Winter storm Rmarkdown session

Meg Cychosz

1/12/2021

Introduction

There are three components to any RMarkdown file:

- Code chunks where you do the bulk of your data processing
- Prose to walk the reader through your analyses
- Header to format your output file

Why RMarkdown?

- Facilitates clean, organized code
- Allows dynamic prose to accompany your results that updates when analyses change, addtl. data is added, etc.
- Flexibility in terms of which components of the script are run, saved, or ignored
- Permits integration of additional computing languages and environments like the command line and Python
- Generates attractive reports in .tex, .pdf, .html, and .doc formats

Let's get started!

File parameters

- RMarkdown files have a .Rmd extension to differentiate them from traditional R scripts
- You can specify several parameters in the header of your file
  - the output file format
  - parameters of your output file, like whether or not you want a table of contents
  - Let's play!

Code chunks

- Code chunks are where the magic happens
- They allow you to run little bits of your code instead of entire scripts at once
- Great for debugging!
- Chunks are surrounded by ticks ```
- Give your chunk a name to find easily find it in large scripts
- Then you specify some parameters for your chunk
  - the language (R is default; more on this later)
  - do you want warnings? messages? ignore the chunk completely?
- Then you run individual chunks by clicking the green button

```
# after this chunk, I don't want messages or warnings displayed
# what happens if I set message=TRUE?
library('dplyr')
```

```
library('ggplot2')
library('kableExtra')
library('lme4')
```

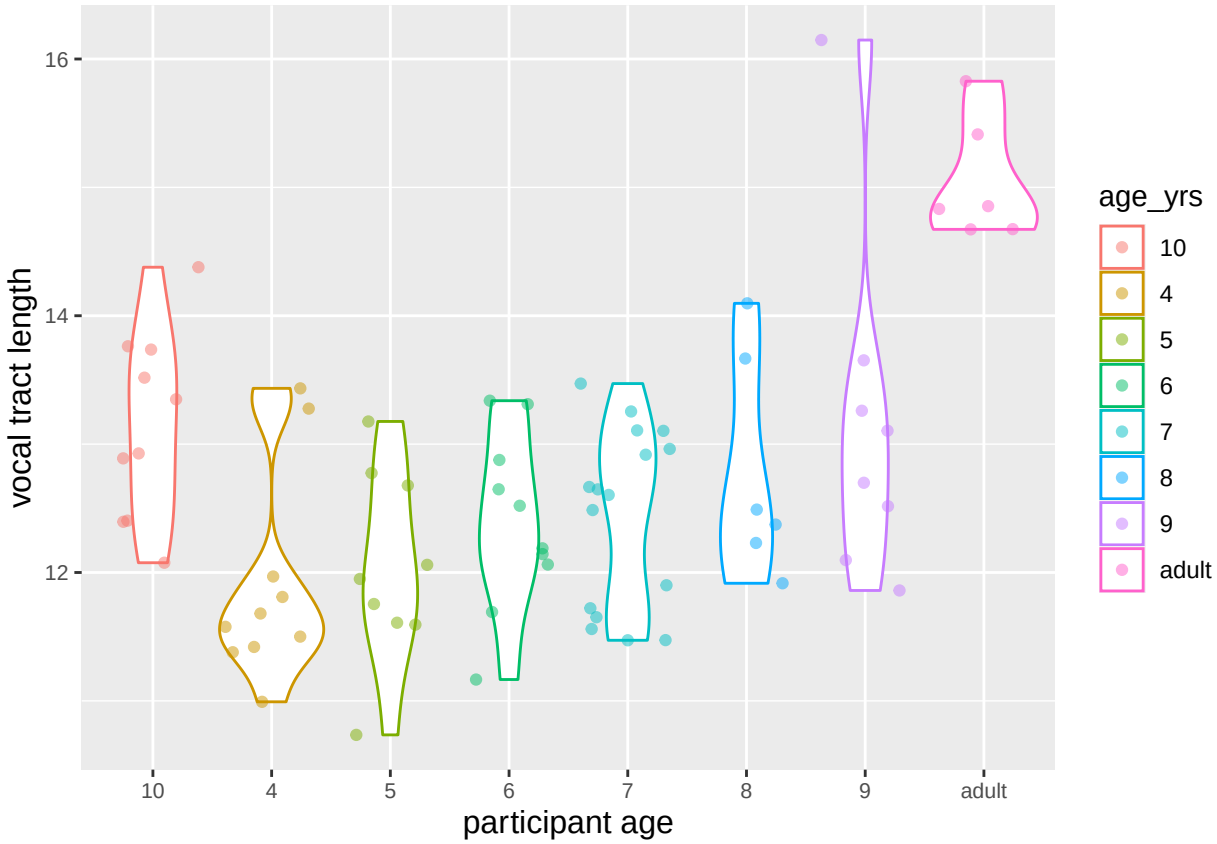

Figure 1: Vocal tract length as a function of participant age

- If you don't want to specify the same parameters for each chunk, just set them in the global options at the beginning of the file

Plots and Tables

- You can generate nice tables and plots in your report
- By saving the file as .tex or .doc, you have an editable table to import into your paper
  - No more copying and pasting into .tex and Word tables
  - Or, better yet, write your whole paper in RMarkdown!
  - A separate directory containing your figures is also generated
  - Let's make a table
- There are tons of styling options to ready your table for publication/submission
  - kable is a powerful library to generate tables
  - Specify the column widths
  - Add a caption
  - Change the table style
  - Add a header above
  - Group rows together
- Now you try! Play around with the data entered into the table:
  - Try filtering different age groups in and out

Table 1: Formant frequency measurements by participant age

Age	F1 mean (SD)	F2 mean (SD)
4	551.81 (325)	1942.63 (816)
5	520.49 (281)	1982.17 (758)
6	475.19 (319)	1915.05 (823)
7	493.8 (271)	2011.94 (809)
8	426.25 (288)	2049.45 (755)
9	420.7 (268)	1948.24 (754)

  - Try calculating different summary statistics (mean, range)
  - Try calculating statistics over vtl instead of formant measurements

Prose

Now that you have plotted some data, conducted summary statistics, and created a table, it's time to write up the results. First let's talk about the basics of Markdown.

Basics of markdown

- We can make headings of different sizes like we've been doing throughout this document.
  - Try adding and deleting # from 'Basics of markdown' to watch the font size change
- We can make bullet lists like the one you're reading right now!
  - And sublists!

We can write in paragraphs.

We can write in multiple paragraphs, in fact, just be leaving a line between them in the document and two spaces after the previous line.

We can **emphasize** text while we're writing in *several* different ways that would ***surprise*** you.

You can also embed links to direct your readers elsewhere like this one. Do you know where you can change the color of your hyperlink?

There are several Markdown cheatsheets available to help you remember this syntax. Try to memorize one new markdown feature each time you sit down to analyze data and you'll have a large repertoire in no time.

In-line coding

Now that you know some basics of Markdown, you can write up your results!

```
young_children <- data %>%
  filter(age_yrs=='4' | age_yrs=='5')
```

One of the best features of RMarkdown is dynamic coding. Essentially, you include tiny little chunks of code in your prose. For example, you may want to reference descriptive statistics or model summary statistics. You surround each in-line chunk with backwards apostrophes and specify the language. You could write up the results saying that the average value of the first formant in the four- and five-year-olds is 651.27. Or maybe you just want to report that there are n=19 four- and five-year-olds in the study. You can also cross-reference figures and tables within your in-line code.

```
# let's correlate some vocal tract length with age
vtl_cor <- data %>% # this produces an object;
  filter(age_yrs!='adult') %>% # we can index the parameters of the statistic in our prose
```

```
  mutate(age_yrs = as.numeric(age_yrs)) %>%
  distinct(spkr, .keep_all = T) %>%
  summarize(., cor(age_yrs, vtl))
```

You could also reference components of your statistical tests like reporting that there is a positive correlation between age and vocal tract length (whew, sanity check!) (r=0.48). This means that even if (ahem) forget a participant or two in your first analysis, you don't have to rewrite your entire results section to include the new participants. It's updated automatically every time you compile your script.

```
# fit a model and create an object containing its summary
vtl_model <- data %>%
  filter(age_yrs!='adult') %>%
  mutate(age_yrs = as.numeric(age_yrs)) %>%
  lmer(f1_midpt_med_clean~age_yrs + (1|spkr), data = .) %>% # using age to predict F1
  summary()
```

You can also create and reference model fit summaries. For example, with the object you created above, you can can report that there is a negative relationship between child age and the frequency of the first formant (β=-23.07, t=-4.36).

Integrating other languages

Finally, you have the option to sew snippets of code from other languages right into your script! Even if you're not super familiar with shell scripting or Python, this could still be helpful. Imagine that a colleague sends you a Python script, or you find a few lines of a shell code online. Now instead of translating it into R, you could copy it verbatim.

Let's look at a bash example because Shell scripts are often simple, short, and easily found on the internet. We're going to trim an audio file.

```
# in your terminal below, you will need to install 'sox' in order to run this line of code
# either 'brew install sox' or 'pip install sox'
sox audio_file.mp3 chopped_file.mp3 trim 30
```

```
## sox WARN mp3: MAD lost sync
```

Why might you want to include snippets of code from other languages in your .Rmd file instead of just saving them as extra scripts? 1) Keeping them all in one place helps you stay organized. 2) When you find a bug in your code, you just have one script to re-run, instead of multiple, leading to fewer human errors.

You could also execute python code, which requires a bit more finagling. Sewing python requires install of 'reticulate', in addition to setting the python environment, and python modules (see below)

```
library(reticulate) # note: you do NOT want to run reticulate via r-miniconda; do not install it when p
use_python("/usr/bin/python")
```

Then you can execute your Python code chunks and can easily switch back to R syntax once you've finished.