Univariate Categorical Analysis

2025-02-04

M1 MIDS/MFA/LOGOS Université Paris Cité Année 2024 Course Homepage Moodle



Objectives

In Exploratory analysis of tabular data, univariate analysis is the first step. It consists in exploring, summarizing, visualizing columns of a dataset. In this Lab, we describe univariate categorical analysis

Try to load (potentially) useful packages in a chunk at the beginning of your file.

```
to_be_loaded <- c(</pre>
  "tidyverse",
 "lobstr",
  "ggforce",
  "patchwork",
  "glue",
  "magrittr",
  "gt",
  "DT",
 "lobstr",
  "kableExtra",
  "viridis"
purrr::map_lgl(
 to_be_loaded,
 \(x) require(x,
   character.only = T,
   quietly = T)
Set the (graphical) theme
old_theme <- theme_set(theme_minimal())</pre>
```



In this lab, we load the data from the hard drive. The data are read from some file located in our tree of directories. Loading requires the determination of the correct filepath. This filepath is often a *relative filepath*, it is relative to the directory where the R session/the R script has been launched. Base R offers functions that can help you to find your way the directories tree.

```
getwd() # Where are we?
## [1] "/home/boucheron/Documents/MA7BY020/core/labs-solutions"
head(list.files()) # List the files in the current directory
## [1] "_metadata.yml" "_preamble.qmd" "DATA"
## [4] "lab-bivariate_cache" "lab-bivariate_files" "lab-bivariate.qmd"
head(list.dirs()) # List sub-directories
## [1] "." "./DATA"
## [3] "./lab-bivariate_cache" "./lab-bivariate_cache/html"
## [5] "./lab-bivariate_cache/pdf" "./lab-bivariate_files"
```

Package here for navigating the working tree.

Objectives

In this lab, we introduce univariate analysis for *categorical* variables.

This amounts to exploring, summarizing, visualizing categorical columns of a dataset.

This also often involves table wrangling: retyping some columns, relabelling, reordering, lumping levels of *factors*, that is factor re-engineering.

Summarizing univariate categorical samples amounts to counting the number of occurrences of levels in the sample.

Visualizing categorical samples starts with

- Bar plots
- Column plots

This exploratory work seldom makes it to the final report. Nevertheless, it has to be done in an efficient, reproducible way.

This is an opportunity to revisit the DRY principle.

At the end, we shall see that skimr::skim() can be very helpful.

Dataset Recensement

Have a look at the text file. Choose a loading function for each format. Rstudio IDE provides a valuable helper.

Load the data into the session environment and call it df.

i Question

Create if it does not already exist, a subdirectory DATA in your working directory.

•

The fs package contains a number of useful functions for handling files and directories with a consistent API.

Solution if (!fs::dir_exists('./DATA')) { fs::dir_create('./DATA') }

Question

Download file Recensement from URL https://stephane-v-boucheron.fr/data/Recensement.csv. Base function download.file() is enough.

solution download.file('https://stephane-v-boucheron.fr/data/Recensement.csv', "./DATA/Recensement.csv") df <- readr::read table("./DATA/Recensement.csv") # check that the path is correct Have a glimpse at the dataframe df %>% glimpse() ## Rows: 599 ## Columns: 11 ## \$ AGE <dbl> 58, 40, 29, 59, 51, 19, 64, 23, 47, 66, 26, 23, 54, 44, 56,~ ## \$ SEXE "W", "S", "NE", "W", "NW", "S", "NE", "NW", "S". "NE"~ $\langle chr \rangle$ "NE". ## \$ REGION ## \$ STAT MARI <chr> "C", "M", "C", "D", "M", "C", "M", "C", "M", "D", "M", "C", ~ <dbl> 13.25, 12.50, 14.00, 10.60, 13.00, 7.00, 19.57, 13.00, 20.1~ ## \$ SAL HOR ## \$ SYNDICAT <chr> "non", "non", "non", "oui", "non", "non", "non", "non", "ou~ ## \$ CATEGORIE <dbl> 5, 7, 5, 3, 3, 9, 1, 8, 5, 2, 5, 3, 2, 2, 2, 5, 9, 2, 2,~ ## \$ NIV_ETUDES <dbl> 43, 38, 42, 39, 35, 39, 40, 43, 40, 40, 42, 40, 34, 40, 43,~ ## \$ NB PERS
<dbl> 2, 2, 2, 4, 8, 6, 3, 2, 3, 1, 3, 2, 6, 5, 4, 4, 3, 2, 3, 2,~ <dbl> 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, ~ ## \$ NB_ENF ## \$ REV_FOYER <dbl> 11, 7, 15, 7, 15, 16, 13, 11, 12, 8, 10, 8, 13, 11, 14, 7, ~

Column (re)coding

In order to understand the role of each column, have a look at the following coding tables.

- SEXE
 - F: Female
 - M: Male
- REGION
 - NE: North-East
 - W: West
 - S: South
 - NW: North-West
- STAT_MARI
 - C (Unmarried)
 - M (Married)
 - D (Divorced)
 - S (Separated)

- V (Widowed)
- SYNDICAT:
 - "non": not affiliated with any Labour Union
 - "oui": affiliated with a Labour Union
- CATEGORIE: Professional activity
 - 1: Business, Management and Finance
 - 2: Liberal professions
 - 3: Services
 - 4: Selling
 - 5: Administration
 - 6: Agriculture, Fishing, Forestry
 - 7: Building
 - 8: Repair and maintenance
 - 9: Production
 - 10: Commodities Transportation
- NIV_ETUDES: Education level
 - 32: at most 4 years schooling
 - 33: between 5 and 6 years schooling
 - 34: between 7 and 8 years schooling
 - 35: 9 years schooling
 - 36: 10 years schooling
 - 37: 11 years schooling
 - 38: 12 years schooling, dropping out from High School without a diploma
 - 39: 12 years schooling, with High School diploma
 - 40: College education with no diploma
 - 41: Associate degree, vocational. Earned in two years or more
 - 42: Associate degree, academic. Earned in two years or more
 - 43: Bachelor
 - 44: Master
 - 45: Specific School Diploma
 - 46: PhD
- REV_FOYER : Classes of annual household income in dollars.
- NB_PERS: Number of people in the household.
- NB_ENF: Number of children in the household.

Handling factors

We build lookup tables to incorporate the above information. In R, named vectors are convenient.

```
category_lookup = c(
   "1"= "Business, Management and Finance",
   "2"= "Liberal profession",
   "3"= "Services",
   "4"= "Selling",
   "5"= "Administration",
   "6"= "Agriculture, Fishing, Forestry",
   "7"= "Building ",
   "8"= "Repair and maintenance",
   "9"= "Production",
   "10"= "Commodities Transport"
)
```

In the next chunk, the named vectors are turned into two-columns dataframes (tibbles).

```
vector2tibble <- function(v) {
   tibble(name=v, code= names(v))
}

code_category <- category_lookup %>%
   vector2tibble()

code_category
```

```
# A tibble: 10 x 2
   name
                                        code
   <chr>
                                        <chr>
 1 "Business, Management and Finance"
2 "Liberal profession"
                                        2
3 "Services"
                                        3
4 "Selling"
                                        4
                                        5
5 "Administration"
                                        6
6 "Agriculture, Fishing, Forestry"
                                        7
7 "Building "
8 "Repair and maintenance"
                                        8
9 "Production"
                                        9
10 "Commodities Transport"
                                        10
```

Using the magrittr pipe %>%, the function vector2tibble could have been defined using the concise piping notation. Then . serves as a *pronoun*.

```
vector2tibble <- . %>%
  tibble(name=., code= names(.)) # The lhs of the pipe is used twice.
```

Note the use of . as pronoun for the function argument.

This construction is useful for turning a pipeline into a univariate function.

The function vector2tibble could also be defined by binding identifier vector2tibble with an *anonymous function*.

```
vector2tibble <- \(v) tibble(name=v, code= names(v))</pre>
```

```
education_lookup = c(
   "32"= "<= 4 years schooling",
   "33"= "between 5 and 6 years",
   "34"= "between 7 and 8 years",
   "35"= "9 years schooling",
   "36"= "10 years schooling",
   "37"= "11 years schooling",
   "38"= "12 years schooling, no diploma",
   "39"= "12 years schooling, HS diploma",
   "40"= "College without diploma",
   "41"= "Associate degree, vocational",
   "42"= "Associate degree, academic",
   "43"= "Bachelor",
   "44"= "Master",
   "45"= "Specific School Diploma",</pre>
```

```
"46"= "PhD"
code_education <- vector2tibble(education_lookup)</pre>
status_lookup <- c(</pre>
  "C"="Single",
  "M"="Maried",
  "V"="Widowed",
  "D"="Divorced",
  "S"="Separated"
code_status <- status_lookup %>%
  vector2tibble()
breaks_revenue <-c(</pre>
  0,
  5000,
  7500,
  10000,
  12500,
  15000,
  17500,
  20000,
  25000,
  30000,
  35000,
  40000,
  50000,
  60000,
  75000,
  100000,
  150000
```

Table wrangling

i Question

Which columns should be considered as categorical/factor?

• Deciding which variables are categorical sometimes requires judgement.

Let us attempt to base the decision on a checkable criterion: determine the number of distinct values in each column, consider those columns with less than 20 distinct values as factors.

We can find the names of the columns with few unique values by iterating over the column names.

solution

We already designed a pipeline to determine which columns should be transformed into a factor (categorized). In the next chunk, we turn the pipeline into a univariate function named to_be_categorized with one argument (the dataframe)

```
to be categorized <- . %>%
  summarise(across(everything(), n_distinct)) %>%
 pivot_longer(cols = everything(), values_to = c("n_levels")) %>%
 filter(n_levels < 20) %>%
  arrange(n_levels) %>%
 pull(name)
```

to_be_categorized can be used like a function.

```
to_be_categorized
## Functional sequence with the following components:
##
   1. summarise(., across(everything(), n_distinct))
##
## 2. pivot_longer(., cols = everything(), values_to = c("n_levels"))
## 3. filter(., n_levels < 20)
## 4. arrange(., n_levels)
## 5. pull(., name)
## Use 'functions' to extract the individual functions.
tbc <- to_be_categorized(df)
tbc
[1] "SEXE"
                 "SYNDICAT"
                              "REGION"
                                            "STAT_MARI"
                                                         "NB ENF"
[6] "NB_PERS"
                 "CATEGORIE"
                              "NIV_ETUDES" "REV_FOYER"
```

Note that columns NB_PERS and NB_ENF have few unique values and nevertheless we could consider them as quantitative.

Question

Coerce the relevant columns as factors.

Use dplyr and forcats verbs to perform this coercion.

Use the across() construct so as to perform a kind if tidy selection (as with select) with verb mutate.

You may use forcats::as_factor() to transform columns when needed.

Verb dplyr::mutate is a convenient way to modify a dataframe.

solution

We can repeat the categorization step used in the preceding lab.

```
mutate(across(all_of(tbc), as_factor)) %>%
 glimpse()
Rows: 599
Columns: 11
$ AGE
            <dbl> 58, 40, 29, 59, 51, 19, 64, 23, 47, 66, 26, 23, 54, 44, 56,~
$ SEXE
            <fct> F, M, M, M, M, M, F, F, M, F, M, F, F, F, F, F, F, M, M, F,~
$ REGION
            <fct> NE, W, S, NE, W, NW, S, NE, NW, S, NE, NE, W, NW, S, S, NW,~
            <fct> C, M, C, D, M, C, M, C, M, D, M, C, M, C, M, C, S, M, S, C,~
$ STAT_MARI
$ SAL_HOR
            <dbl> 13.25, 12.50, 14.00, 10.60, 13.00, 7.00, 19.57, 13.00, 20.1~
$ SYNDICAT
            <fct> non, non, non, oui, non, non, non, oui, non, non, ~
$ CATEGORIE <fct> 5, 7, 5, 3, 3, 3, 9, 1, 8, 5, 2, 5, 3, 2, 2, 2, 5, 9, 2, 2,~
$ NIV_ETUDES <fct> 43, 38, 42, 39, 35, 39, 40, 43, 40, 40, 42, 40, 34, 40, 43,~
            <fct> 2, 2, 2, 4, 8, 6, 3, 2, 3, 1, 3, 2, 6, 5, 4, 4, 3, 2, 3, 2,~
$ NB PERS
$ NB ENF
            <fct> 11, 7, 15, 7, 15, 16, 13, 11, 12, 8, 10, 8, 13, 11, 14, 7, ~
$ REV FOYER
★ What is the meaning of all_of(tbc)?
The pronoun mechanism that comes with the pipe %>% offers a alternative:
```

```
df <- df %>%
  mutate(across(all_of(to_be_categorized(.)), as_factor))

df %>%
  glimpse()
```

```
Rows: 599
Columns: 11
```

```
$ AGE
            <dbl> 58, 40, 29, 59, 51, 19, 64, 23, 47, 66, 26, 23, 54, 44, 56,~
$ SEXE
            <fct> F, M, M, M, M, M, F, F, M, F, M, F, F, F, F, F, F, M, M, F,~
            <fct> NE, W, S, NE, W, NW, S, NE, NW, S, NE, NE, W, NW, S, S, NW,~
$ REGION
            <fct> C, M, C, D, M, C, M, C, M, D, M, C, M, C, M, C, S, M, S, C,~
$ STAT MARI
$ SAL_HOR
            <dbl> 13.25, 12.50, 14.00, 10.60, 13.00, 7.00, 19.57, 13.00, 20.1~
            <fct> non, non, non, oui, non, non, non, oui, non, non, ~
$ SYNDICAT
$ CATEGORIE <fct> 5, 7, 5, 3, 3, 3, 9, 1, 8, 5, 2, 5, 3, 2, 2, 2, 5, 9, 2, 2,~
$ NIV_ETUDES <fct> 43, 38, 42, 39, 35, 39, 40, 43, 40, 40, 42, 40, 34, 40, 43,~
$ NB PERS
            <fct> 2, 2, 2, 4, 8, 6, 3, 2, 3, 1, 3, 2, 6, 5, 4, 4, 3, 2, 3, 2,~
$ NB ENF
            $ REV_FOYER <fct> 11, 7, 15, 7, 15, 16, 13, 11, 12, 8, 10, 8, 13, 11, 14, 7, ~
The dot . in all_of(to_be_categorized(.)) refers to the left-hand side of %>%.
```

solution

Evaluation of pull(to_be_categorized, name) returns a character vector containing the names of the columns to be categorized. all_of() enables mutate to perform as_factor() on each of these columns and to bind the column names to the transformed columns.

i Question

Relabel the levels of REV_FOYER using the breaks.

solution

We first built readable labels for REV_FOYER. As each level of REV_FOYER corresponds to an interval, we use intervals as labels.

```
income_slices <- levels(df$REV_FOYER)

l <- length(breaks_revenue)

names(income_slices) <- paste(
    "[",
    breaks_revenue[-1],
    "-",
    lead(breaks_revenue)[-1],
    ")",
    sep=""
)</pre>
```

₩ What does lead() do?

```
df <- df %>%
  mutate(REV_FOYER=forcats::fct_recode(REV_FOYER, !!!income_slices))

df %>%
  relocate(REV_FOYER) %>%
  head()
```

```
# A tibble: 6 x 11
```

```
REV_FOYER
                AGE SEXE REGION STAT_MARI SAL_HOR SYNDICAT CATEGORIE NIV_ETUDES
              <dbl> <fct> <fct> <fct>
                                               <dbl> <fct>
                                                              <fct>
                                                                         <fct>
  <fct>
1 [35000-400~
                                                                         43
                 58 F
                           NE
                                  С
                                                13.2 non
                                                              5
                                                              7
2 [17500-200~
                 40 M
                           W
                                  Μ
                                                12.5 non
                                                                         38
3 [75000-1e+~
                 29 M
                                  C
                                                14
                                                     non
                                                              5
                                                                         42
4 [17500-200~
                                  D
                                                              3
                                                                         39
                 59 M
                           NE
                                                10.6 oui
5 [75000-1e+~
                 51 M
                           W
                                  Μ
                                                              3
                                                                         35
                                                13
                                                     non
6 [1e+05-150~
                 19 M
                                  C
                                                7
                                                              3
                                                                         39
                          NW
                                                     non
```

i 2 more variables: NB_PERS <fct>, NB_ENF <fct>

Note the use of !!! (bang-bang-bang) to unpack the named vector income_slices. The bang-bang-bang device is offered by rlang, a package from tidyverse. It provides a very handy way of calling functions like fct_recode that take an unbounded list of key-values pairs as argument. This is very much like handling keyword arguments in Python using dictionary unpacking.

Relabel the levels of the different factors so as to make the data more readbale

```
solution
df %>%
  select(where(is.factor)) %>%
  head()
# A tibble: 6 x 9
  SEXE REGION STAT MARI SYNDICAT CATEGORIE NIV ETUDES NB PERS NB ENF REV FOYER
                                                            <fct>
  <fct> <fct> <fct>
                                                <fct>
                                                                     <fct>
                           <fct>
                                     <fct>
                                                                            <fct>
1 F
        NE
                С
                                                43
                                                            2
                                                                     0
                                                                             [35000-40~
                           non
                                     5
                                     7
                                                                     0
2 M
        W
                М
                                                38
                                                            2
                                                                             [17500-20~
                           non
        S
                С
                                     5
                                                42
                                                            2
                                                                     0
                                                                             [75000-1e~
3 M
                           non
                                     3
                                                            4
4 M
        NE
                D
                           oui
                                                39
                                                                     1
                                                                             [17500-20~
5 M
                Μ
                                     3
                                                35
                                                            8
                                                                     1
                                                                             [75000-1e~
                           non
                                     3
                                                            6
                                                                     0
                                                                             [1e+05-15~
6 M
        NW
                С
                           non
                                                39
The columns that call for relabelling the levels are:
   • CATEGORIE
   • NIV_ETUDES
```

```
solution
lookup_category <- code_category$code</pre>
names(lookup_category) <- code_category$name</pre>
lookup_niv_etudes <- code_education$code</pre>
names(lookup_niv_etudes) <- code_education$name</pre>
df <- df %>%
  mutate(CATEGORIE=forcats::fct_recode(CATEGORIE, !!!lookup_category)) %>%
  mutate(NIV ETUDES=forcats::fct recode(NIV ETUDES, !!!lookup_niv_etudes))
df %>%
 head()
# A tibble: 6 x 11
    AGE SEXE REGION STAT_MARI SAL_HOR SYNDICAT CATEGORIE
                                                                  NIV_ETUDES NB_PERS
  <dbl> <fct> <fct> <fct>
                                   <dbl> <fct>
                                                   <fct>
                                                                  <fct>
                                                                              <fct>
     58 F
              NE
                      С
                                    13.2 non
1
                                                   "Administrat~ Bachelor
                                                                              2
                                                                  12 years ~
2
     40 M
                      Μ
                                    12.5 non
                                                   "Building "
                      C
3
     29 M
                                    14
                                         non
                                                   "Administrat~ Associate~
                                                   "Services"
4
                      D
     59 M
              NE
                                    10.6 oui
                                                                  12 years ~
5
     51 M
               W
                      М
                                    13
                                         non
                                                   "Services"
                                                                  9 years s~ 8
                      C
                                                   "Services"
6
     19 M
               NW
                                     7
                                         non
                                                                  12 years ~ 6
# i 2 more variables: NB_ENF <fct>, REV_FOYER <fct>
t We should be able to DRY this.
# TODO
```

Search for missing data (optional)

i Question

Check whether some columns contain missing data (use is.na).

• Useful functions:

- dplyr::summarise
- across
- tidyr::pivot_longer
- dplyr::arrange

i solution

```
df %>%
  is.na() %>%
  as_tibble %>%
  summarise(across(everything(), sum)) %>%
  kable()
```

AGE	SEXE	REGIOS	TAT_M	AARI_H	NR NDICO	ATTEGON	UME_ETU		RB_EN	EV_FO	YER
0	0	0	0	0	0	0	0	0	0	0	

Summarizing categorical data

Counting

i Question

Use table, prop.table from base R to compute the frequencies and proportions of the different levels. In statistics, the result of table() is a (one-way) contingency table.

```
solution
df |>
 pull(REV_FOYER) |>
 table()
      [0-5000)
                   [5000-7500)
                                  [7500-10000)
                                                 [10000-12500)
                                                                 [12500-15000)
                              5
                                              5
                                                                              7
 [15000-17500)
                 [17500-20000)
                                 [20000-25000)
                                                 [25000-30000)
                                                                  [30000-35000)
                             26
                                             38
                                                             30
                                                                             35
             19
 [35000-40000)
                 [40000-50000)
                                 [50000-60000)
                                                 [60000-75000)
                                                                  [75000-1e+05)
                             70
                                                                             77
                                             71
                                                             89
[1e+05-150000)
df |>
  count(REV_FOYER)
# A tibble: 16 x 2
   REV_FOYER
                       n
   <fct>
                   <int>
 1 [0-5000)
                       9
 2 [5000-7500)
                       5
                       5
 3 [7500-10000)
 4 [10000-12500)
                       9
 5 [12500-15000)
                       7
 6 [15000-17500)
                      19
 7 [17500-20000)
                      26
 8 [20000-25000)
                      38
 9 [25000-30000)
                      30
10 [30000-35000)
                      35
11 [35000-40000)
                      61
12 [40000-50000)
                      70
13 [50000-60000)
                      71
14 [60000-75000)
                      89
15 [75000-1e+05)
                      77
16 [1e+05-150000)
                      48
```

What is the *class* of the *object* generated by table? Is it a vector, a list, a matrix, an array?

```
solution
ta <- df %>%
 pull(REV_FOYER) %>%
  table()
1 <- list(
  is.vector=is.vector,
  is.list=is.list,
  is.matrix=is.matrix,
  is.array=is.array
map_lgl(1, \sim .x(ta))
is.vector
            is.list is.matrix is.array
              FALSE
                         FALSE
                                    TRUE
    FALSE
```

```
as.data.frame() (or as_tibble) can transform a table object into a dataframe.
  ta <- rename(as.data.frame(ta), REV_FOYER=`.`)</pre>
  ta
           REV_FOYER Freq
  1
            [0-5000)
  2
         [5000-7500)
                         5
        [7500-10000)
   3
                         5
   4
       [10000-12500)
                         9
                         7
  5
       [12500-15000)
  6
       [15000-17500)
                        19
  7
       [17500-20000)
                        26
  8
       [20000-25000)
                        38
  9
       [25000-30000)
                        30
   10 [30000-35000)
                        35
   11 [35000-40000)
                        61
  12 [40000-50000)
                        70
  13 [50000-60000)
                        71
   14 [60000-75000)
                        89
   15
      [75000-1e+05)
                        77
   16 [1e+05-150000)
                        48
```

You may use knitr::kabble(), possibly knitr::kable(., format="markdown") to tweak the output.

If you are more ambitious, use gt::....

In order to feed ggplot with a contingency table, it is useful to build contingency tables as dataframes. Use dplyr::count() to do this.

```
skimr::skim() allows us to perform univariate categorical analysis all at once.
df %>%
  skimr::skim(where(is.factor)) %>%
  print(n=50)
-- Data Summary -----
                           Values
Name
                           Piped data
Number of rows
                           599
Number of columns
                           11
Column type frequency:
  factor
Group variables
                           None
-- Variable type: factor ------
  skim_variable n_missing complete_rate ordered n_unique
1 SEXE
                                      1 FALSE
2 REGION
                        0
                                      1 FALSE
3 STAT_MARI
                        0
                                                      5
                                      1 FALSE
                                      1 FALSE
4 SYNDICAT
                        0
                                                       2
5 CATEGORIE
                        0
                                      1 FALSE
                                                      10
6 NIV ETUDES
                        0
                                      1 FALSE
                                                      15
7 NB_PERS
                        0
                                      1 FALSE
8 NB ENF
                        0
                                                       7
                                      1 FALSE
9 REV_FOYER
                        0
                                      1 FALSE
                                                      16
  top_counts
1 M: 302, F: 297
2 S: 200, W: 148, NE: 129, NW: 122
3 M: 325, C: 193, D: 61, S: 14
4 non: 496, oui: 103
5 Lib: 133, Ser: 125, Adm: 94, Sel: 48
6 12 : 187, Col: 148, Bac: 114, Ass: 45
7 2: 196, 4: 130, 3: 122, 1: 63
8 0: 413, 1: 86, 2: 76, 3: 18
9 [60: 89, [75: 77, [50: 71, [40: 70
The output can be tailored to your specific objectives and fed to functions that are
geared to displaying large tables (see packages knitr, DT, and gt)
```

Save the (polished) data

Saving polished data in self-documented formats can be time-saving. Base $\tt R$ offers the $\tt .RDS$ format

```
df %>%
  saveRDS("./DATA/Recensement.RDS")
```

By saving into this format we can persist our work.

```
dt <- readRDS("./DATA/Recensement.RDS")
dt %>%
```

```
glimpse()
```

Compare the size of csv and RDS files.

Plotting

Plot the counts, first for column SEXE

We shall use barplots to visualize counts.

barplot belongs to the bar graphs family.

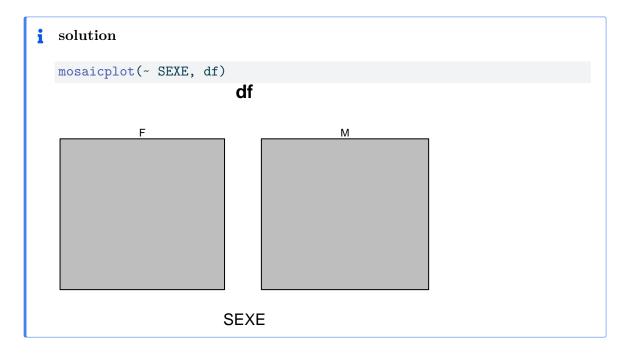
Build a barplot to visualize the distribution of the SEXE column.

Use

- geom_bar (working directly with the data)
- geom_col (working with a contingency table)

```
solution
  df %>%
    ggplot() +
    aes(x=SEXE) +
    geom_bar() +
    ggtitle("With geom_bar")) + (
  df %>%
    count(SEXE) %>%
    ggplot() +
    aes(x=SEXE, y=n) +
    geom_col() +
    ggtitle("With geom_col")
  )
      With geom_bar
                                          With geom_col
  300
                                      300
  200
                                      200
count
  100
                                      100
    0
                                        0
             F
                                                 F
                         M
                                                             M
                 SEXE
                                                     SEXE
```

When investigating relations between categorical columns we will often rely on mosaicplot(). Indeed, barplot and mosaicplot belong to the collection of area plots that are used to visualize counts (statistical summaries for categorical variables).



Repeat the same operation for each qualitative variable (DRY)

Using a for loop

We have to build a barplot for each categorical variable. Here, we just have nine of them. We could do this using cut and paste, and some editing. In doing so, we would not comply with the DRY (Don't Repeat Yourself) principle.

In order to remain DRY, we will attempt to abstract the recipe we used to build our first barplot.

This recipe is pretty simple:

- 1. Build a ggplot object with df as the data layer.
- 2. Add an aesthetic mapping a categorical column to axis x
- 3. Add a geometry using geom_bar
- 4. Add labels explaining the reader which column is under scrutiny

We first need to gather the names of the categorical columns. The following chunk does this in a simple way.

```
i solution

col_names <- df %>%
    select(where(is.factor))%>%
    names()
```

In the next chunk, we shall build a named list of ggplot objects consisting of barplots. The for loop body is almost obtained by cutting and pasting the recipe for the first barplot.



Note an important difference: instead of something aes(x=col) where col denotes a column in the dataframe, we shall write aes(x=.data[[col]]) where col is a string that matches a column name. Writing aes(x=col) would not work.

The loop variable col iterates over the column names, not over the columns themselves.

When using ggplot in interactive computations, we write aes(x=col), and, under the hood, the interpreter uses the *tidy evaluation* mechanism that underpins R to map df\$col to the x axis.

ggplot functions like aes() use $data\ masking$ to alleviate the burden of the working Statistician.

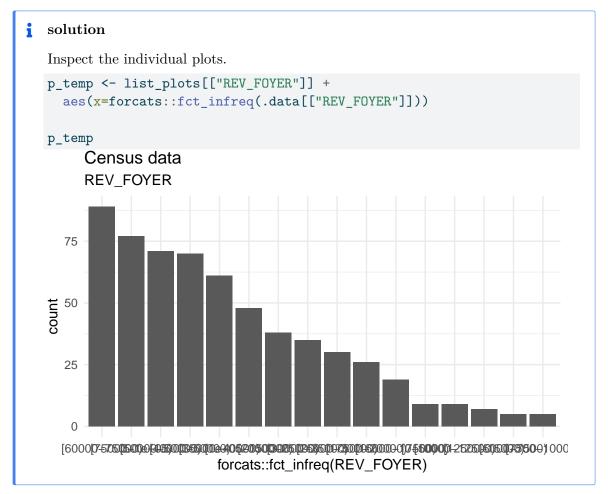
Within the context of ggplot programming, pronoun .data refers to the data layer of the graphical object.

solution

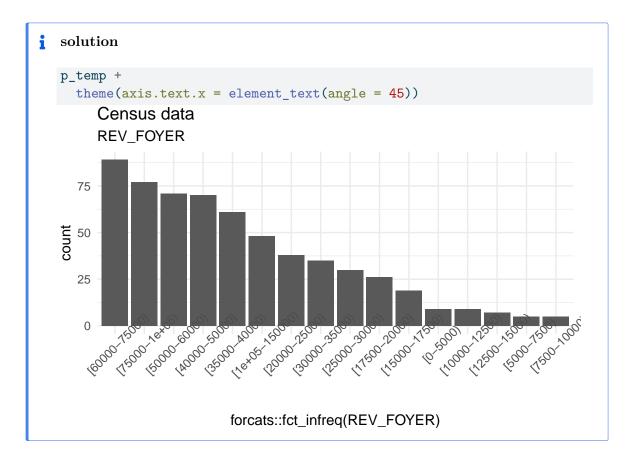
```
list_plots <- list()

for (col in col_names){
   p <- df %>%
        ggplot() +
        aes(x=.data[[col]]) +  # mind the .data pronoun
        geom_bar() +
        labs(
        title="Census data",
        subtitle = col
   )

   list_plots[[col]] <- p # add the ggplot object to the list
}</pre>
```



If the labels on the x-axis are not readable, we need to tweak them. This amounts to modifying the theme layer in the ggplot object, and more specifically the axis.text.x attribute.



Using functional programming (lapply, purrr::...)

Another way to compute the list of graphical objects replaces the for loop by calling a functional programming tool. This mechanism relies on the fact that in R, functions are first-class objects.

•

Package purrr offers a large range of tools with a clean API. Base R offers lapply().

We shall first define a function that takes as arguments a datafame, a column name, and a title. We do not perform any defensive programming. Call your function foo.

```
i solution

foo <- function(df, col, .title= "WE NEED A TITLE!!!"){
   p <- df %>%
        ggplot() +
        aes(x=fct_infreq(.data[[col]])) +
        geom_bar() +
        labs(
        title=.title,
        subtitle = col
   ) +
        theme(axis.text.x = element_text(angle = 45))
        return(p)
}
```

Functional programmming makes code easier to understand.

Use foo, lapply or purrr::map() to build the list of graphical objects.

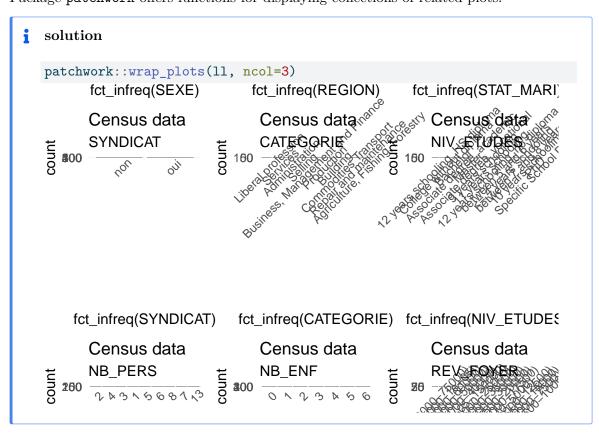
With purrr::map(), you may use either a formula or an anonymous function. With lapply use an anonymous function.

Package patchwork offers functions for displaying collections of related plots.

 $ll[[.x]] \leftarrow foo(df, .x, "Census data")$

for (.x in col_names){

}



Useful links

- dplyr
- ggplot2
- R Graphic Cookbook. Winston Chang. O' Reilly.
- A blog on ggplot objects
- skimr
- rmarkdown
- quarto