

한국한의학연구원, 구본초

통계 프로그래밍 언어

2020년도 1학기 충남대학교 정보통계학과 강의 노트



Contents

List of Tables	vii
List of Figures	ix
Course Overview	xi
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	18
1.4 RStudio	21
1.4.1 RStudio 설치하기	21
1.4.2 RStudio IDE 화면 구성	24
1.4.3 RStudio 환경 설정	30
1.4.4 RStudio 프로젝트	39
1.5 R 패키지	42
1.5.1 R 패키지 경로 확인 및 변경	43
1.5.2 R 패키지 설치하기	45
1.5.3 R 패키지 불러오기	46
1.6 R 기초 문법	47

1.7 R Markdown (맛보기)	51
2 R 객체(R object)	61
2.1 스칼라(scalar)	62
2.1.1 선언	63
2.1.2 숫자형	64
2.1.3 문자형	66
2.1.4 논리형 스칼라	67
2.1.5 결측값(missing value)	71
2.1.6 NULL 값	72
2.1.7 무한대/무한소/숫자아님	73
2.2 벡터(vector)	74
2.2.1 벡터의 특징	74
2.2.2 벡터의 연산	79
2.2.3 벡터의 색인(indexing)	87
2.2.4 벡터 관련 함수	90
2.3 리스트(list)	100
2.3.1 리스트 생성	101
2.3.2 리스트 색인	104
2.3.3 리스트에 원소 추가/제거	108
2.3.4 리스트의 결합	111
2.4 행렬(matrix)	113
2.4.1 행렬의 연산	117
2.4.2 행렬의 색인	129
2.4.3 행과 열 추가 및 제거	135
2.4.4 행렬 관련 함수	137
2.4.5 벡터와 행렬의 차이점	142

Contents v

2.4.6	의도치 않은 차원축소 피하기	142
2.5	배열(array)	144
2.5.1	배열의 생성 및 색인	144
2.5.2	배열의 확장 예제	147
2.6	요인(factor)과 테이블(table)	151
2.6.1	요인(factor)	152
2.6.2	테이블(table)	159
2.7	데이터 프레임(data frame)	171
2.7.1	데이터 프레임 생성	171
2.7.2	데이터 프레임 접근 및 필터링	179
2.7.3	데이터 프레임 관련 함수	186
2.7.4	*apply() 계열 함수	195
2.8	Homework #2-1	208



List of Tables

0.1	강의 계획표	xiv
1.1	R help 관련 명령어 리스트	17
2.1	R언어의 기본 수치 연산자	64
2.2	R언어의 논리형 연산자	68
2.3	R언어의 비교 연산자	68
2.4	리스트 데이터 접근 방법	105
2.5	스프레드시트 기본 형태 예시	171



List of Figures

1.1	Windows에서 R 실행화면(콘솔 창, SDI 모드)	13
1.2	정규분포 100개의 히스토그램	17
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.	20
1.4	RStudio 화면 구성 : 우하단 그림은 http://r-statistics.co/Top50-Ggplot2-Visualizations-MasterList-R-Code.html 에서 발췌	24
1.5	RStudio 콘솔창에서 명령어 실행 후 출력결과 화면	25
1.6	RStudio 스크립트 새로 열기	26
1.7	RStudio Environment 창 객체 상세 정보 및 스프레드 시트 출력 결과	27
1.8	R General option 팝업 창	31
1.9	R Markdown의 최종 결과물 산출과정 (http://r-project-reporting-template/)	52
1.10	test.html 문서 화면(저장 폴더 내 ‘test.html‘을 크롬 브라우저로 실행)	55

2.1 R 데이터 탑입 구조 다이어그램: [R, Python 분석과 프로그래밍 (by R Friend)](http://rfriend.tistory.com/)에서 발췌 후 수정	63
2.2 https://www.geeksforgeeks.org/matlab-rgb-image-representation/ 에서 발췌	147

Course Overview



본 문서는 2020년도 1학기 정보통계학과에서 개설한 “통계 프로그래밍 언어” 강의를 위해 개발한 강의 노트이고 주 단위로 업데이트될 예정임. 본 강의 노트는 <https://zorba78.github.io/cnu-r-programming-lecture-note/> 에서 확인할 수 있고, 해당 페이지에서 pdf 파일 다운로드가 가능함. 본 문서는 Yihui Xie가 개발한 **bookdown** 패키지 (Xie, 2016)를 활용하여 생성한 문서이고 Google Chrome 또는 Firefox 브라우저에 최적화 됨. 아울러 충남대학교 정보통계학과 이상인 교수님의 2019년도 2학기 “통계패키지활용” 강의 노트와 동국대학교 ICT빅데이터학부 김진석 교수님의 R 프로그래밍 및 실습¹ 강의 자료 내용과 구성을 참고하여 작성함. 재택 수업 시 학생들이 사용하고 있는 컴퓨터의 인터넷 접속이 원활하다는 가정 하에서 강의를 진행할 예정이기 때문에 수강 시 온라인 상태 유지가 필수임.

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 package를 활용한 데이터 핸들링 및 자료 시각화
- R 시뮬레이션을 통한 통계학 기초 이론 확인
- R을 이용한 데이터 분석 실습
- R markdown을 이용한 재현가능(reproducible)한 보고서 작성 방법
습득

선수과목

통계학 개론

수업 방법

- 강의: 50 %
- 실험/실습: 50%

평가방법

- 중간고사: 40 %
- 기말고사: 40 %
- 출석: 10 %
- 과제: 10 %

수업 규정

- 3번 지각은 1번 결석으로 처리
- 특별한 사유 없이 수업 중간에 이탈한 경우 결석으로 처리
- 특별한 사유로 인해 결석 또는 지각을 할 경우 사유를 증빙할 수 있는 서류 제출 시 출석으로 인정
- 출결 미달, 중간 또는 기말고사 미 응시인 경우 F 학점으로 처리
- 수업 중 휴대폰 및 각종 모바일 기기 사용 금지

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.

참고 자료

- 빅데이터 분석 도구 R 프로그래밍 ([매트로프](#), 2012)
- 실리콘밸리 데이터과학자가 알려주는 따라하며 배우는 데이터 과학 ([권재명](#), 2017)
- R을 이용한 데이터 처리&분석 ([서민구](#), 2014)
- R 그래픽스 ([유충현 et al.](#), 2005)
- ggplot2: elegant graphics for data analysis² ([Wickham](#), 2016)
- R for data science³ ([Wickham and Grolemund](#), 2016)
- Statistical Computing with R ([Rizzo](#), 2019)

²<https://ggplot2-book.org/>

³<https://r4ds.had.co.nz/>

강의 계획

TABLE 0.1: 강의 계획표

주차	강의 내용	과제
Week 1	R 소개, R/R Studio 설치, R 패키지 설치, 과제 1 R 맛보기 및 markdown 문서 만들기	
Week 2	R 자료형: 스칼라, 벡터, 리스트	
Week 3	R 자료형: 행렬 및 배열	과제 2
Week 4	R 자료형: 팩터, 테이블, 데이터 프레임	
Week 5	R 자료형: 문자열과 정규 표현식	과제 3
Week 6	데이터 프레임 가공 및 시각화 I	
Week 7	데이터 프레임 가공 및 시각화 II	과제 4
Week 8	중간고사	
Week 9	데이터 프레임 가공 및 시각화 III	
Week 10	R 프로그래밍: 조건문, 반복문, 함수	과제 5
Week 11	통계시뮬레이션 I: 표본분포 및 중심극한정리	
Week 12	통계시뮬레이션 2: 신뢰구간과 가설검정	과제 6
Week 13	R을 이용한 기초통계 분석	
Week 14	R markdown 활용	과제 7
Week 15	기말고사	

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<http://cran.r-project.org/web/view>

- 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

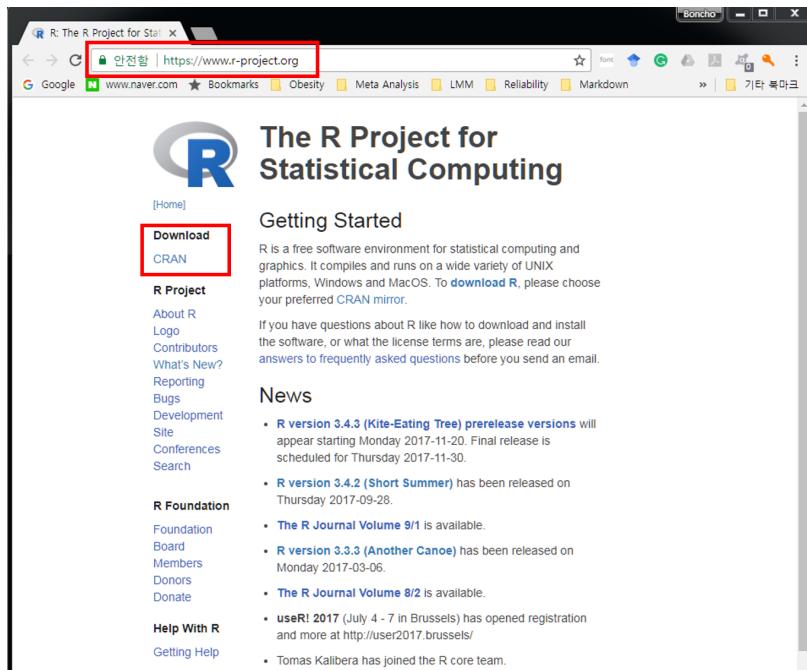
1.1 R 설치하기

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

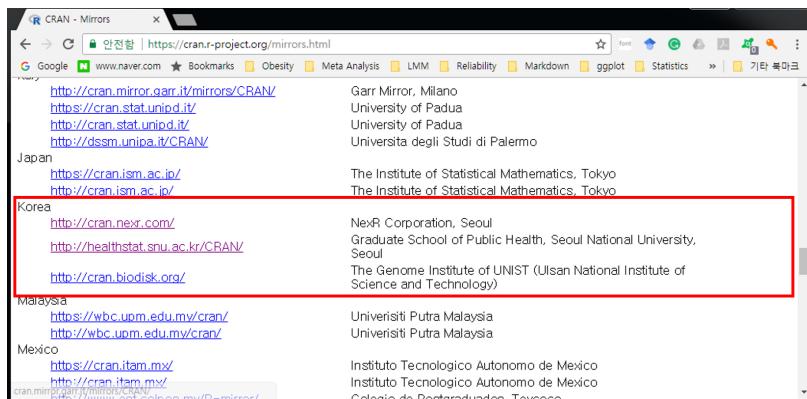
1. 웹 브라우저 (i.e. Explore, Chrome, Firefox 등)의 주소 입력창에 <https://www.r-project.org>
2. 좌측 R Logo 하단 Download 아래 CRAN 클릭

1.1 R 설치하기

5



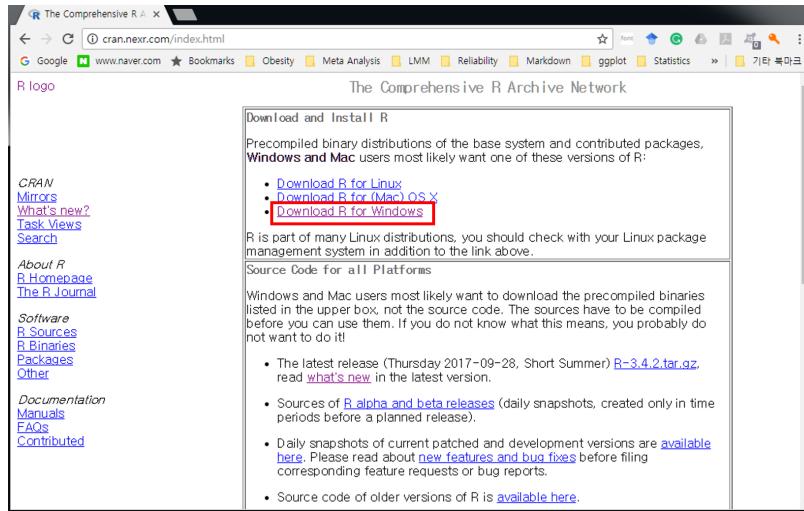
3. 클릭 후 연결한 페이지를 스크롤 후 Korea 아래 링크⁷ 클릭



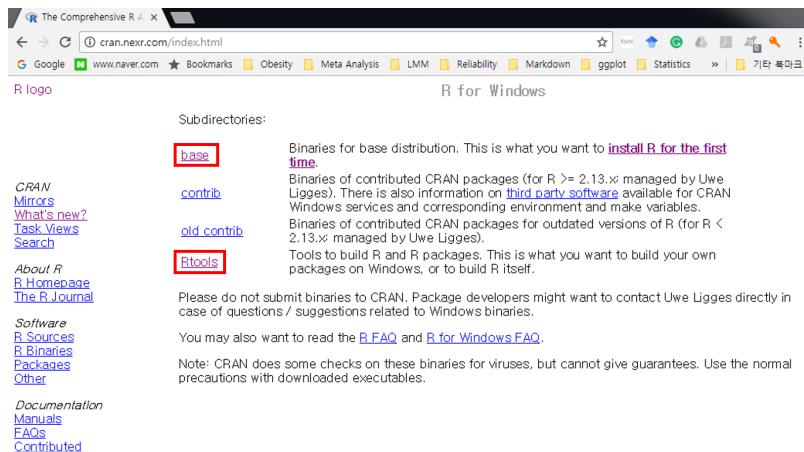
4. 클릭 후 세 가지 운영체제(Linux, Mac OS X, Windows)에 따른 R 버전 선택 가능⁸

⁷ 해당 링크들은 접속 시점에 따라 변경될 수 있음

⁸ 본 노트는 Windows 버전 설치만 다룸



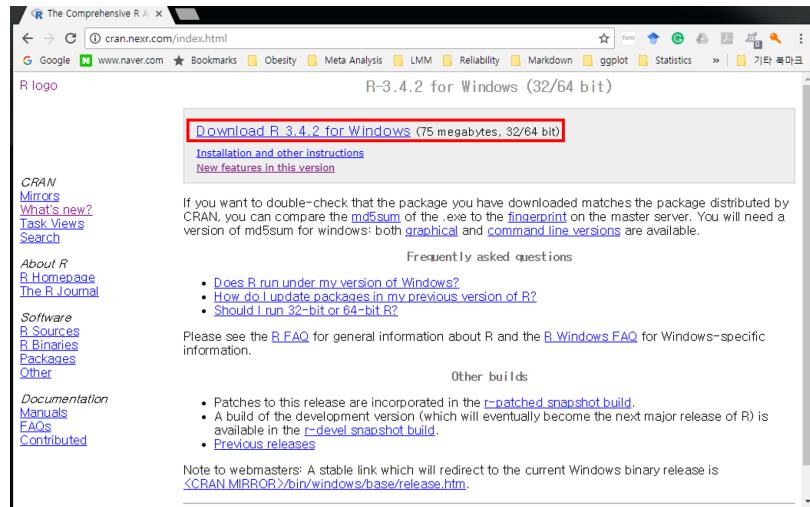
5. Downloads R for Windows 링크 클릭하면 다음과 같은 화면으로 이동



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

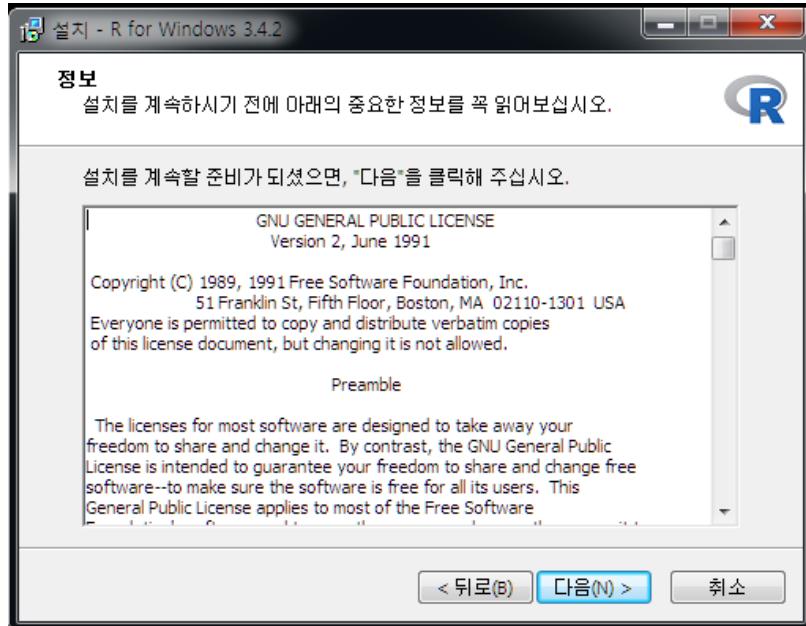


7. 다운로드한 파일을 실행하면 아래와 같은 대화창이 나타남

- 한국어 선택 → 환영 화면에서 [다음(N)>] 클릭

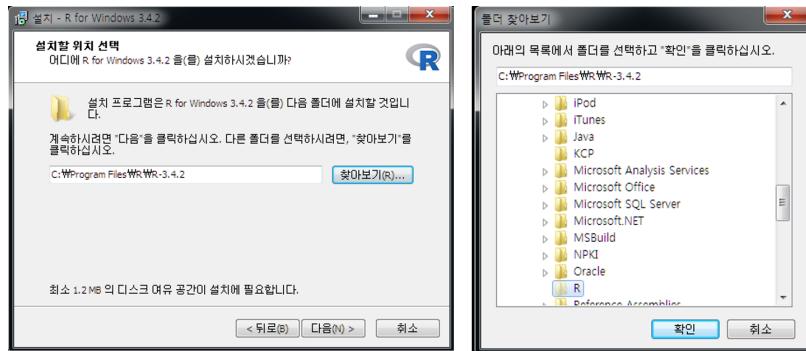


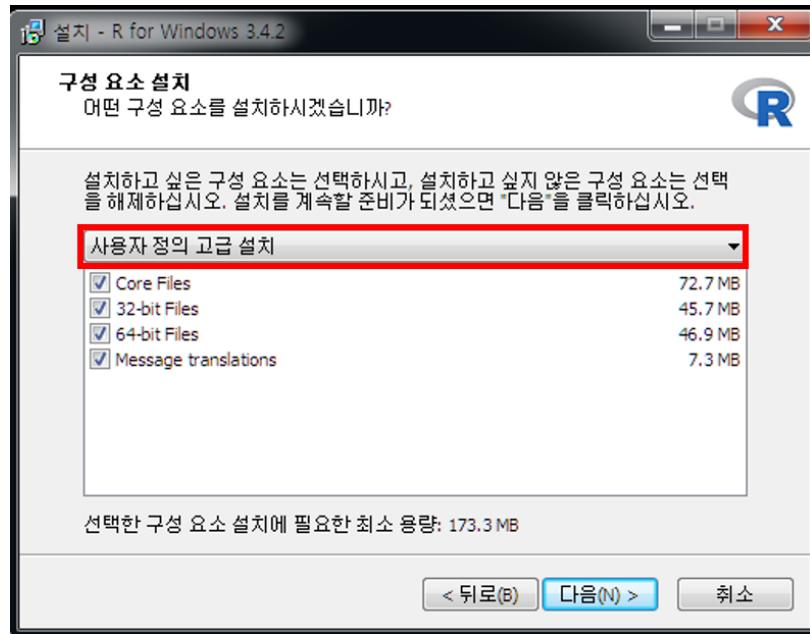
8. GNU 라이센스에 대한 설명 및 동의 여부([다음(N)>]) 클릭



9. 설치 디렉토리 설정 및 구성요소 설치 여부

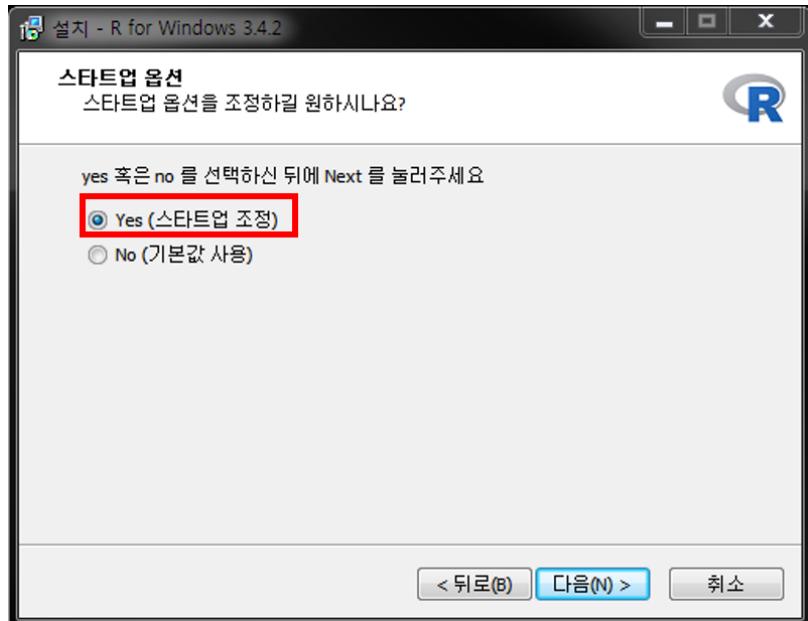
- 원하는 디렉토리 설정 (예: C:\R\R-3.x.x)
 - 기본 프로그램 (“Core Files”), 32 또는 64 bit 용 설치 파일, R console
- 한글 번역 모두 체크 뒤 [다음(N)>] 클릭





10. R 스타트업 옵션 지정

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

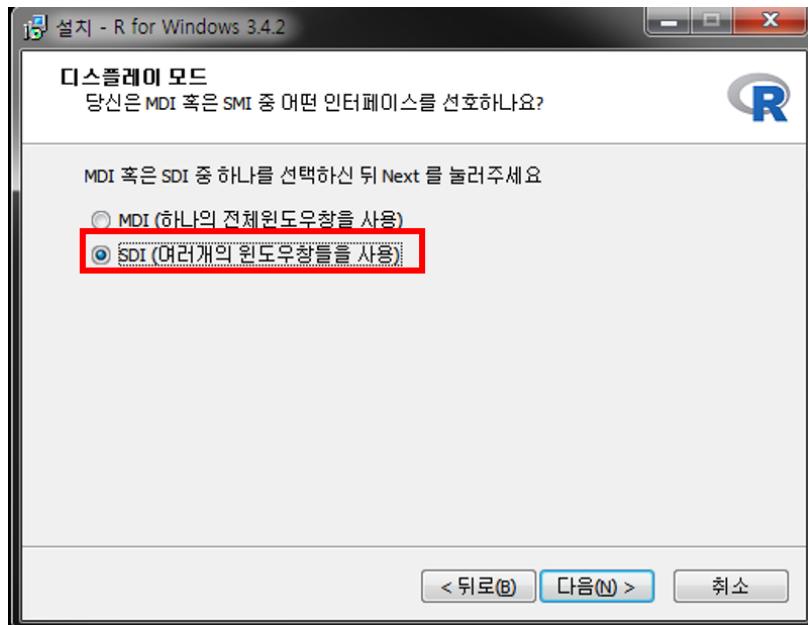


11. 화면표시방식(디스플레이) 모드 설정 변경

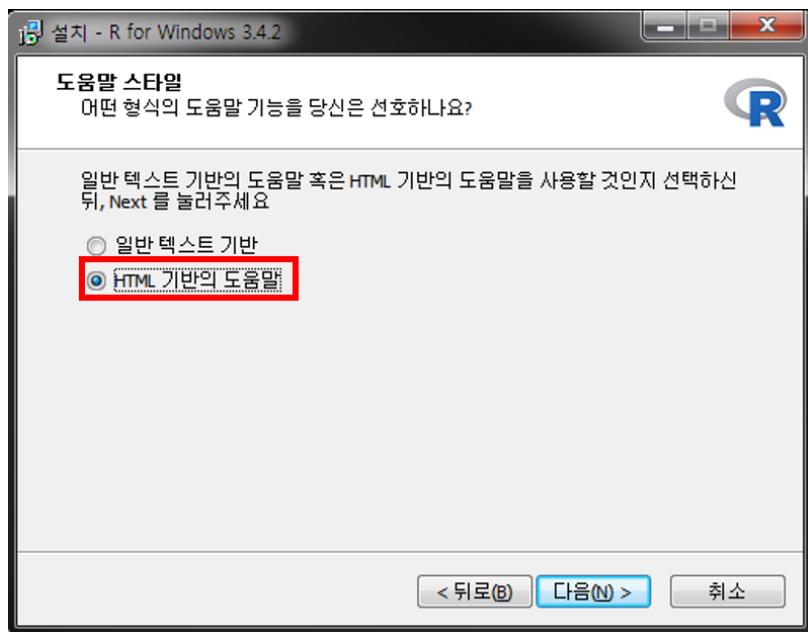
- MDI: 한 윈도우 내에서 script 편집창, 출력, 도움말 창 사용
- SDI: 다중 창에서 각각 script 편집창, 출력, 도움말 등을 독립적으로 열기

1.1 R 설치하기

11

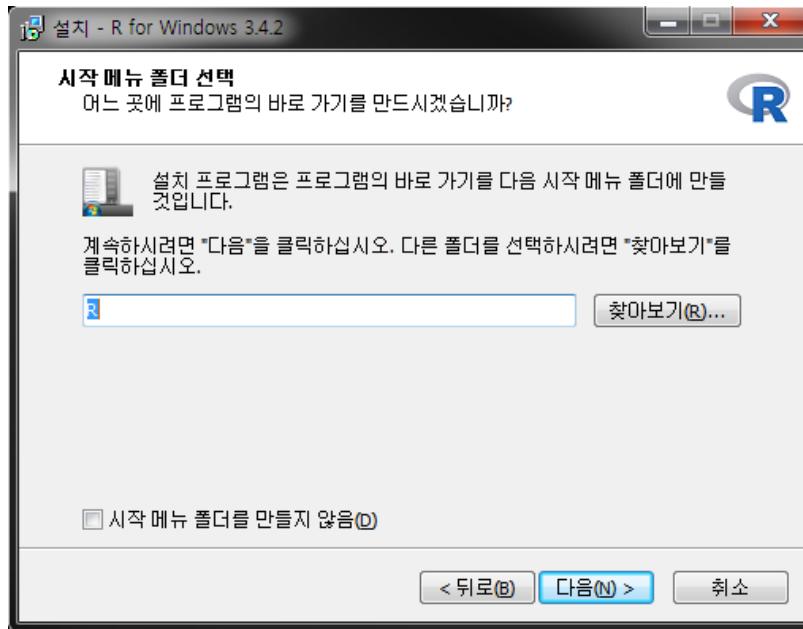


12. 도움말 형식에서 HTML 도움말 기반 선택



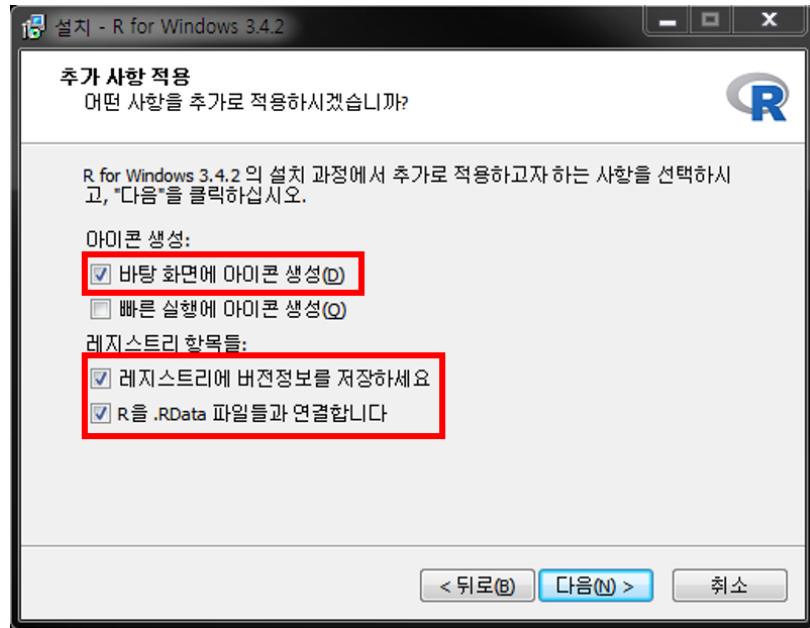
13. 시작메뉴 폴더 선택

- “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

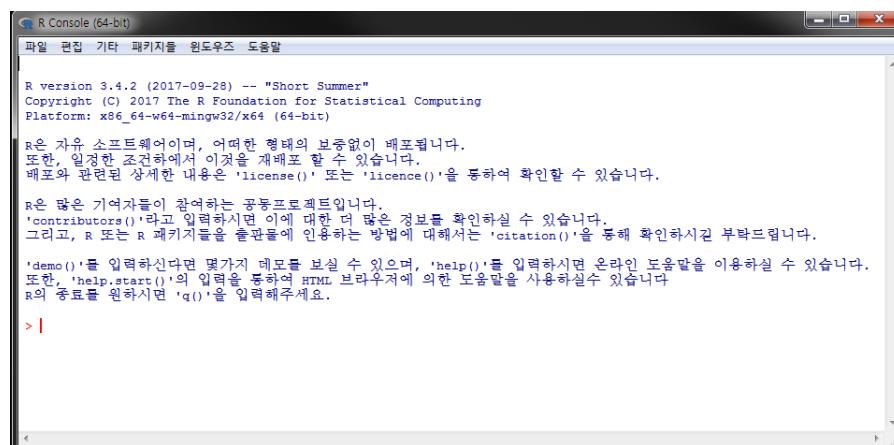


FIGURE 1.1: Windows에서 R 실행화면(콘솔 창, SDI 모드)

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 3.6.3 (2020-02-29)
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
[3] 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:
```

⁹현재 실행되고 있는 R의 작업공간

```
[1] knitr_1.28
```

```
loaded via a namespace (and not attached):  
[1] compiler_3.6.3  magrittr_1.5    bookdown_0.18.1 htmltools_0.4.0  
[5] tools_3.6.3     yaml_2.2.1      Rcpp_1.0.4       stringi_1.4.6  
[9] rmarkdown_2.1    stringr_1.4.0   digest_0.6.25   xfun_0.12  
[13] rlang_0.4.5     evaluate_0.14
```

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
```

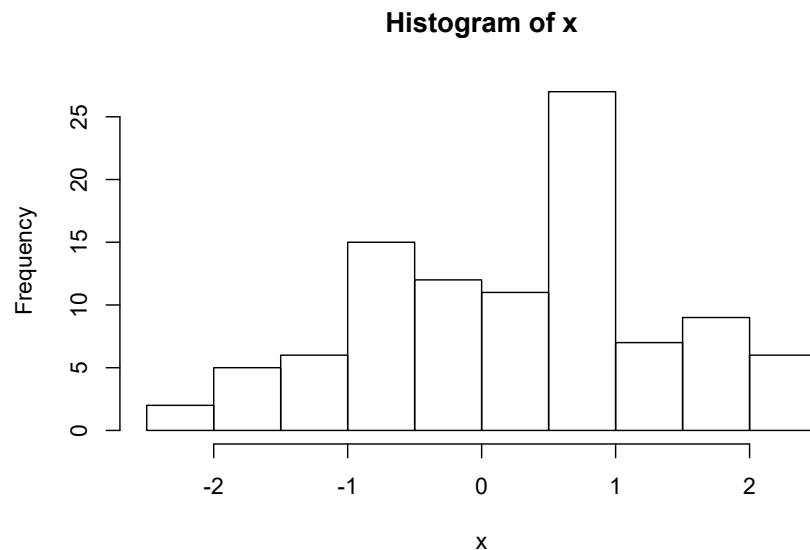
```
[1] 63
```

```
[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

– vignette()에서 제공하는 문서는 데이터를 기반으로 사용하고자 하는 패키지의 실

제 활용 예시를 작성한 문서이기 때문에 초보자들이 R 패키지 활용에 대한 접근성을 높혀줌.

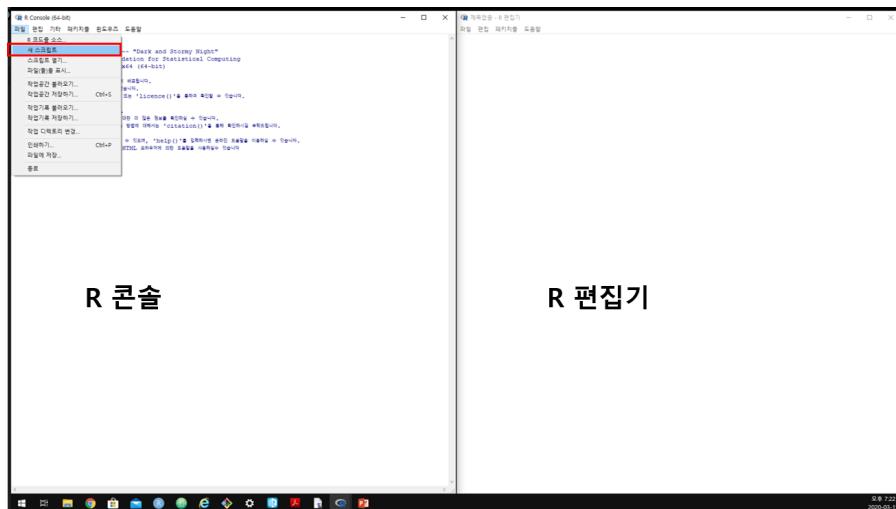
- `browseVignettes()` 명령어를 통해 vignette을 제공하는 R 패키지 및 해당 vignette 문서 확인 가능

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

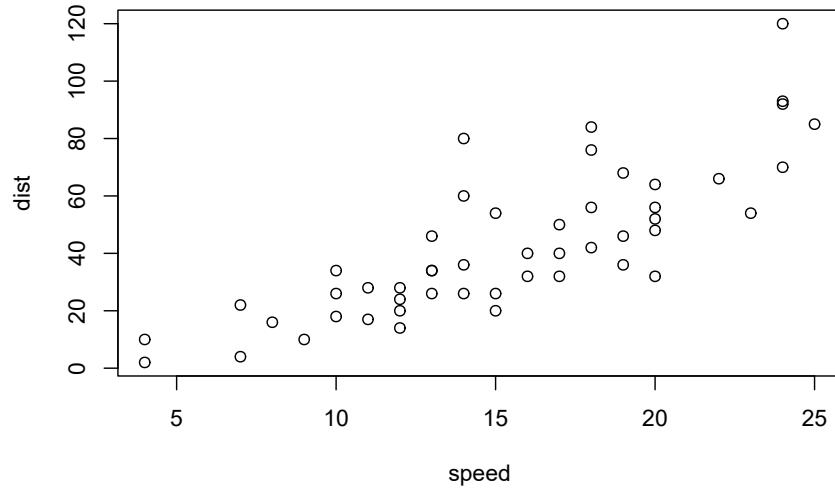
data(cars)
help(cars) # cars 데이터셋에 대한 설명 help 창에 출력
head(cars) # cars 데이터셋 처음 6개 행 데이터 출력
summary(cars) # cars 데이터셋 요약
plot(cars) # 변수가 2개인 경우 산점도 출력
```

- 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
- 콘솔창에서 ↑/↓를 누르면 이전/이후 실행 명령 기록 확인 가능

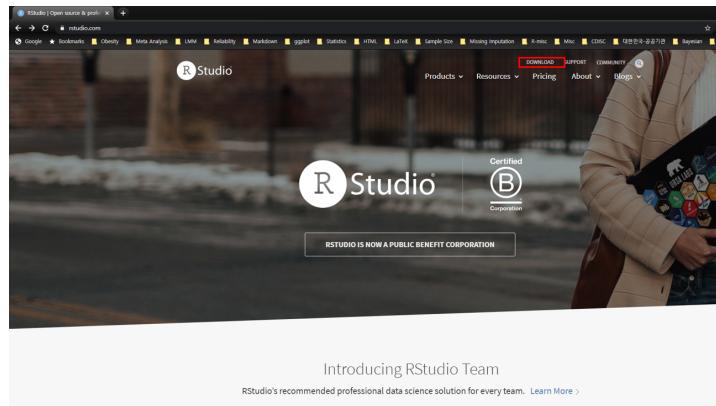


1.4 RStudio

- 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



2. Desktop 또는 Server 버전 중 택일

¹³<https://rstudio.com/>

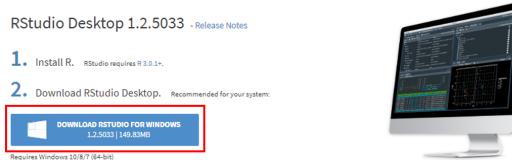
¹⁴<https://rstudio.com/products/rstudio/download/>

- 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 Products, Resources, Pricing, About, and Blogs. Below the navigation is a large blue banner with the text "Download RStudio". Underneath the banner, there's a section titled "Choose Your Version" with a brief description of what RStudio is. To the right of this description is a small image of the RStudio team logo. Below this, there are four options: RStudio Desktop (Free), RStudio Desktop (Commercial License \$995/year), RStudio Server (Free), and RStudio Server Pro (\$4,975/year). Each option has a "DOWNLOAD" or "BUY" button. The "DOWNLOAD" button for RStudio Desktop (Free) is highlighted with a red box. Below these buttons is a comparison chart showing features like "Integrated Tools for R", "Priority Support", and "Access via Web Browser" across the different versions.

	RStudio Desktop Open Source License Free	RStudio Desktop Commercial License \$995 /year	RStudio Server Open Source License Free	RStudio Server Pro Commercial License \$4,975 /year (5 Named Users)
DOWNLOAD		BUY	DOWNLOAD	BUY
Learn more	Learn more	Learn more	Learn more	Evaluation Learn more
Integrated Tools for R	✓	✓	✓	✓
Priority Support		✓		✓
Access via Web Browser			✓	✓

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



All Installers

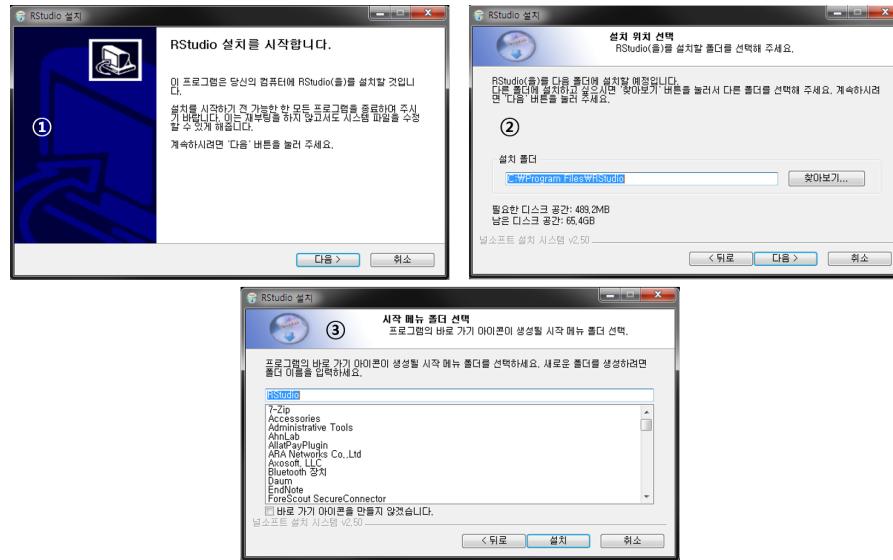
Linux users may need to import RStudio's public code-signing key prior to installation, depending on the operating system's security policy.
RStudio 1.2 requires a 64-bit operating system. If you are on a 32 bit system, you can use an older version of RStudio.

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	94.18 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	a52b1d0d

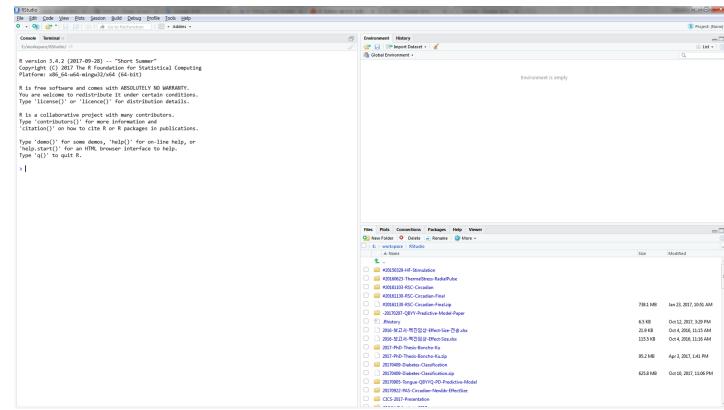
4. RStudio installer 다운로드 시 파일이 저장된 폴더에서 보통

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

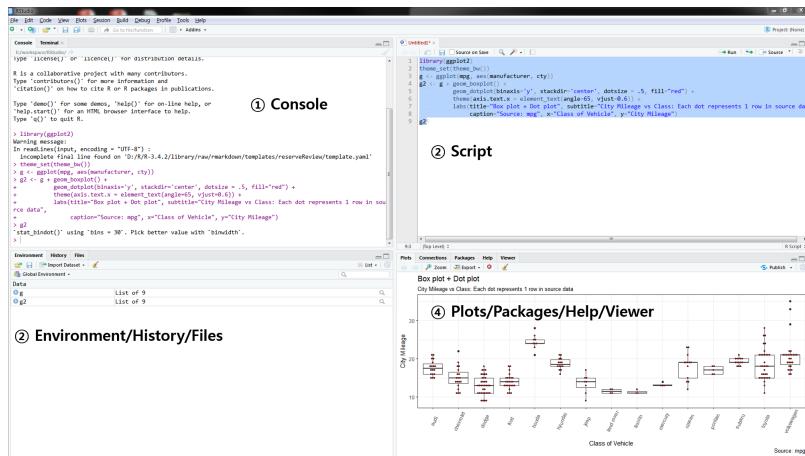
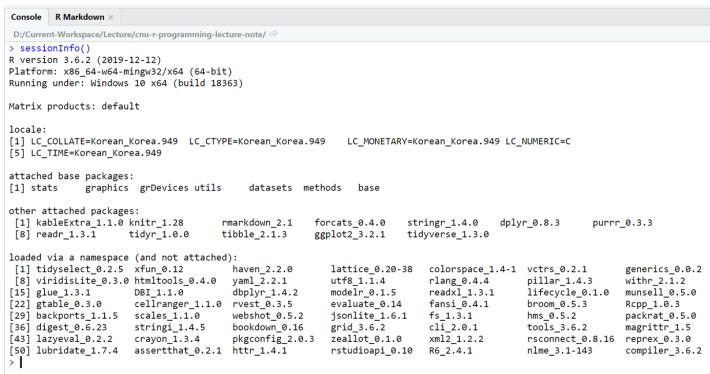


FIGURE 1.4: RStudio 화면구성: 우하단 그림은 <http://r-statistics.co/Top50-Ggplot2-Visualizations-MasterList-R-Code.html>에서 발췌

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

1. 콘솔(console)

- R 명령어 실행 공간 (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



```

Console | R Markdown | D:/Current Workspace/Lecture/cnu-r-programming-lecture-note/ ⓘ
> sessionInfo()
R version 3.6.2 (2019-12-12)
Platform: x86_64-w64-mingw32/x64 (64-bit)
Running under: Windows 10 10364 (build 18363)

Matrix products: default

locale:
[1] LC_COLLATE=korean_Korea.949  LC_CTYPE=korean_Korea.949   LC_MONETARY=korean_Korea.949 LC_NUMERIC=korean_Korea.949
[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    tidyverse_1.3.0 tibble_2.1.3   ggplot2_3.2.1 tidyverse_1.3.0

loaded via a namespace (and not attached):
[1] tidyselect_0.2.5 xfun_0.12      haven_2.2.0    lattice_0.20-38 colorspace_1.4-1  vctrs_0.2.1    generics_0.0.2
[8] viridislite_0.3.0 htmltools_0.4.0 yaml_2.2.1     utf8_1.1.4     rlang_0.4.4     pillar_1.4.3    withr_1.1.2
[15] glue_1.3.2     DBI_1.1.0     dbplyr_1.4.2   modelr_0.1.5  readxl_1.3.1   lifecycle_0.1.0  munsell_0.5.0
[22] grid_3.6.2     curl_4.3.1    scales_1.1.0   assertthat_0.1.4  rlang_0.4.1    broom_0.5.1    viridis_0.5.1
[29]槐花包_1.1.5  scales_1.1.0   webshot_0.5.2  jsonlite_1.6.1  fansi_0.3.1    here_0.5.2    packrat_0.5.0
[36] digest_0.6.23  stringr_1.4.5  bookdown_0.16   grid_3.6.2     cli_1.2.0.1   tools_3.6.2    magritr_1.5
[43] lazyeval_0.2.2 crayon_1.3.4    pkgconfig_2.0.3  grid_3.6.2     xrl_1.2.2    rconnect_0.8.16  reprex_0.3.0
[50] lubridate_1.7.4 assertthat_0.2.1  httr_1.4.1    zealot_0.1.0   R6_2.4.1     nime_3_1-143   compiler_3.6.2
> |

```

FIGURE 1.5: RStudio 콘솔창에서 명령어 실행 후 출력결과 화면

2. 스크립트(script) (Figure 1.6)

- R 명령어 입력 공간으로 일괄처리 (batch processing) 가능
- 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]

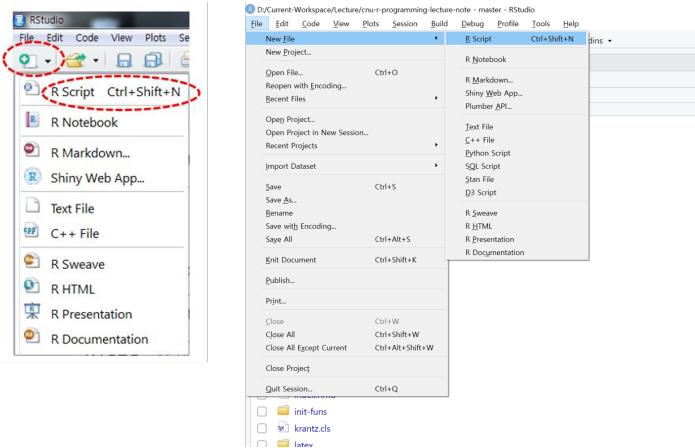


FIGURE 1.6: RStudio 스크립트 새로 열기

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 tab, each showing a different way to inspect objects:

- Top Left:** Shows the standard Environment view with objects: cars, mpg, and tab.
- Top Right:** Shows the same environment with the 'cars' object selected (highlighted by a red box), displaying its details: 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
- Bottom Left:** Shows the same environment with the 'cars' object selected (highlighted by a red box), displaying its details: 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
- Bottom Right:** Shows the 'cars' dataset as a spread sheet with columns 'speed' and 'dist'. The data rows are numbered 1 to 10, with values corresponding to the first 10 observations of the 'cars' dataset.

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

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

The figure shows the RStudio History tab with the following R script history:

```

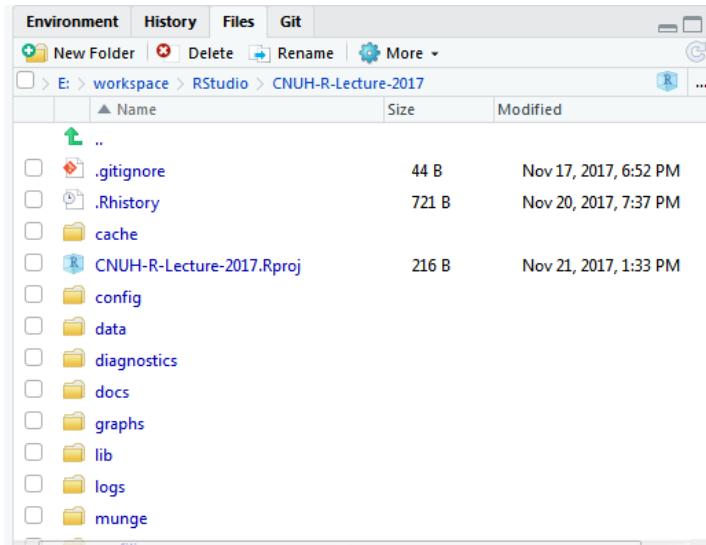
Environment History Connections Git
To Console To Source ⚡ 🔍

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)
}

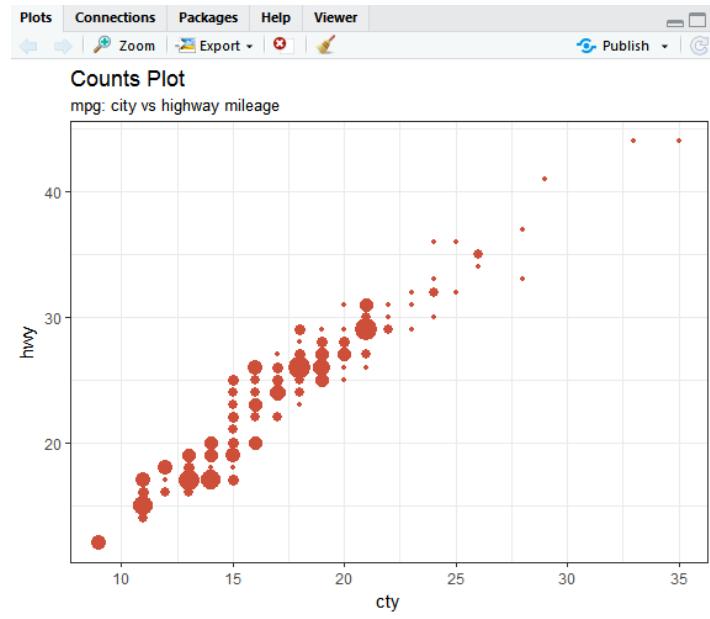
```

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

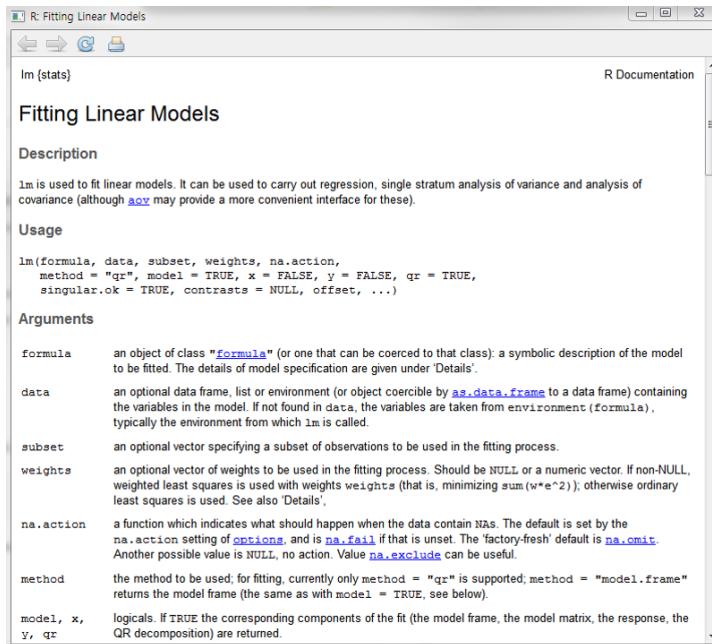


- **Packages:** 현재 컴퓨터에 설치된 R 패키지 목록 출력
 - 신규 설치 및 업데이트 가능

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

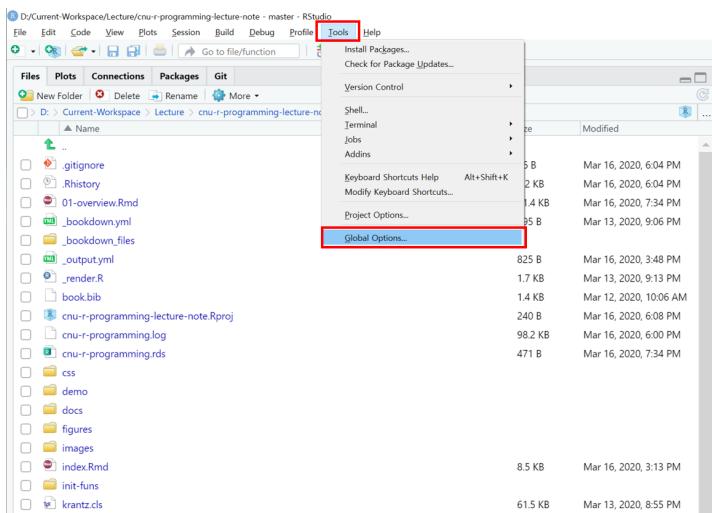
- **Help:** `help(topic)` 입력 시 도움말 창이 출력되는 공간

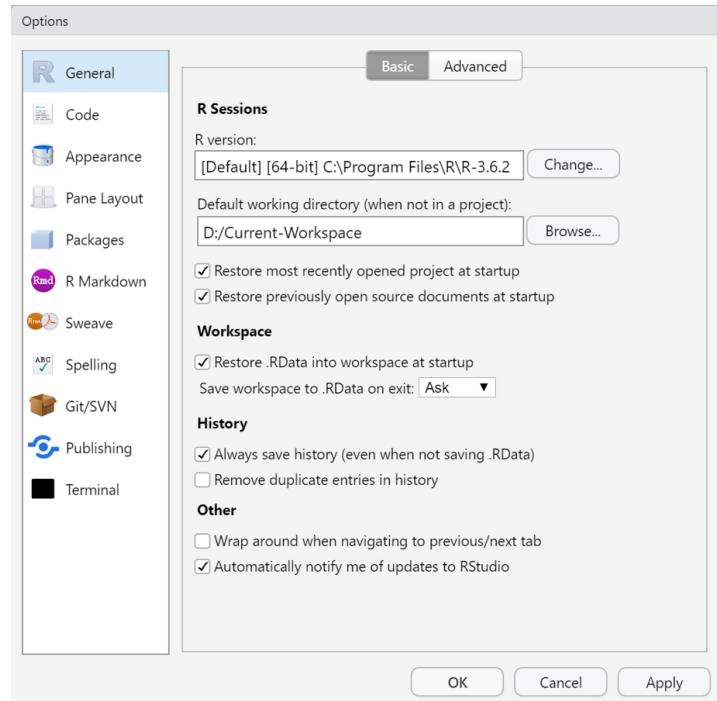
```
help(lm)
```



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

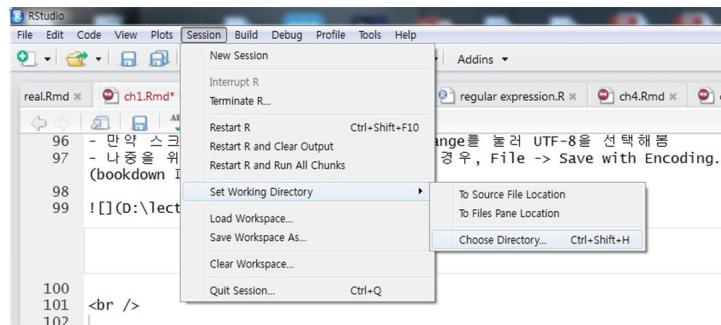
작업폴더(Working Directory)는 현재 R session에서 사용하는 기본 폴더로서 R 소스파일 및 데이터의 저장 및 로딩시 기본이 되는 폴더임.

- 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
```

```
[1] "D:/Current-Workspace/Lecture/cnu-r-programming-lecture-note"
```

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

```
[1] "D:/Current-Workspace/Lecture"
```

```
setwd("../..") # 차차상위 폴더로 이동
getwd()
```

```
[1] "D:/"
```

```
setwd("D:/Current-Workspace/Lecture/misc/") # 절대 폴더 명 입력
setwd("../")
# dir() # 폴더 내 파일 명 출력
getwd()
```

```
[1] "D:/Current-Workspace/Lecture"
```

```
setwd("misc") # Current-Workspace 하위폴더인 misc 으로 이동
getwd()
```

```
[1] "D:/Current-Workspace/Lecture/misc"
```

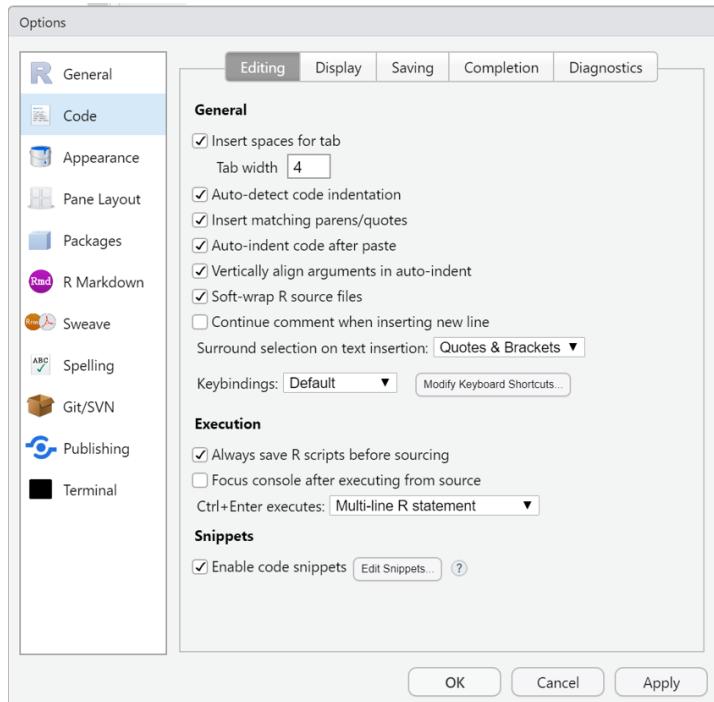
```
setwd("D:/Current-Workspace/Lecture/cnu-r-programming-lecture-note/")
getwd()
```

```
[1] "D:/Current-Workspace/Lecture/cnu-r-programming-lecture-note"
```



R에서 디렉토리 또는 폴더 구분자는 / 입. Windows에서 사용하는 구분자는 \인데, R에서 \는 특수문자로 간주하기 때문에 Windows의 폴더명을 그대로 사용 시 에러 메세지를 출력함. 이를 해결하기 위해 Windows 경로명을 그대로 복사한 경우 경로 구분자 \ 대신 \\로 변경
실습: C:\\r-project를 컴퓨터에 생성 후 해당 폴더를 default 작업폴더로 설정

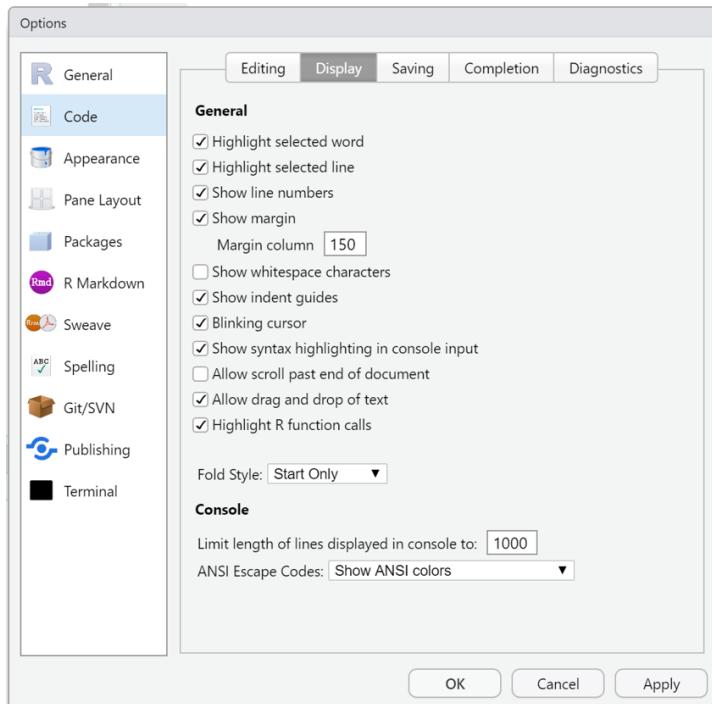
Code: Editing: 들여쓰기, 자동 줄바꿈 등 코드 편집에 대한 전반적 설정



- **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 따옴 표/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

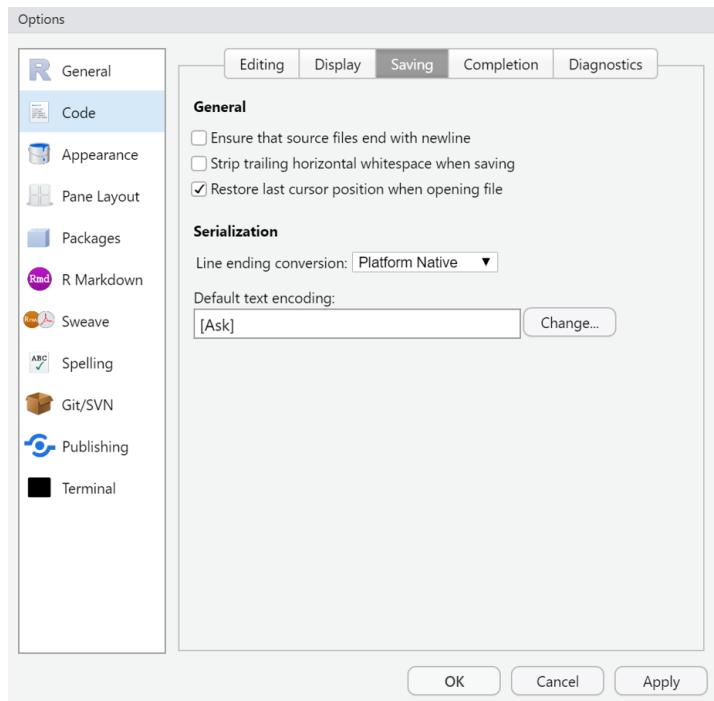
Code: Display: 스크립트(소스) 에디터 표시 화면 설정



- **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 보여주기 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

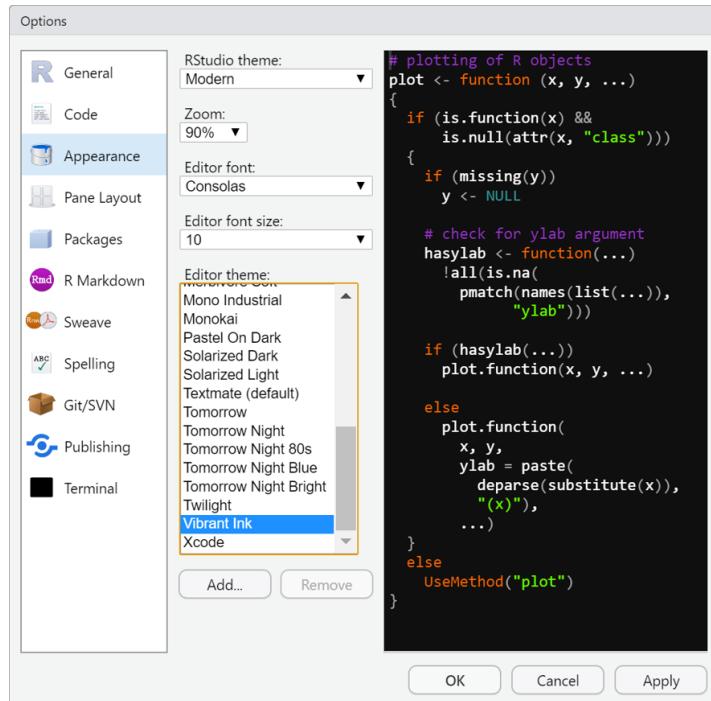
Code: Saving: 스크립트(소스) 에디터 저장 설정



- **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.

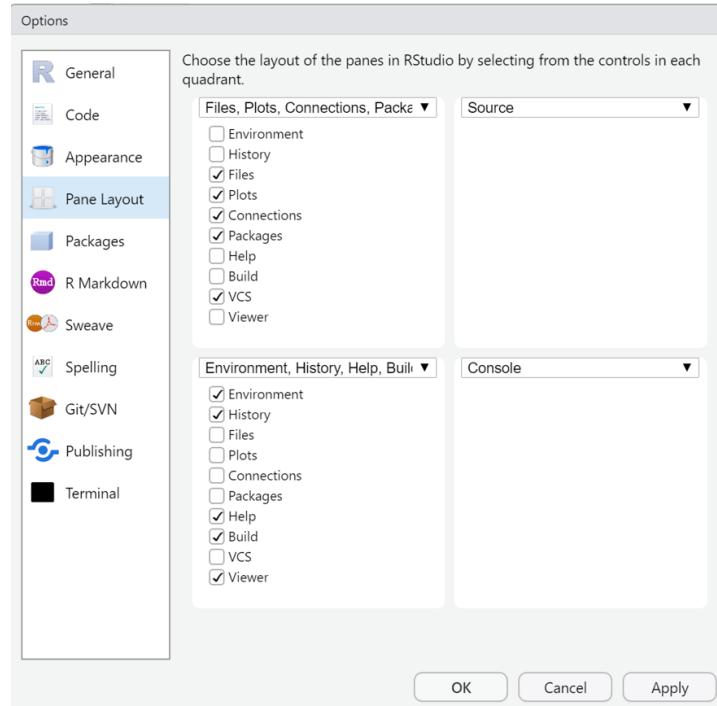
- R 언어는 Linux 환경에서 개발되었기 때문에 UTF-8 인코딩과 호환성이 더 좋음
- 스크립트 파일의 한글이 깨질 때는 [File] -> [Reopen with Encoding...]에서 encoding 방식 변경

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 고란해 질 수 있음

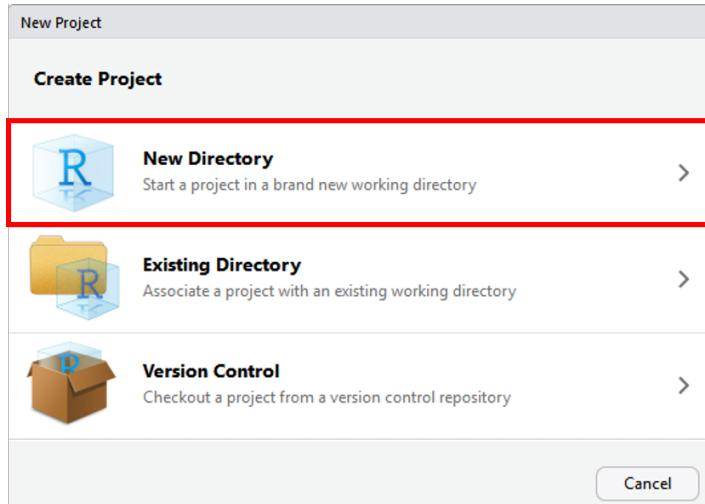
- 실제 분석 및 그래프 생성에 사용한 정확한 프로그램 또는 코드 연결이 어려움

3. 좋은 프로젝트 구성을 위한 방법

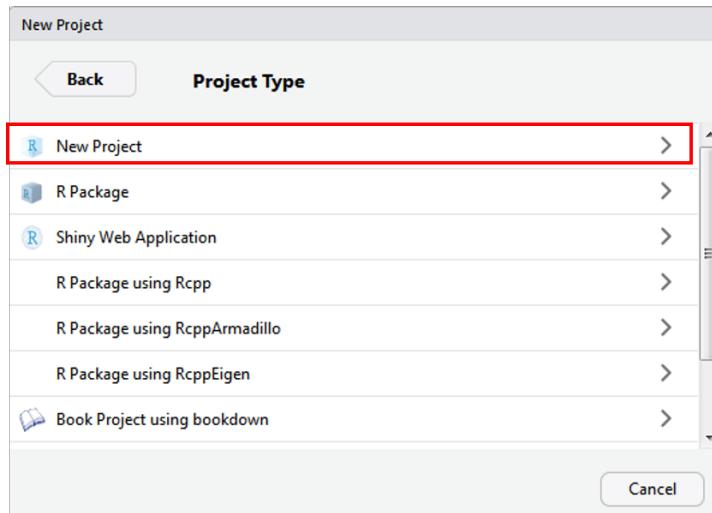
- 원자료(raw data)의 보호: 가급적 자료를 읽기 전용(read only) 형태로 다루기
- 데이터 정제(data wrangling 또는 data munging)를 위한 스크립트와 정제 자료를 보관하는 읽기 전용 데이터 디렉토리 생성
- 작성한 스크립트로 생성한 모든 산출물(테이블, 그래프 등)을 “일회용 품”처럼 처리 → 스크립트로 재현 가능
- 한 프로젝트 내 각기 다른 분석마다 다른 하위 디렉토리에 출력결과 저장하는 것이 유용

4. RStudio 새로운 프로젝트 생성

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

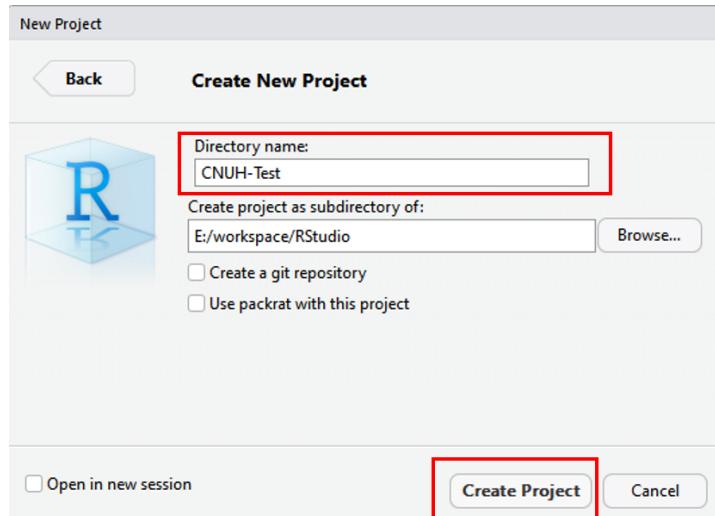


4. 위 그림에서 New Directory를 선택하면 아래와 같은 팝업 창이 나타나면 아래와 같은 프로젝트 유형이 나타남. 여기서는 New Project 선택



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



6. 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

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 name¹⁶ 또는 available package by date¹⁷에서 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] "C:/Users/user/Documents/R/win-library/3.6"  
[2] "C:/Program Files/R/R-3.6.3/library"
```

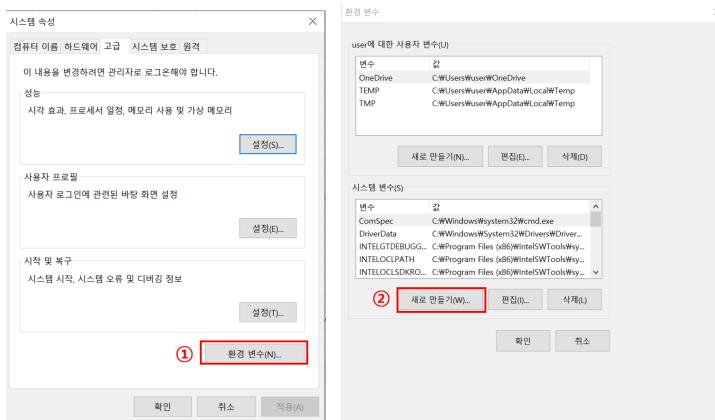
- 일반적으로 첫 번째 경로를 디폴트 라이브러리 폴더로 사용
- 사용자 지정 라이브러리 경로를 설정 하려면 아래와 같은 절차로 진행

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

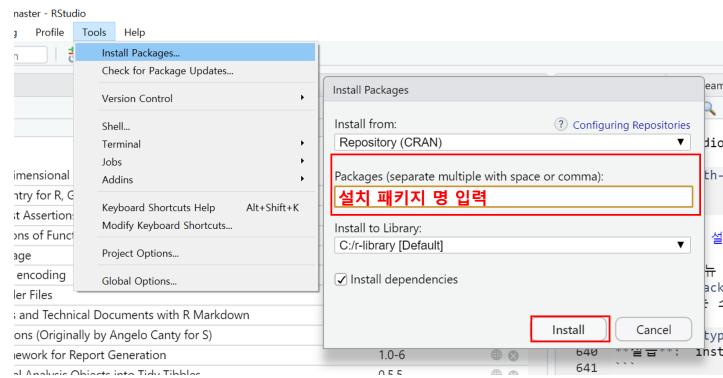
경로 C:\r-library 입력 후 확인 버튼 클릭



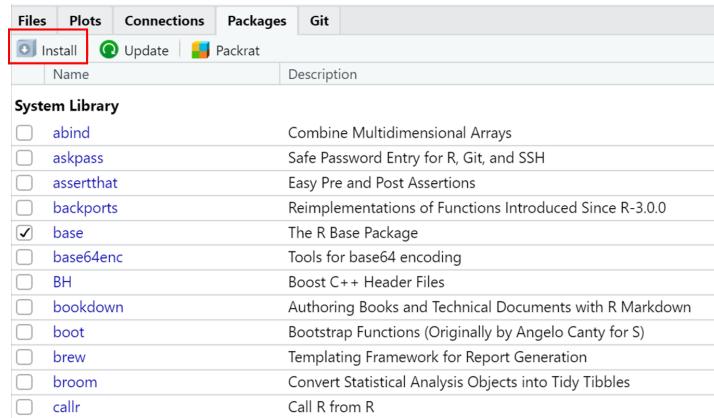
5) 현재 RStudio 종료 후 재실행한 다음 콘솔창에 .libPaths() 입력 후 라이브러리 경로 확인

1.5.2 R 패키지 설치하기

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



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



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



설명: `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 클릭
하면 R workspace로 해당 패키지 로드 가능
- 스크립트 이용

 실습: `tidyverse` 패키지 불러오기

```
require(tidyverse)
```

필요한 패키지를 로딩중입니다: `tidyverse`

```
-- Attaching packages ---

v ggplot2     3.3.0     v purrr      0.3.3
v tibble       2.1.3     v dplyr      0.8.5
v tidyverse    1.0.2     v stringr    1.4.0
v readr        1.3.1     vforcats   0.5.0
```

```
-- Conflicts -----
x dplyr::filter()      masks stats::filter()
x dplyr::group_rows()  masks kableExtra::group_rows()
x dplyr::lag()         masks stats::lag()
```



실무에서 R의 활용능력은 패키지 활용 여부에 달려 있음. 즉, 목적에 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 번들이고, 현재 R 프로그램 환경에 지대한 영향을 미침. 본 강의 “데이터프레임 가공 및 시각화”에서 해당 패키지 활용 방법을 배울 예정

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. 심화 내용은 2-3주 차에 다룰 예정임.

- R은 객체지향언어(object-oriented language)
 - 객체(object): 숫자, 데이터셋, 단어, 테이블, 분석결과 등 모든 것을 칭함
 - “객체지향”의 의미는 R의 모든 명령어는 객체를 대상으로 이루어진다는 것을 의미



알아두면 유용한(콘솔창에서 매우 많이 사용되는) 명령어 및 단축기

- **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

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

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

```
rm(x) # 객체 x 삭제
ls()
```

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

```
rm(a,b) # 객체 a, b 동시 삭제
ls()
```

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

```
# 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
# 1에서부터 (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** 결과 마찬가지로 R Markdown을 이용해 최소한의 문서(html 문서)를 작성하고 생성하는 방법에 대해 기술함. R Markdown에 대한 보다 상세한 내용은 본 수업의 마지막 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 연구¹⁸ 가능

¹⁸과학적 연구의 결과물을 오픈소스로 내놓고 누구라도 검증 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

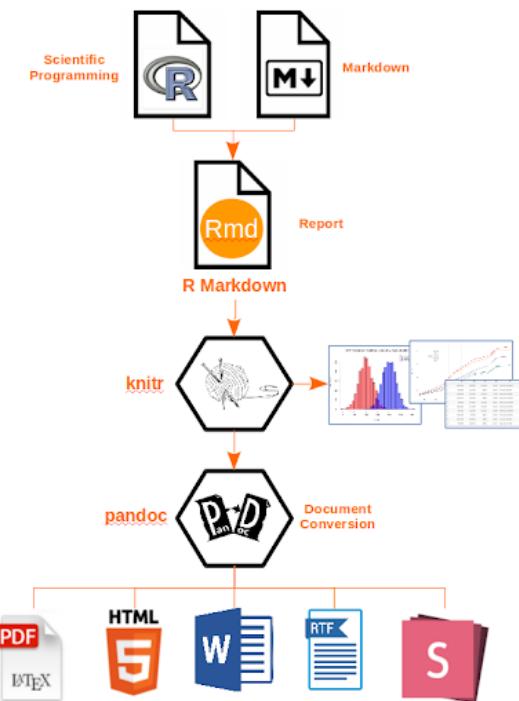


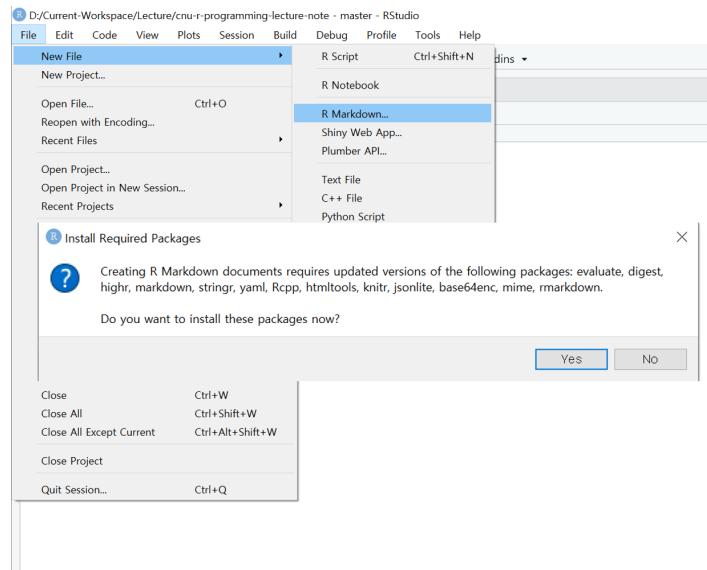
FIGURE 1.9: R Markdown의 최종 결과물 산출과정 (<http://appliedr.com/project-reporting-template/>)

R Markdown 문서 시작하기

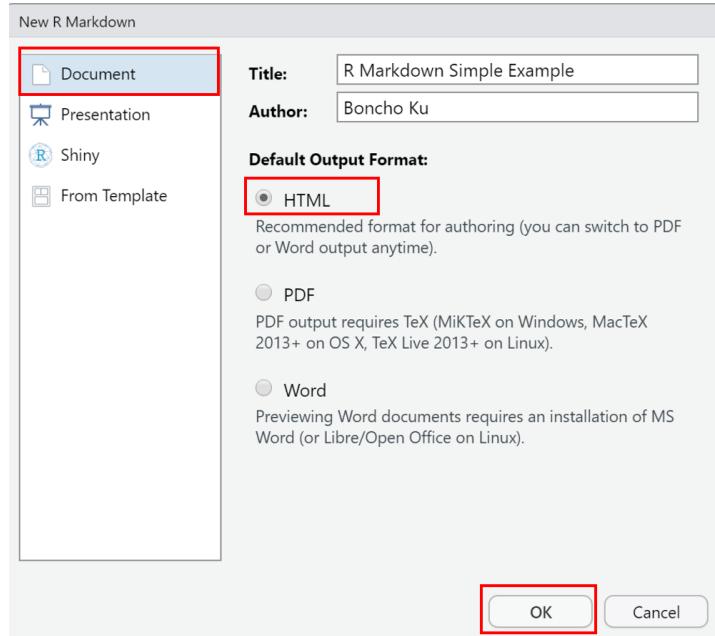
- R Markdown 문서 생성 : [File] -> [New File] -> [R Markdown..]을 선택



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



- 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 저장¹⁹

```

1 <!-- 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
2 # Markdown see <a href="http://rmarkdown.rstudio.com">http://rmarkdown.rstudio.com.
3
4 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
5 within the document. You can embed an R code chunk like this:
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
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

```

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

¹⁹RStudio 프로젝트에서 생성한 폴더 내에 파일 저장

The screenshot shows the RStudio interface with several tabs at the top: Untitled1, README.md, preamble-krantz.tex, and _bookdown.yml. A context menu is open over the following R Markdown code:

```

1 -> Knit to HTML
2 au Knit to PDF
3 da Knit to Word
4 ou Knit with Parameters...
5 -> Knit Directory
6 << E>
7 kr Clear Knitr Cache...
8 << E>
9 << E>
10 << E>
11 << E>
12 ## R Markdown
13
14 This is an R Markdown document. Markdown is a simple form
      of plain text. For more details see <http://rmarkdown.rstudio.com>.

```

The 'Knit' button in the menu is highlighted, and the tooltip 'Example' is visible.

- 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 풀더 내 'test.html'을 크롬 브라우저로 실행)

1.7.0.1 R Markdown 문서 구성

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

The screenshot shows the RStudio interface with several tabs open. The active tab is 'Untitled1.Rmd'. The code editor contains the following content:

```

1 title: "R Markdown Simple Example"
2 author: "Boncho Ku"
3 date: "2020-03-17"
4 output: html_document
5
6
7 (r setup, include=FALSE)
8 knitr::opts_chunk$set(echo = TRUE)
9 ...
10
11 ## R Markdown
12
13 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
14 Markdown see <http://rmarkdown.rstudio.com>.
15
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
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
```

Annotations highlight specific sections: 'YAML' highlights the first six lines, 'R Markdown text' highlights the explanatory text starting at line 13, and 'Code Chunk' highlights the code chunk starting at line 26.

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.2²⁰ 참고
- 최소 형태의 YAML 예시

```

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

2. Markdown 텍스트

- Markdown 문법은 15주 차 강의에서 배울 예정임
- R Markdown 레퍼런스 가이드²¹ 참조
- 그림 삽입: ![] (path/filename)

²⁰<https://bookdown.org/yihui/bookdown/r-markdown.html>

²¹<https://rstudio.com/wp-content/uploads/2015/03/rmarkdown-reference.pdf>

그립 삽입 구문

```
! [] (figures/son.jpg)
```



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 `~~`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 가이드²² 참조

Code chunk 예시

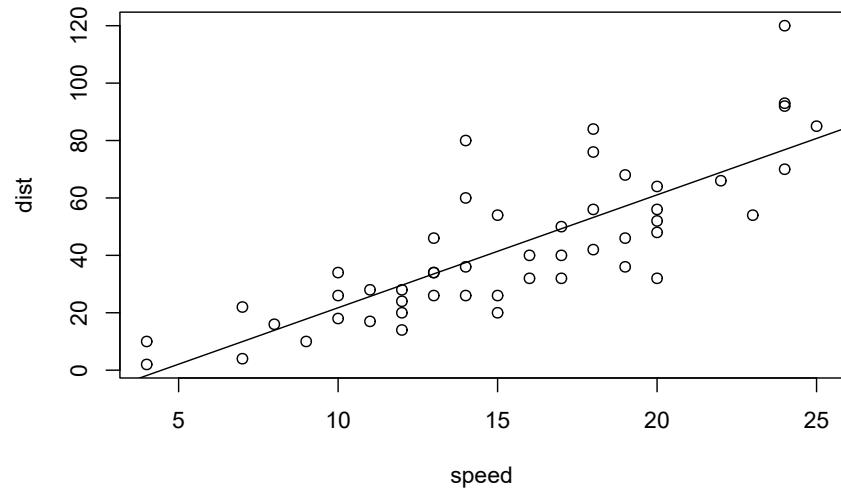
Xie의 R Markdown: The Definitive Guide에서 발췌

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

```

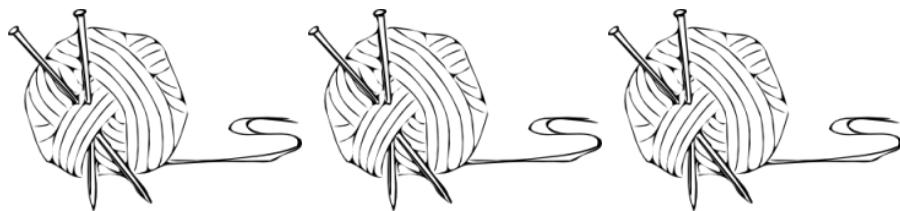
```
fit = lm(dist ~ speed, data = cars)
b   = coef(fit)
plot(cars)
abline(fit)
```

²²<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)
- 패키지 `ggplot2`를 불러오고 `cars` 데이터셋의 2 차원 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

- R 언어는 타 프로그래밍 언어와 유사한 데이터 타입(정수형, 실수형, 문자형 등)을 제공
- R 언어가 다른 언어와 차이점 → 데이터 분석에 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

R의 데이터 타입

- 수치형(numeric): 숫자(정수, 소수)
- 문자열(string): "충남대학교", "R강의"

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

- 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)

아래 그림은 2~4 주차에 배운 R 주요 객체에 대한 개요도임

2.1 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.

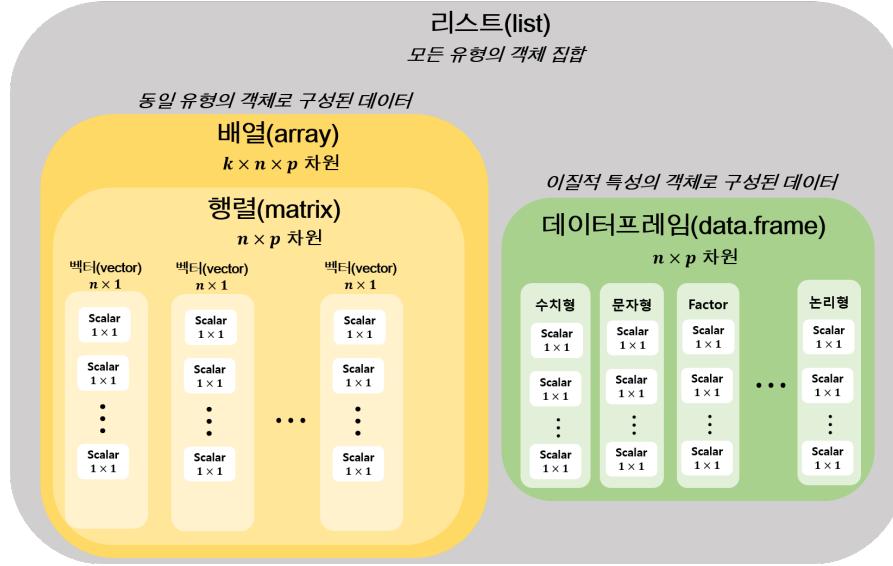


FIGURE 2.1: R 데이터 타입 구조 다이어그램: [R, Python 분석과 프로그래밍 (by R Friend)](<http://rfriend.tistory.com/>)에서 발췌 후 수정

2.1.1 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
```

2.1.2 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 2.1: R언어의 기본 수치 연산자

| 수치형 연산자 | 설명 |
|-----------------|----------------|
| +, -, *, / | 사칙연산 |
| n %% m | n을 m 으로 나눈 나머지 |
| n %%/ m | n을 m 으로 나눈 몫 |
| n ^ m 또는 n ** m | n 의 m 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
```

```
# 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
```

```
# 멱승  
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
```

2.1.3 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
```

2.1.4 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 2.2: R 언어의 논리형 연산자

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

- 비교 연산자를 적용할 경우 논리값을 반환

TABLE 2.3: R 언어의 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 가능하지만, 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 # 원상복귀
# 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
l.01 <- c(TRUE, TRUE, FALSE, TRUE) # 논리형 값으로 구성된 벡터
l.02 <- c(FALSE, TRUE, TRUE, TRUE)
l.01 & l.02 # l.01과 l.02 각 원소 별 & 연산
```

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

```
l.01 && l.02 # l.01과 l.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
```

2.1.5 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

2.1.6 NULL 값

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

```
x <- NULL # NULL 지정
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 다른  
x <- NA  
is.null(NA)
```

```
[1] FALSE
```

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

```
logical(0)
```

2.1.7 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.

2.2 벡터 (vector)

2.2.1 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.
- 2.1** 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
 - 변환 규칙: 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
```

2.2.2 벡터의 연산

- 원소 단위 사칙연산 및 비교연산 수행 → 벡터화 연산(vectorized operation)
 - 예를 들어 $\mathbf{x} = [1, 2, 3]^T$ 이고, $\mathbf{y} = [2, 3, 4]^T$ 라고 할 때 $\mathbf{x} + \mathbf{y}$ 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(+, -, *, /)
# 벡터 vs. 벡터
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 (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 벡터를 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 # 5을 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 연산이든 NA 값을 반환

```
# 결측을 포함한 벡터
x <- c(1:10, 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
```

2.2.3 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 x 의 i 번 째 요소
- $x[start:end]$: x 의 $start$ 부터 end 까지 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
```

- $x[idx_vec]$: 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 반복
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
```

2.2.4 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)
)

# 기타 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 x 의 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 x 의 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)**: X 와 Y 가 동일한지 판단 ($X = Y$) \rightarrow 논리값 TRUE 또는 FALSE 반환

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

```
[1] TRUE
```

- **union(X, Y)**: X 와 Y 의 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)**: X 와 Y 의 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"

2.3 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
```

2.3.1 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 10이고 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

2.3.2 리스트 색인

- 리스트에 포함된 객체에 접근은 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 2.4: 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 lst 에서 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
```

2.3.3 리스트에 원소 추가/제거

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

```
# 리스트 lst에 5회 차 퀴즈 점수 추가  
lst$quiz <- c(10, 8, 9, 9, 8)  
  
# 리스트 lst의 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-"
```

2.3.4 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] "BO"
```

2.4 행렬 (matrix)



학습목표(3 주차): 행렬, 배열, 요인형과 테이블에 대해 살펴보고, 이를 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 뉴어리 $\rightarrow n \times p$ 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,]    3    6    9    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 배수가 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
```

2.4.1 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 재사용되지 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 재사용

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

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

```
#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 \mathbf{y} 의 길이가 같은 경우 \rightarrow 벡터 \mathbf{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 길이와 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 \mathbf{X} 와 벡터 \mathbf{y} 의 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, 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 $ad - bc$ 가 2×2 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

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

2.4.2 행렬의 색인

- R의 행렬 객체 내 데이터 접근은 벡터와 유사하게 행과 열에 대응하는 색인 또는 이름으로 접근 가능
- 행렬의 행과 열은 괹쇠 '[']' 안에서 ,(콤마)로 구분
- $X[\text{idx_row}, \text{idx_col}]$: 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
```

2.4.3 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 붙혀서 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.

2.4.4 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"
```

```
# 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
```

```
# x와 X에 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"
```

```
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 지정해서 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"
```

2.4.5 벡터와 행렬의 차이점

- 행렬은 개념적으로 $n \times 1$ 벡터가 2 개 이상 묶여져서 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"
```

```
attributes(U)
```

```
$dim
```

```
[1] 4 2
```

2.4.6 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"
```

```
z # 행렬이 변환됨을 유의
```

```
[,1]
[1,]    2
[2,]    6
```

2.5 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 네 단위로 수집한다면? → 한 대상자에 해당하는 변수들은 시간에 따라 변함 → 시간 차원이 하나 더 존재!
- R에서 이러한 형태의 데이터 구조를 배열 (array)이라고 지칭함

2.5.1 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
```

2.5.2 배열의 확장 예제

데이터 사이언스 스쿨^{L2} 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

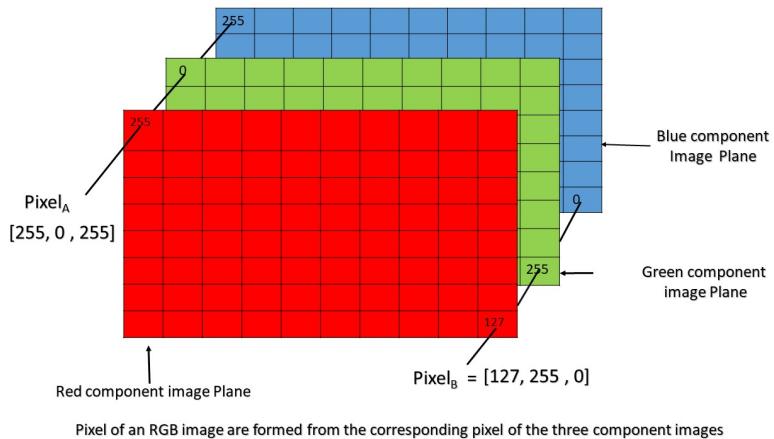


FIGURE 2.2: <https://www.geeksforgeeks.org/matlab-rgb-image-representation/>에서 발췌

 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
```

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)
```



2.6 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

2.6.1 요인 (factor)

- 범주형 자료를 표현하기 위한 R의 객체 클래스
- Factor는 정수형 벡터를 기반으로 levels (수준)이라는 속성이 추가된 객체임
- 숫자 또는 문자로 표현되었다 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
```

```
# 값은 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

2.6.1.0.1 *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

2.6.2 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 u를 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 3
4 4 NA

```
5 2 2
```

- u의 값과 상관 없이 두 factor 형 변수 f1과 f2의 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

2.6.2.0.1 *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
```

2.6.2.0.2 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 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
```

2.6.2.0.3 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"
```

```
# 테이블 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
```

| | gender | F | M |
|-------|--------|---|---|
| grade | year | | |
| 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 |

2.6.2.0.4 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 O O O
```

```
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
```

2.6.2.0.5 prop.table() 함수

- table 객체 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

$$\text{비율 } (n_{ij}/n_{.j}), n_{.j} = \sum_{i=1}^I n_{ij}$$

```
# 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
```

TABLE 2.5: 스프레드시트 기본 형태 예시

| 이름 | 직장 | 나이 |
|-----|------|----|
| 김어준 | 딴지일보 | 51 |
| 주진우 | 시사인 | 46 |
| 김용민 | 프리랜서 | 45 |
| 정봉주 | 정당인 | 59 |

2.7 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
- 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

2.7.1 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 | 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 함수로 특히 데이터 프레임이 가지고 있는 변수들의 특징을 손쉽게 알아볼 수 있기 때문에 가장 많이 호출되는 함수 중 하나임. 숫자형 벡터에 대해서는 최솟값(minimum), 1/4 분위수(1st quantile), 중앙값(median), 평균(mean), 3/4 분위수(3rd quantile), 최댓값을 출력하고, factor 형 객체에 대해서는 factor의 각 수준 별 빈도를 출력함. 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 df에 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 factor로 형 변환 되지 않음

```
x <- 1:nrow(df)
dbp <- c(73, 70, 88, 82, 75, 77, 74, 81, 72, 64)

# df에 "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
```

```
# df의 dbp에 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

```

```

# df0에 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 프레임을 손쉽게 html 파일에 출력하는 방법

- R Markdwon의 YAML 부분에 다음과 같이 옵션을 추가하면 별다른 함수 처리 없이 데이터 프레임을 테이블 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 것이기 때문에
꼭 들여쓰기를 정확히 일치시켜야 함

```
---
```

```
title: "문서 제목"
author: "이름"
date: "`r Sys.Date()`"
output:
  html_document:
    df_print: paged
---
```

2.7.2 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, ]
```

| | 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 |

```
# id가 3보다 작은 데이터 추출
df[df[, 1] < 3, ]
```

| | 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 |

```
# 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 데이터 수정 및 추가 가능

```
df2 <- within(df, {
  hospital <- c("A", "B", "B", "A", "C",
               "A", "A", "B", "C", "B")
  mean_age <- mean(age)
})
```

2.7.3 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

2.7.3.0.1 *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 1에서 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 |

2.7.3.0.2 *View()* 함수

- 2차원 데이터의 readable 한 스프레드시트 제공

```
View(abalone)
```

데이터 프레임 결합 및 분리 함수

2.7.3.0.3 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 다르기 때문에 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)

```

2.7.3.0.4 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 | Mercy | Swiss | 37 |
| 3 | Park | Korea | 19 |
| 4 | Soldier76 | USA | 56 |
| 5 | Mei | <NA> | 31 |

2.7.3.0.5 *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

2.7.3.0.6 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
```

2.7.3.0.7 *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 |

2.7.4 *apply() 계열 함수

- `apply()`, `lapply()`, `sapply()` 등 `apply` 계열 함수는 R에서 가장 일반적으로 사용되는 함수 중 하나
- 보통 `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 2.6.1 요인 (factor) 절에서 잠깐 예시로 사용된 전복(abalone) 데이터셋 사용



Abalone dataset 변수 설명

| | 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 반환함. 따라서 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

- 2.6.1 요인 (factor) 절에서 설명
- `tapply()`는 1개의 벡터를 대상으로만 함수를 호출
- 데이터 프레임에 적용하려면? → `aggregate()` 함수 사용

2.7.4.0.1 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
)
```

- 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
```

```
# 요인인 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 | 171.6667 | 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"  
 ... . . .$. : chr "x"  
 ... .- attr(*, "term.labels")= chr "x"  
 ... .- attr(*, "order")= int 1  
 ... .- attr(*, "intercept")= int 1  
 ... .- attr(*, "response")= int 1  
 ... .- attr(*, ".Environment")=<environment: 0x000000035c165a0>  
 ... .- attr(*, "predvars")= language list(abalone$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_reg에서 coefficients만 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

- mapply()와 유사하지만 다수의 인수를 함수에 전달해 적용
- 임의의 함수 FUN()이 있고, FUN()이 사용할 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
```

2.8 Homework #2-1

1. `seq()` 함수를 사용하여 $\log(\exp(10))$ 부터 0 까지 길이가 100인 벡터를 생성 후 객체 `lambda`를 생성하시오.
2. 두 벡터 $p = c(1, 4, 2, 3, 4, 7, 9, 12)$, $q = c(4, 5, 3, 2)$ 의 사칙연산 결과를 출력하고, 왜 이런 형태로 계산이 이루어졌는지 기술하시오.
3. 집합 $A = \{1, 3, 5, 7, 8, 9, 12, 15\}$ 이고 집합 $B = \{3, 6, 9, 12, 15, 18\}$ 일 때, $A \cup B$, $A \cap B$, $A - B$ 의 결과를 출력하시오.
4. `year`라는 객체에 $\{2000, 2001, \dots, 2020\}$, `month` 객체에 $\{\text{Jan}, \text{Feb}, \dots, \text{Dec}\}$, `day` 객체에 $\{1, \dots, 31\}$ 을 저장하고 `Date`라는 `list`를 생성 후 생성 결과를 출력 하시오.
5. `x` 벡터에 $\{23, 22, 24.5, \text{NA}, \text{NA}, 28, 27.8, 31, \text{NA}, \text{NA}\}$ 를 입력하고 결측의 개수를 구하시오.
6. `tidyverse` 패키지를 불러온 후 `mpg` 데이터 셋에서 `hwy` 변수을 `x`라는 객체에 저장한 후, `x` 객체에서 24보다 작은 값들의 개수를 구하시오.
7. 1부터 150 까지 1 단위 수열을 생성 후 객체 `x`에 저장하고 `x`에서 홀수 값만 추출 하시오.

8. 두 벡터 $\{1, 2, 3, 0, -1, -2, -1, 0, 7\}$ 와 $\{6, -3, 0, 0, 4, -5, 0, 0, 2\}$ 를 각각 x와 y 객체에 저장하고, 해당 객체를 이용해 다음 행렬을 생성하시오

$$\mathbf{X} = \begin{bmatrix} 1 & 2 & 3 \\ 0 & -1 & -2 \\ -1 & 0 & 7 \end{bmatrix}, \quad \mathbf{Y} = \begin{bmatrix} 6 & 0 & 0 \\ -3 & 4 & 0 \\ 0 & -5 & 2 \end{bmatrix}$$

9. 위 두 행렬의 연산 결과를 출력 하시오

- \mathbf{XX}^T
- \mathbf{XY}
- \mathbf{YX}
- $\det(\mathbf{X})$
- \mathbf{Y}^{-1}

10. `runif()` 함수를 이용해 난수 200개를 생성하여 x라는 객체에 저장 하시오.

- 생성한 x 를 이용해 x가 0.5 보다 작으면 0, 0.5 보다 크거나 같으면 1 값을 재할당 하시오.
- 수준이 0, 1이고 수준이름이 각각 “Male”, “Female”인 요인형 객체 sex를 생성하시오.

과제 제출 방식

- R Markdown 문서(`Rmd`) 파일과 해당 문서를 컴파일 후 생성된 `html` 파일 모두 제출할 것
- 모든 문제에 대해 작성한 R 코드 및 결과가 `html` 문서에 포함되어야 함.
- 해당 과제에 대한 R Markdown 문서 템플릿은 https://github.com/zorba78/cnu-r-programming-lecture-note/blob/master/assignment/homework2_template.Rmd에서 다운로드 또는 스크래이핑 가능

- 최종 파일명은 학번-성명.Rmd, 학번-성명.html로 저장

Bibliography

- Rizzo, M. L. (2019). *Statistical computing with R*. CRC Press.
- Wickham, H. (2016). *ggplot2: elegant graphics for data analysis*. Springer.
- 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). *R*을 이용한 데이터 처리 & 분석. 길벗, 1st edition. ISBN 978-8966188260.

유충현, 이상호, and 김정일 (2005). *R* 그래픽스. 자유아카데미, 1st edition. ISBN 978-8973385539.