Take Advantage of Script Programming Languages:

Rees Morrison

2023-01-05

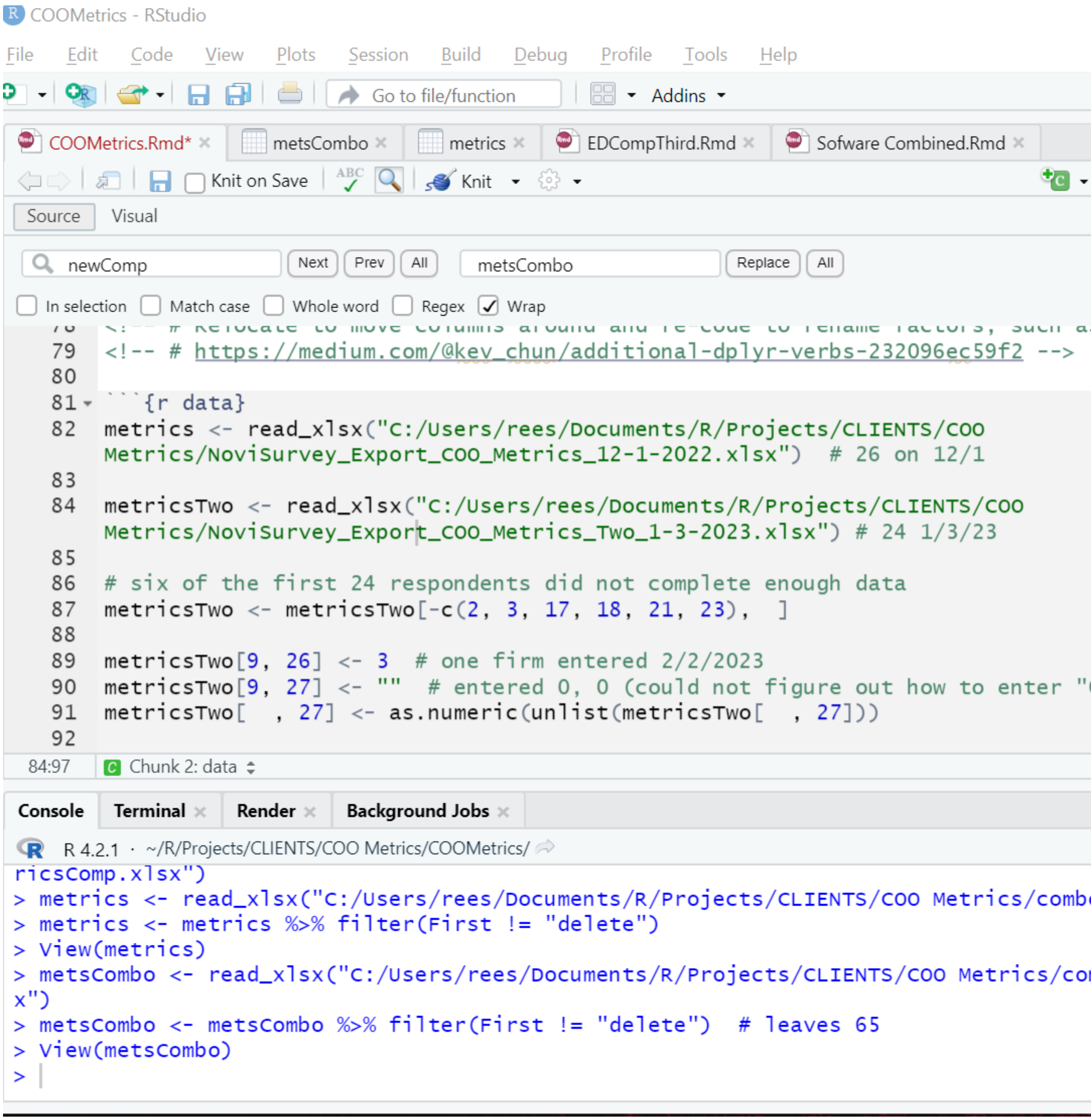
A “script” is the term I will use for the programming an analyst does to turn raw survey responses into groomed output, such as tables, plots, or complete reports. Scripts have huge advantages over spreadsheet manipulations.

Reproducible: The small chunk of R code shown below reads in the Excel file that was exported from my **hosting software**, renames several variables (column headers in Excel) to make them more tractable, and drops a few columns that won’t be needed for the next step of the analysis. If you change a header in a spreadsheet file, that change does not carry over to the next iteration of the file you download. Whenever the next export of respondent data occurs, all I do is rename the Excel file in the first line of code (change the date and off we go!).

Preserves data. Note that unlike with a spreadsheet, your script only makes changes to the object stored in the computer’s memory; the original data in the exported spreadsheet remains untouched. With spreadsheets, if you correct a compensation figure from 5000 to 50000, that change alters the original (unless you copy the original and work on a second version). Perhaps techniques exist to redline changes, but you still must make them all again (correctly!) on subsequent versions.

Comments: Good programmers write comments to explain what their code does. In the R programming language, comments are preceded by a hash sign (“# “). The computer ignores comments when it runs the code chunk. Comments can also cover error messages (shown on your console when code does not run) so that your learning accumulates.

Searchable: Programmers use integrated development environments (IDEs) to make life much easier than command line programming. Below is what I see when programming in R, where I make heavy use of the RStudio (now Posit) IDE. Of inestimable value to me is the ability to have the IDE search in my past scripts for an example of code. So, if I forget how to change the position of the legend on a plot, I can search for a previous time when I did exactly that.



~~~
<!-- # https://medium.com/@kev_chun/additional-dplyr-verbs-232096ec59f2 -->

```{r data}
metrics <- read_xlsx("C:/Users/rees/Documents/R/Projects/CLIENTS/COO Metrics/NoviSurvey_Export_COO_Metrics_12-1-2022.xlsx")  # 26 on 12/1

metricsTwo <- read_xlsx("C:/Users/rees/Documents/R/Projects/CLIENTS/COO Metrics/NoviSurvey_Export_COO_Metrics_Two_1-3-2023.xlsx") # 24 1/3/23

# six of the first 24 respondents did not complete enough data
metricsTwo <- metricsTwo[-c(2, 3, 17, 18, 21, 23),  ]

metricsTwo[9, 26] <- 3  # one firm entered 2/2/2023
metricsTwo[9, 27] <- ""  # entered 0, 0 (could not figure out how to enter "
metricsTwo[  , 27] <- as.numeric(unlist(metricsTwo[  , 27]))
~~~

```
ricsComp.xlsx")
> metrics <- read_xlsx("C:/Users/rees/Documents/R/Projects/CLIENTS/COO Metrics/comb
> metrics <- metrics %>% filter(First != "delete")
> View(metrics)
> metsCombo <- read_xlsx("C:/Users/rees/Documents/R/Projects/CLIENTS/COO Metrics/co
x")
> metsCombo <- metsCombo %>% filter(First != "delete")  # leaves 65
> View(metsCombo)
> 
```

Fast: Interpreted code (such as R and Python) running on today's personal computers completes its work in seconds. Spreadsheets may be as fast for certain computations and manipulations, but overall, survey data analyses and report generation by means of scripts has to be the fastest method.

Trustworthy: Once the code runs to your satisfaction, unless you make a change somewhere you can absolutely rely on it. From this comes the adage, DRY ("Don't repeat yourself!"); write the code once and then reuse it.

Debugging: With scripts of code, you can zero in on why a portion is not producing what you want. If my code shows “NAs” when I tell it to calculate medians, I can figure out that one value is missing, or that another value is in character format. You can step through your code, disassemble it, try a new statement, and figure out where you made a mistake (the computer did not!). While debugging, you can run small chunks of code.

CoPilot (ChatGPT): A powerful tool from GitHib supplements the coding skills of an analyst. For a small monthly fee, if you are using certain IDEs (notably, several for Python, but none yet for R), CoPilot will autocomplete partial code you write. If you type “subset my data by firm size and number of offices,” CoPilot will write the full, elegant statement for you! For fundamental tasks of data wrangling, CoPilot will serve you with dramatic ease.

Builds cumulatively: As you program more, you can draw on the code that precedes what you are currently writing, and expand from there. I am not aware of a similarly reliable and cumulative capability in spreadsheets.