Univariate Categorical Analysis

2025-01-28

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.

```r
to_be_loaded <- c(
  "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)
  )
##  [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

Set the (graphical) theme

```r
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/core/labs"
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`.

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.

Question

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

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"
)

# 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
```

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))
```

```
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

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.

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.

Question

Relabel the levels of `REV_FOYER` using the breaks.

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

Search for missing data (optional)

Question

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

Useful functions:

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

Summarizing categorical data

Counting

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*.

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

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

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

ta
```

```
        REV_FOYER Freq
1         [0-5000)    9
2      [5000-7500)    5
3     [7500-10000)    5
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
```

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.

```r
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                   9
________________________
Group variables            None

-- Variable type: factor -------------------------------------------------------
  skim_variable n_missing complete_rate ordered n_unique
1 SEXE                  0             1 FALSE          2
2 REGION                0             1 FALSE          4
3 STAT_MARI             0             1 FALSE          5
4 SYNDICAT              0             1 FALSE          2
5 CATEGORIE             0             1 FALSE         10
6 NIV_ETUDES            0             1 FALSE         15
7 NB_PERS               0             1 FALSE          9
8 NB_ENF                0             1 FALSE          7
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 `R` offers the `.RDS` format

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

By saving into this format we can persist our work.

```r
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)

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).

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.

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.

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`.

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.

Useful links

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