Week2

Kwon Namtaek

2021년 1월 13일

1 본격적으로 R을 배워보기 전에…

1.1 R 환경설정

R의 기본적인 자료형들을 보기 전에, 조금 꾸미고 사용하기 편한 환경으로 바꿔줍시다!

```
print('R')
```

```
## [1] "R"
```

```
cat('gogo')
```

```
## gogo
```

- 화면배치 : Tools - Global options - pane layout
- 색 변경 : Tools - Global options - appearance - Dracula
- 인코딩 설정 : Tools - Global options - code - Savings - 인코딩을 UTF-8로

1.2 워킹디렉토리

워킹디렉토리의 설정은 중요하다. 컴퓨터는 바보이기 때문에, 우리가 상상하는대로 모든 작업을 해주지 않는다. 우리가 'a.csv'라는 파일을 열고 싶다면, R을 그 파일이 있는 위치로 움직여줘야 한다. 진짜 움직이는게 아니라 위치를 찾아준다는 것! 현재 R 위치가 어디 있는지 확인해볼까?

```
getwd()
```

`getwd()` 함수를 사용하면, 현재 R이 어디에서 활동하는지를 확인할 수 있다. 그런데 우리가 필요한 데이터가 있는 위치로 R을 옮기고 싶다면 어떻게 해야 할까?

```
setwd("C:/Users/Namtaek/Desktop/Studyroom_SungKyun/Statistical_Data_Analysis_21_01/Week2")
```

사람들마다 setwd() 안에 들어가는 주소가 다를 것이다. 여기서 일반적으로 중요한 것은, 주소에 한글이 없는게 좋다는 점을 생각하자. 한글은 프로그래밍 과정에서 최대한 배척해야 한다.

1.3 슬쩍 맛보기

세부적으로 R이 어떤 구조인지 확인하기 전에, 슬쩍 만져보자. 먼저 계산기 처럼 사용해보자. 나는 돈계산할 때 R을 켜서 계산한다.

```
3 + 8 #
```

```
## [1] 11
```

```
3 - 1 #
```

```
## [1] 2
```

```
3 * 4 #
```

```
## [1] 12
```

```
4/2 #
```

```
## [1] 2
```

```
5%%3  #
```

```
## [1] 2
```

```
5%/%3 #
```

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

변수 할당하는 것은 = 혹은 <-를 이용한다. <-는 `alt + -`를 통해 빠르게 입력이 가능하다. <-가 =보다 우선순위라고 하지만, 나는 =를 더 많이 사용한다.

```
x = 3; x
```

```
## [1] 3
```

```
y <- 3; y
```

```
## [1] 3
```

```
z = x+y;z
```

```
## [1] 6
```

```
ch = 'hello world'
ch
```

```
## [1] "hello world"
```

R은 벡터를 기본 자료구조로 지니는데, 미리 맛보자. R은 또 처음 인덱스가 파이썬과 다르게 1이다. 매우 중요!

```r
x = c(1, 3, 6, 10); x
```

```
## [1]  1  3  6 10
```

```r
x + 0.2
```

```
## [1]  1.2  3.2  6.2 10.2
```

```r
x[1]
```

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

```r
x[3]
```

```
## [1] 6
```

변수 할당은 가능하면 기존 함수에다가 하지 않는게 좋다. 예를 들어 c에는 하면 안된다. c는 벡터를 만들기 때문에…!!! q는 R을 끄는거!

```r
c
```

```
## function (...)  .Primitive("c")
```

```r
q
```

```
## function (save = "default", status = 0, runLast = TRUE)
## .Internal(quit(save, status, runLast))
## <bytecode: 0x0000000015ccfc68>
## <environment: namespace:base>
```

help나 ?는 함수의 사용법이 궁금할 때 사용하는 것!

```r
help(c)
?c
```

2 자료형

R에서 많이 쓰이는 자료형은 수치형/문자형/팩터형/논리형/기타 총 5개가 있습니다.

2.1 수치형

```r
a1 = c(1, 2, 3)
a1
```

```
## [1] 1 2 3
```

```r
is.numeric(a1)
```

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

```r
typeof(a1)
```

```
## [1] "double"
```

```r
class(a1)
```

```
## [1] "numeric"
```

2.2 문자형

```r
b1 = c('a', 'b', '3')
b1
```

```
## [1] "a" "b" "3"
```

```r
is.character(b1)
```

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

```r
typeof(b1)
```

```
## [1] "character"
```

```r
class(b1)
```

```
## [1] "character"
```

수치형과 자료형을 변환하는 것. as.(자료형) 으로 변형 가능하다.

```r
b2 = c('1', '2', '5')
as.numeric(b2)
```

```
## [1] 1 2 5
```

```r
as.character(a1)
```

```
## [1] "1" "2" "3"
```

2.3 팩터형

```
c1 = c('a', 'b', 'a', 'a', 'b')
c1 = factor(c1) # as.factor(c1)
c1
```

```
## [1] a b a a b
## Levels: a b
```

```
levels(c1)  #
```

```
## [1] "a" "b"
```

```
is.factor(c1)
```

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

```
typeof(c1) # R     +            .
```

```
## [1] "integer"
```

```
class(c1)
```

```
## [1] "factor"
```

팩터는 말했다시피 복잡하다. 변형을 확인하자.

```
c2 = c('11', '9', '9', '11', '5', '11', '11')
c2 = as.factor(c2); c2
```

```
## [1] 11 9  9  11 5  11 11
## Levels: 11 5 9
```

```
as.numeric(c2)
```

```
## [1] 1 3 3 1 2 1 1
```

팩터는 저런 숫자처럼 보여도, 바로 numeric으로 바꾸면 안된다. character로 바꾼 다음에 numeric으로 바꿔주자

```
as.numeric(as.character(c2))
```

```
## [1] 11  9  9 11  5 11 11
```

2.4 논리형

```
c(T, TRUE, F, FALSE)
```

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

```
d1 = TRUE; d1
```

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

```
d2 = 'TRUE'; d2
```

```
## [1] "TRUE"
```

```
'a' == 'a'
```

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

```
'a' != 'b'
```

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

2.5 기타

NA(Not Available)는 결측값이다. 말그대로 비어있는 값. NULL은 비어있는 값이니 해당 변수를 삭제가 아니라 비어있게 만든다. Nan(Not a Number)은 수학적으로 정의되지 않는 수. 0/0같은 값이 나오면 발생한다.

```
NA
```

```
## [1] NA
```

```
e1 = c(1, 3, NA, 6); e1
```

```
## [1]  1  3 NA  6
```

```
is.na(e1)
```

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

```
e1 = NULL; e1
```

```
## NULL
```

```
0/0
```

```
## [1] NaN
```

자료구조를 다루기 전에, 지금가지 정의한 변수들을 다 삭제하자.

```r
rm(list = ls())
```

3 자료구조

자료구조로는 vector(벡터), matrix(행렬), dataframe(데이터프레임), list(리스트) 등이 있다. 여기서 리스트는 빼고 다루자.

3.1 벡터

3.1.1 벡터의 기본 구조

R의 가장 기본적인 자료구조다. 파이썬과 R의 가장 큰 차이점중 하나는, 이런 벡터, 행렬, 데이터프레임이 R자체에서 정의되어 있어서 데이터분석에 있어서 자료구조에 대한 깊은 이해 없이도 비교적 편하게 함수들을 사용할 수 있다는 점이다. 파이썬은 해당 자료구조가 없기 때문에, numpy와 pandas를 끌어오게 된다.

```r
x = c(1, 5, 3, 12, 52)
length(x) #
```

```
## [1] 5
```

```r
x[1:5] #
```

```
## [1]  1  5  3 12 52
```

```r
x = c(x[1:3], 153, x[5]); x
```

```
## [1]   1   5   3 153  52
```

```r
x > 3
```

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

```r
names(x) = c('g', 'b', 'a', 'b', 'c')
x
```

```
##   g   b   a   b   c
##   1   5   3 153  52
```

3.1.2 벡터 생성

벡터를 생성하는데, 규칙을 갖고 생성하는 방법은 다음과 같다.

```r
seq(from = -2, to = 3, by = 1) # -2  3  1     , seq  sequence   .
```

```
## [1] -2 -1  0  1  2  3
```

```
rep(c(1,3), 3) # rep  repeat   .
```

```
## [1] 1 3 1 3 1 3
```

```
rep(c(1,3), each = 3)
```

```
## [1] 1 1 1 3 3 3
```

3.1.3 벡터 연산

```
x = c(1, 3, 5, 7); y = c(1, 3, 5); z = c(1, 3)
x + y #                  .
```

```
x = c(1, 3, 5, 7); y = c(1, 3, 5); z = c(1, 3)
x + z #          !          R
```

```
## [1]  2  6  6 10
```

3.1.4 벡터에 함수 적용하기

```
x = c(1,2,3,4)
sum(x); mean(x)
```

```
## [1] 10
```

```
## [1] 2.5
```

```
x = c(x, NA)
mean(x)
```

```
## [1] NA
```

```
mean(x, na.rm = T)
```

```
## [1] 2.5
```

```
x = c(1, 2, 3, NULL, 4)
x
```

```
## [1] 1 2 3 4
```

```
sum(x)
```

```
## [1] 10
```

3.2 행렬

3.2.1 행렬 생성

```
matrix(nrow = 2, ncol = 3)
```

```
##      [,1] [,2] [,3]
## [1,]   NA   NA   NA
## [2,]   NA   NA   NA
```

```
x = 1:12
matrix(x, nrow = 3, ncol = 4) # default : byrow = F
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
```

```
matrix(x, nrow = 3, ncol = 4, byrow = T)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    5    6    7    8
## [3,]    9   10   11   12
```

R은 컬럼벡터를 기본으로 한다고 생각하면 된다. 모든 입력이 세로로 들어가기 때문에(그리고 그게 더 빠름), 따로 방식을 지정하지 않으면 위에서부터 아래로 넣는다.

```
A = matrix(x, nrow = 3, ncol = 4)
length(A)
```

```
## [1] 12
```

```
dim(A)  # row - column
```

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

3.2.2 행렬연산

행렬의 연산자는 다양하다. 성분별로 연산을 하게 해주는 연산자로는 `+`, `-`, `*`, `/`, `^`, `sqrt()`, `log()`, `exp()`, `etc...` 가 있다. 그리고 진짜 행렬간의 연산을 해주는 연산자는 다음과 같다.

- %*% : 행렬곱
- solve() : 역행렬
- eigen() : 고유값&고유벡터
- t() : 전치행렬 생성

```r
A = matrix(1:4, 2, 2)
B = matrix(2:5, 2, 2)
A*B
```

```
##      [,1] [,2]
## [1,]    2   12
## [2,]    6   20
```

```r
A%*%B
```

```
##      [,1] [,2]
## [1,]   11   19
## [2,]   16   28
```

```r
t(A)
```

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

```r
eigen(A)
```

```
## eigen() decomposition
## $values
## [1]  5.3722813 -0.3722813
##
## $vectors
##            [,1]       [,2]
## [1,] -0.5657675 -0.9093767
## [2,] -0.8245648  0.4159736
```

3.2.3 행렬 인덱싱

행렬의 인덱싱은 데이터프레임과 유사하면서 차이가 있다. 확인해보기 전에 행과 열에 이름을 붙여보자.

```r
A = matrix(1:20, nrow = 5, ncol = 4)
rownames(A)
```

```
## NULL
```

```r
rownames(A) = c('a', 'b', 'c', 'd', 'e')
colnames(A) = c('a', 'b', 'c', 'd')
A[1:2, 3:4] #
```

```
##    c  d
## a 11 16
## b 12 17
```

```
A[1:2, c('a', 'c')]  #
```

```
##   a  c
## a 1 11
## b 2 12
```

3.2.4 apply

apply는 파이썬에서도 사용해봤을 것이다. 행렬안에서 행/열 별로 연산을 해준다. for로 억지 코딩하는 것보다 간편하면서 빠르다.

```
A = matrix(1:20, nrow = 5, ncol = 4); A
```

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

```
apply(A, 1, sum) # 1
```

```
## [1] 34 38 42 46 50
```

```
apply(A, 2, mean, na.rm = T) # 2
```

```
## [1]  3  8 13 18
```

```
#               ,                    .
```

3.2.5 rbind, cbind

행렬을 붙일 수도 있다. 열을 붙이면 cbind, 행을 붙이면 rbind이다.

```
x = c(1, 3, 5)
y = matrix(1:9, 3, 3)
rbind(x,y)
```

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

```
cbind(x,y)
```

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

행렬은 다음에 배울 데이터프레임과 유사하다. 행렬은 데이터프레임보다 다소 경직되어 있다고 볼 수 있다. 하지만 그런 경직때문에 계산측면에서는 데이터프레임보다 훨씬 빠르다. 연산에 대한 이슈가 발생할 경우, 데이터프레임은 모두 행렬 혹은 벡터로 변환해서 계산해주어야 편-안하다.

3.3 데이터프레임

데이터프레임은 행렬과 유사하지만 다소 다르다. 행렬은 수학적인 개념이라면, 데이터프레임은 조금 더 데이터를 담아두는 공간 같은 느낌이다. 그래서 행렬의 각 성분에는 character와 numeric이 혼재할 수 없지만, 데이터프레임 안에서는 가능하다.

3.3.1 데이터프레임 생성

```
name = c('Kim', 'Park', 'Lee')
age = c(25, 23, 21)
male = c(T, F, T)
X = data.frame(name, age, male, stringsAsFactors = F) # character  factor     . default  R 4.0  F
X
```

```
##   name age  male
## 1  Kim  25  TRUE
## 2 Park  23 FALSE
## 3  Lee  21  TRUE
```

```
cbind(name, age, male) #                .
```

```
##      name   age  male
## [1,] "Kim"  "25" "TRUE"
## [2,] "Park" "23" "FALSE"
## [3,] "Lee"  "21" "TRUE"
```

만들어진 데이터프레임을 확인해보자. 다음의 함수는 데이터를 확인할 때 자주 쓴다. 3회차에는 더 좋은 함수를 소개하겠다.

```
str(X)
```

```
## 'data.frame':    3 obs. of  3 variables:
##  $ name: chr  "Kim" "Park" "Lee"
##  $ age : num  25 23 21
##  $ male: logi  TRUE FALSE TRUE
```

```
summary(X)
```

```
##      name                age        male
##  Length:3           Min.   :21   Mode :logical
##  Class :character   1st Qu.:22   FALSE:1
##  Mode  :character   Median :23   TRUE :2
##                     Mean   :23
##                     3rd Qu.:24
##                     Max.   :25
```

```
dim(X)
```

```
## [1] 3 3
```

3.3.2 데이터프레임 인덱싱

데이터프레임의 인덱싱은 행렬과 유사하지만 다르다. 데이터프레임은 기본적으로 list의 하위개념이기 때문에…하지만 list자체를 우리가 다룰 일이 많지 않아서 제외했다. 아무튼 확인해보자.

```
X[1] #
```

```
##   name
## 1  Kim
## 2 Park
## 3  Lee
```

```
X[1:2] #
```

```
##   name age
## 1  Kim  25
## 2 Park  23
## 3  Lee  21
```

```
X[[1]] #
```

```
## [1] "Kim"  "Park" "Lee"
```

```
X[, 1:2]
```

```
##   name age
## 1  Kim  25
## 2 Park  23
## 3  Lee  21
```

```
X[1:2, ]
```

```
##   name age  male
## 1  Kim  25  TRUE
## 2 Park  23 FALSE
```

```
#              .

X$name #            .
```

```
## [1] "Kim"  "Park" "Lee"
```

```
X$name = c('Tim', 'Lee', 'Lim')
X
```

```
##   name age  male
## 1  Tim  25  TRUE
## 2  Lee  23 FALSE
## 3  Lim  21  TRUE
```

3.3.3 여러가지 활용

데이터를 다루다보면 결측값이 있기 마련이다. 이런 값을 대체하거나 삭제해보자.

```
X = data.frame(x1 = c(6, 3, NA, 3, NA), x2 = 1:5, x3 = 7:11)
complete.cases(X)
```

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

```
X[complete.cases(X), ] #
```

```
##   x1 x2 x3
## 1  6  1  7
## 2  3  2  8
## 4  3  4 10
```

```
X[is.na(X)] = 0; X #    0
```

```
##   x1 x2 x3
## 1  6  1  7
## 2  3  2  8
## 3  0  3  9
## 4  3  4 10
## 5  0  5 11
```

데이터에 접근해보자. 우리가 원하는 데이터만 얻고 싶어서 인덱싱을 통해 가져올 것이다.

```
data = mtcars
colnames(data)
```

```
##  [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"   "gear"
## [11] "carb"
```

```
#    mpg  25 , gear  4              ?
(data$mpg >= 25) & (data$gear >= 4)
```

```
##  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [13] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE
## [25] FALSE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE
```

```
data[(data$mpg >= 25) & (data$gear >= 4), ]
```

```
##                 mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Fiat 128       32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic    30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla 33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Fiat X1-9      27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
```

4 기본 함수와 플랏

R의 기본적인 함수정의와 루프문, 그리고 플랏을 살펴본다.

4.1 Function

파이썬에서는 `def`로 함수를 정의했는데, R에서는 `function`으로 정의한다.

```
average = function(x) {
  value = sum(x) / length(x)
  return(value) # return
}
y = c(1, 2, 3, 4, 5)
average(y)
```

```
## [1] 3
```

4.2 for loop

```
for (k in x) {do something}
```

- {} 안에서는 반복적인 작업을 수행
- 반복횟수 = length of x
- `break` : early stopping
- `next` : 다음 반복으로 넘어감

```
x = c(5, 6, 10, 7, 12)
z = NULL
for (k in x){
  z = c(z, k+3)
}
z
```

```
## [1]  8  9 13 10 15
```

```
z <- NULL
for (k in x) {
  z <- c(z, k + 3)
  if (k >= 10) break
}
z
```

```
## [1]  8  9 13
```

```
z <- NULL
for (k in x) {
  if (k >= 10) next
  z <- c(z, k + 3)
}
z
```

```
## [1]  8  9 10
```

4.3 while & repeat loop

`while (condition0) {do something}`

- {} 은 반복할 내용, condtion이 True일 때 까지
- `break` : early stopping

`repeat{}`

- `while`과 비슷한데, `repeat`은 종료와 관련한 조건이 없음
- 종료를 위해서는 `break`가 달려있어야함.

```
x <- 0
while (x < 5){
  x <- x + 1
}
x
```

```
## [1] 5
```

```
x <- 0
while (x < 5){
  x <- x + 1
  if (x > 3) break
}
x
```

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

```
x <- 0
repeat {
  x <- x + 1
  if (x >= 5) break
}
x
```

```
## [1] 5
```

4.4 ifelse

ifelse는 정말 많이 쓰인다. 보통 데이터 전처리할때 재범주화하는 경우 많이 사용했다. 예를 들어 나이가 13, 26, 80, 50, ··· 이렇게 있는데, 이를 청소년, 청년, 장년, 중년, 노년 등으로 재범주화할 때 사용하기 편하다.

먼저 if문 먼저 보고, ifelse를 확인하자.

```
r <- 3
if (r==4) {
  x <- 1
} else {
  x <- 3
  y <- 5
}
x
```

```
## [1] 3
```

```
y
```

```
## [1] 5
```

```
r <- 3
if (r==4) {
  x <- 1
} else if (r == 3) {
  x <- 3
  y <- 5
} else {
  x = 7
}
x
```

```
## [1] 3
```

```
y
```

```
## [1] 5
```

ifelse를 확인하자.

```r
ifelse(c(3, 5, 7, 9, 11) > 3, 'a', 'b')
```

```
## [1] "b" "a" "a" "a" "a"
```

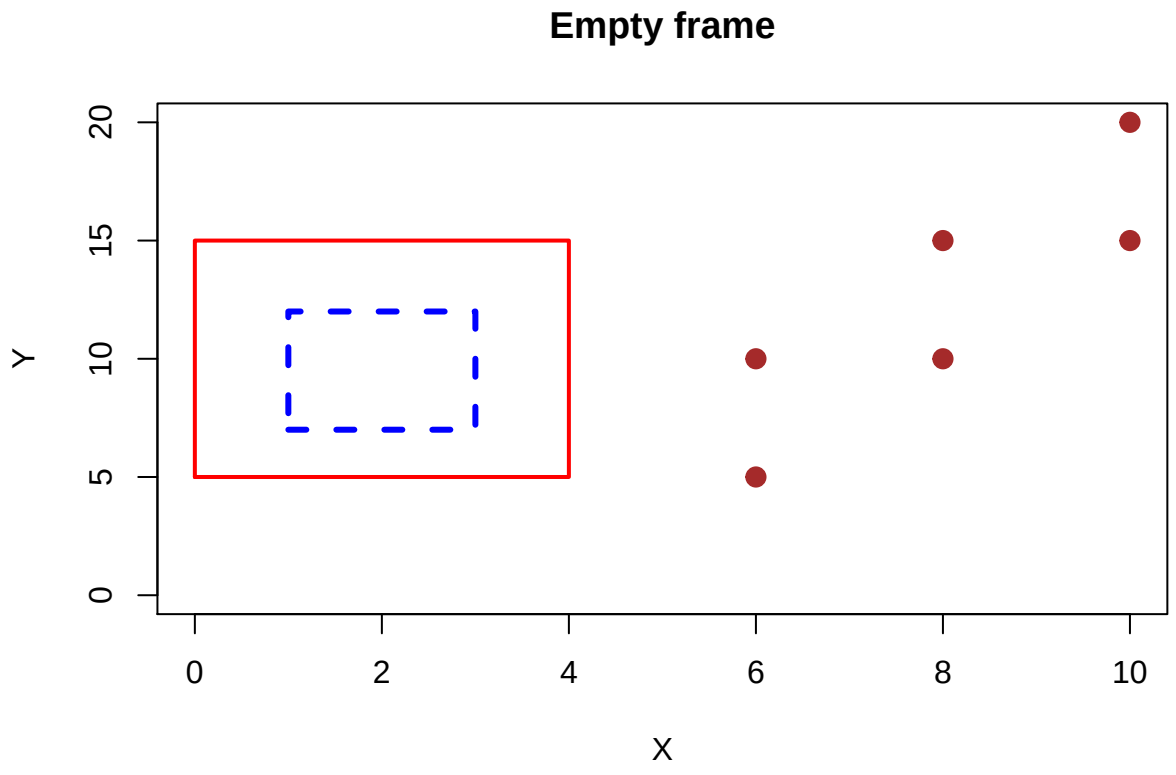
```r
ifelse(c(3, 5, 7, 9, 11) > 7, 'a', ifelse(c(3, 5, 7, 9, 11) > 4, 'b', 'c'))
```

```
## [1] "c" "b" "b" "a" "a"
```

4.5 Plot

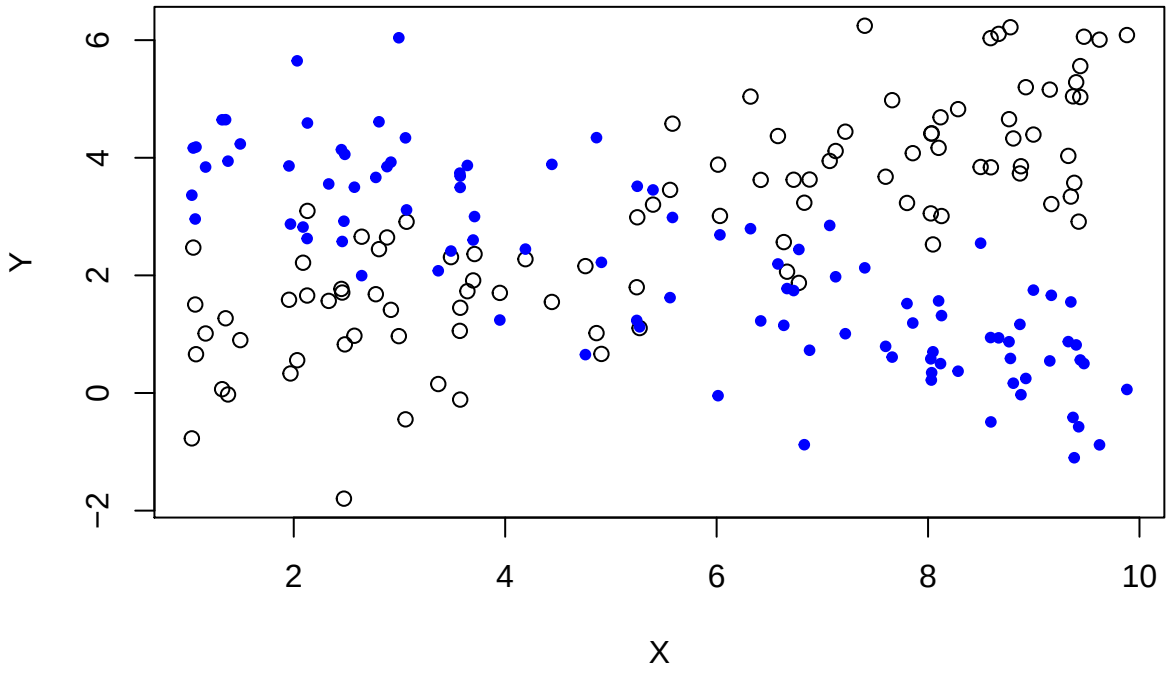
plot 함수를 통해 그래프를 그리고 시각화를 할 수 있다.

```r
plot(c(0, 10), c(0, 20), type = "n", xlab = "X", ylab = "Y",
     main = "Empty frame")
points(c(6, 8, 10, 6, 8, 10), c(5, 10, 15, 10, 15, 20),
       col = "brown", cex = 2, pch = 20)
lines(c(0, 4, 4, 0, 0), c(5, 5, 15, 15, 5), col = "red", lwd = 2)
lines(c(1, 3, 3, 1, 1), c(7, 7, 12, 12, 7), col = "blue", lwd = 3, lty = 2)
```



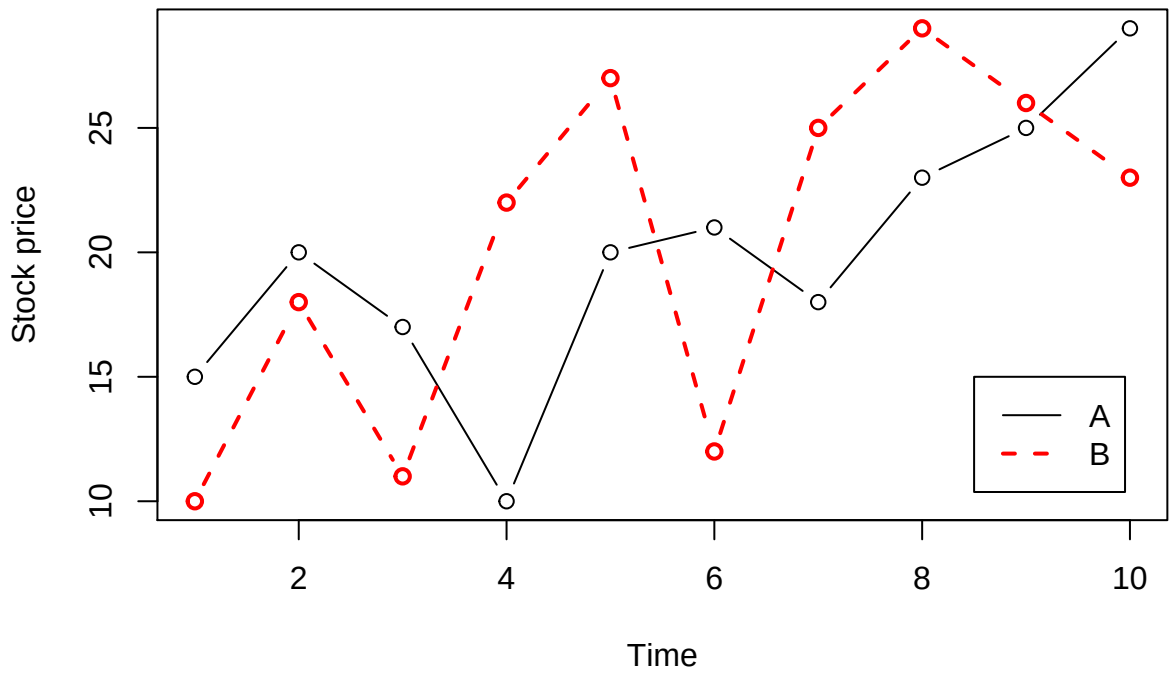
산점도를 그려보자

```r
n <- 100
x <- runif(n, 1, 10)
y1 <- 0.5 * x + rnorm(n)
y2 <- 5 - 0.5 * x + rnorm(n)
plot(x, y1, xlab = "X", ylab = "Y")
points(x, y2, col = "blue", pch = 20)
```



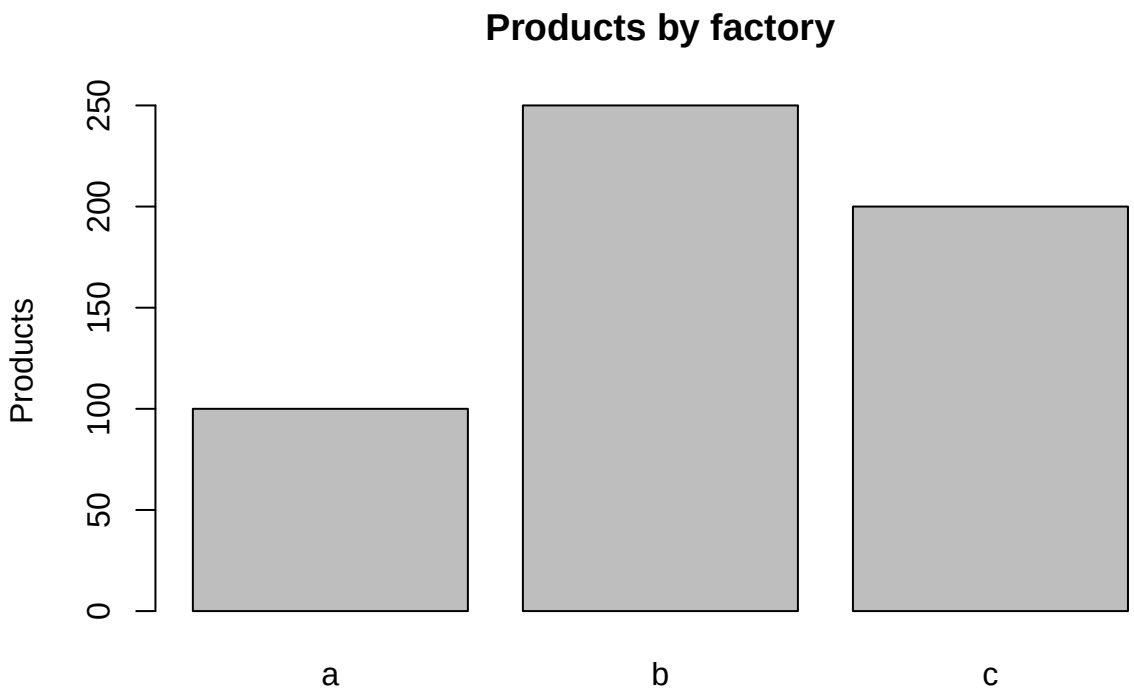
선 그래프를 그려보자

```r
Time <- 1:10
stock1 <- c(15, 20, 17, 10, 20, 21, 18, 23, 25, 29)
stock2 <- c(10, 18, 11, 22, 27, 12, 25, 29, 26, 23)
plot(Time, stock1, type = "b", xlab = "Time", ylab = "Stock price")
lines(Time, stock2, type = "b", col = "red", lwd = 2, lty = 2)
legend(8.5, 15, c("A", "B"), col = c("black", "red"),
       lty = c(1, 2), lwd = c(1, 2))
```



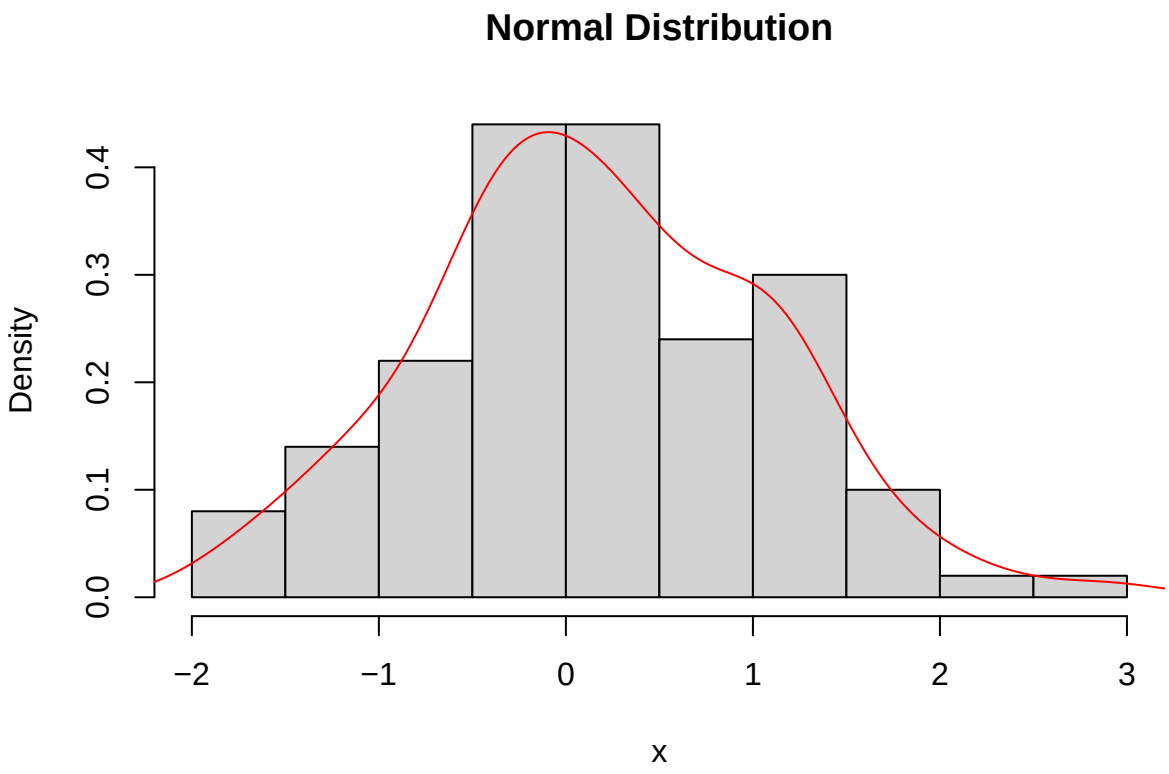
막대그래프를 그려보자

```
x <- c(100, 250, 200)
barplot(x, names.arg = c("a", "b", "c"), ylab = "Products", main = "Products by factory")
```



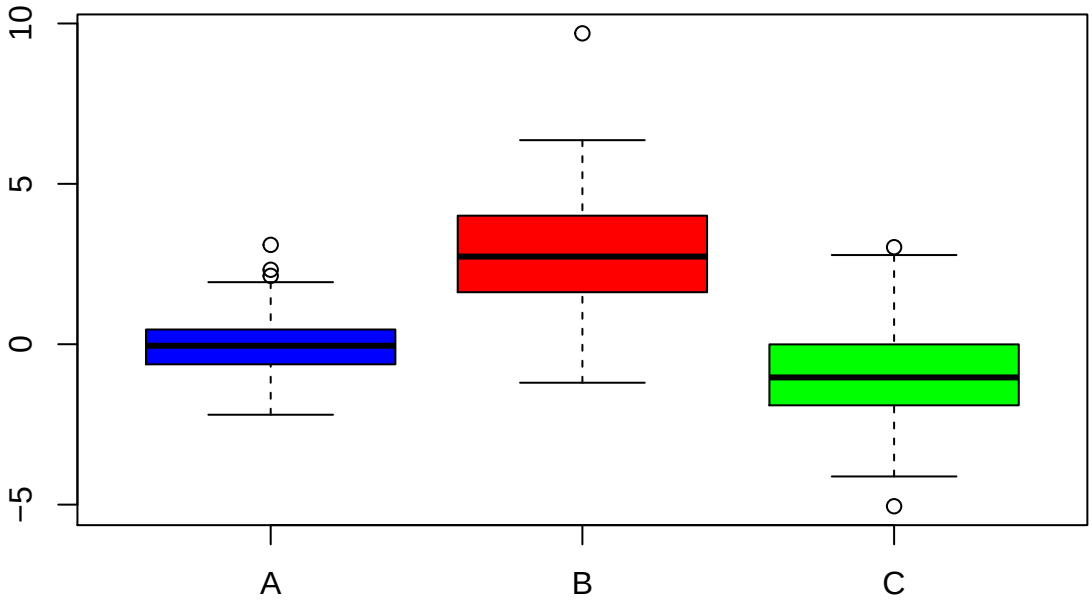
히스토그램을 그려보자

```
x <- rnorm(100)
hist(x, main = "Normal Distribution", freq = F)
lines(density(x), col = "red")
```



박스플랏을 그려보자

```
x <- rnorm(100)
y <- rnorm(100, mean = 3, sd = 2)
z <- rnorm(100, mean = -1, sd = 1.5)
boxplot(x, y, z, names = c("A", "B", "C"), col = c("blue", "red", "green"))
```



하지만 이런 기본 플랏들은 잘 사용하지 않는다. 이런 시각화는 대충 확인해보는 의미도 있지만, 다른 사람들에게 예쁘게 보여주려는 목적도 있기 때문에, 기본 플랏은 문법은 정말 단순하지만 예쁘지가 않다. 그래서 R에서 많이 쓰이는 그래픽 패키지는 `ggplot2`로 `grammar of graphics`라는 엄청난 이름을 가진 패키지를 사용하게 된다. 파이썬의 `matplotlib, seaborn`보다 훨씬 문법적으로도 간편하고, 플랏에 소소한 기능들을 추가하는 것이 상대적으로 매우 쉽다.

하지만 ggplot은 따로 다룰 시간이 없기 때문에… 관심있으신 분들은 `ggplot2 examples`라고 구글에 치면 정말 많은 예제가 나옵니다!

5 Statistics

5.1 기본함수

- exp(): Exponential function with base e.
- log(): Natural log.
- log10(): Logarithm with base 10.
- sqrt(): square root.
- abs(): absolute value.
- sin(), cos(), tan(): Trigonometric functions.
- min(), max(): Minimum / maximum value within a vector.
- which.min(), which.max(): Index of the minimal / maximal element of a vector.

```
x <- c(6, 3, 4, 1, 5)
min(x)
```

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

```
max(x)
```

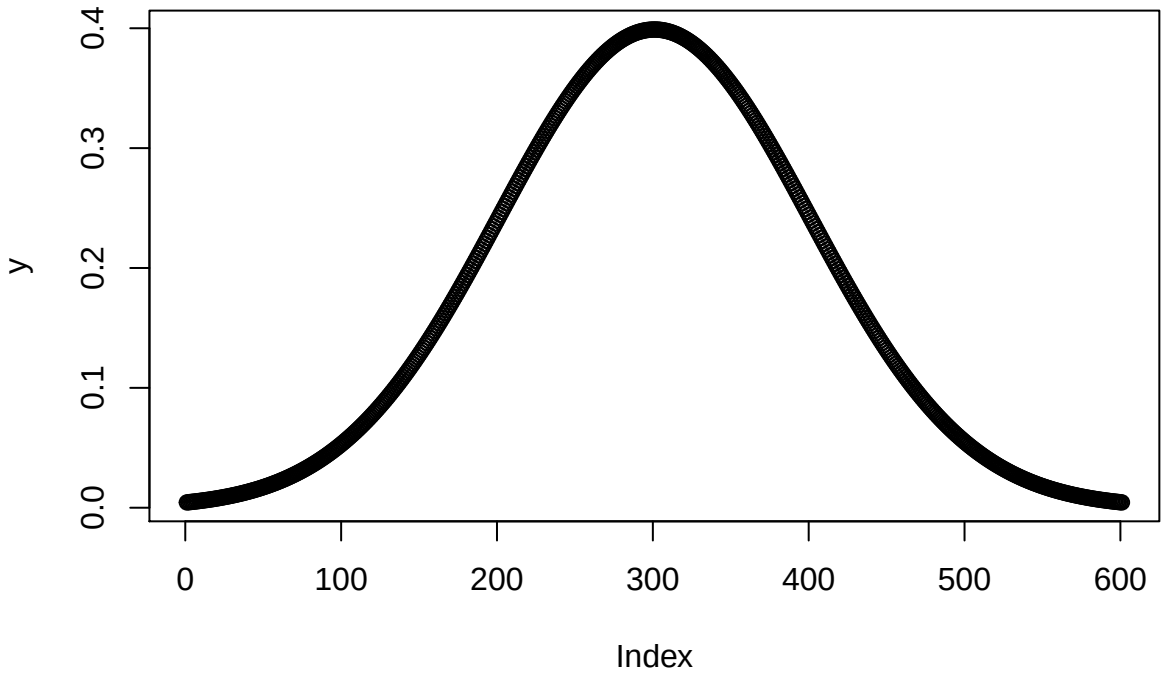
```
## [1] 6
```

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

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

5.2 확률변수

```
x <- seq(from = -3, to = 3, by = 0.01)
y <- dnorm(x, mean = 0, sd = 1, log = FALSE)
plot(y)
```




```
x <- rbinom(2000, size = 50, prob = 1/5)
mean(x)
```

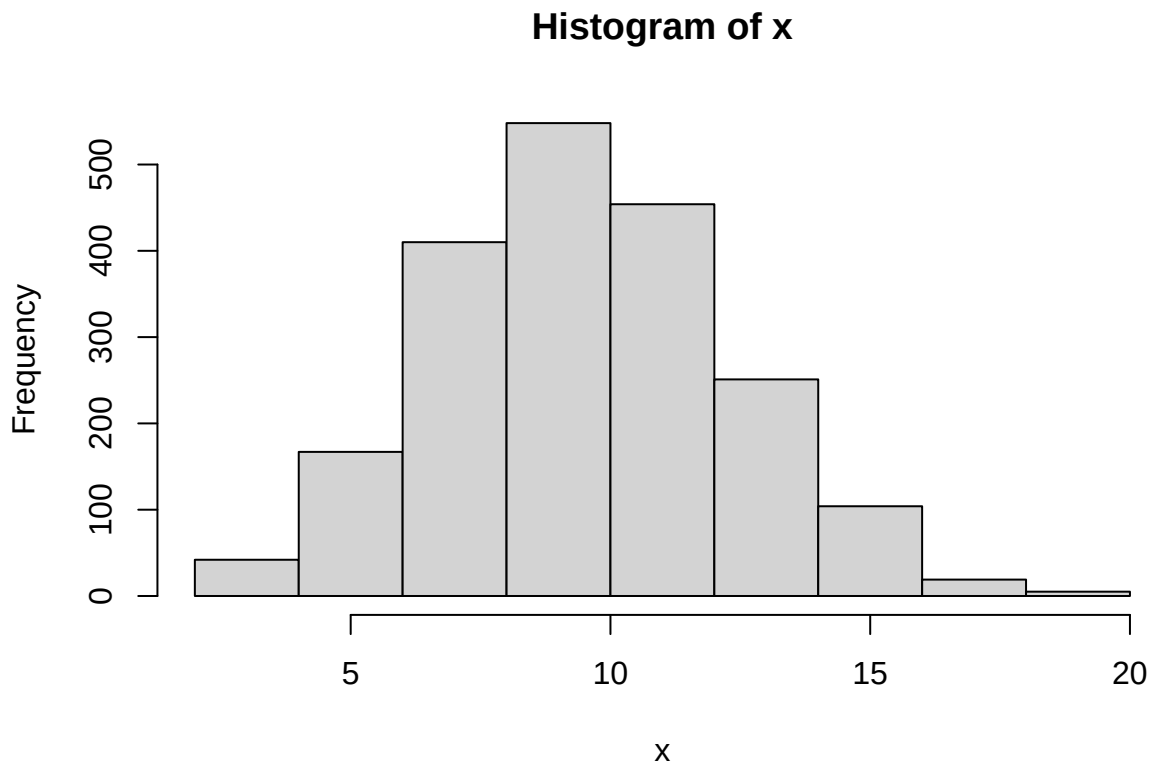
```
## [1] 9.998
```

Functions for Statistical Distributions

	pdf/pmf	cdf	quantile	random #
Uniform	`dunif()`	`punif()`	`qunif()`	`runif()`
Normal	`dnorm()`	`pnorm()`	`qnorm()`	`rnorm()`
Exponential	`dexp()`	`pexp()`	`qexp()`	`rexp()`
Binomial	`dbinom()`	`pbinom()`	`qbinom()`	`rbinom()`
t	`dt()`	`pt()`	`qt()`	`rt()`
Chi-square	`dchisq()`	`pchisq()`	`qchisq()`	`rchisq()`

Figure 1: Functions for Statistical Distribution

```
hist(x)
```




```
pbinom(2, 10, 0.3) - pbinom(1, 10, 0.3) # P(X <= 2) - P(X <= 1) == P(X == 2)
```

```
## [1] 0.2334744
```

```
qnorm(0.975, 0, 1)
```

```
## [1] 1.959964
```

```r
pnorm(1.645, 0, 1)
```

```
## [1] 0.9500151
```

```r
knitr::purl('Week 2.rmd','Week 2.R')
```

```
## [1] "Week 2.R"
```