3-3장. 데이터 다루기

박영식(youngsik.park@bsl-lausanne.ch)

예: 패키지 ggplot2의 데이터 프레임 mpg에 있는 trans(변속기)

```
> x <- ggplot2::mpg$trans
> table(x)
x
auto(av)  auto(l3)  auto(l4)  auto(l5)  auto(l6)  auto(s4)
       5         2        83        39         6         3
auto(s5)  auto(s6) manual(m5) manual(m6)
       3        16        58        19
```

- auto(av)부터 auto(s6)까지를 auto로 통합하고 manual(m5)와 manual(m6) 를 manual로 구분하여 데이터를 봐보자!

```
> y <- substr(x, 1, nchar(x)-4)
> table(y)
y
  auto Manual
   157     77
```

함수 strsplit( ):문자열 데이터 관리!

◎ 옵션 split에 기준을 사용자가 지정함으로써 분리. 결과는 리스트로 출력!

- 3-2장에서 살펴본 예제를 활용해보자.

```
>  x  <- c ( "New York, NY", "Ann Arbor, MI", "Chicago, IL" )
```

```
> (y <- strsplit(x, split=“ , "))
[[1]]
[1] "New York" " NY"

[[2]]
[1] "Ann Arbor" " MI"

[[3]]
[1] "Chicago" " IL"

> unlist(y)
[1] "New York"  " NY"    "Ann Arbor"  " MI"     "Chicago"
[6] " IL
```

문자열 데이터 관리: 인덱싱!

- 세 도시가 속한 주 이름만을 선택하여 출력하기

리스트의 인덱싱 방식

```
> c ( z1[[1]][2], z1[[2]][2], z1[[3]][2] )
[1] " NY" "MI"  " IL"
```

함수 unlist( )로 만들어진 벡터의 인덱싱 방식

```
> unlist(z1) [ c(2,4,6) ]
[1] " NY" "MI"  " IL"
```

문자열 데이터 관리: 분리!

- 문자열을 구성된 문자를 알파벳으로 분리

```
> unlist(strsplit("PARK", split=" "))
[1] "P" "A" "R" "K""
```

- 점(.)을 기준으로 문자를 분리하고 싶을 경우
  - 옵션 split=“.” : 원하는 결과를 얻을 수 없다.

```
> unlist(strsplit("P.A.R.K", split="."))
[1] "" "" "" "" "" "" ""
```

  - 옵션 split=“[.]” 또는 split=“\\.”으로 표현해보자!

```
> unlist(strsplit("P.A.R.K", split="[.]"))
[1] "P" "A" "R" "K"
```

  - 옵션 split에는 정규표현식이 사용됨
  - 정규표현식에서 점(.)은 다른 의미가 있음

문자열 데이터 관리: 대(소)문자로 수정!

```
> x<-c("park","lee","kim")
> y<-toupper(x)
> y
[1] "PARK" "LEE"  "KIM“

> tolower(y)
[1] "park" "lee"  "kim"
```

- 벡터 x의 첫 글자만 대문자로 변화시켜보자!

```
> substr(x,1,1) <- toupper(substr(x,1,1))
> x
[1] "Park" "Lee"  "Kim"
```

문자열 데이터 관리: 치환!

- sub(old, new, 문자열): 문자열의 첫 번째 old만 new로 치환
- gsub(old, new, 문자열): 문자열의 모든 old가 new로 치환

```
> x <- "Park hates stats. He hates math, too."

> sub("hates","loves",x)
[1] "YSP loves cakes. He hates math, too "

> gsub("hates","loves",x)
[1] "YSP loves cakes. He loves math, too"
```

예제

- "tomato1", "tomato2", "tomato3" 생성
- 첫번째 t를 T로 바꿔보기 "Tomato1", "Tomato2", "Tomato3" 생성
- 모든 t를 T로 바꿔보기 "TomaTo1", "TomaTo2", "TomaTo3" 생성

```
> y<-paste0("tomato",1:3)

> sub("t","T",y)
[1] "Tomato1" "Tomato2" "Tomato3"

> gsub("t","T",y)
[1] "TomaTo1" "TomaTo2" "TomaTo3"
```

- 문자열의 일부 삭제는 바꿀 new 부분에 " "로 입력

```
> z<-"Every body cannot do it"

> sub("not","",z)
[1] "Every body can do it"
```

예제

- Wikipedia의 HTML 테이블 불러온 자료들을 다시금 활용해보자!

```
> library(rvest)
> URL <- "https://en.wikipedia.org/wiki/World_population"
> web_1 <- read_html(URL)
> node_1 <- html_nodes(web_1, "table")
> tbl_1 <- html_table(node_1[7])
```

```
> top <- tbl_1[[1]]
> names(top) <- c("rank","country","pop","area","density")
> str(top)
'data.frame':    10 obs. of  5 variables:
 $ rank   : int  1 2 3 4 5 6 7 8 9 10
 $ country: chr  "Singapore" "Bangladesh" "Taiwan" "Lebanon" ...
 $ pop    : chr  "5,638,700" "166,490,000" "23,577,488" "6,093,509" ...
 $ area   : chr  "710" "143,998" "36,193" "10,452" ...
 $ density: chr  "7,942" "1,156" "651" "583" ...
```

예제

- 변수 pop(인구수)의 평균 계산해보기!

```
> pop <- top$pop

> mean(pop)
[1] NA
Warning message:
In mean.default(pop) : 인자가 수치형 또는 논리형이 아니므로 NA를 반환합니다
```

- 어떻게 하면 될까?
  1) pop에 있는 콤마(,)를 제거하고
  2) 숫자처럼 변수 pop를 다뤄준다면, 되지 않을까?

```
> pop<-gsub(",","",pop)

> mean(as.numeric(pop))

[1] 165013903
```

벡터의 비교연산!

◎ 비교/논리 연산자

연산자	기능
<	작다
<=	작거나 같다
>	크다
>=	크거나 같다
==	같다
!=	같지 않다
! X	X가 아니다 (NOT)
x|y	x 또는 y (OR)
x&y	x 그리고 y (AND)

벡터의 비교연산!

◎ 벡터끼리의 비교연산

```
> x<-c(1,5,3)
> y<-c(4,2,3)

> x>y
[1] FALSE  TRUE FALSE

> x>=y
[1] FALSE  TRUE  TRUE

> x<y
[1]  TRUE FALSE FALSE

> x<=y
[1]  TRUE FALSE  TRUE

> x==y
[1] FALSE FALSE  TRUE

> x!=y
[1]  TRUE  TRUE FALSE
```

벡터의 비교연산!

◎ 벡터와 스칼라의 비교연산: 순환법칙이 적용됨

```
> x<-1:5

> x>3
[1] FALSE FALSE FALSE  TRUE  TRUE

> x<3
[1]  TRUE  TRUE FALSE FALSE FALSE

> x<=2 | x<=5
[1] TRUE TRUE TRUE TRUE TRUE

> x<=3 & x>=1
[1]  TRUE  TRUE  TRUE FALSE FALSE
```

◎ 각 요소끼리의 비교가 아닌 전체로 비교를 원할때: any( ), all( )

```
> x<-1:5

> any(x>=2)
[1] TRUE

> all(x>=3)
[1] FALSE
```

벡터의 비교연산!

◎ 벡터의 구성요소 중 특정조건을 만족하는 요소의 개수 혹은 비율

```
> x<-1:5

> x>=3
[1] FALSE FALSE  TRUE  TRUE  TRUE

> sum(x>=3)
[1] 3

> mean(x>=3)
[1] 0.6
```

전체 1,2,3,4,5 중 3보다 큰 요소의 개수

전체 1,2,3,4,5 중 3개/5개(총 수) = 0.6

벡터의 인덱싱!

◎ 대괄호에 색인 직접 지정

- 양수 색인: 해당요소를 선택
- 음수 색인: 해당요소를 제외하고 선택

◎ 논리형 벡터에 의한 색인 지정

- TRUE의 위치에 있는 요소 선택
- 논리형 벡터: 비교 결과에 따른 선택

```
> x<-c(80,88,90,93,95,94,99,78,101)

> x>=mean(x)
[1] FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE

> x[x>=mean(x)]
[1]  93  95  94  99 101
```

예제

◎ Z검정

$(x-\bar{x})$ ← 편차

$(x-\bar{x})^2$ ← 편차 제곱

$\sum(x-\bar{x})^2$ ← 편차 제곱합

$\frac{\sum(x-\bar{x})^2}{n}$ ← 편차 제곱의 평균(분산)

$\sqrt{\frac{\sum(x-\bar{x})^2}{n}}$ ← 표준편차(루트분산)

$$z = \frac{x-\mu}{\frac{\delta}{\sqrt{n}}}$$

- z검증을 통한 정규화
- (관찰값-평균)/ 분산

예제

◎ Z검정

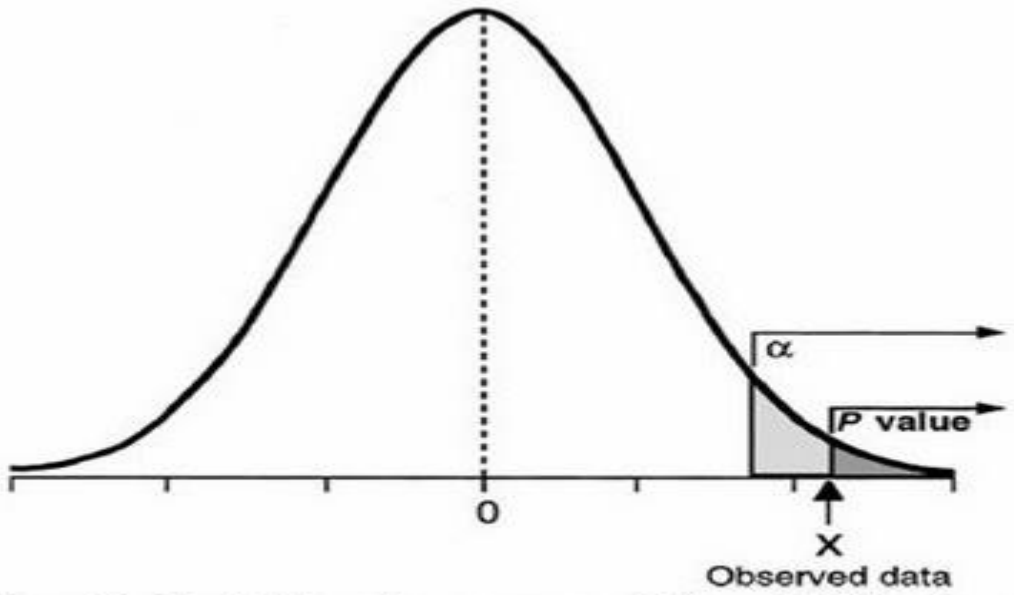

Figure 3. The bell-shaped curve represents the probability of every possible outcome under the null hypothesis. Both α (the type I error rate) and the *P* value are "tail areas" under this curve. The tail area for α is set before the experiment, and a result can fall anywhere within it. The *P* value tail area is known only after a result is observed, and, by definition, the result will always lie on the border of that area.

N (0 , 1)

- 평균이 0
- 분산이 1

예제

◎ 방금 전 입력한 벡터 x의 표준편차 구간별 관찰값 확인하기!

1) 평균으로부터 1 표준편차 범위 안의 데이터
2) 평균으로부터 1 표준편차 범위를 벗어나고 2 표준편차 사이에 있는 데이터
3) 평균으로부터 2 표준편차를 벗어나는 데이터

```
> z<-(x-mean(x))/sd(x)

> x[abs(z)<=1]
[1] 88 90 93 95 94

> x[abs(z)>1&abs(z)<=2]
[1]  80  99  78 101

> x[abs(z)>2]
numeric(0)
```

변수 변환(Recoding)

◎ 연속형 변수 → 범주형 변수로 변환: Recoding

- 변환방법 1: 논리형 벡터 이용
- 변환방법 2: 함수 cut( ) 이용

◎ 예제: 숫자형 벡터 x를 '90 이상', '90 미만 80 이상', '80 미만'의 세 등분으로 구분하고 각각 A, B, C의 값을 갖는 요인으로 변환

```
> x <- c (80, 88, 90, 93, 95, 94, 100, 78, 65)
```

변수 변환(Recoding)

1) '이상'을 기준으로 논리형 벡터 활용

```
 x_1<-(x>=0)+(x>=80)+(x>=90)
> cat.x<-factor(x_1,labels = c("c","B","A"))
> cat.x
[1] B B A A A A A c c
Levels: c B A
```

2) '미만'을 기준으로 논리형 벡터 활용

```
> x_2<-(100>=x)+(90>x)+(80>x)
> cat.x<-factor(x_2,labels = c("A","B","C"))
> cat.x
[1] B B A A A A A C C
Levels: A B C
```

변수 변환(Recoding)

3) '이상과 미만'을 기준으로 논리형 벡터 활용

```
> x_3<- 1*(x>=0 & x<80)+2*(x>=80 & x<90)+3*(x>=90)
> cat.x<-factor(x_3,labels = c("c","B","A"))
> cat.x
[1] B B A A A A A c c
Levels: c B A
```

변수 변환(Recoding)

4) 함수 cut( )에 의한 방법

```
> cat.x <- cut(x, breaks=c(0,80,90,100), include.lowest=TRUE,
               right=FALSE, labels=c("c","b","a"))
> cat.x

[1] b b a a a a a c c
Levels: c b a
```

breaks: 구간의 최소값, 최대값을 포함한 구간 설정 벡터

right: 구간이 $(a < x \leq b)$이면 TRUE, $(a \leq x < b)$이면 FALSE. 디폴트는 TRUE

include.lowest: 구간의 최소값(right=TRUE) 또는 최대값(right=FALSE)과 같은 관찰값도 변환에 포함시킬지 여부. 디폴트는 FALSE.

labels: 수준(level)의 라벨 지정

변수 변환(Recoding)

4) 함수 cut( )으로 순서형 요인 생성: 옵션 ordered_result = TRUE

```
> cat.x <- cut(x, breaks=c(0,80,90,100), include.lowest=TRUE,
               right=FALSE, labels=c("c","b","a"), ordered_result = TRUE)

> cat.x

[1] b b a a a a a c c
Levels: c < b < a
```

결측값

- 결측값 기호: NA(not available)
- 데이터에 결측값 포함여부 확인: 함수 is.na( )

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

> is.na(x)

[1] FALSE FALSE FALSE FALSE  TRUE
```

- NA가 포함된 데이터 계산해보면???

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

> mean(x);sd(x)

[1] NA
[1] NA
```

결측값

- 연산과정에서 NA를 제거하는 방법은? 옵션 na.rm= TRUE를 설정

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

> mean(x);sd(x)

[1] NA
[1] NA
```

```
> mean(x,na.rm = TRUE);sd(x,na.rm = TRUE)
[1] 2.5
[1] 1.290994
```

데이터 객체를 변환시켜 연산해보자!

◎ 데이터 객체의 유형 전환 함수

유형 확인	유형 전환
is.numeric( )	as.numeric( )
is.character( )	as.character( )
is.vector( )	as.vector( )
is.factor( )	as.factor( )
is.matrix( )	as.matrix( )
is.data.frame( )	as.data.frame( )

데이터 객체를 변환시켜 연산해보자!

◎ 연산자 우선순위

- 예: 0부터 n-1까지의 정수 표현

```
> n <- 5

> 0:n-1
[1] -1 0 1 2 3 4

> 0:(n-1)
[1] 0 1 2 3 4
```

우선순위: 콜론 (:) 연산자 > 빼기 (-) 연산자

데이터 객체를 변환시켜 연산해보자!

◎ 연산자들 간의 우선순위

연산자	의미
^	지수
:	콜론 연산자
%any%	특별 연산자
* /	곱하기, 나누기
+ -	더하기, 빼기
== != < > <= >=	비교 연산자
!	논리 연산자 NOT
& &&	논리 연산자 AND
| ||	논리 연산자 OR

◎ %any% 연산자

- %% 나머지 연산자
- %/% 정수 나누기 연산자 (몫)
- %*% 행렬의 곱하기 연산자
- %in% 데이터에 특정 값의 포함여부 확인

```
> 1 %% 2; 2 %% 2; 3 %% 2; 4 %% 2
[1] 1
[1] 0
[1] 1
[1] 0
> 1 %/% 2; 2 %/% 2; 3 %/% 2; 4 %/% 2
[1] 0
[1] 1
[1] 1
[1] 2
```

```
> c(0,3) %in% 1:3
[1] FALSE TRUE
```

3-4장. 행렬 데이터 다루기

박영식(youngsikrex@naver.com)

행렬 다루기

◎ 행렬 생성하기 (1: matrix 함수 이용하기, 2: rbind 및 cbind이용하기)

◎ 행렬 연산

◎ 행이나 열단위로 행렬에 함수 적용

행렬의 생성

◎ matrix가 아닌 다른방법: rbind와 cbind로 생성해보기

- 함수 cbind( ): 벡터들을 열 단위로 묶어 행렬 생성
- 함수 rbind( ): 벡터들을 행 단위로 묶어 행렬 생성

```
> x1 <- 1:3

> x2 <- letters[1:3]

> x_12<-cbind(x1,x2)

> x_12

     x1  x2
[1,] "1" "a"
[2,] "2" "b"
[3,] "3" "c“
```

```
> x_12a<-rbind(x1,x2)

> x_12a
   [,1] [,2] [,3]
x1 "1"  "2"  "3"
x2 "a"  "b"  "c"

> typeof(x_12)
[1] "character"

> typeof(x_12a)
[1] "character"
```

행렬의 생성

◎ 기존 행렬에 행이나 열 추가해보기

```
> #기존 행렬에 열과 행추가

> cbind(x_12,x3=1:3)
     x1  x2  x3
[1,] "1" "a" "1"
[2,] "2" "b" "2"
[3,] "3" "c" "3"

> rbind(x_12,7:8)
     x1  x2
[1,] "1" "a"
[2,] "2" "b"
[3,] "3" "c"
[4,] "7" "8"
```

```
> # 길이가 다른 경우 순환법칙 적용

> x1 <- 1:4
> x2 <- 5:6
> x3 <- 7

> cbind(x1,x2,x3)

     x1 x2 x3
[1,]  1  5  7
[2,]  2  6  7
[3,]  3  5  7
[4,]  4  6  7
```

행렬의 연산

◎ 선형대수 문제뿐만이 아닌 다양한통계에서도 활용되는 행렬 연산

연산자 및 함수	기능
+ - * / ^ A %*% B	행렬을 구성하는 숫자 각각에 적용 행렬 A와 B의 곱하기
cbind(A, B, . . .)	행렬이나 벡터를 열 단위로 결합
colMeans(A)	행렬 A 각 열의 평균값으로 구성된 벡터
crossprod(A) crossprod(A, B)	t(A) %*% A (ATA) t(A) %*% B (ATB)
colSums(A)	행렬 A 각 열의 합으로 구성된 벡터
diag(A) diag(x) diag(k)	행렬 A의 대각선 원소로 구성된 벡터 벡터 x를 대각선 원소로 하는 대각행렬 k * k 단위행렬

- 표: 행렬(A, B), 벡터(x, b), 스칼라(k)

eigen(A)	행렬 A의 고유값과 고유벡터로 구성된 리스트
rbind(A, B, . . .)	행렬이나 벡터를 행 단위로 결합
rowMeans(A) rowSums(A)	행렬 A 각 행의 평균값으로 구성된 벡터 행렬 A 각 행의 합으로 구성된 벡터
solve(A) solve(A, b)	행렬 A의 역행렬 연립방정식 Ax=b의 해
t(A) tcrossprod(A) tcrossprod(A, B)	행렬 A의 전치 (AT) A %*% t(A) A %*% t(B)

- 표: 행렬(A, B), 벡터(x, b), 스칼라(k)

```
> A
     [,1] [,2]
[1,]    1    2
[2,]    3    4
> B
     [,1] [,2]
[1,]    5    6
[2,]    7    8
> A*B
     [,1] [,2]
[1,]    5   12
[2,]   21   32
> A%*%B
     [,1] [,2]
[1,]   19   22
[2,]   43   50
> crossprod(A,B)
     [,1] [,2]
[1,]   26   30
[2,]   38   44
> t(A)%*%B
     [,1] [,2]
[1,]   26   30
[2,]   38   44
```

```
> colMeans(A)
[1] 2 3
> rowSums(A)
[1] 3 7
> diag(A)
[1] 1 4
> x <- c(10,20); diag(x)
     [,1] [,2]
[1,]   10    0
[2,]    0   20
> diag(2)
     [,1] [,2]
[1,]    1    0
[2,]    0    1
> solve(A)
     [,1] [,2]
[1,] -2.0  1.0
[2,]  1.5 -0.5
> solve(A)%*%A
     [,1]         [,2]
[1,]    1 4.440892e-16
[2,]    0 1.000000e+00
> b<-c(5,6); solve(A,b)
[1] -4.0  4.5
```

$$x + 2y = 5$$
$$3x + 4y = 6$$

행과 열의 단위에 함수적용

◎ 함수 apply ( )

- apply(행렬, 1, 함수): 행 단위로 행렬에 함수 적용
- apply(행렬, 2, 함수): 열 단위로 행렬에 함수 적용

```
> A
    trial(1) trial(2) trial(3) trial(4)
Park      0.8      1.1      0.0      0.6
Kim       1.3      1.3      1.2      1.4
Lee       1.0      1.3      0.2      0.6
```

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

> apply(A,1,sd);apply(A,1,range)
                Park                 Kim                 Lee
   0.46457866          0.08164966    0.47871355

   Park Kim Lee
[1,]  0.0 1.2 0.2
[2,]  1.1 1.4 1.3
```

행과 열의 단위에 함수적용

◎ 함수 rowMeans( )를 이용한 방법

- 이런 종류의 함수: colMeans( ), rowSums( ), colSums( )
- 단점: apply에 비해 자유도가 떨어짐

```
> rowMeans(A)
  Park       Kim       Lee
0.625     1.300     0.775

> rowSums(A)
Park       Kim       Lee
 2.5        5.2       3.1
```

3-5장. 데이터프레임 다루기

박영식(youngsikrex@naver.com)

데이터 프레임 다루기

◎ 데이터 프레임: 통계 데이터 셋에 가장 적합함
- 가장 많이 사용되는 객체
- 함수 read.table( )등으로 불러온 외부 파일은 데이터 프레임으로 저장
- 데이터 다듬기는 주로 데이터 프레임이 대상

◎ 데이터 프레임을 대상으로 하는 다루기 기법
1. 데이터 프레임 가공을 위한 함수
2. 데이터 프레임 가공 및 수정
3. 데이터의 취사선택
4. 데이터 프레임의 결합
5. 데이터 프레임에 함수 적용
6. 데이터의 재구성

데이터 프레임을 편리하게 사용하기

◎ 데이터 프레임을 대상으로 하는 통계 분석
- 데이터 프레임의 개별 변수를 벡터 형태로 선택하여 분석 진행
- 인덱싱 기법에 의한 변수 선택: 매우 번거로움 ex)airquality$Temp 등

◎ 편리성을 높인 데이터 프레임에 수식 적용
- 함수 with( )
- 함수 attach( )

with()의 사용법

◎ 함수 with ( )의 사용법

- 일반적인 사용 형태: with(데이터 프레임, R명령문)
  - with( ) 함수 내에서는 지정된 데이터 프레임의 변수(Variable)을 인덱싱 없이 사용이 가능함.

- 예제: 데이터 프레임 airquality

  - 미국 뉴욕시의 공기 질과 관련된 데이터
  - 변수 Temp의 표준화: $(x - \bar{x})/s$

```
> z.Temp <- (Temp-mean(Temp))/sd(Temp)
Error: object 'Temp' not found
>
> z.Temp<-(airquality$Temp-mean(airquality$Temp)/sd(airquality$Temp))
>
> z.Temp<-with(airquality, (Temp-mean(Temp)/sd(Temp)))
```

attach()의 사용법

◎ 함수 attach ( ) 의 사용법

- 여러 줄을 명령문에서 특정 데이터 프레임을 계속 이용할 경우

```
> attach(airquality)
>
> mean(Temp); mean(Wind)
[1] 77.88235
[1] 9.957516
> sd(Temp); sd(Wind)
[1] 9.46527
[1] 3.523001
>
> detach(airquality)
>
> mean(Temp); mean(Wind)
Error in mean(Temp) : object 'Temp' not found
>
> sd(Temp); sd(Wind)
Error in is.data.frame(x) : object 'Temp' not found
```

attach()의 사용법:주의사항1

◎ 함수 attach ( ) 의 사용시 주의할 점1

- DF 변수 중 현 작업공간에 다른 객체와 같은 이름이 같은 변수가 있는 경우

```
> attach(airquality)

> cor(Temp, Wind)
[1] -0.4579879

> Temp <-c(77,65,89,80)

> cor(Temp,Wind)
Error in cor(Temp, Wind) : 호환되지 않는 차원들입니다

> length(Wind)
[1] 153

> Temp
[1] 77 65 89 80

> detach(airquality)

> Wind
Error: object 'Wind' not found
```

데이터 프레임의
변수 Temp보다
벡터 Temp가 더 우선 순위

attach()의 사용법:주의사항1

◎ 함수 attach ( ) 로 불러오는 데이터 프레임의 변수가 현재 작업공간에 있는 다른 객체와 이름이 같으면 경고가 뜬다

```
> Temp <- c(77,65,89,80)

> attach(airquality)
The following object is masked _by_ .GlobalEnv:

    Temp

> Temp; mean(Temp)
[1] 77 65 89 80
[1] 77.75

> mean(airquality$Temp)
[1] 77.88235

> rm(Temp)

> mean(Temp)
[1] 77.88235
```

데이터 프레임의 변수 Temp라는 이름이 있을시
1) 인덱싱 기법 사용
2) 벡터 Temp를 제거 혹은 치환 후 사용

attach( )의 사용법:주의사항2

◎ 함수 attach ( ) 로 불러와진 데이터 프레임이 임시적으로 그 유형을 유지하려는 경향이 있음.

```
> x <- c(19,35,32,27,26)
> y <- c("M","M","F","F","F")
> z <- c(2000,3100,3800,3600,3400)
> df1 <- data.frame(age=x, gender=y, income=z)
> df1
  age gender income
1  19      M   2000
2  35      M   3100
3  32      F   3800
4  27      F   3600
5  26      F   3400
```

attach( )의 사용법:주의사항2

◎ 함수 attach ( ) 로 불러와진 데이터 프레임이 임시적으로 그 유형을 유지하려는 경향이 있음.

```
> attach(df1)
> income
[1] 2000 3100 3800 3600 3400

> df1$income<-c(2500,3600,4100,3000,6000)
> df1
  age gender income
1  19      M   2500
2  35      M   3600
3  32      F   4100
4  27      F   3000
5  26      F   6000

> income
[1] 2000 3100 3800 3600 3400
```

```
> detach(df1)
> attach(df1)
>income
[1] 2500 3600 4100 3000 6000
```

데이터 프레임의 수정

◎ 통계분석을 진행하는 과정에서 데이터 프레임의 내용을 수정해야 하는 경우가 종종 존재함
- 데이터 프레임에 새 변수 생성
- 특정 조건을 만족하는 관찰값(행) 선택
- 몇몇 변수만 선택하기
- 변수 이름을 수정
- 데이터 프레임의 정렬

◎ 현재 3장에서 소개되는 기법: base R에 의한 방법

◎ 앞으로 다음 장에서 소개할 기법: dplyr에 의한 방법
- 훨씬 더 간단하고 효과적 작업이 가능
- 앞으로 주로 활용할 방법

데이터 프레임에 새 변수 생성

◎ 숫자형 변수 생성

- 생성된 변수가 데이터 프레임의 일원이 되도록 해야 함
- '데이터 프레임$변수'의 형태 사용

```
> df2 <- data.frame(x1=c(1,2,3,4),x2=c(5,6,7,8))
>
> df2$sum_x <- df2$x1+df2$x2
> df2$avg_x <- with(df2,(x1+x2)/2)
>
> df2
  x1 x2 sum_x avg_x
1  1  5     6     3
2  2  6     8     4
3  3  7    10     5
4  4  8    12     6
```

데이터 프레임에 새 변수 생성

◎ 함수 transform( )에 의한 방법

```
> df2 <- data.frame(x1=c(1,2,3,4),x2=c(5,6,7,8))
> df2<-transform(df2,sum_x=x1+x2,avg_x=(x1+x2)/2)
> df2
  x1 x2 sum_x   avg_x
1  1  5     6     3
2  2  6     8     4
3  3  7    10     5
4  4  8    12     6
```

◎ 함수 within( )에 의한 방법

```
> df2 <- data.frame(x1=c(1,2,3,4), x2=c(5,6,7,8))
> df2 <- within(df2,
        {
        sum_x <- x1+x2
        avg_x <- (x1+x2)/2
        })
```

기존의 숫자형 변수를 요인으로 변환

◎ 함수 within( ) 사용

```
> df2<-within(df2,
  {
    type <- character(0)
    type[sum_x<10] <- 'Small'
    type[sum_x>=10] <- 'Large'
    type <- factor(type,order=TRUE, levels= c('Small','Large'))
   })
> df2
 x1 x2 avg_x sum_x  type
1  1  5    3     6 Small
2  2  6    4     8 Small
3  3  7    5    10 Large
4  4  8    6    12 Large
```

◎ '데이터 프레임$변수' 형태 사용

```
> df2$type[df2$sum_x<10] <- "Small"
> df2$type[df2$sum_x>=10] <- "Large"
> df2$type <-factor(df2$type,order=TRUE,level=c("Small","Large")
```

데이터 프레임에 행 or 열 추가하기!

◎ 데이터 셋 구성 후 새로운 관찰값 추가 및 새로운 변수 추가

◎ 함수 rbind( )와 cbind( ): 결합되는 객체 중 데이터 프레임이 존재한다면 데이터 프레임으로 출력됨

◎ 기존의 데이터 프레임에 행 추가

- 추가되는 행이 데이터 프레임일 경우: 결합되는 데이터 프레임의 변수(열=variable=feature)의 개수와 변수의 이름이 같게 설정되어야 함

```
> df3 <- data.frame(x1=1:3,x2=4:6,x3=7:9)

> rbind(df3,data.frame(x1=1,x2=2,x3=3))
  x1 x2 x3
1  1  4  7
2  2  5  8
3  3  6  9
4  1  2  3

> rbind(df3,data.frame(x1=1,x2=2))
Error in rbind(deparse.level, ...) :
  numbers of columns of arguments do not match

> rbind(df3,data.frame(y1=1,y2=2,y3=3))
Error in match.names(clabs, names(xi)) :
  names do not match previous names
```

데이터 프레임에 행 or 열 추가하기!

- 벡터를 행 단위로 결합하는 경우: 벡터의 각 요소에 이름 부여할 필요 없음.
  길이가 다른 경우 순환법칙 적용

```
> df3 <- data.frame(x1=1:3,x2=4:6,x3=7:9)

> rbind(df3,1:3)
  x1 x2 x3
1  1  4  7
2  2  5  8
3  3  6  9
4  1  2  3

> rbind(df3,1:2)
  x1 x2 x3
1  1  4  7
2  2  5  8
3  3  6  9
4  1  2  1
```

데이터 프레임: 열 추가!

- 결합 대상 객체의 행 개수는 **반드시** 같아야 함

```
> df3 <- data.frame(x1=1:3,x2=4:6,x3=7:9)

> cbind(df3,data.frame(x4=10:12))
  x1 x2 x3 x4
1  1  4  7 10
2  2  5  8 11
3  3  6  9 12

> cbind(df3,x4=10:11)
Error in data.frame(..., check.names = FALSE) :
  arguments imply differing number of rows: 3, 2
```

데이터 프레임: 변수(=열) 이름 수정!

◎ 수정 방법

- 함수 names( )
- 함수 reshape::rename( )

◎ 예제: df1의 변수 x를 age로, y는 gender, z를 income으로 수정

```
> x <- c(24,28,31,25)
> y <- c("F","M","F","F")
> z <- c(2000,3100,3800,2800)
>
> df1<-data.frame(x,y,z)
> df1
  x y    z
1 24 F 2000
2 28 M 3100
3 31 F 3800
4 25 F 2800
```

데이터 프레임: 변수(=열) 이름 수정!

◎ 함수 names( )에 의한 방법

```
> #함수 names( )에 의한 방법
> names(df1)
[1] "x" "y" "z"
> names(df1)<-c("age","gender","income")
> names(df1)
[1] "age"    "gender" "income"
```

- > names(df1)<-c(x="age", y="gender", z="income")로 작성하여도 무방!
- 이름을 할당하는 문자형 벡터의 길이가 변수의 개수와 같아야 함.
- 변수가 많고, 그 중 일부만 수정하는 경우에는 적합하지 않은 방법


```
> #함수 names( )에 의한 방법
> names(df1)
[1] "x" "y" "z"
> names(df1)<-c("age","gender","income")
> names(df1)
[1] "age"    "gender" "income"
```

데이터 프레임: 변수(=열) 이름 수정!

◎ 함수 names( )에 의한 방법

```
> #함수 names( )에 의한 방법
> names(df1)
[1] "x" "y" "z"
> names(df1)<-c("age","gender","income")
> names(df1)
[1] "age"    "gender" "income"
```

- > names(df1)<-c(x="age", y="gender", z="income")로 작성하여도 무방!
- 이름을 할당하는 문자형 벡터의 길이가 변수의 개수와 같아야 함.
- 변수가 많고, 그 중 일부만 수정하는 경우에는 적합하지 않은 방법

```
> names(df1)<-c("age","gender")
> names(df1)
[1] "age"    "gender" NA
```

데이터 프레임: 변수(=열) 이름 수정!

◎ 패키지 reshape의 함수 rename( )을 이용하는 방법

```
> library(reshape)
> df1<-data.frame(x,y,z)

> df1<-rename(df1,c(x="age", y="gender", z="income"))
> names(df1)
[1] "age"    "gender" "income"
```

- 변수가 많고, 그 중 일부만 수정하는 경우가 가능
- > rename(df1)<-c(x="age", y="gender", z="income")로 작성하여야 함!!!

```
> df1<-data.frame(x,y,z)
> df1<-rename(df1,c(x="age",y="gender"))
> names(df1)
[1] "age"    "gender" "z"
```

데이터 프레임의 정렬!

◎ 벡터의 정렬: 함수 sort( ), order( )

```
> #벡터의 정렬: 함수 sort( ), order( )
> x<-c(24,28,31,25)
[1] 24 28 31 25
> sort(x)
[1] 24 25 28 31
> sort(x,decreasing = TRUE)
[1] 31 28 25 24
```

```
> x
[1] 24 28 31 25
> order(x)
[1]  1  4  2  3
```

```
> x[order(x)]
[1] 24 25 28 31
> x[order(-x)]
[1] 31 28 25 24
```

데이터 프레임의 정렬: 함수 order( )활용

◎ age를 기준으로 하여 오름차순 정렬을 해보자.

```
> df1
  age gender income
1  24      F   2000
2  28      M   3100
3  31      F   3800
4  25      F   2800

> df1[order(df1$age),]
  age gender income
1  24      F   2000
4  25      F   2800
2  28      M   3100
3  31      F   3800
```

- 여러 변수를 순차적으로 기준 설정하여 정렬하는 방법: age(1차 오름차순), gender (2차 오름차순), income(3차 오름차순)

```
> #여러 변수를 기준으로 하는 정렬:
> df3<-data.frame(age=c(24,28,28,24),gender=c("F","M","F","F"),income=c(2000,3100,3800,2800))
> df3
  age gender income
1  24      F   2000
2  28      M   3100
3  28      F   3800
4  24      F   2800
```

```
> #1
> df3[order(df3$age,df3$gender,df3$income),]
  age gender income
1  24      F   2000
4  24      F   2800
3  28      F   3800
2  28      M   3100
>
> #2 with를 활용하기
> with(df3,df3[order(age, gender, income), ])
  age gender income
1  24      F   2000
4  24      F   2800
3  28      F   3800
2  28      M   3100
```

데이터 프레임의 정렬: 함수 order( )활용

- 응용: age(1차 오름차순), gender(2차 오름차순), income(3차 내림차순)

```
> #1-1, 2-1 income 부분만 내림차순으로 하고 싶다면?

> df3[order(df3$age,df3$gender,-df3$income),]
  age gender income
4  24      F   2800
1  24      F   2000
3  28      F   3800
2  28      M   3100

> with(df3,df3[order(age,gender,-income),])
  age gender income
4  24      F   2800
1  24      F   2000
3  28      F   3800
2  28      M   3100
```

데이터의 취사선택

◎ 분석과정에서 데이터 일부분의 선택 혹은 제외

◎ 데이터에 있는 결측값 제거

◎ 대규모 데이터에서 임의표본 추출

데이터 선택: 인덱싱 vs subset( )

◎ 대괄호를 이용한 인덱싱: 선택조건이 복잡하다면 매우 번거로움

◎ 예제- MASS:: Cars93

- 패키지 MASS에 있는 데이터 프레임 Cars93;
  1993년 미국에서 판매된 자동차에 관한 데이터

```
> library(MASS)
> names(Cars93)
 [1] "Manufacturer"      "Model"         "Type"          "Min.Price"
 [5] "Price"             "Max.Price"     "MPG.city"      "MPG.highway"
 [9] "AirBags"           "DriveTrain"    "Cylinders"     "EngineSize"
[13] "Horsepower"        "RPM"           "Rev.per.mile"  "Man.trans.avail"
[17] "Fuel.tank.capacity" "Passengers"   "Length"        "Wheelbase"
[21] "Width"             "Turn.circle"   "Rear.seat.room" "Luggage.room"
[25] "Weight"            "Origin"        "Make"
```

데이터를 선택해보자!

◎ 변수 MPG.city가 30보다 큰 값을 갖는 Model은 무엇인가? (with를 활용!)

◎ 변수 Cylinders가 4이고 Manufacturer가 Hyundai인 자동차의 Model과 Min.Price, Max.Price의 값을 출력하라. (attach를 활용)

인덱싱 기법에 의한 데이터 선택

1번 문제

```
> with(Cars93,Cars93[MPG.city>30,"Model"])
[1] Festiva Metro     Civic         LeMans   Justy      Swift Tercel
93 Levels: 100 190E 240 300E 323 535i 626 850 90 900 Accord Achieva ... Vision
```

인덱싱 기법에 의한 데이터 선택

2번 문제

```
> attach(Cars93)

> my_vars <-c("Model", "Min.Price", "Max.Price")
> my_case <-Cylinders==4&Manufacturer=="Hyundai"

> Cars93[my_case,my_vars]
    Model  Min.Price  Max.Price
44   Excel        6.8        9.2
45 Elantra        9.0       11.0
46  Scoupe        9.1       11.0
47  Sonata       12.4       15.3

> detach(Cars93)
```

함수 subset( )에 의한 데이터 선택

- 사용법: subset(data, select, subset)
  select: 선택 혹은 제거될 변수 지정
  subset: 케이스를 선택하기 위한 조건 지정

1번 문제

```
> subset(Cars93,select = Model, subset = (MPG.city>30))

    Model
31 Festiva
39   Metro
42   Civic
73  LeMans
80   Justy
83   Swift
84  Tercel
```

함수 subset( )에 의한 데이터 선택

- 사용법: subset(data, select, subset)
  select: 선택 혹은 제거될 변수 지정
  subset: 케이스를 선택하기 위한 조건 지정

2번 문제

```
> subset(Cars93,select = c(Model,Min.Price,Max.Price),
          subset = (Cylinders==4&Manufacturer=="Hyundai"))

      Model  Min.Price  Max.Price
44     Excel        6.8        9.2
45   Elantra        9.0       11.0
46   Scoupe         9.1       11.0
47    Sonata       12.4       15.3
```

◎ 여러 개의 변수를 선택하고자 하는 경우 함수 c( ) 이용

함수 subset( )에 의한 데이터 선택

- 순서를 이용한 변수 선택: 함수 c( )대신 콜론 연산자 이용
  두 번째 변수부터 여덟 번째 변수 선택

c( ) 대신 콜론 연산자 이용

```
> subset(Cars93,select = Model: MPG.highway,
          subset = (Cylinders==4&Manufacturer=="Hyundai"))
```

	Model	Type	Min.Price	Price	Max.Price	MPG.city	MPG.highway
44	Excel	Small	6.8	8.0	9.2	29	33
45	Elantra	Small	9.0	10.0	11.0	22	29
46	Scoupe	Sporty	9.1	10.0	11.0	26	34
47	Sonata	Midsize	12.4	13.9	15.3	20	27

함수 subset( )에 의한 데이터 선택

- 특정 변수의 제외: 마이너스 부호

```
> subset(Cars93,select = -(Model: Origin),
            subset = (Cylinders==4&Manufacturer=="Hyundai"))

 Manufacturer              Make
44     Hyundai     Hyundai Excel
45     Hyundai   Hyundai Elantra
46     Hyundai   Hyundai Scoupe
47     Hyundai   Hyundai Sonata
```

함수 subset( )에 의한 데이터 선택

- 특정 변수의 제외: 마이너스 부호

```
> subset(Cars93,select = -(Model: Origin),
            subset = (Cylinders==4&Manufacturer=="Hyundai"))

 Manufacturer              Make
44     Hyundai     Hyundai Excel
45     Hyundai   Hyundai Elantra
46     Hyundai   Hyundai Scoupe
47     Hyundai   Hyundai Sonata
```

데이터의 취사선택

◎ 분석과정에서 데이터 일부분의 선택 혹은 제외

◎ 데이터에 있는 결측값 제거

◎ 대규모 데이터에서 임의표본 추출

데이터 프레임에서 결측값 제거

◎ 함수 na.omit( )

◎ 데이터 프레임에서 결측값을 제거:
- 결측값이 존재시 연산이 불가한 경우 다수
- 결측값이 있는 행 전체를 삭제하는 함수

◎ 결측값 제거시 많은 정보의 손실이 발생할 가능성이 단점

```
> head(airquality)
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
5    NA      NA 14.3   56     5   5
6    28      NA 14.9   66     5   6
```

데이터 프레임에서 결측값 제거

```
> head(na.omit(airquality))
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
7    23     299  8.6   65     5   7
8    19      99 13.8   59     5   8
```

```
> air <- na.omit(airquality)

>dim(air); dim(airquality)
[1] 111   6
[1] 153   6
```

대규모 데이터에서 임의표본 추출

◎ 빅데이터 시대가 되어감에 따라 데이터 셋의 규모가 방대해짐

◎ 데이터 세트의 일부분을 임의 추출하여 다른 목적으로 활용

◎ 단순임의추출방법으로 데이터의 일부분을 추출하고자 할때:

- 함수 sample( )

```
> sample(1:5, size=3)
[1] 1 5 4
> sample(1:5, size=3,replace=TRUE)
[1] 3 4 4
```

예제: airquality의 전체 관찰값 중 10개 관찰값을 비복원추출

◎ 옆 학우와 데이터 출력값을 비교해보자!

```
> my.index <- sample(1:nrow(airquality),5)

> airquality[my.index,]
```

	Ozone	Solar.R	Wind	Temp	Month	Day
40	71	291	13.8	90	6	9
38	29	127	9.7	82	6	7
120	76	203	9.7	97	8	28
26	NA	266	14.9	58	5	26
5	NA	NA	14.3	56	5	5

예제: airquality의 전체 관찰값 중 10개 관찰값을 비복원추출

◎ 옆 학우와 데이터 출력값을 비교해보자!

```
> my.index <- sample(1:nrow(airquality),5)

> airquality[my.index,]
```

	Ozone	Solar.R	Wind	Temp	Month	Day
40	71	291	13.8	90	6	9
38	29	127	9.7	82	6	7
120	76	203	9.7	97	8	28
26	NA	266	14.9	58	5	26
5	NA	NA	14.3	56	5	5

데이터 프레임의 결합

◎ 데이터가 따로 수집되어 몇 개의 파일로 쪼개져 정리된 경우:

- 분석 전 데이터의 통합

(1) 데이터 수평적 결합= 변수의 확대: DataFrame의 행 수가 같고, 행이 동일한 대상에 관한 것

(2) 데이터 수직적 결합= 케이스 확대: DataFrame의 열 수가 같고,

(3) 기준열을 이용한 수평적 결합: 각 DataFrame에 기준 열(key)이 있는 경우

(1) 데이터의 수평적 결합: 함수 cbind( )

```
> data1
  name n.child
1   Kim       2
2   Lee       1
3 Park        2

> data2
   birth.year birth.place
1        1987       Seoul
2        1970    New York
3        1988     Chicago
```

```
> cbind(data1,data2)
  name n.child birth.year birth.place
1  Kim       2       1987       Seoul
2  Lee       1       1970    New York
3 Park       2       1988     Chicago
```

(2) 데이터 프레임의 수직적 결합: 함수 rbind( )

```
> data1
  name n.child
1   Kim       2
2   Lee       1
3 Park        2

> data3
  name n.child
1  Nam        2
2    Yoo      1
```

```
> rbind(data1,data3)
  name n.child
1   Kim       2
2   Lee       1
3  Park       2
4  Nam        2
5    Yoo      1
```

(3) 기준 열에 의한 수평적 결합: 함수 merge( )

```
> data1
  name n.child
1   Kim       2
2   Lee       1
3 Park        2

> data4
  name n.son
1 Park      0
2  Kim      1
3  Lee      2
4 Hyun      1
```

- 한 쪽 데이터 프레임에만 있는 케이스는 지워짐
- sql에서의 join과 비슷한 개념
- 실제적으로 활용도가 높은 방법

```
> merge(data1,data4,by='name')
  name n.child n.son
1  Kim       2     1
2  Lee       1     2
3 Park       2     0
```

데이터 프레임의 함수 적용

◎ 그룹 데이터에 함수 적용:

- 데이터 프레임에 요인이 있는 경우, 요인의 level에 따라 그룹 형성이 가능
- 특정 변수의 각 그룹별 분포 비교: 매우 중요한 분석

◎ 예제: 패키지 MASS에 있는 데이터 프레임 Cars93에는 Origin이라는 요인이 존재. 수준은 USA와 non-USA. 변수 MPG.city의 평균치를 Origin의 수준별로 구해보자

1) 함수 split( )과 sapply( )에 의한 계산
2) 함수 tapply( )에 의한 계산

함수 split( )과 sapply( )에 의한 계산

- 함수 split( ): 벡터를 요인에 따라 그룹으로 분리
- 사용법: split(벡터, 요인)
- 결과는 리스트

- 함수 sapply 또는 lapply( ): 리스트의 각 요소에 동일한 함수 적용
- 사용법: sapply(리스트, 함수) / lapply(리스트, 함수)
- sapply( ): 결과가 벡터 혹은 행렬
- lapply( ): 결과가 리스트

```
> library(MASS)

> 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);lapply(x_g,length)
$`USA`
[1] 20.95833
$`non-USA`
[1] 23.86667

$`USA`
[1] 48
$`non-USA`
[1] 45
```

```
> sapply(x_g,mean)
     USA   non-USA
20.95833  23.86667

> sapply(x_g,length)
    USA non-USA
     48      45
```

함수 tapply( )에 의한 계산

◎ 함수 tapply()의 사용법: tapply(벡터, 요인, 함수)

◎ 요인의 각 수준별로 벡터를 분리하여 함수를 적용시키자!

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

데이터 프레임의 모든 변수에 함수 적용

- 행렬의 경우 모든 열에 동일한 함수를 적용: apply(matrix, 2, FUN)
- 데이터 프레임도 2차원 배열이기에 함수 apply( )를 사용할 수 있음
- 단, 함수 apply( )를 사용하기 위해서 2차원 배열 데이터가 **모두 동일한 유형**이 되어야 함
- 각 변수의 유형이 혼재된 데이터 프레임: 함수 apply( )를 사용하기 어려운 상황이 많음

- 함수 sapply( ) 또는 lapply( )가 적합
- 데이터 프레임의 유형은 리스트

예제: 패키지 MASS에 있는 데이터 프레임 cabbages의 변수 속성 확인하기.

```
> library(MASS)

> head(cabbages, n=3)

  Cult Date HeadWt VitC
1 c39  d16    2.5   51
2 c39  d16    2.2   55
3 c39  d16    3.1   45
```

```
> sapply(cabbages, class)
    Cult     Date     HeadWt     VitC
 "factor"  "factor" "numeric" "integer"
```

```
> apply(cabbages, 2, class)
        Cult         Date      HeadWt         VitC
"character" "character" "character" "character"
```

모두 문자형인 이유는?

예제: 데이터 프레임 airquality의 모든 변수의 평균값 계산

```
> str(airquality)
'data.frame':   153 obs. of  6 variables:
 $ Ozone  : int  41 36 12 18 NA 28 23 19 8 NA ...
 $ Solar.R: int  190 118 149 313 NA NA 299 99 19 194 ...
 $ Wind   : num  7.4 8 12.6 11.5 14.3 14.9 8.6 13.8 20.1 8.6 ...
 $ Temp   : int  67 72 74 62 56 66 65 59 61 69 ...
 $ Month  : int  5 5 5 5 5 5 5 5 5 5 ...
 $ Day    : int  1 2 3 4 5 6 7 8 9 10 ...
```

- 측정 일자를 나타내는 변수 제거

```
> air<- airquality [1:4]
```

- 네 변수의 평균

```
> sapply(air, mean)
   Ozone  Solar.R     Wind     Temp
      NA       NA 9.957516 77.882353
```

- 결측값 제거 후 재계산

```
> sapply(air, mean, na.rm=TRUE)
     Ozone    Solar.R       Wind       Temp
 42.129310 185.931507   9.957516  77.882353
```

데이터의 재구성

◎ 데이터 합계: 입력된 데이터의 통합이 필요한 경우
- 함수 aggregate 행과 열의 변경
- 함수 split( )과 lapply( )

◎ 데이터 재구성:
- 패키지 reshape2의 함수 melt( ), dcast( )
- 데이터의 행과 열의 변경

(1) 데이터의 합계

◎ 함수 aggregate( )의 사용법: aggregate(data, by, FUN)

- data: 데이터 집계가 되는 대상. 대부분 DataFrame
- by: 리스트로 구성된 그룹 변수
- FUN: 데이터 집계에 사용될 함수

◎ 예제: airquality의 변수 Ozone, Solar.R, Wind, Temp의 월별 평균 계산

```
> air<-subset(airquality,select = -c(Month,Day))
> aggregate(air,by=list(Month=airquality$Month),
                FUN = mean, na.rm=TRUE)
  Month      Ozone   Solar.R     Wind     Temp
1     5   23.61538  181.2963 11.622581 65.54839
2     6   29.44444  190.1667 10.266667 79.10000
3     7   59.11538  216.4839  8.941935 83.90323
4     8   59.96154  171.8571  8.793548 83.96774
5     9   31.44828  167.4333 10.180000 76.90000
```

함수 split( )과 lapply( )에 의한 데이터 합계

◎ 함수 split( )으로 airquality를 월별로 구분

◎ 구분된 데이터 프레임의 모든 변수의 평균 계산: 함수 colMeans( )

```
> air_q <- split(air,airquality$Month)
> lapply(air_q,colMeans,na.rm=TRUE)
$`5`
   Ozone   Solar.R     Wind     Temp
 23.61538 181.29630  11.62258  65.54839
$`6`
   Ozone   Solar.R     Wind     Temp
 29.44444 190.16667  10.26667  79.10000
$`7`
    Ozone    Solar.R      Wind      Temp
 59.115385 216.483871   8.941935  83.903226
$`8`
    Ozone    Solar.R      Wind      Temp
 59.961538 171.857143   8.793548  83.967742
$`9`
   Ozone   Solar.R     Wind     Temp
 31.44828 167.43333  10.18000  76.90000
```

(2) 데이터의 재구성

- '넓게 펼쳐진' 구조의 데이터 프레임 ⟺ '좁고 긴' 구조의 데이터 프레임

(일반적인 통계 데이터 세트 구조)

	name	A	B	C
1	Park	14	12	4
2	Lee	21	15	8
3	Kim	15	5	10

	name	item	amount
1	Park	A	14
2	Lee	A	21
3	Kim	A	15
4	Park	B	12
5	Lee	B	15
6	Kim	B	5
7	Park	C	4
8	Lee	C	8
9	Kim	C	10

- reshape2의 함수 melt( )와 dcast( )이용
- melt( ): 데이터 프레임의 구조를 녹여서 해체
- dcast( ): 녹아 해체된 데이터 프레임을 원하는 구조로 변경

함수 melt( ):

- 데이터 프레임의 구조를 녹여냄
- 변수의 자료 하나하나가 한 행을 차지

◎ melt(df, id.vars, measure.vars)

- df: 구조를 녹여 해체할 데이터 프레임
- id.vars: 각 자료의 ID로 사용할 변수
- measure.vars: 해체 대상이 되는 변수. 생략 시 id.vars를 제외한 나머지 변수

예제

```
> df1
  age gender income region
1  22      F   2000      S
2  28      M   3100      S
3  34      M   3800      G
4  24      F   2800      G
```

id.vars: gender, region

```
> library(reshape2)
> df1_melt <- melt(df1,id.vars=c("gender","region"))
> df1_melt
  gender region variable value
1      F      S      age    22
2      M      S      age    28
3      M      G      age    34
4      F      G      age    24
5      F      S   income  2000
6      M      S   income  3100
7      M      G   income  3800
8      F      G   income  2800
```

함수 dcast( ):

- 녹아서 해체된 데이터 프레임의 재구성

◎ dcast(md, formula, fun=NULL)

- md: melt( )로 녹아서 해체된 데이터 프레임
- formula: 재구성할 데이터의 구조 지정

row_var1 + row_var2 + … ~ col_var1 + col_var2+…

row_var1 + row_var2 + … : 행을 새롭게 정의할 변수 조합
col_var1 + col_var2 + … : 열을 새롭게 정의할 변수 조합

- fun: 변수의 조합으로 인식된 관찰값의 개수가 하나 이상인 경우, 해당 관찰값의 합계 함수

예제

```
> df1_melt
  gender region variable value
1      F      S      age    22
2      M      S      age    28
3      M      G      age    34
4      F      G      age    24
5      F      S   income  2000
6      M      S   income  3100
7      M      G   income  3800
8      F      G   income  2800
```

- 변수 gender만으로 행 구성
- 각 변수의 조합에 region 두 케이스가 해당
- Function함수: 평균 사용

```
> dcast(df1_melt,gender~variable,fun=mean)
  gender age income
1      F  23   2400
2      M  31   3450
```

예제

```
> df1_melt
  gender region variable value
1      F      S      age    22
2      M      S      age    28
3      M      G      age    34
4      F      G      age    24
5      F      S   income  2000
6      M      S   income  3100
7      M      G   income  3800
8      F      G   income  2800
```

- 변수 region만으로 행 구성
- 각 변수의 조합에 gender두 케이스가 해당
- Function함수: 평균 사용

```
> dcast(df1_melt,region~variable,fun=mean)
  region age income
1      G  29   3300
2      S  25   2550
```

예제

```
> df1_melt
  gender region variable value
1      F      S      age    22
2      M      S      age    28
3      M      G      age    34
4      F      G      age    24
5      F      S   income  2000
6      M      S   income  3100
7      M      G   income  3800
8      F      G   income  2800
```

- 두 ID 변수를 모두 사용하여 행 구성

```
> dcast(df1_melt,gender+region~variable)
```

- 원래의 구조

-긴 구조의 데이터를 넓게 펼쳐 놓은 구조로 변경

```
> df1_melt
  gender region variable value
1      F      S      age    22
2      M      S      age    28
3      M      G      age    34
4      F      G      age    24
5      F      S   income  2000
6      M      S   income  3100
7      M      G   income  3800
8      F      G   income  2800
```

- 하나의 ID 변수로 행 구성
- 나머지 ID 변수는 열 구성에 참여

```
> dcast(df1_melt,gender~region+variable)
  gender G_age G_income S_age S_income
1      F    24     2800    22     2000
2      M    34     3800    28     3100

> dcast(df1_melt,region~gender+variable)
  region F_age F_income M_age M_income
1      G    24     2800    34     3800
2      S    22     2000    28     3100
```

예제: 좌측의 데이터들을 우측으로 변경해보기

```
> df2
  name  A  B  C
1 Park 14 12  4
2  Lee 21 15  8
3  Kim 15  5 10
```

```
> melt(df2,id.vars="name")
  name variable value
1 Park        A    14
2  Lee        A    21
3  Kim        A    15
4 Park        B    12
5  Lee        B    15
6  Kim        B     5
7 Park        C     4
8  Lee        C     8
9  Kim        C    10
```

옵션
Variable.name과 value.name 활용

```
  name item amount
1 Park    A     14
2  Lee    A     21
3  Kim    A     15
4 Park    B     12
5  Lee    B     15
6  Kim    B      5
7 Park    C      4
8  Lee    C      8
9  Kim    C     10
```

```
> melt(df2,id.vars="name",variable.name="item",value.name="amount")
  name item amount
1 Park    A     14
2  Lee    A     21
3  Kim    A     15
4 Park    B     12
5  Lee    B     15
6  Kim    B      5
7 Park    C      4
8  Lee    C      8
9  Kim    C     10
```

앞으로의 진행순서

1. R프로그래밍 활용을 통한 EDA 기초 정리 및 마무리
   - EDA1 & 2: 탐색적 접근 방법
   - 회귀분석 기초: (R강의 마무리)

2. Python 다루기 및 성능 파악
   - 기본적 인덱싱 및 데이터 자료구조
   - For와 If/ While에 대한 개념 잡기 및 연습문제

3. ML과 DL기본적 개념잡기
   - Python 기본적 이해도에 따른 1)텍스트 마이닝/ 2) 이미지 인식코드 진도
   - Python 으로 해당 파일을 변환시켜보기

주요 이력
現) ㈜RTMC 전략기획실장
前) ㈜B사 웹로그분석 및 DP사업 完
前) ㈜H금속사 회계팀
前) ㈜B건설사 회계팀
前) K문고 CRM VIP 군집전략 CRM프로젝트 보조연구원
前) L백화점 CRM Alert 전략 CRM프로젝트 보조연구원

BSL(스위스 로잔 비즈니스 스쿨) MBA
ASSIST 빅데이터경영통계 MBA

국가공인 ADSP(빅데이터 준전문가)
現 **코리아IT아카데미 빅데이터 R 강사**
現 **코리아IT아카데미 빅데이터 기초 파이썬 강사**
現 **코리아IT아카데미 빅데이터 기초통계 전담강사**

"자료는 대가이신 박동련 교수님께 도움을 받았음을 밝힙니다."

[박영식] 완성에 이르기까지