Statistical Programming and Open Science Methods

Debugging tools

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

Disclaimer

Some of the following is borrowed from chapter 22 of Hadley Wickham (2019): Advanced R, https://adv-r.hadley.nz

Principle

> Finding your bug is a process of confirming the many things that you believe are true — until you find one which is not true.
>
> —— Norm Matloff

- Google the error message
- Make the bug repeatable by creating a reproducible example
- Figure out where it is
- Fix it and test it

Most bugs are simple

- Typos are paramount and sometimes hard to spot (a good editor with syntax high-lightening helps!)
- The best way to avoid bugs is do adopt a readable coding style
- Debugging and unit testing go hand in hand (we will talk about unit testing in the February block)
- Do not ignore messages and warnings that your code throws at you. They are there for a reason,

Error messages are not always self-explanatory but almost always informative

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

read_csv("../data/sub.csv") %>%
  count_distinct_obs[cik, name] -> count_sec_reg
## Error in `[.tbl_df`(., count_distinct_obs, cik, name): object 'cik'
```

Debugging tools

- Traceback
- Debug on error
- Breakpoints
- Logging
- Dump analysis

Let's do a case: Take a look at `code/calc_state_of_incorp_distance_bugged.R`

Corner cases

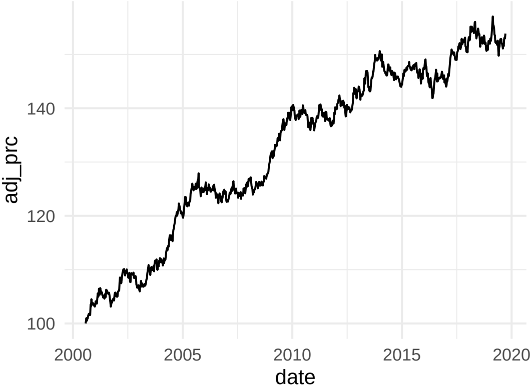
```
# This function returns a p % sample of a vector v. It always
# chooses at least q observations to avoid small samples

sample_my_sample <- function(v, p, q) {
  n <- ceiling(max(p * length(v), q))
  do_replace <- (n > length(v))
  sample(v, n, do_replace)
}

for (i in 1:10) {
  print(sample_my_sample(i:10, 0.5, 3))
}
## [1]  3  7 10  1  2
## [1] 5 9 8 4 3
## [1] 5 7 6 9
## [1]  8 10  9  7
## [1] 7 6 8
## [1] 9 8 7
## [1] 9 7 8
## [1]  8 10  9
## [1]  9 10 10
## [1] 6 8 8
```

Code meets data

```
read_csv("../raw_data/stock_price.csv") %>%
  ggplot(aes(x = date, y = adj_prc)) +
  geom_line() + theme_minimal()
```



Calculate weekly mean return and weekly return variance

```
read_csv("../raw_data/stock_price.csv") %>%
  mutate(return = (adj_prc - lag(adj_prc))/lag(adj_prc)) %>%
  summarise(mn_return = mean(return, na.rm = TRUE),
            var_return = sd(return, na.rm = TRUE)^2)
## # A tibble: 1 x 2
##   mn_return var_return
##       <dbl>      <dbl>
## 1  0.000467  0.0000262

## Correct values
## # A tibble: 1 x 2
##   mn_return var_return
##       <dbl>      <dbl>
## 1  0.000434  0.0000247
```