# R 기초입문

## R에 대해

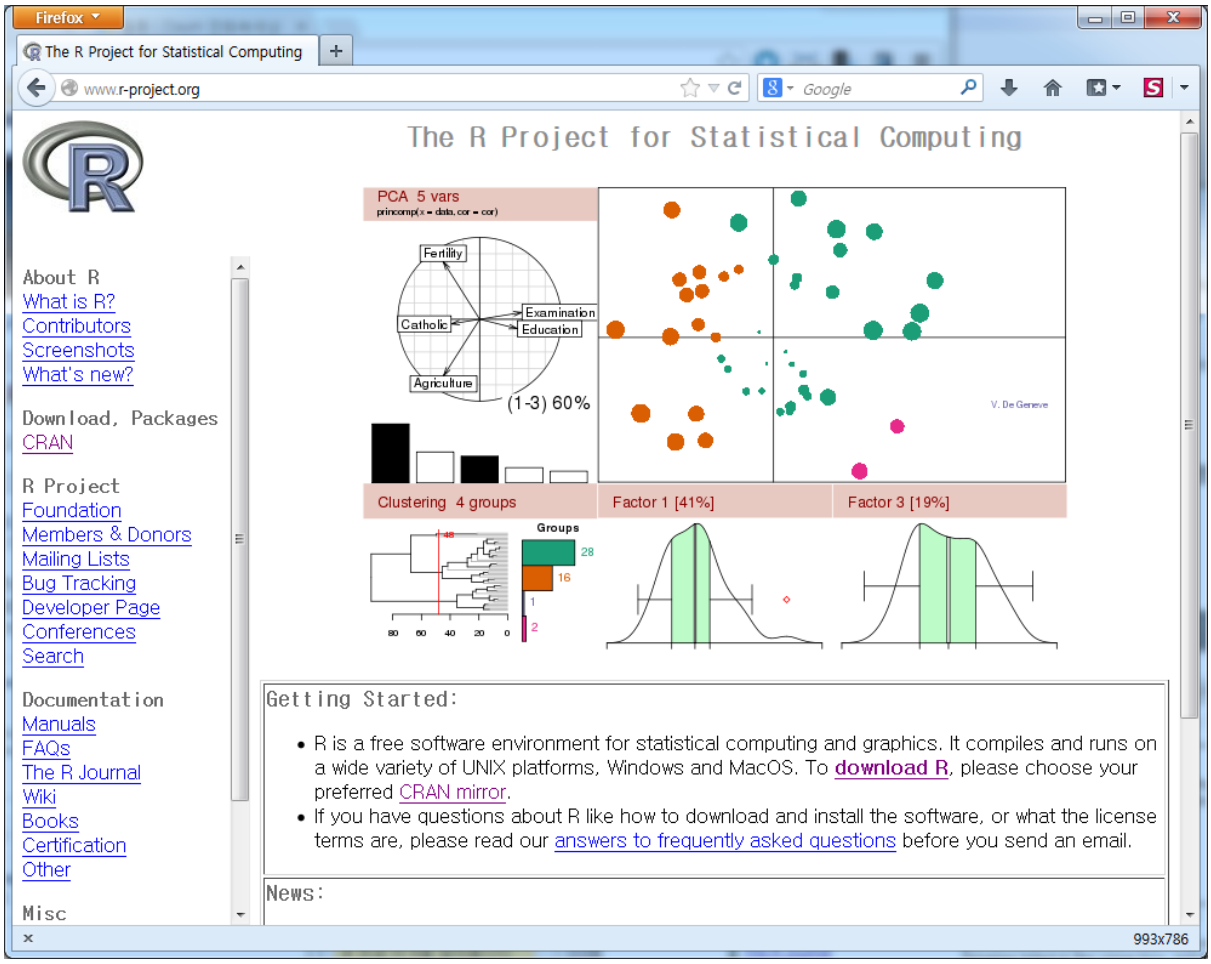
R 프로그래밍 언어 (줄여서 R) 는 통계 계산과 그래픽을 위한 프로그래밍 언어이자 소프트웨어 환경으로 뉴질랜드 오클랜드 대학의 로스 이하카와 로버트 젠틀맨에 의해 개발되었으며 GPL 하에 배포되어 비용에 부담없이 자유롭게 사용할 수 있습니다. R은 통계 소프트웨어 개발과 자료 분석에 널리 사용되고 있으며, 패키지 개발이 용이하여 통계학자들 뿐만 아니라 각종 계량 연구를 하는 분야에서 널리 사용되고 있습니다.

또한 R은 많은 연구자들에 의해 새롭게 만들어진 최신의 알고리즘과 로직들을 Package 형태로 제공하여 다른 어떤 통계 소프트웨어들보다도 다양한 분석방법 등을 제공합니다.

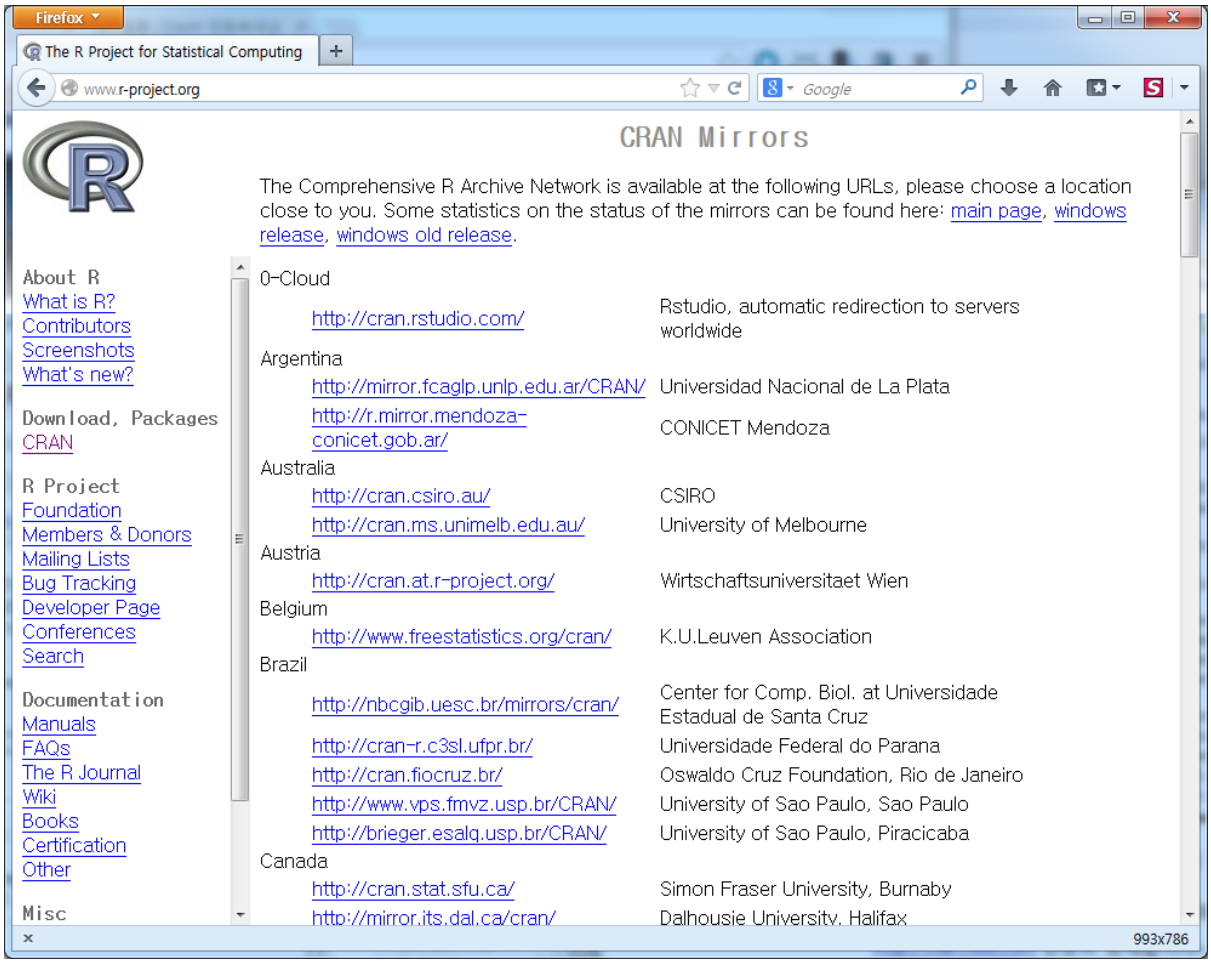
기본적으로 Command Line 에서 입력하는 방식을 취하고 있어 초기 접근이 다른 통계 소프트웨어보다 불편하게 느껴질 수 있지만 익숙해지면 그 어떤 것보다 편하게 다룰 수 있습니다.

## R 다운로드와 설치

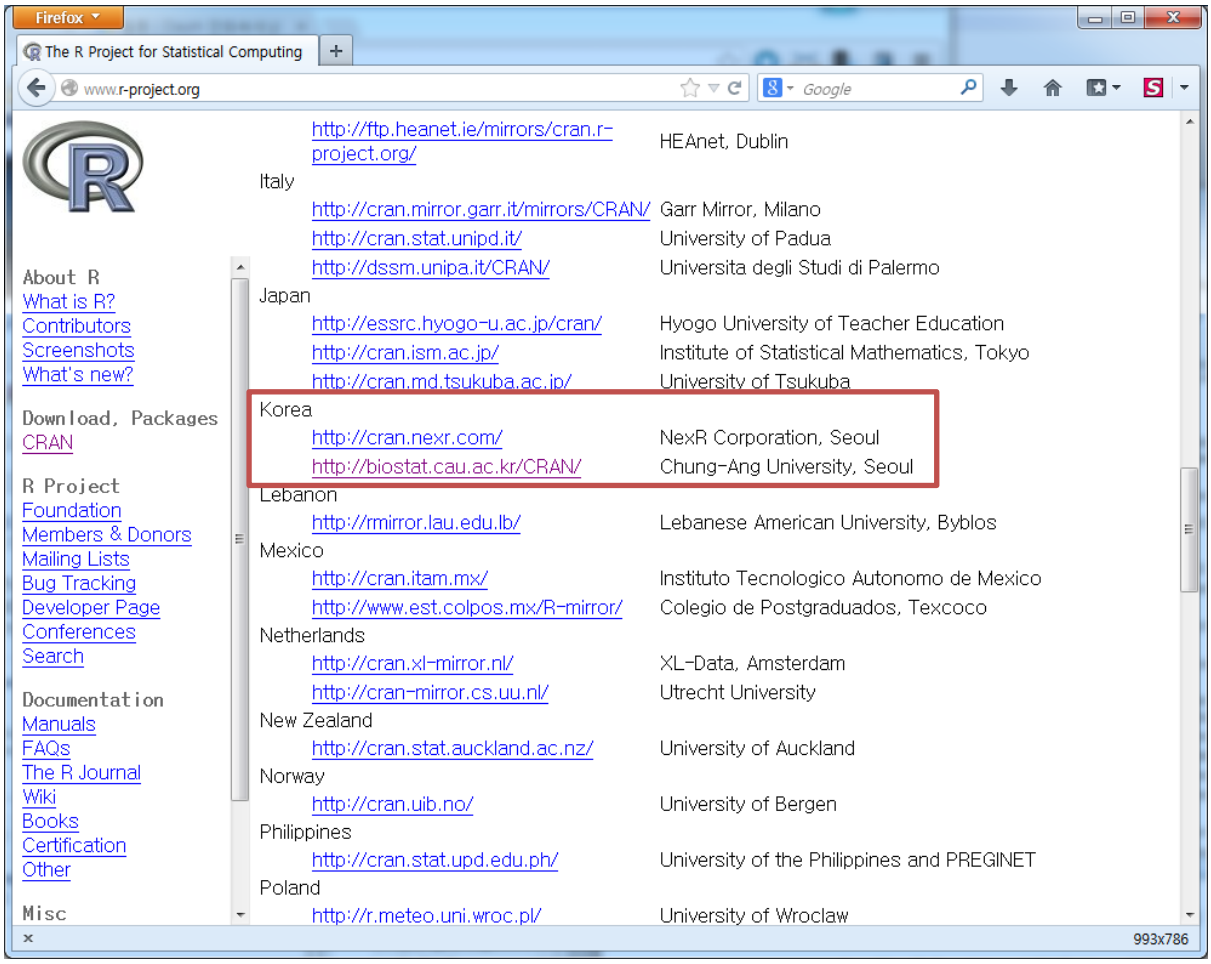
R 의 공식 홈 페이지 http://www.r-project.org 입니다. 이곳은 R 설치파일을 다운로드 받을 수 있으며 R이 제공하는 각종 문서들을 함께 볼 수 있습니다. R의 설치는 이곳에서 설치파일을 아래의 설명과 같이 다운로드 받아 사용하시면 됩니다.(Windows 기반 환경을 기준으로 설명드리겠습니다.)



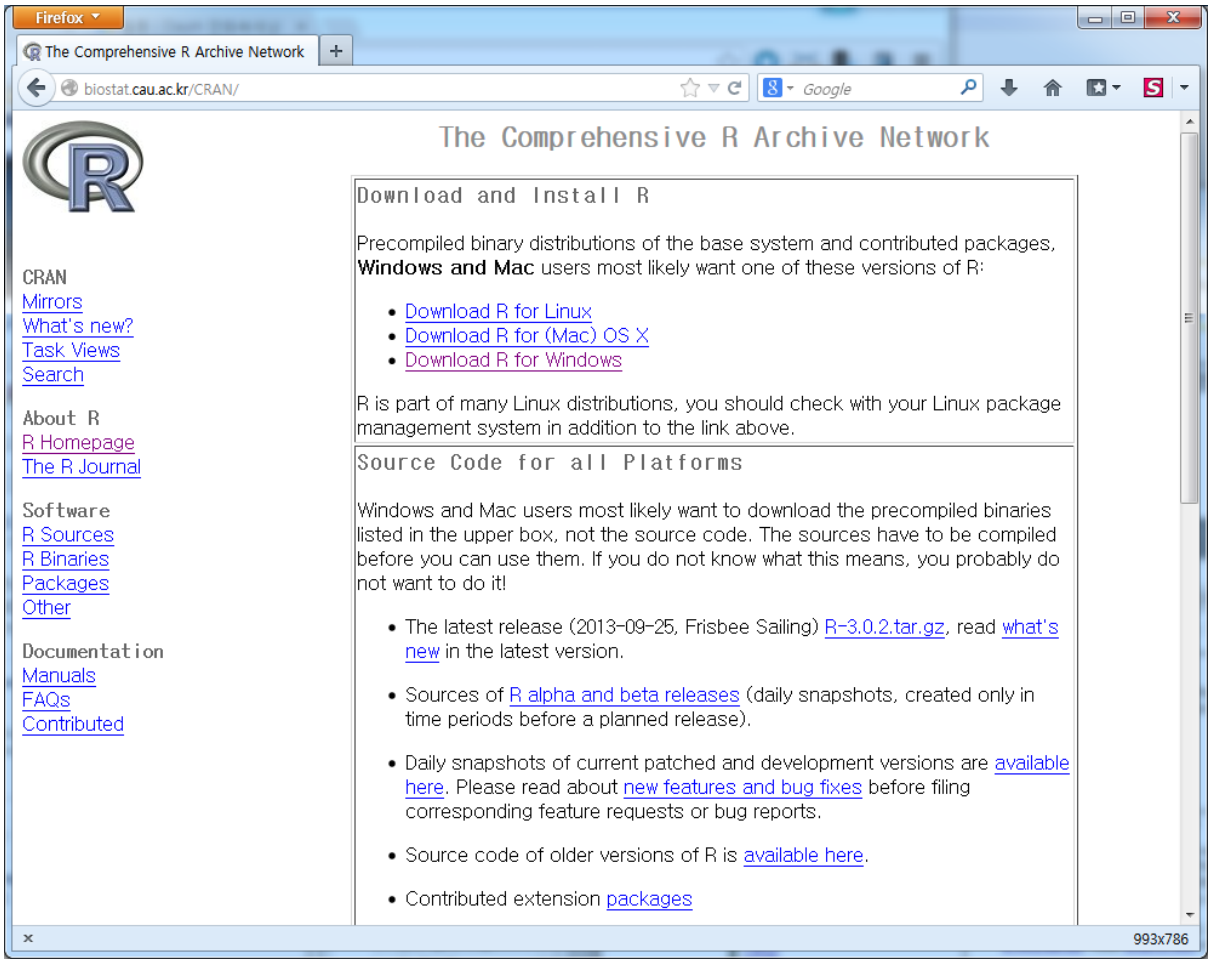

- 웹 브라우저의 주소창에서 http://cran.r-project.org 를 입력 혹은 R 홈페이지 좌측 메뉴에서 Download, Packages 섹션의 "CRAN" 을 클릭합니다.
- R 홈페이지에서 CRAN을 클릭하여 이동하면 다음과 같이 다운로드 받을 위치를 선택하도록 합니다.
  - ■ 지리적으로 가까운 곳을 선택하는 것이 좋습니다.



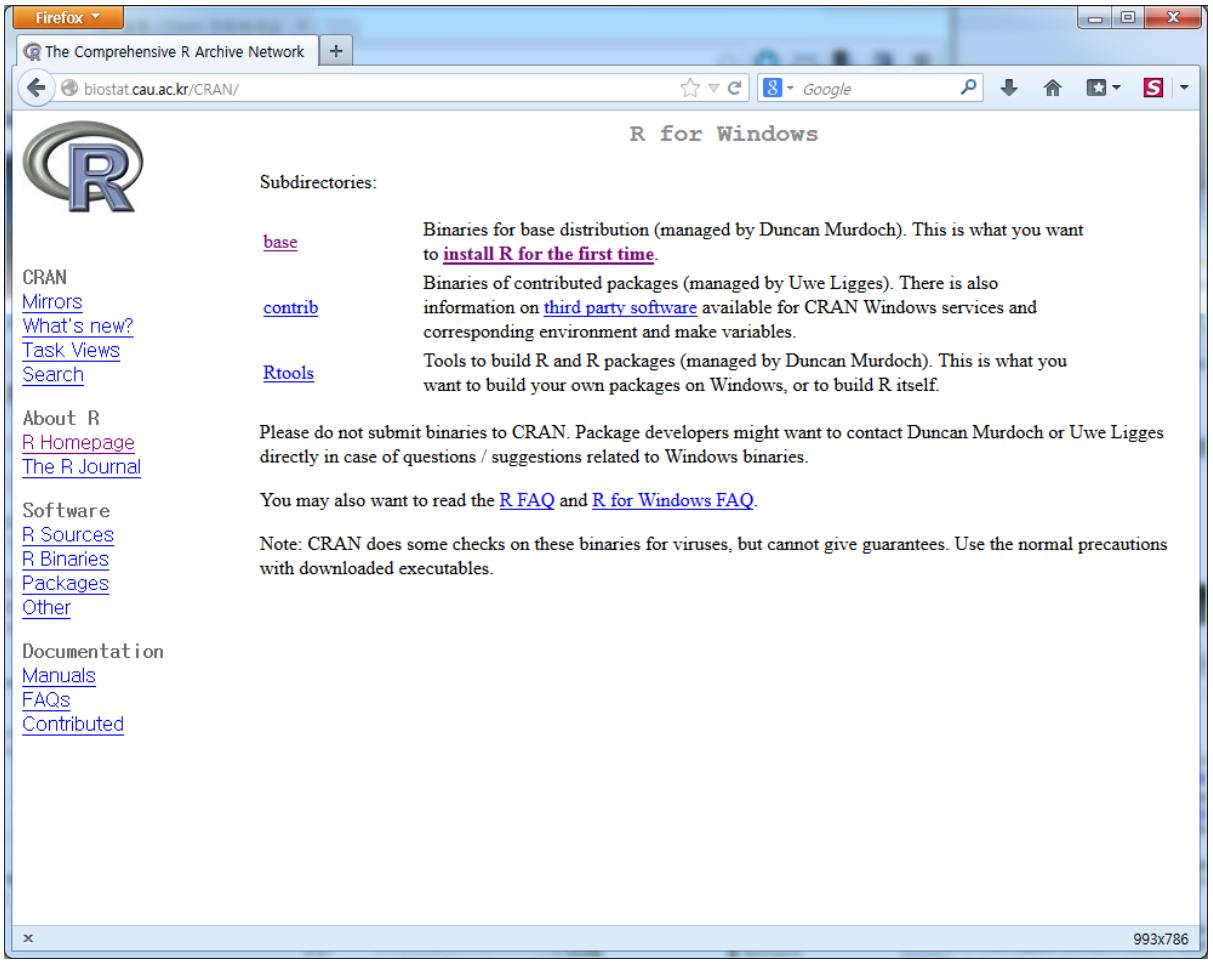

- 우리나라의 경우 "Korea" 로 되어 있는 곳을 찾아가시면 됩니다.
  - 이 글을 쓰는 시점에서는 우리나라에서 다운로드를 받을 수 있는 곳은 http://cran.nexr.com 과 http://biostat.cau.ac.kr/CRAN/ 의 두 곳이 있습니다.



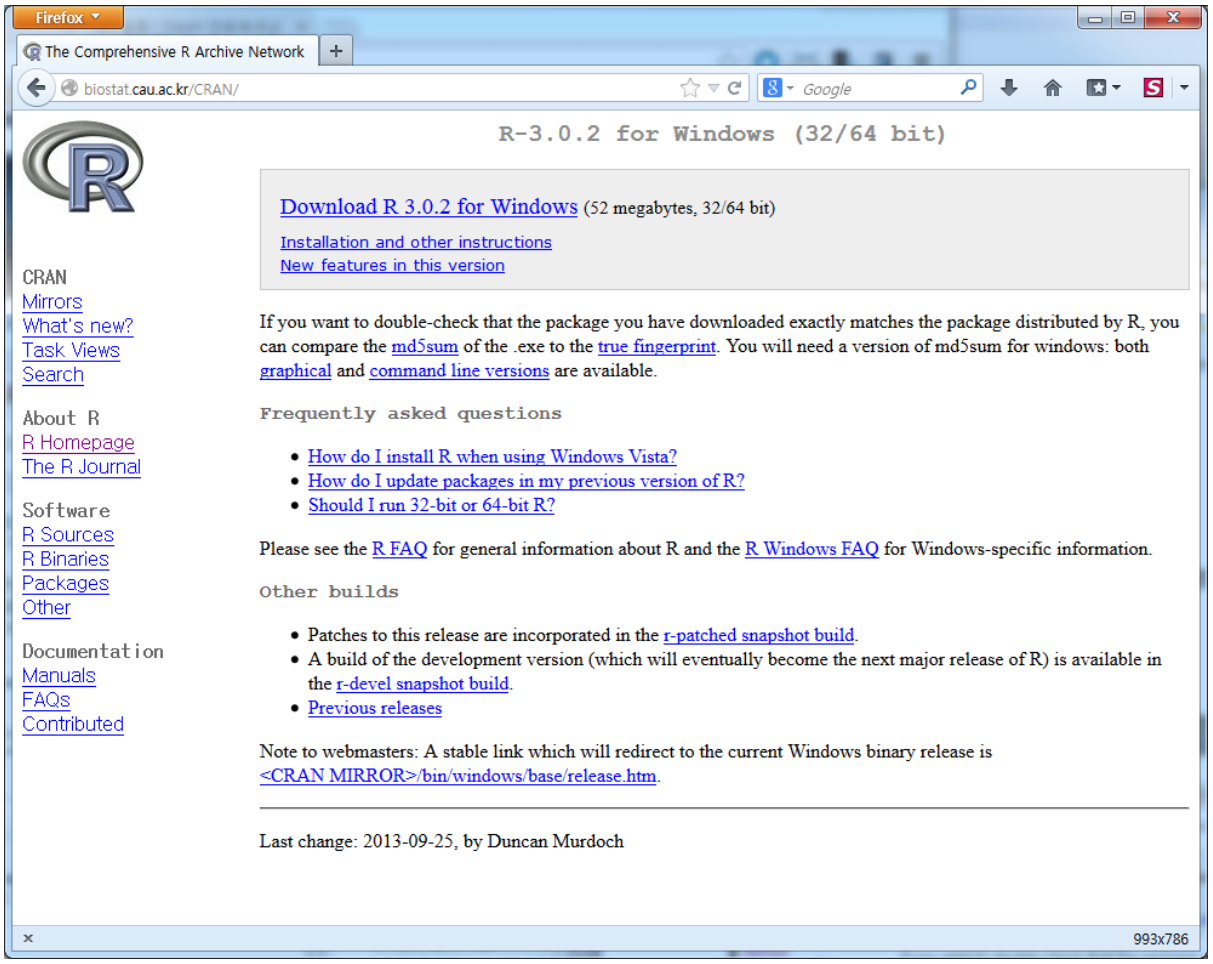

- cran.r-project.org를 직접 입력하거나 다운로드 받을 곳을 선택하시거나 둘중에 하나를 선택해서 이동하시면 다음과 같은 화면이 나옵니다.




- R을 실행할 수 있는 운영체제는 Linux, MacOS X, Windows로 왠만한 PC 운영체제를 거의 다 지원합니다. 일단 이 글에서는 Windows를 기준으로 설명드리도록 하겠습니다.
  - ■ "Download R for Windows" 를 클릭하여 이동합니다.




- 화면에 보시면 "base", "contrib" 그리고 "Rtools" 중에 선택할 수 있습니다. 각각은 다음과 같은 파일들을 받게 해 줍니다.
  - base : R의 설치 파일을 다운로드 받습니다. R을 설치하고자 하는 경우 이곳을 통해 설치파일을 다운로드 받습니다.
  - contrib : R 자체에도 많은 기능이 있습니다만, 사용자들이 사용하면서 만든 추가적인 package 들을 다운로드 받습니다.
  - Rtools : R에서 사용할 package 들을 제작하고자 할 때 사용되는 도구들을 다운로드 받습니다.
- 설치를 위해 base 를 클릭합니다.

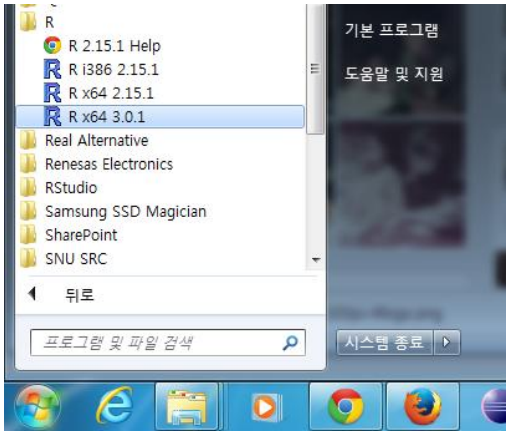



- R을 실제로 다운로드 받는 페이지로 간단한 도움말과 함께 다운로드 받기 위한 링크를 제공합니다.
  - 상단의 "Download R x.xx.x for Windows"를 클릭하면 바로 다운로드를 받을 수 있습니다.
  - 클릭하여 파일을 다운로드 받은 후 실행하시면 됩니다.

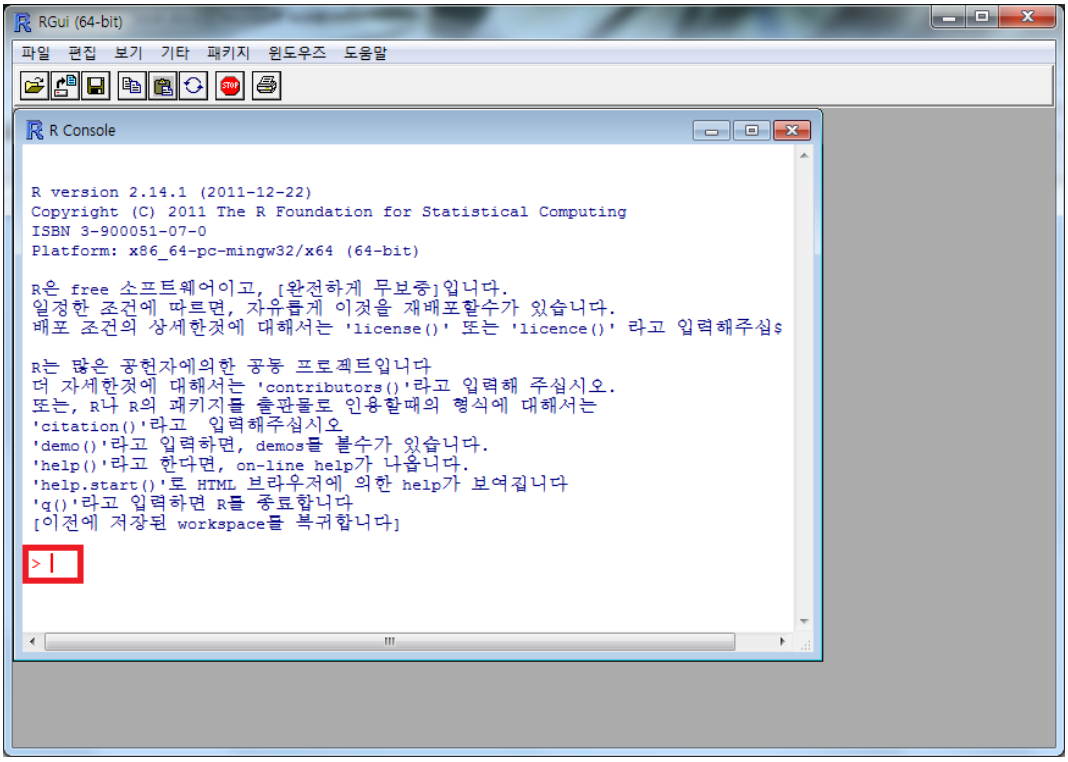
# R 살펴보기

R 설치를 완료하셨으면 이제 R을 실행하고 둘러보겠습니다.

R 을 실행하는 것은 다른 Windows용 응용프로그램과 다르지 않습니다. 바탕화면 R 아이콘 이 있다면 아이콘을 클릭하여 실행하시면 됩니다. 바탕화면 아이콘이 없다면 윈도우 좌측 하단의 시작탭을 클릭하고 프로그램을 목록 살펴보시면 R 디렉토리가 있습니다. 여기서 설치된 R을 클릭하여 실행하시면 됩니다. (이 화면의 경우 서로 다른 버전의 R이 설치되어 있는 관계로 몇 개의 R 실행파일이 보입니다)

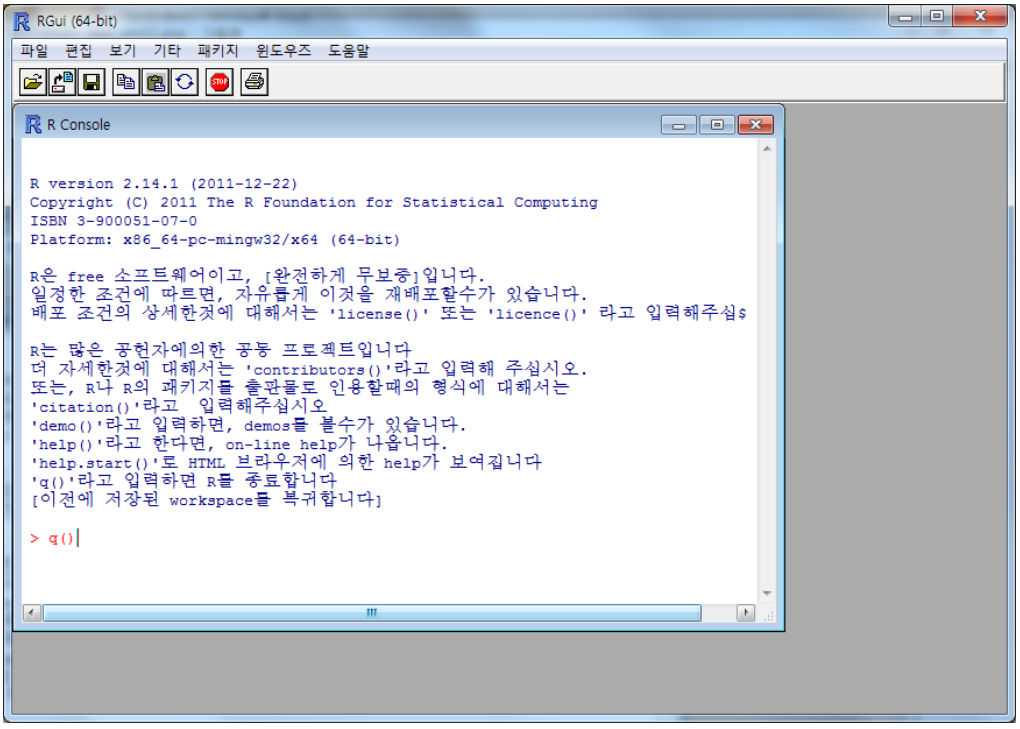



R을 실행시키시면 다음과 같은 화면이 나옵니다.

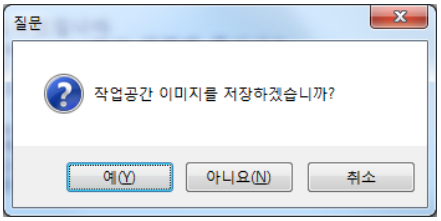



R을 실행시킨 화면은 윈도우 안에 또 하나의 윈도우 창이 있는 모습으로 안에 있는 입력창을 R 코드를 입력하는 R Console 창이라고 부릅니다. 이 창안 붉은 사각형이 사용자가 입력하는 부분입니다(Prompt라고 부릅니다).

프로그램은 실행만큼 중요한 것이 종료하는 것입니다. 이제 우리가 실행시킨 R을 종료해보겠습니다. 앞서 말씀드린 붉은 사각형 안에 종료 명령인 q( )를 다음과 같이 입력해 봅시다.

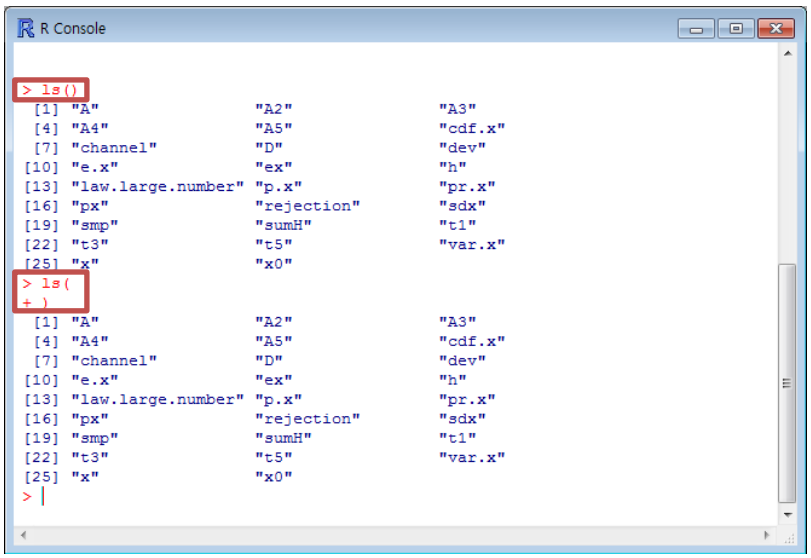



다음과 같은 확인창이 생성될 것입니다. 이 창의 기능은 사용자가 현재 생성해 놓은 자료등을 보관할 것인지 물어보는 것으로 연속된 작업중일 경우 "예"를 눌러 현재 사용한 자료등을 다음 번 실행에서도 다시 생성하지 않고 계속해서 작업할 수 있도록 합니다.



# R을 시작합니다.

R 은 명령어를 직접 입력하고 해당 명령에 대한 응답을 출력으로 사용합니다. 먼저 앞서 보신 R Console 창에 명령을 입력합니다. 이 때 주의하실 점은 R 은 정상적인 입력을 기다리고 있을 때 프롬프트로 부등호 보다 크다('>')를 사용합니다. 만일 프롬프트가 > 가 아닌 플러스 기호('+') 일 경우 하나의 명령이 완전히 끝나지 않았음을 의미합니다. 이런 경우 각종 괄호('{', '[', '('')가 닫히지 않았거나 문자열을 나타내는 큰 따옴표('"')가 닫히지 않은 경우 나올 수 있습니다. 다음 화면을 보시면 현재 R 이 갖고 있는 각종 자료들을 나타내는 함수 ls() 를 입력하여 제대로 출력된 경우와 작은 괄호('('')를 열고 닫지 않았을 경우의 프롬프트입니다. 잘 비교해 주시기 바랍니다.




R 은 기본적으로 한 줄에 하나씩 명령을 입력합니다. 하나의 명령이 끝났음을 알리는 것은 줄바꿈(Enter 키)가 합니다. 위에 입력한 ls( ) 도 입력후 줄바꿈 한 것입니다.

계산기로써의 R

산술연산

각종 연산을 위한 연산자를 사용하여 계산기 기능으로써의 R 을 알아보겠습니다.

먼저 산술 연산 결과를 반환하는 산술 연산자에 대해 알아보겠습니다. 다음은 R 의 산술연산자에 대한 표 입니다.

| 연산자 | 설명 | 예제 | 결과 |
|---|---|---|---|
| + | 더하기 | 3 + 2 | 5 |
| - | 빼기 | 3 - 2 | 5 |
| * | 곱하기 | 3 * 2 | 6 |
| / | 나누기 | 3 / 2 | 1.5 |
| ^ or ** | 승수 | 3 ^ 2 | 9 |
| x %% y | X 를 y 로 나눈 나머지 값 반환 | 3 %% 2 | 1 |
| x %/% y | 나누기의 결과를 정수로 | 3 %/% 2 | 1 |

다음은 R 의 입력과 출력 결과입니다.

```
> 3 + 2
[1] 5
> 3 - 2
[1] 1
> 3 * 2
[1] 6
> 3 / 2
[1] 1.5
> 3 ^ 2
[1] 9
> 3 %% 2
[1] 1
> 3 %/% 2
[1] 1
```

논리연산

참과 거짓에 대한 논리 연산을 R 에서 실시할 수 있으며 논리연산의 결과는 TRUE 아니면 FALSE 로 나타납니다. 다음은 R 에서 지원하는 논리연산자입니다.

| 연산자 | 설명 | 예제 | 결과 |
|---|---|---|---|
| < | 좌변이 보다 작은 | 5 < 5 | FALSE |
| <= | 좌변 이하 | 5 <= 5 | TRUE |
| > | 좌변이 보다 큰 | 5 > 5 | FALSE |
| >= | 좌변 이상 | 5 >= 5 | TRUE |

| | | | |
|---|---|---|---|
| **==** | 값이 같은 | 5 == 5 | TRUE |
| **!=** | 값이 다른 | 5 != 5 | FALSE |
| **!x** | 부정형 연산 | !TRUE | FALSE |
| **x \| y** | x OR y | TRUE \| FALSE | TRUE |
| **x & y** | x AND y | TRUE & FALSE | FALSE |
| **isTRUE(x)** | X 의 TRUE 여부조사 | isTRUE(TRUE) | TRUE |

다음은 위의 표에 있는 명령을 입력하고 얻은 출력 결과입니다.

```
> 5 < 5
[1] FALSE
> 5 <= 5
[1] TRUE
> 5 > 5
[1] FALSE
> 5 >= 5
[1] TRUE
> 5 == 5
[1] TRUE
> 5 != 5
[1] FALSE
> x <- TRUE
> !x
[1] FALSE
> y = FALSE
> x | y
[1] TRUE
> x & y
[1] FALSE
> isTRUE(x)
[1] TRUE
```

변수의 사용

R 은 계산기로써의 기능 외에 프로그래밍 언어로써의 기능도 완벽히 지원합니다. (물론 R 자체가 R 언어를 의미합니다.) 프로그래밍 언어로써 R 은 자주 사용하는 값과 현재 상태 등을 저장하기 위해 변수를 사용합니다. 앞서 예로 사용한 "**x <- TRUE**", "**y = FALSE**" 에서 x, y 가 각각 변수입니다. 변수는 연산자 "<-" 나 "=" 을 통해서 해당 연산자의 우변의 값을 좌변에 대입하는 것으로 값을 초기화하고 변경할 수 있습니다. 다음은 여러 변수들을 만들고 사용하는 예제입니다. (R 에서 # 이후는 주석 처리되어 줄바꿈하기 전까지 아무 기능을 하지 않습니다.)

```
> x <- 3     # 변수 x 에 값 3을 대입합니다.
> x          # 변수 x의 값을 사용(값 부르기)합니다.
[1] 3
> x <- 5     # 변수 x의 값을 5로 변경합니다.
> x
[1] 5
> y = 3
> temp <- y
➔ 변수 x와 y의 값을 서로 바꾸기 위해 새로운 변수 temp에 y의 값을 대입합니다.

> temp
[1] 3
➔ temp 는 y 의 값 3을 가집니다.

> y
[1] 3

> y <- x      # 변수의 y에 x 가 갖고 있는 값을 대입합니다.
> y
[1] 5
> x
[1] 5
> x <- temp      # 변수 x 에 temp가 갖고 있던 y의 기존 값을 대입합니다.

> x
[1] 3
> y
[1] 5
➔ x와 y가 갖고 있는 값이 성공적으로 바뀌었습니다.
```

R의 기본 자료형

자료 저장을 위해 R 에서 사용하는 기본 자료형은 하나의 객체이며 그 종류는 다음과 같습니다. (괄호 안의 영문은 R 내부에서 사용하면 자료 객체의 이름입니다.)

- 숫자형
  - 정수 (integer)
  - 실수 (numeric)
  - 복소수 (complex)
- 문자형 (character)
  - 큰따옴표로 둘러 쌓아 표현 : "abc", "123" 등
- 논리형 (logical)
  - TRUE(T), FALSE(F)
- 특수한 상태를 나타내는 상수
  - NULL : 정의되지 않은 값
  - NA : Missing Value
  - -Inf, Inf : 음과 양의 무한대
  - NaN : 수의 연산에서 불능의 경우 표현 0/0, Inf/Inf 등

다음은 각 기본 자료형의 사용 예 입니다.

```
> x <- 3
> y <- 2
> x / y
[1] 1.5
> xi <- 1 + 2i
> yi <- 1 - 2i
> xi + yi
[1] 2+0i
> str <- "string"
> str
[1] "string"
> TRUE
[1] TRUE
> T
[1] TRUE
> FALSE
[1] FALSE
> F
[1] FALSE
> xinf <- Inf
> yinf <- -Inf
> xinf / yinf
[1] NaN
```

자료형을 위한 R 내장 함수

변수 등을 위해 다음과 같은 R 내장 함수를 갖추고 있습니다. 자료에 대한 판단과 변환 등을 통해 필요한 작업을 원활히 할 수 있도록 도와줍니다.

- 자료형 정보 함수 : 함수별로 비슷한 결과를 가져오지만 각각 차이가 있습니다.
  - class() : R 객체지향 관점에서의 자료형
  - typeof() : 원시 자료형 (R 에서의)
  - mode() : 원시 자료형 (S[1] 에서의)
- 자료형 확인 함수(is."RClass"명())
  - is.integer(x) : x 가 정수형이면 TRUE, 아니면 FALSE
  - is.numeric(x) : x 가 실수형이면 TRUE, 아니면 FALSE
  - is.complex(x) : x 가 복소수형이면 TRUE, 아니면 FALSE
  - is.character(x) : x 가 문자열이면 TRUE, 아니면 FALSE
  - is.na(x) : x 가 NA 이면 TRUE, 아니면 FALSE

함수 (Function, method)

변수는 자료를 저장하는 역할을 한다면 함수는 행위를 나타냅니다. 예를들어 "1부터 10까지의 합을 구하라"는 행위를 R에게 하기위해 합을 구하는 행위에 맞는 함수의 이름을 표시하고 합을 구하기 위해 필요한 자료를 전달해 주어야 합니다. 이를 위해 R은 "함수명(필요로 하는 데이터, ...)"의 구조를 통해 원하는 행위의 결과를 얻습니다.

다음은 1부터 10까지 합을 구하는 함수의 사용예제입니다.

```
> sum(c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10))
[1] 55
```

함수 sum은 주어진 값들의 합을 구하는 함수인데 위에서 보면 sum함수에 전달하는 필요로 하는 데이터 부분에 또 다른 함수 c()가 사용됐습니다. 이 함수의 역할은 R의 자료구조중 벡터를 만드는 역할을 하는 함수로 필요로 하는 데이터를 각각 콤마(,)로 구별 했음을 볼 수 있습니다. 함수에서 필요로 하는 데이터는 전달인자라고 부르며 전달인자가 여러개일 경우 콤마(,)로 구별합니다.

다음 함수는 앞서 본 ls 함수로 R이 현재 갖고 있는 자료들의 list(줄여서 ls)들을 출력하는 함수입니다.

```
> ls()
 [1] "age"        "character"  "characters" …
```

어떤 함수들은 위와 같이 전달인자 없을 수도 있습니다. 이 경우에는 그냥 소괄호를 열고 닫으면 됩니다.

그리고 어떤 전달인자들은 입력하지 않으면 해당 함수들의 기본값으로 처리할 경우가 있습니다.

---

[1] "S programming language" Becker, Chambers & Wilks (1988) 에서 분류한 자료형으로 S는 R이 나오기 이전 상업용 Package로써 R에게 모티브를 제공한 S-PLUS에서 사용하는 언어입니다.

다음의 예를 보겠습니다.

```
> seq(1, 10)
 [1]  1  2  3  4  5  6  7  8  9 10
> seq(1, 10, by=2)
[1] 1 3 5 7 9
```

seq 함수는 순열(sequence)를 만드는 함수로 처음의 예제처럼 by 전달인자가 생략될 경우 1씩 증가/감소하는 순열을 만들지만 사용자에 의해 by 전달인자가 지정되면 주어진 값 만큼 증가/감소하면서 순열을 만들어 줍니다.

도움말 관련 함수 help()

R에는 활용할 수 있는 많은 함수와 각종 자원들이 있습니다. 또한 외부 Pacakage를 통해 계속해서 증가하는 다양한 자원들을 모두 알기는 힘듭니다. 이럴 경우 사용자들에게 기본적인 도움말을 help() 함수를 통해 제공해 줍니다. 다음은 help() 함수의 사용예입니다.

```
> help(c)
```
➔ 함수 c()의 도움말을 기본 브라우져를 통해 보여줍니다.

```
> ?c
```
➔ help() 함수의 단축키는 ?로 물음표(?) 뒤에 도움말을 얻을 함수 이름을 적습니다.

```
> help("<")
> help("for")
```
➔ R의 연산자나 문법에 사용되는 예약어 등은 큰 따옴표로 묶어 질의합니다.

```
> help(package="datasets")
```
➔ 패키지에 대한 도움말은 위와같이 package=”패키지명”으로  합니다.

```
> help.search("Normal Distribution")
```
➔ 찾고자 하는 대상을 명확히 모를 경우 검색어를 통한 검색을 실시할 수 있습니다.

```
> ??"Normal Distribution"
```
➔ 검색어를 통한 검색의 단축키는 ?? (물음표 두개)입니다.

```
> example(mean)

mean> x <- c(0:10, 50)

mean> xm <- mean(x)

mean> c(xm, mean(x, trim = 0.10))
[1] 8.75 5.50
```
➔ 함수의 사용예제를 제공합니다.

R의 자료구조

앞선 R 의 기본 자료형은 단일값(scalar)의 자료형을 말한다면 R 의 자료구조는 이런 단일값들로 구성된 자료 모음을 말하며, 실제 사용에 있어서는 단일값보다 자주 사용되므로 R 에서 자료형이라면 자료구조를 말합니다. R 이 지원하는 자료구조는 vector, matrix, array, list, data.frame 의 다섯가지입니다.

벡터(Vector) : 단일값(scalar) 들의 모임

vector는 동일한 자료형을 갖는 값들의 집합을 말하며 다음의 생성연산자와 생성함수를 통해 만들 수 있습니다.

**Vector 생성 연산자**

- "시작값**:**종료값" 시작값부터 종료값으로 숫자 1씩 더하거나 빼서 vector 생성
  - > 1:5 # 1, 2, 3, 4, 5의 다섯개로 구성된 vector 생성
  - > 5:1 # 5, 4, 3, 2, 1 의 다섯개로 구성된 vector 생성

**Vector 생성함수**

- vector(length=n) : vector 생성함수로 n개의 원소를 갖는 빈 벡터를 생성합니다.
- c() : vector 생성함수로 기존 벡터를 결합하여 새로운 벡터를 반환하며 다음과 같이 사용합니다.
  - > c(1, 2, 3) # 1, 2, 3의 세 개의 자료로 구성된 vector 생성
  - > c(4, 5, c(6, 7, 8)) # c() 함수 안에 사용된 c(6, 7, 8)의 원소를 갖는 vector로 4, 5, 6, 7, 8 의 다섯 개의 자료로 구성된 vector 생성
  - > x <- c(1, 2, 3) : x는 1, 2, 3의 세 개의 원소를 갖는 vector
- seq() : 순열(sequence)을 만드는 함수
  - 전달인자
    - from : 초기값
    - to : 종료값
    - by : 증가분
    - length.out : 생성할 vector 의 개수 지정
  - 사용 예
    - > seq(from=1, to=5, by=2) # 1 부터 5 까지 2 씩 증가하는 vector 생성
    - > seq(1, 5, by=2) # 위의 예와 동일
    - > seq(0, 1, by=0.001) # 0 부터 1 까지 0.001 씩 증가.<br># 생성된 자료는 1001 개
    - > seq(0, 1, length.out=1000) # 0 부터 1 까지 1000 개가 되는 자료 생성<br># 위의 seq()와 비교

- rep() : 반복 생성 함수
  - 전달인자
    - x : 반복할 자료(vector)
    - times : 자료 x 의 전체 반복 횟수
    - each : 자료 x 의 개별 원소들의 반복 횟수
  - 사용예
    - rep(c(1, 2, 3), times=2)  # vector 1, 2, 3 을 두 번 반복하는 vector
      # 결과 : 1, 2, 3, 1, 2, 3
    - rep(c(1, 2, 3), each=2)  # vector 1, 2, 3 의 개별 원소를 두 번 반복
      # 결과 : 1, 1, 2, 2, 3, 3

**벡터 원소 다루기**

```
> x <- c(5, 4, 3, 2, 1)
> length(x)
[1] 5
```
➔ 벡터 x 의 원소의 갯수를 알려고 할 때 length() 함수 사용

```
> x[1]
[1] 5
```
➔ 벡터 x 의 첫번째 원소를 가져올때

```
> x[1, 2, 3]
이하에 에러x[1, 2, 3] : 차원수가 올바르지는 않습니다
```
➔ 여러 위치의 원소를 가져오려 할 때 위와 같이 입력하면 오류 발생

```
> x[c(1, 2, 3)]
[1] 5 4 3
```
➔ 첫번째, 두번째, 세번째 원소를 가져오려면 이와 같이 갖고 오고자 하는 위치 정보를 담고 있는 벡터 전달

```
> x[-c(1, 2, 3)]
[1] 2 1
```
➔ 음수를 전달하면 해당 위치의 값을 제외하고 가져온다.

**벡터 계산**

```
> x <- c(1, 2, 3, 4)
> y <- c(5, 6, 7, 8)
> z <- c(1, 2)
> w <- c(1, 2, 3)
> 2 + x
[1] 3 4 5 6
```

➔ 개별값(scalar)와 벡터의 연산은 개별값이 벡터의 모든 원소와 연산 수행

```
> x + y
[1]  6  8 10 12
```

➔ 동일한 크기(원소의 갯수가 같은 경우)의 벡터 연산은 서로 같은 위치간 연산, x 의 첫 원소와 y 의 첫 원소가 계산되고 두번째, 세번째, 네번째 역시 같은 위치간에 연산

```
> x + z
[1] 2 4 4 6
```

➔ 한 벡터의 크기가 다른 벡터 크기의 정수배일 경우(예에서는 x 가 4개로 z의 2개보다 2배 큼) 작은 쪽 원소가 순환하며 연산

```
> x + w
[1] 2 4 6 5
Warning message:
In x + w : longer object length is not a multiple of shorter object
length
```

➔ 두 벡터의 크기가 다르고 어느 한 벡터가 다른 벡터의 원소의 갯수가 정수배가 아닐 경우 연산 실패

**벡터 관련 함수**

- is.vector(x) : 주어진 자료가 벡터이면 TRUE, 아니면 FALSE 반환
- as.vector(x) : 주어진 자료를 벡터로 변환한 자료구조를 반환

```
> x <- c(1, 2, 3, 4)
> is.vector(x)
[1] TRUE
```
➔ 주어진 자료 x는 벡터

```
> a.f <- factor(x)
```
➔ x를 factor(요인)으로 만들어 a.f에 저장

```
> is.vector(a.f)
[1] FALSE
```
➔ a.f는 벡터가 아니므로 FALSE 반환

```
> a.v <- as.vector(a.f)
```
➔ a.f를 벡터로 만들어 a.v에 저장

```
> a.f
[1] 1 2 3 4
Levels: 1 2 3 4

> a.v
[1] "1" "2" "3" "4"
```
➔ a.v는 문자열로 구성된 벡터

**유용한 함수**

all()과 any()

all()과 any()는 논리식을 받아 벡터의 모든 구성원이 전부 만족하는지 혹은 적어도 하나의 원소는 만족하는지 TRUE와 FALSE로 반환하는 함수입니다. 옵션 전달인자로 na.rm을 갖고 있으며 기본값은 FALSE 입니다. na.rm 을 TRUE로 할 경우 검사시 NA인 원소는 제외하고 검사합니다.

```
> x <- 1:5
> x > 3
[1] FALSE FALSE FALSE  TRUE  TRUE
```
➔ x가 3보다 큰 원소는 TRUE, 그렇지 않으면 FALSE

```
> all(x > 3)
[1] FALSE
```
➔ x가 모두 3보다 크지 않으므로 FALSE

```
> any(x > 3)
[1] TRUE
```
➔ x가 3보다 큰 원소가 있으므로 TRUE

이름 함수 : names()

```
names(x)
names(x) <- value
```

- x : R 객체

names() 함수는 R 객체의 이름을 불러오거나 지정하는 함수입니다. 이름을 지정할 때는 "names(x) <- 이름 벡터"의 형태로 이름을 지정합니다. 벡터외에 다른 자료구조에서도 많이 사용합니다.

```
> height <- c(80, 90, 70, 170)
> names(height) <- c("희동이", "둘리", "도우너", "또치")
➔ height 벡터의 순서대로 "희동이", "둘리", "도우너", "또치" 로 이름 지정

> height
희동이   둘리 도우너   또치
   80    90    70   170
> names(height)
[1] "희동이" "둘리"   "도우너" "또치"
➔ height 벡터의 이름 출력
```

자료의 앞과 뒤에서 일부를 추출하는 head(x, n)와 tail(x, n)

자료(x)로부터 n개(기본값은 6)의 자료를 앞과(head()) 뒤(tail())에서 가져옵니다.

```
> x <- 1:100
> head(x)
[1] 1 2 3 4 5 6
➔ x 의 처음 6개를 가져옵니다.

> head(x, n=7)
[1] 1 2 3 4 5 6 7
➔ x 의 처음 7개를 가져옵니다.

> tail(x)
[1]  95  96  97  98  99 100
➔  x의 마지막 6개를 가져옵니다.

> tail(x, n=7)
[1]  94  95  96  97  98  99 100
➔  x의 마지막 7개를 가져옵니다.
```

임의의 자료를 추출하는 sample()

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

- x : 선택할 모집단에 해당하는 양의 스칼라 혹은 벡터(스칼라의 경우 1:x의 벡터)
- size : 추출할 개수
- replace : 복원여부(기본값 FALSE)
- prob : x의 각 원소별 추출 확률(기본값은 NULL)

```
> sample(10)
 [1]  4  1  9 10  3  6  8  5  2  7
```
➔ 10 개를 임의로 섞습니다.

```
> sample(45, 6)
[1] 33 11 30  1 27 26
```
➔ 45개중 6개를 임의로 추출합니다.(로또 시뮬레이터?)

```
> sample(10, 3, replace=TRUE)
[1] 3 3 3
```
➔ 동일한 확률 (1/10)로 10 개중 3개를 임의로 복원 추출합니다.

```
> sample(10, 3, prob=(1:10)/55)
[1]  9 10  4
```
➔ 10 개중 3개를 임의로 복원 추출하는데 1:10이 추출될 확률은 (1:10)/55로 각각 다릅니다.

```
> x <- seq(0, 1, by=0.1)
> x
 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
> x[sample(length(x), 3)]
[1] 0.6 0.5 0.8
```
➔ 대괄호 안에 쓰여 자료 구조중 랜덤으로 자료를 선택합니다.

```
> sample(x, 3)
[1] 0.0 0.5 1.0
```
➔ x를 벡터로 넣을 경우의 예입니다.

선택함수 which()

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

- x : 조건 판단을 위한 벡터 혹은 배열을 반환하는 수식
- arr.ind : x가 배열일 경우 배열의 인덱스(첨자)를 출력할 것인지 지정. 기본값 FALSE

선택함수 which()는 탐색할 벡터를 포함하는 수식을 전달인자로 받아 해당 조건을 만족하는 원소의 인덱스를 반환합니다.

```
> x <- c(2, 4, -1, 3)
> which(x > 2)
[1] 2 4
```

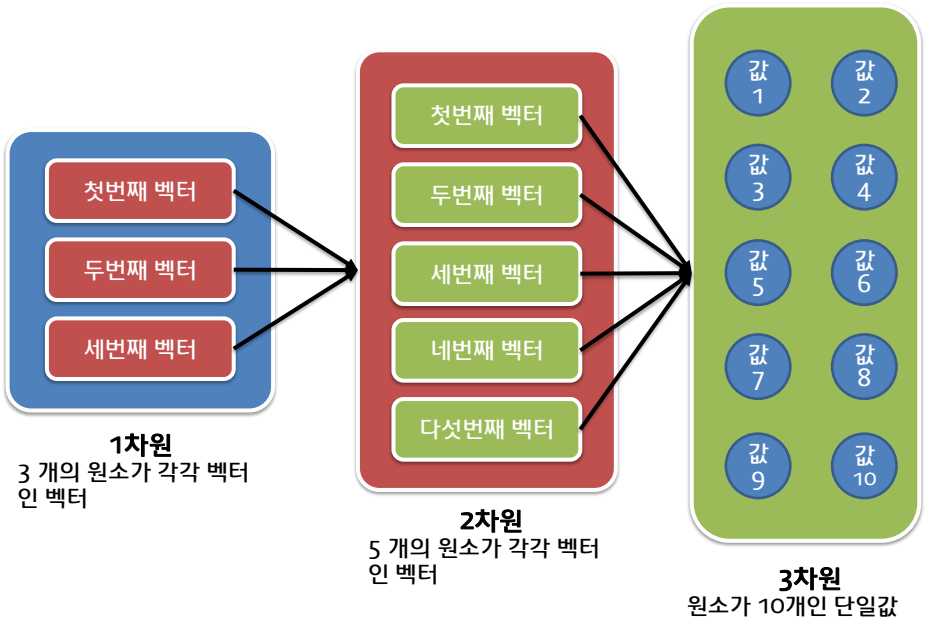
➔ x의 원소중 2보다 큰 원소의 인덱스 출력

```
> names(x) <- c("1st", "2nd", "3rd", "4th")
> which(x > 2)
2nd 4th
  2   4
```

➔ x의 원소중 2보다 큰 원소의 인덱스와 해당 이름 출력

배열(Array ) : 벡터의 원소들이 벡터로 구성된 형태

배열은 수학에서 이야기하는 차원을 갖는 자료 구조로 각 자료들이 벡터로 구성됩니다. 만일 3차원 자료일 경우 1차원이 세 개의 벡터, 2차원이 5개의 벡터, 3차원이 10개의 단일 값(scalar)로 구성될 경우 전체 원소들의 개수는 3 x 5 x 10 = 150 개의 원소이고 1차원 원소들은 각각 5개 씩의 원소를 갖는 세 개의 벡터의 집합이 되고 2차원의 원소는 각각 10개의 단일값을 갖는 벡터로 구성되어 있습니다.




**배열 생성하기 : array() 함수**

```
array(data = NA, dim = length(data), dimnames = NULL)
```

- data : vector 자료
- dim : 각 차원을 정의하는 vector
  - Ex) c(2, 5, 10) : 전체는 3차원이고 1차원의 원소는 2, 2차원은 5, 3차원은 10개로 전체 원소의 개수는 2 X 5 X 10 = 100 개
- [dimnames] : (옵션) 각 차원의 이름을 갖는 vector

- 사용 예

```
> arr <- array(1:3, c(2, 4))
> arr
     [,1] [,2] [,3] [,4]
[1,]    1    3    2    1
[2,]    2    1    3    2
```

➔ 2행 4열의 배열을 만들고 원소들은 1, 2, 3 으로 채운다(부족할 경우 반복)

➔ 생성된 배열은 변수 arr이 가리킨다

```
> arr[1,]
[1] 1 3 2 1
```

➔ 배열 arr의 1행의 원소 반환

```
> arr[,3]
[1] 2 3
```

➔ 배열 arr의 3열의 원소 반환

```
> dimnamearr = list(c("1st", "2nd"), c("1st", "2nd", "3rd", "4th"))
```

➔ 배열의 행과 열에 이름 지정, list(c(), c(), c(), …)의 형태로 위의 예에서 첫번째 전달된 벡터가 행의 이름 두번째 전달된 벡터가 열의 이름

```
> arr2 <- array(1:3, c(2, 4), dimnames=dimnamearr)
    1st 2nd 3rd 4th
1st   1   3   2   1
2nd   2   1   3   2

> arr2["1st", ]
1st 2nd 3rd 4th
  1   3   2   1

> arr2[ ,"3rd"]
1st 2nd
  2   3
```

행렬(Matrix) : 배열의 특수한 경우(2차원 배열)

**행렬 생성하기**

matrix() 함수

```
matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE,
       dimnames = NULL)
```

- data : 행렬로 재구성할 Vector 혹은 외부 데이터
- nrow : 행 요소의 개수
- ncol : 열 요소의 개수
  - nrow 혹은 ncol 중에 하나만 결정한다.
- byrow = FALSE : data를 행 단위로 배치할지 여부
- [dimnames = NULL] : 행과 열의 이름 list

- 사용 예

```
> tmp <- 1:12
> tmp
[1]  1  2  3  4  5  6  7  8  9 10 11 12
```

➔ 행렬에 배치할 벡터

```
> matrix(tmp, nrow=3)
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

➔ 위에서 만든 벡터를 행의 개수가 3인 행렬에 재배치

➔ 1열부터 자료를 채워감

```
> matrix(tmp, nrow=3, byrow=TRUE)
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
```

➔ 위에서 만든 벡터를 행의 개수가 3인 행렬에 재배치

➔ 1행부터 자료를 채워감(byrow=TRUE)

cbind(), rbind() 사용

cbind()는 벡터를 열 단위로 합치고 rbind()는 행 단위로 합친다.

**사용 예**

```
> v1 <- c(1, 2, 3, 4)
> v2 <- c(5, 6, 7, 8)
> v3 <- c(9, 10, 11, 12)
> cbind(v1, v2, v3)
     v1 v2 v3
[1,]  1  5  9
[2,]  2  6 10
[3,]  3  7 11
[4,]  4  8 12
```

➔ v1, v2, v3 를 열로 묶어 4행 3열짜리 행렬 생성

```
> rbind(v1, v2, v3)
   [,1] [,2] [,3] [,4]
v1    1    2    3    4
v2    5    6    7    8
v3    9   10   11   12
```

➔ v1, v2, v3 를 행으로 묶어 3행 4열짜리 행렬 생성

**같이 사용할 수 있는 함수**

apply() : 배열 등의 특정 차원별로 함수 적용

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

- X : apply를 적용할 배열 (우리의 경우 대부분 행렬(matrix))
- MARGIN : 함수를 적용할 차원, 행렬의 경우 1이면 행별로, 2이면 열별로 FUN 적용
- FUN : 적용할 함수
- ... : FUN의 추가적인 전달인자

**사용예**

```
> v1 <- c(1, 2, 3, 4)
> v2 <- c(5, 6, 7, 8)
> v3 <- c(9, 10, 11, 12)
> m1 <- cbind(v1, v2, v3)
```

➔ cbind를 통해 4행 3열짜리 행렬 생성

```
> apply(m1, 1, mean)
[1] 5 6 7 8
```

➔ 각 행별로 평균 계산

```
> apply(m1, 2, mean)
  v1   v2   v3
 2.5  6.5 10.5
```

➔ 각 열별로 평균 계산

요인(Factor) : 범주형 자료 저장

구별되는 기호로 값을 표시하는 요인은 저장값이 갖는 의미보다 구별하기 위한 값으로 사용됩니다. 예를 들어 숫자 1, 2, 3은 산술연산을 통해 계산되어지는 본래의 숫자로써의 기능을 하지만 요인으로 지정된 1, 2, 3은 계산할 수 없이 단지 세 개의 그룹 혹은 상태를 구별짓는 의미로 사용됩니다. R에서는 이런 요인을 별도의 함수를 통해 생성합니다.

**생성 함수 : factor()**

```
factor(x = character(), levels, labels = levels,
       exclude = NA, ordered = is.ordered(x), nmax = NA)
```

- x : 요인으로 만들 벡터
- levels : 주어진 데이터 중 요인으로 할 데이터 지정(벡터), 여기서 빠진 값은 NA로 처리
- labels : 실제 값 외에 사용할 요인 이름(벡터), 예를 들어 데이터에 1이 남자를 가리킬 경우 labels 를 통해 "남자" 혹은 "M" 등으로 변경
- exclude : 요인으로 사용하지 않을 값 지정(벡터)
- ordered : 순서 여부 지정(TRUE/FALSE), 순서있는 범주의 경우 사용하며 순서는 levels에 의해 명시적으로 지정하는 것을 추천
- nmax : 최대 level의 수 (즉, 최대 요인수)

사용예

```
> x <- c(1, 2, 3, 4, 5)
> factor(x, levels=c(1, 2, 3, 4))
[1] 1    2    3    4    <NA>
Levels: 1 2 3 4
```
➔ levels를 통해 자료중 1, 2, 3, 4 네 개의 값만 요인으로 사용

```
> factor(x, levels=c(1, 2, 3, 4), exclude=c(1, 2))
[1] <NA> <NA> 3    4    <NA>
Levels: 3 4
```
➔ exclude를 사용해서 1, 2 를 사용할 요인에서 제거

```
> factor(x, levels=c(1, 2, 3, 4), exclude=c(1, 2), ordered=TRUE)
[1] <NA> <NA> 3    4    <NA>
Levels: 3 < 4
```
➔ ordered 에 TRUE를 주어 leveles에 나열된 1, 2, 3, 4를 순서대로 지정하지만 exclude를 통해 1, 2 가 제거되어 3, 4가 순서를 갖게 함

❖ 요인을 만드는 것은 factor() 함수 외에 순서있는 범주를 만드는 ordered() 함수도 있지만 factor() 함수를 통해 충분히 만들 수 있으므로 생략합니다.

**같이 사용할 수 있는 함수**

is.factor(x) : 주어진 자료가 요인이면 TRUE를 그렇지 않으면 FALSE 반환

as.factor(x) : 주어진 자료를 요인으로 변환하는 함수

```
> v <- c(1, 2, 3, 4)
> x <- factor( v )
```
➔ 벡터 1, 2, 3, 4 로 요인 x 생성

```
> is.factor(x)
[1] TRUE
```
➔ x 는 요인이므로 TRUE 반환

```
> is.factor(v)
[1] FALSE
```
➔ v 는 요인이 아니므로 FALSE 반환

```
> v
[1] 1 2 3 4
> as.factor(v)
[1] 1 2 3 4
Levels: 1 2 3 4
```
➔ 주어진 벡터 v 를 요인으로 변환

tapply()

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

- X : 집계할 자료 일반적으로 벡터
- INDEX : X를 집계할 요인이나 요인의 리스트로 요인이 아닐 경우 as.factor()가 호출되어 요인으로 만든다. 또한 사용되는 자료의 길이가 X와 길이가 같아야 한다.
- FUN : 집계에 사용할 함수
- ... : FUN에 사용되는 함수의 추가적인 전달인자.
- simplify : TRUE이면 결과를 스칼라로 FALSE이면 리스트형의 배열로 반환, 기본값 TRUE

사용예

```
> score <- c(92, 90, 82, 88, 78, 64, 82, 90)
➔ 학생들의 성적 벡터
> subject <- c("k", "k", "m", "m", "m", "m", "k", "k")
➔ 각 학생들의 과목 벡터
> tapply(score, subject, mean)
   k    m
88.5 78.0
➔ 과목별로 학생들 성적의 평균을 구한다.
```

데이터 프레임(Data frame) : 테이블 구조의 자료 생성

자료 처리를 위해 가장 많이 사용된 자료구조로 행렬과 유사한 구조를 가지지만 서로 다른 기본 자료형을 갖는 자료들의 모임입니다. 거의 모든 통계자료는 데이터 프레임의 형태를 가지므로 많이 사용되는 자료구조입니다. 테이블에서 각 열에 해당하는 위치에 개별 자료들이 위치하는데 이를 속성이라 부르겠습니다. 각 속성은 벡터 혹은 요인(factor) 형태의 자료이며 모든 속성들의 크기는 동일해야 합니다. 또한 각 속성별로 저장된 자료의 위치가 동일해야 합니다. 설명을 위해 지금은 자료를 직접 입력해서 만드는 데이터 프레임에 대해 알아보겠습니다만 데이터 프레임은 직접 입력하는 경우보다 외부 데이터로부터 가져오는 경우가 많습니다. 외부에서 데이터를 R로 가져오는 방법은 뒤에 설명드리겠습니다.

**생성 함수 : data.frame()**

사용예

```
> name <- c("철수", "영희", "길동")
> age <- c(21, 20, 31)
> gender <- factor(c("M", "F", "M"))
> character <- data.frame(name, age, gender)
```

➔ data frame을 구성할 속성은 name, age, gender

```
> character
  name age gender
1 철수  21      M
2 영희  20      F
3 길동  31      M

> character$name
[1] 철수 영희 길동
Levels: 길동 영희 철수
```

➔ name 속성의 값 가져오기

```
> character[1, ]
  name age gender
1 철수  21      M
```

➔ 첫번째 행에 해당하는 값 가져오기 (세 속성 모두 포함)

```
> character[ , 2]
[1] 21 20 31
```

➔ 두번째 열에 해당하는 값 가져오기 (모든 행의 두번째 열의 값을 가져옴. 벡터)

```
> character[3, 1]
[1] 길동
Levels: 길동 영희 철수
```

➔ 세번째 행의 첫번째 열의 값 가져오기 (벡터)

**유용한 함수**

attach()와 detach()

데이터 프레임의 각 열을 임시로 변수명처럼 사용하게 하고(attach()) 이를 다시 해제합니다(detach()). 만일 열 이름이 기존 자료명과 같을 경우 열 이름으로 쓰지 못합니다.

```
> head(airquality)
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
5    NA      NA 14.3   56     5   5
6    28      NA 14.9   66     5   6
> Ozone
```

에러:개체 'Ozone'이 없습니다

➔ Ozone은 현재 없습니다.

```
> attach(airquality)
> Ozone[1:5]
[1] 41 36 12 18 NA
```

➔ attach() 후에는 데이터 프레임의 열이름으로 접근할 수 있습니다.

```
> detach(airquality)
```

➔ 임시로 풀어놓은 열이름을 해제 합니다.

```
> Ozone
```

에러:개체 'Ozone'이 없습니다

➔ 해제후에는 열이름으로 접근할 수 없습니다.

```
> Ozone <- c(1, 2, 3)
```

➔ Ozone 이라는 자료를 생성합니다.

```
> attach(airquality)
The following object(s) are masked _by_ '.GlobalEnv':

    Ozone
```

➔ Ozone을 쓸 수 없음을 나타냅니다.

```
The following object(s) are masked from 'airquality (position 3)':

    Day, Month, Ozone, Solar.R, Temp, Wind
> Ozone
[1] 1 2 3
```

➔ 기존 존재하는 자료로 나올뿐 데이터 프레임의 자료가 아닙니다.

```
> Month[1:5]
[1] 5 5 5 5 5
```

➔ 나머지는 열 이름으로 접근 가능합니다.

with() : 데이터 프레임에 함수를 수행합니다.

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

- data : 데이터 프레임 혹은 리스트
- expr : 수행할 함수 혹은 수식 등의 expression

```
> head(cars)
  speed dist
1     4    2
2     4   10
3     7    4
4     7   22
5     8   16
6     9   10
> mean(cars$speed)
[1] 15.4
> mean(cars$dist)
[1] 43
> with(cars, mean(speed))
[1] 15.4
> with(cars, mean(dist))
[1] 43
```

➔ attach()를 사용하지 않고도 열 이름을 바로 사용할 수 있습니다. attach()는 지속적으로 사용할 경우 유용합니다. 그렇지 않은 경우 with()를 추천합니다.

subset() : 조건에 맞는 부분집합을 추출합니다.

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

- x : 부분집합을 추출할 R 객체
- subset : 부분집합을 생성할 조건(결측치는 FALSE)
- select : 반환할 열 조건식
- drop : "["로 전달될 슬라이싱 조건, 기본값 FALSE

데이터 프레임에 대괄호("[']연산을 통해 원하는 자료를 추출할 수 있는 것과 마찬가지로 데이터 프레임(혹은 벡터와 행렬에서도 작동)에서 어떤 조건을 만족하는 데이터 프레임의 부분집합을 추출하는 함수입니다.

```
> airquality
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
…
> subset(airquality, Temp > 80)
   Ozone Solar.R Wind Temp Month Day
29    45     252 14.9   81     5  29
35    NA     186  9.2   84     6   4
36    NA     220  8.6   85     6   5
…
```

➔ airquality에서 Temp가 80보다 큰 자료를 가져옵니다.

```
> subset(airquality, Temp > 80, select = c(Ozone, Temp))
   Ozone Temp
29    45   81
35    NA   84
36    NA   85
…
```

➔ airquality에서 Temp가 80보다 큰 자료중 Ozone 열과 Temp열을 가져옵니다.

```
> subset(airquality, Temp > 80, select = -c(Ozone, Temp))
   Solar.R Wind Month Day
29     252 14.9     5  29
35     186  9.2     6   4
36     220  8.6     6   5
…
```

➔ airquality에서 Temp가 80보다 큰 자료중 Ozone 열과 Temp열을 제외하고 가져옵니다.

na.omit() : NA인 데이터를 제외하고 가져옵니다.

데이터 프레임에서 어떤 한 열의 값이 NA일 경우 해당하는 행의 자료를 제외하고 가져오는 함수입니다. 다음 예를 통해 살펴보겠습니다.

```
> str( airquality )
'data.frame':   153 obs. of  6 variables:
 $ Ozone  : int  41 36 12 18 NA 28 23 19 8 NA ...
 $ Solar.R: int  190 118 149 313 NA NA 299 99 19 194 ...
 $ Wind   : num  7.4 8 12.6 11.5 14.3 14.9 8.6 13.8 20.1 8.6 ...
 $ Temp   : int  67 72 74 62 56 66 65 59 61 69 ...
 $ Month  : int  5 5 5 5 5 5 5 5 5 5 ...
 $ Day    : int  1 2 3 4 5 6 7 8 9 10 ...
```

➔ airqulity 자료는 153개의 관찰자료(행)를 갖는 자료구조입니다.

```
> str ( na.omit(airquality) )
'data.frame':   111 obs. of  6 variables:
 $ Ozone  : int  41 36 12 18 23 19 8 16 11 14 ...
 $ Solar.R: int  190 118 149 313 299 99 19 256 290 274 ...
 $ Wind   : num  7.4 8 12.6 11.5 8.6 13.8 20.1 9.7 9.2 10.9 ...
 $ Temp   : int  67 72 74 62 65 59 61 69 66 68 ...
 $ Month  : int  5 5 5 5 5 5 5 5 5 5 ...
 $ Day    : int  1 2 3 4 7 8 9 12 13 14 ...
 - attr(*, "na.action")=Class 'omit'  Named int [1:42] 5 6 10 11 25 26
27 32 33 34 ...
  .. ..- attr(*, "names")= chr [1:42] "5" "6" "10" "11" ...
```

➔ na.omit을 하고 나면 열에 NA를 포함하는 행을 제외하고 가져옵니다. na.action 속성에 제외된 행 번호를 기록합니다.

merge() : 두 개의 데이터 프레임을 합칩니다.

```
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"),
       …)
```

- x, y : 합칠 대상이 되는 데이터 프레임
- by : 합칠때 기준이 되는 열로 기본값은 두 데이터프레임 중 동일한 이름을 갖는 열
- by.x : 합칠 기준이 되는 x 의 열 이름
- by.y : 합칠 기준이 되는 y 의 열 이름
- all.x : TRUE 로 지정시 모든 x 의 행이 합쳐지고 이에 해당하는 y 가 없을 경우 y 열에 해당하는 값은 NA

- all.y : TRUE 로 지정시 모든 y 의 행이 합쳐지고 이에 해당하는 x 가 없을 경우 x 열에 해당하는 값은 NA
- all : all.x 와 all.y 가 동일한 TRUE 나 FALSE 를 갖게 함
- sort : TRUE 이면 합쳐질 열 이름 순으로 정렬
- suffixes : X 와 Y 의 이름이 서로 동일할 경우 두개의 문자로 구성된 suffix 추가

```
> authors <- data.frame(
+         surname  =  I(c("Tukey",  "Venables",  "Tierney",  "Ripley",
"McNeil")),
+     nationality = c("US", "Australia", "US", "UK", "Australia"),
+     deceased = c("yes", rep("no", 4)))
> books <- data.frame(
+     name = I(c("Tukey", "Venables", "Tierney",
+             "Ripley", "Ripley", "McNeil", "R Core")),
+     title = c("Exploratory Data Analysis",
+              "Modern Applied Statistics ...",
+              "LISP-STAT",
+              "Spatial Statistics", "Stochastic Simulation",
+              "Interactive Data Analysis",
+              "An Introduction to R"),
+     other.author = c(NA, "Ripley", NA, NA, NA, NA,
+                      "Venables & Smith"))
>
> (m1 <- merge(authors, books, by.x = "surname", by.y = "name"))
   surname nationality deceased                      title other.author
1   McNeil   Australia      no    Interactive Data Analysis        <NA>
2   Ripley        UK      no          Spatial Statistics        <NA>
3   Ripley        UK      no       Stochastic Simulation        <NA>
4  Tierney        US      no                  LISP-STAT        <NA>
5    Tukey        US     yes    Exploratory Data Analysis        <NA>
6  Venables    Australia          no  Modern  Applied  Statistics  ...
Ripley
```

➔ authors와 books를 authors의 surname 값과 books의 name 값이 같은 자료를 한 행으로 하여 합친다.

```
> (m2 <- merge(books, authors, by.x = "name", by.y = "surname"))
     name                      title other.author nationality deceased
1   McNeil    Interactive Data Analysis        <NA>   Australia      no
2   Ripley          Spatial Statistics        <NA>         UK      no
3   Ripley       Stochastic Simulation        <NA>         UK      no
4  Tierney                  LISP-STAT        <NA>         US      no
5    Tukey    Exploratory Data Analysis        <NA>         US     yes
6 Venables  Modern  Applied  Statistics  ...         Ripley    Australia
no
```

➔ authors와 books를 authors의 name 값과 books의 surname 값이 같은 자료를 한 행으로 하여 합친다.

리스트(List) : 서로 다른 기본 자료형을 가질 수 있는 자료구조들의 모임

앞선 배열과 행렬등은 포함되는 원소들의 기본 자료형이 모두 동일한 경우와 크기가 일정한 경우에만 사용할 수 있는 반면 리스트는 서로 다른 기본 자료형을 갖는 자료구조들(벡터, 배열, 행렬, 리스트, 데이터 프레임 등)을 하나의 이름으로 모아 놓은 형태입니다. Data frame 과 다른 점은 Data frame은 모든 속성(열 요소)들의 크기가 일정해야 하는 반면 list는 자유롭습니다. 리스트는 list() 함수를 써서 생성할 수 있는데 형태는 다음과 같습니다.

- myList <- list(자료구조 1, 자료구조 2, …)
- myList <- list(name1=자료구조 1, name2=자료구조 2, …)

리스트는 순서를 갖고 있으며 위의 예에서 첫번째 리스트의 경우 각 순서를 list() 함수 안에 집어넣은 순서대로 갖습니다. 두번째 예의 경우에도 순서는 list() 함수 안에 배치한 순서로 결정되지만 이름을 주어 나타낼 수 있습니다. 사용 예를 통해 자세히 살펴보겠습니다.

**생성 함수 : list()**

사용예

```
> title <- "My List"
> ages <- c(31, 41, 21)
> numbers <- matrix(1:9, nrow=3)
> names <- c("Baby", "Gentle", "none")
> listEx <- list(title, ages, numbers, names)
```

➔ 순서대로 title, ages, numbers, names 의 순서대로 list를 생성합니다.

```
> listEx
[[1]]
[1] "My List"

[[2]]
[1] 31 41 21

[[3]]
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9

[[4]]
[1] "Baby"   "Gentle" "none"
```

➔ 리스트에 포함된 하위 요소들이 순서대로 저장됩니다. 순서는 대괄호 두개([[]) 사이에 숫자로 지정합니다.

```
> listEx[[1]]
[1] "My List"
```

➔ 해당 리스트의 첫번째 요소를 가져옵니다.

```
> listEx2 <- list(title=title, age=ages, number=numbers, name=names)
```
➔ 리스트를 구성하는 자료구조에 이름을 주어 배치합니다. 순서는 앞선 예제와 똑같지만 실제 사용에 있어 이름을 지정하면 순서보다 이름을 통해 값을 가져오는 것이 일반적입니다. 이 경우 data frame 과 비슷하게 처리할 수 있습니다.

```
> listEx2
$title
[1] "My List"

$age
[1] 31 41 21

$number
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9

$name
[1] "Baby"   "Gentle" "none"

> listEx2[[1]]
[1] "My List"
```
➔ 첫번째 요소의 값을 가져옵니다.

```
> listEx2$title
[1] "My List"
```
➔ 이름을 지정한 경우 리스트자료명 뒤에 달러표시 ($)를 붙히고 지정한 이름을 적어 해당 자료를 가져올 수 있습니다.

```
> listEx2$age
[1] 31 41 21

> listEx2$number
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9

> listEx2$name
[1] "Baby"   "Gentle" "none"
```

**같이 사용할 수 있는 함수**

is.list(x) : 주어진 자료가 리스트이면 TRUE를 그렇지 않으면 FALSE 반환

as.list(x) : 주어진 자료를 리스트로 변환하는 함수

```
> x <- list(c(1,2,3,4), c(3, 2, 1))
> v <- c(1, 2, 3, 4)
> is.list(x)
[1] TRUE
```

➔ x 가 리스트이므로 TRUE 반환

```
> is.list(v)
[1] FALSE
```

➔ x 가 리스트가 아니므로 FALSE 반환

```
> a.l <- as.list(v)
> a.l
[[1]]
[1] 1

[[2]]
[1] 2

[[3]]
[1] 3

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

➔ 벡터 v 를 리스트로 변환시 각 원소가 리스트의 요소로 변환

lapply(), sapply() : 리스트에 함수 적용하기

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

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

- X : 적용할 리스트
- FUN : 적용할 함수
- ... : FUN에 사용되는 함수의 추가적인 전달인자.
- simplify : TRUE이면 결과를 스칼라로 FALSE이면 리스트형의 배열로 반환, 기본값 TRUE
- USE.NAMES : TRUE이면 X가 문자열로 구성되어 있을 때 결과의 이름이 정해지지 않으면 X를 이름으로 결과 반환

사용예

```
> x <- list(a = 1:10, beta = exp(-3:3),
+ logic = c(TRUE,FALSE,FALSE,TRUE))
> lapply(x, mean)
$a
[1] 5.5

$beta
[1] 4.535125

$logic
[1] 0.5
```

➔ 리스트의 각 요소에 평균을 구하고 결과를 리스트로 반환

```
> sapply(x, mean)
      a    beta   logic
5.500000 4.535125 0.500000
```

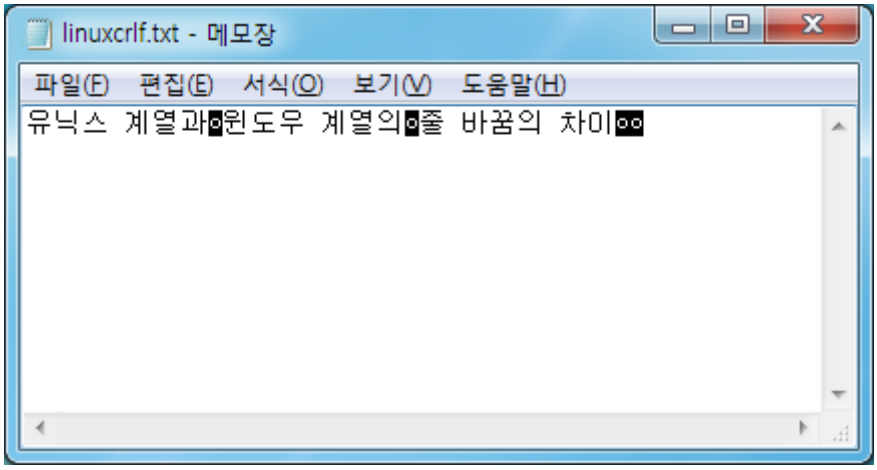
➔ 리스트의 각 요소에 평균을 구하고 결과를 벡터로 반환

외부 파일로부터 자료 불러오기

행과 열의 구분

일반 텍스트 파일로부터 테이블 구조(행과 열)를 저장하는 것은 열 구분자와 행 구분자를 사용하여 R이나 다른 Application이 이를 읽어서 테이블 구조로 인식하도록 하면 됩니다. 대개의 경우 행 구분자로는 줄 바꿈(키보드 상의 Enter 키)을 사용하는데 유닉스 계열의 OS와 윈도우(Microsoft)에서 사용하는 방식이 조금 다릅니다.

유닉스 계열의 줄바꿈은 ASCII상의 New Line에 해당하는 "₩n"을 사용하는데 반해 윈도우 계열에서는 기존 타자기에서처럼 Carriage Return(₩r)과 New Line(₩n)을 같이 사용하여 "₩r₩n"으로 줄바꿈합니다.




이런 이유에서 메모장 등에서는 유닉스 계열에서 만든 텍스트 파일을 전부 한 줄에 표시하거나 하는데 R에서 읽어올 경우 이에 구애받지 않고 잘 읽어옵니다.

열 구분자

앞서 대개의 경우 행 구분은 줄 바꿈으로 한다는 것을 이야기했으니 열 구분자에 대해 알아보자. Fortran 등의 언어에서는 열 구분자 보다 열 위치를 가지고 열 구분을 하였는데 현재 이 방법은 극히 일부에서만 사용하고 열을 나타내는 구분자(Seperator)를 통해 열을 구별합니다.

열 구분을 위해 White Space(공백문자)를 사용하는 경우 적당한 White Space로는 Space(공백문자), Tab(수평 탭) 등이 많이 사용되며, 이 경우 동일한 공백문자를 연속으로 입력시 매 공백문자마다 열로 구분할 것인지 아닌지를 결정하여야 합니다. 파일의 확장자로는 .txt 를 비롯하여 .dat, .prn 등 크게 구애받지 않습니다. 다음은 Space로 구분된 텍스트 파일을 Excel에서 읽어오는 과정으로 우리가 고려해 볼 부분이 많이 있으니 참고하겠습니다.

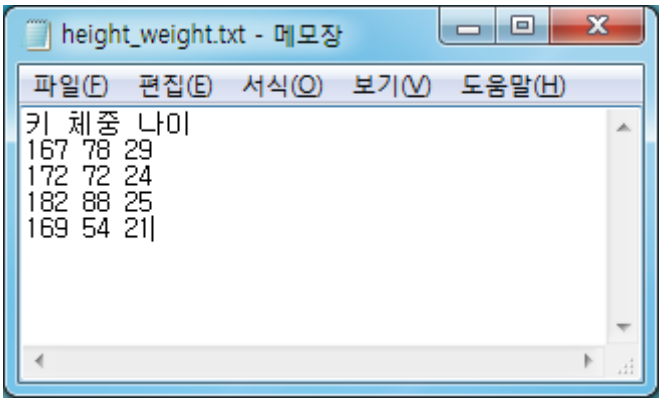



**Figure 1 불러올 원본 파일**

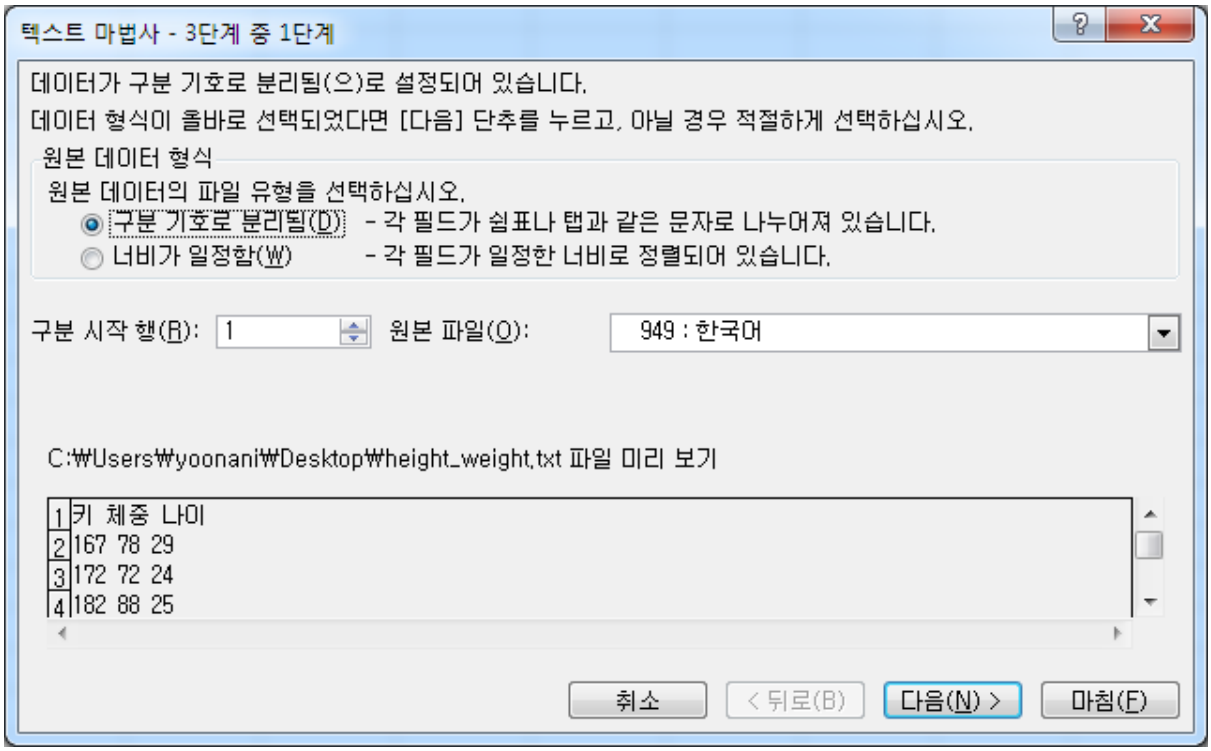



**Figure 2 텍스트 마법사 1단계**

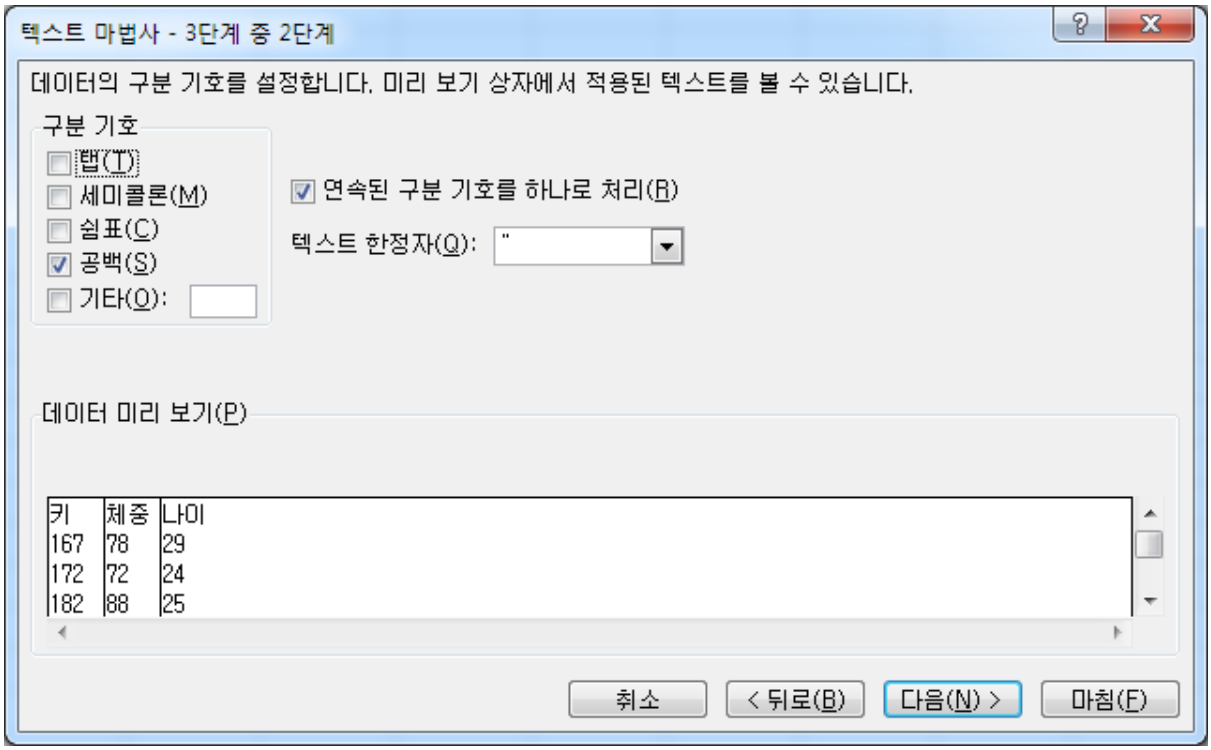



**Figure 3 텍스트 마법사 2단계**

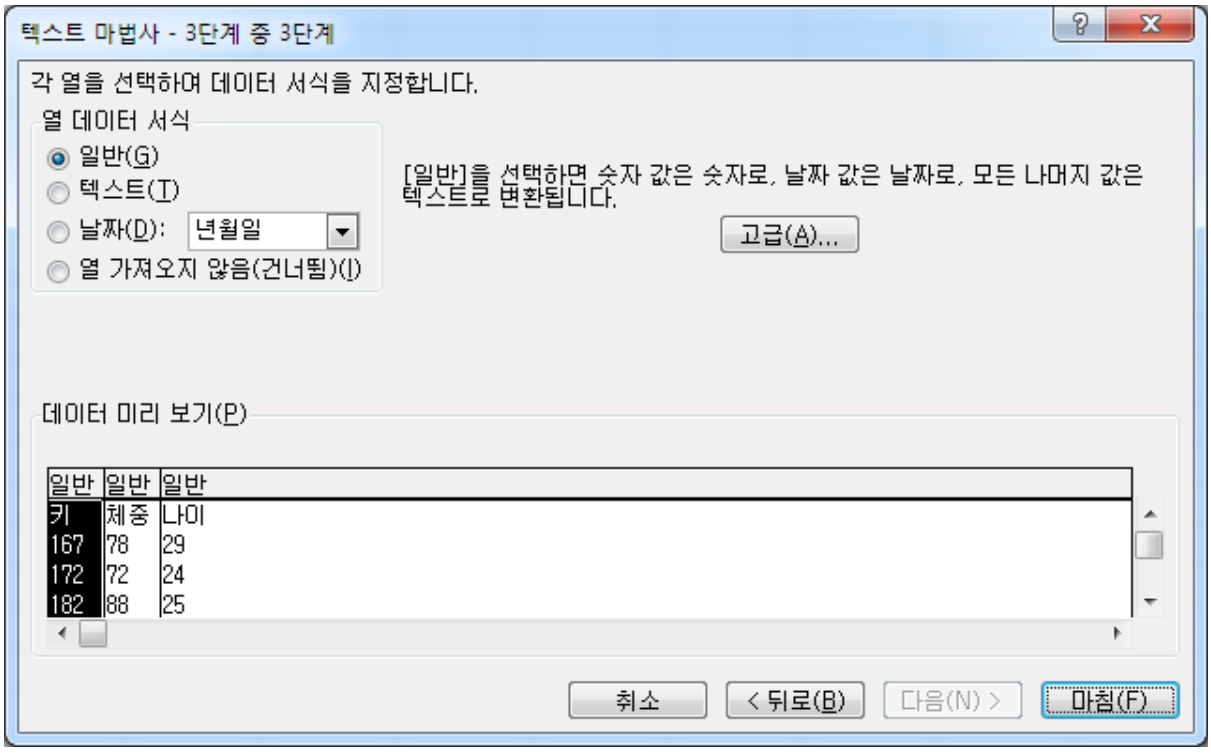



**Figure 4 텍스트 마법사 3단계**

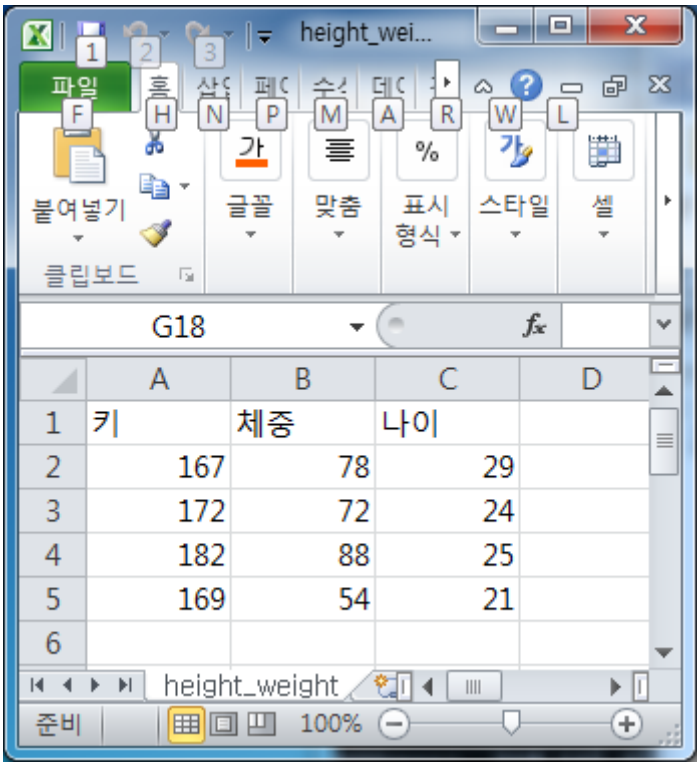



**Figure 5 불러오기 완료**

R에서는 read.table() 함수가 위의 Excel의 텍스트 마법사의 역할을 담당하고 있습니다.

```
read.table(file, header = FALSE, sep = "", quote = "₩"'",
           dec = ".", row.names, col.names,
           as.is = !stringsAsFactors,
           na.strings = "NA", colClasses = NA, nrows = -1,
           skip = 0, check.names = TRUE, fill = !blank.lines.skip,
           strip.white = FALSE, blank.lines.skip = TRUE,
           comment.char = "#",
           allowEscapes = FALSE, flush = FALSE,
           stringsAsFactors = default.stringsAsFactors(),
           fileEncoding = "", encoding = "unknown", text)
```

- file : 불러올 파일명
- header : 첫 줄을 데이터로 읽을 것인지 여부, TRUE이면 첫줄을 데이터가 아닌 열의 이름으로 인식(기본값은 FALSE)
- sep : 열구분자로 기본값은 공백문자이며 연속된 구분자는 하나로 처리
- quote : 값을 둘러싸는 인용부호로 문자열 값에 사용(기본값은 큰따옴표 ")
- dec : 소수점의 표기 방식(기본값 : .)으로 유럽 일부지역에서는 콤마(,)를 사용

```
> students <- read.table("height_weight.txt", header=T)
경고메시지:
In read.table("height_weight.txt", header = T) :
  incomplete final line found by readTableHeader on
'height_weight.txt'
```
➔ 이 오류는 파일이 제대로 끝나지 않았음을 의미합니다. 마지막 데이터 입력후 줄바꿈을 해줘야 합니다.

```
> students <- read.table("height_weight.txt", header=T)
```
➔ 현재의 작업 경로에서 "height_weight.txt"를 읽어오는데 첫 줄은 각 열의 이름으로 인식합니다.

```
> students
   키 체중 나이
1 167   78   29
2 172   72   24
3 182   88   25
4 169   54   21
```

CSV 파일 - 열 구분자로 콤마(,) 사용

특별히 열 구분을 콤마(,)로 하는 경우 csv(Comma Separated Value)라고 하여 사용하는 경우가 있습니다. 개인적으로 제가 선호하는 방법으로 공백문자를 열 구분자로 할 경우 반복되는 공백문자의 난해함을 해소하고 열 구분이 사람 눈에 잘 드러나도록 구분자를 사용하고 있습니다. 확장자는 앞선 공백문자를 사용할 때와 마찬가지로 어떤 것을 사용해도 무방합니다만 관습적으로 .csv를 많이 사용합니다.

R에서는 read.csv()가 이 역할을 수행하고 있으며 read.csv()는 read.table()과 동일하나 sep의 기본값이 콤마(,) 입니다.

```
> students2 <- read.csv("height_weight.csv", header=T)
```
➔ 현재의 작업 경로에서 "height_weight.csv"를 읽어오는데 첫 줄은 각 열의 이름으로 인식합니다.

```
> students2
   키 체중 나이
1 167   78   29
2 172   72   24
3 182   88   25
4 169   54   21
```

# 조금 더 R을 알아봅시다

조건 탐색 : 자료구조의 특정 원소 가져오기

앞서 자료구조를 설명드리면서 간단히 특정 원소를 가져오는 방법에 대해 알아보았습니다. 이번에는 조건에 맞는 값을 자료구조에서 가져오는 방법에 대해 알아보겠습니다. 특정원소와 같이 조건을 주어 특정 원소만 가져올 때도 조건들을 자료구조명 뒤에 대괄호 안에 넣어 사용합니다.

```
> ex <- c(1, 3, 7, NA, 12)
> ex[ex < 10]
➔ ex 에서 값이 10보다 작은 원소 추출
[1]  1  3  7 NA

> ex[ex %%  2 == 0]
➔ ex에서 2로 나누어 나머지가 0인 즉, 2로 나누어 떨어지는 짝수 추출
[1] NA 12

> ex[is.na(ex)]
➔  ex에서 값이 NA 인 것 추출
[1] NA

> ex[!is.na(ex)]
➔  ex에서 값이 NA가 아닌 것 추출
[1]  1  3  7 12

> ex[ex %%  2 == 0 & !is.na(ex)]
➔ ex에서 짝수이며 NA가 아닌 원소 추출
➔ 두 개의 조건을 & 로 연결
[1] 12

> name <- c("철수", "영희", "길동")
> age <- c(21, 20, 31)
> gender <- factor(c("M", "F", "M"))
> characters <- data.frame(name, age, gender)
> characters[characters$gender == "F", ]
➔ data frame의 경우 조건에 맞는 행을 선택할 때 주로 사용, 성별이 F인 행 추출
➔ [행조건, 열조건] 구조에서 열조건이 빠지면 모든 열의 값 선택
  name age gender
2 영희  20      F

> characters[characters$age < 30 & characters$gender=="M",  ]
➔ 서로 다른 조건 두개를 &로 결합하여 조건 제시
  name age gender
1 철수  21      M
```

흐름 제어 : 조건문과 반복문

조건문

**조건문 함수 : ifelse()**

주어진 조건에 따라 조건이 참일 경우와 거짓일 경우 전달인자를 선택해서 전달하는 함수입니다. ifelse() 는 문장이 아닌 함수입니다.

```
ifelse(test, yes, no)
```

- test : 조건 판별식
- yes : test가 참일 경우 반환할 값
- no : test가 거짓일 경우 반환할 값

다음 예를 통해 ifelse() 에 대해 간략히 알아보겠습니다.

```
> x <- c(6:-4)
> options(digits=3)
```

➔ 출력물을 세자리로 설정

```
> sqrt(x)
 [1] 2.45 2.24 2.00 1.73 1.41 1.00 0.00  NaN  NaN  NaN  NaN
```

경고 메시지가 손실되었습니다

In sqrt(x) : 계산결과가 NaN가 생성되었습니다

➔ 음수에 제곱근을 구할 경우 NaN가 발생하여 경고 발생

```
> options(digits=3)
> sqrt(ifelse(x >= 0, x, NA))
 [1] 2.45 2.24 2.00 1.73 1.41 1.00 0.00   NA   NA   NA   NA
```

➔ ifelse()를 이용하여 0 이상일 경우에 x값을 반환하고 그렇지 않을 경우 NA 반환

**전통의 조건문 : if**

주어진 값, 수식등의 조건이 참과 거짓에 따라 두가지 상태로 나눠 처리하는 구문입니다. 프로그래밍 요소의 기본인 if 를 다양하게 활용하여 상황에 맞는 코드들을 수행할 수 있습니다. 사용예를 통해 자세히 알아보겠습니다.

사용예

```
> x <- c(1, 2, 3)
> x <- factor(x)
```
➔ 숫자로 구성된 벡터 x를 요인으로 재구성

```
> if(is.factor(x)) length(x)
[1] 3
```
➔ is.factor(x) 는 x 가 요인일 경우 TRUE를 그렇지 않으면 FALSE 반환.
➔ x 가 요인이므로 TRUE
➔ if 문에서 TRUE 일 경우 length(x)를 실행한다. 이 경우 TRUE 이므로 length(x) 실행, FALSE 일 경우 실행하지 않고 다음으로 이동
➔ length(x)의 실행값인 3 출력

```
> if(is.factor(x)) {
+       length(x)
+ } else {
+       sum(x)
+ }
[1] 3
```
➔ if문에서 조건을 판단해서 TRUE일 경우와 FALSE일 경우 각각 나눠서 처리할 경우 else 를 같이 사용
➔ 중괄호({)는 여러개의 코드를 하나의 코드 블럭으로 묶는 것으로 하나의 처리단위로 인식. 중괄호 내의 코드는 순차적으로 실행
➔ else를 사용하기 위해 위와 같이 중괄호로 묶어서 사용
➔ else 이후는 주어진 조건이 거짓일 경우 실행, 이 예에서는 조건이 참이므로 length(x)의 결과 출력

```
> if(is.factor(x)) {
+       length(x)
+ } else if(is.integer(x)) {
+       sum(x)
+ } else {
+       paste(x, "element")
+ }
```
➔ else 이후 다시 등장한 if는 앞선 if 조건이 FALSE가 되는 나머지 상황에서 또 다시 조건을 판단할 경우 사용하는 것으로 이 경우 앞선 if 가 판별한 것은 x 가 요인인지 여부에서 요인이 아닐 경우 즉, 요인이 아닌 다양한 상황일 경우 중에서 다시 조건 판단을 하겠다는 의미
➔ if 문은 이와 같이 else 이후에 중첩해서 또 쓸 수 있음.
➔ 이 문장은 처음 판단하는 것은 주어진 x 가 요인인 경우이고, 주어진 x 가 요인이 아닐 경우, 정수형인지 판단하고 정수형이면 갯수를 그렇지 않다면, 즉, 주어진 x 가 요인이 아니고 나머지 상황 중 정수가 아닌 모든 상황에 대해 paste() 함수를 실행

반복문

반복문은 주어진 문장을 반복하면서 수행하는 문장으로 구구단과 같이 일정한 패턴으로 반복되는 문장을 작성할 때 편이를 제공합니다. 현대의 수치해석학에서는 컴퓨터를 이용하여 근사값을 찾는 경우가 많으며 이 중 반복처리는 필수 요소입니다. 반복문을 잘 사용하면 전체 코드의 길이를 획기적으로 줄일 수 있으며 아주 쉽게 계산할 수 있지만 반복문을 잘못 사용할 경우 "무한루프"에 빠져 쓸데없는 자원의 낭비는 물론 심한 경우 컴퓨터 자체를 먹통으로 만들 수 있으니 처음 사용하실 때 많은 주의를 필요로 하며, 반복문은 무한루프에 빠지지 않게 하기 위해 종료 조건을 직접주거나 판단하여 처리하는 경우가 있으므로 잘 살펴봐 주시기 바랍니다.

**무조건 반복하고 본다 : repeat**

반복이라는 기능에 가장 충실한 repeat 문입니다. repeat 은 단순히 repeat 뒤에 나오는 코드 혹은 코드블럭을 반복하므로 코드 자체의 요소가 어느 일정한 시점에 멈출 수 있는 기능이 있거나 판단할 부분을 넣어 주어야 합니다. 사용예를 통해 알아보겠습니다.

사용예

```
> i <- 20
> repeat {
➔ 중괄호로 둘러싸인 부분 반복

+      if(i > 25) {
➔ 반복문을 중지하기 위한 조건 : i가 25 보다 크면 아래의 break 문을 통해 반복중지

+             break
➔ break 문 : 현재의 반복문 정지

+      } else {
➔ 반복을 중지하지 않을 경우, 즉 반복되는 부분

+             print(i)
+             i <- i + 1
➔ i를 1씩 증가시켜 주는 코드
➔ 생략시 i는 계속해서 앞서 지정한 값 20만 사용되고 무한반복에 빠짐

+      }
+ }
[1] 20
[1] 21
[1] 22
[1] 23
[1] 24
[1] 25
➔ 이 코드는 결과적으로 20부터 25까지 1씩 증가하면서 값을 출력
```

**주어진 조건이 참 일때만 실행한다 : while**

while 문은 다른 프로그래밍 언어에서도 흔히 볼 수 있는 반복문입니다. while은 조건을 주어 해당 조건이 참일 경우에만 코드를 반복하는 문장으로 repeat 과 마찬가지로 조건 자체가 FALSE 를 가질 수 있도록 하여 어느 일저시점에서 멈춰줘야 합니다. 마찬가지로 사용예를 통해 알아보겠습니다.

```
> dan <- 2
> i <- 2
> while (i < 10) {
➔ i 가 10보다 작을 때까지 코드 반복 수행
+      times <- i * dan
➔ 변수 times에 현재의 i값과 dan의 값인 2를 곱한 값 저장
+      print(paste(dan, "X", i, " = ", times))
➔ 출력 결과물은 단수 X i의 현재값 = 곱한값의 형태
+      i <- i + 1
➔ i 값 1 증가
+ }
[1] "2 X 2  =  4"
[1] "2 X 3  =  6"
[1] "2 X 4  =  8"
[1] "2 X 5  =  10"
[1] "2 X 6  =  12"
[1] "2 X 7  =  14"
[1] "2 X 8  =  16"
[1] "2 X 9  =  18"
➔ 출력 결과는 구구단의 2단
```

**전통의 반복문! 하지만 R에선 조금 다르다 : for (수식)**

for는 어느 프로그래밍 언어에나 존재합니다. 하지만 R에서는 약간 다르게 표현됩니다. for는 수식 부분에 보통 자료구조가 들어가 해당 구조 안의 모든 원소의 갯수 만큼 반복을 수행합니다. 마찬가지로 사용예를 보면서 알아보겠습니다.

```
> dan <- 9
> for( i in 2:9) {
➔ 반복은 벡터 2:9 이며 매 반복시 2부터 9 사이의 값을 가져와 i에 저장
➔ 첫번째 반복에서 i에 벡터 (2, 3, 4, …, 9) 중 첫번째 원소인 2 저장
+      times <- i * dan
+      print(paste(dan, "X", i, " = ", times))
+ }
➔ 반복 횟수는 벡터 (2, 3, 4, …, 9)의 원소의 갯수인 8회
➔ 반복이 종료되는 시점은 벡터의 마지막 원소까지 가져와 반복하고 종료
[1] "9 X 2  =  18"
[1] "9 X 3  =  27"
[1] "9 X 4  =  36"
[1] "9 X 5  =  45"
[1] "9 X 6  =  54"
[1] "9 X 7  =  63"
[1] "9 X 8  =  72"
[1] "9 X 9  =  81"

> str <- c("a", "b", "c")
> for (i in str) {
➔ i는 벡터 str을 순회하면서 각 원소를 가져옴
➔ 처음 i에 저장되는 값은 벡터 str의 첫번째 값인 “a”
+      print(i)
+ }
[1] "a"
[1] "b"
[1] "c"
➔ 결과적으로 벡터 str에 있는 모든 원소 출력
```

사용자 정의 함수 만들기

R은 다양한 내장 함수를 갖추고 있습니다만 사용자가 원하는 기능을 함수로 만들어 사용할 수 있는 기능도 제공합니다. 이 때 필요로 하는 지시어는 function입니다.

function : 사용자 정의 함수 생성문

```
> 함수명 <- function(전달인자1, 전달인자2, …) {
+ 함수 작성을 위한 코드 들
+ return(반환값)
+ }
```

- function 이 후 생성되는 기능을 담당하는 함수의 이름은 "함수명"
- 소괄호 사이에 필요로 하는 전달인자를 통해 값등을 전달받을 변수 나열
- 함수 기능을 위한 코드 작성
- return 을 통해 함수의 기능을 다하고 함수를 호출한 코드에 전달할 값 전달

사용예 : 팩토리얼을 구하는 함수

```
> myfactorial <- function (x) {
➔ 작성할 함수의 이름은 myfactorial
➔ 필요로 하는 전달인자는 하나이며 사용자가 함수 사용시 전달되는 값은 변수 x에 저장

+       fact <- 1
+       i <- x
+       while (i > 1) {
+               fact <- fact * i
+               i <- i - 1
+       }
+       return(fact)
➔ 함수 수행후 전달할 값은 변수 fact의 값
+ }
>
> myfactorial(5)
➔ 함수 myfactorial 사용
➔ 전달할 값은 5

[1] 120
➔ 함수 myfactorial 이 수행하고 결과값 120 반환(return문을 통한 반환)
```

심화 예제 : z-test

모집단의 표준편차를 알 경우 단일표본의 평균검정에서 사용하는 검정통계량은 표준정규분포로부터 계산합니다. 하지만 R은 z-test와 관련한 함수를 갖고 있지 않습니다. 이를 위해 R의 t검정 함수인 t.test() 를 참조하여 z.test 함수를 만들어 보겠습니다. (t.test() 함수는 뒤에서 다루겠습니다.) t.test() 함수는 검정할 평균과 대안가설을 받아 검정통계량과 p-value를 반환해주는 함수로 이와 유사하게 만들어 보겠습니다.

```
ztest <- function(x, sd.p=NULL, mu=0, alternative="two.sided") {
  n <- length(x)
  sd <- ifelse(is.null(sd.p), sd(x), sd.p)
  se <- sd / sqrt(n)
  z <- (mean(x) - mu) / se
  t.stat <- z
  bias <- x - mean(x)
  p.value <- 2*(ifelse(t.stat < 0, pnorm(t.stat), 1-pnorm(t.stat)))
  out <- list(statistic=t.stat, p.value=p.value, bias=bias)
  return( out )
}
x <- rnorm(30)
ztest(x)
```

Package 사용하기

Package의 설치와 사용하기

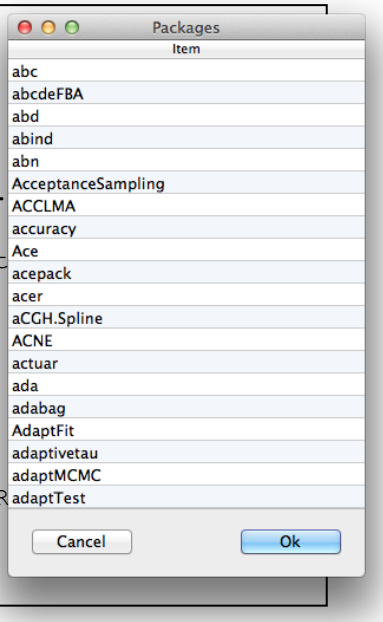
R은 자체로도 많은 일을 할 수 있습니다만, 여러 개발자들로부터 다양한 Package를 제공받아 사용할 수 있습니다. R을 기본 플랫폼으로하여 전세계의 다양한 분야의 다양한 아이디어들을 사용할 수 있도록 하는 것으로 사용자의 필요에 따라 설치하고 사용할 수 있습니다. R의 다양한 Package 들에 대한 정보는 http://cran.nexr.com/web/packages/ 에서 찾으실 수 있습니다. 또한 R을 설치하고 나면 기본적으로 설치되는 Package들이 있는 데 getOption() 으로 확인하실 수 있습니다.

```
> getOption("defaultPackages")
[1] "datasets"  "utils"   "grDevices" "graphics"  "stats"   "methods"
```

Package를 설치하고 사용하는 방법에 대해 알아보겠습니다.

R에서 Package를 설치하는 함수는 install.packages() 입니다. 아무런 전달인자 없이 입력하면 전체 Package들의 목록이 나옵니다. 전달인자로 Package의 이름을 넣으면 해당 패키지를 설치하는 과정을 거칩니다.

```
> install.packages()
> install.packages("vcd")
also installing the dependency ‘colorspace’

URL
'http://cran.nexr.com/bin/macosx/leopard/contrib/2.
1.tgz'를 시도하고 있습니다
Content type 'application/x-gzip' length 502980 byt
열린  URL
==================================================
downloaded 491 Kb
…

다운로드된
/var/folders/7n/25h8nh855l74w8nzw37901jm0000gn/T//R
ed_packages에 있습니다
>
```




Package 설치가 완료되었습니다. 이제 설치된 Package를 사용해 보겠습니다. 설치된 Package를 사용하기 위한 함수는 library(“Package 명”) 입니다.

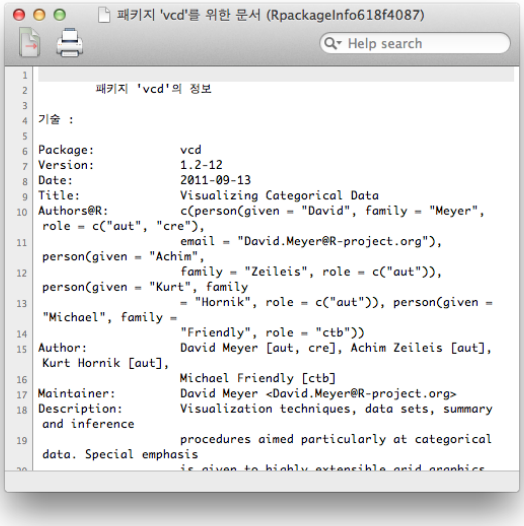
```
> library("vcd")
요구된 패키지 MASS를 로드중입니다
요구된 패키지 grid를 로드중입니다
요구된 패키지 colorspace를 로드중입니다
Warning messages:
1: 패키지 'vcd'는 버전 2.13.1의 R미만에서 작성되어졌습니다
2: 패키지 'colorspace'는 버전 2.13.2의 R미만에서 작성되어졌습니다
>
```

설치과정중에도 마찬가지로 필요로 하는 다른 Package를 R이 알아서 설치하는데 Package를 사용하기 위해 library()를 입력하면 마찬가지로 필요한 Package를 같이 로드합니다. 그리고 조금 시간이 지난 자료에 대해서는 약간의 경고 메세지를 같이 보내줍니다. 만일 결과가 예상과 다르다면 필요로 하는 Package들을 update.packages() 을 통해 package를 최신 버전으로 유지할 수 있습니다.

```
> update.packages()
boot :
 Version 1.2-43 installed in
/Library/Frameworks/R.framework/Versions/2.13/Resources/library
 Version 1.3-2 available at http://cran.nexr.com
Update (y/N/c)?  y
➔ update 하려면 y를 하지않고 다음으로 넘어가려면 N을 멈추려면 c 입력
cluster :
 Version 1.13.3 installed in
 /Library/Frameworks/R.framework/Versions/2.13/Resources/library
 Version 1.14.2 available at http://cran.nexr.com
Update (y/N/c)?  y
…

>
```

설치한 Package에 대해 도움말을 얻고자 하면 help(package="package명") 을 실행해서 도움말을 얻으실 수 있습니다.

```
> help(package="vcd")
➔ vcd packeage의 도움말을 얻는다.
➔ 경우에 따라 웹 브라우저 혹은 다음과 같은 도움말 창을 통해 결과 확인
```

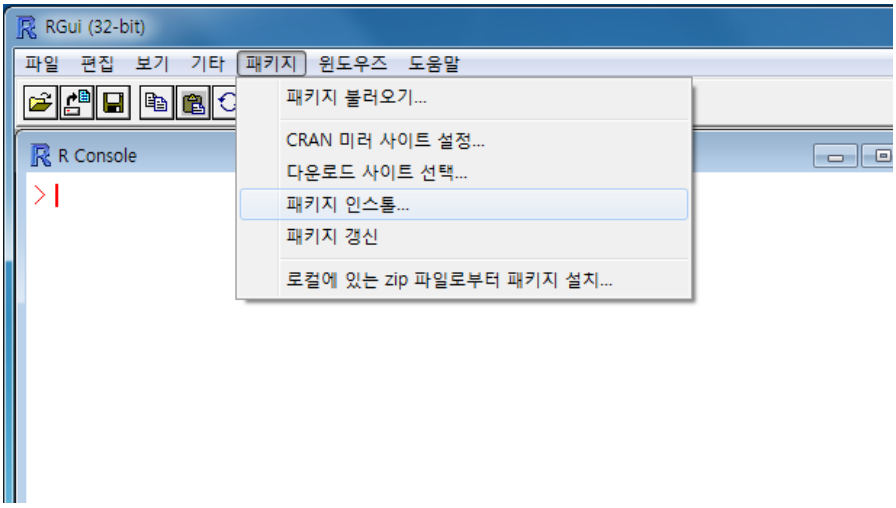



Package의 설치 : 메뉴 활용

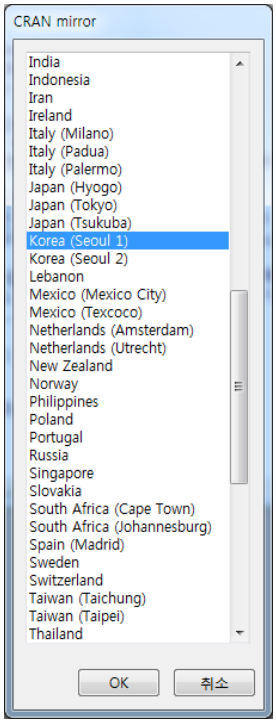
앞서 설명드린 과정은 일반적인 명령어를 입력하여 패키지를 설치하는 과정입니다만 Windows를 사용할 경우 좀 더 쉽게 메뉴를 통해 설치할 수 있습니다[2]. 이 과정을 화면과 함께 살펴보도록 하겠습니다. 설치할 패키지는 Hmisc 입니다.

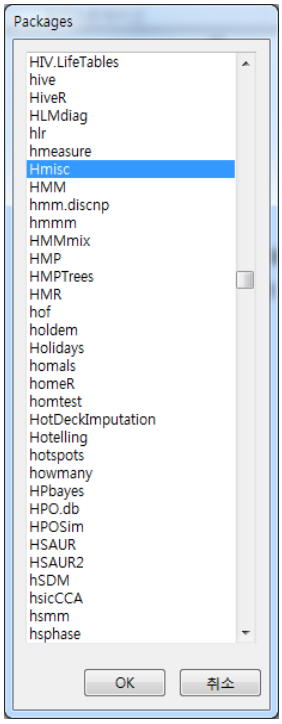
[2] 물론 Mac 등에서도 메뉴를 제공합니다.

1. 패키지 메뉴의 "패키지 인스톨" 클릭

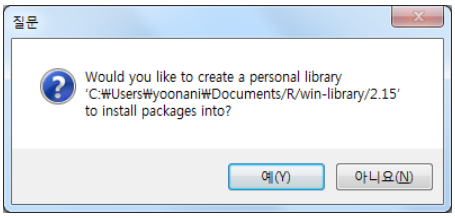



2. 가까운 곳 설정 (우리나라의 경우 "Korea (Seoul1)", "Korea (Seoul2)" 중 선택) 과 설치할 패키지 선택 (우리가 설치할 "Hmisc" 선택)

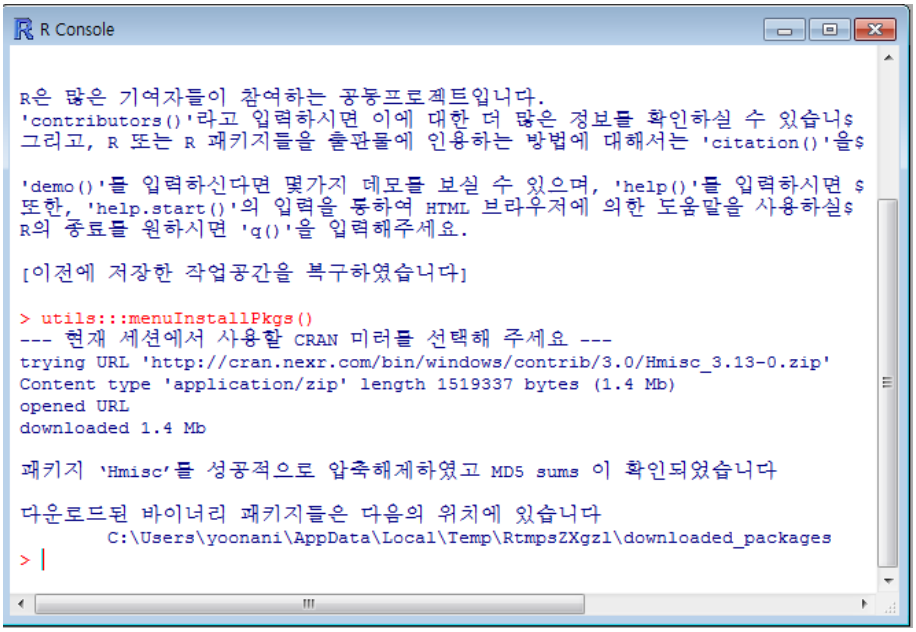


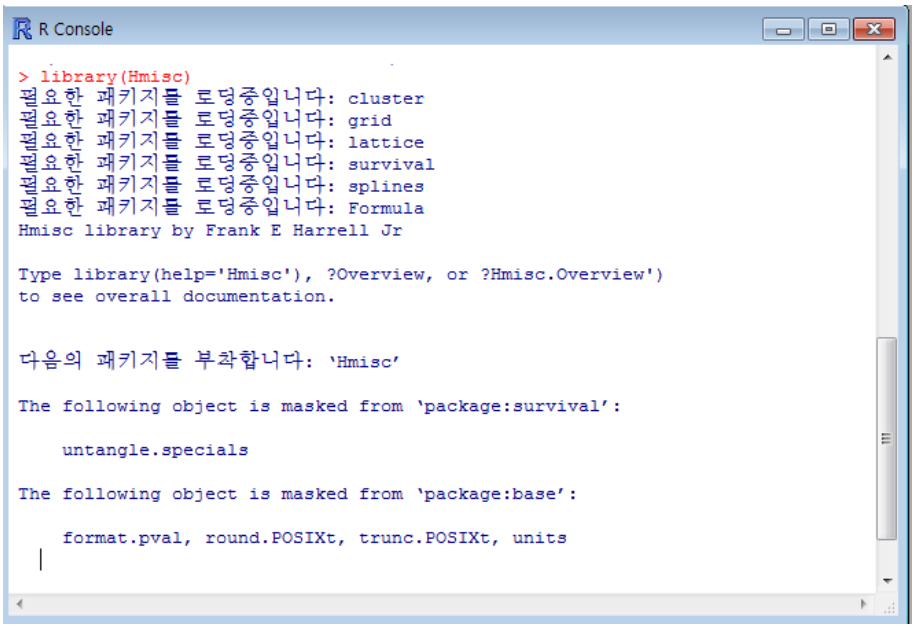

3. 설치할 디렉토리에 대한 안내 (R이 사용 사용자 하드디스크의 특정 위치에 쓰는 작업을 하므로 사용자에게 진행할 것인지 확인, 물어보지 않을 수도 있음)



4. Hmisc를 다운로드 받고 설치하는 과정




5. Hmisc를 사용하기 위해 library(Hmisc) 입력




이상의 과정을 통해 Windows 에 설치된 R에서 원하는 패키지를 설치할 수 있습니다.

# R에서의 그래프

plot () : 그래프의 기초[3]

R의 내장함수인 plot()은 산점도를 출력하는 가장 기본적인 그래프 함수입니다. plot()을 통해 R의 그래프의 기초에 대해 알아보겠습니다. 여기서 사용될 data는 설치될 때 기본으로 설치되는 R의 datasets package에 포함되어 있는 Puromycin Data[4]를 사용하겠습니다.

```
> str(Puromycin)
'data.frame':   23 obs. of  3 variables:
 $ conc : num  0.02 0.02 0.06 0.06 0.11 0.11 0.22 0.22 0.56 0.56 ...
 $ rate : num  76 47 97 107 123 139 159 152 191 201 ...
 $ state: Factor w/ 2 levels "treated","untreated": 1 1 1 1 1 1 1 1 ...
 - attr(*, "reference")= chr "A1.3, p. 269"
```

자료 준비를 위해 Puromycin 자료를 처리군과 비처리군으로 나누어 보겠습니다.

```
> PuroTrt <- subset(Puromycin, state=="treated")
> PuroUnTrt <- subset(Puromycin, state=="untreated")
```

위와 같이 자료를 나누면 PuroTrt는 state 변수의 값이 "treated"인 데이터 프레임으로 저장되고 PuroUnTrt는 state 변수의 값이 "untreated"인 데이터 프레임으로 저장됩니다.

산점도를 그려보자

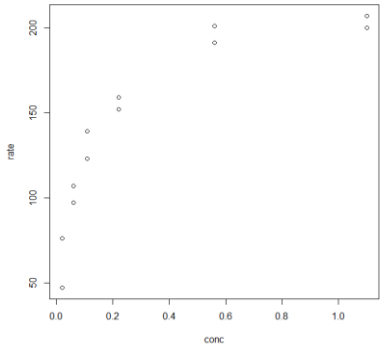
R에서 그래프를 작성하기 위해 그래프를 작성할 그래픽 장치를 활성화하고 그래프의 기본 요소인 축을 포함하는 그래프 영역을 확보하고 그 위에 그려질 그래프의 형태를 결정하는 plot()과 같은 "고수준 그래픽 함수"와 기존에 그려진 그래프 위에 각종 객체를 그려넣는 "저수준 그래픽 함수"가 제공됩니다. 지금 우리가 그려볼 그래프는 고수준 그래픽 함수인 plot()을 이용하여 가장 단순한 그래프를 작성해보도록 하겠습니다.

```
> plot(rate ~ conc, data=PuroTrt)
```

---

[3] Ulrich Halekoh의 Basic Graphics in R(http://gbi.agrsci.dk/statistics/courses/phd07/material/Day1-R-Intro/basic-raphics.pdf) 를 참조하여 작성하였습니다.

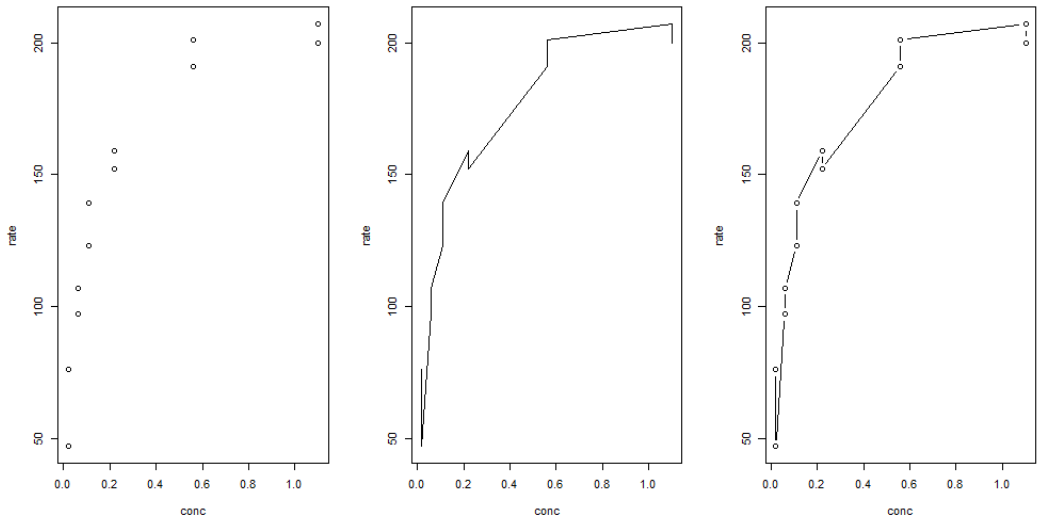
[4] Puromycin Data는 conc, rate, state를 변수로 갖는 데이터 프레임 자료로 Puromycin이 처리된 셀과 그렇지 않은 셀을 포함하는 요소반응에서의 기질 농도 대비 반응 속도를 담고 있다.




plot() 함수의 첫번째 전달인자로 된 rate ~ conc 는 y축을 rate로 x 축을 conc로 하여 산점도를 작성하라는 의미입니다. R에서 ~ 는 모형을 나타낼 때 사용하는데 이 경우 rate = conc 와 같은 수식으로 인지하시면 됩니다. 그리고 두번째 전달인자인 data는 산점도를 그릴 데이터를 지정합니다. 우리의 경우 처리된 자료(PuroTrt)를 산점도로 작성하였습니다.

R의 plot() 함수는 기본적으로 xy 좌표를 바탕으로 점을 찍습니다만 type 이라는 전달인자를 이용하여 점들을 연결하는 그래프를 그릴 수 있도록 합니다. 다음의 코드를 살펴보겠습니다.

```
> par(mfrow=c(1, 3))
➔ 하나의 그래픽 장치에 1행 3열로 그래프를 배치하도록 하는 명령입니다.
> plot(rate ~ conc, data=PuroTrt, type="p")
> plot(rate ~ conc, data=PuroTrt, type="l")
> plot(rate ~ conc, data=PuroTrt, type="b")
```



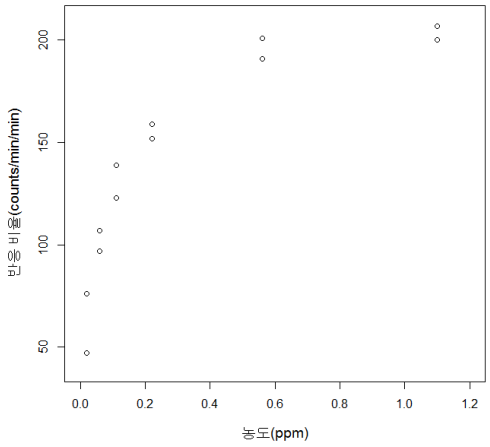

type으로 전달되는 값이 "p"일(**p**unctuation) 경우 기본값으로 점만 출력하고, "l"일(**l**ine) 경우 점들을 연결하는 선만 출력합니다. 점과 연결선 모두를 나타내려면 "b"를(**b**oth) 전달합니다.

그래프를 꾸며봅시다

앞서 작성한 그래프는 가장 기본적인 형태의 출력결과입니다. 이제 몇 가지 plot()에 전달되는 전달인자를 통해 원하는 결과물로 바꿔보기로 하겠습니다. 먼저 그래프의 각 축 요소들을 변경시켜 보겠습니다. 다음과 같은 전달인자를 사용합니다.

- 각 축의 축 제목 : xlab, ylab
- 축 제목의 크기 : cex.lab
- 각 축의 범위 : xlim, ylim

```
>  plot(rate ~ conc, data=PuroTrt, xlim=c(0, 1.2), ylim=c(40, 210),
+ xlab="농도(ppm)", ylab="반응 비율(counts/min/min)", cex.lab=1.2)
```



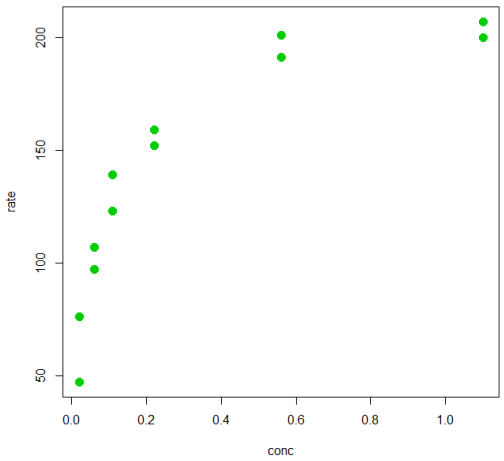

각 전달인자에 대한 간략한 설명입니다.

- xlim=c(0, 1.2) : x 축의 범위를 0부터 1.2 까지로 작성
- ylim=c(40, 210) : y 축의 범위를 40부터 210 까지로 작성
- xlab="농도(ppm)" : x축 제목을 "농도(ppm)"로 변경
- ylab="반응 비율(counts/min/min)" : y축 제목을 "반응 비율(counts/min/min)"로 변경
- cex.lab=1.2 : 축 제목의 크기를 기준 크기의 1.2배로 함

이제 plot()을 통해 출력되는 각 점들을 변경해 보겠습니다. 여기에 사용되는 전달인자는 다음과 같습니다.

- 산점도로 찍히는 점들의 모양 변경 : pch
- 산점도로 찍히는 점들의 색깔 변경 : col
- 산점도로 찍히는 점들의 크기 변경 : cex

```
> plot(rate ~ conc, data=PuroTrt, pch=16, col=3, cex=1.5)
```

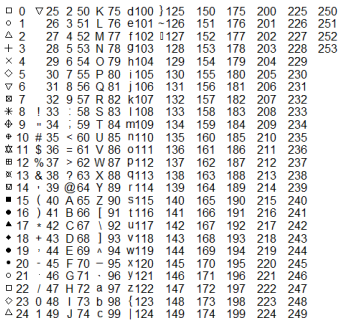


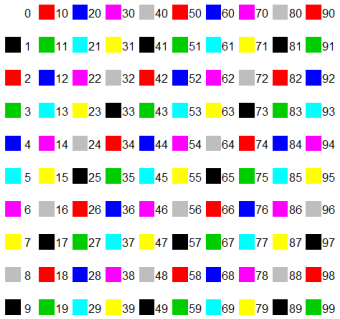

각 전달인자에 대한 간략한 설명입니다.

- pch=16 : 찍히는 점의 모양을 16번(disc)으로 함, 또한 특정문자 입력시 해당하는 문자가 출력(예. pch="T", pch="1" 등)
- col=3( 혹은 col='green') : 점의 색깔을 3번(green)으로 함
- cex=1.5 : 찍히는 점의 크기를 기준 크기의 1.5배로 함

pch, col 등에 값으로 들어가는 16, 3등의 의미를 알려면 R의 Hmisc 패키지를 설치하고 어떤 의미를 갖고 있는지 다음과 같이 확인할 수 있습니다.

```
library(Hmisc)
show.pch()
show.col()
```



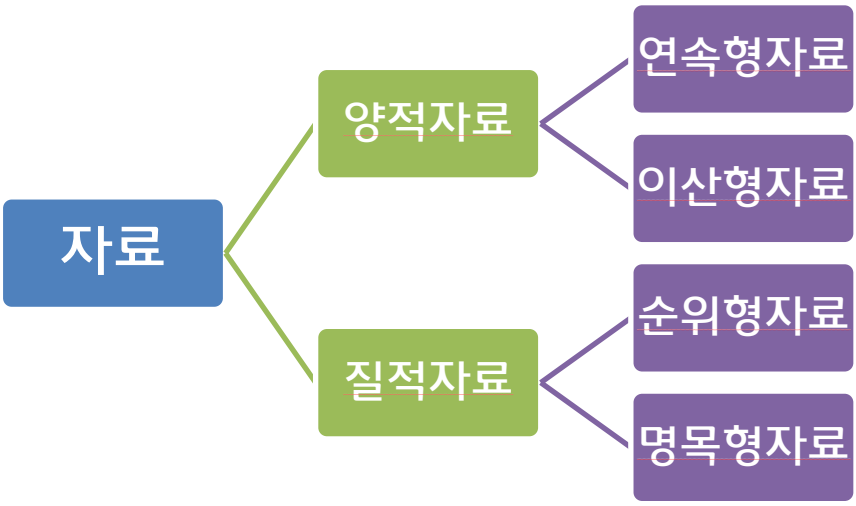


저수준 그래프 함수의 사용

고수준 그래프 함수는 그래프를 그리기 위한 그래픽 장치를 활성시키고 각 그래프 함수에 맞는 그래프를 작성하는 기능을 한다고 앞서 말씀드렸습니다. 지금 우리가 알아보고자 하는 것은 저수준 그래프 함수로 이 함수들은 고수준 그래프 함수를 통해 생성된 모든 그래프 위에 추가적인 객체들을 삽입할 수 있도록 합니다.

자료의 종류에 따른 그래프 작성

기초통계학 텍스트들에서 이야기하는 자료의 종류에 따른 그래프를 작성하는 방법에 대해 알아보도록 하겠습니다. 통계학에서 다루는 자료의 유형은 다음의 네가지입니다.




자료의 양적인 크기를 다루는 자료인 양적 자료는 우리가 보통 사칙연산을 통해 계산할 수 있는 자료들로 관측 대상이 되는 자료의 증감이 연속적으로 발생하는 연속형자료와 자료의 증감이 계단 형태로 발생하는 이산형 자료를 말합니다.

자료의 양적인 크기에 상관없이 자료 자체에 의미를 부여하는 질적 자료는 범주를 나타내는 명목형 자료와 범주를 나타내지만 순서를 갖고 있는 순위형 자료를 말합니다.

양적 자료의 예를 들면 키와 체중 등은 우리가 사용되는 단위를 통해 숫자를 나타내지만 실제 키와 체중은 지속적으로 증가 혹은 감소하는 자료는 이 경우 연속형 자료이며, 각 반의 안경을 쓴 학생의 수, 동전을 두 번 던져 앞면이 나오는 횟수 등은 이산형 자료입니다.

질적 자료의 예를 들면 성별, 왼손 잡이 여부 등은 명목형 자료이며 설문지에서 자주 보는 "매우 좋다", "좋다", "보통이다", "안 좋다", "매우 안 좋다" 등의 점 측도는 순위형 자료입니다.

그럼 이제 각 자료 유형에 맞는 그래프를 R을 통해 작성해 보도록 하겠습니다.

막대그래프 : barplot()

R의 내장 데이터 중 ldeaths 는 영국의 3개 질병(기관지염, 폐기종, 천식)으로 인한 월별 사망자수를 1974년부 1979년까지 모아놓은 자료입니다. 이 자료를 통해 이산형 자료의 그래프로 많이 사용되는 barplot()에 대해 알아보겠습니다. 먼저 ldeaths 자료입니다.

```
> ldeaths
     Jan  Feb  Mar  Apr  May  Jun  Jul  Aug  Sep  Oct  Nov  Dec
1974 3035 2552 2704 2554 2014 1655 1721 1524 1596 2074 2199 2512
1975 2933 2889 2938 2497 1870 1726 1607 1545 1396 1787 2076 2837
1976 2787 3891 3179 2011 1636 1580 1489 1300 1356 1653 2013 2823
1977 3102 2294 2385 2444 1748 1554 1498 1361 1346 1564 1640 2293
1978 2815 3137 2679 1969 1870 1633 1529 1366 1357 1570 1535 2491
1979 3084 2605 2573 2143 1693 1504 1461 1354 1333 1492 1781 1915
> str(ldeaths)
Time-Series [1:72] from 1974 to 1980: 3035 2552 2704 2554 2014 ...
```

**여기서 잠깐!!!**

위에 보시면 str() 함수를 사용했습니다. 함수명 str은 구조를 뜻하는 영어단어 structure의 줄임말로 전달되는 자료의 구조를 나타냅니다. 위에 주어진 자료 ldeaths는 겉보기에 데이터 프레임처럼 보이지만 ts()를 함수를 이용해 만든 시계열(Time-Series)자료입니다. 이 자료에서 1월의 자료만 따로 불러오기 위해서는 ldeaths[seq(1, 72, by=12)] 를 통해 1번째, 13번째, 35번째, 47번째, 59번째 자료를 가져와야 합니다. 물론 이렇게 해도 됩니다만 이전에 설명드리지 못한 몇가지 함수를 같이 설명할 겸 matrix 형태로 바꿔보도록 하겠습니다. 다음 코드를 보실까요?

```
> ld <- matrix(ldeaths, nrow=6, byrow=T)
> colnames(ld) <- month.abb
> rownames(ld) <- 1974:1979
> ld
     Jan  Feb  Mar  Apr  May  Jun  Jul  Aug  Sep  Oct  Nov  Dec
1974 3035 2552 2704 2554 2014 1655 1721 1524 1596 2074 2199 2512
1975 2933 2889 2938 2497 1870 1726 1607 1545 1396 1787 2076 2837
1976 2787 3891 3179 2011 1636 1580 1489 1300 1356 1653 2013 2823
1977 3102 2294 2385 2444 1748 1554 1498 1361 1346 1564 1640 2293
1978 2815 3137 2679 1969 1870 1633 1529 1366 1357 1570 1535 2491
1979 3084 2605 2573 2143 1693 1504 1461 1354 1333 1492 1781 1915
> str(ld)
 num [1:6, 1:12] 3035 2933 2787 3102 2815 ...
 - attr(*, "dimnames")=List of 2
  ..$ : chr [1:6] "1974" "1975" "1976" "1977" ...
  ..$ : chr [1:12] "Jan" "Feb" "Mar" "Apr" ...
```

1) ldeaths 자료를 6개의 행이고 행부터 채워나가 6행 12열짜리 matrix를 만들고 이름을 ld로 합니다.
2) ld의 열 이름을 각 달의 약자로 변경합니다. month.abb 는 미국식 달 이름의 약자들의 벡터를 나타냅니다.
3) ld의 행 이름을 1974, 1975, … , 1979 로 변경합니다.

위와 같이 간단하게 주어진 자료의 구조를 변경하고 원하는 행과 열의 이름을 줄 수 있습니다.

**함수의 원형**

- 전달인자
  - height : 각 기둥의 높이를 나타내는 벡터 혹은 열을 갖고 있는 자료구조로 각 열이 한 개의 기둥을 나타내는 기둥 그룹.
  - width(=1) : 각 기둥의 폭으로 기본 값은 1로 모두 동일
  - space : 각 기둥간의 간격. 상황에 따라 matrix 로 줄 수 있으며 beside 전달인자와 함께 작동
  - names.arg : 각 기둥 혹은 기둥 그룹의 이름을 나타내는 문자열 벡터
  - legend.text : 각 기둥의 범례를 위한 문자열
  - beside : 논리값으로 기둥 그룹에서 FALSE의 경우 기둥 그룹 내 각 막대가 쌓아진 형태로 나타나고 TRUE일 경우 옆에 배열
  - horiz : 논리값으로 FALSE 의 경우 막대가 아래에서 위로 나타나며 TRUE일 경우 왼쪽에서 오른쪽으로 크기 만큼 나타남
  - density : 기둥을 채우는 선의 밀도로 1인치당 빗금의 갯수를 나타낸다. 기본값은 NULL로 음영선을 나타내지 않는다.
  - angle : 기둥을 채우는 선의 각도로 시계방향을 기준으로 각도를 나타낸다.
  - border : 기둥의 테두리 색을 나타내는 것으로 border=NA 일 경우 테두리가 없는 막대이며 col을 통해 빗금을 나타내면 border=TRUE 를 통해 빗금 색과 동일하게 나타낸다.
- 그래프 공통 전달인자 : 거의 모든 그래프 함수에서 함께 사용
  - main, sub : 그래프의 주제목과 부제목에 해당하는 문자열
  - xlab, ylab : 그래프의 x 축과 y 축의 이름
  - xlim, ylim : 그래프의 x 축과 y 축 값의 하한과 상한의 벡터
  - axes : 논리값으로 TRUE는 기본값으로 축을 나타내고 FALSE 는 축을 제거
  - col : 그래프의 색을 나타내는 벡터값으로 barplot의 경우 기둥을 채우는 선의 색으로써 값이 주어지지 않으면 회색(grey)가 사용되고 기둥 그룹의 경우 gamma로 조정된 회색 계열들이 채워진다.

이제 막대그래프를 작성하기 위해 다음과 같이 입력해 보겠습니다.

```
> par(mfrow=c(2,2))
```
→ 그래프를 한 화면을 2행 2열로 나눠 각 칸에 들어가도록 합니다.

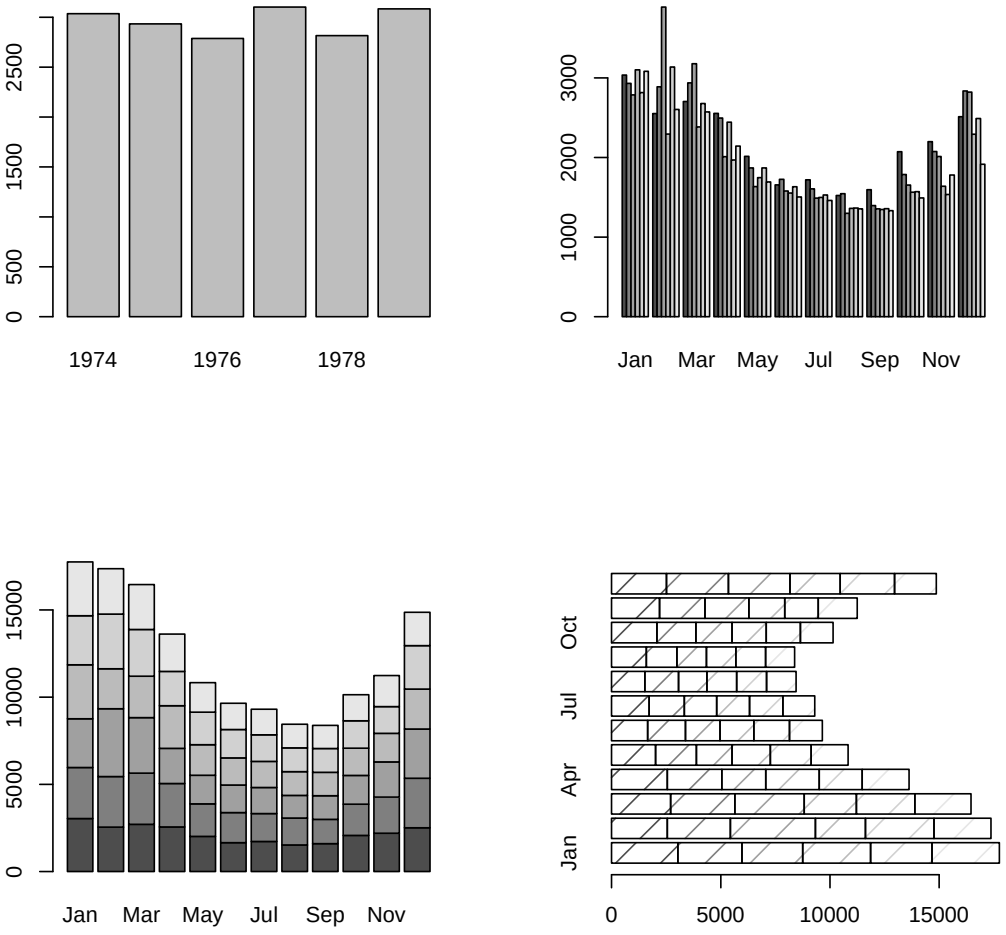
```
> barplot(ld[,"Jan"])
```
→ 1월달의 막대그래프를 그립니다.

```
> barplot(ld, beside=T)
```
→ 각 월별로 년도를 하위 그룹으로 하여 막대 그래프를 그립니다. 각 년도의 자료는 옆으로 표시됩니다.

```
> barplot(ld, beside=F)
```
→ 위의 그래프와 동일하나 각 년도는 차곡차곡 쌓아서 표시합니다.

```
> barplot(ld, beside=F, horiz=T, density=5)
```
→ 위의 그래프를 아래에서 위 방향이 아닌 좌에서 우방향이 되도록 막대를 눕히고 채우는 선은 인치당 5개가 들어가도록 합니다.

이 명령에 대한 결과입니다.



히스토그램 : hist()

R의 내장 데이터 중 mtcars는 미국의 자동차 잡지 Motor Trend에서 1974년도에 소개한 32종의 차에 대해 10가지 특성(연비, 실린더 수, 무게, 변속기 등)을 정리한 자료입니다. 이 자료로부터 연속형 자료에 해당하는 mpg 속성(Miles/Gallon, 연비)으로 연속형 자료의 그래프로 많이 사용되는 hist()에 대해 알아보겠습니다. 먼저 mtcars 자료입니다.

```
> head(mtcars)
                  mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4        21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag    21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710       22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive   21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant          18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
> str(mtcars)
'data.frame':   32 obs. of  11 variables:
 $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
 $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
 $ disp: num  160 160 108 258 360 ...
 $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
 $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
 $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
 $ qsec: num  16.5 17 18.6 19.4 17 ...
 $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
 $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
 $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
 $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

**함수의 원형**

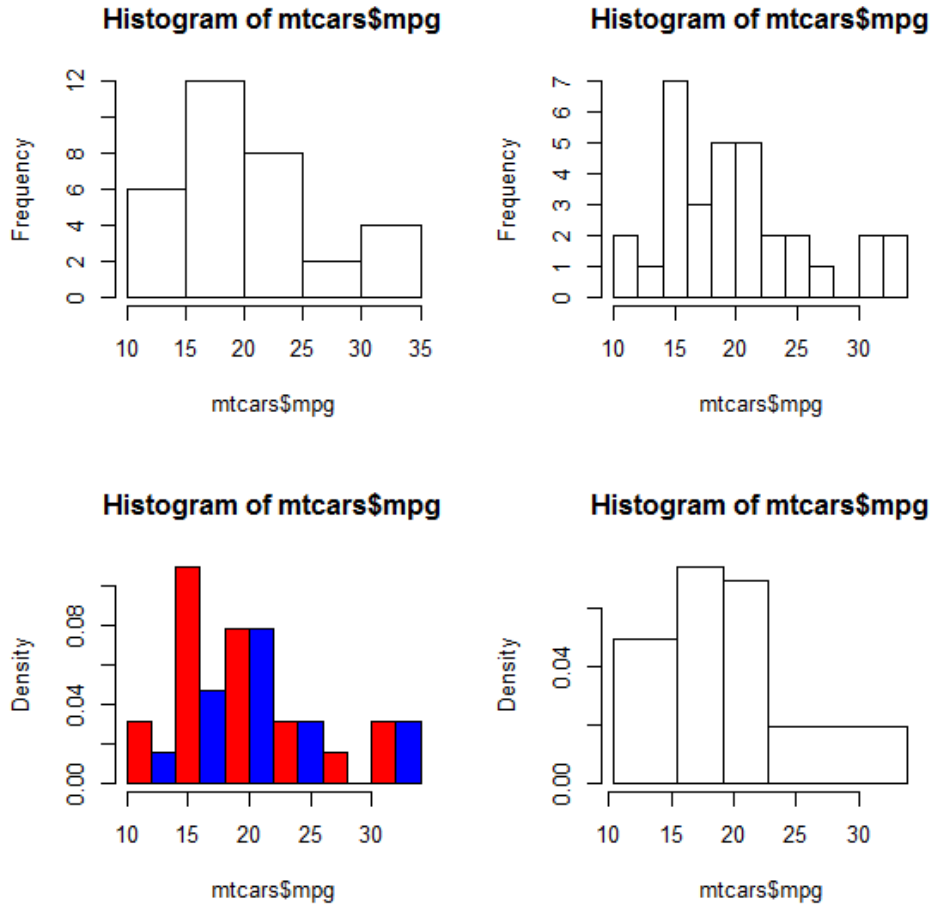
- 전달인자
    - breaks : 기둥의 절단점으로 다음 중 하나의 형태를 취한다.
        - 막대간의 절단점 벡터
        - 절단점 벡터를 계산하기 위한 함수
        - 기둥 개수(스칼라)
        - "Scott" and "FD" 같은 기둥 개수를 계산하는 알고리즘 이름
        - 기둥개수를 계산하는 함수
    - freq : TRUE이면 빈도로 나타내고 FALSE이면 확률밀도로 나타냄
    - probability : freq 값이 FALSE일 경우의 alias
    - include.lowest : TRUE이면 각 기둥의 최소값 포함, breaks가 벡터가 아니면 무시
    - right : TRUE이면 기둥의 우측이 닫힌(right-closed) 구간
    - density : 기둥을 채우는 선의 밀도로 1인치당 빗금의 갯수를 나타낸다. 기본값은 NULL로 음영선을 나타내지 않는다.
    - angle : 기둥을 채우는 선의 각도로 시계방향을 기준으로 각도를 나타낸다.

- border : 기둥의 테두리 색을 나타내는 것으로 border=NA 일 경우 테두리가 없는 막대이며 col을 통해 빗금을 나타내면 border=TRUE 를 통해 빗금 색과 동일하게 나타낸다.
- 그래프 공통 전달인자 : barplot() 참조

이제 히스토그램을 작성하기 위해 다음과 같이 입력해 보겠습니다.

```
> par(mfrow=c(2,2))
→ 그래프를 한 화면을 2행 2열로 나눠 각 칸에 들어가도록 합니다.

> hist(mtcars$mpg)
→ 가장 기본적인 히스토그램을 그립니다.

> hist(mtcars$mpg, breaks=12)
→ 히스토그램의 막대를 12개로 합니다.

> hist(mtcars$mpg, breaks=12, freq=FALSE, col=c("red", "blue"))
→ y축의 값을 빈도가 아닌 비율로 하고 막대의 색을 빨강과 파랑이 교차로 나오게 합니다.

> hist(mtcars$mpg, breaks=quantile(mtcars$mpg))
→ 막대가 25%, 50%, 75%, 100% 가 되는 값이 되도록 합니다.
```

이 명령에 대한 결과입니다.



그래프 활용 예제 : 가설검정을 위한 그래프 출력

```
> height <- c(178, 172, 172, 172, 170, 176, 174, 175, 178, 179, 171,
+ 166, 175, 177, 178, 169, 172, 177, 171, 185, 172, 168, 175, 181, 180,
+ 168, 169, 173, 175, 175)
> mean <- height
> num <- mean(height) - 170
> denom <- 4 / sqrt(length(height))
> z <- num / denom
```

→ z-test 검정통계량 계산

```
> x <- seq(-1, 6, 0.01)
```

→ x축을 위한 값 생성

```
> d <- dnorm(x)
```

→ x에 해당하는 표준정규분포의 pdf 계산

```
> plot(x, d, type="l", axes=F, xlab="", ylab="")
```

→ 축을 없앤 산점도를 그려 선으로 연결

```
> axis(side=1)
```

→ x축만 생성

```
> segments(seq(1.645, 6, 0.01), 0, seq(1.645, 6, 0.01),
+ dnorm(seq(1.645, 6, 0.01)), col=2)
```

→ 붉은 선분을 만들어 색칠한 효과 생성

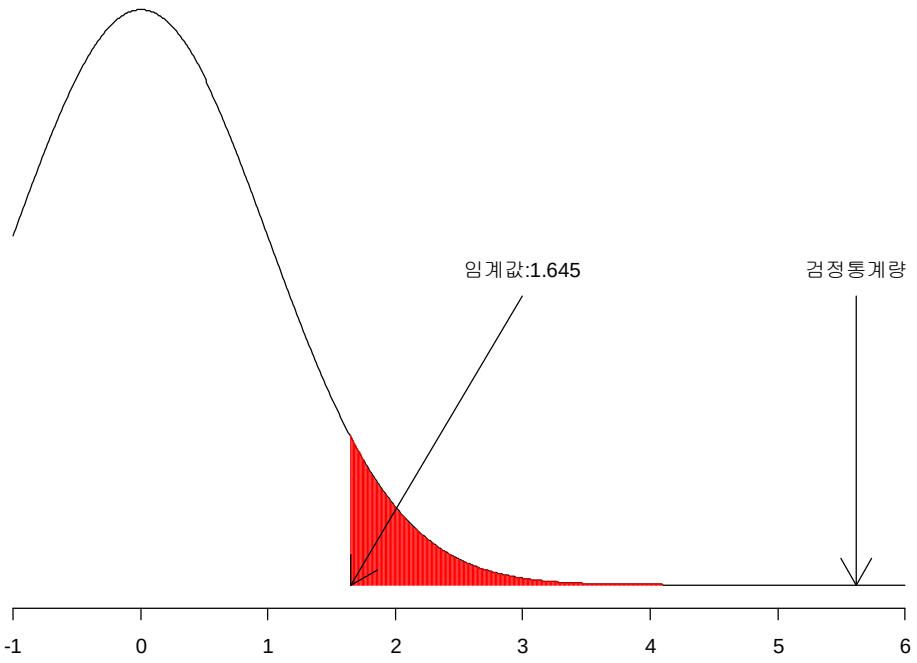
```
> arrows(z, 0.2, z, 0)
```

→ 화살표 생성

```
> text(z, 0.22, labels=c("검정통계량"))
```

→ 문자열 출력

```
> arrows(3, 0.2, 1.645, 0)
> text(3, 0.22, labels=c("임계값:1.645"))
```



# R의 유용한 함수 : 통계

## 분포함수와 관련된 함수

R에서의 통계 분포 함수는 "접두어 + 분포명"의 구조를 갖습니다.
접두어는 분포함수를 통해 얻고자 하는 값을 가리키는 것으로 다음과 같습니다.

| 접두어 | 기능 |
|---|---|
| d(ensity) | 확률밀도함수(이산형의 경우 확률질량함수) 값을 구한다. |
| p(robability) | 누적분포함수(통계에서 분포함수는 누적분포함수) 값을 구한다. |
| q(uantile) | 누적확률에 해당하는 분포함수의 x값(분위)을 구한다. |
| r(andom) | 난수 생성 |

또한 R에서 제공하는 통계분포함수와 R에서의 이름은 다음과 같습니다.

| 분포함수 | **R 이름** | 분포함수 | **R 이름** |
|---|---|---|---|
| Beta | **beta** | Lognormal | **lnorm** |
| Binomial | **binom** | Negative Binomial | **nbinom** |
| Cauchy | **cauchy** | Normal | **norm** |
| Chisquare | **chisq** | Poisson | **pois** |
| Exponential | **exp** | Student t | **t** |
| F | **f** | Uniform | **unif** |
| Gamma | **gamma** | Tukey | **tukey** |
| Geometric | **geom** | Weibull | **weib** |
| Hypergeometric | **hyper** | Wilcoxon | **wilcox** |
| Logistic | **logis** | | |

접두어와 결합하여 R에서 사용할 수 있는 분포함수를 통해 쉽게 분포함수에 기반한 시뮬레이션을 실시 할 수 있는 기능은 R이 막강한 통계 프로그램임을 알게 해줍니다. 다음은 자주 사용하는 분포함수와 접두어가 결합한 예입니다.

| 분포함수 | R에서의 분포함수(전달인자는 입력값만) | | | |
|---|---|---|---|---|
| | 밀도<br>P(X=x) | 누적확률<br>P(X<x) | 분위<br>P=P(X<?) | 난수 |
| 균등분포 | dunif(x) | punif(x) | qunif(p) | runif(n) |
| 정규분포 | dnorm(x) | pnorm(x) | qnorm(p) | rnorm(n) |
| 이항분포 | dbinom(x) | pbinom(x) | qbinom(p) | rbinom(n) |
| 카이제곱분포 | dchisq(x) | pchisq(x) | qchisq(p) | rchisq(n) |

자주 사용하는 분포함수의 전달인자

균등분포

- dunif(x, min = 0, max = 1, log = FALSE)
- punif(q, min = 0, max = 1, lower.tail = TRUE, log.p = FALSE)
- qunif(p, min = 0, max = 1, lower.tail = TRUE, log.p = FALSE)
- runif(n, min = 0, max = 1)

| 전달인자 | 설명 |
|---|---|
| x, q | 구하고자 하는 확률변수의 값 벡터 |
| p | 구하고자 하는 확률값 벡터 |
| n | 추출하려는 개수 |
| min, max | 균등분포의 하한과 상한 |
| log, log.p | TRUE이면 log(p)로 확률이 주어진다. |
| lower.tail | TRUE이면 P[X ≤ x], 아니면 P[X > x]. |

정규분포

- dnorm(x, mean = 0, sd = 1, log = FALSE)
- pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
- qnorm(p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
- rnorm(n, mean = 0, sd = 1)

| 전달인자 | 설명 |
|---|---|
| mean | 정규분포의 평균 (기본값 0) |
| sd | 정규분포의 표준편차 (기본값 1) |

이항분포

- dbinom(x, size, prob, log = FALSE)
- pbinom(q, size, prob, lower.tail = TRUE, log.p = FALSE)
- qbinom(p, size, prob, lower.tail = TRUE, log.p = FALSE)
- rbinom(n, size, prob)

| 전달인자 | 설명 |
|---|---|
| size | 전체 시행횟수 |
| prob | 성공의 확률 |

카이제곱분포

- dchisq(x, df, ncp = 0, log = FALSE)
- pchisq(q, df, ncp = 0, lower.tail = TRUE, log.p = FALSE)
- qchisq(p, df, ncp = 0, lower.tail = TRUE, log.p = FALSE)
- rchisq(n, df, ncp = 0)

| 전달인자 | 설명 |
|---|---|
| df | 자유도 |
| ncp | 양(+)의 비중심 모수 |

사용예

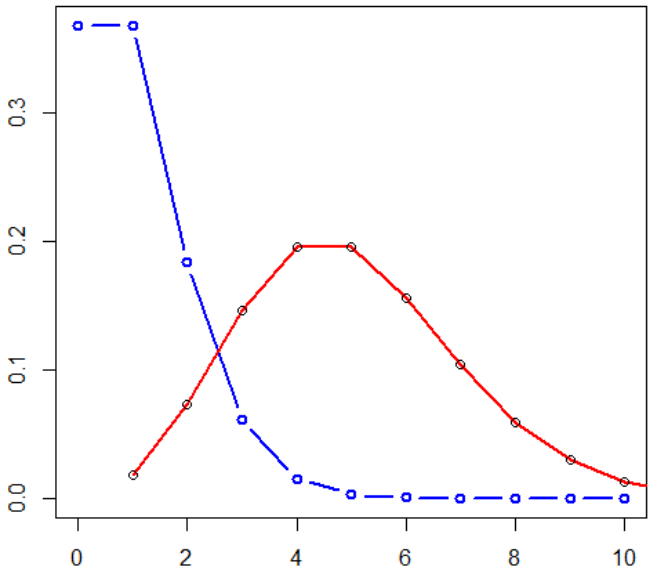
```
> dnorm(1.645, mean=0, sd=1)
[1] 0.1031108
→ 평균이 0이고 표준편차가 1인 정규분포에서 x가 1.645일때의 확률값

> pnorm(1.645, mean=0, sd=1)
[1] 0.9500151
→ 평균이 0이고 표준편차가 1인 정규분포에서 x가 1.645가 이하일 확률

> qnorm(0.025, mean=0, sd=1)
[1] -1.959964
→ 평균이 0이고 표준편차가 1인 정규분포에서 P[X<x]가 0.025가 되는 x 값

> rnorm(10)
 [1]  0.590 -1.806 -1.601 -0.594 -1.051 -1.371 -0.280 -0.453
 [9]  2.491 -1.282
> rnorm(10)
 [1]  1.509  0.069 -0.331 -0.103 -0.447 -1.711  1.115 0.103
 [9]  1.145 -0.219
→ 평균이 0이고 표준편차가 1인 정규분포로부터 10개의 난수 생성
```

심화 예제 : 중심극한정리에 대한 시뮬레이션

통계에서 중심극한 정리는 모집단의 분포에 상관없이 평균 μ, 표준편차 σ가 정해져 있을 경우 표본의 수가 크면 표본 평균의 분포가 근사적으로 평균이 μ이고 표준편차가 $\frac{\sigma}{\sqrt{n}}$인 정규분포를 따르는 것을 의미하는 것으로 중요한 정리입니다. 본 예에서는 한 쪽으로 포아송분포 치우쳐진 포아송 분포로부터 추출한 표본 평균이 표본의 개수가 증가함에 따라 정규분포를 따르는 것을 R을 이용하여 시각적으로 나타내보려 합니다.

먼저 포아송 분포의 모양을 결정하는 모수 람다가 1일때와 4일때를 그려보겠습니다.
다음 그림에서 파란색이 람다가 1일때의 포아송 분포이고 붉은색이 4일때의 포아송 분포입니다. 포아송분포에서 람다가 10보다 작을 때 작은 쪽으로 많이 분포합니다(10보다 클 경우는 어찌될지 아래의 코드를 변형해서 확인해보세요).

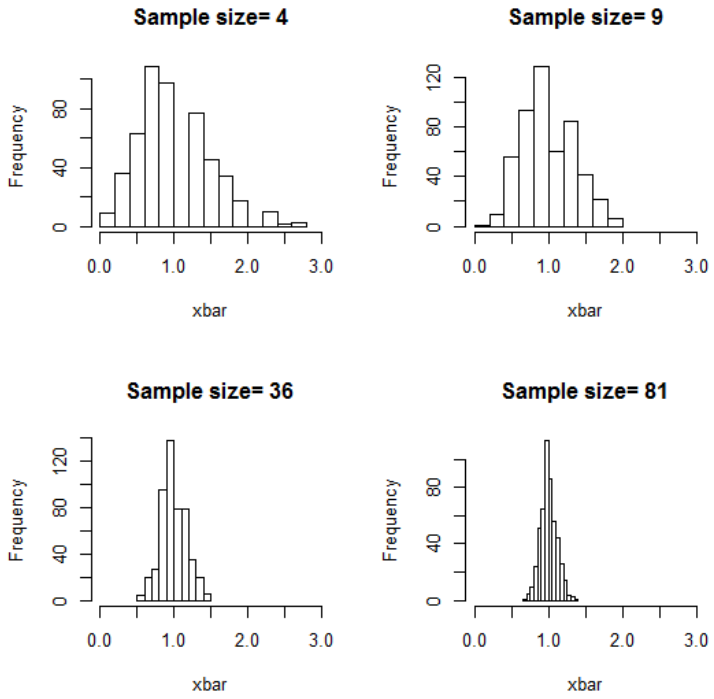
```
> x <- 0:10
> y <- dpois(x, lambda=1)
> y.4 <- dpois(x, lambda=4)
> plot(x, y, type="b", col="blue", lwd=2)
> lines(y.4, col="red", lwd=2)
> points(y.4)
```




다음은 이런 포아송 분포로부터 표본의 수가 4일 때, 9일 때, 36일 때, 그리고 81일 때에 대해 500번의 표본 추출을 하고 매 추출마다 평균을 구한다음 이 평균의 분포를 그린 결과입니다. 표본 추출수가 증가함에 따라 정규분포에 가까워지고 포아송 분포의 기대값인 람다주변으로 퍼진

정도가 줄어듦을 알 수 있습니다.[5]

```
> central.Poisson = function(lambda) {
+   nt = c(4, 9, 36, 81)
+   xbar = rep(0, 500)
+   windows()
+   par(mfrow=c(2,2))
+
+   for(i in 1:4) {
+     for(j in 1:500) {
+       xbar[j] = mean(rpois(nt[i], lambda))
+     }
+     hist(xbar, main=paste("Sample size=", nt[i]), xlim=c(0, lambda*3))
+   }
+   mtext("Central Limit Theorem : Poisson", side=3, outer=T,cex=1.5)
+ }
>
> central.Poisson(1)
```




[5] 본 시뮬레이션은 "기초통계학 – R을 이용한 통계분석", 노맹석 외 5인 저, 2011, 자유아카데미 간의 자료를 인용했습니다.

# 통계적 가설검정

평균비교 : t.test()

t분포를 이용하여 가설검정을 실시할 경우 사용하는 함수입니다. t분포를 이용한 가설검정은 단일 표본의 평균비교, 짝을 이룬 두 표본, 서로 독립인 두 표본에서의 평균비교 검정을 실시합니다.

```
t.test(x, y = NULL,
       alternative = c("two.sided", "less", "greater"),
       mu = 0,
       paired = FALSE,
       var.equal = FALSE,
       conf.level = 0.95, ...)
```

- X : 숫자로 구성된 벡터
- Y : 숫자로 구성된 벡터로 전달인자로 전달하지 않을 시 단일 표본 검정실시
- alternative : 기본값은 "two.sided"이며 기본값 상태에서는 양쪽 검정을 실시하고 대안가설 (Alternative Hypothesis, H1)에 따라 "less", "greater"를 지정한다.
- mu : 검정할 평균 값으로 입력하지 않을 시 기본값은 0
- paired : 짝을 이룬 두 표본의 경우 TRUE이며 기본값은 FALSE
- var.equal : x, y 두 집단의 분산이 동일할 경우 TRUE, 기본값은 FALSE
- conf.level : 신뢰구간을 구할 때 사용되는 신뢰수준으로 기본값은 0.95)

사용 예) 단일 표본의 평균 검정

mtcars 자료를 이용하여 1973 년부터 1974 년까지 미국에서 생산된 자동차들의 평균 연비는 갤런당 20 마일(20mpg)로 알려져 있다. 수동미션 차량들이 자동미션 차량보다 연비가 좋다는 것을 밝히기 위해 수동 미션 차량들의 연비는 20mpg 보다 크다고 할 수 있는지 유의수준 0.05 에서 검정하시오.

```
> with(mtcars, t.test(mpg[am==1], mu=20, alternative="greater"))

        One Sample t-test

data:  mpg[am == 1]
t = 2.57, df = 12, p-value = 0.01231
alternative hypothesis: true mean is greater than 20
95 percent confidence interval:
 21.3  Inf
sample estimates:
mean of x
     24.4
```

사용 예) 짝을 이룬 표본의 평균 검정

R 내장자료인 sleep 은 서로 다른 두 종류(grou)의 수면제로 각각의 약을 투여하고 수면시간의 증감(extra)을 조사하였다. 서로 다른 두 약물은 수면시간의 증감에 차이가 있는지 유의수준 0.05 에서 검정하시오.

```
> with(sleep,
+ t.test(extra[group == 1], extra[group == 2],
+ paired = TRUE)
+ )

       Paired t-test

data:  extra[group == 1] and extra[group == 2]
t = -4.06, df = 9, p-value = 0.002833
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -2.46 -0.70
sample estimates:
mean of the differences
                  -1.58
```

사용 예) 독립인 두 표본의 평균 검정

R 내장자료인 InsectSpray 는 서로 다른 여섯종류(spray)의 살충제를 뿌려 죽은 벌레의 수(count)를 관찰한 데이터이다. 이중 A 와 B 살충제의 성능을 비교하려고 한다. A 와 B 두 살충제는 단위 면적당 죽인 벌레 수에 차이가 있는지 유의수준 0.05 에서 검정하시오.(A 와 B 의 단위 면적당 죽인 벌레 수의 분산은 서로 동일하다고 가정한다.)

```
> with(InsectSprays,
+  t.test(count[spray=="A"], count[spray=="B"], var.equal=T)
+  )

        Two Sample t-test

data:  count[spray == "A"] and count[spray == "B"]
t = -0.454, df = 22, p-value = 0.6546
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -4.64  2.98
sample estimates:
mean of x mean of y
     14.5      15.3
```

상관계수와 상관분석 : cor(), cor.test()

두 변수 사이의 상관계수를 구하는 함수는 cor()이고 상관계수의 유의성을 검사하는 R 함수는 cor.test() 입니다. 이 두 변수를 이용하여 상관계수를 구하고 검사해 봅시다.

```
cor(x, y = NULL,
        use = "everything",
        method = c("pearson", "kendall", "spearman"))
```

- x, y : 상관계수를 구할 두 변수
- use : 결측치에 대한 처리 방법의 문자열
- method : 상관계수의 종류로 "pearson"이 기본값. (kendall과 spearman은 순위 기반의 상관계수)

```
cor.test(x, y,
         alternative = c("two.sided", "less", "greater"),
         method = c("pearson", "kendall", "spearman"),
         exact = NULL, conf.level = 0.95, continuity = FALSE, ...)
```

- x, y : 상관계수를 구할 두 변수
- alternative : 대안가설에 따른 문자열
- method : 분석할 상관계수의 종류
- exact : kendall과 spearman 상관계수에서의 정확한 p-value 계산여부
- continuity : kendall과 spearman 상관계수에서의 연속성 수정 여부

사용 예)

R 내장자료인 cars 는 자동차의 속도(speed)별 제동거리(dist)를 측정한 자료이다. 두 값의 상관계수를 구하고 유의수준 0.05 에서 유의한 상관관계를 갖는지 검정하시오.

```
> with(cars, plot(dist, speed))
> with(cars, cor(speed, dist))
[1] 0.807
> with(cars, cor.test(speed, dist))

        Pearson's product-moment correlation

data:  speed and dist
t = 9.46, df = 48, p-value = 1.49e-12
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.682 0.886
sample estimates:
  cor
```

회귀 분석

설명변수(독립변수)들의 결합을 통해 반응변수(종속변수)의 결합을 나타내는 회귀모형 중 선형회귀모형은 다음과 같습니다.

$$\mathrm{Y} = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \varepsilon$$

- Y : 반응변수
- $X_1, X_2, X_3$ : 설명변수
- $\beta_0, \beta_1, \beta_2, \beta_3$ : 회귀계수
- $\varepsilon$ : 오차항 (평균이 0이고 표준편차가 $\sigma$인 정규분포를 따름)

위의 모형으로부터 표본을 추출하고 이를 바탕을 회귀모형을 구축하면 다음과 같이 나타낼 수 있으며

$$y_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \beta_3 x_{3i} + \varepsilon_i, \qquad i = 1, \ldots, n$$

- $y_i,\ x_i$ : 각 변수의 i번째 관찰값
- $\varepsilon_i$ : i번째 오차항

최종적으로 회귀계수를 추정하여 반응변수를 예측한 회귀모형은 다음과 같습니다.

$$\hat{y_i} = b_0 + b_1 x_{1i} + b_2 x_{2i} + b_3 x_{3i}, \qquad i = 1, \ldots, n$$

- $\hat{y_i}$ : 예측값
- $b_0, b_1, b_2, b_3$: (최소제곱법을 통해 구한) 추정된 회귀계수
- 잔차 : $e_i = y_i - \hat{y_i}$

회귀분석의 가장 간단한 예인 단순선형회귀분석을 통해 R에서 어떻게 회귀분석을 실시하는지 알아보도록 하겠습니다.

R에서의 모형

R에서의 모형을 나타내는 표현식은 "반응변수 ~ 종속변수의 결합" 입니다. 즉, 위의 식의 경우 "y ~ x1 + x2 + x3" 로 나타냅니다. (선형회귀분석에서 원점을 지나는 회귀의 경우 "y ~ x1 – 1"과 같이 모형을 구축합니다.)

사용 예)

R의 내장자료인 cars는 자동차의 속도(speed)에 따른 제동거리(dist)를 담고 있다. 이 자료를 통해 속도에 따른 제동거리의 회귀모형을 구축하시오.

```
> with(cars, plot(dist, speed))
> out <- with(cars, lm(dist ~ speed))
> out
Call:
lm(formula = dist ~ speed)
Coefficients:
(Intercept)       speed
   -17.579       3.932
```

➔ Step 1 : 회귀계수 추정

```
> summary( aov(out) )
            Df Sum Sq Mean Sq F value   Pr(>F)
speed        1  21185   21185   89.57 1.49e-12 ***
Residuals   48  11354     237
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.11
```

➔ Step 2 : 분산분석표를 통한 모형의 유의성 검정

```
> abline(out, col="red")
```

➔ 회귀직선 출력

```
> summary( out )
Call:
lm(formula = dist ~ speed)
Residuals:
    Min      1Q  Median      3Q     Max
-29.069  -9.525  -2.272   9.215  43.201
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -17.5791     6.7584  -2.601   0.0123 *
speed         3.9324     0.4155   9.464 1.49e-12 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Residual standard error: 15.38 on 48 degrees of freedom
Multiple R-squared:  0.6511,    Adjusted R-squared:  0.6438
F-statistic: 89.57 on 1 and 48 DF,  p-value: 1.49e-12
```

➔ Step 3 : 설명력과 회귀계수의 추정

```
> par(mfrow=c(2,2))
> plot(out)
```

➔ Step 4 : 회귀진단

R에서의 테이블과 범주형 자료의 분석

table()과 prop.table()

table() 함수는 주어진 자료로부터 요인(Factor)별로 자료의 수를 세어 table 형의 자료를 반환해 줍니다. 또한 prop.table() 은 table 자료로부터 비율을 구해 반환해 줍니다. 다음의 사용예를 통해 알아보겠습니다.

```
> rpois(20, 5)
 [1] 6 7 9 6 6 9 3 5 3 2 8 5 5 1 2 6 5 4 6 2
```

➔ 람다가 5인 포아송분포로부터 20개의 난수 생성

```
> table(rpois(100, 5))
 1  2  3  4  5  6  7  8  9 10 11
 3  5 17 18 14 14 13  4  6  2  4
```

➔ 람다가 5인 포아송분포로부터 100개의 난수를 생성하고 이를 table 형태로 요약

```
> str(warpbreaks)
'data.frame':   54 obs. of  3 variables:
 $ breaks : num  26 30 54 25 70 52 51 26 67 18 ...
 $ wool   : Factor w/ 2 levels "A","B": 1 1 1 1 1 1 1 1 1 1 ...
 $ tension: Factor w/ 3 levels "L","M","H": 1 1 1 1 1 1 1 1 1 2 ...
```

➔ warpbreaks는 wool의 종류와 장력(L, M, H)에 따른 breaks의 수를 저장한 자료

```
> head(warpbreaks)
  breaks wool tension
1     26    A       L
2     30    A       L
3     54    A       L
4     25    A       L
5     70    A       L
6     52    A       L
> with(warpbreaks, table(wool, tension))
    tension
wool L M H
   A 9 9 9
   B 9 9 9
```

➔ wool 과 tension 별로 몇 개의 자료가 관찰되었는지 표로 출력

```
> prop.table( table(rpois(100, 5)) )
   0    1    2    3    4    5    6    7    8    9   10   13
0.01 0.04 0.11 0.14 0.15 0.16 0.13 0.17 0.04 0.02 0.02 0.01
```

➔ 비율 테이블을 구합니다.

```
> prop.table( with(warpbreaks, table(wool, tension)) )
    tension
wool         L         M         H
   A 0.1666667 0.1666667 0.1666667
   B 0.1666667 0.1666667 0.1666667
```

➔ 비율 테이블을 구합니다. 비율은 전체 중 각 셀의 크기 입니다.

xtabs() : 테이블 만들기 위한 최적의 함수

실제 데이터로부터 테이블을 구하는 것은 위의 table() 함수도 좋지만 xtabs() 함수가 더 유연하게 작동합니다. xtabs()는 각 셀의 수를 알고 있는 경우에도 혹은 데이터로부터 추출할 때 모두 사용할 수 있습니다. 사용시 "**xtabs( 도수 ~ 행구분변수 + 열구분변수)**" 의 형태로 사용되며 다음의 예를 통해 xtabs()의 활용법을 알아보겠습니다.

```
> (data1 <- data.frame( group=c("placebo", "placebo", "treatment",
+ " treatment "), outcome=c(1, 0, 1, 0), count=c(16, 48, 40, 20)))
    group outcome count
1 placebo       1    16
2 placebo       0    48
3 treatment     1    40
4 control       0    20
```

➔ 돗수를 담고 있는 데이터 프레임 data1을 만들었습니다.

```
> (tab1 <- xtabs(count ~ group + outcome, data=data1))
            outcome
  group        0  1
  treatment   20 40
  placebo     48 16
```

➔ 변수 count는 돗수를 담고 있으며 행 구분은 group으로 열 구분 outcome으로 합니다.

```
> str(ToothGrowth)
'data.frame':   60 obs. of  3 variables:
 $ len : num  4.2 11.5 7.3 5.8 6.4 10 11.2 11.2 5.2 7 ...
 $ supp: Factor w/ 2 levels "OJ","VC": 2 2 2 2 2 2 2 2 2 2 ...
 $ dose: num  0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 ...
> head(ToothGrowth)
   len supp dose
1  4.2   VC  0.5
2 11.5   VC  0.5
3  7.3   VC  0.5
4  5.8   VC  0.5
5  6.4   VC  0.5
6 10.0   VC  0.5
> xtabs(~supp+dose, data=ToothGrowth)
    dose
supp 0.5  1  2
  OJ  10 10 10
  VC  10 10 10
```

➔ 돗수 자리를 비우면 행 구분 변수와 열 구분 변수별 데이터를 세어서 출력합니다.

범주형 자료분석

평균을 계산할 수 없는 범주형에 대한 분석은 테이블 구조를 통해 할 수 있습니다.
테이블 내의 각 셀별로 관찰빈도와 이론적 확률 값에 맞는 기대빈도와의 차이를 이용하여 이론적 확률 값에 따라 발생되었는지 검정하는 것으로 이때 사용하는 분포는 카이제곱분포입니다. 카이제곱분포는 자유도에 따라 그 모양이 결정되며 0부터 무한대까지를 갖습니다.

**적합도 검정 : 이론적 확률값에 따른 표본의 동일 발생여부 검정**

멘델의 유전법칙에 의하면 4 가지 완두콩의 형질이 나타날 비율은 9:3:3:1 이라고 한다. 멘델의 유전법칙을 알아보기 위하여 완두콩을 재배한 결과 다음과 같은 수확을 얻었다(총 556 종)

| | 둥글고 노랑 (RY) | 둥글고 녹색 (Ry) | 주름지고 노랑 (rY) | 주름지고 녹색 (ry) |
|---|---|---|---|---|
| 개체수 | 315 | 101 | 108 | 32 |

위의 표로부터 유의수준 0.05 에서 멘델의 유전법칙과 동일한지 검정하시오.

```
> mendel <- c(315, 101, 108, 32)
> prob <- c(9, 3, 3, 1)/16
> chisq.test(mendel, p=prob)

        Chi-squared test for given probabilities

data:  mendel
X-squared = 0.47, df = 3, p-value = 0.9254
```

**독립성 검정 : 두 변수간의 연관 여부**

독립성 검정에 사용되는 자료는 "(r x c)분할표" 입니다. 분할표는 table 구조이거나 행렬 등 행과 열로 나타낼 수 있으면 됩니다. 독립성 검정 역시 카이제곱 통계량을 사용하며 이때의 자유도는 "행의 개수 - 1 (r-1) x 열의 개수 (c-1)"로 정해집니다.

앞서 xtabs()에서 사용한 자료는 비타민 투여 여부(treatment : 비타민 투어)에 따라 감기 환자의 발생(outcome = 1)을 나타낸 표이다. 비타민 투여여부가 감기 발생과 연관이 있는지 유의수준 0.05 에서 검정하여라.

```
> data1 <- data.frame( group=c("placebo", "placebo", "treatment",
+ "treatment"), outcome=c(1, 0, 1, 0), count=c(16, 48, 40, 20))
> tab1 <- xtabs(count ~ group + outcome, data=data1)
> chisq.test(tab1)

        Pearson's Chi-squared test with Yates' continuity correction

data:  tab1
X-squared = 20.0589, df = 1, p-value = 7.509e-06
```

**동일성 검정 : 행의 하위 모집단 간 열 분포가 동일한지 검정**

동일성 검정 또한 독립성 검정과 마찬가지로 "(r x c)분할표"를 사용하고 카이제곱 통계량을 사용합니다. 독립성 검정과 검정 방법 또한 동일하지만 검정을 바라보는 관점이 다릅니다. 독립성은 두 변수간의 연관이라면 동일성 검정은 주로 행에 위치하는 하위 집단 별로 열 범주별 분포가 서로 같은지를 봅니다. 설문조사 등에서 조사 집단간 동일성 여부를 판단하는 데 많이 사용됩니다.

내장 자료 Titanic 은 타이타닉 호의 객실(Class), 성별(Sex), 연령대(Age), 생존여부(Survived)에 대한 자료이다. 이 자료로부터 생존여부에 따라 객실별 분포가 동일한지 유의수준 0.05 에서 검정하라.

```
> str(Titanic)
 table [1:4, 1:2, 1:2, 1:2] 0 0 35 0 0 0 17 0 118 154 ...
 - attr(*, "dimnames")=List of 4
  ..$ Class   : chr [1:4] "1st" "2nd" "3rd" "Crew"
  ..$ Sex     : chr [1:2] "Male" "Female"
  ..$ Age     : chr [1:2] "Child" "Adult"
  ..$ Survived: chr [1:2] "No" "Yes"
> apply(Titanic, c(1, 4), sum)
     Survived
Class   No Yes
  1st  122 203
  2nd  167 118
  3rd  528 178
  Crew 673 212
```

➔ Class와 Survived 별로 자료를 합함

```
> t(apply(Titanic, c(1, 4), sum))
       Class
Survived 1st 2nd 3rd Crew
    No  122 167 528  673
    Yes 203 118 178  212
```

➔ Survived 값에 따른 하위 집단으로 하기 위해 전치(Transpose) 시킴

```
> data2 <- t(apply(Titanic, c(1, 4), sum))
> chisq.test(data2)

       Pearson's Chi-squared test

data:  data2
X-squared = 190.4011, df = 3, p-value < 2.2e-16
```