

빅데이터 분석 기초 : R 을 중심으로

- 필요 패키지 : MASS
- 필요 데이터 : -

A. R 소개

A.1 R is...

R 은 데이터 조작, 통계 계산, 시각화 등을 위한 프로그래밍 언어(혹은 환경)이다.

오픈 소스(open source)이고, 엄청난 양의 패키지와 최신 분석 기법을 제공하며, 모든 플랫폼에서 작동하는 특징 때문에, 최근 데이터 과학에서 공용어(*lingua franca*)로 자리잡아가고 있다.

- Download at [<http://www.r-project.org>].



[Home]

Download

[CRAN](#)

R Project

[About R](#)
[Contributors](#)
[What's New?](#)
[Mailing Lists](#)
[Bug Tracking](#)
[Conferences](#)
[Search](#)

R Foundation

[Foundation](#)
[Board](#)
[Members](#)
[Donors](#)
[Donate](#)

Documentation

[Manuals](#)
[FAQs](#)
[The R Journal](#)
[Books](#)

The R Project for Statistical Computing

Getting Started

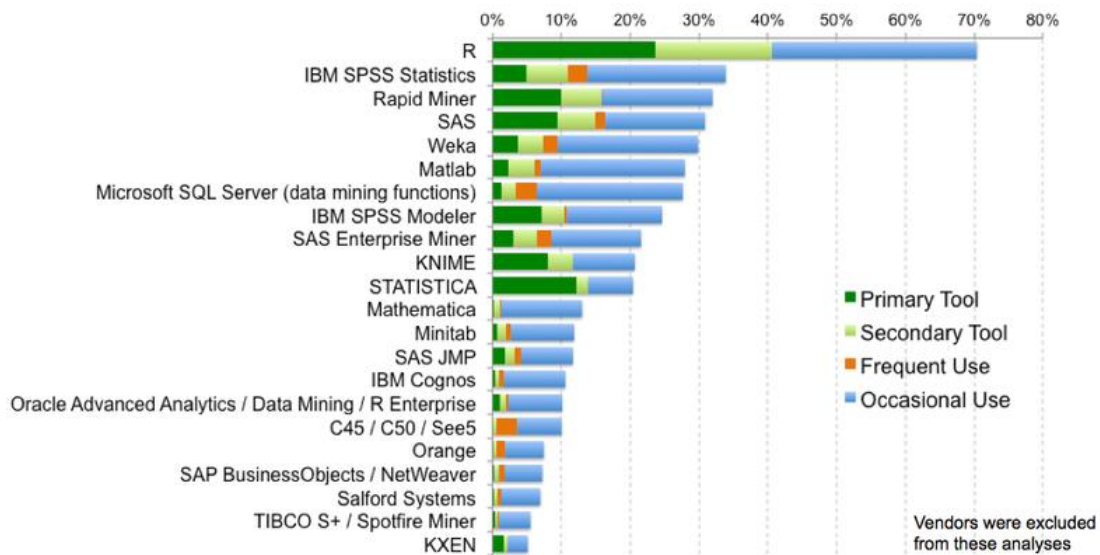
R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS. To [download R](#), please choose your preferred [CRAN mirror](#).

If you have questions about R like how to download and install the software, or what the license terms are, please read our [answers to frequently asked questions](#) before you send an email.

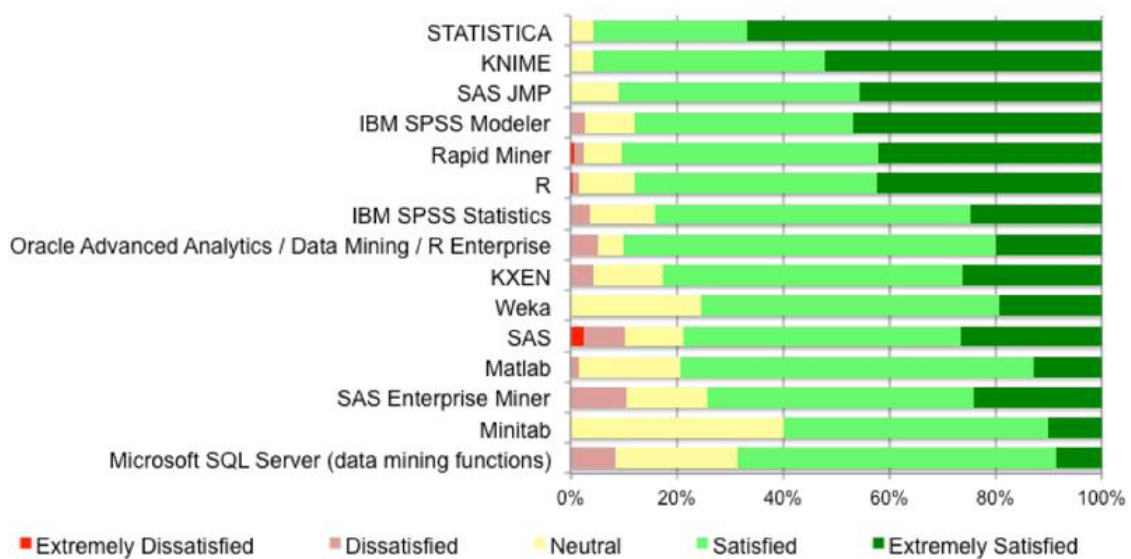
News

- [R version 3.2.3 \(Wooden Christmas-Tree\)](#) has been released on 2015-12-10.>
- [The R Journal Volume 7/1](#) is available.
- [R version 3.1.3 \(Smooth Sidewalk\)](#) has been released on 2015-03-09.
- [useR! 2015](#), took place at the University of Aalborg, Denmark, June 30 - July 3, 2015.

- R is now the most popular tool...[<http://www.r-bloggers.com/r-usage-skyrocketing-rexer-poll/>]



- Most users are satisfied with R...[<http://www.r-bloggers.com/r-usage-skyrocketing-rexer-poll/>]



A.2 R-Studio

R 을 여타 상용 프로그램처럼 편리하게 사용할 수 있는 사용자 인터페이스를 제공하는 프로그램으로 아래 사이트에서 무료로 다운로드 받아 사용할 수 있다.

- Download at [<https://www.rstudio.com/>]

A.3 R Markdown

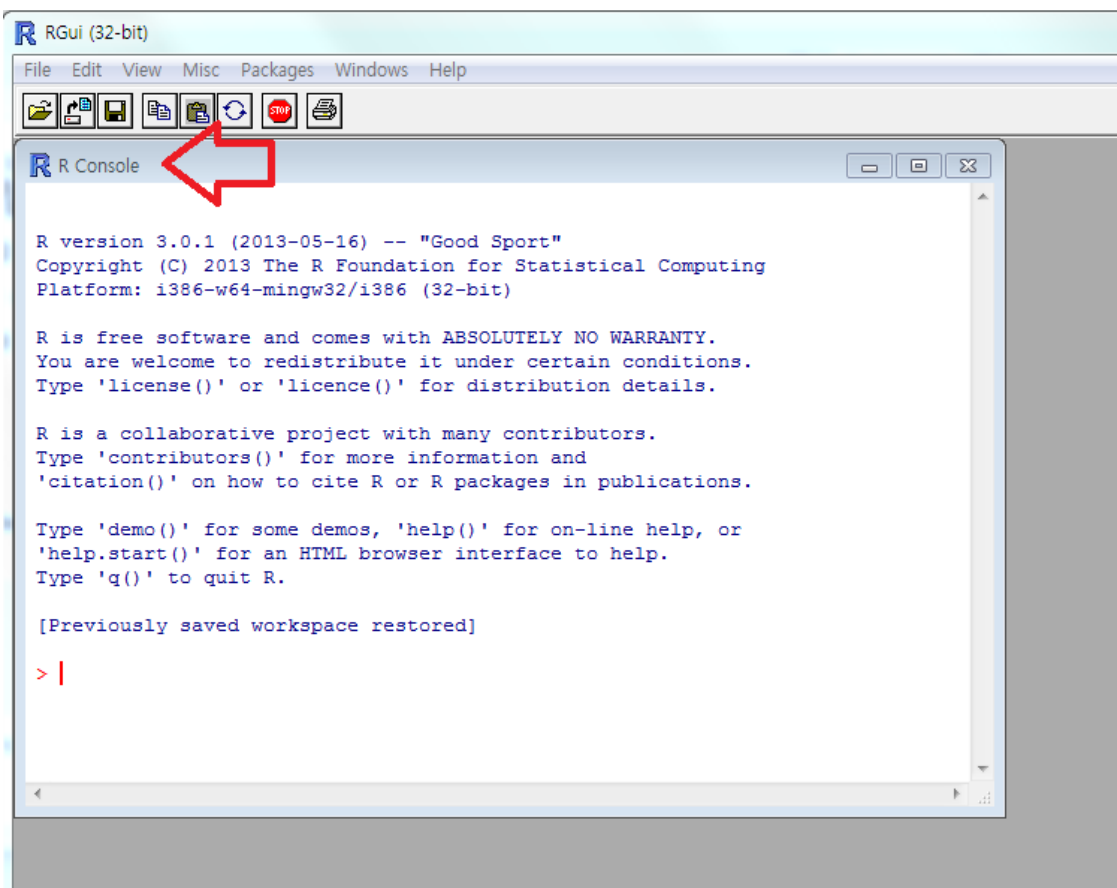
- 마크다운이란?
 - 이 문서는 R-Studio 을 이용해 마크다운(markdown)으로 작성되었음
 - 마크다운이란 일반 텍스트 문서를 편집할 때 쓰는 문법의 하나로서, 주로 README 파일이나 온라인 문서 등을 편집할 때 쓰임
 - 태그(tag)를 이용해 글자의 굵기를 조절하거나 문서 내에 이미지, 하이퍼링크, 수식 등을 삽입하는 것이 가능
 - 마크다운을 이용해 작성된 문서는 쉽게 HTML, pdf, MS-WORD 등 다른 문서 형태로 손쉽게 변환할 수 있음
 - R-Studio 에서 작성 가능한 R 마크다운에 대한 자세한 내용은 [<http://rmarkdown.rstudio.com>]을 참고

A.4 Console and script

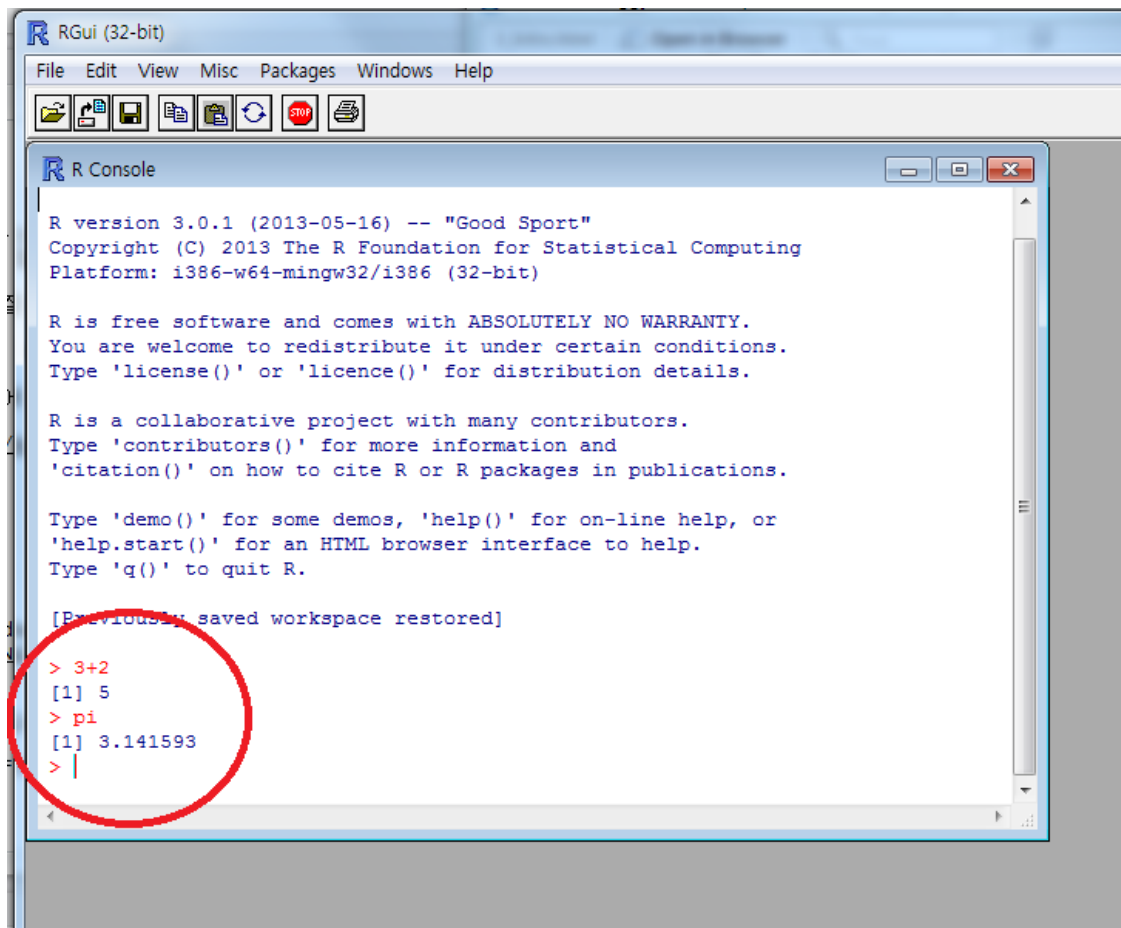
A 장에서는 일단 R-Studio 를 사용하지 않는 것을 전제로 설명을 진행한다.

R 환경에서 익숙하게 R 을 다룰 수 있다면 이후 R-Studio 를 사용하게 되는 시점이 오더라도 아무 어려움없이 사용할 수 있을 것이다.

- R 을 실행하면 R-GUI 창이 뜬
- R-GUI 창 내부에 **R console** 창이 있는데, 명령어 입력 및 결과물 출력이 이루어지는 공간임



- R console 내 명령 프롬프트:



```
RGui (32-bit)
File Edit View Misc Packages Windows Help

R Console

R version 3.0.1 (2013-05-16) -- "Good Sport"
Copyright (C) 2013 The R Foundation for Statistical Computing
Platform: i386-w64-mingw32/i386 (32-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

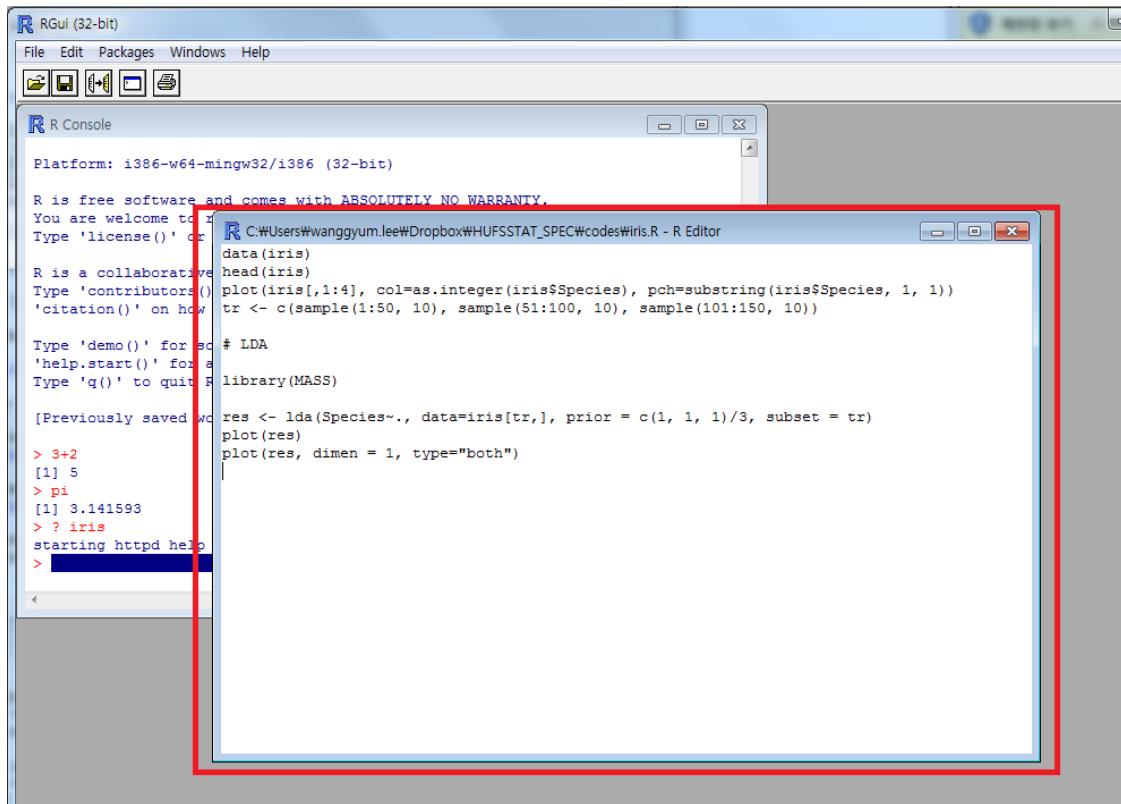
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Previously saved workspace restored]

> 3+2
[1] 5
> pi
[1] 3.141593
> |
```

- **R script** 란 R 콘솔에 입력할 일련의 명령어 및 코드를 모은 텍스트 파일을 의미



A.5 Working directory

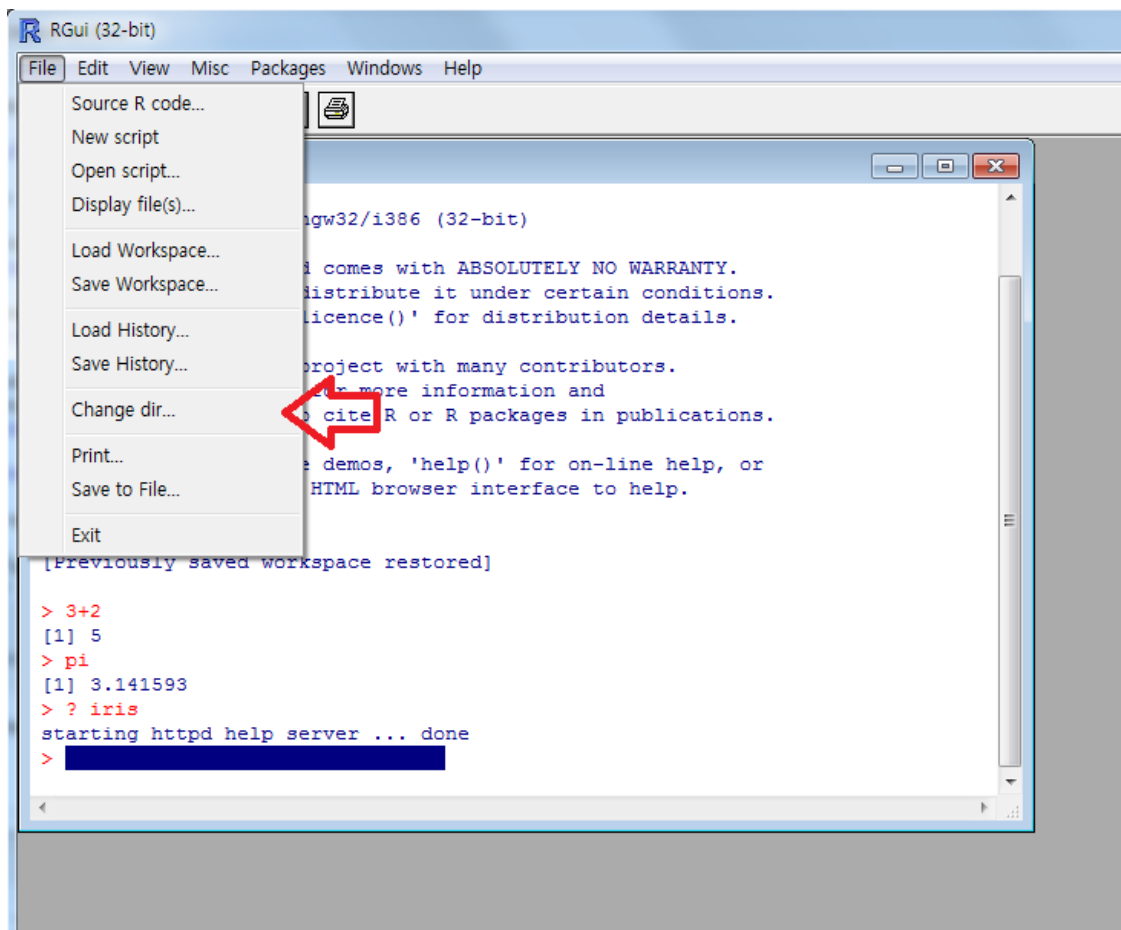
작업 디렉토리(Working directory)란 모든 파일 입출력의 기본 위치가 되는 디렉토리를 의미한다.

- 현재 작업 디렉토리를 확인하려면 `getwd()` 함수를 콘솔에 입력하고, 특정 디렉토리를 작업 디렉토리로 지정하려면 `setwd()` 함수를 이용

```
getwd()
```

```
setwd("c:/mywork/codes/")    # 각자의 상황에 맞게 수정해 사용하시오
```

- 또는 메인 메뉴를 이용하는 것도 가능



A.6 도움말 Help

도움말 시스템이 매우 편리하게 제공된다.

- 예를 들어 `persp()`라는 이름의 함수에 대해 궁금하면 명령어 프롬프트에 다음과 같이 입력하면 됨

```
? persp
```

or

```
help(persp)
```

- 확장 도움말 기능을 제공. 예를 들어 "log"를 포함한 키워드 혹은 함수에 대해 궁금하면...

```
?? log
```

or

```
help.search("log")
```

- Online documentation: Visit R-project website and click on "Manuals".



The R Project for Statistical Computing

[\[Home\]](#)

Download

[CRAN](#)

R Project

[About R](#)

[Contributors](#)

[What's New?](#)

[Mailing Lists](#)

[Bug Tracking](#)

[Conferences](#)

[Search](#)

R Foundation

[Foundation](#)

[Board](#)

[Members](#)

[Donors](#)

[Donate](#)

Documentation

[Manuals](#)

[FAQs](#)

[The R Journal](#)

[Books](#)

[Certification](#)

[Other](#)

Getting Started

R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS. To **download R**, please choose your preferred [CRAN mirror](#).

If you have questions about R like how to download and install the software, or what the license terms are, please read our [answers to frequently asked questions](#) before you send an email.

News

- **R version 3.2.2 (Fire Safety)** has been released on 2015-08-14.
- **The R Journal Volume 7/1** is available.
- **R version 3.1.3 (Smooth Sidewalk)** has been released on 2015-03-09.
- **useR! 2015**, will take place at the University of Aalborg, Denmark, June 30 - July 3, 2015.
- **useR! 2014**, took place at the University of California, Los Angeles, USA June 30 - July 3, 2014.



A.7 R command

- 변수명은 알파벳, 숫자, 마침표(.), underscore(_) 등을 조합해 만듦
- 변수에 특정 값(혹은 변수)을 할당할 때 <-을 사용하면 됨. =을 사용할 수도 있음
- 모든 이름의 시작은 알파벳 또는 마침표(.)로

```
abc <- 3      # OK
.jeong <- abc # OK
2.res <- 3    # 예러...
```

- 여러 명령어를 한 줄에 입력할 때는 세미콜론(;)으로 구분

```
beta.0 <- 3; beta.1 <- 2
```

- 주석문(comment)은 #을 이용

```
rnorm(10) # to generate 10 random numbers

## [1]  0.12706735 -0.09481749  0.53859706 -2.49568205 -0.84342272
## [6] -0.09498363  2.29564916 -0.39050955 -1.30794342  0.92254107
```

- 키보드의 화살표 키(↑, ↓)를 이용하면 예전에 입력한 명령어를 탐색해 다시 불러올 수 있음
- 대소문자를 구별함

```
a <- 1
A <- 2
a == A

## [1] FALSE
```

A.8 Mathematical computation

- 산술연산

```
x <- 11; y <- 3
x+y

## [1] 14

x-y
```

```
## [1] 8
x*y
## [1] 33
x/y
## [1] 3.666667
x^y
## [1] 1331
x%/%y # integer quotient
## [1] 3
x%%y # modulo
## [1] 2
```

- 지수 표현 Numbers with exponents

```
1.2e3 # 1.2 * 1,000
## [1] 1200
1.2e-3 # 1.2 * 1/1,000
## [1] 0.0012
```

- 복소수 Complex numbers

```
z1 <- 1.2+3.4i
z2 <- 4i
z1 + z2
## [1] 1.2+7.4i
```

- 수학 함수

```
x <- 10; y <- 3.21; n <- 2
log(x)
## [1] 2.302585
log10(x)
## [1] 1
log(n, x)
```

```
## [1] 0.30103
exp(x)
## [1] 22026.47
sin(x) # cos(x), tan(x), asin(x), acos(x), atan(x)
## [1] -0.5440211
abs(x)
## [1] 10
sqrt(x)
## [1] 3.162278
floor(y)
## [1] 3
ceiling(x)
## [1] 10
round(y, digits=0)
## [1] 3
gamma(x)
## [1] 362880
lgamma(x)
## [1] 12.80183
factorial(x)
## [1] 3628800
choose(x, n)
## [1] 45
```

A.9 Packages

모든 R 함수 및 dataset 은 패키지로 묶여서 저장되어 있다.

따라서 특정 함수 및 dataset 은 해당 패키지를 load 해야 사용 가능하다.

- 메모리 관리 및 검색 시 효율적임
- 패키지 개발자들이 기존 패키지와 충돌 걱정 없이 개발 가능 & 확장성 ↑
- 패키지 설치 및 업데이트는 인터넷을 통해 항상 가능
- Standard (or base) package 들은 R source code 의 일부이기도 함
 - 기본적인 R 함수, 통계 및 데이터 분석을 위한 함수, 시각화 등을 위한 함수 등을 포함
 - R 설치 시 함께 설치되고 따로 load 할 필요 없음 *c.f.* Contributed packages

B. R 에서 자료 타입에 따른 data manipulation

본 장에서는 데이터 타입에 따른 자료 생성 및 조작법의 기초를 익힌다.

B.1 벡터(Vector)

벡터를 구성하는 모든 성분은 같은 타입이어야 한다.

실수(double), 정수(integer), 문자열(string), 논리값(logical) 등으로 구성할 수 있다.

```
x <- c(1, 2.5, 3.2)           # double
y <- c(1L, 2L, 3L)           # integer
z <- c("KTX", "Saemaul", "Mugunghwa") # string
v <- c(TRUE, FALSE, FALSE, TRUE) # logical
```

벡터의 각 성분은 위치에 따라 인덱싱한다.

R 에서 인덱싱은 []를 이용한다.

여러 성분을 동시에 인덱싱해서 벡터의 일부를 추출해 부분 벡터를 만드는 것도 가능하다.

```
x[3]           # x 의 세 번째 성분
## [1] 3.2

x[c(1,3)]      # x 의 첫 번째, 세 번째 성분을 추출한 부분 벡터
## [1] 1.0 3.2
```

벡터의 성분은 숫자로 된 인덱스 대신 이름을 가질 수 있다.

```
fruit <- c(5, 3, 2)
names(fruit) <- c("apple", "orange", "peach")
fruit

## apple orange peach
##      5      3      2

fruit[c("apple", "peach")]

## apple peach
##      5      2

fruit <- setNames(c(5, 3, 2), c("apple", "orange", "peach"))
```

- `names()` : 객체의 이름을 알아내거나 이름을 부여할 때 사용하는 함수
- `setNames()` : 객체의 이름을 부여할 때 편리한 함수

벡터에 관련된 주요 기본 함수를 소개하면 다음과 같다.

```
typeof(x)                                # double
## [1] "double"
is.double(x)                             # TRUE
## [1] TRUE
as.double(y)                             # as.integer(x)
## [1] 1 2 3
is.numeric(z)                            # FALSE
## [1] FALSE
as.numeric(v)                            # 1 0 0 1
## [1] 1 0 0 1
length(x)                                # Length of x
## [1] 3
```

- `typeof()`: 벡터를 구성하는 성분의 타입을 알려주는 함수
- `is.double()`: 실수 타입인지 여부를 논리값으로 알려주는 함수
- `as.double()`: 실수 벡터로 변환해주는 함수
- `is.numeric()`: 수치 벡터인 지 여부를 논리값으로 알려주는 함수
- `as.numeric()`: 수치 벡터로 변환해주는 함수
- `length()`: 벡터의 길이를 계산해주는 함수

새로운 벡터를 생성하려면 `c()` 함수를 이용한다.

```
a <- c(1, 2, 3)
b <- c(5, 6)
x <- c(a, 4, b)                        # x <- c(1,2,3,4,5,6)
```

벡터 인덱스의 범위를 벗어나는 위치의 성분에 값을 할당하면 빈 자리는 결측치 처리를 한다. 즉, 위에서 생성한 길이가 3 인 벡터 `a` 에 대해 7 번째 성분값에 2 를 할당하면, 4, 5, 6 번째 성분의 값은 결측치(결측치는 NA 로 표시)로 채워진 길이가 7 인 벡터가 생성된다.

```
a[7] <- 2                                # assign to the 7th element
a                                          # R extends the vector automatically
## [1] 1 2 3 NA NA NA 2
```

append() 함수를 이용하면 기존 벡터에 새로운 값 추가해 새로운 벡터 만들 수 있다.

```
append(x, 99, after=3)    # x 의 세 번째 성분 다음에 99 삽입
## [1]  1  2  3 99  4  5  6

append(x, -99, after=0)   # x 의 맨 앞에 -99 삽입
## [1] -99  1  2  3  4  5  6
```

: 연산자, seq() 함수, rep() 함수를 이용해 수열을 생성해 벡터를 만들 수 있다.

```
x <- seq(from=0, to=1, by=0.1)    # 0 부터 1 까지 0.1 씩 증가하는 등차수열
y <- seq(from=0, to=1, length=11) # 0 부터 1 까지 길이가 11 인 등차수열
z <- 1:10                        # 1, 2, 3, ... , 9, 10
5:-5                             # 5, 4, 3, ... , -4, -5
## [1]  5  4  3  2  1  0 -1 -2 -3 -4 -5

rep(1, 10)                       # repeat 1 ten times
## [1] 1 1 1 1 1 1 1 1 1 1
```

- m:n: m 부터 n 까지 1 씩 증가(감소)하는 수열 생성
- seq(): 등차수열 만들기
- rep(): 같은 값을 반복해서 수열 만들기

벡터 간 산술연산은 성분별로 이루어진다.

```
x <- 1:3; y <- c(2,2,2)
x+y
## [1] 3 4 5

x-y
## [1] -1 0 1

x*y
## [1] 2 4 6

x/y
## [1] 0.5 1.0 1.5

x^y
```

```
## [1] 1 4 9
```

길이가 서로 다른 벡터 간에 연산을 실시하면 **재사용규칙**(recycling rule)이 적용된다.

즉, 길이가 짧은 벡터를 반복적으로 다시 사용해 길이가 긴 벡터와 같은 길이를 갖도록 연장해 연산을 실시한다.

재사용 규칙이 적용되는 경우, R 은 연산 결과와 함께 혹시 실수가 있었을 지 모르기 때문에 경고메세지를 함께 제공한다.

```
z <- rep(2, 5)
x+z          # c(1, 2, 3, 1, 2) + c(2, 2, 2, 2, 2)와 같은 효과
```

```
## [1] 3 4 5 3 4
```

- z 는 길이가 5 인 벡터인데, 길이가 3 인 벡터 x 와 덧셈연산을 실시하면 x 의 길이가 5 가 되도록 첫 두 성분을 재사용해 길이를 늘여 덧셈을 실시

벡터와 스칼라 간의 연산은 재사용규칙의 원리에 따라 이해하면 쉽다.

즉, 스칼라 객체를 길이가 1 인 벡터로 보고 재사용 규칙을 적용하면 된다.

즉, 위에서 생성한 벡터 y 의 모든 성분에서 3 을 빼려면 다음과 같이하면 된다.

```
y-3
```

```
## [1] -1 -1 -1
```

- y-rep(3, length(y))와 같은 효과

논리값으로 구성된 논리값 벡터(logical vector)를 사용할 수 있다.

```
x <- 1:10; y <- rep(5, 10)
z <- x<5          # less than
sum(z)
```

```
## [1] 4
```

```
x<=5          # less than or equal to
```

```
## [1] TRUE TRUE TRUE TRUE TRUE FALSE FALSE FALSE FALSE FALSE
```

```
x==5          # equal
```

```
## [1] FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE
```

```
x!=5          # not equal
```

```
## [1] TRUE TRUE TRUE TRUE FALSE TRUE TRUE TRUE TRUE TRUE
```

```
(x>5)&(y<2)    # and
```

```
## [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```



```
(x<5)|(y<2)      # or
## [1]  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
```

R 은 다양한 수학/통계 함수를 제공한다.

```
x <- rnorm(10)
x
## [1] -0.34246270  0.58343596  1.54340829 -1.68275292 -0.07331261
## [6]  1.02802626 -0.71574842 -0.62199705  0.03085437 -1.39492435

y <- 1:10
y
## [1]  1  2  3  4  5  6  7  8  9 10

z <- -5:4
z
## [1] -5 -4 -3 -2 -1  0  1  2  3  4
```

- rnorm(): 정규분포를 따르는 난수를 발생시켜주는 함수

```
max(x)      # 최대값
## [1] 1.543408

min(x)      # 최소값
## [1] -1.682753

sum(x)      # 모든 성분의 합
## [1] -1.645473

prod(x)     # 모든 성분의 곱
## [1] 0.0007493877

mean(x)     # 평균
## [1] -0.1645473

median(x)   # 중앙값
## [1] -0.2078877

range(x)    # 최대값과 최소값의 차이
## [1] -1.682753  1.543408
```

```

quantile(x, probs=c(0.2, 0.7))    # 분위수
##          20%          70%
## -0.8515836  0.1966288

var(x)    # 분산
## [1] 1.034316

sd(x)     # 표준편차
## [1] 1.017013

cov(x, y)  # 공분산
## [1] -1.120272

cor(x, y)  # 상관계수
## [1] -0.363824

x

## [1] -0.34246270  0.58343596  1.54340829 -1.68275292 -0.07331261
## [6]  1.02802626 -0.71574842 -0.62199705  0.03085437 -1.39492435

sort(x)    # sort (or order) a vector or factor (partially) into ascending or descending order
## [1] -1.68275292 -1.39492435 -0.71574842 -0.62199705 -0.34246270
## [6] -0.07331261  0.03085437  0.58343596  1.02802626  1.54340829

rank(x)     # returns the sample ranks of the values in a vector
## [1]  5  8 10  1  6  9  3  4  7  2

order(x)     # returns a permutation which rearranges its first argument into ascending or descending order
## [1]  4 10  7  8  1  5  9  2  6  3

x[order(x)]  # sort(x)와 같은 효과
## [1] -1.68275292 -1.39492435 -0.71574842 -0.62199705 -0.34246270
## [6] -0.07331261  0.03085437  0.58343596  1.02802626  1.54340829

cumsum(x)    # 누적합
## [1] -0.34246270  0.24097327  1.78438156  0.10162864  0.02831603
## [6]  1.05634228  0.34059387 -0.28140319 -0.25054882 -1.64547316

cumprod(x)   # 누적곱

```

```
## [1] -0.3424626951 -0.1998050528 -0.3083807743 0.5189286481 -0.0380440145
## [6] -0.0391102458 0.0279930964 -0.0174116235 -0.0005372246 0.0007493877

cummax(x) # 누적최대값

## [1] -0.3424627 0.5834360 1.5434083 1.5434083 1.5434083 1.5434083
## [7] 1.5434083 1.5434083 1.5434083 1.5434083

cummin(x) # 누적최소값

## [1] -0.3424627 -0.3424627 -0.3424627 -1.6827529 -1.6827529 -1.6827529
## [7] -1.6827529 -1.6827529 -1.6827529 -1.6827529

pmax(x, y, z) # 성분별 최대값

## [1] 1 2 3 4 5 6 7 8 9 10

pmin(x, y, z) # 성분별 최소값

## [1] -5.00000000 -4.00000000 -3.00000000 -2.00000000 -1.00000000
## [6] 0.00000000 -0.71574842 -0.62199705 0.03085437 -1.39492435
```

결측치는 NA 로 표시한다.

```
x <- c(1, 2, 3, NA, 5)
is.na(x)

## [1] FALSE FALSE FALSE TRUE FALSE
```

- is.na(): 결측치 여부를 논리값으로 알려주는 함수

인덱스 벡터를 이용하면 주어진 벡터의 일부를 추출해 부분 벡터를 만들어 사용할 수 있다.

```
x <- -10:10
x[3]

## [1] -8

x[1:3]

## [1] -10 -9 -8

x[c(1,3,5)]

## [1] -10 -8 -6

x[-1]

## [1] -9 -8 -7 -6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6 7 8 9 10

x[-c(1,3,5)]

## [1] -9 -7 -5 -4 -3 -2 -1 0 1 2 3 4 5 6 7 8 9 10
```

```

y <- x[x<0]           # x 에서 음수인 성분을 추출해 y 에 할당
x[x<0] <- -x[x<0]     # 절대값 벡터
x <- c(1, 2, 3, NA, 5)
x[!is.na(x)]

## [1] 1 2 3 5

x[is.na(x)] <- 4      # 결측인 성분을 4 로 채우기

```

- 인덱스에 -를 사용하면 해당성분을 제외하라는 의미임

B.2 배열(arrays)과 행렬(matrices)

배열 또는 행렬을 생성하는 가장 기본적인 방법은 벡터를 변형하는 것이다.

즉, 데이터를 일단 벡터 형태로 읽어서 배열 혹은 행렬 형태로 모양을 바꾼다.

array() 함수, matrix() 함수를 이용해 생성한다.

```

z <- array(1:20, dim=c(4,5))  # 벡터 1:20 를 4*5 배열로 재배치
z

##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    5    9   13   17
## [2,]    2    6   10   14   18
## [3,]    3    7   11   15   19
## [4,]    4    8   12   16   20

z <- matrix(1:20, 4, 5)
z

##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    5    9   13   17
## [2,]    2    6   10   14   18
## [3,]    3    7   11   15   19
## [4,]    4    8   12   16   20

z <- matrix(2, 4, 5)
z

##      [,1] [,2] [,3] [,4] [,5]
## [1,]    2    2    2    2    2
## [2,]    2    2    2    2    2
## [3,]    2    2    2    2    2
## [4,]    2    2    2    2    2

z <- matrix(c(1, 2, 3,
              4, 5, 6),

```

```

      nrow = 2, ncol = 3, byrow=T) # Readability enhanced
z
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6

      z[2, 3] # indexed by the position
## [1] 6

```

기존의 벡터(들) 또는 행렬(들)을 결합해 새로운 행렬을 만들 수 있다.

```

x <- 1:4
y <- 5:8

cbind(x, y)
##      x y
## [1,] 1 5
## [2,] 2 6
## [3,] 3 7
## [4,] 4 8

rbind(x, y)
##      [,1] [,2] [,3] [,4]
## x      1    2    3    4
## y      5    6    7    8

B <- matrix(0, 4, 5)
cbind(B, 1:4)
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    0    0    0    0    0    1
## [2,]    0    0    0    0    0    2
## [3,]    0    0    0    0    0    3
## [4,]    0    0    0    0    0    4

A <- matrix(1:20, 4, 5)
B <- matrix(1:20, 4, 5)
C <- cbind(A, B)
C
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]    1    5    9   13   17    1    5    9   13   17
## [2,]    2    6   10   14   18    2    6   10   14   18
## [3,]    3    7   11   15   19    3    7   11   15   19
## [4,]    4    8   12   16   20    4    8   12   16   20

```

- cbind(): 열 방향으로 결합

- `rbind()`: 행 방향으로 결합

행렬 간 산술연산은 벡터와 마찬가지로 성분별 연산이다. 즉, 같은 성분에 있는 값끼리 연산을 실시한다.

```
A+B
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    2   10   18   26   34
## [2,]    4   12   20   28   36
## [3,]    6   14   22   30   38
## [4,]    8   16   24   32   40

A-B
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    0    0    0    0    0
## [2,]    0    0    0    0    0
## [3,]    0    0    0    0    0
## [4,]    0    0    0    0    0

A*B
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1   25   81  169  289
## [2,]    4   36  100  196  324
## [3,]    9   49  121  225  361
## [4,]   16   64  144  256  400

A/B
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    1    1    1    1
## [2,]    1    1    1    1    1
## [3,]    1    1    1    1    1
## [4,]    1    1    1    1    1
```

- 두 행렬의 차원이 같지 않은 경우 벡터와 달리 재사용규칙이 적용되지 않고 에러메시지 발생

```
matrix(1:20, 4, 5) + matrix(1:10, 2, 5)  # Not run...
```

행렬 대수에서 사용하는 전치행렬, 행렬 곱, 역행렬, 단위행렬, 고유치(eigenvalue), 고유벡터(eigenvector), 외적(outer product) 등을 계산할 수 있다.

```
A <- matrix(runif(20), 5, 4)
t(A)  # matrix transposition

##      [,1] [,2] [,3] [,4] [,5]
## [1,] 0.09322308 0.1439579 0.3752732 0.2081587 0.48607438
## [2,] 0.01076963 0.3981393 0.9109081 0.7525530 0.50526267
```

```
## [3,] 0.58973203 0.1532971 0.1512634 0.4680938 0.02143471
## [4,] 0.03015954 0.7546799 0.9172832 0.5942066 0.40688646

B <- t(A)%*%A      # %*%: Matrix multiplication
solve(B)           # Inverse matrix

##           [,1]      [,2]      [,3]      [,4]
## [1,] 10.8349300 -5.9461743  0.26166795  1.07313024
## [2,] -5.9461743 10.0950556 -0.90147753 -6.70089942
## [3,]  0.2616679 -0.9014775  2.30368313  0.06294342
## [4,]  1.0731302 -6.7008994  0.06294342  6.27789774

diag(5)            # 5-by-5 diagonal identity matrix

##           [,1] [,2] [,3] [,4] [,5]
## [1,]      1    0    0    0    0
## [2,]      0    1    0    0    0
## [3,]      0    0    1    0    0
## [4,]      0    0    0    1    0
## [5,]      0    0    0    0    1

diag(B)            # diagonal matrix with the diagonal elements of B

## [1] 0.4498428 1.8100110 0.6137357 1.9304979

C <- outer(1:3, 4:6, FUN="*") # outer product
```

- `runif()`: uniform 난수 생성 함수
- 성분끼리의 곱이 아닌 행렬곱은 `%*%`를 사용해야 함에 특히 주의
- `solve(A, B)`는 원래 $Ax = B$ 형태의 연립방정식을 푸는 데 사용하는 함수임

행렬의 각 행과 열에 이름을 부여해 사용하면 편리한 때가 있다.

`colnames()` 함수와 `rownames()`를 사용하면 된다.

```
z <- matrix(1:20, 4, 5)
z

##           [,1] [,2] [,3] [,4] [,5]
## [1,]      1    5    9   13   17
## [2,]      2    6   10   14   18
## [3,]      3    7   11   15   19
## [4,]      4    8   12   16   20

nrow(z)

## [1] 4
```

```

colnames(z) <- c("alpha", "beta", "gamma", "delta", "eps")
rownames(z) <- c("a", "b", "c", "d")
z
##   alpha beta gamma delta eps
## a     1    5     9    13   17
## b     2    6    10    14   18
## c     3    7    11    15   19
## d     4    8    12    16   20

```

- `nrow()`: 행의 개수를 계산하는 함수
- `ncol()`: 열의 개수를 계산하는 함수

B.3 리스트(List)

리스트(list)는 성분의 타입이 동일하지 않은 벡터로 `list()`를 이용해 생성한다.

R 식으로 표현하자면 리스트의 성분들은 서로 다른 모드(mode)를 가진다고 할 수 있다.

심지어 리스트 또는 데이터프레임같은 구조적 개체를 성분으로 가질 수 있어 재귀적(recursive) 자료구조를 갖게 할 수도 있다.

```

Hong <- list(kor.name="홍길동",
             eng.name="Gil-dong",
             age=43,
             married=T,
             no.child=2,
             child.ages=c(13, 10))

Hong$age
## [1] 43

Hong["age"]
## $age
## [1] 43

Hong$child.age[2]
## [1] 10

str(Hong)
## List of 6
## $ kor.name : chr "홍길동"
## $ eng.name : chr "Gil-dong"
## $ age      : num 43

```



```
## $ married : logi TRUE
## $ no.child : num 2
## $ child.ages: num [1:2] 13 10
```

- 리스트의 각 성분은 이름을 가질 수 있음: 위의 kor.name, eng.name, age, married, no.child, child.age 등.
- 리스트의 특정 성분을 가리키고 싶을 때에는 \$를 사용함. ""를 이용해 성분 이름을 지정할 수도 있음
- str(): 객체의 자료구조를 간단히 요약해 보여주는 함수로서, 앞으로 매우 자주 사용하게 될 함수이므로 꼭 기억해두기 바람

벡터와 마찬가지로 리스트의 각 성분에 대해 이름 대신 위치에 따라 인덱싱이 가능하다.

다만 인덱스를 표현할 때 [[1]], [[2]]와 같이 이중으로 된 []를 사용해야 한다는 점에 유의하면 된다.

즉 위에서 생성한 Hong 의 두 번째 성분인 kor.name 의 값을 가리키려면 Hong[[2]]과 같이 나타내면 된다.

```
Hong[[2]]
## [1] "Gil-dong"
```

인덱스 벡터를 이용하면 리스트 역시 부분 추출이 가능하다.

즉, Hong[c(2,3)] 는 Hong 의 두 번째, 세 번째 성분을 추출하게 된다.

다만 이중으로 된 []이 아니고 홑겹 []을 사용함에 유의하자.

```
Hong[c(1,2)]
## $kor.name
## [1] "홍길동"
##
## $eng.name
## [1] "Gil-dong"
```

리스트에 자주 사용되는 기본 함수에 대해 익혀보자.

is.list()는 객체가 리스트인 지 여부를 논리값으로 알려주는 함수이고, lapply()는 리스트의 각 성분에 대해 같은 함수를 반복적으로 적용한 값을 리턴하는 함수이다.

```
is.list(Hong)
## [1] TRUE

x <- list(a = 1:10,
          beta = exp(-3:3),
          logic = c(TRUE,FALSE,FALSE,TRUE))
x
```

```
## $a
## [1] 1 2 3 4 5 6 7 8 9 10
##
## $beta
## [1] 0.04978707 0.13533528 0.36787944 1.00000000 2.71828183 7.3890561
0
## [7] 20.08553692
##
## $logic
## [1] TRUE FALSE FALSE TRUE

  lapply(x, mean)                                # x의 각 성분의 평균값을 계산

## $a
## [1] 5.5
##
## $beta
## [1] 4.535125
##
## $logic
## [1] 0.5

  lapply(x, quantile, probs = (1:3)/4) # x의 각 성분의 사분위수를 계산

## $a
## 25% 50% 75%
## 3.25 5.50 7.75
##
## $beta
##      25%      50%      75%
## 0.2516074 1.0000000 5.0536690
##
## $logic
## 25% 50% 75%
## 0.0 0.5 1.0
```

- `exp()`: 지수함수 e^x 의 값을 계산
- `(1:3)/4`: $(1/4, 2/4, 3/4)$

B.4 데이터프레임(Data frame)

데이터프레임은 R에서 사용하는 데이터 저장 방식 중 가장 널리 사용되는 방식으로, 같은 길이의 벡터들을 성분으로 갖는 리스트라고 이해하면 된다.

따라서 2 차원의 자료구조를 갖게 된다.

새로운 데이터프레임을 생성하려면 `data.frame()` 함수를 이용한다.

```
x <- c(100, 75, 80)
y <- c("A302043", "A302044", "A302045")
z <- data.frame(score=x, ID=y)
```

`data.frame()` 함수에서 `stringsAsFactors` 옵션을 이용해 문자열 벡터 성분의 성격을 factor 로 지정할 수 있다.

```
dat.1 <- data.frame(x=1:3, y=c('a', 'b', 'c'))
str(dat.1)

## 'data.frame':    3 obs. of  2 variables:
## $ x: int  1 2 3
## $ y: Factor w/ 3 levels "a","b","c": 1 2 3

dat.2 <- data.frame(x=1:3, y=c('a', 'b', 'c'), stringsAsFactors=F)
str(dat.2)

## 'data.frame':    3 obs. of  2 variables:
## $ x: int  1 2 3
## $ y: chr  "a" "b" "c"
```

- factor 에 대한 자세한 소개는 다음 절에서 익힘

다음은 데이터프레임에 자주 사용되는 기본 함수의 예이다.

특히 `class()` 함수는 객체지향(object oriented) 프로그래밍의 측면에서 객체의 클래스 정보를 알려주거나 지정할 수 있는 유용한 함수이니 기억해두자.

```
typeof(dat.2)      # list
## [1] "list"

class(dat.2)       # data frame
## [1] "data.frame"

is.data.frame(dat.2) # TRUE
## [1] TRUE
```

지정한 크기로 미리 할당해 초기화할 수 있다.

아래는 `initial`, `dosage`, `blood`, `response` 라는 변수를 갖고 1,000,000 건의 데이터를 저장할 수 있는 데이터프레임을 미리 할당해 초기화하기 위한 코드이다.

```
N <- 1000000
dat <- data.frame(initial=character(N),
                  dosage=numeric(N),
```

```
blood=factor(N, levels=c("O", "A", "B", "AB")),
response=numeric(N) )
```

데이터프레임의 특정 열(column)을 선택하하려면 리스트와 같은 방법을 사용하면 된다.

다음은 R 에 데이터프레임 형태로 내장된 데이터셋인 `faithful` 의 각 성분을 추출하는 예이다.

```
str(faithful)

## 'data.frame':    272 obs. of  2 variables:
## $ eruptions: num  3.6 1.8 3.33 2.28 4.53 ...
## $ waiting : num  79 54 74 62 85 55 88 85 51 85 ...

faithful[[1]]

## [1] 3.600 1.800 3.333 2.283 4.533 2.883 4.700 3.600 1.950 4.350 1.833
## [12] 3.917 4.200 1.750 4.700 2.167 1.750 4.800 1.600 4.250 1.800 1.750
## [23] 3.450 3.067 4.533 3.600 1.967 4.083 3.850 4.433 4.300 4.467 3.367
## [34] 4.033 3.833 2.017 1.867 4.833 1.833 4.783 4.350 1.883 4.567 1.750
## [45] 4.533 3.317 3.833 2.100 4.633 2.000 4.800 4.716 1.833 4.833 1.733
## [56] 4.883 3.717 1.667 4.567 4.317 2.233 4.500 1.750 4.800 1.817 4.400
## [67] 4.167 4.700 2.067 4.700 4.033 1.967 4.500 4.000 1.983 5.067 2.017
## [78] 4.567 3.883 3.600 4.133 4.333 4.100 2.633 4.067 4.933 3.950 4.517
## [89] 2.167 4.000 2.200 4.333 1.867 4.817 1.833 4.300 4.667 3.750 1.867
## [100] 4.900 2.483 4.367 2.100 4.500 4.050 1.867 4.700 1.783 4.850 3.683
## [111] 4.733 2.300 4.900 4.417 1.700 4.633 2.317 4.600 1.817 4.417 2.617
## [122] 4.067 4.250 1.967 4.600 3.767 1.917 4.500 2.267 4.650 1.867 4.167
## [133] 2.800 4.333 1.833 4.383 1.883 4.933 2.033 3.733 4.233 2.233 4.533
## [144] 4.817 4.333 1.983 4.633 2.017 5.100 1.800 5.033 4.000 2.400 4.600
## [155] 3.567 4.000 4.500 4.083 1.800 3.967 2.200 4.150 2.000 3.833 3.500
## [166] 4.583 2.367 5.000 1.933 4.617 1.917 2.083 4.583 3.333 4.167 4.333
## [177] 4.500 2.417 4.000 4.167 1.883 4.583 4.250 3.767 2.033 4.433 4.083
## [188] 1.833 4.417 2.183 4.800 1.833 4.800 4.100 3.966 4.233 3.500 4.366
## [199] 2.250 4.667 2.100 4.350 4.133 1.867 4.600 1.783 4.367 3.850 1.933
## [210] 4.500 2.383 4.700 1.867 3.833 3.417 4.233 2.400 4.800 2.000 4.150
## [221] 1.867 4.267 1.750 4.483 4.000 4.117 4.083 4.267 3.917 4.550 4.083
## [232] 2.417 4.183 2.217 4.450 1.883 1.850 4.283 3.950 2.333 4.150 2.350
## [243] 4.933 2.900 4.583 3.833 2.083 4.367 2.133 4.350 2.200 4.450 3.567
## [254] 4.500 4.150 3.817 3.917 4.450 2.000 4.283 4.767 4.533 1.850 4.250
## [265] 1.983 2.250 4.750 4.117 2.150 4.417 1.817 4.467

faithful$waiting

## [1] 79 54 74 62 85 55 88 85 51 85 54 84 78 47 83 52 62 84 52 79 51 47 78
## [24] 69 74 83 55 76 78 79 73 77 66 80 74 52 48 80 59 90 80 58 84 58 73 83
## [47] 64 53 82 59 75 90 54 80 54 83 71 64 77 81 59 84 48 82 60 92 78 78 65
## [70] 73 82 56 79 71 62 76 60 78 76 83 75 82 70 65 73 88 76 80 48 86 60 90
## [93] 50 78 63 72 84 75 51 82 62 88 49 83 81 47 84 52 86 81 75 59 89 79 59
## [116] 81 50 85 59 87 53 69 77 56 88 81 45 82 55 90 45 83 56 89 46 82 51 86
## [139] 53 79 81 60 82 77 76 59 80 49 96 53 77 77 65 81 71 70 81 93 53 89 45
## [162] 86 58 78 66 76 63 88 52 93 49 57 77 68 81 81 73 50 85 74 55 77 83 83
```

```
## [185] 51 78 84 46 83 55 81 57 76 84 77 81 87 77 51 78 60 82 91 53 78 46 77
## [208] 84 49 83 71 80 49 75 64 76 53 94 55 76 50 82 54 75 78 79 78 78 70 79
## [231] 70 54 86 50 90 54 54 77 79 64 75 47 86 63 85 82 57 82 67 74 54 83 73
## [254] 73 88 80 71 83 56 79 78 84 58 83 43 60 75 81 46 90 46 74
```

데이터프레임의 일부 행(row)를 추출할 때는 `subset()` 함수를 사용한다.

함수 `subset()`의 옵션 `subset=`을 이용해 추출 조건을 지정하고, `select=` 옵션을 이용해 추출 대상 변수(열)을 지정할 수 있다.

다음은 MASS 패키지의 내장 데이터셋인 `Cars93` 에서 데이터를 조건에 맞도록 추출하는 예이다.

```
library(MASS) # for dataset Car93
subset(Cars93, subset=(MPG.city > 32))

##      Manufacturer Model  Type Min.Price Price Max.Price MPG.city MPG.highway
## 39          Geo Metro Small      6.7   8.4      10.0      46          50
## 42          Honda Civic Small      8.4  12.1      15.8      42          46
## 80          Subaru Justy Small      7.3   8.4       9.5      33          37
## 83          Suzuki Swift Small      7.3   8.6      10.0      39          43
##      AirBags DriveTrain Cylinders EngineSize Horsepower  RPM
## 39          None      Front        3         1.0        55 5700
## 42 Driver only      Front        4         1.5       102 5900
## 80          None      4WD         3         1.2        73 5600
## 83          None      Front        3         1.3        70 6000
##      Rev.per.mile Man.trans.avail Fuel.tank.capacity Passengers Length
## 39          3755             Yes          10.6           4      151
## 42          2650             Yes          11.9           4      173
## 80          2875             Yes           9.2           4      146
## 83          3360             Yes          10.6           4      161
##      Wheelbase Width Turn.circle Rear.seat.room Luggage.room Weight  Origin
## 39          93    63           34          27.5          10   1695 non-USA
## 42         103    67           36          28.0          12   2350 non-USA
## 80          90    60           32          23.5          10   2045 non-USA
## 83          93    63           34          27.5          10   1965 non-USA
##      Make
## 39    Geo Metro
## 42   Honda Civic
## 80  Subaru Justy
## 83   Suzuki Swift

subset(Cars93, select=Model, subset=(MPG.city > 32))

##      Model
## 39 Metro
## 42 Civic
## 80 Justy
## 83 Swift
```

데이터프레임 내에 결측치가 포함된 경우, `na.omit()` 함수를 이용해 해당 행(row)을 골라내어 삭제할 수 있다.

```

x <- data.frame(a=1:5, b=c(1,2,NA,4,5))
cumsum(x)           # fails

##      a  b
## 1  1  1
## 2  3  3
## 3  6 NA
## 4 10 NA
## 5 15 NA

cumsum(na.omit(x))

##      a  b
## 1  1  1
## 2  3  3
## 4  7  7
## 5 12 12

```

데이터프레임의 특정 열을 제외시킬 때는 `subset()` 함수의 `select=` 옵션에 `-`을 사용한다.

```

str(EuStockMarkets)

## mts [1:1860, 1:4] 1629 1614 1607 1621 1618 ...
## - attr(*, "dimnames")=List of 2
## ..$ : NULL
## ..$ : chr [1:4] "DAX" "SMI" "CAC" "FTSE"
## - attr(*, "tsp")= num [1:3] 1991 1999 260
## - attr(*, "class")= chr [1:3] "mts" "ts" "matrix"

cor(EuStockMarkets)

##           DAX           SMI           CAC           FTSE
## DAX  1.0000000 0.9911539 0.9662274 0.9751778
## SMI  0.9911539 1.0000000 0.9468139 0.9899691
## CAC  0.9662274 0.9468139 1.0000000 0.9157265
## FTSE 0.9751778 0.9899691 0.9157265 1.0000000

cor(subset(EuStockMarkets, select=-SMI)) # SMI excluded

##           DAX           CAC           FTSE
## DAX  1.0000000 0.9662274 0.9751778
## CAC  0.9662274 1.0000000 0.9157265
## FTSE 0.9751778 0.9157265 1.0000000

cor(subset(EuStockMarkets, select=-c(SMI, CAC))) # SMI and CAC excluded

##           DAX           FTSE
## DAX  1.0000000 0.9751778
## FTSE 0.9751778 1.0000000

```

여러 개의 데이터프레임을 행방향 혹은 열방향으로 붙여 새로운 데이터프레임을 만드려면 `cbind()` 함수와 `rbind()` 함수를 이용한다.

이 때 붙일 데이터프레임들의 차원이 서로 잘 맞아야 한다.

```
a <- data.frame(x=c(5, 10, 15), y=c("a", "b", "c"))
b <- data.frame(z=c(10, 20, 30))
cbind(a, b)

##      x y  z
## 1   5 a 10
## 2  10 b 20
## 3  15 c 30

a1 <- data.frame(x=c(20, 25, 30), y=c("d", "e", "f"))
rbind(a, a1)

##      x y
## 1   5 a
## 2  10 b
## 3  15 c
## 4  20 d
## 5  25 e
## 6  30 f
```

지정한 변수를 기준으로 두 데이터프레임을 결합할 때는 `merge()` 함수를 이용한다.

결합 기준 변수는 `by=` 옵션에서 지정하는데, 변수명에 ""를 붙여야 함에 유의하자.

```
a <- data.frame(Name=c("Mary", "Jane", "Alice", "Bianca"), score=c(90, 95, 100, 100))
b <- data.frame(Name=c("Jane", "Alice", "Ana"), weight=c(70, 55, 60))
merge(a, b, by="Name")

##      Name score weight
## 1 Alice   100     55
## 2 Jane    95     70

merge(a, b, by="Name", all=T)

##      Name score weight
## 1 Alice   100     55
## 2 Bianca  100     NA
## 3 Jane    95     70
## 4 Mary    90     NA
## 5 Ana     NA     60
```

`with()` 함수를 이용하면 보다 간결한 코딩이 가능하다.

즉, 매번 데이터프레임의 이름을 타이핑할 필요없이 열(변수) 이름만을 사용해 코딩이 가능하도록 해 준다.

```

z <- (Cars93$Price-mean(Cars93$Price))/sd(Cars93$Price) # Cars93$... 지/
저분...
head(z)
## [1] -0.3736947  1.4897694  0.9928456  1.8831674  1.0860189 -0.3943998

z <- with(Cars93, (Price-mean(Price))/sd(Price)) # 깔끔...
head(z)
## [1] -0.3736947  1.4897694  0.9928456  1.8831674  1.0860189 -0.3943998

```

- `head()`: 데이터프레임의 처음 몇 개의 행을 화면에 출력해 내용을 확인할 수 있게 해 주는 함수
- `tail()`: 데이터프레임의 마지막 몇 개의 행을 화면에 출력해 내용을 확인할 수 있게 해 주는 함수

B.5 Factor

R에서 사용하는 데이터 타입 중 요인(factor)은 문자형 변수로서 특정 수준값만을 가질 수 있는 데이터 타입을 의미한다.

예를 들어 ABO 식 혈액형을 나타내는 변수를 문자형 변수를 정의할 때 이 변수가 취할 수 있는 값은 A, B, AB, O의 네 가지 값만을 가져야 할 것이고, 다른 종류의 문자가 들어오면 에러가 나야 할 것이다.

이러한 종류의 데이터 타입을 요인형이라 한다. 요인형 벡터는 `factor()` 함수를 이용해 생성할 수 있다. 구체적 용법은 다음 코드를 살펴보면 쉽게 이해할 수 있다.

```

blood.type <- factor(c("A", "A", "AB", "O", "O"), levels=c("A", "B", "AB",
"O"))
table(blood.type)

## blood.type
##  A  B AB  O
##  2  0  1  2

```

- `table()` 함수: 도수분포표를 작성해주는 함수

많은 경우 요인 타입임을 명시할 필요가 없다.

만일 특별히 요인형으로 지정하지 않은 문자열 벡터에 대해 요인형 자료를 위한 함수를 적용시키면(예: `table()`), R은 자동으로 데이터 타입을 요인형으로 바꾸어 처리한다.

```

blood.type <- c("A", "B", "AB", "O", "O")
table(blood.type)

```



```
## blood.type
##  A AB  B  O
##  1  1  1  2
```

B.6 데이터 타입 변환하기

`as.numeric()`, `as.character()`와 같이 `as.`로 시작하는 함수를 이용하면 된다.

```
x <- 1:3
y <- as.character(x)
y
## [1] "1" "2" "3"
as.numeric(y)
## [1] 1 2 3
```

- `as.vector()`, `as.matrix()`, `as.list()`, `as.data.frame()`

B.7 데이터 입맛대로 변형하기

데이터셋을 여러 그룹으로 쪼개어서 새로운 데이터셋을 만드려면 `split()` 함수를 사용하면 된다.

아래는 MASS 패키지의 내장 데이터인 `Cars93` 의 `MPG.city` 변수를 `Origin` 변수값(USA, non-USA)에 따라 두 개의 그룹으로 쪼개어 두 개의 성분을 갖는 리스트를 만드는 예이다.

```
library(MASS)      # for the dataset Cars93
summary(Cars93)
```

##	Manufacturer	Model	Type	Min.Price	Price
##	Chevrolet: 8	100 : 1	Compact:16	Min. : 6.70	Min. : 7.40
##	Ford : 8	190E : 1	Large :11	1st Qu.:10.80	1st Qu.:12.20
##	Dodge : 6	240 : 1	Midsize:22	Median :14.70	Median :17.70
##	Mazda : 5	300E : 1	Small :21	Mean :17.13	Mean :19.51
##	Pontiac : 5	323 : 1	Sporty :14	3rd Qu.:20.30	3rd Qu.:23.30
##	Buick : 4	535i : 1	Van : 9	Max. :45.40	Max. :61.90
##	(Other) :57	(Other):87			
##	Max.Price	MPG.city	MPG.highway	AirBags	
##	Min. : 7.9	Min. :15.00	Min. :20.00	Driver & Passenger:16	
##	1st Qu.:14.7	1st Qu.:18.00	1st Qu.:26.00	Driver only :43	
##	Median :19.6	Median :21.00	Median :28.00	None :34	
##	Mean :21.9	Mean :22.37	Mean :29.09		
##	3rd Qu.:25.3	3rd Qu.:25.00	3rd Qu.:31.00		
##	Max. :80.0	Max. :46.00	Max. :50.00		
##					

```

## DriveTrain Cylinders EngineSize Horsepower RPM
## 4WD :10 3 : 3 Min. :1.000 Min. : 55.0 Min. :3800
## Front:67 4 :49 1st Qu.:1.800 1st Qu.:103.0 1st Qu.:4800
## Rear :16 5 : 2 Median :2.400 Median :140.0 Median :5200
## 6 :31 Mean :2.668 Mean :143.8 Mean :5281
## 8 : 7 3rd Qu.:3.300 3rd Qu.:170.0 3rd Qu.:5750
## rotary: 1 Max. :5.700 Max. :300.0 Max. :6500
##
## Rev.per.mile Man.trans.avail Fuel.tank.capacity Passengers
## Min. :1320 No :32 Min. : 9.20 Min. :2.000
## 1st Qu.:1985 Yes:61 1st Qu.:14.50 1st Qu.:4.000
## Median :2340 Median :16.40 Median :5.000
## Mean :2332 Mean :16.66 Mean :5.086
## 3rd Qu.:2565 3rd Qu.:18.80 3rd Qu.:6.000
## Max. :3755 Max. :27.00 Max. :8.000
##
## Length Wheelbase Width Turn.circle
## Min. :141.0 Min. : 90.0 Min. :60.00 Min. :32.00
## 1st Qu.:174.0 1st Qu.: 98.0 1st Qu.:67.00 1st Qu.:37.00
## Median :183.0 Median :103.0 Median :69.00 Median :39.00
## Mean :183.2 Mean :103.9 Mean :69.38 Mean :38.96
## 3rd Qu.:192.0 3rd Qu.:110.0 3rd Qu.:72.00 3rd Qu.:41.00
## Max. :219.0 Max. :119.0 Max. :78.00 Max. :45.00
##
## Rear.seat.room Luggage.room Weight Origin
## Min. :19.00 Min. : 6.00 Min. :1695 USA :48
## 1st Qu.:26.00 1st Qu.:12.00 1st Qu.:2620 non-USA:45
## Median :27.50 Median :14.00 Median :3040
## Mean :27.83 Mean :13.89 Mean :3073
## 3rd Qu.:30.00 3rd Qu.:15.00 3rd Qu.:3525
## Max. :36.00 Max. :22.00 Max. :4105
## NA's :2 NA's :11
##
## Make
## Acura Integra: 1
## Acura Legend : 1
## Audi 100 : 1
## Audi 90 : 1
## BMW 535i : 1
## Buick Century: 1
## (Other) :87
##
## tmp <- split(Cars93$MPG.city, Cars93$Origin) # grouping by 'Origin'
## str(tmp) # List of 2
##
## List of 2
## $ USA : int [1:48] 22 19 16 19 16 16 25 25 19 21 ...
## $ non-USA: int [1:45] 25 18 20 19 22 46 30 24 42 24 ...

```

리스트 객체의 각 성분에 대해 지정한 함수를 적용해 얻은 결과를 새로운 리스트 혹은 벡터로 만드려면 `lapply()` 함수, `sapply()` 함수를 사용하면 된다.

`lapply` 는 리스트를, `sapply` 는 벡터를 리턴해 준다.

다음은 5 개의 성분을 갖는 리스트 `Jeong` 에 `length()` 함수를 적용해 각 성분의 벡터 길이를 알려주는 예이다.

```
Jeong <- list(first.name="Samuel", age=44, gender="M", n.child=2, child.g
ender=c("M", "F"))
lapply(Jeong, length)    # returns a list

## $first.name
## [1] 1
##
## $age
## [1] 1
##
## $gender
## [1] 1
##
## $n.child
## [1] 1
##
## $child.gender
## [1] 2

sapply(Jeong, length)    # returns a vector

##   first.name      age      gender  n.child child.gender
##           1         1           1         1           2
```

행렬 혹은 데이터프레임의 각 행과 열에 대해 같은 작업을 반복적으로 실행해 새로운 벡터를 생성하려면 `apply()` 함수를 이용한다.

`apply()`의 두 번째 입력값을 MARGIN 이라 하는데, 이 값을 1 로 지정하면 각 행에 대해, 2 로 지정하면 각 열에 대해 반복 작업을 실행한다.

다음의 코드는 행렬 `a` 에 대해 행 방향 평균과 열 방향 평균을 계산하는 예이다.

```
a <- matrix(1:20, 4, 5)
a

##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    5    9   13   17
## [2,]    2    6   10   14   18
## [3,]    3    7   11   15   19
## [4,]    4    8   12   16   20

apply(a, 1, mean)    # to every row

## [1]  9 10 11 12
```

```
apply(a, 2, mean)    # to every column
```

```
## [1]  2.5  6.5 10.5 14.5 18.5
```

주어진 데이터에 그룹별로 함수를 적용하려면 `tapply()` 함수를 사용한다.

다음 코드는 Cars93 데이터의 Price 변수의 평균을 Origin 별로 계산하는 예이다.

```
with(Cars93, tapply(Price, Origin, mean)) # tapply(Cars93$Price, Cars93$Origin, mean) 과 같은 결과
```

```
##      USA  non-USA
```

```
## 18.57292 20.50889
```

- `with()` 함수: 데이터프레임의 성분 이름만으로 깔끔하게 (지저분한 \$ 없이) 코딩을 가능하게 해 주는 매우 유용한 함수임.

행 방향 그룹별로 함수를 적용하려면 `by()` 함수를 이용한다.

아래 코드는 Cars93 의 Origin 별로 `summary()` 함수를 적용한 예이다.

```
by(Cars93, Cars93$Origin, summary)
```

```
## Cars93$Origin: USA
```

```
##      Manufacturer      Model      Type      Min.Price
## Chevrolet : 8    Achieva   : 1  Compact: 7    Min.    : 6.90
## Ford       : 8    Aerostar  : 1  Large   :11   1st Qu.:11.40
## Dodge      : 6    Astro     : 1  Midsize:10   Median  :14.50
## Pontiac    : 5    Bonneville: 1  Small   : 7    Mean    :16.54
## Buick      : 4    Camaro    : 1  Sporty  : 8    3rd Qu.:19.43
## Oldsmobile: 4    Capri     : 1  Van     : 5    Max.    :37.50
## (Other)    :13    (Other)   :42
##      Price      Max.Price      MPG.city      MPG.highway
## Min.    : 7.40   Min.    : 7.90   Min.    :15.00   Min.    :20.00
## 1st Qu.:13.47   1st Qu.:14.97   1st Qu.:18.00   1st Qu.:26.00
## Median :16.30   Median :18.40   Median :20.00   Median :28.00
## Mean    :18.57   Mean    :20.63   Mean    :20.96   Mean    :28.15
## 3rd Qu.:20.73   3rd Qu.:24.50   3rd Qu.:23.00   3rd Qu.:30.00
## Max.    :40.10   Max.    :42.70   Max.    :31.00   Max.    :41.00
```

```
##
##      AirBags  DriveTrain  Cylinders  EngineSize
## Driver & Passenger: 9  4WD   : 5  3      : 0  Min.    :1.300
## Driver only       :23  Front:34  4      :22  1st Qu.:2.200
## None              :16  Rear  : 9  5      : 0  Median  :3.000
##                  6      :20  Mean    :3.067
##                  8      : 6  3rd Qu.:3.800
##                  rotary: 0  Max.    :5.700
```

```
##
##      Horsepower      RPM      Rev.per.mile  Man.trans.avail
## Min.    : 63.0   Min.    :3800   Min.    :1320   No :26
```

```

## 1st Qu.:108.8 1st Qu.:4750 1st Qu.:1771 Yes:22
## Median :143.5 Median :4900 Median :2035
## Mean :147.5 Mean :4991 Mean :2119
## 3rd Qu.:170.0 3rd Qu.:5200 3rd Qu.:2482
## Max. :300.0 Max. :6500 Max. :3285
##
## Fuel.tank.capacity Passengers Length Wheelbase
## Min. :10.00 Min. :2.000 Min. :141.0 Min. : 90.0
## 1st Qu.:15.47 1st Qu.:5.000 1st Qu.:177.0 1st Qu.:101.0
## Median :16.45 Median :5.000 Median :188.5 Median :105.0
## Mean :17.05 Mean :5.333 Mean :188.3 Mean :105.7
## 3rd Qu.:19.05 3rd Qu.:6.000 3rd Qu.:199.2 3rd Qu.:111.0
## Max. :27.00 Max. :8.000 Max. :219.0 Max. :119.0
##
## Width Turn.circle Rear.seat.room Luggage.room
## Min. :63.00 Min. :32.00 Min. :19.00 Min. : 6.00
## 1st Qu.:68.00 1st Qu.:39.00 1st Qu.:26.25 1st Qu.:13.00
## Median :71.50 Median :41.00 Median :28.00 Median :14.50
## Mean :70.96 Mean :40.48 Mean :28.13 Mean :14.88
## 3rd Qu.:74.00 3rd Qu.:43.00 3rd Qu.:30.50 3rd Qu.:17.00
## Max. :78.00 Max. :45.00 Max. :36.00 Max. :22.00
##
## NA's :1 NA's :6
##
## Weight Origin Make
## Min. :1845 USA :48 Buick Century : 1
## 1st Qu.:2705 non-USA: 0 Buick LeSabre : 1
## Median :3282 Buick Riviera : 1
## Mean :3195 Buick Roadmaster: 1
## 3rd Qu.:3639 Cadillac DeVille: 1
## Max. :4105 Cadillac Seville: 1
##
## (Other) :42
## -----
## Cars93$Origin: non-USA
## Manufacturer Model Type Min.Price
## Mazda : 5 100 : 1 Compact: 9 Min. : 6.70
## Hyundai : 4 190E : 1 Large : 0 1st Qu.: 9.10
## Nissan : 4 240 : 1 Midsize:12 Median :16.30
## Toyota : 4 300E : 1 Small :14 Mean :17.76
## Volkswagen: 4 323 : 1 Sporty : 6 3rd Qu.:22.90
## Honda : 3 535i : 1 Van : 4 Max. :45.40
## (Other) :21 (Other):39
##
## Price Max.Price MPG.city MPG.highway
## Min. : 8.00 Min. : 9.10 Min. :17.00 Min. :21.00
## 1st Qu.:11.60 1st Qu.:12.90 1st Qu.:19.00 1st Qu.:25.00
## Median :19.10 Median :21.70 Median :22.00 Median :30.00
## Mean :20.51 Mean :23.26 Mean :23.87 Mean :30.09
## 3rd Qu.:26.70 3rd Qu.:28.50 3rd Qu.:26.00 3rd Qu.:33.00
## Max. :61.90 Max. :80.00 Max. :46.00 Max. :50.00
##
## AirBags DriveTrain Cylinders EngineSize

```

```

## Driver & Passenger: 7 4WD : 5 3 : 3 Min. :1.000
## Driver only :20 Front:33 4 :27 1st Qu.:1.600
## None :18 Rear : 7 5 : 2 Median :2.200
## 6 :11 Mean :2.242
## 8 : 1 3rd Qu.:2.800
## rotary: 1 Max. :4.500
##
## Horsepower RPM Rev.per.mile Man.trans.avail
## Min. : 55.0 Min. :4500 Min. :1955 No : 6
## 1st Qu.:102.0 1st Qu.:5400 1st Qu.:2325 Yes:39
## Median :135.0 Median :5600 Median :2505
## Mean :139.9 Mean :5590 Mean :2560
## 3rd Qu.:168.0 3rd Qu.:6000 3rd Qu.:2710
## Max. :278.0 Max. :6500 Max. :3755
##
## Fuel.tank.capacity Passengers Length Wheelbase
## Min. : 9.20 Min. :2.000 Min. :146.0 Min. : 90
## 1st Qu.:13.20 1st Qu.:4.000 1st Qu.:170.0 1st Qu.: 97
## Median :15.90 Median :5.000 Median :180.0 Median :103
## Mean :16.25 Mean :4.822 Mean :177.8 Mean :102
## 3rd Qu.:18.50 3rd Qu.:5.000 3rd Qu.:187.0 3rd Qu.:106
## Max. :22.50 Max. :7.000 Max. :200.0 Max. :115
##
## Width Turn.circle Rear.seat.room Luggage.room
## Min. :60.00 Min. :32.00 Min. :23.00 Min. : 8.00
## 1st Qu.:66.00 1st Qu.:36.00 1st Qu.:26.00 1st Qu.:11.00
## Median :67.00 Median :37.00 Median :27.50 Median :14.00
## Mean :67.69 Mean :37.33 Mean :27.51 Mean :12.85
## 3rd Qu.:70.00 3rd Qu.:39.00 3rd Qu.:28.50 3rd Qu.:14.00
## Max. :74.00 Max. :43.00 Max. :35.00 Max. :17.00
## NA's :1 NA's :5
##
## Weight Origin Make
## Min. :1695 USA : 0 Acura Integra: 1
## 1st Qu.:2475 non-USA:45 Acura Legend : 1
## Median :2950 Audi 100 : 1
## Mean :2942 Audi 90 : 1
## 3rd Qu.:3405 BMW 535i : 1
## Max. :4100 Geo Metro : 1
## (Other) :39

```

B.8 문자열과 날짜 변수

문자열(string)의 길이를 알고 싶으면 `nchar()` 함수를 사용한다.

벡터의 길이(성분 개수)를 계산해주는 `length()`와 다른 함수이다.

아래 코드를 살펴보면 쉽게 이해할 수 있을 것이다.

```
nchar("ABC")
## [1] 3

length("ABC")
## [1] 1

nchar(c("A", "B", "C"))
## [1] 1 1 1

length(c("A", "B", "C"))
## [1] 3
```

여러 개의 문자열을 붙여서 하나로 만드려면 `paste()` 함수를 이용한다.
붙일 때 사용할 구분자는 `sep=` 옵션에서 지정할 수 있다.

```
paste("A", "B", "C")           # default separator: " "
## [1] "A B C"

paste("A", "B", "C", sep="")
## [1] "ABC"

paste("A", "B", "C", sep="/")
## [1] "A/B/C"
```

문자열의 일부를 추출하려면 `substr()` 함수를 사용한다.
`start=`, `stop=` 옵션으로 추출 시작 위치와 끝 위치를 지정할 수 있다.

```
substr("12345678", start=5, stop=7)
## [1] "567"

cities <- c("New York, NY", "Los Angeles, CA", "Peoria, IL")
substr(cities, start=nchar(cities)-1, stop=nchar(cities)) # extract last
two characters
## [1] "NY" "CA" "IL"
```

지정한 구분자를 기준으로 문자열을 쪼개려면 `strsplit()` 함수를 이용한다.
`strsplit()`는 쪼개진 부분 문자열을 각 성분으로 갖는 리스트 형태의 객체를 리턴한다.
다음은 문자열 `path` 를 `/` 를 기준으로 쪼개는 예이다.

```
path <- "/home/data/test.csv"
strsplit(path, "/")
## [[1]]
## [1] ""      "home"  "data"  "test.csv"
```

- 문자열 벡터에 이 함수를 적용한 예:

```
path <- c("/home/data/test1.csv",
          "/home/data/test2.csv",
          "/home/data/test2.csv")
strsplit(path, "/")

## [[1]]
## [1] ""      "home"    "data"    "test1.csv"
##
## [[2]]
## [1] ""      "home"    "data"    "test2.csv"
##
## [[3]]
## [1] ""      "home"    "data"    "test2.csv"
```

현재 날짜를 알고싶으면 Sys.Date() 함수를 이용한다.

이 함수가 리턴하는 값의 데이터 타입은 날짜형이다.

```
Sys.Date()

## [1] "2015-12-21"

class(Sys.Date()) # The class of the result from Sys.Date() is...

## [1] "Date"
```

문자열을 날짜변수로 변환하려면 as.Date() 함수를 이용한다.

as.Date()는 기본적으로 문자열이 yyyy-mm-dd 형태임을 가정하고 변환을 시도한다.

다른 형태로 주어진 문자열을 날짜로 변환할 때는 format= 값을 지정해야 한다.

아래 예제를 살펴보면 쉽게 이해할 수 있을 것이다.

```
as.Date("2015-10-09") # standard format

## [1] "2015-10-09"

as.Date("10/09/2015") # error (not in a standard format)

## [1] "0010-09-20"

as.Date("10/09/2015", format="%m/%d/%Y") # American date format

## [1] "2015-10-09"

as.Date("10/09/15", format="%m/%d/%y") # 2-digit year

## [1] "2015-10-09"
```

- format 값에서 %Y 는 네 자리, %y 는 두 자리로 연도를 표시한다는 뜻임

날짜 변수를 문자열 변수로 바꾸려면 `format()` 함수 또는 `as.character()` 함수 등을 사용하면 된다. 두 함수 모두 문자형으로 표현할 때 날짜 형태를 위한 `format=` 옵션을 지정할 수 있다. 아래 예제를 살펴보면 쉽게 이해할 수 있을 것이다.

```
format(Sys.Date())
## [1] "2015-12-21"

as.character(Sys.Date())
## [1] "2015-12-21"

format(Sys.Date(), format="%m/%d/%Y")
## [1] "12/21/2015"

as.character(Sys.Date(), format="%m/%d/%y")
## [1] "12/21/15"

format(Sys.Date(), format="%b/%d/%y")    # %b: Abbreviated month name
## [1] "12/21/15"

format(Sys.Date(), format="%B %d, %Y")    # %B: Full month name
## [1] "12 월 21, 2015"
```

따로 따로 주어진 연, 월, 일 정보를 하나로 합쳐 하나의 날짜형 데이터를 만드려면 `ISOdate()` 함수를 이용한다.

이 함수는 날짜 정보와 시간 정보를 모두 포함한 POSIXct 객체를 리턴하므로 날짜 정보만을 원하는 경우 `as.Date()` 함수를 이용해 날짜형으로 변환하는 절차가 필요하다.

2015 년 2 월 29 일과 같이 무효인 날짜값은 결측으로 처리한다.

연, 월, 일 정보를 벡터값으로 제공하면 날짜형 벡터를 만들어준다.

```
ISOdate(2015, 10, 9)    # year, month, day
## [1] "2015-10-09 12:00:00 GMT"

as.Date(ISOdate(2015, 10, 9))
## [1] "2015-10-09"

ISOdate(2015, 2, 29)    # an invalid date
## [1] NA

years <- 2011:2015
months <- rep(1, 5)
days <- c(4, 9, 14, 19, 24, 29)
as.Date(ISOdate(years, months, days))
```

```
## [1] "2011-01-04" "2012-01-09" "2013-01-14" "2014-01-19" "2015-01-24"
## [6] "2011-01-29"
```

```
as.Date(ISOdate(years, 1, days))    # 재사용 규칙
```

```
## [1] "2011-01-04" "2012-01-09" "2013-01-14" "2014-01-19" "2015-01-24"
## [6] "2011-01-29"
```

날짜-시간 데이터에서 연, 월, 일, 요일, 시간 등의 세부 정보를 추출하려면 `as.POSIXlt()` 함수를 이용한다. 리턴값에서 각 세부 정보를 추출하는 방법은 리스트에서와 같이 `$`를 이용한다.

```
d <- as.Date("2015-10-09")
p <- as.POSIXlt(d)
p$year

## [1] 115

p$mon      # Month (0 = January)

## [1] 9

p$mday     # Day of the month

## [1] 9
```

- Date parts
 - `$sec`: Seconds (0-61)
 - `$min`: Minutes (0-59)
 - `$hour`: Hours (0-23)
 - `$mday`: Day of the month (1-31)
 - `$mon`: Month (0-11)
 - `$year`: Years since 1900
 - `$wday`: Day of the week (0-6, 0 = Sunday)
 - `$yday`: Day of the year (0-365)
 - `$isdst`: Daylight savings time flag

규칙적으로 연속된 날짜값의 열(sequence)를 만들 때도 `seq()` 함수를 이용한다.

아래 예제를 살펴보면 쉽게 이해할 수 있을 것이다.

```
s <- as.Date("2015-10-01")
e <- as.Date("2015-10-10")
seq(from=s, to=e, by=1)

## [1] "2015-10-01" "2015-10-02" "2015-10-03" "2015-10-04" "2015-10-05"
## [6] "2015-10-06" "2015-10-07" "2015-10-08" "2015-10-09" "2015-10-10"

seq(from=s, by=1, length.out=7)

## [1] "2015-10-01" "2015-10-02" "2015-10-03" "2015-10-04" "2015-10-05"
## [6] "2015-10-06" "2015-10-07"
```

```
seq(from=s, by="month", length.out=12)
## [1] "2015-10-01" "2015-11-01" "2015-12-01" "2016-01-01" "2016-02-01"
## [6] "2016-03-01" "2016-04-01" "2016-05-01" "2016-06-01" "2016-07-01"
## [11] "2016-08-01" "2016-09-01"

seq(from=s, by="3 months", length.out=4)
## [1] "2015-10-01" "2016-01-01" "2016-04-01" "2016-07-01"

seq(from=s, by="year", length.out=5)
## [1] "2015-10-01" "2016-10-01" "2017-10-01" "2018-10-01" "2019-10-01"
```

C. 데이터 입출력

C.1 데이터 입력

- 콘솔에서 직접 자료값을 입력하려면 `scan()` 함수 사용

```
x <- scan()
1: 1
2: 2
3: 3
4:
Read 3 items
> x
[1] 1 2 3
```

- 텍스트 파일의 내용을 읽어들이 백터로 저장할 때에도 `scan()` 함수 사용

```
x <- scan(file="c:/mydata/data_x.txt")
y <- matrix(scan("c:/mydata/data_y.txt"), ncol=3, byrow=T)
```

- 직사각형 형태로 정리된 텍스트 파일의 내용을 읽어들이 데이터프레임으로 저장하려면 `read.table()` 함수 사용

```
x <- read.table(file="table.txt", header=T, sep=" ")
```

- csv 파일을 읽어들이 데이터프레임으로 저장하려면 `read.csv()` 함수 사용

```
x <- read.csv(file="table.csv", header=T)
```

- SAS, SPSS 등의 데이터셋을 읽어들이려면 `foreign` 패키지의 함수들을 이용
 - `read.spss()` reads a file stored by the SPSS save or export commands

```
read.spss("data.sav", to.data.frame=T)
```

- `read.ssd()` generates a SAS program to convert the ssd contents to SAS transport format and then uses `read.xport` to obtain a data frame

- 내장 데이터셋(built-in datasets)

```
library(MASS)
data(geyser)
```

```
# Want the list of built-in datasets contained in the currently loaded packages? Just type data()
## data()
```

C.2 데이터 내보내기

- 콘솔에 출력: print() 함수 이용

```
x <- 1;3  
print(x)  
[1] 1 2 3
```

- 벡터를 파일로 내보내기: write() 함수

```
x <- seq(from=0, to=1, by=0.1)  
write(x, file="output.txt")
```

- 데이터프레임, 행렬 등과 같이 표 형태로 저장/작성된 데이터를 텍스트 파일로 내보내기: write.table() 함수 이용

```
x <- matrix(1:20, 4, 5)  
write.table(x, file="table.txt")
```

- 데이터프레임, 행렬 등과 같이 표 형태로 저장/작성된 데이터를 csv 파일로 내보내기: write.csv() 함수 이용

```
write.table(faithful, file="faithful.csv")
```

D. 간단한 도표를 이용한 자료 요약

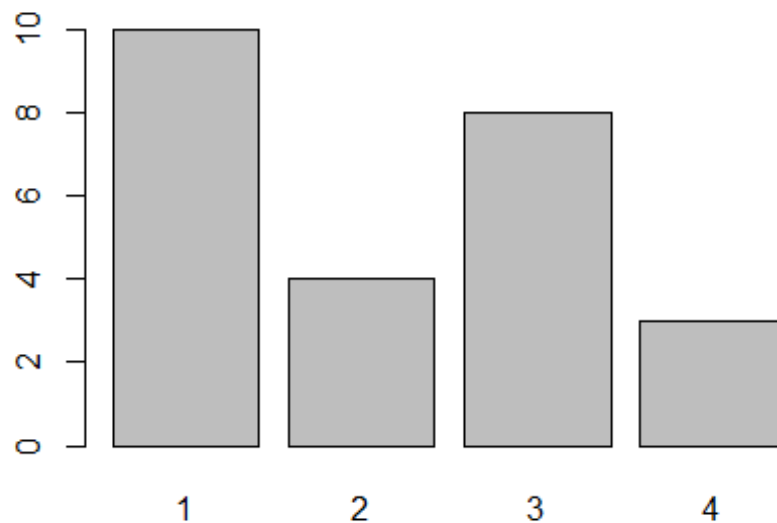
자료의 성격에 따라 적절한 도표 작성법을 사용해야 의미있는 정보가 담긴 도표를 작성할 수 있다.

D.1 질적 자료(Qualitative data)의 요약

- Bar chart: `barplot()` 함수 이용

```
# Beer Preference example
beer <- c(3, 4, 1, 1, 3, 4, 3, 3, 1, 3, 2, 1, 2,
         1, 2, 3, 2, 3, 1, 1, 1, 1, 4, 3, 1)
# (1) Domestic can   (2) Domestic bottle,
# (3) Microbrew      (4) Import

barplot(table(beer))
```

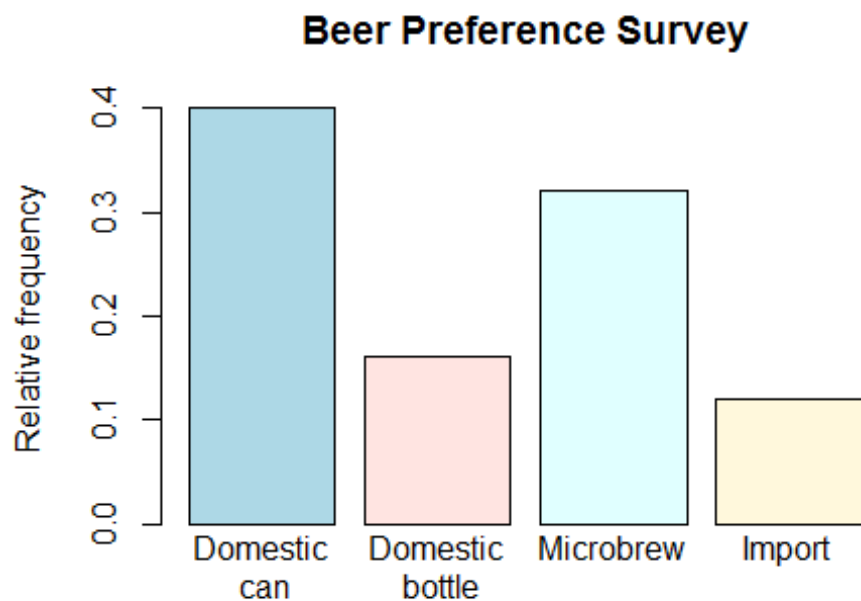


- `table()` 함수는 도수분포표를 작성
- 위 그림... 너무 무성의한 도표임. 약간의 화장을 하면...

```

barplot(table(beer)/length(beer), # 상
대도수
        col=c("lightblue", "mistyrose", "lightcyan", "cornsilk"), # 막
대 색깔
        names.arg=c("Domestic\n can", "Domestic\n bottle", "Microbrew\n",
"Import\n"), # 막대 라벨
        ylab="Relative frequency",
        main="Beer Preference Survey")

```

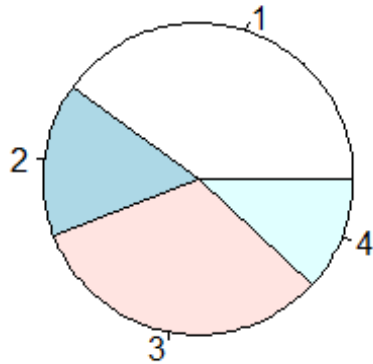


- names.arg= 옵션의 \\n 는 줄바꿈을 하라는 의미
- Pie chart: pie() 함수 이용

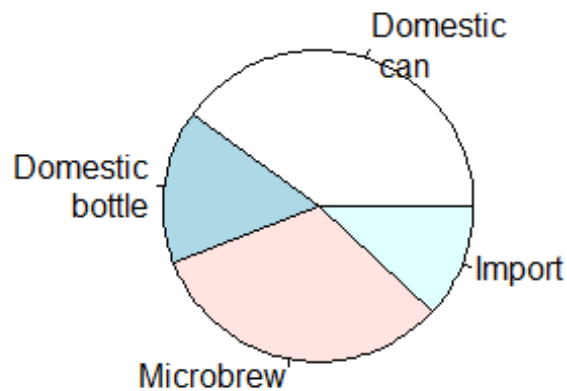
```

beer.counts <- table(beer) # store the table result
pie(beer.counts) # first pie -- kind of dull

```



```
names(beer.counts) <- c("Domestic\n can", "Domestic\n bottle", "Microbrew",
"Import") # give names
pie(beer.counts) # prints out names
```



D.2 양적 자료(Quantitative data)의 요약

- Stem-and-leaf plot: `stem()` 함수 이용

```
scores <- c(2, 3, 16, 23, 14, 12, 4, 13, 2, 0, 0, 0, 6, 28, 31, 14, 4, 8,
2, 5)
stem(scores)

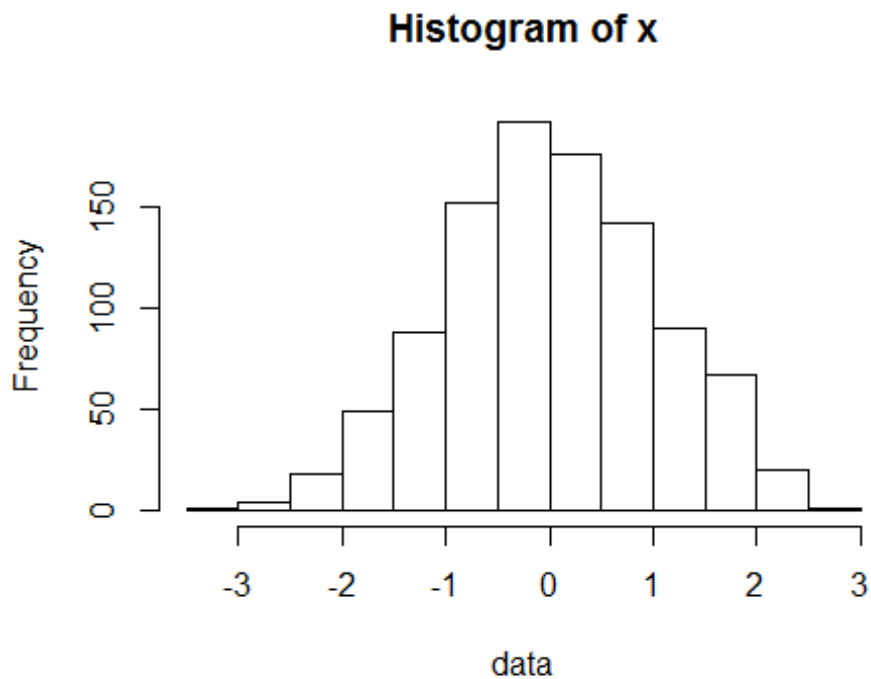
##
## The decimal point is 1 digit(s) to the right of the |
##
## 0 | 000222344568
```



```
## 1 | 23446
## 2 | 38
## 3 | 1
```

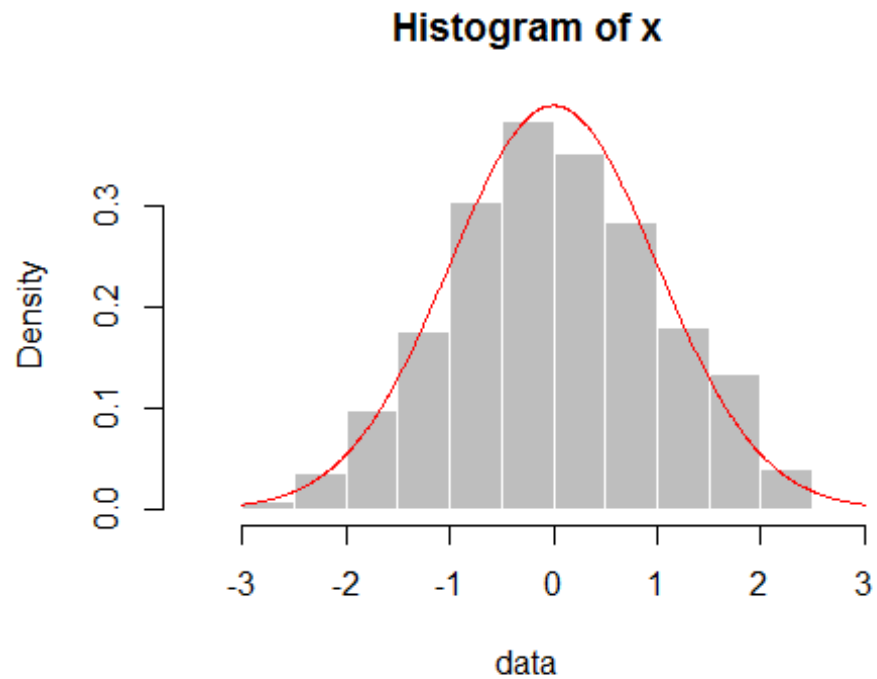
- Histogram: hist() 함수 이용

```
x <- rnorm(1000) # To generate 1,000 random numbers from N(0,1)
hist(x, xlab="data")
```



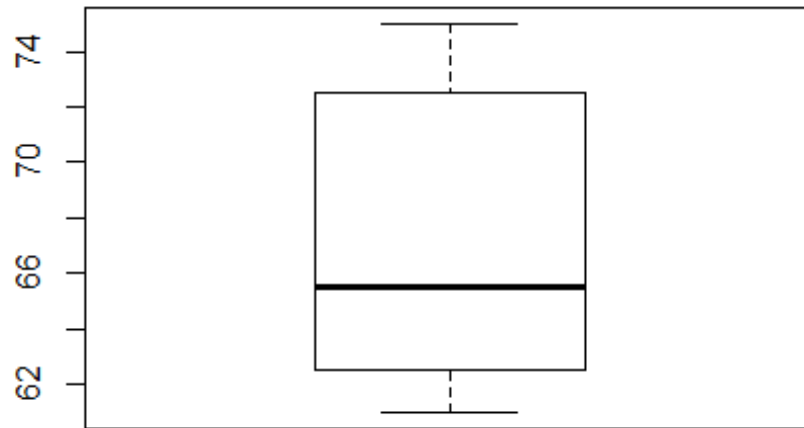
- 대체로 난수의 질이 괜찮아보임. 보다 확실히 알아보기 위해
상대도수히스토그램을 target 분포인 $N(0,1)$ 의 밀도함수와 비교

```
hist(x, probability=T, xlab="data", col="gray", border="white")
z <- seq(from=-3, to=3, by=0.01) # N(0,1)의 밀도함수 곡선을 위한 grid
                                  # 잡기
lines(z, dnorm(z), col=2)         # 빨간 색으로 N(0,1)의 밀도함수 곡선
                                  # 덧그리기
```



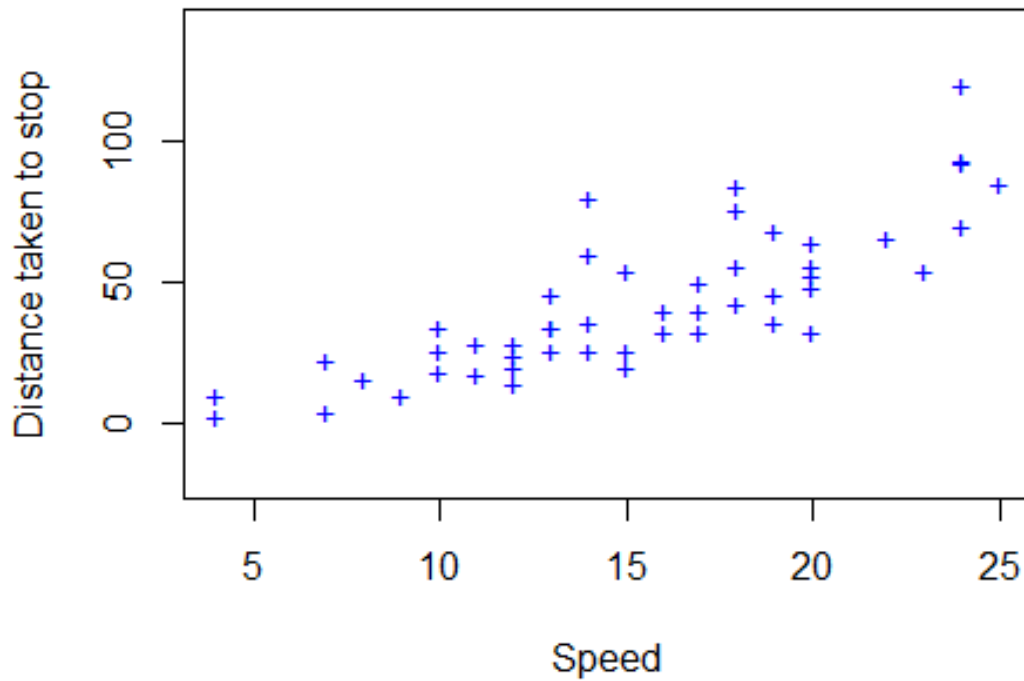
- `lines()` 함수는 기존의 plot 위에 선을 추가하는 함수임
- Boxplot: `boxplot()` 함수 이용

```
growth <- c(75,72,73,61,67,64,62,63) # the size of flies
sugar <- c("C","C","C","F","F","F","S","S") # diet
fly <- data.frame(growth=growth, sugar=sugar)
boxplot(fly$growth)
```



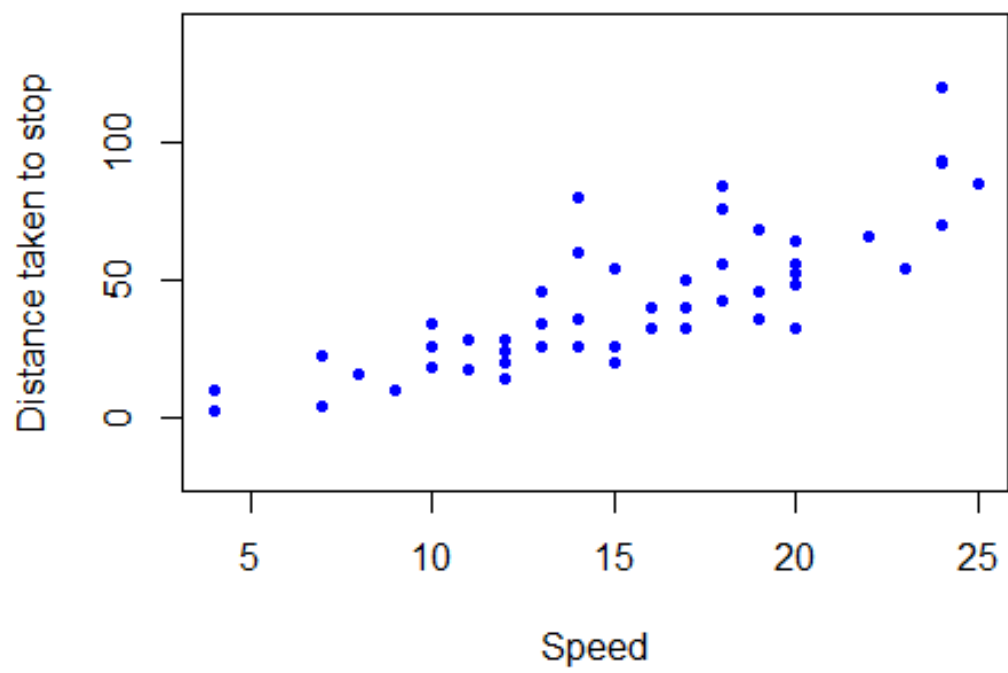
- Scatterplot: plot() 함수 이용

```
# cars: built-in dataset with the speed of cars and the distances taken to stop  
plot(cars$speed, cars$dist,  
      col="blue", pch="+",  
      ylab="Distance taken to stop", xlab="Speed",  
      ylim=c(-20, 140))
```



- with() 함수를 사용하면 변수명에 cars\$를 사용하지 않아도 되기 때문에 보다 깔끔한 코딩 가능

```
with(cars, plot(speed, dist,  
                col="blue", pch=20,  
                ylab="Distance taken to stop", xlab="Speed",  
                ylim=c(-20, 140)))
```



E. 프로그래밍

이 절에서는 반복적인 작업을 단순화하거나 코드를 보다 읽기 좋도록 하기 위해 R 프로그래밍에서 사용하는 제어문들을 익힌다.

E.1 조건문

- 'if-else'를 사용한 조건문

```
x <- 10
if (x < 3) print("x < 3") else print("x >= 3")

## [1] "x >= 3"
```

- ifelse() 함수

```
y <- -3:3
z <- ifelse(y < 0, -1, 1)
cbind(y, z)

##      y  z
## [1,] -3 -1
## [2,] -2 -1
## [3,] -1 -1
## [4,]  0  1
## [5,]  1  1
## [6,]  2  1
## [7,]  3  1
```

- Commands can be grouped by braces.

```
x <- 4
if ( x < 3 ) {
  print("x<3")
  z <- "M"
} else {
  print("x>3")
  z <- "F"
}

## [1] "x>3"
```

E.2 반복문

루프(loop)란 반복적으로 실행되는 명령 사이클을 의미한다.

R 에서 루프를 사용할 때 대표적 방법은 `for` 문을 사용하는 방법과 `while` 문을 사용하는 방법이 있다.

- `for` 문: `for(i in 벡터){명령어...}` 와 같은 형태로 사용.

벡터 내 모든 성분에 대해 루프를 돌림

- 아래 코드는 `for` 문을 사용해 1 부터 10 까지의 합을 계산하고 루프를 돌리는 동안 부분합을 화면에 출력하는 예이다.

```
n <- 10
x <- 1:n
sum.so.far <- 0
for ( i in 1:n ) {
  sum.so.far <- sum.so.far + x[i]
  print(sum.so.far)
}

## [1] 1
## [1] 3
## [1] 6
## [1] 10
## [1] 15
## [1] 21
## [1] 28
## [1] 36
## [1] 45
## [1] 55

sum.so.far

## [1] 55
```

- `while` 문: `while(조건){명령어...}` 와 같은 형태로 사용.

반복 횟수를 미리 알 수 없고 대신 루프를 지속할 수 있는 조건만 줄 수 있을 때 사용

- 아래 코드는 1 부터 시작해 자연수를 누적 합을 구하되 누적 합이 1,000 이내인 동안만 루프를 돌리고 1,000 을 초과하면 루프를 중지하는 예임

```
n <- 0
sum.so.far <- 0
while ( sum.so.far <= 1000 ) {
  n <- n+1
  sum.so.far <- sum.so.far + n
}
```

```

}
print(c(n, sum.so.far))

## [1] 45 1035

sum(1:45)

## [1] 1035

```

- 루프를 강제로 빠져나오게 하려면 break 문 사용
 - 아래 코드는 1 부터 시작해 자연수를 누적 합을 구하되 누적 합이 1,000 이내인 동안만 루프를 돌리는 예임

```

n <- 0
sum.so.far <- 0
while (1) {           # 조건문 자리에 1(TRUE 와 동일한 효과)이 있으므로 무한루프
  if
  if(sum.so.far > 1000) break
  n <- n+1
  sum.so.far <- sum.so.far + n
}
print(c(n, sum.so.far))

## [1] 45 1035

```

- 가능하면 루프는 사용하지 않는 것이 좋음
 - 벡터 연산 혹은 행렬 연산으로 해결할 수 있는 작업은 루프를 사용하지 않아야 함
 - 계산 시간을 절약하고 코드의 가독성을 향상시키기 위함임
 - 아래 코드는 1 부터 100,000,000 까지의 자연수로 이루어진 벡터를 다른 이름의 벡터로 복사하는 예임. 시스템 시간을 비교하기 바람.

```

n <- 100000000
x <- 1:n
system.time(y <- x)

##      user      system elapsed
##       0         0         0

system.time(for (i in 1:n) y[i] <- x[i])

##      user      system elapsed
## 139.75      0.25    143.30

```


E.3 나만의 R 함수 만들기

함수(function)란 특정한 작업을 해 결과물을 객체로 되돌려주는 코드 덩어리를 의미한다.

R의 여러 패키지에서 제공하는 함수를 이용하는 것으로 충분하지 않아서 직접 새로운 함수를 만들어 사용해야 하는 경우를 종종 만나게 된다.

- 함수의 정의 방법: 함수이름 <- function(함수 인자...) {명령어들...}
 - 아래 코드는 자료의 평균과 표준편차를 계산해 리턴하고 자료의 분포를 요약한 상자그림 및 히스토그램을 작성하는 함수의 예이다.

```
my.stat <- function(x)
{
  m <- mean(x)
  s <- sd(x)
  res <- list(m=m, s=s)

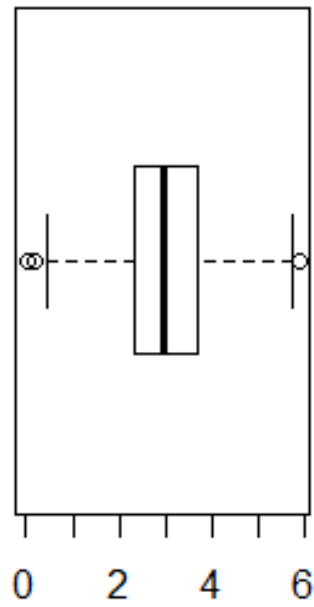
  par(mfrow=c(1, 2))
  boxplot(x, main="Boxplot", horizontal=T)
  hist(x, prob=T, col="skyblue", border="white", main="Histogram", xlab=
"data")

  return(res)
}
```

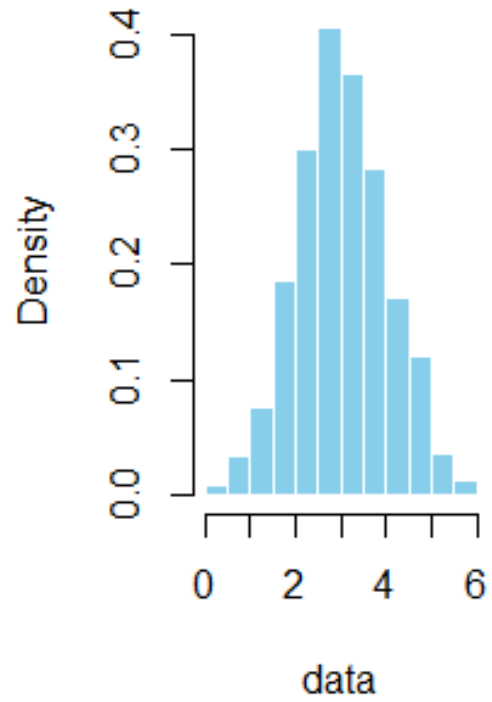
- 함수를 정의하는 {} 안에서 정의된 변수의 이름은 함수 정의 내에서만 유효한 로컬 변수임에 유의
 - 위의 예에서 사용한 m, s, res 등은 함수 정의 내에서만 유효한 로컬 변수
- 위에서 작성한 새로운 함수를 호출해 사용해 보자.

```
dat <- rnorm(1000, mean=3, sd=1)
my.stat(x=dat)
```

Boxplot



Histogram



```
## $m
## [1] 3.022449
##
## $s
## [1] 0.9992066
```