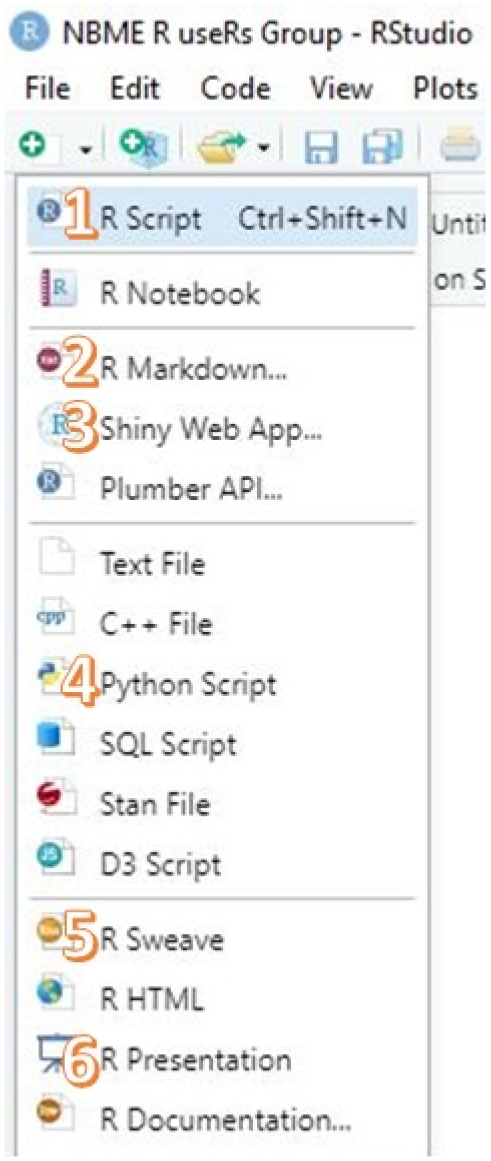
“there is art to data science”

NBME useRs

VOL.I...No.2 *NOVEMBER 30, 2020* PRICELESS

Unpacking RStudio

RStudio is a versatile IDE. While its most common use is for R scripts, several different types of products / files can be created from this single interface. The associated image shows what options are listed when “create new file” (white page with green circle and white plus) is clicked. Your menu may differ from mine; some options depend on having other packages installed (e.g., RStan). To start this newsletter I’ll cover 6 of most common ways that I use RStudio.



1. **R Scripts (.R)** This is a plain script file that will be used to carry out most R commands. It is most like the regular SPSS and SAS script files (I think - it’s been a while since I’ve written those file types). You can also save user-created functions in script files and call them from other R files, just like macro files in other languages.

2. **Rmarkdown (.Rmd)** This file type is a blend of R and LaTeX or html syntax, so you can integrate R chunks and text chunks in a single .pdf or .html file. This is a popular format for creating automated documents, and it is used to create this newsletter!

3. **Shiny Apps** are interactive applications that ‘sit’ on the R programming language. It allows users to complete different tasks without having any knowledge of R syntax. Shiny Apps can be hosted on internal servers or sourced to run on the internet where they can be accessed by any user with the address.

4. **Python Scripts** The most recent versions of RStudio now allow users to write and execute Python syntax. It requires a bit more setup in order to have this functionality, but given the recent interest in the Python language at NBME this is of note.

5. **R Sweave (.Rnw)** These files are similar to Rmarkdown files, but are closer to pure LaTeX files. Rmarkdown files are more structured files that have more preset stylistic defaults, whereas R Sweave files require explicit syntax to set many of the defaults.

6. **R Presentation** It's also possible to create PowerPoint-like presentations with R. Doing so provides the same advantages of making documents or reports in Rmarkdown: you can incorporate / show chunks of R syntax in the presentation; you can seamlessly integrate output from analyses in the presentation (such as graphs and tables); and, much like using LaTeX, there are options to modify almost all of the visual aspects of the presentation through syntax commands.

Learning Resources

A Tip When Using Older R Books

When starting to learn and use R, one natural source of guidance can come from textbooks, like those at Bookdown.org that were mentioned in the first newsletter. These books are undoubtedly a valuable resource, but the R programming language is constantly evolving, becoming more robust and user-friendly as time goes by. New packages are frequently introduced, and even old standard packages sometimes go through significant changes to improve / extend their functionality (such as when a new version of the popular data-wrangling package dplyr came out last year). The basic logic and structure of the R language has remained the same over the years, so much of what you read will still apply, but I encourage the reader to do a little research on what they're learning to see if an easier or more efficient method has been developed.

The R Inferno

Dan Jurich was kind enough to share this humorous take on Dante's Inferno that covers some common misconceptions and errors that new (and experienced) R users make. This document is a few years old, so some of the pain points might have been addressed by now (see above), but the majority of the document is widely applicable.

Solving Soduku Puzzles with R

One of the fun topics mentioned in a recent R-Bloggers newsletter was an entry on how to solve Sudoku puzzles with R. I know that part of the fun with Sudoku is working through the possibilities and coming up with a problem-solving strategy yourself, but one of most challenging and fun aspects of programming is learning how to translate these rules into a formal language. The process of translating thoughts into a formal language forces the idea to be written simply and clearly, and doing so helps you better understand *exactly* what you're thinking. This translation process helps you better understand the problem you're trying to solve and also allows you correct any mistakes in thought.

Contribution idea: What other game-like uses for R can you think of? Do any packages exist that help with figuring out the optimal checkers or chess move? What other games could be solved by writing a program in R? (The idea generalizes to all programming languages, but this is an R newsletter.) Have an idea or found something neat online? Email me at CRunyon@nbme.org and I'll include your contribution in an upcoming newsletter.

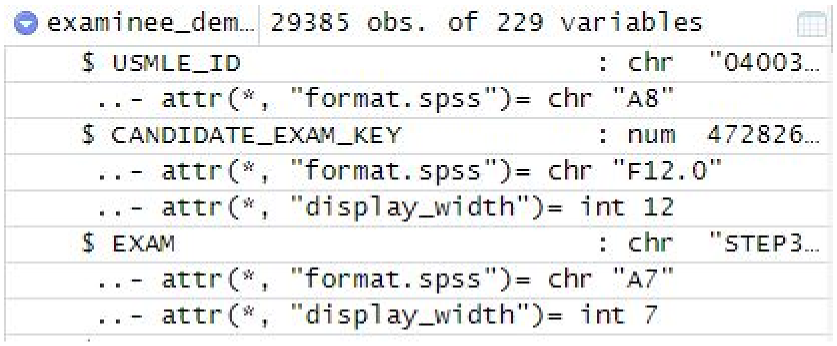
Featured R packages

This newsletter will feature two packages that will be practically useful to those persons who will be using files that were created using other statistical computing packages[1], `haven` and `foreign`. Knowledge of these packages will be especially helpful as NBME moves forward with changing over to from SPSS to R and Python. `haven` and `foreign` aren't the only packages available to import or read files from other statistical packages, but they are the most popular. The packages differ a bit in their speed and full functionality. More details below.

R package: `haven`

haven pdf vignette
Read and write various data formats used by other statistical packages.
Suggested audience: Anyone who needs to read in data from other statistical programs (e.g. SAS, SPSS, Stata).

Of these two packages, I tend to use `haven` more often because (1) I'm most often reading in SPSS or SAS files; (2) I like the different `zap_` functions that are available; and (3) **haven can read in the sas .sas7bdat files without having SAS installed (foreign requires a SAS installation on your system)**. When reading in an SPSS file, R takes care to ensure that all of the file metadata is preserved. Take the image below for example.



```
examinee_dem… 29385 obs. of 229 variables
    $ USMLE_ID                    : chr  "04003…
     ..- attr(*, "format.spss")= chr "A8"
    $ CANDIDATE_EXAM_KEY          : num  472826…
     ..- attr(*, "format.spss")= chr "F12.0"
     ..- attr(*, "display_width")= int 12
    $ EXAM                        : chr  "STEP3…
     ..- attr(*, "format.spss")= chr "A7"
     ..- attr(*, "display_width")= int 7
```

The image shows the formatting of the data frame for an SPSS file that I read into R. As you can see, in addition to the basic information (the data itself), there are metadata attributes that have been preserved from the SPSS file. These attributes can be useful in working across programming language platforms to ensure compatibility and other logistical matters, but if you're just doing analyses this metadata is often unnecessary. These attributes make file sizes much larger, and thus they're slower to load and manipulate. The `zap_formats` and `zap_widths` functions can remove this unnecessary metadata.

[1] Thanks to Dan Jurich for the idea to feature these packages

R package: `foreign`

‘foreign‘ pdf
Read and write various data formats used by other statistical packages.
Suggested audience: Anyone who needs to read in data from other statistical programs (e.g. SAS, SPSS, Stata, Weka, etc).

While `haven` is now considered the standard for importing SPSS / SAS files, `foreign` was the main package used for importing data files from other statistical programming languages prior to the release of `haven`. I mention `foreign` here because (1) it’s likely you’ll come across the package in an R book / online course; and (2) `foreign` has the ability to import / read file types that aren’t available in `haven`, so it’s a good backup for when `haven` can’t do what you need it to do.

Comparing Import Speeds

Using the `rbenchmark` package, I will now compare the speeds of the `haven` and `foreign` packages to import SPSS data files. This process is replicated 100 times and the reported times are the sum of these replications.

Two files are used in this demonstration. The first file is a **large** (48.1 MB) public-use data file downloaded from the National Center for Education Statistics website. Specifically, this file contains data from the National Household Education Survey of 2016. The second file is from a data pull for a recent project, which will probably more closely resemble tasks that we will do at the board (file size = 9.7 MB). For ease of readability, I’ve saved the file paths to objects named `big_spss_file` and `small_spss_file`.

```r
library(rbenchmark); library(haven); library(foreign);

benchmark("foreign_large" = {read.spss(big_spss_file, to.data.frame = TRUE)},
          "haven_large" = {read_sav(big_spss_file)},
          columns = c("test", "elapsed", "relative", "user.self", "sys.self"))
```

```
##            test elapsed relative user.self sys.self
## 1 foreign_large 1387.75    5.789   1316.83    12.19
## 2   haven_large  239.72    1.000    230.17     6.11
```

```r
benchmark("foreign_small" = {read.spss(small_spss_file, to.data.frame = TRUE)},
          "haven_small" = {read_sav(small_spss_file)},
          columns = c("test", "elapsed", "relative", "user.self", "sys.self"))
```

```
##            test elapsed relative user.self sys.self
## 1 foreign_small  154.63    2.319    144.27     2.09
## 2   haven_small   66.67    1.000     35.72     1.61
```

As can be seen on the previous page, there are obvious speed advantages when using the `haven` package, and this advantage increases for larger files. Thus, it seems as if `haven` should be your first choice for reading in files from other statistical software, but keep `foreign` in your back pocket for those times when you need to read in a file that `haven` can't handle.

NHS-R Conference

A few weeks ago the NHS-R Conference concluded. Fortunately the Hutsons-Hacks website has posted recordings of many of the talks given at the conference, which includes the "Introduction to dplyr and data wrangling" workshop, which could be of interest to new R users (and me too, given the recent dplyr update).

Another workshop that caught my eye was "Pretty R Markdown and Presentations with Xaringan" because I'd never heard of Xaringan before. It turns out that Xaringan is a presentation format (think Powerpoint or Beamer) that is built with remark.js, and is targeted toward people that know "HTML and CSS." I know a *little* about these things, but look forward to learning more.

I did some digging on the NHS-R website and found some links to other workshops that might be of interest to some of you. Topics include "Database Connections in R", "Advanced Modeling Supervised ML", "Functional Programming with the Purrr Package", "Shiny", and "Task Automation with R".

Hex Logos

As you may have seen by browsing different R-related sites, a common thing to do in the R community is to create a hex-shaped logo that advertises your R package, website, conference, group, etc. If you attend an R conference or attend an R talk by one of the RStudio developers, there will most likely be relevant R hex stickers available for you to proudly decorate your laptop. **The NBME useRs group needs to have its own R hex sticker!** It's a fun way to indicate group membership and be proud of your R skills. Appropriately, there is an R package (hexSticker; pdf) to facilitate making new R hex stickers. Here's a gallery of examples to help inspire ideas.

Below are some of my attempts at making a hex sticker for the NBME useRs group. **I welcome your creative ideas and suggestions! Please design a sticker for us to use!** After we decide upon an "official" NBME useRs hex sticker, I'll gladly have them printed and send them to those members that would like one. In keeping with the R hex sticker tradition, final stickers are 2 inches tall so they all fit nicely together.