R 언어 사용법 기초

제 1 장 R 소개

분석 환경의 이해

가. 통계 패키지 R

▶ R은 통계 분석 / 그래픽을 제공하는 공개 소프트웨어로 강력하고 유연한 프로그래밍 언어

▶ 역량

- Sophisticated statistical analyses
- Predictive analytics
- Data visualization

▶ 적용

- Real-time trading
- Finance
- Risk assessment
- Forecasting
- Bio-technology
- Drug development
- Social networks

분석 환경의 이해

나. R의 역사

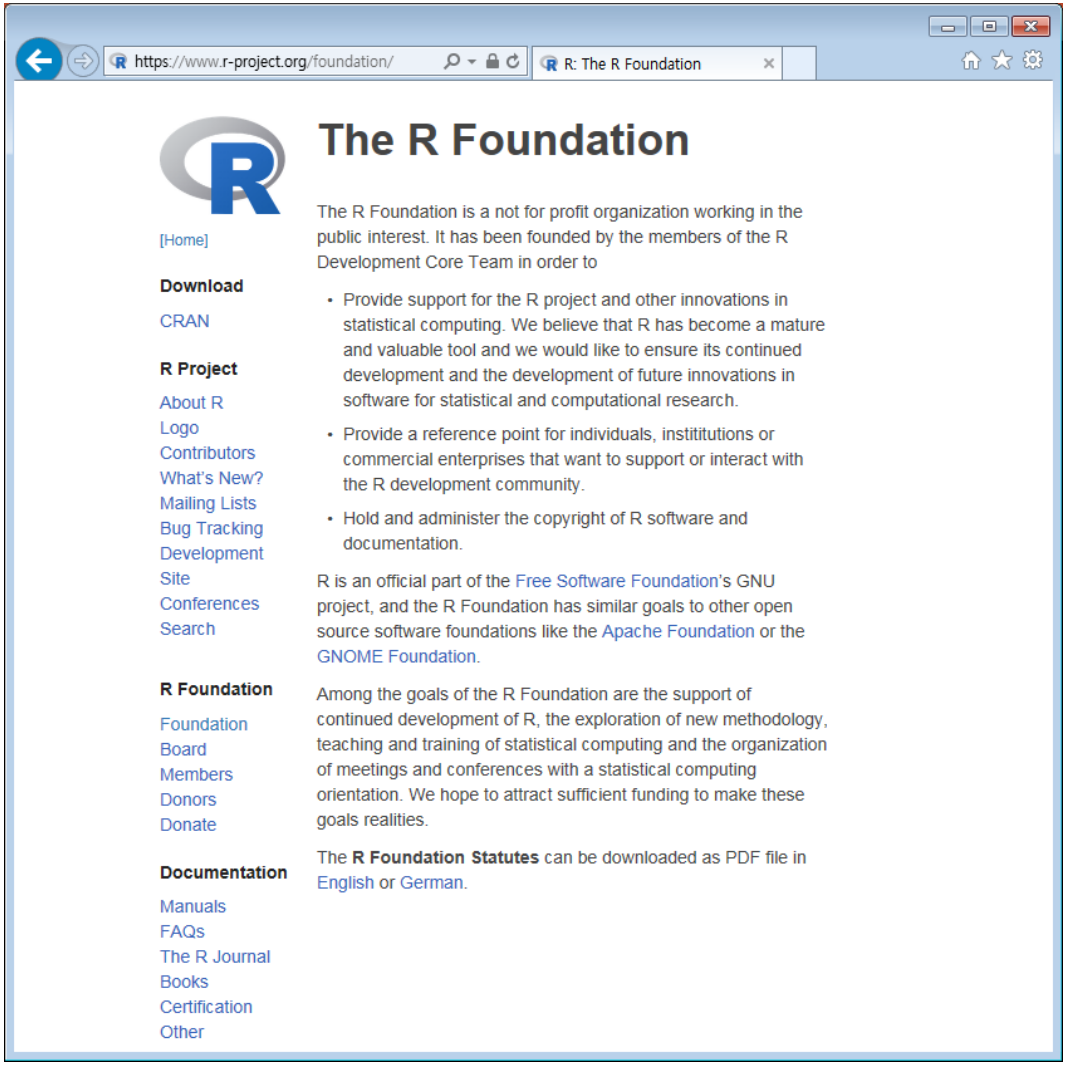
- 2000년 후반부터 Google, Facebook, Amazon, Yahoo 등에서 대형 데이터 분석에 활용되면서 폭발적으로 도입이 늘어난 오픈소스 기반의 분석 패키지로써 Functional, Objected Oriented 개념을 갖고 있다.
- 1975~1976년 Bell Lab에서 개발된 S language(Statistical Programming Language)를 참고하여 누구나 사용할 수 있도록 만든 것이 현재의 R 이다.
- 1993년 뉴질랜드 오클랜드 대학(University of Auckland)의 Robert Gentleman과 Ross Ihaka에 의하여 개발된 언어이다.

분석 환경의 이해

나. R의 역사

The R Foundation

- R Development Core Team 멤버들에 의하여 설립된 비영리 단체이다.
- R의 배포와 수정은 R Development Core Team과 많은 기여자들에 의하여 이루어지고 있다



분석 환경의 이해

다. R의 특징

R의 가장 강력한 특징

1) 그래픽 처리

- 사용 소프트웨어에 버금가는 상당한 수준의 그래프와 그림을 그릴 수 있다.
- Spotfire, Qlikview 등 다양한 Visualization Tool에서 직접 연동 가능하다.

2) 데이터 처리 및 계산 능력

- R은 Base system 과 Package로 구성되어 있으며, 메모리 기반으로 동작한다.
  (데이터 처리 작업이 빠르며, 하드웨어의 메모리 크기가 처리 시간에 영향을 미침)
- 모든 플랫폼(Windows, MacOS, Linux)에서 운영이 가능 ('R is Free')
- Object 기반으로 데이터, 함수가 관리되어진다.
- Java, C# 등과의 연동을 위한 API를 제공하여, 웹 시스템 환경에서 분석엔진으로 활용이 가능하다.
- SAS, SPSS 등 다른 통계분석 소프트웨어에서 Plug In 형태 등으로 R의 Script를 이용 가능하다.
- 사용자 함수를 만들어 처리 로직을 자동화하기 편하다.

분석 환경의 이해

다. R의 특징

R의 가장 강력한 특징

3) 패키지

- 새로운 알고리즘이 적용된 Package가 CRAN(Comprehensive R Archive Network)을 통하여 사용자에게 빠르게 공유된다. (상용 소프트웨어의 버전 업데이트 속도보다 새로운 알고리즘 적용이 빠름)
- Package를 만들어 다른 R 사용자와 공유 가능하다. (동일 업무를 수행하는 조직 내에서 재사용 성이 가능)
- 다양한 함수 및 데이터가 내장되어 있어, Help의 Examples를 바로 사용할 수 있다.

4) 사용자 커뮤니티

- 사용자 커뮤니티가 활성화되어 있어 다양한 정보를 공유할 수 있다.
- http://www.r-bloggers.com/
- http://www.r-statistics.com/
- http://www.r-tutor.com/
- http://www.statmethods.net/

제 2 장 R 설치

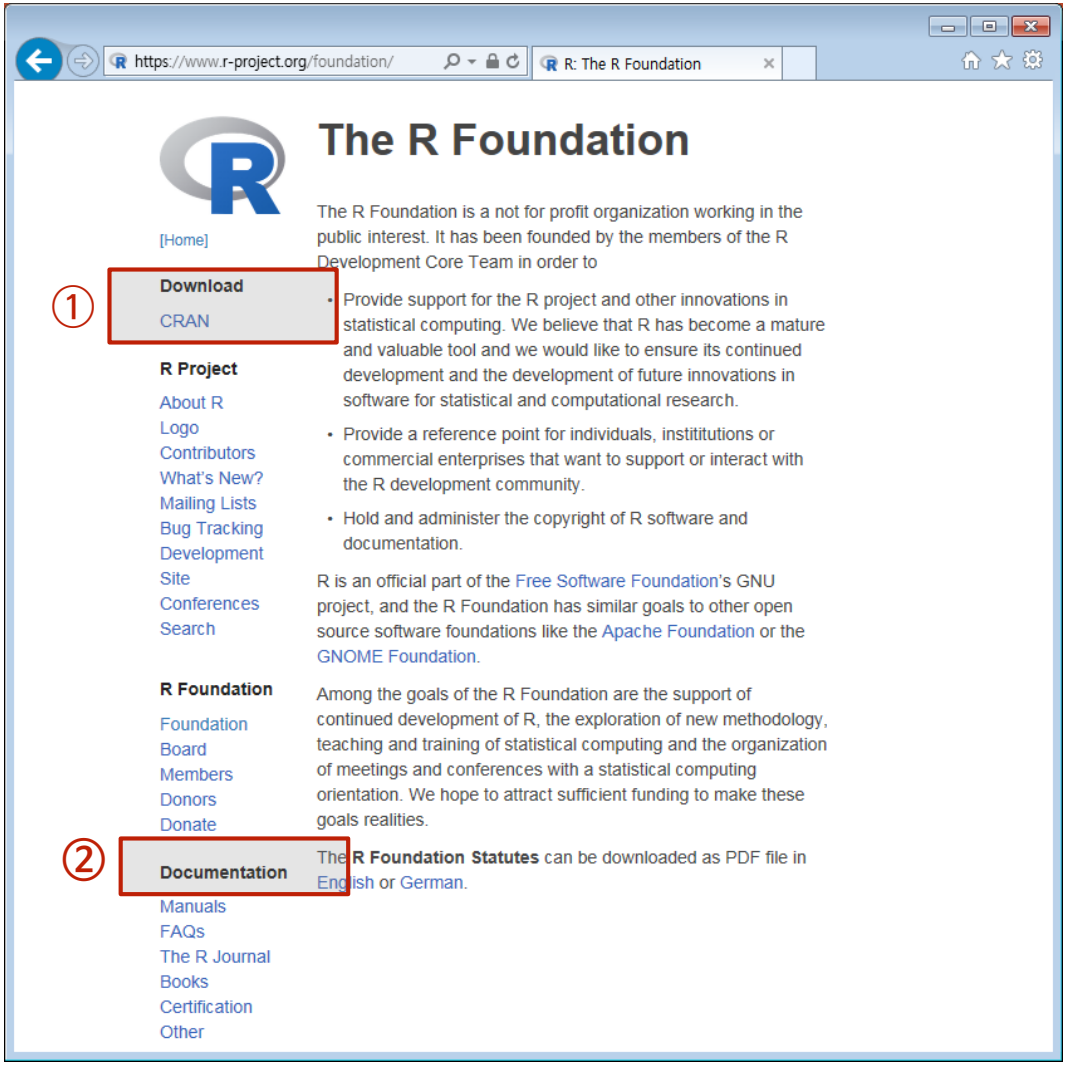
R 설치 방법

R 다운로드

- 웹 브라우저를 이용하여 접속 (http://www.r-project.org)하여 어디서든지 R을 다운로드 할 수 있다.

R download & R packages download

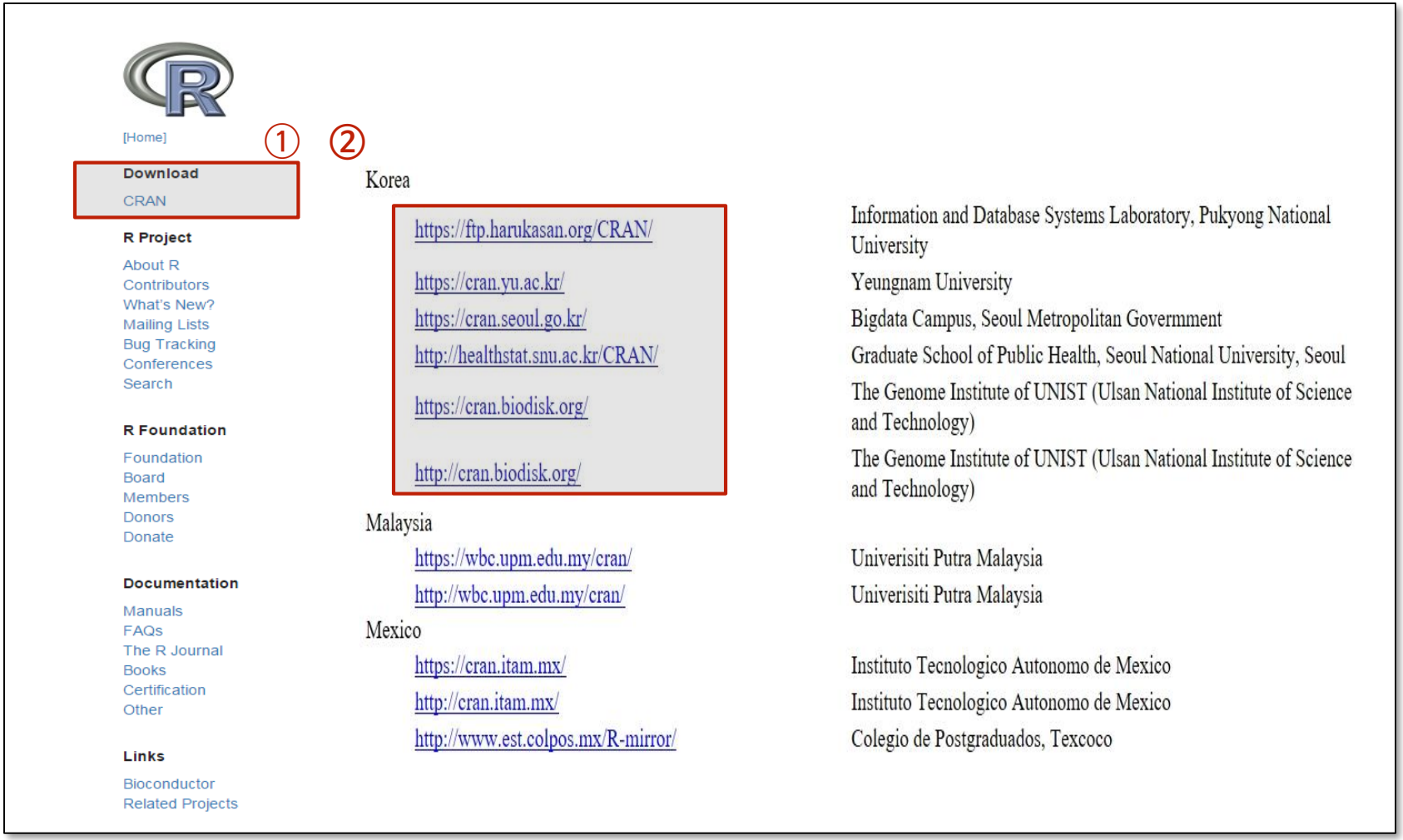
R Manuals



R 설치 방법

R 다운로드

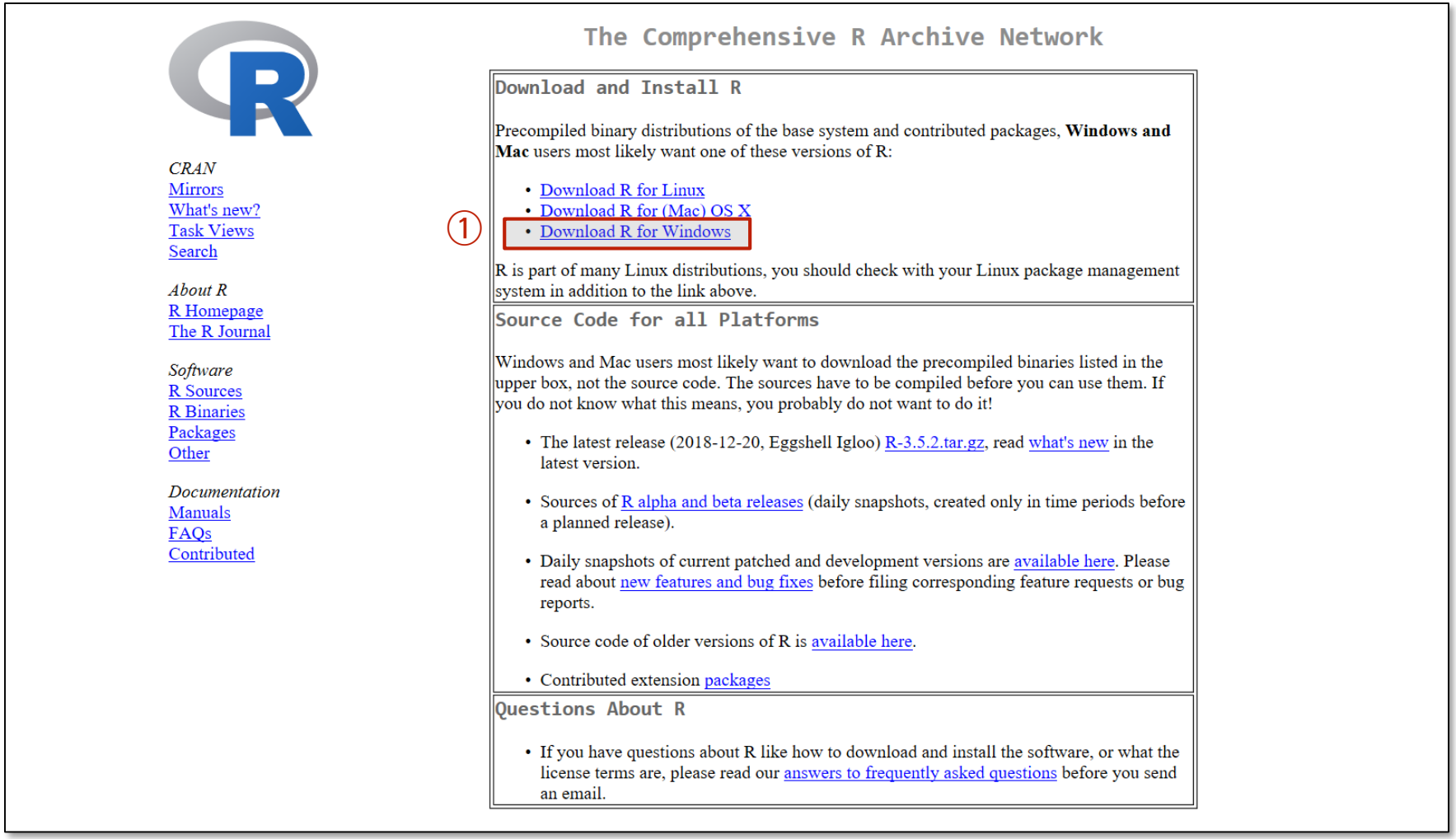
- CRAN Mirror Site 목록 중 본인과 가까운 나라의 사이트를 고른다. (Korea → http://cran.nexr.com/)



R 설치 방법

R 다운로드

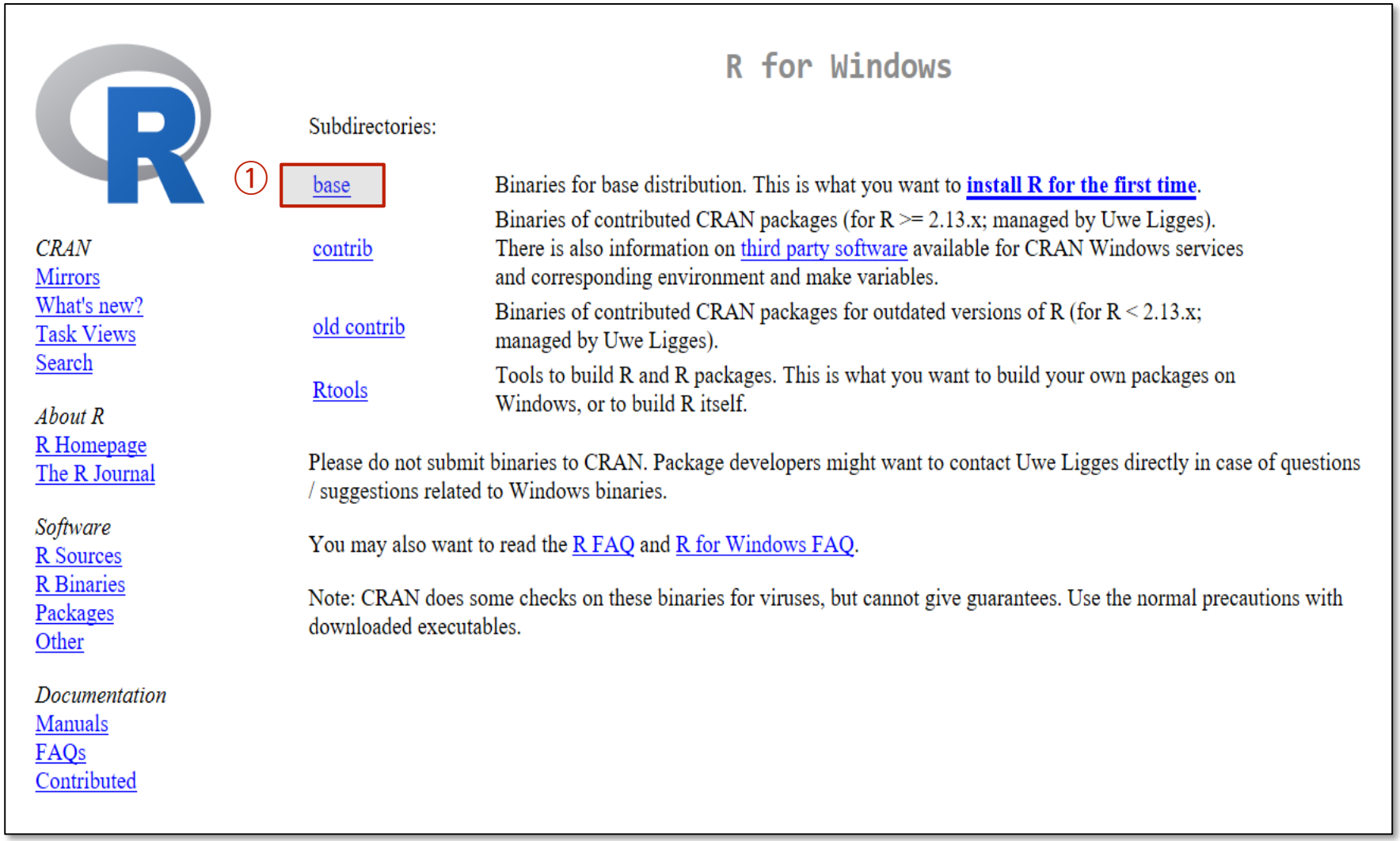
- **설치 H/W의 OS에 해당하는 R Software 를 선택 (Windows인 경우 'Download R for Windows' 선택)**



R 설치 방법

R 다운로드

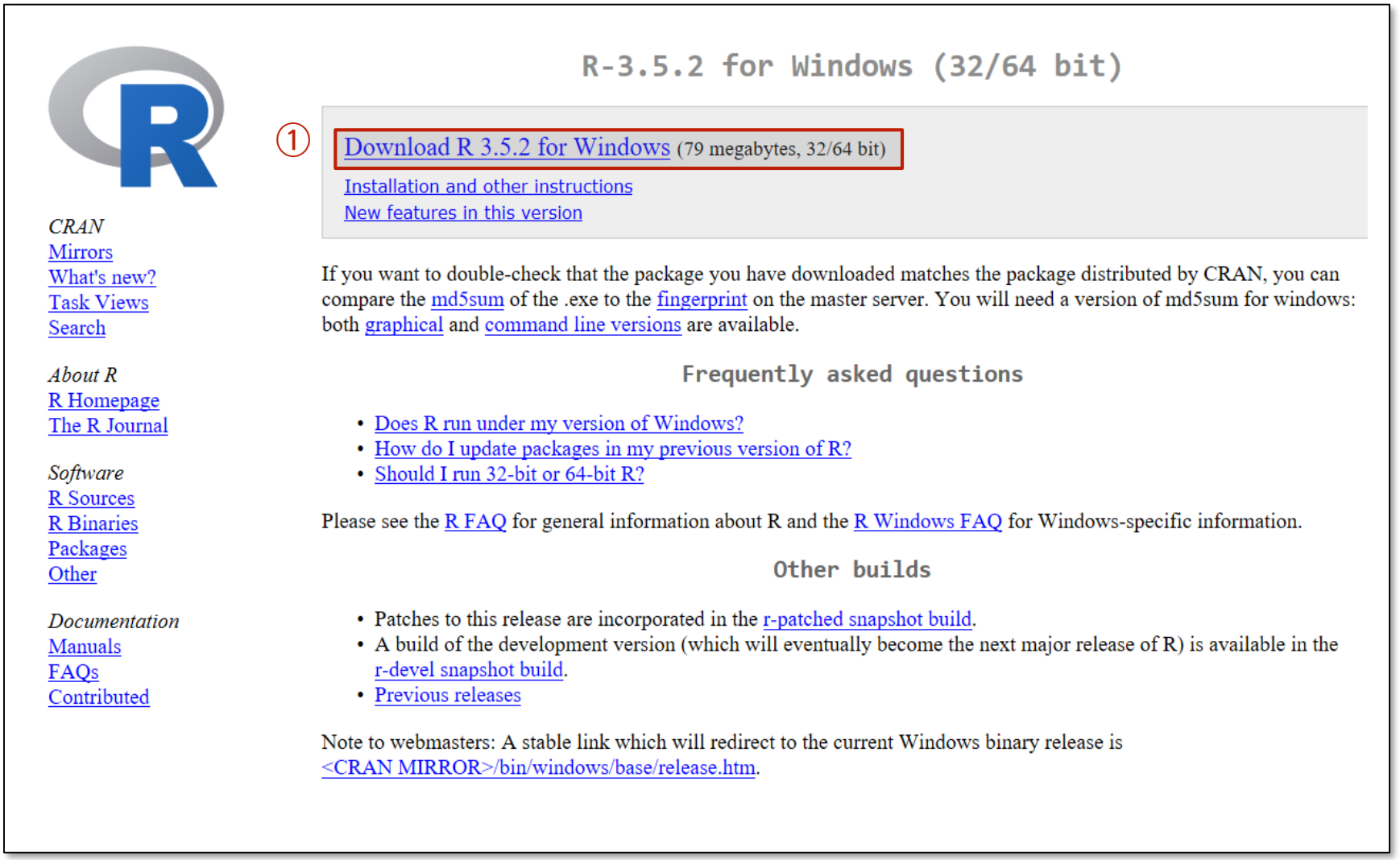
- **Base R Software를 선택한다.**



R 설치 방법

R 다운로드

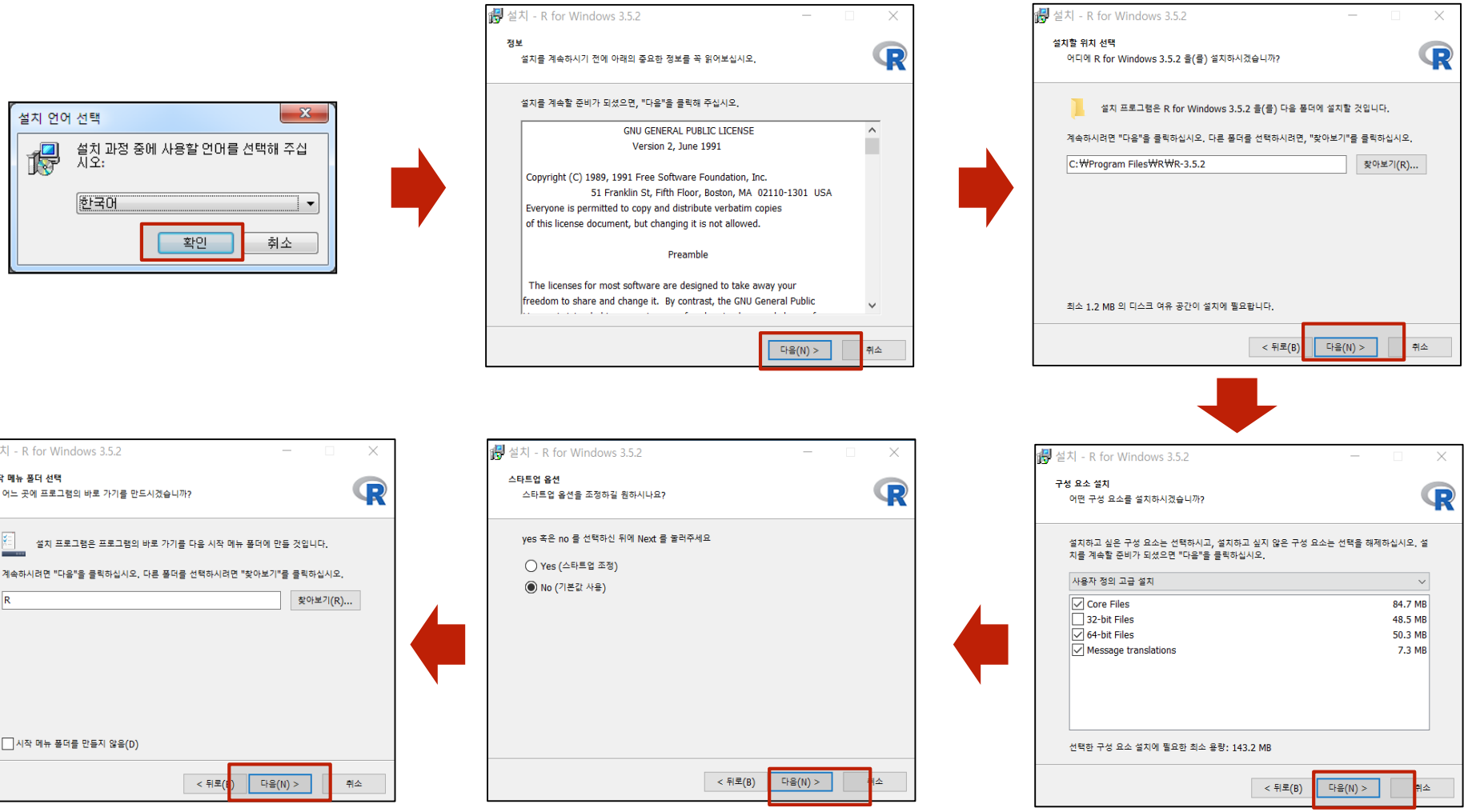
- **최신 Version의 R을 설치하기 위해서는 Download R 3.5.2 for Windows를 선택한다.**



R 설치 방법

R 설치 순서 (1/2)

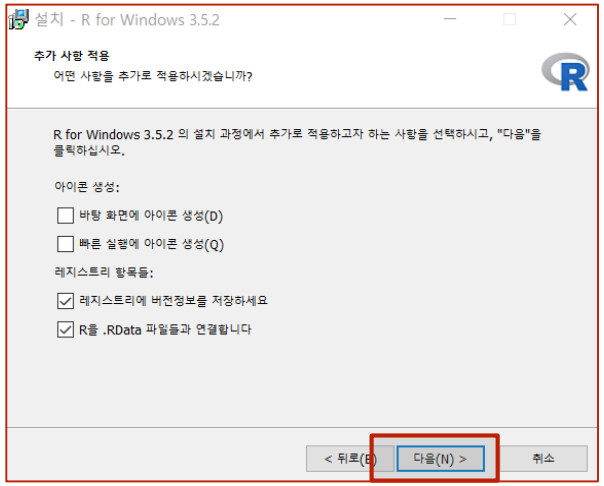
- **다운로드가 끝나면 설치파일을 더블 클릭한다. 아래의 그림 순서대로 진행하면 무리 없이 설치가 된다.**

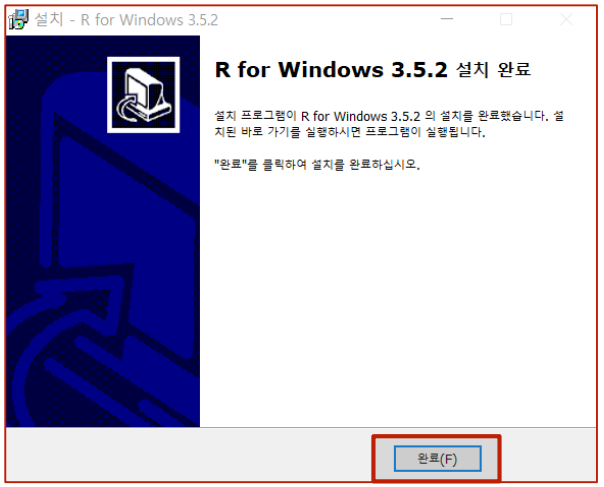


R 설치 방법

R 설치 순서 (2/2)

- **다운로드가 끝나면 설치파일을 더블 클릭한다. 아래의 그림 순서대로 진행하면 무리 없이 설치가 된다.**



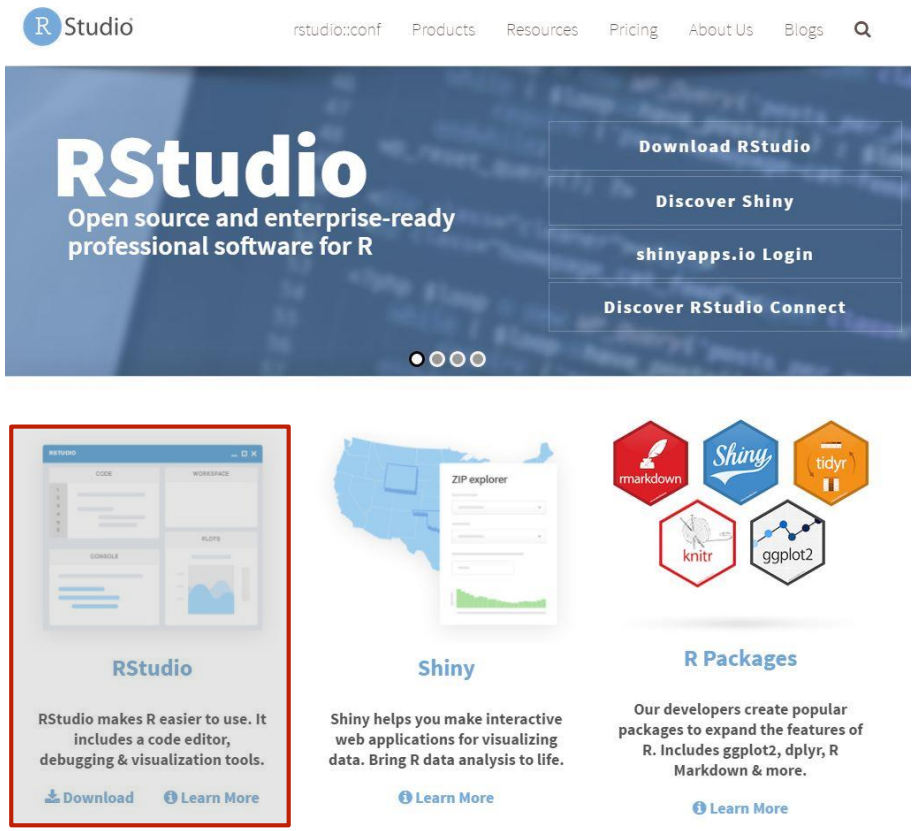


R Studio 설치 방법

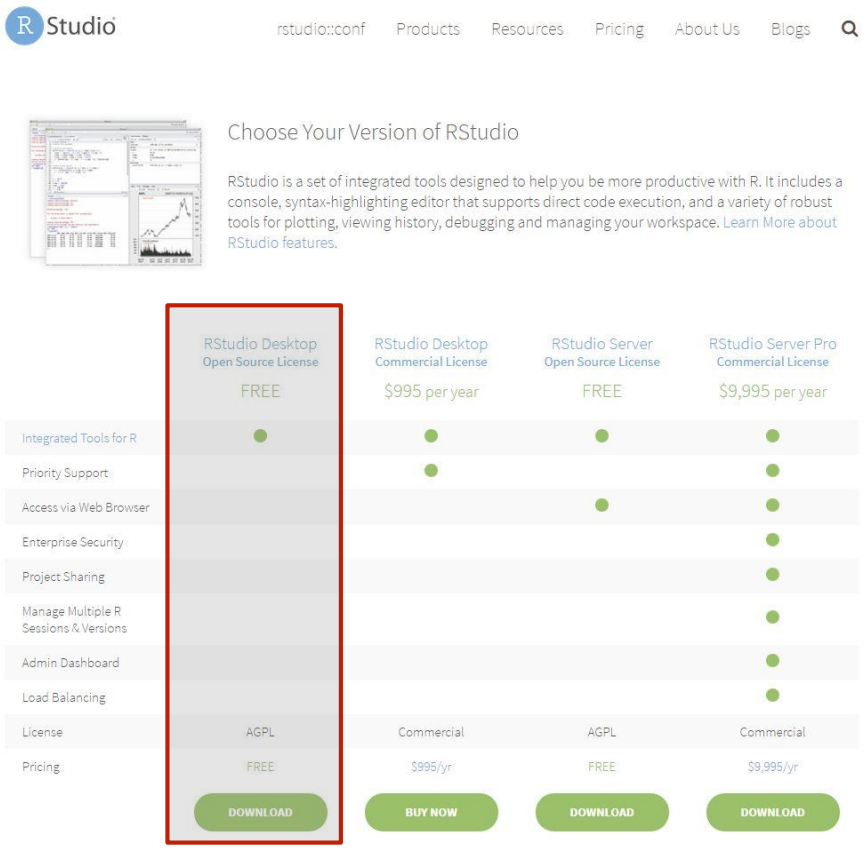
R Studio 다운로드

- R Studio는 R을 사용하기 쉽게 돕는 통합 개발 환경(Integrated Development Environment IDE) 이다.
- 웹 브라우저를 이용하여 접속 (http://www.rstudio.com)하여 R Studio를 다운로드 할 수 있다.

[R Studio Home Page]



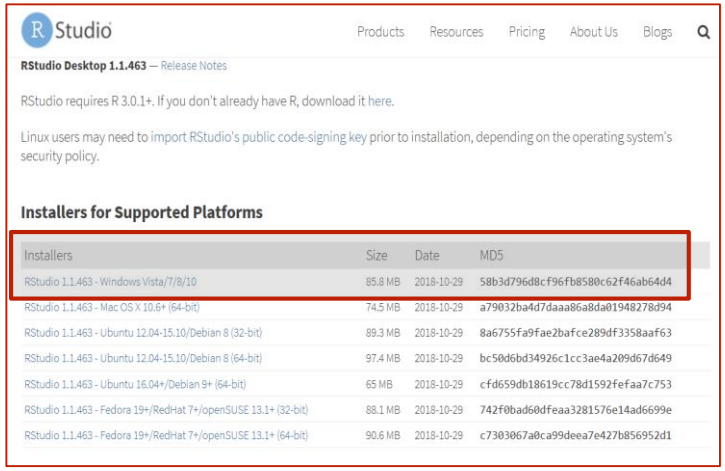
[R Studio Download]

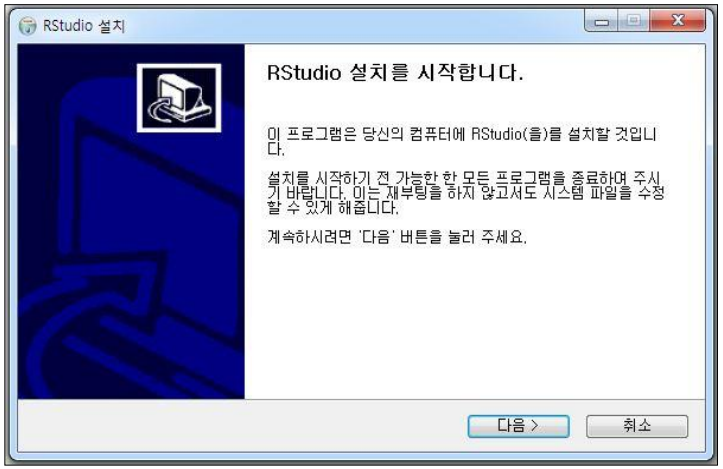


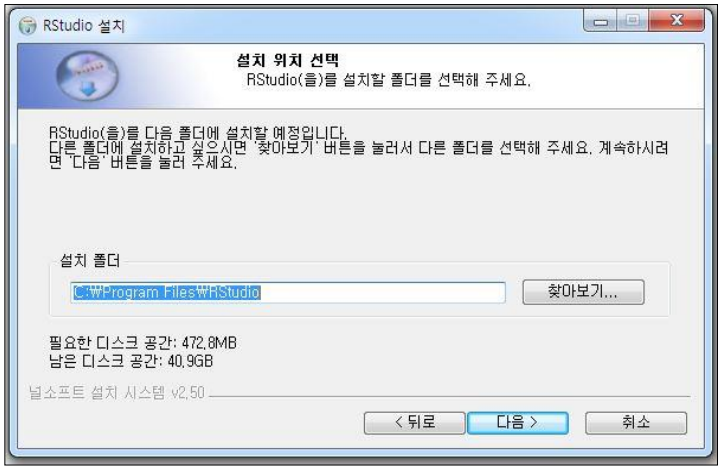
R Studio 설치 방법

R Studio 설치 순서

- **R Studio 설치는 어렵지 않다. 아래의 그림 순서대로 진행하면 설치가 된다.**



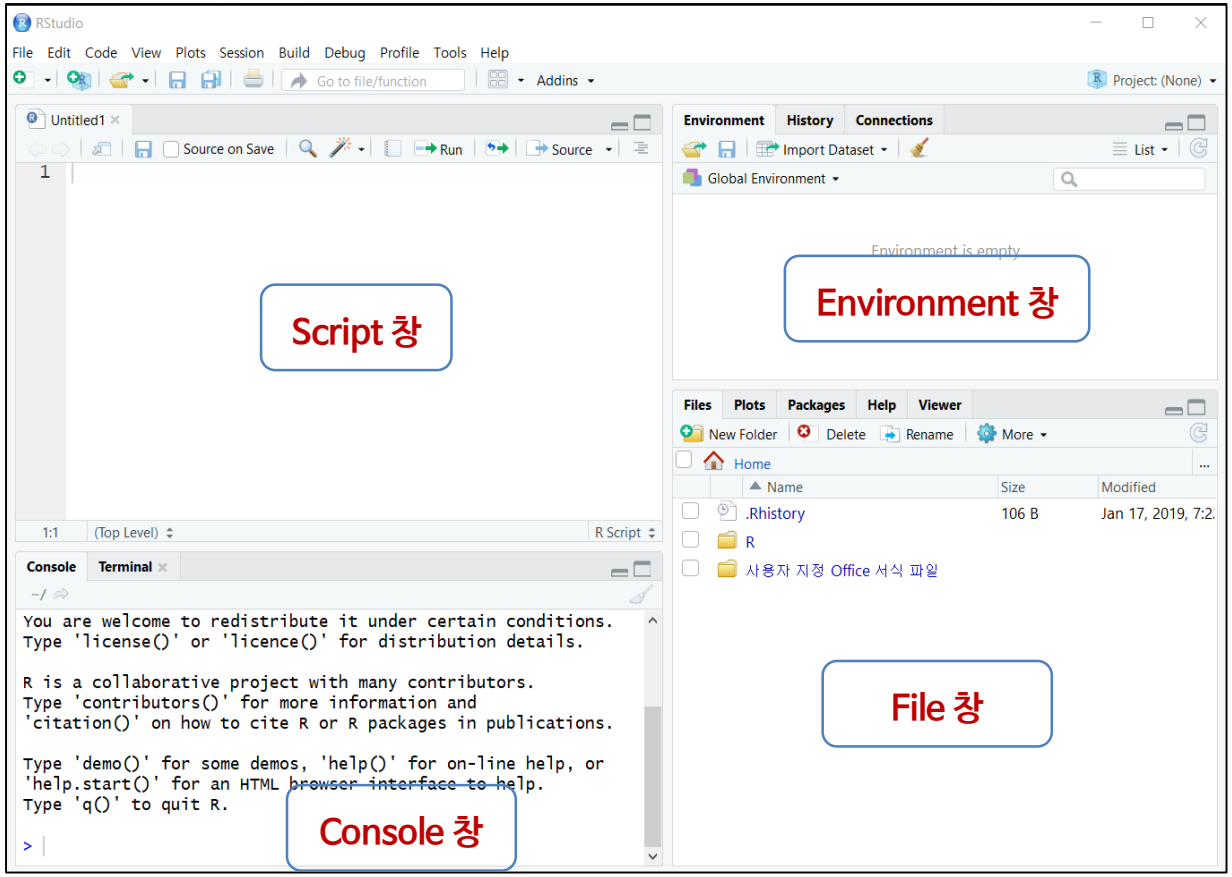




R Studio 인터페이스

R Studio 인터페이스

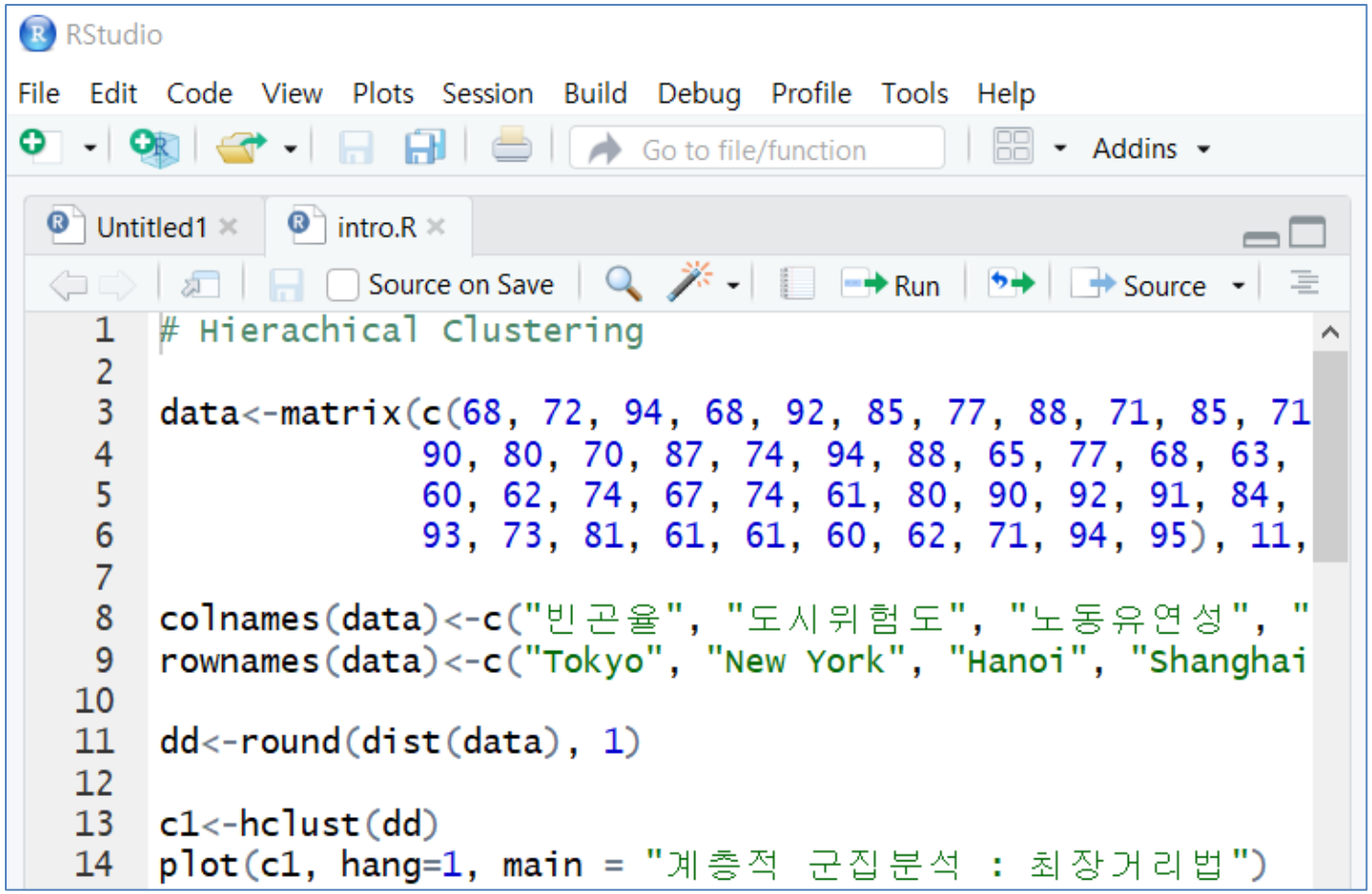
- R Studio를 실행하면 왼쪽에 Console 창, 오른쪽 위에 Environment 창, 오른쪽 아래에 Files 창이 표시된다.
- Console 창 오른쪽 위의 아이콘을 누르면 Script 창이 나타난다.



R Studio 인터페이스

Script 창

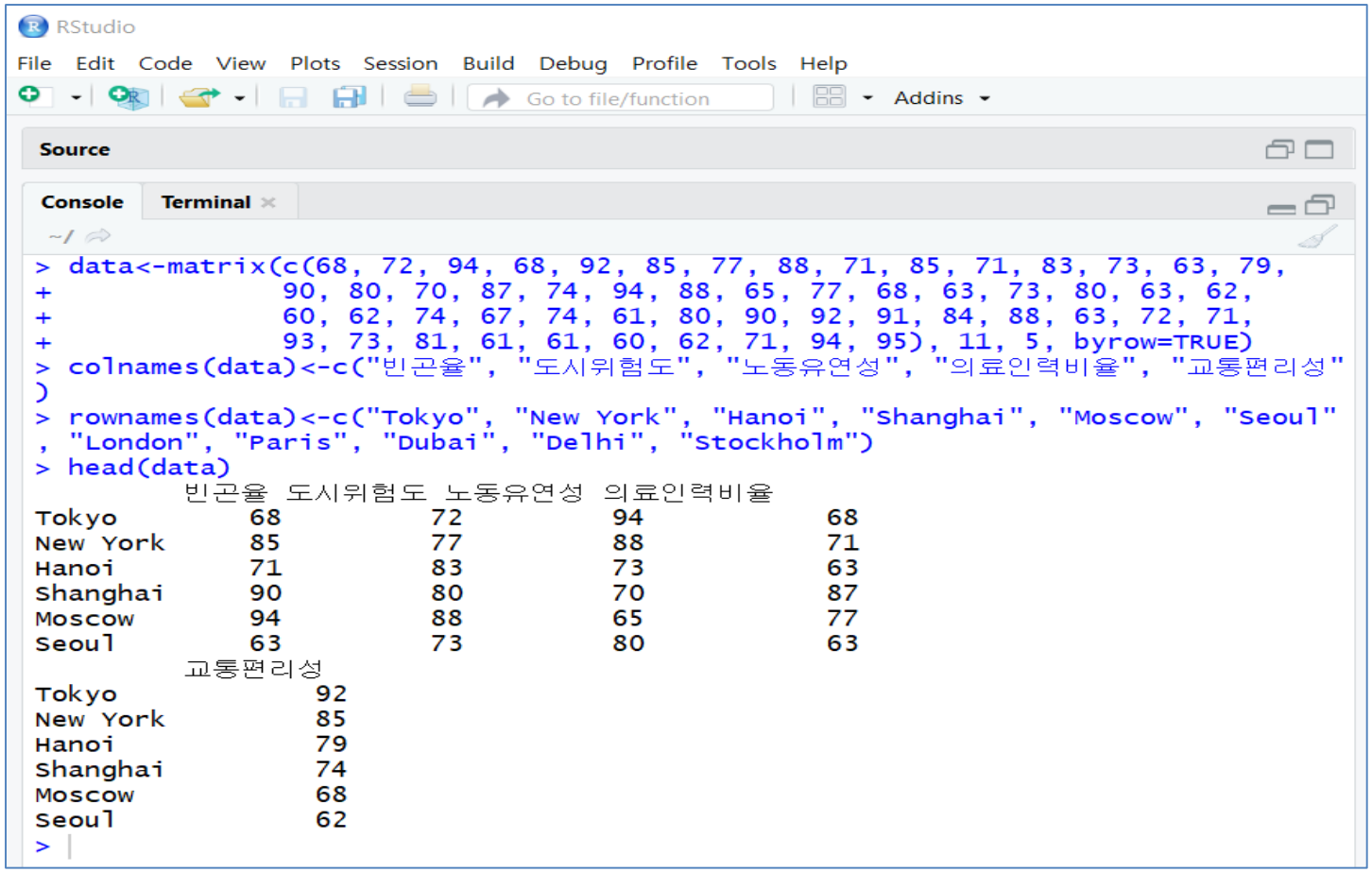
- **코드를 입력하고 저장하기 위한 창**
- **스크립트를 생성하고 코드를 실행할 수 있다.**



R Studio 인터페이스

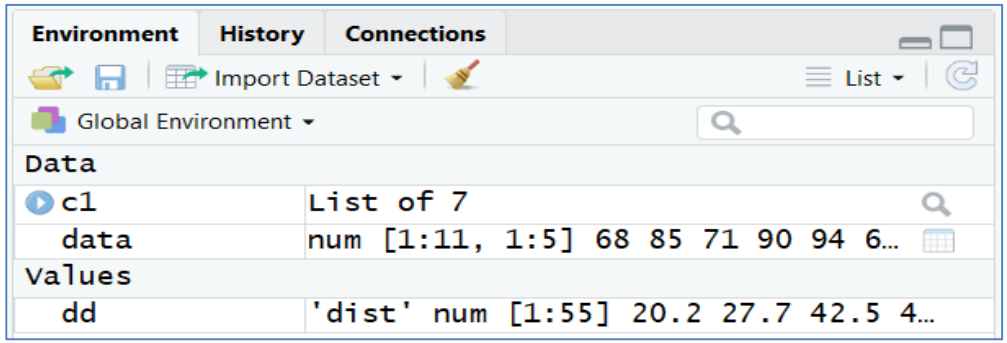
Console 창

- R 코드를 바로 입력하고 실행할 때 사용하는 창
- Script 창에서 작성하고 실행한 코드 및 결과, 실행한 코드의 오류나 오류 메시지 등도 Console 창에 표시된다.

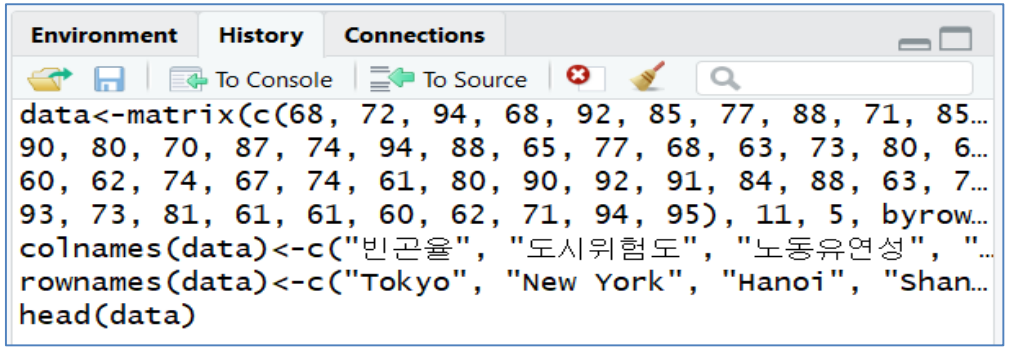


R Studio 인터페이스

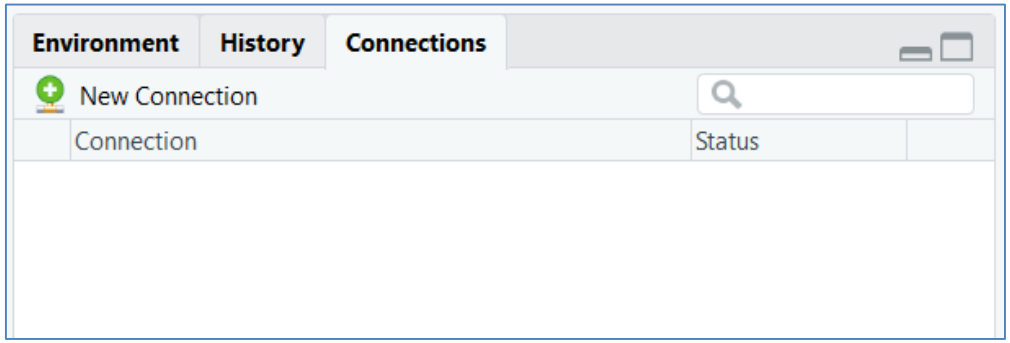
Environment 창 / History 창 / Connections 창



- 데이터 분석을 하면서 사용한 데이터 세트를 볼 수 있다.
- 데이터 세트의 이름과 데이터 값을 확인



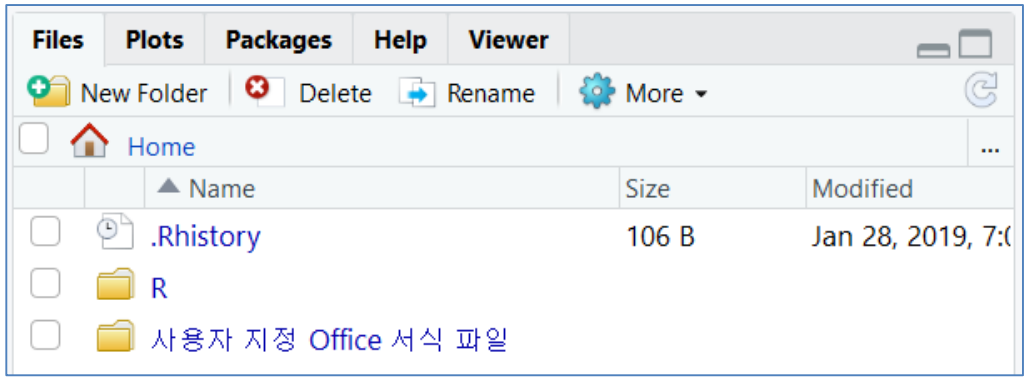
- R Studio에서 실행한 코드, 결과, 패키지 설치, 오류 등 모든 작업 과정을 확인



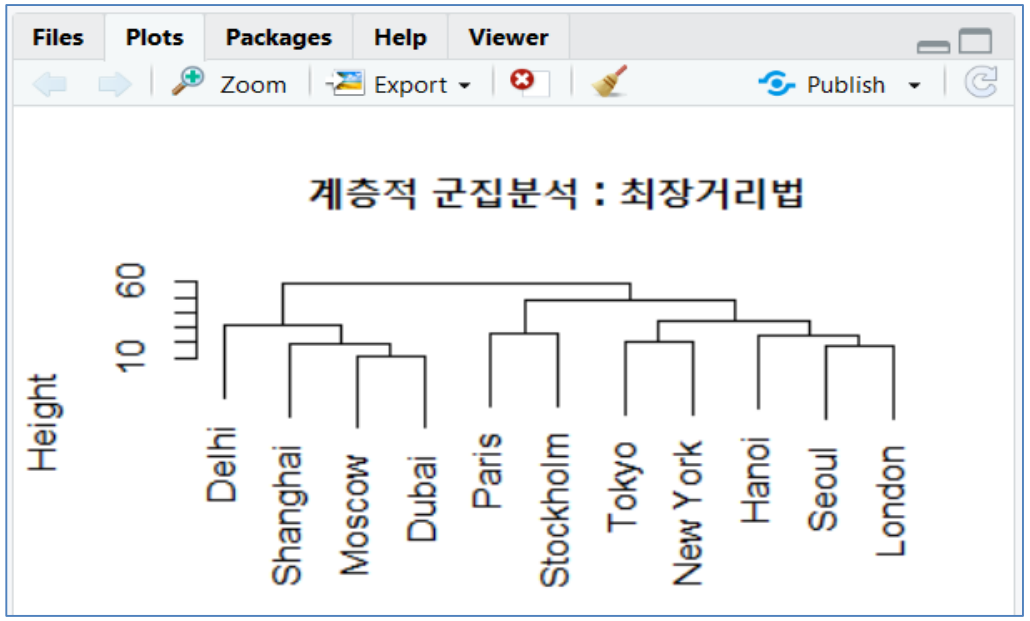
- 데이터 사이즈가 커질 때, 데이터베이스 서버와 R을 연결

R Studio 인터페이스

Files 창 / Plots 창 / Packages 창 / Help 창 / Viewer 창



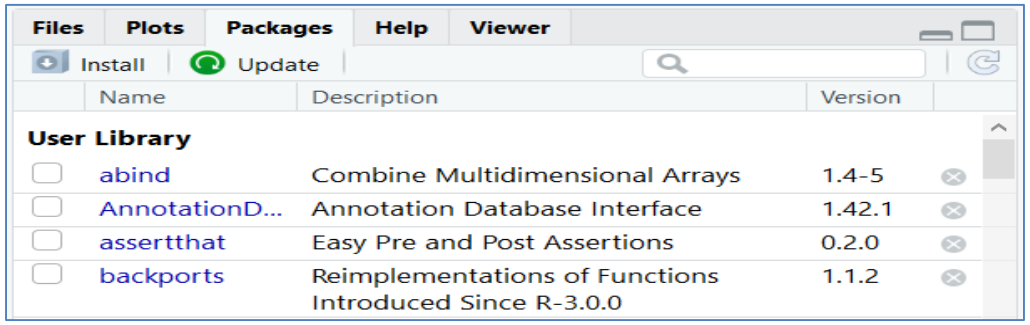
- 윈도우 파일 탐색기와 형태 및 용도가 비슷함



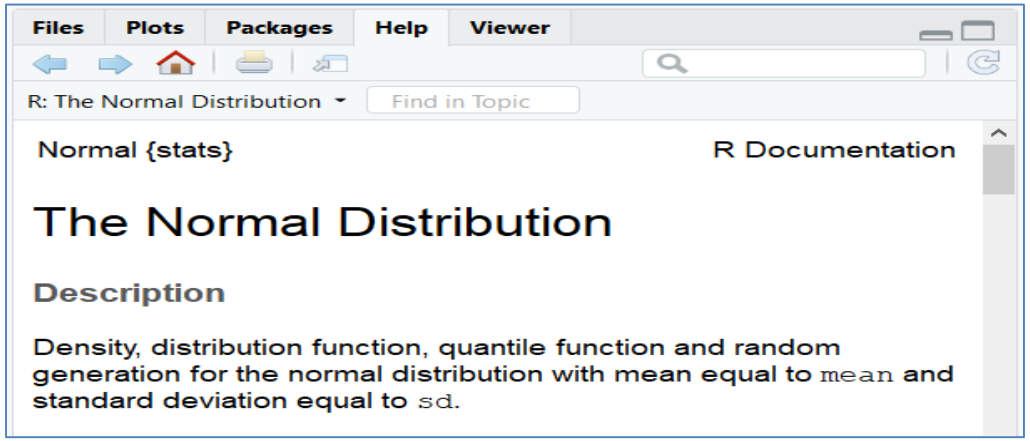
- R 코드로 작성한 그래프를 확인할 때 사용
- 그래프를 이미지 파일이나 PDF로 내보냄

R Studio 인터페이스

Files 창 / Plots 창 / Packages 창 / Help 창 / Viewer 창



- 현재까지 설치된 패키지 목록 표시
- 새로운 패키지를 설치하거나 업데이트

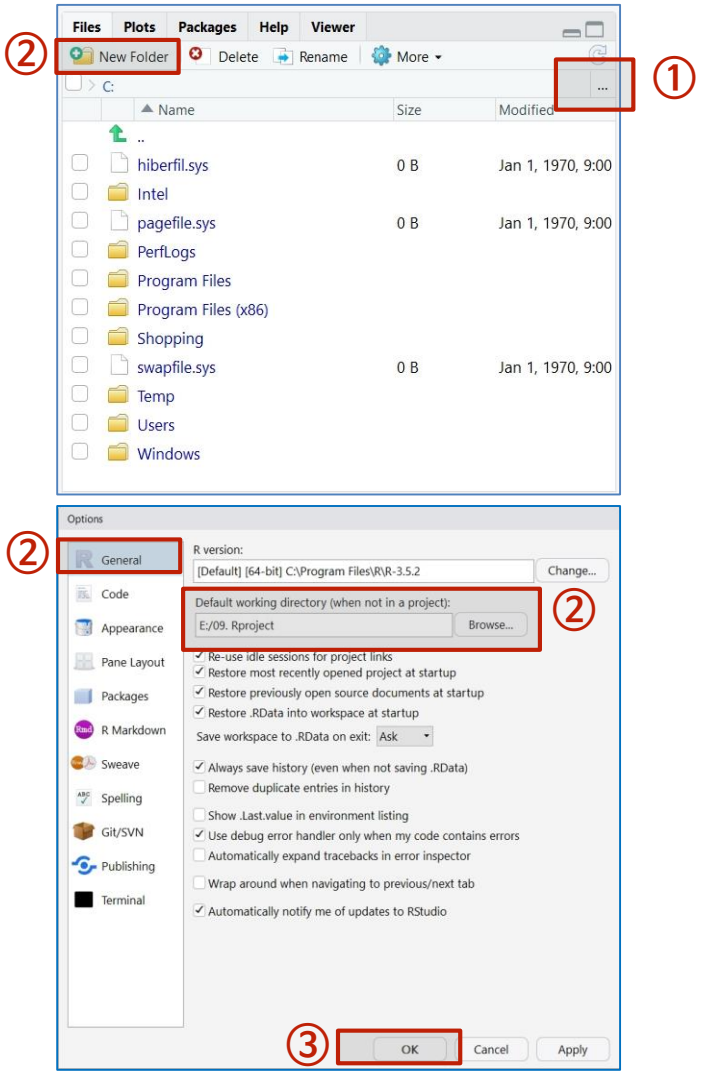


- R 을 사용하는 도중 도움말을 확인
- Help 창 오른쪽 상단의 검색 필드에 조회할 넥스트를 입력해서 결과를 확인

- R 코드를 웹 브라우저에 출력했을 때 결과를 확인할 수 있음

R Studio 작업 환경 설정

작업 폴더 생성



① Files 창에서 오른쪽 위에 있는 [···] 버튼을 클릭하여 작업 폴더를 생성할 경로를 선택한다.

② 새로운 폴더를 추가하기 위해서는 [New Folder] 버튼을 클릭한다.

③ New Folder 창이 열리면 폴더 이름을 입력 (예, R_Project 등) 하고 [OK] 버튼을 클릭한다.

R Studio는 실행될 때 작업 폴더를 자동으로 불러올 수 있도록 설정할 수 있다.

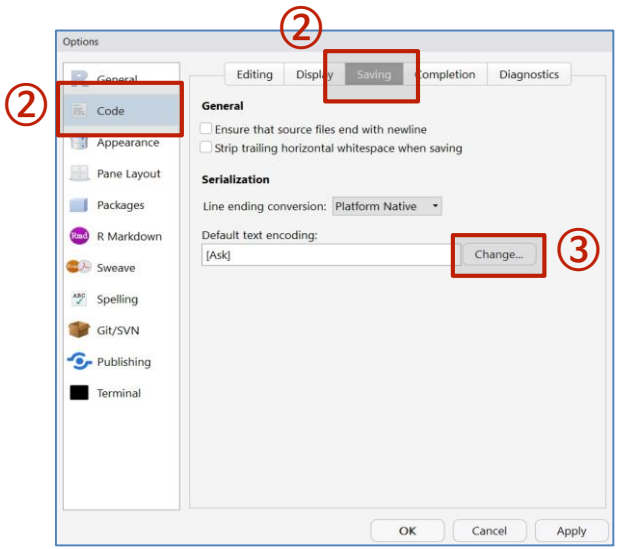
① [Tools → Global Options] 를 선택한다.

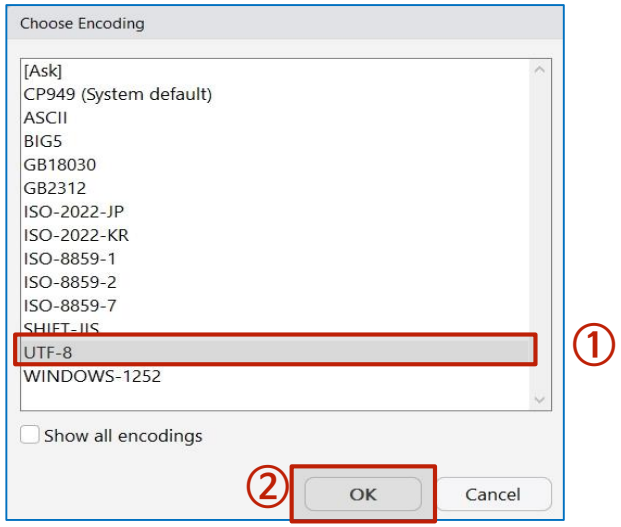
② Options 창이 열리면 [General] 분류에서 Default working directory 항목의 [Browse] 버튼을 클릭한다.

③ 기본 작업 공간으로 사용할 폴더를 선택하고 [OK] 버튼을 클릭한다.

R Studio 작업 환경 설정

작업 환경 설정





R Studio 에서는 스크립트에 한글이 제대로 표시되지 않는 경우가 있다. 이럴 때는 인코딩 설정을 UTF-8 방식으로 설정해야 한다.

① [Tools → Global Options] 를 선택한다.

② Options 창이 열리면 [Code] 분류에서 [Saving] 탭을 선택하고

③ Default text encoding 항목의 [Change] 버튼을 클릭한다.

① Choose Encoding 창이 열리면 [UTF-8]을 선택한다.

② [OK] 버튼을 클릭하여 설정을 완료한다.

제 3 장 R 데이터 구조

데이터 유형

가. R 데이터 유형

데이터 유형

- **R에서 사용되는 데이터 유형은 mode( ) 함수를 이용하여 확인할 수 있다.**

R 데이터 유형

- 수치형(numeric) : 숫자로 이루어졌으며 정수형(integer)과 실수형(double)으로 구분됨
- 논리형(logical) : 참 또는 거짓의 논리값이나 논리 연산자로 계산된 논리값
- 문자형(character) : 문자나 문자열
- 복소수형(complex) : 실수와 허수로 구성된 복소수

is.numeric(x)	수치형 여부	is.integer(x)	정수형	is.double(x)	실수형
is.logical(x)	논리형 여부	is.character(x)	문자형 여부	is.complex(x)	복소수형 여부
is.na(x)	NA여부	is.nan(x)	NaN 여부	is.null	NULL 여부
is.infinite(x)	무한수치 여부	is.finite(x)	유한수치 여부		

NA : 결측값(Missing value)

NULL : 데이터 유형과 자료의 길이도 0인 비어있는 값

NaN : 수학적으로 정의가 불가한 수

데이터 구조

가. R에서 사용되는 데이터 구조

데이터 구조

- **R에서 사용되는 데이터 구조는 다음과 같다.**

구분	내용	자료입력함수
벡터(vector)	하나 이상의 자료 값으로 1차원의 자료 구조	c(), rep(), seq(), sequence() 등
행렬(matrix)	동일한 유형의 자료 값으로 구성된 행과 열의 2차원 자료구조	cbind(), rbind(), matrix() 등
데이터 프레임 (data.frame)	변수와 관측치로 구성된 2차원의 자료구조	data.frame()
배열(arrary)	동일한 유형의 자료 값으로 구성된 2차원 이상의 자료구조	array() 등
리스트(list)	서로 다른 자료 유형으로 구성이 가능하며 자료 객체 중 가장 자유로운 자료구조	list()
요인(factor)	벡터 객체 중 범주형 데이터를 원소로 갖는 자료구조	factor(), ordered() 등
시계열(time series)	시간 등과 같이 일련의 시간 자료를 표현하는 자료구조	ts()

데이터 구조

나. 벡터(Vector)

숫자형 벡터

- 벡터를 생성하기 위해서는 c(…) 연산자를 사용해서 주어진 값으로부터 벡터를 생성한다.
- 벡터란 한 개 이상의 원소로 구성된 자료구조로서 R의 자료 객체 중에서 가장 기본이 되는 자료 객체

R CODE

```
x <- 10
y <- c(1, 2, 3, 4, 5)
z <- c(x, y)
str(z)
length(z)
```

R OUTPUT

```
> x <- 10
> y <- c(1, 2, 3, 4, 5)
> z <- c(x, y)
> z
[1] 10  1  2  3  4  5
> str(z)
 num [1:6] 10 1 2 3 4 5
> length(z)
[1] 6
```

데이터 구조

나. 벡터(Vector)

문자형 벡터

- 문자로 이루어진 데이터
- 변수를 생성할 때와 마찬가지로 할당할 문자 데이터를 따옴표(“” 또는 ‘’) 로 감싼 형식으로 구성

R CODE

```
y <- c(1, 2, 3, 4, 5)

a <- c('one', 'two', 'three')

d <- c(y, a)

str(y)

# 숫자형과 문자형이 결합되면 문자형으로 변환된다.
str(d)
```

R OUTPUT

```
> y <- c(1, 2, 3, 4, 5)
> a <- c('one', 'two', 'three')
> d <- c(y, a)
> d
[1] "1"     "2"     "3"     "4"     "5"     "one"   "two"   "three"
> str(y)
 num [1:5] 1 2 3 4 5
> str(d)
 chr [1:8] "1" "2" "3" "4" "5" "one" "two" "three"
```

데이터 구조

나. 벡터(Vector)

논리형 벡터

- TRUE와 FALSE 라는 논리값으로 이루어진 데이터
- 논리형 벡터는 주로 데이터 값을 비교할 때 사용한다.

R CODE

```
b <- c(TRUE, FALSE, TRUE)

str(b)
```

R OUTPUT

```
> b <- c(TRUE, FALSE, TRUE)
> b
[1]  TRUE FALSE  TRUE
> str(b)
 logi [1:3] TRUE FALSE TRUE
```

데이터 구조

나. 벡터(Vector)

벡터 예제

벡터 예제 (1/2)

```
# 지정된 값만큼 증감하는 데이터 생성
seq(1, 7, by=2)   # from, to, by, length, along
seq(1, -1, by=-0.5)
seq(1, 7, length=3)
rev(seq(1:5))   # rev : 자료의 순서를 역순으로 만드는 함수, 5:1

# 값이 반복되는 데이터 생성
rep(c(1,2,3), 3)  # rep(a, b)는 a를 b만큼 반복
rep(1:3, 3)     # a:b는 a부터 b까지의 수
rep(c(4, 2), times=2)
rep(c(4, 2), times=c(2, 1))
paste('no', 1:3)    # 반복되는 문자에 첨자를 붙여줌
```

데이터 구조

나. 벡터(Vector)

벡터 예제

벡터 예제 (2/2)

```
vec1 <- c(1, 2, 3, 4, 5)  #1~5까지 자료를 갖는 vec1 변수 생성
vec1[2]    # 두 번째 자료
vec1[c(2, 3, 5)]   # vec1의 2, 3, 5의 값만 표현
vec1[c(-2, -3)]   # vec1의 2, 3번째 자료 값 삭제
vec1[vec1 > 2]   # vec1에서 2보다 큰 값만 표현
vec1[2] <- 6   # 두 번째 위치의 2값이 6으로 대체됨

replace(vec1, 3, 2)   # vec1의 세 번째 자료를 2로 변경 ← replace(벡터, 위치, 값)
append(vec1, 8, after=5)   # vec1의 5번째 자료 다음에 8을 삽입 ← append(벡터, 값, after=위치)
```

데이터 구조

다. 행렬(Matrix)

행렬 생성하기

- 행렬(matrix)은 동일한 형으로 구성된 2차원의 데이터 구조
- 행의 차원과 열의 차원을 갖는 행렬은 수학에서의 행렬과는 달리 문자형이나 논리형 등을 원소로 사용

형식 : matrix(변수명, nrow = 행 개수, ncol = 열 개수)

행렬 예제 (1/2)

```
#행렬(matrix)은 여러 변수들이 이차원적으로 모여 있는 개체로,
#행렬을 생성하기 위해서는 matrix() 함수를 사용
#matrix() 함수 이외에 cbind(), rbind(), dim() 등을 이용하여 행렬을 생성시킬 수 있음

matrix(1:9, nrow=3)                    # nrow : 행의 개수 지정
matrix(c(1, 4, 7, 2, 5, 8, 3, 6, 9), byrow=T, ncol=3)  # ncol : 열의 개수 지정,  byrow=T : 행 기준 행렬을 생성
r1 <- c(1, 4, 7)                       #r1, r2, r3 행 벡터 생성
r2 <- c(2, 5, 8)
r3 <- c(3, 6, 9)
rbind(r1, r2, r3)                      # rbind : 행을 기준으로 결합
cbind(r1, r2, r3)                      # cbind : 열을 기준으로 결합
```

데이터 구조

다. 행렬(Matrix)

행렬 생성하기

- 행렬(matrix)은 동일한 형으로 구성된 2차원의 데이터 구조
- 행의 차원과 열의 차원을 갖는 행렬은 수학에서의 행렬과는 달리 문자형이나 논리형 등을 원소로 사용

행렬 예제 (2/2)

```
m1 <- 1:9
dim(m1) <- c(3, 3)

#행렬과 관련된 여러 함수와 성분의 추출과 삭제 등에 관해 알아봄
mat <- matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9), ncol=3, byrow=T) #행 기준 3열의 행렬 생성
mat[1,]                         #행렬 mat의 1행의 값
mat[,3]                         #행렬 mat의 3열의 값
mat[mat[,3] > 4, 1]                 #3열에서 4보다 큰 행의 값 중 1열의 모든 값
mat[mat[,3] > 4, 2]                 #3열에서 4보다 큰 행의 값 중 2열의 모든 값
mat[2,, drop=F]                   #2행 값만을 행렬 형태로 추출
is.matrix(mat[2,, drop=F])          #mat[2,,drop=F]가 행렬인지 확인
```

데이터 구조

라. 배열(Array)

배열 생성하기

- 배열(Array)은 행렬을 2차원 이상으로 확장시킨 객체로써 단일형 데이터
- 그러나, 일반적으로 3차원 이상의 차원을 갖는 데이터 객체를 배열이라고 부름

형식 : array(변수명, dim = c(행 수, 열 수, 차원 수))

배열 예제 (1/2)

```
#배열의 속성 :  행렬의 속성과 같이 자료의 개수를 나타내는 length, 형태를 보여주는 mode,
#각 차원의 벡터의 크기를 나타내는 dim 그리고 각 차원의 리스트 이름을 나타내는 dimnames로 구성


#배열의 생성
#배열을 생성하기 위한 함수로 array() 함수와 dim() 함수가 있음
array(1:6)                          #1~6의 자료로 1차원 배열 생성
array(1:6, c(2, 3))                 #1~6의 자료로 2차원 배열 생성
array(1:8, c(2, 2, 2))              #1~8의 자료로 3차원 배열 생성
arr <- c(1:24)                      #1~24의 자료 생성
dim(arr) <- c(3, 4, 2)              #dim() 함수를 이용하여 3행 4열의 행렬 2개 생성
```

데이터 구조

라. 배열(Array)

배열 생성하기

- 배열(Array)은 행렬을 2차원 이상으로 확장시킨 객체로써 단일형 데이터
- 그러나, 일반적으로 3차원 이상의 차원을 갖는 데이터 객체를 배열이라고 부름

배열 예제 (2/2)

```
#배열의 연산
ary1 <- array(1:8, dim = c(2, 2, 2))
ary2 <- array(8:1, dim = c(2, 2, 2))
ary1 + ary2                    #자료의 덧셈
ary1 * ary2                    #자료의 곱셈
ary1 %*% ary2                    #두 배열 원소들의 곱의 합
sum(ary1 * ary2)                    #ary1 %*% ary2 와 같은 결과를 냄

#배열원소의 추출 및 삭제
ary1[,,1]
ary1[1,1,]
ary1[1,,-2]
```

데이터 구조

마. 리스트(List)

리스트 생성하기

- 리스트(List)는 서로 다른 형태(mode)의 자료를 포함하는 1차원의 다중형 데이터
- 즉, 숫자형, 문자형 등 여러 가지 데이터 형을 동시에 포함할 수 있는 데이터 세트

리스트 예제 (1/2)

```
lst <- list('top', c(2, 4, 6), c(T, F, T))          #list(문자, 숫자, 논리형 객체)
lst[[1]]                                             #[[1]] 첫 번째 성분
lst[1]                                               #[1] 첫 번째 리스트
mat1 <- matrix(1:4, nrow=2)
list1 <- list('A', 1:8, mat1)
son <- list(son.name = c('Minsu', 'Minchul'), son.cnt = 2, son.age = c(2, 6))

#리스트 속성 : 벡터의 속성과 같이 자료의 개수, 형태, 구성요소의 이름 등을 보여주는 length, mode, names로 구성
length(son)                                          #son 리스트 자료의 개수
mode(son)                                            #son 리스트 자료의 형태
names(son)                                           #son 리스트 각 구성요소의 이름
```

데이터 구조

마. 리스트(List)

리스트 생성하기

- 리스트에서의 성분을 추출하는 방법은 [[]]를 사용하며, 성분의 이름을 지정하였다면 $연산자를 사용
- 단, 성분의 원소를 추출하고자 하는 경우에는 [] 연산자를 사용

리스트 예제 (2/2)

```
#관련함수
#리스트는 성분에 리스트와 벡터 등을 사용할 수 있음
#예제1
a <- 1:10
b <- 11:15
klist <- list(vec1=a, vec2=b, descrip='example')
klist[[2]][5]                    #두 번째 성분 vec2의 5번째 원소
klist$vec2[c(2, 3)]              #vec2의 2, 3번째 원소
```

데이터 구조

마. 데이터 프레임(Data Frame)

데이터 프레임 생성하기

- 데이터 프레임은 행렬과 비슷한 형태로 되어 있으나, 행렬은 차원으로 표시되며 같은 형태(mode)의 객체를 가지는 반면,
- 데이터 프레임은 각 열(column)들이 서로 다른 형태(mode)의 객체를 가질 수 있으므로 범주형 변수를 가질 수도 있기 때문에 범주형 자료분석에도 유용하게 사용

데이터프레임 예제 (1/2)

```
#data.frame() : 이미 생성되어 있는 벡터들을 결합하여 데이터 프레임을 생성
char1 <- rep(LETTERS[1:3], c(2, 2, 1))        #벡터 char1
num1 <- rep(1:3, c(2, 2, 1))                  #벡터 num1
test1 <- data.frame(char1, num1)              #test1 데이터 프레임 생성

#as.data.frame() :모든 다른 종류의 자료객체들을 데이터 프레임으로 변환
a1 <- c('a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o')
dim(a1) <- c(5,3)
test3 <- as.data.frame(a1)                    #a1 메트릭스를 데이터 프레임으로 변환
```

데이터 구조

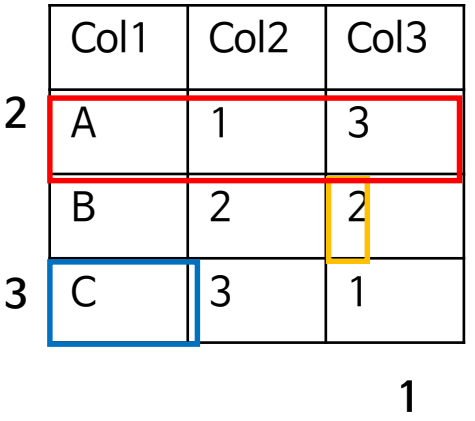
마. 데이터 프레임(Data Frame)

데이터 프레임 생성하기

- 데이터 프레임은 실무에서 가장 많이 사용하는 데이터 세트
- 숫자형 벡터, 문자형 벡터 등 서로 다른 형태의 데이터를 묶을 수 있는 다중형 데이터 세트
- 행렬과 유사하나 데이터 프레임의 각 열에는 변수명이 있어야 한다.

데이터프레임 예제 (2/2)

```
df1 <- data.frame(Col1 = c('A', 'B', 'C'), Col2 = c(1, 2, 3), Col3 = c(3, 2, 1))
#df1 [행, 열]
df1 [ , 'Col3']   #결과 : 3, 2, 1 출력
df1 [1, ]         #결과 : A  1  3 출력
df1 [3, 'Col1']   #결과 : C 출력
```



제 4 장 R 데이터 수집

데이터 수집

가. 외부로 데이터 저장하기

데이터 Export

- 데이터를 저장하는 함수 중에서 write.table(), write.csv() 함수에 대하여 알아본다.

데이터 Export

```
#Sample Data 생성

n <- 1e6
DT <- data.frame( a=sample(1:1000, n, replace=TRUE),
          b=sample(1:1000, n, replace=TRUE),
          c=rnorm(n),
          d=sample(c('foo', 'bar', 'baz', 'qux', 'quux'), n, replace=TRUE),
          e=rnorm(n),
          f=sample(1:1000, n, replace=TRUE) )

# txt 파일로 저장하기 예시 1
write.table(DT, 'E:₩₩09. Rproject₩₩DATA₩₩test.txt', sep=',', row.names=FALSE, quote=FALSE)

# CSV 파일로 저장하기 예시 2
write.csv(DT, 'E:₩₩09. Rproject₩₩DATA₩₩test.csv', row.names=FALSE, quote=FALSE)
```

데이터 수집

나. 외부 데이터 불러오기

데이터 Import

- 파일에서 데이터를 가져오기는 다양한 방법이 존재
- 이번 장에서는 read.csv(), read.table(), fread(), read.csv.sql() 함수에 대하여 알아본다.

데이터 Import

```
# read.csv()로 CSV 파일 가져오기 예시 1
DF1 <-read.csv('E:₩₩09. Rproject₩₩DATA₩₩test.csv', stringsAsFactors=FALSE)

# read.table()로 txt 파일 가져오기 예시 2
DF2 <- read.table('E:₩₩09. Rproject₩₩DATA₩₩test.txt', header=TRUE, sep=',', quote='',
                  stringsAsFactors=FALSE, comment.char='', nrows=n,
                  colClasses=c('integer', 'integer', 'numeric', 'character', 'numeric', 'integer')

# fread()로 CSV 파일 가져오기 예시 3 : data.table::fread()로 csv file import.
install.packages("data.table")
library(data.table)
DT1 <- fread('E:₩₩09. Rproject₩₩DATA₩₩test.csv')

# read.csv.sql()로 CSV 파일 가져오기 예시 4 : sqldf::read.csv.sql()로 csv file import
install.packages("sqldf")
library(sqldf)
SQLDF <- read.csv.sql('E:₩₩09. Rproject₩₩DATA₩₩test.csv', dbname=NULL)
```

제 5 장 R 데이터 가공

데이터 분석의 기초

가. 연산자

산술 연산

- **산술 연산은 데이터 값을 계산할 때 필요한 기본 연산자**

산술 연산자	기능
+	더하기
-	빼기
*	곱하기
/	나누기
%/%	몫
%%	나머지
** 또는 ^	제곱

산술 연산 예제

```
> 1 + 3     # 1과 3 더하기
[1] 4
> 6 - 1     # 6에서 1 빼기
[1] 5
> 3 * 9     # 3과 9 곱하기
[1] 27
> 21 / 7    # 21을 7로 나누기
[1] 3
> 30 %/% 8     # 30을 8로 나눈 몫
[1] 2
> 30 %% 8      # 30을 8로 나눈 나머지
[1] 6
> 8^2       # 8의 2 제곱
[1] 64
```

데이터 분석의 기초

가. 연산자

비교 연산

- 비교 연산자는 다양한 데이터를 서로 비교하여 TRUE 또는 FALSE 값을 반환
- 비교한 내용이 맞으면 TRUE를 반환, 틀리면 FALSE를 반환

비교 연산자	기능
〉	크다(Greater than)
〉=	크거나 같다(Greater than or Equal to)
〈	작다(Less than)
〈=	작거나 같다(Less than or Equal to)
==	같다(Equal to)
!=	같지 않다(Not Equal to)

비교 연산 예제

```
> 7 > 5        # 7은 5보다 크다
[1] TRUE
> 3 >= 6       # 3은 6보다 크거나 같다
[1] FALSE
> 4 < 3        # 4는 3보다 작다
[1] FALSE
> 2 <= 9       # 2는 9보다 작거나 같다
[1] TRUE
> 6 == 6       # 6은 6과 같다
[1] TRUE
> 6 != 6       # 6은 6과 같지 않다
[1] FALSE
```

데이터 분석의 기초

가. 연산자

논리 연산

- 비교 연산자를 통해 얻은 진리 값을 다시 연산할 때 사용
- & 연산자는 양쪽의 조건 값이 모두 충족될 때만 TRUE를 반환
- | 연산자를 사용하면 한 쪽의 조건 값이라도 충족되는 경우에 TRUE를 반환

논리 연산자	기능
&	and
|	or

논리 연산 예제

```
> x<-1:5
> y<-5:1

# x 값이 3보다 크고 y 값이 1보다 크면 TRUE, 아니면 FALSE
> (x > 3) & (y > 1)
[1] FALSE FALSE FALSE  TRUE FALSE

# x 값이 3보다 크거나 y 값이 2보다 크면 TRUE, 아니면 FALSE
> (x > 3) | (y > 2)
[1] TRUE TRUE TRUE TRUE TRUE
```

데이터 분석의 기초

나. 변수명 변경하기

- 변수명은 기억하기 쉽게, 그리고 규칙을 정해서 사용하는 것이 필요
- 변수명을 변경할 때는 dplyr 패키지에 들어 있는 rename() 함수를 사용

변수명 변경하기 예제

```
library(dplyr)
library(MASS)

data(Boston)
str(Boston)
Boston_rename <- rename(Boston, room = rm, Y = medv)
View(Boston_rename)
```

rm	age	dis	rad	tax	ptratio	black	lstat	medv
6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4

room	age	dis	rad	tax	ptratio	black	lstat	Y
6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4

데이터 분석의 기초

다. 파생 변수 생성하기

- **연산을 통해 새로운 결과를 얻고, 그 결과로 새로운 변수를 생성하는 것을 파생 변수 생성이라 한다.**
- **변수의 특징에 따라 다양한 분석을 할 수 있도록 추가로 생성하는 변수**

파생 변수 생성 예제

```
#  rm 변수는 주택 1 가구당 평균 방의 개수를 나타내는 변수이다.
#  rm 의 중앙값(6.208)과 평균(6.285)의 근사치를 기준으로 6 이상이면 'lot' 값, 그 외의 경우는 'few' 값을 가지는
#  rm_6 라는 파생 변수를 생성

Boston$rm_6 <- ifelse(Boston$rm >= 6, "lot", "few")
```

rm	age	dis	rad	tax	ptratio	black	lstat	medv	rm_6
5.813	90.3	4.6820	4	307	21.0	376.88	14.81	16.6	few
6.047	88.8	4.4534	4	307	21.0	306.38	17.28	14.8	lot
6.495	94.4	4.4547	4	307	21.0	387.94	12.80	18.4	lot
6.674	87.3	4.2390	4	307	21.0	380.23	11.98	21.0	lot
5.713	94.1	4.2330	4	307	21.0	360.17	22.60	12.7	few
6.072	100.0	4.1750	4	307	21.0	376.73	13.04	14.5	lot
5.950	82.0	3.9900	4	307	21.0	232.60	27.71	13.2	few

데이터 분석의 기초

라. 조건문(if/ifelse)

- **ifelse() 조건문은 다음과 같은 형식으로 사용하며, 여러 조건이 중첩될 때 사용한다.**

형식 : ifelse(조건, 참일 때 값, 거짓일 때 값)

조건문

```
#특정한 조건을 만족했을 경우에만 프로그램 코드를 수행하는 제어 구문. 항상 논리 연산이 수반 된다
#if(조건) 실행문
x <- c(1, 2, 3, 4); y <- c(2, 1, 4, 5)
if(sum(x) < sum(y)) print(x)    #x의 합이 y의 합보다 작을 경우 실행

#if(조건) {
#   조건이 True 일때 실행문
# } else {
#   조건이 False 일때 실행문
# }
x <- c(1,2,3,4)
y <- c(2,1,4,5)
if(mean(x)>mean(y)) print('Mean(x)>Mean(y)') else print('Mean(x)<Mean(y)')
```

데이터 전처리

가. 필요한 데이터 추출

원하는 변수를 선택하는 select() 함수

- **데이터셋에서 있는 변수 중 필요한 변수만 별도로 추출할 때 사용**
- **주로 특정 열을 추출하는 기능**

select() 함수 예제

```
# Boston 데이터 셋에서 여러 변수들 중 지정한 rm 변수만 추출
# dplyr 패키지의 select() 함수를 사용한다는 것을 명시적으로 표현
Boston %>% dplyr::select(rm)

# Boston 데이터 셋에서 rm, lstat, medv 변수 추출
Boston %>% dplyr::select(rm, lstat, medv)

# Boston 데이터 셋에서 지정한 rm 변수만을 제외하고 추출
Boston %>% dplyr::select(-rm)

# Boston 데이터 셋에서 rm, lstat 변수를 제외하고 추출
Boston %>% dplyr::select(-rm, -lstat)
```

데이터 전처리

가. 필요한 데이터 추출

조건절을 추출하는 filter() 함수

- **filter() 함수는 필요한 조건을 지정하여 조건에 맞는 데이터만 추출**

filter() 함수 예제

```
# Boston 데이터 셋에서 rm 변수가 6 이상인 것만 추출
Boston %>% filter(rm > 6)

# Boston 데이터 셋에서 lstat 변수는 모집단의 하위계층의 비율(%)이다. 평균 12.65
# Boston 데이터 셋에서 rm 이 6 이상이면서 lstat 가 12 이상인 경우만 추출
Boston %>% filter(rm > 6 & lstat > 12)
```

데이터 전처리

다. 결측값(missing value)

- 결측값은 데이터가 없는 것을 말한다.
- R에서 결측값은 NA로 표시되며, 결측값은 연산을 해도 결과가 NA이다. 결측값을 제외하고 연산할 때는 na.rm=T 옵션을 사용한다.

결측값 예제

```
> x<-c(1, 2, 3, NA, 5)                # 벡터 x에 결측값 할당
> x
[1]  1  2  3 NA  5
> x*2                              # 결측값이 있으면 연산을 해도 결과가 NA
[1]  2  4  6 NA 10
> is.na(x)                            # 벡터 x에서 결측값 조회
[1] FALSE FALSE FALSE  TRUE FALSE
> table(is.na(x))                        # 벡터 x의 결측값 건수 조회
FALSE  TRUE
    4     1
> sum(x)                              # 벡터 x의 합계는 결측값이 있어서 결과가 NA
[1] NA
> sum(x, na.rm=T)                        # 결측값을 제외하고 합계를 구함
[1] 11
```

사용자 정의 함수

- **함수의 정의 : 함수(function)란 특정한 작업을 독립적으로 수행하는 프로그램 코드의 집합체로서 R객체의 한 종류**
- **함수 사용의 이점**
  - **1) 커다란 프로그램을 작업 단위 별로 분할하여 작성할 수 있음**
  - **2) 동일한 프로그램 코드를 매번 기입하는 수고를 덜어 줌**

사용자 정의 함수 작성법

함수이름 <- function(인수) {함수의 몸체}

- function : 함수를 정의하는데 사용하는 예약어이다
- 인수 : 함수가 사용할 인수들.
- 함수의 몸체 : 함수가 호출되면 실행하게 될 프로그램 코드들의 집합.
- 함수이름 : 정의된 함수를 할당하므로 생성되는 객체이름.

[사용자 함수 예]

```
mean_fn <- function(data) {
            result <- sum(data) / length(data)
            return(result)
           }
```

```
x <- rnorm(100, mean = 4, sd = 1)
mean_fn(data = x)
```

사용자 정의 함수

- **사용자 정의 함수 예**

  1) 반올림을 위한 round() 함수를 사용자 정의 함수로 작성

사용자 정의 함수 예

```
# 반올림 (R은 'go to the even digit')

round2 <- function(x, n) {
  sing.x <- sign(x)
  z <- abs(x)*10^n
  z <- z + 0.5
  z <- trunc(z)
  z <- z/10^n
  result <- z*sing.x

  return(result)
}

> x.c <- c(1.75, 1.54, 1.55, 1.85, 1.65)    # 벡터 x.c 생성

> round(x.c, 1)    # R 의 {base} 함수를 사용한 결과
[1] 1.8 1.5 1.6 1.8 1.6

> round2(x.c, 1)   # 사용자 정의 함수 round2() 를 사용한 결과
[1] 1.8 1.5 1.6 1.9 1.7
```

제 6 장 R 그래프 함수

그래프 함수

가. 대표적인 R 그래프 함수

- **R에서 많이 사용되어지는 대표적인 함수는 다음과 같다.**

대표적인 R 그래프 함수

plot	일반적인 도표 함수이다.
boxplot	박스 플랏을 그린다.
hist	히스토그램을 그려준다.
qqnorm	quantile-quantile 플랏을 그린다.
curve	함수를 그린다.
barplot	막대그래프를 그린다.
sunflowerplot	중첩된 케이스가 있을 때 자료의 크기를 꽃잎 형태로 표현한다.
pie	파이 차트를 그린다.
pairs	행렬 산점도를 그린다.
contour	벡터 x, y와 행렬 z에 대해 2차원 등고선 그래프를 그린다.
persp	3차원 그래프를 생성한다.
coplot	요인 별 산점도 그리기

그래프 함수

가. 대표적인 R 그래프 함수

- R에서 많이 사용되어지는 대표적인 함수는 다음과 같다.

부가적인 R 그래프 함수

points	점을 찍는다.
lines	선을 추가한다.
abline	직선을 긋는다.
segments	구획을 추가한다.
polygon	다각형을 추가한다.
text	텍스트를 추가한다.
grid	격자를 그린다.

그래프 함수

나. 산점도 그래프

- 그래프는 데이터를 시각적으로 보여주는데 중요한 역할을 한다.
- R 프로그램에서 그래프 함수를 잘 활용하면 데이터를 보다 명료하고 해석이 용이하게 할 수 있다.

산점도 그리기

#plot() 함수

#plot(x,y,main=,sub=,xlab =, ylab=, type=,axes='',col='',pch='')

#x : X축의 자료, y: Y축의 자료

#main : plot의 전체 제목

#sub : plot의 부 제목

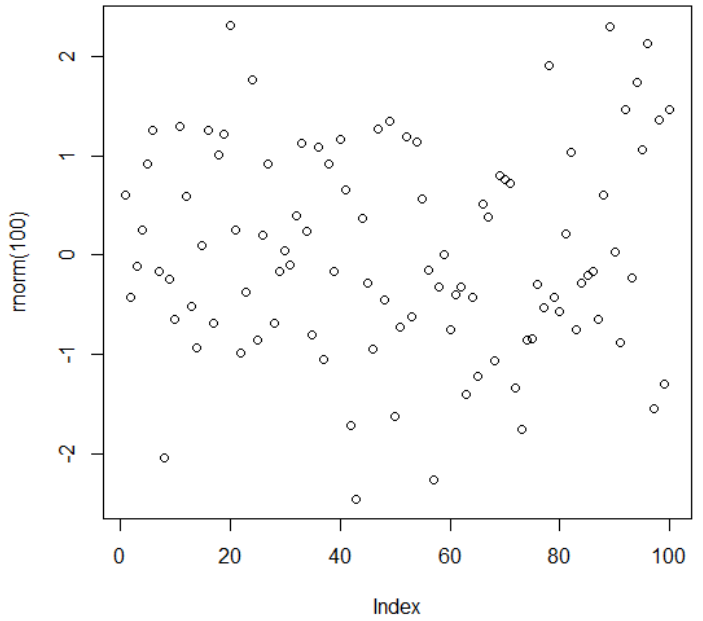
#xlab : x축 제목, ylab: y축의 제목.

#type : plot의 형태를 결정

#axes : plot의 테두리 관련

#col : plot의 색

plot(rnorm(100)) #100개의 난수에 대하여 산점도 생성



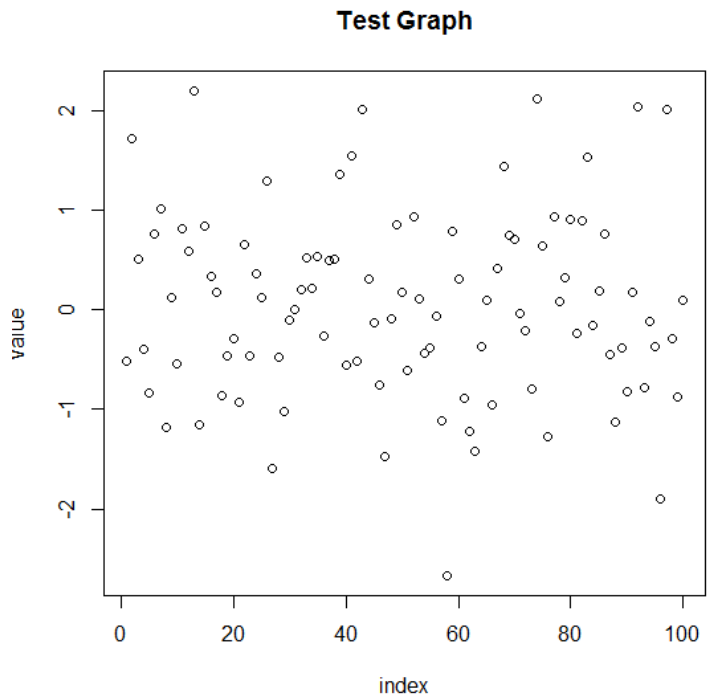
그래프 함수

나. 산점도 그래프

- 그래프에 제목을 추가하거나 축에 라벨을 추가하는 방법에 대하여 알아본다.

산점도 그리기 – 제목과 라벨 추가하기

```
#plot() 함수
#plot(x,y,main=,sub=,xlab =, ylab=, type=,axes='',col='',pch='')
# x : X축의 자료, y: Y축의 자료
#main : plot의 전체 제목
#sub : plot의 부 제목
#xlab : x축 제목, ylab: y축의 제목.
#type : plot의 형태를 결정
#axes : plot의 테두리 관련
#col : plot의 색
plot(rnorm(100), main = 'Test Graph', xlab = 'index', ylab = 'value')
#또는 아래의 방법을 사용
plot(rnorm(100), ann = FALSE)
title(main = 'Test Graph', xlab = 'index', ylab = 'value')
```



그래프 함수

나. 산점도 그래프

- 그래프 안에 범례를 추가 하기 위해서는 legend 함수를 사용하면 된다.

산점도 그리기 – 범례 추가하기

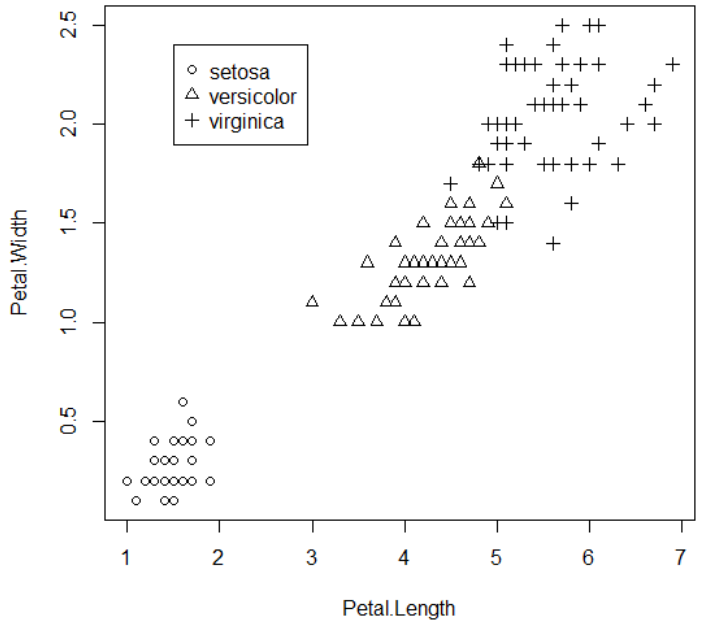
```
#점 범례
legend(x, y, labels, pch = c(유형1, 유형2, …)

#선의 유형에 다른 범례
legend(x, y, labels, lty = c(유형1, 유형2, …)

#선의 두께에 따른 범례
legend(x, y, labels, lwd = c(유형1, 유형2, …)

#색상 범례
legend(x, y, labels, col = c(유형1, 유형2, …)

with(iris, plot(Petal.Length, Petal.Width, pch = as.integer(Species)))
legend(1.5, 2.4, c('setosa', 'versicolor', 'virginica'), pch = 1:3)
```



그래프 함수

나. 산점도 그래프

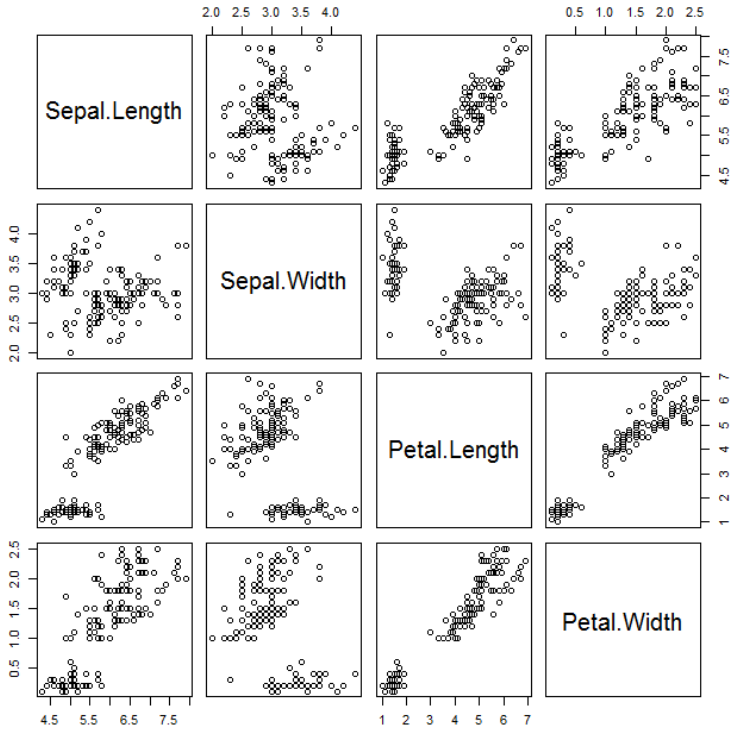
- 모든 변수들간 그래프 그리기에 대하여 알아본다.
- plot으로 그래프를 생성할 컬럼들의 조합의 데이터 프레임을 입력 값으로 넣으면 된다.

산점도 그리기 – 모든 변수들간 그래프 그리기

```
#모든 변수들간 그래프 그리기
plot(iris[, c('Sepal.Length', 'Sepal.Width', 'Petal.Length',
'Petal.Width')])

#또는

plot(iris[,c(1:4)])
```



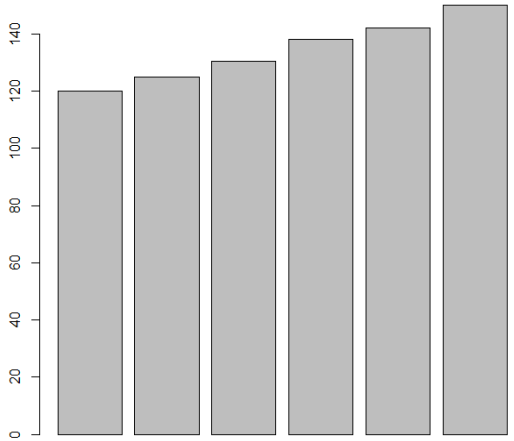
그래프 함수

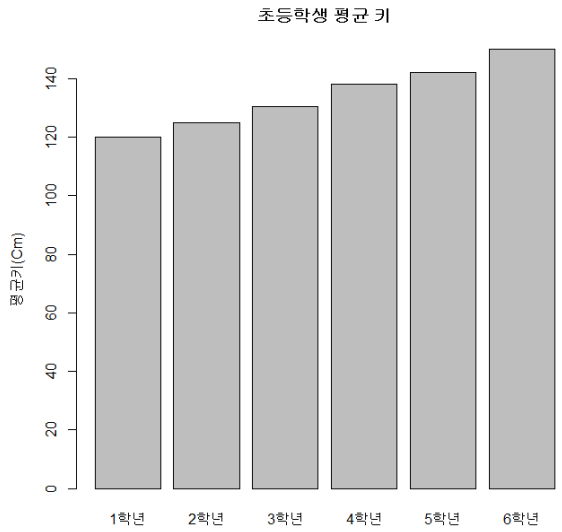
다. 막대 그래프

- 항목별 도수를 막대로 상대적인 길이로 나타내는 막대 그리프 그리기에 대하여 알아본다.
- 막대 그래프를 그리기 위해서는 barplot 함수를 사용한다.

막대 그래프 그리기

```
#막대 그래프 그리기
hight_x <- c(120, 125, 130.5, 138, 142, 150)
barplot(hight_x)
barplot(hight_x, main = '초등학생 평균 키', names.arg = c('1학년', '2학년', '3학년', '4학년', '5학년', '6학년'), ylab = '평균키(Cm)')
```





그래프 함수

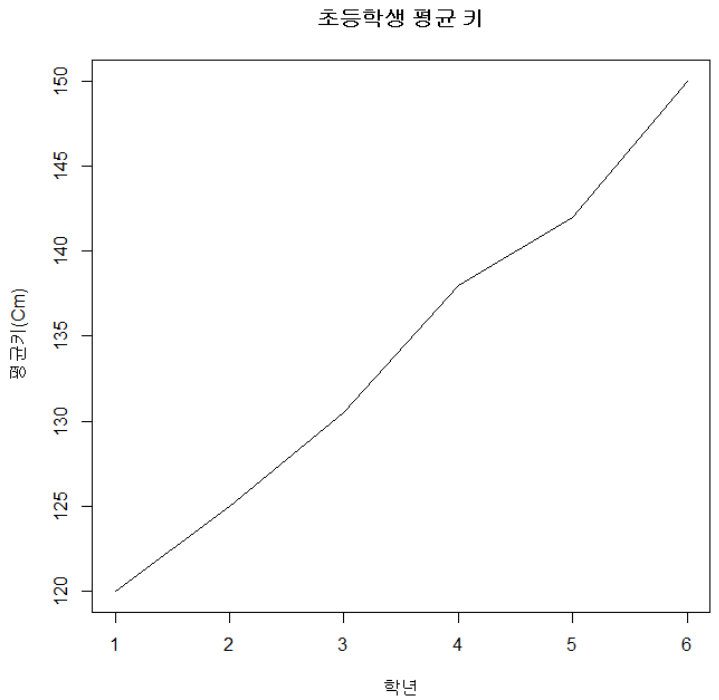
라. 선 그래프

- 선 그리기에 대하여 알아본다.
- 그래프 유형을 나타내는 type 인자를 ' l '로 설정한 plot함수를 사용한다.

선 그래프 그리기

```
#선 그래프 그리기
#plot() 함수
#plot(x,y,main=,sub=,xlab =, ylab=, type=,axes='',col='',pch='')
#x : X축의 자료, y: Y축의 자료
#main : plot의 전체 제목
#sub : plot의 부 제목
#xlab : x축 제목, ylab: y축의 제목.
#type : plot의 형태를 결정
#axes : plot의 테두리 관련
#col : plot의 색

 hight_x <- c(120, 125, 130.5, 138, 142, 150)
 class_x <- c(1, 2, 3, 4, 5, 6)
 plot(class_x, hight_x, type = 'l', main = '초등학생 평균 키',
        ylab = '평균키(Cm)', xlab = '학년')
```



그래프 함수

라. 선 그래프

- **선의 유형, 두께, 색상 변경하기에 대하여 알아본다.**

선의 유형 그리기

#선의 유형을 조정하려면 lty인자를 사용

실선(기본)	lty = 'solid' or lty = 1
대시선	lty = 'dashed' or lty = 2
점선	lty = 'dotted' or lty = 3
점대시선	lty = 'dotdash' or lty = 4
긴 대시선	lty = 'longdash' or lty = 5
이중 대시선	lty = 'twodash' or lty = 6
없음	lty = 'blank' or lty = 0

#선의 너비나 두께를 조정하려면 lwd 인자를 사용. 선의 두께는 1로 기본 설정되어있음
plot(x, y, type = 'l', lwd = 2)

#선의 색상을 조절하려면 col 매개변수를 사용. 선의 기본 색은 검은색으로 설정되어있음
plot(x, y, type = 'l', col = 'red')

그래프 함수

마. 히스토그램 (histogram)

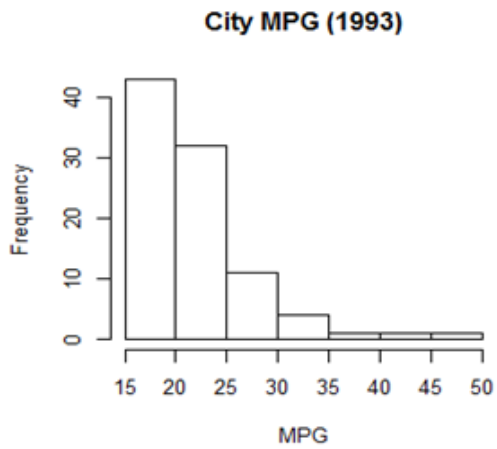
- **연속형 변수의 구간별 도수를 막대의 상대적 길이로 나타내는 히스토그램에 대하여 알아본다.**
- **hist()함수를 이용하여 숫자형 데이터 객체를 표현할 수 있다.**

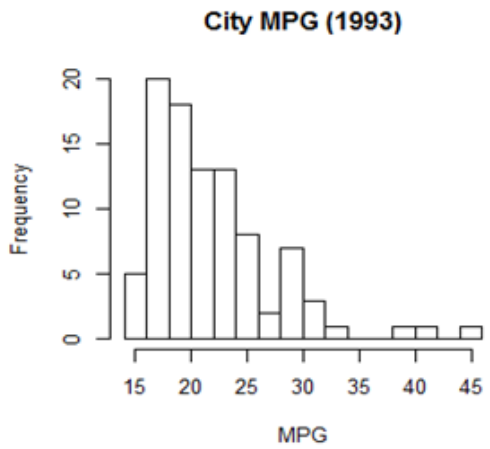
히스토그램 그리기

#객체 다음자리의 숫자는 breaks= 옵션인데, 간단한 숫자로 셀(막대)의 영역을 나누는 것 부터 셀에 대한 함수계산 수행까지 가능

```
data(Cars93, package  = 'MASS')
hist(Cars93$MPG.city, main = 'City MPG (1993)',  xlab= 'MPG')

hist(Cars93$MPG.city, breaks = 20,main = 'City MPG (1993)',  xlab= 'MPG')
```





그래프 함수

바. 상자 그림(box plot)

- 박스 플롯은 자료의 다섯 수치요약, 즉 최소값, 1사분위수, 중앙값, 3사분위수, 최대값으로 작성된다.
- 박스 플롯을 그리기 위해서는 boxplot 함수를 사용한다.

박스 플롯 그리기

```
x<-c(2, 5, 8, 5, 7, 10, 11, 3, 4, 7, 12, 15)
z<-c(3.5, 2.2, 1.5, 4.6, 6.9)

boxplot(x,z)

#또는

tmp_df <- data.frame(gubun = c(rep(1, 12), rep(2, 5)), value = c(x, z))
boxplot(value ~ gubun, data = tmp_df)
```