

Buổi 0. Thực hành XSTK cơ bản

NGUYỄN THÀNH DUY

Ngày 8 tháng 10 năm 2025

- Mục tiêu chính: Làm quen R.
- Nội dung thực hành: Giao diện RStudio, các thao tác với kiểu dữ liệu và cấu trúc dữ liệu trong R.

1 Giới thiệu R và RStudio

1.1 Giới thiệu

R là một gói phần mềm thống kê có nhiều điểm tương đồng với ngôn ngữ lập trình thống kê S. Phiên bản sơ bộ của S được Bell Labs tạo ra vào những năm 1970, được thiết kế để trở thành một ngôn ngữ lập trình tương tự C nhưng dành cho thống kê. John Chambers là một trong những người phát minh ra ngôn ngữ này, và ông đã giành được Giải thưởng của Hiệp hội Máy tính năm 1999 cho ngôn ngữ này. R là một phần mềm hoàn toàn miễn phí.

R và RStudio có mối liên hệ chặt chẽ — chúng không phải là cùng một phần mềm, mà là hai lớp khác nhau của cùng một hệ thống làm việc.

R là ngôn ngữ lập trình và môi trường tính toán thống kê. Nó được phát triển để:

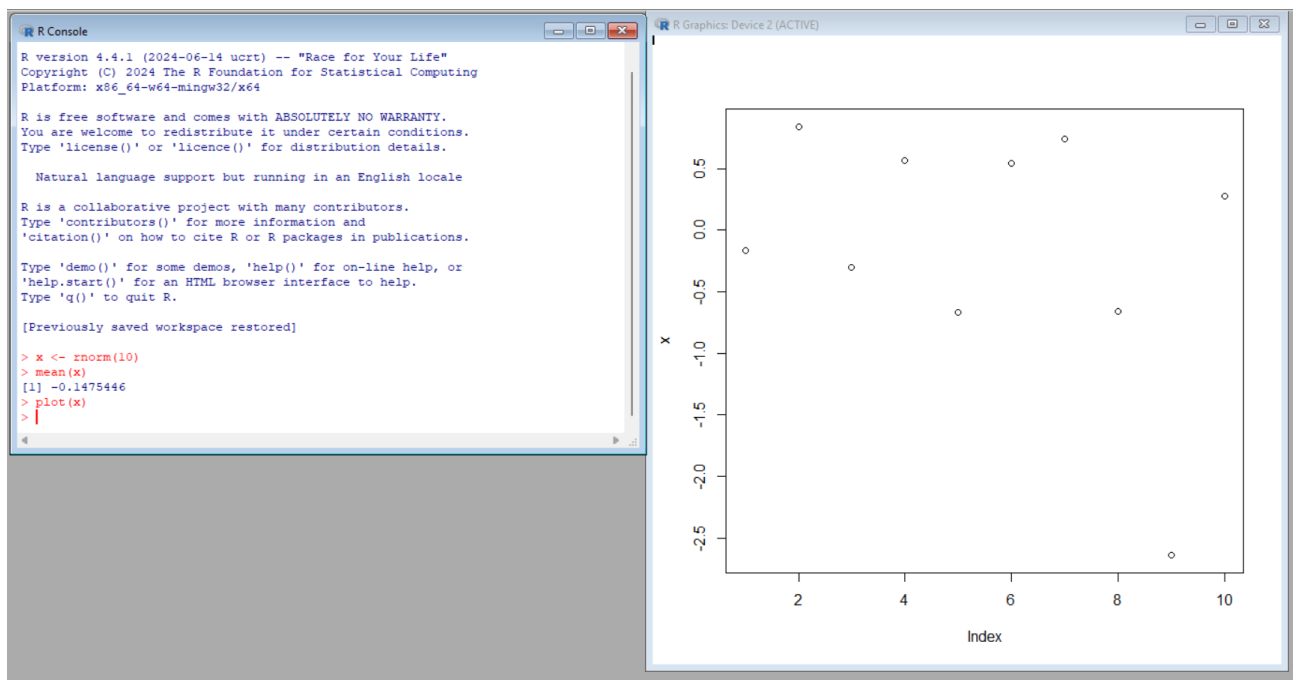
- Xử lý dữ liệu,
- Phân tích thống kê,
- Mô phỏng ngẫu nhiên,
- Vẽ đồ thị và trực quan hóa dữ liệu.

Khi cài R, người dùng đã có thể chạy lệnh trong giao diện dòng lệnh (console) đơn giản của R, nhưng giao diện này rất thô sơ. Mở R (không dùng RStudio), sẽ thấy một cửa sổ đen hoặc trắng, nơi chỉ có thể gõ lệnh kiểu:

```
x <- rnorm(10)
mean(x)
plot(x)
```

RStudio là một môi trường phát triển tích hợp (IDE - Integrated Development Environment) dành riêng cho R. Nó không thay thế R, mà chạy dựa trên R. RStudio giúp người dùng làm việc với R dễ dàng và trực quan hơn thông qua:

- Source Editor: viết, lưu và chạy script `.R`.
- Console: chạy lệnh trực tiếp.
- Environment: xem toàn bộ biến, dataset đang tồn tại.



Hình 1.1: Giao diện thô sơ của R

- Plots: xem và lưu biểu đồ.
- Help / Packages / Files: quản lý gói, tìm tài liệu, duyệt file.

Trong học phần này, chúng ta sẽ sử dụng R trong RStudio.

1.2 Cài đặt R và RStudio

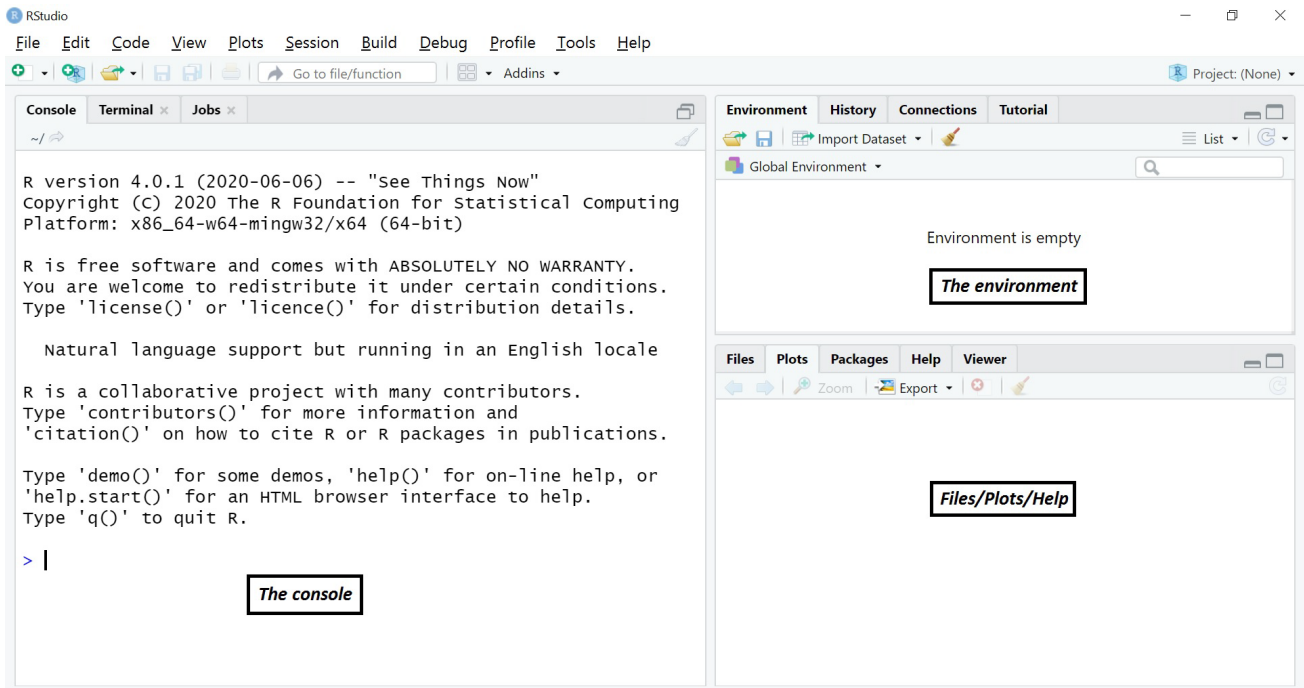
Bước 1. Cài R từ <https://cran.rstudio.com/>

Bước 2. Cài RStudio từ <https://posit.co/download/rstudio>

Bước 3. Khi mở RStudio, phần mềm sẽ tự động tìm bản cài R trong máy và chạy R ở nền.

1.3 Giao diện của RStudio

Lần đầu tiên mở RStudio, sẽ thấy một cửa sổ gồm ba bảng điều khiển riêng biệt: một bảng bên trái và hai bảng bên phải. Đó là bảng điều khiển (*The console*), bảng môi trường (*The environment*) và bảng tệp/biểu đồ/trợ giúp (*Files/ Plots/ Help*) như minh họa bên dưới:



Hình 1.2: Giao diện cơ bản của RStudio

Mỗi bảng (panel) này đều có một mục đích sử dụng cụ thể. Chi tiết hơn về các mục đích này được trình bày dưới đây:

Bảng Console

The console là bảng ở phía bên trái cửa sổ. Văn bản ở đầu console khi mở RStudio lần đầu tiên sẽ mô tả phiên bản R đang chạy trong RStudio. Có thể nhập code trực tiếp vào bảng điều khiển bên cạnh mũi tên `>` hướng sang phải. Tuy nhiên, không nên làm như vậy vì không thể lưu nội dung code của console, do đó sẽ mất mọi phân tích đã thực hiện. Thay vào đó, nên sử dụng một file R script, sẽ được giới thiệu sau.

Bảng Environment

Bảng environment có nhiều tab, nhưng tab được sử dụng phổ biến nhất là tab có nhãn *Environment*. Tab này hiển thị những đối tượng có sẵn trong không gian làm việc. Khi mở R lần đầu, nó sẽ trống (vì chưa có hoạt động nào). Sao chép và dán dòng lệnh sau vào bảng console (bên cạnh biểu tượng `>`), sau đó nhấn Enter để chạy:

```
x <- 4
```

Lúc này, sẽ thấy rằng có một mục trong tab *Environment*. Đoạn code trên mô tả cấu trúc cơ bản để tạo một đối tượng trong R. Tên đối tượng nằm ở phía bên trái, trong trường hợp này, chúng ta đang tạo một đối tượng mà chúng ta có thể gọi là `x`. `<-` là toán tử gán, vì vậy trong trường hợp này, chúng ta đang gán giá trị 4 cho một đối tượng có tên là `x`. Nhập `x` vào console và nhấn Enter. Nếu đã chạy đoạn code trước đó, sẽ thấy giá trị 4 được in ra.

Tab thứ hai là tab *History*, tab này hiển thị code đã được chạy trong phiên RStudio hiện tại. Nếu đã chạy dòng code trước đó, khi mở tab *History*, nó sẽ chỉ hiển thị `x <- 4`.

Chức năng trong tab *Connections* liên quan đến một số chức năng R nâng cao hơn (như kết nối với cơ sở dữ liệu) và sẽ không xem xét điều này trong lớp học.

Tab cuối cùng là tab *Tutorial*, cung cấp các bộ hướng dẫn tương tác cho các phần khác nhau của lập trình R, giúp người dùng học R tương tác trực tiếp trong chính RStudio. Cụ thể, nó cho phép làm các bài hướng dẫn có tương tác (interactive tutorials) — gồm cả phần lý thuyết, ví dụ, bài tập code, câu hỏi trắc nghiệm — ngay bên trong giao diện RStudio, không cần mở trình duyệt hay phần mềm ngoài. Đây là một kênh hỗ trợ trong việc tự học R.

Files/Plots/Help

Bảng này cũng có nhiều tab.

- Tab đầu tiên là tab *Files*, hoạt động như một trình duyệt tệp. Có thể điều hướng qua các tệp và mở các tệp tương thích trực tiếp từ bảng này.
- Tab thứ hai là tab *Plots*. Bất kỳ biểu đồ nào được tạo sẽ được hiển thị ở đây. Có thể điều hướng qua các biểu đồ và xuất biểu đồ thành các file khác từ tab này.
- Tab thứ ba là tab *Packages*, hiển thị những gói R nào đã cài đặt và những gói nào đang được tải. Các gói R được sử dụng để mở rộng chức năng của R cơ bản.
- Tab thứ tư là tab *Help*. Có thể tìm trợ giúp về bất kỳ hàm nào trong R bằng cách nhập ? theo sau là tên hàm vào bảng điều khiển và nhấn Enter. Các tài liệu hỗ trợ/ trợ giúp/ hướng dẫn cho câu lệnh sẽ hiển thị bên dưới tab *Help*.
- Tab cuối cùng *Viewer* tương ứng với một số tính năng nâng cao của R và sẽ không được thảo luận ở đây.

1.4 Sử dụng file R script

Như đã đề cập ở trang trước, có thể nhập và chạy code trực tiếp trong Console. Tuy nhiên, đây không phải là một ý tưởng hay vì một số lý do:

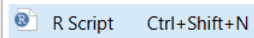
1. Khó theo dõi những gì đang thực hiện (đặc biệt nếu đang thực hiện rất nhiều dòng code!)
2. Không thể lưu bất cứ thứ gì bạn nhập vào console.

Vì lý do này, nếu đang thực hiện bất kỳ phân tích nào, việc sử dụng file R script rất quan trọng. Về cơ bản, đây là một file văn bản để nhập code vào và lưu lại để sử dụng sau này.

Mở một file script

Cách dễ nhất để mở một file script mới là nhấp vào biểu tượng trông giống như một tờ giấy



. Thao tác này sẽ mở ra một menu, chọn tùy chọn trên cùng . Hoặc có thể sử dụng phím tắt: CTRL+SHIFT+N trên Windows (Command+SHIFT+N trên Mac).

Sau khi một file script mới được mở, một bảng điều khiển thứ tư (phía trên bên trái) sẽ mở ra và đây sẽ chỉ là một tài liệu văn bản trống.

Sử dụng script để thực thi code


Trong bảng script, có thể nhập bất kỳ dòng code nào. Nếu sao chép code từ bảng console vào file script, hãy đảm bảo xóa ký hiệu > ở đầu mỗi dòng đã sao chép. Ví dụ, hãy sao chép và dán đoạn code sau (tạo hai đối tượng x và y rồi cộng chúng lại với nhau) vào một file R script trống:

```
x <- 10
y <- 11
x + y
```

Để chạy code, hãy chọn code muốn chạy và phím tắt là CTRL+ENTER trên máy Windows hoặc Command+ENTER trên máy Mac. Để chạy code từng dòng, hãy đảm bảo con trỏ đang nhấp nháy ở dòng muốn chạy, sau đó sử dụng phím tắt.

Khi chạy lệnh này, sẽ thấy từng dòng code xuất hiện trong bảng điều khiển (mỗi dòng sẽ nằm cạnh dấu >). Khi dòng code cuối cùng được chạy, sẽ thấy giá trị được tính toán (21) - điều này là do chúng ta chưa gán giá trị này cho bất kỳ mục nào bằng <-.

Lưu một file script

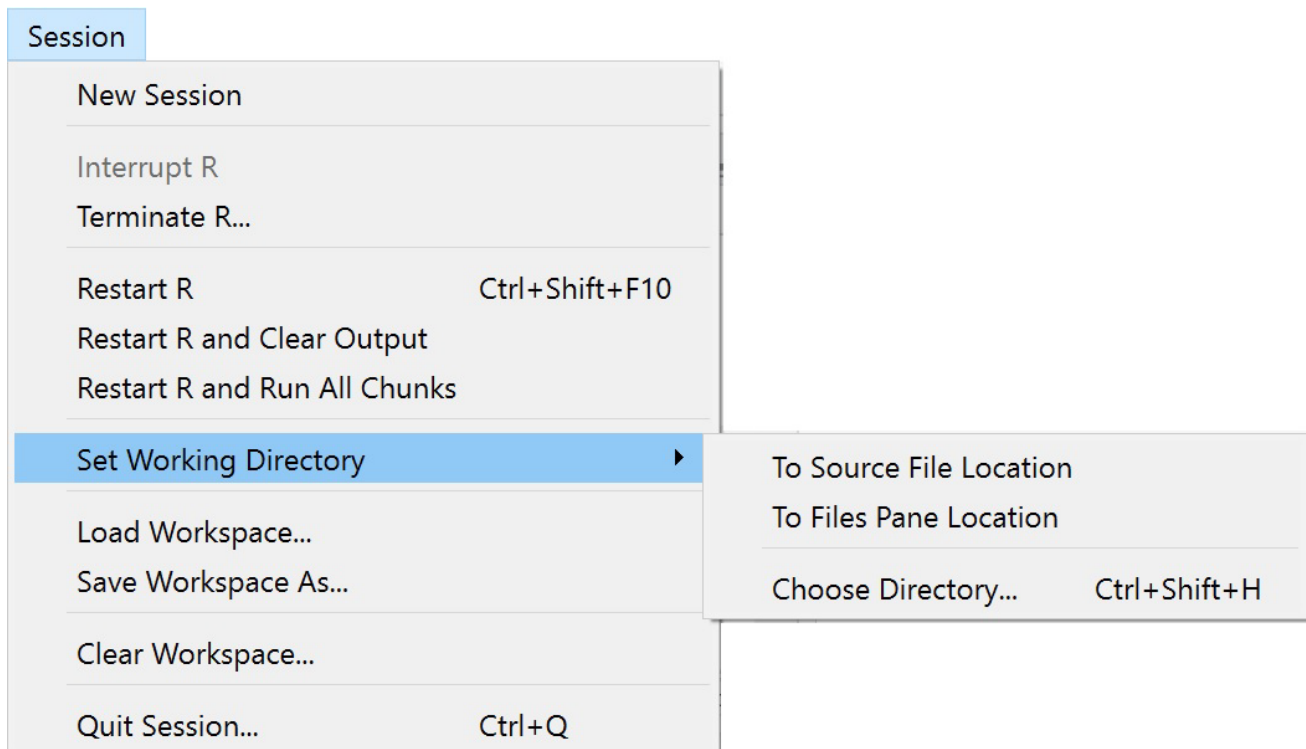
Trong khi phân tích bằng R, người dùng sẽ muốn lưu file script trong quá trình thực hiện. Có thể làm điều này bằng cách nhấp vào biểu tượng lưu  ở đầu bảng điều khiển. Nếu chưa lưu trước đó, thao tác này sẽ mở trình duyệt tệp để có thể chọn nơi lưu tệp.

1.5 Working Directory - Thư mục làm việc

Thiết lập working directory rất cần thiết khi làm việc trong R. Nên thực hiện điều này đầu tiên khi mở R. Thư mục làm việc là vị trí thực hiện các phân tích.

Thiết lập Working Directory

Theo mặc định, working directory sẽ là folder *Documents* trong máy tính. Thông thường, tốt nhất là nên có một thư mục riêng để làm việc hoặc phân tích. Trong trường hợp này, sẽ cần thiết lập working directory. Sử dụng menu ở đầu cửa sổ, vào *Session > Set Working Directory*. Bạn sẽ thấy ba tùy chọn trong menu phụ tiếp theo (xem bên dưới).



Hình 1.3: Thiết lập Working Directory

Tùy chọn đầu tiên (*To Source File Location*) sẽ đặt working directory của vào thư mục lưu tập lệnh. Sử dụng tùy chọn này nếu tôi thực hiện một phân tích nhỏ (chỉ cần một file script) với một tập dữ liệu, sau đó sẽ thiết lập một thư mục chứa tập lệnh và dữ liệu ở cùng một nơi.

Tùy chọn thứ hai (*To Files Pane Location*) sẽ đặt thư mục làm việc vào vị trí được chỉ định trong ngăn Tệp (tab *Files* ở phần 1.3).

Nếu tôi muốn tự tay chọn thư mục đang sử dụng cho dữ liệu, chọn tùy chọn thứ ba (*Choose Directory. . .*). Tùy chọn này sẽ mở ra một cửa sổ trình duyệt tệp để có thể điều hướng đến bất kỳ thư mục nào. Thực hiện tùy chọn này cho các phân tích dài hơn với nhiều tập lệnh và/hoặc tập dữ liệu (tương tự như một báo cáo dài hoặc luận văn), trong đó lưu trữ các tập lệnh trong một thư mục và dữ liệu trong một thư mục khác.

Thông báo lỗi

Nếu không thiết lập working directory, có thể thấy rằng khi bạn cố gắng sử dụng dữ liệu, sẽ gặp lỗi sau:

```
read.csv("Data.csv")
## Warning in file(file, "rt"): cannot open file 'Data.csv': No
such file or directory
## Error in file(file, "rt"): cannot open the connection
```

Lưu ý rằng lỗi đầu tiên hiển thị *"No such file or directory"*. Đây là cách nói của R, có nghĩa là R không tìm thấy tệp đang tìm kiếm trong working directory nên R sẽ dừng tìm kiếm ngay bây giờ. Nếu thấy thông báo lỗi này, hãy kiểm tra xem thiết lập working directory chưa. Nếu đã làm điều này và vẫn nhận được thông báo, hãy kiểm tra xem tệp đã ở đúng vị trí chưa và đã nhập đúng tên tệp vào R chưa.

Một số câu lệnh liên quan đến `working directory`

Dưới đây là một số câu lệnh mà khi nhập trực tiếp vào console sẽ hiển thị hoặc thay đổi các thông tin về `working directory`.

- `getwd()`: Kiểm tra thư mục làm việc hiện tại.
- `setwd('path')`: Thiết lập (thay đổi) thư mục làm việc.
- `list.files()` hoặc `dir()`: Xem các files trong thư mục làm việc.
- `dir.create('folder')`: Tạo thư mục mới.

1.6 Sử dụng các gói lệnh (packages) trong R

R là một ngôn ngữ lập trình mã nguồn mở, nghĩa là các hàm có sẵn trong R đều do người dùng R đóng góp. Vì lý do này, R có phạm vi chức năng rộng trong toán học, thống kê và khoa học dữ liệu. Tuy nhiên, để thực hiện các hàm mở rộng, chúng ta sẽ cần thao tác với các gói lệnh - packages trong R.

Gói là tập hợp mã, hàm và tài liệu hướng dẫn cho phép thực hiện các chức năng không được bao hàm trong các gói R cơ bản được tải sẵn khi khởi động. Các gói này thường bao gồm một quy trình hoặc phân tích khá cụ thể. Nhìn chung, nếu muốn thực hiện một phân tích cụ thể mà không thể thực hiện bằng các lệnh cơ bản, hãy sử dụng Google để tìm gói hoặc hàm phù hợp. Nhập "[tên phân tích] in R" và nhận được kết quả (ví dụ, nếu tôi muốn thực hiện mô hình mạng nơ-ron, tôi sẽ nhập "Neural Network in R"). Hãy tìm kết quả có CRAN trong địa chỉ web.

<https://cran.r-project.org> › package=neuralnet ⋮

Package neuralnet - CRAN

7 Feb 2019 — **neuralnet**: Training of **Neural Networks** (2005). The **package** allows flexible settings through custom-choice of error and activation function. ...

CRAN checks: neuralnet results

Version: 1.44.2

<https://cran.r-project.org> › web › packages › neur... PDF ⋮

Package 'neuralnet'

by S Fritsch · 2019 · Cited by 83 — Train **neural networks** using backpropagation, resilient backpropagation (RPROP) with (Riedmiller, 1994) or without weight backtracking (...

Điều này cho thấy có một gói cho phân tích neural network trong R được gọi là `neuralnet`.

Cài đặt Packages

Để sử dụng các gói này, trước tiên cần cài đặt nó. Để kiểm tra xem gói đã được cài đặt hay chưa, hãy tìm tên của gói trong tab *Packages* trong bảng điều khiển tệp của RStudio. Có hai cách để cài đặt gói. Cách đầu tiên là sử dụng một dòng code duy nhất:

```
install.packages("neuralnet")
```

Có thể thay thế `neuralnet` bằng bất kỳ tên gói nào hoặc một vector ký tự tên gói nếu muốn cài đặt nhiều gói.

Cách tiếp theo là sử dụng menu trong RStudio. Vào *Tools > Install Packages* và nhập tên gói muốn cài đặt.

Tải Packages

Sau khi cài đặt gói phù hợp, sẽ cần tải gói đó trước khi có thể sử dụng. Để làm điều này, hãy sử dụng hàm `library()` hoặc `require()`. Vì vậy, để tải gói `neuralnet`, có thể chạy một trong hai dòng mã sau:

```
# sử dụng
library(neuralnet)

# hoặc
require(neuralnet)
```

Sẽ cần viết một dòng code riêng cho mỗi gói bạn muốn tải.

Tip. Nếu chỉ muốn sử dụng một hàm từ một gói, có thể sử dụng `pkgname::function`. Thay `pkgname` bằng tên gói đang sử dụng và `function` bằng lệnh gọi hàm như thường dùng.

Ví dụ,

```
neuralnet::gwplot(x)
```

Trong trường hợp này, không cần phải tải cả một packages vào phiên làm việc, mà chỉ cần dùng đúng lệnh `gwplot(x)` của gói đó thôi.

2 Các kiểu dữ liệu (data type) cơ bản trong R

Trong phần này, hàm `class()` sẽ được sử dụng để minh họa các kiểu dữ liệu khác nhau. Hàm này sẽ lấy một đối tượng hoặc giá trị và in ra kiểu dữ liệu của đối tượng hoặc giá trị cụ thể đó.

2.1 Numeric - Kiểu số

Kiểu numeric chính là số. Kiểu numeric bao phủ toàn bộ số thực. Nếu muốn chỉ rõ một giá trị là số nguyên (\mathbb{Z}), hãy thêm chữ cái L in hoa vào cuối số:

```
a <- 1
class(a)

## [1] "numeric"
```

```
b <- 1L
class(b)

## [1] "integer"
```

2.2 String - Kiểu chuỗi

Một biến string là một ký tự hoặc tên. Về cơ bản, biến string là bất kỳ từ (hoặc câu) nào. Khi tạo hoặc sử dụng string, chúng phải được đặt trong dấu ngoặc đơn hoặc dấu ngoặc kép:


```
fruit <- "Apple"
fruit
## [1] "Apple"
```

```
colour <- 'red'
colour
## [1] "red"
```

```
country <- Scotland
## Error: object 'Scotland' not found
```

Lưu ý rằng dòng lệnh cuối cùng tạo ra lỗi. Nếu không có dấu ngoặc kép, R sẽ nhận dạng đây là một đối tượng trong không gian làm việc, vì vậy trong trường hợp này, nó đang tìm kiếm một đối tượng có tên Scotland trong không gian làm việc. Chúng ta chưa tạo một đối tượng có tên Scotland, vì vậy nó sẽ báo lỗi.

Điều đáng chú ý là các giá trị chuỗi (giống như tên biến) phân biệt chữ hoa chữ thường trong R. Điều này có nghĩa là "red" và "Red" không được coi là giống nhau.

2.3 Logical

Một giá trị logical chỉ có thể nhận giá trị hoặc đúng hoặc sai. Các giá trị đúng và sai được biểu diễn bằng TRUE hoặc FALSE. Lưu ý rằng các giá trị này phải được viết hoa hoàn toàn để được nhận dạng chính xác. Sau đó, có thể gán giá trị TRUE hoặc FALSE cho một đối tượng bằng cách sử dụng vector gán <-.

```
# tạo một đối tượng có tên là x có giá trị TRUE
x <- TRUE
# In nội dung của x ra console
x
## [1] TRUE
```

```
class(x)
## [1] "logical"
```

```
y <- FALSE
y
## [1] FALSE
```

```
class(y)
## [1] "logical"
```

Giá trị logic là kết quả của một điều kiện. Các điều kiện phổ biến nhất trong R như sau:

- Bằng - được biểu thị bởi ==
- Không bằng - được biểu thị bởi !=
- Lớn hơn - được biểu thị bởi >
- Lớn hơn hoặc bằng - được biểu thị bởi >=

- Nhỏ hơn - được biểu thị bởi >
- Nhỏ hơn hoặc bằng - được biểu thị bởi >=

Chúng được sử dụng giữa hai đối tượng hoặc giá trị để thực hiện các kiểm tra điều kiện (ví dụ: `x == y` sẽ kiểm tra xem hai đối tượng trong R `x` và `y` có bằng nhau (`TRUE`) hay không (`FALSE`)). Hãy xem một số ví dụ về điều này.

```
x <- 2
y <- 3
z <- 2
x == y
## [1] FALSE
```

```
x > y
## [1] FALSE
```

```
x < z
## [1] FALSE
```

```
x <= z
## [1] TRUE
```

Kết quả của những so sánh này sau đó có thể được gán cho một đối tượng giống như bất kỳ giá trị nào khác:

```
comparison <- x!=y
comparison
## [1] TRUE
```

Các so sánh điều kiện cũng có thể được sử dụng để so sánh các chuỗi như có thể thấy trong ví dụ sau, điều này cũng chứng minh rằng các chuỗi trong R phân biệt chữ hoa chữ thường.

```
"red" == "Red"
## [1] FALSE
```

2.4 Kiểm tra kiểu dữ liệu

Hàm `class()` được sử dụng để hiển thị kiểu dữ liệu, nhưng đôi khi người dùng muốn kiểm tra cụ thể về một kiểu dữ liệu. Có những hàm cho phép làm điều này. Tất cả các hàm này sẽ trả về `TRUE` nếu đối tượng thuộc kiểu dữ liệu đang được kiểm tra, và `FALSE` nếu không.

Hàm `is.logical()` sẽ kiểm tra xem một đối tượng có phải là giá trị logic (đúng/sai) hay không.

```
is.logical(x)
## [1] TRUE
```

Đối tượng `x` có chứa giá trị logic, do đó trả về `TRUE`.

```
is.logical(a)
## [1] FALSE
```

Đối tượng `a` thì không, do đó giá trị `FALSE` được trả về.

Hàm `is.numeric()` sẽ kiểm tra xem một đối tượng có phải là giá trị số hay không.

```
is.numeric(fruit)
## [1] FALSE
```

`fruit` chứa một chuỗi, thay vì một số nên trả về `FALSE`.

```
is.numeric(a)
## [1] TRUE
```

`a` chứa một số, vì vậy điều này trả về `TRUE`.

Việc kiểm tra các giá trị số có thể được thực hiện sâu thêm một bước nữa bằng cách sử dụng `is.integer()` để kiểm tra xem một đối tượng có phải là số nguyên hay không.

```
is.integer(a)
## [1] FALSE
```

`a` không phải là số nguyên (vì chúng ta đã chỉ định `1` thay vì `1L` khi tạo `a`) nên trả về `FALSE`.

```
is.integer(b)
## [1] TRUE
```

`b` là một số nguyên (vì chúng ta đã chỉ định `1L` khi tạo nó) nên `TRUE` được trả về.

Hàm để kiểm tra xem một giá trị có phải là chuỗi hay không là `is.character()`.

```
is.character(colour)
## [1] TRUE
```

Câu lệnh này trả về `TRUE` vì `colour` là một chuỗi. Đoạn code bên dưới trả về `FALSE` vì `x` là một giá trị logic.

```
is.character(x)
## [1] FALSE
```

2.5 Thay đổi kiểu dữ liệu

Có thể thay đổi kiểu biến từ kiểu này sang kiểu khác, điều này được gọi là ép kiểu dữ liệu hoặc chuyển đổi kiểu dữ liệu. Các hàm tương tự như đã mô tả trong phần trước. Để chuyển đổi thành một đối tượng logic, sử dụng `as.logical()`, để chuyển đổi thành một biến số, hãy sử dụng `as.numeric()` hoặc `as.integer()`, để chuyển đổi thành một chuỗi, hãy sử dụng `as.character()`.

Tip. Để chú thích code giúp việc xem lại dễ dàng hơn, có thể thêm chú thích vào mã bằng ký hiệu `#`. Điều này sẽ báo cho R biết rằng bạn không muốn chạy bất kỳ đoạn code nào theo sau nó.

Các ví dụ về ép kiểu dữ liệu đối với từng kiểu dữ liệu đã thảo luận được hiển thị trong các đoạn code bên dưới.

```
# numeric sẽ đổi thành logical
# Số 0 sẽ đổi thành giá trị FALSE
as.logical(0)
```

```
## [1] FALSE
```

```
# Mọi số khác sẽ đổi thành giá trị TRUE  
as.logical(2)
```

```
## [1] TRUE
```

```
as.logical(-1)
```

```
## [1] TRUE
```

```
as.logical(0.2)
```

```
## [1] TRUE
```

```
## Strings sẽ không đổi thành logical
```

```
as.logical("ten")
```

```
## [1] NA
```

Giá trị số 0 sẽ được chuyển thành FALSE, trong khi bất kỳ giá trị nào khác không sẽ trở thành TRUE. String không bị ép thành giá trị logic. Thực hiện điều này sẽ trả về NA, đây là cách R biểu thị các giá trị dữ liệu missing.

```
## logical sang numeric
```

```
as.numeric(TRUE)
```

```
## [1] 1
```

```
as.numeric(FALSE)
```

```
## [1] 0
```

```
## string sang numeric
```

```
# một từ sẽ không đổi thành numeric được
```

```
as.numeric("ten")
```

```
## Warning: NAs introduced by coercion
```

```
## [1] NA
```

```
# nhưng một số sẽ đổi thành một string được
```

```
as.numeric("4.5")
```

```
## [1] 4.5
```

```
as.integer("4.5")
```

```
## [1] 4
```

Khi ép các giá trị logic thành numeric, TRUE sẽ luôn trở thành 1 và FALSE sẽ luôn trở thành 0. Khi ép string thành numeric, một từ thông thường sẽ không đổi được và R sẽ đưa ra warning để thông báo về điều này, tuy nhiên một số được bao quanh bởi dấu ngoặc kép sẽ được đổi đúng thành một numeric.

Bất kỳ giá trị nào cũng có thể được ép thành một chuỗi. Điểm khác biệt duy nhất khi được hiển thị là các giá trị sẽ được hiển thị với dấu ngoặc kép bao quanh, thay vì không có.

```
## logical sang character
as.character(TRUE)
## [1] "TRUE"
```

```
as.character(FALSE)
## [1] "FALSE"
```

```
## Numeric sang character
as.character(10)
## [1] "10"
```

```
as.character(-2.263)
## [1] "-2.263"
```

Errors vs. Warnings (Lỗi vs. Cảnh báo)

R sẽ đánh dấu các vấn đề theo hai cách: lỗi - errors và cảnh báo - warnings.

- Warnings được sử dụng để nhấn mạnh rằng một thao tác có thể không được thực hiện theo cách mong đợi, nhưng code vẫn sẽ tiếp tục chạy. Nếu nhận được warnings, nên kiểm tra xem bất kỳ thao tác nào được thực hiện thông qua code đã tạo ra kết quả như mong đợi hay chưa. Thông báo warnings sẽ luôn bắt đầu bằng "Warning".
- Ngược lại, error có nghĩa là R đã được yêu cầu thực hiện một việc mà nó không thể thực hiện. Nó sẽ dừng chạy code cho đến khi lỗi trong mã được sửa. Thông báo lỗi sẽ luôn bắt đầu bằng "Error".

Exercise 2.1. Dùng R, tạo các đối tượng sau:

- một biến x là số và nhận giá trị 16.89
- một biến y là số nguyên và nhận giá trị 10
- một biến z là logic và FALSE
- một biến name là chuỗi chứa tên của bạn
- biến giá trị logic, z , thành giá trị số
- biến giá trị số, x , thành giá trị chuỗi

3 Cấu trúc dữ liệu (data structure) trong R

Cấu trúc dữ liệu là cách R tổ chức và lưu trữ các giá trị (có thể gồm nhiều kiểu hoặc cùng kiểu dữ liệu).

3.1 Vectors

Các vector được sử dụng để lưu trữ nhiều quan sát có cùng thuộc tính. Các vector được tạo bằng cách gói các phần tử của vector trong `c()`, với mỗi quan sát được phân tách bằng dấu phẩy. Ví dụ: nếu muốn tạo một vector chứa các phần tử "Apple", "Orange", "Pear" và "Grapes" (và lưu trữ nó trong một đối tượng gọi là `fruits`):

```
fruits <- c("Apple", "Orange", "Pear", "Grapes")
```

Nếu bạn chạy đoạn code này, nó sẽ đưa một đối tượng vào môi trường, gọi là `fruits`. Đối tượng này thuộc kiểu "chr" (viết tắt của character). Giá trị [1:4] cho biết đây là một vector có 4 phần tử.

Một vector cũng có thể chứa các giá trị numeric hoặc giá trị logic. Hãy tạo các vector chiều cao (in) và tình trạng hút thuốc cho một mẫu gồm 10 cá nhân:

```
height <- c(1.68, 1.83, 1.57, 1.87, 1.7, 1.57, 1.73, 1.85, 1.6, 1.58)
smokes <- c(TRUE, TRUE, FALSE, TRUE, FALSE, FALSE, TRUE, FALSE, FALSE, FALSE)
```

Trích xuất giá trị

Bạn có thể trích xuất các giá trị riêng lẻ từ một vector bằng cách sử dụng "ký hiệu ngoặc vuông [...]". Sử dụng vector `fruits`, chúng ta có thể trích xuất giá trị thứ hai bằng cách chạy `fruits[2]` để lấy "Orange". Giá trị trong ngoặc vuông là vị trí của phần tử bạn muốn trích xuất. Đây cũng có thể là một vector, vì vậy việc nhập `fruits[c(2,4)]` sẽ trả về kết quả "Orange, Grapes".

Nếu muốn trích xuất cụ thể một vài giá trị đầu tiên trong dữ liệu, bạn có thể sử dụng `head()`:

```
head(height)
## [1] 1.68 1.83 1.57 1.87 1.70 1.57
```

Theo mặc định, 6 giá trị đầu tiên sẽ được trích xuất nhưng có thể kiểm soát điều này bằng cách sử dụng `n` như sau:

```
head(height, n = 4)
## [1] 1.68 1.83 1.57 1.87
```

Hàm `tail()` có thể được sử dụng theo cách tương tự để trích xuất các giá trị từ cuối một vector.

Một cách cuối cùng để trích xuất giá trị từ các vector là sử dụng các câu lệnh điều kiện. Khi các câu lệnh điều kiện liên quan đến ít nhất một vector, phép so sánh sẽ được thực hiện theo từng phần tử. Để thực hiện phép so sánh có chiều cao > 1.7 :

```
height > 1.7
## [1] FALSE TRUE FALSE TRUE FALSE FALSE TRUE TRUE FALSE FALSE
```

Nhận được một vector logic mô tả xem mỗi phần tử có chiều cao lớn hơn 1.7 hay không.

Điều này có nghĩa là, nếu muốn trích xuất tất cả các phần tử có chiều cao lớn hơn 1.7, có thể sử dụng như sau:

```
height[height > 1.7]
## [1] 1.83 1.87 1.73 1.85
```

The `height` before the square brackets says that I want to extract values from the vector `height`, and then within the square brackets says I want the elements where height is greater than 1.7. When reading this type of code I tend to read it as “extract the elements of height where height is greater than 1.7”.

The smokes vector is already a logical vector so I can use this directly:

height trước dấu ngoặc vuông cho biết muốn trích xuất các giá trị từ chiều cao của vector, và sau đó trong dấu ngoặc vuông cho biết muốn các phần tử có chiều cao lớn hơn 1,7. Khi đọc loại code này, thường đọc nó là "trích xuất các phần tử có chiều cao lớn hơn 1,7".

Vì smokes là một vector logic nên có thể sử dụng trực tiếp:

```
smokes[smokes]
## [1] TRUE TRUE TRUE TRUE
```

Về cơ bản đoạn code trên nói rằng muốn trích xuất tất cả các phần tử của smokes mà smokes là TRUE. Để lấy tất cả các quan sát mà smokes là FALSE, có thể sử dụng dấu chấm than (!) để biểu thị "không":

```
smokes[!smokes]
## [1] FALSE FALSE FALSE FALSE FALSE FALSE
```

Ép kiểu giá trị

Có thể ép các vector như mô tả đối với các giá trị đơn lẻ:

```
as.character(height)
## [1] "1.68" "1.83" "1.57" "1.87" "1.7" "1.57" "1.73" "1.85" "1.6"
## [10] "1.58"
```

```
as.logical(height)
## [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

```
as.numeric(smokes)
## [1] 1 1 0 1 0 0 1 0 0 0
```

Lưu ý rằng nếu bạn cố gắng lưu trữ nhiều kiểu biến trong một vector, các giá trị sẽ bị ép về một kiểu biến chung (thường là character). Hãy chú ý cách các giá trị numeric và logic trở thành character (được bao quanh bởi dấu ngoặc kép) trong ví dụ dưới đây vì "banana" được bao gồm trong vector.

```
c(10, "banana", 1, 3, 5, TRUE)
## [1] "10" "banana" "1" "3" "5" "TRUE"
```

Có những cách hiệu quả khác để tạo vector numeric. Nếu bạn muốn tạo một chuỗi giá trị tăng dần theo từng bước 1, có thể chỉ định như sau:

```
1:10
## [1] 1 2 3 4 5 6 7 8 9 10
```

Giá trị bên trái là giá trị bắt đầu và giá trị bên phải là giá trị kết thúc. Giá trị này có thể được đọc là từ 1 đến 10. Có thể chỉ định các giá trị không phải số nguyên theo định dạng này, mặc dù nó vẫn sẽ tăng dần theo từng bước một:

```
3.4:12.5
## [1] 3.4 4.4 5.4 6.4 7.4 8.4 9.4 10.4 11.4 12.4
```

Factor - Nhân tố

Vector factor là một loại vector cụ thể thường được sử dụng trong R để lưu trữ thông tin phân loại. Chúng được lưu trữ dưới dạng một vector số nguyên, và mỗi giá trị số nguyên có một nhãn tương ứng. Để tạo một factor, nên sử dụng hàm `factor` và truyền vào một vector ký tự. Để minh họa điều này, hãy xem xét giới tính của một mẫu gồm mười cá nhân.

```
gender <- factor(c("Male", "Male", "Female", "Male", "Female",  
                  "Female", "Female", "Male", "Female", "Female"))
```

Trong môi trường, bạn sẽ thấy một đối tượng có tên là `gender` có mô tả sau:

```
## Factor w/ 2 levels "Female", "Male": 2 2 1 2 1 1 1 2 1 1
```

Mức độ đề cập đến các nhãn được liên kết với mỗi số nguyên. Theo mặc định, các nhãn này được sắp xếp theo thứ tự bảng chữ cái, vì vậy `Female` tương ứng với 1 và `Male` tương ứng với 2. Điều này phù hợp với ví dụ này, nhưng nếu muốn biết một cá nhân có mức độ hoạt động trung bình hàng ngày thấp, trung bình hay cao thì sao?

```
activity <- factor(c("Low", "High", "Low", "Moderate", "High", "High",  
                   "Moderate", "Moderate", "Low", "Low"))  
# Hàm levels() in các nhãn ra theo thứ tự  
levels(activity)  
## [1] "High"      "Low"       "Moderate"
```

The order doesn't really make sense with the meaning of the labels. We would much prefer the levels to follow the order Low, Moderate and then High. We can do this by specifying the levels argument within the `factor()` function as follows.

Thứ tự này không thực sự hợp lý với ý nghĩa của các nhãn. Các mức độ nên được sắp xếp theo thứ tự Low, Moderate rồi đến High. Có thể thực hiện điều này bằng cách chỉ định đối số `levels` trong hàm `factor()` như sau.

```
activity <- factor(c("Low", "High", "Low", "Moderate", "High", "High",  
                   "Moderate", "Moderate", "Low", "Low"),  
                 levels = c("Low", "Moderate", "High"))  
  
# Hàm levels() in các nhãn ra theo thứ tự  
levels(activity)  
## [1] "Low"      "Moderate" "High"
```

Bây giờ chúng đã được sắp xếp theo đúng thứ tự.

Hàm `as.factor()` có thể được sử dụng tương tự như `as.numeric()`, v.v. để ép các giá trị thành các nhân tố. Tương tự, `as.numeric()` và `as.character()` có thể được sử dụng để trích xuất các giá trị số và chuỗi liên quan đến các quan sát trong một vector nhân tố.

```
# Trích xuất các nhãn của từng quan sát  
as.character(activity)  
## [1] "Low"      "High"      "Low"      "Moderate" "High"      "High"  
## [7] "Moderate" "Moderate" "Low"      "Low"
```



```
# Trích xuất giá trị số
as.numeric(activity)
## [1] 1 3 1 2 3 3 2 2 1 1
```

Exercise 3.1. Sử dụng R, hãy tạo các đối tượng sau:

- Một vectơ số gọi là *z* chứa các phần tử 2.3, 4.8, 10.3, 12.5, 3.5, 6.7.
- Một vectơ ký tự chứa ba màu: Blue, Red, Green.
- Một vectơ nhân tố với các quan sát “Scotland”, “England”, “Wales”, “Wales”, “Scotland”, “Scotland”, “England”, “Wales”, “Scotland”.
- Một vectơ nhân tố với các phần tử “Small”, “Medium”, “Medium”, “Small”, “Big”, “Big” trong đó các mức được sắp xếp từ “Small” đến “Big”.
- Trích xuất phần tử thứ 2 từ vectơ *z*.
- Trích xuất phần tử thứ 1, 3 và 5 từ vectơ *z* bằng một dòng code.

Một số hàm dùng cho vectors

- `length()`: trả về chiều dài vector.

```
length(c(1, 2, 3, 4, 5, 6))
## [1] 6
```

- `sort()`: sắp xếp thứ tự các phần tử của vector, theo thứ tự tăng hay giảm.

```
x <- c(1, 0, 3, 2, 4)
sort(x)
## [1] 0 1 2 3 4
```

```
sort(x, decreasing=TRUE)
## [1] 4 3 2 1 0
```

- `order()`: trả về vị trí các phần tử của vector khi đã sắp xếp theo thứ tự tăng

```
order(x)
## [1] 2 1 4 3 5
```

- `rev()`: đảo thứ tự các phần tử của vector.

```
rev(x)
## [1] 4 2 3 0 1
```

- `sum()`: trả về tổng giá trị các phần tử của vector.

```
sum(x)
## [1] 10
```

Exercise 3.2. Sử dụng các hàm trên cho các vector được tạo trong bài tập 3.1.

3.2 Ma trận

Tạo ma trận

Các đối tượng ma trận chứa các giá trị được lưu trữ theo hàng và cột trong mảng hình chữ nhật và được tạo bởi hàm `matrix()`. Ma trận thường được sử dụng để lưu trữ thông tin số, nhưng cũng có thể lưu trữ các kiểu biến khác. Các đối số sau cần được truyền vào `matrix()`:

- một vector giá trị của các phần tử ma trận,
- số hàng hoặc cột.
- có thể muốn thiết lập thứ tự đọc theo hàng hoặc theo cột.

Một ví dụ về việc tạo ma trận:

```
data <- c(190, 8, 22, 191, 4, 1.7, 223, 80, 2)
barley.data <- matrix(data, nrow = 3)
barley.data
##      [,1] [,2] [,3]
## [1,] 190 191.0 223
## [2,]   8  4.0  80
## [3,]  22  1.7   2
```

Chỉ định số hàng bằng cách sử dụng `nrow` như ở trên, nhưng có thể chỉ định số cột bằng cách sử dụng `ncol` theo cách tương tự:

```
data <- c(190, 8, 22, 191, 4, 1.7, 223, 80, 2)
barley.data <- matrix(data, ncol = 3)
barley.data
##      [,1] [,2] [,3]
## [1,] 190 191.0 223
## [2,]   8  4.0  80
## [3,]  22  1.7   2
```

Lưu ý rằng các giá trị từ vector được đặt vào ma trận theo từng cột. Nếu muốn các giá trị được đặt vào ma trận theo từng hàng, bạn có thể chỉ định `byrow = TRUE` như sau:

```
data <- c(190, 8, 22, 191, 4, 1.7, 223, 80, 2)
barley.data <- matrix(data, ncol = 3, byrow = TRUE)
barley.data
##      [,1] [,2] [,3]
## [1,] 190   8 22.0
## [2,] 191   4  1.7
## [3,] 223  80  2.0
```

Giống như vector, tất cả các phần tử ma trận phải cùng kiểu, tức là numeric, character, v.v. Một ví dụ về ma trận logic là:

```
matrix(c(FALSE, TRUE, FALSE, TRUE), nrow = 2)

##           [,1]  [,2]
## [1,] FALSE FALSE
## [2,]  TRUE  TRUE
```

Nếu bạn cố gắng tạo một ma trận có kiểu hỗn hợp, kết quả sẽ bị ép đổi kiểu dữ liệu. Ví dụ:

```
matrix(c(1, TRUE, "a", 4), nrow = 2)

##           [,1]  [,2]
## [1,] "1"      "a"
## [2,] "TRUE"   "4"
```

Ở đây, dữ liệu chứa các số nguyên, ký tự và kiểu logic, nhưng đầu ra đã bị ép thành ký tự vì đây là kiểu duy nhất có thể chứa tất cả các kiểu khác.

Trích xuất giá trị từ ma trận

Việc trích xuất dữ liệu từ ma trận và vector cũng tương tự nhau. Sử dụng dấu ngoặc vuông để xác định hàng và cột của ma trận mà muốn trích xuất. Trong dấu ngoặc, số đầu tiên chỉ các hàng và số thứ hai chỉ các cột, ví dụ: [1,2] sẽ là hàng đầu tiên và cột thứ hai.

```
barley.data[1,1]

## [1] 190
```

```
barley.data[1,2]

## [1] 8
```

Có thể trích xuất toàn bộ hàng hoặc cột. Ví dụ, hãy xem xét việc trích xuất tất cả các cột trừ hàng đầu tiên.

```
barley.data[1,]

## [1] 190 8 22
```

Trích xuất tất cả các hàng của cột thứ hai.

```
barley.data[,2]

## [1] 8 4 80
```

Sử dụng dấu âm (-) để biểu thị rằng chúng ta muốn chọn tất cả các hàng hoặc cột ngoại trừ một hàng. Ví dụ, giả sử chúng ta muốn chọn tất cả các hàng ngoại trừ hàng đầu tiên, chúng ta có thể code

```
barley.data[-1,]

##           [,1] [,2] [,3]
## [1,] 191      4  1.7
## [2,] 223     80  2.0
```

Thêm tên cho hàng và cột

Tên có thể được gán cho các hàng và cột bằng cách sử dụng đối số `dimnames()` trong hàm `matrix()`. Ví dụ, giả sử chúng ta có một số vị trí và loại lúa mạch:

```
data <- c(190, 8, 22, 191, 4, 1.7, 223, 80, 2)
barley.data = matrix(
  data,
  nrow=3,
  dimnames = list(
    c("Navarra", "Zaragoza", "Madrid"),
    c("TypeA", "TypeB", "TypeC")
  )
)
barley.data
```

	TypeA	TypeB	TypeC
Navarra	190	191.0	223
Zaragoza	8	4.0	80
Madrid	22	1.7	2

Đầu tiên, chúng ta chỉ định tên hàng và sau đó là tên cột. Việc này có thể được thực hiện tuần tự bằng cách sử dụng hàm `dimnames()`.

```
data <- c(190, 8, 22, 191, 4, 1.7, 223, 80, 2)
barley.data <- matrix(data, nrow=3)
area <- c("Navarra", "Zaragoza", "Madrid")
type <- c("TypeA", "TypeB", "TypeC")
dimnames(barley.data) <- list(area, type)
barley.data
```

	TypeA	TypeB	TypeC
Navarra	190	191.0	223
Zaragoza	8	4.0	80
Madrid	22	1.7	2

Cả hai trường hợp đều tạo ra cùng một kết quả.

Các phép toán trên ma trận

Các toán tử `+`, `-`, `*` và `/` có thể được áp dụng cho các ma trận có cùng kích thước. Ví dụ:

```
x <- matrix(1:4, nrow=2)
y <- x
x + y
```

	[,1]	[,2]
[1,]	2	6
[2,]	4	8

Như bạn đã thấy với vector, các toán tử được vector hóa.

Để tính tích vô hướng của hai ma trận, chúng ta sử dụng `%*%`.

```
x %*% y
```

	[,1]	[,2]
[1,]	7	15
[2,]	10	22

Chúng ta cũng có thể lấy được phép chuyển vị của ma trận bằng cách sử dụng hàm `t()`:

```
x
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

```
t(x)
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
```

Exercise 3.3. Dùng R, thực hiện các công việc sau:

- Tạo ma trận $x = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$.
- Tạo ma trận $y = \begin{bmatrix} 5 & 7 \\ 6 & 8 \end{bmatrix}$.
- Tạo ma trận $p = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$.
- Tính $x + y$.
- Cho $g = \begin{bmatrix} 6 & 8 \\ 7 & 9 \end{bmatrix}$ và $h = \begin{bmatrix} 2 & 4 \\ 3 & 5 \end{bmatrix}$, tìm $g \times h$.
- Tìm ma trận chuyển vị của ma trận $z = \begin{bmatrix} 1 & 3 & 5 & 7 \\ 2 & 4 & 6 & 8 \end{bmatrix}$.

Exercise 3.4. Dùng R, thực hiện:

- Tạo ma trận $m = \begin{bmatrix} 3.8 & 4.7 \\ 3.9 & 7.0 \end{bmatrix}$.
- In ma trận m .
- Trích xuất phần tử hàng thứ nhất và cột thứ hai.
- Thay đổi phần tử của hàng thứ nhất và cột thứ hai của ma trận m thành 4.5 mà không cần phải tạo lại m .

Exercise 3.5. Code cho một ma trận, được gọi là `grades`, chứa phần trăm điểm của 5 sinh viên (giả định) tham gia 5 kỳ thi trong học kỳ 2, cùng với họ và môn học dưới dạng nhãn hàng và cột tương ứng, được đưa ra bên dưới:

```
matrix(c(67, 72, 40, 65, 56,
         70, 81, 32, 69, 48,
         45, 85, 25, 57, 63,
         34, 90, 42, 55, 65,
         62, 79, 36, 61, 60),
       byrow=FALSE, nrow=5,
       dimnames = list(
         c('Bisset', 'Finlayson', 'Milne', 'Stewart', 'Jones'),
         c('Calculus', 'Algebra', 'Statistics', 'Finance', 'Computing'))
```

```
)  
)
```

- Tính điểm trung bình của Milne.
- Tính điểm trung bình môn Thống kê (Statistics).

3.3 Data Frames

R là một ngôn ngữ lập trình thống kê. Do đó, R có khả năng lưu trữ dữ liệu một cách trực quan. Vector là lựa chọn tốt để lưu trữ một biến, nhưng nếu muốn sử dụng nhiều biến thuộc nhiều kiểu dữ liệu khác nhau, chúng ta cần một thứ gì đó linh hoạt hơn. Đây chính là lúc một data frame phát huy tác dụng.

Data frame trong R mô phỏng cấu trúc của một bảng tính (spreadsheet). Chúng có nhiều cột, mỗi cột là một biến cụ thể và nhiều hàng, mỗi hàng là một quan sát cụ thể. Tạo một data frame trong R bằng hàm `data.frame()` và chỉ định mỗi biến là một vector quan sát. Ví dụ, trước tiên hãy tạo một số vector rồi kết hợp chúng thành một data frame.

```
gender <- c("Male", "Male", "Female", "Male", "Female", "Female",  
           "Female", "Male", "Female", "Female")  
activity <- c("Low", "High", "Low", "Moderate", "High", "High",  
            "Moderate", "Moderate", "Low", "Low")  
height <- c(1.68, 1.83, 1.57, 1.87, 1.70, 1.57,  
           1.73, 1.85, 1.60, 1.58)  
smokes <- c(TRUE, TRUE, FALSE, TRUE, FALSE,  
            FALSE, TRUE, FALSE, FALSE, FALSE)  
activity.levels <- data.frame(  
  gender = gender,  
  Activity = activity,  
  Height = height,  
  Smokes = smokes  
)  
activity.levels  
  
##      gender Activity Height Smokes  
## 1     Male      Low   1.68    TRUE  
## 2     Male      High   1.83    TRUE  
## 3  Female      Low   1.57   FALSE  
## 4     Male Moderate   1.87    TRUE  
## 5  Female      High   1.70   FALSE  
## 6  Female      High   1.57   FALSE  
## 7  Female Moderate   1.73    TRUE  
## 8     Male Moderate   1.85   FALSE  
## 9  Female      Low   1.60   FALSE  
## 10 Female      Low   1.58   FALSE
```

Các cột được tạo sao cho `name = value`, từ bên trái mỗi dấu bằng là tên cột, còn bên phải mỗi dấu bằng là một vector quan sát.

Nhìn chung, thường sẽ không tạo dữ liệu trong R (mặc dù đây là một kỹ năng rất hữu ích!), mà có thể sẽ đọc dữ liệu từ một tệp.

Để đọc tệp csv có tên là `file.csv`:

```
my_data <- read.csv("file.csv")
```

Có một số tham số bổ sung mà nhìn chung (mặc dù không phải lúc nào cũng) hữu ích khi chỉ định. Việc sử dụng những tham số này sẽ tùy thuộc vào dữ liệu đang đọc.

- `header = TRUE` cho R biết rằng tập dữ liệu chứa tên cột.
- `na.strings` - đây là giá trị trong dữ liệu đại diện cho một giá trị bị thiếu. Giá trị này có thể là một khoảng trống, một dấu sao (*), hoặc thậm chí là một giá trị như 999.

Ví dụ nếu tập dữ liệu (hiển thị bên dưới) được lưu trữ trong tệp csv.

Age	Gender
23	M
66	*
43	F
58	M
*	M
31	F

Nhìn vào đây, bạn có thể thấy rằng các tiêu đề cột trong dữ liệu và các giá trị bị thiếu được biểu thị bằng dấu *, thì code để đọc trong dữ liệu sẽ là:

```
my_data <- read.csv("file.csv", header = TRUE, na.strings = "*")
```

Tip. Một lỗi phổ biến khác là không lưu trữ dữ liệu trong một đối tượng khi đọc dữ liệu. Khi sử dụng bất kỳ hàm nào có sẵn để đọc dữ liệu, hãy đảm bảo sử dụng toán tử gán `<-` để đặt tên cho dữ liệu trong R để có thể sử dụng sau này!

Code như giống hệt nhau nếu dữ liệu được lưu trữ trong tệp văn bản với khoảng trắng phân tách các giá trị ở các cột khác nhau. Hàm được sử dụng lần này là `read.table()`.

```
my_data <- read.table("file.txt", header = TRUE, na.strings = "*")
```

Trích xuất giá trị

Giá trị có thể được trích xuất từ một data frame theo nhiều cách khác nhau. Cách đầu tiên là sử dụng ký hiệu ngoặc vuông theo cách hoàn toàn giống như đối với ma trận, trong đó nhập số cột và/hoặc số hàng cho các quan sát muốn trích xuất. Điều này hữu ích trong một số trường hợp nhất định, nhưng nếu quan tâm đến một biến cụ thể, việc phải đếm dọc theo tập dữ liệu để tìm số cột quan tâm có thể khá phiền phức. Thay vào đó, có thể tham chiếu đến các cột theo tên của chúng bằng cách sử dụng "\$", trong đó chỉ định data frame và tên cột của mình như sau: `data_name$column_name`. Có thể lấy danh sách tên cột cho data frame của mình bằng hàm `names()`. Vì vậy, việc nhập `names(activity.levels)` sẽ xuất ra tên cột của data frame `activity.levels`.

Ví dụ nếu muốn trích xuất cột về chiều cao từ tập dữ liệu `activity.levels`:

```
names(activity.levels)
## [1] "gender" "Activity" "Height" "Smokes"
```

Cột chứa dữ liệu về chiều cao được gọi là `Height`.

```
activity.levels$Height
## [1] 1.68 1.83 1.57 1.87 1.70 1.57 1.73 1.85 1.60 1.58
```

Điều này cho chúng ta một vector về chiều cao của tất cả những người tham gia nghiên cứu. Chúng ta có thể tiến thêm một bước nữa và trích xuất một quan sát duy nhất từ cột này bằng cách sử dụng ký hiệu ngoặc vuông. Vì vậy, nếu tôi muốn trích xuất quan sát chiều cao thứ 8:

```
activity.levels$Height[8]
## [1] 1.85
```

Lưu ý rằng dấu phẩy không cần phải được đưa vào trong dấu ngoặc vuông ở đây vì \$ đã trả về một vector.

Các câu lệnh điều kiện có thể được sử dụng để phân tập con data frame theo cách tương tự như cho vector. Ví dụ, chúng ta có thể muốn trích xuất tất cả các cá nhân từ tập dữ liệu mức độ hoạt động của mình có chiều cao lớn hơn hoặc bằng 1.8. Việc này sẽ được thực hiện như sau:

```
activity.levels[activity.levels$Height > 1.8, ]
##   gender Activity Height Smokes
## 2   Male      High   1.83   TRUE
## 4   Male Moderate   1.87   TRUE
## 8   Male Moderate   1.85  FALSE
```

Lưu ý rằng câu lệnh điều kiện nằm ở vị trí đầu tiên, trước dấu phẩy, trong ngoặc vuông để chỉ ra rằng chúng ta muốn trích xuất các hàng. Cũng lưu ý rằng ký hiệu \$ đã được sử dụng để chỉ định cột Height trong data frame activity.levels. Phải cho R biết rằng đang tham chiếu đến cột trong data frame. Vì vậy, tôi sẽ đọc dòng code này (theo cách hiểu thông thường) là "trích xuất tất cả các hàng từ activity.levels có giá trị trong cột Height trong activity.levels lớn hơn hoặc bằng 1.8".

Một điều cần lưu ý, nếu bạn đang đọc về lập trình R và thấy từ tibble được đề cập, thì chúng giống như data frame, chỉ là chúng được định dạng cụ thể để hiển thị trên màn hình nhỏ theo cách khác với data frame - thao tác và sử dụng tibble được thực hiện theo cách hoàn toàn giống như thao tác và sử dụng data frame.

3.4 List

List hoạt động khá giống với data frame (data frame là một loại danh sách đặc biệt) nhưng linh hoạt hơn một chút. Trong data frame, mỗi biến phải có cùng độ dài. Điều này không đúng với danh sách. Các cấu trúc dữ liệu phức tạp (ví dụ như dữ liệu được tạo bởi trang web) thường được lưu trữ trong R dưới dạng list.

Hàm tạo list là list(). Các phần tử khác nhau của list được phân tách bằng dấu phẩy, tương tự như cách phân tách các cột trong một data frame đầu vào. Hãy bắt đầu đơn giản bằng cách tạo một list gồm các phần tử có giá trị đơn:

```
simple_list <- list(number = 10.36, string = "Apple",
                    logical = TRUE, integer = 6L)
simple_list
## $number
## [1] 10.36
```



```
##
## $string
## [1] "Apple"
##
## $logical
## [1] TRUE
##
## $integer
## [1] 6
```

Trong kết quả đầu ra ở trên, các phần tử được biểu diễn dưới dạng \$ theo sau là tên phần tử - điều này có nghĩa là bạn có thể sử dụng ký hiệu \$ để trích xuất các phần tử. Để trích xuất giá trị chuỗi từ `simple_list`, có thể nhập:

```
simple_list$string
## [1] "Apple"
```

Tuy nhiên, đôi khi có thể gặp phải một list không có phần tử nào được đặt tên (hoặc có thể việc tham chiếu đến index sẽ thuận tiện hơn là tên phần tử). Để tạo một list không có tên:

```
unnamed_list <- list(pi, "Orange", FALSE, 9L)
unnamed_list

## [[1]]
## [1] 3.141593
##
## [[2]]
## [1] "Orange"
##
## [[3]]
## [1] FALSE
##
## [[4]]
## [1] 9
```

Lưu ý rằng khi in ra, mỗi phần tử đều có một số được đặt trong dấu ngoặc vuông kép `[[...]]`. Điều này cho biết nên tham chiếu đến từng phần tử theo cách này. Để trích xuất phần tử thứ hai:

```
unnamed_list[[2]]
## [1] "Orange"
```

Trích xuất các phần tử bằng cách sử dụng dấu ngoặc vuông đơn, mặc dù điều này sẽ ảnh hưởng đến kiểu biến được xuất ra. Lưu ý bên dưới: dấu ngoặc vuông kép trả về một ký tự, trong khi dấu ngoặc vuông đơn trả về một list. Điều này không quá quan trọng, nhưng rất hữu ích khi bạn cần thực hiện các thao tác dữ liệu cụ thể!

```
class(unnamed_list[[2]])
## [1] "character"
```

```
class(unnamed_list[2])
## [1] "list"
```

Ưu điểm của list là tính linh hoạt. Mỗi phần tử trong list có thể khác nhau, cả về kiểu biến lẫn kích thước biến. Điều này được minh họa bên dưới:

```
list(
  vector = c(10, 36.7, 96, 5, 25.3, 67.8),
  data_frame = data.frame(
    Name = c("John", "Paul", "Mick", "Ruth", "Sandra"),
    Age = c(28, 32, 58, 72, 54)
  ),
  list = simple_list)
## $vector
## [1] 10.0 36.7 96.0 5.0 25.3 67.8
##
## $data_frame
##      Name Age
## 1   John  28
## 2   Paul  32
## 3   Mick  58
## 4   Ruth  72
## 5 Sandra  54
##
## $list
## $list$number
## [1] 10.36
##
## $list$string
## [1] "Apple"
##
## $list$logical
## [1] TRUE
##
## $list$integer
## [1] 6
```

Đầu ra cho các mục số và data frame là những gì thường thấy. Đối với mục nhập list, bạn có thể thấy R ghi là `$list` và sau đó là nhiều mục nhập có dạng, ví dụ: `$list$number`. Điều này là do chúng ta có một list trong một list khác, vì vậy để tham chiếu đến các phần tử trong list thứ hai đó, trước tiên bạn cần truy cập phần tử list (`$list`), sau đó truy cập phần tử cụ thể trong list bên trong đó bằng `$number`.

3.5 Missing data

Trong R, các giá trị missing (thiếu hoặc khuyết) được biểu thị bằng ký hiệu NA, nghĩa là không khả dụng. Các giá trị không thể tính (ví dụ: chia cho số 0) được biểu thị bằng ký hiệu NaN.

Kiểm tra Missing Values

Chúng ta có thể sử dụng hàm `is.na()` để cung cấp một vector logic ghi chú xem từng giá trị của vector có bị thiếu hay không.

```
y <- c(1, 2, 3, 4, NA)

is.na(y)

## [1] FALSE FALSE FALSE FALSE TRUE
```

Ở đây chúng ta có thể thấy rằng R trả về TRUE nếu quan sát bị thiếu.

Mã hóa Missing Values

Trong một số trường hợp, chúng ta có thể thấy một quan sát không chính xác và muốn thay đổi nó thành NA. Cũng có những trường hợp dữ liệu được tạo trong các phần mềm khác sử dụng một giá trị khác để chỉ missing values. Điều này có thể được thực hiện đơn giản trong R. Giả sử chúng ta có một biến cấp 99 bị thiếu, chúng ta có thể mã hóa lại thành NA như sau:

```
mydata$v1[mydata$v1 == 99] <- NA
```

Đầu tiên, lệnh này sẽ chọn các hàng có v1 bằng chính xác 99 (ý nghĩa của ==), sau đó mã hóa lại thành NA. Bạn có thể thay thế 99 bằng bất kỳ giá trị nào còn thiếu.

Loại bỏ missing values trong phân tích

Có thể cần loại bỏ các giá trị bị thiếu khỏi phép phân tích. Điều này được thể hiện trong các hàm tính toán như `mean()`. Sử dụng đối số `na.rm = TRUE` sẽ loại bỏ các giá trị bị thiếu để hoàn tất phép tính.

```
mean(y) # trả về NA

## [1] NA

mean(y, na.rm = TRUE)

## [1] 2.5
```

Hàm `complete.cases()` trả về một vector logic cho biết trường hợp nào đã hoàn tất. Code cho hàm này sẽ như sau:

```
mydata[!complete.cases(mydata), ]
```

Hàm `na.omit()` trả về đối tượng với danh sách xóa các giá trị bị thiếu.

```
newdata <- na.omit(mydata)
```

4 Các tính toán cơ bản trong R

Một trong những ứng dụng cơ bản nhất của R là thực hiện các phép tính - về cơ bản là sử dụng R như một máy tính. Các toán tử cơ bản là:

- Cộng: +
- Trừ: -

- Nhân: *
- Chia: /
- Lũy thừa: ^

Vì vậy, nhập $2 + 3$ vào bảng điều khiển sẽ cho kết quả là 5; $2 - 3$ sẽ cho kết quả là -1; $2 * 3$ sẽ cho kết quả là 6; và $2 / 3$ sẽ cho kết quả là 0.6666667 và $2 ^ 3$ sẽ cho kết quả là 8.

Xin lưu ý rằng mặc dù phần lớn các ví dụ đều cung cấp giá trị vào vị trí vào phép tính, nhưng bất kỳ đối tượng số nào được đặt tên (giá trị đơn hoặc vector) trong môi trường của bạn đều có thể được sử dụng trong các phép tính này chỉ bằng cách nêu tên của nó. Một ví dụ đơn giản về điều này là:

```
x <- 10
y <- 196
x * y
## [1] 1960
```

Exercise 4.1. Dùng R để tính các phép tính sau:

1. $9^{(1/2)}$
2. $2.8^3 - 2.2^2$
3. $\frac{2^{5+0.003}}{(7+5)8}$
4. $\frac{2^{5+0.003}}{7+5 \times 8}$
5. $6.3 \times 5 \times 10^{-2} + \frac{(7.2)^{1/3}}{4 \times 3 - 5}$

4.1 Vector hóa tính toán

Các hàm này cũng được vector hóa, do đó mỗi phép tính được thực hiện theo từng phần tử:

```
# Cộng
c(2, 3, 4) + c(4, 5, 6)
## [1] 6 8 10
```

```
# Trừ
c(2, 3, 4) - c(4, 5, 6)
## [1] -2 -2 -2
```

```
# Nhân
c(2, 3, 4) * c(4, 5, 6)
## [1] 8 15 24
```

```
# Chia
c(2, 3, 4) / c(4, 5, 6)
## [1] 0.5000000 0.6000000 0.6666667
```

Các phép tính này cũng sẽ được thực hiện theo từng phần tử nếu có hai data frame hoặc hai ma trận tham gia vào phép tính.

Lưu ý rằng nếu độ dài của các vector không khớp, sẽ nhận được thông báo cảnh báo. Trong ví dụ sau, vector đầu tiên có độ dài là 3 và vector thứ hai có độ dài là 4.

```
# Nhân
c(2, 3, 4) + c(4, 5, 6, 7)
## Warning in c(2, 3, 4) + c(4, 5, 6, 7): longer object length is
## [1] 6 8 10 9 not a multiple of shorter object length
```

Cảnh báo này nhắc nhở nên kiểm tra độ dài của các vector để đảm bảo chúng bằng nhau. Nếu truyền vào hai ma trận có kích thước khác nhau, sẽ nhận được thông báo lỗi:

```
# Ma trận đầu kích thước 2x3 và ma trận thứ hai kích thước 2x2
matrix(c(2, 3, 4, 5, 6, 7), nrow = 2) + matrix(c(4, 5, 6, 7), nrow = 2)
## Error in matrix(c(2, 3, 4, 5, 6, 7), nrow = 2) + matrix(c(4, 5,
## 6, 7), : non-conformable arrays
```

Điều này một lần nữa cho thấy nên kiểm tra kích thước ma trận của mình. Lỗi trả về cho các data frame có kích thước không giống nhau sẽ cung cấp nhiều thông tin hơn so với lỗi trả về cho ma trận:

```
data.frame(c(2, 3), c(4, 5), c(6, 7)) + data.frame(c(4, 5), c(6, 7))
## Error in Ops.data.frame(data.frame(c(2, 3), c(4, 5), c(6, 7)),
## data.frame(c(4, : '+' only defined for equally-sized data frames
```

Exercise 4.2. Tạo vector $x = c(3, 5, 7)$ và $y = c(4, 6, 8)$ trong R và tìm:

- $x + y$
- $x - y$
- $x \times y$
- $\frac{x}{y}$

4.2 Các phép tính khác

Logarit và Số mũ

Các phép tính logarit và số mũ được thực hiện bằng cách sử dụng các hàm:

- `log(x, base)` - sẽ lấy logarit của x theo một cơ số cụ thể (nếu không chỉ định giá trị cho cơ số, nó sẽ lấy logarit tự nhiên $\ln x$)
- `exp(x)` - lấy số mũ của e^x .

Ví dụ

```
# log của 1000 theo cơ số 10
log(1000, 10)
## [1] 3
```

```
## Cách khác
```

```
log10(1000)
```

```
## [1] 3
```

```
# Logarit tự nhiên của 10
```

```
log(10)
```

```
## [1] 2.302585
```

```
# e mũ 2
```

```
exp(2)
```

```
## [1] 7.389056
```

Exercise 4.3. Dùng R trả lời các câu hỏi sau:

- Log 50 cơ số 10 là bao nhiêu?
- Log 50 cơ số 2 là bao nhiêu?
- Log tự nhiên của 10 là bao nhiêu?
- Số mũ của 3 là bao nhiêu?

Tính toán Ma trận

Trong một số trường hợp, có thể sẽ phải thực hiện phép nhân ma trận. Toán tử cho phép nhân này là `%*%`, do đó đối với hai ma trận `X` và `Y`, `X %*% Y` sẽ cho ra tích vô hướng của hai ma trận.

```
X <- matrix(c(1, 2, 3, 4), nrow = 2)
```

```
Y <- matrix(c(5, 6, 7, 8), nrow = 2)
```

```
X %*% Y
```

```
##      [,1] [,2]
```

```
## [1,]    23    31
```

```
## [2,]    34    46
```

Lưu ý rằng hai ma trận được nhân phải có kích thước tương thích, nếu không bạn sẽ gặp lỗi về mảng không tương thích. Nếu cần chuyển vị một ma trận hoặc vector, sử dụng hàm `t()`:

```
X <- matrix(c(1, 2, 3, 4, 5, 6, 7, 8), nrow = 2)
```

```
X
```

```
##      [,1] [,2] [,3] [,4]
```

```
## [1,]    1    3    5    7
```

```
## [2,]    2    4    6    8
```

```
t(X)
```

```
##      [,1] [,2]
```

```
## [1,]    1    2
```

```
## [2,]    3    4
```

```
## [3,]    5    6
```

```
## [4,]    7    8
```

Exercise 4.4. Trong R, thực hiện các thao tác sau:

- Tạo một vectơ x chứa các giá trị nguyên từ 1 đến 10.
- Tạo một vectơ z chứa các giá trị 1.48, 2.63, 7.86.
- Tạo một ma trận M như sau

$$\begin{bmatrix} 1.1 & 4.4 & 7.7 \\ 2.2 & 5.5 & 8.8 \\ 3.3 & 6.6 & 9.9 \end{bmatrix}$$

- Tạo đối tượng $a = \frac{3.78 \times 10^{-3}}{0.0832}$.
- Nhân $x \times a$.
- Thực hiện phép nhân $M \times z$.

5 Các hàm (functions) trong R

Hàm là một phần quan trọng của lập trình R. Chúng giúp việc phân tích trở nên dễ dàng hơn nhiều và giảm thiểu nhu cầu phải lập trình lại các phép tính thông thường (ví dụ: giá trị trung bình, phương sai, v.v.). Điều đầu tiên chúng ta cần làm khi sử dụng hàm là chỉ định tên hàm.

Function_name(arg1, arg2, arg3)

Ví dụ, **Function_name** ở đây sẽ là `sqrt` khi xét căn bậc hai hoặc `as.numeric` khi cố gắng ép một vector cụ thể thành dạng số. Tên hàm luôn đứng trước, sau đó là dấu ngoặc tròn. Trong ví dụ này, chúng ta có ba đối số: `arg1`, `arg2`, `arg3`. Điều này có nghĩa là chúng ta phải chỉ định ba tham số trong hàm.

Giả sử chúng ta muốn lấy căn bậc hai của một số. Bạn có thể không chắc chắn 100% về những gì cần phải nhập vào hàm đó, vì vậy bạn có thể muốn xem tệp help. Cách dễ nhất để truy cập tệp help là

```
?sqrt
help(sqrt)
```

Các tệp help cho chúng ta biết những đối số nào cần thiết để hàm hoạt động. Hàm `sqrt()` chỉ yêu cầu một đối số. Hàm yêu cầu một đối số `x`, là một "vector hoặc mảng số thực hay số phức". Về cơ bản, điều này có nghĩa là bạn có thể truyền một tập hợp số, có thể là vector, ma trận, data frame, v.v., và nó sẽ lấy căn bậc hai. Giả sử bạn muốn tính căn bậc hai của 4, chúng ta có thể viết code:

```
sqrt(4)
## [1] 2
```

Câu lệnh này chỉ có một đối số nên bạn không cần chỉ định thêm bất kỳ đối số nào khác. Bạn cũng có thể nêu tên của đối số, ví dụ:

```
sqrt(x = 4)
## [1] 2
```

Điều này sẽ tạo ra cùng một kết quả, nhưng vì chỉ có một đối số ở đây nên cách làm này không thực sự hữu ích. Tuy nhiên, hãy xem xét một ví dụ với nhiều đối số. Hãy xem hàm `log()`. Hàm này có hai đối số. Đối số đầu tiên là `x`, một vector số hoặc phức, và đối số thứ hai được gọi là

base. Base hơi khác một chút vì nó có một toán tử bằng, và toán tử bằng này nói rằng nếu đối số base không được chỉ định, nó sẽ mặc định là một giá trị cụ thể, trong trường hợp này là $\exp(1)$. Vì vậy, nếu chúng ta để base trống, nó sẽ lấy logarit tự nhiên.

```
log(8)
## [1] 2.079442
```

Vì vậy, nếu bạn muốn sử dụng tùy chọn mặc định, bạn không cần phải đưa nó vào hàm. Giả sử chúng ta muốn thay đổi cơ số, chúng ta có thể chỉ định đối số cơ số và như vậy

```
log(8, base = 2)
## [1] 3
```

Miễn là các đối số này theo đúng thứ tự (thứ tự mà R chỉ định), kết quả sẽ chính xác. Vì vậy, trong ví dụ này, kết quả luôn phải là số theo sau là cơ số. Nếu chúng ta bỏ `base =`, R sẽ cho ra kết quả tương tự.

```
log(8, 2)
## [1] 3
```

Nếu hoán đổi chúng với nhau, kết quả sẽ khác.

```
log(2, 8)
## [1] 0.3333333
```

Nếu chúng ta không biết thứ tự chính xác, chúng ta có thể sử dụng tên đối số để chỉ định số.

```
log(base = 2, x = 8)
## [1] 3
```

Một số hàm hiển thị ... dưới dạng đối số. Điều này có thể xảy ra vì nhiều lý do:

1. Hàm có thể nhận một số lượng đối số không xác định, không nhất thiết phải có tên nhất định. Xem tệp help của hàm `c()` dùng để tạo một vector. Các hàm `data.frame()` và `list()` cũng là ví dụ về điều này.
2. Hàm này sử dụng các hàm khác và có những đối số mà có thể truyền cho chúng (nhưng không bắt buộc).
3. Hàm này được các hàm khác sử dụng và thích nghi theo cách nó được sử dụng để đáp lại các đối số được truyền vào bởi các hàm này.

Trong trường hợp đầu tiên, bạn có thể đã biết cách sử dụng dấu ba chấm, hoặc tệp help sẽ cung cấp chi tiết về những đối số nào có thể hữu ích. Trong trường hợp thứ hai và thứ ba, trừ khi bạn đang sử dụng hàm ở mức độ rất nâng cao, bạn không cần phải lo lắng về những đối số này.

Các hàm thường dùng

Có rất nhiều hàm tích hợp thực sự hữu ích trong R. Một số hàm được sử dụng phổ biến nhất được liệt kê dưới đây:

Các hàm tính toán thống kê

- `mean()` - tính toán giá trị trung bình của một vector.
- `median()` - tính toán trung vị của một vector.

- `var()` - tính toán phương sai của một vector.
- `sd()` - tính toán độ lệch chuẩn của một vector.
- `min()` - tính toán giá trị nhỏ nhất của một vector.
- `max()` - tính toán giá trị lớn nhất của một vector.
- `cor()` - tính toán hệ số tương quan cho hai vector hoặc ma trận tương quan cho một data frame

Phân tích thống kê

Nhiều phân tích thống kê dành riêng cho một số ứng dụng nhất định và yêu cầu phải cài đặt gói, nhưng hai phân tích thường được thực hiện là:

- `lm()` - hồi quy tuyến tính.
- `t.test()` - kiểm định t.

Các hàm thông dụng khác

- `is.na()` - phát hiện các giá trị bị thiếu trong một data frame.
- `any()` - kết hợp một kiểm tra điều kiện trên một vector để tìm ra xem có bài giá trị nào trả về giá trị TRUE không.
- `all()` - tương tự như `any()` nhưng kiểm tra xem tất cả các giá trị có trả về giá trị TRUE hay không.
- `which()` - trả về chỉ mục cho mỗi giá trị trong một vector trả về giá trị TRUE cho điều kiện đang kiểm tra.
- `rep()` - tạo ra các vector có giá trị lặp lại.
- `seq()` - tạo ra chuỗi các vector.