# 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

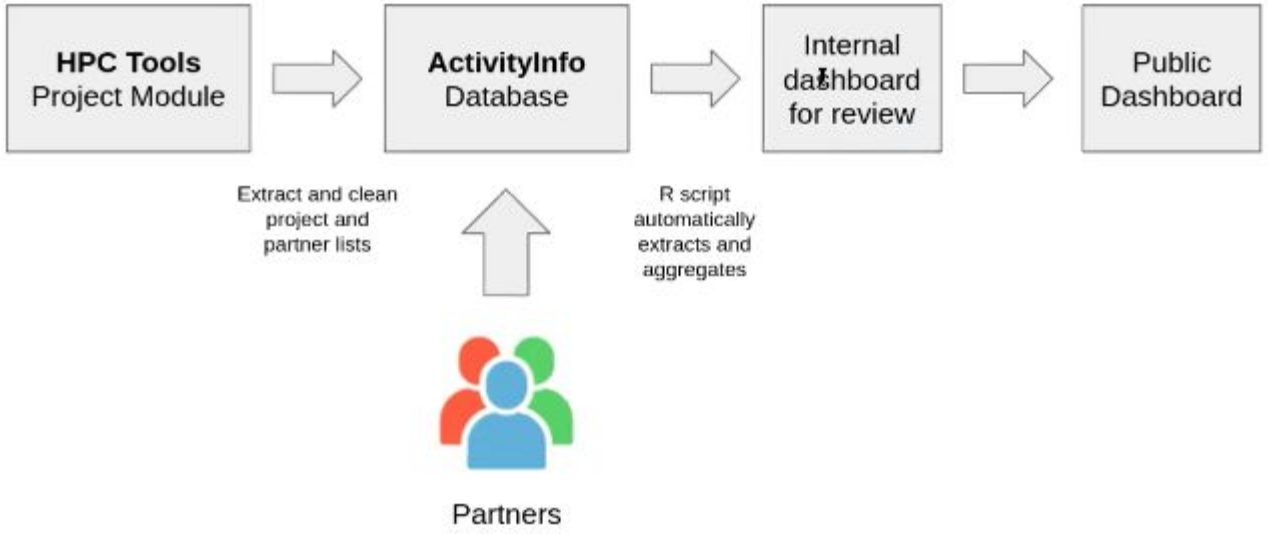
ActivityInfo is a user-friendly relational database for M&E, Case Management, and Humanitarian Coordination that seamlessly integrates with R.

https://www.activityinfo.org/signUp

# What is a “large code base” ?

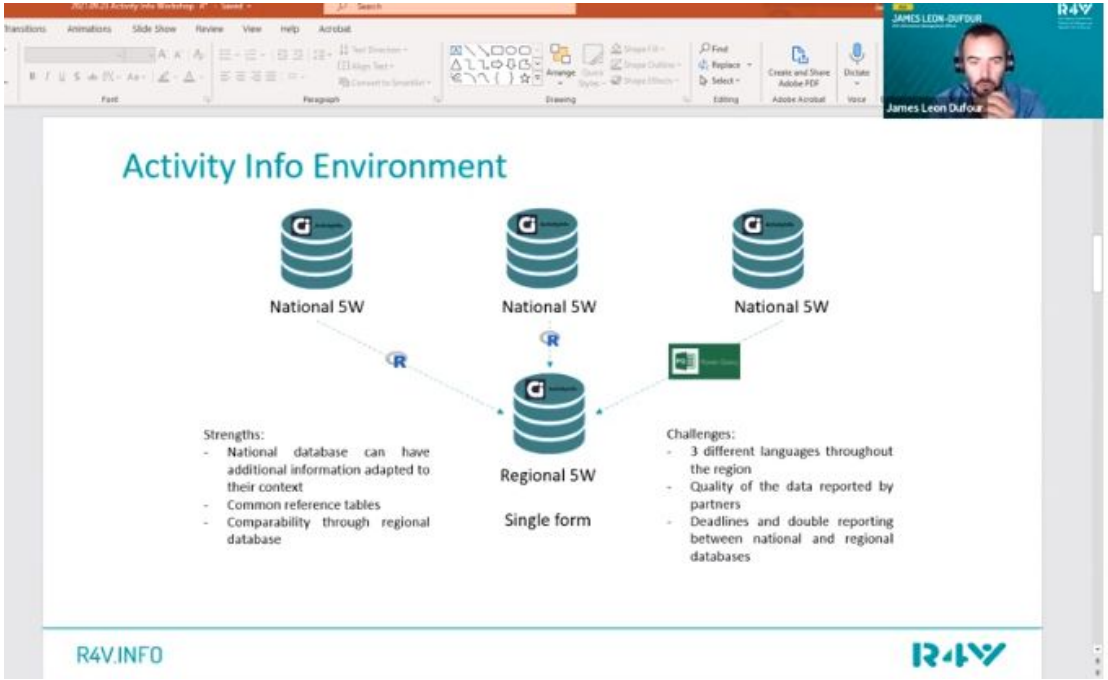
- More than one person working on the code
- More than a few files…

# Examples - OCHA Libya

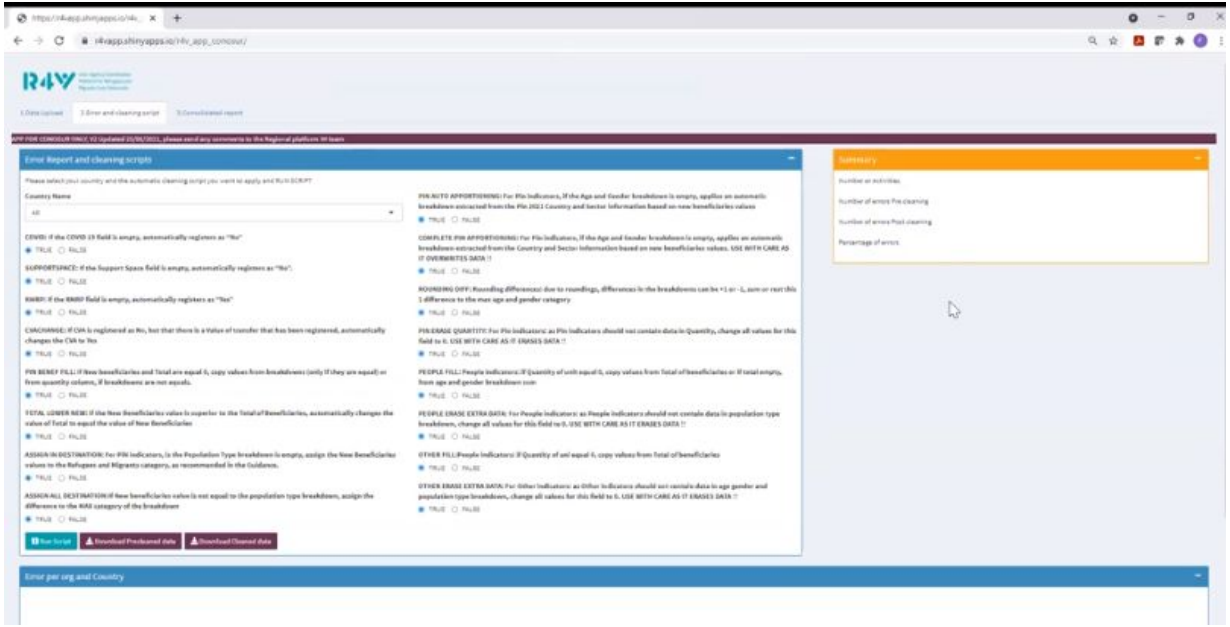



ActivityInfo

# Examples - R4V

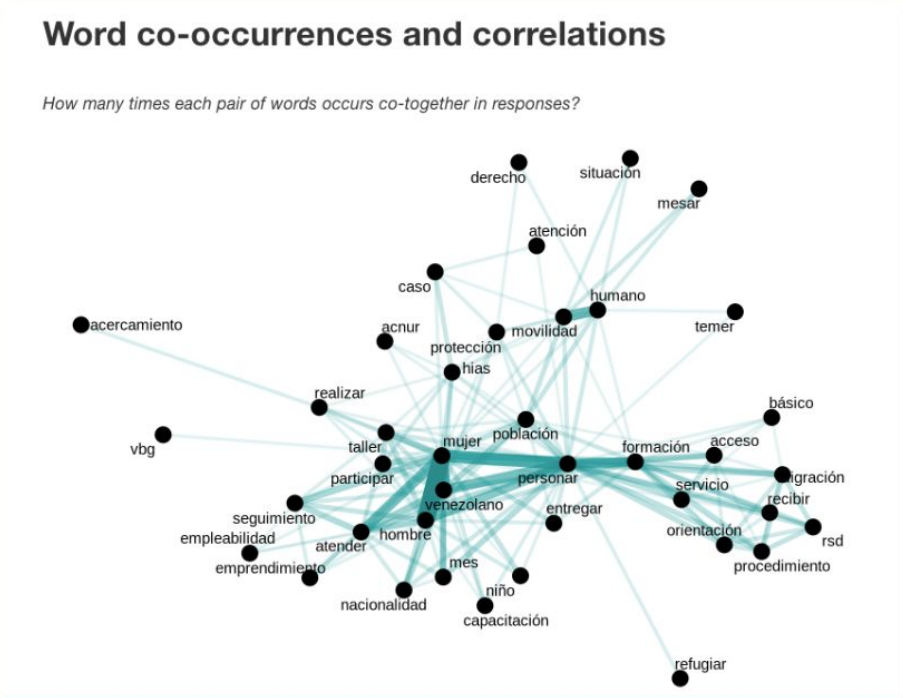


# Examples - R4V Shiny App




ActivityInfo

# Examples - Qualminer in Ecuador




QualMiner project: Word co-occurrences and correlations in qualitative data used in the Response for Venezuela

ActivityInfo

# 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()`
  - `???`

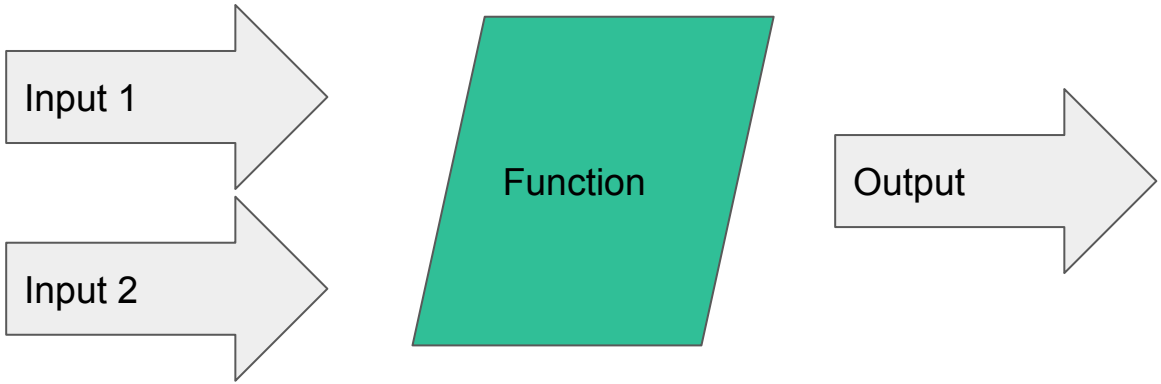
# How to get started

## 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



# 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”.

```
########### Make Arabic names consistence
{

  arabic_letter <- read.xlsx("../../pcodes/arabic_letters.xlsx", sheet = "consis

  raw_data$hoh_arabic_name <- mgsub("\\s+"," ", raw_data$hoh_arabic_name)
  raw_data$hoh_arabic_name <- mgsub(arabic_letter$old_char, arabic_letter$new_ch


  raw_data$hoh_spouse_name <- mgsub("\\s+"," ",raw_data$hoh_spouse_name)
  raw_data$hoh_spouse_name <- mgsub(arabic_letter$old_char, arabic_letter$new_ch
}

############ HH names duplicates - Similarities check
{
  HH_duplicates_checks <- raw_data %>% dplyr::select(c("X.id","QA_Code","intervi
                                                       "district","sub_district"
                                                       #second row should be spe
                                                       "hoh_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$interviewer[i] == "Master data") {
        break
      }else{
        duplicate_vector_simi<- 1 - stringdist(HH_duplicates_checks$hoh_arabic_r
                                               HH_duplicates_checks$hoh_arabic_r
          nchar(as.character(HH_duplicates_checks$hoh_arabic_name[i]))

        HH_duplicates_checks$HH_duplicates_checks[!is.na(duplicate_vector_simi)
```

ActivityInfo

# Splitting pure and imperative parts

## Organizing code into functions

- The “Functional Core, Imperative Shell” Pattern

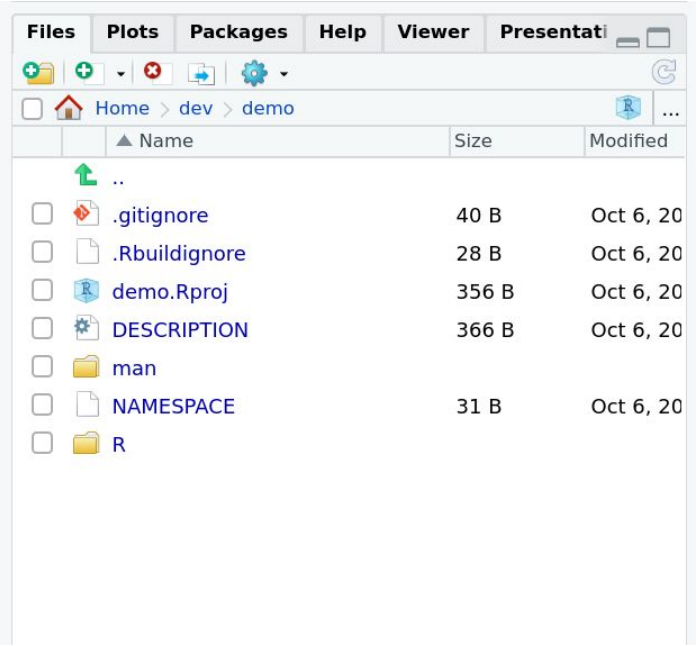
```
upload_from_csv()
```

```
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




ActivityInfo

# 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