Statistical Programming and Open Science Methods

Writing readable and reusable code

Joachim Gassen
Humboldt-Universität zu Berlin

September 02, 2022

SFB/Transregio 266
ACCOUNTING FOR TRANSPARENCY

Time table October 11

When?	What?
09:00	Writing readable and reusable code
10:30	Coffee
11:00	Debugging tools
12:30	Lunch and coffee
13:30	Relational databases and the concept of normalized data
14:30	Data wrangling and visualization fundamentals
15:30	Assignments and wrap up
16:00	End of event

Most of us have been going through loads of this

```
clear all
import excel "clean_data\interviews_quant_data_final_15-09-16.xlsx", sheet("MainData") firstrow
gen treated_case = .
replace treated_case = 1 if treatment == "1" & (obj_val == 1 & obj_perf_mgt == 0 & incent_finacc == 0 & in
replace treated_case = 2 if treatment == "2" & (obj_val == 0 & obj_perf_mgt == 1 & incent_finacc == 0 & in
replace treated_case = 3 if treatment == "3" & (obj_val == 1 & obj_perf_mgt == 0 & incent_finacc == 1 & in
replace treated_case = 4 if treatment == "4" & (obj_val == 0 & obj_perf_mgt == 1 & incent_finacc == 1 & in
gen treated = treated_case != .
gen exp_pilot = (treatment == "C" | treatment == "V")
gen failed_man_checks = !exp_pilot & treated_case == .
gen highest_college_ed = "none"
replace highest_college_ed = "bachelor" if bachelor==1
replace highest_college_ed = "masters" if masters==1
replace highest_college_ed = "phd" if phd==1
gen acctg_expertise = "none"
replace acctg_expertise = "intermediate (bus/econ and/or CFA)" if cfa | business_econ
replace acctg_expertise = "advanced (actg degree or acctg responsibility in current position)" if acctg_qu
gen occupation = ""
replace occupation = "fund manager" if (fund_manager | (value_funds_direct_mgmtM >0 & value_funds_direct_m
replace occupation = "analyst buy-side" if (occupation != "fund manager" & analyst  == 1 & buy_side ==1)
replace occupation = "analyst sell-side" if (occupation == "" & analyst  == 1 & sell_side ==1)
replace occupation = "other" if occupation == ""
gen asset_focus = ""
replace asset_focus = "equity only" if (equity == 1 & debt == 0)
replace asset_focus = "debt only" if (equity == 0 & debt == 1)
replace asset_focus = "equity and debt" if (equity == 1 & debt == 1)
replace asset_focus = "other" if (equity == 0 & debt == 0)
gen firm_focus = ""
replace firm_focus = "public only" if (public == 1 & private == 0)
replace firm_focus = "private only" if (public == 0 & private == 1)
replace firm_focus = "public and private" if (public == 1 & private == 1)
replace firm_focus = "other" if (public == 0 & private == 0)
```

Open Science needs readable code!

- To make an impact, code needs to be digestible for the reader
- Besides a well-designed project setup, accessible code is a key ingredient to enable others to contribute to your work

But what are the key ingredients to make your code readable and reusable?

Rule #1: Use a style guide

Each serious programming language has established style guides, e.g.:

- https://google.github.io/styleguide/ for various languages
- https://style.tidyverse.org for R
- https://www.python.org/dev/peps/pep-0008/ The "official" python style guide

Linters can help with getting your code in shape

Rule #2: Write code as you speak

```r
# Bad:

df <- read_csv("data/sub.csv")
x <- nrow(df[!duplicated(df[, 3:4]), 3:4])
sprintf("There are %d registrants", x)

# Good:

read_csv("data/sub.csv") %>%
  select(cik, name) %>%
  distinct() %>%
  nrow() -> count_sec_reg

sprintf("There are %d registrants", count_sec_reg)
```

Rule #3: Use functions for reusable steps

```
count_distinct_obs <- function(df, ...) {
  as_tibble(df) %>%
    select(...) %>%
    distinct() %>%
    nrow()
}

read_csv("data/sub.csv") %>%
  count_distinct_obs(cik, name) -> count_sec_reg

sprintf("There are %d registrants", count_sec_reg)
```

Rule #4: Keep functions short and indentation levels low

```
# Bad:

if (use_server_ok()) {
  if (ping_server_ok()) {
    if (connect_server_ok()) {
      df <- read_server_data()
    } else {
      if (fall_back_ok()) {
        df <- read_fall_back()
      } else stop("No data")
    }
  } else {
    if (fall_back_ok()) {
      df <- read_fall_back()
    } else stop("No data")
  }
} else {
  if (fall_back_ok()) {
    df <- read_fall_back()
  } else stop("No data")
}
do_something_with_data(df)
```

Rule #4: Keep functions short and indentation levels low

```
# Good:

test_server_ok <- function{
  if (!ping_server_ok()) return(FALSE)
  if (!connect_server_ok()) return(FALSE)
  return(TRUE)
}

get_data <- function() {
  if (use_server_ok()) {
    if (test_server_ok()) return(read_server_data())
  }
  if (fall_back_ok()) return(read_fall_back())
  else stop("No data")
}

get_data() %>%
  do_something_with_data()
```

Rule #5: Check for errors

```
count_distinct_obs <- function(df, ...) {
  as_tibble(df) %>%
    select(...) %>%
    distinct() %>%
    nrow()
}

count_distinct_obs(42)
## [1] 1
```

Rule #5: Check for errors

```r
count_distinct_obs <- function(df, ...) {
  if (!is.data.frame(df)) {
    stop("First parameter must be a data frame", call. = FALSE)
  }
  if (missing(...)) {
    stop("Must have at least one column variable", call. = FALSE)
  }

  df %>%
      select(...) %>%
      distinct() %>%
      nrow()
}

count_distinct_obs(42)
## Error: First parameter must be a data frame
count_distinct_obs(data.frame(a = 1:10))
## Error: Must have at least one column variable
count_distinct_obs(data.frame(a = 1:10), b)
## Error in `select()`:
## ! Can't subset columns that don't exist.
## x Column `b` doesn't exist.
count_distinct_obs(data.frame(a = 1:10), a)
## [1] 10
```

Rule #6: Don't document code, document functions

```r
#' @title Counts the distinct observations for a sub-set of a data frame
#'
#' @description
#' Reads a data frame and counts the number of distinct observations based
#' on a subset of provided columns.
#' @param df Data frame
#' @param ... The list of columns to include (cannot be empty)
#' @return The number of observations that are not identical across the selected
#' columns
#'
#' @examples
#' df <- data.frame(a = rnorm(5), b = 1, c = c(1, 2, 1, 2, 3))
#' count_distinct_obs(df, b, c)
#' @export

count_distinct_obs <- function(df, ...) {
  if (!is.data.frame(df)) {
    stop("First parameter must be a data frame", call. = FALSE)
  }
  if (missing(...)) {
    stop("Must have at least one column variable", call. = FALSE)
  }

  df %>%
      select(...) %>%
      distinct() %>%
      nrow()
}
```