
Illustration: Amrei Binzer-Panchal

Basic Biostatistics and Bioinformatics

# Writing functions in R

Swedish University of Agricultural Sciences, Alnarp

26 February 2024

# Basic Biostatistics and Bioinformatics

A seminar series on the fundamentals

Organised by *SLUBI* and *Statistics at SLU*

Presentation of background and a practical exercise

Topics

- ~~15 January. Introduction to Markdown~~
- ~~12 February. Population Structure~~
- 26 February. Writing own functions in R

Topic suggestions are welcome

Survey link in previous mail

☰

## SLUBI

- SLU bioinformatics center
- Weekly online drop-in (Wednesdays at 13.00)
- slubi@slu.se, https://www.slubi.se
- Alnarp: Lizel Potgieter (Dept. of Plant Breeding)

## Statistics at SLU

- SLU statistics center
- Free consultations for all SLU staff
- statistics@slu.se
- Alnarp: Jan-Eric Englund and Adam Flöhr (Dept. of Biosystems and Technology)

# Today’s Presentation

Writing own functions in R

How is it done?

When is it good to do?

# What is a function?

A function is an object with can take an input (or *arguments*) and produce an output

- The `sqrt()` function takes numerical value(s) as *input* and gives the square root as *output*

The input and output can be of any form

- `*()`, used as `5 * 7` takes two values and give their product
- `sum()` takes many values and give a single numerical output
- `plot()` takes multiple values and produce a graphic
- `t.test()` takes numerical values and give the results of a t-test

# How to write a function in R

A function is created with the `function()` function

The function is stored with the assign arrow pointing to the function name

This creates a function called `add_five()` which add five to a number

```r
add_five <- function(x){
  return(x + 5)
}

add_five(10)
```

```
[1] 15
```

The return() call is used to assign the output of the function

# Output of a function

The output can be set with the `return()` function

If there is no return the output of the function is the final printed object

```
foo <- function(x){
  print(x + 4)
  x + 5
}

foo(1)
```

```
[1] 5
```

```
[1] 6
```

The value within `print()` is printed but not an actual output of the function

# Function arguments

We can set multiple arguments to our function

```r
add_three_numbers <- function(a, b, c){
  a + b + c
}
add_three_numbers(1,3,5)
```

```
[1] 9
```

### Defaults

It is often convenient to set defaults for some arguments

This will be the value used if no argument is given

```r
add_three_numbers <- function(a, b = 2, c = 2){
  a + b + c
}
add_three_numbers(1,3,5) # Setting all arguments
```

```
[1] 9
```

```r
add_three_numbers(1)     # Default for two later arguments
```

```
[1] 5
```

# Multiple output

A single output can be specified with `return()` or printed at the end if the function *body*

For multiple outputs several values can be collected in a list

```r
calculate_mean_and_sum <- function(x){
  m <- mean(x)
  s <- sum(x)
  list(m, s)
}
calculate_mean_and_sum(c(1,3,4,5,6))
```

```
[[1]]
[1] 3.8

[[2]]
[1] 19
```

# When is a function useful?

For practical purposes there are a few advantages of functions

- Avoiding repetition
- Simplifying long sequences
- Creating structure and increasing readability

☰

# Examples of avoiding repetitions

We have previously seen the `palmerpenguins` data

```r
library(palmerpenguins)
penguins
```

```
# A tibble: 344 × 8
   species island    bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
   <fct>   <fct>              <dbl>         <dbl>             <int>       <int>
 1 Adelie  Torgersen           39.1          18.7               181        3750
 2 Adelie  Torgersen           39.5          17.4               186        3800
 3 Adelie  Torgersen           40.3          18                 195        3250
 4 Adelie  Torgersen           NA            NA                  NA          NA
 5 Adelie  Torgersen           36.7          19.3               193        3450
 6 Adelie  Torgersen           39.3          20.6               190        3650
 7 Adelie  Torgersen           38.9          17.8               181        3625
 8 Adelie  Torgersen           39.2          19.6               195        4675
 9 Adelie  Torgersen           34.1          18.1               193        3475
10 Adelie  Torgersen           42            20.2               190        4250
# ℹ 334 more rows
# ℹ 2 more variables: sex <fct>, year <int>
```

Say we want to do an Anova model comparing species, with a normality test and a heteroskedasticity test

We can wrap this in a function with the name of a column as the input

# Continued

The following functon runs the model with the set variable as response

Collects anova, Shapiro test (normality) and Levene test (heteroskedasticity)

```r
perform_anova_for_variable <- function(variable){
  y <- penguins %>% pull(variable)
  mod <- lm(y ~ species, penguins)
  list(anova(mod),
       shapiro.test(residuals(mod)),
       car::leveneTest(mod))
}
```

# Continued

```r
perform_anova_for_variable("bill_length_mm")
```

```
[[1]]
Analysis of Variance Table

Response: y
           Df Sum Sq Mean Sq F value    Pr(>F)    
species     2 7194.3  3597.2   410.6 < 2.2e-16 ***
Residuals 339 2969.9     8.8                      
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

[[2]]

	Shapiro-Wilk normality test

data:  residuals(mod)
W = 0.98903, p-value = 0.01131


[[3]]
Levene's Test for Homogeneity of Variance (center = median)
       Df F value Pr(>F)
group   2  2.2425 0.1078
      339               
```

# Naming a function

Functions are usually named after verbs

If the name contains multiple words they are typically separated with underscores

# R Studio

RStudio contains a feature where a piece of code can be made into a function

Mark a code section in a script and go to Code > Extract Function

This will create a function

Undefined variables will be set as function arguments

# The End.

Thank for listening.

Tutorial in five minutes: https://www.dataquest.io/blog/write-functions-in-r/