R 기초 프로그래밍

호서대학교 빅데이터경영공학부 연규필

R 기초 프로그래밍

1. R의 계산
2. 벡터 (vector)
3. 리스트
4. 행렬
5. 데이터 프레임
6. 데이터 유형 판별 및 변환
7. 제어문과 사용자 정의 함수
8. 결측치 처리

■ 스칼라 (scalar)

- 스칼라는 단일 차원의 값을 의미하며 길이가 1인 벡터로 볼 수 있음.
- 숫자 스칼라

```
> a <- -3
> b <- -4
> a+b
[1] -7
```

- 문자열 스칼라 : 어떠한 따옴표로 묶어도 무관함

```
> c <- "Hello"
> c
[1] "Hello"
```

■ NA, NaN, NULL

- NA: not available, 결측치

```
> (x <- NA)
[1] NA
> is.na(x)
[1] TRUE
```

- NaN: not a number

```
> 0/0
[1] NaN
> Inf - Inf
[1] NaN
> is.nan(c(1:2, 0/0, 3))
[1] FALSE FALSE  TRUE FALSE
```

- NULL은 어떤 형식도 취하지 않는 특별한 객체(object)임

■ R의 연산자 역할

- R은 수학계산기(calculator)의 기능을 제공한다.

< 주요 연산자 >

연산자	기능
성분추출	$
첨자표현	[ ] [[ ]]
사칙연산	+ - * /
제곱 연산자	^ **
몫 나머지 연산자	%/% %%
행렬의 곱	%*%
비교 연산자	< > <= >= == !=
부정 연산자	!
논리 연산자	& |
할당 연산자	<- = ->

1. R의 계산

■ R의 연산자 역할

- 간단한 연산

```
> 4+5
[1] 9
> 4/5
[1] 0.8
> 3^4
[1] 81
> 10+3*2-8
[1] 8
```

- 비교 및 논리 연산

```
> x=1:15
> x
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
> x[2<=x & x<=10]
[1]  2  3  4  5  6  7  8  9 10
> x[2<x & x<10]
[1] 3 4 5 6 7 8 9
```

- 성분추출

```
> x=10:15
> x
[1] 10 11 12 13 14 15
> x[x>13]=50
> x
[1] 10 11 12 13 50 50
```

■ 함수를 이용한 계산

- R은 다양한 내장된 수학적 통계적 함수(function)을 가지고 있다.

< 수학함수>

수학 함수	기능
abs(x)	• X의 절대값
ceiling(x), floor(x), trunc(x)	• x보다 큰 수 중 가장 작은 정수 • x보다 작은 수 중 가장 큰 정수 • 0과 x 사이의 가장 큰 정수를 출력한다
round(x, digits=y)	• x의 반올림 함수로 각각 x의 소수점 y자리에서의 반올림(디폴트 y=0), 10의 지수 형태의 표현으로 반올림(디폴트 y=6) 한다.
exp(x)	• 지수함수(exponential function)
log(x), log10(x), log2(x), log(x, base=y)	• 로그함수로 각각 밑(base)이 자연대수 e, 10, 2, x인 로그함수
sign(x)	• 부호함수 (±)
sqrt(x)	• 제곱근 함수
factorial(x)	• x의 계승(階乘) 출력(x!)
choose(x,y)	• x에서 y를 뽑는 조합의 수
pi	• 원주율
sin(x)	• sinx
cos(x)	• cosx
tan(x)	• tanx

■ 함수를 이용한 계산

▪ 함수를 이용한 연산

```
> pi                # 내장함수 3.141593
[1] 3.141593
> sin(5*pi/6)       # 삼각함수 sin
[1] 0.5
> sqrt(5)           # 루트
[1] 2.236068
> log(5, base=10)   # 밑이 10인 로그
[1] 0.69897
> log(10, base=5)   # 밑이 5인 로그
[1] 1.430677
> log(100)          # 밑이 e 인 로그
[1] 4.60517
> abs(-50)          # 절대값
[1] 50
> factorial(6)      # 6!
[1] 720
> choose(10, 7)     # 조합(combination)
[1] 120
```

```
> a= c(-3.465, 5.789, 4.275, -2.676)
> ceiling(a)                    # 올림
[1] -3  6  5 -2
> floor(a)                      # 내림
[1] -4  5  4 -3
> trunc(a)                      # 버림
[1] -3  5  4 -2
> round(a, digit=2)             #반올림
[1] -3.46  5.79  4.28 -2.68
```

2. 벡터 (vector)

■ c 함수를 이용한 벡터 생성

- 벡터는 여러 개의 관측치를 원소(element)로 갖는다.
- 벡터로 저장하거나 결합하고자 할 때 c 함수를 이용한다.
  - 1반 10명의 학생들의 수학 시험점수가 다음과 같다.

    98, 90, 86, 76, 78, 85, 88, 93, 82, 90
  - 2반 10명의 학생들의 수학 시험점수가 다음과 같다.

    87, 93, 96, 91, 87, 85, 92, 88, 94, 97
  - 1반과 2반 학생 20명의 성적을 하나로 합쳐 저장하고자 한다.

```
> m_score1 = c(98, 90, 86, 76, 78,  85, 88, 93, 82, 90)        # 데이터 할당
> m_score1
 [1] 98 90 86 76 78 85 88 93 82 90
> m_score2 = c(87, 93, 96, 91, 87,  85, 92, 88, 94, 97)        # 데이터 할당
> m_score2
 [1] 87 93 96 91 87 85 92 88 94 97
> score =c(m_score1,m_score2)                                   # 데이터 병합
> score
 [1] 98 90 86 76 78 85 88 93 82 90 87 93 96 91 87 85 92 88 94 97
```

2. 벡터 (vector)

■ 수열(sequence)에 의한 벡터 생성

- seq, rep 함수, n:m 표현식 이용

< 연속된 숫자 또는 반복으로 구성된 벡터 관련 함수>

함수	기능
seq(from, to, by)	• from부터 to까지의 값을 by 간격으로 저장한 숫자 벡터를 반환함.
seq_along(along.with)	• 반환 값은 along.with의 길이가 N일 때, 1부터 N까지의 숫자를 저장한 벡터임.
from:end	• from부터 end까지의 숫자를 저장한 벡터를 반환함.
rep(x, times, each)	• 반환값은 반복된 값이 저장된 x와 같은 타입의 객체임.
rev(from, to, by)	• from부터 to까지의 값을 by 간격으로 역으로 숫자 벡터를 반환함.

- 연속적인 정수 자료 생성

```
> x = seq(1,10)
> x
 [1]  1  2  3  4  5  6  7  8  9 10
> x=1:10
> x
 [1]  1  2  3  4  5  6  7  8  9 10
```

- 간격을 둔 연속자료 생성

```
> seq(from=1, to=10, by=2)
[1] 1 3 5 7 9
> seq(1,10,2)
[1] 1 3 5 7 9
> seq(5,30,by=3)
[1]  5  8 11 14 17 20 23 26 29
```

2. 벡터 (vector)

■ 수열(sequence)에 의한 벡터 생성

- 반복된 자료 생성

```
> rep(1,5)
[1] 1 1 1 1 1
> rep(seq(-3,3,by=2),3)
 [1] -3 -1  1  3 -3 -1  1  3 -3 -1  1  3
> rep(c(2,5,8),4)
 [1] 2 5 8 2 5 8 2 5 8 2 5 8
> x=rep(0,7)
> x
[1] 0 0 0 0 0 0 0
> rep(1:2, times=5)
 [1] 1 2 1 2 1 2 1 2 1 2
> rep(1:2, each=5)
 [1] 1 1 1 1 1 2 2 2 2 2
> rep(1:2, each=5, times=5)
[1] 1 1 1 1 1 2 2 2 2 2 1 1 1 1 1 2 2 2 2 2
1 1 1 1 1 2 2 2 2 2 1 1 1 1 1 2 2 2 2 2 1 1
1 1 1 2 2 2 2
[50] 2
```

- 어떤 수를 역으로 생성

```
> rev(1:10)
 [1] 10  9  8  7  6  5  4  3  2  1
> 10:1
 [1] 10  9  8  7  6  5  4  3  2  1
```

- seq_along ( ) 함수

```
> x=c(2,4,6,8,10)
> 1:NROW(x)
[1] 1 2 3 4 5
> seq_along(x)
[1] 1 2 3 4 5
```

2. 벡터 (vector)

■ 벡터의 형태

- R의 기본 데이터 형태는 벡터(vector) 이며, 숫자형(numeric), 문자형(character), 논리값(logical value: TRUE, FALSE) 등의 데이터 형태가 있다.
- 벡터를 만들 때 한 가지 제한 점은 같은 형태의 데이터 값을 가져야 한다는 것이다.
- 문자형 데이터는 “ ” 또는 ‘ ’ 부호를 사용하여 입력한다.

• **숫자형 데이터 벡터 만들기**

```
> z = c(7,19,27,35,97)          # numeric
> z
[1]  7 19 27 35 97
```

• **문자형 데이터 벡터 만들기**

```
> family=c("Min","Ko","Jiyong","Sangho")                  # character
> names(family)=c("father","mother","daughter","son")     # 이름(name) 달기
> family
  father   mother daughter      son
   "Min"     "Ko" "Jiyong" "Sangho"
```

2. 벡터 (vector)

■ 벡터의 형태 (Type)

- 문자형과 숫자형을 섞어 저장하면 모두 문자형으로 인식된다.

```
> y=c("Min",95,97,90)          # 모두 문자형으로 인식됨
> y
[1] "Min" "95"  "97"  "90"
```

- 논리형 데이터 벡터 만들기

```
> logic1 = c(T,F,F,F,T,F,F)      # 논리형 벡터
> logic1
[1]  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE
> x = (-4: 3)                    # -4에서 3까지 정수 생성
> x
[1] -4 -3 -2 -1  0  1  2  3
> v = (x<1)                      # 논리연산자로 사용 논리형벡터 생성
> v
[1]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE
> sum(v)                  # 논리형 데이터인경우 TRUE이면 1, FALSE이면 0으로 계산함
[1] 5
```

2. 벡터 (vector)

■ 데이터 형태의 변환

- 필요에 따라 데이터 또는 데이터 객체의 형태(type)를 변환시킬 수 있다.

< 데이터 유형 변환 함수>

다음으로 변환	변환 함수	변환규칙
numeric	as.numeric	FALSE -> 0 TRUE -> 1 “1”,”2”,…-> 1,2,… “A” -> NA
logical	as.logical	0 -> FALSE 그 외 다른 수 -> TRUE “FALSE”,”F” -> FALSE “TRUE”,”T” -> TRUE 그 외 다른 문자 -> NA
character	as.character	1,2,… -> “1”,”2”,… FALSE -> “FALSE” TRUE -> “TRUE”
factor	as.factor	범주형 factor 형식으로
vector	as.vector	vector 형식으로
matrix	as.matrix	matrix 형식으로
data.frame	as.data.frame	데이터프레임

2. 벡터 (vector)

■ 데이터 형태의 변환

- 예제

```
> Min = c(10,20,30,40)                  # 숫자형(numeric) 벡터
> Min
[1] 10 20 30 40
> Min1 = as.factor(Min)                 # 범주형(factor) 벡터로 변환
> Min1
[1] 10 20 30 40
Levels: 10 20 30 40
> M.log = as.logical(Min)               # 논리형(logical) 벡터로 변환
> M.log
[1] TRUE TRUE TRUE TRUE
> sub.fact = factor(c("Kor","Math","Eng","Eng","Kor","Math"))    # 범주형(factor) 벡터생성
> sub.fact
[1] Kor  Math Eng  Eng  Kor  Math
Levels: Eng Kor Math
> sub.fact1 =as.numeric(sub.fact)    # 숫자형(nemeric) 벡터로 변환
> sub.fact1
[1] 2 3 1 1 2 3
```

2. 벡터 (vector)

■ 기초 통계량 계산 함수

< 기초 통계량 계산 함수 >

함수	내용	함수	내용
ave, mean	평균	median	중앙값
var	분산	sd	표준편차
sum	합계	range, IQR	범위, 사분위범위
weighted.mean	가중평균	quantile	사분위수
min, max	최소값, 최대값	rank	순위
summary, fivenum	다섯수치요약	scale	표준화

```
> stat = c(87,85,86,96,78,83,89,95,92,68)
> sum(stat)
[1] 859
> mean(stat)    # 평균
[1] 85.9
> max(stat)     # 최대값
[1] 96
> min(stat)     # 최소값
[1] 68
> range(stat)   # 범위
[1] 68 96
```

```
> var(stat)     # 분산
[1] 69.43333
> sd(stat)      # 표준편차
[1] 8.332667
> median(stat)  # 중앙값
[1] 86.5
> rank(stat)    # 순위
 [1]  6  4  5 10  2  3  7  9  8  1
> summary(stat) # 다섯수치요약
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  68.00   83.50   86.50   85.90   91.25   96.00
```

2. 벡터 (vector)

■ 벡터의 이름과 길이

- names: 객체의 이름
- length, NROW: 벡터의 길이 (참고) nrow( )는 행렬에서 사용함

함수	기능
names(x)	• 객체의 이름을 반환함.
names(x) <-	• 객체에 이름을 저장함.
length(x)	• 객체의 길이를 반환함.
NROW(x):	• 배열의 행 또는 열의 수를 반환함.

```
> c(1,2,3,c(1,2,3))
[1] 1 2 3 1 2 3
> x<-c(1,3,4)
> names(x) <- c("kim","seo","park")
> x
 kim  seo park
   1    3    4
```

```
> length(x)
[1] 3
> NROW(x)
[1] 3
> nrow(x)
NULL
```

2. 벡터 (vector)

■ 벡터 인덱싱 (indexing)

- 벡터의 데이터를 접근하는 데에는 색인을 사용하는 방법이 있음
- 벡터의 각 셀(cell)에 이름이 부여되어 있다면 이름을 사용할 수 있음
- 벡터에서 특정 요소를 제외한 나머지 데이터를 가져오거나 동시에 여러 셀의 데이터에 접근이 가능함

< 벡터 데이터 접근 문법>

문법	기능
x[n]	• 벡터 x의 n번째 요소. n은 숫자 또는 셀의 이름을 뜻하는 문자열
x[-n]	• 벡터 x의 n번째 요소를 제외한 나머지. n은 숫자 또는 셀의 이름을 뜻하는 문자열
x[idx_vector]	• 벡터 x로부터 idx_vector에 지정된 요소를 가져옴. 이때 idx_vector는 색인을 표현하는 숫자 벡터 또는 셀의 이름을 뜻하는 문자열 벡터임
x[start:end]	• 벡터 x의 start부터 end까지의 값을 반환함. Start 위치의 값과 end위치의 값을 모두 포함함

```
> x<-c("a","b","c")
> x[1]
[1] "a"
> x[3]
[1] "c"
> x[-1]            #  첫번째 원소 제외
[1] "b" "c"
```

```
> x[-2]
[1] "a" "c"
> x[c(1,2)]      # 첫번째 원소와 세번째 원소만 선택
[1] "a" "b “
> x[c(1,3)]
[1] "a" "c"
```

2. 벡터 (vector)

■ 벡터 인덱싱 (indexing)

- [ ]로 표현하여 선택

```
> x=c(50,58,65,70,72,77,80,84,91,94,100,101)
> x[1]                                  # 벡터 x의 첫번째 값
[1] 50
> x[9]                                  # 벡터 x의 9번째 값
[1] 91
> x[1:3]                                # 벡터 x의 1~3번째 값
[1] 50 58 65
> x[c(6,7,9)]                           # 벡터 x의 6,7,9번째 값
[1] 77 80 91
> names(x)=seq(1,12)                    # 벡터 x의 이름 생성 1~12
> x
  1   2   3   4   5   6   7   8   9  10  11  12
 50  58  65  70  72  77  80  84  91  94 100 101
> x1=x[-1]                              # 벡터 x의 1번째 값을 제외한 나머지 벡터
> x1
  2   3   4   5   6   7   8   9  10  11  12
 58  65  70  72  77  80  84  91  94 100 101
```

2. 벡터 (vector)

■ 벡터 인덱싱 (indexing)

- [ ]로 표현하여 선택

```
> x2=x[-c(2,5,6)]                    # 벡터 x의 2,5,6번째 값을 제외한 나머지 벡터
> x2
  1   3   4   7   8   9  10  11  12
 50  65  70  80  84  91  94 100 101
> x3=x[x<=77]                        # 벡터 x의 개별값이 77이하인 벡터
> x3
 1  2  3  4  5  6
50 58 65 70 72 77
> x4=x[x>80]                         # 벡터 x의 개별값이 88초과인 벡터
> x4
  8   9  10  11  12
 84  91  94 100 101
> x5=x[x!=77]                        # 벡터 x의 개별값이 77이 아닌 벡터
> x5
  1   2   3   4   5   7   8   9  10  11  12
 50  58  65  70  72  80  84  91  94 100 101
```

2. 벡터 (vector)

■ 벡터 인덱싱 (indexing) 및 비교

```
> v <- c(3, pi, 4)
> w <- c(pi, pi, pi)
> v == w
[1] FALSE  TRUE FALSE
> v[v == w]
[1] 3.141593
> v[v <= w]
[1] 3.000000 3.141593
> any(v == pi)
[1] TRUE
> all(v == pi)
[1] FALSE
> new.v <- c(v, NA)
> new.v[!is.na(new.v)]
[1] 3.000000 3.141593 4.000000
```

2. 벡터 (vector)

■ 벡터 연산자

< 벡터 연산자 함수>

벡터 연산자	기능
value %in% x	• 벡터 x에 value가 저장되어 있는지 판단함.
x+n	• 벡터 x의 모든 요소에 n을 더한 벡터를 구함. • (마찬가지로 *, /, == 등의 연산자도 적용 가능함.
identical(x,y)	• 두 벡터가 같은 값을 담고 있는지를 확인함.
union(x,y)	• 벡터 x와 벡터 y의 합집합을 반환함.
intersect(x,y)	• 벡터 x와 벡터 y의 교집합을 반환함.
setdiff(x,y)	• 벡터 x와 벡터 y의 차집합을 반환함.
setequal(x,y)	• 벡터 x와 벡터 y가 같은 집합인지를 판단함.

2. 벡터 (vector)

■ 벡터 연산자

```
> identical(c(1,2,3),c(1,2,3))
[1] TRUE
> identical(c(1,2,3),c(1,2,100))
[1] FALSE
> "a" %in% c("a","b","c")
[1] TRUE
> "d" %in% c("a","b","c")
[1] FALSE
> x<-c(1,2,3,4,5)
> x+1
[1] 2 3 4 5 6
> 10-x
[1] 9 8 7 6 5
> c(1,2,3)==c(1,2,100)
[1]  TRUE  TRUE FALSE
```

```
> c(1,2,3)!=c(1,2,100)
[1] FALSE FALSE  TRUE
> union(c("a","b","c"), c("a","d"))
[1] "a" "b" "c" "d"
> intersect(c("a","b","c"), c("a","d"))
[1] "a "
> setdiff(c("a","b","c"), c("a","d"))
[1] "b" "c"
> setequal(c("a","b","c"), c("a","d"))
[1] FALSE
> setequal(c("a","b","c"), c("a","b","c","d"))
[1] FALSE
> setequal(c("a","b","c"), c("a","b","c","c"))
[1] TRUE
```

■ 문자열(character string), 문자열 벡터(character vector)

- nchar 함수: 문자열의 길이를 출력함
- substr 함수: 문자열의 일부를 추출함
- paste 함수: 벡터를 문자열로 변환한 후에 연결함
- paste0 함수: paste함수와 비슷한 역할을 수행하지만 sep이 항상 빈 문자열임

```
> c1 <- "Kim Chulsoo"
> nchar(c1)
[1] 11
> substr(c1, 1, 3)
[1] "Kim"
> paste("Today is", date())
[1] "Today is Wed Jan 24 15:49:04 2018"
> paste("Name :", c1)
[1] "Name : Kim Chulsoo"
> paste("1st", "2nd", "3rd", sep = ", ")
[1] "1st, 2nd, 3rd"
> paste("1st", "2nd", "3rd", sep = "")
[1] "1st2nd3rd"
```

```
> paste("1st", "2nd", "3rd")
[1] "1st 2nd 3rd “
> paste0("1st", "2nd", "3rd")
[1] "1st2nd3rd"
```

■ 리스트 (list)

- 리스트는 벡터와는 달리 서로 다른 데이터 타입을 담을 수 있음
- 리스트를 활용하여 성분 형태가 동일하지 않은 벡터로 자료구조를 만들 수 있음

```
> Hong <- list(name = "홍길동",age = 40,
    married = T, no.child = 2, child.ages = c(3,7))
> Hong
$name
[1] "홍길동"

$age
[1] 40

$married
[1] TRUE

$no.child
[1] 2

$child.ages
[1] 3 7
```

```
> Hong$age
[1] 40
> Hong$name
[1] "홍길동"
> Hong$child.ages
[1] 3 7
> Hong[[2]]   # 리스트의 2번째 원소만 추출
[1] 40
> Hong[c(1, 2)]
$name
[1] "홍길동"

$age
[1] 40

> is.list(Hong)
[1] TRUE
```

■ 리스트 (list) 데이터 접근법

- list에 저장된 데이터는 색인 또는 키를 사용해 접근할 수 있다.

< list 데이터 접근 문법>

문법	의미
x$key	• list x에서 키 값 key에 해당하는 값
x[n]	• list x에서 n번째 데이터의 서브리스트
x[[n]]	• list x에서 n번째 저장된 값

```
> x=list(name='foo', height=c(1,3,5))
> x$name
[1] "foo"

> x$height
[1] 1 3 5

> x[1]
$name
[1] "foo"
```

```
> x[2]
$height
[1] 1 3 5

> x[[1]]
[1] "foo"

> x[[2]]
[1] 1 3 5
```

■ 행렬 만들기

- 행렬은 행(row), 열(column)의 수가 지정된 구조임
- matrix 함수를 이용하여 행렬을 생성
- matrix(data=, nrow= , ncol= , byrow=)

```
> x = 1:12
> matrix(x,nrow=3,byrow=TRUE)   #  행의 수가 3이고 행우선인 행렬생성
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
> matrix(x,nrow=3)              #  행의 수가 3이고 열우선인 행렬생성
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> matrix(1:12, nrow = 3, byrow = TRUE, dimnames = list(c("r1","r2", "r3"),
         c("c1", "c2", "c3", "c4")))
   c1 c2 c3 c4
r1  1  2  3  4
r2  5  6  7  8
r3  9 10 11 12
```

■ 행렬 만들기

- 여러 개의 벡터를 가지고 하나의 행렬을 생성할 수 있다.

```
> A=c(100,150,120)                                  # A회사 데이터
> B=c(90,80,66)                                     # B회사 데이터
> total = matrix(c(A,B),nrow=2, byrow=TRUE)         #c(A,B)로 결합된 벡터로 행렬생성
> dimnames(total)[[1]]=c("A회사","B회사")            # 행별이름
> dimnames(total)[[2]]=c("1분기","2분기","3분기")    # 열별이름
> total
      1분기 2분기 3분기
A회사   100   150   120
B회사    90    80    66
```

■ 행렬 만들기

- 문자형 자료는“ ” 또는‘ ’를 사용한다.

```
> names1 = c(" 즐거운", " 데이터분석")      # 문자형 벡터 생성
> names2 = c(" 보람찬", "R 프로그래밍")       # 문자형 벡터 생성
> name = c(names1, names2)                 # 문자형 벡터 병합
> name
[1] " 즐거운" " 데이터분석"   " 보람찬"  "R 프로그래밍"
> data = c("Top", "Mid", "Bottom")         # 문자형 벡터 생성
> data
[1] "Top"    "Mid"    "Bottom"
> rep(data, c(4,1,2))
[1] "Top"    "Top"    "Top"    "Top"    "Mid"    "Bottom" "Bottom"
```

■ 행렬의 인덱싱

- 이름이 부여되어 있다면 이름을 사용해서 자료를 가져올 수 있음.

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

```
> rownames(x) <- c("r1", "r2")
> colnames(x) <- c("c1", "c2", "c3", "c4")
> x
   c1 c2 c3 c4
r1  1  3  5  7
r2  2  4  6  8
> x["r1", ]
c1 c2 c3 c4
 1  3  5  7
> x[, c("c1", "c3")]
   c1 c3
r1  1  5
r2  2  6
```

■ 행렬 관련 함수

행렬 관련 함수

함수	내용
nrow(x), ncol(x)	• 행의 수, 열의 수
dim(x)	• 객체의 차원 수
t(x)	• 전치행렬
cbind(...)	• 열을 더해 행렬을 생성하는 함수
rbind(...)	• 행을 더해 행렬을 생성하는 함수
diag(x)	• 대각행렬
det(x)	• 행렬식
apply(x, m, fun)	• 행(m=1) 또는 열(m=2)에 함수(fun)를 적용함
ginv(x)	• 역 행렬을 구함, library(MASS)사용해야 한다.
solve(a, b)	• 수식 a%*%x = b 를 구한다. b를 지정하지 않으면 a의 역행렬을 구함
svd(x)	• Singular Value Decomposition
qr(x)	• qr Decomposition (QR 분해)
eigen(x)	• Eigenvalues(고유값)
chol(x)	• choleski decomposition (촐레스키 분해)

4. 행렬

■ 행렬 관련 함수

```
> x=matrix(c(1:9), nrow=3)
> X
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> t(x)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
> x=matrix(c(1,2,3,4), ncol=2)
> x
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> solve(x)
     [,1] [,2]
[1,]   -2  1.5
[2,]    1 -0.5
```

```
> x%*%solve(x)
     [,1] [,2]
[1,]    1    0
[2,]    0    1
> (x=matrix(c(1:6), ncol=3))
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> nrow(x)
[1] 2
> ncol(x)
[1] 3
> dim(x)
[1] 2 3
> dim(x) = c(3,2)
> x
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

■ 행렬 관련 함수

- rbind, cbind
- t 함수 : 전치행렬 (transpose matrix), 행과 열을 바꾸어 줌

```
> a=c(5,6,7)
> b=c(1,2,3)
> c=c(4,5,6)
> rbind(a,b,c)      # 행별로 결합
  [,1] [,2] [,3]
a    5    6    7
b    1    2    3
c    4    5    6
> cbind(a,b,c)     # 열별로 결합
     a b c
[1,] 5 1 4
[2,] 6 2 5
[3,] 7 3 6
> total=cbind(total,c(110,88))
       # 열벡터 (110,88) 추가
> dimnames(total)[[2]][4]=c("4분기")
       # 추가 열에 이름 생성(열의 4번째)
```

```
> total
     1분기 2분기 3분기 4분기
A회사  100   150   120   110
B회사   90    80    66    88
> total=rbind(total,c(110,90,88,100))
> dimnames(total)[[1]][3]=c("C회사")
       # 추가 행에 이름 생성(행의 3번째)
> total
     1분기 2분기 3분기 4분기
A회사  100   150   120   110
B회사   90    80    66    88
C회사  110    90    88   100
> t_total = t(total); t_total      # 전치행렬
     A회사 B회사 C회사
1분기  100    90   110
2분기  150    80    90
3분기  120    66    88
4분기  110    88   100
```

4. 행렬

■ 벡터와 행렬의 기본 연산

```
> x=seq(10,50,by=10)
> x
[1] 10 20 30 40 50
> y=seq(1,5,by=1)
> y
[1] 1 2 3 4 5
> x+y                                  # 벡터 합
[1] 11 22 33 44 55
> t(x)%*%y                             # 행렬곱
     [,1]
[1,]  550
> x%*%t(y)                             # 행렬곱
     [,1] [,2] [,3] [,4] [,5]
[1,]   10   20   30   40   50
[2,]   20   40   60   80  100
[3,]   30   60   90  120  150
[4,]   40   80  120  160  200
[5,]   50  100  150  200  250
> x*y                                  # 원소들간 벡터곱
[1]  10  40  90 160 250
> x*x                                  # 원소들간 벡터곱
[1]  100  400  900 1600 2500
```

```
> vec = c(x,y)                         # 벡터 병합(x,y)
> vec
 [1] 10 20 30 40 50  1  2  3  4  5
> mat2 = cbind(x,y)                    # 열별로 벡터 병합
> mat2
      x y
[1,] 10 1
[2,] 20 2
[3,] 30 3
[4,] 40 4
[5,] 50 5
> mat3=rbind(x,y)                      # 행별로 벡터 병합
> mat3
  [,1] [,2] [,3] [,4] [,5]
x   10   20   30   40   50
y    1    2    3    4    5
> mat3*mat3                            # 원소들간 행렬곱
  [,1] [,2] [,3] [,4] [,5]
x  100  400  900 1600 2500
y    1    4    9   16   25
> dim(mat3)                            # 행렬의 차원
[1] 2 5
```

■ 역행렬

- library(MASS)의 ginv 함수를 이용한다.

```
> library(MASS)
> mat = matrix(c(1,2,3,3,2,1,0,-1,3), nrow=3)
> mat
     [,1] [,2] [,3]
[1,]    1    3    0
[2,]    2    2   -1
[3,]    3    1    3
> inv = ginv(mat)                  # ginv함수를 이용한 역행렬
> inv
      [,1]  [,2]  [,3]
[1,] -0.35  0.45  0.15
[2,]  0.45 -0.15 -0.05
[3,]  0.20 -0.40  0.20
> inv2 = solve(mat)                # solve 함수를 이용한 역행렬
> inv2
      [,1]  [,2]  [,3]
[1,] -0.35  0.45  0.15
[2,]  0.45 -0.15 -0.05
[3,]  0.20 -0.40  0.20
```

■ 행렬에 함수 적용

- apply 함수를 이용한다.

```
> data = seq(1,16,by=1)
> mat =matrix(data, nrow=4, byrow=TRUE)
> mat
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
[4,]   13   14   15   16
> apply(mat,1,FUN=sum)            # 행기준으로 sum 함수 적용
[1] 10 26 42 58
> apply(mat,2,FUN=sum)            # 열기준으로 sum 함수 적용
[1] 28 32 36 40
> apply(mat,1,FUN=mean)           # 행기준으로 mean 함수 적용
[1]  2.5  6.5 10.5 14.5
> apply(mat,2,FUN=mean)           # 열기준으로 mean 함수 적용
[1]  7  8  9 10
> apply(mat,1,FUN=max)            # 행기준으로 max 함수 적용
[1]  4  8 12 16
> apply(mat,2,FUN=max)            # 열기준으로 max 함수 적용
[1] 13 14 15 16
```

5. 데이터 프레임

■ 데이터프레임 개념

- 데이터프레임은 R에서 데이터 저장 방식 중 가장 널리 사용되는 방식임
- Data frame 은 엑셀의 스프레드시트와 같이 표의 형태로 된 데이터 또는 테이블이다.
- str 함수를 통해 구조를 살펴볼 수 있음

< data frame 생성 및 관련 함수>

함수	기능
data.frame( )	• data frame을 생성함.
str(x)	• 임의의 R 객체에서 내부 구조(structure)를 보인다.
d$colname	• 데이터 프레임 d에서 컬럼 이름이 colname인 데이터를 접근한다.
d$colname <- y	• 데이터 프레임 d에서 컬럼 이름이 colname인 컬럼에 데이터 y를 저장한다. • 만약 colname이 d에 없는 새로운 이름이면 새로운 컬럼이 추가된다.

```
> d = data.frame(x=c(1:5), y=seq(2, 10, by=2),
  z=c('M','F','M','F','M'))
> d
  x  y z
1 1  2 M
2 2  4 F
3 3  6 M
4 4  8 F
5 5 10 M
```

```
> str(d)
'data.frame':    5 obs. of  3 variables:
 $ x: int  1 2 3 4 5
 $ y: num  2 4 6 8 10
 $ z: Factor w/ 2 levels "F","M": 2 1 2 1 2
> d$w = c('A','B','C','D', 'D') ; d
> str(d)
> d[,1:2]
```

■ 데이터프레임의 특징

- 데이터프레임의 각 컬럼은 서로 다른 데이터 타입일 수 있음

```
> x <- c(100, 75, 80)
> y <- c("10452", "10455", "10565")
> z <- data.frame(score = x, ID = y)
> z
  score    ID
1   100 10452
2    75 10455
3    80 10565
```

- 다른 형태의 데이터에서는 불가능한 접근 방식

```
> z$score
[1] 100  75  80
> z$ID
[1] 10452 10455 10565
Levels: 10452 10455 10565
```

■ 데이터프레임의 특징

- 컬럼 추가도 가능

```
> z$name <- c("Mike", "Tony", "Evan")
> z
  score    ID name
1   100 10452 Mike
2    75 10455 Tony
3    80 10565 Evan
```

- colnames( ) 함수를 통해 데이터 변수 명을 얻을 수 있음

```
> colnames(z)
[1] "score" "ID"    "name"
```

- str( ) 함수를 통해 구조를 살펴볼 수 있음

```
> str(z)
'data.frame':         3 obs. of  3 variables:
 $ score: num  100 75 80
 $ ID   : Factor w/ 3 levels "10452","10455",..: 1 2 3
 $ name : chr  "Mike" "Tony" "Evan"
```

■ data frame의 접근

< data frame 접근 문법>

함수	기능
d[m, n, drop=TRUE]	• 데이터 프레임 d 의 m행 n컬럼에 저장된 데이터를 말함. 색인뿐만 아니라 행이름이나 컬럼이름을 지정할 수 있음. 만약, m, n 중 하나를 생략하면 모든 행 또는 컬럼의 데이터를 의미한다. drop=FALSE를 지정하면 형태 변환을 하지 않음.

```
> z[1, 2]
[1] 10452
Levels: 10452 10455 10565
> z[1, ]
  score    ID name
1   100 10452 Mike
> z[, c("name", "ID")]
  name    ID
1 Mike 10452
2 Tony 10455
3 Evan 10565
```

```
>d=data.frame(x=c(1:5),y=seq(2,10,2)); d
  x  y
1 1  2
2 2  4
3 3  6
4 4  8
5 5 10
> d[1,]
  x y
1 1 2
> d[1,2]
[1] 2
> d[c(1,3),2]
[1] 2 6
```

■ data frame의 접근

```
> d[-1,-2]
[1] 2 3 4 5

> d[,c("x","y")]
  x  y
1 1  2
2 2  4
3 3  6
4 4  8
5 5 10

> d[,c("x")]
[1] 1 2 3 4 5
> d[,c("x"), drop=FALSE]
  x
1 1
2 2
3 3
4 4
5 5
```

```
> d = data.frame(a=1:3, b=4:6, c=7:9)
> d
  a b c
1 1 4 7
2 2 5 8
3 3 6 9

> d[,names(d) %in% c("b","c")]
  b c
1 4 7
2 5 8
3 6 9

> d[,!names(d) %in% c("b","c")]
[1] 1 2 3
```

■ data frame 유틸리티 함수

< data frame을 보기 위한 유틸리티 함수>

함수	기능
head(x,n)	• x의 앞부분을 n만큼 잘라낸 데이터이다. Default=6
tail(x,n)	• x의 뒷부분을 n만큼 잘라낸 데이터이다. Default=6
veiw(x,title)	• 데이터 뷰어를 호출한다.

```
> d = data.frame(a=1:3, b=4:6, c=7:9)
> d
  a b c
1 1 4 7
2 2 5 8
3 3 6 9

> head(d, 2)
  a b c
1 1 4 7
2 2 5 8
```

```
> tail(d,2)
  a b c
2 2 5 8
3 3 6 9
```

■ 자료의 입력과 출력

▪ scan, write 함수

- scan 함수는 데이터를 벡터로 읽어 들이고 write 함수는 행렬을 저장한다.

```
> x = scan( "D:/data/데이터벡터.txt")
> x2 = scan( "D:₩₩data₩₩데이터벡터.txt")
> x
 [1] 8 8 8 5 2 5 4 4 1 2 3 6 7 7 7 3 9 4 6 2 3 1 2 1
> y=matrix(scan( "D:/data/데이터벡터.txt"),ncol=4,byrow=TRUE)
Read 24 items
> y
     [,1] [,2] [,3] [,4]
[1,]    8    8    8    5
[2,]    2    5    4    4
[3,]    1    2    3    6
[4,]    7    7    7    3
[5,]    9    4    6    2
[6,]    3    1    2    1
> sale=c(100,150,120,110,90,80,66,88,110,90,88,100)
> total = matrix(sale,nrow=3, byrow=TRUE)
> dimnames(total)[[1]]=c("A회사","B회사","C회사")              # 행별이름
> dimnames(total)[[2]]=c("1분기","2분기","3분기","4분기")     # 열별이름
> write(total, "D:/data/실적.txt")
```

5. 데이터 프레임

■ 자료의 입력과 출력

▪ Text 파일 입출력

- text 파일을 데이터 프레임으로 읽으려면 read.table 함수, 데이터 프레임을 text로 저장하려면 write.table 함수를 사용한다.

함수	의미
read.table("file", header = FALSE, sep=" ", skip=0, nrows = -1, na.strings="NA", stringsAsFactors=FALSE)	• Text 파일을 데이터 프레임으로 읽어 들인다. • file : 불러올 자료의 경로와 파일를 설정해 준다. 경로 설정 시 각 디렉터리는 『/』를 이용하여 구분한다. • header = FALSE : 자료의 첫 행에 변수명이 있는 경우 header=TRUE 를 이용하여 자료의 첫 행을 변수명으로 지정하고 다음 행부터 자료를 읽어 오게 한다. default는 FALSE 이다. • sep = " " : 자료 값의 분류 기준을 지정해 준다. 디폴트는 공백 이다. • skip : 자료를 불러올 때 처음 읽을 행의 번호를 지정해 준다. 디폴트는 0이다. • nrows : 자료에서 불러올 행의 개수(자료의 개수)를 지정한다. 디폴트는 -1 이며 음수나 사용하지 않는 경우 모든 자료를 불러온다. • na.strings: "NA"로 저장된 문자열들은 NA로 저장한다. default="NA". • stringsAsFactors=TRUE, default는 TRUE이다.
write.table(x, file=" ", row.names = TRUE)	• 데이터 프레임을 text 파일로 저장한다. • file : 저장할 파일의 경로와 파일이름을 설정한다. • row.names = TRUE이면 행이름을 text파일에 저장한다.

■ 자료의 입력과 출력

▪ CSV 파일 입출력

- CSV 파일을 데이터 프레임으로 읽으려면 read.csv 함수, 데이터 프레임을 CSV로 저장하려면 write.csv 함수를 사용한다.

함수	의미
read.csv("file", header=FALSE, sep=" ", skip=0, nrows = -1, na.strings="NA", stringsAsFactors=FALSE)	• CSV 파일을 데이터 프레임으로 읽어 들인다. • file : 불러올 자료의 경로와 파일을 설정해 준다. 경로 설정 시 각 디렉터리는 『/』를 이용하여 구분한다. • header = FALSE : 자료의 첫 행에 변수명이 있는 경우 header=TRUE 를 이용하여 자료의 첫 행을 변수명으로 지정하고 다음 행부터 자료를 읽어 오게 한다. 디폴트는 FALSE 이다. • sep = " " : 자료 값의 분류 기준을 지정해 준다. 디폴트는 공백 이다. • skip : 자료를 불러올 때 처음 읽을 행의 번호를 지정해 준다. 디폴트는 0이다. • nrows : 자료에서 불러올 행의 개수(자료의 개수)를 지정한다. 디폴트는 -1 이며 음수나 사용하지 않는 경우 모든 자료를 불러온다. • na.strings: "NA"로 저장된 문자열들은 NA로 저장한다. default="NA". • stringsAsFactors=TRUE, default는 TRUE이다.
write.csv(x, file=" ", row.names = TRUE)	• 데이터 프레임을 CSV 파일로 저장한다. • file : 저장할 파일의 경로와 파일이름을 설정한다. • row.names = TRUE이면 행이름을 CSV파일에 저장한다.

■ 자료의 입력과 출력

▪ CSV 파일 입출력

```
> write.table(total, "D:/data/실적2.txt",row.names=TRUE,col.names=TRUE)
> x = read.csv( "D:/data/a.csv"); x
  id    name score
1  1 Mr. Foo    95
2  2 Ms. Bar    97
3  3 Mr. Baz    92
> str(x)
'data.frame':         3 obs. of  3 variables:
 $ id   : int  1 2 3
 $ name : Factor w/ 3 levels "Mr. Baz","Mr. Foo",..: 2 3 1
 $ score: int  95 97 92
> x$name = as.character(x$name)
> str(x)
'data.frame':         3 obs. of  3 variables:
 $ id   : int  1 2 3
 $ name : chr  "Mr. Foo" "Ms. Bar" "Mr. Baz"
 $ score: int  95 97 92
> x = read.csv( "D:/data/a.csv", stringsAsFactors=FALSE)
> str(x)
```

■ 자료의 입력과 출력

▪ CSV 파일 입출력

```
> write.csv(x, "D:/d.csv", row.names=FALSE)

> talent = read.csv( "D:/data/연예인.csv")
> head(talent)
    name gender  ht wt bld
1 소지섭      M 182 70   O
2 조인성      M 186 72   B
3   현빈      M 184 74   B
4 김태희      F 165 45   O
5 송혜교      F 164 45   A
6 김아중      F 170 48   A

> str(talent)
'data.frame':          18 obs. of  5 variables:
 $ name  : Factor w/ 18 levels "강동원","강호동",..: 7 15 18 5 8 4 6 13 1 16 ...
 $ gender: Factor w/ 2 levels "F","M": 2 2 2 1 1 1 2 1 2 2 ...
 $ ht    : int  182 186 184 165 164 170 186 163 186 188 ...
 $ wt    : int  70 72 74 45 45 48 80 52 68 80 ...
 $ bld   : Factor w/ 5 levels "A","AB","B","C",..: 5 3 3 5 1 1 1 1 3 3 ...
```

■ 자료의 입력과 출력

▪ 엑셀 파일 입출력

- 엑셀파일 형태로 저장된 자료를 불러오기 위해서는 {readxl)} 또는 {xlsx }라는 패키지를 설치 해야 함
- 함수 read_excel 함수 또는 read.xlsx 함수를 사용한다.

함수	의미	
read_excel(path, sheet = 1, col_names = TRUE, col_types = NULL, na = "", skip = 0)	path	• Path to the xls/xlsx file
	sheet	• Sheet to read. Either a string (the name of a sheet), or an integer (th e position of the sheet). Defaults to the first sheet.
	col_names	• Either TRUE to use the first row as column names, FALSE to numbe r columns sequentially from X1 to Xn, or a character vector giving a name for each column.
	col_types	• Either NULL to guess from the spreadsheet or a character vector co ntaining "blank", "numeric", "date" or "text".
	na	• Missing value. By default readxl converts blank cells to missing dat a. Set this value if you have used a sentinel value for missing values .
	skip	• Number of rows to skip before reading any data.
read.xlsx(file, sheetIndex, sheetName=NULL, rowIndex=NULL, startRow=NULL, endRow=NULL, colIndex=NULL, as.data.frame=TRUE, header=TRUE, colClasses=NA, keepFormulas=FALSE, encoding="unknown", ...)		

■ 자료의 입력과 출력

▪ 엑셀 파일 입출력

```
> install.packages("readxl")
> library(readxl)
> talent2<-read_excel( "D:/data/연예인.xlsx")
> head(talent2)
    name gender  ht wt bld
1 소지섭      M 182 70   O
2 조인성      M 186 72   B
3   현빈      M 184 74   B
4 김태희      F 165 45   O
5 송혜교      F 164 45   A
6 김아중      F 170 48   A

> install.packages("xlsx")
> library(xlsx)
> a=read.xlsx( "D:/data/a.xlsx", header=TRUE, sheetName="a")
> a
  id     name score
1  1 "Mr Foo"    95
2  2 "Ms Bar"    97
3  3 "Mr Baz"    92
```

■ 객체의 파일 입출력

▪ 객체의 파일 입출력

- Binary 파일로 R 객체를 저장하고 불러들이는 함수에는 save, load가 있다.

함수	의미
save(…, list=character(), file)	• 메모리에 있는 객체를 파일에 저장한다.
load(file)	• 파일로부터 객체를 메모리로 읽어 들인다.

```
> a <- 1:5 ;  b <- 6:10 ;  c <- 11:15
> save(a, b, c, file= “D:/data/abc1.RData")
> save(list=ls(), file= “D:/data/abc.RData")
> rm(list=ls())
> ls()
character(0)
> load( “D:/data/abc1.RData")
```

```
> ls()
[1] "a" "b" "c"
> a
[1] 1 2 3 4 5
> b
[1]  6  7  8  9 10
> c
[1] 11 12 13 14 15
```

■ data frame의 결합

< data frame을 보기 위한 유틸리티 함수>

함수	기능
head(x,n)	• x의 앞부분을 n만큼 잘라낸 데이터이다. Default=6
tail(x,n)	• x의 뒷부분을 n만큼 잘라낸 데이터이다. Default=6
Veiw(x,title)	• 데이터 부어를 호출한다.

```
> d = data.frame(a=1:3, b=4:6, c=7:9)
> d
  a b c
1 1 4 7
2 2 5 8
3 3 6 9

> head(d, 2)
  a b c
1 1 4 7
2 2 5 8
```

```
> tail(d,2)
  a b c
2 2 5 8
3 3 6 9
```

■ data frame의 결합

- cbind, rbind 함수 이용

```
> a1 <- data.frame(x = c(5, 10, 15),
  y = c("a", "b", "c"))
> b1 <- data.frame(z = c(10, 20, 30))
> ab=cbind(a1, b1)
> ab
   x y  z
1  5 a 10
2 10 b 20
3 15 c 30
> str(ab)
'data.frame':          3 obs. of  3 variables:
 $ x: num  5 10 15
 $ y: Factor w/ 3 levels "a","b","c": 1 2 3
 $ z: num  10 20 30
```

```
> c1 <- data.frame(x = c(20, 25, 30),
      y = c("d", "e", "f"))
> ac=rbind(a1, c1)
> ac
   x y
1  5 a
2 10 b
3 15 c
4 20 d
5 25 e
6 30 f
> str(ac)
'data.frame':          6 obs. of  2 variables:
 $ x: num  5 10 15 20 25 30
 $ y: Factor w/ 6 levels "a","b","c","d",..: 1
2 3 4 5 6
```

■ data frame의 결합

- merge 함수는 두 데이터 프레임을 공통된 값을 기준으로 묶는 함수이다.

함수		옵션 내용
merge(x, y, by = intersect(names(x), names(y)), by.x = by, by.y = by, all = FALSE, all.x = all, all.y = all, sort = TRUE, suffixes = c(".x",".y"), incomparables = NULL, ...)	x, y	데이터 프레임 또는 결합 해야 할 객체
	by, by.x, by.y	결합하는 데 사용되는 컬럼의 지정
	all	논리적 표현; all = L은 all.x = L과 all.y = L의 약어적 표현, 여기서 L은 TRUE 또는 FALSE임
	all.x	논리적 표현; all.x=TRUE이면 x의 행이 y의 행과 일치되는 행이 없더라도 결과에 포함된다. 일치되는 않은 y의 컬럼에는 NA로 표시됨. all.x=FALSE이면 x와 y의 일치된 행들만 포함된다.
	all.y	논리적 표현; all.x과 유사함
	sort	논리적 표현. 컬럼에 의해 정렬이 되어야 하는지?
	suffixes	결합되어 사용되지 않은 컬럼들의 이름을 주고자 할 때 길이 2자리의 문자벡터로 사용 (etc에 의해 표현됨).
	incomparables	결합될 수 없는 값들

■ data frame의 결합

- merge 함수

```
> a <- data.frame(Name = c("Kim","Cho",
      "Choi","Lee"),
       math.score = c(90, 95, 98, 100))
> b <- data.frame(Name =
        c("Na","Cho","Lee"),
        eng.score = c(80,90, 95))
> merge(a, b, by = "Name") # inner Join
  Name math.score eng.score
1  Cho         95        90
2  Lee        100        95
> merge(a, b, by = "Name", all = T)
                    # outer Join
  Name math.score eng.score
1  Cho         95        90
2 Choi         98        NA
3  Kim         90        NA
4  Lee        100        95
5   Na         NA        80
```

```
> x=data.frame(name=c("a","b","c"), math=c(1,2,3))
> y=data.frame(name=c("c","b","a"), english=c(4,5,6))
> merge(x,y)
  name math english
1    a    1       6
2    b    2       5
3    c    3       4
> z=data.frame(name=c("a","b","d"), english=c(4,5,6))
> merge(x,z)                # inner Join
  name math English
1    A    1       4
2    b    2       5
> merge(x,z, all=TRUE)    # outer Join
  name math english
1    a    1       4
2    b    2       5
3    c    3      NA
4    d   NA       6
```

■ 데이터의 정렬

- sort 함수: 주어진 벡터를 정렬함
- order 함수: 주어진 인자를 정렬하기 위한 색인을 반환함

```
> v1 <- c(20, 11, 33, 50, 44)
> sort(v1)
[1] 11 20 33 44 50
> sort(v1, decreasing = TRUE)
[1] 50 44 33 20 11

> order(v1)
[1] 2 1 3 5 4
> order(v1, decreasing = TRUE)
[1] 4 5 3 1 2
```

5. 데이터 프레임

■ 데이터의 정렬

```
> sorted <- talent[order(talent$wt), ]
> sorted
     name gender  ht  wt bld
4  김태희      F 165  45   O
5  송혜교      F 164  45   A
6  김아중      F 170  48   A
11 전지현      F 172  48   B
15 공효진      F 172  48   A
8  이효리      F 163  52   A
13 윤은혜      F 167  52   O
12 이수근      M  NA  62   O
18   원빈      M 178  63   O
17 유재석      M 178  65   C
9  강동원      M 186  68   B
16 한석규      M 178  68  AB
1  소지섭      M 182  70   O
2  조인성      M 186  72   B
3    현빈      M 184  74   B
7  성시경      M 186  80   A
10 차승원      M 188  80   B
14 강호동      M 182 115   O
```

```
> sorted_r <- talent[order(talent$wt,decreasing =
TRUE), ]
> sorted_r
     name gender  ht  wt bld
14 강호동      M 182 115   O
7  성시경      M 186  80   A
10 차승원      M 188  80   B
3    현빈      M 184  74   B
2  조인성      M 186  72   B
1  소지섭      M 182  70   O
9  강동원      M 186  68   B
16 한석규      M 178  68  AB
17 유재석      M 178  65   C
18   원빈      M 178  63   O
12 이수근      M  NA  62   O
8  이효리      F 163  52   A
13 윤은혜      F 167  52   O
6  김아중      F 170  48   A
11 전지현      F 172  48   B
15 공효진      F 172  48   A
4  김태희      F 165  45   O
5  송혜교      F 164  45   A
```

5. 데이터 프레임

■ 데이터 프레임의 분리

- split함수를 이용하여 조건에 따라 데이터를 분리하여 리스트로 저장됨

함수		Arguments
split(x, f, drop = FALSE)	x	분할되는 그룹 안에 들어가는 값을 포함하는 벡터 또는 데이터프레임
	f	분할 되는 그룹으로 'factor' 로 정의됨
	drop	f가 factor 또는 list이다면 빠지지 말아야 할 논리를 나타냄

```
> split(iris, iris$Species)
$setosa
   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
7           4.6         3.4          1.4         0.3  setosa
8           5.0         3.4          1.5         0.2  setosa
9           4.4         2.9          1.4         0.2  setosa
10          4.9         3.1          1.5         0.1  setosa
```

```
> x = split(iris$Sepal.Length, iris$Species)
> lapply(x, mean)
$setosa
[1] 5.006

$versicolor
[1] 5.936

$virginica
[1] 6.588
```

■ 데이터 프레임의 분리

- split함수를 이용하여 조건에 따라 데이터를 분리할 수 있음

```
> library(MASS)
> head(Cars93)
> split(Cars93$AirBags, Cars93$Type)
$Compact
 [1] Driver only       None              Driver only       Driver & Passenger Driver only
 [6] None              Driver & Passenger Driver only       Driver only       Driver only
[11] None              None              Driver only       Driver only       None
[16] Driver only
Levels: Driver & Passenger Driver only None

$Large
 [1] Driver only       Driver only       Driver only       Driver only       Driver & Passenger
 [6] Driver only       Driver & Passenger Driver only       Driver & Passenger Driver only
[11] Driver & Passenger
Levels: Driver & Passenger Driver only None

$Midsize
 [1] Driver & Passenger Driver & Passenger Driver only       Driver only       Driver only
 [6] Driver & Passenger None               Driver only       Driver only       None
[11] Driver only       Driver only       Driver & Passenger Driver & Passenger Driver & Passenger
[16] None              Driver only       Driver only       Driver only       None
[21] Driver only       Driver & Passenger
Levels: Driver & Passenger Driver only None
........
```

■ 데이터 프레임의 일부 추출

- subset 함수는 특정 조건을 만족하는 일부부분만을 취하는 용도로 사용됨

함수		Arguments
subset(x,subset, select, drop = FALSE, ...)	x	추출하고자 하는 대상
	subset	유지해야할 요소(element) 또는 행을 나타내는 논리적인 표현. 결측치는 FALSE로 처리됨
	select	데이터프레임으로 부터 선택되어야 할 컬럼을 나타내는 표현
	drop	접근하고자 하는 연산을 통과함

```
> subset(iris, Species =="setosa")
   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
...
> subset(iris, subset=(Species =="setosa" & Sepal.Length > 5.0)) # subset은 생략 가능

> subset(iris, select=c(Sepal.Length, Species))      # 특정 컬럼 포함
> subset(iris, select=-c(Sepal.Length, Species))     # 특정 컬럼 제외
> iris[, !names(iris) %in% c("Sepal.Length", "Species")]
                                          # 바로 위 select문을 사용한 것과 같은 결과임
```

■ 데이터 프레임의 일부 추출

- subset 함수는 특정 조건을 만족하는 일부부분만을 취하는 용도로 사용됨

```
> subset(Cars93, select = c(Model, MPG.city, AirBags),subset = (MPG.city > 30))
     Model MPG.city     AirBags
31 Festiva       31        None
39   Metro       46        None
42   Civic       42 Driver only
73  LeMans       31        None
80   Justy       33        None
83   Swift       39        None
84  Tercel       32 Driver only

> subset(Cars93, select = c(Model, MPG.city, AirBags),subset = (MPG.city > 30 &
         AirBags != "None"))
    Model MPG.city     AirBags
42  Civic       42 Driver only
84 Tercel       32 Driver only
```

■ 데이터 프레임 접근 함수

- with, within, attach, detach 함수
  - 데이터 프레임 또는 리스트 내 필드 이름만으로 접근할 수 있게 해줌
  - with, within은 인자로 넘긴 표현 안에서만 데이터 프레임의 컬럼들을 직접 접근할 수 있게 해준 반면, attach, detach은 이들 함수 호출 후 모든 컬럼들을 직접 접근할 수 있게 한다.

< 데이터 프레임 접근 함수 >

함수	의미
with(data, expr, ...)	• 코드 블록 안에서 필드 이름만으로 데이터를 곧바로 접근할 수 있게 함
within(data, expr, ...)	• with 와 동일한 기능을 제공하지만 데이터에 저장된 값을 손쉽게 변경하는 기능을 제공함
attach	• attach이후 코드에서는 필드 이름만으로 데이터를 곧바로 접근할 수 있게 함
detach	• attach의 반대 역할로 detach 이후 코드에서는 필드 이름만으로 데이터를 곧바로 접근할 수 없게 함

Note: with와 attach 함수의 차이

- with와 within 함수는 명시된 데이터를 환경(environment)으로 하여 표현식을 평가한다.
  반면 attach, detach함수는 이름을 찾는 검색경로를 수정하는 방식으로 실행한다.

5. 데이터 프레임

■ 데이터 프레임 접근 함수

▪ with, within 함수

```
> library(MASS)
> table(Cars93$Type, Cars93$DriveTrain)

             4WD Front Rear
  Compact      1    13    2
  Large        0     7    4
  Midsize      0    17    5
  Small        2    19    0
  Sporty       2     7    5
  Van          5     4    0
> with(Cars93, table(Type, DriveTrain))
          DriveTrain
Type       4WD Front Rear
  Compact    1    13    2
  Large      0     7    4
  Midsize    0    17    5
  Small      2    19    0
  Sporty     2     7    5
  Van        5     4    0
```

```
> (x=data.frame(val=c(1,2,3,4,NA,5,NA)))
  val
1   1
2   2
3   3
4   4
5  NA
6   5
7  NA
> x=within(x,{val = ifelse(is.na(val),
    median(val,na.rm=TRUE), val) })
> x
  val
1   1
2   2
3   3
4   4
5   3
6   5
7   3
```

■ 데이터 프레임 접근 함수

- with, within 함수

```
> data(iris)
> iris[1, 1] = NA
> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1           NA         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
> iris_split = split(iris$Sepal.Length, iris$Species)
> median_per_species = sapply(iris_split, median, na.rm=TRUE)
> median_per_species
    setosa versicolor  virginica
       5.0        5.9        6.5
> iris = within(iris, {
+   Sepal.Length=ifelse(is.na(Sepal.Length), median_per_species[Species], Sepal.Length)})
> head(iris,3)                    # NA가 median 5.0 으로 바뀌었음
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.0         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
```

■ 데이터 프레임 접근 함수

▪ attach, detach 함수

- attach, detach은 이들 함수 호출 후 모든 컬럼들을 직접 접근할 수 있게 한다.
- attach 한 후 이루어진 변수의 수정은 detach 시 원래의 데이터 프레임에는 반영되지 않음.

```
> data(iris)
> Sepal.Width
Error: object 'Sepa.Width' not found
> attach(iris)
> head(Sepal.Width)
[1] 3.5 3.0 3.2 3.1 3.6 3.9
> detach(iris)
> Sepal.Width
Error: object 'Sepa.Width' not found
```

```
> library(MASS)
> attach(Cars93)
> table(Type, DriveTrain)
         DriveTrain
Type       4WD Front Rear
  Compact    1    13    2
  Large      0     7    4
  Midsize    0    17    5
  Small      2    19    0
  Sporty     2     7    5
  Van        5     4    0
> detach(Cars93)
```

■ 데이터의 색인 찾기

- which(x), which.max(x), which.min(x)는 조건을 만족하는 데이터의 색인 자체를 찾는데 이용된다.

함수	의미
which(x)	• 조건이 참인 색인을 반환함. X는 논리값 벡터 또는 배열임
which.max(x)	• 최대값의 위치를 반환함. x는 숫자 벡터임
which.min(x)	• 최소값의 위치를 반환함. X는 숫자 벡터임

```
> subset(iris, Species == "setosa")
   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
...
> iris[iris$Species == "setosa",]
   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
...
```

■ 데이터의 색인 찾기

```
> idx=which(iris$Species == "setosa")
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
[28] 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50

> iris[idx, ]
 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
...
> which.min(iris$Sepal.Length)
[1] 14
> which.max(iris$Sepal.Length)
[1] 132
> idx2 <- which.max(iris$Sepal.Length)
> max(iris$Sepal.Length)
[1] 7.9
> iris$Sepal.Length[idx2]
[1] 7.9
```

6. 데이터 유형 판별 및 변환

■ 데이터 유형 판별 및 검증

▪ 데이터 유형 (data type) 판별 및 검증 함수

함수	의미
class(x)	• 객체 x의 클래스 판별
str(x)	• 객체 x의 내부 구조
is.factor(x)	• 주어진 객체 x가 factor인지를 검증
is.numeric(x)	• 주어진 객체 x가 숫자를 저장한 벡터인지를 검증
is.character(x)	• 주어진 객체 x가 문자를 저장한 벡터인지를 검증
is.matrix(x)	• 주어진 객체 x가 matrix인지를 검증
is.array(x)	• 주어진 객체 x가 array인지를 검증
is.data.frame(x)	• 주어진 객체 x가 data.frame인지를 검증

```
> class(c(1,2))
[1] "numeric"
> class(matrix(c(1,2)))
[1] "matrix “
> class(data.frame(x=1:2, y=3:4))
[1] "data.frame “
> str(c(1,2))
 num [1:2] 1 2
> str(matrix(c(1,2)))
 num [1:2, 1] 1 2
```

```
> str(list(c(1,2)))
List of 1
 $ : num [1:2] 1 2
> is.factor(factor(c("m","f")))
[1] TRUE
> is.numeric(1:5)
[1] TRUE
> is.character(c("a","b"))
[1] TRUE
> is.data.frame(data.frame(x=1:5))
[1] TRUE
```

6. 데이터 유형 판별 및 변환

■ 데이터 유형 변환

▪ 데이터 유형 (data type) 변환 함수

함수	의미
as.factor(x)	주어진 객체 x를 factor로 변환
as.numeric(x)	주어진 객체 x가 숫자를 저장한 벡터로 변환
as.character(x)	주어진 객체 x가 문자를 저장한 벡터로 변환
as.matrix(x)	주어진 객체 x가 matrix로 변환
as.array(x)	주어진 객체 x가 array로 변환
as.data.frame(x)	주어진 객체 x가 dataframe으로 변환

```
> x = c("a", "b", "c")
> as.factor(x)
[1] a b c
Levels: a b c
> (x = matrix(1:9, ncol=3))
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> as.data.frame(x)
  V1 V2 V3
1  1  4  7
2  2  5  8
3  3  6  9
```

```
> data.frame(matrix(c(1:4), ncol=2))
  x1 x2
1  1  3
2  2  4
> data.frame(list(x=c(1,2), y=c(3,4)))
  x y
1 1 3
2 2 4
> as.factor(c("m","f"))
[1] m f
Levels: f m
> factor(c("m","f"), level=c("m","f"))
[1] m f
Levels: m f
```

■ 조건문 if

- if 문은 조건에 따라 코드의 수행 여부를 결정한다.

함수	의미
if (condition) 실행코드	조건을 만족하면 실행코드를 실행한다.

if (condition) {
} else {
}
	조건 cond가 참, 거짓인 경우에 따라 { } 블록을 실행한다.
	필요한 경우 else 블록을 지정할 수 있다.

ifelse(test, yes, no)	주어진 test 값에 따라 yes 또는 no 값을 반환한다.

```
> x=runif(1,min=0,max=1); x        # 균일분포 난수 생성
[1] 0.6784378
> if (x>0.5) y=TRUE ; y            # x가 0.5보다 작거나 같으면 어떤 실행도 하지 않음
[1] TRUE
> rm(y)
> x=runif(1,min=0,max=1); x
[1] 0.2408431
> if (x>0.5) y=TRUE ;
> y
 y Error: object 'y' not found
```

■ 조건문 if

```
> x=runif(1,min=0,max=1)            # 균일분포 난수 생성
> if (x>0.5) y=TRUE else y=FALSE
> x
[1] 0.09822831
> y
[1] FALSE

> x=runif(1,min=0,max=1)
> if (x>0.5) y=x else y=-x
> x
[1] 0.7242042
> y
[1] 0.7242042
> x=runif(1,min=0,max=1)            # 균일분포 난수 생성
> if (x>0.5) y=x else y=-x
> x
[1] 0.489445
> y
[1] -0.489445
```

7. 제어문과 사용자 정의 함수

■ 반복수행문

- 반복수행문은 for, while, repeat 문이 있다.

함수	의미
for (i in data) { i를 사용한 문장 }	data에 들어 있는 값을 i에 할당하여 실행함
while (cond) { 조건이 참일 때 수행할 문장 }	주어진 test 값에 따라 yes 또는 no 값을 반환한다.
repeat { 반복해서 수행할 문장 }	블럭안 의 문장을 반복해서 실행함. 다른 언어의 do-while과 동일.

```
> sum=0                  # 최초값 생성
> for (i in 1:10) {      # 변수 i의 범위: 1~10
    sum=sum+i            # i를 더한 값의 재정의
  }
> i
> Sum                    # 결과 출력
[1] 10
[1] 55
```

```
> total=1                # 최초값 생성
> for (i in 1:10) {      # 변수 i의 범위: 1~10
    total=total*i        # i를 곱한 값의 재정의
  }
> i
[1] 10
> total
[1] 3628800
> factorial(10)
[1] 3628800
```

■ 반복수행문

- 반복수행문은 for, while, repeat 문이 있다.

```
> x=rep(0,7)
> for(i in 1:length(x)){
    x[i]=i
  }
> x
[1] 1 2 3 4 5 6 7

> total=0              # 최초값 생성
> x=c(6,7,8,9)
> for (i in x) {       # 변수 i의 범위:   c(6,7,8,9)
   total=total+i       # i를 더한 값의 재정의
 }
> i
[1] 9
> total                # 결과 출력
[1] 30
```

```
> x=c(0,1,1,0,0,1,0)
> y=rep(0,7)              # 초기 결과값 저장
> for(i in 1:length(x)){
     if(x[i]==0) {y[i]="남"} else {y[i]="여"}
    }
> y
[1] "남" "여" "여" "남" "남" "여" "남"
```

■ 반복수행문

- 반복수행문 while, repeat 문

```
> i=0
> while (i <= 9) {
    i=i+1
    if (i %% 2 !=0) {
    next                  # while 문으로
   }
   print(i)
 }
[1] 2
[1] 4
[1] 6
[1] 8
[1] 10
```

```
> i=0; sum=0
> repeat {
   if (i >= 10) {
     break
   }
   i=i+1
   sum=sum+i
 }
> i
[1] 10
> sum
[1] 55
```

■ 사용자 정의 함수

- 함수이름 <= function(인수) {실행코드}
- 함수를 정의하고 계산된 값을 출력을 해야 하는 경우에는 return 함수를 사용함

```
> square = function(x){
   x^2
 }
> square(5)
[1] 25

> square2 = function(x){
   answer =x^2
   return(answer)
 }
> square2(5)
[1] 25
```

```
> my_median = function(x){
              n=length(x)
              if(n%%2==1) y=x[(n+1)/2]
              else y=(x[n/2]+x[n/2+1])/2
              return(y)
              }
> x1=c(1,2,3,4,5)
> my_median(x1)
[1] 3
> x2=c(1,2,3,4,5,6)
> my_median(x2)
[1] 3.5
```

■ 사용자 정의 함수

- 함수에서 여러 개 값을 출력을 해야 하는 경우에는 return, list 함수를 사용함

```
> testF2 <- function(x) {
    y <- x + 1
    z <- x * 2
    print(paste("x+1=", y))
    return(z)
 }
> testF2(10)
[1] "x+1= 11"
[1] 20
```

```
> testF3 <- function(x) {
y <- x + 1
z <- x * 2
return(list(y = y, z = z))
}
> (answer <- testF3(10))
$y
[1] 11

$z
[1] 20
```

8. 결측치

■ NA의 처리

- NA (결측치) 처리 함수

함수	의미
is.na( )	object가 NA 여부인지를 논리값으로 출력한다.
na.fail( )	object에 NA가 포함되어 있으면 실패한다.
na.omit( )	object에 NA가 포함되어 있으면 이를 제외한다.
na.exclue( )	object에 NA가 포함되어 있으면 이를 제외한다는 점에서 동일함. 그러나 naresid, napredict를 사용하는 함수에서 NA로 제외한 행을 결과에 다시 추가한다는 점이 다르다.
na.pass( )	object에 NA가 포함되어 있더라도 통과시킨다.

```
> x=c(1,2,3,NA,5,6,7,NA,9,10)              # 결측값 2개 포함
> is.na(x)                                 # x의 결측값 여부 출력
 [1] FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE
> sum(is.na(x))                            # TRUE이면 1 FALSE이면 0으로 인식함
[1] 2
> x[x>8]                                   # NA 값은 가장 큰 수임
[1] NA NA  9 10
> sum(x)                                   # NA 를 포함하면 NA임
[1] NA
> var(x)                                   # NA 를 포함하면 NA임
[1] NA
```

8. 결측치 처리

■ NA의 처리

```
> sum(x,na.rm=TRUE)                    # NA 를 제거하고  계산함
[1] 43
> var(x,na.rm=TRUE)                    # NA 를 제거하고 계산함
[1] 10.55357
> x2=na.omit(x)                        # 처음부터 NA 를 제거할 경우
> x2[x2>8]
[1]  9 10
> rank(x)                              # NA 값은 가장 큰 수임
 [1]  1  2  3  9  4  5  6 10  7  8
> r=rank(x,na.last=FALSE); r           # NA 값을 가장 작은 수로 설정
 [1]  3  4  5  1  6  7  8  2  9 10
> x=c(1,NA,3,NA,5)
> mean(x)                              # NA 를 포함하면 NA임
[1] NA
> mean(x,na.rm=TRUE)
[1] 3
> table(x); x                          # NA 는 빈도수 계산에서 제외함
1 3 5
1 1 1
> sort(x,na.last=TRUE)                 # NA 값은 가장 큰 수임
[1] 1 3 5 NA NA
```