5장. 데이터 전처리

문자셋과 인코딩

- 문자셋 (charset, Character Set)
  - 하나의 언어권에서 사용하는 언어를 표현하기 위한 모든 문자(활자)의 모임
  - 우리가 얘기하는 언어를 책으로 출판할 때 필요한 문자(활자)를 모두 모은 것
  - 부호와 공백 등과 같은 특수 문자도 문자셋에 포함

- 인코딩 (encoding)
  - 인코딩은 문자셋을 컴퓨터가 이해할 수 있는 바이트와의 매핑 규칙
  - 예)
    - ASCII Code에서 ABC 등은 문자셋
    - A는 코드 65, B는 코드 66 등 바이트 순서와 매핑한 것이 인코딩
    - 입니다. 따라서 문자셋을 어떻게 매핑하느냐에 따라 하나의 문자셋이 다양한 인코딩을 가질 수 있습니다.
  - 가장 많이 사용하는 인코딩은 "UTF-8", "KSC5601", "ISO-8859-1" 이고 Windows는 "CP949"를 사용

- 문자셋과 인코딩 관련 R 명령어
  - Sys.getlocale() : R의 인코딩 정보 확인

- 기본적인 encoding 값
  - Windows의 경우, "CP949"를 사용합니다.
  - Linux의 경우, "UTF-8"을 사용합니다.
  - Windows와 Linux의 문자셋(charset) 설정에 따라 encoding 값이 달라진다.

write.table()


```
write.table(data, file="파일명", append=FALSE,
                  quote=TRUE, sep=",", row.names=TRUE)
```

구문에서...

- data : data 변수에 저장된 데이터를 저장합니다.
- file : 저장할 파일명을 입력합니다.
- quote : TRUE이면(기본값) 필드의 값을 "로 묶습니다.
- sep : 필드의 구분자를 지정합니다. Default 필드 구분자는 공백(blank)입니다.
- row.names : TRUE(기본값)이면 각 행의 이름도 저장합니다.

read.table()


```
data <- read.table("파일명", header=TRUE, sep=",",
                   stringsAsFactors=FALSE, comment.char="#",
                   fileEncoding="UTF-8", encoding="CP949")
```

구문에서...

- header : TRUE이면 파일의 첫 번째 행을 열 이름으로 지정해 줍니다.
- sep : 데이터파일이 필드 구분자를 지정합니다. 디폴트 필드 구분자는 공백(blank)입니다.
- stringsAsFactors : TRUE이면 문자열을 Factor 형태로 저장합니다.
- comment.char : 주석의 시작 문자를 지정합니다. 디폴트는 #입니다.
- fileEncoding : 파일의 인코딩을 지정합니다.
- encoding : R의 인코딩을 지정합니다.

write.csv()

CSV(Comma-separated values) 파일 형식으로 저장할 것이라면 write.csv() 함수를 이용하면 설정 할 파라미터의 수를 줄일 수 있습니다.

기본적으로 행 이름 열의 열 이름은 없습니다. col.names=NA 및 row.names=TRUE이면 스프레드시트에서 CSV 파일을 읽는 데 사용되는 규칙 인 빈 열 이름이 추가됩니다. 이러한 CSV 파일은 R로 읽을 수 있습니다.

```
write.csv (data, file="파일명")
```

write.csv 함수와 write.csv2 함수는 CSV 파일을 작성하기 위한 편리한 래퍼를 제공합니다. append, col.names, sep, dec 또는 qmethod 등 속성들의 수정은 경고와 함께 무시됩니다.

read.csv()

CSV 형식 파일을 조회할 때 read.csv() 함수를 사용할 수 있습니다.

```
data <- read.csv (file="파일명")
```

다음 코드는 파일에서 csv 데이터를 읽는 예입니다.

```
> newData <- read.csv("iris2.csv")
> head(newData)
  X Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1 1          5.1         3.5          1.4         0.2  setosa
2 2          4.9         3.0          1.4         0.2  setosa
3 3          4.7         3.2          1.3         0.2  setosa
4 4          4.6         3.1          1.5         0.2  setosa
5 5          5.0         3.6          1.4         0.2  setosa
6 6          5.4         3.9          1.7         0.4  setosa
```

iris2.csv 파일은 write.csv() 함수를 이용했기 때문에 새로운 열이 추가됐습니다.

cat

분석 결과 등을 저장할 때 cat()을 이용할 수 있습니다.

```
cat(... , file="", sep=" ", fill=FALSE, labels=NULL,
          append=FALSE)
```

cat()의 주요 용도는 문자열을 이어 출력하는데 사용하지만 file 인자를 지정하면 파일에 데이터를 저장할 수 있습니다.

다음 코드는 iris 데이터의 요약 정보를 파일에 저장하는 예입니다.

```
> irisSummary <- summary(iris)
> cat("iris 데이터 요약:", "\n", irisSummary, "\n",
+      file="summary.txt", append=TRUE)
```

위 코드는 summary.txt 파일에 iris 데이터의 summary 정보를 저장합니다. append=TRUE 속성 때문에 cat 구문을 실행시킬 때마다 기존 파일에 데이터를 추가 저장합니다.

apply ; 계열 함수를 이용하면 반복문 사용보다 코드 최소화, 수행속도 빠름

- apply ; 행렬이나 배열 차원별로 지정함 함수 적용
- lapply ; list apply (함수를 적용한 결과를 list로 반환)
- sapply ; lappply와 유사하나 결과를 리스트 대신 행렬, 벡터로 반환
- vapply ; sapply와 유사하나 FUN의 모든 값이 FUN.VALUE와 호환되는지 확인
- mapply ; sapply와 유사하나, 여러 개 인자를 함수에 전달할 수 있다.
- tapply ; 그룹별 처리를 위한 apply (대상은 list)
- by ; 데이터프레임 대상 tapply함수

apply


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

구문에서...

- X : 대상 자료 객체(행렬, 배열)입니다.
- MARGIN : 차원을 입력합니다. 1이면 행별로 함수를 적용하고, 2이면 열별로 함수를 적용합니다. 3이면 차원(3차원 배열에서)별로 함수를 적용합니다. c(1,2)이면 (행,열) 별로 함수를 적용합니다.
- FUN : 적용할 함수 이름입니다.

다음 코드는 iris 데이터에서 setosa 종의 꽃잎의 길이와 너비 평균을 구하는 예입니다.

```
> setosaData <- subset(iris, select=c(3:4),
+                      subset=iris$Species=="setosa" )

> apply(setosaData, 2, mean)
Petal.Length  Petal.Width
       1.462        0.246
```

lapply

lapply는 list apply입니다. lapply() 함수는 리스트에 지정한 함수를 적용합니다. 함수를 적용한 결과는 리스트로 반환합니다.

```
lapply(X, FUN, ...)
```

구문에서...

- X : 함수를 적용할 벡터 또는 리스트 객체입니다.
- FUN : 적용할 함수 이름입니다.
- ... : 함수에서 사용할 인수들입니다.

sapply

sapply는 simplification apply입니다. sapply() 함수는 lapply() 함수와 유사하지만 리스트대신 행렬, 벡터 등으로 결과를 반환하는 함수입니다.

```
sapply(X, FUN, ..., SIMPLIFY=TRUE, USE.NAMES=TRUE)
```

구문에서...

- X : 대상 리스트 객체입니다.
- FUN : 적용할 함수 이름입니다.
- ... : 함수에서 사용할 인수입니다.
- SIMPLIFY : TRUE(기본값) 이면 연산 결과를 벡터, 행렬 등으로 반환합니다. FALSE 이면 연산 결과를 리스트로 반환합니다.
- USE.NAMES : TRUE(기본값) 이면 이름 속성도 반환합니다. FALSE이면 이름 속성 없이 반환합니다.

vapply

vapply는 values simplification apply입니다. vapply() 함수는 sapply() 함수와 비슷하지만 FUN의 모든 값이 FUN.VALUE와 호환되는지 확인합니다. vapply() 함수는 FUN.VALUE에 의해 미리 지정된 유형의 반환 값을 가지므로 사용하는 것이 더 안전 할 수 있습니다. 그래서 sapply() 함수에 비해 더 안전하게 사용할 수 있고, 더 빠릅니다.

```
vapply(X, FUN, FUN.VALUE, ..., USE.NAMES=TRUE)
```

구문에서...

- X : 대상 리스트 객체입니다.
- FUN : 적용할 함수 이름입니다.
- FUN.VALUE : 반환하는 데이터의 유형을 지정합니다. FUN에서 리턴 값을 위한 템플릿입니다. length(FUN.VALUE) == 1이면 X와 동일한 길이의 벡터가 반환되고, 그렇지 않으면 배열이 반환됩니다. FUN.VALUE가 배열이 아니면 결과는 FUN.VALUE의 길이 행과 X의 길이 열을 가진 행렬입니다. 그렇지 않으면 dim(a) == c(dim(FUN.VALUE), length(X)) 식에 의한 차원을 가집니다.
- ... : 함수에서 사용할 인수입니다.

mapply

mapply()는 sapply()와 유사하지만 다수의 인자를 함수에 넘긴다는 데서 차이가 있습니다. mapply는 가변인자(...)를 통해 전달되는 두 번째 인자, 세 번째 인자 등이 첫 번째 인자인 FUN 함수의 인자로 적용됩니다.

```
mapply(FUN, ..., MoreArgs=NULL, SIMPLIFY=TRUE,
       USE.NAMES=TRUE)
```

구문에서...

- FUN : 적용할 함수 이름입니다.
- ... : FUN의 인자로 전달한 값 들입니다.
- MoreArgs : FUM 함수에 전달할 다른 인자 목록입니다.
- SIMPLIFY : TRUE(기본값) 이면 연산 결과를 벡터, 행렬 등으로 반환합니다. FALSE 이면 연산 결과를 리스트로 반환합니다.
- USE.NAMES : TRUE(기본값) 이면 이름 속성도 반환합니다. FALSE이면 이름 속성 없이 반환합니다.

mapply

고객들의 직업과 소득을 임의로 샘플링 하여 데이터 프레임으로 만들었습니다.

```
> job <- c(3, 3, 5, 2, 2, 3, 5, 3, 4, 4, 6, 3)
> income <- c(4879, 6509, 4183, 0, 3894, 0, 3611, 6454, 4975, 8780, 0, 4362)
> cust <- data.frame(job, income)
```

	job	income
1	3	4879
2	3	6509
3	5	4183
4	2	0
5	2	3894
6	3	0
7	5	3611
8	3	6454
9	4	4975
10	4	8780
11	6	0
12	3	4362

다음 함수는 고객의 소득이 0인 경우 미리 계산된 고객들의 직업별 평균 소득으로 대체하는 함수입니다. zero2mean 함수는 인수를 두 개 필요합니다. 인수 job은 고객의 직업이며, income은 고객의 소득입니다.

```
> zero2mean <- function(job, income) {
+   if(income==0) {
+     return (income.avg[job+1])
+   }else {
+     return (income)
+   }
+ }
```

다음 코드는 zero2mean 함수를 모든 고객 데이터에 적용시키는 예입니다.

```
> cust$income.new <- mapply(zero2mean, cust$job, cust$income)
```

	job	income	income.new
1	3	4879	4879
2	3	6509	6509
3	5	4183	4183
4	2	0	3806
5	2	3894	3894
6	3	0	3990
7	5	3611	3611
8	3	6454	6454
9	4	4975	4975
10	4	8780	8780
11	6	0	3556
12	3	4362	4362

jobid	income.avg
0	862
1	0
2	3806
3	3990
4	3891
5	3359
6	3556
7	2199
8	227

tapply

tapply()는 그룹별 처리를 위한 apply 함수입니다. 데이터가 주어졌을 때 각 데이터가 속한 그룹별로 주어진 함수를 수행합니다.

```
tapply(X, INDEX, FUN=NULL, ...,
       default=NA, simplify=TRUE)
```

구문에서...

- X : 대상 리스트 객체입니다.
- INDEX : X와 같은 길이의 하나 이상의 범주형변수(factor) 목록입니다. as.factor() 함수에 의해 범주형변수 타입으로 강제 형변환됩니다.
- FUN : 적용할 함수입니다. NULL. +, % * % 등의 경우, 이름을 역 인용부호(`) 또는 인용부호(')로 묶어야 합니다. FUN이 NULL이면 tapply는 벡터를 리턴합니다.
- ... : FUN의 인자로 전달한 값 들입니다.
- default : 기본값은 NA이며 결측값일 경우 출력될 값을 지정합니다.
- SIMPLIFY : TRUE(기본값)이면 FUN이 항상 스칼라를 반환하면 tapply는 스칼라 모드의 배열을 반환합니다. FALSE이면, tapply은 항상 "list"모드의 배열을 리턴합니다. 즉, dim 속성이 있는 목록입니다.

tapply

다음 코드는 iris 데이터셋에서 종별 꽃받침의 길이 평균을 구합니다.

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

다음 코드는 iris 데이터셋에서 종별 꽃받침의 폭(너비)의 평균을 구합니다.

```
> tapply(iris$Sepal.Width, iris$Species, mean)
    setosa versicolor  virginica
     3.428      2.770      2.974
```

tapply의 default 속성

default 속성에 대해 알아보기 위해 앞에서 사용했던 고객의 직업과 소득 정보를 이용해 직업별 평균 소득을 계산해 보겠습니다. 이를 위해 먼저 데이터를 준비합니다.

```
> job <- c(3, 3, 5, 2, 2, 3, 5, 3, 4, 4, 6, 3)
> job <- factor(job, levels=c(0:8))
> str(job)
 Factor w/ 9 levels "0","1","2","3",..: 4 4 6 3 3 4 6 4 5 5 ...
> income <- c(4879, 6509, 4183, 0, 3894, 0, 3611, 6454, 4975, 8780, 0, 4362)
```

직업별로 평균을 구해야 하기 때문에 job을 범주형변수로 정의합니다. 이때 범주형변수의 레벨은 직업코드 전체인 0부터 8까지로 합니다.

다음 코드는 default 속성이 어떤 역할을 하는지 보여줍니다. job 범주형변수의 레벨에는 있지만 income 데이터에 존재하지 않는 직업들은 NA로 출력됩니다. default 속성은 NA로 출력되어야 할 값을 다른 값으로 바꿔 출력되게 할 수 있습니다. 다음 코드는 실제 평균이 0인 경우와 결측값을 구분하기 위해 결측값이 -1로 출력되게 한 것입니다.

```
> tapply(income, job, mean)
     0      1      2      3      4      5      6      7      8
    NA     NA 1947.0 4440.8 6877.5 3897.0    0.0     NA     NA

> tapply(income, job, mean, default=-1)
     0      1      2      3      4      5      6      7      8
  -1.0   -1.0 1947.0 4440.8 6877.5 3897.0    0.0   -1.0   -1.0
```

by

by()는 함수는 데이터 프레임에 적용되는 tapply를 위한 함수입니다.

```
by(data, INDICES, FUN, ..., simplify=TRUE)
```

구문에서...

- data : R 객체입니다. 일반적으로 데이터 프레임이거나 행렬입니다.
- INDICES : 팩터 또는 팩터 리스트입니다.
- FUN : 데이터의 서브세트에 적용되는 함수입니다.
- ... : FUN의 인자로 전달한 값 들입니다.
- SIMPLIFY : TRUE(기본값)이면 FUN이 항상 스칼라를 반환하면 by는 스칼라 모드의 배열을 반환합니다. FALSE이면, by는 항상 "list"모드의 배열을 리턴합니다. 즉, dim 속성이 있는 목록입니다.

tapply vs. by

tapply() 함수는 한 번에 여러 열에 대하여 집계 연산을 수행할 수 없습니다.

```
> tapply(iris[,1:4], iris[,"Species"], sum)
Error in tapply(iris[, 1:4], iris[, "Species"], sum) :
  인자들은 반드시 같은 길이를 가져야 합니다
```

by() 함수는 데이터 프레임의 여러 열에 함수를 적용시킬 수 있습니다.

```
> by(iris[,1:4], iris[,"Species"], sum)
iris[, "Species"]: setosa
[1] 507.1
------------------------------------------------
iris[, "Species"]: versicolor
[1] 714.6
------------------------------------------------
iris[, "Species"]: virginica
[1] 857
```

doBy 패키지

doBy 패키지에는 다양한 유틸리티 기능이 포함되어 있습니다. 이 패키지는 원래 SAS 시스템의 PROC 요약과 같은 그룹 별 요약 통계를 계산할 필요성에서 비롯되었지만 현재 패키지에는 많은 유틸리티가 포함되어 있습니다.

doBy 패키지는 install.packages("doBy")명령으로 설치할 수 있습니다.

```
> install.packages("doBy")
trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.4/doBy_4.5-15.zip'
Content type 'application/zip' length 3353650 bytes (3.2 MB)
downloaded 3.2 MB

package 'doBy' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
    C:\Users\COM\AppData\Local\Temp\Rtmp2ZAHIr\downloaded_packages
```

패키지를 사용하기 위해서는 로드해야 합니다. 패키지 로드는 require() 또는 library() 함수를 사용합니다.

```
> require(doBy)
필요한 패키지를 로딩중입니다: doBy
```

summaryBy

summaryBy() 함수는 그룹별로 그룹을 특징짓는 통계적 요약 값 계산에 사용됩니다. 예를 들면 두 요인 A와 B의 각 조합에 대한 x와 y의 평균과 분산 등의 계산에 사용됩니다.

```
summaryBy(formula, data=parent.frame(), id=NULL,
          FUN=mean, keep.names=FALSE, p2d=FALSE,
          order=TRUE, full.dimension=FALSE,
          var.names=NULL, fun.names=NULL, ...)
```

구문에서...

- formula : 포뮬러를 지정합니다. 포뮬러에 대한 자세한 내용은 다음 3.4절에서 설명됩니다.
- data : 데이터 프레임입니다.
- id : 데이터가 그룹화되지 않지만 출력에 나타나야하는 변수를 지정하는 포뮬러입니다.

교재에 더 많은 구문에 대한 설명이 있습니다.

summaryBy

다음 코드는 iris 데이터의 종별 꽃받침의 폭과 길이의 통계적 요약을 출력합니다. 함수를 지정하지 않으면 평균(mean) 값을 출력합니다.

```
> 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
```

다음 코드는 iris 데이터의 종별 꽃받침의 폭과 길이의 합(sum)을 출력합니다.

```
> summaryBy(Sepal.Width + Sepal.Length ~ Species, iris, FUN=sum)
     Species Sepal.Width.sum Sepal.Length.sum
1     setosa           171.4            250.3
2 versicolor           138.5            296.8
3  virginica           148.7            329.4
```

다음 코드는 FUN에 여러 개 함수를 지정하는 예입니다.

```
> summaryBy(Sepal.Width + Sepal.Length ~ Species, iris, FUN=c(mean,sum))
     Species Sepal.Width.mean  Sepal.Length.mean Sepal.Width.sum  Sepal.Length.sum
1     setosa            3.428             5.006           171.4             250.3
2 versicolor            2.770             5.936           138.5             296.8
3  virginica            2.974             6.588           148.7             329.4
```

OrderBy

orderBy()는 데이터 프레임의 특정 변수로 데이터 프레임의 행을 정렬(ordering, sorting)합니다. 이 함수는 본질적으로 order() 함수에 대한 래퍼입니다. 중요한 차이점은 정렬하기 위한 변수가 모델 포뮬러에 의해 제공 될 수 있다는 것입니다.

```
orderBy(formula, data)
```

구문에서...

- formula : 포뮬러를 지정합니다.
- data : 데이터 프레임입니다.

```
> head(orderBy(~Species+Sepal.Length, data=iris))
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
14          4.3         3.0          1.1         0.1  setosa
9           4.4         2.9          1.4         0.2  setosa
39          4.4         3.0          1.3         0.2  setosa
43          4.4         3.2          1.3         0.2  setosa
42          4.5         2.3          1.3         0.3  setosa
4           4.6         3.1          1.5         0.2  setosa
```

sampleBy

sampleBy() 함수에 의해 데이터 프레임은 포뮬러의 변수에 따라 분할되고 각각 분할된 그룹에서 특정 비율의 샘플이 추출됩니다.

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

구문에서...

- formula : 포뮬러를 지정합니다. 포뮬러에 대한 자세한 내용은 다음 3.4절에서 설명됩니다.
- frac : 추출할 샘플의 비율입니다. 기본값은 0.1(10%)입니다.
- replace : 복원추출 여부를 설정합니다. 기본값은 FALSE이며, 이 경우 비복원추출 입니다. 비복원추출은 한번 뽑은 것은 다시 뽑을 수 없는 추출입니다. TRUE 이면 복원추출이고 한 번 뽑은 데이터를 다시 뽑을 수 있습니다.
- data : 데이터 프레임입니다.

계통 추출

- Systematic : 계통추출을 사용할 지 여부를 결정합니다. 계통추출은 체계적 표집(systematic sampling)이라고도 하며 첫 번째 요소를 선정한 후 그 샘플로부터 동일한 간격에 있는 데이터를 샘플로 추출하는 방법입니다. 기본값은 임의추출(FALSE)입니다. 계통추출(TRUE)일 경우 frac=.1 이면 1/.1 즉 처음부터 각 열 번째(1, 11, 21, 31, 41, ...) 데이터가 추출되고, frac=0.2 이면 1/.2 즉, 처음부터 각 다섯 번째(1, 6, 11, 16, 21, ...) 데이터가 추출됩니다. 계통추출법은 만약 표본이 추출되기 전 요소들의 목록이 무작위로 되어 있지 않고 주기성(periodicity)을 띄고 있다면, 계통추출법을 통해 추출된 표본은 매우 어긋난 표본이 될 수 있으며 모집단을 전혀 반영하지 못하게 됩니다.

비복원, 임의 추출

다음 코드는 iris 데이터에서 종별로 각 10% 씩 데이터를 샘플링 합니다. 이는 비복원, 임의추출 방법입니다. 계통추출이 아니라면 실행 결과가 이 책의 내용과 다를 수 있습니다.

```
> sampleBy(~Species, data=iris, frac=0.1)
             Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
setosa.5              5.0         3.6          1.4         0.2     setosa
setosa.12             4.8         3.4          1.6         0.2     setosa
setosa.36             5.0         3.2          1.2         0.2     setosa
setosa.45             5.1         3.8          1.9         0.4     setosa
setosa.47             5.1         3.8          1.6         0.2     setosa
versicolor.58         4.9         2.4          3.3         1.0 versicolor
versicolor.66         6.7         3.1          4.4         1.4 versicolor
versicolor.76         6.6         3.0          4.4         1.4 versicolor
versicolor.80         5.7         2.6          3.5         1.0 versicolor
versicolor.87         6.7         3.1          4.7         1.5 versicolor
virginica.103         7.1         3.0          5.9         2.1  virginica
virginica.114         5.7         2.5          5.0         2.0  virginica
virginica.123         7.7         2.8          6.7         2.0  virginica
virginica.136         7.7         3.0          6.1         2.3  virginica
virginica.148         6.5         3.0          5.2         2.0  virginica
```

복원, 임의 추출

다음 코드는 복원추출입니다. versicolor 데이터가 86번째 데이터가 두 번 샘플링 된 것을 확인할 수 있습니다.

```
> sampleBy(~Species, data=iris, frac=0.1, replace=TRUE)
               Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
setosa.2                4.9         3.0          1.4         0.2     setosa
setosa.12               4.8         3.4          1.6         0.2     setosa
setosa.16               5.7         4.4          1.5         0.4     setosa
setosa.23               4.6         3.6          1.0         0.2     setosa
setosa.47               5.1         3.8          1.6         0.2     setosa
versicolor.55           6.5         2.8          4.6         1.5 versicolor
versicolor.57           6.3         3.3          4.7         1.6 versicolor
versicolor.86           6.0         3.4          4.5         1.6 versicolor
versicolor.86.1         6.0         3.4          4.5         1.6 versicolor
versicolor.100          5.7         2.8          4.1         1.3 versicolor
virginica.112           6.4         2.7          5.3         1.9  virginica
virginica.119           7.7         2.6          6.9         2.3  virginica
virginica.130           7.2         3.0          5.8         1.6  virginica
virginica.136           7.7         3.0          6.1         2.3  virginica
virginica.139           6.0         3.0          4.8         1.8  virginica
```

비복원, 계통추출

다음 코드는 비복원추출이며, 계통추출입니다.

```
> sampleBy(~Species, data=iris, frac=0.1, systematic=TRUE)
             Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
setosa.1              5.1         3.5          1.4         0.2     setosa
setosa.11             5.4         3.7          1.5         0.2     setosa
setosa.21             5.4         3.4          1.7         0.2     setosa
setosa.31             4.8         3.1          1.6         0.2     setosa
setosa.41             5.0         3.5          1.3         0.3     setosa
versicolor.51         7.0         3.2          4.7         1.4 versicolor
versicolor.61         5.0         2.0          3.5         1.0 versicolor
versicolor.71         5.9         3.2          4.8         1.8 versicolor
versicolor.81         5.5         2.4          3.8         1.1 versicolor
versicolor.91         5.5         2.6          4.4         1.2 versicolor
virginica.101         6.3         3.3          6.0         2.5  virginica
virginica.111         6.5         3.2          5.1         2.0  virginica
virginica.121         6.9         3.2          5.7         2.3  virginica
virginica.131         7.4         2.8          6.1         1.9  virginica
virginica.141         6.7         3.1          5.6         2.4  virginica
```

포뮬러

R 함수, 특히 선형 회귀를 맞추기 위한 lm()과 물류 회귀를 맞추기 위한 glm()들은 formula(포뮬러) 구문을 사용하여 통계 모델의 형식을 지정합니다. 회귀 분석을 위한 함수들만 아니라 많은 함수들이 포뮬러를 인수로 갖습니다. 포뮬러 형식을 잘 알아두기 바랍니다.

이러한 포뮬러의 기본 형식은 다음과 같습니다.

응답 변수 ~ 예측 변수

물결표(~)는 "함수로 모델링 되었습니다"라고 읽힙니다. 보통 회귀 분석에서 응답 변수는 종속 변수라 부르고, 예측 변수는 독립 변수라 부릅니다. 공식화 된 기본 회귀 분석 식은 다음과 같습니다.

$$Y \sim X$$

그러므로 X에 Y를 회귀하는 선형 모델을 다음과 같이 R 코드로 작성할 수 있습니다.

```
fit <- lm(Y ~ X)
```

포뮬러 기호

기호	의미	예
+	이 변수를 포함합니다.	+Z
-	이 변수를 제외합니다.	-Z
:	이 변수들 사이의 상호 작용(interaction)을 포함합니다.	X:Z
*	이 변수들과 그것들을 조합한 모든 상호 작용들을 포함합니다.	X*Y
^	예는 모든 상호 작용을 최대 세 가지 방법으로 포함합니다.	(X + Z + W)^3
I	수식으로 구성된 새 변수를 추가합니다.	I(*expr*)
-1	절편(intercept)을 삭제합니다. +0과 같습니다.	X - 1
.	데이터에서 종속변수를 제외한 모든 변수를 독립변수로 사용합니다.	Y ~ .

포뮬러 예


```
> plot(cars)
> abline(lm(dist~speed, cars))
> abline(lm(dist~speed-1, cars), lty="dotted")
```

다음 그림은 cars 데이터로 산점도 그래프를 그리고 그 위에 절편을 포함한 회귀 직선(실선)과, 절편을 포함하지 않는 회귀 직선(점선)을 표시한 것입니다.

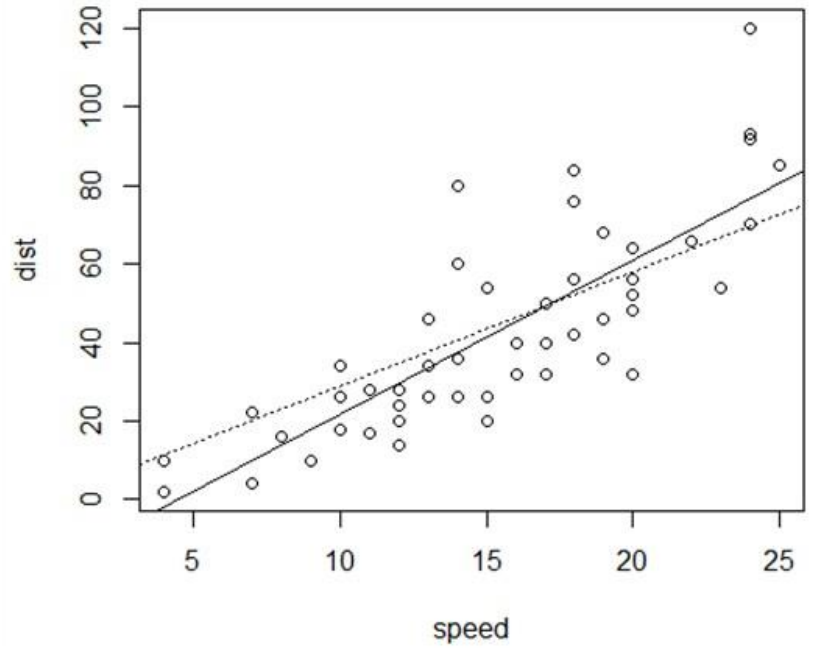

그림 6. 절편을 제외시킨 포뮬러로 구한 회귀 직선

split

split()은 벡터 또는 데이터 프레임 x의 데이터를 범주형변수(팩터) f로 정의 된 그룹으로 나눕니다. 대체 양식은 이러한 구분에 해당하는 값을 대체합니다.

```
split(x, f, drop = FALSE, ...)
```

구문에서...

- x : 그룹으로 나눌 값을 포함한 벡터 또는 데이터 프레임입니다.
- f : 그룹화를 정의하기 위한 팩터 또는 팩터를 포함하는 리스트입니다.
- drop : 만일 f가 팩터 또는 리스트일 경우 발생하지 않는 레벨을 삭제해야 하는지를 나타내는 논리값입니다. 기본값은 FALSE입니다.
- ... : 함수에 전달한 인수들입니다.

split 예

다음 코드는 iris 데이터를 종(Species) 별로 분리하는 코드입니다.

```
> iris.species <- split(iris, iris$Species)
> str(iris.species)
List of 3
 $ setosa    :'data.frame':    50 obs. of  5 variables:
  ..$ Sepal.Length: num [1:50] 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
  ..$ Sepal.Width : num [1:50] 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
  ..$ Petal.Length: num [1:50] 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
  ..$ Petal.Width : num [1:50] 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
  ..$ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1111111111...
 $ versicolor:'data.frame':    50 obs. of  5 variables:
  ..$ Sepal.Length: num [1:50] 7 6.4 6.9 5.5 6.5 5.7 6.3 4.9 6.6 5.2 ...
  ..$ Sepal.Width : num [1:50] 3.2 3.2 3.1 2.3 2.8 2.8 3.3 2.4 2.9 2.7 ...
  ..$ Petal.Length: num [1:50] 4.7 4.5 4.9 4 4.6 4.5 4.7 3.3 4.6 3.9 ...
  ..$ Petal.Width : num [1:50] 1.4 1.5 1.5 1.3 1.5 1.3 1.6 1 1.3 1.4 ...
  ..$ Species     : Factor w/ 3 levels "setosa","versicolor",..: 2222222222...
 $ virginica :'data.frame':    50 obs. of  5 variables:
  ..$ Sepal.Length: num [1:50] 6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3 6.7 7.2 ...
  ..$ Sepal.Width : num [1:50] 3.3 2.7 3 2.9 3 3 2.5 2.9 2.5 3.6 ...
  ..$ Petal.Length: num [1:50] 6 5.1 5.9 5.6 5.8 6.6 4.5 6.3 5.8 6.1 ...
  ..$ Petal.Width : num [1:50] 2.5 1.9 2.1 1.8 2.2 2.1 1.7 1.8 1.8 2.5 ...
  ..$ Species     : Factor w/ 3 levels "setosa","versicolor",..: 3333333333...
```

split 예

다음 코드는 꽃받침의 길이가 5보다 큰 데이터와 그렇지 않은 데이터로 나눕니다.

```
> iris.sepal.length <- split(iris, iris$Sepal.Length > 5)
> str(iris.sepal.length)
List of 2
 $ FALSE:'data.frame':  32 obs. of  5 variables:
  ..$ Sepal.Length: num [1:32] 4.9 4.7 4.6 5 4.6 5 4.4 4.9 4.8 4.8 ...
  ..$ Sepal.Width : num [1:32] 3 3.2 3.1 3.6 3.4 3.4 2.9 3.1 3.4 3 ...
  ..$ Petal.Length: num [1:32] 1.4 1.3 1.5 1.4 1.4 1.5 1.4 1.5 1.6 1.4 ...
  ..$ Petal.Width : num [1:32] 0.2 0.2 0.2 0.2 0.3 0.2 0.2 0.1 0.2 0.1 ...
  ..$ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1111111111...
 $ TRUE :'data.frame':  118 obs. of  5 variables:
  ..$ Sepal.Length: num [1:118] 5.1 5.4 5.4 5.8 5.7 5.4 5.1 5.7 5.1 5.4 ...
  ..$ Sepal.Width : num [1:118] 3.5 3.9 3.7 4 4.4 3.9 3.5 3.8 3.8 3.4 ...
  ..$ Petal.Length: num [1:118] 1.4 1.7 1.5 1.2 1.5 1.3 1.4 1.7 1.5 1.7 ...
  ..$ Petal.Width : num [1:118] 0.2 0.4 0.2 0.2 0.4 0.4 0.3 0.3 0.3 0.2 ...
  ..$ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1111111111...
```

split에 의한 서브 데이터의 구분

그룹화하기 위해 조건식을 사용할 경우 그룹화되는 서브 데이터의 변수 이름이 TRUE, FALSE로 만들어 집니다. 그런데 TRUE, FALSE는 논리형 값으로 정의되어 있기 때문에 직접 변수 이름으로 사용할 수 없습니다. 이럴 경우에는 TRUE, FALSE를 역따옴표(`) 또는 인용부호(' '또는 " ")로 열 이름을 둘러싸면 됩니다.

```
> head(iris.sepal.length$'TRUE')
   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
15          5.8         4.0          1.2         0.2  setosa
16          5.7         4.4          1.5         0.4  setosa
17          5.4         3.9          1.3         0.4  setosa
```

```
> head(iris.sepal.length$'FALSE')
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
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
7          4.6         3.4          1.4         0.3  setosa
8          5.0         3.4          1.5         0.2  setosa
```

subset

subset()은 조건을 만족하는 벡터, 행렬 또는 데이터 프레임의 하위 집합을 반환합니다.

```
subset(x, subset, select, drop=FALSE, ...)
```

구문에서...

- x : subset 될 객체입니다.
- subset : 유지할 변수 또는 행을 나타내는 논리식입니다. 결측값은 false로 간주됩니다.
- select : 데이터 프레임에서 선택할 열을 나타내는 표현식입니다.
- drop : drop 인수는 행렬 및 데이터 프레임의 인덱싱 방법으로 전달됩니다. 행렬의 기본값은 인덱싱의 기본값과 다릅니다. 요소는 부분 집합 후에 빈 레벨을 가질 수 있습니다. 사용하지 않은 레벨은 자동으로 제거되지 않습니다.
- ... : 다른 함수로 또는 다른 함수로부터 전달 될 추가 인수입니다.

subset 예


```
> iris.species.setosa <- subset(iris, Species=="setosa")
> str(iris.species.setosa)
'data.frame': 50 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
> iris.setosa.sepal <- subset(iris,
+                            select=c(Sepal.Length, Sepal.Width, Species))
> str(iris.setosa.sepal)
'data.frame': 150 obs. of  3 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
> iris.sub <- subset(iris, select=-c(Sepal.Length, Sepal.Width, Species))
> str(iris.sub)
'data.frame': 150 obs. of  2 variables:
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
```

데이터셋 결합 : rbind, cbind

rbind()와 cbind()는 데이터셋을 결합합니다.

```
cbind(..., deparse.level=1)
rbind(..., deparse.level=1)
```

구문에서...

- ... : 벡터들 또는 행렬들 입니다.
- deparse.level : 행렬이 아닌 인수의 경우 행 또는 열의 이름을 제어하는 정수입니다. 0일 경우 결합할 데이터의 파라미터 이름을 이용하여 레이블을 지정하며, 1(기본값)은 0의 레이블 지정에 추가로 파라미터 이름이 없을 경우 변수의 이름을 이용하여 레이블을 지정하고, 2는 0과 1의 레이블 지정에 추가로 표현식을 이용하여 레이블로 추가해 줍니다.

deparse.level


```
> dd <- 10
> rbind(1:4, c=2, "a++"=10, dd, deparse.level=0)
    [,1] [,2] [,3] [,4]
       1    2    3    4
c      2    2    2    2
a++   10   10   10   10
      10   10   10   10

> rbind(1:4, c=2, "a++"=10, dd, deparse.level=1)
    [,1] [,2] [,3] [,4]
       1    2    3    4
c      2    2    2    2
a++   10   10   10   10
dd    10   10   10   10

> rbind(1:4, c=2, "a++"=10, dd, deparse.level=2)
    [,1] [,2] [,3] [,4]
1:4    1    2    3    4
c      2    2    2    2
a++   10   10   10   10
dd    10   10   10   10
```

cbind

cbind()는 열 단위로 데이터를 합쳐줍니다.

벡터 데이터를 합칠 때 데이터의 길이가 다를 경우 길이가 짧은 벡터 데이터는 재활용 됩니다.

```
> m <- cbind(1, 1:5)
> m
     [,1] [,2]
[1,]    1    1
[2,]    1    2
[3,]    1    3
[4,]    1    4
[5,]    1    5
```

[,]는 행 또는 열 정보를 조회하는데 사용합니다. 이를 이용하면 합치는 열을 중간에 포함시킬 수 있습니다.

```
> m <- cbind(m, 8:12)[, c(1, 3, 2)]
> m
     [,1] [,2] [,3]
[1,]    1    8    1
[2,]    1    9    2
[3,]    1   10    3
[4,]    1   11    4
[5,]    1   12    5
```

rbind

rbind()는 행 단위로 데이터를 합쳐줍니다.

다음 코드는 데이터를 분리합니다.

```
> iris.setosa <- subset(iris, Species=="setosa")
> iris.versicolor <- iris[51:100, ]
> iris.virginica <- split(iris, iris$Species)[[3]]
```

다음 코드는 행 단위로 데이터를 합칩니다.

```
> iris.rbind <- rbind(iris.setosa, iris.versicolor, iris.virginica)
> head(iris.rbind)
  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
```

merge

merge()는 공통된 열을 기준으로 데이터를 병합해 줍니다.

```
merge(x, y, by=intersect(names(x), names(y)),
      by.x=by, by.y=by, all=FALSE, all.x=all, all.y=all,
      sort=TRUE, suffixes=c(".x",".y"),
      incomparables=NULL, ...)
```

구문에서...

- x, y : 병합할 데이터 프레임 또는 객체입니다.
- by : 병합에 사용할 기준이 되는 열을 지정합니다. 기본값은 병합할 두 데이터의 교집합입니다.
- by.x, by.y : x 데이터에서 기준이 되는 열과, y 데이터에서 기준이 되는 열을 지정합니다. 기본값은 by 인수와 같습니다.
- all : TRUE 이면 기준이 되는 열에 값을 가지고 있지 않는 경우에도 행을 생성해 줍니다. 즉, 모든 행이 병합에 사용됩니다. 기본값은 FALSE 이며 기준이 되는 열의 값이 x와 y에 모두 있는 데이터만 병합됩니다.
- all.x, all.y : all.x는 x의 모든 행은 병합에 사용되고, all.y는 y의 모든 행이 병합에 사용되도록 합니다. 기본값은 all 인수와 같습니다.

merge 예

다음 코드는 두 데이터를 병합하는 예입니다. 기본적으로 두 데이터의 기준열에 모두 있는 경우에만 병합됩니다.

```
> student.merged <- merge(studentData, studentMathData)
> student.merged
  student.name student.eng student.kor student.math
1         Eric          85          90           95
2          Jin          60          70           95
3          Kei          95          90           90
```

다음 코드는 all=TRUE 인수를 포함시켜 모든 데이터가 병합되도록 한 예입니다.

```
> student.merged <- merge(studentData, studentMathData, all=TRUE)
> student.merged
  student.name student.eng student.kor student.math
1          Den          90          85           NA
2         Eric          85          90           95
3          Jin          60          70           95
4          Kei          95          90           90
```

sOrt

sort()는 벡터 또는 팩터 데이터를 오름차순 또는 내림차순으로 정렬합니다. sort()는 값을 정렬한 그 결과를 반환할 뿐 인자로 받은 벡터 자체를 변경하지 않습니다.

```
sort(x, partial=NULL, decreasing=FALSE, na.last=NA,
       method=c("auto", "shell", "quick", "radix"),
       index.return=FALSE)
```

구문에서...

- x : 정렬할 데이터입니다.
- partial : NULL 또는 부분정렬 할 데이터입니다.
- decreasing : FALSE(기본값)일 경우 오름차순으로 정렬합니다. TRUE일 경우 내림차순으로 정렬합니다.
- na.last : 결측값들의 처리를 제어하기 위한 것입니다. TRUE 일 경우 데이터에 누락 된 값을 마지막에 놓고 FALSE 일 경우 먼저 놓습니다. NA이면 제거됩니다.
- method : 사용 된 알고리즘을 지정하는 문자열입니다. 부분 정렬에는 사용할 수 없습니다. 약어로 표시 할 수 있습니다. 알고리즘에 따라 실행 시간이 다를 수 있습니다.
- index.return : 정렬한 데이터의 색인이 반환되어야 하는지를 나타내는 논리값입니다. method가 "radix"일 때는 모든 na.last 모드와 모든 데이터타입을 지원하고, 팩터가 아닌 데이터를 전체 정렬하고 na.last=NA(기본값) 일 때 다른 method가 지원합니다.

sort 예


```
> head(sort(iris$Sepal.Length))
[1] 4.3 4.4 4.4 4.4 4.5 4.6

> head(sort(iris$Sepal.Length, decreasing=TRUE))
[1] 7.9 7.7 7.7 7.7 7.7 7.6
```

```
> sort(iris$Sepal.Length, index.return=TRUE)
$x
  [1] 4.3 4.4 4.4 4.4 4.5 4.6 4.6 4.6 4.6 4.7 4.7 4.8 4.8 4.8 4.8 4.8 4.9 4.9
 [19] 4.9 4.9 4.9 4.9 5.0 5.0 5.0 5.0 5.0 5.0 5.0 5.0 5.0 5.0 5.1 5.1 5.1 5.1
 [37] 5.1 5.1 5.1 5.1 5.1 5.2 5.2 5.2 5.2 5.3 5.4 5.4 5.4 5.4 5.4 5.4 5.5 5.5
 [55] 5.5 5.5 5.5 5.5 5.5 5.6 5.6 5.6 5.6 5.6 5.6 5.7 5.7 5.7 5.7 5.7 5.7 5.7
 [73] 5.7 5.8 5.8 5.8 5.8 5.8 5.8 5.8 5.9 5.9 5.9 6.0 6.0 6.0 6.0 6.0 6.0 6.1
 [91] 6.1 6.1 6.1 6.1 6.1 6.2 6.2 6.2 6.2 6.3 6.3 6.3 6.3 6.3 6.3 6.3 6.3 6.3
[109] 6.4 6.4 6.4 6.4 6.4 6.4 6.4 6.5 6.5 6.5 6.5 6.5 6.6 6.6 6.7 6.7 6.7 6.7
[127] 6.7 6.7 6.7 6.7 6.8 6.8 6.8 6.9 6.9 6.9 6.9 7.0 7.1 7.2 7.2 7.2 7.3 7.4
[145] 7.6 7.7 7.7 7.7 7.7 7.9

$ix
  [1]  14   9  39  43  42   4   7  23  48   3  30  12  13  25  31  46   2  10
 [19]  35  38  58 107   5   8  26  27  36  41  44  50  61  94   1  18  20  22
 [37]  24  40  45  47  99  28  29  33  60  49   6  11  17  21  32  85  34  37
 [55]  54  81  82  90  91  65  67  70  89  95 122  16  19  56  80  96  97 100
 [73] 114  15  68  83  93 102 115 143  62  71 150  63  79  84  86 120 139  64
 [91]  72  74  92 128 135  69  98 127 149  57  73  88 101 104 124 134 137 147
[109]  52  75 112 116 129 133 138  55 105 111 117 148  59  76  66  78  87 109
[127] 125 141 145 146  77 113 144  53 121 140 142  51 103 110 126 130 108 131
[145] 106 118 119 123 136 132
```

정렬 방법에 따른 실행 속도

다음 코드는 method 인자에 따라 실행속도가 다름을 보이기 위한 예입니다. rnorm() 함수[22]는 평균(mean)이 0이고, 표준편차(sd)가 1인 정규분포를 따르는 데이터 n개를 만드는 함수입니다.

```
> x <- rnorm(1e7) #mean 0, sd 1

> system.time(x1 <- sort(x, method = "shell"))
 사용자  시스템 elapsed
   1.34    0.04    1.39

> system.time(x2 <- sort(x, method = "quick"))
 사용자  시스템 elapsed
   0.88    0.02    0.90

> system.time(x3 <- sort(x, method = "radix"))
 사용자  시스템 elapsed
   0.95    0.07    1.03
```

Order

order()는 주어진 인자를 정렬하기 위한 각 요소의 색인을 반환합니다. 큰 수부터 정렬한 결과를 얻고 싶다면 값에 -1 을 곱합니다.

```
order(..., na.last=TRUE, decreasing=FALSE,
      method=c("auto", "shell", "radix"))
```

구문에서...

- ... : 정렬할 데이터입니다.
- na.last : 결측값들의 처리를 제어하기 위한 것입니다. TRUE 일 경우 데이터에 누락 된 값을 마지막에 놓고 FALSE 일 경우 먼저 놓습니다. NA이면 결측값들은 제거됩니다.
- decreasing : FALSE(기본값)일 경우 오름차순으로 정렬합니다. TRUE일 경우 내림차순으로 정렬합니다.
- method : 사용 된 알고리즘을 지정하는 문자열입니다. 부분 정렬에는 사용할 수 없습니다. 약어로 표시 할 수 있습니다. 알고리즘에 따라 실행 시간이 다를 수 있습니다.

Order 예

다음 코드는 iris 데이터에서 꽃받침(Sepal)의 폭(Width)를 내림차순으로 정렬하고 그의 색인을 출력합니다.

```
> order(iris$Sepal.Width, decreasing=TRUE)
  [1]  16  34  33  15   6  17  19  20  45  47 118 132  11  22  49   5  23  38
 [19] 110   1  18  28  37  41  44   7   8  12  21  25  27  29  32  40  86 137
 [37] 149  24  50  57 101 125 145   3  30  36  43  48  51  52  71 111 116 121
 [55] 126 144   4  10  31  35  53  66  87 138 140 141 142   2  13  14  26  39
 [73]  46  62  67  76  78  85  89  92  96 103 105 106 113 117 128 130 136 139
 [91] 146 148 150   9  59  64  65  75  79  97  98 104 108  55  56  72  74  77
[109] 100 115 122 123 127 129 131 133 134  60  68  83  84  95 102 112 124 143
[127]  80  91  93 119 135  70  73  90  99 107 109 114 147  58  81  82  42  54
[145]  88  94  63  69 120  61
```

Order를 이용한 정렬

order를 이용하면 데이터 프레임의 데이터를 특정 열(변수)을 기준으로 정렬할 수 있습니다. 다음 코드는 Sepal.Length(꽃받침 길이)를 기준으로 내림차순으로 정렬하고, 만일 꽃 받침의 길이가 같을 경우 Sepal.Width(꽃받침 폭)을 기준으로 오름차순으로 정렬합니다.

```
> temp <- iris[order(-iris$Sepal.Length, iris$Sepal.Width),]
> head(temp)
    Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
132          7.9         3.8          6.4         2.0 virginica
119          7.7         2.6          6.9         2.3 virginica
123          7.7         2.8          6.7         2.0 virginica
136          7.7         3.0          6.1         2.3 virginica
118          7.7         3.8          6.7         2.2 virginica
106          7.6         3.0          6.6         2.1 virginica
```

with, within

with()는 데이터로 구성된 로컬 환경에서 expr을 평가하는 함수입니다. 환경은 호출자의 환경을 부모로 가집니다. 데이터 프레임 또는 리스트의 필드를 데이터 이름 없이 접근할 수 있기 때문에 모델링 함수 호출을 단순화 할 때 유용합니다.

within()은 expr을 평가 한 후 환경을 검사하고 데이터 복사본에 해당 수정 사항을 적용한다는 점을 제외하면 with()와 비슷합니다.

```
with(data, expr, ...)
within(data, expr, ...)
```

구문에[서...

- data : 데이터입니다.
- expr : 평가할 표현식입니다. 데이터프레임 또는 리스트의 data의 이름을 생략하고 필드를 이용하여 표현식을 작성할 수 있습니다.
- ... : 함수에 전달될 인수입니다.

with, within 사용 예


```
> iris.with <- with(iris.temp, {
+       mps <- sapply (split(Sepal.Length, Species), median, na.rm=TRUE)
+       Sepal.Length <- ifelse(is.na(Sepal.Length), mps[Species], Sepal.Length)
+ })
> str(iris.with)
 num [1:150] 5 4.9 5 4.6 5 5.4 4.6 5 4.4 4.9 ...
```

```
> iris.within <- within(iris.temp, {
+   mps <- sapply (split(Sepal.Length, Species), median, na.rm=TRUE)
+   Sepal.Length <- ifelse(is.na(Sepal.Length), mps[Species], Sepal.Length)
+ })
> str(iris.within)
'data.frame': 150 obs. of  6 variables:
 $ Sepal.Length: num  5 4.9 5 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1111111111...
 $ mps         : num  5 5.9 6.5 5 5.9 6.5 5 5.9 6.5 5 ...
```

attach, detach

attach()는 데이터 객체가 R 검색 경로에 첨부됩니다. 즉, 변수를 평가할 때 R이 데이터를 검색하므로 객체의 필드 이름을 지정하여 액세스 할 수 있습니다. attach()를 이용하면 인자로 주어진 데이터 프레임이나 리스트의 필드를 곧바로 접근할 수 있게 해줍니다. 이를 해제하려면 detach()를 사용합니다.

attach()한 변수 값은 detach()시 원래의 데이터 프레임에는 반영되지 않습니다.

```
attach(what, name=deparse(substitute(what)),
       warn.conflicts=TRUE)
```

구문에서...

- what : 데이터 프레임 또는 리스트 등 R 객체입니다.
- name : 작업공간의 이름을 지정합니다. what 이름을 다른 이름으로 사용할 때 지정합니다.
- warn.conflicts : TRUE(디폴트)이면 데이터가 현재 작업공간과 attach하는 작업공간에 같은 이름의 데이터를 포함하고 있을 경우 경고가 출력됩니다. FALSE이면 경고가 출력되지 않습니다.

attach 사용과 작업공간


```
> attach(iris)
> summary(Sepal.Length)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  4.300   5.100   5.800   5.843   6.400   7.900
```

attach 한 환경에서 변수를 변경시키려면 <<-를 이용합니다. 이것이 원본 데이터의 변경을 의미하는 것은 아닙니다. detach 하면 이 작업공간을 사라집니다. attach 한 변수 값은 detach 시 원래의 데이터 프레임에는 반영되지 않습니다.

```
> Sepal.Length <<- Sepal.Length*25.4 # attach 한 환경 내에서 복사본의 변경
> find("Sepal.Length")
[1] "iris"
> summary(Sepal.Length)                # 작업환경내에서 변경된 데이터
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  109.2   129.5   147.3   148.4   162.6   200.7
> detach("iris")
> summary(iris$Sepal.Length)           # 원래의 작업공간 데이터는 변경되지 않음
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  4.300   5.100   5.800   5.843   6.400   7.900
```

cOnflicts

다음 코드는 서로 다른 작업공간에 같은 이름의 객체가 있는 지 확인하는 예입니다. conflicts() 함수는 충돌하는 변수의 작업공간을 확인할 수 있습니다. 다음 코드에서 lm 변수를 추가했습니다. lm은 stats 패키지에 선형회귀식을 구하기 위한 함수가 lm()으로 정의되어 있습니다. 다음 코드는 이를 확인시켜 줍니다.

```
> lm <- 1:3
> conflicts(, TRUE)
$.GlobalEnv
[1] "lm"

$`package:stats`
[1] "lm"
```

find

find() 함수를 이용해 Sepal.Length가 있는 영역을 확인합니다.

```
> find("Sepal.Length")
[1] ".GlobalEnv" "iris"
> summary(Sepal.Length)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   1.00    3.25    5.50    5.50    7.75   10.00
> summary(iris$Sepal.Length)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  4.300   5.100   5.800   5.843   6.400   7.900

> detach("iris")     #다음 예제를 위해 detach해 주세요.
```

table

table()은 교차 분류 계수를 사용하여 각 팩터 수준의 조합마다 수의 표를 작성합니다. 즉, 데이터를 팩터로 묶을 수 있는 가능한 모든 쌍의 조합을 카운트 한 수의 표를 만듭니다. table() 함수의 결과를 이용하여 막대그래프(barplot) 또는 파이차트(pie)등을 통해 시각화 할 때 주로 사용합니다.

```
table(...,
      exclude=if (useNA=="no") c(NA, NaN),
      useNA=c("no", "ifany", "always"),
      dnn=list.names(...), deparse.level=1)
```

구문에서...

- ... : 팩터로 인터프리트 될 수 있는 한 개 또는 그 이상의 객체입니다. 리스트 또는 데이터 프레임이 될 수 있습니다.
- exclude : ... 에 대해 제거할 수준을 지정합니다. 만일 이 인수가 NA를 포함하지 않고 useNA 인수가 설정되지 않으면 useNA="ifany"를 의미합니다.
- useNA : NA 값을 테이블에 포함할지 여부를 지정합니다.
- dnn : dimnames names를 의미합니다. 결과의 차원에 부여 할 이름을 지정합니다.

table 예

다음 코드는 table() 함수의 가장 간단한 예입니다. iris 데이터의 종별 개수를 출력합니다.

```
> table(iris$Species)                #종별 개수

    setosa versicolor  virginica
        50         50         50
```

addNa()와 is.NA()


```
> d.temp <- addNA(c(1,NA,1:2,1:3))
> d.temp
[1] 1    <NA> 1    2    1    2    3
Levels: 1 2 3 <NA>
> is.na(d.temp) <- 3:4
> d.temp
[1] 1    <NA> <NA> <NA> 1    2    3
Levels: 1 2 3 <NA>
```

as.integer()는 엘리먼트를 정수로 형변환 합니다. 그런데 코드 결과에서 두 번째 값 〈NA〉가 4로 바뀐 이유는 두 번째 〈NA〉는 세 번째, 네 번째 〈NA〉와는 다르게 팩터의 요소이기 때문입니다. 〈NA〉가 팩터의 4번째 요소이기 때문에 as.integer()에 의해 4로 바뀐 것입니다.

```
> d.temp
[1] 1    <NA> <NA> <NA> 1    2    3
Levels: 1 2 3 <NA>

> as.integer(d.temp) # 1 4 NA NA 1 2 3
[1]  1  4 NA NA  1  2  3
```

useNA 속성


```
> table(d.temp)
d.temp
   1    2    3 <NA>
   2    1    1    1
```

useNA="ifany"이면 NA 값을 테이블에 포함시킵니다.

```
> table(d.temp, useNA="ifany")
d.temp
   1    2    3 <NA>
   2    1    1    3
```

exclude=NULL 이면 useNA="ifany"와 동일하게 동작합니다.

```
> table(d.temp, exclude=NULL)
d.temp
   1    2    3 <NA>
   2    1    1    3
```

exclude=NA 이면 NA를 테이블에서 제외합니다. 그러므로 NA는 팩터의 레벨별로 개수를 새는 것에서 제외됩니다.

```
> table(d.temp, exclude=NA)
d.temp
1 2 3
2 1 1
```

aggregate

aggregate()는 데이터를 하위 집합으로 분할하고 각각에 대한 요약 통계를 계산 한 다음 결과를 편리한 형식으로 반환합니다.

```
aggregate(x, ...)
aggregate(x, by, FUN, ..., simplify=TRUE, drop=TRUE)
```

구문에서...

- x : R 객체입니다.
- ... : 함수에서 사용할 인수입니다.
- by : 데이터 프레임 x의 변수와 길이가 같은 그룹화 할 엘리먼트의 목록입니다. 엘리먼트는 사용 전에 팩터로 강제 변환됩니다.
- FUN : 모든 데이터 하위 집합에 적용 할 수 있는 요약 통계를 계산하는 함수입니다.
- simplify : 가능한 경우 결과를 벡터 또는 행렬로 단순화할지 여부를 나타내는 논리값입니다. TRUE(기본값) 이면 연산 결과를 벡터, 행렬 등으로 반환합니다. FALSE 이면 연산 결과를 리스트로 반환합니다.
- drop : 그룹화 값의 사용되지 않은 조합을 삭제할지 여부를 나타내는 논리값입니다. 기본값(TRUE)이 아닌 경우 drop=FALSE는 R 3.3.0부터 사용할 수 있으며 사용되지 않은 조합이 여전히 삭제되는 경우도 있습니다.

aggregate 예

오른쪽 그림을 이용해 aggregate() 함수의 다른 예를 설명해 보겠습니다. 오른쪽 그림은 H 보험회사의 고객들의 보험 청구 데이터 중에서 고객별로 병원에 입원한 입원일 정보만 조회한 데이터입니다.

실제 데이터를 이용해 aggregate() 데이터를 실습 할 수 없으므로 먼저 오른쪽 그림과 같은 데이터를 임의로 만들고 실습해 보겠습니다. CUST_ID는 고객의 아이디를 저장하는 변수이며, HOST_DAY는 병원에 입원한 입원일 수를 저장하는 변수입니다.

```
> CUST_ID <- c(5936, 5936, 5936, 1043, 8545, 4734,
9416, 20267, 2778, 9019, 9019, 9019, 6989, 3372, 1274,
21906, 3362, 3362, 16781, 294)
> HOSP_DAY <- c(2, 2, 2, 6, 0, 4, 0, 23, 29, 13, 13,
13, 13, 0, 0, 13, 0, 12, 23, 0)
> data_claim <- data.frame(CUST_ID, HOSP_DAY)
> head(data_claim)
  CUST_ID HOSP_DAY
1    5936        2
2    5936        2
3    5936        2
4    1043        6
5    8545        0
6    4734        4
```

Filter

	CUST_ID	HOSP_DAY
1	5936	2
2	5936	2
3	5936	2
4	1043	6
5	8545	0
6	4734	4
7	9416	0
8	20267	23
9	2778	29
10	9019	13
11	9019	13

이 데이터에서 고객별로 보험 청구 데이터에서 병원에 입원한 평균 입원일수를 알고자 할 때 aggregate() 함수를 사용할 수 있습니다.

```
> hosp_day_per_cust <- aggregate(data_claim$HOSP_DAY,
+                                by=list(data_claim$CUST_ID), mean)
> names(hosp_day_per_cust) <- c("CUST_ID", "MEAN_DAY")
> hosp_day_per_cust
   CUST_ID MEAN_DAY
1      294        0
2     1043        6
3     1274        0
4     2778       29
5     3362        6
```

조건에 맞는 색인 찾기 : which, which.min, which.max

which()는 벡터 또는 배열에서 주어진 조건을 만족하는 값이 있는 곳의 색인을 찾습니다.

which.min()과 which.max()는 주어진 벡터에서 최솟값 또는 최댓값이 저장된 색인을 찾는 함수입니다.

```
which(x, arr.ind=FALSE, useNames=TRUE)
```

```
which.min(x)
which.max(x)
```

구문에서...

- x : 논리형 벡터 또는 배열입니다. NA 값은 가능하지만 FALSE로 처리됩니다.
- arr.ind : TRUE이면 x가 배열 인 경우 배열 색인을 반환합니다. 기본값은 FALSE입니다.
- useNames : which()의 결과가 dimnames를 가져야 하는지를 나타내는 논리값입니다.

which 예

다음 코드는 기본적인 which() 함수의 사용 예입니다. LETTERS 데이터에서 'R'의 위치를 찾습니다.

```
> LETTERS
 [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R"
[19] "S" "T" "U" "V" "W" "X" "Y" "Z"
> which(LETTERS == "R")
[1] 18
```

다음 코드는 iris 데이터에서 versicolor 종 데이터의 색인을 출력합니다.

```
> which(iris$Species=="versicolor")
 [1]  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68
[19]  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86
[37]  87  88  89  90  91  92  93  94  95  96  97  98  99 100
```

다음 코드는 iris 데이터에서 꽃받침의 길이가 5.0인 데이터의 색인을 출력합니다.

```
> head(iris$Sepal.Length)
[1] 5.1 4.9 4.7 4.6 5.0 5.4
> which(iris$Sepal.Length==5.0)
 [1]  5  8 26 27 36 41 44 50 61 94
```

which.max, which.min 예

which.max()는 가장 큰 데이터의 색인을 출력합니다. which.min()은 가장 작은 데이터의 색인을 출력합니다.

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

다음 코드는 arr.ind=TRUE인 경우의 예입니다. 행렬인 경우 행과 열의 색인을 출력합니다.

```
> ( m <- matrix(1:12, 3, 4) )
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> div.3 <- m %% 3 == 0
> which(div.3)
[1]  3  6  9 12
> which(div.3, arr.ind=TRUE)
     row col
[1,]   3   1
[2,]   3   2
[3,]   3   3
[4,]   3   4
```

useNames 속성

다음 코드는 행렬에 행의 이름이 있을 경우 행렬의 이름이 출력됩니다.

```
> rownames(m) <- paste("Case", 1:3, sep="_")
> m
       [,1] [,2] [,3] [,4]
Case_1    1    4    7   10
Case_2    2    5    8   11
Case_3    3    6    9   12
> which(m %% 5 == 0, arr.ind=TRUE)
       row col
Case_2   2   2
Case_1   1   4
```

useNames=FALSE인 경우 행의 이름이 출력되지 않고 색인이 출력됩니다.

```
> which(m %% 5 == 0, arr.ind=TRUE, useNames=FALSE)
     [,1] [,2]
[1,]    2    2
[2,]    1    4
```