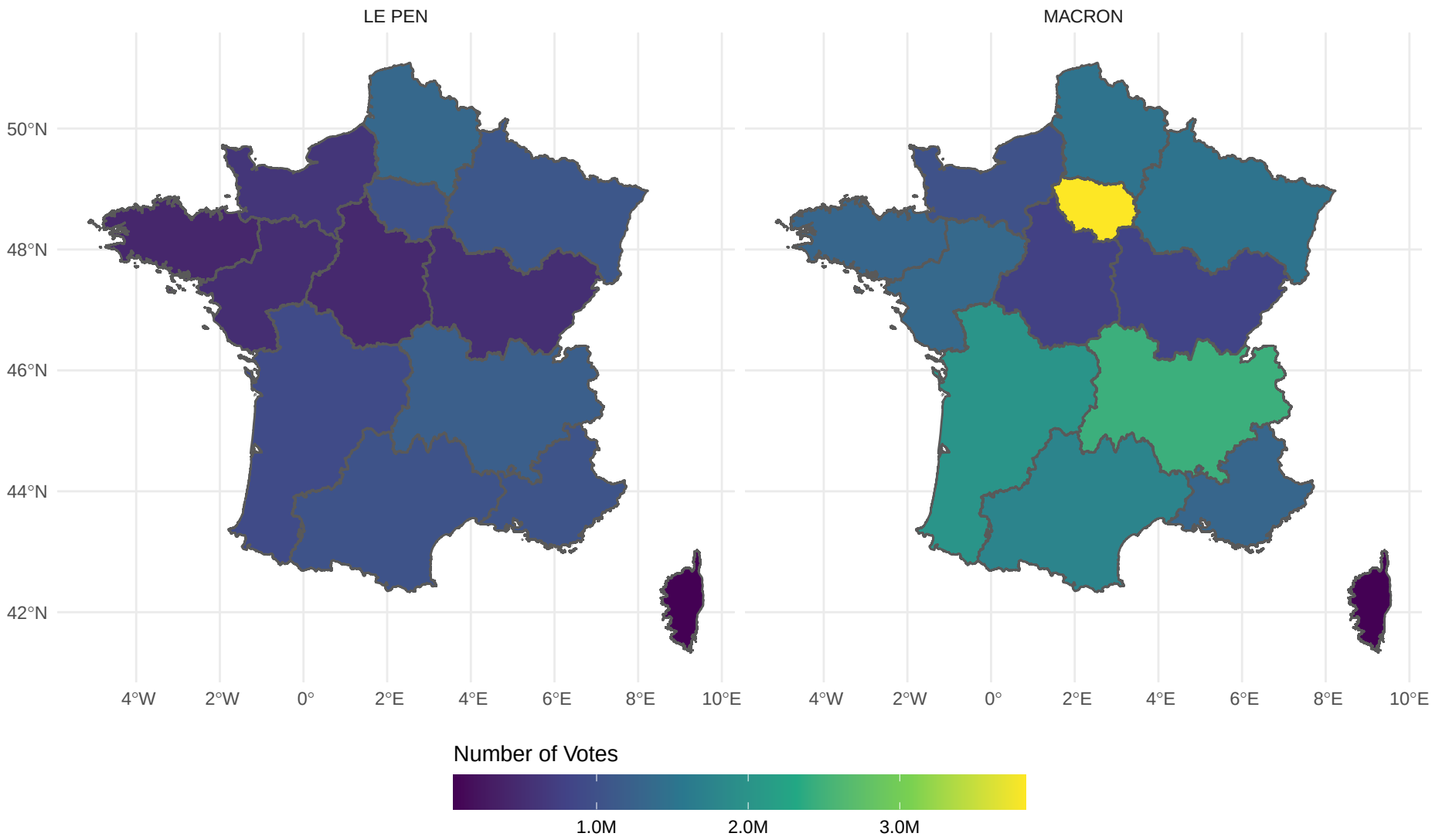
ggplot2 provides a set of geom (geom_sf()), stat (stat_sf()), and coord (coord_sf()) components to visualize sf objects.

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 scales provide color maps that are perceptually uniform in both color and
black-and-white, and are designed to be perceived by viewers with common forms of color blindness
```

# Static thematic maps with ggplot2



# Dynamic thematic maps with `leaflet`

Leaflet is one of the most popular open-source JavaScript libraries for interactive maps.

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

```
library(leaflet)
```

```
## Warning: package 'leaflet' was built under R version 4.0.5
```

```
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'))

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

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

map_leaflet <- leaflet(data = plot_sf) %>%
  addProviderTiles(providers$OpenStreetMap) %>%
  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")
```

The htmlwidgets package allows to save the graphic as an HTML object for further use.

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

```
library(htmlwidgets)
```

```
## Warning: package 'htmlwidgets' was built under R version 4.0.5
```

# Dynamic thematic maps with `leaflet`

# Interactive Visualization Dashboards

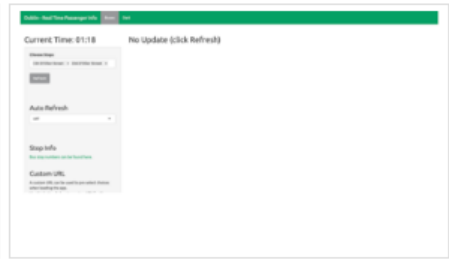
# Shiny from RStudio (R Shiny)

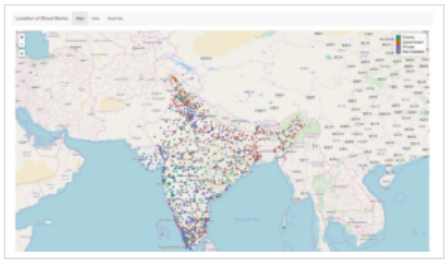
Shiny is a R package for building interactive web apps directly from R.

Voronoys - Understanding voters' profile in Brazilian elections

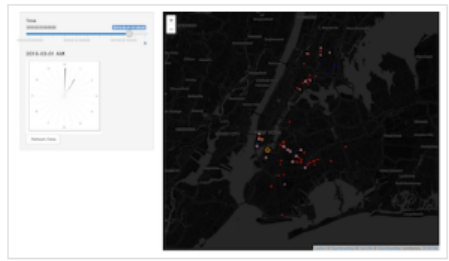

Crime Watch

Pasture Potential Tool for improving dairy farm profitability and environmental impact


Dublin Transport Info

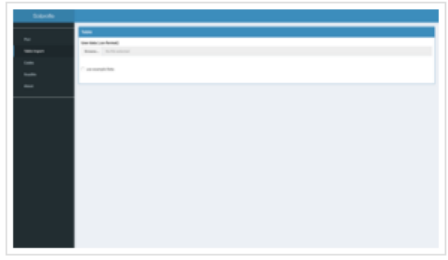

Locating Blood Banks in India

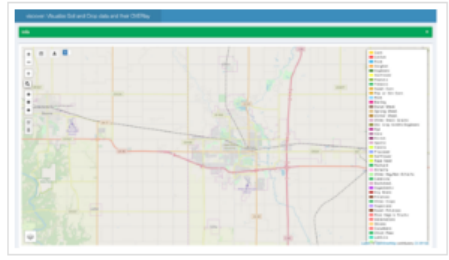
Utah Lake Water Quality Profile Dashboard


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

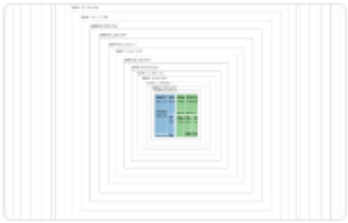
  - Possibly the main advantage of using R Shiny

However

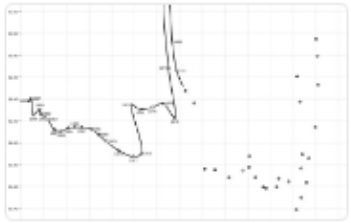
- if you are not a relatively experienced R user, learning how to use R Shiny can be difficult.

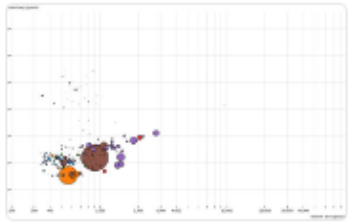
# JavaScript-based Visualizations (d3.js)

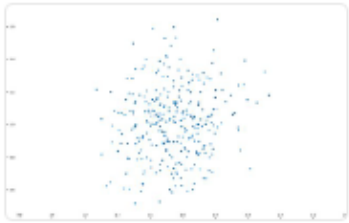
D3 (Data Driven Documents) is a JavaScript library developed by Mike Bostock and huge community of volunteer developers for manipulating documents based on data.


Animated treemap

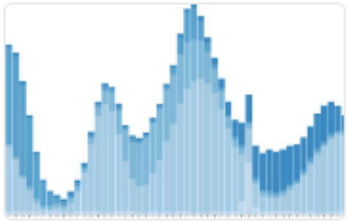

Temporal force-directed graph

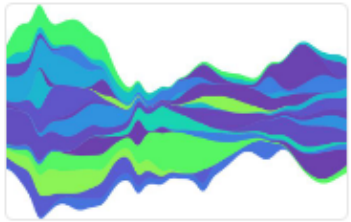

Connected scatterplot


The wealth & health of nations


Scatterplot tour

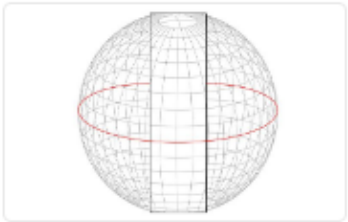
Bar chart race


Stacked-to-grouped bars


Streamgraph transitions

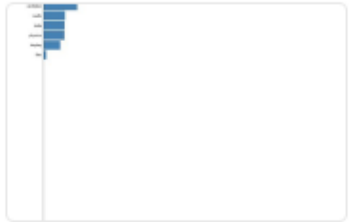

Smooth zooming


Zoom to bounding box

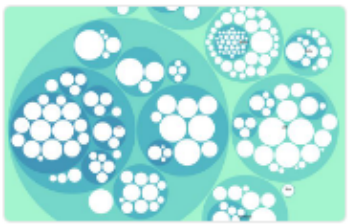

Orthographic to equirectangular

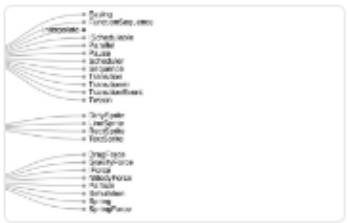

World tour


Walmart's growth

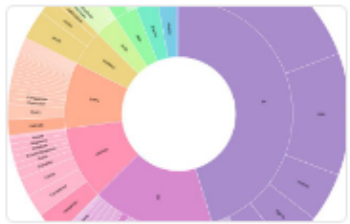

Hierarchical bar chart

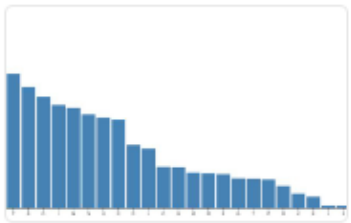

Zoomable treemap


Zoomable circle packing


Collapsible tree


Zoomable icicle


Zoomable sunburst


Sortable bar chart

# Why should we use d3.js?

- If you want to go beyond static graphics and create powerful and fully interactive visualizations, d3 is your choice.
- d3 relies completely on you, the developer, to specify what you want to see.
- It can be difficult to learn d3, 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.

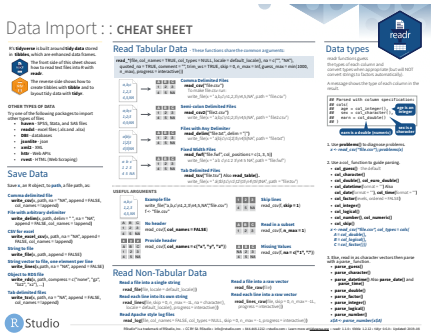
However

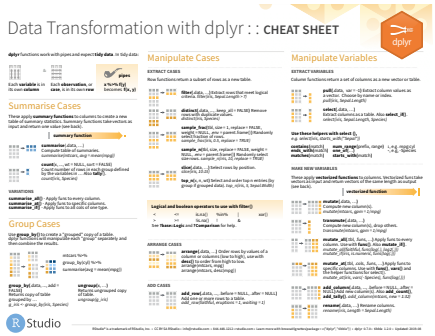
- there is no such a thing as whipping up a quick d3 dashboard. Even the most basic dashboard can take hundreds of lines of code. But, for commonly used chart, you can usually find reusable code with a few Google searches.
- JavaScript is not very easy for beginners.

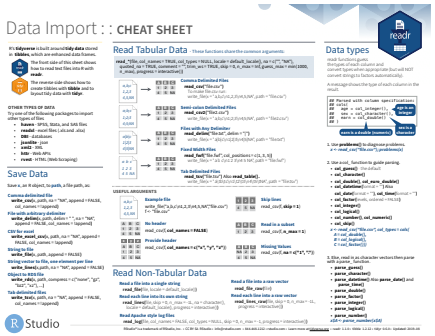
More about D3 in the next class

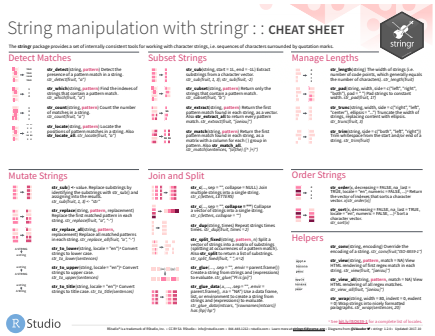
# Cheatsheets

## Data manipulation




## Data visualization