Alternatives to Classic Loop Structures in R

Olusoji Oluwafemi Daniel

9 January 2019

Some Basics

- Repeating a process/task is part of our daily life and sciences isn't left out of this.
- Repeating a process in programming languages is referred to as looping, which forms one of the basic thing to learn when learning a new language
- The standard function for repeatition in programming are; `repeat`, `do`, `for` and `while`
- In `R`, this classic loop structures are not as fast as they are in other High Level Languages, hence the need for alternatives.
- However, they are not as slow as people make them seem.

The Apply set of Functions

- The `Apply` set of functions are the first set of alternatives to loops in `R`
- There are about 4 to 5 of then (including a multicore version, `mclapply` in the *parallel package*)
- `?apply`, `?lapply` and `?sapply` opens up the help pages in `R`
- `apply(x, MARGIN, FUN, ...)` allows you **apply** a function over the rows **(MARGIN = 1)** or columns **(MARGIN = 2)** of a matrix or dataframe
- `sapply(x, FUN, ...)` and `lapply(x, FUN, ...)` allows you **apply** a function over a list, matix or dataframe as well. `lapply()` always returns a list, `sapply()` tries to simplify the results.

Tidyverse Offering (purrr Package)

- *tidyverse* is an ecosystem of packages dedicated to making data analysis faster https://tidyverse.org/
- It thrives on a set of data structures, grammar and philosophy aimed at making data analysis faster and reproducible.
- *purrr*'s `map(.x, .f, ...)`, its variants and `walk(.x, .f, ...)` function makes it possible to apply a function that allows more than one input to a list, array or dataframe.
- `?purrr::map()` gives you the relating help pages.

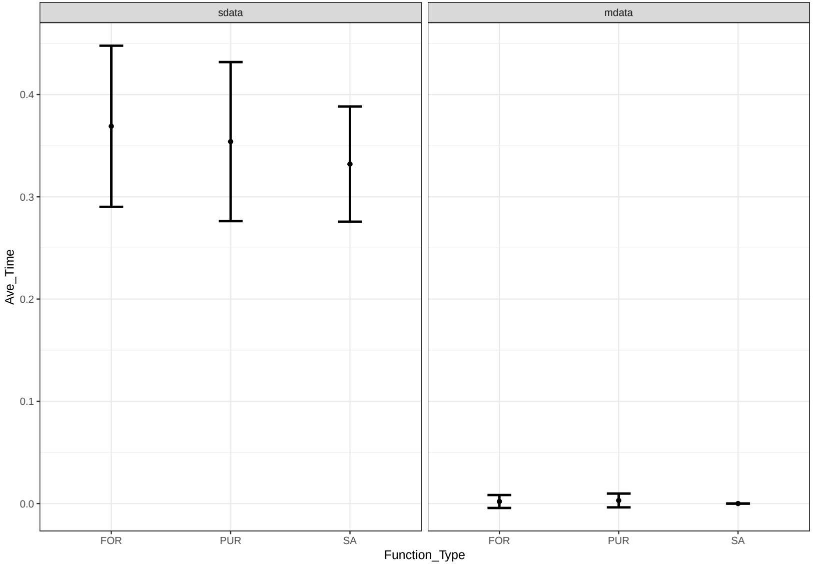
An Hello World Example

- Let's start with a toy example
- Bootstrapping and Determinants

```r
#bootstrap function
boot_func <- function(dat) {
  # sample from data
  sample(dat, length(dat), replace = T)
  # mean and standard deviation of dat
  return(mean(dat))
}
sdata <- rnorm(80000, 0.5, 1.5)
# 3 X 3 matrices from uniform distribution
mdata <- vector("list", 80000)
mdata<- lapply(mdata, function(x) {
  matrix(runif(15, 0, 1), nrow = 3, ncol = 3)
})
```

- To be used `for`, `sapply`, `purrr::rerun()` & `purrr::map()`

How They fared



- Considerably less time for the determinants, why?

A Model Fitting Example

- The chicken dataset contains birth information of 628 chickens obtained via in-breeding or cross-bredding.
- Let's try to fit a regression model to relate the Age and Birth Weight of the 628 chickens.

```r
model_func <- function(ddata) {
  if( length(which(!is.na(ddata$BW))) >= 3 ) {
   model  <- lm(BW ~ AGE, data = ddata)
   slope <- coef(model)[2]
  } else {
    slope <- NA
  }
  return(slope)
}
```

Using for() loop

```r
slopes <- c()
system.time( for(i in 1:length(bychick$data)) {
  result <- model_func(ddata = bychick$data[[i]])
  slopes <- c(slopes, result)
})
>    user  system elapsed
>    0.83    0.00    1.06
```

Using lapply()

```
system.time(lapply(bychick$data, model_func))
>    user  system elapsed
>    0.72    0.00    0.72
```

Using purrr::map()

```
system.time(map(bychick$data, model_func))
>    user  system elapsed
>    0.85    0.00    0.97
```

Why Use these Alternatives?

- Cleaner and more readable codes
- Cleaner environment
- Dedicated outputs

Still to Come

- Introduction to Parallel Programming in R
- Tidyverse, A language within a Language
- Check https://github.com/fomotis/AvoidingLoops for updates on this tutorial and associated files