

R CƠ BẢN

- I. R - Overview
- II. R - Get started
- III. R - Basic Syntax
- IV. R Comments
- V. Variables
- VI. Data types
- VII. User Input
- VIII. R Operators
- IX. R Decision Making
- X. R Loops
- XI. R Functions
- XII. Strings
- XIII. Vectors
- XIV. Lists
- XV. Matrices
- XVI. Arrays
- XVII. Factors
- XVIII. DataFrames
- XIX. R - Packages
- XX. File I/O

R ĐỒ HỌA

- XXI. R Plot & R Scatter Plot
- XXII. R Line
- XXIII. R Line & Point
- XXIV. Pie Charts
- XXV. Bar Charts

R Phân tích thống kê

- XXVI. Dataset
- XXVII. Max and Min
- XXVIII. Mean, Median and Mode
- XXIX. Permutations and Combinations
- XXX. Linear Regression
- XXXI. Multiple Regression
- XXXII. Logistic Regression
- XXXIII. Decision Tree
- XXXIV. Random Forest
- XXXV. Time Series Analysis
- XXXVI. Survival Analysis

I. Giới thiệu về R

R là một phần mềm sử dụng cho phân tích thống kê và đồ thị. Thật ra về bản chất, R là ngôn ngữ máy tính đa năng, có thể sử dụng cho nhiều mục tiêu khác nhau, từ tính toán đơn giản, toán học giả trí, tính toán ma trận, đến các phân tích thống kê phức tạp. R đang nổi lên như là một ngôn ngữ mang nhiều tiềm năng và ngày càng mở rộng với hàng ngàn gói (packages) cung cấp cho nhiều ứng dụng.

II. Bắt đầu với R

Để làm việc với ngôn ngữ R, trước tiên bạn cần cài đặt R trong môi trường của bạn. Sau đây là cách cài đặt R trong hệ điều hành Windows, MacOS và Linux:

- *Với Windows:*

Bước 1: Truy cập trang web theo đường dẫn sau: <https://cran.r-project.org/bin/windows/base/>

Bước 2: Chọn "Download R-4.2.2 for Windows" (Phiên bản 4.2.2 đang là phiên bản mới nhất trong thời điểm này)

Bước 3: Nhấp vào tab sẽ tải xuống trình cài đặt R. Nhấp đúp chuột vào trình cài đặt để khởi chạy nó.

Bước 4: Sau khi cài đặt thành công, biểu tượng R sẽ xuất hiện trên màn hình desktop và bạn có thể bắt đầu sử dụng R.

- *Với MacOS:*

Bước 1: Tải tệp .pkg theo đường dẫn sau: <https://cran.r-project.org/bin/macosx/> (Đây sẽ là phiên bản mới nhất hiện hành)

Bước 2: Mở tệp vừa được tải xuống và cài đặt R.

- *Với Linux:*

Chạy các câu lệnh sau ở giao diện dòng lệnh:

- Debian:
apt-get update
apt-get install r-base r-base-dev
- Fedora/Redhat:
\$ sudo dnf install R
- Ubuntu:
sudo apt install --no-install-recommends r-base

Bên cạnh đó, để thuận tiện hơn khi làm việc với R, bạn có thể cài đặt *RStudio*. Đây là một IDE vô cùng hữu ích giúp bạn làm việc dễ dàng hơn với R.

Bước 1: Truy cập trang chủ RStudio theo đường dẫn sau: <https://posit.co/>. Sau đó, chọn "DOWNLOAD RSTUDIO".

Bước 2: Cài đặt gói "RStudio Desktop" bản free.

Bước 3: Cài đặt R. Nếu bạn đã cài đặt R rồi thì chuyển sang bước 4.

Bước 4: Lựa chọn phiên bản RStudio phù hợp với môi trường của bạn để tải về.

Bước 5: Mở file vừa cài đặt lên và cài đặt RStudio.

Bài tập

1. Sau khi đã cài đặt thành công các môi trường cần thiết, tạo một đoạn mã R đầu tiên in ra màn hình "Hello World!"

III. Cú pháp cơ bản

Sau khi đã cài đặt môi trường cho R thành công, bạn có thể làm việc với R ở giao diện dòng lệnh hay kể cả là phiên bản R mới cài đặt. Khi nhấp đúp chuột vào phiên bản R mới cài đặt, nó sẽ hiện lên giao diện RGui để giúp bạn làm việc với R. Bên cạnh đó, bạn có thể dùng RStudio hoặc trên các trang [web](#) trực tuyến.

Sau khi chạy trình biên dịch R, bạn sẽ nhận được lời nhắc > để nhập câu lệnh R.

Ví dụ:

```
> # Create a String.  
> my_str = 'Hello World!'  
> print(my_str)  
[1] "Hello World!"
```

Đây là đoạn mã cơ bản khởi tạo một chuỗi ký tự tên "my_str" với giá trị là "Hello World!" và in nó ra màn hình.

Thông thường, bạn có thể viết các câu lệnh R trong một tệp văn bản, sau đó bạn có thể thực thi những câu lệnh đó ở giao diện dòng lệnh với sự trợ giúp của trình thông dịch R được gọi là Rscript.

Ví dụ:

```
# Create a String.  
my_str = 'Hello World!'  
print(my_str)
```

Lưu những câu lệnh trên vào một tệp có tên là my_file.R. Sau đó, thực thi câu lệnh sau ở giao diện dòng lệnh:

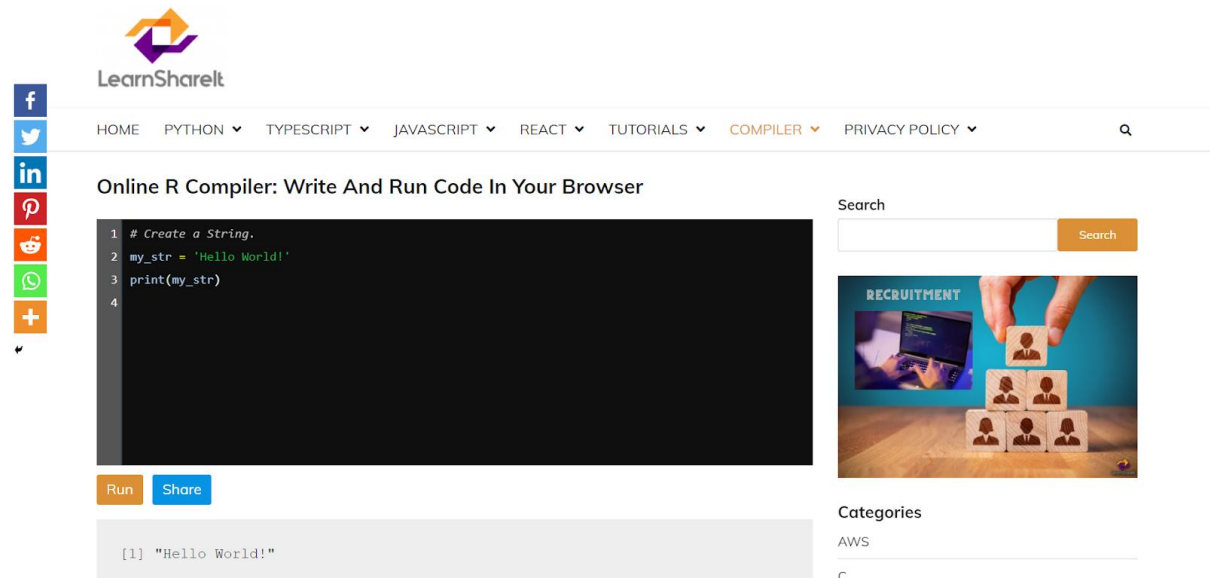
```
$ Rscript my_file.R
```

Kết quả thu được:

```
[1] "Hello World!"
```

Ngoài ra, bạn có thể sao chép đoạn mã và chạy nó ở trên các trang web trực tuyến và kết quả vẫn sẽ không thay đổi.

Ví dụ:



IV. R Comments

Comment trong R là các dòng văn bản dùng để hỗ trợ cho các đoạn mã của bạn. Nó được ứng dụng nhiều nhất trong việc giải thích các đoạn mã, ý tưởng của bạn và được trình thông dịch bỏ qua. Một câu comment đơn được bắt đầu với dấu #.

Ví dụ:

```
# Get started with R.  
print('Hello World!')
```

R không hỗ trợ comment nhiều dòng, vì vậy bạn phải kết hợp nhiều comment đơn để tạo ra một comment nhiều dòng.

Ví dụ:

```
# Get started with R.  
# My first program with R.
```

Nhưng bạn có thể thực hiện một thủ thuật nhỏ sau để comment nhiều dòng trên R. Comment được đưa vào dấu nháy kép.

Ví dụ:

```
"Get started with R.
```

```
My first program with R."
```

Thực chất, bản chất của việc này là tạo ra một chuỗi nhưng vì không gán tên biến hay sử dụng cho mục đích khác thì nó có thể được coi là một multi-line comment trong R.

Bài tập

1. Chú thích trong R bắt đầu bởi dấu gì?
 - A. `\\`
 - B. `//`
 - C. `#`
 - D. `@`
2. R có hỗ trợ chú thích nhiều dòng không?
 - A. Có
 - B. Không
3. Thêm chú thích "Get started with R" vào đầu đoạn code:

```
print('Hello World!')
```
4. Thêm chú thích:

```
Get started with R  
My first program
```

vào đầu đoạn code:

```
print('Hello World!')
```

V. R Variables

Biến là một vùng nhớ trong máy tính dùng để lưu trữ giá trị dữ liệu có thể thay đổi được trong chương trình. Thông thường, vùng nhớ này được quản lý bởi một tên (gọi là tên biến) và thay vì truy xuất đến vùng nhớ, ta truy xuất đến tên biến. Một tên biến hợp lệ bao gồm các ký tự chữ cái, số, dấu chấm và dấu gạch dưới.

Các lưu ý khi đặt tên biến trong R:

- Không cho phép dùng ký tự đặc biệt như: `!`, `@`, `#`, `$`, ...
- Không được bắt đầu bằng một chữ số.
- Được bắt đầu bằng dấu chấm nhưng theo sau đó không được là một chữ số.
- Không được bắt đầu bằng dấu gạch dưới.

Bạn có thể tạo ra một biến trong R bằng toán tử gán trong R. Gán tên biến trước dấu =, dấu <- hoặc <<- và giá trị ở sau đó. Cách còn lại là đặt giá trị trước dấu -> hoặc ->> và tên biến ở sau đó.

Ví dụ:

```
# Create a variable named 'my_var' with the equal operator.
```

```
my_var = 1
```

```
# Create a variable named 'my.var' with the leftward operator.
```

```
my.var <- 0.5
```

```
# Create a variable named 'my_var.2' with the rightward operator.
```

```
TRUE -> my_var.2
```

Ngoài ra, bạn có thể xuất toàn bộ các biến trong chương trình bằng hàm ls().

Cú pháp: ls()

Ví dụ:

```
print(ls())
```

Bên cạnh đó, bạn có thể xóa một biến trong chương trình bằng hàm rm().

Cú pháp: rm(tên_biến)

Ví dụ:

```
# Delete the variable named 'my_var'.
```

```
rm(my_var)
```

Bài tập

1. Tạo tên biến 'name' và gán giá trị bằng 'Ronaldo' bằng 5 toán tử:

- =
- <-
- <<-
- ->>
- ->

2. Xóa toán tử 'name' vừa tạo ra khỏi chương trình.

VI. Các kiểu dữ liệu cơ bản trong R

R có rất đa dạng các kiểu dữ liệu nhưng trong đó có 5 kiểu dữ liệu cơ bản mà bạn sẽ thường xuyên tiếp xúc và xuất hiện khi làm việc với R:

- Numeric
- Integer
- Complex
- Character

- Logical

Bạn có thể dùng hàm `class()` để kiểm tra kiểu dữ liệu của một biến.

Data type	Description	Example
Numeric	Là các số như số thập phân, số nguyên (Ví dụ: 0.5, 10, 25, 5.5)	<code>my_var <- 0.5</code> <code>class(my_var)</code> Output: [1] "numeric"
Integer	Là số nguyên nhưng có kí tự 'L' theo sau để phân biệt với kiểu dữ liệu numeric (Ví dụ: -1L, 1L, 5L)	<code>my_var <- -1L</code> <code>class(my_var)</code> Output: [1] "integer"
Complex	Là số phức có dạng: $a + bi$. Trong đó: a,b là số thực i là phần ảo (Ví dụ: -1 +2i, 5 - 6i)	<code>my_var <- -1 + 3i</code> <code>class(my_var)</code> Output: [1] "complex"
Character	Là chuỗi ký tự được khai báo bên trong dấu nháy đơn hoặc dấu nháy kép (Ví dụ: 'FALSE', "10.56", '1+3i')	<code>my_var <- '1+3i'</code> <code>class(my_var)</code> Output: [1] "character"
Logical	Có hai giá trị: TRUE và FALSE	<code>my_var <- TRUE</code> <code>class(my_var)</code> Output: [1] "logical"

Bài tập

1. Tên biến nào sau đây là hợp lệ:

- 2var
- .var
- _var
- .2var
- var!

- #var

Chạy môi trường R và kiểm tra lại kết quả

2. Kiểu dữ liệu của các giá trị sau:

- 0
- 0.5
- '0'
- 'TRUE'
- TRUE
- 1L
- $3 + 2i$

Dùng hàm class() để kiểm tra lại kết quả.

VII. User Input

Hàm readline() trong R dùng để nhập giá trị từ bàn phím và gán nó trong 1 biến.

Cú pháp:

<tên biến> <- readline()

Giá trị của biến sau khi nhập bằng hàm readline() là một chuỗi ký tự. Sau khi nhập xong, bạn có thể chuyển giá trị đó thành kiểu dữ liệu khác bằng các hàm như sau:

- as.numeric() để chuyển sang kiểu dữ liệu numeric.
- as.integer() để chuyển sang kiểu dữ liệu integer.
- as.logical() để chuyển sang kiểu dữ liệu logical.
- as.complex() để chuyển sang kiểu dữ liệu complex.

Bài tập

1. Nhập các giá trị: 0.5, 1L, "R", FALSE, $1 + 3i$ và chuyển chúng về đúng kiểu dữ liệu.

VIII. R Operators

Có 4 loại toán tử thường dùng trong R:

- Toán tử số học
- Toán tử logic
- Toán tử gán
- Toán tử khác

Toán tử số học:

Toán tử số học được dùng để làm việc với các số như nhân, chia, cộng trừ, lũy thừa, ...

Operator	Description	Example
+	Cộng 2 số	47 + 22 Output: [1] 69
-	Trừ 2 số	47 - 22 Output: [1] 25
*	Nhân 2 số	47 * 22 Output: [1] 1034
/	Chia 2 số	47 / 22 Output: [1] 2.136364
^ hoặc **	Lũy thừa	2 ^ 3 Output: [1] 8
%%	Phép chia dư	47 %% 22 Output: [1] 3
%/%	Phép chia lấy phần nguyên	47 %/% 22 Output: [1] 2

Toán tử logic:

Toán tử logic trả về TRUE hoặc FALSE miêu tả sự chính xác của một mệnh đề.

Operator	Description	Example
<	Bé hơn	1 < 2

		Output: [1] TRUE
<=	Bé hơn hoặc bằng	3 <= 2 Output: [1] FALSE
>	Lớn hơn	3 > 2 Output: [1] TRUE
>=	Lớn hơn hoặc bằng	2 <= 2 Output: [1] TRUE
==	Bằng	1 == 2 Output: [1] FALSE
!=	Không bằng	1 != 2 Output: [1] TRUE
!	Toán tử NOT operator: đảo lại giá trị logic của một mệnh đề	!TRUE Output: [1] FALSE
	Toán tử OR. TRUE khi một trong 2 vế là TRUE. FALSE khi và chỉ khi cả 2 vế đều FALSE.	FALSE TRUE Output: [1] TRUE
&	Toán tử AND. TRUE khi cả 2 vế là TRUE. FALSE khi một trong 2 vế là FALSE	FALSE & TRUE Output: [1] FALSE

Toán tử gán:

Toán tử gán dùng để gán giá trị cho một biến.

Operator	Description	Example
= hoặc <- hoặc <<-	Tên biến ở bên trái và giá trị ở bên phải.	my_var = 10 my_var <- 100 my_var <<- 1000
-> hoặc ->>	Giá trị ở bên trái và tên biến ở bên phải	10 -> my_var 100 ->> my_var

Toán tử khác:

Operator	Description	Example
:	Tạo một vector gồm các phần tử liên tiếp nhau từ về trái cho đến về phải	vec <- 1:9 vec Output: [1] 1 2 3 4 5 6 7 8 9
%in%	Logical. Kiểm tra xem phần tử có thuộc vector hay không.	vec <- 1:9 print(10 %in% vec) print(5 %in% vec) Output: [1] FALSE [1] TRUE
%*%	Nhân ma trận.	mat <- matrix(1:6, nrow = 3, ncol = 2) new_mat = mat %*% t(mat) new_mat Output: [,1] [,2] [,3] [1,] 17 22 27 [2,] 22 29 36 [3,] 27 36 45

Bài tập

1. Tính 1275933 chia dư cho 4728
2. Tính 1275933 chia lấy phần nguyên cho 4728
3. Chuyển đổi 28,7 độ F sang độ C theo công thức sau
 $^{\circ}\text{C} = (^{\circ}\text{F} - 32)/18$
4. Tính thể tích của hình tròn có bán kính 2cm theo công thức sau: $(4/3)\pi R^3$
5. Kiểm tra xem 5 và 6 có nằm trong tập giá trị $\{1, 5, 7, 9, 2\}$ không?
6. Cho ma trận:
1 5 6
8 9 2
5 7 9
4 3 6

Tính tích ma trận đã cho và ma trận chuyển vị của nó.
7. Tính: $e^x + x^2$ tại $x = 0.746$

IX. Cấu trúc rẽ nhánh

Biểu thức logic

Biểu thức logic là một biểu thức luôn cho kết quả ĐÚNG (TRUE) hoặc SAI (FALSE).

Example:

Hôm nay là chủ nhật thì ngày mai là thứ 2: TRUE

Mặt trời mọc ở đằng Tây: FALSE

$5 < 10$: TRUE

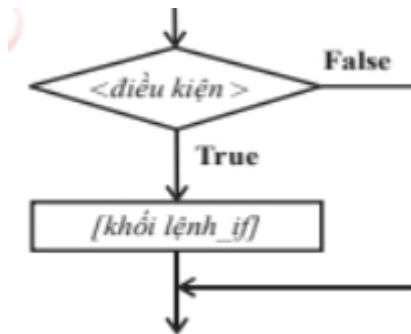
Ngày mai trời sẽ đổ mưa: Không phải là một biểu thức logic

Để chuyển hóa các điều kiện (biểu thức logic) từ thực tế về dạng các đoạn mã, bạn cần dùng các toán tử logic hoặc các hàm, câu lệnh mà nó trả về giá trị logic (TRUE hoặc FALSE)

If - else

Từ khóa if

Câu lệnh if dùng để thực hiện một khối lệnh nếu điều kiện (biểu thức logic) là TRUE.



Lưu đồ thuật toán câu lệnh if

Cú pháp:

```
if(<điều kiện>){  
    [khối lệnh]  
}
```

Trong đó:

- <điều kiện>: Là một biểu thức logic luôn trả về một trong hai giá trị TRUE hoặc FALSE.
- [khối lệnh]: Bao gồm một hoặc nhiều lệnh sẽ được thực hiện nếu điều kiện của câu lệnh if trả về TRUE.

Example:

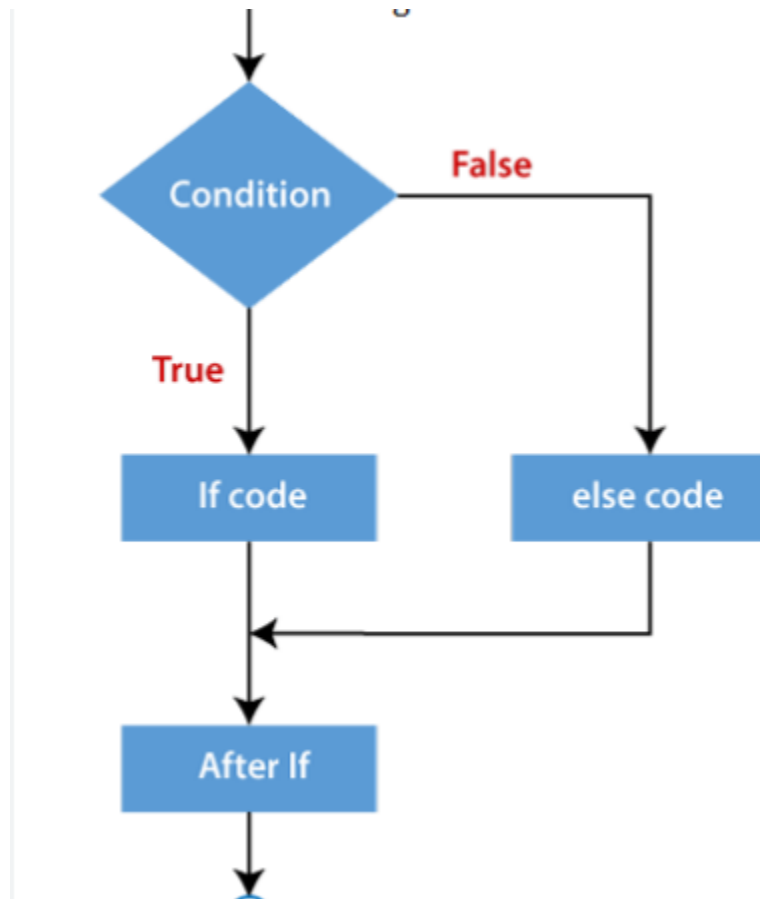
```
today <- 'Monday'  
if (today == 'Monday'){  
    print('Today is Monday')  
}
```

Output:

```
[1] "Today is Monday"
```

Từ khóa else

Khác với if, câu lệnh else dùng để thực hiện một hoặc nhiều câu lệnh khi một điều kiện (biểu thức logic) là FALSE.



Lưu đồ thuật toán

Cú pháp:

```
if (<điều kiện>){  
    [khối lệnh if]  
} else {  
    [khối lệnh else]  
}
```

Example:

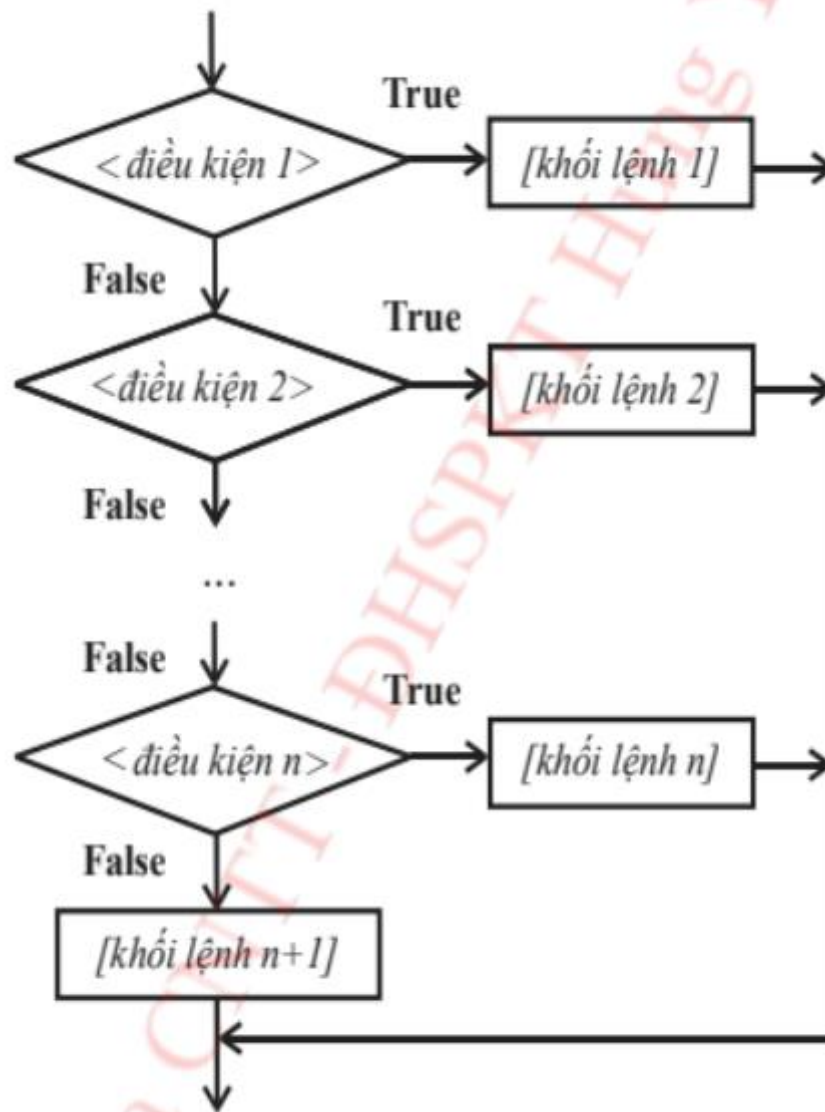
```
today <- 'Friday'  
if (today == 'Monday'){  
    print('Today is Monday')  
} else {  
    print('Today is not Monday')  
}
```

Output:

```
[1] "Today is not Monday"
```

Else if

Câu lệnh else if dùng để kiểm tra điều kiện tiếp theo nếu điều kiện trước đó là SAI. Nếu ĐÚNG thì khối lệnh trong câu lệnh else if sẽ được thực hiện.



Lưu đồ thuật toán

Cú pháp:

```
if (<điều kiện 1>){  
    [khối lệnh 1]  
} else if(<điều kiện 2>){  
    [khối lệnh 2]  
} else if(<điều kiện 3>){  
    [khối lệnh 3]  
}  
...  
else if(<điều kiện n>){
```

```
    [khối lệnh n]
  } else {
    [khối lệnh else]
  }
```

Câu lệnh else có thể có hoặc không.

Example:

```
today <- 'Friday'
if (today == 'Monday'){
  print('Today is Monday')
} else if (today == 'Friday') {
  print('Today is Friday')
}
```

Output:

```
[1] "Today is Friday"
```

Switch

Trong R, câu lệnh rẽ nhánh switch được thực hiện bởi hàm switch().

Cú pháp:

```
switch(expression, case1, case2, ...)
```

Hàm switch đánh giá một biểu thức và chọn một trong các trường hợp tiếp theo thỏa mãn. Nếu không có trường hợp nào được chọn, nó sẽ trả về NULL.

Example:

```
print(switch(1, 'R', 'Python', 'Java'))
print(switch(0, 'R', 'Python', 'Java'))
```

Output:

```
[1] "R"
```

```
NULL
```

Bài tập

1. Nhập ba số từ bàn phím và tìm giá trị nhỏ nhất.
2. Nhập điểm của một sinh viên từ bàn phím và xếp loại biết:
 - < 5.0: Yếu
 - < 6.5: TB
 - < 8: Khá
 - < 9: Giỏi
 - >= 9: Xuất sắc

3. Giải phương trình bậc 2: $ax^2 + bx + c = 0$ với a, b, c được nhập từ bàn phím.
4. Viết chương trình cho trò chơi kinh điển: kéo búa bao.
5. Lập menu cho dịch vụ sau với dữ liệu là 1 số được nhập từ bàn phím:
 - 1: Tính tổng 2 số
 - 2: Tính hiệu 2 số
 - 3: Tính tích 2 số
 - 4: Tính thương 2 số
 - 0: Quit

X. Cấu trúc lặp

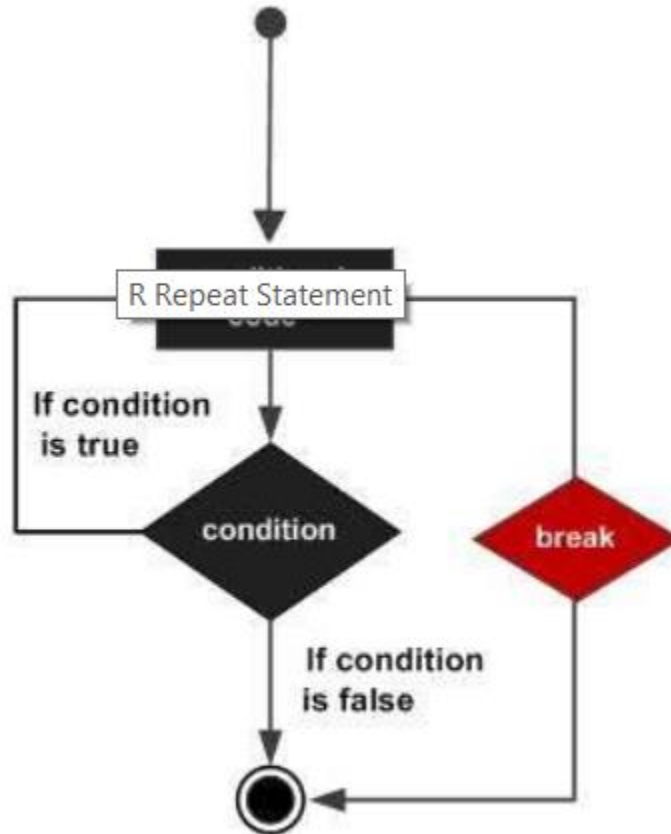
Cấu trúc lặp dùng để thực hiện các khối lệnh lặp đi lặp lại cho đến khi gặp điều kiện mong muốn. Trường hợp không gặp được điều kiện mong muốn, các khối lệnh sẽ lặp vô hạn cho đến khi tràn bộ nhớ. Trường hợp này rất tệ khi thực hiện các khối lệnh nên bạn cần phải phòng tránh nó.

Repeat Loop

Repeat loop lặp cho đến khi gặp câu lệnh break trong khối block.

Cú pháp:

```
repeat{  
    [khối lệnh]  
    if(<điều kiện>){  
        break  
    }  
}
```



Lưu đồ thuật toán

Example:

```
count <- 1
```

```
repeat{
```

```
  # Print "Hello World!" five times.
```

```
  print('Hello World!')
```

```
  if(count == 5) {
```

```
    break
```

```
  }
```

```
  count <- count + 1
```

```
}
```

Output:

```
[1] "Hello World!"
```

```
[1] "Hello World!"
```

```
[1] "Hello World!"
```

```
[1] "Hello World!"
```

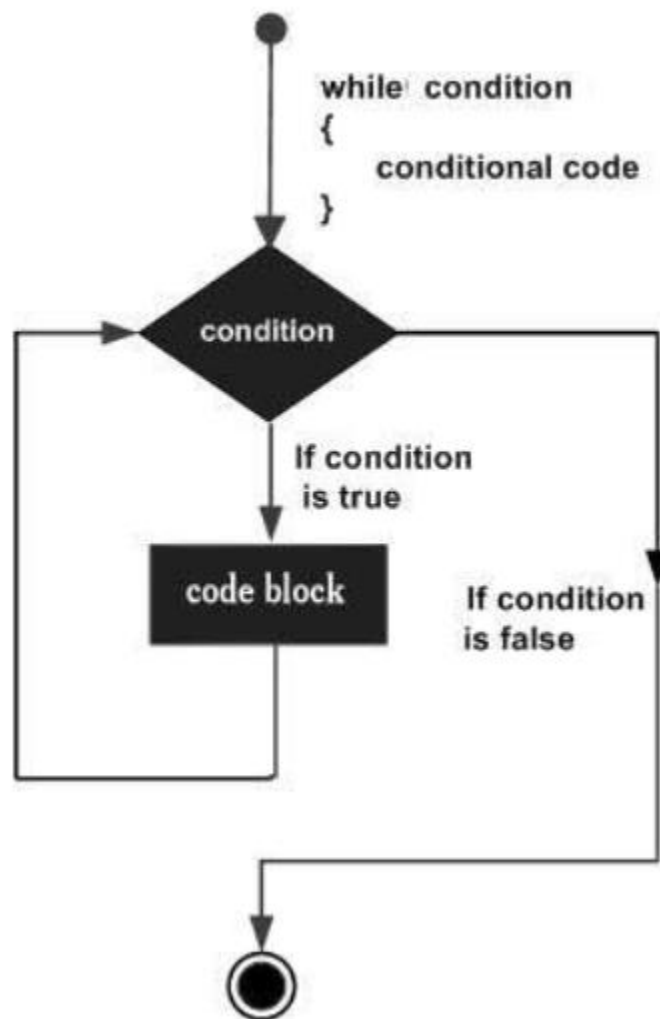
```
[1] "Hello World!"
```

While Loop

While loop lặp cho đến khi điều kiện đã cho không còn thỏa mãn(FALSE).

Cú pháp:

```
while(<điều kiện>){  
    [khởi lệnh]  
}
```



Lưu đồ thuật toán

Example:

```
count <- 1
```

```
# Print "Hello World!" five times.
```

```
while(count <= 5){  
    print('Hello World!')
```

```
count <- count + 1  
}
```

Output:

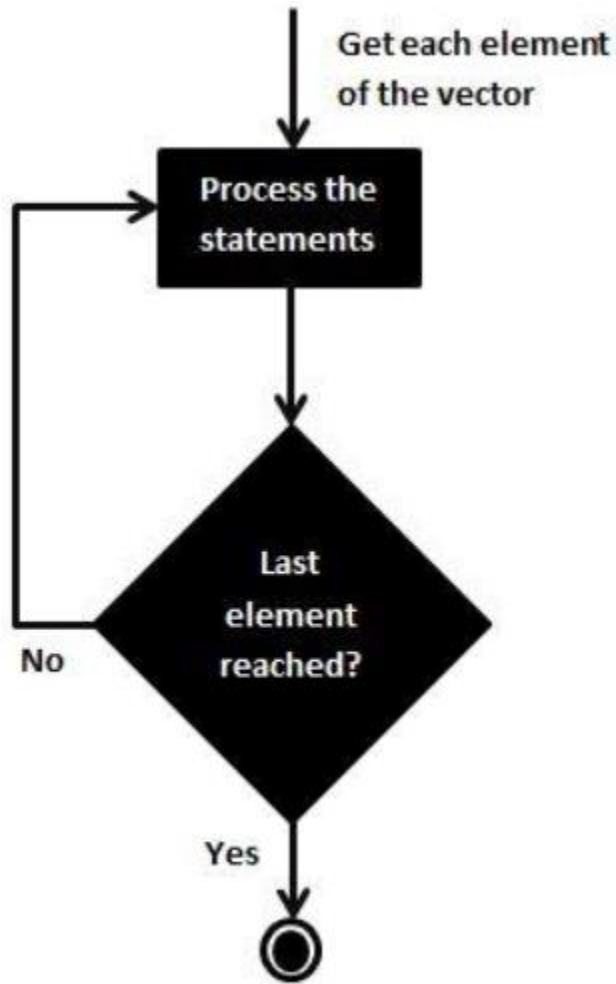
```
[1] "Hello World!"  
[1] "Hello World!"  
[1] "Hello World!"  
[1] "Hello World!"  
[1] "Hello World!"
```

For Loop

For loop dùng thực hiện các khối lệnh lặp đi lặp lại với số lần cụ thể. Nói cách khác, for loop sử dụng để lặp qua từng phần tử của một tập hợp.

Cú pháp:

```
for(<tên biến> in <tập hợp>){  
  [khối lệnh]  
}
```



Lưu đồ thuật toán

Example:

```
vec <- c(1, 5, 6, 9, 10)
# Print all elements in 'vec' vector.
for(element in vec){
  print(element)
}
```

Output:

```
[1] 1
[1] 5
[1] 6
[1] 9
[1] 10
```

Break

Câu lệnh break dùng để ngắt vòng lặp.

Example:

```
vec <- c(1, 5, 6, 9, 10)
```

```
for(element in vec){  
  # Terminate the loop if the element is 9.  
  if(element == 9){  
    break  
  }  
  print(element)  
}
```

Output:

```
[1] 1
```

```
[1] 5
```

```
[1] 6
```

Next

Giống như câu lệnh continue ở các ngôn ngữ lập trình khác, câu lệnh next trong R dùng để bỏ qua lần lặp hiện thời và ngay lập tức nhảy qua lần lặp khác.

Example:

```
vec <- c(1, 5, 6, 9, 10)
```

```
for(element in vec){  
  # Skip an iteration if the element is 9.  
  if(element == 9){  
    next  
  }  
  print(element)  
}
```

Output:

```
[1] 1
```

```
[1] 5
```

```
[1] 6
```

```
[1] 10
```

Bài tập

1. In ra màn hình "Hello World!" 50 lần.
2. Tính tổng: $S = 1+2+3+ \dots +1234$
3. Tính $15!$
4. Vẽ hình chữ nhật:

5. Vẽ hình thang cân ngược:

**

*

6. Nhập vào một số nguyên và hãy đếm xem số đó có bao nhiêu chữ số và tính tổng các chữ số.

XI. Functions

Hàm là một chương trình con gồm một tập hợp các câu lệnh giải quyết một công việc cụ thể từ các tham số, giá trị truyền vào hàm sau khi thực hiện các câu lệnh hàm kết thúc có hoặc không trả về một giá trị cụ thể.

R cung cấp cho ta rất nhiều hàm sẵn như là: `print()`, `sum()`, `class()`, ... Bên cạnh đó, chúng ta cũng có thể tạo ra các hàm riêng của chính mình.

Create A Function

Cú pháp:

```
<tên hàm> <- function([<dsts>]){  
  [khởi lệnh]  
  return (<giá trị trả về>)  
}
```

Câu lệnh `return` có thể có hoặc không. Lưu ý rằng, giá trị trả về của lệnh `return` được đặt trong dấu ngoặc đơn.

Truyền tham số cho hàm và gọi hàm

Để gọi một hàm, đơn giản chúng ta chỉ cần gọi đúng tên hàm cần gọi và có cặp dấu ngoặc đơn ở sau nó. Ở bên trong dấu ngoặc đơn có thể có hoặc không các tham số dựa vào hàm được định nghĩa lúc cài đặt.

Cú pháp:

`function_name(<parameters>)`

Trong đó:

- `function_name`: Tên hàm
- `<parameters>`: Danh sách các tham số.

Hàm không có tham số

Việc truyền tham số cho hàm là không bắt buộc, bạn có thể thực hiện một lời gọi hàm mà không cần tham số khi hàm được tạo không có các tham số với giá trị bắt buộc.

Example:

```
# Create a function without parameters.
```

```
my_func <- function(){  
  print('Hello World!')  
}
```

```
# Call this function.
```

```
my_func()
```

Output:

```
[1] "Hello World!"
```

Tham số với giá trị bắt buộc

Tham số bắt buộc là các tham số yêu cầu chúng ta phải truyền vào khi gọi hàm. Ví dụ như bạn tạo một hàm với 3 tham số bắt buộc, thì lúc gọi hàm cũng phải gán 3 giá trị cho 3 tham số đó.

Example:

```
# Calculate the multiplication of two numbers.
```

```
my_func <- function(x, y){  
  return (x*y)  
}
```

```
print(my_func(2, 3))
```

Output:

```
[1] 6
```

Tham số với giá trị mặc định

Tham số với giá trị mặc định là tham số không bắt buộc có trong lúc gọi hàm. Nếu không gán giá trị cho tham số ấy, thì nó sẽ tự động lấy giá trị mặc định trong lúc tạo hàm để thực hiện

Example:

```
my_func <- function(name = 'Ronaldo'){  
  paste('I am', name)  
}
```

Without parameters.

```
my_func()
```

```
my_func('Messi')
```

Output:

```
[1] I am Ronaldo
```

```
[1] I am Messi
```

Biến toàn cục và biến cục bộ (Global variables and local variables)

Biến toàn cục

Biến toàn cục được đặt ở bên ngoài hàm và chúng ta có thể sử dụng nó ở bên trong hàm mà không cần khai báo lại.

Example:

```
name <- 'Ronaldo'  
my_func <- function() {  
  paste("I am", name)  
}
```

```
my_func()
```

Output:

```
[1] I am Ronaldo
```

Biến cục bộ

Biến cục bộ là biến được đặt bên trong hàm, khi ở bên ngoài hàm, biến cục bộ sẽ không thể được sử dụng nữa.

Example:

```
my_func <- function() {  
  name <- 'Ronaldo'  
  paste("I am", name)  
}
```

```
my_func()
```

Output:

```
[1] I am Ronaldo
```

Thay đổi giá trị biến toàn cục

Trước tiên, chúng ta hãy xem qua ví dụ sau đây:

```
name <- 'Ronaldo'  
my_func <- function() {  
  name <- 'Messi'  
  print(name)  
}
```

```
my_func()  
print(name)
```

Output:

```
[1] Messi  
[1] Ronaldo
```

Giá trị của biến 'name' chỉ thay đổi ở trong hàm, còn ở ngoài thì nó vẫn không thay đổi, vì sao?

Khi sử dụng các toán tử gán bình thường như: =, <-, -> ở trong hàm thì giá trị của biến toàn cục sẽ không thay đổi. Để thay đổi giá trị của biến toàn cục ở trong hàm, bạn có thể sử dụng toán tử <<- hoặc ->>.

Example:

```
name <- 'Ronaldo'  
my_func <- function() {  
  name <<- 'Messi'  
}
```

```
my_func()  
print(name)
```

[1] Messi

Bài tập

1. Tạo hàm tính: $e^x + x^2$ với giá trị mặc định cho tham số x là 1.
2. Tạo hàm nhân 2 ma trận.
3. Tạo hàm vẽ hình vuông với tham số là cạnh hình vuông.
4. Tạo hàm vẽ hình vuông với tham số là chiều dài và chiều rộng của hình chữ nhật.
5. Tạo hàm tính $n!$ với tham số đầu vào là giá trị của n .
6. Tạo hàm xếp loại sinh viên dựa trên điểm là tham số giá trị đầu vào:
 - < 5.0 : Yếu
 - < 6.5 : TB
 - < 8 : Khá
 - < 9 : Giỏi
 - ≥ 9 : Xuất sắc

XII. Strings

Trong R, chuỗi (character) là tất cả các giá trị, dữ liệu được đặt bên trong dấu cặp dấu nháy đơn hoặc nháy kép.

Example:

'Hello World!'

"Hello World!"

Các kí tự đặc biệt trong chuỗi

Value	Description	Example
\\	\	> str <- '\\' > cat(str) \
\n	New line	> str <- 'Hello\nWorld!' > cat(str) Hello World!
\t	Tab	> str <- 'Hello\tWorld!'

		> cat(str) Hello World!
\b	Backspace	> str <- 'Hello\bWorld!' > cat(str) HellWorld!
\'	'	> str <- 'Hello\'World!' > cat(str) Hello'World!
\"	"	> str <- 'Hello"World!' > cat(str) Hello"World!

Các thao tác cơ bản với chuỗi:

nchar(): Độ dài chuỗi

Bạn có thể lấy độ dài của chuỗi bằng hàm nchar().

Cú pháp:

nchar(str)

Example:

```
> str <- 'Hello World!'
```

```
> nchar(str)
```

```
[1] 12
```

paste(): Nối hai chuỗi

Hàm paste() dùng để dấu 2 chuỗi lại với nhau.

Cú pháp:

paste(str1, str2)

Trong đó:

- str1: chuỗi thứ nhất
- str2: chuỗi thứ hai

Example:

```
> str1 <- 'Hello'
```

```
> str2 <- 'World!'
```

```
> paste(str1, str2)
```

```
[1] "Hello World!"
```

toupper() and tolower()

Hàm `toupper()` dùng để chuẩn hóa tất cả các kí tự của chuỗi về dạng chữ hoa.

Cú pháp:

`toupper(str)`

Example:

```
> str <- 'hEllo'
```

```
> toupper(str)
```

```
[1] "HELLO"
```

Hàm `tolower()` dùng để chuẩn hóa tất cả các kí tự của chuỗi về dạng chữ thường.

Cú pháp:

`tolower(str)`

Example:

```
> str <- 'HeLLO'
```

```
> tolower(str)
```

```
[1] "hello"
```

`substring()`

Hàm `substring()` trong R dùng để cắt chuỗi dựa trên vị trí được chỉ định.

Cú pháp:

`substring(str, start, end)`

Trong đó:

- `str`: Chuỗi ban đầu.
- `start`: Vị trí cắt bên trái.
- `end`: Vị trí cắt bên phải.

Example:

```
> # Slice the String from the first index to the seventh index.
```

```
> substring(str, 1, 7)
```

```
[1] "Hello W"
```

`strsplit()`

Hàm `strsplit()` trong R dùng để phân chia chuỗi dựa trên chuỗi phân tách được chỉ định.

Cú pháp cơ bản:

`strsplit(str, split)`

Trong đó:

- `str`: Chuỗi ban đầu.
- `split`: Chuỗi phân tách.

Example:

```
> str <- 'Hello World!'
>
> # Split the String by the whitespace.
> strsplit(str, ' ')[[1]]
[1] "Hello" "World!"
```

Bài tập

1. Cho 2 chuỗi như sau: str1 = "i ComE fROm", str2 = "VietNAM". Yêu cầu nối 2 chuỗi và chuẩn hóa kết quả về dạng chữ thường và chữ hoa.
2. Cắt 5 ký tự đầu tiên của chuỗi "i ComE fROm".
3. Phân tách chuỗi "The, weather, is, so, cold!" theo dấu phẩy.
4. Đếm số từ của chuỗi "The weather is so cold!".
5. Đảo ngược chuỗi "The weather is so cold!".

XIII. Vectors

Vector trong R dùng để lưu trữ nhiều giá trị cùng kiểu dữ liệu trong 1 biến duy nhất.

Create a vector

Để tạo một vector trong R, đơn giản nhất, bạn có thể dùng hàm c().

Example:

```
# Create a numeric vector.
```

```
vec <- c(56, 78, 12.34, 15)
```

```
vec
```

Output

```
[1] 56.00 78.00 12.34 15.00
```

Tuy R không cho phép lưu trữ giá trị mà có kiểu dữ liệu khác nhau. Nhưng nếu bạn khởi tạo một vector gồm các giá trị có kiểu dữ liệu khác nhau thì vẫn không gây ra lỗi. Nhưng R sẽ tự chuyển đổi các giá trị đó về một kiểu dữ liệu duy nhất.

Example:

```
# Tạo một vector có nhiều kiểu dữ liệu khác nhau.
```

```
vec <- c(56, 7.8, 1L, TRUE, FALSE)
```

```
vec
```

Output

```
[1] 56.0 7.8 1.0 1.0 0.0
```

Ngoài hàm `c()`, bạn cũng có thể tạo 1 vector trong R bằng rất nhiều cách khác. Diễn hình ở đây, chúng tôi sẽ giới thiệu qua cách tạo 1 vector bằng toán tử `:`.

Toán tử `:` dùng để tạo một vector mà các giá trị liên tiếp cách nhau đúng 1 đơn vị có miền giá trị trong đoạn trước và sau dấu `:`.

Example:

```
# Tạo một vector có giá trị liên tiếp cách nhau 1 đơn vị trong đoạn 1.1 đến 10.
```

```
vec <- 1.1:10
```

```
vec
```

Output

```
[1] 1.1 2.1 3.1 4.1 5.1 6.1 7.1 8.1 9.1
```

The length of a vector

Để lấy độ dài của một vector, bạn có thể sử dụng hàm `length()`.

Example:

```
vec <- c(5, 4, 6, 8, 7)
```

```
# Get the length of this vector.
```

```
length(vec)
```

Output

```
[1] 5
```

Sort a vector

Để sắp xếp một vector theo thứ tự từ điển, bạn có thể sử dụng hàm `sort()`.

Example:

```
vec <- c(5, 4, 6, 8, 7)
```

```
# Sort this vector.
```

```
sort(vec)
```

Output

```
[1] 4 5 6 7 8
```

Mặc định của hàm `sort()` là sắp xếp theo thứ tự từ điển, ngoài ra, bạn có thể tùy chỉnh nó.

Access and change items

Để lấy giá trị của các phần tử trong vector, bạn có thể dùng cặp dấu ngoặc vuông và đặt chỉ mục vị trí ở trong nó. Khác với hầu hết các ngôn ngữ khác, phần tử đầu tiên có vị trí là 1 và phần tử cuối cùng bằng độ dài của vector. Để thay đổi thì bạn chỉ cần kết hợp nó với các toán tử gán trong R.

Example:

```
vec <- c(5, 4, 6, 8, 7)
```

```
# Get the first element of this vector.
```

```
vec[1]
```

```
# Change the first element's value to 100.
```

```
vec[1] <- 100
```

```
vec
```

Output

```
[1] 5
```

```
[1] 100 4 6 8 7
```

Bài tập

1. Tạo một vector lưu trữ tập hợp: {1, 5, 3, 2, 9, 3, 6, 4, 1, -1}
2. Tìm độ dài của vector đó.
3. Thay đổi 3 phần tử đầu thành 1, 2, 3
4. Sắp xếp lại vector.
5. Tạo một vector lưu trữ tập hợp có giá trị từ 1 đến 1000, 2 phần tử liên tiếp nhau cách nhau đúng 1 đơn vị.

XIV. Lists

List trong R dùng để lưu trữ tập dữ liệu trong 1 biến duy nhất. Nhưng khác với vector, list cho phép lưu trữ nhiều kiểu dữ liệu khác nhau. Ngoài các kiểu dữ liệu cơ bản, list còn có thể lưu trữ cả vector, ma trận dataframe cũng như nhiều kiểu dữ liệu khác.

Create a list

List được tạo ra bằng hàm `list()` và dữ liệu của nó được đặt bên trong cặp dấu ngoặc đơn.

Example:


```
my_list <- list(0, '0', c(1, 7, 9), 'R', TRUE, 'FALSE', 1L)
my_list
```

Output:

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

```
[[2]]
[1] "0"
```

```
[[3]]
[1] 1 7 9
```

```
[[4]]
[1] "R"
```

```
[[5]]
[1] TRUE
```

```
[[6]]
[1] "FALSE"
```

```
[[7]]
[1] 1
```

Đặt tên các phần tử của list

Hàm `names()` trong R dùng để đặt tên các phần tử của list. Giá trị tên của các phần tử sẽ được đặt trong một vector.

Example:

```
my_list <- list(0, 'R', c(1, 7, 9), TRUE)
```

```
# Set names of elements in this list.
```

```
names(my_list) <- c('first', 'second', 'third', 'fourth')
my_list
```

Output:

```
$first
[1] 0
```

```
$second  
[1] "R"
```

```
$third  
[1] 1 7 9
```

```
$fourth  
[1] TRUE
```

Truy cập và thay đổi phần tử của list

Để truy cập các phần tử của list, chúng ta có thể dùng cặp dấu ngoặc vuông và bên trong nó là vị trí của các phần tử cần truy cập. Cách thứ 2 là dùng dấu \$ và theo sau đó là tên của phần tử đã được đặt.

Để thay đổi giá trị của các phần tử trong list chúng ta có thể dùng toán tử gán đã được học ở trước.

Example:

```
> my_list <- list(0, 'R', c(1, 7, 9), TRUE)  
> names(my_list) <- c('first', 'second', 'third', 'fourth')  
>  
> # Get the first element.  
> print(my_list[1])  
$first  
[1] 0
```

```
> print(my_list$first)  
[1] 0  
>  
> # Change the first element to 10.  
> my_list[1] <- 10  
> my_list  
$first  
[1] 10
```

```
$second  
[1] "R"
```

```
$third  
[1] 1 7 9
```

```
$fourth
```

```
[1] TRUE
```

Nối các danh sách

Chúng ta có thể nối các danh sách lại thành một danh sách bằng hàm `c()`.

Example:

```
list1 <- list(0, 'R', c(1, 7, 9), TRUE)
```

```
list2 <- list(FALSE, 10)
```

```
# Merge the first list and the second list.
```

```
my_list <- c(list1, list2)
```

```
my_list
```

Output:

```
[[1]]
```

```
[1] 0
```

```
[[2]]
```

```
[1] "R"
```

```
[[3]]
```

```
[1] 1 7 9
```

```
[[4]]
```

```
[1] TRUE
```

```
[[5]]
```

```
[1] FALSE
```

```
[[6]]
```

```
[1] 10
```

Chuyển đổi danh sách thành vector

Danh sách cũng có thể được chuyển đổi thành vector thông qua hàm `unlist()`. Với tính chất của hàm vector, nếu giá trị của các phần tử ban đầu không cùng kiểu dữ liệu thì lập tức kết quả trả về của vector sẽ được tự động chuyển thành các giá trị cùng kiểu dữ liệu.

Example:

```
> my_list <- list(0, 'R', TRUE)
```

```

> my_list
[[1]]
[1] 0

[[2]]
[1] "R"

[[3]]
[1] TRUE

>
> # Convert the list to a vector.
> vec <- unlist(my_list)
> vec
[1] "0"  "R"  "TRUE"

```

Bài tập

1. Tạo một danh sách tập hợp các phần tử sau: 1L, -99, '-10', 'Hello World!', TRUE, TRUE
2. Đặt tên cho các phần tử trong thứ tự chữ cái alphabet.
3. Thay đổi 3 phần tử đầu thành FALSE.
4. Chuyển danh sách đã tạo thành vector.

XV. Matrices

Ma trận là một cấu trúc dữ liệu lưu trữ các tập dữ liệu 2 chiều bao gồm các cột và các hàng.

Tạo ma trận

Hàm `matrix()` trong R được dùng để tạo một ma trận dựa trên tập dữ liệu cho trước, số cột và số hàng nhất định.

Cú pháp cơ bản:

```
matrix(data, nrow, ncol)
```

Trong đó:

- data: Tập dữ liệu cho trước, thường là một chiều ví dụ như vector hay list.
- nrow: Số hàng mong muốn.
- ncol: Số cột mong muốn.

Example:

```
> mat <- matrix(c(4, 7, 9, 8, 6, 2), nrow = 2, ncol = 3)
> mat
      [,1] [,2] [,3]
[1,]  4   9   6
[2,]  7   8   2
```

Kích thước ma trận

Bạn có thể lấy kích thước ma trận (hàng và cột) bằng hàm `dim()`. Kết quả trả về sẽ là một vector gồm 2 phần tử lần lượt là số hàng và số cột.

Example:

```
mat <- matrix(c(4, 7, 9, 8, 6, 2), nrow = 2, ncol = 3)
dim(mat)
```

Output:

```
[1] 2 3
```

Đặt tên hàng và cột

Để đặt tên cho các hàng và cột trong ma trận, bạn có thể sử dụng hàm `rownames()` và `colnames()`.

Cú pháp cơ bản:

```
rownames(mat) <- values
```

```
colnames(mat) <- values
```

Trong đó:

- `mat`: Tên ma trận.
- `values`: Tập hợp tên hàng.

Example:

```
mat <- matrix(c(4, 7, 9, 8, 6, 2), nrow = 2, ncol = 3)
rownames(mat) <- c('row1', 'row2')
colnames(mat) <- c('col1', 'col2', 'col3')
mat
```

Output:

```
col1 col2 col3
row1  4   9   6
```

```
row2 7 8 2
```

Truy cập phần tử trong ma trận

Để truy cập các phần tử trong ma trận, bạn có thể sử dụng cặp dấu ngoặc vuông.

Truy cập một phần tử trong ma trận

Để truy cập một phần tử trong ma trận, bạn cần kết hợp giữa cặp dấu ngoặc vuông và dấu phẩy. Bên trái dấu phẩy là chỉ số hàng, bên phải là chỉ số cột. Nếu không có dấu phẩy, mặc định chỉ số cột sẽ bằng một và chỉ số còn lại sẽ là chỉ số hàng.

Example:

```
# Create a matrix.
```

```
mat <- matrix(c(4, 7, 9, 8, 6, 2), nrow = 2, ncol = 3)
```

```
# Truy cập phần tử có chỉ số hàng bằng 2, chỉ số cột bằng 2.
```

```
mat[2, 2]
```

```
# Truy cập phần tử có chỉ số hàng bằng 2, chỉ số cột bằng 1 theo 2 cách.
```

```
mat[2, 1]
```

```
mat[2]
```

Output:

```
[1] 8
```

```
[1] 7
```

```
[1] 7
```

Truy cập một hàng hoặc cột

Để truy cập một hàng hoặc cột trong ma trận, bạn cũng kết hợp cặp dấu ngoặc vuông và dấu phẩy. Để truy cập một hàng, bên trái dấu phẩy sẽ là chỉ số hàng còn bên phải sẽ bỏ trống. Ngược lại, để truy cập một cột thì bên phải dấu phẩy sẽ bỏ trống.

Example:

```
# Create a matrix.
```

```
mat <- matrix(c(4, 7, 9, 8, 6, 2), nrow = 2, ncol = 3)
```

```
# Get the second row.
```

```
mat[2,]
```

```
# Get the second column.
```

```
mat[, 2]
```

Output:

```
[1] 7 8 2  
[1] 9 8
```

Truy cập nhiều phần tử, hàng hoặc cột trong ma trận.

Để truy cập nhiều phần tử, hàng hoặc cột trong ma trận, bạn có thể sử dụng theo 2 phương pháp ở trên nhưng bạn sẽ gán các vector chỉ số hàng hoặc cột thay vì một chỉ số.

Example:

```
# Create a matrix.
```

```
mat <- matrix(c(4, 7, 9, 8, 6, 2), nrow = 2, ncol = 3)
```

```
# Truy cập các phần tử có chỉ số hàng và cột lần lượt là (1, 2) và (2, 2).
```

```
mat[c(1, 2), c(2, 2)]
```

```
# Get the first and second rows.
```

```
mat[c(1, 2),]
```

```
# Get the first and second columns.
```

```
mat[, c(1, 2)]
```

Output:

```
[,1] [,2]  
[1,] 9 9  
[2,] 8 8
```

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

```
[,1] [,2]  
[1,] 4 9  
[2,] 7 8
```

Phép nhân ma trận

Để nhân 2 ma trận với nhau, bạn có thể sử dụng toán tử `%*%`. Nhưng lưu ý rằng, 2 ma trận đó phải cùng số cột.

Example:

```
# Create the first matrix.
```

```
mat1 <- matrix(c(4, 7, 9, 8, 6, 2), nrow = 2, ncol = 3)
```

```
mat1
```

```
# Create the second matrix.
```

```
mat2 <- matrix(c(1, 3, 5, 2, 6, 4), nrow = 3, ncol = 2)
```

```
mat2
```

```
# Multiply two matrices.
```

```
mat1 %*% mat2
```

Output:

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

```
[,1] [,2]  
[1,]  1  2  
[2,]  3  6  
[3,]  5  4
```

```
[,1] [,2]  
[1,] 61 86  
[2,] 41 70
```

Thêm hàng, cột, ma trận vào ma trận

Để thêm một hàng vào trong ma trận, bạn có thể dùng hàm `rbind()`. Lưu ý rằng, số cột của tập hợp phần tử được thêm (nếu là ma trận) hoặc kích thước của tập hợp (nếu là vector hoặc list) phải bằng số cột của ma trận hiện thời.

Cú pháp cơ bản:

```
rbind(mat, data)
```

Trong đó:

- `mat`: Tên ma trận.

- data: tập hợp phần tử được thêm.

Kết quả trả về

Ma trận mới bao gồm các phần tử được thêm.

Example:

```
# Create a matrix.
```

```
mat <- matrix(c(4, 7, 9, 8, 6, 2), nrow = 2, ncol = 3)
```

```
mat
```

```
new_mat <- rbind(mat, c(5, 5, 5))
```

```
new_mat
```

Output:

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

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

Để thêm một hàng vào trong ma trận, bạn có thể dùng hàm `cbind()`. Lưu ý rằng, nếu số phần tử của tập hợp được thêm không khớp với số hàng, R sẽ tự động thêm phần vào hàng bị khuyết dựa trên tập hợp được thêm.

Cú pháp cơ bản:

```
cbind(mat, data)
```

Trong đó:

- mat: Tên ma trận.
- data: tập hợp phần tử được thêm.

Kết quả trả về

Ma trận mới bao gồm các phần tử được thêm.

Example:

```
# Create a matrix.
```

```
mat <- matrix(c(4, 7, 9, 8, 6, 2), nrow = 2, ncol = 3)
```

```
cbind(mat, c(2, 2))
```

Output:

```
nd(mat, c(2, 2))
  [,1] [,2] [,3] [,4]
[1,]  4   9   6   2
[2,]  7   8   2   2
```

Ngoài thêm hàng, cột ra thì hàm `cbind()` và `rbind()` còn được dùng để hợp nhất 2 ma trận theo hàng hoặc cột.

Duyệt qua một ma trận

Để duyệt qua các phần tử của một ma trận, bạn có thể dùng hai vòng lặp `for` lồng nhau và truy cập phần tử dựa trên chỉ số hàng và cột.

Example:

```
mat <- matrix(c(4, 7, 9, 8, 6, 2), nrow = 2, ncol = 3)
```

```
for(row in 1:dim(mat)[1]){
  for(column in 1:dim(mat)[2]){
    print(mat[row, column])
  }
}
```

Output:

```
[1] 4
[1] 9
[1] 6
[1] 7
[1] 8
[1] 2
```

Bài tập

1. Tạo một ma trận 3 hàng 5 cột và các phần tử lần lượt từ 1 đến 15 theo mẫu sau:

```
1  4  7 10 13
2  5  8 11 14
```

- 3 6 9 12 15
2. In ra ma trận nghịch đảo của ma trận vừa tạo.
 3. Nhân ma trận ban đầu và ma trận nghịch đảo của nó.
 4. Nối ma trận sau đây vào ma trận của bài 1 theo hàng:

1	3	5	7	9
2	4	6	8	10
 5. Đổi các phần tử ở vị trí (1, 1), (2, 1), (3, 1), (2, 2) thành 100.

XVI. Arrays

Khác với vector, list và matrix, arrays trong R có thể coi là cha của 3 cấu trúc dữ liệu trên khi mà nó có thể linh hoạt số chiều (lớn hơn 1 và không có giới hạn). Nhưng nó thường được sử dụng với số chiều là 3, để lưu trữ các ma trận. Các trường hợp số chiều lớn hơn thì rất ít khi xuất hiện.

Tạo một array

Để tạo một array, bạn có thể sử dụng hàm `array()` trong R.

Cú pháp cơ bản:

`array(data, dim)`

Trong đó:

- data: Tập dữ liệu.
- dim: Vector kích thước của các chiều trong array.

Example:

```
arr <- array(1:18, c(3, 3, 2))
```

```
arr
```

Output:

```
,, 1
```

```

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

,, 2

```
      [,1] [,2] [,3]  
[1,]  10  13  16  
[2,]  11  14  17  
[3,]  12  15  18
```

Số lượng phần tử trong Array

Để tìm số lượng các phần tử trong mảng, bạn có thể dùng hàm `length()`.

Example:

```
arr <- array(1:18, c(3, 3, 2))  
length(arr)
```

Output:

```
[1] 18
```

Kích thước Array

Cũng giống như ma trận, để lấy kích thước của mảng, chúng ta cũng có thể dùng hàm `dim()`. Hàm `dim()` sẽ trả về một vector với kích thước là số chiều của mảng. Mỗi phần tử trong vector lần lượt là các kích thước tương ứng trong các chiều của mảng.

Example:

```
arr <- array(1:18, c(3, 3, 2))  
dim(arr)
```

Output:

```
[1] 3 3 2
```

Truy cập phần tử trong mảng

Cũng tương tự như ma trận, để truy cập và thay đổi phần tử trong mảng, chúng ta cần kết hợp cặp dấu ngoặc vuông và dấu phẩy. Cách truy cập một hay nhiều tập phần tử trong mảng hoàn toàn giống với ma trận. Chỉ khác số lượng dấu phẩy sẽ được tùy chỉnh dựa trên số chiều hiện có của ma trận.

Example:

```
> arr <- array(1:18, c(3, 3, 2))
```

```
> arr  
, , 1
```

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

```
, , 2
```

```
      [,1] [,2] [,3]  
[1,]  10  13  16  
[2,]  11  14  17  
[3,]  12  15  18
```

```
>
```

```
> # Truy cập phần tử ở hàng 1, cột 2, ma trận thứ 2.
```

```
> arr[1, 2, 2]
```

```
[1] 13
```

```
>
```

```
> # Truy cập các phần tử ở hàng 1, ma trận thứ nhất.
```

```
> arr[1, , 1]
```

```
[1] 1 4 7
```

```
>
```

```
> # Truy cập các phần tử ở cột 2, ma trận thứ 2.
```

```
> arr[, 2, 2]
```

```
[1] 13 14 15
```

```
>
```

```
> # Truy cập nhiều phần tử.
```

```
> arr[c(1, 2), c(2, 1), 2]
```

```
      [,1] [,2]  
[1,]  13  10  
[2,]  14  11
```

Duyệt qua một mảng

Để duyệt qua một mảng, bạn có thể kết hợp vòng lặp for và toán tử in.

Example:

```
arr <- array(1:18, c(3, 3, 2))
```

```
arr
```

```
for (element in arr){  
    print(element)  
}
```

Output:

```
, , 1
```

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

```
, , 2
```

```
    [,1] [,2] [,3]  
[1,] 10  13  16  
[2,] 11  14  17  
[3,] 12  15  18
```

```
[1] 1  
[1] 2  
[1] 3  
[1] 4  
[1] 5  
[1] 6  
[1] 7  
[1] 8  
[1] 9  
[1] 10  
[1] 11  
[1] 12  
[1] 13  
[1] 14  
[1] 15  
[1] 16  
[1] 17  
[1] 18
```

Bài tập

1. Tạo một mảng chứa 2 ma trận có kích thước 3x4. Các phần tử của 2 ma trận lần lượt từ 1 đến 24.
2. Tính kích thước của mảng vừa tạo.
3. Đổi phần tử ở hàng 2 cột 4 trong ma trận thứ nhất thành 1000.
4. Đổi phần tử ở cột 1 và cột 2 trong ma trận thứ 2 thành 999.
5. Duyệt và in ra các phần tử của mảng sau khi đã được đổi.

XVII. Factors

Factors là cấu trúc dữ liệu dùng để phân loại và lưu trữ dữ liệu dựa trên các cấp. Phần tử của cấp trong factors là các phần tử khác nhau trong tập dữ liệu.

Tạo một factor

Để tạo một factor, bạn có thể dùng hàm `factor()`.

Cú pháp cơ bản:

`factor(data, levels)`

Trong đó:

- `data`: Tập dữ liệu.
- `levels`: Vector lưu trữ các phần tử khác nhau trong tập dữ liệu theo thứ hạng. Mặc định thì thứ hạng sẽ được sắp xếp theo thứ tự từ điển.

Example:

```
rank <- c('Gold', 'Silver', 'Bronze', 'Bronze', 'Gold', 'Silver', 'Silver')
```

```
# Tạo một factor với levels được tính theo thứ tự từ điển.
```

```
rank_factor1 <- factor(rank)
```

```
rank_factor1
```

```
# Tạo một factor với levels theo thứ tự cụ thể.
```

```
rank_factor2 <- factor(rank, levels = c('Gold', 'Silver', 'Bronze'))
```

```
rank_factor2
```

Output:

```
[1] Gold Silver Bronze Bronze Gold Silver Silver
Levels: Bronze Gold Silver
```

```
[1] Gold Silver Bronze Bronze Gold Silver Silver
Levels: Gold Silver Bronze
```

Kích thước của một factor

Cũng giống như các cấu trúc dữ liệu ở trước, bạn có thể lấy độ dài của một factor bằng hàm `length()`.

Example:

```
rank <- c('Gold', 'Silver', 'Bronze', 'Bronze', 'Gold', 'Silver', 'Silver')
rank_factor <- factor(rank, levels = c('Gold', 'Silver', 'Bronze'))
length(rank_factor)
```

Output:

```
[1] 7
```

Thay đổi thứ tự trong levels của factor

Bạn có thể thay đổi thứ tự trong levels của một factor sau khi factor đó đã được tạo bằng hàm `levels()`.

Cú pháp cơ bản:

```
levels(factor_name) <- values
```

Trong đó:

- `factor_name`: Tên factor.
- `values`: Vector lưu trữ các phần tử khác nhau trong tập dữ liệu theo thứ hạng.

Example:

```
rank <- c('Gold', 'Silver', 'Bronze', 'Bronze', 'Gold', 'Silver', 'Silver')
rank_factor <- factor(rank)
rank_factor
```

```
# Change levels of the factor.
```

```
levels(rank_factor) <- c('Gold', 'Silver', 'Bronze')
rank_factor
```

Output:


```
[1] Gold Silver Bronze Bronze Gold Silver Silver
Levels: Bronze Gold Silver
```

```
[1] Silver Bronze Gold Gold Silver Bronze Bronze
Levels: Gold Silver Bronze
```

Truy cập và thay đổi phần tử trong factor

Giống như vector và list, bạn có thể truy cập phần tử trong factor bằng cặp dấu ngoặc vuông.

Example:

```
rank <- c('Gold', 'Silver', 'Bronze', 'Bronze', 'Gold', 'Silver', 'Silver')
rank_factor <- factor(rank, levels = c('Gold', 'Silver', 'Bronze'))
```

```
# Get the third element of the factor.
```

```
rank_factor[3]
```

Output:

```
[1] Bronze
```

```
Levels: Gold Silver Bronze
```

Tuy nhiên, khi thay đổi giá trị phần tử trong factor, giá trị này phải nằm trong tập giá trị của levels, nếu không sẽ xảy ra lỗi.

Example:

```
rank <- c('Gold', 'Silver', 'Bronze', 'Bronze', 'Gold', 'Silver', 'Silver')
rank_factor <- factor(rank, levels = c('Gold', 'Silver', 'Bronze'))
```

```
# Change the value of the third element to 'Champion'.
```

```
rank_factor[3] <- 'Champion'
```

Output:

Warning message:

```
In `[<-factor`(*tmp*, 3, value = "Champion") :
  invalid factor level, NA generated
```

Để có thể thêm giá trị 'Champion' vào factor thì nó cũng phải xuất hiện trong levels của factor.

Example:

```
# Change levels of the factor.  
levels(rank_factor) <- c('Champion', 'Gold', 'Silver', 'Bronze')  
  
# Change the value of the third element to 'Champion'.  
rank_factor[3] <- 'Champion'  
rank_factor[3]
```

Output:
[1] Champion
Levels: Champion Gold Silver Bronze

Bài tập

1. Cho tập dữ liệu: {'Ronaldo', 'Ronaldo', 'Messi', 'Haland', 'Haland', 'Messi', 'Mbappe'}. Hãy tạo một factor dựa trên tập dữ liệu đó, levels được xếp theo thứ tự từ điển.
2. Thay đổi thứ tự của levels thành ('Ronaldo', 'Messi', 'Mbappe', 'Haland').
3. Thay đổi phần tử thứ nhất thành 'Mbappe'.
4. Thay đổi phần tử thứ nhất thành 'Benzema' (cần lưu ý trường hợp levels không chứa phần tử cần được thêm).

XVIII. Dataframes

Dataframe là cấu trúc dữ liệu có dạng table và nó có 2 chiều dạng hàng và cột giống như mảng 2 chiều hay ma trận. Nhưng dataframe có thể lưu trữ giá trị với nhiều kiểu dữ liệu khác nhau. Ngoài ra, số phần tử trong một cột hay hàng là đều bằng nhau, nếu nó bị khuyết, R sẽ tự động gán cho nó với giá trị NA.

Tạo một dataframe

Để tạo một dataframe, bạn có thể sử dụng hàm `data.frame()` trong R.

Cú pháp cơ bản:

```
df <- data.frame(data)
```

Trong đó:

- df: Tên dataframe.
- data: Tập giá trị của dataframe được tạo theo mẫu: `tên_cột = <tập giá trị>` hoặc cấu trúc dữ liệu có số chiều bằng 2.

Example1:

```
# Create a dataframe.
df <- data.frame(
  name = c("Ronaldo", "Messi", "Benzema", "Mbappe"),
  age = c(38, 35, 35, 24)
)
df
```

Output:

```
   name age
1 Ronaldo 38
2  Messi  35
3 Benzema 35
4 Mbappe  24
```

Example2:

```
# Create a dataframe from a matrix.
df <- data.frame(matrix(1:15, nrow = 3, ncol = 5))
df
```

Output:

```
  X1 X2 X3 X4 X5
1  1  4  7 10 13
2  2  5  8 11 14
3  3  6  9 12 15
```

Kích thước của dataframe

Cũng giống như các cấu trúc dữ liệu khác, bạn có thể tính kích thước của dataframe bằng hàm `dim()`. Giá trị trả về là một vector gồm 2 phần tử lần lượt là số hàng và cột của dataframe.

Example:

```
df <- data.frame(
  name = c("Ronaldo", "Messi", "Benzema", "Mbappe"),
  age = c(38, 35, 35, 24)
)
```

```
# Get the size of this dataframe.
```

```
dim(df)
```

Output:

[1] 4 2

Tóm lược thông tin của một dataframe

Bạn có thể tóm lược thông tin các cột của một dataframe bằng hàm `summary()`. Kết quả trả về của các cột sẽ theo format khác nhau tùy theo kiểu dữ liệu của các giá trị trong cột.

Example:

```
df <- data.frame(  
  name = c("Ronaldo", "Messi", "Benzema", "Mbappe"),  
  age = c(38, 35, 35, 24)  
)
```

Summarize this dataframe.

```
summary(df)
```

Output:

name	age
Length:4	Min. :24.00
Class :character	1st Qu.:32.25
Mode :character	Median :35.00
	Mean :33.00
	3rd Qu.:35.75
	Max. :38.00

Visualize data

Ngoài việc xem dữ liệu của một data dataframe theo dạng bảng, bạn có thể xem dữ liệu của dataframe một cách tổng quan hơn bằng hàm `str()`. Giá trị trả về bao gồm số hàng, cột, thông tin các cột bao gồm dữ liệu và kiểu dữ liệu của nó.

Example:

```
df <- data.frame(  
  name = c("Ronaldo", "Messi", "Benzema", "Mbappe"),  
  age = c(38, 35, 35, 24)  
)
```

Visualize this dataframe.

```
str(df)
```

Output:

```
'data.frame': 4 obs. of 2 variables:
```

```
$ name: chr "Ronaldo" "Messi" "Benzema" "Mbappe"
```

```
$ age : num 38 35 35 24
```

Truy cập phần tử của dataframe

Bạn có thể truy cập các cột của dataframe bằng 3 cách: cặp dấu ngoặc vuông [], cặp dấu ngoặc vuông lồng nhau [[]] và dấu \$.

Với cặp dấu ngoặc vuông, bên trong nó bạn có thể để tên cột, số thứ tự cột hoặc vector tên hoặc số thứ tự cột. Kết quả trả về sẽ là một dataframe chứa các cột bạn muốn truy cập.

Example:

```
df <- data.frame(  
  name = c("Ronaldo", "Messi", "Benzema", "Mbappe"),  
  age = c(38, 35, 35, 24)  
)
```

```
# Get the first column.
```

```
df[1]
```

```
# Get the second column.
```

```
df["age"]
```

Output:

```
  name  
1 Ronaldo  
2  Messi  
3 Benzema  
4 Mbappe
```

```
  age  
1 38  
2 35  
3 35  
4 24
```

Để chặt chẽ hơn, bạn có thể sử dụng dấu \$, nó chỉ cho phép bạn truy cập dữ liệu bằng tên cột. Giá trị trả về vẫn là một dataframe chứa các cột bạn muốn truy cập.

Example:

```
df <- data.frame(  
  name = c("Ronaldo", "Messi", "Benzema", "Mbappe"),  
  age = c(38, 35, 35, 24)  
)
```

Get the first column.

```
df$name
```

Output:

```
  name  
1 Ronaldo  
2  Messi  
3 Benzema  
4 Mbappe
```

Ngoài ra, bạn có thể dùng cặp dấu ngoặc vuông lồng nhau [[]] để truy cập một hoặc các giá trị trong cột. Khác với 2 phương thức ở trên, phương thức này sẽ trả về một vector trả về các giá trị cần tìm.

Example:

```
df <- data.frame(  
  name = c("Ronaldo", "Messi", "Benzema", "Mbappe"),  
  age = c(38, 35, 35, 24)  
)
```

Get the element at the first column and second row.

```
df[[c(1,2)]]
```

Get the first column.

```
df[[1]]
```

Output:

```
[1] "Messi"
```

```
[1] "Ronaldo" "Messi"  "Benzema" "Mbappe"
```

Thêm hàng, cột

Giống như ma trận, bạn có thể thêm hàng với hàm `cbind()` và cột với hàm `rbind()` trong dataframe. Bạn có thể xem lại cú pháp và cách vận hành của nó ở phần ma trận.

Example:

```
df <- data.frame(  
  name = c("Ronaldo", "Messi", "Benzema", "Mbappe"),  
  age = c(38, 35, 35, 24)  
)
```

Add a row to the current dataframe.

```
df1 <- rbind(df, c("Haland", 23))  
df1
```

Add a column to the current dataframe.

```
df2 <- cbind(df, titles = c(35, 42, 31, 15))  
df2
```

Output:

```
      name age  
1 Ronaldo 38  
2   Messi 35  
3 Benzema 35  
4 Mbappe 24  
5  Haland 23
```

```
      name age titles  
1 Ronaldo 38    35  
2   Messi 35    42  
3 Benzema 35    31  
4 Mbappe 24    15
```

Hợp nhất 2 dataframe

Ngoài thêm hàng và cột ra, hàm `cbind()` và `rbind()` trong R có thể giúp bạn hợp nhất 2 dataframe theo hàng hoặc cột. Lưu ý, với việc hợp nhất dataframe theo hàng, tên các cột trong 2 dataframe phải giống nhau.

Example:

```
# Create the first dataframe.
df1 <- data.frame(
  name = c("Ronaldo", "Messi", "Benzema", "Mbappe"),
  age = c(38, 35, 35, 24)
)
```

```
# Add this dataframe to the first dataframe horizontally.
df2 <- data.frame(name = "Haland", age = 23)
df_row <- rbind(df1, df2)
df_row
```

```
# Add this dataframe to the first dataframe vertically.
df3 <- data.frame(titles = c(35, 43, 31, 15))
df_column <- cbind(df1, df3)
df_column
```

Output:

```
      name age
1 Ronaldo 38
2   Messi 35
3 Benzema 35
4 Mbappe 24
5  Haland 23
```

```
      name age titles
1 Ronaldo 38    35
2   Messi 35    42
3 Benzema 35    31
4 Mbappe 24    15
```

Bài tập

1. Tạo ma trận có kích thước 3x4 với các phần tử lần lượt từ 1 đến 12.
2. Tạo một dataframe dựa trên ma trận đã tạo.
3. Thay đổi phần tử ở hàng 1 cột 2 thành 1000.
4. Thay đổi các phần tử ở cột 1 thành 1000.
5. Thêm 2 hàng có tất cả giá trị đều bằng 500 vào cuối hàng của dataframe đã tạo.

XIX. R Packages

Package là gói tài nguyên mà trong đó nó đã được tạo các hàm, dữ liệu. Mặc định, khi cài đặt R thì nó đã tích hợp rất nhiều package để người dùng có thể sử dụng. Tuy nhiên, trong nhiều trường hợp chúng ta cần các hàm hoặc dữ liệu được tạo sẵn cho mình sử dụng thì chúng ta cần phải cài đặt package đó vào trong môi trường R.

Để tích hợp package vào trong môi trường R, bạn có thể dùng hàm `install.packages()`.

Cú pháp cơ bản:

```
install.packages('package_name')
```

Trong đó:

- `package_name`: Tên package cần tích hợp.

Example:

```
# Install the 'dplyr' package.
```

```
install.packages('dplyr')
```

Để có thể sử dụng các hàm và dữ liệu trong package đã cài đặt, bạn có thể dùng hàm `library()`.

Cú pháp cơ bản:

```
library('package_name')
```

Trong đó:

- `package_name`: Tên package.

Example:

```
# Import the 'dplyr' package.
```

```
library('dplyr')
```

Ngoài ra, bạn có thể sử dụng hàm `library()` mà không gán bất kì thuộc tính nào để liệt kê các package đang hiện có trong môi trường của bạn.

Example:

```
# List all available packages in your environment.
```

```
library()
```

Bài tập

1. Cài đặt thư viện dplyr trong môi trường của bạn

XX. Vào ra file I/O

Chúng ta có thể đọc các file từ bên ngoài như là: csv, excel, xml, json, ... vào trong môi trường R bằng các hàm có sẵn được xây dựng trong R. Trong thực tế, chúng ta sẽ không tạo ra dữ liệu bằng các câu lệnh trong R mà thường đọc dữ liệu từ bên ngoài vào trong môi trường R vì đa số dữ liệu khá lớn. Vì ngôn ngữ R thường được dùng trong việc phân tích, thống kê dữ liệu nên sẽ dùng nhiều với file csv và excel. Vậy nên, chúng ta sẽ đi sâu hơn để tìm hiểu về nó.

Trước tiên, chúng ta hãy cùng tìm hiểu khái niệm cơ bản về đường dẫn tuyệt đối và tương đối trong file:

Đường dẫn từ gốc của cây thư mục, đi qua các thư mục trung gian, dẫn tới file hiện thời.

Đường dẫn tuyệt đối là đường dẫn từ gốc của cây thư mục, đi qua các thư mục trung gian dẫn tới file hiện thời.

Ví dụ: C:\Users\DELL\Documents\R_programming\R_file.csv

Đường dẫn tương đối là đường dẫn được tính từ thư mục hiện thời đến file hiện thời.

Ví dụ: \R_programming\R_file.xlsx

Khi làm việc với các hàm trong R để đọc dữ liệu từ file, bạn có thể lấy từ đường dẫn tuyệt đối hoặc để ngắn gọn hơn, bạn có thể dùng đường dẫn tương đối.

Khi dùng với đường dẫn tương đối, bạn cần phải biết thư mục hiện thời mình đang dùng là thư mục nào. Bạn có thể xem thư mục đó bằng hàm `getwd()`. Giá trị trả về của hàm `getwd()` là một chuỗi đường dẫn tuyệt đối của thư mục hiện thời.

Example:

```
getwd()
```

Output:

```
[1] "C:/Users/DELL/Documents"
```

Ngoài ra, bạn cũng có thể chuyển thư mục hiện thời bằng hàm `setwd()`.

Example:

```
# Set the current folder to 'R-programming'.
setwd('/R-programming')
print(getwd())
```

Output:

```
[1] "C:/Users/DELL/Documents/R-programming"
```

Đọc file csv

Để đọc file csv từ bên ngoài vào trong môi trường R, bạn có thể dùng hàm `read.csv()`. Giá trị trả về là một dataframe chứa tập dữ liệu trong file csv.

Cú pháp cơ bản:

```
read.csv(file_path)
```

Trong đó:

- `file_path`: Địa chỉ tuyệt đối hoặc tương đối của file.

Example:

Đọc file `input.csv` với tập dữ liệu sau:

```
name,age
1,Ronaldo,38
2,Messi,35
3,Benzema,35
4,Mbappe,24
```

Code sample:

```
# Read 'test.csv'.
df <- read.csv('test.csv')
print(df)
class(df)
```

Output:

```
      name age
1 Ronaldo 38
2   Messi 35
3 Benzema 35
4  Mbappe 24
```

```
[1] "data.frame"
```

Đọc file excel

Có rất nhiều package hỗ trợ bạn có thể đọc file excel từ bên ngoài vào trong môi trường R. Ở đây, tôi sẽ giới thiệu cho bạn cách đọc file excel với thư viện 'xlsx'.

Trước tiên, bạn cần cài đặt thư viện 'xlsx' bằng câu lệnh sau.

```
install.packages('xlsx')
```

Sau đó, bạn có thể sử dụng hàm `read.xlsx()` để đọc file excel vào môi trường của bạn. Giá trị trả về là một dataframe chứa dữ liệu trong file excel.

Cú pháp cơ bản:

```
read.xlsx(file_path, sheetIndex)
```

Trong đó:

- `file_path`: Đường dẫn tuyệt đối hoặc tương đối của file.
- `sheetIndex`: Số thứ tự sheet trong file excel.

Example:

Đọc file `test.xlsx` với dữ liệu như sau:

	A	B
1	name	age
2	Ronaldo	38
3	Messi	37
4	Benzema	35
5	Mbappe	24

Code sample:

```
# Import 'xlsx' package.
```

```
library('xlsx')
```

```
# Read the 'test.xlsx' file.
```

```
df <- read.xlsx('test.xlsx', sheetIndex = 1)
```

```
print(df)
```

```
class(df)
```

Output:

```
name age
```

```
1 Ronaldo 38
```

```
2 Messi 37
```

3 Benzema 35
4 Mbappe 24

[1] "data.frame"

Bài tập

1. Hãy tạo và đọc file trong 2 ví dụ ở trên vào môi trường của bạn.

XXI. Plot & Scatter Plot

Để vẽ một hay nhiều điểm trên một biểu đồ trong R, chúng ta thường sử dụng hàm `plot()`.

Cú pháp cơ bản:

`plot(x, y, type, xlim, ylim, main, xlab, ylab, axes, cex, col)`

Trong đó:

- `x`: Một hay một tập giá trị theo trục hoành.
- `y`: Một hay một tập giá trị theo trục tung.
- `type`: Mặc định sẽ là "p". Kiểu biểu đồ mong muốn với các giá trị như sau:
 - "p": dạng điểm.
 - "l": dạng đường.
 - "b": cả đường và điểm, điểm không đè lên đường.
 - "c": dạng đường nhưng ở vị trí điểm sẽ trống.
 - "o": cả đường và điểm nhưng điểm đè lên đường.
 - "s": dạng bậc thang.
 - "h": dạng các đoạn thẳng đứng có 1 đầu mút là: giá trị hoành độ và chiều cao là trị tuyệt đối của giá trị tung độ.
- `xlim`: Giới hạn giá trị của hoành độ.
- `ylim`: Giới hạn giá trị của tung độ.
- `main`: Tiêu đề của biểu đồ.
- `xlab`: Nhãn của trục hoành.
- `ylab`: Nhãn của trục tung.
- `axes`: TRUE hoặc FALSE. Nếu đúng, 2 trục sẽ được vẽ trên biểu đồ. Ngược lại thì không.
- `cex`: Kích thước của điểm, đường, ...
- `col`: Màu của điểm, đường, ...

Vẽ một điểm trên một biểu đồ

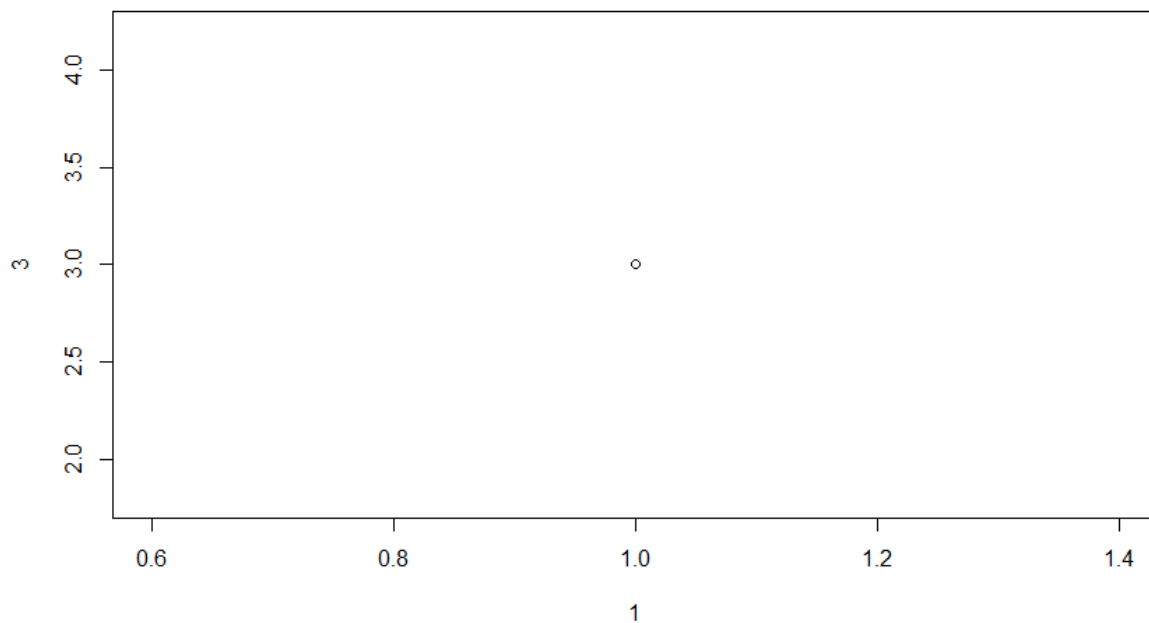
Để vẽ một điểm trên biểu đồ, đơn giản bạn chỉ cần gán lần lượt giá trị hoành độ và tung độ vào hàm plot().

Example:

```
# Plot point (1,3)
```

```
plot(1, 3)
```

Output:



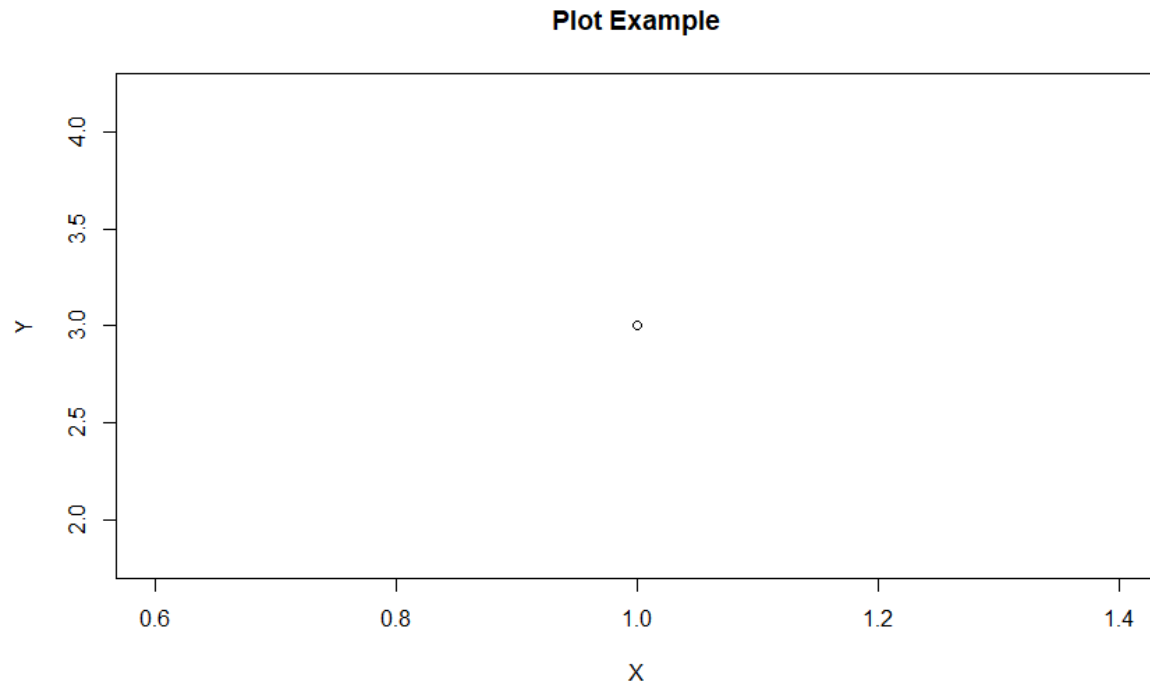
Đặt tên tiêu đề, trục hoành, trục tung

Để đặt tên cho tiêu đề, trục hoành, trục tung lần lượt bạn cần gán giá trị cho thuộc tính main, xlab, ylab cho hàm plot().

Example:

```
plot(1, 3, xlab = "X", ylab = "Y", main = "Plot Example")
```

Output:



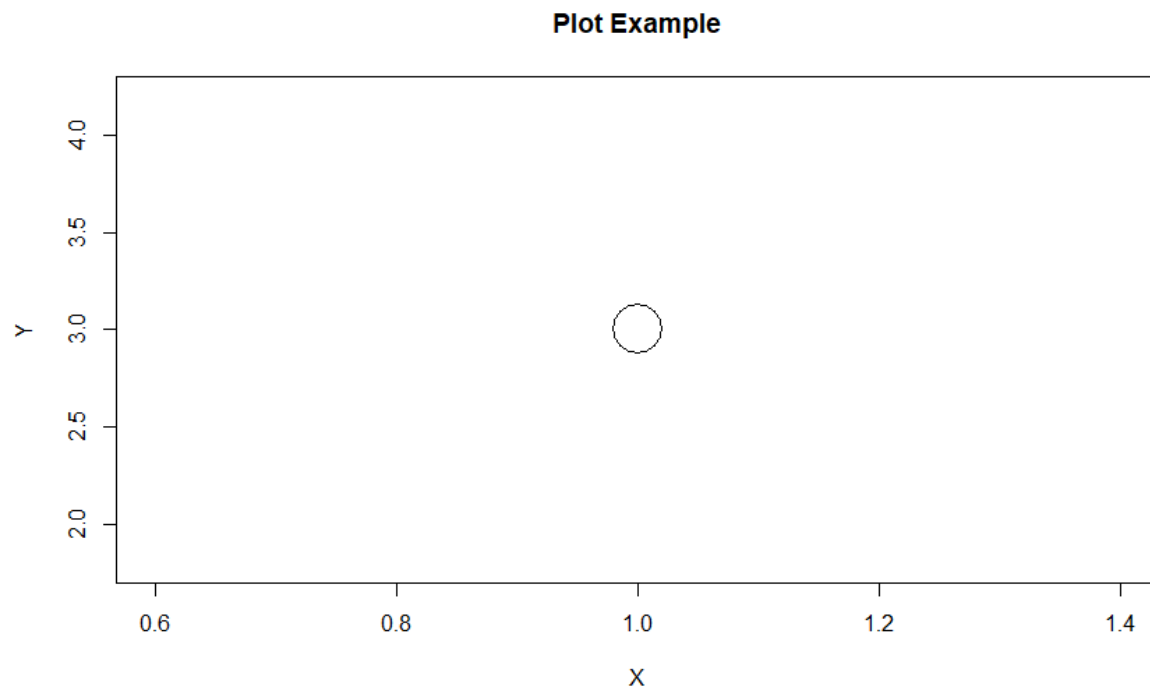
Kích thước của điểm

Để thay đổi kích thước của điểm, bạn cần gán giá trị cho thuộc tính `cex`.

Example:

```
plot(1, 3, xlab = "X", ylab = "Y", main = "Plot Example", cex = 5)
```

Output:



Màu của điểm

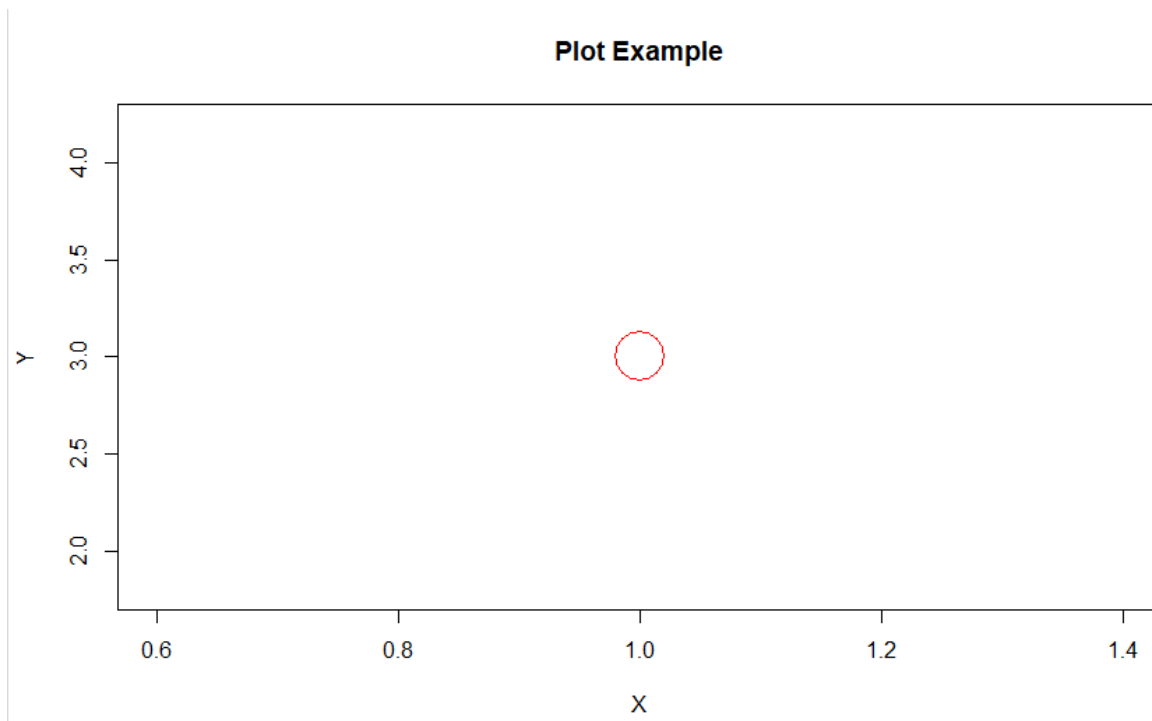
Để thay đổi màu của điểm, bạn cần gán giá trị cho thuộc tính col.

Example:

Change the color of a point to red.

```
plot(1, 3, xlab = "X", ylab = "Y", main = "Plot Example", cex = 5, col = 'red')
```

Output:



Vẽ nhiều điểm trên một biểu đồ

Để vẽ nhiều điểm trên một biểu đồ, bạn chỉ cần gán các vector giá trị vào thuộc tính x và y của hàm `plot()`.

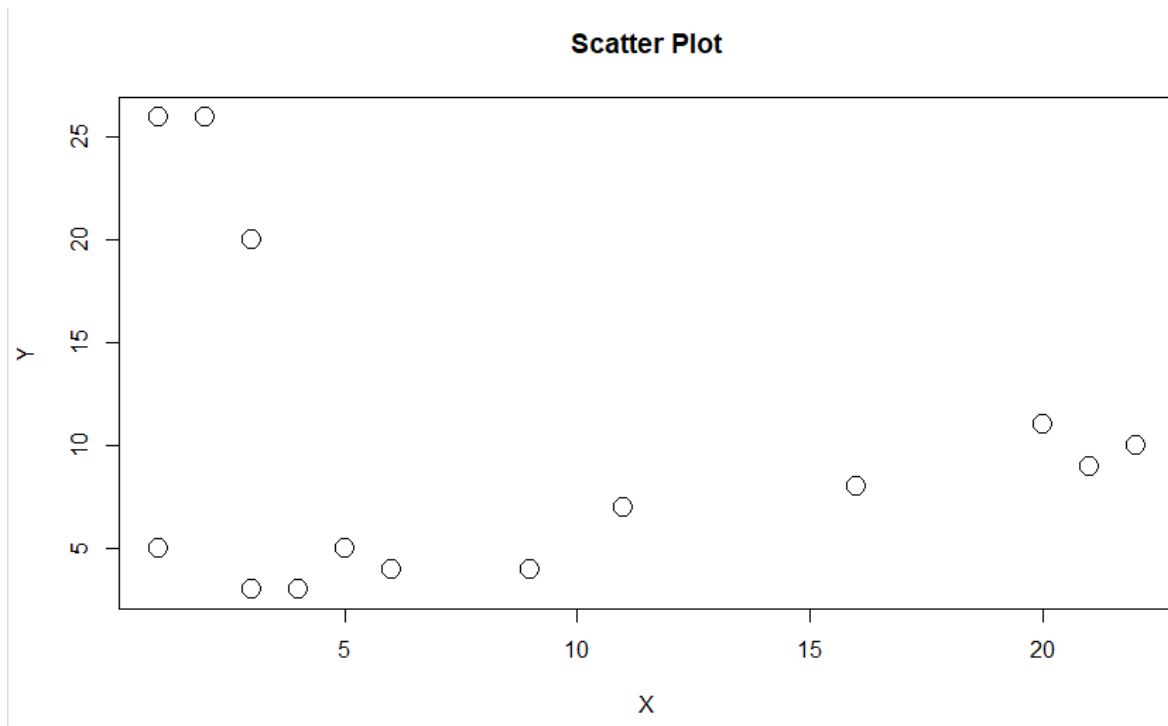
Example:

```
x <- c(1, 5, 3, 4, 6, 9, 11, 16, 20, 21, 22, 3, 2, 1)
```

```
y <- c(5, 5, 3, 3, 4, 4, 7, 8, 11, 9, 10, 20, 26, 26)
```

```
plot(x, y, cex = 2, xlab = 'X', ylab = 'Y', main = 'Scatter Plot')
```

Output:



Phân biệt các tập dữ liệu trên một biểu đồ

Ta thường phân biệt các tập dữ liệu trên một biểu đồ dựa trên kích thước hoặc màu sắc của các tập điểm. Mỗi tập dữ liệu sẽ có màu sắc hoặc kích thước tập điểm riêng.

Để tạo một biểu đồ với nhiều tập dữ liệu khác nhau, bạn cần kết hợp hàm `plot()` và `points()`. Trước tiên, bạn cần tạo một biểu đồ với hàm `plot()`. Sau đó, bạn có thể thêm các tập dữ liệu khác mà bạn muốn bằng hàm `points()`.

Example:

The first dataset.

```
x1 <- c(1, 5, 3, 4, 6, 9, 11, 16, 20, 21, 22, 3, 2, 1)
```

```
y1 <- c(5, 5, 3, 3, 4, 4, 7, 8, 11, 9, 10, 20, 26, 26)
```

The second dataset.

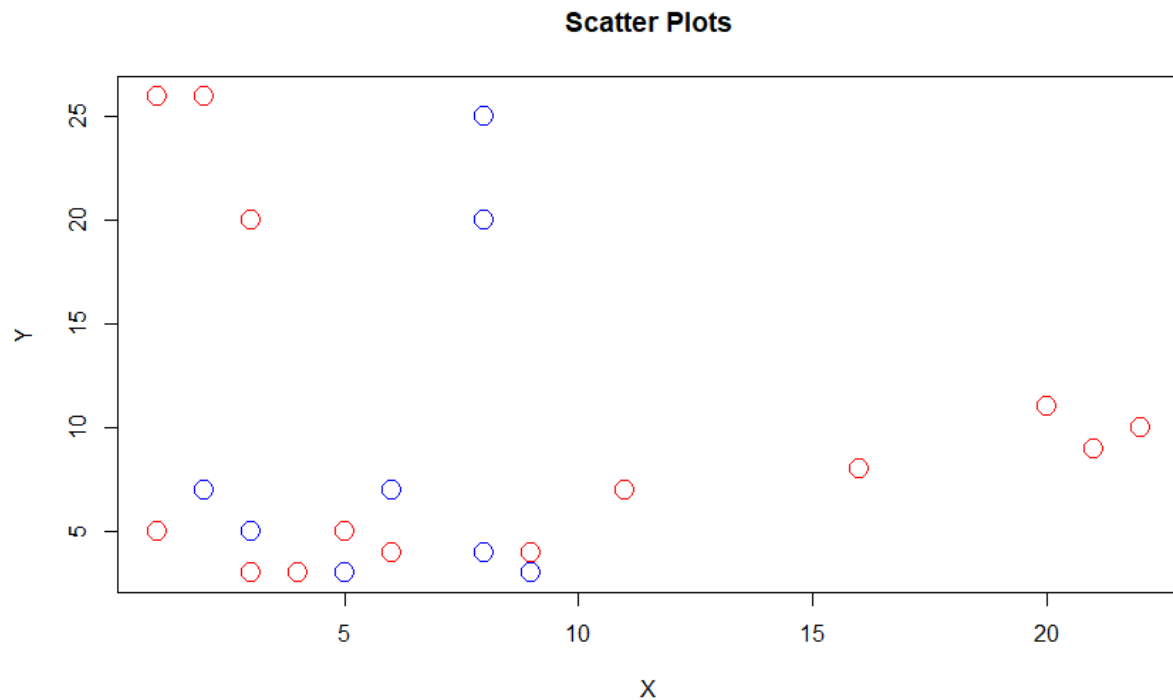
```
x2 <- c(3, 3, 3, 5, 6, 8, 2, 9, 8, 8)
```

```
y2 <- c(5, 36, 5, 3, 7, 4, 7, 3, 25, 20)
```

```
plot(x1, y1, cex = 2, xlab = 'X', ylab = 'Y', main = 'Scatter Plots', col = 'red')
```

```
points(x2, y2, cex = 2, col = 'blue')
```

Output:



Bài tập

1. Cho tập dữ liệu: `data <- iris`. Hãy vẽ biểu đồ phân tán trong đó tập dữ liệu x là cột `Sepal.length` y là cột `Sepal.width`.
2. Đặt tiêu đề là `Visualized Data` với tên của trục hoành và trục tung lần lượt là `Sepal.length`, `Sepal.width`.
3. Tăng kích thước của điểm lên 3 lần kích thước mặc định.
4. Đổi màu của điểm thành màu đỏ.

XXII. Line

Biểu đồ đường là biểu đồ gồm một đường nối các điểm trong tập dữ liệu. Để vẽ một biểu đồ đường, ta dùng hàm `plot()` với thuộc tính `type = "l"` như đã đề cập ở phần trước.

Để thay đổi tên tiêu đề, trục hoành, trục tung, kích thước đường kẻ, màu của đường, bạn có thể áp dụng cách tương tự như phần `Plot & Scatter Plot`.

Vẽ một đường trên một biểu đồ

Để vẽ một đường trên một biểu đồ, chúng ta sẽ dùng hàm `plot()`. Ngoài ra, để làm cho biểu đồ bắt mắt hơn, bạn có thể gán giá trị cho các thuộc tính `xlab`, `ylab`, `main`, `cex`, `col`, ...

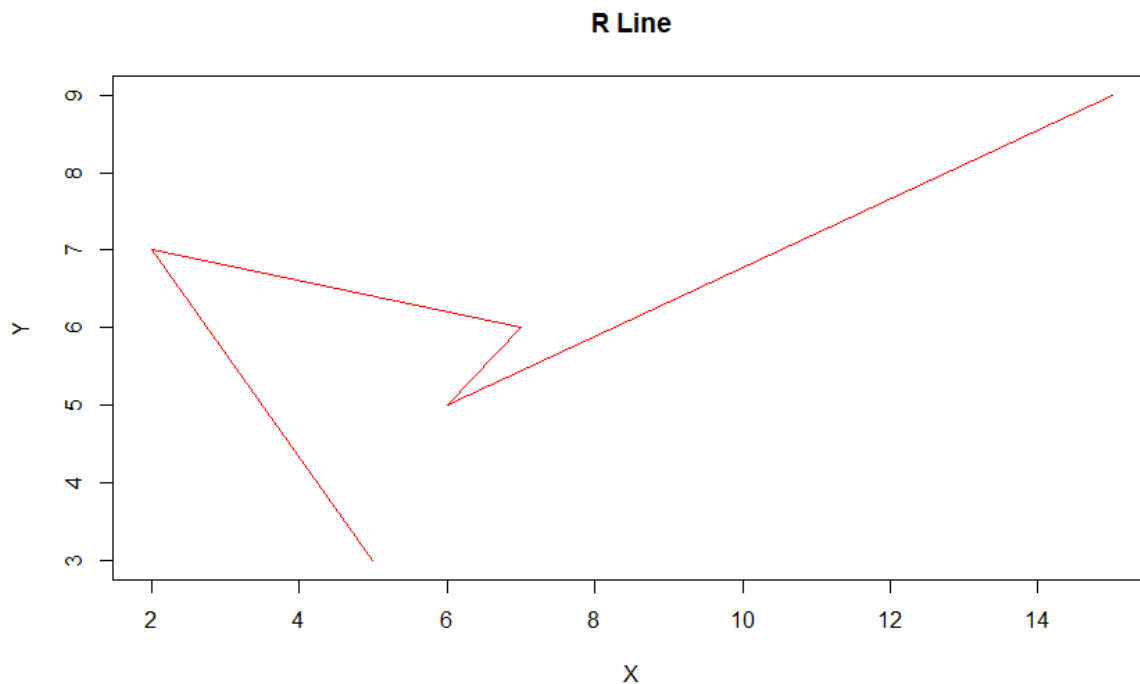
Example:

```
x <- c(5, 2, 7, 6, 15)
```

```
y <- c(3, 7, 6, 5, 9)
```

```
plot(x, y, type = 'l', col = 'red', main = 'R Line', xlab = 'X', ylab = 'Y')
```

Output:



Vẽ nhiều đường trên một biểu đồ

Bạn có thể vẽ nhiều đường trên một biểu đồ bằng việc kết hợp giữa hàm `plot()` và `lines()`. Trước tiên, bạn cần tạo một biểu đồ với hàm `plot()`. Sau đó, bạn có thể thêm các đường bạn muốn bằng hàm `lines()`.

Example:

```
# The first data.
```

```
x <- c(5, 2, 7, 6, 15)
```

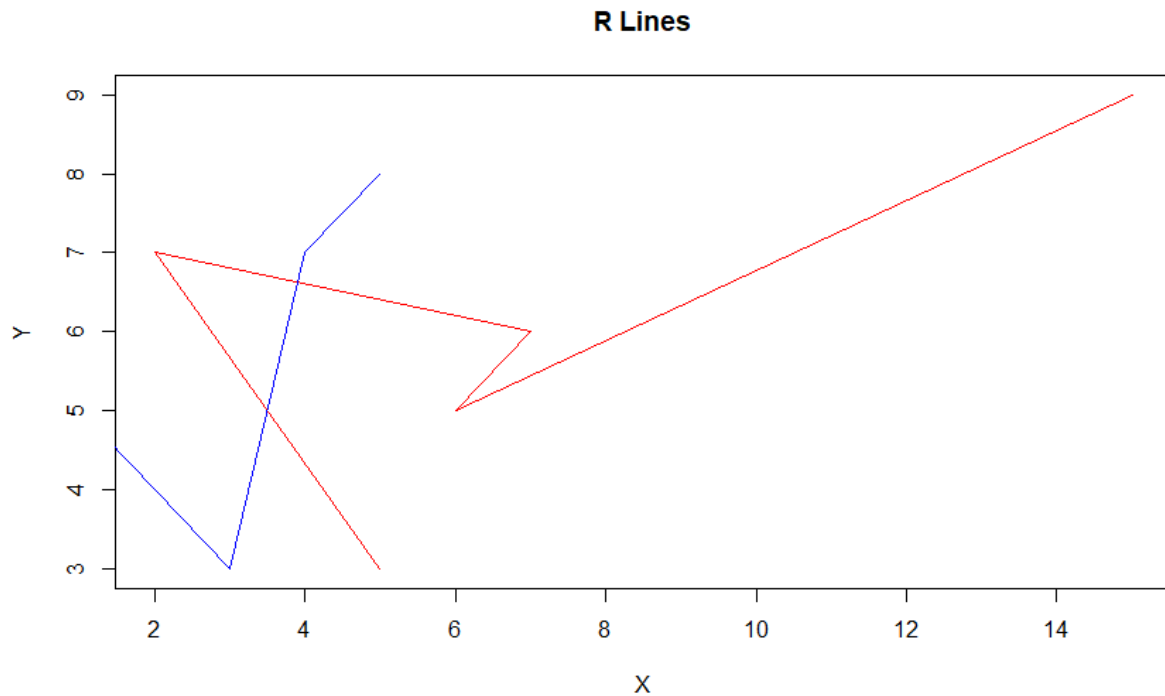
```
y <- c(3, 7, 6, 5, 9)
```

```
# The second data.
```

```
vec <- c(5, 4, 3, 7,8)
```

```
plot(x, y, type = 'l', col = 'red', main = 'R Lines', xlab = 'X', ylab = 'Y')  
lines(vec, col = 'blue')
```

Output:



Bài tập

1. Cho tập dữ liệu:

```
x <- c(5, 7, 8, 10, 15)
```

```
y <- c(3, 9, 3, 6, 9)
```

Hãy vẽ biểu đồ đường cho tập dữ liệu đó.

2. Đặt tên tiêu đề là Line Example, tên trục hoành và trục tung lần lượt là X và Y.

3. Đổi màu đường thành màu vàng.

4. Thêm đường có dữ liệu sau vào biểu đồ và đổi màu đường đó thành màu đỏ:

```
x <- c(5, 2, 7, 6, 15)
```

```
y <- c(3, 7, 6, 5, 9)
```

XXIII. Line & Point

Cũng với hàm `plot()`, bạn có thể vẽ biểu đồ có thể kết hợp điểm và đường bằng thay đổi giá trị của thuộc tính `type`:

- "b": cả đường và điểm, điểm không đè lên đường.
- "c": dạng đường nhưng ở vị trí điểm sẽ trống.
- "o": cả đường và điểm nhưng điểm đè lên đường.

Điểm không đè lên đường

Để vẽ biểu đồ kết hợp đường và điểm tạo nên các đường đó mà điểm sẽ tạo nên các vùng trống của đường, ta cần gán giá trị "b" cho thuộc tính `type` trong hàm `plot()`.

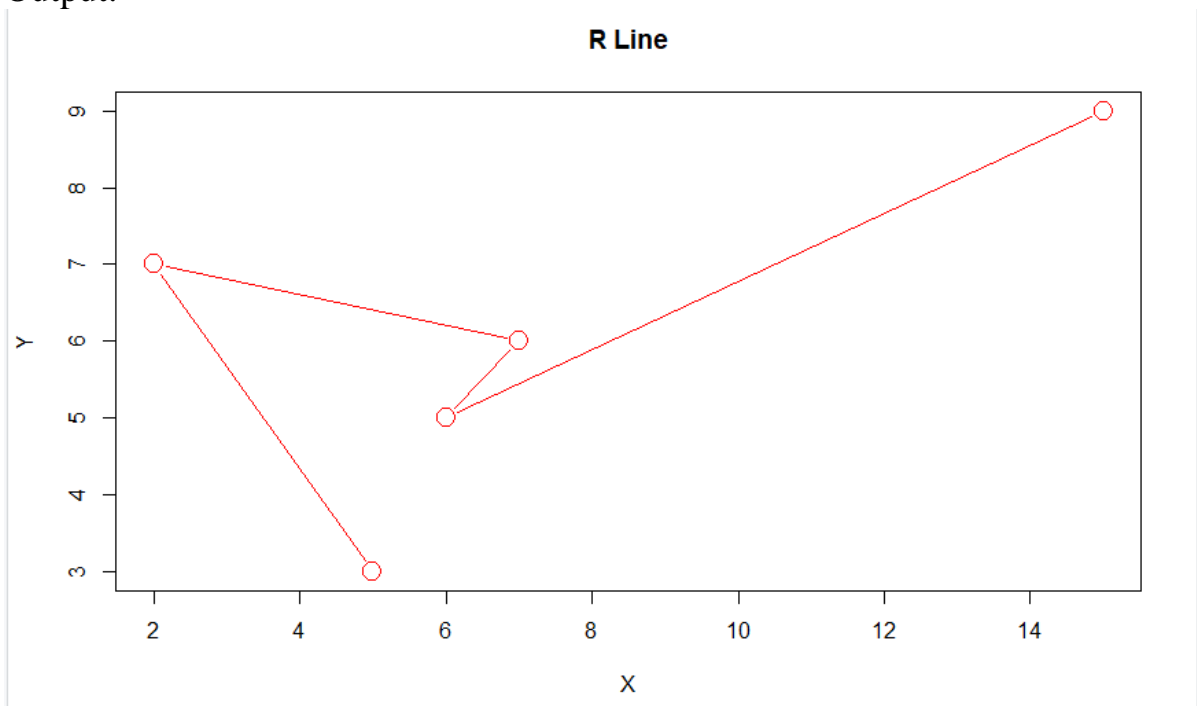
Example:

```
x <- c(5, 2, 7, 6, 15)
```

```
y <- c(3, 7, 6, 5, 9)
```

```
plot(x, y, type = 'b', cex = 2, col = 'red', main = 'R Line', xlab = 'X', ylab = 'Y')
```

Output:



Điểm đè lên đường

Để vẽ biểu đồ kết hợp đường và điểm tạo nên các đường đó mà đường không bị chia cắt, ta cần gán giá trị "o" cho thuộc tính type trong hàm plot().

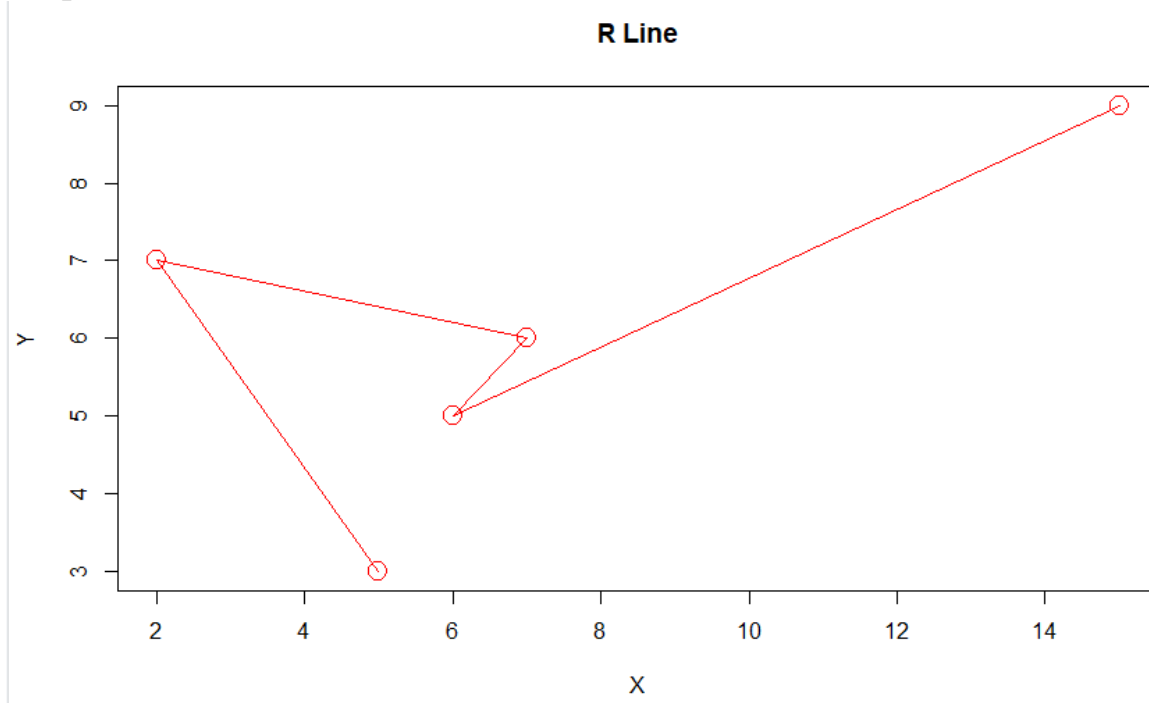
Example:

```
x <- c(5, 2, 7, 6, 15)
```

```
y <- c(3, 7, 6, 5, 9)
```

```
plot(x, y, type = 'o', cex = 2, col = 'red', main = 'R Line', xlab = 'X', ylab = 'Y')
```

Output:



Điểm tạo nên vùng trống

Để vẽ biểu đồ kết hợp đường và điểm tạo nên các đường đó mà ở vị trí các điểm sẽ tạo nên vùng trống cho đường, ta cần gán giá trị "c" cho thuộc tính type trong hàm plot().

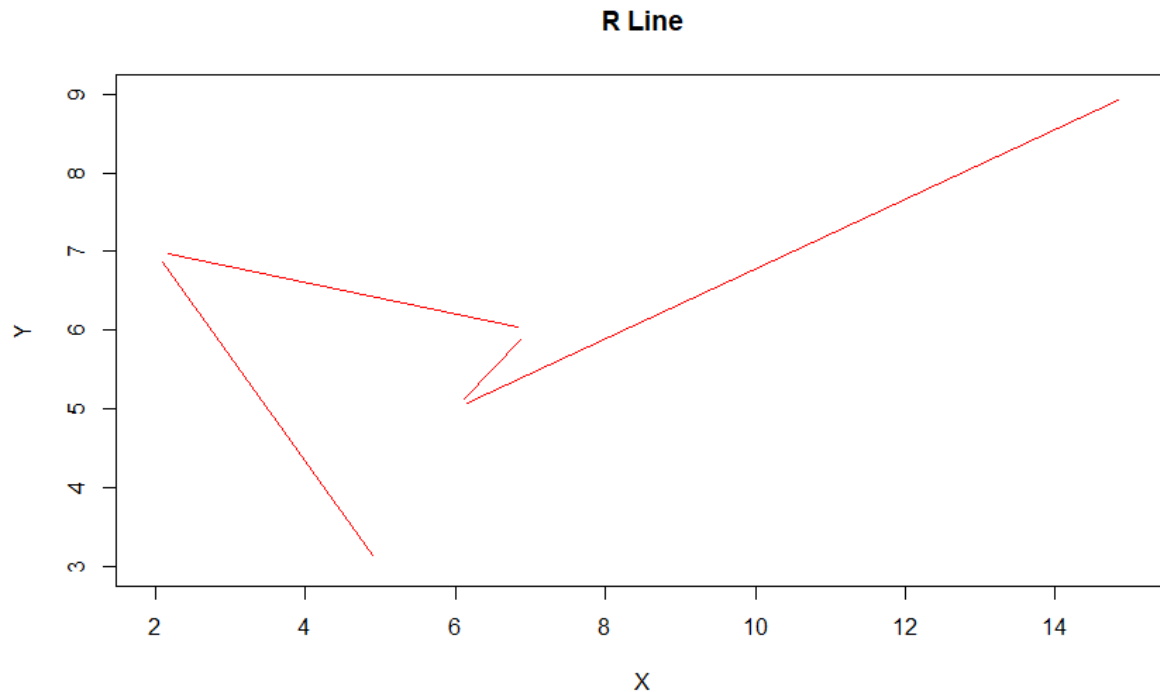
Example:

```
x <- c(5, 2, 7, 6, 15)
```

```
y <- c(3, 7, 6, 5, 9)
```

```
plot(x, y, type = 'c', cex = 2, col = 'red', main = 'R Line', xlab = 'X', ylab = 'Y')
```

Output:



Bài tập

1. Cho tập dữ liệu sau:

```
x <- c(5, 7, 8, 10, 15)
```

```
y <- c(3, 9, 3, 6, 9)
```

và thực hiện các yêu cầu sau:

- Vẽ biểu đồ kết hợp điểm và đường sao cho điểm không đè lên đường.
- Vẽ biểu đồ kết hợp điểm và đường sao cho điểm đè lên đường.
- Vẽ biểu đồ kết hợp điểm và đường sao cho điểm tạo nên vùng trống.

XXIV. Pie Charts

Biểu đồ tròn là dạng biểu đồ thường được dùng để vẽ các biểu đồ liên quan đến cơ cấu, tỷ lệ các thành phần trong một tổng thể chung hoặc cũng có thể vẽ biểu đồ tròn khi tỷ lệ % trong bảng số liệu cộng lại tròn 100.

Để vẽ biểu đồ tròn trong R, ta sử dụng hàm `pie()`.

Cú pháp cơ bản:

```
pie(x, labels, edges, clockwise, radius, init.angle, main, col)
```


Trong đó:

- x: Vector tỷ trọng của các lát cắt trong biểu đồ.
- labels: Vector lưu trữ tên của các lát cắt trong biểu đồ.
- edges: Mặc định là 200. Kích thước đường viền của biểu đồ tròn.
- clockwise: Mặc định là FALSE. Tỷ trọng của các lát cắt trong vector x được vẽ theo chiều kim đồng hồ hay không.
- radius: Để điều chỉnh kích thước bán kính của biểu đồ tròn.
- init.angle: Giá trị số đo góc (độ) bắt đầu của các lát cắt. Mặc định là 0 độ.
- main: Tên tiêu đề của biểu đồ.
- col: Vector gồm các màu sắc sử dụng cho các lát cắt trong biểu đồ

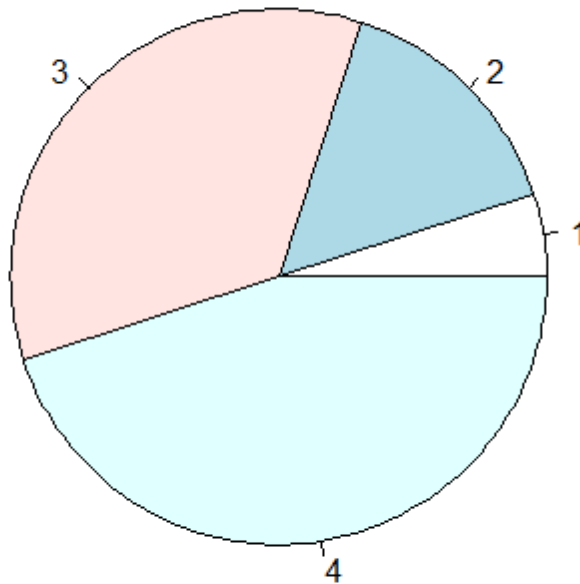
Example:

```
# Vector of pies.
```

```
x <- c(5, 15, 35, 45)
```

```
pie(x)
```

Output:



Thay đổi góc bắt đầu

Bạn có thể thay đổi góc bắt đầu của các lát cắt bằng gán giá trị cho thuộc tính `init.angle` trong hàm `pie()`.

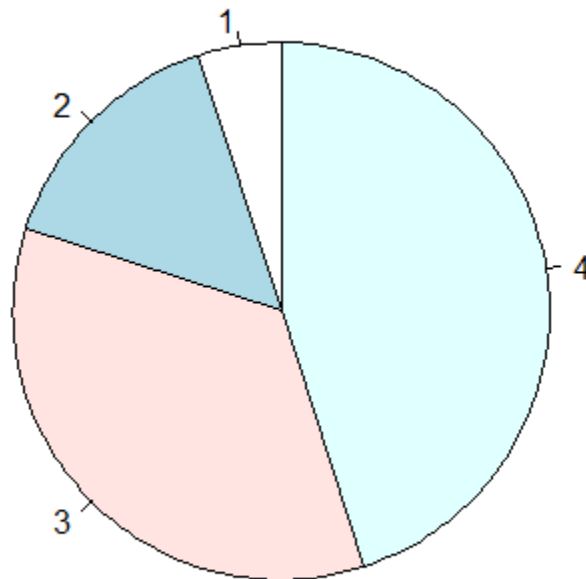
Example:

```
x <- c(5, 15, 35, 45)
```

```
# Change the start of the first pie at 90 degrees.
```

```
pie(x, init.angle = 90)
```

Output:



Tiêu đề và nhãn của các lát cắt

Để đặt tiêu đề và nhãn của các lát cắt trong biểu đồ, bạn cần điều chỉnh giá trị lần lượt cho thuộc tính `main` và `label` trong hàm `pie()`.

Example:

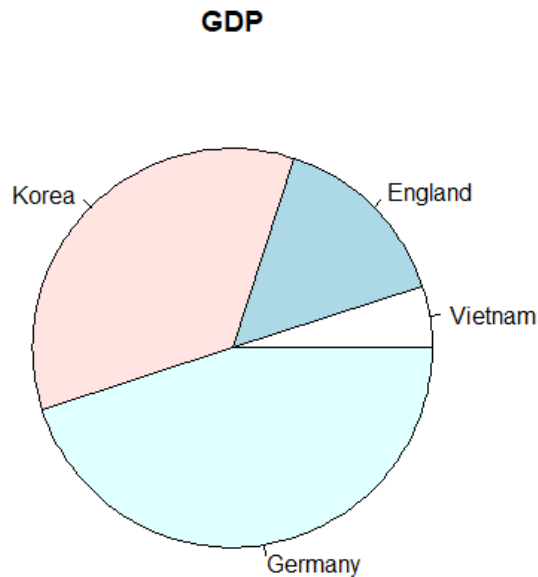
```
x <- c(5, 15, 35, 45)
```

```
# Create a vector whose values are the name of labels.
```

```
label_name <- c('Vietnam', 'England', 'Korea', 'Germany')
```

```
# Set the title of this chart as 'GDP'.  
pie(x, main = 'GDP', label = label_name)
```

Output:



Thay đổi chiều của các lát cắt trong biểu đồ tròn

Mặc định của hàm `pie()`, các lát cắt của nó sẽ được chia lần lượt theo chiều ngược kim đồng hồ. Để thay đổi thành chiều kim đồng hồ, bạn cần gán giá trị `TRUE` cho thuộc tính `clockwise` trong hàm `pie()`.

Example:

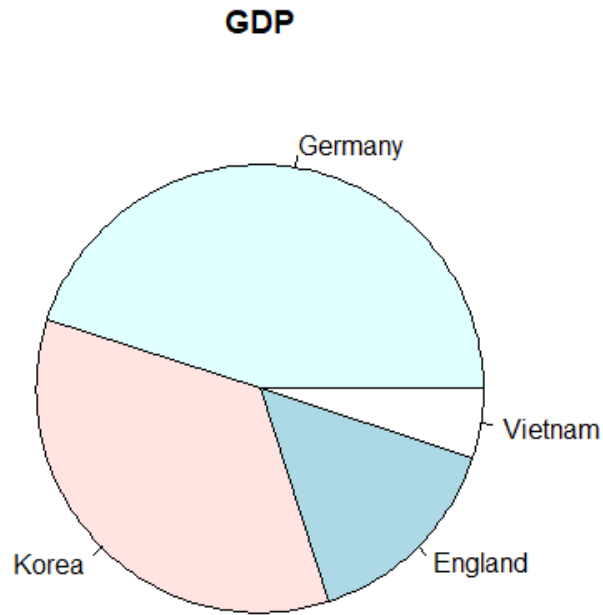
```
x <- c(5, 15, 35, 45)
```

```
# Create a vector whose values are the name of labels.
```

```
label_name <- c('Vietnam', 'England', 'Korea', 'Germany')
```

```
pie(x, main = 'GDP', label = label_name, clockwise = TRUE, init.angle = 0)
```

Output:



Gán màu sắc của các lát cắt

Để gán màu sắc của các lát cắt trong biểu đồ tròn, bạn cần gán một vector bao gồm các màu sắc mong muốn vào thuộc tính col của hàm pie().

Example:

```
x <- c(5, 15, 35, 45)
```

```
# Create a vector whose values are the name of labels.
```

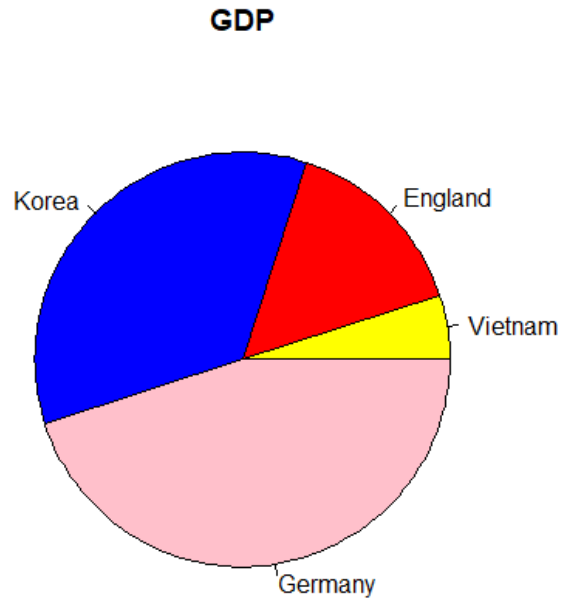
```
label_name <- c('Vietnam', 'England', 'Korea', 'Germany')
```

```
# Vector of slices's color.
```

```
colors <- c('Yellow', 'Red', 'Blue', 'Pink')
```

```
pie(x, main = 'GDP', label = label_name, col = colors)
```

Output:



Chú thích

Để thêm chú thích cho các nhãn và màu sắc của các lát cắt, bạn có thể dùng hàm `legend()`.

Example:

```
x <- c(5, 15, 35, 45)
```

```
# Create a vector whose values are the name of labels.
```

```
label_name <- c('Vietnam', 'England', 'Korea', 'Germany')
```

```
# Vector of slices's color.
```

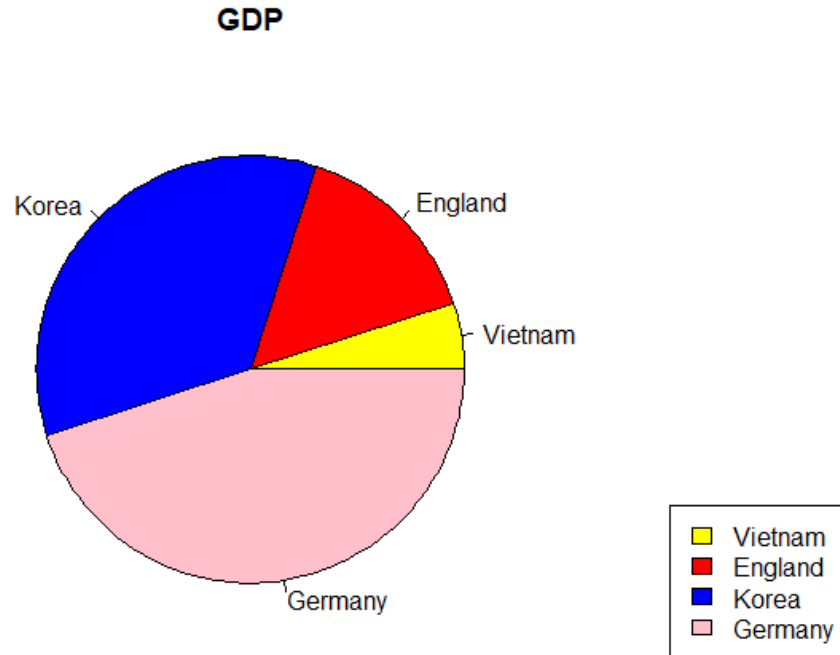
```
colors <- c('Yellow', 'Red', 'Blue', 'Pink')
```

```
pie(x, main = 'GDP', label = label_name, col = colors)
```

```
# Add the explanation box.
```

```
legend('bottomright', label_name, fill = colors)
```

Output:



Pie Charts 3D

Ngoài ra, bạn có thể trực quan hóa biểu đồ tròn của bạn theo dạng 3D. Để có thể vẽ một biểu đồ tròn theo dạng 3D, bạn có thể dùng hàm `pie3D()` trong thư viện `plotrix`. Nếu môi trường của bạn chưa có thư viện `plotrix`, bạn cần cài đặt nó trước khi sử dụng hàm `pie3D()`.

Example:

```
x <- c(5, 15, 35, 45)
```

```
# Create a vector whose values are the name of labels.
```

```
label_name <- c('Vietnam', 'England', 'Korea', 'Germany')
```

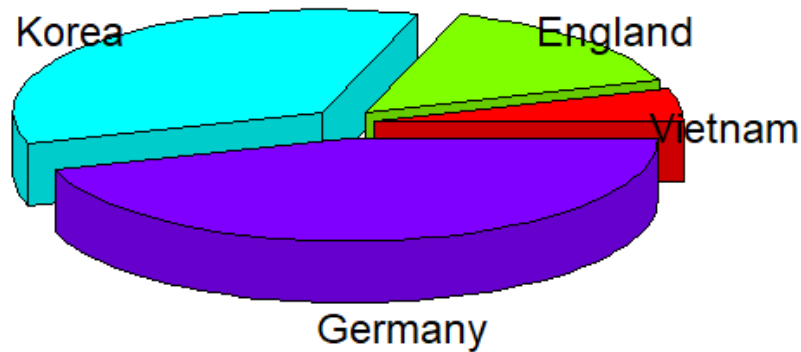
```
# Vector of slices's color.
```

```
colors <- c('Yellow', 'Red', 'Blue', 'Pink')
```

```
plotrix::pie3D(x, labels = label_name, explode = 0.1, main = 'GDP')
```

Output:

GDP



Bài tập

1. Cho số liệu tỷ trọng dân số theo độ tuổi của Việt Nam năm 2017 như sau:

- < 15 tuổi: 25.2%
- 15 - 64 tuổi: 69.3%
- > 64 tuổi: 5.5%

Hãy trực quan hóa số liệu trên bằng biểu đồ tròn.

2. Đặt tên tiêu đề cho biểu đồ là: Cơ cấu dân số Việt Nam năm 2017.
3. Đặt tên nhãn lần lượt là: <15, 15-64, >64
4. Đổi màu của các lát cắt thành: đỏ, vàng, xanh.
5. Đặt hộp chú thích ở góc trái dưới.
6. Trực quan hóa biểu đồ trên theo dạng 3D.

XXV. Bar Charts

Biểu đồ cột là biểu đồ trong đó dữ liệu của nó được biểu diễn qua các hình chữ nhật. Độ dài của hình chữ nhật tỉ lệ thuận với giá trị trực quan của biến mà chúng đại diện. Biểu đồ cột có thể được biểu diễn qua trục nằm dọc hoặc nằm ngang. Trong R, hàm `barplot()` được dùng để vẽ một biểu đồ cột.

Cú pháp cơ bản:

`barplot(data, width, height, main, xlab, ylab, col, names.arg, density, horiz)`

Trong đó:

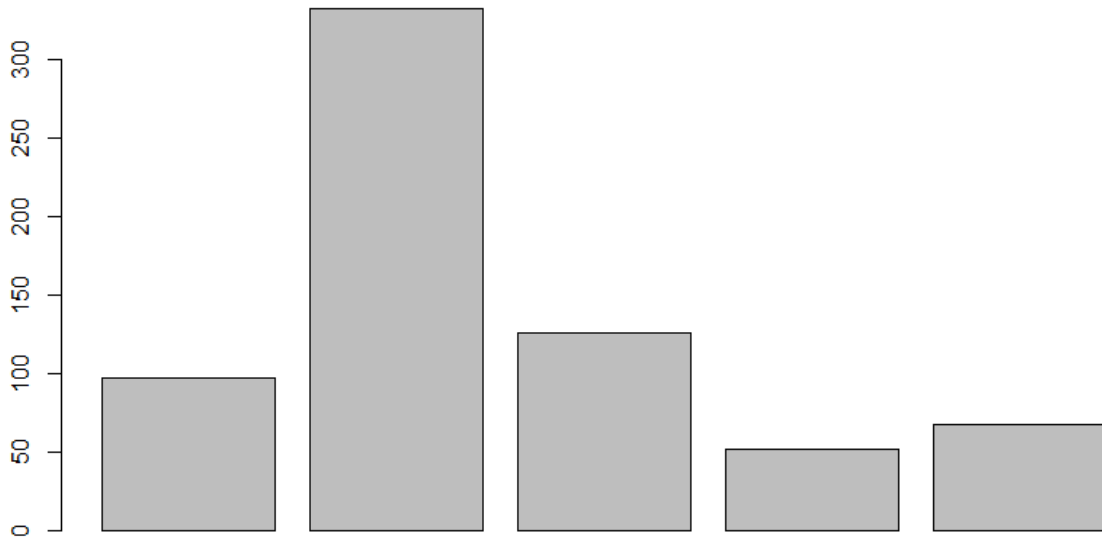
- `data`: Dữ liệu số của biểu đồ.
- `width`: Vector gồm các giá trị độ rộng của cột lần lượt của các biến trong tập dữ liệu.
- `height`: Vector gồm các giá trị độ cao của cột lần lượt của các biến trong tập dữ liệu.
- `main`: Tiêu đề của biểu đồ.
- `xlab`: Tên trục hoành của biểu đồ.
- `ylab`: Tên trục tung của biểu đồ.
- `col`: Vector gồm các giá trị màu sắc của cột lần lượt của các biến trong tập dữ liệu.
- `names.arg`: Tên của các biến của biểu đồ được biểu diễn ở dưới trục hoành.
- `density`: Vector gồm mật độ các đường gạch chéo của cột lần lượt của các biến trong tập dữ liệu.
- `horiz`: Biểu diễn các cột theo chiều ngang hay không. Mặc định là `FALSE`, cột được biểu diễn theo chiều dọc.

Example:

```
populations <- (97.47, 331.9, 125.7, 51.74, 67.33)
```

```
barplot(populations)
```

Output:



Tên tiêu đề, trục hoành, trục tung của biểu đồ cột

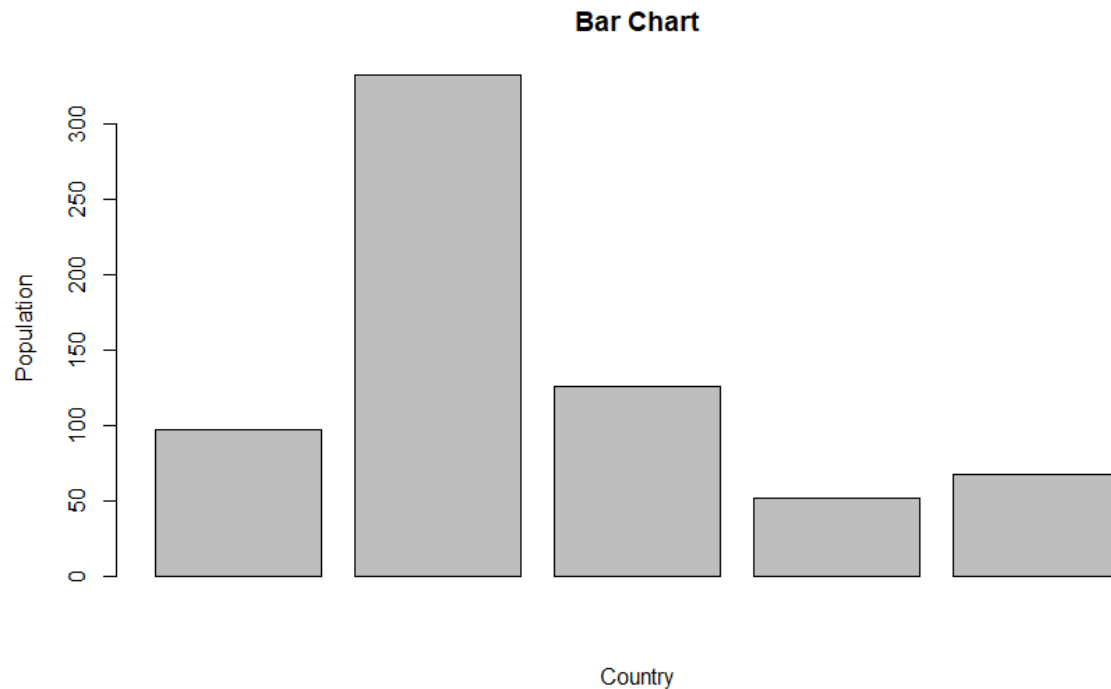
Cũng giống như hàm `plot()` ở các phần trước, để đặt tên tiêu đề, trục hoành, trục tung của biểu đồ, bạn cần gán giá trị vào lần lượt các thuộc tính `main`, `xlab`, `ylab` trong hàm `barplot()`.

Example:

```
populations <- c(97.47, 331.9, 125.7, 51.74, 67.33)
```

```
barplot(populations, main = 'Bar Chart', xlab = 'Country', ylab = 'Population')
```

Output:



Nhãn của các cột trong biểu đồ

Nhãn của các cột trong biểu đồ được thể hiện ở dưới trục hoành. Để đặt tên nhãn của các cột, bạn cần gán giá trị vector bao gồm tên của chúng vào thuộc tính `names.arg` trong hàm `barplot()`.

Example:

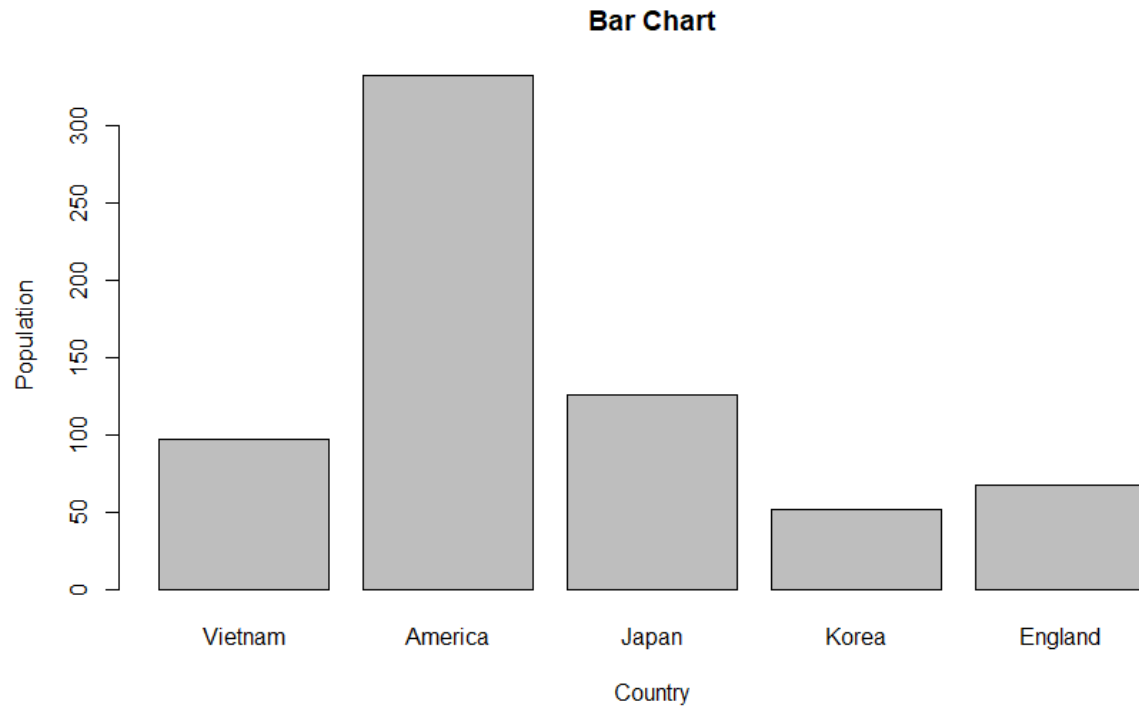
```
populations <- c(97.47, 331.9, 125.7, 51.74, 67.33)
```

```
# Labels of the previos data.
```

```
countries <- c('Vietnam', 'America', 'Japan', 'Korea', 'England')
```

```
barplot(populations, main = 'Bar Chart', xlab = 'Country', ylab = 'Population',  
names.arg = countries)
```

Output:



Màu sắc của các cột trong biểu đồ

Để dễ phân biệt các cột với nhau hơn, ta có thể gán từng màu sắc lần lượt cho chúng bằng việc gán tập giá trị màu sắc cho thuộc tính `col` trong hàm `barplot()`.

Example:

```
populations <- c(97.47, 331.9, 125.7, 51.74, 67.33)
```

```
# Labels of the previous data.
```

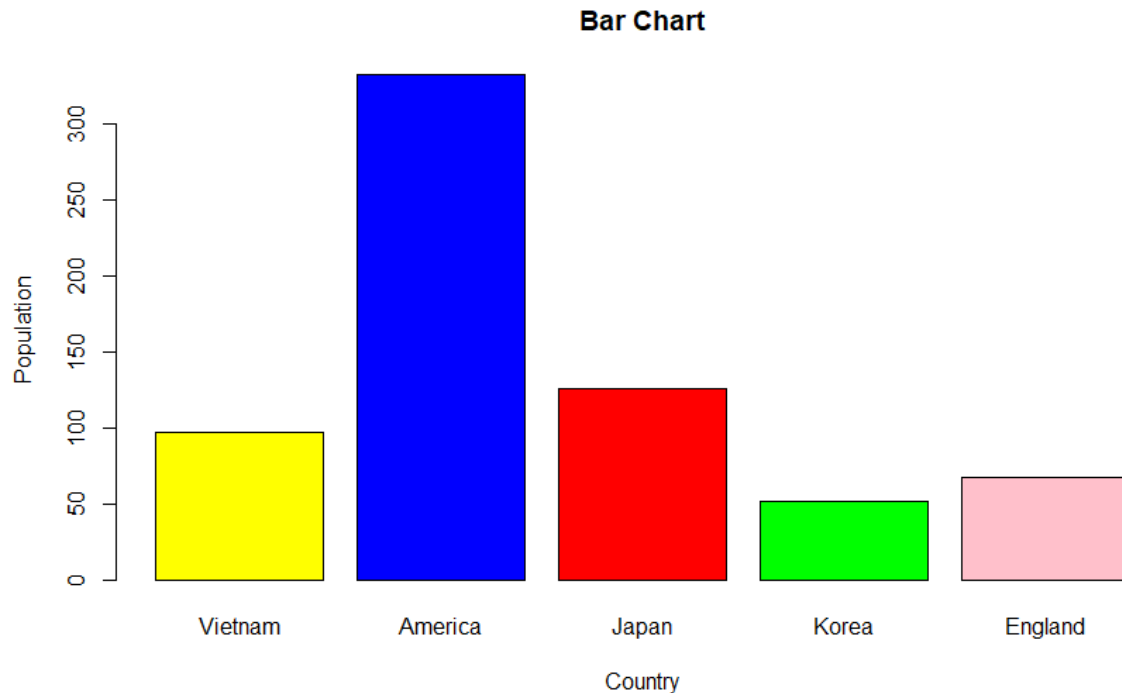
```
countries <- c('Vietnam', 'America', 'Japan', 'Korea', 'England')
```

```
# Colors of rectangles.
```

```
colors <- c('Yellow', 'Blue', 'Red', 'Green', 'Pink')
```

```
barplot(populations, main = 'Bar Chart', xlab = 'Country', ylab = 'Population',  
names.arg = countries, col = colors)
```

Output:



Độ rộng của các cột trong biểu đồ

Bạn có thể tùy chỉnh độ rộng và độ cao của các cột trong biểu đồ bằng thuộc tính `width`, `height` của hàm `barplot()`. Tuy nhiên, chúng ta chỉ thường xuyên làm việc với độ rộng, còn độ cao thường là để biểu diễn giá trị của các biến trong biểu đồ. Để tùy chỉnh độ rộng cho các cột trong biểu đồ, bạn cần gán một vector bao gồm tỷ lệ lần lượt của các biến cho thuộc tính `width`.

Example:

```
populations <- c(97.47, 331.9, 125.7, 51.74, 67.33)
```

```
# Labels of the previos data.
```

```
countries <- c('Vietnam', 'America', 'Japan', 'Korea', 'England')
```

```
# Colors of retangulars.
```

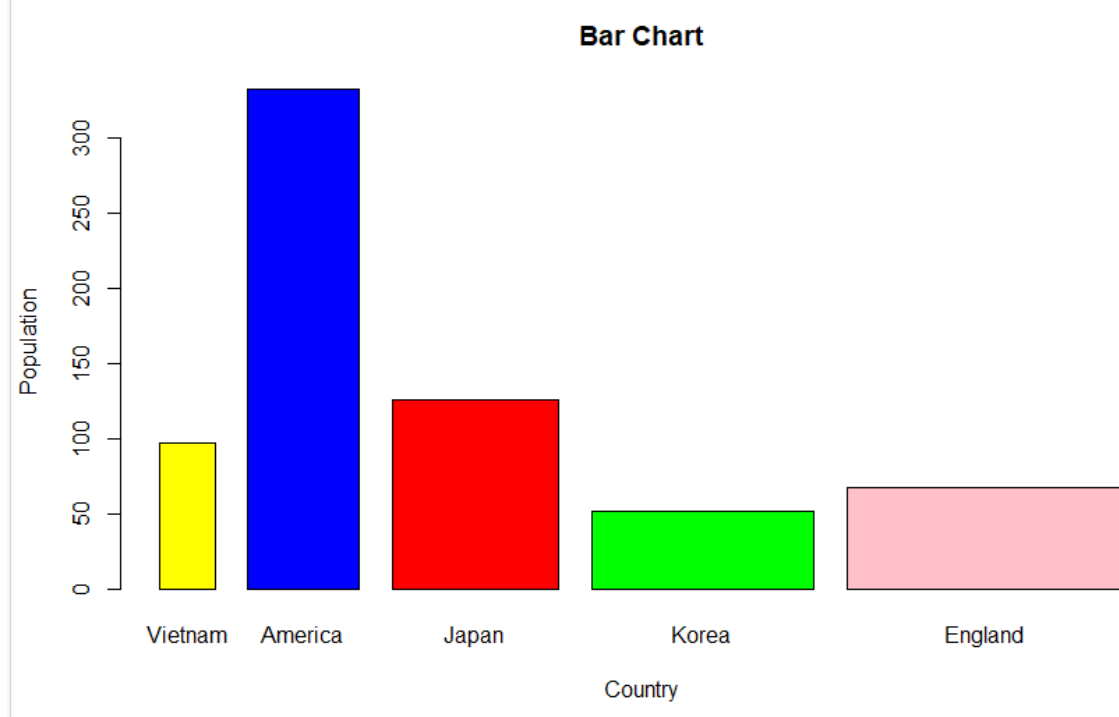
```
colors <- c('Yellow', 'Blue', 'Red', 'Green', 'Pink')
```

```
# The width of retangulars.
```

```
bar_width <- c(1, 2, 3, 4, 5)
```

```
barplot(populations, main = 'Bar Chart', xlab = 'Country', ylab = 'Population',  
names.arg = countries, col = colors, width = bar_width)
```

Output:



Mật độ đường chéo trong các cột của biểu đồ

Để biểu diễn mật độ các đường chéo của các cột trong biểu đồ theo inch, bạn cần gán giá trị cho thuộc tính density trong hàm barplot().

Example:

```
populations <- c(97.47, 331.9, 125.7, 51.74, 67.33)
```

```
# Labels of the previos data.
```

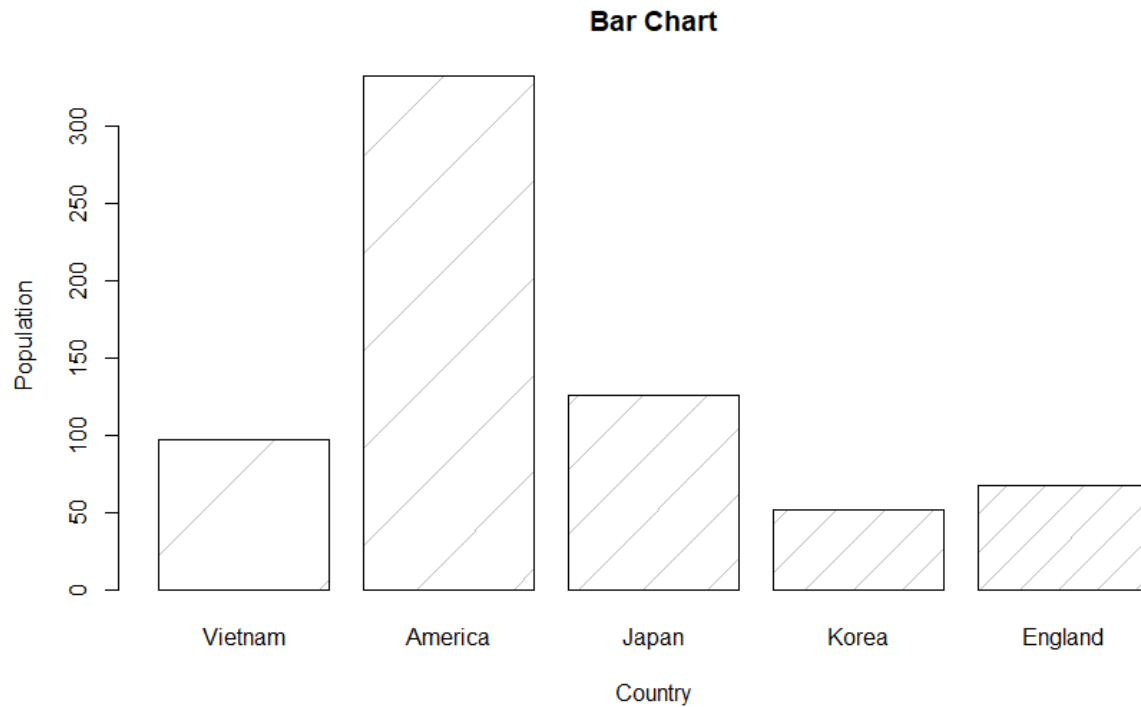
```
countries <- c('Vietnam', 'America', 'Japan', 'Korea', 'England')
```

```
# The density of retangulars.
```

```
bar_density <- c(1, 2, 3, 4, 5)
```

```
barplot(populations, main = 'Bar Chart', xlab = 'Country', ylab = 'Population',  
names.arg = countries, density = bar_density)
```

Output:



Biểu đồ cột nằm ngang

Để biểu diễn biểu đồ cột nằm ngang, bạn cần gán giá trị TRUE cho thuộc tính `horiz` trong hàm `barplot()`.

Example:

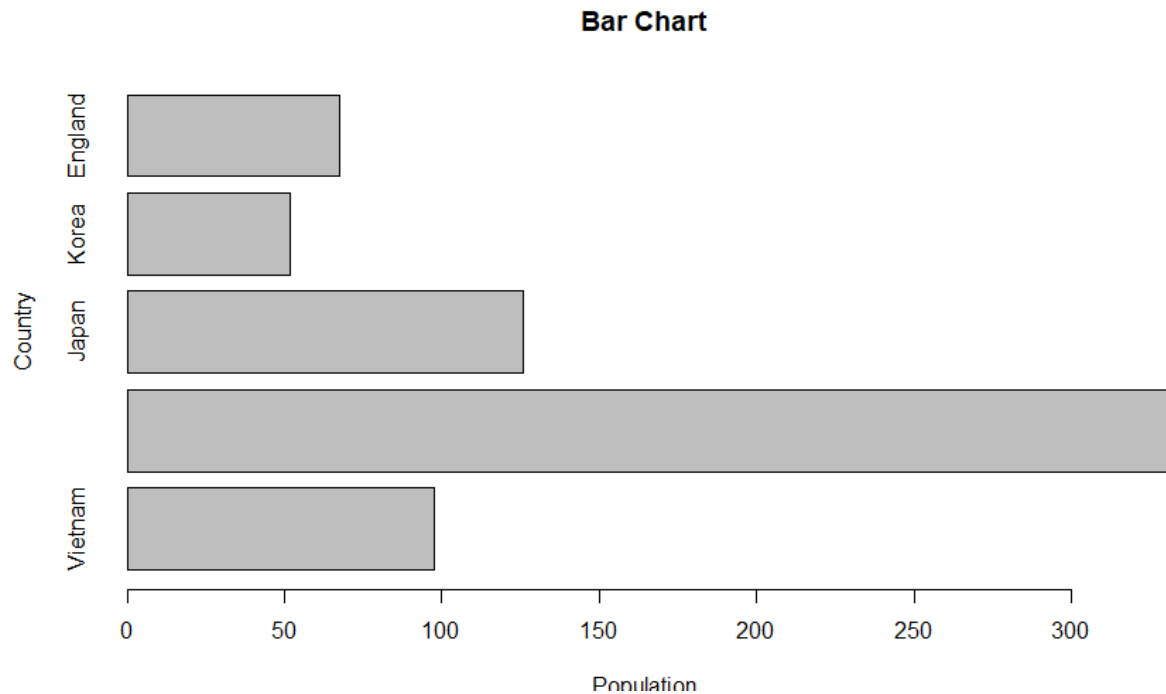
```
populations <- c(97.47, 331.9, 125.7, 51.74, 67.33)
```

```
# Labels of the previos data.
```

```
countries <- c('Vietnam', 'America', 'Japan', 'Korea', 'England')
```

```
barplot(populations, main = 'Bar Chart', xlab = 'Population', ylab = 'Country',  
names.arg = countries, horiz = TRUE)
```

Output:



Chú thích trong biểu đồ cột

Tương tự các biểu đồ khác, bạn có thể dùng hàm legend để chú thích cho các biến trong biểu đồ cột.

Example:

```
populations <- c(97.47, 331.9, 125.7, 51.74, 67.33)
```

```
# Labels of the previos data.
```

```
countries <- c('Vietnam', 'America', 'Japan', 'Korea', 'England')
```

```
# Colors of retangulars.
```

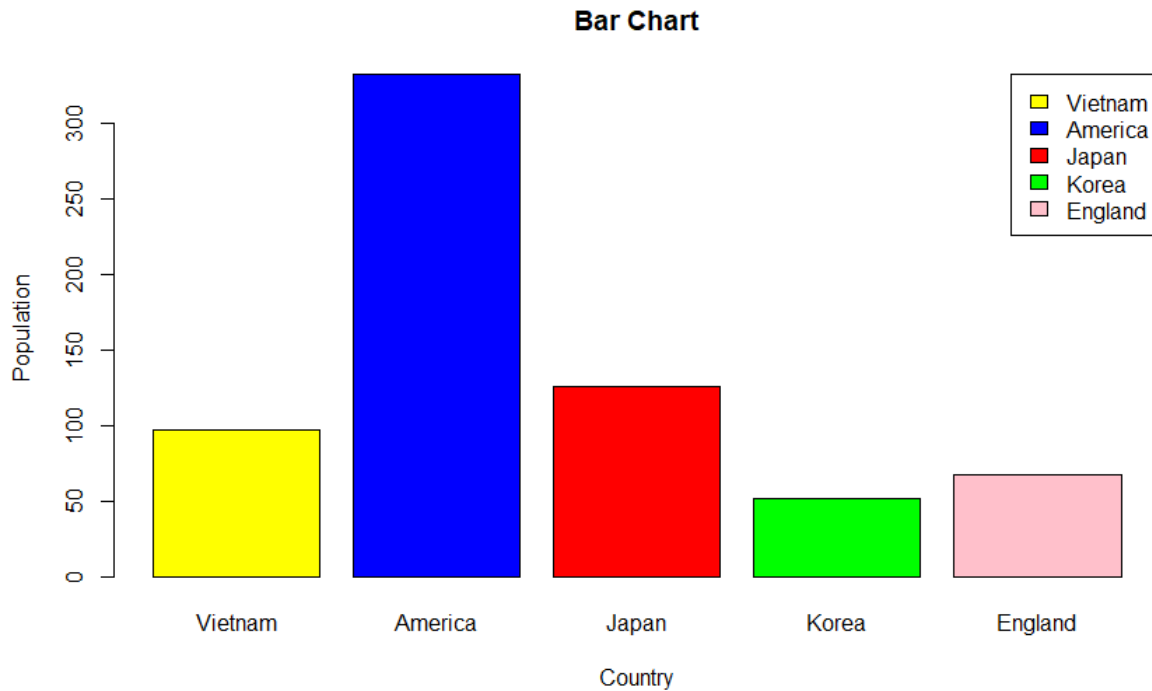
```
colors <- c('Yellow', 'Blue', 'Red', 'Green', 'Pink')
```

```
barplot(populations, main = 'Bar Chart', xlab = 'Country', ylab = 'Population',
names.arg = countries, col = colors)
```

```
# Add the explanation box.
```

```
legend('topright', countries, fill = colors)
```

Output:



Bài tập

1. Cho số liệu tỷ trọng dân số theo độ tuổi của Việt Nam năm 2017 như sau:
 - < 15 tuổi: 25.2%
 - 15 - 64 tuổi: 69.3%
 - > 64 tuổi: 5.5%

Hãy trực quan hóa số liệu trên bằng biểu đồ cột.

2. Đặt tên tiêu đề cho biểu đồ là: Cơ cấu dân số Việt Nam năm 2017.
3. Đặt tên nhãn cho các cột lần lượt là: <15, 15-64, >64
4. Đổi màu của các cột thành: đỏ, vàng, xanh.
5. Đổi độ rộng của các cột theo tỷ lệ là 1:2:3.
6. Đặt hộp chú thích ở góc phải trên.
7. Đổi chiều của biểu đồ thành nằm ngang.

XXVI. R Dataset

Để bắt đầu với việc phân tích dữ liệu và thống kê với R, chúng ta sẽ bắt đầu tìm hiểu về các tập dữ liệu có sẵn trong R. Dataset là một tập hợp các dữ liệu được lưu

trữ dưới dạng bảng (thường thì cấu trúc dữ liệu của nó là dataframe). Cơ bản, R đã được xây dựng sẵn rất nhiều dataset trong thư viện 'datasets' để giúp bạn có thể tìm hiểu và làm quen với việc phân tích, thống kê dữ liệu với R.

Datasets package in R

Để xem thông tin của thư viện 'datasets' và các tập dữ liệu có sẵn trong thư viện, bạn hãy thực thi câu lệnh sau:

```
library(help = "datasets")
```

Output:

```
Information on package 'datasets'

Description:

Package:      datasets
Version:      4.2.2
Priority:      base
Title:        The R Datasets Package
Author:       R Core Team and contributors worldwide
Maintainer:   R Core Team <do-use-Contact-address@r-project.org>
Contact:      R-help mailing list <r-help@r-project.org>
Description:  Base R datasets.
License:      Part of R 4.2.2
Built:        R 4.2.2; ; 2022-10-31 14:28:30 UTC; windows

Index:

AirPassengers      Monthly Airline Passenger Numbers 1949-1960
BJSales            Sales Data with Leading Indicator
BOD               Biochemical Oxygen Demand
CO2               Carbon Dioxide Uptake in Grass Plants
ChickWeight       Weight versus age of chicks on different diets
DNase             Elisa assay of DNase
EuStockMarkets     Daily Closing Prices of Major European Stock
                  Indices, 1991-1998
Formaldehyde       Determination of Formaldehyde
HairEyeColor       Hair and Eye Color of Statistics Students
Harman23.cor       Harman Example 2.3
Harman74.cor       Harman Example 7.4
Indometh           Pharmacokinetics of Indomethacin
InsectSprays       Effectiveness of Insect Sprays
JohnsonJohnson    Quarterly Earnings per Johnson & Johnson Share
...

```

Đầu ra của nó sẽ là thông tin về thư viện datasets ở phần Description và ở phần Index là các tập dữ liệu có sẵn ở trong thư viện 'datasets'.

Đương nhiên, thư viện này hầu như chỉ giúp bạn trong việc học tập, rèn luyện. Trong thực tế, đa phần bạn sẽ làm việc với tập dữ liệu cá nhân của bạn.

Bạn có thể sử dụng trực tiếp các tập dữ liệu trong thư viện 'datasets' vì nó đã được cài đặt sẵn trong môi trường R.

Example:

```
# Print the rock dataset.
```

```
rock
```

Output:

	area	peri	shape	perm
1	4990	2791.900	0.0903296	6.3
2	7002	3892.600	0.1486220	6.3
3	7558	3930.660	0.1833120	6.3
4	7352	3869.320	0.1170630	6.3
5	7943	3948.540	0.1224170	17.1
6	7979	4010.150	0.1670450	17.1
7	9333	4345.750	0.1896510	17.1
8	8209	4344.750	0.1641270	17.1
9	8393	3682.040	0.2036540	119.0
10	6425	3098.650	0.1623940	119.0
11	9364	4480.050	0.1509440	119.0
12	8624	3986.240	0.1481410	119.0
13	10651	4036.540	0.2285950	82.4
14	8868	3518.040	0.2316230	82.4
15	9417	3999.370	0.1725670	82.4
16	8874	3629.070	0.1534810	82.4
17	10962	4608.660	0.2043140	58.6
18	10743	4787.620	0.2627270	58.6
19	11878	4864.220	0.2000710	58.6
20	9867	4479.410	0.1448100	58.6
21	7838	3428.740	0.1138520	142.0
22	11876	4353.140	0.2910290	142.0
23	12212	4697.650	0.2400770	142.0
24	8233	3518.440	0.1618650	142.0
25	6360	1977.390	0.2808870	740.0
26	4193	1379.350	0.1794550	740.0
27	7416	1916.240	0.1918020	740.0
28	5246	1585.420	0.1330830	740.0
29	6509	1851.210	0.2252140	890.0
30	4895	1239.660	0.3412730	890.0
31	6775	1728.140	0.3116460	890.0
32	7894	1461.060	0.2760160	890.0
33	5980	1426.760	0.1976530	950.0
34	5318	990.388	0.3266350	950.0
35	7392	1350.760	0.1541920	950.0
36	7894	1461.060	0.2760160	950.0
37	3469	1376.700	0.1769690	100.0

```
38 1468 476.322 0.4387120 100.0
39 3524 1189.460 0.1635860 100.0
40 5267 1644.960 0.2538320 100.0
41 5048 941.543 0.3286410 1300.0
42 1016 308.642 0.2300810 1300.0
43 5605 1145.690 0.4641250 1300.0
44 8793 2280.490 0.4204770 1300.0
45 3475 1174.110 0.2007440 580.0
46 1651 597.808 0.2626510 580.0
47 5514 1455.880 0.1824530 580.0
48 9718 1485.580 0.2004470 580.0
```

Xem thông tin của 1 tập dữ liệu

Để xem thông tin của 1 tập dữ liệu có sẵn trong thư viện 'datasets', bạn có thể dùng toán tử ?.

Example:

```
# Get the information of the rock dataset.
?rock
```

Output:

R: Measurements on Petroleum Rock Samples ▾ Find in Topic

rock {datasets} R Documentation

Measurements on Petroleum Rock Samples

Description

Measurements on 48 rock samples from a petroleum reservoir.

Usage

```
rock
```

Format

A data frame with 48 rows and 4 numeric columns.

```
[,1] area    area of pores space, in pixels out of 256 by 256  
[,2] peri    perimeter in pixels  
[,3] shape   perimeter/sqrt(area)  
[,4] perm    permeability in milli-Darcies
```

Details

Twelve core samples from petroleum reservoirs were sampled by 4 cross-sections. Each core sample was measured for permeability, and each cross-section has total area of pores, total perimeter of pores, and shape.

Source

Data from BP Research, image analysis by Ronit Katz, U. Oxford.

[Package *datasets* version 4.2.2 [Index](#)]

Vì các tập dữ liệu trong thư viện 'datasets' có cấu trúc dữ liệu là dataframe nên bạn có thể khai thác đầy đủ các chức năng của một dataframe với các tập dữ liệu này.

Bài tập

1. Load và xem thông tin tập dữ liệu iris.

XXVII. R Max and Min

Tìm giá trị lớn nhất và nhỏ nhất của một tập dữ liệu

Hàm `max()` và `min()` trong R được dùng để lấy giá trị lớn nhất và nhỏ nhất của một tập hợp.

Cú pháp cơ bản:

`max(data)`

`min(data)`

Trong đó:

- data: Tập dữ liệu cần lấy giá trị lớn nhất hoặc nhỏ nhất.

Example:

```
# Lấy giá trị lớn nhất và của thuộc tính Sepal.Length trong tập dữ liệu iris.  
print(max(iris$Sepal.Length))  
print(min(iris$Sepal.Length))
```

Output:

```
[1] 7.9  
[1] 4.3
```

Tìm vị trí của giá trị lớn nhất và nhỏ nhất của một tập dữ liệu

Để tìm số chỉ mục (vị trí) của giá trị lớn nhất hoặc nhỏ nhất của một tập dữ liệu, bạn có thể dùng hàm `which.max()` và `which.min()`

Example:

```
# Tìm số chỉ mục của giá trị lớn nhất và của thuộc tính Sepal.Length trong tập dữ liệu iris.  
print(which.max(iris$Sepal.Length))  
print(which.min(iris$Sepal.Length))
```

Output:

```
[1] 132  
[1] 14
```

Tìm hàng của giá trị lớn nhất và nhỏ nhất của một tập dữ liệu

Để tìm tên hàng của giá trị lớn nhất hoặc nhỏ nhất của một tập dữ liệu, bạn có thể kết hợp hàm `rownames()` và `which.max()` hoặc `which.min()`. Trước tiên, hàm `rownames()` được dùng để lấy ra một vector bao gồm tên các hàng trong tập dữ liệu. Sau đó, kết hợp số chỉ mục được lấy ở hàm `which.max()` hoặc `which.min()` để lấy ra tên của hàng cần tìm.

Example:

```
row_names_arr <- rownames(iris)
```

```
# Tìm tên hàng của giá trị lớn nhất và của thuộc tính Sepal.Length trong tập dữ liệu iris.
```

```
row_names_arr[which.max(iris$Sepal.Length)]  
row_names_arr[which.min(iris$Sepal.Length)]
```

Output:

```
[1] "132"  
[1] "14"
```

Thông thường, tên hàng của một tập dữ liệu cũng chính là số chỉ mục của tập dữ liệu đó nên trường hợp này sẽ ít khi xuất hiện hơn.

Bài tập

1. Cho tập dữ liệu: {195, 195, 187, 168}. Tạo một dataframe có tên cột là 'height' chứa tập dữ liệu đó.
2. Đổi tên chỉ mục hàng lần lượt thành Haland, Weghorst, Ronaldo, Messi.
3. Tìm giá trị lớn nhất và nhỏ nhất của cột height.
4. Tìm tên hàng của giá trị lớn nhất và nhỏ nhất của cột height.

XXVIII. R Mean, Median and Mode

Mean

Mean trong R là giá trị trung bình của một tập hợp số. Để tính giá trị đó, ta có thể sử dụng hàm `mean()` trong R. Bản chất, nó bằng tổng tất cả các giá trị trong tập hợp chia cho tổng số lượng của chúng.

Cú pháp cơ bản:

```
mean(x, na.rm = FALSE)
```

Trong đó:

- x: Tập dữ liệu.
- na.rm: Mặc định là FALSE. Nếu TRUE, bỏ qua các giá trị NA trước khi tính toán.

Example:

```
vec <- c(1, 2, 3, 2.5, 2, 3.7)
```

```
# Find the average value of this vector.  
mean(vec)
```

Output:

[1] 2.366667

Median

Median trong R là giá trị ở giữa của một tập hợp sau khi sắp xếp. Nếu kích thước của một tập hợp là lẻ, giá trị median chính bằng giá trị có số chỉ mục ở giữa của tập hợp. Nếu kích thước của một tập hợp là chẵn, giá trị median chính bằng giá trị trung bình của 2 giá trị có số chỉ mục ở giữa của tập hợp.

Cú pháp cơ bản:

```
median(x, na.rm = FALSE)
```

Trong đó:

- x: Tập dữ liệu.
- na.rm: Mặc định là FALSE. Nếu TRUE, bỏ qua các giá trị NA trước khi tính toán.

Example:

```
vec <- c(1, 2, 3, 2.5, 2, 3.7)
```

```
# Find the median value of this vector.
```

```
median(vec)
```

Output:

```
[1] 2.25
```

Mode

Giá trị mode trong R là giá trị có số lần xuất hiện nhiều nhất trong một tập hợp. Để tính giá trị mode trong R, bạn có thể làm theo những bước sau:

- Bước 1: Sắp xếp tập hợp theo tần suất xuất hiện của các phần tử giảm dần.
- Bước 2: Dùng hàm `names()` để lấy giá trị đầu tiên của tập hợp vừa được sắp xếp.
- Bước 3: Chuyển giá trị vừa lấy thành dạng numeric bằng hàm `as.numeric()`.

Example:

```
vec <- c(1, 2, 3, 2.5, 2, 3.7)
```

```
# Find the mode value of this vector.
```

```
as.numeric(names(sort(-table(vec)))[1])
```

Output:

```
[1] 2
```

Bài tập

1. Cho tập dữ liệu: `weight <- c(45, 60, 47, 85, 80, 80, 79, 78, 66)`.
Tìm giá trị mean, median và mode của tập dữ liệu đã cho.

XXIX. Permutations and Combinations

Permutations

Hoán vị là cách sắp xếp n phần tử trên n vị trí được định sẵn. Ví dụ như: Sắp xếp 10 học sinh thành một hàng dọc.

Để có thể thực thi, tính toán hoán vị trong R, bạn có thể dùng hàm `permn()` trong gói thư viện `combinat()`.

Cú pháp cơ bản:

```
permn(x)
```

Trong đó:

- `x`: Tập dữ liệu được hoán vị.

Ví dụ:

```
library(combinat)
```

```
players <- c('Ronaldo', 'Messi', 'Benzema')
```

```
# Permutes the 'players' data.
```

```
perm_players <- permn(players)
```

```
perm_players
```

```
# The number of permutation's ways.
```

```
cat('Quantity: \n')
```

```
length(perm_players)
```

Output

```
[[1]]
```

```
[1] "Ronaldo" "Messi"  "Benzema"
```



```
[[2]]
```

```
[1] "Ronaldo" "Benzema" "Messi"
```

```
[[3]]
```

```
[1] "Benzema" "Ronaldo" "Messi"
```

```
[[4]]
```

```
[1] "Benzema" "Messi" "Ronaldo"
```

```
[[5]]
```

```
[1] "Messi" "Benzema" "Ronaldo"
```

```
[[6]]
```

```
[1] "Messi" "Ronaldo" "Benzema"
```

Quantity:

```
[1] 6
```

Combinations

Tổ hợp chập k của n phần tử là số cách chọn k phần tử trong n phần tử mà không quan tâm thứ tự của các phần tử. Ví dụ: Chọn 3 viên bi trong 10 viên bi.

Để có thể thực thi, tính toán hoán vị trong R, bạn có thể dùng hàm `combn()` trong gói thư viện `combinat()`.

Cú pháp cơ bản:

`combn(x, m)`

Trong đó:

- x: Tập dữ liệu ban đầu.
- m: Số lượng phần tử cần chọn.

Ví dụ:

```
library(combinat)
```

```
players <- c('Ronaldo', 'Messi', 'Benzema')
```

```
# Choose two players.
```

```
combn_players <- combn(players, m = 2)
```

```
combn_players
```

```
# The number of combination's ways.
```

```
cat('Quantity: \n')
```

```
ncol(combn_players)
```

Output

```
[,1] [,2] [,3]
```

```
[1,] "Ronaldo" "Ronaldo" "Messi"
```

```
[2,] "Messi" "Benzema" "Benzema"
```

```
Quantity:
```

```
[1] 3
```

Bài tập

1. Cho tập dữ liệu: `company <- ('Google', 'Vinfast', 'Samsung', 'Tesla')`.
 - a. Tìm các hoán vị của tập dữ liệu trên.
 - b. Tìm các cách chọn 3 tập đoàn trong số các tập đoàn trên.

XXX. Linear Regression

Định nghĩa

Hồi quy tuyến tính là mô hình quan hệ tuyến tính giữa 2 biến: biến độc lập và biến phụ thuộc từ một tập dữ liệu cho trước. Từ mỗi quan hệ đó, chúng ta có thể đoán được giá trị của biến phụ thuộc dựa trên các biến độc lập trong tương lai. Ví dụ: dự đoán giá trị cổ phiếu của một doanh nghiệp dựa trên doanh thu, lợi nhuận, tình hình kinh doanh và các yếu tố tài chính khác.

Về mặt toán học, mỗi quan hệ giữa 2 biến trong hồi quy tuyến tính có dạng:

$$y = ax + b$$

Trong đó:

- y: Biến phụ thuộc.
- x: Biến độc lập.
- a, b: Các hằng số.

Các bước thực hiện

Để áp dụng thuật toán hồi quy tuyến tính trong R cũng như trong các ngôn ngữ lập trình khác, bạn hãy làm theo những bước sau:

- Bước 1: Thu thập và xử lý dữ liệu về các biến độc lập và phụ thuộc cho mục đích của mình. Ví dụ: Bạn cần dự đoán giá trị cổ phiếu thì hãy thu thập các dữ liệu về các chỉ số tài chính và giá trị cổ phiếu của các doanh nghiệp.
- Bước 2: Xây dựng mô hình hồi quy tuyến tính. Trong R, hàm `lm()` được sử dụng để giúp bạn làm điều đó.
- Bước 3: Đánh giá mô hình đã xây dựng để kiểm tra độ chính xác của nó bằng các giá trị R-squared, Mean Square Error(MSE), ...
- Bước 4: Áp dụng mô hình để dự đoán các giá trị cần tìm. Trong R, hàm `predict()` được sử dụng cho nhiệm vụ này.

Các hàm cần dùng trong R

Để thực hiện hoàn chỉnh mô hình hồi quy tuyến tính trong R, ít nhất bạn cần 2 hàm: `lm()` và `predict()`.

Hàm `lm()` được sử dụng để xây dựng mô hình hồi quy tuyến tính.

Cú pháp cơ bản:

`lm(formula, data)`

Trong đó:

- `formula`: Mối quan hệ giữa biến phụ thuộc và biến độc lập.
- `data`: Dữ liệu cho hàm `lm()`.

Hàm `predict()` được sử dụng để dự đoán giá trị phụ thuộc từ các giá trị độc lập dựa trên một mô hình cho trước.

Cú pháp cơ bản:

`predict(object, newdata)`

Trong đó:

- `object`: Mô hình (mối quan hệ giữa các biến).
- `newdata`: Dữ liệu cho việc dự đoán giá trị cần tìm.

Bên cạnh 2 hàm trên, để kiểm tra độ chính xác của mô hình đã xây dựng, bạn có thể dựa trên các giá trị R-squared, Mean Square Error (MSE), ... Ở đây, chúng tôi sẽ mô tả cách tính giá trị R-squared trong R.

```
r_square = function (y_test, y_pred){  
  cor(y_test, y_pred)^2  
}
```

Ví dụ

Trong ví dụ này, chúng tôi sẽ xây dựng mô hình hồi quy tuyến tính tính giá trị speed dựa trên giá trị biến dist trong tập dữ liệu cars.

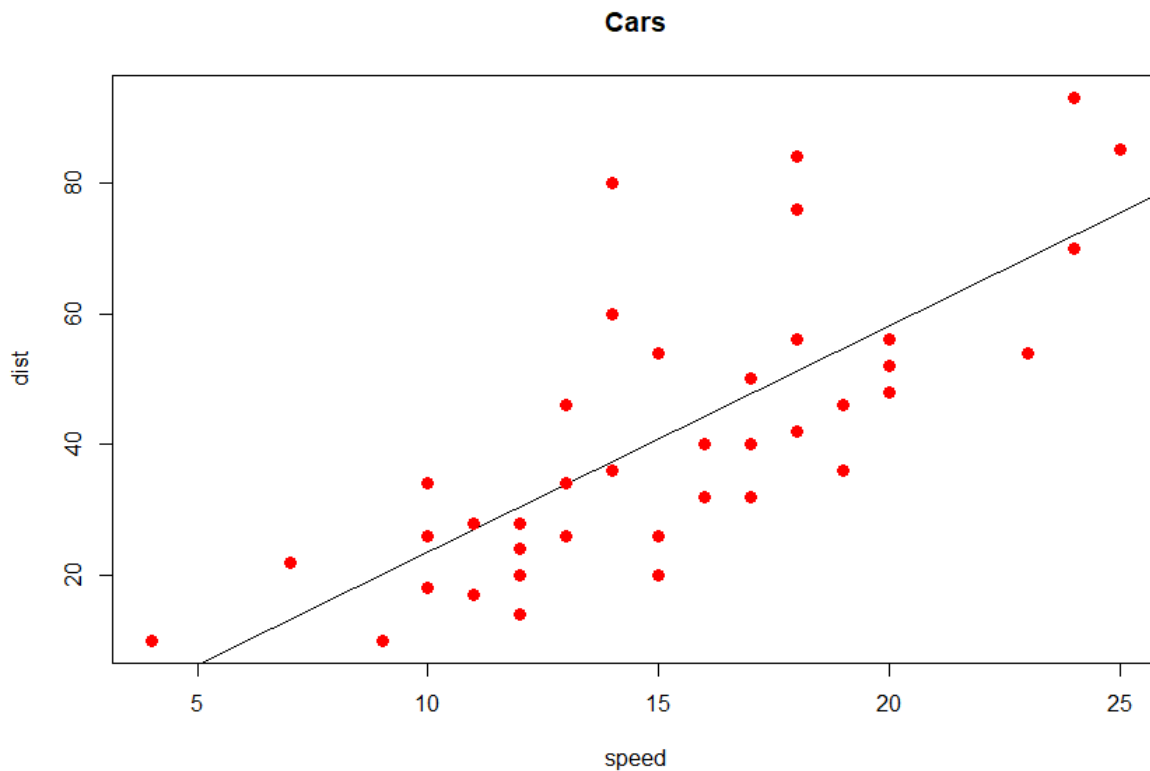
Đánh giá dữ liệu

Trước tiên, chúng ta cần đánh giá dữ liệu xem có phù hợp để sử dụng mô hình hồi quy tuyến tính hay không thông qua đồ thị. Nếu các điểm trong tập dữ liệu phân bố gần giống như một đường thẳng thì bạn có thể dùng mô hình hồi quy tuyến tính. Nếu không, bạn nên dùng mô hình khác dựa trên kinh nghiệm của bản thân để tránh sai số.

```
# Plot the chart.
```

```
plot(x, y, col = "red",main = "Cars", xlab = "speed",ylab = "dist",  
abline(lm(y~x)),cex = 1.3,pch = 16)
```

Output:



Các điểm dữ liệu trong đồ thị phân bố cách không gần đường thẳng của mô hình. Trong thực tế, với các trường hợp như này, bạn không nên sử dụng mô hình hồi quy tuyến tính.

Chuẩn bị dữ liệu để xây dựng mô hình

Ở bước này, chúng ta sẽ chia tập dữ liệu ra 2 phần. Một phần để xây dựng mô hình, phần còn lại để đánh giá mô hình

```
# Shuffle the 'cars' data.
```

```
cars = cars[sample(1:nrow(cars)), ]
```

```
# Data training.
```

```
x <- cars$speed[1:40]
```

```
y <- cars$dist[1:40]
```

Xây dựng mô hình hồi quy tuyến tính.

```
# Build the linear regression model.
linear_relation <- lm(y~x)
linear_relation
```

Output:

Call:

```
lm(formula = y ~ x)
```

Coefficients:

```
(Intercept)      x
   -15.552    3.637
```

Đánh giá mô hình

```
# Find the predicted value.
```

```
y_pred <- predict(linear_relation, data.frame(x = X_test))
```

```
# The function to find the R squared error metric.
```

```
r_square = function (y_test, y_pred){
  cor(y_test, y_pred)^2
}
```

```
r_squared_value <- as.character(r_square(y_test, y_pred))
paste('R-squared: ', r_squared_value)
```

Output:

```
[1] "R-squared: 0.551212685126128"
```

Áp dụng mô hình

Sau khi đã xây dựng thành công mô hình, bạn có thể áp dụng nó bằng hàm `predict()`.

Example:

```
# Find the stopping distance of a car with the current speed is 30.
```

```
y_pred <- predict(linear_relation, data.frame(x = 30))
y_pred
```

Output:

Bài tập

1. Cho tập dữ liệu về chiều cao và cân nặng của các học sinh trong một lớp học.

Student	Height (in cms)	Weight (in lbs)
1	162	165
2	190	150
3	183	225
4	172	160
5	161	145
6	168	129
7	163	147
8	180	152
9	172	150
10	172.72	138
11	182	200
12	172	200
13	178	155
14	170	145
15	160	155
16	185	135
17	181	152
18	177	160
19	162.56	130
20	173	174
21	175	170
22	160	155
24	191	210
25	178	140

- a. Hãy vẽ đồ thị và đánh giá xem có thể sử dụng mô hình hồi quy tuyến tính cho tập dữ liệu này hay không?
- b. Xây dựng và đánh giá mô hình hồi quy tuyến tính cho tập dữ liệu ở trên.
- c. Nếu sinh viên 26 có chiều cao là 205 cms thì cân nặng dự đoán theo mô hình hồi quy tuyến tính của sinh viên đó là bao nhiêu.

XXXI. Multiple Regression

Định nghĩa

Cũng giống như hồi quy tuyến tính, hồi quy bội số là mô hình quan hệ giữa biến độc lập và biến phụ thuộc. Nhưng hồi quy tuyến tính chỉ có một biến độc lập còn hồi quy bội số thì có nhiều biến độc lập.

Về mặt toán học, mối quan hệ giữa 2 biến trong hồi quy bội số có dạng:

$$y = a_0 + a_1x_1 + a_2x_2 + a_3x_3 + \dots + a_nx_n$$

Trong đó:

- y : Biến phụ thuộc.
- x_i : Các biến độc lập.
- a_i : Các hằng số.

Về cách áp dụng thì nó khá tương tự với mô hình hồi quy tuyến tính.

Các hàm cần dùng cũng là hàm `lm()` và `predict()` nhưng hàm `lm()` sẽ được sử dụng với ý nghĩa khác. Bạn có thể tìm hiểu hơn về cách sử dụng trong trường hợp mô hình hồi quy bội số bằng ví dụ sau dưới đây.

Ví dụ

Ở ví dụ này, chúng tôi sẽ sử dụng tập dữ liệu rock để tìm độ thấm (perm) dựa trên các thông số kích thước của khối đá.

```
rock <- rock[,c('area', 'peri', 'perm')]  
head(rock)
```

Output:

```
   area  peri perm  
48 9718 1485.58 580.0  
30 4895 1239.66 890.0  
14 8868 3518.04  82.4  
12 8624 3986.24 119.0  
24 8233 3518.44 142.0  
33 5980 1426.76 950.0
```

Phân tích:

- Biến phụ thuộc: perm
- Các biến độc lập: area, peri

Code sample:

Xây dựng mô hình

```
# Data training.
```



```
x1 <- rock$area
x2 <- rock$peri
y <- rock$perm

# Build the multiple regression model.
mul_regr_model <- lm(y~x1+x2)
mul_regr_model
```

Output:

Call:

```
lm(formula = y ~ x1 + x2)
```

Coefficients:

(Intercept)	x1	x2
696.6883	0.1064	-0.3899

Áp dụng

```
# Find the permeability when the area is 7447 and the perimeter is 3800.
predict(mul_regr_model, data.frame( x1= 7447, x2 = 3800))
```

Output:

1

7.156464

Bài tập

1. Cho tập dữ liệu trees. Trong đó biến phụ thuộc là Volume và các biến độc lập là Girth, Height. Hãy xây dựng mô hình hồi quy cho tập dữ liệu trên. Sau đó hãy tìm Volume với Height = 90, Girth = 21.

XXXII. Logistic Regression

Định nghĩa

Hồi quy logistic là một mô hình học máy có giám sát nhằm để phân loại biến phụ thuộc dựa trên các biến độc lập. Ở mô hình hồi quy tuyến tính và hồi quy bội số ở trên, giá trị biến phụ thuộc cần tìm là giá trị liên tục. Trong khi đó, ở mô hình hồi quy logistic, giá trị biến phụ thuộc cần tìm lại là giá trị rời rạc. Ví dụ như giá trị 0/1/2, Đúng/Sai. Mô hình này được ứng dụng rộng rãi để làm việc với các dữ liệu cần phân loại như là: kiểm tra sim rác, phân loại bệnh nhân, nhận diện đối tượng, ...

Về mặt toán học, mối quan hệ giữa 2 biến trong hồi quy logistic có dạng:

$$y = 1/(1 + e^{-(a_0 + a_1x_1 + a_2x_2 + a_3x_3 + \dots + a_nx_n)})$$

Trong đó:

- y : Biến phụ thuộc.
- x_i : Các biến độc lập.
- a_i : Các hằng số.

Cách áp dụng của mô hình hồi quy logistic cũng giống với đa số mô hình hồi quy khác nhưng nó được thực hiện bởi hàm `glm()`:

Cú pháp cơ bản:

`glm(formula, data, family)`

Trong đó:

- `formula`: Mối quan hệ giữa biến phụ thuộc và các biến độc lập.
- `data`: Dữ liệu cho việc xây dựng mô hình.
- `family`: Loại mô hình. Với hồi quy logistic thì nó là: `binomial`.

Các bước thực hiện

Để áp dụng thuật toán hồi quy logistic trong R cũng như trong các ngôn ngữ lập trình khác, bạn hãy làm theo những bước sau:

- Bước 1: Chia dữ liệu huấn luyện ra thành 2 phần. Một phần để tạo mô hình hồi quy logistic, phần còn lại để kiểm tra độ chính xác của mô hình (Có thể dùng hàm `sample.split()` của thư viện `caTools` trong R)
- Bước 2: Xây dựng mô hình hồi quy logistic (trong R sử dụng hàm `glm()`)
- Bước 3: Đánh giá mô hình đã xây dựng để kiểm tra độ chính xác của nó bằng các giá trị `accuracy_score` hoặc `f1_score`.
- Bước 4: Áp dụng mô hình để dự đoán các giá trị thực nghiệm cần tìm.

Ví dụ

Trong ví dụ này, chúng tôi sẽ sử dụng tập dữ liệu `mtcars` với biến phụ thuộc là `vs` (Động cơ) và các biến độc lập: `cyl` (Số lượng xi-lanh), `hp` (Tổng mã lực), `wt` (Trọng lượng). Bài toán đặt ra là phân loại động cơ dựa trên số lượng xi-lanh, tổng mã lực và trọng lượng của ô tô.

```
head(mtcars)
```

Output:

```

mpg cyl disp hp drat wt qsec vs am gear carb
Mazda RX4      21.0  6 160 110 3.90 2.620 16.46 0 1 4 4
Mazda RX4 Wag  21.0  6 160 110 3.90 2.875 17.02 0 1 4 4
Datsun 710     22.8  4 108  93 3.85 2.320 18.61 1 1 4 1
Hornet 4 Drive 21.4  6 258 110 3.08 3.215 19.44 1 0 3 1
Hornet Sportabout 18.7  8 360 175 3.15 3.440 17.02 0 0 3 2
Valiant        18.1  6 225 105 2.76 3.460 20.22 1 0 3 1

```

Chuẩn bị dữ liệu

Ở bước này, chúng tôi sẽ chia dữ liệu ra làm 2 phần. Một phần để tạo mô hình hồi quy logistic, phần còn lại để kiểm tra độ chính xác của mô hình bằng hàm `sample.split()` trong thư viện `caTools`.

```
# Split the data.
```

```
split_data <- caTools:: sample.split(mtcars, SplitRatio = 0.9)
```

```
# Data for building the model.
```

```
training_data <- subset(mtcars, split_data == "TRUE")
```

```
dim(training_data)
```

```
# Data for checking the effectiveness of the model.
```

```
testing_data <- subset(mtcars, split_data == "FALSE")
```

```
dim(testing_data)
```

Output:

```
[1] 26 11
```

```
[1] 6 11
```

Xây dựng mô hình

Chúng ta sẽ xây dựng mô hình biến 'vs' phụ thuộc vào biến 'cyl', 'hp', và 'wt' bằng hàm `glm()`.

```
# Building the logistic regression model.
```

```
logistic_regr_model <- glm(vs ~ cyl + hp + wt, data = training_data, family =
"binomial")
```

```
logistic_regr_model
```

Output:

```
Call: glm(formula = vs ~ cyl + hp + wt, family = "binomial", data = training_data)
```

Coefficients:

(Intercept)	cyl	hp	wt
82.609	-41.506	-3.971	198.317

Degrees of Freedom: 25 Total (i.e. Null); 22 Residual

Null Deviance: 35.43

Residual Deviance: 6.828e-09 AIC: 8

Đánh giá mô hình

Chúng ta sẽ dự đoán giá trị của biến 'vs' dựa trên mô hình vừa tạo bằng hàm predict(). Sau đó, chúng ta cần phải chuyển đổi giá trị trả về của hàm predict() thành các giá trị rời rạc. Cuối cùng, bạn có thể dùng giá trị accuracy_score hoặc f1_score để đánh giá mô hình. Ở ví dụ này, chúng tôi sẽ dùng giá trị accuracy_score.

```
# Predict the 'vs' attributes.
```

```
y_pred <- predict(logistic_regr_model,  
                  testing_data, type = "response")
```

```
# Transform to the sporadic values.
```

```
vs_pred <- ifelse(y_pred > 0.5, 1, 0)
```

```
cat('Predicted result: \n')
```

```
data.frame(vs_pred)
```

```
accuracy_score <- mean(vs_pred != testing_data$vs)
```

```
print(paste('Accuracy_score: ', 1 - accuracy_score))
```

Output:

Predicted result:

	vs_pred
Datsun 710	1
Hornet Sportabout	0
Merc 450SLC	0
Lincoln Continental	0
Pontiac Firebird	0
Porsche 914-2	1

[1] "Accuracy_score: 0.8333333333333333"

Bài tập

1. Tạo mô hình hồi quy tuyến tính cho tập dữ liệu iris. Biến phụ thuộc là Species, các biến còn lại là biến độc lập.

Yêu cầu:

- Chia tập dữ liệu theo tỉ lệ dữ liệu huấn luyện : kiểm tra là 9:1
- Xây dựng và đánh giá mô hình theo giá trị accuracy_score

Chú ý: Cần chuyển đổi các giá trị character của thuộc tính Species thành các giá trị numeric.

XXXIII. Decision Tree

Định nghĩa

Cây quyết định là một trong những thuật toán học máy được sử dụng khá rộng rãi hiện nay. Nó được sử dụng trong bài hồi quy và rộng rãi hơn với bài toán phân lớp. Cũng giống như hồi quy logistic, cây quyết định được dùng để phân loại đối tượng ví dụ như: kiểm tra sim rác, phân loại bệnh nhân, dự đoán thời tiết, ...

Cây quyết định được biểu diễn dưới đồ thị dạng cây mà mỗi nút biểu diễn một đặc trưng, mỗi nhánh biểu diễn một quy luật và mỗi lá biểu diễn một kết quả.

Cách thực hiện trong R

Để làm việc với cây quyết định trong R, bạn cần dùng hàm `ctree()` trong thư viện `party`.

Vì vậy, trước hết, bạn cần cài đặt thư viện `party` để bắt đầu làm việc với cây quyết định trong R bằng câu lệnh sau:

```
install.packages('party')
```

Cú pháp cơ bản của hàm `ctree()`:

```
ctree(formula, data)
```

Trong đó:

- formula: Biểu thức thể hiện mối quan hệ giữa biến phụ thuộc và các biến độc lập.
- data: Tập dữ liệu huấn luyện.

Về các bước thực hiện trong R, nó khá tương tự với hồi quy logistic, bạn chỉ cần thay đổi hàm glm() bằng hàm ctree().

Ví dụ

Ở ví dụ này, chúng tôi sẽ sử dụng tập dữ liệu iris. Biến phụ thuộc là Species và các biến còn lại là biến độc lập. Bài toán đặt ra là: phân loại hoa diên vĩ theo các chỉ số độ dài, độ rộng của lá đài và cánh hoa.

head(iris)

Output:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

Chuẩn bị dữ liệu

Trong ví dụ này, chúng tôi sẽ tách tập dữ liệu iris ra làm 2 phần bằng thư viện caTools. Một phần để tạo mô hình cây quyết định, phần còn lại để kiểm tra độ chính xác của mô hình.

```
# Split the data.
```

```
split_data <- caTools:: sample.split(iris, SplitRatio = 0.8)
```

```
# Data for building the model.
```

```
training_data <- subset(iris, split_data == "TRUE")
```

```
dim(training_data)
```

```
# Data for checking the effectiveness of the model.
```

```
testing_data <- subset(iris, split_data == "FALSE")
```

```
dim(testing_data)
```

Output:

```
[1] 120 5
```

```
[1] 30 5
```

Xây dựng mô hình

Chúng ta sẽ xây dựng mô hình cây quyết định bằng hàm `ctree()` trong thư viện `party`.

```
library(party)
```

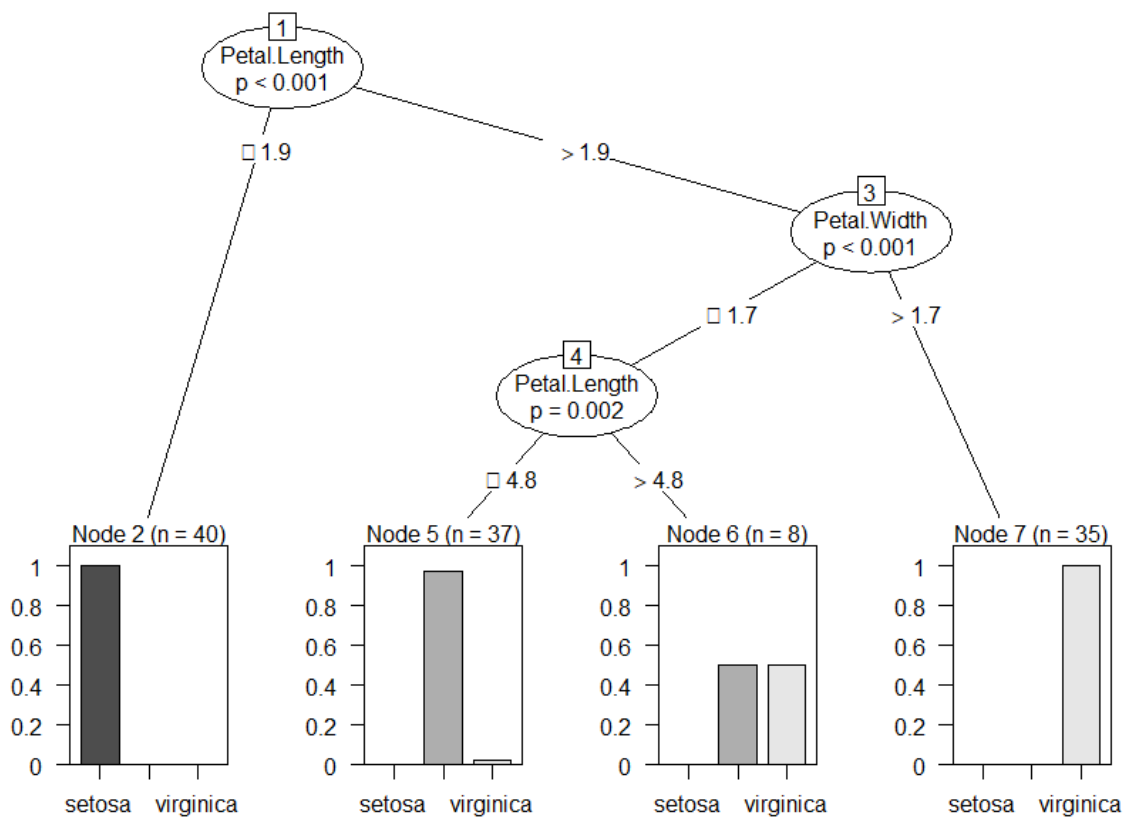
```
# Build the decision tree model.
```

```
ctree_model <- ctree(Species ~ ., training_data)
```

```
# Plot the model.
```

```
plot(ctree_model)
```

Output:



Đánh giá mô hình

Sau khi đã xây dựng mô hình thành công, chúng ta sẽ dự đoán thuộc tính Species trên dữ liệu kiểm tra. Và so sánh kết quả thu được với kết quả thực tế để đánh giá mô hình đã tạo.

```
# Predicted data.
```

```
y_pred <- predict(ctree_model, testing_data)
data.frame(y_pred)
```

```
accuracy_score <- mean(y_pred!= testing_data$Species)
print(paste('Accuracy_score: ', 1 - accuracy_score))
```

Output:

```
   y_pred
1   setosa
2   setosa
3   setosa
4   setosa
5   setosa
6   setosa
7   setosa
8   setosa
9   setosa
10  setosa
11 versicolor
12 versicolor
13 versicolor
14 versicolor
15  virginica
16 versicolor
17 versicolor
18 versicolor
19 versicolor
20 versicolor
21  virginica
22  virginica
23  virginica
24  virginica
25  virginica
26  virginica
27  virginica
```


28 virginica
29 virginica
30 virginica

[1] "Accuracy_score: 0.9666666666666667"

Bài tập

1. Xây dựng mô hình thuật toán cây quyết định cho tập dữ liệu readingSkills với biến phụ thuộc là nativeSpeaker và các biến còn lại là biến độc lập. Chia tập dữ liệu ngẫu nhiên theo tỷ lệ dữ liệu huấn luyện : dữ liệu kiểm tra là 8:2 và đánh giá mô hình đã tạo.

XXXIV. Random Forest

Định nghĩa

Random forest là một thuật toán học máy có giám sát được sử dụng cho các bài toán phân lớp và hồi quy. Random forest được tạo ra từ nhiều cây quyết định và đưa từng cây quyết định vào mẫu quan sát. Kết quả phổ biến nhất trong tập mẫu quan sát sẽ là kết quả cuối cùng cần tìm. Cũng giống như mô hình hồi quy logistic và cây quyết định, mô hình random forest được áp dụng rộng rãi cho các bài toán phân loại đối tượng như: phân loại hình ảnh, chẩn đoán bệnh, phân loại email, ...

Cách thực hiện trong R

Để sử dụng thuật toán random forest trong R, bạn cần sử dụng hàm randomForest() trong gói thư viện cùng tên.

Trước hết, bạn cần phải cài đặt thư viện randomForest.

```
install.packages('randomForest')
```

Cú pháp cơ bản hàm randomForest():

```
randomForest(formula, data)
```

Trong đó:

- formula: Biểu thức thể hiện mối quan hệ giữa biến phụ thuộc và các biến độc lập.
- data: Tập dữ liệu huấn luyện.

Về các bước thực hiện, random forest khá tương đồng với các thuật toán phân lớp khác.

Ví dụ

Ở ví dụ này, chúng tôi cũng sẽ thực hiện bài toán phân loại hoa diên vĩ trên tập dữ liệu iris được mô tả ở phần cây quyết định. Hãy cùng xem nó có gì khác biệt so với mô hình random forest không nhé!

Chuẩn bị dữ liệu

Chúng tôi sẽ tách tập dữ liệu iris ra làm 2 phần bằng thư viện caTools. Một phần để tạo mô hình random forest, phần còn lại để kiểm tra độ chính xác của mô hình.

```
# Split the data.
```

```
split_data <- caTools:: sample.split(iris, SplitRatio = 0.8)
```

```
# Data for building the model.
```

```
training_data <- subset(iris, split_data == "TRUE")
```

```
dim(training_data)
```

```
# Data for checking the effectiveness of the model.
```

```
testing_data <- subset(iris, split_data == "FALSE")
```

```
dim(testing_data)
```

Output:

```
[1] 120  5
```

```
[1] 30  5
```

Xây dựng mô hình

Chúng ta sẽ xây dựng mô hình cây quyết định bằng hàm randomForest() trong thư viện cùng tên.

```
library(randomForest)
```

```
# Build the random forest model.
```

```
rand_forest_model <- randomForest(Species ~ ., training_data)
```

```
rand_forest_model
```

Output:

Call:

```
randomForest(formula = Species ~ ., data = training_data)
```

Type of random forest: classification

Number of trees: 500

No. of variables tried at each split: 2

OOB estimate of error rate: 5%

Confusion matrix:

	setosa	versicolor	virginica	class.error
setosa	40	0	0	0.000
versicolor	0	37	3	0.075
virginica	0	3	37	0.075

Đánh giá mô hình

Sau khi đã xây dựng mô hình random forest thành công, chúng ta sẽ dự đoán thuộc tính Species trên dữ liệu kiểm tra. Và tính giá trị accuracy_score trên mô hình này.

Split the data.

```
split_data <- caTools:: sample.split(iris, SplitRatio = 0.9)
```

Data for building the model.

```
training_data <- subset(iris, split_data == "TRUE")  
dim(training_data)
```

Data for checking the effectiveness of the model.

```
testing_data <- subset(iris, split_data == "FALSE")  
dim(testing_data)
```

```
library(randomForest)
```

Build the random forest model.

```
rand_forest_model <- randomForest(Species ~ ., training_data)
```

Predicted data.

```
y_pred <- predict(rand_forest_model, testing_data)  
data.frame(y_pred)
```

```
accuracy_score <- mean(y_pred!= testing_data$Species)
print(paste('Accuracy_score: ', 1 - accuracy_score))
```

Output:

```
y_pred
4      setosa
9      setosa
14     setosa
19     setosa
24     setosa
29     setosa
34     setosa
39     setosa
44     setosa
49     setosa
54 versicolor
59 versicolor
64 versicolor
69 versicolor
74 versicolor
79 versicolor
84 virginica
89 versicolor
94 versicolor
99 versicolor
104 virginica
109 virginica
114 virginica
119 virginica
124 virginica
129 virginica
134 virginica
139 virginica
144 virginica
149 virginica
```

```
[1] "Accuracy_score: 0.9666666666666667"
```

Bài tập

1. Xây dựng mô hình thuật toán random forest cho tập dữ liệu readingSkills với biến phụ thuộc là nativeSpeaker và các biến còn lại là biến độc lập. Chia tập dữ liệu ngẫu nhiên theo tỷ lệ dữ liệu huấn luyện : dữ liệu kiểm tra là 9:1 và đánh giá mô hình đã tạo.

XXXV. Time Series Analysis

Định nghĩa

Time series là một tập dữ liệu trong đó mỗi điểm dữ liệu đều được liên kết với một mốc thời gian xác định. Time series được dùng để giúp chúng ta có thể quan sát một đối tượng hoạt động như thế nào trong một khoảng thời gian nhất định. Ví dụ như là: Thống kê giá vàng hàng ngày, thống kê lượng mưa trong năm, thống kê doanh số hàng tháng, ... Từ mô hình đó, chúng ta có thể nhận thấy được xu hướng của đối tượng nhằm để dự đoán tương lai để đưa ra các quyết định phù hợp.

Áp dụng trong R

Trong R, một đối tượng Time Series được tạo ra bởi hàm `ts()`.

Cú pháp cơ bản:

```
ts(data, start, end, frequency)
```

Trong đó:

- data: Một cấu trúc dữ liệu lưu trữ các giá trị quan sát của đối tượng Time Series.
- start: Thời gian bắt đầu quan sát.
- end: Thời gian kết thúc quan sát.
- frequency: Tần suất, chu kỳ của mỗi lần quan sát.

Single Time Series

Trong ví dụ này, chúng tôi sẽ lấy ví dụ phân tích doanh số bán hàng (ô tô) hàng tháng của tập đoàn Vinfast năm 2021.

```
# The number of cars per month that Vinfast sold out in 2021.
```

```
vinfast_sales_2021 <- c(2801, 1718, 2330, 2717, 2855, 3517, 3782, 2310, 3497, 3320, 3829, 3047)
```

```
# Convert this data set to the Time Series object.
```

```
vinfast_2021_ts <- ts(vinfast_sales_2021, start = c(2021, 1), frequency = 12)
```

```
vinfast_2021_ts
```

```
# Plot the graph.
```

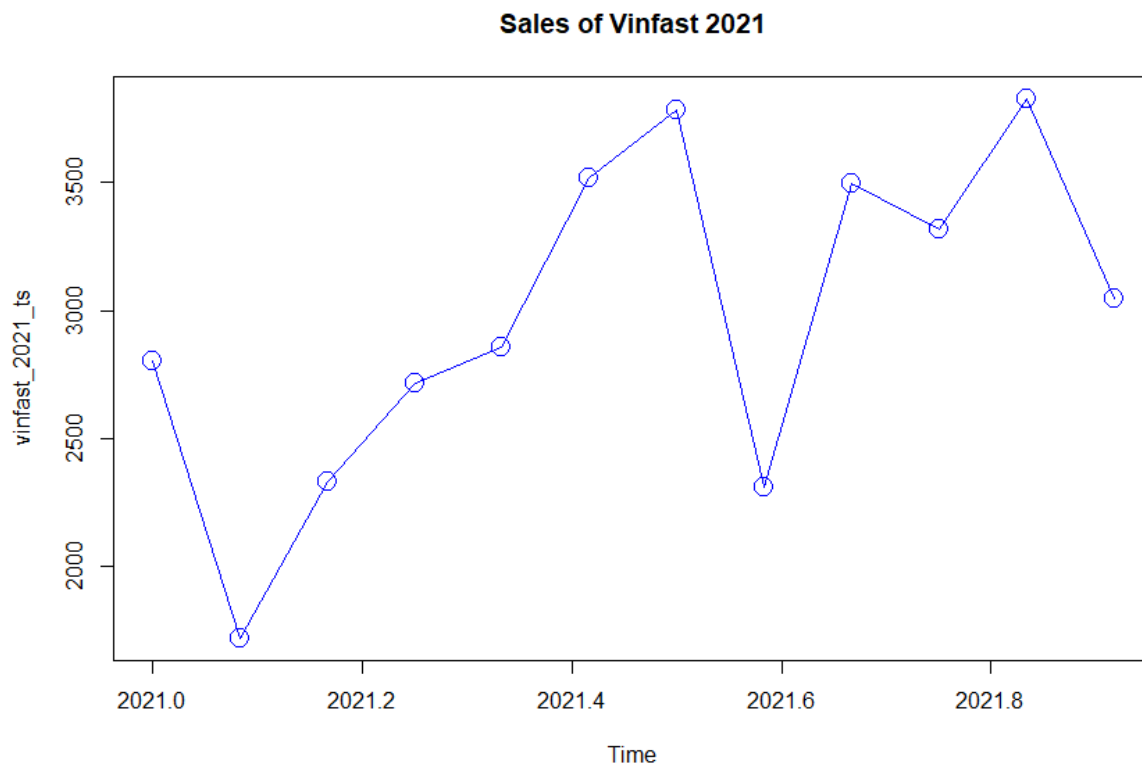
```
cat('Graph: \n')
```

```
plot(vinfast_2021_ts, type = 'o', cex = 2, col = 'blue', main = 'Sales of Vinfast 2021')
```

Output:

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
2021	2801	1718	2330	2717	2855	3517	3782	2310	3497	3320	3829	3047

Graph:



Multiple Time Series

Ngoài ra, bạn cũng có thể tạo ra nhiều quan sát nhiều đối tượng trên cùng một Time Series. Ví dụ: Cũng là doanh số bán hàng của tập đoàn Vinfast năm 2021 nhưng bạn có thể quan sát cùng lúc doanh số ở Hà Nội và TP. Hồ Chí Minh.

```
# The number of cars per month that Vinfast sold out in Ha Noi.
```

```
vinfast_sales_HN <- c(2801, 1718, 2330, 2717, 2855, 3517, 3782, 2310, 3497, 3320, 3829, 3047)
```

```

# The number of cars per month that Vinfast sold out in Ho Chi Minh City.
vinfast_sales_HCM <- c(2103, 1154, 3471, 2427, 3050, 2490, 2137, 1220, 0, 0,
594, 4278)

# Combine two datasets to a matrix.
vinfast_sales <- matrix(c(vinfast_sales_HN, vinfast_sales_HCM), nrow = 12)

# Convert this data set to the Time Series object.
vinfast_ts <- ts(vinfast_sales, start = c(2021, 1), frequency = 12)
vinfast_ts

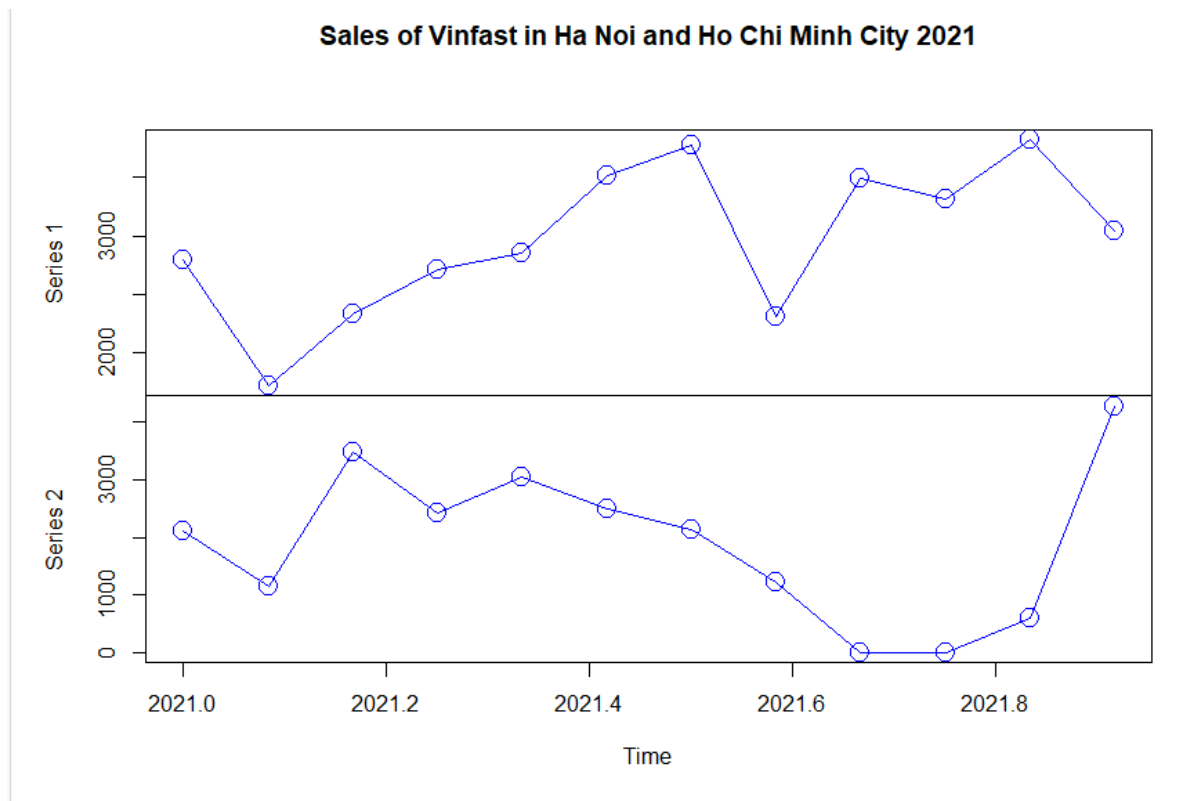
# Plot the graph.
cat('Graph: \n')
plot(vinfast_ts, type = 'o', cex = 2, col = 'blue', main = 'Sales of Vinfast in Ha Noi
and Ho Chi Minh City 2021')

```

Output:

	Series 1	Series 2
Jan 2021	2801	2103
Feb 2021	1718	1154
Mar 2021	2330	3471
Apr 2021	2717	2427
May 2021	2855	3050
Jun 2021	3517	2490
Jul 2021	3782	2137
Aug 2021	2310	1220
Sep 2021	3497	0
Oct 2021	3320	0
Nov 2021	3829	594
Dec 2021	3047	4278

Graph:



Bài Tập

1. Hãy tạo một đối tượng Time Series phân tích giá trị của đồng Bitcoin vào tháng 2 năm 2023. Bạn có thể lấy dữ liệu ở [đây](#).

XXXVI. Survival Analysis

Định nghĩa

Survival analysis là phương pháp thống kê được sử dụng để tìm thời gian xảy ra một sự kiện quan tâm. Ví dụ như phân tích thời gian tồn tại của một đối tượng hay thời gian thất bại của một sự việc. Ngày nay, survival analysis được áp dụng rộng rãi cho các bài toán dự đoán các sự kiện xảy ra trong tương lai như là: phân tích thời gian sống của một bệnh nhân, phân tích lịch sử của một sự kiện, phân tích thời gian khách hàng vỡ nợ, ...

Cách thực hiện trong R

Gói thư viện survival được tích hợp trong R để có thể giúp bạn vận hành mô hình survival analysis.

Để cài đặt gói thư viện survival, bạn có thể thực hiện theo câu lệnh sau:

```
install.packages('survival')
```

Hai hàm chủ đạo được thực hiện để phân tích sự sống trong thư viện survival là Surv() và survfit(). Hàm Surv() để tạo một đối tượng survival. Hàm survfit() để tính toán ước lượng thời gian sinh tồn theo Kaplan-Meier.

Cú pháp cơ bản:

```
Surv(time, event)
```

```
survfit(formula)
```

Trong đó:

- time: Thời gian theo dõi hoặc là thời gian bắt đầu của một sự kiện.
- event: Trạng thái của sự kiện.
- formula: Biểu thức mối quan hệ giữa các biến dự đoán.

Ví dụ

Chuẩn bị dữ liệu

Trong ví dụ này, chúng tôi sẽ lấy tập dữ liệu 'lung' trong gói thư viện 'survival' nhằm để phân tích thời gian sống sót của bệnh nhân mắc ung thư phổi dựa trên các đặc điểm của bệnh nhân.

```
library(survival)
```

```
# The lung data set.
```

```
head(lung)
```

Output:

```
inst time status age sex ph.ecog ph.karno pat.karno meal.cal wt.loss
```

1	3	306	2	74	1	1	90	100	1175	NA
2	3	455	2	68	1	0	90	90	1225	15
3	3	1010	1	56	1	0	90	90	NA	15
4	5	210	2	57	1	1	90	60	1150	11
5	1	883	2	60	1	0	100	90	NA	0
6	12	1022	1	74	1	1	50	80	513	0

Xây dựng mô hình

Trong bài toán này, chúng tôi sẽ phân tích thời gian sống sót của bệnh nhân dựa trên thuộc tính time và status trong thư viện survival. Trước tiên, chúng ta cần tạo một đối tượng survival bằng hàm Surv() và lấy nó làm đầu vào cho hàm survfit() để tạo ra mô hình survival.

```
# Create a survival object with the Surv() function.
```

```
surv <- Surv(lung$time, lung$status)
```

```
# Fit the survival model with the survfit() function.
```

```
survival_model <- survfit(surv ~ 1)
```

```
survival_model
```

```
# Plot the graph.
```

```
plot(survival_model, col = c('blue', 'red', 'black'),
```

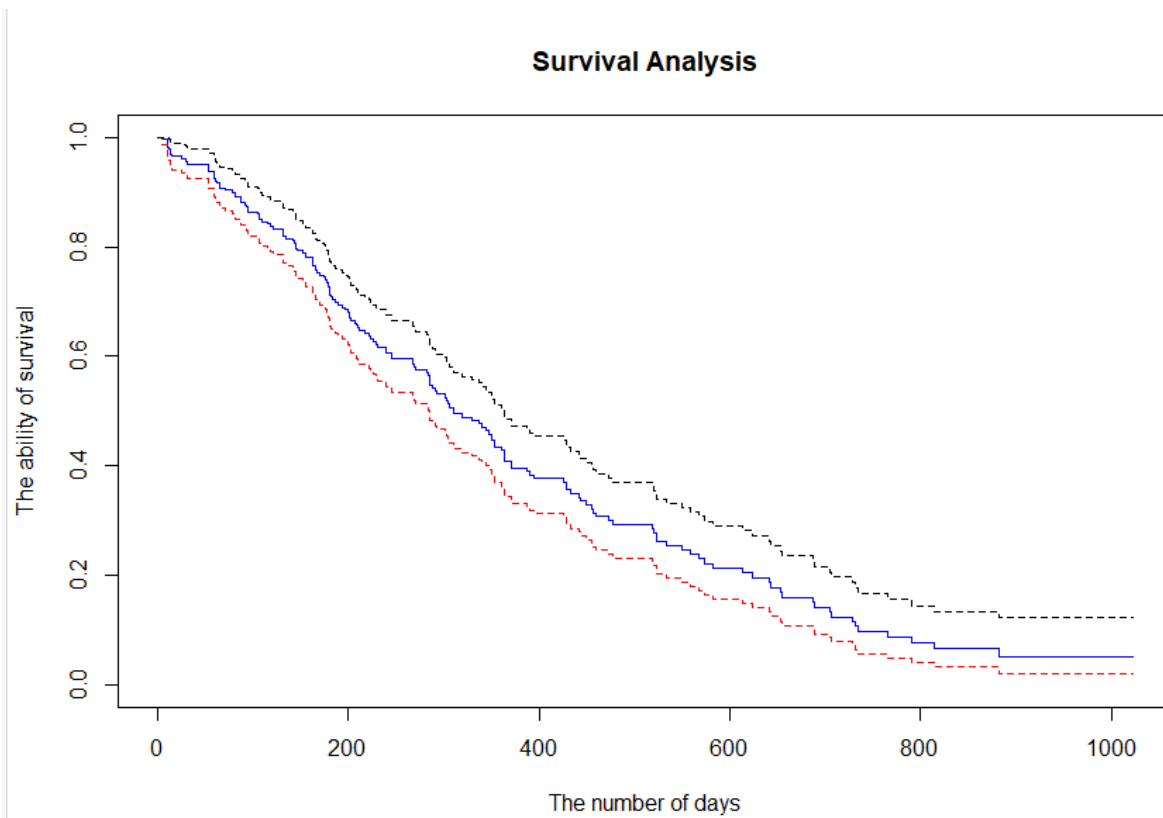
```
     main = 'Survival Analysis', xlab = 'The number of days', ylab = 'The ability of  
survival'
```

```
)
```

Output:

```
Call: survfit(formula = surv ~ 1)
```

```
      n events median 0.95LCL 0.95UCL  
[1,] 228   165   310   285   363
```



Bài tập

1. Xây dựng mô hình survival analysis cho tập dữ liệu 'pbc' trong thư viện 'survival' (Dữ liệu bệnh nhân bị viêm đường mật của phòng khám Mayo) dựa trên chỉ số time và status.