

Chapter 13-2

R 기초 (1) Vector

오 세 종

Contents



1. R 기본사용
 2. 변수(variable)
 3. 벡터(vector)
 4. 벡터 연산, 함수, 논리값
- [R 사용 Tip]

1. R 기본사용

- 계산기로 사용하기

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

- 사칙연산자 : +, -, *, /, ^
- 나머지 : %%
- 주석문 (comment) : #

- 함수 사용하기

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

- log(), sqrt(), max(), min(), ...

[연습 1]

- R 을 사용하여 다음의 계산식에 대한 해답을 구하시오

$$25+99$$

$$456 - 123$$

$$2 \times (3+4)$$

$$(3+5 \times 6) \div 7$$

$$(7 - 4) \times 3$$

$$2^{10} + 3^5$$

1256 을 7 로 나눈 나머지

184 를 5 로 나눈 나머지

$$1976 \div 24$$

$$16 \times 25 + 186 \times 5 - 67 \times 22$$

2. 변수(variable)

- 변수(variable) 사용하기

```
a <- 10
b <- 20
c <- a+b
c
```

c 에 저장된 값을 출력하라는 의미

- 프로그래밍 언어의 변수와 유사함
- 변수의 자료형(data type)은 지정하지 않는다
- 올바른 자료가 저장되었는를 검사하지 않으므로 주의
- 문자형 자료의 저장 : "" 또는 '' 이용

```
a <- 10 # a는 숫자저장 변수
b <- 20
a+b
a <- "A" # a는 문자저장 변수
a+b # 에러발생
d<-10; e<-15; f<-20 # 한줄에 여러 명령문을 입력
```

2. 변수(variable)

- 변수이름 규칙
 - 첫글자는 문자나 . (dot) 으로 시작
 - 그 이후에는 문자, 숫자, dot, underline 사용 가능
 - 대소문자를 구분한다.

```
avg <- 10
AVG <- 20
val.a <- 15
val.b <- 20
val.A <- 19
```

- 변수에 값을 할당
 - '`<-`' 사용 (`'='` 도 많이 사용)

```
var2 <- 15
```

2. 변수(variable)

- Note1. 한번 만들어 사용한 변수는 R 을 종료할 때 까지 사라지지 않는다

```
avg <- 10
AVG <- 20
val.a <- 15
val.b <- 20
val.A <- 19
...
print(avg)                # 변수의 내용 출력
```

- Note2. 하나의 변수는 다양한 유형의 값을 저장할 수 있다

```
V1 <- 10
V1                # 변수의 내용 출력
V1 <- "Good Morning"
V1                # 변수의 내용 출력
```

2. 변수(variable)

- Note3. Rstudio 에서 '<-' 쉽게 입력하기

Alt + -

- Script 창에서 Console 창으로 이동하기

Ctrl + 2

- Console 창에서 Script 창으로 이동하기

Ctrl + 1

2. 변수(variable)

- R 에서 사용할 수 있는 자료형(data type)
 - 숫자 : 1, 456, -123, 2.15
 - 문자 : "a", "b", "c", "hello", "good"
 - 논리형 : TRUE, FALSE (반드시 대문자. T, F 로 줄여 쓸 수 있다)
 - 특수한 값
 - NULL : 비어있는 값. 자료형도 없고 길이도 0
 - NA : 결측값 (missing value)
 - NaN : 수학적으로 정의가 불가능한 값 (예 : sqrt(-3))
 - Inf, -inf : 양의 무한대, 음의 무한대

2. 변수(variable)

```
A <- 10  
B <- "Good"  
C <- NULL
```

[연습 2]

- (문제1)
 - 반지름이 10, 15, 20 인 원의 면적을 구하시오

```
10 * 10 * 3.14  
15 * 15 * 3.14  
20 * 20 * 3.14
```

```
PI <- 3.14  
R <- 10  
R * R * PI  
R <- 15  
R * R * PI  
R <- 20  
R * R * PI
```

[연습 2]

- (문제2) 2차식 $y = 2x^2 + 5x + 10$ 에 대해 x 가 6,8,10 일 때 y 의 값을 각각 구하시오

3. 벡터(vector)

- 우리가 분석하고자 하는 데이터는 대부분 1차원 배열 또는 2차원 배열의 형태를 가지고 있다
- 1차원 배열 데이터
 - 1학년 학생들의 몸무게 자료
 - 2학년 학생들의 영어성적
 - 1학년 학생들의 선호하는 색깔 자료

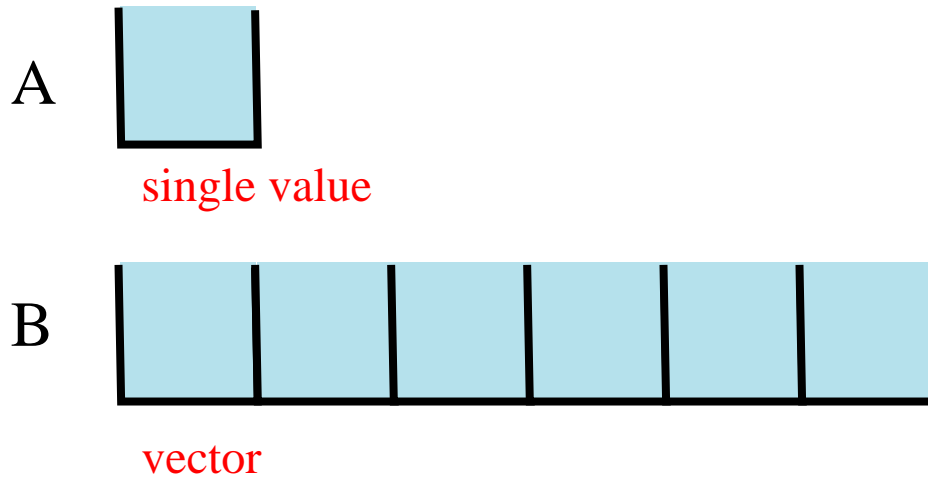
56	60	72	64	80	55	59	69	70
----	----	----	----	----	----	----	----	----

- 2차원 배열 데이터
 - 3학년 학생들의 전과목 성적 자료

92	2	1	1	2	0
36	2	2	1	1	0
105	3	2	1	1	0
81	1	2	1	1	0
94	1	1	2	1	0
20	1	1	3	3	0
50	1	2	1	1	0
68	3	3	2	1	0
89	3	1	3	2	0
19	3	2	1	3	0
118	2	1	2	1	0

3. 벡터(vector)

- 1차원 데이터를 저장하기 위한 자료 구조를 R에서는 벡터(vector)라고 한다
- 수학적 의미의 vector 와 다루는 방법이 동일
- 벡터 : 동일한 자료형의 값이 여러 개 연속되어 있음



3. 벡터(vector)

- 벡터(data vector) 만들기

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

- c() 함수를 이용해서 생성
- 하나의 벡터는 동일한 자료형의 값들만 포함해야 한다
- 하나의 벡터에 문자, 숫자를 섞어서 넣으면 모두 문자형으로 인식

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

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

3. 벡터(vector)

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

x

1	2	3
---	---	---

z

"a"	"b"	"c"
-----	-----	-----

3. 벡터(vector)

- 연속적인 숫자로 이루어진 벡터 만들기

```
v1 <- 50:90  
v1
```

```
> 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 72 73  
[25] 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
```

```
> 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  
[22] 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88  
[43] 89 90  
>
```

3. 벡터(vector)

- 일정 간격의 숫자로 구성된 벡터 만들기: seq()

```
v3 <- seq(1,101,3)      # 시작,종료,간격  
v3
```

```
> v3 <- seq(1,101,3)      # 시작,종료,간격  
> 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)   # 시작,종료,간격  
v4
```

```
> v4 <- seq(0.1,1.0,0.1)   # 시작,종료,간격  
> v4  
[1] 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

3. 벡터(vector)

- 반복값에 의한 벡터 만들기 : rep()

```
v5 <- rep(1, times = 5)      # 1 을 5번 반복  
v5
```

```
> v5 <- rep(1, times = 5)    # 1 을 5번 반복  
> v5  
[1] 1 1 1 1 1
```

```
v6 <- rep(1:5, times = 3)  
v6
```

```
> v6 <- rep(1:5, times = 3)  
> v6  
[1] 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
```

3. 벡터(vector)

```
v7 <- rep(c("a", "b", "c"), each = 3)  
v7
```

```
> v7 <- rep(c("a", "b", "c"), each = 3)  
> v7  
[1] "a" "a" "a" "b" "b" "b" "c" "c" "c"
```

```
v8 <- rep(c("a", "b", "c"), times = 3)  
v8
```

```
> v8 <- rep(c("a", "b", "c"), times = 3)  
> v8  
[1] "a" "b" "c" "a" "b" "c" "a" "b" "c"
```

3. 벡터(vector)

- 데이터 벡터는 요소 값에 이름 부여 가능
 - names() 함수 이용
 - 요소값에 이름을 붙여도 계산에는 아무 영향을 미치지 않음. 단지 값의 의미를 분명하게 밝히는 역할

```
score <- c(90, 85, 70)      # 성적
names(score) <- c("John", "Tom", "Jane")
score                       # 이름과 함께 값이 출력
```

```
> score <- c(90, 85, 70)
> names(score) <- c("John", "Tom", "Jane")
> score
John  Tom  Jane
  90   85   70
>
```

names(score)[2] ?

3. 벡터(vector)

- 데이터 벡터에서 값 추출하기
 - 한 개의 값 추출

```
d <- c(1,4,3,7,8)  
d[2] # 벡터에서 두번째 값
```

d	1	4	3	7	8
---	---	---	---	---	---

d[1] d[2] d[3] d[4] d[5]



index. R에서는 index 를 1부터 시작한다.

3. 벡터(vector)

- 데이터 벡터에서 값 추출하기

- 구간의 값 추출

```
d <- c(1,4,3,7,8)
d[1:3]           # 처음 세개의 자료 출력
d[c(1,3,5)]
d[seq(1,5,2)]    # 홀수번째 자료 출력
```

- Negative index

```
d <- c(1,4,3,7,8,9)
d[-2]           # - 는 '제외하고' 의 의미.
d[-c(3:5)]      # 세번째에서 다섯번째 값은 제외하고
```

3. 벡터(vector)

- 데이터 벡터에서 값 추출하기
 - 이름으로 값 추출하기

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

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


[연습3]

- 101 ~ 200 의 값으로 구성된 벡터 d 를 생성하시오

```
d <- 101:200
```

- d 에 어떤 값이 저장되었는지 확인하시오
- d 에서 10번째 값은 무엇인가
- d 에서 뒤에서 10개의 값을 잘라내어 보이시오
- d 에서 짝수만 출력하시오
- d 에서 앞에서 20 개의 값을 잘라내어 d.20 에 저장하시오. d.20 의 값을 보이시오
- d.20 에서 5번째 값을 제외한 나머지 값들을 보이시오
- d.20 에서 5,7,9 번째 값을 제외한 나머지 값을 보이시오

4. 벡터 연산, 함수, 논리값

- 벡터에 대한 산술 연산

```
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
>
```

4. 벡터 연산, 함수, 논리값

- 데이터 벡터간 연산
 - 두 벡터의 연결

```
x <- c(1,2,3)
y <- c(4,5)
c(x,y)          # 단순히 x,y 를 연결하여 출력
z <- c(x,y)     # x,y 를 연결하여 z에 저장
```

- 두 벡터의 합 (두 벡터의 길이가 같아야 됨. 다르면?)

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

(사칙연산 모두 적용가능)

4. 벡터 연산, 함수, 논리값

- 데이터 벡터에 적용 가능한 함수들

함수명	설명
sum()	자료의 합
mean()	자료의 평균
median()	자료의 중앙값
max(), min()	자료의 최대, 최소값
var()	자료의 분산 값
sd()	자료의 표준편차
sort()	자료를 정렬하여 출력
range()	자료의 범위 (최대값 ~ 최소값)
length()	자료의 개수

4. 벡터 연산, 함수, 논리값

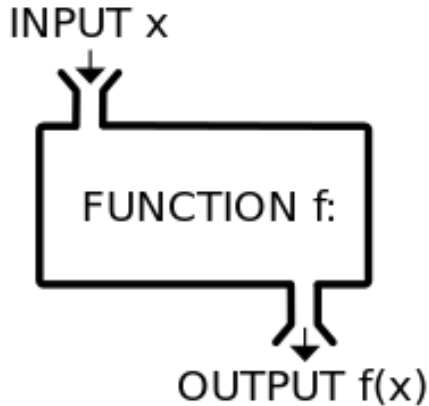
- 데이터 벡터에 적용 가능한 함수들

```
d <- c(1,2,3,4,5,6,7,8,9,10)
sum(d)
sum(2*d)
length(d)
mean(d[1:5])
max(d)
min(d)
sort(d) # 올림차순 정렬
sort(d, decreasing = FALSE) # 올림차순 정렬
sort(d, decreasing = TRUE) # 내림차순 정렬

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

4. 벡터 연산, 함수, 논리값

- 함수(function)



$$y = f(x)$$

[https://en.wikipedia.org/wiki/Function_\(mathematics\)](https://en.wikipedia.org/wiki/Function_(mathematics))

```
d <- c(1,2,3,4,5,6,7,8,9,10)
```

```
sort(d, decreasing = FALSE)
```

매개변수(parameter)
이름

매개변수(parameter)
값

4. 벡터 연산, 함수, 논리값

- 함수(function) 의 호출

함수명(매개변수명1=값1, 매개변수명2=값2)

```
v1 <- c(4,2,3,1,6,10,8,9)

sort(x = v1, decreasing = TRUE)
sort(v1, FALSE)
sort(v1)                                # decreasing = FALSE
```

```
> sort(x = v1, decreasing = TRUE)
[1] 10  9  8  6  4  3  2  1
> sort(v1, FALSE)
[1]  1  2  3  4  6  8  9 10
> sort(v1)                                # decreasing = FALSE
[1]  1  2  3  4  6  8  9 10
```

[연습4]

- d1, d2 가 다음과 같을 때 질문에 답하시오

```
d1 <- 1:50  
d2 <- 51:100
```

- d1, d2 의 값을 보이시오
- d1+d2, d2-d1, d1*d2, d2/d1 의 결과를 각각 보이시오
- d1, d2 의 값들의 합(sum)을 각각 보이시오
- d1, d2 에 있는 모든 값들의 합(sum)을 보이시오
- d2 에서 가장 큰 값과 가장 작은 값을 보이시오
- d2 와 d1 의 값들의 평균값을 각각 구하고 두 평균의 차이를 보이시오
- d1 의 값들을 큰수에서 작은 수 순으로 정렬하여 보이시오
- d1 과 d2 에서 큰수 순으로 각각 10개씩을 추출하여 d3 에 저장하시오 (결과적으로 d3 에는 20개의 수가 저장)

4. 벡터 연산, 함수, 논리값

- 논리연산자 : <, <=, >, >=, ==, !=, | (or), & (and)

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

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

Note.

- (1) 프로그래밍 언어에서 equal 연산자는 = 가 아닌 ==
- (2) 논리값이 계산에 사용되면 TRUE 는 1, FALSE 는 0 으로 취급

[연습5]

- 다음과 같이 벡터 v1 을 생성하시오

```
v1 <- 51:90
```

- 1) v1 에서 60 보다 작은 수 들을 보이시오
- 2) v1 에서 70 보다 작은 수 들은 몇개인지 보이시오
- 3) v1 에서 65 보다 큰 수들의 합을 보이시오
- 4) v1 에서 60보다 크고 73 보다 작은 수들을 보이시오
- 5) v1 에서 65 보다 작거나 80 보다 큰 수들을 보이시오
- 6) v1 에서 7로 나눈 나머지가 3 인 숫자들만 보이시오
- 7) v1 에서 짝수들의 합계를 보이시오
- 8) v1 에서 홀수이거나 80 보다 큰 수를 보이시오
- 9) v1 에서 3과 5의 공배수를 보이시오

R 사용 Tip: help

- help 기능

- 함수의 사용법 알고 싶을 때 (함수 이름을 알면)

```
help(sum)                # help("sum") 도 가능  
? sum
```

- 함수의 사용법 알고 싶을 때 (함수 이름을 모르면)

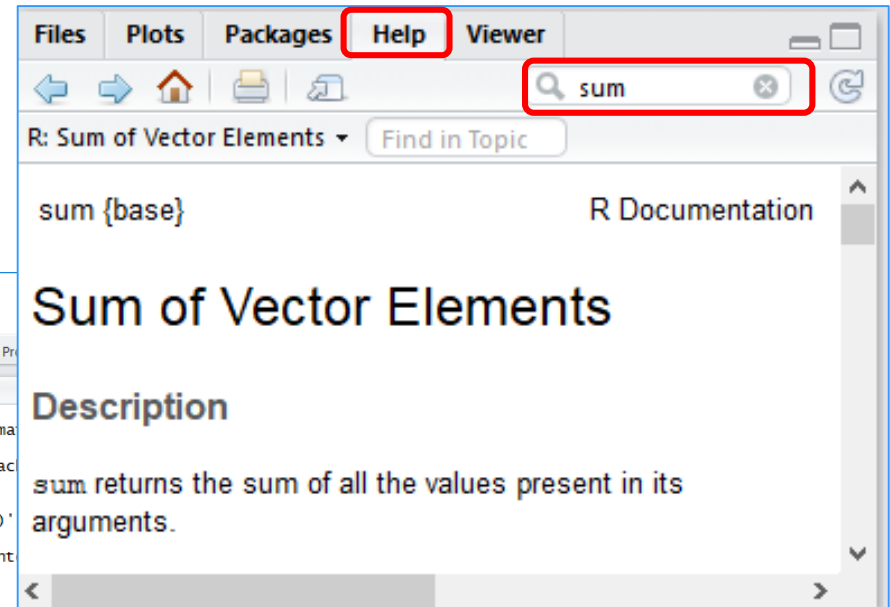
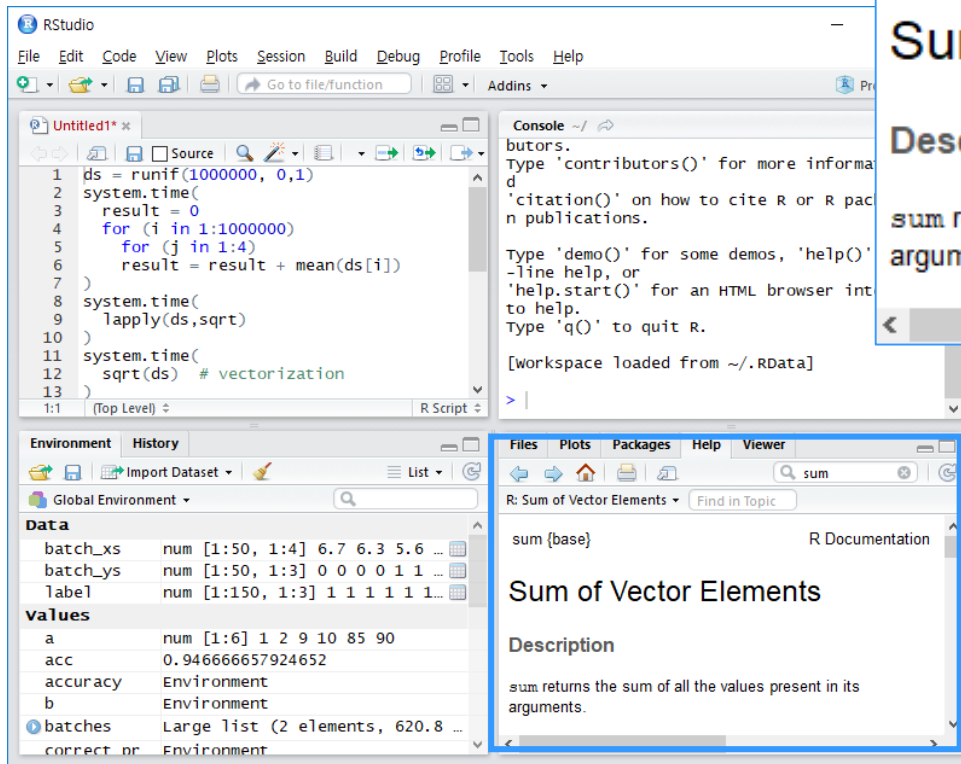
```
help.search("average")
```

- history()

- R은 최근에 사용한 명령어를 25개까지 기억
- 위아래 화살표 키를 이용해서 이전에 사용한 명령어를 불러올 수 있다
- history() 함수를 이용하여 25개의 목록을 한눈에 확인 가능

R 사용 Tip : help

- Rstudio 에서 help 사용



```
sum {base}
```

Package 이름

Sum of Vector Elements

Description

`sum` returns the sum of all the values present in its arguments.

Usage

```
sum(..., na.rm = FALSE)
```

Arguments ← 함수의 parameter 값 설명

...
numeric or complex or logical vectors.

`na.rm`
logical. Should missing values (including `NaN`) be removed?

Details ← 함수에 대한 상세 설명

This is a generic function: methods can be defined for it directly or via the [Summary](#) group generic. For this to work properly, the arguments ... should be unnamed, and dispatch is on the first argument.

If `na.rm` is `FALSE` an `NA` or `NaN` value in any of the arguments will cause a value of `NA` or `NaN` to be returned, otherwise `NA` and `NaN` values are ignored.

Logical true values are regarded as one, false values as zero. For historical reasons, `NULL` is accepted and treated as if it were `integer(0)`.

Loss of accuracy can occur when summing values of different signs: this can even occur for sufficiently long integer inputs if the partial sums would cause integer overflow. Where possible extended-precision accumulators are used, but this is platform-dependent.

Value ← 함수의 return 값

The sum. If all of ... are of type integer or logical, then the sum is integer, and in that case the result will be NA (with a warning) if integer overflow occurs. Otherwise it is a length-one numeric or complex vector.

NB: the sum of an empty set is zero, by definition.

S4 methods

This is part of the S4 [Summary](#) group generic. Methods for it must use the signature `x, ..., na.rm`.

'[plotmath](#)' for the use of `sum` in plot annotation.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

[colSums](#) for row and column sums.

Examples ← 함수의 사용 예제

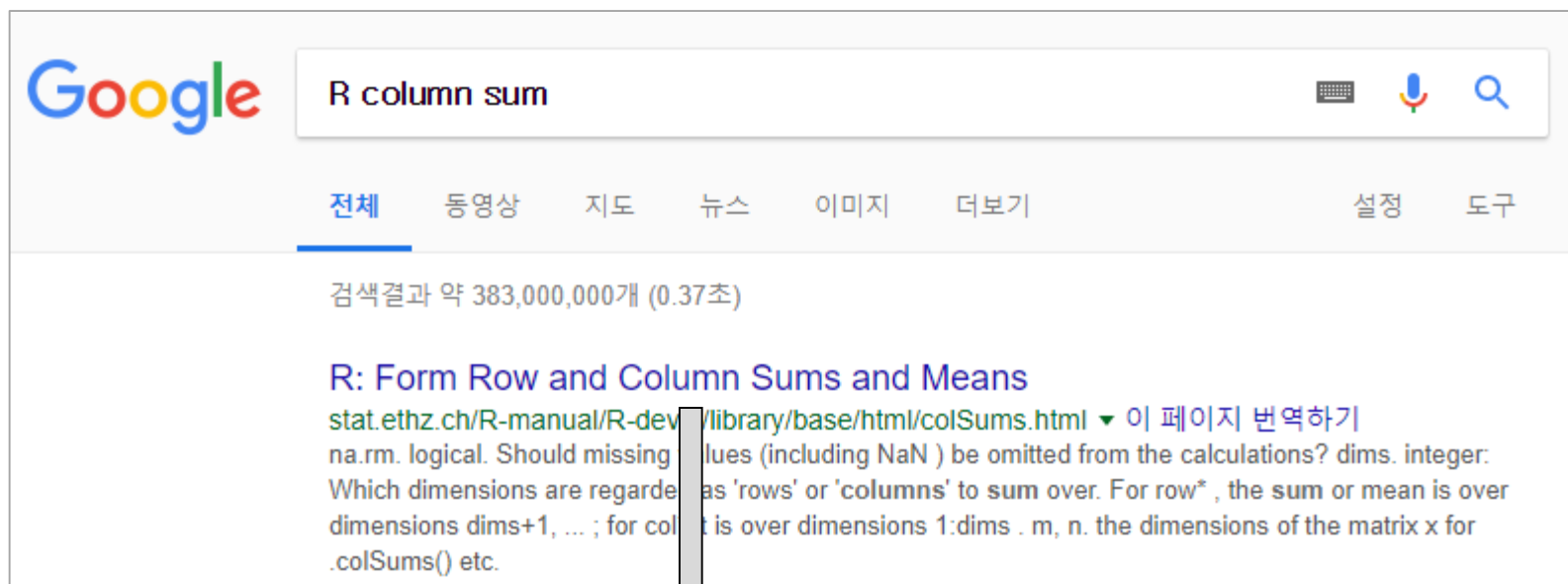
```
## Pass a vector to sum, and it will add the elements together.
sum(1:5)

## Pass several numbers to sum, and it also adds the elements.
sum(1, 2, 3, 4, 5)

## In fact, you can pass vectors into several arguments, and everything gets added.
sum(1:2, 3:5)

## If there are missing values, the sum is unknown, i.e., also missing, ....
sum(1:5, NA)
## ... unless we exclude missing values explicitly:
sum(1:5, NA, na.rm = TRUE)
```

구글에서 정보 검색



A screenshot of a Google search interface. The search bar contains the text "R column sum". Below the search bar, there are tabs for "전체" (All), "동영상" (Videos), "지도" (Maps), "뉴스" (News), "이미지" (Images), "더보기" (More), "설정" (Settings), and "도구" (Tools). The "전체" tab is selected. Below the tabs, it says "검색결과 약 383,000,000개 (0.37초)". The first search result is titled "R: Form Row and Column Sums and Means" and includes a link to "stat.ethz.ch/R-manual/R-devel/library/base/html/colSums.html". A large grey arrow points from this search result down to the next block.

`colSums {base}`

Form Row and Column Sums and Means

Description

Form row and column sums and means for numeric arrays (or data frames).

Usage

```
colSums(x, na.rm = FALSE, dims = 1)
rowSums(x, na.rm = FALSE, dims = 1)
colMeans(x, na.rm = FALSE, dims = 1)
rowMeans(x, na.rm = FALSE, dims = 1)
```

```
.colSums(x, m, n, na.rm = FALSE)
.rowSums(x, m, n, na.rm = FALSE)
.colMeans(x, m, n, na.rm = FALSE)
.rowMeans(x, m, n, na.rm = FALSE)
```