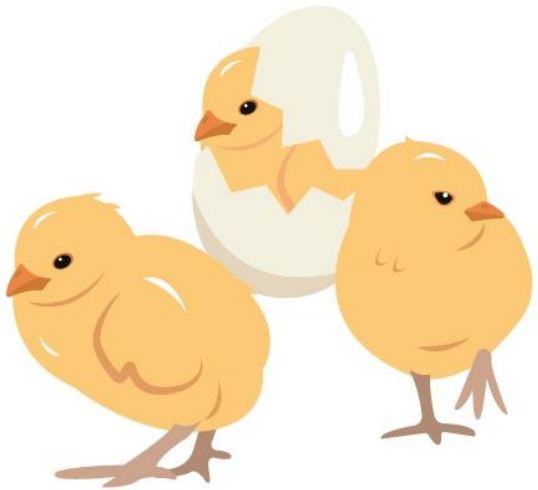
모두를 위한

R 데이터 분석 입문

2판



Chapter 02
변수와 벡터

목차

Section 01

R의 기본 연산

1. 산술연산과 주석

코드 2-1

```
2+3
(3+6)*8
2^3                              # 2의 세제곱
```

```
> 2+3
[1] 5
> (3+6)*8
[1] 72
> 2^3                            # 2의 세제곱
[1] 8
```

- 일반적으로 R에서는 한 줄에 하나의 명령문을 입력한다.
- 한 줄 내에서 # 이후의 내용은 주석으로 간주하여 실행하지 않는다.

1. R의 기본연산

표 2-1 산술연산자

연산자	의미	사용 예
+	덧셈	3+5+8
-	뺄셈	9-3
*	곱셈	7*5
/	나눗셈	8/3
%%	나눗셈의 나머지	8%%3
^	제곱	2^3

코드 2-2

```
7+4
# 2^3
```

```
> # 2^3
```

2. 산술연산 함수

코드 2-3

```
log(10)+5                    # 로그함수
sqrt(25)                     # 제곱근
max(5,3,2)                   # 가장 큰 값
```

```
> log(10)+5                  # 로그함수
[1] 7.302585
> sqrt(25)                   # 제곱근
[1] 5
> max(5,3,2)                 # 가장 큰 값
[1] 5
```

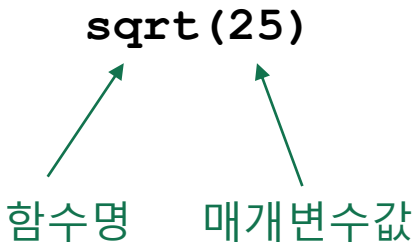


표 2-2 산술연산 함수

함수	의미	사용 예
log( )	로그함수	log(10), log(10, base=2)
sqrt( )	제곱근	sqrt(36)
max( )	가장 큰 값	max(3,9,5)
min( )	가장 작은 값	min(3,9,5)
abs( )	절댓값	abs(-10)
factorial( )	팩토리얼	factorial(5)
sin( ), cos( ), tan( )	삼각함수	sin(pi/2)

Section 02

변수

코드 2-4

```
a <- 10
b <- 20
c <- a+b
print(c)
```

```
> a <- 10
> b <- 20
> c <- a+b
> print(c)
[1] 30
```

1. 변수의 개념

- 프로그램에서 어떤 값을 저장하는 저장소나 보관 박스

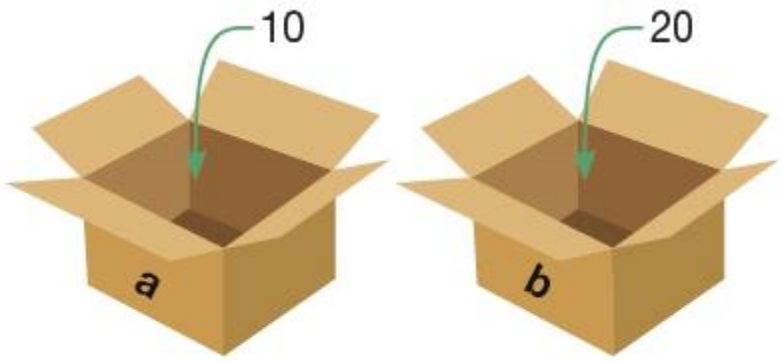


그림 2-1 변수의 개념: 변수, 변수명, 값

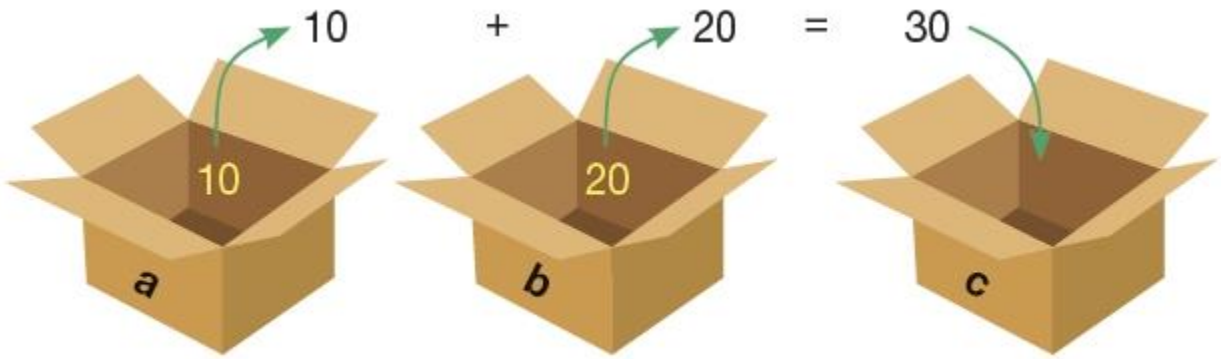


그림 2-2 c <- a + b의 실행 과정

2. 변수

- `a <- 10`

  10을 변수 a에 저장

- `b <- 20`

  20을 변수 b에 저장

- `c <- a+b`

  변수 a의 값과 변수 b의 값을 더하여 변수 c에 저장

- `print(c)`

  변수 c의 값을 출력('c'만 입력해도 출력)

2. 변수명 지정

❶ 첫 글자는 영문자(알파벳)나 마침표(.)로 시작하는데, 일반적으로 영문자로 시작

ex) avg, .avg

ex) 12th는 숫자로 시작했기 때문에 변수명 사용 불가

❷ 두 번째 글자부터는 영문자, 숫자, 마침표(.), 밑줄(_) 사용 가능

ex) v.1, a_sum, d10

ex) this-data, this@data은 변수명 사용 불가(@과 – 같은 특수문자 사용 불가)

❸ 대문자와 소문자를 구분

ex) var_A 와 var_a는 서로 다른 변수

❹ 변수명 중간에 빈칸을 넣을 수 없음

ex) first ds는 변수명 사용 불가

3. 변수에 값 저장 및 확인

```
a <- 10          # 권장
b = 20           # 권장하지 않음
```

코드 2-5

```
a <- 125
a
print(a)
```

```
> a <- 125
> a
[1] 125
> print(a)
[1] 125
```

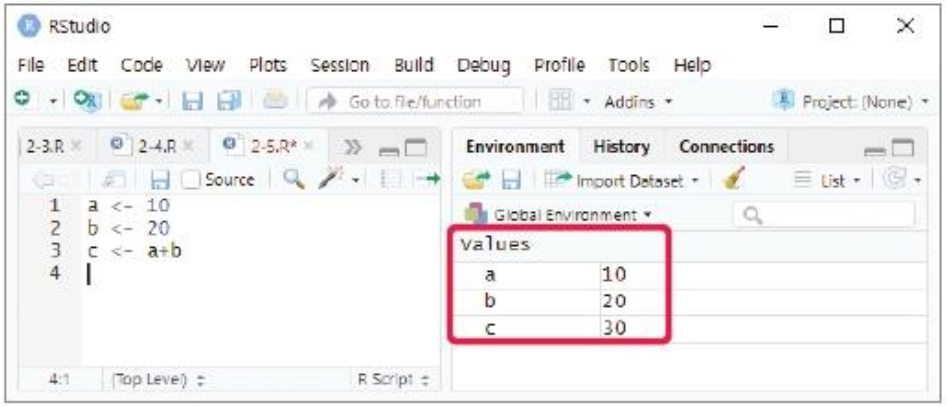


그림 2-3 R스튜디오의 환경 창에서 변수의 내용 확인하기

[Alt]와 '–'를 누르면,
'<-'가 입력

4. 변수의 자료형

표 2-3 R에서 사용할 수 있는 값의 자료형

자료형	사용 예	비고
숫자형	1, 2, 3, -4, 12.8	정수와 실수 모두 가능
문자형	'Tom', "Jane"	작은따옴표나 큰따옴표로 묶어서 표현
논리형	TRUE, FALSE	반드시 따옴표가 없는 대문자로 표기하며, T나 F로 줄여서 사용하는 것도 가능
특수값	NULL	정의되어 있지 않음을 의미하며, 자료형도 없고 길이도 0임
	NA	결측값(missing value)
	NaN	수학적으로 정의가 불가능한 값 예 sqrt(-3)
	Inf, -Inf	양의 무한대(Inf), 음의 무한대(-Inf)

5. 변수의 값 변경

- 변수에 저장된 값은 언제라도 변경 가능
- 변수의 자료형은 어떤 값을 저장하는가에 따라 유동적으로 바뀜

코드 2-6

```
a <- 10                    # a에 숫자 저장
b <- 20
a+b                        # a+b의 결과 출력
a <- "A"                   # a에 문자 저장
a+b                        # a+b의 결과 출력. 에러 발생
```

```
> a <- 10                  # a에 숫자 저장
> b <- 20
> a+b                      # a+b의 결과 출력
[1] 30
> a <- "A"                 # a에 문자 저장
> a+b                      # a+b의 결과 출력. 에러 발생
Error in a + b : non-numeric argument to binary operator
```

Section 03

벡터의 이해

1. 벡터의 개념

- 1차원 배열 데이터

- 1학년 학생들의 **몸무게** 자료
- 2학년 학생들의 **영어 성적** 자료
- 3학년 학생들의 **선호하는 색** 자료

몸무게	56	67	84	63	65	56	49	90

그림 2-4 **1차원 배열 데이터의 예: 몸무게**

- 2차원 배열 데이터

- 4학년 학생들의 전 과목 성적
- 국가별 GDP

전 과목 성적

이름	국어	영어	수학	물리	음악	체육	미술
김철수	90	85	60	75	65	100	90
전혜련	95	90	85	85	90	85	85
강재국	85	70	75	90	85	65	75
최원식	70	65	80	100	75	85	65
정대열	80	80	85	90	100	90	80

그림 2-5 2차원 배열 데이터의 예: 전 과목 성적

(a) 하나의 값이 저장된 변수 (b) 벡터가 저장된 변수

그림 2-6 변수와 벡터

2. 벡터 만들기

코드 2-7

```
x <- c(1,2,3)                          # 숫자형 벡터
y <- c("a","b","c")                    # 문자형 벡터
z <- c(TRUE,TRUE, FALSE, TRUE)         # 논리형 벡터
x                                      # x에 저장된 값을 출력
y
z
```

```
> x <- c(1,2,3)                        # 숫자형 벡터
> y <- c("a","b","c")                  # 문자형 벡터
> z <- c(TRUE,TRUE, FALSE, TRUE)       # 논리형 벡터
> x                                    # x에 저장된 값을 출력
[1] 1 2 3
> y
[1] "a" "b" "c"
> z
[1]  TRUE  TRUE FALSE  TRUE
```

코드 2-8

```
w <- c(1,2,3, "a","b","c")
w
```

```
> w <- c(1,2,3, "a","b","c")
> w
[1] "1" "2" "3" "a" "b" "c"
```

2.1 연속적인 숫자로 이루어진 벡터의 생성

코드 2-9

```
v1 <- 50:90
v1
v2 <- c(1,2,5, 50:90)
v2
```

```
> v1 <- 50:90
> v1
 [1] 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71
[23] 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90
> v2 <- c(1,2,5, 50:90)
> v2
 [1]  1  2  5 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68
[23] 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90
```

2.2 일정한 간격의 숫자로 이루어진 벡터 생성

코드 2-10

```
v3 <- seq(1,101,3)
v3
v4 <- seq(0.1,1.0,0.1)
v4
```

```
> v3 <- seq(1,101,3)
> v3
 [1]   1   4   7  10  13  16  19  22  25  28  31  34  37  40  43  46  49  52
[19]  55  58  61  64  67  70  73  76  79  82  85  88  91  94  97  100
> v4 <- seq(0.1,1.0,0.1)
> v4
 [1] 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

2.3 반복된 숫자로 이루어진 벡터 생성

코드 2-11

```
v5 <- rep(1,times=5)                    # 1을 5번 반복
v5
v6 <- rep(1:5,times=3)                  # 1에서 5까지 3번 반복
v6
v7 <- rep(c(1,5,9), times=3)            # 1, 5, 9를 3번 반복
v7
```

```
> v5 <- rep(1,times=5)                  # 1을 5번 반복
> v5
[1] 1 1 1 1 1
> v6 <- rep(1:5,times=3)                # 1에서 5까지 3번 반복
> v6
[1] 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
> v7 <- rep(c(1,5,9), times=3)          # 1, 5, 9를 3번 반복
> v7
[1] 1 5 9 1 5 9 1 5 9
```

3. 벡터의 원소값에 이름 지정

코드 2-12

```
score <- c(90,85,70)                          # 성적
score
names(score)                          # score에 저장된 값들의 이름을 보이시오
names(score) <- c("John","Tom","Jane") # 값들에 이름을 부여
names(score)                          # score에 저장된 값들의 이름을 보이시오
score                                         # 이름과 함께 값이 출력
```

```
> score <- c(90,85,70)                  # 성적
> score
[1] 90 85 70
> names(score)                          # score에 저장된 값들의 이름을 보이시오
NULL
> names(score) <- c("John","Tom","Jane")    # 값들에 이름을 부여
> names(score)                          # score에 저장된 값들의 이름을 보이시오
[1] "John" "Tom"  "Jane"
> score                                 # 이름과 함께 값이 출력
John  Tom Jane
  90   85   70
```

4. 벡터에서 원소값 추출

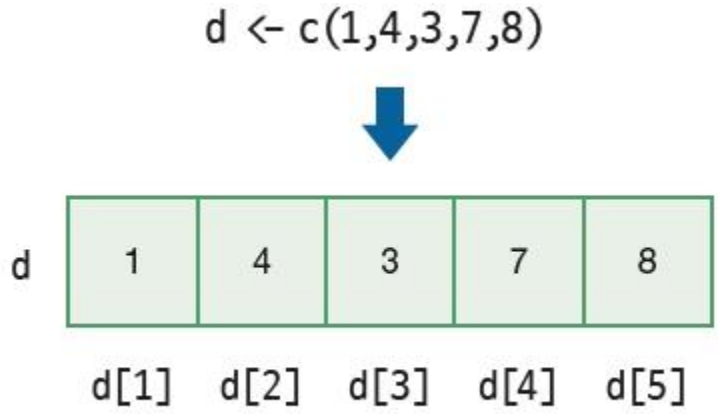


그림 2-7 벡터의 인덱스

코드 2-13

```
d <- c(1,4,3,7,8)
d[1]
d[2]
d[3]
d[4]
d[5]
d[6]
```

3. 벡터의 이해

```
> d <- c(1,4,3,7,8)
> d[1]
[1] 1
> d[2]
[1] 4
> d[3]
[1] 3
> d[4]
[1] 7
> d[5]
[1] 8
> d[6]
[1] NA
```

4.1 벡터에서 여러 개의 값을 한 번에 추출하기

코드 2-14

```
d <- c(1,4,3,7,8)
d[c(1,3,5)]                # 1, 3, 5번째 값 출력
d[1:3]                     # 처음 세 개의 값 출력
d[seq(1,5,2)]              # 홀수 번째 값 출력
d[-2]                      # 2번째 값 제외하고 출력
d[-c(3:5)]                 # 3~5번째 값은 제외하고 출력
```

```
> d <- c(1,4,3,7,8)
> d[c(1,3,5)]              # 1, 3, 5번째 값 출력
[1] 1 3 8
> d[1:3]                   # 처음 세 개의 값 출력
[1] 1 4 3
> d[seq(1,5,2)]            # 홀수 번째 값 출력
[1] 1 3 8
> d[-2]                    # 2번째 값 제외하고 출력
[1] 1 3 7 8
> d[-c(3:5)]               # 3~5번째 값은 제외하고 출력
[1] 1 4
```

4.2 벡터에서 이름으로 값을 추출하기

코드 2-15

```
GNP <- c(2090,2450,960)
GNP
names(GNP) <- c("Korea","Japan","Nepal")
GNP
GNP[1]
GNP["Korea"]
GNP[c("Korea","Nepal")]
```

```
> GNP <- c(2090,2450,960)
> GNP
[1] 2090 2450  960
> names(GNP) <- c("Korea","Japan","Nepal")
> GNP
Korea Japan Nepal
 2090  2450   960
```

3. 벡터의 이해

```
> GNP[1]
Korea
 2090
> GNP["Korea"]
Korea
 2090
> GNP[c("Korea","Nepal")]
Korea Nepal
 2090   960
```

5. 벡터에 저장된 원소값 변경

코드 2-16

```
v1 <- c(1,5,7,8,9)
v1
v1[2] <- 3                    # v1의 2번째 값을 3으로 변경
v1
v1[c(1,5)] <- c(10,20)        # v1의 1, 5번째 값을 각각 10, 20으로 변경
v1
```

```
> v1 <- c(1,5,7,8,9)
> v1
[1] 1 5 7 8 9
> v1[2] <- 3                  # v1의 2번째 값을 3으로 변경
> v1
[1] 1 3 7 8 9
> v1[c(1,5)] <- c(10,20)      # v1의 1, 5번째 값을 각각 10, 20으로 변경
> v1
[1] 10  3  7  8 20
```

Section 04

벡터의 연산

1. 벡터와 숫자값 연산

- 벡터에 대한 산술 연산은 벡터 안에 포함된 값들에 대한 연산으로 바뀌어 실행

코드 2-17

```
d <- c(1,4,3,7,8)
2*d
d-5
3*d+4
```

```
> d <- c(1,4,3,7,8)
> 2*d
[1]  2  8  6 14 16
> d-5
[1] -4 -1 -2  2  3
> 3*d+4
[1]  7 16 13 25 28
```

2. 벡터와 벡터 간의 연산

- 벡터 간의 대응되는 위치에 있는 값끼리의 연산으로 바꾸어 실행

코드 2-18

```
x <- c(1,2,3)
y <- c(4,5,6)
x+y                    # 대응하는 원소끼리 더하여 출력
x*y                    # 대응하는 원소끼리 곱하여 출력
z <- x + y             # x, y를 더하여 z에 저장
z
```

```
> x <- c(1,2,3)
> y <- c(4,5,6)
> x+y                  # 대응하는 원소끼리 더하여 출력
[1] 5 7 9
> x*y                  # 대응하는 원소끼리 곱하여 출력
[1]  4 10 18
> z <- x + y           # x, y를 더하여 z에 저장
> z
[1] 5 7 9
```

3. 벡터에 적용 가능한 함수

표 2-4 벡터에 적용 가능한 함수

함수명	설명
sum( )	벡터에 포함된 값들의 합
mean( )	벡터에 포함된 값들의 평균
median( )	벡터에 포함된 값들의 중앙값
max( ), min( )	벡터에 포함된 값들의 최댓값, 최솟값
var( )	벡터에 포함된 값들의 분산
sd( )	벡터에 포함된 값들의 표준편차
sort( )	벡터에 포함된 값들을 정렬(오름차순이 기본)
range( )	벡터에 포함된 값들의 범위(최솟값~최댓값)
length( )	벡터에 포함된 값들의 개수(길이)

코드 2-17

```
d <- c(1,2,3,4,5,6,7,8,9,10)
sum(d)                          # d의 포함된 값들의 합
sum(2*d)                        # d의 포함된 값들에 2를 곱한 후 합한 값
length(d)                       # d에 포함된 값들의 개수
mean(d[1:5])                    # 1~5번째 값들의 평균
max(d)                          # d에 포함된 값들의 최댓값
min(d)                          # d에 포함된 값들의 최솟값
sort(d)                         # 오름차순 정렬
sort(d, decreasing = FALSE)     # 오름차순 정렬
sort(d, decreasing = TRUE)      # 내림차순 정렬

v1 <- median(d)
v1
v2 <- sum(d)/length(d)
v2
```

4. 벡터의 연산

```
> d <- c(1,2,3,4,5,6,7,8,9,10)
> sum(d)                                # d의 포함된 값들의 합
[1] 55
> sum(2*d)                              # d의 포함된 값들에 2를 곱한 후 합한 값
[1] 110
> length(d)                             # d에 포함된 값들의 개수
[1] 10
> mean(d[1:5])                          # 1~5번째 값들의 평균
[1] 3
> max(d)                                # d에 포함된 값들의 최댓값
[1] 10
> min(d)                                # d에 포함된 값들의 최솟값
[1] 1
> sort(d)                               # 오름차순 정렬
 [1]  1  2  3  4  5  6  7  8  9 10
> sort(d, decreasing = FALSE)           # 오름차순 정렬
 [1]  1  2  3  4  5  6  7  8  9 10
> sort(d, decreasing = TRUE)            # 내림차순 정렬
 [1] 10  9  8  7  6  5  4  3  2  1
```

여기서 잠깐! 함수의 사용

수학시간에 배운 함수와 동일한 개념

함수에 어떤 입력값(input)을 주면 과정을 거쳐서 계산된 결과값(output)을 내는 구조

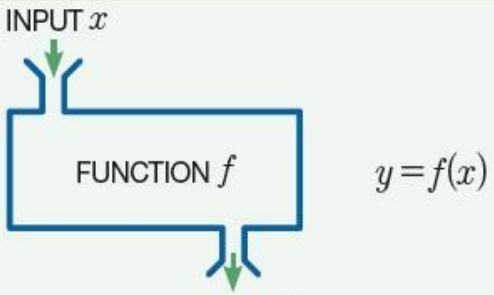


그림 2-8 함수의 개념

1. 매개변수의 입력

```
d <- c(1,2,3,4,5,6,7,8,9,10)
sort(x = d, decreasing = FALSE)
```

매개변수명 매개변수값

그림 2-9 함수와 매개변수

- 매개변수명 = 매개변수값과 같이 쌍으로 입력하는 것이 기본
- 매개변수명이 있는 이유는 매개변수값이 무엇인지를 명확히 설명할 수 있기 때문
- 매개변수명과 매개변수 값을 입력하는 여러가지 방법이 있음

여기서 잠깐! 함수의 사용

```
> sort(x = d, decreasing = TRUE)      # 정상 작동
> sort(x = d, TRUE)                   # 정상 작동
> sort(d, TRUE)                       # 정상 작동
> sort(TRUE, d)                       # 오류 발생 (매개변수의 순서가 잘못됨)
```

2. 매개변수의 생략

- 함수를 사용하기 위해서는 함수에 정의된 매개변수들에 대해 모두 대응하는 입력값을 주는 것이 원칙
- 매개변수값을 생략할 수 있는 경우가 있음 (초기값이 설정되어 있는 경우)
- 초기값(default value) : 사용자가 매개변수값을 입력하지 않으면 자동적으로 적용하도록 사전에 설정된 값

```
> sort(d)                             # 정상 작동
```

여기서 잠깐! 함수의 사용

3. 함수에 대한 도움말 확인

1) 편집창에서 함수명 선택 후 <F1> 키 입력

2) 파일 창의 HELP 에서 함수명 검색

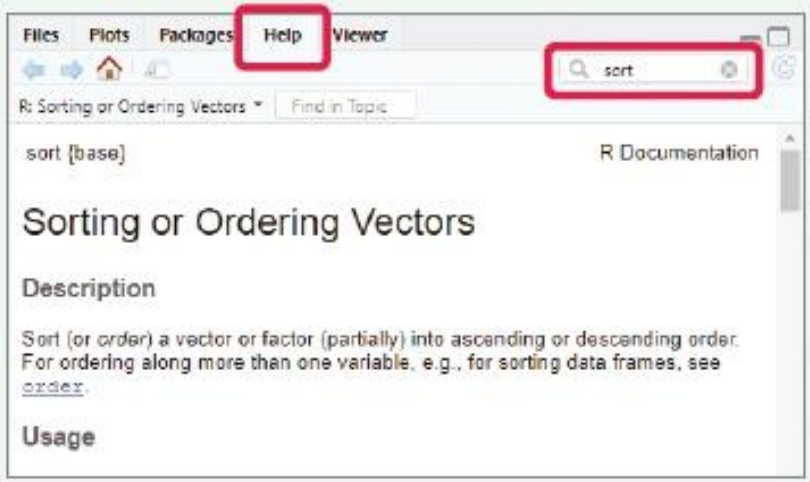


그림 2-10 R스튜디오의 도움말(Help) 창

4. 벡터에 논리연산자 적용

표 2-5 논리연산자

연산자	사용 예	설명
<	A < B	B가 A 보다 크면 TRUE
<=	A <= B	B가 A 보다 크거나 같으면 TRUE
>	A > B	A가 B 보다 크면 TRUE
>=	A >= B	A가 B 보다 크거나 같으면 TRUE
==	A == B	A와 B가 같으면 TRUE
!=	A != B	A와 B가 같지 않으면 TRUE
|	A | B	A 또는 B 어느 한쪽이라도 TRUE면 TRUE
&	A & B	A와 B 모두 TRUE일 때만 TRUE

코드 2-20

```
d <- c(1,2,3,4,5,6,7,8,9)
d>=5
d[d>5]                          # 5보다 큰 값
sum(d>5)                        # 5보다 큰 값의 개수를 출력
sum(d[d>5])                     # 5보다 큰 값의 합계를 출력
d==5

condi <- d > 5 & d < 8          # 조건을 변수에 저장
d[condi]                        # 조건에 맞는 값들을 선택
```

```
> d <- c(1,2,3,4,5,6,7,8,9)
> d>=5
[1] FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
> d[d>5]                        # 5보다 큰 값
[1] 6 7 8 9
[1] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE
```

4. 벡터의 연산

```
d     :  [1]  1     2     3     4     5     6     7     8     9
d>5   :  [1] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE
d[d>5] :                                    6     7     8     9
> sum(d>5)                    # 5 보다 큰 값의 개수를 출력
[1] 4
```

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

```
> sum(d[d>5])                 # 5 보다 큰 값의 합계를 출력
[1] 30
```

```
d>5          : [1] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE
d[d>5]       : [1] 6 7 8 9
sum(d[d>5]) : [1] 30
```

```
condi <- d > 5 & d < 8        # 조건을 변수에 저장
> d[condi]                    # 조건에 맞는 값들을 선택
[1] 6 7
```

Section 05

리스트와 팩터

1. 리스트

- 서로 다른 자료형의 값들을 1차원 배열에 저장하고 다룰 수 있도록 해주는 수단

코드 2-21

```
ds <- c(90, 85, 70, 84)
my.info <- list(name='Tom', age=60, status=TRUE, score=ds)
my.info                 # 리스트에 저장된 내용을 모두 출력
my.info[[1]]            # 리스트의 첫 번째 값을 출력
my.info$name            # 리스트에서 값의 이름이 name인 값을 출력
my.info[[4]]            # 리스트의 네 번째 값을 출력
```

```
> ds <- c(90, 85, 70, 84)
> my.info <- list(name='Tom', age=60, status=TRUE, score=ds)
> my.info                    # 리스트에 저장된 내용을 모두 출력
$name
[1] "Tom"

$age
[1] 60
```

```
$status
[1] TRUE

$score
[1] 90 85 70 84
> my.info[[1]]                # 리스트의 첫 번째 값을 출력
[1] "Tom"
> my.info$name                # 리스트에서 값의 이름이 name인 값을 출력
[1] "Tom"
> my.info[[4]]                # 리스트의 네 번째 값을 출력
[1] 90 85 70 84
```

리스트의 원소에 대한 인덱스를 표기할 때 [ ] 가 아닌 [[ ]]를 사용함에 주의한다

2. 팩터

- 벡터의 일종으로서 값의 종류가 정해져 있는 범주형 자료의 저장에 사용
- 범주형 자료의 예: 성별, 혈액형, 선호 정당, 선호 계절 등

코드 2-22

```
bt <- c('A', 'B', 'B', 'O', 'AB', 'A')        # 문자형 벡터 bt 정의
bt.new <- factor(bt)                          # 팩터 bt.new 정의
bt                                            # 벡터 bt의 내용 출력
bt.new                                        # 팩터 bt.new의 내용 출력
bt[5]                                         # 벡터 bt의 5번째 값 출력
bt.new[5]                                     # 팩터 bt.new의 5번째 값 출력
levels(bt.new)                                # 팩터에 저장된 값의 종류를 출력
as.integer(bt.new)                            # 팩터의 문자값을 숫자로 바꾸어 출력
bt.new[7] <- 'B'                              # 팩터 bt.new의 7번째에 'B' 저장
bt.new[8] <- 'C'                              # 팩터 bt.new의 8번째에 'C' 저장
bt.new                                        # 팩터 bt.new의 내용 출력
```

5. 리스트와 팩터

```
> bt <- c('A', 'B', 'B', 'O', 'AB', 'A')      # 문자형 벡터 bt 정의
> bt.new <- factor(bt)                        # 팩터 bt.new 정의
> bt                                          # 벡터 bt의 내용 출력
[1] "A"  "B"  "B"  "O"  "AB" "A"
> bt.new                                      # 팩터 bt.new의 내용 출력
[1] A  B  B  O  AB A
Levels: A AB B O
> bt[5]                                       # 벡터 bt의 5번째 값 출력
[1] "AB"
> bt.new[5]                                   # 팩터 bt.new의 5번째 값 출력
[1] AB
Levels: A AB B O
> levels(bt.new)                              # 팩터에 저장된 값의 종류를 출력
[1] "A"  "AB" "B"  "O"
> as.integer(bt.new)                          # 팩터의 문자값을 숫자로 바꾸어 출력
[1] 1 3 3 4 2 1
> bt.new[7] <- 'B'                            # 팩터 bt.new의 7번째에 'B' 저장
> bt.new[8] <- 'C'                            # 팩터 bt.new의 8번째에 'C' 저장
```

```
Warning message:
In `[<-.factor`(`*tmp*`, 8, value = "C") :
  invalid factor level, NA generated
> bt.new                                    # 팩터 bt.new의 내용 출력
[1] A    B    B    O    AB   A    B    <NA>
Levels: A AB B O
```

Thank you!

한빛아카데미 Hanbit Academy, Inc.