KNU 강원대학교
KANGWON NATIONAL UNIVERSITY

# 8. 연구방법의 적용I. 들어가며

## 정치학연구방법론

박상훈 (sh.park.poli@gmail.com)
강원대학교

# 연구방법의 적용I. 들어가며

## R 프로그래밍이란?

프로그래밍이란? 결국에는 계산기와 다르지 않음. 옛날 주판에서 시작해서 오늘날 컴퓨터가 등장한 것

하지만 분명 다른 점은 있음!

- 고급 연산 가능
- 반복되는 연산 저장 사용: 함수
- 개인 코드를 공유 가능: 패키지

# 연구방법의 적용I. 들어가며

## R 프로그래밍이란?

R은 1991년 뉴질랜드의 University of Auckland의 **R**oss Ihaka와 **R**obert Gentleman 교수 둘이서 기존에 존재하던 S라는 언어를 바탕으로 두 사람 이름의 첫 글자와 S 보다 앞선 **R**이라는 이름의 프로그램을 만들어냄.

**R의 장점: 프로그래머가 아닌 사람들을 위한 언어**

- R 언어가 추구하는 방향성은 바로 '사용자 친화적'(user-friendly)일 것
- 비 프로그래머를 위한 프로그램 언어

**RGUI vs. RStudio: 그저 인터페이스의 차이**

# 연구방법의 적용I. 들어가며

## R 코드 실행방법 2가지

### Interactive Mode (99%)

R 콘솔(Console) 창에서 코드를 돌리는 방법: 명령어를 작성 후 `Enter`

R script에서 코드를 콘솔로 보내는 방법:

- 해당 라인에 커서를 위치 후 `Ctrl + Enter`
- 돌리고 싶은 부분을 블락 지정 후 `Ctrl + Enter`

### Batch Mode (Advanced level)

미리 저장된 `.R` 파일을 윈도우 커맨드라인(cmd)이나 파워쉘(PowerShell)에서 돌리는 방법

# 연구방법의 적용I. 들어가며

## R 세션 실행하기

### 변수 만들기

변수는 우리가 만들고자 하는 것(object)가 들어갈 수 있는 상자

- 변수에 어떠한 값일 할당(assignment)할 때, R에서는 <- 오퍼레이터를 사용

R에서는 상자의 크기가 자유자재로 늘어날 수 있음.

```
x <- 1
x
```

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

# 연구방법의 적용I. 들어가며

## R에서의 데이터

데이터의 유형은 다양하지만, 주로 다루게 될 것은 데이터셋(data set)

- 행(row)과 열(column)으로 이루어짐
  - 행: 개별 관측치(observation)
  - 열: 변수(variable)
- 간단하게 표(table) 혹은 엑셀 스프레드 시트를 생각하면 됨

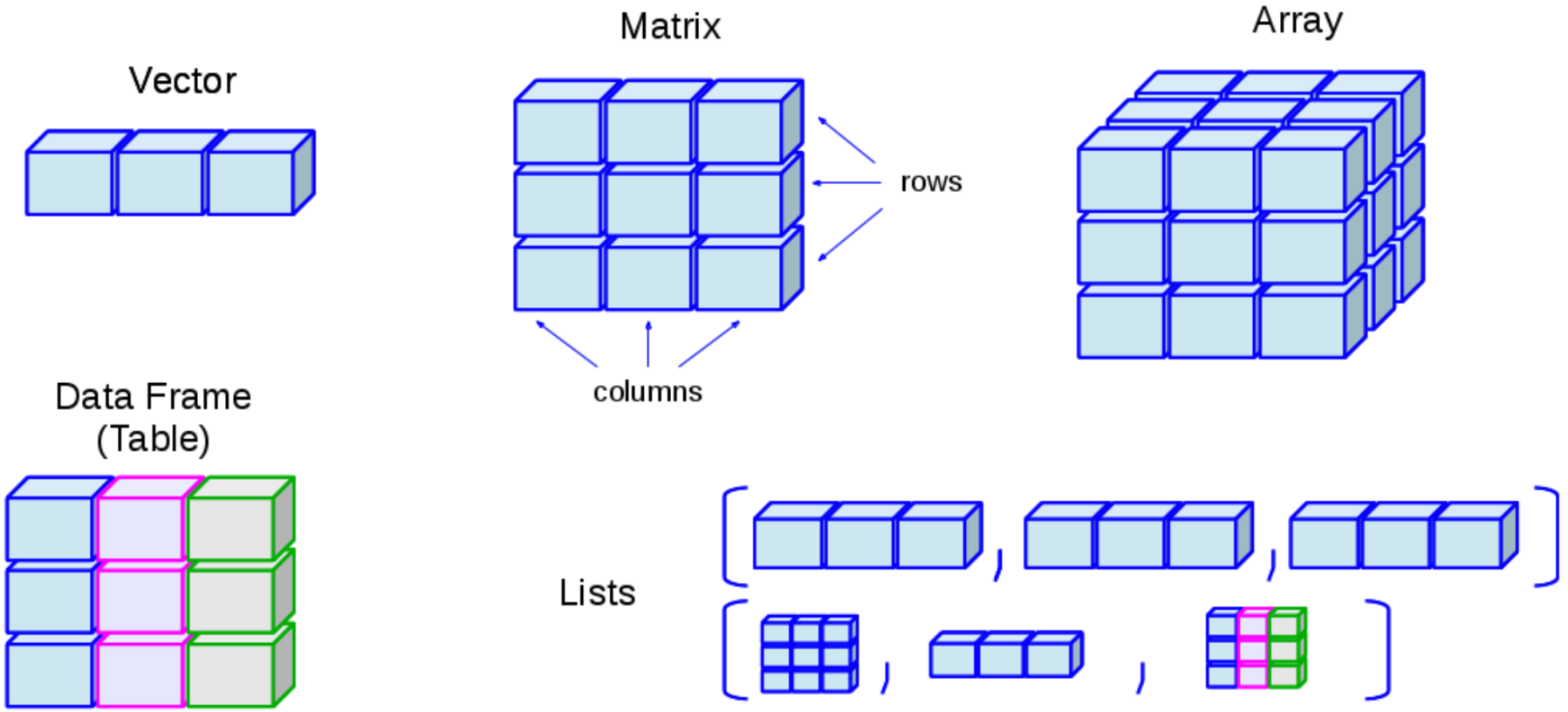
# 연구방법의 적용I. 들어가며

## R에서의 데이터

- 데이터의 종류: 벡터, 행렬, 배열(array), 리스트, 데이터프레임(or 티블)
- 데이터의 유형: 숫자형, 문자형, 논리형(TRUE or FALSE)
    - 단일형 데이터: 한 가지 유형만 가지고 있는 데이터
        - 다중형 데이터: 여러 유형의 데이터로 구성된 데이터

# 연구방법의 적용I. 들어가며

## R에서의 데이터




Source: http://venus.ifca.unican.es

# 연구방법의 적용I. 들어가며

## 데이터의 유형

### 숫자형(numeric)

- 실수형(double)이라고도 하며, 이 중에 소수점이 없는 정수형(integer)는 따로 분류할 수 있음

### 문자형(character)

- " "로 감싸서 표현. 숫자지만 따옴표로 감쌀 경우에는 문자형으로 인식. 단, 숫자를 따옴표를 써서 문자형으로 만든 경우, `as.numeric()`을 이용하여 숫자로 재변환 가능
- 반면, 일반적인 문자형을 숫자형으로 바꿀 수는 없음
    - 데이터 유형이 동일한 단일형 데이터의 경우, 주의하지 않으면 숫자형으로 생각한 변수가 문자형으로 불려올 수 있음

# 연구방법의 적용I. 들어가며

## 데이터의 유형

### 논리형(logical)

- TRUE 혹은 FALSE로 구성된 데이터 유형. TRUE 의 값은 1, FALSE 는 0과 같이 인식됨

### 범주형(categorical)

- 요인형(factor)이라고도 함
- 범주 간 순위(ranking) 혹은 수준(level)이 존재할 수 있고, 존재하지 않을 수도 있음

```
## 수준이 존재하지 않는 명목형 자료를 만들 때
factor(범주화할 자료, labels = c("범주1", "범주2"))

## 수준이 존재하는 명목형 자료(순위형)를 만들 때
factor(범주화할 자료, labels = c("범주1", "범주2"),
       levels = c("범주1", "범주2"))
```

# 연구방법의 적용I. 들어가며

## 벡터(Vector)

여러 개의 원소들을 묶어서 하나의 객체처럼 만들 수 있는 방법

```
c(1, 2, 3, 4)
```

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

```
c("하나", "둘", "셋", "넷")
```

```
## [1] "하나" "둘"   "셋"   "넷"
```

이때, c는 영어 단어 concatenate의 약자로 결합하다라는 의미

# 연구방법의 적용I. 들어가며

## 벡터(Vector)

벡터를 만드는 2가지 방법

- 빈 벡터 선언 후 채우기
- 만들면서 채우기: c( ) 함수, : 연산자, seq( ) 함수

```
x <- vector(length = 3)
x
```

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

- 자동으로 FALSE를 채워줌.

# 연구방법의 적용I. 들어가며

## 벡터(Vector)

[]을 사용하여 원소에 접근

```
x[2] <- 3
x
```

```
## [1] 0 3 0
```

R에서 FALSE는 0, TRUE는 1로 간주됨.

# 연구방법의 적용I. 들어가며

## 벡터(Vector)

### 만들면서 채우기: c() 함수와 : 연산자 활용

```
c(1:5)
```

```
## [1] 1 2 3 4 5
```

```
c(1:5) * 2
```

```
## [1]  2  4  6  8 10
```

```
c(1:5) * 2 - 1
```

```
## [1] 1 3 5 7 9
```

# 연구방법의 적용I. 들어가며

## 벡터(Vector)

### 만들면서 채우기: c( ) 함수와 : 연산자 활용

: 연산자의 연산 순서는 다른 사칙 연산보다 위에 있음.

```
1:3 - 2
```

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

```
1:(3 - 2)
```

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

```
1:3 * 2
```

```
## [1] 2 4 6
```

```
1:(3 * 2)
```

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

# 연구방법의 적용I. 들어가며

## 벡터(Vector)

### 만들면서 채우기: seq() 함수 이해하기

seq(시작값, 끝값, 옵션): by 옵션과 length.out 옵션 활용

```
seq(2, 10, by = 2)
```

```
## [1]  2  4  6  8 10
```

```
seq(2, 10, length = 3)
```

```
## [1]  2  6 10
```

# 연구방법의 적용I. 들어가며

## 벡터(Vector)

### 같은 원소들로 채우는 방법

rep() 함수 이해하기: rep(반복대상, 반복횟수)

```
rep(8, 4)
```

```
## [1] 8 8 8 8
```

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

```
## [1] 1 2 4 1 2 4
```

```
rep(c(1, 2, 4), each = 2)
```

```
## [1] 1 1 2 2 4 4
```

# 연구방법의 적용I. 들어가며

## 객체 접근방법: [ ] 연산자

벡터는 여러 숫자를 색인을 통하여 접근하며, 첫 번째 자리가 1에서부터 시작

대괄호 [ ]를 사용해서 부분-원소에 접근할 수 있음. 4개의 숫자가 들어있는 벡터 x의 3번째 숫자에 접근하는 방법

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

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

2번째와 4번째 숫자에 접근할 수도 있음. 이때는 벡터의 색인의 벡터를 이용

```
x[c(2, 4)]
```

```
## [1] 2 5
```

# 연구방법의 적용I. 들어가며

## 객체 접근방법: 콜론(:) 연산자

처음 숫자와 마지막 숫자까지 1씩 증가 혹은 감소하는 수열을 만들어 줌.

```
# 직접 손으로 작성할 때
c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

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

```
# 콜론을 활용할 때
c(1:10)
```

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

```
# 숫자가 줄어드는 수열도 만들 수 있음
c(10:(-1))
```

```
##  [1] 10  9  8  7  6  5  4  3  2  1  0 -1
```

# 연구방법의 적용I. 들어가며

## 벡터 인덱싱(Indexing)

원하는 벡터 원소만을 선택하기: `vector1[vector2]`

```
x <- 1:10 * 2
x
```

```
##  [1]  2  4  6  8 10 12 14 16 18 20
```

```
x[3:6]
```

```
## [1]  6  8 10 12
```

# 연구방법의 적용I. 들어가며

## 벡터 인덱싱(Indexing)

### 인덱싱 중복 가능

```
x[c(2, 2, 4, 3)]
```

```
## [1] 4 4 8 6
```

### 특정 원소 빼고 선택

```
x[-1]
```

```
## [1]  4  6  8 10 12 14 16 18 20
```

```
x[-c(1:3)]
```

```
## [1]  8 10 12 14 16 18 20
```

# 연구방법의 적용I. 들어가며

## 벡터에 대한 논리 연산

### all() 함수, any() 함수

```
x
```

```
##  [1]  2  4  6  8 10 12 14 16 18 20
```

```
all(x < 10)
```

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

```
any(x < 10)
```

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

# 연구방법의 적용I. 들어가며

## 벡터에 대한 논리 연산과 필터링

x 벡터의 각 원소가 10보다 작은가?

```
x < 10
```

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

벡터 필터링: `vector[condition]`

```
x[x < 10]
```

```
## [1] 2 4 6 8
```

# 연구방법의 적용I. 들어가며

## 논리 연산자와 조건문

### 알아두면 쓸데있는 연산자들

==, !=

```
x[x == 4]
```

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

```
x[x != 4]
```

```
## [1]  2  6  8 10 12 14 16 18 20
```

%%(나머지), %/%(몫)

```
x[x %% 4 == 0]
```

```
## [1]  4  8 12 16 20
```

```
x[x %/% 4 == 2]
```

```
## [1]  8 10
```

# 연구방법의 적용I. 들어가며

## 논리 연산자와 조건문

### 알아두면 쓸데있는 연산자들

```
y <- 1:5000

# 26으로 나누어 떨어지는 애들의 갯수는?

sum(y %% 26 == 0)
```

```
## [1] 192
```

```
length(y[y %% 26 == 0])
```

```
## [1] 192
```

짝수나 홀수번의 관측치만을 사용해 분석해야할 때 유용하게 사용할 수 있음.

# 연구방법의 적용I. 들어가며

## 논리 연산자와 조건문

```
a <- c(TRUE, TRUE, FALSE)
b <- c(TRUE, FALSE, FALSE)
```

### & (AND) 와 | (OR)

```
a & b
```

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

```
a | b
```

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

# 연구방법의 적용I. 들어가며

## 논리 연산자와 조건문

아래의 코드가 의미하는 것을 풀어서 설명해보자:

```
x[x == 4 | x > 15]
```

```
## [1]  4 16 18 20
```

```
x[x < 8 | x > 13]
```

```
## [1]  2  4  6 14 16 18 20
```

# 연구방법의 적용I. 들어가며

## 논리 연산자와 조건문

### 조건을 만족하는 which()

```
x
```

```
##  [1]  2  4  6  8 10 12 14 16 18 20
```

```
x < 7
```

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

```
which(x < 7)
```

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

# 연구방법의 적용I. 들어가며

## 논리 연산자와 조건문

### 필터링을 이용한 벡터 변경

```
x
```

```
##  [1]  2  4  6  8 10 12 14 16 18 20
```

```
x[x >= 10] <- 10
x
```

```
##  [1]  2  4  6  8 10 10 10 10 10 10
```

# 연구방법의 적용I. 들어가며

## 데이터가 없을 때에는 NA

NA (Not Available): 결측치(missing data)

```
a <- c(20, NA, 13, 24, 309)
a
```

```
## [1]  20  NA  13  24 309
```

NA 무시 옵션

```
mean(a)
```

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

```
mean(a, na.rm = T)
```

```
## [1] 91.5
```

# 연구방법의 적용I. 들어가며

## 존재하지 않음을 나타내는 NULL

### NA와 NULL의 차이

```
NULL_is_not_blank <- NULL
c(1, NULL_is_not_blank)
```

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

```
NA_is_blank <- NA
c(1, NA_is_blank)
```

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

# 연구방법의 적용I. 들어가며

## 벡터에 이름 붙여주기

경우에 따라서 벡터의 각 원소에 이름을 붙여줄 수 있음.

### 보통의 경우

```
my_vector <- c(1, 20, 300)
names(my_vector)
```

```
## NULL
```

```
my_vector
```

```
## [1]   1  20 300
```

### 이름 붙여준 경우

```
names(my_vector) <- c("first", "second")
my_vector
```

```
##  first second   <NA>
##      1     20    300
```

```
my_vector["second"]
```

```
## second
##     20
```

# 연구방법의 적용I. 들어가며

## 여러 벡터들을 묶어보자

`cbind()`와 `rbind()`

### 세로로 붙여주는 column bind

```
cbind(1:4, 12:15)
```

```
##      [,1] [,2]
## [1,]    1   12
## [2,]    2   13
## [3,]    3   14
## [4,]    4   15
```

### 가로로 쌓아주는 row bind

```
rbind(1:4, 12:15)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]   12   13   14   15
```

# 연구방법의 적용I. 들어가며

## 기본 제공함수 사용하기

### base 함수들

R에서 기본으로 제공하는 함수들의 모임. R이 시작될 때부터 자동으로 R의 시스템에 올라와 있음. 그냥 쓰면 됨.

```
mean(x)
```

```
## [1] 8
```

### 함수 사용법/설치법 확인법

사용법: 콘솔 창에 ?mean()을 쳐보자

설치법: 기본적으로 설치하고자 하는 {패키지}를 install.packages("패키지")로 설치

# 연구방법의 적용I. 들어가며

## 기본 제공함수 사용하기

### 함수 사용법/설치법 확인법

패키지 사용법: 설치된 패키지에 대해 `library(패키지)`

- 불러온 패키지 안의 함수들이 `library(패키지)`를 통해 글로벌 namespace로 올려지면, 사용자는 그 패키지 안에 속한 함수들을 기본함수처럼 바로 사용할 수 있게 됨.
- R에서는 서로 다른 패키지가 동일한 이름의 함수를 가지고 있을 수 있음.
- 더 최근에 불려진 패키지의 함수가 로드됨.
    - 만약 특정한 패키지의 함수를 사용하고 싶다면, 더블콜론을 사용하여 해당 패키지의 그 함수를 불러와 사용할 수 있음: 패키지`::`함수

# 연구방법의 적용I. 들어가며

## 벡터화(Vectorized) 코드

R을 R로 만들어주는 대표적 특징: 수학에서 배우는 벡터 연산을 기본적으로 지원함.

$$\begin{pmatrix} 1 \\ 2 \\ 4 \end{pmatrix} + \begin{pmatrix} 2 \\ 3 \\ 5 \end{pmatrix} = \begin{pmatrix} 3 \\ 5 \\ 9 \end{pmatrix}$$

```
c(1, 2, 4) + c(2, 3, 5)
```

```
## [1] 3 5 9
```

$$2 \begin{pmatrix} 1 \\ 2 \\ 4 \\ 5 \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \\ 8 \\ 10 \end{pmatrix}$$

```
x <- c(1, 2, 4, 5)
x * 2
```

```
## [1]  2  4  8 10
```

# 연구방법의 적용I. 들어가며

## 재활용(Recyclying)

길이가 맞지 않는 벡터들을 자동으로 길이를 맞춰줌.

```
c(1:4) + c(1, 2)
```

```
## [1] 2 4 4 6
```

```
c(1:4) + c(1:3)
```

```
## [1] 2 4 6 5
```

# 연구방법의 적용I. 들어가며

## 행렬이란 무엇일까?

벡터들을 모아놓은 것으로 꼭 사각형의 모양을 가짐. dim()으로 사각형의 크기를 잴 수 있음.

```
cbind(1:4, 12:15)
```

```
##      [,1] [,2]
## [1,]    1   12
## [2,]    2   13
## [3,]    3   14
## [4,]    4   15
```

```
dim(cbind(1:4, 12:15))
```

```
## [1] 4 2
```

# 연구방법의 적용I. 들어가며

## 행렬이란 무엇일까?

행렬 선언하기: `matrix()` 함수

- 행과 열 중 하나만 입력해도 자동으로 계산해서 행렬로 만들어줌.
- 위치는 대괄호 안에 순서 쌍으로 나타냄: `[row, col]`

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

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

```
y[1, 2]
```

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

# 연구방법의 적용I. 들어가며

## 행렬을 채우는 방법: byrow 옵션

byrow 옵션을 통하여 행렬에 숫자를 채우는 방향을 정해줄 수 있음.

### 세로로 채우기

```
matrix(1:6, nrow = 2)
```

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

### 가로로 채우기

```
matrix(1:6, nrow = 2,
       byrow = TRUE)
```

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

# 연구방법의 적용I. 들어가며

## 행렬의 원소에 접근하기

### 인덱싱(Indexing)

```
x <- matrix(1:10 * 2, ncol = 2)
x
```

```
##      [,1] [,2]
## [1,]    2   12
## [2,]    4   14
## [3,]    6   16
## [4,]    8   18
## [5,]   10   20
```

#### 전체를 나타내는 빈 칸

```
x[, 2]
```

```
## [1] 12 14 16 18 20
```

#### 선택적으로 골라오기

```
x[c(2, 3, 5), 2]
```

```
## [1] 14 16 20
```

# 연구방법의 적용I. 들어가며

## 필터링

TRUE, FALSE 벡터를 사용해서 필터링이 됨.

```
x[c(TRUE, TRUE, FALSE, FALSE, TRUE), 1]
```

```
## [1]  2  4 10
```

조건문 사용한 필터 가능

```
x[x[, 2] > 15, 1]
```

```
## [1]  6  8 10
```

# 연구방법의 적용I. 들어가며

## 행렬 클래스

class() 함수를 통해 우리가 만든 행렬이 matrix, array (행렬의 확장 개념) 클래스라는 것을 확인할 수 있음.

```
class(x)
```

```
## [1] "matrix" "array"
```

특정 클래스에는 접근가능한 속성(attribute) 들이 정의되어 있음.

```
attributes(x)
```

```
## $dim
## [1] 5 2
```

# 연구방법의 적용I. 들어가며

## 행렬 뒤집기

`t()`: transpose를 의미

```
x
```

```
##      [,1] [,2]
## [1,]    2   12
## [2,]    4   14
## [3,]    6   16
## [4,]    8   18
## [5,]   10   20
```

```
t(x)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    2    4    6    8   10
## [2,]   12   14   16   18   20
```

# 연구방법의 적용I. 들어가며

## 행렬의 연산

행렬의 곱셈: %*%, 행렬의 곱셈은 크기가 맞아야지 가능

```
dim(x)
```

```
## [1] 5 2
```

```
dim(y)
```

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

```
x %*% y
```

```
##      [,1] [,2]
## [1,]   26   54
## [2,]   32   68
## [3,]   38   82
## [4,]   44   96
## [5,]   50  110
```

# 연구방법의 적용I. 들어가며

## 행렬의 연산

원소별 곱셈(hadamard product, element-wise product)

```
y
```

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

```
z <- matrix(10:13, ncol = 2)
z
```

```
##      [,1] [,2]
## [1,]   10   12
## [2,]   11   13
```

```
y * z
```

```
##      [,1] [,2]
## [1,]   10   36
## [2,]   22   52
```

# 연구방법의 적용I. 들어가며

## 행렬의 역행렬

`solve()` 함수

행렬이 입력값으로 들어가면 역행렬이 구해지도록 설계됨.

```
solve(y)
```

```
##      [,1] [,2]
## [1,]   -2  1.5
## [2,]    1 -0.5
```

역행렬이 존재하지 않는 경우

```
no_inverse <- matrix(c(1, 2, 1, 2))
solve(no_inverse)
```

```
## Error in solve.default(no_inverse): 'a'
```

# 연구방법의 적용I. 들어가며

## 행렬의 연산과 recycling

행렬 연산에서의 recycling

- 행렬에 벡터를 곱하면, 벡터 길이를 맞춰 계산하듯, 행렬 크기도 맞춰서 계산

```
y
```

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

```
y * c(1, 2)
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    4    8
```

# 연구방법의 적용I. 들어가며

## 행렬의 연산과 recycling

행렬 연산에서의 recycling

- 단, 행렬과 행렬 계산에서는 적용 안됨.

```
y * matrix(c(1, 2), ncol = 1)
```

```
## Error in y * matrix(c(1, 2), ncol = 1): non-conformable arrays
```

# 연구방법의 적용I. 들어가며

## 행렬에 이름 붙이기

```
y
```

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

```
colnames(y)
```

```
## NULL
```

```
colnames(y) <- c("col_1", "col_2")
y
```

```
##      col_1 col_2
## [1,]     1     3
## [2,]     2     4
```

```
rownames(y) <- c("row_1", "row_2")
y
```

```
##       col_1 col_2
## row_1     1     3
## row_2     2     4
```

# 연구방법의 적용I. 들어가며

## 배열(array())

고차원 행렬: 행렬을 붙여놓을 수 있지 않을까?

```
mat1 <- matrix(1:6, nrow = 2)
mat2 <- matrix(7:12, nrow = 2)
my_array <- array(
  data = c(mat1, mat2),
  dim = c(2, 3, 2)
)
```

```
my_array
```

```
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]    7    9   11
## [2,]    8   10   12
```

# 연구방법의 적용I. 들어가며

## 배열 다루기

행렬의 필터링 접근 방식이 그대로 적용됨

```
my_array[, , 1]
```

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

```
my_array[, -3, ]
```

```
## , , 1
##
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
##
## , , 2
##
##      [,1] [,2]
## [1,]    7    9
## [2,]    8   10
```

# 연구방법의 적용I. 들어가며

## 배열 차원 다루기

### 배열에서의 transpose

```
my_array
```

```
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]    7    9   11
## [2,]    8   10   12
```

```
aperm(my_array, c(2, 1, 3))
```

```
## , , 1
##
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
## [3,]    5    6
##
## , , 2
##
##      [,1] [,2]
## [1,]    7    8
## [2,]    9   10
## [3,]   11   12
```

# 연구방법의 적용I. 들어가며

## 리스트(List)란 무엇일까?

언제 사용할까?

- 벡터의 경우 원소들이 모두 같은 타입이어야 함.
- 각기 다른 타입의 원소를 가진 객체를 만들 수는 없을까?

```
as.vector(c(1, "test"))
```

```
## [1] "1"    "test"
```

```
as.vector(c(1, TRUE))
```

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

```
as.vector(c(FALSE, TRUE))
```

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

# 연구방법의 적용I. 들어가며

## 리스트(List) 만들기

**장점**

여러 개의 다른 객체들(objects)을 모아 놓을 수 있음.

**선언방법**

`list()` 함수를 사용하여 선언

각 구성원의 태그(tag)와 내용을 같이 설명해줌.

```
mylist <- list(
  name = "sanghoon",
  id = 20250514,
  order = c(1, 2))

mylist
```

```
## $name
## [1] "sanghoon"
##
## $id
## [1] 20250514
##
## $order
## [1] 1 2
```

# 연구방법의 적용I. 들어가며

## 벡터 vs. 리스트

**공통점**

작은 원소들을 모아놓은 객체

리스트도 벡터의 한 종류

**차이점**

`atomic vector`

- 작은 구성원들로 쪼갤 수 없기 때문

`recursive vector`

- 작은 구성원들로 쪼개짐(접근 가능)

```
mylist$name
```

```
## [1] "sanghoon"
```

```
mylist$id
```

```
## [1] 20250514
```

```
mylist$order
```

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

# 연구방법의 적용I. 들어가며

## 리스트 인덱싱(indexing)

**구성원소 접근하기**

$ 기호를 이용하여 접근

**[[]] vs. []**

구성원소의 접근을 위해서 [[]] 이용

원래 리스트의 부분을 잡아내기 위해서는 [] 이용

```
mylist$name
```

```
## [1] "sanghoon"
```

```
mylist[["name"]]
```

```
## [1] "sanghoon"
```

```
mylist["name"]
```

```
## $name
## [1] "sanghoon"
```

# 연구방법의 적용I. 들어가며

## 리스트 인덱싱(indexing)

**숫자를 사용한 접근**

상황에 따라서 편리한 접근 방법을 사용

[[]]의 결과는 구성원소 그 자체를 반환하는 반면, []의 결과는 미진

```
mylist[[1]]
```

```
## [1] "sanghoon"
```

```
mylist[1]
```

```
## $name
## [1] "sanghoon"
```

# 연구방법의 적용I. 들어가며

## 리스트 구성원소 추가/삭제/변경

변경 및 추가

$ 기호를 사용함.

```
mylist$id <- 202124
mylist
```

```
## $name
## [1] "sanghoon"
##
## $id
## [1] 202124
##
## $order
## [1] 1 2
```

```
mylist$add <- "new element"
```

# 연구방법의 적용I. 들어가며

## 리스트 안의 리스트: 재귀(recursive) 리스트

```
mylist$new <- list("hello", c(1, 3, 2))
mylist
```

```
## $name
## [1] "sanghoon"
##
## $id
## [1] 202124
##
## $order
## [1] 1 2
##
## $add
## [1] "new element"
##
## $new
## $new[[1]]
## [1] "hello"
##
## $new[[2]]
```

# 연구방법의 적용I. 들어가며

## 리스트에 함수 적용하기: `lapply()`, `sapply()`

리스트의 각 원소에 동일한 함수를 적용

- `l(ist) apply(list, function)`
- `s(implied) (l)apply()`
- 결과는 리스트로 나옴

KNU 강원대학교
KANGWON NATIONAL UNIVERSITY

# 감사합니다!

## 궁금한 것이 있으면 언제든 연락하세요.

강사 연락처

| 연락처 | 박상훈 |
| --- | --- |
| | sh.park.poli@gmail.com |
| | sanghoon-park.com/ |
| | 영상바이오관 405 |