

한국한의학연구원, 구본초

통계 프로그래밍 언어

2021년도 1학기 충남대학교 정보통계학과 강의노트



Contents

| | |
|-----------------------------------|-----------|
| List of Tables | ix |
| List of Figures | xi |
| Course Overview | xv |
| I Get Started | 1 |
| 1 Introduction | 3 |
| 1.1 R 설치하기 | 4 |
| 1.2 R 시작 및 작동 체크 | 14 |
| 1.3 R script 편집기 사용 | 19 |
| 1.4 RStudio | 22 |
| 1.4.1 RStudio 설치하기 | 22 |
| 1.4.2 RStudio IDE 화면 구성 | 25 |
| 1.4.3 RStudio 환경 설정 | 32 |
| 1.4.4 RStudio 프로젝트 | 45 |
| 1.5 R 패키지 | 49 |

| | | |
|----------|------------------------------|-----------|
| 1.5.1 | R 패키지 경로 확인 및 변경 | 49 |
| 1.5.2 | R 패키지 설치하기 | 51 |
| 1.5.3 | R 패키지 불러오기 | 53 |
| 1.6 | R 기초 문법 | 54 |
| 1.7 | R Markdown (맛보기) | 59 |
| 2 | R 객체(R object) | 71 |
| 2.1 | 프로그래밍 | 72 |
| 2.1.1 | Prerequisites | 73 |
| 2.1.2 | 프로그램 | 76 |
| 2.2 | 스칼라(scalar) | 80 |
| 2.2.1 | 선언 | 80 |
| 2.2.2 | 숫자형 | 81 |
| 2.2.3 | 문자형 | 84 |
| 2.2.4 | 논리형 스칼라 | 86 |
| 2.2.5 | 결측값(missing value) | 91 |
| 2.2.6 | NULL 값 | 92 |
| 2.2.7 | 무한대/무한소/숫자아님 | 93 |
| 2.3 | 벡터(vector) | 95 |
| 2.3.1 | 벡터의 특징 | 95 |
| 2.3.2 | 벡터의 연산 | 101 |
| 2.3.3 | 벡터의 색인(indexing) | 112 |

| | |
|--------------------------------------|-----|
| <i>Contents</i> | v |
| 2.3.4 벡터 관련 함수 | 116 |
| 2.4 리스트(list) | 128 |
| 2.4.1 리스트 생성 | 130 |
| 2.4.2 리스트 색인 | 133 |
| 2.4.3 리스트에 원소 추가/제거 | 138 |
| 2.4.4 리스트의 결합 | 141 |
| 2.5 행렬(matrix) | 143 |
| 2.5.1 행렬의 연산 | 148 |
| 2.5.2 행렬의 색인 | 163 |
| 2.5.3 행과 열 추가 및 제거 | 169 |
| 2.5.4 행렬 관련 함수 | 172 |
| 2.5.5 벡터와 행렬의 차이점 | 178 |
| 2.5.6 의도치 않은 차원축소 피하기 | 179 |
| 2.6 배열(array) | 180 |
| 2.6.1 배열의 생성 및 색인 | 181 |
| 2.6.2 배열의 확장 예제 | 184 |
| 2.7 요인(factor)과 테이블(table) | 189 |
| 2.7.1 요인(factor) | 189 |
| 2.7.2 테이블(table) | 198 |
| 2.8 데이터 프레임(data frame) | 211 |
| 2.8.1 데이터 프레임 생성 | 212 |

| | | |
|----------|--------------------------------------|------------|
| 2.8.2 | 데이터 프레임 접근 및 필터링 | 220 |
| 2.8.3 | 데이터 프레임 관련 함수 | 229 |
| 2.8.4 | *apply() 계열 함수 | 239 |
| 2.9 | 객체의 유형 판별 및 변환 | 254 |
| 3 | 문자열 처리와 정규표현식 | 261 |
| 3.1 | 유용한 문자열 관련 함수 | 264 |
| 3.1.1 | nchar() | 264 |
| 3.1.2 | paste(), paste0() | 266 |
| 3.1.3 | sprintf() | 269 |
| 3.1.4 | substr() | 273 |
| 3.1.5 | tolower(), toupper() | 274 |
| 3.1.6 | glue 패키지를 활용한 문자열 다루기 | 275 |
| 3.2 | 정규표현식 기본 함수 | 279 |
| 3.2.1 | grep(), grep1() | 279 |
| 3.2.2 | regexp(), gregexpr() | 282 |
| 3.2.3 | sub(), gsub() | 286 |
| 3.2.4 | regexec() | 288 |
| 3.2.5 | strsplit() | 293 |
| 3.3 | 정규 표현식(regular expression) | 294 |
| 3.3.1 | 기본 메타 문자 | 295 |
| 3.3.2 | 문자 집합 | 304 |

| | |
|--|------------|
| <i>Contents</i> | vii |
| 3.3.3 문자 클래스 | 307 |
| 3.3.4 정규 표현식 예시 | 309 |
| 4 R 수학 함수, 분포 함수, 모형식 표현 | 317 |
| 4.1 수학함수 | 317 |
| 4.2 통계 분포 함수 | 323 |
| 4.3 모형식 표현 | 327 |
| 5 R Markdown | 1 |
| 5.1 R Markdown의 구성 | 1 |
| 5.2 R Markdown 기본 문법(syntax) | 4 |
| 5.2.1 텍스트 문법 | 6 |
| 5.2.2 Block-level elements | 10 |
| 5.2.3 수식표현(math expression) | 12 |
| 5.3 R Code Chunks | 14 |
| 5.4 인라인(inline) R 코드 | 32 |
| 5.5 YAML | 33 |
| 5.6 참고문헌 인용 | 35 |
| 6 R 외부 데이터 입출력 | 1 |
| 6.1 텍스트 파일 입출력 | 2 |
| 6.2 R 바이너리(binary) 파일 입출력 | 9 |
| 6.3 Excel 파일 입출력 | 13 |

| | |
|---|----------|
| 7 제어문(Control Structure) | 1 |
| 7.1 조건문(Conditionals) | 2 |
| 7.1.1 기본 구문 | 3 |
| 7.1.2 연쇄 조건문(chained condition) | 5 |
| 7.1.3 중첩 조건문(nested condition) | 7 |
| 7.1.4 <code>ifelse()</code> 함수 | 10 |
| 7.2 반복문(Looping) | 11 |
| 7.2.1 <code>repeat</code> 구문 | 12 |
| 7.2.2 <code>while</code> 구문 | 15 |
| 7.2.3 <code>for</code> 구문 | 19 |
| 7.3 함수 (function) | 24 |
| 7.3.1 함수의 정의 | 25 |
| 7.3.2 함수의 인수 전달 방법 | 30 |
| 7.3.3 함수의 기본 구성 요소 | 34 |
| 7.3.4 함수의 적용 범위(scoping rule) | 46 |

List of Tables

| | |
|-------------------------------------|-----|
| 1.1 R help 관련 명령어 리스트 | 18 |
| 2.1 R 예약어 종류 및 설명 | 74 |
| 2.2 R언어의 기본 수치 연산자 | 82 |
| 2.3 R언어의 논리형 연산자 | 86 |
| 2.4 R언어의 비교 연산자 | 87 |
| 2.5 리스트 데이터 접근 방법 | 134 |
| 2.6 스프레드시트 기본 형태 예시 | 212 |
| 2.7 R 객체 타입 판별 및 변환 함수 | 255 |
| 3.1 정규표현식 메타 문자: 기본 | 295 |
| 3.2 정규표현식 메타 문자: 문자집합 | 304 |
| 3.3 정규표현식 주요 문자 클래스 | 307 |
| 3.4 정규표현식: POSIX 문자 클래스 | 308 |
| 4.1 일반적인 R 통계 분포함수(일부 제외) | 325 |
| 5.1 코드 실행 관련 청크 | 15 |

| | |
|---------------------------------|----|
| 5.2 소스 코드 출력 결과 관련 첨크 | 18 |
| 5.3 코드 서식 관련 첨크 | 25 |
| 5.4 Plot 출력 관련 첨크 | 28 |

List of Figures

| | | |
|------|---|----|
| 1.1 | Windows에서 R 실행화면(콘솔 창, SDI 모드) | 13 |
| 1.2 | 정규분포 100개의 히스토그램 | 18 |
| 1.3 | cars 데이터셋의 speed와 dist 간 2차원 산점도: speed는 자동차 속도(mph)이고 dist는 해당 속도에서 브레이크를 밟았을 때 멈출 때 까지 걸린 거리(ft)를 나타냄. | 21 |
| 1.4 | RStudio 화면구성: 우하단 그림은 http://r-statistics.co/Top50-Ggplot2-Visualizations-MasterList-R-Code.html 에서 발췌 | 26 |
| 1.5 | RStudio 콘솔창에서 명령어 실행 후 출력결과 화면 | 26 |
| 1.6 | RStudio 스크립트 새로 열기 | 27 |
| 1.7 | RStudio Environment 창 객체 상세 정보 및 스프레드 시트 출력 결과 | 28 |
| 1.8 | R General option 팝업 창 | 33 |
| 1.9 | R Markdown의 최종 결과물 산출과정(http://applied-r.com/project-reporting-template/) | 61 |
| 1.10 | test.html 문서 화면(저장 폴더 내 ‘test.html‘을 크롬 브라 우저로 실행) | 63 |

| | |
|---|-----|
| 2.1 R 데이터 탑 구조 다이어그램: [R, Python 분석과 프로그래밍 (by R Friend)](http://rfriend.tistory.com/)에서 발췌 후 수정 | 73 |
| 2.2 https://www.geeksforgeeks.org/matlab-rgb-image-representation/ 에서 발췌 | 185 |
| 5.1 R markdown 세계(https://ulyngs.github.io/rmarkdown-workshop-2019 에서 발췌) | 2 |
| 5.2 R markdown structure | 5 |
| 5.3 R Markdown의 최종 결과물 산출과정(http://applied-r.com/project-reporting-template/) | 6 |
| 5.4 장난꾸러기 | 8 |
| 5.5 Chunk anatomy (https://ulyngs.github.io/rmarkdown-workshop-2019 에서 발췌) | 15 |
| 5.6 청크 옵션 results = 'markup'인 경우 rmd vs. md 파일 비교 . | 21 |
| 5.7 청크 옵션 results = 'asis'인 경우 rmd vs. md 파일 비교 . | 21 |
| 5.8 Taj Mahal | 30 |
| 5.9 Scatterplot of the car dataset | 31 |
| 7.1 Flow-control example (https://homerhanumat.github.io/r-notes/flow.html) | 1 |
| 7.2 if 구문 기본 flow-chart | 3 |
| 7.3 대안실행(if-else 구문) flow-chart | 6 |
| 7.4 연쇄조건(if-else if-else 구문) flow-chart | 7 |

List of Figures

xiii

| | |
|------------------------------------|----|
| 7.5 중첩 조건문 flow-chart | 8 |
| 7.6 REPEAT 구문 flow-chart | 13 |
| 7.7 WHILE 구문 flow-chart | 16 |
| 7.8 FOR 구문 flow-chart | 20 |
| 7.9 함수 | 24 |
| 7.10 함수의 기본 구조 | 34 |



Course Overview



- 본 문서는 2021년도 1학기 충남대학교 정보통계학과에서 개설한 “통계 프로그래밍 언어” 강의를 위해 개발한 강의 노트임
- 주 단위로 업데이트 될 예정
- <https://zorba78.github.io/cnu-r-programming-lecture-note/> 에서 확인
- pdf 파일 다운로드가 가능하지만 권장하지는 않음.
- **Google Chrome** 또는 **Firefox** 브라우저 사용 권장
- **온라인 상태 유지** 필수

본 문서는 Yihui Xie가 개발한 **bookdown** 패키지 (Xie, 2016)를 활용하여 생성한 문서임. 충남대학교 정보통계학과 이상인 교수님의 2019년도 2학기 “통계패키지활용” 강의 자료 내용과 구성을 참고하여 작성함.

강의소개

R은 뉴질랜드 오클랜드 대학의 Robert Gentleman 과 Ross Ihaka 가 AT&T 벨 연구소에서 개발한 S 언어를 기반으로 개발한 GNU 환경의 통계 계산 및 프로그래밍 언어이다. 현재 R 소프트웨어는 통계학 뿐 아니라 데이터 과학을 포함한 의학, 생물학 등 다양한 분야에서 활용되고 있으며 특히 통계 소프트웨어 개발과 데이터 분석에 많이 활용되고 있다.

본 강의는 데이터 분석을 위한 R의 기초 문법과 통계학 입문에서 학습한 몇 가지 중요한 통계적 이론에 대한 시뮬레이션 방법을 다룬다. 아울러 R package를 활용한 데이터 핸들링 및 시각화 그리고 Rmarkdown을 활용한 재현가능(reproducible)한 문서 작성법에 대해 학습하고자 한다.

교과 목표

- R 기초 문법 습득
- R 프로그래밍 능력 향상
- R 시뮬레이션을 통한 통계학 기초 이론 확인
- R markdown을 이용한 재현가능(reproducible)한 보고서 작성 방법 이해

선수과목

통계학 개론, 통계수학 1/통계수학 2 (필수는 아님)

수업 방법

- 강의: 40 %
- 실험/실습: 60%

평가방법

- **기말고사: 70 %**
- **출석: 10 %**
- **과제: 10 %**
- **퀴즈: 10 %**

교재 및 참고문헌

별도의 교재 없이 본 강의 노트로 수업을 진행할 예정이며, 수업의 이해도 향상을 위해 아래 소개할 도서 및 웹 문서 등을 참고할 것을 권장함.

참고문헌

- 빅데이터 분석 도구 R 프로그래밍 ([매트로프, 2012](#))
- 실리콘밸리 데이터과학자가 알려주는 따라하며 배우는 데이터 과학 ([권재명, 2017](#))
- R을 이용한 데이터 처리&분석 ([서민구, 2014](#))
- R for data science¹ ([Wickham and Gromelund, 2016](#))
- Statistical Computing with R ([Rizzo, 2019](#))
- R programming for data science² ([Peng, 2016](#))

¹<https://r4ds.had.co.nz/>

²<https://bookdown.org/rdpeng/rprogdatascience/>



Part I

Get Started



1

Introduction

1. R 프로그램

- 데이터 분석을 위한 자료 전처리, 통계 및 시각화를 지원하는 컴퓨터 언어 및 환경
- 1980년 AT&T 벨 연구소의 John Chambers가 개발한 S 언어를 기반으로 1995년 뉴질랜드 Auckland 대학의 통계학과 교수 Robert Gentleman과 Ross Ihaka 가 개발
- GNU¹ 기반의 오픈 소스
- 통계학, 전산학, 생물학, 의학 등 거의 모든 학문분야에서 분석도구로 활용되고 있고, 최근 data science 분야에서 널리 활용

2. R 언어의 특징

- 무료 소프트웨어
- CRAN (Comprehensive R Archive Network)²에서 배포
- 특정 vendor가 아닌 전 세계 연구자들이 개발한 알고리즘 및 최신 함수 활용 가능(packaging system)
- 범용적으로 사용되는 거의 대부분의 운영체제(Windows, Mac, Linux)에서 작동 가능

¹https://en.wikipedia.org/wiki/GNU_Project

²<https://cran.r-project.org/>

- 방대한 개발 및 사용 생태계 형성
- 강력한 그래픽 기능



유용한 웹 사이트: R과 관련한 거의 모든 문제는 Googling (구글을 이용한 검색)을 통해 해결 가능(검색주제 + “in R” or “in R software”)하고 많은 해답들이 아래 열거한 웹 페이지에 게시되어 있음.

- R 프로그래밍에 대한 Q&A: Stack Overflow³
- R 관련 웹 문서 모음: Rpubs⁴
- R package에 대한 raw source code 제공: Github⁵
- R을 이용한 통계 분석: Statistical tools for high-throughput data analysis (STHDA)⁶

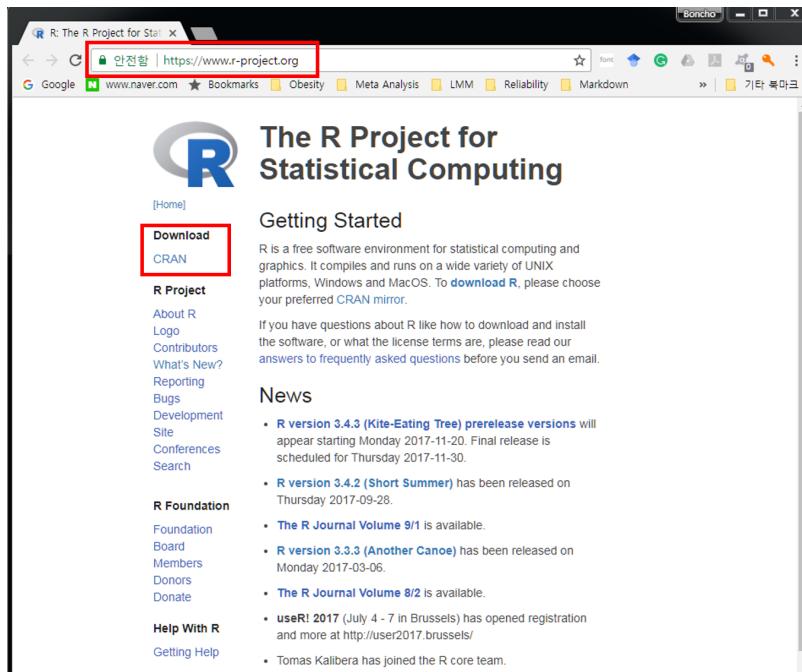
1.1 R 설치하기

R 다운로드 사이트: <https://www.r-project.org> 또는 <https://cran.r-project.org>

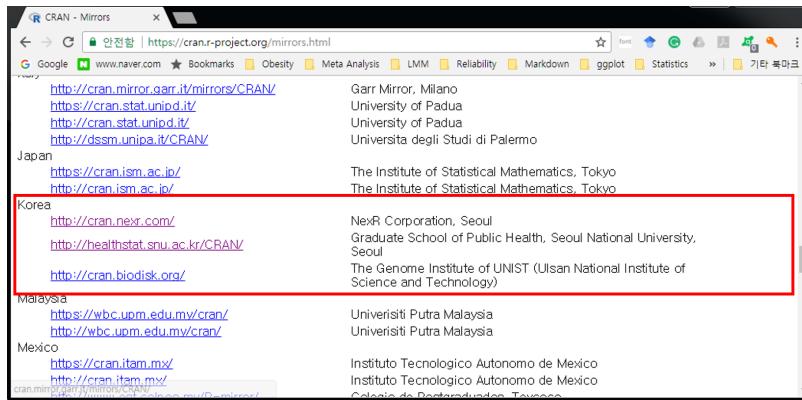
1. 웹 브라우저(i.e. Explore, Chrome, Firefox 등)의 주소 입력창에
<https://www.r-project.org>
2. 좌측 R Logo 하단 Download 아래 CRAN 클릭

1.1 R 설치하기

5



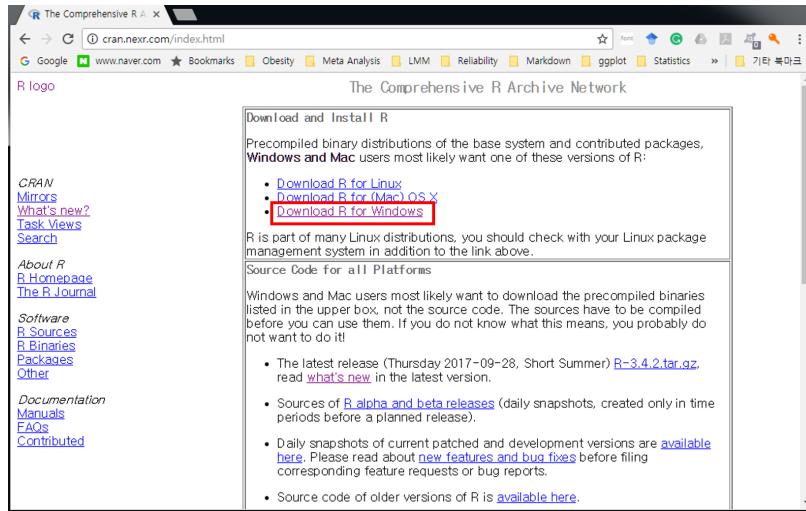
3. 클릭 후 연결한 페이지를 스크롤 후 Korea 아래 링크⁷ 클릭



4. 클릭 후 세 가지 운영체제(Linux, Mac OS X, Windows)에 따른 R 버전 선택 가능⁸

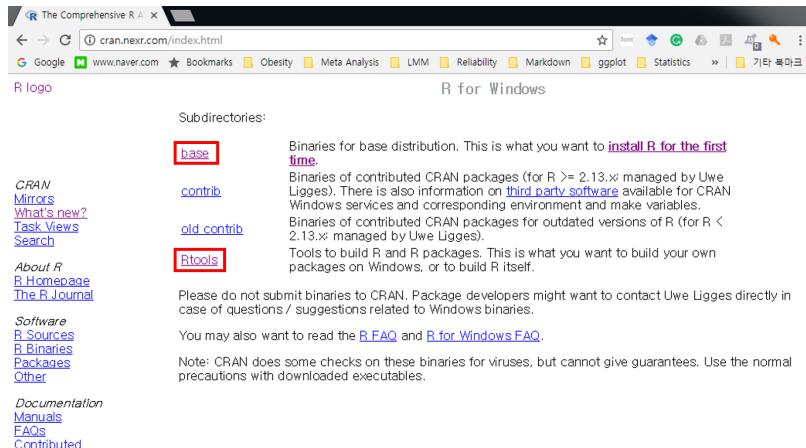
⁷해당 링크들은 접속 시점에 따라 변경될 수 있음

⁸본 노트는 Windows 버전 설치만 다룸



5. Downloads R for Windows 링크 클릭하면 다음과 같은 화면으로

o]동

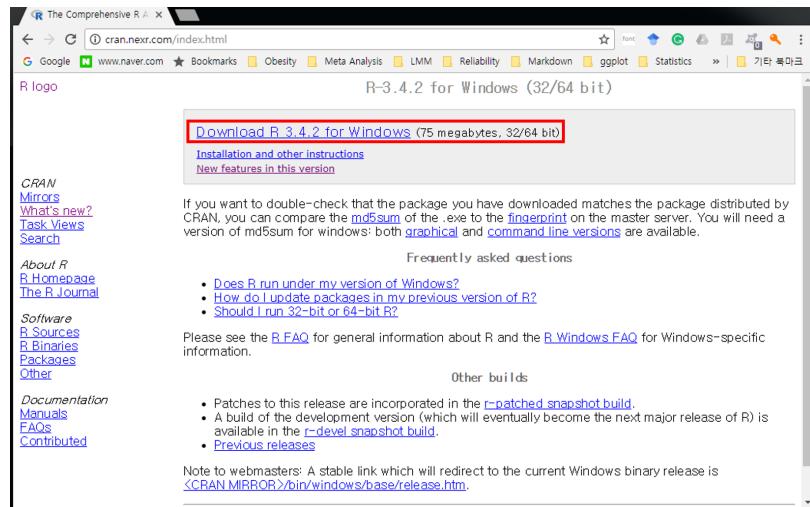


다음 하위폴더에 대한 간략 설명

- **base:** R 실행 프로그램

- **contrib:** R package의 바이너리 파일
- **Rtools:** R package 개발 및 배포를 위한 프로그램

6. 위 화면에서 **base** 링크 클릭 후 아래 화면에서 **Downloads R 3.x.x for Windows** 를 클릭 후 설치 파일을 임의의 디렉토리에 저장 및 실행

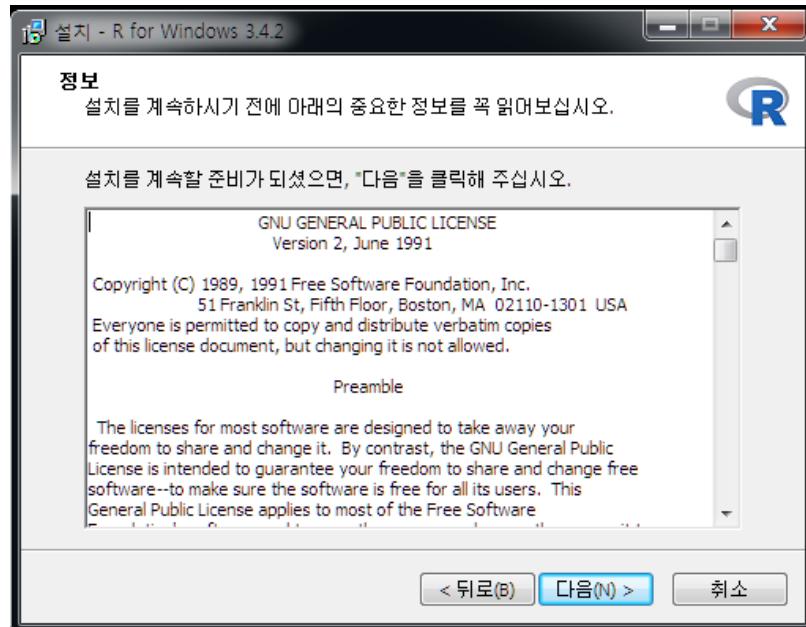


7. 다운로드한 파일을 실행하면 아래와 같은 대화창이 나타남

- 한국어 선택 → 환영 화면에서 [다음(N)>] 클릭



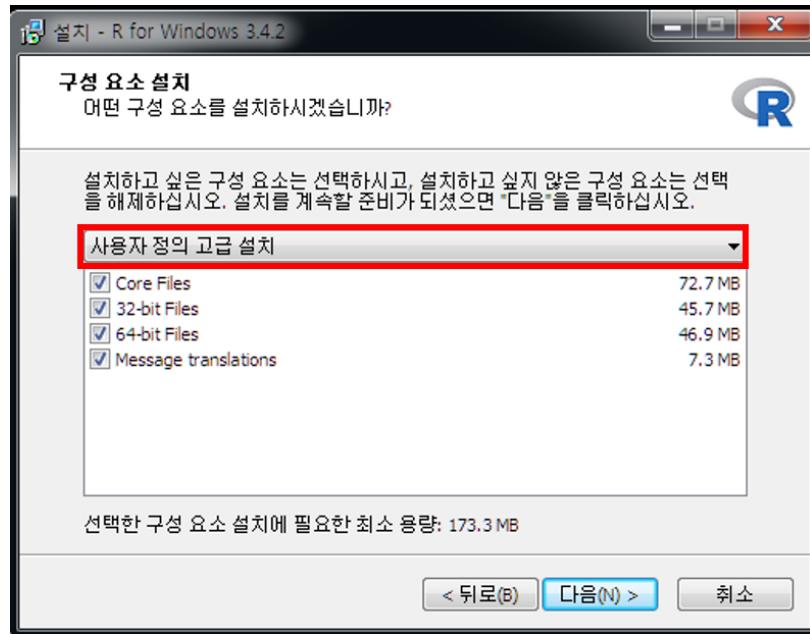
8. GNU 라이센스에 대한 설명 및 동의 여부([다음(N)>]) 클릭



9. 설치 디렉토리 설정 및 구성요소 설치 여부

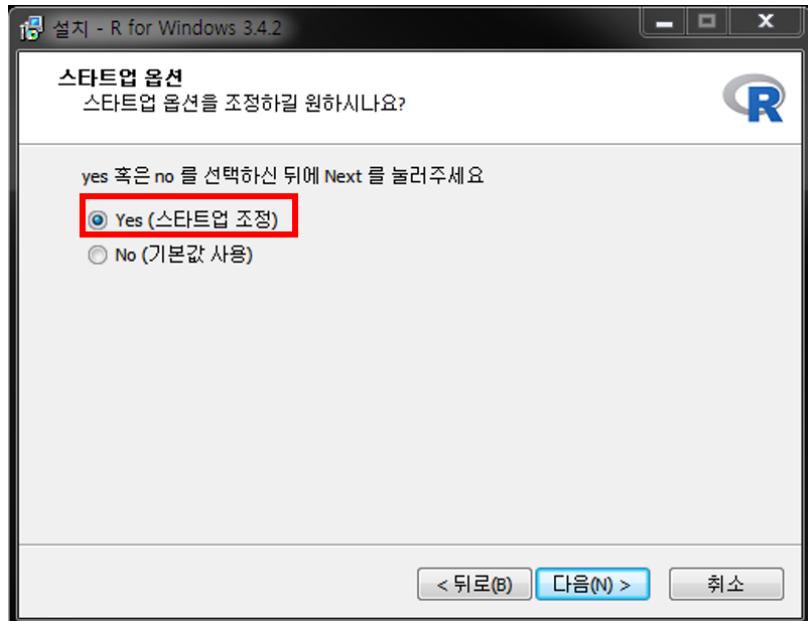
- 원하는 디렉토리 설정(예: C:\R\R-3.x.x)
- 기본 프로그램("Core Files"), 32 또는 64 bit 용 설치 파일, R console 한글 번역 모두 체크 뒤 [다음(N)>] 클릭





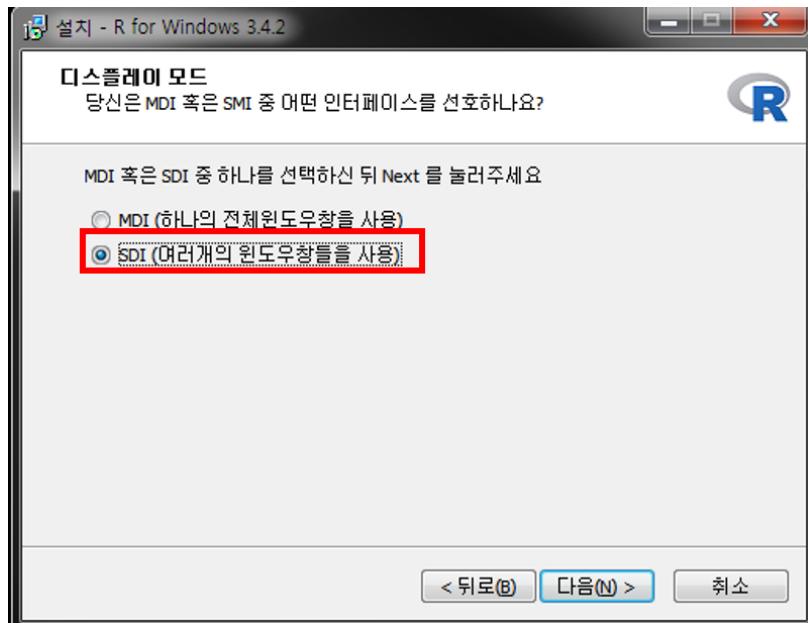
10. R 스타트업 옵션 지정

- 기본값("No" check-button)으로도 설치 진행 가능
- 본 문서에서는 스타트업 옵션 변경으로 진행

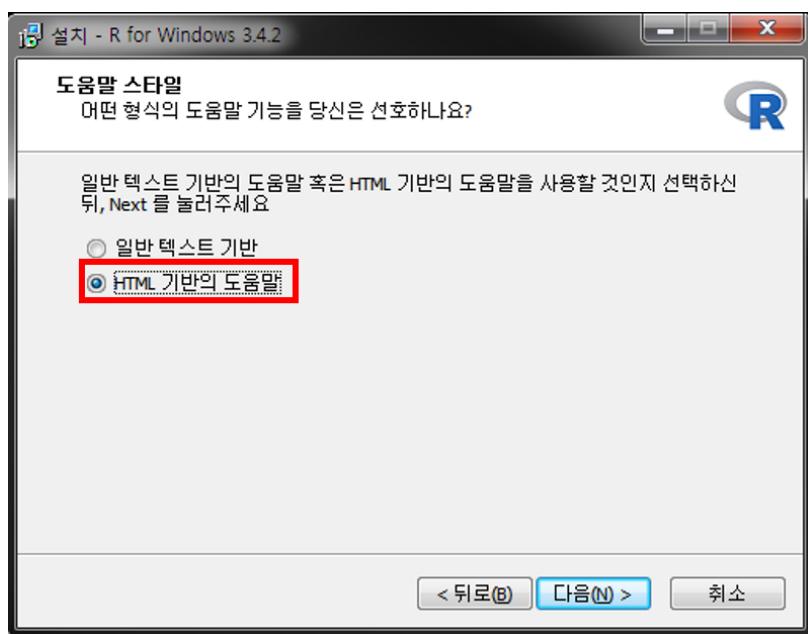


11. 화면표시방식(디스플레이 모드) 설정 변경

- MDI: 한 윈도우 내에서 script 편집창, 출력, 도움말 창 사용
- SDI: 다중 창에서 각각 script 편집창, 출력, 도움말 등을 독립적으로 열기

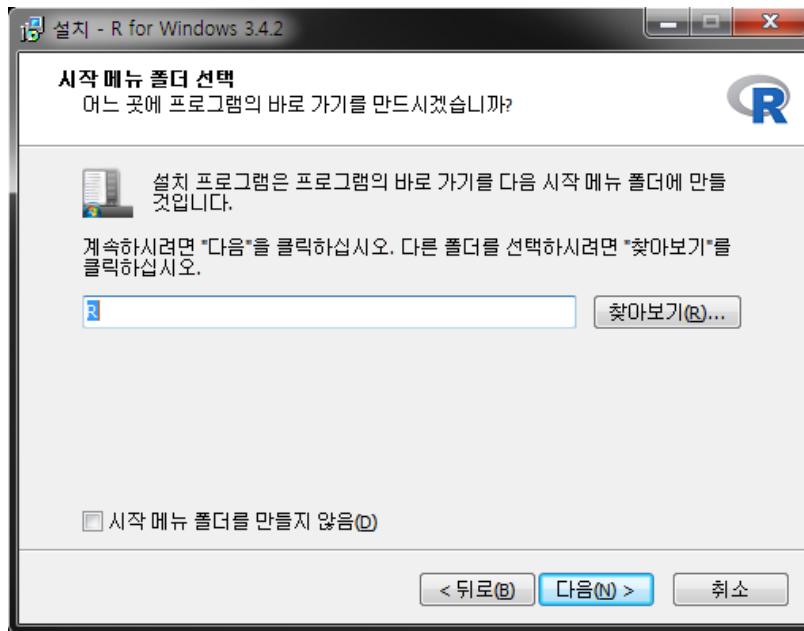


12. 도움말 형식에서 HTML 도움말 기반 선택



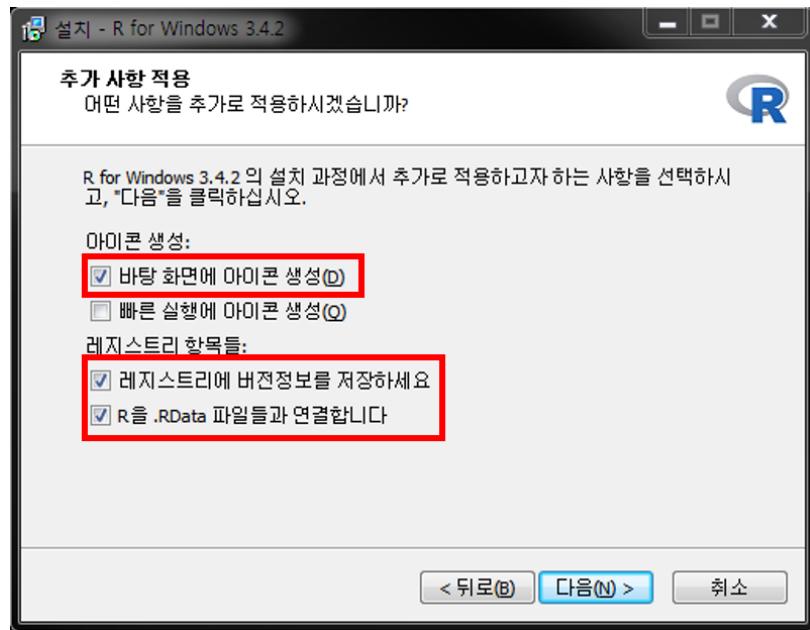
13. 시작메뉴 폴더 선택

- “바로가기”를 생성할 시작 메뉴 폴더 지정 후 [다음(N)>] 클릭 후 설치 진행
- 하단 “시작메뉴 폴더 만들지 않음” 체크박스 표시 시 시작메뉴에 “바로가기” 아이콘이 생성되지 않음(실행에 전혀 지장 없음)



14. 추가 옵션 지정: 바탕화면 아이콘 생성 등 추가적 작업 옵션 체크 후 [다음(N)>] 클릭 → 설치 진행

- 설치된 R 버전 정보 레지스트리 저장 여부
- .Rdata 확장자를 R 실행파일과 자동 연계



15. 설치 완료 후 바탕화면의 R 아이콘을 더블클릭하면 Rgui가 실행

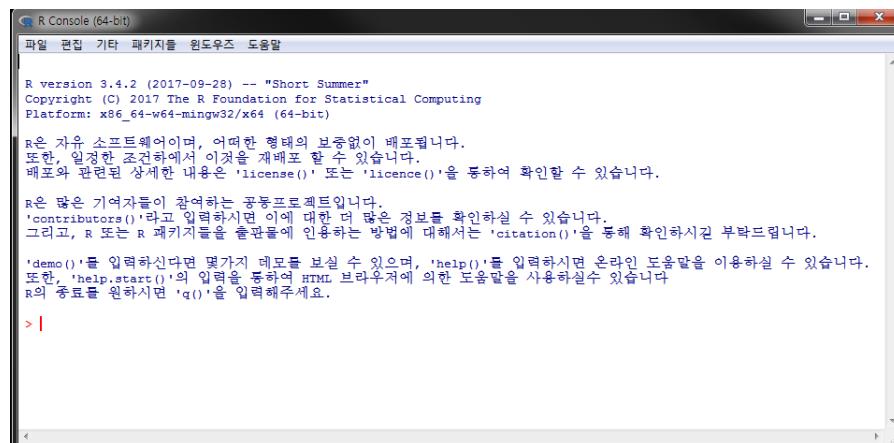


FIGURE 1.1: Windows에서 R 실행화면(콘솔 창, SDI 모드)

1.2 R 시작 및 작동 체크



실습: 설치된 R을 실행 후 보이는 R 콘솔(console) 창에서 명령어를 실행하고 결과 확인

Figure 1.1 에서 > 기호는 R의 명령 프롬프트(command prompt) 임

- → 컴퓨터가 사용자 명령을 기다리고 있다는 기호

1. 현재 R session⁹ 정보(R 설치 버전, locale, 로딩 packages) 출력

```
# R의 설치 버전 및 현재 설정된 locale(언어, 시간대) 및 로딩된 R package 정보 출력
sessionInfo()
```

```
R version 4.0.5 (2021-03-31)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 18.04.5 LTS

Matrix products: default
BLAS:    /usr/lib/x86_64-linux-gnu/openblas/libblas.so.3
LAPACK:  /usr/lib/x86_64-linux-gnu/libopenblas-r0.2.20.so

locale:
```

```
[1] LC_CTYPE=ko_KR.UTF-8      LC_NUMERIC=C
[3] LC_TIME=ko_KR.UTF-8       LC_COLLATE=ko_KR.UTF-8
```

⁹현재 실행되고 있는 R의 작업공간

```
[5] LC_MONETARY=ko_KR.UTF-8      LC_MESSAGES=ko_KR.UTF-8
[7] LC_PAPER=ko_KR.UTF-8        LC_NAME=C
[9] LC_ADDRESS=C                  LC_TELEPHONE=C
[11] LC_MEASUREMENT=ko_KR.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] stats      graphics   grDevices utils      datasets  methods   base

other attached packages:
[1] kableExtra_1.3.4  gtsummary_1.4.0  gt_0.2.2       glue_1.4.2
[5] forcats_0.5.1    stringr_1.4.0   dplyr_1.0.5     purrr_0.3.4
[9] readr_1.4.0      tidyverse_1.3.1 tibble_3.1.1     ggplot2_3.3.3
[13] tidyverse_1.3.1   rmarkdown_2.7   knitr_1.33

loaded via a namespace (and not attached):
[1] Rcpp_1.0.6          svglite_2.0.0      lubridate_1.7.10
[4] lattice_0.20-44     assertthat_0.2.1   digest_0.6.27
[7] utf8_1.2.1          R6_2.5.0          cellranger_1.1.0
[10] backports_1.2.1     reprex_2.0.0      evaluate_0.14
[13] httr_1.4.2          pillar_1.6.0      rlang_0.4.10
[16] readxl_1.3.1        rstudioapi_0.13   Matrix_1.3-2
[19] splines_4.0.5       webshot_0.5.2      munsell_0.5.0
[22] broom_0.7.6         compiler_4.0.5    modelr_0.1.8
[25] xfun_0.22           pkgconfig_2.0.3    systemfonts_1.0.1
[28] htmltools_0.5.1.1   tidyselect_1.1.0   bookdown_0.22
[31] fansi_0.4.2          viridisLite_0.4.0  crayon_1.4.1
[34] dbplyr_2.1.1         withr_2.4.2       grid_4.0.5
[37] jsonlite_1.7.2       gtable_0.3.0      lifecycle_1.0.0
[40] DBI_1.1.1            magrittr_2.0.1     scales_1.1.1
[43] cli_2.5.0             stringi_1.5.3    broom.helpers_1.3.0
[46] fs_1.5.0              xml2_1.3.2       ellipsis_0.3.1
[49] generics_0.1.0        vctrs_0.3.7       tools_4.0.5
[52] hms_1.0.0             survival_3.2-11  yaml_2.2.1
```

```
[55] colorspace_2.0-0      rvest_1.0.0          haven_2.4.1
```

2. 문자열 출력

```
#문자열 출력  
print("Hello R") #문자열
```

```
[1] "Hello R"
```

```
# 기호는 주석의 시작을 의미하고 실제로 실행되지 않음 같은 행에서 #  
뒤 내용의 코드 역시 실행되지 않음
```

3. a라는 변수에 숫자 9, b라는 변수에 숫자 7를 할당 후 출력

```
# 수치형 값(scalar)을 변수에 할당(assign)  
# 여러 명령어를 한줄에 입력할 때에는 세미콜론(;)으로 구분  
a = 9; b = 7  
a
```

```
[1] 9
```

```
b
```

```
[1] 7
```

4. 변수 a와 b의 사칙연산

```
a+b; a-b; a*b; a/b
```

```
[1] 16
```

```
[1] 2
```

```
[1] 63
```

```
[1] 1.285714
```

5. R 그래픽 맛보기: 정규분포로부터 난수 100개 생성 후 생성된 데이터에 대한 히스토그램 작성

```
# 난수 생성 시 같은 매번 달라지기 때문에 seed를 주어 일정값이 생성되도록 고정
# "="과 "<-"는 모두 동일한 기능을 가진 할당 연산자임
# 평균이 0이고 분산이 1인 정규분포에서 난수 100개 생성
set.seed(12345) # random seed 지정
x <- rnorm(100) # 난수 생성
hist(x) # 히스토그램
```



R 명령어 또는 전체 프로그램 소스 실행 시 매우 빈번히 오류가 나타나는데, 이를 해결할 수 있는 가장 좋은 방법은 앞에서 언급한 Google을 이용한 검색 또는 R 설치 시 자체적으로 내장되어 있는 도움말을 참고하는 것이 가장 효율적임.

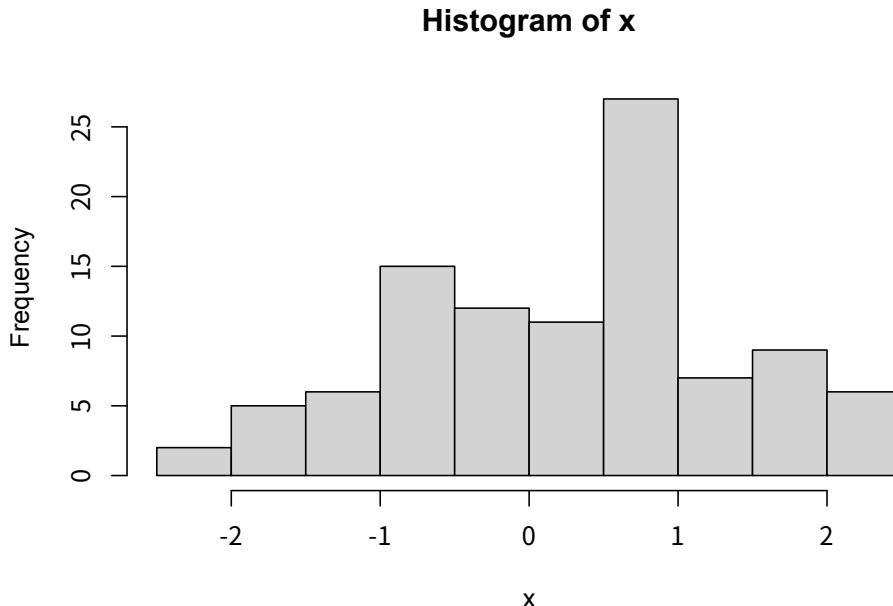


FIGURE 1.2: 정규분포 100개의 히스토그램

TABLE 1.1: R help 관련 명령어 리스트

| 도움말 보기 명령어 | 설명 | 사용법 |
|----------------------|---|------------------------|
| 'help' 또는 '?' | 도움말 시스템 호출 | 'help(함수명)' |
| 'help.search' 또는 '?' | 주어진 문자열을 포함한 문서 검색 | 'help.search(pattern)' |
| 'example' | topic의 도움말 페이지에 있는 examples section 실행 | 'example(함수명)' |
| 'vignette' | topic의 pdf 또는 html 레퍼런스 메뉴얼 불러오기 | 'vignette(패키지명 또는 패턴)' |



Vignette 의 활용: 데이터를 기반으로 사용하고자 하는 패키지의 실제 활용 예시를 작성한 문서이기 때문에 초보자들이 R 패키지 활용에 대한 접근성을 높혀줌.

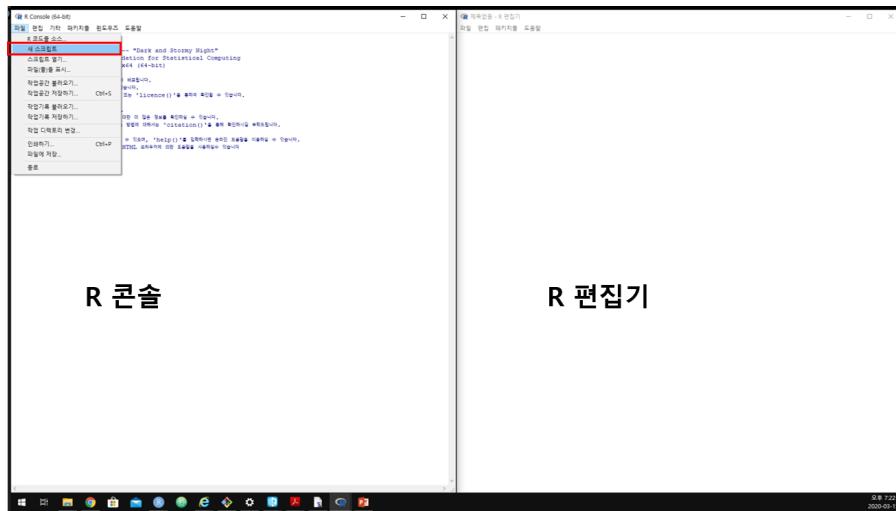
- vignette()
- browseVignettes()

1.3 R script 편집기 사용



실습: R 설치 후 Rgui에서 제공하는 편집기(R editor)에 명령어를 입력하고 실행

설치된 R을 실행 후 상단 pull-down 메뉴에서 [File] → [새 스크립트]를 선택하면 아래 그림과 같이 편집창(R 인스톨 시 SDI 옵션 기준)이 나타남



편집기 창에 다음 명령어 입력

```
# R에 내장된 cars 데이터셋 불러오기 | cars dataset에 포함된 변수들의 기초통계량
# 출력 2차원 산점도

data(cars)
help(cars) # cars 데이터셋에 대한 설명 help 창에 출력
head(cars) # cars 데이터셋 처음 6개 행 데이터 출력
summary(cars) # cars 데이터셋 요약
plot(cars) # 변수가 2개인 경우 산점도 출력
```

- 편집창에서 한 줄을 실행시키려면 명령어가 입력된 줄에서 [Ctrl] + [R] 입력
- 편집창에 입력한 모든 명령어를 실행시키려면 모든 줄을 선택(마우스 또는 [Shift] + ↓)

```
speed dist
1     4    2
2     4   10
3     7    4
4     7   22
5     8   16
6     9   10

      speed           dist
Min.   : 4.0   Min.   : 2.00
1st Qu.:12.0   1st Qu.: 26.00
Median :15.0   Median : 36.00
Mean   :15.4   Mean   : 42.98
3rd Qu.:19.0   3rd Qu.: 56.00
Max.   :25.0   Max.   :120.00
```

- R은 명령어를 입력하고 실행결과를 확인하는 대화형(interpreter) 방식

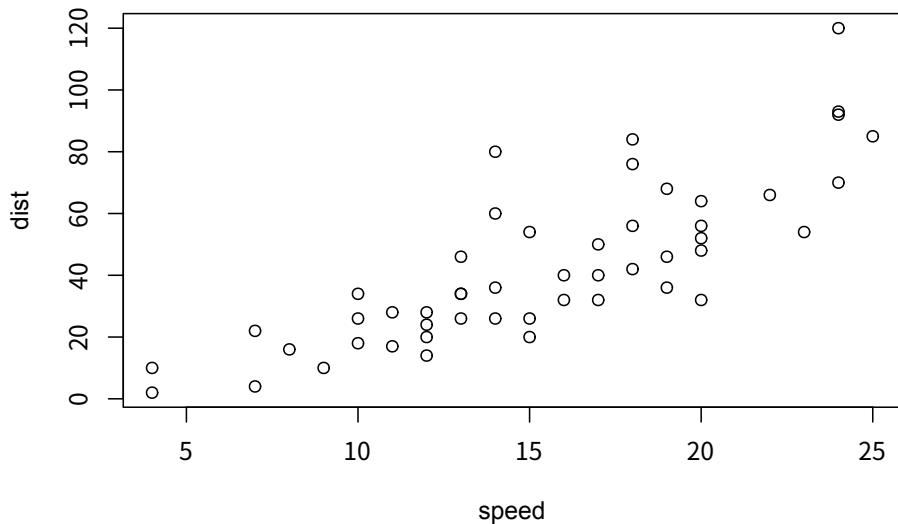


FIGURE 1.3: cars 데이터셋의 speed와 dist 간 2차원 산점도: speed는 자동차 속도(mph)이고 dist는 해당 속도에서 브레이크를 밟았을 때 멈출 때 까지 걸린 거리(ft)를 나타냄.

- 콘솔창에서 \uparrow/\downarrow 를 누르면 이전/이후 실행 명령 기록 확인 가능
- 여러 줄 이상 R 명령어라든가 반복적, 장기간 작업을 수행해야 할 경우 R 명령어로 구성된 스크립트 작성 후 일괄 실행하는 것이 일반적
- 여러 다중 명령 코딩 시 콘솔창에 직접 입력하는 것은 비효율적이므로 스크립트 에디터를 사용
- 위 예시처럼 R 에디터 사용할 수 있으나 가독성 및 코딩 효율이 떨어짐
- 과거 많이 사용됐던 R 에디터: WinEdt¹⁰, Tinn-R¹¹, Vim¹²
- 현재 가장 범용적 R 에디터: Rstudio

¹⁰<http://www.winedt.com>

¹¹<https://sourceforge.net/projects/tinn-r/>

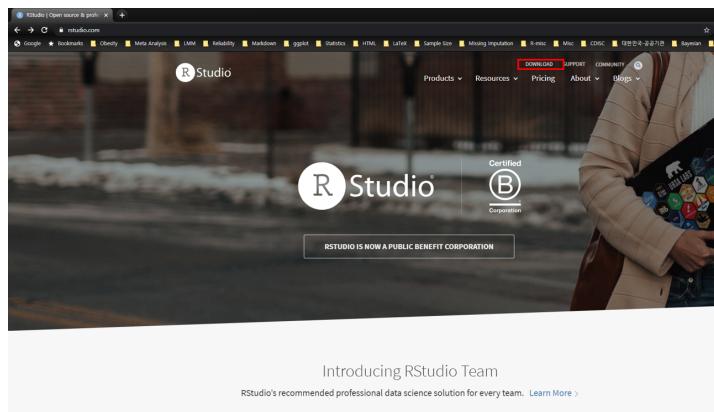
¹²http://www.vim.org/scripts/script.php?script_id=2628

1.4 RStudio

- RStudio¹³: R 통합 분석/개발 환경(integrated development environment, IDE)으로 현재 가장 대중적으로 사용되고 있는 R 사용 환경
- 명령 곤솔 외 파일 편집, 데이터 객체, 명령 기록(.history), 그래프 등에 쉽게 접근 가능
- RStudio 독자적인 개발 환경 제공: Rmarkdown, Rnotebook, Shiny Web Application 등 다양한 R 환경을 제공
- 버전관리(git, subversion)를 통해 project 관리 가능
- 무료 및 유료 소프트웨어 제공

1.4.1 RStudio 설치하기

1. 웹 브라우저를 통해 <https://rstudio.com> 접속 후 상단 DOWNLOAD¹⁴ 링크 클릭



¹³<https://rstudio.com/>

¹⁴<https://rstudio.com/products/rstudio/download/>

2. Desktop 또는 Server 버전 중 택일

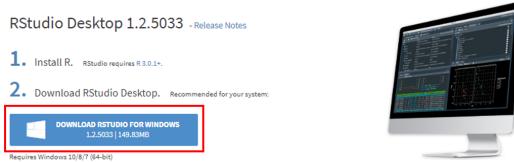
- 서버용 설치를 위해서는 Server 클릭 → 소규모 자료 분석용으로는 불필요
- 여기서는 **Desktop** 버전 선택 후 다음 링크로 이동

The screenshot shows the RStudio download page. At the top, there's a navigation bar with links for DOWNLOAD, SUPPORT, and COMMUNITY. Below the navigation, a large blue banner says "Download RStudio". Underneath the banner, there's a section titled "Choose Your Version" with a brief description of RStudio's features. To the right of this description is a box for the "RStudio Team" which includes a team logo and a brief overview of the product. Below these sections, there are four main download/buy options:

| RStudio Desktop | RStudio Desktop | RStudio Server | RStudio Server Pro |
|---------------------|--------------------|---------------------|----------------------------------|
| Open Source License | Commercial License | Open Source License | Commercial License |
| Free | \$995 /year | Free | \$4,975 /year (5 Named Users) |
| DOWNLOAD | BUY | DOWNLOAD | BUY |

Under each option, there are "Learn more" links. The "DOWNLOAD" button for the free desktop version is highlighted with a red box.

3. 운영체제에 맞는 Rstudio installer 다운로드(여기서는 Windows 버전 다운로드)



All Installers

Linux users may need to import RStudio's public code-signing key prior to installation, depending on the operating system's security policy.

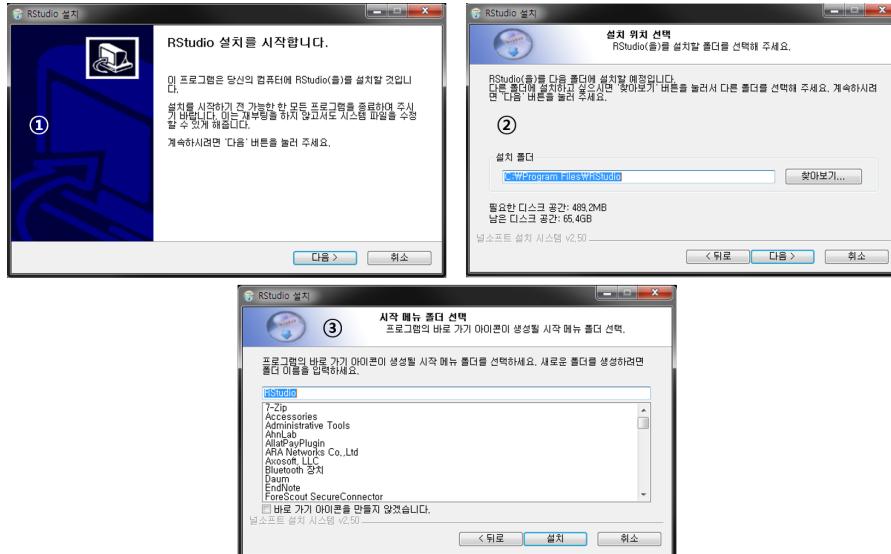
RStudio 1.2 requires a 64-bit operating system. If you are on a 32 bit system, you can use an older version of RStudio.

| OS | Download | Size | SHA-256 |
|---------------------|---|-----------|----------|
| Windows 10/8/7 | RStudio-1.2.5033.exe | 149.83 MB | 77d08c1b |
| macOS 10.12+ | RStudio-1.2.5033.dmg | 128.89 MB | b07v9075 |
| Ubuntu 14/Debian 8 | rstudio-1.2.5033-amd64.deb | 96.18 MB | 05cc6e22 |
| Ubuntu 16 | rstudio-1.2.5033-amd64.deb | 104.14 MB | a1591ed7 |
| Ubuntu 18/Debian 10 | rstudio-1.2.5033-amd64.deb | 108.21 MB | 05ea4295 |
| Fedor 18/Red Hat 7 | rstudio-1.2.5033-x86_64.rpm | 120.23 MB | 35e14bd8 |
| Fedor 28/Red Hat 8 | rstudio-1.2.5033-x86_64.rpm | 120.87 MB | 4f2b1d0d |

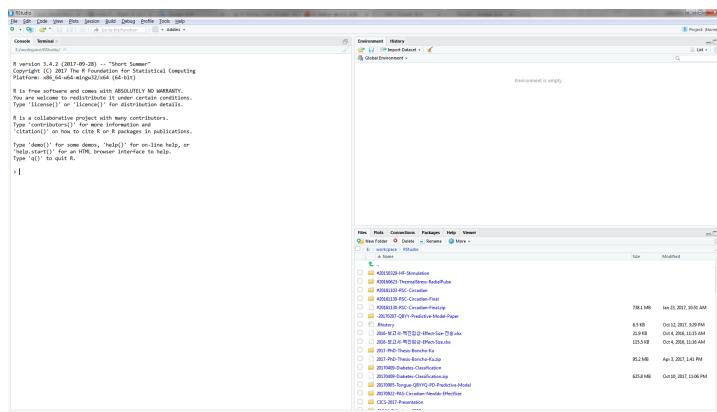
4. RStudio installer 다운로드 시 파일이 저장된 폴더에서 보통

RStudio-xx.xx.xxx.exe 형식의 파일명 확인

- 더블 클릭 후 실행
- [다음>] 몇 번 클릭 후 설치 종료



5. 바탕화면 혹은 시작 프로그램에 새로 설치된 RStudio 아이콘 클릭 후 아래와 같은 프로그램 창이 나타나면 설치 성공



1.4.2 RStudio IDE 화면 구성

RStudio는 아래 그림과 같이 4개 창으로 구성¹⁵

1. 콘솔(console)

- R 명령어 실행공간(RGui, 정확하게는 R 설치 디렉토리에서 “~/R/R.x.x/bin/x64/Rterm.exe” 가 구동되고 있는 공간)
- R script 또는 콘솔 창에서 작성한 명령어(프로그램) 실행 및 그 결과 출력
- 경고, 에러/로그 등의 메세지 확인

2. 스크립트(script) (Figure 1.6)

- R 명령어 입력 공간으로 일괄처리(batch processing) 가능

¹⁵각 창의 위치는 세팅 구성에 따라 달라질 수 있음. 창 구성 방법은 RStudio 환경 옵션 설정에서 설명함.

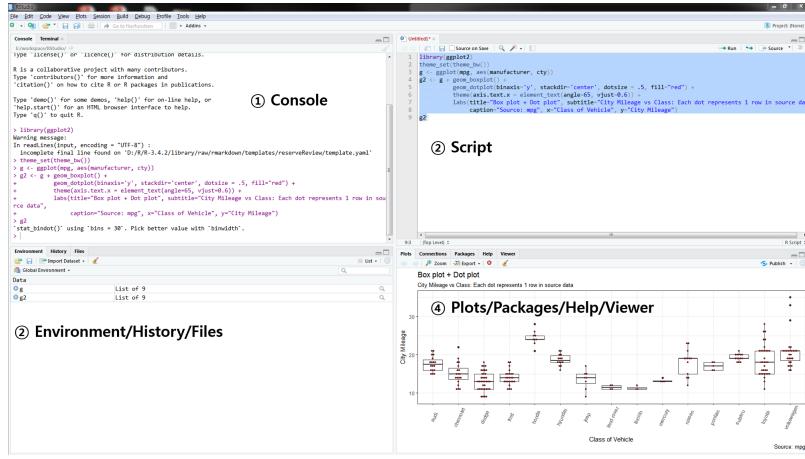


FIGURE 1.4: RStudio 화면구성: 우하단 그림은 <http://r-statistics.co/Top50-Ggplot2-Visualizations-MasterList-R-Code.html>에서 발췌

```
Console | R Markdown | D:/Current-Workspace/Lecture/cmu-r-programming-lecture-note/ | ↻
R version 3.6.2 (2019-12-12)
Platform: x86_64-w64-mingw32/x64 (64-bit)
Running under: Windows 10 x64 (build 18363)

Matrix products: default
locale:
[1] LC_COLLATE=Korean_Korea.949  LC_CTYPE=Korean_Korea.949   LC_MONETARY=Korean_Korea.949 LC_NUMERIC=C
[5] LC_TIME=Korean_Korea.949

attached base packages:
[1] stats      graphics    grDevices  utils      datasets   methods    base

other attached packages:
[1] kableExtra_1.1.0 knitr_1.28   rmarkdown_2.1    forcats_0.4.0  stringr_1.4.0  dplyr_0.8.3   purrr_0.3.3
[8] readr_1.3.1     tibble_2.1.3   ggplot2_3.2.1   tidyverse_1.3.0

loaded via a namespace (and not attached):
[1] digest_0.6.23  stringi_1.4.3  grid_3.6.2     reshape_0.8.8  gtable_0.2.0
[8] assertthat_0.2.1 gridExtra_2.3.0  lattice_0.20-38 colorspace_1.4-1  vctrs_0.2.1  generics_0.0.2
[15] glue_1.3.1     DBI_1.1.0     dplyr_1.4.2   modelr_0.1.5  readxl_1.3.1  utf8_1.1.4
[22] gttable_0.3.0   cellranger_1.1.0 rvest_0.3.5   evaluate_0.14  fansi_0.4.1   rlang_0.4.4
[29] backports_1.1.5 scales_1.1.0    webshot_0.5.2 jsonlite_0.5.2  fs_1.3.1    pillar_1.4.3
[36] digest_0.6.23  stringi_1.4.5  bookdown_0.16  grid_3.6.2    cli_2.0.1    lifecycle_0.1.0
[43] lazyeval_0.2.2  crayon_1.3.4   pkgconfig_2.0.3 gridExtra_0.1.0  broom_0.5.3   munspell_0.5.0
[50] lubridate_1.7.4 assertthat_0.2.1   httr_1.4.1    zeallot_0.1.0  hms_0.5.2    Rcpp_1.0.3
[50] rstudioapi_0.10 R6_2.4.1     rmarkdown_0.10  xrlat_1.2.2   tools_3.6.2   packrat_0.5.0
[50] compiler_3.6.2
```

FIGURE 1.5: RStudio 콘솔창에서 명령어 실행 후 출력결과 화면

- 새로운 스크립트 창 열기

- 아래 그림과 같이 pull-down 메뉴 좌측 상단 아이콘 클릭 후 [R script] 선택
- [File] → [New File] → [R Script] 선택
- 단축 키: [Ctrl] + [Shift] + [N]

- 일괄 명령어 처리를 위한 RStudio 제공 단축 키

- [Ctrl] + [Enter]: 선택한 블럭 내 명령어 실행
- [Alt] + [Enter]: 선택 없이 커서가 위치한 라인의 명령어 실행
- R 스크립트 이외 R Markdown, R Notebook, Shiny web application 등 새 문서의 목적에 따라 다양한 종류의 소스 파일 생성 가능
- 저장된 R 스크립트 파일은 파일명.R로 저장됨
- 파일 실행 방법
 - 실행하고자 하는 파일을 읽은 후 ([File] → [Open File] + 파일명 선택 또는 파일명.R 더블 클릭) 입력된 모든 라인을 선택한 뒤 [Ctrl] + [Enter]
 - 파일 읽은 후 [Ctrl] + [Shift] + [S] (현재 열려있는 *.R 파일에 대해) 또는 [Ctrl] + [Shift] + [Enter]

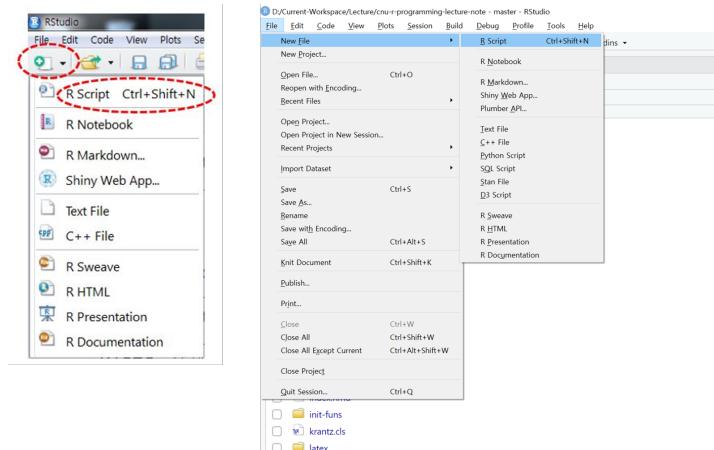


FIGURE 1.6: RStudio 스크립트 새로 열기



RStudio는 코딩 및 소스 작성의 효율성을 위해 여러 가지 단축 키를 제공하고 있음.
단축키는 아래 그림과 같이 pull down 메뉴 [Tools] 또는 [Help]에서 [Keyboard shortcut help] 또는 [Alt] + [Shift] + [K] 단축키를 통해 확인할 수 있음.
또는 Rstudio cheatsheet에서 단축키에 대한 정보를 제공하는데 pull down 메뉴

[Help] → [Cheatsheets] → [RStudio IDE Cheat Sheet]을 선택하면 각 아이콘 및 메뉴 기능에 대한 개괄적 설명 확인 가능함.

3. 환경/명령기록(Environment/History) (Figure 1.7)

- **Environment:** 현재 R 작업환경에 저장되어 있는 객체의 특성 및 값 등을 요약 제시
 - 좌측 아래 화살표 버튼 클릭: 해당 객체의 상세 정보 확인
 - 우측 사각형 버튼 또는 객체(데이터셋명) 클릭: 객체가 데이터셋(데이터프레임)인 경우 스프레드 시트 형태로 데이터셋 확인

The figure consists of four screenshots of the RStudio Environment pane, each showing a different way to view objects in the Global Environment:

- Screenshot 1 (Top Left):** Shows the standard list of objects: cars, mpg, and tab. The 'cars' object is selected and highlighted with a red box.
- Screenshot 2 (Top Right):** Shows the same list, but the 'cars' object has been clicked, revealing its detailed structure: 50 obs. of 2 variables, speed: num 4 4 7 7 8 9 10 10 10 11 ..., dist: num 2 10 4 22 16 10 18 26 34 17 ...
- Screenshot 3 (Bottom Left):** Shows the list again with 'cars' selected. A red box highlights the 'cars' entry.
- Screenshot 4 (Bottom Right):** Shows the detailed structure of the 'cars' dataset as a spread sheet. The first few rows are:

| | speed | dist |
|----|-------|------|
| 1 | 4 | 2 |
| 2 | 4 | 10 |
| 3 | 7 | 4 |
| 4 | 7 | 22 |
| 5 | 8 | 16 |
| 6 | 9 | 10 |
| 7 | 10 | 18 |
| 8 | 10 | 26 |
| 9 | 10 | 34 |
| 10 | 11 | 17 |

FIGURE 1.7: RStudio Environment 창 객체 상세 정보 및 스프레드 시트 출력 결과

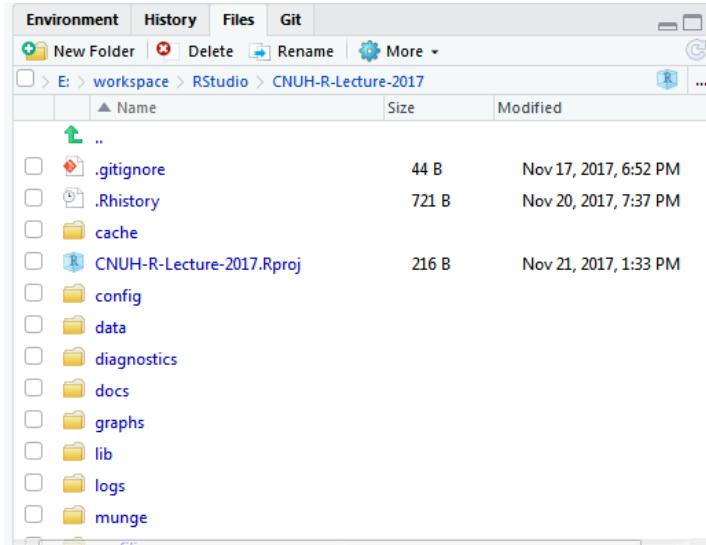
- History: R 콘솔에서 실행된 명령어(스크립트)들의 이력 확인



```
Environment History Connections Git
To Console To Source ⚡ 🔍
ReadExcel <- function(filename) {
  require(XLConnect)
  require(plyr)
  WB <- loadWorkbook(filename)
  SheetName <- getSheets(WB)
  DF1 <- llply(SheetName, function(name) readWorksheet(WB, sheet=name))
  names(DF1) <- SheetName
  return(DF1)
}
ReadExcel <- function(filename) {
  require(XLConnect)
  require(plyr)
  WB <- loadWorkbook(filename)
  SheetName <- getSheets(WB)
  DF1 <- llply(SheetName, function(name) readWorksheet(WB, sheet=name))
  names(DF1) <- SheetName
  return(DF1)
}
```

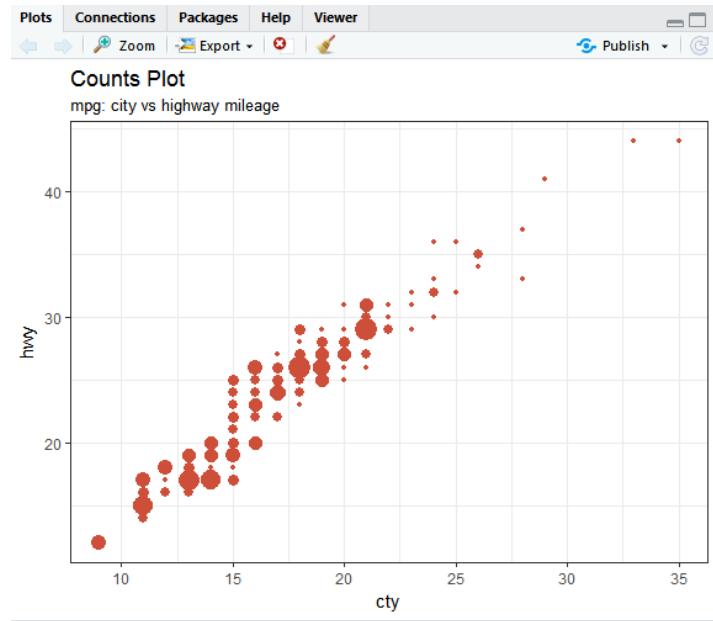
4. File/Plots/Packages/Help/Viewer

- File: Windows 파일 탐색기와 유사한 기능 제공
 - 파일 및 폴더 생성, 삭제/파일 및 폴더명 수정, 그리고 작업경로 설정



- **Plots:** 생성한 그래프 출력

- 작업 중 생성한 그래프 이력이 Plots 창에 저장: ← 이전, → 최근
- **Zoom:** 클릭 시 해당 그래프의 팝업창이 생성되고 팝업창의 크기 조정을 통해 그래프의 축소/확대 가능
- **Export:** 선택한 그래프를 이미지 파일(.png, .jpeg, .pdf 등)로 저장할 수 있고, 클립보드로 복사 가능

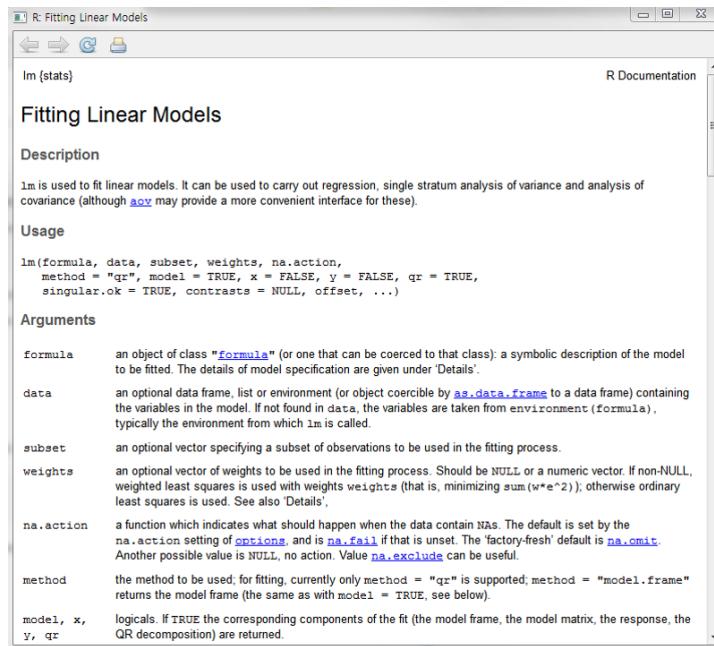


- **Packages:** 현재 컴퓨터에 설치된 R 패키지 목록 출력
 - 신규 설치 및 업데이트 가능

| Packages | | | |
|----------------|--|---------|---|
| System Library | | | |
| Name | Description | Version | |
| A3 | Accurate, Adaptable, and Accessible Error Metrics for Predictive Models | 1.0.0 | x |
| abbyyR | Access to Abbyy Optical Character Recognition (OCR) API | 0.5.1 | x |
| abc | Tools for Approximate Bayesian Computation (ABC) | 2.1 | x |
| abc.data | Data Only: Tools for Approximate Bayesian Computation (ABC) | 1.0 | x |
| ABC.RAP | Array Based CpG Region Analysis Pipeline | 0.9.0 | x |
| ABCAnalysis | Computed ABC Analysis | 1.2.1 | x |
| abcdefBA | ABCDE_FBA: A-Biologist-Can-Do-Everything of Flux Balance Analysis with this package. | 0.4 | x |
| ABCOptim | Implementation of Artificial Bee Colony (ABC) Optimization | 0.15.0 | x |
| ABCp2 | Approximate Bayesian Computational Model for Estimating P2 | 1.2 | x |
| abcrf | Approximate Bayesian Computation via Random Forests | 1.7 | x |
| abctools | Tools for ABC Analyses | 1.1.1 | x |
| abiol | The Analysis of Biological Data | 0.2.0 | x |

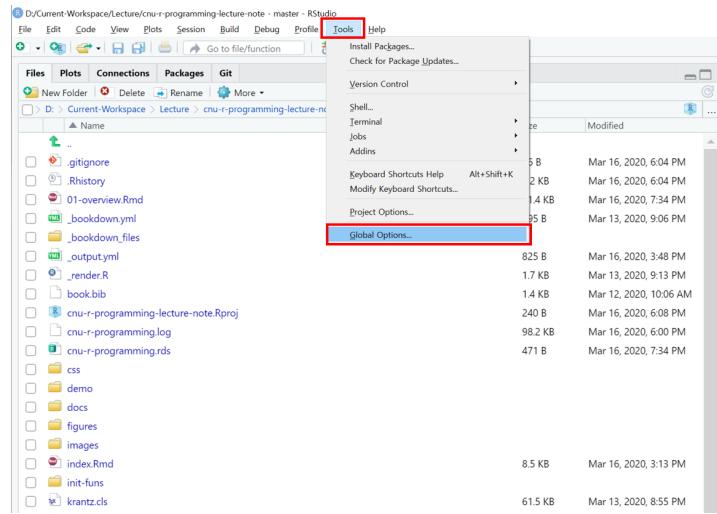
- **Help:** `help(topic)` 입력 시 도움말 창이 출력되는 공간

```
help(lm)
```



1.4.3 RStudio 환경 설정

Pull-down 메뉴에서 [Tools] → [Global Options...]를 선택



General: RStudio 운용 관련 전반적 설정 세팅

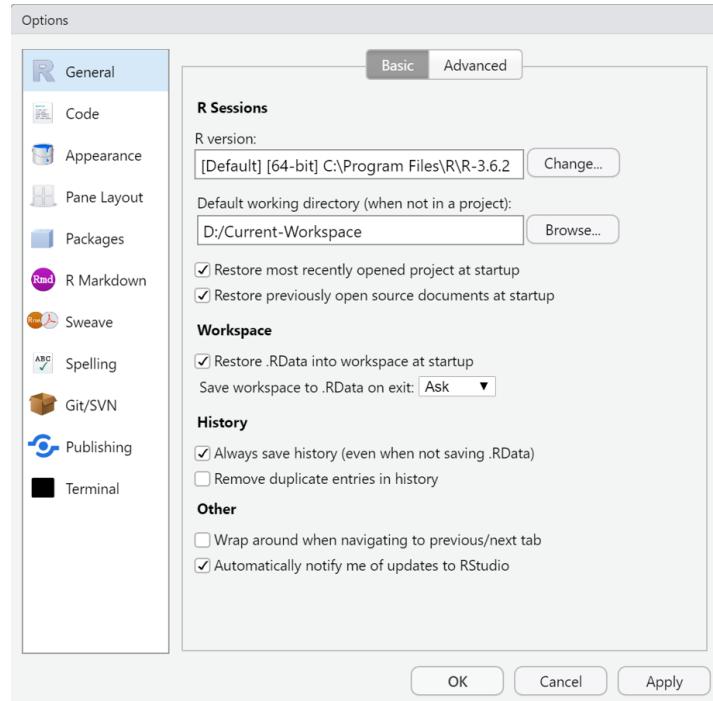


FIGURE 1.8: R General option 팝업 창

- **R version:** 만약 컴퓨터에 두 개 이상 다른 R 버전이 설치되어 있는 경우 [Change] 클릭 후 설정 변경 가능
- **Default Working directory:** 작업 디렉토리 지정([Browse] 클릭 후 임의 폴더 설정 가능)
- **Restore most recently opened project at startup:** RStudio 실행 시 가장 최근에 작업한 프로젝트로 이동
- **Restore previously open source documents at startup:** RStudio 실행 시 현재 프로젝트에서 가장 최근에 작업한 소스코드 문서를 함께 열어줌.
- **Restore .RData into workspace at startup:** 작업 디렉토리에 존재하는 .RData 파일을 RStudio 실행 시 불러옴
- **Save workspace to .RData on exit:** R workspace 자동 저장 (.RData) 여부
- **Always save history (even when not saving .RData) :** R 실행 명령 history 저장 여부(Always/Never/Ask)
- **Remove duplicate entries in history:** history 저장 시 중복 명령 제거 여부

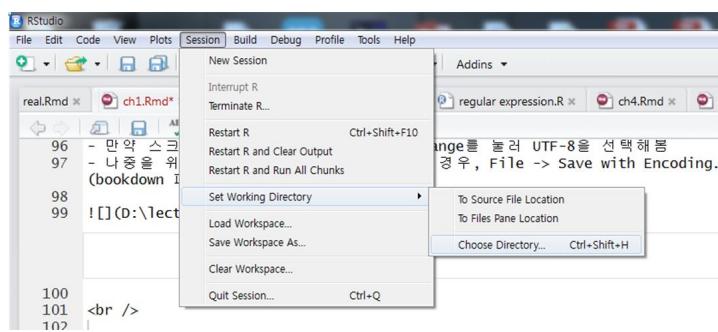
작업폴더(Working Directory)는 현재 R session에서 사용하는 기본 폴더로서 R 소스파일 및 데이터의 저장 및 로딩시 기본이 되는 폴더임.

- 소스파일이나 데이터를 불러들일 때 작업 폴더에 있는 파일은 경로명을 지정하지 않고 파일명만 사용해도 됨
- 작업폴더가 아닌 곳에 있는 파일을 불러들일 때는 경로명까지 써 주어야함.
- R 데이터를 저장할때도 파일명만 쓰면 기본적으로 작업폴더에 저장되며, 다른 폴더에 저장하기 위해서는 경로명까지 써 주어야 함.

처음 컴퓨터에 RStudio를 설치하면 Working directory는 Windows 사용자 폴더(예: user)의 Document 폴더가 기본값으로 설정되어 있음. 기본 작업폴더를 변경하려면 Figure 1.8에서 설정 가능.

현재 R session의 작업 디렉토리 설정 방법

- [Session] -> [Set Working Directoy] -> [Choose Directory]에서 설정



R 콘솔에서 다음과 같은 명령어로 작업폴더를 확인 및 변경 가능

```
getwd() # 작업폴더 확인 (현재 R 작업폴더)
```

```
[1] "/home/user/R-project/Lecture-note/cnu-r-programming-lecture-note"
```

```
# 참고: UNIX 환경에서 작성
# 상대경로
setwd("../") # 상위 폴더로 이동(getwd() 폴더 기준)
getwd()
```

```
[1] "/home/user/R-project/Lecture-note"
```

```
# 상대경로  
setwd("../..") # 차상위 폴더로 이동(getwd() 폴더 기준)  
getwd()
```

```
[1] "/home/user"
```

```
setwd("/home/user/R-project/Lecture-note/cnu-r-programming-lecture-note/") # 절대 폴더 명 입력  
getwd()
```

```
[1] "/home/user/R-project/Lecture-note/cnu-r-programming-lecture-note"
```

```
# 폴더 내 파일 명 출력  
dir()
```

```
[1] "_bookdown_files"  
[2] "_bookdown.yml"  
[3] "_output.yml"  
[4] "_render.R"  
[5] "01-introduction_files"  
[6] "01-introduction.Rmd"  
[7] "02-data-type_files"  
[8] "02-data-type.Rmd"  
[9] "03-string-regexp.Rmd"  
[10] "04-math-distribution-functions_files"  
[11] "04-math-distribution-functions.Rmd"  
[12] "05-rmarkdown-more_files"  
[13] "05-rmarkdown-more.Rmd"  
[14] "06-file-import-export.Rmd"  
[15] "07-control-flow.Rmd"
```

```
[16] "11-references.Rmd"  
[17] "2020"  
[18] "assignment"  
[19] "book.bib"  
[20] "cnu-r-programming-lecture-note.Rproj"  
[21] "cnu-r-programming.rds"  
[22] "code"  
[23] "css"  
[24] "data"  
[25] "dataset"  
[26] "dataset.zip"  
[27] "demo"  
[28] "docs"  
[29] "examples"  
[30] "figures"  
[31] "images"  
[32] "index.md"  
[33] "index.Rmd"  
[34] "index.utf8.md"  
[35] "init-funs"  
[36] "krantz.cls"  
[37] "latex"  
[38] "output"  
[39] "packages.bib"  
[40] "README.md"  
[41] "render1228e3f191848.rds"  
[42] "test"
```

```
# 상대경로  
dir("..")
```

```
[1] "cnu-r-programming-lecture-note" "quiz-repository"
```

```
[3] "r-programming-2020-01"      "stat-package-lecture"
[5] "talk-gallery"              "test-learnr"
[7] "ust-medical-statistics"
```

```
# 상대경로
setwd("../ust-medical-statistics/") # Lecture-note 파일 폴더로 stat 으로 이동
getwd(); dir()
```

```
[1] "/home/user/R-project/Lecture-note/ust-medical-statistics"
```

```
[1] "_workflowr.yml"           "analysis"
[3] "code"                     "data"
[5] "docs"                     "output"
[7] "README.md"                "render-slide.R"
[9] "slides"                   "ust-medical-statistics.Rproj"
```

```
# 절대경로
setwd("/home/user/R-project/Lecture-note/cnu-r-programming-lecture-note/")
getwd(); dir()
```

```
[1] "/home/user/R-project/Lecture-note/cnu-r-programming-lecture-note"
```

```
[1] "_bookdown_files"
[2] "_bookdown.yml"
[3] "_output.yml"
[4] "_render.R"
[5] "01-introduction_files"
[6] "01-introduction.Rmd"
[7] "02-data-type_files"
[8] "02-data-type.Rmd"
```

```
[9] "03-string-regexp.Rmd"  
[10] "04-math-distribution-functions_files"  
[11] "04-math-distribution-functions.Rmd"  
[12] "05-rmarkdown-more_files"  
[13] "05-rmarkdown-more.Rmd"  
[14] "06-file-import-export.Rmd"  
[15] "07-control-flow.Rmd"  
[16] "11-references.Rmd"  
[17] "2020"  
[18] "assignment"  
[19] "book.bib"  
[20] "cnu-r-programming-lecture-note.Rproj"  
[21] "cnu-r-programming.rds"  
[22] "code"  
[23] "css"  
[24] "data"  
[25] "dataset"  
[26] "dataset.zip"  
[27] "demo"  
[28] "docs"  
[29] "examples"  
[30] "figures"  
[31] "images"  
[32] "index.md"  
[33] "index.Rmd"  
[34] "index.utf8.md"  
[35] "init-funs"  
[36] "krantz.cls"  
[37] "latex"  
[38] "output"  
[39] "packages.bib"  
[40] "README.md"  
[41] "render1228e3f191848.rds"
```

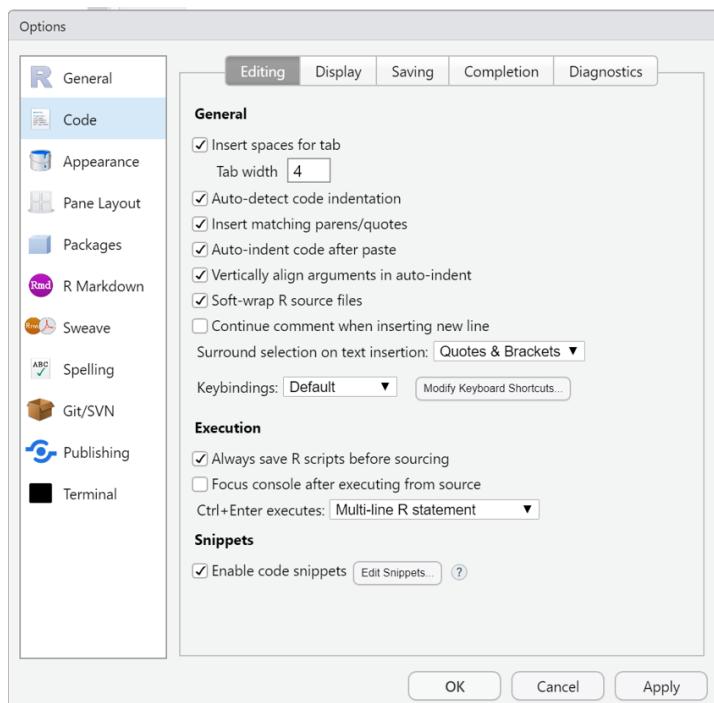
[42] "test"



R에서 디렉토리 또는 폴더 구분자는 / 입. Windows에서 사용하는 구분자는 \인데, R에서 \는 특수문자로 간주하기 때문에 Windows 의 폴더명을 그대로 사용 시 에러 메세지를 출력함. 이를 해결하기 위해 Windows 경로명을 그대로 복사한 경우 경로 구분자 \ 대신 \\로 변경

실습: R에서 폴더 경로 자유롭게 이동해 보기

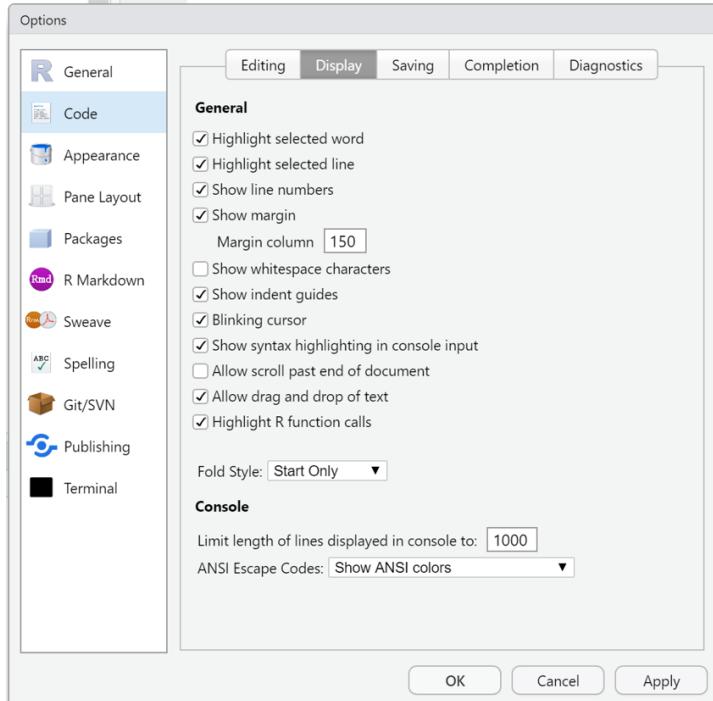
Code: Editing: 들여쓰기, 자동 줄바꿈 등 코드 편집에 대한 전반적 설정



- **Insert spaces for tab:** [Tab] 키를 눌렀을 때 공백(space) 개수 결정
(본 강의노트: Tab width = 4)
- **Auto-detect code indentation:** 코드 들여쓰기 자동 감지

- **Insert matching parens/quotes:** 따옴표, 괄호 입력 시 커서를 따옴표/괄호 사이로 자동 이동
- **Auto-indent code after paste:** 코드 복사 시 들여쓰기 일괄 적용
- **Vertically align arguments in auto-indent:** 함수 작성 시 들여쓰기 레벨 유지 여부
- **Soft-wrap R source file:** 스크립트 편집기 너비를 초과하는 경우 R 코드 행을 자동 줄바꿈
- **Continue comment when inserting new line:** 주석 표시를 다음 행에도 자동 적용 여부
- **Surround selection on text insertino:** 스크립트 상 text 선택 후 자동 따옴표 및 괄호 적용 여부
- **Focus console after executing from source:** 스크립트 실행 후 커서 위치를 콘솔로 이동 여부

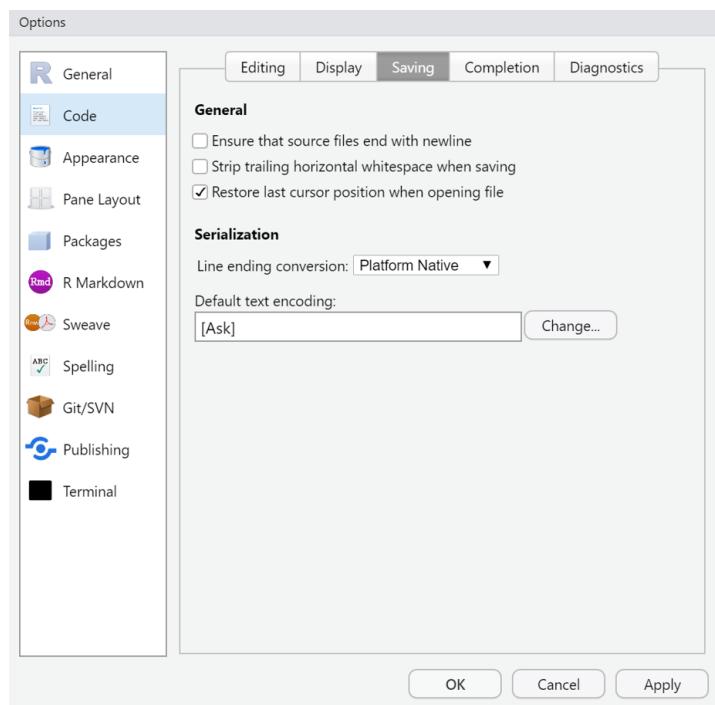
Code: Display: 스크립트(소스) 에디터 표시 화면 설정



- **Highlight selected word:** 스크립트 내 text 선택 시 동일한 text에 대해 배경강조 효과 여부
- **Highlight selected line:** 선택된 행에 대해 배경 강조효과 여부
- **Show line numbers:** 행 번호 보여주기 여부
- **Show margin:** 소스 에디터 오른 쪽에 지정한 margin column 보여 주기 여부
- **Show whitespace characters:** 에디터에 공백 표시 여부
- **Show indent guides:** 현재 들여쓰기 열 표시 여부
- **Blinking cursor:** 커서 깜박임 여부
- **Show syntax highlighting in console output:** 콘솔 입력 라인에 R 구문 강조 표시 적용 여부
- **Allow scroll past end of document:** 문서 마지막 행 이후 스크롤 허용 여부

- **Allow drag and drop of text:** 선택한 복수의 행으로 구성된 text에 대해 마우스 drag 허용
- **Highlight R function calls:** R 내장 및 패키지 제공함수에 대해 강조 여부

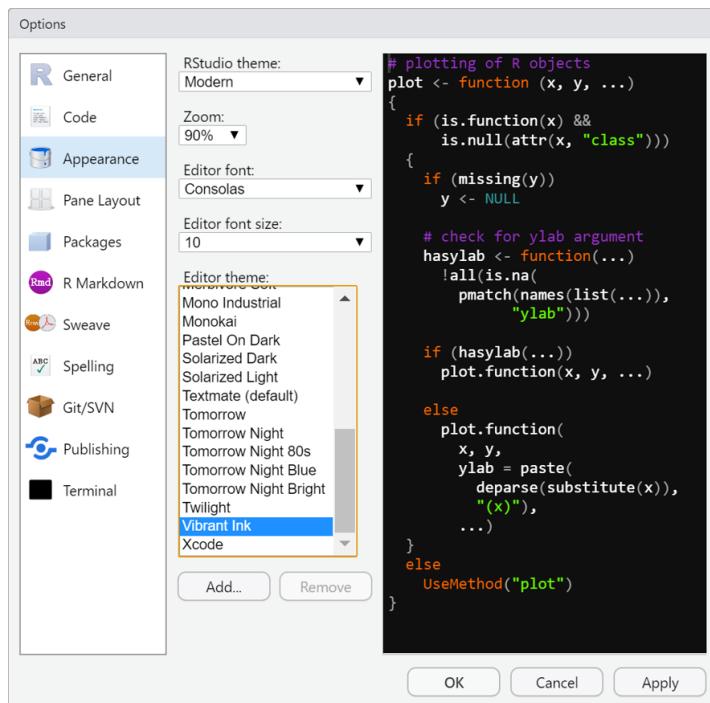
Code: Saving: 스크립트(소스) 에디터 저장 설정



- **Ensure that source file end with newline**
- **String trailing horizontal whitespace when saving**
- **Restore last cursor position when opening file**
- **Default text encoding:** 소스 에디터의 기본 설정 인코딩 설정 변경
 - RStudio의 Windows 버전 기본 text encoding은 CP949 입
 - Linux나 Mac OS의 경우 한글은 UTF-8로 인코딩이 설정되어 있음.

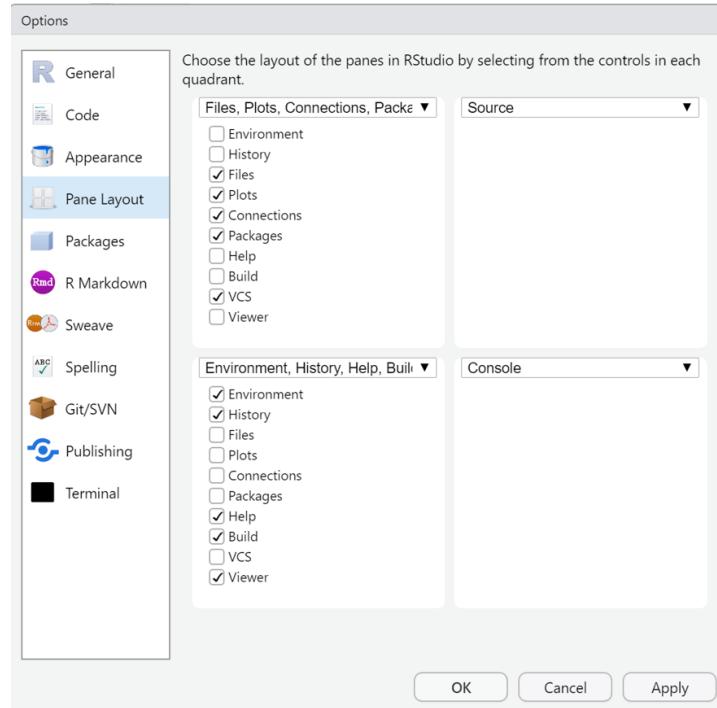
- R 언어는 Linux 환경에서 개발되었기 때문에 UTF-8 인코딩과 호환성이 더 좋음
- 스크립트 파일의 한글이 깨질 때는 [File] -> [Reopen with Encoding...]에서 encoding 방식 변경

Appearance: RStudio 전체 폰트, 폰트 크기, theme 설정



- 본인의 취향에 맞게 폰트 및 테마(theme) 설정
- 취향 → 가독성이 제일 좋고 편안한 theme

Pane Layout: RStudio 구성 패널들의 위치 및 항목 등을 수정/추가/삭제(4개 패널은 항상 유지)



실습: 개인 취향에 맞게 RStudio 에디터 및 theme을 변경해 보자!!



1.4.4 RStudio 프로젝트

1. 프로젝트

- 물리적 측면: 최종 산출물(문서)를 생성하기 위한 데이터, 사진, 그림 등을 모아 놓은 폴더
- 논리적 측면: R session 및 작업의 버전 관리

2. 프로젝트의 필요성

- 자료의 정합성 보장

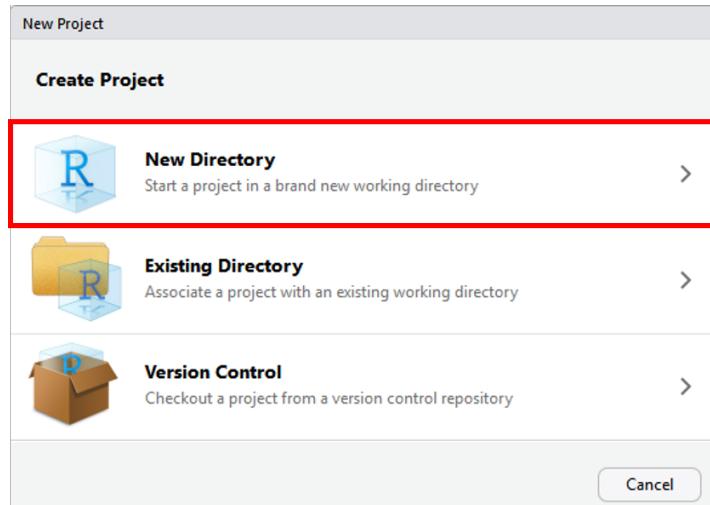
- 다양한 확장자를 갖는 파일들이 한 폴더 내에 뒤섞일 때 곤란해 질 수 있음
- 실제 분석 및 그래프 생성에 사용한 정확한 프로그램 또는 코드 연결이 어려움

3. 좋은 프로젝트 구성을 위한 방법

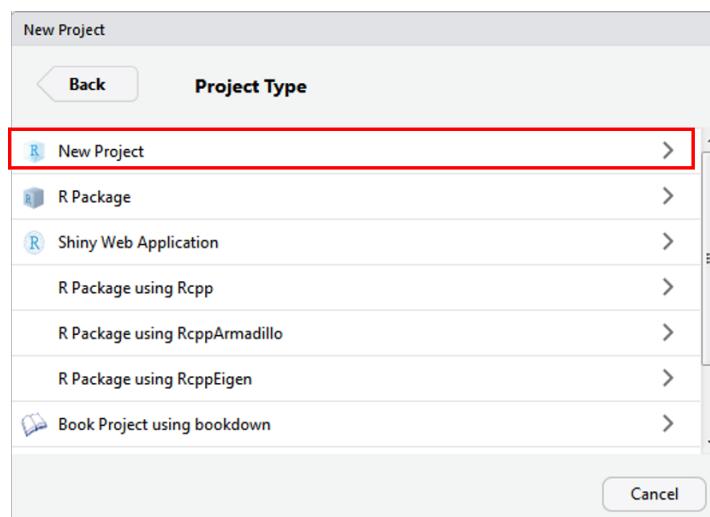
- 원자료(raw data)의 보호: 가급적 자료를 읽기 전용(read only) 형태로 다루기
- 데이터 정제(data wrangling 또는 data munging)를 위한 스크립트와 정제 자료를 보관하는 읽기 전용 데이터 디렉토리 생성
- 작성한 스크립트로 생성한 모든 산출물(테이블, 그래프 등)을 “일회용품”처럼 처리 → 스크립트로 재현 가능
- 한 프로젝트 내 각기 다른 분석마다 다른 하위 디렉토리에 출력 결과 저장하는 것이 유용

4. RStudio 새로운 프로젝트 생성

- RStudio의 강력하고 유용한 기능
- 새로운 프로젝트 생성: RStudio 메뉴에서 [File] → [New Project] 선택하면 아래와 같은 팝업 메뉴 생성

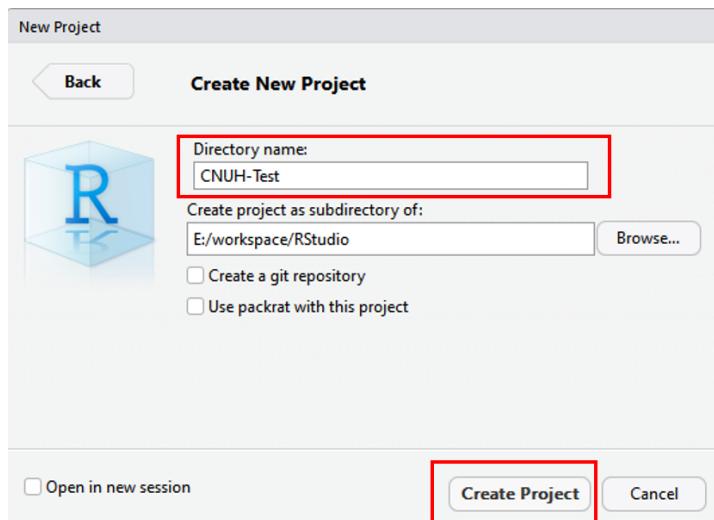


4. 위 그림에서 New Directory를 선택하면 아래와 같은 팝업 창이 나타나면 아래와 같은 프로젝트 유형이 나타남. 여기서는 New Project 선택



5. 다음 팝업창에서 새로운 프로젝트의 폴더명을 지정 후 Create Project 클릭

- 아래 [Create projects as subdirectories of]에서 생성하고자 하는 프로젝트의 상위 디렉토리 설정 → 보통 RStudio의 기본 작업폴더로 설정



- 현재 R session 종료 후 새로운 프로젝트로 session 화면이 열리면 프로젝트 생성 완료



실습: 프로젝트 생성

- 위에서 설정한 작업폴더 내에 학번-r-programming 프로젝트 생성
- 생성한 프로젝트 폴더 내에 docs, figures, script 폴더 생성



RStudio Cloud¹⁶ 사용

- R의 구동 환경은 Windows 보다는 Linux 운영 환경에 최적화됨
- 온라인에서 리눅스 환경의 R Studio 사용 가능

1.5 R 패키지



R 패키지(package): 특수 목적을 위한 로직으로 구성된 코드들의 집합으로 R에서 구동되는 분석툴을 통칭

- CRAN을 통해 배포: 3자가 이용하기 쉬움 → R 시스템 환경에서 패키지는 가장 중요한 역할
- CRAN available package by name¹⁷ 또는 available package by date¹⁸에서 현재 등재된 패키지 리스트 확인 가능
- R console에서 `available.packages()` 함수를 통해서도 확인 가능
- 현재 CRAN 기준(2020-03-17) 배포된 패키지의 개수는 16045 개임

목적: RStudio 환경에서 패키지를 설치하고 불러오기

1.5.1 R 패키지 경로 확인 및 변경

- 패키지 설치 시 일반적으로 R 환경에서 기본값으로 지정한 라이브러리 폴더에 저장
- 패키지 설치 전 R 패키지 설치 경로(path) 지정
- `.libPaths()` 함수를 통해 현재 설정된 패키지 저장 경로 확인

```
.libPaths()
```

```
[1] "/home/user/R/x86_64-pc-linux-gnu-library/4.0"
[2] "/usr/local/lib/R/site-library"
[3] "/usr/lib/R/site-library"
[4] "/usr/lib/R/library"
```

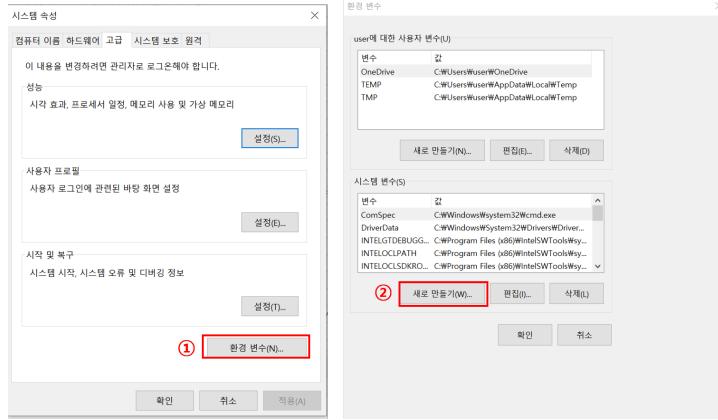
- 일반적으로 첫 번째 경로를 디폴트 라이브러리 폴더로 사용
- 사용자 지정 라이브러리 경로를 설정 하려면 아래와 같은 절차로 진행

실습: c:/r-library 폴더를 패키지 경로로 지정

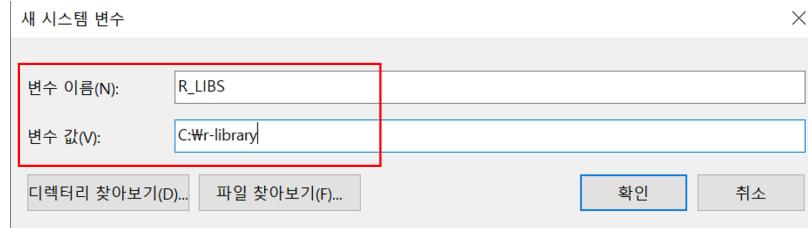
- 1) C:\에서 [새로 만들기(W)] -> [폴더(F)] 선택 후 생성 폴더 이름을 r-library로 변경
- 2) 윈도우즈 [제어판] -> [시스템 및 보안] -> [시스템] -> [고급 시스템 설정] 클릭



- 3) [환경변수(N)...] 선택 후 시스템 변수에서 [새로 만들기(W)...] 클릭



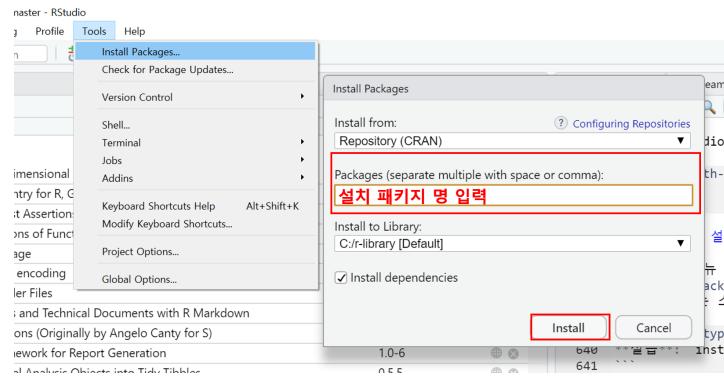
- 4) 아래 그림과 같이 변수 이름(N)에 R_LIBS, 변수 값(V)에 해당 디렉토리 경로 C:\r-library 입력 후 확인 버튼 클릭



- 5) 현재 RStudio 종료 후 재실행한 다음 콘솔창에 .libPaths() 입력 후 라이브러리 경로 확인

1.5.2 R 패키지 설치하기

- RStudio 메뉴 [Tools] → [Install packages] 클릭 후 생성된 팝업 창에서 설치하고자 하는 패키지 입력 후 설치



- RStudio Packages 창에서 [Install] 버튼 누르면 위와 동일한 팝업 창이 나타남(위와 동일)

| Name | Description |
|--|---|
| abind | Combine Multidimensional Arrays |
| askpass | Safe Password Entry for R, Git, and SSH |
| assertthat | Easy Pre and Post Assertions |
| backports | Reimplementations of Functions Introduced Since R-3.0.0 |
| <input checked="" type="checkbox"/> base | The R Base Package |
| base64enc | Tools for base64 encoding |
| BH | Boost C++ Header Files |
| bookdown | Authoring Books and Technical Documents with R Markdown |
| boot | Bootstrap Functions (Originally by Angelo Canty for S) |
| brew | Templating Framework for Report Generation |
| broom | Convert Statistical Analysis Objects into Tidy Tibbles |
| callr | Call R from R |

- R 콘솔 또는 스크립트 창에서 `install.packages(package_name)` 함수를 사용해서 패키지 설치



실습: `install.packages()` 함수를 이용해 tidyverse 패키지 설치

```
install.packages("tidyverse")
```

위 명령어를 실행하면 tidyverse 패키지 뿐 아니라 연관된 패키지들이
동시에 설치됨

1.5.3 R 패키지 불러오기

1. library() vs. require()

- `library()`: 불러오고자 하는 패키지가 시스템에 존재하지 않는 경우 에러 메세지 출력(에러 이후 명령어들이 실행되지 않음)
- `require()`: 패키지가 시스템에 존재하지 않는 경우 경고 메세지 출력(경고 이후 명령어 정상적으로 실행)

2. 다중 패키지 동시에 불러오기

- RStudio Packages 창에서 설치하고자 하는 패키지 선택 버튼 클릭하면 R workspace로 해당 패키지 로드 가능
- 스크립트 이용

실습: tidyverse 패키지 불러오기

```
require(tidyverse)
```



실무에서 R의 활용능력은 패키지 활용 여부에 달려 있음. 즉, 목적이 맞는 업무를 수행하기 위해 가장 적합한 패키지를 찾고 활용하느냐에 따라 R 활용능력의 차이를 보임. 앞서 언급한 바와 같이 CRAN에 등록된 패키지는 16000 개가 넘지만, 이 중 많이 활용되고 있는 패키지의 수는 약 200 ~ 300 개 내외이고, 실제 데이터 분석 시 10 ~ 20개 정도의 패키지가 사용됨. 앞 예제에서 설치하고 불러온 `tidyverse` 패키지는 Hadley Wickham ([Wickham et al., 2019](#))이 개발한 데이터 전처리 및 시각화 패키지 번들임. 해당 패키지에 대한 자세한 내용은 2학기 “통계패키지활용” 수업에 다룰 예정임

1.6 R 기초 문법



본 절에서 다루는 R 문법은 R 입문 시 객체(object)의 명명 규칙과 R 콘솔 창에서 가장 빈번하게 사용되는 기초적인 명령어만 다룰 예정임. 자세한 내용은 2-3주 차에 다룰 예정.

- R은 객체지향언어(object-oriented language)
 - 객체(object): 숫자, 데이터셋, 단어, 테이블, 분석결과를 저장하고 있는 R 내부의 모든 변수를 통칭함
 - “객체지향”的 의미는 R의 모든 명령어는 객체를 대상으로 이루어진다는 것을 의미



알아두면 유용한(콘솔창에서 매우 많이 사용되는) 명령어 및 단축키

- `ls()`: 현재 R 작업공간에 저장된 모든 객체 리스트 출력
- `rm(object_name)`: `object_name`에 해당하는 객체 삭제

- `rm(list = ls())`: R 작업공간에 저장된 모든 객체들을 일괄 삭제
- 단축키 [Ctrl] + [L]: R 콘솔 창 일괄 청소
- 단축키 [Ctrl] + [Shift] + [F10]: R session 초기화

예시

```
print("Hello R World!!")
```

```
[1] "Hello R World!!"
```

```
x <- 7  
y <- 1:30 #10에서 30까지 정수 입력  
ls() #현재 작업공간 내 객체명 출력
```

```
[1] "도움말 보기 명령어" "사용법"           "설명"  
[4] "a"                      "b"                  "cars"  
[7] "def.chunk.hook"        "fig_cap"            "hook_output"  
[10] "tab"                   "x"                  "y"
```

```
rm(x) # 객체 x 삭제  
ls()
```

```
[1] "도움말 보기 명령어" "사용법"           "설명"  
[4] "a"                      "b"                  "cars"  
[7] "def.chunk.hook"        "fig_cap"            "hook_output"  
[10] "tab"                   "y"
```

```
rm(a,b) # 객체 a,b 동시에 삭제
ls()
```

```
[1] "도움말 보기 명령어" "사용법"           "설명"
[4] "cars"                      "def.chunk.hook"      "fig_cap"
[7] "hook_output"                "tab"                  "y"
```

```
# rm(list = ls()) # 모든 객체 삭제
```

R 객체 입력 방법 및 변수 설정 규칙

객체를 할당하는 두 가지 방법:=, <-

- 두 할당 지시자의 차이점
 - :=: 명령의 최상 수준에서만 사용 가능
 - <=: 어디서든 사용 가능
 - 함수 호출과 동시에 변수에 값을 할당할 목적으로는 <-만 사용 가능

```
# mean(): 입력 벡터의 평균 계산
mean(y <- 1:5)
```

```
[1] 3
```

```
y
```

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

```
mean(x = 1:5)
```

```
[1] 3
```

```
x
```

```
Error in eval(expr, envir, enclos): 객체 'x'를 찾을 수 없습니다
```

객체 또는 변수의 명명 규칙

- 알파벳, 한글, 숫자, _, .의 조합으로 구성 가능(-은 사용 불가)
- 변수명의 알파벳, 한글, .로 시작 가능
- .로 시작한 경우 뒤에 숫자 올 수 없음(숫자로 인지)
- 대소문자 구분

```
# 1:10은 1부터 10까지 정수 생성
# 'c()'는 벡터 생성 함수
x <- c(1:10)
# 1:10으로 구성된 행렬 생성
X <- matrix(c(1:10), nrow = 2, ncol = 5, byrow = T)
x
```

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

```
x
```

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

```
# 논리형 객체
.x <- TRUE
.x
```

```
[1] TRUE
```

```
# 알파벳 + 숫자
# seq(): 수열을 만드는 함수
# 1 에서부터(from) 10 까지(to) 공차가 2(by)인 수열
a1 <- seq(from = 1, to = 10, by = 2)
# 한글 변수명
가수 <- c("Damian Rice", "Beatles", "최백호", "Queen", "Carlos Gardel", "BTS", "조용필")
가수
```

```
[1] "Damian Rice" "Beatles" "최백호"
"Queen"
[5] "Carlos Gardel" "BTS" "조용필"
```

3. 잘못된 객체 또는 변수 명명 예시

```
3x <- 7
```

Error: <text>:1:2: 예상하지 못한 기호(symbol)입니다.

1: 3x

^

```
_x <- c("M", "M", "F")
```

Error: <text>:1:1: 예상하지 못한 입력입니다.

1: _

^

```
.3 <- 10
```

Error in 0.3 <- 10: 대입에 유효하지 않은 (do_set) 좌변입니다

1.7 R Markdown (맛보기)



R **기초 문법** 절과 마찬가지로 R Markdown을 이용해 최소한의 문서(html 문서)를 작성하고 생성하는 방법에 대해 기술함. R Markdown에 대한 보다 상세한 내용은 9 주차에 다룰 예정임.

1. R Markdown은 R 코드와 분석 결과(표, 그림 등)을 포함한 문서 또는 컨텐츠를 제작하는 도구로 일반적으로 아래 열거한 형태로 활용함

- 문서 또는 논문(pdf, html, docx)
- 프리젠테이션(pdf, html, pptx)
- 웹 또는 블로그

2. 재현가능(reproducible)한 분석 및 연구¹⁹ 가능

- 신뢰성 있는 문서 작성
- Copy & paste를 하지 않고 효율적 작업 가능

3. R Markdown 문서를 통해 최종 결과물(pdf, html, docx)이 도출되는 process

- 현재 공식적인 프로세스는 knitr + rmarkdown + pandoc + RStudio + github

R Markdown 문서 시작하기

- R Markdown 문서 생성: [File] -> [New File] -> [R Markdown..]을 선택



RStudio를 처음 설치하고 위와 같이 진행할 경우 아래와 같은 패키지 설치 여부를 묻는 팝업 창이 나타남. 패키지 설치 여부에 [Yes]를 클릭하면 R Markdown 문서 생성을 위해 필요한 패키지들이 자동으로 설치

¹⁹과학적 연구의 결과물을 오픈소스로 내놓고 누구라도 검증 가능

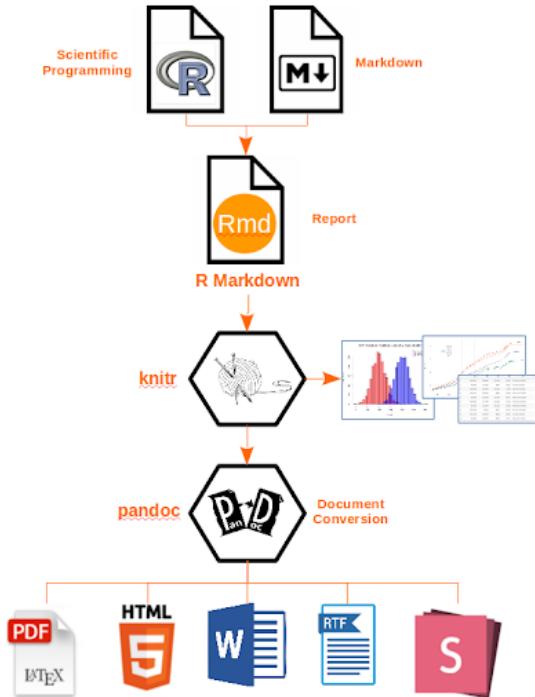
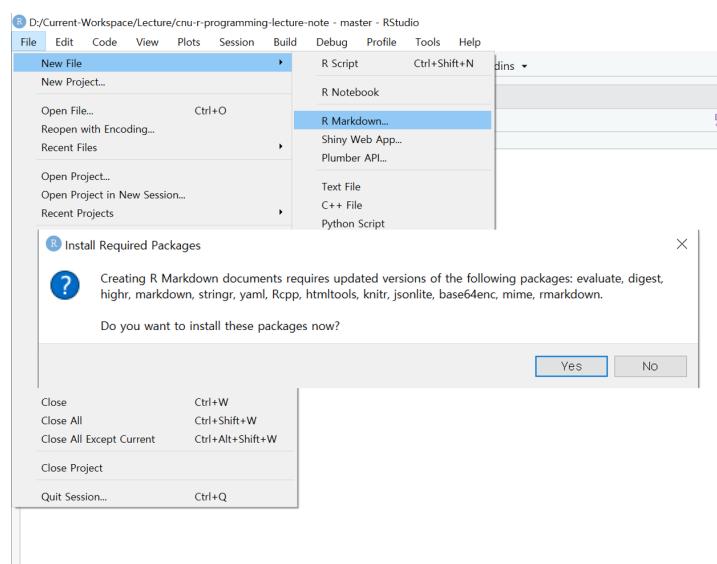
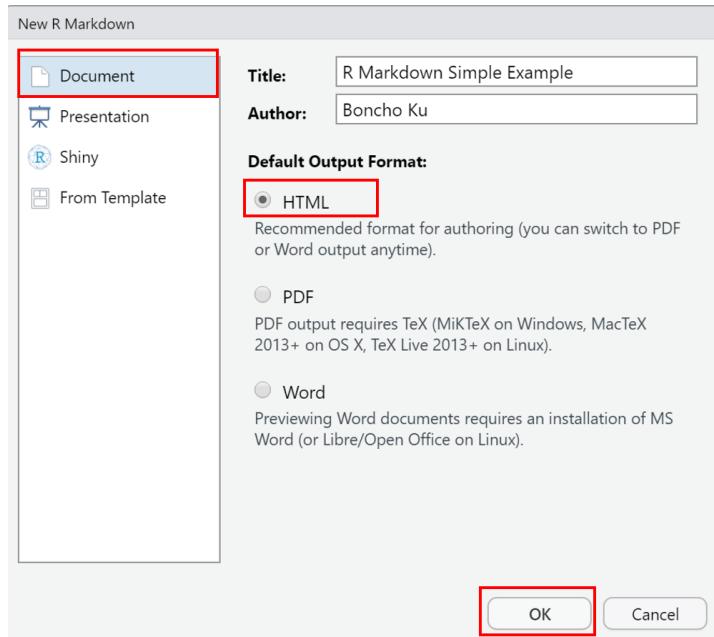


FIGURE 1.9: R Markdown의 최종 결과물 산출과정(<http://applied-r.com/project-reporting-template/>)



- 설치 완료 후 R Markdown으로 생성할 최종 문서 유형 선택 질의 창이 나타남. 아래 창에서 제목(Title)과 저자(Author) 이름 입력 후 [OK] 버튼 클릭(Document, html 문서 선택)



- 아래 그림과 같이 새로운 문서 창이 생성되고 test.Rmd 파일로 저장²⁰

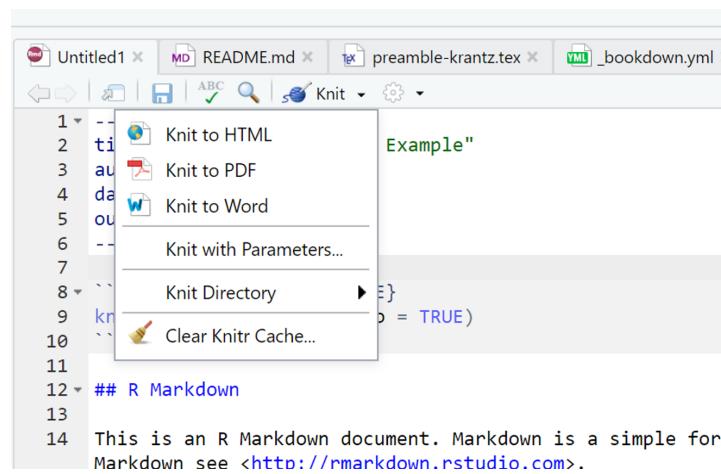
```

1<code>#</code>
2<code>title: "R Markdown Simple Example"</code>
3<code>author: "Boncho Ku"</code>
4<code>date: '2020-3-17'"</code>
5<code>output: html_document</code>
6<code>---

```

²⁰RStudio 프로젝트에서 생성한 폴더 내에 파일 저장

- 문서 상단에 Knit 아이콘을 클릭 후 Knit to HTML 클릭 또는 문서 아무 곳에 커서를 위치하고 단축키 [Ctrl] + [Shift] + [K] 입력



- knitr + R Markdown + pandoc → html 파일 생성 결과

R Markdown Simple Example

Boncho Ku
2020 3 17

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see <http://rmarkdown.rstudio.com>.

When you click the Knit button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

summary(cars)

```

##      speed      dist
##  Min.   :4.0   Min.   : 2.00
##  1st Qu.:12.0  1st Qu.:26.00
##  Median :15.0  Median :36.00
##  Mean   :15.4  Mean   :42.98
##  3rd Qu.:19.0  3rd Qu.:56.00
##  Max.   :25.0  Max.   :120.00

```

Including Plots

You can also embed plots, for example:

FIGURE 1.10: test.html 문서 화면(저장 폴더 내 ‘test.html’을 크롬 브라우저로 실행)

1.7.0.1 R Markdown 문서 구성

R Markdown 문서는 아래 그림과 같이 YAML, Markdown 텍스트, Code Chunk 세 부분으로 구성됨.

```

1 <!--
2   title: "R Markdown Simple Example"
3   author: "Boncho Ku"
4   date: '2020-3-17'
5   output: html_document
6   ---
7
8 ---{r setup, include=FALSE}
9 knitr::opts_chunk$set(echo = TRUE)
10 ...
11
12 ## R Markdown
13
14 This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R
15 Markdown see <http://rmarkdown.rstudio.com>.
16 When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks
17 within the document. You can embed an R code chunk like this:
18
19 {r cars}
20 summary(cars)
21 ...
22 ## Including Plots
23
24 You can also embed plots, for example:
25
26 ---{r pressure, echo=FALSE}
27 plot(pressure)
28
29 Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.
30
31

```

1. YAML (YAML Ain't Markup Language)

- R Markdown 문서의 metadata로 문서의 맨 처음에 항상 포함되어야 함.
- R Markdown 문서의 최종 출력 형태, 제목, 저자, 날짜 등의 정보 등을 포함
- YAML 언어에 대한 사용 예시는 Xie (2016) 의 Appendix B.2²¹ 참고
- 최소 형태의 YAML 예시

```

---
title: "Hello R Markdown"
author: "Zorba"
date: "2020-03-17"
output: html_document
---
```

²¹<https://bookdown.org/yihui/bookdown/r-markdown.html>

2. Markdown 텍스트

- Markdown 문법은 15주 차 강의에서 배울 예정임
- R Markdown 레퍼런스 가이드²² 참조
- 그림 삽입: ![] (path/filename)

그림 삽입 구문

![] (figures/son.jpg)

²²<https://rstudio.com/wp-content/uploads/2015/03/rmarkdown-reference.pdf>



3. Code Chunk

- 실제 R code가 실행되는 부분임

- Code chunk 실행 시 다양한 옵션들이 있으나 이 부분 역시 15주 차 강의에서 간략히 다룰 예정임
- Code chunk는 `r`로 시작되며 r은 code 언어 이름을 나타냄.
- Code chunk는 `r`로 종료
- R Markdown 문서 작성 시 단축키 [Ctrl] + [Alt] + [I]를 입력하면 Chunk 입력창이 자동 생성됨
- Chunk option에 대한 상세 내용은 <https://yihui.org/knitr/options/> 또는 R Markdown 레퍼런스 가이드²³ 참조

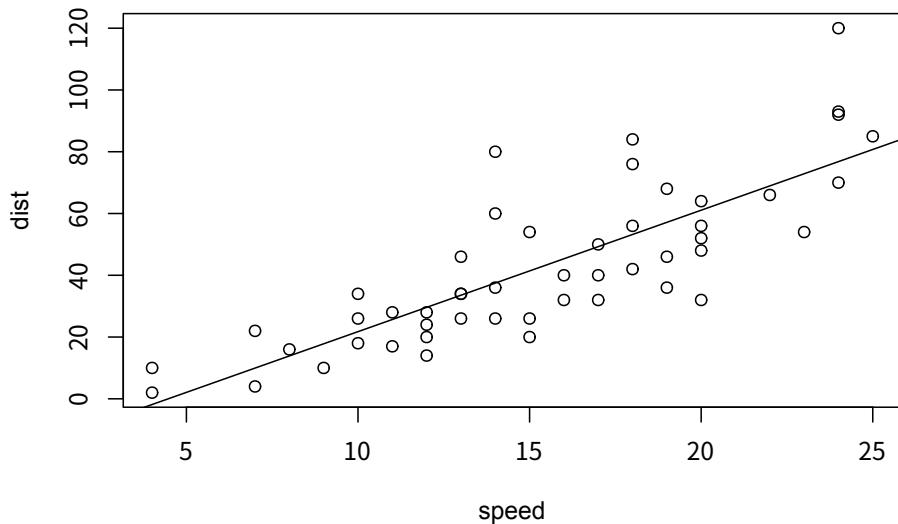
Code chunk 예시

Xie의 R Markdown: The Definitive Guide에서 발췌

```
```{r}
fit = lm(dist ~ speed, data = cars)
b = coef(fit)
plot(cars)
abline(fit)
````
```

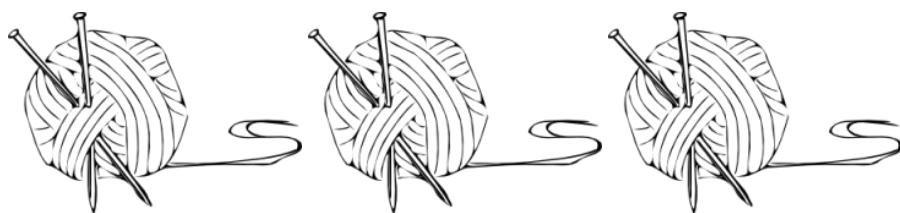
```
fit = lm(dist ~ speed, data = cars)
b   = coef(fit)
plot(cars)
abline(fit)
```

²³<https://rstudio.com/wp-content/uploads/2015/03/rmarkdown-reference.pdf>



- Code chunk에서 외부 그림 파일 불러오기([Xie et al. \(2018\)](#)에서 예시
발췌)

```
knitr::include_graphics(rep('figures/knit-logo.png', 3))
```



Homework 1: R Markdown 문서에 아래 내용을 포함한 문서를 `html` 파일 형식으로 출력 후 제출

- 간략한 자기소개 및 “통계 프로그래밍 언어” 수업에 대한 본인만의 목표 기술
- 본인이 setting 한 RStudio 구성 캡쳐 화면을 그림 파일로 저장하고 R Markdown 문서에 삽입(화면 캡쳐 시 생성 프로젝트 내 폴더 내용 반드시 포함)
- 현재 R 작업폴더(처음 R 시작 시 디폴트로 설정된 폴더) 및 작업폴더 내에 있는 파일명 출력

- 현재 R 작업폴더에서 차상위 폴더의 파일명 출력
- 패키지 `ggplot2` 패키지 설치 후 `cars` 데이터셋의 2차원 산점도(`hint: help(geom_point)` 또는 googling 활용)를 문서에 포함



2

R 객체(R object)



학습목표(2 주차): R에서 사용 가능한 데이터 타입에 대해 알아보고, 고유 데이터 타입으로 구성한 객체(스칼라, 벡터, 리스트)와 이와 연관된 함수들을 익힌다.

학습 필요성

- R언어는 타 프로그래밍 언어와 유사한 데이터 타입(정수형, 실수형, 문자형 등)을 제공
- R 언어가 다른 언어와 차이점 → **데이터 분석**에 특화된 벡터(vector), 행렬(matrix), 데이터프레임(data frame), 리스트(list)와 같은 객체¹ 제공
- R 패키지에서 제공되는 함수 사용 방법은 R의 객체에 따라 달라질 수 있음
- R 언어를 원활히 다룰 수 있으려면 R에서 데이터 객체의 형태, 자료 할당 및 그 연산 방법에 대한 이해가 필수적으로 선행되어야 함

¹R에서 사용자가 데이터 입력을 위해 생성 또는 읽어온 객체(object)는 종종 변수(variable)라는 말과 혼용. 본 문서에서는 최상위 데이터 저장장소를 객체라고 명명하며 데이터프레임과 같이 여러 종류의 데이터타입으로 이루어진 객체의 1차원 속성을 변수라고 칭함

R의 데이터 타입

- 수치형(numeric): 숫자(정수, 소수)
- 문자열(string): "충남대학교", "R강의"
- 논리형(logical): TRUE/FALSE
- 결측값(NA): 자료에서 발생한 결측 표현
- 공백(NULL): 지정하지 않은 값
- 요인(factor): 범주형 자료 표현(수치 + 문자 결합 형태로 이해하면 편함)
- 기타: 숫자아님(NaN), 무한대(Inf) 등

R 객체의 종류

- 스칼라(상수형, scalar 또는 atomic)
- 벡터(vector): R의 기본연산 단위
- 리스트(list)
- 행렬(matrix)
- 배열(array)
- 데이터프레임(data frame)

2.1 프로그래밍

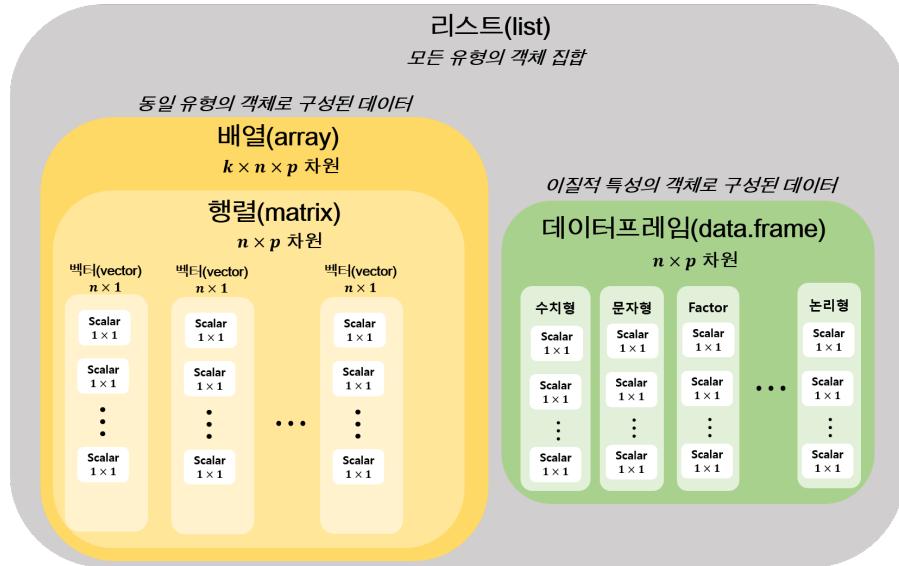


FIGURE 2.1: R 데이터 타입 구조 다이어그램: [R, Python 분석과 프로그래밍 (by R Friend)](<http://rfriend.tistory.com/>)에서 발췌 후 수정

2.1.1 Prerequisites

- 예약어(researved words): R에서 의미(semantic)를 미리 정해 놓은 단어

TABLE 2.1: R 예약어 종류 및 설명

| R 예약어 | 설명 |
|--|----------------------------|
| if, else, while, function, in, next, break | 조건, 함수, 반복문에 사용 |
| TRUE/FALSE | 논리 상수(logical constants) |
| NULL | 정의되지 않은 값 혹은 값이 없음 표현 |
| Inf | 무한(infinity) |
| NaN | 숫자가 아님(not a number) |
| NA | 결측값(not available) |
| NA_integer_, NA_real_, NA_complex_, NA_character_ | 결측값을 처리하는 상수 |
| ... | 함수가 다른 함수에 인자를 전달하도록 지원 |

- **변수(variable):** 사용자가 프로그램 처리를 위해 지정한 단어
 - 적당한 값을 저장하고 나중에 필요시 해당 값을 호출해 사용하기 위한 목적으로 사용되는 표식(label)
 - 예약어를 변수명으로 사용할 수 없음
 - 통계프로그래밍언어 강의노트: R 기초문법² 참고
- **고수준 언어(high-level language):** 사람이 읽고 쓰기 쉬운 형태의 명령어를 컴퓨터가 읽고 처리할 수 있도록 고안된 프로그래밍 언어

²<https://zorba78.github.io/cnu-r-programming-lecture-note/r-basic.html>

- 컴퓨터가 이해할 수 있는 언어 → 중앙처리장치(central processing unit, CPU)가 이해하는 언어 → 기계어(machine language)
 - 기계어는 0과 1로 구성된 이진수(binary number)임(예:
0100101001001001001110110101101010110)
 - 고수준 언어의 종류: C, C++, JAVA, 베이직, Perl, Python, R,
...
- **번역기(translator):** 사람이 이해할 수 있는 표현(언어)를 기계(컴퓨터)가 이해할 수 있는 언어(기계어)로 변환
 - 인터프리터(interpreter)
 - 컴파일러(compiler)
 - ****인터프리터*:** 코드(스크립트) 한 줄을 즉석에서 읽고, 파싱(프로그램을 검사하고 구문론적 구조를 분석)하고, 해석
 - R, Python, MATLAB 등은 인터프리터를 번역기로 사용
 - 인터랙티브 모드 → R 프롬프트(>) 뒤에 한 줄의 명령어를 작성하면 즉석해서 처리 후 다음 입력에 대해 준비(prompt)함.

안녕하세요!!

통계패키지활용 수업에서 R을 배우고 있습니다.

처음이라 실수가 많습니다.

앞으로 잘 부탁해요!!

Error: <text>:1:6: 예기치 않은 '!'입니다

1: 안녕하세요!

^

```
print("안녕하세요!!")
print("통계패키지 활용 수업을 위해 R을 배우고 있습니다.")
print("처음이라 실수가 많습니다.")
print("앞으로 잘 부탁해요!!")
```

```
[1] "안녕하세요!!"
[1] "통계패키지 활용 수업을 위해 R을 배우고 있습니다."
[1] "처음이라 실수가 많습니다."
[1] "앞으로 잘 부탁해요!!"
```

- **컴파일러:** 완전한 프로그램을 하나의 파일에 담고 파일 안에 저장되어 있는 소스코드를 기계어로 번역 후 다음 실행할 수 있도록 변환한 기계어를 파일에 담음.
 - 보통은 .exe, .dll 파일 형태로 저장됨

2.1.2 프로그램

- **프로그램(program):** 특정 작업(목적)을 수행할 수 있도록 작성한 일련의 R 문장(명령어)의 집합
 - 일련의 문장(명령어)들은 텍스트 편집기를 통해 작성하며, **스크립트(script)**로 명칭되는 파일로 저장 → R 스크립트 .R 확장자를 가짐

```
# Hello.R
print("안녕 R!!") #한국어
print("Hi R!!") # 영어
print("こんにちはR!!") # 일본어
print("Hello R!!") #그리스어
```

```
source("examples/hello.R", encoding = "UTF-8")
```

```
[1] "안녕 R!!"
[1] "Hi R!!"
[1] "こんにちはR!!"
[1] "Γεια R!!"
```

- 예시: 텍스트 파일에서 가장 자주 나오는 단어 찾기 프로그램

– <https://statklee.github.io/r4inf/r-intro.html#r-intro-what-is-a-program> 참고

```
require(tidyverse)
require(stringr)
require(ggpubr)
require(ggthemes)

text_dat <- readLines("data/text-example-01.txt")
# 공백 또는 구둣점 문자를 기준으로 텍스트 나누기

# 공백 또는 구둣점 문자 기준으로 텍스트 토큰화
split_wd <- str_split(text_dat, pattern = "\\b|[[:punct:]]")
split_wd <- do.call(c, split_wd)
id <- grep("[a-zA-Z]+", split_wd) # 알파벳을 포함한 단어 인덱스
split_wd <- split_wd[id]
unique_wd <- unique(split_wd) # 중복을 제외한 총 사용 단어
res_v <- vector("integer", length(unique_wd)) # 저장 벡터 생성

for (i in seq_along(unique_wd)) {
  for (j in seq_along(split_wd)) {
```

```

    if (unique_wd[i] == split_wd[j]) {
      res_v[i] <- res_v[i] + 1
    }
  }
}

bind_cols("word" = unique_wd, "freq" = res_v) %>%
  arrange(desc(freq))

```

```

# A tibble: 428 x 2
  word   freq
  <chr> <dbl>
1 the     57
2 in      24
3 of      23
4 to      20
5 South    17
6 a       15
7 and     14
8 Korea    13
9 was      9
10 which   9
# ... with 418 more rows

```

- 프로그램 작성을 위한 개념적 요소
 - **입력(input)**: 외부로부터 가져온 데이터, 값 등
 - **출력(output)**: 입력에 대한 반응(결과 출력, 파일 저장, 음악 재생, ...)
 - **순차실행(sequential execution)**: 스크립트 또는 코드 작성 순서에 따라 한줄씩 실행

- 조건실행(conditional execution): 특정 조건에 따라 문장(명령)을 실행하거나 건너뜀
 - 반복실행(iterative execution): 특정 명령을 반복적으로 실행
 - 재사용(resuse): 스크립트의 집합(다수 줄로 구성된 코드 또는 스크립트)에 이름을 부여하고 저장 → 사용자 지정 함수(function)
- 프로그램 오류의 종류
 - 구문오류(syntax error): R 언어가 이해할 수 없는 문장 또는 문법으로 실행했을 때 나타나는 오류 → 가장 고치기 쉽고 즉각적으로 알려줌
 - 논리 또는 run-time 오류(logic or run-time error): 구문은 완벽하지만 실행 순서 또는 논리적으로 연관방식에 문제가 있어서 명령어를 수행할 수 없는 경우
 - 의미론적 오류(semantic error): 프로그램은 구문적으로 오류가 없고 실행되지만 올바른 결과를 출력하지 않는 경우 → 제일 고치기 어려움
 - 가장 간단한 프로그래밍은 순차적으로 명령을 실행하되 입력 시 흐름을 잠시 중단하고 대기하는 방법 → 프롬프트 상 명령어 한 줄씩 입력

```
# 아주 간단한 프로그래밍 예제
# readline() 함수 이용해 R한테 인사 받기
name <- readline("What's your name?: ")
cat("Hello, ", name, "!\\n", sep = "")
```

```
# readline() 함수를 이용해 알바비 계산

x <- as.numeric(readline(prompt = "하루 아르바이트 시간을 입력하시오: "))
y <- as.numeric(readline(prompt = "시급을 입력하시오 (단위=원): "))
z <- as.numeric(readline(prompt = "한달 동안 총 몇 일 동안 일을 하셨나요? "))
cat("월 급여는 ", x * y * z, " 원 입니다.\n", sep = "")
```

2.2 스칼라(scalar)

- 단일 차원의 값(하나의 값): 1×1 벡터로 표현 \rightarrow R 데이터 객체의 기본은 벡터!!
- 데이터 객체의 유형은 크게 숫자형, 문자열, 논리형이 있음



스칼라를 입력시 R의 벡터 지정 함수인 `c()`(벡터 부분에서 상세 내용 학습)를 꼭 사용해서 입력할 필요가 없다. 단, 연속되지 않은 두 개 이상 스칼라면 벡터이므로 꼭 `c()`를 써야 한다.

2.2.1 선언

- 일반적으로 컴파일이 필요한 언어(예: C 언어)의 경우 변수 또는 객체를 사용 전에 선언이 필요

```
int x;
x = 1;
```

- 위 코드에서 `int x;` 없이 `x = 1`을 입력 후 컴파일 하면 에러가 나타나지만 R 언어에서는 변수를 선언할 필요가 전혀 없음
- `z` 가 어떤 데이터 타입인지 언급할 필요가 전혀 없음 → Python, Perl, Matlab 등과 같은 스크립트 언어의 특징. 아래 코드 참조

```
z <- 3  
z
```

```
[1] 3
```

2.2.2 숫자형

- 정수형(integer)과 실수형(double)로 구분됨
- 정수형 구분시 숫자 뒤 L을 표시

```
# 정수형 구분자 사용 예시  
# typeof(): R 객체의 데이터 타입 반환하는 함수  
typeof(10L)
```

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

```
typeof(10)
```

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

- 수치연산(+, -, *, ^, **, /, %%, %%) 가능: R은 함수형 언어이기 때문에 앞에 기술한 연산자도 하나의 함수로 인식함.

- 수치 연산자(operator) 및 기본 수학 함수

TABLE 2.2: R언어의 기본 수치 연산자

| 수치형 연산자 | 설명 |
|---|---|
| <code>+, -, *, /</code> | 사칙연산 |
| <code>n %% m</code> | <code>n</code> 을 <code>m</code> 으로 나눈 나머지 |
| <code>n %/% m</code> | <code>n</code> 을 <code>m</code> 으로 나눈 몫 |
| <code>n ^ m</code> 또는 <code>n ** m</code> | <code>n</code> 의 <code>m</code> 승 |

숫자형 스칼라 연산 적용 예시

```
# 숫자형 스칼라
a <- 3
b <- 10
a; b
```

[1] 3

[1] 10

```
# 덧셈
c <- a + b
c
```

[1] 13

2.2 스칼라(scalar)

83

```
# 덧셈을 함수로 입력  
# "+"(a, b)로 입력한 결과  
c <- "+"(a, b)
```

```
# 뺄셈  
d <- b - a  
d
```

[1] 7

```
# 곱셈  
m <- a * b  
m
```

[1] 30

```
# 나누기  
dd <- b/a  
dd
```

[1] 3.333333

```
# 몹승  
b^a
```

[1] 1000

```
# 나누기의 나머지(remainder) 반환
r <- b %% a
r
```

[1] 1

```
# 나누기의 몫(quotient) 반환
q <- b %/% a
q
```

[1] 3

```
# 연산 우선 순위
nn <- (3 + 5)*3 - 4**2/4
nn
```

[1] 20

2.2.3 문자형

- 수치형이 아닌 문자 형식의 단일 원소
- C와 같은 언어에서 볼수 있는 한개 문자에 대한 데이터 타입 존재하지 않음
- 수치연산 불가능
- 따옴표(" 또는 ')로 문자를 묶어서 문자열 표시
- 문자열을 다루는 자세한 설명은 5주차에서 자세히 설명할 예정임

```
h1 <- c("Hello CNU!!")  
h2 <- c("R is not too difficult.")  
typeof(h1); typeof(h2)
```

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

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

```
h1
```

```
[1] "Hello CNU!!"
```

```
h2
```

```
[1] "R is not too difficult."
```

```
# 문자열의 문자 수 반환  
nchar(h1); nchar(h2)
```

```
[1] 11
```

```
[1] 23
```

```
# 문자열 연산 error 예시  
h1 - h2
```

Error in h1 - h2: 이항연산자에 수치가 아닌 인수입니다

2.2.4 논리형 스칼라

- 참(TRUE, T) 또는 거짓(FALSE, F)를 나타내는 값
- TRUE/FALSE: 예약어(reserved word)
- T/F: TRUE와 FALSE로 초기화된 전역 변수
 - T에 FALSE 또는 어떤 값도 할당 가능 → 가급적 TRUE/FALSE를 명시하는 것이 편함
- 논리형 연산자(logical operator)

TABLE 2.3: R언어의 논리형 연산자

| 논리형 연산자 | 설명 |
|---------|------------------|
| & | AND (vectorized) |
| && | AND (atomic) |
| | OR (vectorized) |
| | OR (atomic) |
| ! | NOT |

- 비교 연산자를 적용할 경우 논리값을 반환

TABLE 2.4: R언어의 비교 연산자

| 비교 연산자 | 설명 |
|--------|----------------------------|
| > | 크다(greater-than) |
| < | 작다(less-than) |
| == | 같다(equal) |
| >= | 크거나 같다(greater than equal) |
| <= | 작거나 같다(less than equal) |
| != | 같지 않다(not equal) |

Note:

기술한 비교 연산자는 수치형 및 논리형 데이터 타입 모두에 적용 가능 하지만, 문자형은 비교 연산은 ==, != 만 가능함

참고

- 논리형 스칼라도 숫자형 연산 가능 → 컴퓨터는 TRUE/FALSE를 1과 0 숫자로 인식
- 수치 연산자는 스칼라 뿐 아니라 아래에서 다룰 벡터, 행렬, 리스트, 데이터프레임 객체의 연산에 사용 가능
- &/|와 &&/||는 동일하게 AND/OR를 의미하지만 연산 결과가 다름.
- &의 연산 대상이 벡터인 경우 벡터 구성 값 각각에 대해 & 연산을 실행 하지만 &&는 하나의 값(스칼라)에만 논리 연산이 적용(아래 예시 참고)

- 논리형 스칼라의 논리 및 비교 연산 예시

```
typeof(TRUE) # TRUE의 데이터 타입
```

```
[1] "logical"
```

```
TRUE & TRUE # TRUE 반환
```

```
[1] TRUE
```

```
TRUE & FALSE # FALSE 반환
```

```
[1] FALSE
```

```
# 아래 연산은 모두 TRUE 반환
```

```
TRUE | TRUE
```

```
[1] TRUE
```

```
TRUE | FALSE
```

```
[1] TRUE
```

```
# TRUE와 FALSE의 반대
```

```
!TRUE
```

```
[1] FALSE
```

```
!FALSE
```

```
[1] TRUE
```

```
# 전역변수 T에 FALSE 값 할당  
T <- FALSE  
T
```

```
[1] FALSE
```

```
T <- TRUE # 원상복구  
# TRUE/FALSE에 값을 할당할 수 없음  
TRUE <- 1
```

Error in TRUE <- 1: 대입에 유효하지 않은 (do_set) 좌변입니다

```
TRUE <- FALSE
```

Error in TRUE <- FALSE: 대입에 유효하지 않은 (do_set) 좌변입니다

```
# &()와 &&()의 차이  
1.01 <- c(TRUE, TRUE, FALSE, TRUE) # 논리형 값으로 구성된 벡터  
1.02 <- c(FALSE, TRUE, TRUE, TRUE)  
1.01 & 1.02 # 1.01과 1.02 각 원소 별 & 연산
```

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

```
1.01 && 1.02 # 1.01과 1.02의 첫 번째 원소에 대해 & 연산
```

```
[1] FALSE
```

```
# 비교 연산자  
x <- 9  
y <- 4  
# x > y 의 반환값 데이터 타입  
typeof(x > y)
```

```
[1] "logical"
```

```
# 논리형 값 반환  
x > y
```

```
[1] TRUE
```

```
x < y
```

```
[1] FALSE
```

```
x == y
```

```
[1] FALSE
```

```
x != y
```

```
[1] TRUE
```

2.2.5 결측값(missing value)

- 결측치 지정 상수: NA → R과 다른 언어의 가장 큰 차이점 중 하나
- 예를 들어 4명의 통계학과 학생 중 3명의 통계학 개론 중간고사 점수가 각각 80, 90, 75점이고 4번 째 학생의 점수가 없는 경우 NA로 결측값 표현
- `is.na()` 함수를 이용해 해당 값이 결측을 포함하고 있는지 확인

```
one <- 80; two <- 90; three <- 75; four <- NA  
four
```

```
[1] NA
```

```
# 'is.na()' 결측 NA가 포함되어 있으면 TRUE  
is.na(four)
```

```
[1] TRUE
```



`is.na(object_name)`: 객체를 구성하고 있는 원소 중 NA를 포함하고 있는지 확인 → NA를 포함하면 TRUE, 아니면 FALSE 반환

참고: 자료에 NA가 포함된 경우 연산 결과는 모두 NA가 반환

```
NA + 1
```

```
[1] NA
```

```
NA & TRUE
```

```
[1] NA
```

```
NA <= 3
```

```
[1] NA
```

2.2.6 NULL 값

- NULL: 초기화 되지 않은 변수 또는 객체를 지칭함
- `is.null()` 함수를 통해 객체가 NULL인지 판단

```
x <- NULL # NULL 설정  
is.null(x) # NULL 객체인지 판단
```

```
[1] TRUE
```

```
x <- 1  
is.null(x)
```

```
[1] FALSE
```



NA와 NULL의 차이점: 자료의 공백을 의미한다는 점에서 유사한 측면이 있으나 아래 내용처럼 큰 차이가 있음

- **NULL:** 값을 지정하지 않은 객체를 표현하는데 사용. 즉 아직 변수 또는 객체의 상태가 아직 미정인 상태를 나타냄
- **NA:** 데이터 값이 결측임을 지정해주는 논리형 상수

```
# NA와 NULL은 다름
```

```
x <- NA
```

```
is.null(NA)
```

```
[1] FALSE
```

```
is.na(NULL)
```

```
logical(0)
```

2.2.7 무한대/무한소/숫자아님

- **Inf:** 무한대($+\infty$, $1/0$)
- **-Inf:** 무한소($-\infty$, $-1/0$)
- **NaN:** 숫자아님(Not a Number, $0/0$)
- **is.finite()**, **is.infinite()**, **is.nan()** 함수를 통해 객체가 Inf 또는 NaN을 포함하는지 확인

```
x <- Inf  
is.finite(x)
```

```
[1] FALSE
```

```
is.infinite(x)
```

```
[1] TRUE
```

```
x <- 0/0  
is.nan(x)
```

```
[1] TRUE
```

```
is.infinite(x)
```

```
[1] FALSE
```



지금까지 요인형(factor)을 제외하고 R 언어에서 객체가 가질 수 있는 데이터 유형에 대해 알아봄. 요인형은 4주 차에 예정된 “R 자료형: 팩터, 테이블, 데이터 프레임”에서 상세하게 배울 예정임.

2.3 벡터(vector)

2.3.1 벡터의 특징

- 타 프로그래밍 언어의 배열(array)의 개념으로 **동일한 유형**의 데이터 원소가 하나 이상($n \times 1$, $n \geq 1$) 으로 구성된 자료 형태
- R 언어의 가장 기본적인 데이터 형태로 R에서 행해지는 모든 연산의 기본(vectorization) → 벡터 연산 시 반복구문(예: for loop)이 필요 없음.
- 2.2 절에서 기술한 **스칼라(scalar)**는 사실 1×1 벡터임
- 수학적으로 벡터는 아래와 같이 나타낼 수 있음

$$\mathbf{x} = [x_1, x_2, x_3, \dots, x_n]^T$$

- 벡터는 앞의 예시에서 본 바와 같이 c() 함수를 사용해 생성

```
# 숫자형 벡터
x <- c(2, 0, 2, 0, 0, 3, 2, 4)
x
```

[1] 2 0 2 0 0 3 2 4

```
# 문자형 벡터
y <- c("Boncho Ku", "R programming", "Male", "sophomore", "2020-03-24")
y
```

```
[1] "Boncho Ku"      "R programming" "Male"          "sophomore"
[5] "2020-03-24"
```

- 두 개 이상의 벡터는 `c()` 함수를 통해 결합 가능

- 함수 내, 구분자를 통해 결합

```
# 두 벡터의 결합 (1)
```

```
x <- 1:5
y <- 10:6
z <- c(x, y)
x
```

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

```
y
```

```
[1] 10 9 8 7 6
```

```
z
```

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

```
x <- 5:10
x1 <- x[1:3] # x 벡터에서 1에서 4번째 원소 추출
x2 <- c(x1, 15, x[4])
x2
```

```
[1] 5 6 7 15 8
```

- 서로 다른 자료형으로 벡터를 구성한 경우 표현력이 높은 자료형으로 변환한 값 반환
 - 예: 문자열 + 숫자로 구성된 벡터 → 문자형 벡터
 - 변환구조: NULL < raw < logical < integer < double < complex < character < list < expression

```
# 숫자형 벡터와 문자열 벡터 혼용  
k <- c(1, 2, "3", "4")  
k
```

```
[1] "1" "2" "3" "4"
```

```
is.numeric(k) # 벡터가 숫자형인지 판단하는 함수
```

```
[1] FALSE
```

```
is.character(k) # 벡터가 문자열인지 판단하는 함수
```

```
[1] TRUE
```

```
# 숫자형 벡터와 문자열 벡터 결합  
x <- 1:3  
y <- c("a", "b", "c")  
z <- c(x, y)  
z
```

```
[1] "1" "2" "3" "a" "b" "c"
```

```
is.numeric(z)
```

```
[1] FALSE
```

```
is.character(z)
```

```
[1] TRUE
```

```
# 숫자형 벡터와 논리형 벡터 결합  
x <- 9:4  
y <- c(TRUE, TRUE, FALSE)  
z <- c(x, y)  
  
z # TRUE/FALSE 가 1과 0으로 변환
```

```
[1] 9 8 7 6 5 4 1 1 0
```

```
is.numeric(z)
```

```
[1] TRUE
```

```
is.logical(z)
```

```
[1] FALSE
```

- 두 벡터는 중첩이 불가능 → 동일한 벡터 2개를 결합 시 단일 차원 벡터 생성

```
x <- y <- 1:3 # x와 y 동시에 [1, 2, 3] 할당
```

```
x
```

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

```
y
```

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

```
z <- c(x, y)
```

```
z
```

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

- 벡터 각 원소에 이름 부여 가능
 - `names()` 함수를 이용해 원소 이름 지정
 - 사용 프로토타입: `names(x) <- 문자열 벡터`, 단 x와 이름에 입력할 문자열 벡터의 길이는 같아야 함.
 - `c()` 함수에서 직접 이름 지정 → `c(atom_name1 = value, atom_name2 = value, ...)`

```
x <- c("Boncho Ku", "R programming", "Male", "sophomore", "2020-03-24")

# 벡터 원소 이름 지정
names(x) <- c("name", "course", "gender", "grade", "date")

x
```

| name | course | gender | grade | date |
|-------------|-----------------|--------|-------------|--------------|
| "Boncho Ku" | "R programming" | "Male" | "sophomore" | "2020-03-24" |

```
y <- c(a = 10, b = 6, c = 9)

names(y)
```

[1] "a" "b" "c"

- 벡터의 길이(차원) 확인

- `length()` 또는 `NROW()` 사용

```
x <- 1:50

# 객체의 길이 반환
# length(): 벡터, 행렬인 경우 원소의 개수, 데이터프레임인 경우 열의 개수 반환
length(x)
```

[1] 50

```
# NROW(): 벡터인 경우 원소의 개수, 행렬, 데이터 프레임인 경우 행의 개수 반환
NROW(x)
```

[1] 50

2.3.2 벡터의 연산

- 원소 단위 사칙연산 및 비교연산 수행 → 벡터화 연산(vectorized operation)
 - 예를 들어 $x = [1, 2, 3]^T$ 이고, $y = [2, 3, 4]^T$ 라고 할 때 $x + y$ 의 연산은 아래와 같음

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} + \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 3 \\ 5 \\ 7 \end{bmatrix}$$

- * 연산 시 행렬 대수학에서 벡터의 곱(product)과 다름을 주의

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} * \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 2 \\ 6 \\ 12 \end{bmatrix}$$

```
x <- 1:3; y <- 2:4
length(x); length(y)
```

[1] 3

[1] 3

```
x; y
```

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

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

```
# 사칙연산(+, -, *, /)
# 백터 vs. 백터
x + y
```

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

```
x - y
```

```
[1] -1 -1 -1
```

```
x * y
```

```
[1] 2 6 12
```

```
x / y
```

```
[1] 0.5000000 0.6666667 0.7500000
```

```
# 그외 연산
# 나머지(remainder)
y %% x
```

```
[1] 0 1 1
```

```
# 몫(x)(quotient)
y %/% x
```

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

```
# 멱승(exponent)
y ^ x
```

```
[1] 2 9 64
```

- 차원이 서로 맞지 않는 경우 작은 차원(짧은 쪽)의 백터를 재사용함

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} + [5] = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} + \begin{bmatrix} 5 \\ 5 \\ 5 \end{bmatrix} = \begin{bmatrix} 6 \\ 7 \\ 8 \end{bmatrix}$$

```
# 벡터(n by 1) vs. 스칼라(1 by 1)
x * 5 # x의 길이 만큼 재사용(반복) 후 곱 연산 수행
```

```
[1] 5 10 15
```

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

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

```
y
```

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

```
length(x); length(y)
```

```
[1] 5
```

```
[1] 3
```

```
# x의 길이가 5이고 y의 길이가 3이기 때문에 5를 맞추기 위해
```

```
# y의 원소 중 1-2 번째 원소를 재사용
```

```
x + y
```

Warning in x + y: 두 객체의 길이가 서로 배수관계에 있지 않습니다

```
[1] 4 4 7 7 7
```

```
x / y
```

Warning in x/y: 두 객체의 길이가 서로 배수관계에 있지 않습니다

```
[1] 1.0000000 0.3333333 0.7500000 2.5000000 1.3333333
```

- 연산 순서는 일반적인 사칙연산의 순서를 준용
 - 단 1단위 수열을 생성하는 : 연산자가 사칙연산을 우선함

```
# 연산 우선 순위  
1:5 * 3
```

```
[1] 3 6 9 12 15
```

```
1:(5 * 3)
```

```
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

- 논리형 값으로 구성된 벡터의 기본 연산 시 수치형으로 변환된 연산 결과를 반환

```
# 논리형 벡터  
b1 <- c(TRUE, TRUE, FALSE, TRUE, TRUE, TRUE, FALSE, FALSE)  
b2 <- c(FALSE, TRUE, TRUE, TRUE, TRUE, TRUE, FALSE, TRUE)  
  
is.numeric(b1); is.numeric(b2)
```

```
[1] FALSE
```

```
[1] FALSE
```

```
is.logical(b1); is.logical(b2)
```

```
[1] TRUE
```

```
[1] TRUE
```

```
# 논리형 벡터 연산
b3 <- b1 + b2
is.numeric(b3)
```

[1] TRUE

b3

[1] 1 2 1 2 2 2 0 1

b1 - b2

[1] 1 0 -1 0 0 0 0 -1

b1 * b2

[1] 0 1 0 1 1 1 0 0

b1/b2

[1] Inf 1 0 1 1 1 NaN 0

- 두 벡터 간 비교 연산은 사칙연산과 마찬가지로 각 원소단위 연산을 수행하고 논리형 벡터 반환
 - 재사용 규칙은 그대로 적용됨

```
# 두 벡터의 비교 연산  
x <- c(2, 4, 3, 10, 5, 9)  
y <- c(3, 4, 6, 2, 10, 7)  
  
x == y
```

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

```
x != y
```

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

```
x > y
```

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

```
x < y
```

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

```
x >= y
```

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

```
x <= y
```

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

```
# 비교 연산 시 두 벡터의 길이가 다른 경우
```

```
x <- 1:5; y <- 2:4
```

```
x == y
```

```
Warning in x == y: 두 객체의 길이가 서로 배수관계에 있지 않습니다
```

```
[1] FALSE FALSE FALSE FALSE FALSE
```

```
x != y
```

```
Warning in x != y: 두 객체의 길이가 서로 배수관계에 있지 않습니다
```

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

```
x > y
```

```
Warning in x > y: 두 객체의 길이가 서로 배수관계에 있지 않습니다
```

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

```
x < y
```

Warning in x < y: 두 객체의 길이가 서로 배수관계에 있지 않습니다

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

```
x >= y
```

Warning in x >= y: 두 객체의 길이가 서로 배수관계에 있지 않습니다

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

```
x <= y
```

Warning in x <= y: 두 객체의 길이가 서로 배수관계에 있지 않습니다

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

- 문자열 벡터의 연산은 == 또는 != 만 가능(사칙연산 불가능)

```
# 문자열 벡터 연산 (==, !=)
c1 <- letters[1:5]
# a-z로 구성된 벡터에서 1-2, 6-8 번째 원소 추출
c2 <- letters[c(1:2, 6:8)]
c1
```

```
[1] "a" "b" "c" "d" "e"
```

```
c2
```

```
[1] "a" "b" "f" "g" "h"
```

```
c1 == c2
```

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

```
c1 != c2
```

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

- NA를 포함한 두 벡터 연산 시 동일 위치에 NA가 존재하면 어떤 연산이
는 NA 값을 반환

```
# 결측을 포함한 벡터
x <- c(1:10, c(NA, NA))
y <- c(NA, NA, 1:10)
x
```

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

```
y
```

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

```
is.na(x); is.na(y)
```

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

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

```
# 결측을 포함한 벡터의 연산
```

```
x + y
```

```
[1] NA NA 4 6 8 10 12 14 16 18 NA NA
```

```
x / y
```

```
[1] NA NA 3.000000 2.000000 1.666667 1.500000 1.400000 1.333333  
[9] 1.285714 1.250000 NA NA
```

```
x < y
```

```
[1] NA NA FALSE FALSE FALSE FALSE FALSE FALSE FALSE NA NA
```

```
x > y
```

```
[1] NA NA TRUE TRUE TRUE TRUE TRUE TRUE TRUE NA NA
```

- NULL이 벡터에 포함되더라도 벡터의 길이에는 변동이 없음

```
# NULL을 포함한 벡터
x <- c(1, 2, 3, NULL, NULL, NULL) # 길이가 6?
length(x)
```

[1] 3

x

[1] 1 2 3

2.3.3 벡터의 색인(indexing)

- 벡터의 특정 위치에 있는 원소를 추출
- 색인(indexing)을 통해 벡터의 원소에 접근 가능
- 타 언어는 대체로 첫 번째 색인이 0에서 시작하지만, R은 1부터 시작
- x[i]: 벡터 x의 i번째 요소
- x[start:end]: x의 start부터 end까지 값 반환

```
x <- c(1.2, 3.1, 4.2, 2.8, 3.3)
x[3] # x 원소 중 3 번째 원소 추출
```

[1] 4.2

```
# x 원소 중 2-3번째 원소 추출  
x[2:3]
```

```
[1] 3.1 4.2
```

- $x[-i]$: 벡터 x 에서 i 번 째 요소를 제외한 나머지 값 반환

```
# x의 3 번째 원소 제거  
x[-3]
```

```
[1] 1.2 3.1 2.8 3.3
```

```
# 맨 마지막 원소(5 번째) 제거  
# 아래 script는 동일한 결과 출력  
x[1:(length(x) - 1)]
```

```
[1] 1.2 3.1 4.2 2.8
```

```
x[-length(x)]
```

```
[1] 1.2 3.1 4.2 2.8
```

- $x[\text{idx_vec}]$: idx_vec 가 인덱싱 벡터라고 할 때 idx_vec 에 지정된 요소를 얻어옴. 일반적으로 idx_vec 는 벡터의 행 순서 번호 또는 각 벡터 원소의 이름에 대응하는 문자열 벡터를 인덱싱 벡터로 사용할 수 있음.

```
# 벡터를 이용한 인덱싱
# x 원소 중 1, 5번째 원소 추출
x[c(1, 5)] # c(1,5)는 벡터
```

[1] 1.2 3.3

```
v <- c(1, 4)
x[v]
```

[1] 1.2 2.8

```
# 인덱스 번호 중복 가능
x[c(1, 2, 2, 4)]
```

[1] 1.2 3.1 3.1 2.8

```
# 원소 이름으로 인덱싱
# 원소 이름 지정
names(x) <- paste0("x", 1:length(x)) # 문자열 "x"와 숫자 1:5(벡터 길이)를 결합한 문자열 반환
x["x3"]
```

x3

4.2

```
x[c("x2", "x4")]
```

```
x2 x4  
3.1 2.8
```

- 필터링(filtering): 특정한 조건을 만족하는 원소 추출
 - 비교 연산자를 이용한 조건 생성 → 논리값을 이용한 원소 추출

```
z <- c(5, 2, -3, 8)  
# z의 원소 중 z의 제곱이 8보다 큰 원소 추출  
w <- z[z^2 > 8]  
w
```

```
[1] 5 -3 8
```

- 작동 원리
 - $z^2 > 8$ 은 벡터 z의 모든 원소 제곱값이 8 보다 큰 케이스를 논리형 값으로 반환

```
z^2
```

```
[1] 25 4 9 64
```

```
idx <- z^2 > 8  
idx
```

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

```
z[idx]
```

```
[1] 5 -3 8
```

- 특정 조건을 만족하는 벡터의 위치에 임의의 값을 치환할 수 있음

```
# 위 벡터 z 의 원소 중 z^2 > 8 인 원소의 값을 0으로 치환
z[idx] <- 0
```

2.3.4 벡터 관련 함수

- c() 함수 외에 R은 벡터 생성을 위해 몇 가지 유용한 함수를 제공함

seq 계열 함수

보다 자세한 사용 설명은 `help(seq)` 참고

`seq()`: 등차 수열 생성하는 함수로 `from`에서 `end` 까지 숫자 내에서 공차(간격)가 `by` 인 수열 생성

```
# seq(): 수열 생성 함수
seq(
  from, # 시작값
  to,   # 끝값
```

```
by      # 공차(증가치)
)

# 거리 인수
# length.out = n
#   - 생성하고자 하는 벡터의 길이가 n인 수열 생성
# along.with = 1:n
#   - index가 1에서 n 까지 길이를 갖는 수열 생성
```

- 사용 예시

```
x <- seq(from = 2, to = 30, by = 2)
x
```

[1] 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30

```
# 간격이 꼭 정수가 아니어도 사용 가능
x <- seq(from = 0, to = 3, by = 0.2)
```

```
# by 대신 length.out 으로 생성된 수열의 길이 조정
x <- seq(from = -3, to = 3, length.out = 10)
x
```

[1] -3.0000000 -2.3333333 -1.6666667 -1.0000000 -0.3333333 0.3333333
[7] 1.0000000 1.6666667 2.3333333 3.0000000

```
# from, to 인수 없이 length.out=10 인 경우
seq(length.out = 10)
```

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

```
# by 대신 along.width
seq(along.with=1:10)
```

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

```
seq(1, 5, along.with=1:10)
```

```
[1] 1.000000 1.444444 1.888889 2.333333 2.777778 3.222222 3.666667 4.111111
[9] 4.555556 5.000000
```

```
# 벡터 x에 seq() 함수 적용 시 1:length(x) 값 반환
seq(x)
```

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

seq_along(): 주어진 객체의 길이 만큼 1부터 1 간격의 수열 생성

- seq() 함수와 매우 유사하나, 무조건 1부터 시작해서 인수로 seq()의 along.with 값을 이용한 함수
- seq() 함수보다 조금 빠름
- 사용 예시

```
# 1부터 x 벡터의 길이 까지 1 단위 수열 값 반환
seq_along(x)
```

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

seq_len(): 인수로 받은 값 만큼 1부터 해당 값 까지 1 간격의 수열 생성

- `seq()` 함수의 인수 중 `length.out` 값을 이용한 함수
- 사용 예시

```
# 1부터 n 까지 1 단위 수열 값 반환
seq_len(10)
```

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

rep 계열 함수

help(rep)을 통해 상세 내용 참고

rep(): 주어진 벡터의 원소를 반복

```
# rep(): 벡터 또는 벡터의 개별 원소를 반복한 값 반환
rep(
  x, # 반복할 값이 저장된 벡터
  times, # 전체 벡터의 반복 횟수
  each # 개별 원소의 반복 횟수
)
```

- 사용 예시

```
x <- rep(4, 5) # 4를 5번 반복
x
```

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

```
# x <- c(1:3) 전체를 3번 반복한 벡터 반환
x <- c(1:3)
xr1 <- rep(x, times = 3)
xr1
```

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

```
# 벡터 x의 각 원소를 4번씩 반복한 벡터 반환
xr2 <- rep(x, each = 4)
xr2
```

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

```
# 벡터 x의 각 원소를 3번 반복하고 해당 벡터를 2회 반복
xr3 <- rep(x, each = 3, times = 2)
xr3
```

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

```
# 문자형 벡터의 반복  
# 아래 sex 벡터의 각 원소를 2 번 반복하고 해당 벡터를 4회 반복  
sex <- c("Male", "Female")  
sexr <- rep(sex, each = 2, times = 4)  
sexr
```

```
[1] "Male"    "Male"    "Female"  "Female"  "Male"    "Male"    "Female"  "Female"  
[9] "Male"    "Male"    "Female"  "Female"  "Male"    "Male"    "Female"  "Female"
```

rep.int() & rep_len(): rep() 함수의 simple 버전으로 속도(performance)가 요구되는 프로그래밍 시 사용

- 사용 예시

```
# 1:5 벡터를 3 번 반복  
rep.int(1:5, 3)
```

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

```
# 불완전한 사이클로 벡터 반복  
rep_len(1:5, length.out = 7)
```

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

Filtering 관련 함수

```
help(subset) 참고
```

subset(): 기존 필터링 방식과 비교할 때 NA를 처리하는 방식에서 차이를 보임

- 벡터 뿐 아니라 앞으로 배울 행렬 및 데이터프레임 객체에도 적용 가능

```
x <- c(6, 1:3, NA, NA, 12)  
x
```

```
[1] 6 1 2 3 NA NA 12
```

```
# 일반적 필터링 적용  
x[x > 5]
```

```
[1] 6 NA NA 12
```

```
# subset() 함수 적용  
subset(x, x > 5)
```

```
[1] 6 12
```

which(): 한 벡터에서 특정 조건에 맞는 위치(인덱스)를 반환

```
# which(): 논리형 벡터를 인수로 받고 해당 논리형 벡터가 참인 index 반환
which(
  logical_vec # 논리형 벡터
)
```

- 사용 예시

```
x <- c(3, 8, 3, 1, 7)

# x의 원소값이 3인 index 반환
which(x == 3)
```

```
[1] 1 3
```

```
# x의 원소가 4보다 큰 원소의 index 반환
which(x > 4)
```

```
[1] 2 5
```

```
# 9월(Sep)과 12월(Dec)와 같은 원소 index
# month.abb: R 내장 벡터로 월 약어(Jan ~ Dec)를 저장한 문자열 벡터
which(month.abb == c("Sep", "Dec"))
```

```
[1] 9 12
```

```
# 조건을 만족하는 원소가 존재하지 않는다면?
x <- which(x > 9)
x
```

```
integer(0)
```

```
length(x) # 길이가 0인 벡터 반환 is.null(x) == TRUE ??
```

```
[1] 0
```

```
is.null(x)
```

```
[1] FALSE
```

```
# 특정 조건 만족 여부를 확인
# any(condition) -> 하나라도 condition을 만족하는 원소가 존재하는지 판단
# TRUE 또는 FALSE 값 반환
any(x > 9)
```

```
[1] FALSE
```

집합 관련 함수

- 벡터는 숫자, 문자열의 묶음, 즉 원소들의 집합(set)으로 볼 수 있기 때문에 집합 연산이 가능
- 두 집합을 X 와 Y 로 정의 했을 때 아래와 같은 집합 연산 가능

- **setequal(X, Y)**: X와 Y가 동일한지 판단 ($X = Y$) → 논리값 TRUE 또는 FALSE 반환

```
x <- y <- c(1, 9, 7, 3, 6)
setequal(x, y)
```

```
[1] TRUE
```

- **union(X, Y)**: X와 Y의 합집합 ($X \cup Y$)

```
y <- c(1, 9, 8, 2, 0, 3)
union(x, y)
```

```
[1] 1 9 7 3 6 8 2 0
```

- **intersect(X, Y)**: X와 Y의 교집합 ($X \cap Y$)

```
intersect(x, y)
```

```
[1] 1 9 3
```

- **setdiff(X, Y)**: X와 Y의 차집합 ($X - Y$)

```
setdiff(x, y)
```

```
[1] 7 6
```

```
setdiff(y, x)
```

```
[1] 8 2 0
```

- X %in% Y: X(기준)가 집합 Y의 원소인지 논리값 반환

```
x <- c("apple", "banana", "strawberry", "mango", "peach", "orange")
y <- c("strawberry", "orange", "mango")

x %in% y
```

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

```
y %in% x
```

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

두 벡터의 동일성 테스트

- 두 벡터가 동일한지 테스트 하기 위해 x == y 연산의 반환 값은 위의 예제에서 확인한 것처럼 각 원소에 대한 논리값을 반환(아래 예제 확인)

```
x <- 1:3
y <- c(1, 3, 4)
x == y
```

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

- 단지 두 벡터가 동일한지 아닌지를 확인하기 위해서는 하나의 논리값만 필요한 경우 all() 사용

```
all(x == y)
```

```
[1] FALSE
```

- 보다 나은 방법으로 identical() 함수 적용

```
# 두 객체의 동일성 여부 테스트  
identical(x, y)
```

```
[1] FALSE
```

- identical() 함수는 벡터가 갖는 데이터 타입의 동일성 까지 체크함

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

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

```
y
```

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

```
# all() 함수로 동일성 확인  
all(x == y)
```

```
[1] TRUE
```

```
# identical 함수로 동일성 확인  
identical(x, y)
```

```
[1] FALSE
```

```
# x, y 데이터 타입 확인  
typeof(x)
```

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

```
typeof(y)
```

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

2.4 리스트(list)

- **리스트(list)**: (key, value) 형태로 데이터를 저장한 배열(벡터)
- 서로 다른 데이터 타입을 가진 객체를 원소로 가질 수 있는 벡터

- 예: 한 리스트 안에는 상이한 데이터 타입(숫자형, 문자형, 논리형 등)을 갖는 원소(객체)들을 포함할 수 있음



리스트 예시: 통계프로그래밍언어 중간고사 성적 테이블

- 중간고사 성적 테이블은 이름, 학번, 출석률, 점수, 등급으로 이루어졌다고 가정하면 “김상자”의 성적 리스트는 다음과 같이 나타낼 수 있음
- LIST(이름 = "김상자", 학번 = "202015115", 점수 = 95, 등급 = "A-")
- 위 record에서 보듯이 문자형과 숫자형이 LIST 안에 같이 표현되고 있음

- 위 record를 벡터 생성함수 c()로 생성한 경우

```
# 벡터로 위 record를 입력한 경우
vec <- c(`이름` = "김상자", `학번` = "202015115",
           `점수` = 95, `등급` = "A-")
vec
```

| 이름 | 학번 | 점수 | 등급 |
|-------|-------------|------|------|
| "김상자" | "202015115" | "95" | "A-" |

```
typeof(vec)
```

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



객체 명칭 규칙을 벗어나는 이름을 객체명으로 사용하고 싶다면 다음과 같이 훌따옴표 ‘object_name‘ 표시를 통해 사용 가능함

```
> #공백이 있는 이름을 객체 명칭으로 사용
> `golf score` <- c(75, 82, 92)
> `golf score`
```

[1] 75 82 92

```
> `3x` <- c(3, 6, 9, 12)
> `3x`
```

[1] 3 6 9 12

2.4.1 리스트 생성

- `list()` 함수를 사용해 list 객체 생성

```
# list 함수 사용 prototype
list(name_1 = object_1, ..., name_m = object_m)

# name_1, ..., name_m: 리스트 원소 이름
# object_1, ..., object_m: 리스트 원소에 대응한 객체
```

- 중간고사 성적 테이블 예시

```
# lst 객체 생성
lst <- list(`이름` = "김상자",
            `학번` = "202015115",
            `점수` = 95,
            `등급` = "A-")
lst
```

\$이름

[1] "김상자"

\$학번

[1] "202015115"

\$점수

[1] 95

\$등급

[1] "A-"

```
# lst 내 객체의 데이터 타입 확인
# lapply(): lst 객체에 동일한 함수 적용 (추후 학습)
lapply(lst, typeof)
```

\$이름

[1] "character"

\$학번

[1] "character"

\$점수

[1] "double"

\$등급

[1] "character"

- 리스트 원소에 이름이 부여된 경우 `names()`를 통해 확인 가능

```
names(lst)
```

```
[1] "이름" "학번" "점수" "등급"
```

- 이름(name_1, ..., name_n) 없이도 리스트 생성 가능하나, 가급적 이름을 부여 하는 것이 더 명확

```
list("김상자", "202015115", 95, "A-")
```

```
[[1]]
```

```
[1] "김상자"
```

```
[[2]]
```

```
[1] "202015115"
```

```
[[3]]
```

```
[1] 95
```

```
[[4]]
```

```
[1] "A-"
```

- 리스트는 벡터이므로 vector() 함수를 통해 생성 가능

```
# 길이가 1이고 객체가 NULL인 리스트 생성
z <- vector(mode = "list", length=1)
z
```

```
[[1]]
```

```
NULL
```

- 리스트의 값이 어떤 객체든 관계 없음

```
x <- list(name = c("A", "B", "C"),
           salary = c(500, 450, 600), union = T)
x
```

```
$name
[1] "A" "B" "C"
```

```
$salary
[1] 500 450 600
```

```
$union
[1] TRUE
```

2.4.2 리스트 색인

- 리스트에 포함된 객체에 접근하는 기본적으로 벡터의 색인 방법과 동일하게 색인 번호 또는 키(이름)을 통해 접근 가능
- 리스트에 포함된 모든 객체의 원소값을 쉽게 확인하는 함수는 `unlist()`임

```
lval <- unlist(x)
typeof(lval)
```

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

TABLE 2.5: 리스트 데이터 접근 방법

| 색인방법 | 동작 |
|--|--|
| <code>x\$name</code> | 리스트 <code>x</code> 에서 객체명(<code>name</code>)에 해당하는 객체에 접근 |
| <code>x[[i]]</code> 또는 <code>x[[name]]</code> | 리스트 <code>x</code> 에서 <code>i</code> 번째 또는 <code>name</code> 에 해당하는 객체 반환 |
| <code>x[i]</code> 또는 <code>x[name]</code> | 리스트 <code>x</code> 에서 <code>i</code> 번째 또는 <code>name</code> 에 해당하는 부분 리스트 반환 |

- `x$name`을 통해 리스트 내 객체 접근

```
lst$`학번`
```

```
[1] "202015115"
```

- `x[[i]]` 또는 `x[[name]]` 을 통해 리스트 내 객체 접근

```
lst[[2]]
```

```
[1] "202015115"
```

```
z <- lst[["학번"]]
z
```

```
[1] "202015115"
```

```
typeof(z)
```

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

- `x[i]` 또는 `x[name]` 을 통해 리스트 내 부분 리스트 추출

```
lst[2]
```

\$학번

```
[1] "202015115"
```

```
j <- lst["학번"]  
j
```

\$학번

```
[1] "202015115"
```

```
typeof(j)
```

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

- 리스트 또한 벡터로 볼 수 있기 때문에 여러 개의 부분 리스트 추출 가능

```
# 리스트 lst에서 1 ~ 3 번째 까지 부분 리스트 추출
lst[1:3]
```

\$이름

```
[1] "김상자"
```

\$학번

```
[1] "202015115"
```

\$점수

```
[1] 95
```

- 리스트를 구성하는 객체 내 색인

```
x
```

\$name

```
[1] "A" "B" "C"
```

\$salary

```
[1] 500 450 600
```

\$union

```
[1] TRUE
```

```
# salary에서 2-3번째 원소 추출
```

```
x$salary[2:3]
```

```
[1] 450 600
```

```
x[[2]][2:3]
```

```
[1] 450 600
```

```
x[["salary"]][2:3]
```

```
[1] 450 600
```

```
# 부분 리스트도 길이가 1인 리스트이므로,  
# 부분 리스트 내 객체 접근 시 리스트 접근이 선행  
# x의 2번째 부분 리스트에서 첫 번째 객체의 2-3번째 원소 추출  
x[2][[1]][2:3]
```

```
[1] 450 600
```

- 리스트의 길이 반환: 벡터와 마찬가지로 `length()` 함수 적용 가능

```
length(lst); length(x)
```

```
[1] 4
```

```
[1] 3
```

2.4.3 리스트에 원소 추가/제거

- 주어진 리스트 x에 새로운 원소를 x\$new_obj <- value 명령어 형태로 추가
- 이미 존재하고 있는 리스트 원소 제거는 x\$exist_obj <- NULL 형태로 제거

```
# 리스트 lst 0/ 5호 차 퀴즈 점수 추가
lst$quiz <- c(10, 8, 9, 9, 8)
```

```
# 리스트 lst0/ 원소 quiz 제거
lst$quiz <- NULL
lst
```

\$이름
[1] "김상자"

\$학번
[1] "202015115"

\$점수
[1] 95

\$등급
[1] "A-"

```
# 벡터 색인을 이용해 원소 추가 가능
lst[[5]] <- c(10, 8, 9, 9, 8)
lst
```

```
$이름
```

```
[1] "김상자"
```

```
$학번
```

```
[1] "202015115"
```

```
$점수
```

```
[1] 95
```

```
$등급
```

```
[1] "A-"
```

```
[[5]]
```

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

```
# 부분 리스트 괄호에서도 색인 통해 추가/삭제 가능
```

```
lst[5] <- NULL
```

```
lst
```

```
$이름
```

```
[1] "김상자"
```

```
$학번
```

```
[1] "202015115"
```

```
$점수
```

```
[1] 95
```

```
$등급
```

```
[1] "A-"
```

```
# 여러 개의 리스트 동시 추가/삭제 가능
lst[5:9] <- c(10, 8, 9, 9, 8)
lst
```

\$이름

```
[1] "김상자"
```

\$학번

```
[1] "202015115"
```

\$점수

```
[1] 95
```

\$등급

```
[1] "A-"
```

[[5]]

```
[1] 10
```

[[6]]

```
[1] 8
```

[[7]]

```
[1] 9
```

[[8]]

```
[1] 9
```

[[9]]

```
[1] 8
```

```
lst[5:9] <- NULL  
lst
```

\$이름
[1] "김상자"

\$학번
[1] "202015115"

\$점수
[1] 95

\$등급
[1] "A-"

2.4.4 리스트의 결합

- 두 개 이상의 리스트를 결합 시 c() 사용

```
# 리스트 lst와 x 결합  
c(lst, x)
```

\$이름
[1] "김상자"

\$학번
[1] "202015115"

\$점수
[1] 95

\$등급

```
[1] "A-"
```

\$name

```
[1] "A" "B" "C"
```

\$salary

```
[1] 500 450 600
```

\$union

```
[1] TRUE
```



리스트 내에 리스트를 가질 수 있다. 이를 재귀 리스트(recursive list)라고 한다. 예를 들어 위 예제에서 각 학생의 성적 데이터가 리스트로 구성되어 있다면, 전체 성적 데이터베이스는 리스트로 구성된 리스트임. 아래 예제 처럼 간단한 재귀 리스트 구현이 가능

```
kim <- list(id = "20153345", sex = "Male", score = 85, grade = "B+")
lee <- list(id = "20153348", sex = "Female", score = 75, grade = "B0")

gr <- list(kim=kim, lee=lee)
gr
```

\$kim

\$kim\$id

```
[1] "20153345"
```

\$kim\$sex

```
[1] "Male"
```

```
$kim$score  
[1] 85
```

```
$kim$grade  
[1] "B+"
```

```
$lee  
$lee$id  
[1] "20153348"
```

```
$lee$sex  
[1] "Female"
```

```
$lee$score  
[1] 75
```

```
$lee$grade  
[1] "B0"
```

2.5 행렬(*matrix*)



학습목표(3 주차): 행렬, 배열, 요인형과 테이블에 대해 살펴보고, 이들 객체에 대한 연산과 연관된 함수에 대해 익힌다.

행렬의 정의

- 동일한 데이터 타입의 원소로 구성된 2차원 데이터 구조

- $n \times 1$ 차원 벡터 p 개로 둑여진 데이터 덩어리 $\rightarrow n \times p$ 행렬로 명칭함
- 행렬의 형태

$$\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \cdots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{bmatrix}$$

- R에서 행렬은 동일한 유형의 데이터 타입으로 구성 가능 \rightarrow 첫 번째 행은 숫자형, 두 번째 행은 문자열로 입력해도 행렬을 만들 수 있지만, 표현력이 더 높은 문자형 행렬 반환
- 행렬의 내부 저장공간은 “열 우선 배열”
- 행렬 생성을 위한 R 함수는 `matrix()` 함수이고 사용 형태는 아래와 같음

```
# matrix(): 행렬 생성 함수
# 상세 내용은 help(matrix)를 통해 확인

matrix(data, # 행렬을 생성할 데이터 벡터
       nrow, # 행의 개수 (정수)
       ncol, # 열의 개수 (정수)
       byrow, # TRUE: 행 우선, FALSE: 열 우선
       # default = FALSE
       dimnames # 행렬을 각 차원에 부여할 이름 (리스트)
       )
```

- 행렬 생성 예시

```
# byrow = FALSE
x <- matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9), nrow = 3, ncol = 3)
x
```

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

```
# byrow = TRUE
x <- matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9), nrow = 3, ncol = 3, byrow = T)
x
```

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

- 행의 개수(*nrow*)나 열의 개수(*ncol*)로 나머지를 추정 가능하다면 둘 중 어떤 인수도 생략 가능

```
x <- matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9), ncol = 3)
x
```

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

```
x <- matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9), nrow = 3)
x
```

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

- `nrow × ncol` 이 입력한 데이터(벡터)의 길이보다 작거나 큰 경우

```
# length(x) < nrow * ncol 인 경우
# nrow * ncol에 해당하는 길이 만큼
# x의 원소를 사용해 행렬 생성
x <- c(1, 2, 3, 4, 5, 6, 7, 8, 9)
y <- matrix(x, nrow = 3, ncol = 4)
```

Warning in `matrix(x, nrow = 3, ncol = 4)`: 데이터의 길이[9]가 열의 개수[4]의 배수
가 되지 않습니다

```
y
```

```
[,1] [,2] [,3] [,4]
[1,]    1    4    7    1
[2,]    2    5    8    2
[3,]
```

```
# length(x) > nrow * ncol 인 경우  
# x의 첫 번째 원소부터 초과하는 만큼  
# x 원소의 값을 재사용  
z <- matrix(x, nrow = 2, ncol = 3)
```

Warning in `matrix(x, nrow = 2, ncol = 3)`: 데이터의 길이[9]가 행의 개수[2]의 배수 가 되지 않습니다

z

```
[,1] [,2] [,3]  
[1,] 1 3 5  
[2,] 2 4 6
```

- 행렬 구성 시 길이에 대한 약수가 아닌 값을 `nrow` 또는 `ncol`의 인수로 받은 경우

```
# x (length=9)로 행렬 생성 시 nrow=4 를  
# 인수로 입력한 경우  
h <- matrix(x, nrow = 4)
```

Warning in `matrix(x, nrow = 4)`: 데이터의 길이[9]가 행의 개수[4]의 배수 가 되지 않 습니다

```
h
```

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

```
# x (length=9)로 행렬 생성 시 ncol=2 만
# 인수로 입력한 경우
h <- matrix(x, nrow = 2)
```

Warning in matrix(x, nrow = 2) : 데이터의 길이[9]가 행의 개수[2]의 배수가 되지 않습니다

```
h
```

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

2.5.1 행렬의 연산

- 선형대수(linear algebra)에서 배우는 행렬-스칼라, 행렬-행렬 간 연산 가능

행렬-스칼라 연산

합 연산: 스칼라가 자동적으로 행렬의 차원에 맞춰서 재사용

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} + 4 = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} + \begin{bmatrix} 4 & 4 & 4 \\ 4 & 4 & 4 \\ 4 & 4 & 4 \end{bmatrix} = \begin{bmatrix} 5 & 6 & 7 \\ 8 & 9 & 10 \\ 11 & 12 & 13 \end{bmatrix}$$

```
x <- matrix(1:9, 3, 3, byrow = T)
x + 4
```

```
[,1] [,2] [,3]
[1,]    5    6    7
[2,]    8    9   10
[3,]   11   12   13
```

곱 연산

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \times 4 = \begin{bmatrix} 4 & 8 & 12 \\ 16 & 20 & 24 \\ 28 & 32 & 36 \end{bmatrix}$$

```
x*4
```

```
[,1] [,2] [,3]
[1,]    4    8   12
[2,]   16   20   24
[3,]   28   32   36
```

행렬-행렬 연산

- 행렬 간 연산에서 스칼라 연산(일반 연산)과 다른 점은 차원이 개입

행렬 간 합(차)

- 두 행렬의 동일 차원 간 합 연산 수행(+ 또는 - 연산자 사용)

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} + \begin{bmatrix} 1 & -1 & 2 \\ 3 & 2 & 4 \\ -6 & 3 & -7 \end{bmatrix} = \begin{bmatrix} 2 & 1 & 5 \\ 7 & 7 & 10 \\ 1 & 11 & 2 \end{bmatrix}$$

```
x <- matrix(1:9, 3, 3, byrow = T)
y <- matrix(c(1, 3, -6, -1, 2, 3, 2, 4, -7), ncol = 3)
x + y
```

```
[,1] [,2] [,3]
[1,]    2     1     5
[2,]    7     7    10
[3,]    1    11     2
```

행렬 곱/나누기(elementwise product/division)

- 연산자 * 또는 / 사용

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} * \begin{bmatrix} 1 & -1 & 2 \\ 3 & 2 & 4 \\ -6 & 3 & -7 \end{bmatrix} = \begin{bmatrix} 1 & -2 & 6 \\ 12 & 10 & 24 \\ -42 & 24 & -63 \end{bmatrix}$$

```
x * y
```

```
[,1] [,2] [,3]  
[1,]    1   -2    6  
[2,]   12   10   24  
[3,]  -42   24  -63
```

- 행렬-행렬 합(차) 또는 곱(나누기) 연산 시 행렬의 열단위 원소가 채사 용되지 않음

동일 차원 간 연산만 가능!!

```
z <- y[, 1:2] # y 행렬에서 1-2 번째 열 추출  
z # 3 by 2 행렬
```

```
[,1] [,2]  
[1,]    1   -1  
[2,]    3    2  
[3,]   -6    3
```

```
x + z
```

Error in x + z: 배열의 크기가 올바르지 않습니다

```
x * z
```

Error in x * z: 배열의 크기가 올바르지 않습니다

```
x / z
```

Error in x/z: 배열의 크기가 올바르지 않습니다

행렬 간 곱(matrix product)

- 두 행렬 $\mathbf{X}_{n \times m}$, $\mathbf{Y}_{m \times k}$ 이 주어졌을 때 두 행렬의 곱(matrix product) $\mathbf{Z} = \mathbf{X} \cdot \mathbf{Y}$ 는 $n \times k$ 행렬이고 \mathbf{Z} 원소 z_{ij} ($i = 1, \dots, n$, $j = 1, \dots, k$) 아래와 같이 정의됨

$$z_{ij} = \sum_{r=1}^m x_{ir}y_{rj}, \quad \forall \{i, j\}$$

- R에서 위와 같은 연산은 %*%를 사용

- 예시: 행렬 $\mathbf{X}_{2 \times 4}$, $\mathbf{Y}_{4 \times 3}$ 이 아래와 같이 주어졌을 때 두 행렬의 곱 $\mathbf{Z}_{2 \times 3} = \mathbf{X}_{2 \times 4}\mathbf{Y}_{4 \times 3}$ 은 아래와 같음

$$\mathbf{X} = \begin{bmatrix} 1 & 1 & -1 & 1 \\ 1 & -1 & 1 & 1 \end{bmatrix}, \quad \mathbf{Y} = \begin{bmatrix} 1 & -2 & -1 \\ 1 & 1 & 2 \\ 1 & 3 & 1 \\ 1 & 2 & 2 \end{bmatrix}$$

$$\mathbf{Z} = \mathbf{XY} = \begin{bmatrix} 1 & 1 & -1 & 1 \\ 1 & -1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & -2 & -1 \\ 1 & 1 & 2 \\ 1 & 3 & 1 \\ 1 & 2 & 2 \end{bmatrix} = \begin{bmatrix} 2 & -2 & 2 \\ 2 & 2 & 0 \end{bmatrix}$$

```
X <- matrix(c(1,1,1,-1,-1,1,1,1), nrow = 2, ncol = 4)
Y <- matrix(c(1,1,1,1, -2, 1, 3, 2, -1, 2, 1, 2), nrow = 4, ncol = 3)
Z <- X %*% Y
Z
```

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

행렬-벡터 연산

- 행렬 \mathbf{X} 의 행 길이와 벡터 \mathbf{y} 의 길이가 같은 경우 $\rightarrow \mathbf{y}$ 를 열 단위로
제사용

$$\mathbf{X} = \begin{bmatrix} 1 & 2 & 4 \\ 1 & 3 & 2 \\ 1 & 2 & 1 \end{bmatrix}, \quad \mathbf{y} = [20, 18, 23]^T$$

$$\mathbf{X} + \mathbf{y} = \begin{bmatrix} 1 & 2 & 4 \\ 1 & 3 & 2 \\ 1 & 2 & 1 \end{bmatrix} + \begin{bmatrix} 20 & 20 & 20 \\ 18 & 18 & 18 \\ 23 & 23 & 23 \end{bmatrix} = \begin{bmatrix} 21 & 22 & 24 \\ 19 & 21 & 20 \\ 24 & 25 & 24 \end{bmatrix}$$

```
# 행렬-벡터 합 연산
# X = 3 by 3 행렬; y = 3 by 1 벡터
x <- c(1, 1, 1, 2, 3, 2, 4, 2, 1)
X <- matrix(x, nrow = 3)
y <- c(20, 18, 23) # 재사용

x + y
```

```
[,1] [,2] [,3]
[1,] 21 22 24
[2,] 19 21 20
[3,] 24 25 24
```

- 행렬 \mathbf{X} 의 길이와 벡터 y 의 길이가 같은 경우 \rightarrow 벡터 y 를 자동으로 원소를 행렬(열단위)로 변환

$$\mathbf{X} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}, \quad \mathbf{y} = [1, 2, \dots, 9]^T$$

$$\mathbf{X} + \mathbf{y} = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix} + \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix} = \begin{bmatrix} 2 & 8 & 14 \\ 4 & 10 & 16 \\ 6 & 12 & 18 \end{bmatrix}$$

```
# 행렬-벡터 합 연산
# 행렬 x의 길이와 벡터 y의 길이가 같은 경우
x <- c(1:9); X <- matrix(x, nrow = 3)
length(X); y <- x
```

```
[1] 9
```

```
X + y
```

```
[,1] [,2] [,3]  
[1,]    2     8    14  
[2,]    4    10    16  
[3,]    6    12    18
```

```
# 길이가 다른 경우  
# 1) 행렬 길이보다 큰 경우  
y <- c(1:10)  
X + y
```

Warning in X + y: 두 객체의 길이가 서로 배수관계에 있지 않습니다

Error in eval(expr, envir, enclos): dims [product 9]가 객체 [10]의 길이
와 일치하지 않습니다

```
# 1) 행렬 길이의 약수가 아닌 경우  
# y 재사용  
y <- c(1:4)  
X + y
```

Warning in X + y: 두 객체의 길이가 서로 배수관계에 있지 않습니다

```
[,1] [,2] [,3]  
[1,]    2     8    10  
[2,]    4     6    12  
[3,]    6     8    10
```

- 행렬-벡터 `%*%` 적용 시 벡터는 $n \times 1$ 행렬로 간주하고 행렬 곱 연산 수행(단 X와 벡터 y의 길이는 같아야 함).

$$\mathbf{X}_{4 \times 3} = \begin{bmatrix} 1 & 2 & 1 \\ 1 & 1 & 1 \\ 1 & 3 & 3 \\ 1 & 4 & 4 \end{bmatrix}, \quad \mathbf{y}_{3 \times 1} = [7, 6, 8]^T$$

$$\mathbf{X}\mathbf{y} = \begin{bmatrix} 1 & 2 & 1 \\ 1 & 1 & 1 \\ 1 & 3 & 3 \\ 1 & 4 & 4 \end{bmatrix} \cdot \begin{bmatrix} 7 \\ 6 \\ 8 \end{bmatrix} = \begin{bmatrix} 27 \\ 21 \\ 49 \\ 63 \end{bmatrix}$$

```
x <- c(1, 1, 1, 1, 2, 1, 3, 4, 1, 1, 3, 4)
y <- c(7, 6, 8)
X <- matrix(x, nrow = 4, ncol = 3)
X %*% y
```

```
[,1]
[1,] 27
[2,] 21
[3,] 49
[4,] 63
```

행렬의 전치(transpose)

- 전치 행렬(transpose matrix)는 임의의 행렬의 행과 열을 서로 맞바꾼 행렬임

- 행렬 \mathbf{X} 의 전치 행렬은 \mathbf{X}^T 또는 \mathbf{X}' 으로 나타냄
- 행렬 \mathbf{X} 가 다음과 같이 주어졌을 때 전치 행렬 결과

$$\mathbf{X} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \quad \mathbf{X}^T = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

- R에서 행렬을 전치시키는 함수는 `t()` 입

```
# t(object_name): 전치행렬 반환
x <- 1:6
X <- matrix(x, nrow = 2, ncol = 3, byrow = T)
t(X)
```

```
[,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

```
# 전치행렬과 행렬 간 곱
x <- c(1, 1, 1, 1, 1, 22.3, 23.2, 21.5, 25.3, 28.0)
X <- matrix(x, nrow = 5)
t(X) %*% X
```

```
[,1]     [,2]
[1,] 5.0 120.30
[2,] 120.3 2921.87
```

- 벡터-벡터 곱 연산(%*% 사용)

$$\mathbf{x} = [1, 2, 3, 4]^T$$

$$\mathbf{x}\mathbf{x}^T = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 6 & 8 \\ 3 & 6 & 9 & 12 \\ 4 & 8 & 12 & 16 \end{bmatrix}$$

$$\mathbf{x}^T\mathbf{x} = \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} = 1 + 4 + 9 + 16 = 30$$

```
x <- 1:4
x %*% t(x) # 행렬 반환
```

```
[,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    2    4    6    8
[3,]    3    6    9   12
[4,]    4    8   12   16
```

```
t(x) %*% x # 스칼라 반환 x %*% x와 동일 결과 출력
```

```
[,1]
[1,] 30
```

 참고: 전치행렬의 성질(통계수학 II 강의내용 참고)

- $(\mathbf{X}^T)^T = \mathbf{X}$
- $(\mathbf{X} + \mathbf{Y})^T = \mathbf{X}^T + \mathbf{Y}^T$
- $(\mathbf{XY})^T = \mathbf{Y}^T \mathbf{X}^T$
- $(c\mathbf{X})^T = c\mathbf{X}^T$, c 는 임의의 상수

역행렬(inverse matrix)

- 행렬의 나눗셈 형태
- 행렬 \mathbf{X} 가 $n \times n$ 정방행렬(square matrix)일 때, 아래를 만족하는 행렬 $\mathbf{Y}_{n \times n}$ 가 존재하면 \mathbf{Y} 를 \mathbf{X} 의 역행렬(inverse matrix)라고 하고 \mathbf{X}^{-1} 로 나타냄.

$$\mathbf{XX}^{-1} = \mathbf{X}^{-1}\mathbf{X} = \mathbf{I}_{n \times n}$$

- 여기서 $\mathbf{I}_{n \times n}$ 은 대각 원소가 1이고 나머지 원소는 0인 항등 행렬임
- 2×2 행렬의 역행렬은 아래와 같이 구함(3×3 이상 역행렬 구하는 방법은 통계수학 II 강의 참고)

$$\mathbf{X} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}, \quad \mathbf{X}^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

- R에서 정방 행렬의 역행렬은 `solve()` 함수를 사용해 구함

```
# 2 by 2 행렬의 역행렬
x <- c(1, 2, 3, 4)
X <- matrix(x, 2)
solve(X)
```

```
[,1] [,2]
[1,] -2 1.5
[2,] 1 -0.5
```

```
# 항등 행렬이 나오는지 확인
X %*% solve(X)
```

```
[,1] [,2]
[1,] 1 0
[2,] 0 1
```

 참고: 역행렬의 성질(통계수학 II 강의내용 참고)

- $(\mathbf{X}^{-1})^{-1} = \mathbf{X}$
- $(\mathbf{X}^T)^{-1} = (\mathbf{X}^{-1})^T$
- $(\mathbf{XY})^{-1} = \mathbf{Y}^{-1}\mathbf{X}^{-1}$

행렬식(determinant)

- 행렬의 성질을 대표할 수 있는 하나의 값으로 $n \times n$ 정방행렬(square matrix)에서 정의
- 역행렬을 구할 때 임의의 행렬이 0, 즉 위 2×2 행렬에서 $ad - bc$ 의 값이 0이라면 역행렬이 존재할 수 없는데 여기서 $ad - bc \neq 2 \times 2$ 행렬의 정방행렬임

- 임의의 정방행렬 \mathbf{X} 의 행렬식은 $|\mathbf{X}|$ 또는 $\det(\mathbf{X})$ 로 표시함
- 2×2 행렬의 행렬식은 넓이, 3×3 이상인 정방 행렬에서는 부피의 개념으로 이해할 수 있음
- 정방행렬 $\mathbf{X}_{n \times n} = \{x_{ij}\}$ 가 주어졌을 때, i 번째 행과 j 번째 열을 제외한 나머지 $(n-1) \times (n-1)$ 정방행렬의 행렬식을 $|\mathbf{X}_{ij}|$ 라고 하면 이를 x_{ij} 의 소행렬식(minor)이라 부르고 x_{ij} 의 여인수(co-factor) C_{ij} 는 아래와 같이 정의됨

$$c_{ij} = (-1)^{i+j} |\mathbf{X}_{ij}|$$

- 이때 $\mathbf{X}_{n \times n}$ 행렬식은 임의의 i 또는 j 에 대해 아래의 식을 통해 구할 수 있음

$$\det(\mathbf{X}) = \sum_{i=1}^n x_{ij} c_{ij} = \sum_{j=1}^n x_{ij} c_{ij}$$

- 행렬식 계산 예시

$$\mathbf{X} = \begin{bmatrix} 1 & 5 & 0 \\ 2 & 4 & -1 \\ 0 & -2 & 0 \end{bmatrix}$$

$$\det(\mathbf{X}) = x_{11} \det(\mathbf{X}_{11}) - x_{12} \det(\mathbf{X}_{12}) + x_{13} \det(\mathbf{X}_{13})$$

$$= 1 \begin{vmatrix} 4 & -1 \\ -2 & 0 \end{vmatrix} - 5 \begin{vmatrix} 2 & -1 \\ 0 & 0 \end{vmatrix} + 0 \begin{vmatrix} 2 & 4 \\ 0 & -2 \end{vmatrix} = -2$$

- R에서 임의 행렬의 행렬식은 `det()` 함수를 이용해 구함

```
X <- matrix(c(1, 2, 0, 5, 4, -2, 0, -1, 0), ncol = 3)
det(X)
```

[1] -2

 참고: 행렬식의 성질(통계수학 II 강의내용 참고)

- 행렬 \mathbf{X}, \mathbf{Y} 가 정방행렬이면 $\det(\mathbf{XY}) = \det(\mathbf{X})\det(\mathbf{Y})$
- $\det(\mathbf{X}) = \det(\mathbf{X}^T)$
- $\det(c\mathbf{X}) = c^n \det(\mathbf{X})$ 여기서 c 는 임의의 상수
- $\det(\mathbf{X}^{-1}) = \det(\mathbf{X})^{-1}$

그외 정칙(non-singular), 비정칙(non-singular), 양정치(positive definite) 행렬 모두 행렬식으로 정의할 수 있고 자세한 내용은 통계수학 II를 통해 학습. 추가적으로 여인수 c_{ij} 를 이용한 역행렬 공식은 아래와 같음

$$\mathbf{X}^{-1} = \frac{1}{\det(\mathbf{X})} \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix}$$



예습: 3×3 정방행렬 \mathbf{X} 가 아래와 같이 주어졌을 때, \mathbf{X} 의 행렬식과 역행렬 \mathbf{X}^{-1} 을 직접 계산해 보고, R에서 각각을 구하는 함수를 사용하여 계산 결과가 맞는지 확인

$$\mathbf{X} = \begin{bmatrix} 6 & 1 & 4 \\ 2 & 5 & 3 \\ 1 & 1 & 2 \end{bmatrix}$$

2.5.2 행렬의 색인

- R의 행렬 객체 내 데이터 접근은 벡터와 유사하게 행과 열에 대응하는 색인 또는 이름으로 접근 가능
- 행렬의 행과 열은 꺽쇠 ‘[]’ 안에서 ,(콤마)로 구분
- X[idx_row, idx_col]: 행렬 X의 idx_row 행, idx_col행에 저장된 값 반환(색인번호는 1부터 시작)
- idx_row, idx_col을 지정하지 않으면 전체 행 또는 열을 선택

```
x <- 1:12  
X <- matrix(x, ncol = 4)  
X
```

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

```
# 1행만 선택  
X[1, ]
```

```
[1] 1 4 7 10
```

```
# 3열만 선택  
X[, 3]
```

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

```
# 1:3행만 선택
X[1:3, ]
```

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

```
# 1-2행, 3-4열 선택
X[1:2, 3:4]
```

```
[,1] [,2]
[1,]    7   10
[2,]    8   11
```

- 행렬의 각 행과 열에 이름 부여 가능 → `matrix()` 함수 인수 중 `dimnames`에 속성 부여와 동일
- `dimnames()` 함수를 통해 각 행과 열의 이름 확인 및 부여 가능
- `dimnames(object)[[i]]`, $i = 1, 2$ 를 통해 행($i = 1$)과 열($i = 2$) 이름 변경 및 부여 가능
- 위와 유사한 기능을 하는 함수
 - `rownames()`: 행 이름 반환 및 부여
 - `colnames()`: 열 이름 반환 및 부여

```
# matrix 함수 내에서 행렬 이름 동시 부여
X <- matrix(1:9, ncol = 3,
```

```
dimnames = list(c("1", "2", "3"), # 행 이름
                c("A", "B", "C")) # 열 이름
X
```

```
A B C
1 1 4 7
2 2 5 8
3 3 6 9
```

```
# dimnames()를 이용한 이름 확인
dimnames(X) # 행렬에 대한 리스트 반환
```

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

```
[[2]]
[1] "A" "B" "C"
```

```
# dimnames() 함수로 행 이름 변경
dimnames(X)[[1]] <- c("r1", "r2", "r3")

# dimnames() 함수로 열 이름 변경
dimnames(X)[[2]] <- c("c1", "c2", "c3")
dimnames(X)
```

```
[[1]]
[1] "r1" "r2" "r3"
```

```
[[2]]
[1] "c1" "c2" "c3"
```

```
X
```

```
c1 c2 c3  
r1 1 4 7  
r2 2 5 8  
r3 3 6 9
```

```
# rownames()를 통해 행 이름 확인  
rownames(X)
```

```
[1] "r1" "r2" "r3"
```

```
# colnames()를 통해 열 이름 확인  
colnames(X)
```

```
[1] "c1" "c2" "c3"
```

```
# rownames()를 이용해 행 이름 변경  
rownames(X) <- c("apple", "strawberry", "orange")  
rownames(X)
```

```
[1] "apple"      "strawberry" "orange"
```

```
# colnames()를 이용해 행 이름 변경  
colnames(X) <- c("costco", "emart", "homeplus")  
colnames(X)
```

```
[1] "costco"   "emart"    "homeplus"
```

```
X
```

| | costco | emart | homeplus |
|------------|--------|-------|----------|
| apple | 1 | 4 | 7 |
| strawberry | 2 | 5 | 8 |
| orange | 3 | 6 | 9 |

- 행과 열에 대한 이름이 존재한다면 벡터와 마찬가지로 이름으로 색인 가능

```
X[c("apple", "orange"), c("emart")]
```

| | apple | orange |
|--|-------|--------|
| | 4 | 6 |

```
# 2번째 열에 해당(emart)를 제외한 나머지 열 반환
X[, colnames(X)[-2]]
```

| | costco | homeplus |
|------------|--------|----------|
| apple | 1 | 7 |
| strawberry | 2 | 8 |
| orange | 3 | 9 |

- 색인한 행렬 원소에 다른 값 할당

```
y <- c(1:12); Y <- matrix(y, ncol = 3)
Y
```

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

```
# 2, 4 행과 2-3열에 다른 값 할당
Y[c(2, 4), 2:3] <- matrix(c(1, 2, 1, 4), ncol = 2)

# 행렬 값 할당 다른 예시
X <- matrix(nrow = 4, ncol = 3) # NA 값으로 구성된 4 by 3 행렬
X
```

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

```
y <- c(1, 0, 0, 1); Y <- matrix(y, ncol = 2)
X[3:4, 2:3] <- Y
X
```

```
[,1] [,2] [,3]
[1,]    NA    NA    NA
[2,]    NA    NA    NA
[3,]    NA     1     0
[4,]    NA     0     1
```

- 행렬 필터링 → 색인 대신 조건 사용(벡터와 동일)

```
X = matrix(c(1,2,4,3,2,3,5,6), nrow = 4, ncol = 2)
```

```
# X의 1열이 3보다 작거나 같은 행 필터링
```

```
X[X[,1] <= 3, ]
```

```
[,1] [,2]
```

```
[1,] 1 2
```

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

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

```
# 논리값을 활용한 필터링
```

```
idx <- X[, 1] <= 3; idx
```

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

```
X[idx, ]
```

```
[,1] [,2]
```

```
[1,] 1 2
```

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

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

2.5.3 행과 열 추가 및 제거

- 행렬 재할당(re-assignment)를 통해 열이나 행을 직접 추가하거나 삭제 가능

- `cbind()` (열 붙이기), `column bind`, `rbind()` (행 붙이기), `row bind`

함수 사용

```
j <- rep(1, 4)
Z <- matrix(c(1:4, 1, 1, 0, 0, 1, 0, 1, 0), nrow = 4, ncol = 3)
Z
```

```
[,1] [,2] [,3]
[1,]    1    1    1
[2,]    2    1    0
[3,]    3    0    1
[4,]    4    0    0
```

```
cbind(j, Z) # 열 기준으로 붙이기
```

```
j
[1,] 1 1 1 1
[2,] 1 2 1 0
[3,] 1 3 0 1
[4,] 1 4 0 0
```

```
# 길이가 다른 경우 재사용
cbind(1, Z)
```

```
[,1] [,2] [,3] [,4]
[1,]    1    1    1    1
[2,]    1    2    1    0
[3,]    1    3    0    1
[4,]    1    4    0    0
```

```
# Z 행렬 앞에 j 열 붙여서 새로운 z 생성  
Z <- cbind(j, Z)  
  
# 행 기준으로 붙이기  
Z <- rbind(Z, 2)
```

- 행 또는 열의 제거는 벡터에서와 마찬가지로 색인 앞에 - 사용

```
# 첫 번째 행 제거  
Z[-1, ]
```

```
j  
[1,] 1 2 1 0  
[2,] 1 3 0 1  
[3,] 1 4 0 0  
[4,] 2 2 2 2
```

```
# 1, 5행 , 3열 제거  
Z[-c(1, 5), -3]
```

```
j  
[1,] 1 2 0  
[2,] 1 3 1  
[3,] 1 4 0
```



cbind() 또는 rbind() 함수는 다음 주에 배울 데이터 프레임에도 적용 가능하다.

2.5.4 행렬 관련 함수

- `diag()`: 대각행렬 생성 또는 대각원소(diagonal elements) 추출
- 대각행렬: 주 대각선을 제외한 모든 원소가 0인 $n \times n$ 정방행렬로 다음과 같이 정의

$$\mathbf{D} = \{d_{ij}\}, \quad i, j \in \{1, 2, \dots, n\}, \quad \forall i \neq j \rightarrow d_{ij} = 0$$

```
D <- diag(c(1:5), 5)
D
```

```
[,1] [,2] [,3] [,4] [,5]
[1,]    1    0    0    0    0
[2,]    0    2    0    0    0
[3,]    0    0    3    0    0
[4,]    0    0    0    4    0
[5,]    0    0    0    0    5
```

```
# 3차원 항등 행렬(모든 대각원소가 1인 행렬)
I3 <- diag(1, 3)

# 대각원소 추출
diag(D)
```

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

```
# 대각원소 재할당
diag(D) <- rep(1, 5)
```



객체는 속성(attribute)을 갖고 그 속성에 따라 데이터의 구조가 정해짐. 즉 속성은 데이터에 대한 메타 데이터임. 객체의 속성은 대표적으로 이름(names), 차원(dimension), 클래스(class)로 정의되고 객체에 대한 자세한 정보를 파악하기 위해 제공되는 몇 가지 함수들에 대해 알아봄.

R은 앞서 언급한 바와 같이 객체지향언어(object oriented program, OOP)이고 세 가지 유형의 객체지향 시스템(S3, S4, S5)이 존재함. R의 핵심적인 함수 및 패키지는 S3 객체 시스템을 사용하고 있기 때문에 알아둘 필요가 있으나 본 강의의 범위를 벗어나기 때문에 이번 학기에는 다루지 않을 것임.

- `dim(object_name)`: 행렬 또는 데이터 프레임의 행과 열의 개수(차원)를 반환

```
# dim(): 객체의 차원(dimension)을 반환
Z
```

```
j
[1,] 1 1 1 1
[2,] 1 2 1 0
[3,] 1 3 0 1
[4,] 1 4 0 0
[5,] 2 2 2 2
```

```
dim(Z)
```

```
[1] 5 4
```

- `nrow()` 또는 `NROW()`: 행렬의 행 길이 반환
- `ncol()` 또는 `NCOL()`: 행렬의 행 길이 반환

```
nrow(Z); ncol(Z)
```

```
[1] 5
```

```
[1] 4
```

`nrow()`/`ncol()`과 `NROW()`/`NCOL()`의 차이점

- `nrow()`/`ncol()`은 행렬 또는 데이터 프레임에 적용되며 벡터가 인수로 사용될 때 `NULL` 값을 반환하는데 비해 `NROW()`/`NCOL()`은 벡터의 길이도 반환 가능

- `attributes()`: 객체가 갖는 속성을 반환함

```
x <- 1:9; X <- matrix(x, ncol = 3)
# 객체의 속성 확인
attributes(x)
```

```
NULL
```

```
attributes(X)
```

```
$dim
```

```
[1] 3 3
```

- `class()`: 객체의 클래스 명칭 반환 및 클래스 부여

```
# 객체의 class 확인
```

```
class(x); class(X)
```

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

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

```
# 객체의 class 부여
```

```
class(x) <- "this is a vector"
```

- `str()`: 객체가 갖고 있는 데이터의 구조 확인

```
# 객체의 구조 파악
```

```
str(x); str(X)
```

```
'this is a vector' int [1:9] 1 2 3 4 5 6 7 8 9
```

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

```
# x가 X의 이름(name) 속성을 추가한 경우
names(x) <- paste0("x", 1:9)
dimnames(X) <- list(paste0("r", 1:3),
                      paste0("c", 1:3))
attributes(x); attributes(X)
```

```
$class
[1] "this is a vector"
```

```
$names
[1] "x1" "x2" "x3" "x4" "x5" "x6" "x7" "x8" "x9"
```

```
$dim
[1] 3 3
```

```
$dimnames
$dimnames[[1]]
[1] "r1" "r2" "r3"
```

```
$dimnames[[2]]
[1] "c1" "c2" "c3"
```

```
class(x); class(X)
```

```
[1] "this is a vector"
```

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

```
str(x); str(X)
```

```
'this is a vector' Named int [1:9] 1 2 3 4 5 6 7 8 9
- attr(*, "names")= chr [1:9] "x1" "x2" "x3" "x4" ...
int [1:3, 1:3] 1 2 3 4 5 6 7 8 9
- attr(*, "dimnames")=List of 2
..$ : chr [1:3] "r1" "r2" "r3"
..$ : chr [1:3] "c1" "c2" "c3"
```

- `attr(object, "attribute_name")`: 객체가 갖고 있는 속성을 지정 해서 확인

```
# 객체 속성 요소 확인
attr(x, "names")
```

```
[1] "x1" "x2" "x3" "x4" "x5" "x6" "x7" "x8" "x9"
```

```
attr(X, "dimnames")
```

```
[[1]]
[1] "r1" "r2" "r3"
```

```
[[2]]
[1] "c1" "c2" "c3"
```

2.5.5 벡터와 행렬의 차이점

- 행렬은 개념적으로 $n \times 1$ 벡터가 2 개 이상 둉어져서 행과 열의 속성을 갖지만 기본적으로는 벡터

```
z <- 1:8  
U <- matrix(z, 4, 2)  
length(z) # 입력 벡터 원소의 길이가 8
```

```
[1] 8
```

- R에서 U가 행렬임을 나타내기 위해 추가적인 속성(attribute)를 부여

```
class(z) # 벡터
```

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

```
attributes(z)
```

```
NULL
```

```
class(U) # 행렬
```

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

```
attributes(U)
```

```
$dim
```

```
[1] 4 2
```

2.5.6 의도치 않은 차원축소 피하기

- 다음 행렬에서 한 행을 추출

```
Z <- matrix(c(1:8), 4, 2)
z <- Z[2, ]

attributes(Z) # 행과 열의 차원 수를 표시
```

```
$dim
```

```
[1] 4 2
```

```
# 객체 z의 속성 및 형태는?
attributes(z) # 차원이 존재하지 않음
```

```
NULL
```

- 차원축소를 방지하는 방법 → r을 벡터가 아닌 1×2 행렬로 인식

```
z <- Z[2, , drop = FALSE]
attributes(z)
```

```
$dim
[1] 1 2
```

- `as.matrix()`를 이용한 직접 변환

```
z <- as.matrix(Z[2, ])
class(z)
```

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

```
z # 행렬이 변환됨을 유의
```

```
[,1]
[1,]    2
[2,]    6
```

2.6 배열(array)

- 통계학의 관점에서 R의 행렬의 행은 조사 대상이 되는 사람, 동물 등 관측 대상에 해당하고, 열은 대상의 특성을 표현하는 변수(예: 몸무게, 키, 혈압 등)에 해당 → 2차원 구조
- 위와 같은 데이터를 년 단위로 수집한다면? → 한 대상자에 해당하는 변수들은 시간에 따라 변함 → 시간 차원이 하나 더 존재!
- R에서 이러한 형태의 데이터 구조를 배열(array)이라고 지칭함

2.6.1 배열의 생성 및 색인

- 동일한 유형의 데이터가 2차원 이상으로 구성된 데이터 구조
- 동일한 차원($n \times p$)의 배열(행렬)이 k 개 방에 저장된 데이터 구조
- 배열 생성 함수

```
# array() 함수 인수 구조
array(data, # 저장할 데이터 벡터 또는 행렬
      dim, # 배열의 차원 지정
      dimnames # 배열 차원 명칭
      )
```

- 통계학과 3명의 학생에 대한 중간고사 기준 한 번의 퀴즈와 중간고사 점수, 그리고 기말고사 기준 한 번의 퀴즈와 기말고사 점수 데이터 가정

```
x <- c(75, 84, 93, 65, 78, 92)
y <- c(82, 78, 85, 88, 75, 88)

first_term <- matrix(x, nrow = 3, ncol = 2)
second_term <- matrix(y, nrow = 3, ncol = 2)

first_term
```

```
[,1] [,2]
[1,] 75   65
[2,] 84   78
[3,] 93   92
```

```
second_term
```

```
[,1] [,2]
[1,] 82 88
[2,] 78 75
[3,] 85 88
```

```
# 위 두 데이터를 2층 짜리 배열로 구성
Z <- array(data = c(first_term, second_term),
            dim = c(3, 2, 2))
Z
```

```
, , 1
```

```
[,1] [,2]
[1,] 75 65
[2,] 84 78
[3,] 93 92
```

```
, , 2
```

```
[,1] [,2]
[1,] 82 88
[2,] 78 75
[3,] 85 88
```

```
# Z의 속성
attributes(Z)
```

```
$dim  
[1] 3 2 2
```

```
# Z의 클래스  
class(Z)
```

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

```
# Z의 구조  
str(Z)
```

```
num [1:3, 1:2, 1:2] 75 84 93 65 78 92 82 78 85 88 ...
```

- 배열 내 데이터 접근은 색인을 통해 가능(벡터 행렬과 동일)

```
# 첫 번째 층만 추출  
Z[, , 1]
```

```
[,1] [,2]  
[1,] 75 65  
[2,] 84 78  
[3,] 93 92
```

```
# 두 번째 층에서 2-3행 만 추출  
Z[2:3, , 2]
```

```
[,1] [,2]  
[1,] 78 75  
[2,] 85 88
```

2.6.2 배열의 확장 예제

데이터 사이언스 스쿨³ 참고

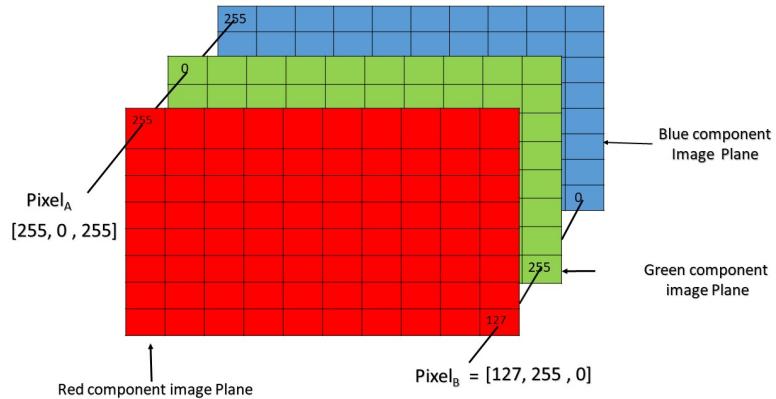
- 배열 구조를 갖는 가장 대표적인 데이터 중 하나가 이미지(사진)
- 이미지 데이터는 픽셀(pixel)이라는 세분화된 작은 이미지를 직사각형 형태로 모은 형태
- 전체 이미지는 세로픽셀수 × 가로픽셀수로 표현됨 → 행렬
- 픽셀의 색을 숫자로 표현하는 방식을 색공간(color space)라고 명칭
- 대표적 색공간은 흑백스케일(grey scale), RGB (Red-Green-Blue), HSV(Hue-Saturation-Value) 방식
- RGB 색공간을 사용한 경우 각 색공간별로 동일한 크기의 행렬이 3개 층으로 저장된 상태 → 배열
- RGB는 0 ~ 255 까지 값을 갖고 빨강색 (255, 0, 0), 녹색 (0, 255, 0), 파란색은 (0, 0, 255)임

목표

- R에서 웹 url로 이미지를 불러오기
- 불러온 이미지를 R에서 plotting 해보기
- 이미지 데이터를 직접 수정 해보기

1. 이미지 입출력 패키지 installation

```
install.packages("jpeg") # jpeg 파일 입출력 관련 package
install.packages("cowplot") # ggplot add-on package
```



Pixel of an RGB image are formed from the corresponding pixel of the three component images

FIGURE 2.2: <https://www.geeksforgeeks.org/matlab-rgb-image-representation/>에서 발췌

2. 관련 패키지 불러오기

```
require(tidyverse)
require(jpeg)
require(cowplot)
```

3. 이미지 불러오기

```
myurl <- "https://img.livescore.co.kr/data/editor/1906/ba517de8162d92f4ea0e9de0ec98ba02.jpg"
z <- tempfile()
download.file(myurl,z,mode="wb")
pic <- readJPEG(z)
```

4. 이미지 그래프 출력창에서 확인



5. 이미지 임의 부분 편집하기

```
pic[300:460, 440:520, 1] <- 0.5  
pic[300:460, 440:520, 2] <- 0.5  
pic[300:460, 440:520, 3] <- 0.5  
  
ggdraw() +  
  draw_image(pic)
```



6. RGB값을 무작위로 샘플링 후 매개변수로 노이즈 가중치 조절해 보기

```
pic <- readJPEG(z)
yr <- pic[300:460, 440:520, 1]
yg <- pic[300:460, 440:520, 2]
yb <- pic[300:460, 440:520, 3]
n <- nrow(yr); p <- ncol(yr)

t <- 0.2
wr <- t * yr + (1 - t)*matrix(runif(length(yr)), nrow = n, ncol = p)
wg <- t * yg + (1 - t)*matrix(runif(length(yg)), nrow = n, ncol = p)
```

```
wb <- t * yb + (1 - t)*matrix(runif(length(yb)), nrow = n, ncol = p)

pic[300:460, 440:520, 1] <- wr
pic[300:460, 440:520, 2] <- wg
pic[300:460, 440:520, 3] <- wb

ggdraw() +
  draw_image(pic)
```



2.7 요인(factor)과 테이블(table)

- 요인(factor) 데이터 타입은 통계학에서 범주형 변수(categorical variable)을 표현하기 위한 R의 데이터 타입으로 범주형 자료는 크게 명목형(nominal)과 순서형(ordinal)으로 구분
- 테이블(table) 객체는 factor 객체에 대한 빈도를 나타내기 위해 사용

범주형 자료

- 데이터가 사전에 정해진 특정 유형으로만 분류되는 경우: 성별, 인종, 혈액형 등
- 범주형 자료는 명목형과 순서형으로 구분 가능
- 순서형 자료 예: 성적, 교육수준, 선호도, 중증도 등

2.7.1 요인(factor)

- 범주형 자료를 표현하기 위한 R의 객체 클래스
- Factor는 정수형 벡터를 기반으로 levels (수준)이라는 속성이 추가된 객체임
- 숫자 또는 문자로 표현 되었다 하더라도 범주형으로 이해
- Factor는 level에 해당하는 값만 가질 수 있는 벡터로 간주
- Factor 생성 함수

```
# factor 정의 함수  
factor(data, # factor로 표현하고자 하는 값. 주로 문자형  
       levels, # 요인의 수준, 미리 정한 값)
```

```

  labels, # 수준에 대한 레이블링
  ordered # 순서형 자료 표시 여부
    # TRUE/FALSE, default = FALSE
)

```

- 수치형을 factor로 만들어도 처음 입력 값은 문자형으로 변하고 level 값으로 치환
- 대신 (1, 2, 3)이 중심값이 됨 → 정수형 벡터임

```

score <- rep(c(4:6), each = 4)
fscore <- factor(score)

typeof(fscore) # factor의 기본 데이터 타입

```

[1] "integer"

```
attributes(fscore) # factor의 속성
```

```
$levels
[1] "4" "5" "6"
```

```
$class
[1] "factor"
```

```
# factor의 구조
str(fscore)
```

```
Factor w/ 3 levels "4","5","6": 1 1 1 1 2 2 2 2 3 3 ...
```

```
# levels(): factor의 수준(levels) 반환 함수  
levels(fscore)
```

```
[1] "4" "5" "6"
```

```
# nlevels(): level의 개수 반환  
nlevels(fscore)
```

```
[1] 3
```

- Factor를 벡터 결합 함수 c()로 결합

```
c(fscore, factor(4)) # 강제로 정수형 벡터로 변환
```

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

- Factor의 범주 수준(level) 및 범주명(label) 지정

```
x <- rep(c(1:2), each = 4)  
  
# factor의 범주 수준 지정  
sex <- factor(x, levels = 1:2)  
sex
```

```
[1] 1 1 1 1 2 2 2 2
Levels: 1 2
```

```
# factor의 범주 수준 및 범주 명칭 지정
sex <- factor(x, levels = 1:2, labels = c("male", "female"))
sex # level의 값이 명칭으로 변경
```

```
[1] male   male   male   male   female female female female
Levels: male female
```

```
str(sex)
```

```
Factor w/ 2 levels "male","female": 1 1 1 1 2 2 2 2
```

```
# 같은 존재하지 않으나 수준을 미리 정해 놓은 경우
severity <- factor(1:2, levels = c(1, 2, 3), labels = c("Mild", "Moderate", "Severe"))
severity[2] <- "Severe"

# 존재하지 않는 수준 할당
severity[1] <- "Good"
```

Warning in `<-.factor`(`*tmp*`, 1, value = "Good"): 요인의 수준
(factor level)이
올바르지 않아 NA가 생성되었습니다.

```
severity
```

```
[1] <NA> Severe  
Levels: Mild Moderate Severe
```

- 순서형 factor 생성

```
severity <- factor(rep(1:3, times = 3), levels = 1:3,  
                     labels = c("Mild", "Moderate", "Severe"),  
                     ordered = T)  
severity
```

```
[1] Mild      Moderate Severe    Mild      Moderate Severe    Mild      Moderate  
[9] Severe  
Levels: Mild < Moderate < Severe
```

```
is.ordered(severity) # 순서형 범주 체크
```

```
[1] TRUE
```

요인형 객체에 적용되는 일반적인 함수

tapply() 함수

- 특정 요인 수준의 고유한 조합으로 각 그룹에 속한 값에 특정 함수를 적용한 결과를 반환
- 일반적인 함수 사용 형태는 아래와 같음

```
# tapply() 함수 사용 인수
tapply(
  x, # 벡터,
  INDEX, # 벡터를 그룹화할 색인(factor)
  FUN, # 각 그룹마다 적용할 함수
)
```

- 예시: 2020년 4월 15일 총선의 연령별 지지율

```
# 문자열을 INDEX의 인수로 받은 경우

x <- c(48, 43, 27, 52, 38,
      67, 23, 58, 72, 85) # 유권자 연령
f <- rep(c("더불어민주당", "미래통합당"), each = 5)
t <- tapply(x, f, mean) # f의 요인 수준 별 x (연령) 평균 계산
t
```

더불어민주당 미래통합당

41.6 61.0

```
# x, f 순서를 랜덤하게 섞은 다음 결과
set.seed(12345) # 난수 생성 결과 고정
idx <- order(runif(10))
x <- x[idx]
f <- f[idx]

tapply(x, f, mean)
```

더불어민주당 미래통합당

41.6 61.0

- Factor가 2개 이상인 경우 두 factor 객체의 수준의 조합(AND 조건)에 따른 그룹을 만든 후 그룹별 함수 적용

```
s <- rep(c("M", "F"), each = 6)
income <- c(35, 42, 68, 29, 85, 55,
          30, 40, 63, 27, 83, 52) * 100 # 단위: 만원
age <- c(32, 36, 44, 25, 55, 41,
        28, 33, 46, 23, 54, 44)

set.seed(12345) # 난수 생성 결과 고정
idx <- order(runif(12))
s <- s[idx]; income <- income[idx]; age <- age[idx]

# age <= 40 -> 1, 40 < age <= 50 -> 2,
# age >= 50 -> 3 할당: ifelse() 함수 사용
age <- ifelse(age <= 40, 1,
              ifelse(age <= 50, 2, 3))

tapply(income, list(sex = s, age = age), mean)
```

| | age | | |
|-----|----------|------|------|
| sex | 1 | 2 | 3 |
| F | 3233.333 | 5750 | 8300 |
| M | 3533.333 | 6150 | 8500 |



R에서 가장 많이 활용되는 함수 계열 중 하나로 *apply()를 들 수 있다. 벡터, 행렬 등과 같은 R 객체에 for loop 대신 반복적으로 동일한 함수를 적용할 때 활용된다. *apply() 계열 함수에 대해서는 데이터 프레임에서 더 상세하게 배울 것임

***split()* 함수**

- *tapply()*는 주어진 요인의 수준에 따라 특정 함수를 적용하지만, *split()*은 데이터를 요인의 수준(그룹) 별로 데이터를 나누어 리스트 형태로 반환

```
# split() 함수 사용 인수
split(
  x, # 분리할 데이터(벡터)
  f, # 데이터를 분리할 기준이 되는 factor 지정
)
```

- *split()* 함수 사용 예시

```
# 성별의 수준 남녀 별 소득 수준 분리
split(income, s)
```

```
$F
[1] 8300 5200 3000 4000 6300 2700
```

```
$M
[1] 5500 8500 3500 6800 4200 2900
```

```
# 두 개 요인 조합으로 income 벡터 분리
split(income, list(s, age))
```

\$F.1

```
[1] 3000 4000 2700
```

\$M.1

```
[1] 3500 4200 2900
```

\$F.2

```
[1] 5200 6300
```

\$M.2

```
[1] 5500 6800
```

\$F.3

```
[1] 8300
```

\$M.3

```
[1] 8500
```

```
# 요인의 각 수준에 대한 인덱스를 반환하고자 하는 경우
```

```
abalone <- read.csv("http://archive.ics.uci.edu/ml/machine-learning-databases/abalone/abalone.data",
header = FALSE) # 전복 데이터셋
# V1: 전복의 종류
# F=암컷; M=수컷, I=새끼
g <- abalone[, 1] # 전복종류만 추출

set.seed(20200410)
idx <- sample(1:length(g), size = 10)
g <- g[idx]
split(1:length(g), g)
```

\$F

```
[1] 1 6 8
```

```
$I
[1] 2 3 5 7

$M
[1] 4 9 10
```

2.7.2 테이블(table)

- 범주형 변수의 빈도 또는 분할표(교차표)를 표현하기 위한 객체(클래스)
- 범주 별 통계량(평균, 표준편차, 중위수, ...) 요약

`tapply()` 함수를 이용한 테이블 만들기

- 길이가 12인 임의의 벡터 `u`를 수준의 개수가 각각 3, 2인 factor의 조합으로 부분벡터로 분리 후 `length()` 적용 → `tapply()` 함수 사용

```
u <- runif(12)
f1 <- factor(c(4, 4, 3, 5, 5, 4,
              3, 3, 4, 5, 5, 3))
f2 <- factor(c("a", "a", "a", "a", "b", "a",
              "b", "b", "a", "a", "b", "b"))
tapply(u, list(f1, f2), length)
```

| | |
|---|----|
| a | b |
| 3 | 1 |
| 4 | 4 |
| 5 | 2 |
| | NA |

- `u`의 값과 상관 없이 두 factor 형 변수 `f1`과 `f2`의 조합에 따른 개수 반환 → 분할표(contingency table)

- 위 예시에서 f1이 “4”이고 f2가 “b”인 경우는 없기 때문에 0 값이 있어야 하나, tapply() 함수 적용 시 결측값 NA를 반환
- table(): 하나 이상의 factor의 수준 또는 수준의 조합으로 분할표 생성
- Factor가 3개 이상인 경우 배열로 다차원 분할표 표현

```
# table() 적용 예시  
t1 <- table(f1, f2)  
t1
```

```
f2  
f1  a  b  
 3 1 3  
 4 4 0  
 5 2 2
```

```
typeof(t1); attributes(t1); str(t1)
```

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

```
$dim  
[1] 3 2
```

```
$dimnames  
$dimnames$f1  
[1] "3" "4" "5"
```

```
$dimnames$f2  
[1] "a" "b"
```

```
$class
[1] "table"

'table' int [1:3, 1:2] 1 4 2 3 0 2
- attr(*, "dimnames")=List of 2
..$ f1: chr [1:3] "3" "4" "5"
..$ f2: chr [1:2] "a" "b"
```

```
# factor가 한개인 경우
table(f1)
```

```
f1
3 4 5
4 4 4
```

```
# factor가 3개인 경우
year = c("1", "1", "2", "3", "3", "4")
gender = c("M", "M", "F", "M", "F", "F")
grade = c("A", "C", "B", "B", "A", "C")

table(gender, grade, year)
```

```
, , year = 1
```

```
grade
gender A B C
F 0 0 0
M 1 0 1
```

```
, , year = 2
```

```
grade  
gender A B C  
F 0 1 0  
M 0 0 0
```

```
, , year = 3
```

```
grade  
gender A B C  
F 1 0 0  
M 0 1 0
```

```
, , year = 4
```

```
grade  
gender A B C  
F 0 0 1  
M 0 0 0
```

테이블 관련 함수

tabulate() 함수

- 정수로 이루어진 벡터에 각 정수 값이 발생한 횟수를 카운팅한 결과를 반환 → `table()` 함수의 핵심 함수

```
# tabulate() 함수 사용 인수(argument)  
tabulate(
```

```
bin, # 정수형(수치형) 벡터 또는 factor
nbins, # 사용할 수준(bin)의 개수
)
```

- `tabulate()` 함수 예시

```
x <- c(2, 2, 2, 1, 3, 4, 5, 5, 10, 8, 8)
tabulate(x)
```

```
[1] 1 3 1 1 2 0 0 2 0 1
```

```
tabulate(x, nbins = 3)
```

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

`addmargins()` 함수

- 테이블 객체(분할표)를 인수로 받아 각 요인의 수준 및 수준 조합 별 합계 값을 테이블과 동시 반환

```
# addmargins() 함수 사용 인수
addmargins(
  T # 테이블 또는 배열 객체
)
```

- `addmargins()` 예시

```
t1 <- table(f1, f2)
addmargins(t1)
```

```
f2
f1      a  b Sum
 3      1  3   4
 4      4  0   4
 5      2  2   4
Sum    7  5  12
```

```
# 3차원 0 이상 테이블
t2 <- table(gender, grade, year)
is.table(t2); is.array(t2)
```

```
[1] TRUE
```

```
[1] TRUE
```

```
addmargins(t2)
```

```
, , year = 1
```

```
grade
gender A B C Sum
  F    0 0 0    0
  M    1 0 1    2
Sum  1 0 1    2
```

```
, , year = 2
```

```
grade
gender A B C Sum
F   0 1 0   1
M   0 0 0   0
Sum 0 1 0   1
```

, , year = 3

```
grade
gender A B C Sum
F   1 0 0   1
M   0 1 0   1
Sum 1 1 0   2
```

, , year = 4

```
grade
gender A B C Sum
F   0 0 1   1
M   0 0 0   0
Sum 0 0 1   1
```

, , year = Sum

```
grade
gender A B C Sum
F   1 1 1   3
M   1 1 1   3
Sum 2 2 2   6
```

ftable() 함수

- “평평한(flat)” 교차표 생성
- 다차원 교차표 작성 시 행변수와 열변수 교환을 통해 재사용 가능

```
ftable(  
  x, # factor, table 또는 ftable 클래스를 갖는 객체  
  row.vars, # 행 변수 지정 색인(정수, 문자)  
  col.vars # 열 변수 지정 색인(정수, 문자)  
)
```

- ***ftable()*** 함수 사용 예시

```
t3 <- ftable(t2)  
t3; attributes(t3); str(t3)
```

```
year 1 2 3 4  
gender grade  
F      A      0 0 1 0  
          B      0 1 0 0  
          C      0 0 0 1  
M      A      1 0 0 0  
          B      0 0 1 0  
          C      1 0 0 0
```

```
$dim  
[1] 6 4
```

```
$class
```

```
[1] "ftable"

$row.vars
$row.vars$gender
[1] "F" "M"

$row.vars$grade
[1] "A" "B" "C"

$col.vars
$col.vars$year
[1] "1" "2" "3" "4"

'ftable' int [1:6, 1:4] 0 0 0 1 0 1 0 1 0 0 ...
- attr(*, "row.vars")=List of 2
..$ gender: chr [1:2] "F" "M"
..$ grade : chr [1:3] "A" "B" "C"
- attr(*, "col.vars")=List of 1
..$ year: chr [1:4] "1" "2" "3" "4"
```

```
# E//0/를 내 행 변수 바꾸기
t4 <- ftable(t2, row.vars = c("year", "gender"))
t4
```

| | | grade | A | B | C |
|------|--------|-------|---|---|---|
| year | gender | | | | |
| 1 | F | | 0 | 0 | 0 |
| | M | | 1 | 0 | 1 |
| 2 | F | | 0 | 1 | 0 |
| | M | | 0 | 0 | 0 |
| 3 | F | | 1 | 0 | 0 |

| | | |
|---|---|-------|
| | M | 0 1 0 |
| 4 | F | 0 0 1 |
| | M | 0 0 0 |

```
# 테이블 내 열 변수 바꾸기  
t5 <- ftable(t2, col.vars = 1)  
t5
```

| | grade | year | gender | F | M |
|---|-------|------|--------|-----|---|
| A | 1 | | | 0 1 | |
| | 2 | | | 0 0 | |
| | 3 | | | 1 0 | |
| | 4 | | | 0 0 | |
| B | 1 | | | 0 0 | |
| | 2 | | | 1 0 | |
| | 3 | | | 0 1 | |
| | 4 | | | 0 0 | |
| C | 1 | | | 0 1 | |
| | 2 | | | 0 0 | |
| | 3 | | | 0 0 | |
| | 4 | | | 1 0 | |

margin.table() 함수

- 배열 형식으로 지정된 교차표(table() 반환 결과)에서 지정된 차원 색인에 대한 표 합계 계산 결과 반환

```
margin.table(  
  x,  # table 또는 ftable 클래스를 갖는 객체  
  margin # 차원 색인 번호  
)
```

- `margin.table()` 예시

```
t2
```

```
, , year = 1
```

```
      grade  
gender A B C  
      F 0 0 0  
      M 1 0 1
```

```
, , year = 2
```

```
      grade  
gender A B C  
      F 0 1 0  
      M 0 0 0
```

```
, , year = 3
```

```
      grade  
gender A B C  
      F 1 0 0  
      M 0 1 0
```

```
, , year = 4
```

```
grade  
gender A B C  
F 0 0 1  
M 0 0 0
```

```
margin.table(t2, 1) # 1 차원(행): 성별
```

```
gender  
F M  
3 3
```

```
margin.table(t2, 2) # 2 차원(열): 성적
```

```
grade  
A B C  
2 2 2
```

```
margin.table(t2, 3) # 3 차원(배열 방 번호): 학년
```

```
year  
1 2 3 4  
2 1 2 1
```

prop.table() 함수

- *table* 객체의 빈도에 대한 비율 계산

- 전체, 차원 단위 비율 계산 가능

```
prop.table(
  x,  # table 또는 ftable 클래스를 갖는 객체
  margin # 차원 색인 번호
)
```

- `prop.table()` 예시

- `margin = NULL`: 각 셀을 전체 cell의 합으로 나눈 비율
- `margin = 1`: 각 행 별 셀에 대해 각 행에 해당하는 cell 합으로 나눈 비율 ($n_{ij}/n_{i\cdot}$), $n_{i\cdot} = \sum_{j=1}^J n_{ij}$
- `margin = 2`: 각 열 별 셀에 대해 각 열에 해당하는 cell 합으로 나눈 비율 ($n_{ij}/n_{\cdot j}$), $n_{\cdot j} = \sum_{i=1}^I n_{ij}$

```
# 2차원 교차표
prop.table(t1) # margin = NULL
```

```
f2
f1          a          b
3 0.08333333 0.25000000
4 0.33333333 0.00000000
5 0.16666667 0.16666667
```

```
prop.table(t1, 1) # margin = 1 (row)
```

```
f2
```

```
f1      a      b  
3 0.25 0.75  
4 1.00 0.00  
5 0.50 0.50
```

```
prop.table(t1, 2) # margin = 2 (column)
```

```
f2  
f1          a          b  
3 0.1428571 0.6000000  
4 0.5714286 0.0000000  
5 0.2857143 0.4000000
```

2.8 데이터 프레임(data frame)



학습목표(4 주차): 데이터 프레임 클래스에 대해 알아보고, 데이터 프레임을 생성, 병합(merge), 연산에 대한 함수들에 대해 알아본다.

- Excel 스프레드시트와 같은 형태
- 데이터 프레임은 데이터 유형에 상관없이 2차원 형태의 데이터 구조
- 행렬과 리스트를 혼합한 자료 형태 → 동일한 길이의 벡터로 이루어진 리스트를 구성요소로 갖는 리스트
- 행렬과 유사한 구조를 갖고 있지만 각기 다른 유형의 자료형태로 자료행렬을 구성할 수 있다는 점에서 행렬과 차이를 갖음

TABLE 2.6: 스프레드시트 기본 형태 예시

| 이름 | 직장 | 나이 |
|-----|------|----|
| 김어준 | 딴지일보 | 51 |
| 주진우 | 시사인 | 46 |
| 김용민 | 프리랜서 | 45 |
| 정봉주 | 정당인 | 59 |

- 행렬과 마찬가지로 변수(열)의 길이(행의 개수)는 모두 동일해야 함
- R에서 가장 빈번하게 활용되고 있는 데이터 클래스임
- 데이터 프레임의 각 열(컬럼)은 벡터로 간주

2.8.1 데이터 프레임 생성

- 데이터 프레임 생성 함수: `data.frame()`

```
data.frame(
  # 값 또는 이름(tag) = 값
  ...,
  # 논리값.
  # 변수명(열 이름)이 구문 상 유효한 변수인지 또는 중복이 있는지 확인
  check.names,
  # 논리값. 문자형 벡터의 factor 형 강제 변환 여부
  stringsAsFactors,
)
```

- 데이터 프레임 생성 예시: 모 병원에서 얻은 환자의 인구학적 정보

```
id <- c(1:10)
sex <- rep(c("Female", "Male"), each = 5)
age <- c(34, 22, 54, 43, 44, 39, 38, 28, 31, 42)
sbp <- c(112, 118, 132, 128, 128, 124, 121, 119, 124, 109)
height <- c(165, 158, 161, 160, 168, 172, 175, 182, 168, 162)
weight <- c(52, 48, 59, 60, 48, 72, 73, 82, 64, 60)

df <- data.frame(id, sex, age, sbp, height, weight,
                  stringsAsFactors = FALSE)
df
```

| | id | sex | age | sbp | height | weight |
|----|----|--------|-----|-----|--------|--------|
| 1 | 1 | Female | 34 | 112 | 165 | 52 |
| 2 | 2 | Female | 22 | 118 | 158 | 48 |
| 3 | 3 | Female | 54 | 132 | 161 | 59 |
| 4 | 4 | Female | 43 | 128 | 160 | 60 |
| 5 | 5 | Female | 44 | 128 | 168 | 48 |
| 6 | 6 | Male | 39 | 124 | 172 | 72 |
| 7 | 7 | Male | 38 | 121 | 175 | 73 |
| 8 | 8 | Male | 28 | 119 | 182 | 82 |
| 9 | 9 | Male | 31 | 124 | 168 | 64 |
| 10 | 10 | Male | 42 | 109 | 162 | 60 |

```
attributes(df); str(df); summary(df)
```

```
$names
[1] "id"      "sex"     "age"     "sbp"     "height"   "weight"

$class
[1] "data.frame"
```

```
$row.names
[1] 1 2 3 4 5 6 7 8 9 10

'data.frame':   10 obs. of  6 variables:
 $ id    : int  1 2 3 4 5 6 7 8 9 10
 $ sex   : chr "Female" "Female" "Female" "Female" ...
 $ age   : num  34 22 54 43 44 39 38 28 31 42
 $ sbp   : num  112 118 132 128 128 124 121 119 124 109
 $ height: num  165 158 161 160 168 172 175 182 168 162
 $ weight: num  52 48 59 60 48 72 73 82 64 60

      id          sex         age        sbp
Min.   : 1.00  Length:10     Min.   :22.00  Min.   :109.0
1st Qu.: 3.25  Class :character  1st Qu.:31.75  1st Qu.:118.2
Median : 5.50  Mode  :character  Median :38.50  Median :122.5
Mean   : 5.50                  Mean   :37.50  Mean   :121.5
3rd Qu.: 7.75                  3rd Qu.:42.75  3rd Qu.:127.0
Max.   :10.00                  Max.   :54.00  Max.   :132.0

      height       weight
Min.   :158.0  Min.   :48.00
1st Qu.:161.2  1st Qu.:53.75
Median :166.5  Median :60.00
Mean   :167.1  Mean   :61.80
3rd Qu.:171.0  3rd Qu.:70.00
Max.   :182.0  Max.   :82.00
```

```
# stringsAsFactors = TRUE 인 경우 sex의 summary() 결과
df <- data.frame(id, sex, age, sbp, height, weight,
                  stringsAsFactors = TRUE)
summary(df)
```

| | id | sex | age | sbp | height |
|---------|-------|------------------|---------------|---------------|---------------|
| Min. | 1.00 | Length:10 | Min. :22.00 | Min. :109.0 | Min. :158.0 |
| 1st Qu. | 3.25 | Class :character | 1st Qu.:31.75 | 1st Qu.:118.2 | 1st Qu.:161.2 |
| Median | 5.50 | Mode :character | Median :38.50 | Median :122.5 | Median :166.5 |
| Mean | 5.50 | | Mean :37.50 | Mean :121.5 | Mean :167.1 |
| 3rd Qu. | 7.75 | | 3rd Qu.:42.75 | 3rd Qu.:127.0 | 3rd Qu.:171.0 |
| Max. | 10.00 | | Max. :54.00 | Max. :132.0 | Max. :182.0 |

```

Min.    : 1.00   Female:5   Min.    :22.00   Min.    :109.0   Min.    :158.0
1st Qu.: 3.25   Male   :5   1st Qu.:31.75   1st Qu.:118.2   1st Qu.:161.2
Median  : 5.50           Median :38.50   Median :122.5   Median :166.5
Mean    : 5.50           Mean   :37.50   Mean   :121.5   Mean   :167.1
3rd Qu.: 7.75           3rd Qu.:42.75   3rd Qu.:127.0   3rd Qu.:171.0
Max.    :10.00           Max.   :54.00   Max.   :132.0   Max.   :182.0

      weight
Min.    :48.00
1st Qu.:53.75
Median  :60.00
Mean    :61.80
3rd Qu.:70.00
Max.    :82.00

```



- `summary()` 함수는 객체의 클래스에 따라 요약 통계량을 출력해주는 함수
- 데이터 프레임이 가지고 있는 변수들의 특징을 손쉽게 알아볼 수 있기 때문에 가장 많이 호출되는 함수 중 하나
- 숫자형 벡터: 최솟값(minimum), 1/4 분위수(1st quantile), 중앙값(median), 평균(mean), 3/4 분위수(3rd quantile), 최댓값을 출력
- 요인형 객체: 요인의 각 수준 별 빈도를 출력
- 2차원 이상 `table()` 객체에 적용 시 χ^2 검정(독립성 검정) 결과값을 출력함.

- 이미 정의된 데이터 프레임에 데이터를 추가 가능
 - 예를 들어 `dbp`라는 벡터에 이완기 혈압(diastolic blood pressure) 데이터가 입력되어 있고 `df`에 `dbp` 변수를 새롭게 추가 시 `df$dbp <- x` 형태로 추가
 - 위 형태로 이미 존재하고 있는 변수(열)에 새로운 값 재할당 가능
 - 이러한 형태로 문자형 벡터 추가 시 문자형 벡터는 자동으로 `factor`로 형 변환 되지는 않음

```

x <- 1:nrow(df)
dbp <- c(73, 70, 88, 82, 75, 77, 74, 81, 72, 64)

# df0의 "dbp" 열을 생성하고 x 값 대입
df$dbp <- x
df

```

| | id | sex | age | sbp | height | weight | dbp |
|----|----|--------|-----|-----|--------|--------|-----|
| 1 | 1 | Female | 34 | 112 | 165 | 52 | 1 |
| 2 | 2 | Female | 22 | 118 | 158 | 48 | 2 |
| 3 | 3 | Female | 54 | 132 | 161 | 59 | 3 |
| 4 | 4 | Female | 43 | 128 | 160 | 60 | 4 |
| 5 | 5 | Female | 44 | 128 | 168 | 48 | 5 |
| 6 | 6 | Male | 39 | 124 | 172 | 72 | 6 |
| 7 | 7 | Male | 38 | 121 | 175 | 73 | 7 |
| 8 | 8 | Male | 28 | 119 | 182 | 82 | 8 |
| 9 | 9 | Male | 31 | 124 | 168 | 64 | 9 |
| 10 | 10 | Male | 42 | 109 | 162 | 60 | 10 |

```

# df0의 dbp0의 dbp 벡터의 값을 재할당
df$dbp <- dbp
df

```

| | id | sex | age | sbp | height | weight | dbp |
|---|----|--------|-----|-----|--------|--------|-----|
| 1 | 1 | Female | 34 | 112 | 165 | 52 | 73 |
| 2 | 2 | Female | 22 | 118 | 158 | 48 | 70 |
| 3 | 3 | Female | 54 | 132 | 161 | 59 | 88 |
| 4 | 4 | Female | 43 | 128 | 160 | 60 | 82 |
| 5 | 5 | Female | 44 | 128 | 168 | 48 | 75 |
| 6 | 6 | Male | 39 | 124 | 172 | 72 | 77 |
| 7 | 7 | Male | 38 | 121 | 175 | 73 | 74 |

```

8   8   Male  28 119      182      82  81
9   9   Male  31 124      168      64  72
10 10   Male  42 109      162      60  64

```

```

# df0의 운동여부 exercyn 라는 변수 추가
# exercyn 는 "Y" 또는 "N" 두 값을 가짐
df$exercyn <- c("Y", "Y", "N", "Y", "N",
                 "N", "N", "Y", "N", "Y")
str(df)

```

```

'data.frame': 10 obs. of 8 variables:
 $ id     : int 1 2 3 4 5 6 7 8 9 10
 $ sex    : Factor w/ 2 levels "Female","Male": 1 1 1 1 1 2 2 2 2 2
 $ age    : num 34 22 54 43 44 39 38 28 31 42
 $ sbp    : num 112 118 132 128 128 124 121 119 124 109
 $ height: num 165 158 161 160 168 172 175 182 168 162
 $ weight: num 52 48 59 60 48 72 73 82 64 60
 $ dbp    : num 73 70 88 82 75 77 74 81 72 64
 $ exercyn: chr "Y" "Y" "N" "Y" ...

```

- 행렬 및 벡터에서 언급 되었던 rownames(), colnames(), names(), dim(), ncol()/NCOL(), nrow()/NROW() 함수 적용 가능

```
rownames(df); colnames(df); names(df)
```

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

```
[1] "id"       "sex"       "age"       "sbp"       "height"    "weight"    "dbp"
[8] "exercyn"
```

```
[1] "id"      "sex"     "age"     "sbp"     "height"   "weight"   "dbp"
[8] "exercyn"
```

```
dim(df); ncol(df); nrow(df)
```

```
[1] 10 8
```

```
[1] 8
```

```
[1] 10
```

```
# rownames() 함수를 통해 행이름 변경
rownames(df) <- letters[1:10]
df
```

| | id | sex | age | sbp | height | weight | dbp | exercyn |
|---|-----------|------------|------------|------------|---------------|---------------|------------|----------------|
| a | 1 | Female | 34 | 112 | 165 | 52 | 73 | Y |
| b | 2 | Female | 22 | 118 | 158 | 48 | 70 | Y |
| c | 3 | Female | 54 | 132 | 161 | 59 | 88 | N |
| d | 4 | Female | 43 | 128 | 160 | 60 | 82 | Y |
| e | 5 | Female | 44 | 128 | 168 | 48 | 75 | N |
| f | 6 | Male | 39 | 124 | 172 | 72 | 77 | N |
| g | 7 | Male | 38 | 121 | 175 | 73 | 74 | N |
| h | 8 | Male | 28 | 119 | 182 | 82 | 81 | Y |
| i | 9 | Male | 31 | 124 | 168 | 64 | 72 | N |
| j | 10 | Male | 42 | 109 | 162 | 60 | 64 | Y |

```
#colnames() 함수를 통해 열 이름 변경
varname_orig <- colnames(df)
colnames(df) <- paste0("V", 1:ncol(df))
df
```

| | V1 | V2 | V3 | V4 | V5 | V6 | V7 | V8 |
|---|----|--------|----|-----|-----|----|----|----|
| a | 1 | Female | 34 | 112 | 165 | 52 | 73 | Y |
| b | 2 | Female | 22 | 118 | 158 | 48 | 70 | Y |
| c | 3 | Female | 54 | 132 | 161 | 59 | 88 | N |
| d | 4 | Female | 43 | 128 | 160 | 60 | 82 | Y |
| e | 5 | Female | 44 | 128 | 168 | 48 | 75 | N |
| f | 6 | Male | 39 | 124 | 172 | 72 | 77 | N |
| g | 7 | Male | 38 | 121 | 175 | 73 | 74 | N |
| h | 8 | Male | 28 | 119 | 182 | 82 | 81 | Y |
| i | 9 | Male | 31 | 124 | 168 | 64 | 72 | N |
| j | 10 | Male | 42 | 109 | 162 | 60 | 64 | Y |

```
# names() 함수와 colnames()는 거의 동일한 기능 수행
# 두 함수의 차이점?
names(df)
```

```
[1] "V1" "V2" "V3" "V4" "V5" "V6" "V7" "V8"
```

```
names(df) <- varname_orig
df
```

| | id | sex | age | sbp | height | weight | dbp | exercyn |
|---|----|--------|-----|-----|--------|--------|-----|---------|
| a | 1 | Female | 34 | 112 | 165 | 52 | 73 | Y |
| b | 2 | Female | 22 | 118 | 158 | 48 | 70 | Y |

| | | | | | | | | |
|---|----|--------|----|-----|-----|----|----|---|
| c | 3 | Female | 54 | 132 | 161 | 59 | 88 | N |
| d | 4 | Female | 43 | 128 | 160 | 60 | 82 | Y |
| e | 5 | Female | 44 | 128 | 168 | 48 | 75 | N |
| f | 6 | Male | 39 | 124 | 172 | 72 | 77 | N |
| g | 7 | Male | 38 | 121 | 175 | 73 | 74 | N |
| h | 8 | Male | 28 | 119 | 182 | 82 | 81 | Y |
| i | 9 | Male | 31 | 124 | 168 | 64 | 72 | N |
| j | 10 | Male | 42 | 109 | 162 | 60 | 64 | Y |



참고: R Markdown에서 데이터 프레임의 데이터를 손쉽게 테이블로 출력하는 방법
(html 문서)

- R Markdwon의 YAML 부분에 다음과 같이 옵션을 추가하면 별다른 함수 처리 없이 데이터 프레임을 표 형태로 html 문서에 붙일 수 있음. 아래 예시에서 `output` 이후 `df_print: paged` 옵션을 추가
- 옵션 추가 시 들여쓰기(탭 구분)은 YAML 문서의 트리 구조를 표현한 것인기 때문에 **꼭** 들여쓰기를 정확히 일치시켜야 함

```
---
title: "문서 제목"
author: "이름"
date: "`r Sys.Date()`"
output:
  html_document:
    df_print: paged
---
```

2.8.2 데이터 프레임 접근 및 필터링

접근방법

- 리스트 데이터 접근 방식

```
# 추출(접근) 연산자(함수) `df$col_name` 형태로 접근  
df$height
```

```
[1] 165 158 161 160 168 172 175 182 168 162
```

```
# df[[index]] 또는 df[["col_name"]]
```

형태로 접근
df[[4]]

```
[1] 112 118 132 128 128 124 121 119 124 109
```

```
df[["sex"]]
```

```
[1] Female Female Female Female Female Male   Male   Male   Male   Male  
Levels: Female Male
```

```
w <- df[[4]]  
attributes(w); str(w)
```

```
NULL
```

```
num [1:10] 112 118 132 128 128 124 121 119 124 109
```

```
# df[index] 또는 df[["col_name"]]
```

형태로 접근
h <- df[["height"]]
attributes(h); str(h)

```
$names
[1] "height"

$row.names
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"

$class
[1] "data.frame"

'data.frame':   10 obs. of  1 variable:
 $ height: num  165 158 161 160 168 172 175 182 168 162
```

- 행렬 데이터 접근 방식

```
# df[idx_row, idx_col] 또는 df[row_name, col_name]
# 형태 데이터 접근

# 열 index 접근
df[, 3];
```

```
[1] 34 22 54 43 44 39 38 28 31 42
```

```
# 형 강제 변환 방지
df[, 3, drop = FALSE]
```

```
age
a 34
b 22
c 54
```

d 43
e 44
f 39
g 38
h 28
i 31
j 42

```
# 행 index 접근  
df[8, ]
```

```
id sex age sbp height weight dbp exercyn  
h 8 Male 28 119 182 82 81 Y
```

```
# 행과 열 index 접근  
df[1:4, 5:6]
```

```
height weight  
a 165 52  
b 158 48  
c 161 59  
d 160 60
```

```
# 열 이름으로 접근  
df[, c("sex", "sbp")]
```

```
sex sbp  
a Female 112
```

```
b Female 118
c Female 132
d Female 128
e Female 128
f Male 124
g Male 121
h Male 119
i Male 124
j Male 109
```

```
# 행 이름으로 접근
```

```
df[c("d", "e", "f"), ]
```

| | id | sex | age | sbp | height | weight | dbp | exercyn |
|---|----|--------|-----|-----|--------|--------|-----|---------|
| d | 4 | Female | 43 | 128 | 160 | 60 | 82 | Y |
| e | 5 | Female | 44 | 128 | 168 | 48 | 75 | N |
| f | 6 | Male | 39 | 124 | 172 | 72 | 77 | N |

```
# 행과 열 이름으로 접근
```

```
df[c("a", "f"), c("sex", "height", "dbp")]
```

| | sex | height | dbp |
|---|--------|--------|-----|
| a | Female | 165 | 73 |
| f | Male | 172 | 77 |

```
# 행 또는 열 제외
```

```
df[-c(2:6), ]
```

| | id | sex | age | sbp | height | weight | dbp | exercyn |
|---|----|--------|-----|-----|--------|--------|-----|---------|
| a | 1 | Female | 34 | 112 | 165 | 52 | 73 | Y |
| g | 7 | Male | 38 | 121 | 175 | 73 | 74 | N |
| h | 8 | Male | 28 | 119 | 182 | 82 | 81 | Y |
| i | 9 | Male | 31 | 124 | 168 | 64 | 72 | N |
| j | 10 | Male | 42 | 109 | 162 | 60 | 64 | Y |

```
df[-c(1, 5:7), -c(1, 8)]
```

| | sex | age | sbp | height | weight | dbp |
|---|--------|-----|-----|--------|--------|-----|
| b | Female | 22 | 118 | 158 | 48 | 70 |
| c | Female | 54 | 132 | 161 | 59 | 88 |
| d | Female | 43 | 128 | 160 | 60 | 82 |
| h | Male | 28 | 119 | 182 | 82 | 81 |
| i | Male | 31 | 124 | 168 | 64 | 72 |
| j | Male | 42 | 109 | 162 | 60 | 64 |

필터링

- 벡터, 행렬과 마찬가지로 비교 연산자를 이용해 조건에 맞는 부분 데이터 추출 가능

```
# %in% 연산자를 이용해 데이터 프레임의 부분 변수 추출
# id, age 열을 제외한 나머지 데이터 프레임 추출
varname_df <- names(df)
df[, !varname_df %in% c("id", "age")]
```

| | sex | sbp | height | weight | dbp | exercyn |
|---|--------|-----|--------|--------|-----|---------|
| a | Female | 112 | 165 | 52 | 73 | Y |

| b | Female | 118 | 158 | 48 | 70 | Y |
|---|--------|-----|-----|----|----|---|
| c | Female | 132 | 161 | 59 | 88 | N |
| d | Female | 128 | 160 | 60 | 82 | Y |
| e | Female | 128 | 168 | 48 | 75 | N |
| f | Male | 124 | 172 | 72 | 77 | N |
| g | Male | 121 | 175 | 73 | 74 | N |
| h | Male | 119 | 182 | 82 | 81 | Y |
| i | Male | 124 | 168 | 64 | 72 | N |
| j | Male | 109 | 162 | 60 | 64 | Y |

```
# 조건 연산자 사용
# sex 가 Female이고 나이가 40 이상인 데이터 추출
df[df$sex == "Female" & df$age >= 40, ]
```

| c | Female | 54 | 132 | 161 | 59 | 88 | N |
|---|--------|----|-----|-----|----|----|---|
| d | Female | 43 | 128 | 160 | 60 | 82 | Y |
| e | Female | 44 | 128 | 168 | 48 | 75 | N |

```
# id가 3보다 작은 데이터 추출
df[df[, 1] < 3, ]
```

| a | Female | 34 | 112 | 165 | 52 | 73 | Y |
|---|--------|----|-----|-----|----|----|---|
| b | Female | 22 | 118 | 158 | 48 | 70 | Y |

```
# subset 함수 이용한 데이터 추출
# sbp 가 120 이상이고 dbp 가 80 이상인 데이터 추출
subset(df, sbp >= 120 & dbp >= 80)
```

```

id      sex age sbp height weight dbp exercyn
c  3 Female  54 132     161      59   88       N
d  4 Female  43 128     160      60   82       Y

```

```

# 성별, 수축기, 이완기 혈압 변수만 추출
subset(df, select = c(sex, sbp, dbp))

```

```

sex sbp dbp
a Female 112 73
b Female 118 70
c Female 132 88
d Female 128 82
e Female 128 75
f Male 124 77
g Male 121 74
h Male 119 81
i Male 124 72
j Male 109 64

```

```

# id 변수 제거
subset(df, select = -c(id))

```

```

sex age sbp height weight dbp exercyn
a Female 34 112     165      52   73       Y
b Female 22 118     158      48   70       Y
c Female 54 132     161      59   88       N
d Female 43 128     160      60   82       Y
e Female 44 128     168      48   75       N
f Male 39 124     172      72   77       N
g Male 38 121     175      73   74       N

```

| | | | | | | | |
|---|------|----|-----|-----|----|----|---|
| h | Male | 28 | 119 | 182 | 82 | 81 | Y |
| i | Male | 31 | 124 | 168 | 64 | 72 | N |
| j | Male | 42 | 109 | 162 | 60 | 64 | Y |



데이터 프레임 또는 리스트 접근 시 `df$col_name` 를 사용한다면 매번 데이터 프레임 이름과 `$` 을 반복하기 때문에 코드가 불필요하게 복잡해짐. R에서는 데이터 프레임 내부의 열 이름을 직접 접근할 수 있도록 도와주는 몇 가지 함수(예: `with()`, `attach()` 등)가 있는데, `with()`와 `within()` 활용법에 대해 간략히 알아봄.

- 위 예제에서 `sex` 가 `Female`이고 나이가 40 이상인 데이터 추출한다고 했을 때 `with()` 함수 사용

```
# with() 함수: 데이터 환경(객체 내)에서 주어진 표현식의 결과를 반환
with(
  data, #리스트 또는 데이터 프레임
  expr, # 실제 명령을 수행할 표현식,
)
```

```
with(df, df[sex == "Female" & age >= 40, ])
```

| | id | sex | age | sbp | height | weight | dbp | exercyn |
|---|----|--------|-----|-----|--------|--------|-----|---------|
| c | 3 | Female | 54 | 132 | 161 | 59 | 88 | N |
| d | 4 | Female | 43 | 128 | 160 | 60 | 82 | Y |
| e | 5 | Female | 44 | 128 | 168 | 48 | 75 | N |

- `within()` 함수는 `with()` 와 유사하지만 코드블록(`{...}`) 을 이용해 보다 자유롭게 데이터를 수정 및 추가할 수 있음

```
df2 <- within(df, {  
  hospital <- c("A", "B", "B", "A", "C",  
    "A", "A", "B", "C", "B")  
  mean_age <- mean(age)  
})
```

2.8.3 데이터 프레임 관련 함수

유ти리티 함수

- 보통 데이터 분석은 외부에서 데이터를 읽은 후 특정 객체에 읽어온 데이터를 할당하는데, 이 경우 데이터가 저장된 객체는 대부분은 데이터 프레임 형태임.
- 읽어온 데이터는 보통 많은 행(표본)으로 구성되어 있기 때문에 데이터를 손쉽게 살펴보는 방법이 필요

head()/*tail()* 함수

- 객체(벡터, 행렬, 테이블, 데이터 프레임 등)의 처음 또는 끝에서부터 몇 개의 데이터(`default = 6L`)를 순차적으로 보여줌

```
# 앞에서 불러온 전복 데이터셋 확인  
dim(abalone)
```

```
[1] 4177     9
```

```
#처음 10에서 6행 까지 데이터 출력
head(abalone)
```

| | V1 | V2 | V3 | V4 | V5 | V6 | V7 | V8 | V9 |
|---|----|-------|-------|-------|--------|--------|--------|-------|----|
| 1 | M | 0.455 | 0.365 | 0.095 | 0.5140 | 0.2245 | 0.1010 | 0.150 | 15 |
| 2 | M | 0.350 | 0.265 | 0.090 | 0.2255 | 0.0995 | 0.0485 | 0.070 | 7 |
| 3 | F | 0.530 | 0.420 | 0.135 | 0.6770 | 0.2565 | 0.1415 | 0.210 | 9 |
| 4 | M | 0.440 | 0.365 | 0.125 | 0.5160 | 0.2155 | 0.1140 | 0.155 | 10 |
| 5 | I | 0.330 | 0.255 | 0.080 | 0.2050 | 0.0895 | 0.0395 | 0.055 | 7 |
| 6 | I | 0.425 | 0.300 | 0.095 | 0.3515 | 0.1410 | 0.0775 | 0.120 | 8 |

```
# 제일 마지막 행부터 위로 6개 데이터 까지 출력
tail(abalone)
```

| | V1 | V2 | V3 | V4 | V5 | V6 | V7 | V8 | V9 |
|------|----|-------|-------|-------|--------|--------|--------|--------|----|
| 4172 | M | 0.560 | 0.430 | 0.155 | 0.8675 | 0.4000 | 0.1720 | 0.2290 | 8 |
| 4173 | F | 0.565 | 0.450 | 0.165 | 0.8870 | 0.3700 | 0.2390 | 0.2490 | 11 |
| 4174 | M | 0.590 | 0.440 | 0.135 | 0.9660 | 0.4390 | 0.2145 | 0.2605 | 10 |
| 4175 | M | 0.600 | 0.475 | 0.205 | 1.1760 | 0.5255 | 0.2875 | 0.3080 | 9 |
| 4176 | F | 0.625 | 0.485 | 0.150 | 1.0945 | 0.5310 | 0.2610 | 0.2960 | 10 |
| 4177 | M | 0.710 | 0.555 | 0.195 | 1.9485 | 0.9455 | 0.3765 | 0.4950 | 12 |

View() 함수

- 2차원 데이터의 readable 한 스프레드시트 제공

```
View(abalone)
```

데이터 프레임 결합 및 분리 함수

rbind()/*cbind()* 함수

- 행렬에서 사용한 *rbind()*/*cbind()*를 데이터 프레임에도 적용 가능

```
a = data.frame(x1 = rep(0,5), x2 = rep("x",5))
b = data.frame(x1 = rep(1,5), x2 = rep("d",5))
c = data.frame(x3 = rep(2,5), x4 = rep("z",5))
d <- list(1, "d")
e <- list(x5 = rep(4, 5), x6 = rep("y", 5))
# rbind()를 이용해 두 데이터 프레임 a-b 합치기
ab <- rbind(a, b)
ab
```

| | x1 | x2 |
|----|----|----|
| 1 | 0 | x |
| 2 | 0 | x |
| 3 | 0 | x |
| 4 | 0 | x |
| 5 | 0 | x |
| 6 | 1 | d |
| 7 | 1 | d |
| 8 | 1 | d |
| 9 | 1 | d |
| 10 | 1 | d |

```
# 변수명이 다른 경우
rbind(a, c) #변수명이 다르기때문에 행으로 묶을 수 없다.
```

Error in match.names(clabs, names(xi)): names do not match previous names

```
# rbind()를 이용해 데이터 프레임-리스트 합치기
abd <- rbind(ab, d)
abd
```

```
x1 x2
1 0 x
2 0 x
3 0 x
4 0 x
5 0 x
6 1 d
7 1 d
8 1 d
9 1 d
10 1 d
11 1 d
```

```
# cbind()를 이용해 두 데이터 프레임 a-c 합치기
ac <- cbind(a, c)
ac
```

```
x1 x2 x3 x4
1 0 x 2 z
2 0 x 2 z
3 0 x 2 z
4 0 x 2 z
5 0 x 2 z
```

```
# 행 길이가 다르면 작은 길이의 데이터를 재사용
cbind(a, d)
```

```
x1 x2 1 "d"
1 0 x 1 d
2 0 x 1 d
3 0 x 1 d
4 0 x 1 d
5 0 x 1 d
```

```
# cbind()를 이용해 두 데이터 프레임-리스트 합치기
ace <- cbind(ac, e)
```

merge() 함수

- 두 데이터 프레임을 공통된 값을 기준으로 병합
- Excel의 vlookup() 함수 또는 데이터베이스 SQL 쿼리 중 join과 동일한 역할을 함
- cbind()의 경우는 단순히 열을 합치는 것이지만 merge()는 공통되는 열을 기준으로 두 데이터셋을 병합
- 공통된 데이터가 있을 때만 데이터 병합 수행

```
# merge() 함수 인수
merge(
  x, # 병합할 데이터 프레임
  y, # 병합할 데이터 프레임
  by, # 병합 기준으로 사용할 컬럼 (문자열 벡터)
  by.x, # 병합에 사용할 x와 y의 열 이름이 다른 경우
```

```

by.y, # by.x와 by.y에 각각 공통 데이터에 해당하는 열 이름 지정
      # 둘 다 문자형 스칼라 또는 벡터값 인수로 받음
all, # 논리값 이순
      # TRUE인 경우 x, y 중 공통된 값을 갖는 행이 없을 때
      # 해당 쪽을 NA를 채워 병합
      # 결과적으로 x, y 전체 행이 결과에 포함
all.x, # x,y 중 특정 쪽에 공통된 값이 없더라도 항상
all.y, # 결과에 포함
)

```

- merge() 함수 예시

```

d1 = data.frame(Name = c("Park", "Hanzo", "Mercy", "Soldier76"),
                 country = c("Korea", "Japan", "Swiss", "USA"))
d2 = data.frame(Age = c(19,38,37,56,31),
                 Name = c("Park", "Hanzo", "Mercy", "Soldier76", "Mei"))
d1; d2

```

| | Name | country |
|---|-----------|---------|
| 1 | Park | Korea |
| 2 | Hanzo | Japan |
| 3 | Mercy | Swiss |
| 4 | Soldier76 | USA |

| | Age | Name |
|---|-----|-----------|
| 1 | 19 | Park |
| 2 | 38 | Hanzo |
| 3 | 37 | Mercy |
| 4 | 56 | Soldier76 |
| 5 | 31 | Mei |

```
dim(d1); dim(d2)
```

```
[1] 4 2
```

```
[1] 5 2
```

```
# 두 데이터 병합 01  
merge(d1, d2, by = "Name")
```

| | Name | country | Age |
|---|-----------|---------|-----|
| 1 | Hanzo | Japan | 38 |
| 2 | Mercy | Swiss | 37 |
| 3 | Park | Korea | 19 |
| 4 | Soldier76 | USA | 56 |

```
# 두 데이터 병합 02  
names(d2)[2] <- "Surname"  
merge(d1, d2, by.x = "Name", by.y = "Surname")
```

| | Name | country | Age |
|---|-----------|---------|-----|
| 1 | Hanzo | Japan | 38 |
| 2 | Mercy | Swiss | 37 |
| 3 | Park | Korea | 19 |
| 4 | Soldier76 | USA | 56 |

```
# 두 데이터 병합 03  
merge(d1, d2,  
      by.x = "Name", by.y = "Surname",  
      all = T)
```

| | Name | country | Age |
|---|-----------|---------|-----|
| 1 | Hanzo | Japan | 38 |
| 2 | Mei | <NA> | 31 |
| 3 | Mercy | Swiss | 37 |
| 4 | Park | Korea | 19 |
| 5 | Soldier76 | USA | 56 |

split() 함수

- Factor 형에서 언급한 *split()* 함수를 통해 그룹 별로 데이터 분할
- 분할된 데이터는 리스트에 저장

```
split(df, df$sex)
```

\$Female

| | id | sex | age | sbp | height | weight | dbp | exercyn |
|---|----|--------|-----|-----|--------|--------|-----|---------|
| a | 1 | Female | 34 | 112 | 165 | 52 | 73 | Y |
| b | 2 | Female | 22 | 118 | 158 | 48 | 70 | Y |
| c | 3 | Female | 54 | 132 | 161 | 59 | 88 | N |
| d | 4 | Female | 43 | 128 | 160 | 60 | 82 | Y |
| e | 5 | Female | 44 | 128 | 168 | 48 | 75 | N |

\$Male

| | id | sex | age | sbp | height | weight | dbp | exercyn |
|---|----|------|-----|-----|--------|--------|-----|---------|
| f | 6 | Male | 39 | 124 | 172 | 72 | 77 | N |
| g | 7 | Male | 38 | 121 | 175 | 73 | 74 | N |
| h | 8 | Male | 28 | 119 | 182 | 82 | 81 | Y |
| i | 9 | Male | 31 | 124 | 168 | 64 | 72 | N |
| j | 10 | Male | 42 | 109 | 162 | 60 | 64 | Y |

데이터 정렬 함수

sort() 함수

- 데이터(벡터)의 정렬(오름차순 또는 내림차순) 결과 반환

```
# sort() 함수 인수
sort(
  x, # 정렬할 벡터
  decreasing, # 논리값, 내림차순 여부
  # default = FALSE
  na.last # 논리값. 결측 존재 시 NA 값 위치 지정
)        # TRUE: 정렬 후 결측은 마지막에 위치
          # FALSE: 맨 처음 NA 위치
```

- 예시

```
# 오름차순 정렬
sort(df2$age)
```

```
[1] 22 28 31 34 38 39 42 43 44 54
```

```
# 내림차순 정렬
sort(df$height, decreasing = TRUE)
```

```
[1] 182 175 172 168 168 165 162 161 160 158
```

order() 함수

- 데이터 정렬을 위해 순서에 대한 색인 생성 결과 반환
- 데이터 프레임에서 특정 열 기준으로 데이터 정렬 시 주로 사용

```
# 나이 기준으로 오름차순으로 데이터 정렬
with(df, df[order(age), ])
```

| | id | sex | age | sbp | height | weight | dbp | exercyn |
|---|----|--------|-----|-----|--------|--------|-----|---------|
| b | 2 | Female | 22 | 118 | 158 | 48 | 70 | Y |
| h | 8 | Male | 28 | 119 | 182 | 82 | 81 | Y |
| i | 9 | Male | 31 | 124 | 168 | 64 | 72 | N |
| a | 1 | Female | 34 | 112 | 165 | 52 | 73 | Y |
| g | 7 | Male | 38 | 121 | 175 | 73 | 74 | N |
| f | 6 | Male | 39 | 124 | 172 | 72 | 77 | N |
| j | 10 | Male | 42 | 109 | 162 | 60 | 64 | Y |
| d | 4 | Female | 43 | 128 | 160 | 60 | 82 | Y |
| e | 5 | Female | 44 | 128 | 168 | 48 | 75 | N |
| c | 3 | Female | 54 | 132 | 161 | 59 | 88 | N |

```
# 키 순으로 내림차순 정렬
df[order(df$height, decreasing = T), ]
```

| | id | sex | age | sbp | height | weight | dbp | exercyn |
|---|----|--------|-----|-----|--------|--------|-----|---------|
| h | 8 | Male | 28 | 119 | 182 | 82 | 81 | Y |
| g | 7 | Male | 38 | 121 | 175 | 73 | 74 | N |
| f | 6 | Male | 39 | 124 | 172 | 72 | 77 | N |
| e | 5 | Female | 44 | 128 | 168 | 48 | 75 | N |
| i | 9 | Male | 31 | 124 | 168 | 64 | 72 | N |
| a | 1 | Female | 34 | 112 | 165 | 52 | 73 | Y |

| | | | | | | | | |
|---|----|--------|----|-----|-----|----|----|---|
| j | 10 | Male | 42 | 109 | 162 | 60 | 64 | Y |
| c | 3 | Female | 54 | 132 | 161 | 59 | 88 | N |
| d | 4 | Female | 43 | 128 | 160 | 60 | 82 | Y |
| b | 2 | Female | 22 | 118 | 158 | 48 | 70 | Y |

2.8.4 *apply() 계열 함수

- `apply()`, `lapply()`, `sapply()` 등 `apply` 계열 함수는 R에서 가장 일 반적으로 사용되는 함수 중 하나
- 반복문(for-loop)를 대신하기 위해 활용되며, R 객체를 입력 받아 원소 별 혹은 그루 별 함수를 적용
- 데이터 전체에 함수를 한번에 적용하는 vectorizing 연산을 수행함

apply() 함수

- 배열 또는 행렬에 주어진 함수를 적용한 뒤 그 결과를 벡터 또는 리스트로 반환
- 행 또는 열 차원 기준 함수 적용
- 동일한 유형의 벡터로 구성된 데이터셋에 적용

```
apply(
  X, # 배열, 행렬, 또는 같은 형태로 정의된 데이터 프레임
  MARGIN, # MARGIN = 1: 행 기준
            # MARGIN = 2: 열 기준
            # MARGIN = c(1,2): 행과 열 방향 모두
  FUN # 적용할 함수
)
```

- 예시1: 행렬 및 배열 `apply()` 적용

```
X <- matrix(1:9, nrow = 3)
X
```

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

```
# 행 기준으로 합계 계산
apply(X, 1, sum)
```

```
[1] 12 15 18
```

```
# 열 기준으로 합계 계산
apply(X, 2, sum)
```

```
[1] 6 15 24
```

```
# 배열에 apply 적용
# 각 학생의 퀴즈와 중간-기말 각각 평균 계산
Z # 성적
```

```
, , 1
```

```
[,1] [,2]
[1,] 75 65
[2,] 84 78
```

```
[3,] 93 92
```

```
, , 2
```

```
[,1] [,2]  
[1,] 82 88  
[2,] 78 75  
[3,] 85 88
```

```
apply(Z, c(1,2), mean)
```

```
[,1] [,2]  
[1,] 78.5 76.5  
[2,] 81.0 76.5  
[3,] 89.0 90.0
```

```
# 각 시점 별 개별 학생의 퀴즈-중간, 퀴즈-기말 평균 계산  
apply(Z, c(1, 3), mean)
```

```
[,1] [,2]  
[1,] 70.0 85.0  
[2,] 81.0 76.5  
[3,] 92.5 86.5
```

- 예시2: 데이터 프레임

- 위에서 사용한 df와 2.7.1 요인(factor) 절에서 잠깐 예시로 사용 된 전복(abalone) 데이터셋 사용



Abalone dataset 변수 설명(코드북)

| | Origin | Variable | Name | Unit | Description |
|---|--------|------------|----------------|-------|-------------------------|
| 1 | V1 | sex | Sex | | 성별(M: 수컷; F: 암컷; I: 새끼) |
| 2 | V2 | length | Length | mm | 길이(최장길이) |
| 3 | V3 | diameter | Diameter | mm | 직경 |
| 4 | V4 | height | Height | mm | 껍질 내 육질의 높이 |
| 5 | V5 | whole.wt | Whole weight | grams | 전체 중량 |
| 6 | V6 | shucked.wt | Shucked weight | grams | 육질 무게 |
| 7 | V7 | viscera.wt | Viscera weight | grams | 내장 무게 |
| 8 | V8 | shell.wt | Shell weight | grams | 껍질 무게 |
| 9 | V9 | rings | Rings | | 전복 나이 |

```
names(abalone) <- c("sex", "length", "diameter",
                      "height", "whole.wt",
                      "shucked.wt", "viscera.wt",
                      "shell.wt", "rings")
head(abalone)
```

| | sex | length | diameter | height | whole.wt | shucked.wt | viscera.wt | shell.wt | rings |
|---|-----|--------|----------|--------|----------|------------|------------|----------|-------|
| 1 | M | 0.455 | 0.365 | 0.095 | 0.5140 | 0.2245 | 0.1010 | 0.150 | 15 |
| 2 | M | 0.350 | 0.265 | 0.090 | 0.2255 | 0.0995 | 0.0485 | 0.070 | 7 |
| 3 | F | 0.530 | 0.420 | 0.135 | 0.6770 | 0.2565 | 0.1415 | 0.210 | 9 |
| 4 | M | 0.440 | 0.365 | 0.125 | 0.5160 | 0.2155 | 0.1140 | 0.155 | 10 |
| 5 | I | 0.330 | 0.255 | 0.080 | 0.2050 | 0.0895 | 0.0395 | 0.055 | 7 |
| 6 | I | 0.425 | 0.300 | 0.095 | 0.3515 | 0.1410 | 0.0775 | 0.120 | 8 |

```
# sex를 제외한 나머지 수치형 변수에 대한 기초통계량 계산
# 평균: mean() 함수 사용
apply(abalone[, -1], 2, mean)
```

```

length      diameter      height      whole.wt shucked.wt viscera.wt      shell.wt
0.5239921  0.4078813  0.1395164  0.8287422  0.3593675  0.1805936  0.2388309
      rings
9.9336845

```

```

# 표준편차: sd() 함수 사용
apply(abalone[, -1], 2, sd)

```

```

length      diameter      height      whole.wt shucked.wt viscera.wt      shell.wt
0.12009291 0.09923987 0.04182706 0.49038902 0.22196295 0.10961425 0.13920267
      rings
3.22416903

```

```

# 개별 전복에 대해 내장, 육질, 껍질 무게 합계 계산
apply(abalone[, c("shucked.wt",
                  "viscera.wt",
                  "shell.wt")], 1,
      sum) -> ab_wt_sum

head(ab_wt_sum, 10)

```

```
[1] 0.4755 0.2180 0.6080 0.4845 0.1840 0.3385 0.7085 0.7035 0.4940 0.7855
```

```

# 데이터에 결측이 포함된 경우
# diameter 변수에 10개의 결측을 임의 생성
set.seed(20200410)
idx <- sample(1:NROW(abalone), 10) # 비복원 추출
ab2 <- abalone

```

```
ab2[idx, 3] <- NA

# 성별 제외한 나머지 변수의 평균 계산
apply(ab2[, -1], 2, mean)
```

```
length      diameter      height      whole.wt shucked.wt viscera.wt      shell.wt
0.5239921          NA  0.1395164  0.8287422  0.3593675  0.1805936  0.2388309
rings
9.9336845
```

```
# NA 결과를 피하려면?
apply(ab2[, -1], 2, mean, na.rm = TRUE)
```

```
length      diameter      height      whole.wt shucked.wt viscera.wt      shell.wt
0.5239921  0.4079578  0.1395164  0.8287422  0.3593675  0.1805936  0.2388309
rings
9.9336845
```

참고 1:

- 결측이 포함된 벡터 연산 시 결측에 대한 처리 지정 없이 함수를 적용하면 결측값을 반환
- R에서 제공되는 연산 관련 일반 함수는 결측처리에 대한 옵션을 인수로 받음
- 보통 인수 형태는 `na.rm = T/F` 형태이고 다음의 함수를 통해 데이터에 결측 처리에 대한 속성 및 클래스를 부여함
- `na.omit()`/`na.exclude()`: NA가 포함되어 있는 행 생략

위 두 함수는 기본적으로 동일하지만, 특정 함수(예: 회귀분석을 수행하는 `lm()`) 함수에서는 다른 결과를 출력



참고 2: 위 예제에서 보여준 행 또는 열의 합 또는 평균 계산은 매우 자주 사용되기 때문에 `rowSums()`, `colSums()`, `rowMeans()`, `colMeans()` 함수가 제공됨

```
colMeans(abalone[,-1]) # apply 결과와 비교
```

```
length      diameter      height      whole.wt shucked.wt viscera.wt      shell.wt
0.5239921   0.4078813   0.1395164   0.8287422   0.3593675   0.1805936   0.2388309
rings
9.9336845
```

tapply() 함수

- 2.7.1 요인(factor) 절에서 설명
- `tapply()`는 1개의 벡터를 대상으로만 함수를 호출
- 데이터 프레임에 적용하려면? → `aggregate()` 함수 사용

aggregate()

- 데이터를 특정 factor의 수준 별로 나눈 후, 각 그룹마다 함수 적용
- `aggregate()`는 다음 두 가지 형태로 함수 적용 가능

```
# aggregate() 기본 인수
aggregate(
  x, # R 객체, 주로 데이터 프레임
  by, # 그룹으로 묶을 값의 리스트
  FUN, # 그룹별 적용할 함수
)
```

```
aggregate(
  formula, # y ~ x 형태로 y는 계산에 사용할 값
  # x는 group 변수
  # y 대신 .은 그룹 변수를 제외하고
  # 적용할 함수의 대상이 되는 모든 변수
  data, # formula를 적용할 데이터
  FUN # 적용할 함수
)
```

- 예시: 임상연구 자료(df)

```
# 성별에 따라 연속형 변수의 평균 계산
aggregate(df[,-c(1:2, 8)], by =df[["sex"]], mean)
```

```
sex   age    sbp height weight  dbp
1 Female 39.4 123.6 162.4   53.4 77.6
2   Male 35.6 119.4 171.8   70.2 73.6
```

```
# 수식 표현형 사용
aggregate(. ~ sex,
  data = df[,-8],
  mean)
```

```
sex id   age    sbp height weight  dbp
1 Female 3 39.4 123.6 162.4   53.4 77.6
2   Male 8 35.6 119.4 171.8   70.2 73.6
```

```
# 요인의 2개인 경우
aggregate(df[,-c(1:2, 8)],
           by = list(sex = df[["sex"]], exercise = df[["exercyn"]]),
           mean)
```

| | sex | exercise | age | sbp | height | weight | dbp |
|---|--------|----------|-----|----------|-----------|----------|----------|
| 1 | Female | N | 49 | 130.0000 | 164.5000 | 53.50000 | 81.50000 |
| 2 | Male | N | 36 | 123.0000 | 171.66667 | 69.66667 | 74.33333 |
| 3 | Female | Y | 33 | 119.3333 | 161.0000 | 53.33333 | 75.00000 |
| 4 | Male | Y | 35 | 114.0000 | 172.0000 | 71.00000 | 72.50000 |

```
aggregate(. ~ sex + exercyn,
           data = df[,-1],
           mean)
```

| | sex | exercyn | age | sbp | height | weight | dbp |
|---|--------|---------|-----|----------|-----------|----------|----------|
| 1 | Female | N | 49 | 130.0000 | 164.5000 | 53.50000 | 81.50000 |
| 2 | Male | N | 36 | 123.0000 | 171.66667 | 69.66667 | 74.33333 |
| 3 | Female | Y | 33 | 119.3333 | 161.0000 | 53.33333 | 75.00000 |
| 4 | Male | Y | 35 | 114.0000 | 172.0000 | 71.00000 | 72.50000 |

lapply() 함수

- 특정 함수를 벡터, 리스트, 데이터 프레임 등에 적용하고 그 결과를 리스트로 반환

```
lapply(
  X, # 벡터, 리스트, 표현식, 또는 데이터 프레임
  FUN, # 적용할 함수
)
```

-예시: abalone 데이터를 사용해 일변량 회귀분석 실시

```
# abalone 데이터를 이용해 단순회귀분석 결과 출력
# 종속변수: rings
# 설명변수: 성별을 제외한 모든 연속형 변수

# lm() 함수를 이용한 일변량 회귀분석 실시
univ_reg <- lapply(abalone[,-c(1, 9)], function(x) lm(abalone$rings ~ x))
# univ_reg

# 위 객체로부터 회귀모형 요약 통계량 결과
summ_reg <- lapply(univ_reg, summary)
# summ_reg

# 추정 회귀계수를 데이터로 저장
## summary(lm_object)의 속성 파악
str(summ_reg[[1]])
```

```
List of 11
$ call      : language lm(formula = abalone$rings ~ x)
$ terms     :Classes 'terms', 'formula' language abalone$rings ~ x
... . - attr(*, "variables")= language list(abalone$rings, x)
... . - attr(*, "factors")= int [1:2, 1] 0 1
... . . - attr(*, "dimnames")=List of 2
... . . . $ : chr [1:2] "abalone$rings" "x"
... . . . $ : chr "x"
... . - attr(*, "term.labels")= chr "x"
... . - attr(*, "order")= int 1
... . - attr(*, "intercept")= int 1
... . - attr(*, "response")= int 1
... . - attr(*, ".Environment")=<environment: 0x5614619a0ec8>
... . - attr(*, "predvars")= language list(abalone$rings, x)
```

```

... .- attr(*, "dataClasses")= Named chr [1:2] "numeric" "numeric"
... . .- attr(*, "names")= chr [1:2] "abalone$rings" "x"
$ residuals      : Named num [1:4177] 6.0975 -0.3331 -1.0235 1.3217 -0.0342 ...
..- attr(*, "names")= chr [1:4177] "1" "2" "3" "4" ...
$ coefficients   : num [1:2, 1:4] 2.102 14.946 0.186 0.345 11.328 ...
..- attr(*, "dimnames")=List of 2
... .$. : chr [1:2] "(Intercept)" "x"
... .$. : chr [1:4] "Estimate" "Std. Error" "t value" "Pr(>|t|)"
$ aliased        : Named logi [1:2] FALSE FALSE
..- attr(*, "names")= chr [1:2] "(Intercept)" "x"
$ sigma          : num 2.68
$ df             : int [1:3] 2 4175 2
$ r.squared       : num 0.31
$ adj.r.squared: num 0.31
$ fstatistic     : Named num [1:3] 1875 1 4175
..- attr(*, "names")= chr [1:3] "value" "numdf" "dendf"
$ cov.unscaled  : num [1:2, 1:2] 0.0048 -0.0087 -0.0087 0.0166
..- attr(*, "dimnames")=List of 2
... .$. : chr [1:2] "(Intercept)" "x"
... .$. : chr [1:2] "(Intercept)" "x"
- attr(*, "class")= chr "summary.lm"

```

```
summ_reg[[1]]$coefficients
```

| | Estimate | Std. Error | t value | Pr(> t) |
|-------------|-----------|------------|----------|--------------|
| (Intercept) | 2.101883 | 0.1855477 | 11.32800 | 2.544353e-29 |
| x | 14.946411 | 0.3451570 | 43.30323 | 0.000000e+00 |

```
# summ_reg[[1]]$coefficients 를 추출
res <- lapply(summ_reg, function(lst) lst$coefficients)
res
```

```
$length
      Estimate Std. Error t value Pr(>|t|)
(Intercept) 2.101883  0.1855477 11.32800 2.544353e-29
x          14.946411  0.3451570 43.30323 0.000000e+00

$diameter
      Estimate Std. Error t value Pr(>|t|)
(Intercept) 2.318574  0.1727366 13.42260 3.01241e-40
x          18.669921  0.4114954 45.37091 0.000000e+00

$height
      Estimate Std. Error t value Pr(>|t|)
(Intercept) 3.938464  0.1442530 27.30248 3.678478e-151
x          42.971441  0.9904086 43.38759 0.000000e+00

$whole.wt
      Estimate Std. Error t value Pr(>|t|)
(Intercept) 6.989239  0.08244313 84.77648 0.000000e+00
x          3.552909  0.08561680 41.49780 1.888678e-315

$shucked.wt
      Estimate Std. Error t value Pr(>|t|)
(Intercept) 7.736643  0.0861330 89.82206 0.000000e+00
x          6.113633  0.2039254 29.97976 5.087464e-179

$viscera.wt
      Estimate Std. Error t value Pr(>|t|)
(Intercept) 7.257427  0.08306853 87.36674 0.000000e+00
x          14.819227  0.39322428 37.68645 8.574726e-268

$shell.wt
      Estimate Std. Error t value Pr(>|t|)
(Intercept) 6.462117  0.07714642 83.76431      0
```

```
x      14.535675 0.27908233 52.08382      0
```

```
# res 결과를 2차원 배열 형태로 변환
# do.call() 함수 사용
# 리스트로 주어진 인자에 함수를 적용하여 결과 반환
res <- do.call(rbind, res)
res
```

| | Estimate | Std. Error | t value | Pr(> t) |
|-------------|-----------|------------|----------|---------------|
| (Intercept) | 2.101883 | 0.18554766 | 11.32800 | 2.544353e-29 |
| x | 14.946411 | 0.34515697 | 43.30323 | 0.000000e+00 |
| (Intercept) | 2.318574 | 0.17273658 | 13.42260 | 3.012410e-40 |
| x | 18.669921 | 0.41149538 | 45.37091 | 0.000000e+00 |
| (Intercept) | 3.938464 | 0.14425297 | 27.30248 | 3.678478e-151 |
| x | 42.971441 | 0.99040860 | 43.38759 | 0.000000e+00 |
| (Intercept) | 6.989239 | 0.08244313 | 84.77648 | 0.000000e+00 |
| x | 3.552909 | 0.08561680 | 41.49780 | 1.888678e-315 |
| (Intercept) | 7.736643 | 0.08613300 | 89.82206 | 0.000000e+00 |
| x | 6.113633 | 0.20392536 | 29.97976 | 5.087464e-179 |
| (Intercept) | 7.257427 | 0.08306853 | 87.36674 | 0.000000e+00 |
| x | 14.819227 | 0.39322428 | 37.68645 | 8.574726e-268 |
| (Intercept) | 6.462117 | 0.07714642 | 83.76431 | 0.000000e+00 |
| x | 14.535675 | 0.27908233 | 52.08382 | 0.000000e+00 |

sapply() 함수

- lapply() 함수와 유사하나(lapply()의 wrapper 함수), 결과를 벡터 또는 행렬로 반환하는 점에서 차이를 보임
- 예시: 회귀분석

```
# 각 변수별 회귀계수(절편항과 설명변수) 반환
univ_reg2 <- sapply(abalone[,-c(1, 9)], function(x) {
  coef(lm(abalone$rings ~ x)) # lm 클래스에서 회귀계수 반환
})
univ_reg2
```

| | length | diameter | height | whole.wt | shucked.wt | viscera.wt |
|-------------|-----------|-----------|-----------|----------|------------|------------|
| (Intercept) | 2.101883 | 2.318574 | 3.938464 | 6.989239 | 7.736643 | 7.257427 |
| x | 14.946411 | 18.669921 | 42.971441 | 3.552909 | 6.113633 | 14.819227 |
| | shell.wt | | | | | |
| (Intercept) | 6.462117 | | | | | |
| x | 14.535675 | | | | | |

```
attributes(univ_reg2) # 행렬 반환
```

```
$dim
```

```
[1] 2 7
```

```
$dimnames
```

```
$dimnames[[1]]
```

```
[1] "(Intercept)" "x"
```

```
$dimnames[[2]]
```

```
[1] "length"      "diameter"    "height"      "whole.wt"    "shucked.wt"
```

```
[6] "viscera.wt" "shell.wt"
```

```
# as.data.frame(univ_reg2)
```

mapply() 함수

- sapply()와 유사하지만 다수의 인수를 함수에 전달해 적용
- 임의의 함수 FUN()이 있고, FUN()을 수행하기 위해 필요한 인수가 데이터로 저장되어 있을 때 이를 불러들여 함수를 적용

```
mapply(  
  FUN, # 적용할 함수  
  ..., # 적용할 인수  
)
```

- 난수 생성 예시: rnorm() 함수 사용
- rnorm(n, mean, sd): 평균이 mean이고 표준편차가 sd인 정규분포에서 n개의 난수 생성

```
# 평균이 각각 0, 1, 2, 4이고  
# 표준편차가 1, 1, 1, 1인 정규난수를  
# 각각 20, 40, 60, 100 개 생성  
  
rn_res <- mapply(rnorm,  
  c(20, 40, 60, 100),  
  c(0:2, 4),  
  rep(1, 4))  
  
# rn_res  
  
# 생성한 난수의 평균과 표준편차 확인  
sapply(rn_res, mean); sapply(rn_res, sd)
```

```
[1] 0.1277941 0.8607565 1.9070456 3.9012846
```

```
[1] 0.8954115 0.7916029 0.9990486 0.9088889
```

2.9 객체의 유형 판별 및 변환

지금까지 R 객체를 알아보면서 `is.na()`, `is.null()` 등 스칼라의 데이터 타입을 확인하는 함수부터 `str()`, `attributes()`, `class()`와 같이 객체의 속성 및 구조에 대해 확인하는 함수들에 대해 간략히 소개함.

R은 스크립트 언어이기 때문에 모든 명령 실행이 함수 기반으로 이루어짐. 특정 객체에만 적용할 수 있는 함수들이 있는 반면, 함수를 통해 새로운 속성을 갖는 객체가 생성되기도 함. 그렇기 때문에 함수 적용 또는 반환 후 생성된 객체의 타입을 확인하거나 객체의 유형을 변환하는 작업은 R에서 데이터 분석을 진행하는 과정에서 빈번하게 발생함.

객체 유형 판별을 위해 `is.type_name()`, 객체 타입 변환을 위해 `as.type_name()` 형태의 함수를 제공함. 지금까지 배운 R 객체에 대한 `is.`과 `as.` 계열 함수는 아래와 같음.

TABLE 2.7: R 객체 타입 판별 및 변환 함수

| is 계열 함수 | as 계열 함수 | 설명 |
|------------------------------|------------------------------|----------------------------|
| <code>is.factor()</code> | <code>as.factor()</code> | 주어진 객체가 factor 형인지 판단/변환 |
| <code>is.ordered()</code> | <code>as.ordered</code> | 주어진 객체가 순서형 factor인지 판단/변환 |
| <code>is.numeric()</code> | <code>as.numeric()</code> | 주어진 객체가 수치형인지 판단/변환 |
| <code>is.character()</code> | <code>as.character()</code> | 주어진 객체가 문자형인지 판단/변환 |
| <code>is.matrix()</code> | <code>as.matrix()</code> | 주어진 객체가 행렬인지 판단/변환 |
| <code>is.array()</code> | <code>as.array()</code> | 주어진 객체가 배열인지 판단/변환 |
| <code>is.list()</code> | <code>as.list()</code> | 주어진 객체가 리스트인지 판단/변환 |
| <code>is.data.frame()</code> | <code>as.data.frame()</code> | 주어진 객체가 데이터 프레임인지 판단/변환 |

- is/as계열 함수 사용 예시

```
x <- c("M", "F"); f <- factor(x)
# x가 문자열인가?
is.character(x)
```

[1] TRUE

```
# f가 factor인가?
is.factor(f)
```

```
[1] TRUE
```

```
# f가 숫자형인가?  
is.numeric(f)
```

```
[1] FALSE
```

```
# f를 수치형으로 변환  
f <- as.numeric(f)  
is.numeric(f)
```

```
[1] TRUE
```

```
f
```

```
[1] 2 1
```

```
# 다시 f를 factor형으로 변환  
as.factor(f)
```

```
[1] 2 1  
Levels: 1 2
```

- 2차원 데이터 객체 유형 판별 및 변환

```
X <- matrix(rnorm(9), 3)
d <- data.frame(group = rep(LETTERS[1:3], each = 2),
                 meas = c(mapply(rnorm,
                                 c(2, 2, 2),
                                 c(1, 2, 3),
                                 c(1, 1, 1))))
```

객체 유형 확인
is.matrix(X); is.data.frame(X)

[1] TRUE

[1] FALSE

```
is.matrix(d); is.data.frame(d)
```

[1] FALSE

[1] TRUE

```
# 객체 유형 변환
as.data.frame(X); as.matrix(d)
```

| | V1 | V2 | V3 |
|---|------------|-------------|------------|
| 1 | 0.3288415 | -0.02180829 | -0.3734678 |
| 2 | 0.2479955 | 1.35400192 | -2.2851782 |
| 3 | -0.7521625 | -0.15786514 | -1.1439634 |

```
group meas
[1,] "A"   "2.028870"
[2,] "A"   "1.996936"
[3,] "B"   "2.835169"
[4,] "B"   "1.105901"
[5,] "C"   "1.095014"
[6,] "C"   "2.927883"
```

```
as.list(X); as.list(d)
```

```
[[1]]
[1] 0.3288415
```

```
[[2]]
[1] 0.2479955
```

```
[[3]]
[1] -0.7521625
```

```
[[4]]
[1] -0.02180829
```

```
[[5]]
[1] 1.354002
```

```
[[6]]
[1] -0.1578651
```

```
[[7]]
[1] -0.3734678
```

```
[[8]]
```

```
[1] -2.285178
```

```
[[9]]
```

```
[1] -1.143963
```

```
$group
```

```
[1] "A" "A" "B" "B" "C" "C"
```

```
$meas
```

```
[1] 2.028870 1.996936 2.835169 1.105901 1.095014 2.927883
```



3

문자열 처리와 정규표현식



학습 목표

- 텍스트 문자 처리에 있어 가장 기본인 정규 표현식(regular reexpression)에 대해 알아본다.
- R에서 기본으로 제공하는 문자열 처리 함수에 대해 알아본다

학습 필요성

- 실제 데이터는 다양한 형태의 텍스트(문자열)을 포함
- R에서 문자열을 이용한 반복 계산 가능
- 대규모 텍스트 데이터(웹문서, 블로그, SNS, 뉴스, 논문, 상품평, ...)로부터 새로운 정보 및 지식을 도출하기 위한 텍스트 처리에 대한 기본적 이해
- 여러 문자열로 이루어진 방대한 텍스트 벡터에서 특정 패턴을 갖고 있는 구문을 선별해야 할 경우, 패턴을 도식화 할 수 있는 함축적 표현 필요 → **정규 표현식**

정규 표현식의 기본함수

- **grep()**, **grep1()**: 문자형 벡터에서 정규 표현식 또는 문자 패턴의 일치를 검색.
 - **grep()**: 일치하는 특정 문자열을 포함하는 문자형 벡터 또는 인덱스를 반환
 - **grep1()**: 문자열 포함 여부에 대한 논리값 반환
- **regexpr()**, **gregexpr()**: 문자형 벡터에서 정규 표현식 또는 문자열 패턴과 일치하는 원소를 검색하고, 일치가 시작되는 문자열의 인덱스와 일치 길이를 반환
- **sub()**, **gsub()**: 문자열 벡터에서 정규 표현식 또는 문자열 패턴과 일치하는 원소를 검색하고 해당 문자열을 다른 문자열로 변경
- **regexec()**: **regexpr()**와 동일하게 일치가 시작되는 문자열의 인덱스를 반환하지만 팔호로 묶인 하위 표현식의 위치를 추가로 반환



정규 표현식 및 문자열 처리를 위한 함수의 종류는 매우 다양하지만, 본 강의에서는 정규 표현식의 이해를 위해 일부만 소개할 것임

문자열 기초

- 탈출 지시자(escape indicator): \
 - 키보드로 입력할 수 없는 문자를 입력하기 위해 사용
 - 문자열에 백슬래쉬 \를 입력하려면 \\로 표시

```
# 문자열에 따옴표(single of double quote, ', ") 입력
double_quote <- "\"
double_quote
```

```
[1] "\\"
```

```
single_quote <- '\'
single_quote
```

```
[1] "\\"
```

```
x <- c("\\"", "\\\\", '\\\\')
writeLines(x)
```

```
"\"
\\
\'
```

```
# 백슬래시가 포함된 문자열
x <- "abc\n\tabc"

# \n: Enter
# \t: tab 문자를 표현

writeLines(x)
```

```
abc
```

```
abc
```

```
# 특수문자 표현
x <- "\u00b5" # 그리스 문자 μu 표현 (유니코드)
x
```

[1] "μ"

참고자료

- Youtube 동영상¹: 영어 강의가 옥외 티...
- regexr.com²: 정규 표현식의 패턴 확인 가능
- Wikibooks R programming: Text processing³

3.1 유용한 문자열 관련 함수

3.1.1 nchar()

- 인간이 눈으로 읽을 수 있는 문자의 개수(길이)를 반환
- 공백, 줄바꿈 표시자(예: \n)도 하나의 문자 개수로 인식
- 한글의 한 글자는 2 바이트(byte)지만 한 글자로 인식 → byte 단위 반환 가능

```
# 문자열을 구성하는 문자 개수 반환
nchar(
  x, # 문자형 벡터
  type # "bytes": 바이트 단위 길이 반환
    # "char": 인간이 읽을 수 있는 글자 길이 반환
```

```
# "width": 문자열이 표현된 폭의 길이 반환  
)
```

- 예시

```
x <- "Carlos Gardel's song: Por Una Cabeza"  
nchar(x)
```

```
[1] 36
```

```
y <- "abcde\nfghij"  
nchar(y)
```

```
[1] 11
```

```
z <- "양준일: 가나다라마바사"  
nchar(z)
```

```
[1] 12
```

```
# 문자열 벡터  
str <- sentences[1:10]  
nchar(str)
```

```
[1] 42 43 38 40 36 37 43 43 35 40
```

```
s <- c("abc", "가나다", "1234[] ", "R programming\n", "\\R\\")  
  
nchar(s, type = "char")
```

```
[1] 3 3 6 14 3
```

```
nchar(s, type = "byte")
```

```
[1] 3 9 6 14 3
```

```
nchar(s, type = "width")
```

```
[1] 3 6 6 14 3
```



벡터의 원소 개수를 반환하는 `length()` 함수와는 다름.

3.1.2 `paste()`, `paste0()`

- 하나 이상의 문자열을 연결하여 하나의 문자열로 만들어주는 함수
- Excel의 문자열 연결자인 &와 거의 동일한 기능을 수행

```
paste(  
  ..., # 한 개 이상의 R 객체. 강제로 문자형 변환  
  sep # 연결 구분자: 디폴트 값은 공백(" ")  
  collapse # 뒤를 객체가 하나의 문자열 벡터인 경우  
           # 모든 원소를 collapse 구분자로 묶은 길이가 1인 벡터 반환  
)
```

- `paste0()`은 `paste()`의 wrapper 함수이고 `paste()`의 구분자 인수 `sep = ""` 일 때와 동일한 결과 반환
- 예시

```
i <- 1:length(letters)

paste(letters, i) # sep = " "
```

```
[1] "a 1"  "b 2"  "c 3"  "d 4"  "e 5"  "f 6"  "g 7"  "h 8"  "i 9"  "j 10"
[11] "k 11" "l 12" "m 13" "n 14" "o 15" "p 16" "q 17" "r 18" "s 19" "t 20"
[21] "u 21" "v 22" "w 23" "x 24" "y 25" "z 26"
```

```
paste(letters, i, sep = "_") # sep = "-"
```

```
[1] "a_1"  "b_2"  "c_3"  "d_4"  "e_5"  "f_6"  "g_7"  "h_8"  "i_9"  "j_10"
[11] "k_11" "l_12" "m_13" "n_14" "o_15" "p_16" "q_17" "r_18" "s_19" "t_20"
[21] "u_21" "v_22" "w_23" "x_24" "y_25" "z_26"
```

```
paste0(letters, i) # paste(letters, i, sep = "") 동일
```

```
[1] "a1"  "b2"  "c3"  "d4"  "e5"  "f6"  "g7"  "h8"  "i9"  "j10" "k11" "l12"
[13] "m13" "n14" "o15" "p16" "q17" "r18" "s19" "t20" "u21" "v22" "w23" "x24"
[25] "y25" "z26"
```

```
# collapse 인수 활용
paste(letters, collapse = "")
```

```
[1] "abcdefghijklmnopqrstuvwxyz"
```

```
writeLines(paste(str, collapse = "\n"))
```

The birch canoe slid on the smooth planks.
 Glue the sheet to the dark blue background.
 It's easy to tell the depth of a well.
 These days a chicken leg is a rare dish.
 Rice is often served in round bowls.
 The juice of lemons makes fine punch.
 The box was thrown beside the parked truck.
 The hogs were fed chopped corn and garbage.
 Four hours of steady work faced us.
 Large size in stockings is hard to sell.

```
# 3가지 이상 객체 묶기
```

```
paste("Col", 1:2, c(TRUE, FALSE, TRUE), sep = " ", collapse = "<->")
```

```
[1] "Col 1 TRUE<->Col 2 FALSE<->Col 1 TRUE"
```

```
# paste 함수 응용
```

```
# 스트링 명령어 실행
```

```
exprs <- paste("lm(mpg ~", names(mtcars)[3:5], ", data = mtcars)")  

exprs
```

```
[1] "lm(mpg ~ disp , data = mtcars)" "lm(mpg ~ hp , data = mtcars)"  

[3] "lm(mpg ~ drat , data = mtcars)"
```

```
sapply(1:length(exprs), function(i) coef(eval(parse(text = exprs[i]))))
```

```
[,1]          [,2]          [,3]  
(Intercept) 29.59985476 30.09886054 -7.524618  
disp         -0.04121512 -0.06822828  7.678233
```

3.1.3 sprintf()

- C 언어의 `sprintf()` 함수와 동일하며 특정 변수들의 값을 이용해 문자열을 반환함
- 수치형 값의 소수점 자리수를 맞추거나 할 때 유용하게 사용
- 포맷팅 문자열을 통해 수치형의 자릿수를 지정 뿐 아니라 전체 문자열의 길이 및 정렬 가능
- 대표적인 포맷팅 문자열은 아래 표와 같음.

| Format | 설명 |
|--------|----------|
| %s | 문자열 |
| %d | 정수형 |
| %f | 부동 소수점 수 |
| %e, %E | 지수형 |

- 예시

```
options()$digits #
```

```
[1] 7
```

```
pi # π의 값
```

```
[1] 3.141593
```

```
sprintf("%f", pi)
```

```
[1] "3.141593"
```

```
# 소수점 자리수 3자리 까지 출력  
sprintf("%.3f", pi)
```

```
[1] "3.142"
```

```
# 소수점 출력 하지 않음  
sprintf("%1.0f", pi)
```

```
[1] "3"
```

```
# 출력 문자열의 길이를 5로 고정 후  
# 소수점 한 자리까지 출력  
sprintf("%5.1f", pi)
```

```
[1] " 3.1"
```

```
nchar(sprintf("%5.1f", pi))
```

```
[1] 5
```

```
# 빈 공백에 0값 대입  
sprintf("%05.1f", pi)
```

```
[1] "003.1"
```

```
# 양수/음수 표현  
sprintf("%+f", pi)
```

```
[1] "+3.141593"
```

```
sprintf("%+f", -pi)
```

```
[1] "-3.141593"
```

```
# 출력 문자열의 첫 번째 값을 공백으로  
sprintf("%_f", pi)
```

```
[1] " 3.141593"
```

```
# 왼쪽 정렬  
sprintf("%-10.3f", pi)
```

```
[1] "3.142      "
```

```
# 수치형에 정수 포맷을 입력하면?  
sprintf("%d", pi)
```

Error in sprintf("%d", pi): '%d'는 유효하지 않은 포맷입니다; 수치형 객체들
에는 포맷 %f, %e, %g 또는 %a를 사용해 주세요

```
 sprintf("%d", 100); sprintf("%d", 20L)
```

```
[1] "100"
```

```
[1] "20"
```

```
# 지수형  
sprintf("%e", pi)
```

```
[1] "3.141593e+00"
```

```
 sprintf("%E", pi)
```

```
[1] "3.141593E+00"
```

```
sprintf("%.2E", pi)
```

```
[1] "3.14E+00"
```

```
# 문자열
sprintf("%s = %.2f", "Mean", pi)
```

```
[1] "Mean = 3.14"
```

```
# 응용
mn <- apply(cars, 2, mean)
std <- apply(cars, 2, sd)

# Mean ± SD 형태로 결과 출력 (소수점 2자리 고정)
res <- sprintf("%.2f \U00B1 %.2f", mn, std)
resp <- paste(paste0(names(cars), ":", res), collapse = "\n")
writeLines(resp)
```

```
speed: 15.40 ± 5.29
dist: 42.98 ± 25.77
```

3.1.4 substr()

- 문자열에서 특정 부분을 추출하는 함수
- 보통 한 문자열이 주어졌을 때 start에서 end 까지 추출

```
substr(
  x, # 문자형 벡터
  start, # 문자열 추출 시작 위치
  stop # 문자열 추출 종료 위치
)
```

- 예시

```
cnu <- "충남대학교 자연과학대학 정보통계학과"
substr(cnu, start = 14, stop = nchar(str))
```

[1] "정보통계학과"

```
# 문자열 벡터에서 각 원소 별 적용
substr(str, 5, 15)
```

```
[1] "birch canoe" " the sheet " " easy to te" "e days a ch" " is often s"
[6] "juice of le" "box was thr" "hogs were f" " hours of s" "e size in s"
```

3.1.5 tolower(), toupper()

- 대문자를 소문자(tolower()) 혹은 소문자를 대문자(toupper())로 변환

```
LETTERS; tolower(LETTERS)
```

```
[1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S"
```

```
[20] "T" "U" "V" "W" "X" "Y" "Z"
```

```
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
```

```
[20] "t" "u" "v" "w" "x" "y" "z"
```

```
letters; toupper(letters)
```

```
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
```

```
[20] "t" "u" "v" "w" "x" "y" "z"
```

```
[1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S"
```

```
[20] "T" "U" "V" "W" "X" "Y" "Z"
```

3.1.6 glue 패키지를 활용한 문자열 다루기



- glue 패키지에서 제공하는 `glue()` 함수를 통해 지금까지 학습한 `paste()`, `paste0()` 보다 손쉽게 문자열을 결합할 수 있음
- 중괄호({}, curly bracket)를 활용해 R 객체에 저장되어 있는 값과 지정한 문자열을 결합

3.1.6.1 glue 패키지 설치

```
# CRAN상 배포된 패키지 설치
install.packages('glue')

# 개발자 버전 설치
install.packages("devtools")
devtools::install_github("tidyverse/glue")
```

3.1.6.2 사용 방법

R 객체를 직접 문자열과 결합 → 문자열 + {R 객체명}

```
library(glue)
```

다음의 패키지를 부착합니다: 'glue'

The following object is masked from 'package:dplyr':

```
collapse
```

```
name <- "Boncho Ku"  
glue("My name is {name}.")
```

My name is Boncho Ku.

긴 문자열과 R 객체를 결합 시 함수 내 ,로 구분 후 결합 가능

```
name <- "Boncho Ku"  
age <- 42  
anniversary <- as.Date("2013-03-24")  
  
glue("My name is {name}, ",  
      "My age next year is {age + 1}, ",  
      "My wedding anniversary is {anniversary}.")
```

My name is Boncho Ku, My age next year is 43, My wedding anniversary is 2013-03-24.

함수 내 임시변수를 활용한 문자열 결합

```
glue("My name is {name_temp}, ",  
      "My age next year is {age_temp + 1}, ",  
      "My wedding anniversary is {anniversary_temp}.",  
      name_temp = "Boncho Ku",  
      age_temp = 42,  
      anniversary_temp = as.Date("2013-03-24"))
```

My name is Boncho Ku, My age next year is 43, My wedding anniversary is 2013-03-24.

데이터 프레임 변수에 일괄 적용 가능

```
dat <- head(iris)  
glue("The {dat$Species} has a sepal length {dat$Sepal.Length}")
```

```
The setosa has a sepal length 5.1  
The setosa has a sepal length 4.9  
The setosa has a sepal length 4.7  
The setosa has a sepal length 4.6  
The setosa has a sepal length 5  
The setosa has a sepal length 5.4
```

단순히 새로 줄바꿈(Enter)으로 두 줄 이상 문자열을 생성 가능

```
glue("The first line  
      The second line  
      The third line")
```

```
The first line
The second line
The third line
```

문장 끝에 \\를 붙이면 줄바꿈이 실행되지 않음

```
glue("The first line, \\
      The second line, \\
      The third line")
```

```
The first line, The second line, The third line
```

이중 중괄호({{}}): {R 객체명}을 그대로 출력

```
name <- "Boncho Ku"
glue("My name is {name}, not {{name}}.")
```

```
My name is Boncho Ku, not {name}.
```

.open 및 .close를 사용해 대체 구분기호 지정 가능

```
one <- 1
glue("The value of $\int (\sqrt{2\pi}\sigma)^{-1} \cdot \\
      \exp[-(x - \mu)^2/2\sigma^2] dx$",
      " is $<<one>>$", .open = "<<", .close = ">>")
```

```
The value of $\int (\sqrt{2\pi}\sigma)^{-1} \cdot \exp[-(x - \mu)^2/2\sigma^2] dx$ is $1$
```

위 문자열을 Rmarkdown 문서에 수식으로 표현하고 싶다면 inline R code chunk 사용

```
r   glue("The value of $\int (\sqrt{2\pi})^\sigma^{-1} \cdot
" \exp[-(x - \mu)^2/2\sigma^2] dx$", " is $<<one>>$",
.open = "<<", .close = ">>")`
```

The value of $\int (\sqrt{2\pi}\sigma)^{-1} \exp[-(x - \mu)^2/2\sigma^2]dx$ is 1

3.2 정규표현식 기본 함수

3.2.1 grep(), grepl()

정규표현식을 이용한 특정 문자 패턴 검색 시 가장 빈번히 사용되는 함수들 중 하나임.

grep()

특정 문자 벡터에서 찾고자 하는 패턴과 일치하는 원소의 인덱스, 원소값 반환

```
# 일치하는 특정 문자열을 포함하는 원소값(문자형) 또는 인덱스(정수)를 반환

grep(
  pattern, # 정규 표현식 또는 문자 패턴
  string, # 패턴을 검색할 문자열 벡터
  value    # 논리값
  # TRUE: pattern에 해당하는 원소값 반환
```

```
# FALSE: pattern 0/ 있는 원소의 색인 반환  
)
```

```
x <- c("Equator", "North Pole", "South Pole")  
  
# x에서 Pole 0/ 있는 원소의 문자열 반환  
grep("Pole", x, value = T)
```

```
[1] "North Pole" "South Pole"
```

```
# x에서 Pole 0/ 있는 원소의 색인 반환  
grep("Pole", x, value = F)
```

```
[1] 2 3
```

```
# x에서 Eq를 포함한 원소 색인 반환  
grep("Eq", x)
```

```
[1] 1
```

```
grep1()
```

grep()과 유사한 기능을 갖지만, 함수의 반환값이 논리형 벡터임

```
# 일치하는 특정 문자열을 포함하는 원소 색인에 대한 논리값 반환

grepel(
  pattern, # 정규 표현식 또는 문자 패턴
  string   # 패턴을 검색할 문자열 벡터
)
```

- 사용 예시

```
# grepel() 예시

# Titanic data 불러오기
url1 <- "https://raw.githubusercontent.com/"
url2 <- "agconti/kaggle-titanic/master/data/train.csv"
titanic <- read.csv(paste0(url1, url2),
                     stringsAsFactors = FALSE)

# 승객이름 추출
pname <- titanic$Name

# 승객 이름이 James 인 사람만 추출
g <- grepel("James", pname)
pname[g]
```

```
[1] "Moran, Mr. James"
[2] "Crease, Mr. Ernest James"
[3] "Sobey, Mr. Samuel James Hayden"
[4] "Bateman, Rev. Robert James"
[5] "Watt, Mrs. James (Elizabeth \"Bessie\" Inglis Milne)"
[6] "Smith, Mr. James Clinch"
[7] "Brown, Mrs. James Joseph (Margaret Tobin)"
```

- [8] "Bracken, Mr. James H"
- [9] "Reed, Mr. James George"
- [10] "Baxter, Mrs. James (Helene DeLaudeniere Chaput)"
- [11] "Drew, Mrs. James Vivian (Lulu Thorne Christian)"
- [12] "Flynn, Mr. James"
- [13] "Scanlan, Mr. James"
- [14] "Webber, Mr. James"
- [15] "McGough, Mr. James Robert"
- [16] "Farrell, Mr. James"
- [17] "Sharp, Mr. Percival James R"
- [18] "Downton, Mr. William James"
- [19] "Elsbury, Mr. William James"
- [20] "Kelly, Mr. James"
- [21] "Hawksford, Mr. Walter James"
- [22] "Lester, Mr. James"
- [23] "Slemen, Mr. Richard James"
- [24] "Banfield, Mr. Frederick James"

3.2.2 `regexpr()`, `gregexpr()`

`grep()`과 `grep1()`의 한계점 보완: 특정 문자 패턴의 일치여부에 대한 정보를 제공하지만 위치 및 정규식의 일치 여부를 알려주지는 않음

`regexpr()`

- 문자열에서 패턴이 일치하는 문자(표현)가 첫 번째 등장하는 위치와 몇 개의 문자로 구성(길이) 되어 있는지를 반환
- 예시

```
x <- c("Darth Vader: If you only knew the power of the Dark Side.  
      Obi-Wan never told you what happend to your father",  
      "Luke: He told me enough! It was you who killed him!",  
      "Darth Vader: No. I'm your father")  
  
# grep 계열 함수  
grep("you", x); grepl("you", x)
```

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

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

```
# regexpr()  
regexpr("you", x) # 각 x의 문자열에서 you가 처음 나타난 위치 및 길이 반환
```

```
[1] 17 33 22  
attr("match.length")  
[1] 3 3 3  
attr("index.type")  
[1] "chars"  
attr("useBytes")  
[1] TRUE
```

```
regexpr("father", x) # 패턴을 포함하지 않은 경우 -1 반환
```

```
[1] 111 -1 27  
attr("match.length")  
[1] 6 -1 6  
attr("index.type")
```

```
[1] "chars"
attr(,"useBytes")
[1] TRUE
```

- `substr()` 함수와 `regexpr()` 함수를 이용해 텍스트 내 원하는 문자 추출 가능

```
idx <- regexpr("father", x)
substr(x, idx, idx + attr(idx, "match.length") - 1)
```

```
[1] "father" "" "father"
```

gregexpr()

- 영역에 걸쳐 패턴과 일치하는 문자의 위치 및 길이 반환(`regexpr()`의 global 버전)

```
gregexpr("you", x) # 각 x의 문자열에서 you가 나타난 모든 위치 및 길이 반환
```

```
[[1]]
[1] 17 86 106
attr(,"match.length")
[1] 3 3 3
attr(,"index.type")
[1] "chars"
attr(,"useBytes")
[1] TRUE
```

```
[[2]]
```

```
[1] 33  
attr("match.length")  
[1] 3  
attr("index.type")  
[1] "chars"  
attr("useBytes")  
[1] TRUE  
  
[[3]]  
[1] 22  
attr("match.length")  
[1] 3  
attr("index.type")  
[1] "chars"  
attr("useBytes")  
[1] TRUE
```

```
gregexpr("father", x) # 패턴을 포함하지 않은 경우 -1 반환
```

```
[[1]]  
[1] 111  
attr("match.length")  
[1] 6  
attr("index.type")  
[1] "chars"  
attr("useBytes")  
[1] TRUE
```

```
[[2]]  
[1] -1  
attr("match.length")  
[1] -1
```

```
attr("index.type")
[1] "chars"
attr("useBytes")
[1] TRUE

[[3]]
[1] 27
attr("match.length")
[1] 6
attr("index.type")
[1] "chars"
attr("useBytes")
[1] TRUE
```

3.2.3 sub(), gsub()

- 검색하고자 하는 패턴을 원하는 문자로 변경
- 문자열 벡터의 패턴을 일치시키거나 문자열 정리가 필요할 때 사용

sub()

- 문자열에서 첫 번째 일치하는 패턴만 변경

```
sub(pattern, # 검색하고자 하는 문자, 패턴, 표현
      replacement, # 검색할 패턴 대신 변경하고자 하는 문자 및 표현
      x # 문자형 벡터
      )
```

- 예시

```
jude <- c("Hey Jude, don't make it bad",
         "Take a sad song and make it better",
         "Remember to let her into your heart",
         "Then you can start to make it better")

sub("a", "X", jude)
```

```
[1] "Hey Jude, don't mXke it bad"
[2] "TXke a sad song and make it better"
[3] "Remember to let her into your heXrt"
[4] "Then you cXn start to make it better"
```

```
gsub()
```

- 문자열에서 일치하는 모든 패턴 변경
- 예시

```
sub(" ", "_", jude)
```

```
[1] "Hey_Jude, don't make it bad"
[2] "Take_a sad song and make it better"
[3] "Remember_to let her into your heart"
[4] "Then_you can start to make it better"
```

```
gsub(" ", "_", jude)
```

```
[1] "Hey_Jude,_don't_make_it_bad"
[2] "Take_a_sad_song_and_make_it_better"
[3] "Remember_to_let_her_into_your_heart"
[4] "Then_you_can_start_to_make_it_better"
```

```
gsub("a", "X", jude)
```

```
[1] "Hey Jude, don't mXke it bXd"
[2] "TXke X sXd song Xnd mXke it better"
[3] "Remember to let her into your heXrt"
[4] "Then you cXn stXrt to mXke it better"
```

3.2.4 regexexec()

`regexpr()`과 유사하게 작동하지만 괄호(`()`)로 묶인 하위 표현식에 대한 인덱스를 제공

- (): 정규 표현식의 메타 문자 중 하나로 그룹을 나타냄 → 정규표현식 내 논리적 테스트 수행 가능

```
bla <- c("I like statistics",
       "I like R programming",
       "I like bananas",
       "Estates and statues are too expensive")

grep("like", bla)
```

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

```
grep("are", bla)
```

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

```
grep("(like|are)", bla)
```

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

- 찾고자 하는 패턴을 두 그룹으로 나눌 때 유용
- 예시

```
gregexpr("stat", bla)
```

```
[[1]]  
[1] 8  
attr(),"match.length")  
[1] 4  
attr(),"index.type")  
[1] "chars"  
attr(),"useBytes")  
[1] TRUE
```

```
[[2]]  
[1] -1  
attr(),"match.length")  
[1] -1  
attr(),"index.type")  
[1] "chars"  
attr(),"useBytes")  
[1] TRUE
```

```
[[3]]  
[1] -1  
attr(),"match.length")
```

```
[1] -1  
attr("index.type")  
[1] "chars"  
attr("useBytes")  
[1] TRUE
```

```
[[4]]  
[1] 2 13  
attr("match.length")  
[1] 4 4  
attr("index.type")  
[1] "chars"  
attr("useBytes")  
[1] TRUE
```

```
gregexpr("(st)(at)", bla)
```

```
[[1]]  
[1] 8  
attr("match.length")  
[1] 4  
attr("index.type")  
[1] "chars"  
attr("useBytes")  
[1] TRUE
```

```
[[2]]  
[1] -1  
attr("match.length")  
[1] -1  
attr("index.type")  
[1] "chars"
```

```
attr("useBytes")
[1] TRUE

[[3]]
[1] -1
attr("match.length")
[1] -1
attr("index.type")
[1] "chars"
attr("useBytes")
[1] TRUE

[[4]]
[1] 2 13
attr("match.length")
[1] 4 4
attr("index.type")
[1] "chars"
attr("useBytes")
[1] TRUE
```

```
# "at"에 대한 패턴을 찾지 못하고
# "stat" 패턴과 과 동일한 결과 반환

regexec("(st)(at)", bla)
```

```
[[1]]
[1] 8 8 10
attr("match.length")
[1] 4 2 2
attr("index.type")
[1] "chars"
```

```
attr("useBytes")
[1] TRUE

[[2]]
[1] -1
attr("match.length")
[1] -1
attr("index.type")
[1] "chars"
attr("useBytes")
[1] TRUE

[[3]]
[1] -1
attr("match.length")
[1] -1
attr("index.type")
[1] "chars"
attr("useBytes")
[1] TRUE

[[4]]
[1] 2 2 4
attr("match.length")
[1] 4 2 2
attr("index.type")
[1] "chars"
attr("useBytes")
[1] TRUE
```

```
# "stat" 패턴도 동시에 반환됨을 유의  
# 첫 번째 일치 패턴만 반환
```

3.2.5 strsplit()

- 문자열에서 매칭되는 특정 패턴(문자)을 기준으로 문자열을 분할함

```
strsplit(  
  x,      # 문자형 벡터  
  split # 분할 구분 문자(정규표현식 포함)  
)
```

- 예시

```
jude_w1 <- strsplit(jude, " ")  
jude_w1
```

```
[[1]]
```

```
[1] "Hey"    "Jude," "don't" "make"   "it"     "bad"
```

```
[[2]]
```

```
[1] "Take"   "a"      "sad"    "song"   "and"    "make"   "it"     "better"
```

```
[[3]]
```

```
[1] "Remember" "to"      "let"    "her"    "into"   "your"   "heart"
```

```
[[4]]
```

```
[1] "Then"    "you"    "can"    "start"  "to"     "make"   "it"     "better"
```

```
# 공백, 쉼표가 있는 경우 구분
jude_w2 <- strsplit(jude, "(\s|,)")
```

```
jude_w2
```

```
[[1]]
```

```
[1] "Hey"    "Jude"   ""        "don't"  "make"   "it"     "bad"
```

```
[[2]]
```

```
[1] "Take"   "a"      "sad"    "song"   "and"    "make"   "it"     "better"
```

```
[[3]]
```

```
[1] "Remember" "to"     "let"    "her"    "into"   "your"   "heart"
```

```
[[4]]
```

```
[1] "Then"   "you"   "can"   "start"  "to"     "make"   "it"     "better"
```

3.3 정규 표현식(regular expression)

- 주어진 문자열에 특정한 패턴이 있는 경우, 해당 패턴을 일반화(수식화)한 문자열
- 특정 패턴을 표현한 문자열을 메타 문자(meta character)라고 지칭
- 일반적으로 특정 규칙 또는 패턴이 문자열을 찾고(to find), 해당 규칙에 해당하는 문자열을 대체(replace, substitute)하기 위해 사용
- R 언어 뿐 아니라 타 프로그래밍 언어(C, Perl, Python 등) 워드 프로세서, 텍스트 편집기, 검색 엔진, 운영체제(Windows, Linux 등)에서도 범용적으로 사용

- 정규식이라고도 불리우며 영어로는 regex 또는 regexp로 명칭됨

3.3.1 기본 메타 문자

TABLE 3.1: 정규표현식 메타 문자: 기본

| Expression | Name | 설명 |
|------------|---------------|--|
| \. | Period | 무엇이든 한 글자를 의미 (마침표) |
| \+ | Plus | \+ 앞에 오는 표현이 하나 이상 포함 |
| * | Asterisk | * 앞에 오는 표현이 0 또는 하나 이상 포함 |
| ? | Question mark | ? 앞에 오는 표현이 0 또는 하나 포함 |
| ^ | Caret | ^ 뒤에 오는 표현으로 시작 |
| \$ | Dollar | \$ 앞에 오는 표연으로 끝나는 경우 |
| {} | Curly bracket | { } 앞에 정확히 {}에 있는 숫자만큼 반복되는 패턴 (예시 참고) |
| () | Parenthesis | () 정규 표현식 내 하위(그룹) 표현식 (예시 참고) |
| | Vertical bar | 의 왼쪽 또는 오른쪽 표현이 존재하는지 |

- 메타 문자를 메타 문자가 아닌 문자 자체로 인식하기 위해서는 해당 문자 앞에 \\를 붙임

```
# 마침표가 있는 위치 반환  
str2 <- str[1:2]  
regexp(".", str2)
```

```
[1] 1 1  
attr("match.length")  
[1] 1 1  
attr("index.type")  
[1] "chars"  
attr("useBytes")  
[1] TRUE
```

```
# 에러 출력  
regexp("\.", str2)
```

Error: ""\."로 시작하는 문자열 중에서 '\.'는 인식할 수 없는 이스케이프입니다

```
# 정확한 표현  
regexp("\\.", str2)
```

```
[1] 42 43  
attr("match.length")  
[1] 1 1  
attr("index.type")  
[1] "chars"  
attr("useBytes")  
[1] TRUE
```

- . 마침표(period)

- 어떤 임의의 한 문자를 의미

```
# 문자열 자체가 존재하니까 참값 반환  
grep1(".", jude)
```

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

```
grep1(".", "#@%{@FDSAGF$%")
```

```
[1] TRUE
```

```
# 문자없음 ""  
grep1(".", "")
```

```
[1] FALSE
```

```
# a로 시작하고 중간에 어떤 글자가 하나 존재하고 b로 끝나는 패턴  
bla2 <- c("aac", "aab", "accb", "acadb")  
g <- grep1("a.b", bla2)  
bla2[g]
```

```
[1] "aab"    "acadb"
```

```
# a와 b 사이 어떤 두 문자 존재하는 패턴
g <- grep1("a..b", bla2)
bla2[g]
```

```
[1] "accb"
```

+ (plus)

- +에 선행된 패턴이 한 번 이상 매칭 → + 앞에 문자를 1개 이상 포함

```
# "a"를 적어도 하나 이상 포함한 원소 반환
grep1("a+", c("ab", "aa", "aab", "aaab", "b"))
```

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

```
# "l"과 "n" 사이에 "o"가 하나 이상인 원소 반환
grep1("lo+n", c("bloon", "blno", "leno", "lnooon", "lololon"))
```

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

* (asterisk)

- * 앞에 선행된 문자 또는 패턴이 0번 이상 매치 → * 앞에 문자를 0개 또는 1개 이상 포함

```
# xx가 "a"를 0 또는 1개 이상 포함하고 있는가?
xx <- c("bbb", "acb", "def", "cde", "zde", "era", "xsery")
# "a" 존재와 상관 없이 모든 문자열이 조건에 부합
g <- grep1("a*", xx)
xx[g]
```

```
[1] "bbb"    "acb"    "def"    "cde"    "zde"    "era"    "xsery"
```

```
# "aab"와 "c" 사이에 "d"가 없거나 하나 이상인 경우
# "caabec"인 경우 "aab"와 "c" 사이에 "e"가 존재하기 때문에 FALSE
grep1("aabd*c", c("aabddc", "caabec", "aabc"))
```

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

? (question)

- ? 앞에 항목은 선택 사항이며 많아야 한 번 매치 → ? 앞의 문자를 0 개 또는 1개 포함

```
xx <- c("ac", "abbc", "abc", "abcd", "abbdc")
g <- grep1("ab?c", xx) ## "a"와 "c" 사이에 "b"가 0개 또는 1개 포함
xx[g]
```

```
[1] "ac"    "abc"   "abcd"
```

```
yy <- c("aabc", "aabbc", "daabec", "aabbcc", "aabbbbc")
g <- grep1("aabb?c", yy) ## "aab"와 "c" 사이에 "b"가 0개 또는 1개 있는 경우 일치
yy[g]
```

```
[1] "aabc"  "aabbc"
```

[^] (caret)

- [^] 뒤에 나오는 문자(열)로 시작하는 문자열 반환

```
# str0:// "The"로 시작하는 문자열 반환
g <- grep1("^The", str)
str[g]
```

```
[1] "The birch canoe slid on the smooth planks."
[2] "These days a chicken leg is a rare dish."
[3] "The juice of lemons makes fine punch."
[4] "The box was thrown beside the parked truck."
[5] "The hogs were fed chopped corn and garbage."
```

- [^]: 대괄호(straight bracket) 안에 첫 번째 문자가 ^인 경우 ^뒤에 있는 문자들을 제외한 모든 문자와 매치

```
xx <- c("abc", "def", "xyz", "werx", "wbcsp", "cba")
# "a", "b", "c"를 순서 상관 없이 동시에 포함하지 않은 문자열 반환
g <- grep1("[^abc]", xx)
xx[g]
```

```
[1] "def"  "xyz"  "werx"  "wbcsp"
```

- ^ []: [] 안에 들어간 문자 중 어느 한 단어로 시작하는 문자열 반환

```
xx <- c("def", "wasp", "sepcial", "statisitc", "abbey load", "cross", "batman")
g <- grep1("^[abc]", xx)
xx[g]
```

```
[1] "abbey load" "cross"      "batman"
```

\$ (dollar)

- \$ 앞에 나오는 문자 및 패턴과 문자열의 맨 마지막 문자 패턴과 매치

```
g <- grep1("father$", x)
writeLines(x[g])
```

Darth Vader: If you only knew the power of the Dark Side.

Obi-Wan never told you what happend to your father

Darth Vader: No. I'm your father

{ } (curly bracket)

- {} 앞의 문자 패턴이 {} 안에 숫자만큼 반복되는 패턴을 매치
 - {n}: 정확히 n 번 매치
 - {n,m}: n 번에서 m 번 매치
 - {n, }: 적어도 n 번 이상 매치

```
xx <- c("tango", "jazz", "swing jazz", "hip hop",
       "groove", "rock'n roll", "heavy metal")

# "z"가 정확히 2번 반복되는 원소 반환
g <- grep1("z{2}", xx)
xx[g]
```

```
[1] "jazz"      "swing jazz"
```

```
# "e"가 2번 이상 반복되는 원소 반환
yy <- c("deer", "abacd", "abcd", "daaeb", "eel", "greeeeg")
g <- grep1("e{2,}", yy)
xx[g]
```

```
[1] "tango"      "groove"      "rock'n roll" "heavy metal"
```

```
# "b"가 2번 이상 4번 이하 반복되고 앞에 "a"가 있는 원소 반환
zz <- c("ababababab", "abbb", "cbbe", "xabbabbc")
g <- grep1("ab{2,4}", zz)
zz[g]
```

```
[1] "abbb"      "xabbabbc"
```



참고: 위에서 소개한 메타 문자 중 *는 {0,}, +는 {1,}, ?는 {0,1}과 동일한 의미를 가짐

() (parenthesis)

- 특정 문자열을 ()로 grouping
- 한 개 이상의 그룹 지정 가능

```
# ab가 1~4회 0이상 반복되는 문자열 반환
g <- grep1("(ab){1,4}", zz)
zz[g]
```

```
[1] "ababababab" "abbb" "xabbbbcd"
```

```
# "The"로 시작하고 "punch"가 포함된 문자열 반환
g <- grep1("^(The)+.*(punch)", str)
str[g]
```

```
[1] "The juice of lemons makes fine punch."
```

| (vertical bar)

- |를 기준으로 좌우 문자 패턴 중 하나를 의미하며 OR 조건과 동일한 의미를 가짐
- [] 의 경우 메타문자나 문자 한글자에 대해서만 적용되는 반면 |는 문자를 묶어 문자열로 지정 가능

```
g <- grep1("(is|was)", str)
str[g]
```

```
[1] "These days a chicken leg is a rare dish."
[2] "Rice is often served in round bowls."
[3] "The box was thrown beside the parked truck."
[4] "Large size in stockings is hard to sell."
```

```
g <- grep1("(are|were)", str)
str[g]
```

```
[1] "These days a chicken leg is a rare dish."
[2] "The hogs were fed chopped corn and garbage."
```

3.3.2 문자 집합

TABLE 3.2: 정규표현식 메타 문자: 문자집합

| Expression | 설명 |
|------------|---|
| \w | 문자(letter), 숫자(digit), 또는 _ (underscore) 포함 |
| \d | 숫자 0에서 9 |
| \s | 공백문자(line break, tab, spaces) |
| \W | \w에 포함하지 않는 표현 |
| \D | 숫자가 아닌 표현 |
| \S | 공백이 아닌 표현 |

```
# \w 를 이용해 email 추출

email <- c("demo@naver.com",
          "sample@gmail.com",
          "coffee@daum.net",
          "redbull@nate.com",
          "android@gmail.com",
          "secondmoon@gmail.com",
          "zorba1997@korea.re.kr")

# 이메일 주소가 naver 또는 gmail만 추출
g <- grep1("\\w+@(naver|gmail)\\.\\w+", email)
email[g]
```

```
[1] "demo@naver.com"      "sample@gmail.com"      "android@gmail.com"
[4] "secondmoon@gmail.com"
```

```
# 숫자를 포함하는 문자열 추출: \d
ex <- c("ticket", "51203", "+-.!@#", "ABCD", "_", "010-123-4567")
g <- grep1("\\d", ex)
ex[g]
```

```
[1] "51203"      "010-123-4567"
```

```
# 뒤쪽 공백문자 제거
xx <- c("some text on the line 1; \n and then some text on line two      ")
sub("\\s+$", "", xx)
```

```
[1] "some text on the line 1; \n and then some text on line two"
```

```
# 영문자(소문자 및 대문자 포함), 숫자, 언더바(_)를 제외한 문자 포함
g <- grep1("\\W", ex)
ex[g]
```

```
[1] "+-.,!@#"      "010-123-4567"
```

```
# 숫자를 제외한 모든 문자 반환
g <- grep1("\\D", ex)
ex[g]
```

```
[1] "ticket"      "+-.,!@#"      "ABCD"      "_"      "010-123-4567"
```

```
# 영문자, 숫자, 언더바를 제외한 모든 문자 포함하고
# 숫자와 특수문자를 포함하는 문자열도 제외
g <- grep1("\\W\\D", ex)
ex[g]
```

```
[1] "+-.,!@#"
```

```
## 공백, 탭을 제외한 모든 문자 포함
```

```
blank <- c(" ", "_", "abcd", "\t", "%^$##*")
g <- grep1("\\S", blank)
blank[g]
```

```
[1] "_"      "abcd"      "%^$##*"
```

3.3.3 문자 클래스

- 문자 집합을 더 세분화하여 특정 목적에 맞는 정규 표현형
- 대괄호([]) 안에 특정 패턴에 해당하는 문자로 규칙 표현하고 하이픈 (-)을 사용해 특정 문자의 범위 지정 가능
- 응용 가능한 문자 클래스

TABLE 3.3: 정규표현식 주요 문자 클래스

| Expression | 설명 |
|-------------|------------------------|
| [a-z] | 알파벳 소문자 중 하나 |
| [A-Z] | 알파벳 대문자 중 하나 |
| [0-9] | 0에서 9까지 숫자 중 하나 |
| [a-zA-Z] | 모든 알파벳 중 하나 |
| [a-zA-Z0-9] | 알파벳 소문자나 숫자 중 한 문자 |
| [가-힝] | 모든 한글 중 하나 |
| [(abc)d] | 문자열 'abc'와 문자 'd' 중 하나 |

- POSIX (Portable Operating System Interface): 서로 다른 UNIX OS 의 API를 정리하여 이식성이 높은 유닉스 응용 프로그램을 개발하기 위한 목적으로 IEEE가 책정한 애플리케이션 인터페이스 규격 ([POS, 4 16](#))

TABLE 3.4: 정규표현식: POSIX 문자 클래스

| Expression | 설명 |
|-------------------------|--|
| <code>[:punct:]</code> | 구둣점 문자 [] [!#\$%&'()*+,.:/;<=>?@\\^_`{ }~-] |
| <code>[:alpha:]</code> | 알파벳 [A-Za-z]와 동일한 표현 |
| <code>[:lower:]</code> | 소문자 알파벳 [a-z]와 동일 |
| <code>[:upper:]</code> | 대문자 알파벳 [A-Z]와 동일 |
| <code>[:digit:]</code> | 숫자 0 ~ 9 [0-9]와 동일 |
| <code>[:alnum:]</code> | 알파벳과 숫자 [0-9A-Za-z]와 동일 |
| <code>[:cntrl:]</code> | 제어문자 b |
| <code>[:print:]</code> | 모든 인쇄 가능한 문자 |
| <code>[:space:]</code> | 공백문자 \\t\\r\\n\\v\\f |
| <code>[:blank:]</code> | 공백문자 중 \\t \\n |
| <code>[:xdigit:]</code> | 16 진수 |

```
movie <- c("terminator 3: rise of the machiens",
         "les miserables",
         "avengers: infinity war",
         "iron man",
         "indiana jones: the last crusade",
         "irish man",
         "mission impossible",
         "the devil wears prada",
         "parasite (gisaengchung)",
         "once upon a time in hollywood")
```

```
# 각 영화제목의 첫글자를 대문자로 변경
# \b는 단어의 양쪽 가장 자리의 빈 문자를 의미
# \1은 () 첫 번째 그룹, \\U는 대문자(perl)
gsub("\\b(\\w)", "\\U\\1", movie, perl = T)
```

```
[1] "Terminator 3: Rise Of The Machiens" "Les Miserables"
[3] "Avengers: Infinity War"           "Iron Man"
[5] "Indiana Jones: The Last Crusade"   "Irish Man"
[7] "Mission Impossible"                 "The Devil Wears Prada"
[9] "Parasite (Gisaengchung)"          "Once Upon A Time In Hollywood"
```

```
# 엑셀에서 ()로 표시된 음수 데이터를 읽어온 경우
# 0/를 음수로 표시
num <- c("0.123", "0.456", "(0.45)", "1.25")
gsub("\\(([0-9.]+)\\)", "-\\1", num)
```

```
[1] "0.123" "0.456" "-0.45" "1.25"
```

3.3.4 정규 표현식 예시

- 텍스트 데이터를 처리할 때 일반적으로 많이 활용되는 정규 표현식
- 정제되지 않은 데이터 가공 시 유용하게 활용

공백 제거

- 선행 예제에서 문자열 뒤에 존재하는 공백 제거 예시 확인
- 다음 예시들은 선행 및 모든 공백 제거에 대한 정규 표현식에 대해 살펴봄

필요 표현식

- 1) 공백을 다른 문자로 교체 해주는 함수 → gsub()
- 2) 공백 character class: \\s
- 3) 처음과 끝 지정 meta character: ^, \$
- 4) 조건을 찾기 위한 meta character: +, |

- 모든 공백을 제거하려면 → \\s
- 앞쪽 공백만 제거하려면? → ^\\s+
- 뒤쪽 공백만 제거하려면? → \\s+\$
- 양쪽 공백 모두를 제거하려면? 문장의 맨 앞에 공백이 하나 이상 존재하거나(OR, |), 문장 맨 끝에 공백이 하나 이상 존재 → (^\\s+|\\s+\$)

```
txt <- c("신종 코로나바이러스 감염증(코로나19) 환자 가운데 회복해서 항체가  
생긴 사람 중 절반가량은 체내에 바이러스가 남아 있는 것으로 나타났다. ")  
txt
```

```
[1] "신종 코로나바이러스 감염증(코로나19) 환자 가운데 회복해서 항체  
가\n생긴 사람 중 절반가량은 체내에 바이러스가 남아 있는 것으로 나타  
났다. "
```

```
# 모근 공백 제거  
gsub("\\s", "", txt)
```

```
[1] "신종코로나바이러스감염증(코로나19)환자가운데회복해서항체가생긴사람중절반  
가량은체내에바이러스가남아있는것으로나타났다."
```

```
# 앞쪽 공백만 제거  
gsub("^\\s+", "", txt)
```

[1] "신종 코로나바이러스 감염증(코로나19) 환자 가운데 회복해서 항체
가\n 생긴 사람 중 절반가량은 체내에 바이러스가 남아 있는 것으로 나타
났다. "

```
# 뒤쪽 공백만 제거  
gsub("\\s+$", "", txt)
```

[1] " 신종 코로나바이러스 감염증(코로나19) 환자 가운데 회복해서 항체
가\n 생긴 사람 중 절반가량은 체내에 바이러스가 남아 있는 것으로 나타
났다."

```
# 양쪽에 존재하는 공백들 제거  
gsub("(^\\s+|\\s+$)", "", txt)
```

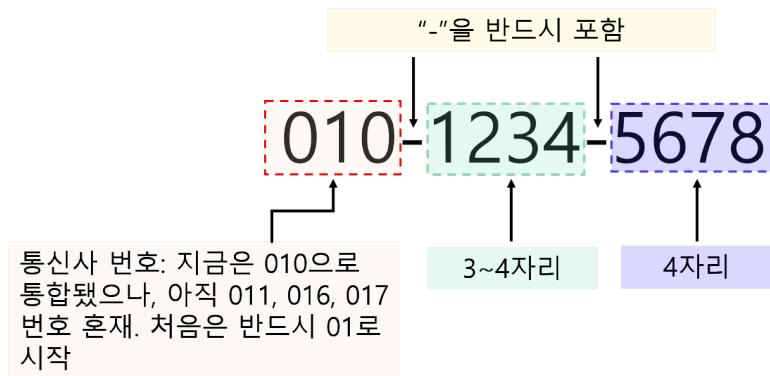
[1] "신종 코로나바이러스 감염증(코로나19) 환자 가운데 회복해서 항체
가\n 생긴 사람 중 절반가량은 체내에 바이러스가 남아 있는 것으로 나타
났다."

```
# 가운데 \n 뒤에 존재하는 공백들을 없애려면??  
gsub("(^\\s+| {2,}|\\s+$)", "", txt)
```

[1] "신종 코로나바이러스 감염증(코로나19) 환자 가운데 회복해서 항체가\n 생긴 사람 중 절반가량은 체내에 바이러스가 남아 있는 것으로 나타났다."

핸드폰 번호 추출

- 대한민국 핸드폰 번호의 형태



필요한 표현식

- 처음 세 자리: `^(01)\d{1}`
- 가운데 3~4자리: `\d{3,4}`
- 마지막 4자리: `\d{4}`

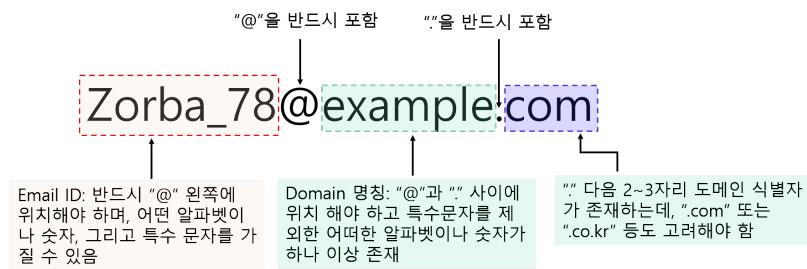
```
phone <- c("042-868-9999", "02-3345-1234",
          "010-5661-7578", "016-123-4567",
          "063-123-5678", "070-5498- 1904",
          "011-423-2912", "010-6745-2973")

g <- grep1("^(01)\d{1}-\d{3,4}-\d{4}", phone)
phone[g]
```

```
[1] "010-5661-7578" "016-123-4567"  "011-423-2912"  "010-6745-2973"
```

이메일 추출

- 정규 표현식을 이용해 이메일(e-mail) 주소만 텍스트 문서에서 추출
- 이메일 주소 구성



필요한 표현식

- E-mail ID(@ 왼쪽): 어떤 알파벳, 숫자, 특수문자가 1개 이상 → `[A-Za-z0-9[:punct:]]+`
- E-mail ID(@ 오른쪽-1): 어떤 알파벳이나 숫자가 하나 이상 존재하고 특수문자 포함(.xx.xx 추출에 필요) → `@[A-Za-z0-9[:punct:]]+`
- E-mail ID(@ 오른쪽-2): .xx 또는 .xxx 검색 → `\.\.[A-Za-z]+`

예시

- 네이버 뉴스 크롤링⁴ (nav, 2018)
 - 검색 포털: 네이버
 - 검색 범위: 2020년 4월 21일
 - 검색 keyword: 21대 국회위원 선거
 - 검색 뉴스 개수: 39개

⁴<https://dr-hkim.github.io/Naver-News-Web-Scraping-using-Keywords-in-R/>

– 검색결과 저장 파일: dataset/news-scraping.csv

- 개별 기사에 해당하는 URL로부터 ID 생성
- 각 뉴스로부터 기자들의 e-mail 추출
- 추출 후 기사 ID, 기사제목, e-mail 주소로 구성된 데이터 프레임 생성

```
# 크롤링한 데이터 불러오기
news_naver <- read.csv("dataset/test.news-scraping.csv", header = T,
stringsAsFactors = FALSE)

# regmatches 함수: regexpr(), gregexpr(), regexec()로 검색한 패턴을
# 텍스트(문자열)에서 추출

# ID 추출
id <- regmatches(news_naver$url, regexpr("\d{10}", news_naver$url))
contents <- news_naver$news_content
news_naver2 <- data.frame(id, title = news_naver$news_title,
stringsAsFactors = FALSE)
tmp <- regmatches(contents,
gregexpr("\b[A-Za-z0-9[:punct:]]+@[A-Za-z0-9[:punct:]]+\.\w\w+", contents))

names(tmp) <- id
x <- t(sapply(tmp, function(x) x[1:2], simplify = "array"))
colnames(x) <- paste0("email", 1:2)
email <- data.frame(id = row.names(x), x, stringsAsFactors = F)
res <- merge(news_naver2, email, by = "id")
head(res)
```

| | id |
|---|------------|
| 1 | 0000026769 |
| 2 | 0000091243 |

3 0000425146

4 0000425152

5 0000489588

6 0000535054

title

1 그림으로 보는 제21대 국회의원 선거 결과

2 '스트레이트' 꼼수로 얼룩진 21대 총선... 후퇴
한 정치개혁

3 [뉴스1번지] 20대 국회 막판 달구는 긴급재난지원금 논쟁

4 [뉴스1번지] 주호영 당선인에게 듣는 슬기로운 국회생활

5 [엄창섭의 몸과 삶] 나쁜 유전자, 나쁜 국회의원

6 고용진·전재수·유동수 의원 다시 금배지...'보험업계 숙원법안' 21대 국회 통과 기대

email1 email2

1 cyr@sisain.co.kr <NA>

2 <NA> <NA>

3 <NA> <NA>

4 <NA> <NA>

5 <NA> <NA>

6 huropa@inews24.com <NA>

```
# stringr 패키지 사용
```

```
# email <- str_extract_all(contents,
#                           "\\\\b[A-Za-z0-9[:punct:]]+@[A-Za-z0-9[:punct:]]+\\\\. [A-Za-z]+",
#                           simplify = TRUE)
# email <- data.frame(email, stringsAsFactors = FALSE)
# names(email) <- paste0("email", 1:2)
```


4

R 수학 함수, 분포 함수, 모형식 표현

4.1 수학함수

R은 광범위한 수학 함수를 내장하고 있고 다음 열거한 함수 목록은 그 일부임

- `exp()`: 지수(e)를 밑으로 하는 지수 함수
- `log()`: 자연 로그 함수
- `log10()`: 10을 밑으로 하는 로그
- `sqrt()`: 제곱근
- `abs()`: 절대값
- `sin()`, `cos()`, `tan()` ... : 삼각함수
- `min()`, `max()`: 벡터 내 최솟값과 최댓값 반환
- `which.min()`, `which.max()`: 벡터 내 최댓값과 최솟값에 대한 인덱스 반환
- `sum()`, `prod()`: 벡터 원소들의 합과 곱 결과 반환
- `cumsum()`, `cumprod()`: 벡터 원소들의 누적합과 누적곱
- `round()`, `floor()`, `ceiling()`: 수치형 값의 반올림, 내림, 올림
- `factorial()`: 팩토리얼 함수 $n!$

- choose(): 조합 함수 ($nC_r = \frac{n!}{r!(n-r)!}$)
- rev(): 역순으로 배열
- rank(): 백터 원소 값들의 순위 반환
- sweep(): 각 원소에서 요약통계량(예: 평균)으로부터 편차 계산 시 유용

학장예제1: 확률계산

- P_i : n 개의 독립적인 사건이 있고 i 번째 사건이 발생할 확률
- $n = 3$ 일 때, 각 사건의 이름을 각각 A, B, C 라고 할 때 이 중 사건 하나가 발생할 확률

$$\begin{aligned} P(\text{사건 하나가 발생할 확률}) &= \\ P(A\text{가 일어나고 } B\text{와 } C\text{가 일어나지 않을 확률}) &+ \\ P(A\text{가 일어나지 않고 } B\text{가 일어나고 } C\text{가 일어나지 않을 확률}) &+ \\ P(A, B\text{가 일어나지 않고 } C\text{만 일어날 확률}) \end{aligned}$$

- 여기서 $P(A\text{가 일어나고 } B\text{와 } C\text{가 일어나지 않을 확률}) = P_A(1 - P_B)(1 - P_C) \rightarrow$ 나머지도 마찬가지임
- 일반화 하면

$$\sum_{i=1}^n P_i(1 - P_1)(1 - P_2) \cdots (1 - P_{i-1})(1 - P_{i+1}) \cdots (1 - P_n)$$

- 수학함수로 구현: prod() 함수 활용

```
# 벡터 p에서 p_i 계산 함수
p <- c(0.2, 0.4, 0.3)
notp <- 1 - p
p[1] * prod(notp[-1]) +
p[2] * prod(notp[-2]) +
p[3] * prod(notp[-3])
```

[1] 0.452

```
p <- runif(10, 0, 1)
notp <- 1 - p
# 일반화 하려면 어떻게 해야 할까? -> 반복문 활용
tot <- 0
for (i in 1:length(p)) {
  tot <- tot + p[i] * prod(notp[-i])
}
```

확장예제2: 누적합, 누적곱

```
# cumsum, cumprod 함수 사용 예시
x <- c(2, 4, 1, 3, 7, 8)
cumsum(x); cumprod(x)
```

[1] 2 6 7 10 17 25

[1] 2 8 8 24 168 1344

확장예제3: 최솟값, 최댓값

```
# which.min, which.max 사용 예시
set.seed(100)
x <- sample(1:100, 100)
idx_min <- which.min(x)
x[idx_min]
```

[1] 1

```
idx_max <- which.max(x)
x[idx_max]
```

[1] 100

```
# min(), max(), pmin(), pmax() 허가
set.seed(5)
x <- runif(5, 2, 4)
y <- runif(5, 2, 4)
z <- cbind(x, y)

min(z); max(z) # z의 전체 값 중 최솟값과 최댓값 반환
```

[1] 2.2093

[1] 3.913

```
pmin(z); pmax(z) # 아무 값을 반환하지 않음
```

```
x      y  
[1,] 2.400429 3.402115  
[2,] 3.370437 3.055920  
[3,] 3.833752 3.615870  
[4,] 2.568799 3.913000  
[5,] 2.209300 2.220906
```

```
x      y  
[1,] 2.400429 3.402115  
[2,] 3.370437 3.055920  
[3,] 3.833752 3.615870  
[4,] 2.568799 3.913000  
[5,] 2.209300 2.220906
```

```
# 두 열을 비교해 각 행에서 최솟값, 최댓값을 반환  
pmin(z[, 1], z[, 2])
```

```
[1] 2.400429 3.055920 3.615870 2.568799 2.209300
```

```
pmax(z[, 1], z[, 2])
```

```
[1] 3.402115 3.370437 3.833752 3.913000 2.220906
```

학장예제5: sweep() 함수 활용

```
X <- matrix(1:12, 3, 4)  
m <- apply(X, 2, mean)  
M <- matrix(m, ncol = 4, nrow = 3, byrow = TRUE)  
X - M
```

```
[,1] [,2] [,3] [,4]
[1,] -1 -1 -1 -1
[2,] 0 0 0 0
[3,] 1 1 1 1
```

```
# sweep 함수 활용
sweep(X, 2, colMeans(X))
```

```
[,1] [,2] [,3] [,4]
[1,] -1 -1 -1 -1
[2,] 0 0 0 0
[3,] 1 1 1 1
```

확장예제6: 미분/적분

- 문자의 미분 및 수치 적분 가능

```
# 도함수 구하기
## D() 함수 사용
dx <- D(expression(exp(x^2)), "x") # exp(x^2)을 x에 대해서 1차 미분한 도함수
set.seed(3)
x <- runif(3, 1, 2)
eval(dx) # 위 입력한 x에 대한 도함수 값 출력
```

```
[1] 9.141245 94.842390 18.856751
```

```
## deriv() 함수 사용
grad <- D(expression(x*sin(x)), "x")
# 도함수를 R의 function으로 바로 반환 가능
dx2 <- deriv(expression(x*sin(x)), "x", function.arg = TRUE)
dx2(x)
```

```
[1] 1.074580 1.757109 1.361092
```

```
attr("gradient")
```

```
x
```

```
[1,] 1.3778035
```

```
[2,] 0.5482219
```

```
[3,] 1.2386966
```

```
# 수치 적분
```

```
## integrate() 함수 사용
```

```
## 주어진 함수의 적분식을 구한 후, 입력 구간에 대한 적분값 계산
```

```
integrate(f = function(x) x^2, lower = 0, upper = 1)
```

```
0.3333333 with absolute error < 3.7e-15
```

4.2 통계 분포 함수

R은 현존하는 대부분의 통계 확률 분포 함수를 제공하고 접두사 + 분포 이름 형태의 함수명을 갖고 있으며, 보통 다음과 같은 접두사를 통해 분포 함수 생성

- d: 밀도(density)의 약어로 확률 밀도함수(probability density function, pdf) 또는 이산형 분포의 확률 질량 함수(probability mass function, pmf)
- q: 분위수(quantile)의 약어로 상위 %에 해당하는 x 값을 반환
- p: 누적분포함수(cumulative density function, cdf)
- r: 특정 분포로부터 난수(확률변수) 생성

예: `dnorm()`, `qnorm()`, `pnorm()`, `rnorm()` 은 정규분포 관련 함수임

예제: 확률 분포 함수

```
## 카이제곱분포
x <- seq(0, 30, by = 0.1)
y <- dchisq(x, df = 3) # 자유도가 3인 카이제곱분포 밀도 함수
plot(x, y, type = "l",
      bty = "n",
      xlab = "", ylab = "",
      main = expression(paste("PDF of ", ~chi^2, " distribution")),
      lwd = 2,
      cex.main = 2)

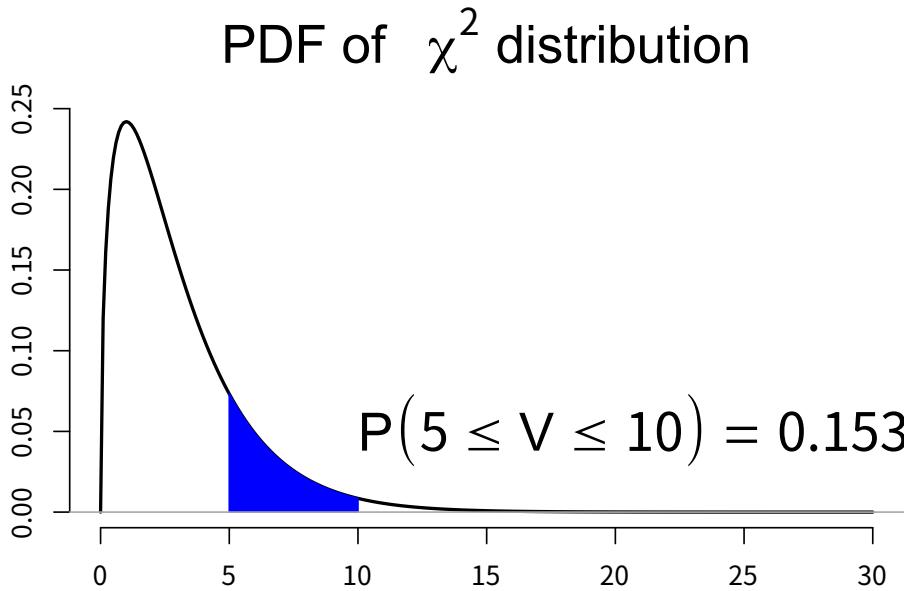
#  $P(5 < V < 10)$ 
pchisq(10, df = 3) - pchisq(5, df = 3)
```

[1] 0.153231

TABLE 4.1: 일반적인 R 통계 분포함수(일부 제시)

| Distribution | Density/Map | df | R cdf | R | RV | Parameter |
|--------------|---|---------------------|---------------------|---------------------|---------------------|---------------------------------------|
| 균일분포 | $\frac{1}{b-a}$, for $x \in [a, b]$ | <code>dunif</code> | <code>punif</code> | <code>qunif</code> | <code>rnorm</code> | min (a), max (b) |
| 지수분포 | $\lambda \exp(-\lambda x)$ | <code>dexp</code> | <code>pexp</code> | <code>qexp</code> | <code>rexp</code> | rate (λ) |
| 정규분포 | $\frac{1}{\sqrt{2\pi}\sigma} \exp\left\{-\frac{(x-\mu)^2}{2\sigma^2}\right\}$ | <code>dnorm</code> | <code>pnorm</code> | <code>qnorm</code> | <code>rnorm</code> | mean (μ), sd (σ) |
| χ^2 | $\frac{1}{\Gamma(\nu/2)2^{\nu/2}} x^{\nu/2-1} e^{-x/2}$ | <code>dcisq</code> | <code>pcisq</code> | <code>qchisq</code> | <code>rchisq</code> | df (ν) |
| <u>분포</u> | | | | | | |
| t 분포 | $\frac{\Gamma(\frac{\nu+1}{2})}{\Gamma(\frac{\nu}{2})} \frac{1}{\sqrt{\nu\pi}} \frac{1}{(1+x^2/\nu)^{\nu/2}}$ | | | <code>qt</code> | <code>rt</code> | df (ν) |
| 이항분포 | $\binom{n}{x} p^x (1-p)^{n-x}$ | <code>dbinom</code> | <code>pbinom</code> | <code>qbinom</code> | <code>rbinom</code> | size (n), prob (p) |
| 포아송분포 | $\frac{e^{-\lambda} \lambda^x}{x!}$ | <code>dpois</code> | <code>ppois</code> | <code>qpois</code> | <code>rpois</code> | lambda (λ) |

```
## 그림에 표현
idx <- x >= 5 & x <= 10
polygon(c(5, x[idx], 10),
         c(0, y[idx], 0),
         col = "blue",
         border = "blue")
abline(h = 0, col = "darkgray")
text(x = 10, y = 0.05, cex = 2,
      bquote(P({5 <= V} <= 10) ==
             .(sprintf("%.3f", pchisq(10, df = 3) - pchisq(5, df = 3)))),
      adj = 0)
```



```
# 분위수
qchisq(pchisq(10, df = 3), df = 3)
```

[1] 10

```
# 난수 생성
v <- rchisq(1000, df = 3)
mean(v) # 카이제곱분포의 평균은 이론적으로 자유도 값과 같음
```

[1] 2.994

4.3 모형식 표현

R에서 통계적 모형 표현 방법

- 지금까지 별다른 설명 없이 ~가 들어간 수식표현을 특정함수(예: lm(), t.test(), 심지어 그래프 생성에 필요한 함수 등)의 인수로 사용함.
- R은 (통계적) 모형을 표현하기 위해 formula 표현을 사용 → 일반적으로 좌변
 - ~ 우변형태로 표시
- 보통은 특정 함수 내에서 호출되며 데이터에 포함되어 있는 변수를 평가하지 않고 해당 함수에서 해석할 수 있도록 변수값을 불러올 수 있음.
- formula는 “language” 객체의 일종이며 “formula” 클래스를 속성으로 갖는 평가되지 않은 표현식(unevaluated expression)

```
typeof(quote(x + 10)) # 객체의 형태가 "language"
```

[1] "language"

```
class(quote(x + 10)) # 객체의 클래스가 "call"
```

[1] "call"

- R에서 **formula**을 특정하는 ~의 의미는 “즉시 평가(evaluate)하지 않고 이 코드의 의미를 전달(캡쳐)” → 인용(quote) 연산자로 볼 수 있는 이유임

```
# 수식 표현
a <- y ~ x
b <- y ~ x + b
c <- ~ x + y + z

typeof(c); class(c); attributes(c)
```

[1] "language"

[1] "formula"

```
$class
[1] "formula"

$.Environment
<environment: R_GlobalEnv>
```

- 가장 기본적인 **formula**의 형태는 아래와 같음

반응변수(response variable) ~ 독립변수(independent variables)

- ~ 는 “(좌변)은 (우변)의 함수로 나타낸 모형”으로 해석됨.
- 우변과 좌변 모두 일반적으로 여러 개의 변수들이 있을 수 있으며, 해당 변수의 추가는 +로 표시됨
- 좌변은 반응변수, 우변은 설명변수를 의미



일반적으로 좌변에 y 로 표현되는 반응변수는 학문 분야에 따라 종속변수(dependent variable), 표적변수(target variable), 결과변수(outcome variable), 레이블(label, y 가 범주형일 경우) 등으로 명칭되며, 우변에 y 를 설명하기 위해 사용하는 변수(x)를 마찬가지로 분야와 성격에 따라 독립변수(independent variable), 설명변수(exploratory variable), 예측변수(predictor variable), 위험 인자(risk factor), 공변량(covariate) 등으로 명칭된다.

- **formula**를 구성하는 항으로 벡터 객체를 직접 사용할 수 있으나, 데이터 프레임의 변수명을 **formula**의 항으로 구성할 수 있음.
- **formula**의 항으로 지정된 벡터 또는 변수들의 값은 **formula**를 호출해 사용할 때 까지 연결되지 않은 “언어”로써만 존재
- **formula**는 양면수식(two-sided formula, 좌변과 우변 모두에 하나 이상의 항이 존재)과 단면수식(one-sided formula, 우변에만 하나 이상의 항 존재)

```
set.seed(1)
x1 <- rnorm(100, 2, 4)
x2 <- runif(100, 2, 4)
x3 <- rpois(100, 3)
y <- 2*x1 + 3*x2 + 0.5*x3 + 5 + rnorm(100, 0, 4)

f1 <- y ~ x # y는 x의 함수
f2 <- y ~ x1 + x2 # y는 x1과 x2의 함수를 지칭하는 모형
typeof(f2); class(f2); attributes(f2)
```

[1] "language"

[1] "formula"

```
$class  
[1] "formula"
```

```
$.Environment  
<environment: R_GlobalEnv>
```

```
names(iris)
```

```
[1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"   "Species"
```

```
f3 <- Species ~ Sepal.Length + Sepal.Width + Petal.Length  
# 붓꽃의 종은 꽃받침 길이와 너비, 꽃잎의 길이에 대한 함수
```

```
f4 <- ~ x1 + x2  
f5 <- y ~ x1 + x2 + x3  
length(f3); length(f4)
```

```
[1] 3
```

```
[1] 2
```

```
f4[[1]]; f4[[2]]
```

```
~ ~ ~
```

```
x1 + x2
```

```
f5[[1]]; f5[[2]]; f5[[3]]
```

~ ~ ~

y

x1 + x2 + x3

수식 사용 이유

- 변수 간의 관계를 알기 쉽게 표현
- 복잡한 변수의 관계를 표현해 함수 내에서 손쉽게 해당 항에 대응하는 데이터 값에 접근 가능

수식표현 방법

- 위에서 기술환대로 좌변항 ~ 우변항으로 표현
- `formula()` 또는 `as.formula()` 함수를 통해 텍스트를 `formula` 형태로 생성 가능

```
f6 <- "y ~ x1 + x2 + x3"  
h <- as.formula(f6)  
h
```

y ~ x1 + x2 + x3

```
h <- formula(f6)
h
```

$y \sim x_1 + x_2 + x_3$

```
fs <- c(f1, f2, f6) # formula 객체를 concatenate 할 경우 list 객체
f1 <- lapply(fs, as.formula)
```

formula로 표현한 모형의 항에 대응하는 값으로 데이터 행렬 및 데이터 프레임 생성

- `model.frame()`: formula 객체에 표현된 항에 대응하는 데이터 값으로 이루어진 데이터 프레임 반환
- `model.matrix()`: 디자인 행렬을 생성하는 함수

```
d1 <- model.frame(y ~ x1 + x2) # 벡터값을 데이터 프레임으로 반환
head(d1)
```

| | y | x1 | x2 |
|---|----------|------------|----------|
| 1 | 15.23103 | -0.5058152 | 2.535016 |
| 2 | 25.03651 | 2.7345733 | 2.437291 |
| 3 | 19.26211 | -1.3425144 | 3.033594 |
| 4 | 29.55232 | 8.3811232 | 2.537901 |
| 5 | 10.58213 | 3.3180311 | 2.362337 |
| 6 | 25.53836 | -1.2818735 | 3.037152 |

```
# formula를 구성하고 있는 변수명에 대응하는 변수를 데이터 프레임에서 추출  
f3
```

```
Species ~ Sepal.Length + Sepal.Width + Petal.Length
```

```
d2 <- model.frame(f3, iris)  
head(d2)
```

| | Species | Sepal.Length | Sepal.Width | Petal.Length |
|---|---------|--------------|-------------|--------------|
| 1 | setosa | 5.1 | 3.5 | 1.4 |
| 2 | setosa | 4.9 | 3.0 | 1.4 |
| 3 | setosa | 4.7 | 3.2 | 1.3 |
| 4 | setosa | 4.6 | 3.1 | 1.5 |
| 5 | setosa | 5.0 | 3.6 | 1.4 |
| 6 | setosa | 5.4 | 3.9 | 1.7 |

```
# model.matrix()에서는 디자인 행렬만 반환  
# y 값은 포함하지 않고 우변에 해당하는 항에 대응하는 데이터 반환  
X1 <- model.matrix(y ~ x1 + x2)  
head(X1)
```

| | (Intercept) | x1 | x2 |
|---|-------------|------------|----------|
| 1 | 1 | -0.5058152 | 2.535016 |
| 2 | 1 | 2.7345733 | 2.437291 |
| 3 | 1 | -1.3425144 | 3.033594 |
| 4 | 1 | 8.3811232 | 2.537901 |
| 5 | 1 | 3.3180311 | 2.362337 |
| 6 | 1 | -1.2818735 | 3.037152 |

formula 관련 함수

- **terms()**: formula 객체에 포함되어 있는 항의 구조 파악
- **all.vars()**: formula에 포함되어 있는 변수명 추출
- **update()**: formula를 구성하는 항을 수정

```
terms(f2)
```

```
y ~ x1 + x2
attr(", "variables")
list(y, x1, x2)
attr(", "factors")
  x1 x2
y   0  0
x1  1  0
x2  0  1
attr(", "term.labels")
[1] "x1" "x2"
attr(", "order")
[1] 1 1
attr(", "intercept")
[1] 1
attr(", "response")
[1] 1
attr(", ".Environment")
<environment: R_GlobalEnv>
```

```
all.vars(f2)
```

```
[1] "y"  "x1" "x2"
```

```
update(f2, ~ . + x3)
```

$y \sim x_1 + x_2 + x_3$

formula에 사용되는 연산자와 의미

| Symbol | Example | Meaning |
|--------|-----------|-------------------------|
| + | X + Z | 변수 항 포함 |
| - | X + Z - X | 변수 항 제거 |
| : | X:W | X와 W의 교호작용 |
| %in% | X%in%W | 지분(nesting) |
| * | X*Z | $X + Z + X:Z$ |
| ^ | (X+W+Z)^3 | 3차 교호작용항까지 모든 항을 포함 |
| I | I(X^2) | as is: X^2 을 포함 |
| 1 | X - 1 | 절편 제거 |
| . | Y ~ . | 모든 변수 포함($X + W + Z$) |



일반 연산 시 A %in% B의 의미는 A가 B의 원소를 포함하는지에 대한 논리값을 반환해 주지만, formula에서 %in%은 중첩 또는 지분(nesting)을 내포함. R의 리스트 객체는 중첩 및 지분 구조의 대표적 형태임. 예를 들어 리스트에 포함된 한 원소에 대응하는 데이터의 형태 및 값은 동일 리스트의 다른 원소에 대응한 데이터의 형태 및 값이 다름. 즉, 리스트 객체는 한 객체에 여러 형태의 데이터 구조를 가질 수 있고 이를 중첩된 구조라고 함.

또한 실험계획법(experimental design)에서 지분 설계(nested design) 방법이 있는데, 두 개 요인(factor) A와 B가 존재할 때 A의 수준에 따른 B의 수준이 모두 다른 경우, 즉 교호작용이 존재하지 않는 형태의 실험설계법을 지칭함. 예를 들어 A사와 B사의 오렌지 주스 당도에 차이가 있는지를 알기 위해 각 회사에서 생산하고 있는 주

스 3종을 랜덤하게 선택했다고 가정해 보자. 여기서 주 관심사는 두 회사에서 생산한 주스의 당도의 동질성이지 오랜지 주스 간 당도 차이는 관심사항이 아님. 이 경우, 주 관심요인은 회사(C)이고, 요인 C는 회사 A, B라는 두 개의 수준(level)을 갖고 있음. 오랜지 주스(O)는 각 회사 별로 3개의 수준을 갖고 있는데, 각 회사에서 생산하는 오랜지 주스는 생산 공정에 차이가 있기 때문에 각 회사에 지분되어 있음. 즉, 회사 A에서 생산한 오랜지 주스 O₁, O₂, O₃은 회사 B에서 생산한 O₁, O₂, O₃과 다름.

```
set.seed(10)
x <- runif(30)
z <- runif(30, 2, 3)
w <- sample(0:1, 30, replace = TRUE)
y <- 3 + 2.5*x + x^2 + 1.5*z + 0.5 * w + 2*w*x + rnorm(30, 0, 2)

head(model.matrix(~ x))
```

| | (Intercept) | x |
|---|-------------|------------|
| 1 | 1 | 0.50747820 |
| 2 | 1 | 0.30676851 |
| 3 | 1 | 0.42690767 |
| 4 | 1 | 0.69310208 |
| 5 | 1 | 0.08513597 |
| 6 | 1 | 0.22543662 |

```
head(model.matrix(~ x + z)) # x + z
```

| | (Intercept) | x | z |
|---|-------------|------------|----------|
| 1 | 1 | 0.50747820 | 2.535597 |
| 2 | 1 | 0.30676851 | 2.093088 |
| 3 | 1 | 0.42690767 | 2.169803 |

```
4          1 0.69310208 2.899832  
5          1 0.08513597 2.422638  
6          1 0.22543662 2.747746
```

```
head(model.matrix(~ x + z - x)) # x + z0//서 z 항 제거
```

| | (Intercept) | z |
|---|-------------|----------|
| 1 | 1 | 2.535597 |
| 2 | 1 | 2.093088 |
| 3 | 1 | 2.169803 |
| 4 | 1 | 2.899832 |
| 5 | 1 | 2.422638 |
| 6 | 1 | 2.747746 |

```
head(model.matrix(~ x:w)) # 교호작용항만 포함
```

| | (Intercept) | x:w |
|---|-------------|-----------|
| 1 | 1 | 0.5074782 |
| 2 | 1 | 0.3067685 |
| 3 | 1 | 0.0000000 |
| 4 | 1 | 0.0000000 |
| 5 | 1 | 0.0000000 |
| 6 | 1 | 0.0000000 |

```
head(model.matrix(~ x*w)) # x + w + x:w
```

| | (Intercept) | x w | x:w |
|---|-------------|------------|-------------|
| 1 | 1 | 0.50747820 | 1 0.5074782 |

```

2      1 0.30676851 1 0.3067685
3      1 0.42690767 0 0.0000000
4      1 0.69310208 0 0.0000000
5      1 0.08513597 0 0.0000000
6      1 0.22543662 0 0.0000000

```

```
head(model.matrix(~ (x + z + w)^3)) # x + z + w + x:z + z:w + x:w + x:w:z
```

| | (Intercept) | x | z w | x:z | x:w | z:w | x:z:w |
|---|--------------|----------|-------------|---------------------|---------------------|-----|-------|
| 1 | 1 0.50747820 | 2.535597 | 1 1.2867602 | 0.5074782 2.535597 | 1.2867602 | | |
| 2 | 1 0.30676851 | 2.093088 | 1 0.6420935 | 0.3067685 2.093088 | 0.6420935 | | |
| 3 | 1 0.42690767 | 2.169803 | 0 0.9263056 | 0.0000000 0.0000000 | 0.0000000 0.0000000 | | |
| 4 | 1 0.69310208 | 2.899832 | 0 2.0098799 | 0.0000000 0.0000000 | 0.0000000 0.0000000 | | |
| 5 | 1 0.08513597 | 2.422638 | 0 0.2062536 | 0.0000000 0.0000000 | 0.0000000 0.0000000 | | |
| 6 | 1 0.22543662 | 2.747746 | 0 0.6194427 | 0.0000000 0.0000000 | 0.0000000 0.0000000 | | |

```
head(model.matrix(~ x + I(x^2))) # x + x^2
```

| | (Intercept) | x | I(x^2) |
|---|--------------|-------------|--------|
| 1 | 1 0.50747820 | 0.257534127 | |
| 2 | 1 0.30676851 | 0.094106916 | |
| 3 | 1 0.42690767 | 0.182250156 | |
| 4 | 1 0.69310208 | 0.480390494 | |
| 5 | 1 0.08513597 | 0.007248133 | |
| 6 | 1 0.22543662 | 0.050821668 | |

```
# head(model.matrix(~ x + x^2))
head(model.matrix(~ x - 1))
```

```
x  
1 0.50747820  
2 0.30676851  
3 0.42690767  
4 0.69310208  
5 0.08513597  
6 0.22543662
```

```
dat <- data.frame(y, x, w, z)  
head(model.matrix(y ~ ., data = dat))
```

```
(Intercept)          x   w       z  
1            1 0.50747820 1 2.535597  
2            1 0.30676851 1 2.093088  
3            1 0.42690767 0 2.169803  
4            1 0.69310208 0 2.899832  
5            1 0.08513597 0 2.422638  
6            1 0.22543662 0 2.747746
```

```
head(model.matrix(y ~ .^2, data = dat))
```

```
(Intercept)          x   w       z      x:w      x:z      w:z  
1            1 0.50747820 1 2.535597 0.5074782 1.2867602 2.535597  
2            1 0.30676851 1 2.093088 0.3067685 0.6420935 2.093088  
3            1 0.42690767 0 2.169803 0.0000000 0.9263056 0.000000  
4            1 0.69310208 0 2.899832 0.0000000 2.0098799 0.000000  
5            1 0.08513597 0 2.422638 0.0000000 0.2062536 0.000000  
6            1 0.22543662 0 2.747746 0.0000000 0.6194427 0.000000
```

```
# nested 구조
dat2 <- expand.grid(C = c("A", "B"), O = paste0("O", 1:3),
                      y = runif(3, 1, 2))
dat2 <- dat2[order(dat2$C, dat2$O), ]
model.matrix(y ~ C + O %in% C, data = dat2)
```

| | (Intercept) | CB | CA:002 | CB:002 | CA:003 | CB:003 |
|----|-----------------------|-------------------|--------|--------|--------|--------|
| 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 7 | 1 | 0 | 0 | 0 | 0 | 0 |
| 13 | 1 | 0 | 0 | 0 | 0 | 0 |
| 3 | 1 | 0 | 1 | 0 | 0 | 0 |
| 9 | 1 | 0 | 1 | 0 | 0 | 0 |
| 15 | 1 | 0 | 1 | 0 | 0 | 0 |
| 5 | 1 | 0 | 0 | 0 | 1 | 0 |
| 11 | 1 | 0 | 0 | 0 | 1 | 0 |
| 17 | 1 | 0 | 0 | 0 | 1 | 0 |
| 2 | 1 | 1 | 0 | 0 | 0 | 0 |
| 8 | 1 | 1 | 0 | 0 | 0 | 0 |
| 14 | 1 | 1 | 0 | 0 | 0 | 0 |
| 4 | 1 | 1 | 0 | 1 | 0 | 0 |
| 10 | 1 | 1 | 0 | 1 | 0 | 0 |
| 16 | 1 | 1 | 0 | 1 | 0 | 0 |
| 6 | 1 | 1 | 0 | 0 | 0 | 1 |
| 12 | 1 | 1 | 0 | 0 | 0 | 1 |
| 18 | 1 | 1 | 0 | 0 | 0 | 1 |
| | attr(,"assign") | | | | | |
| | [1] | 0 1 2 2 2 2 | | | | |
| | attr(,"contrasts") | | | | | |
| | attr(,"contrasts")\$C | | | | | |
| | [1] | "contr.treatment" | | | | |
| | attr(,"contrasts")\$O | | | | | |

[1] "contr.treatment"



5

R Markdown

Sketch

- 동일한 문서에 코드, 결과, 텍스트가 동시에 있을 수 있을까?
- 만약 결과와 도표가 자동으로 생성된 경우 데이터가 변경 되더라도 자동으로 문서를 업데이트 할 수 있을까?
- 최종 완료한 문서가 미래에도 열 수 있을까?
- 이러한 모든 과정이 매우 쉽다면??

5.1 R Markdown의 구성



본 절의 내용 중 일부는 지난 학기 강의노트 1.7절과 중복되거나 재구성한 내용이 포함됨.

1. R Markdown은 R 코드와 분석 결과(표, 그림 등)을 포함한 문서 또는 컨텐츠를 제작하는 도구로 일반적으로 아래 열거한 형태로 활용함
 - 문서 또는 논문(pdf, html, docx)

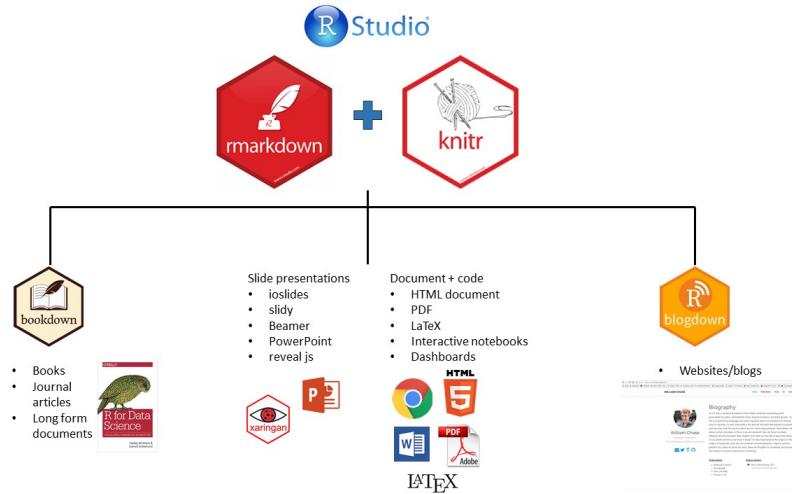


FIGURE 5.1: R markdown 세계(<https://ulyngs.github.io/rmarkdown-workshop-2019>에서 발췌)

- 프리젠테이션(pdf, html, pptx)
- 웹 또는 블로그

2. 재현가능(reproducible)한 분석 및 연구¹ 가능

- 신뢰성 있는 문서 작성
- Copy & paste를 하지 않고 효율적 작업 가능

R 마크다운 파일 = .Rmd 확장자를 가진 일반 텍스트 파일

```
---
title: "Untitled.Rmd"
date: "2020-09-11"
```

¹과학적 연구의 결과물을 오픈소스로 내놓고 누구라도 검증 가능

```
output: html_document
---
`r setup, include=FALSE`
knitr::opts_chunk$set(echo = TRUE)
```

R Markdown
```

Markdown은 HTML, PDF 및 MS Word 문서를 작성하기 위한 간단한 형식의 지정 구문입니다.  
R Markdown 사용에 대한 자세한 내용은 <<http://rmarkdown.rstudio.com>>을 참조하십시오.

\*\*Knit\*\* 버튼을 클릭하면 두 가지를 모두 포함하는 문서가 생성됩니다.  
문서에 포함 된 R 코드 청크의 출력 내용뿐 아니라  
다음과 같이 R 코드 청크를 포함 할 수 있습니다.

```
```{r cars}
summary(cars)
``
```

```
## Including Plots
```

You can also embed plots, for example:

```
```{r pressure, echo=FALSE}
```

```
plot(pressure)
````
```

`echo = FALSE` 매개 변수가 코드 청크에 추가 되었습니다.

플롯을 생성 한 R 코드의 인쇄를 방지합니다.

위 R Markdown 문서는 아래 그림과 같이 YAML, Markdown 텍스트, Code Chunk 세 부분으로 구성됨.

YAML (YAML Ain't Markup Language)

- R Markdown 문서의 metadata로 문서의 맨 처음에 항상 포함(header)되어야 함.
- R Markdown 문서의 최종 출력 형태(html, pdf, docx, pptx 등), 제목, 저자, 날짜 등의 정보 등을 포함

최종 문서 생성 과정

- Rmd 파일을 knitr 을 통해 .md 파일로 변환 후 pandoc 이라는 문서 변환기를 통해 원하는 문서 포맷으로 출력

5.2 R Markdown 기본 문법(syntax)

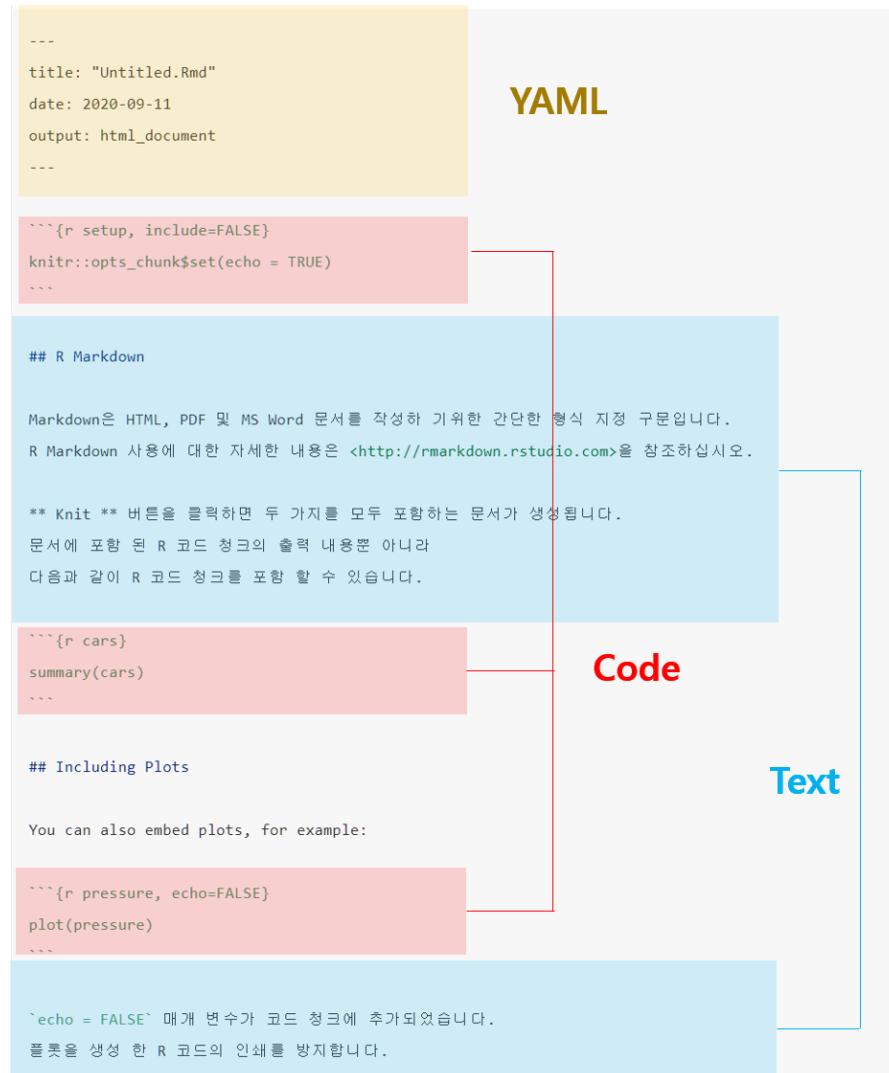


FIGURE 5.2: R markdown structure

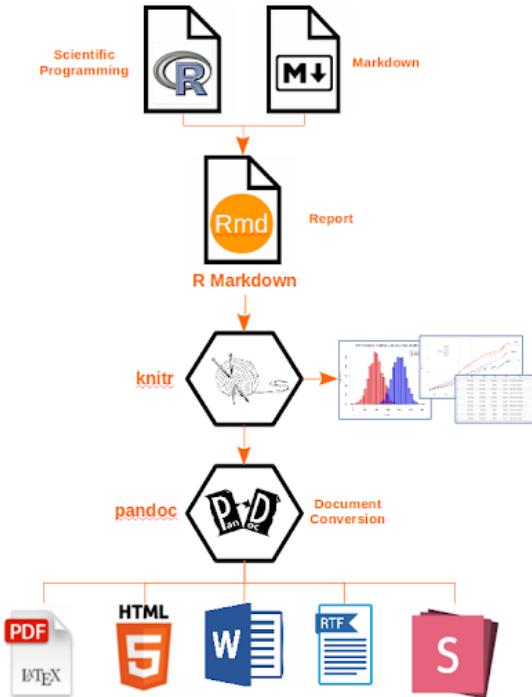


FIGURE 5.3: R Markdown의 최종 결과물 산출과정(<http://applied-r.com/project-reporting-template/>)

R Markdown의 기본 문법은 Rstudio 풀다운 메뉴 [Help] → [Markdown Quick Reference]에서 확인 가능

5.2.1 텍스트 문법

강조(emphasis)

- 이탤릭체: `*italic1*`, `_italic2_` → *italic1*, *italic2*
- 볼드(굵은)체: `*bold1*`, `__bold2__` → **bold1**, **bold2**

Inline code

- ‘inline code’ → `inline code`

아래/위 첨자(sub/superscript)

- $\text{subscript} \sim 2 \sim \rightarrow \text{subscript}_2$
- $\text{superscript} \wedge 2 \wedge \rightarrow \text{superscript}^2$

삭제표시(strike through)

- $\sim\sim\text{strikethrough}\sim\sim \rightarrow \text{strikethrough}$

생략표시(ellipsis)

- $\dots \rightarrow \dots$

긴/짧은 대쉬(en/em-dash)

- 긴 대쉬: $\text{---} \rightarrow \text{—}$
- 짧은 대쉬: $\text{--} \rightarrow -$

특수문자 탈출 지정자

- $\backslash^*, \backslash_, \backslash\sim, \backslash\backslash \rightarrow *, _, \sim, \backslash$

하이퍼링크

-[text](link) → R Markdown Cheat-sheet²

외부그림 삽입



FIGURE 5.4: 장난꾸러기

- ! [image title] (path/to/image):
! [장난꾸러
기]] (figures/son-02.jpg)

강제 줄바꿈(line breaks)

²<https://rmarkdown.rstudio.com/lesson-15.HTML>

- 하나의 줄에서 공백(space) 두 개 이상 또는 백슬레시(\) 입력 후 [Enter]

```
End a line with two spaces to start  
a new paragraph
```

End a line with two spaces to start a new paragraph

```
End a line with two spaces to start\  
a new paragraph
```

End a line with two spaces to start
a new paragraph

각주(footnote)

- A footnote³ [주석내용] → A footnote³

주석(comment)

- <!-- this is a comment that won't be shown --> →

 RStudio에서 단축키 [Ctrl] + [Shift] + [C]를 통해 전체 line에 대해 주석처리 가능

³주석내용

5.2.2 Block-level elements

장/절(header)

- # Header 1 (chapter, 장)
- ## Header 2 (section, 절)
- ### Header 3 (subsection, 관)

목록(list)

- 비순서(unordered) 목록: -, *, + 중 어느 하나로 입력 가능

```
- one item
* two item
  + sub-item 1
  + sub-item 2
    - subsub-item 1
    - subsub-item 2
```

- one item
- two item
 - sub-item 1
 - sub-item 2
 - * subsub-item 1
 - * subsub-item 2
- 순서(ordered) 목록: 비순서 목록의 기호 대신 숫자로 리스트 생성

```
1. the first item
  - sub-item 1
2. the second item
3. the third item
```

1. the first item
 - sub-item 1
 2. the second item
 3. the third item
- 같은 숫자로 적어도 순서대로 목록 생성

```
1. the first item
  - sub-item 1
1. the second item
1. the third item
```

1. the first item
 - sub-item 1
2. the second item
3. the third item

인용구(blockquote): >로 시작

```
> "There are three kinds of lies: lies, damn lies, and statistics"
>
> --- Benjamin Disraeli
```

“There are three kinds of lies: lies, damn lies, and statistics”

— Benjamin Disraeli

5.2.3 수식표현(math expression)

- 줄 안에 수식 입력 시 \$수식표현\$ 으로 입력
- 수식 display style (보통 교과서에 정리 및 정의에 기술된 수식들) 적용
시 \$\$ ~ \$\$ 안에 수식 입력
- 수식 표현은 LaTeX 의 수식 표현을 동일하게 준용(<https://www.latex4technics.com/>, <https://latex.codecogs.com/legacy/eqneditor/editor.php>에서 수식 입력 명령어 학습 가능)
- LaTeX 수식 입력 코드는
- 예시

$$P(X = x) = f(x; n, p) = \binom{n}{x} p^x (1-p)^{n-x}$$

- Inline equation: \$P(X = x) = f(x; n, p) = \{n \choose x} p^x (1-p)^{n-x} \rightarrow P(X = x) = f(x; n, p) = \binom{n}{x} p^x (1-p)^{n-x}\$
- Math block: \$\$P(X = x) = f(x; n, p) = \{n \choose x} p^x (1-p)^{n-x}\$\$

$$P(X = x) = f(x; n, p) = \binom{n}{x} p^x (1-p)^{n-x}$$

- \$ \$ 또는 \$\$ \$\$ 안에 LaTeX에서 제공하는 수식 함수 사용 가능

```
$$\begin{array}{ccc}
x_{11} & x_{12} & x_{13} \\
x_{21} & x_{22} & x_{23}
\end{array}$$
```

$$\begin{matrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \end{matrix}$$

```
$$\Theta = \begin{pmatrix} \alpha & \beta \\ \gamma & \delta \end{pmatrix}
```

$$\begin{aligned}
g(X_n) &= g(\theta) + g'(\tilde{\theta})(X_n - \theta) \notag \\
\sqrt{[g(X_n) - g(\theta)]} &= \sqrt{[X_n - \theta]}
\end{aligned}$$

$$g(X_n) = g(\theta) + g'(\tilde{\theta})(X_n - \theta)$$

$$\sqrt{n}[g(X_n) - g(\theta)] = g'(\tilde{\theta})\sqrt{n}[X_n - \theta]$$

5.3 R Code Chunks

- 실제 R code가 실행되는 부분임
- Code chunk 실행 시 다양한 옵션 존재(본 강의에서는 몇 개의 옵션만 다룰 것이며, 더 자세한 내용은 <https://yihui.org/knitr/options/> 또는 R Markdown 레퍼런스 가이드⁴ 참조)
- Code chunk는 `{{r}}`로 시작되며 r은 code 언어 이름을 나타냄.
- Code chunk는 `{{`}`}`로 종료
- R Markdown 문서 작성 시 단축키 [Ctrl] + [Alt] + [I]를 입력하면 Chunk 입력창이 자동 생성됨
- Code chunk의 옵션 조정을 통해 코드의 출력여부, 코드 출력 시 코드의 출력 형태, 코드의 결과물 출력 조정 가능

자주 활용하는 chunk 옵션

코드 실행 관련 청크

⁴<https://rstudio.com/wp-content/uploads/2015/03/rmarkdown-reference.pdf>

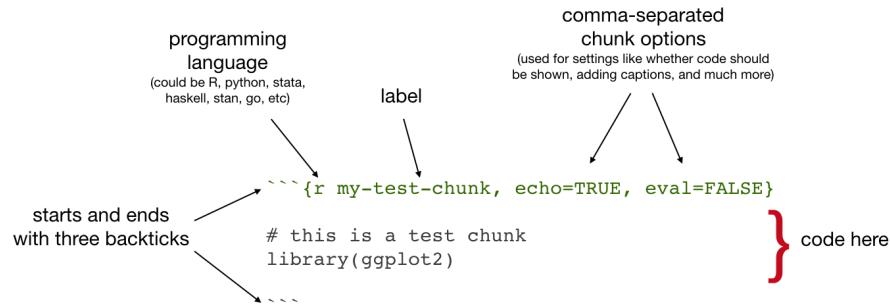


FIGURE 5.5: Chunk anatomy (<https://ulyngs.github.io/rmarkdown-workshop-2019>에서 발췌)

TABLE 5.1: 코드 실행 관련 청크

| Chunk 옵션 | Default | 설명 |
|----------|---------|-------------------------------|
| eval | TRUE | R 실행(코드 실행 결과)에 대응하는 결과 출력 여부 |
| include | TRUE | 출력 문서에 코드 청크의 내용을 포함할지 여부 |

```
```{r ex01-1, eval=TRUE}
summary(iris)
hist(iris$Sepal.Length)
```

```{r ex01-2, eval=FALSE}
summary(iris)
```
```

```
hist(iris$Sepal.Length)
```

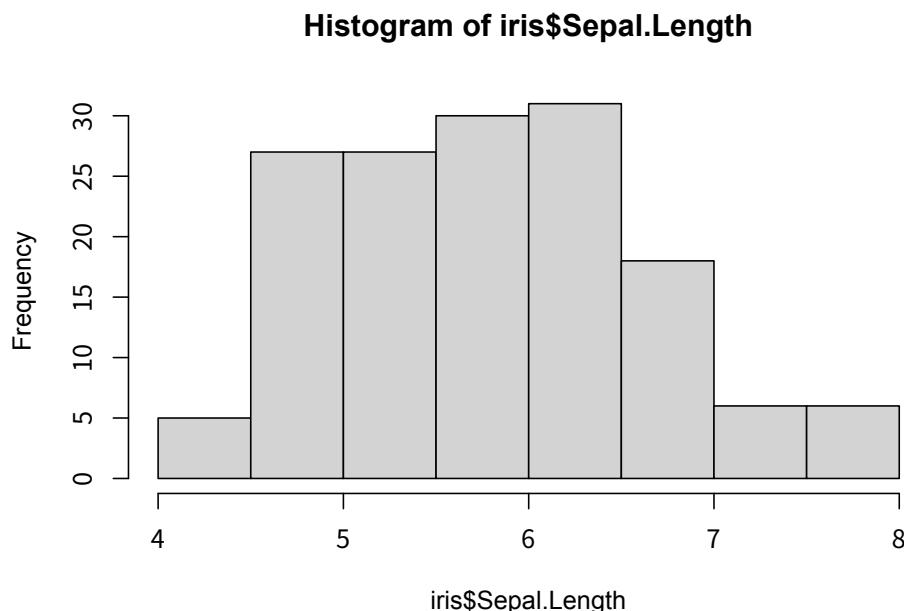
```
...
```

```
#첨크 옵션 eval=TRUE  
summary(iris)
```

| Sepal.Length | Sepal.Width | Petal.Length | Petal.Width |
|---------------|---------------|---------------|---------------|
| Min. :4.300 | Min. :2.000 | Min. :1.000 | Min. :0.100 |
| 1st Qu.:5.100 | 1st Qu.:2.800 | 1st Qu.:1.600 | 1st Qu.:0.300 |
| Median :5.800 | Median :3.000 | Median :4.350 | Median :1.300 |
| Mean :5.843 | Mean :3.057 | Mean :3.758 | Mean :1.199 |
| 3rd Qu.:6.400 | 3rd Qu.:3.300 | 3rd Qu.:5.100 | 3rd Qu.:1.800 |
| Max. :7.900 | Max. :4.400 | Max. :6.900 | Max. :2.500 |

| Species |
|---------------|
| setosa :50 |
| versicolor:50 |
| virginica :50 |

```
hist(iris$Sepal.Length)
```



```
# 청크 옵션 eval=FALSE  
summary(iris)  
hist(iris$Sepal.Length)
```

소스 코드 출력(텍스트) 결과 관련 청크

TABLE 5.2: 소스 코드 출력 결과 관련 청크

| Chunk 옵션 | Default | 설명 |
|----------|---------|--|
| echo | TRUE | R 실행 결과에 대응하는 코드 출력 여부 |
| results | markup | 출력 결과 포맷 지정을 위한 옵션으로 추가적으로 3 가지 옵션 선택 가능: 'hide', 'asis', 'hold', 'markup' |
| error | TRUE | 코드 또는 스크립트에 구문오류 메세지 출력 여부 |
| message | TRUE | 코드로부터 생성된 메세지 출력 여부 |
| warning | TRUE | 경고 메세지 출력 여부 |

- echo: 코드 청크에 작성한 R-script 출력 여부 결정
 - echo = FALSE 이면 소스 코드 출력 없이 그림 결과만 출력

```
```{r ex01-2, echo=TRUE}
require(ggthemes) # ggtheme 패키지 불러오기
require(ggpubr) # ggpubr 패키지 불러오기
iris %>%
 ggplot(aes(x = Sepal.Length, y = Petal.Width, color = Species)) +
 geom_point(size = 5) +
 theme_pubclean() +
```

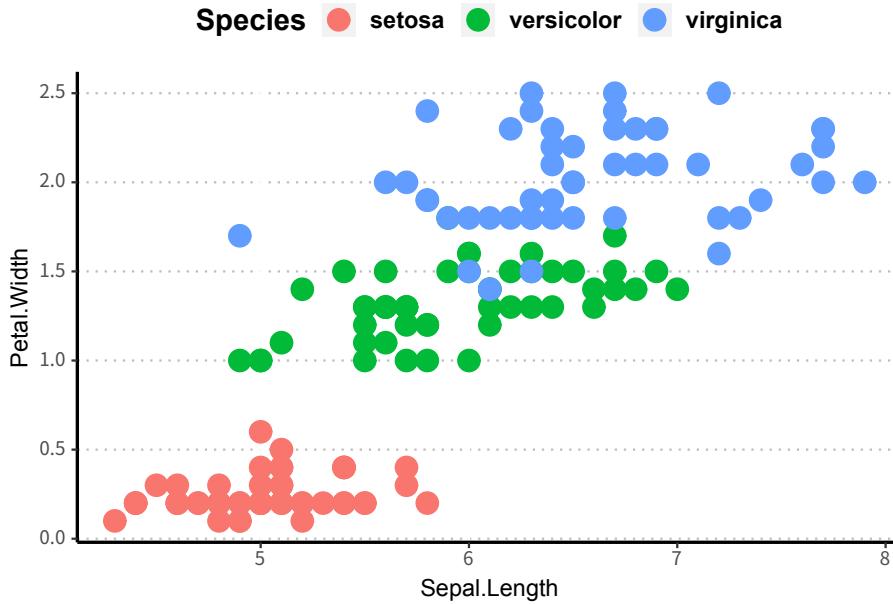
```
theme(axis.line = element_line(size = 0.8),
 legend.title = element_text(face = "bold", size = 15),
 legend.text = element_text(face = "bold", size = 12))

```  
```{r ex01-3, echo=FALSE}  
require(ggthemes) # ggtheme 패키지 불러오기
require(ggpubr) # ggpubr 패키지 불러오기
iris %>%
 ggplot(aes(x = Sepal.Length, y = Petal.Width, color = Species)) +
 geom_point(size = 5) +
 theme_pubclean() +
 theme(axis.line = element_line(size = 0.8),
 legend.title = element_text(face = "bold", size = 15),
 legend.text = element_text(face = "bold", size = 12))

```
```

```
# echo = TRUE  
require(ggthemes) # ggtheme 패키지 불러오기  
require(ggpubr) # ggpubr 패키지 불러오기  
iris %>%  
  ggplot(aes(x = Sepal.Length, y = Petal.Width, color = Species)) +  
  geom_point(size = 5) +  
  theme_pubclean() +
```

```
theme(axis.line = element_line(size = 0.8),
      legend.title = element_text(face = "bold", size = 15),
      legend.text = element_text(face = "bold", size = 12))
```



- **results:** 코드의 텍스트 출력 결과 포맷 지정
 - **markup** (default): 코드 청크 내 스크립트의 출력 형태에 따라 텍스트 출력 결과를 mark-up
 - **asis**: 변환하지 않은 원래 R 출력 결과 그대로(as is) 출력
 - **hide**: R 스크립트로 생성된 텍스트 출력을 보여주지 않음(warning, message 출력 예외)
 - **hold**: 코드 청크로 생성된 모든 소스 및 출력을 단일 블록으로 축소

```
# results = 'markup'인 경우 아래 텍스트를 mark-up
# (0) 경우 아래 텍스트는 ```` 블럭 처리)한 결과를 md 파일로 전송
cat("I'm raw **Markdown** content.\n")
```

I'm raw **Markdown** content.

The screenshot shows two files side-by-side:

- code-chunk.Rmd** (left): Contains R code for generating raw Markdown content. It includes a YAML header and an R chunk with `results = 'markup'` set to TRUE.
- code-chunk.md** (right): The resulting Markdown file, which contains the text "I'm raw **Markdown** content.".

FIGURE 5.6: 청크 옵션 results = 'markup'인 경우 rmd vs. md 파일 비교

```
# results = 'asis' 인 경우 텍스트를 그대로 md 파일에 입력
cat("I'm raw **Markdown** content.\n")
```

I'm raw **Markdown** content.

The screenshot shows two files side-by-side:

- code-chunk.Rmd** (left): Contains R code for generating raw Markdown content using the 'asis' results option. It includes a YAML header and an R chunk with `results = 'asis'` set to TRUE.
- code-chunk.md** (right): The resulting Markdown file, which contains the text "I'm raw **Markdown** content.".

FIGURE 5.7: 청크 옵션 results = 'asis'인 경우 rmd vs. md 파일 비교

```
# results = 'hide'

cat("I'm raw **Markdown** content.\n")

# 텍스트 결과를 출력하지 않음
```

```
# results = 'hold'가 아닌 경우 한 라인 별 출력 결과 생성
x <- rnorm(10)

x
```

```
[1] 0.16825442 -0.79707719 -0.51588813 0.07336067 -2.76822493 0.12431142
[7] -0.44223684 0.18220431 -0.03180977 -0.62468010
```

```
y <- rnorm(10, 1, 2)

y
```

```
[1] 1.3829024 0.4000121 -0.7486707 1.1772496 0.2623380 3.5739766
[7] 3.1862142 2.3155745 4.4854996 4.2218606
```

```
x + y
```

```
[1] 1.5511568 -0.3970651 -1.2645588 1.2506102 -2.5058870 3.6982880
[7] 2.7439773 2.4977788 4.4536898 3.5971805
```

```
# results = 'hold'인 경우 코드 부분과 출력 부분이 따로 블록 처리
x <- rnorm(10)

x

y <- rnorm(10, 1, 2)

y

x + y
```

```
[1] -1.09459925 -0.21053489 -1.07717535 -0.26660768  0.09909734 -0.60882269
[7] -0.01662671 -0.24816660  0.89862826  0.67173425
[1]  1.462578 -1.678951  5.994292  1.003081 -2.579507  1.928053 -1.685209
[8]  1.421291 -4.770008 -2.449254
[1]  0.3679787 -1.8894855  4.9171170  0.7364735 -2.4804097  1.3192306
[7] -1.7018360  1.1731243 -3.8713799 -1.7775193
```

- `error`: 코드 청크 내 스크립트에 오류에 대한 보존 여부(`stop()`)
 - 기본적으로 Rmarkdown 컴파일 시 `error`에 대한 옵션이 FALSE
이기 때문에 스크립트(코드)에 오류가 포함되면 컴파일이 정지됨.
 - `error = TRUE` 이면 오류 메세지를 포함한 텍스트 결과를 출력

```
3x <- 3
x <- 25 # 위 행이 구문 오류를 포함하고 있기 때문에
          # 오류 이후의 코드는 실행되지 않음
x
```

Error: <text>:1:2: 예상하지 못한 기호(symbol)입니다.

1: 3x

- `message/warning`: 텍스트 출력물 중 경고(warning, `warning()`) 함수의 출력 결과) 메세지 출력 여부 결정

```
# message = TRUE 인 경우 함수 message 출력
testit <- function() {
  message("testing package startup messages")
```

```
packageStartupMessage("initializing ...", appendLF = FALSE)
Sys.sleep(1)
packageStartupMessage(" done")
} # help(message) 예시 중 발췌

testit()
```

testing package startup messages

initializing ... done

```
# message=FALSE -> 메세지 출력하지 않음
testit()
```

```
# 경고 메세지 출력
x <- c(1, 2, "new", 4:10)
x <- as.numeric(x)
```

Warning: 강제형변환에 의해 생성된 NA입니다

코드 서식 관련 청크 옵션

TABLE 5.3: 코드 서식 관련 청크

| Chunk 옵션 | Default | 설명 |
|-----------|---------|--|
| comment | TRUE | 소스 코드 실행 출력의 각 줄 앞에 붙는 표시문자 출력 여부: 기본 값은 '# #' 임 |
| highlight | TRUE | 구문 강조 여부 |
| prompt | FALSE | R 프롬프트 출력 여부 |
| tidy | FALSE | R 소스 코드 출력 정리 여부 |

- comment: 텍스트 출력물에 주석 표시(default)를 함으로써 소스 코드 와 출력 결과를 동시 선택과 복사를 가능(##는 주석 표시이기 때문에 실행되지 않음)
 - 주석 표시를 제거하고 싶다면 comment = NA 또는 comment = ''

```
# 디플트 comment 사용
summary(iris)
```

```
##   Sepal.Length   Sepal.Width    Petal.Length   Petal.Width
##   Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
##   1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
##   Median :5.800   Median :3.000   Median :4.350   Median :1.300
##   Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
##   3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
##   Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
```

```
##      Species
##  setosa    :50
##  versicolor:50
##  virginica :50
##
##
```

- `highlight`: 구문 강조 표시 여부
 - `highlight=FALSE` 일 때 소스 코드 출력 결과

```
# highlight=FALSE

iris %>%
  ggplot(aes(x = Sepal.Length, y = Petal.Width, color = Species)) +
  geom_point(size = 5) +
  theme_pubclean() +
  theme(axis.line = element_line(size = 0.8),
        legend.title = element_text(face = "bold", size = 15),
        legend.text = element_text(face = "bold", size = 12))
```

- `prompt`: R 콘솔 상 프롬프트 >, + 출력 여부

```
> # prompt = TRUE 인 경우 코드 출력 결과
> require(ggthemes) # ggtheme 패키지 불러오기
> require(ggpubr) # ggpubr 패키지 불러오기
> iris %>%
+   ggplot(aes(x = Sepal.Length, y = Petal.Width, color = Species)) +
+   geom_point(size = 5) +
+   theme_pubclean() +
```

```
+     theme(axis.line = element_line(size = 0.8),
+           legend.title = element_text(face = "bold", size = 15),
+           legend.text = element_text(face = "bold", size = 12))
```

- tidy: 코드를 사용자가 지정(혹은 `formatR::tidy_source()` 함수에 초기값으로 지정된 코드 정리 값)한 줄 당 문자 길이 등을 반영해 코드를 정리
 - `tidy=TRUE` 인 경우 자동으로 줄 바꿈

```
> # tidy = FALSE 인 경우 코드 출력 결과
> require(ggthemes) # ggtheme 패키지 불러오기
> require(ggpubr) # ggpubr 패키지 불러오기
> iris %>% ggplot(aes(x = Sepal.Length, y = Petal.Width, color = Species)) + geom_point(size = 5) +
```

```
> # tidy = TRUE 인 경우 코드 출력 결과
> require(ggthemes) # ggtheme 패키지 불러오기
> require(ggpubr) # ggpubr 패키지 불러오기
> iris %>%
+   ggplot(aes(x = Sepal.Length, y = Petal.Width, color = Species)) + geom_point(size = 5) +
+   theme_pubclean() + theme(axis.line = element_line(size = 0.8), legend.title = element_text(
+     face = "bold", size = 15), legend.text = element_text(face = "bold", size = 12))
```

그림(plot) 출력 관련 청크 옵션

TABLE 5.4: Plot 출력 관련 청크

| Chunk 옵션 | Default | 설명 |
|----------------------|---------|--|
| fig.align | default | 최종 문서에 plot 정렬 방식 (center/left/right) |
| fig.height/fig.width | 7 | 그림 크기(단위: 인치) |
| fig.cap | NULL | 그림 캡션(문자열 입력) |
| dpi | 72 | dot per inche: 출력 그림 해상도 |

알아두면 좋은 청크 형태

Setup 청크

- 일반적으로 Rmarkdown 문서는 YAML 해더 뒤에 전역적 청크 옵션 지정과 R 패키지를 불러오는 것으로 시작
- 청크 옵션은 `knitr::opts_chunk$set(청크 옵션 지정)` 형태로 지정 가능
- 다음은 RStudio에서 Rmd 문서 생성 시 맨 처음 나오는 코드 청크 예시임

```
```{r ex01-2, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```

```

- 일반적 활용 예시

```
```{r option-init, include=FALSE}
knitr::opts_chunk$set(root.dir = '../..', # 프로젝트 폴더 지정
 eval = TRUE,
 echo = FALSE,
 cache = FALSE,
 include = TRUE,
 tidy = TRUE,
 tidy.opts = list(blank=FALSE, width.cutoff=120), # 소스 출력
 message = FALSE,
 warning = FALSE,
 engine = "R", # Chunks will always have R code, unless noted
 error = TRUE,
 fig.path="Figures/", # Set the figure options
 fig.align = "center",
 fig.width = 7,
 fig.height = 7,
 fig.keep='all', fig.retina=2)
```

```

이미지 불러오기

```
```{r, fig.cap = "Taj Mahal"}
knitr::include_graphics("figures/taj.JPG", dpi = NA)
```

```



FIGURE 5.8: Taj Mahal

```
```{r, fig.cap = "Scatterplot of the car dataset"}  
cars %>%
 ggplot(aes(x = speed, y = dist)) +
 geom_point(size = 5) +
 theme_tufte(base_size = 15) # ggtheme::theme_tufte()
```
```

R 생성 도표 포함

테이블 삽입

- 가장 간단한 테이블은 knitr::kable() 함수를 통해 생성 가능
- kable() 함수는 가장 단순한 형태의 표만 생성하기 때문에 복잡한 표를 만들기에는 한계가 존재함
- 이를 보완하기 위해 다음과 같은 패키지 활용

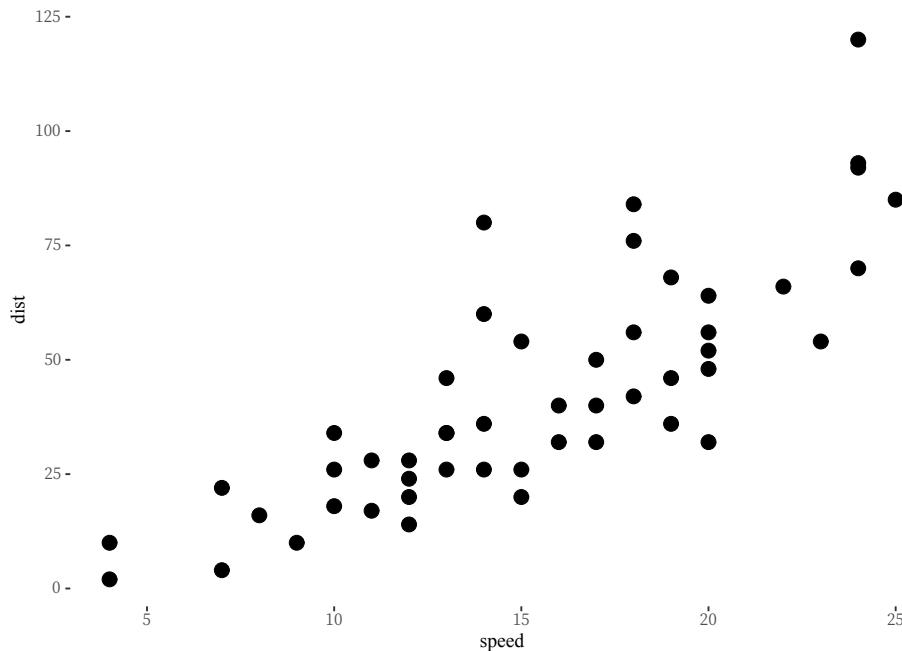


FIGURE 5.9: Scatterplot of the car dataset

– `kableExtra`: HTML 또는 LaTeX 용 표 생성

- * https://cran.r-project.org/web/packages/kableExtra/vignettes/awesome_table_in_html.html
- * https://cran.r-project.org/web/packages/kableExtra/vignettes/awesome_table_in_pdf.pdf

– `flextable` + `officer`: HTML, 워드 문서 표 작성

- * <https://davidgohel.github.io/flextable/>

```
```{r}
knitr::kable(head(iris))
```
```

| Sepal.Length | Sepal.Width | Petal.Length | Petal.Width | Species |
|--------------|-------------|--------------|-------------|---------|
| 5.1 | 3.5 | 1.4 | 0.2 | setosa |
| 4.9 | 3.0 | 1.4 | 0.2 | setosa |
| 4.7 | 3.2 | 1.3 | 0.2 | setosa |
| 4.6 | 3.1 | 1.5 | 0.2 | setosa |
| 5.0 | 3.6 | 1.4 | 0.2 | setosa |
| 5.4 | 3.9 | 1.7 | 0.4 | setosa |

5.4 인라인(inline) R 코드

- 문서의 모든 숫자를 인라인 R 코드를 통해 재현가능하게 생성 가능
- 인라인 R 코드는 `r` 과 ` ` 사이에 변수 계산 스크립트를 입력해 작성 가능
- 예를 들어 ‘r 10 + 4’ 는 14 출력
- 활용 예시

```
head(mtcars, 5)
```

| | mpg | cyl | disp | hp | drat | wt | qsec | vs | am | gear | carb |
|-------------------|------|-----|------|-----|------|-------|-------|----|----|------|------|
| Mazda RX4 | 21.0 | 6 | 160 | 110 | 3.90 | 2.620 | 16.46 | 0 | 1 | 4 | 4 |
| Mazda RX4 Wag | 21.0 | 6 | 160 | 110 | 3.90 | 2.875 | 17.02 | 0 | 1 | 4 | 4 |
| Datsun 710 | 22.8 | 4 | 108 | 93 | 3.85 | 2.320 | 18.61 | 1 | 1 | 4 | 1 |
| Hornet 4 Drive | 21.4 | 6 | 258 | 110 | 3.08 | 3.215 | 19.44 | 1 | 0 | 3 | 1 |
| Hornet Sportabout | 18.7 | 8 | 360 | 175 | 3.15 | 3.440 | 17.02 | 0 | 0 | 3 | 2 |

```
N <- nrow(mtcars)
```

`mtcars` 데이터셋에 포함된 자동차는 ‘r N ‘ 개다.

→

`mtcars` 데이터셋에 포함된 자동차는 32 개다.

5.5 YAML

- R Markdown 문서의 가장 처음에 정의하는 metadata
- .Rmd 파일을 .md 파일로 변환 후 최종 출력문서 생성 시 필요한 pandoc 의 옵션을 설정하는 것과 같은 의미임
- 일반적으로 문서 형태 및 생성을 위해 사용하는 R package (예: bookdown, officedown, rticles 등)에 따라 YAML 구성요소가 달라짐

기본 문법

- /#: 주석 처리
- YAML 문서의 시작과 끝은 --- 로 정의함
- 기본적으로 콜론(:)으로 구분된 태그(키): 값 쌍으로 구성됨 → key:
value
 - 여기서 콜론 바로 다음에는 반드시 공백문자가 있어야 함
- 한 key의 하위 키는 리스트 형태로 표현하고, 하위 키는 두 개 이상의
스페이스로 공백을 주어 표현

```
---
```

```
key : value
  subkey1: value1
  subkey2: value2
    subsubkey1: value3
---
```

R Markdown 기본 YAML 구조

```
---
```

```
title: "문서 제목" # 일반적으로 따옴표 사용
subtitle: "문서 부제목"
author: "문서 작성자"
date: "문서 작성일자"
output:
  - "html_document"
  - "word_document"
  - "pdf_document"
  - "md_document"
  - "isoslates_presentation"
  - "slidy_presentation"
  - "beamer_presentation"
bibliography: 참고문헌.bib # bibtex 서식 활용
```

```
.
```

```
.
```

```
.
```

- <https://bookdown.org/yihui/rmarkdown/documents.html> 에 자세한 예시 참고

5.6 참고문헌 인용

- 참고문헌 정보가 BibTeX 포맷으로 저장된 .bib 파일을 YAML에 선언 후 인용 가능

Bibtex 참고문헌 입력 형태

```
@article{Shea2014,
  author = {Shea, Nicholas and Boldt, Annika},
  journal = {Trends in Cognitive Sciences},
  pages = {186--193},
  title = {{Supra-personal cognitive control}},
  volume = {18},
  year = {2014},
  doi = {10.1016/j.tics.2014.01.006},
}
```

YAML에 bib 파일 지정

```
---
title: "Citation test"
bibliography: example.bib
output: html_document
---
```

- 참고문헌 표현: [@citation-identifier] 또는 @citation-identifier

This...

Blah blah [@Shea2014; @Lottridge2012].
Shea et al. says blah [-@Shea2014].
@Shea2014 says blah.
Blah blah [see @Shea2014, pp. 33-35; also
@Wu2016, ch. 1].

turns into this...

Blah blah (Shea et al. 2014; Lottridge et al. 2012).
Shea et al. says blah (2014).
Shea et al. (2014) says blah.
Blah blah (see Shea et al. 2014, 33–35; also Wu 2016, ch. 1).

- BibTeX 포맷은 Google Scholar에서 쉽게 획득 가능
- Citation 스타일은 YAML 헤더에 `csl: style.csl`로 변경 가능하며 Zotero⁵에서 .csl 파일 다운로드 가능

⁵<https://www.zotero.org>

6

R 외부 데이터 입출력



R 기본 함수를 이용해서 데이터 저장 파일의 가장 기본적인 형태인 텍스트 파일을 읽고 저장하는 방법에 대해 먼저 살펴봄. Base R에서 외부 데이터를 읽고 저장을 위한 함수는 매우 다양하지만 가장 많이 사용되고 있는 함수들에 대해 살펴볼 것임

- 기본 R(base R)에서 제공하는 함수를 이용해 외부 데이터를 읽고, 내보내고, 저장하는 방법에 대해 살펴봄.
- 가장 일반적인 형태의 데이터는 보통 텍스트 파일 형태로 저장되어 있음, 일반적으로
 - 첫 번째 줄: 변수명
 - 두 번째 줄부터: 데이터 입력

```
id sex age edulev height
1 Male 65 12 168
2 Female 74 9 145
3 Male 61 12 171
4 Male 85 6 158
5 Female 88 0 134
```

- 데이터의 자료값과 자료값을 구분하는 문자를 구분자(separator)라고 하며 주로 공백(), 콤마(,), tab 문자(\t) 등이 사용됨

- 주로 확장자 명이 *.txt 이며, 콤마 구분자인 경우 보통은 *.csv (comma separated values)로 저장

#titanic3.csv 파일 일부

```
"pclass","survived","name","sex","age",
1,1,"Allen, Miss. Elisabeth Walton","female"
1,1,"Allison, Master. Hudson Trevor","male"
1,0,"Allison, Miss. Helen Loraine", "female"
1,0,"Allison, Mr. Hudson Joshua Creighton","male"
1,0,"Allison, Mrs. Hudson J C (Bessie Waldo Daniels)","female"
```

6.1 텍스트 파일 입출력



외부 데이터를 불러온다는 것은 외부에 저장되어 있는 파일을 R 작업환경으로 읽어온다는 의미이기 때문에, 현재 작업공간의 작업 디렉토리(working directory) 확인이 필요.

- read.table()/write.table():
 - 가장 범용적으로 외부 텍스트 데이터를 R 작업공간으로 데이터 프레임으로 읽고 저장하는 함수
 - 텍스트 파일의 형태에 따라 구분자 지정 가능

```
# read.table(): 텍스트 파일 읽어오기
read.table(
  file, # 파일명. 일반적으로 폴더명 구분자
  # 보통 folder/파일이름.txt 형태로 입력
  header = FALSE, # 첫 번째 행을 헤더(변수명)으로 처리할지 여부
  sep = "", # 구분자 ",", "\t" 등의 형태로 입력
  comment.char = "#", # 주석문자 지정
  stringsAsFactors = TRUE, # 문자형 변수를 factor으로 변환할지 여부
  encoding = "unknown" # 텍스트의 encoding 보통 CP949 또는 UTF-8
  # 한글이 입력된 데이터가 있을 때 사용
)
```

- `read.table()` 예시



예시에 사용된 데이터들은 Clinical trial data analysis using R (Chen and Peace, 2010)에서 제공되는 데이터임.

```
# tab 구분자 데이터 불러오기
# dataset 폴더에 저장되어 있는 DBP.txt 파일 읽어오기
dbp <- read.table("dataset/DBP.txt", sep = "\t", header = TRUE)
str(dbp)
```

```
'data.frame': 40 obs. of 9 variables:
 $ Subject: int 1 2 3 4 5 6 7 8 9 10 ...
 $ TRT     : chr "A" "A" "A" "A" ...
 $ DBP1    : int 114 116 119 115 116 117 118 120 114 115 ...
 $ DBP2    : int 115 113 115 113 112 112 111 115 112 113 ...
 $ DBP3    : int 113 112 113 112 107 113 100 113 113 108 ...
 $ DBP4    : int 109 103 104 109 104 104 109 102 109 106 ...
```

```
$ DBP5    : int  105 101 98 101 105 102 99 102 103 97 ...
$ Age     : int  43 51 48 42 49 47 50 61 43 51 ...
$ Sex     : chr  "F" "M" "F" "F" ...
```

```
# 문자형 값들을 factor로 변환하지 않는 경우
dbp2 <- read.table("dataset/DBP.txt", sep = "\t",
                     header = TRUE,
                     stringsAsFactors = FALSE)

str(dbp2)
```

```
'data.frame': 40 obs. of 9 variables:
 $ Subject: int 1 2 3 4 5 6 7 8 9 10 ...
 $ TRT     : chr "A" "A" "A" "A" ...
 $ DBP1    : int 114 116 119 115 116 117 118 120 114 115 ...
 $ DBP2    : int 115 113 115 113 112 112 111 115 112 113 ...
 $ DBP3    : int 113 112 113 112 107 113 100 113 113 108 ...
 $ DBP4    : int 109 103 104 109 104 104 109 102 109 106 ...
 $ DBP5    : int 105 101 98 101 105 102 99 102 103 97 ...
 $ Age     : int 43 51 48 42 49 47 50 61 43 51 ...
 $ Sex     : chr "F" "M" "F" "F" ...
```

```
# 데이터 형태 파악
head(dbp)
```

| | Subject | TRT | DBP1 | DBP2 | DBP3 | DBP4 | DBP5 | Age | Sex |
|---|---------|-----|------|------|------|------|------|-----|-----|
| 1 | 1 | A | 114 | 115 | 113 | 109 | 105 | 43 | F |
| 2 | 2 | A | 116 | 113 | 112 | 103 | 101 | 51 | M |
| 3 | 3 | A | 119 | 115 | 113 | 104 | 98 | 48 | F |
| 4 | 4 | A | 115 | 113 | 112 | 109 | 101 | 42 | F |
| 5 | 5 | A | 116 | 112 | 107 | 104 | 105 | 49 | M |
| 6 | 6 | A | 117 | 112 | 113 | 104 | 102 | 47 | M |

6.1 텍스트 파일 입출력

5

```
# 콤마 구분자 데이터 불러오기  
# dataset 폴더에 저장되어 있는 diabetes_csv.txt 파일 읽어오기  
diab <- read.table("dataset/diabetes_csv.txt", sep = ",", header = TRUE)  
str(diab)
```

```
'data.frame': 403 obs. of 19 variables:  
 $ id      : int  1000 1001 1002 1003 1005 1008 1011 1015 1016 1022 ...  
 $ chol     : int  203 165 228 78 249 248 195 227 177 263 ...  
 $ stab.glu: int  82 97 92 93 90 94 92 75 87 89 ...  
 $ hdl      : int  56 24 37 12 28 69 41 44 49 40 ...  
 $ ratio    : num  3.6 6.9 6.2 6.5 8.9 ...  
 $ glyhb   : num  4.31 4.44 4.64 4.63 7.72 ...  
 $ location: chr  "Buckingham" "Buckingham" "Buckingham" "Buckingham" ...  
 $ age      : int  46 29 58 67 64 34 30 37 45 55 ...  
 $ gender   : chr  "female" "female" "female" "male" ...  
 $ height   : int  62 64 61 67 68 71 69 59 69 63 ...  
 $ weight   : int  121 218 256 119 183 190 191 170 166 202 ...  
 $ frame    : chr  "medium" "large" "large" "large" ...  
 $ bp.1s    : int  118 112 190 110 138 132 161 NA 160 108 ...  
 $ bp.1d    : int  59 68 92 50 80 86 112 NA 80 72 ...  
 $ bp.2s    : int  NA NA 185 NA NA NA 161 NA 128 NA ...  
 $ bp.2d    : int  NA NA 92 NA NA NA 112 NA 86 NA ...  
 $ waist    : int  29 46 49 33 44 36 46 34 34 45 ...  
 $ hip      : int  38 48 57 38 41 42 49 39 40 50 ...  
 $ time.ppn: int  720 360 180 480 300 195 720 1020 300 240 ...
```

```
head(diab)
```

| | id | chol | stab.glu | hdl | ratio | glyhb | location | age | gender | height | weight | frame |
|---|------|------|----------|-----|-------|-------|------------|-----|--------|--------|--------|--------|
| 1 | 1000 | 203 | 82 | 56 | 3.6 | 4.31 | Buckingham | 46 | female | 62 | 121 | medium |
| 2 | 1001 | 165 | 97 | 24 | 6.9 | 4.44 | Buckingham | 29 | female | 64 | 218 | large |

| | | | | | | | | | | | | |
|--|------|-----|-----|----|-----|------|------------|-----|--------|----|-----|--------|
| 3 | 1002 | 228 | 92 | 37 | 6.2 | 4.64 | Buckingham | 58 | female | 61 | 256 | large |
| 4 | 1003 | 78 | 93 | 12 | 6.5 | 4.63 | Buckingham | 67 | male | 67 | 119 | large |
| 5 | 1005 | 249 | 90 | 28 | 8.9 | 7.72 | Buckingham | 64 | male | 68 | 183 | medium |
| 6 | 1008 | 248 | 94 | 69 | 3.6 | 4.81 | Buckingham | 34 | male | 71 | 190 | large |
| bp.1s bp.1d bp.2s bp.2d waist hip time.ppn | | | | | | | | | | | | |
| 1 | 118 | 59 | NA | NA | 29 | 38 | | 720 | | | | |
| 2 | 112 | 68 | NA | NA | 46 | 48 | | 360 | | | | |
| 3 | 190 | 92 | 185 | 92 | 49 | 57 | | 180 | | | | |
| 4 | 110 | 50 | NA | NA | 33 | 38 | | 480 | | | | |
| 5 | 138 | 80 | NA | NA | 44 | 41 | | 300 | | | | |
| 6 | 132 | 86 | NA | NA | 36 | 42 | | 195 | | | | |

```
# Encoding이 다른 경우(한글데이터 포함):
# 한글자체 사전 데이터 (CP949 encoding으로 저장)
# tab 구분자 데이터 사용
# UTF-8로 읽어오기

herb <- read.table("dataset/herb_dic_sample.txt", sep = "\t",
                    header = TRUE,
                    encoding = "UTF-8",
                    stringsAsFactors = FALSE)

head(herb)

# CP949로 읽어오기
herb <- read.table("dataset/herb_dic_sample.txt", sep = "\t",
                    header = TRUE,
                    encoding = "CP949",
                    stringsAsFactors = FALSE)

head(herb)
```

- `read.table() + textConnection()`: 웹사이트나 URL에 있는 데이터를 Copy + Paste 해서 읽어올 경우 유용하게 사용

6.1 텍스트 파일 입출력

7

- `textConnection()`: 텍스트에서 한 줄씩 읽어 문자형 벡터처럼 인식할 수 있도록 해주는 함수

```
# Plasma E||O|E|: http://lib.stat.cmu.edu/datasets/Plasma_Retinol
input1 <- ("64 2 2 21.4838 1 1298.8 57 6.3 0 170.3 1945 890 200 915
76 2 1 23.87631 1 1032.5 50.1 15.8 0 75.8 2653 451 124 727
38 2 2 20.0108 2 2372.3 83.6 19.1 14.1 257.9 6321 660 328 721
40 2 2 25.14062 3 2449.5 97.5 26.5 0.5 332.6 1061 864 153 615
72 2 1 20.98504 1 1952.1 82.6 16.2 0 170.8 2863 1209 92 799
40 2 2 27.52136 3 1366.9 56 9.6 1.3 154.6 1729 1439 148 654
65 2 1 22.01154 2 2213.9 52 28.7 0 255.1 5371 802 258 834
58 2 1 28.75702 1 1595.6 63.4 10.9 0 214.1 823 2571 64 825
35 2 1 23.07662 3 1800.5 57.8 20.3 0.6 233.6 2895 944 218 517
55 2 2 34.96995 3 1263.6 39.6 15.5 0 171.9 3307 493 81 562")

input2 <- ("AGE: Age (years)
SEX: Sex (1=Male, 2=Female).
SMOKSTAT: Smoking status (1=Never, 2=Former, 3=Current Smoker)
QUETELET: Quetelet (weight/(height^2))
VITUSE: Vitamin Use (1=Yes, fairly often, 2=Yes, not often, 3=No)
CALORIES: Number of calories consumed per day.
FAT: Grams of fat consumed per day.
FIBER: Grams of fiber consumed per day.
ALCOHOL: Number of alcoholic drinks consumed per week.
CHOLESTEROL: Cholesterol consumed (mg per day).
BETADIET: Dietary beta-carotene consumed (mcg per day).
RETDIET: Dietary retinol consumed (mcg per day)
BETAPLASMA: Plasma beta-carotene (ng/ml)
RETPLASMA: Plasma Retinol (ng/ml)")

plasma <- read.table(textConnection(input1), sep = "\t", header = F)
```

```
codebook <- read.table(textConnection(input2), sep = ":", header = F)
varname <- gsub("^\\s+", "", codebook$V1) # 변수명 지정

names(plasma) <- varname
head(plasma)
```

| | AGE | SEX | SMOKSTAT | QUETELET | VITUSE | CALORIES | FAT | FIBER | ALCOHOL | CHOLESTEROL |
|---|----------|---------|------------|-----------|--------|----------|------|-------|---------|-------------|
| 1 | 64 | 2 | 2 | 21.48380 | 1 | 1298.8 | 57.0 | 6.3 | 0.0 | 170.3 |
| 2 | 76 | 2 | 1 | 23.87631 | 1 | 1032.5 | 50.1 | 15.8 | 0.0 | 75.8 |
| 3 | 38 | 2 | 2 | 20.01080 | 2 | 2372.3 | 83.6 | 19.1 | 14.1 | 257.9 |
| 4 | 40 | 2 | 2 | 25.14062 | 3 | 2449.5 | 97.5 | 26.5 | 0.5 | 332.6 |
| 5 | 72 | 2 | 1 | 20.98504 | 1 | 1952.1 | 82.6 | 16.2 | 0.0 | 170.8 |
| 6 | 40 | 2 | 2 | 27.52136 | 3 | 1366.9 | 56.0 | 9.6 | 1.3 | 154.6 |
| | BETADIET | RETDIET | BETAPLASMA | RETPLASMA | | | | | | |
| 1 | 1945 | 890 | 200 | 915 | | | | | | |
| 2 | 2653 | 451 | 124 | 727 | | | | | | |
| 3 | 6321 | 660 | 328 | 721 | | | | | | |
| 4 | 1061 | 864 | 153 | 615 | | | | | | |
| 5 | 2863 | 1209 | 92 | 799 | | | | | | |
| 6 | 1729 | 1439 | 148 | 654 | | | | | | |

- `write.table()`: R의 객체(벡터, 행렬, 데이터 프레임)를 저장 후 외부 텍스트 파일로 내보내기 위한 함수

```
# write.table() R 객체를 텍스트 파일로 저장하기
write.table(
  data_obj, # 저장할 객체 이름
  file, # 저장할 위치 및 파일명 또는
        # 또는 "파일쓰기"를 위한 연결 명칭
  sep, # 저장 시 사용할 구분자
```

```
row.names = TRUE # 행 이름 저장 여부  
)
```

- 예시

```
# 위에서 읽어온 plasma 객체를 dataset/plasma.txt로 내보내기  
# 행 이름은 생략, tab으로 데이터 구분  
  
write.table(plasma, "dataset/plasma.txt",  
            sep = "\t", row.names = F)
```

- 파일명 대신 Windows clipboard로 내보내기 가능

```
# clipboard로 복사 후 excel 시트에 해당 데이터 붙여넣기  
# Ctrl + V  
  
write.table(plasma, "clipboard",  
            sep = "\t", row.names = F)
```

- `read.csv()`/`write.csv()`: `read.table()` 함수의 wrapper 함수로 구
분자 인수 `sep`이 콤마(,)로 고정(예시 생략)

6.2 R 바이너리(binary) 파일 입출력

R 작업공간에 존재하는 한 개 이상의 객체들을 저장하고 읽기 위한 함수

- R 데이터 관련 바이너리 파일은 한 개 이상의 객체가 저장된 바이너리 파일인 경우 *.Rdata 형태를 갖고, 단일 객체를 저장할 경우 보통 *.rds 파일 확장자로 저장
- *.Rdata 입출력 함수
 - `load()`: *.Rdata 파일 읽어오기
 - `save()`: 한 개 이상 R 작업공간에 존재하는 객체를 .Rdata 파일로 저장
 - `save.image()`: 현재 R 작업공간에 존재하는 모든 객체를 .Rdata 파일로 저장

```
# 현재 작업공간에 존재하는 모든 객체를 "output" 폴더에 저장
# output 폴더가 존재하지 않는 경우 아래 명령 실행
# dir.create("output")
ls()
```

```
[1] "codebook"          "dbp"                 "dbp2"                "def.chunk.hook"
[5] "diab"               "hook_output"        "input1"              "input2"
[9] "plasma"             "varname"
```

```
save.image(file = "output/all_obj.Rdata")

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

```
character(0)
```

6.2 R 바이너리(binary) 파일 입출력

11

```
# 저장된 binary 파일(all_obj.Rdata) 불러오기  
load("output/all_obj.Rdata")  
ls()
```

```
[1] "codebook"      "dbp"           "dbp2"          "def.chunk.hook"  
[5] "diab"          "hook_output"    "input1"         "input2"  
[9] "plasma"        "varname"
```

```
# dnp, plasma 데일리만 output 폴더에 sub_obj.Rdata로 저장  
save(dbp, plasma, file = "output/sub_obj.Rdata")  
rm(list = c("dbp", "plasma"))  
ls()
```

```
[1] "codebook"      "dbp2"          "def.chunk.hook" "diab"  
[5] "hook_output"   "input1"         "input2"         "varname"
```

```
# sub_obj.Rdata 파일 불러오기  
load("output/sub_obj.Rdata")  
ls()
```

```
[1] "codebook"      "dbp"           "dbp2"          "def.chunk.hook"  
[5] "diab"          "hook_output"    "input1"         "input2"  
[9] "plasma"        "varname"
```

- *.rds 입출력 함수

- readRDS() / saveRDS(): 단일 객체가 저장된 *.rds 파일을 읽거나 저장

- 대용량 데이터를 다룰 때 유용함
- `read.table()` 보다 데이터를 읽는 속도가 빠르며, 다른 확장자 명의 텍스트 파일보다 높은 압축율을 보임

```
# 대용량 파일 dataset/pulse.csv 불러오기
# system.time(): 명령 실행 시가 계산 함수
system.time(pulse <- read.csv("dataset/pulse.csv", header = T))
```

사용자 시스템 elapsed
 4.676 0.026 4.704

```
# saveRDS() 함수를 이용해 output/pulse.rds 파일로 저장
saveRDS(pulse, "output/pulse.rds")
rm(pulse); ls()
```

```
[1] "codebook"      "dbp"           "dbp2"          "def.chunk.hook"
[5] "diab"          "hook_output"    "input1"         "input2"
[9] "plasma"        "varname"
```

```
system.time(pulse <- readRDS("output/pulse.rds"))
```

사용자 시스템 elapsed
 0.078 0.001 0.078

6.3 Excel 파일 입출력

- R에서 기본적으로 제공하는 파일 입출력 함수는 대부분 텍스트 파일 (*.txt, *.csv, *.tsv¹)을 대상으로 하고 있음
- **readr** 패키지에서도 이러한 원칙은 유지됨
- Excel 파일을 R로 읽어오기(과거 방법)
 - *.xls 또는 *.xlsx 파일을 엑셀로 읽은 후 해당 데이터를 위 텍스트 파일 형태로 내보낸 후 해당 파일을 R로 읽어옴
 - **xlsx** 패키지 등을 이용해 엑셀 파일을 직접 읽어올 수 있으나, Java 기반으로 개발된 패키지이기 때문에 Java Runtime Environment를 운영체제에 설치해야만 작동
- 최근 tidyverse 중 하나인 **readxl** 패키지를 이용해 간편하게 R 작업환경에 엑셀 파일을 읽어오는 것이 가능(Hadley Wickham이 개발...)
 - tidyverse의 한 부분임에도 불구하고 tidyverse 패키지 번들에는 포함되어 있지 않기 때문에 별도 설치 필요

readxl 패키지 구성 주요 함수

- **read_xls()**, **read_xlsx()**, **read_excel**: 엑셀 파일을 읽어오는 함수로 각각 Excel 97 ~ 2003, Excel 2007 이상, 또는 버전 상관 없이 저장된 엑셀 파일에 접근함
- **excel_sheets()**: 엑셀 파일 내 시트 이름 추출 → 한 엑셀 파일의 복수 시트에 데이터가 저장되어 있는 경우 활용

¹tab separated values

- 예시: 2020년 4월 23일 COVID-19 유병률 데이터 (Our World in Data²)

```
read_xlsx(
  path, # Excel 폴더 및 파일 이름
  sheet = NULL, # 불러올 엑셀 시트 이름
  # default = 첫 번째 시트
  col_names = TRUE, # read_csv()의 인수와 동일한 형태 입력
  col_types = NULL # read_csv()의 인수와 동일한 형태 입력
)
```

```
# 2020년 4월 21일자 COVID-19 국가별 유병률 및 사망률 집계 자료
# dataset/owid-covid-data.xlsx 파일 불러오기
# install.packages("readxl")
require(readxl)
```

필요한 패키지를 로딩중입니다: readxl

```
covid19 <- read_xlsx("dataset/covid-19-dataset/owid-covid-data.xlsx")
covid19
```

```
# A tibble: 14,315 x 16
  iso_code location date      total_cases new_cases total_deaths new_deaths
  <chr>     <chr>   <chr>        <dbl>      <dbl>       <dbl>      <dbl>
1 ABW       Aruba   2020-03-13      2          2          0          0
2 ABW       Aruba   2020-03-20      4          2          0          0
3 ABW       Aruba   2020-03-24     12          8          0          0
```

²<https://github.com/owid/covid-19-data/tree/master/public/data>

```
4 ABW      Aruba    2020-03-25      17      5      0      0
5 ABW      Aruba    2020-03-26      19      2      0      0
6 ABW      Aruba    2020-03-27      28      9      0      0
7 ABW      Aruba    2020-03-28      28      0      0      0
8 ABW      Aruba    2020-03-29      28      0      0      0
9 ABW      Aruba    2020-03-30      50     22      0      0
10 ABW     Aruba    2020-04-01      55      5      0      0
# ... with 14,305 more rows, and 9 more variables:
#   total_cases_per_million <dbl>, new_cases_per_million <dbl>,
#   total_deaths_per_million <dbl>, new_deaths_per_million <dbl>,
#   total_tests <dbl>, new_tests <dbl>, total_tests_per_thousand <dbl>,
#   new_tests_per_thousand <dbl>, tests_units <chr>
```

```
# 여러 시트를 동시에 불러올 경우
# dataset/datR4CTDA.xlsx 의 모든 시트 불러오기
path <- "dataset/datR4CTDA.xlsx"
sheet_name <- excel_sheets(path)
dL <- lapply(sheet_name, function(x) read_xlsx(path, sheet = x))
names(dL) <- sheet_name
```



7

제어문(Control Structure)

Sketch

- 프로그램 안의 특정 구문을 주어진 조건에 맞게 실행 여부를 제어하거나 동일한 작업을 반복할 수 있을까?
- 프로그램을 통해 특정 목적을 위한 나만의 함수를 만들 수 있을까?

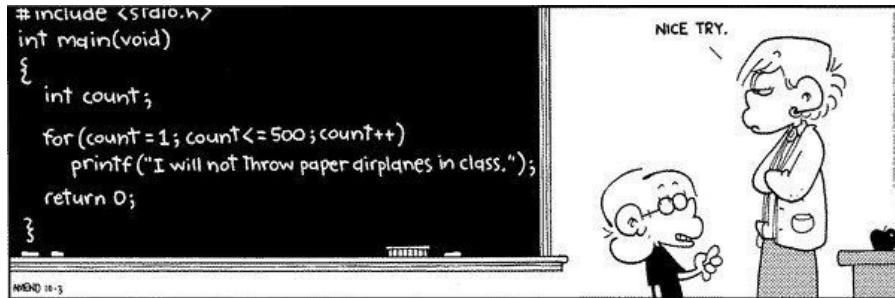


FIGURE 7.1: Flow-control example (<https://homerhanumat.github.io/r-notes/flow.html>)



참고: 본 장의 내용은 데이터과학 민주화¹와 Beginning Computer Programming with R²의 내용을 기반으로 재구성함

7.1 조건문(Conditionals)

- if 구문을 통해 조건문 생성
- 불린 표현식(boolean expression): 참(TRUE) 또는 거짓(FALSE) 두 값 중 하나로 값이 도출되는 표현식³
 - 비교 연산자(comparison operators)
 - * 같다, 같지 않다, 크다 등을 표현하기 위한 연산자
 - * ==, !=, >, <, >=, <=
 - 논리 연산자(logical operator)
 - * AND (&, &&), OR (|, ||), NOT (!)

```
x <- 10; y <- 13

# x가 2의 배수이고 y가 3의 배수
# 두 조건이 모두 참이여야 참
x %% 2 == 0 & y %% 3 == 0

# x가 2의 배수이거나 y가 3의 배수
# 두 개 조건 중 하나만 참을 만족하면 참임
x %% 2 == 0 | y %% 3 == 0

# NOT (x > y)
!(x > y) # 부정에 부정은 참
```

[1] FALSE

[1] TRUE

[1] TRUE

³비교 및 논리 연산자(통계프로그래밍언어 2.1.4절 참고⁴)

7.1.1 기본 구문

if (조건) 표현식

↳ 괄호 안 조건을 만족하면 표현식을 실행하고 조건을 만족하지 않으면 실행하지 않음

```
x <- 10
if (x > 0) {
  print("x is positive")
}

x <- -5
if (x > 0) {
  print("x is positive")
}
```

[1] "x is positive"

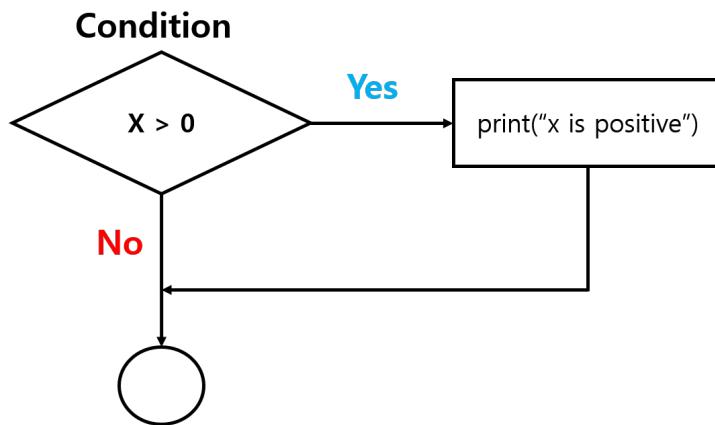


FIGURE 7.2: if 구문 기본 flow-chart

- **if 구문의 사용 규칙**

- if 문은 조건을 정의하는 헤더 부분((),)과 표현식이 위치하는 몸통 블록(body block, {표현식})으로 구성됨
- (),에 표현되는 조건은 벡터가 아닌 단일 값으로 나타내야 함.
- {},의 표현 또는 문장이 한 줄인 경우 블록 지정이 필요하지 않지만, 두 줄 이상인 경우 if 문의 범위를 지정해줘야 하기 때문에 꼭 중괄호(curly bracket, {})가 사용되어야 함.

```
# 조건문 사용 예시
x <- c(TRUE, FALSE, FALSE)
y <- c(TRUE, TRUE, FALSE)
z <- "Both TRUE!!"

if (x[1] & y[1]) print(z) # x, y 첫 번째 원소만 사용
if (x && y) print(z) # 강제로 첫 번째 원소만 사용
if (x & y) print(z) # 경고 표시
```

Warning in if (x & y) print(z): length > 1 이라는 조건이 있고, 첫번째 요소만이 사용될 것입니다

```
[1] "Both TRUE!!"
[1] "Both TRUE!!"
[1] "Both TRUE!!"
```

대안 실행(alternative execution)

- 두 가지 경우가 존재하고 조건에 따라 어떤 명령을 실행할지를 결정
- if와 else로 표현 가능

7.1 조건문(Conditionals)

5

- 조건에 따라 실행이 분기(branch) 되기 때문에 if-else 구문을 분기 문이라고도 함
- else 는 if 조건을 배제(exclusive)한 나머지 경우이기 때문에 조건을 따로 지정하지 않으면, if와 동일하게 중괄호 내에 표현되어야 함

```
x <- 9
if (x %% 2 == 0) {
  print("x is even")
} else {
  print("x is odd")
}
```

```
[1] "x is odd"
```

7.1.2 연쇄 조건문(chained condition)

- 두 가지 이상의 분기가 존재하는 경우 조건 표현식
- 연쇄 조건문의 표현은 아래와 같음

```
if (조건1) {
  표현식1
  ...
} else if (조건2) {
  표현식2
  ...
} else {
  표현식3
  ...
}
```

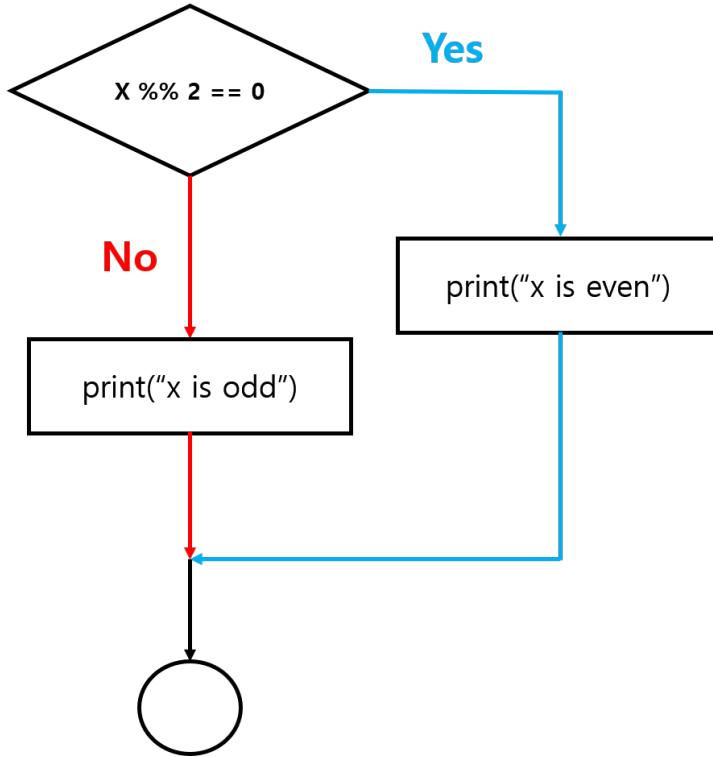


FIGURE 7.3: 대안실행(if-else 구문) flow-chart

```

x <- 5; y <- 10
if (x < y) {
    print("x is less than y")
} else if (x > y) {
    print("x is greater than y")
} else {
    print("x is equal to y")
}
  
```

[1] "x is less than y"

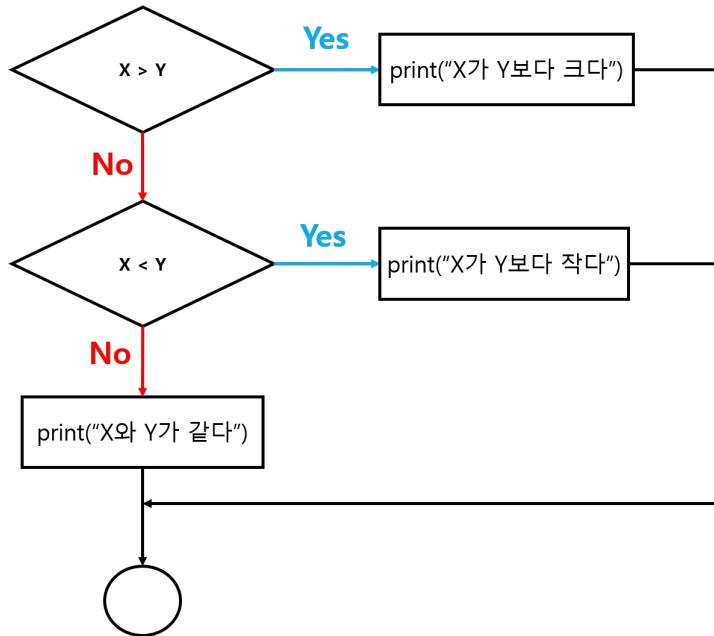


FIGURE 7.4: 연쇄조건(if-else if-else 구문) flow-chart

7.1.3 중첩 조건문(nested contition)

- 하나의 조건문 내부에 하위 조건식이 존재하는 형태

```

if (조건1) {
    표현식1
    ...
} else {
    if (조건2) {
        표현식2
        ...
    } else {
        표현식3
        ...
    }
}
  
```

```

}
}
```

```

x <- 10; y <- 10
if (x == y) {
    print("x is equal to y")
} else {
    if (x > y) {
        print("x is greater than y")
    } else {
        print("x is less than y")
    }
}
```

[1] "x is equal to y"

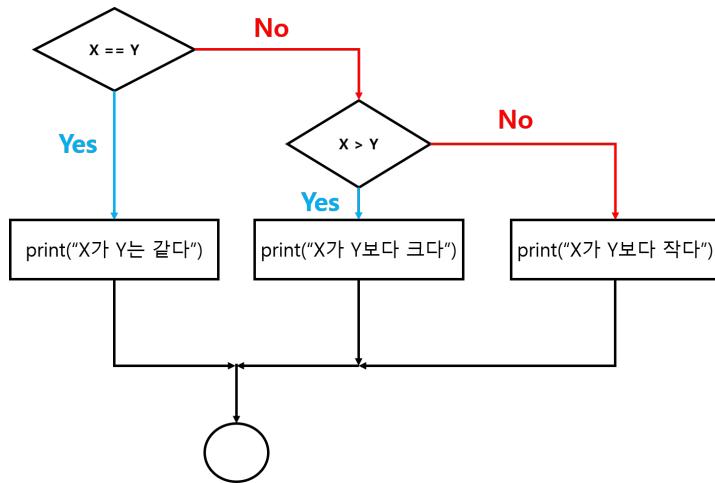


FIGURE 7.5: 중첩 조건문 flow-chart

7.1 조건문(Conditionals)

9



- 중첩 조건문은 코드의 가독성을 떨어뜨리기 때문에 피하는 것을 권장
- 중첩 조건문을 피하기 위한 한 가지 방법은 논리 연산자를 활용

```
# 중첩조건
x <- 58
if (x > 0) {
  if (x < 10) {
    print("x는 한 자리 양수")
  } else {
    if (x < 100) {
      print("x는 두 자리 양수")
    } else {
      print("x는 세 자리 이상 양수")
    }
  }
}
```

[1] "x는 두 자리 양수"

```
# 연쇄 조건
x <- 2020
if (x > 0 & x < 10) {
  print("x는 한 자리 양수")
} else if (x >=10 & x < 100) {
  print("x는 두 자리 양수")
} else {
  print("x는 세 자리 이상 양수")
}
```

[1] "x는 세 자리 이상 양수"

7.1.4 `ifelse()` 함수

- `if-else` 구문을 사용하기 쉽게 구현된 R 내장 함수
- `if-else` 구문과 다르게 조건 부분에 한 값(스칼라)이 아닌 논리형 벡터를 입력값으로 받아 조건에 따른 값(벡터)을 반환

```
# ifelse() 함수 인수
# help(ifelse) 참고

ifelse(
  test, 조건에 따른 논리형 벡터
  yes,  test에 정의한 조건이 참인 경우 새로운 벡터에 대입할 값
  no,   test 조건이 거짓인 경우 대입할 값
)
```

- 사용 예시

```
# 평균이 23이고 표준편차가 5인 정규분포로부터 30개의 난수 추출
set.seed(12345)
bmi <- rnorm(30, 23, 5)
bmi_cat <- ifelse(bmi < 25, "normal", "overweight")
bmi_cat
```

```
[1] "overweight" "overweight" "normal"      "normal"      "overweight"
[6] "normal"       "overweight" "normal"      "normal"      "normal"
[11] "normal"       "overweight" "normal"      "overweight"  "normal"
[16] "overweight"  "normal"     "normal"      "overweight"  "normal"
[21] "overweight"  "overweight" "normal"      "normal"      "normal"
[26] "overweight"  "normal"     "overweight"  "overweight"  "normal"
```

```
# ifelse() 함수를 연쇄조건문 처럼 사용할 수 있다
bmi_cat2 <- ifelse(bmi < 18.5, "underweight",
                     ifelse(bmi < 24.9, "normal",
                           ifelse(bmi < 29.9, "overweight", "obesity")))
bmi_cat2
```

```
[1] "overweight" "overweight" "normal"      "normal"      "overweight"
[6] "underweight" "overweight"  "normal"      "normal"      "underweight"
[11] "normal"       "obesity"     "normal"      "overweight"   "normal"
[16] "overweight"   "normal"      "normal"      "overweight"   "normal"
[21] "overweight"   "obesity"     "normal"      "underweight"  "underweight"
[26] "obesity"      "normal"      "overweight"   "overweight"   "normal"
```

7.2 반복문(Looping)

Prerequisite

- 프로그램 또는 알고리즘 구현 시 특정 문장 또는 표현을 반복해야만 하는 상황이 발생
- 특히 시뮬레이션 시 반복문은 거의 필수적임
- 반복문을 통해 코딩의 효율을 극대화 할 수 있음
- 반복문은 특정 변수의 값을 갱신(update) 하기 위해 주로 사용

```
x <- x + 1 # 현재 값에 1을 더해서 x를 새로운 값으로 update
```

- 통상적으로 특정 변수의 값을 갱신하기 위해 변수 값을 초기화(initialize)

```
x <- 0 # x 변수 초기화  
x <- x + 1
```

- 몇 번 반복이라는 정의가 없는 상태에서 특정 조건이 거짓(FALSE)이 될 때 까지 계속 반복

7.2.1 repeat 구문

repeat 표현식

- repeat 다음에 오는 표현식을 무한 반복(infinite loop)

```
repeat print("무한 루프에 걸림...ESC 키 누르시오!!")
```

```
[1] "무한 루프에 걸림...ESC 키 누르시오!!"  
...  
...
```

- 특정 작업에 대해 블록을 지정(중괄호)하고 블록 안에 표현 가능
- 일반적으로 특정 조건(if (조건) break)을 두어 무한루프에서 탈출

- if 문의 조건은 언제 반복이 끝날지를 제어하는 변수로 반복변수 (iteration variable) 이라고도 함
- 언제까지(until) 반복(repeat) → REPEAT-UNTIL 구문으로 표현

```
repeat {  
    표현식 1  
    if (조건) break  
    반복변수 update  
}
```

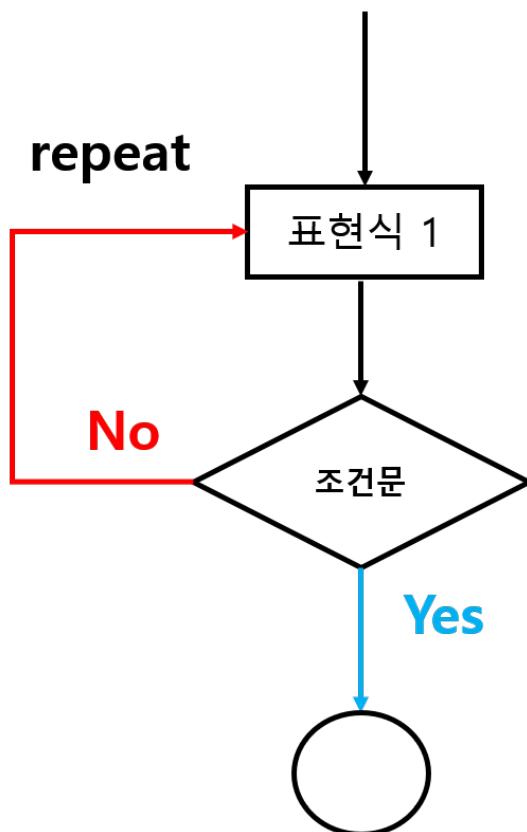


FIGURE 7.6: REPEAT 구문 flow-chart

```
# REPEAT-UNTIL 예시 1
# 1:100 까지 합 계산 함수
tot <- 0; i <- 1 # 사용 변수 초기화 (update 변수)
repeat {
  tot <- tot + i
  if (i >= 100) break # i는 반복 변수
  i <- i + 1
}
tot
# check
sum(1:100)
```

[1] 5050

[1] 5050

1. tot에 i를 더한 후 i 가 조건을 만족하는지 확인
2. 조건에 부합하지 않으면 다음 문장 실행(i에 1을 증가 후 업데이트)
 1. 의 작업을 반복(loop)
3. i가 조건에 부합하면 반복 종료

```
# REPEAT 예시 2
# 1에서 20 사이 숫자 알아맞추기 게임
set.seed(1)
n <- 20
number <- sample(1:n, size = 1)
cat("1에서 ", n, "까지 숫자 알아 맞추기", sep = "")
repeat {
  guess <- readline("어떤 숫자를 생각하시나요? (종료: q 입력) ")
```

```
if (guess == "q") {  
    cat("재미가 없나봐요.\n")  
    break  
} else if (as.numeric(guess) == number) {  
    cat("천재인데요?ㅋㅋㅋ")  
    break  
}  
# 틀리면 계속 반복  
}
```

1. guess에 readline() 으로부터 값 입력
2. guess 값이 q 이면 종료
3. guess 값이 number 와 일치하면 종료
4. 2와 3. 조건에 부합하지 않으면 guess 값을 반복적으로 입력

어떤 숫자를 생각하시나요? (종료: q 입력) 1

어떤 숫자를 생각하시나요? (종료: q 입력) 2

어떤 숫자를 생각하시나요? (종료: q 입력) 3

천재인데요?ㅋㅋㅋ

7.2.2 while 구문

while (조건) 표현식 ...

- while에 지정된 조건이 참이면 계속해서 반복

- repeat는 반복이 처음부터 시작되는 반면, while 문은 조건을 먼저 평가한 후 반복이 시작됨.
- while (FALSE)인 경우 루프 본문 코드가 실행되지 않음
- while (TRUE)는 repeat 구문과 동일
- while문의 일반적 형태

```
while (조건) {
    표현식 1
    반복변수 update
}
```

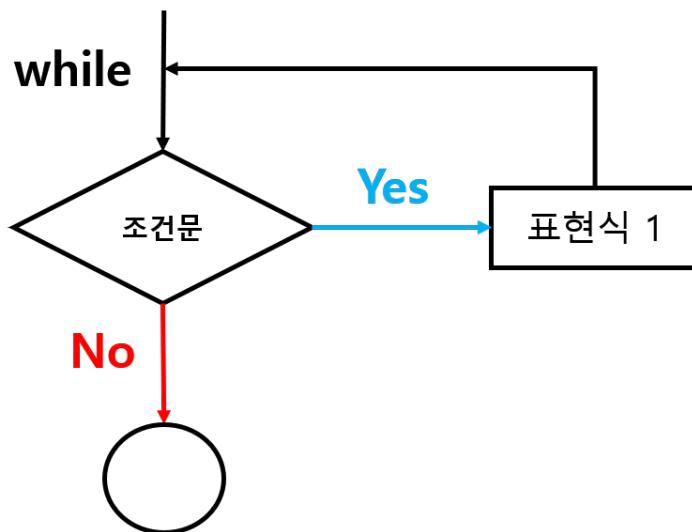


FIGURE 7.7: WHILE 구문 flow-chart

```
# WHILE 구문 예시 1
# 1:100 까지 합 계산 함수
tot <- 0; i <- 1 # 사용 변수 초기화 (update 변수)
while (i <= 100) {
```

```
tot <- tot + i  
i <- i + 1  
}  
tot
```

```
[1] 5050
```

1. 초기값 i가 조건 $i \leq 100$ 인지 확인
2. 참인 경우 $tot + i$ 를 통해 tot을 업데이트 한 다음 i를 1만큼 증가
3. 만약 i에 대한 조건 평가 결과가 거짓이면 while 구문을 빠져나감

```
# while 문 조건이 TRUE 인 경우  
tot <- 0; i <- 1 # 사용 변수 초기화 (update 변수)  
while (TRUE) {  
  tot <- tot + i  
  if (i >= 100) break  
  i <- i + 1  
}  
tot
```

```
[1] 5050
```

1. while 의 조건이 참이기 때문에 무한 반복
2. 단 i가 100과 같거나 클 경우 구문 탈출
3. 그 전 까지는 tot와 i를 갱신

```
# WHILE 구문 예시 2
# 문자열 벡터에서 특정 문자열의 인덱스를 반환

txtvec <- c("R", "package", "flow-control", "while", "if", "for", "repeat")
found <- FALSE
i <- 1

word <- readline("검색할 텍스트: ")
while (!found & i <= length(txtvec)) {
  if (txtvec[i] == word) {
    found <- TRUE
    break
  }
  cat(i, " 번째 위치에 해당 단어가 존재하지 않습니다.\n", sep="")
  i <- i + 1
}

if (found) {
  cat(i, " 번째 위치에 ", word, "를 찾았습니다.", sep = "")
} else {
  cat(word, " 단어는 해당 문자열 벡터에 존재하지 않습니다.\n", sep = "")
}
```

1. found = FALSE, i = 1을 초기값으로 입력
2. readline()으로 입력한 텍스트를 word에 저장
3. found 가 참이고 i가 텍스트 벡터의 길이 값과 같을 때 까지 다음 구문
반복
4. txtvec 각 원소와 word 값이 같은지 확인

`while` 입력 결과

- 1 번째 위치에 해당 단어가 존재하지 않습니다.
- 2 번째 위치에 해당 단어가 존재하지 않습니다.
- 3 번째 위치에 해당 단어가 존재하지 않습니다.
- 4 번째 위치에 while 를 찾았습니다.

temp 입력 결과

- 1 번째 위치에 해당 단어가 존재하지 않습니다.
- 2 번째 위치에 해당 단어가 존재하지 않습니다.
- 3 번째 위치에 해당 단어가 존재하지 않습니다.
- 4 번째 위치에 해당 단어가 존재하지 않습니다.
- 5 번째 위치에 해당 단어가 존재하지 않습니다.
- 6 번째 위치에 해당 단어가 존재하지 않습니다.
- 7 번째 위치에 해당 단어가 존재하지 않습니다.

temp 단어는 해당 문자열 벡터에 존재하지 않습니다.



- repeat, while과 같이 반복의 횟수가 지정되지 않는 반복구문을 불확정 반복문(indefinite loop)이라고 함.
- 다음에 배울 for 구문은 위 두 반복문과는 다르게 반복의 범위를 명확히 지정하기 때문에 확정 반복문(definite loop)라고 함.

7.2.3 for 구문

- 가장 많이 사용되는 반복구문으로 일반적인 형태는 아래와 같음

```
for (반복변수 in sequence) {  
    표현식 1
```

```
...  
}
```

- R에서 sequence는 특정 유형의 벡터이며, 반복변수에 sequence의 원소를 순차적으로 할당함
- 반복변수는 for 반복문 안의 표현식 1에서 사용됨

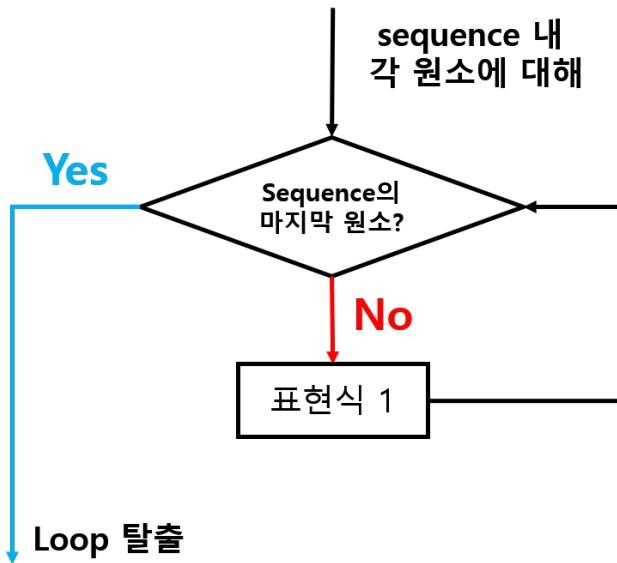


FIGURE 7.8: FOR 구문 flow-chart

```
#for 문 예시 1
student <- read.table("dataset/students.txt", sep = "\t", header = TRUE)
student_name <- student$name
for (s in student_name) {
  cat(s, "학생!! R을 배우면 통계가 쉬워져요!!^^\n")
}
```

송은철 학생!! R을 배우면 통계가 쉬워져요!!^^

윤지호 학생!! R을 배우면 통계가 쉬워져요!!^^
노자홍 학생!! R을 배우면 통계가 쉬워져요!!^^
박경민 학생!! R을 배우면 통계가 쉬워져요!!^^
윤지우 학생!! R을 배우면 통계가 쉬워져요!!^^
장민영 학생!! R을 배우면 통계가 쉬워져요!!^^
권혁제 학생!! R을 배우면 통계가 쉬워져요!!^^
김요한 학생!! R을 배우면 통계가 쉬워져요!!^^
김진현 학생!! R을 배우면 통계가 쉬워져요!!^^
박종현 학생!! R을 배우면 통계가 쉬워져요!!^^
신지성 학생!! R을 배우면 통계가 쉬워져요!!^^
오정우 학생!! R을 배우면 통계가 쉬워져요!!^^
이명현 학생!! R을 배우면 통계가 쉬워져요!!^^
전지원 학생!! R을 배우면 통계가 쉬워져요!!^^
조현모 학생!! R을 배우면 통계가 쉬워져요!!^^
최소미 학생!! R을 배우면 통계가 쉬워져요!!^^
김선재 학생!! R을 배우면 통계가 쉬워져요!!^^
김지윤 학생!! R을 배우면 통계가 쉬워져요!!^^
장유진 학생!! R을 배우면 통계가 쉬워져요!!^^
김하진 학생!! R을 배우면 통계가 쉬워져요!!^^
김민서 학생!! R을 배우면 통계가 쉬워져요!!^^
김준섭 학생!! R을 배우면 통계가 쉬워져요!!^^
남현준 학생!! R을 배우면 통계가 쉬워져요!!^^
채승훈 학생!! R을 배우면 통계가 쉬워져요!!^^
강현지 학생!! R을 배우면 통계가 쉬워져요!!^^
권사랑 학생!! R을 배우면 통계가 쉬워져요!!^^
김민선 학생!! R을 배우면 통계가 쉬워져요!!^^
김민영 학생!! R을 배우면 통계가 쉬워져요!!^^
박승원 학생!! R을 배우면 통계가 쉬워져요!!^^
박우담 학생!! R을 배우면 통계가 쉬워져요!!^^
소아영 학생!! R을 배우면 통계가 쉬워져요!!^^
안성재 학생!! R을 배우면 통계가 쉬워져요!!^^
이다빈 학생!! R을 배우면 통계가 쉬워져요!!^^
이연하 학생!! R을 배우면 통계가 쉬워져요!!^^

정진경 학생!! R을 배우면 통계가 쉬워져요!!^^
 조은아 학생!! R을 배우면 통계가 쉬워져요!!^^
 최보경 학생!! R을 배우면 통계가 쉬워져요!!^^
 한민형 학생!! R을 배우면 통계가 쉬워져요!!^^
 황연지 학생!! R을 배우면 통계가 쉬워져요!!^^

1. student_name의 첫 번째 원소를 s에 할당
2. for 구문 안에 표현 실행
3. student_name의 마지막 원소까지 반복

```
# 위 예시와 동일한 표현
## 인덱싱을 사용
for (i in 1:length(student_name)) {
  cat(student_name[i], "학생!! R을 배우면 통계가 쉬워져요!!^^\n")
}
```

```
## sequence를 만드는 함수 seq_along() 사용

for (i in seq_along(student_name)) {
  cat(student_name[i], "학생!! R을 배우면 통계가 쉬워져요!!^^\n")
}
```

- for 구문 안에 for 문을 1개 이상 중첩 가능

```
## 2중 for 문 예시
set.seed(12345)
id <- sample(1:length(student_name), 5)
```

```

sel_student <- student_name[id]

for (i in seq_along(student_name)) {
  for (j in seq_along(sel_student)) {
    if (student_name[i] == sel_student[j]) {
      cat(sel_student[j], "님!! 당첨 축하 드립니다!!\n")
    }
  }
}

```

전지원 님!! 당첨 축하 드립니다!!
 최소미 님!! 당첨 축하 드립니다!!
 채승훈 님!! 당첨 축하 드립니다!!
 권사랑 님!! 당첨 축하 드립니다!!
 김민영 님!! 당첨 축하 드립니다!!



- 불확정 반복문 학습 시 무한루프로부터 `break`를 통해 루프에서 탈출
- 루프를 완전히 탈출하지 않고 현재 반복을 중지하고 그 다음 반복을 진행하고 싶을 경우 `next` 예약어를 사용

```

# 알파벳 e와 일치하는 경우에만 텍스트 메세지 출력
vec <- c("a", "e", "e", "i", "o", "u", "e", "z")
word <- "e"

for (i in 1:length(vec)) {
  if (vec[i] != word) next
  cat(word, "가", i, "번 째 인덱스에 있네요!!\n")
}

```

e 가 2 번 째 인덱스에 있네요!!
 e 가 3 번 째 인덱스에 있네요!!
 e 가 7 번 째 인덱스에 있네요!!

7.3 함수 (function)

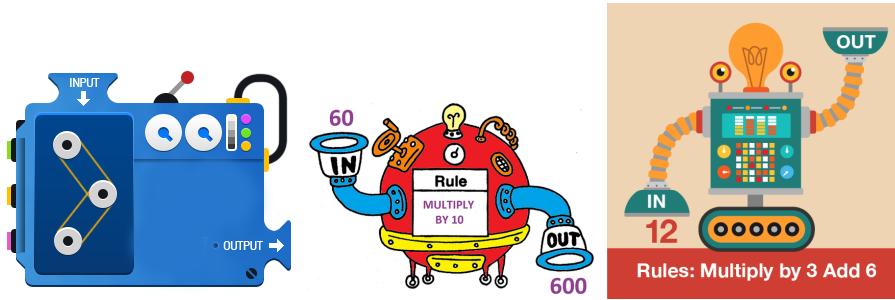


FIGURE 7.9: 함수

- **함수:** 특정한 목적을 위한 연산을 수행하기 위해 명명된 일련의 문장 (추상화)
 - 예: `sum(x)` → 벡터 x 의 값을 모두 합산하는 함수로 “`sum`”이라고 명명된 내장 함수
 - R 콘솔에서 함수 명칭(예: `sum`)을 입력 후 실행하면 함수 내부 확인 가능

```
sum
```

```
function (... , na.rm = FALSE) .Primitive("sum")
```

- 함수의 명칭(위의 예에서 `sum`)으로 특정 함수를 호출(call)



R의 스크립트는 내장된 혹은 사용자가 정의한 함수들을 호출함으로써 작성됨

함수를 사용해야만 하는 이유

- 매우 큰 프로그램 작업을 해야할 경우 함수를 통해 작업 단위 별로 분할 가능
- 한 번 작성한 함수는 재사용 가능
- 프로그램의 체계적 관리가 가능하기 때문에 유지 및 보수가 용이
- 프로그램 코드의 간결화

7.3.1 함수의 정의

- `function`이라는 R의 예약어를 통해 사용자 함수 정의
- 함수 정의 시 함수의 명칭을 반드시 부여해야 함

함수 이름 <- `function()`

- 함수는 일반적으로 인수(argument)로 입력값을 전달 받으면 그 결과값을 반환(return)
- 함수의 인수와 반환에 따라 다음과 같이 4 가지 유형의 함수 정의 가능
 - 인수를 갖는 함수
 - 인수를 갖지 않는 함수
 - 값을 반환하는 함수
 - 값을 반환하지 않는 함수

```
# (1) 인수를 갖는 함수
## (모)분산을 계산하는 함수

var_pop <- function(x) {
  n <- length(x)
  if (n < 2) {
    stop("적어도 두 개 이상의 관찰값이 존재해야 합니다")
  }
  mx <- mean(x)
  v <- sum((x - mx)^2)/n
  return(v) # 결과를 반환하는 함수: v를 함수의 출력값으로 설정
}

## test
set.seed(1) # 동일한 난수 생성을 위해 seed 번호 부여
x <- rnorm(1)
var_pop(x)
```

Error in var_pop(x): 적어도 두 개 이상의 관찰값이 존재해야 합니다

```
set.seed(1000)
x <- rnorm(1000, 2, 4) # 평균이 20이고 표준편차가 4인 정규분포로부터 1000개 난수 추출
var_pop(x)
```

[1] 15.40581

```
# (2) 인수를 갖지 않는 함수
print_lyrics_let_it_be <- function() {
  print("When I find myself in times of trouble, ")
  print("Mother Mary comes to me.")
```

```
print("Speaking words of wisdom 'let it be'.")  
}  
  
print_lyrics_let_it_be()
```

```
[1] "When I find myself in times of trouble, "  
[1] "Mother Mary comes to me."  
[1] "Speaking words of wisdom 'let it be'."
```

```
print_lyrics_let_it_be(beatles)
```

Error in print_lyrics_let_it_be(beatles): 사용되지 않은 인자 (beatles)

```
## 주사위를 둘리는 함수  
rolling_dice <- function() {  
  sample(1:6, 1, replace = TRUE)  
}  
  
rolling_dice(); rolling_dice(); rolling_dice();
```

```
[1] 4
```

```
[1] 4
```

```
[1] 4
```

```
# (3) 값을 반환하는 함수
manual_mean <- function(x) {
  n <- length(x)
  sumi <- 0
  for (i in 1:n) {
    sumi <- sumi + x[i]
  }
  return(sumi/n)
}

set.seed(20)
x <- sample(1:200, 20, replace = FALSE) # 1 ~ 200 중 랜덤하게 20개 추출(비복원)
manual_mean(x)
```

[1] 107

```
# 미리 정의하지 않은 인수를 입력한 경우
set.seed(4)
na_idx <- sample(1:length(x), 4)
x[na_idx] <- NA
manual_mean(x, na.rm = TRUE)
```

Error in manual_mean(x, na.rm = TRUE): 사용되지 않은 인자 (na.rm = TRUE)

```
# (4) 값을 반환하지 않는 함수(void function)
summary_mean <- function(x, ...) {
  n <- sum(!is.na(x))
  mx <- sum(x, ...)/n
  cat("Data: ", sprintf("%.2f", x), "\n") # 소수점 2째 자리 까지 출력
```

```
cat("전체 관찰값 개수(결측 제외) = ", n, "\n")
cat("산술평균 = ", mx, "\n")
}

set.seed(20)
x <- rnorm(20)
summary_mean(x)
```

Data: 1.16 -0.59 1.79 -1.33 -0.45 0.57 -2.89 -0.87 -0.46 -0.56 -0.02 -0.15 -0.63 1.32 -1.52 -0.44 0
전체 관찰값 개수(결측 제외) = 20
산술평균 = -0.1877639

```
result <- summary_mean(x)
```

Data: 1.16 -0.59 1.79 -1.33 -0.45 0.57 -2.89 -0.87 -0.46 -0.56 -0.02 -0.15 -0.63 1.32 -1.52 -0.44 0
전체 관찰값 개수(결측 제외) = 20
산술평균 = -0.1877639

```
result
```

NULL

```
x[na_idx] <- NA
# ...를 통해 미리 정하지 않은 인수를
# 함수 내부에서 호출한 다른 함수로 전달 가능
summary_mean(x, na.rm = TRUE)
```

```
Data:  1.16 -0.59 NA -1.33 -0.45 0.57 NA -0.87 -0.46 -0.56 NA -0.15 -0.63 1.32 -1.52 -0.44 0.97 0.03
전체 관찰값 개수(결측 제외) =  16
산술평균 = -0.1590692
```

```
x <- summary_mean(x, na.rm = TRUE)
```

```
Data:  1.16 -0.59 NA -1.33 -0.45 0.57 NA -0.87 -0.46 -0.56 NA -0.15 -0.63 1.32 -1.52 -0.44 0.97 0.03
전체 관찰값 개수(결측 제외) =  16
산술평균 = -0.1590692
```

7.3.2 함수의 인수 전달 방법

- 함수는 입력값(input)을 가지며, 이러한 입력값은 함수의 인수(argument)에 해당 값을 할당함으로써 입력값이 함수로 전달됨
- 함수의 인수 정의는 내 마음대로 가능(개수 무관)
- R에서 함수 호출 시 인수 전달은 “값”을 호출 하는 방식(call by value)



call by value와 상반되는 개념으로 참조에 의한 호출(**call by reference**)로 값이 아니라 값이 저장되어 있는 메모리의 주소 값을 전달하는 방식(대표적인 예: C 언어의 포인터)임. 계산 효율은 참조에 의한 호출이 월등히 뛰어나지만, 프로그램의 구조가 복잡하다는 단점을 가짐. R은 데이터 분석에 특화된 프로그램이기 때문에 직관적인 **call by value** 방식을 택함.

- 예시

```
# 두 변수의 값을 바꾸는 함수: swap
swap <- function(x, y) {
  temp <- x
  x <- y
  y <- temp
  cat("두 값이 바뀌었습니다.", sprintf("x = %d, y = %d", x, y), "\n")
}

x <- 3; y <- 10
swap(x, y)
```

두 값이 바뀌었습니다. x = 10, y = 3

x; y # x, y 두 값이 바뀌지 않음

[1] 3

[1] 10

- 인수를 전달하는 방법은 다음 두 가지임
 - 인수의 위치 순서에 의한 전달: 정의한 인수의 순서대로 각 인수에 대응하는 값을 전달
 - 인수의 이름에 의한 전달: 위치와 관계 없이 정의한 인수의 이름을 지정하여 값을 전달

```
# 표준편차 계산 함수: stdev
stdev <- function(x, na.rm = TRUE) {
  if (is.matrix(x)) apply(x, 2, sd, na.rm = na.rm)
  else if (is.vector(x)) sqrt(var(x, na.rm = na.rm))
  else if (is.data.frame(x)) sapply(x, sd, na.rm = na.rm)
  else sqrt(var(as.vector(x), na.rm = na.rm))
}

set.seed(1000)
X <- matrix(rnorm(1000), 100, 10)
x <- rpois(50, lambda = 10) # 포아송 분포(lambda = 10)에서 50개 난수 추출
dat <- mtcars # R 내장 데이터를 dat에 저장

# (1) 순서에 의한 전달
stdev(X, T); stdev(X) # 동일한 결과
```

```
[1] 1.0065940 0.9033927 0.9727257 0.9905631 0.8202803 1.0114516 0.9855547
[8] 1.0211373 1.0716219 1.0426811
```

```
[1] 1.0065940 0.9033927 0.9727257 0.9905631 0.8202803 1.0114516 0.9855547
[8] 1.0211373 1.0716219 1.0426811
```

```
stdev(x)
```

```
[1] 3.41569
```

```
stdev(dat)
```

| mpg | cyl | disp | hp | drat | wt |
|-----|-----|------|----|------|----|
|-----|-----|------|----|------|----|

```
6.0269481  1.7859216 123.9386938  68.5628685  0.5346787  0.9784574  
qsec          vs          am          gear          carb  
1.7869432  0.5040161  0.4989909  0.7378041  1.6152000
```

```
stdev(TRUE, dat) # 오류 why??
```

Warning in if (na.rm) "na.or.complete" else "everything": length > 1 0|
라는 조건
이 있고, 첫번째 요소만이 사용될 것입니다

```
Error in if (na.rm) "na.or.complete" else "everything": argument is not interpretable as logical
```

```
# (2) 이름에 의한 전달  
set.seed(5)  
na_idx <- sample(1:50, 5)  
x[na_idx] <- NA  
  
stdev(na.rm = T, x = x)
```

```
[1] 3.411211
```

```
stdev(dat = dat, na.rm = TRUE) # 오류 why???
```

```
Error in stdev(dat = dat, na.rm = TRUE): 사용되지 않은 인자 (dat = dat)
```

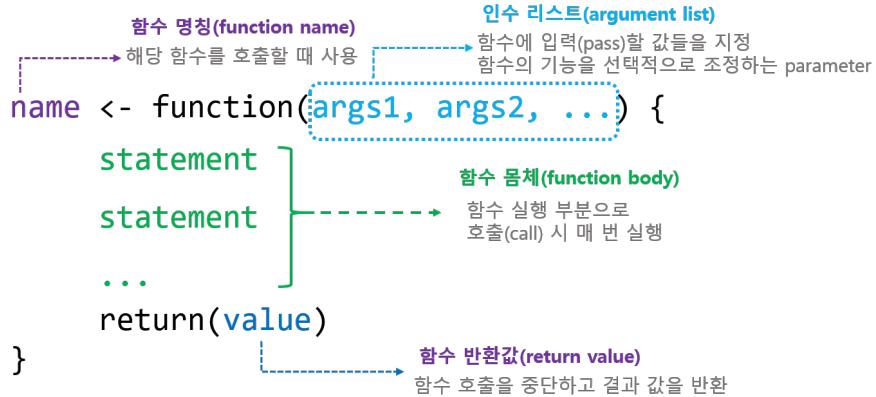


FIGURE 7.10: 함수의 기본 구조

7.3.3 함수의 기본 구성 요소

- `function()`에서 () 안의 부분(일반적으로 첫 번째 줄)을 머리(header) 부분
 - 함수의 초기 형태(매개변수 또는 인수의 형태)를 지정
- 연산 또는 명령이 수행되는 부분은 함수의 몸통(body) 부분({}로 표시)
 - 함수 내부에서 실행되는 연산 명령어들의 집합으로 구성
- 인수(argument): 함수의 기능을 선택적으로 조정하는 parameter로 함수 안에서 작동하는 매개변수들을 통칭
 - 인수는 argument 또는 argument = default value로 설정
 - 복수의 인수는 콤마(,)로 구분 → `fun_name <- function(arg1, arg2, arg3)`
 - 특수 인수: 어떠한 개수의 인수를 함수로 전달할 수 있음
 - * 일반적으로 인수의 개수가 불특정하거나 함수 안에서 다른

함수를 호출할 때 특정 인수를 다른 함수로 전달시킬 때 유용
(위 예시 참고)

```
# (1) 인수에 default 값을 주지 않은 함수
fun_without_arg_default <- function(x, y) {
  x*y
}

set.seed(10)
a <- sample(1:20, 10, replace = TRUE) # 복원 추출
a[7] <- NA
b <- 5

fun_without_arg_default(a, b)
```

```
[1] 55 45 50 80 60 40 NA 95 75 75
```

```
# (2) 인수에 default 값을 부여한 함수
fun_with_arg_default <- function(x = 5, y = 8) {
  x*y
}

fun_with_arg_default()
```

```
[1] 40
```

```
trim_mean <- function(x, trim = 0, na.rm = F) {
  mean(x, trim = trim, na.rm = na.rm)
}
trim_mean(a)
```

```
[1] NA
```

```
trim_mean(x = a, trim = 0.2, na.rm = TRUE) # 인수 이름으로 값 전달
```

```
[1] 12.57143
```

```
trim_mean(a, 0.3, TRUE) # 인수 순서대로 값 전달
```

```
[1] 12.6
```

```
# (3) ... 인수 사용 예제
# list() 함수를 이용해 `...`에 해당하는 인수들을 리스트 객체로 만든 후
# 이를 함수에서 사용

dot_example <- function(x, ...) {
  # browser()
  trim <- 0
  na.rm <- FALSE

  dots <- list(...) # ...에 해당하는 인수 추출
  for (arg in names(dots)) {
    if (arg == "trim") trim <- as.numeric(dots[arg])
    if (arg == "na.rm") na.rm <- as.logical(dots[arg])
  }

  mean(x, trim = trim, na.rm = na.rm)
}

dot_example(a)
```

7.3 힘수 (function)

37

```
[1] NA
```

```
set.seed(30)
a <- sample(1:30, 15, replace = TRUE) # 농원 추출
dot_example(a)
```

```
[1] 17.06667
```

```
a[9] <- NA
dot_example(a)
```

```
[1] NA
```

```
dot_example(a, trim = 0.1, na.rm = TRUE)
```

```
[1] 17.75
```

```
# (4) `...` 인수가 함수 내 사용(호출)된
# 다른 함수의 인수로 전달하는 경우
# summary_mean() 함수 예제와 유사
mean_manual <- function(x, ...) {
  mean(x, ...)
}

set.seed(30)
x <- rnorm(30, mean = 10, sd = 5)
```

```
na_idx <- sample(1:30, 3, replace = TRUE)
xna <- x; xna[na_idx] <- NA

mean_manual(x)
```

```
[1] 8.347683
```

```
mean_manual(xna)
```

```
[1] NA
```

```
mean_manual(xna, na.rm = TRUE)
```

```
[1] 8.127862
```

```
mean_manual(x = xna, trim = 0.2, na.rm = TRUE)
```

```
[1] 7.534424
```



함수 몸체 안에 `browser()`을 입력하면, `browser()` 전 까지 함수 몸체 안 명령들이 실행되고, 이후 명령들이 어떻게 실행되는지 확인할 수 있음. 함수 작성 시 함수로직을 세우고 디버깅 할 때 매우 유용하게 사용

인수 관련 몇 가지 유용한 함수들

- `args()`: 특정 함수에서 사용되는 인수 확인

```
args(fun_without_arg_default)
```

```
function (x, y)  
NULL
```

```
args(rnorm)
```

```
function (n, mean = 0, sd = 1)  
NULL
```

- `body()`: 험수의 몸체 조회

```
body(var_pop)
```

```
{  
  n <- length(x)  
  if (n < 2) {  
    stop("적어도 두 개 이상의 관찰값이 존재해야 합니다")  
  }  
  mx <- mean(x)  
  v <- sum((x - mx)^2)/n  
  return(v)  
}
```

```
body(dot_example)
```

```
{
  trim <- 0
  na.rm <- FALSE
  dots <- list(...)
  for (arg in names(dots)) {
    if (arg == "trim")
      trim <- as.numeric(dots[arg])
    if (arg == "na.rm")
      na.rm <- as.logical(dots[arg])
  }
  mean(x, trim = trim, na.rm = na.rm)
}
```

- `match.arg()`: 인수를 매치하는 함수로 매치할 대상의 인수를 지정
 - arg: 매치할 대상 인수 지정
 - choice: 매치될 인수값 목록
 - several.ok: 복수 선택 여부(TRUE/FALSE)

```
# 인수의 매치
match.arg(arg = c("med", "max"),
          choices = c("mean", "median", "iqr", "minimum", "maximum", "range"),
          several.ok = TRUE)
```

```
[1] "median"  "maximum"
```

```
match.arg(arg = c("median", "maximum"),
          choices = c("mean", "med", "iqr", "minimum", "max", "range"),
          several.ok = TRUE) # 오류 why??
```

Error in match.arg(arg = c("median", "maximuum"), choices = c("mean", : 'arg' 은 반드시 "mean", "med", "iqr", "minimum", "max", "range" 중 하나이어야 합니다

```
match.arg(arg = c("med", "max"),
          choices = c("mean", "median", "iqr", "minimum", "maximum", "range"),
          several.ok = FALSE)
```

Error in match.arg(arg = c("med", "max"), choices = c("mean", "median", : 'arg' 는 반드시 길이가 10이어야 합니다

```
# match.arg() 함수 응용
# 중심값 관련 통계량 계산 함수
# 평균(mean), 절삭평균(trimmed mean), 중앙값(median), 최빈수(mode) 계산

# pkg_list <- rownames(installed.packages()) # 설치된 패키지 목록
# if (!("DescTools" %in% pkg_list))
#   install.packages("DescTools") # 최빈수를 구하기 위한 패키지 설치

center <- function(x,
                    type = c("mean", "trimmed", "median", "mode"),
                    ...
                    )
{
  # browser()
  trim = 0; na.rm = FALSE # dot 인수 초기값
  type <- match.arg(type)
  dots <- list(...)
  for (arg in names(dots)) {
    if (arg == "trim") trim <- as.numeric(dots[arg])
```

```

if (arg == "na.rm") na.rm <- as.logical(dots[arg])
}

switch(type,
       mean = mean(x, na.rm = na.rm),
       trimmed = mean(x, trim = trim, na.rm = na.rm),
       median = median(x, na.rm = na.rm),
       mode = DescTools::Mode(round(x, 1), na.rm = na.rm)
       # DescTools 패키지 내 Mode 함수를
       # workspace0:// 불러오지 않고 사용
)
}

set.seed(100)
x <- rchisq(100, df = 3) # 자유도가 3인 카이제곱분포에서 난수 추출
xna <- x; xna[na_idx] <- NA

center(x, "mean"); center(x, "me")

```

[1] 2.929673

Error in match.arg(type): 'arg'은 반드시 "mean", "trimmed", "median", "mode" 중 하나
나이어야 합니다

[1] 2.929673

```
center(x, "trimmed", trim = 0.1)
```

[1] 2.565866

7.3 합수 (function)

43

```
center(x, "median")
```

```
[1] 2.45614
```

```
center(x, "mode")
```

```
[1] 1.7  
attr("freq")  
[1] 6
```

```
center(xna, "median")
```

```
[1] NA
```

```
center(xna, "median", na.rm = TRUE)
```

```
[1] 2.423723
```



switch() 함수는 ifelse() 함수의 확장 버전으로 n 개의 조건에 대한 분기 가능

함수 제어 관련 주요 함수

- return(): 계산된 결과를 반환하는 함수로 함수의 흐름에서 return()이 나타나면 결과값을 반환하고 함수 종료
 - 강제 종료가 필요한 경우 응용 가능

```
# (1) 객체// 반환
set.seed(100)
x <- rnorm(100, mean = 24, sd = 2.2)
value_return1 <- function(x) {
  tot <- sum(x)
  n <- length(x)
  result <- list(size = n, total = tot, average = mean(x), stdev = sd(x))
  return(result)
}
value_return1(x)
```

```
$size
```

```
[1] 100
```

```
$total
```

```
[1] 2400.641
```

```
$average
```

```
[1] 24.00641
```

```
$stdev
```

```
[1] 2.245563
```

```
desc <- value_return1(x)
desc$stdev
```

```
[1] 2.245563
```

```
value_return2 <- function(x) {  
  return(sum(x)/length(x))  
}  
value_return2(x)
```

```
[1] 24.00641
```

```
# (2) 강제 종료 시 활용  
value_return3 <- function(x) {  
  if (anyNA(x)) return  
  return(sum(x)/length(x))  
}  
  
xna <- x; xna[na_idx] <- NA  
value_return3(xna)
```

```
[1] NA
```

```
value_return3(x)
```

```
[1] 24.00641
```

- `stop()`: 예외처리 함수의 일종으로 특정 조건일 경우 (오류) 메세지를 출력하고 함수 종료
 - 인수로 문자열을 가짐

```
# (1) stop() 함수 사용
# 복소수 값을 실수와 허수로 분할
split_complex <- function(z) {
  if(!is.complex(z))
    stop("입력값이 복소수가 아닙니다")
  re <- Re(z)
  im <- Im(z)
  return(list(real = re, imaginary = im))
}

split_complex(pi)
```

Error in split_complex(pi): 입력값이 복소수가 아닙니다

```
split_complex(23 + 7i)
```

```
$real
```

```
[1] 23
```

```
$imaginary
```

```
[1] 7
```

7.3.4 함수의 적용 범위(scoping rule)

Scoping rule: 변수 또는 객체가 어디에서 사용 가능한지를 결정하는 규칙

1. **매개변수(parameter):** 함수를 적용할 때 사용되는 변수로 인수로부터 발생함

- 함수의 인수 리스트에서 인수값이 매개변수로 할당
2. 지역변수(local variable): 함수의 몸체 부분에서 정의된 변수들을 지칭하며 함수의 종료와 동시에 재사용 불가
 3. 전역변수(global variable): 함수의 외부(workspace)에서 정의된 변수로 함수 내부에서 값을 할당하지 않더라도 사용 가능

```
# (1) 매개변수, 지역변수, 전역변수 구분
x <- 10 # 전역변수
y <- 5 # 전역변수

scope1 <- function(x) {
  y <- x^2
  print(x) # 매개변수
  print(y) # 지역변수
}

x; y # 전역변수가 출력
```

```
[1] 10
```

```
[1] 5
```

```
scope1(x = 10)
```

```
[1] 10
```

```
[1] 100
```

- 작업공간에서 x와 y는 각각 10, 5 값이 할당
- 작업공간 상에서 x y 값은 변하지 않음
- 지역변수 y의 사용 범위는 함수 몸체이기 때문에 함수 밖에 있는 y는 값이 변하지 않음

```
x <- 10 # 전역변수
y <- 5 # 전역변수
rm(z)

scope2 <- function(x) {
  y <- x^2
  print(x) # 매개변수
  print(y) # 지역변수
  print(z)
}
scope2(x = 5)
```

```
[1] 5
[1] 25
```

Error in print(z): 객체 'z'를 찾을 수 없습니다

```
z <- 13 # 전역변수로 z 할당
scope2(x = 5)
```

```
[1] 5
[1] 25
[1] 13
```

- 함수 외부와 내부 모두에서 z가 정의되지 않았기 때문에 에러 출력
- 작업공간 상에 z를 정의한 경우 함수 내부에서 workspace에서 정의한 z를 그대로 사용 → 함수 외부와 내부 자유로이 사용 가능한 변수를 자유변수(free variable)이라고 지칭함.

지역변수의 사용 범위는 함수 몸체 안이지만 그 범위를 밖으로 확장할 수 있음 → <<- 또는 ->> 사용

```
# 지경변수의 확장 예제
x <- 1; y <- 2; z <- 3; k <- 10
scope3 <- function(x) {
  y <<- x + 10
  y * 3 ->> z

  print(x) # 매개변수
  print(y) # 지역변수
  print(z) # 지역변수
  print(k) # 자유변수
}

x;y;z;k
```

[1] 1

[1] 2

[1] 3

[1] 10

```
scope3(x = 2)
```

```
[1] 2  
[1] 12  
[1] 36  
[1] 10
```

```
x;y;z;k
```

```
[1] 1  
  
[1] 12  
  
[1] 36  
  
[1] 10
```

하나의 함수 내부에 또 다른 함수 생성 가능

```
mean_manual2 <- function(x) {  
  tot <- sum(x)  
  size <- function(x) {  
    return(length(x))  
  }  
  return(tot/size(x))  
}  
  
mean_manual2(1:10)
```

```
[1] 5.5
```

Bibliography

(2018). R에서 원하는 키워드의 뉴스를 웹크롤링(스크래핑) 하는 방법. <https://dr-hkim.github.io/Naver-News-Web-Scraping-using-Keywords-in-R/>.

(2018 Accessed: 2020-04-16). POSIX. <https://ko.wikipedia.org/wiki/POSIX>.

Chen, D.-G. D. and Peace, K. E. (2010). *Clinical trial data analysis using R*. CRC Press.

Peng, R. D. (2016). *R programming for data science*. Learnpub.

Rizzo, M. L. (2019). *Statistical computing with R*. CRC Press.

Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L. D., Francois, R., Grolemund, G., Hayes, A., Henry, L., Hester, J., Kuhn, M., Pedersen, T. L., Miller, E., Bache, S. M., Muller, K., Ooms, J., Robinson, D., Seidel, D. P., Spinu, V., Takahashi, K., Vaughan, D., Wilke, C., Woo, K., and Yutani, H. (2019). Welcome to the tidyverse. *Journal of Open Source Software*, 4(43):1686.

Wickham, H. and Grolemund, G. (2016). *R for data science: import, tidy, transform, visualize, and model data*. "O'Reilly Media, Inc.".

Xie, Y. (2016). *bookdown: Authoring Books and Technical Documents with R Markdown*. Chapman and Hall/CRC, Boca Raton, Florida. ISBN 978-1138700109.

Xie, Y., Allaire, J., and Grolemund, G. (2018). *R Markdown: The Definitive Guide*. Chapman and Hall/CRC, Boca Raton, Florida. ISBN 9781138359338.

권재명 (2017). *실리콘밸리 데이터 과학자가 알려주는 따라하며 배우는 데이터 과학*. 제이펍, 1st edition. ISBN 979-1185890869.

매트로프, 2012). *빅데이터 분석 도구 R 프로그래밍*. 에이콘출판, 1st edition. ISBN 978-8960773332.

서민구 (2014). *R을 이용한 데이터 처리와 분석*. 길벗, 1st edition. ISBN 978-8966188260.