Univariate analysis II

2024-09-05

- **M1 MIDS & MFA**
- **Université Paris Cité**
- Année 2024-2025
- Course Homepage

- Moodle

> **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, re-ordering, lumping levels of factors, that is factor re-engineering.

Setup

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

```
stopifnot(
  require(lobstr),
  require(rlang),
  require(ggforce),
  require(patchwork),
  require(glue),
  require(magrittr),
  require(DT),
  require(gt),
  require(kableExtra),
  require(viridis),
  require(vcd),
  require(skimr),
  require(tidyverse)

)
```

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/MA7BY020/labs-solutions"
head(list.files())  # List the files in the current directory
## [1] "lab-babynames_cache" "lab-babynames_files" "lab-babynames.html"
## [4] "lab-babynames.pdf"   "lab-babynames.qmd"   "lab-bivariate_cache"
head(list.dirs())   # List sub-directories
## [1] "."                                  "./lab-babynames_cache"
## [3] "./lab-babynames_cache/html"         "./lab-babynames_cache/pdf"
## [5] "./lab-babynames_files"              "./lab-babynames_files/figure-html"
```

Use package `fs` for files maniulations

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)

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] "./lab-babynames_cache"             "./lab-babynames_files"
##  [3] "./lab-bivariate_cache"             "./lab-bivariate_files"
##  [5] "./lab-corr-babynames_cache"        "./lab-corr-babynames_files"
##  [7] "./lab-exercices-glm_cache"         "./lab-gapminder_cache"
##  [9] "./lab-gapminder_files"             "./lab-gapminder-plotly_cache"
## [11] "./lab-gapminder-plotly_files"      "./lab-gss-r_cache"
## [13] "./lab-gss-r_files"                 "./lab-histo-density_cache"
## [15] "./lab-histo-density_files"         "./lab-kmeans_cache"
## [17] "./lab-kmeans_files"                "./lab-life-tables_cache"
## [19] "./lab-lifeexp_cache"               "./lab-lin-reg_cache"
## [21] "./lab-lin-reg_files"               "./lab-lt-miashs_cache"
## [23] "./lab-lt-miashs_files"             "./lab-nycflights-pq_cache"
## [25] "./lab-pca_cache"                   "./lab-pca_files"
## [27] "./lab-progr_cache"                 "./lab-report-census_cache"
```

```
## [29] "./lab-template_cache"                "./lab-test-miashs_cache"
## [31] "./lab-test-miashs_files"             "./lab-univariate-categorical_cache"
## [33] "./lab-univariate-categorical_files" "./lab-univariate-numeric_cache"
## [35] "./lab-univariate-numeric_files"      "./lab-vctrs_cache"
## [37] "./lab-vectorization_cache"
list.files('./DATA/')
## character(0)
```

solution

```
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        <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=va
```

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)
```

```
status_lookup <- c(
  "C"="Single",
```

```
  "M"="Maried",
  "V"="Widowed",
  "D"="Divorced",
  "S"="Separated"
)

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

```
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 [35000-400~    58 F     NE     C            13.2 non      5         43
2 [17500-200~    40 M     W      M            12.5 non      7         38
3 [75000-1e+~    29 M     S      C            14   non      5         42
4 [17500-200~    59 M     NE     D            10.6 oui      3         39
5 [75000-1e+~    51 M     W      M            13   non      3         35
6 [1e+05-150~    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      [35000-40~
2 M     W      M         non      7         38         2       0      [17500-20~
3 M     S      C         non      5         42         2       0      [75000-1e~
4 M     NE     D         oui      3         39         4       1      [17500-20~
5 M     W      M         non      3         35         8       1      [75000-1e~
6 M     NW     C         non      3         39         6       0      [1e+05-15~
```

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>
```

< fa-hand-point-right > We should be able to DRY this.

```
# TODO
```

Search for missing data (optional)

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