LAB: Univariate analysis

2026-01-30

M1 MIDS/MFA/LOGOS
Université Paris Cité
Année 2025
Course Homepage
Moodle

Univariate numerical samples

```
to_be_loaded <- c("tidyverse",
                  "magrittr",
                  "skimr",
                  "lobstr"
)

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

Objectives

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

In common circumstances, table wrangling is a prerequisite.

Then, univariate techniques depend on the kind of columns we are facing.

For *numerical* samples/columns, to name a few:

- Boxplots
- Histograms
- Density plots
- CDF
- Quantile functions
- Miscellanea

For categorical samples/columns, we have:

- Bar plots

- Column plots

Dataset

> 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.

In this lab, we investigate the numerical colums of the dataset.

After downloading, dataset `Recensement` can be found in file `Recensement.csv`.

Choose a loading function for the format. `Rstudio` IDE provides a valuable helper.

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

solution

```
if (!fs::dir_exists('./DATA')) {
    warning(glue::glue("Creating directory ./DATA in {getwd()}"))
    fs::dir_create('./DATA')
  }

if (!fs::file_exists("./DATA/Recensement.csv")) {
  warning("Downloading Recensement.csv from  https://stephane-v-boucheron.fr/data/Rece
  download.file('https://stephane-v-boucheron.fr/data/Recensement.csv',
                "./DATA/Recensement.csv")
  warning("Downloaded!!")
}

df <- readr::read_table("./DATA/Recensement.csv")
##
## -- Column specification --------------------------------------------------------
## cols(
##   AGE = col_double(),
##   SEXE = col_character(),
##   REGION = col_character(),
##   STAT_MARI = col_character(),
##   SAL_HOR = col_double(),
##   SYNDICAT = col_character(),
##   CATEGORIE = col_double(),
##   NIV_ETUDES = col_double(),
##   NB_PERS = col_double(),
##   NB_ENF = col_double(),
##   REV_FOYER = col_double()
## )

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, ~
```

solution cont'd

A byproduct of `readr::read_table` default behavior is the display of the (guessed) table schema. If we want to store this tentative table schema, we can use `readr::spec_table()`.

```
df_schm <- readr::spec_table("./DATA/Recensement.csv")
```

Have a look at object `df_schm`. Is it a list? What are the list component.
The next time we will load `Recensement`, we want `SEXE` and `NIV_ETUDES` to be handled as factors. We can tweak the guessed schema and use it to reload the dataframe using the optional `col_types` argument.

```
df_schm$cols$SEXE <- readr::col_factor(levels=c("F", "M"))
df_schm$cols$NIV_ETUDES <- readr::col_factor(levels= df|> pull(NIV_ETUDES) |> unique()

df <- readr::read_table("./DATA/Recensement.csv",
                        col_types = df_schm)
```

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.

solution

```
to_be_categorized  <- df |>
  summarise(across(everything(), n_distinct)) |>
  pivot_longer(cols = everything(),
               # names_to = "nom_colonne",
               values_to = c("n_levels")) |>
  filter(n_levels < 20) |>
  arrange(n_levels)

to_be_categorized |>
  gt::gt()
```

name	n_levels
SEXE	2
SYNDICAT	2
REGION	4
STAT_MARI	5
NB_ENF	7
NB_PERS	9
CATEGORIE	10
NIV_ETUDES	15
REV_FOYER	16

```
to_be_categorized |>
  pull(name)
```

```
[1] "SEXE"       "SYNDICAT"   "REGION"     "STAT_MARI"  "NB_ENF"
[6] "NB_PERS"    "CATEGORIE"  "NIV_ETUDES" "REV_FOYER"
```

Columns `NB_PERS` and `NB_ENF` have few unique values and nevertheless we could consider them as quantitative.

Coerce the relevant columns as factors.

solution

We could proceed by iteration over the relevant columns. We use `lobstr::...` to monitor the copy on modify process.

```
lobstr::obj_addr(df)
```

```
[1] "0x561d8b01ff98"
```

```
lobstr::ref(df)
```

```
  [1:0x561d8b01ff98] <spc_tbl_[,11]>
 AGE = [2:0x561d8b006cc0] <dbl>
 SEXE = [3:0x561d8b008c40] <chr>
 REGION = [4:0x561d8b00abc0] <chr>
 STAT_MARI = [5:0x561d8b00cb40] <chr>
 SAL_HOR = [6:0x561d8b00eac0] <dbl>
 SYNDICAT = [7:0x561d8b010a40] <chr>
 CATEGORIE = [8:0x561d8b0129c0] <dbl>
 NIV_ETUDES = [9:0x561d8b014940] <dbl>
 NB_PERS = [10:0x561d8b0168c0] <dbl>
 NB_ENF = [11:0x561d8b018840] <dbl>
 REV_FOYER = [12:0x561d8b01a7c0] <dbl>
```

```
df_copy <- df

lobstr::ref(df_copy)
```

```
  [1:0x561d8b01ff98] <spc_tbl_[,11]>
 AGE = [2:0x561d8b006cc0] <dbl>
 SEXE = [3:0x561d8b008c40] <chr>
 REGION = [4:0x561d8b00abc0] <chr>
 STAT_MARI = [5:0x561d8b00cb40] <chr>
 SAL_HOR = [6:0x561d8b00eac0] <dbl>
 SYNDICAT = [7:0x561d8b010a40] <chr>
 CATEGORIE = [8:0x561d8b0129c0] <dbl>
 NIV_ETUDES = [9:0x561d8b014940] <dbl>
 NB_PERS = [10:0x561d8b0168c0] <dbl>
 NB_ENF = [11:0x561d8b018840] <dbl>
 REV_FOYER = [12:0x561d8b01a7c0] <dbl>
```

```
for (cl in pull(to_be_categorized,name)) {
  df_copy[[cl]] <- as_factor(df_copy[[cl]])
}

lobstr::obj_addr(df_copy)
```

```
[1] "0x561d8b5fdeb8"
```

```
lobstr::ref(df_copy)
```

```
  [1:0x561d8b5fdeb8] <spc_tbl_[,11]>
 AGE = [2:0x561d8b006cc0] <dbl>
 SEXE = [3:0x561d88504290] <fct>
 REGION = [4:0x561d8855c0e0] <fct>
 STAT_MARI = [5:0x561d88588940] <fct>
 SAL_HOR = [6:0x561d8b00eac0] <dbl>
 SYNDICAT = [7:0x561d885357d8] <fct>
```

solution (cont'd)

We will kill several birds with one stone.
`across()` allows us to pick the columns to be categorized, to apply `as_factor()` to each of them, and to replace the old column by the result of `as_factor(....)`.
`across()` allows us to perform `tidy_selection` while using verb `mutate`.

```
df_cp <- df |>
  mutate(across(all_of(pull(to_be_categorized, name)), as_factor))

lobstr::ref(df_cp)
##   [1:0x561d8ab82308] <tibble[,11]>
##  AGE = [2:0x561d8b006cc0] <dbl>
##  SEXE = [3:0x561d898c1408] <fct>
##  REGION = [4:0x561d89828900] <fct>
##  STAT_MARI = [5:0x561d89814ea0] <fct>
##  SAL_HOR = [6:0x561d8b00eac0] <dbl>
##  SYNDICAT = [7:0x561d89862180] <fct>
##  CATEGORIE = [8:0x561d84ee9780] <fct>
##  NIV_ETUDES = [9:0x561d84e3d100] <fct>
##  NB_PERS = [10:0x561d84ed2490] <fct>
##  NB_ENF = [11:0x561d84ebefe0] <fct>
##  REV_FOYER = [12:0x561d8a898900] <fct>

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, ~
```

We could do this using the WET approach

```
df_wet <- df |>
  mutate(SEXE = as_factor(SEXE),
         SYNDICAT = as_factor(SYNDICAT),
         REGION = as_factor(REGION),
         STAT_MARI = as_factor(STAT_MARI),
         NB_ENF = as_factor(NB_ENF),
         NB_PERS = as_factor(NB_PERS),
         CATEGORIE = as_factor(CATEGORIE),
         NIV_ETUDES = as_factor(NIV_ETUDES),
         REV_FOYER = as_factor(REV_FOYER)
  )

df_wet |>
  glimpse()
```

Compare `pull` and `pluck`

Search for missing data (optional)

Question

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

Useful functions:

- `dplyr::summarise_all`
- `tidyr::pivot_longer`
- `dplyr::arrange`

solution

```
df |>
  is.na() |>
  as_tibble() |>
  summarise(across(everything(), sum))  |>
  gt::gt()
```

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

or

```
df |>
  summarise(across(everything(), \(x) sum(is.na(x))))
```

```
# A tibble: 1 x 11
    AGE  SEXE REGION STAT_MARI SAL_HOR SYNDICAT CATEGORIE NIV_ETUDES NB_PERS
  <int> <int>  <int>     <int>   <int>    <int>     <int>      <int>   <int>
1     0     0      0         0       0        0         0          0       0
# i 2 more variables: NB_ENF <int>, REV_FOYER <int>
```

or

```
df |>
  summarise(across(everything(), ~ sum(is.na(.))))
```

```
# A tibble: 1 x 11
    AGE  SEXE REGION STAT_MARI SAL_HOR SYNDICAT CATEGORIE NIV_ETUDES NB_PERS
  <int> <int>  <int>     <int>   <int>    <int>     <int>      <int>   <int>
1     0     0      0         0       0        0         0          0       0
# i 2 more variables: NB_ENF <int>, REV_FOYER <int>
```

Note the different ways of introduction anonymous functions.

Analysis of column `AGE`

Numerical summary

solution

```
df |>
    pull(AGE) |>
    summary()
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  16.00   29.00   42.00   41.85   53.50   80.00
```

```
sd(df$AGE) ; IQR(df$AGE) ; mad(df$AGE)
```

```
[1] 14.11648
[1] 24.5
[1] 17.7912
```

Use `skimr::skim()`

solution

```
df |>
    pull(AGE) |>
    skimr::skim()
```

Table 4: Data summary

Name	pull(df, AGE)
Number of rows	599
Number of columns	1
_____	
Column type frequency:	
numeric	1
_____	
Group variables	None

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
data	0	1	41.85	14.12	16	29	42	53.5	80	

```
skm <- df |>
  skimr::skim(AGE)

class(skm)
```

```
[1] "skim_df"    "tbl_df"     "tbl"        "data.frame"
```

```
attributes(skm)
```

```
$class
[1] "skim_df"    "tbl_df"     "tbl"        "data.frame"

$row.names
[1] 1

$names
 [1] "skim_type"      "skim_variable" "n_missing"     "complete_rate"
 [5] "numeric.mean"   "numeric.sd"    "numeric.p0"    "numeric.p25"
```

Question

Compare `mean` and `median`, `sd` and `IQR`.
Are mean and median systematically related?

solution

Ask the bot.
There is at least one relation between median and mean for square-integrable distributoins:

$$|\text{Median} - \text{Mean}| \leq \text{sd}$$

Lévy's inequality.

Question

Are standard deviation and IQR systematically related ?

solution

Ask the bot.
Yes.

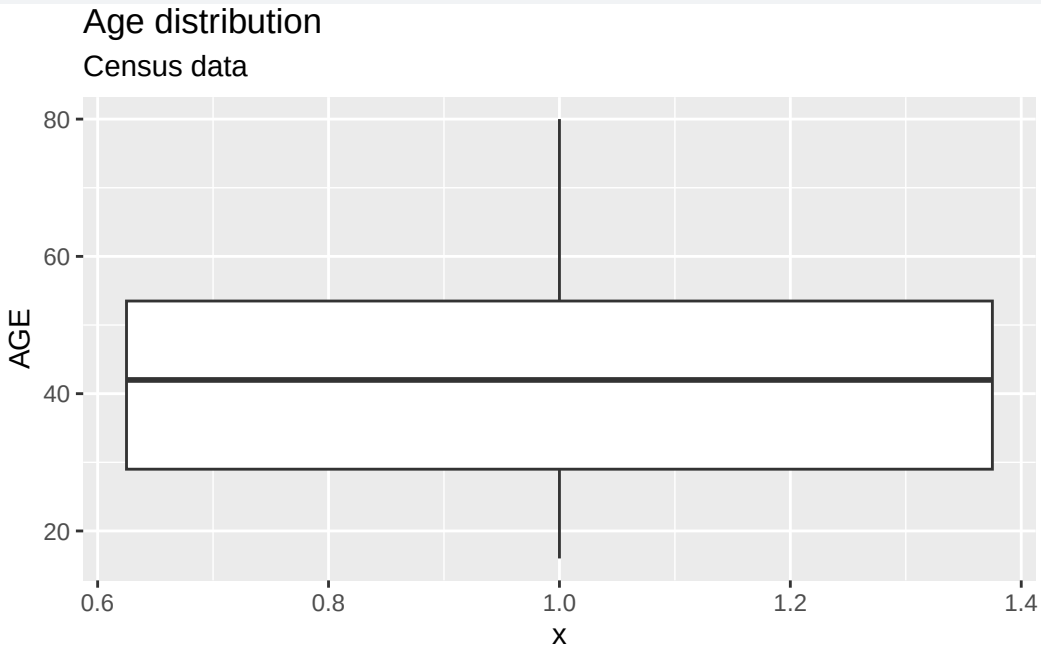
Boxplots

Question

Draw a boxplot of the Age distribution

solution

```
p <- df |>
    ggplot() +
    aes(x=1L, y=AGE) +
    geom_boxplot() +
    labs(
        title="Age distribution",
        subtitle = "Census data"
    )

p
```



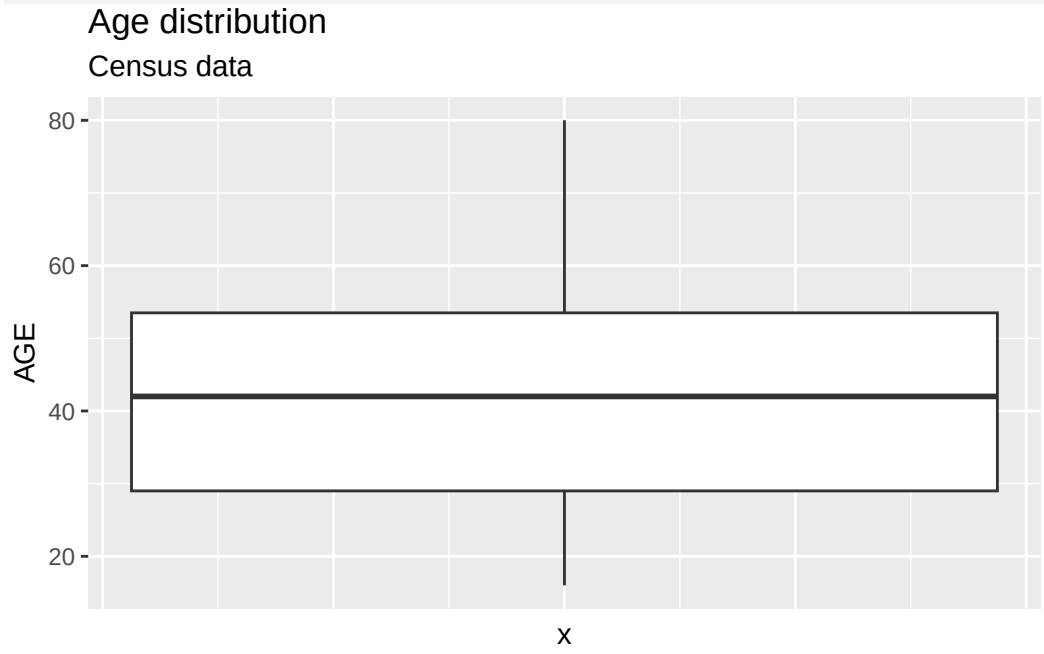
Question

How would you get rid of the useless ticks on the x-axis?

solution

Ask the bot (*In a boxplot built using ggplot2, how can I get rid from the ticks on the x axis?*).
Yes. This is a matter of `theme`.

```
p +
  theme(
    axis.ticks.x = element_blank(),
    axis.text.x = element_blank()
  )
```



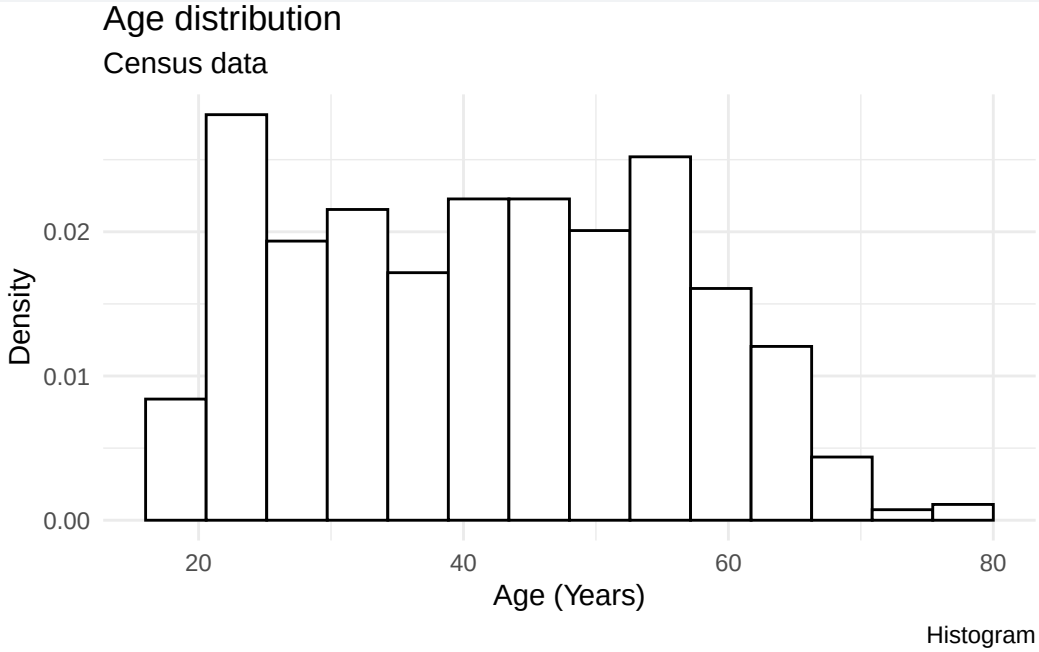
Histograms

Question

Plot a *histogram* of the empirical distribution of the `AGE` column

solution

```
p <- df |>
  ggplot() +
  aes(x=AGE) +
  labs(
    title = "Age distribution",
    subtitle = "Census data",
    x = "Age (Years)",
    y = "Density"
  ) +
  theme_minimal()

p +
  geom_histogram(aes(y=after_stat(density)),
                 bins=15,
                 fill="white",
                 color="black") +
  labs(
    caption = "Histogram"
  )
```



Question

Try different values for the `bins` parameter of `geom_histogram()`

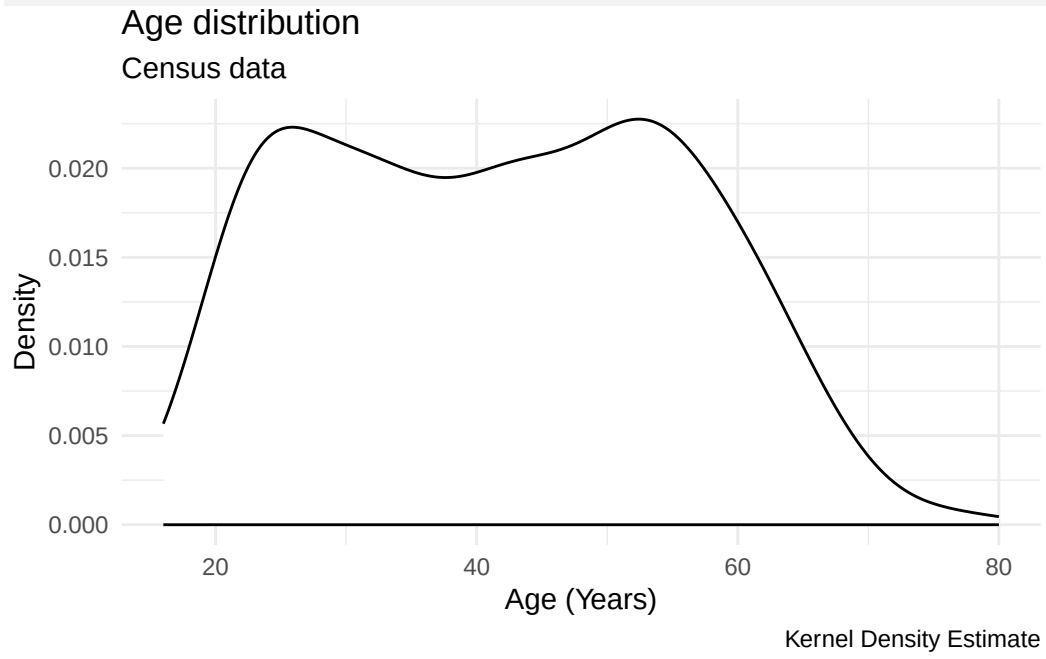
Density estimates

Question

Plot a *density* estimate of the `AGE` column (use `stat_density`.

solution

```
p +
  stat_density(
              fill="white",
              color="black") +
  labs(
    caption = "Kernel Density Estimate"
  )
```



Question

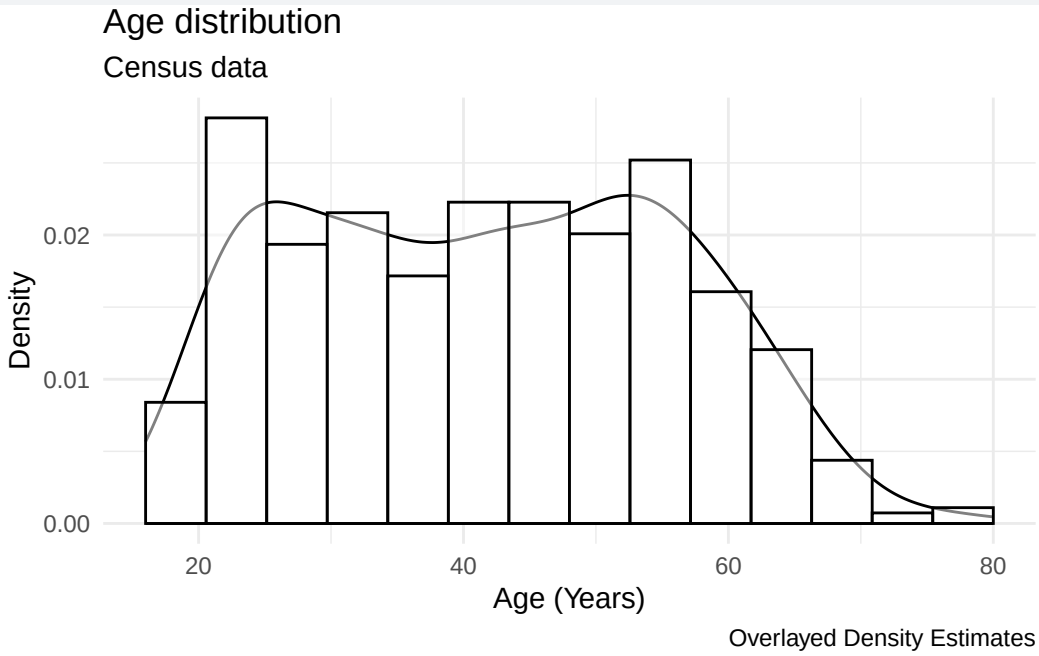
Play with parameters `bw`, `kernel` and `adjust`.

Question

Overlay the two plots (histogram and density).

solution

```
p +
  stat_density(
              fill="white",
              color="black") +
  geom_histogram(aes(y=after_stat(density)),
                 bins=15,
                 fill="white",
                 color="black",
                 alpha=.5) +
  labs(
    caption = "Overlayed Density Estimates"
  )
```



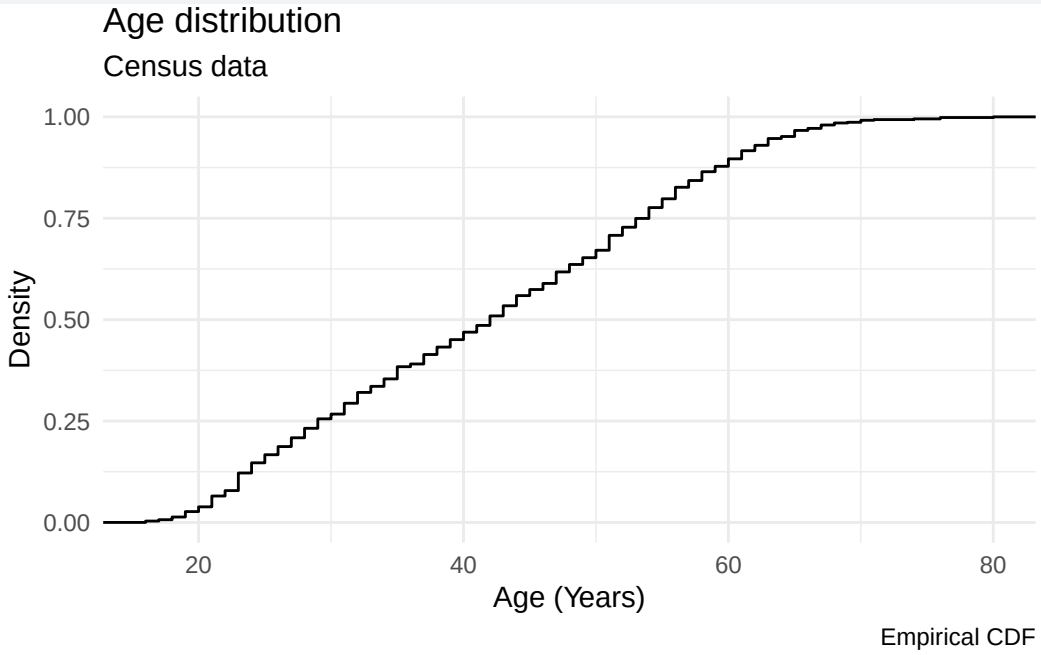
ECDF

Question

Plot the Empirical CDF of the AGE distribution

solution

```
p +
    stat_ecdf()  +
    labs(
        caption = "Empirical CDF"
  )
```



Question

Can you read the quartiles from the ECDF pplot?

solution

Of course. Yes, we can.

Quantile function

Question

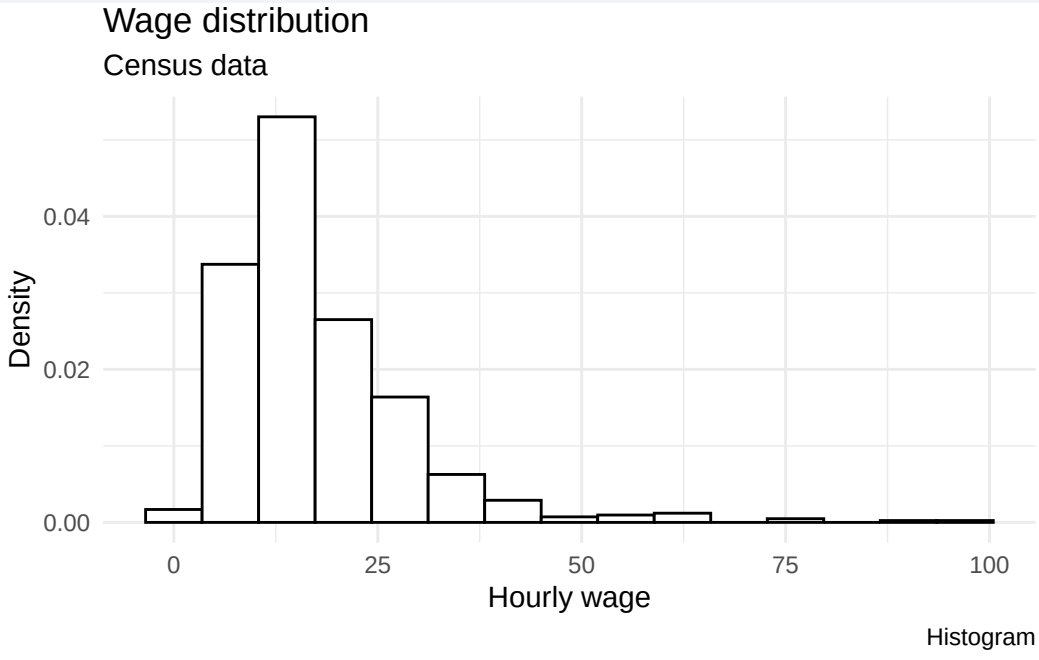
Plot the quantile function of the AGE distribution.

Repeat the analysis for `SAL_HOR`

solution

```
p <- df |>
  ggplot() +
  aes(x=SAL_HOR) +
  labs(
    title = "Wage distribution",
    subtitle = "Census data",
    x = "Hourly wage",
    y = "Density"
  ) +
  theme_minimal()
```

```
p +
  geom_histogram(aes(y=after_stat(density)),
                 bins=15,
                 fill="white",
                 color="black") +
  labs(
    caption = "Histogram"
  )
```



solution

```
p +
  stat_density(
              fill="white",
              color="black") +
  geom_histogram(aes(y=after_stat(density)),
                 bins=15,
                 fill="white",
                 color="black",
                 alpha=.5) +
  labs(
    caption = "Overlayed Density Estimates"
  )
```


```
truc <- rlang::expr({fill=alpha("white",.5)})

p <- df |>
  ggplot() +
  aes(x=SAL_HOR, y=after_stat(density)) +
  labs(
    title = "Wage distribution", 23
    subtitle = "Census data",
```

Question

How could you comply with the DRY principle ?

solution

This amounts to programming with `ggplot2` function. This is not straightforward since `ggplot2` relies on data masking.

> A major requirement of a good data analysis is flexibility. If your data changes, or you discover something that makes you rethink your basic assumptions, you need to be able to easily change many plots at once. The main inhibitor of flexibility is code duplication. If you have the same plotting statement repeated over and over again, you'll have to make the same change in many different places. Often just the thought of making all those changes is exhausting! This chapter will help you overcome that problem by showing you how to program with ggplot2.
>
> To make your code more flexible, you need to reduce duplicated code by writing functions. When you notice you're doing the same thing over and over again, think about how you might generalise it and turn it into a function. If you're not that familiar with how functions work in R, you might want to brush up your knowledge at https://adv-r.hadley.nz/functions.html.

From Hadley Wickham

solution

An attempt:

```
getwd()
```

```
[1] "/home/boucheron/Documents/MA7BY020/core/labs-solutions"
```

```
fs::dir_exists('UTILS')
```

```
UTILS
 TRUE
```

```
pct_format <- scales::percent_format(accuracy = .1)

make_biotifoul <-  function(df, .f=is.factor, .bins=30){

  .scales <- "free"

  if (identical(.f, is.factor)) {
    .scales <- "free_x"
  }

  p <- df |>
    select(where(.f)) |>
    pivot_longer(
      cols = everything(),
      names_to = "var",
      values_to = "val"
    ) |>
    ggplot() +
    aes(x = val) +
    facet_wrap(~var, scales=.scales) +
    xlab("")

  if(identical(.f, is.factor)){
    p +
    geom_bar(fill=alpha("grey", .5)) +
```

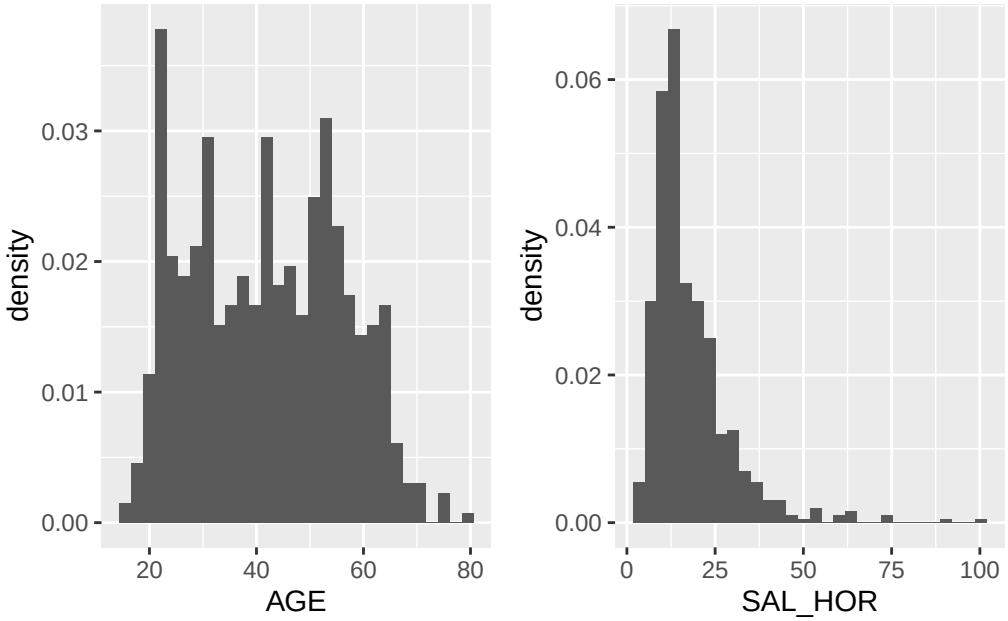
solution

Another attempt. This delivers a list of histogram plots, one for each numeric column.

```
my_histo <- function(df, x_var, ...){

df |>
    ggplot() +
    aes(x= {{x_var}}) +
    geom_histogram(aes(y=after_stat(density)), ...)

}
```

Have a look at the call to `my_histo()`. How is the column passed to the aesthetic mapping? How is the ellipsis `...` used?

```
list_plots <- df_cp |>
  select(where(is.numeric)) |>
  colnames() |>
  map(rlang::parse_expr) |>
  map (\(x) my_histo(df, {{x}}, bins=30))

patchwork::wrap_plots(list_plots)
```



Useful links

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