

R프로그래밍

3장. 작업 환경 활용하기

박 혜 승 교 수



3장. 작업 환경 활용하기

3.1 R의 작업 디렉토리

3.2 전역 환경 둘러보기

3.3 전역 설정 수정하기

3.4 패키지 라이브러리 관리하기

3.5 마치기

3.1 R의 작업 디렉터리

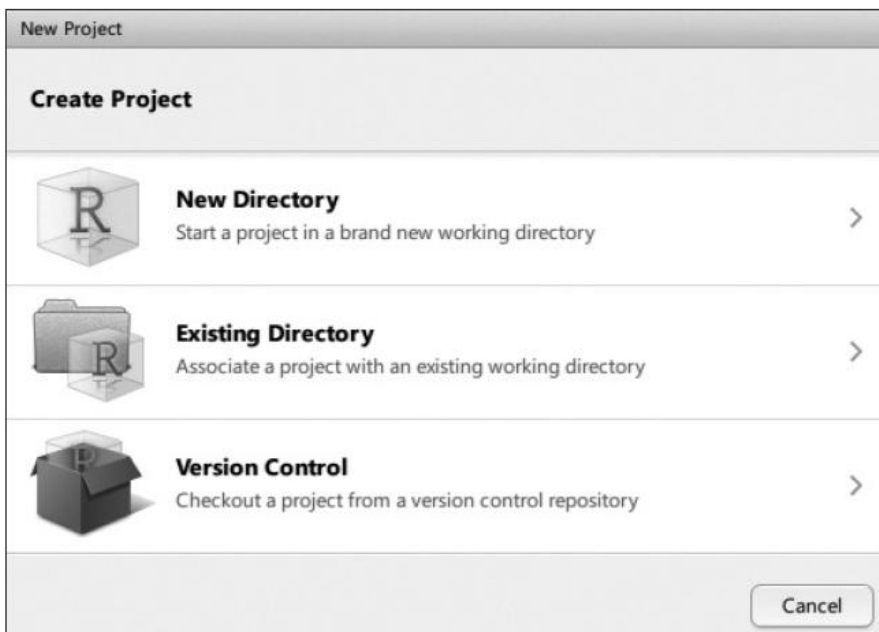
• R의 작업 디렉터리

- ✓ R 세션을 R 터미널을 이용하여 만들거나 Rstudio에서 시작하면 항상 정해진 디렉터리에서 출발.
- ✓ R이 동작하고 있는 디렉터를 R 세션의 작업 디렉터리(working directory)라고 함.
- ✓ 하드 드라이브에 있는 다른 파일에 접근하는 방법.
 - ❖ 현재 작업 디렉터를 D:\Workspaces\test-project\ 라고 가정할 때
 - ❖ 절대 경로: 예) D:\Workspaces\testproject\data\2015.csv
 - ❖ 상대 경로: 예) data\2015.csv
 - ❖ 상대 경로를 사용하면 파일 경로 자체를 변경하지 않아도 되고 명령도 더 짧아짐.
 - ❖ 스크립트도 더 쉽게 이동 가능.
 - ❖ 어떤 디렉터리에 있는 여러 데이터 파일로 그래프를 만드는 R 스크립트를 작성한다고 가정할 때
 - ❖ 절대 경로: 다른 사용자가 이 스크립트 실행 시 경로를 자기 하드 드라이브의 위치로 수정해야 함.
 - ❖ 상대 경로: 데이터가 동일한 상대 위치에 있을 경우, 스크립트를 따로 수정하지 않아도 잘 동작.
- ✓ `getwd()` 함수: 현재 R 세션의 작업 디렉터리 확인 가능.
- ✓ 기본적으로 R 콘솔 창에서는 사용자 디렉터리에서 새로운 R 세션 시작.
- ✓ RStudio는 사용자 문서 디렉터리에서 R 세션을 실행.
- ✓ RStudio에서는 기본값이 아닌 다른 디렉터를 선택하여 R 프로젝트 생성 가능.

3.1 R의 작업 디렉터리

• R의 작업 디렉터리

- ✓ 새 프로젝트를 만들려면 [File] > [New Project]를 선택하거나 메인 창의 오른쪽 위에 있는 프로젝트 드롭 다운 버튼을 눌러 [New Project]를 선택해야 함
- ✓ 이후 새 프로젝트를 위해 디렉터를 새로 만들거나 기존 디렉터를 선택.
- ✓ 선택한 디렉터리에 새 R 프로젝트 생성.
- ✓ R 프로젝트란 결국 어떤 설정 정보들이 들어 있는 **.Rproj 파일**.
- ✓ RStudio에서 프로젝트 파일을 열면 프로젝트 내부 설정들이 적용되고, 프로젝트 파일이 있는 디렉터를 작업 디렉터리 설정.
- ✓ [그림 3-1] 새 프로젝트를 위한 디렉터리 선택



3.1 R의 작업 디렉터리

- RStudio에서 R 프로젝트 생성하기

- ✓ RStudio가 제공하는 또 다른 유용한 기능은 바로 '자동 완성 기능'.
- ✓ 이 기능은 파일 경로를 좀 더 효과적으로 작성할 수 있게 도움.
- ✓ 절대 경로나 상대 경로를 작성하려고 문자열을 입력할 때 **Tab** 을 누르면 그 디렉터리에 있는 파일들을 보여줌.
- ✓ [그림 3-2] 파일 경로의 자동 완성 기능



3.1 R의 작업 디렉터리

- 절대 경로와 상대 경로

- ✓ 현재 사용 중인 프로젝트의 작업 디렉터리는 다음과 같음.

```
> getwd()
```

```
[1] "D:/Workspaces/learn-r-programming"
```

- ✓ 이전에는 작업 디렉터리를 다룰 때 ₩가 아닌 /를 사용했는데, 아마 기억할 것임
- ✓ 윈도우 시스템에서는 경로를 구분하는 기호로 ₩를 기본적으로 사용함.
- ✓ 이 기호는 특수 문자를 표시하는 데 이미 사용 중.
- ✓ 예를 들어 문자형 벡터를 생성할 때 줄 바꿈을 하는 데 ₩n을 사용 가능.

```
> "Hello₩nWorld"
```

```
[1] "Hello₩nWorld"
```

- ✓ 이 특수 문자는 문자형 벡터를 문자열 그대로 직접 출력할 때는 그대로 보존됨

3.1 R의 작업 디렉터리

- 절대 경로와 상대 경로

- ✓ cat() 함수를 사용하면 이스케이프 문자를 줄 바꿈을 표시하는 문자로 인식하여 출력.

```
> cat("Hello\nWorld")
```

```
Hello
```

```
World
```

- ✓ 두 번째 문자가 줄 바꿈 된 후에 정상적으로 시작하는 것을 볼 수 있음.
- ✓ ₩가 정말 특별하다면 문자열에서 ₩ 문자 자체를 ₩₩로 입력.

```
> cat("The string with '\\\' is translated")
```

```
The string with '\\' is translated
```

3.1 R의 작업 디렉터리

- 절대 경로와 상대 경로

- ✓ 윈도우 시스템에서 파일 경로를 표기할 때는 \나 /를 사용.
- ✓ macOS나 리눅스 등 유닉스 계열의 시스템에서는 단순히 /를 사용.
- ✓ 윈도우 사용자가 \를 잘못 사용하면 다음 오류 메시지를 표시.

```
> filename <- "d:\data\test.csv"
```

```
Error: '\d' is an unrecognized escape in character string starting ""d:\d"
```

- ✓ 오류가 표시되지 않게 하려면 다음과 같이 입력해야 함.

```
> filename <- "d:\\data\\test.csv"
```


3.1 R의 작업 디렉터리

- 절대 경로와 상대 경로

- ✓ 물론 대부분 윈도우에서는 / 사용 가능.
- ✓ 상대 경로를 지원하는 거의 모든 운영 체제에서는 똑같이 동작.

```
> absolute_filename <- "d:/data/test.csv"
```

```
> relative_filename <- "data/test.csv"
```

3.1 R의 작업 디렉터리

- 절대 경로와 상대 경로

- ✓ `setwd()` 함수로 현재 R 세션의 작업 디렉터를 변경 가능하지만, 추천하지 않음.
- ✓ 스크립트에 있는 모든 상대 경로가 다른 디렉터를 가리키게 되고, 이것 때문에 뭔가가 잘못될 수도 있기 때문.
- ✓ 이러한 면에서 R 프로젝트를 만들어 작업을 시작하는 것이 경로 문제를 해결하는 데는 실제로 도움이 됨.

3.1 R의 작업 디렉터리

• 프로젝트 파일 관리하기

- ✓ RStudio에서 프로젝트를 하나 만들면 프로젝트 디렉터리에 오직 .Rproj 파일만 하나 생성.
- ✓ R 언어는 통계 계산을 하거나 데이터를 시각화하는 도구 → R 프로젝트는 주로 통계 처리(아니면 다른 프로그래밍 작업)를 하는 R 스크립트, 데이터 파일(csv 파일), 기타 문서(마크다운 ; .md 파일)와 출력 그래프를 포함.
- ✓ 프로젝트 디렉터리 안에 다양한 종류의 파일이 섞여 있다면, 특히 입력 데이터가 점점 증가.
- ✓ 출력 데이터나 그래프가 디렉터리 안에 어지럽게 쌓이면 파일을 관리하기가 더욱 어려움.
- ✓ 이때는 '하위 디렉터리'를 만들어 서로 다른 종류의 작업에서 얻은 여러 가지 파일을 관리하는 것을 권장.

3.1 R의 작업 디렉터리

- 프로젝트 파일 관리하기

- ✓ 예를 들어 다음과 같이 모든 파일을 한꺼번에 넣은 일반적인 디렉터리 구조가 있음

- project/

- household.csv
 - population.csv
 - national-income.png
 - population-density.png
 - utils.R
 - import-data.R
 - check-data.R
 - plot.R
 - README.md
 - NOTES.md

3.1 R의 작업 디렉터리

- 프로젝트 파일 관리하기

✓ 반면에 다음과 같이 정리된 디렉터리 구조는 훨씬 깔끔.

```
project/  
  - data/  
    - household.csv  
    - population.csv  
  - graphics/  
    - national-income.png  
    - population-density.png  
  - R/  
    - utils.R  
    - import-data.R  
    - check-data.R  
    - plot.R  
  - README.md  
  - NOTES.md
```

3.1 R의 작업 디렉터리

• 프로젝트 파일 관리하기

- ✓ 첫 번째 디렉터리 구조는 "디렉터리_이름/"과 "파일_이름.파일_확장자" 형태로 구성.
- ✓ 프로젝트가 점점 커지고 복잡해질수록 첫 번째 구조는 더욱 복잡화.
- ✓ 두 번째 구조는 여전히 정리 정돈이 잘된 형태이기 때문에 더 유용.
- ✓ 구조 문제와 별개로, 일반적으로 프로젝트 소개를 담은 README.md 파일과 추가 기록을 담은 NOTES.md 파일을 만들어 보자.
 - ❖ 정말 단순한 구문을 사용하여 만든 마크다운 문서(.md)로, 간단한 마크다운 구문법을 익혀 두면 유용.
 - ❖ Daring Fireball 블로그의 마크다운 구문법(<https://daringfireball.net/projects/markdown/syntax>) 참고.
 - ❖ github 도움말의 마크다운 베이직(<https://help.github.com/articles/markdown-basics/>) 참고.
 - ❖ 15장에서 R과 마크다운 활용에 대해 다룰 예정.

3.2 작업 환경 둘러보기

• 작업 환경 둘러보기

- ✓ R은 모든 표현식을 어떤 특정 환경에서 평가.
 - ❖ 표현식을 평가(evaluate an expression)은 단순히 '코드를 실행한다'보다는 '코드 일부를 실행하여 어떤 결과를 얻는 것'을 의미
- ✓ R 환경이란 기호와 그것에 바인딩(매핑)된 값의 집합.
 - ❖ 어떤 기호에 값을 연결하거나 함수를 호출하거나 어떤 이름을 참조할 때, R은 현재 환경에서 이름이 같은 기호를 탐색.
- ✓ Rstudio에서 어떤 명령을 입력하면 R은 전역 환경(global environment)에서 이 명령을 평가.
 - ❖ 터미널이나 RStudio에서 새 R 세션을 시작한다면, '전역 환경이 비어 있는 상태'로 작업을 시작.
 - ❖ 즉, 환경에 정의된 기호가 하나도 없는 상태.
 - ❖ `x <- c(1, 2, 3)`이라는 명령을 실행하면, 전역 환경에서 x 기호가 수치형 벡터 `c(1, 2, 3)`에 바인딩.
 - ❖ 결과적으로, 전역 환경에는 x를 `c(1, 2, 3)` 벡터에 매핑하는 하나의 바인딩이 존재.
 - ❖ 즉, 표현식 x를 평가하면 여기에 연결된 값을 얻게 됨.

3.2 작업 환경 둘러보기

- 이미 있는 기호 살펴보기

- ✓ `objects()` 함수: 환경 안에 들어 있는 객체 정보 표시.
 - ❖ 현재 환경에 있는 객체 이름을 담은 문자형 벡터를 리턴.
- ✓ 다음과 같이 새로 시작하는 R 세션에는 어떤 기호도 없음.

```
> objects()
```

```
character(0)
```

- ✓ 다음과 같이 객체들을 만들어 보자.

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

```
> y <- c("a", "b", "c")
```

```
> z <- list(m = 1:5, n = c("x", "y", "z"))
```


3.2 작업 환경 둘러보기

• 이미 있는 기호 살펴보기

- ✓ 이제는 객체 이름을 확인할 수 있음.

```
> objects()
```

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

- ✓ ls() 함수: objects() 함수와 동일한 기능.

```
> ls()
```

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

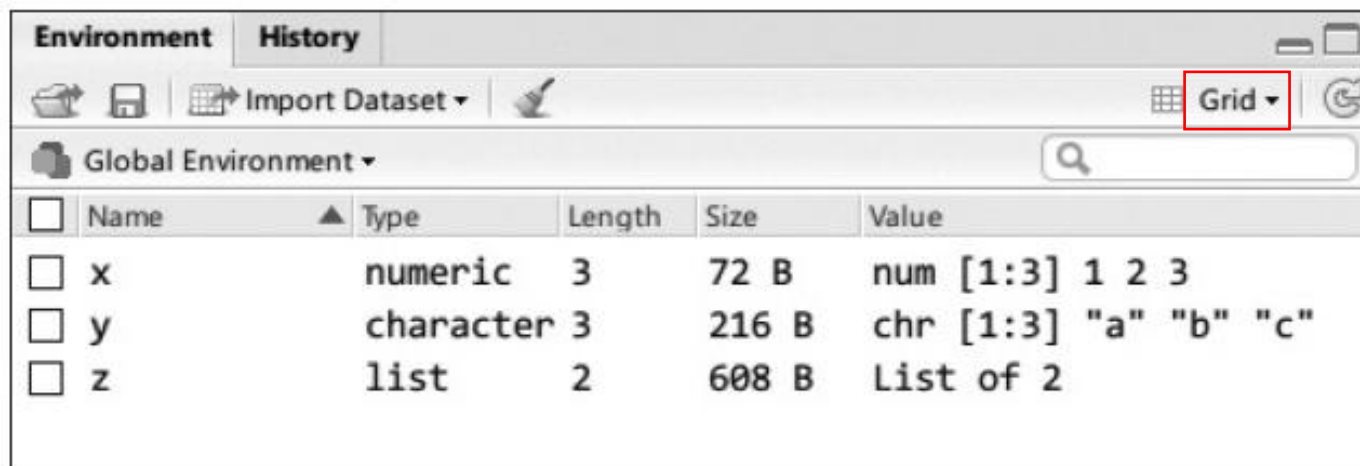
- ✓ Rstudio 환경 창에서 전역 환경의 모든 기호 확인 가능.

Environment		History
<div> <div> <div></div> <div></div> <div></div> </div> <div>Import Dataset</div> <div></div> </div> <div>List</div>		
Global Environment		
Values		
x	num [1:3] 1 2 3	
y	chr [1:3] "a" "b" "c"	
z	List of 2	
m:	int [1:5] 1 2 3 4 5	
n:	chr [1:3] "x" "y" "z"	

3.2 작업 환경 둘러보기

• 이미 있는 기호 살펴보기

- ✓ 환경 창에서 객체의 기호와 값 확인 가능.
 - ❖ 리스트나 데이터 프레임 객체는 확장 버튼을 사용하여 벡터 안의 값을 더 자세히 확인 가능.
- ✓ 추가로 격자 보기(grid)도 지원.
 - ❖ 격자 보기는 객체의 이름, 종류, 값, 크기 확인 가능.
- ✓ [그림 3-4] 환경 창의 격자 보기



<input type="checkbox"/>	Name	Type	Length	Size	Value
<input type="checkbox"/>	x	numeric	3	72 B	num [1:3] 1 2 3
<input type="checkbox"/>	y	character	3	216 B	chr [1:3] "a" "b" "c"
<input type="checkbox"/>	z	list	2	608 B	List of 2

3.2 작업 환경 둘러보기

- 객체 구조 보기

- ✓ `str()` 함수: 환경 창에 표시되는 간략한 객체 설명.
 - ❖ 주어진 객체의 구조 정보를 출력
- ✓ 예) 간단한 수치형 벡터에 이 함수를 적용하면 객체의 종류, 크기, 값 확인 가능.

```
> x
```

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

```
> str(x)
```

```
num [1:3] 1 2 3
```

3.2 작업 환경 둘러보기

- 객체 구조 보기

- ✓ 벡터 안에 원소가 10개 이상 있다면 str() 함수는 첫 원소 10개만 출력.

```
> str(1:30)
```

```
int [1:30] 1 2 3 4 5 6 7 8 9 10 ...
```

- ✓ 리스트는 콘솔 창에서 직접 실행하거나 print() 함수를 사용하면 내용 상세히 확인 가능.

```
> z
```

```
$m
```

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

```
$n
```

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

3.2 작업 환경 둘러보기

• 객체 구조 보기

- ✓ 리스트에 str() 함수를 사용하면 객체의 종류, 길이, 원소 구조 확인 가능.

```
> str(z)
List of 2
 $ m: int [1:5] 1 2 3 4 5
 $ n: chr [1:3] "x" "y" "z"
```

- ✓ 다음과 같이 중첩 리스트를 만들었다고 가정해 보자

```
> nested_list <- list(m = 1:15, n = list("a", c(1, 2, 3)),
+   p = list(x = 1:10, y = c("a", "b")), q = list(x = 0:9, y = c("c", "d")))
```

- ✓ 직접 이 리스트를 출력하면 리스트 안에 있는 모든 원소와 내용 표시.

```
> nested_list
 $p
 $p$x
 [1]  1  2  3  4  5  6  7  8  9 10

 $n
 $n[[1]]
 [1] "a"

 $n[[2]]
 [1] 1 2 3

 $q
 $q$x
 [1] 0 1 2 3 4 5 6 7 8 9

 $q$y
 [1] "c" "d"
```

3.2 작업 환경 둘러보기

- 객체 구조 보기

- ✓ 좀 더 간결하게 보고 싶다면 `str()` 함수를 사용하자.

```
> str(nested_list)
```

```
List of 4
```

```
$ m: int [1:15] 1 2 3 4 5 6 7 8 9 10 ...
```

```
$ n:List of 2
```

```
..$ : chr "a"
```

```
..$ : num [1:3] 1 2 3
```

```
$ p:List of 2
```

```
..$ x: int [1:10] 1 2 3 4 5 6 7 8 9 10
```

```
..$ y: chr [1:2] "a" "b"
```

```
$ q:List of 2
```

```
..$ x: int [1:10] 0 1 2 3 4 5 6 7 8 9
```

```
..$ y: chr [1:2] "c" "d"
```

3.2 작업 환경 둘러보기

- 객체 구조 보기

- ✓ `str()` 함수는 한 객체에 대한 구조 정보 표시.
- ✓ `ls.str()` 함수는 **현재 환경에 대한 구조** 정보 표시.

```
> ls.str()
```

```
nested_list : List of 4
```

```
$ m: int [1:15] 1 2 3 4 5 6 7 8 9 10 ...
```

```
$ n:List of 2
```

```
$ p:List of 2
```

```
$ q:List of 2
```

```
x : num [1:3] 1 2 3
```

```
y : chr [1:3] "a" "b" "c"
```

```
z : List of 2
```

```
$ m: int [1:5] 1 2 3 4 5
```

```
$ n: chr [1:3] "x" "y" "z"
```

3.2 작업 환경 둘러보기

• 객체 구조 보기

- ✓ ls.str() 함수 기능은 RStudio 환경 창의 기능과 유사.
- ✓ 사용자가 원하는 대로 설정한 환경을 조사하거나 어떤 특정 변수의 구조를 알고 싶을 때 유용.
- ✓ ls.str() 함수에는 원하는 객체만 필터링할 수 있는 mode라는 인수 존재.
- ✓ 다음과 같이 설정하면 리스트 객체만 확인 가능.

```
> ls.str(mode = "list")
```

```
nested_list : List of 4
```

```
 $ m: int [1:15] 1 2 3 4 5 6 7 8 9 10 ...
```

```
 $ n:List of 2
```

```
 $ p:List of 2
```

```
 $ q:List of 2
```

```
z : List of 2
```

```
 $ m: int [1:5] 1 2 3 4 5
```

```
 $ n: chr [1:3] "x" "y" "z"
```


3.2 작업 환경 둘러보기

- 객체 구조 보기

- ✓ 필터링하는 또 다른 방법은 바로 **pattern 인수**를 사용하는 것.
- ✓ 설정한 패턴에 맞는 이름만 선택 가능.
- ✓ pattern 인수는 **정규 표현식**을 입력으로 받음.
- ✓ 단일 문자로 된 이름을 갖는 객체만 보고 싶다면 다음과 같이 할 수 있음.

```
> ls.str(pattern = "^\\w$")  
x :  num [1:3] 1 2 3  
y :  chr [1:3] "a" "b" "c"  
z :  List of 2  
  $ m: int [1:5] 1 2 3 4 5  
  $ n: chr [1:3] "x" "y" "z"
```

3.2 작업 환경 둘러보기

- 객체 구조 보기

- ✓ 이름이 단일 문자이면서 리스트인 객체만 원한다면 pattern과 mode 인수를 동시에 사용 가능.

```
> ls.str(pattern = "^\\w$", mode = "list")
```

```
z : List of 2
```

```
$ m: int [1:5] 1 2 3 4 5
```

```
$ n: chr [1:3] "x" "y" "z"
```

- ✓ 명령에서 `^\\w$` 부분이 어렵더라도 걱정하지 말자.

- ❖ (문자열 시작) (a, b, c 같은 문자 하나) (문자열 끝) 같은 형식에 일치하는 모든 문자열을 의미.

3.2 작업 환경 둘러보기

- 기호 제거하기

- ✓ `remove()` 또는 `rm()` 함수: 사용 중인 환경의 기호 제거 가능.
- ✓ `x`를 제거하기 전, 환경 안에 있는 객체는 다음과 같음.

```
> ls()  
[1] "nested_list" "x"          "y"          "z"
```

- ✓ `rm()` 함수로 `x`를 제거한 후 객체는 다음과 같음.

```
> rm(x)  
> ls()  
[1] "nested_list" "y"          "z"
```

3.2 작업 환경 둘러보기

- 기호 제거하기

- ✓ 이 함수를 사용할 때 “객체 이름의 문자열” 입력 가능.
- ✓ 즉, `rm("x")` 역시 같은 결과.
- ✓ 다음과 같이 여러 객체를 동시에 제거 가능.

```
> rm(y, z)
```

```
> ls()
```

```
[1] "nested_list"
```

- ✓ 지우려는 객체가 현재 환경에 없을 때는 다음 경고 메시지를 표시.

```
> rm(x)
```

```
Warning message:
```

```
In rm(x) : object 'x' not found
```

3.2 작업 환경 둘러보기

- 기호 제거하기

- ✓ `rm()` 함수는 삭제하고자 하는 “객체 이름이 담긴 문자형 벡터” 입력 가능.

```
> p <- 1:10  
> q <- seq(1, 20, 5)  
> v <- c("p", "q")  
> rm(list = v)  
> ls()  
[1] "nested_list" "v"
```

- ✓ 환경에 있는 모든 객체를 삭제하고 싶다면, `rm()`과 `ls()` 함수를 조합해서 다음과 같이 사용.

```
> rm(list = ls())  
> ls()  
character(0)
```

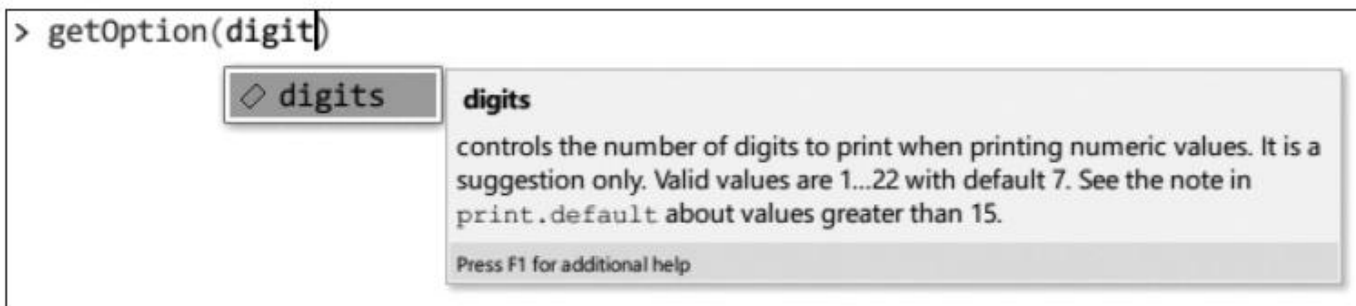
3.3 전역 설정 수정하기

• 전역 설정 수정하기

- ✓ 작업 환경에서 객체를 다루는 것과는 달리, R 옵션은 현재 R 세션에 전체적으로 영향을 미침.
- ✓ `getOption()` 함수: 해당하는 옵션의 현재 값 확인.
- ✓ `options()` 함수: 옵션 수정 가능.

• 표기되는 숫자 개수 조정하기

- ✓ RStudio에서 `getOption(+ Tab)` 을 입력하면 사용할 수 있는 옵션 목록과 해당 설명 확인 가능.
- ✓ [그림 3-5] `getOption`에서 Tab 을 활용하여 가능한 옵션 목록 확인



- ✓ 일반적으로 자주 쓰는 옵션 가운데 하나는 바로 '표기되는 숫자 개수'.
 - ❖ 숫자를 더 정밀하게 표기해야 할 때 현재 설정이 충분하지 않을 수 있음.
 - ❖ R 세션에서 화면에 표기되는 숫자 개수는 모두 이 옵션으로 관리.
 - ❖ `getOption()`을 호출하여 숫자의 현재 값 확인.
 - ❖ `options()` 함수에 `digits` 옵션을 활용하여 숫자 개수를 더 큰 값으로 설정.

3.3 전역 설정 수정하기

- 표기되는 숫자 개수 조정하기

- ✓ R 세션을 시작하면 기본적으로 이 옵션 값은 7.

- ✓ 다음 코드를 실행하여 이것을 확인해 보자.

```
> 123.12345678
```

```
[1] 123.1235
```

- ✓ 입력한 값은 모두 숫자 11개이지만, 7개만 표시.

- ✓ 결국 뒷부분 숫자는 사라지고 앞부분 숫자 7개만 출력된 것.

- ✓ digits = 7이라는 설정 때문에 실제 값의 정밀도가 떨어지는 것은 아님.

```
> 0.10000002
```

```
[1] 0.1
```

```
> 0.10000002 - 0.1
```

```
[1] 2e-08
```

3.3 전역 설정 수정하기

- 표기되는 숫자 개수 조정하기

- ✓ 기본 설정으로 소수점 7자리에서 반올림해서 0.10000002가 0.1이 된 것이라면, 두 번째 표현식의 결과는 0이 되어야 함.
- ✓ `digits = 7`이라는 설정은 말 그대로 숫자를 표시할 때 사용할 숫자 개수를 의미. 값의 반올림은 일어나지 않음.
- ✓ 소수점 앞의 숫자가 너무 크면 소수점 이후의 숫자가 모두 무시될 때도 있을 수 있음. 이 경우 기본 설정에서는 정수 부분만 출력.

```
> 1234567.12345678
```

```
[1] 1234567
```


3.3 전역 설정 수정하기

- 표기되는 숫자 개수 조정하기

✓ 출력되는 숫자 개수를 늘리고 싶다면, 다음과 같이 기본 설정 값인 7보다 더 큰 값으로 설정.

```
> getOption("digits")  
[1] 7  
  
> 1e10 + 0.5  
[1] 1e+10  
  
> options(digits = 15)  
  
> 1e10 + 0.5  
[1] 10000000000.5
```

3.3 전역 설정 수정하기

- 표기되는 숫자 개수 조정하기

- ✓ options() 함수가 호출되는 즉시 모든 명령에 동일한 효과가 적용.
- ✓ 다시 옵션을 원래대로 되돌리고 싶다면 다음과 같이 설정.

```
> options(digits = 7)
```

```
> 1e10 + 0.5
```

```
[1] 1e+10
```

3.3 전역 설정 수정하기

- 경고 메시지 레벨 조정하기

- ✓ 또 다른 옵션은 경고 메시지의 레벨을 조정할 수 있는 warn.

```
> getOption("warn")
```

```
[1] 0
```

- ✓ 기본 경고 레벨은 0.
- ✓ 경고가 발생한다고 해도 코드는 멈추지 않음.
- ✓ 오류가 발생하면 그 즉시 코드가 종료.
- ✓ 경고가 여러 개 발생하면 한꺼번에 표시.
- ✓ 예) 다음과 같이 어떤 문자열을 수치형 벡터로 변환하는 코드는 경고를 발생시키고, 결과로 결측 값을 얻음. 경고 옵션을 warn=-1로 설정할 수 있으나, 결과는 동일.

```
> as.numeric("hello")
```

```
[1] NA
```

경고메시지(들):

강제 형변환에 의해 생성된 NA 입니다

```
> options(warn=-1)
```

```
> as.numeric("hello")
```

```
[1] NA
```

3.3 전역 설정 수정하기

• 경고 메시지 레벨 조정하기

- ✓ warn을 1 또는 2로 설정하면 코드에 버그가 될 만한 부분이 있을 때 더 빨리 실패하게 함.
- ✓ warn = 0일 때 함수를 실행하면 기본적으로 일단 값을 반환한 후 모든 경고 메시지를 함께 표시.
- ✓ 예) 다음 문자열 2개를 받는 함수를 호출해서 살펴보자.

```
> f <- function(x, y) {  
+   as.numeric(x) + as.numeric(y)  
+ }
```

- ❖ 경고 레벨이 기본값(0)일 때, 함수의 결과값을 출력하고 모든 경고 메시지가 뒤이어 나옴.

```
> options(warn = 0)  
> f("hello", "world")  
[1] NA
```

Warning messages:

```
1: In f("hello", "world") : NAs introduced by coercion  
2: In f("hello", "world") : NAs introduced by coercion
```

- ❖ 함수는 입력 인수를 2개 받아 모두 수치형 벡터로 강제 변환.
- ❖ 입력 인수가 모두 문자열이기 때문에 경고가 2개 발생했지만, 함수의 결과를 출력한 뒤에 경고 표시.
- ❖ 계산량이 많은 작업을 수행하려고 시간이 상당히 많이 걸릴 때도 최종 결과를 출력한 뒤에 경고 표시.
- ❖ 사실 계산 결과는 처음부터 정확한 결과에서 벗어남.

3.3 전역 설정 수정하기

- 경고 메시지 레벨 조정하기

- ✓ `warn = 1`을 사용하면, 경고가 발생하는 순간 경고 메시지를 바로 출력.

```
> options(warn = 1)
```

```
> f("hello", "world")
```

```
Warning in f("hello", "world") : NAs introduced by coercion
```

```
Warning in f("hello", "world") : NAs introduced by coercion
```

```
[1] NA
```

- ✓ 결과는 같지만 경고 문구는 결과보다 앞에 나옴.
- ✓ 시간이 오래 걸리는 함수는 경고 메시지를 먼저 볼 수 있으니 코드 실행을 멈출지 결정 가능하고, 뭐가 잘못되었는지도 파악 가능.

3.3 전역 설정 수정하기

- 경고 메시지 레벨 조정하기

- ✓ 경고 레벨을 더 엄격하게 설정 가능.
- ✓ warn = 2는 경고를 '직접적인 오류'로 간주.

```
> options(warn = 2)
```

```
> f("hello", "world")
```

```
Error in f("hello", "world") :
```

```
(converted from warning) NAs introduced by coercion
```

- ✓ 이러한 옵션은 전역 환경에 영향을 미침.
- ✓ 모든 R 세션을 공통으로 관리하는 것은 편리하지만, **옵션을 변경하는 것은 위험**할 수 있음.
- ✓ 작업 디렉터리를 변경하면 스크립트 안의 모든 상대 경로가 제대로 실행되지 않을 수 있음.
- ✓ 전역 옵션을 변경하면 전역 옵션과 호환되지 않는 모든 후속 코드가 동작하지 않을 수 있음.
- ✓ 일반적으로 절대적으로 필요한 경우가 아니라면, **전역 옵션은 수정하지 않는 것을 권장**.

3.4 패키지 라이브러리 관리하기

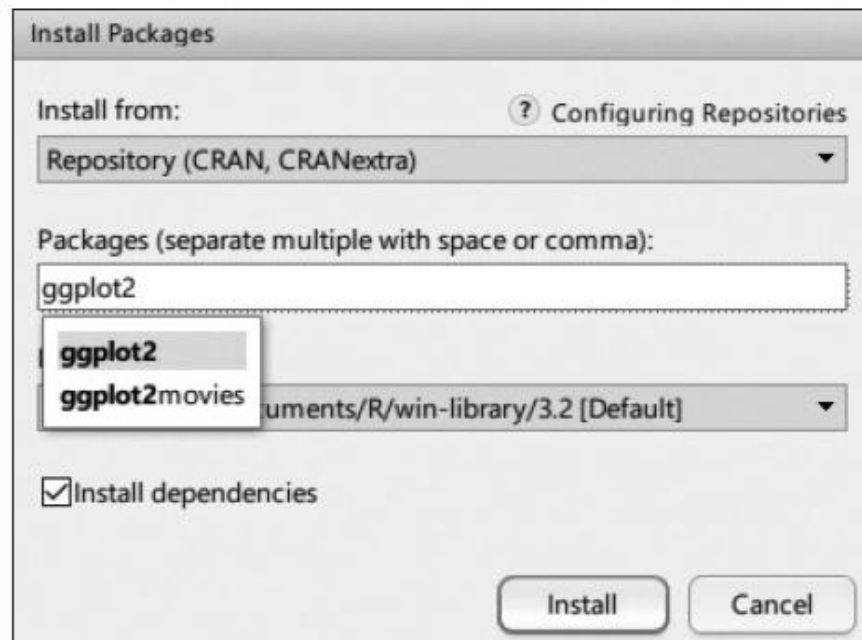
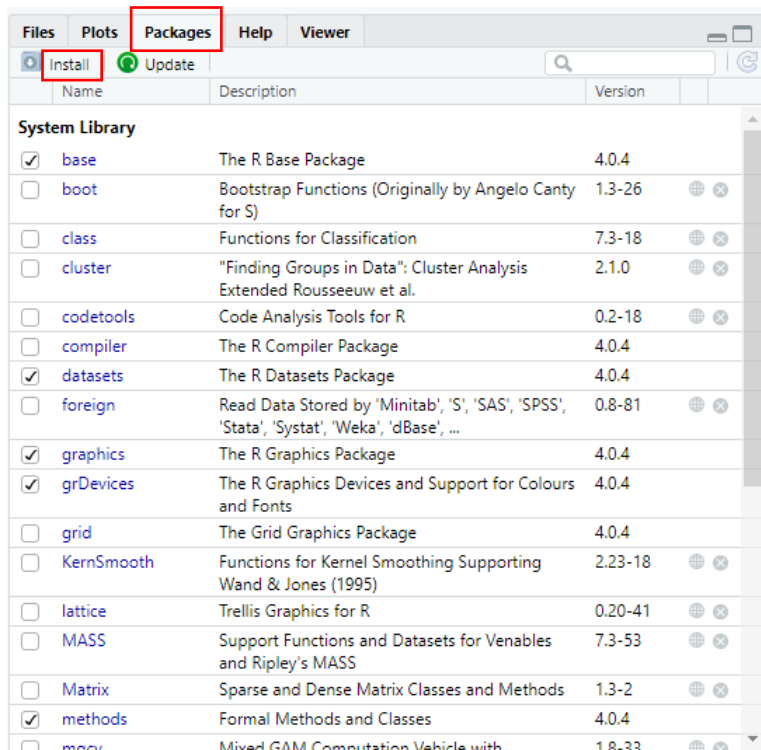
• 패키지 라이브러리 관리하기

- ✓ 패키지는 미리 정의된 함수들을 담은 컨테이너.
- ✓ 패키지는 특정 범위의 문제를 해결하는 함수의 집합.
- ✓ 통계 평가, 데이터 마이닝, 데이터베이스 인터페이스 또는 최적화 도구들을 구현한 것.
- ✓ 특정 범위의 문제를 해결하고자 충분히 일반화되도록 설계.
- ✓ Comprehensive R Archive Network (CRAN)
 - ❖ R의 소스 코드와 패키지 수천 개가 이 보관소에 존재.
 - ❖ 전 세계 8832명이 넘는 패키지 관리자가 활성 패키지 1만 5523개를 관리. (2020년 3월 30일 기준)
 - ❖ 매주 패키지가 400개 정도 업데이트되며, 2000만 번 이상 다운로드.
 - ❖ 모든 패키지를 알 필요는 없고, 가장 유용하고 분야와 관련이 깊은 패키지만 알아도 충분.
 - ❖ CRAN Task Views(<https://cran.rstudio.com/web/views/>)
 - ❖ METACRAN(<http://www.r-pkg.org>)
 - ❖ 가장 일반적으로 사용하는 패키지나 자신의 분야와 밀접하게 관련한 패키지를 먼저 배우는 것 권장.
 - ❖ 특정 패키지 사용법을 배우기 전에 패키지 설치 방법과 동작 방식에 대해 이해해야 함.

3.4 패키지 라이브러리 관리하기

• CRAN에서 패키지 설치하기

- ✓ RStudio에서는 패키지를 설치할 수 있는 쉬운 방법 제공.
- ✓ 패키지 창으로 가서 Install 버튼을 클릭하면 다음 창이 생성.
- ✓ [그림 3-7] 패키지 설치 창



3.4 패키지 라이브러리 관리하기

- CRAN에서 패키지 설치하기

- ✓ 어떤 패키지에서 함수를 호출할 때 이 함수가 다른 패키지에 있는 함수를 일부 호출하는 경우가 있음. 이때는 당연히 다른 해당 패키지들도 함께 설치해야 함.
- ✓ `install.packages()` 함수는 설치할 패키지 의존성 구조를 미리 알고 이 패키지들을 먼저 설치.
- ✓ 명령어 한 줄로 여러 패키지 설치 가능.
- ✓ 다음과 같이 패키지 이름을 문자형 벡터로 쓰면 됨.

```
> install.packages(c("ggplot2", "shiny", "knitr", "dplyr", "data.table"))
```

3.4 패키지 라이브러리 관리하기

• CRAN에서 패키지 업데이트하기

- ✓ 기본적으로 `install.packages()` 함수는 지정된 패키지의 가장 최신 버전 설치.
- ✓ 설치가 완료되면 패키지 버전은 고정된 상태로 유지.
- ✓ 버그를 수정하거나 새로운 기능을 추가하려고 패키지 업데이트 가능.
- ✓ 버전을 업데이트한 패키지에서 이전 버전에서 문제가 있던 함수를 사용하지 않기도 함.
- ✓ 이전 버전을 계속 사용하거나 패키지 설명 페이지에서 제공하는 소식(NEWS)을 읽고 업데이트 할 수도 있음.
- ✓ 패키지 업데이트 방법
 - ❖ RStudio의 패키지 창에서 Install 버튼 옆에 있는 Update 버튼 클릭.
 - ❖ 다음 함수를 사용하여 업데이트

> `update.packages()`

3.4 패키지 라이브러리 관리하기

• 온라인 저장소에서 패키지 설치하기

- ✓ 요즘에는 많은 패키지 개발자가 github에서 자신의 패키지를 공유.
 - ❖ 이슈 추적 시스템과 병합 요청 시스템 덕분에 버전 제어와 커뮤니티 개발이 수월.
- ✓ 일부 저자는 자기 것을 CRAN에 아예 공개하지 않음.
- ✓ 어떤 사람은 안정된 버전만 CRAN에 릴리스하고, 새로운 버전은 github에서 개발.
- ✓ 새로운 기능이 있거나 버그가 수정된 최신 개발 버전을 사용하려면 devtools 패키지로 온라인 저장소에서 패키지를 직접 설치 가능.
- ✓ devtools 패키지가 라이브러리에 없다면, 다음 패키지를 설치.
 - > `install.packages("devtools")`
- ✓ devtools 패키지의 `install_github()` 함수로 가장 최신 개발 버전의 ggplot2를 설치.
 - > `library(devtools)`
 - > `install_github("hadley/ggplot2")`

```
> install.packages("devtools")
Installing package into 'C:/Users/Park/Documents/R/win-library/4.0'
(as 'lib' is unspecified)
also installing the dependencies 'utf8', 'BH', 'pillar', 'pkgconfig', 'vctrs',
'askpass', 'credentials', 'sys', 'zip', 'gitcreds', 'ini', 'lazyeval',
'base64enc', 'later', 'fastmap', 'highr', 'markdown', 'xfun', 'diffobj', 'fansi',
'rematch2', 'tibble', 'clipr', 'crayon', 'curl', 'fs', 'gert', 'gh', 'glue',
'lifecycle', 'purrr', 'rappdirs', 'rprojroot', 'whisker', 'yaml', 'processx',
'R6', 'assertthat', 'digest', 'rex', 'htmltools', 'htmlwidgets', 'magrittr',
'crosstalk', 'promises', 'mime', 'openssl', 'cachem', 'prettyunits', 'xopen',
'brew', 'commonmark', 'knitr', 'Rcpp', 'stringi', 'stringr', 'xml2', 'brio',
'evaluate', 'praise', 'ps', 'waldo', 'usethis', 'callr', 'cli', 'covr', 'desc',
'DT', 'ellipsis', 'httr', 'jsonlite', 'memoise', 'pkgbuild', 'pkgload',
'rcmdcheck', 'remotes', 'rlang', 'roxygen2', 'rstudioapi', 'rversions',
'sessioninfo', 'testthat', 'withr'
```

```
> library("devtools")
필요한 패키지를 로딩중입니다: usethis
> install_github("hadley/ggplot2")
Downloading GitHub repo hadley/ggplot2@HEAD
Installing 9 packages: colorspace, viridisLite, RColorBrewer, munsell, labeling,
farver, scales, isoband, gtable
Installing packages into 'C:/Users/Park/Documents/R/win-library/4.0'
(as 'lib' is unspecified)
URL 'https://cran.rstudio.com/bin/windows/contrib/4.0/colorspace_2.0-0.zip'
Content type 'application/zip' length 2647408 bytes (2.5 MB)
downloaded 2.5 MB
```

3.4 패키지 라이브러리 관리하기

- 온라인 저장소에서 패키지 설치하기

- ✓ devtools 패키지는 github에서 일단 소스 코드를 내려 받은 후, 이를 라이브러리에서 패키지화.
- ✓ 라이브러리에 이 패키지가 이미 설치되어 있다면 원래 버전을 이 버전으로 변경.
- ✓ 이 버전을 다시 최신 CRAN 버전으로 돌려놓고 싶다면 다음 코드 재실행.

```
> install.packages("ggplot2")
```

3.4 패키지 라이브러리 관리하기

• 패키지 함수 사용하기

✓ 패키지 함수를 사용하는 방법

- ❖ 첫째, `library()` 함수를 호출하여 패키지를 불러오고, 그 안에 있는 함수를 직접 호출.
- ❖ 둘째, 전체 패키지를 환경에 연결하지 않고 해당하는 함수만 사용하려면 `package::function()` 함수 호출.
 - 예) 일부 함수는 기본 제공되는 R 패키지가 아닌 다른 패키지에 구현되어 있음.
 - 비대칭도 혹은 왜도(skewness) 함수가 그 예이며, `moments` 패키지에서 이 함수를 제공.
 - 수치형 벡터 `x`의 비대칭도를 계산하려면 먼저 이 패키지를 불러오고, 함수를 직접 호출.

```
library(moments)
skewness(x)
```
 - 전체 패키지를 불러오지 않고 다음과 같이 `::` 사용 가능.

```
moments::skewness(x)
```
 - 앞의 두 방법 모두 결과는 같지만 완전히 다른 방식으로 동작하고, 환경에 미치는 영향도 다름.
 - 첫 번째 방법(`library()` 함수)은 기호의 검색 경로를 수정하지만, 두 번째 방법(`::`)은 그렇지 않음.
 - `library(moments)`를 호출하면 패키지를 불러오면서 검색 경로에 패키지 경로를 추가. 그 후 이어지는 코드에서 패키지 함수를 사용할 수 있음.

3.4 패키지 라이브러리 관리하기

- 패키지 함수 사용하기

- ✓ sessionInfo() 함수: 현재 어떤 패키지를 활용하고 있는지 확인.

```
> sessionInfo()
```

```
R version 3.5.3 (2019-03-11)
```

```
Platform: x86_64-w64-mingw32/x64 (64-bit)
```

```
Running under: Windows >= 8 x64 (build 9200)
```

```
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
```

3.4 패키지 라이브러리 관리하기

- 패키지 함수 사용하기

attached base packages:

```
[1] stats4      stats      graphics  grDevices  utils      datasets  methods
[8] base
```

other attached packages:

```
[1] RSQLite_2.1.2
```

loaded via a namespace (and not attached):

```
[1] Rcpp_1.0.3      rstudioapi_0.10  magrittr_1.5     tidyselect_0.2.5
[5] munsell_0.5.0   bit_1.1-14       colorspace_1.4-1 R6_2.4.0
[9] rlang_0.4.0     blob_1.2.0       dplyr_0.8.3      tools_3.5.3
[13] grid_3.5.3      gtable_0.3.0     DBI_1.0.0        yaml_2.2.0
[17] digest_0.6.21   bit64_0.9-7      lazyeval_0.2.2   assertthat_0.2.1
```

3.4 패키지 라이브러리 관리하기

• 패키지 함수 사용하기

```
[21] tibble_2.1.3      mongolite_2.1.0    crayon_1.3.4      purrr_0.3.2
[25] ggplot2_3.2.1     vctrs_0.2.0       zeallot_0.1.0     rpart_4.1-15
[29] memoise_1.1.0     glue_1.3.1        compiler_3.5.3    pillar_1.4.2
[33] backports_1.1.4   scales_1.0.0      jsonlite_1.6      pkgconfig_2.0.3
```

- ✓ R 버전, 현재 첨부된(attached) 패키지 목록, 이미 로드된(loaded) 패키지 목록을 확인 가능.
- ✓ :: 을 사용했을 때는 패키지가 첨부되는 것은 아니지만 메모리에 로드됨. 이때 해당 패키지 (moments 패키지)에 있는 다른 내부 함수를 직접 호출해서 사용하는 것은 여전히 불가능.

```
> moments::skewness(c(1, 2, 3, 2, 1))
[1] 0.3436216
> sessionInfo()
R version 3.5.3 (2019-03-11)
Platform: x86_64-w64-mingw32/x64 (64-bit)
Running under: Windows >= 8 x64 (build 9200)
```

```
Matrix products: default
```


3.4 패키지 라이브러리 관리하기

- 패키지 함수 사용하기

```
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] stats4      stats      graphics  grDevices  utils      datasets  methods
```

```
[8] base
```

```
other attached packages:
```

```
[1] RSQLite_2.1.2
```

3.4 패키지 라이브러리 관리하기

- 패키지 함수 사용하기

loaded via a namespace (and not attached):

```
[1] Rcpp_1.0.3      rstudioapi_0.10  magrittr_1.5     tidyselect_0.2.5
[5] munsell_0.5.0   bit_1.1-14       colorspace_1.4-1 R6_2.4.0
[9] rlang_0.4.0     blob_1.2.0       dplyr_0.8.3      tools_3.5.3
[13] grid_3.5.3      gtable_0.3.0     DBI_1.0.0        yaml_2.2.0
[17] digest_0.6.21   bit64_0.9-7      lazyeval_0.2.2   assertthat_0.2.1
[21] tibble_2.1.3    mongolite_2.1.0  crayon_1.3.4     purrr_0.3.2
[25] ggplot2_3.2.1   vctrs_0.2.0      zeallot_0.1.0    rpart_4.1-15
[29] memoise_1.1.0   glue_1.3.1       compiler_3.5.3   pillar_1.4.2
[33] moments_0.14    backports_1.1.4  scales_1.0.0     jsonlite_1.6
[37] pkgconfig_2.0.3
```

3.4 패키지 라이브러리 관리하기

- 패키지 함수 사용하기

- ✓ 앞 결과에서 볼 수 있듯이, moments 패키지를 호출했지만 첨부되지 않음.
- ✓ library(moments) 함수를 호출하여 패키지를 첨부

```
> library(moments)
```

```
> sessionInfo()
```

```
R version 3.5.3 (2019-03-11)
```

```
Platform: x86_64-w64-mingw32/x64 (64-bit)
```

```
Running under: Windows >= 8 x64 (build 9200)
```

```
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
```

3.4 패키지 라이브러리 관리하기

- 패키지 함수 사용하기

```
attached base packages:
```

```
[1] stats4      stats      graphics  grDevices  utils      datasets  methods  
[8] base
```

```
other attached packages:
```

```
[1] moments_0.14  RSQLite_2.1.2
```

3.4 패키지 라이브러리 관리하기

- 패키지 함수 사용하기

loaded via a namespace (and not attached):

```
[1] Rcpp_1.0.3      rstudioapi_0.10  magrittr_1.5     tidyselect_0.2.5
[5] munsell_0.5.0   bit_1.1-14       colorspace_1.4-1 R6_2.4.0
[9] rlang_0.4.0     blob_1.2.0       dplyr_0.8.3      tools_3.5.3
[13] grid_3.5.3      gtable_0.3.0     DBI_1.0.0        yaml_2.2.0
[17] digest_0.6.21   bit64_0.9-7      lazyeval_0.2.2   assertthat_0.2.1
[21] tibble_2.1.3    mongolite_2.1.0  crayon_1.3.4     purrr_0.3.2
[25] ggplot2_3.2.1   vctrs_0.2.0      zeallot_0.1.0    rpart_4.1-15
[29] memoise_1.1.0   glue_1.3.1       compiler_3.5.3   pillar_1.4.2
[33] backports_1.1.4 scales_1.0.0     jsonlite_1.6     pkgconfig_2.0.3
```

3.4 패키지 라이브러리 관리하기

• 패키지 함수 사용하기

- ✓ 이젠 skewness() 함수처럼, moments 패키지 안에 존재하는 다른 함수들도 사용 가능.
- ✓ 첨부된 패키지 목록을 찾아보는 더 쉬운 방법은 search() 함수를 사용하는 것.

```
> search()
```

```
[1] ".GlobalEnv"      "package:moments"  "tools:rstudio"    "package:stats"
[5] "package:graphics" "package:grDevices" "package:utils"     "package:datasets"
[9] "package:methods" "Autoloads"        "package:base"
```

- ✓ 이 함수는 현재 환경에 있는 모든 기호의 검색 경로를 보여줌.
- ✓ skewness 함수를 호출하면,
 - ❖ 먼저 현재 환경에서 skewness 기호 검색.
 - ❖ package:moments로 이동 후, 이 기호가 있는지 확인.
 - ❖ 패키지가 첨부되지 않았다면 이 기호를 찾을 수 없게 되고, 결국 오류 발생.

3.4 패키지 라이브러리 관리하기

- 패키지 함수 사용하기

- ✓ require() 함수:

- ❖ 어떤 패키지를 첨부한다는 점에서 library()와 비슷한 역할.
- ❖ '패키지를 성공적으로 첨부했는지 나타내는 논리형 변수 값을 반환한다'는 점에서 차이.

```
> loaded <- require(moments)
> loaded
[1] TRUE
```
- ❖ 이 기능을 이용하여 다음 코드를 만들 수 있음.
- ❖ moments 패키지가 설치되어 있다면 바로 첨부하고, 아직 설치되지 않았다면 설치한 후에 첨부.

```
> if (!require(moments)) {
+   install.packages("moments")
+   library(moments)
+ }
```

3.4 패키지 라이브러리 관리하기

• 패키지 함수 사용하기

✓ require() 함수:

❖ 보통은 다음과 같이 사용.

```
> require(moments)
```

❖ library() 함수와 동일하게 작업하지만, 패키지를 불러오는 것을 실패하더라도 그냥 넘어가는 단점 존재

```
> require(testPkg)
```

```
Loading required package: testPkg
```

```
Warning message:
```

```
In library(package, lib.loc = lib.loc, character.only = TRUE, logical.return = TRUE, :
```

```
  there is no package called 'testPkg'
```

❖ 첨부하고자 하는 패키지가 아직 설치되지 않았거나 아예 없더라도(패키지 이름을 잘못 입력했을 때) library 함수가 '오류'를 발생시키는 것과 달리 require 함수는 '경고'만 발생.

```
> library(testPkg)
```

```
Error in library(testPkg) : 'testPkg'이라고 불리는 패키지가 없습니다
```

❖ “빨리 실패하라”의 원리 → library()

3.4 패키지 라이브러리 관리하기

• 마스킹과 이름 충돌

- ✓ 새 R 세션을 시작하면 기본(base), 통계(stats), 그래픽(graphics) 등 기본 패키지가 자동 첨부.
- ✓ 이 패키지가 '이미 첨부된 상태'이므로 `base::mean()`이나 `stats::median()`처럼 `::`을 사용하거나 `base`, `stats` 패키지를 다시 첨부할 필요 없이, `mean()`과 `median()`을 '직접' 사용하여 수치형 벡터의 평균값과 중간 값 바로 계산 가능.
- ✓ 실제로 자동으로 첨부된 패키지를 이용하여 함수 수천 개 즉시 사용 가능.
- ✓ 각 패키지는 특정 목적에 맞는 수많은 함수를 제공.
- ✓ 두 패키지에 있는 함수가 서로 충돌할 수 있음.

3.4 패키지 라이브러리 관리하기

- **마스킹과 이름 충돌**

- ✓ 예를 들어 두 패키지 A와 B 모두 X 함수를 갖고 있다 가정.
- ✓ A를 첨부한 후 B를 첨부하면 "A::X 함수는 B::X 함수로 마스크 처리"됨.
- ✓ 즉, A를 첨부하고 X()를 호출하면 A에 있는 X 함수가 호출.
- ✓ B를 첨부하고 X()를 호출하면 B에 있는 X 함수가 호출.
- ✓ 이 메커니즘을 '마스킹'이라고 함.

3.4 패키지 라이브러리 관리하기

- **마스킹과 이름 충돌**

- ✓ dplyr은 테이블 형식의 데이터를 좀 더 쉽게 조작할 수 있는 함수의 집합을 제공.
- ✓ 이 패키지를 첨부하면 기존 함수 가운데 어떤 것이 패키지 함수로 마스킹되는지 메시지 출력.

```
> library(dplyr)
```

다음의 패키지를 부착합니다: 'dplyr'

The following objects are masked from 'package:stats':

filter, lag

The following objects are masked from 'package:base':

intersect, setdiff, setequal, union

3.4 패키지 라이브러리 관리하기

- **마스킹과 이름 충돌**

- ✓ 다행히 dplyr에서 제공하는 이러한 함수는 의미나 용도가 크게 달라지지 않고 오히려 일반화됨.
- ✓ 이러한 함수들은 마스킹된 버전과 함께 호환 가능.
- ✓ 마스킹된 함수를 더 이상 사용할 수 없다고 걱정하지 않아도 됨.
- ✓ 기본 함수들을 마스킹하는 패키지 함수는 이것을 대체하기보다는 거의 기존 함수를 일반화함.
- ✓ 동일한 이름을 공유하는 함수를 가진 패키지 2개를 사용해야 한다면 두 패키지를 첨부하는 대신, 다음과 같이 두 패키지에서 필요한 함수를 추출하는 편이 더 나음.

```
fun1 <- package1::some_function
```

```
fun2 <- package2::some_function
```

3.4 패키지 라이브러리 관리하기

- 마스킹과 이름 충돌

- ✓ 어떤 패키지를 첨부했다가 다시 취소하고 싶다면 `unloadNamespace()` 함수를 사용.
- ✓ 예를 들어 `moments` 패키지를 첨부했다 다음과 같이 다시 취소 가능.

```
> unloadNamespace("moments")
```

- ✓ 패키지를 취소하면 패키지 함수는 더 이상 사용 불가.

```
> skewness(c(1, 2, 3, 2, 1))
```

```
Error in skewness(c(1, 2, 3, 2, 1)) : could not find function "skewness"
```

- ✓ 물론 여전히 `::`을 사용하여 함수 호출 가능.

```
> moments::skewness(c(1, 2, 3, 2, 1))
```

```
[1] 0.3436216
```

3.4 패키지 라이브러리 관리하기

- 패키지 설치 여부 확인하기

- ✓ `installed.packages()` 함수: 이미 설치된 패키지 정보 확인.
- ✓ 열 16개로 구성된 행렬에 다양한 정보를 제공.

```
> pkgs <- installed.packages()
```

```
> colnames(pkgs)
```

```
[1] "Package"
```

```
"LibPath"
```

```
"Version"
```

```
[4] "Priority"
```

```
"Depends"
```

```
"Imports"
```

```
[7] "LinkingTo"
```

```
"Suggests"
```

```
"Enhances"
```

```
[10] "License"
```

```
"License_is_FOSS"
```

```
"License_restricts_use"
```

```
[13] "OS_type"
```

```
"MD5sum"
```

```
"NeedsCompilation"
```

```
[16] "Built"
```

3.4 패키지 라이브러리 관리하기

- 패키지 설치 여부 확인하기

- ✓ 패키지가 이미 설치되었는지 확인할 때 유용.

```
> c("moments", "testPkg") %in% installed.packages()[, "Package"]  
[1] TRUE FALSE
```

- ✓ 가끔 패키지의 버전 정보가 필요할 때가 있음.

```
> installed.packages()["moments", "Version"]  
[1] "0.14"
```

- ✓ 패키지 버전을 확인하는 더 간단한 방법도 있음.

```
> packageVersion("moments")  
[1] '0.14'
```

3.4 패키지 라이브러리 관리하기

- 패키지 설치 여부 확인하기

- ✓ 다음과 같이 패키지 버전이 주어진 버전보다 더 최근 버전인지 확인 가능.

```
> packageVersion("moments") >= package_version("0.14")  
[1] TRUE
```

- ✓ 비교를 위해 문자열을 직접 사용해도 됨.

```
> packageVersion("moments") >= "0.14"  
[1] TRUE
```


3.5 마치며

- 마치며

- ✓ 작업 디렉터리 개념과 이를 다루는 기법을 학습.
- ✓ 작업 환경을 검사하고 전역 옵션을 수정하고 패키지 라이브러리를 관리하는 기능을 살펴봄.
- ✓ 이제 작업 공간을 관리하는 기본 지식을 갖춘 것.