

R을 이용한 자료처리 및 시각화

기본 환경 설정부터 재현가능한 연구 방법 입문

한국한의학연구원 한의기반연구부

Korea Institute of Oriental Medicine, KM Fundamental Research Division

구본초, *Ph.D. in Statistics*

2017-11-23

Contents

1 R을 사용하기 위한 환경 설정	5
1.1 Overview	5
1.2 R 설치하기(Windows 용)	7
1.3 R 시작 및 작동 체크	14
1.4 RStudio 설치하기	18
1.5 RStudio의 구성	21
1.6 RStudio에서 배치 파일 생성 및 실행	25
1.7 R 패키지 설치	26
1.8 rJava 설치하기	27
1.9 RStudio 프로젝트 생성 및 ProjectTemplate 패키지 연동	30
2 R의 기본 사용	37
2.1 R의 데이터 유형	37
2.2 R 객체 유형	40
2.3 행렬	53
2.4 배열	55
2.5 데이터 프레임	56
2.6 리스트	57

Chapter 1

R을 사용하기 위한 환경 설정

1.1 Overview

1.1.1 What is R?

- R 언어: 통계 및 자료 시각화를 지원하는 언어 및 환경
- 1980년 AT&T 연구소의 John Chambers가 개발한 S 언어를 기반으로 1995년 뉴질랜드 Auckland Univ.의 Robert Gentleman & Ross Ihaka 개발이 그 기원임
- GNU 기반 오픈소스

1.1.2 Why R?

1. 장점

- Free software
- 주요 운영체제(Unix, Windows, Macintosh)에서 사용 가능
- 현존하는 거의 대부분의 통계 방법론들이 package로 구현
- 강력한 그래픽 기능
- 빠른 update 및 연산속도(?)
- 방대한 community 및 R package에 공개 및 공유된 자료

2. Do we have to learn R language?

- 물론 꼭 배울 필요는 없다!! (요리를 하는데 꼭 좋은 칼이 필요 없듯이...)

- 다양한 통계 소프트웨어 및 분석 언어(SPSS, SAS, WEKA, MATLAB, PYTHON, ...) 존재
- 프로그래밍에 익숙하지 않은 사용자들의 접근성이 떨어짐
- 하지만 배워두면 좋은 이유
 - 통계분석과 보고서 작성을 위한 최적 환경
 - * R + Rstudio + Rmarkdown을 통해 분석에서부터 보고서 작성까지 한번에 가능
 - SPSS, SAS에 비해 확장성이 높기 때문에 다양한 문제에 적용 가능
 - 방대한 양의 메뉴얼 및 서적들
 - 그리고 이 모든 것을 거의 대부분 무료 사용 가능

Tips

- R에서 구현한 모든 package들은 CRAN (The Comprehensive R Archive Network, <http://cran.r-project.org/web/view>)에서 살펴볼 수 있음
- R과 관련한 대부분의 문제는 Google 검색과 스택 오버플로우 (<http://stackoverflow.com>)에서 해결할 수 있음
- 해당 문서는 서민구 선생님의 “R을 이용한 데이터 처리 & 실무” [[서민구, 2014](#)], 고석범 선생님의 “R과 Knitr을 활용한 데이터 연동형 문서 만들기” [[고석범, 2014](#)], R Friend 님의 블로그 “R, Python 분석과 프로그래밍” [[R-F](#)] 의 내용을 주로 참조함.
- 해당 문서는 RMarkdown + L^AT_EX + knitr로 작성된 문서이며, 이를 위해 작성한 모든 소스 코드 및 자료는 <https://github.com/Secondmoon-Kiom/CNUH-R-Lecture-2017>에서 확인할 수 있음.

1.2 R 설치하기(Windows 용)

R은 공개 소프트웨어로 <http://www.r-project.org/>에서 다운로드 및 설치 가능

1. 웹브라우저(i.e. explore, chrome, firefox 등)에서 <http://www.r-project.org> 이동
2. 좌측 R Logo 하단 Download 아래 'CRAN' 클릭

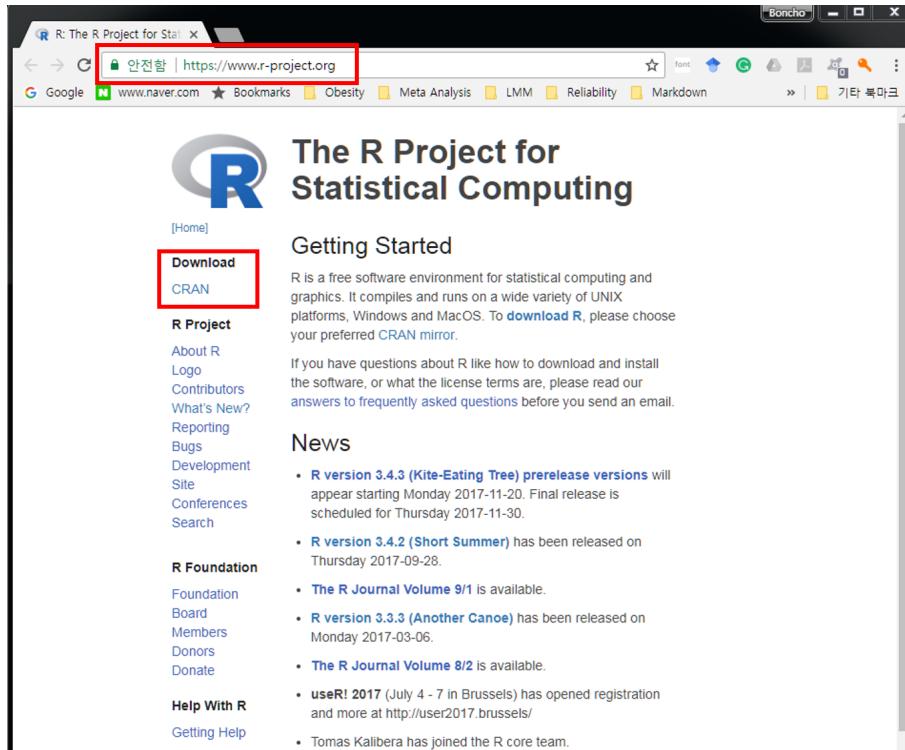


Figure 1.1: www.r-project.org 메인화면

3. 클릭 후 연결한 페이지를 스크롤 후 “Korea” 아래 링크 클릭 (그림 1.2 참조)

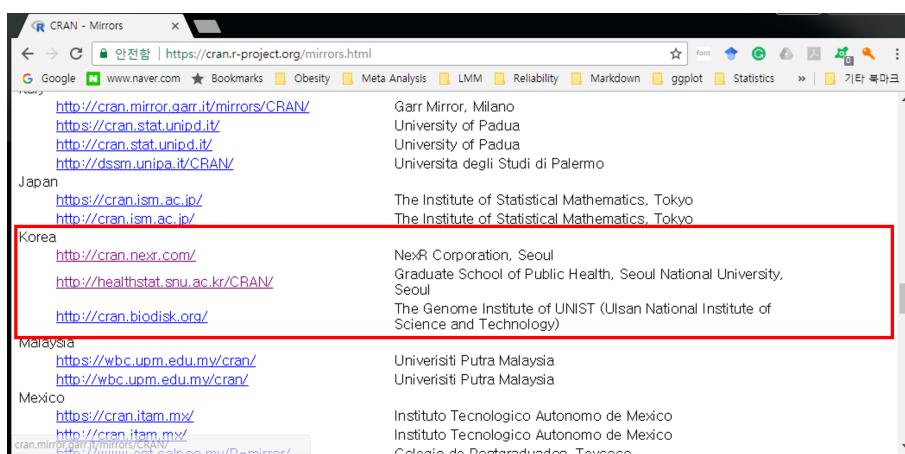


Figure 1.2: CRAN 국가별 mirrors

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

- 본 문서에서는 Windows 버전 설치만 다룸

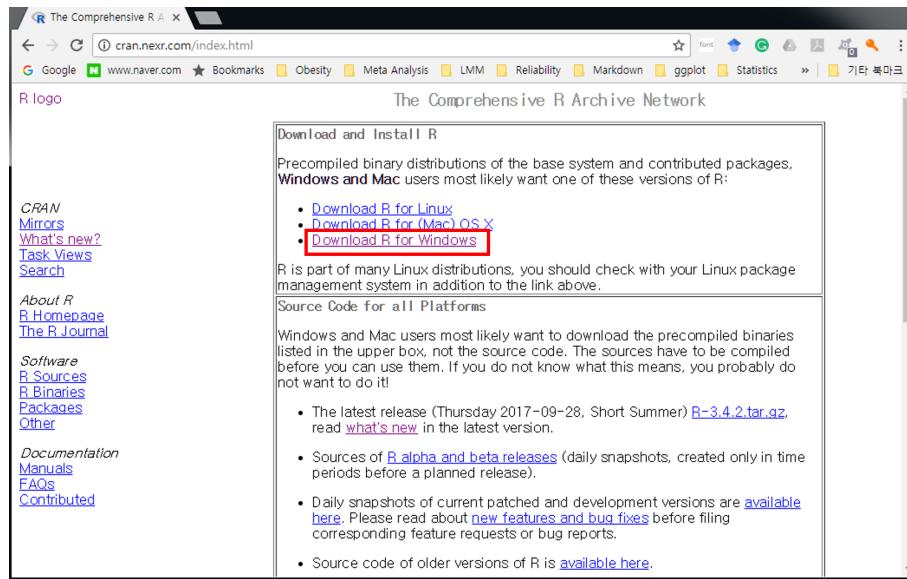


Figure 1.3: 운영체제 별 R 버전 선택

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

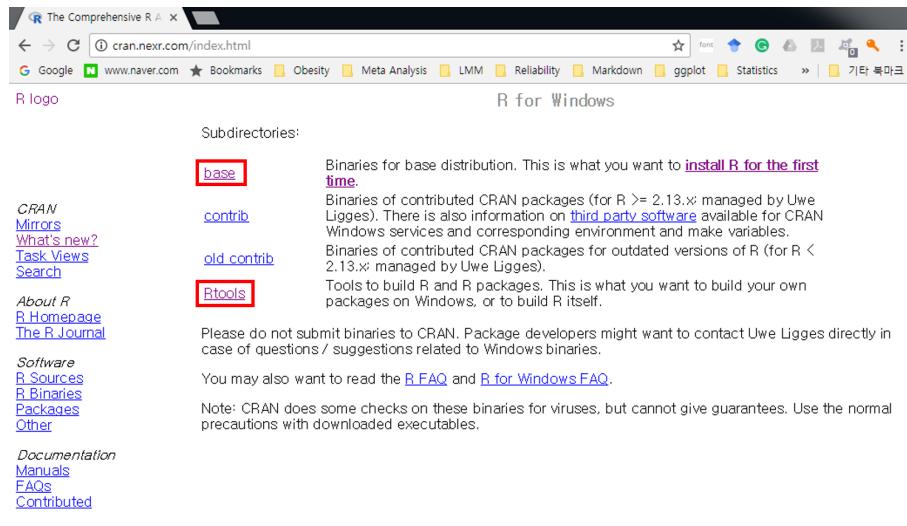


Figure 1.4: Windows 용 R base 및 구성요소 다운로드

6. R을 구성하는 하위구조 중 “base” 링크 클릭 후 다음 화면에서 가장 최신버전2017-11-20 현재 “Downloads R 3.4.2 for Windows”)를 클릭 후 설치 파일을 임의의 디렉토리에 저장 후 실행 (그림 1.5 참조)

7. 참고로 3개 subdirectories에 대한 간략한 설명은 아래와 같음

- **base:** R 실행 프로그램
- **contrib:** R package의 바이너리 파일

- Rtools: R package 개발 및 배포를 위한 프로그램

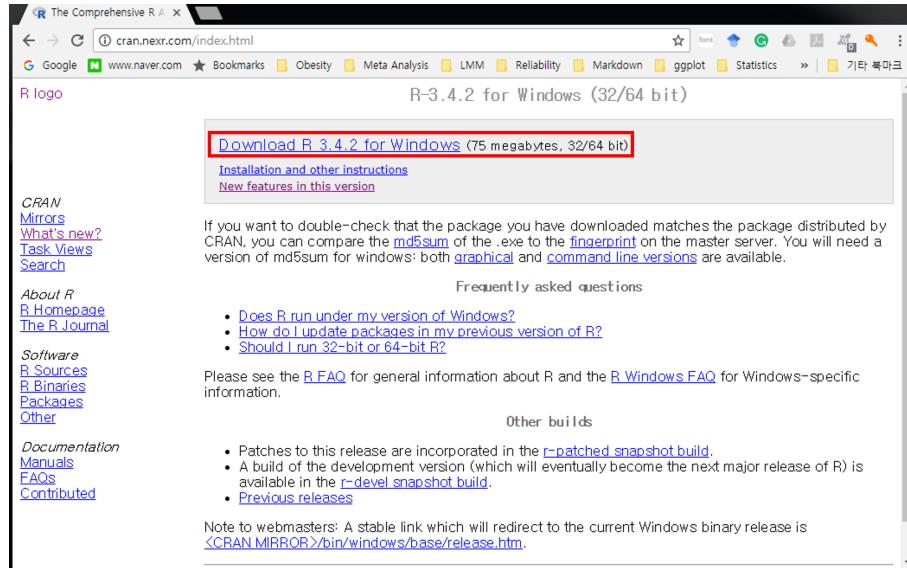


Figure 1.5: Windows 용 R 설치 파일 다운로드 페이지

8. 다운로드한 파일을 실행하면 아래와 같은 대화창이 나타남

- 한국어 선택 → 환영 화면에서 [다음(N)>] 클릭

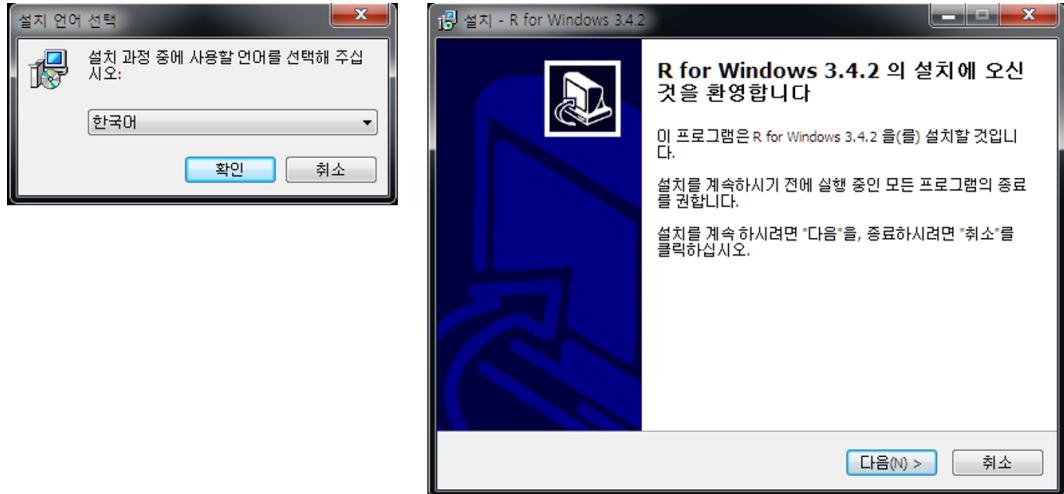


Figure 1.6: R 설치과정 01

9. GNU 라이센스에 대한 설명 및 동의 여부([다음(N)>]) 클릭 (그림 1.7)

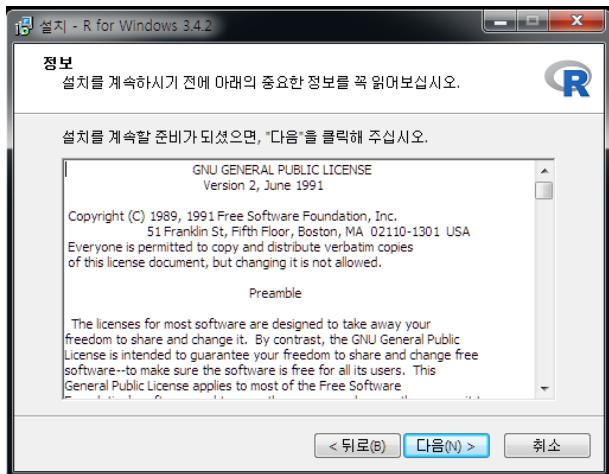


Figure 1.7: R GNU general license

10. 설치 디렉토리 설정 및 구성요소 설치 여부

- 원하는 디렉토리 설정(예: C:\R\R-3.4.2) (그림 1.8)
- 기본 프로그램("Core Files"), 32 또는 64 bit 용 설치 파일, R console 한글 번역 모두 체크 뒤 [다음(N)>] 클릭 (그림 1.9)

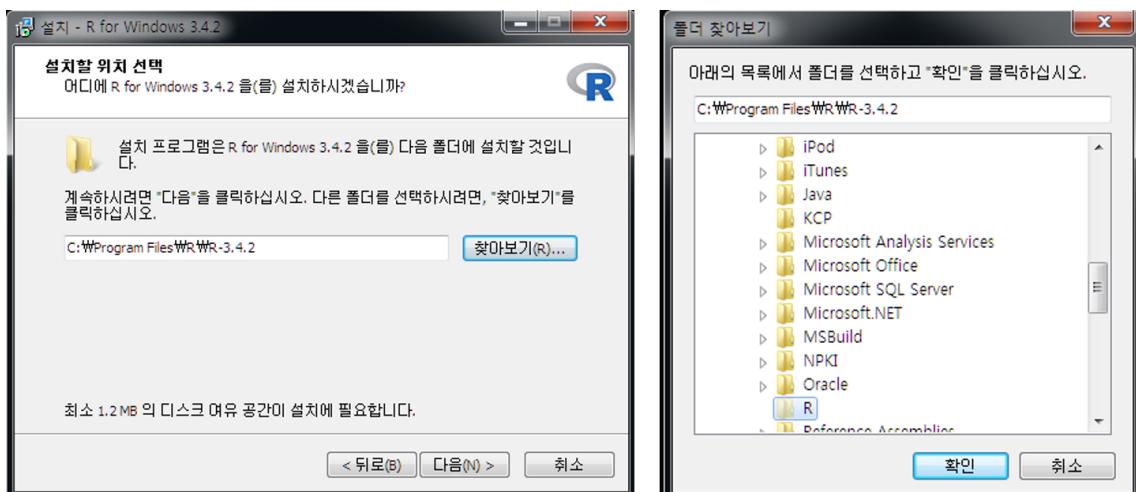


Figure 1.8: R 설치 디렉토리 설정

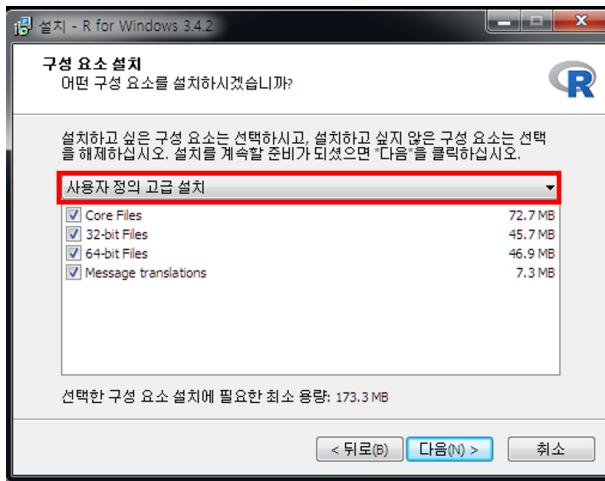


Figure 1.9: R 구성요소 설치

11. R 스타트업 옵션 지정

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

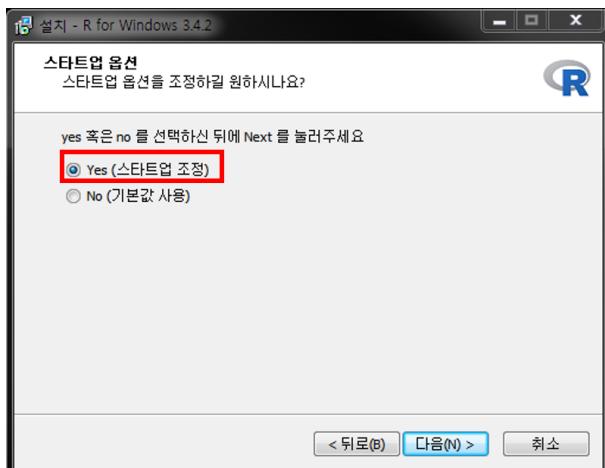


Figure 1.10: R 스타트업 옵션 변경

12. 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

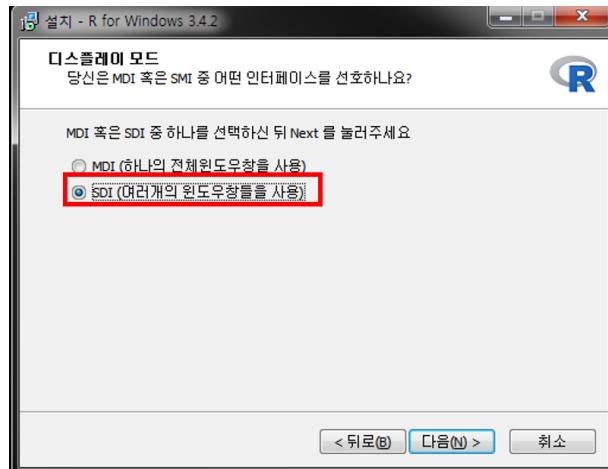


Figure 1.11: R 화면표시방식 설정 변경

13. 도움말 형식에서 HTML 도움말 기반 선택

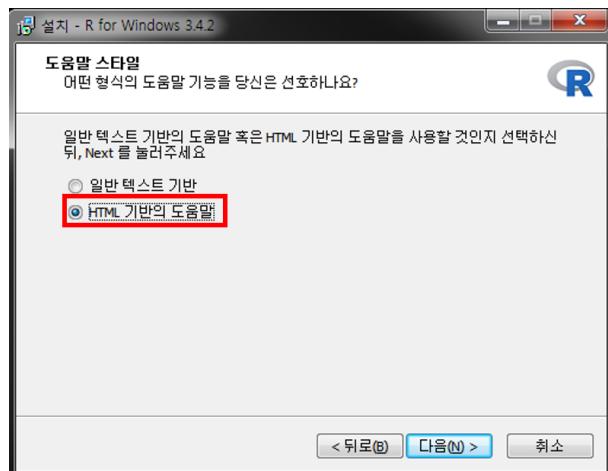


Figure 1.12: R 도움말 형식 변경

14. 시작메뉴 폴더 선택(그림 1.13)

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



Figure 1.13: 시작메뉴 폴더 선택

15. 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

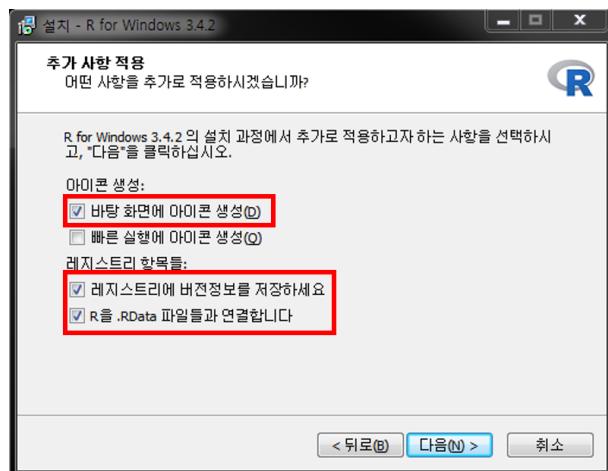


Figure 1.14: R 추가옵션 사항 선택

16. 설치 완료 후 바탕화면의 R 아이콘을 더블클릭하면 Rgui가 실행

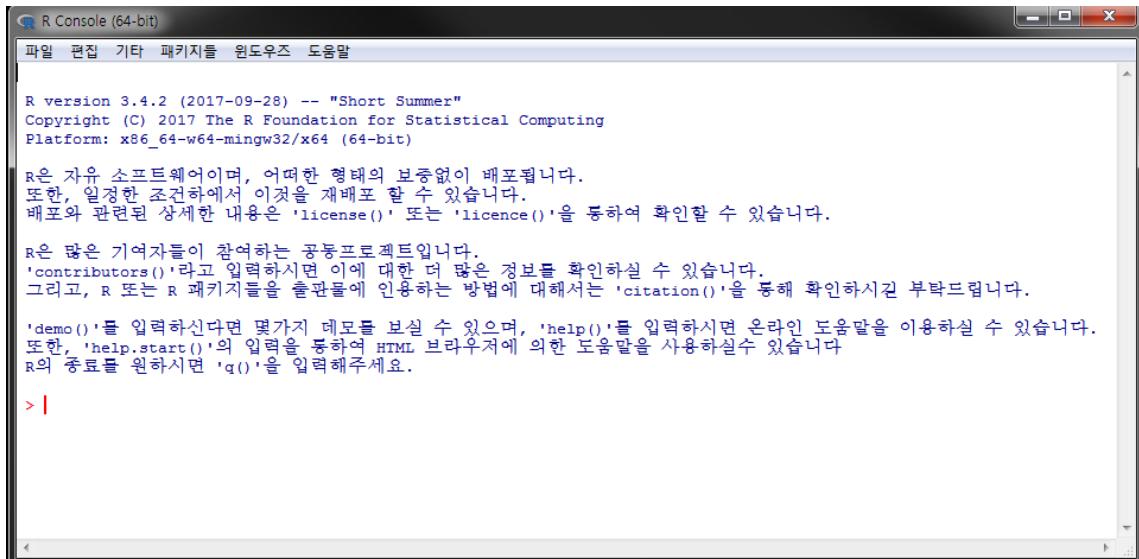


Figure 1.15: Windows에서 R 실행화면(콘솔 창, SDI 모드)

1.3 R 시작 및 작동 체크

위 R 시작화면에서 간단한 명령어들을 체크

- 그림 1.15에서 > 기호는 R의 명령 프롬프트임.
- Checklist
 - 1) “Hello R” 출력
 - 2) 1부터 100까지 정수 출력
 - 3) 간단한 histogram 출력

```
> # 문자열 출력
> print("Hello R")
```

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

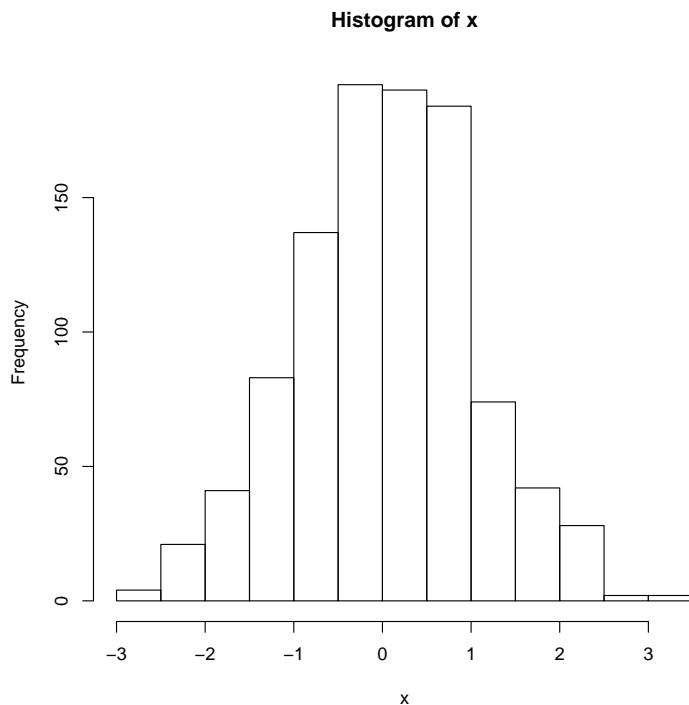
여기서 # 기호는 주석의 시작을 의미하며 같은 행에서 # 뒤 내용의 코드는 실행되지 않음

```
> # 1부터 100까지 수열 출력
> seq(1:100) # print('Hello R')
```

```
[1]   1   2   3   4   5   6   7   8   9   10  11  12  13  14  15  16  17
[18]  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34
[35]  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51
```

```
[52] 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68
[69] 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85
[86] 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
```

```
> # 간단한 히스토그램
> set.seed(12345) # random seed 지정
> x <- rnorm(1000) # 평균 0, 분산 1인 정규분포에서 난수 1000개 생성
> hist(x) # 히스토그램
```



Tips

R 명령어 실행 시 간혹 아니 매우 빈번히 오류가 나타나는데, 이를 해결할 수 있는 가장 손쉬운 방법은 Googling과 R의 도움말을 이용하는 것이 가장 효율적임.

도움말을 볼 수 있는 R 함수 리스트

Table 1.1: R help 관련 명령어 리스트

도움말 보기 명령어	설명	사용법
help 또는 ?	도움말 시스템 호출	help(topic # 도움말을 찾고자 하는 대상 또는 함수)
help.search 또는 ??	주어진 문자열을 포함한 문서 검색	help.search(pattern # 찾고자 하는 문자열)
example	topic의 도움말 페이지에 있는 examples section 실행	example(topic # 예제를 실행하고자 하는 topic 또는 함수)
vignette	topic의 pdf 또는 html 레퍼런스 메뉴 열 불러오기	vignette(topic # topic에 저장된 reference manual)

Tips

- vignette에서 제공하는 문서는 실제 분석을 기반으로 작성한 문서이기 때문에 초보자들이 R 패키지의 접근성을 높여줌. browseVignettes()으로 현존하는 모든 R 패키지들의 vignette을 볼 수 있기 때문에 매우 유용함.
- R 세션(R 콘솔이 시작해서 종료까지)에 설정된 옵션은 options() 실행으로 확인 가능
- 현재 R 세션에 대한 정보는 sessionInfo() 함수로 확인 가능

참고

```
> # 현재 설정된 출력 자리수 확인  
> options("digits")
```

```
$digits
```

```
[1] 7
```

```
> pi
```

```
[1] 3.141593
```

```
> # 출력 자리수를 7에서 3으로 변경  
> options(digits = 3)  
> pi
```

```
[1] 3.14
```

1.3.1 이것으로 R 설치 완료??

- 기본적 R 사용방식은 입력한 명령어와 실행결과를 확인하는 대화형(interpreter) 방식
- R 기본 콘솔창 안에서도 TAB을 누르면 자동완성 기능이라던가 ↑, ↓를 누르면 이전/이후 명령 기록을 볼 수 있음.
- 여러 줄 이상의 R 명령어라든가 반복적, 장기간 작업을 수행해야 하는 경우라면 R 명령어로 구성된 스크립트 작성 후 일괄 실행하는 것이 일반적임.
- 이러한 다중 명령 코딩 시 콘솔창에 직접 입력하는 것은 비효율적 → 스크립트 에디터를 주로 사용
- R 자체적으로 기본적인 스크립트 에디터 제공(R editor) → 가독성 및 코딩 효율이 떨어짐
- 대표적 R 에디터: WinEdt (<http://www.winedt.com>), Tinn-R (<https://sourceforge.net/projects/tinn-r/>), Vim (http://www.vim.org/scripts/script.php?script_id=2628)

- 대부분의 분석 및 개발 환경이 R GUI 만으로 구성되어 있지 않음 → **RStudio**를 이용한 통합 분석 환경 설정

1.4 RStudio 설치하기

- Rstudio: R 통합 분석/개발 환경(integrated development environment, IDE)으로 현재 가장 대중적으로 사용되고 있는 R 사용 환경
- 명령 콘솔 외 파일 편집, 데이터 보기, 명령 기록(.history), 그래프 등에 쉽게 접근 가능
- R과 마찬가지로 무료 소프트웨어임

1. Rstudio 사이트 접속

- 웹 브라우저를 통해 <https://www.rstudio.com> 연결 후 메인 화면에서 “Download RStudio” 클릭
- 혹은 상단 Pop-up 메뉴 중 Products → RStudio 클릭 후 연결된 화면에서 다운로드 진행

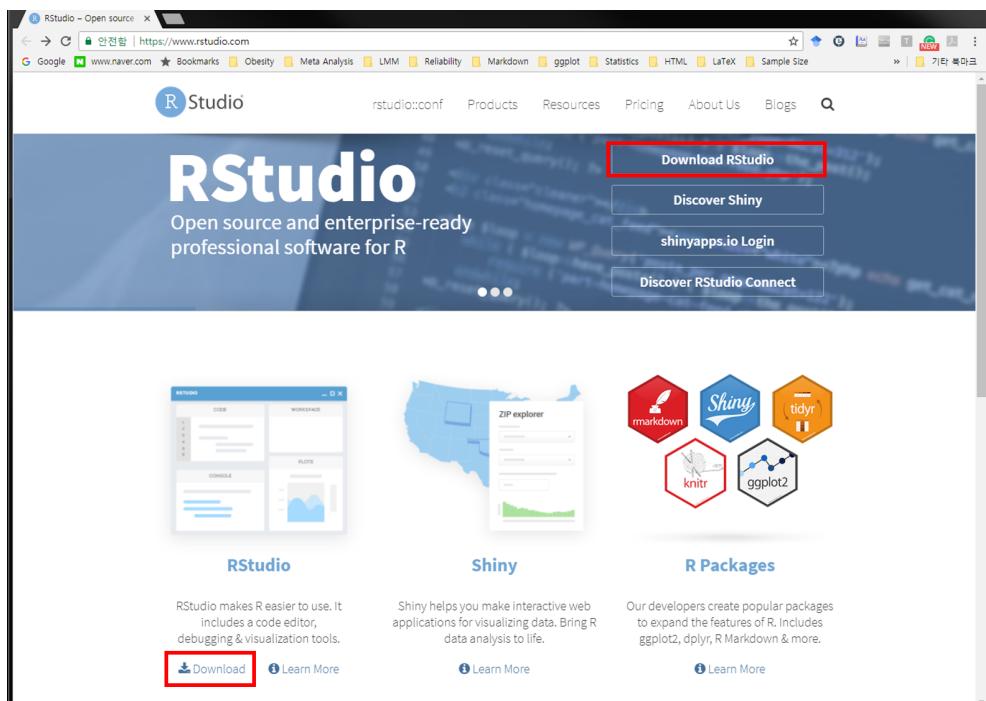


Figure 1.16: RStudio 메인 페이지 화면

2. Desktop 또는 Server 버전 중 택일

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

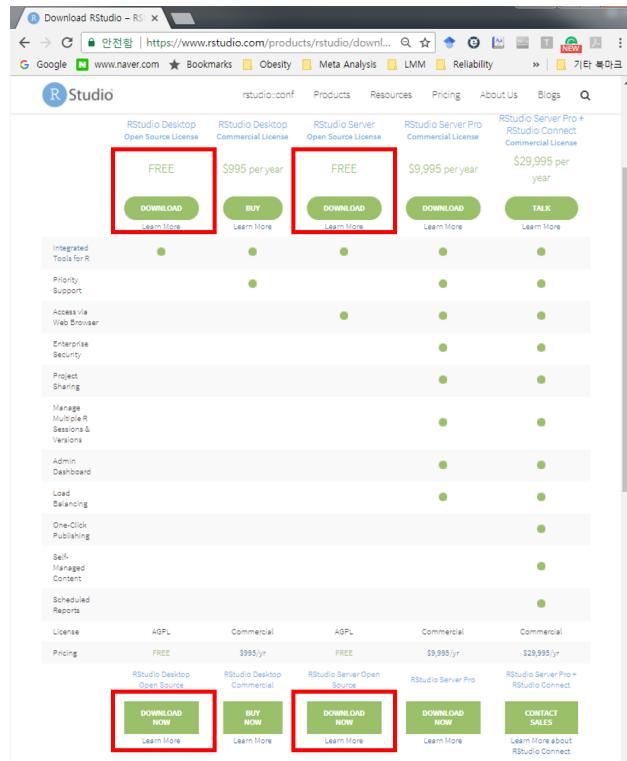


Figure 1.17: RStudio 다운로드 페이지

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

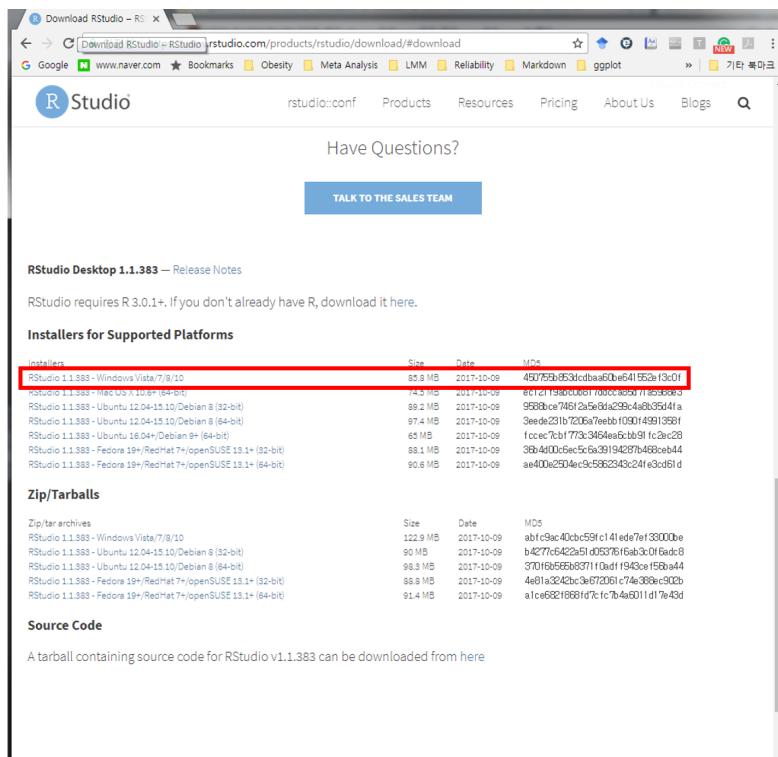


Figure 1.18: RStudio 운영체제 선택

4. RStudio installer 다운로드 시 파일이 저장된 폴더에서 보통 RStudio-xx.xx.xxx.exe 형식

의 파일명 확인

- 더블 클릭 후 실행
- 다음> 몇 번 누르면 설치 종료

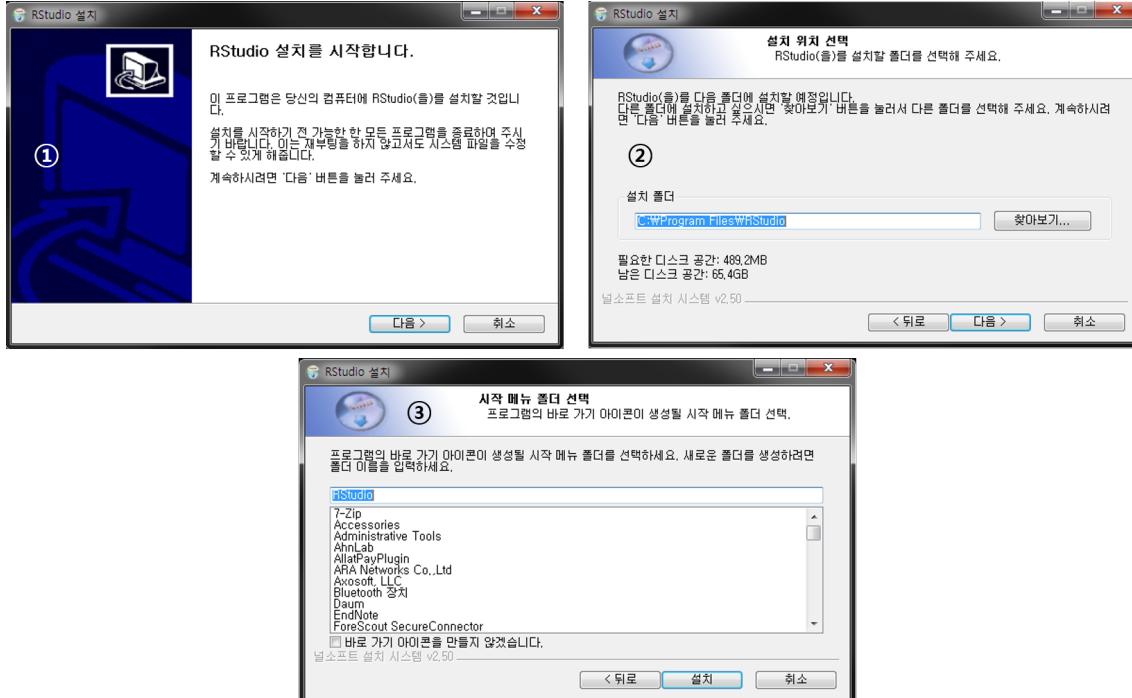


Figure 1.19: RStudio 설치화면

5. 아래와 같은 실행화면이 나타나면 RStudio 설치 성공

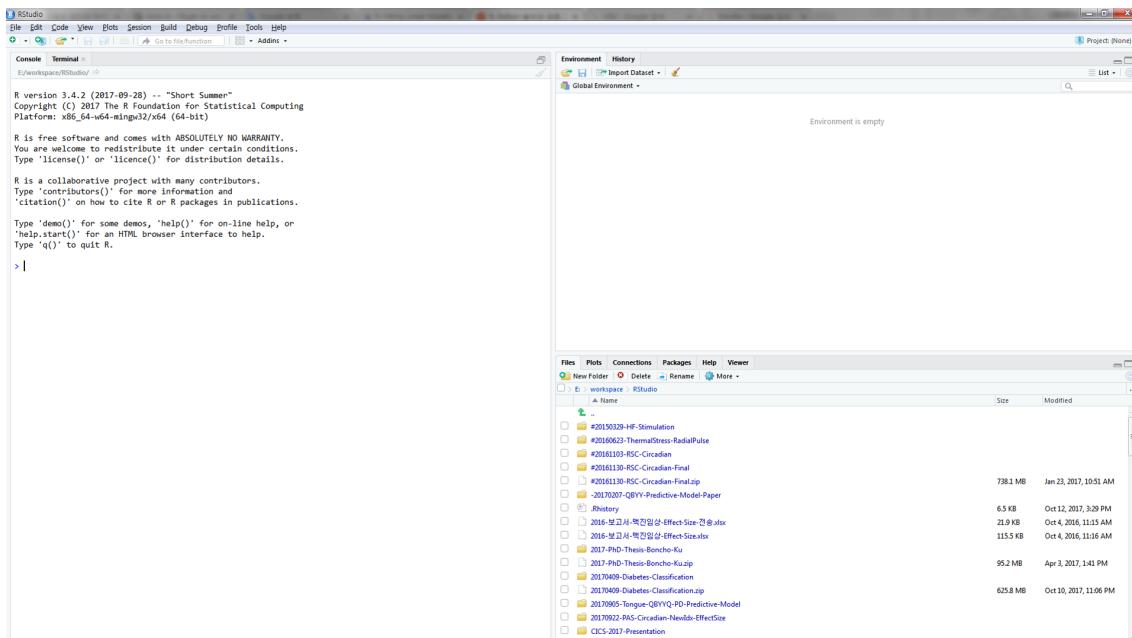


Figure 1.20: RStudio 초기 실행화면

1.5 RStudio의 구성

1.5.1 RStudio IDE 화면 구성

RStudio는 아래 그림 1.21과 같이 크게 4개 창으로 구성

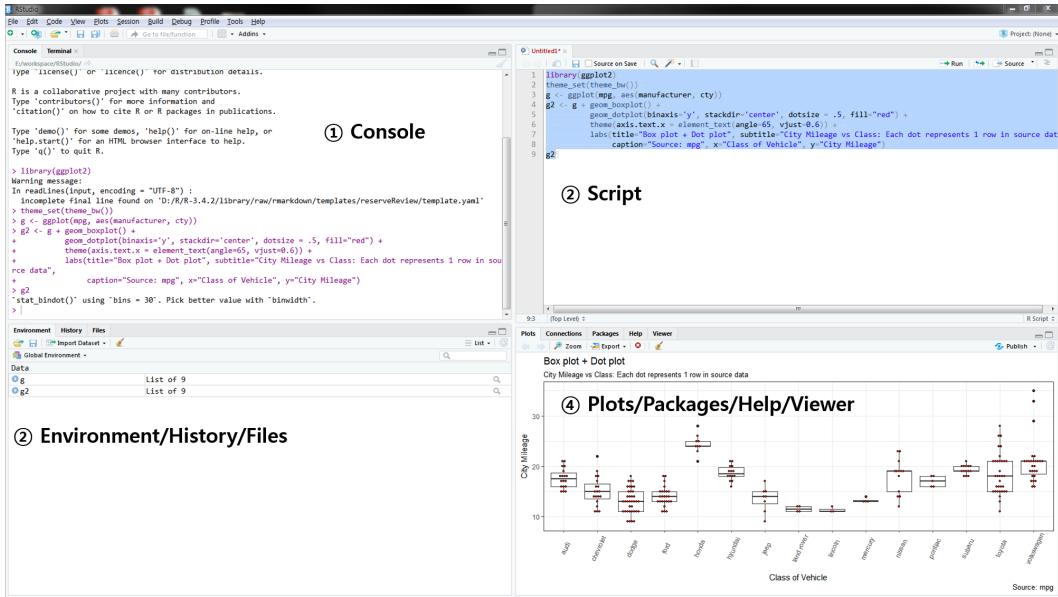


Figure 1.21: RStudio 화면구성: 우하단 그림은 <http://r-statistics.co/Top50-Ggplot2-Visualizations-MasterList-R-Code.html>에서 발췌

1. 콘솔(console)

- R 명령어 실행 공간(RGui, 정확하게는 Rterm.exe가 실행되고 있는 창)
- R 스크립트 또는 콘솔 창에서 작성한 명령어(프로그램) 실행 결과 출력
- 경고, 에러/로그 등의 메세지 확인

2. 스크립트(script) R 명령어 입력 공간으로 일괄처리(batch processing) 가능

- 일괄 처리를 위한 RStudio 제공 단축키
 - **Ctrl** + **Enter**: 선택한 블럭 내 명령어 실행
 - **Alt** + **Enter**: 선택 없이 커서가 위치한 라인의 명령어 실행
- 새 R 스크립트 파일 열기
 - R 상단 메뉴: [File] → [New File] → [R Script]
 - 단축키: **Ctrl** + **Shift** + **N**
- R 스크립트 이외 R Markdown, R Notebook, Shiny Web Application 등 새 문서의 목적에 따라 다양한 종류의 문서 생성 가능

3. Environment/History

- 1) Environment: 현재 R 작업환경에 저장되어 있는 객체의 특성을 요약 제시

- 좌측 화살표 버튼 클릭: 해당 객체의 상세 정보 확인(그림 1.22)
- 우측 사각형 버튼 클릭: 객체가 데이터프레임인 경우 스프레드 시트 형태로 데이터 셋 확인(그림 1.23)
- 디스크 아이콘: 작업공간 저장 여부
- 빗자루 아이콘: 작업공간 객체 일괄 삭제
- Global Environment 아이콘: 현재 작업공간에서 사용중인 패키지 및 선택 패키지 내장 함수 목록 확인

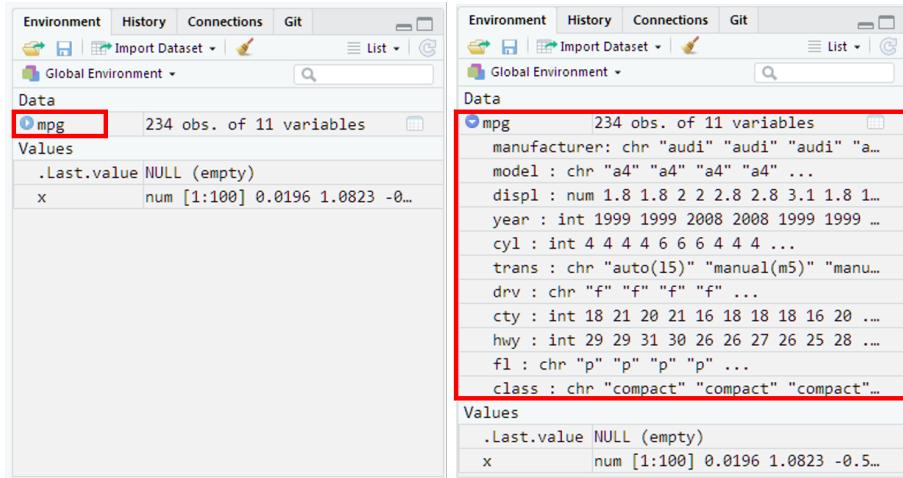


Figure 1.22: RStudio Environment 창: 객체 상세정보

	manufacturer	model	displ	year	cyl	trans	drv	cty	hwy	fl	class
1	audi	a4	1.8	1999	4	auto(l5)	f				compact
2	audi	a4	1.8	1999	4	manual(m5)	f				compact
3	audi	a4	2.0	2008	4	manual(m6)	f				compact
4	audi	a4	2.0	2008	4	auto(av)	f				compact
5	audi	a4	2.8	1999	6	auto(l5)	f				compact
6	audi	a4	2.8	1999	6	manual(m5)	f				compact
7	audi	a4	3.1	2008	6	auto(av)	f				compact
8	audi	a4 quattro	1.8	1999	4	manual(m5)	4				compact
9	audi	a4 quattro	1.8	1999	4	auto(l5)	4				compact
10	audi	a4 quattro	2.0	2008	4	manual(m6)	4				compact
11	audi	a4 quattro	2.0	2008	4	auto(s6)	4				compact
12	audi	a4 quattro	2.8	1999	6	auto(l5)	4				compact
13	audi	a4 quattro	2.8	1999	6	manual(m5)	4				compact
14	audi	a4 quattro	3.1	2008	6	auto(s6)	4				compact
15	audi	a4 quattro	3.1	2008	6	manual(m6)	4				compact

Figure 1.23: RStudio Environment 창: 스프레드 시트

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

```

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

```

Figure 1.24: RStudio History 창

4. File/Plots/Packages/Help/Viewer

1) File: Windows 탐색기와 유사

- 파일 및 폴더 생성, 삭제 수정, 그리고 작업경로 설정

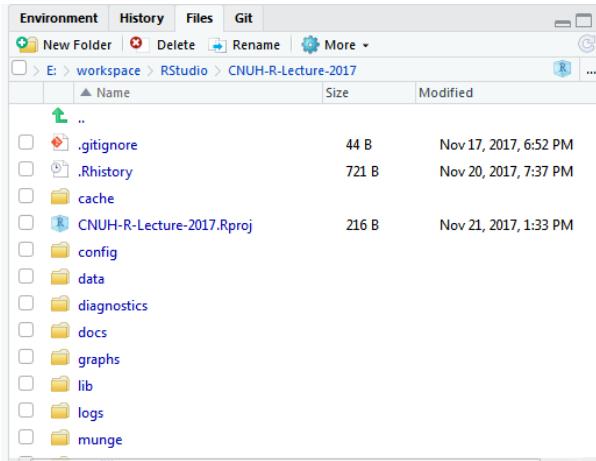


Figure 1.25: RStudio File 창

2) Plots: 콘솔 또는 스크립트 창으로부터 생성한 그래프 출력

- 작업 중 생성한 그래프가 이력에 따라 저장: ⌛ 이전, ⌛ 최근
- Zoom** 기능으로 그래프 확대 가능
- Export** 를 통해 선택 그래프를 이미지 파일(.png, .jpeg 등), PDF 파일 및 PDF 출력 파일로 저장 가능
- 엑스박스는 현재 화면에 출력된 그래프 삭제, 빗자루 아이콘은 생성한 모든 그래프 삭제

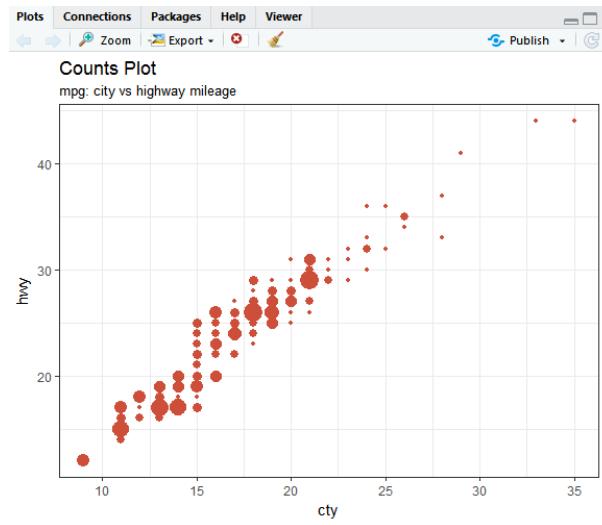


Figure 1.26: RStudio Plots 창 화면

3) Packages: 현재 컴퓨터에 설치된 R 패키지 목록 출력: 신규 설치 및 업데이트 가능

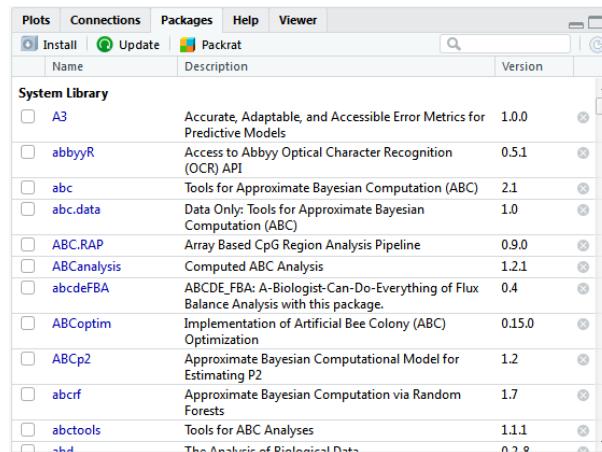


Figure 1.27: RStudio Packages 창 화면

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

```
> help(lm)
```

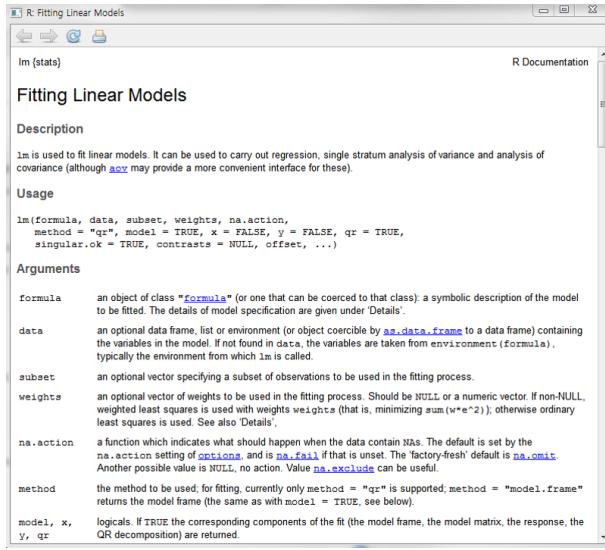


Figure 1.28: help(lm) 실행 후 RStudio Help 창 화면

5. RStudio의 창 layout은 [Tools] → [Global Options] → [Pane Layout]에서 변경 가능

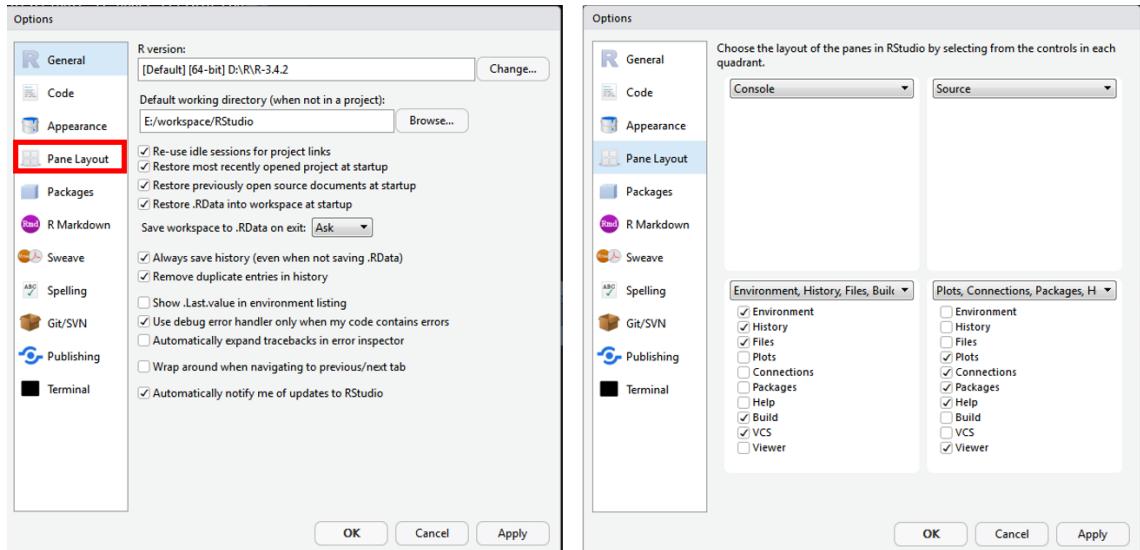


Figure 1.29: RStudio Option 선택화면 및 Pane 레이아웃 선택 화면

1.6 RStudio에서 배치 파일 생성 및 실행

스크립트 파일 생성, 저장, 실행

1. R 소스 파일 실행

- **[Ctrl]** + **[Shift]** + **S**
- **[Ctrl]** + **[Shift]** + **Enter**
- **source("파일이름")**

2. 파일 경로 설정

- R에서 디렉토리 구분자는 /임
- Windows에서 사용하는 \는 특수문자로 간주
- 경로 복사 후 R에서 사용하는 경우 \를 \\로 변경해야 인식

1.7 R 패키지 설치

1. RStudio 메뉴 [Tools] → [Install packages] 클릭 후 생성된 팝업 창에서 설치하고자 하는 패키지 입력 후 설치
2. RStudio Packages 창에서 버튼 누르고 설치(위와 동일)
3. R 콘솔 또는 스크립트 창에서 `install.packages()`

1.7.1 패키지 불러오기

1. `library()` vs. `require()`

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

2. 다중 패키지 동시에 불러오기

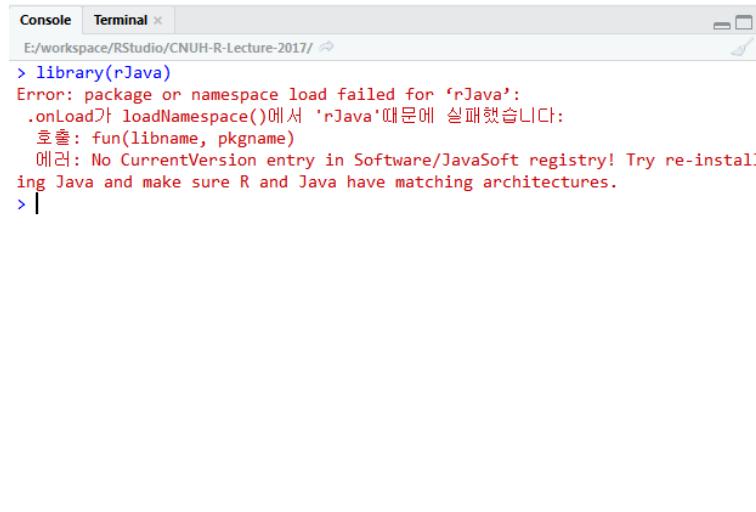
- RStudio Packages 창에서 설치하고자 하는 패키지 선택 버튼 클릭하면 R workspace로 해당 패키지 로드 가능
- 스크립트 이용

```
> pkgName <- c("MASS", "tidyverse", "ggthemes", "readxl", "ProjectTemplate", "kableExtra",
+      "ztable", "car", "lsmeans")
> # 'lapply()': 벡터, 리스트 또는 표현식에 함수를 적용하여 그 결과를 리스트로
> # 반환
> lapply(pkgName, require, character.only = T)
```

1.8 rJava 설치하기

1. 자바 명령을 호출하는 패키지를 정상적으로 사용하기 위한 패키지

- Java 계열 머신러닝 어플리케이션 활용 또는 최근까지 Excel 파일을 불러오기 위해 설치 필수(Excel에 관해서는 Hadley Wickham의 `readxl` 개발 이전에는...)
- 다음은 Java가 설치되지 않은 환경에서 `rJava`를 불렀을 때 발생한 오류 예시임



The screenshot shows the RStudio interface with the 'Console' tab selected. The console window displays the following error message:

```
Console Terminal ×
E:/workspace/RStudio/CNUH-R-Lecture-2017/
> library(rJava)
Error: package or namespace load failed for 'rJava':
.onLoad가 loadNamespace()에서 'rJava' 패키지에 실패했습니다:
호출: fun(libname, pkgname)
에러: No CurrentVersion entry in Software/JavaSoft registry! Try re-installing Java and make sure R and Java have matching architectures.
> |
```

Figure 1.30: rJava 에러 메세지

2. `rJava` 설치 전 자바 설치 및 Windows 시스템 환경 설정 필요

3. 자바 개발 키트(Java Development Kit, JDK)는 <http://www.oracle.com/technetwork/java/javase/downloads/index.html>에서 다운로드 가능

- [Java Platform (JDK)] → [Download] 클릭

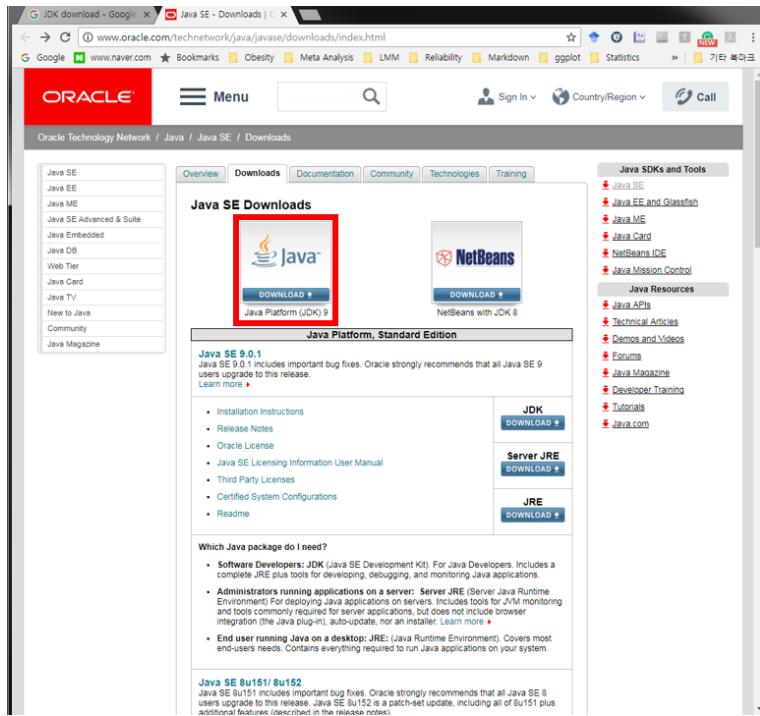


Figure 1.31: JDK 다운로드 페이지

4. JDK 다운로드 전 현재 주로 작업하고 있는 R 버전 정보(32 bit vs. 64 bit) 확인

- R 세션 시작 시 또는 RStudio에서 [Tools] → [Global Options]에서 R 버전 확인 가능
- 라이센스 계약에 동의 후 확인한 R 버전에 대응하는 링크 클릭 후 다운로드 진행

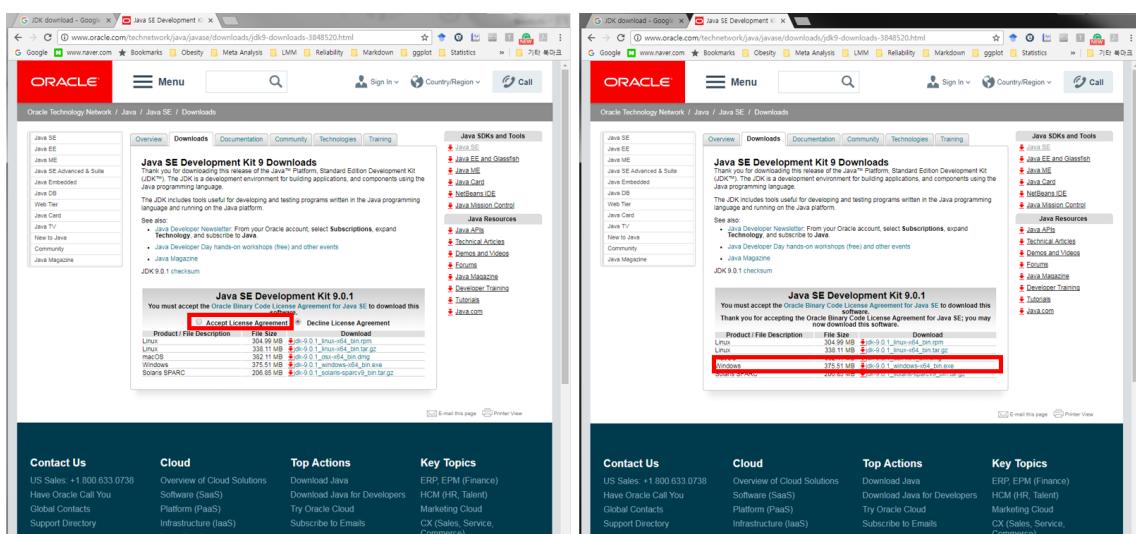


Figure 1.32: JDK 라이센스 및 다운로드 링크 페이지

5. 다운로드 완료 후 설치 확인

- 정상적으로 설치한 경우라면 C:\Program Files\Java\jdk-9.0.1 디렉토리에서 설치

확인 가능

- Java runtime environment (JRE)는 C:\Program Files\Java\jre-9.0.1에서 설치
확인 가능

6. 새 사용자 변수 생성

- Windows 시작 클릭 → [컴퓨터] 우 클릭 → [속성] 클릭 후 다음 그림 1.33에서 [고급 시스템 설정] 클릭

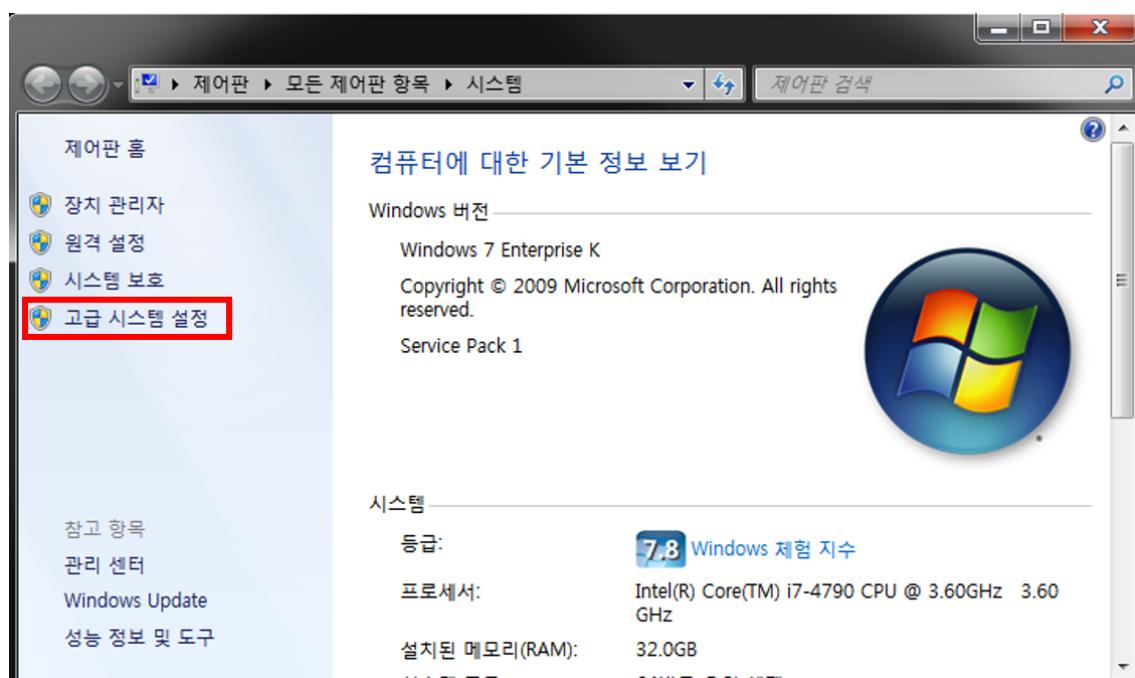


Figure 1.33: Windows 시스템 설정 화면

- 시스템 속성 창에서 [환경변수(N)...] 클릭 후 환경변수 창에서 사용자 변수 [새로 만들기(N)...] 클릭
- 변수 이름에 JAVA_HOME, 변수 값에 JDK 설치 디렉토리 이름(예: C:\Program Files\Java\jdk-9.0.1) 입력

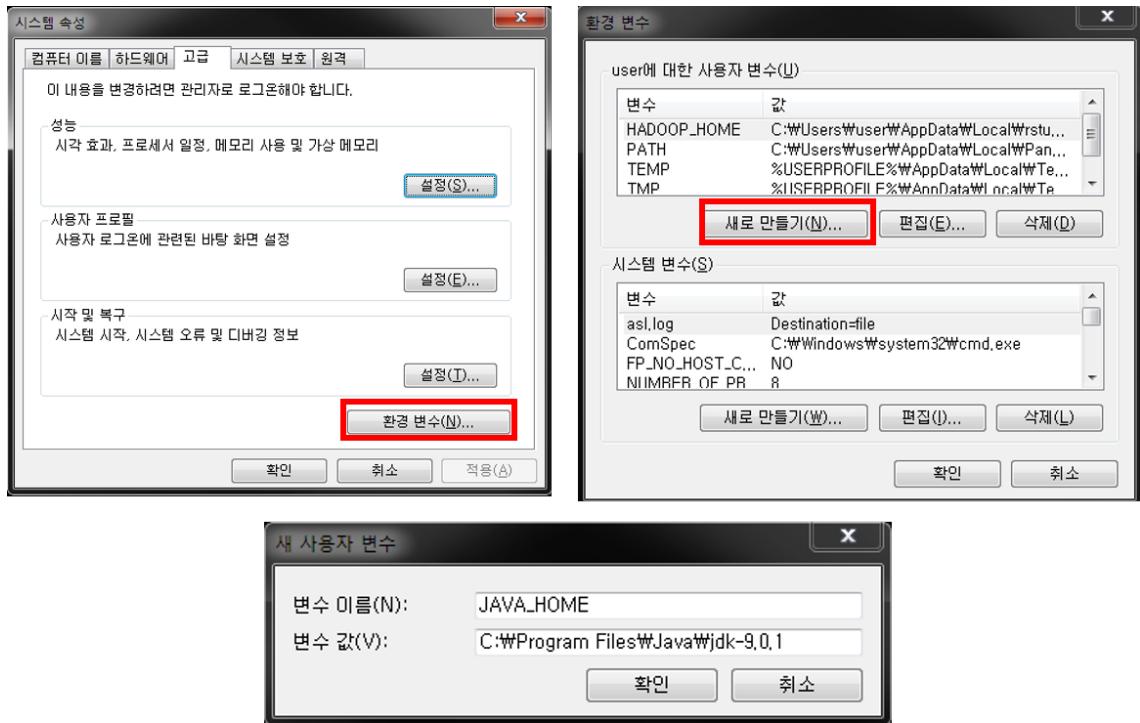


Figure 1.34: Windows 시스템 설정 화면

7. 시스템 변수 PATH 생성

- JRE가 설치된 디렉토리(아래 나열)를 기준 PATH에 추가(디렉토리 구분자: ;)
 - C:\Program Files\Java\jre-9.0.1\bin
 - C:\Program Files\Java\jre-9.0.1\bin\server

8. 설정 완료 후 R 또는 RStudio 재실행 후 rJava 설치

9. 별 다른 에러 메세지 출력이 없으면 성공적으로 설치

1.9 RStudio 프로젝트 생성 및 ProjectTemplate 패키지 연동

1.9.1 RStudio 프로젝트

1. 프로젝트

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

2. 프로젝트의 필요성

- 자료의 정합성 보장

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

3. 좋은 프로젝트 구성을 위한 방법

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

4. RStudio 새로운 프로젝트 생성

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

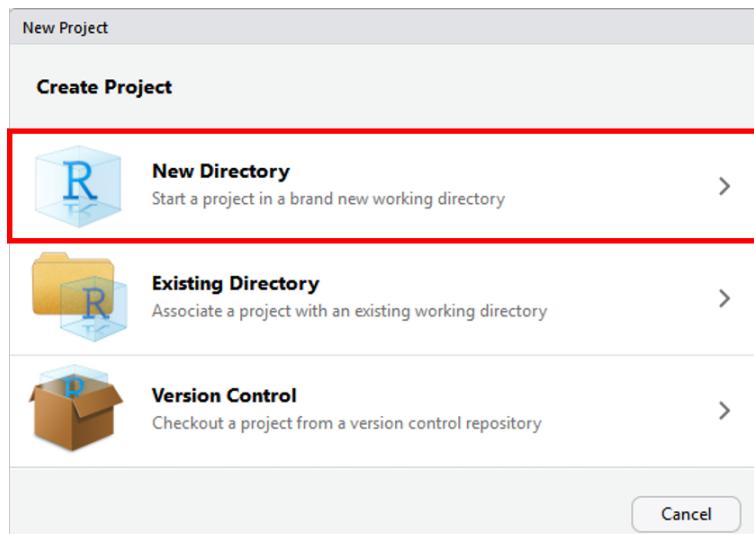


Figure 1.35: RStudio 새로운 프로젝트 생성

- **Existing Directory**: 이미 작업공간이 존재하고 있는 경우 해당 디렉토리 선택(다루지 않음)
- **Version Control**: 버전관리 시스템(Git, SVN)의 저장소에 존재하는 프로젝트 작업 선택(다루지 않음)

2) 그림 1.35에서 **New Directory** 선택 후 다음 창으로 이동

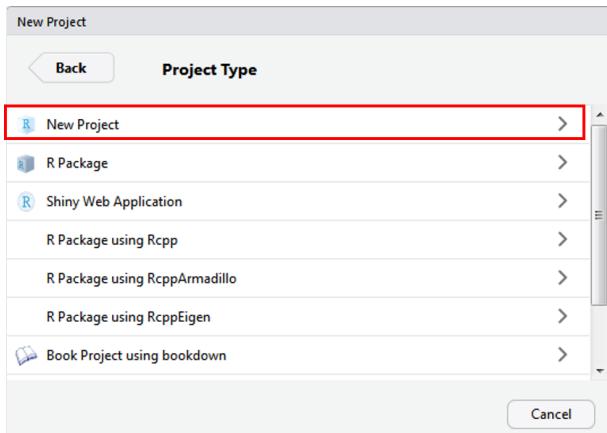


Figure 1.36: RStudio 새로운 프로젝트 유형 선택

- **New Project** : 자료분석을 위한 프로젝트 생성
- **R Package** : 패키지 개발을 위한 작업 프로젝트 생성
- **Shiny Web Application** : Shiny를 이용한 데이터 연동형 웹 어플리케이션 개발 프로젝트 생성
- **Book Project using bookdown** : 책 지필 작업을 위한 프로젝트 생성

3) **New Project** 선택 후 프로젝트 저장 폴더 생성 창으로 이동

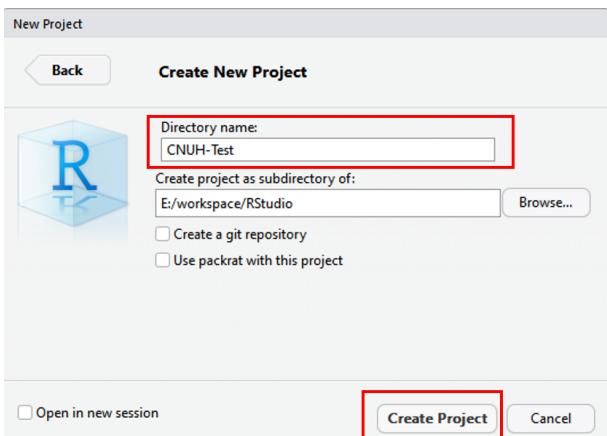


Figure 1.37: 새로운 프로젝트의 폴더명 지정

- 여기서는 CNUH-Test라는 프로젝트 이름으로 폴더 생성
- 아래 [Create projects as subdirectories of]에서 생성하고자 하는 프로젝트의 상위 디렉토리 설정 → 보통 RStudio의 default working directory로 기본 설정되어 있음.
- 입력 완료 후 **Create Project** 클릭 후 새로운 R 세션 화면이 열리는 것을 확인 했으면 새로운 프로젝트 생성 완료

4) 생성한 프로젝트 확인

- 새로운 R 세션 콘솔 창에 `getwd()`로 생성한 프로젝트 폴더 확인
- 아래 그림처럼 생성 프로젝트 폴더에 `CNUH-Test.RProj` 실행파일 확인

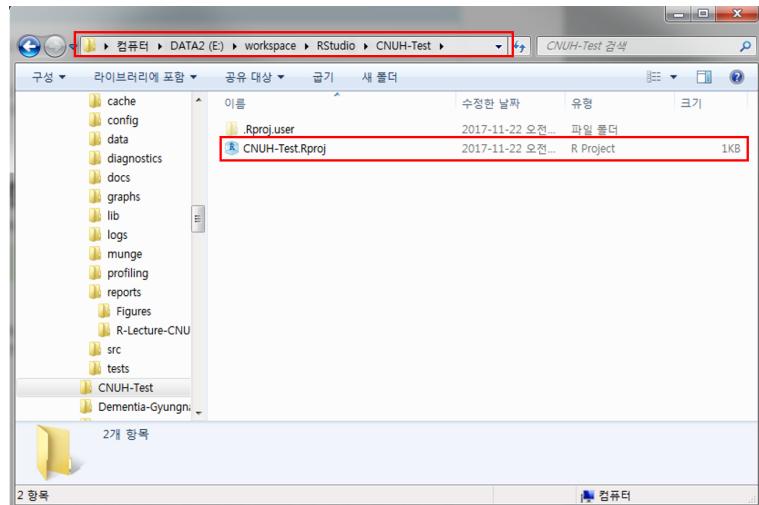


Figure 1.38: 프로젝트 폴더 내 실행 파일 생성 확인

1.9.2 ProjectTemplate

1. 프로젝트 관리를 자동화 하기 위한 솔루션을 제공하는 R 패키지
 - 프로젝트 관리에 이상적인 directory 구조 제공
 - 자동으로 분석 파이프라인 및 작업흐름을 구성해서 구조화 함
 - RStudio Project + ProjectTemplate + Git: 작업 기록 및 공동 작업 가능
 - ProjectTemplate의 자세한 사항은 다음 링크 <http://projecttemplate.net/index.html> 참조

2. ProjectTemplate 설치 및 생성

- 1) `install.packages()` 함수를 이용해 설치

```
> install.packages("ProjectTemplate")
```

- 2) 패키지 불러오기

```
> library(ProjectTemplate)
```

- 3) ProjectTemplate 작업환경 생성

```

> # '..'는 default working directory 지시변수
> create.project("../CNUH-Test", merge.strategy = "allow.non.conflict")

```

4) 프로젝트 폴더 내 아래 폴더 생성 확인

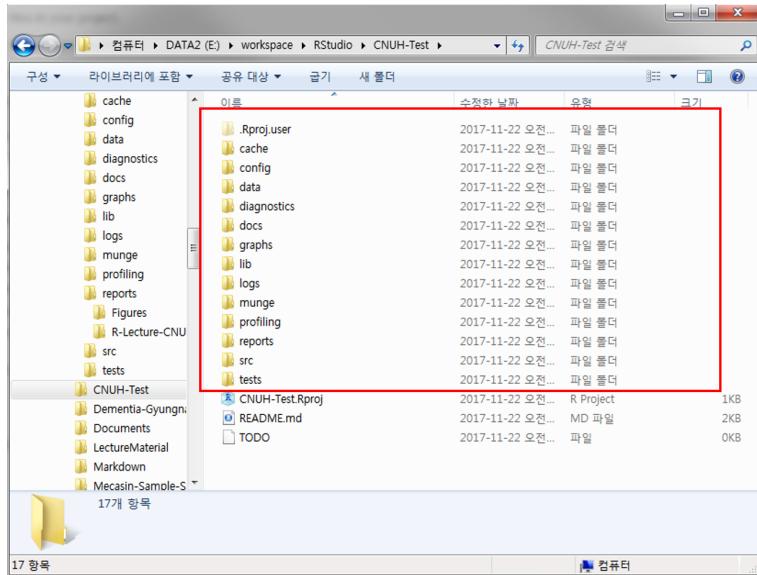


Figure 1.39: `create.project()` 후 프로젝트 폴더 내 생성 디렉토리

- `cache/`: 자료 전처리 과정 중 생성한 cache 처리 데이터 저장
- `config/`: 프로젝트 설정(`global.gcf`에 저장) 관리
- `data/`: 원자료 또는 읽기 전용 분석 자료 저장
- `diagnostics/`: 자료의 오류 및 이상치 탐지를 위한 스크립트 저장
- `doc/`: 프로젝트 산출물을 위한 모든 종류의 문서 저장
- `graphs/`: 스크립트로 출력된 그래프 저장
- `libs/`: 재사용 혹은 자체 제작한 R 함수 스크립트 저장
- `munge/`: 자료 전처리를 위한 스크립트 저장
- `profiling/`: 코드 실행 시간 체크를 위해 작성한 스크립트 저장
- `reports/`: 프로젝트 보고 문서(HTML, PDF 등) 저장
- `src/`: 자료분석을 위해 사용한 스크립트 저장
- `tests/`: 함수 작성을 위해 생성한 test 스크립트 저장

3. ProjectTemplate 관리

- 1) `ProjectTemplate`은 `config/global.dcf` 파일로 프로젝트 불러올 때 옵션 설정 가능

```
version: 0.8 # 현재 `ProjectTemplate` 버전 정보  
data_loading: TRUE # 'data/' 폴더 내 자료읽기 여부  
data_loading_header: TRUE # 'data/' 폴더 내 자료읽기 여부  
data_ignore: # 'data/' 폴더 내 읽기무시 파일 목록  
cache_loading: TRUE # 'cache/' 자료읽기 여부  
recursive_loading: FALSE  
munging: TRUE # 'munging/' 자료전처리 스크립트 읽기여부  
logging: FALSE # Log 정보 기록여부  
logging_level: INFO  
load_libraries: FALSE # 아래 패키지 리스트 불러오기 여부  
libraries: reshape, plyr, dplyr, ggplot2, stringr, lubridate  
as_factors: TRUE # 문자형 변수 요인화 여부  
data_tables: FALSE # data.frame()// data.table 속성 추가  
attach_internal_libraries: FALSE # 'lib\''에 저장된 자체 생성 함수 읽기 여부  
cache_loaded_data: TRUE  
sticky_variables: NONE
```

실습

새로 만든 CNUH-Test 프로젝트에서 데이터 불러오기

Chapter 2

R의 기본 사용

목적: R 활용을 위해 기초적으로 알아야 할 객체 속성 및 기본 연산 논리 학습

- R은 객체지향언어(object-oriented language)
- 객체: 숫자, 데이터셋, 단어, 테이블, 분석결과 등 모든 것을 칭함
 - “객체지향”의 의미는 R의 모든 명령어는 객체를 대상으로 이루어진다는 것을 의미
- R에서 사용자가 데이터 입력을 위해 생성 또는 읽어온 객체(object)는 종종 변수(variable)라는 말과 혼용됨.
- 본 문서에서는 최상위 데이터 저장장소를 객체라고 명명하며 데이터프레임과 같이 여러 종류의 데이터타입으로 이루어진 객체의 1차원 속성을 변수라고 칭함.
- R을 다룰 수 있으려면 기본적인 R의 데이터 할당 방식과 데이터 타입, 그리고 연산 방법에 대한 이해가 있어야 함.
- 특히 데이터 타입에 따라 사용하는 함수 또는 분석 방법이 조금씩 다른 것이 R의 특징임
- 따라서 R의 데이터 타입에 좀 더 익숙해져야 R을 보다 쉽게 다룰 수 있음

2.1 R의 데이터 유형

2.1.1 R 객체 입력 방법 및 변수 설정 규칙

2.1.1.1 객체 입력 및 할당

R의 가장 기본적인 데이터 입력 및 할당 방법과 객체 또는 변수의 명명 규칙 설명

1. 객체를 할당하는 두 가지 방법:=, <-

1) 두 할당 지시자의 차이점

- :=: 명령의 최상 수준에 사용 가능
- <=: 어디서든 사용 가능
- 함수 호출과 동시에 변수에 값을 할당할 목적으로는 <-만 사용 가능

```
> # 이전 히스토그램 생성 위해 생성한 x 삭제
```

```
> rm(x)
```

```
> # `mean()` 입력 벡터의 평균 계산
```

```
> mean(y <- c(1, 2, 3, 4, 5))
```

```
[1] 3
```

```
> y
```

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

```
> mean(x = c(1, 2, 3, 4, 5))
```

```
[1] 3
```

```
> x
```

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

2. 객체 또는 변수의 명명 규칙

- 1) 알파벳, 한글, 숫자, _, .의 조합으로 구성 가능(-은 사용 불가)
- 2) 변수명의 알파벳, 한글, .로 시작 가능
 - .로 시작한 경우 뒤에 숫자 올 수 없음(숫자로 인지)
- 3) 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  
> a1 <- seq(from = 1, to = 10, by = 2)  
> # 한글 변수명  
> 가수 <- c("Damian Rice", "Beatles", "최백호", "Queen")  
> 가수
```

```
[1] "Damian Rice" "Beatles"      "최백호"      "Queen"
```

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: _
```

```
^
```

```
> 0.3 <- 10
```

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

2.2 R 객체 유형

1. 객체의 종류

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

2. 객체에 입력 가능한 값

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

2.2.1 스칼라

1. 단일 차원의 값(하나의 값): 1×1 벡터로 간주 가능

- R 자료형의 기본은 벡터

2. 자료형은 크게 숫자형, 문자열, 논리형이 있음

참고: 스칼라를 입력시 R의 벡터 지정 함수인 c()를 꼭 사용해서 입력할 필요가 없다.

단, 두 개 이상 스칼라면 벡터이므로 꼭 c()를 써야 한다.

2.2.1.1 숫자형: 수치연산(+, -, *, ^, **, /, %%, %%) 가능

```
> # 숫자형 스칼라  
> a <- 3
```

```
> b <- 10
```

```
> a
```

```
[1] 3
```

```
> b
```

```
[1] 10
```

```
> # 덧셈
```

```
> c <- a + b
```

```
> c
```

```
[1] 13
```

```
> # 뺄셈
```

```
> d <- b - a
```

```
> d
```

```
[1] 7
```

```
> # 곱셈
```

```
> a * b
```

```
[1] 30
```

```
> # 역승
```

```
> b^a
```

```
[1] 1000
```

```
> b^a
```

```
[1] 1000
```

```
> # 나누기
```

```
> b/a
```

```
[1] 3.33
```

```
> # 나누기 나머지 반환
```

```
> b%%a
```

```
[1] 1
```

```
> # 나누기 뒷 반환  
> b%/%a
```

```
[1] 3
```

```
> # 연산 우선순위  
> n <- (3 + 5) * 3 - 4^2/2  
> n
```

```
[1] 16
```

2.2.1.2 문자형 스칼라: 수치연산 불가능

```
> h1 <- c("Hello CNU Hospital CTS.")  
> g2 <- c("R is not too difficult.")  
> h1
```

```
[1] "Hello CNU Hospital CTS."
```

```
> g2
```

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

```
> h1 + g2
```

Error in h1 + g2: 이항연산자에 수치가 아닌 인수입니다

2.2.1.3 논리형 스칼라

- TRUE, T: 참
- FALSE, F: 거짓
- &, && (AND); |, || (OR); ! (NOT) 연산자 사용 가능
- 비교연산: 숫자값들의 비교를 통해 논리값 반환
 - >, <, >=, <=, ==, !=
 - 문자형 비교 연산은 ==, !=만 가능

참고 1: 논리형 스칼라도 숫자형 연산 가능하다. 왜냐하면 컴퓨터는 TRUE/FALSE를 1과 0의 숫자로 인식하기 때문이다.

참고 2: &와 &&는 동일하게 AND를 의미하지만 연산 결과가 다르다. &의 연산 대상이 벡터인 경우 백터 구성 값 각각에 대해 & 연산을 실행 하지만 &&는 한 개(스칼라)에만 논리 연산이 적용된다(아래 예시 참고).

```
> TRUE & TRUE
```

```
[1] TRUE
```

```
> TRUE & FALSE
```

```
[1] FALSE
```

```
> TRUE | TRUE
```

```
[1] TRUE
```

```
> TRUE | FALSE
```

```
[1] TRUE
```

```
> !TRUE
```

```
[1] FALSE
```

```
> !FALSE
```

```
[1] TRUE
```

```
> # T/F는 각각 TRUE/FALSE의 전역변수
```

```
> T <- FALSE
```

```
> T
```

```
[1] FALSE
```

```
> TRUE <- FALSE
```

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

```
> T <- TRUE
```

```
> # &와 &&의 차이
```

```
> l1 <- c(TRUE, TRUE, FALSE, TRUE)
```

```
> 12 <- c(FALSE, TRUE, TRUE, TRUE)
> 11 & 12
```

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

```
> 11 && 12
```

```
[1] FALSE
```

```
> x <- 10
> y <- 13
> x > y
```

```
[1] FALSE
```

```
> x < y
```

```
[1] TRUE
```

```
> x >= y
```

```
[1] FALSE
```

```
> x <= y
```

```
[1] TRUE
```

```
> x == y
```

```
[1] FALSE
```

```
> x != y
```

```
[1] TRUE
```

```
> y <- 10
```

```
> x != y
```

```
[1] FALSE
```

2.2.1.4 결측값(NA)

- 데이터 값이 없음을 의미

- 예를 들어 4명의 학생 중 3명의 수학 시험 점수가 80, 95, 90 점인데 한 학생의 점수를 모르는 경우 NA를 이용해 점수 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(NA)
```

[1] TRUE

2.2.1.5 NULL 값(NULL)

- 초기화 되지 않은 값 표현
- `is.null()` 함수를 통해 NULL 지정 여부 확인

참고: NA와 NULL은 자료의 공백을 의미한다는 점에서 유사한 측면이 있으나 아래 측면에서 큰 차이가 있음!!

- NULL: 값을 지정하지 않은 객체(object) 표현
- NA: 자료값이 결측임을 지정해주는 논리형 상수

```
> # NA와 NULL의 차이점 'is.null()' 객체 또는 변수에 NULL이 저장되어 있는지
> # 판단 'class()' 객체의 유형을 반환
> x <- NULL
> is.null(x)
```

[1] TRUE

```
> is.null(1)
```

[1] FALSE

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

```
[1] FALSE
```

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

```
logical(0)
```

```
> class(NA)
```

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

```
> class(NULL)
```

```
[1] "NULL"
```

2.2.1.6 요인형

- 범주형 자료를 표현하기 위한 값
- 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
- factor() 함수를 통해 표현 가능
 - 프로토타입: factor(x, levels, labels, ordered) (help(factor) 참조)
 - * x 벡터(스칼라)에 대응하는 요인의 수준(levels)과 수준의 레이블(labels)을 지정.
만약 요인 x가 순서형이면 ordered인지 아닌지를 입력
 - 숫자형에 적용되는 연산 불가
 - 요인형을 조금 더 부연설명 하면 숫자 또는 문자 값에 수준의 속성을 부여한 값의 형태임

예시 사용 함수

- ‘is.factor()’: 벡터(스칼라)가 요인형인지 확인
- ‘nlevels()’: factor의 level 개수 반환
- ‘levels()’: level 목록 반환
- ‘labels()’: level에 대응하는 label 목록 반환
- ‘ordered()’: 순서형 factor 생성
- ‘is.ordered()’: 순서형 factor 확인

```
> # factor() 기본 사용법 factor(x, levels, labels, ordered = F) x: factor  
> # 형으로 지정하고자 하는 스칼라 또는 벡터 levels: factor의 수준 지정 labels:  
> # factor의 레이블 지정  
> sex <- factor("M", levels = c("M", "F"))  
> sex
```

[1] M

Levels: M F

```
> # factor 형 판단 TRUE/FALSE  
> is.factor(sex)
```

[1] TRUE

```
> # factor의 level 개수 반환  
> nlevels(sex)
```

[1] 2

```
> # factor의 level 목록 반환  
> levels(sex)
```

[1] "M" "F"

```
> # factor의 level 목록 수정  
> levels(sex) <- c("Male", "Female")  
> sex
```

[1] Male

Levels: Male Female

```
> # 순서형 factor  
> severity <- factor(1, levels = c(1, 2, 3), labels = c("Mild", "Moderate", "Severe"))  
> # severity <- ordered(severity) 또는 factor() 함수 내에서 ordered 옵션을  
> # TRUE로 지정  
> severity <- factor(1, levels = c(1, 2, 3), labels = c("Mild", "Moderate", "Severe"),  
+     ordered = TRUE)  
> severity
```

```
[1] Mild
```

```
Levels: Mild < Moderate < Severe
```

```
> is.ordered(severity)
```

```
[1] TRUE
```

2.2.2 벡터

- 동일한 유형의 자료(스칼라)가 2개 이상 1차원 ($n \times 1, n \geq 2$) 으로 구성되어 있는 자료 형태
- 벡터는 앞의 예시에서 본 바와 같이 `c()` 함수를 사용해 생성
- 서로 다른 자료형으로 벡터를 구성한 경우 표현력이 둑은 자료형으로 변환한 값 반환
 - 예: 문자열 + 숫자로 구성된 벡터 → 문자형 벡터
- 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.
 - 벡터의 길이(차원) 확인
 - * `length()` 또는 `NROW()` 사용
- 색인(indexing)을 통해 벡터의 원소에 접근 가능
 - `x[i]`: 벡터 x의 i번째 요소
 - `x[-i]`: 벡터 x에서 i번째 요소를 제외한 나머지
 - `x[I]`: I가 인덱싱 벡터라고 할 때 I에 지정된 요소를 얻어옴. 일반적으로 I는 벡터의 행 순서 번호 또는 각 벡터 원소의 이름에 대응하는 문자열 벡터를 인덱싱 벡터로 사용할 수 있음.
 - `x[start:end]`: x의 start부터 end까지 값 반환
- 벡터의 원소가 NULL을 포함한 경우 해당 원소가 완전히 삭제(예시 참조)

```
# 숫자형 벡터  
x <- c(1:25)  
y <- c(26:50)  
is.numeric(x)
```

```
[1] TRUE
```

```
# 벡터의 연산 사칙연산
```

```
x + y
```

```
[1] 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59 61 63 65 67 69 71  
[24] 73 75
```

```
x - y
```

```
[1] -25 -25 -25 -25 -25 -25 -25 -25 -25 -25 -25 -25 -25 -25 -25 -25 -25 -25 -25  
[18] -25 -25 -25 -25 -25 -25 -25 -25 -25
```

```
x/y
```

```
[1] 0.0385 0.0741 0.1071 0.1379 0.1667 0.1935 0.2188 0.2424 0.2647 0.2857  
[11] 0.3056 0.3243 0.3421 0.3590 0.3750 0.3902 0.4048 0.4186 0.4318 0.4444  
[21] 0.4565 0.4681 0.4792 0.4898 0.5000
```

```
x * y
```

```
[1] 26 54 84 116 150 186 224 264 306 350 396 444 494 546  
[15] 600 656 714 774 836 900 966 1034 1104 1176 1250
```

```
## 비교 연산
```

```
a <- c(1, 2, 3, 3, 4, 2, 5, 1, 2, 3, 4, 4, 4)  
b <- c(2, 2, 4, 4, 3, 2, 5, 1, 2, 4, 4, 4, 3)  
a > b
```

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

```
a >= b
```

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

```
a < b
```

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

```
a <= b
```

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

```
a == b
```

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

```
a != b
```

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

```
# 벡터의 길이: `length()` 또는 `NROW` 함수 이용
```

```
z <- 1
```

```
length(z)
```

```
[1] 1
```

```
NROW(x)
```

```
[1] 25
```

```
# 문자형 벡터
```

```
str1 <- c("Boncho Ku", "Linear Regression", "307", "Male")
```

```
str1
```

```
[1] "Boncho Ku"           "Linear Regression" "307"
```

```
[4] "Male"
```

```
# 벡터 원소의 이름 지정
```

```
names(str1) <- c("Name", "Subject", "Room", "Sex")
```

```
str1
```

Name	Subject	Room
"Boncho Ku"	"Linear Regression"	"307"
	Sex	
	"Male"	

```
is.numeric(str1)
```

```
[1] FALSE
```

```
is.character(str1)
```

```
[1] TRUE
```

```
# 숫자와 문자를 동시에 포함한 벡터라면?
```

```
vec1 <- c(1, 2, 5, "1", "Clinical Trial")
```

```
### 모두 문자열으로 변환된 벡터
```

```
# 논리형 벡터
```

```
boolean <- c(TRUE, TRUE, FALSE, TRUE, TRUE, TRUE, FALSE, FALSE)
```

```
is.numeric(boolean)
```

```
[1] FALSE
```

```
is.logical(boolean)
```

```
[1] TRUE
```

```
# 결측을 포함한 벡터
```

```
vecNA <- c(1:10, NA, NA)
```

```
is.na(vecNA)
```

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

```
[12] TRUE
```

```
is.numeric(vecNA)
```

```
[1] TRUE
```

```
# NULL 포함 벡터
```

```
vecNULL <- c(NULL, 1, 2, NULL, 3)
```

```
vecNULL
```

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

```
# seq_along(), rep() 함수 사용
```

```
seq_along(x)
```

```
[1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23  
[24] 24 25
```

```
z <- c(3, 8)  
# `rep(x, each)` 사용 예  
rep(z, each = 3)
```

```
[1] 3 3 3 8 8 8
```

```
# `rep(x, times)` 사용 예  
rep(z, times = 3)
```

```
[1] 3 8 3 8 3 8
```

```
# `rep(x, each, times)` 사용 예  
rep(z, each = 2, times = 3)
```

```
[1] 3 3 8 8 3 3 8 8 3 3 8 8
```

```
# 요인 벡터 rep() 함수를 이용한 0,1 벡터 생성  
sex <- rep(c(0, 1), each = 10)  
sex
```

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

```
sex <- factor(sex, levels = c(0, 1), labels = c("Male", "Female"))  
sex
```

```
[1] Male   Male   Male   Male   Male   Male   Male   Male   Male  
[11] Female Female Female Female Female Female Female Female Female  
Levels: Male Female
```

```
str(sex)
```

```
Factor w/ 2 levels "Male", "Female": 1 1 1 1 1 1 1 1 1 ...
```

```
# 벡터의 색인  
x[10]
```

```
[1] 10
```

```
x[-3]
```

```
[1] 1 2 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24  
[24] 25
```

```
x[1:5]
```

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

```
x[c(1:3, 10:12)]
```

```
[1] 1 2 3 10 11 12
```

```
x[-c(1:10)]
```

```
[1] 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
```

```
str1["Name"]
```

```
Name
```

```
"Boncho Ku"
```

2.3 행렬

동일한 유형의 2차원 데이터 구조 ($n \times p$ 자료행렬) - $n \times 1$ 차원 벡터 p 개로 묶여진 데이터 봉어리

```
> # 1~12까지의 숫자에 대한 행(row)의 개수가 4개인 행렬 생성  
> m1 <- matrix(1:12, nrow = 4)  
> m1
```

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

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

```
> # 1~12까지의 숫자로 행 기준 행 개수가 4개인 행렬 생성  
> m2 <- matrix(1:12, nrow = 4, byrow = TRUE)  
> m2
```

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

```
> # 행렬 연산
```

```
> m1 + m2
```

```
[,1] [,2] [,3]  
[1,]    2    7   12  
[2,]    6   11   16  
[3,]   10   15   20  
[4,]   14   19   24
```

```
> m1 - m2
```

```
[,1] [,2] [,3]  
[1,]    0    3    6  
[2,]   -2    1    4  
[3,]   -4   -1    2  
[4,]   -6   -3    0
```

```
> # 곱과 나누기 연산이 각 원소에 대응해서 이루어진 것에 주의
```

```
> m1/m2
```

```
[,1] [,2] [,3]  
[1,] 1.000 2.500 3.00  
[2,] 0.500 1.200 1.67  
[3,] 0.429 0.875 1.22  
[4,] 0.400 0.727 1.00
```

```
> m1 * m2
```

```
[,1] [,2] [,3]  
[1,]    1   10   27  
[2,]    8   30   60
```

```
[3,] 21 56 99
```

```
[4,] 40 88 144
```

```
> # 원래 행렬의 곱셈과 나누기
```

```
> t(m1) %*% m2
```

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

```
[1,] 70 80 90
```

```
[2,] 158 184 210
```

```
[3,] 246 288 330
```

2.4 배열

- 동일한 유형의 데이터가 2차원 이상으로 구성된 구조
 - 동일 차원($n \times p$)의 행렬 k 개의 방에 저장되어 있는 구조

```
> # 1~24까지의 숫자를 '2 x 3 행렬'로 해서 '4층' 짜리의 데이터 구조를 만들어라
```

```
> a1 <- array(1:24, c(2, 3, 4))
```

```
> a1
```

```
, , 1
```

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

```
[1,] 1 3 5
```

```
[2,] 2 4 6
```

```
, , 2
```

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

```
[1,] 7 9 11
```

```
[2,] 8 10 12
```

```
, , 3
```

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

```
[1,] 13 15 17
```

```
[2,] 14 16 18
```

, , 4

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

```
[1,] 19 21 23
```

```
[2,] 20 22 24
```

```
> # 1~24까지의 숫자를 '3 x 4' 행렬로 해서 '2층'짜리의 데이터 구조를 만들어라
```

```
> a2 <- array(1:23, c(3, 4, 2))
```

```
> a2
```

, , 1

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

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

```
[2,] 2 5 8 11
```

```
[3,] 3 6 9 12
```

, , 2

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

```
[1,] 13 16 19 22
```

```
[2,] 14 17 20 23
```

```
[3,] 15 18 21 1
```

```
> # 출처: http://rfriend.tistory.com/14?category=601862 [R, Python 분석과
```

```
> # 프로그래밍 (by R Friend)]
```

2.5 데이터 프레임

- 데이터 프레임은 데이터 유형에 상관없이 2차원 형태의 데이터 구조

- 외부로부터 불러오는 자료의 형태 대부분은 데이터 프레임
- 참고: 행렬은 동일한 유형의 자료형으로 이루어짐

2.6 리스트

```
> # Vector(L1), Matrix(L2), Array(L3), Data Frame(L4)를 만들어서, 하나의
> # List(L5)로 묶어라
> L1 <- c(1, 2, 3, 4) # Vector
> L2 <- matrix(1:6, 3, byrow = TRUE) # Matrix
> L3 <- array(1:24, c(3, 4, 2)) # Array
> L4 <- data.frame(cust_id = c(1, 2, 3, 4), last_name = c("Kim", "Lee", "Choi",
+     "Park")) # Data Frame
> L5 <- list(L1, L2, L3, L4) # List
> # [[1]]는 Vector(L1), [[2]]는 Matrix(L2), [[3]]는 Array(L3), [[4]]는 Data
> # Frame(L4)가 묶인 것임
> L5
```

[[1]]

[1] 1 2 3 4

[[2]]

[,1] [,2]

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

[[3]]

, , 1

[,1] [,2] [,3] [,4]

[1,]	1	4	7	10
[2,]	2	5	8	11

```
[3,]    3    6    9   12
```

```
, , 2
```

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

```
[1,] 13 16 19 22
```

```
[2,] 14 17 20 23
```

```
[3,] 15 18 21 24
```

```
[[4]]
```

```
cust_id last_name  
1      1      Kim  
2      2      Lee  
3      3      Choi  
4      4      Park
```

```
> # 출처: http://rfriend.tistory.com/14?category=601862 [R, Python 분석과
```

```
> # 프로그래밍 (by R Friend)]
```

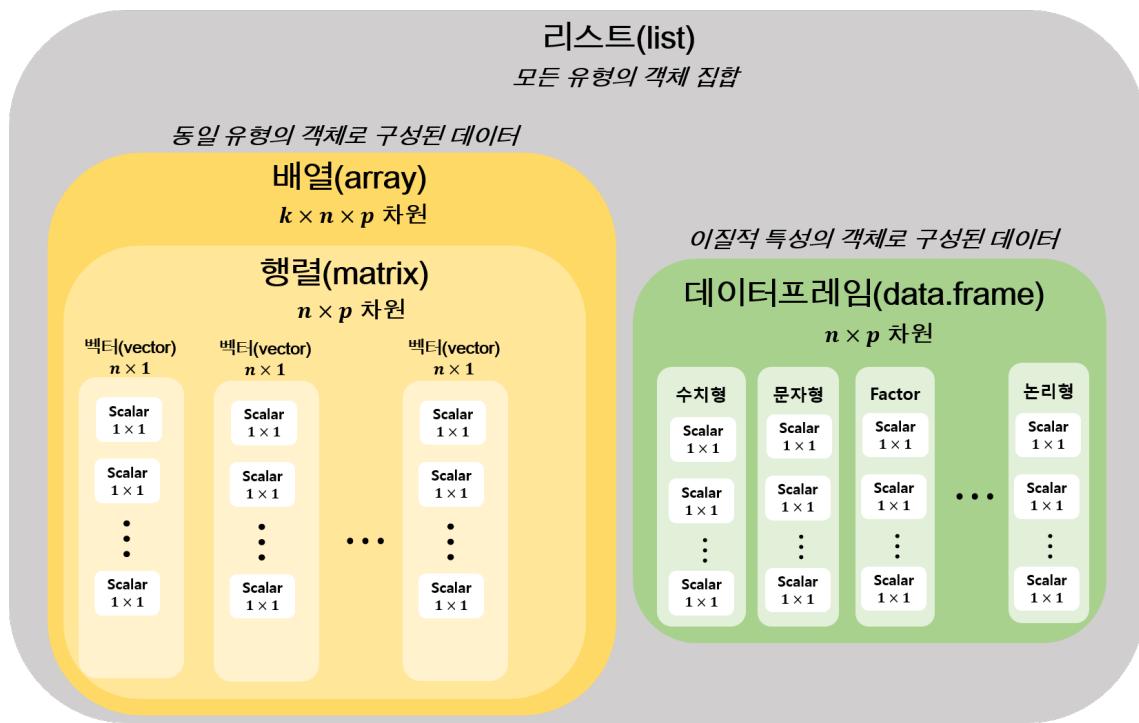


Figure 2.1: R 데이터 타입 구조 다이어그램: R, Python 분석과 프로그래밍 (by R Friend)에서 발췌 후 수정

References

RFriend r, python 분석과 프로그래밍. <http://rfriend.tistory.com>. Accessed: 2017-11-20.

고석범. *R과 Knitr를 활용한 데이터 연동형 문서 만들기(에이콘 데이터 과학 13)*. 에이콘출판, 2014. ISBN 9788960775510. URL <https://books.google.co.kr/books?id=7UXFoAEACAAJ>.

서민구. *R을 이용한 데이터 처리 & 분석 실무*. 길벗, 2014. ISBN 9788966188260. URL <https://books.google.co.kr/books?id=0GjAoQEACAAJ>.