전산통계학 실습

R 데이터처리

리스트

- 리스트
  - 벡터와 비슷하게 선형적인 형태를 가진 자료구조
  - key-value 형태로 데이터를 저장
  - 기존 벡터가 서로 다른 자료형을 저장할 수 없는 반면, 리스트는 key 별로 서로 다른 자료형의 데이터를 저장 가능
- 리스트의 생성 예시
  - 한 회사의 직원 데이터베이스
    - 직원A <- (number=1, name="A", salary=50000, union=T, part="A")
    - 직원B <- (number=2, name="B", salary=52000, union=T, part="A")

NUMBER	NAME	SALARY	UNION	PART
1	A	50000	T	A
2	B	52000	T	A

리스트의 생성

```
> employee <- list(name="Kim", salary=50000, union=T, part="Server")
> employee
$`name`
[1] "Kim"

$salary
[1] 50000

$union
[1] TRUE

$part
[1] "Server"
```

```
> fruitbox <- list(box1=c("Apple", "Banana"), box2=c("Melon", "Strawberry"), box3=c("Grape"))
> fruitbox
$`box1`
[1] "Apple"  "Banana"

$box2
[1] "Melon"      "Strawberry"

$box3
[1] "Grape"
```

- 리스트의 생성
  - > list(key1=value1, key2=value2, ...)
    - key는 기존 index에 해당되는 것에 name을 붙이는 것과 같다.
    - value에는 다른 자료구조인 벡터 혹은 행렬 또한 저장 가능하다.

리스트의 접근

```
> fruitbox
$`box1`
[1] "Apple"  "Banana"

$box2
[1] "Melon"      "Strawberry"

$box4
[1] "Kiwi"   "Orange"

> names(fruitbox)
[1] "box1" "box2" "box4"
> fruit <- unlist(fruitbox)
> fruit
      box11       box12       box21       box22       box41       box42
   "Apple"    "Banana"     "Melon" "Strawberry"      "Kiwi"    "Orange"
```

- 리스트 key 추출
  - > names(list)
    - 각 값에 접근할 수 있는 key의 목록을 확인할 수 있다.
- 리스트 value 추출
  - > unlist(list)
    - 모든 key 내부의 모든 value들을 벡터로 추출할 수 있다.
    - unlist()의 결과는 기존 리스트의 값과 동일한 자료형으로 반환된다.

리스트의 변경

```
> fruitbox
$`box1`
[1] "Apple"  "Banana"

$box2
[1] "Melon"      "Strawberry"

$box3
[1] "Grape"
```

```
> fruitbox$box4 <- c("Kiwi", "Orange")
> fruitbox
$`box1`
[1] "Apple"  "Banana"

$box2
[1] "Melon"      "Strawberry"

$box3
[1] "Grape"

$box4
[1] "Kiwi"   "Orange"
```

```
> fruitbox$box3 <- NULL
> fruitbox
$`box1`
[1] "Apple"  "Banana"

$box2
[1] "Melon"      "Strawberry"

$box4
[1] "Kiwi"   "Orange"
```

- 추가
  - 새로운 key를 이용하여 새로운 데이터를 추가할 수 있다.
- 제거
  - 기존 key로 접근하여 NULL을 할당하여 저장된 값을 제거할 수 있다.

데이터프레임

- 데이터프레임
  - 리스트와 같이 다양한 자료형을 행렬 형태로 저장한다.
    - 일반적으로 데이터 처리에 유용한 Excel Spread Sheet 형태로 저장된다.
    - 각 행은 데이터 값을 저장하고 각 열은 해당 데이터의 변수를 저장한다.
  - 실질적인 데이터베이스를 구현할 수 있도록 구현된 자료구조이다.
  - 가장 데이터를 효율적으로 다룰 수 있다.

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 같이 key-value 형태로 데이터를 저장하고 선언할 수 있다.
    - 리스트와 달리 데이터가 행렬 형태로 구성된다.

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 (변수로 접근하므로 열이 출력)
  - > DataFrame[index]
    - 대괄호 하나로 index에 접근하는 경우 해당 저장된 변수 열이 출력된다.

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 형태를 띄고 있기 때문에 행, 열 index로 접근할 수 있다.
    - 기존 행렬과 같이 row 혹은 column 중 한 쪽을 지우는 경우 해당되는 전체 column 혹은 row를 접근할 수 있다.

데이터프레임의 함수

- 데이터프레임의 함수
  - 일반적으로 데이터프레임 또한 행렬의 형태를 가지고 있고, 각각의 행과 열은 벡터이기 때문에 벡터에 대한 다양한 함수들을 적용 가능하다.

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=5)
tail(x, [n])	뒤에서부터 n개의 행 반환 (기본값=5)
names(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 병합

```
> 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 병합
  - 행렬에서 bind 함수를 이용한 것과 같이 데이터프레임 혹은 벡터 등을 통하여 행이나 열을 추가할 수 있다.
    - 단, 각자 bind 되는 행과 열은 각각 기존 데이터의 열과 행의 개수가 같아야 한다.

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 병합하는 경우 내부의 변수를 고려하여 병합할 수 있다.
    - 두 데이터에서 중복이 되는 열(변수)의 데이터 중 동일한 데이터에 대해서만 병합된다.
    - 만약 중복되는 열이 없는 경우 모든 데이터에 대해 '전체 곱'을 수행한다.

데이터의 처리

- 데이터 처리 함수
  - 벡터, 행렬, 데이터프레임 등 여러 데이터들을 저장한 자료구조에 각 목적에 따라 함수를 효율적으로 적용하기 위해 사용하는 함수
    - 어떤 조건, 연산, 함수 등을 반복적으로 데이터 구조에 적용하기 위해 사용한다.
    - 기본적으로 벡터 단위로 함수가 적용되어 벡터 단위 별로 수행된다.

함수	설명
apply()	특정 함수를 해당 범위 행 혹은 열에 적용하여 반환
lapply()	특정 함수를 리스트 및 벡터에 적용하여 리스트로 반환
sapply()	특정 함수를 리스트 및 벡터에 적용하고 효율적인 자료형으로 반환
tapply()	특정 함수를 요인에 따라 구분된 그룹별로 적용하여 반환

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 방향으로 function을 각각 적용한다.
    - margin=1 이면 행 방향으로 function을 적용
    - margin=2 이면 열 방향으로 function을 적용
    - function이 적용될 수 있는 변수들(행, 열)에 적용하여야 한다.

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)
  - data의 각 열에 function을 적용하고, 해당 결과를 리스트로 반환한다.
  - unlist(...) 함수를 이용하여 벡터 형태로 변환할 수 있다.

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
```

함수 선언 방법
> function(...) { ... }

- > sapply(data, function)
  - lapply 함수와 비슷하게 사용하나, 반환되는 자료구조가 서로 다르다.
    - 결과 리스트 내부 원소가 각각 하나인 경우, 벡터로 반환된다.
    - 결과 리스트 내부 원소가 하나 이상이고 타입이 모두 같은 경우, 행렬로 반환된다.
    - 이외의 경우 모두 리스트로 반환된다.

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)에 따라 그룹별로 function을 적용한다.

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 반환한다.
    - decreasing=T이면 내림차순 정렬한다.
  - > order(data, [decreasing=T/F])
    - 정렬 결과 데이터가 현재 데이터 내에서 존재하는 위치를 반환한다.
    - 전체 데이터에서 각 데이터의 순위에 따른 index를 보는데 유리하다.
    - decreasing=T이면 내림차순 결과에서 각 순위에 따른 index를 반환한다.

과제6

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)
  - 결과를 출력한 터미널 캡처(이미지)를 Word 혹은 HWP에 붙여 넣어 PDF 파일로 변환
- 제출 방법
  - 목록의 파일들을 압축
  - 아래 서식으로 압축파일 이름 지정
  - 이메일 제목 또한 지정
- 제출 서식 (X=과제번호)
  - 파일 이름: 통계학실습_과제X_학번_이름.zip
  - 이메일 제목: 통계학실습_과제X_학번_이름
- 제출 이메일
  - gtsk623@gmail.com