R 프로그래밍 기초 Part1

한상곤(sangkon@pusan.ac.kr)

2023.06.13(화)

Contents

1 R 프로그래밍 관련 주요 Topic

1.1 R 프로그래밍 언어 소개

- 1993년 뉴질랜드 오클랜드 대학의 통계학과 교수인 Ross Ihake, Robert Gentleman에 의해서 개발
- 통계 언어인 S의 계보를 잇고 있음
- R의 S+(S+PLUS)의 무료 버전으로 1993년 배포되어 현재는 GNU 프로젝트로 개발 및 개선

1.2 R의 주요 특징

- 통계분석(Statistical Analysis)과 시각화(Visualization)을 위한 공개 소프트웨어로 활용되고 있음
  - 모든 데이터가 객체 형태로 관리
  - 고속 메모리 처리
  - 다양한 자료구조를 제공
  - 패키지 제공
  - 시각화 도구를 제공

1.3 작업 환경 구축하기

- R 공식 사이트에서 무료로 다운로드 할 수 있음
  - 교재 p.18을 참고해서 진행
  - 패키지 설치를 위해서 Rtools 다운로드 후 설치
- RStudio 다운로드 후 설치

1.4 패키지와 Session 보기

- R 패키지는 통계학 관련 교수와 학자 그리고 관련 개발자에 의해서 꾸준히 관리
  - 대부분의 패키지는 무료로 사용할 수 있음

1.5 패키지 설치와 삭제

1. 패키지는 install.packages("패키지명") 명령어를 사용해서 설치할 수 있다. stringr 패키지를 설치해보자.

```
install.packages("stringr")
```

2. RStudio의 Packages 탭을 사용해도 설치가 가능하다.
3. 패키지를 사용하는 방법은 library(패키지명), require(패키지명)으로 가능하다.

```
library(stringr)
search() # 현재 사용하는 패키지 확인
```

```
##  [1] ".GlobalEnv"        "package:stringr"   "package:stats"
##  [4] "package:graphics"  "package:grDevices" "package:utils"
##  [7] "package:datasets"  "package:methods"   "Autoloads"
## [10] "package:base"
```

4. 패키지 삭제는 remove.packages("패키지명")으로 가능하다.

1.6 R Session

R Session이란 사용자가 R 프로그램을 기동한 이후 R 콘솔 시작과 종료 전까지의 기간에 수행된 정보를 의미

```
sessionInfo()
```

```
## R version 4.3.0 (2023-04-21 ucrt)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 11 x64 (build 22621)
##
## Matrix products: default
##
##
## locale:
## [1] LC_COLLATE=Korean_Korea.utf8  LC_CTYPE=Korean_Korea.utf8
## [3] LC_MONETARY=Korean_Korea.utf8 LC_NUMERIC=C
## [5] LC_TIME=Korean_Korea.utf8
##
## time zone: Asia/Seoul
## tzcode source: internal
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] stringr_1.5.0
##
## loaded via a namespace (and not attached):
##  [1] digest_0.6.31    codetools_0.2-19 fastmap_1.1.1    xfun_0.39
##  [5] magrittr_2.0.3   glue_1.6.2       knitr_1.43       htmltools_0.5.5
##  [9] rmarkdown_2.22   lifecycle_1.0.3  cli_3.6.1        compiler_4.3.0
## [13] rstudioapi_0.14  tools_4.3.0      evaluate_0.21    yaml_2.3.7
```

```
## [17] rlang_1.1.1      stringi_1.7.12
```

1.7 작업 공간

- setwd(), 지정된 작업공간을 변경하기 위해서 사용
- getwd(), 지정된 작업공간을 확인하기 위해서 사용

```
getwd()
```

```
## [1] "C:/Users/sigma/works/practice-r"
```

1.8 데이터의 유형과 구조

R에서 제공하는 5가지 주요 자료구조는 다음과 같이 5가지 입니다. - Vector(1차원) - Matrix(2차원) - Array(다차원) - Data Frame(2차원 테이블 구조) - List(중첩 자료구조)

1.9 Vector(벡터)(1차원)

- 일반적인 벡터는 c() 함수를 사용해서 작성할 수 있음

```
c(1,2,3,4,5)
```

```
## [1] 1 2 3 4 5
```

```
c(1:20)
```

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

```
seq(1, 10, 2)
```

```
## [1] 1 3 5 7 9
```

```
rep(1:3, 3)
```

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

- 벡터 연산

```
x <- c(1, 3, 5, 7)
y <- c(3, 5)
union(x, y) # 합집합
```

```
## [1] 1 3 5 7
```

```
setdiff(x, y) # 차집합
```

```
## [1] 1 7
```

```
intersect(x, y) # 교집합
```

```
## [1] 3 5
```

- 벡터는 동일한 자료형으로 생성

```
v1 <- c(33, -5, 20:23, 12, -2:3)
v2 <- c("홍길동", "이순신", "유관순")
v3 <- c(T, TRUE, FALSE, T, TRUE, F, T)
v4 <- c(33, 05, 20:23, 12, "4")
v1; v2; v3; v4
```

```
##  [1] 33 -5 20 21 22 23 12 -2 -1  0  1  2  3
```

```
## [1] "홍길동" "이순신" "유관순"
```

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

```
## [1] "33" "5"  "20" "21" "22" "23" "12" "4"
```

- name을 사용하면 컬럼명을 지정

```
age <- c(30, 35, 40)
names(age) <- c("홍길동", "이순신", "강감찬") # 컬럼명을 지정할 수 있음
age
```

```
## 홍길동 이순신 강감찬
##     30     35     40
```

- 인덱스를 사용해서 벡터 참조 가능

```
v1 <- c(13, -5, 20:23, 12, -2:3) # 생성
v1[1]
```

```
## [1] 13
```

```
v1[c(2, 4)]
```

```
## [1] -5 21
```

```
v1[c(3:5)]
```

```
## [1] 20 21 22
```

```
v1[c(4, 5:8, 7)]
```

```
## [1] 21 22 23 12 -2 12
```

```
v1[-1]; v1[-c(2, 4)]; v1[-c(2:5)]; v1[-c(2, 5:10, 1)]
```

```
##  [1] -5 20 21 22 23 12 -2 -1  0  1  2  3
```

```
##  [1] 13 20 22 23 12 -2 -1  0  1  2  3
```

```
## [1] 13 23 12 -2 -1  0  1  2  3
```

```
## [1] 20 21  1  2  3
```

1.10 Matrix(2차원)

- 행렬은 2차원 배열의 구조를 가지며, 벡터를 활용해서 행렬 객체를 생성할 수 있음

```
m <- matrix(c(1:5))
m
```

```
##      [,1]
## [1,]    1
## [2,]    2
## [3,]    3
## [4,]    4
## [5,]    5
```

- 벡터를 사용해서 열 우선으로 행렬 객체를 생성

```
m <- matrix(c(1:10), nrow = 2)
m
```

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

- 행과 열의 수가 일치하지 않는 경우는 경우 부족한 데이터는 첫번째 데이터를 재사용

```
m <- matrix(c(1:11), nrow = 2)
m
```

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

- 행 우선으로 데이터를 생성할 경우 가능하면 byrow를 사용해서 가독성을 확보

```
m <- matrix(c(1:10), nrow = 2, byrow = T)
m
```

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

- 기존에 만들어준 벡터를 활용해서 행렬을 생성

```
x1 <- c(m, 40, 50:52)
x2 <- c(30, 5, 6:8)
mr <- rbind(x1, x2)
mr
```

```
##    [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
## x1    1    6    2    7    3    8    4    9    5    10    40    50    51    52
## x2   30    5    6    7    8   30    5    6    7     8    30     5     6     7
```

- 열을 기준으로 행렬을 생성할 수 있음

```
mc <- cbind(x1, x2)
mc
```

```
##       x1 x2
##  [1,]  1 30
##  [2,]  6  5
##  [3,]  2  6
##  [4,]  7  7
##  [5,]  3  8
##  [6,]  8 30
##  [7,]  4  5
##  [8,]  9  6
##  [9,]  5  7
## [10,] 10  8
## [11,] 40 30
## [12,] 50  5
## [13,] 51  6
## [14,] 52  7
```

- 행렬 생성 함수를 사용해서 행렬 객체를 생성할 수 있음

```
m3 <- matrix(10:19, 2)
m4 <- matrix(10:20, 2)
m3
```

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

```
m4
```

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

```
mode(m3); class(m3)
```

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

```
## [1] "matrix" "array"
```

- 인덱스(첨자)를 사용하여 행렬 객체에 접근할 수 있음

```
m3[1, ]
```

```
## [1] 10 12 14 16 18
```

```
m3[ , 5]
```

```
## [1] 18 19
```

```
m3[2, 3]
```

```
## [1] 15
```

```
m3[1, c(2:5)]
```

```
## [1] 12 14 16 18
```

- 3행 3열의 행렬 객체 생성하는 방법인 행과 열의 크기를 정하는 것

```
x <- matrix(c(1:9), nrow = 3, ncol = 3)
x
```

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

- 행렬의 크기와 형태를 확인하는 방법은 `length`와 `ncol`을 사용할 수 있음

```
length(x)
```

```
## [1] 9
```

```
ncol(x)
```

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

- `apply`를 사용해서 함수나 사용자 정의 함수를 적용할 수 있음

```
apply(x, 1, max)
```

```
## [1] 7 8 9
```

```
apply(x, 1, min)
```

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

```
apply(x, 2, mean)
```

```
## [1] 2 5 8
```

- 사용자 정의 함수도 가능

```r
f <- function(x) {
  x * c(1, 2, 3)
}
result <- apply(x, 1, f)
result
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    8   10   12
## [3,]   21   24   27
```

```r
result <- apply(x, 2, f) # 1은 행, 2는 열
result
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    4   10   16
## [3,]    9   18   27
```

- 행렬 객체에 칼럼명을 지정할 수 있음

```r
colnames(x) <- c("one", "two", "three")
x
```

```
##      one two three
## [1,]   1   4     7
## [2,]   2   5     8
## [3,]   3   6     9
```

1.11 Array(다차원)

- 배열을 생성하는 방법은 array를 활용하는 것

```r
vec <- c(1:12)
arr <- array(vec, c(3, 2, 2))
arr
```

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

- 배열 객체의 자료 조회하는 방법은 행렬과 유사

```r
arr[ , , 1]
```

```
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
```

```
## [3,]    3    6
```

```
arr[ , , 2]
```

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

```
mode(arr); class(arr)
```

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

```
## [1] "array"
```

1.12 Data Frame(2차원 테이블 구조)

- 데이터 프레임은 열(!) 단위로 서로 다른 자료형을 포함할 수 있음

```
no <- c(1, 2, 3)
name <- c("hong", "lee", "kim")
pay <- c(150, 250, 300)
vemp <- data.frame(No = no, Name = name, Pay = pay)
vemp
```

```
##   No Name Pay
## 1  1 hong 150
## 2  2  lee 250
## 3  3  kim 300
```

```
m <- matrix(
  c(1, "hong", 150,
    2, "lee", 250,
    3, "kim", 300), 3, by = T)
memp <- data.frame(m)
memp
```

```
##   X1   X2  X3
## 1  1 hong 150
## 2  2  lee 250
## 3  3  kim 300
```

- 텍스트 파일을 활용해서 데이터프레임 객체를 생성

```
getwd()
```

```
## [1] "C:/Users/sigma/works/practice-r"
```

```
txtemp <- read.table('./data/emp.txt', header = 1, sep = "", fileEncoding = "euc-kr")
txtemp
```

```
##   사번 이름 급여
## 1  101 hong  150
## 2  201  lee  250
## 3  301  kim  300
```

- CSV 파일도 가능

```
csvtemp <- read.csv('./data/emp.csv', header = T, fileEncoding = "euc-kr")
csvtemp
```

```
##    no   name pay
## 1 101 홍길동 150
## 2 102 이순신 450
## 3 103 강감찬 500
## 4 104 유관순 350
## 5 105 김유신 400
```

```
help(read.csv)
name <- c("사번", "이름", "급여")
read.csv('./data/emp.csv', header = F, col.names = name, fileEncoding = "euc-kr")
```

```
##   사번   이름 급여
## 1   no   name  pay
## 2  101 홍길동  150
## 3  102 이순신  450
## 4  103 강감찬  500
## 5  104 유관순  350
## 6  105 김유신  400
```

- 데이터 프레임을 생성하는 간단한 예제

```
df <- data.frame(x = c(1:5), y = seq(2, 10, 2), z = c('a', 'b', 'c', 'd', 'e'))
df$x
```

```
## [1] 1 2 3 4 5
```

```
str(df)
```

```
## 'data.frame':    5 obs. of  3 variables:
##  $ x: int  1 2 3 4 5
##  $ y: num  2 4 6 8 10
##  $ z: chr  "a" "b" "c" "d" ...
```

```
ncol(df)
```

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

```
nrow(df)
```

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

```
names(df)
```

```
## [1] "x" "y" "z"
```

```
df[c(2:3), 1]
```

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

```
summary(df)
```

```
##        x           y           z
##  Min.   :1   Min.   : 2   Length:5
##  1st Qu.:2   1st Qu.: 4   Class :character
##  Median :3   Median : 6   Mode  :character
##  Mean   :3   Mean   : 6
##  3rd Qu.:4   3rd Qu.: 8
##  Max.   :5   Max.   :10
```

- 데이터프레임 자료에 함수 적용하는 방법도 유사

```
apply(df[ , c(1, 2)], 2, sum)
```

```
##  x  y
## 15 30
```

- 데이터프레임의 부분 객체 만들기는 방법은 subset을 활용

```
x1 <- subset(df, x >= 3)
y1 <- subset(df, y <= 8)
xyand <- subset(df, x >= 2 & y <= 6)
xyor <- subset(df, x >= 2 | y <= 6)
```

- 데이터 프레임 병합은 merge를 사용

```
height <- data.frame(id = c(1, 2), h = c(180, 175))
weight <- data.frame(id = c(1, 2), w = c(80, 75))
user <- merge(height, weight, by.x = "id", by.y = "id")
user
```

```
##   id   h  w
## 1  1 180 80
## 2  2 175 75
```

1.12.1 List(중점 자료구조)

- 리스트를 생성하는 방법은 매우 간단

```
list <- list("lee", "이순신", 95)
list
```

```
## [[1]]
## [1] "lee"
##
## [[2]]
## [1] "이순신"
##
## [[3]]
## [1] 95
```

- 기존의 리스트를 벡터 구조로 변경하는 것도 가능

```
unlist <- unlist(list)
unlist
```

```
## [1] "lee"    "이순신" "95"
```

- 1개 이상의 값을 갖는 리스트 객체 생성

```
num <- list(c(1:5), c(6, 10))
num
```

```
## [[1]]
## [1] 1 2 3 4 5
##
## [[2]]
## [1]  6 10
```

- 리스트를 제대로 활용하는 방법은 key와 value 형식으로 리스트를 생성하는 것

```
member <- list(name = c("홍길동", "유관순"),
               age = c(35, 25),
```

```
                address = c("한양", "충남"),
                gender = c("남자", "여자"),
                htype = c("아파트", "오피스텔"))
member$name
```

```
## [1] "홍길동" "유관순"
```

```
member$name[1]
```

```
## [1] "홍길동"
```

```
member$name[2]
```

```
## [1] "유관순"
```

- 리스트의 key를 이용하여 value에 접근할 수 있다.

```
member$age[1] <- 45
member$id <- "hong"
member$pwd <- "1234"
member$age <- NULL
length(member)
```

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

```
mode(member); class(member)
```

```
## [1] "list"
```

```
## [1] "list"
```

- 리스트 객체에 함수 적용하는 것도 유사

```
a <- list(c(1:5))
b <- list(c(6:10))
lapply(c(a, b), max)
```

```
## [[1]]
## [1] 5
##
## [[2]]
## [1] 10
```

- 결과값을 벡터형식으로 확인하기 위해서는 별도의 함수를 사용

```
sapply(c(a, b), max)
```

```
## [1]  5 10
```

- 다차원 리스트 객체 생성하는 방법은 내포된 리스트 형태

```
multi_list <- list(c1 = list(1, 2, 3),
                   c2 = list(10, 20, 30),
                   c3 = list(100, 200, 300))
multi_list$c1; multi_list$c2; multi_list$c3
```

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

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

```
## [[1]]
## [1] 10
##
## [[2]]
## [1] 20
##
## [[3]]
## [1] 30
```

```
## [[1]]
## [1] 100
##
## [[2]]
## [1] 200
##
## [[3]]
## [1] 300
```

- do.call을 사용해서 다차원 리스트를 열 단위로 바인딩할 수 있음

```
do.call(cbind, multi_list)
```

```
##      c1 c2 c3
## [1,] 1  10 100
## [2,] 2  20 200
## [3,] 3  30 300
```

```
class(do.call(cbind, multi_list))
```

```
## [1] "matrix" "array"
```

1.13 문자열 처리

- 문자열 처리는 교재를 참고

1.14 데이터 불러오기 및 저장하기

- 교재를 참고