- **M1 MIDS & MFA**
- **Université Paris Cité**
- Année 2023-2024
- Course Homepage
- Moodle

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

```
to_be_loaded <- c("tidyverse",
                  "lobstr",
                  "ggforce",
                  "patchwork",
                  "glue",
                  "magrittr",
                  "DT",
                  "lobstr",
                  "kableExtra",
                  "viridis")

for (pck in to_be_loaded) {
  if (!require(pck, character.only = T)) {
    install.packages(pck, repos="http://cran.rstudio.com/")
    stopifnot(require(pck, character.only = T))
  }
}
```

Set the (graphical) theme

```
old_theme <- theme_set(theme_minimal())
```

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/COURS/EDA_LABS"
head(list.files())  # List the files in the current directory
## [1] "_extensions"        "_handout"           "_handout_fr"
## [4] "_handout_solution"  "_metadata.yml"      "_quarto-french.yml"
head(list.dirs())   # List sub-directories
## [1] "."
## [2] "./_extensions"
## [3] "./_extensions/quarto-ext"
## [4] "./_extensions/quarto-ext/fontawesome"
## [5] "./_extensions/quarto-ext/fontawesome/assets"
## [6] "./_extensions/quarto-ext/fontawesome/assets/css"
```

Objectives

In this lab, we pursue our walk in univariate analysis, by introducing 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 introduce the DRY principle.

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

Dataset Recensement (Census, bis)

> Since 1948, the US Census Bureau carries out a monthly Current Population Survey, collecting data concerning residents aged above 15 from 150000 households. This survey is one of the most important sources of information concerning the american workforce. Data reported in file `Recensement.txt` originate from the 2012 census.

Dataset `Recensement` can be found in file `Recensement.csv` in your `DATA` repository.

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`.

```
list.dirs(recursive = F)
##  [1] "./_extensions"        "./_handout"           "./_handout_fr"
##  [4] "./_handout_solution" "./.git"               "./.quarto"
##  [7] "./.Rproj.user"        "./DATA"               "./IMG"
## [10] "./UTILS"
list.files('./DATA/')
##  [1] "Banque.csv"
##  [2] "OECD-DP_LIVE time series.csv"
##  [3] "OECD.CFE.EDS,DSD_REG_DEMO@DF_LIFE_EXP,1.0+all.csv"
##  [4] "OECD.SDD.NAD,DSD_NAMAIN1@DF_QNA_EXPENDITURE_CAPITA,1.0+all.csv"
##  [5] "OECD.SDD.NAD,DSD_NAMAIN10@DF_TABLE1_EXPENDITURE_CPC,1.0+all.csv"
##  [6] "Recensement.csv"
##  [7] "Recensement.RDS"
##  [8] "REGION_DEMOGR_12012024150444803.csv"
##  [9] "semmelweis.csv"
## [10] "titanic"
```

solution

```
df <- readr::read_table("./DATA/Recensement.csv")    # check that the path is corre
```

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        <chr> "F", "M", "M", "M", "M", "M", "F", "F", "M", "F", "M", "F",~
## $ REGION      <chr> "NE", "W", "S", "NE", "W", "NW", "S", "NE", "NW", "S", "NE"~
## $ STAT_MARI   <chr> "C", "M", "C", "D", "M", "C", "M", "C", "M", "D", "M", "C",~
## $ SAL_HOR     <dbl> 13.25, 12.50, 14.00, 10.60, 13.00, 7.00, 19.57, 13.00, 20.1~
## $ SYNDICAT    <chr> "non", "non", "non", "oui", "non", "non", "non", "non", "ou~
## $ CATEGORIE   <dbl> 5, 7, 5, 3, 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,~
## $ NB_ENF      <dbl> 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,~
## $ 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.

```
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"
)

# code_category <- as_tibble() %>% rownames_to_column() %>% rename(code = rowname, name=
```

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" 1
 2 "Liberal profession"                2
 3 "Services"                          3
 4 "Selling"                           4
 5 "Administration"                    5
 6 "Agriculture, Fishing, Forestry"    6
 7 "Building "                         7
 8 "Repair and maintenance"            8
 9 "Production"                        9
10 "Commodities Transport"             10
```

The function `vector2tibble` could be defined using the concise piping notation. `.` serves as a pronoun.

```
vector2tibble <- . %>%
  tibble(name=., code= names(.))
```

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))
```

```
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",
  "46"= "PhD"
)

code_education <- vector2tibble(education_lookup)
```

```r
status_lookup <- c(
  "C"="Single",
  "M"="Maried",
  "V"="Widowed",
  "D"="Divorced",
  "S"="Separated"
)

code_status <- status_lookup %>%
  vector2tibble()
```

```r
breaks_revenue <-c(
  0,
  5000,
  7500,
  10000,
  12500,
  15000,
  17500,
  20000,
  25000,
  30000,
  35000,
  40000,
  50000,
  60000,
  75000,
  100000,
  150000
)
```

Table wrangling

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.

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.

```
df %>%
  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,~
$ STAT_MARI   <fct> C, M, C, D, M, C, M, C, M, D, M, C, M, C, M, C, S, M, S, C,~
$ 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, non, oui, non, 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,~
$ 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      <fct> 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,~
$ REV_FOYER   <fct> 11, 7, 15, 7, 15, 16, 13, 11, 12, 8, 10, 8, 13, 11, 14, 7, ~
```

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,~
$ REGION      <fct> NE, W, S, NE, W, NW, S, NE, NW, S, NE, NE, W, NW, S, S, NW,~
$ STAT_MARI   <fct> C, M, C, D, M, C, M, C, M, D, M, C, M, C, M, C, S, M, S, C,~
$ 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, non, oui, non, 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,~
$ 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      <fct> 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,~
$ 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.

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[-l],
  "-",
  lead(breaks_revenue)[-l],
  sep=""
)
```

```
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
  <fct>       <dbl> <fct> <fct>  <fct>       <dbl> <fct>    <fct>     <fct>
1 35000-40000    58 F     NE     C            13.2 non      5         43
2 17500-20000    40 M     W      M            12.5 non      7         38
3 75000-1e+05    29 M     S      C            14   non      5         42
4 17500-20000    59 M     NE     D            10.6 oui      3         39
5 75000-1e+05    51 M     W      M            13   non      3         35
6 1e+05-1500~    19 M     NW     C             7   non      3         39
# 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     C         non      5         43         2       0      35000-400~
2 M     W      M         non      7         38         2       0      17500-200~
3 M     S      C         non      5         42         2       0      75000-1e+~
4 M     NE     D         oui      3         39         4       1      17500-200~
5 M     W      M         non      3         35         8       1      75000-1e+~
6 M     NW     C         non      3         39         6       0      1e+05-150~
```

The columns that call for relabelling the levels are:

- CATEGORIE
- NIV_ETUDES

solution

```
lookup_category <- code_category$code
names(lookup_category) <- code_category$name

lookup_niv_etudes <- code_education$code
names(lookup_niv_etudes) <- code_education$name

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>
1    58 F     NE     C            13.2 non      "Administrat~ Bachelor   2
2    40 M     W      M            12.5 non      "Building "   12 years ~ 2
3    29 M     S      C            14   non      "Administrat~ Associate~ 2
4    59 M     NE     D            10.6 oui      "Services"    12 years ~ 4
5    51 M     W      M            13   non      "Services"    9 years s~ 8
6    19 M     NW     C             7   non      "Services"    12 years ~ 6
# i 2 more variables: NB_ENF <fct>, REV_FOYER <fct>
```

We should be able to DRY this.

```
# TODO
```

Search for missing data (optional)

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

::: {.callout-tip} Useful functions:

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

solution

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

AGE	SEXE	REGION	STAT_MARI	SAL_HOR	SYNDICAT	CATEGORIE	NIV_ETUI
0	0	0	0	0	0	0	

SEXE

Counting

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 %>%
  count(SEXE)
```

```
# A tibble: 2 x 2
  SEXE      n
  <fct> <int>
1 F       297
2 M       302
```

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

solution

```
ta <- df %>%
  pull(SEXE) %>%
  table()

l <- list(is.vector=is.vector, is.list=is.list, is.matrix=is.matrix, is.array=is.ar

map_lgl(l, ~ .x(ta))
```

```
is.vector   is.list is.matrix  is.array
    FALSE     FALSE     FALSE      TRUE
```

`as.data.frame()` (or `as_tibble`) can transform a `table` object into a dataframe.

```
ta <-  rename(as.data.frame(ta), SEXE=`.`)

ta
```

```
  SEXE Freq
1    F  297
2    M  302
```

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

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)) %>%
  skimr::to_long() %>%
  print(n=50)
```

```
# A tibble: 45 x 4
   skim_type skim_variable stat              formatted
   <chr>     <chr>         <chr>             <chr>
 1 factor    SEXE          n_missing         0
 2 factor    REGION        n_missing         0
 3 factor    STAT_MARI     n_missing         0
 4 factor    SYNDICAT      n_missing         0
 5 factor    CATEGORIE     n_missing         0
 6 factor    NIV_ETUDES    n_missing         0
 7 factor    NB_PERS       n_missing         0
 8 factor    NB_ENF        n_missing         0
 9 factor    REV_FOYER     n_missing         0
10 factor    SEXE          complete_rate     1
11 factor    REGION        complete_rate     1
12 factor    STAT_MARI     complete_rate     1
13 factor    SYNDICAT      complete_rate     1
14 factor    CATEGORIE     complete_rate     1
15 factor    NIV_ETUDES    complete_rate     1
16 factor    NB_PERS       complete_rate     1
17 factor    NB_ENF        complete_rate     1
18 factor    REV_FOYER     complete_rate     1
19 factor    SEXE          factor.ordered    FALSE
20 factor    REGION        factor.ordered    FALSE
21 factor    STAT_MARI     factor.ordered    FALSE
22 factor    SYNDICAT      factor.ordered    FALSE
23 factor    CATEGORIE     factor.ordered    FALSE
24 factor    NIV_ETUDES    factor.ordered    FALSE
25 factor    NB_PERS       factor.ordered    FALSE
26 factor    NB_ENF        factor.ordered    FALSE
27 factor    REV_FOYER     factor.ordered    FALSE
28 factor    SEXE          factor.n_unique   2
29 factor    REGION        factor.n_unique   4
30 factor    STAT_MARI     factor.n_unique   5
31 factor    SYNDICAT      factor.n_unique   2
32 factor    CATEGORIE     factor.n_unique   10
33 factor    NIV_ETUDES    factor.n_unique   15
34 factor    NB_PERS       factor.n_unique   9
35 factor    NB_ENF        factor.n_unique   7
36 factor    REV_FOYER     factor.n_unique   16
37 factor    SEXE          factor.top_counts M: 302, F: 297
38 factor    REGION        factor.top_counts S: 200, W: 148, NE: 129, NW: 122
39 factor    STAT_MARI     factor.top_counts M: 325, C: 193, D: 61, S: 14
40 factor    SYNDICAT      factor.top_counts non: 496, oui: 103
41 factor    CATEGORIE     factor.top_counts Lib: 133, Ser: 125, Adm: 94, Sel: ~
42 factor    NIV_ETUDES    factor.top_counts 12 : 187, Col: 148, Bac: 114, Ass:~
43 factor    NB_PERS       factor.top_counts 2: 196, 4: 130, 3: 122, 1: 63
```

Save the (polished) data

Saving polished data in self-documented formats can be time-saving. Base `R` offers the `.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.

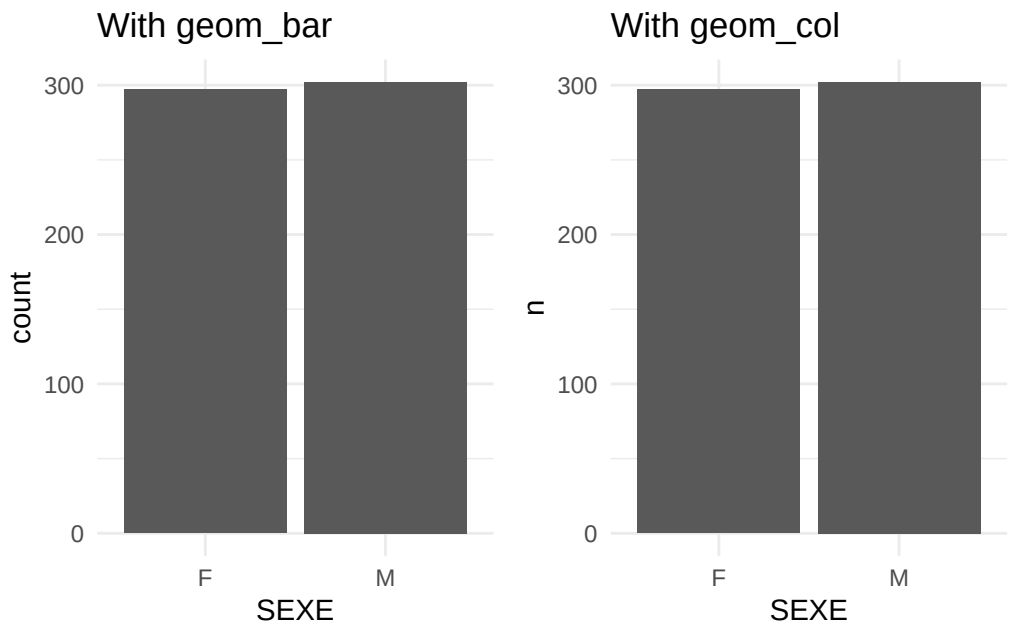
Build a barplot to visualize the distribution of the `SEXE` column.

Use

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

solution

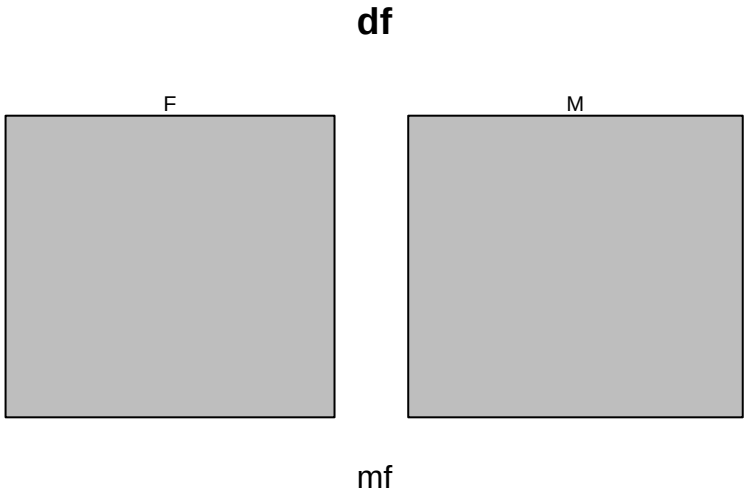
```
(
  df %>%
    ggplot() +
    aes(x=SEXE) +
    geom_bar() +
    ggtitle("With geom_bar")) + (
  df %>%
    count(SEXE) %>%
    ggplot() +
    aes(x=SEXE, y=n) +
    geom_col() +
    ggtitle("With geom_col")
  )
```



When investigating relations between categerical 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).

solution

```
mosaicplot(~ SEXE, df)
```



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.

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.

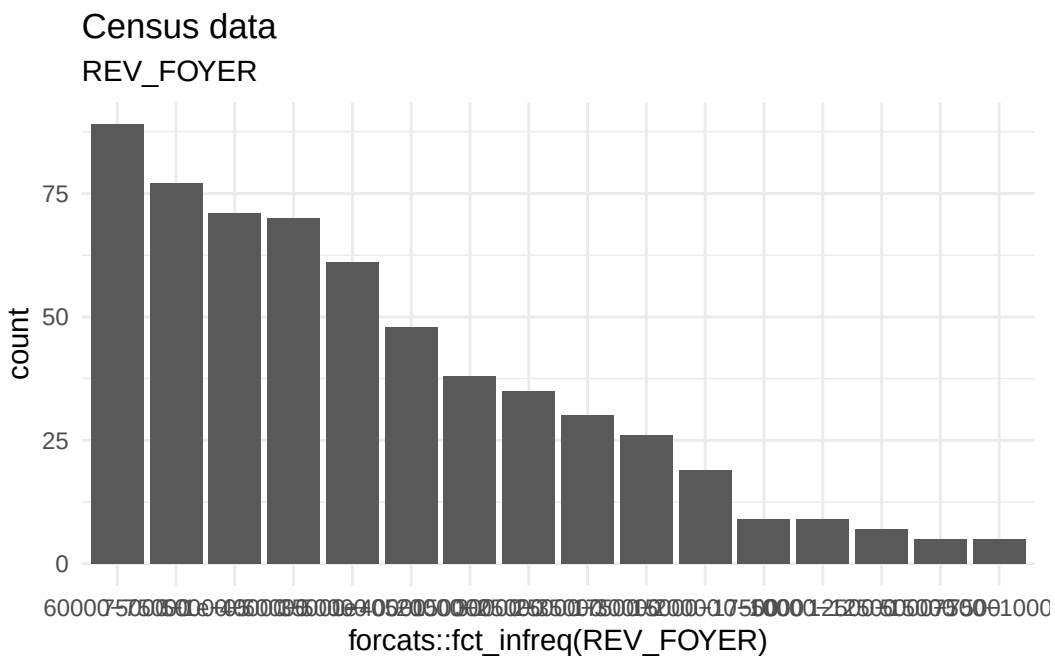
solution

```
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
}
```

solution

Inspect the individual plots.

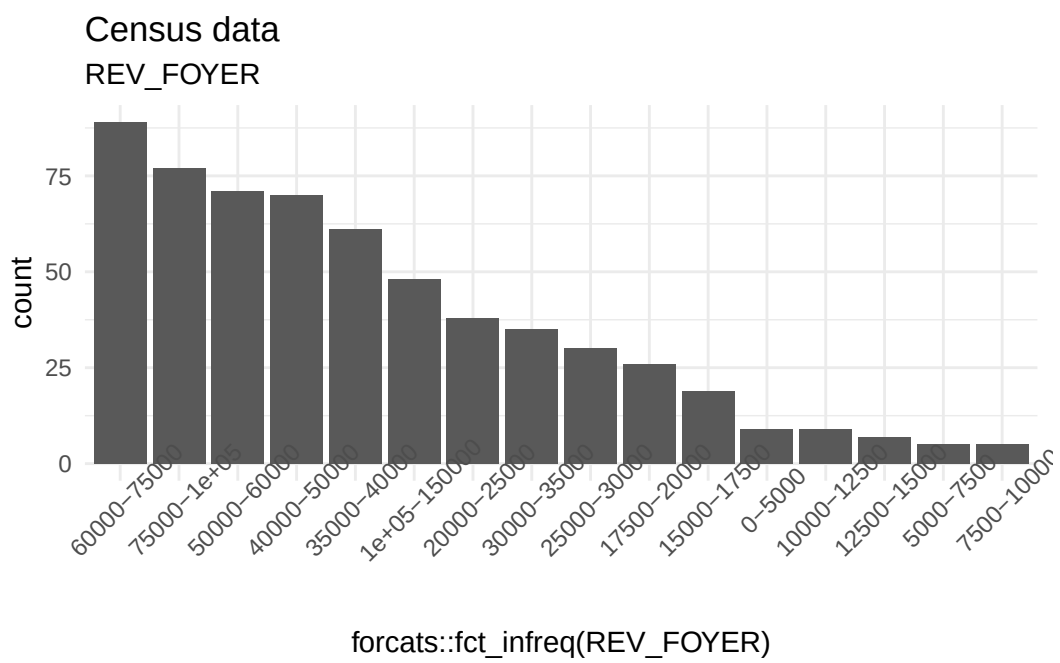
```
p_temp <- list_plots[["REV_FOYER"]] +
  aes(x=forcats::fct_infreq(.data[["REV_FOYER"]]))

p_temp
```



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.

solution

```
p_temp +
  theme(axis.text.x = element_text(angle = 45))
```



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`.

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.

solution

```
ll <- map(col_names, ~ foo(df, .x, "Census data"))
```

solution

```
map(col_names, \(x) foo(df, x, "Census data"))
```

solution

```
lapply(col_names, \(x)  foo(df, x, "Census data"))
```

solution

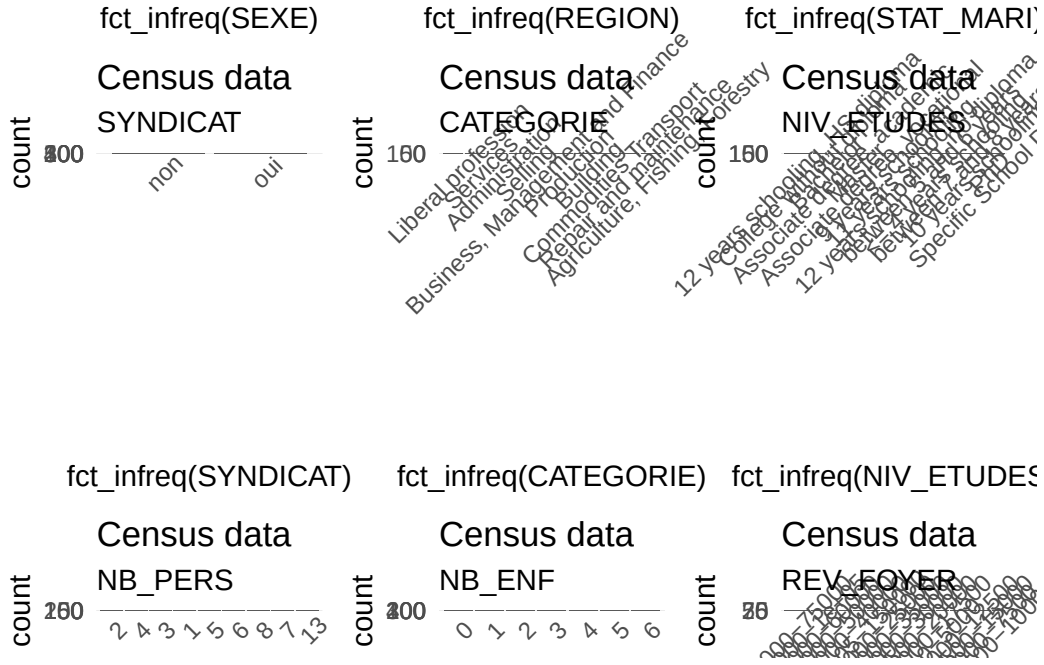
This is essentially like executing

```
ll <- list()

for (.x in col_names){
  ll[[.x]] <- foo(df, .x, "Census data")
}
```

Package `patchwork` offers functions for displaying collections of related plots.

solution

```
patchwork::wrap_plots(ll, ncol=3)
```



Useful links

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