**자료는 보기 편하게 하기 위하여 R 팩키지에서 수행되는 모양으로  
제공합니다. 실제 사용하실 때는 명령어 앞에 있는 > 와 + 표시를**

**빼고 사용하시면 됩니다**

**Chapter 1**

**04 기본적인 사용 방법**

> # 주석은 #을 앞에 놓고 쓰면 된다

> # [R을 계산기처럼 사용하기]

> 1+2 # 계산할 것을 지시하는 부분

> (11+54-13)\*76/9

> # [ R의 연산자 목록 ]

> # + - \* / : 기본 연산자

> # %/% : 정수 나눗셈

> # %% : 나머지

> # \*\* : 거듭제곱

>

> 9 %% 3 # 9를 3으로 나누면 나머지는 0 이다

> 4 %/% 2

>4\*\*2 #4\*4는 16이다

> # [ R에서 지원하는 수학함수 목록 ]

> # sin(x) : sin x ==> cos(x), tan(x),

> # sinh(x) : sinh x ==> cosh(x), tanh(x),

> # asin(x) : sin-1 x ==> acos(x), atan(x), asinh(x), acosh(x), atanh(x)

> # log(x) : 로그

> # log10(x) : 상용로그

> # log2(x) : 밑이 2인 로그

> # log1p(x) : log(1+x)

> # exp(x) : ex ==> expm1(x) : ex-1

> # sqrt(x) : 루트

> # trunk(x) : 소수점이하 버림표

> # abs(x) : x의 절대값

> # runif(n) : 0과 1사이의 난수 n개 생성

> # floor(x) : x 보다 작은 최대 정수

> # Round(x, digits=0) : x값을 반올림한 정수

>

> sqrt(5) #루트 5를 의미하는 함수

> # [이전 명령어의 재호출]

> # 위의 화살표키를 누르면 앞에서 수행했던 명령어들을 다시 수행할 수 있다

> # 이때, 커서를 움직여서 일부를 수정해서 수행할 수 있다

>

> # [R의 종료] > quit() >

======================================================

> rm(w) # 변수 w를 삭제합니다

> rm(list=ls()) # 현재 시스템에서 사용 중인 모든 변수를 지운다

> setwd("C:/study/R")

> ls() # 현재 시스템에서 사용되는 변수의 리스트를 보여준다(또는, >object())

> rm(list=ls()) # 현재 시스템에서 사용 중인 모든 변수를 지운다

> help("for") # for 명령어에 대한 도움말을 보여준다

> is() # 현재 시스템에서 사용되는 변수의 리스트를 보여준다

> bb # is()에서 보여준 변수 중의 하나인 bb 변수의 내용을 화면에 보여준다

> cat(" I love you \n") # 문자열을 표시하고, 개행(\n)을 실행한다(\t도 가능)

>sink("output.txt") # 저장할 파일명을 입력한다(저장될 경로명은 사전에 정의한다)

> cat(" I love you \n") # 출력 결과 화면이 아닌 파일로 저장된다

> sink() # 원래 위치로 돌아간다

> head(iris) # R이 제공하는 기본 데이터 iris의 앞부분을 보여준다

> tail(iris) # iris의 뒷부분을 보여준다

> view(iris) # iris 데이터를 별도의 창에 보여준다

> summary(iris) # iris 데이터의 기본 통계량(최대,최소,평균...)을 보여준다

> getwd() # 현재 사용하는 디렉터리 정보 [1] "C:/Users/home/Documents"

> cat("This is a test Message. \n") # 화면에 메시지 표시하기 This is a test Message.

> sink("output.txt") # 이후에 발생하는 모든 출력을 화면 외에 output.txt에 저장한다

> cat("This is a sink test. \n") # 메시지가 output.txt 파일로 보내짐

>

> head(iris) #iris 데이터의 앞부분을 보여주어야 하는데, output.txt로 보내짐

> sink() #파일로 보내지는 것을 풀고, 다시 화면에 보여지도록 함

> head(iris)

> ls() character(0)

> summary(iris) # iris 데이터의 통계량을 정리해서 보여줌

>

**05 기본적인 데이터 타입**

> x <- c(1,2,3,4,5) # 벡터형의 자료를 생성해서 변수 x에 할당한다

> x # x의 값을 확인한다 [1] 1 2 3 4 5

> xy <- rnorm(30) # 30개의 정규분포를 하는 난수를 생성해서 x에 할당한다

> xy # xy의 값을 확인한다

======================================================

> x <- c(1,2,3,4,5) # 벡터형의 자료를 생성해서 변수 x에 할당한다

> x # x의 값을 확인한다

> mean(x) # 평균을 구한다

> order(x) # 커지는 순서대로 배열한다

> rev(x) # 역순(작아지는 순서)으로 배열한다

> range(x) # x값의 범위를 구한다

> sd(x) # x값의 표준편차를 구한다

> sort(x) # x값을 커지는 순서대로 정렬한다

> sort(x, decreasing=TRUE) # x값을 작아지는 순서대로 정렬한다

> length(x) # x의 길이를 구한다

===================================================

> x <- c(1,4,6,8,9) # 벡터형의 자료를 생성해서 변수 x에 할당한다

> x # x의 값을 확인한다

> x[2] # x의 두 번째 값을 구한다

> x[-2] # x의 두 번째 값을 빼고 나머지를 구한다

> x[3] <- 4 # x의 세 번째 값을 4로 바꾼다

> x # x의 값을 확인한다

====================================================

> x[2<x & x<5] # x의 값이 2보다 크고, 5보다 작은 것을 구한다

> names(x) <- c("first","second","third","forth","fifth")

> x["third"]

> # 벡터형 자료의 일부를 바꾸는 법

>

> x <- c(1,4,6,8,9) # 벡터형의 자료를 생성해서 변수 x에 할당한다

> x # x의 값을 확인한다

> y <- replace(x, c(2,4), c(32,24)) # x의 2, 4번째 자료를 32, 24로 바꿔라

> y

> w <- append(x,y) # x에 y를 합친다

> w

> z <- append(x,y, after=2) # x의 2번째 다음에 y를 합친다

> z

> # 벡터형 자료의 연산 기능

> c(1,2)+c(4,5) # 1+4, 2+5를 수행한다

> c(1,2,3)+1 # c에 1을 더한다

====================================

> > vector <- -5:5 # -5에서 5까지 정수를 생성하여 할당한다

> vector

> q <-seq(from=1, to=5, by=0.5) # 1부터 5까지 0.5 단위로 생성하여 할당한다

> q

> qq <- seq(10) # 1부터 10까지 생성하여 할당

> qq

==============================================

> x <-c(1,2,3)

> y <-c(4,2,8)

> x==y # x값과 y값이 같은지 순서에 따라 비교한다

[1] FALSE TRUE FALSE

>

> # 벡터의 집합 연산

> # union(x,y) # 합집합

> # intersect(x,y) # 곱집합

> # setdiff(x,y) # 차집합

> # setequal(x,y) # 같은 집합인가?

> # is.element(a,x) # a는 x에 포함되는가?

>

> (x <-c(sort(sample(1:99, 9)))) # 1~99까지 9개를 뽑고, 정렬할 것

> (y <-c(sample(3:60, 7))) # 3에서 60까지 7개를 뽑을 것

> union(x,y) # x와 y를 합친 것

> intersect(x,y) # x와 y의 공통적인 것

> setdiff(x,y) # x와 y의 다른 것

[1] 12 29 56 61 66 71 85 95

=========================================

> # 벡터에 문자 자료를 가지는 경우

>

> (x <- rep(c("a","b","c"), times=4)) # a,b,c,d를 4번 반복하여 x를 생성

> unique(x) # 반복된 값을 제거하여 보여주고, 데이터는 유지

> match(x, c("a")) # 벡터에서 a가 있으면 1을 보여준다

> # 문자열 조작을 위한 함수 : 무지하게 중요 !!!!

>

> # chamatch() : 문자열 부분 매칭

> # chrtr() : 문자열 치환

> # grep() : 정규표현식을 이용한 패턴 매칭

> # nchar() : 문자수

> # paste() : 문자열 결합

> # substr() : 부분 문자열 추출

> # strsplit() : 문자열 분할

> # tolower() : 소문자로 치환, 반대는 toupper()

>

> xx <- c("a","b","c","d","e") # 문자 자료를 가지는 벡터의 생성

> xx # 생성 확인

> k=paste(xx[1], xx[2]) # 벡터 xx의 첫 번째와 두 번째를 연결해서 k에 할당

> k

> paste("I love ", "you and ", "you love ", "me!", sep="$") # 문자를 결합한다

> paste(x, collapse="%") # 문자 사이에 %를 넣는다

> paste(x, collapse="") # 문자 사이에 공백을 넣는다

> substring("abcdefghijklmn", 2:5) # 2~5, 3~5, 4~5, 5~5까지 문자열 생성

> substring("abcdefghijklmn", 2,5) # 2에서 5까지의 문자열 뽑아내기

> name <- c("Jungwon", "University", "Computer","Science","Major", ↲ + "Communication")

> grep('Co', name) # name에서 Co로 시작하는 것

> grep('(om)',name) # name에서 om을 가지고 있는 것

===================================================

> #논리형 벡터 : 논리값을 요소로 하는 벡터

>

> x <-runif(5) # 0~1 사이의 값을 5개 생성

> x

> (0.4 <= x) & (x <=0.7) # x가 0.4~ 0.7 사이에 있는가?

> any (x>0.9) # x 중에 0.9 이상이 있는가?

> all (x < 0.9) # x의 값이 모두 0.9 이하인가?

>

> is.vector(x) # x가 벡터형의 데이터인가 ?

> ## rep와 gl에 대한 보충 설명 : 많이 사용되는 기능입니다

>

> rep (9, 5) # 9를 5번 반복하라

> rep (1:4, 2) # 1~4를 2번 반복하라

> rep (1:4, each=2) # 1을 2번, 2를 2번, 3을 2번, 4를 2번하라

> gl (4,3 ) # 1을 3번, 2를 3번, 3을 3번, 4를 3번하라

================================================

> # 행렬은 2차원의 자료형을 관리하는 방식이다

>

> # 행렬의 생성을 위해 3개의 벡터 데이터를 생성하고, 이것을 합친다

>

> vec1 <- c(1,2,3) # 배열을 생성해서 vec1에 할당

> vec1

> vec2 <- c(4,5,6) # 배열을 생성해서 vec2에 할당

> vec2

> vec3 <- c(7,8,9) # 배열을 생성해서 vec3에 할당

> vec3

> # 배열 3개를 묶어서 행렬을 생성한다

> # rbind와 cbind의 차이를 확인할 것 !!

>

> mat1 <- rbind(vec1, vec2, vec3)

> mat1 # 행렬의 모습을 확인하고, rbind의 역할을 확인

> (mat2 <- cbind(vec1, vec2, vec3)) #()로 묶으면 바로 결과 출력

>

> # 행렬 데이터의 조작

> apply(mat1, 1, max) # mat1의 세로(=1) 중에서 가장 큰 값을 출력

> apply(mat1, 2, max) # mat1의 가로(=2) 중에서 가장 큰 값을 출력

> colnames(mat1) <-c("A", "B", "C") # 행렬 데이터 컬럼에 이름 할당

> mat1 # 결과 확인

=========================================

> # 문자형 행렬 데이터 생성

> chars <- c("a","b","c","d","e","f","g","h","i","j")

> # 아래에 벡터형 데이터를 기반으로 3가지 타입의 행렬 데이터를 만든다

> mat11 <- matrix(chars)

>

> mat22 <- matrix(chars, nrow=5)

> mat22

> mat33 <- matrix(chars, ncol=5)

> mat33

>

> # 행렬 데이터의 조작

> x1 <- matrix(1:8, nrow=2) # 1에서 8까지의 숫자를 생성하고 row는 2이다

> x1 # 생성된 데이터의 형태를 확인해 본다

> x1\*3 ; x1 # 행렬 x1에 3을 곱한다. 결과를

> x1\*c(10, 20); x1 # 행렬 x1에 10과, 20을 각각 곱한다

============================================

> # 실습을 위한 행렬을 matrix를 이용하여 생성한다

>

> x <- matrix(1:12, nrow=3, dimnames=list(c("R1","R2","R3"), ↲ + c("C1","C2","C3","C4")))

> x # x 값을 확인

> x[7] # x 7번째 값을 확인

> x[1, ] # x의 첫번째 행의 값을 확인

> x[, 2:4] # x의 2번째에서 4번째 열의 값을 출력

> x[,-2] # x의 2번째 열의 값을 빼고 출력

> # 연습문제 : 아래의 예를 실행하여 결과를 확인해 본다

> x <- matrix(c(1,2,3,4,5,6), nrow=2)

> colnames(x) <- c("First","Second","Third")

> x

=================================================

> # 배열 데이터를 선언한다

> no <- c(1,2,3,4)

> name <- c("Apple", "Banana", "Peach", "Berry")

> prices <- c(500, 200, 200,50)

> qty <- c(5,2,7,9)

> # 선언된 배열 데이터(다른 형태도 좋다)를 모아서 행렬을 구성한다

> # 이것을 데이터 프레임이라고 한다

> fruit <-data.frame(No=no, Name=name, PRICE=prices, QTY=qty)

> fruit

> # 데이터 프레임 데이터의 조작

> fruit[1,] # 1번 행만 출력할 것

> fruit[, -2] #2번 열을 제외하고 출력할 것

==============================================

> # 리스트형의 자료 생성 및 모양의 확인

> member <- list(name="minho", address="seoul", tel="2345", pay=300)

> member

> # 리스트형 자료의 조작 방법(데이터 삽입 확인)

> member$name # name만 확인

> member[1:3] # 1번에서 3번까지의 자료만 확인

> member$birth <-"1975-10-23" # 새로운 데이터의 삽입

> member # 자료가 입력된 것을 확인

=========================================

># 실습을 위하여 작업할 디렉터리를 설정하고, 실습에 사용할 데이터를 읽어서 score에 저장한다

> setwd("c:/Study/R") # 작업 디렉터리 설정

> score <- read.csv("top50.csv", sep=",",header=TRUE) # 설정된 곳에서 데이터를 읽어옴

> head(score, 5) # 읽어온 데이터 중에서 앞의 5줄만 화면에 보여줌

> dim(score) # 읽어온 자료의 행과 열의 숫자를 보여줌

> # 읽은 자료를 편집하겠다는 명령으로 별도의 윈도우에 자료가 보여지고, 편집이 가능

> edit(score)

> summary(score$SCI) # score 데이터의 SCI열에 대한 기본 통계량을 보여줌

> mscore <- score[1:2,] # score의 1,2 행을 분리하여 mscore에 저장

> mscore

> mscore2 <-score[3:6,] # score의 3,4,5,6 행을 분리하여 mscore2에 저장

> mscore2

> newscore <-rbind(mscore,mscore2) # 행으로 2개의 자료 결합하여 newscore에 저장

> newscore

> mscore3 <-score[, 4:7] # score를 4,5,6,7열을 분리하여 mscore3에 저장

> mscore3

> newscore2 <-cbind(mscore3,mscore4) # mscore3, mscore4를 열을 중심으로 합병

> newscore2

============================================

># 많이 사용되는 apply 명령어 실습이다

># 자료형 준비

>x <-array(c(5,6,7,8,9,10), dim=c(2,3))

>x

>mean(x[2,]) # x의 2번째 행의 평균을 구한다

>apply(x, 1, sum) # x의 행(=1)에 대한 합을 구한다

>rowSums(x) # 위와 동일한 효과

>apply(x, 2, sum) # x의 열(=2)에 대한 합을 구한다

>colSums(x) # 위와 동일한 효과

>apply(x, 1, mean) # x의 행에 대한 평균을 구한다 = rowMean(x)

>apply(x, 2, mean) # x의 열에 대한 평균을 구한다 = colMean(x)

============================================

> x <-array(1:24, dim=c(2,4,3)) # 배열의 생성

> x

> x[1,3,] # 3개 행렬의 1번 행의 값 중에서 3번째 행의 값을 표시

> x[2,4,3] # 생각해 보자 !!!

> mean(x[1,,]) # 1번 행 값들의 평균

> mean(x[,2,]) # 생각해 보자 !!

> mean(x[1,2,])

**06 데이터의 저장하기와 불러오기**

> setwd("c:/temp") # 작업할 디렉터리의 설정

> ls() # 현재 시스템에 선언되어 있는 변수의 리스트를 보여준다

> rm(list=ls()) # 현재 시스템에 선언되어 있는 변수를 모두 지운다

> ls() # 모든 변수가 지워졌음을 확인한다 character(0)

> # 벡터(Vector) 형식의 데이터를 선언한다.

> no <- c(1,2,3,4)

> name <- c("Apple", "Banana", "Peach", "Berry")

> prices <- c(500, 200, 200,50)

> qty <- c(5,2,7,9)

>

> # 벡터 형식의 데이터를 모아서 데이터 프레임 형식의 데이터를 구성한다

> fruit <-data.frame(No=no, Name=name, PRICE=prices, QTY=qty)

> fruit # fruit 형식의 데이터 모습을 확인한다

>

> # 데이터를 저장하고 꺼내는 과정 수행

> ls() # 현재 시스템에 선언되어 있는 변수의 리스트를 보여준다

> save(no, name, fruit, file="test.data") # no,name,fruit을 test.data 파일에 저장

> rm(fruit, no, name) # 선언된 변수를 삭제

> ls() # no,name,fruit 변수가 지워졌음을 확인

> load("test.data") # 변수를 읽어옴

> ls() # 변수가 읽어졌음을 확인

> no # 변수 확인

> name # 변수 확인

>fruits

===============================================

> # 외부(예:Excel)에서 만들어진 데이터를 읽어오는 것을 실습한다

> setwd("c:/temp") # 작업할 디렉터리의 설정

> ls() # 현재 시스템에 선언되어 있는 변수의 리스트를 보여준다

> rm(list=ls()) # 현재 시스템에 선언되어 있는 변수를 모두 지운다

> ls() # 모든 변수가 지워졌음을 확인한다 character(0)

> # csv형(=콤마로 구분된 자료형) 데이터를 읽어온다

> x <- read.csv("a.csv")

> x # 읽은 데이터를 확인한다

> x[1,] # 읽은 데이터에서 첫 번째 행을 보여준다

> str(x) # 읽은 데이터의 형식을 보여준다.

> x$name <- as.character(x$name) # name을 chr 형으로 변환한다

> str(x) # 변환한 것을 확인한다

> # 만약, csv 파일에 헤더가 없는 경우가 있다면, 파일을 읽은 후에 헤더를 따로 넣는다

> (xx <-read.csv("b.csv")) # 헤더없는 csv 파일을 읽는다

> names(xx) <- c("id", "name", "score") # 헤더를 넣는다

> xx

> # R에서 작업을 위하여 데이터를 만든다

> no <- c(1,2,3,4)

> name <- c("Apple", "Banana", "Peach", "Berry")

> prices <- c(500, 200, 200,50)

> qty <- c(5,2,7,9)

> # 벡터 형식의 데이터를 모아서 데이터 프레임 형식의 데이터를 구성한다

> fruit <-data.frame(No=no, Name=name, PRICE=prices, QTY=qty)

> fruit # fruit 형식의 데이터 모습을 확인한다

> # 데이터를 csv 형식으로 저장하고 꺼내는 과정을 수행한다

> ls() # 현재 시스템에 선언되어 있는 변수의 리스트를 보여준다

> write.csv(fruit, file="fruit.csv") # 데이터를 csv 형으로 저장한다

> rm(fruit) # fruit 자료를 지운다

> ls() # fruit 변수가 지워졌음을 확인

> fruit <- read.csv("fruit.csv") # 변수를 읽어옴

> ls() # 변수가 읽어졌음을 확인

================================================

> # 이번에는 일반 편집기로 만들어진 txt 형태의 데이터를 R로 읽는 방법이다

> # 기본적으로 테스트 데이터는 testdata.txt의 이름으로 만들어져 있다

> setwd("c:/temp") # 작업할 디렉터리 설정

> ls() # 현재 시스템에 선언되어 있는 변수의 리스트를 보여준다

> rm(list=ls()) # 현재 시스템에 선언되어 있는 변수를 모두 지운다

> ls() # 모든 변수가 지워졌음을 확인한다

> a <- scan("testdata.txt", what="")# 데이터를 읽어들인다

> a # 데이터를 배열 형태로 읽은 것을 확인한다

> a[4] # 4번째 데이터를 화면에 보여준다

> str(a) # a의 자료들의 형을 보여준다

> # 데이터를 배열 형태가 아닌 원래의 형태인 행렬(Matrix)모양으로 읽는 방법이다

> c <- read.table("testdata.txt", header=T) # 원 데이터의 타이틀을 살려서 읽는다

> c # 읽은 데이터의 모양을 보여준다

=============================================

> vec1 <-c(1,2,3)

> vec2 <-c(4,5,6)

> vec3 <-c(7,8,9)

> mat <-rbind(vec1, vec2, vec3)

> mat

> setwd("c:/temp")

> save(mat,file="testfile2.data")

> datafile <-load("testfile2.data")

> datafile [1] "mat"

===============================================

> # 정규식의 연습을 위하여 필요한 샘플 데이터를 만든다

> text <- c("Game", "GAME", "game", "gAME", "Tetris1", "game", "tetris5")

> text # 데이터의 확인

> # 지금부터는 정규식을 이용하는 예를 제시한다

> grep("game", text) # text에서 game을 찾아서 몇 번째에 있는지 보여준다

> grep("game", text, value=TRUE) # text에서 game을 찾아서 내용을 보여준다

> # text에서 첫 문자가 g로 시작되는 것을 찾아서 내용을 보여준다

> grep("^g+", text, value=TRUE)

> # text에서 첫 문자가 G로 시작되는 것을 찾아서 내용을 보여준다

> grep("G+", text, value=TRUE)

> # text에서 첫 문자가 ME로 끝나는 것을 찾아서 내용을 보여준다

> grep("ME$",text, value=TRUE)

> grep("[2-5]", text, value=TRUE) # text에서 2,3,4,5를 포함한 것을 찾아서 내용을 보여준다

> grep("[[:digit:]]", text, value=TRUE) # text에서 숫자를 포함한 것을 찾아서 내용을 보여준다

> grep("[[:upper:]]", text, value=TRUE) # text에서 대문자를 포함한 것을 찾아서 내용을 보여준다

> nchar(text) # 글자수를 보여준다

> x <- "abcd efgh" > nchar(x)

> y <- "xyz"

> paste(x,y) # x와 y를 합친다

> substr(x, 6,8) # x에서 6~8번째의 글자를 보여준다

> strsplit(x, split="c") # x에서 c를 기준으로 분리한다

**Chapter 2**

**01 개요**

**02 함수**

> # R에서 프로그램 기능을 사용하려면

> # 함수기능과 순환 기능 그리고 이것을 순차적으로 수행할 수 있어야 한다

> # 함수의 정의 및 활용 >

> # [1단계] 인수가 없는 함수의 선언

> minho <- function() {

+ x <-10

+ y <-20

+ return(x\*y) # 돌려주는 값의 선언

+ }

>

> # [2단계] 인수가 있는 함수의 선언

> minho2 <-function(x,y) {

+ xx <-x

+ yy <-y

+ return(sum(xx, yy)) # 시스템이 정의한 특정 함수를 이용한 결과를 돌려줌

+ }

> # [3단계] 함수에서 함수를 부르는 경우

> minho3 <-function(x,y) {

+ x3 <-x+1

+ y3 <-y+1

+ x4 <- minho2(x3, y3) # 함수에서 함수를 부르는 경우

+ return(x4) # 특정값의 반환

+ }

>

> minho() # minho 함수의 수행

> minho2(10, 20) # minho2 함수의 수행

> minho3(9, 19) # minho3 함수의 수행

> # [4단계] 함수의 계산 결과를 화면에 보이지 않고 변수에 대입하는 경우

> minho4 <-function() {

+ x <-10

+ y <-10

+ return(invisible(x\*y))

+ }

>

> minho4() # minho4 함수의 수행. 그러나 결과는 화면에 보이지 않는다

> result <- minho4() # minho4 함수의 수행결과가 result 변수에 할당된다

> result # 결과 값의 확인

> # [5단계] 함수에서 함수 외부의 변수를 조작해야 하는 경우

> x <- 70 # 시스템 변수 x에 70을 할당

> ls() # 변수 x의 존재 확인

> minho5 <- function() {

+ x <-10 # 함수내에서 사용하는 변수 x에 10을 할당

+ y <-20 # 함수내에서 사용하는 변수 y에 20을 할당

+

+ x<<-40 # 시스템에서 사용하는 변수 x에 40을 할당

+ return(x+y)

+ }

> minho5() # 함수의 수행

> x # 시스템에서 사용하는 변수의 값이 바뀌었음을 확인

**03 조건문**

> # [조건문 기능 요약]

> # if (조건) 명령문 --> 조건에 맞으면 명령문 수행

> # if (조건) 명령문 else 명령문 2 --> 조건에 맞으면 명령문 수행, 맞지 않으면 명령문2 수행

> # ifelse(조건, 명령문1, 명령문2) --> 위와 동일

> # switch(기준, 조건1, 명령문1, 조건2, 명령문2, ....)

> # --> 기준을 정하고 조건1이 발생하면 명령문1 수행, 조건2가 발생하면 명령문2 수행

> # 테스트를 위한 준비

> x <-runif(1) # 0과 1 사이의 난수를 생성해서 x에 할당

> x

> # [if문의 사용 예 1]

> if(x>0) print(abs(x)) # x가 0 보다 크면 절대값을 화면에 출력

> if(x<0.5) print(1-x) else print(x) # x가 0.5보다 작으면 1-x를 출력한다. 아니면 x를 출력

> ifelse(x<0.5, 1-x, x) # 위와 동일

> # [if문의 사용 예 2]

> xx <-(x-0.5) # 변수에 값을 설정한다

> if( xx <0) {

+ print(xx)

+ print("x is negative")

+ print(abs(x))

+ } else {

+ print(xx)

+ print("x is positive")

+ print(x)

+ }

>

> # [switch문의 사용 예]

> xxx <- c(1:10) # 변수의 값을 설정한다

> switch(xxx[2],

+ "1"=print("one"),

+ "2"=print("two"),

+ "3"=print("three"),

+ print("NOT")

+ )

>

**04 반복문**

> # 반복기능 요약

> # [반복 기능의 종류와 문법]

> # for (반복) 명령문

> # while (조건) 명령문

> # repeat 명령문

> # [for문의 사용 예 1]

> sum1 <- 0 # 변수를 설정

>

> for(i in seq(1,10, by=1)) sum1 <-sum1+i # 1에서 10을 1단위로 차례로 넣고, 합을 구한다

> sum1 # 결과 확인

> # [for문의 사용 예 2]

> sum1 <-0 # 변수를 설정

> for(i in 1:5) { # 1\*1, 1\*2.... 5\*5의 합을 구한다

+ for(j in 1:5)

+ sum1 <-sum1+i\*j

+ }

>

> sum1 # 결과 확인

> # [while문의 사용 예 1]

> sum2 <- 0

> i <-1

>

> while(i <=10) { # 1,2,3... 10을 차례대로 넣고 더한다

+ sum2 <-sum2+i;

+ i <-i+1

+ }

> sum2 # 결과 확인

> # [while문의 사용 예 2]

> sum2 <-0

> i <- 0

>

> while(i<=5) { # 1\*1, 1\*2.... 5\*5의 합을 구한다

+ j<-0

+ while(j <=5) {

+ sum2 <-sum2+i\*j

+ j <- j+1

}

+ i<-i+1

+ }

>

> sum2 # 결과 확인

> # [repeat문의 사용 예 1]

> sum3 <-0

> i <-1

>

> repeat{

+ sum3 <- sum3 + i

+ i <- i+1

+ if(i>10) break

+ }

>

> sum3

> i

> # [repeat문의 사용 예 2]

> sum3 <- 0

> i <- 0

>

> repeat { # 1\*1, 1\*2.... 5\*5의 합을 구한다

+ if(i>5) break

+ j<-0

+ repeat {

+ if(j>5) break

+ sum3 <-sum3+i\*j

+ j<-j+1

+ }

+ i<-i+1

+ }

>

> sum3

>

**05 연산자와 인덱스**

**06 추가적인 기능들**

> myF <- function(x){

+ return(x\*x)

+ }

>

> myF (2) # 함수를 사용해 본다

> setwd("c:/temp") # 작업할 디렉터리 설정

> save(myF, file="myF.Rdata") # 함수를 저장

> rm("myF") # 함수를 메모리에서 지운다

> myF(2) # 함수가 메모리에 없으므로 당연히 에러가 난다

> load("myF.Rdata") # 파일에 있는 함수를 읽어서 사용할 수 있게 한다

> myF(2) # 함수가 수행된다

> # 1개의 파일에 여러 개의 함수를 저장하는 방법에 대하여 실습한다

> rm("myF")

> myF <- function(x) { return (x\*x)} # 함수 선언

> myF2 <-function(x){ return(x)} # 함수 선언

> save(myF,myF2,file="myFF.Rdata") # 2개의 함수를 하나의 파일에 저장

> rm("myF", "myF2") # 선언된 함수를 지운다

> ls() # 함수가 지워진 것을 확인

> load("myFF.Rdata") # 함수를 사용할 수 있게 한다

> ls() # 함수가 시스템에 올라온 것을 확인

>

=============================================

> # 프로그램을 제작하다 보면 연속적인 변수를 생성해야 하는 경우가 있다.

> # 이런 경우에는 다음과 같이 한다

> for(i in 1:10) {

+ assign(paste("x", i, sep=""),i) # xi에 i의 값을 대입

+ }

>

> x1 # x1의 값을 확인

> x2

=================================================

> # 프로그램을 제작하다 보면 사용자의 입력이 있을 때까지 실행을 멈출 필요가 있다.

> # 이런 경우에는 다음과 같이 한다

> fun <- function() {

+ answer <- readline("데이터를 입력하세요 : ") # 입력을 기다린다

+ if(substr(answer,1,1)=="n")

+ cat("입력된 글자는 N 입니다 !\n")

+ else

+ cat("입력된 글자는 Y 입니다 ! \n")

+ }

>

> fun()

> # 프로그램의 실행 중에 멈추어서 사용자가 스페이스로 구분된 여러 개의 입력을

> # 받고자 하는 경우에는 다음과 같이 한다

> fun2 <-function() {

+ x<-readline("데이터를 입력하세요 :")

+ unlist(strsplit(x, " "))

+ }

>

> y <-fun2() 데이터를 입력하세요

> y ================================================

> # 프로그램의 실행 중에 간단한 메뉴를 보여주고, 처리과정을 조정하는 경우가 필요하면 다음과 같이 한다

> fuMenu <- function(){

+ answer <- menu (c("SEOUL","BUSAN","DAEGU")) # 공간을 띄우면 안됨. 메뉴를 보여줌

+ if(answer ==1)

+ {cat ("서울을 선택\n")}

+ else if (answer ==2)

+ {cat ("부산을 선택\n")}

+ else

+ {cat ("대구를 선택\n")}

+

+ }

>

> fuMenu() # 함수를 수행하면 화면에 메뉴를 보여준다

===============================================

> # 사용자의 입력으로 특정 데이터 집단을 사용하는 경우

> k <- c("a","b","c") # k라는 데이터 집단을 생성

>

> p <-get(readline())

> p

**Chapter 3**

**01 개요**

**02 화면을 분할하는 방법**

**03 그래프 그리는 방법(산포도)**

**> abc <- c(260, 300, 250, 280, 310)**

**> def <- c(180, 200, 210, 190, 170)**

**> ghi <- c(210, 250, 260, 210, 270)**

**> plot(abc, type="o", col="red", ylim=c(0,400), axes=F, ann=F)**

**> axis(1, at=1:5, lab=c("A", "B", "C", "D", "E"))**

**> axis(2, ylim=c(0,400))**

**> title(main="Fruit", col.main="red", font.main=4)**

**> title(xlab="Day", col.lab="black")**

**> title(ylab="Price", col.lab="blue")**

**> lines(def, type="o", pch=21, col="green", lty=2)**

**> lines(ghi, type="o", pch=22, col="blue", lty=2)**

**> #pch는 0~25까지 번호마다 표시가 할당되어 있음. 22번은 사각형 모양임**

**> legend(4,400, c("BaseBall","SoccerBall","BeachBall"), cex=0.8,**

**+ col=c("red","green","blue"), pch=21, lty=1:3)**

**> #pch는 0~25까지 번호마다 표시가 할당되어 있음. 21번은 동그란 모양임**

**04 그래프 그리는 방법(막대 그래프)**

**> x <-c(1,2,3,4,5,6)**

**> x # 데이터의 모양 확인**

**> barplot(x, names="매출") # 가장 기본적인 막대 그래프의 생성**

**> # 복잡한 형태의 막대 그래프를 그리기 위한 데이터의 설정**

**> xx <-matrix(c(1,2,3,4,5,6), 3, 2) # 1,2,3과 4,5,6 2개의 묶음으로 나눈다**

**> xx # 데이터의 모양 확인**

**> barplot(xx) # 막대 그래프가 행으로 그려짐을 확인한다**

**> barplot(xx, beside=T, names=c("Korea","America")) # 열로 그려진 막대 그래프에 레이블을 부여**

**> # 복잡한 형태의 막대 그래프를 그리기 위한 데이터의 설정 2번째**

**> xxx <-matrix(c(1,2,3,4,5,6), 2, 3) # 1,2와 3,4 그리고 5,6 3개의 묶음으로 나눈다**

**> xxx # 데이터의 모양 확인**

**> barplot(xxx) # 막대 그래프가 행으로 그려짐을 확인한다**

**> barplot(xxx, beside=T) # 막대 그래프가 열로 그려짐을 확인한다**

**> barplot(xxx, beside=T, names=c("Korea", "America", "Europe"))**

**> abc <- c(110, 300, 150, 280, 310) # 시즌별(=A,B,C,D,E) BaseBall 판매현황**

**> barplot(abc, main="Base Ball 판매량", xlab="Season", ylab="판매량",**

**+ names.arg=c("A","B","C","D","E"), border="blue",density=c(10, 30, 50, + 30, 10))**

**=========================================**

**> abc <- c(110, 300, 150, 280, 310) # 시즌별(=A,B,C,D,E) BaseBall 판매현황**

**> def <- c(180, 200, 210, 190, 170) # 시즌별 SoccerBall 판매현황**

**> ghi <- c(210, 150, 260, 210, 70) # 시즌별 BeachBall 판매현황**

**> B\_Type2 <- matrix(c(abc,def,ghi), 5,3)**

**> B\_Type2**

**==========================================**

**> barplot(B\_Type2, main="Ball Type별 시즌의 판매량", xlab="Ball Type", ylab="매출",**

**+ beside=T,**

**+ names.arg=c("BaseBall","SoccerBall","BeachBall"),**

**+ border="blue",col=rainbow(5), ylim=(c(0,400)))**

**>**

**> # 그림에 범례를 추가하는 과정 > legend(16, 400, c("A 시즌","B 시즌","C 시즌","D 시즌","E 시즌"), cex=0.8, + fill=rainbow(5)) >**

**=========================================**

**> barplot(t(B\_Type2), main="시즌 별 볼타입에 따른 판매량", xlab="Season",**

**+ ylab="Price",beside=T,**

**+ names.arg=c("A","B","C","D", "E"), border="blue",col=rainbow(3),**

**+ ylim=(c(0,400)))**

**> legend(16, 400, c("BaseBall","SoccerBall","BeachBall"), cex=0.8,**

**+ fill=rainbow(5))**

**==========================================**

**> barplot(t(B\_Type2), main="시즌별 볼 타입에 따른 판매량(누적 표시형)", xlab="Season",**

**+ ylab="매+ 출",**

**+ names.arg=c("A","B","C","D","E"), border="blue",col=rainbow(3),**

**+ ylim=(c(0,1000)))**

**> legend(4.5,1000, c("BaseBall","SoccerBall","BeachBall"), cex=0.8, + fill=rainbow(3))**

**>**

**==========================================**

**05 그래프 그리는 방법(점 그래프)**

**06 그래프 그리는 방법(히스토그램)**

**07 그래프 그리는 방법(파이 그래프)**

**> T\_sales <- c(210, 110, 400, 550, 700, 130) # 자료를 준비한다**

**> pie(T\_sales)**

**> pie(T\_sales, init.angle=90, col=rainbow(length(T\_sales)), labels=c("Monday",**

**+ "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"))**

**>**

**> legend(1,1,c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday", + "Saturday"), cex=0.8, fill=rainbow(length(T\_sales)))**

**================================================**

**> week <- c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday")**

**> ratio <- round(T\_sales/sum(T\_sales) \* 100, 1)**

**> label <- paste(week," \n",ratio, "%")**

**> pie3D(T\_sales,main="주간 매출 변동", col=rainbow(length(T\_sales)), cex=0.8,**

**+ labels=label)**

**================================================**

**> legend(0.5,1,c("Mond", "Tue", "Wed", "Thu", "Fri", "Sat"), cex=0.8,**

**+ fill=rainbow(length(T\_sales)))**

**> pie3D(T\_sales,main="주간 매출 변동", col=rainbow(length(T\_sales)), cex=0.8,**

**+ labels=label, explode=0.05)**

**08 그래프 그리는 방법(상자모양 차트)**

**> abc <- c(110, 300, 150, 280, 310) # 시즌별(=A,B,C,D,E) BaseBall 판매현황**

**> def <- c(180, 200, 210, 190, 170) # 시즌별 SoccerBall 판매현황**

**> ghi <- c(210, 150, 260, 210, 70) # 시즌별 BeachBall 판매현황**

**> boxplot(abc,def,ghi)**

**> boxplot(abc,def,ghi,col=c("yellow","cyan","green"), names=c("BaseBall","Socce**

**+ rBall","BeachBall"), horizontal=TRUE)**

**09 그래프에 필요한 추가적인 기능들**

**> # 간단한 그림을 그린다**

**> x <-c(1:10)**

**> dotchart(x)**

**> # 현재 상태에서 다른 그림을 그리면, 이전의 그림이 지워진다**

**> par(new=T) # 현재의 그림을 유지하는 명령**

**> # 새로운 그림을 그린다**

**> b <- c(1,2,1,4,1,5,6,8,7,6,5,3,4,6,7,8,6)**

**> hist(b)**

**>**

**> plot.new() # 그림 지우기**

**=================================================**

**> plot.new() # 그림을 삭제**

**> plot(-4:4, -4:4, type="n") # 점을 찍기 위해 바탕 사각형과 눈금을 그린다**

**> points(rnorm(200), rnorm(200), pch="+", col="red") # + 표시의 빨간 점을 200개 그린다**

**> par(new=T) # 현재 그림을 유지한다**

**> points(rnorm(200), #0 표시의 점을 200개 추가한다**

**+ rnorm(200),**

**+ pch="o",**

**+ col="cyan")**

**=================================================**

**> x <- c(1:10)**

**> y <- x\*x**

**> plot(x, y, type='n', main="Title") # 그림 그리는 좌표와 외곽을 그린다**

**> for( i in 1:5) lines(x, (y+i\*5), col=i, lty=i) # 꺾은선을 그린다**

**================================================**

**> library(ggplot2)**

**> use (iris)**

**> head(iris, n=10)**

**> qplot(Sepal.Length, Petal.Length, data=iris)**

**> # 앞에서 그린 그림의 분석 차원에서 회귀 분석 라인을 추가하는 경우에 사용된다.**

**> abline(0, 1) # 절편이 0, 기울기가 1이다**

**==============================================**

**> x <- c(1,3,6,8,9)**

**> y <- c(12,56,78,32,9)**

**> plot(x,y)**

**>**

**> segments(6,78, 8,32) # (3,3)과 (4,4)를 연결하는 선분 그리기**

**> arrows( 3,56,1,12) # (6,6)과 (7,7)을 연결하는 화살표 그리기**

**> rect(4,20,6,30, density=3) # (4,20)과 (6, 30)을 연결하는 사각형 그리기**

**>**

**> text(4,40, "이것은 샘플입니다", srt=55) # 55는 각도**

**> mtext("상단의 문자열입니다", side=3 )**

**> mtext("우측의 문자열입니다", side=4, adj=0.3) # adj가 0dlaus x축에 붙는다**

**>**

**> box(lty=2, col="red") # 그림의 테두리 그리기, 색은 빨간색**

**>**

**> # x축의 추가, y축 40의 위치에 0~10까지 표시하고, 색은 빨간색이다**

**> axis(1, pos=40, at=0:10, col=2)**

**> axis(2, pos=5, at=10:60) # y축의 추가, 10~60까지만 축이 존재**

**10 다양한 그래프 종류의 소개**

**> x <- c(1,1,1,2,2,2,2,2,2,3,3,4,5,6)**

**> y <- c(2,1,4,2,3,2,2,2,2,2,1,1,1,1)**

**> zz <- data.frame(x, y) # 산포도를 위한 데이터를 생성한다.**

**> zz**

**>**

**> sunflowerplot(zz)**

**=============================================**

**> data(mtcars) # mtcars를 사용하겠다는 명령어**

**> stars(mtcars[ , 1:4]) # mtcars 중 1~4번째 항목을 대상으로 그래프를 그리는 명령어**

**============================================**

**> xx <- c(1,2,3,4,5)**

**> yy <-c(2,3,4,5,6)**

**> zz <-c(10, 5, 100, 20, 10)**

**> symbols(xx, yy, zz)**

**==============================================**

**> xx <- c(1,2,3,4,5)**

**> yy <- c(20,13,40,50,60)**

**> zz <- c(10,5,100,20,10)**

**> c <- matrix(c(xx,yy,zz),5,3)**

**> c**

**> pairs(c)**

**11 3차원 그래프**

**> x1 = seq(-3, 3, length = 50) # -3에서 3까지 50개의 자료 생성**

**> x2 = seq(-4, 4, length = 60) # -4에서 4까지 60개의 자료 생성**

**> f = function(x1, x2) {x1^2 + x2^2 + x1\*x2 } # 함수 만들기**

**> y = outer(x1, x2, FUN = f) # 외적**

**> persp(x1,x2,y)**

**============================================**

**> x1 = seq(-3, 3, length = 50) # -3에서 3까지 50개의 자료 생성**

**> x2 = seq(-4, 4, length = 60) # -4에서 4까지 60개의 자료 생성**

**> f = function(x1, x2) {x1^2 + x2^2 + x1\*x2 } # 함수 만들기**

**> y = outer(x1, x2, FUN = f) # 외적**

**> contour(x1,x2,y)**

**12 그래픽용 패키지 사용**

**> # 데이터를 준비한다**

**> data(quakes) # 데이터를 준비한다**

**> head(quakes) # 데이터의 모습을 확인한다**

**> library(lattice) # lattice 패키지 사용 준비, 필요하다면 설치하는 과정을 실행한다**

**> mini = min(quakes$depth) # depth 최소값**

**> maxi = max(quakes$depth) # depth 최대값**

**> r = ceiling((maxi-mini)/8) # depth 구간 크기**

**> inf = seq(mini, maxi, r) # 구간 만들기**

**> quakes$depth.cat = factor(floor((quakes$depth – mini)/r), labels = paste(inf, + inf + r, sep = "-")) # 팩터 변수 만들기**

**> xyplot(lat ~ long | depth.cat**

**+ , data = quakes**

**+ , main = "Fiji earthquakes**

**+ data") # xy plot**

**==========================================**

**> cloud(mag ~ lat \* long**

**+ , data = quakes**

**+ , sub = "magnitude with longitude and lattide") # cloud 3\*dim plot**

**============================================**

**> splom(quakes[,1:4])**

**=============================================**

**> bwplot(mag ~ depth.cat**

**+ , data = quakes**

**+ , main = "깊이 범부에 따른 지진 강도 상자그림") # depth.cat에 따른 상자그림**

**> op = par(mfrow = c(1,2)) # 행 1, 열 2의 레이아웃**

**> hist(quakes$mag) # 히스토그램 그래프**

**> hist(quakes$mag, probability = T, main = "histogram with density line")**

**> lines(density(quakes$mag)) # 그래프에 라인 넣기**

**> par(op)**

**=============================================**

**> install.packages("ggplot2") # 패키지를 설치한다**

**> library(ggplot2) # 패키지를 사용할 수 있게 준비한다**

**> head(iris) # 데이터의 모습을 확인한다**

**> qplot(Sepal.Length, Petal.Length, data=iris)**

**> qplot(Sepal.Length, Petal.Length, data = iris, color = Species, size = Petal. + Width)**

**> qplot(Sepal.Length, Petal.Length, data = iris, geom = "line", color = + Species)**

**> qplot(age, circumference, data = Orange, geom = "line", colour = Tree, main =**

**+ "How does orange tree circumference vary with age?")**

**============================================**

**Chapter 4**

**01 개요**

**02 통계 분석의 전체범위**

**03 데이터 분석 과정**

**04 통계 분석**

**05 데이터 분석에 사용되는 중요기법**

**Chapter 5**

**01 개요**

**02 시뮬레이션**

**> # 주사위 던지기 시뮬레이션 프로그램**

**>**

**> die <- function(n) # n번 주사위를 던지는 함수**

**+ {**

**+ # 주사위의 숫자가 6개이므로 각 숫자가 몇 개 나왔는지 저장할 변수가 필요하다**

**+**

**+ count1 <- 0 # 던진 횟수를 저장하는 변수, 0으로 초기화한다**

**+ count2 <- 0 # 던진 횟수를 저장하는 변수, 0으로 초기화한다**

**+ count3 <- 0 # 던진 횟수를 저장하는 변수, 0으로 초기화한다**

**+ count4 <- 0 # 던진 횟수를 저장하는 변수, 0으로 초기화한다**

**+ count5 <- 0 # 던진 횟수를 저장하는 변수, 0으로 초기화한다**

**+ count6 <- 0 # 던진 횟수를 저장하는 변수, 0으로 초기화한다**

**+**

**+ for(i in 1:n)**

**+ {**

**+ # 0~1 사이의 숫자를 뽑고, 여기에 6을 곱하고, 소수점 이하를 버린다**

**+ # 그러면 0 이 나오는 경우가 있으므로 1을 더한다**

**+ x <- (trunc(runif(1)\*6) +1)**

**+**

**+ cat("x is ",x, "\n") # 나온 숫자를 보여준다**

**+**

**+ if(x==1) count1=count1+1 # 숫자별로 변수에 더한다**

**+ if(x==2) count2=count2+1**

**+ if(x==3) count3=count3+1**

**+ if(x==4) count4=count4+1**

**+ if(x==5) count5=count5+1**

**+ if(x==6) count6=count6+1**

**+ }**

**+**

**+ cat(" 1 number occurs : ",count1,"\n") # 결과를 표시한다**

**+ cat(" 2 number occurs : ",count2,"\n")**

**+ cat(" 3 number occurs : ",count3,"\n")**

**+ cat(" 4 number occurs : ",count4,"\n")**

**+ cat(" 5 number occurs : ",count5,"\n")**

**+ cat(" 6 number occurs : ",count6,"\n")**

**+**

**+ }**

**> die(10) # 시뮬레이션을 수행하기 위하여 die() 함수를 호출한다. 총 10번 던진다**

**03 소셜네트워크 분석**

**> # 소셜네트워크의 실습을 위해 자료를 준비한다**

**> # 일단 4X4 matrix를 임시로 만든다**

**> x <-matrix(1:16, nrow=4, dimname=list(c("a","b","c","d"), c("a","b","c","d")))**

**> x # 데이터의 모습을 보여준다**

**> # x의 값을 위의 규칙을 고려하여 내가 원하는 형태로 바꾸어 주기**

**> # 물론 꼭 아래와 같이 해야 하는 것은 아니다. 그래프의 모양이 다르면, 데이터는 바뀐다.**

**> x[1,1] <-0 # 데이터의 값을 내가 원하는 형태로 재정의 하고 있다**

**> x[1,2] <-1**

**> x[1,3] <-1**

**> x[1,4] <-1**

**> x[2,1] <-1**

**> x[2,2] <-0**

**> x[2,3] <-1**

**> x[2,4] <-1**

**> x[3,1] <-1**

**> x[3,2] <-1**

**> x[3,3] <-0**

**> x[3,4] <-1**

**> x[4,1] <-1**

**> x[4,2] <-1**

**> x[4,3] <-1**

**> x[4,4] <-0**

**> library(igraph) # 만약 설치되어 있지 않으면, 설치한다**

**> install.packages("igraph")**

**> # 그래프의 연결선을 그리고, 연결선은 방향이 없도록 한다. 데이터의 이름은 g로 한다**

**> g <-graph.adjacency(x, weighted=T, mode="undirected")**

**>**

**> # g의 이름은 그래프의 노드 이름이 된다.**

**> V(g)$label <-V(g)$name**

**> # g의 숫자는 연결을 의미한다**

**> V(g)$degree <- degree(g)**

**> layout1 <-layout.fruchterman.reingold(g)**

**> plot(g, layout=layout1)**

**> plot(g, layout=layout1) # 다른 스타일의 그림**

**> tkplot(g, layout=layout.kamada.kawai)**

**> # 필요한 라이브러리를 부른다. 만약 에러 문구가 나오면 설치해야 한다**

**> # install.packages("rgl") > library(rgl)**

**> # 그림의 모양과 3차원을 설정한다**

**> coords <-layout.kamada.kawai(g, dim=3)**

**> open3d() # 새 창이 열림**

**> # 그림을 그린다**

**> rglplot(g, vertex.size=3, vertex.label=NA, edge.arrow.size=2, layout=coords)**

**04 의사결정 트리**

**> # 필요한 패키지의 설치**

**> install.packages("rpart")**

**> install.packages("rpart.plot")**

**>**

**> # 사용을 위한 준비**

**> library(rpart)**

**> library(rpart.plot)**

**> iris # 데이터의 모습을 보인다**

**> # Species를 기반으로 데이터를 분리하다**

**> m1<-rpart(Species~., data=iris)**

**> m1**

**> # 분리된 데이터 정보가 읽기 어렵기 때문에, 이것을 그래프로 표시한다**

**> prp(m1, type=4, extra=2, digit=3) #그래프로 표시하라**

**> # Species를 기반으로 Sepal.Length에 맞추어 데이터를 분리하라**

**>**

**> m3<-rpart(Species~Sepal.Length, data=iris)**

**> m3 # 분리된 데이터 정보를 보여라**

**> # 분리된 데이터 정보가 읽기 어렵기 때문에, 이것을 그래프로 표시한다**

**> prp(m3, type=4, extra=2, digit=3) # 그래프로 표시**

**05 워드 클라우드**

**> # 작업 디렉터리의 설정**

**> setwd("c:/workLog/R")**

**> # 필요한 패키지 설치**

**> # KoNLP는 한글 문장에서 명사를 추출하는 기능 등을 제공**

**> # wordcloud는 워드 클라우드를 만들기 위한 패키지**

**> install.packages("KoNLP")**

**> install.packages("wordcloud")**

**>**

**> # 설치한 패키지 로드**

**> library(KoNLP)**

**> library(wordcloud)**

**>**

**> # 세종 사전을 사용함 >**

**useSejongDic()**

**>**

**> # 분석용 데이터를 txt 변수로 불러오기**

**> # 분석용 데이터는 인터넷 등에서 문자 형태로 된 메일이나 게시물의 내용을**

**> # 복사하여 텍스트의 형태로 하나의 파일에 모아 놓은 것이다.**

**> # 이 자료가 크면 클수록 결과의 정확도가 높아지는 것은 당연한 일 !!!**

**> # 자료의 모습은 뒷부분에 있다.**

**>**

**> txt<-readLines("propose.txt") # 내용은 뒷부분에 있음**

**>**

**> # 데이터 중에서 명사만 골라서 place에 할당**

**> # KoNLP에서 제공하는 extractNoun을 사용하여 명사 추출**

**> place<-sapply(txt, extractNoun, USE.NAMES=F)**

**> place**

**>**

**> # 추출된 명사를 30개만 출력해봄**

**> head(unlist(place), 30)**

**>**

**> # 필터링을 위해 unlist 작업을 해서 저장**

**> c<-unlist(place)**

**>**

**> # 두 글자 이상 되는 것만 필터링하기**

**> place<-Filter(function(x) {nchar(x) >=2 }, c)**

**>**

**> # 원하지 않는 내용 걸러내기**

**> place <-gsub("네이버","", place)**

**> place <-gsub("프로포즈","", place)**

**> place <-gsub("선물","", place)**

**> place <-gsub("조회","", place)**

**> place <-gsub("와이프","", place)**

**> place**

**>**

**> # 파일로 저장한 후, 테이블 형태로 변환하여 불러온다.**

**> write(unlist(place), "propose\_2.txt")**

**>**

**> # 수정 완료된 파일을 table 형식으로 변환하여 다시 불러온다**

**> rev<-read.table("propose\_2.txt")**

**>**

**> # text 형태로 결과를 확인**

**> nrow(rev) #데이터 건수의 확인**

**> wordcount<-table(rev) #테이블 형태로 변환하여 wordcount 변수에 할당**

**> head(sort(wordcount, decreasing=T), 30) #상위 30개의 조회**

**># 그래픽 출력을 위한 라이브러리 로드**

**> library(RColorBrewer) > palete<-brewer.pal(9, "Set1") # 글자 색의 지정**

**> # 출력 지시**

**> wordcloud(names(wordcount), freq=wordcount, scale=c(5,0.5), >rot. per=0.25,min.**

**+ freq=1, random.order=F, random.color=T, colors=palete)**

**>**

**> # 결과의 저장 > savePlot("proposepng", type="png")**

**06 몬테카를로 시뮬레이션**

**> # 부분 모듈을 함수로 정의한다**

**>**

**> coin<-function() {**

**+ x<-runif(1) # 0~1 사이의 난수 생성**

**+**

**+ if(x<=1/2)**

**+ result<-1**

**+ else**

**+ result<-0**

**+**

**+ return(result)**

**+ }**

**>**

**> # 동전 던지기의 몬테카를로**

**> # 시뮬레이션을 실행한다.**

**>**

**> montecarlo<-function(n) {**

**+ count<-0**

**+ for(i in 1:n) {**

**+ count<-count+coin()**

**+ }**

**+**

**+ return(count/n) + + }**

**>**

**> # 시뮬레이션 실행**

**> montecarlo(10000)**

**====================================================**

**> # 부분 모듈을 함수로 정의한다**

**>**

**> coin<-function() {**

**+ x<-runif(1) # 0~1 사이의 난수 생성**

**+**

**+ if(x<=1/2) men<-1**

**+ else men<-0**

**+ return(men)**

**+ }**

**>**

**> # 부분 모듈을 함수로 정의한다.**

**> coinFunction <- function(n) {**

**+ count <- 0**

**+ for( i in 1:n) {**

**+ x<-coin()**

**+ if(x==1) count<-count+1;**

**+ }**

**+ return(count)**

**+}**

**>**

**> # 시뮬레이션 정의 > coinSimulation <-function(n) {**

**+ count <- 0;**

**+ for(i in 1:n) {**

**+ count <- count + coinFunction(20)**

**+ }**

**+**

**+ return(count/n)**

**+}**

**>**

**> # 시뮬레이션 실행**

**> # 20번을 한단위로 50단위 수행**

**> coinSiumulation(50)**

**===============================================**

**> area <- function(n) {**

**+ count <-0;**

**+**

**+ for(i in 1:n) {**

**+ point <-runif(2) # 0~1 사이의 난수 2개 발생(X, Y 값으로 가정할 수 있다)**

**+**

**+ if(sqrt(point[1]^2+point[2]^2) < 1) {**

**+ count <- count +1; #속하는 점의 숫자를 센다**

**+ }**

**+**

**+ }**

**+ return(count/n)**

**+}**

**>**

**> area(1000)**

**Chapter 6**

**01 개요**

**02 일반적인 명령어**

**03 데이터 처리**

**04 데이터를 그래프로 표현하는 방법**

> setwd("c:/workLog/R/plots/plots") # 작업할 디렉터리를 설정

> data <-read.csv("categorical2.csv") # 데이터를 불러온다

> head(data) #불러온 데이터를 확인한다

> install.packages("reshape") # 데이터의 변형을 위해 필요한 패키지 설치

> library(reshape) # 패키지를 사용하겠다고 선언

> data2 <-melt(data, measure.var=c("사무직", "생산직","영업직"))

> head(data2)

> library(lattice) # 그래프를 위해 필요한 패키지를 사용하겠다고 선언한다

> plot <-bwplot(value~variable, data=data2) # data2로 그래프를 그린다

> plot$ylab <-"How many coffee drink in a day?“ # 그래프 y축에 이름을 부여

> print(plot) # 최종적으로 완료된 그림을 화면에 보여준다

==========================================

> setwd("c:/workLog/R/plots/plots")

> data <- read.csv("continuous2.csv")

> head(data)

> library(lattice)

> plot <-xyplot(이동거리 ~ 생산성, data=data)

> plot$xlab <-"커피를 위한 이동 거리"

> plot$ylab <-"주간 생산성"

> print(plot)

=================================================

> data <-read.csv("categorical\_categorical2.csv") # 데이터를 읽어온다

> head(data) # 읽어온 데이터의 확인

> winter <- data[, 1:3] # 불러온 데이터의 앞 3개 열을 winter 변수에 할당

> summer <- data[, 4:6] # 불러온 데이터의 뒤 3개 열을 summer 변수에 할당

> winter <-melt(winter, measure.var=c("겨울사무직.","겨울생산직.","겨울영업직."))

> summer <-melt(summer, measure.var=c("여름사무직.","여름생산직.","여름영업직."))

> head(winter) # 1열로 고쳐진 데이터의 모습을 확인한다. >head(summer)도 동일

> summer <-cbind(summer, season="summer") # summer 데이터에 season 열을 추가

> winter <-cbind(winter, season="winter") # winter 데이터에 season 열을 추가

> head(winter) # 데이터의 모습 확인 (앞부분)

> tail(summer) # 데이터의 모습 확인 (뒷부분)

> data <-rbind(winter,summer) # winter와 summer를 연결하여 새로운 데이터 생성

> head(data)

> # 만들어진 데이터의 변수 이름을 축소한다

> levels(data$variable)[4:6] <-levels(data$variable)[1:3]

> levels(data$variable) # 변수의 이름을 확인한다

>

> plot <-bwplot(value~variable|season, data=data) # 그래프를 그리고

> plot$ylab="하루 커피 소비량" # y축의 이름을 부여한다

> print(plot) # 그림을 화면에 보여준다

=================================================

> data <-read.csv("continuous\_categorical2.csv") # 데이터를 불러온다

> tail(data) # 데이터 내용을 확인한다

> water <- cbind(data[,1:2], drink="water") # 데이터 1,2번 열과 drink 열을 추가

> tea <-cbind(data[,3:4], drink="tea")

> flask <-cbind(data[,5:6], drink="flask")

>

> head(water) # 작업된 데이터의 내용을 파악한다

> # 데이터의 모습을 보니, 열의 이름이 모두 다르다. 이것을 통일시키자

> col.names <- c("volumn","productivity") # 열의 이름을 설정

> names(water)[1:2] <- col.names # 열의 이름을 기존 자료에 할당

> names(tea)[1:2] <- col.names

> names(flask)[1:2] <- col.names

>

> # 새로운 열의 이름을 부여한 3개의 데이터를 하나의 긴 데이터로 통합

> data <-rbind(water,tea,flask)

> head(data)

> plot <-xyplot(productivity~volumn|drink, data=data)

> plot$xlab <-"하루에 평균 마시는 양"

> plot$ylab <-"주간 생산성"

> panel <- function(x,y) {

+ panel.xyplot(x,y) + panel.loess(x,y, col="red", lty=2)

+ }

> plot$panel <- panel

> print(plot)

>

**05 R을 활용하는 도구**

**06 R 명령어 정리**

> # R에서는 주석을 달고자 하면 # 다음에 원하는 글을 쓴다

> # 키보드 입력 대기 표시

+ # 명령을 완료하라는 표시

> # (예) summary 명령어 입력에 ")"를 넣지 않고 Enter 키를 입력하였으므로 완료하라고 + 표시가 나옴

> summary(a

+

> # 명령어 입력 중에 잘못 입력하면, Esc 키를 누르면 명령어의 입력이 취소된다

> # 명령어 입력 대기 상태에서(> 상태) up, down 키를 이용하여 이전 또는 다음에 수행했던 명령문을 가져

> # 올 수 있다

> # = or <- ,->는 할당 연산자이다 (예, a=10, a<-10, 10->a)

> # 함수명을 넣으면, 함수의 모습이 보여진다

> # (예)아래는 summary와 ls 함수에 대한 실행 예이다

> ls #ls()라고 하면 수행이 되고, ls라고 하면 함수의 소스가 보여진다

> # R에서 문자열은 인용부호로 감싼다 (예) "abcdefg"

> # 변수명은 대소문자로 구분하고 문자, 숫자 그리고 점을 가질 수 있고 숫자로 시작하면 안 된다

> # names(list) list의 요소 이름을 보여준다. 그리고

>list$element는 list의 요소 접근 방법이다

> # (예) list 요소 접근에 대한 예이다

> member <- list(name="minho", tel="1234") # list형의 자료 선언

> member # 자료의 모양 확인

> names(member) # list 자료의 구성 항목명 확인

> member$name # list의 구성 요소에 있는 데이터 확인

> # R에서 자주 사용되는 명령어 정리

> help.start() # R 도움말 화면을 기동하여 필요한 정보를 찾는다

> help(command) # command에 대한 도움말

> demo() # 사용자의 R에 설치된 패키지가 제공하는 R 데모 실행

> # (예) > demo(graphics))

> example(function) # 함수의 예제 표시 (예) > example(summary)

> str(a) # a의 내부 구조 표시

> summary(a) # a의 요약 표시

> dir() # 현재 디렉터리 파일 표시

> setwd(path) # 작업 디렉터리 설정

> ls() or > objects() # 작업 공간의 객체 표시

> ls.str() # 작업 공간의 객체 구조 표시

> rm(x,y,...) # 작업 공간에서 객체 x,y,... 제거

> rm(list=objects()) # 작업 공간에 있는 변수, 함수를 모두 제거

> methods(a) # 함수 a의 여러 가지 실행문 표시

> methods(class=class(a)) # 클래스 a의 실행문 표시

> # (예) 함수를 선언하고, method를 사용하는 예이다

> minho <-function() { # 함수 선언

+ x <-10;

+ y <-20;

+ return(x\*y)

+ }

> minho # 함수명을 넣으면 함수의 모습이 보인다 function() { x <-10; y <-20; return(x\*y) }

> minho() # 함수를 수행한다. 결과를 보여준다

> methods("minho") # 함수의 메소드를 찾는다. 메소드가 없다고 보여준다

> search() # 작업 공간에 있는 패키지 이름을 표시

> library(package) # 패키지를 불러와서 작업 공간에 로드함

> find("command") # command가 포함된 패키지 표시

> library(help=package) # 패키지의 함수와 데이터셋 목록 표시

> # (예) > search() # 작업 공간의 패키지 확인

> find("add") # add 메소드가 포함된 패키지 검색 character(0)

> library(help="graphics") # graphics 패키지에 대한 필요 정보 표시

> find("abline") # abline 메소드가 포함된 패키지 검색

> install.packages("suplib") # R 웹사이트에서 suplib 패키지 설치

> update.packages() # R 웹사이트에서 패키지 업데이트

> save.image(file) # 작업 공간 저장

> load(file) # 작업 공간 불러오기

> # (예) > save.image(file="list2")

> dir()

> load(file="list2")

> ls()

> data(x) # 데이터셋(프레임) x 불러오기

> source(file) # R 스크립트 파일 실행

> sink(file) # 출력을 화면 대신 파일로 저장, until sink()

> # (예)

> sink("abdc") # 출력을 저장할 파일을 지정

> ls() # 출력이 파일로 저장되고 화면에는 보이지 않음

> sink() # sink를 푼다

> ls()

> #===============================================================> # 파일과 관련된 명령어

> scan(file)

> read.table(file,header=TRUE)

> read.csv(file,header=TRUE) # 콤마 분리 파일 읽기

> read.delim(file,header=TRUE) # 탭 분리 파일 읽기

> read.fwf(file,widths) # 고정 길이 형식 파일 읽기

> save(file,x,y,. . . ) # x,y를 작업 공간 형식으로 파일 저장

> print(x) # x를 기본 형식으로 표시

> cat(x,y,. . . ,file="", sep=" ") # x와 y를 합하여 표시

> # (예)

> a <-"abc" > b <-"def"

> cat(a,b)

> print(a)

>

> write.table(x,file="") # 데이터 프레임으로 저장

> write.table(x,"clipboard") # 클립보드로 저장

> read.delim("clipboard") # 클립보드에서 읽어오기

> read.dta("statafile") # Stata 데이터 읽기

> read.ssd("sasfile") # SAS 데이터 읽기

> read.spss("spssfile") # SPSS 데이터 읽기

> #=============================================================== > # 자료형의 생성 및 사용에 관한 명령어

> x=numeric(n) # 모든 요소가 0인 n개의 수치 벡터 생성

> # (예)

> x <-numeric(100)

> c(x,y,. . . ) # 벡터형의 자료를 생성한다

> seq(from,to,by=) # by만큼 증가하는 수 생성

> x <- c (seq (from=1, to=20, by=2))

> gl(n,k,length=n\*k,labels=1:n) # n개의 인자를 k번 반복함

> # (예)

> gl(2,1,20)

> gl(3,1,20) # 1을 1번, 2를 1번 ~~~

> gl(3,3,20) # 1을 3번, 2를 3번 ~~~

> #================================================================= > # 데이터의 조작에 관련된 명령어

> X=matrix(x,nrow=r,ncol=c) # rxc 행렬을 생성

> rbind(x,y,. . . ) # 행 묶음

> cbind(x,y,. . . ) # 열 묶음

> # (예)

> x <- c(1:12)

> X <- matrix(x, nrow=3, ncol=4)

> rbind(X)

> rbind(X[,2])

> cbind(X[2,])

> rbind(X[1,2])

> t(X) # 전치행렬(=뒤집은 행렬)

> det(X) # 행렬식을 구함

> norm(x) # 행렬의 크기를 재는 것으로, norm 값을 구함

> diag(X) # 대각행렬

> x <- c(1:16)

> x

> X <- matrix(x, nrow=4, ncol=4)

> X

> t(X)

> det(X)

> norm(X)

> diag(X)

>

> rowSums(X), rowMeans(X) # 행의 합, 평균

> colSums(X), colMeans(X) # 열의 합, 평균

> dim(X) # 차원

> # (예)

> rowSums(X)

> colMeans(X)

> dim(X)

> # [기타 명령어] > # X%\*%Y 행렬곱, solve(X) 역행렬, solve(X,b) Ax=b의 해,forwardsolve(A,b)

> # 하삼각 행렬의 해,backsolve(A,b) 상삼각 행렬의 해, qr.solve(X) QR 분해 역행렬,

> # qr(X) QR 분해, eigen(X) 고유값 분해, svd(X) 특이값 분해

> #=============================================================== > # 문자열의 조작에 관련된 명령어

> x=character(n) # 빈 문자열 벡터 생성

> nchar(x) # 문자수

> substring(x,start), substr(x,start,stop) # 부분 문자열

> strsplit(x,split) # 문자열 분리

> grep(pattern,x) # 문자열 찾기 -- 본문의 내용을 참조할 것 !! 중요함

> gsub(pattern,replacement,x) # 문자열 대치

> tolower(x), toupper(x) # 소문자 변환, 대문자 변화

> #(예)

> x <-character(10)

> x [1] "" "" "" "" "" "" "" "" "" ""

> nchar(x) [1] 0 0 0 0 0 0 0 0 0 0

> x <- "abc"

> y <- "cefghij"

> substr(y, 3, 5)

> splitC <-"g" > strsplit(y,splitC)

> toupper(x)

> #============================================================== > # 데이터형에 따른 데이터 조작에 관련된 명령어

> X=data.frame(x,y,. . . ) # 데이터 프레임 생성

> X$v # 데이터 프레임 X의 변수 v

> # (예)

> no <- c(1,2,3,4)

> name <-c("Apple","Banana","Peach","Berry")

> prices <- c(500,200,150,50)

> qty <- c(5,2,7,8)

>

> fruit <- data.frame(No=no, Name=name,Prices=prices,Qty=qty)

> fruit

> fruit$No

> fruit$Name

> fruit[1,]

> fruit[, 2]

> fruit[, -2]

> attach(X) # X의 내부변수를 게시, 해제하는 경우에는

> detach(X) 사용

> # 벡터나 데이터 프레임과 같은 데이터형을 다룰 때, 공통적으로 사용되는 데이터 조작 방법

> x[n] # 벡터의 n번째 요소

> x[-n] # n번째 요소를 제외한 모든 요소

> x[1:n] # 첫 번째부터 n번째까지의 요소

> x[-(1:n)] # 첫 번째부터 n번째까지를 제외한 요소

> x[c(1,4,2)] # 1, 4, 2번째의 요소

> x["name"] # "name" 이름의 요소

> x[x > 3] # 3보다 큰 요소

> x[x > 3 & x < 5] # 3과 5사이의 요소

> X[i,j] # i행, j열 요소

> X[i,] # i행 X[,j] j열 요소

> X[,c(1,3)] # 행렬의 1, 3열

> X["name",] # 행렬에서 행 이름이 "name"인 행

> X[["name"]] or X$name # 데이터 프레임의 열 이름이 "name"

> x[["name"]] or x$name # 리스트의 요소 이름 "name"

> x[[n]] or x[n] # 리스트 n번째 요소

> # (예)

> no <- c(1,2,3,4)

> name <-c("Apple","Banana","Peach","Berry")

> prices <- c(500,200,150,50)

> qty <- c(5,2,7,8)

>

> fruit <- data.frame(No=no, Name=name,Prices=prices,Qty=qty)

> fruit

> fruit[1,1]

> fruit["Name"]

> fruit[1

> fruit[,c(1,2)]

> fruit[["Qty"]]

> #=============================================================== > # 데이터에 대한 특성을 파악하고 조작하는 명령어

> which.max(x) # 최대값 위치

> which.min(x) # 최소값 위치

> rev(x) # 반전

> sort(x) # 오름차순 정렬

> cut(x,breaks=n) # 구간 나누기

> match(x, y) # 일치 벡터

> which(x == a) # 위치 찾기

> na.omit(x) # 데이터 손실 제거

> na.fail(x) # 데이터 손실이 있으면 에러 메시지

> unique(x) # 반복 제거

> table(x) # 테이블 작성

> subset(x, ...) # 부분집합

> sample(x, n) # 표본추출

># (예)

> y <- c( 11,12,16,11,18,19,20,19,20)

> y

> which.max(y)

> sort(y)

> which(y==16)

> table(y

> unique(y)

> sample(y, 4)

>

> max(x) # 최대값

> min(x) # 최소값

> fivenum(x) # 다섯 수치 요약 (min,Q1,median,Q3,max)

> sum(x) # 합계

> diff(x) # 반복 차이 계산

> prod(x) # 곱하기

> mean(x) # 평균

> median(x) # 중위수

> quantile(x,probs=) # 사분위수 (defaults to 0,.25,.5,.75,1)

> weighted.mean(x, w) # 가중평균

> rank(x) # 순위

> sd(x) # 표준편차

> var(x, y) or cov(x, y) # 분산, 공분산

> cor(x, y) # 상관계수

> scale(x) # 표준화 변환

> pmin(x,y,...) # 각각의 요소 최소값

> pmax(x,y,...) # 각각의 요소 최대값

> cumsum(x) # 1번 요소부터 합계값

> cumprod(x) # 1번 요소부터 곱셈값

> cummin(x) # 1번 요소부터 최소값

> cummax(x) # 1번 요소부터 최대값

> union(x,y) 합집합, > intersect(x,y) 교집합, > setdiff(x,y) 차집합

> # (예)

> x <-c(1,2,4,5,7,8,9,10)

> sum(x)

> fivenum(x)

> diff(x)

> prod(x) [1] 201600

> rank(x) [1] 1 2 3 4 5 6 7 8

> scale(x)

> cumsum(x)

> cummin(x)

> pmin(x)

>

> #=============================================================== > # R에서 다양한 분포를 가지는 자료를 생성하는 경우에 사용하는 명령어

> rnorm(n, mean=0, sd=1) # 가우시안 정규분포 생성

> rexp(n, rate=1) # 지수분포 생성

> rgamma(n, shape, scale=1) # 감마분포 생성

> rbinom(n, size, prob) # 이항분포 생성

> rpois(n, lambda) # 포아송분포 생성

> rgeom(n, prob) # 기하분포 생성

> rhyper(nn, m, n, k) # 초기하분포 생성

> rweibull(n, shape, scale=1) # 웨이블분포 생성

> rcauchy(n, location=0, scale=1) # 코시분포 생성

> rbeta(n, shape1, shape2) # 베타분포 생성

> rt(n, df) # 스튜던트 t 분포 생성

> rf(n, df1, df2) # 피셔의 f 분포 생성

> rchisq(n, df) # 피어슨 카이제곱 생성

> rlogis(n, location=0, scale=1) # 로지스틱 분포 생성

> rlnorm(n, meanlog=0, sdlog=1) # 로그 정규분포 생성

> rnbinom(n, size, prob) # 음수 이항분포 생성

> runif(n, min=0, max=1) # 균등분포 생성

> rwilcox(nn, m, n), rsignrank(nn, n) # 윌콕슨분포 생성

> # (예)

> runif(20, min=0, max=40)

> rt(20,5)

> rnorm(20, mean0, sd=1)

> # 각종 검정 및 분석에 사용되는 명령어 (상세한 사용법은 책의 본문 참조)

> anova(fit,...) # 분산 분석

> density(x) # 커널밀도 추정

> binom.test() # 이항분포 검정

> pairwise.t.test() # 대응표본 t-검정

> power.t.test() # 독립표본 t-검정

> prop.test() # 비율 검정

> t.test() # t-검정

> lm(formula) # 선형 회귀 모형

> glm(formula,family=) # 일반화 선형 모형

> nls(formula) # 비선형 회귀 모형

===============================================================

> # R의 그래프에 대한 명령어(상세한 사용법은 책의 본문 참조)

> plot(x, y) # 산점도

> curve(expr(x),a,b) # 수식에서 x값 [a, b] 곡선 그리기

> hist(x) # 히스토그램

> stem(x) # 줄기-잎 그림

> barplot(x) # 막대 그래프

> dotchart(x) # 점 그래프

> pie(x) # 원 그래프

> boxplot(x,y,. . . ) # 상자 그림

> sunflowerplot(x, y) # 빈도에 따른 꽃모양 표현, 꽃의 크기가 빈도의 크기

> stripplot(x) # 일차원 산점도

> coplot(x~y | z) # z값에 따른 이차원 그래프(\* 본문에 설명이 없는 그림이다)

> interaction.plot (f1, f2, y) # 교호작용 그림(\* 본문에 설명이 없는 그림이다)

> mosaicplot(x) # 모자익 그래프, 다차원 분할표에 대한 비율과 합계를 사각형 면적으로 표시

> pairs(X) # 산점도 행렬

> plot.ts(x) # 시계열 그래프

> s.plot(x) # 다중 시계열 그래프

> qqnorm(x) # 누적 정규확률 분포 그래프

> qqplot(x, y) # 분위수 그래프(\* 본문에 설명이 없는 그림이다)

> qqline(x) # 분위수 직선 그리기(\* 본문에 설명이 없는 그림이다)

> contour(x, y, z) # 입체형의 등고선 그래프

> filled.contour(x, y, z) # 내부 색칠 등고선 그래프

> image(x, y, z) # 3차원 자료 이미 지표시

> persp(x, y, z) # 3차원의 그래프

> stars(x) # 별 그림

> symbols(x, y, ...) # 산점도 심볼 그래프

> points(x, y) # 점 그리기

> lines(x, y) # 선 그리기

> text(x, y, labels, ...) # 그림 위에 문자 나타내기

> mtext(text, side=3, line=0, ...) # 문자 나타내기

> arrows(x0, y0, x1, y1) # 화살표 나타내기

> abline(a,b) # a 절편 b 기울기 직선

> rect(x1, y1, x2, y2) # 사각형 그리기

> polygon(x, y) # 다각형 그리기

> box() # 외곽선 그리기

> #============================================================== > # 어렵고, 많이 사용되는 명령어 몇 개를 정리한다. 내용은 본문을 참조한다

> replicate(n, func(x=x)) # 반복 실행

> apply(x,index,fun) # 행렬의 행(index=1), 열(index=2) 함수 실행

> x<- cbind(x1 = 3, x2 = c(4:1, 2:5))

> # 컬럼은 x1, x2의 2개이고, 값은 3 / 4,3,2,1,2,3,4,5

> x

> dimnames(x)[[1]] <- letters[1:8] # row에 a~h까지의 문자 할당

> x

> apply(x, 2, mean, trim = .2) # x의 row(=2)의 평균값을 구함

> col.sums <- apply(x, 2, sum) # x의 row(=2)의 합을 구함

> col.sums

> row.sums <- apply(x, 1, sum) # x의 컬럼(=1)의 합을 구함

> row.sums a b c d e f g h 7 6 5 4 5 6 7 8

>

> lapply(x,fun) or sapply(x,fun) # 리스트 함수 실행

> #(예)

> require(stats); require(graphics)

> #자료형 생성

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

> x # 생성된 자료의 모습 확인

> lapply(x, mean) # 각 자료의 평균을 구한다

> lapply(x, quantile, probs = 1:3/4)

> sapply(x, quantile)

> tapply(X,INDEX,FUN=) # 행렬의 모든 셀에 적용

> # (예)

> groups <- as.factor(rbinom(32, n = 5, prob = 0.4))

> groups

> tapply(groups, groups, length) # table과 동일한 효과

> table(groups) # 위의 tapply와 동일한 효과

> #=============================================================== > # Programming (프로그래밍)에 대한 부분을 정리한다

> sum1 <- 0 # 변수를 설정

> # [for문의 사용 예 1]

> for(i in seq(1,10, by=1)) sum1 <-sum1+i # 1에서 10을 1단위로 차례로 넣고, 이것의 합을

> # 구한다

> # [for문의 사용 예 2]

> sum1 <-0 # 변수를 설정

> for(i in 1:5) { # 1\*1, 1\*2.... 5\*5의 합을 구한다

+ for(j in 1:5)

+ sum1 <-sum1+i\*j

+ }

> # [while문의 사용 예 1]

> sum2 <- 0

> i <-1

> while(i <=10) { # 1,2,3... 10을 차례대로 넣고 더한다

+ sum2 <-sum2+i;

+ i <-i+1

+}

> # [while문의 사용 예 2]

>sum2 <-0

>i <- 0

>while(i<=5) { # 1\*1, 1\*2.... 5\*5의 합을 구한다

> j<-0

> while(j <=5) {

> sum2 <-sum2+i\*j

> j <- j+1 > }

> i<-i+1

>}

# [repeat문의 사용 예 1]

>sum3 <-0

>i <-1

>repeat{

> sum3 <- sum3 + i

> i <- i+1

> if(i>10) break

>}

# [repeat문의 사용 예 2]

>sum3 <- 0

>i <- 0

>repeat { # 1\*1, 1\*2.... 5\*5의 합을 구한다

> if(i>5) break

> j<-0

> repeat {

> if(j>5) break

> sum3 <-sum3+i\*j

> j<-j+1

> }

> i<-i+1

>}

> # [if문의 사용 예]

> if(x>0) print(abs(x))

> if(x<0.5) print (1-x) else print(x)

> #==============================================================