Managing large codebases in R

Presented by
Alex Bertram
Ryo Nakagawara
Presentation outline

● Intro

● Principles & Practice
  ○ Adopting a coding style for your team
  ○ Organizing code into functions
  ○ Organizing functions into packages
  ○ Documenting code
  ○ Using version control
Introduction
ActivityInfo & R

ActivityInfo is a user-friendly relational database for M&E, Case Management, and Humanitarian Coordination that seamlessly integrates with R.

https://www.activityinfo.org/signUp
What is a “large code base”? 

- More than one person working on the code 
- More than a few files...
Examples - OCHA Libya
Examples - R4V

Activity Info Environment

National SW  National SW  National SW

Regional SW

Single form

Strengths:
- National database can have additional information adapted to their context
- Common reference tables
- Comparability through regional database

Challenges:
- 3 different languages throughout the region
- Quality of the data reported by partners
- Deadlines and double reporting between national and regional databases
Examples - R4V Shiny App
Examples - Qualminer in Ecuador

Word co-occurrences and correlations

How many times each pair of words occurs co-together in responses?

QualMiner project: Word co-occurrences and correlations in qualitative data used in the Response for Venezuela
Principles
What is a coding style?

Adopting a common code style

- Rules that the whole team agrees on
- How to name functions, variables, datasets
- When to use spacing
- When and how to document functions
Why is a common code style important?

Adopting a common code style

- Code is written once, read a hundred times
- Is it …?
  - check_duplicates()
  - checkDuplicates()
  - check.duplicates()
  - CheckDuplicates()
  - ???
How to get started

Adopting a common code style

- **Recommendation**: http://adv-r.had.co.nz/Style.html
- Automated with **formatR**: https://yihui.org/formatr/
What is a function?

Organizing code into functions

Input 1
Input 2
Function
Output
Pure vs Impure functions

Pure functions

● Same inputs, same outputs
● No side effects (no reading, writing)
● Can be tested

Examples:

● flag_duplicates(hh_list)
● score_eligibility(hh_list)

Impure (imperative) functions

● Outputs depend on the outside world (reading from a file, from a server)
● Same inputs, (maybe) different outputs

Examples

● launch_missiles()
● read_hh_from_ai_form()
● write_updates_to_db(df)
Why functions?

Organizing code into functions

- Breaking code into smaller functions makes the code easier to read and understand
- Easier to compose functions together
- Individual (pure) functions can be tested
Function length

Organizing code into functions

- Strive for functions with max twenty lines.
- A function should “fit in your head”.

ActivityInfo
##### Make Arabic names consistence

```r
arabic_letter <- read.xlsx("./codes/arabic_letters.xlsx", sheet = "const"
raw_data$hob_arabic_name <- ngsub("\\s+", " ", raw_data$hob_arabic_name)
raw_data$hob_arabic_name <- ngsub(arabic_letter[old_char], arabic_letter[new_char]

raw_data$hob_spouse_name <- ngsub("\\s+", " ", raw_data$hob_spouse_name)
raw_data$hob_spouse_name <- ngsub(arabic_letter[old_char], arabic_letter[new_char]
```

##### HH names duplicates - Similarities check

```r
HH_duplicates_checks <- raw_data %>% dplyr::select("X_id", "QA_Code", "Interv"
  "district", "sub_district"
  #second row should be split
  "hob_arabic_name") %>%
dplyr::mutate(HH_duplicates_checks = 0)

if (nrow(HH_duplicates_checks) > 0) {
  for (i in 1:nrow(HH_duplicates_checks)){
    if (HH_duplicates_checks$interviewer[i] == "Master data") {
      break
    }
else{
  duplicate_vector_sim <- 1 - stringdist(HH_duplicates_checks$hob_arabic,HH_duplicates_checks$hob_arabic,
  nchar(as.character(HH_duplicates_checks$hob_arabic_name[i])))
  HH_duplicates_checks$HH_duplicates_checks[i] <- n(ahead(vector_sim))
```
Splitting pure and imperative parts

Organizing code into functions

● The “Functional Core, Imperative Shell” Pattern

upload_from_csv()

read_hh_from_csv() |> remove_duplicates() |> score_eligibility() |> import_hh_to_ai()
R Packages

Organizing functions into packages

- Combination of functions, dependencies, and documentation
- Standard structure
Why packages?

Organizing functions into packages

- Re-use common code
- Easier to work with many files than endlessly source()’ing.
Version control system

Using version control

- Most commonly used VCS today is Git
- Free hosting at GitHub.com, GitLab.com, BitBucket.com
- Supports collaboration