4장 데이터 조작

(1) 내부데이터

- **data(데이터 셋 이름)** - 데이터 불러오기

(2) 파일 입출력 --- 외부데이터

- CSV파일 : ,로 구분된 형식을 갖는 파일

csv파일 입출력

출력

read.csv(file, header=TRUE(디폴트), na.strings=c("NA"),
stringsAsFactors=default.stringsAsFactors())

- **na.strings = " NA "** / " NA " 데이터를 NA처리할 때 사용 ~ **벡터 형식으로** 전달
- **header = TRUE(기본값)** / 맨 처음 행을 행이름으로 처리
  cf. read.table은 FALSE가 기본값
- **stringsAsFactors = FALSE** / 평소 하는 습관 들이고 나중에 해당 칼럼만 factor로 수정하기!

cf. 공백 이외에 구분자가 있다면 명시하는 옵션 : **sep = "구분자표시"**

입력

write.csv(x,file="",row.names=TRUE)

- **row.names=TRUE (기본값)** / TRUE면 행 이름을 csv파일에 포함하여 저장한다.

```
> x <- read.csv(file="emp.csv")
> str(x)
'data.frame':   14 obs. of  8 variables:
 $ EMPNO   : int  7369 7499 7521 7566 7654 7698 7782 7788 7839 7844 ...
 $ ENAME   : Factor w/ 14 levels "ADAMS","ALLEN",..: 12 2 14 7 9 3 4 11 8
13 ...
 $ JOB     : Factor w/ 5 levels "ANALYST","CLERK",..: 2 5 5 3 5 3 3 1 4
5 ...
 $ MGR     : int  7902 7698 7698 7839 7698 7839 7839 7566 NA 7698 ...
 $ HIREDATE: Factor w/ 13 levels "1980/12/17 00:00:00",..: 1 2 3 4 8 5 6 12
9 7 ...
 $ SAL     : int  800 1600 1250 2975 1250 2850 2450 3000 5000 1500 ...
 $ COMM    : int  NA 300 500 NA 1400 NA NA NA NA 0 ...
 $ DEPTNO  : int  20 30 30 20 30 30 10 20 10 30 ...

> x2 <- read.csv(file="emp.csv", stringsAsFactors = FALSE)
> str(x2)
'data.frame':   14 obs. of  8 variables:
 $ EMPNO   : int  7369 7499 7521 7566 7654 7698 7782 7788 7839 7844 ...
 $ ENAME   : chr  "SMITH" "ALLEN" "WARD" "JONES" ...
 $ JOB     : chr  "CLERK" "SALESMAN" "SALESMAN" "MANAGER" ...
 $ MGR     : int  7902 7698 7698 7839 7698 7839 7839 7566 NA 7698 ...
```

```
 $ HIREDATE: chr  "1980/12/17 00:00:00" "1981/02/20 00:00:00" "1981/02/22
00:00:00" "1981/04/02 00:00:00" ...
 $ SAL     : int  800 1600 1250 2975 1250 2850 2450 3000 5000 1500 ...
 $ COMM    : int  NA 300 500 NA 1400 NA NA NA NA 0 ...
 $ DEPTNO  : int  20 30 30 20 30 30 10 20 10 30 ...
```

hiredate -- 날짜 아님! -> **as.Date로 해야 함!**

```
> x2$HIREDATE <- as.Date(x2$HIREDATE,format="%Y/%m/%d")
> str(x2)
'data.frame':  14 obs. of  8 variables:
 $ EMPNO   : int  7369 7499 7521 7566 7654 7698 7782 7788 7839 7844 ...
 $ ENAME   : chr  "SMITH" "ALLEN" "WARD" "JONES" ...
 $ JOB     : chr  "CLERK" "SALESMAN" "SALESMAN" "MANAGER" ...
 $ MGR     : int  7902 7698 7698 7839 7698 7839 7839 7566 NA 7698 ...
 $ HIREDATE: Date, format: "1980-12-17" "1981-02-20" ...
 $ SAL     : int  800 1600 1250 2975 1250 2850 2450 3000 5000 1500 ...
 $ COMM    : int  NA 300 500 NA 1400 NA NA NA NA 0 ...
 $ DEPTNO  : int  20 30 30 20 30 30 10 20 10 30 ...
```

```
> x<-read.csv("c.csv", na.strings=c("NIL"))
```

★★ **na.strings -> 벡터형식으로 전달**

★★ **"NIL"이라는 데이터 NA로 바꾸기**

~p134

프린트 - 텍스트 파일 읽어서 벡터에 저장하기

scan 함수

텍스트 파일은 scan()으로 불러들여 벡터 형태로 저장 가능

/ 문자는 what="" 옵션 표기

scan과 read.table의 차이

return되는 객체

scan -> 벡터형 (나열만), 정수형. txt분석시 정형화되지 않은 ~0~

read.table -> 데이터프레임형

setwd("c:₩₩r_temp") <-- 데이터가 있는 작업용 디렉터리를 설정

```
> v1 <- scan("t1.txt")
Read 4 items
> v1
[1] 111 222 333 444   <-- 기본적으로 벡터로 저장됨
> class(v1)
[1] "numeric"

> scan2 <- scan('t1.txt',what="") <-- 문자나 실수인 경우 what="" 명시 주의!
Read 4 items
> scan2
[1] "111" "222" "333" "444"
> class(scan2)
[1] "character"

> scan3 <- scan("t.txt", what="") <-- 문자나 실수인 경우 what="" 명시 주의!
Read 4 items
> scan3
[1] "aaa"  "bbb"  "111"  "2.34"
> class(scan3)
[1] "character"

> input <- scan() #숫자 입력받기
1: 1
2: 2
3: 3
4:                  #그만하려면 엔터치기
Read 3 items
> input
[1] 1 2 3

> input<-scan(what="")  #문자 입력시 what="" 꼭 쓰기
1: a
2: b
3: c
4: 2.34
5:                   #그만하려면 엔터치기
Read 4 items
> input
[1] "a"    "b"    "c"    "2.34"
```

readline()

- 함수로 한 줄 읽어 들이기
- 리드라인을 통해 불러들인 값 -> 변수 통해 저장하기

```
> input3 <- readline()    #입력 전 까지 대기!
R is very fun!
>   input3
[1] "R is very fun!"

> input4 <- readline("Are you OK ? :")
Are you OK ? : Yes!!
> input4
[1] "Yes!!"
```

cf프로그램 작성시 사용자가 입력과 동시에 다음 프로그래밍에 영향을 끼치고 싶을 때

```
> ans <- readline("input value (y or n) :")
input value (y or n) :

> if (ans == "y") {
+   d <- c(1)
+ } else if (ans=="n"){
+   d<-c(0)
+ } else{
+   reture("invalid input value")
+ }
```

readLines()

- 텍스트 파일을 **한 줄씩** 읽어 문자열 벡터로 생성 (cf. 원소별 나열) - 주로 텍스트 분석할 때 --> 빈도처리시 활용, 비정형화된 데이터를 불러올 때

```
> input5 <- readLines('scan_4.txt')
> input5
[1] "aaa" "bbb" "111" "2.34"

> data1 <- readLines("seoul_new.txt")
> data1
[1] "305 무료법률상담에 대한 부탁의 말씀 입니다. 2014-09-27 2 "
[2] "304 [교통불편접수] 6715 버스(신월동->상암동) 2014-09-26 2"
```

★★★ read.table()

- 파일을 읽어들여 데이터 프레임 구조로 생성
- read.csv랑 다르게 첫 번째 행을 헤더로 읽어들이지 않음**, header=TRUE** 입력해야 함
- 주석이나 공백이 있을 경우 자동으로 제거함. -> csv파일- **sept = ","** 옵션을 통해 구분자 입력
- 지원 가능한 파일 형식이 read.csv보다 확장됨

```
> fruit2 <- read.table('fruits_2.txt') <-- 주석은 자동 제외됨
> fruit2
  V1      V2   V3   V4
1  1   apple 500   6
2  2 banana 200   2
3  3   peach 200   7
4  4   berry 50   9

> fruit2 <- read.table('fruits_2.txt', skip=2) <--2행을 skip한다는 뜻 / 유의 :행 count시 주석 포함
> fruit2
  V1      V2   V3   V4
1  1   apple 500   6
2  2 banana 200   2
3  3   peach 200   7
4  4   berry 50   9

> fruit2 <- read.table('fruits_2.txt', nrows=2)   <--입력할 줄 수 지정. (for 데이터 형식 일부만 확인)
> fruit2
  V1      V2   V3   V4
1  1   apple 500   6
2  2 banana 200   2
```

t2 텍스트파일 생성

name,deptno,sal

smith,10,900

mike,20,1200

allen,40,2330

```
> f3 <- read.table(file="t2.txt", header=T, sep=",", stringsAsFactors = F
)
> str(f3)
'data.frame':   3 obs. of  3 variables:
 $ name  : chr  "smith" "mike" "allen"
 $ deptno: int  10 20 40
 $ sal   : int  900 1200 2330
```

-데이터가 많은 경우 skip, nrows 사용

cf 텍스트파일에서 마지막 줄 엔터 안 하면 나타남

```
Warning message:
In read.table(file = "t2.txt", header = T, sep = ",") :
  't2.txt'에서 readTableHeader에 의하여 발견된 완성되지 않은 마지막 라인입니다
```

write.table()

외부 텍스트 파일에 저장하기

WriteXLS()

- 패키지 설치 필요

```
install.packages("WriteXLS")
library(WriteXLS)
name <- c("Apple","Banana","Peach")
price <- c(300,200,100)
item <- data.frame(NAME = name, PRICE = price)
WriteXLS("item","item.xls"
```

readLines() / read.table() / read.csv()의 차이점

csv_test.txt

번호,이름,가격

1,사과,1000

2,배,2000

3,귤,3000

```
> (txt1<- read.csv("csv_test.txt"))
  번호 이름 가격
1    1 사과 1000
2    2   배 2000
3    3   귤 3000

> (txt2<-readLines("csv_test.txt"))   # 파일의 한 줄 씩 -> 벡터형식으로 불러옴
[1] "번호,이름,가격" "1,사과,1000"    "2,배,2000"      "3,귤,3000"


> (txt3<-read.table("csv_test.txt",sep=","))
    V1   V2   V3
1 번호 이름 가격
2    1 사과 1000
3    2   배 2000
4    3   귤 3000
> (txt3<-read.table("csv_test.txt",sep=",",header=T))
  번호 이름 가격
1    1 사과 1000
2    2   배 2000
3    3   귤 3000
```

저장하는 다양한 방법 정리하기

[readLines() -> write]

```
> txt1 <- readLines("write_test.txt")
> txt1
[1] "이 문장은 write 연습하는 문장인데" "별로 어렵지 않아요"
[3] "열심히 해주세요"

> write(txt1,"write_test2.txt")   #write - 벡터형태
> txt2<-readLines("write_test2.txt")
> txt2
[1] "이 문장은 write 연습하는 문장인데" "별로 어렵지 않아요"
[3] "열심히 해주세요"
```

----------write() 함수는 벡터 형태만 저장 가능

[read.table() -> write.table()]

[read.csv() -> write.csv()]

----------프린트 끝

04 apply계열 함수

함수	설명	다른 함수와 비교시 특징
apply()	배열/ 행렬/ 데이터프레임에 주어진 함수를 적용한 뒤, 그 결과를 벡터 / 배열 / 리스트로 반환 (행/열 별)	배열/행렬/ 데이터프레임 적용
lapply()	벡터, 리스트 또는 표현식에 함수를 적용하여 그 결과를 리스트로 반환	결과가 리스트
sapply()	lapply와 유사하지만 결과를 벡터/행렬/배열로 반환	결과가 벡터/행렬/배열
tapply()	벡터에 있는 데이터를 특정 기준에 따라 그룹으로 묶은 뒤, 각 그룹마다 주어진 함수를 적용하고 그 결과를 반환	데이터 그룹화 후 함수 적용
mapply()	sapply의 확장된 버전 - 여러 벡터/리스트를 인자로 받아 함수에 각 데이터의 첫째, 둘째, 셋째 ... 요소들을 적용한 결과들을 반환	여러 데이터를 함수 인자로 적용

- 원소별/벡터별로 함수를 적용시키는 함수!
- 어떤 인자를 불러와야 하는지 위주로 보자

cf. if는 벡터 연산 X.

apply()

apply(X, MARGIN, FUN)

X : 행렬 또는 데이터프레임
MARGIN : 행 방향 :1 / 열 방향 : 2 / 행+열, 즉 원소별 : c(1,2)
FUN : 적용할 함수
- **행렬, 데이터프레임**만 원소별 함수 전달 가능

rowSums(x, na.rm=TRUE)
rowMeans(x, na.rm=TRUE)
colSums(x, na.rm=TRUE)
colMeans(x, na.rm=TRUE)
na.rm = TRUE : NA를 제외할지 여부

```
> apply (m1, c(1,2), max)
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
 # c(1,2) : 원소별 최대값만 반환

> apply (m1, 1,max)
[1] 7 8 9
 # 같은 행 내 최대값만 반환
> apply (m1, 2,max)
[1] 3 6 9
 # 같은 열 중 최대값만 반환


> apply(iris[,1:4],2,sum)
Sepal.Length  Sepal.Width Petal.Length  Petal.Width
       876.5       458.6       563.7       179.9
```

```
# 팩토리얼 함수를 행/열에 적용시키려 함
> myf7 <- function(m){
+   if (m==1) {
+     return(1)
+   }
+   else {
+     return(m*myf7(m-1))
+   }
+ }

> apply(m1,1,myf7)
In addition: Warning messages:
1: In if (m == 1) { :
  the condition has length > 1 and only the first element will be used
2: In if (m == 1) { :
  the condition has length > 1 and only the first element will be used
```

!!!!!!!!!! 뭐가 되고 뭐가 안 되는지 알아야 함!!!!!!!!!!!!!!!!

안되는 이유 : **인자의 개수(**=벡터형태) <> 팩토리얼 함수 인수 : 1개

다시!! 일단은 전체에 적용시켜보기

```
> apply(m1,c(1,2),myf7)
     [,1] [,2]   [,3]
[1,]    1   24   5040
[2,]    2  120  40320
[3,]    6  720 362880
```

cf. Python 원소별 절대 불가능. 무조건 행별 / 열별

cf. 대문자로 표시 -> c(1,2), toupper 사용

UPPER는 행별만 / 열별만 다름. 함수에 따라 인자 맞게끔 적용 방향을 잘 설정해줘야 함.

data프레임구조 형태로 출력은 불가능

다시 점검

emp 테이블 중 deptno에 팩토리얼 apply 함수 통해서 대입

1. 전체가 아닌 선택할 목적. (deptno 열별)
2. 그런데 f2$DEPTNO, **벡터 형식은 전달 불가**
   (행/열 없어서 전달 안 됨)
3. 다른 방법 f2[,c(8,8)] => 벡터가 아니고 데이터 프레임 형태. 따라서 이 때에는 원소별 가능함.

```
> apply(f1[,c(8,8)],c(1,2),myf7) [,1]
```

이건 내 방법

```
> apply(f1[,8,drop=FALSE],c(1,2),myf7)
```

이건 매트릭스 형식으로 바꿔 풀기

```
> apply(as.matrix(f1$DEPTNO),c(1,2),myf7)
```

원소별 적용하는 함수는 다른 함수를 이용할 것을 권장함. c(1,2) 하긴 하지만!

lapply() 함수

- 행, 열별 연산 방향 선택 옵션 없음. (MARGIN -- **X**)
- **벡터, 행렬은 원소별** 함수 전달
- **리스트와 데이터 프레임은** 각 **key별**로 함수 전달, 키의 원소별 전달 불가
- 리스트로 반환 -> 서로 다른 데이터 타입을 가지고 있는 경우 추천

```
> lapply(f1$DEPTNO,myf7)
[[1]]
[1] 2.432902e+18

[[2]]
[1] 2.652529e+32

[[3]]
[1] 2.652529e+32

[[4]]
[1] 2.432902e+18

...
```

```
> l1 <- list (a=c(1,2,3),b=c(4,5,6))

> lapply(l1,myf7) # list 형태 : 개별 연산 불가능
```

(리스트, 데이터프레임은 lapply에서 개별 연산 불가능, key별만 가능)

```
> lapply(l1,sum) #sum 은 인자가 제한이 없는 함수. key 별로 계산됨
```

-----lapply()의 결과를 벡터 또는 데이터 프레임으로 변활할 때 사용하는 함수----

unlist()

unlist(x, recursive=TRUE use.names=TRUE) # 옵션 잘 사용 안 함

- 리스트 구조를 다시 벡터로 변환

recursive = TRUE**(기본값)** : 리스트 속에 리스트가 있을 경우 (하위리스트) 벡터로 변환

use.names = TRUE**(기본값)** : 리스트의 키값 보존 여부

do.call()

do.call(what , args)

what : 함수

args : 전달할 인자의 **리스트(키별)**

- 함수에 인수를 리스트 형식으로 전달 시 사용 (lapply랑 비슷)
- 속도가 느림. => 이후 배울 rbindlist() 사용

```
> (result<-lapply(1:3,function(x){x*2}))
[[1]]
[1] 2

[[2]]
[1] 4

[[3]]
[1] 6

> result[[1]]
[1] 2
> unlist(result)
[1] 2 4 6
```

색인 관련 문제

```
> l1<-list(c(1,2,3),c(4,5,6))
> l1[1]
[[1]]
[1] 1 2 3
```

첫 번째 층의 두 번째 원소 값은?

```
> l1[[1]][2]
[1] 2
```

데이터프레임, 행렬 - 색인시 콤마 잊지 말기!!!!!!!!!!!!!

 # lapply()를 통해 데이터 프레임을 리스트로 얻고 다시 데이터 프레임 구조로 변경하려면

1. unlist()
2. matrix() 벡터->행렬
3. as.data.frame 행렬 -> 데이터프레임
4. names() 리스트로부터 변수명 얻어와 각 컬럼에 이름 부여

(1번 2번 생략 가능,
(1번 2번 위험-> 리스트 : key별로 서로 다른 데이터 타입을 가질 수 있음
벡터 : 모든 데이터 타입이 같아야 함)

따라서 모든 데이터 타입이 같은 경우에 사용 하기!

-> 데이터 타입 가장 먼저 확인하는 습관 가지기

```
> data.frame(do.call(cbind,lapply(iris[,1:4],mean)))
```

sapply()

sapply(X,FUN,...)

- lapply()와 유사

- **행렬, 벡터** 등의 데이터 타입으로 결과를 **반환** -> 반환 값에 여러 데이터 타입이 섞이면 안 됨

```
> lapply(iris[,1:4],mean)
$`Sepal.Length`
[1] 5.843333
$Sepal.Width
[1] 3.057333
$Petal.Length
[1] 3.758
$Petal.Width
[1] 1.199333

> sapply(iris[,1:4],mean)
Sepal.Length  Sepal.Width Petal.Length  Petal.Width
    5.843333     3.057333     3.758000     1.199333
```

#sapply(0에서 반환한 벡터는 as.data.frame()을 통해 데이터프레임으로 변환 가능.

#t(x)를 사용하지 않으면 기대한 것과 다른 모양의 데이터 프레임을 얻게 된다.

```
> x <- sapply(iris[,1:4],mean)
> as.data.frame(x)
                    x
Sepal.Length 5.843333
Sepal.Width  3.057333
Petal.Length 3.758000
Petal.Width  1.199333
> as.data.frame(t(x))
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1     5.843333    3.057333        3.758    1.199333
```

 # 각 컬럼의 데이터 타입을 구하는 예

```
sapply(iris, class)
Sepal.Length  Sepal.Width Petal.Length  Petal.Width     Species
   "numeric"    "numeric"    "numeric"    "numeric"    "factor"
```

 # sapply()의 함수 출력이 벡터로 된다면, sapply는 행렬을 반환한다.

```
> y<-sapply(iris[,1:4],function(x){x>3})
> class(y)
[1] "matrix"
> head(y)
     Sepal.Length Sepal.Width Petal.Length Petal.Width
[1,]         TRUE        TRUE        FALSE       FALSE
[2,]         TRUE       FALSE        FALSE       FALSE
[3,]         TRUE        TRUE        FALSE       FALSE
[4,]         TRUE        TRUE        FALSE       FALSE
[5,]         TRUE        TRUE        FALSE       FALSE
[6,]         TRUE        TRUE        FALSE       FALSE
```

sapply() 참고

- sapply()는 한 가지 타입만 저장 가능한 데이터 타입인 벡터/행렬을 반환
- 따라서 함수의 반환 값에 여러 데이터 타입이 섞여 있으면 안 된다.
- 그럴 경우 lapply() / plyr패키지를 사용해야 함 (5장)

tapply()

tapply(x, index, fun, ...)

- **그룹별** 함수를 적용하는 함수
- **x** : **벡터로 만들어서 전환한다.** (행렬이나 데이터프레임인 경우 쭉 펼쳐서 매칭시킴)
- **index** : 벡터화된 x의 인자들을 순서대로 매칭시킬 그룹에 대한 정보

 # rep()의 결과값을 그룹 색인 값으로 사용

```
> tapply(1:10, rep(1,10),sum)
 1
55

> tapply(1:10, rep(c(1,2),each=5),sum)
 1  2
15 40
```

 # 불리언 색인 값을 그룹 색인 값으로 사용

```
> tapply(1:10, 1:10 %% 2 == 1, sum)
FALSE  TRUE
   30    25
```

★★★ **# 데이터 프레임의 컬럼값을 그룹 색인 값으로 사용--------- 자주 쓰임!** ★★★

index의 distinct value끼리 묶어서 앞 데이터를 뒤의 함수 적용

```
> tapply(iris$Sepal.Length, iris$Species, mean)
    setosa versicolor  virginica
     5.006      5.936      6.588
```

----> 특정 값을 가지고 있는 걸 바로 그룹화하는 값으로 사용!

emp2 테이블의 그룹별 페이 평균은?

```
> tapply(emp2$PAY, emp2$EMP_TYPE, mean)
  계약직   수습직   인턴직   정규직
0000000 22000000 20000000 61600000
```

연습

행+열으로 되어있는 경우 / 12 34 56 78 따로따로 묶고 싶다

```
> m2 <- matrix(1:8, ncol=2)
> m2
     [,1] [,2]
[1,]    1    5
[2,]    2    6
[3,]    3    7
[4,]    4    8
                        #for 그룹화 벡터
```

```
> tapply(m2,list(c(1,1,2,2,1,1,2,2),c(1,1,1,1,2,2,2,2)),sum)
  1  2
1 3 11
2 7 15
```

```
> tapply(m2[,1],c(1,1,2,2),sum)
1 2 # 그룹 이름
3 7 # 결과값
```

cf.

```
tapply(m2,c(1,1,2,2),sum)
```

m2 : 매트릭스, **but** tapply : **벡터만 전달**

-> m2 매트릭스가 하나로 합쳐져 **총 8개 인자의 벡터로** 전달됨

```
> tapply(m2,c(1,1,2,2),sum)
Error in tapply(m2, c(1, 1, 2, 2), sum) :
  인자들은 반드시 같은 길이를 가져야 합니다

> tapply(m2,c(1,1,1,1,2,2,2,2),sum)
 1  2
10 26
# 원래 형식처럼 같은 컬럼으로 묶었다
```

mapply()

mapply(FUN, ...,)

- 다수의 인자를 함수에 넘기는 경우
- 행렬/ 데이터프레임은 컬럼별로 묶어 전달

```
> myf1 <- function(x,y,z){
+   return(x+y+z)
+ }

> mapply(myf1,0,10,2)
[1] 12


> mapply(myf1,c(1,10),c(2,20),c(3,30))
[1]  6 60


> mapply(myf1,
+        data2$승차, #벡터형식이므로 각 원소를 myf1의 첫 번째 원소로 전달(x값)
+        data2$하차, #벡터형식이므로 각 원소를 myf1의 첫 번째 원소로 전달(y값)
+        0)          #벡터형식이므로 각 원소를 myf1의 첫 번째 원소로 전달(z값)

 [1]  123530  309656  742444 1550595 1107501  926731  982442 1049056 1183423
1203369
[11] 1211061 1256185 1329747 1659163 1347457  940711  903687  795902  404601
70519
[21]  748914 2027026 5284464 9598808 6101079 4038065 3805067 4039584
4514359 4540402
[31] 4765922 5203662 6212610 8731655 7360363 5216365 4902910 4820557
2768078  697156
[41]  182310  711896 1868452 3266477 2241562 1582417 1511632 1576143
1725447 1739337
[51] 1823129 1960685 2222467 2877946 2380598 1676469 1485100 1331381
709017  167141
[61]  283398  904491 2288950 3540529 2480898 1842351 1785801 1891861
2055130 2100246
[71] 2182016 2322556 2677994 3296133 2898330 2157034 1946397 1834121
1115222  269667
```

매트릭스, 데이터프레임이라면 각 컬럼을 묶어서 전달 -> 컬럼별 연산

```
> mapply(mean,iris[,1:4])
Sepal.Length  Sepal.Width Petal.Length  Petal.Width
    5.843333     3.057333     3.758000     1.199333
```

cf. 난수생성함수

rnorm(n, mean, sd)

/ 평균이 mean, 표준편차가 sd인 정규 분포를 따르는 난수 n개 발생

runinf(n, min, max)

/ 최솟값이 min, 최댓값이 max인 균등 분포를 따르는 난수 n개 발생

rpois(n,lambada)

/ 람다 값이 lambda인 포아송 분포를 따르는 난수 n개 발생

rexp(n,rate=1)

/ 람다가 rate인 지수 분포를 따르는 난수 n개 발생

6 함수 심화문제

1. data1.csv 파일을 불러와서

data1<- read.csv("data1.csv", stringsAsFactors = F)

cf. 숫자로 변수 이름 시작 X -> X가 붙음

1) 년도별 합계

apply(data1[,-1],2,sum)

2) 연령별 합계

apply(data1[,2:15],1,sum)

#이름 가지고 싶으면 names~ / 로 이름 별로 전달

rownames(data1) <- data1[,1]

2. data2.csv 파일을 불러와서

data2 <- read.csv("data2.csv",stringsAsFactors = F)

1) 노선 번호 별 승차인원수 합계

tapply(data2[,3],data2[,1],sum)

2) 노선 번호 별 하차인원수 합계

tapply(data2[,4],data2[,1],sum)

3) 노선 번호 상관없이 승차 / 하차 인원수 합계

apply(data2[,c("승차","하차")],2,sum)

sapply(data2[,c("승차","하차")],sum)

4) 노선 번호별 승차+하차 총 인원수

tapply(data2$하차 + data2$승차, data2$노선번호, sum)

여기 데이터 형태에서부터 연산 가능!

나 ㅠ

tapply(c(data2[,3],data2[,4]),rep(data2$노선번호,2),sum)

apply계열 함수 공통 특징 : 출력되는 결과 = 벡터/리스트/행렬 (not 데이터프레임)
--> apply()적용 후 데이터프레임은 후에 만들기.
--> for문 없어도 각 원소별 함수 적용 가능

05 데이터를 그룹으로 묶은 후 함수 호출하기 p149-157

- 데이터를 그룹별로 나눈 후 그룹별로 함수를 호출하는 경우

doBy 패키지함수

install.packages("doBy")

library(doBy)

연산자	예	의미
+	Y1 ~	
-		
|		
:		
*		

summaryBy() - 컬럼별로 요약통계

- 장점 : formula에 의해 확장이 자유로움
- 단점 : 평균만 표시됨

```
> summaryBy(Sepal.Width + Sepal.Length ~ Species, iris)
     Species Sepal.Width.mean Sepal.Length.mean
1     setosa           3.428            5.006
2 versicolor           2.770            5.936
3  virginica           2.974            6.588
```

formula

A + B ~ C : A와 B를 C별로 요약

A + B ~ C + D D별로 2차분류 하고 싶을 경우

cf. 기본패키지 **summary() / quantile()**

```
> summary(iris)
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width          Species
 Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100   setosa    :50
 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
versicolor:50
 Median :5.800   Median :3.000   Median :4.350   Median :1.300
virginica :50
 Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
 3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
 Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500

> quantile(iris$Sepal.Length)
  0%  25%  50%  75% 100%
 4.3  5.1  5.8  6.4  7.9
> quantile(iris$Sepal.Length,seq(0,1,by=0.1))
  0%  10%  20%  30%  40%  50%  60%  70%  80%  90% 100%
4.30 4.80 5.00 5.27 5.60 5.80 6.10 6.30 6.52 6.90 7.90
```

orderBy() - 정렬

- 행 번호를 리턴, not 값 리턴 => 그 후 색인을 해야 함
- 1차정렬, 2차정렬, 3차정렬, ... 확장 편함

cf. sort -> 데이터를 정렬한 값 리턴/ 1차정렬만.

정렬할 컬럼을 작성

```
orderBy(~ Sepal.Width, iris)
    Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
61           5.0         2.0          3.5         1.0 versicolor
63           6.0         2.2          4.0         1.0 versicolor
69           6.2         2.2          4.5         1.5 versicolor
120          6.0         2.2          5.0         1.5  virginica
42           4.5         2.3          1.3         0.3     setosa
54           5.5         2.3          4.0         1.3 versicolor
...
```

```
> orderBy(~ Species + Sepal.Width, iris)
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
42          4.5         2.3          1.3         0.3  setosa
9           4.4         2.9          1.4         0.2  setosa
2           4.9         3.0          1.4         0.2  setosa
13          4.8         3.0          1.4         0.1  setosa
14          4.3         3.0          1.1         0.1  setosa
26          5.0         3.0          1.6         0.2  setosa
```

cf. 기본 패키지에 있는 order()

order(..., na.last=TRUE, decreasing=FALSE)

na.last - NA를 마지막에 배치할 것인지

decreasing : 내림차순 여부

=> 값의 **행 순서를 return**

```
> order(iris$Sepal.Width)
  [1]  61  63  69 120   ...    #행번호에 따른 순서 리턴
```

=> 데이터 프레임의 인덱스 사용으로 사용해서 정렬된 결과 나타낼 수 있음 (정수색인)

```
> iris[order(iris$Sepal.Width),]
```

///여러 컬럼을 기준으로 데이터를 정렬하고자 할 때! 해당 컬럼들을 order()에 나열한다.

다음은 Sepal.legnth, Sepal.Width 순으로 정렬한 예

```
> iris[order(iris$Sepal.Length,iris$Sepal.Width),]
```

20180516

sampleBy() - 샘플링

sampleBy(formula, frac=0.1, replace=FALSE, data=parent.frame(), systematic=FALSE)

- 데이터를 그룹으로 묶은 후 각 그룹에서 샘플을 추출
- 임의한 샘플링을 하면서 동일한 모델에 대한 재학습시킴 / 머신러닝... 훈련용데이터~샘플~
- sample()함수와 비슷하지만 그룹을 포물려로 전달 가능
- 비복원추출이 디폴트 (replace=FALSE)
- **sampleBy()는 랜덤샘플링 시 각 class별 균일한 샘플링을 가능하게 함**

~ 우측에 나열한 이름에 따라 데이터가 그룹으로 묶인다

frac : 추출할 샘플의 비율

data= 뒷쪽에, 따라서 data=와 함께 해서 입력하는 것이 좋겠다

sampleBy()를 사용하여 각 class별 균등 샘플링 수행

cf. 기본패키지 sample()

sample(x,size,replace=FALSE,prob)

x - 샘플을 뽑을 데이터 벡터. 만약 길이 1인 숫자 n인 경우 1:n에서 샘플이 설정된다.

size - 샘플의 크기

prob - 데이터가 뽑힐 가중치

/ 랜덤한 재배치에도 사용함.

x 인자 개수랑 size 개수 맞춰서 sample() 사용한 함수를 row색인으로 사용하면 랜덤 나열 가능

```
> sample(10,10)
 [1]  5  7  6  4  8  1  3  2  9 10
> sample(10,5)
[1] 3 5 8 6 1
> sample(1:10,5,replace=T)
[1] 9 8 8 5 4
```

```
> iris[sample(1:150,150),]
Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
25          4.8         3.4          1.9         0.2     setosa
150         5.9         3.0          5.1         1.8  virginica
60          5.2         2.7          3.9         1.4 versicolor
47          5.1         3.8          1.6         0.2     setosa
123         7.7         2.8          6.7         2.0  virginica
7           4.6         3.4          1.4         0.3     setosa
...
```

```
> iris[sample(NROW(iris),NROW(iris)),]
    Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
125          6.7         3.3          5.7         2.1  virginica
9            4.4         2.9          1.4         0.2     setosa
100          5.7         2.8          4.1         1.3 versicolor
23           4.6         3.6          1.0         0.2     setosa
130          7.2         3.0          5.8         1.6  virginica
141          6.7         3.1          5.6         2.4  virginica
136          7.7         3.0          6.1         2.3  virginica
103          7.1         3.0          5.9         2.1  virginica
51           7.0         3.2          4.7         1.4 versicolor
115          5.8         2.8          5.1         2.4  virginica
```

but 특정 class별로 치우칠 수도 있따 -> sampleBy()사용

180516 유인물

문자열 관련 함수

- stringr() 패키지 : 문자열 관련 패키지

- 대개 벡터연산 가능 -> 원소별로 살펴볼 수 있음

(1) str_dect() 함수

- 특정 문자 포함 여부 확인
- 활용해서 row index(불리언 색인) -> 해당 행만 추출

```
> v1<-c('abc','bbc','cda')

> str_detect(v1,'^[abc]')   # 첫번째 글자는 a or b or c
[1] TRUE TRUE TRUE
> str_detect(v1,'^[bc][bc]')  #첫 번째 글자는 b or c + 두 번째 글자는 b or c
[1] FALSE  TRUE FALSE
> str_detect(v1,'^[a-z][bc]') #첫 번째 글자는 영문소문자 + 두 번째 글자는 b or c
[1]  TRUE  TRUE FALSE
> str_detect(v1,'^[a-zA-Z][bc]') #첫 번째 글자는 영문 + 두 번째 글자는 b or c
[1]  TRUE  TRUE FALSE

> str_detect(v1,'[bc]$') #마지막 글자가 b or c
[1]  TRUE  TRUE FALSE
```

(2) ignore.case() 함수 - 대소문자 무시하기

---- not in stringr()패키지

```
> str_detect(fruits,ignore.case('a'))
```

(3) str_count()함수 - 특정 문자 출현 횟수 세기

```
> str_count(fruits,'a')
[1] 1 0 3 1
```

(4) str_c() 함수 - 문자열을 합치기 - cf. concate 연결연산자 in oracle

텍스트 + 벡터 결합 -> 텍스트 + 벡터 인자 별 결합시킴

```
> str_c("apple","banana","peach")
[1] "applebananapeach"

> str_c(fruits," name is ",fruits)
[1] "apple name is apple"        "Apple name is Apple"
[3] "banana name is banana"      "pineapple name is pineapple"

> str_c(fruits, collapse="")
[1] "appleApplebananapineapple"

> str_c(fruits, collapse=" ")      #구분 옵션 collapse=
[1] "apple Apple banana pineapple"
```

(5) str_dup()함수 - 반복 출력하기 **cf.** rep()

```
> str_dup(fruits,3)
[1] "appleappleapple"            "AppleAppleApple"
[3] "bananabananabanana"         "pineapplepineapplepineapple"
```

(6) str_length() 함수 - 문자열의 길이 출력하기 / 원소별 길이

- db에서 처리하는 게 더 편하고 범위가 넓음
- 같은 문자를 연속해서 찾는 경우 시작/끝 위치가 다르게 나타날 수 있음

```
> str_length('apple')
[1] 5
> str_length(fruits)
[1] 5 5 6 9
```

cf. length() : 벡터 개수 출력

```
> length(fruits)
[1] 4
```

(7) str_locate() 함수

- 특정 문자의 위치 값 찾기 / **cf.** in orcle - instr (instr보다 제한적)
- str_locate_all() 모두 찾기

```
> str_locate('apple','a')
     start end
[1,]     1   1


> str_locate(fruits,'a')
     start end
[1,]     1   1
[2,]    NA  NA        # 없는 경우 NA 값 출력
[3,]     2   2
[4,]     5   5


> str_locate(fruits,'pp')
     start end
[1,]     2   3
[2,]     2   3
[3,]    NA  NA
[4,]     6   7
```

```
> str_locate_all(fruits,"a")  # 모든 위치를 출력
[[1]]
     start end
[1,]     1   1

[[2]]
     start end

[[3]]
     start end
[1,]     2   2          # 첫 번째 a의 위치
[2,]     4   4          # 두 번째 a의 위치
[3,]     6   6          # 세 번째 a의 위치


[[4]]
     start end
[1,]     5   5
```

(8) str_replace() 함수

- str_replace_all() : 모두 치환하기

```
> str_replace('apple','p','*') # 처음 찾은 것만 한 번 치환 함
[1] "a*ple"
> str_replace('apple','p','**')
[1] "a**ple"

> str_replace_all('apple','p','*') # 전체, 여러 번 치환함
[1] "a**le"
```

(9) str_split() 함수

- ex. 분리해서 첫 번째 것만 추출할 때

```
> fruits<-str_c('apple','/','orange','/','banana')
> fruits
[1] "apple/orange/banana"

> str_split(fruits,"/") # / 기호를 기준으로 분리해서 리스트 벡터로 추출함
                        n 번째 찾기 -> 색인방식 잘 생각해야 함
[[1]]
[1] "apple"  "orange" "banana"
```

(10) str_sub() 함수 - cf in orcle substr과의 차이점 / end : 개수 X, 위치 O

- end생략시 끝까지 출력됨
- -로 시작해도 출력되는 값은 정방향으로 출력됨

```
> fruits
[1] "apple/orange/banana"

> str_sub(fruits,start=1,end=3)
[1] "app"

> str_sub(fruits,start=6,end=9)
[1] "/ora"

> str_sub(fruits,start=-5)
[1] "anana"
```

(11) str_trim() 함수 - 공백 제거하기 (중간에 있는 공백은 X, replace함수 등으로 대체)

- side 옵션 선택시 left / right만 제거

```
> str_trim(' apple banana berry ')
[1] "apple banana berry"

> str_trim('  apple banana berry ', side="left")
[1] "apple banana berry "
```

연습문제

1. emp 데이터에서 S로 시작하는 사람의 name, deptno, sal 출력

```
> f1<- emp[str_detect(emp$ENAME,'^[S]'),c("ENAME","DEPTNO","SAL")]
  ENAME DEPTNO  SAL
1 SMITH     20  800
8 SCOTT     20 3000
```

2. 위의 데이터를 deptno, sal별로 정렬해서 출력

```
> library(doBy)
> orderBy( ~ DEPTNO + SAL, f1)
```

3. emp2 데이터에서 각 직원의 지역번호를 추출하여 새로운 컬럼에 추가

##특수기호를 일반기호로 인식할 때 (detect, locate, split) -> "₩₩)" 특수기호 무력화 가능

```
> library(stringr)
> emp2$LOC<-unlist(as.data.frame(str_split(emp2$TEL,"\\)"))[1,])
----------------------------------------과정

1. 리스트 구조(첫 번째 요소만 뽑기 어려움)
str_split(emp2$TEL,"\\)")
2. 데이터프레임으로 변경(각 첫 번째 행만 가져오기 위해서)
as.data.frame(str_split(emp2$TEL,"\\)"))[1,]
3. 데이터 프레임에 새로운 칼럼 추가는 벡터형 권장
(데이터프레임/리스트 형태로 데이터프레임에 추가 시 key 값 때문에 오류 가능성 있음)
unlist(as.data.frame(str_split(emp2$TEL,"\\)"))[1,])
------------------------------------------------

cf. 파이썬은
str_split(emp2$TEL,"\\)")[[:]][1]
리스트 구조 편하게 출력 가능 (R 은 불가)


+ 만약 벡터연산이 안 되는 함수라면
mapply(str_split, emp2$TEL,'\\)') 형태로 적용
------------------------------------------------

-내가 푼 방법
> emp2$LOC <- str_sub(emp2$TEL, 1, str_locate(emp2$TEL,"\\)")[,1]-1)
```

split : 벡터연산 가능 -> apply 적용 안 해도 됨 / **cf** python은 적용함수 써야 함

4. data1.csv 데이터에서 컬럼이름에 X를 제거

```
> library(stringr)

> colnames(data1) <- str_replace(colnames(data1),"X","")
```

5. instr(X,x,m,n)과 같은 함수를 만들고 데이터에 적용시켜보자

```
> library(stringr)

instr<- function(data,x,m=1,n=1){
  if (m > 0){
    return(str_locate_all(str_sub(data,m),x)[[1]][n,1] + m - 1)
```

```
  }
  else if (m<0){
    return(str_locate_all(str_sub(data, start=1, end=str_length(data)+m+1), x)[[1]]
    [NROW(str_locate_all(str_sub(data, start=1, end=str_length(data)+m+1),x)[[1]])-n+1, 1])
  }
}
```

----------내 흔적들----------

```
> instr<-function(X,x,m,n){
+   for(i in 1:m){
+   if(m <= str_locate_all(X,x)[[1]][i,1]){
+     return(str_locate_all(X,x)[[1]][i+n-1,1])
+   }
+   }
+ }
```

---------- 마이너스 어렵다 ㅠ^ㅠ

```
 instr<-function(X,x,m,n){
   for(i in 1:m){
   if(m > 0 && m <= str_locate_all(X,x)[[1]][i,1] ){
     return(str_locate_all(X,x)[[1]][i+n-1,1])
   } else if (m < 0 && str_length(X)+m+1 >= str_locate_all(X,x)[[1]][NROW(str_locate_all(X,x)[[1]])-
i+1,1] ){
     return(str_locate_all(X,x)[[1]][NROW(str_locate_all(X,x))[[1]]-i+1-n+1,1])
   }
   }
}
```

```
str_length(data)+m-1
```

```
instr('pineapple','p',2,1)
```

```
instr('pineapple','p')
```

```
mappply(instr,emp2$TEL,'₩₩)')
```

정규식 표현식

1) 정규식에서 사용되는 주요 기호들과 의미

사용법	설명요약
₩₩d	Digit, 0,1,2,3,4,5,...,9
₩₩D	숫자가 아닌 것
₩₩s	공백
₩₩S	공백이 아닌 것
₩₩w	단어
₩₩W	단어가 아닌 것
₩₩t	Tab
₩₩n	New line (엔터문자)
^	시작되는 글자
$	마지막 글자
₩	Escape character(탈출문자) e.g. ₩₩ is "₩", ₩+ is "+"
|	두 개 이상의 조건을 동시에 지정 e.g. /(e|d)n/ matches "en" and "dn" - 패턴 두 개 쓸 때
.	엔터(New line)을 제외한 모든 문자

[ ] : 한 글자

[ab]	a 또는 b
[^ab]	a와 b를 제외한 모든 문자 ^이 [ ] 안에 있으면 제외조건
[0-9]	모든 숫자 =[:digit:]
[A-Z]	영어 대문자
[a-z]	영어 소문자
[A-z] [A-Za-z]	모든 영문자 (대소문자 전부) -> 다른 프로그램과 공용되는 형태
i+	i가 최소 1회 이상 나오는 경우 / e.g. 카톡분석-> ㅋ, ㅋㅋ, ㅋㅋㅋ, ... 같이 인식
i*	i가 최소 0회 이상 나오는 경우
i?	i가 최소 0회에서 최대 1회만 나오는 경우
i{n}	i가 연속적으로 n회 나오는 경우

[:alnum:]	문자와 숫자가 나오는 경우 [:alpha:] and [:digit:]
[:alpha:]	문자가 나오는 경우 [:lower:] and [:upper]
[:digit:]	Digits : 0 1 2 3 4 5 6 7 8 9
[:blank:]	공백이 있는 경우 e.g. space,tab
[:cntrl:]	제어문자가 있는 경우
[:graph:]	Graphical characters [:alnum:] and [:punct:]
[:punct:]	특수문자 e.g. ! " # $ % ^ ' ( ) * + - . / : ; < = > ? @ [₩] % ^ _ ` { | } ~
[:print:]	숫자, 문자, 특수문자, 공백 모두 [:alnum:], [:punct:] and space
[:lower:]	소문자가 있는 경우
[:upper:]	대문자가 있는 경우
[:space:]	공백문자 : tab, newline, vertical tab, form feed, carriage return, space
[:xdigit:]	16진수가 있는 경우 : 0 1 2 3 4 5 6 7 8 9 A B C D E F a b c d e f

2) grep(pattern, a)

- 특정 패턴만 골라내기 / 기본함수
- 패턴이 해당되는 **벡터의 위치**를 출력. (not T/F형태)
- 두 가지 패턴을 벡터를 통해 동시에 찾는 경우 paste() 함수, collapse='|' 옵션 사용하기

```
> char <- c('apple','Apple','APPLE','banana','grape')
> grep('apple',char)
[1] 1
> char2<-c('apple','banana')
> grep(char2,char)      #벡터 형태로 grep 에 바로 넣을 경우 첫 원소만 인식함
[1] 1
Warning message:
In grep(char2, char) :
  argument 'pattern' has length > 1 and only the first element will be used


# 두 가지 패턴을 벡터를 통해 동시에 찾는 경우

> grep(paste(char2,collapse='|'),char)
[1] 1 4

# 찾는 문자가 두 개 이상 존재할 경우

> grep('pp',char)
[1] 1 2


# Value=T로 값 자체 출력하기
> grep('^A',char,value=T)
[1] "Apple" "APPLE"
> grep('e$',char,value=T)
[1] "apple" "Apple" "grape"
```

```
> char2<-c('grape1','apple1','apple','orange','Apple')
> grep('[1-9]',char2,value=T)
[1] "grape1" "apple1"
> grep('[[:upper:]]',char2,value=T)
[1] "Apple"
```

-> [ ] 한 글자 + [:upper:] 표현식 = [[:upper:]]

3) nchar()

- 입력된 배열이나 문자열의 길이를 리턴 / 기본함수

```
> char
[1] "apple"  "Apple"  "APPLE"  "banana" "grape"
> nchar(char)
[1] 5 5 5 6 5
> nchar('James Seo')
[1] 9
> nchar('홍길동')
[1] 3
```

4) paste("a","b","c")

- a,b,c를 합쳐 하나의 문자열처럼 출력 (cf. 연결연산자 ||, strings패키지 str_c)
- 구분 기본값 : 공백

```
> paste('Seo','Jae','Su')
[1] "Seo Jae Su"

> paste('Seo','Jae','Su', sep='') # 구분자(공백)를 지움
[1] "SeoJaeSu"

> paste('Seo','Jae','Su', sep='/') # 구분자 /로 지정
[1] "Seo/Jae/Su"

> paste('I','\'m Hungry')  # 특수문자가 있을 경우 escape character(\) 주의
[1] "I 'm Hungry"
```

5)substr("a",시작위치, 끝나는 위치)

- 문자열 추출
- 끝나는 '위치' != 문자 길이

```
> substr('abc123',3,3)
[1] "c"

> substr('abc123',3,5)
[1] "c12"
```

6) strsplit("문자열", split="기준문자")

- 문자열 분리

```
> strsplit('2014/11/22',split='/')
[[1]]
[1] "2014" "11"   "22"
```

7) regexpr("pattern", text)

- 정규식 표현을 활용한 패턴 검색

```
> grep('-','010-8706-4712')   # grep 으로는 위치를 찾을 수 없음. 행위치 반환만
[1] 1

> regexpr("-","010-8706-4712")  # 처음 나오는 '-' 문자 찾기
[1] 4
```

cf.

str_locate() -> 패턴 찾기 불가

----유인물 끝

06 데이터 분리 및 병합

데이터 분리

split()

split(x, f)

- 결과 : **리스트**

x : 분리할 벡터 또는 데이터프레임

f : 분리할 기준을 저장한 팩터/벡터

```
> head(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
…
$versicolor
   Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
51          7.0         3.2          4.7         1.4 versicolor
52          6.4         3.2          4.5         1.5 versicolor
53          6.9         3.1          4.9         1.5 versicolor
...
$virginica
    Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
101          6.3         3.3          6.0         2.5 virginica
102          5.8         2.7          5.1         1.9 virginica
103          7.1         3.0          5.9         2.1 virginica
…
```

<<<<lapply + split >>>> / sapply + split

list key별 계산 -> lapply / sapply

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

$versicolor
[1] 5.936

$virginica
[1] 6.588
```

```
> sapply(split(iris$Sepal.Length, iris$Species),mean)
    setosa versicolor  virginica
     5.006      5.936      6.588
```

subset()

subset(x, subset) / subset(x, subset, select)

- 조건을 만족하는 특정 부분만 취하는 용도로 사용한다.
 - 장점 : 여러 조건 연산 가능
 - 주의 ; &, |는 한 개씩만 사용한다 (원소별 T/F값 반환해야 함)
   cf.if문 -> &&, || 중첩사용 for 하나만의 TRUE / FALSE값
 - 불리언 -> row index에 사용 / subset에 사용

x : 일부를 취할 객체

subset : 데이터를 취할 것인지 여부

select : 데이터프레임의 경우 선택하고자 하는 컬럼

```
> subset(iris, Species=="setosa")   # setosa만 뽑아낸 데이터
   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
…

> subset(iris, Species=="setosa" & Sepal.Length > 5.0)
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1           5.1         3.5          1.4         0.2  setosa
6           5.4         3.9          1.7         0.4  setosa
11          5.4         3.7          1.5         0.2  setosa
…

> subset(iris, select = c(Sepal.Length, Species))
    Sepal.Length    Species
1            5.1     setosa
2            4.9     setosa
3            4.7     setosa
…

> subset(iris, select = -c(Sepal.Length, Species)) cf. 벡터 바깥에 -쓰기
    Sepal.Width Petal.Length Petal.Width
1           3.5          1.4         0.2
2           3.0          1.4         0.2
3           3.2          1.3         0.2
4           3.1          1.5         0.2
…
```

cf. names()와 %in%을 사용하여 제외하는 방법과 비교 / 원소제외 : - 사용, 조건제외 : ! 사용

```
> iris[,!names(iris) %in% c("Sepal.Length","Species")]
    Sepal.Width Petal.Length Petal.Width
1           3.5          1.4         0.2
2           3.0          1.4         0.2
3           3.2          1.3         0.2
4           3.1          1.5         0.2
```

데이터 병합

merge()

merge(x, y, by, by.x=by, by.y=by, all=FALSE)

- **두 데이터 프레임**만을 공통된 값을 기준으로 묶는 함수 cf. in db - JOIN
- 조인조건이 없는 경우 공통된 이름의 column으로 자동 조인 연산 (cf. db->카티시안 곱)
- all ~ (all=TRUE / all.x=TRUE / all.y=TRUE ---- FULL / LEFT / RIGHT OUTER JOIN)

```
> x<-data.frame(name=c("a","b","c"),math=c(15,75,57))
> y<-data.frame(name=c("c","b","a"),english=c(4,5,6))
> merge(x,y)
  name math english
1    a   15       6
2    b   75       5
3    c   57       4
```

```
> d1<-
data.frame(name=c("a","b","c"),deptno1=c(10,20,30),sal=c(151,457,357))
> d2<-
data.frame(ename=c("d","b","c"),deptno2=c(20,20,30),comm=c(100,200,300))
#merge는 조인 컬럼을 지정하지 않으면 양쪽에 같은 이름으로 된 컬럼으로 자동조인 수행
```

#조인 컬럼이 여러개일 경우 벡터로 전달

```
> merge( d1, d2, by.x=c("name","deptno1"),by.y=c("ename","deptno2"))
  name deptno1 sal comm
1    b      20 457  200
2    c      30 357  300
```

★merge 교재에 없는 예제 ~ 아우터조인★

공통된 column 이름이 없는 경우

```
> d1<- data.frame(name=c("a","b","c"),sal=c(151,457,357))
> d2<- data.frame(ename=c("d","b","c"),comm=c(100,200,300))

> merge( d1, d2, by.x="name", by.y="ename")
  name sal comm
1    b 457  200
2    c 357  300
```

```
cf.  # 조인 컬럼이 없으므로(공통된 이름 컬럼) 카티시언 곱 발생
> merge(d1,d2)
  name sal ename comm
1    a 151     d  100
2    b 457     d  100
3    c 357     d  100
4    a 151     b  200
5    b 457     b  200
6    c 357     b  200
7    a 151     c  300
8    b 457     c  300
9    c 357     c  300
************************
```

조인 설정 옵션

all=FALSE : 기본값 ~ 내추럴조인

all=TRUE FULL OUTER JOIN

all.x=TRUE LEFT OUTER JOIN

all.y=TRUE RIGHT OUTER JOIN

```
> merge(d1, d2, by.x="name", by.y="ename")
#조인의 기본 조건은 조인 조건에 맞는 데이터만 출력
  name sal comm
1    b 457  200
2    c 357  300

> merge(d1, d2, by.x="name", by.y="ename", all= TRUE)
#full outer join 수행
  name sal comm
1    a 151   NA
2    b 457  200
3    c 357  300
4    d  NA  100


> merge(d1, d2, by.x="name", by.y="ename", all.x= TRUE)
#left outer join 수행
  name sal comm
1    a 151   NA
2    b 457  200
3    c 357  300


> merge(d1, d2, by.x="name", by.y="ename", all.y= TRUE)
#right outer join 수행
  name sal comm
1    b 457  200
2    c 357  300
3    d  NA  100
```

(열 이름을 바꾸지 말고 by.x, by.y 옵션을 사용해서 조인하기!)

20180517

07) 데이터 정렬

sort()

sort(x, decreasing=FALSE, na.last=NA)

- 주어진 벡터를 정렬한 결과를 반환한다.
- order()와 같은 역할을 하지만, 정렬된 결과를 바로 반환함
- 벡터만 정렬이 가능

#na.last - NA값을 마지막에 둔다

```
> x<-c(20,45,78,12,4)
> sort(x)
[1]  4 12 20 45 78
> sort(x,decreasing=T)
[1] 78 45 20 12  4
```

order()

order(x, decreasing=FALSE)

- 위치를 반환함

cf. orderBy - doBy패키지 함수

08) 데이터 프레임 컬럼 접근

cf. 인덱싱 방법 - 컬럼색인 / df$colname / 데이터프레임 함수접근

데이터 프레임 함수접근

with()

with(data, expr, ...)

- 코드 블록 안에서 필드 이름만으로 데이터를 곧바로 접근할 수 있게 한다

```
> with(iris,{
+   print(mean(Sepal.Length))
+   print(mean(Sepal.Width))
+ })
[1] 5.843333
[1] 3.057333
```

within()

- with()와 동일한 기능을 제공하지만, 데이터에 저장된 값을 손쉽게 변경하는 기능 제공

```
> iris111<-iris
> iris111[1,1] <- NA
> head(iris111)
  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

> median_per_species <- sapply(split(iris$Sepal.Length,iris$Species),
median, na.rm=T)
> iris111 <- within(iris,{
+   Sepal.Length <- ifelse(is.na(Sepal.Length), median_per_species[Species],
Sepal.Length)
+
+ })

> head(iris111)
  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
```

cf. within()없이 사용한다면

iris111$Sepal.Length[is.na(Sepal.Length)] <- median_per_species[iris$Species]

attach()

- attach()이후 코드에서는 필드 이름만으로 데이터를 곧바로 접근할 수 있게 한다.

detach()

- attach()의 반대 역할로, detach() 이후 코드에서 더 이상 필드 이름으로 데이터를 곧바로 접근할 수 없게 한다.

```
> Sepal.Width
Error: object 'Sepal.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 'Sepal.Width' not found
```

주의

attah()한 후 이루어진 변수의 수정은 detach()시 원래의 데이터 프레임에는 반영되지 않는다.

search()

09) 조건에 맞는 데이터의 색인 찾기

which()

조건이 참인 색인을 반환한다.

which.max()

최댓값의 위치를 반환한다. (숫자벡터)

which.min()

최솟값의 위치를 반환한다. (숫자벡터)

★which -> 행 번호를 출력 -> 재색인 필요

cf. 색인찾기

-subset()

-row색인 ~ 불리언인덱스

-which()

```
> 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 29 30 31 32 33 34 35
[36] 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
```

```
> head(iris[which(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
```

```
> which.min(iris$Sepal.Length)
[1] 14
> which.max(iris$Sepal.Length)
[1] 132
```

10) 그룹별 연산

aggregate()

aggregate(x, by, FUN)

x - R 객체 / by - 그룹으로 묶을 값의 리스트 / FUN - 그룹별 요약치 계산에 사용할 함수

aggregate(formula, data, FUN)

formula - y ~ x 형태로 y는 출력할 값, x는 그룹으로 묶을 때 사용할 기준값

data - formula를 적용할 데이터

- 데이터를 그룹으로 묶은 후 임의의 함수를 그룹에 적용할 수 있다.
- 포뮬라 형태를 사용하면 확장성이 용이하고, 편리하다.

install.packages("googleVis")

library(googleVis)

```
> aggregate(Sales ~ Fruit, Fruits, sum)
    Fruit Sales
1  Apples   298
2 Bananas   260
3 Oranges   287

> aggregate(Sales + Sales ~ Fruit, Fruits, sum)
    Fruit Sales + Sales
1  Apples          596
2 Bananas          520
3 Oranges          574
```

formula 형탠데 +로 추가를 하는데 값이 하나로 됨 @0@ -함수마다 formula 조금씩 다름.

aggregate에서는 컬럼추가가 cbind()말고 불가능

```
> aggregate(cbind(Sales,Fruit) ~ Fruit, Fruits, sum)
    Fruit Sales Fruit
1  Apples   298     3
2 Bananas   260     6
3 Oranges   287     9
```

cf. 그룹별 연산 함수

- rapply, sapply (+split)
- tapply
- aggregate
- ddply

11) 편리한 처리를 위한 데이터의 재표현 - 시각화 때 사용되는 함수

stack()

- 컬럼화된 데이터를 하나의 컬럼으로 쌓아야 할 때 (for 분석 쉽게) ~ Tidy Data

unstack()

- Tidy Data -> 그래프 작성시 어려움 , 다시 column별 데이터 저장
- 포뮬라 사용 / 왼쪽에는 관측값 ~ 오른쪽에는 관측값이 온 곳을 표현하는 팩터 (cf. 파이썬 X)

```
> x <- data.frame(a=c(3, 2, 9),
+                 b=c(5, 3, 2),
+                 c=c(4, 5, 7))
> x
  a b c
1 3 5 4
2 2 3 5
3 9 2 7
> stack(x)
  values ind
1      3   a
2      2   a
3      9   a
4      5   b
5      3   b
6      2   b
7      4   c
8      5   c
9      7   c
```

```
> unstack(x_stacked, values ~ ind)
  a b c
1 3 5 4
2 2 3 5
3 9 2 7
```

연습문제

1. emp 데이터를 사용하여 각 부서별 최대 연봉값을 구하여라

sapply(split(empSAL,empDEPTNO), max) #벡터

tapply(empSAL,empDEPTNO,max) #벡터

aggregate(SAL ~ DEPTNO, emp, max) #데이터프레임

ddply(emp, .(DEPTNO), summarise, max=max(SAL)) #데이터프레임

/ 컬럼 여러개 표현할 때엔 aggregate에 cbind()형식으로만 aggregate() 함수로 동시에 여러 개 컬럼 그룹핑 가능

2. 위의 데이터를 활용하여 부서별 최대값을 갖는 직원의 이름, 부서, 연봉, 최대연봉을 함께 출력하여라.

+ in plyr -> ddply(emp, .(DEPTNO), subset, SAL==max(SAL)) #5장

cf inline view in SQL

#merge로 조인해야 하므로(두 데이터 프레임 결합)
 데이터프레임으로 만들어야 함 -> aggregate() 사용
#merge는 EQUI 조인만 가능 -> NON EQUI 조인은 merge 말고 사용자 정의 함수 만들거나
 / SQL 구문 통해서 먼저 그 조건에 맞는 데이터를 가져오는 것이 좋음
 / 혹은 library (SQLDF) 불러오기

```
f1<-aggregate(SAL ~ DEPTNO, emp, max)
colnames(f1)<-c("DEPTNO","MAX_SAL")
merge(emp, f1, by.x=c("DEPTNO","SAL"), by.y=c("DEPTNO", "MAX_SAL"))[, c(4,1,2,2)]
```

3. data2 데이터를 다음과 같은 형식으로 만들어라

```
      line_1   line_2   line_3   line_4
1      88136   608841   149909   244178
2     114628 1079186   344768   555232
3     259282 2960789 1008133 1478641
.....
```

```
data2<-read.csv("data2.csv",stringsAsFactors = F)
unstack(data2, 승차~노선번호)
```

cf. dcast랑 비교- unstack은 왼쪽 기준 컬럼 안 생김

시험문제

dD$gender의 값이 1이나 2가 아닌 값을 NA로 처리하겠다

```
dD$gender[!(dD$gender==1 | dD$gender==2)] <- NA
```