

# R 데이터분석 입문

## Chapter 04 조건문,반복문,함수

1

### IT CookBook, R 프로그래밍 입문

#### [강의교안 이용 안내]

- 본 강의교안의 저작권은 오세종과 한빛아카데미에 있습니다.
- 이 자료는 강의 보조자료로 제공되는 것으로, 학생들에게 배포되어서는 안 됩니다.

2

## 목차

1. 조건문
2. 반복문
3. `apply()` 함수
4. 사용자 정의 함수
5. 조건에 맞는 데이터의 위치 찾기

3/42

3

## 단원 요약

- R을 이용한 데이터 분석은 대부분 R에서 제공하는 함수들을 사용하여 작업할 수 있지만 경우에 따라서는 프로그래밍 기법을 적용해야 가능
- 프로그래밍(programming)이란 컴퓨터를 이용하여 문제를 해결하기 위해 주어진 절차를 문법에 맞게 작성하는 과정
- 이번 단원에서는 프로그래밍의 기본이 되는 조건문, 반복문, 사용자 정의 함수에 대해 학습
- 프로그래밍 문법이 무엇인지를 알고 타인이 작성한 R 코드를 이해할 수 있으며, 간단히 프로그래밍 문법을 응용할 수 있는 정도를 목표로함

4/40

4

## 1. 조건문

### ■ If~else문

- 조건문(conditional statement)은 조건에 따라 특정 명령을 실행을 하도록 하는 프로그래밍 명령문
- 조건에 따라 실행할 명령문을 달리해야 하는 경우에 사용
- if-else문의 기본 문법

```
if(비교 조건) {  
  조건이 참일 때 실행할 명령문(들)  
} else {  
  조건이 거짓 일 때 실행할 명령문(들)  
}
```

5/40

5

## 1. 조건문

### ➤ 기본 if~else

코드4-1

```
job.type <- 'A'  
if(job.type == 'B') {  
  bonus <- 200      # 직무 유형이 B일 때 실행  
} else {  
  bonus <- 100      # 직무 유형이 B가 아닌 나머지 경우 실행  
}  
print(bonus)
```

```
> job.type <- 'A'  
> if(job.type == 'B') {  
+   bonus <- 200      # 직무 유형이 B일 때 실행  
+ } else {  
+   bonus <- 100      # 직무 유형이 B가 아닌 나머지 경우 실행  
+ }  
> print(bonus)  
[1] 100
```

주의. 두 값이 같은지를 비교하는 연산자는 = 가 아닌 ==

6/40

6

## 1. 조건문

- else가 생략된 if문

코드4-2

```
job.type <- 'B'
bonus <- 100
if(job.type == 'A') {
  bonus <- 200      # 직무 유형이 A일 때 실행
}
print(bonus)
```

```
> job.type <- 'B'
> bonus <- 100
> if(job.type == 'A') {
+   bonus <- 200      # 직무 유형이 A일 때 실행
+ }
> print(bonus)
[1] 100
```

7/40

7

## 1. 조건문

- 다중 if~else문

코드4-3

```
score <- 85

if (score > 90) {
  grade <- 'A'
} else if (score > 80) {
  grade <- 'B'
} else if (score > 70) {
  grade <- 'C'
} else if (score > 60) {
  grade <- 'D'
} else {
  grade <- 'F'
}

print(grade)
```

```
> score <- 85
>
> if (score > 90) {
+   grade <- 'A'
+ } else if (score > 80) {
+   grade <- 'B'
+ } else if (score > 70) {
+   grade <- 'C'
+ } else if (score > 60) {
+   grade <- 'D'
+ } else {
+   grade <- 'F'
+ }
>
> print(grade)
[1] "B"
```

8/40

8

## 1. 조건문

- **Note. 코드 블록**

```
a <- 10
if(a<5) {
  print(a)
} else {
  print(a*10)
  print(a/10)
}
```

코드블록

코드블록

- 중괄호 { }는 프로그래밍에서 **코드블록(code block)**이라고 부르는데, 코드 블록은 여러 명령문을 하나로 묶어주는 역할
- 코드블록에 의해 묶인 명령문은 조건에 의해 모두 실행이 되거나 모두 실행이 되지 않는다.

9/40

9

## 1. 조건문

- 조건문에서 논리 연산자의 사용

- if문에 논리 연산자를 사용하면 복잡한 조건문을 서술할 수 있다.
- 대표적인 논리연산자는 **&(and)**와 **|(or)**

코드4-4

```
a <- 10
b <- 20
if(a>5 & b>5) {          # and 사용
  print (a+b)
}
if(a>5 | b>30) {          # or 사용
  print (a*b)
}
```

```
> a <- 10
> b <- 20
> if(a>5 & b>5) {          # and 사용
+   print (a+b)
+ }
[1] 30
> if(a>5 | b>30) {          # or 사용
+   print (a*b)
+ }
[1] 200
```

10/40

10

## 1. 조건문

### ■ ifelse문

- 조건에 따라 둘중 하나의 값 또는 변수를 선택할 때 사용
- ifelse문의 문법

```
ifelse(비교 조건, 조건이 참일 때 선택할 값/변수, 조건이 거짓일 때 선택할 값/변수)
```

코드4-5

```
a <- 10
b <- 20

if (a>b) {                                # 일반적인 if~else 문
  c <- a
} else {
  c <- b
}
print(c)

a <- 10
b <- 20

c <- ifelse(a>b, a, b)                    # ifelse 문
print(c)
```

11/40

11

## 1. 조건문

```
> a <- 10
> b <- 20
>
> if (a>b) {                                # 일반적인 if~else 문
+   c <- a
+ } else {
+   c <- b
+ }
> print(c)
[1] 20
>
> a <- 10
> b <- 20
>
> c <- ifelse(a>b, a, b)                    # ifelse 문
> print(c)
[1] 20
```

12/40

12

## 1. 조건문

### ■ Note. If~else 주의 사항

- if-else문을 작성할 때, 다음과 같이 사용하면 오류가 발생한다. else는 반드시 if문의 코드 블록이 끝나는 부분에 있는 }와 같은 줄에 작성해야 한다.

```
job.type <- 'A'
if (job.type == 'B') {
  bonus <- 200
}
else {
  bonus <- 100
}
```

# 에러 발생, 윗 줄로 옮겨야 한다.

```
job.type <- 'A'
if (job.type == 'B') {
  bonus <- 200
} else {
  bonus <- 100
}
```

13/40

13

## 2. 반복문

### ■ for문

- 반복문(repetitive statement)은 정해진 동작을 반복적으로 수행할 때 사용하는 명령문
- 동일 명령문을 여러 번 반복해서 실행할 때 사용
- for문의 문법

```
for (반복 변수 in 반복 범위) {
  반복할 명령문(들)
}
```

코드4-6

```
for(i in 1:5) {
  print('*')
}
```

```
> for(i in 1:5) {
+   print('*')
+ }
[1] "*"
[1] "*"
[1] "*"
[1] "*"
[1] "*"
```

14/40

14

## 2. 반복문

- 반복 범위에 따른 반복 변수의 값 변화

코드4-7

```
for(i in 6:10) {  
  print(i)  
}
```

```
> for(i in 6:10) {  
+   print(i)  
+ }  
[1] 6  
[1] 7  
[1] 8  
[1] 9  
[1] 10
```

15/40

15

## 2. 반복문

- 반복 변수를 이용한 구구단 출력

코드4-8

```
for(i in 1:9) {  
  cat('2 *', i, '=', 2*i, '\n')  
}
```

```
> for(i in 1:9) {  
+   cat('2 *', i, '=', 2*i, '\n')  
+ }  
2 * 1 = 2  
2 * 2 = 4  
2 * 3 = 6  
2 * 4 = 8  
2 * 5 = 10  
2 * 6 = 12  
2 * 7 = 14  
2 * 8 = 16  
2 * 9 = 18
```

cat()함수: 여러 개의 값을 연결하여 출력할 때 사용  
'\n' : 줄바꿈을 의미하는 특수문자

16/40

16



## 2. 반복문

### > for문 안에서 if문의 사용

- 1~20 사이의 숫자 중 짝수만 골라서 출력

코드4-9

```
for(i in 1:20) {  
  if(i%%2==0) {          # 짝수인지 확인  
    print(i)  
  }  
}
```

```
> for(i in 1:20) {  
+   if(i%%2==0) {          # 짝수인지 확인  
+     print(i)  
+   }  
+ }  
[1] 2  
[1] 4  
[1] 6  
[1] 8  
[1] 10  
[1] 12  
[1] 14  
[1] 16  
[1] 18  
[1] 20
```

17/40

17

## 2. 반복문

### > 1~100 사이의 숫자의 합 출력

코드4-10

```
sum <- 0  
for(i in 1:100) {  
  sum <- sum + i          # sum에 i 값을 누적  
}  
print(sum)
```

```
> sum <- 0  
> for(i in 1:100) {  
+   sum <- sum + i          # sum에 i 값을 누적  
+ }  
> print(sum)  
[1] 5050
```

18/40

18

## 2. 반복문

➤ iris에서 꽃잎의 길이에 따른 분류 작업

- 꽃잎의 길이(Petal.Length)에 따라 1.6 이하이면 "L", 5.1 이상이면 "H", 나머지는 "M"으로 분류하여 레이블을 부여하는 작업

코드4-11

```

norow <- nrow(iris)                # iris의 행의 수
mylabel <- c()                     # 비어있는 벡터 선언
for(i in 1:norow) {               # 레이블 결정
  if (iris$Petal.Length[i] <= 1.6) {
    mylabel[i] <- 'L'
  } else if (iris$Petal.Length[i] >= 5.1) {
    mylabel[i] <- 'H'
  } else {
    mylabel[i] <- 'M'
  }
}
print(mylabel)                    # 레이블 출력
# 꽃잎의 길이와 레이블 결합
newds <- data.frame(iris$Petal.Length, mylabel)
head(newds)                      # 새로운 데이터셋 내용 출력

```

19/40

19

## 2. 반복문

```

> norow <- nrow(iris)                # iris의 행의 수
> mylabel <- c()                     # 비어있는 벡터 선언
> for(i in 1:norow) {
+   if (iris$Petal.Length[i] <= 1.6) {   # 레이블 결정
+     mylabel[i] <- 'L'
+   } else if (iris$Petal.Length[i] >= 5.1) {
+     mylabel[i] <- 'H'
+   } else {
+     mylabel[i] <- 'M'
+   }
+ }
> print(mylabel)                    # 레이블 출력
[1] "L" "L" "L" "L" "M" "L" "L" "L" "L" "L" "L" "L" "L"
[16] "L" "L" "L" "M" "L" "M" "L" "M" "M" "L" "L" "L" "L"
[31] "L" "L" "L" "L" "L" "L" "L" "L" "L" "L" "L" "L" "M"
[46] "L" "L" "L" "L" "L" "M" "M" "M" "M" "M" "M" "M" "M"
[61] "M" "M" "M" "M" "M" "M" "M" "M" "M" "M" "M" "M" "M"
[76] "M" "M" "M" "M" "M" "M" "M" "M" "H" "M" "M" "M" "M"
[91] "M" "M" "M" "M" "M" "M" "M" "M" "H" "H" "H" "H" "H"
[106] "H" "M" "H" "H" "H" "H" "H" "M" "H" "H" "H" "H" "H"
[121] "H" "M" "H" "M" "H" "H" "M" "H" "H" "H" "H" "H" "H"
[136] "H" "H" "H" "M" "H" "H" "H" "H" "H" "H" "M" "H" "H"
> # 꽃잎의 길이와 레이블 결합
> newds <- data.frame(iris$Petal.Length, mylabel)
> head(newds)                      # 새로운 데이터셋 내용 출력
  iris.Petal.Length mylabel
1             1.4      L
2             1.4      L
3             1.3      L
4             1.5      L
5             1.4      L
6             1.7      M

```

20/40

20

## 2. 반복문

### ■ while문

- while문은 어떤 조건이 만족하는 동안 코드블록을 수행하고, 해당 조건이 거짓일 경우 반복을 종료하는 명령문
- while문의 기본 문법

```
while (비교조건) {  
  반복할 명령문(들)  
}
```

21/40

21

## 2. 반복문

코드4-12

```
sum <- 0  
i <- 1  
while(i <=100) {  
  sum <- sum + i      # sum에 i 값을 누적  
  i <- i + 1          # i 값을 1 증가시킴  
}  
print(sum)
```

```
> sum <- 0  
> i <- 1  
> while(i <=100) {  
+   sum <- sum + i      # sum에 i 값을 누적  
+   i <- i + 1          # i 값을 1 증가시킴  
+ }  
> print(sum)  
[1] 5050
```

22/40

22

## 2. 반복문

### > while문 주의사항

- while문을 실행하기 전에 sum과 i를 먼저 선언해야함
- 코드블록 안에서 반복변수에 해당하는 i 값을 1 증가시키는 부분 필요 -> 없으면 무한루프에 빠짐

```
sum <- 0
i <- 1
while(i <= 100) {
  sum <- sum + i          # sum에 i 값을 누적
  i <- i + 1             # i 값을 1 증가시킴
}
print(sum)
```

어떤 작업의 반복횟수가 정해져 있다면 for문을 사용하고,  
반복횟수가 가변적이라면 while문을 사용하는 것이 편리

for나 while 을 사용하지 않고도 동일한 결과를 얻을 수 있으면 그것이 더 좋은 방법임

23/40

23

## 3. apply() 함수

- 반복 작업이 필요한 경우에는 반복문을 적용
- 반복 작업의 대상이 매트릭스나 데이터프레임의 행(row) 또는 열(column)인 경우는 for문이나 while문 대신에 apply() 함수를 이용할 수 있음
- apply() 함수의 문법

*어떤 함수를 반복할지*  
apply(데이터셋, 행/열방향 지정, 적용 함수)

데이터셋	반복 작업을 적용할 대상 매트릭스나 데이터 프레임의 이름을 입력
행/열방향 지정	행방향 작업의 경우 1, 열방향 작업의 경우 2를 지정
적용함수	반복 작업의 내용을 알려주는 것으로, R 함수이거나 다음 절에서 배울 사용자 정의 함수를 지정

24/40

24

### 3. apply() 함수

#### ■ apply() 함수의 적용

코드4-13

```
apply(iris[,1:4], 1, mean) # row 방향으로 함수 적용
apply(iris[,1:4], 2, mean) # col 방향으로 함수 적용

> apply(iris[,1:4], 1, mean) # row 방향으로 함수 적용
[1] 2.550 2.375 2.350 2.350 2.550 2.850 2.425 2.525 2.225 2.400
[11] 2.700 2.500 2.325 2.125 2.800 3.000 2.750 2.575 2.875 2.675
[21] 2.675 2.675 2.350 2.650 2.575 2.450 2.600 2.600 2.550 2.425
[31] 2.425 2.675 2.725 2.825 2.425 2.400 2.625 2.500 2.225 2.550
[41] 2.525 2.100 2.275 2.675 2.800 2.375 2.675 2.350 2.675 2.475
[51] 4.075 3.900 4.100 3.275 3.850 3.575 3.975 2.900 3.850 3.300
[61] 2.875 3.650 3.300 3.775 3.350 3.900 3.650 3.400 3.600 3.275
[71] 3.925 3.550 3.800 3.700 3.725 3.850 3.950 4.100 3.725 3.200
[81] 3.200 3.150 3.400 3.850 3.600 3.875 4.000 3.575 3.500 3.325
[91] 3.425 3.775 3.400 2.900 3.450 3.525 3.525 3.675 2.925 3.475
[101] 4.525 3.875 4.525 4.150 4.375 4.825 3.400 4.575 4.200 4.850
[111] 4.200 4.075 4.350 3.800 4.025 4.300 4.200 5.100 4.875 3.675
[121] 4.525 3.825 4.800 3.925 4.450 4.550 3.900 3.950 4.225 4.400
[131] 4.550 5.025 4.250 3.925 3.925 4.775 4.425 4.200 3.900 4.375
[141] 4.450 4.350 3.875 4.550 4.550 4.300 3.925 4.175 4.325 3.950
> apply(iris[,1:4], 2, mean) # col 방향으로 함수 적용
Sepal.Length Sepal.Width Petal.Length Petal.Width
5.843333 3.057333 3.758000 1.199333
```

25/40

25

### 3. apply() 함수

apply(iris[,1:4], 1, mean)


	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	
	5.1	3.5	1.4	0.2	mean()
	4.9	3.0	1.4	0.2	mean()
.	4.7	3.2	1.3	0.2	
.	4.6	3.1	1.5	0.2	
	5.0	3.6	1.4	0.2	
	5.4	3.9	1.7	0.4	
	4.6	3.4	1.4	0.3	
	5.0	3.4	1.5	0.2	
.	4.4	2.9	1.4	0.2	
.	4.9	3.1	1.5	0.1	
	5.4	3.7	1.5	0.2	
	4.8	3.4	1.6	0.2	
	4.8	3.0	1.4	0.1	
	4.3	3.0	1.1	0.1	
	5.8	4.0	1.2	0.2	mean()

26/40

26

### 3. apply() 함수

`apply(iris[,1:4], 2, mean)`



Sepal.Length	Sepal.width	Petal.Length	Petal.width
5.1	3.5	1.4	0.2
4.9	3.0	1.4	0.2
4.7	3.2	1.3	0.2
4.6	3.1	1.5	0.2
5.0	3.6	1.4	0.2
5.4	3.9	1.7	0.4
4.6	3.4	1.4	0.3
5.0	3.4	1.5	0.2
4.4	2.9	1.4	0.2
4.9	3.1	1.5	0.1
5.4	3.7	1.5	0.2
4.8	3.4	1.6	0.2
4.8	3.0	1.4	0.1
4.3	3.0	1.1	0.1
5.8	4.0	1.2	0.2

mean()      mean()      mean()      mean()

apply() 함수와 유사한 함수로 lapply(), sapply(), tapply(), mapply() 함수 등이 있다

27/40

27

### 4. 사용자 정의 함수

#### ■ 사용자 정의 함수 만들기

- 사용자 정의 함수 : R은 사용자들도 자신만의 함수를 만들어 사용할 수 있는 기능을 제공 하는데, 이를 **사용자 정의 함수**라고 한다.
- 사용자 정의 함수 문법

```
함수명 <- function(매개변수 목록) {  
  실행할 명령문(들)  
  return(함수의 실행 결과)  
}
```

28/40

28

## 4. 사용자 정의 함수

- 두 개의 값을 입력 받아 둘 중에서 큰 수를 결과 값으로 돌려주는 함수

코드4-14 함수의 정의

```
mymax <- function(x,y) {  
  num.max <- x  
  if (y > x) {  
    num.max <- y  
  }  
  return(num.max)  
}
```

코드4-15 함수의 사용

```
mymax(10,15)  
a <- mymax(20,15)  
b <- mymax(31,45)  
print(a+b)
```

29/40

29

## 4. 사용자 정의 함수

```
> mymax <- function(x,y) {  
+   num.max <- x  
+   if (y > x) {  
+     num.max <- y  
+   }  
+   return(num.max)  
+ }  
>  
> mymax(10,15)  
[1] 15  
> a <- mymax(20,15)  
> b <- mymax(31,45)  
> print(a+b)  
[1] 65
```

30/40

30

## 4. 사용자 정의 함수

- 사용자 정의 함수의 매개변수에 **초깃값** 설정하기

코드4-16

```
mydiv <- function(x,y=2) {  
  result <- x/y  
  return(result)  
}  
  
mydiv(x=10,y=3)      # 매개변수 이름과 매개변수값을 쌍으로 입력  
mydiv(10,3)          # 매개변수값만 입력  
mydiv(10)            # x에 대한 값만 입력 (y 값이 생략됨)  
  
> mydiv <- function(x,y=2) {  
+   result <- x/y  
+   return(result)  
+ }  
>  
> mydiv(x=10,y=3)    # 매개변수 이름과 매개변수값을 쌍으로 입력  
[1] 3.333333  
> mydiv(10,3)        # 매개변수값만 입력  
[1] 3.333333  
> mydiv(10)          # x에 대한 값만 입력 (y 값이 생략됨)  
[1] 5
```

31/40

31

## 4. 사용자 정의 함수

- 함수가 반환하는 결과값이 여러 개일 때의 처리
  - 함수는 일반적으로 결과값을 하나만 반환(return)
  - 여러 개의 값을 반환해야 하는 경우는 list() 함수를 이용

코드4-17

```
myfunc <- function(x,y) {  
  val.sum <- x+y  
  val.mul <- x*y  
  return(list(sum=val.sum, mul=val.mul))  
}  
  
result <- myfunc(5,8)  
s <- result$sum      # 5, 8의 합  
m <- result$mul      # 5, 8의 곱  
cat('5+8=', s, '\n')  
cat('5*8=', m, '\n')
```

32/40

32



## 4. 사용자 정의 함수

```
> myfunc <- function(x,y) {  
+   val.sum <- x+y  
+   val.mul <- x*y  
+   return(list(sum=val.sum, mul=val.mul))  
+ }  
>  
> result <- myfunc(5,8)  
> s <- result$sum           # 5, 8의 합  
> m <- result$mul           # 5, 8의 곱  
> cat('5+8=', s, '\n')  
5+8= 13  
> cat('5*8=', m, '\n')  
5*8= 40
```

33/40

33

## 4. 사용자 정의 함수

### ■ 사용자 정의 함수의 저장 및 호출

- 사용자 정의 함수를 파일에 저장해 두었다가 필요할 때 불러서 사용할 수 있다.
- 다음의 함수를 작성하여 d:/source 폴더에 저장

```
mydiv <- function(x,y=2) {  
  result <- x/y  
  return(result)  
}
```

코드4-18

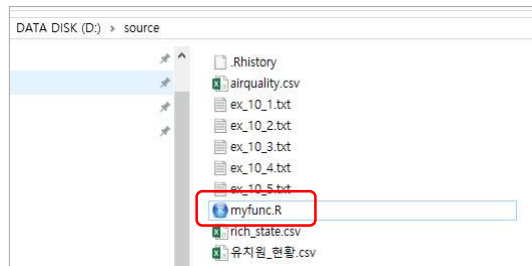
```
setwd("d:/source")           # myfunc.R이 저장된 폴더  
source("myfunc.R")           # myfunc.R 안에 있는 함수 실행  
  
# 함수 사용  
a <- mydiv(20,4)              # 함수 호출  
b <- mydiv(30,4)              # 함수 호출  
a+b  
mydiv(mydiv(20,2),5)          # 함수 호출
```

34/40

34

## 4. 사용자 정의 함수

```
> setwd("d:/source")      # myfunc.R이 저장된 폴더
> source("myfunc.R")      # myfunc.R 안에 있는 함수 실행
>
> # 함수 사용
> a <- mydiv(20,4)         # 함수 호출
> b <- mydiv(30,4)         # 함수 호출
> a+b
[1] 12.5
> mydiv(mydiv(20,2),5)     # 함수 호출
[1] 2
```



35/40

35

## 5. 조건에 맞는 데이터의 위치 찾기

- 데이터 분석을 하다보면 자신이 원하는 데이터가 벡터나 매트릭스, 데이터 프레임 안에서 어디에 위치하고 있는지를 알기 원하는 때가 있음.
- 예를 들어, 50명의 학생 성적이 저장된 벡터가 있는데 가장 성적이 좋은 학생은 몇 번째에 있는지를 알고 싶은 경우
- 이런 경우 편리하게 사용할 수 있는 함수가 `which()`, `which.max()`, `which.min()` 함수

36/40

36

## 5. 조건에 맞는 데이터의 위치 찾기

코드4-19

```
score <- c(76, 84, 69, 50, 95, 60, 82, 71, 88, 84)
which(score==69)      # 성적이 69인 학생은 몇 번째에 있나
which(score>=85)      # 성적이 85 이상인 학생은 몇 번째에 있나
max(score)            # 최고 점수는 몇 점인가
which.max(score)       # 최고 점수는 몇 번째에 있나
min(score)            # 최저 점수는 몇 점인가
which.min(score)       # 최저 점수는 몇 번째에 있나
```

```
> score <- c(76, 84, 69, 50, 95, 60, 82, 71, 88, 84)
> which(score==69)      # 성적이 69인 학생은 몇 번째에 있나
[1] 3
> which(score>=85)      # 성적이 85 이상인 학생은 몇 번째에 있나
[1] 5 9
> max(score)            # 최고 점수는 몇 점인가
[1] 95
> which.max(score)       # 최고 점수는 몇 번째에 있나
[1] 5
> min(score)            # 최저 점수는 몇 점인가
[1] 50
> which.min(score)       # 최저 점수는 몇 번째에 있나
[1] 5
```

37/40

37

## 5. 조건에 맞는 데이터의 위치 찾기

코드4-20

```
score <- c(76, 84, 69, 50, 95, 60, 82, 71, 88, 84)
idx <- which(score<=60)  # 성적이 60 이하인 값들의 인덱스
score[idx] <- 61         # 성적이 60 이하인 값들은 61 점으로 상향
score                    # 상향 조정된 성적 확인

idx <- which(score>=80)  # 성적이 80 이상인 값들의 인덱스
score.high <- score[idx] # 성적이 80 이상인 값들만 추출하여 저장
score.high               # score.high의 내용 확인
```

```
> score <- c(76, 84, 69, 50, 95, 60, 82, 71, 88, 84)
> idx <- which(score<=60)  # 성적이 60 이하인 값들의 인덱스
> score[idx] <- 61         # 성적이 60 이하인 값들은 61 점으로 상향
> score                    # 상향 조정된 성적 확인
[1] 76 84 69 61 95 61 82 71 88 84
>
> idx <- which(score>=80)  # 성적이 80 이상인 값들의 인덱스
> score.high <- score[idx] # 성적이 80 이상인 값들만 추출하여 저장
> score.high               # score.high의 내용 확인
[1] 84 95 82 88 84
```

38/40

38

## 5. 조건에 맞는 데이터의 위치 찾기

- 2차원 배열에서 조건에 맞는 행(row)의 인덱스를 알아내기

코드4-21

```
# 꽃잎의 길이가 5.0 이상인 값들의 행 인덱스
idx <- which(iris$Petal.Length>5.0)
idx
# 인덱스에 해당하는 값만 추출하여 저장
iris.big <- iris[idx,]
iris.big

> # 꽃잎의 길이가 5.0 이상인 값들의 행 인덱스
> idx <- which(iris$Petal.Length>5.0)
> idx
[1] 84 101 102 103 104 105 106 108 109 110 111 112 113 115 116
[16] 117 118 119 121 123 125 126 129 130 131 132 133 134 135 136
[31] 137 138 140 141 142 143 144 145 146 148 149 150
> # 인덱스에 해당하는 값만 추출하여 저장
> iris.big <- iris[idx,]
> iris.big
  Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
84          6.0         2.7          5.1         1.6 versicolor
101          6.3         3.3          6.0         2.5 virginica
102          5.8         2.7          5.1         1.9 virginica
103          7.1         3.0          5.9         2.1 virginica
104          6.3         2.9          5.6         1.8 virginica
105          6.5         3.0          5.8         2.2 virginica
106          7.6         3.0          6.6         2.1 virginica
108          7.3         2.9          6.3         1.8 virginica
109          6.7         2.5          5.8         1.8 virginica
```

39/40

39

## 5. 조건에 맞는 데이터의 위치 찾기

- 2차원 배열에서 조건에 맞는 행(row)과 열(column)의 인덱스를 알아내기

```
# 값의 크기가 7.0 이상인 값들의 행,열 인덱스
rowcol <- which(iris[,1:4] >= 7.0, arr.ind = T)
rowcol
```

```
> # 값의 크기가 7.0 이상인 값들의 행,열 인덱스
> rowcol <- which(iris[,1:4] >= 7.0, arr.ind = T)
> rowcol
      row col
[1,]  51   1
[2,] 103   1
[3,] 106   1
[4,] 108   1
[5,] 110   1
[6,] 118   1
[7,] 119   1
[8,] 123   1
[9,] 126   1
[10,] 130   1
[11,] 131   1
[12,] 132   1
[13,] 136   1
```

40/40

40