UNIVERSITÉ CÔTE D'AZUR

Data Manipulation with R

Aline Menin

Maîtresse de Conférences, Université Côte d'Azur

aline.menin@univ-cotedazur.fr

2023/2024

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

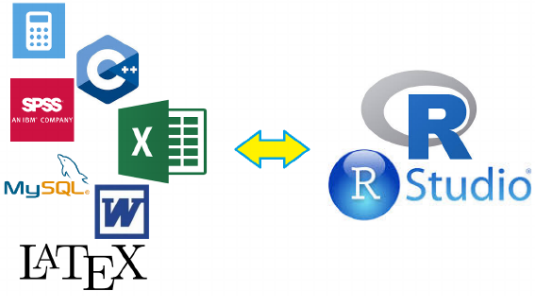
It allows us to do

- 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



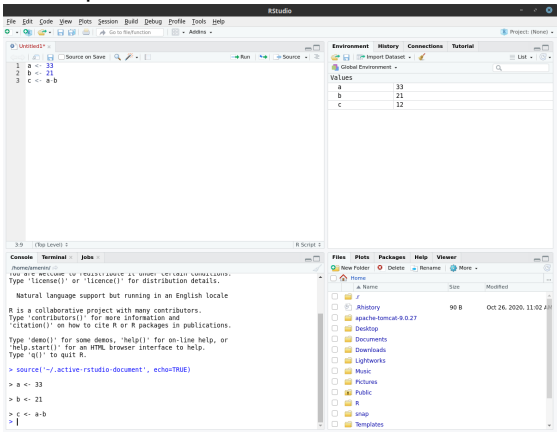
Although R is quite cumbersome at first, it quickly reveals itself as an incredible asset compared to spreadsheets.

How to use R?

To simplify the usage of R, we normally use it via an adapted editor: RStudio

RStudio contains four windows:

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




```
Download at https://rstudio.com/products/rstudio/download/
```

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 and readxl packages for reading/writing data tables
- 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
- the ggplot2 package for producing nice looking and configurable graphics

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

Data import with readr

read_*(file, col_names = TRUE, col_types = NULL, col_select = NULL, id = NULL, locale, n_max = Inf, skip = 0, na = c("", "NA"), guess_max = min(1000, n_max), show_col_types = TRUE) See ?read_delim

A	B	C
1	2	3
4	5	NA

read_delim("file.txt", delim = "|") Read files with any delimiter. If no delimiter is specified, it will automatically guess.
To make file.txt, run: write_file("A|B|C\n1|2|3\n4|5|NA", file = "file.txt")

A	B	C
1	2	3
4	5	NA

read_csv("file.csv") Read a comma delimited file with period decimal marks.
write_file("A,B,C\n1,2,3\n4,5,NA", file = "file.csv")

A	B	C
1.5	2	3
4.5	5	NA

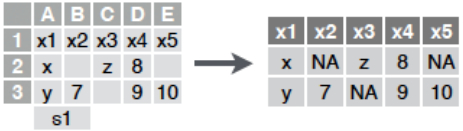
read_csv2("file2.csv") Read semicolon delimited files with comma decimal marks.
write_file("A;B;C\n1,5;2;3\n4,5;5;NA", file = "file2.csv")

A	B	C
1	2	3
4	5	NA

read_tsv("file.tsv") Read a tab delimited file. Also **read_table()**.
read_fwf("file.tsv", fwf_widths(c(2, 2, NA))) Read a fixed width file.
write_file("A\tB\tC\n1\t2\t3\n4\t5\tNA\n", file = "file.tsv")

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

Data import with `readxl`



read_excel(path, sheet = NULL, range = NULL)
Read a .xls or .xlsx file based on the file extension. See front page for more read arguments. Also **read_xls()** and **read_xlsx()**.
read_excel("excel_file.xlsx")

More information at https://readxl.tidyverse.org/

Handling JSON files with `tidyjson`

Tidyjson provides a grammar for turning complex JSON data into tidy data frames that are easy to work with in the tidyverse.

```
# Define a simple people JSON collection
people <- c('{"age": 32, "name": {"first": "Bob",   "last": "Smith"}}',
            '{"age": 54, "name": {"first": "Susan", "last": "Doe"}}',
            '{"age": 18, "name": {"first": "Ann",   "last": "Jones"}}')

# Tidy the JSON data
people %>% spread_all
#> # A tbl_json: 3 x 5 tibble with a "JSON" attribute
#>   ..JSON                  document.id   age name.first name.last
#>   <chr>                         <int> <dbl> <chr>      <chr>
#> 1 "{\"age\":32,\"name..."           1    32 Bob        Smith
#> 2 "{\"age\":54,\"name..."           2    54 Susan      Doe
#> 3 "{\"age\":18,\"name..."           3    18 Ann        Jones
```

More information at https://cran.r-project.org/web/packages/tidyjson/vignettes/tidyjson.html

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')
```

Note: backticks are added to the name of variables containing white spaces, which must be used to access it later.

Data format: data.frame

A data frame is a table in which each column contains values of one variable and each row contains one set of values from each column.

- Non-empty column names
- Unique row names
- Data type: numeric, factor or character
- All columns have the same amount of data records

```
##   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
##   registered_voters absent_voters present_voters blank_ballot null_ballot
## 1            415950         93130         322820        28852        8738
## 2            375791         90745         285046        22838        9067
## 3            253479         59294         194185        18877        8522
## 4            126459         29255          97204         9671        3722
## 5            109892         24895          84997         8670        2699
##   votes_cast LE PEN MACRON
## 1     285230 111421 173809
## 2     253141 133939 119202
## 3     166786  60207 106579
## 4      83811  34817  48994
## 5      73628  26417  47211
```

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_voters absent_voters
##          <dbl> <chr>         <chr>     <chr>                 <dbl>         <dbl>
##  1          84 Auvergne-Rhô~ 01        Ain                  415950         93130
##  2          32 Hauts-de-Fra~ 02        Aisne                375791         90745
##  3          84 Auvergne-Rhô~ 03        Allier               253479         59294
##  4          93 Provence-Alp~ 04        Alpes-de~            126459         29255
##  5          93 Provence-Alp~ 05        Hautes-A~            109892         24895
##  6          93 Provence-Alp~ 06        Alpes-Ma~            761780        198631
##  7          84 Auvergne-Rhô~ 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
## # i 86 more rows
## # i 6 more variables: present_voters <dbl>, blank_ballot <dbl>,
## #   null_ballot <dbl>, votes_cast <dbl>, `LE PEN` <dbl>, MACRON <dbl>
```

Variable selection

Use the $ symbol to select a particular variable in both data.frame and tibble

```
round_2$region_name
```

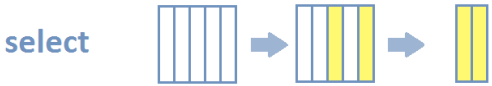
```
##  [1] "Auvergne-Rhône-Alpes"       "Hauts-de-France"
##  [3] "Auvergne-Rhône-Alpes"       "Provence-Alpes-Côte d'Azur"
##  [5] "Provence-Alpes-Côte d'Azur" "Provence-Alpes-Côte d'Azur"
##  [7] "Auvergne-Rhône-Alpes"       "Grand Est"
##  [9] "Occitanie"                  "Grand Est"
## [11] "Occitanie"                  "Occitanie"
## [13] "Provence-Alpes-Côte d'Azur" "Normandie"
## [15] "Auvergne-Rhône-Alpes"       "Nouvelle-Aquitaine"
## [17] "Nouvelle-Aquitaine"         "Centre-Val de Loire"
## [19] "Nouvelle-Aquitaine"         "Bourgogne-Franche-Comté"
## [21] "Bretagne"                   "Nouvelle-Aquitaine"
## [23] "Nouvelle-Aquitaine"         "Bourgogne-Franche-Comté"
## [25] "Auvergne-Rhône-Alpes"       "Normandie"
## [27] "Centre-Val de Loire"        "Bretagne"
## [29] "Corse"                      "Corse"
## [31] "Occitanie"                  "Occitanie"
## [33] "Occitanie"                  "Nouvelle-Aquitaine"
## [35] "Occitanie"                  "Bretagne"
## [37] "Centre-Val de Loire"        "Centre-Val de Loire"
```

Manipulating data with `dplyr`

- 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(): create 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()



Select variables by 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
```

dplyr::select()

Select variables by position

```
round_2 %>% select(1:4)
```

```
## # A tibble: 96 x 4
##    region_code region_name                dept_code dept_name
##          <dbl> <chr>                      <chr>     <chr>
##  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
## # i 86 more rows
```

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

dplyr::select()

Select variables by excluding variables

```
round_2 %>% select(-c(2:7), -region_code)
```

```
## # A tibble: 96 x 5
##    blank_ballot null_ballot votes_cast `LE PEN` MACRON
##           <dbl>       <dbl>      <dbl>    <dbl>  <dbl>
##  1        28852        8738     285230   111421 173809
##  2        22838        9067     253141   133939 119202
##  3        18877        8522     166786    60207 106579
##  4         9671        3722      83811    34817  48994
##  5         8670        2699      73628    26417  47211
##  6        47784       12414     502951   224544 278407
##  7        19800        7221     167708    63109 104599
##  8        12787        4681     126995    62571  64424
##  9        10133        4518      76057    28074  47983
## 10        13662        4478     139990    64180  75810
## # i 86 more rows
```

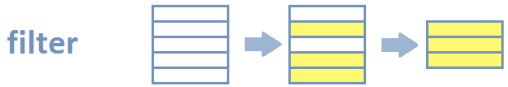
dplyr::select()

Select variables by matching patterns in their names (`starts_with()`, `ends_with()`, `contains()`)

```
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
## # i 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:  in R, missing values are identified by 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_voters absent_voters
##         <dbl> <chr>          <chr>     <chr>                 <dbl>         <dbl>
## 1          93 Provence-Alpe~ 04        Alpes-de~            126459         29255
## 2          93 Provence-Alpe~ 05        Hautes-A~            109892         24895
## 3          93 Provence-Alpe~ 06        Alpes-Ma~            761780        198631
## 4          93 Provence-Alpe~ 13        Bouches-~           1370057        364020
## 5          93 Provence-Alpe~ 83        Var                  794665        202023
## 6          93 Provence-Alpe~ 84        Vaucluse             402307         94108
## # i 6 more variables: present_voters <dbl>, blank_ballot <dbl>,
## #   null_ballot <dbl>, votes_cast <dbl>, `LE PEN` <dbl>, MACRON <dbl>
```

Note: R is case-sensitive.

dplyr::filter()

Filtering by multiple criteria within a single logical expression

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

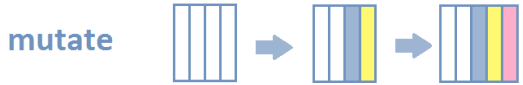
```
## # A tibble: 87 x 12
##    region_code region_name   dept_code dept_name registered_voters absent_voters
##          <dbl> <chr>         <chr>     <chr>                 <dbl>         <dbl>
##  1          84 Auvergne-Rhô~ 01        Ain                  415950         93130
##  2          32 Hauts-de-Fra~ 02        Aisne                375791         90745
##  3          84 Auvergne-Rhô~ 03        Allier               253479         59294
##  4          93 Provence-Alp~ 06        Alpes-Ma~            761780        198631
##  5          84 Auvergne-Rhô~ 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-Alp~ 13        Bouches-~           1370057        364020
## # i 77 more rows
## # i 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_voters absent_voters
##          <dbl> <chr>         <chr>     <chr>                 <dbl>         <dbl>
##  1          76 Occitanie     48        Lozère                59496         11884
##  2          75 Nouvelle-Aqu~ 23        Creuse                93122         21738
##  3          27 Bourgogne-Fr~ 90        Territoi~             95258         23179
##  4          94 Corse         2A        Corse-du~            108760         38512
##  5          93 Provence-Alp~ 05        Hautes-A~            109892         24895
##  6          84 Auvergne-Rhô~ 15        Cantal               117394         25279
##  7          76 Occitanie     09        Ariège               117406         26698
##  8          94 Corse         2B        Haute-Co~            124873         45639
##  9          93 Provence-Alp~ 04        Alpes-de~            126459         29255
## 10          44 Grand Est     52        Haute-Ma~            134119         32081
## # i 86 more rows
## # i 6 more variables: present_voters <dbl>, blank_ballot <dbl>,
## #   null_ballot <dbl>, votes_cast <dbl>, `LE PEN` <dbl>, MACRON <dbl>
```

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: new variables overwrite existing variables of same name.

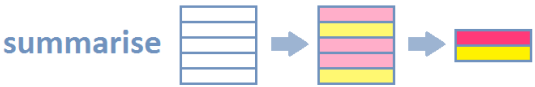
dplyr::mutate()

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_voters
##          <dbl> <chr>           <chr>     <chr>           <dbl>             <dbl>
##  1          84 Auvergne-Rhône~ 01        Ain             0.776            415950
##  2          32 Hauts-de-France 02        Aisne           0.759            375791
##  3          84 Auvergne-Rhône~ 03        Allier          0.766            253479
##  4          93 Provence-Alpes~ 04        Alpes-de~       0.769            126459
##  5          93 Provence-Alpes~ 05        Hautes-A~       0.773            109892
##  6          93 Provence-Alpes~ 06        Alpes-Ma~       0.739            761780
##  7          84 Auvergne-Rhône~ 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
## # i 86 more rows
## # i 7 more variables: absent_voters <dbl>, present_voters <dbl>,
## #   blank_ballot <dbl>, null_ballot <dbl>, votes_cast <dbl>, `LE PEN` <dbl>,
## #   MACRON <dbl>
```

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.

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

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

dplyr::summarise()

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::group_by()

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

```
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 Normandie                    1633534
##  9 Nouvelle-Aquitaine           2929305
## 10 Occitanie                    2793669
## # i 3 more rows
```

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

Load geographic information about the regions

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

dplyr::left_join()

Include geographic information in the tibble about the elections result

```
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-Rhône-Alpes            45.5      4.54 01        Ain
##  2          32 Hauts-de-France                 50.0      2.77 02        Aisne
##  3          84 Auvergne-Rhône-Alpes            45.5      4.54 03        Allier
##  4          93 Provence-Alpes-Côte d'Azur      44.0      6.06 04        Alpes-de~
##  5          93 Provence-Alpes-Côte d'Azur      44.0      6.06 05        Hautes-A~
##  6          93 Provence-Alpes-Côte d'Azur      44.0      6.06 06        Alpes-Ma~
##  7          84 Auvergne-Rhône-Alpes            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
## # i 86 more rows
## # i 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"))
```

dplyr::bind_rows()

```
## # 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-France Round~    2897       8337      1472            17997
##  2          11 Île-de-France Round~    3706       8195      1247            41505
##  3          11 Île-de-France Round~    2872       8148      1358            32906
##  4          11 Île-de-France Round~    2924       7514      1241            44793
##  5          11 Île-de-France Round~    2447       8453      1345            21359
##  6          11 Île-de-France Round~    3235       8739      1088            16601
##  7          11 Île-de-France Round~    2749       7303      1066            26252
##  8          11 Île-de-France Round~    2752       7702       978            24790
##  9          11 Île-de-France Round~      NA         NA        NA               NA
## 10          11 Île-de-France Round~      NA         NA        NA               NA
## # i 182 more rows
## # i 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

Most "real" data is untidy

- Most people are not familiar with the principles of tidy data (it can be hard to derive it unless you spend a lot of time working with data)
- Data is often organized to facilitate some use other than analysis (e.g. to make entry as easy as possible)

Common issues

One variable might be spread across multiple columns (i.e. they are values of a variable)

```
## # A tibble: 3 x 3
##   country     `1999` `2000`
##   <chr>        <dbl>  <dbl>
## 1 Afghanistan    745   2666
## 2 Brazil       37737  80488
## 3 China       212258 213766
```

One observation might be scattered across multiple rows

```
## # A tibble: 6 x 4
##   country      year type           count
##   <chr>       <dbl> <chr>          <dbl>
## 1 Afghanistan  1999 cases            745
## 2 Afghanistan  1999 population  19987071
## 3 Afghanistan  2000 cases           2666
## 4 Afghanistan  2000 population  20595360
## 5 Brazil       1999 cases           37737
## 6 Brazil       1999 population 172006362
```

Useful 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
##    <chr>                <chr>      <dbl>       <dbl> <chr>     <chr>
##  1 Auvergne-Rhône-Alpes LE PEN    111421          84 01        Ain
##  2 Auvergne-Rhône-Alpes MACRON    173809          84 01        Ain
##  3 Auvergne-Rhône-Alpes LE PEN     60207          84 03        Allier
##  4 Auvergne-Rhône-Alpes MACRON    106579          84 03        Allier
##  5 Auvergne-Rhône-Alpes LE PEN     63109          84 07        Ardèche
##  6 Auvergne-Rhône-Alpes MACRON    104599          84 07        Ardèche
##  7 Auvergne-Rhône-Alpes LE PEN     23938          84 15        Cantal
##  8 Auvergne-Rhône-Alpes MACRON     55411          84 15        Cantal
##  9 Auvergne-Rhône-Alpes LE PEN     94312          84 26        Drôme
## 10 Auvergne-Rhône-Alpes MACRON    157992          84 26        Drôme
## # i 182 more rows
## # i 6 more variables: registered_voters <dbl>, 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

Directly on the input data: variables spread over multiple columns

```
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 Normandie                    621472     1012062
##  9 Nouvelle-Aquitaine           918237     2011068
## 10 Occitanie                   1033853     1759816
## # i 3 more rows
```

Why should we use tidyr?

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
## # i 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-Maritimes         MACRON    278407     1 TRUE
##  8 Alpes-Maritimes         LE PEN    224544     2 FALSE
##  9 Alpes-de-Haute-Provence MACRON     48994     1 TRUE
## 10 Alpes-de-Haute-Provence LE PEN     34817     2 FALSE
## # i 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_voters
##    <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
## # i 86 more rows
## # i 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/
```

Example of use for `stringr::str_glue()`

```
round_2 %>% select(region_name, candidate, votes) %>%
  group_by(region_name, candidate) %>%
  summarise(total = sum(votes)) %>%
  mutate(message = str_glue("Candidate {candidate} received {total} votes in region {region_name}"))
```

```
## # A tibble: 26 x 4
## # Groups:   region_name [13]
##    region_name             candidate   total message
##    <chr>                   <chr>       <dbl> <glue>
##  1 Auvergne-Rhône-Alpes    LE PEN    1200726 Candidate LE PEN received 1200726 ~
##  2 Auvergne-Rhône-Alpes    MACRON    2452191 Candidate MACRON received 2452191 ~
##  3 Bourgogne-Franche-Comté LE PEN     532935 Candidate LE PEN received 532935 v~
##  4 Bourgogne-Franche-Comté MACRON     815709 Candidate MACRON received 815709 v~
##  5 Bretagne                LE PEN     425462 Candidate LE PEN received 425462 v~
##  6 Bretagne                MACRON    1301226 Candidate MACRON received 1301226 ~
##  7 Centre-Val de Loire     LE PEN     450750 Candidate LE PEN received 450750 v~
##  8 Centre-Val de Loire     MACRON     777973 Candidate MACRON received 777973 v~
##  9 Corse                   LE PEN      62982 Candidate LE PEN received 62982 vo~
## 10 Corse                   MACRON      66819 Candidate MACRON received 66819 vo~
## # i 16 more rows
```

Cheatsheets

Data manipulation