전산통계학 실습

08. R 데이터처리

목차

- 리스트
- 데이터프레임

리스트(List)

- 리스트
  - key-value 형태의 자료구조
  - 벡터는 단일 자료형 밖에 저장할 수 없음
  - 반면, 리스트는 key 별로 서로 다른 자료형을 저장할 수 있음

```
> list_ex <- list(1, "String", c(33, 42, 88), matrix(c(1, 2, 3, 4), nrow=2), TRUE)
> list_ex
[[1]]
[1] 1

[[2]]
[1] "String"

[[3]]
[1] 33 42 88

[[4]]
     [,1] [,2]
[1,]    1    3
[2,]    2    4

[[5]]
[1] TRUE
```

리스트의 생성

```
> list_ex <- list(1, "String", c(33, 42, 88), matrix(c(1, 2, 3, 4), nrow=2), TRUE)
> names(list_ex) <- c("Num", "Char", "V", "M", "Logical")
> list_ex
$`Num`
[1] 1

$Char
[1] "String"

$V
[1] 33 42 88

$M
     [,1] [,2]
[1,]    1    3
[2,]    2    4

$Logical
[1] TRUE
```

- 리스트의 생성 & 이름 지정
  - > list(...)
    - 함수 내에 다양한 데이터 혹은 자료구조들을 나열하여 저장 및 생성

리스트의 생성

```
> list_ex <- list(1, "String", c(33, 42, 88), matrix(c(1, 2, 3, 4), nrow=2), TRUE)
> names(list_ex) <- c("Num", "Char", "V", "M", "Logical")
> list_ex
$`Num`
[1] 1

$Char
[1] "String"

$V
[1] 33 42 88

$M
     [,1] [,2]
[1,]    1    3
[2,]    2    4

$Logical
[1] TRUE
```

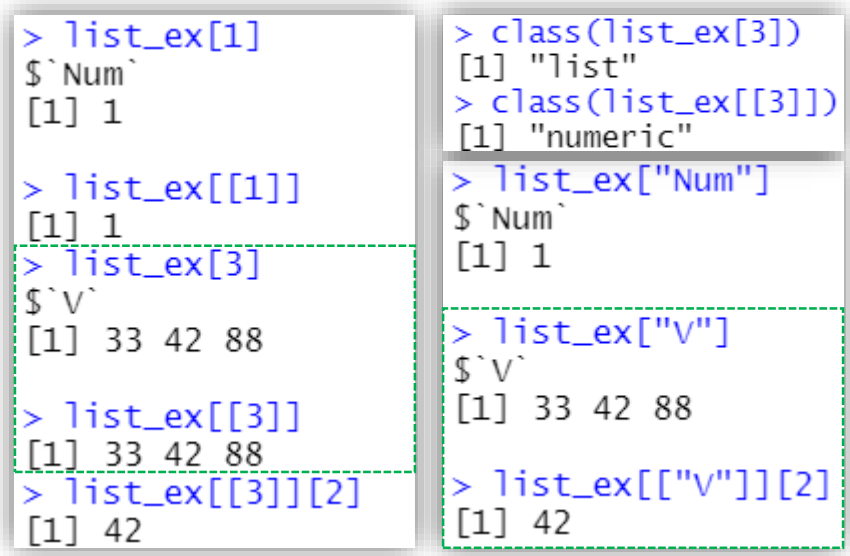
- 리스트의 생성 & 이름 지정
  - > names(list)
    - 기존 다른 자료구조처럼 각 원소의 이름을 반환/지정
    - 지정되는 이름이 key 와 같은 역할을 수행

리스트의 생성

```
> list_ex <- list(Num=1, Char="String", V=c(33,42,88), M=matrix(c(1, 2, 3, 4), nrow=2), Logical=TRUE)
> list_ex
$`Num`
[1] 1

$Char
[1] "String"

$V
[1] 33 42 88

$M
     [,1] [,2]
[1,]    1    3
[2,]    2    4

$Logical
[1] TRUE
```

- 리스트의 생성 & 이름 지정
  - > list(key1=value1, key2=value2, ...)
    - 또는 리스트 생성 시에 미리 이름을 지정하여 생성할 수 있음

리스트의 접근



- 리스트의 접근
  - 다른 자료구조들처럼 key 와 index 를 통해 원소들에 접근 가능
  - value(리스트)에 접근하는 방법
    - 대괄호 1 개가 사용되어 '부분 리스트'가 반환됨
  - value(값)에 접근하는 방법
    - 대괄호 2 개가 사용되어 '저장된 값'이 반환됨

리스트의 접근

```
> list_ex$Num
[1] 1
> list_ex$V
[1] 33 42 88
> class(list_ex$V)
[1] "numeric"
> list_ex[["V"]]
[1] 33 42 88
> class(list_ex[["V"]])
[1] "numeric"
```

- 리스트의 접근
  - > list$key
    - key 로 접근하는 경우 '$' 기호를 통해 접근할 수 있음
    - 대괄호 2개를 사용하여 직접 값에 접근하는 것과 같은 결과를 반환

리스트의 접근

```
> unlist_ex <- unlist(list_ex)
> unlist_ex
     Num     Char       V1       V2       V3       M1       M2       M3       M4  Logical
     "1" "String"     "33"     "42"     "88"      "1"      "2"      "3"      "4"   "TRUE"
> class(unlist_ex)
[1] "character"
> list_ex[[3]]
[1] 33 42 88
> list_ex[[3]] * 3
[1]  99 126 264
> unlist(list_ex[3]) * 3
 V1  V2  V3
 99 126 264
```

- 리스트 해제
  - > unlist(list)
    - 모든 key 내부의 모든 value 들을 벡터로 추출하여 반환
    - 하나의 벡터가 되므로 자료형 우선순위를 반영하여 자료형이 변형됨
    - '직접 값 접근'이 아닌 '부분 리스트 반환'의 연산 수행 등에 이용될 수 있음

리스트의 변경

```
> list_ex
$`Num`
[1] 10

$Char
[1] "String"

$V
[1] 33 42 88

$M
     [,1] [,2]
[1,]    1    3
[2,]    2    4

$Logical
[1] TRUE
```

```
> list_ex$Num <- 5:10
> list_ex
$`Num`
[1]  5  6  7  8  9 10

$Char
[1] "String"

$V
[1] 33 42 88

$M
     [,1] [,2]
[1,]    1    3
[2,]    2    4

$Logical
[1] TRUE
```

- 리스트의 원소 변경
  - 리스트의 접근을 통해 내부 값을 변경할 수 있음
  - key$value 방식을 통해 접근하여 새로운 값으로 변경

리스트의 변경

```
> list_ex[2] <- c("Char", "Char2")
경고메시지(들):
In list_ex[2] <- c("Char", "Char2") :
  number of items to replace is not a multiple of replacement length
> list_ex[2] <- "New Char"
> list_ex
$`Num`
[1]  5  6  7  8  9 10

$Char
[1] "New Char"

$V
[1] 33 42 88

$M
     [,1] [,2]
[1,]    1    3
[2,]    2    4

$Logical
[1] TRUE
```

```
> list_ex$Char <- c("Char", "Char2")
> list_ex
$`Num`
[1]  5  6  7  8  9 10

$Char
[1] "Char"  "Char2"

$V
[1] 33 42 88

$M
     [,1] [,2]
[1,]    1    3
[2,]    2    4

$Logical
[1] TRUE
```

- 리스트의 원소 변경
  - 만약 부분 리스트 접근으로 변경하면 하나의 값만 저장됨
  - 저장된 값을 변경할 때는 직접 값에 접근하여 변경하는 것이 편리함

리스트의 변경

```
> list_ex[6] <- "New"
> list_ex
$`Num`
[1] 1

$Char
[1] "Char"

$V
[1] 33 42 88

$M
     [,1] [,2]
[1,]    1    3
[2,]    2    4

$Logical
[1] TRUE

[[6]]
[1] "New"
```

```
> list_ex[6] <- NULL
> list_ex
$`Num`
[1] 1

$Char
[1] "Char"

$V
[1] 33 42 88

$M
     [,1] [,2]
[1,]    1    3
[2,]    2    4

$Logical
[1] TRUE
```

```
> list_ex$New <- c("Apple", "Banana", "Cherry")
> list_ex
$`Num`
[1] 1

$Char
[1] "Char"

$V
[1] 33 42 88

$M
     [,1] [,2]
[1,]    1    3
[2,]    2    4

$Logical
[1] TRUE

$New
[1] "Apple"  "Banana" "Cherry"
```

- 리스트의 원소 추가/제거
  - 새로운 key 혹은 index 를 통해 원소를 추가 가능
  - 부분 리스트로 접근하여 추가하는 경우 1개의 원소를 추가 가능

리스트의 변경

```
> list_ex[6] <- "New"
> list_ex
$`Num`
[1] 1

$Char
[1] "Char"

$V
[1] 33 42 88

$M
     [,1] [,2]
[1,]    1    3
[2,]    2    4

$Logical
[1] TRUE

[[6]]
[1] "New"
```

```
> list_ex[6] <- NULL
> list_ex
$`Num`
[1] 1

$Char
[1] "Char"

$V
[1] 33 42 88

$M
     [,1] [,2]
[1,]    1    3
[2,]    2    4

$Logical
[1] TRUE
```

```
> list_ex$New <- c("Apple", "Banana", "Cherry")
> list_ex
$`Num`
[1] 1

$Char
[1] "Char"

$V
[1] 33 42 88

$M
     [,1] [,2]
[1,]    1    3
[2,]    2    4

$Logical
[1] TRUE

$New
[1] "Apple"  "Banana" "Cherry"
```

- 리스트의 원소 추가/제거
  - 반대로, 기존 key 혹은 index 를 통해 원소를 제거 가능

리스트의 병합

```
> list1 <- list(1, 5:10, TRUE)
> list2 <- list("Apple", c("Mon", "Tue", "Wen"))
> merged <- c(list1, list2)
> merged
[[1]]
[1] 1

[[2]]
[1]  5  6  7  8  9 10

[[3]]
[1] TRUE

[[4]]
[1] "Apple"

[[5]]
[1] "Mon" "Tue" "Wen"
```

- 리스트의 병합
  - 벡터를 생성하는 combine 함수를 이용하여 리스트 병합 가능

실습 문제

- 리스트의 생성과 변경
  1. 아래 출력과 같은 day, month, year 등의 key 들과 내부 value 들을 가진 'calendar' 리스트와 'calendar2' 리스트를 생성한다.
     - calendar 리스트는 숫자벡터 생성을 응용하여 만들고, calendar2 리스트는 calendar 를 이용하여 만든다.
  2. 'calendar2' 리스트의 year 로부터 1900을 제외한 값들을 추출한 'year1900' 벡터를 만든다.
     - calendar2 리스트의 year 에 저장된 value 들을 연산에 이용한다.

```
> calendar
$`day`
[1] "MON" "TUE" "WEN" "THU" "FRI" "SAT" "SUN"

$month
 [1]  1  2  3  4  5  6  7  8  9 10 11 12

$year
[1] 1900 1920 1940 1960 1980 2000

> calendar2
$`day`
[1] "MON" "TUE" "WEN" "THU" "FRI" "SAT" "SUN"

$month
 [1]  1  2  3  4  5  6  7  8  9 10 11 12

$year
 [1] 1900 1910 1920 1930 1940 1950 1960 1970 1980 1990 2000

> year1900
 [1]   0  10  20  30  40  50  60  70  80  90 100
```

데이터프레임(DataFrame)

- 데이터프레임
  - 행렬과 같은 2D(Dimension) 형태의 자료구조
    - 데이터 처리에 유용한 Spread Sheet 형태로 저장
    - 데이터를 효율적으로 다룰 수 있는 자료구조
  - 각 행은 데이터 값, 각 열은 데이터 변수(속성)
    - 각 열은 동일한 길이를 가진 벡터로 저장해야 함
    - 따라서, 각 열은 동일한 자료형의 데이터를 가짐

데이터프레임의 생성

```
> d <- data.frame(name=c('A', 'B', 'C'), salary=c(30000, 42000, 38000), check=c(T, F, T))
> d
  name salary check
1    A  30000  TRUE
2    B  42000 FALSE
3    C  38000  TRUE
> d$name
[1] A B C
Levels: A B C
> d$salary
[1] 30000 42000 38000
> d[1]
  name
1    A
2    B
3    C
> d[2, 3]
[1] FALSE
> d[2, ]
  name salary check
2    B  42000 FALSE
```

- 데이터프레임의 생성
  - > data.frame(...)
    - 리스트와 비슷하게 key-value 형태로 데이터를 저장하여 생성
    - 각 value 들은 동일한 길이를 가진 벡터여야 함

데이터프레임의 접근

```
> d <- data.frame(name=c('A', 'B', 'C'), salary=c(30000, 42000, 38000), check=c(T, F, T))
> d
  name salary check
1    A  30000  TRUE
2    B  42000 FALSE
3    C  38000  TRUE
> d$name
[1] A B C
Levels: A B C
> d$salary
[1] 30000 42000 38000
> d[1]
  name
1    A
2    B
3    C
> d[2, 3]
[1] FALSE
> d[2, ]
  name salary check
2    B  42000 FALSE
```

- 데이터프레임의 접근
  - > DataFrame$key
    - 리스트와 같이 key를 통해 데이터 접근 가능 (column 벡터가 반환됨)
  - > DataFrame[index]
    - 대괄호 하나와 index 로 접근하는 경우 부분 데이터프레임이 반환됨

데이터프레임의 접근

```
> d <- data.frame(name=c('A', 'B', 'C'), salary=c(30000, 42000, 38000), check=c(T, F, T))
> d
  name salary check
1    A  30000  TRUE
2    B  42000 FALSE
3    C  38000  TRUE
> d$name
[1] A B C
Levels: A B C
> d$salary
[1] 30000 42000 38000
> d[1]
  name
1    A
2    B
3    C
> d[2, 3]
[1] FALSE
> d[2, ]
  name salary check
2    B  42000 FALSE
```

- 데이터프레임의 접근
  - > DataFrame[row, column]
    - 행렬과 같은 형태이므로 행, 열 index로 접근 가능
    - 행렬처럼 한 쪽만을 기재하여 전체 행 혹은 열에 접근 가능

데이터프레임의 함수

- 데이터프레임의 함수
  - 데이터프레임은 행렬 형태이며, 각각 행과 열은 벡터

함수	설명
dim(x)	차원(행의 개수, 열의 개수) 반환
nrow(x)	행의 개수 반환
ncol(x)	열의 개수 반환
head(x, [n])	앞에서부터 n개의 행 반환 (기본값=6)
tail(x, [n])	뒤에서부터 n개의 행 반환 (기본값=6)
colnames(x)	열 이름 벡터 반환
rownames(x)	행 이름 벡터 반환
class(x)	해당 데이터의 자료형 타입을 반환
is.data.frame(x)	자료형이 데이터프레임인지 확인하여 Boolean 반환
as.data.frame(x)	데이터프레임 자료형으로 변환

데이터프레임의 함수

```
> str(Cars93)
'data.frame':   93 obs. of  27 variables:
 $ Manufacturer      : Factor w/ 32 levels "Acura","Audi",..: 1 1 2 2 3 4 4 4 4 5 ...
 $ Model             : Factor w/ 93 levels "100","190E","240",..: 49 56 9 1 6 24 54 74 73 35 ...
 $ Type              : Factor w/ 6 levels "Compact","Large",..: 4 3 1 3 3 3 2 2 3 2 ...
 $ Min.Price         : num  12.9 29.2 25.9 30.8 23.7 14.2 19.9 22.6 26.3 33 ...
 $ Price             : num  15.9 33.9 29.1 37.7 30 15.7 20.8 23.7 26.3 34.7 ...
 $ Max.Price         : num  18.8 38.7 32.3 44.6 36.2 17.3 21.7 24.9 26.3 36.3 ...
 $ MPG.city          : int  25 18 20 19 22 22 19 16 19 16 ...
 $ MPG.highway       : int  31 25 26 26 30 31 28 25 27 25 ...
 $ AirBags           : Factor w/ 3 levels "Driver & Passenger",..: 3 1 2 1 2 2 2 2 2 2 ...
 $ DriveTrain        : Factor w/ 3 levels "4WD","Front",..: 2 2 2 2 3 2 2 3 2 2 ...
 $ Cylinders         : Factor w/ 6 levels "3","4","5","6",..: 2 4 4 4 2 2 4 4 4 5 ...
 $ EngineSize        : num  1.8 3.2 2.8 2.8 3.5 2.2 3.8 5.7 3.8 4.9 ...
 $ Horsepower        : int  140 200 172 172 208 110 170 180 170 200 ...
 $ RPM               : int  6300 5500 5500 5500 5700 5200 4800 4000 4800 4100 ...
 $ Rev.per.mile      : int  2890 2335 2280 2535 2545 2565 1570 1320 1690 1510 ...
 $ Man.trans.avail   : Factor w/ 2 levels "No","Yes": 2 2 2 2 2 1 1 1 1 1 ...
 $ Fuel.tank.capacity: num  13.2 18 16.9 21.1 21.1 16.4 18 23 18.8 18 ...
 $ Passengers        : int  5 5 5 6 4 6 6 6 5 6 ...
 $ Length            : int  177 195 180 193 186 189 200 216 198 206 ...
 $ Wheelbase         : int  102 115 102 106 109 105 111 116 108 114 ...
 $ Width             : int  68 71 67 70 69 69 74 78 73 73 ...
 $ Turn.circle       : int  37 38 37 37 39 41 42 45 41 43 ...
 $ Rear.seat.room    : num  26.5 30 28 31 27 28 30.5 30.5 26.5 35 ...
 $ Luggage.room      : int  11 15 14 17 13 16 17 21 14 18 ...
 $ Weight            : int  2705 3560 3375 3405 3640 2880 3470 4105 3495 3620 ...
 $ Origin            : Factor w/ 2 levels "USA","non-USA": 2 2 2 2 2 1 1 1 1 1 ...
 $ Make              : Factor w/ 93 levels "Acura Integra",..: 1 2 4 3 5 6 7 9 8 10 ...
```

- 데이터프레임의 데이터 요약
  - > str(x)
    - 데이터프레임의 구조, 클래스(자료형), 데이터의 수, 열의 내용 등의 정보를 제공
    - 존재하는 변수 및 저장된 자료형 등 데이터프레임의 구성 자체를 확인하는데 사용

데이터프레임의 함수

```
> summary(Cars93)
   Manufacturer      Model        Type      Min.Price        Price      Max.Price      MPG.city      MPG.highway                    AirBags
 Chevrolet: 8     100    : 1   Compact:16   Min.   : 6.70   Min.   : 7.40   Min.   : 7.9   Min.   :15.00   Min.   :20.00   Driver & Passenger:16
 Ford     : 8     190E   : 1   Large  :11   1st Qu.:10.80   1st Qu.:12.20   1st Qu.:14.7   1st Qu.:18.00   1st Qu.:26.00   Driver only       :43
 Dodge    : 6     240    : 1   Midsize:22   Median :14.70   Median :17.70   Median :19.6   Median :21.00   Median :28.00   None              :34
 Mazda    : 5     300E   : 1   Small  :21   Mean   :17.13   Mean   :19.51   Mean   :21.9   Mean   :22.37   Mean   :29.09
 Pontiac  : 5     323    : 1   Sporty :14   3rd Qu.:20.30   3rd Qu.:23.30   3rd Qu.:25.3   3rd Qu.:25.00   3rd Qu.:31.00
 Buick    : 4     535i   : 1   Van    : 9   Max.   :45.40   Max.   :61.90   Max.   :80.0   Max.   :46.00   Max.   :50.00
 (Other)  :57     (Other):87
 DriveTrain  Cylinders    EngineSize      Horsepower         RPM        Rev.per.mile  Man.trans.avail Fuel.tank.capacity   Passengers
 4WD  :10    3      : 3   Min.   :1.000   Min.   : 55.0   Min.   :3800   Min.   :1320   No :32          Min.   : 9.20      Min.   :2.000
 Front:67    4      :49   1st Qu.:1.800   1st Qu.:103.0   1st Qu.:4800   1st Qu.:1985   Yes:61          1st Qu.:14.50      1st Qu.:4.000
 Rear :16    5      : 2   Median :2.400   Median :140.0   Median :5200   Median :2340                   Median :16.40      Median :5.000
             6      :31   Mean   :2.668   Mean   :143.8   Mean   :5281   Mean   :2332                   Mean   :16.66      Mean   :5.086
             8      : 7   3rd Qu.:3.300   3rd Qu.:170.0   3rd Qu.:5750   3rd Qu.:2565                   3rd Qu.:18.80      3rd Qu.:6.000
             rotary: 1   Max.   :5.700   Max.   :300.0   Max.   :6500   Max.   :3755                   Max.   :27.00      Max.   :8.000

     Length        Wheelbase        Width        Turn.circle    Rear.seat.room   Luggage.room      Weight          Origin                Make
 Min.   :141.0   Min.   : 90.0   Min.   :60.00   Min.   :32.00   Min.   :19.00   Min.   : 6.00   Min.   :1695   USA    :48   Acura Integra: 1
 1st Qu.:174.0   1st Qu.: 98.0   1st Qu.:67.00   1st Qu.:37.00   1st Qu.:26.00   1st Qu.:12.00   1st Qu.:2620   non-USA:45   Acura Legend : 1
 Median :183.0   Median :103.0   Median :69.00   Median :39.00   Median :27.50   Median :14.00   Median :3040                Audi 100     : 1
 Mean   :183.2   Mean   :103.9   Mean   :69.38   Mean   :38.96   Mean   :27.83   Mean   :13.89   Mean   :3073                Audi 90      : 1
 3rd Qu.:192.0   3rd Qu.:110.0   3rd Qu.:72.00   3rd Qu.:41.00   3rd Qu.:30.00   3rd Qu.:15.00   3rd Qu.:3525                BMW 535i     : 1
 Max.   :219.0   Max.   :119.0   Max.   :78.00   Max.   :45.00   Max.   :36.00   Max.   :22.00   Max.   :4105                Buick Century: 1
                                                                 NA's   :2       NA's   :11                                  (Other)      :87
```

- 데이터프레임의 데이터 요약
  - > summary(x)
    - 데이터프레임의 변수에 따라 각각 요약된 통계량 정보를 제공
    - 문자열 변수 혹은 Factor 변수 등에서는 각 값의 빈도 수를 출력
    - 수치 자료의 경우 최솟값, 최댓값, 중앙값 등 다양한 통계량 제공

데이터프레임의 변경

```
> d
  name salary check
1    A  30000  TRUE
2    B  42000 FALSE
3    C  38000  TRUE
> newrow <- data.frame(name="D", salary=50000, check=T)
> newrow
  name salary check
1    D  50000  TRUE
> d <- rbind(d, newrow)
> d
  name salary check
1    A  30000  TRUE
2    B  42000 FALSE
3    C  38000  TRUE
4    D  50000  TRUE
> newcol <- data.frame(part=c(1,1,2,2))
> d <- cbind(d, newcol)
> d
  name salary check part
1    A  30000  TRUE    1
2    B  42000 FALSE    1
3    C  38000  TRUE    2
4    D  50000  TRUE    2
```

- 데이터프레임의 행/열 추가/제거
  - 행렬처럼 bind 함수를 이용하여 행/열 추가
  - 부분 데이터프레임 추출 및 재저장을 통해 제거 가능

데이터프레임의 병합

```
> d1
  name salary check part
1    A  30000  TRUE    1
2    B  42000 FALSE    1
3    C  38000  TRUE    2
4    D  50000  TRUE    2
> d2
  name bonus
1    A   YES
2    C   YES
3    D    NO
> d3
  bonus
1   YES
2   YES
3    NO
```

```
> merge(d1, d2)
  name salary check part bonus
1    A  30000  TRUE    1   YES
2    C  38000  TRUE    2   YES
3    D  50000  TRUE    2    NO
> merge(d1, d3)
   name salary check part bonus
1     A  30000  TRUE    1   YES
2     B  42000 FALSE    1   YES
3     C  38000  TRUE    2   YES
4     D  50000  TRUE    2   YES
5     A  30000  TRUE    1   YES
6     B  42000 FALSE    1   YES
7     C  38000  TRUE    2   YES
8     D  50000  TRUE    2   YES
9     A  30000  TRUE    1    NO
10    B  42000 FALSE    1    NO
11    C  38000  TRUE    2    NO
12    D  50000  TRUE    2    NO
```

- 데이터프레임의 병합
  - > merge(DataFrame1, DataFrame2)
    - 연관성 있는 두 데이터프레임을 병합 시 내부 변수가 고려되어 병합됨
    - 중복이 되는 열(변수)의 데이터 중 동일 데이터에 대해서만 병합
    - 만약 중복되는 열이 없는 경우 모든 데이터에 대해 '전체 곱'이 수행됨

데이터의 처리

- 데이터 처리 함수
  - 벡터, 행렬, 데이터프레임 등 여러 데이터들이 저장된 자료구조들에 각 목적에 따라 함수를 효율적으로 적용하기 위해 사용하는 함수
    - 어떤 조건, 연산, 함수 등을 반복적으로 데이터 구조에 적용할 수 있음
    - 기본적으로 벡터 단위로 함수가 적용되어 수행됨

함수	설명
apply()	특정 함수를 해당 '범위 행 혹은 열에 적용'하여 반환
lapply()	특정 함수를 '리스트 및 벡터에 적용'하여 '리스트로 반환'
sapply()	특정 함수를 '리스트 및 벡터에 적용'하고 '효율적인 자료형'으로 반환
tapply()	특정 함수를 요인에 따라 '구분된 그룹별로 적용'하여 반환

데이터의 처리

```
> data
    a  b  c
1   1  3  2
2   2  4  4
3   3  5  6
4   4  6  8
5   5  7 10
6   6  8 12
7   7  9 14
8   8 10 16
9   9 11 18
10 10 12 20
> apply(data, 2, sum)
  a   b   c
 55  75 110
> apply(data, 2, function(x) { sum(x) })
  a   b   c
 55  75 110
```

- 데이터 처리 함수에 대입되는 함수(function)
  - 반복하여 처리하고자 하는 특정 함수를 대입
  - 구현된 함수들은 함수명으로 대입
  - 함수 생성 문법인 function(...) { } 으로도 구현 가능

데이터의 처리

```
> class
  kor mat eng sci
1  92  55 100  44
2  99  60  72  56
3  72  22  55  88
4  45  33 100  92
> apply(class, 2, sum)
kor mat eng sci
308 170 327 280
> apply(class, 1, sum)
[1] 291 287 237 270
> str(Cars93[,4:6])
'data.frame':   93 obs. of  3 variables:
 $ Min.Price: num  12.9 29.2 25.9 30.8 23.7 14.2 19.9 22.6 26.3 33 ...
 $ Price    : num  15.9 33.9 29.1 37.7 30 15.7 20.8 23.7 26.3 34.7 ...
 $ Max.Price: num  18.8 38.7 32.3 44.6 36.2 17.3 21.7 24.9 26.3 36.3 ...
> apply(Cars93[,4:6], 2, mean)
Min.Price     Price Max.Price
 17.12581  19.50968  21.89892
```

- > apply(data, margin, function)
  - data에 대하여 margin 방향의 각 벡터에 대해 function을 각각 적용
    - margin=1 이면 각 행 벡터에 대하여 function 을 적용
    - margin=2 이면 각 열 벡터에 대하여 function 을 적용
    - function이 적용될 수 있는 변수들(행, 열)에 적용

데이터의 처리

```
> str(Cars93[,4:6])
'data.frame':   93 obs. of  3 variables:
 $ Min.Price: num  12.9 29.2 25.9 30.8 23.7 14.2 19.9 22.6 26.3 33 ...
 $ Price    : num  15.9 33.9 29.1 37.7 30 15.7 20.8 23.7 26.3 34.7 ...
 $ Max.Price: num  18.8 38.7 32.3 44.6 36.2 17.3 21.7 24.9 26.3 36.3 ...
> apply(Cars93[,4:6], 2, mean)
Min.Price     Price Max.Price
 17.12581  19.50968  21.89892
> lapply(Cars93[,4:6], sum)
$`Min.Price`
[1] 1592.7

$Price
[1] 1814.4

$Max.Price
[1] 2036.6

> unlist(lapply(Cars93[,4:6], sum))
Min.Price     Price Max.Price
   1592.7    1814.4    2036.6
```

- > lapply(data, function)
  - data 의 각 열에 대하여 function 을 적용하여 결과를 '리스트'로 반환
    - 리스트인 경우에는 각 key 에 따른 value 에 대하여 function 이 적용됨
  - 앞서 배운 unlist(...) 함수를 이용하여 벡터 형태로 변환 가능

데이터의 처리

```
> s1 <- sapply(Cars93[,4:6], sum)
> class(s1)
[1] "numeric"
> s1
Min.Price     Price Max.Price
   1592.7    1814.4    2036.6
> s2 <- sapply(Cars93[,4:6], function(x) { x < 30 })
> class(s2)
[1] "matrix"
> head(s2)
     Min.Price Price Max.Price
[1,]      TRUE  TRUE      TRUE
[2,]      TRUE FALSE     FALSE
[3,]      TRUE  TRUE     FALSE
[4,]     FALSE FALSE     FALSE
[5,]      TRUE FALSE     FALSE
[6,]      TRUE  TRUE      TRUE
```

- > sapply(data, function)
  - lapply 함수와 비슷하게 사용되나, 반환되는 자료구조가 유연함
    - 반복문을 대체하기에 가장 편리한 함수
    - 일반적으로, 수행 결과가 벡터 형태로 반환됨
    - 반환될 결과의 각 value 길이가 다른 경우에도 리스트 형태로 반환됨
    - 이외에는 입력과 결과의 형태에서 차이가 없으면 동일 자료형으로 반환됨

데이터의 처리

```
> tapply(Cars93$Price, Cars93$Manufacturer, mean)
       Acura         Audi          BMW        Buick     Cadillac    Chevrolet 
    24.90000     33.40000     30.00000     21.62500     37.40000     18.18750 
     Chrylser     Chrysler        Dodge        Eagle         Ford          Geo 
    18.40000     22.65000     15.70000     15.75000     14.96250     10.45000 
       Honda      Hyundai     Infiniti        Lexus      Lincoln        Mazda 
    16.46667     10.47500     47.90000     31.60000     35.20000     17.60000 
Mercedes-Benz      Mercury   Mitsubishi       Nissan   Oldsmobile     Plymouth 
    46.90000     14.50000     18.20000     17.02500     17.50000     14.40000 
      Pontiac         Saab       Saturn       Subaru       Suzuki       Toyota 
    16.14000     28.70000     11.10000     12.93333      8.60000     17.27500 
  Volkswagen        Volvo 
    18.02500     24.70000 
```

- > tapply(data, factor, function)
  - data 내부의 factor(범주, 요인)에 따라 그룹별로 function을 적용

데이터의 처리

```
> head(split(Cars93$Price, Cars93$Manufacturer))
$Acura
[1] 15.9 33.9

$Audi
[1] 29.1 37.7

$BMW
[1] 30

$Buick
[1] 15.7 20.8 23.7 26.3

$Cadillac
[1] 34.7 40.1

$Chevrolet
[1] 13.4 11.4 15.1 15.9 16.3 16.6 18.8 38.0

> subset(Cars93$Price, Cars93$Manufacturer=="Audi")
[1] 29.1 37.7
```

- 데이터 분리
  - > split(data, factor)
    - 주어진 factor의 범주에 따라 데이터를 분리하여 리스트로 반환
  - > subset(data, condition)
    - 주어진 condition(조건)에 따라 만족하는 데이터만 분리하여 값을 반환(벡터)

데이터의 처리

```
> Cars93$Price
 [1] 15.9 33.9 29.1 37.7 30.0 15.7 20.8 23.7 26.3 34.7 40.1 13.4 11.4 15.1 15.9 16.3 16.6 18.8 38.0 18.4 15.8 29.5  9.2 11.3 1$
[47] 13.9 47.9 28.0 35.2 34.3 36.1  8.3 11.6 16.5 19.1 32.5 31.9 61.9 14.1 14.9 10.3 26.1 11.8 15.7 19.1 21.5 13.5 16.3 19.5 2$
[93] 26.7
> sort(Cars93$Price, decreasing=T)
 [1] 61.9 47.9 40.1 38.0 37.7 36.1 35.2 34.7 34.3 33.9 32.5 31.9 30.0 29.5 29.1 28.7 28.0 26.7 26.3 26.1 25.8 24.4 23.7 23.3 2$
[47] 17.7 17.5 16.6 16.5 16.3 16.3 15.9 15.9 15.9 15.8 15.7 15.7 15.6 15.1 14.9 14.4 14.1 14.0 13.9 13.5 13.4 13.3 12.5 12.2 1$
[93]  7.4
> sort(Cars93$Price, decreasing=F)
 [1]  7.4  8.0  8.3  8.4  8.4  8.6  9.0  9.1  9.2  9.8 10.0 10.0 10.1 10.3 10.9 11.1 11.1 11.3 11.3 11.4 11.6 11.8 12.1 12.2 1$
[47] 17.7 18.2 18.4 18.4 18.5 18.8 19.0 19.1 19.1 19.3 19.5 19.5 19.7 19.8 19.9 20.0 20.2 20.7 20.8 20.9 21.5 22.7 22.7 23.3 2$
[93] 61.9
> order(Cars93$Price, decreasing=T)
 [1] 59 48 11 19  4 52 50 10 51  2 57 58  5 22  3 78 49 93  9 63 28 77  8 91 87 92 67 38  7 71 37 90 36 41 89 70 82 30 56 66 2$
[78] 79 81 62 32 45 46 84 23 88 73 83 39 80 53 44 31
```

- 데이터 정렬
  - > sort(data, [decreasing=T/F])
    - 데이터를 직접 정렬하여 반환
    - decreasing=T이면 내림차순 정렬
  - > order(data, [decreasing=T/F])
    - 정렬 결과 데이터가 현재 데이터 내에서 존재하는 위치를 반환
    - 전체 데이터에서 각 데이터의 순위에 따른 index를 보는데 유리
    - decreasing=T이면 내림차순 결과에서 각 순위에 따른 index를 반환

데이터의 처리

```
> table(Cars93$Manufacturer)

      Acura        Audi         BMW       Buick     Cadillac   Chevrolet
          2           2           1           4           2           8
    Chrylser    Chrysler       Dodge       Eagle        Ford         Geo
          1           2           6           2           8           2
      Honda     Hyundai    Infiniti       Lexus     Lincoln       Mazda
          3           4           1           2           2           5
Mercedes-Benz     Mercury   Mitsubishi      Nissan  Oldsmobile    Plymouth
          2           2           2           4           4           1
    Pontiac        Saab      Saturn      Subaru      Suzuki      Toyota
          5           1           1           3           1           4
 Volkswagen       Volvo
          4           2
```

- 데이터의 정리
  - > table(data)
    - data 내부에 저장된 중복되지 않은 범주에 따른 데이터의 개수를 정리하여 반환
    - 즉, 저장된 데이터에 대한 도수분포표를 반환

과제

MEDICINE	SUBJECT	EFFECT	PROBLEM
M1	S1	0.62	0.11
M1	S2	0.77	0.05
M1	S3	0.33	0.16
M1	S4	0.27	0.89
M1	S5	0.84	0.77
M2	S1	0.15	0.13
M2	S2	0.13	0.77
M2	S3	0.29	0.22
M2	S4	0.18	0.10
M2	S5	0.62	0.21
M3	S1	0.72	0.33
M3	S2	0.13	0.31
M3	S3	0.09	0.42
M3	S4	0.57	0.55
M3	S5	0.44	0.66

- 문제.
  - 왼쪽 표는 어느 대학에서 안전 약물 3가지를 각각 피실험자 5명에게 실험한 뒤의 결과를 나타낸 표이다.
    - MEDICIN: 약의 이름
    - SUBJECT: 피실험자 이름
    - EFFECT: 약물 효과 발생률
    - PROBLEM: 문제 발생률
  - 왼쪽 표를 데이터프레임으로 저장한다.
  - 아래의 값들을 계산한다.
    - 약물 별 평균 효과 발생률
    - 약물 별 평균 문제 발생률
    - 평균 문제 발생률이 높은 약물부터 순서대로 출력

과제 제출

- 제출 목록
  - 작성한 코드 파일(.R)
  - 결과 출력 화면 (.PDF)
    - 터미널 캡처(이미지)를 Word 혹은 HWP 에 붙여넣어 PDF 로 변환

- 제출 방법
  - 위 목록의 파일들을 하나로 압축
  - 아래 서식으로 압축파일 이름 지정
  - 블랙보드를 통해 제출

- 제출 서식 (XX = 주차번호 ex. 01, 02, ...)
  - 파일 이름: 전산통계학_실습과제_XX주차_학번_이름.zip