Creating the large multivariate dataset in R

Julian

Contents

1 Executive summary

The goal of this document:
- demonstrate various R commands
- how to potentially import and inspect a large dataset
- creating statistical models for large datasets
- how to author a report using Sweave or R Markdown

2 Steps

- create subject ids
- set a random seed (makes tracking what was done harder)
- randomly creating subject ages
- simulating subject responses
- generating reaction times and introducing real data
- assigning attempts
- assigning subjects to groups
- adding trial, token phase data in long format

3 Execution

This document was created in R Markdown. All the data were generated using R code. To execute the commands, change the chunk options to `eval = TRUE`.

The source code (R) is contained in the Day 3 directory.

4 Data generation

4.1 Set random seed

```
RNGkind('Mersenne-Twister')
set.seed(2340875)
old.seed <- .Random.seed
```

4.2 Create `Subject`, `Age`, `Group` vectors

```
subj.nums <- seq(1, 102, 1)

subj.ids <- character(length(subj.nums))

for (i in 1:length(subj.nums)) {
  subj.ids[i] <- paste('subj-', subj.nums[i], sep = '')
}

subj.rep <- rep(subj.ids, 800)

groups <- list('treatment-1', 'treatment-2', 'control')
age.vec <- runif(min = 18, max = 52, n = length(subj.ids))
```

4.3 Assign subjects to an age

Replicates the ages to match the subject repetitions

```
age.rep <- rep(age.vec, 800)
```

4.4 Simulating correct responses

Much slower than generating vector (especially since R is making a deep copy each time), but harder to figure out what was done

```
correctness <- runif(min = 0, max  = 1, n = length(subj.rep))

correct.vec <- integer()

for (i in 1:length(subj.rep)) {
  correct.vec[i] <- rbinom(n = 1, prob = correctness[i], size = 1)
}
```

4.5 Generating reaction times

Once again, very slow but an extra hurdle to jump through

4.6 Replacing values selectively with real data

```
data.dir <- '~/GitHub/reproducible-research/Day-2/datasets'
data.file <- 'psycho-data-april-2015.csv'

real.data <- read.csv(file.path(data.dir, data.file), header = TRUE)

real.rt <- real.data$absRT

for (i in 1:length(subj.rep)) {
  if (rt.vec[i] > 600) {
    rt.vec[i] <- sample(real.rt, replace = TRUE, size = 1)
  } else {
    next # R's version of a continue statement
  }
}
```

4.7 Creating timed out values

```
for (i in 1:length(subj.rep)) {
  if (rt.vec[i] > 2200) {
    rt.vec[i] = 200
  } else {
    next
  }
}
```

Creating attempts vector Since there is no random integer function in base R, round numbers from a uniform distribution

```
attempt.vec <- round(runif(n = length(subj.rep), min = 1, max = 15))
```

4.8 Create token vector

```
token.rep <- rep(seq(1, 20, 1), each = 20)
token.vec <- rep(token.rep, 800)
```

4.9 Create experimental `data.frame`

Using `data.table` since it is faster

```
library(data.table)

sample.data$Subject <- subj.rep

sample.data <-
  data.table(Subject = subj.rep, Age = age.rep, Attempts = attempt.vec,
             Correct = correct.vec, RT = rt.vec)
```

4.10 Add in trials

For the trials, tokens, groups, and phases a new function will be introduced: `gl()`. This allows you to generate factor levels and it will be used to create the labels for each level. Generally, you would use this to create new factors with appropriate levels for statistical analysis, but this is a good way to introduce this function.
It should be noted that you should be careful with this function, as you will need to know the pattern for the labels you are creating.

The first step will be to sort the data into long format (well, the convention anyway). This makes it easier to see what is going on with the levels. You shold also take advantage of the viewer (`View()` function) in RStudio to explore the data structure.

4.11 Sorting data into proper long format

Easiest to sort the data using `dplyr` (though `plyr` would also work well).

```
library(dplyr)
sample.data <- arrange(sample.data, Subject)
```

4.12 Generate tokens

```
sample.data$Token <- gl(n = 20, k = 40, labels = seq(1, 20, 1))
```

4.13 Generate trials

```
sample.data$Trial <- gl(n = 20, k = 1, length = 20 * length(subj.ids))
```

4.14 Generate phases

```
sample.data$Phase <- gl(n = 2, k = 400, labels = c(1, 2))
```

4.15 Generate groups

The groups will not be generated in the same manner as was done in Python. I could have mapped values as was done previously, but instead I will just assign groups using `rep()`.

```
treat.1 <- rep(c('treatmet-1'), 34 * 800)
treat.2 <- rep(c('treatment-2'), 34 * 800)
control <- rep(c('control'), 34 * 800)

group.vec <- c(treat.1, treat.2, control)

sample.data$Group <- group.vec
```

4.16 Changing data to appropriate types

This is not really necessary, but we can change the columns `Group` and `Subject` to factors - they would be coerced anyway if a model were to be run. The numeric columns could also be coerced into factors (or integers), but this is likewise not really necessary. This is mentioned elsewhere, but use `methods(as)` to see how to manilulate data types. The packages `stringr`, `stringi`, `lubdridate` and others also have methods to convert data types.

```
sample.data$Subject <- as.factor(sample.data$Subject)

sample.data$Group <- as.factor(sample.data$Group)
```

4.17 Export to CSV

This is pretty self-explanatory; use the documentation to see all of the options. Depending on needs, the `readr`, `xlsx` and other packages have the option to output to a variety of different formats.

5 Appendix: vectorized functions

The data were created in a rather slow, tedious manner, mostly to introduce control flow and to be explicit about the steps that were taken. This is further complicated by the fact that R handles data in a less than ideal way sometimes, especially with making deep copies on loops see here for more details.

Below are examples of using vectorized functions instead of `for` loops to manipulate data. There are also additional packages (such as `data.table`) that vectorize functions and work with large datasets.

5.1 Simulating correctness

Here we use the vector of probabilities and specify the size of the final vector to speed up computation:

```
correct.vec <- rbinom(n = 81600, prob = correctness, size = 1)
```

5.2 Simulating reaction time

Generating the reaction times is done in the same manner:

```
rt.vec <-
  rlnorm(n = 81600, meanlog = 6.0, sdlog = 0.5) + 5 +
    runif(n = 81600, min = 10, max = 100) +
    rnorm(n = 81600, mean = 70, sd = 15)
```

5.3 Replacing simulated values with real values

There are several options here that we could go with. We will use base R functions for indexing, replacement and sampling.

```
gr.600 <- which(rt.vec > 600)

real.sample <- sample(real.rt, replace = TRUE, size = length(gr.600))

rt.vec <- replace(rt.vec, gr.600, real.sample)
```

5.4 Creating timed out values

Follow the same procedure as the above to create the timed out values:

```
gr.2200 <- which(rt.vec > 2200)

rt.vec <- replace(rt.vec, gr.2200, 2200)
```