Session 2-Functions and Loops

Ya-Feng Wen

2020-06-17

Contents

Objectives

- Functions
- Loops
  - For loops
  - While loops

Functions

When should you write a function?

When you've copied and pasted a block of code more than twice! It's called the DRY ("Do not repeat yourself") principle.

Benefits of a function

- An evocative function name makes code easier to understand
- Only need to update code in one place
- Reduce incidental mistakes when copying and pasting code

Three key steps to creating a new function:

1. Choose an evocative function name
   - Ideally should be verbs
   - Use snake_case or camelCase
   - Use consistent names and arguments if you have a family of functions that do similar things
   - Avoid overriding existing functions and variables
2. List the inputs, or **arguments** (should be nouns), to the function inside `function`

3. Place the code you have developed in body of the function, a `{` block that immediately follows `function(...)`

It's easier to start with working code and turn it into a function; it's harder to create a function and then try to make it work.

4. Unit testing to turn informal, interactive tests into formal, automated tests

How to write a function?

- Basic structure: `function(arglist) {body}`
- One line function (drop curly braces): `function(arglist) body`. For example, `function(x) file.info(x)$isdir` to check whether path `x` is a directory or not.
- Using snippet. Type `fun` + Tab

Practice Function Writing

1. Create a function which calculate the proportion of NA values in a vector.

```
# create a testing vector called test
test <- c(1:5, NA, 6:8, NA)

# check which element is NA
is.na(test)

# count the proportion of NA
as.numeric(is.na(test)) # convert Boolean to numeric
sum(as.numeric(is.na(test))) # sum up number of NA
sum(as.numeric(is.na(test)))/length(test) # calculate the proportion

# alternatively, use the following expression to replace three lines of code above
mean(is.na(test))


prop_NA <- function(x) {
  mean(is.na(x))
}

# testing your function
prop_NA(test)
```

2. Write a function to calculate variance of a vector.

$$Var(x) = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

```
# number of element
length(test) # including NA
length(na.omit(test))

# mean of the vector
mean(test) # returns NA
mean(test, na.rm = TRUE)
```

```r
# mean square error
(test - mean(test, na.rm = TRUE))^2

# sum of mean square error
sum((test - mean(test, na.rm = TRUE))^2) # returns NA
sum((test - mean(test, na.rm = TRUE))^2, na.rm = T)


variance <- function(x, na.rm=TRUE) {
  n <- length(na.omit(x))
  m <- mean(x, na.rm = na.rm)
  mean_sqaure_error <- (x - m)^2

  sum(mean_sqaure_error, na.rm = na.rm) / (n-1)
}


variance(test)
variance(test) == var(test, na.rm = T)
```

3. Write a function to calculate drug concentrations until time T and plot the concentration time curve. This function allows you to calcuate drug concentrations following either the one-compartment or two-compartment PK (given as IV) with linear elimination. Recall that concentration can be described as follow: For One-Compartment:

$$C(t) = C_0 \times e^{-kt}$$

For Two-Compartment:

$$C(t) = Ae^{-\alpha t} + Be^{-\beta t}$$

```r
Dose = 100 # mg
Vc = 2 # L
k = 0.1 # hr-1

t = seq(0,24,1)

C0 = Dose/Vc

Conc = C0 * exp(-k*t)
#plot(Conc~t)

oneCmtIV <- function(Dose,Time) {
  # define parameters
  Vc = 2 # L
  k = 0.1 # hr-1
  # simulation time
  t = seq(0, Time, 1)
  # calculate concentration
  C0 = Dose/Vc
  Conc = C0 * exp(-k*t)
  # plot
  plot(Conc ~ t)
}

#oneCmtIV(Dose = 100, Time = 24)
```

```r
twoCmtIV <- function(Dose, Time) {

  A = 30 # mg/L
  B = 20 # mg/L
  alpha = 1.5 # hr-1
  beta = 0.1 # hr-1

  t = seq(0, Time, 1)

  C0 = Dose/Vc
  Conc = A*exp(-alpha*t) + B*exp(-beta*t)

  plot(Conc ~ t)
}

#twoCmtIV(100, 24)


simulateOneTwoCmtIV <- function(Dose, Time, Cmt) {
  # parameters for 1-CMT
  Vc = 2 # L
  k = 0.1 # hr-1

  # parameters for 2-CMT
  A = 30 # mg/L
  B = 20 # mg/L
  alpha = 1.5 # hr-1
  beta = 0.1 # hr-1

  t = seq(0, Time, 0.1)

  C0 = Dose/Vc

  if (Cmt == 1) {

    Conc = C0 * exp(-k*t)

  } else if (Cmt == 2) {

    Conc = A*exp(-alpha*t) + B*exp(-beta*t)

  } else {

    stop("Cmt must be 1 or 2")

  }

  plot(Conc ~ t)

}

#simulateOneTwoCmtIV(100, 24, 1)
#simulateOneTwoCmtIV(100, 24, 2)
```

```
#simulateOneTwoCmtIV(100, 24, 3)
```

Resources:

1. R for Data Science by Grlemund and Wickham (2017), online at https://r4ds.had.co.nz/
2. Advanced R by Wickham, online at http://adv-r.had.co.nz/