7장. R에서의 프로그래밍

1

프로그래밍 언어로서의 R

- 매우 뛰어난 프로그래밍 언어
  - 통계학 이외의 분야에서도 데이터를 기반으로 하는 다양한 프로그래밍 목적으로 활발하게 사용되고 있음
- 프로그래밍 기법중 통계분석 과정에서 자주 사용되는 기법 소개
  - R 언어의 문법: 통계분석 관점에서 많은 부분이 다루어졌음
  - 본격적인 프로그래밍의 관점에서 추가적인 학습 필요

2

2

R 언어의 기본 용어

- 표현식(Expression)
  - R 프로그램은 일련의 표현식으로 이루어짐.
  - 예: 데이터 할당 문장, 조건을 비교하는 문장, 산술식 등등
  - 새로운 줄 또는 세미콜론으로 분리
- 객체(Object)
  - R 프로그램의 기본 요소
  - R에서 다루는 모든 것이 객체; 데이터 객체, 함수, 표현식
- 함수(Function)
  - 인수(변수)라는 객체를 입력시키면 결과물 객체를 생성시키는 또 다른 R 객체
  - R에서 수행되는 거의 모든 작업이 함수로 수행됨

3

3

7.1 사용자 정의 함수

- R의 큰 장점 중 하나: 사용자가 직접 함수를 정의하여 사용할 수 있음
  - 전체적으로 프로그램이 상당히 간결해진다.
  - 분석 절차가 훨씬 더 효율적인 된다.
- 패키지
  - 사용자정의 함수의 묶음

4

4

7.1.1 함수의 정의

```
> my_func <- function(arg1, agr2, …) {
                    표현식
}
```

my_func : 함수 이름
arg1, arg2, … : 함수의 변수 이름

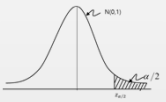
```
> f <- function(x,y) x + y
> f <- function(x,y){
    x+y
  }

> f(5,10)
[1] 15
```

표현식이 한 줄인 경우:
중괄호 필요 없음

7.1.2 변수

- 함수에 입력되는 변수 종류
  1) 연산 대상이 되는 데이터(첫 번째 변수)
  2) 연산과 관련된 세부 옵션(그 이후 변수)
- 디폴트값이 지정된 변수 : 함수실행에서 생략
- 예제: 모평균의 신뢰구간 계산하는 함수 정의
  1) 첫 번째 변수: 표본 자료
  2) 두 번째 변수: 신뢰수준



$(1-\alpha)\cdot 100\%$ 신뢰구간
$(\bar{X} - z_{\alpha/2}\cdot S/\sqrt{n}\ , \bar{X} + z_{\alpha/2}\cdot S/\sqrt{n})$

```
> CI_mean <- function(x, conf=0.95){
    m <- mean(x)
    se <- sd(x)/sqrt(length(x))
    alpha <- 1-conf
    c(m-qnorm(1-alpha/2)*se, m+qnorm(1-alpha/2)*se)
  }
```

- 함수 qnorm( ): 정규분포 분위수 계산

- 표준정규분포에 100개 난수 발생
- 모평균에 대한 95% 신뢰구간 추정

```
> set.seed(1234579) # seed번호 지정, 고정된 난수 발생
> x <- rnorm(n=100)
> CI_mean(x)        # conf에 디폴트값 0.95 지정되어 있슴
[1] -0.2778891  0.1660690
```

- 모평균에 대한 90% 신뢰구간 추정

```
> CI_mean(x, conf=0.9) # conf에 값 입력하면 디폴트값 0.95 무시됨
[1] -0.2422007  0.1303807
```

7

7

- 생략 부호(...) 변수
  - 일반적으로 함수의 마지막 변수로 지정
  - 기존의 함수를 이용하여 함수를 정의하는 경우 매우 유용함

- 예: 함수 `CI_mean( )`
  - 자료에 NA 포함: 함수 `mean()`, `sd()`의 결과도 NA
  - 변수 `na.rm`을 함수 `CI_mean()`에서 사용할 방법은?

```
> y <- c(x,NA)
> CI_mean(y)
[1] NA NA
> CI_mean(y, na.rm=TRUE)
Error in CI_mean(y, na.rm = TRUE) : unused argument (na.rm =
TRUE)
```

```
<함수 CI_mean>

> CI_mean <- function(x, conf=0.95){
    m <- mean(x)
    se <- sd(x)/sqrt(length(x))
    alpha <- 1-conf
    c(m-qnorm(1-alpha/2)*se, m+qnorm(1-alpha/2)*se)
  }
```

8

8

- 생략 부호를 포함시킨 함수 CI_mean()

```
> CI_mean_dot <- function(x,conf=0.95,...){
    m <- mean(x,...)
    se <- sd(x,...)/sqrt(sum(!is.na(x)))
    alpha <- 1-conf
    c(m-qnorm(1-alpha/2)*se, m+qnorm(1-alpha/2)*se)
  }
```

```
> CI_mean_dot(y, na.rm=TRUE)
[1] -0.2778891  0.1660690
```

9

9

- 생략 부호대신 구체적인 옵션을 포함시킨 함수 CI_mean()

```
> CI_mean_dot1 <- function(x,conf=0.95,aa=F){
   m <- mean(x,na.rm=aa)
   se <- sd(x,na.rm=aa)/sqrt(sum(!is.na(x)))
   alpha <- 1-conf
   c(m-qnorm(1-alpha/2)*se, m+qnorm(1-alpha/2)*se)
 }

> x=c(1,3,5,7,9,NA)
> CI_mean_dot1(x,aa=T)
[1] 2.228192 7.771808
```

```
> CI_mean_dot2 <- function(x,conf=0.95,na.rm=F){
   m <- mean(x,na.rm=na.rm)
   se <- sd(x,na.rm=na.rm)/sqrt(sum(!is.na(x)))
   alpha <- 1-conf
   c(m-qnorm(1-alpha/2)*se, m+qnorm(1-alpha/2)*se)
 }

> x=c(1,3,5,7,9,NA)
> CI_mean_dot2(x,na.rm=T)
[1] 2.228192 7.771808
```

10

10

- 생략 부호 예제
  - 두 벡터를 표준화 시킨 후 산점도 작성 함수 정의

```
> my_plot <- function(x, y, ...){
    z_x <- (x-mean(x))/sd(x)
    z_y <- (y-mean(y))/sd(y)
    ggplot(data.frame(x=z_x, y=z_y)) +
      geom_point(aes(x, y), ...)
  }
```

  - 함수 my_plot( )의 변수: x, y, 생략 부호 변수
  - 함수 geom_point( )에 생략 부호 변수 지정

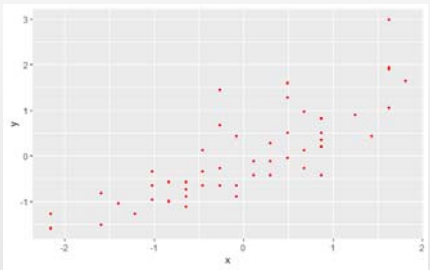
→ geom_point( )의 변수를 my_plot( )의 변수처럼 사용 가능

11

11

- 데이터 프레임 cars의 두 변수 speed와 dist를 표준화하고 산점도 작성

```
> library(ggplot2)
> with(cars, my_plot(x=speed, y=dist, shape=20,
                     color="red", size=2))
```



<함수 my_plot>

```
> my_plot <- function(x, y, ...){
    z_x <- (x-mean(x))/sd(x)
    z_y <- (y-mean(y))/sd(y)
    ggplot(data.frame(x=z_x, y=z_y)) +
      geom_point(aes(x, y), ...)
  }
```

12

12

7.1.3 변수의 지정

- 함수의 변수를 지정하는 방법
- 예

```
> my_power <- function(first, second) {first^second}
```

1) 변수의 전체 이름

```
> my_power(second=5, first=2)
[1] 32
```

2) 변수의 부분 이름

```
> my_power(s=5, f=2)
[1] 32
```

3) 변수의 순서

```
> my_power(2, 5)
[1] 32
```

13

13

7.1.4 결과의 출력

1. 함수 return( )에 의한 출력
2. 함수 return( )이 없는 경우: 마지막 표현식의 실행 결과

- 예: 함수 my_desc( ) 입력된 벡터의 평균과 표준편차를 계산하여 리스트 형태로 출력

- 함수 return( )에 의한 실행 결과 출력

```
> my_desc <- function(x,...){
    m.x <- mean(x,...); sd.x <- sd(x,...)
    res <- list(mean=m.x, sd=sd.x)
    return(res)
  }
```

14

14

- 데이터 프레임 cars의 변수 dist의 평균, 표준편차 출력

```
> with(cars, my_desc(x=dist))
$mean
[1] 42.98

$sd
[1] 25.76938
```

- 데이터 프레임 airquality의 변수 Ozone의 평균, 표준편차 출력

```
> with(airquality, my_desc(x=Ozone, na.rm=TRUE))
$mean
[1] 42.12931

$sd
[1] 32.98788
```

```
<함수 my_desc>

> my_desc <- function(x,...){
    m.x <- mean(x,...); sd.x <- sd(x,...)
    res <- list(mean=m.x, sd=sd.x)
    return(res)
  }
```

15

15

- 함수 return( )이 없는 함수의 결과 출력

```
> my_desc_1 <- function(x,...){
    m.x <- mean(x,...); sd.x <- sd(x,...)
    list(mean=m.x, sd=sd.x)
  }
> with(cars,my_desc_1(x=dist))
$mean
[1] 42.98

$sd
[1] 25.76938
```

- 마지막 표현식이 할당문이면 아무런 결과도 출력되지 않음

```
> my_desc <- function(x,...){
    m.x <- mean(x,...); sd.x <- sd(x,...)
    res <- list(mean=m.x, sd=sd.x)
  }
> with(cars, my_desc(x=dist))
>
```

16

16

7.2 조건 연산

- 주어진 조건의 만족 여부에 따라 실행되는 표현식을 다르게 하는 연산
  - 함수 if( )
  - 함수 ifelse( )
  - 함수 switch( )

17

17

7.2.1 함수 if( )에 의한 조건 연산

```
if (조건) {
            표현식
 }
```

- 조건이 만족되면 표현식 실행

```
if (조건) {
            표현식 1
} else {
            표현식 2
}
```

- 조건이 만족되면 표현식 1 실행
- 조건이 만족되지 않으면 표현식 2 실행

18

18

```
if (조건 1) {
        표현식 1
} else if (조건 2) {
        표현식 2
} else {
        표현식 3
}
```

- 조건 1이 만족되면 표현식 1 실행
- 조건 2가 만족되면 표현식 2 실행
- 그 외의 경우에는 표현식 3 실행

조심해야 할 사항

```
if (조건) {
        표현식 1 }

else {
        표현식 2
}
```

if( ) 문이 종료된 것으로 간주

else로 시작하는 명령문이 없어 오류 발생

19

19

- 예: 근의 공식을 이용하여 이차방정식의 근을 구하는 프로그램 작성

$ax^2 + bx + c = 0$ 의 실근 구하기

판별식: $D = b^2 - 4ac$

$if\ D > 0, \quad roots = \frac{-b \pm \sqrt{D}}{2a}$

$if\ D = 0, \quad root = \frac{-b}{2a}$

$if\ D < 0, \quad No\ real\ roots$

```
if ( D > 0 ) {
        roots =

} else if ( D == 0) {
        roots =

} else {
        roots =
}
```

20

20

- 이차방정식 근을 구하는 사용자 정의 함수 작성

```
> find_roots <- function(a, b, c){
    if(a==0){
      roots <- c("Not quadratic equation")
    } else{
      D <- b^2-4*a*c

      if(D > 0){
        roots <- c((-b-sqrt(D))/(2*a),(-b+sqrt(D))/(2*a))
      } else if(D == 0){
        roots <- -b/(2*a)
      } else{
        roots=c("No real root")
      }
    }
    return(roots)
  }
```

- 방정식$x^2 + 4x + 3 = 0$의 근

```
> find_roots(a=1,b=4,c=3)
[1] -3 -1
```

21

21

- 함수 if( ) 사용 시 유의할 점
  - if( )의 조건에는 하나의 논리값만 사용됨
  - 예: 두 벡터 x와 y의 구성요소를 비교하여 값이 큰 요소를 출력

```
> x <- c(10,3,6,9)
> y <- c(1,5,4,12)
> if(x > y) x else y
[1] 10  3  6  9
Warning message:
In if (x > y) x else y :
  the condition has length > 1 and only the first element will be
used

 # x>y 에는 T, F, T, F 의 네개의 논리값이 있음 #
```

22

22

7.2.2 함수 ifelse( )에 의한 조건 연산

- 사용하고자 하는 조건이 하나의 논리값이 아닌 논리벡터의 경우

```
ifelse(조건, 표현식 1, 표현식 2)
```

조건이 만족되면 표현식 1
만족되지 않으면 표현식 2를 실행

예: 두 벡터 x와 y의 구성요소를 비교하여 값이 큰 요소를 출력

```
> x <- c(10,3,6,9)
> y <- c(1,5,4,12)
> ifelse(x>y,x,y)
[1] 10  5  6 12
```


- 예: 주어진 점수가 50미만이면 'Fail', 50 이상이면 'Pass'를 점수와 함께 출력

```
> score <- c(80,75,40,98)
```

```
> grade <- ifelse(score>=50, "Pass", "Fail")
> data.frame(score, grade)
  score grade
1    80  Pass
2    75  Pass
3    40  Fail
4    98  Pass
```

7.2.3 함수 switch( )에 의한 조건 연산

`switch(표현식, 선택 항목 리스트)`

- 표현식이 갖는 값에 따라 선택 항목 중 선택
- 선택할 항목 리스트: 콤마로 구분된 리스트
- 표현식값이 숫자인 경우: 선택할 항목의 위치 지정
- 표현식값이 문자인 경우: 선택할 항목 중 항목이름이 동일한 항목 선택

예: Park, Lee, Kim 중 한 사람 임의로 선택

```
> (x <- sample(1:3,1))
[1] 2
> switch(x, "Park", "Lee", "Kim")
[1] "Lee"
```

```
> switch("aa", aa="bb", bb="aa")
[1] "bb"
```

25

25

- 예: 주어진 자료의 특성을 보고, 자료의 대푯값으로 평균과 중앙값 중 선택하는 함수 작성

```
> my_center <- function(x, type){
    switch(type, mean=mean(x), med=median(x))
  }
```

- 자료:

```
> x <- c(1,2,3,4,50)
```

- 함수 my_center( )로 자료의 평균과 중앙값 각각 계산

```
> my_center(x, type="med")
[1] 3
> my_center(x, type="mean")
[1] 12
```

26

26

복합 조건

&, |	: vector 의 element 별로 조건 체크
&&, ||	: vector 의 첫번째 값만 체크

```
> x=c(1,2,5,7,8)
> y=c(3,7,5,2,1)
> w=2;u=7
> (x>2) & (y>3)
[1] FALSE FALSE  TRUE FALSE FALSE
> (x>2) && (y>3)
[1] FALSE
> (w>2) & (u>3)
[1] FALSE
> (w>2) && (u>3)
[1] FALSE
```

```
> (x>2) | (y>3)
[1] FALSE  TRUE  TRUE  TRUE
TRUE
> (x>2) || (y>3)
[1] FALSE
> (w>2) | (u>3)
[1] TRUE
> (w>2) || (u>3)
[1] TRUE
```

27

27

IF 문 예제(종합)

```
> x=runif(1)-0.5;x
[1] 0.002352862

> if (x<0) print(abs(x))                                    # if

> if (x<0) print(abs(x)) else print(x)                      # if      else
[1] 0.002352862

> ifelse(x<0,abs(x),x)                                      # ifelse
[1] 0.002352862

> if(x<0) {print(x);print("x is negative")} else {print(x);print("x is positive")} #복합실행
[1] 0.002352862
[1] "x is positive"

> if(x>=-0.5 && x<=0.5) print(x) else print("wrong number")        #복합조건
[1] 0.002352862
```

28

28

7.3 루프 연산

- 프로그램의 특정 부분을 일정 횟수 반복하는 작업
  - 함수 for( )
  - 함수 while( )

29

29

7.3.1 for 루프

- 기본적인 사용법: 함수 for( )

```
for ( var  in  seq ) {
      표현식
}
```

```
> for(i  in 1:5) { print(i) }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

var : 변수 이름
seq: 벡터

변수 var는 벡터 seq의 값을 차례로 취하면서 표현식을 실행

30

30

- 예: 정규분포에서 10개의 임의표본을 추출하여 평균을 계산하는 과정이 다섯 번 반복
  - 거의 동일한 명령문 다섯 번 반복하는 것은 바람직하지 않음
  - 비슷한 연산의 반복은 for 루프 사용

```
> res <- vector("double", 5)

> for(i in seq_along(res)) {
    res[i] <- mean(rnorm(10))
  }
```

- 첫 번째 요소: 루프 연산으로 생성될 객체를 위한 빈 공간 생성
- 루프 연산 중 결과값의 할당에 사용되는 인덱싱 대비
- vector("유형", 길이)
  `vector("double",5) = c(0,0,0,0,0)`

- 두 번째 요소: 반복 횟수 및 인덱스 변수 지정
- `for(var in seq)`: 인덱스 변수(var)가 seq의 값을 차례로 취하면서 루프 수행
- seq_along (x): seq(length=length(x))

- 세번 째 요소: 중괄호 안에서 반복 수행되는 명령문
- 수행 결과를 미리 확보한 벡터에 인덱싱 기법으로 할당

```
> round(res,3)
[1]  0.302  0.466 -0.906 -0.774 -0.268
```

31

31

- 예: 함수 for( )로 Factorial 계산하고 출력

```
> fac.x <- 1
> for(i in 1:5){
        fac.x <- fac.x*i
        cat(i, "!=", fac.x, "\n", sep="")
  }
1!=1
2!=2
3!=6
4!=24
5!=120
>
```

함수 cat() : next slide

32

32

Ref) print( ) & cat( )

- 함수 print( ): 한 번에 한 객체만 출력 가능

```
> pi                                    # print(pi)와 동일
[1] 3.141593
> print(pi)
[1] 3.141593

> print( "원주율은", pi ,"이다" )
다음에 오류print.default("원주율은", pi, "이다") : 'quote' 인수가 잘못되었습니다

> cat( "원주율은", pi ,"이다", " ₩n " , sep=" " )  # ₩n : 줄바꿈, sep: 요소구분 문자 지정
원주율은 3.141593 이다
```

- 함수 cat ( ): 벡터로만 출력가능

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

> print( y )
     [,1] [,2]
[1,]    1    3
[2,]    2    4

> cat( y , " ₩n " )
1 2 3 4
```

33

33

7.3.2 while( ) 루프

- for 루프: 반복 횟수 미리 고정
- 특정 조건이 만족될 때까지 반복해야 하는 경우: for 루프 사용 불가능

- while 루프: 조건이 만족되는 동안 표현식 실행

```
while ( 조건) {
                표현식
 }
```

```
> i=1
> while(i <= 5) {
+     print(i)
+     i=i+1
+ }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

34

34

- 예: 함수 while( )로 Factorial 계산하고 출력

```
> fac.x <- 1
> i <- 1
> while(i <= 5){
        fac.x <- fac.x*i
        cat(i, "!=", fac.x, "\n", sep="")
        i <- i+1
  }
1!=1
2!=2
3!=6
4!=24
5!=120
>
```

35

35

연습)
1 에서 5 까지 각각 한 개 숫자씩 더한 결과를 보여주는 프로그램

```
> sum.x=0
> i=1
> while(i <= 5){
+    sum.x = sum.x + i
+    cat("sum to ",i, " =", sum.x, "₩n",
sep="")
+    i = i + 1
+ }
sum to 1 =1
sum to 2 =3
sum to 3 =6
sum to 4 =10
sum to 5 =15
```

연습)
함수 while() 을 이용하여 5개의 자료 1,1,1,8,7,6 의 평균을 계산하는 프로그램

```
> sum.x=0
> x=c(1,1,1,8,7,6)
> k=length(x)
> i=1
> while(i <= k){
+            sum.x = sum.x + x[i]
+            i = i + 1
+ }
> sum.x/k
[1] 4
```

36

36

IF문 + 루프문 연습

연습)

1 에서 10까지 정수를 더할때 홀수는 원래 값을 더하고 짝수는 원래값의 2배를 더한 결과를 출력하라.

```
> sum.x=0
> for (i in 1:10) {
+ if (i %in% c(2,4,6,8,10)) sum.x=sum.x+2*i else sum.x=sum.x+i}
> sum.x
[1] 85
```

연습 문제 7.1 : 피보나치 수열 , 12항 까지

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

37

37

7.4 함수형(functional) 프로그래밍

- 동일한 작업이 반복되는 상황
- 루프 연산: 좋은 대안이나 프로그램 의미 파악이 어렵다는 단점
- 루프 연산의 대안: 함수형(functional) 프로그래밍

- Functional: 함수를 입력 변수로 받는 함수

- 예: `> my_desc <- function(x, fun) {fun(x)}`

```
> x <- rnorm(20)

> my_desc(x, mean)
[1] -0.1992002

> my_desc(x, median)
[1] 0.07421808
```

rnorm( ): 정규분포에서 난수 발생
`rnorm(n, mean=0, sd=1)`

38

38

- 중요하게 사용되는 functional의 예: lapply( ), sapply( )

```
lapply(x, FUN, ...)
```

  - x: 벡터 또는 리스트
  - 입력된 x의 각 요소에 FUN에 지정한 함수를 적용
  - 결과는 리스트 (l="list")

Sapply : 동일한 사용법. 결과가 벡터 또는 행렬 (s="subgroup")

39

39

- 예: 리스트 x의 각 요소의 평균값 계산

```
> x <- list(a1=1:5, a2=rnorm(5), a3=c(TRUE, FALSE, TRUE, TRUE))
```

```
> lapply(x, mean)
$a1
[1] 3

$a2
[1] -0.0727762

$a3
[1] 0.75
```

```
> sapply(x, mean)  # 결과 vector
        a1         a2         a3
 3.0000000 -0.0727762  0.7500000
```

```
> unlist(lapply(x, mean))
        a1         a2         a3
 3.0000000 -0.0727762  0.7500000
```

40

40

- 루프 연산과 함수형 프로그래밍의 비교

  - 예) 평균이 -2, -1, 0, 1, 2이고 표준편차가 0.5인 정규분포에서 10개의 임의표본을 각각 추출하여 평균 계산

1) 루프 연산

```
> set.seed(1234)
> m <- -2:2
> res <- vector("double", length(m))
> for(i in seq_along(res)){
    res[i] <- mean(rnorm(n=10, mean=m[i], sd=0.5))
  }
> res
[1] -2.1915787 -1.0590854 -0.1939734  0.6169035  1.6951015
```

41

41

2) 함수형 프로그래밍

- 다섯 정규분포에서 10개씩 임의표본 추출하여 리스트에 할당
- 리스트 각 요소에 평균 적용

```
> set.seed(1234)
> m <- -2:2
> x <- lapply(m, rnorm, n=10, sd=0.5)
> sapply(x, mean)
[1] -2.1915787 -1.0590854 -0.1939734  0.6169035  1.6951015
```

rnorm( ): 정규분포에서 난수 발생

```
rnorm(n, mean=0, sd=1)
```

```
> rnorm(n=5,mean=-2,sd=0.5)
> rnorm(-2,n=5,sd=2)
> rnorm(n=5,-2,sd=2)
> rnorm(n=5,sd=2,-2)
```

42

42

7.5 함수형 프로그래밍으로 행렬과 데이터 프레임 다루기

- 행렬과 데이터 프레임을 대상으로 반복된 작업이 필요한 경우가 있음
  - 예: 행렬의 각 열 또는 각 행의 평균값 계산
- 루프 연산보다 함수형 프로그래밍을 적용하는 것이 더 효율적

7.5.1 행렬에 함수형 프로그래밍 적용하기

- 행렬에 적용할 수 있는 functional: 함수 apply( )

```
apply(X, MARGIN, FUN, ...)
```

X: 행렬 또는 배열
MARGIN: FUN에 지정한 함수가 적용되는 방향
MARGIN=1 행 방향
MARGIN=2 열 방향

- 예: 세 사람에 대한 반복 측정값

```
> A
      trial1 trial2 trial3 trial4
Park     0.8    1.1    0.0    0.6
Lee      1.3    1.3    1.2    1.4
Kim      1.0    1.3    0.2    0.6
```

```
> A <- matrix(c(0.8,1.3,1.0, 1.1, 1.3, 1.3, 0, 1.2, 0.2,
                0.6, 1.4, 0.6), nrow=3)
> rownames(A) <- c("Park", "Lee", "Kim")
> colnames(A) <- paste0("trial", 1:4)
```

- 각 사람마다 반복 측정된 자료의 평균값 / 최소, 최대 계산
- 각 반복마다 측정된 자료의 평균값 계산

```
> apply(A, 1, mean)
 Park   Lee   Kim
0.625 1.300 0.775
```

```
> apply(A, 1, range)
     Park Lee Kim
[1,]  0.0 1.2 0.2
[2,]  1.1 1.4 1.3
```

```
> apply(A, 2, mean)
   trial1    trial2    trial3    trial4
1.0333333 1.2333333 0.4666667 0.8666667
```

45

45

7.5.2 데이터 프레임에 함수형 프로그래밍 적용하기

- 그룹별 요약 통계량 계산
  - 데이터 프레임에 요인(factor)이 있는 경우, 요인의 수준에 따라 구성되는 그룹별 다른 변수의 분포 비교는 중요한 분석
  - 사용 가능한 방법
    1) `tapply(X, INDEX, FUN, simplify=TRUE)`
       X: 벡터
       INDEX: 요인
       FUN: 요약 통계량을 계산하는 함수
       simplify: 출력 형태 지정. 벡터(TRUE), 리스트(FALSE)

    2) 함수 split( )과 lapply( )를 연결해서 사용

    3) dplyr의 group_by( )와 summarise( )을 이용

46

46

- 예: MASS의 Cars93에 있는 요인 Origin의 수준별 MPG.city의 평균값 비교 (세가지 방법)

```
> data(Cars93, package="MASS")
```

- 함수 tapply( )에 의한 방법

```
> with(Cars93, tapply(MPG.city, Origin, mean))
     USA  non-USA
20.95833 23.86667

> with(Cars93, tapply(MPG.city, Origin, mean,
simplify=FALSE))
$USA
[1] 20.95833

$`non-USA`
[1] 23.86667
```

47

47

- 함수 split( )과 lapply( )에 의한 방법

```
> x_g <- with(Cars93, split(MPG.city, Origin))
> str(x_g)
List of 2
 $ USA    : int [1:48] 22 19 16 19 16 16 25 25 19 21 ...
 $ non-USA: int [1:45] 25 18 20 19 22 46 30 24 42 24 ...

> lapply(x_g, mean)
$USA
[1] 20.95833

$`non-USA`
[1] 23.86667
```

```
split(A, B))
: 벡터 또는 데이터 프레임 A 를 요인 B의 값 별로 분리, 결과 list
```

48

48

- dplyr의 group_by( )와 summarise( )에 의한 방법

```
> library(dplyr)
> Cars93 %>%
    group_by(Origin) %>%
    summarise(m=mean(MPG.city), n=n())
# A tibble: 2 x 3
  Origin      m     n
  <fct>   <dbl> <int>
1 USA      21.0    48
2 non-USA  23.9    45
```

49

49

- 데이터 프레임의 모든 변수에 함수 적용
  - 데이터 프레임를 typeof( )로 확인할 때 데이터 유형은 리스트
  - 함수 lapply( ) 또는 sapply( )로 데이터 프레임의 각 변수에 동일한 함수 적용 가능

50

50

- 예: 데이터 프레임 airquality에서 변수 Month와 Day를 제외한 나머지 변수의 평균값 계산

  - 함수 sapply( )에 의한 계산

```
> library(dplyr)
> airs <- airquality %>%
    select(-Month, -Day)
> sapply(airs, mean, na.rm=TRUE)
     Ozone    Solar.R       Wind       Temp
 42.129310 185.931507   9.957516  77.882353
```

  - 함수 summarise ( ), across( )에 의한 계산

```
> airs %>%
    summarise(across(.fns = mean, na.rm=TRUE)
     Ozone  Solar.R     Wind     Temp
1 42.12931 185.9315 9.957516 77.88235
```

51

51

7.6 purrr에 의한 프로그래밍

- purrr: tidyverse에 속한 패키지

  - 대표적인 함수: map( )

```
map(.x, .f, ...)
```

.x 벡터 또는 리스트
.f .x의 각 요소에 적용되는 함수
결과는 리스트

  - Base R 함수 lapply( )와 동일한 기능을 갖고 있으나 더 개선된 기능이 있는 함수

```
> library(tidyverse)
```

52

52

- 예: 리스트 x 각 요소의 평균값 계산

```
> x <- list(a1=1:5, a2=rnorm(5), a3=c(TRUE, FALSE, TRUE, TRUE))
```

```
> map(x, mean)
$a1
[1] 3

$a2
[1] 0.4293029

$a3
[1] 0.75
```

- 결과를 벡터로 출력하고자 하는 경우 출력되는 벡터의 유형에 따라

  논리형: map_lgl( ),
  정수형: map_int( ),
  숫자형: map_dbl( ),
  문자형: map_chr( ) 중 선택

- 리스트 x 각 요소의 평균값을 벡터로 출력

```
> map_dbl(x, mean)
       a1        a2        a3
3.0000000 0.4293029 0.7500000
```

53

53

- map( )에서 사용자 정의 함수 사용하기
  - 예: airquality의 변수 Ozone, Solar.R, Wind, Temp의 평균값 계산을 사용자 정의 함수로 시행

```
> airs <- airquality %>%
    select(-c(Month, Day))
```

```
> airs %>%
    map_dbl(function(x) sum(x, na.rm=TRUE)/sum(!is.na(x)))
     Ozone    Solar.R       Wind       Temp
 42.129310 185.931507   9.957516  77.882353
```

map( )에서 가능한 기능
- 사용자 정의 함수의 시작인 function( )을 물결표(~)로 대치
- 한 변수만 사용되는 함수인 경우, 변수 대신 점(.) 또는 .x 사용

```
> airs %>%
    map_dbl(~ sum(.x, na.rm=TRUE)/sum(!is.na(.x)))
     Ozone    Solar.R       Wind       Temp
 42.129310 185.931507   9.957516  77.882353
```

54

54

- 함수 map( )에서 .f에 함수 대신 숫자 또는 문자 입력
  - 리스트의 특정 요소를 선택하는 인덱싱이 실행
    - 예: 숫자 입력

```
> df1 <- list(x1=1:3, x2=2:4, x3=3:6)
> map_int(df1, 2)
x1 x2 x3
 2  3  4
```

- 리스트 df1 각 요소의 두 번째 자료 선택
- map_int: 결과 integer

55

55

- 예: N(-1,1), N(1,1)에서 발생시킨 5개 난수를 summary( )에 입력하여 얻은 요약 통계량 중 "Mean" 이름이 붙은 자료 선택

```
> df2 <- list(x1=rnorm(n=5, mean=-1), x2=rnorm(n=5, mean=1)) %>%
    map(summary)

> df2
$x1
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-2.0796 -1.6069 -0.1894 -0.5972  0.1608  0.7293

$x2
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-1.0074  0.6359  1.6669  1.1643  1.7736  2.7525
```

```
> df2 %>%
    map_dbl("Mean")
       x1        x2
-0.597173  1.164287
```

56

56

- 함수 map2( )

  - map( ): 하나의 리스트(혹은 벡터)의 각 요소에 동일 함수 적용
  - map2( ): 두 개의 리스트(혹은 벡터)를 입력 변수로 특정 함수를 반복 적용

  - 사용법: `map2(.x, .y, .f, ...)`
    `.x`, `.y`: 두 개의 입력 리스트(벡터). 길이가 같아야 함. 길이가 1인 벡터의 경우만 순환법칙 적용

57

57

- 예: $N(-5, 2^2)$과 $N(5,1)$에서 각각 5개 난수 발생

```
> mu <- c(x1 = -5, x2 = 5)
> sigma <- c(x1 = 2, x2 = 1)

> map2(.x=mu, .y=sigma, rnorm, n=5)
$x1
[1] -4.867399 -4.994178 -4.581368 -2.355002 -9.357798

$x2
[1] 4.943467 4.510158 5.130411 3.759665 5.204177
```

- 결과를 데이터 프레임(tibble) 으로 출력: map2_dfc( )

```
> map2_dfc(.x=mu, .y=sigma, rnorm, n=3)
# A tibble: 3 x 2
      x1    x2
   <dbl> <dbl>
1  -3.27  4.14
2  -3.04  3.89
3 -10.6   5.22
```

- x1은 첫번째 결과, x2는 두번째 결과
- 변수명은 .x의 요소 이름에 의해 결정

58

58