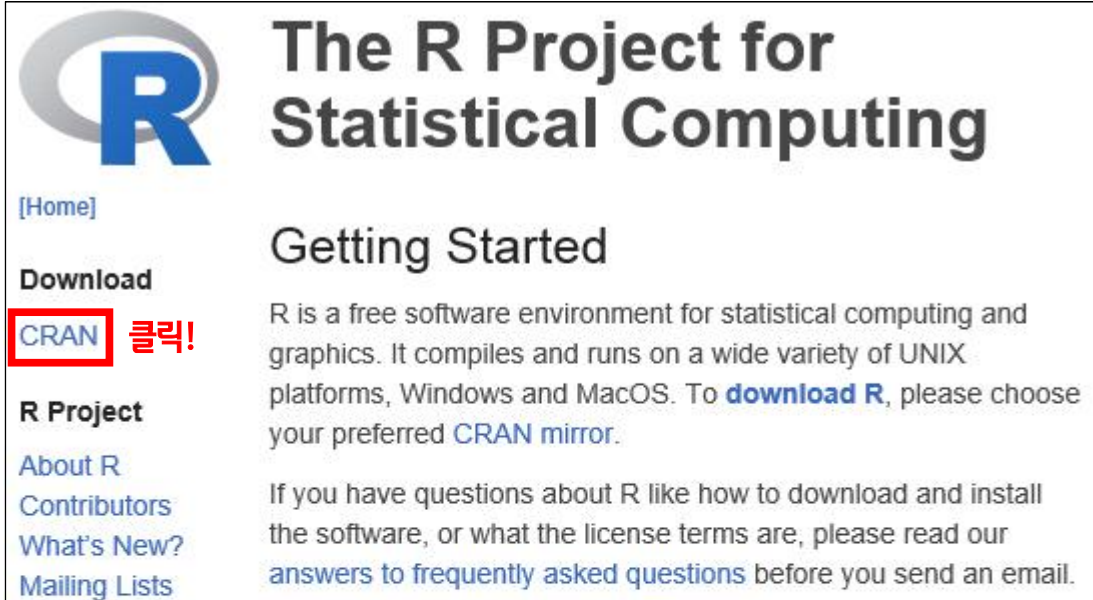


## - R 프로그램 기초 1 -

### 1. R 설치

<http://www.r-project.org>에 접속 후, 좌측 메뉴의 “Download > CRAN”을 클릭.



The R Project for Statistical Computing

[Home]

**Download**

**CRAN** 클릭!

**R Project**

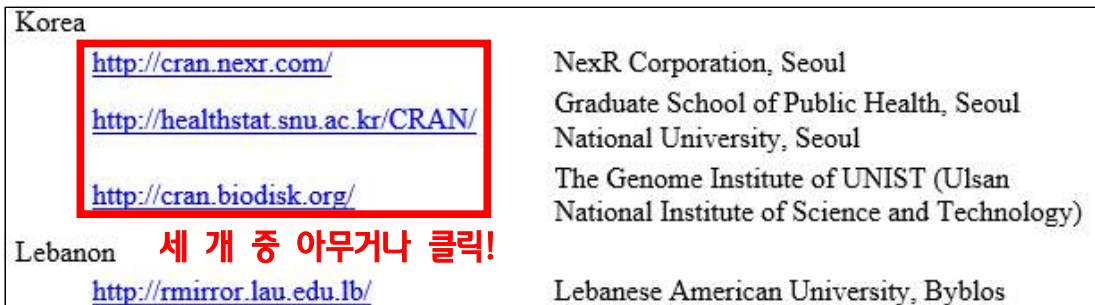
About R  
Contributors  
What's New?  
Mailing Lists

**Getting Started**

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

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

맨 위 중앙에 “CRAN Mirrors”라는 회색 제목이 적힌 화면으로 연결됨. 가장 왼쪽을 보면 영문 국가명이 알파벳 순으로 정렬되어 있음. 화면을 아래로 내려 “Korea”에 연결된 세 개의 서버 주소 중 아무거나 한 가지 클릭.



Korea

<http://cran.nexr.com/>  
<http://healthstat.snu.ac.kr/CRAN/>  
<http://cran.biodisk.org/>

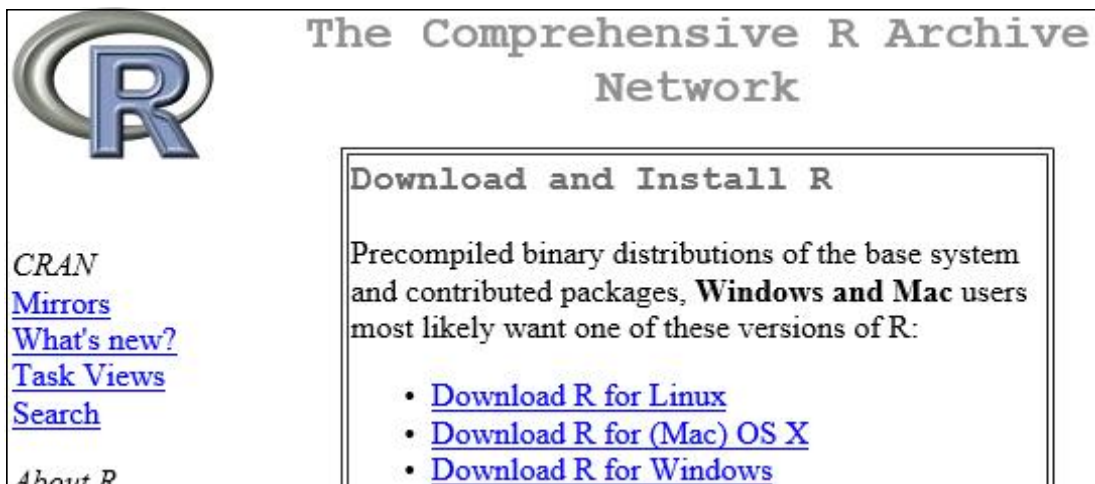
NexR Corporation, Seoul  
Graduate School of Public Health, Seoul  
National University, Seoul  
The Genome Institute of UNIST (Ulsan  
National Institute of Science and Technology)

Lebanon **세 개 중 아무거나 클릭!**

<http://rmirror.lau.edu.lb/>

Lebanese American University, Byblos

운영체제 선택 화면으로 연결됨. 세 가지 운영체제 중 PC에 설치된 것을 선택. (여기서는 Windows를 선택하겠음.)



The Comprehensive R Archive Network

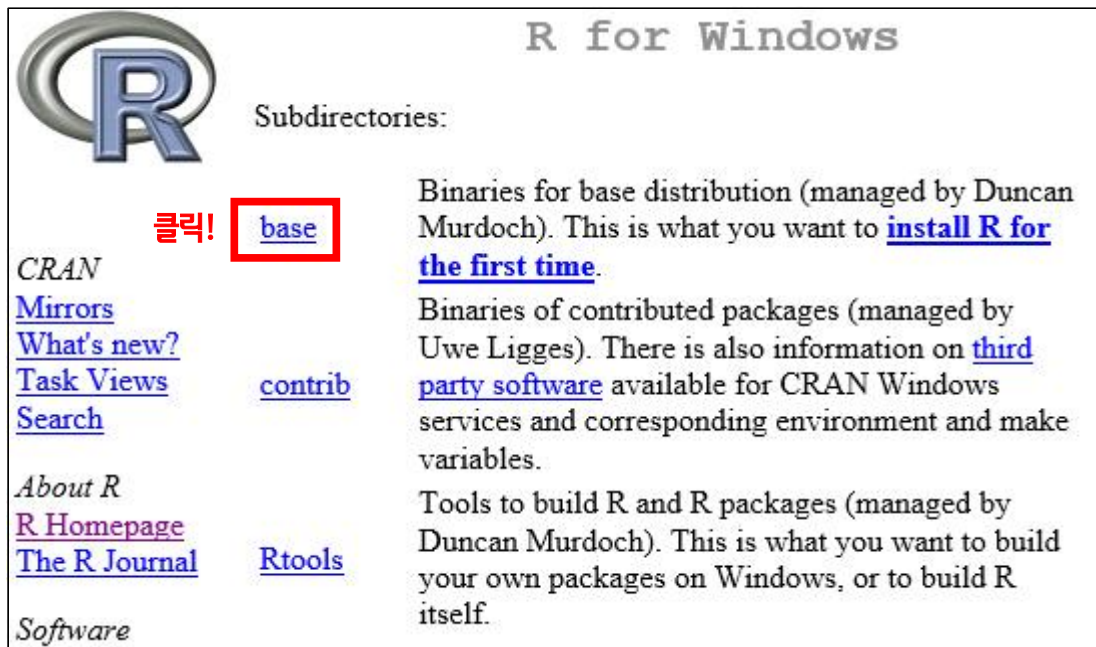
**Download and Install R**

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- [Download R for Linux](#)
- [Download R for \(Mac\) OS X](#)
- [Download R for Windows](#)

CRAN  
[Mirrors](#)  
[What's new?](#)  
[Task Views](#)  
[Search](#)  
About R

아래 화면으로 연결됨. 왼쪽의 "base" / "contrib" / "Rtools" 중 "base" 클릭.



연결되는 화면에서 "[Download R 3.2.1 for Windows](#)"를 클릭하여 설치파일을 다운로드 후 실행. 계속 "다음"을 클릭하면 설치 완료됨.

## 2. 벡터/행렬 관련 기초 사용법

### (1) 벡터/행렬 만들기 (생성)

① 벡터 만들기: `c()`, `append()`, `seq()`, `rep()` 함수

<pre>a &lt;- 1 #스칼라 만들기 a &lt;- c(1, 2); b &lt;- c(3, 4) #벡터 만들기 c &lt;- c(a, b) #a와 b를 연결해서 벡터 c 생성 d &lt;- append(a, b) e &lt;- c(1:9, 0, 9:5) #class(d) 확인 f &lt;- a + b #연산 가능 : +, -, *, /</pre>	<pre>a &lt;- seq(1:10) #연속되는 자료의 벡터 만들기 b &lt;- seq(from=1, to=10, by=2) #class(b) 확인 c &lt;- seq(1, 10, 2) #b와 동일 d &lt;- seq(-3, 3, length=100) a &lt;- rep(1, 5) #반복되는 자료의 벡터 만들기 b &lt;- rep(c(2,3), 4) #class(b) 확인</pre>
---	--

② 행렬 만들기: `matrix()` 함수

<pre>A &lt;- matrix(c(1,2,3,4,5,6), nrow=3, ncol=2, byrow=F) #3x2 행렬. 1열부터 데이터 채우기</pre>	
<pre>B &lt;- matrix(c(1,2,3,4,5,6), 3, 2) C &lt;- matrix(1:6, 2, byrow=T)</pre>	<pre>O &lt;- matrix(0, 2, 2) #영행렬 I &lt;- matrix(1, 2, 2) #항등행렬</pre>

③ 벡터를 이용하여 행렬 만들기 : `cbind()`, `rbind()` 함수 (\*결합 함수)

<pre>a &lt;- c(1, 2, 3); b &lt;- c(3, 4, 5) c &lt;- cbind(a, b) #열로 결합. class(c) 확인. d &lt;- rbind(a, b) #행으로 결합</pre>	<pre>#데이터 속성을 matrix로 변환 a &lt;- c(1, 2, 3) b &lt;- as.matrix(a) #행렬로 변환. class(b) 확인</pre>
--	---

④ 원소 indexing

a <- c(1:9); a[2]; a[c(3,5)]; a[-2]; a[-c(3,5)] b <- a[3:5] #class(b) 확인	A <- matrix(a, 3); A[2,1] B <- A[1:2, 2:3]; C = A[,2] #class(B) 확인
---	---

⑤ 행/열 이름 만들기: names(), colnames(), rownames() 함수

a <- c(3, pi, 4) names(v) <- c("num1", "num2", "num3") v["num1"] #이름을 통한 indexing 가능 v[1]	A <- matrix(c(75, 80, 78, 92), 2, 2) colnames(A) <- c("Man", "Woman") rownames(B) <- c("Eng", "Math") B["Eng", "Woman"]; B[1,2]
--	--

(2) 연산

① 산술 연산: "+", "-", "%\*%", "\*\*", "/", "\*\*", "%/" 연산자, crossprod() 함수

A <- matrix(c(5,6,7,8), 2); B = matrix(c(2,2,2,2), 2)	
A+B; A-B #A B A%*%B #AxB crossprod(A, B) # $\times B$ . t(A)%*%B와 동일	A*B; A/B #원소별로 곱하기, 나누기 A**B # $(a_j)^{b_j}$ A%/%B; #나머지 연산자

② 논리 연산: "!", "&", "|" ⇒ (원소별) TRUE 또는 FALSE를 결과값으로 반환. (※TRUE, FALSE: 진리값)

as.numeric(T); #TRUE⇔1 as.numeric(F); #FALSE⇔0 !(TRUE); !(FALSE) T&T; T&F; F&F T T; T F; F T	x <- c(TRUE, TRUE, FALSE) y <- c(TRUE, TRUE, TRUE) !x x&y x y
--	---

③ 비교 연산: "==", "!=", ">", ">=", "<", "<=" ⇒ (원소별) TRUE 또는 FALSE를 결과값으로 반환.

1==1; 2>1; 1>=1; 3<1 #모두 TRUE 1!=1; 2<1; 1>1; 3<=1 #모두 FALSE (2>1) (3<1) #OR 연산. TRUE (2>1)&(3<1) #AND 연산. FALSE	A <- matrix(c(1,5,8,3), 2) B <- matrix(c(2,4,8,0), 2) A==B; A!=B A>B; A<B
---	--

(3) 벡터/행렬 관련 기본 함수

① 성질과 관련된 함수: length(), dim(), nrow(), ncol() 함수

a <- c(1, 2, 3, 4, 5, 6) length(a) #원소의 총 개수 dim(a) #Error! nrow(a) #Error! ncol(a) #Error!	A <- matrix(c(1,2,3,4,5,6), 2) length(A) #원소의 총 개수 dim(A) #행렬의 크기 nrow(A) #행 개수 ncol(A) #열 개수
---	---

② 조작 함수: t(), sort(), diag(), solve() 함수

A <- matrix(4:1, 2) #2x2 행렬 t(A) # a <- sort(A) #정렬. class(a) 확인.	diag(B) #행렬 B의 대각성분 diag(2) #I invA <- solve(A); A*%invA #invA=A <sup>-1</sup>
---	--

③ 최대값/최소값 관련 함수: max()/min(), pmax()/pmin(), which.max()/which.min()

a<-c(1,6,9,11); b<-c(2,5,7,10); c<-c(3,4,8,12) max(a); min(a,b) max(a,b,c); min(a,b,c) pmax(a,b); pmin(a,b,c) #원소별 최대/최소값 which.max(a) #최대값 위치의 index which.min(a) #최소값 위치의 index	A <- matrix(c(1,6,9,11), 2) B <- matrix(c(2,5,7,10), 2) C <- matrix(c(3,4,8,12), 2) max(A) #최대값 위치의 index pmax(A,B); pmin(A,B,C) #원소별 최대/최소값 pmax(A,B,C); pmin(A,B,C)
--	--

④ 기타 유용한 함수: any(), all()

a<-c(1,2,3,4); b<-c(1,5,3,6)	any(a==b) #한 원소라도 TRUE이면 TRUE all(a==b) #모든 원소가 TRUE이면 TRUE
---------------------------------	--

#### (4) 그 외 기초 사용법

① 논리 연산을 활용한 벡터/행렬 indexing

a <- c(1,-1,2,-2,3,-3,4,-4,5,-5,6,-6); b <- seq(1,length(a))/5	
a>0 a[a>0] #0보다 큰 원소만 추출 a[a%%2==0] #짝수 원소만 추출	a[a>b] idx <- which(a>b) #a>b를 만족하는 인덱스 a[idx] #결과 비교

② 특수한 변수: NA(Not Available), NaN(Not a Number), NULL

x<-NA; NA+2 is.na(x) #결측치(NA)가 있으면 TRUE sum(1,2,3,NA) sum(1,2,3,NA, na.rm=T) #결측치 제거 후 sum y<-c(1,2,NA); y[!is.na(y)] #결측치 제거	0/0 Inf - Inf is.nan(c(1:2, 0/0, 3)) #NaN이 있으면 TRUE
	a <- NULL #초기화 a[1]<-2; a[3]<-5; a

### 3. 기초 문법

#### (1) 변수 이름

- 변수 이름은 알파벳, 숫자, 마침표("."), 밑줄("\_")의 조합으로 만들 수 있음.
- 변수 이름의 시작은 알파벳 or 마침표(.)이어야 함. 숫자나 밑줄은 안됨. (예) \_A(X), .A(O)
- 대문자/소문자 구별.

## (2) 정규분포 관련 함수

- ①  $\mu, \sigma$  ) 관련 기능별 R 내장함수:

분포	확률밀도	누적확률	분위수	난수 생성
Normal 분포	$\text{dnorm}(x, \mu, \sigma)$	$\text{pnorm}(x, \mu, \sigma)$	$\text{qnorm}(p, \mu, \sigma)$	$\text{rnorm}(n, \mu, \sigma)$

- ② 연습 :  $N(0,1)$  난수를 생성하여 성질 체크하기

$x \leftarrow \text{rnorm}(10000, 0, 1)$ #N(0,1) 난수 10,000개 생성. $\text{rnorm}(10000)$ 로 해도 결과 동일.	
$\text{mean}(x)$ #평균 체크	$\text{quantile}(x, c(0.05, 0.5, 0.95))$ #분위수 체크
$\text{sd}(x)$ #분산 체크	

## (3) 그래프 그리기

- ① 고수준 그래프 함수:  $\text{hist}()$ ,  $\text{plot}()$  함수

$x \leftarrow \text{rnorm}(1000)$ #N(0,1) 난수 1,000개 생성 $\text{hist}(x, \text{prob}=T)$ #히스토그램.	$x = \text{seq}(-3, 3, \text{length}=100); y = \text{dnorm}(x)$ $\text{plot}(x, y, \text{type}="h")$ #type="h": 히스토그램
---	--

- ② 저수준 그래프:  $\text{lines}()$  함수 ← 그래프 겹쳐 그리기

$x = \text{seq}(-6, 6, \text{length}=100)$ $y1 = \text{dnorm}(x)$ #N(0,1) $y2 = \text{dnorm}(x, 2, 1)$ #N(2,1)	$\text{plot}(x, y1, \text{type}="l");$ #N(0,1). type="l" : 실선 $\text{lines}(x, y2, \text{lty}=2);$ #N(2,1) 겹쳐 그리기. lty=2 : 점선 $\text{text}(-2, 0.2, "N(0,1)"); \text{text}(4, 0.3, "N(2,1)");$
--	--

- ③ 다중 그래프 그리기 :  $\text{par}()$  함수의  $\text{mfrow}=c(\text{nrow}, \text{ncol})$  인수

$x = \text{seq}(-6, 6, \text{length}=100)$ $y1 = \text{dnorm}(x)$ #N(0,1) $y2 = \text{dnorm}(x, 2, 1)$ #N(2,1)	$\text{par}(\text{mfrow}=c(1,2))$ #다중 그래프를 그리기 위한 옵션 $\text{plot}(x, y1, \text{type}="l")$ #N(0,1). type="l" : 실선 $\text{plot}(x, y2, \text{type}="h")$ #N(2,1)
--	---

## (4) 제어문

- ① 반복(Loop)문 :  $\text{for}(\text{Loop 변수 in 시작값:끝값}) \{ \text{실행문} \}$

#1부터 10까지 제곱합 계산하기 $\text{sum2} \leftarrow 0$ #제곱합 $\text{for}(i \text{ in } 1:10) \{ \text{sum2} \leftarrow \text{sum2} + i^2 \}$ $\text{sum2}$
---

- ② 조건문 :  $\text{if}(\text{Condition}) \{ \text{Condition이 True일 때의 실행문} \} \text{ else } \{ \text{Condition이 False일 때의 실행문} \}$

#1부터 10까지 홀수 제곱합 계산하기 $\text{sum2} \leftarrow 0$ #제곱합, $\text{for}(i \text{ in } 1:10) \{ \text{if}(i \% 2 == 1) \{ \text{sum2} \leftarrow \text{sum2} + i^2 \} \}$ $\text{sum2}$
--

### (5) 사용자 정의함수

- 형식 : 함수이름 <- function(입력변수1, ..., 입력변수n) {실행 내용 return(결과값)}

(예1) 두 개의 수 중 큰 값을 반환하는 함수

```
getMax <- function(x, y)
{
  if(x>=y) {return(x)} else {return(y)}
}

#함수 사용
x=1; y=3
getMax(n1, n2)
```

(예2) 제곱 합 계산함수 :  $\sum_{i=n_1}^{n_2} i^2$

```
getSum2 <- function(n1, n2)
{
  sum2 = 0 #제곱합
  i = 0 #첨자
  for (i in n1:n2) { sum2 <- sum2 + i^2 }
  return(sum2)
}

#함수 사용
n1=1; n2=10
getSum2(n1, n2)
```

(예3) European Vanilla Call 옵션에 대한 Black-Scholes 공식:

$$C(T, K, S, \sigma, r) = SN(d_1) - Ke^{-rT}N(d_2). \text{ 단, } d_1 = \frac{\ln(S/K) + (r + \frac{1}{2}\sigma^2)T}{\sigma\sqrt{T}} \text{ \& } d_2 = d_1 - \sigma\sqrt{T}.$$

```
getBSCallPrice = function(T, K, S, sig, r)
{
  #T=옵션 만기, K=옵션 행사가격, S=기초자산 가격, sig = 변동성, r = 이자율
  d1 = (log(S/K) + (r+0.5*sig^2)*T)/(sig*sqrt(T))
  d2 = d1 - sig*sqrt(T)
  BSCall = S*pnorm(d1) - K*exp(-r*T)*pnorm(d2)

  return(BSCall)
}

#함수 사용
T=1; K=95; S=100; sig=0.2; r=0.02
getBSCallPrice(T, K, S, sig, r)
```