R MIEF Skills Workshop

Session 4

September 27, 2024

Today

- Learn how to:
    - Create custom functions
    - Use iterative coding in R using:
        - The "for" loop
        - The `across()` function
        - The `map()` function from the `purrr` package
- Be introduced to:
    - Communicating process and results using R Markdown
- Practice the above!

Functions in R

What is a function?

A function allows you to efficiently save an operation you may have to use repeatedly, and to run it while only having to modify its inputs, or **arguments**.

I want to rescale the vectors a, b, c, and d so that the smallest value is 0 and the largest value is 1. Here's how I would do it in R:

```
(a-min(a, na.rm = TRUE)) / (max(a, na.rm = TRUE)-min(a, na.rm = TRUE))
(b-min(b, na.rm = TRUE)) / (max(b, na.rm = TRUE)-min(b, na.rm = TRUE))
(c-min(c, na.rm = TRUE)) / (max(c, na.rm = TRUE)-min(c, na.rm = TRUE))
(d-min(d, na.rm = TRUE)) / (max(d, na.rm = TRUE)-min(d, na.rm = TRUE))
```

This is not considered good coding; it requires a lot of copying/pasting, it increases the number of opportunities to make a typo mistake, and it is not efficient.

Functions in R

Solution: custom functions. Custom functions in R are written in the following manner:

```r
name ← function(arguments) {
  body
}
```

In the case of our reshaping function, it would look like this:

```r
# Example for the argument a:
# (a-min(a, na.rm = TRUE)) / (max(a, na.rm = TRUE)-min(a, na.rm = TRUE)

reshape_function ← function(x) {
  (x-min(x,na.rm = TRUE)) / (max(x,na.rm = TRUE)-min(x,na.rm = TRUE))
}
reshape_function(c(-10, 0, 10))
```

```
## [1] 0.0 0.5 1.0
```

```r
reshape_function(c(1, 2, 3, 4, 5))
```

```
## [1] 0.00 0.25 0.50 0.75 1.00
```

Iterative Functions in R

What if I want to apply the same function to multiple objects?

What if I wanted to apply `reshape_function`() to 30 variables in my dataset? Rewriting it would be inefficient...

A possible, base R solution: use a "for" loop.

```r
for(vector in list(c(-10, 0, 10), c(1, 2, 3, 4, 5))) {
    print(reshape_function(vector))
}
```

```
## [1] 0.0 0.5 1.0
## [1] 0.00 0.25 0.50 0.75 1.00
```

However, the use of "for" loops is discouraged in R because they have to run sequentially through each element of the object. To loop over large objects, this is much slower than alternative options, which directly apply to the whole object.

Iterative Functions in R

A more efficient solution: the function `map()` from the package `purrr`.

```r
library(purrr)
purrr::map(
    .x = list(
        c(-10, 0, 10), c(1, 2, 3, 4, 5)
    ),
    .f = reshape_function
)
```

```
## [[1]]
## [1] 0.0 0.5 1.0
##
## [[2]]
## [1] 0.00 0.25 0.50 0.75 1.00
```

Iterative Functions in R

Notice that the previous chunk's output is a list. If you want to output a different class of object, `purrr` has the functions `map_chr()`, `map_dbl()`, and `map_df()` amongst others.

`map()` has a shorthand to simplify its use: instead of writing the function out, you can replace `function() {}` with `~` and the function's argument with `.x`.

```r
map_chr(
    c(1, 3),
    function(x) {
        x + 10
    }
)
```

```
## [1] "11.000000" "13.000000"
```

```r
map_chr(
    c(1, 3),
    ~ .x + 10
)
```

```
## [1] "11.000000" "13.000000"
```

Iterative Functions in R

We can also apply `map()` to a data frame. If we do so, `map()` will apply the same function to every component of the data frame, i.e. its **columns**. In practice, this is equivalent to applying `map()` to a list of vectors.

```r
library(palmerpenguins) # A fun practice dataset about penguins!
data(package = 'palmerpenguins')
# Let's take a glimpse at the dataset:
penguins %>% glimpse()
```

```
## Rows: 344
## Columns: 8
## $ species           <fct> Adelie, Adelie, Adelie, Adelie, Adelie, Adelie, A
## $ island            <fct> Torgersen, Torgersen, Torgersen, Torgersen, Torge
## $ bill_length_mm    <dbl> 39.1, 39.5, 40.3, NA, 36.7, 39.3, 38.9, 39.2, 34.
## $ bill_depth_mm     <dbl> 18.7, 17.4, 18.0, NA, 19.3, 20.6, 17.8, 19.6, 18.
## $ flipper_length_mm <int> 181, 186, 195, NA, 193, 190, 181, 195, 193, 190,
## $ body_mass_g       <int> 3750, 3800, 3250, NA, 3450, 3650, 3625, 4675, 347
## $ sex               <fct> male, female, female, NA, female, male, female, m
## $ year              <int> 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2
```

Iterative Functions in R

I want to reshape the numerical variables of the `penguins` dataset. Using `map()`:

```r
penguins_reshaped ← penguins %>%
  map_dfr(
    ~ if(is.numeric(.x)) {
      reshape_function(.x)
    } else {
      .x
    }
  )
penguins_reshaped %>% glimpse()
```

```
## Rows: 344
## Columns: 8
## $ species           <fct> Adelie, Adelie, Adelie, Adelie, Adelie, Adelie, A
## $ island            <fct> Torgersen, Torgersen, Torgersen, Torgersen, Torge
## $ bill_length_mm    <dbl> 0.25454545, 0.26909091, 0.29818182, NA, 0.1672727
## $ bill_depth_mm     <dbl> 0.6666667, 0.5119048, 0.5833333, NA, 0.7380952, 0
## $ flipper_length_mm <dbl> 0.15254237, 0.23728814, 0.38983051, NA, 0.3559322
## $ body_mass_g       <dbl> 0.2916667, 0.3055556, 0.1527778, NA, 0.2083333, 0
## $ sex               <fct> male, female, female, NA, female, male, female, m
## $ year              <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
```

Iterative Functions in R

The `dplyr` package offers us a function that makes it much easier to modify multiple dataset columns at once: `across()`:

```
penguins_reshaped ← penguins %>%
  mutate(
    across(
      c(bill_length_mm, bill_depth_mm, flipper_length_mm, body_mass_g),
      ~ reshape_function(.x)
    )
  )
penguins_reshaped %>% glimpse()
```

```
## Rows: 344
## Columns: 8
## $ species           <fct> Adelie, Adelie, Adelie, Adelie, Adelie, Adelie, A
## $ island            <fct> Torgersen, Torgersen, Torgersen, Torgersen, Torge
## $ bill_length_mm    <dbl> 0.25454545, 0.26909091, 0.29818182, NA, 0.1672727
## $ bill_depth_mm     <dbl> 0.6666667, 0.5119048, 0.5833333, NA, 0.7380952, 0
## $ flipper_length_mm <dbl> 0.15254237, 0.23728814, 0.38983051, NA, 0.3559322
## $ body_mass_g       <dbl> 0.2916667, 0.3055556, 0.1527778, NA, 0.2083333, 0
## $ sex               <fct> male, female, female, NA, female, male, female, m
## $ year              <int> 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2
```

`across()` identifies columns by name, or can be combined with `where()` to identify columns by their class. The following are equivalent:

```
penguins %>%
  mutate(
    across(
      c(bill_length_mm, bill_depth_mm, flipper_length_mm, body_mass_g, y
      ~ reshape_function(.x)
    )
  )
penguins %>%
  mutate(
    across(
      matches("length_mm|depth_mm|body_mass|year"),
      ~ reshape_function(.x)
    )
  )
penguins %>%
  mutate(
    across(
      where(is.numeric),
      ~ reshape_function(.x)
    )
  )
```

Iterative Functions in R

Beyond the scope of this class: sometimes we want to iterate functions that have more than one argument.

To do so, `purrr` provides the following functions: `map2()` (two arguments) and `pmap()` (unlimited arguments)

```
animals ← c("tiger", "elephant", "giraffe")
colors ← c("red", "orange", "blue")
map2( # paste0() concatenates its arguments into a single string
    animals, colors, ~ paste0("The ", .x, " is ", .y, ".")
)
```

```
## [[1]]
## [1] "The tiger is red."
##
## [[2]]
## [1] "The elephant is orange."
##
## [[3]]
## [1] "The giraffe is blue."
```

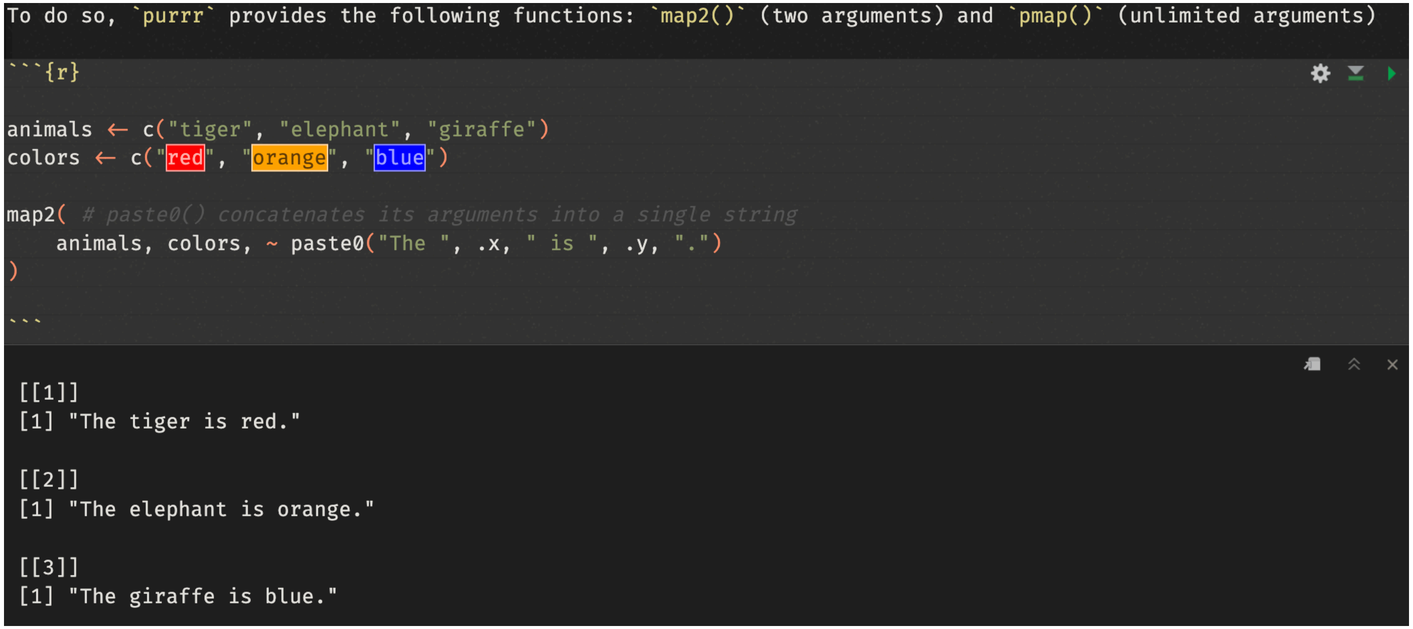
For more on iterative functions, I strongly recommend this blog post by Rebecca Barter.

A (Quick) Introduction to R Markdown

RStudio comes with a text and code compiler that allows you to craft narrated scripts, slide decks, blogs, books, and more: R Markdown.

Instead of creating an R Script, you can select "R Markdown" when creating a new document in RStudio.

R Markdown allows you to use the "markdown" markup language to format your text, while include code and outputs using code "chunks":



~~~
To do so, `purrr` provides the following functions: `map2()` (two arguments) and `pmap()` (unlimited arguments)

```{r}

animals <- c("tiger", "elephant", "giraffe")
colors <- c("red", "orange", "blue")

map2( # paste0() concatenates its arguments into a single string
    animals, colors, ~ paste0("The ", .x, " is ", .y, ".")
)

```
~~~

```
[[1]]
[1] "The tiger is red."

[[2]]
[1] "The elephant is orange."

[[3]]
[1] "The giraffe is blue."
```

A (Quick) Introduction to Quarto

Quarto is a brand new publishing system from the RStudio creators, intended to improve on R Markdown. It allows to combine different coding languages (e.g. both R and Python) into reports, slide decks, websites, etc.

See https://quarto.org/ for more information.

Practical Exercise — Using the World Values Survey Dataset

World Values Survey

Background

"The survey, which started in 1981, seeks to use the most rigorous, high-quality research designs in each country. The WVS consists of nationally representative surveys conducted in almost 100 countries which contain almost 90 percent of the world's population, using a common questionnaire. [...] WVS seeks to help scientists and policy makers understand changes in the beliefs, values and motivations of people throughout the world."

Survey Contents

- Social values, attitudes & stereotypes
- Societal well-being
- Social capital, trust and organizational membership
- Economic values
- Corruption
- Migration
- Post-materialist index
- Science & technology
- Religious values
- Security
- Ethical values & norms
- Political interest and political participation
- Political culture and political regimes
- Demography

Today's practical component

1. Successfully **fix** the code in the `session_4.R` script.
2. Work on your final assignment! The final assignment is to complete the Session 2, 3, and 4 challenges. You can find the challenges rewritten together on the next slide.

NOTE — You should refer to documentation for the dataset, which can be found in the "Module" section, "Course Resources" Module on Canvas, for details on the variables and their given values.

End of Course Assignment (Due on Thursday, October 10)

Session 2 Challenge | Session 3 Challenge | Session 4 Challenge | Overall

Using child_values_country_data, create a scatter plot showing an interesting comparison between two child values across countries.

Using child_values_continent_data, create a bar plot comparing a specific child value across continents.

Using child_values_continent_data, create a `gt` table showcasing the same data as in your bar plot.

End of Course Assignment (Due on Thursday, October 10)

Session 2 Challenge | **Session 3 Challenge** | Session 4 Challenge | Overall

Create your own script and do the following:

1. Find mean values for 'importance in life' variables (Q1-6) for countries in another region than Europe

2. Calculate average 'enthusiasm' for these life subjects in countries in that non-Europe region

3. Perform the same analysis, either on European countries or other countries, for one of the following group of indicators in the dataset:

    - Important child qualities: Q7-18
    - Neighbors: Q19-26
    - Statements to agree with: Q27-41

4. Save one dataset for each of the tasks above.

End of Course Assignment (Due on Thursday, October 10)

Session 2 Challenge | Session 3 Challenge | **Session 4 Challenge** | Overall

CHALLENGE 1 — You are going to use the map() function to rewrite the data importing code below. Currently, the code loads year-by-year datasets individually and then uses bind_rows() to bring them together. Modify it to use map() and list_rbind() instead. Remember to use help(...) if you're unsure how a function works.

CHALLENGE 2 — You are going to rewrite the data wrangling section from Session 3, replacing repetitive code with more efficient uses of the across() function. Hint — Steps 1, 2, and 5 shouldn't be affected. Focus on rewriting Steps 3 and 4 to be more efficient.

End of Course Assignment (Due on Thursday, October 10)

Session 2 Challenge   Session 3 Challenge   Session 4 Challenge   **Overall**

Final deliverable: a .zip file of the the "**your name** R Course Final Assignment" folder. It should have the following:

- An .rproj file in the main folder.
- "code", "data", and "output" folders.
- Three scripts in the "code" folder: `session_2_assignment.R`, `session_3_assignment.R`, and `session_4_assignment.R`. You can use the `session_2.R`, `session_3.R`, and `session_4.R` scripts as foundations for your assignment scripts.
- The requisite data in the "data" folder. You should just need the data from the three sessions, which can be found in each session's Module on Canvas.
- The outputs from your `session_2_assignment.R` script in the "output" folder.

Course Feedback

https://forms.gle/TbaSCbqqKAfpjtF59

Links

Hadley Wickham, Mine Çetinkaya-Rundel & Garrett Grolemund, R for Data Science, 2e — Custom Functions

Hadley Wickham, “dplyr 1.0.0: working across columns”

Rebecca Barter, "Learn to purrr"

RStudio, RStudio Cheatsheets