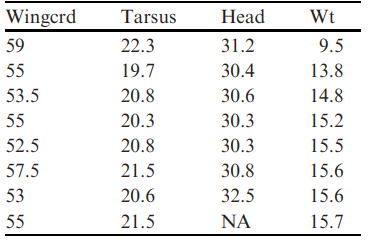
Chương 2: Nhập dữ liệu vào R

2.1. Các bước đầu tiên trong R

2.1.1. Nhập vào một tập dữ liệu nhỏ

Ta có bảng dữ liệu sau:



Cách đơn giản nhất tuy khó khăn không kém là nhập từng giá trị vô hướng, ta có thể nhập như sau:

> a <- 59

> b <- 55

> c <- 53.5

> d <- 55

> e <- 52.5

Ta có thể thay thế “<-“ bằng dấu bằng “=”. Để có thể thấy kết quả, bạn nhập như sau và nhấn Enter:

> a

[1] 59

Tuy nhiên việc đặt tên biến a, b, c có vẻ như là không hiệu quả vì nó chẳng đại diện cho gì cả. Do đó ta có thể đặt tên như sau:

> Wing1 <- 59

> Wing2 <- 55

> Wing3 <- 53.5

> Wing4 <- 55

> Wing5 <- 52.5

Bạn có thể thực hiện một vài tính toán đơn giản và quan sát kết quả, nếu bạn muốn kết quả được lưu trữ hãy khởi tạo một biến với tên phù hợp như sau:

> SQ.wing1 <- sqrt(Wing1)

> Mul.W1 <- 2 \* Wing1

> Sum.12 <- Wing1 + Wing2

> SUM12345 <- Wing1 + Wing2 + Wing3 + Wing4 + Wing5

> Av <- (Wing1 + Wing2 + Wing3 + Wing4 + Wing5) / 5

Để hiển thị kết quả của SQ.wing1, bạn cần nhấn:

> SQ.wing1

[1] 7.681146

Hoặc có một cách thay thế như sau:

> (SQ.wing1 <- sqrt(Wing1))

[1] 7.681146

2.1.2. Kết hợp dữ liệu với hàm c

Với bảng dữ liệu ở trên nếu bạn chỉ dùng từng biến để lưu trữ từng quan sát bạn sẽ cần 32 biến. R cung cấp cho chúng ta hàm c() để kết hợp các dữ liệu bằng cách như sau:

> Wingcrd <- c(59, 55, 53.5, 55, 52.5, 57.5, 53, 55)

Chúng tôi khuyến nghị các bạn nên đặt dấu cách giữa dấu phẩy, cũng như ở “ + ”, “ <- ” để cải thiện khả năng đọc mã. Để hiển thị dữ liệu, nhấn Wingcrd vào R và nhấn Enter:

> Wingcrd

[1] 59.0 55.0 53.5 55.0 52.5 57.5 53.0 55.0

Hàm c tạo một vector với độ dài là 8. Để quan sát giá trị đầu tiên của Wingcrd, ta nhấn:

> Wingcrd

[1] [1] 59

Hiển thị 5 giá trị đầu tiên:

> Wingcrd [1:5]

[1] 59.0 55.0 53.5 55.0 52.5

Hiển thị tất cả phần tử trừ phần tử thứ 2:

> Wingcrd [-2]

[1] 59.0 53.5 55.0 52.5 57.5 53.0 55.0

Thao tác tương tự với các cột dữ liệu còn lại của bảng ở phần trên:

> Tarsus <- c(22.3, 19.7, 20.8, 20.3, 20.8, 21.5, 20.6, 21.5)

> Head <- c(31.2, 30.4, 30.6, 30.3, 30.3, 30.8, 32.5, NA)

> Wt <- c(9.5, 13.8, 14.8, 15.2, 15.5, 15.6, 15.6, 15.7)

Nhận thấy có một giá trị của biến Head không được đo lường (NA), nó có thể gây ra rắc rối như sau:

> sum(Head)

[1] NA

Để xử lí vấn đề trên, ta them một đối số nữa trong hàm sum như sau:

> sum(Head, na.rm = TRUE)

[1] 216.1

2.1.3 Kết hợp các biến với các hàm c, cbind và rbind.

> BirdData <- c(Wingcrd, Tarsus, Head, Wt)

Để hiển thị kết quả ta nhấn:

> BirdData

[1] 59.0 55.0 53.5 55.0 52.5 57.5 53.0 55.0 22.3

[10] 19.7 20.8 20.3 20.8 21.5 20.6 21.5 31.2 30.4

[19] 30.6 30.3 30.3 30.8 32.5 NA 9.5 13.8 14.8

[28] 15.2 15.5 15.6 15.6 15.7

BirdData là một vector với chiều dài 32. Các số [1], [10], [19], [28] biểu thị cho chỉ số của phân tử đầu tiên của dòng đó trong BirdData.

Ta tạo một vector Id (dùng để định dannh) như sau:

> Id <- c(1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4)

> Id

[1] 11111111222222223333333

[24] 344444444

Hoặc một cách nhanh gọn hơn với hàm rep():

> Id <- rep(c(1, 2, 3, 4), each = 8)

> Id

[1] 11111111222222223333333

[24] 344444444

Hoặc:

> Id <- rep(1 : 4, each = 8)

> Id

[1] 11111111222222223333333

[24] 344444444

Hoặc dùng hàm seq() như sau:

> a <- seq(from = 1, to = 4, by = 1)

> rep(a, each = 8)

[1] 11111111222222223333333

[24] 344444444

Ngoài ra, không những định danh bằng số ta có thể định danh bằng tên như sau:

> VarNames <- c("Wingcrd", "Tarsus", "Head", "Wt")

> Id2 <- rep(VarNames, each = 8)

> Id2

[1] "Wingcrd" "Wingcrd" "Wingcrd" "Wingcrd"

[5] "Wingcrd" "Wingcrd" "Wingcrd" "Wingcrd"

[9] "Tarsus" "Tarsus" "Tarsus" "Tarsus"

[13] "Tarsus" "Tarsus" "Tarsus" "Tarsus"

[17] "Head" "Head" "Head" "Head"

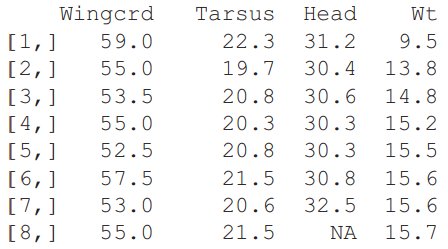
[21] "Head" "Head" "Head" "Head"

[25] "Wt" "Wt" "Wt" "Wt" [29] "Wt" "Wt" "Wt" "Wt"

Ta dùng hàm cbind như sau:

> Z <- cbind(Wingcrd, Tarsus, Head, Wt)

> Z



Ta truy cập đến các giá trị của Z như sau:

> Z[, 1]

[1] 59.0 55.0 53.5 55.0 52.5 57.5 53.0 55.0

> Z[1 : 8, 1]

[1] 59.0 55.0 53.5 55.0 52.5 57.5 53.0 55.0

> Z[2, ]



> Z[2, 1:4]



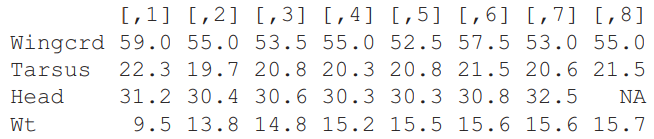
> dim(Z)

[1] 8 4

Tương tự với rbind tuy nhiên lúc này sẽ hiển thị theo chiều ngang:

> Z2 <- rbind(Wingcrd, Tarsus, Head, Wt)

> Z2



2.1.4. Kết hợp dữ liệu với hàm vector

Ta muốn tạo một vector với chiều dài 8 bao gồm dữ liệu Wingcrd. Trong R, ta có thể làm như sau:

> W <- vector(length = 8)

> W[1] <- 59

> W[2] <- 55

> W[3] <- 53.5

> W[4] <- 55

> W[5] <- 52.5

> W[6] <- 57.5

> W[7] <- 53

> W[8] <- 55

Sau khi chạy các dòng lệnh trên, ta nhấn W vào R để hiển thị kết quả:

> W

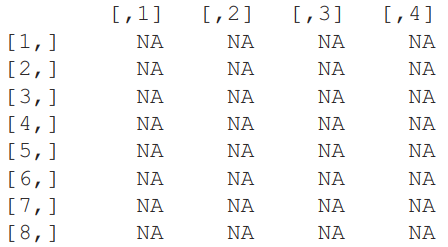
[1] 59.0 55.0 53.5 55.0 52.5 57.5 53.0 55.0

2.1.5 Kết hợp dữ liệu với hàm matrix

Tạo một matrix rỗng với 8 dòng và 4 cột:

> Dmat <- matrix (nrow = 8, ncol =4)

> Dmat



Tiến hành nhập dữ liệu:

> Dmat[, 1] <- c(59, 55, 53.5, 55, 52.5, 57.5, 53, 55)

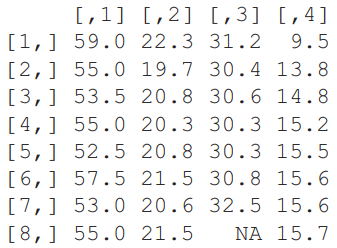
> Dmat[, 2] <- c(22.3, 19.7, 20.8, 20.3, 20.8, 21.5, 20.6, 21.5)

> Dmat[, 3] <- c(31.2, 30.4, 30.6, 30.3, 30.3, 30.8, 32.5, NA)

> Dmat[, 4] <- c(9.5, 13.8, 14.8, 15.2, 15.5, 15.6, 15.6, 15.7)

Hiển thị kết quả:

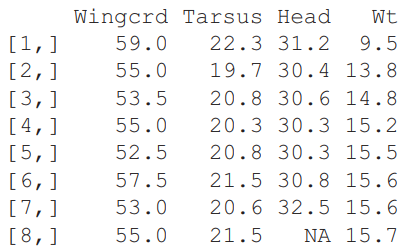
> Dmat



Ta thêm tên của các cột:

> colnames(Dmat) <- c("Wingcrd", "Tarsus", "Head","Wt")

> Dmat



2.1.6. Kết hợp dữ liệu với hàm data.frame

Ngoài việc sử dụng các hàm c, cbind, rbind, vector và matrix để kết hợp dữ liệu, ta còn có một cách khác là data frame.

> Dfrm <- data.frame(WC = Wingcrd, TS = Tarsus, HD = Head, W = Wt)

> Dfrm

Ta truy cập vào từng cột của data.frame bằng cách sau:

> Dfrm$W

[1] 9.5 13.8 14.8 15.2 15.5 15.6 15.6 15.7

2.1.7. Kết hợp dữ liệu với hàm list

> x1 <- c(1, 2, 3)

> x2 <- c("a", "b", "c", "d")

> x3 <- 3

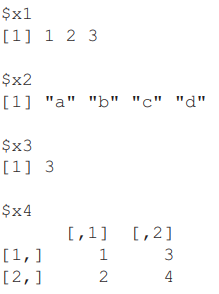
> x4 <- matrix(nrow = 2, ncol = 2)

> x4[, 1] <- c(1, 2) > x4[, 2] <- c( 3, 4)

> Y <- list(x1 = x1, x2 = x2, x3 = x3, x4 = x4)

Ta tiến hành quan sát kết quả:

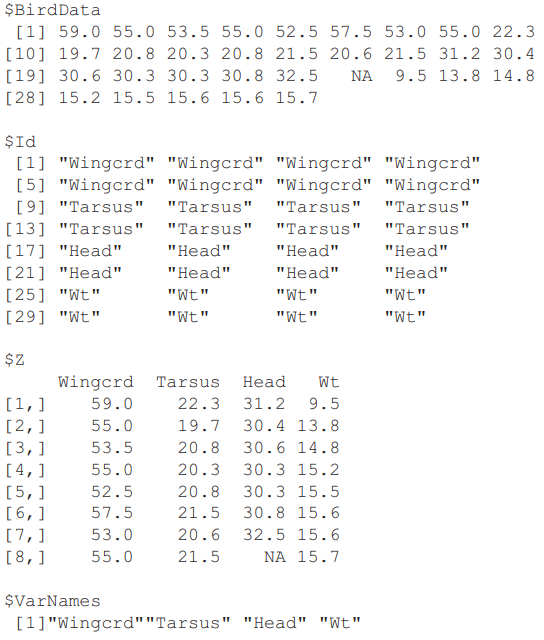
> Y



Với dữ liệu từ bảng trên:

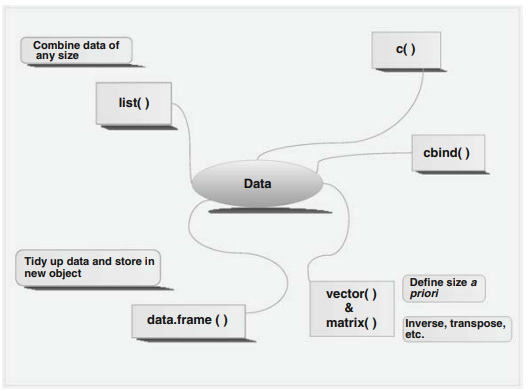
> AllData <- list(BirdData = BirdData, Id = Id2, Z = Z, VarNames = VarNames)

> AllData



Hiển nhiên, lưu trữ trong định dạng này là không cần thiết, chỉ khi chúng ta cần một định dạng. Một lợi thế so với nhiều định dạng khác, là chúng ta đã được chuẩn bị hầu hết các chức năng. Tuy nhiên, chúng tôi chỉ thay đổi định dạng khi nào thực sự cần thiết.

Bạn không thể sử dụng “<-“ bên trong hàm list, chỉ dấu “=” được cho phép. Hình sau đây cho bạn thấy được tổng quan về các phương pháp lưu trữ dữ liệu cho đến hiện nay:



2.2. Nhập dữ liệu

2.2.1. Thông qua Excel

2.2.1.1. Chuẩn bị dữ liệu trong Excel

Để mọi thứ được tiến hành một cách đơn giản, chúng tôi khuyên bạn nên sắp xếp các dữ liệu theo một dạng mẫu. Đó là các cột chứa các biến và các hàng chứa các giá trị quan sát. Nhập NA ở các ô mà ta không có dữ liệu. Sử dụng tên chứa các kí tự như £, $, %, ^ , &, \*, (, ), , #, ?, , ,. ,, /, |, \, ,[ ,] ,{, } có thể xảy ra lỗi ở R, cũng như bạn nên tránh việc đặt tên có chứa dấu cách. Những tên ngắn thường được ưu tiên để tránh trường hợp khi chứa những tên dài hình vẽ sẽ khó quan sát.

2.2.1.2. Chuyển dữ liệu sang tập tin Tab-Delimited ascii

Trong Excel, vào File -> Save As -> Save As -> Save as Type và chọn Text (Tab delimited). Tùy thuộc vào máy tính của bạn dùng ngôn ngữ gì để điều chỉnh sao cho phù hợp. Ta chuyển dữ liệu squid sang dạng squid.txt và lưu trữ ở vị trí phù hợp, ở đây chúng tôi lưu trữ ở “C:\RBook”. Sau đó, chúng ta đóng Excel để nó không chặn các chương trình khác khi truy cập tập tin mới tạo.

Cảnh bảo: Excel có xu hướng thêm các các cột với các giá trị NA trong tập tin ascii trong trường hợp bạn gõ ghi chú trong trang tính. Trong R, những cột này có thể xuất hiện với các giá trị NA. Để tránh trường hợp trên, hãy xóa những cột như vậy ở trong Excel trước khi chuyển sang tập tin ascii

2.2.1.3. Sử dụng hàm read.table

Với tập tin tab-delimited ascii không chứa ô trống hoặc tên có chứa dấu cách, bây giờ ta có thể nhập dữ liệu vào R. Hàm mà chúng ta sử dụng là read.table và cách sử dụng cơ bản của nó như sau:

> Squid <- read.table(file = "C:\\RBook\\squid.txt", header = TRUE)

Dòng lệnh trên đọc dữ liệu từ file squid.txt và lưu trữ vào biến Squid dưới dạng data frame. Chúng tôi cực kỳ khuyến khích việc sử dụng tên ngắn gọn nhưng phải đủ ý. Ví dụ, việc đặt tên SquidNorthSeaMalesFemales là không nên, bởi bạn sẽ dùng từ này khá nhiều lần, bạn dễ mắc lỗi chính tả và chương trình sẽ không chạy được. Đối số header = TRUE trong hàm read.table cho R biết rằng hàng đầu tiên là chứa những nhãn (tên cột), trong trường hợp tập dữ liệu của bạn không có các nhãn hãy chuyển thành header = FALSE

Nếu bạn cần đọc nhiều tệp từ cùng một thư mục, việc đặt thư mực làm việc bằng hàm setwd sẽ hiệu quả hơn:

> setwd("C:\\RBook\\")

> Squid <- read.table(file = "squid.txt", header = TRUE)

Trong cuốn sách này, chúng tôi nhập dữ liệu bằng cách chỉ định thư mục bằng hàm setwd sau đó đọc tệp bằng hàm read.table. Nguyên nhân là không phải ai cũng đều lưu trữ tập dữ liệu ở ổ đĩa C. Do đó, họ chỉ cần thay đổi đường dẫn đến tập dữ liệu trong hàm setwd. Ngoài hàm read.table để đọc dữ liệu, bạn cũng có thể dùng hàm scan.

Để biết thêm chi tiết về hàm scan, bạn nhập dòng lệnh sau trong R:

> ?scan

2.2.2. Truy cập dữ liệu từ những gói phần mềm thống kê khác

Ngoài việc truy cập dữ liệu từ tập ascii, R có thể nhập dữ liệu từ những chương trình thống kê khác như Minitab, S-PLUS, SAS, SPSS,… Tuy nhiên, chúng tôi nhấn mạnh rằng cách tốt nhất là làm việc trực tiếp với dữ liệu gốc thay vì nhập dữ liệu có thể bị sửa đổi từ một gói phần mềm thống kê khác. Bạn cần gõ như sau:

> library(foreign)

Để đọc file Minitab, bạn tìm hiểu thêm về hàm read.mtp

> ?read.mtp

Ngoài ra, ta còn có hàm write.foreign với cú pháp

write.foreign(df, datafile, codefile, package = c("SPSS", "Stata", "SAS"), ...)

Với hàm trên bạn có thể xuất thông tin trong R sang một số gói phần mềm thống kê.

2.2.3. Truy cập cơ sở dữ liệu

Có một số gói đặt biệt có sẵn trong R cung cấp các công cụ bạn cần để nhanh chóng có được quyền truy cập vào bất kì cơ sở dữ liệu nào:

> library(RODBC)

Bạn có thể kết nối với cơ sở dữ liệu của Microsoft Access bằng việc sử dụng odbcConnectAccess. Bạn tiến hành như sau:

> setwd("C:/RBook")

> channel1 <- odbcConnectAccess(file = "MyDb.mdb", uid = "", pwd = "")

Nếu bạn truy cập một cơ sở dữ liệu mfa không có mã định danh hay mật khẩu thì bạn truy cập vào cơ sở dữ liệu một cách trực tiếp bằng tên của nó:

> Channel1 <- odbcConnect("MyDb.mdb")

Ta sử dụng sqlFetch để lấy và lưu trữ dữ liệu vào MyData:

> MyData <- sqlFetch(channel1, "MyTable")

SQL (Structured Query Language) là một ngôn ngữ truy vấn cấu trúc và không khó để học nó. Một lệnh sử dụng trong RODBC để gửi một truy vấn SQL đến cơ sở dữ liệu là sqlQuery(channel, query), bạn có thể sử dụng SqlTables để truy xuất thông tin bảng trong cơ sở dữ liệu hoặc SqlColumns(channel, “MyTable”) để truy xuất thông tin trong các cột của bảng cơ sở dữ liệu “MyTable”.

Chương 3: Truy cập các biến và quản lí các tập con dữ liệu

3.1. Truy cập các biến từ một Data Frame

Trong quá trình phân tích thống kê, điều quan trọng là phải loại bỏ các phần dữ liệu, chọn một số tập hợp con nhất định hoặc sắp xếp chúng. Hầu hết các thao tác này có thể được thực hiện trong Excel hoặc chương trình bảng tính(Hoặc cơ sở dữ liệu) khác trước khi nhập vào R, nhưng vì nhiều lí dó khác nhau, tốt nhất không nên làm điều này. Bạn có thể cần phải nhập lại dữ liệu mỗi lần bạn chọn lựa. Ngoài ra, một số tệp dữ liệu có thể quá lớn để nhập từ bảng tính. Do đó, một số hiểu biết nhất định về thao tác với tệp dữ liệu trong R là hữu ích. Tuy nhiên, có lẽ đây là khía cạnh khó nhất của R, nhưng khi đã thành thạo thì tất cả các thao tác dữ liệu tẻ nhạt trong Excel( hoặc bất cứ bảng tính nào khác) có thể được thực hiện trong R.

Chúng ta sẽ sử dụng tập dữ liệu squid trong chương trước:

> setwd("C:/RBook/")

> Squid <- read.table(file = "squid.txt", header = TRUE)

Chúng tôi khuyên bạn nên dung lệnh names ngay sau read.table để thấy được những biến mà chúng ta sẽ xử lí:

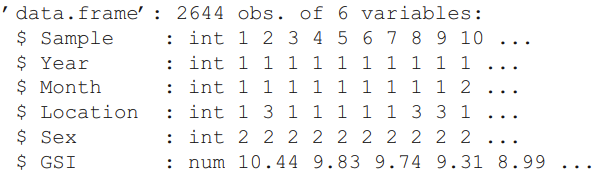
> names(Squid)



3.1.1. Hàm str

Lệnh str (viết tắt của structure) cho chúng ta biết về trạng thái của từng biến trong data frame:

> str(Squid)



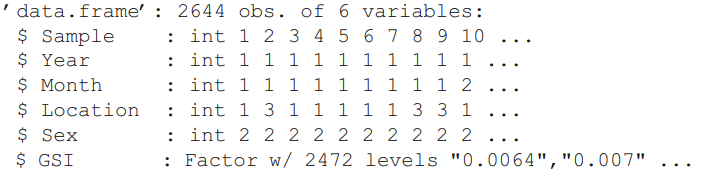
Từ kết quả trên, dễ thấy các biến Sample, Year, Month, Location, Sex có kiểu số nguyên và GSI có kiểu số. Giả sử bạn mắc lỗi trong việc phân tách dữ liệu:

> setwd("C:/RBook/")

> Squid2 <- read.table(file = "squidGSI.txt", dec = ",", header = TRUE)

Squid2 vẫn chứa dữ liệu như Squid, tuy nhiên khi ta lệnh str ta sẽ phát hiện ra vấn đề:

> str(Squid2)



Giờ đây biến GSI là một factor. Điều này có nghĩa là nếu ta dùng một số hàm như mean hay boxplot thì R sẽ xuất hiện lỗi bởi GSI không phải là kiểu số.

Do đó, chúng tôi khuyên bạn nên dùng hàm names và str ngay sau khi read.table

3.1.2. Đối số trong một hàm

Xác định một hàm trong R, chẳng hạn, hàm lm cho hồi quy tuyến tính, xác định mô hình theo các biến GSI, Month, Year và Location và cho hàm lm biết rằng dữ liệu có thể tìm thấy trong data frame Squid. Mặc dù chúng ta không thảo luận thêm về hồi quy tuyến tính trong cuốn sách này nhưng nó trông như sau:

> M1 <- lm(GSI ~ factor(Location) + factor(Year), data = Squid)

Đối số cuối là data = Squid cho biết các biến nằm trong data frame Squid. Đây là cách tiếp cận nhanh bởi ta không cần xác định thêm các biến ngoài data frame. Điểm hạn chế là không phải hàm nào cũng có đối số này, ví dụ:

> mean(GSI, data = Squid)

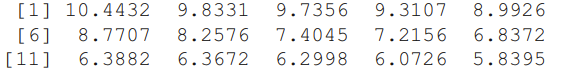
Và R sẽ xuất hiện lỗi:



3.1.3. Kí tự $

Vậy chúng ta nên làm gì nếu hàm không có đối số là data? Có 2 cách để truy cập các biến và cách đầu tiên là sử dụng kí tự $:

> Squid$GSI



Hoặc bạn có thể chọn cột thứ 6 của Squid, cũng cho ra kết quả tương tự:

> Squid[, 6]

Và từ đây, bạn có thể dùng hàm mean để tính trung bình của biến GSI:

> mean(Squid$GSI)

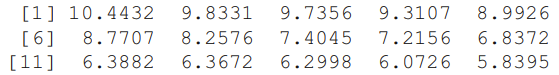
[1] 2.187034

3.1.4. Hàm attach

Bây giờ chúng tôi sẽ cho bạn thấy một cách không khuyến khích để truy cập dữ liệu. Thay vì bạn phải có Squid$ để truy cập vào từng biến thì bây giờ bạn chỉ cần sử dụng hàm attach như sau:

> attach(Squid)

> GSI



Một vài lưu ý khi dùng hàm attach:

- Trành trường hợp các biến bị trùng lặp, không dùng lệnh attach (Squid) hai lần

- Khi bạn dùng lệnh attach, bạn phải chắc chắn rằng mình sử dụng các biến với tên duy nhất. Tránh các tên thông thường như Month, Location,...

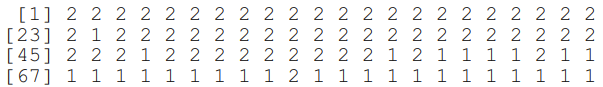
- Khi bạn muốn xử lí nhiều tập dữ liệu và chỉ làm việc với một tập dữ liệu trong một thời điểm nhất định, cân nhắc đến việc loại bỏ data frame khỏi đường dẫn tìm kiếm của R bằng lệnh detach:

> detach(Squid)

3.2. Truy cập tập hợp con dữ liệu

Trong phần này, chúng ta thảo luận về cách truy cập và trích xuất các thành phần của data frame Squid. Ví dụ ta làm việc với biến sex (giới tính) của Squid. Để xuất các tập dữ liệu con, ta cần biết biến sex trong như thế nào:

> Squid$Sex



Bạn có thể dùng lệnh unique để xem các giá trị duy nhất trong biến này:

> unique(Squid$Sex)

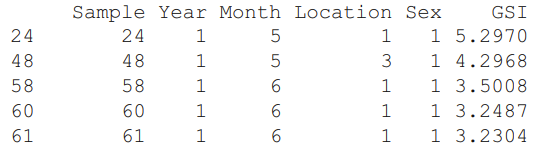
[1] 2 1

Ta có giá trị 1 đại diện cho giống đực và 2 cho giống cái. Để truy cập đến dữ liệu giống đực:

> Sel <- Squid$Sex == 1

> SquidM <- Squid[Sel, ]

> SquidM



Hoặc một cách ngắn gọn hơn như sau:

> SquidM <- Squid[Squid$Sex == 1, ]

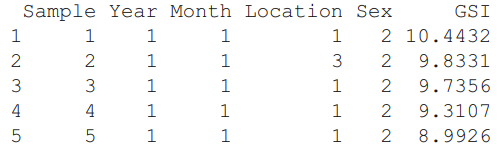
> SquidM

3.2.1 Sắp xếp dữ liệu

Ngoài việc trích xuất các tập dữ liệu con, đôi khí việc sắp xếp lại dữ liệu sẽ rất hữu ích. Đối với dữ liệu squid, bạn có thể muốn sắp xếp dữ liệu GSI từ giá trị thấp đến giá trị cao của biến Month:

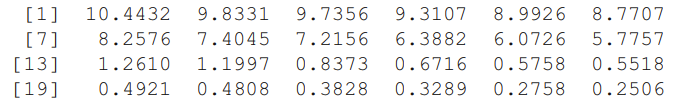
> Ord1 <- order(Squid$Month)

> Squid[Ord1, ]



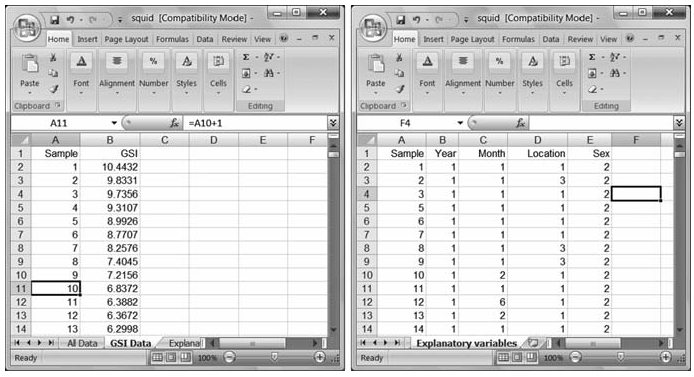
Bạn chỉ muốn quan sát giá trị của biến GSI:

> Squid$GSI[Ord1]



3.3. Kết hợp hai tập dữ liệu

Cho tới nay, chúng ta đã thấy các ví dụ trong đó tất cả các điểm dữ liệu được lưu trữ trong cùng một tệp. Tuy nhiên thực tế không phải luôn như vậy. Ví dụ ta kết hợp hai dữ liệu như hình dưới đây:



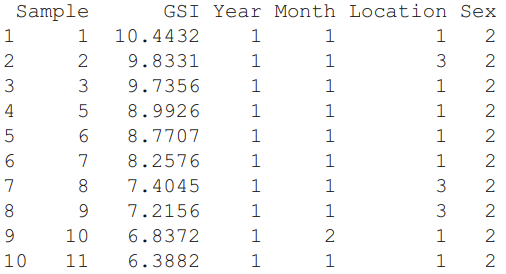
> setwd("C:/RBook/")

> Sq1 <- read.table(file = "squid1.txt", header = TRUE)

> Sq2 <- read.table(file = "squid2.txt", header = TRUE)

> SquidMerged <- merge(Sq1, Sq2, by = "Sample")

> SquidMerged

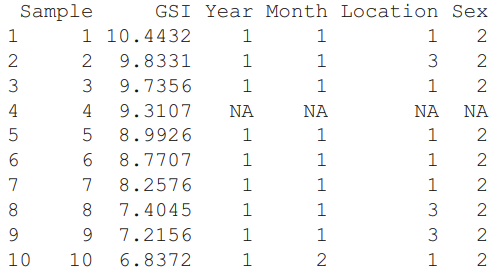


Ta kết hợp hai tập dữ liệu thông qua một biến chung là Sample

Có một đối số hữu ích trong hàm merge là all. Mặc định all có giá trị FALSE, tức là Sq1 hoặc Sq2 có giá trị bị thiếu thì sẽ bỏ qua. Khi chúng ta thiết lập thành TRUE, NA sẽ được điền nếu Sq1 không có dữ liệu và ngược lại:

> SquidMerged <- merge(Sq1, Sq2, by = "Sample", all = TRUE)

> SquidMerged



3.4. Xuất dữ liệu

Ngoài lệnh read.table, R còn có lệnh write.table. Với hàm này, bạn có thể xuất thông tin dạng số sang tệp dạng ascii:

> SquidM <- Squid[Squid$Sex == 1, ]

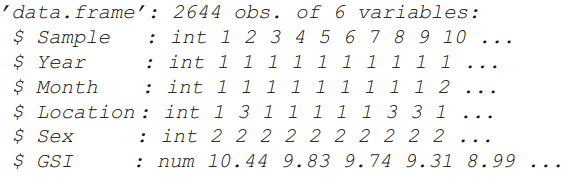
> write.table(SquidM, file = "MaleSquid.txt", sep = " ", quote = FALSE, append = FALSE, na = "NA")

Đối số đầu tiên trong hàm write.table là biến bạn muốn xuất, đối số thứ hai là tên biến. Với đối số sep = “ ” đảm bảo rằng dữ liệu của bạn phân tách bởi dấu cách, quote = FALSE tránh dấu ngoặc kép ở chuỗi kí tự, na = “NA” cho phép bạn chỉ định cách biểu diễn các giá trị bị thiếu, append = FALSE sẽ mở ra một tập tin mới.

3.5 Ghi lại các biến định tính

Trong phần 3.1, ta đã sử dụng hàm str để đưa ra kết quả sau cho dữ liệu Squid:

> str(Squid)

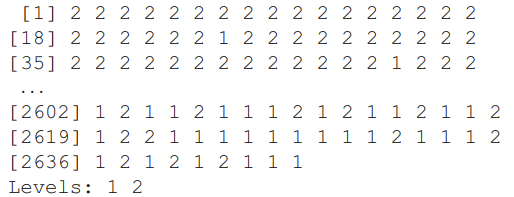


Biến Location được mã hóa thành 1, 2, 3 hoặc 4 và biến Sex là 1 hoặc 2. Các biến như vậy gọi là biến định tính. Trong Excel, chúng ta có thể mã hóa Sex là giống đực và cái. Trong R, ta làm như sau:

> Squid$fLocation <- factor(Squid$Location)

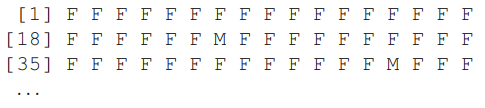
> Squid$fSex <- factor(Squid$Sex)

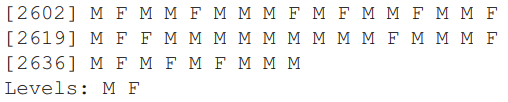
> Squid$fSex



Một cách trực quan hơn:

> Squid$fSex <- factor(Squid$Sex, levels = c(1, 2), labels = c("M", "F")) > Squid$fSex





Lưu ý trước đây ta có dòng lệnh:

> SquidM <- Squid[Squid$Sex == 1, ]

Bây giờ, nếu ta muốn thực viện với biến fSex thì phải làm như sau:

> SquidM <- Squid[Squid$fSex == "1", ]

Chương 4: Một số hàm đơn giản

4.1 Hàm tapply

Ở chương này chúng ta sử dụng tập dữ liệu thảm thực vật từ một chương trình giám sát được thực hiện ở hai cộng đồng ôn đới, Vườn quốc gia Yellowstone và Khu Vực Quốc Gia Bision Range, Hoa Kỳ.

> setwd("C:/RBook/")

> Veg <- read.table(file="Vegetation2.txt", header= TRUE)

4.1.1 Tính giá trị trung bình từng lắt cắt

Một trong những điều đầu tiên chúng tôi muốn biết là liệu mức độ đa dạng trung bình từng lát cắt khác nhau. Đoạn mã dưới đây tính giá trị mức độ đa dạng trung bình, cũng như giá trị mức độ đa dạng có trung bình từng lát cắt khác nhau.

> m <- mean(Veg$R)

> m1<- mean(Veg$R[Veg$Transect == 1])

> m2<- mean(Veg$R[Veg$Transect == 2])

> m3<- mean(Veg$R[Veg$Transect == 3])

> m4<- mean(Veg$R[Veg$Transect == 4])

> m5<- mean(Veg$R[Veg$Transect == 5])

> m6<- mean(Veg$R[Veg$Transect == 6])

> m7<- mean(Veg$R[Veg$Transect == 7])

> m8<- mean(Veg$R[Veg$Transect == 8])

> c(m, m1, m2, m3, m4, m5, m6, m7, m8)

Kết quả:

[1] 9.965517 7.571429 6.142857 10.375000 9.250000

[6] 12.375000 11.500000 10.500000 11.833333

* + 1. Tính trung bình từng lát cắt hiệu quả hơn

Thật là rườm rà khi gõ tám lệnh để tính giá trị trung bình cho mỗi lát cắt. Hàm tapply thực hiện hoạt động tương tự như mã ở trên (từ m1 đến m8), nhưng với một dòng mã:

> tapply(Veg$R, Veg$Transect, mean)

Hoặc:

> tapply(X = Veg$R, INDEX = Veg$Transect, FUN = mean)

Hàm tapply tách dữ liệu của biến đầu tiên (R), dựa trên các mức của biến thứ hai (Transect). Đối với mỗi nhóm dữ liệu con, nó áp dụng một hàm, trong trường hợp này là giá trị trung bình, nhưng chúng ta cũng có thể sử dụng độ lệch chuẩn (hàm sd), phương sai (hàm var), độ dài (hàm length).

* 1. Hàm sapply, và hàm lapply

Để tính giá trị trung bình, tối thiểu, tối đa, độ lệch chuẩn và độ dài của từng thuộc tính. Có 20 thuộc tính trong tập dữ liệu này (từ 5–25). Tuy nhiên, chúng ta không cần gõ lệnh mean 20 lần. R cung cấp các chức năng khác tương tự như tapply để giải quyết tình huống này.

> sapply(Veg[, 5:9], FUN= mean)

Hoặc:

> lapply(Veg[, 5:9], FUN= mean)

Đầu ra của lapply được trình bày dưới dạng list, trong khi sapply cung cấp cho nó dưới dạng vectơ.

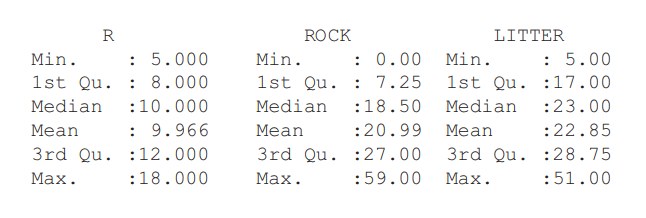
* 1. Hàm summary

Một hàm khác cung cấp thông tin cơ bản về các biến là hàm summary.

> Z <-cbind(Veg$R, Veg$ROCK, Veg$LITTER)

> colnames(Z) <- c("R", "ROCK", "LITTER")

> summary(Z)



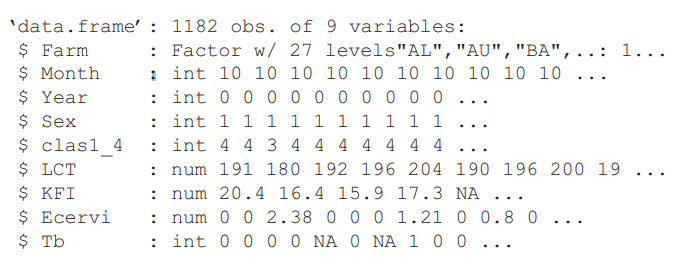
* 1. Hàm table

Sử dụng dữ liệu về hươu được lấy từ Vicente et al (2006).

> setwd("c:/RBook/")

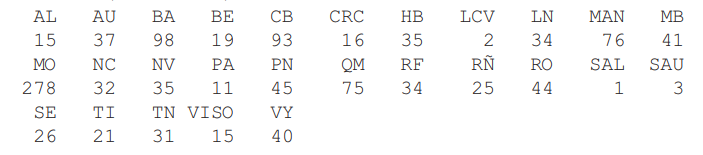
> Deer <- read.table(file="Deer.txt", header= TRUE)

> str(Deer)



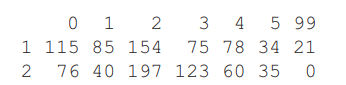
Tên mỗi trang trại đã được mã hóa là AL, AU và được nhập tự động. Số lượng quan sát mỗi trang trại thu được là:

> table(Deer$Farm)



Hay có thể sử dụng như sau:

> table(Deer$Sex, Deer$Year)



Năm 1999, các động vật chỉ có một giới tính đã được đo. Chúng tôi khuyên bạn nên luôn luôn sử dụng hàm table trước khi tương tác giữa cột dữ liệu định tính trong các mô hình hồi quy.

Chương 6: Hàm và vòng lặp

6.1 Giới thiệu vòng lặp

Giả sử bạn đang làm việc với một số lượng lớn các tập dữ liệu với nhiều thuộc tính, và mỗi người trong số họ mà bạn muốn tính toán một chỉ số nào đó. Nếu bạn may mắn, công thức cho chỉ số đa dạng bạn chọn đã được lập trình bởi người khác, và nếu bạn rất may mắn, nó có sẵn ở một trong những các gói, mã phần mềm được ghi chép đầy đủ, được kiểm tra đầy đủ và không có lỗi. Nhưng nếu bạn không thể tìm thấy mã phần mềm cho chỉ mục đa dạng đã chọn, đã đến lúc tự lập trình!

Nếu bạn có khả năng sử dụng một tập hợp các phép tính nhiều hơn một lần, bạn sẽ nên trình bày mã theo cách mà nó có thể được sử dụng lại mà không cần phải gõ lệnh nhiều lần. Điều này đưa bạn vào thế giới của các hàm và vòng lặp (và câu lệnh điều kiện chẳng hạn như lệnh if)

6.2 Vòng lặp

Để minh họa một tình huống sử dụng vòng lặp tiết kiệm thời gian đáng kể, chúng tôi sử dụng tập dữ liệu về hành vi xin thức ăn của các con cú trong tổ. Đối với ví dụ này, chúng tôi sử dụng '' thương lượng anh chị em '', được định nghĩa là số lần gọi bởi những con chim non trong khoảng thời gian 30 giây ngay trước khi bố mẹ xuất hiện. Giả sử rằng bạn đã được giao nhiệm vụ viết báo cáo về những dữ liệu này và để tạo ra một biểu đồ phân tán của “thương lượng anh chị em” theo thời gian bố mẹ đến cho mỗi tổ khác nhau, tốt nhất là ở định dạng jpeg. Có 27 tổ, vì vậy bạn sẽ cần tạo và lưu 27 biểu đồ. R có các công cụ để vẽ 27 biểu đồ phân tán trong một biểu đồ duy nhất (chúng tôi trình bày điều này trong Chương 8), nhưng giả sử rằng khách hàng đã yêu cầu rõ ràng 27 tệp jpeg riêng biệt. Đây không phải là điều bạn sẽ muốn làm theo cách thủ công.

6.2.1 Hãy là kiến trúc sư cho mã của bạn

Trước khi viết mã, bạn sẽ cần lập kế hoạch và tạo ra một kiến trúc thiết kế phác thảo các bước trong nhiệm vụ của bạn:

Bước 1: Nhập dữ liệu và tự làm quen với các tên biến, sử dụng các lệnh read.table, names và str.

Bước 2: Trích xuất dữ liệu của một tổ và lập biểu đồ phân tán của “thương lượng anh chị em” so với thời gian bố mẹ đến cho tập hợp con này.

Bước 3: Thêm tiêu đề hình và các nhãn thích hợp dọc theo trục x- và y. Tên của tổ phải nằm trong tiêu đề chính.

Bước 4: Trích xuất dữ liệu từ tổ thứ hai và xác định những sửa đổi nào đối với đồ thị gốc là cần thiết.

Bước 5: Xác định cách lưu đồ thị vào tệp jpeg.

Bước 6: Viết một vòng lặp để trích xuất dữ liệu cho tổ i, vẽ biểu đồ dữ liệu từ tổ i và lưu vẽ biểu đồ thành tệp jpeg có tên dễ nhận dạng.

Nếu bạn có thể triển khai thuật toán này, bạn là một kiến trúc sư giỏi!

6.2.2 Bước 1: Nhập dữ liệu

> setwd("C:/RBook/")

> Owls <- read.table(file = "Owls.txt", header = TRUE)

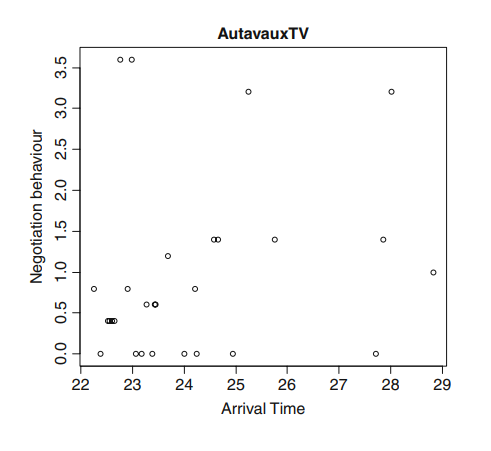
6.2.3 Bước 2, 3: Tạo biểu đồ phân tán và thêm nhãn

Chúng tôi gọi dữ liệu được trích xuất cho tổ này là Owls.ATV, trong đó ATV đề cập đến tên tổ. Biểu đồ phân tán này thể hiện thời gian đến so với “thương lượng anh chị em” trong tập dữ liệu Owls.ATV

> Owls.ATV <- Owls[Owls$Nest=="AutavauxTV", ]

> Owls.ATV <- Owls[Owls$Nest == "AutavauxTV", ]

> plot(x = Owls.ATV$ArrivalTime, y = Owls.ATV$NegPerChick, xlab = "Arrival Time", main = "AutavauxTV", ylab = "Negotiation behaviour")



6.2.4 Bước 4: Thiết kế mã chung

Thực hiện cùng một với dữ liệu từ tổ khác. Mã cho tổ thứ hai chỉ cần sửa từ AutavauxTV bằng Bochet.

> Owls.Bot <- Owls[Owls$Nest == "Bochet", ]

> plot(x = Owls.Bot$ArrivalTime, y = Owls.Bot$NegPerChick, xlab = "Arrival Time", ylab = "Negotiation behaviour", main = "Bochet")

Đổi tên Owls.Bot sang Owls.i

> Owls.i <- Owls[Owls$Nest == "Bochet", ]

> plot(x = Owls.i$ArrivalTime, y = Owls.i$NegPerChick, xlab = "Arrival Time", ylab = "Negotiation behaviour", main = "Bochet")

Gán tên tổ vào biến Nest.i và thay đổi “Bochet” bằng Nest.i

> Nest.i <- "Bochet"

> Owls.i <- Owls[Owls$Nest == Nest.i, ]

> plot(x = Owls.i$ArrivalTime, y = Owls.i$NegPerChick, xlab = "Arrival Time", main = Nest.i, ylab = "Negotiation behaviour")

6.2.5 Bước 5 Lưu đồ thị

Chọn thư mục lưu các file ảnh, rồi tạo ra file jpeg mới bằng hàm jpeg rồi sau khi vẽ đồ thị thì sử dụng dev.off() để lưu đồ thì vừa vẽ đó vào file ảnh

> setwd("C:/AllGraphs/")

> Nest.i <- "Bochet"

> Owls.i <- Owls[Owls$Nest == Nest.i, ]

> YourFileName <- paste(Nest.i, ".jpg", sep="")

> jpeg(file = YourFileName)

> plot(x = Owls.i$ArrivalTime, y = Owls.i$NegPerChick, xlab = "Arrival Time", main = Nest.i, ylab = "Negotiation behaviour")

> dev.off()

6.2.6 Bước 6: Xây dựng vòng lặp

Sử dụng hàm unique để lấy tên tất cả các tổ rồi lưu vào một biến, sau đó cho thực hiện vòng lặp thì tên của tổ được thay đổi sau một lần lặp và vì có 27 tổ nên ta sẽ cho lặp 27 lần.

> AllNests <- unique(Owls$Nest)

> for (i in 1:27){

Nest.i <- AllNests[i]

Owls.i <- Owls[Owls$Nest == Nest.i, ]

YourFileName <- paste(Nest.i, ".jpg", sep = "")

jpeg(file = YourFileName)

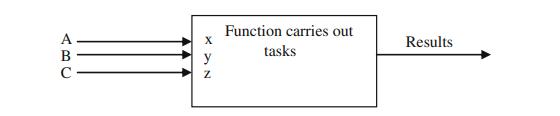
plot(x = Owls.i$ArrivalTime, y = Owls.i$NegPerChick, xlab = "Arrival Time", ylab = "Negotiation behaviour", main = Nest.i)

dev.off()

}

6.3 Hàm

Khái niệm của hàm được thể hiện qua hình sau:



Đầu vào của hàm là một tập hợp các biến A, B và C, có thể là vectơ, ma trận, data frame hoặc danh sách. Sau đó, nó thực hiện các tính toán được lập trình và trả về thông tin cho người dùng.

6.3.1 Giá trị 0 và Nas

Trước khi thực hiện phân tích thống kê, điều quan trọng là phải xác định vị trí và xử lí các giá trị thiếu, vì chúng có thể gây ra một số khó khăn. Một số kỹ thuật, chẳng hạn như hồi quy tuyến tính, sẽ loại bỏ bất kỳ trường hợp nào (quan sát) có chứa giá trị bị thiếu. Các biến có nhiều số 0 cũng gây ra rắc rối, đặc biệt là trong phân tích đa biến để có thể minh họa cách sử lí những dữ liệu này qua tập dữ liệu về thảm thực vật ở chương 4.

> setwd("C:/RBook/")

> Veg <- read.table(file = "Vegetation2.txt", header = TRUE)

Giả sử bạn muốn một hàm tính toán số lượng giá trị bị thiếu trong mỗi biến. Cú pháp của một hàm như vậy là:

> NAPerVariable <- function(X1) {

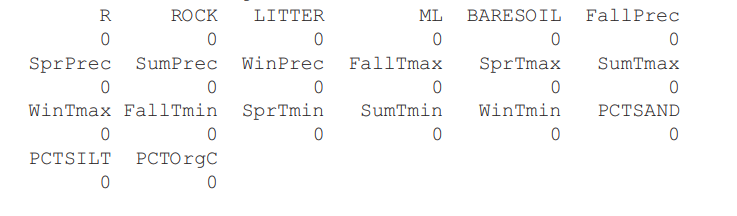
D1 <- is.na(X1)

colSums(D1)

}

Sau đó gọi hàm vừa tạo:

> NAPerVariable(Veg[,5:24])



Chúng tôi đã bỏ qua bốn cột đầu tiên, vì những cột này chứa thông tin về lát cắt và thời gian. Dường như không có giá trị nào bị thiếu trong danh sách. Tham số của hàm là X1. Chúng tôi giả định rằng các biến nằm trong cột và các quan sát theo hàng. Lệnh is.na (X1) tạo Boolean ma trận có cùng bậc với X1, với giá trị TRUE nếu tương ứng phần tử của X1 là một giá trị bị thiếu và FALSE nếu không. Hàm colSums là một hàm nhận tổng các phần tử trong mỗi cột. Thông thường, colSums được áp dụng cho ma trận dữ liệu với các số, nhưng nếu nó được áp dụng sang ma trận Boolean, nó chuyển đổi TRUE thành 1 và FALSE thành 0. Kết quả là đầu ra của colSums (D1) là số lượng giá trị còn thiếu trên mỗi biến. Nếu bạn thay thế hàm colSums bằng hàm rowSums, thì hàm của bạn sẽ cung cấp số lượng giá trị bị thiếu cho mỗi lần quan sát.

6.3.2 Thông tin kĩ thuật

Bạn không nên sử dụng các tên bên trong một hàm cũng đang tồn tại bên ngoài hàm. Ví dụ: nếu bạn sử dụng D1 <- is.na (X) thay vì D1 <- is.na (X1), R sẽ xem xét đầu tiên bên trong hàm để tìm các giá trị của X. Nếu nó không tìm thấy biến này bên trong hàm, nó sẽ nhìn ra bên ngoài hàm. Nếu như một biến tồn tại bên ngoài hàm, R sẽ sử dụng nó mà không cần nói cho bạn biết. Thay vì tính toán số lượng giá trị bị thiếu trong biến Veg, nó sẽ hiển thị cho bạn số lượng giá trị còn thiếu trong X, bất kể X có thể là gì.

R sẽ trả về kết quả dòng cuối cùng của hàm. Hàm NAPerVariable có colSums (D1) ở dòng cuối cùng, vì vậy đây là thông tin được trả về.

Một lưu ý khác nữa là bạn nên chú thích chức năng và giá trị trả về cũng như tham số của hàm (Sử dụng #)

6.3.3 Ví dụ thứ hai về giá trị 0 và Nas

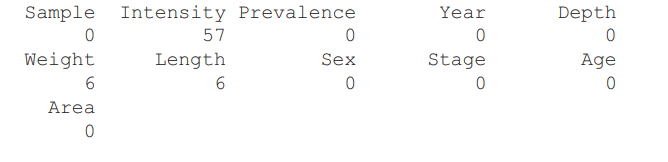
Hemmingsen và cộng sự. (2005) đã kiểm tra một số lượng lớn cá tuyết để tìm trypanosome nhiễm trùng trong các chuyến du lịch hàng năm dọc theo bờ biển Finnmark ở Bắc Na Uy. Chúng tôi sử dụng dữ liệu của họ ở đây. Dữ liệu bao gồm sự hiện diện hay vắng mặt của ký sinh trùng trong cá cũng như số lượng ký sinh trên mỗi con cá.

> setwd("c:/RBook/")

> Parasite <- read.table(file = "CodParasite.txt", header = TRUE)

Sử dụng lại hàm đã xây dựng ở trên cho data frame này

> NAPerVariable(Parasite)



Và để xác định có bao nhiêu số không trong mỗi biến, đặc biệt là trong Intensity thì chúng ta cũng nên xây dựng một hàm tương tự như trên:

> ZerosPerVariable <- function(X1) {

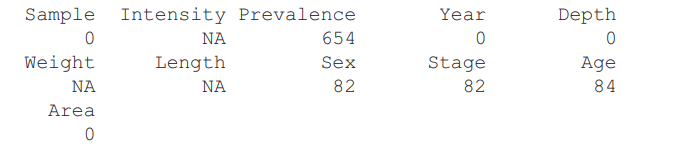
D1 = (X1 == 0)

colSums(D1)

}

Sau đó gọi lại hàm:

> ZerosPerVariable(Parasite)



Ta thấy kết quả có các giá trị NA tại vì các cột như Length, Weight, Intensity có các dữ liệu trống, ta cần giải quyết vấn đề đó như sau:

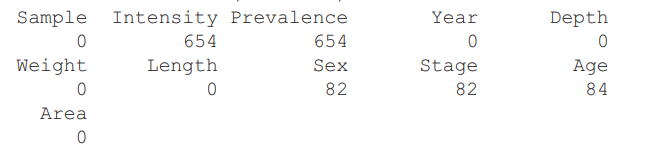
> ZerosPerVariable <- function(X1) {

D1 = (X1 == 0)

colSums(D1, na.rm = TRUE)

}

> ZerosPerVariable(Parasite)



6.3.4 Hàm với nhiều tham số

Trong phần trước, chúng ta đã tạo hai hàm, một hàm để xác định số trong số các giá trị bị thiếu cho mỗi biến và một giá trị khác để tìm số lượng giá trị không trên mỗi Biến đổi. Trong phần này, chúng tôi kết hợp chúng và yêu cầu hàm tính toán tổng số quan sát bằng 0 hoặc số quan sát bằng NA.

> VariableInfo <- function(X1, Choice1) {

if (Choice1 == "Zeros"){ D1 = (X1 == 0) }

if (Choice1 == "NAs") { D1 <- is.na(X1)}

colSums(D1, na.rm = TRUE)

}

Hàm có hai đối số: X1 và Choice1. Như trước đây, X1 nên chứa một data frame và Choice1 là một biến phải chứa giá trị ‘‘ Zeros ’’ hoặc ‘‘ Nas ”.

> VariableInfo(Parasite, "Zeros")

Hoặc:

> VariableInfo(Parasite, "NAs")

6.3.5 Hàm hoàn chỉnh

6.3.5.1 Tham số mặc định của hàm

Biến Choice1 có thể được cung cấp một giá trị mặc định để nếu bạn không nhập một giá trị cho Choice1, hàm sẽ thực hiện các phép tính cho giá trị mặc định.

> VariableInfo <- function(X1, Choice1 = "Zeros") {

if (Choice1 == "Zeros"){ D1 = (X1 == 0) }

if (Choice1 == "NAs") { D1 <- is.na(X1)}

colSums(D1, na.rm = TRUE)

}

6.3.5.2 Sai chính tả

Chúng tôi cũng muốn một hàm thực thi mã thích hợp, tùy thuộc vào giá trị của Choice1 và đưa ra thông báo cảnh báo nếu Choice1 không bằng ‘‘ Zeros ’’ hoặc ‘‘ NAs ’’.

VariableInfo <- function(X1, Choice1 = "Zeros") {

if (Choice1 == "Zeros"){ D1 = (X1 == 0) }

if (Choice1 == "NAs") { D1 <- is.na(X1)}

if (Choice1 != "Zeros" & Choice1 != "NAs") {

print("You made a typo")

} else {

colSums(D1, na.rm = TRUE)}

}

Khi bạn thử gọi hàm với Choice1 không phải là “Zeros” hoặc “NAs”

> VariableInfo(Parasite, "abracadabra")

Hoặc ta có thể sử dụng hàm sau:

> ifelse(Choice1 == "Zeros", D1 <- (X1 == 0), D1 <- is.na(X1))

Kết quả:

[1] "You made a typo"

6.4 Hàm và câu điều kiện

Viện chính phủ Hà Lan RIKZ đã thực hiện một chương trình lấy mẫu sinh vật đáy biển ở mùa hè năm 2002. Dữ liệu về khoảng 75 loài sinh vật đáy biển được thu thập tại 45 điểm trên 9 bãi biển dọc theo bờ biển Hà Lan.

6.4.1 Trở thành nhà thiết kế một lần nữa

Cũng giống như ví dụ trước được trình bày trong chương này, hãy bắt đầu bằng cách phác thảo các nhiệm vụ cần thực hiện.

Bước 1: Nhập dữ liệu và điều tra những gì bạn có về các loại biến, tên biến, số cột, dòng của dữ liệu, ...

Bước 2: Tính toán tổng mức độ phong phú cho địa điểm 1. Lặp lại quy trình này cho địa điểm thứ 2. Tự động hóa điều này quy trình, làm cho mã càng chung chung càng tốt.

Bước 3: Tính toán số lượng loài khác nhau cho địa điểm 1. Lặp lại quy trình này cho địa điểm thứ 2. Tự động hóa quá trình này và làm cho mã càng chung chung càng tốt.

Bước 4: Làm tương tự đối với chỉ số Shannon.

Bước 5: Kết hợp mã và sử dụng câu lệnh if để chọn giữa các chỉ số cần tính.

Bước 6. Đặt tất cả mã vào một hàm và cho phép người dùng chọn dữ liệu và

chỉ số cần tính.

6.4.2 Bước 1: Nhập và đánh giá dữ liệu

> Benthic <- read.table("C:/RBook/RIKZ.txt", header = TRUE)

6.4.3 Bước 2: Tổng số sinh vật trên mỗi địa điểm

Tính tổng tất cả các sinh vật ở vị trí 1:

> sum(Species[1, ], na.rm = TRUE)

Tương tự với vị trí 2:

> sum(Species[2, ], na.rm = TRUE)

Sử dụng vòng lặp để tính tất cả các vị trí:

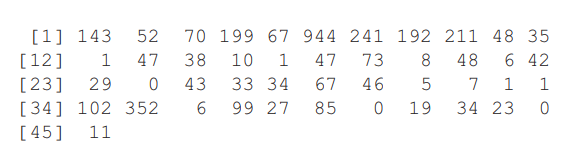
> TA <- vector(length = n[1])

> for (i in 1:n[1]){

TA[i] <- sum(Species[i, ], na.rm = TRUE)

}

> TA



Để tối ưu hơn ta có thể sử dụng:

> TA <- rowSums(Species, na.rm = TRUE)

> TA

6.4.4 Bước 3: Tổng số loài trên mỗi địa điểm

Tính tổng tất cả các loài ở vị trí 1:

> sum(Species[1, ] > 0, na.rm = TRUE)

Tương tự với vị trí 2:

> sum(Species[2, ] > 0, na.rm = TRUE)

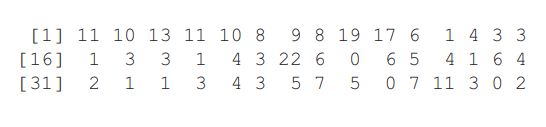
Sử dụng vòng lặp ở để tính số loại vật tại một địa điểm

> Richness <- vector(length = n[1])

> for (i in 1:n[1]){

Richness[i] <- sum(Species[i, ] > 0, na.rm = TRUE) }

> Richness



Hoặc có thể sử dụng hàm:

> Richness <- rowSums(Species > 0, na.rm = TRUE)

> Richness

6.4.5 Bước 4: Chỉ số Shannon tại mỗi địa điểm

> RS <- rowSums(Species, na.rm = TRUE)

> prop <- Species / RS

> H <- -rowSums(prop \* log10(prop), na.rm = TRUE)

> H

6.4.6 Bước 5: Kết hợp mã

> Choice <- "Richness"

> if (Choice == "Richness") { Index <- rowSums(Species >0, na.rm = TRUE)}

> if (Choice == "Total Abundance") { Index <- rowSums(Species, na.rm = TRUE) }

> if (Choice == "Shannon") {

RS <- rowSums(Species, na.rm = TRUE)

prop <- Species / RS

Index <- -rowSums(prop\*log10(prop), na.rm = TRUE)}

6.4.7 Bước 6: Đưa mã vào hàm

> Index.function <- function(Spec, Choice1){

if (Choice1 == "Richness") { Index <- rowSums(Spec > 0, na.rm = TRUE)}

if (Choice1 == "Total Abundance") { Index <- rowSums(Spec, na.rm = TRUE) }

if (Choice1 == "Shannon") {

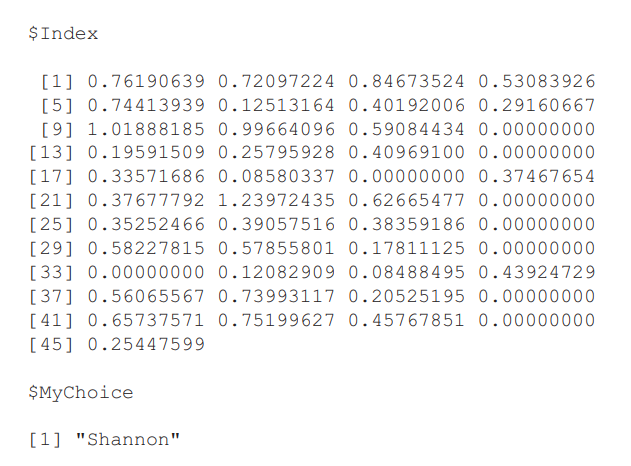
RS <- rowSums(Spec, na.rm = TRUE)

prop <- Spec / RS

Index <- -rowSums(prop \* log10(prop), na.rm = TRUE)}

list(Index = Index, MyChoice = Choice1) }

> Index.function(Species, "Shannon")



Hoặc bạn có thể nâng cấp thêm chức năng báo lỗi khi không có phương thức tính mà người dùng truyền vào:

Index.function <- function(Spec,Choice1){

if (Choice1 == "Richness") { Index <- rowSums(Spec > 0, na.rm = TRUE) }

else if (Choice1 == "Total Abundance") { Index <- rowSums(Spec, na.rm = TRUE) }

else if (Choice1 == "Shannon") {

RS <- rowSums(Spec, na.rm = TRUE)

prop <- Spec / RS Index <- -rowSums(prop\*log(prop),na.rm=TRUE)

} else { print("Check your choice") Index <- NA }

list(Index = Index, MyChoice = Choice1)}