

Bài 1

GIỚI THIỆU VỀ R

1 Đôi nét về R

Từ lâu, phân tích số liệu và biểu đồ đã được tiến hành thông qua các phần mềm thông dụng như SAS, SPSS, Stata, Statistica, S-Plus. Nhưng việc duy trì sử dụng lâu dài các phần mềm này gặp trở ngại vì chi phí để sử dụng phần mềm tương đối lớn (có khi lên đến hàng trăm nghìn đô la mỗi năm),. Do đó, các nhà nghiên cứu thống kê trên thế giới đã hợp tác với nhau để phát triển một phần mềm mới, với chủ trương mã nguồn mở, sao cho tất cả các thành viên trong ngành thống kê học và toán học trên thế giới có thể sử dụng một cách thống nhất và hoàn toàn miễn phí. Vì vậy, từ đó R xuất hiện. Vậy R là gì ?

R là phần mềm sử dụng cho phân tích thống kê và vẽ biểu đồ, nhưng về bản chất, R là một ngôn ngữ máy tính đa năng, có thể sử dụng cho nhiều mục tiêu khác nhau, từ tính toán đơn giản cho đến các phân tích thống kê phức tạp. Ngoài ra, vì R là một ngôn ngữ, nên người ta có thể sử dụng R để phát triển thành các phần mềm chuyên môn cho một vấn đề tính toán cá biệt.

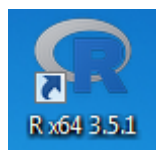
Một trong những nhược điểm nhưng cũng là ưu điểm là R là môi trường làm việc với những dòng lệnh, do đó, ta có thể thấy được và lưu lại đường như toàn bộ quá trình, từ đó ta có thể sử dụng lại, tìm lỗi, sửa lỗi,...

2 Tải và cài đặt R

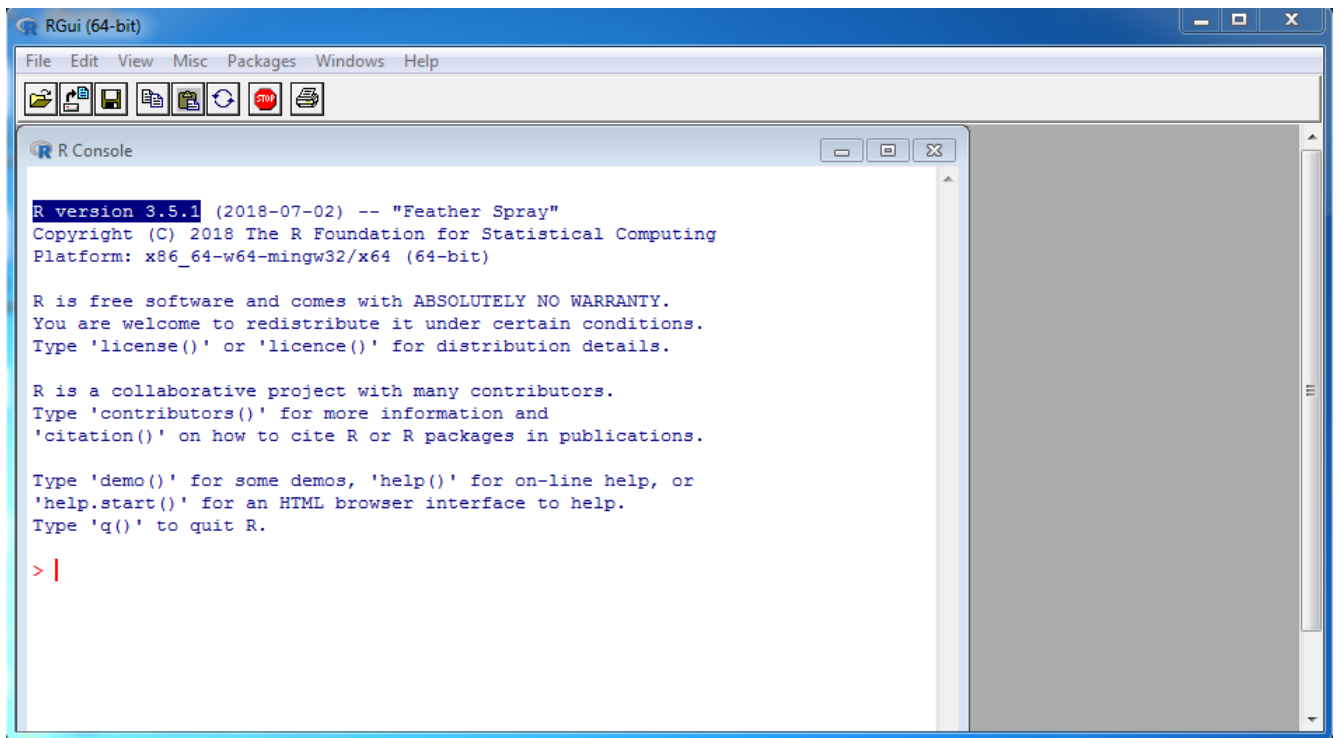
Để sử dụng R, việc đầu tiên là chúng ta phải cài đặt R vào máy của mình. Để làm được việc này, ta tải phần mềm bằng cách truy cập vào trang web: <http://cran.us.r-project.org/>

Chọn phiên bản phù hợp với hệ điều hành và với máy, sau đó nhấn chọn “Download R [phiên bản] for ...”. Khi đã hoàn tất bước tải R, ta tiến hành cài đặt bằng cách click chuột phải vào file tải về trên và làm theo hướng dẫn cài đặt trên màn hình.

Khi hoàn tất bước cài đặt, một icon như sau sẽ xuất hiện trên màn hình máy tính của các bạn.



Đến đây các bạn hãy click đúp vào icon trên và trên màn hình sẽ hiện ra một cửa sổ làm việc như sau (ở đây phiên bản sử dụng là R 3.5.1 nên chúng ta sẽ thấy dòng chữ “R version 3.5.1” như trong hình)



3 Cấu trúc lệnh chung trong R

3.1 Cấu trúc lệnh trong R

R là một ngôn ngữ có tính **tương tác** nghĩa là khi chúng ta ra một lệnh nào đó nếu đúng cú pháp R sẽ **thực hiện và đáp lại kết quả**. Cấu trúc câu lệnh của R thường viết dưới dạng **hàm** theo các thông số (**đôi khi không cần thông số**) kèm theo, những thông số này là những thông số định tính hay định lượng. Cụ thể, cấu trúc câu lệnh chung trong R như sau:

đối tượng \leftarrow hàm (thông số 1, thông số 2,..., thông số n)

hay

đối tượng = hàm (thông số 1, thông số 2,..., thông số n)

thường người dùng được khuyến khích sử dụng dấu mũi tên hơn.

Ví dụ 1:

```
> x <- c()  
> x <- sample(5)  
> x  
[1] 2 1 5 4 3
```

x ở đây là 1 **đối tượng**, còn $c()$ là 1 hàm nhưng không có thông số, $sample$ là 1 hàm với thông số định lượng là 5.

Ví dụ 2:

```
> reg <- lm(y~x)
```

reg ở đây là 1 đối tượng, còn lm là 1 hàm với thông số định tính $y \sim x$. Để biết 1 hàm cần có những thông số nào, chúng ta dùng lệnh **args(...)**, với “...” chứa hàm ta quan tâm.

Ví dụ 3:

```
> args(sample)
function (x, size, replace = FALSE, prob = NULL)
NULL
```

Để biết rõ ràng hơn về hàm nào đó (ý nghĩa, tác dụng, khả năng xử lý,...) ta thực hiện cú pháp:
? tên hàm hoặc **help(tên hàm)**

Ví dụ 4:

```
> ? sample
starting httpd help server ... done
```

Một trang sample.html sẽ được mở ra để mô tả 1 cách rõ ràng cách thức dùng hàm, cú pháp, tác dụng của hàm,... thậm chí có cả 1 số ví dụ cụ thể để cung cấp 1 cái nhìn trực quan về hàm đó cho người dùng. Ngoài ra, các bạn có thể **tự học R thông qua phần chỉ dẫn được tích hợp sẵn trong R bằng cách chọn mục “Help” rồi chọn Html help** và bắt đầu tìm hiểu, các bạn còn có thể tìm hiểu về hàm thông qua mục **“R function(text)”**, các bạn gõ vào tên hàm trong hộp thoại “Help on”.

Khi các bạn quan tâm đến 1 hàm nào đó các bạn cần phải học về cấu trúc và cú pháp hàm của câu lệnh đó. Như chúng ta đã thấy qua mục help thì hầu hết các hàm trên R đều được hỗ trợ bằng các trang html hướng dẫn có kèm theo ví dụ cụ thể. Do đó, cách tốt nhất để các bạn học về hàm là các bạn có thể copy các đoạn ví dụ trong trang html hướng dẫn về hàm vào trang console của R để chạy thử và quan sát kết quả. Các bạn có thể làm việc này một cách nhanh chóng thông qua hàm **example(tên hàm)**.

Ví dụ 5:

```
> example(seq)
```

Và kết quả trên màn hình chúng ta quan sát được là các câu lệnh ví dụ và kết quả số thực tế mà R xuất ra để mô tả về hàm seq (hàm tạo ra 1 dãy số có thứ tự nào đó)

```
seq> seq(0, 1, length.out = 11)
[1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

```
seq> seq(stats::rnorm(20)) # effectively 'along'
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

```
seq> seq(1, 9, by = 2) # matches 'end'
[1] 1 3 5 7 9
```

```
seq> seq(1, 9, by = pi) # stays below 'end'
[1] 1.000000 4.141593 7.283185
```

```
seq> seq(1, 6, by = 3)
[1] 1 4
```

```
seq> seq(1.575, 5.125, by = 0.05)
[1] 1.575 1.625 1.675 1.725 1.775 1.825 1.875 1.925 1.975 2.025 2.075 2.125
[13] 2.175 2.225 2.275 2.325 2.375 2.425 2.475 2.525 2.575 2.625 2.675 2.725
[25] 2.775 2.825 2.875 2.925 2.975 3.025 3.075 3.125 3.175 3.225 3.275 3.325
[37] 3.375 3.425 3.475 3.525 3.575 3.625 3.675 3.725 3.775 3.825 3.875 3.925
[49] 3.975 4.025 4.075 4.125 4.175 4.225 4.275 4.325 4.375 4.425 4.475 4.525
[61] 4.575 4.625 4.675 4.725 4.775 4.825 4.875 4.925 4.975 5.025 5.075 5.125
```

```
seq> seq(17) # same as 1:17, or even better seq_len(17)
```

```
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
```

R còn là 1 ngôn ngữ “đối tượng” (object oriented language), các dữ liệu trong R được chứa trong object, tức nghĩa là khi ta khởi tạo 1 biến thì R sẽ tự động tạo ra 1 vùng nhớ để lưu giá trị của biến. Sau khi ta gán giá trị nào đó cho biến được khởi tạo thì ta có thể dùng biến đó như là đối tượng để thực hiện các tính toán của ta, còn nếu ta không gán giá trị cho biến trước mà lại đem thực hiện các tính toán thì R sẽ tự động báo **lỗi**. Lưu ý rằng đối tượng ở đây rất đa dạng, có thể là số, véc-tơ, ma trận, chuỗi,...

Ví dụ 6:

```
> x <- 5
> y <- 6
> x + y
[1] 11
> z + y
Error: object 'z' not found
```

Theo kết quả trên, ta thấy rằng R sẽ lập tức báo lỗi vì chúng ta gán giá trị cho biến z nhưng lại đem thực hiện tính toán. Ngoài ra, trong R chúng ta cũng có thể vừa gán giá trị cho biến vừa thực hiện các tính toán.

Ví dụ 7:

```
> (z=5) + (t=7) - (x+y)
[1] 1
```

3.2 Một số phép toán so sánh hay logic trong R

$x == 3$	x bằng 3
$x != 3$	x khác 3
$x < y$	x nhỏ hơn y
$x > y$	x lớn hơn y
$x <= y$	x nhỏ hơn hay bằng y
$x >= y$	x lớn hơn hay bằng y
$z <= 5$	z nhỏ hơn hay bằng 5
$z >= 5$	z lớn hơn hay bằng 5
<code>is.na(x)</code>	có phải x là biến trống không
$A \& B$	A và B (so sánh chân trị (AND) trên mỗi thành phần tương ứng của 2 vectơ A và B)
$A \&\& B$	A và B (so sánh chân trị (AND) trên thành phần đầu tiên kể từ trái qua phải của 2 vectơ A và B)
$A B$	A và B (so sánh chân trị (OR) trên mỗi thành phần tương ứng của 2 vectơ A và B)
$A B$	A và B (so sánh chân trị (AND) trên thành phần đầu tiên kể từ trái qua phải của 2 vectơ A và B)
$!A$	phủ định A (NOT A)
$xor(x, y)$	lấy XOR trên mỗi thành phần tương ứng của 2 vectơ x và y

Với các lệnh trên, R sẽ xuất ra kết quả là TRUE hay FALSE.

Ví dụ 8:

```
> x=5
```

```

> x==5
[1] TRUE
> x==6
[1] FALSE
> x>y
[1] FALSE
> !(x==5)
[1] FALSE
> A=c(1,2,3);B = c(3,0,4)
> A
[1] 1 2 3
> B
[1] 3 0 4
> A&B
[1] TRUE FALSE TRUE

```

Lưu ý rằng khi ta muốn hiểu là x có giá trị là 3 không thì R yêu cầu chúng ta viết $x == 3$, còn viết $x = 3$ thì R sẽ hiểu là ta đã gán giá trị 3 cho 1 đối tượng có ký hiệu là x (nhớ rằng vai trò của dấu “=” tương đương với dấu “←” như đã giới thiệu ở trên). Hơn nữa, ta có thể gán 1 giá trị mới đè lên 1 biến nào đó, R sẽ tự hiểu là ta xóa giá trị cũ lưu trong biến đó và đưa vào giá trị mới, chẳng hạn ở những ví dụ trên thì $x = 5$, còn $y = 6$, chúng ta vẫn có thể gán đè giá trị khác lên x và y như sau:

Ví dụ 9:

```

> x=9
> y=4
> x+y
[1] 13

```

Ta cũng có thể gán cùng 1 lúc nhiều biến trên cùng 1 dòng, chỉ cần ngăn cách nhau bởi dấu chấm phẩy ;

Ví dụ 10:

```

> x=2;y=3
> x
[1] 2
> y
[1] 3

```

Trong R, chúng ta có thể sử dụng kí hiệu # để ghi vào các ghi chú riêng cho người dùng, vì tất cả các câu chữ đi sau dấu # đều không có hiệu lực:

Ví dụ 11:

```

> # Lệnh sau đây mô tả việc tạo ra 1 dãy số ngẫu nhiên gồm 8 số có giá trị từ 1 tới 12:
> k=sample(12,8)
> k
[1] 7 10 11 6 1 3 2 9

```

3.3 Các hàm toán học thường dùng trong R:

$\log(x)$	logarit cơ số e
$\log_{10}(x), \log(x, n)$	logarit cơ số 10, cơ số n của x .
$\exp(x)$	e^x
\sqrt{x}	căn bậc 2 của x
$\text{factorial}(x)$	$x!$
$\text{choose}(n, k)$	tổ hợp n chập k
$\text{floor}(x)$	giá trị nguyên $< x$ (sàn của x)
$\text{ceiling}(x)$	giá trị nguyên $> x$ (trần của x)
$\text{trunc}(x)$	làm tròn tới giá trị nguyên gần nhất giữa x và 0
$\text{round}(x, \text{digits} = n)$	làm tròn x đến n chữ số
$\text{signif}(x, \text{digits} = n)$	hiển thị x dưới dạng dấu chấm thập phân, n tổng chữ số hiển thị
$\sin(x), \cos(x), \tan(x)$	hàm sin, cos, tan
$\text{abs}(x)$	$ x $
$x\%/\%y$	lấy phần nguyên của phép chia x/y
$x\%\%y$	lấy phần dư của phép chia x/y

Một số lệnh liên quan

$\text{length}(x)$	chiều dài của x .
$x[i]$	phần tử thứ i của mảng x .
$x[-i]$	tất cả các phần tử của x trừ phần tử thứ i ra.
$x[1:5]$	trích x_1 cho đến x_5 .
$x[c(1, 3, 5)]$	trích các phần tử thứ 1, 3 và 5.
$x[x > 3]$	trích tất cả những phần tử lớn hơn 3.
$x[x < -2 x > 2]$	trích những phần tử $ x > 2$.

Một số hàm về vectơ: Cho vectơ x

$\text{max}(x), \text{min}(x)$	giá trị lớn nhất, bé nhất của x .
$\text{sum}(x)$	tổng các giá trị trong x .
$\text{mean}(x)$	trung bình của x .
$\text{median}(x)$	trung vị của x .
$\text{range}(x)$	bằng $\text{max}(x) - \text{min}(x)$.
$\text{var}(x)$	phương sai của x .
$\text{sort}(x)$	sắp xếp x , mặc định theo thứ tự tăng dần.
$\text{order}(x)$	trả về các vị trí của x khi đã sắp theo thứ tự tăng dần.
$\text{quantile}(x)$	tính các phân vị của x .
$\text{cumsum}(x)$	tổng tích lũy.
$\text{cumprod}(x)$	tích tích lũy.

3.4 Cách đặt tên trong R

Đặt tên 1 đối tượng (object) hay một biến số (variable) trong R khá linh hoạt, vì R không có nhiều giới hạn như các phần mềm khác. Tên một object phải được viết liên nhau (tức không được cách nhau bằng 1 ký tự trống). Chẳng hạn như R chấp nhận “myobject” nhưng sẽ không chấp nhận “my object”. **Ví dụ 1:**

```
> myobject <- sample(6, 4)
> my object <- sample(6, 4)
Error: unexpected symbol in "my_object"
```

Đôi khi để việc gọi tên, gọi nhớ dễ hơn người ta có thể tách rời các ký tự ra bởi dấu chấm “.” như hướng đối tượng, chẳng hạn như “my.object”.

Ví dụ 2:

```
> my.object <- sample(6,4)
```

Điều lưu ý tiếp theo là trong R có phân biệt mẫu tự viết hoa và mẫu tự viết thường tức là `My.object` khác với `my.object`.

Ví dụ 3:

```
> my.object <- 7
> My.object <- 20
> my.object == My.object
[1] FALSE
> my.object == my.object
[1] TRUE
```

Rõ ràng theo ví dụ trên thì R có sự phân biệt giữa 2 đối tượng `my.object` và `My.object`. Một vài điều lưu ý khác khi đặt tên biến trong R:

- Không nên đặt tên biến hay đối tượng mà trong đó có sự xuất hiện các dấu “_” (underscore) như `my_object` hay `my-object`.
- Không nên đặt tên biến hay đối tượng trùng lặp 1 biến trong dữ liệu. Ví dụ như nếu chúng ta có 1 dữ liệu chỉ chứa 1 thành phần là biến `length` thì không nên đặt biến mới có tên là `length` nữa, tức là `length <- length`. Tuy nhiên, trong trường hợp dữ liệu gồm nhiều thành phần, trong đó có thành phần `length` thì ta có thể đặt 1 biến tên `length` như sau: `length <- data$length`.

4 Bước đầu làm quen với R

4.1 Thiết lập thư mục làm việc

Phần mềm R ra đời với mục đích chủ yếu xử lý số liệu thống kê, do đó, R cần một không gian để truy xuất dữ liệu. Vì vậy trước khi sử dụng R, ta cần thiết lập 1 môi trường làm việc cho R thông qua cú pháp: `setwd("Tên ổ đĩa:/Tên thư mục làm việc")`.

```
> setwd("D:/R_Works")
```

Hay trong thanh công cụ ta click chuột để chọn File → Change dir... và ta chọn Folder mà ta muốn môi trường làm việc và các thao tác của R sẽ được thực hiện và lưu lại đó. Trên đây, ta chọn “D:/R_Works” làm môi trường làm việc của chúng ta.

Để xem môi trường làm việc hiện thời ở đâu ta thực hiện cú pháp:

```
> getwd()
[1] "D:/R_Works"
```

và kết quả mà R trả về là “D:/R_Works” cho ta biết thư mục làm việc hiện thời của chúng ta là “D:/R_Works”. Ngoài ra, chúng ta có một số thao tác trên thư mục làm việc của chúng ta như sau:

– Liệt kê các file trong thư mục làm việc:

```
> list.files()
```

hay

```
> dir()
```

– Lưu Workspace đang làm việc (dưới tên là `ten_file`) để tiện cho việc xem lại:

```
> save.image("ten_file.rda")
```

– Khôi phục biến x (thực chất là ta load lại file chứa biến x):

```
> load("ten_file.rda")
```

– Xóa 1 biến khỏi bộ nhớ:

```
> rm(x)
```

– Xóa tất cả các biến:

```
> rm(list = ls())
```

– Liệt kê tất cả các biến hiện hành:

```
> ls()
```

– Xem thông tin của 1 biến nào đó:

```
> str()
```

– Xem thông tin của tất cả các biến đang làm việc:

```
> ls.str()
```

4.2 Nhập dữ liệu

4.2.1 Nhập số liệu trực tiếp

a) Dùng hàm `c()`

Ta có số liệu về độ tuổi và huyết áp của 6 người đi khám tại 1 phòng mạch như sau:

Age	Bloodpress
45	14
47	13
54	15
50	12
43	11
53	13

Chúng ta có thể nhập những số liệu trên vào R bằng cách sử dụng hàm `c()` như sau:

```
> age <- c(45,47,54,50,43,53)
> bloodpress <- c(14,13,15,12,11,13)
```

trong đó dòng lệnh thứ nhất, chúng ta muốn cho R biết rằng chúng ta muốn tạo ra 1 cột dữ liệu (hay còn gọi là 1 biến số) có tên là `age` và dòng thứ 2 là tạo ra cột dữ liệu có tên là `bloodpress`. Hiển nhiên là tên biến có thể thay đổi tùy thích nhưng nên là tên có thể gợi nhớ đến dữ liệu. Để tiện cho việc xử lý và quản lý, người ta thường nhập 2 đối tượng này lại nhau thành 1 `data.frame` bằng cách sử dụng hàm `data.frame`:

```
> benhnhan <- data.frame(age, bloodpress)
> benhnhan
  age bloodpress
1  45          14
2  47          13
3  54          15
4  50          12
5  43          11
6  53          13
```

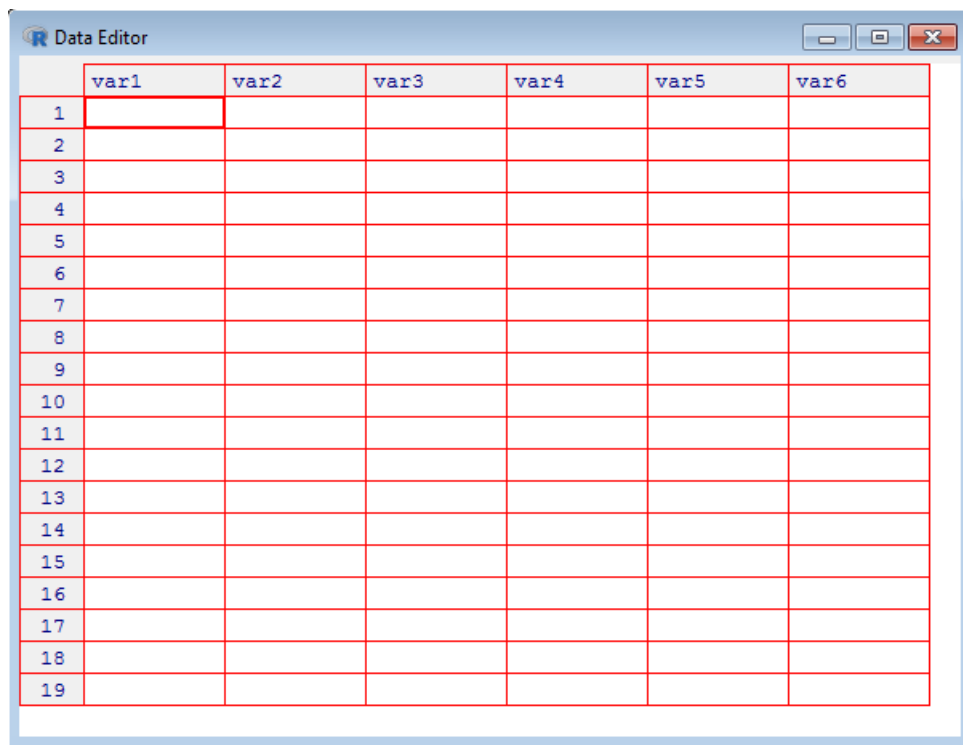

Như đã nói ở phần đặt tên trong R, nếu chỉ muốn làm việc trên thành phần age thôi, ta có thể gán thành phần age trong data.frame benhnhan trên cho 1 biến số (hay đối tượng) cũng mang tên age:

```
> age <- benhnhan$age
> age
[1] 45 47 54 50 43 53
```

b) Dùng lệnh `edit(data.frame())`

Khi gõ hàm này vào thì trong cửa sổ của R sẽ tự động xuất hiện một cửa sổ mới với các dãy cột và dòng giống như trong Excel và chúng ta chỉ việc nhập dữ liệu vào trong bảng này. Mặc định tên biến sẽ là var1, var2,... ta đổi tên biến bằng cách nhấp chuột vào ô var1 và thay đổi bằng cách gõ vào age nhấn Enter, nhấp chuột vào ô var2 và thay đổi bằng gõ bloodpress nhấn Enter. Sau khi nhập xong chúng ta nhấn vào nút X của bảng dữ liệu và lập tức chúng ta sẽ có 1 data.frame như chúng ta muốn:

```
> benhnhan <- edit(data.frame())
```



	var1	var2	var3	var4	var5	var6
1						
2						
3						
4						
5						
6						
7						
8						
9						
10						
11						
12						
13						
14						
15						
16						
17						
18						
19						

Sau khi nhập dữ liệu chúng ta sẽ có hình ảnh sau:

	age	bloodpress	var3	var4	var5	var6
1	45	14				
2	47	13				
3	54	15				
4	50	12				
5	43	11				
6	53	13				
7						
8						
9						
10						
11						
12						
13						
14						
15						
16						
17						
18						
19						

4.2.2 Nhập từ file

a) Nhập từ file *.txt: Dùng hàm read.table

Chúng ta có thể nhập số liệu từ một file .txt bằng hàm read.table (với lưu ý là trước tiên chúng ta phải copy file .txt này vào thư mục làm việc của R).

Ví dụ: Ta muốn phân tích số liệu thống kê số liệu y khoa được lưu dưới file Solieu.txt như sau:

ID	Tuổi	FPSA	TPISA	F_TPISA	V_tuyen	ung_thu
1	66	0.673	2.8	24.03571429	49	0
2	77	0.435	1	43.5	17.3	0
3	82	1.06	3.5	30.28571429	44.1	0
4	79	1.215	2.7	45	26.3	0
5	75	3.607	23.3	15.4806867	76.1	0
6	77	1.712	7	24.45714286	65.4	0
7	78	1.618	6.4	25.28125	52.7	0
8	83	0.371	1	37.1	30.4	0
9	65	5.996	19.6	30.59183673	55.7	0
10	88	11.056	44.2	25.01357466	125.7	0
11	80	0.845	14.3	5.909090909	16	0
12	76	3.534	14.4	24.54166667	72.8	0
13	73	2.97	22.5	13.2	60.5	0
14	81	9.143	16.5	55.41212121	53.8	0
15	77	2.356	17.1	13.77777778	92.6	0
16	52	0.749	1.1	68.09090909	35.2	0
17	77	0.617	0.7	88.14285714	31.2	0
18	66	2.47	18.7	13.20855615	56.4	0
19	65	2.167	3.6	60.19444444	74	0
20	74	0.388	1.2	32.33333333	29.2	0

Ta sẽ gõ cú pháp như sau:

```
> setwd("D:/R_Works")
> solieu <- read.table("Solieu.txt", header = TRUE)
```

Dòng lệnh thứ nhất để R **truy cập đúng địa chỉ** mà file Solieu.txt đang được lưu giữ. Dòng lệnh thứ hai là muốn R nhập số liệu từ file Solieu.txt và **gán vào một biến trong R mang tên solieu**.

Ở đây, phần `header = TRUE` có nghĩa là yêu cầu R **đọc dòng đầu tiên** trong file đó như là tên của từng cột dữ liệu.

Thực tế, chúng ta phải làm việc với dữ liệu rất lớn, có thể nhiều file dữ liệu, do đó để tiện quản lý dữ liệu, ta sẽ tạo một thư mục mới có tên data trong thư mục làm việc R_Works và chuyển tất cả file chứa dữ liệu vào thư mục mang tên data này. Ở ví dụ này chính là file Solieu.txt được mang vào thư mục data. Ta có cú pháp sau khi đã đem file Solieu.txt vào thư mục data như sau:

```
> setwd("D:/R_Works/data")
> solieu <- read.table("Solieu.txt", header = TRUE)
```

Nếu ta kiểm lại thì kết quả hoàn toàn giống nhau.

Nếu muốn kiểm tra xem R đã đọc hết các dữ liệu chưa thì chúng ta có thể ra lệnh:

```
> solieu
hay
```

```
> names(solieu)
```

Ở đây hàm names dùng để mô tả trong dữ liệu bao gồm những cột nào và tên là gì

```
> names(solieu)
[1] "ID"      "Tuoi"    "FPSA"    "TPSA"    "F_TPSA"  "V_tuyen" "Ung_thu"
```

Để lưu lại biến này trong R (tiện lợi cho việc xử lý về sau) thì như đã giới thiệu ở trên, ta dùng hàm save:

```
> save(solieu, file="solieu.rda")
```

b) Nhập từ file excel: Dùng read.csv

R có thể nhận biết và đọc dữ liệu từ một file Excel với lưu ý là file dữ liệu trong Excel được lưu trữ dưới đuôi mở rộng là .csv. Điều đó có nghĩa là ta sẽ tiến hành như sau: Trước tiên khi đã nhập dữ liệu vào Excel (hay 1 dữ liệu có sẵn) ta sẽ dùng lệnh “Save as” trong Excel và lưu lại dưới đuôi “.csv”, sau đó chúng ta sẽ dùng lệnh read.csv trong R để đọc dữ liệu này:

Ví dụ: Ta có một dữ liệu Excel là data01.xls, đầu tiên ta sẽ mở file này rồi chọn File → Save as, ta ghi tên file và đuôi là data01.csv và click Save. Sau đó bằng lệnh read.csv ta có thể nhập dữ liệu từ file data01.csv trên vào R như sau:

```
> setwd("D:/R_Works/data")
> test <- read.csv("data01.csv", header = TRUE)
> test
```

	Age	FPSA	TPSA	K
1	66	0.673	2.80	0
2	77	0.435	1.00	0
3	82	1.060	3.50	0
4	79	1.215	2.70	0
5	75	3.607	23.30	0
6	77	1.712	7.00	0
7	78	1.618	6.40	0
8	83	0.371	1.00	0
9	65	5.996	19.60	0
10	88	11.056	44.20	0
11	80	0.845	14.30	0
12	76	3.534	14.40	0
13	73	2.970	22.50	0

14	81	9.143	16.50	0
15	77	2.356	17.10	0
16	52	0.749	1.10	0
17	77	0.617	0.70	0
18	66	2.470	18.70	0
19	65	2.167	3.60	0
20	74	0.388	1.20	0
21	67	0.389	2.40	0
22	76	0.206	1.40	0
23	59	0.516	5.10	0
24	67	0.830	5.70	0
25	60	1.029	3.10	0
26	67	0.320	1.10	0
27	80	1.681	10.30	0
28	84	0.713	3.70	0
29	75	1.072	6.10	0
30	82	0.610	7.40	0
31	73	1.279	9.50	0
32	67	1.798	5.90	0
33	86	1.649	9.80	0
34	67	5.338	14.00	0
35	75	1.287	10.20	0
36	82	4.118	8.70	0
37	55	0.832	6.50	0
38	82	5.626	9.40	0
39	83	0.081	0.30	0
40	82	1.535	7.70	0
41	80	3.475	25.80	0
42	72	0.257	1.30	0
43	86	1.694	8.00	0
44	64	0.592	4.20	0
45	83	13.373	110.00	1
46	80	14.376	45.20	1
47	71	5.771	100.70	1
48	77	7.863	70.40	1
49	79	0.915	8.50	1
50	70	5.385	68.00	1
51	86	18.221	100.30	1
52	72	15.669	91.00	1
53	82	0.832	11.10	1
54	82	13.097	100.15	1

Ta có thể lưu lại để sử dụng về sau bằng hàm save:

```
> save(test, file="data01.rda")
```

Trên đây, chúng ta đã làm quen với một số cách đọc dữ liệu trong R trực tiếp cũng như gián tiếp thông qua các file dữ liệu. Ngoài ra, đối với việc tạo ra các dãy số hay dữ liệu có tính thứ tự chúng ta còn có thể sử dụng một số lệnh hữu ích dưới đây:

4.2.3 Tạo dãy số bằng các hàm seq, rep, gl

R có tính năng đặc biệt hữu ích là tạo ra các dãy số có tính trật tự hoặc thứ tự nhất định rất tiện cho việc mô phỏng, thiết kế thí nghiệm và tạo ra các ví dụ.

a) Lệnh seq

Cấu trúc tổng quát của lệnh seq : “seq(from = 1, to = 1, by = ((to - from)/(length.out - 1)), length.out = NULL, along.with = NULL,...)”, các bạn có thể tham khảo thêm trong mục help của R hay đơn giản với cú pháp ?seq trong console. Dưới đây là một số ví dụ cụ thể và có tính trực quan.

- Tạo ra một dãy số (dạng vector) từ 1 đến 6:

```
> x <- seq(6)
> x
[1] 1 2 3 4 5 6
```

- Tạo ra một dãy số từ 3 đến 15

```
> x <- seq(3,15)
> x
[1] 3 4 5 6 7 8 9 10 11 12 13 14 15
```

- Tạo ra một dãy số từ 17 đến 5 (chú ý cú pháp này dùng cho dãy chạy ngược)

```
> x <- seq(17,5)
> x
[1] 17 16 15 14 13 12 11 10 9 8 7 6 5
```

- Ta có thể không cần viết hàm seq ra màn hình, tạo ra dãy số từ 15 đến 4

```
> x <- (15:4)
> x
[1] 15 14 13 12 11 10 9 8 7 6 5 4
```

- Tạo ra hai dãy số từ 1 đến 23 với công sai là 2 và công sai là π

```
> seq(1,23,by=2)
[1] 1 3 5 7 9 11 13 15 17 19 21 23
> seq(1,23,by=pi)
[1] 1.000000 4.141593 7.283185 10.424778 13.566371 16.707963 19.849556
[8] 22.991149
```

- Tạo ra dãy số từ 2.55 tới 3.15 với công sai 0.05

```
> seq(2.55,3.15,by=0.05)
[1] 2.55 2.60 2.65 2.70 2.75 2.80 2.85 2.90 2.95 3.00 3.05 3.10 3.15
```

- Tạo ra một dãy số gồm 8 số với số nhỏ nhất là 4 và số lớn nhất là 16

```
> seq(length=8,from=4,to=16)
[1] 4.000000 5.714286 7.428571 9.142857 10.857143 12.571429 14.285714
[8] 16.000000
```

b) Lệnh rep

Lệnh rep dùng để mô phỏng một dãy số gồm các giá trị nào đó và được lặp lại với một số lần do chúng ta quy định. Cấu trúc tổng quát rất đơn giản “rep(x,times)”, trong đó x là một số hay một dãy số và times là số lần lặp lại đối tượng x đó. Dưới đây là một số ví dụ trực quan cho cách dùng hàm rep.

- Tạo ra một dãy số gồm toàn số 3 được lặp lại 4 lần:

```
> rep(3,4)
[1] 3 3 3 3
```

- Tạo ra một vectơ gồm các giá trị từ 3 đến 7 và được lặp lại 4 lần:

```
> rep(3:7,4)
[1] 3 4 5 6 7 3 4 5 6 7 3 4 5 6 7 3 4 5 6 7
```

- Tạo ra một vectơ gồm các giá trị từ 2 đến 8 với công sai 2 và được lặp lại 3 lần:

```
> rep(seq(2,8,by=2),3)
[1] 2 4 6 8 2 4 6 8 2 4 6 8
```

- Tạo ra một dãy số gồm ba giá trị 1.2, 1.3, 1.8 được lặp lại 4 lần:

```
> rep(c(1.2,1.3,1.8),4)
[1] 1.2 1.3 1.8 1.2 1.3 1.8 1.2 1.3 1.8 1.2 1.3 1.8
```

- Tạo ra một dãy số gồm bốn số 1,2,3,4 và với số lần lặp của mỗi số được quy định theo một vectơ c(2,3,4,5):

```
> rep(1:4,c(2,3,4,5))
[1] 1 1 2 2 2 3 3 3 3 4 4 4 4 4
```

- Tạo một dãy gồm bốn số 1,2,3,4 và mỗi số có số lần lặp là 2:

```
> rep(1:4,each=2)
[1] 1 1 2 2 3 3 4 4
```

c) Dùng hàm gl

Hàm gl được dùng để tạo ra các **biến cố có thứ bậc**, nói một cách ngắn gọn đó là các **biến đếm**. Cú pháp tổng quát là: “gl(n,k,length=n*k,labels = 1:n,ordered = FALSE)”, ta có thể tìm hiểu thêm qua phần help. Dưới đây là một số ví dụ đơn giản và trực quan:

- Tạo ra biến gồm hai bậc 1 và 2, mỗi bậc được lặp lại 6 lần:

```
> gl(2,6)
[1] 1 1 1 1 1 1 2 2 2 2 2 2
Levels: 1 2
```

- Tạo ra biến gồm bậc 1 và 2, mỗi bậc được lặp lại 3 lần và cứ lặp như vậy cho đến khi dãy có độ dài là 24:

```
> gl(2,3,length=24)
[1] 1 1 1 2 2 2 1 1 1 2 2 2 1 1 1 2 2 2 1 1 1 2 2 2
Levels: 1 2
```

- Có 2 ký tự A và B , tạo dãy gồm 2 ký tự này và mỗi ký tự lặp lại 4 lần:

```
> gl(2,4,label=c("B","A"))
[1] B B B B A A A A
Levels: B A
```

4.3 Các dạng số liệu

4.3.1 Kiểu dữ liệu nhân tố (factor)

Kiểu dữ liệu factor (hay còn gọi là nhân tố) không dùng để tính toán mà dùng để phân loại.

Ví dụ: Ta có dữ liệu về điểm và số thứ tự sinh viên tương ứng đạt được những điểm ấy:

```
> stt <- c(4,6,5,7)
> mark <- c(6,7,9,10)
> class <- data.frame(mark, stt)
> class
mark stt
1      6  4
2      7  6
3      9  5
4     10  7
> mean(class$mark)
[1] 8
> mean(class$stt)
[1] 5.5
```

Rõ ràng ở đây chúng ta thấy rằng “stt” chỉ thể hiện số thứ tự của sinh viên nên nó là một nhân tố vì nếu đem lấy giá trị trung bình chẳng hạn thì sẽ chẳng có ý nghĩa gì về mặt thực tế cả. Nhưng trong đây, “stt” lại bị R hiểu nhầm là một biến số học, do đó chúng ta cần sử dụng hàm `factor(...)` để biến “stt” thành một biến nhân tố theo đúng bản chất của nó.

```
> sott.sv <- factor(stt)
> sott.sv
[1] 4 6 5 7
Levels: 4 5 6 7
```

Bây giờ thì R thông báo cho chúng ta biết `sott.sv` có 4 bậc: 4,5,6,7. Nếu chúng ta yêu cầu R tính trung bình cộng của `sott.sv` thì R lập tức sẽ báo lỗi vì bây giờ nó đã hiểu `sott.sv` không còn là biến số học nữa.

```
> mean(sott.sv)
[1] NA
Warning message:
In mean.default(sott.sv) : argument is not numeric or logical: returning NA
```

Hay chúng ta có thể kiểm tra:

```
> is.na(sott.sv)
[1] FALSE FALSE FALSE FALSE
```

Theo kết quả trên, R báo cho chúng ta biết `sott.sv` bây giờ không còn là biến số học nữa.

4.3.2 Kiểu dữ liệu chuỗi

Trong R có hỗ trợ kiểu dữ liệu chuỗi. Ví dụ như trong R thì ta có thể tạo một vectơ các ký tự và lưu trữ chúng trong một data.frame.

Ví dụ 1:

```
> char <- c('a', 'b', 'c')
> char
[1] "a" "b" "c"
```

Ví dụ 2:

```
> v <- c('the', 'rain', 'in', 'Vietnam')
> df <- data.frame(factors = v, strings = v)
```

Theo mặc định, R sẽ chuyển chuỗi thành nhân tố.

```
> c(is.character(df$factors), is.character(df$string))
[1] FALSE FALSE
```

Chúng ta chuyển cột thứ hai của df sang dạng kí tự.

```
> df[,2] <- as.character(df[,2])
> c(is.character(df$factors), is.character(df$string)) [1] FALSE TRUE
Chúng ta có thể chuyển định dạng chuỗi sang nhân tố với
> options(stringsAsFactors = FALSE)
```

4.3.3 Kiểu dữ liệu Ma trận

a) Khởi tạo ma trận

Trong R chúng ta có thể thể hiện và thực hiện các tính toán trên lớp các ma trận. Ví dụ như ta muốn tạo ra một ma trận, chúng ta có hàm “matrix” và thực hiện như sau:

```
> t <- c(3,5,7,1,6,6,3,9,11)
> # Thông báo cho R là ta muốn có ma trận có số dòng là 3.
> A <- matrix(t, nrow = 3)
> A
      [,1] [,2] [,3]
[1,]    3    1    3
[2,]    5    6    9
[3,]    7    6   11
```

Ở trên thì R mặc định là sẽ rút các phần tử từ t ra và sắp xếp từ trên xuống dưới (sắp theo cột) trước. Nếu ta muốn sắp theo dòng thì ta làm như sau:

```
> B <- matrix(t, nrow = 3, byrow = TRUE)
> B
      [,1] [,2] [,3]
[1,]    3    5    7
[2,]    1    6    6
[3,]    3    9   11
```

Hay nói cách khác B chính là ma trận chuyển vị của A. Để tìm ma trận chuyển vị ta có thể dùng hàm t() như sau:


```
> C <- t(A)
> C
      [,1] [,2] [,3]
[1,]    3    5    7
[2,]    1    6    6
[3,]    3    9   11
```

Ta có thể khởi tạo một ma trận bất kì mà các phần tử đều bắt đầu bằng một số nào đó tùy thích.

> # Tạo ma trận 3x3 với các phần tử đều bằng 0.

```
> (A <- matrix(0,3,3))
      [,1] [,2] [,3]
[1,]    0    0    0
[2,]    0    0    0
[3,]    0    0    0
```

Khởi tạo các giá trị trên đường chéo bằng lệnh diag(..)

> # Cho các phần tử trên đường chéo đều bằng 1

```
> diag(A) <- 1
> diag(A)
[1] 1 1 1
```

> # Bây giờ ta thấy A chính là một ma trận đơn vị

```
> A
      [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
```

Ta có thể kiểm tra xem đối tượng nào đó có phải là một ma trận hay không bằng cách dùng hàm is.matrix(..)

```
> is.matrix(A)
[1] TRUE
```

b) Chiết phần tử từ ma trận

Cũng giống như trong phần vectơ, chúng ta cũng có thể chiết xuất một hoặc một số phần tử từ ma trận một cách dễ dàng với cách viết rất thân thiện.

```
> A <- matrix(1:9, nrow = 3, byrow = TRUE)
```

```
> A
      [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
```

> # Cột 1 của ma trận A

```
> A[,1]
[1] 1 4 7
```

> # Dòng 2 của ma trận A

```
> A[2,]
[1] 4 5 6
```

> # Cột 1 và 3 của ma trận A

```
> A[, c(1,3)]
      [,1] [,2]
[1,]    1    3
[2,]    4    6
[3,]    7    9
```

> # Cột 1, 3 và dòng 2, 3 của ma trận A

```
> A[c(2,3), c(1,3)]
      [,1] [,2]
[1,]    4    6
[2,]    7    9
```

> # Phần tử dòng 2 cột 3 của ma trận A

```
> A[2,3]
[1] 6
```

> # Tất cả các phần tử của ma trận ngoại trừ dòng 1

```
> A[-1,]
      [,1] [,2] [,3]
[1,]    4    5    6
[2,]    7    8    9
```

> # Tất cả các phần tử của ma trận ngoại trừ cột 2

```
> A[, -2]
      [,1] [,2]
[1,]    1    3
[2,]    4    6
[3,]    7    9
```

```
> C <- matrix(1:36, nrow = 6)
```

> # Lấy ra các phần tử có chỉ số dòng là chẵn và chỉ số cột là lẻ

```
> C[seq(2,6,by = 2), seq(1,6,by = 2)]
      [,1] [,2] [,3]
[1,]    2   14   26
[2,]    4   16   28
[3,]    6   18   30
```

Ta có các phép so sánh với ma trận

> # Xét trong ma trận phần tử nào lớn hơn 5

```
> A > 5
      [,1] [,2] [,3]
[1,] FALSE FALSE FALSE
[2,] FALSE FALSE  TRUE
[3,]  TRUE  TRUE  TRUE
```

> # So sánh chân trị trên các thành phần ma trận

```
> D <- matrix(1,3,3)
> diag(D) <- 0
> A & D
      [,1] [,2] [,3]
[1,]    0    0    0
[2,]    1    1    1
[3,]    1    1    1
```

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

> # Phủ định các chân trị của các phần tử trong D

> !D

```
[,1] [,2] [,3]
[1,]  TRUE FALSE FALSE
[2,] FALSE  TRUE FALSE
[3,] FALSE FALSE  TRUE
```

c) Tính toán với ma trận

Ta có thể cộng hay trừ hai ma trận cùng cấp với nhau. Ví dụ như sau:

> A + D - B

```
[,1] [,2] [,3]
[1,] -2 -2 -3
[2,]  4 -1  1
[3,]  5  0 -2
```

> # Tạo ma trận gồm 3 dòng 4 cột và các giá trị từ 1 đến 12

> U <- matrix(1:12, 3, 4)

> V <- matrix(seq(-3,20,by = 2), 3, 4)

> U + V

```
[,1] [,2] [,3] [,4]
[1,] -2  7 16 25
[2,]  1 10 19 28
[3,]  4 13 22 31
```

Ta có thể nhân hai ma trận với nhau (với điều kiện là số cột của ma trận đứng trước phải bằng với số dòng của ma trận viết sau).

> A %*% V

```
[,1] [,2] [,3] [,4]
[1,] -2 34 70 106
[2,] -11 79 169 259
[3,] -20 124 268 412
```

> U %*% V

Error in U %*% V : non-conformable arguments

Ta thấy trong trường hợp 2 thì R lập tức báo lỗi vì cấp hai ma trận không tương thích với nhau. Thêm một lưu ý là ở đây nếu ta chỉ dùng dấu “*” thì R hiểu là ta tạo một ma trận mới với các phần tử là tích các phần tử tương ứng của hai ma trận kia (điều kiện bây giờ là hai ma trận phải cùng số dòng và số cột với nhau).

> A * V

Error in A * V : non-conformable arrays

> U * V

```
[,1] [,2] [,3] [,4]
[1,] -3 12 63 150
[2,] -2 25 88 187
[3,]  3 42 117 228
```

Ta có thể tính toán một số đại lượng quan trọng liên quan tới ma trận như là định thức (điều kiện là ma trận đó phải là **ma trận vuông**), trị riêng và vectơ riêng.

```
> # Tính định thức
> det(A + D - B)
[1] -45
> det(U*V)
Error in determinant.matrix(x, logarithm = TRUE, ...) :
  'x' must be a square matrix

> # Tính trị riêng, vectơ riêng
> eigen(A)
eigen() decomposition
$`values`
[1] 1.611684e+01 -1.116844e+00 -1.303678e-15

$`vectors`
      [,1]      [,2]      [,3]
[1,] -0.2319707 -0.78583024 0.4082483
[2,] -0.5253221 -0.08675134 -0.8164966
[3,] -0.8186735 0.61232756 0.4082483
```

Ta có thể tính nghịch đảo B^{-1} của một ma trận B bằng cách dùng hàm `solve(B)` như sau:

```
> solve(B)
      [,1] [,2] [,3]
[1,] 1.500 1.0 -1.500
[2,] 0.875 1.5 -1.375
[3,] -1.125 -1.5 1.625
```

4.4 Biên tập dữ liệu

4.4.1 Tách rời dữ liệu

Trước khi tiến hành làm biên tập với dữ liệu, chúng ta phải đưa dữ liệu vào hệ thống của R để có thể xử lý trên R bằng hàm `attach()`

```
> setwd("D:/R_Works/data")
> dulieu <- read.csv("basalmet_2.csv", header = TRUE)
> attach(dulieu)
```

Nếu chúng ta muốn phân tích riêng chon nam giới thôi thì chúng ta có thể tách dữ liệu “dulieu” ra thành hai data frame là Nam và Nu. Để thực hiện việc này chúng ta dùng lệnh `subset(data, cond)`, với `data` là dữ liệu của chúng ta còn `cond` là điều kiện. Ví dụ:

```
> Nam <- subset(dulieu, Sex == "nam")
> Nu <- subset(dulieu, Sex == "nu")
> Nam
Sex age    wt    ht metab
1  nam  29 0.100 0.957 0.502
2  nam  37 0.223 0.997 0.524
6  nam  47 0.199 0.999 0.549
8  nam  49 0.272 1.041 0.574
10 nam  52 0.288 1.043 0.597
```

```
11 nam 56 0.272 1.050 0.576
12 nam 64 0.212 1.042 0.672
```

Chúng ta có thể tách rời dữ liệu thành nhiều data frame khác nhau tùy theo điều kiện khác nhau và dựa vào những biến khác nữa. Ví dụ:

```
> Old <- subset(dulieu, age >= 50)
> Old
Sex age wt ht metab
9 nu 50 0.207 1.027 0.594
10 nam 52 0.288 1.043 0.597
11 nam 56 0.272 1.050 0.576
12 nam 64 0.212 1.042 0.672
13 nu 66 0.328 1.060 0.634
14 nu 68 0.270 1.061 0.633
> dim(Old)
[1] 6 5
```

Hay là những người trên 50 tuổi và là nữ giới chẳng hạn:

```
> Nu50 <- subset(dulieu, age >= 50 & Sex == "nu")
> Nu50
Sex age wt ht metab
9 nu 50 0.207 1.027 0.594
13 nu 66 0.328 1.060 0.634
14 nu 68 0.270 1.061 0.633
```

4.4.2 Chiết dữ liệu từ một data frame

Trong R chúng ta có thể **tách ra các cột dữ liệu** mà chúng ta quan tâm bằng kí hiệu mà chúng ta đã biết trong phần giới thiệu về cách rút trích dữ liệu từ một véc tơ đó là kí hiệu [...]. Ví dụ như ta chỉ quan tâm đến 3 cột dữ liệu là Sex, wt và ht (tương ứng ở các vị trí 1, 3 và 4) thì chúng ta có thể cắt chúng ra như sau:

```
> cutdulieu <- dulieu[, c(1,3,4)]
> cutdulieu
Sex wt ht
1 nam 0.100 0.957
2 nam 0.223 0.997
3 nu 0.223 1.001
4 nu 0.220 1.025
5 nu 0.212 1.013
6 nam 0.199 0.999
7 nu 0.220 1.016
8 nam 0.272 1.041
9 nu 0.207 1.027
10 nam 0.288 1.043
11 nam 0.272 1.050
12 nam 0.212 1.042
13 nu 0.328 1.060
14 nu 0.270 1.061
```

4.4.3 Nhập hai data frame thành một

Trong thực tế có những lúc chúng ta cần phải nhập hai data frame vào thành một data frame mới mà thôi. Để làm việc này chúng ta dùng lệnh `merge(data.frame1, data.frame2, ...)`. Giả sử chúng ta cần nhập 2 data frame như sau: data.frame1 gồm có 3 cột Sex, age, và wt gồm những người từ tuổi 50 trở lên:

```
> data1 <- subset(dulieu, age >= 50)
> dataframe1 <- data1[, 1:3]
> dataframe1
  Sex age   wt
9   nu  50 0.207
10  nam  52 0.288
11  nam  56 0.272
12  nam  64 0.212
13  nu   66 0.328
14  nu   68 0.270
```

Và data.frame2 gồm 3 cột Sex, age và ht gồm những người có độ tuổi từ 50 trở xuống:

```
> data2 <- subset(dulieu, age < 50)
> dataframe2 <- data2[, c(1, 2, 4)]
> dataframe2
  Sex age   ht
1  nam  29 0.957
2  nam  37 0.997
3  nu   42 1.001
4  nu   44 1.025
5  nu   46 1.013
6  nam  47 0.999
7  nu   47 1.016
8  nam  49 1.041
```

Ta thêm vào thành phần số thứ tự (ID) cho mỗi data frame.

```
> dataframe1 = data.frame(ID = 1:6, dataframe1)
> dataframe2 = data.frame(ID = 1:8, dataframe2)
```

Ta thấy 2 hệ dữ liệu này có chung 2 yếu tố là Sex và age nhưng dữ liệu dataframe1 thì có 6 dòng, dataframe2 thì có 8 dòng. Chúng ta sẽ nhập 2 data frame này lại thành chỉ 1 data frame thôi bằng cách dùng lệnh merge và phân loại theo ID như sau:

```
> uniqueframe <- merge(dataframe1, dataframe2, by = "ID", all = TRUE)
> uniqueframe
  ID Sex.x age.x   wt Sex.y age.y   ht
1  1   nu   50 0.207  nam   29 0.957
2  2  nam   52 0.288  nam   37 0.997
3  3  nam   56 0.272   nu   42 1.001
4  4  nam   64 0.212   nu   44 1.025
5  5   nu   66 0.328   nu   46 1.013
6  6   nu   68 0.270  nam   47 0.999
7  7 <NA>   NA   NA   nu   47 1.016
8  8 <NA>   NA   NA  nam   49 1.041
```

4.4.4 Biến đổi số liệu

Trong xử lý thống kê đôi khi người ta cần phải biến đổi số liệu từ một biến liên tục sang thành một **biến mang tính phân loại**, hay nói một cách ngắn gọn ta biến một dữ liệu gồm các số liên tục thành các vùng số liệu. Giả sử ở ví dụ trên, chúng ta quan tâm đến dữ liệu “wt” của 14 người ở 3 mức độ khác nhau: mức 1 là dưới 0.175, mức 2 là từ 0.175 đến 0.25 và mức 3 là từ 0.25 trở lên. Ta sẽ tiến hành các bước phân mức ra như sau:

Đầu tiên ta chiết dữ liệu cần quan tâm là biến “wt” ra:

```
> (wt <- dulieu$wt)
[1] 0.100 0.223 0.223 0.220 0.212 0.199 0.220 0.272 0.207 0.288 0.272 0.212
[13] 0.328 0.270
```

Sau đó ta tiến hành phân mức và đặt tên cho 3 mức đó lần lượt là 1, 2 và 3, đồng thời lưu vào một biến khác có tên là “muc”.

```
> muc <- wt
> muc[wt <= 0.175] <- 1
> muc[wt > 0.175 & wt <= 0.25] <- 2
> muc[wt > 0.25] <- 3

> # Ta tạo thành 1 data.frame là muc.wt
> (muc.wt <- data.frame(wt, muc))
wt muc
1 0.100 1
2 0.223 2
3 0.223 2
4 0.220 2
5 0.212 2
6 0.199 2
7 0.220 2
8 0.272 3
9 0.207 2
10 0.288 3
11 0.272 3
12 0.212 2
13 0.328 3
14 0.270 3
```

4.4.5 Lập trình với R - Script

Để tiến hành lập trình hay thiết lập một hàm mới, đầu tiên ta nên mở ra một trang script bằng cách click chuột vào File, chọn New script, tức thì một trang trắng hiện ra để chúng ta có thể viết lên trên đó (gọi là script), và bây giờ chúng ta có thể gõ vào tùy thích đoạn code của chúng ta. Tất nhiên chúng ta vẫn có thể tiến hành lập trình thẳng trên console của R, tuy nhiên chúng tôi khuyến khích nên dùng script hơn vì trong script thì chúng ta có thể di chuyển con trỏ để chỉnh sửa ở bất kì vị trí nào, còn trong console thì chúng ta không thể thực hiện được điều đó.

Sau khi đã gõ xong phần code của chúng ta thì chúng ta tô đen phần code rồi click chuột vào phần “Run lines or selection” trên thanh công cụ để chạy đoạn code chúng ta vừa soạn thảo. Tuy nhiên chúng ta có thể dùng tổ hợp phím Ctrl + A để tô đen đoạn code rồi bấm tổ hợp phím Ctrl + R để chạy đoạn code.

```
D:\GD\Thuc hanh LTTK\code\LamquenvoiR.R - R Editor
#Chúng ta nên ghi chú lại thông tin của script ví dụ như:
# Tên dự án: Làm quen với R - Script
# Ngày thực hiện: DD / MM / YY
# Người thực hiện: Nguyen Van A
#-----//-----
#Thiết lập thư mục làm việc
setwd("D:/R_Works/data")
#Đọc dữ liệu và gán cho biến dulieu
dulieu <- read.csv("basalmet_2.csv", header = TRUE)
#Đưa dữ liệu vào R
attach(dulieu)
(wt <- dulieu$wt)
muc <- wt
muc[wt <= 0.175] <- 1
muc[wt > 0.175 & wt <= 0.25] <- 2
muc[wt > 0.25] <- 3
# Ta tạo thành 1 data.frame là muc.wt
(muc.wt <- data.frame(wt, muc))
```

Ta có thể lưu lại Script hay hàm do chúng ta tự đặt để dùng cho những lần khác bằng cách vào File, chọn Save as, rồi đặt tên và lưu lại Script hay hàm đó. Sau này khi cần sử dụng lại thì ta chỉ cần **chọn đường dẫn** vào đúng nơi ta lưu những Script hay hàm đó rồi truy xuất lại bằng cách gõ lệnh **source('ten Script.R')** hay **source('ten ham.R')**.

BÀI TẬP

Bài 1 Định nghĩa

```
> x<-c(4,2,6)
> y<-c(1,0,-1)
```

Đoán kết quả của các lệnh sau đây:

- a) $length(x)$
- b) $sum(x)$
- c) $sum(x^2)$
- d) $x + y$
- e) $x * y$
- f) $x - 2$
- g) x^2

Bài 2: Đoán các dãy sau đây và dùng R để kiểm tra lại:

- a) $7 : 11$
- b) $seq(2, 9)$
- c) $seq(2, 4, by = 2)$
- d) $seq(3, 30, length = 10)$

e) $seq(6, -4, by = -2)$

Bài 3 Xác định các kết quả của các biểu thức R sau đây, và sau đó sử dụng R để kiểm tra:

a) $rep(2, 4)$

b) $rep(c(1, 2), 4)$

c) $rep(c(1, 2), c(4, 4))$

d) $rep(1 : 4, 4)$

e) $rep(1 : 4, rep(3, 4))$

Bài 4 Sử dụng hàm `rep` để xác định các vector sau trong R.

a) 6, 6, 6, 6, 6, 6

b) 5, 8, 5, 8, 5, 8, 5, 8

c) 5, 5, 5, 5, 8, 8, 8, 8

Bài 5 Nếu $x <- c(5, 9, 2, 3, 4, 6, 7, 0, 8, 12, 2, 9)$ xác định biểu thức sau là gì và sử dụng R để kiểm tra lại:

a) $x[2]$

b) $x[2 : 4]$

c) $x[c(2, 3, 6)]$

d) $x[c(1 : 5, 10 : 12)]$

e) $x[-(10 : 12)]$

Bài 6 Dữ liệu $y <- c(33, 44, 29, 16, 25, 45, 33, 19, 54, 22, 21, 49, 11, 24, 56)$ chứa doanh thu bán sữa theo lít trong 5 ngày trong ba cửa hàng khác nhau (3 giá trị đầu là cho cửa hàng 1, 2 và 3 vào Thứ Hai, v.v.) Tạo một tóm tắt thống kê về doanh thu cho từng ngày trong tuần và cũng vậy cho mỗi cửa hàng.

(HD: Dùng lệnh `summary(y[1:3], ...)`)

Bài 7 Tạo trong R các ma trận

$$x = \begin{bmatrix} 3 & 2 \\ -1 & 1 \end{bmatrix}$$

và

$$y = \begin{bmatrix} 1 & 4 & 0 \\ 0 & 1 & -1 \end{bmatrix}$$

Tính biểu thức sau và kiểm tra kết quả trong R:

a) $2 * x$

b) $x * x$

c) $x \% * \% x$

d) $x \% * \% y$

e) $t(y)$

f) $solve(x)$

Bài 8 (sử dụng Bài 7) Với x và y như trên, tính tác động của các phép toán subscript sau và kiểm tra trong R.

a) $x[1,]$

b) $x[2,]$

c) $x[, 2]$

d) $y[1, 2]$

e) $y[, 2 : 3]$

Bài 9

1. Attach dataset *quakes* và tạo một tóm tắt thống kê về các biến *depth* và *mag*.
2. Attach dataset *mtcars* và tìm trọng lượng trung bình và mức tiêu thụ nhiên liệu trung bình cho các xe trong dataset(gõ `help(mtcars)`) để mô tả của các biến có sẵn.