Paper SS02

A Beginner's Roadmap to Map Functions for Clinical Reporting in R

Samuel Curlee, Pfizer Inc., Collegeville, PA
Gabriela Piasecki, Pfizer Inc., Collegeville, PA
Xiaoqing Tang, Pfizer Inc., Collegeville, PA
Wenjian Xu, Pfizer Inc., Collegeville, PA

ABSTRACT

As in many other programming languages, the for loop in R is widely known as a means of providing iterations. An alternative, and perhaps stronger option, is to use the map functions from the R package purrr. As beginning or intermediate R users may not be familiar with this family of functions, this paper seeks to demonstrate its usage, specifically its ability to create commonly produced clinical study report (CSR) tables in clinical trials. The pharmaceutical industry continues to incorporate R to its standards; therefore, providing new and unique approaches to making such tables, listings, and figures (TLFs) is both forward-thinking and dynamic. In this paper, we explore making a demographic characteristics table with the help of the map functions in R.

INTRODUCTION

There are multiple ways to iterate over items in an object in R. The for loop is likely the most well-known option, although the map functions in the purrr package offer a different approach with good advantages. The map functions work such that they apply a function to each element of a list or atomic vector and return an object of the same length as the input. These functions accomplish this with efficiency and consistency, enabling us to generate clinical trial reports with compact code. We will demonstrate this by using it to generate a demographic characteristics summary table.

FOR LOOP

A for loop is commonly used in programming as it allows for code to be executed repeatedly. Not only does it simplify complex problems into simple ones, but it enables us to adjust the flow of a program such that we control the number of times it is executed. The basic syntax of for loop in R is as follows:

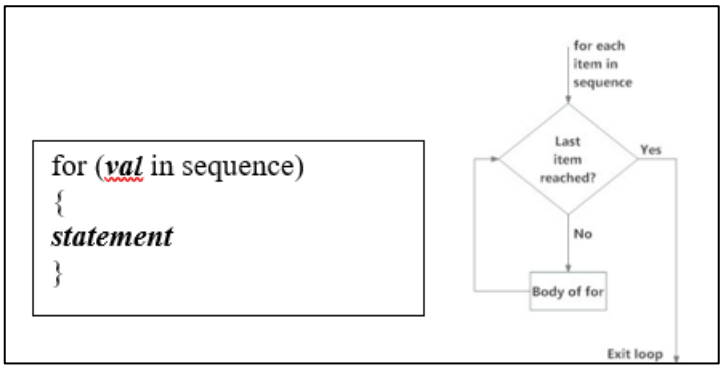

Figure 1. Operation of for loop (image taken from https://www.datamentor.io/r-programming/for-loop/)

where sequence is a vector and ***val*** takes on each of its values during the loop. In each iteration, ***statement*** is evaluated.

A for loop is one of the most basic ways to run through an array regardless of its length. It runs the code inside of a loop over and over until the entire list "sequence" has been iterated over. The syntax of the procedure is straightforward, and if the item number in the vector or list is small, the lag time of the loop is negligible. One can insert keywords such as *break* and *next* to stop the loop before it has gone through all items, and skip an iteration without terminating the loop, respectively. Conditional logic can also be added with an *if/else* statement.

However, when using for loops in R beyond trivial tasks, they often become a bottleneck of scripts. A proposed solution is to allocate the memory for the output object; R just needs to fill in the cells in a vector instead of extending the vector every loop cycle. We then look to identify if there are any R packages that may help with this task.

MAP FUNCTIONS

The map() is the most fundamental function in the purrr package. It takes a vector and a function as arguments, applies the function to each element of the vector, and returns the results in a list (Figure 2). It has for loops "embedded" in itself, and not only do we not need to manage an index variable, but it also provides immutability, meaning any action performed on an array creates a new array and leaves the original as is. It automatically allocates space for the output and doesn't update input data unexpectedly, which allows us to utilize our original array whenever we choose to. Using this function typically requires less code than the for loop.

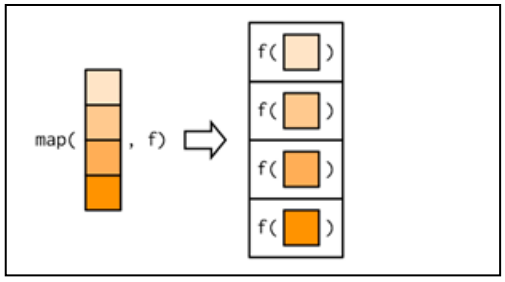

Figure 2. The map function scheme illustration (image taken from https://adv-r.hadley.nz/functionals.html)

The authors would like to point out that the base R package has a Map() with capital M. It is a wrapper of mapply which iterates through multiples elements of each argument and applies a function. The map() in purrr is more similar to lapply in base R. The formats of Map() and map() are different. In this paper, map() refers to the one in the purrr package.

The following example looks at different ways to calculate the means of a list of randomly generated samples, using for loops and map(), respectively. The two outputs show the exact same results therefore we only list one for illustration. The main advantage of map() is its clarity. It is more concise and easier to read.

```
library(purrr)

#Create a list of random vectors
set.seed(100)
list_of_samples <- (replicate(5, sample(1:100, 5, replace=TRUE),
simplify=FALSE) )

Method 1:
#Get the mean of each vector in the list using a For loop
list_of_means <- list()
for (i in seq_along(list_of_samples)) {
  list_of_means[[i]] <- mean(list_of_samples[[i]])
}

Method 2:
#Get the mean of each vector in the list using map()
list_of_means2 <- map(list_of_samples, mean)
```

[[1]]	double [1]	70
[[2]]	double [1]	59.4
[[3]]	double [1]	38.8
[[4]]	double [1]	59
[[5]]	double [1]	32.8

Figure 3. Output of calculation of means of a list of vectors using for loop and map()

Parts of the for loop and map() function call that correspond to each other have been highlighted in the same color to help illustrate how the two are similar (Figure 3). The output list, which had to be manually created in the for loop example, is naturally created as part of the map() function call and can be assigned to a variable. Instead of having to set an index variable for the for loop to iterate over the elements of the input list, we can just provide the input list as

an argument to map() and the function will automatically iterate in a similar way. The function that is applied to each element in the for loop, mean() in this instance, is passed as the second argument to map() and similarly applied to each element of the first argument.

There are variations of map() which allow us to apply single or multiple arguments functions and return different types of output. This paper attempts to illustrate some basic ideas and will only use map() and map_dfr() for demonstration. In purrr, suffixes, for example _dfr(), specify the structure of the output.

map(.x, .f, ...) – returns a list the same length as .x.
map_dfr(.x, .f, ..., .id = NULL) – returns a data frame.

... – a list of arguments passed on to the Map functions.
.x – a list or atomic vector.
.f– a function, formula, or vector.
.id– either a string or NULL.

In the next section, we will describe how to implement map_dfr() to create a clinical summary table.

MAP_DFR() TO GENERATE CLINICAL STUDY REPORT (CSR) TABLES

The clinical study report (CSR) is a comprehensive look at all the data produced in a clinical study, presented in text, tables, and figure (TLFs) formats. It often includes discussions and conclusions that provide context to the findings regarding the drug, biological product, device, or any other type of therapeutic product or practice under study. The report should include demographic and other potentially predictive characteristics of the study population and, when the study is large enough to permit this, present data for demographic and other subgroups so that possible differences in efficacy or safety can be identified.

The demographic characteristics tables in the CSR report are used to describe the demographic characteristics of the study population, compare various study groups to evaluate the balance, and aid in the interpretation of the efficacy and safety data. A sample demographic characteristics table mock shell is shown here:

Demographic Characteristics		
	Drug A (N=xxx)	**Drug B (N=xxx)**
Age (years)		
< 18	xx (xx.x)	xx (xx.x)
18 - 44	xx (xx.x)	xx (xx.x)
45 - 65	xx (xx.x)	xx (xx.x)
>65	xx (xx.x)	xx (xx.x)
Mean (SD)	xx.x (xx.xx)	xx.x (xx.xx)
Median (Range)	xx.x (xx, xx)	xx.x (xx, xx)
Gender		
Male	xx (xx.x)	xx (xx.x)
Female	xx (xx.x)	xx (xx.x)
Race		
White	xx (xx.x)	xx (xx.x)
Black	xx (xx.x)	xx (xx.x)
Asian	xx (xx.x)	xx (xx.x)
Other	xx (xx.x)	xx (xx.x)
Ethnicity		
Hispanic or Latino	xx (xx.x)	xx (xx.x)
Not Hispanic or Latino	xx (xx.x)	xx (xx.x)
Not reported	xx (xx.x)	xx (xx.x)

Table 1. Demographic characteristics table mock

To create the categorical analysis blocks of these demographic tables, the map functions are a good candidate as they make the code look concise and provide the flexibility to adapt to the changes based on needs. The variables age, sex, race, and ethnicity appear in almost all demographic tables; therefore, we will use them in our example.

Usually, only one ADSL dataset is needed. The simulated study population is shown below:

	usubjid	trtan	trta	age	agegr1n	agegr1	sexn	sex	race	racen	ethnic	ethnicn
1	ABC123-001	1	Drug_A	42	2	18-44	2	F	Black	2	Hispanic or Latino	1
2	ABC123-002	2	Drug_B	35	2	18-44	2	F	White	1	Hispanic or Latino	1
3	ABC123-003	1	Drug_A	37	2	18-44	1	M	Asian	3	Not Hispanic or Latino	2
4	ABC123-004	1	Drug_A	67	4	>65	2	F	Other	4	Not Hispanic or Latino	2
5	ABC123-005	2	Drug_B	17	1	<18	1	M	Black	2	Not Hispanic or Latino	2
6	ABC123-006	2	Drug_B	51	3	45-64	2	F	White	1	Not reported	3

Table 2. ADSL dataset

We need to define a set of factors for the final output. Sometimes, the formats shown in the dataset are not the same as what is wanted in the analysis table's output, i.e. "F" and "M" in the dataset vs "Female" and "Male" in the final table.

```
age_map <- c("1" = "< 18", "2" = "18 - 44", "3" = "45 - 64", "4" = "> 65")
race_map <- c("1" = "White", "2" = "Black", "3" = "Asian", "4" = "Other")
sex_map <- c("1" = "Male", "2" = "Female")
ethnic_map <- c("1" = "Hispanic or Latino", "2" = "Not Hispanic or Latino", "3" =
"Not reported")
```

The simplest method is to input a vector of variables and get a data frame created by row-binding. To do this, we use the map_dfr() which maintains the structure of the map functions conceptually.

```
npct <-  function(var) {
  adsl %>%
    group_by(trtan, trta, !!sym(var)) %>%
    summarise(n_pat = n()) %>%
    group_by(trtan, trta) %>%
    mutate(N_pat = sum(n_pat),
           pct = round(n_pat / N_pat*100,digits = 1),
           txt = paste0(n_pat, " (", pct, "%)"),
           trta = paste0(trta, "\\\n (N=", N_pat, ")")) %>%
    ungroup() %>%
    select( trta, !!sym(var), txt) %>%
    pivot_wider(values_from = txt, names_from = trta) %>%
    arrange(!!sym(var)) %>%
    mutate(Label = case_when(!!var == "agegr1n" ~ "Age (years)",
                             !!var == "sexn" ~ "Gender",
                             !!var == "racen" ~ "Race",
                             !!var == "ethnicn" ~ "Ethnicity"),
           Subgroup=paste0("    ", case_when(!!var == "agegr1n"~ age_map[!!sym(var)],
                                             !!var == "sexn"~ sex_map[!!sym(var)],
                                             !!var == "racen"~ race_map[!!sym(var)],
                                             !!var== "ethnicn" ~ ethnic_map[!!sym(var)])),
           order=case_when(!!var == "agegr1n" ~ 1,
                           !!var == "sexn" ~ 2,
                           !!var == "racen" ~ 3,
                           !!var == "ethnicn" ~ 4)) %>%
    select(Label, Subgroup, starts_with("Drug"), order) %>%
    replace(is.na(.), "0")
}

tbl <- map_dfr(c("agegr1n", "sexn", "racen", "ethnicn"), function(var) npct(var))
```

As shown above, one vector is used as input to simplify and illustrate the idea of using map functions for this type of table, and npct() is the function to be applied to each element in the vector. The first time npct() goes through the process, it uses ADSL to calculate the counts of the treatment groups as denominators. The subgroup counts come from the first element of the vector, as numerators, to compute the percentages of the subgroups with respect to the treatment group counts. After being transposed to a wide table with subgroups as labels, the first analysis block is generated. The process then sequentially goes through the rest of the elements in the vectors one at a time and produces the respective analysis blocks. Note that the numeric values of age, gender, race, and ethnic groups are used as input for easy row binding. Some special characters, i.e. "\\\n" are added for Knitr::kable to generate the proper html copy for reporting (Table 3).

Demographic Characteristics

	Drug A (N=3)	Drug B (N=3)
Age (years)		
< 18	0	1 (33.3%)
18 - 44	2 (66.7%)	1 (33.3%)
45 - 64	0	1 (33.3%)
> 65	1 (33.3%)	0
Mean(SD)	48.7 (16.10)	34.3 (17.00)
Median(Range)	42.0 (37, 67)	35.0 (17, 51)
Gender		
Male	1 (33.3%)	1 (33.3%)
Female	2 (66.7%)	2 (66.7%)
Race		
White	0	2 (66.7%)
Black	1 (33.3%)	1 (33.3%)
Asian	1 (33.3%)	0
Other	1 (33.3%)	0
Ethnicity		
Hispanic or Latino	1 (33.3%)	1 (33.3%)
Not Hispanic or Latino	2 (66.7%)	1 (33.3%)
Not reported	0	1 (33.3%)

Table 3: HTML output of demographic characteristics table

As the output shows, the npct() generates the desired count and percentages of each treatment arm in the study. The two descriptive statistics of age are produced separately. The order of the vector elements can be rearranged, or new elements be added as required. Map_dfr() uses one vector as input; other parameters such as proper formats must be derived in the npct function. Be aware there are variations of map functions that can handle multiple vectors as input.

It is important to note that one limitation of using map_dfr() is it can only generate similar analysis blocks. If the demographic tables get more complex, i.e., nesting country information within the ethnicity subgroups, or mixing the

descriptive statistic blocks within the subgroups, then we suggest defining parallel functions to handle these special cases.

CONCLUSION

Map functions in the purrr package provide user control over the analysis on the input vector(s) or data frame. Even with introductory knowledge, a programmer can make effective use of its functionality to make powerful custom reports. New learning of this can be expanded by practicing and deep diving into this area and gaining expertise. There are additional resources below that explore other applications of the map() family and how they can be used in other CSR tables.

REFERENCES

1. https://Purrr.tidyverse.org/reference/Map.html
2. https://adv-r.hadley.nz/functionals.html
3. https://intro2r.com/loops.html
4. How to make nice publishable adverse event tables using tidyverse | R-bloggers
5. FDA: Guideline for Industry- Structure and Content of Clinical Study Reports
6. https://www.rdocumentation.org/packages/purrr/versions/0.2.5/topics/map
7. https://rdrr.io/cran/purrr/man/map.html
8. https://rpubs.com/christianthieme/589762
9. https://www.w3schools.com/r/r_for_loop.asp

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Samuel Curlee, Gabriela Piasecki, Xiaoqing Tang, Wenjian Xu
Pfizer Inc
500 Arcola Rd.
Collegeville, PA 19426
Email: Samuel.Curlee@Pfizer.com;
Web: www.pfizer.com