Data analysis with R

From loading the data to generating results

Carrasco, D., PhD

Centro de Medición MIDE UC

CIES 2023: an LCSE SIG workshop

Washington DC, February 18th, 2023

Workshop

Data analysis as code development

Data analysis as a design problem

Data analysis as a design problem

- The production of statistical analysis is not always acknowledge as a **craft**, requiring the reflection of its practices. Most of the training goes onto the statistical models, and less on how these are implemented or created (Parker, 2017).

- To amend the previous gap, we will be using the idea of **design constraints** from industrial design, (Broniwski, 1978; in Yau, 2015) to think about how to develop code, that gives us what we need. Constraints can be viewed as either limits or boundaries, but also as **conditioning factors**. For the purposes of the current workshop we will use the second interpretation.

- These constraints where thought for industrial design. We will adapt these for the problem of **producing results using secondary data from large scale assessment**.

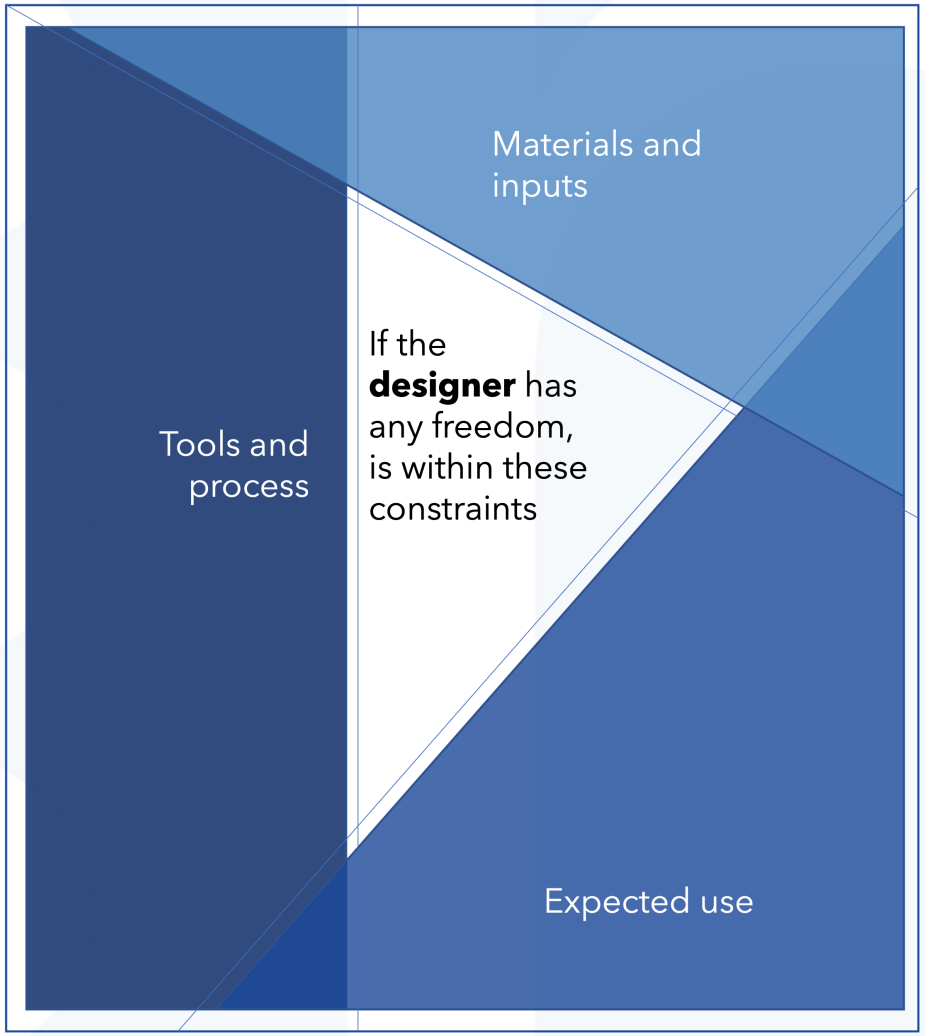


Figure 1. Triangle of constraints for insdustrial design

Data analysis as a design problem

> **Statistical training** often focuses on the narrative aspects of this process: mathematical derivations and proofs of statistical tests, methods and models. This foundational training is crucial to understanding the strengths and **limitations of conclusions** that can be drawn from a particular approach to analysing data.
>
> However, the **process of developing** the technical artifact is less frequently taught, or even acknowledged as a set of necessary skills. Given that this **process is complex and prone to error**, this hamstrings practitioners, keeping them from establishing **fluency** in the tools and allowing them to make common, avoidable and time-consuming mistakes.
>
> Parker, 2017

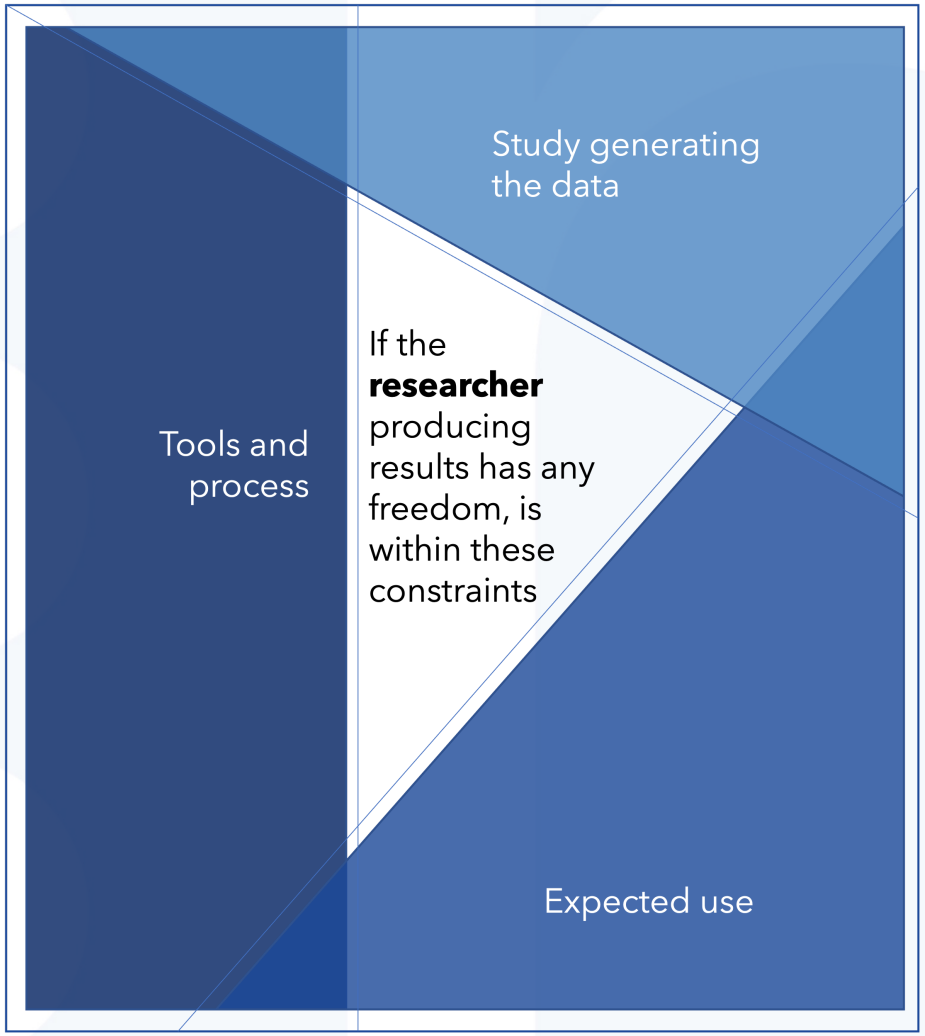


Figure 2. Triangle of constraints for code development of results

Data analysis as a design problem

- Researchers can move the boundaries by selecting the factors of the constrictions.
- In this workshop, we have selected **R** as our main **tool** for code development
- We will be using **ERCE 2019** as our main **study** providing data for us to develop the code to generate results.
- To illustrate the generation of results as a code development process, we will start by reproducing regional results from the ERCE 2019. In particular, we will be using distributed results published by UNESCO with the hashtag **#ponlelupa**. We will replicate the proportion of students in sixth grade reaching the minimum expected reading competence (31.2%).
- Let see the published results!

- source: https://twitter.com/UNESCOSantiago/status/1495840758087176204/photo/1)

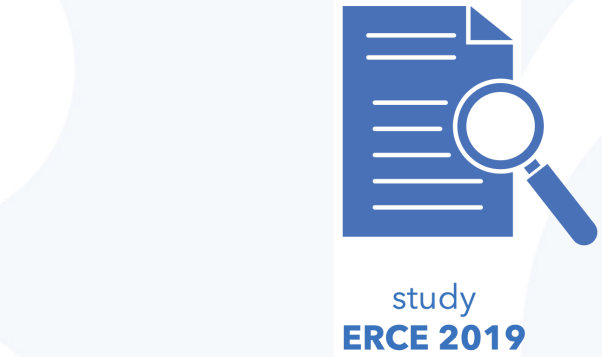


Figure 3. Triangle of constraints for code development of the current example

#ponlelupa

alcanza el **nivel mínimo de competencia en Lectura**

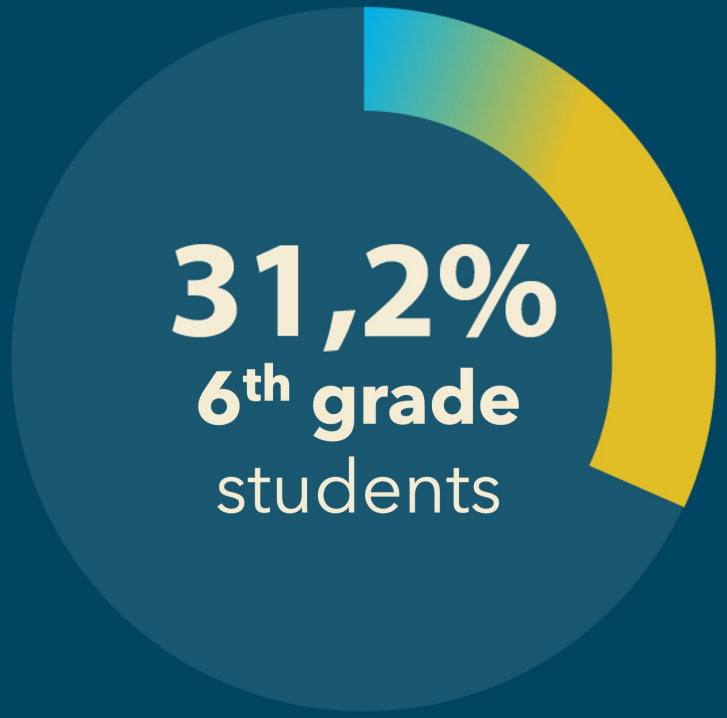
Laboratorio Latinoamericano Evaluación Calidad Educación

unesco

Con el apoyo de:

unicef

#ponlelupa



reaches the **minimal level of reading competence**

Laboratorio Latinoamericano Evaluación Calidad Educación

unesco

Con el apoyo de:

unicef

Workshop

Data analysis as code development

Writting R code as code development

Data analysis as code development

- Reproducing the percentage of students reaching a minimum level of profficiency, is an example that involves
    - a) estimating **percentages**
    - b) out of ordinal **plausible values**
    - c) while including **the sampling design**
- We will need code for the following steps:
    1. load the data
    2. prepare the data
    3. creating a survey object
    4. estimating percentage per plausible value
    5. combined estimates
    6. edit a table
- In the following sections we will develop code in sequence from loading the data, to create an editable table. The idea is to have a single structured code that can generate the results we want. We want this procedure to be **reproducible**, **scalable**, and **transferable**.

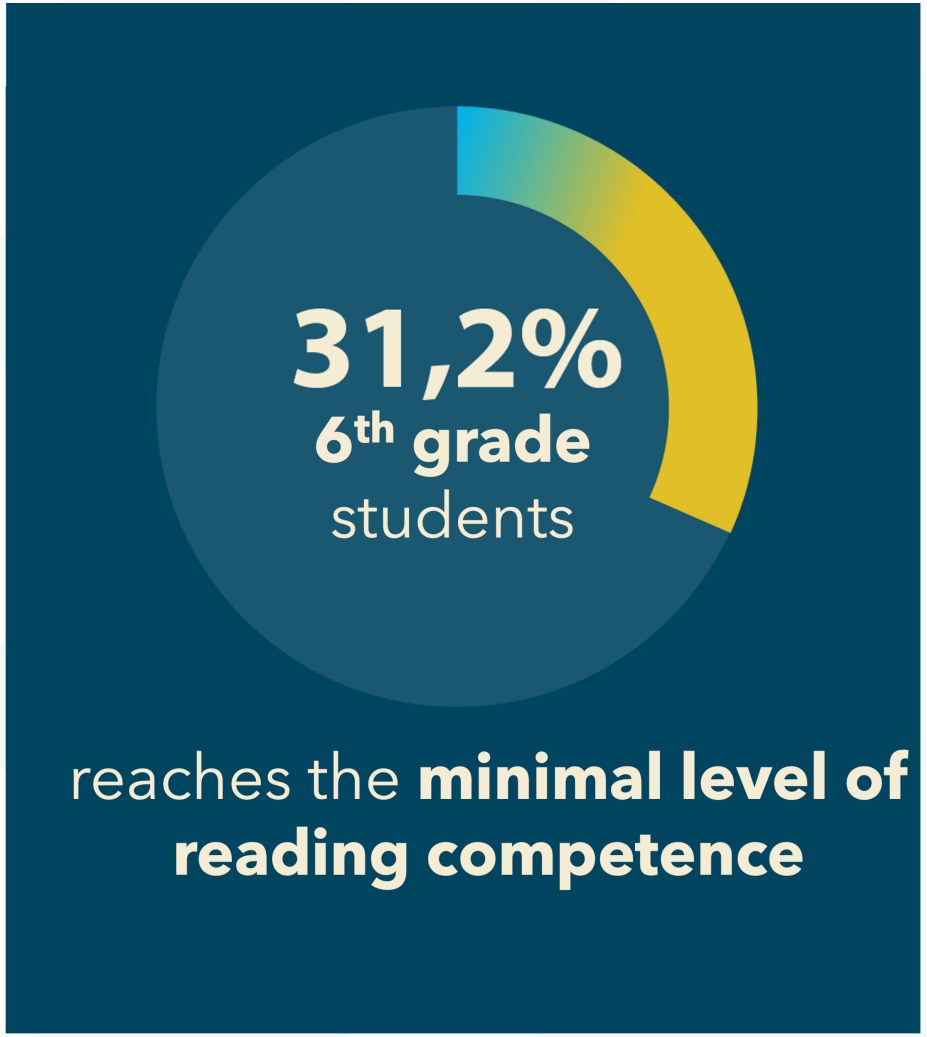


Figure 4. mimimum reading comprehension of 6th graders (ERCE 2019)

Step 0: **install needed libraries**

```
# ------------------------------------------------------------------
# libraries use in the following code
# ------------------------------------------------------------------

#-----------------------------------------------
# libraries
#-----------------------------------------------

# collection of libraries to handle data
install.packages('tidyverse')

# library to install libraries from github
install.packages('devtools')

# library with ERCE 2019 data
devtools::install_github('dacarras/erce',force = TRUE)

# library to generate survey estimates
install.packages('survey')

# library to generate survey estimates
install.packages('srvyr')

# library to get estimates with plausible values
install.packages('mitools')

# library to save tables into excel
install.packages('openxlsx')

# library to fit multilevel models
install.packages('WeMix')
```

Step 0: **install needed libraries**

```
# ------------------------------------------------------------------
# libraries use in the following code
# ------------------------------------------------------------------

#----------------------------------------------
# libraries
#----------------------------------------------

# collection of libraries to handle data
install.packages('tidyverse')

# library to install libraries from github
install.packages('devtools')

# library with ERCE 2019 data
devtools::install_github('dacarras/erce',force = TRUE)

# library to generate survey estimates
install.packages('survey')

# library to generate survey estimates
install.packages('srvyr')

# library to get estimates with plausible values
install.packages('mitools')

# library to save tables into excel
install.packages('openxlsx')

# library to fit multilevel models
install.packages('WeMix')
```

To generate results with ERCE 2019 data, we need different R libraries.

R libraries are collection of functions, commands and procedures use to produce results.

The selected libraries on the left are a minimal collection of libraries to

Steps 1 and 2: **load data** and **prepare data**

```
# ------------------------------------------------------------------------
# minimal reading profficiency
# ------------------------------------------------------------------------

#-----------------------------------------------
# load data
#-----------------------------------------------

library(dplyr)
data_raw <- erce::erce_2019_qa6

#-----------------------------------------------
# clustering variables
#-----------------------------------------------

data_clu <- data_raw %>%
erce::remove_labels() %>%
mutate(id_s = as.numeric(as.factor(paste0(IDCNTRY, "_", STRATA)))) %>%
mutate(id_j = as.numeric(as.factor(paste0(IDCNTRY, "_", IDSCHOOL)))) %>%
mutate(id_i = seq(1:nrow(.)))

#-----------------------------------------------
# recoding profficiency levels
#-----------------------------------------------

data_rec <- data_clu %>%
mutate(lan_min_1 = case_when(
LAN_L1 == 'I'   ~ 0,
LAN_L1 == 'II'  ~ 0,
LAN_L1 == 'III' ~ 1,
LAN_L1 == 'IV'  ~ 1)) %>%
mutate(lan_min_2 = case_when(
LAN_L2 == 'I'   ~ 0,
LAN_L2 == 'II'  ~ 0,
LAN_L2 == 'III' ~ 1,
LAN_L2 == 'IV'  ~ 1)) %>%
mutate(lan_min_3 = case_when(
LAN_L3 == 'I'   ~ 0,
LAN_L3 == 'II'  ~ 0,
LAN_L3 == 'III' ~ 1,
LAN_L3 == 'IV'  ~ 1)) %>%
mutate(lan_min_4 = case_when(
LAN_L4 == 'I'   ~ 0,
LAN_L4 == 'II'  ~ 0,
LAN_L4 == 'III' ~ 1,
LAN_L4 == 'IV'  ~ 1)) %>%
mutate(lan_min_5 = case_when(
LAN_L5 == 'I'   ~ 0,
LAN_L5 == 'II'  ~ 0,
LAN_L5 == 'III' ~ 1,
LAN_L5 == 'IV'  ~ 1))
```

Steps 1 and 2: **load data** and **prepare data**

```
# ------------------------------------------------------------------------
# minimal reading profficiency
# ------------------------------------------------------------------------

#-----------------------------------------------
# load data
#-----------------------------------------------

library(dplyr)
data_raw <- erce::erce_2019_qa6

#-----------------------------------------------
# clustering variables
#-----------------------------------------------

data_clu <- data_raw %>%
erce::remove_labels() %>%
mutate(id_s = as.numeric(as.factor(paste0(IDCNTRY, "_", STRATA)))) %>%
mutate(id_j = as.numeric(as.factor(paste0(IDCNTRY, "_", IDSCHOOL)))) %>%
mutate(id_i = seq(1:nrow(.)))

#-----------------------------------------------
# recoding profficiency levels
#-----------------------------------------------

data_rec <- data_clu %>%
mutate(lan_min_1 = case_when(
LAN_L1 == 'I'   ~ 0,
LAN_L1 == 'II'  ~ 0,
LAN_L1 == 'III' ~ 1,
LAN_L1 == 'IV'  ~ 1)) %>%
mutate(lan_min_2 = case_when(
LAN_L2 == 'I'   ~ 0,
LAN_L2 == 'II'  ~ 0,
LAN_L2 == 'III' ~ 1,
LAN_L2 == 'IV'  ~ 1)) %>%
mutate(lan_min_3 = case_when(
LAN_L3 == 'I'   ~ 0,
LAN_L3 == 'II'  ~ 0,
LAN_L3 == 'III' ~ 1,
LAN_L3 == 'IV'  ~ 1)) %>%
mutate(lan_min_4 = case_when(
LAN_L4 == 'I'   ~ 0,
LAN_L4 == 'II'  ~ 0,
LAN_L4 == 'III' ~ 1,
LAN_L4 == 'IV'  ~ 1)) %>%
mutate(lan_min_5 = case_when(
LAN_L5 == 'I'   ~ 0,
LAN_L5 == 'II'  ~ 0,
LAN_L5 == 'III' ~ 1,
LAN_L5 == 'IV'  ~ 1))
```

The **first step** is straightforward if we already installed erce library. To **load the data** of interest, we can use the command **`erce::`**. and call the file name we need. The sixth graders data file names is **`erce_2019_qa6`**. Thus the line of code to load the data is:

```
data_raw <- erce::erce_2019_qa6
```

The **second step** requires more tasks to **prepare the data**. We will **remove the data labels** from the data object.

```
erce::remove_labels() %>%
```

Then, we create the minimal clustering variables. These are **easier to write clustering variables** including students unique id (**`id_i`**), a unique cluster variable for schools (**`id_j`**), and a unique vector for stratification variables (**`id_s`**).

```
mutate(ids = as.numeric(as.factor(paste0(IDCNTRY, "", STRATA)))) %>%
mutate(idj = as.numeric(as.factor(paste0(IDCNTRY, "", IDSCHOOL)))) %>%
mutate(id_i = seq(1:nrow(.)))
```

Finally we **recode** the profficiency level variables, from **`LAN_1`** to **`LAN_5`**.

```
[...]
mutate(lan_min_1 = case_when(
LAN_L1 == 'I'   ~ 0,
LAN_L1 == 'II'  ~ 0,
LAN_L1 == 'III' ~ 1,
LAN_L1 == 'IV'  ~ 1)) %>%
[...]
```

Step 3: **Create survey object**

```r
# ------------------------------------------------------------------
# minimal reading profficiency
# ------------------------------------------------------------------

# [...]

#-----------------------------------------------
# data with survey object
#-----------------------------------------------

# survey method: taylor series linearization
data_tsl  <- survey::svydesign(
             data    = data_rec,
             weights = ~WS,
             strata  = ~id_s,
             id = ~id_j,
             nest = TRUE)

# Note: we correct that strata with a single cluster.

library(survey)
options(survey.lonely.psu="adjust")
```

Step three is creating a **survey object**. This is an special type of data table, where the sampling survey design is declared. In this case we are using **Taylor Series Linearization** (TSL) to get corrected standard errors. There are other methods for this procedure, but TSL is enough to get the corrected point estimates. Here, we are using the `library(survey)`.

Additionally, we are letting R know what to do, in case a strata contains a single primary sampling unit. In this case is adjusting the standard errors for scenarios where there are only a single cluster within strata.

Another observation to bear in mind, is we want to get regional estimates. These are **pooled** estimates from all the countries simultaneously. Two feratures are relevant for this task: the scaled survey weights `WS`, and the stratification variable `id_s`.

Step 3: **Create survey object**

```
# ------------------------------------------------------------------
# minimal reading profficiency
# ------------------------------------------------------------------

# [...]

#-----------------------------------------------
# data with survey object
#-----------------------------------------------

# survey method: taylor series linearization
data_tsl  <- survey::svydesign(
             data    = data_rec,
             weights = ~WS,
             strata  = ~id_s,
             id = ~id_j,
             nest = TRUE)

# Note: we correct that strata with a single cluster.

library(survey)
options(survey.lonely.psu="adjust")
```

Step three is creating a **survey object**. This is an special type of data table, where the sampling survey design is declared. In this case we are using **Taylor Series Linearization** (TSL) to get corrected standard errors. There are other methods for this procedure, but TSL is enough to get the corrected point estimates. Here, we are using the `library(survey)`.

Additionally, we are letting R know what to do, in case a strata contains a single primary sampling unit. In this case is adjusting the standard errors for scenarios where there are only a single cluster within strata.

Another observation to bear in mind, is we want to get regional estimates. These are **pooled** estimates from all the countries simultaneously. Two feratures are relevant for this task: the scaled survey weights `WS`, and the stratification variable `id_s`.

Scaled survey weights helps us to get regional estimates, where all countries contribute equally (Gonzalez, 2012). These are survey weights scaled up to 1000, which expands sampled observations per country in an equivalent way. As such, each country contributes to the estimates in the same manner. In contraste, if we use `WT`, which is the **total survey weight**, our estimates will be distorted. Survey weights expand observations to the expected total of cases in the sampling frame. Hence, countries with a larger population of students would have more weight (e.g., Brasil, Mexico), than countries with a smaller population of students (e.g., Uruguay, Paraguay).

The second feature relevant to get regional estimates are the stratification variables. When we are generating results for all countries simultaneously, we need to be sure our stratification variables are unique across countries. We are making sure our stratification variablea are unique across countries using the variable we created earlier, in the **data preparation step**.

```
mutate(ids = as.numeric(as.factor(paste0(IDCNTRY, "", STRATA)))) %>%
```

Steps 4 and 5: **Estimating percentages per plausible value** and **combined** estimates.

```
# ------------------------------------------------------------------
# minimal reading profficiency
# ------------------------------------------------------------------

# [...]

#-----------------------------------------------
# percentages with plausible values
#-----------------------------------------------

results <- mitools::withPV(
   mapping = lan_min ~ lan_min_1 + lan_min_2 + lan_min_3 + lan_min_4 + lan_min_5,
   data = data_tsl,
   action = quote(
    survey::svymean( ~lan_min, design = data_tsl)
    ),
   rewrite = TRUE
  )


#-----------------------------------------------
# display results
#-----------------------------------------------

summary(mitools::MIcombine(results))
```

```
Multiple imputation results:
      withPV.survey.design(mapping = lan_min ~ lan_min_1 + lan_min_2 +
    lan_min_3 + lan_min_4 + lan_min_5, data = data_tsl, action = quote(survey::svymean(~lan_min,
    design = data_tsl)), rewrite = TRUE)
      MIcombine.default(results)
              results             se       (lower       upper) missInfo
lan_min_1 0.3118589523 0.003815336074 0.3041889541 0.3195289505     32 %
```

The following code is a bit cumbersome. We will deconstruct its pieces and arguments to describe it. Every function in R, consist of a statement, within a library, with a number of arguments. In general these can written in the following way:

```
library_name::function_of_interest(argument_1 = 'input_1', argument_2 = 'input_2').
```

Steps 4 and 5: **Estimating percentages per plausible value** and **combined** estimates.

```
# ------------------------------------------------------------------
# minimal reading profficiency
# ------------------------------------------------------------------

# [...]

#-----------------------------------------------
# percentages with plausible values
#-----------------------------------------------

results <- mitools::withPV(
   mapping = lan_min ~ lan_min_1 + lan_min_2 + lan_min_3 + lan_min_4 + lan_min_5,
   data = data_tsl,
   action = quote(
    survey::svymean( ~lan_min, design = data_tsl)
    ),
   rewrite = TRUE
  )


#-----------------------------------------------
# display results
#-----------------------------------------------

summary(mitools::MIcombine(results))
```

```
Multiple imputation results:
      withPV.survey.design(mapping = lan_min ~ lan_min_1 + lan_min_2 +
    lan_min_3 + lan_min_4 + lan_min_5, data = data_tsl, action = quote(survey::svymean(~lan_min,
    design = data_tsl)), rewrite = TRUE)
      MIcombine.default(results)
              results             se       (lower       upper) missInfo
lan_min_1 0.3118589523 0.003815336074 0.3041889541 0.3195289505     32 %
```

The following code is a bit cumbersome. We will deconstruct its pieces and arguments to describe it. Every function in R, consist of a statement, within a library, with a number of arguments. In general these can written in the following way:

```
library_name::function_of_interest(argument_1 = 'input_1', argument_2 = 'input_2').
```

The first highlighted argument, **`mapping`**, is a way to tell the library **`mitools`**, which are the variables containing the recoded **plausible values**. The general logic of this argunent, is that we can say:

```
theta ~ pv1 + pv2 + pv3 + pv4 + pv5
```

and then we can use **`theta`** to get **descriptives** and **regression models**. In other words, that line tells the library to get the estimate for the first, second, and so forth plausible value. We will combined the estimates in a later stage.

The second argument, **`data`**, expects our survey object. That is, a data frame containing the variables of interest, that the library survey can use to produce estimates using the sampling design.

Within the third argument, within **`action = quote()`**, we can include the survey commands to get descriptives and regressions. For the present example we need a mean.

```
survey::svymean( ~lan_min, design = data_tsl)
```

Finally, all estimates are save in the object **`results`**. To read those resuls we need to combined these, following the Rubin-Schafer rules. To achieve this task, we use the following code:

```
summary(mitools::MIcombine(results))
```

Steps 6: **edit table of results**

```
# ------------------------------------------------------------------
# minimal reading profficiency
# ------------------------------------------------------------------

# [...]

#-----------------------------------------------
# save estimates
#-----------------------------------------------

estimates <- summary(mitools::MIcombine(results))

#-----------------------------------------------
# edit table of results
#-----------------------------------------------


table_read <- estimates %>%
               tibble::rownames_to_column("lan_min") %>%
               rename(
                lan = results,
                lan_se = se,
                ll = 4,
                ul = 5,
                miss = 6
                ) %>%
               mutate(lan = lan*100) %>%
               mutate(lan_se = lan_se*100) %>%
               mutate(ll = ll*100) %>%
               mutate(ul = ul*100)

# -----------------------------------------------
# display table
# -----------------------------------------------

options(digits=10)
options(scipen = 999999)

knitr::kable(table_read, digits = 1)

#-----------------------------------------------
# export results
#-----------------------------------------------

table_read %>%
openxlsx::write.xlsx(.,
  'table_minimum_reading_profficiency_6th_graders.xlsx',
  overwrite = TRUE)
```

Steps 6: **edit table of results**

```
# ------------------------------------------------------------------
# minimal reading profficiency
# ------------------------------------------------------------------

# [...]

#-----------------------------------------------
# save estimates
#-----------------------------------------------

estimates <- summary(mitools::MIcombine(results))

#-----------------------------------------------
# edit table of results
#-----------------------------------------------


table_read <- estimates %>%
               tibble::rownames_to_column("lan_min") %>%
               rename(
                lan = results,
                lan_se = se,
                ll = 4,
                ul = 5,
                miss = 6
                ) %>%
               mutate(lan = lan*100) %>%
               mutate(lan_se = lan_se*100) %>%
               mutate(ll = ll*100) %>%
               mutate(ul = ul*100)

# ------------------------------------------------
# display table
# ------------------------------------------------

options(digits=10)
options(scipen = 999999)

knitr::kable(table_read, digits = 1)

#-----------------------------------------------
# export results
#-----------------------------------------------

table_read %>%
openxlsx::write.xlsx(.,
  'table_minimum_reading_profficiency_6th_graders.xlsx',
  overwrite = TRUE)
```

The results generated by **`summary(mitools::MIcombine(results))`** are displayed in the are console, in the following manner:

```
> summary(mitools::MIcombine(results))
Multiple imputation results:
    withPV.survey.design(mapping = lan_min ~ lan_min_1 + lan_min_2 +
    lan_min_3 + lan_min_4 + lan_min_5, data = data_tsl,
    action = quote(
      survey::svymean(~lan_min, design = data_tsl)
      ), rewrite = TRUE)
      MIcombine.default(results)
               results             se       (lower       upper) missInfo
lan_min_1 0.3118589523 0.003815336074 0.3041889541 0.3195289505     32 %
```

We can edit the generated results in a table, to make it look closer to our expected results, a number figure with a single decimal. We **edit** the object **`estimates`**, renaming its columns. And finally, we use the function **`knitr::kable()`** to display the results with a table in the console.

```
|lan_min   |   lan| lan_se|    ll|    ul|miss |
|:---------|-----:|------:|-----:|-----:|:----|
|lan_min_1 | 31.19|   0.38| 30.42| 31.95|32 % |
```

Finally we can export the generated results to an excel file. We use the function **`openxlsx::write.xlsx()`** for this purposes. The created excel file name is **`table_minimum_reading_profficiency_6th_graders.xlsx`**

In the following link the full code can be found, as an **rmarkdown file**. **We will run the code before the next slides.**

```
# url address
https://github.com/dacarras/cies_2023_erce_2019/
blob/main/code_examples/
02_developing_code_example.rmd
```

Workshop

Closing Remarks

Writting R code as code development

Data Analysis workflow

From data preparation to the presentation of results, we can view the code development as a way to resolve different problems.

In this view, each step becomes an answer for a “how can i” or a “how can we” statement. Under this view, we can set an ordered sequence of the problems we need to tackle:

- How can we import the study data
- How can we merge the data from each country
- How can we know the content of the data
- How can we deal with the complex sample designs
- How can we make sensible interpretation of variables
- How can we transfer the prepare data for other statistical packages

The present workflow is an opinionated analysis (Parker, 2017). This is not the only way to prepare data. This is reasonable way to prepare data among others.

The presented example cover these different problems, in a compact way. For more details on this framework see https://youtu.be/RSkzQ_SNN8Q?t=1631

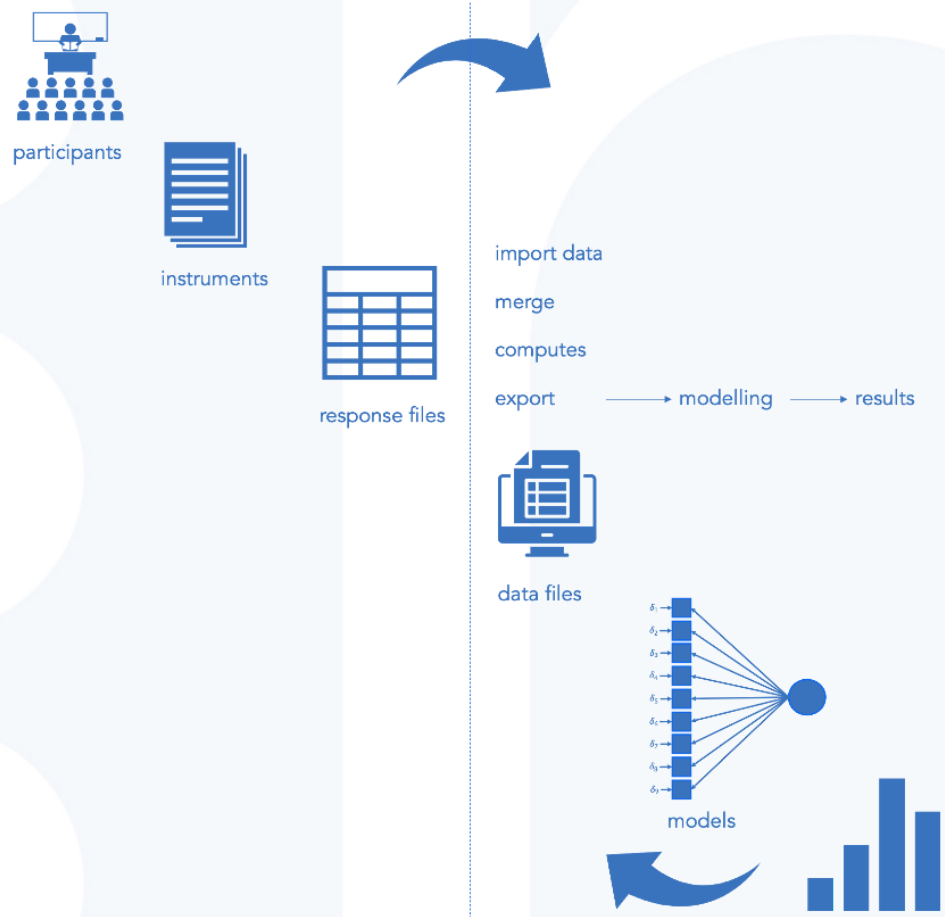


Figure 5. Data Analysis workflow

Data Analysis workflow

- One thing to considered when the process of data analysis is view as a **code development task**, is its **iterative process nature**.
- This latter feature means **one codes progressively**, and ends when one has reach a goal. In between, starting and reaching one of the goals within the ordered sequence, **is common to make many mistakes**.
    - Sometimes, the experience of this process is very **frustrating**.
    - In time, making coding errors, shouldn't be surprise. Amending code could be view in a similar light to copy-editing manuscripts. **Writing** is **an iterative process. Coding** for **data analysis is also an iterative process.** The end goal is to produce an error free, and bug free code; whereas the process of writting itself necesarily may contain errors.
- Is not relevant how many errors were committed to reach the goal. What is important is that the **developed code** reaches the **goal**, **bug free**, and **error free**.

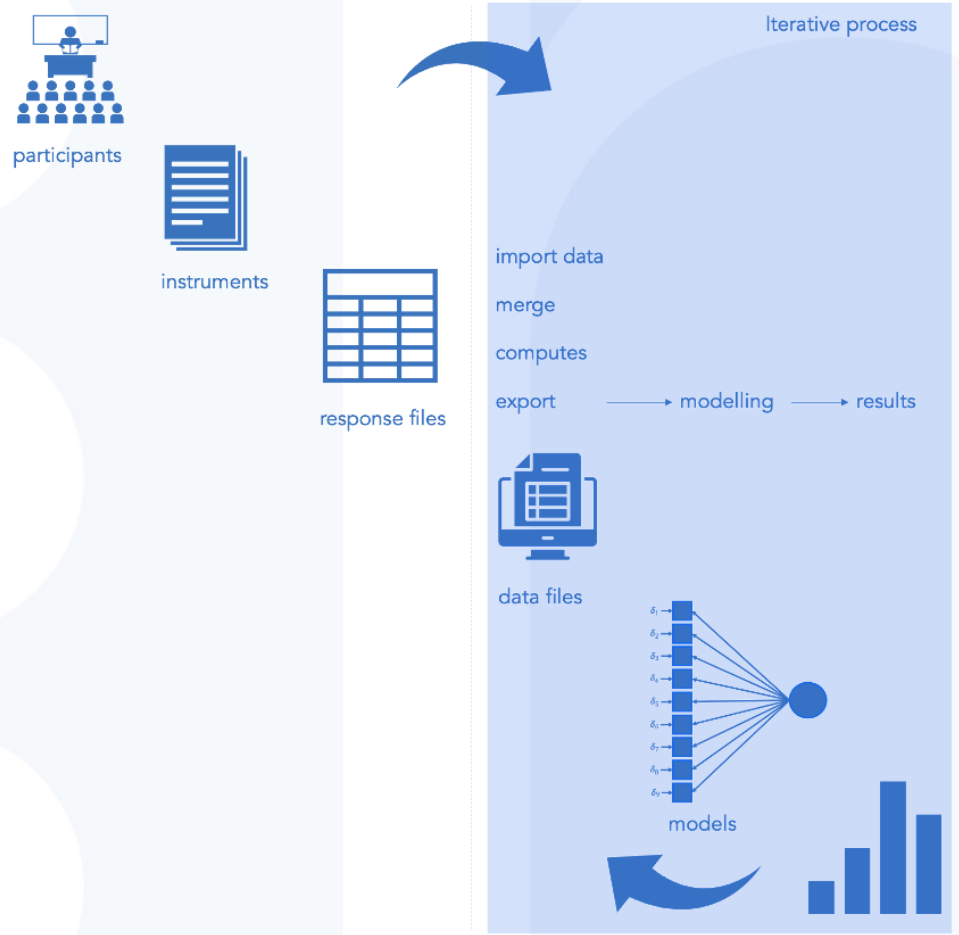


Figure 6. Data Analysis workflow and its iterative feature

Many thanks!

Carrasco, D., PhD
Centro de Medición MIDE UC,
Pontificia Universidad Católica de Chile
https://dacarras.github.io/

References

Parker, H. (2017). Opinionated analysis development (pp. 1–13). https://doi.org/10.7287/peerj.preprints.3210

Yau, N. (2015) https://flowingdata.com/2015/02/12/visualization-constraints

Gonzalez, E. J. (2012). Rescaling sampling weights and selecting mini-samples from large-scale assessment databases. IERI Monograph Series Issues and Methodologies in Large-Scale Assessments, 5, 115–134.