SVH Neurology Code, Stats & AI (with R)

임상 데이터 과학

목차

서문

이 자료는 R 언어를 사용한 임상 데이터 과학(Clinical Data Science) 교육 자료를 정리하여 공유하기 위해 만들었습니다.

무엇보다 R 언어는 통계학의 세계 공용어로 자리잡았으며, 자신의 연구 결과를 발표하기 위해 또는 새로운 정보를 얻기 위해 논문을 읽어야 하는 의료인에게는 통계는 늘 따라다니는 존재입니다.

R 언어 세계는 넓어, 넓은 세계에서 처음 시작하는 사람들은 한참을 헤매고 나서야 길을 찾는 경우가 많습니다. 물론 그런 행위도 가치가 없는 것은 아니지만, 바쁜 의료인에게는 지레 포기하게 만드는 원인이 될 수 있습니다. 따라서 바로 사용할 수 있고, 늘 찾아볼 수 있도록 정리하는 것이 이 사이트의 목적입니다.

요즘에는 AI coding assistant가 발전하여 이런 작업을 하기에도 편리해졌습니다. 어떤 기초 개념을 소개하는 예시 등을 만들 때는 이런 도구들이 큰 도움이 되었으며, 이 사이트의 콘텐츠 역시 그런 결과물을 다수 포함시켰습니다.

포함시킬 내용

임상 데이터 과학 교육 자료는 다음과 같은 내용을 포함시켰거나 앞으로 그럴 계획입니다. 데이터 과학에서 핵심적 역할을 하는 R, Python, SQL(DuckDB) 등 3개의 언어를 중심으로 내용을 채우려고 하는데, 하나의 사이트에서 다루는 것은 너무 많은 내용이 될 것 같아 분리하여 준비하려고 합니다.

1. 기초 R 언어
2. Tidyverse 패키지
    - 데이터 전처리와 가공
3. ggplot2 패키지를 사용한 데이터 시각화
4. 통계학
    - 가설 검정

    - 머신러닝

5. 데이터 과학 노트북

    - Jupyter Notebook
    - Quarto
    - 컴퓨테이셔널 재현가능성(computational reproducibility)을 보장하는 논문 작성 방법
    - 인공지능과 함께 하는 모던 텍스트 편집기
        - Cursor: The AI Code Editor
        - VS Code와 GitHub Copilot
        - RStudio와 OpenAI Chat
    - 인공지능과 함께 데이터 과학 노트북 사용하여 (논문, 보고서, 프리젠테이션 등) 글쓰기

6. RAG(Retrieval-Augmented Generation, 추출-증강 생성)

    - 이 사이트를 지식 기반 데이터베이스로 사용하기

- 파이썬 언어: SVH Neurology Code, Stats & AI (with Python)

(R) 프로그래밍 언어 학습 방법에 대한 권고

이제 본격적으로 R 프로그래밍 언어를 학습할 것이다. 처음 시작하는 사용자들을 위해 다음과 같은 조언을 하고 싶다.

- **코드는 읽기만 해서는 내 것이 되지 않는다.**

  프로그래밍 언어도 일종의 언어라서 읽기만 해서는 자기 것으로 만들 수 없다. "항상 손으로 써보고, 실제로 입력해서 결과를 확인"하는 습관을 들이는 것이 좋다. 나는 컴퓨터 코드는 그림과 텍스트 중간에 있다는 생각을 많이 한다. 텍스트처럼 읽히기는 하지만 실제로 그런 문장을 써 보면 잘 안된다. 이런 것은 누구가 그러하다. 극복하는 방법은 끊임없는 연습이다.

- **코드는 할당 연산자를 기준으로 오른쪽을 먼저 읽고, 안에서 밖으로 읽는다.**

  다음을 간단한 R 함수를 사용하여 분산을 계산한 예이다. 분산의 공식은 다음과 같은데, 이걸 R 코드로 작성해 보면 다음과 같다.

$$\frac{\sum (x - \bar{x})^2}{n - 1}$$

```r
x <- c(58, 78, 77, 65, 87, 66, 45, 66, 100)
var_x <- sum((x - mean(x))^2) / (length(x) - 1)
# 간단히 var(x)로도 계산 가능하다.
var_x
```

```
[1] 264
```

이 경우 할당 연산자 <- 오른쪽을 먼저 읽는데, 이처럼(이렇게 중첩시키는 코드는 좋은 코드가 아니다) 중첩된 코드는 안쪽에서 밖으로 읽는다. 이 경우에는 `x - mean(x)`를 마음 속으로 읽고, 그 다음 이것을 제곱하고, 제곱한 것들은 `sum()`하고, 이것은 `length(x) - 1` 값으로 나눈다고 읽는다. 그렇게 계산된 값을 `var_x` 변수에 할당한다.

- **때론 중간 과정을 풀어서 차례로 확인할 필요가 있다.**

  R 언어에서 계산은 하나의 값이 아니라 값들을 모은 벡터(vector)를 대상으로 하는 경우가 많아 처음 읽을 때 직관적으로 파악되지 않을 수 있다. 이런 경우에는 중간 과정을 풀어서 읽어 보자. 앞에 본 코드를 풀어서 실행해 보면 이해가 더 잘 될 수도 있다.

```r
x <- c(58, 78, 77, 65, 87, 66, 45, 66, 100)
dev_x <- x - mean(x) # 편차 계산
dev_x
x_sum_square <- sum(dev_x^2) # 편차 제곱의 합
x_sum_square
var_x <- x_sum_square / (length(x) - 1) # 편차 제공의 합을 n-1로 나눔
var_x
```

- **코드가 눈에 들어오기 시작하면 도움말(help page)을 자주 볼 필요가 있다. 하지만 또 너무 자세히 매달릴 필요가 없다.**

  R 도움말 페이지는 간결하고 정리가 잘 되어 있지만, 초보자들은 처음에 읽기 어려울 수 있다. 하지만 어느 정도 공부하고 나면 자주 찾아보는 습관을 들이는 것이 좋다.

- **마스터하기 보다 끊임없는 학습이 필요하다.**

  R 언어의 세계는 방대해서 실제로 이것을 마스터하기는 불가능하다. 그래서 지엽적인 문제에 너무 집착하면 앞으로 나갈 수가 없다. 어느 정도 이해하고 앞으로 나가고, 어떤 문제를 만나거나 잘 이해가 안 되는 부분이 있으면 좀 더 깊게 들어 가는 방법을 취한다.

- **니즈(needs)을 찾아라.**

  니즈는 우리에게 앞으로 나갈 동기를 부여한다. 나는 이전에 **오피스 프로그램 없이 살아가기**가 컴퓨터 공부의 중요한 니즈였다.

  난 주변 의료인들에게 페이지 아래에서 소개하는 Quarto를 사용해 볼 것을 권고한다. 논문, 저술, 프레젠테이션, 블로그, 책 등 생각하는 모든 것을 같은 원리로 만들 수 있다. R이든 Python이든 이것만 있어도 오피스 프로그램 없이도 살아가는 것이 가능하고, 그래서 늘 사용하고 배우는 기회를 가질 수 있다.

- **프로그래밍 언어는 단순한 도구가 아니다. 언어를 사랑하자.**

  우린 항상 뭔가를 해결해야 하고 항상 바쁘다. 인공지능 시대에 프로그래밍 언어를 배워야 살아남을 것 같고 그렇지 않으면 뒷처지는 느낌을 가지게 된다. 그래서 프로그래밍 언어를 배울 때도 숙제를 하듯 할 수 있다. 그럴 필요가 없다고 생각한다. 마음을 열고 이 사랑스런 존재를 대하면 좋을 것 같다. 새로운 세상을 만들어 내는 이 위대한 존재를 어떻게 사랑하지 않을 수 있을까?

  프로그래밍 언어 학습은 어떤 앱 사용법을 배우는 것과 완전히 다르다. 언어는 쓰고 버리는 것이 아니라 우리 뇌 안에서 영원히 존재할 것이기 때문이다.

  언어마다 그 언어가 추상화하여 세상을 표현하는 방법과 세상의 문제를 해결하는 방법이 있다. 그리고 그 세상을 확장해 나가가는 사람들이 있다는 것을 알 필요가 있다. 그런 사람들을 존중하고 그 열정을 배우는 것이 프로그래밍 언어 학습의 중요한 부분이라 생각한다.

참고한 주요 자료

- 통계학

  1. 기초 통계학의 숨은 원리 이해하기, 김권현, 경문사
  2. Statistical Inference via Data Science (2e)
  3. Introduction to Modern Statistics (2e)
  4. Statistical Thinking for the 21st Century

- R 언어

  1. Advanced R
     - R 프로그래밍 언어의 프로그래밍 언어로서의 특징을 자세히 설명하는 해들리 위캄의 명저로 웹사이트에서도 누구나 읽을 수 있다.

2. R for Data Science (2e)
    - R 언어를 사용하여 데이터 과학을 수행하는 전반적인 과정을 다룬다.
    - The tidy tools manifesto에 따른 일관된 접근법을 취한다.
3. Fundamentals of Data Visualization
    - 아름다운 데이터 시각화 방법을 설명한다.
4. R로 하는 빅 데이터분석 (제3판), 김권현 저, 숨은원리

사이트 제작

이 사이트는 **오픈 소스 과학기술 출판 시스템** Quarto를 사용하여 만들었습니다. **재현가능한 방식 (computational reproducibility)**으로, 논문쓰기와 같은 데이터 작업을 하는 데 최적의 환경을 제공합니다. 마크다운 텍스트와 코드를 적절히 혼합하여 article, book, website, presentation 등을 만들 수 있습니다.

또 웹브라우저에서 바로 코드를 연습할 수 있게 Quarto Live라는 WebAssembly 기능을 사용하여 코드 연습이 많이 필요한 곳에 사용하였습니다.

- 사용된 R 버전: 4.5.0
- Quarto 버전: 1.7.31

Part I

I. R 언어의 기초

1 R과 RStudio 설치

1.1 R 프로그래밍 언어란?

- R is a free software environment for statistical computing and graphics.
    - 즉, 통계 분석과 그래픽을 위한 프로그래밍 언어이다.
    - 요즈음은 Python과 더불어 데이터 과학(data science) 분야에서 가장 많이 사용되는 언어로 꼽힌다.

그림 1.1: R 로고

1.2 R 설치

- 오픈 소스로 누구나 무료로 사용할 수 있다.
- 구글에서 “CRAN”으로 검색하여, The Comprehensive R Archive Network에 가서 컴퓨터 시스템에 맞는 것을 다운로드하여 설치한다. 설치할 때 특별히 고려할 내용은 없다.
    - 윈도우, 맥, 리눅스 버전 모두 제공된다.
    - R의 핵심을 이루는 부분은 의외로 아주 작다(필요한 것들은 packages를 사용하여 추가하여 사용한다).
- R 언어는 보통 1년에 2 ~ 4회 업데이트 되는데, 업데이트 하려면 위 과정을 반복하면 된다.

1.2.1 R 설치후 확인

R을 설치하고 실행하면 그림 1.2과 같이 실행된다(실제는 안내문이 한글로 보일 것이다).

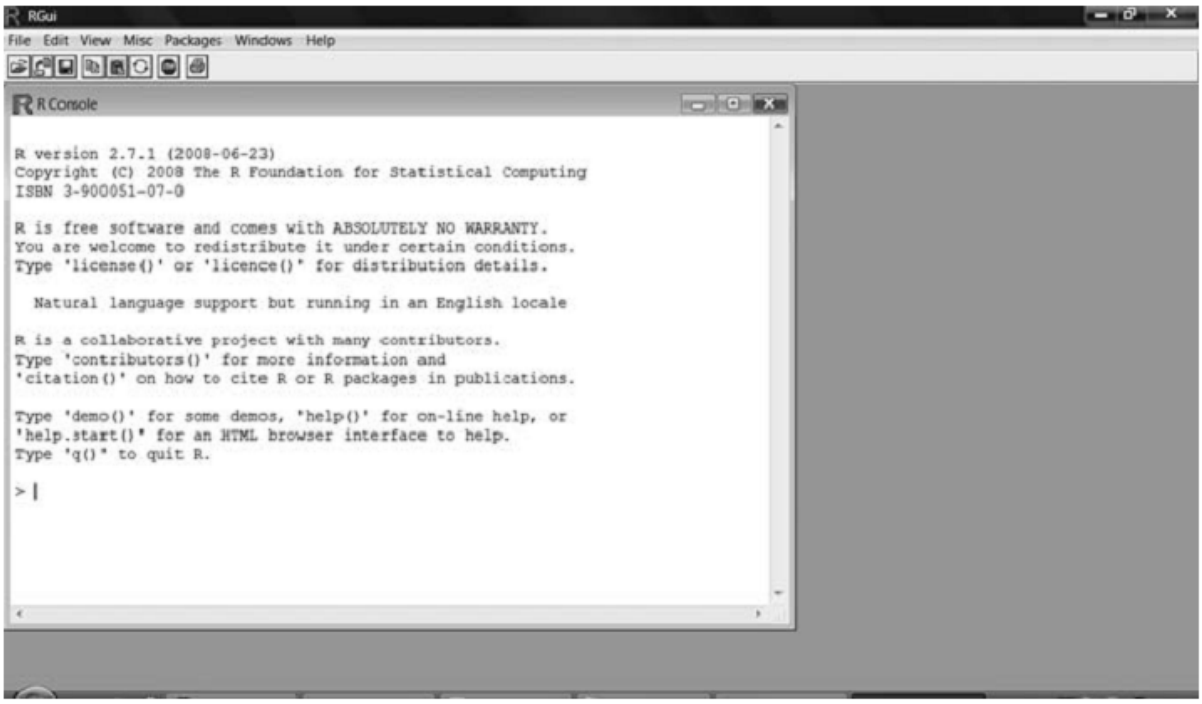


그림 1.2: 처음 R을 설치하고 실행한 화면(R 콘솔)

끝에 >을 볼 수 있다. 이것은 사용자의 명령을 기다린다는 뜻으로 **"프롬프트(prompt)"**라고 한다. 여기에 다음과 입력하여 엔터(Enter) 키를 치면 실행된다.

```
> 3 + 5
```

```
[1] 8
```

이와 같이 R은 기본적으로 사용자가 명령을 입력하면 R 해석기가 그것을 해석하고 실행한 다음 결과를 다시 출력해 주는 것을 반복하면서 실행된다. 이 과정을 Read-Evaluate-Print Loop라고 하고 REPL(레플)이라고 부른다. 또 이렇게 사용자가 R와 대화하듯이 작업하는 것을 대화형 모드(interactive mode)라고 부른다(나중에 R 명령들을 하나의 파일에 모아서 한꺼번에 실행시킬 수도 있는데 이것을 스크립트 모드(script mode)라고 한다).

출력에서 앞에 보이는 [1]은 첫 번째 위치라는 것을 의미하는데 일단 무시해도 된다.

그런데, R 콘솔만 사용하여 일하는 경우는 거의 없다. R 언어를 중심에 두고, 사용자를 위한 여러 편이 기능들을 갖춰 놓은 RStudio라는 프로그램을 사용하여 R을 사용하는 경우가 많다.

1.3 RStudio IDE(통합개발환경)

- RStudio는 R 언어 사용의 편이를 제공하여 위해서 Posit.co라는 미국 회사가 개발하여 무료로 (상용도 있지만) 배포하는 프로그램이다.
- 구글에서 "RStudio"라고 검색하여 "RStudio Desktop" 다운로드 사이트를 찾는다.

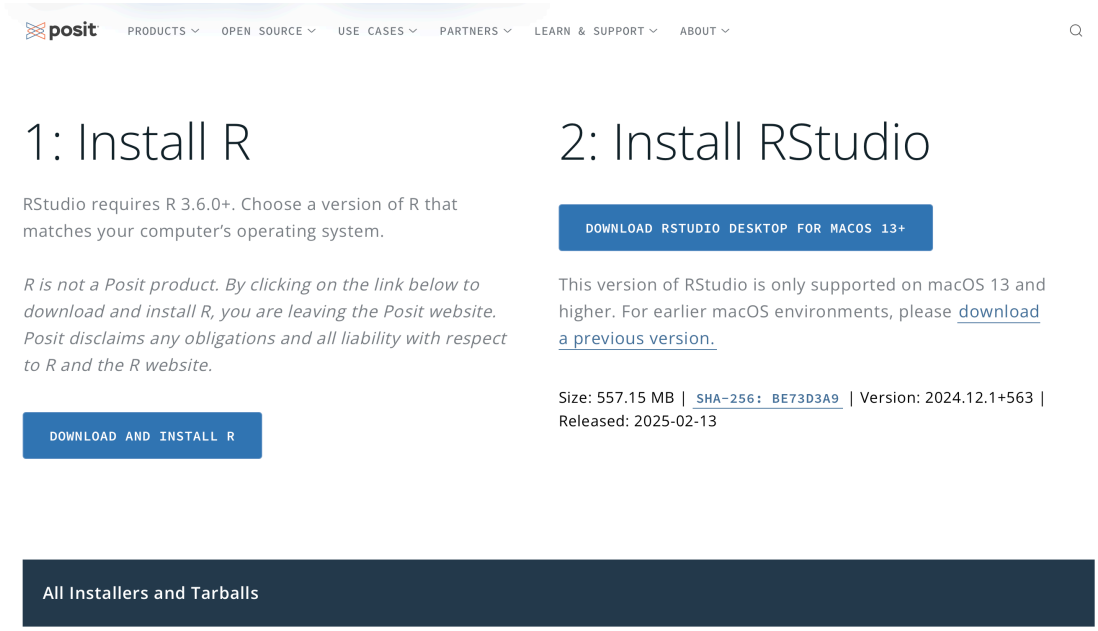


그림 1.3: 앞에서 R은 설치했기 때문에 2번 Install RStudio를 클릭하여 자기 시스템에 맞는 것을 다운로드해서 설치한다.

RStudio는 R 언어로 가지고 코드 작성, 패키지 제작, 각종 문서 만들기 등 다양한 일을 하는데 편리한 기능을 제공한다. 핵심은 코드를 작성하는 텍스트 에디터(text editor)이고 이외에도 아주 다양한 기능을 제공한다. 이런 종류의 소프트웨어를 통합개발환경(integrated development environment)이라고 한다.

1.3.1 RStudio 실행

설치하고 RStudio를 실행하면, 윈도우인 경우 어떤 R을 사용할지 묻는 창이 보일 수도 있다.

- `64bit`용 R을 사용한다고 선택하고 넘어가면 된다.

RStudio를 실행하면 그림 1.4와 같이 실행된다.

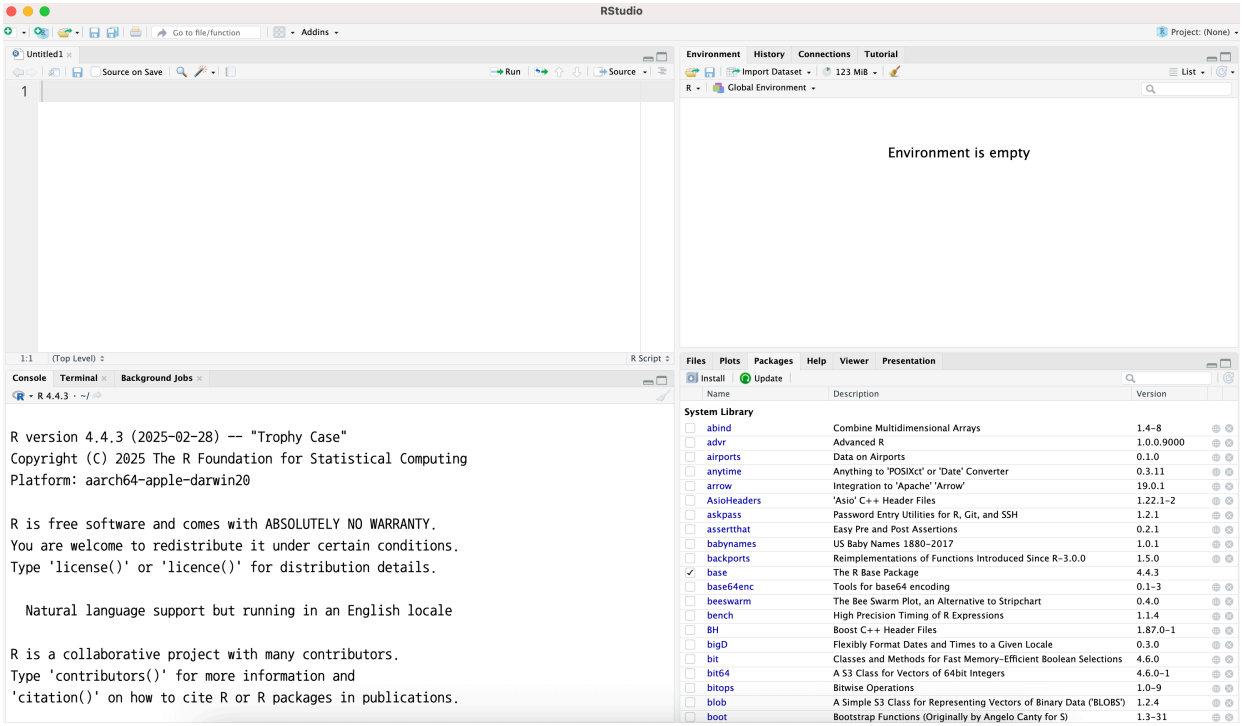


그림 1.4: RStudio는 4개의 창으로 나눠 화면을 사용한다(배치를 사용자가 쉽게 바꿀 수도 있다)

그림 1.4 같이 된 경우 - 왼쪽 아래가 앞에서 본 R 콘솔이다. - 왼쪽 위가 코드를 작성하는 텍스트 에디터이다. - 오른쪽에 현재 디렉터리의 파일을 관리하는 “Files”, 플롯을 보여주는 “Plots” 창, 패키지를 관리하는 “Packages” 창, 현재 R 환경에 있는 객체들을 보여주는 “Environment” 창 등이 있다.

R 콘솔에서 다음과 같이 코드를 입력하고 엔터(Enter) 키를 치면 실행해 보자. 이것은 표준 정규 분포(평균이 0이고 표준편차가 1인 정규분포)를 따르는 1000개의 난수 값을 생성한 다음, 그 값들의 분포를 히스토그램으로 만들어 본 예이다. 플롯은 “Plots” 창에서 보여준다.

```
> hist(rnorm(1000))
```

그림 1.5: 난수에 대한 히스토그램 만들어 본 예

1.3.2 RStudio에서 프로젝트(Project) 만들어 사용하기

보통 우리가 데이터 분석이나 논문 작업을 할 때, 관련된 데이터 파일, 논문, 그림 등을 하나의 폴더에 모아서 관리하는 경우가 많다. 여기서 말하는 프로젝트란 이런 폴더를 의미한다. RStudio는 이런 폴더를 쉽게 관리할 수 있는 기능을 제공한다.

여기서 R 언어와 관련된 핵심 개념 하나를 알 필요가 있다. R이 실행되어 종료될 때까지를 R 세션이라고 한다. 또 R은 현재 실행된 디렉터리를 기준으로 어떤 일을 한다. 그러니까 실행된 디렉터리를 기준으로 엑셀에 있는 데이터를 읽을 수 있고, 이 실행된 디렉터리를 기준으로 데이터를 저장한다. 이처럼 R이 실행된 디렉터리로, 모든 일을 할 때 그 기준이 되는 디렉터리를 **작업 디렉터리(working directory)**라고 한다. `getwd()`라고 하면 현재 R 세션의 작업 디렉터리를 출력한다(보통 처음 시작할 때는 홈 디렉터리가 된다).

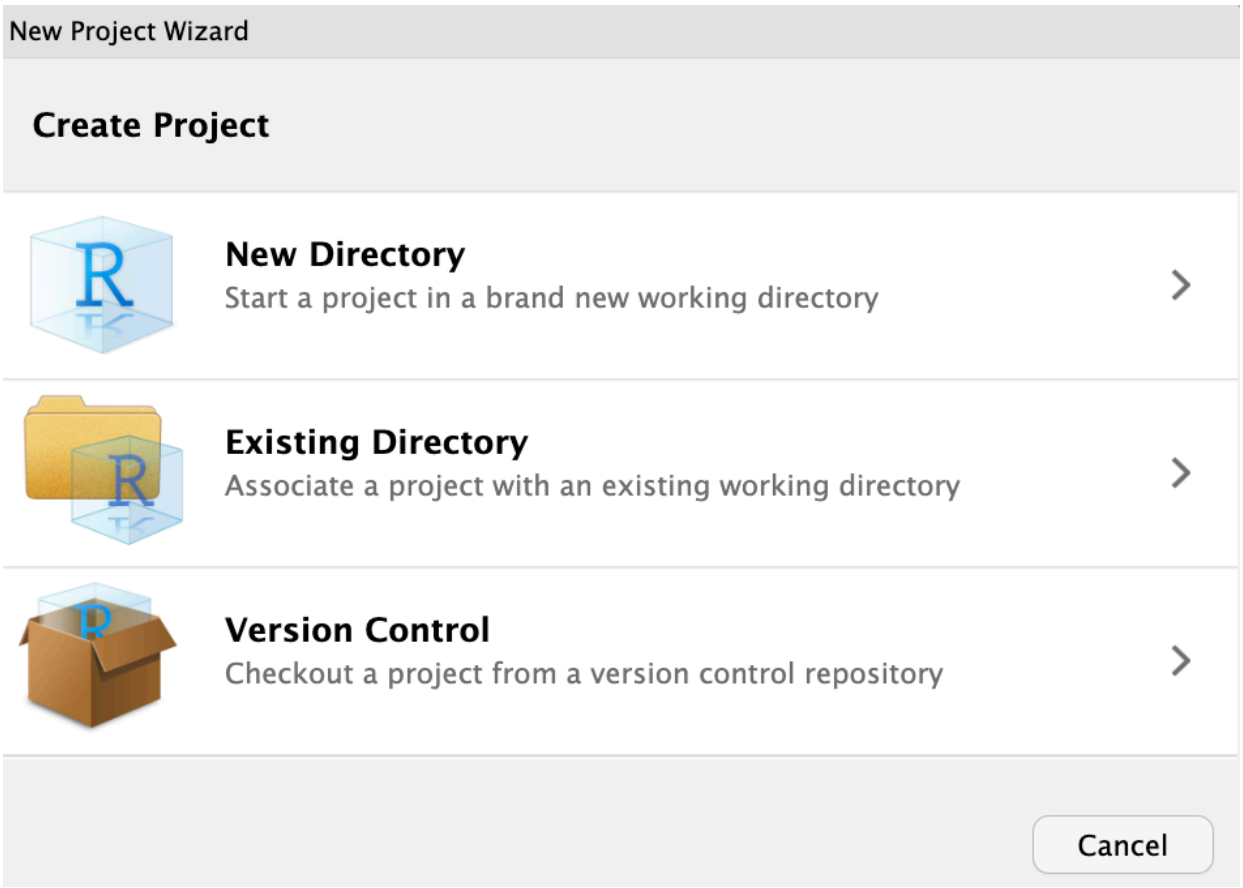
```
getwd()
```

```
[1] "/Users/seokbumko/Learning/Education/cds-R"
```

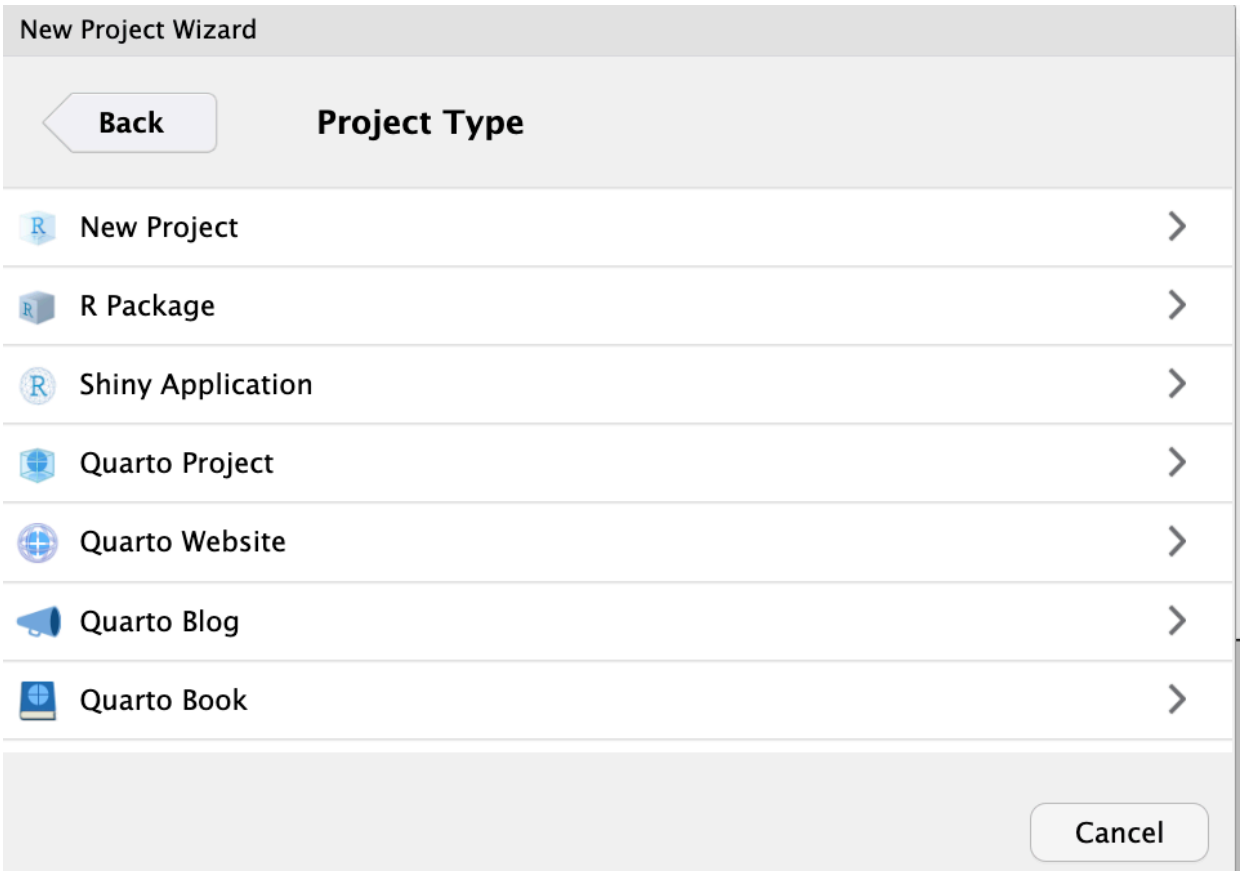
(물론 이런 개념을 이해하는 것이 중요할 수도 있지만…) RStudio 프로젝트 기능을 사용하면 굳이 이것에 신경쓰지 않아도 된다. 뒤에서 보면 알겠지만 RStudio가 자동으로 작업 디렉터리를 잡아 주기 때문이다.

- RStudio 오른쪽 위를 보면 파란색의 Project라는 아이콘을 클릭한다.

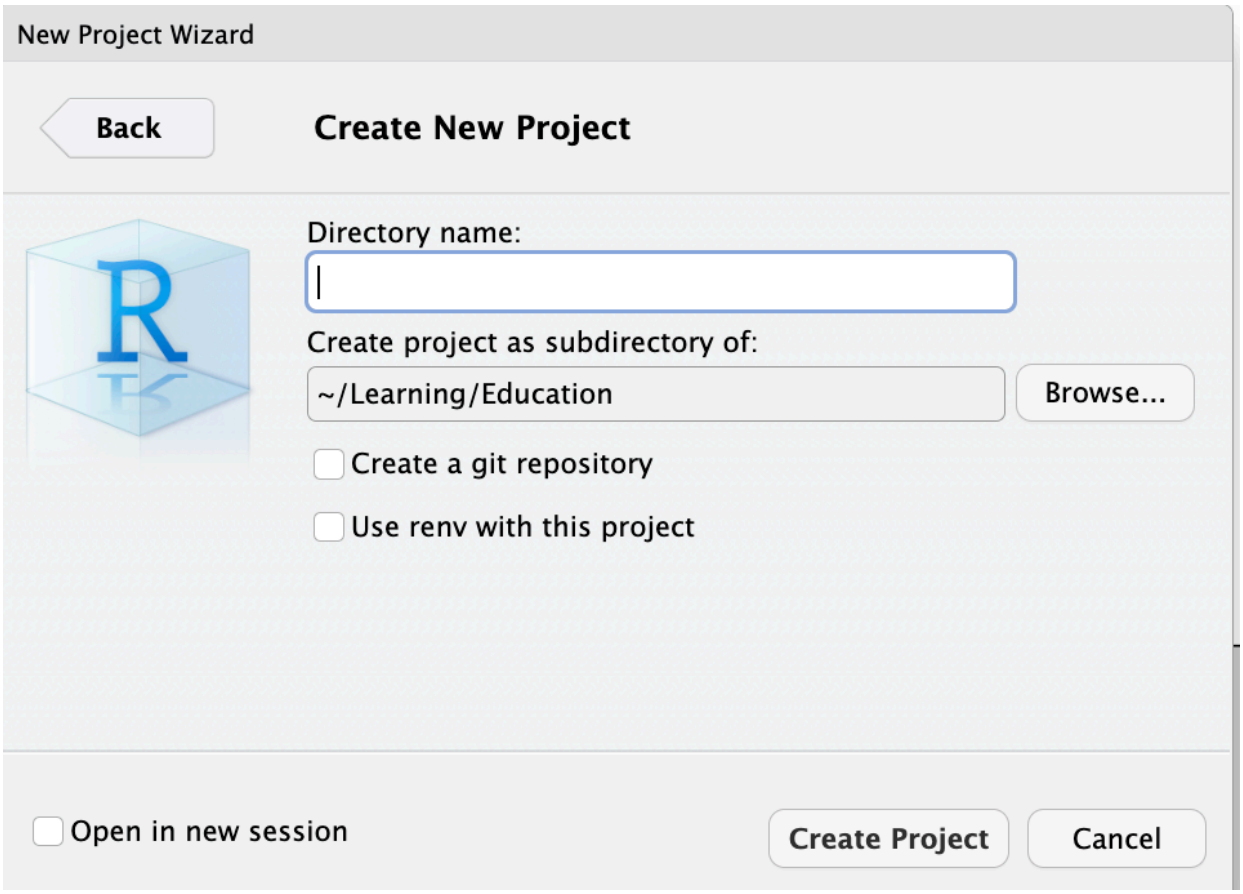
- "New Project"를 클릭한다. 여기에서 "New Directory"를 선택한다.



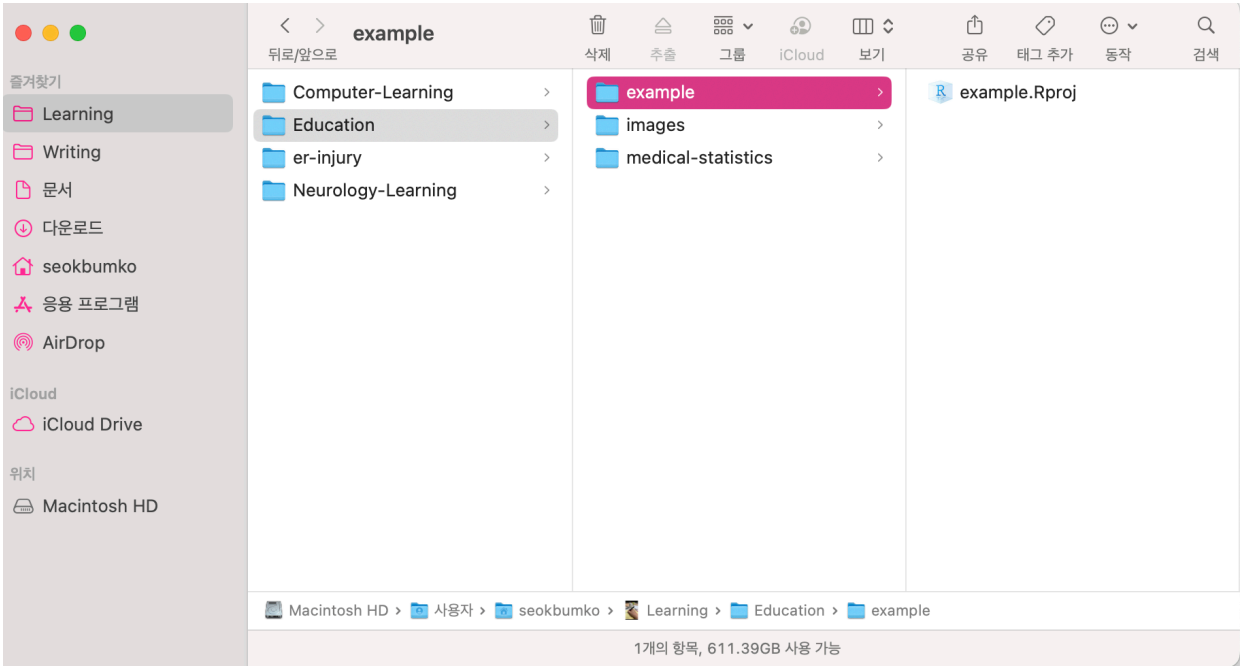
- 다음은 프로젝트 타입을 선택하는 창이 뜬다. "New Project"를 선택한다.



- 다음은 이 프로젝트를 놓을 위치와 이름을 지정하는 창이 열린다. 중간에 “Browse…” 버튼을 클릭하여 폴더를 놓을 위치를 잡은 다음 이름을 입력하면 된다. 그리고 나서 “Create Project”를 클릭한다.



- 그 다음은 RStudio/R이 지정한 디렉터리를 작업 디렉터리로 새로 설정하면서 RStudio가 새로 실행된다. 그리고 컴퓨터에 해당 폴더에 이 프로젝트를 대표하는 아이콘 파일이 하나 생성된다. 위에서 `example`이라는 프로젝트 이름을 줬기 때문에 해당 위치에 `example`이라는 폴더와 그 안에 `example.Rproj`라는 아이콘 파일이 만들어진다.



** 이 아이콘 파일은 프로젝트를 대표하는 파일이다. 윈도우 탐색기에서 이것을 클릭하면, 이 파일이 있는 곳을 작업 디렉터리로 설정하면서 RStudio가 실행된다.

1.3.3 RStudio에서 프로젝트 열기

RStudio에서 특정 프로젝트는 여러 방법으로 열 수 있다.

1. 윈도우 탐색기, 맥 파인더에서 폴더를 찾아서 앞에서 설명한 아이콘 파일을 클릭한다.
2. RStudio에서 오른쪽 위 Project 아이콘을 클릭하면 작업하던 디렉터리들이 보인다. 여기서 원하는 디렉터리를 클릭한다.
3. 이 리스트에서 보이지 않는다면 여기서 “Open Project…”를 선택하여 앞에서 설명한 아이콘 파일을 클릭하면 된다.

1.4 파일의 위치(Path, 경로)

컴퓨터에서 어떤 파일은 전체 파일시스템의 어떤 폴더 안에 존재한다. 윈도우나 맥이나 이런 파일시스템은 거꾸로 된 나무(tree) 모양으로 구성된다. 가장 상단에 뿌리(root)가 있고, 그 안에 여러 폴더가

있고, 그 폴더 안에 여러 폴더가 있는 식이다. 이런 시스템에서 파일의 위치를 해당 파일에 대한 경로(path)라고 한다.

폴더(folder)나 디렉터리(directory)는 같은 뜻을 가진 용어이다.

다음과 같이 폴더가 구성되어 있다고 해 보자. `C:` 폴더 안에 `ABC`, `DEF`라는 폴더가 있고, `ABC` 폴더 안에 `Stroke`, `Peripheral`, `Dementia` 등의 폴더가 있고, `Stroke` 폴더 안에 `Embolic`이라는 폴더가 있고 그 안에 `embolic.data`라는 폴더가 있는 구조이다.

```
- C:
  - ABC
    - Stroke
      - Embolic
        - raw-data
          - embolic.data
        - analysis.R
    - Peripheral
    - Dementia
    - Movements
    - Headache
    - Epilepsy
    - NeuroOpthalmo-Otology
  - DEF
    - Hello
    - World
```

이 경우 `emolic.data`를 루트(root) `C:`에서 시작하여 찾아가는 경로는 `C:\ABC\Stroke\Embolic\raw-data\ embolic.data`가 된다.

- 이처럼 윈도우에서는
  - 역슬래쉬 `\`를 사용하여 디렉터리를 구분한다.
  - 또한 `C:`나 `D:`와 같은 여러 개의 루트 디레터리를 가진다.
- 맥오에스(macOS)에서 `/ABC/Stroke/Embolic/raw-data/embolic.data`라고 사용한다.
  - 맥에에스는 루트의 이름은 `/`로 표시한다.
  - 디렉터리 간의 경계는 `/`로 구분한다.

이와 같이 **루트에서 시작하여** 어떤 파일을 표기하는 경우를 **절대 경로(absolute path)**라고 한다

이 경우는 다르게 현재 작업 디렉터리를 기준으로 어떤 파일의 위치를 지목하는 것을 **상대 경로(relative path)**라고 한다. 상대 경로를 잘 사용하기 위해서는 윈도우/맥오에스 관계없이 다음 약어를 기억할 필요가 있다.

- `.`: 현재 디렉터리
- `..`: 부모 디렉터리

위에서 `analysis.R`이라는 코드에서 `embolic.data`를 어떻게 표시할지 생각해 보자. `analysis.R` 파일과 `raw-data` 폴더는 모두 `Embolic` 디렉터리에 동급으로 존재한다. `analysis.R` 입장에서 현재 디렉터리에 있는 `raw-data` 폴더를 찾고, 그 안에 `embolic.data`를 찾아가면 된다. 따라서 이 경우에는 (맥오에스에서처럼 슬래쉬를 사용한다면) `./raw-data/embolic.data`라고 써주면 되는 것이다.

R에서 경로를 잡을 때는 윈도우 방식 보다는 맥오에스 방식으로 잡는 것이 보편적이다. 왜냐하면 윈도우에서 쓰는 역슬래쉬를 문자열 안에서 쓰려면 이것이 다른 의미가 있는 역슬래쉬가 아닌 문자 그대로의 역슬래쉬를 표현해야 해서 역슬래쉬를 하나 더 써서 escaping을 해야 하기 때문이다.

1.5 RStudio 프로젝트 폴더 구성하기

프로젝트 안에서 자기가 원하는 방식대로 폴더 등을 구성해서 사용하면 된다. 다음과 같은 형태로 자신이 원하는 대로 하면 된다.

- `excel-data` 폴더
- `clean-data` 폴더
- `analysis-code` 폴더
- `results` 폴더
- `images` 폴더 …

`excel-data` 폴더에 `mtcars.xlsx`라는 데이터 파일이 있다고 가정해 보자. 이 파일을 읽고, 결과를 `results` 폴더에 저장하는 예이다.

```
library(readxl)
library(dplyr)
df <- read_excel("./excel-data/mtcars.xlsx")
result <- df |>
```

```
    group_by(cyl) |>
    summarize(n = n(), mean_mpg = mean(mpg))

saveRDS(result, file = "./results/cars-by-cyl.rds")
```

아직 R 패키지에 대한 개념을 설명하지 않고, dplyr 패키지를 사용한 데이터 핸들링 등을 설명하지 않았다. 자세히 볼 것은 이 파일의 입장에서 읽어올 파일의 위치나 결과를 저장할 파일의 위치를 지정하는 방법이다.

궁금한 분들을 위해서 이 코드가 하는 일은 다음과 같다.

```
# 필요한 패키지 로딩
library(readxl)
library(dplyr)
```

```
# 엑셀 파일을 읽어와서 df 데이터프레임에 할당
df <- read_excel("./excel-data/mtcars.xlsx")
head(df) # 처음 6개의 행을 보여준다.
```

```
# A tibble: 6 x 11
    mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1  21       6   160   110  3.9   2.62  16.5     0     1     4     4
2  21       6   160   110  3.9   2.88  17.0     0     1     4     4
3  22.8     4   108    93  3.85  2.32  18.6     1     1     4     1
4  21.4     6   258   110  3.08  3.22  19.4     1     0     3     1
5  18.7     8   360   175  3.15  3.44  17.0     0     0     3     2
6  18.1     6   225   105  2.76  3.46  20.2     1     0     3     1
```

```
# 각 열 이름 아래 <dbl>  등은 해당 열의 데이터 타입이다.
```

```
result <- df |> # 이 데이터프레임에 대하여
    group_by(cyl) |> # cyl의 값에 따라 그룹을 나누고
    summarize(n = n(), mean_mpg = mean(mpg)) # 그룹별로 개수를 카운트하고, mpgd의 평균을 구하라
# 그 결과를 result에 할당한다.
```

```r
# 표로 모양으로 출력
knitr::kable(result)
```

cyl	n	mean_mpg
4	11	26.66364
6	7	19.74286
8	14	15.10000

```r
# 그 결과를 results 폴더에 rds 포맷으로 저장
saveRDS(result, file = "./results/cars-by-cyl.rds")
```

2 값, 이름, 할당, 연산

2.1 값

R 세계에서 어떤 데이터를 값(value)라고 하고 한다. R에서 값의 종류에는 다음과 같은 것들이 있다.

- 숫자: `1`, `3.14` 처럼 숫자는 그대로 입력한다.
- 문자열: `"stroke"`, `'peripheral'` 처럼 문자열은 작은따옴표 또는 큰따옴표를 감싸서 표현한다.
    - 작은따옴표 안에서 큰따옴표를 쓸 수 있고, 그 반대도 가능하다. 하지만 작은따옴표 안에서 작은따옴표를 사용할 수는 없다.
- 논리값: `TRUE` 또는 `T`, `FALSE` 또는 `F` 처럼 따옴표 없이 사용한다.
- 결측값: 결측값은 `NA`로 표시한다.

R 프롬프트에서 값을 입력하고 엔터키를 치면 그 값이 바로 출력된다.

```
3.14
```

```
[1] 3.14
```

```
"stroke"
```

```
[1] "stroke"
```

```
TRUE
```

```
[1] TRUE
```

2.2 이름과 할당(Assignment)

값에 이름을 부여하는 것을 할당(assignment)라고 하고 <- 연산자를 사용한다.

다음 코드는 85라는 정수에 wt라는 이름을 부여하고, 180에 ht라는 이름을 할당한다.

```
wt <- 85
ht <- 180
```

```
bmi <- wt / (ht / 100)^2
bmi
```

```
[1] 26.23457
```

R에서 이름을 만들 때는 **알파벳으로 시작하고**, 그 이후에 숫자나 언더스코어(_) 등을 붙여서 만든다. 한글 이름도 사용할 수는 있다. 하지만 보통 컴퓨터 코드는 ASCII 코드로 작성하는 것이 기본이다.

할당을 할 때 = 연산자를 사용할 수도 있지만, 보통 이것은 R 함수에서 인자의 값을 할당할 때 주로 사용된다.

2.3 연산자

수학에서 사용되는 연산자들이 모두 지원된다.

```
# 사칙연산
3 + 5
## [1] 8
5 - 2
## [1] 3
5 / 2
## [1] 2.5
5 * 3
## [1] 15
```

```
# 거듭제곱
5**2
## [1] 25
5^2
## [1] 25
```

다음은 비교 연산자 사용 예이다. 비교 연산은 논리값을 반환한다.

```
5 > 3 # 큰가?
## [1] TRUE
3 <= 5 # 작거나 같은가?
## [1] TRUE
5 == 5 # 같은가?
## [1] TRUE
4 != 5 # 다른가?
## [1] TRUE
```

논리 연산자도 있다. ||는 OR, &&는 AND를 의미한다. 하나로 된 |, &는 벡터를 대상으로 하는데 나중에 설명하고자 한다.

```
(5 > 3) && (4 > 3)
```

```
[1] TRUE
```

```
(5 > 3) && (4 < 3)
```

```
[1] FALSE
```

```
(5 > 3) || (4 < 3)
```

```
[1] TRUE
```

2.4 더 자세한 내용

더 자세한 내용은 Advanced R, Chapter 2, Names and values을 참고한다. ㄴ

3 벡터(vector)의 기초

R 언어는 통계 계산을 염두에 두고 설계된 언어답게, 하나의 값이 아니라 여러 값들을 모은 벡터(vector)라는 데이터 타입을 제공한다. 모든 함수가 그런 것은 아니지만, 많은 함수들이 벡터를 처리할 수 있는 기능을 제공한다. 따라서 R 언어를 잘 활용하기 위해서는 벡터와 나중에 설명할 데이터프레임(`data.frame`)을 잘 이해하는 것이 필수이다.

여기서 벡터에 관해서 어떻게 만들고, 어떻게 사용하고 하는 등 낮은 수준에서 높은 수준으로 설명을 해 나가니까, 실제로도 이런 밑작업을 해야 하는가라고 질문을 할 수도 있을 것이다. 실제로는 반대이다. 엑셀에서 데이터를 읽으면 대부분은 데이터프레임으로 저장되고, 여기서 벡터를 뽑아서 사용하게 된다.

R 벡터는 R 언어의 핵심 데이터 구조이기 때문에, 이해하고 충분히 연습해야 한다.

3.1 벡터 만들기, 종류

R 언어에서 `34`, `154`, `'ksb'` 같은 하나의 값들도 일종의 벡터이며, 이렇게 하나의 값으로 된 벡터를 **atomic vector**라고도 한다.

통계는 하나의 값보다는 여러 개의 값들을 한꺼번에 다루는 경우가 많다. 값들을 모아 하나의 벡터로 만들 때는 **`c()`(combine) 함수**를 사용한다. 값들을 코마로 구분해서 넣으면 된다. 어떤 값들을 하나의 벡터로 묶으려면 다음 조건을 만족해야 한다.

- 하나의 벡터에는 값들은 **"같은 데이터 타입"**이어야 한다. 숫자는 숫자, 문자는 문자, 불리언은 불리언끼리 묶을 수 있다.

```
height <- c(167, 180, 189, 165, 176)
height
```

```
[1] 167 180 189 165 176
```

연속된 정수로 된 벡터를 만들 때 :를 사용하면 편리하다.

```
ids <- 1:50
ids
```

```
 [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 25
[26] 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
```

숫자로 구성된 벡터를 숫자형 벡터(numeric vector), 논리값(불리언)으로 구성된 벡터를 로직컬 벡터(logical vector), 문자열로 구성된 벡터를 문자열 벡터(character vector)라고 부른다. 참고로 `typeof()` 함수를 사용하면 어떤 타입의 벡터인지 알 수 있다.

```
better_than_this <- c(TRUE, FALSE, TRUE, TRUE, FALSE)
typeof(better_than_this)
```

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

```
pt_names <- c("가", "나", "다", "라", "마")
typeof(pt_names)
```

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

3.2 결측값이 포함된 벡터

R에서 결측값은 `NA`로 표시한다. 실제 현장에서 수집되는 데이터에 결측값이 없는 경우란 거의 없다.

다음은 `NA`가 들어간 숫자형 벡터이다.

```
my_vec <- c(1:7, NA, NA, 10)
my_vec
```

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

다음은 `NA`가 들어간 문자형 벡터이다.

```
p_gender <- c("M", "M", "F", NA, NA, "F")
p_gender
```

```
[1] "M" "M" "F" NA  NA  "F"
```

결측값 NA는 데이터 분석에서 항상 문제를 일으킬 수 있다. NA를 가지고 하는 연산의 결과는 대부분 NA로 처리되기 때문이다.

그래서 R에서 벡터에 대한 계산을 수행하는 대부분의 함수는 NA를 빼고 계산하도록 하는 인자를 가지고 있다. na.rm = TRUE를 주어야 이 NA를 제외하고 나머지 유효한 값만을 가지고 계산을 수행한다.

3.3 벡터를 대상으로 하는 계산

벡터를 대상으로 계산을 하면, 벡터에서 같은 위치에서 있는 값끼리 계산된다.

```
x <- c(1, 2, 3, 4)
y <- c(4, 5, 6, 7)
x + y
## [1]  5  7  9 11
x - y
## [1] -3 -3 -3 -3
x / y
## [1] 0.2500000 0.4000000 0.5000000 0.5714286
x * y
## [1]  4 10 18 28
x^y
## [1]     1    32   729 16384
```

(가급적 피해야할 방법이기는 하지만), 벡터를 구성하는 값의 개수가 다른 경우에는, 작은 쪽에서 큰 쪽을 맞추도록 처음으로 돌아가 값을 가지고 와서 채운다. 이런 방식을 가급적 사용하지 않게 경고문도 출력되는 것을 알 수 있다. 이렇게 하는 경우를 리사이클링(recyling)이라고 한다.

```
c(1, 2, 3) + c(4, 5, 6, 7, 8)
```

```
Warning in c(1, 2, 3) + c(4, 5, 6, 7, 8): longer object length is not a
multiple of shorter object length
```

```
[1]  5  7  9  8 10
```

이것은 내부에서 다음과 같이 리사이클링을 되면서 계산되는 것이다.

```
c(1, 2, 3, 1, 2) + c(4, 5, 6, 7, 8)
```

```
[1]  5  7  9  8 10
```

어떤 벡터의 각 요소에 5를 더할 때는 다음과 같이 사용하는 경우가 대부분이다. 이런 경우에는 5가 atomic vector이고 이것이 앞의 요소의 개수만큼 리사이클링된 다음 계산된다고 이해할 수 있다.

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

```
[1] 6 7 8
```

NA가 들어가면 어떻게 되는지 보자. NA는 NA로 되는 것이 대부분이다.

```
c(1, 2, NA, 4) + 5
```

```
[1]  6  7 NA  9
```

3.4 벡터의 길이와 인덱스

벡터의 길이(length)란 벡터 요소 개수이며, `length()` 함수를 사용하여 계산한다.

```
my_vec <- c(1, 2, 3, 4, 5, NA, 7)
length(my_vec)
```

```
[1] 7
```

벡터의 인덱스는 벡터 안에서 값의 위치를 말하고, 벡터 이름 뒤에 [정수]의 형태로 사용된다. R 벡터는 1부터 시작한다[1].

```
my_vec[1] # 첫 번째 값
```

```
[1] 1
```

length() 함수를 사용하면 벡터의 길이를 구할 수 있고, 이 값을 [] 안에 넣으면 마지막 값을 알 수 있다.

```
my_vec[length(my_vec)]
```

```
[1] 7
```

3.5 벡터 서브셋팅(subsetting)

[] 안에 다양한 식을 넣어서, 벡터의 값들을 꺼낼 수 있다. 이것을 벡터에 대한 서브셋팅(subsetting)이라고 한다. 다음과 같은 벡터를 가지고 시작해 보자.

```
x <- 1:30
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 25
[26] 26 27 28 29 30
```

앞의 내용은 복수하면 10번째 값은 다음과 같이 해서 얻을 수 있다.

```
x[10]
```

```
[1] 10
```

10, 20, 30번째 값을 한꺼번에 꺼내려면 이 정보를 하나의 벡터로 만들어 전달하면 된다.

[1] Python 같은 언어는 0부터 시작한다

```
x[c(10, 20, 30)]
```

```
[1] 10 20 30
```

안에 들어가는 벡터 앞에 마이너스 기호(-)를 붙이면, 이 벡터의 인덱스에 해당하는 값들을 제외하라(exlusion)는 의미를 가진다.

```
x[-c(10, 20, 30)]
```

```
 [1]  1  2  3  4  5  6  7  8  9 11 12 13 14 15 16 17 18 19 21 22 23 24 25 26 27
[26] 28 29
```

[] 안에 불리언 벡터를 넣으면 참인 위치에 있는 값들을 추출한다. 이 기능은 참, 거짓을 반환하는 식을 사용하면 조건에 맞는 값들만 추출하는 데 사용된다.

```
x[x %% 2 == 0] # 짝수만
```

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

```
x[x > 24] # 24보다 큰 값들만
```

```
[1] 25 26 27 28 29 30
```

이것을 읽을 때는, 먼저 [] 안에서 식이 계산되고, 그 결과가 [] 에 사용된다고 이해해야 한다.

안의 것을 먼저 계산하면 다음과 같다.

```
byp x %% 2 == 0
```

위 결과는 원래의 x와 길이가 같다. 값들이 일렬로 나열되었다고 상상하고 x 옆에 이것을 가져다 놓았을 때, TRUE에 해당되는 값만을 가지고 온다고 이해하면 된다.

다음의 경우에는 [] 안의 벡터가 밖에 있는 벡터보다 짧기 때문에 recyling이 발생한다. 그래서 TRUE가 2개의 값을 건너뛰면서 나타나기 때문에 이런 결과가 나온다.

```
x[c(TRUE, FALSE, FALSE)]
```

```
 [1]  1  4  7 10 13 16 19 22 25 28
```

[] 안에 아무것도 넣지 않으면 이것은 '모든(all)'을 의미한다.

```
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 25
[26] 26 27 28 29 30
```

> ℹ []를 사용한 서브셋팅 정리
>
> - 하나의 정수, 또는 정수형 벡터: 해당 위치의 값
> - 음의 정수 또는 정수형 벡터 앞에 마이너스: 해당 위치의 값을 제외
> - 아무 것도 없음: All
> - 불리언 또는 불리언 벡터: TRUE의 위치의 있는 값들을 추출

이 내용은 2차원의 데이터를 가지는 데이터프레임(data.frame)에서도 확장된다.

3.6 벡터의 값 변경

벡터가 가지고 있는 값을 바꾸려면 앞에서 본 세브셋팅과 거의 같은 문법을 사용한다.

할당 기호 <- 중심으로 좌변에 이 문법을 사용하여 타깃(target)을 정하고, 할당 기호 우변에 바꿀 값을 준다.

다음 x 벡터를 사용하여 설명한다.

```
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 25
[26] 26 27 28 29 30
```

```
# 첫번째 요소를 101로
x[1] <- 101
# 10, 20, 30번째 요소를 각각 100, 200, 300으로
x[c(10, 20, 30)] <- c(100, 200, 300)
```

```
x
```

```
 [1] 101   2   3   4   5   6   7   8   9 100  11  12  13  14  15  16  17  18  19
[20] 200  21  22  23  24  25  26  27  28  29 300
```

```
# 2로 나눈 몫이 0인, 즉 짝수인 경우는 모두 0으로
x[x %% 2 == 0] <- 0
x
```

```
 [1] 101   0   3   0   5   0   7   0   9   0  11   0  13   0  15   0  17   0  19
[20]   0  21   0  23   0  25   0  27   0  29   0
```

같은 이름에 새로운 값을 할당하면 (원래의 값을 사라지고) 새로운 값을 가진 벡터가 된다.

1번째에서 10번째 값만 남기고 나머지를 버린 x를 만들어 보자.

```
x <- x[1:10]
x
```

```
 [1] 101   0   3   0   5   0   7   0   9   0
```

3.7 벡터의 요소 값에 이름을 붙여서 사용하기

벡터의 요소에 이름을 부여할 수 있다. c() 함수를 사용하여 벡터를 만들 때, 그 인자로 이름 = 값을 주면 해당 값에 대한 이름을 가진 벡터가 생성된다.

```
y <- c(a = 11, b = 22, c = 33, d = 44)
y
```

```
 a  b  c  d
11 22 33 44
```

이렇게 이름이 부여되면, 그 이름을 문자열로 []에 주면 해당 값에 접근할 수 있다.

```
y["d"]
```

```
 d
44
```

이름을 사용하여 서브셋팅을 할 수도 있다.

```
y[c("a", "c")]
```

```
 a  c
11 33
```

벡터(나중에 데이터프레임도 마찬가지고)에 어떤 이름들을 가지고 있는지 확인하려면 `names()` 함수를 사용한다.

```
names(y)
```

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

이것을 할당 기호 좌변에 써서 이름을 바꾸는 데 사용할 수 있다.

```
names(y)[2] <- "bb"
y
```

```
 a bb  c  d
11 22 33 44
```

이 기능을 사용하여 처음에 이름 없이 만들었던 벡터에 이름을 부여할 수 있다.

```
z <- c(1, 2, 3, 4)
names(z) <- c("a", "b", "c", "d")
z
```

```
a b c d
1 2 3 4
```

4 데이터프레임(data.frame)의 기초

여기서는 R의 가장 중요한 데이터 구조라 할 수 있는 데이터프레임(data.frame)에 대해서 설명한다.

현장에서 엑셀 파일로 데이터를 수집하여 사용하는 경우가 많아, R에서 엑셀 파일을 읽는 일을 흔히 보게 된다. 이런 엑셀 파일을 읽으면 R 데이터프레임으로 읽는다. 그래서 엑셀 파일을 읽는 방법과 데이터프레임의 기초에 대해서 순서대로 설명한다.

다음 엑셀 파일이 워킹 디렉터리의 `excel-data` 폴더에서 `mtcars.xlsx` 파일로 존재하다고 가정한다. 파일을 다운로드하여 현재 디렉터리에 `excel-data` 폴더를 만들고 그 안에 저장하자.

- 엑셀 데이터 다운로드

4.1 엑셀 파일 열어서 보기

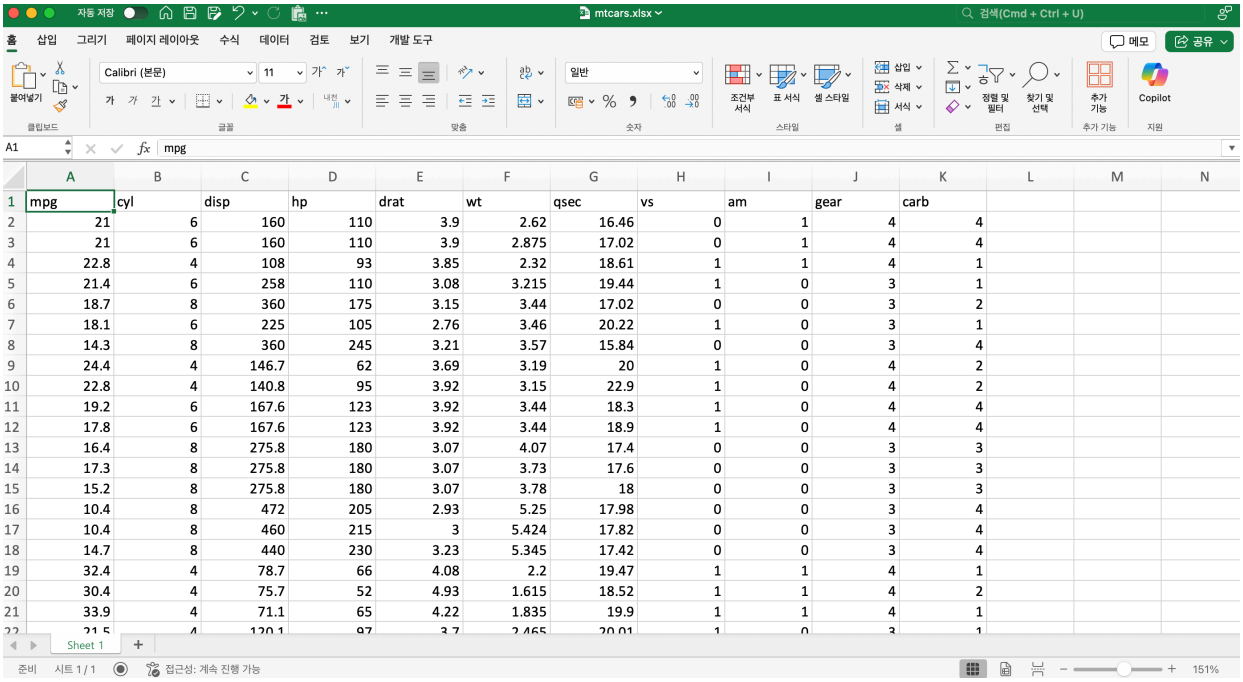
이 파일에 들어 있는 데이터는 R을 설치할 때 함께 포함되는 `mtcars`라는 데이터를 엑셀로 만들어 저장한 것이다.

```
> head(mtcars)
```

```
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

R (또는 Python)과 같은 언어를 통해 데이터를 분석할 것을 고려한다면 엑셀 데이터가 어떤 식으로 정리되어야 하는지 생각해 보자. 엑셀에서는 많은 사용자가 그러하듯이 사용자가 유연하게 데이터를 입력하고, 계산을 하고, 그래프나 표를 만들어 쓸 수 있다. 만약에 하나의 엑셀 시트에서 이런 것들이 뒤섞여 있으면 R과 같은 도구로 데이터를 불러올 때 원 데이터를 읽어올 위치 등을 상세하게 지정해 주어야 한다. 따라서 R 언어를 사용하여 분석할 데이터는 다음 그림과 같이 내용을 구성할 것을 권한다.

- 첫 행은 열 이름으로 사용한다.
    - 제목을 쓰고, (보기 편하게) 빈 행을 만들거나 하지 않는다.
    - 열 이름을 (한글 이름 대신) 알파벳으로 빈칸 없이 만든다. 나중에 이것을 변수의 이름으로 사용해야 하기 때문에 다시 다른 이름으로 바꾸거나 하는 등의 작업이 필요할 수 있다.
- 두 번째 행부터 바로 데이터가 채워진다.
- 메타 정보는 별도의 시트나 파일로 따로 정리하여 사용하는 것이 좋다.



	A	B	C	D	E	F	G	H	I	J	K
1	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
2	21	6	160	110	3.9	2.62	16.46	0	1	4	4
3	21	6	160	110	3.9	2.875	17.02	0	1	4	4
4	22.8	4	108	93	3.85	2.32	18.61	1	1	4	1
5	21.4	6	258	110	3.08	3.215	19.44	1	0	3	1
6	18.7	8	360	175	3.15	3.44	17.02	0	0	3	2
7	18.1	6	225	105	2.76	3.46	20.22	1	0	3	1
8	14.3	8	360	245	3.21	3.57	15.84	0	0	3	4
9	24.4	4	146.7	62	3.69	3.19	20	1	0	4	2
10	22.8	4	140.8	95	3.92	3.15	22.9	1	0	4	2
11	19.2	6	167.6	123	3.92	3.44	18.3	1	0	4	4
12	17.8	6	167.6	123	3.92	3.44	18.9	1	0	4	4
13	16.4	8	275.8	180	3.07	4.07	17.4	0	0	3	3
14	17.3	8	275.8	180	3.07	3.73	17.6	0	0	3	3
15	15.2	8	275.8	180	3.07	3.78	18	0	0	3	3
16	10.4	8	472	205	2.93	5.25	17.98	0	0	3	4
17	10.4	8	460	215	3	5.424	17.82	0	0	3	4
18	14.7	8	440	230	3.23	5.345	17.42	0	0	3	4
19	32.4	4	78.7	66	4.08	2.2	19.47	1	1	4	1
20	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
21	33.9	4	71.1	65	4.22	1.835	19.9	1	1	4	1

그림 4.1: R 등으로 분석하고자 데이터를 엑셀로 저장하는 경우 데이터는 데이터, 메타데이터는 메타데이터로 정리하는 것이 좋다.

이 파일이 `excel-data` 폴더에 `mtcars.xlsx`으로 있고 이제 이것을 읽어보자.

4.2 RStudio에서 엑셀 파일 읽기

R에는 인터넷만 연결되어 있으면 데이터에 어디에 있던 어떤 포맷으로 되어 있던 데이터를 읽는 도구들이 개발되어 있다.

여기선 가장 기초적인 현재 컴퓨터에 있는 엑셀 파일을 읽어보자. RStudio에는 이런 작업을 쉽게 할 수 있는 기능이 내장되어 있다.

RStudio를 실행하고 따라해 보자.

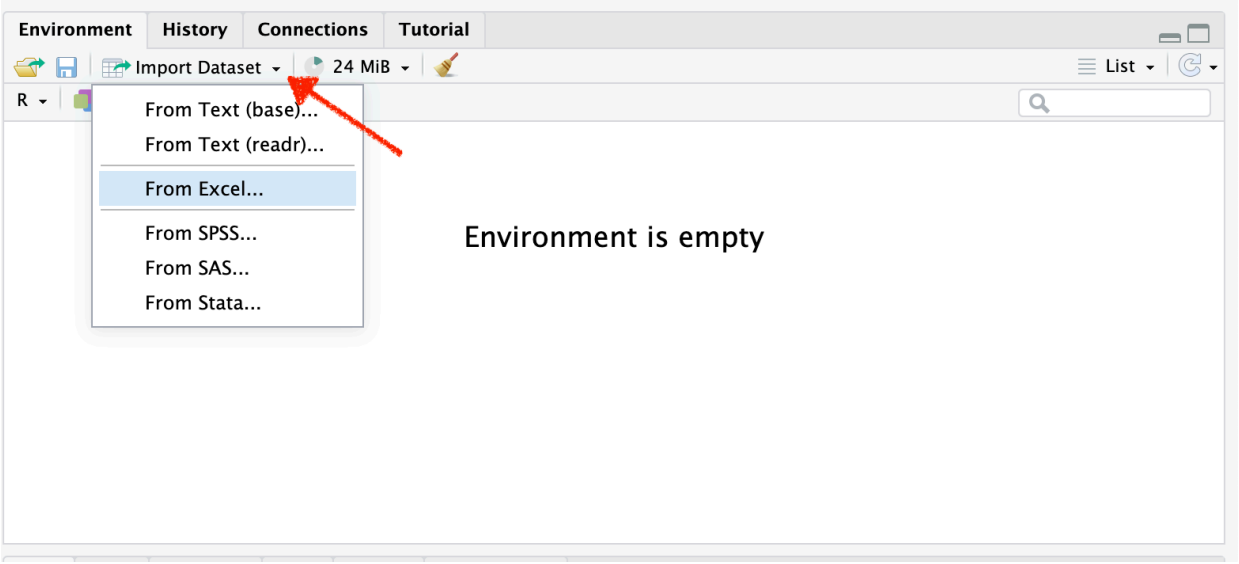


그림 4.2: RStudio의 “Environment” 창에서 “Import Dataset” 오른쪽 화살표를 클릭한다. 여기서 “From Excel…”를 선택한다.

이 기능은 `readxl`이라는 패키지를 이용하는데, 만약 현재 컴퓨터에 이 패키지가 설치되어 있지 않다면 이 패키지를 다운로드하라는 안내를 보게 될 것이다. 다운로드하여 설치한다.

“From Excel…”을 클릭하면 그림 4.3 창이 열린다. 여기서 **Browse…** 버튼을 클릭하여 읽을 파일을 찾아서 선택한다.

만약 읽을 때 문제가 없을 경우 그림 4.4 창과 같이 보인다.

여기서 데이터를 잘 읽었는지 확인한다. 오른쪽 아래에 이 과정을 코드로 만들어 주고 있다. 이것을 클릭보드에 복사하여 스크립트 등에 사용하면 편리한다.

여기서 오른쪽 아래 **Import** 버튼을 클릭하면 이 데이터를 `mtcars`라는 데이터프레임으로 읽어 온다. 이 방법은 엑셀 파일 이름과 같은 이름으로 데이터프레임으로 읽어 온다(정확히는 tibble이라는 데이터

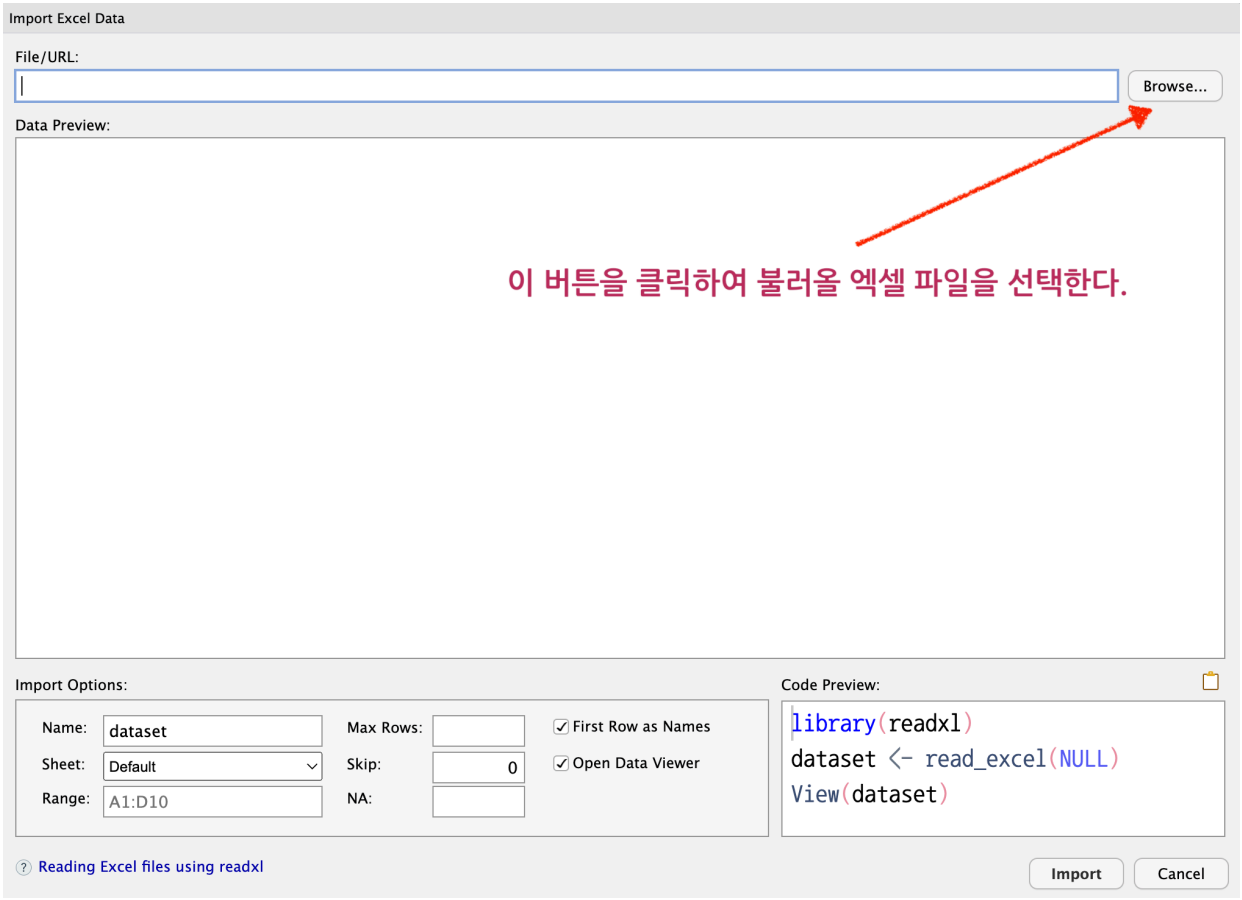

Import Excel Data
File/URL:
Browse...
Data Preview:
이 버튼을 클릭하여 불러올 엑셀 파일을 선택한다.
Import Options:
Name:
dataset
Max Rows:
First Row as Names
Sheet:
Default
Skip:
0
Open Data Viewer
Range:
A1:D10
NA:
Code Preview:
library(readxl)
dataset <- read_excel(NULL)
View(dataset)
Reading Excel files using readxl
Import
Cancel

그림 4.3

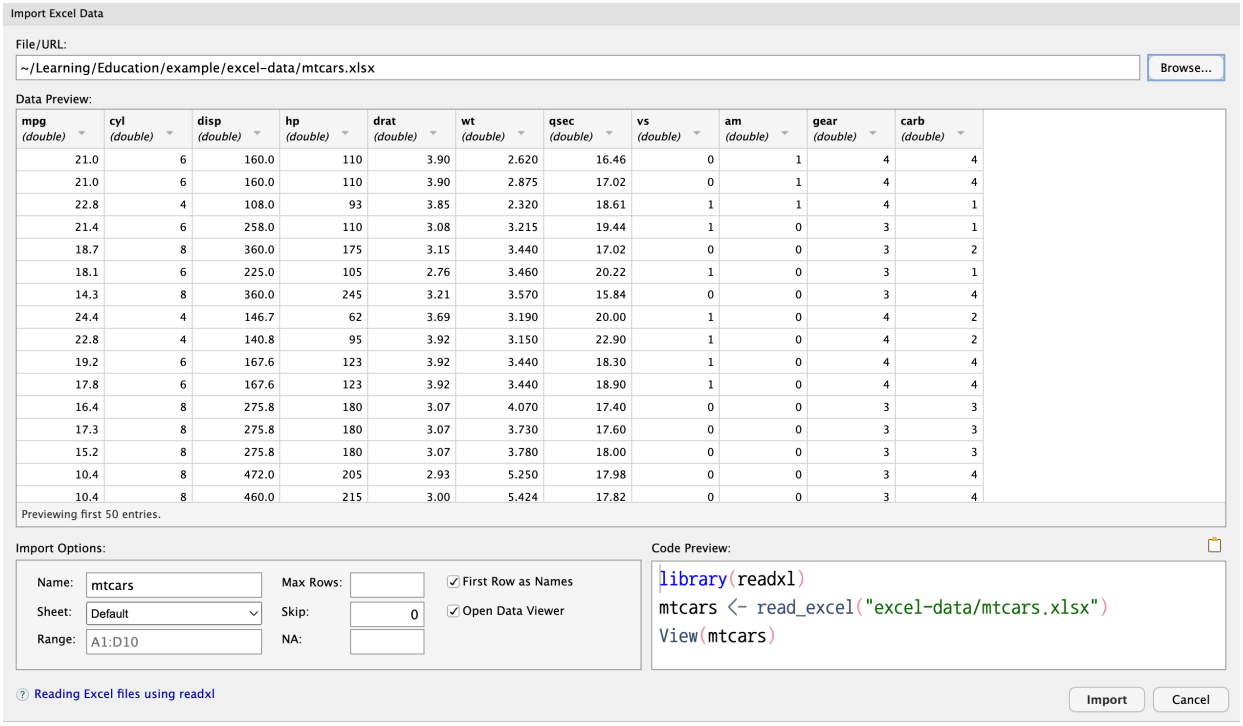


그림 4.4: 엑셀 파일은 읽은 모습

구조로 읽는데, 전통적인 데이터프레임을 강화한 형태의 데이터프레임이다). 이제 R 세션에 `mtcars`라는 데이터프레임이 만들어졌다.

4.3 데이터프레임

R 데이터프레임(`data.frame`)은 엑셀 시트와 비슷한 2차원으로 된 데이터 구조이다. 다음과 같이 데이터를 읽었을 때 tibble 데이터프레임으로 저장하는 데 설명을 위해서, 원래의 데이터프레임으로 만든다(그 이름은 `df`로 줬다).

```
library(readxl)
mtcars <- read_excel("excel-data/mtcars.xlsx")
df <- as.data.frame(mtcars) # 원래의 데이터프레임으로 변환
head(df)
```

```
   mpg cyl disp  hp drat    wt  qsec vs am gear carb
1 21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
```

```
2 21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
3 22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
4 21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
5 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
6 18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

- `str()` 함수는 데이터프레임의 구조를 한눈에 파악하는 데 좋다. 결과를 보면 32개의 관측값(observations)과 11개의 변수(열)로 구성되어 있고, 모든 열이 숫자(`num`)으로 되어 있다는 것을 알 수 있다.

```
str(df)
```

```
'data.frame':   32 obs. of  11 variables:
 $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
 $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
 $ disp: num  160 160 108 258 360 ...
 $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
 $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
 $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
 $ qsec: num  16.5 17 18.6 19.4 17 ...
 $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
 $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
 $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
 $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

- 행의 개수는 `nrow()`, 열의 개수는 `ncol()` 함수로 알 수 있고, `dim()` 함수를 사용하여 행과 열이 한꺼번에 계산된다.

```
nrow(df)
```

```
[1] 32
```

```
ncol(df)
```

```
[1] 11
```

```
dim(df)
```

```
[1] 32 11
```

- 변수, 즉 열 일이름은 `names()` 함수를 사용하면 알 수 있다.

```
names(df)
```

```
 [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"   "gear"
[11] "carb"
```

- 데이터프레임에서 앞 부분을 볼 때는 `head()` 함수를, 끝 부분을 볼 때는 `tail()` 함수를 사용한다. 디폴트는 6개의 행을 보여주는 것이다.

```
head(df)
```

```
   mpg cyl disp  hp drat    wt  qsec vs am gear carb
1 21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
2 21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
3 22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
4 21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
5 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
6 18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```
tail(df)
```

```
    mpg cyl  disp  hp drat    wt qsec vs am gear carb
27 26.0   4 120.3  91 4.43 2.140 16.7  0  1    5    2
28 30.4   4  95.1 113 3.77 1.513 16.9  1  1    5    2
29 15.8   8 351.0 264 4.22 3.170 14.5  0  1    5    4
30 19.7   6 145.0 175 3.62 2.770 15.5  0  1    5    6
31 15.0   8 301.0 335 3.54 3.570 14.6  0  1    5    8
32 21.4   4 121.0 109 4.11 2.780 18.6  1  1    4    2
```

4.4 데이터프레임은 벡터를 하나로 모은 것

데이터프레임은 1차원 벡터를 세워서 2차원으로 만든 것이라 보면 된다. 하나의 벡터를 구성하는 값의 데이터 타입은 모두 같아야 하는데(homogeneous), 데이터프레임을 구성하는 벡터는 타입이 달라도 상관없다(heterogeneous).

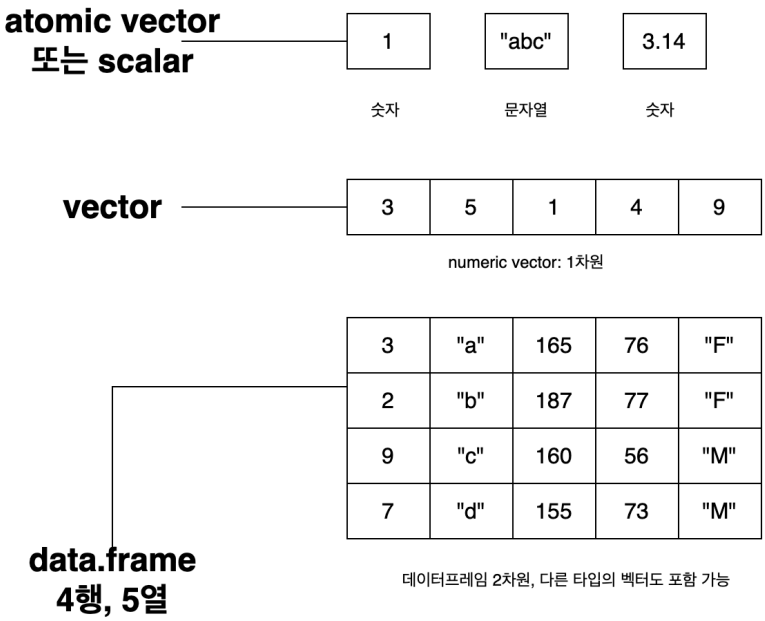


그림 4.5: R 언어에서 가장 핵심적인 데이터 구조

값들을 `c()` 함수로 모아서 하나의 벡터로 만들 것과 같이, `data.frame()`이라는 함수를 사용하여 벡터들을 하나의 테이터프레임으로 묶을 수 있다.

```
id <- 1:3
p_name <- c("A", "B", "C")
p_wt <- c(90, 76, 67)
p_ht <- c(185, 179, 160)
p_df <- data.frame(id, p_name, p_wt, p_ht)
p_df
```

```
  id p_name p_wt p_ht
```

```
1  1      A   90  185
2  2      B   76  179
3  3      C   67  160
```

이것을 하나의 데이터프레임으로 묶을 수 있는 이유는 4개의 벡터의 길이가 모두 3이기 때문이다.

벡터에서는 []라는 사용한 인덱싱 방법으로 값에 접근했다. 데이터프레임도 같은 방법으로 해당 데이터프레임에 있는 벡터나 또는 개별 값을 가지고 올 수 있다. 이 방법에 대해 알아보자.

- 먼저 $를 사용하는 방법이 있다. 다음과 같이 데이터프레임이름$열이름이라는 문법을 사용하여 데이터프레임에서 벡터를 가지고 올 수 있다.

```
p_df$p_wt
```

```
[1] 90 76 67
```

앞 절에서 엑셀 데이터로 만든 df 데이터프레임에서 mpg라는 변수를 가지고 올 때는 다음과 같이 실행한다.

```
df$mpg
```

```
 [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2 10.4
[16] 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4 15.8 19.7
[31] 15.0 21.4
```

어쩔 수 없이 데이터프레임의 열 이름이 띄어쓰기 있다고 가정해 보자. 벡터에서 값의 이름을 바꾸는 경우와 값이 names() 함수를 할당 좌측에 놓아서 이름을 바꿀 수 있다.

```
names(p_df)[4] <- "환자 키"
p_df
```

```
  id p_name p_wt 환자 키
1  1      A   90     185
2  2      B   76     179
3  3      C   67     160
```

이렇게 된 경우에는 다음과 같이 따옴표를 사용해야 한다.

```
p_df$"환자 키"
```

```
[1] 185 179 160
```

4.5 [] 또는 [[]]을 사용한 세브셋팅

벡터의 서브셋팅을 다룰 때 다음과 같이 정리했었다.

> []를 사용한 서브셋팅 정리
>
> - 하나의 정수, 또는 정수형 벡터: 해당 위치의 값
> - 음의 정수 또는 정수형 벡터 앞에 마이너스: 해당 위치의 값을 제외
> - 아무 것도 없음: All
> - 불리언 또는 불리언 벡터: TRUE의 위치의 있는 값들을 추출

이 원리를 2차원 벡터에 적용할 수 있다. 다만 [행, 열]과 같이 코마로서 행의 조건과 열의 조건을 다르게 지정할 수 있다는 점이 다르다.

또 약간 다른 점은 벡터에서는 값에 이름을 붙여서 사용하는 경우가 드물기 때문에 [] 안에 인덱스 숫자를 많이 사용하지만, 데이터프레임의 경우에는 행은 보통 행 번호 인덱스롤, 열은 보통 열 이름을 사용한다.

다음 코드를 보자. 이 경우에는 행의 조건을 주는 부분이 빈 채로 되어 있어서 “모든 행”을 의미한다. 그러고 열의 자리에 "mpg"라는 열 이름을 주었다. 열을 기준으로 보면 "mpg"라는 열의 값을 가지고 오는데 모든 행의 값을 가지고 오라는 뜻이다.

```
df[, "mpg"]
```

```
 [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2 10.4
[16] 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4 15.8 19.7
[31] 15.0 21.4
```

다음은 "mpg"라는 열에서 1번에서 5번 행까지의 값만 가지고 오라는 뜻이다.

```
df[1:5, "mpg"]
```

```
[1] 21.0 21.0 22.8 21.4 18.7
```

다음 경우에는 "mpg", "wt" 열을 가지고 오는 1번에서 5번까지의 행만 가지고 오라는 뜻이 된다.

```
df[1:5, c("mpg", "wt")]
```

```
   mpg    wt
1 21.0 2.620
2 21.0 2.875
3 22.8 2.320
4 21.4 3.215
5 18.7 3.440
```

다음은 p_df에서 3번째 행의 값을 보는데, 열 조건이 빈 상태이기 때문에 "모든 열"이 된다.

```
p_df[3, ]
```

```
  id p_name p_wt 환자 키
3  3      C   67     160
```

다음은 df의 4행, 10열의 값을 가지고 오는 코드이다.

```
df[4, 10]
```

```
[1] 3
```

주의 깊게 살펴보면 사실 인덱싱의 결과가 어떤 때는 하나의 값(atomic vector)가 되고, 어떤 경우에는 벡터, 어떤 경우에는 2차원 데이터프레임을 유지한다. 이런 경우들은 처음 배울 때는 넘어가고, 나중에 하나씩 고민해도 된다.

5 R 팩터(factor)

R 언어는 원래 통계 분석을 염두에 두고 개발된 언어답게, 통계학에서 다루는 카테고리형(categorical) 변수 및 순서형(ordinal) 변수를 위한 특수한 형태의 벡터를 지원한다.

참고로 통계학에서는 (책마다 조금씩 다르기는 하지만) **scales of measurement**을 다음과 같이 정의한다.

- Nominal Scale
    - (“치료군”, “대조군”), (“남자”, “여자”), (“발생함”, “발생하지 않음”), (“호전”, “호전되지 않음”) 등
- Ordinal Scale
    - Cancer stage, 사회경제적인 수준, 교육 수준
- Numerical Scales
    - 체중, 나이 등

통계학에는 어떤 변수가 어떤 scales을 가지는지에 따라서 데이터를 써머리하거나 그래프를 만들거나, 통계 검정법을 어떤 것을 쓸지가 결정되기 때문에 무척 중요하다.

- Nominal Scale
    - 주요 통계량: percentage, proportion
    - 주요 요약 방법: contingency table, bar chart
- Ordinal Scale
    - 주요 통계량: median
    - 주요 요약 방법: contingency table, bar chart
- Numerical Scales
    - 주요 통계량: mean, var와 sd

- 주요 요약 방법: histogram, box plot, frequency polygon, scatter plot 등

이런 내용은 R 언어에서도 중요하다. 여기에서 카테고리형 변수를 표현하기 위한 R 팩터 데이터 타입에 대해서 설명한다.

5.1 문자열 벡터와 팩터(factor)

뇌전증 약물의 효과를 "sz_free", "more_than_fifty"(>= 50%), "less_than_fifty"(< 50%), "no_response"로 구분한다고 생각해 보자.

20명의 결과가 다음과 같다고 해보자.

```
treatment_effect <- c("sz_free", "sz_free", "more_than_fifty", "less_than_fifty", "less_than_fi
treatment_effect
```

```
 [1] "sz_free"         "sz_free"         "more_than_fifty" "less_than_fifty"
 [5] "less_than_fifty" "no_response"     "sz_free"         "more_than_fifty"
 [9] "less_than_fifty" "less_than_fifty" "no_response"     "sz_free"
[13] "more_than_fifty" "less_than_fifty" "less_than_fifty" "no_response"
[17] "no_response"     "sz_free"         "sz_free"         "more_than_fifty"
```

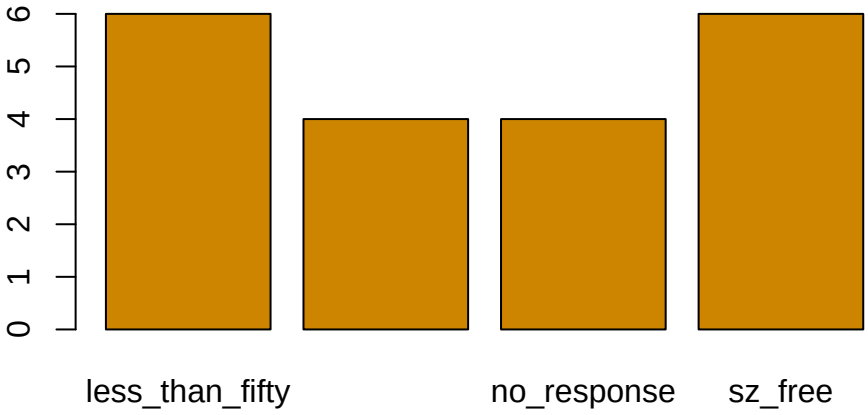
지금 `treatment_effect` 벡터는 문자열 벡터이다. 이 상태로도 기본 계산 등은 가능하다. 예를 들어 각 카테고리와 거기에 해당되는 개수를 카운팅해 보자. 이런 것을 contingency table이라고 한다. R에 `table()` 함수는 이런 contigency table을 만들어 주는 함수이다.

```
table(treatment_effect)
```

```
treatment_effect
less_than_fifty more_than_fifty     no_response         sz_free
              6               4               4               6
```

이를 바탕으로 막대 그래프(bar plot)을 만들 수 있다.

```
barplot(table(treatment_effect), col = "orange3")
```



Contingency table이나 bar plot을 자세히 관찰해 보면 이 데이터가 가지고 있는 순서(ordinal)의 의미를 제대로 반영하지 못하고 있다. 이런 문제를 해결하기 위해서는 이 벡터를 팩터, 특히 ordinal 의미가 있는 **팩터(factor)**로 만들어줄 필요가 있다.

5.2 팩터로 만들기

다음 문자열 벡터 `treatment_effect`를 `factor()` 함수를 사용하여 팩터로 만들고자 한다.

```
treatment_effect <- c("sz_free", "sz_free", "more_than_fifty", "less_than_fifty", "less_than_fi
treatment_effect
```

```
 [1] "sz_free"         "sz_free"         "more_than_fifty" "less_than_fifty"
 [5] "less_than_fifty" "no_response"     "sz_free"         "more_than_fifty"
 [9] "less_than_fifty" "less_than_fifty" "no_response"     "sz_free"
[13] "more_than_fifty" "less_than_fifty" "less_than_fifty" "no_response"
[17] "no_response"     "sz_free"         "sz_free"         "more_than_fifty"
```

R 언어에서 어떤 팩터가 취할 수 있는 값들의 종류를 그 팩터의 **레벨(levels)**라고 한다. 그리고 그 레벨은 다음과 같이 `factor()` 함수에서 `levels`라는 인자로 지정할 수 있다.

```
tx_factor <- factor(treatment_effect,
    levels = c("sz_free", "more_than_fifty", "less_than_fifty", "no_response")
)
tx_factor
```

```
 [1] sz_free         sz_free         more_than_fifty less_than_fifty
 [5] less_than_fifty no_response     sz_free         more_than_fifty
 [9] less_than_fifty less_than_fifty no_response     sz_free
[13] more_than_fifty less_than_fifty less_than_fifty no_response
[17] no_response     sz_free         sz_free         more_than_fifty
Levels: sz_free more_than_fifty less_than_fifty no_response
```

levels를 factor() 함수에서 지정하지 않으면 값들을 읽어서 자동으로 레벨을 정하고 (의미없는) 알파벳 순서에 따르게 된다. 성별과 같이 순서가 중요하지 않은 완전한 nominal data라면 이 방법을 따라도 좋다. 하지만 분석과 결과에서 일관성을 유지하려면 levels을 지정하는 것이 좋다.

위에서 팩터를 출력한 결과를 보면, 마지막에 "Levels: …"가 보인다. 이것을 보더라도 이 변수가 팩터이고, 이런 순서로 레벨이 정해져 있음을 알 수 있다. 또 어떤 팩터 벡터의 레벨을 확인할 때는 levels() 라는 함수를 사용한다.

```
levels(tx_factor)
```

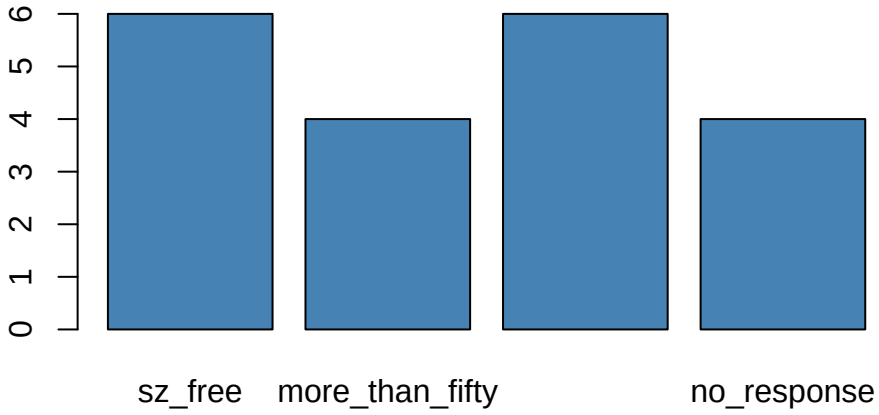
```
[1] "sz_free"         "more_than_fifty" "less_than_fifty" "no_response"
```

이제 팩터로 제대로 바뀌었으니까, 다시 contingency table과 bar plot을 만들어 보자. 위에 문자열 벡터로 시행했던 결과들과 비교해 보면, 치료의 효과라는 순서(order)에 따라, 우리가 지정한 levels에 따라서 표가 정리되고, 막대 그래프가 만들어지는 것을 확인할 수 있다.

```
table(tx_factor)
```

```
tx_factor
        sz_free more_than_fifty less_than_fifty     no_response
              6               4               6               4
```

```
barplot(table(tx_factor), col = "steelblue")
```



5.3 더 참고할 자료

R의 카테고리형 데이터를 표시하는 팩터(factor)는 때로는 아주 까다로운 문제로 다가올 수 있다. 이 페이지에서 설명한 기초를 이해하고, 팩터를 사용하다 곤란한 문제가 생기면 `forcats`라는 팩터를 전문으로 다루는 패키지가 해법이 될 수 있다. 팩터와 이 패키지 사용법은 다은 자료에 아주 잘 설명되어 있다.

- R for Data Science (2e): 16장 Factors

6 조건, 반복, 함수

이 장에서는 R 언어에서의 Control flow, 즉 조건부 실행과 반복 실행에 대해서 설명한다. 또 가끔 반복된 코드를 사용하는 경우 함수를 만들어 사용하는 것이 편리한 경우들이 있다. 그래서 사용자 정의 함수를 만들고 사용하는 방법에 대해서도 설명한다.

6.1 조건부 실행

R 언어는 벡터(vector)를 기본 데이터로 사용한다. 그리고 벡터화라는 방식을 통해서 작동하기 때문에 보통의 프로그래밍 언어와는 달리 이런 조건, 반복을 잘 사용하지는 않지만, 그래도 가끔 필요할 때가 있다.

6.1.1 `if/else` 문

`if` 문(statement)의 문법은 다음과 같다.

```
if (조건) 문장1
if (조건) 문장1 else 문장2
```

첫 번째인 경우에는 조건이 TRUE인 경우에만 문장1이 실행된다. 두 번째인 경우에는 조건이 TRUE인 경우에는 문장1이 실행되고, 그렇지 않은 경우(조건 == FALSE)에는 문장2가 실행된다. 만약 여러 문장을 사용할 필요가 있는 경우에는 그것들을 문장1, 문장2를 {}로 감싸면 된다.

어떤 사람의 나이 데이터가 있을 때 이 값이 50 이상이면 50 이상이라고 출력하도록 만들어 보자.

```
age <- 67
if (age >= 50) print("50 이상")
print("계산을 완료함.")
## [1] "50 이상"
```

```
## [1] "계산을 완료함."
```

위 경우에는 age >= 50이 TRUE이기 때문에 그 다음 print() 문이 실행되었다.

```
age <- 45
if (age >= 50) print("50 이상")
print("계산을 완료함.")
## [1] "계산을 완료함."
```

위 경우에는 age >=50이 FALSE이기 때문에 print() 문이 실행되지 않고 바로 다음 문장이 실행된다.

이번에는 50 이상이면 50 이상이라고 출력하고, 50 보다 작으면 50 미만이라고 출력해 보는 코드를 만들어 본다. age >= 50이 FALSE이기 때문에 첫 번째 print() 문을 실행되지 않고 나중 print() 문만 실행된다.

```
age <- 45
if (age >= 50) {
    print("50 이상")
} else {
    print("50 미만")
}
```

```
[1] "50 미만"
```

6.1.2 벡터화된 if: ifelse() 함수

R의 ifelse()라는 함수는 벡터화된 if라고 한다. 예를 들어 스코어가 70 이상이면 pass이고 70 미만이면 fail이라고 코딩해 보고 싶을 수 있다.

기본 문법은 다음과 같다.

```
ifelse(조건, 참일때값, 거짓일때값)
```

다음과 같이 6명의 점수가 있다고 해 보자.

```
scores <- c(78, 67, 80, 90, 83, 59)
```

이 값을 바탕으로 Pass, Fail로 된 새로운 벡터를 만들어 보자.

```
pass_fail <- ifelse(scores >=70, "Pass", "Fail")
pass_fail
```

```
[1] "Pass" "Fail" "Pass" "Pass" "Pass" "Fail"
```

위 코드는 다음과 같이 단계별로 생각해야 한다.

먼저 ifelse 함수의 첫 번째 인자가 scores >= 70이다.

```
scores >= 70
```

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

이렇게 만들어진 벡터를 가지고 ifelse() 함수가 TRUE인 위치에는 "Pass" 값으로, FALSE인 위치에는 "Fail" 값으로 대체한다.

ifelse() 함수는 조건에 따라 데이터를 2진(2개)로 나눌 때 편리하게 사용된다. 만약, 어떤 값에 따라 "mild", "moderate", "severe", "catastrophic"이라고 나누는 것처럼 3개 이상으로 그룹화할 필요가 있는 경우에는 dplyr 패키지의 case_when()을 고려한다. A general vectorised if-else를 참고한다.

6.2 반복

6.2.1 for 문: 정해진 개수만큼 반복 실행

for 문을 사용하면 정해진 개수만큼 코드를 반복한다. 기본 문법은 다음과 같다.

```
for (v in 데이터) {
    문장들
}
```

이 문장은 데이터에 포함된 값들을 하나씩 꺼내서 v(이름은 마음대로 정할 수 있음)에 할당하고 그것을 가지고 문자들을 가지고 어떤 일을 할 수 있게 한다. 첫 번째 값을 가지고 와서 일을 하고, 그 다음은 두 번째 값을 가지고 와서 일을 하고, 그 다음은 세 번째 값을 가지고 와서 일을 한다. 쭉 이어진다.

다음은 아주 간단한 예로, 값들을 하나씩 꺼내서 출력한다.

```
for (i in c(1, 3, 5)) {
    print(i)
}
```

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

다음은 좋은 코드는 아닌데, for문 이해를 돕기 위해 사용한다. 1에서 10까지 값을 제곱하여 새로운 벡터를 만드는 것이다. vector() 함수는 빈 벡터를 만든다. for 문 안에서 result <- c(result, i^2) 코드를 통해서 이 벡터에 값들이 추가된다.

```
result <- vector()

for (i in 1:10) {
    result <- c(result, i^2)
}
result
```

```
 [1]   1   4   9  16  25  36  49  64  81 100
```

사실 위와 같이 하면 코드의 효율성이 떨어지고, 무엇보다 코드를 다른 사람들이 보았을 때 무엇을 하려고 하는지 이해하는 것이 쉽지 않다. 가독성이 떨어진다. R 언어는 벡터화(vectorization)를 기본으로 사용하기 때문에 다음과 같이 x^2 코드를 통해 벡터 x의 각 요소의 값을 제곱하라는 의도를 쉽게 전달하고 실행한다. 그래서 R 언어에서는 for 문이 필요한 경우 이와 같은 방식으로 가능한지 한번 더 생각하는 것이 권장된다.

```
x <- 1:10
x
##  [1]  1  2  3  4  5  6  7  8  9 10
y <- x^2
y
##  [1]   1   4   9  16  25  36  49  64  81 100
```

6.2.2 더 자세한 내용

- `if` 문 관련: Conditionals and Control Flow in R Tutorial
- `for` 문 등: A Loops in R Tutorial - Usage and Alternatives

6.3 사용자 정의 함수

R에서 사용자 정의 함수를 만드는 방법은 다음과 같다.

```
func_name <- function(인자1, 인자2) {
    문장들
}
```

`function()` 이라는 키워드 안에 사용할 파라미터를 정하고, `{}` 블록안에서 그 값들을 사용할 코드를 작성하면 된다. R 함수는 맨 마지막 표현식이 그 함수의 호출값이 된다. 함수 호출(실행시킨다고도 한다)은 이름 끝에 `()`를 쓰고, 그 안에서 파라미터에 대응하는 값을 주면 된다. 이것을 `func_name`이라는 변수(이름은 사용자 마음대로 정한다)에 할당하면 `func_name`이 이 함수의 이름이 된다.

그래서 다음 함수는 `x`, `y`라는 파라미터를 가지고 있고, 마지막 표현식이 `x + y`여서 이 값이 함수를 호출했을 때이 반환하는 값(return value, 결과값)이 된다.

```
my_sum <- function(x, y) {
    x + y
}
my_sum(1, 5)
```

```
[1] 6
```

Python 언어도 마찬가지이지만 R 언어에서 어떤 함수는 (1) 값을 반환하거나 (2) 부수효과(side effect)를 낼 수 있다. 어떤 경우에는 2가지 모두 하는 경우도 있다. 부수 효과는 실행되는 함수를 기준으로 그 외부에 영향을 주는 것을 말한다. 출력, 데이터베이스 기록, 플롯팅 등을 말한다. 값은 그야말로 값이다. 함수는 하나의 값만을 반환할 수 있다. 만약 여러 개의 값을 반환할 필요가 있는 경우에는 이것들을 하나로 묶어서 하나의 벡터로 반환시키면 된다. 벡터는 하나의 값이기 때문이다.

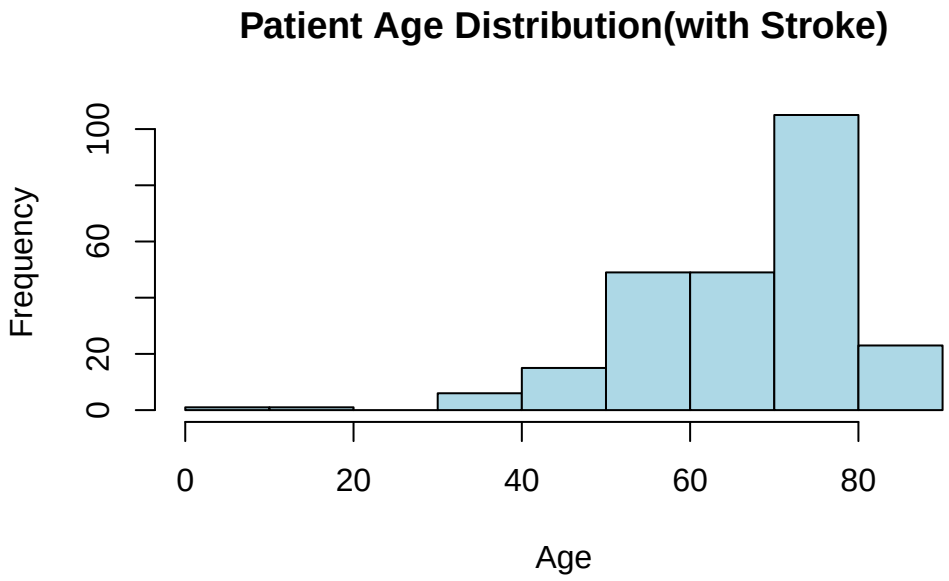
값을 반환하고 또 부수효과를 내는 함수 가운데 하나가 base R의 히스토그램을 만들어 주는 `hist()` 이다.

다음 예를 보자.

```
# 데이터 로딩
stroke_df <- readRDS("./data/stroke_df.rds")
head(stroke_df)
```

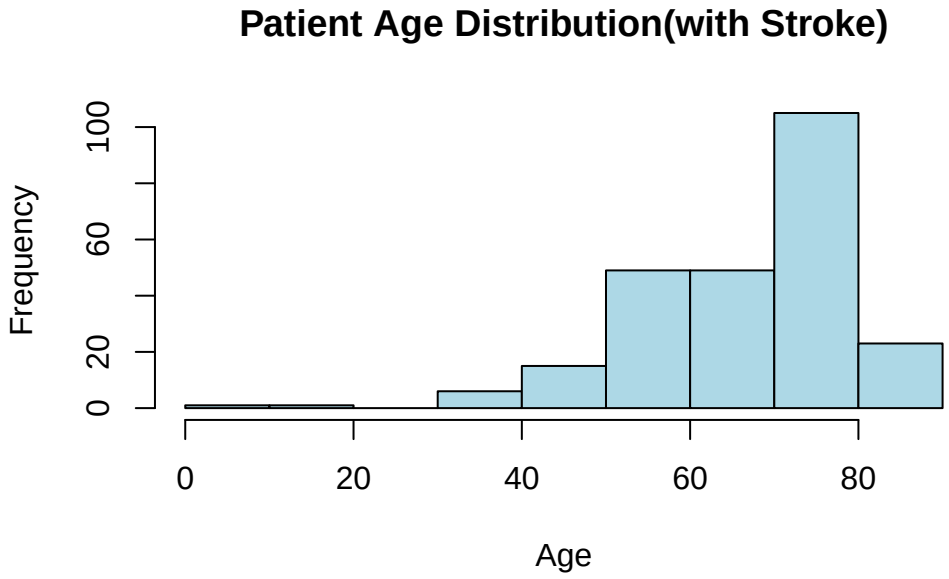
```
     id gender age hypertension heart_disease ever_married     work_type
1  9046   Male  67           No            Yes          Yes       Private
2 51676 Female  61           No             No          Yes Self-employed
3 31112   Male  80           No            Yes          Yes       Private
4 60182 Female  49           No             No          Yes       Private
5  1665 Female  79          Yes             No          Yes Self-employed
6 56669   Male  81           No             No          Yes       Private
  residence_type avg_glucose_level  bmi  smoking_status stroke
1          Urban            228.69 36.6 formerly smoked    Yes
2          Rural            202.21   NA    never smoked    Yes
3          Rural            105.92 32.5    never smoked    Yes
4          Urban            171.23 34.4          smokes    Yes
5          Rural            174.12 24.0    never smoked    Yes
6          Urban            186.21 29.0 formerly smoked    Yes
```

```
# 히스토그램 만들기
hist(stroke_df$age[stroke_df$stroke == "Yes"],
     main = "Patient Age Distribution(with Stroke)",
     xlab = "Age",
     ylab = "Frequency",
     col = "lightblue",
     border = "black")
```



위와 같이 해서 히스토그램을 만들 수 있는데, 실은 이렇게 하는 것은 `hist()` 함수가 반환하는 값들을 무시하는 셈이다. `hist()` 함수는 빈(bin)을 정하고 그 빈에 속하는 값들의 개수를 막대의 높이로 표시한 것이다. 이런 정보들이 함수를 값을 받아서 보면 알 수 있게 되어 있다.

```
hist_info <- hist(stroke_df$age[stroke_df$stroke == "Yes"],
     main = "Patient Age Distribution(with Stroke)",
     xlab = "Age",
     ylab = "Frequency",
     col = "lightblue",
     border = "black")
```




```
hist_info
```

```
$breaks
 [1]  0 10 20 30 40 50 60 70 80 90

$counts
[1]   1   1   0   6  15  49  49 105  23

$density
[1] 0.0004016064 0.0004016064 0.0000000000 0.0024096386 0.0060240964
[6] 0.0196787149 0.0196787149 0.0421686747 0.0092369478

$mids
[1]  5 15 25 35 45 55 65 75 85

$xname
[1] "stroke_df$age[stroke_df$stroke == \"Yes\"]"

$equidist
[1] TRUE
```

```
attr(,"class")
[1] "histogram"
```

? 여기서 counts라는 속성을 꺼내서 다시 사용할 수 있다.

```
hist_info$counts
```

```
[1]   1   1   0   6  15  49  49 105  23
```

6.3.1 함수형 언어의 특징을 이해하면

R 언어에서는 함수가 하나의 데이터로 취급된다. 변수에 저장할 수도 있고, 리스트나 다른 벡터로 저장할 수도 있고, 함수 안에서 다른 함수를 반환할 수도 있다. 이런 특징들을 처음부터 이해할 수도 없다고 하더라도 이해하면 할수록 복잡한 코드를 더욱 간결하게 만들 수 있다.

아주 간단한 예로 lappy()(결과를 리스트로), sapply()(결과를 가급적 간단하게) 함수를 사용법을 소개한다.

```
lapply(함수를적용할리스트, 함수이름, 이함수가필요한인자들)
sapply(함수를적용할리스트, 함수이름, 이함수가필요한인자들)
```

예를 들어 다음과 같은 데이터프레임이 있을 때, 이 데이터프레임을 구성하는 벡터들의 데이터 타입을 한꺼번에 계산할 수 있다.

```
sapply(stroke_df, typeof)
```

```
               id            gender               age      hypertension
      "character"         "integer"          "double"         "integer"
    heart_disease      ever_married         work_type    residence_type
        "integer"         "integer"         "integer"         "integer"
avg_glucose_level               bmi    smoking_status            stroke
         "double"          "double"         "integer"         "integer"
```

이런 함수를 결합하여 사용자 정의 함수를 하나 만들어 기술 통계를 한꺼번에 출력하게 할 수도 있다.

```r
# 양적인 변수
q_vars <- c("age", "avg_glucose_level", "bmi")
stats <- function(x, na.omit=FALSE) {
    if (na.omit)
        x <- x[!is.na(x)]
    m <- mean(x)
    n <- length(x)
    s <- sd(x)
    skew <- sum( (x-m)^3/s^3) / n
    kurt <- sum((x-m)^4/s^4)/n - 3
    return(c(n=n, mean=m, stdev=s, skew=skew, kurtosis=kurt))
}
# 양적인 변수를 골라서, `stats`라는 함수를 적용한다.
sapply(stroke_df[q_vars], stats, na.omit=TRUE)
```

```
                  age avg_glucose_level         bmi
n        5110.0000000       5110.000000 4909.000000
mean       43.2266145        106.147677   28.893237
stdev      22.6126467         45.283560    7.854067
skew       -0.1369789          1.571361    1.054695
kurtosis   -0.9920009          1.675830    3.355423
```

이런 식으로 프로그래밍하는 것을 함수형 프로그래밍이라고 한다. purrr 패키지는 R 함수형 프로그래밍을 지원하는 훌륭한 패키지이다.

6.3.2 더 자세한 내용

- Advanced R 책에 함수형 프로그래밍에 대한 설명이 자세히 되어 있다.
- R for Data Science(25장)에 실제 용도에 맞는 함수를 어떻게 정의하는지 등에 관해 자세히 설명한다.

7 R package 기초

R의 핵심은 아주 작다. 그 핵심을 보통 `base R`이라고 부른다. R의 가장 큰 장점의 하나는 이 `base R`에 다양한 기능을 추가하여 사용할 수 있는 패키지(package)가 다양하다는 것이다.

이런 패키지는 R 사용자들이 개발하여 공유하는 것으로 CRAN 사이트를 통해서 배포된다. 오늘 기준 22,251개의 패키지가 등록되어 있다. 이것은 공식 등록된 숫자에 불과하다. 깃허브 등에는 더 많은 패키지들이 있다. 그리고 점점 더 많은 사람들이 패키지를 개발하여 공유한다.

7.1 (RStudio에서) 패키지 설치하고 사용하는 과정

처음에 베이스 R을 설치하고 어떤 패키지가 필요해졌다고 생각해 보자. 아래 패키지를 설치하고 사용하는 과정을 간단하다.

1. 인터넷을 통해서 CRAN 등의 사이트에서 필요한 패키지를 다운로드한다.
2. R 세션에서 그 패키지가 필요하면 로딩하여 사용한다.

7.1.1 패키지 설치

RStudio는 패키지를 설치 관리하는 창을 제공한다. "Packages"라는 창을 선택한다.

RStudio는 패키지를 다운로드할 CRAN 사이트를 디폴트로 지정한다. "Install" 창에서 다운로드할 패키지 이름을 입력하면 자동으로 내 컴퓨터에 설치된다.[1]

어떤 R 패키지가 단독으로 개발되는 경우는 흔하지 않다. 오히려 다른 패키지를 의존하여 개발되는 경우도 많다. 그림 그림 7.2를 보면 아래에 `Install dependencies`가 디폴트로 체크되어 있다. 이런 경우은 그 의존하는 패키지도 함께 설치되기 때문에 굳이 신경쓸 필요는 없다.

[1] 보통 `.libPaths()`라는 함수를 실행하여 컴퓨터에서 패키지가 저장된 곳을 찾을 수 있는데, 이것을 사용할 일은 거의 없을 것이기 때문에 무시해도 된다. 단, 지정된 위치에 저장된다는 것쯤은 알고 있자.

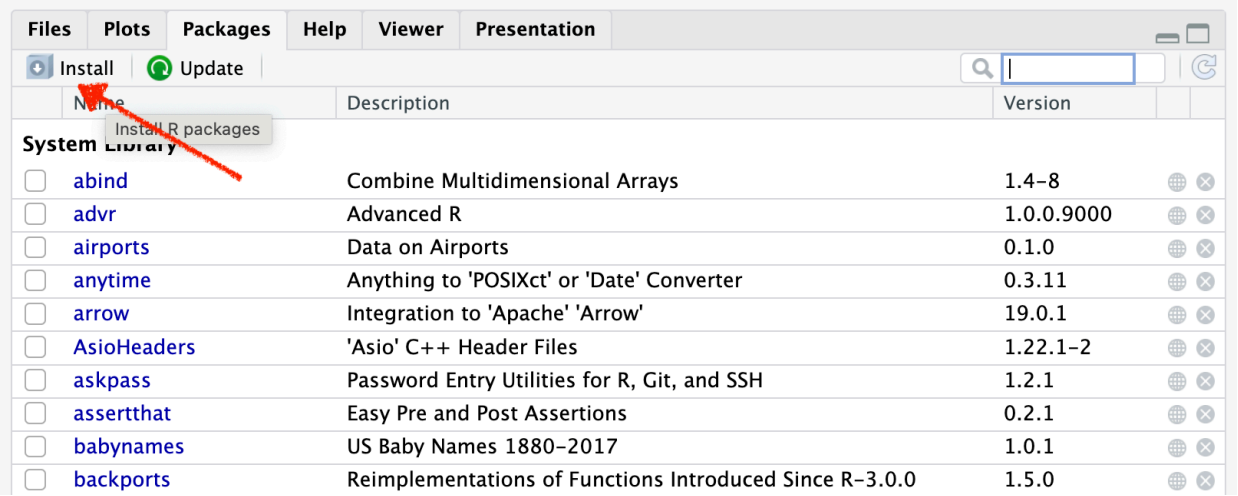


그림 7.1: Packages 창에서 Install 메뉴를 클릭한다.

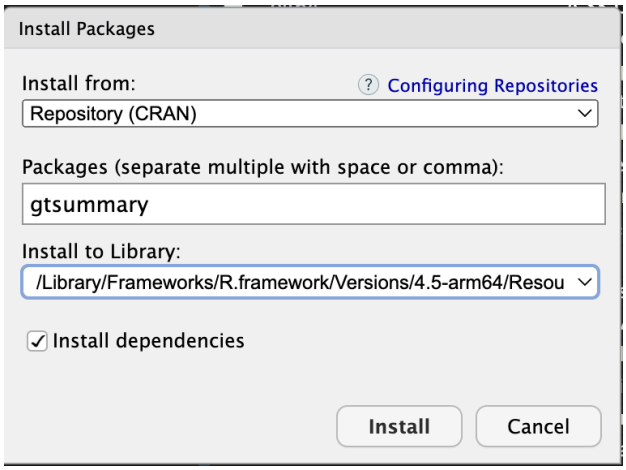


그림 7.2: 보통 중간 빈칸에 다운로드할 패키지 이름을 입력하면 드롭다운 메뉴가 보인다. 해당 패키지를 선택하고, 아래 Install 버튼을 클릭한다.

7.1.2 패키지 사용

이렇게 다운로드된 패키지를 사용하고자 할 때는 `library()` 함수를 사용한다. 함수 안에 패키지 이름을 쓰면 된다. 따옴표로 붙여도 되고 붙이지 않아도 된다.

```
> library(gtsummary)
```

그리고 R 패키지들은 자주 업데이트된다. 그림 그림 7.1에서 "Update"를 클릭하면 CRAN에 등록된 패키지의 버전과 내 컴퓨터에 저장된 패키지의 버전을 서로 비교하여 업데이트가 필요한 패키지들을 자동으로 보여준다.

7.2 어떤 패키지를 사용할 것인가?

수많은 R 패키지들이 있는데 여기서 어떤 것을 사용해야 할지 모를 수 있다. 좋은 답은 아니겠지만 R을 공부하고 사용하다 보면 대체로 알게 된다. 모르면 검색이나 ChatGPT에게 질문해도 좋다.

7.3 이름 공간(namespace)

서로 다른 저자들이 `cal_abc()`라는 함수를 여러 개 만들어 R 패키지롤 올렸고, 이런 패키지를 몇 개 다운로등하여 사용한다고 생각해 보자.

이런 경우, `library()` 함수로 나중에 로딩한 패키지의 `cal_abc()` 함수가 이전에 로딩한 패키지의 `cal_abc()`를 덮어버린다.

R `search()` 함수는 R 인터프리터가 어떤 이름을 찾아가는 순서를 말해준다.

```
library(ggplot2)
library(dplyr)
library(gtsummary)
search()
```

```
 [1] ".GlobalEnv"        "package:gtsummary" "package:dplyr"
 [4] "package:ggplot2"   "package:stats"     "package:graphics"
 [7] "package:grDevices" "package:utils"     "package:datasets"
[10] "package:methods"   "Autoloads"         "package:base"
```

이 순서를 보면 제일 먼저 `.GlobalEnv`가 오고 그 다음은 맨 마지막에 로딩된 패키지순으로 나열되어 있다.

7.4 이름 공간 연산자

`A` 패키지의 `cal_abc()`가 나중에 로딩된 `B` 패키지의 `cal_abc()` 함수에 의해 덮어진 경우에 `::`를 사용하여 `A::cal_abc()`라는 문법을 사용하여 이 패키지의 `cal_abc()`를 사용할 수 있다.

7.5 패키지에는 어떤 것이 들어 있는가?

패키지에는 보통 특수한 목적에 맞게 개발된 R 함수들이 주로 들어 있다. 간간히 데이터셋만 제공하는 패키지도 있고, 많은 경우 데이터셋, R 함수들이 같이 포함되어 있다.

7.6 패키지 사용법을 익히는 방법

어떤 패키지의 사용법을 익히고자 할 때 가장 유용한 정보를 제공하는 문서가 비니에트(vignette)이다. 비니에트는 사용법을 전체적인 맥락에서 설명하는 문서이고, 많은 R 패키지 개발자들이 비니에트를 제공한다(이에 반해 R 도움말은 (영어) 사전과 같은 역할을 한다고 생각하면 된다).

찾는 방법은 간단하다. 그림 7.1에서 패키지 이름을 클릭한다. 이 경우엔 “gtsummary”를 클릭한다.

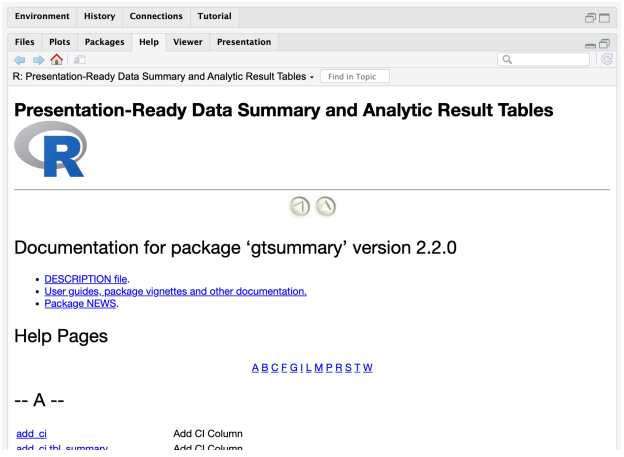


그림 7.3: 패키지 비니에트 읽기

위 그림에서 “User guides, package vignettes and other documentation.”을 클릭해서 읽는다.

또는 그림 7.3에서 “DESCRIPTION file”을 클릭한다. 그 안에 “URL” 항목이 있고, 여기에 개발 소스 코드가 있는 곳을 알려준다. 여기를 클릭한다. 대부분 코드를 공유하는 깃허브(GitHub) 사이트인 경우가 많다. 사이트를 가 보면 그림 7.4과 같이 오른쪽에 패키지 설명 사이트가 있는 경우가 많다. 여기도 배울 것이 많다.

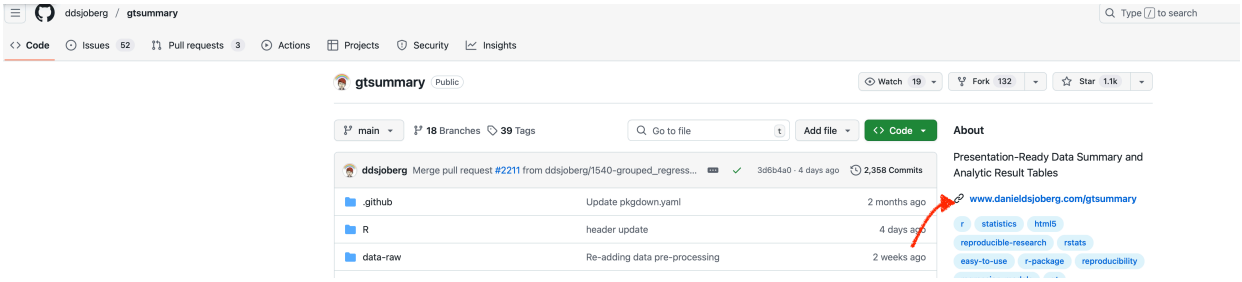


그림 7.4: 깃허브 사이트

8 R 파이프 연산자(pipe)

프로그래밍 언어에서 **파이프(pipe)**는 앞선 함수의 결과를 바로 받아서 사용할 수 있게 하여, 중간 임시 변수들을 만들지 않고 함수 등을 연결하여, 마치 문장을 연결하듯이 코딩할 수 있게 해 주는 기능을 말한다.

예를 들어 다음과 같은 코드를 보자. `data`라는 데이터프레임이 있다고 가정하자. 이것을 여러 함수를 거치면서 그 내용을 `result1`, `result2`, `result3`라는 임시 변수에 저장하고, 마지막에 `result4`를 출력한다고 하자. 이렇게 하면 중간 변수를 만들어야 하고, 가독성도 떨어진다.

```
result1 <- function1(data)
result2 <- function2(result1)
result3 <- function3(result2)
result4 <- function4(result3)
result4 # 최종 결과
```

만약 R 네이티브(native) 파이프 연산자 |>를 사용하면 다음과 같이 쓸 수 있어서 좋고, 코드를 이해하기도 편하다.

```
result <- data |>
            function1() |>
            function2() |>
            function3() |>
            function4()
```

여기서 네이트브(native)라고 하는 것을 R 언어의 핵심에 내재되어 있음을 의미한다. 따라서 별도의 외부 패키지를 사용하지 않아도 사용할 수 있다.

그런데 R 네이티브 파이프 |>는 R 4.1.0 버전부터 지원되기 시작했다. 그 이전 버전에서는 `magrittr` 패키지의 %>% 연산자를 사용하여 이 기능을 이용했다. 이 둘의 차이도 정리해 두려고 한다.

8.1 파이프 연산자의 사용

R 4.1.0 버전부터 지원되는 네이티브 파이프 연산자 |>는 다음과 같이 사용한다.

```
library(dplyr)
library(ggplot2)
mpg |>
    group_by(cyl) |>
    summarize(n = n(), mean_cty = mean(cty))
```

```
# A tibble: 4 x 3
    cyl     n mean_cty
  <int> <int>    <dbl>
1     4    81     21.0
2     5     4     20.5
3     6    79     16.2
4     8    70     12.6
```

magrittr 패키지의 %>% 연산자도 같은 방법으로 사용한다.

```
library(magrittr)
mpg %>%
    group_by(cyl) %>%
    summarize(n = n(), mean_cty = mean(cty))
```

```
# A tibble: 4 x 3
    cyl     n mean_cty
  <int> <int>    <dbl>
1     4    81     21.0
2     5     4     20.5
3     6    79     16.2
4     8    70     12.6
```

8.2 네이티브 파이프와 magrittr 파이프의 차이점

네이티브 파이프와 magrittr 파이프는 사용법은 비슷한데 약간의 차이가 있다. 대표적인 차이점은 "넘어온 그것"을 표현하는 문법이다.

- 네이티브 파이프 |>은 _를 사용하여 "넘어온 그것"을 표현한다.
- magrittr 파이프 %>%는 .을 사용하여 "넘어온 그것"을 표현한다.

네이티브 파이프와 플레이스홀더 _를 사용한 예이다.

```
mtcars |> lm(mpg ~ wt, data = _)
```

```
Call:
lm(formula = mpg ~ wt, data = mtcars)

Coefficients:
(Intercept)           wt
     37.285       -5.344
```

magrittr 파이프와 플레이스홀더 .을 사용한 예이다. magrittr 패키지는 dplyr 패키지에서 상당히 많이 사용되어 왔기 때문에 이것을 사용하는 예시를 인터넷 등에서 많이 볼 수 있을 것이다.

```
mtcars %>% lm(mpg ~ wt, data = .)
```

```
Call:
lm(formula = mpg ~ wt, data = .)

Coefficients:
(Intercept)           wt
     37.285       -5.344
```

8.3 정리

- 네이티브 파이프: |>과 _
- magrittr 파이프: %>%과 .

파이프에 대한 도움말은 다음과 같이 실행하여 볼 수 있다.

```
?`|>`


?`%>%`
```

> R에서 +, [, |> 등과 같은 특수한 기호 등에 대한 도움말 확인
>
> 위와 같이 ?를 쓰고 관련 내용을 백틱에 쓰면 된다. 백틱은 보통 키보드에서 숫자 1 앞에서 키가 있다.

Part II

II. Tidyverse와 Tidy Data

9 Tidy Data의 개념과 데이터 수집

R 언어의 핵심을 구성하는 base R은 크기도 얼마되지 않는다. 반면 여러 용도로 개발된 수많은 패키지들이 있어 효용성이 증폭된다. R의 이러한 organic한 특성은 다양한 방법으로 확장되는 장점이 있는 반면, 일관성이라는 부분에서는 취약하다.

R tidyverse는 tidy data라는 일관된 철학을 바탕으로 개발된 여러 패키지들을 일컫는 패키지로, 패키지들을 모은 일종의 메타패키지이다. 이 패키지들은 데이터 과학 작업 흐름 전반에 걸쳐 다양하게 사용되며, 나머지 R 세계에도 지대한 영향을 미쳤다. 초기 개념은 해들리 위캄의 Tidy Data(Wickham 2014)이라는 article에서 찾을 수 있다.

위 논문에서 소개된 핵심 내용은 바뀌지 않았지만, 도구는 많이 바뀌었다. 현대판 설명문은 tidyr 패키지에 포함된 비니에트는 Tidy data에 나와 있으므로, 이것을 번역하여 같이 읽어보고자 한다. 다음은 비니에트의 번역이다.

9.0.1 Data Tidying

- 데이터 분석의 80%는 data cleaning을 통해 데이터를 정리하는 데 사용된다. 그럼에도 불구하여 아직 이 분야에 대한 연구가 잘 되어 있지 않았다. 저자는 이런 측면의 일부를 논의하려고 하는데, 분석을 촉진시킬 수 있도록 데이터를 구조화하는 방법이고, 이것을 “data tidying”라고 부르려 한다.
- tidy data 원리는 데이터 안에서 데이터를 재구성하는 표준화된 방법을 제시한다. 이런 표준을 통해서 사용자들은 새로운 것을 다시 만들고, 새로 시작하거나 하는 등의 수고를 덜 수 있다. tidy dataset와 tidy data tools의 조합으로 데이터 분석을 쉽게 할 수 있다.

9.0.2 Defining tidy data

> 레오 톨스토이는 "행복한 집안은 대체로 비슷한다. 반대로 불행한 집안은 각자의 방식으로 불행하다."라고 했다.

- 가족 문제처럼, 정돈된 데이터셋은 모두 비슷한데, 정돈되지 않은 모든 데이터셋은 각자의 방식으로 문제가 있다. Tidy dataset은 데이터긔 구조와 의미를 연결하는 표준적인 방법을 제공한다.

9.0.2.1 Data structure

- 대부분의 통계 데이터셋은 행(rows)과 열(columns)로 구성된 데이터프레임이다. 열은 항상 레이블이 있고, 행은 레이블이 있는 경우도 있고, 없는 경우도 있다. 같은 데이터셋이라고 해도 여러 가지 방식으로 다른 레이아웃을 취할 수 있다.
- 다음 예를 보자.

```
library(tibble)
classroom <- tribble(
    ~name,    ~quiz1, ~quiz2, ~test1,
    "Billy",  NA,     "D",    "C",
    "Suzy",   "F",    NA,     NA,
    "Lionel", "B",    "C",    "B",
    "Jenny",  "A",    "A",    "B"
)
classroom
```

```
# A tibble: 4 x 4
  name   quiz1 quiz2 test1
  <chr>  <chr> <chr> <chr>
1 Billy  <NA>  D     C
2 Suzy   F     <NA>  <NA>
3 Lionel B     C     B
4 Jenny  A     A     B
```

- 이 데이터셋은 다음과 같이 행, 열을 바꾸어 표현할 수도 있다.

```
tribble(
    ~assessment, ~Billy, ~Suzy, ~Lionel, ~Jenny,
    "quiz1",     NA,     "F",   "B",     "A",
    "quiz2",     "D",    NA,    "C",     "A",
    "test1",     "C",    NA,    "B",     "B"
)
```

```
# A tibble: 3 x 5
  assessment Billy Suzy  Lionel Jenny
  <chr>      <chr> <chr> <chr>  <chr>
1 quiz1      <NA>  F     B      A
2 quiz2      D     <NA>  C      A
3 test1      C     <NA>  B      B
```

- 같은 데이터이지만 레이아웃은 다르다. 왜 이 두 테이블이 같은 데이터인지를 설명하기에 충분한 행과 열에 대한 개념적인 단어가 충분하지 않다. 겉보기에 더해서, 우리는 테이블에 표시되는 값들의 의미론(semamantic, 의미)을 기술할 수 있는 방법이 필요하다.

9.0.2.2 Data Semantics

> ❗ 중요
>
> - 하나의 데이터셋은 값(values)의 집합이고, 그 값들은 숫자(양적인 경우) 또는 문자열(질적인 경우)이다. 값은 두 가지 방향으로 구성되는데, 하나의 값은 하나의 **변수(variable)**와 하나의 **관측(observation)**에 속한다. 변수는 전체 관측 단위(observational units)에 대하여 키, 온도, 기간과 같은 똑같은 속성에 대한 측정값을 가진다. 하나이 관측(observation)은 하나의 관측 대상에 대하여 모든 속성에 대한 값을 가진다(한 사람, 하루, 하나의 경주 등).

앞의 `classroom` 데이터셋의 tidy 버전은 다음과 같다.

```
library(tidyr)
library(dplyr)
classroom2 <- classroom %>%
    pivot_longer(quiz1:test1, names_to = "assessment", values_to = "grade") %>%
    arrange(name, assessment)
```

```
classroom2
```

```
# A tibble: 12 x 3
   name   assessment grade
   <chr>  <chr>      <chr>
 1 Billy  quiz1      <NA>
 2 Billy  quiz2      D
 3 Billy  test1      C
 4 Jenny  quiz1      A
 5 Jenny  quiz2      A
 6 Jenny  test1      B
 7 Lionel quiz1      B
 8 Lionel quiz2      C
 9 Lionel test1      B
10 Suzy   quiz1      F
11 Suzy   quiz2      <NA>
12 Suzy   test1      <NA>
```

이렇게 하면 값, 변수, 관측이 좀 더 명확해진다. 이 데이터셋은 3개의 변수와 12개의 관측을 가진 총 36개의 값을 포함한다. 그 변수는 다음과 같다.

1. `name`: 4개의 가능한 값
2. `assessment`: 3개의 가능한 값
3. `grade`: 결측값을 어떻게 취급하지에 따라 5 또는 6개의 값

- tidy 데이터프레임은 하나의 관측이 무엇을 의미하는지 명확하게 말해준다. 이 경우 이름(`name`)과 평가(`assessment`)의 모든 조합이 단 하나의 측정된 관측을 의미한다.
- 데이터셋은 결측값이 무엇을 의미하는지에 대한 정보도 제공한다. 그것이 가능하진지, 어떤 의미를 갖는지를 말이다. 예를 들어 Billy는 첫 번째 퀴즈에 불참했는데, 점수를 만회하기 위해서 노력한 것으로 보인다. Suzy는 첫 번째 퀴즈에서 낙방해서 수업을 듣지 않기로 결정했다. Billy의 최종 점수를 계산하려면 우리는 결측값에 `F`로 채워야할 수도 있다(아니면 그가 두 번째 퀴즈를 볼 기회를 얻을 수도 있다). 그렇지만, 만약 우리가 이 수업의 `Test 1`에 평균을 구하고자 할 때는 Suzy의 구조적인 결측값은 적절한 새로운 값으로 대처하는(impute) 대신 이 값을 제외하는 것이 바람직할 수도 있다.

- 주어진 데이터셋에서 어떤 것이 관측이고 어떤 것이 변수인지 구분하는 것은 쉽다. 하지만 일반적으로 변수와 관측을 정확하게 정의하는 것은 놀랍도록 어렵다.

    - Table 1의 열이 **height**, **weight**라면 우리는 쉽게 이것을 변수라고 부를 것이다. 그런데 이것이 **dimension**(차원) 변수의 값들이라면 어떻게 처리해야 할까?
    - ……중략

- 어떤 경우에는 관측에서 복수의 레벨이 존재할 수 있다. 새로운 알러지 약물 임상 시험인 경우 3 종류의 관측 형태가 존재할 수 있다. 개인에 대한 인구학적 특성 데이터(**age**, **sex**, **race**), 각 환자에서 매일 측정되는 의학 데이터((**number of sneeze**, **rendness of eye**), 매일 측정되는 기상학적 데이터(**temperatoure**, **pollen count**) 등이 있다

- 변수들은 분석이 진행되면서 바뀔 수 있다. 종종 원 데이터의 변수들은 매우 세밀하게 쪼개져 있어서, 모델 설명에 대한 이득은 거의 없으면서 복잡성만 더 부여할 수도 있다. 예를 들어, 많은 설문지들은 내재된 특성을 더 잘 파악하기 위해서 같은 질문을 반복하곤 한다. 분석 초기 단계에서는 변수들이 질문에 대응한다. 나중에 가면, 분석자는 여러 질문들을 합계한 평균을 계산한 특성들에 초점을 맞출 수도 있다. 이렇게 하면 hierachical model을 필요로 하지 않기 때문에 분석을 간편하게 만든다. 어떤 경우 이산값이 아니라 연속형인 것처럼 다루기도 한다.

9.0.3 Tidy data

- Tidy data는 데이터셋에 내장된 의미를 그것의 구조에 매핑하는 표준적인 방법을 말한다. 행, 열, 테이블이 관측, 변수, 관측 타입과 어떻게 매칭되는지에 따라 messy인지 tidy인지 결정된다. Tidy data는 다음과 같은 특성을 가진다.

    1. 하나의 변수는 하나의 열을 구성한다.
    2. 하나의 관측은 하나의 행을 구성한다.
    3. 관측 단위의 각 형태는 하나의 테이블을 구성한다.

- 이것은 Codd가 말한 `3rd normal form`인데, 이것을 통계학적인 언어로 재설정한 것으로, 관계형 데이터베애스에서 흔히 보는 여러 연결된 데이터셋이 아니라 단 하나의 데이터셋에 초점을 주는 점이 다르다. **Messy data**란 이 규칙에 어긋나는 것을 말한다.

- Tidy data는 데이터셋을 구성하는 표준적인 방법을 제공하기 때문에 컴퓨터가 필요한 변수를 추출하는 것을 쉽게 해 준다. Table 3을 Table 1과 비교해 보자. Table 1을 사용한다면 서로 다른 변수를 추출하기 위해서 여러 전략들을 구사해야 한다. 이렇게 하다 보면 분석이 느려지고 오류가 생긴다.

- 실험 설계에 따라 fixed variable들이 먼저, measured variables들은 뒤에 두는 것이 일반적이다.

9.0.4 Tidying messy datasets

- 실제 데이터들은 tidy data가 갖춰야 하는 3가지 규칙을 갖가지 방법으로 종종 어긴다. 바로 분석에 사용할 수 있는 데이터셋을 접하는 경우보다는 그렇지 않은 경우가 대부분이다. 이 절에서는 messy dataset의 가장 흔하게 발견되는 5가지 패턴과 수정하는 방법을 설명하고자 한다. 5가지 경우는 다음과 같다.

    1. 열 헤더에 변수 이름이 아닌 값이 들어가는 경우
    2. 여러 개의 변수들이 하나의 열에 저장되는 경우
    3. 변수가 행과 열에 함께 저장되는 경우
    4. 다양한 형태의 관측 단위가 같은 테이블로 저장되는 경우
    5. 하나의 관측 단위가 여러 개의 테이블에 저장되는 경우

- 놀랍게도, 대부분 messy datasets은 몇 가지 도구 집합을 가지고도 쉽게 정렬할 수 있다. 도구는 **Pivoting(longer and wider)과 Separtint**이다. 다음 절에서 실제 messy 데이터셋을 보여주고, 이것을 정리하는 방법을 설명하려고 한다.

9.0.4.1 열 헤더에 변수 이름이 아닌 값이 들어가는 경우

- 흔히 볼 수 있는 messy dataset 가운데 하나는 주로 발표를 위해 만들어진 테이블로, 변수가 행과 열을 구성하고 있고, 열 이름이 변수 이름이 아닌 값으로 구성된 경우이다. 내가 이런 배치를 messy라고 부르지만, 어떤 경우에는 이런 방식이 극도로 유용하다. 특히 완전 교차(crossed) 설계를 가진 경우 효율적인 저장법을 제공하고, 원하는 연산들을 행렬 연산으로 표현할 수 있는 경우 매우 효율적이 된다.

> **노트**
>
> 요인 배치 방법의 하나: Crossed(교차) vs Nested(내포)
> 표 2.4는 두 요인을 교차한 실험이다. 점(‘•’)으로 표시한 20마리의 랫들은 랜덤한 방법으로 병변을 받거가 받지 않도록 할당하고, 다시 약물을 투여하거나 투여받지 않은 군으로 구분한다. 이 경우 요인들은 요인의 모든 수준을 조합한 경우에 대하여 관찰되므로 교차(crossed)되었다고 말한다.

표 2.5 케이지(cage) 요인을 병변(lesion)요인으로 내포(n=20)

	No lesion	Lesion
Cage 1	●●●●●	
Cage 2	●●●●●	
Cage 3		●●●●●
Cage 4		●●●●●

표 2.6 케이지(cage) 요인을 병변(lesion) 요인으로 내포된 설계를 다르게 표시 (n=20)

No lesion		Lesion	
Cage 1	Cage 2	Cage 3	Cage 4
●●●●●	●●●●●	●●●●●	●●●●●

교차와 반대되는 개념: Nested(내포)

표 2.6 케이지(cage) 요인을 병변(lesion) 요인으로 내포된 설계를 다르게 표시 (n=20)

No lesion		Lesion	
Cage 1	Cage 2	Cage 3	Cage 4
●●●●●	●●●●●	●●●●●	●●●●●

요인이 내포되지도 교차되지도 않을 때 중첩되었다(confounded)고 말한다.

- 다음 코드는 이런 유형의 전형을 보여준다. 이 표는 퓨 리서치 센터의 자료로, 미국에서 종교와 수입 사이의 관계를 연구한 데이터셋이다.

```
relig_income
```

```
# A tibble: 18 x 11
   religion `<$10k` `$10-20k` `$20-30k` `$30-40k` `$40-50k` `$50-75k` `$75-100k`
   <chr>      <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>       <dbl>
 1 Agnostic      27        34        60        81        76       137         122
 2 Atheist       12        27        37        52        35        70          73
 3 Buddhist      27        21        30        34        33        58          62
 4 Catholic     418       617       732       670       638      1116         949
```

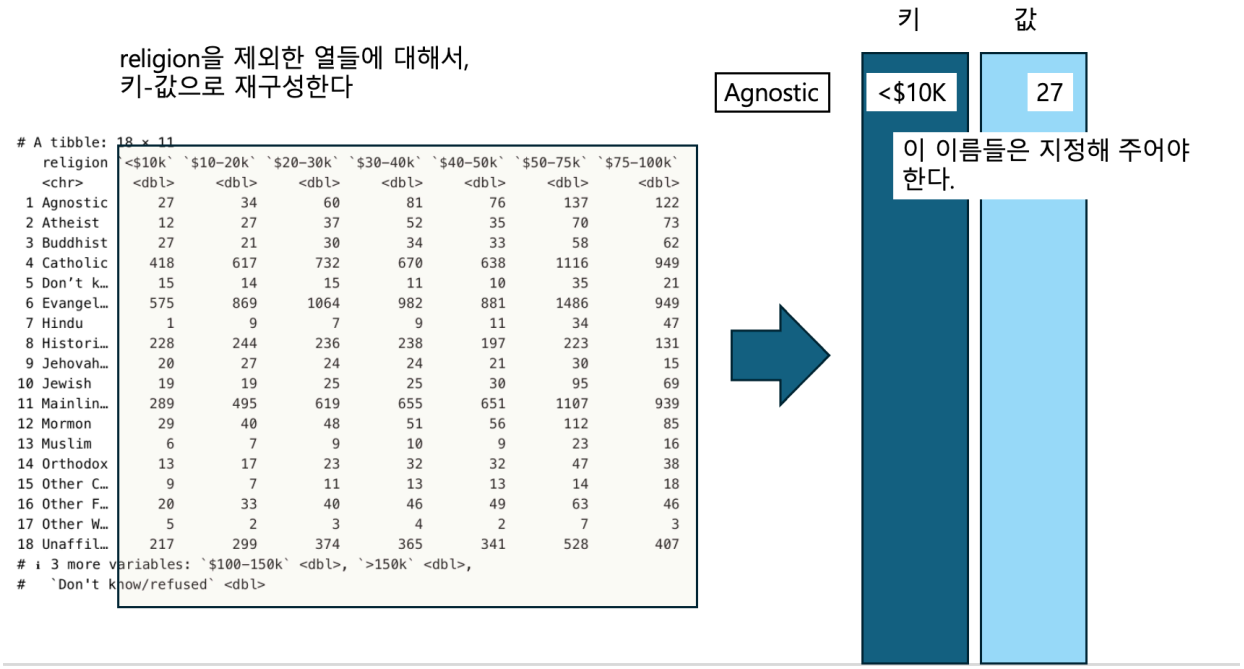
```
 5 Don't k~      15        14        15        11        10        35        21
 6 Evangel~     575       869      1064       982       881      1486       949
 7 Hindu          1         9         7         9        11        34        47
 8 Histori~     228       244       236       238       197       223       131
 9 Jehovah~      20        27        24        24        21        30        15
10 Jewish        19        19        25        25        30        95        69
11 Mainlin~     289       495       619       655       651      1107       939
12 Mormon        29        40        48        51        56       112        85
13 Muslim         6         7         9        10         9        23        16
14 Orthodox      13        17        23        32        32        47        38
15 Other C~       9         7        11        13        13        14        18
16 Other F~      20        33        40        46        49        63        46
17 Other W~       5         2         3         4         2         7         3
18 Unaffil~     217       299       374       365       341       528       407
# i 3 more variables: `$100-150k` <dbl>, `>150k` <dbl>,
#   `Don't know/refused` <dbl>
```

- 이 데이터셋은 **religion**, **income**, **frequency**라는 세 변수를 가지고 있다. 이 데이터셋을 정리하기 위해서, 우리는 변수 역할을 하지는 않는 열들을 피벗팅하여 키-값 쌍을 구성하는 2개이 열로 피벗팅할 필요가 있다. 이런 작업을 종종 넓은(wide) 데이터셋을 긴(long or tall) 데이터셋으로 바꾼다라고 말한다.
- 변수들을 피벗팅할 때, 우리는 새롭게 만들 키-값 열에 대한 이름을 줄 필요가 있다. 피벗에 사용할 열들을 정의하고 나서(이 경우 religion을 제외한 모든 열), 키가 되는 열에 이름을 줄 필요가 있는데 이것은 열 헤더의 값들에 의해서 정의되는 변수의 이름이다. 이 경우에는 `income`이라고 했다. 두 번재는 값 열에 대한 이름으로 `frequency`를 정했다.

```
relig_income %>%
    pivot_longer(-religion, names_to = "income", values_to = "frequency")
```

```
# A tibble: 180 x 3
   religion income             frequency
   <chr>    <chr>                  <dbl>
 1 Agnostic <$10k                     27
 2 Agnostic $10-20k                   34
 3 Agnostic $20-30k                   60
```

```
 4 Agnostic $30-40k                  81
 5 Agnostic $40-50k                  76
 6 Agnostic $50-75k                 137
 7 Agnostic $75-100k                122
 8 Agnostic $100-150k               109
 9 Agnostic >150k                    84
10 Agnostic Don't know/refused       96
# i 170 more rows
```




```
# A tibble: 18 × 11
   religion `<$10k` `$10-20k` `$20-30k` `$30-40k` `$40-50k` `$50-75k` `$75-100k`
   <chr>      <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>      <dbl>
 1 Agnostic      27        34        60        81        76       137        122
 2 Atheist       12        27        37        52        35        70         73
 3 Buddhist      27        21        30        34        33        58         62
 4 Catholic     418       617       732       670       638      1116        949
 5 Don't k…      15        14        15        11        10        35         21
 6 Evangel…     575       869      1064       982       881      1486        949
 7 Hindu          1         9         7         9        11        34         47
 8 Histori…     228       244       236       238       197       223        131
 9 Jehovah…      20        27        24        24        21        30         15
10 Jewish        19        19        25        25        30        95         69
11 Mainlin…     289       495       619       655       651      1107        939
12 Mormon        29        40        48        51        56       112         85
13 Muslim         6         7         9        10         9        23         16
14 Orthodox      13        17        23        32        32        47         38
15 Other C…       9         7        11        13        13        14         18
16 Other F…      20        33        40        46        49        63         46
17 Other W…       5         2         3         4         2         7          3
18 Unaffil…     217       299       374       365       341       528        407
# i 3 more variables: `$100-150k` <dbl>, `>150k` <dbl>,
#   `Don't know/refused` <dbl>
```

그림 9.1: 위 과정을 이해하기 편하게 그림으로 정리함

- 이렇게 변형된 데이터셋은 tidy dataset이라고 할 수 있다. 왜냐하면 각 열이 하나의 변수를, 각 행이 하나의 관측을 나타내기 때문이다. 이 경우 하나의 관측이란 `religion`과 `income` 조합으로 이뤄지는 하나의 인구학적 단위를 의미한다.
- 여기서 설명하는 데이터폼은 시간 흐름에 따라 일정한 간격으로 값을 기록하는데도 사용될 수 있다. 예를 들어, 아래 빌보드 데이터셋은 톱 100에 처음으로 등재된 날을 보여준다. 각 주별 순위는 75개의 열로 기록했고 `wk1`에서 `wk75`까지 열이 그것이다. 이런 형태는 tidy 하지 않지만 데이터 수집/입력에서는 유용하다. 이렇게 하면 하나의 노래가 하나의 행만을 차지하기 때문이다.(일부 번역 생략)

```
billboard
```

```
# A tibble: 317 x 79
   artist     track date.entered   wk1   wk2   wk3   wk4   wk5   wk6   wk7   wk8
   <chr>      <chr> <date>       <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
 1 2 Pac      Baby~ 2000-02-26      87    82    72    77    87    94    99    NA
 2 2Ge+her    The ~ 2000-09-02      91    87    92    NA    NA    NA    NA    NA
 3 3 Doors D~ Kryp~ 2000-04-08      81    70    68    67    66    57    54    53
 4 3 Doors D~ Loser 2000-10-21      76    76    72    69    67    65    55    59
 5 504 Boyz   Wobb~ 2000-04-15      57    34    25    17    17    31    36    49
 6 98^0       Give~ 2000-08-19      51    39    34    26    26    19     2     2
 7 A*Teens    Danc~ 2000-07-08      97    97    96    95   100    NA    NA    NA
 8 Aaliyah    I Do~ 2000-01-29      84    62    51    41    38    35    35    38
 9 Aaliyah    Try ~ 2000-03-18      59    53    38    28    21    18    16    14
10 Adams, Yo~ Open~ 2000-08-26      76    76    74    69    68    67    61    58
# i 307 more rows
# i 68 more variables: wk9 <dbl>, wk10 <dbl>, wk11 <dbl>, wk12 <dbl>,
#   wk13 <dbl>, wk14 <dbl>, wk15 <dbl>, wk16 <dbl>, wk17 <dbl>, wk18 <dbl>,
#   wk19 <dbl>, wk20 <dbl>, wk21 <dbl>, wk22 <dbl>, wk23 <dbl>, wk24 <dbl>,
#   wk25 <dbl>, wk26 <dbl>, wk27 <dbl>, wk28 <dbl>, wk29 <dbl>, wk30 <dbl>,
#   wk31 <dbl>, wk32 <dbl>, wk33 <dbl>, wk34 <dbl>, wk35 <dbl>, wk36 <dbl>,
#   wk37 <dbl>, wk38 <dbl>, wk39 <dbl>, wk40 <dbl>, wk41 <dbl>, wk42 <dbl>, ...
```

- 이 데이터셋도 정돈해 보자. 먼저 `pivot_longer()` 함수를 사용하여 넓은 데이터를 길게 만들어 보자. 우리는 `wk1`에서 `wk76` 열을 피봇팅할 할 것이고, 키 이름은 `week`, 값 이름은 `rank`로 주었다.

```
billboard2 <- billboard %>%
    pivot_longer(
        wk1:wk76,
        names_to = "week",
        values_to = "rank",
        values_drop_na = TRUE
    )
billboard2
```

```
# A tibble: 5,307 x 5
   artist  track                   date.entered week   rank
   <chr>   <chr>                   <date>       <chr> <dbl>
 1 2 Pac   Baby Don't Cry (Keep... 2000-02-26   wk1      87
 2 2 Pac   Baby Don't Cry (Keep... 2000-02-26   wk2      82
 3 2 Pac   Baby Don't Cry (Keep... 2000-02-26   wk3      72
 4 2 Pac   Baby Don't Cry (Keep... 2000-02-26   wk4      77
 5 2 Pac   Baby Don't Cry (Keep... 2000-02-26   wk5      87
 6 2 Pac   Baby Don't Cry (Keep... 2000-02-26   wk6      94
 7 2 Pac   Baby Don't Cry (Keep... 2000-02-26   wk7      99
 8 2Ge+her The Hardest Part Of ... 2000-09-02   wk1      91
 9 2Ge+her The Hardest Part Of ... 2000-09-02   wk2      87
10 2Ge+her The Hardest Part Of ... 2000-09-02   wk3      92
# i 5,297 more rows
```

- 우리는 여기서 `values_drop_na = TRUE` 인자를 사용하여 `rank` 열에 있는 결측값은 제외하게 했다. 이 데이터셋에서 결측값은 해당 노래가 차트에 없었기 때문이기 때문에, 안전하게 제외할 수 있다.
- 좀 더 클리닝 작업을 진행하여 `week` 변수의 값들에서 앞의 `wk`를 없앤 후에 숫자로 변환한다. 그리고 톱 10에 진입한 날짜 이후 몇 주가 아닌 실제 날짜로 계산했다.

```
billboard3 <- billboard2 %>%
    mutate(
        week = as.integer(gsub("wk", "", week)),
        date = as.Date(date.entered) + 7 * (week - 1),
        date.entered = NULL
    )
billboard3
```

```
# A tibble: 5,307 x 5
   artist  track                    week  rank date
   <chr>   <chr>                   <int> <dbl> <date>
 1 2 Pac   Baby Don't Cry (Keep...     1    87 2000-02-26
 2 2 Pac   Baby Don't Cry (Keep...     2    82 2000-03-04
 3 2 Pac   Baby Don't Cry (Keep...     3    72 2000-03-11
```

```
 4 2 Pac   Baby Don't Cry (Keep...     4    77 2000-03-18
 5 2 Pac   Baby Don't Cry (Keep...     5    87 2000-03-25
 6 2 Pac   Baby Don't Cry (Keep...     6    94 2000-04-01
 7 2 Pac   Baby Don't Cry (Keep...     7    99 2000-04-08
 8 2Ge+her The Hardest Part Of ...     1    91 2000-09-02
 9 2Ge+her The Hardest Part Of ...     2    87 2000-09-09
10 2Ge+her The Hardest Part Of ...     3    92 2000-09-16
# i 5,297 more rows
```

- 데이터를 정렬해 보는 것은 좋은 습관이다. `artist`, `track`, `week` 기준으로 정렬을 할 수 있을 것이다.

```
billboard3 |> arrange(artist, track, week)
```

```
# A tibble: 5,307 x 5
   artist  track                   week  rank date
   <chr>   <chr>                  <int> <dbl> <date>
 1 2 Pac   Baby Don't Cry (Keep...     1    87 2000-02-26
 2 2 Pac   Baby Don't Cry (Keep...     2    82 2000-03-04
 3 2 Pac   Baby Don't Cry (Keep...     3    72 2000-03-11
 4 2 Pac   Baby Don't Cry (Keep...     4    77 2000-03-18
 5 2 Pac   Baby Don't Cry (Keep...     5    87 2000-03-25
 6 2 Pac   Baby Don't Cry (Keep...     6    94 2000-04-01
 7 2 Pac   Baby Don't Cry (Keep...     7    99 2000-04-08
 8 2Ge+her The Hardest Part Of ...     1    91 2000-09-02
 9 2Ge+her The Hardest Part Of ...     2    87 2000-09-09
10 2Ge+her The Hardest Part Of ...     3    92 2000-09-16
# i 5,297 more rows
```

- 또는 `data`, `rank`에 따라 정렬할 수 있다.

```
billboard3 %>% arrange(date, rank)
```

```
# A tibble: 5,307 x 5
   artist   track   week  rank date
```

```
   <chr>    <chr>  <int> <dbl> <date>
 1 Lonestar Amazed     1    81 1999-06-05
 2 Lonestar Amazed     2    54 1999-06-12
 3 Lonestar Amazed     3    44 1999-06-19
 4 Lonestar Amazed     4    39 1999-06-26
 5 Lonestar Amazed     5    38 1999-07-03
 6 Lonestar Amazed     6    33 1999-07-10
 7 Lonestar Amazed     7    29 1999-07-17
 8 Amber    Sexual     1    99 1999-07-17
 9 Lonestar Amazed     8    29 1999-07-24
10 Amber    Sexual     2    99 1999-07-24
# i 5,297 more rows
```

9.0.4.2 여러 개의 변수들이 하나의 열에 저장되는 경우

- 이제 열들을 피봇팅하고 나면, 종종 키가 되는 열은 여러 개 변수 이름의 조합이 되는 경우가 있다. 무슨 뜻인고 하니 아래 `tb` 결핵 데이터셋에서 이런 현상이 나타난다. 이 데이터셋은 세계보건기구의 데이터로 여러 국가에서 결핵 발생을 기록한 것이다. 인구 그룹을 성(m, f)과 나이 조합(0-14, 15-25, 25-34, 35-44, 45-54, 55-64, unknown)으로 나눠서 정리했다.

```r
# tidyr github 사이트에 있는 tb.csv를 읽도록 코드를 변경함
tb <- as_tibble(read.csv("https://raw.githubusercontent.com/tidyverse/tidyr/refs/heads/main/vig
tb
```

```
# A tibble: 5,769 x 22
   iso2   year   m04  m514  m014 m1524 m2534 m3544 m4554 m5564   m65    mu   f04
   <chr> <int> <int> <int> <int> <int> <int> <int> <int> <int> <int> <int> <int>
 1 AD     1989    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
 2 AD     1990    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
 3 AD     1991    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
 4 AD     1992    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
 5 AD     1993    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
 6 AD     1994    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
 7 AD     1996    NA    NA     0     0     0     4     1     0     0    NA    NA
```

```
 8 AD      1997    NA    NA      0      0      1      2      2      1      6    NA    NA
 9 AD      1998    NA    NA      0      0      0      1      0      0      0    NA    NA
10 AD      1999    NA    NA      0      0      0      1      1      0      0    NA    NA
# i 5,759 more rows
# i 9 more variables: f514 <int>, f014 <int>, f1524 <int>, f2534 <int>,
#   f3544 <int>, f4554 <int>, f5564 <int>, f65 <int>, fu <int>
```

- 이 데이터셋은 `pivot_longer()`로 피벗팅해 보자.

```
tb2 <- tb %>%
    pivot_longer(
        !c(iso2, year),
        names_to = "demo",
        values_to = "n",
        values_drop_na = TRUE
    )
tb2
```

```
# A tibble: 35,750 x 4
   iso2   year demo      n
   <chr> <int> <chr> <int>
 1 AD      1996 m014      0
 2 AD      1996 m1524     0
 3 AD      1996 m2534     0
 4 AD      1996 m3544     4
 5 AD      1996 m4554     1
 6 AD      1996 m5564     0
 7 AD      1996 m65       0
 8 AD      1996 f014      0
 9 AD      1996 f1524     1
10 AD      1996 f2534     1
# i 35,740 more rows
```

- 이런 형태의 데이터셋들은 종종 특수한 문자(예를 들어, 하이픈, 언더스코어, 콜론 등)로 구분되는 열 이름을 가지거나 이 경우처럼 고정된 폭을 가지는 경우가 많다. `separate()` 함수는 복합

변수를 개별 변수로 분리하는 데 사용된다. 값의 분리를 위해 정규 표현식(regular expression)을 사용할 수도 있고, 디폴트로 이와 같은 비-알파벳-또는-숫자로 나누도록 설정되어 있다. 또는 어떤 문자의 위치를 벡터로 지정할 수 있다. 이 경우에는 첫 번재 문자 바로 다음을 기준으로 쪼개도록 한다.

```
tb3 <- tb2 %>%
    separate(demo, c("sex", "age"), 1)
tb3
```

```
# A tibble: 35,750 x 5
   iso2   year sex   age       n
   <chr> <int> <chr> <chr> <int>
 1 AD     1996 m     014       0
 2 AD     1996 m     1524      0
 3 AD     1996 m     2534      0
 4 AD     1996 m     3544      4
 5 AD     1996 m     4554      1
 6 AD     1996 m     5564      0
 7 AD     1996 m     65        0
 8 AD     1996 f     014       0
 9 AD     1996 f     1524      1
10 AD     1996 f     2534      1
# i 35,740 more rows
```

- 이렇게 해 놓으면 전체 인구수 `population`을 결합시킨 발생률을 계산하기도 편리하다. 원래의 표에는 값이 들어갈 열이 없다.
- `separate()` 과정을 `pivot_longer()` 함수에 녹여 한번에 수행시킬 수도 있다.

```
tb %>% pivot_longer(
    !c(iso2, year),
    names_to = c("sex", "age"),
    names_pattern = "(.)(.+)",
    values_to = "n",
    values_drop_na = TRUE
)
```

```
# A tibble: 35,750 x 5
   iso2   year sex   age       n
   <chr> <int> <chr> <chr> <int>
 1 AD     1996 m     014       0
 2 AD     1996 m     1524      0
 3 AD     1996 m     2534      0
 4 AD     1996 m     3544      4
 5 AD     1996 m     4554      1
 6 AD     1996 m     5564      0
 7 AD     1996 m     65        0
 8 AD     1996 f     014       0
 9 AD     1996 f     1524      1
10 AD     1996 f     2534      1
# i 35,740 more rows
```

9.0.4.3 변수가 행과 열에 함께 저장되는 경우

- Messy data에서 가장 복잡한 형태는 변수들이 행고 열 모두에 저장되는 경우이다. 아래 데이터 셋은 2010년 멕스코의 어느 기상 센터에서 수집된 날씨 정보로 Global Historical Climatology Network에서 가져왔다.

```
weather <- as_tibble(read.csv("https://raw.githubusercontent.com/tidyverse/tidyr/refs/heads/mai
weather
```

```
# A tibble: 22 x 35
   id       year month element    d1    d2    d3    d4    d5    d6    d7    d8
   <chr>   <int> <int> <chr>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
 1 MX17004  2010     1 tmax       NA    NA    NA    NA    NA    NA    NA    NA
 2 MX17004  2010     1 tmin       NA    NA    NA    NA    NA    NA    NA    NA
 3 MX17004  2010     2 tmax       NA  27.3  24.1    NA    NA    NA    NA    NA
 4 MX17004  2010     2 tmin       NA  14.4  14.4    NA    NA    NA    NA    NA
 5 MX17004  2010     3 tmax       NA    NA    NA    NA  32.1    NA    NA    NA
 6 MX17004  2010     3 tmin       NA    NA    NA    NA  14.2    NA    NA    NA
 7 MX17004  2010     4 tmax       NA    NA    NA    NA    NA    NA    NA    NA
```

```
 8 MX17004  2010     4 tmin       NA  NA    NA      NA  NA      NA    NA    NA
 9 MX17004  2010     5 tmax       NA  NA    NA      NA  NA      NA    NA    NA
10 MX17004  2010     5 tmin       NA  NA    NA      NA  NA      NA    NA    NA
# i 12 more rows
# i 23 more variables: d9 <lgl>, d10 <dbl>, d11 <dbl>, d12 <lgl>, d13 <dbl>,
#   d14 <dbl>, d15 <dbl>, d16 <dbl>, d17 <dbl>, d18 <lgl>, d19 <lgl>,
#   d20 <lgl>, d21 <lgl>, d22 <lgl>, d23 <dbl>, d24 <lgl>, d25 <dbl>,
#   d26 <dbl>, d27 <dbl>, d28 <dbl>, d29 <dbl>, d30 <dbl>, d31 <dbl>
```

- 이 데이터셋에는 변수가 `id`, `year`, `month`와 같이 개별 열에, `d1-d31`로 된 것과 같이 여러 열들에, `tmin`, `tmax` 처럼 행에 걸쳐 존재한다. 31일을 채우지 못하는 달은 마지막 날 이후 구조적인 결측값을 가진다.
- 먼저 `pivot_longer()`로 일자열들을 피벗팅한다.

```
weather2 <- weather %>%
    pivot_longer(
        d1:d31,
        names_to = "day",
        values_to = "value",
        values_drop_na = TRUE
    )
weather2
```

```
# A tibble: 66 x 6
   id       year month element day   value
   <chr>   <int> <int> <chr>   <chr> <dbl>
 1 MX17004  2010     1 tmax    d30    27.8
 2 MX17004  2010     1 tmin    d30    14.5
 3 MX17004  2010     2 tmax    d2     27.3
 4 MX17004  2010     2 tmax    d3     24.1
 5 MX17004  2010     2 tmax    d11    29.7
 6 MX17004  2010     2 tmax    d23    29.9
 7 MX17004  2010     2 tmin    d2     14.4
 8 MX17004  2010     2 tmin    d3     14.4
 9 MX17004  2010     2 tmin    d11    13.4
```

```
10 MX17004  2010     2 tmin    d23    10.7
# i 56 more rows
```

- 설명을 위해 결측값을 제외했다. 우리는 해당 월이 며칠을 가지고 있는지 잘 알고 있어서 명시적 결측값을 쉽게 다시 만들 수 있다.
- 데이터 클리닝을 좀 더 한다.

```
weather3 <- weather2 %>%
    mutate(day = as.integer(gsub("d", "", day))) %>% # day 열에서 앞의 "d" 제거
    select(id, year, month, day, element, value)
weather3
```

```
# A tibble: 66 x 6
   id       year month   day element value
   <chr>   <int> <int> <int> <chr>   <dbl>
 1 MX17004  2010     1    30 tmax     27.8
 2 MX17004  2010     1    30 tmin     14.5
 3 MX17004  2010     2     2 tmax     27.3
 4 MX17004  2010     2     3 tmax     24.1
 5 MX17004  2010     2    11 tmax     29.7
 6 MX17004  2010     2    23 tmax     29.9
 7 MX17004  2010     2     2 tmin     14.4
 8 MX17004  2010     2     3 tmin     14.4
 9 MX17004  2010     2    11 tmin     13.4
10 MX17004  2010     2    23 tmin     10.7
# i 56 more rows
```

- 이 데이터셋은 거의 정돈되어 가고 있으나, `element` 열은 아직 하나의 변수가 아니다. 이 열은 변수들의 이름을 저장하고 있다. 이 문제를 해결하기 위해서는 데이터를 넓게 만드는 `pivot_wider()` 함수가 필요하다. 이 함수는 `pivot_longer()`와 정반대의 일을 해서, `element`를 피벗팅하여 그 값들을 여러 행으로 펼친다.

```
weather3 %>%
    pivot_wider(names_from = element, values_from = value)
```

```
# A tibble: 33 x 6
   id       year month   day  tmax  tmin
   <chr>   <int> <int> <int> <dbl> <dbl>
 1 MX17004  2010     1    30  27.8  14.5
 2 MX17004  2010     2     2  27.3  14.4
 3 MX17004  2010     2     3  24.1  14.4
 4 MX17004  2010     2    11  29.7  13.4
 5 MX17004  2010     2    23  29.9  10.7
 6 MX17004  2010     3     5  32.1  14.2
 7 MX17004  2010     3    10  34.5  16.8
 8 MX17004  2010     3    16  31.1  17.6
 9 MX17004  2010     4    27  36.3  16.7
10 MX17004  2010     5    27  33.2  18.2
# i 23 more rows
```

- 이제 정도되었다. 각 열은 하나의 값으로 되어 있고, 각 행은 하루를 표현한다.

9.0.4.4 다양한 형태의 관측 단위가 같은 테이블로 저장되는 경우

- 데이터셋들은 종종 여러 수준에서 또는 서로 다른 관측 단위에서 수집되기도 한다. Tidying할 때는 각각의 관측 단위는 서로 다른 테이블에 저장되어야 한다. 이것은 각각의 팩트는 단 한 곳에서 표현되어 한다는 데이터베이스 정규화와 밀접하게 연관되어 있다. 이렇게 하지 않으면 불-일관성이 발생할 수 있기 때문이다,
- 앞에서 본 빌보드 데이터셋은 실제로 두 종류의 관측 단위를 가지고 있다. 노래와 그 노래의 주별 순위이다. 아티스트의 이름이 여러 번 반복되는 것을 통해서도 이게 드러난다.
- 이 데이터셋을 아티스트와 곡명을 저장하는 노래 데이터셋과 주별 해당 곡의 순위를 나타내는 랭킹 데이터셋으로 나눠보자. 먼저 song 데이테셋을 추출하자.

```
song <- billboard3 %>%
    distinct(artist, track) %>%
    mutate(song_id = row_number())
song
```

```
# A tibble: 317 x 3
```

```
   artist         track                   song_id
   <chr>          <chr>                     <int>
 1 2 Pac          Baby Don't Cry (Keep...       1
 2 2Ge+her        The Hardest Part Of ...       2
 3 3 Doors Down   Kryptonite                   3
 4 3 Doors Down   Loser                        4
 5 504 Boyz       Wobble Wobble                5
 6 98^0           Give Me Just One Nig...      6
 7 A*Teens        Dancing Queen                7
 8 Aaliyah        I Don't Wanna                8
 9 Aaliyah        Try Again                    9
10 Adams, Yolanda Open My Heart               10
# i 307 more rows
```

- 다음은 rank 데이터셋이다.

```
rank <- billboard3 %>%
    left_join(song, c("artist", "track")) %>%
    select(song_id, date, week, rank)
rank
```

```
# A tibble: 5,307 x 4
   song_id date        week  rank
     <int> <date>     <int> <dbl>
 1       1 2000-02-26     1    87
 2       1 2000-03-04     2    82
 3       1 2000-03-11     3    72
 4       1 2000-03-18     4    77
 5       1 2000-03-25     5    87
 6       1 2000-04-01     6    94
 7       1 2000-04-08     7    99
 8       2 2000-09-02     1    91
 9       2 2000-09-09     2    87
10       2 2000-09-16     3    92
# i 5,297 more rows
```

- 우리는 음반 판매량과 같은 “인구학적” 정보 비슷한 내용을 담은 해당 주에 대한 배경 정보를 포함하는 한 주의 데이터셋도 생각해 볼 수 있겠다.
- 정규화(normalization)은 데이터 정돈에 유용하고 일관성이 흩트러지는 것을 막는다. 그런데 관계형 데이터를 직접 다룰 수 있는 데이터 분석 도구들은 흔하지 않가여, 분석을 할 때 정규화 해제(denormalization) 또는 데이터셋의 머징을 통해 하나의 테이블으로 만들 필요가 생긴다.

9.0.4.5 하나의 관측 단위가 여러 개의 테이블에 저장되는 경우

- 단 한 종류의 관측 단위에 대한 데이터가 여러 개의 표나 파일로 나눠져 있는 경우도 흔하다. 이런 테이블이나 파일들은 또다른 변수에 의해서 분리되는 경우도 많다. 특정 연도, 특정 사람, 특정 위치에 대한 정보만을 담을 수 있다. 모두 같은 포맷으로 되어 있다면, 이 문제를 해결하는 것은 쉽다.

    1. 파일들을 하나의 테이블 리스트로 읽는다.
    2. 각 테이블에 대하여, 원본 파일 이름을 저장할 새로운 열을 추가한다(보통 파일 이름이 중요한 변수가 되기도 한다).
    3. 모든 테이블의 결합하여 하나의 테이블을 만든다.

- R `purrr` 패키지는 R로 이런 일을 하는데 안성맞춤이다. 다음 (가상) 코드는 `data` 디렉터리에 있는 `.csv` 확장자를 가지는 파일들을 하나의 벡터로 구성한다. 그런 다음 벡터의 요소마다 파일의 이름을 가져다 붙인다. 이렇게 하는 이유는 최종 데이터프레임의 각 행들이 원 소스에 따라 레이블이 붙여지도록 하기 위함이다. 마지막 `map()` 함수를 통한 루프를 사용하여 `paths`에 있는 요소마다 `read_csv`가 실행되고, 그 결과들이 `list_rbind()` 함수를 통해서 하나의 데이터프레임으로 만들어진다.

```
library(purrr)
library(readr)

paths <- dir("data", pattern = "\\.csv$", full.names = TRUE)
names(paths) <- basename(paths)

map(paths, read_csv) %>% list_rbind(names_to = "filename")
```

- 일단 하나의 테이블로 만든 다음은 필요한 추가 tidying 작업을 수행할 수 있을 것이다.
- 더 복잡한 경우는 시간에 흐름에 따라 데이터셋의 구조가 바뀔 때이다.

10 tidyr 패키지로 데이터 정리

9장에서 Tidy Data의 개념에 대해서 설명했다. 여기서는 실제로 사용되는 함수에 관해 좀 더 자세히 설명한다.

```
library(tidyr)
library(dplyr)
```

10.1 피벗팅(Pivoting)

10.1.1 피벗팅의 기본

우선 `tidyr` 패키지 cheatsheet에 있는 예시를 보면서 기초를 익히자.

다음과 같은 Untidy data가 있다.

```
table4a
```

```
# A tibble: 3 x 3
  country     `1999` `2000`
  <chr>        <dbl>  <dbl>
1 Afghanistan    745   2666
2 Brazil       37737  80488
3 China       212258 213766
```

위 데이터셋은 각 연도별 케이스 수를 나타내는 데이터이다. 값이 되어야할 연도가 열의 이름으로 들어가 있어 이 데이터는 untidy data이다.

이 데이터를 tidy data로 변환하기 위해서는 연도를 값으로 만들어야 하므로, 이 열들 1999, 2000을 묶어서 녹인 다음, `year`라는 열과 `cases`라는 열로 재배열한다.

중요

- `pivot_longer()` 함수
  - `pivot_longer(df, cols, names_to = "새이름", values_to = "새이름")`
  - 여러 열 -> (하나로 녹임) -> 펼쳐서 2개의 열로 재배열
  - 이 2개의 열에 이름을 부여한다.
    - 이름 열은 `names_to` 인자에, 값 열을 `values_to` 인자에 지정한다.

```
pivot_longer(table4a,
    cols = 2:3, names_to = "year",
    values_to = "cases"
)
```

```
# A tibble: 6 x 3
  country     year   cases
  <chr>       <chr>  <dbl>
1 Afghanistan 1999     745
2 Afghanistan 2000    2666
3 Brazil      1999   37737
4 Brazil      2000   80488
5 China       1999  212258
6 China       2000  213766
```

다음 `table2`는 각 나라의 인구와 케이스 수를 정리하는 데이터셋이다.

```
table2
```

```
# A tibble: 12 x 4
   country      year type            count
   <chr>       <dbl> <chr>           <dbl>
 1 Afghanistan  1999 cases             745
 2 Afghanistan  1999 population    19987071
 3 Afghanistan  2000 cases            2666
 4 Afghanistan  2000 population    20595360
```

```
 5 Brazil       1999 cases           37737
 6 Brazil       1999 population  172006362
 7 Brazil       2000 cases           80488
 8 Brazil       2000 population  174504898
 9 China        1999 cases          212258
10 China        1999 population 1272915272
11 China        2000 cases          213766
12 China        2000 population 1280428583
```

이 데이터셋은 변수로 사용되어야 것들이 값으로 들어가 있다. 이것을 정리하려면 `pivot_wider()` 함수를 사용한다.

! 중요

- `pivot_wider()` 함수
  - `pivot_wider(df, names_from = "이름열", values_from = "값열")`
  - 2개의 열 -> (하나로 녹임) -> 펼쳐서 여러 열로 재배열
  - 이 여러 열에 이름을 부여한다.
    * 이름으로 사용되는 값이 있는 열를 `names_from` 인자에 지정
    * 값으로 사용되는 값이 있는 열을 `values_from` 인자에 지정

```
pivot_wider(table2,
    names_from = type,
    values_from = count
)
```

```
# A tibble: 6 x 4
  country      year  cases population
  <chr>       <dbl>  <dbl>      <dbl>
1 Afghanistan  1999    745   19987071
2 Afghanistan  2000   2666   20595360
3 Brazil       1999  37737  172006362
4 Brazil       2000  80488  174504898
5 China        1999 212258 1272915272
6 China        2000 213766 1280428583
```

10.1.2 복잡한 경우의 피벗팅: Longer 예시

이 절의 자료는 `tidyr` 패키지의 `Pivoting` 비니에트를 참고했다. 이 자료에는 여러 경우의 Untidy data들과 이것을 Tidy data로 변환하는 방법들이 설명되어 있다. 번역, 요약, 주석을 추가했다.

10.1.2.1 문자열 데이터가 열 이름으로 들어간 경우

다음 `reglig_incom`를 보자.

```
relig_income |>
    head()
```

```
# A tibble: 6 x 11
  religion  `<$10k` `$10-20k` `$20-30k` `$30-40k` `$40-50k` `$50-75k` `$75-100k`
  <chr>       <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>       <dbl>
1 Agnostic       27        34        60        81        76       137         122
2 Atheist        12        27        37        52        35        70          73
3 Buddhist       27        21        30        34        33        58          62
4 Catholic      418       617       732       670       638      1116         949
5 Don't kn~      15        14        15        11        10        35          21
6 Evangeli~     575       869      1064       982       881      1486         949
# i 3 more variables: `$100-150k` <dbl>, `>150k` <dbl>,
#   `Don't know/refused` <dbl>
```

이 데이터셋은 각 종교별 소득 분포를 나타내는 데이터이다. 이 데이터셋은 열 이름에 수입 범위가 들어가 있어서 untidy data이다. 이것을 tidy data로 변환하자.

```
relig_income |>
    pivot_longer(
        cols = !religion, # 종교 이름이 들어가지 않는 열들을 모두 선택
        names_to = "income", # 수입 범위가 들어가는 열의 이름을 "income"으로 지정
        values_to = "count" # 수입 범위에 해당하는 값이 들어가는 열의 이름을 "count"로 지정
    )
```

```
# A tibble: 180 x 3
   religion income             count
   <chr>    <chr>              <dbl>
 1 Agnostic <$10k                27
 2 Agnostic $10-20k              34
 3 Agnostic $20-30k              60
 4 Agnostic $30-40k              81
 5 Agnostic $40-50k              76
 6 Agnostic $50-75k             137
 7 Agnostic $75-100k            122
 8 Agnostic $100-150k           109
 9 Agnostic >150k                84
10 Agnostic Don't know/refused   96
# i 170 more rows
```

10.1.2.2 숫자형 데이터가 열 이름으로 들어간 경우

다음 billboard 데이터셋을 보자.

```
billboard |>
    head()
```

```
# A tibble: 6 x 79
  artist      track date.entered   wk1   wk2   wk3   wk4   wk5   wk6   wk7   wk8
  <chr>       <chr> <date>       <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 2 Pac       Baby~ 2000-02-26      87    82    72    77    87    94    99    NA
2 2Ge+her     The ~ 2000-09-02      91    87    92    NA    NA    NA    NA    NA
3 3 Doors Do~ Kryp~ 2000-04-08      81    70    68    67    66    57    54    53
4 3 Doors Do~ Loser 2000-10-21      76    76    72    69    67    65    55    59
5 504 Boyz    Wobb~ 2000-04-15      57    34    25    17    17    31    36    49
6 98^0        Give~ 2000-08-19      51    39    34    26    26    19     2     2
# i 68 more variables: wk9 <dbl>, wk10 <dbl>, wk11 <dbl>, wk12 <dbl>,
#   wk13 <dbl>, wk14 <dbl>, wk15 <dbl>, wk16 <dbl>, wk17 <dbl>, wk18 <dbl>,
#   wk19 <dbl>, wk20 <dbl>, wk21 <dbl>, wk22 <dbl>, wk23 <dbl>, wk24 <dbl>,
```

```
#   wk25 <dbl>, wk26 <dbl>, wk27 <dbl>, wk28 <dbl>, wk29 <dbl>, wk30 <dbl>,
#   wk31 <dbl>, wk32 <dbl>, wk33 <dbl>, wk34 <dbl>, wk35 <dbl>, wk36 <dbl>,
#   wk37 <dbl>, wk38 <dbl>, wk39 <dbl>, wk40 <dbl>, wk41 <dbl>, wk42 <dbl>,
#   wk43 <dbl>, wk44 <dbl>, wk45 <dbl>, wk46 <dbl>, wk47 <dbl>, wk48 <dbl>, ...
```

이 데이터셋은 앨범에 들어 있는 곡의 빌보트 차트 순위를 나타내는 데이터이다. 이 데이터셋은 열 이름에 주간 순위가 들어가 있어서 untidy data이다. 이것을 tidy data로 변환하자.

```
billboard |>
    pivot_longer(
        cols = !c(artist, track, date.entered), # 아티스트, 곡, 진입일이 들어가지 않는 열들을 도
        names_to = "week", # 주간 순위가 들어가는 열의 이름을 "week"으로 지정
        values_to = "rank" # 주간 순위에 해당하는 값이 들어가는 열의 이름을 "rank"로 지정
    )
```

```
# A tibble: 24,092 x 5
   artist track                   date.entered week   rank
   <chr>  <chr>                   <date>       <chr> <dbl>
 1 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk1      87
 2 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk2      82
 3 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk3      72
 4 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk4      77
 5 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk5      87
 6 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk6      94
 7 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk7      99
 8 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk8      NA
 9 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk9      NA
10 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk10     NA
# i 24,082 more rows
```

이 데이터 변환 과정에서 어떤 곡이 어느 주에 빌보드 차트에서 빠진 경우에는 NA로 표시된다. 이렇게 의미있는 NA 값이 있는 경우에는 사용자가 제거할 수 있게 하는 옵션이 있다. values_drop_na 인자를 TRUE로 지정하면 된다.

```
billboard |>
    pivot_longer(
        cols = !c(artist, track, date.entered), # 아티스트, 곡, 진입일이 들어가지 않는 열들을 도
        names_to = "week", # 주간 순위가 들어가는 열의 이름을 "week"으로 지정
        values_to = "rank", # 주간 순위에 해당하는 값이 들어가는 열의 이름을 "rank"로 지정
        values_drop_na = TRUE # 값이 없는 경우에는 제거
    )
```

```
# A tibble: 5,307 x 5
   artist  track                   date.entered week   rank
   <chr>   <chr>                   <date>       <chr> <dbl>
 1 2 Pac   Baby Don't Cry (Keep... 2000-02-26   wk1      87
 2 2 Pac   Baby Don't Cry (Keep... 2000-02-26   wk2      82
 3 2 Pac   Baby Don't Cry (Keep... 2000-02-26   wk3      72
 4 2 Pac   Baby Don't Cry (Keep... 2000-02-26   wk4      77
 5 2 Pac   Baby Don't Cry (Keep... 2000-02-26   wk5      87
 6 2 Pac   Baby Don't Cry (Keep... 2000-02-26   wk6      94
 7 2 Pac   Baby Don't Cry (Keep... 2000-02-26   wk7      99
 8 2Ge+her The Hardest Part Of ... 2000-09-02   wk1      91
 9 2Ge+her The Hardest Part Of ... 2000-09-02   wk2      87
10 2Ge+her The Hardest Part Of ... 2000-09-02   wk3      92
# i 5,297 more rows
```

위 결과를 보면 `week` 열에 `wk1` 등과 같이 문자열로 되어 있는데, 이것은 숫자형 데이터가 되어야 맞을 것이다. 그래야 어떤 곡이 몇 주 동안 빌보드에 남아있는지 등을 계산할 수 있다.

이런 경우에 `wk1`에서 `wk`를 제거할 수 있는 옵션이 있다. `names_prefix` 인자가 그것이다.

```
billboard |>
    pivot_longer(
        cols = !c(artist, track, date.entered), # 아티스트, 곡, 진입일이 들어가지 않는 열들을 도
        names_to = "week", # 주간 순위가 들어가는 열의 이름을 "week"으로 지정
        names_prefix = "wk", # "wk" 문자열을 제거
        values_to = "rank", # 주간 순위에 해당하는 값이 들어가는 열의 이름을 "rank"로 지정
        values_drop_na = TRUE # 값이 없는 경우에는 제거
```

```
    )
```

```
# A tibble: 5,307 x 5
   artist  track                   date.entered week   rank
   <chr>   <chr>                   <date>       <chr> <dbl>
 1 2 Pac   Baby Don't Cry (Keep... 2000-02-26   1        87
 2 2 Pac   Baby Don't Cry (Keep... 2000-02-26   2        82
 3 2 Pac   Baby Don't Cry (Keep... 2000-02-26   3        72
 4 2 Pac   Baby Don't Cry (Keep... 2000-02-26   4        77
 5 2 Pac   Baby Don't Cry (Keep... 2000-02-26   5        87
 6 2 Pac   Baby Don't Cry (Keep... 2000-02-26   6        94
 7 2 Pac   Baby Don't Cry (Keep... 2000-02-26   7        99
 8 2Ge+her The Hardest Part Of ... 2000-09-02   1        91
 9 2Ge+her The Hardest Part Of ... 2000-09-02   2        87
10 2Ge+her The Hardest Part Of ... 2000-09-02   3        92
# i 5,297 more rows
```

위 결과를 잘 보면 `week` 열이 아직도 문자열로 되어 있다. 이것을 숫자형 데이터로 변환할 수 있다.

```
billboard |>
    pivot_longer(
        cols = !c(artist, track, date.entered), # 아티스트, 곡, 진입일이 들어가지 않는 열들을 도
        names_to = "week", # 주간 순위가 들어가는 열의 이름을 "week"으로 지정
        names_prefix = "wk", # "wk" 문자열을 제거
        values_to = "rank", # 주간 순위에 해당하는 값이 들어가는 열의 이름을 "rank"로 지정
        values_drop_na = TRUE # 값이 없는 경우에는 제거
    ) |>
    mutate(week = as.numeric(week))
```

```
# A tibble: 5,307 x 5
   artist  track                   date.entered  week  rank
   <chr>   <chr>                   <date>       <dbl> <dbl>
 1 2 Pac   Baby Don't Cry (Keep... 2000-02-26       1    87
 2 2 Pac   Baby Don't Cry (Keep... 2000-02-26       2    82
```

```
 3 2 Pac   Baby Don't Cry (Keep... 2000-02-26       3    72
 4 2 Pac   Baby Don't Cry (Keep... 2000-02-26       4    77
 5 2 Pac   Baby Don't Cry (Keep... 2000-02-26       5    87
 6 2 Pac   Baby Don't Cry (Keep... 2000-02-26       6    94
 7 2 Pac   Baby Don't Cry (Keep... 2000-02-26       7    99
 8 2Ge+her The Hardest Part Of ... 2000-09-02       1    91
 9 2Ge+her The Hardest Part Of ... 2000-09-02       2    87
10 2Ge+her The Hardest Part Of ... 2000-09-02       3    92
# i 5,297 more rows
```

그런데 pivot_longer() 함수에서 이 작업을 해주는 옵션이 있다. pivot_longer() 함수는 names_to에 의해서 생성되는 열들을 디폴트로 **문자열** 타입으로 만들고, values_to에 의해서 생성되는 열들의 타입은 입력되는 열에서 가장 흔한 타입이 되게 한다. 이 과정을 names_transform, values_transform 인자를 사용하여 변경할 수 있다.

```
billboard |>
    pivot_longer(
        cols = !c(artist, track, date.entered), # 아티스트, 곡, 진입일이 들어가지 않는 열들을 도
        names_to = "week", # 주간 순위가 들어가는 열의 이름을 "week"으로 지정
        values_to = "rank", # 주간 순위에 해당하는 값이 들어가는 열의 이름을 "rank"로 지정
        names_prefix = "wk", # "wk" 문자열을 제거
        names_transform = list(week = as.numeric), # 열의 이름을 지정할 때 사용되는 타입을 integ
    )
```

```
# A tibble: 24,092 x 5
   artist track                   date.entered  week  rank
   <chr>  <chr>                   <date>       <dbl> <dbl>
 1 2 Pac  Baby Don't Cry (Keep... 2000-02-26       1    87
 2 2 Pac  Baby Don't Cry (Keep... 2000-02-26       2    82
 3 2 Pac  Baby Don't Cry (Keep... 2000-02-26       3    72
 4 2 Pac  Baby Don't Cry (Keep... 2000-02-26       4    77
 5 2 Pac  Baby Don't Cry (Keep... 2000-02-26       5    87
 6 2 Pac  Baby Don't Cry (Keep... 2000-02-26       6    94
 7 2 Pac  Baby Don't Cry (Keep... 2000-02-26       7    99
 8 2 Pac  Baby Don't Cry (Keep... 2000-02-26       8    NA
```

```
 9 2 Pac  Baby Don't Cry (Keep... 2000-02-26       9    NA
10 2 Pac  Baby Don't Cry (Keep... 2000-02-26      10    NA
# i 24,082 more rows
```

10.1.2.3 여러 개의 변수들이 열 이름에 포함된 경우

다음 who 데이터셋을 보자.

```
who |>
    head()
```

```
# A tibble: 6 x 60
  country   iso2  iso3   year new_sp_m014 new_sp_m1524 new_sp_m2534 new_sp_m3544
  <chr>     <chr> <chr> <dbl>       <dbl>        <dbl>        <dbl>        <dbl>
1 Afghanis~ AF    AFG    1980          NA           NA           NA           NA
2 Afghanis~ AF    AFG    1981          NA           NA           NA           NA
3 Afghanis~ AF    AFG    1982          NA           NA           NA           NA
4 Afghanis~ AF    AFG    1983          NA           NA           NA           NA
5 Afghanis~ AF    AFG    1984          NA           NA           NA           NA
6 Afghanis~ AF    AFG    1985          NA           NA           NA           NA
# i 52 more variables: new_sp_m4554 <dbl>, new_sp_m5564 <dbl>,
#   new_sp_m65 <dbl>, new_sp_f014 <dbl>, new_sp_f1524 <dbl>,
#   new_sp_f2534 <dbl>, new_sp_f3544 <dbl>, new_sp_f4554 <dbl>,
#   new_sp_f5564 <dbl>, new_sp_f65 <dbl>, new_sn_m014 <dbl>,
#   new_sn_m1524 <dbl>, new_sn_m2534 <dbl>, new_sn_m3544 <dbl>,
#   new_sn_m4554 <dbl>, new_sn_m5564 <dbl>, new_sn_m65 <dbl>,
#   new_sn_f014 <dbl>, new_sn_f1524 <dbl>, new_sn_f2534 <dbl>, ...
```

```
names(who)
```

```
 [1] "country"      "iso2"         "iso3"         "year"         "new_sp_m014"
 [6] "new_sp_m1524" "new_sp_m2534" "new_sp_m3544" "new_sp_m4554" "new_sp_m5564"
[11] "new_sp_m65"   "new_sp_f014"  "new_sp_f1524" "new_sp_f2534" "new_sp_f3544"
[16] "new_sp_f4554" "new_sp_f5564" "new_sp_f65"   "new_sn_m014"  "new_sn_m1524"
```

```
 [21] "new_sn_m2534" "new_sn_m3544" "new_sn_m4554" "new_sn_m5564" "new_sn_m65"
 [26] "new_sn_f014"  "new_sn_f1524" "new_sn_f2534" "new_sn_f3544" "new_sn_f4554"
 [31] "new_sn_f5564" "new_sn_f65"   "new_ep_m014"  "new_ep_m1524" "new_ep_m2534"
 [36] "new_ep_m3544" "new_ep_m4554" "new_ep_m5564" "new_ep_m65"   "new_ep_f014"
 [41] "new_ep_f1524" "new_ep_f2534" "new_ep_f3544" "new_ep_f4554" "new_ep_f5564"
 [46] "new_ep_f65"   "newrel_m014"  "newrel_m1524" "newrel_m2534" "newrel_m3544"
 [51] "newrel_m4554" "newrel_m5564" "newrel_m65"   "newrel_f014"  "newrel_f1524"
 [56] "newrel_f2534" "newrel_f3544" "newrel_f4554" "newrel_f5564" "newrel_f65"
```

이 데이터셋에서 `new_` 또는 `new`는 새로운 케이이스를 의미한다. `m` `rel`, `ep`는 진단방법을 의미하고 `m`, `f`는 성별을 의미한다. `1524`, `65` 등은 나이 범위를 의미한다.

이 데이터셋을 `pivot_longer()` 함수를 사용하여 tidy data로 변환해 보자. 먼저 `names_to`에 의해서 생성되는 열들의 이름을 지정한다. 또 `names_pattern` 인자를 사용하여 열 이름에 포함된 문자열을 패턴으로 지정한다.

사실 이 기능을 사용하려면 R 언어의 정규 표현식을 알아야 한다. 정규 표현식은 문자열의 패턴을 지정하는 표현식이다. 관심이 있는 경우는 R for Data Science (2e) 15장 Regular Expressions 장을 참고하라.

> 힌트
>
> `names_pattern` 인자에 사용되는 정규 표현식은 다음과 같다.
>
> - `new_?(.*)_(.)(.*)`
>   - `new_` 또는 `new`: `_?`라고 했고 `?`는 앞의 문자가 0번 또는 1번 나오는 것을 의미한다.
>   - `(.*)`는 하나의 정규 표현식의 그룹(group)인데, 그 안에 `.*`가 들어가 있어 다음 `_` 문자가 나오기 전까지 모든 문자들을 의미한다.
>   - `(.)`도 그룹이고, `.`은 하나의 alphanumeric 문자을 의미한다.
>   - `(.*)`도 그룹으로, 모든 문자열을 의미한다.
>
> 여기서 사용된 그룹은 `names_to`에서 잡는다.

```
who |>
    pivot_longer(
        cols = !c(country, iso2, iso3, year), # 나라, 국제 표준 국가 코드, 국제 표준 국가 코드,
```

```
        names_to = c("diagnosis", "gender", "age"), # 진단방법, 성별, 나이 범위가 들어가는 열의
        names_pattern = "new_?(.*)_(.)(.*)",
        values_to = "count" # 값이 들어가는 열의 이름을 "count"로 지정
    )
```

```
# A tibble: 405,440 x 8
   country     iso2  iso3   year diagnosis gender age   count
   <chr>       <chr> <chr> <dbl> <chr>     <chr>  <chr> <dbl>
 1 Afghanistan AF    AFG    1980 sp        m      014      NA
 2 Afghanistan AF    AFG    1980 sp        m      1524     NA
 3 Afghanistan AF    AFG    1980 sp        m      2534     NA
 4 Afghanistan AF    AFG    1980 sp        m      3544     NA
 5 Afghanistan AF    AFG    1980 sp        m      4554     NA
 6 Afghanistan AF    AFG    1980 sp        m      5564     NA
 7 Afghanistan AF    AFG    1980 sp        m      65       NA
 8 Afghanistan AF    AFG    1980 sp        f      014      NA
 9 Afghanistan AF    AFG    1980 sp        f      1524     NA
10 Afghanistan AF    AFG    1980 sp        f      2534     NA
# i 405,430 more rows
```

앞에서도 설명했지만 `names_to`로 생성되는 열의 데이터타입은 문자열이다. 이것을 `names_transform` 인자를 사용하여 팩터로 변환하였다.

```
who |>
    pivot_longer(
        cols = !c(country, iso2, iso3, year), # 나라, 국제 표준 국가 코드, 국제 표준 국가 코드,
        names_to = c("diagnosis", "gender", "age"), # 진단방법, 성별, 나이 범위가 들어가는 열의
        names_pattern = "new_?(.*)_(.)(.*)",
        names_transform = list(
            diagnosis = as.factor,
            gender = as.factor,
            age = as.factor
        ),
        values_to = "count" # 값이 들어가는 열의 이름을 "count"로 지정
```

```
    ) |>
    rename(
        age_group = age
    ) |>
    tail()
```

```
# A tibble: 6 x 8
  country  iso2  iso3   year diagnosis gender age_group count
  <chr>    <chr> <chr> <dbl> <fct>     <fct>  <fct>     <dbl>
1 Zimbabwe ZW    ZWE    2013 rel       f      1524       2069
2 Zimbabwe ZW    ZWE    2013 rel       f      2534       4649
3 Zimbabwe ZW    ZWE    2013 rel       f      3544       3526
4 Zimbabwe ZW    ZWE    2013 rel       f      4554       1453
5 Zimbabwe ZW    ZWE    2013 rel       f      5564        811
6 Zimbabwe ZW    ZWE    2013 rel       f      65          725
```

10.1.2.4 하나의 행에 여러 관측이 들어간 경우

다음 household 데이터셋을 보자.

```
household
```

```
# A tibble: 5 x 5
  family dob_child1 dob_child2 name_child1 name_child2
   <int> <date>     <date>     <chr>       <chr>
1      1 1998-11-26 2000-01-29 Susan       Jose
2      2 1996-06-22 NA         Mark        <NA>
3      3 2002-07-11 2004-04-05 Sam         Seth
4      4 2004-10-10 2009-08-27 Craig       Khai
5      5 2000-12-05 2005-02-28 Parker      Gracie
```

이 데이터셋은 하나의 행에 자녀의 출생일과 그 이름을 정리할 것인데, 이것을 하나의 열로 모아서 표현한 것이다.

이 경우 names_sep 인자를 사용하여 열을 분리하고, names_to 인자에 .value라는 특수한 값을 지정하면 이 부분에 해당되는 부분을 열의 이름으로 지정한다. dob_child1, name_child2과 같은 열이 있을 때 names_sep = "_"로 지정했기 때문에 dob와 name이 .value에 들어가게 되고, child1, child2 부분이 child에 들어가게 된다.

```
household |>
    pivot_longer(
        cols = !family,
        names_to = c(".value", "child"),
        names_sep = "_",
        values_drop_na = TRUE
    )
```

```
# A tibble: 9 x 4
  family child  dob        name
   <int> <chr>  <date>     <chr>
1      1 child1 1998-11-26 Susan
2      1 child2 2000-01-29 Jose
3      2 child1 1996-06-22 Mark
4      3 child1 2002-07-11 Sam
5      3 child2 2004-04-05 Seth
6      4 child1 2004-10-10 Craig
7      4 child2 2009-08-27 Khai
8      5 child1 2000-12-05 Parker
9      5 child2 2005-02-28 Gracie
```

10.1.3 복잡한 경우의 피벗팅: Wider 예시

다음 fish_encounters 데이터셋은 물고기에 태그를 붙이고, 여러 지점(station)에서 수중 센서를 사용해 해당 물고기의 이동이 감지되었는지를 기록한 데이터이다.

```
fish_encounters
```

```
# A tibble: 114 x 3
```

```
   fish  station  seen
   <fct> <fct>   <int>
 1 4842  Release     1
 2 4842  I80_1       1
 3 4842  Lisbon      1
 4 4842  Rstr        1
 5 4842  Base_TD     1
 6 4842  BCE         1
 7 4842  BCW         1
 8 4842  BCE2        1
 9 4842  BCW2        1
10 4842  MAE         1
# i 104 more rows
```

이 경우 어떤 물고기가 어떤 지점에서 감지되었는 정리해 보고 싶을 수 있다. 이 경우 `pivot_wider()` 함수를 사용할 수 있다. `names_from`에 열로 올려 놓을 이름을 지정하다.

```
fish_encounters |>
    pivot_wider(
        names_from = station,
        values_from = seen
    )
```

```
# A tibble: 19 x 12
   fish  Release I80_1 Lisbon  Rstr Base_TD   BCE   BCW  BCE2  BCW2   MAE   MAW
   <fct>   <int> <int>  <int> <int>   <int> <int> <int> <int> <int> <int> <int>
 1 4842        1     1      1     1       1     1     1     1     1     1     1
 2 4843        1     1      1     1       1     1     1     1     1     1     1
 3 4844        1     1      1     1       1     1     1     1     1     1     1
 4 4845        1     1      1     1       1    NA    NA    NA    NA    NA    NA
 5 4847        1     1      1    NA      NA    NA    NA    NA    NA    NA    NA
 6 4848        1     1      1     1      NA    NA    NA    NA    NA    NA    NA
 7 4849        1     1     NA    NA      NA    NA    NA    NA    NA    NA    NA
 8 4850        1     1     NA     1       1     1     1    NA    NA    NA    NA
 9 4851        1     1     NA    NA      NA    NA    NA    NA    NA    NA    NA
```

```
10 4854        1     1     NA    NA      NA    NA    NA    NA    NA    NA    NA
11 4855        1     1      1     1       1    NA    NA    NA    NA    NA    NA
12 4857        1     1      1     1       1     1     1     1     1    NA    NA
13 4858        1     1      1     1       1     1     1     1     1     1     1
14 4859        1     1      1     1       1    NA    NA    NA    NA    NA    NA
15 4861        1     1      1     1       1     1     1     1     1     1     1
16 4862        1     1      1     1       1     1     1     1     1    NA    NA
17 4863        1     1     NA    NA      NA    NA    NA    NA    NA    NA    NA
18 4864        1     1     NA    NA      NA    NA    NA    NA    NA    NA    NA
19 4865        1     1      1    NA      NA    NA    NA    NA    NA    NA    NA
```

이 경우 감지되지 않은 경우에는 NA로 표시되는데, 이것을 0으로 채워 넣고 싶을 수 있다. 이 경우 values_fill 인자를 사용하면 된다.

```
fish_encounters |>
    pivot_wider(
        names_from = station,
        values_from = seen,
        values_fill = 0
    )
```

```
# A tibble: 19 x 12
   fish  Release I80_1 Lisbon  Rstr Base_TD   BCE   BCW  BCE2  BCW2   MAE   MAW
   <fct>   <int> <int>  <int> <int>   <int> <int> <int> <int> <int> <int> <int>
 1 4842        1     1      1     1       1     1     1     1     1     1     1
 2 4843        1     1      1     1       1     1     1     1     1     1     1
 3 4844        1     1      1     1       1     1     1     1     1     1     1
 4 4845        1     1      1     1       1     0     0     0     0     0     0
 5 4847        1     1      1     0       0     0     0     0     0     0     0
 6 4848        1     1      1     1       0     0     0     0     0     0     0
 7 4849        1     1      0     0       0     0     0     0     0     0     0
 8 4850        1     1      0     1       1     1     1     0     0     0     0
 9 4851        1     1      0     0       0     0     0     0     0     0     0
10 4854        1     1      0     0       0     0     0     0     0     0     0
11 4855        1     1      1     1       1     0     0     0     0     0     0
```

```
12 4857        1     1     1     1       1     1     1     1     1     0     0
13 4858        1     1     1     1       1     1     1     1     1     1     1
14 4859        1     1     1     1       1     0     0     0     0     0     0
15 4861        1     1     1     1       1     1     1     1     1     1     1
16 4862        1     1     1     1       1     1     1     1     1     0     0
17 4863        1     1     0     0       0     0     0     0     0     0     0
18 4864        1     1     0     0       0     0     0     0     0     0     0
19 4865        1     1     1     0       0     0     0     0     0     0     0
```

여러 조합으로 구성된 데이터가 있다고 해 보자. 다음 코드에서 .은 파이프 연산자를 %>%를 사용할 때 앞의 데이터프레임을 대신하는 표현식이다. 그래서 |> 연산자를 사용할 때는 작동하지 않는다.

```
production <-
    expand_grid(
        product = c("A", "B"),
        country = c("AI", "EI"),
        year = 2000:2014
    ) %>%
    as_tibble() %>%
    mutate(production = rnorm(nrow(.)))
production
```

```
# A tibble: 60 x 4
   product country  year production
   <chr>   <chr>   <int>      <dbl>
 1 A       AI       2000     -0.436
 2 A       AI       2001      1.31
 3 A       AI       2002     -0.917
 4 A       AI       2003     -0.225
 5 A       AI       2004      0.977
 6 A       AI       2005     -0.436
 7 A       AI       2006      0.559
 8 A       AI       2007      0.782
 9 A       AI       2008      0.391
10 A       AI       2009      0.328
```

```
# i 50 more rows
```

```
glimpse(production)
```

```
Rows: 60
Columns: 4
$ product    <chr> "A", "A", "A", "A", "A", "A", "A", "A", "A", "A", "A",~
$ country    <chr> "AI", "AI", "AI", "AI", "AI", "AI", "AI", "AI", "AI",~
$ year       <int> 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009,~
$ production <dbl> -0.43581771, 1.30835973, -0.91719742, -0.22504623, 0.977421~
```

이 데이터셋을 연도별로 정리해서 보여줄 필요가 있을 수 있고 다음과 같은 코드를 사용할 수 있다.

```
production |>
    pivot_wider(
        names_from = c(product, country),
        values_from = production
    )
```

```
# A tibble: 15 x 5
    year    A_AI   A_EI   B_AI    B_EI
   <int>   <dbl>  <dbl>  <dbl>   <dbl>
 1  2000 -0.436  -1.18  -0.332 -1.37
 2  2001  1.31    0.451 -0.808  0.291
 3  2002 -0.917   0.411 -2.96   1.17
 4  2003 -0.225   0.882  0.724  0.0484
 5  2004  0.977   1.02   0.406  1.20
 6  2005 -0.436  -1.43   0.297 -0.218
 7  2006  0.559   0.567 -0.427  0.263
 8  2007  0.782  -1.07  -0.865  0.671
 9  2008  0.391   0.919 -1.63   0.200
10  2009  0.328   0.250 -0.508 -0.244
11  2010  0.855  -0.401  0.865 -0.785
12  2011 -0.0933 -0.155 -0.361 -1.03
```

```
13  2012 -0.0266  0.160  0.686  0.234
14  2013  0.253  -1.41  -0.811 -0.744
15  2014  1.16    0.751 -0.583  0.837
```

이런 경우 이름을 _로 연결하여 만들고 있다. 이것을 바꾸려면 names_sep 인자를 사용한다. names_prefix 인자를 사용하면 접두사를 붙일 수 있다.

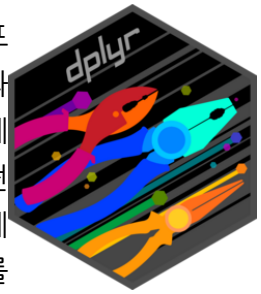
```
production |>
    pivot_wider(
        names_from = c(product, country),
        values_from = production,
        names_sep = ".",
        names_prefix = "production."
    )
```

```
# A tibble: 15 x 5
    year production.A.AI production.A.EI production.B.AI production.B.EI
   <int>           <dbl>           <dbl>           <dbl>           <dbl>
 1  2000         -0.436          -1.18           -0.332          -1.37
 2  2001          1.31            0.451          -0.808           0.291
 3  2002         -0.917           0.411          -2.96            1.17
 4  2003         -0.225           0.882           0.724           0.0484
 5  2004          0.977           1.02            0.406           1.20
 6  2005         -0.436          -1.43            0.297          -0.218
 7  2006          0.559           0.567          -0.427           0.263
 8  2007          0.782          -1.07           -0.865           0.671
 9  2008          0.391           0.919          -1.63            0.200
10  2009          0.328           0.250          -0.508          -0.244
11  2010          0.855          -0.401           0.865          -0.785
12  2011         -0.0933         -0.155          -0.361          -1.03
13  2012         -0.0266          0.160           0.686           0.234
14  2013          0.253          -1.41           -0.811          -0.744
15  2014          1.16            0.751          -0.583           0.837
```

10.2 정리

- Pivoting 비니에트에 있는 사례들을 모두 정리한 것은 아니다. 더 많은 사례를 보려면 이 링크를 참고한다.

11 dplyr로 데이터 가공



(a) dplyr: a grammar of data manipulation

R dplyr 패키지는 tidyverse를 구성하는 주요 패키지의 하나이며, R 데이터프레임을 가공하여 사용자가 원하는 형태로 변환시키는 기능을 제공한다. 개념과 사용법이 쉽고 또한 기능이 강력하여 R 사용자들이 즐겨 사용하는 도구 가운데 하나이다. d는 data.frame, plyr는 공구(tools)을 의미한다. R dplyr은 파이썬 세계의 팬더스(pandas), 폴라스(polars), SQL 세계의 DuckDB 등과 함께 스프레드시트와 같은 4각형 데이터를 처리하는 핵심 도구로 인정받고 있다. 또 이해를 넓혀 보면 관계형 데이터베이스를 다루는 SQL(structured query language)과도 깊게 연관되어 있으며, SQL을 사용하지 않아도 데이터베이스와도 소통할 수 있다.

dplyr 설치와 사용은 다음과 같은 방법 가운데 하나를 선택한다. tidyverse를 설치하고 부르면 여기에 포함되어 있어 따로 관리할 필요가 없기도 하다.

```
# 설치
install.packages("dpylr")
# 로딩
library(dplyr)
```

또는 다음과 같이 해도 된다.

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

11.1 도움이 되는 자료

`dplyr` 패키지는 워낙 인기가 많아 인터넷을 검색하면 수많은 자료를 자료를 찾을 수 있을 것이다. 그래도 가장 읽을 만하다고 보는 문서는 패키지에 내장된 비니에트와 `dplyr` 저자 등이 설명하는 아래 책의 내용이다.

- `dplyr` 패키지에 포함된 비니에트 문서들
    - Introduction to dplyr
    - Grouped data
    - Column-wise operations
    - Row-wise operations
    - Two-table verbs
    - Window functions
    - Programming with dplyr
- R for Data Science (2e) 3장 Data transformation
- dplyr CheatSheet(PDF)

11.2 dplyr을 시작하기에 앞서 알아둘 내용

dplyr에선 파이프(pipe) 연산자를 많이 사용한다. (8 참고) **파이프(pipe)**란 수도나 가스관처럼 흐름을 연결하는 구조를 말한다. R에서는 이런 연산자가 native로 지원하는 것이 없어서 처음에는 `magrittr` 이라는 패키지를 통해서 이 기능을 사용했고, `%>%`라는 연산자를 이용했다. 그러다가 나중에 파이프에 대한 요구가 증가하면서 네이티브로 지원하기 시작했고, `|>`이라는 연산자를 제공한다. 파이프의 개념은 간단하다. 앞의 연산의 결과가 다음 함수의 첫 번째 인자로 전달된다. 예를 들어, 다음과 같은 코드가 있다고 하자.

```
library(dplyr)
```

```
Attaching package: 'dplyr'
```

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

    filter, lag

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

    intersect, setdiff, setequal, union
```

```
mtcars |>
    filter(mpg > 20) |>
    select(mpg, cyl) |>
    head()
```

```
                mpg cyl
Mazda RX4      21.0   6
Mazda RX4 Wag  21.0   6
Datsun 710     22.8   4
Hornet 4 Drive 21.4   6
Merc 240D      24.4   4
Merc 230       22.8   4
```

이 코드는 다음과 같은 의미를 갖는다.

```
df1 <- filter(mtcars, mpg > 20)
df2 <- select(df1, mpg, cyl)
df3 <- head(df2)
```

위 뒤 코드는 같은 결과를 내지만, 앞의 코드의 가독성이 더 높고, 중간 변수들을 만들지 않아도 되어 메모리로 아끼고 이름들을 관리하지 않아도 된다는 장점을 가진다. 여기서는 R 네이티브 연산자인 |> 을 사용한다(%>%를 사용해도 문제가 될 것은 없다).

dplyr를 사용할 때는 non-standard evaluation(NSE)을 사용한다. 이 의미는 별도로 찾아보길 바라고, 단 알아둘 것은 열 이름을 사용할 때 **작은따옴표나 큰따옴표 없이** 바로 열 이름을 사용한다는 점이다. 이런 점은 R 콘솔에서는 아주 편리하지만 코드로 프로그래밍할 때는 부가적으로 고려해야 하는 점들이 존재한다. 그것은 Programming with dplyr 문서를 참고하길 바란다.

dplyr은 데이터프레임을 처리하는 도구이다 그래서 `dplyr`의 주요 동사(verbs)는 데이터프레임을 받아서 데이터프레임을 반환한다. 예를 들어, `filter()` 함수는 데이터프레임을 받아서 데이터프레임을 반환한다. 그래서 대부분 코드는 데이터프레임에서 시작하여 데이터프레임으로 끝난다.

dplyr의 핵심 개발자인 Hadley Wickham은 문법을 좋아한다. 굉장한 인기를 얻은 그래픽 패키지인 `ggplot2`의 gg는 `grammer of graphics`의 약자이다. `dplyr`은 `grammer of data manipulation`의 약자이다. 그는 어떤 도구의 의미를 부여할 때 문법(grammer)라는 단어를 즐겨 사용한다. `dplyr`에서는 핵심 함수들을 뭔가를 하는 동사라는 의미에서 **verbs**라고 부른다.

11.3 dplyr 개론

- 이 글은 dplyr 비니에트 Introduction to dplyr를 요약(주로 번역)한 것임.

패키지를 로딩하고, 그 안에 포함된 `starwars` 데이터셋을 살펴보자.

```
library(dplyr)
head(starwars)
```

```
# A tibble: 6 x 14
  name      height  mass hair_color skin_color eye_color birth_year sex   gender
  <chr>      <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>
1 Luke Sky~    172    77 blond      fair       blue            19   male  mascu~
2 C-3PO        167    75 <NA>       gold       yellow         112   none  mascu~
3 R2-D2         96    32 <NA>       white, bl~ red             33   none  mascu~
4 Darth Va~    202   136 none       white      yellow          41.9 male  mascu~
5 Leia Org~    150    49 brown      light      brown           19   fema~ femin~
6 Owen Lars    178   120 brown, gr~ light      blue            52   male  mascu~
# i 5 more variables: homeworld <chr>, species <chr>, films <list>,
#   vehicles <list>, starships <list>
```

11.4 단일 테이블 동사(single table verbs)

먼저 단일 테이블을 대상하는 함수들을 다룬다. 앞에서 설명한 것처럼 이들을 패키지 저자는 single table verbs라고 부른다. 이들 함수를 배울 때는 행과 열 방향을 머릿속에 그리면 좋다.

- 행
  - `filter()`: 조건에 맞는 행들을 걸러냄
  - `slice()`: 위치에 있는 행들을 선택
  - `arrange()`: 행들을 정렬
- 열
  - `select()`: 사용할 열들을 선택
  - `rename()`: 열 이름 변경
  - `mutate()`: 기존 열들의 값을 가지고 새로운 열을 추가
  - `relocate()`: 열의 위치 변경
- 행 그룹
  - `summarise()`

11.4.1 행을 대상으로 한 함수

11.4.1.1 `filter()`: 조건에 맞는 행들을 필터링

이 함수에 조건을 주면, 조건을 만족하는 행들을 걸러낸다.

다음은 `skin_color`가 `"light"`이고(AND), `eye_color`가 `"brown"` 값을 가지는 행들을 걸러낸다.

```
starwars |> filter(skin_color == "light", eye_color == "brown")
```

```
# A tibble: 7 x 14
  name      height  mass hair_color skin_color eye_color birth_year sex   gender
  <chr>      <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>
1 Leia Org~    150    49 brown      light      brown             19 fema~ femin~
2 Biggs Da~    183    84 black      light      brown             24 male  mascu~
3 Padmé Am~    185    45 brown      light      brown             46 fema~ femin~
4 Cordé        157    NA brown      light      brown             NA <NA>  <NA>
5 Dormé        165    NA brown      light      brown             NA fema~ femin~
6 Raymus A~    188    79 brown      light      brown             NA male  mascu~
7 Poe Dame~     NA    NA brown      light      brown             NA male  mascu~
# i 5 more variables: homeworld <chr>, species <chr>, films <list>,
#   vehicles <list>, starships <list>
```

11.4.1.2 arrange(): 정렬 (행들의 위치 변경)

arrange() 함수 안에는 정렬의 기준이 되는 열을 순서대로 지정한다.

```
starwars |> arrange(height, mass)
```

```
# A tibble: 87 x 14
   name     height  mass hair_color skin_color eye_color birth_year sex   gender
   <chr>     <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>
 1 Yoda         66    17 white      green      brown            896 male  mascu~
 2 Ratts T~     79    15 none       grey, blue unknown           NA male  mascu~
 3 Wicket ~     88    20 brown      brown      brown              8 male  mascu~
 4 Dud Bolt     94    45 none       blue, grey yellow            NA male  mascu~
 5 R2-D2        96    32 <NA>       white, bl~ red               33 none  mascu~
 6 R4-P17       96    NA none       silver, r~ red, blue         NA none  femin~
 7 R5-D4        97    32 <NA>       white, red red               NA none  mascu~
 8 Sebulba     112    40 none       grey, red  orange            NA male  mascu~
 9 Gasgano     122    NA none       white, bl~ black             NA male  mascu~
10 Watto       137    NA black      blue, grey yellow            NA male  mascu~
# i 77 more rows
# i 5 more variables: homeworld <chr>, species <chr>, films <list>,
#   vehicles <list>, starships <list>
```

디폴트는 오름차순이다. 내림차순으로 정렬할 때는 desc() 함수로 감싼다.

```
starwars |> arrange(desc(height))
```

```
# A tibble: 87 x 14
   name     height  mass hair_color skin_color eye_color birth_year sex   gender
   <chr>     <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>
 1 Yarael ~    264    NA none       white      yellow            NA male  mascu~
 2 Tarfful     234   136 brown      brown      blue              NA male  mascu~
 3 Lama Su     229    88 none       grey       black             NA male  mascu~
 4 Chewbac~    228   112 brown      unknown    blue             200 male  mascu~
 5 Roos Ta~    224    82 none       grey       orange            NA male  mascu~
```

```
 6 Grievous    216   159 none       brown, wh~ green, y~       NA   male  mascu~
 7 Taun We     213    NA none       grey       black           NA   fema~ femin~
 8 Rugor N~    206    NA none       green      orange          NA   male  mascu~
 9 Tion Me~    206    80 none       grey       black           NA   male  mascu~
10 Darth V~    202   136 none       white      yellow          41.9 male  mascu~
# i 77 more rows
# i 5 more variables: homeworld <chr>, species <chr>, films <list>,
#   vehicles <list>, starships <list>
```

11.4.1.3 slice(): 인덱스를 가지고 일부 행들을 선택

위치 값을 가지고 행들을 처리한다.

```
# 5번째에서 10번째 행을 잘라낸다.
starwars |> slice(5:10)
```

```
# A tibble: 6 x 14
  name      height  mass hair_color skin_color eye_color birth_year sex   gender
  <chr>      <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>
1 Leia Org~    150    49 brown      light      brown             19 fema~ femin~
2 Owen Lars    178   120 brown, gr~ light      blue              52 male  mascu~
3 Beru Whi~    165    75 brown      light      blue              47 fema~ femin~
4 R5-D4         97    32 <NA>       white, red red               NA none  mascu~
5 Biggs Da~    183    84 black      light      brown             24 male  mascu~
6 Obi-Wan ~    182    77 auburn, w~ fair       blue-gray         57 male  mascu~
# i 5 more variables: homeworld <chr>, species <chr>, films <list>,
#   vehicles <list>, starships <list>
```

비슷한 함수로 slice_head(), slice_tail()는 앞부분, 뒷부분을 잘라낸다.

```
starwars |> slice_head(n = 3)
```

```
# A tibble: 3 x 14
  name      height  mass hair_color skin_color eye_color birth_year sex   gender
```

```
  <chr>      <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>
1 Luke Sky~    172    77 blond      fair       blue              19 male  mascu~
2 C-3PO        167    75 <NA>       gold       yellow           112 none  mascu~
3 R2-D2         96    32 <NA>       white, bl~ red               33 none  mascu~
# i 5 more variables: homeworld <chr>, species <chr>, films <list>,
#   vehicles <list>, starships <list>
```

slice_sample() 함수는 무작위로 n개 행을 추출하거나 prop 비율 만큼 추출한다. 부트스트랩(bootstrap)을 수행할 때는 "복원추출"을 사용하는데, 이런 경우에는 replace = TRUE 옵션을 사용한다.

```
starwars |> slice_sample(n = 5)
```

```
# A tibble: 5 x 14
  name      height  mass hair_color skin_color eye_color birth_year sex   gender
  <chr>      <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>
1 Ratts Ty~     79    15 none       grey, blue unknown           NA male  mascu~
2 Jar Jar ~    196    66 none       orange     orange            52 male  mascu~
3 Yoda          66    17 white      green      brown            896 male  mascu~
4 Wat Tamb~    193    48 none       green, gr~ unknown           NA male  mascu~
5 Adi Gall~    184    50 none       dark       blue              NA fema~ femin~
# i 5 more variables: homeworld <chr>, species <chr>, films <list>,
#   vehicles <list>, starships <list>
```

```
starwars |> slice_sample(prop = 0.1)
```

```
# A tibble: 8 x 14
  name      height  mass hair_color skin_color eye_color birth_year sex   gender
  <chr>      <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>
1 Greedo       173    74 <NA>       green      black             44 male  mascu~
2 Poggle t~    183    80 none       green      yellow            NA male  mascu~
3 Wilhuff ~    180    NA auburn, g~ fair       blue              64 male  mascu~
4 Beru Whi~    165    75 brown      light      blue              47 fema~ femin~
5 Ben Quad~    163    65 none       grey, gre~ orange            NA male  mascu~
6 Leia Org~    150    49 brown      light      brown             19 fema~ femin~
```

```
7 Poe Dame~     NA    NA brown      light      brown            NA male  mascu~
8 Lama Su      229    88 none       grey       black            NA male  mascu~
# i 5 more variables: homeworld <chr>, species <chr>, films <list>,
#   vehicles <list>, starships <list>
```

```
starwars |> slice_sample(prop = 0.1, replace = TRUE)
```

```
# A tibble: 8 x 14
  name      height  mass hair_color skin_color eye_color birth_year sex   gender
  <chr>      <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>
1 Barriss ~    166    50 black      yellow     blue              40 fema~ femin~
2 Finis Va~    170    NA blond      fair       blue              91 male  mascu~
3 Bossk        190   113 none       green      red               53 male  mascu~
4 Quarsh P~    183    NA black      dark       brown             62 male  mascu~
5 Roos Tar~    224    82 none       grey       orange            NA male  mascu~
6 Taun We      213    NA none       grey       black             NA fema~ femin~
7 Bib Fort~    180    NA none       pale       pink              NA male  mascu~
8 Palpatine    170    75 grey       pale       yellow            82 male  mascu~
# i 5 more variables: homeworld <chr>, species <chr>, films <list>,
#   vehicles <list>, starships <list>
```

slice_min(), slice_max()는 어떤 변수의 최소, 최대값을 가지는 행들을 골라낸다.

```
starwars |>
    filter(!is.na(height)) |>
    slice_max(height, n = 3)
```

```
# A tibble: 3 x 14
  name      height  mass hair_color skin_color eye_color birth_year sex   gender
  <chr>      <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>
1 Yarael P~    264    NA none       white      yellow            NA male  mascu~
2 Tarfful      234   136 brown      brown      blue              NA male  mascu~
3 Lama Su      229    88 none       grey       black             NA male  mascu~
# i 5 more variables: homeworld <chr>, species <chr>, films <list>,
#   vehicles <list>, starships <list>
```

```
starwars |>
    filter(!is.na(height)) |>
    slice_min(height, n = 3)
```

```
# A tibble: 3 x 14
  name      height  mass hair_color skin_color eye_color birth_year sex   gender
  <chr>      <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>
1 Yoda          66    17 white      green      brown            896 male  mascu~
2 Ratts Ty~     79    15 none       grey, blue unknown           NA male  mascu~
3 Wicket S~     88    20 brown      brown      brown              8 male  mascu~
# i 5 more variables: homeworld <chr>, species <chr>, films <list>,
#   vehicles <list>, starships <list>
```

이상은 'dplyr cheatsheet에서 다음 내용을 설명한 것이다.

11.4.2 열을 대상으로 하는 함수

`select()` 함수는 관심이 되는 열들만 골라낸다. 이 함수 안에서는 아주 다양한 방식의 열 선택 방법을 구사할 수 있다.

- 정수: 열의 인덱스
- 열 이름들: 해당 열들
- `:`: 데이터프레임에서 맞붙어 있는 변수들 선택
- `!`: 해당되지 않는 변수 선택(논리적으로 NOT)
- `ends_with()`, `starts_with()`, `contains()` 등 보조 함수
- `where(참또는거짓반환하는 함수)`

```
starwars |> select(hair_color, skin_color, eye_color)
```

```
# A tibble: 87 x 3
   hair_color    skin_color  eye_color
   <chr>         <chr>       <chr>
 1 blond         fair        blue
 2 <NA>          gold        yellow
```

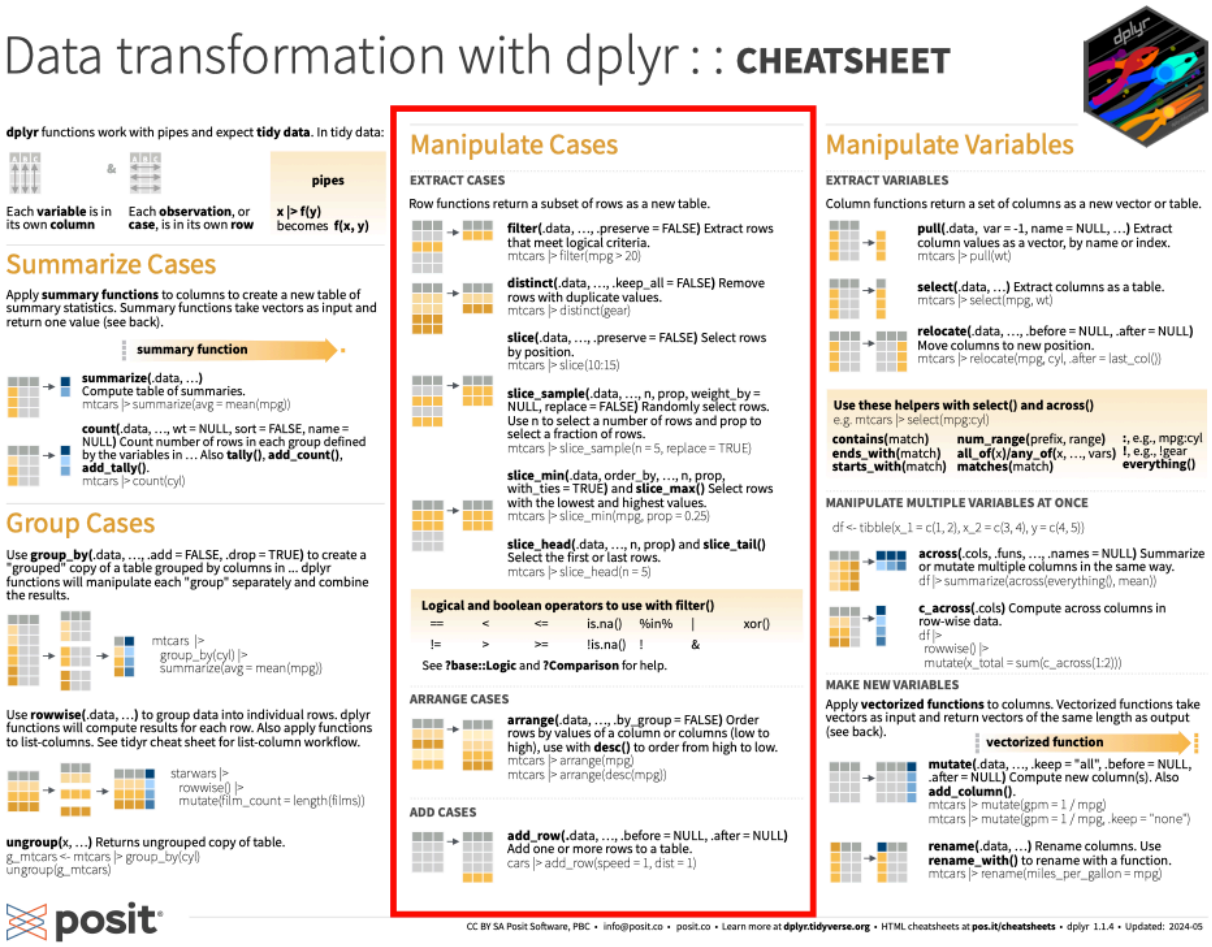


그림 11.2: Dplyr cheatsheet에서

```
 3 <NA>          white, blue red
 4 none          white       yellow
 5 brown         light       brown
 6 brown, grey   light       blue
 7 brown         light       blue
 8 <NA>          white, red  red
 9 black         light       brown
10 auburn, white fair        blue-gray
# i 77 more rows
```

```
starwars |> select(hair_color:eye_color)
```

```
# A tibble: 87 x 3
   hair_color    skin_color  eye_color
   <chr>         <chr>       <chr>
 1 blond         fair        blue
 2 <NA>          gold        yellow
 3 <NA>          white, blue red
 4 none          white       yellow
 5 brown         light       brown
 6 brown, grey   light       blue
 7 brown         light       blue
 8 <NA>          white, red  red
 9 black         light       brown
10 auburn, white fair        blue-gray
# i 77 more rows
```

```
starwars |> select(!(hair_color:eye_color))
```

```
# A tibble: 87 x 11
   name    height  mass birth_year sex   gender homeworld species films vehicles
   <chr>    <int> <dbl>      <dbl> <chr> <chr>  <chr>     <chr>   <lis> <list>
 1 Luke S~    172    77       19   male  mascu~ Tatooine  Human   <chr> <chr>
 2 C-3PO      167    75      112   none  mascu~ Tatooine  Droid   <chr> <chr>
```

```
 3 R2-D2       96    32       33   none  mascu~ Naboo     Droid   <chr> <chr>
 4 Darth ~    202   136       41.9 male  mascu~ Tatooine  Human   <chr> <chr>
 5 Leia O~    150    49       19   fema~ femin~ Alderaan  Human   <chr> <chr>
 6 Owen L~    178   120       52   male  mascu~ Tatooine  Human   <chr> <chr>
 7 Beru W~    165    75       47   fema~ femin~ Tatooine  Human   <chr> <chr>
 8 R5-D4       97    32       NA   none  mascu~ Tatooine  Droid   <chr> <chr>
 9 Biggs ~    183    84       24   male  mascu~ Tatooine  Human   <chr> <chr>
10 Obi-Wa~    182    77       57   male  mascu~ Stewjon   Human   <chr> <chr>
# i 77 more rows
# i 1 more variable: starships <list>
```

```
starwars |> select(ends_with("color"))
```

```
# A tibble: 87 x 3
   hair_color    skin_color  eye_color
   <chr>         <chr>       <chr>
 1 blond         fair        blue
 2 <NA>          gold        yellow
 3 <NA>          white, blue red
 4 none          white       yellow
 5 brown         light       brown
 6 brown, grey   light       blue
 7 brown         light       blue
 8 <NA>          white, red  red
 9 black         light       brown
10 auburn, white fair        blue-gray
# i 77 more rows
```

11.4.2.1 rename() 함수

rename() 함수는 열의 이름을 바꾼다. 새로운이름 = 이전이름 형태로 인자를 지정한다.

```
starwars |> rename(Name = name)
```

```
# A tibble: 87 x 14
   Name     height  mass hair_color skin_color eye_color birth_year sex   gender
   <chr>     <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>
 1 Luke Sk~    172    77 blond      fair       blue            19   male  mascu~
 2 C-3PO       167    75 <NA>       gold       yellow         112   none  mascu~
 3 R2-D2        96    32 <NA>       white, bl~ red             33   none  mascu~
 4 Darth V~    202   136 none       white      yellow          41.9 male  mascu~
 5 Leia Or~    150    49 brown      light      brown           19   fema~ femin~
 6 Owen La~    178   120 brown, gr~ light      blue            52   male  mascu~
 7 Beru Wh~    165    75 brown      light      blue            47   fema~ femin~
 8 R5-D4        97    32 <NA>       white, red red             NA   none  mascu~
 9 Biggs D~    183    84 black      light      brown           24   male  mascu~
10 Obi-Wan~    182    77 auburn, w~ fair       blue-gray       57   male  mascu~
# i 77 more rows
# i 5 more variables: homeworld <chr>, species <chr>, films <list>,
#   vehicles <list>, starships <list>
```

11.4.2.2 mutate(): 새로운 열 추가

mutate() 함수는 보통 기존 열에 있는 값들을 계산하여 새로운 열을 만들면서 추가한 데이터프레임을 반환한다.

```
starwars |> mutate(height_m = height / 100)
```

```
# A tibble: 87 x 15
   name     height  mass hair_color skin_color eye_color birth_year sex   gender
   <chr>     <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>
 1 Luke Sk~    172    77 blond      fair       blue            19   male  mascu~
 2 C-3PO       167    75 <NA>       gold       yellow         112   none  mascu~
 3 R2-D2        96    32 <NA>       white, bl~ red             33   none  mascu~
 4 Darth V~    202   136 none       white      yellow          41.9 male  mascu~
 5 Leia Or~    150    49 brown      light      brown           19   fema~ femin~
 6 Owen La~    178   120 brown, gr~ light      blue            52   male  mascu~
 7 Beru Wh~    165    75 brown      light      blue            47   fema~ femin~
```

```
 8 R5-D4         97    32 <NA>       white, red red             NA   none  mascu~
 9 Biggs D~     183    84 black      light      brown           24   male  mascu~
10 Obi-Wan~     182    77 auburn, w~ fair       blue-gray       57   male  mascu~
# i 77 more rows
# i 6 more variables: homeworld <chr>, species <chr>, films <list>,
#   vehicles <list>, starships <list>, height_m <dbl>
```

만약 새롭게 만든 열만을 남기고자 할 경우(이 열만 가진 데이터프레임을 만듦), `.keep = "none"` 옵션을 사용한다.

```
starwars |>
    mutate(
        height_m = height / 100,
        BMI = mass / (height_m^2),
        .keep = "none"
    )
```

```
# A tibble: 87 x 2
   height_m   BMI
      <dbl> <dbl>
 1     1.72  26.0
 2     1.67  26.9
 3     0.96  34.7
 4     2.02  33.3
 5     1.5   21.8
 6     1.78  37.9
 7     1.65  27.5
 8     0.97  34.0
 9     1.83  25.1
10     1.82  23.2
# i 77 more rows
```

11.4.2.3 `relocate()`: 열의 위치를 재조정

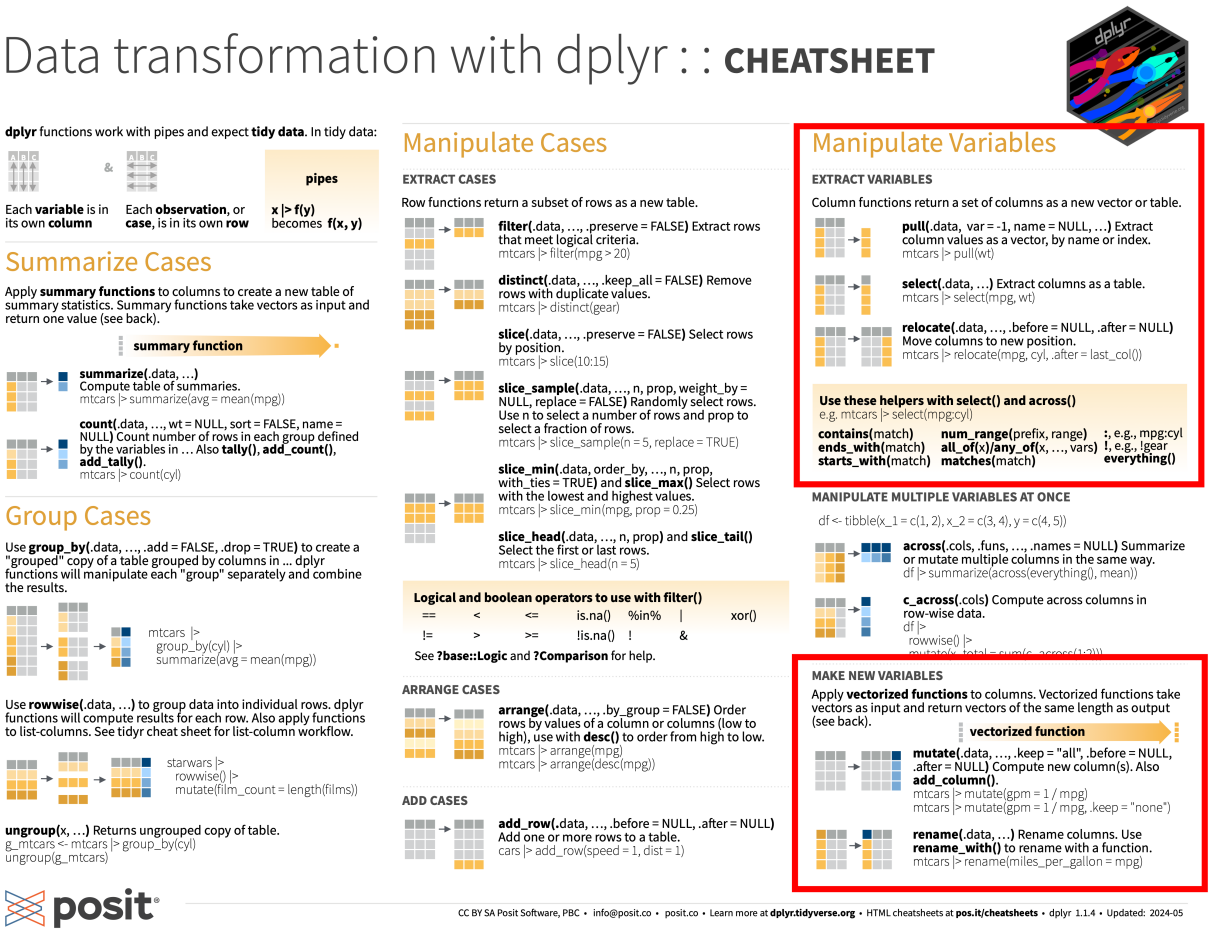
`relocate()` 함수는 열의 위치를 재조정할 때 사용한다. `.after`, `.before`라는 인자를 사용한다.

다음은 sex에서 homeworld까지의 열들을 height 열 앞으로 옮긴다.

```
starwars |> relocate(sex:homeworld, .before = height)
```

```
# A tibble: 87 x 14
   name      sex   gender homeworld height  mass hair_color skin_color eye_color
   <chr>     <chr> <chr>  <chr>      <int> <dbl> <chr>      <chr>      <chr>
 1 Luke Sky~ male  mascu~ Tatooine     172    77 blond      fair       blue
 2 C-3PO     none  mascu~ Tatooine     167    75 <NA>       gold       yellow
 3 R2-D2     none  mascu~ Naboo         96    32 <NA>       white, bl~ red
 4 Darth Va~ male  mascu~ Tatooine     202   136 none       white      yellow
 5 Leia Org~ fema~ femin~ Alderaan     150    49 brown      light      brown
 6 Owen Lars male  mascu~ Tatooine     178   120 brown, gr~ light      blue
 7 Beru Whi~ fema~ femin~ Tatooine     165    75 brown      light      blue
 8 R5-D4     none  mascu~ Tatooine      97    32 <NA>       white, red red
 9 Biggs Da~ male  mascu~ Tatooine     183    84 black      light      brown
10 Obi-Wan ~ male  mascu~ Stewjon      182    77 auburn, w~ fair       blue-gray
# i 77 more rows
# i 5 more variables: birth_year <dbl>, species <chr>, films <list>,
#   vehicles <list>, starships <list>
```

이상은 dplyr cheatsheet에서 다음 내용을 정리한 셈이다.



11.4.3 행 그룹에 대한 써머리: `summarise()` 함수

이 함수는 하나의 데이터프레임을 통계 계산을 통해 한 줄로 요약할 때 편리하다. 나중에 `group_by()`와 함께 사용할 때, 그룹별 차이를 정리하여 볼 때 편리하다.

```
starwars |>
    summarise(N = n(), mean_height = mean(height, na.rm = TRUE))
```

```
# A tibble: 1 x 2
      N mean_height
  <int>       <dbl>
1    87        175.
```

11.5 그룹화 데이터

- 이 글은 `dplyr` 비니에트 [Grouped Data](https://dplyr.tidyverse.org/articles/grouping.html 과 함께 읽으면 도움일 될 것임.

11.5.1 `group_by()` 함수로 그룹화 데이터프레임 만들기

보통 카테고리형 변수(치료군/대조군 등)을 사용하는 이유는 이 변수의 레벨에 따라 데이터셋을 분리하여 그 특징들을 서로 비교하려는 것이다. 여기서 말하는 **그룹화 데이터프레임**이란 논리적으로 전체 데이터셋을 분리한 데이터셋을 말한다.

- 그룹화 변수를 `group_by()` 함수로 넘기면 그룹화된 데이터프레임이 만들어 진다.

```
by_species <- starwars |> group_by(species)
by_sex_gender <- starwars |> group_by(sex, gender)
```

출력하면 그룹화된 데이터프레임인지 알 수 있다. Tibble인 경우 2번째 행에 `Groups:`로 시작되는 행을 볼 수 있다.

```
by_species
```

```
# A tibble: 87 x 14
# Groups:   species [38]
   name     height  mass hair_color skin_color eye_color birth_year sex   gender
   <chr>     <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>
 1 Luke Sk~    172    77 blond      fair       blue            19   male  mascu~
 2 C-3PO       167    75 <NA>       gold       yellow         112   none  mascu~
 3 R2-D2        96    32 <NA>       white, bl~ red             33   none  mascu~
 4 Darth V~    202   136 none       white      yellow          41.9 male  mascu~
 5 Leia Or~    150    49 brown      light      brown           19   fema~ femin~
 6 Owen La~    178   120 brown, gr~ light      blue            52   male  mascu~
 7 Beru Wh~    165    75 brown      light      blue            47   fema~ femin~
 8 R5-D4        97    32 <NA>       white, red red             NA   none  mascu~
 9 Biggs D~    183    84 black      light      brown           24   male  mascu~
10 Obi-Wan~    182    77 auburn, w~ fair       blue-gray       57   male  mascu~
```

```
# i 77 more rows
# i 5 more variables: homeworld <chr>, species <chr>, films <list>,
#   vehicles <list>, starships <list>
```

```
by_sex_gender
```

```
# A tibble: 87 x 14
# Groups:   sex, gender [6]
   name     height  mass hair_color skin_color eye_color birth_year sex   gender
   <chr>     <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>
 1 Luke Sk~    172    77 blond      fair       blue            19   male  mascu~
 2 C-3PO       167    75 <NA>       gold       yellow         112   none  mascu~
 3 R2-D2        96    32 <NA>       white, bl~ red             33   none  mascu~
 4 Darth V~    202   136 none       white      yellow          41.9 male  mascu~
 5 Leia Or~    150    49 brown      light      brown           19   fema~ femin~
 6 Owen La~    178   120 brown, gr~ light      blue            52   male  mascu~
 7 Beru Wh~    165    75 brown      light      blue            47   fema~ femin~
 8 R5-D4        97    32 <NA>       white, red red             NA   none  mascu~
 9 Biggs D~    183    84 black      light      brown           24   male  mascu~
10 Obi-Wan~    182    77 auburn, w~ fair       blue-gray       57   male  mascu~
# i 77 more rows
# i 5 more variables: homeworld <chr>, species <chr>, films <list>,
#   vehicles <list>, starships <list>
```

- `group_keys()` 함수를 사용하여, 각 그룹의 키를 확인할 수 있다. 즉 그룹이 나눠지는 기준 값이 무엇인지를 알 수 있다.

```
by_species |> group_keys()
```

```
# A tibble: 38 x 1
   species
   <chr>
 1 Aleena
 2 Besalisk
 3 Cerean
```

```
 4 Chagrian
 5 Clawdite
 6 Droid
 7 Dug
 8 Ewok
 9 Geonosian
10 Gungan
# i 28 more rows
```

```
by_sex_gender |> group_keys()
```

```
# A tibble: 6 x 2
  sex            gender
  <chr>          <chr>
1 female         feminine
2 hermaphroditic masculine
3 male           masculine
4 none           feminine
5 none           masculine
6 <NA>           <NA>
```

- `group_indices()` 함수를 통해서 행이 어떤 그룹에 속하는지 알 수 있다.

```
by_species |> group_indices()
```

```
 [1] 11  6  6 11 11 11 11  6 11 11 11 11 34 11 24 12 11 38 36 11 11  6 31 11 11
[26] 18 11 11  8 26 11 21 11 11 10 10 10 11 30  7 11 11 37 32 32  1 33 35 29 11
[51]  3 20 37 27 13 23 16  4 38 38 11  9 17 17 11 11 11 11  5  2 15 15 11  6 25
[76] 19 28 14 34 11 38 22 11 11 11  6 11
```

- `group_rows()` 함수로 각 그룹을 구성하는 행들을 알 수 있다.

```
by_sex_gender |> group_rows()
```

```
<list_of<integer>[6]>
[[1]]
 [1]  5  7 27 34 42 45 54 63 64 65 69 72 73 77 84 87

[[2]]
[1] 16

[[3]]
 [1]  1  4  6  9 10 11 12 13 14 15 17 19 20 21 23 24 25 26 28 29 30 31 32 33 35
[26] 36 37 38 39 40 41 43 44 46 47 48 49 50 51 52 53 55 56 57 58 61 62 66 67 68
[51] 70 71 75 76 78 79 80 82 83 85

[[4]]
[1] 74

[[5]]
[1]  2  3  8 22 86

[[6]]
[1] 18 59 60 81
```

- `group_vars()`를 통해서 그룹핑 변수 이름을 확인한다.

```
by_species |> group_vars()
```

```
[1] "species"
```

```
by_sex_gender |> group_vars()
```

```
[1] "sex"    "gender"
```

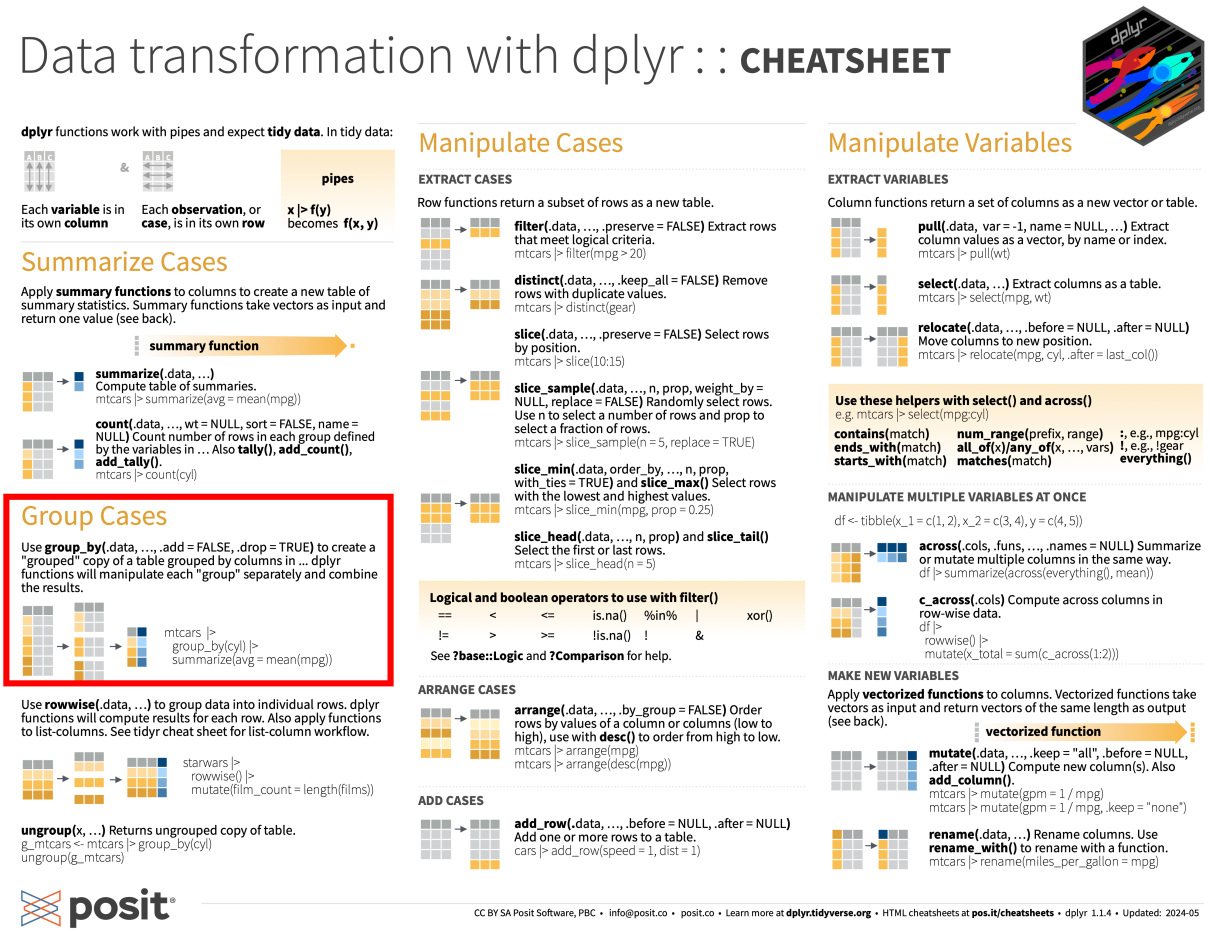
- (이미) 그룹화된 데이터프레임에 대하여 `group_by()` 함수를 적용하면 기존 그룹핑이 해제되고 새로운 변수에 따라 다시 그룹핑된다. 이렇게 새롭게 그룹핑하는 것이 아니라 추가하려고 하면 `.add = TRUE` 인자를 사용한다. 그룹핑을 해제하려면 `ungroup()` 함수를 사용한다.

그룹화 데이터프레임을 사용하면 다음과 같이 그룹에 대한 써머리를 한 문장으로 간결하게 정리할 수 있다.

```
starwars |>
    group_by(sex) |>
    summarise(N = n(), mean_height = mean(height, na.rm = TRUE)) |>
    knitr::kable()
```

sex	N	mean_height
female	16	171.5714
hermaphroditic	1	175.0000
male	60	179.1228
none	6	131.2000
NA	4	175.0000

이상은 dplyr cheatsheet에서 다음 부분을 설명했다.



11.6 여러 열에 대한 오퍼레이션: `across()`을 중심으로

- 도움이 되는 자료
  - Column-wise operations
  - R for Data Science (2e): 26장 Iteration

```
glimpse(penguins)
```

```
Rows: 344
Columns: 8
$ species     <fct> Adelie, Adelie, Adelie, Adelie, Adelie, Adelie, Adelie, Ad~
```

```
$ island      <fct> Torgersen, Torgersen, Torgersen, Torgersen, Torgersen, Tor~
$ bill_len    <dbl> 39.1, 39.5, 40.3, NA, 36.7, 39.3, 38.9, 39.2, 34.1, 42.0, ~
$ bill_dep    <dbl> 18.7, 17.4, 18.0, NA, 19.3, 20.6, 17.8, 19.6, 18.1, 20.2, ~
$ flipper_len <int> 181, 186, 195, NA, 193, 190, 181, 195, 193, 190, 186, 180,~
$ body_mass   <int> 3750, 3800, 3250, NA, 3450, 3650, 3625, 4675, 3475, 4250, ~
$ sex         <fct> male, female, female, NA, female, male, female, male, NA, ~
$ year        <int> 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007~
```

`across()` 함수를 이해하고 잘 활용하려면 약간의 배경 지식이 필요하다. 먼저 **함수형 프로그래밍 (functional programming)** 기법이다.

베이스 R에서 `lapply()`, `sapply()` 함수는 대표적인 함수형 프로그래밍 함수이다. 이 함수들은 데이터 프레임이나 리스트에 대하여 반복적으로 함수를 적용하여 결과를 반환한다. 예를 들어, 다음과 같은 데이터 프레임이 있다고 하자.

```
df <- data.frame(
    x = 1:3,
    y = 4:6,
    z = 7:9
)
```

이 데이터 프레임에 대하여 `lapply()` 함수를 적용하면 다음과 같은 결과를 얻을 수 있다. 두 번째 인자로 각 열에 대하여 적용할 함수(이름)을 전달한다는 점을 주의한다. 이처럼 함수를 하나의 값으로 사용하여 프로그래밍하는 기법을 **함수형 프로그래밍**이라고 한다.

```
lapply(df, mean)
```

```
$x
[1] 2

$y
[1] 5

$z
[1] 8
```

sapply() 함수는 이 결과를 벡터로 반환한다.

```
sapply(df, mean)
```

```
x y z
2 5 8
```

lappy(), sapply() 함수의 두 번째 인자는 함수이다. 이 함수는 데이터 프레임의 각 열에 대하여 적용된다. 만약 우리가 원하는 것을 계산하는 함수가 없으면 직접 만들어서 사용할 수 있고, lapply(), sapply() 함수 안에서 바로 정의하여 사용할 수 있는데, 이런 경우 이름이 없는 함수인 익명 함수(anonymous function)를 사용한다.

R에서 익명 함수는 베이스 R과 특정 패키지를 이용하는 방법들이 존재했는데, R 4.1.0 버전부터는 네이티브로 지원한다. Tidyverse에서도 purrr 패키지를 통해서 R 포뮬러(~)을 이용한 익명 함수를 사용했었는데, 이제는 네이티브 익명함수를 사용하는 쪽으로 권장되고 있다.

> **i** R에서 익명 함수 만들기
>
> 버전 4.1.0 이전에서 베이스 R로 익명함수는 다음과 같이 만들었다.
>
> ```
> lapply(df, function(x) mean(x, na.rm = TRUE))
> ```
>
> 버전 4.1.0 이후에서는 네이티브로 지원하므로 다음과 같이 만들면 된다.
>
> ```
> lapply(df, \(x) mean(x, na.rm = TRUE))
> ```

11.6.1 across() 함수

- 사용할 데이터
  - palmerpenguins
  - 2025년 4월 새로 업데이트된 R 4.5.0 버전에 palmerpenguins이 base R에 포함되었다.

다음과 같은 코드로 현재 사용하는 R 버전을 확인할 수 있다.

```
> R.version.string
```

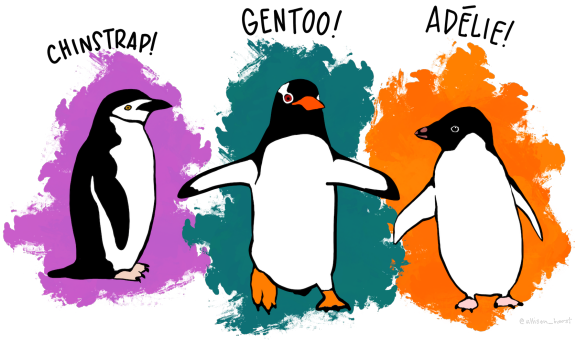


그림 11.3: Palmer Penguins 데이터셋은 기존 iris를 대신하여 교육용으로 많이 사용되는 펭귄 생태학 데이터이다.

```
[1] "R version 4.5.0 (2025-04-11)"
```

`across()` 함수는 주로 `summarise()` 함수와 함께 사용된다. 첫 번째 인자는 `.cols`로 함수를 적용시킬 열을 선택하고, 두 번째 인자는 `.fns`로 적용시킬 함수를 지정한다.

다음은 `where(is.numeric)` 함수를 사용하여 `penguins` 데이터셋에서 모든 숫자형 변수들에 대하여 평균을 계산하는 예이다.

```
penguins |>
    summarise(across(
        where(is.numeric),
        \(x) mean(x, na.rm = TRUE)
    ))
```

```
  bill_len bill_dep flipper_len body_mass     year
1 43.92193 17.15117    200.9152  4201.754 2008.029
```

다음은 모든 행에서 unique한 값의 개수를 계산하는 예이다.

```
penguins |>
    summarise(across(
        everything(),
        n_distinct
    ))
```

```
  species island bill_len bill_dep flipper_len body_mass sex year
1       3      3      165       81          56        95   3    3
```

11.6.2 여러 행에 대한 오퍼레이션: `rowwise()` 함수

`rowwise()` 함수는 각 행 단위로 함수를 적용시킬 때 사용한다.

```
df <- tibble(name = c("Mara", "Hadley"), x = 1:2, y = 3:4, z = 5:6)
df
```

```
# A tibble: 2 x 4
  name       x     y     z
  <chr>  <int> <int> <int>
1 Mara       1     3     5
2 Hadley     2     4     6
```

먼저 `rowwise()` 함수를 사용하지 않을 때를 생각해 보자. 다음 코드는 `df` 데이터프레임을 구성하는 `x`,`y`, `z` 열을 모두 더하는 방식으로 평균을 구한다.

```
df |>
    mutate(m = mean(c(x, y, z), na.rm = TRUE))
```

```
# A tibble: 2 x 5
  name       x     y     z     m
  <chr>  <int> <int> <int> <dbl>
1 Mara       1     3     5   3.5
2 Hadley     2     4     6   3.5
```

만약 각 행에 대한 평균을 구하고자 한다면, `rowwise()` 함수를 사용하여 각 행에 대하여 평균을 계산한다.

```
df |>
    rowwise() |>
    mutate(m = mean(c(x, y, z), na.rm = TRUE))
```

```
# A tibble: 2 x 5
# Rowwise:
  name       x     y     z     m
  <chr>  <int> <int> <int> <dbl>
1 Mara       1     3     5     3
2 Hadley     2     4     6     4
```

이렇게 열이 몇 개 되지 않을 때는 문제가 없지만, 열이 많은 경우에는 `c_across()` 함수를 사용하면 `select()` 함수에서 열을 선택하는 방법을 적용시킬 수 있다.

```
penguins |>
    rowwise() |>
    mutate(m = mean(c_across(bill_len:body_mass), na.rm = TRUE))
```

```
# A tibble: 344 x 9
# Rowwise:
   species island    bill_len bill_dep flipper_len body_mass sex     year     m
   <fct>   <fct>        <dbl>    <dbl>       <int>     <int> <fct>  <int> <dbl>
 1 Adelie  Torgersen     39.1     18.7         181      3750 male    2007  997.
 2 Adelie  Torgersen     39.5     17.4         186      3800 female  2007 1011.
 3 Adelie  Torgersen     40.3     18           195      3250 female  2007  876.
 4 Adelie  Torgersen     NA       NA            NA        NA <NA>    2007  NaN
 5 Adelie  Torgersen     36.7     19.3         193      3450 female  2007  925.
 6 Adelie  Torgersen     39.3     20.6         190      3650 male    2007  975.
 7 Adelie  Torgersen     38.9     17.8         181      3625 female  2007  966.
 8 Adelie  Torgersen     39.2     19.6         195      4675 male    2007 1232.
 9 Adelie  Torgersen     34.1     18.1         193      3475 <NA>    2007  930.
10 Adelie  Torgersen     42       20.2         190      4250 <NA>    2007 1126.
# i 334 more rows
```

11.7 테이블 대상 동사

이 내용을 이해하기 위해서는 **관계형 데이터베이스**에 관한 기본적인 이해가 선행되어야 하기 때문에 14장에서 따로 설명한다.

- 도움이 되는 자료
    - dplyr 비니에트 Two-table verbs
    - R for Data Science (2e) 19장 Joins

12 dplyr 연습

11장 “dplyr로 데이터 가공”에서 설명한 내용을 연습해 보려고 한다.

12.1 사용할 데이터셋

palmerpenguins 패키지에 있는 penguins 데이터프레임을 사용한다.

```
library(dplyr)
library(palmerpenguins)
```

```
penguins
```

데이터의 전체적인 구조와 데이터 타입 확인한다.

```
#| autorun: true
glimpse(penguins)
```

12.2 열 선택(select)

- 열 선택: select() 함수
- 열 선택 방법: 다양한 방식으로 열을 선택할 수 있다. 13장을 참고하여 익힌다.

열 이름을 직접 입력하여 선택할 수 있다. dplyr은 비표준평가를 사용하기 때문에 열 이름을 지정할 때 따옴표를 사용하지 않는다.

```
penguins |>
    select(species, sex, year)
```

변수1:변수3을 사용하여 열의 범위를 사용할 수 있다.

```
penguins |>
    select(bill_length_mm:body_mass_g)
```

!selection을 사용하면 “제외한 열들”을 선택할 수 있다.

```
penguins |>
    select(!contains("mm"))
```

12.3 행 필터(filter)

- 조건에 맞는 행들을 필터링: filter() 함수
- 조건
  - ==, !=, <, <=, >, >= 등의 비교 연산자와 &, | 등의 논리 연산자를 사용한다.
  - is.na(), between(), near() 등의 함수를 사용할 수 있다.

species가 "Adelie"인 행을 필터링한다.

```
penguins |>
    filter(species == "Adelie")
```

"Adelie" 종의 평균 체중은 다음과 같다.

```
#| autorun: true
adelie = penguins |>
    filter(species == "Adelie")
mean(adelie$body_mass_g, na.rm = TRUE)
```

"Adelie" 펭귄에서, 평균 체중 이상의 것들만 추출한다.

```
penguins |>
    select(species, body_mass_g) |>
    filter(species == "Adelie", body_mass_g >= mean(body_mass_g, na.rm = TRUE))
```

species가 "Adelie"이거나 "Chinstrap"인 행을 필터링한다.

```
penguins |>
    filter(species == "Adelie" | species == "Chinstrap")
```

12.4 정렬(arrange)

- 정렬: arrange() 함수
- 정렬 기준: 기준 열을 지정한다. 디폴트는 오름차순(ascending)인데, 내림차순으로 정렬하려면 desc(열이름) 함수를 사용한다.

bill_length_mm를 기준으로 내림차순으로 정렬한다.

```
penguins |>
    arrange(desc(bill_length_mm))
```

12.5 열 이름 변경(rename)

- 열 이름 변경: rename() 함수
- 열 이름 변경 방법: new_name = old_name 형식으로 지정한다.

펭귄 데이터프레임에서 bill_length_mm을 bill_length로, bill_depth_mm을 bill_depth로, flipper_length_mm을 flipper_length로, body_mass_g를 body_mass로 변경한다.

```
penguins |>
    rename(
        bill_length = bill_length_mm,
        bill_depth = bill_depth_mm,
        flipper_length = flipper_length_mm,
        body_mass = body_mass_g
    )
```

- 열 이름을 문자열로 가지고 와서, 이 문자열에 대하여 어떤 함수를 적용한 후의 값을 열 이름으로 사용하고 싶을 때는 rename_with() 함수를 사용한다.

펭귄 데이터프레임의 모든 열 이름을 대문자로 바꾼다.

```
penguins |>
    rename_with(toupper)
```

12.6 열 추가(mutate)

- 열 추가: `mutate()` 함수
    - 기존 데이터프레임에 새로운 열을 추가한다.
- 열 추가 방법: `new_name = expression` 형식으로 지정한다.

펭귄 데이터셋에 `body_mass_kg`라는 열을 추가한다. 이 열은 `body_mass_g`를 1000으로 나눈 값이다.

```
penguins |>
    select(body_mass_g) |>
    mutate(body_mass_kg = body_mass_g / 1000)
```

12.7 그룹화(group_by)와 요약(summarize)

- 그룹화: `group_by()` 함수
- 요약: `summarize()` 또는 `summarise()` 함수

전체 그룹을 소그룹으로 나누고, 각 소그룹에 대해 함수를 적용하여 요약한 다음, 그것을 하나의 데이터프레임으로 반환한다.

펭귄의 `species`별로 `body_mass_g`의 평균을 구한다.

```
penguins |>
    group_by(species) |>
    summarize(mean_body_mass = mean(body_mass_g, na.rm = TRUE))
```

- `summarise()` 함수 안에서 요약에 사용할 수 있는 함수들은 다음과 같다.
    - `n()`: 개수

- mean(): 평균
- median(): 중앙값
- sd(): 표준편차
- var(): 분산
- min(): 최솟값
- max(): 최댓값
- sum(): 합계
- first(): 첫 번째 값
- last(): 마지막 값
- nth(): n번째 값
- n_distinct(): 고유한 값의 개수
- quantile(): 분위수
- IQR(): 사분위수 범위

펭귄의 species별로 카운트와 body_mass_g의 평균을 구한다.

```
penguins |>
    group_by(species) |>
    summarize(count = n(), mean_body_mass = mean(body_mass_g, na.rm = TRUE))
```

펭귄의 species로 나눠서 볼 때 body_mass_g가 가장 큰 값과 작은 값을 구한다.

```
penguins |>
    group_by(species) |>
    summarize(max_body_mass = max(body_mass_g, na.rm = TRUE), min_body_mass = min(body_mass_g,
```

정렬을 사용하여 body_mass_g가 가장 큰 값과 작은 값을 구한다.

```
penguins |>
    filter(!is.na(body_mass_g)) |>
    group_by(species) |>
    arrange(desc(body_mass_g)) |>
    summarise(
        max_body_mass = first(body_mass_g),
        min_body_mass = last(body_mass_g))
```

위 함수들을 모두 n개의 값을 가진 벡터에 적용되어 1개의 값을 반환하는 함수들이다(reducing function).

12.8 그룹화(group_by)와 재구성(reframe)

- 그룹화: `group_by()` 함수
- 재구성: `reframe()` 함수는 `summarise()`와 비슷하지만, 여러 개의 값을 반환하는 요약 함수를 사용할 때 편리하다.

`species`별로 그을 나누고, `range` 함수르로 가지고 `body_mass_g`의 최솟값과 최댓값을 구한다.

```
penguins |>
    group_by(species) |>
    reframe(
        body_mass_range = range(body_mass_g, na.rm = TRUE),
        bill_length_range = range(bill_length_mm, na.rm = TRUE)
    )
```

숫자형 벡터의 개수, 평균, 표준편차를 구하는 사용자 정의함수를 만들어 보자(그다지 훌륭한 예는 아니다).

```
#| autorun: true
summary_fn <- function(x) {
    c(
        n = length(x),
        mean = mean(x, na.rm = TRUE),
        sd = sd(x, na.rm = TRUE)
    )
}
```

이 함수는 `x`라는 숫자형 벡터를 인자로 받아서, 그 벡터의 개수, 평균, 표준편차를 구하여 반환한다. 이 함수는 `summarise()` 안에서 사용할 수 없다. 이런 경우에는 `reframe()` 함수를 사용한다.

```
penguins |>
    group_by(species) |>
    reframe(
        body_mass_summary = summary_fn(body_mass_g),
        bill_length_summary = summary_fn(bill_length_mm)
    )
```

12.9 summarize()와 mutate() 안에서 여러 열에 함수들을 적용: across()

앞에서 본 summarise() 함수를 across() 함수 없이 단독으로 사용할 때는 다음과 같은 문법에 따라 사용한다.

```
df |> summarise(
    new_col1 = functionA(old_col1),
    new_col2 = functionA(old_col2),
    new_col3 = functionA(old_col3),
    new_col4 = functionB(old_col4),
    new_col5 = functionB(old_col5),
    new_col6 = functionB(old_col6),
    ...
))
```

만약 old_col1, old_col2, old_col3이 모두 숫자형 벡터라든지 뭔가 묶을 수 있는 형태라면, summarise() 함수 안에서 across() 함수를 사용하여 한꺼번에 요약할 수 있다. 즉, across() 함수는 summarise() 함수 안에서 tidyselect 문법을 사용하여 **여러 개의 열을 한꺼번에 묶어서** 요약할 수 있도록 해준다.

뿐만 아니라 across() 함수의 두번째 인자 .fns에 함수는 1개 이상 지정할 수 있으며, 각 열에 이들 함수가 모두 적용된다. 이 인자를 적용하는 방법은 다음과 같다.

- 함수 이름: mean
- purrr 스타일의 람다 함수
- \(x) mean(x, na.rm = TRUE)처럼 베이스 R 스타일의 무명 함수(anonymous function)

- `list(mean = mean, median = median)`처럼 여러 개의 함수를 지정할 수 있다. 이 경우에는 각 열에 대해 지정한 함수가 모두 적용된다.

펭귄 데이터셋에서 숫자형 벡터들을 한꺼번에 요약하는 예를 보자.

```
penguins |>
    group_by(species) |>
    summarise(
        across(
            bill_length_mm:body_mass_g,
            \(x) mean(x, na.rm = TRUE)
        )
    )
```

평균과 표준 편차를 한꺼번에 구하는 예를 보자.

```
penguins |>
    group_by(species) |>
    summarise(
        across(
            bill_length_mm:body_mass_g,
            list(mean = \(x) mean(x, na.rm = TRUE),
                 sd = \(x) sd(x, na.rm = TRUE)
            )
        )
    )
```

12.10 누적 계산(cummulative calculation)과 순위(rank):mutate() + 벡터화 함수

누적된 계산을 하려면 `mutate()` 함수와 벡터화 함수를 사용한다. 예를 들어 `cumsum()`은 누적합을 계산하는 벡터화 함수이다. 다음과 같은 함수들을 `mutate()` 함수 안에서 사용하여 누적된 값을 가지는 새로운 열을 추가한다.

- 누적합: `cumsum()`

- 누적곱: `cumprod()`
- 누적최솟값: `cummin()`
- 누적최댓값: `cummax()`

다음과 같은 시계열 데이터에서 이런 누적 계산이 많이 사용된다.

```
#| autorun: true
revenue_data <- tibble(
  Date = seq(as.Date("2022-01-01"), by = "month", length.out = 12),
  Revenue = c(11400, 11000, 12500, 13500, 14000, 14500,
              10000, 10000, 13000, 15000, 14000, 16000)
)
revenue_data
```

누적 매출액을 계산하여 `Cumulative_Revenue`라는 열을 추가한다.

```
revenue_data |>
    mutate(Cumulative_Revenue = cumsum(Revenue))
```

12.11 기준에 맞게 그룹화(팩터) 변수 만들기: `if_else()`와 `case_when()`

다음과 같은 데이터프레임을 가지고 시작해 보자.

```
#| autorun: true
df <- tibble(
  name = c("Alice", "Bob", "Carol", "David", "Eva"),
  score = c(95, 82, 73, 60, 45)
)
df
```

이 데이터프레임에서 `score`에 따라 `grade`라는 새로운 열을 추가할 것인데, `score`가 60점 보다 크면 `pass`, 60점 이하이면 `fail`로 설정하려고 한다. 이 경우 조건이 하나이기 때문에 `if_else()` 함수를 사용한다.

- `if_else()` 함수

- if_else(조건, 참일 때의 값, 거짓일 때의 값)

```
# if_else를 사용한 등급 부여
df <- df %>%
  mutate(
    grade = if_else(score >= 60, "pass", "fail"),
    grade = factor(grade, levels = c("pass", "fail"))
  )
df
```

만약 score에 따라 grade를 A, B, C, D, F로 나누고 싶다면, 이 경우에는 case_when() 함수를 사용한다.

- case_when() 함수
  - case_when(조건1 ~ 값1, 조건2 ~ 값2, ...)
  - 조건이 여러 개일 때는 &와 |를 사용하여 조합할 수 있다. 조건은 specific 한 것에서 general 순으로 나열한다.
  - 마지막에 .default = 값을 사용하면 남은 모든 경우를 처리한다.

```
# case_when을 사용한 등급 부여
df <- df %>%
  mutate(
    grade = case_when(
      score >= 90           ~ "A",
      score >= 80 & score < 90  ~ "B",
      score >= 70 & score < 80  ~ "C",
      score >= 60 & score < 70  ~ "D",
      .default                  = "F"  # 나머지
    ),
    grade = factor(grade, levels = c("A", "B", "C", "D", "F"))
    # 팩터로 변환
  )

df
```

12.12 결측값 다루기: na_if(), coalesce()

실제 데이터에는 결측값이 없는 경우가 드물다. R에서는 결측값을 NA로 표시한다. 이런 결측값을 다루는 데 유용한 함수들을 소개한다. dplyr 패키지의 coalesce()와 na_if() 함수와 tidyr 패키지의 replace_na() 함수가 있다.

- na_if(x, v): 벡터 x 요소의 값과 벡터 v 요소 값이 같으면 NA로 변환한다.
- coalesce(): 여러 개의 벡터를 가지고, 각 위치에서 첫 번째로 NA가 아닌 값을 반환한다.

이 함수들은 벡터화된 함수여서(vectorized functions), 처음에는 함수 도움말을 읽어도 이해가 쉽지 않을 수 있다.

12.12.1 na_if()

먼저 na_if()를 사용법이다.

```
#| autorun: true
x <- c(1, 2, 3, 4, 5)
y <- c(5, 4, 3, 2, 1)
na_if(x, y)
```

3번째 값이 NA로 바뀌는 것은 x의 3번째 값과 y의 3번째 값이 같기 때문이다. 다음을 보자.

```
#| autorun: true
na_if(x, 3)
```

이 경우에는 두 번째 3이 스칼라 값이기 때문에서, 앞의 x와 맞추려고 recyling되어서 암묵적으로 c(3, 3, 3, 3, 3)으로 바뀌어서 x의 세 번째 값이 NA로 바뀌게 된다.

그래서 na_if(x, 3)을 읽으면, x에서 값이 3인 경우라면 NA로 바꿔라라는 의미가 된다.

따라서 이 함수는 다음과 같이 결측값을 999 등으로 코딩한 것을 NA로 바꿀 때 유용하게 사용할 수 있다.

```
# 예시 데이터프레임
df <- tibble(
  name = c("Alice", "Bob", "Carol", "David"),
```

```
  score = c(85, 999, 70, 999)
)

# 999를 NA로 바꾸기
df <- df %>%
  mutate(
    score_clean = na_if(score, 999)
  )
df
```

12.12.2 coalesce()

coalesce()는 여러 개의 벡터를 가지고, 각 위치에서 첫 번째로 NA가 아닌 값을 반환한다.

이 함수의 사용법을 보자.

```
#| autorun: true
x <- c(1, 3, 5, NA, 7, NA)
x
```

```
#| autorun: true
coalesce(x, 0L)
```

결국 coalesce(x, 0L)은 x에서 NA의 값을 0으로 대체한다. 이 의미가 "coalesce()는 여러 개의 벡터를 가지고, 각 위치에서 첫 번째로 NA가 아닌 값을 반환한다."라는 의미와 어떻게 연결되지는 보자.

colacese(x, 0L)은 다음 두 벡터를 가지고 비교한다.

```
c(1, 3, 5, NA, 7, NA)
c(0, 0, 0, 0, 0, 0)
```

1, 3, 5는 NA가 아니기 때문에 x의 값을 그대로 가져온다. 그 다음은 NA이기 때문에 그 다음 벡터에서 NA가 아닌 0을 가지고 온다. 7은 그대로 가지고 오고, 마지막 NA는 0을 가지고 온다.

```
a <- c(1, 3, 5, NA, 7, NA)
b <- c(0, 0, NA, NA, 9, NA)
c <- c(10, 20, 30, 40, 50, 60)
coalesce(a, b, c)
```

이것은 다음과 같은 3개의 벡터의 요소값을 서로 비교한다.

```
c(1, 3, 5, NA, 7, NA)
c(0, 0, NA, NA, 9, NA)
c(10, 20, 30, 40, 50, 60)
```

1, 3, 5는 NA가 아니기 때문에 a 값을 그대로 가져온다. 그 다음은 NA이기 때문에 그 다음 벡터 b에서 값을 가지고 오려 했으나 역시 NA여서 그 다음 벡터 c에서 40을 가지고 온다. 7은 그대로 가지고 온다. 마지막도 벡터 a, b의 값은 모두 NA이기 때문에 c에서 60을 가지고 온다.

12.13 정리

dplyr 패키지의 주요 함수들을 정리하였다. 이 외에도 다양한 함수들이 있으니, dplyr 패키지 문서를 참고하기 바란다.

13 tidyselect 열 선택

Tidyverse 메타패키지에 속하는 `tidyr`, `dplyr`, `purrr`과 같은 패키지의 함수들을 모두 `tidyselect`라는 열 선택 방법을 사용한다. 이것은 그 자체가 아주 작은 언어이기도 한데, 잘 사용하면 작업을 쉽게 할 수 있다. 특히 수십, 수백개의 열 이름을 가진 데이터를 처리할 때 이 방법이 아주 효과적이다.

- Selection language에 있는 내용을 바탕으로 정리했다.

```
library(tidyverse)
```

`penguins` 데이터셋을 사용해서 열 선택 방법을 살펴보자.

```
head(penguins)
```

```
  species    island bill_len bill_dep flipper_len body_mass    sex year
1  Adelie Torgersen     39.1     18.7         181      3750   male 2007
2  Adelie Torgersen     39.5     17.4         186      3800 female 2007
3  Adelie Torgersen     40.3     18.0         195      3250 female 2007
4  Adelie Torgersen       NA       NA          NA        NA   <NA> 2007
5  Adelie Torgersen     36.7     19.3         193      3450 female 2007
6  Adelie Torgersen     39.3     20.6         190      3650   male 2007
```

- 변수1:변수10 형식으로 열 이름을 지정하면 데이터프레임에서 변수1 열부터 변수10 열까지의 열을 선택한다.

```
penguins |>
    select(bill_len:body_mass) |>
    slice(1:5)
```

```
  bill_len bill_dep flipper_len body_mass
```

```
1     39.1     18.7         181       3750
2     39.5     17.4         186       3800
3     40.3     18.0         195       3250
4       NA       NA          NA         NA
5     36.7     19.3         193       3450
```

- `starts_with("변수")` 형식으로 열 이름을 지정하면 데이터프레임에서 변수 열이름으로 시작하는 열을 선택하고 `ends_with("변수")` 형식으로 열 이름을 지정하면 데이터프레임에서 변수 열이름으로 끝나는 열을 선택한다.

```
penguins |>
    select(starts_with("bill")) |>
    slice(1:5)
```

```
  bill_len bill_dep
1     39.1     18.7
2     39.5     17.4
3     40.3     18.0
4       NA       NA
5     36.7     19.3
```

```
penguins |>
    select(ends_with("len")) |>
    slice(1:5)
```

```
  bill_len flipper_len
1     39.1         181
2     39.5         186
3     40.3         195
4       NA          NA
5     36.7         193
```

- `contains("문자열")` 형식으로 열 이름을 지정하면 데이터프레임에서 이 문자열을 포함하는 열을 선택한다.

```
penguins |>
    select(contains("pper")) |>
    slice(1:5)
```

```
  flipper_len
1         181
2         186
3         195
4          NA
5         193
```

- `matches("정규표현식")` 형식으로 열 이름을 지정하면 데이터프레임에서 이 정규표현식에 일치하는 열을 선택한다. 정규 표현식(regular expression)은 문자열을 표현하는 형식으로 문자열 패턴을 지정하는 일반적인 방법이다. 강력한데, 처음 배우는 사람들은 어려워하는 편이다. 예를 들어, `a[a-z]{2}`는 `a` 문자 뒤에 오는 두 개의 영문자가 따라 오는 경우를 말한다.

```
penguins |>
    select(matches("a[a-z]{2}")) |>
    slice(1:5)
```

```
    island body_mass
1 Torgersen      3750
2 Torgersen      3800
3 Torgersen      3250
4 Torgersen        NA
5 Torgersen      3450
```

- `num_range("변수", 1:10)` 형식으로 열 이름을 지정하면 데이터프레임에서 변수1, 변수2, …, 변수10 열을 선택한다. `tidyr` 패키지에 들어 있는 `billoard` 데이터셋을 예로 보자. 여기에서 `wk1`에서 `wk8` 열을 선택해 보자.

```
head(billboard)
```

```
# A tibble: 6 x 79
  artist      track date.entered   wk1   wk2   wk3   wk4   wk5   wk6   wk7   wk8
```

```
  <chr>       <chr> <date>       <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 2 Pac       Baby~ 2000-02-26      87    82    72    77    87    94    99    NA
2 2Ge+her     The ~ 2000-09-02      91    87    92    NA    NA    NA    NA    NA
3 3 Doors Do~ Kryp~ 2000-04-08      81    70    68    67    66    57    54    53
4 3 Doors Do~ Loser 2000-10-21      76    76    72    69    67    65    55    59
5 504 Boyz    Wobb~ 2000-04-15      57    34    25    17    17    31    36    49
6 98^0        Give~ 2000-08-19      51    39    34    26    26    19     2     2
# i 68 more variables: wk9 <dbl>, wk10 <dbl>, wk11 <dbl>, wk12 <dbl>,
#   wk13 <dbl>, wk14 <dbl>, wk15 <dbl>, wk16 <dbl>, wk17 <dbl>, wk18 <dbl>,
#   wk19 <dbl>, wk20 <dbl>, wk21 <dbl>, wk22 <dbl>, wk23 <dbl>, wk24 <dbl>,
#   wk25 <dbl>, wk26 <dbl>, wk27 <dbl>, wk28 <dbl>, wk29 <dbl>, wk30 <dbl>,
#   wk31 <dbl>, wk32 <dbl>, wk33 <dbl>, wk34 <dbl>, wk35 <dbl>, wk36 <dbl>,
#   wk37 <dbl>, wk38 <dbl>, wk39 <dbl>, wk40 <dbl>, wk41 <dbl>, wk42 <dbl>,
#   wk43 <dbl>, wk44 <dbl>, wk45 <dbl>, wk46 <dbl>, wk47 <dbl>, wk48 <dbl>, ...
```

```
billboard |>
    select(num_range("wk", 1:8)) |>
    slice(1:5)
```

```
# A tibble: 5 x 8
    wk1   wk2   wk3   wk4   wk5   wk6   wk7   wk8
  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1    87    82    72    77    87    94    99    NA
2    91    87    92    NA    NA    NA    NA    NA
3    81    70    68    67    66    57    54    53
4    76    76    72    69    67    65    55    59
5    57    34    25    17    17    31    36    49
```

- `all_of(c("변수1", "변수2", "변수3"))` 형식으로 열 이름을 지정하면 데이터프레임에서 이 열들을 선택한다. 지정한 열 이름이 모두 있어야 실행되고, 없으면 에러가 발생한다. 반면 `any_of(c("변수1", "변수2", "변수3"))` 형식으로 열 이름을 지정하면 데이터프레임에서 이 열들을 선택한다. 있는 열들만 선택한다.

```
sel_vars <- c("bill_len", "bill_dep", "flipper_len")
penguins |>
```

```
    select(all_of(sel_vars)) |>
    slice(1:5)
```

```
  bill_len bill_dep flipper_len
1     39.1     18.7         181
2     39.5     17.4         186
3     40.3     18.0         195
4       NA       NA          NA
5     36.7     19.3         193
```

```
sel_vars <- c("bill_len", "bill_depth", "flipper_len", "body_mass")
penguins |>
    select(any_of(sel_vars)) |>
    slice(1:5)
```

```
  bill_len flipper_len body_mass
1     39.1         181      3750
2     39.5         186      3800
3     40.3         195      3250
4       NA          NA        NA
5     36.7         193      3450
```

- `everything()` 형식으로 열 이름을 지정하면 데이터프레임에서 모든 열을 선택한다.

```
penguins |>
    select(everything()) |>
    slice(1:5)
```

```
  species    island bill_len bill_dep flipper_len body_mass    sex year
1  Adelie Torgersen     39.1     18.7         181      3750   male 2007
2  Adelie Torgersen     39.5     17.4         186      3800 female 2007
3  Adelie Torgersen     40.3     18.0         195      3250 female 2007
4  Adelie Torgersen       NA       NA          NA        NA   <NA> 2007
5  Adelie Torgersen     36.7     19.3         193      3450 female 2007
```

- `last_col()` 형식으로 열 이름을 지정하면 데이터프레임에서 마지막 열을 선택한다.

```
penguins |>
    select(last_col()) |>
    slice(1:5)
```

```
  year
1 2007
2 2007
3 2007
4 2007
5 2007
```

- `where(fn)` 형식으로 열 이름을 지정하면 데이터프레임의 열에 함수 `fn`을 적용한 결과가 `TRUE`인 열을 선택한다. 팩퍼(factor) 열인지 확인하는 `is.factor`, 숫자 열인지 확인하는 `is.numeric` 등의 함수를 사용할 수 있다.

```
penguins |>
    select(where(is.factor)) |>
    slice(1:5)
```

```
  species    island    sex
1  Adelie Torgersen   male
2  Adelie Torgersen female
3  Adelie Torgersen female
4  Adelie Torgersen   <NA>
5  Adelie Torgersen female
```

```
penguins |>
    select(where(is.numeric)) |>
    slice(1:5)
```

```
  bill_len bill_dep flipper_len body_mass year
1     39.1     18.7         181      3750 2007
2     39.5     17.4         186      3800 2007
```

```
3     40.3      18.0         195       3250 2007
4       NA        NA          NA         NA 2007
5     36.7      19.3         193       3450 2007
```

이런 선택에서 연산자를 추가하여 사용할 수 있다.

- `!selection`은 `selection`에 해당하는 열을 제외한 나머지 열을 선택한다. `!`에 마이너스 - 기호를 사용할 수도 있다.

```
penguins |>
    select(!starts_with("bill")) |>
    slice(1:5)
```

```
  species    island flipper_len body_mass    sex year
1  Adelie Torgersen         181      3750   male 2007
2  Adelie Torgersen         186      3800 female 2007
3  Adelie Torgersen         195      3250 female 2007
4  Adelie Torgersen          NA        NA   <NA> 2007
5  Adelie Torgersen         193      3450 female 2007
```

```
penguins |>
    select(-starts_with("bill")) |>
    slice(1:5)
```

```
  species    island flipper_len body_mass    sex year
1  Adelie Torgersen         181      3750   male 2007
2  Adelie Torgersen         186      3800 female 2007
3  Adelie Torgersen         195      3250 female 2007
4  Adelie Torgersen          NA        NA   <NA> 2007
5  Adelie Torgersen         193      3450 female 2007
```

- `selection1 & selection2`는 AND 로직을 `selection | selection2`‘ OR 로직을 사용한다.

```
penguins |>
    select(starts_with("bill") | ends_with("len")) |>
```

```
    slice(1:5)
```

```
  bill_len bill_dep flipper_len
1     39.1     18.7         181
2     39.5     17.4         186
3     40.3     18.0         195
4       NA       NA          NA
5     36.7     19.3         193
```

```
penguins |>
    select(starts_with("bill") & ends_with("len")) |>
    slice(1:5)
```

```
  bill_len
1     39.1
2     39.5
3     40.3
4       NA
5     36.7
```

14 관계형 데이터베이스 기초 개념, Joins와 two-table verbs

관계형 데이터베이스(Relational Database)는 구조화된 데이터를 저장하는 데이터베이스로 보이지는 않지만 우리는 늘 이것을 사용한다. 데이터베이스 전문가가 될 것은 아니라 할지라도 그 개념을 이해하고, 좀 더 나아가 활용할 수 있다면 좋을 것이다. 또한 이 개념을 이해하면 **Tidy data** 개념을 이해하는데 큰 도움이 된다(실제로 tidy data의 개념은 관계형 데이터베이스의 개념에서 유래되었다).

관계형 데이터베이스는 테이블(table)의 집합으로 이루어진 데이터베이스이다. 각 테이블은 열(column)과 행(row)으로 이루어져 있으며, 각 열은 특정한 속성을 나타내고, 각 행은 개별 데이터 항목을 나타낸다. 즉, 관계형 데이터베이스의 테이블은 다음과 같은 특징이 있다.

- 테이블은 열(column)과 행(row)으로 이루어져 있다.
- 각 열은 특정한 속성을 나타내고, 각 행은 개별 데이터 항목을 나타낸다.
- 테이블은 키(key)를 통해 연결될 수 있다.

키(key)를 통해 테이블들이 서로 어떤 **관계**를 가지면서 온전한 데이터베이스를 이루기 때문에, 관계형 데이터베이스라고 하는 것이다.

14.1 관계형 데이터베이스 예제: 눈으로 보는 관계형 데이터베이스

관계형 데이터베이스는 **Data Normalization**이라는 개념을 통해 데이터의 중복을 최소화하고, 데이터의 일관성을 유지한다.

간단한 도서관 도서 대출 시스템을 관리하는 데이터베이스를 가지고 설명한다(이 자료는 DuckDB: Up and Running (Wei-Meng Lee 저) 책에서 발췌했다).

이 데이터베이스는 4개의 테이블로 구성되어 있다.

- `authors`: 저자 정보

- books: 도서 정보
- borrowers: 대출자 정보
- borrowings: 대출 정보

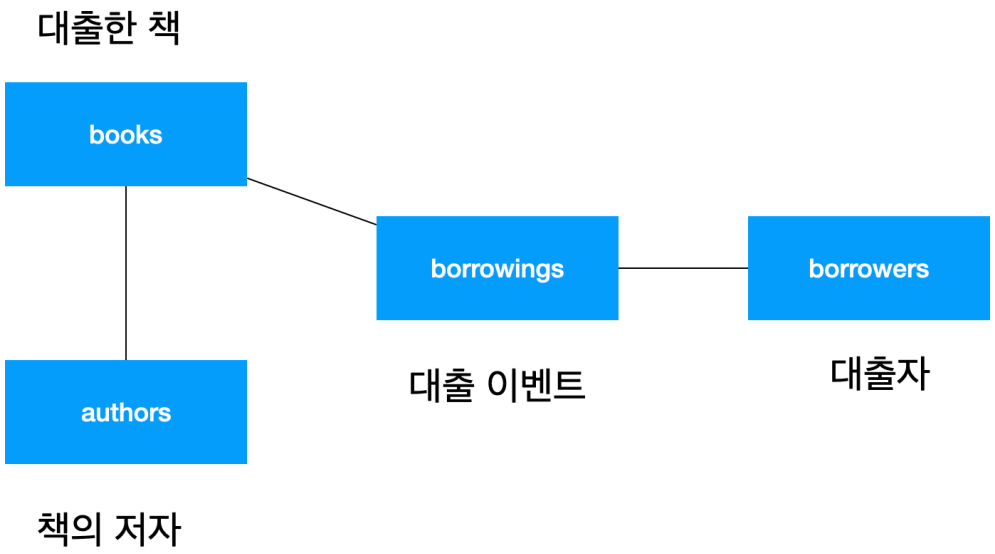


그림 14.1: 데이터베이스의 테이블 사이의 관계

다음 코드는 지금은 이해할 필요가 없는데, 설명을 위해서 가상의 데이터베이스를 만들었고, 이것을 R로 불러오는 코드라고만 이해하면 된다.

```
library(DBI)
library(duckdb)
con <- dbConnect(duckdb::duckdb(), "data/books.duckdb", read_only = TRUE)

authors <- dbReadTable(con, "Authors")
books <- dbReadTable(con, "Books")
borrowers <- dbReadTable(con, "Borrowers")
borrowings <- dbReadTable(con, "Borrowings")
```

먼저 저자 정보이다.

표 14.1: 저자 정보(authors)

author_id	name	nationality	birth_year
1	Jane Austen	British	1775
2	Charles Dickens	British	1812
3	Agatha Christie	British	1890
4	J.K. Rowling	British	1965
5	Tolkien	British	1892
6	Mark Twain	American	1835

도서 정보이다.

표 14.2: 도서 정보(books)

book_id	title	author_id	genre	publication_year
1	Pride and Prejudice	1	Classic	1813
2	Oliver Twist	2	Novel	1837
3	Murder on the Orient Express	3	Mystery	1934
4	Harry Potter and the Philosopher's Stone	4	Fantasy	1997
5	The Hobbit	5	Fantasy	1937

대출자 정보이다.

표 14.3: 대출자 정보(borrowers)

borrower_id	name	email	member_since
1	John Smith	john.smith@example.com	2022-01-01
2	Emma Johnson	emma.johnson@example.com	2021-12-15
3	Michael Brown	michael.brown@example.com	2022-02-20
4	Sophia Wilson	sophia.wilson@example.com	2022-03-10
5	William Taylor	william.taylor@example.com	2022-04-05
6	Jane Doe	jane.doe@example.com	2022-03-05

대출 정보이다.

표 14.4: 대출 정보(borrowings)

borrowing_id	book_id	borrower_id	borrow_date	return_date	status
1	1	1	2022-04-10	2022-04-25	Returned
2	3	2	2022-03-20	NA	On Loan
3	4	3	2022-04-05	NA	On Loan
4	2	4	2022-04-15	NA	On Loan
5	5	5	2022-03-30	2022-04-20	Returned
6	1	3	2022-04-26	NA	On Loan

이 데이터베이스는 "대출"이라는 사건을 중심으로 보는 것이 좋다. 어떻게 데이터를 찾아가는지 살펴보자. `borrowings` 테이블에서 5번 대출 이벤트를 읽어 보자.

표 14.5

	borrowing_id	book_id	borrower_id	borrow_date	return_date	status
5	5	5	5	2022-03-30	2022-04-20	Returned

1. 이 이벤트는 `book_id`가 5인 도서를 대출한 이벤트이다.
2. 이 도서를 대출한 사람은 `borrower_id`가 5인 사람이다.
3. 대출 날짜는 2022-03-30이고 반납 날짜는 2022-04-01으로 대출 이벤트가 종결되었다.

이 데이터를 찾아가는 방법은 다음과 같다. `books` 테이블과 `authors` 테이블에서 책과 그 저자를 확인할 수 있다. 이런 과정은 모두 look-up 프로세스이다.

표 14.6: 도서 정보(books)

	book_id	title	author_id	genre	publication_year
5	5	The Hobbit	5	Fantasy	1937

표 14.7: 저자 정보(authors)

	author_id	name	nationality	birth_year
5	5	Tolkien	British	1892

대출자 정보를 확인해 보자. `borrowers` 테이블에서 대출자 정보를 확인할 수 있다.

표 14.8: 대출자 정보(borrowers)

	borrower_id	name	email	member_since
5	5	William Taylor	william.taylor@example.com	2022-04-05

이번에는 `borrowings` 테이블에서 1번 대출 이벤트를 읽어 보자.

표 14.9: 대출 정보(borrowings)

borrowing_id	book_id	borrower_id	borrow_date	return_date	status
1	1	1	2022-04-10	2022-04-25	Returned

1. 이 이벤트는 `book_id`가 `1`인 도서를 대출한 이벤트이다.
2. 이 도서를 대출한 사람은 `borrower_id`가 `1`인 사람이다.
3. 대출 날짜는 `2022-01-01`이고 반납 날짜는 `2022-01-02`으로 대출 이벤트가 종결되었다.

이 데이터를 찾아가는 방법은 다음과 같다. `books` 테이블과 `authors` 테이블에서 책과 그 저자를 확인할 수 있다.

표 14.10: 도서 정보(books)

book_id	title	author_id	genre	publication_year
1	Pride and Prejudice	1	Classic	1813

표 14.11: 저자 정보(authors)

author_id	name	nationality	birth_year
1	Jane Austen	British	1775

대출자 정보를 확인해 보자. `borrowers` 테이블에서 대출자 정보를 확인할 수 있다.

표 14.12: 대출자 정보(borrowers)

borrower_id	name	email	member_since
1	John Smith	john.smith@example.com	2022-01-01

14.2 테이블 사이의 관계를 규정하는 키(Key)

앞서 `authors`, `books`, `borrowers`, `borrowings` 테이블의 첫 열은 `author_id`, `book_id`, `borrower_id`, `borrowing_id`이다. 이 열은 각 테이블의 행을 고유하게 식별하는 열이다. 이와 같은 열을 기본 키(primary key)라고 한다. 즉, 기본 키는 하나의 테이블에서 고유한 값을 가지고, 다른 행과 구분된다.

기본 키는 어떤 경우에는 고유한 값을 가지는 하나의 열로 구섣되기도 하고, 또 어떤 경우는 여러 개의 열 값들의 조합으로 구성되기도 한다. 이렇게 여러 개의 값으로 구성된 기본 키를 복합 키(composite key)라고 한다.

외래 키(foreign key)는 다른 테이블의 기본 키를 참조하는 열이다. 예를 들어 `books` 테이블에서 `author_id` 열은 `authors` 테이블의 `author_id` 열을 참조한다. 이렇게 참조 관계를 가지는 열을 외래 키(foreign key)라고 하고, 우리는 그 값을 따라가서 정보를 확인할 수 있다.

표 14.13: 도서 정보(books)

book_id	title	author_id	genre	publication_year
1	Pride and Prejudice	1	Classic	1813
2	Oliver Twist	2	Novel	1837
3	Murder on the Orient Express	3	Mystery	1934
4	Harry Potter and the Philosopher's Stone	4	Fantasy	1997
5	The Hobbit	5	Fantasy	1937

`borrowings` 테이블은 `book_id` 열은 `books` 테이블의 `book_id` 열을 참조하고, `borrower_id` 열은 `borrowers` 테이블의 `borrower_id` 열을 참조한다. 이 둘 모두 외래 키이다.

표 14.14: 대출 정보(borrowings)

borrowing_id	book_id	borrower_id	borrow_date	return_date	status
1	1	1	2022-04-10	2022-04-25	Returned
2	3	2	2022-03-20	NA	On Loan
3	4	3	2022-04-05	NA	On Loan
4	2	4	2022-04-15	NA	On Loan
5	5	5	2022-03-30	2022-04-20	Returned
6	1	3	2022-04-26	NA	On Loan

14.3 JOINS과 dplyr two-table verbs

관계형 데이터베이스는 데이터들이 열결된 테이블들에 흩어져 있게 된다. 따라서 필요한 경우 테이블들을 연결하여 원하는 정보를 조회할 수 있는 기능이 필요하다. 이런 기능을 조인(join)이라고 한다. 테이블을 결합하는 논리에 따라 여러 종류의 조인이 있는데, 가장 많이 사용하는 조인만 정리해 보면 다음과 같다.

- inner join: 두 테이블에서 일치하는 값을 기준으로 결합
- 외부 조인(outer join)
    - left (outer) join: 왼쪽 테이블의 모든 데이터를 포함하고, 오른쪽 테이블에서 일치하는 값이 있는 경우 결합
    - right (outer) join: 오른쪽 테이블의 모든 데이터를 포함하고, 왼쪽 테이블에서 일치하는 값이 있는 경우 결합
        * 이것은 왼쪽 조인의 반대이다.
    - full (outer) join: 두 테이블의 데이터를 합치는 것

이제 `dplyr` 패키지에서 제공하는 two-table verbs(joins) 함수를 살펴보자.

```
library(dplyr)
```

먼저 가장 흔히 사용되는 left (outer) join을 살펴보자. 이 함수는 왼쪽 테이블의 모든 데이터를 포함하고, 오른쪽 테이블에서 일치하는 값이 있는 경우 결합한다. 여기서 왼쪽, 오른쪽이라고 하는 말은 조인할 테이블을 좌, 우로 두는 것을 표현한 것이다. 이 left join은 왼쪽 테이블은 그대로 유지하면서, 오른쪽에 있는 테이블의 데이터를 추가하는 것이다.

조인을 이해할 때는 사용자의 의도를 명확하게 하는 것이 중요하다. 예를 들어 다음은 모든 대출 정보를 조회하는 데, 도서 정보를 추가하고 싶다고 하자. 다음은 모든 대출 정보를 가지고 오고, 거기에 매칭되는 도서 정보를 추가한다.

borrowing_id	book_id	borrower_id	borrow_date	return_date	status
1	1	1	2022-04-10	2022-04-25	Returned
2	3	2	2022-03-20	NA	On Loan
3	4	3	2022-04-05	NA	On Loan
4	2	4	2022-04-15	NA	On Loan
5	5	5	2022-03-30	2022-04-20	Returned
6	1	3	2022-04-26	NA	On Loan

book_id	title	author_id	genre	publication_year
1	Pride and Prejudice	1	Classic	1813
2	Oliver Twist	2	Novel	1837
3	Murder on the Orient Express	3	Mystery	1934
4	Harry Potter and the Philosopher's Stone	4	Fantasy	1997
5	The Hobbit	5	Fantasy	1937

```
borrowings |>
    left_join(books, by = c("book_id" = "book_id"))
```

```
  borrowing_id book_id borrower_id borrow_date return_date   status
1            1       1           1  2022-04-10  2022-04-25 Returned
2            2       3           2  2022-03-20        <NA>  On Loan
3            3       4           3  2022-04-05        <NA>  On Loan
4            4       2           4  2022-04-15        <NA>  On Loan
5            5       5           5  2022-03-30  2022-04-20 Returned
6            6       1           3  2022-04-26        <NA>  On Loan
                                  title author_id   genre publication_year
1                     Pride and Prejudice         1 Classic             1813
2             Murder on the Orient Express         3 Mystery             1934
3 Harry Potter and the Philosopher's Stone         4 Fantasy             1997
```

```
4                          Oliver Twist         2   Novel             1837
5                            The Hobbit         5 Fantasy             1937
6                  Pride and Prejudice         1 Classic             1813
```

대출 정보에서 누가 대출을 했는지 알고 싶다고 하자.

borrowing_id	book_id	borrower_id	borrow_date	return_date	status
1	1	1	2022-04-10	2022-04-25	Returned
2	3	2	2022-03-20	NA	On Loan
3	4	3	2022-04-05	NA	On Loan
4	2	4	2022-04-15	NA	On Loan
5	5	5	2022-03-30	2022-04-20	Returned
6	1	3	2022-04-26	NA	On Loan

borrower_id	name	email	member_since
1	John Smith	john.smith@example.com	2022-01-01
2	Emma Johnson	emma.johnson@example.com	2021-12-15
3	Michael Brown	michael.brown@example.com	2022-02-20
4	Sophia Wilson	sophia.wilson@example.com	2022-03-10
5	William Taylor	william.taylor@example.com	2022-04-05
6	Jane Doe	jane.doe@example.com	2022-03-05

```
borrowings |>
    left_join(borrowers, by = c("borrower_id" = "borrower_id"))
```

```
  borrowing_id book_id borrower_id borrow_date return_date   status
1            1       1           1  2022-04-10  2022-04-25 Returned
2            2       3           2  2022-03-20        <NA>  On Loan
3            3       4           3  2022-04-05        <NA>  On Loan
4            4       2           4  2022-04-15        <NA>  On Loan
5            5       5           5  2022-03-30  2022-04-20 Returned
6            6       1           3  2022-04-26        <NA>  On Loan
           name                    email member_since
```

```
1     John Smith     john.smith@example.com   2022-01-01
2   Emma Johnson   emma.johnson@example.com   2021-12-15
3  Michael Brown  michael.brown@example.com   2022-02-20
4  Sophia Wilson  sophia.wilson@example.com   2022-03-10
5 William Taylor william.taylor@example.com   2022-04-05
6  Michael Brown  michael.brown@example.com   2022-02-20
```

만약 이 결과를 가지고 그 도서의 제목 정보도 추가하고 싶을 수 있다. 그러면 다시 연결하면 된다.

```
borrowings |>
    left_join(borrowers, by = c("borrower_id" = "borrower_id")) |>
    left_join(books, by = c("book_id" = "book_id"))
```

```
  borrowing_id book_id borrower_id borrow_date return_date   status
1            1       1           1  2022-04-10  2022-04-25 Returned
2            2       3           2  2022-03-20        <NA>  On Loan
3            3       4           3  2022-04-05        <NA>  On Loan
4            4       2           4  2022-04-15        <NA>  On Loan
5            5       5           5  2022-03-30  2022-04-20 Returned
6            6       1           3  2022-04-26        <NA>  On Loan
           name                      email member_since
1     John Smith     john.smith@example.com   2022-01-01
2   Emma Johnson   emma.johnson@example.com   2021-12-15
3  Michael Brown  michael.brown@example.com   2022-02-20
4  Sophia Wilson  sophia.wilson@example.com   2022-03-10
5 William Taylor william.taylor@example.com   2022-04-05
6  Michael Brown  michael.brown@example.com   2022-02-20
                                  title author_id   genre publication_year
1                     Pride and Prejudice         1 Classic             1813
2             Murder on the Orient Express         3 Mystery             1934
3 Harry Potter and the Philosopher's Stone         4 Fantasy             1997
4                             Oliver Twist         2   Novel             1837
5                               The Hobbit         5 Fantasy             1937
6                     Pride and Prejudice         1 Classic             1813
```

이 도서관으 대출자들은 어떤 저자를 좋아하는지 알고 싶을 수 있다. 이런 경우 대출 정보와 대출자 정보를 연결하고, 그 정보를 가지고 도서 정보를 확인하고, 그 정보를 가지고 저자 정보를 확인하면 된다.

```
borrowings |>
    left_join(borrowers, by = c("borrower_id" = "borrower_id")) |>
    left_join(books, by = c("book_id" = "book_id")) |>
    left_join(authors, by = c("author_id" = "author_id"))
```

```
  borrowing_id book_id borrower_id borrow_date return_date   status
1            1       1           1  2022-04-10  2022-04-25 Returned
2            2       3           2  2022-03-20        <NA>  On Loan
3            3       4           3  2022-04-05        <NA>  On Loan
4            4       2           4  2022-04-15        <NA>  On Loan
5            5       5           5  2022-03-30  2022-04-20 Returned
6            6       1           3  2022-04-26        <NA>  On Loan
          name.x                     email member_since
1     John Smith     john.smith@example.com   2022-01-01
2   Emma Johnson   emma.johnson@example.com   2021-12-15
3  Michael Brown  michael.brown@example.com   2022-02-20
4  Sophia Wilson  sophia.wilson@example.com   2022-03-10
5 William Taylor william.taylor@example.com   2022-04-05
6  Michael Brown  michael.brown@example.com   2022-02-20
                                  title author_id   genre publication_year
1                     Pride and Prejudice         1 Classic             1813
2             Murder on the Orient Express         3 Mystery             1934
3 Harry Potter and the Philosopher's Stone         4 Fantasy             1997
4                             Oliver Twist         2   Novel             1837
5                               The Hobbit         5 Fantasy             1937
6                     Pride and Prejudice         1 Classic             1813
          name.y nationality birth_year
1     Jane Austen     British       1775
2 Agatha Christie     British       1890
3    J.K. Rowling     British       1965
4 Charles Dickens     British       1812
```

```
5          Tolkien     British       1892
6      Jane Austen     British       1775
```

조인은 테이블은 행의 개수가 많으면 헷갈리기 쉽상이다. 조인의 종류를 아주 간단히 정리해 보면 다음과 같다.

```
df1 <- tibble(x = c(1, 2), y = 2:1)
df2 <- tibble(x = c(3, 1), a = 10, b = "a")
```

```
knitr::kable(df1)
```

x	y
1	2
2	1

```
knitr::kable(df2)
```

x	a	b
3	10	a
1	10	a

`inner_join`은 두 테이블에서 일치되는 값이 있을 때 이뤄진다. 다음에서 `by = "x"`라고 했으므로 `x` 열의 보면 된다. 공통으로 가지고 있는 값은 `1`이다. 그래서 `df1`과 `df2`에서 이 `1`에 해당되는 행을 찾아서 결합한다.

```
df1 |>
    inner_join(df2, by = "x") |>
    knitr::kable()
```

x	y	a	b
1	2	10	a

left_join은 왼쪽 테이블의 모든 행을 포함하고, 오른쪽 테이블에서 일치하는 값이 있는 경우 결합한다. 다음에서 by = "x"라고 했으므로 x열의 보면 된다. df1에는 x가 1인 행이 있고, df2에는 x가 1인 행이 있으므로 이 행은 결합된다. 반면 df1에는 x가 2인 행이 있고, df2에는 x가 2인 행이 없으므로 이 행은 결합되지 않지만 left join이므로 왼쪽 테이블의 행은 모두 포함된다. df2에는 여기에 대응하는 값이 없기 때문에 NA로 채워진다.

```
df1 |>
    left_join(df2, by = "x") |>
    knitr::kable()
```

x	y	a	b
1	2	10	a
2	1	NA	NA

full_join은 두 테이블의 모든 행을 포함하고, 일치하는 값이 있는 경우 결합한다. 다음에서 by = "x"라고 했으므로 x열의 보면 된다. df1에는 x가 1인 행이 있고, df2에는 x가 1인 행이 있으므로 이 행은 결합된다. 반면 df1에는 x가 2인 행이 있지만 df2에는 없고, df2에는 x가 3인 행이 있지만 df1에는 없다. 이 행들 결합되지 않지만 full join이므로 모두 포함된다. 값이 존재하지 않은 부분은 NA로 채워진다.

```
df1 |>
    full_join(df2, by = "x") |>
    knitr::kable()
```

x	y	a	b
1	2	10	a
2	1	NA	NA
3	NA	10	a

15 윈도우 함수(window functions)

윈도우 함수(window function)는 그림 15.1와 같이 행 집합에 대해 계산을 수행하는 함수들로, 다음과 같은 계산에 사용된다.

- 누적 계산
- 순위 계산
- 이전/이후의 값 참조

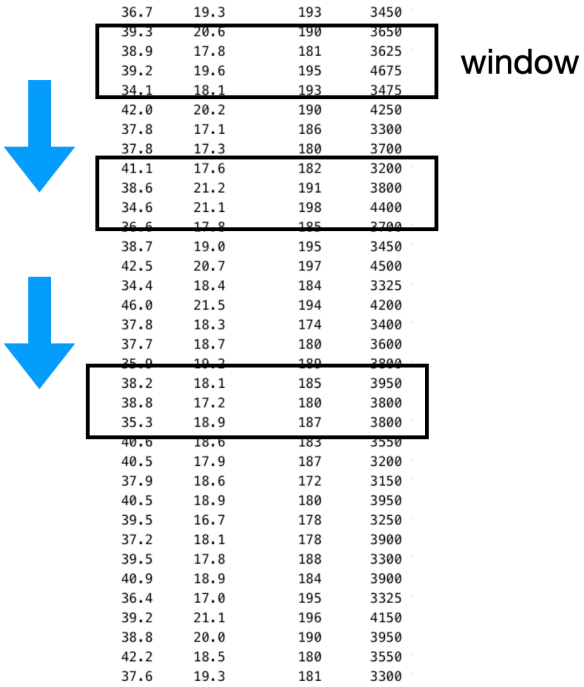


그림 15.1: 개략적인 window 함수의 작동법

간단한 데이터셋을 만들어 윈도우 함수의 작동 방식을 이해하자.

```
library(tidyverse)
df <- tibble(
    group = c("A", "A", "B", "B", "B"),
    value = c(10, 20, 10, 30, 20)
)
df
```

```
# A tibble: 5 x 2
  group value
  <chr> <dbl>
1 A        10
2 A        20
3 B        10
4 B        30
5 B        20
```

누적 합계는 cumsum() 함수를 사용한다.

```
df |>
    mutate(cumsum = cumsum(value))
```

```
# A tibble: 5 x 3
  group value cumsum
  <chr> <dbl>  <dbl>
1 A        10     10
2 A        20     30
3 B        10     40
4 B        30     70
5 B        20     90
```

만약에 그룹화 데이터프레임을 사용하면 그룹별로 누적 합계를 계산한다.

```
df |>
    group_by(group) %>%
```

```
    mutate(cumsum = cumsum(value))
```

```
# A tibble: 5 x 3
# Groups:   group [2]
  group value cumsum
  <chr> <dbl>  <dbl>
1 A        10     10
2 A        20     30
3 B        10     10
4 B        30     40
5 B        20     60
```

앞, 뒤의 값을 참조할 때는 `lag()`, `lead()` 함수를 사용한다.

```
df %>%
    mutate(
        previous_value = lag(value),
        next_value = lead(value)
    )
```

```
# A tibble: 5 x 4
  group value previous_value next_value
  <chr> <dbl>          <dbl>      <dbl>
1 A        10             NA         20
2 A        20             10         10
3 B        10             20         30
4 B        30             10         20
5 B        20             30         NA
```

만약 그룹화 데이터프레임을 사용하면 그룹별로 앞, 뒤의 값을 참조한다.

```
df %>%
    group_by(group) %>%
    mutate(
```

```
        previous_value = lag(value),
        next_value = lead(value)
    )
```

```
# A tibble: 5 x 4
# Groups:   group [2]
  group value previous_value next_value
  <chr> <dbl>          <dbl>      <dbl>
1 A        10             NA         20
2 A        20             10         NA
3 B        10             NA         30
4 B        30             10         20
5 B        20             30         NA
```

순위를 부여하는 함수들이 있다.

- `row_number()`: 순위를 부여하는 데 값이 같으면 데이터 순서대로 순위를 부여한다.
- `min_rank()`: 순위를 부여하는 데 값이 같으면 같은 순위를 부여하고, 그 다음 순위는 건너뛴다.
- `dense_rank()`: 순위를 부여하는 데 값이 같으면 같은 순위를 부여하고, 그 다음 순위는 건너뛰지 않는다.

```
df %>%
    mutate(
        row_number_value = row_number(value),
        rank_value = min_rank(value),
        dense_rank_value = dense_rank(value)
    )
```

```
# A tibble: 5 x 5
  group value row_number_value rank_value dense_rank_value
  <chr> <dbl>            <int>      <int>            <int>
1 A        10                1          1                1
2 A        20                3          3                2
3 B        10                2          1                1
```

```
4 B        30                5          5                3
5 B        20                4          3                2
```

만약 그룹화 데이터프레임을 사용하면 그룹별로 순위를 부여한다.

```
df %>%
    group_by(group) %>%
    mutate(
        row_number_value = row_number(value),
        rank_value = min_rank(value),
        dense_rank_value = dense_rank(value)
    )
```

```
# A tibble: 5 x 5
# Groups:   group [2]
  group value row_number_value rank_value dense_rank_value
  <chr> <dbl>            <int>      <int>            <int>
1 A        10                1          1                1
2 A        20                2          2                2
3 B        10                1          1                1
4 B        30                3          3                3
5 B        20                2          2                2
```

16 R 문자열과 stringr 패키지

문자열(characteor)은 R에서 텍스트 데이터를 의미하고, 문자열은 작은따옴표나 큰따옴표로 둘러싸서 표현한다. 작은따옴표를 사용하든 큰따옴표를 사용하든 차이는 없다. 다만 작은따옴표 안에 다시 작은따옴표를 바로 쓸 수 없고, 큰따옴표 안에 바로 큰따옴표를 쓸 수는 없다.

```
mt_name <- "한라산"
mt_name
```

```
[1] "한라산"
```

```
pt_name <- '홍길동'
pt_name
```

```
[1] "홍길동"
```

base R에는 이런 문자열을 다루는 함수들이 많이 있다. 하지만 함수별로 사용법에 일관성이 없어서 익히기가 어렵다. 그래서 R의 Tidyverse 접근법을 따르는 `stringr`이라는 패키지는 관련된 함수들을 일관성 있게 정리하여 제공한다. 따라서 R을 처음 배우는 경우라 할지라도 이 패키지를 사용하여 문자열 데이터를 다르는 방법을 배우는 것이 좋을 것으로 생각된다.

이 패키지를 로딩하자.

```
library(stringr)
```

16.1 인코딩(Encoding)

R 언어에서 문자열은 플레인 텍스트(plain text)로 저장된다. 그런데 이 플레인 텍스트라고 할지라도, 저장하는 알고리즘(인코딩)에 따라 실제로 컴퓨터에 저장되는 방식이 다르다.

인코딩(encoding)이라는 것은 컴퓨터가 문자를 숫자로 바꾸는 방법을 의미한다. 예를 들어서 "A"라는 문자는 65라는 숫자로 저장된다. 이때 "A"라는 문자를 65라는 숫자로 바꾸는 방법이 인코딩이다.

R `4.2.0` 버전부터는 윈도우(Windows), 맥오에스(macOS), 리눅스(Linux) 시스템에 상관없이 모두 UTF-8 인코딩을 사용한다. UTF-8은 유니코드(Unicode)라는 국제 표준에 따라 만들어진 인코딩 방식으로, 전 세계의 모든 문자를 표현할 수 있는 인코딩 방식이다.[1]

Unicode 시스템에서 한글 문자는 국어사전에서 순서에 맞게 배열되어 있으며, 어떤 글자 하나는 하나의 유니코드 코드포인트와 대응하게 된다. 그래서 하나의 글자는 글자 하나로 카운트되고, 문자열끼리 비교도 가능해지고, 이에 따라 정렬도 가능해진다.

```
"가나다" > "다라마"
```

```
[1] FALSE
```

```
"가나다" < "다라마"
```

```
[1] TRUE
```

현재 R 세션에서 사용하고 있는 인코딩 방식을 확인하려면 `Sys.getlocale()` 함수를 사용한다(저자의 컴퓨터는 맥오에스이다).

```
Sys.getlocale()
```

```
[1] "en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8"
```

[1] 유니코드와 UTF-8에 대한 자세한 설명은 유니코드(Unicode)와 UTF-8 문서를 참조하자.

16.2 정규 표현식(Regular Expression)은 다음 장에서

정규 표현식(Regular Expression, regex)이란 문자열에서 특정한 패턴을 찾기 위해 사용하는 방법으로, 잘 사용하면 아주 간결하면서도 강력한 기능을 제공한다.

`stringr` 패키지의 많은 함수들을 정규 표현식을 사용하는데, 이것을 사용하려면 정규 표현식을 알아야 하기 때문에, 정규 표현식 관련된 내용은 다음 장으로 넘겨 설명한다.

16.3 문자열 길이

R에서 문자열 데이터는 벡터에 저장된다. 그리고 벡터에 저장된 문자열은 각각의 길이(length)는 모두 1이다. 문자열 자체의 길이를 알려면 base R인 경우 `nchar()` 함수를 사용하고, `stringr` 패키지에서는 `str_length()` 함수를 사용한다.

```
about_R <- c("R은 자유 소프트웨어이다.",
             "R은 통계 분석을 위한 프로그래밍 언어이다.",
             "R은 많은 기여자들이 참여하는 공동프로젝트입니다.")
length(about_R)
```

```
[1] 3
```

다음은 base R `nchar()` 함수를 사용하였다.

```
nchar(about_R)
```

```
[1] 14 24 27
```

다음은 `stringr` 패키지의 `str_length()` 함수를 사용하였다.

```
str_length(about_R)
```

```
[1] 14 24 27
```

위와 같이 `stringr` 패키지의 거의 모든 함수는 `str_`로 시작한다. 그리고 대부분 벡터화되어 있다.

16.4 문자열 결합

- `stringr` 패키지의 `str_c()` 함수를 사용하여 문자열을 결합할 수 있다. 이때 `sep` 인자를 사용하여 구분자를 지정할 수 있다.

이 함수는 문자열 벡터를 구성하는 문자열과 문자열을 결합하여 새로운 문자열 벡터를 만든다. 마치 정수형 벡터의 값을 더하는 것과 비슷하다고 생각하면 좋다.

```
x <- c(1, 3, 5)
y <- c(2, 4, 6)
x + y
```

```
[1]  3  7 11
```

```
s1 <- c("A", "B", "C")
s2 <- c("1", "2", "3")
str_c(s1, s2)
```

```
[1] "A1" "B2" "C3"
```

이렇게 같은 위치에 있는 요소 대 요소로 묶인다. 위 코드는 이해를 위한 것이고, 다음과 같이 작성해도 같은 방식이라는 것을 이해할 필요가 있다.

```
str_c(c("A", "B", "C"), 1:3)
```

```
[1] "A1" "B2" "C3"
```

문자열을 결합할 때 사용할 문자를 `sep` 인자에 지정할 수 있다.

```
str_c(s1, s2, sep = "-")
```

```
[1] "A-1" "B-2" "C-3"
```

- `str_flatten()` 함수는 문자열 벡터의 요소들을 결합하여 하나의 문자열로 만든다.

```
str_flatten(s1)
```

```
[1] "ABC"
```

collapse 인자를 사용하여 구분자를 지정할 수 있다.

```
str_flatten(s1, collapse = "-")
```

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

- str_glue() 함수는 문자열을 결합할 수 있고, 중괄호 {} 안에 R 코드를 넣어 계산한 값을 문자열로 결합시킬 수 있다.

```
x <- 1
str_glue("A는 {x}입니다.")
```

```
A는 1입니다.
```

다음 예시처럼 str_glue() 함수는 인자로 준 모든 문자열을 결합하면서 {}에 R 표현식을 넣어서 계산한 값을 문자열로 결합할 수 있다.

```
name <- "Fred"
age <- 50
anniversary <- as.Date("1991-10-12")
str_glue(
  "My name is {name}, ",
  "my age next year is {age + 1}, ",
  "and my anniversary is {format(anniversary, '%A, %B %d, %Y')}."
)
```

```
My name is Fred, my age next year is 51, and my anniversary is Saturday, October 12, 1991.
```

이 문자열에 {}을 문자열로 포함시키고자 한다면 {{}}를 사용하면 된다.

```
str_glue("My name is {name}, not {{name}}.")
```

```
My name is Fred, not {name}.
```

또 str_glue() 함수에서 데이터의 값을 인자로 지정해 줄 수도 있다.

```
str_glue(
  "My name is {name}, ",
  "and my age next year is {age + 1}.",
  name = "Joe",
  age = 40
)
```

```
My name is Joe, and my age next year is 41.
```

- str_glue_data() 함수는 데이터프레임을 인자로 받아서, 이 데이터프레임에 있는 값을 사용하여 문자열을 만드는 데 사용된다.

```
library(dplyr)
result_df <- penguins %>%
  group_by(species) %>%
  summarize(avg_mass = mean(body_mass, na.rm = TRUE))
result_df
```

```
# A tibble: 3 x 2
  species   avg_mass
  <fct>        <dbl>
1 Adelie       3701.
2 Chinstrap    3733.
3 Gentoo       5076.
```

result_df 데이터프레임의 species 열의 값과 avg_mass 열의 값을 사용하여 문자열을 만들고자 한다면 다음과 같이 작성한다. 각 행에서 열의 값을 가지고 와서 문자열을 만든다. ㄴ

```
result_df |>
  str_glue_data(
    "{species} 종의 평균 체중은 {round(avg_mass/1000, 2)} kg이다."
  )
```

```
Adelie 종의 평균 체중은 3.7 kg이다.
Chinstrap 종의 평균 체중은 3.73 kg이다.
Gentoo 종의 평균 체중은 5.08 kg이다.
```

16.5 공백 제거, 패딩 문자 추가, 말줄임표로 표시

- `str_trim()` 함수는 문자열의 앞과 뒤에 있는 공백을 제거한다. `side` 인자를 사용하여 앞, 뒤, 양쪽 모두를 제거할 수 있다. `side` 인자의 기본값은 `both`이다.

```
str_trim("   Hello World!   ")
```

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

- `str_squish()` 함수는 문자열의 앞과 뒤에 있는 공백을 제거하고, 중간에 있는 여러 개의 공백들도 하나로 줄여준다.

```
str_squish("   Hello        World!   ")
```

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

- `str_pad()` 함수는 문자의 앞, 뒤, 양쪽에 패딩 문자를 추가한다. `width` 인자에 지정한 길이만큼 문자열을 만들고, 부족한 부분을 `pad` 인자로 지정한 문자로 채운다.

```
str_pad("Hello", width = 10, side = "both", pad = "*")
```

```
[1] "**Hello***"
```

- `str_trunc()` 함수는 전체 문자열의 최대 길이를 지정하고, 그 길이를 초과하는 부분은 말줄임표(ellipsis, `...`)를 가지는 문자열로 바꾼다.

```
str_trunc("Hello World! This is a long string.", width = 20)
```

```
[1] "Hello World! This..."
```

공백과 ...(3개)를 합쳐 최대 20개의 문자로 구성된 문자열을 만든다. side 인자로 말줄임표가 어디에 위치할지를 지정할 수 있다. 기본값은 right이다.

16.6 대소문자 변환, 제목 또는 문장 형태 변환

영어에는 대소문자가 있다. 문자열을 대문자, 소문자, 제목 형태로 변환할 수 있는 함수들이 있다.

- str_to_lower() 함수는 문자열을 소문자로 변환한다.
- str_to_upper() 함수는 문자열을 대문자로 변환한다.
- str_to_title() 함수는 문자열을 제목 형태로 변환한다. 제목 형태란 각 단어의 첫 글자를 대문자로 바꾸고 나머지 글자는 소문자로 바꾸는 것이다.
- str_to_sentence() 함수는 문자열을 문장 형태로 변환한다. 문장 형태란 첫 글자를 대문자로 바꾸고 나머지 글자는 소문자로 바꾸는 것이다.

```
dog <- "The quick brown dog"
str_to_upper(dog)
```

```
[1] "THE QUICK BROWN DOG"
```

```
str_to_lower(dog)
```

```
[1] "the quick brown dog"
```

```
str_to_title(dog)
```

```
[1] "The Quick Brown Dog"
```

```
str_to_sentence("the quick brown dog")
```

```
[1] "The quick brown dog"
```

16.7 문자열에서 일부 문자열 추출과 수정

- `str_sub()` 함수는 문자열에서 **위치를 기반으로** 일부 문자열을 추출한다. `str_sub(string, start=1L, end=-1L)` 형태로 사용한다.
    - R에서 인덱스가 1에서 시작하고, 끝도 inclusive 이다.
    - `start`와 `end`는 음수로 지정할 수도 있다. `-1`은 문자열의 마지막 문자를 의미한다.

```
about <- "R은 많은 기여자들이 참여하는 공동프로젝트입니다."
str_sub(about, 1, 5)
```

```
[1] "R은 많은"
```

- `str_sub_all()` 함수는 복수의 문자열에서 문자열을 추출할 때 사용한다. `start`와 `end`에 벡터를 지정하여, 이 위치 조합에 따라 문자열을 추출할 수도 있다.

```
x <- c("abcde", "ghifgh")
str_sub_all(x, start = 1, end = 2)
```

```
[[1]]
[1] "ab"

[[2]]
[1] "gh"
```

```
str_sub_all(x, start = c(1, 2), end = c(2, 4))
```

```
[[1]]
[1] "ab"  "bcd"

[[2]]
[1] "gh"  "hif"
```

- `str_sub()` 함수를 할당 <- 좌측에 놓으면, 이 위치에 있는 문자열을 수정할 수 있다.

```r
x <- c("abcde", "ghifgh")
str_sub(x, 1, 2) <- "XY"
x
```

```
[1] "XYcde"  "XYifgh"
```

위치가 아니라 워드 프로세서 등에서 “찾아서 바꾸기”와 같은 기능은 “정규 표현식”을 사용해야 하는데, 이 내용은 다음 장에서 설명한다.

16.8 정리

여기에선 `stringr` 패키지를 사용한 기본 문자열 처리 방법을 설명하였다. 다음 장에서는 정규 표현식(Regular Expression)을 사용하여 문자열을 처리하는 방법을 설명한다. 정규 표현식은 문자열에서 특정한 패턴을 찾기 위해 사용하는 방법으로, 잘 사용하면 아주 간결하면서도 강력한 기능을 제공한다.

17 R 정규 표현식(Regular Expression)

정규 표현식(Regular Expression, regex)은 문자열 패턴을 정의하는 문법이다. 정규 표현식은 문자열을 검색하거나 조작하는 데 사용되는 강력한 도구이다. 이 도구는 1970년대 전후로 개발되었으며, 그 이후로 많은 프로그래밍 언어와 텍스트 편집기에서 널리 사용되고 있다.

처음 배울 때는 복잡하고 어려워 보이지만 충분히 배울 가치가 있다. 완전히 암기하고 이해하지 않아도 개념적으로 이해한 후에 ChatGPT 질의 등을 통해서 필요한 패턴을 만들어 사용할 수도 있을 것이다..

다음 예를 보자. `stringr` 패키지에는 이런 정규 표현식을 사용하여 문자열을 다루는 다양한 함수들이 포함되어 있다.

```
library(stringr)
```

`str_view()` 함수는 문자열에서 정규 표현식에 해당하는 부분을 강조하여 보여준다. 위 예에서는 두 번째 인자 `"ab+"`는 a다음에 b가 1개 이상 있는 경우를 찾는 정규 표현식이다.

```
str_view(c("abc", "aBc", "abc"), "ab+")
```

```
[1] | <ab>c
[3] | <ab>c
```

`str_view()` 함수는 이 패턴을 첫 번째 인자로 지정한 문자열 벡터의 각 요소마다 적용하여 패턴에 맞는 경우가 있는 경우 그 부분을 <>안에 강조하여 보여준다.

17.1 정규표현식을 해석하는 방법

- 정규 표현식은 근본적으로 논리(logic)에 바탕을 둔 도구이기 때문에 **논리적**으로 생각할 필요가 있다.
- 한 글자씩, 왼쪽에서 오른쪽으로 읽어 나간다.

- 텍스트 파일과 같이 여러 행(line)을 가지고 있는 경우, 하나의 문자열(string)을 대상으로 할지, 하나의 행을 대상으로 할지 주의한다.
- 영어의 경우 대소문자를 구분하는 것이 디폴트이다.
- 글자의 종류와 집합 관계에 주의를 기울인다.

    - Characters는 letters와 digits, whitespace, 문장부호 등을 모두 포함한다.
    - Letters는 `a`, `b`, `c` … 와 `A`, `B`, `C`… 등을 말한다.

    - Digits은 `0, 1, 2, 3,...9`을 말한다.
    - Whitespaces는 인쇄되지는 않아도 characters의 일부이라는 점을 기억하자.
        * 빈칸(`" "`), 탭(`\t`), 줄바꿈(`\n` 또는 `\r\n`) 등

17.2 정규 표현식의 기초

17.2.1 Character의 종류

정규 표현식에서 문자들 그대로 그 자체가 패턴이 되는 경우를 **literal characters**라고 한다. 특수한 기능을 하는 문장 부호를 **metacharacters**라고 한다. Metacharacters에는 다음과 같은 것들이 있으며, 이것들을 제외한 문자, 숫자 등을 literal characters라고 한다.

- Backslash: `\`
- Caret: `^`
- Dollar sign: `$`
- Dot: `.`
- Pipe symbol: `|`
- Question mark: `?`
- Asterisk: `*`
- Plus sign: `+`
- Opening parenthesis: `(`
- Closing parenthesis: `)`
- Opening square bracket: `[`
- Opening curly brace: `{`

그래서 `"ab"`라는 패턴은 `"a"`와 그 다음에 `"b"`가 따라오는 `"ab"`에 매칭된다. `"ab+`은 `+`가 앞의 `"b"`가 1개 이상있는 경우를 의미하여 `"ab"`, `"abb"`, ‘`"abbb"`, `"abbbb"` 등과 매칭된다.

17.2.2 Charater class: 단 하나의 문자에 매칭

- `[abc]`는 `[]` 안에 있는 문자 중 하나와 매칭되어, `"a"` 또는 `"b"`, 또는 `"c"`와 매칭된다.
- `[^abc]`는 `[]` 안에 `^`가 있으면 `[]` 안에 있는 문자 이외의 문자와 매칭된다. 예를 들어, `[^a-z]`는 소문자 알파벳이 아닌 문자와 매칭된다(여집합).
- `[]` 안에 `-`을 사용하여 문자 범위를 지정할 수 있다. 예를 들어, `[a-z]`는 소문자 알파벳 가운데 하나를 의미하고, `[A-Z]`는 대문자 알파벳 가운데 하나를 의미한다. `[0-9]`는 숫자 0부터 9까지 가운데 하나를 의미한다. 이런 문자 범위는 `[a-zA-Z0-9]`와 같이 조합하여 사용할 수 있다. 이 경우는 소문자, 대문자, 숫자 가운데 하나와 매칭된다.

다음은 회색을 의미하는 단어인 `gray`와 `grey`를 찾는 예이다.

```
my_str <- c("I like gray color.", "I like grey color.", "No gry color")
str_view(my_str, "gr[ae]y")
```

```
[1] | I like <gray> color.
[2] | I like <grey> color.
```

`stringr` 패키지에 과일 이름을 요소로 가지는 `fruit` 문자열 벡터가 있다. 이 가운데 5개만 사용하려고 한다.

```
fruit5 <- fruit[1:5]
fruit5
```

```
[1] "apple"       "apricot"     "avocado"     "banana"      "bell pepper"
```

이들 단어에서 영어의 모음(`aeiou`)이 아닌 것을 찾아 보려고 한다.

```
str_view(fruit5, "[^aeiou]")
```

```
[1] | a<p><p><l>e
[2] | a<p><r>i<c>o<t>
[3] | a<v>o<c>a<d>o
[4] | <b>a<n>a<n>a
[5] | <b>e<l><l>< ><p>e<p><p>e<r>
```

17.2.3 단축형 character class

- `\d` `\d`는 `[0-9]`와 같고, 숫자 하나를 의미한다. 그 여집합은 `\D`이다.

다음은 사용 예시이다.

```
str_view("031-123-4567", "\\d")
```

```
[1] | <0><3><1>-<1><2><3>-<4><5><6><7>
```

```
str_view("031-123-4567", "\\D")
```

```
[1] | 031<->123<->4567
```

참고로 R에서 `\d`를 문자열에서 정규 표현식으로 사용하기 위해서는 앞에 역슬래쉬를 이스케이핑해야 하기 때문에 `"\\d"`로 써야 한다.

R 버전 `4.0.0`부터는 raw string을 지원한다. raw string은 `r"(...)"` 문법을 사용한다. Raw string을 사용하는 경우에는 이스케이프를 하지 않아도 된다.

```
str_view("031-123-4567", r"(\d)")
```

```
[1] | <0><3><1>-<1><2><3>-<4><5><6><7>
```

```
str_view("031-123-4567", r"(\D)")
```

```
[1] | 031<->123<->4567
```

- `\w` `\w`는 `[a-zA-Z0-9_]`와 같고(word를 이루는 문자라는 뜻), 영문자(letter), 숫자(digit), 밑줄 문자(_) 가운데 하나를 의미한다. 그 여집합은 `\W`이다.

```
my_words <- c("abcd", "a bc", "대한민국", "s125", "s_class", "s_*class")
str_view(my_words, r"(\w\w\w\w)")
```

```
[1] | <abcd>
[3] | <대한민국>
[4] | <s125>
[5] | <s_cl>ass
[6] | s_*<clas>s
```

- **`\s`** `\s`는 whitespace를 의미하고, `[ \t\n\r\f\v]`와 같다. 즉 공백 문자(space), 탭 문자(tab), 줄바꿈 문자(newline), 캐리지 리턴 문자(carriage return), 폼 피드 문자(form feed), 수직 탭 문자(vertical tab) 가운데 하나를 의미한다. 그 여집합은 `\S`이다.

```
my_words <- c("abcd", "a bc", "대한민국", "s125", "s_class", "s_*class")
str_view(my_words, r"(\w\s\w)")
```

```
[2] | <a b>c
```

이런 단축 클래스는 `[]`에 넣을 수도 있다. 다음은 숫자이거나 - 가운데 하나를 의미한다.

```
text = "seoul-2024-가-1234"
str_view(text, r"([\d-])")
```

```
[1] | seoul<->2<0><2><4><->가<->1<2><3><4>
```

17.2.4 Dot: .

- `.`은 모든 characters(letters, digits, spaces 등 모두) 가운데 한 글자와 매칭된다. 단, 줄바꿈 문자는 제외된다. 가장 범위가 넓다.

다음은 `"..."` 패턴을 찾는 예이다. 공백이든 숫자이든 문장부호이든 줄바꿈만 아닌 경우는 모두 3개의 문자와 매칭된다.

```
texts = c("   ", "abc", "1234", "///"," @#$", "대한민국", "ab\ncde")
str_view(texts, r"(...)")
```

```
[1] | <   >
[2] | <abc>
[3] | <123>4
[4] | <///>
[5] | < @#>$
[6] | <대한민>국
[7] | ab
    | <cde>
```

다음 경우는 "a" 다음 3개의 문자가 오고 그 다음 "e"가 오는 경우를 찾는 예이다.

```
str_view(fruit, "a...e")
```

```
 [1] | <apple>
 [7] | bl<ackbe>rry
[48] | mand<arine>
[51] | nect<arine>
[62] | pine<apple>
[64] | pomegr<anate>
[70] | r<aspbe>rry
[73] | sal<al be>rry
```

.은 나중에 설명할 *에 붙여서 ".*"와 같은 형태를 많이 사용한다. 이 경우는 *가 0개 이상을 의미하기 때문에 .과 결합하여 0개 이상의 모든 문자와 매칭된다.

17.2.5 Alternation: 정규식들 가운데 하나와 매칭, |

|는 정규표현식들 가운데 하나와 매칭된다. 예를 들어 "apple|banana"는 "apple" 또는 "banana"와 매칭된다. 다음은 fruit 벡터에서 apple 또는 banana를 찾는 예이다.

```
str_view(fruit, "apple|banana")
```

```
 [1] | <apple>
 [4] | <banana>
[62] | pine<apple>
```

[]를 사용하여 정의되는 character class는 그 가운데 한 글자와 매칭되는데, |은 정규식들 가운데 하나와 매칭된다.

다음은 fruit 벡터에서 apple, banana, kiwi를 찾는 예이다.

```
p <- str_flatten(c("apple", "banana", "kiwi"), "|")
p
```

```
[1] "apple|banana|kiwi"
```

```
str_view(fruit, p)
```

```
 [1] | <apple>
 [4] | <banana>
[42] | <kiwi> fruit
[62] | pine<apple>
```

17.2.6 Anchor: ^, $

- ^는 문자열의 시작을 의미한다. 예를 들어 "^a"는 "a"로 시작하는 문자열과 매칭된다.
- $는 문자열의 끝을 의미한다. 예를 들어 "a$"는 "a"로 끝나는 문자열과 매칭된다.

다음은 fruit 벡터에서 a로 시작하는 문자열을 찾는 예이다.

```
str_view(fruit, "^a")
```

```
[1] | <a>pple
[2] | <a>pricot
[3] | <a>vocado
```

다음은 fruit 벡터에서 a로 끝나는 문자열을 찾는 예이다.

```
str_view(fruit, "a$")
```

```
 [4] | banan<a>
[15] | cherimoy<a>
[30] | feijo<a>
[36] | guav<a>
[56] | papay<a>
[74] | satsum<a>
```

하나의 패턴에서 `"^xxx$"`와 같은 형태로 패턴을 정의하면 전체 문자열이 `xxx`와 매칭되는지 확인할 수 있다. 다음은 `fruit` 벡터에서 `apple`과 매칭되는 문자열을 찾는 예이다.

```
str_view(fruit, "^apple$")
```

```
[1] | <apple>
```

17.2.7 Quantifier

quantifier는 앞의 문자나 패턴이 몇 번 반복되는지를 지정하는 것이다. 다음과 같은 quantifier가 있다.

- `*`: 0개 이상
- `+`: 1개 이상
- `?`: 0개 또는 1개
- `{n}`: n개
- `{n,}`: n개 이상
- `{n,m}`: n개 이상 m개 이하
- `{,m}`: m개 이하

17.2.8 Capturing Group: ()

`()`는 괄호 안의 패턴을 그룹으로 묶어서 이것을 다른 곳에서 다시 참조할 수 있은 캡처링 그룹(capturing group)을 만든다. 상대적으로 non-capturing group이라는 개념도 있는데 base R에서 디폴트로 사용하는 PCRE는 non-capturing group을 지원하지 않기 때문이다.

정의된 capturing group을 다시 참조할 때는 `\1`, `\2`, …과 같이 사용하는 데 이런 것들을 백레퍼런스(backreference)라고 한다.

다음은 `fruit` 벡터에서 `(..)`로 2개의 문자를 그룹으로 묶었다. 이것을 다시 `\1`가 뒤에 오기 때문에 그 두 개의 문자가 다시 반복되는 경우를 찾는다.

```
str_view(fruit, r"((..)\1)")
```

```
 [4] | b<anan>a
[20] | <coco>nut
[22] | <cucu>mber
[41] | <juju>be
[56] | <papa>ya
[73] | s<alal> berry
```

다음은 anchor `^`와 `$`를 사용했기 때문에 문자열 전체를 대상으로 하는다. `(..)`으로 문자 2개를 그룹으로 묶었고, 끝에서 `\1`을 다시 사용했다. 그리고 중간에 `.*`이 있어서 모든 문자가 된다. 그래서 2개 문자가 앞과 뒤에 반복되는 경우에 매칭된다.

```
str_view(words, r"(^(..).*\1$)")
```

```
[152] | <church>
[217] | <decide>
[617] | <photograph>
[699] | <require>
[739] | <sense>
```

17.2.9 정규 표현식 우선 순위

정규 표현식에도 우선 순위가 중요하다. 다음과 같은 우선 순위가 있다.

1. `()`: 괄호 또는 그룹핑
2. `*`, `+`, `?`: Quanatifers
3. Concatenation: 문자와 문자, 문자와 패턴이 붙어 있는 경우
4. `|`: Alternation

그래서 `"ab+"`는 다음과 같이 해석된다.

- `+`는 quantifier이기 때문에 앞의 `"b"`에 딱 붙어서 적용된다.

- 그다음 "a"와 "b"가 붙어 있다(Concatenation).

그래서 "a"가 있고, 그 뒤에 "b"가 1개 이상 있는 경우에 ?매칭된다.

그리고, "ab|abc"라는 패턴을 생각해 보자. 이 경우 concatenation이 먼저 적용되기 때문에 "ab"와 "abc"가 각각의 패턴으로 해석된다. 그런 다음 |가 적용되기 때문에 "ab" 또는 "abc"와 매칭된다. 그런데 alternation이 적용될 때는 앞에 것이 매칭되면 뒤의 것은 무시된다. 따라서 "ab"가 매칭되는 경우에는 "abc"는 매칭되지 않는다.

17.2.10 Flags: 정규 표현식의 작동 방식을 변경하는 옵션

정규 표현식의 작동 방식을 변경하는 옵션을 **flags**라고 한다.

- `ignore_case = TRUE`: 대소문자를 구분하지 않음
- `dotall = TRUE`: .이 줄바꿈 문자도 포함하여 모든 문자와 매칭됨
- `multiline = TRUE`: 여러 줄을 대상으로 함

`stringr` 패키지에는 이런 옵션을 줄 수 있게 `regex()` 함수를 제공한다. 이 함수 안에 패턴과 이런 옵션을 주면 된다.

정규 표현식은 디폴트로 대소문자를 구분한다. 이것을 무시하게 하는 것이 `ignore_case = TRUE`이다.

```
str_view(c("Apple", "apple", "banana"), regex("apple", ignore_case = TRUE))
```

```
[1] | <Apple>
[2] | <apple>
```

.은 디폴트로 줄바꿈 문자를 제외한 모든 문자와 매칭된다. `dotall = TRUE`를 주면 줄바꿈 문자도 포함하여 모든 문자와 매칭되게 만든다.

다음에서 .은 줄바꿈 문자를 제외하기 때문에 매칭되는 문자열이 없다.

```
str_view("a\nb\nc", "a.")
```

반면 `dotall = TRUE`를 주면 줄바꿈 문자도 포함하여 모든 문자와 매칭되기 때문에 매칭되는 문자열이 있다.

```
str_view("a\nb\nc", regex("a.", dotall = TRUE))
```

```
[1] | <a
    | >b
    | c
```

원래 ^와 $가 전체 문자열의 시작과 끝을 의미한다. multiline = TRUE를 주면 각 행의 시작과 끝을 의미하게 된다.

여러 행을 가진 문자열에서 디폴트는 ^와 $가 전체 문자열의 시작과 끝을 의미한다.

```
x <- "Line 1\nLine 2\nLine 3"
str_view(x, "^Line")
```

```
[1] | <Line> 1
    | Line 2
    | Line 3
```

multiline = TRUE를 주면 각 행의 시작과 끝을 의미하게 된다.

```
str_view(x, regex("^Line", multiline = TRUE))
```

```
[1] | <Line> 1
    | <Line> 2
    | <Line> 3
```

17.3 stringr과 정규 표현식의 활용

다음과 같은 Untidy data를 가지고 설명한다. 정규표현식은 이런 Untidy data를 다룰 때 유용하게 사용할 수 있다.

```
df <- tibble::tribble(
  ~Id,   ~Diagnoses,
  "pt1", "DM,HTN,ICH",
```

```
    "pt2", "DM",
    "pt3", "HTN,Obesity,Diabetes",
    "pt4", "HTN,DM,Hyperlipidemia",
    "pt5", "CI,HTN,Gout",
    "pt6", "RA,DM,CAD",
    "pt7", "AF,DM, HTN",
    "pt8", "AF,HTN,Hyperlipidemia"
  )
  df
```

```
# A tibble: 8 x 2
  Id    Diagnoses
  <chr> <chr>
1 pt1   DM,HTN,ICH
2 pt2   DM
3 pt3   HTN,Obesity,Diabetes
4 pt4   HTN,DM,Hyperlipidemia
5 pt5   CI,HTN,Gout
6 pt6   RA,DM,CAD
7 pt7   AF,DM, HTN
8 pt8   AF,HTN,Hyperlipidemia
```

17.3.1 문자열의 존재, 개수, 위치 확인하기

위 df 데이터프레임에서 당뇨가 있는 환자의 인원수를 확인하려고 한다. 당연히 Diagnoses 열에 DM이라는 문자열이 있는지 확인하여 카운트하면 될 것이다. Diabetes라고 된 데이터는 일단 무시하자. Tidy data라면 아주 간단할 문제이지만 Untidy하기 때문에 약간의 작업이 필요하다.

먼저 str_detect() 함수는 문자열에서 정규 표현식에 해당하는 부분이 있는지를 확인하여 TRUE 또는 FALSE로 반환한다. 문자열 벡터에서 각 요소에서 이런 과정이 반복된다.

```
library(stringr)
str_detect(df$Diagnoses, "DM")
```

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

이것을 sum() 함수와 결합하여 TRUE의 개수를 세면 된다.

```
sum(str_detect(df$Diagnoses, "DM"))
```

```
[1] 5
```

str_count() 함수는 문자열에서 정규 표현식에 해당하는 부분의 개수를 세어준다.

```
str_count(df$Diagnoses, "DM")
```

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

마찬가지로 이것을 sum() 함수와 결합하여 TRUE의 개수를 세면 된다.

```
sum(str_count(df$Diagnoses, "DM"))
```

```
[1] 5
```

어느 환자가 당뇨를 가지고 있는지 확인하고 싶을 수 있다. 이런 경우에는 문자열 벡터에서 정규 표현식에 해당하는 문자열을 가지고 있는 곳의 인덱스를 알려주는 str_which() 함수를 사용하면 된다.

```
str_which(df$Diagnoses, "DM")
```

```
[1] 1 2 4 6 7
```

이것을 dplyr 패키지의 filter() 함수와 결합하여, str_detect() 함수로 당뇨가 있는 환자만 추출할 수 있다.

```
library(dplyr)
df |>
  filter(str_detect(Diagnoses, "DM"))
```

```
# A tibble: 5 x 2
  Id    Diagnoses
  <chr> <chr>
1 pt1   DM,HTN,ICH
2 pt2   DM
3 pt4   HTN,DM,Hyperlipidemia
4 pt6   RA,DM,CAD
5 pt7   AF,DM, HTN
```

필요에 따라 `slice()` 함수와 `str_which()` 함수를 결합하여 사용할 수 있다. `slice()` 함수는 데이터 프레임에서 행을 추출하는 함수이다. 결과는 같다.

```
df |>
  slice(str_which(Diagnoses, "DM"))
```

```
# A tibble: 5 x 2
  Id    Diagnoses
  <chr> <chr>
1 pt1   DM,HTN,ICH
2 pt2   DM
3 pt4   HTN,DM,Hyperlipidemia
4 pt6   RA,DM,CAD
5 pt7   AF,DM, HTN
```

17.3.2 문자열 바꾸기

`df` 데이터프레임에서 당뇨병을 `DM` 또는 `Diabetes`로 표현하고 있더 데이터가 제대로 정리되지 않는다. 이것을 `Diabetes`로 통일하고 싶다. 이럴 때는 `str_replace()` 함수를 사용하여 `DM`을 `Diabetes`로 바꿔주면 된다.

```
str_replace(df$Diagnoses, "DM", "Diabetes")
```

```
[1] "Diabetes,HTN,ICH"            "Diabetes"
[3] "HTN,Obesity,Diabetes"        "HTN,Diabetes,Hyperlipidemia"
```

```
[5] "CI,HTN,Gout"                  "RA,Diabetes,CAD"
[7] "AF,Diabetes, HTN"             "AF,HTN,Hyperlipidemia"
```

이런 과정을 반복하면 df의 진단명을 모두 풀네임으로 바꿀 수도 있을 것이다. ……

str_replace() 함수는 문자열 벡터의 각 요소에 대하여 정규 표현식에 해당하는 부분을 찾아서 바꿔준다. 이때 정규 표현식에 해당하는 부분이 여러 개일 경우에는 첫 번째 것만 바꾼다. 모든 매칭을 바꾸려면 str_replace_all() 함수를 사용한다.

str_replace_all() 함수는 또한 여러 매칭/바꿀값을 한꺼번에 지정할 수도 있다. 단어와 단어 사이에는 _를 사용하여 붙였다.

```
str_replace_all(df$Diagnoses,
                c("DM" = "Diabetes",
                  "HTN" = "Hypertension",
                  "CAG" = "Coronary_Artery_Disease",
                  "CI" = "Cerebral_Infarct",
                  "ICH" = "Intracerebral_Hemorrhage",
                  "Af" = "Atrial_Fibrillation",
                  "RA" = "Rheumatoid_Arthritis",
                  "Gout" = "Gouty_Arthritis"))
```

```
[1] "Diabetes,Hypertension,Intracerebral_Hemorrhage"
[2] "Diabetes"
[3] "Hypertension,Obesity,Diabetes"
[4] "Hypertension,Diabetes,Hyperlipidemia"
[5] "Cerebral_Infarct,Hypertension,Gouty_Arthritis"
[6] "Rheumatoid_Arthritis,Diabetes,CAD"
[7] "AF,Diabetes, Hypertension"
[8] "AF,Hypertension,Hyperlipidemia"
```

이 기능을 사용하면 원래의 df 데이터프레임을 바꿀 수 있을 것이다. 이것을 ddf라는 새로운 데이터프레임에 저장했다.

```
ddf <- df |>
  mutate(
```

```
    Diagnoses = str_replace_all(Diagnoses,
                                c("DM" = "Diabetes",
                                  "HTN" = "Hypertension",
                                  "CAG" = "Coronary_Artery_Disease",
                                  "CI" = "Cerebral_Infarct",
                                  "ICH" = "Intracerebral_Hemorrhage",
                                  "AF" = "Atrial_Fibrillation",
                                  "RA" = "Rheumatoid_Arthritis",
                                  "Gout" = "Gouty_Arthritis"))
  )
ddf
```

```
# A tibble: 8 x 2
  Id    Diagnoses
  <chr> <chr>
1 pt1   Diabetes,Hypertension,Intracerebral_Hemorrhage
2 pt2   Diabetes
3 pt3   Hypertension,Obesity,Diabetes
4 pt4   Hypertension,Diabetes,Hyperlipidemia
5 pt5   Cerebral_Infarct,Hypertension,Gouty_Arthritis
6 pt6   Rheumatoid_Arthritis,Diabetes,CAD
7 pt7   Atrial_Fibrillation,Diabetes, Hypertension
8 pt8   Atrial_Fibrillation,Hypertension,Hyperlipidemia
```

17.3.3 문자열 나누기

stringr 패키지의 string_split() 함수는 문자열을 정규 표현식에 해당하는 부분을 기준으로 나누어준다. 이 함수를 사용하여 ddf 데이터프레임의 Diagnoses 열을 나누어 보자. 두 번째 인자에 나누는 기준이 되는 정규 표현식을 넣어주면 된다.

```
str_split(ddf$Diagnoses, ",")
```

```
[[1]]
[1] "Diabetes"                 "Hypertension"
```

```
[3] "Intracerebral_Hemorrhage"

[[2]]
[1] "Diabetes"

[[3]]
[1] "Hypertension" "Obesity"      "Diabetes"

[[4]]
[1] "Hypertension"   "Diabetes"       "Hyperlipidemia"

[[5]]
[1] "Cerebral_Infarct" "Hypertension"     "Gouty_Arthritis"

[[6]]
[1] "Rheumatoid_Arthritis" "Diabetes"             "CAD"

[[7]]
[1] "Atrial_Fibrillation" "Diabetes"            " Hypertension"

[[8]]
[1] "Atrial_Fibrillation" "Hypertension"        "Hyperlipidemia"
```

str_split() 함수는 리스트를 반환한다. 단수화한 형태로 반환하시려련 simplify = TRUE 옵션을 사용한다.

```
str_split(ddf$Diagnoses, ",", simplify = TRUE)
```

```
     [,1]                  [,2]           [,3]
[1,] "Diabetes"            "Hypertension" "Intracerebral_Hemorrhage"
[2,] "Diabetes"            ""             ""
[3,] "Hypertension"        "Obesity"      "Diabetes"
[4,] "Hypertension"        "Diabetes"     "Hyperlipidemia"
[5,] "Cerebral_Infarct"    "Hypertension" "Gouty_Arthritis"
[6,] "Rheumatoid_Arthritis" "Diabetes"    "CAD"
```

```
[7,] "Atrial_Fibrillation"  "Diabetes"     " Hypertension"
[8,] "Atrial_Fibrillation"  "Hypertension" "Hyperlipidemia"
```

이렇게 하면 행렬을 반환한다.

그런데 잘 관찰해 보면 7번 환자의 고혈압 진단이 Hypertension으로 되어 앞에 공백이 붙어 있다. 이럴 때는 str_trim() 함수를 사용하여 공백을 제거할 수 있다.

stringr 패키지에는 패턴의 행동을 바꾸기 위해 regex() 함수를 사용한다고 했는데, 이와 비슷하게 패턴을 만들때 boundary() 함수를 사용하여 단어의 경계를 정의할 수 있다. boundary("word")는 단어로 구분된 경계를 의미한다. 이 함수로 패턴을 재정의하여 작업해 보자.

```
str_split(ddf$Diagnoses, boundary("word"), simplify = TRUE)
```

```
     [,1]                  [,2]           [,3]
[1,] "Diabetes"            "Hypertension" "Intracerebral_Hemorrhage"
[2,] "Diabetes"            ""             ""
[3,] "Hypertension"        "Obesity"      "Diabetes"
[4,] "Hypertension"        "Diabetes"     "Hyperlipidemia"
[5,] "Cerebral_Infarct"    "Hypertension" "Gouty_Arthritis"
[6,] "Rheumatoid_Arthritis" "Diabetes"    "CAD"
[7,] "Atrial_Fibrillation" "Diabetes"     "Hypertension"
[8,] "Atrial_Fibrillation" "Hypertension" "Hyperlipidemia"
```

17.3.4 tidyr 패키지를 활용하여 데이터 분리, 추출

앞에서 사용한 ddf 데이터프레임에 새로운 열을 추가하여 (일부러) 다음과 같은 Untidy 데이터프레임을 만들어 보았다. 실제로 이런 방식으로 정리된 데이터를 종종 만나게 된다.

```
dddf <- ddf |>
  mutate(
    year_tx = str_c(c("2024", "2015", "2016", "2017", "2018", "2019", "2020", "2021"),
                    c("A", "A", "A", "A", "B", "B", "B", "B"))
     )
dddf
```

```
# A tibble: 8 x 3
  Id    Diagnoses                                    year_tx
  <chr> <chr>                                        <chr>
1 pt1   Diabetes,Hypertension,Intracerebral_Hemorrhage 2024A
2 pt2   Diabetes                                     2015A
3 pt3   Hypertension,Obesity,Diabetes                2016A
4 pt4   Hypertension,Diabetes,Hyperlipidemia         2017A
5 pt5   Cerebral_Infarct,Hypertension,Gouty_Arthritis 2018B
6 pt6   Rheumatoid_Arthritis,Diabetes,CAD            2019B
7 pt7   Atrial_Fibrillation,Diabetes, Hypertension   2020B
8 pt8   Atrial_Fibrillation,Hypertension,Hyperlipidemia 2021B
```

tidyr 패키지에는 pivot_longer()와 pivot_wider()라는 함수 이외에도 위와 같은 데이터를 정리하는 데 도움이 되는 다음 함수들을 가지고 있다.

- separate_longer_position()
- separate_longer_delim()
- separate_wider_position()
- separate_wider_delim()
- separate_wider_regex()

longer 함수는 데이터를 분리하여 여러 행에 배치하고, wider 함수는 데이터를 분리하여 여러 열에 배치한다. position 함수는 위치에 따라 분리하고, delim 함수는 구분자에 따라 분리한다.

dddf 데이터프레임에서 Diagnoses 열을 ,로 구분하여 여러 행으로 나누고, year_tx 열은 그대로 두고 싶다. 이럴 때는 separate_longer_delim() 함수를 사용한다.

```
library(tidyr)
dddf |>
  separate_longer_delim(Diagnoses, delim = ",")
```

```
# A tibble: 22 x 3
   Id    Diagnoses            year_tx
   <chr> <chr>                <chr>
 1 pt1   Diabetes             2024A
 2 pt1   Hypertension         2024A
```

```
 3 pt1   Intracerebral_Hemorrhage 2024A
 4 pt2   Diabetes                 2015A
 5 pt3   Hypertension             2016A
 6 pt3   Obesity                  2016A
 7 pt3   Diabetes                 2016A
 8 pt4   Hypertension             2017A
 9 pt4   Diabetes                 2017A
10 pt4   Hyperlipidemia           2017A
# i 12 more rows
```

year_tx 열에 대해서는 2개의 열로 나누고 싶다. 이럴 때는 separate_wider_position() 함수를 사용한다. position 함수를 사용할 때는 c() 함수를 사용하여 각 열의 너비(width)를 지정해 주어야 한다.

```
dddf |>
  separate_wider_position(year_tx, c(years = 4, tx = 1))
```

```
# A tibble: 8 x 4
  Id    Diagnoses                                       years tx
  <chr> <chr>                                           <chr> <chr>
1 pt1   Diabetes,Hypertension,Intracerebral_Hemorrhage  2024  A
2 pt2   Diabetes                                        2015  A
3 pt3   Hypertension,Obesity,Diabetes                   2016  A
4 pt4   Hypertension,Diabetes,Hyperlipidemia            2017  A
5 pt5   Cerebral_Infarct,Hypertension,Gouty_Arthritis   2018  B
6 pt6   Rheumatoid_Arthritis,Diabetes,CAD               2019  B
7 pt7   Atrial_Fibrillation,Diabetes, Hypertension      2020  B
8 pt8   Atrial_Fibrillation,Hypertension,Hyperlipidemia 2021  B
```

Diagnoses 열을 ,로 구분하여 여러 열로 나누고 싶다. 이럴 때는 separate_wider_delim() 함수를 사용한다. 이름을 지정하는 방법과 열의 개수가 맞지 않을 때 처리하는 방법 등은 ?separate_wider_delim 을 참조한다.

```
dddf |>
  separate_wider_delim(Diagnoses, delim = ",",
                       names = str_c("Diagnosis_", 1:3),
                       too_few = "align_start")
```

```
# A tibble: 8 x 5
  Id    Diagnosis_1          Diagnosis_2  Diagnosis_3                year_tx
  <chr> <chr>                <chr>        <chr>                      <chr>
1 pt1   Diabetes             Hypertension "Intracerebral_Hemorrhage" 2024A
2 pt2   Diabetes             <NA>          <NA>                      2015A
3 pt3   Hypertension         Obesity      "Diabetes"                 2016A
4 pt4   Hypertension         Diabetes     "Hyperlipidemia"           2017A
5 pt5   Cerebral_Infarct     Hypertension "Gouty_Arthritis"          2018B
6 pt6   Rheumatoid_Arthritis Diabetes     "CAD"                      2019B
7 pt7   Atrial_Fibrillation  Diabetes     " Hypertension"            2020B
8 pt8   Atrial_Fibrillation  Hypertension "Hyperlipidemia"          2021B
```

17.3.5 데이터 추출

dddf 데이터프레임에서 year_tx 열의 연도만 추출하고 싶다. 이럴 때는 str_extract() 함수를 사용하여 정규 표현식에 해당하는 부분을 추출할 수 있다.

다음 정규 표현식은 숫자 4개에 매칭된다. \d는 숫자 하나를 의미하고, {4}는 숫자가 4개라는 뜻이다.

```
str_extract(dddf$year_tx, "^\\d{4}")
```

```
[1] "2024" "2015" "2016" "2017" "2018" "2019" "2020" "2021"
```

```
dddf |>
  mutate(
    year = str_extract(year_tx, "^\\d{4}")
  )
```

```
# A tibble: 8 x 4
  Id    Diagnoses                                       year_tx year
  <chr> <chr>                                           <chr>   <chr>
1 pt1   Diabetes,Hypertension,Intracerebral_Hemorrhage  2024A   2024
2 pt2   Diabetes                                        2015A   2015
3 pt3   Hypertension,Obesity,Diabetes                   2016A   2016
4 pt4   Hypertension,Diabetes,Hyperlipidemia            2017A   2017
5 pt5   Cerebral_Infarct,Hypertension,Gouty_Arthritis   2018B   2018
6 pt6   Rheumatoid_Arthritis,Diabetes,CAD               2019B   2019
7 pt7   Atrial_Fibrillation,Diabetes, Hypertension      2020B   2020
8 pt8   Atrial_Fibrillation,Hypertension,Hyperlipidemia 2021B   2021
```

```
some_df <- tibble(x = c("202215TX", "202122LA", "202325CA"))
some_df
```

```
# A tibble: 3 x 1
  x
  <chr>
1 202215TX
2 202122LA
3 202325CA
```

some_df 데이터프레임에서 x 열에서 앞 4자리는 연도, 다음 2자리는 나이, 다음 2개는 장소라고 생각해 보자. str_extract() 함수를 사용하여 나이를 추출해 보자.

다음은 정규 표현식에서 (?<=...) 앞에서 설명하지 않은 *positive lookbehind** 문법이고 , (...) 안에 있는 패턴이 앞에 있어야 한다는 뜻이다. 즉, (?<=\\d{4})는 숫자 4개가 앞에 있어야 한다는 뜻이다. (?<=\\d{4})\\d{2}는 숫자 4개가 앞에 있고, 그 다음에 숫자 2개가 오는 경우를 의미한다.

```
str_extract(some_df$x, "(?<=\\d{4})\\d{2}")
```

```
[1] "15" "22" "25"
```

물론 앞에서 tidyr의 separate_wider_position() 함수를 사용하여 나이를 추출할 수도 있다.

```
some_df |>
  separate_wider_position(x, c(year = 4, age = 2, place = 2))
```

```
# A tibble: 3 x 3
  year  age   place
  <chr> <chr> <chr>
1 2022  15    TX
2 2021  22    LA
3 2023  25    CA
```

17.4 문자열 정렬

stringr 패키지의 str_sort() 함수로 문자열을 정렬한다.

```
places <- c("서울", "부산", "대구", "광주", "인천", "수원", "대전", "울산")
str_sort(places)
```

```
[1] "광주" "대구" "대전" "부산" "서울" "수원" "울산" "인천"
```

다음은 내림차순으로 정렬하는 예이다.

```
str_sort(places, decreasing = TRUE)
```

```
[1] "인천" "울산" "수원" "서울" "부산" "대전" "대구" "광주"
```

17.5 정리

정규 표현식은 문자열을 다룰 때 매우 유용한 도구이다. 정규 표현식을 사용하여 문자열을 검색하고, 바꾸고, 나누고, 추출하는 등의 작업을 수행할 수 있다. stringr 패키지는 정규 표현식을 쉽게 사용할 수 있도록 다양한 함수를 제공한다. 다음 cheatsheet에서 stringr과 정규 표현식이 잘 정리되어 있다.

- String manipulation with stringr

좀 더 강력한 정규 표현식을 사용하고 싶다면 구글의 `re2` 패키지를 고민할 수 있다.

- re2 깃허브

18 lubridate 패키지로 날짜와 시간 처리

프로그래밍 언어에서 날짜, 시간, 날짜/시간 데이터를 처리하는 것은 생각보다 간단하지 않다. 간단하지 않은 것은 여러 이유가 있다.

원래 날짜와 시간은 지구의 자전과 공전에 따른 **물리적**인 현상인데, 우리는 보통 달력이라는 **인위적**인 도구를 사용하는 것이 보통이다. 우리 한국만 하더라도 양력과 음력을 사용하고, 단기(檀紀)와 서기(西紀)가 있다. 영어권에서도 `1 May, 2025` 또는 `May 3, 2025` 등 표시 방법이 다르다.

우린 보통 1년을 365일로 친다. 하지만 지구 공전 시간은 물리적으로 365.2422일이다 보니 이것을 보정하지 않으면 4, 5년이 지나면 하루 차이가 나게 된다. 그래서 보통 4로 나누어 떨어지는 해를 윤년(閏年, leap year)이라고 하고, 그 해는 2월에 하루를 더 추가하는 것이 우리가 보통 쓰고 있는 그레고리 달력의 기준이다(좀 더 자세한 것은 인터넷을 참고한다). 따라서 2월 15일에서 3월 18일까지 기간을 계산하자면 윤년인지 아닌지를 알아야 된다.

컴퓨터는 물리적인 계산을 하기 때문에 물리적인 것과 인위적인 것을 처리해 주는 도구가 필요하다.

원래는 base R에는 날짜와 시간 처리 도구가 거의 없었다고 해도 과언이 아니다. 이 빈 지점을 채워주는 `lubridate` 패키지으로, 날짜, 시간, 날짜/시간 데이터를 처리하는 아주 강력한 기능을 제공하기 때문에 많은 사람들이 애용한다. Base R에서 날짜를 처리하는 함수들이 있기는 하지만 여기선 따로 설명하지 않는다.

이 패키지는 `tidyverse` 패키지 중 하나이며, 따라서 이 패키지를 사용하기 위해서는 `tidyverse` 패키지나 `lubridate` 패키지를 로딩하여 사용한다.

```
library(tidyverse)
```

또는 다음과 같이 따로 로딩해도 된다.

```
library(lubridate)
```

```
Attaching package: 'lubridate'
```

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

    date, intersect, setdiff, union
```

18.1 날짜, 시간 데이터와 관련된 문제들

날짜/시간 데이터를 사용하는 것들과 관련해서 흔히 마주칠 수 있는 문제를 생각해 보고 `lubridate` 패키지를 사용법에 대해 설명하고자 한다.

- 우선 R 언어에서 사용할 수 있게 날짜/시간 데이터로 변환되어야 한다.
    - 엑셀 데이터셋을 읽는 경우 문자열을 된 것을 날짜/시간 데이터로 바꿔줘야 한다. 또 그러면서 날짜/시간 표현이 어떻게 되어 있는지를 고려해야 한다. 원 데이터가 문자열로 `2025-05-17`, `2025년 5월 17일`, `2026/05/17`, `May 17, 2025` 등 다양하게 존재할 수 있다.
- 날짜와 시간 데이터를 가지고 여러 가지 연산을 수행할 수 있다. 오늘(날짜 데이터)을 기준으로 150일(기간) 뒤의 날짜를 어떻게 계산할지 고민해야 한다.
- 시간대(timezone) 문제가 있다. 나라가 크거나 지구 여러 곳에서 사용한다면 이런 것을 고려해야 한다.
- 어떤 나라에서 Daylight Saving Time(DST, 일광 절약 시간)을 수행한다.
- 사용자가 이해할 수 있게 날짜/시간 데이터를 정확하게 표시해 주어야 한다. 계산한 요일이 "금요일" 또는 "Friday" 등으로 상황에 맞게 표시해 줄 필요가 있다.

이와 같은 여러 문제들을 다루기 위해 `lubridate` 패키지에는 다양한 함수들이 존재한다.

18.2 텍스트 데이터를 날짜로 변환하기

다음과 같은 데이터프레임으로 시작해 보자.

```
library(tidyverse)

stroke_time <- tibble::tribble(
    ~Id,  ~StrokeOnset,       ~EventRecognition, ~ArrivalTime,      ~CtCompleted,
    "p1", "23-1-15 오후 2:25",  "exact",           "2023.1.15 15:00", "2023-1-15 15:56",
    "p2", "25-2-20 오전 11:30", "presumed",        "2025.2.20 17:00", "2025-2-20 18:05",
```

```
    "p3", "19-6-5 오후 01:00",  "exact",          "2019.6.5 14:00",  "2019-6-5 15:30"
)
stroke_time
```

```
# A tibble: 3 x 5
  Id    StrokeOnset        EventRecognition ArrivalTime     CtCompleted
  <chr> <chr>              <chr>            <chr>           <chr>
1 p1    23-1-15 오후 2:25  exact            2023.1.15 15:00 2023-1-15 15:56
2 p2    25-2-20 오전 11:30 presumed         2025.2.20 17:00 2025-2-20 18:05
3 p3    19-6-5 오후 01:00  exact            2019.6.5 14:00  2019-6-5 15:30
```

현재 위 데이터프레임에는 날짜/시간 정보가 "문자열"로 저장되어 있다. 이것을 날짜/시간 데이터로 변환하는 문제를 생각해 보자. 프로그래밍에서는 이런 종류의 작업을 파싱(parsing)이라고 한다.

`lubridate` 패키지에는 파싱에 필요한 다양한 함수들이 준비되어 있다.

- `y`(year), `m`(month), `d`(day)을 조합하여 날짜 데이터를 변환하는 함수들
- 거기에 `h`(hour), `m`(minute), `s`(second)을 조합하여 시간 데이터를 변환하는 함수들
    - 예를 들어 `ymd_hms()`는 문자열을 날짜/시간 데이터로 변환한다.

보통 ISO 8601 형식은 날짜/시간을 표현하는 표준 형식인데, `2025-05-17 14:30:00` 또는 `2025-05-17T14:30:00` 등으로 표현한다. 문자열 데이터가 이런 형식으로 저장되어 있으면 다음과 같이 변환할 수 있다.

```
ymd_hm(stroke_time$CtCompleted)
```

```
[1] "2023-01-15 15:56:00 UTC" "2025-02-20 18:05:00 UTC"
[3] "2019-06-05 15:30:00 UTC"
```

`ymd_hm()` 함수를 사용한 이유는 데이터가 연도-월-일 시간:분 형식으로 되어 있기 때문이다. 만약 한국 시간대로 변환하려면 `tz` 인자에 `"Asia/Seoul"`을 추가한다.

```
ymd_hm(stroke_time$CtCompleted, tz = "Asia/Seoul")
```

```
[1] "2023-01-15 15:56:00 KST" "2025-02-20 18:05:00 KST"
[3] "2019-06-05 15:30:00 KST"
```

만약 010210 문자열이 첫 두개가 날짜, 다음 두개가 월, 마지막 두개가 2자리 연도라면 다음과 같이 dmy() 함수를 사용해서 날짜/시간 데이터로 변환할 수 있다.

```
dmy(010210)
```

```
[1] "2010-02-01"
```

StrokeOnset과 ArrivalTime 열은 좀 특이하게 되어 있어서 ymd_hms() 함수를 사용해서 쉽게 변경하기 쉽지 않다. 이런 경우에는 parse_date_time() 함수를 사용해서 변환한다. 이 함수를 사용하려면 표 18.1에 정리된 것과 같은 읽을 문자열에 대한 패턴을 참고해야 한다.

표 18.1: lubridate에서 날짜/시간 정보에 대한 문자열 패턴

패턴	의미	예시
%Y	4자리 연도	2025
%y	2자리 연도	25
%m	2자리 월	01, 02, …, 12
%d	2자리 일	01, 02, …, 31
%H	24시간제 시간	00, 01, …, 23
%I	12시간제 시간	01, 02, …, 12
%M	분	00, 01, …, 59
%S	초	00, 01, …, 59
%p	AM/PM	AM, PM
%z	시간대 오프셋	+0900
%Z	시간대 이름	KST, UTC

예를 들어, “2025-05-17 14:30:45” 형식의 문자열을 파싱하려면:

```
parse_date_time("2025-05-17 14:30:45", orders = "%Y-%m-%d %H:%M:%S")
```

```
[1] "2025-05-17 14:30:45 UTC"
```

그래서 위 데이터프레임에서 ArrivalTime 열은 다음과 같이 변환할 수 있다.

```
parse_date_time(stroke_time$ArrivalTime, orders = "%Y.%m.%d %H:%M", tz = "Asia/Seoul")
```

```
[1] "2023-01-15 15:00:00 KST" "2025-02-20 17:00:00 KST"
[3] "2019-06-05 14:00:00 KST"
```

그런데 StrokeOnset 열은 좀 특이하다.

```
stroke_time$StrokeOnset
```

```
[1] "23-1-15 오후 2:25"  "25-2-20 오전 11:30" "19-6-5 오후 01:00"
```

이 경우에는 "오전", "오후" 등의 정보가 있어서 %p 패턴을 사용해서 될 것 같지만 잘 되지 않는다.

```
parse_date_time(stroke_time$StrokeOnset, orders = "%y-%m-%d %p %I:%M")
```

```
Warning: All formats failed to parse. No formats found.
```

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

이것은 "오전", "오후"를 인식하지 못한다. 이런 경우에는 stringr 패키지의 함수를 사용해서 문자열을 변환해야 한다.

```
library(stringr)
stroke_time$StrokeOnset %>%
    str_replace("오전", "AM") %>%
    str_replace("오후", "PM") %>%
    parse_date_time(orders = "%y-%m-%d %p %I:%M", tz = "Asia/Seoul")
```

```
[1] "2023-01-15 14:25:00 KST" "2025-02-20 11:30:00 KST"
[3] "2019-06-05 13:00:00 KST"
```

이제 위의 로직을 dplyr 패키지를 사용하여 한번에 정리해 보자.

```
new_df <- stroke_time %>%
    mutate(
        StrokeOnset = str_replace(StrokeOnset, "오전", "AM") %>%
            str_replace("오후", "PM") %>%
            parse_date_time(orders = "%y-%m-%d %p %I:%M", tz = "Asia/Seoul"),
        ArrivalTime = parse_date_time(ArrivalTime, orders = "%Y.%m.%d %H:%M", tz = "Asia/Seoul'
        CtCompleted = ymd_hm(CtCompleted, tz = "Asia/Seoul")
    )
new_df
```

```
# A tibble: 3 x 5
  Id    StrokeOnset         EventRecognition ArrivalTime
  <chr> <dttm>              <chr>            <dttm>
1 p1    2023-01-15 14:25:00 exact            2023-01-15 15:00:00
2 p2    2025-02-20 11:30:00 presumed         2025-02-20 17:00:00
3 p3    2019-06-05 13:00:00 exact            2019-06-05 14:00:00
# i 1 more variable: CtCompleted <dttm>
```

이렇게 해야 하는 뒤에서 설명할 날짜/시간 데이터에 대한 연산을 정확하게 할 수 있다.

18.3 날짜/시간 정보를 모으기

날짜/시간 정보가 다음과 같이 수집되어 있다고 생각해 보자. 어떤 치료를 시작한 시점을 말한다.

```
tx_time <- tibble::tribble(
    ~Id,  ~TxYear, ~TxMonth, ~TxDay, ~TxHour, ~TxMinute,
    "p3", 2025L,   1L,       21L,    9L,      24L,
    "p4", 2024L,   3L,       30L,    4L,      23L,
    "p5", 2023L,   4L,       23L,    15L,     46L
)
tx_time
```

```
# A tibble: 3 x 6
```

```
  Id    TxYear TxMonth TxDay TxHour TxMinute
  <chr>  <int>   <int> <int>  <int>    <int>
1 p3      2025       1    21      9       24
2 p4      2024       3    30      4       23
3 p5      2023       4    23     15       46
```

이런 데이터가 주어졌을 때 이것을 "숫자" 그대로 사용하는 것은 나중 계산에 오류가 발생할 수 있다. 날짜/시간 데이터 타입으로 변경해 주어야 한다. 이 경우는 `lubridate` 패키지의 `make_date()` 또는 `make_datetime()` 함수를 사용한다.

```
make_date(2025, 1, 21)
```

```
[1] "2025-01-21"
```

```
tx_time %>%
    mutate(
        TxDate = make_date(TxYear, TxMonth, TxDay),
        TxDateTime = make_datetime(TxYear, TxMonth, TxDay, TxHour, TxMinute)
    )
```

```
# A tibble: 3 x 8
  Id    TxYear TxMonth TxDay TxHour TxMinute TxDate     TxDateTime
  <chr>  <int>   <int> <int>  <int>    <int> <date>     <dttm>
1 p3      2025       1    21      9       24 2025-01-21 2025-01-21 09:24:00
2 p4      2024       3    30      4       23 2024-03-30 2024-03-30 04:23:00
3 p5      2023       4    23     15       46 2023-04-23 2023-04-23 15:46:00
```

18.4 날짜/시간 데이터에서 정보 추출

날짜/시간 데이터에서 연도, 월, 일, 시간, 분, 초, 요일, 분기 등 정보를 추출하는 함수들이 있다.

- `year()`, `month()`, `day()`: 연, 월, 일
- `hour()`, `minute()`, `second()`: 시, 분, 초
- `tz()` : 시간대

- wday() : 요일
- quarter() : 분기
- semester() : 반기

```
new_df
```

```
# A tibble: 3 x 5
  Id    StrokeOnset         EventRecognition ArrivalTime
  <chr> <dttm>              <chr>            <dttm>
1 p1    2023-01-15 14:25:00 exact            2023-01-15 15:00:00
2 p2    2025-02-20 11:30:00 presumed         2025-02-20 17:00:00
3 p3    2019-06-05 13:00:00 exact            2019-06-05 14:00:00
# i 1 more variable: CtCompleted <dttm>
```

```
year(new_df$StrokeOnset)
```

```
[1] 2023 2025 2019
```

```
month(new_df$StrokeOnset)
```

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

```
day(new_df$StrokeOnset)
```

```
[1] 15 20  5
```

```
hour(new_df$StrokeOnset)
```

```
[1] 14 11 13
```

```
minute(new_df$StrokeOnset)
```

```
[1] 25 30  0
```

```
second(new_df$StrokeOnset)
```

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

```
wday(new_df$StrokeOnset) # 요일
```

```
[1] 1 5 4
```

```
quarter(new_df$StrokeOnset) # 분기
```

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

```
semester(new_df$StrokeOnset) # 반기
```

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

`wday()` 함수는 요일을 숫자로 반환하는 데, `label = TRUE` 인자를 추가하면 요일을 문자열로 반환해 준다. 윈도우(Windows) 환경에서는 한글로 출력할 것이다.

```
wday(new_df$StrokeOnset, label = TRUE) # 요일
```

```
[1] Sun Thu Wed
Levels: Sun < Mon < Tue < Wed < Thu < Fri < Sat
```

이 함수의 도움말을 보면 `locale` 인자가 있는데, 디폴트 값인 `Sys.getlocale("LC_TIME")` 값을 사용한다. 이 값은 운영체제에 따라 다르다. 현재 R 세션에서 로캘을 보려면 `Sys.getlocale("LC_TIME")` 함수를 사용하고, `Sys.setlocale("LC_TIME", "ko_KR.UTF-8")` 함수를 사용하여 로캘을 변경할 수 있다. ㄴ

```
Sys.setlocale("LC_TIME", "ko_KR.UTF-8")
```

```
[1] "ko_KR.UTF-8"
```

```
Sys.getlocale("LC_TIME")
```

```
[1] "ko_KR.UTF-8"
```

```
wday(new_df$StrokeOnset, label = TRUE) # 요일
```

```
[1] 일 목 수
Levels: 일 < 월 < 화 < 수 < 목 < 금 < 토
```

18.5 날짜/시간 데이터 수정

앞에서 정보를 추출하는 함수를 할당 좌변에 둬서 날짜/시간 데이터를 수정할 수 있다.

```
the_date <- ymd_hms("2025-05-17 14:30:45", tz = "Asia/Seoul")
the_date
```

```
[1] "2025-05-17 14:30:45 KST"
```

연도가 2025가 아니라 2024로 바꾸려면 다음과 같이 실행한다.

```
year(the_date) <- 2024
the_date
```

```
[1] "2024-05-17 14:30:45 KST"
```

만찬가지로 5월이 아니라 6월로 바꾸려면 다음과 같이 실행한다.

```
month(the_date) <- 6
the_date
```

```
[1] "2024-06-17 14:30:45 KST"
```

다른 값들도 마찬가지이다. `update()` 함수를 사용하여 여러 값을 한번에 수정할 수 있다.

```
update(the_date, year = 2023, month = 7, hour = 15)
```

```
[1] "2023-07-17 15:30:45 KST"
```

18.6 기간 연산

lubridate 패키지는 기간을 목적에 따라 3가지 클래스(class)로 제공한다.

- Duration : 시간 간격을 초 단위로 표현하는 클래스
- Interval : 시작 시점과 끝 시점을 표현하는 클래스

- Period : 사람들이 흔히 표현하는 기간

18.6.1 Duration 클래스

Duration 클래스는 시간 간격을 **초 단위로 표현하는 클래스**이다. 다음과 같은 함수들을 사용한 Duration 클래스를 만든다.

- duration() 함수
- dseconds(), dminutes(), dhours(), ddays(), dweeks(), dmonths(), dyears() 함수

다음 예와 같이 duraiton() 함수에 단위를 추가하여 만들 수도 있고, d로 시작되는 함수를 사용해도 같은 의미의 객체를 만들 수 있다. 어찌되었든 결과는 초 단위로 표현되다는 점을 주목한다.

```
duration(10) # 10초
## [1] "10s"
dseconds(10) # 10초
## [1] "10s"
duration(10, "minutes") # 10분을 초 단위로 표현
## [1] "600s (~10 minutes)"
dminutes(10) # 10분을 초 단위로 표현
## [1] "600s (~10 minutes)"
duration(0.5, "hours") # 0.5시간을 초 단위로 표현
## [1] "1800s (~30 minutes)"
```

```r
dhours(0.5) # 0.5시간을 초 단위로 표현
## [1] "1800s (~30 minutes)"
duration(1, "days") # 1일을 초 단위로 표현
## [1] "86400s (~1 days)"
ddays(1) # 1일을 초 단위로 표현
## [1] "86400s (~1 days)"
duration(0.3, "weeks") # 0.3주를 초 단위로 표현
## [1] "181440s (~2.1 days)"
duration(5, "months") # 5개월을 초 단위로 표현
## [1] "13149000s (~21.74 weeks)"
dmonths(5) # 5개월을 초 단위로 표현
## [1] "13149000s (~21.74 weeks)"
duration(1, "years") # 1년을 초 단위로 표현
## [1] "31557600s (~1 years)"
dyears(1) # 1년을 초 단위로 표현
## [1] "31557600s (~1 years)"
```

위에서 본 `new_df` 데이터프레임에서 `StrokeOnset` 열과 `ArrivalTime` 열의 차이를 계산해 보자.

```r
new_df
```

```
# A tibble: 3 x 5
  Id    StrokeOnset         EventRecognition ArrivalTime
  <chr> <dttm>              <chr>            <dttm>
1 p1    2023-01-15 14:25:00 exact            2023-01-15 15:00:00
2 p2    2025-02-20 11:30:00 presumed         2025-02-20 17:00:00
3 p3    2019-06-05 13:00:00 exact            2019-06-05 14:00:00
# i 1 more variable: CtCompleted <dttm>
```

날짜를 날짜를 빼면 timediff 클래스가 된다.

```r
new_df$ArrivalTime - new_df$StrokeOnset
```

```
Time differences in mins
[1]  35 330  60
```

이것을 as.druation() 함수를 사용하여 Duration 클래스로 변환할 수 있다.

```
as.duration(new_df$ArrivalTime - new_df$StrokeOnset)
```

```
[1] "2100s (~35 minutes)" "19800s (~5.5 hours)" "3600s (~1 hours)"
```

이것은 모두 초 단위의 기간이다. 분 단위로 보려면 분 단위로 나누면 된다.

```
as.duration(new_df$ArrivalTime - new_df$StrokeOnset) / dminutes(1)
```

```
[1]  35 330  60
```

이것은 분 단위의 기간이다.

분 단위의 평균 시간을 계산해 보자.

```
mean(as.duration(new_df$ArrivalTime - new_df$StrokeOnset) / dminutes(1))
```

```
[1] 141.6667
```

2025-05-17 14:30:45 시간에서 1시간 50분 후의 시간을 계산해 보자.

```
ymd_hms("2025-05-17 14:30:45", tz = "Asia/Seoul") + dhours(1) + dminutes(50)
```

```
[1] "2025-05-17 16:20:45 KST"
```

18.6.2 Interval 클래스

Interval 클래스는 시작 시점과 끝 시점을 표현하는 클래스이다. 다음과 같은 함수들을 사용한 Interval 클래스를 만든다. Interval 클래스를 설명하기 위해 다음가 같은 페이크 데이터프레임을 사용한다.

```
admission_discharge <- tibble::tribble(
    ~Id, ~admission_date, ~discharge_date,
    "p34", "2025/01/05", "2025/01/10",
    "p64", "2024/12/25", "2025/01/07",
```

```
    "p87", "2024/11/25", "2025/03/01",
    "p79", "2025/01/06", "2025/03/15",
    "p99", "2025/02/05", "2025/3/20",
    "p100", "2025/02/15", "2025/03/25"
)
admission_discharge
```

```
# A tibble: 6 x 3
  Id    admission_date discharge_date
  <chr> <chr>          <chr>
1 p34   2025/01/05     2025/01/10
2 p64   2024/12/25     2025/01/07
3 p87   2024/11/25     2025/03/01
4 p79   2025/01/06     2025/03/15
5 p99   2025/02/05     2025/3/20
6 p100  2025/02/15     2025/03/25
```

먼저 날짜/시간 데이터로 변환한다.

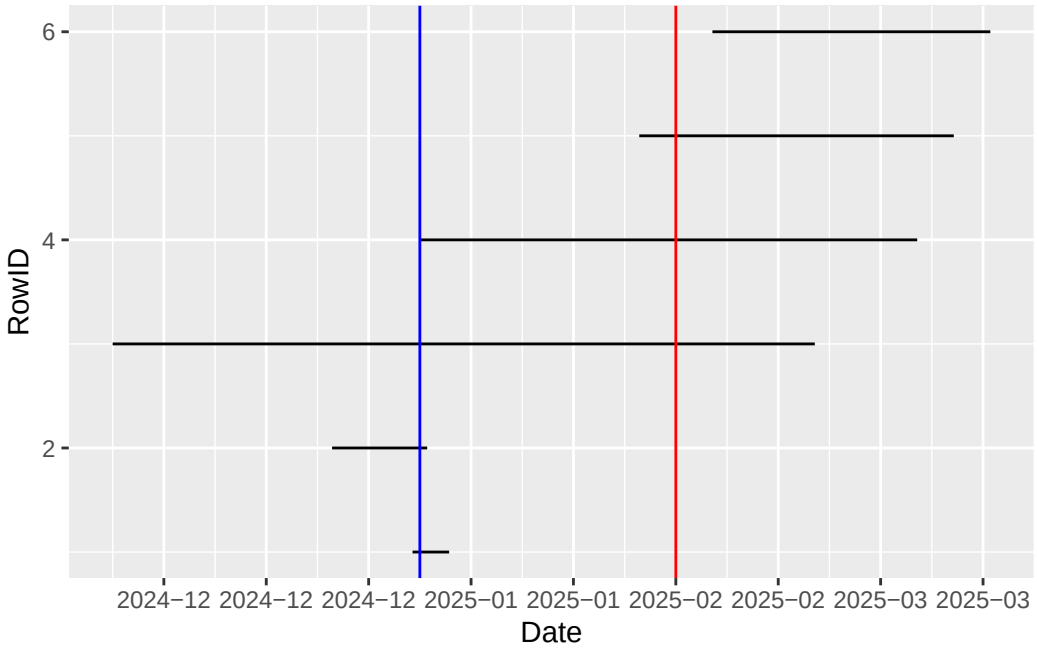
```
ad_df <- admission_discharge %>%
    mutate(
        admission_date = ymd(admission_date),
        discharge_date = ymd(discharge_date)
    )
ad_df
```

```
# A tibble: 6 x 3
  Id    admission_date discharge_date
  <chr> <date>         <date>
1 p34   2025-01-05     2025-01-10
2 p64   2024-12-25     2025-01-07
3 p87   2024-11-25     2025-03-01
4 p79   2025-01-06     2025-03-15
5 p99   2025-02-05     2025-03-20
```

```
6 p100   2025-02-15      2025-03-25
```

이 데이터를 가지고 환자의 입원 기간을 ggplot2 패키지에 있는 geom_segment 함수를 사용하여 표현해 보자.

```
library(ggplot2)

df <- ad_df %>%
    mutate(Y = row_number())

ggplot(df) +
    geom_segment(aes(x = admission_date, y = Y, xend = discharge_date, yend = Y)) +
    geom_vline(xintercept = ymd("2025-01-06"), color = "blue") +
    geom_vline(xintercept = ymd("2025-02-10"), color = "red") +
    scale_x_date(date_breaks = "2 weeks", date_labels = "%Y-%m") +
    labs(x = "Date", y = "RowID")
```



이 데이터프레임에서 2025년 1월 6일과 2025년 2월 10일 입원 환자가 어떻게 되는지 보려고 한다. 이 문제는 위 그래프에서 파란선과 빨간선이 교차하는 지점의 개수를 세는 것과 동일하다.

만약 위 데이터의 Y축이 한 환자가 복용하는 약물들의 ID라고 생각해 보자. 각 약물의 복용 기간을 알고

있다면 우리는 어느 시점에서 어떤 약물들을 복용했는지(몇 개을 복용했는지) 알 수 있을 것이다. 이런 문제를 해결하기 위해서 Interval 클래스를 사용한다.

각 환자의 입원 기간을 표현하는 `Interval` 클래스를 만든다. `inteval(시작일, 마지막일)` 함수를 사용하여 만든다.

```
ad_df %>%
    mutate(admission_interval = interval(admission_date, discharge_date))
```

```
# A tibble: 6 x 4
  Id    admission_date discharge_date admission_interval
  <chr> <date>         <date>         <Interval>
1 p34   2025-01-05     2025-01-10     2025-01-05 UTC--2025-01-10 UTC
2 p64   2024-12-25     2025-01-07     2024-12-25 UTC--2025-01-07 UTC
3 p87   2024-11-25     2025-03-01     2024-11-25 UTC--2025-03-01 UTC
4 p79   2025-01-06     2025-03-15     2025-01-06 UTC--2025-03-15 UTC
5 p99   2025-02-05     2025-03-20     2025-02-05 UTC--2025-03-20 UTC
6 p100  2025-02-15     2025-03-25     2025-02-15 UTC--2025-03-25 UTC
```

`admission_interval` 열은 `Interval` 클래스이며, 각 환자의 입원일과 퇴원일 정보를 포함한다. 이런 객체들 사이에는 `int_overlaps()` 함수를 사용하여 두 기간이 겹치는지 확인할 수 있고, `%within%` 연산자를 사용하여 특정 날짜/시간 또는 interval 객체가 다른 interval 객체에 포함되는지 확인할 수 있다. 여기서는 이 연산자를 사용하려고 한다.

```
# 2025년 1월 6일 입원 환자 수
ad_df %>%
    mutate(admission_interval = interval(admission_date, discharge_date)) %>%
    filter(ymd("2025-01-06") %within% admission_interval) %>%
    count()
```

```
# A tibble: 1 x 1
      n
  <int>
1     4
```

```
# 2025년 1월 8일 입원 환자 수
ad_df %>%
    mutate(admission_interval = interval(admission_date, discharge_date)) %>%
    filter(ymd("2025-02-10") %within% admission_interval) %>%
    count()
```

```
# A tibble: 1 x 1
      n
  <int>
1     3
```

Interval 객체가 시작 시점과 끝 시점을 표현하는 클래스이기 때문에 두 객체의 차이(초 단위)를 계산할 수 있다.

```
ad_df %>%
    mutate(admission_interval = interval(admission_date, discharge_date)) %>%
    mutate(admission_interval_length = int_length(admission_interval)) %>%
    select(Id, admission_interval_length)
```

```
# A tibble: 6 x 2
  Id    admission_interval_length
  <chr>                     <dbl>
1 p34                      432000
2 p64                     1123200
3 p87                     8294400
4 p79                     5875200
5 p99                     3715200
6 p100                    3283200
```

평균 재원일(일 단위)을 계산해 보자. Duration 클래스를 `as.numeric()` 함수를 사용하여 숫자로 변환하는 과정을 거쳤다.

```
ad_df %>%
    mutate(admission_interval = interval(admission_date, discharge_date)) %>%
```

```
    mutate(admission_interval_length = int_length(admission_interval)) %>%
    select(Id, admission_interval_length) %>%
    summarise(StayDays = mean(admission_interval_length / as.numeric(ddays(1))))
```

```
# A tibble: 1 x 1
  StayDays
     <dbl>
1     43.8
```

18.6.3 Period 클래스

Period 클래스는 사람들이 흔히 표현하는 기간을 표현하는 클래스이다. 예를 들어 윤년이냐 아니냐에 상관없이 2월 14일 1년후 날짜를 계산할 수 있다. 보통 4로 나누어 떨어지는 해를 윤년이라고 하는데, 이 경우에는 2월 29일이 있는 해이다.

- `years()`, `months()`, `weeks()`, `days()`, `hours()`, `minutes()`, `seconds()` 함수를 사용하여 Period 클래스를 만든다.

```
# 1년
leap_year(2024)
```

```
[1] TRUE
```

```
leap_year(2025)
```

```
[1] FALSE
```

```
dd <- c(ymd("2024-02-14"), ymd("2025-02-14"))
dd + years(1) # 1년후
```

```
[1] "2025-02-14" "2026-02-14"
```

```
# 1개월
dd + months(1) # 1개월 후
```

```
[1] "2024-03-14" "2025-03-14"
```

18.7 기타 날짜 등 계산

18.7.1 분기/계절/월의 첫 날 계산하기

`floor_date()` 함수는 날짜를 지정된 단위("quarter", "season", "semester" 등)로 내림하여 해당 기간의 첫 날을 반환한다. 이 함수는 다음과 같은 단위를 지원한다.

- "year": 연초
- "quarter": 분기 초
- "month": 월초
- "week": 주초
- "day": 일초
- "season": 계절 초
- "halfyear": 반기 초

```
# 분기 초
dates <- ymd(c("2024-01-15", "2024-05-20", "2024-08-10", "2024-12-25"))
floor_date(dates, "quarter")
```

```
[1] "2024-01-01" "2024-04-01" "2024-07-01" "2024-10-01"
```

```
# 계절 초
floor_date(dates, "season")
```

```
[1] "2023-12-01" "2024-03-01" "2024-06-01" "2024-12-01"
```

```
floor_date(dates, "halfyear")
```

```
[1] "2024-01-01" "2024-01-01" "2024-07-01" "2024-07-01"
```

```
floor_date(dates, "month")
```

```
[1] "2024-01-01" "2024-05-01" "2024-08-01" "2024-12-01"
```

18.7.2 연속된 날짜 데이터 생성하기

어떤 경우는 일정한 간격으로 날짜 데이터를 생성해야 할 필요가 있다. 예를 들어 2025년 1월 1일부터 2025년 12월 31일까지 주 단위로 날짜 데이터를 생성해야 할 필요가 있다. 이런 경우에는 `seq()` 함수를 사용하여 생성할 수 있다.

```
seq(ymd("2025-01-01"), ymd("2025-12-31"), by = "week")
```

```
 [1] "2025-01-01" "2025-01-08" "2025-01-15" "2025-01-22" "2025-01-29"
 [6] "2025-02-05" "2025-02-12" "2025-02-19" "2025-02-26" "2025-03-05"
[11] "2025-03-12" "2025-03-19" "2025-03-26" "2025-04-02" "2025-04-09"
[16] "2025-04-16" "2025-04-23" "2025-04-30" "2025-05-07" "2025-05-14"
[21] "2025-05-21" "2025-05-28" "2025-06-04" "2025-06-11" "2025-06-18"
[26] "2025-06-25" "2025-07-02" "2025-07-09" "2025-07-16" "2025-07-23"
[31] "2025-07-30" "2025-08-06" "2025-08-13" "2025-08-20" "2025-08-27"
[36] "2025-09-03" "2025-09-10" "2025-09-17" "2025-09-24" "2025-10-01"
[41] "2025-10-08" "2025-10-15" "2025-10-22" "2025-10-29" "2025-11-05"
[46] "2025-11-12" "2025-11-19" "2025-11-26" "2025-12-03" "2025-12-10"
[51] "2025-12-17" "2025-12-24" "2025-12-31"
```

3주 단위로 날짜 데이터를 생성해 보자.

```
seq(ymd("2025-01-01"), ymd("2025-12-31"), by = "3 weeks")
```

```
 [1] "2025-01-01" "2025-01-22" "2025-02-12" "2025-03-05" "2025-03-26"
 [6] "2025-04-16" "2025-05-07" "2025-05-28" "2025-06-18" "2025-07-09"
[11] "2025-07-30" "2025-08-20" "2025-09-10" "2025-10-01" "2025-10-22"
[16] "2025-11-12" "2025-12-03" "2025-12-24"
```

1개월 단위로 날짜 데이터를 생성해 보자.

```
seq(ymd("2025-01-01"), ymd("2025-12-31"), by = "month")
```

```
 [1] "2025-01-01" "2025-02-01" "2025-03-01" "2025-04-01" "2025-05-01"
 [6] "2025-06-01" "2025-07-01" "2025-08-01" "2025-09-01" "2025-10-01"
[11] "2025-11-01" "2025-12-01"
```

18.8 정리

이 장에서는 날짜/시간 데이터를 처리하는 방법을 설명했다.

19 귀찮은 것을 해결해 주는 janitor 패키지 (정리중)

Part III

III. 통계 데이터 시각화: ggplot2

20 ggplot2 통계 그래픽의 기초

ggplot2 패키지의 gg는 grammer of graphics의 약자로, 그래프의 문법을 말한다. 이 패키지는 통계학자 Leland Wilkinson가 "The Grammar of Graphics"라는 책으로 개념을 정리한 **통계 그래픽의 문법**에 바탕을 두고, 해들리 위캄(Hadley Wickham) 등이 R 언어로 구현한 것이다.

그래서 ggplot2 패키지를 사용하여 그래프를 만들기 위해서는 먼저 그 문법을 이해해야 한다.

20.1 Grammer of Graphics(그래프 문법)

ggplot2의 핵심 개념을 요약하면 다음과 같다.

- **매핑(mapping)**
  - 데이터셋의 변수를 그래프의 시각적 요소에 매핑한다.
    - 예를 들어 막대 그래프는 (범주형) 데이터의 개수를 카운트하여 그 수를 막대의 높이로 표시한다.
  - ggplot2의 데이터셋은 **tidy 데이터프레임**을 사용한다.
    - 따라서 필요한 경우 데이터프레임으로 변환시킬 필요가 있다.
    - Untidy 데이터프레임은 tidy 데이터프레임으로 만들고 시작한다.
  - 시각적 요소를 **aesthetics**라고 한다.
    - 예를 들어 다음과 같은 aesthetics가 있다.
      - 위치(position)은 position aesthetics라고 하고, 실제로 x, y aesthetics가 있다. 이것은 데이터를 위치라는 시각적 요소로 변환한다.
      - 색상(color)는 color aesthetics라고 한다. 이것은 데이터를 색상이라는 시각적 요소로 변환한다.
      - 형태(shape)는 shape aesthetics라고 한다. 이것은 데이터를 형태라는 시각적 요소로 변환한다.

        - 크기(size)는 `size aesthetics`라고 한다. 이것은 데이터를 크기라는 시각적 요소로 변환한다.
    - 매핑은 `aes()` 함수 안에서 지정한다. `aes(x = wt, y = mpg, color = factor(cyl))`이라고 하면 이것은 `wt` 변수를 `x`축에, `mpg` 변수를 `y`축에, `cyl` 변수를 `color` aesthetics에 매핑한다는 뜻이다.

- **셋팅(setting)**
    - 셋팅은 데이터에 따라 매핑하는 대신 사용자가 임의로 지정하는 것을 말한다.
        * 예를 들어 막대 그래프의 색상은 데이터에 따라 결정되지 않고 사용자가 임의로 지정할 수 있다. 그래서 `setting`이라고 한다.
        * 셋팅은 `aes()` 함수 밖에서 지정한다.
        * 예를 들어 `geom_bar(color = "steelblue")`는 막대의 색상을 파란색으로 지정한다.

- **지옴(geoms)**과 **통계 변환(stat, statistical transformation)**
    - 지옴(geoms)은 시각적 요소를 말하고, 통계 변환(statistic transformation)은 데이터를 통계적으로 변환하는 것을 말한다.
    - 지옴은 `geom_*()` 함수로 만들고, 통계 변환은 `stat_*()` 함수로 만든다.
    - 이 둘은 밀접하게 연결되어 있어서, `geom_*()` 함수 안에서 `stat`이라는 인자에 통계 변환을 지정할 수 있고, `stat_*()` 함수 안에서 `geom`이라는 인자에 지옴을 지정할 수 있다.
        * 따라서, 유연하게 지옴을 정하고 나서 통계 변환 방법을 지정할 수 있고, 반대로 통계 변환 방법을 정하고 나서 지옴을 지정할 수 있다.
    - 예를 들어
        * `geom_bar()` 함수는 디폴트로 데이터를 카운트하여 막대의 높이를 결정한다. `geom_bar(stat = "count")`가 디폴트이다.
        * `stat_count()` 함수는 데이터를 카운트하여 막대의 높이를 결정한다. `stat_count(geom = "bar")`가 디폴트이다.

- **레이어를 쌓아서 그래프를 만든다(layer by layer).**
    - `ggplot2`는 레이어를 쌓아서 만들고, 각 레이어는 **+ 연산자**를 사용하여 추가한다.
    - 레이어를 명시적으로 정의하여 사용하는 대신, `geom_*()` 함수나 `stat_*()` 함수가 레이어를 생성시킨다(명시적으로 지정할 수도 있다).
    - 레이어는 독립적인 데이터, 매핑, 지옴, 통계 변환을 가질 수 있다.
    - 예를 들어 다음과 같은 레이어를 쌓아서 그래프를 만들 수 있다.
        * `ggplot(data = mpg) + geom_point(aes(x = cty, y = hwy)) + geom_smooth(aes(x = cty, y = hwy))`

* 이 코드는 점을 표시하는 레이어와 선을 표시하는 레이어를 쌓아서 그래프를 만든다.

- **스케일(sales)와 가이드(guide)**
  - 스케일(scale)은 데이터를 디자인 요소에 매핑할 때 사용되는 함수이다.
  - 가이드(guide)는 레전드(legend)와 축(axis)을 말하고, 스케일 함수의 역함수(reverse function)이다.
- **좌표계(coordinate system)**
  - 좌표계는 그래프에서 사용할 좌표계를 결정한다.
- **패시팅(facet)**
  - "small multiple"이라고도 하는데, 데이터를 작은 데이터셋으로 나누어 각각의 데이터셋에 대해 그래프를 만들고, 이것들을 모아서 배치하는 것을 말한다. 그룹별 차이 등을 확인하는 데 유용한다.
- **테마(theme)**
  - 테마는 그래프의 디자인을 결정한다.
  - 테마는 `theme()` 함수로 제어한다.
  - 이 함수 안에서 그래프의 주요 요소들을 제어할 수 있다.

처음에는 이런 내용들이 다소 추상적이어서 이해하기 어려울 수 있지만, 실제로 그래프를 만들어 보면 이해가 쉬워진다. 이제 실제로 그래프를 만들어 보면서 이 내용을 이해해 보자.

`ggplot2` 패키지를 로딩한다.

```
library(ggplot2)
```

20.2 ggplot2로 그래프 만들기

`ggplot2` 패키지로 그래프를 만들 때는 일반적으로 다음 문법을 따른다.

```
ggplot(data = <데이터>) +
    geom_<지옴타입>(aes(<매핑>)) +
    <다른 레이어>
```

먼저 그래프에 사용할 데이터프레임을 로딩한다. ggplot2는 tidy 데이터프레임을 사용한다. 따라서 필요한 경우 tidy 데이터프레임으로 변환시킬 필요가 생기기도 한다.

```
# 데이터 로딩
library(tidyverse)
stroke_df <- readRDS("./data/stroke_df.rds")
glimpse(stroke_df)
```

```
Rows: 5,110
Columns: 12
$ id                <chr> "9046", "51676", "31112", "60182", "1665", "56669", ~
$ gender            <fct> Male, Female, Male, Female, Female, Male, Male, Fema~
$ age               <dbl> 67, 61, 80, 49, 79, 81, 74, 69, 59, 78, 81, 61, 54, ~
$ hypertension      <fct> No, No, No, No, Yes, No, Yes, No, No, No, Yes, No, N~
$ heart_disease     <fct> Yes, No, Yes, No, No, No, Yes, No, No, No, No, Yes, ~
$ ever_married      <fct> Yes, Yes, Yes, Yes, Yes, Yes, Yes, No, Yes, Yes, Yes~
$ work_type         <fct> Private, Self-employed, Private, Private, Self-emplo~
$ residence_type    <fct> Urban, Rural, Rural, Urban, Rural, Urban, Rural, Urb~
$ avg_glucose_level <dbl> 228.69, 202.21, 105.92, 171.23, 174.12, 186.21, 70.0~
$ bmi               <dbl> 36.6, NA, 32.5, 34.4, 24.0, 29.0, 27.4, 22.8, NA, 24~
$ smoking_status    <fct> formerly smoked, never smoked, never smoked, smokes,~
$ stroke            <fct> Yes, Yes, Yes, Yes, Yes, Yes, Yes, Yes, Yes, Yes, Ye~
```

이 데이터셋을 ggplot() 함수에 전달하여 그래프를 만든다.

```
ggplot(stroke_df)
```

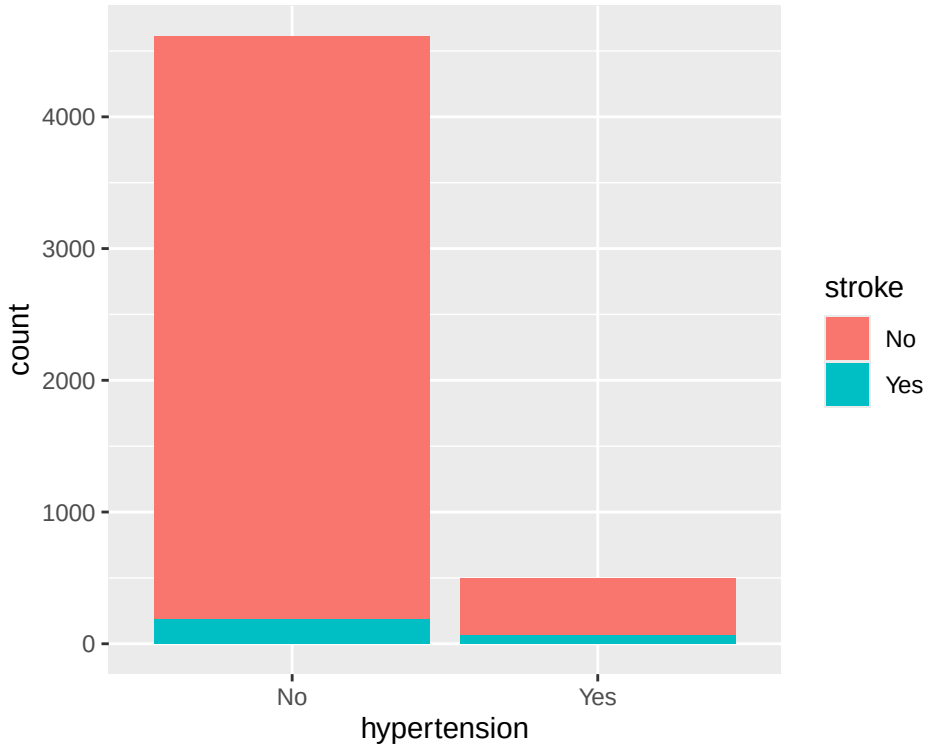
위에서 보는 것처럼 이 단계에는 빈 레이어 하나만 생성된다. 이렇게 그래프를 만들 데이터프레임이 준비되었다면, 그 다음 할 일은 매핑과 그런 매핑으로 어떤 시각적 요소를 만들지를 결정하는 것이다. 모든 통계 분석 과정에서 그러한 것처럼, 변수가 가지고 있는 값의 종류, 즉 데이터 타입에 따라 매핑을 결정되는 경우가 많다.

위 데이터프레임에서 범주형 변수인 `stroke`과 또 범주형 변수인 `hypertension`에 관한 그래프를 만들어 볼 계획이다. 이런 범주형 변수의 시각화는 기본적으로 카운트(count)에 기반하게 되고, 가장 흔히 사용되는 시각화 방법을 카운트된 값을 막대의 높이로 표시하는 막대 그래프(bar plot)이다. 이런 내용을 기반으로 마음 속에 또는 종이에 만들고자 하는 그래프를 먼저 그려보는 것이 좋다. 그 다음 코딩을 시작한다.

- `geom_bar()` 함수를 사용하고, `aes()` 함수를 사용하여 데이터를 매핑한다.
    - `x` aesthetic에 `hypertension` 변수를 매핑한다.
    - `fill` aesthetic에 `stroke` 변수를 매핑한다.
    - 이런 레이어를 정의하면서 + 연산자를 사용하여 기존 레이어에 추가한다.

아래 코드에서 보듯이 `ggplot2`에서 데이터프레임의 변수는 큰따옴표나 작은따옴표를 쓰지 않고 바로 이름으로 사용한다.

```
ggplot(stroke_df) +
    geom_bar(aes(x = hypertension, fill = stroke))
```



`geom_bar()` 함수는 `stat = "count"` 인자를 디폴트로 사용한다. `fill` aesthestic은 뇌졸중의 유무에 따라서 해당 부분의 막대의 색을 다르게 표현한다.

어떤 경우는 카운트나 비율(proportion)이 미리 계산된 데이터를 가지고 그래프 작업을 시작하는 경우도 있을 것이다.

```
df <- stroke_df %>%
    count(hypertension, stroke)
df
```

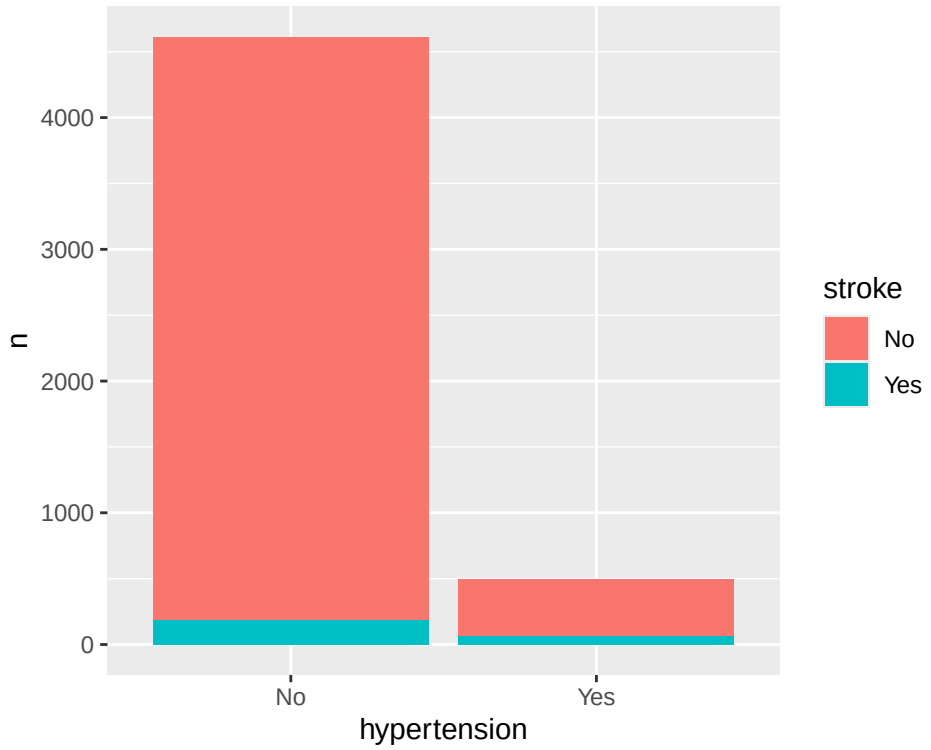
```
  hypertension stroke    n
1           No     No 4429
2           No    Yes  183
3          Yes     No  432
4          Yes    Yes   66
```

(통계적 변환(statistical transformation) 개념을 소개하려 한다)

이 경우에는 2가지 방법을 사용하여 그래프를 만들 수 있다.

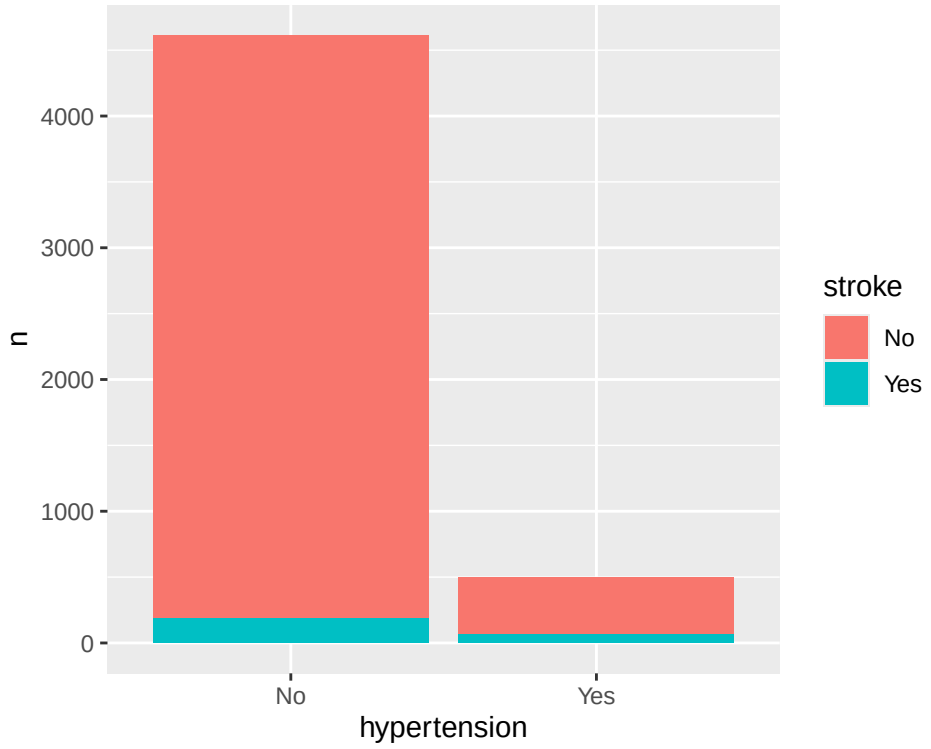
- 이미 카운트가 되어 있기 때문에 `stat = "count"` 인자를 사용하지 않고, `geom_bar()` 함수에 `stat = "identity"` 인자를 사용한다. 그러면 항등함수(값을 받아서 그대로 그 값을 출력하는 함수)가 사용된다.

```
ggplot(df) +
    geom_bar(aes(hypertension, n, fill = stroke), stat = "identity")
```



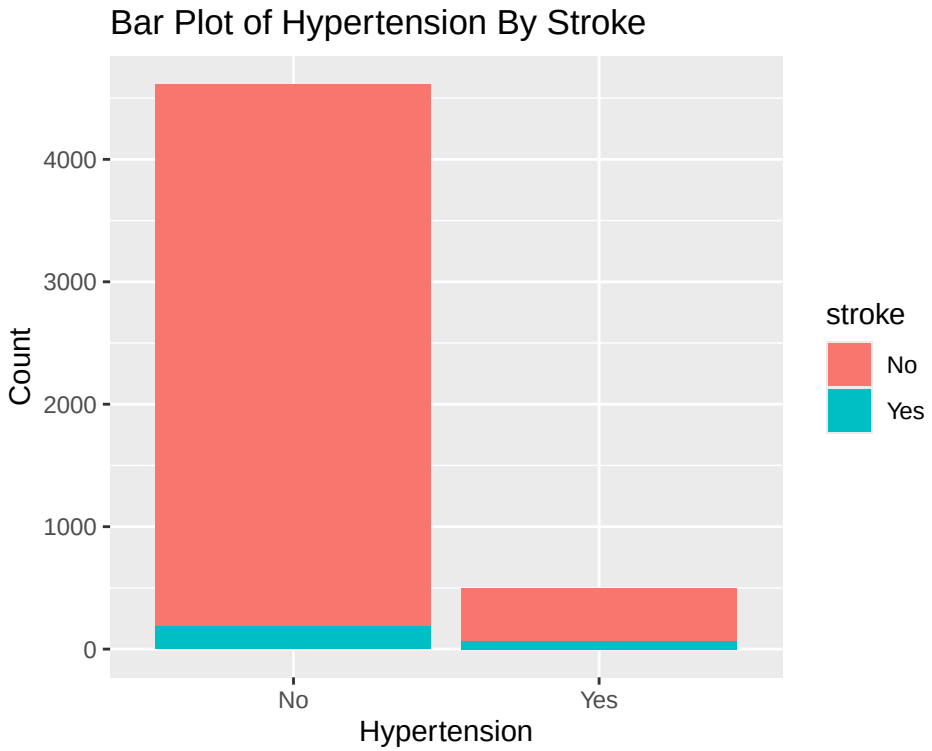
- 두 번째 `geom_col()` 함수를 사용하는 것이다. 이 함수는 데이터를 카운트하는 것이 아니라, 데이터를 그대로 표시한다. 이 함수의 `stat` 인자는 디폴트로 `stat = "identity"`이다.

```
ggplot(df) +
    geom_col(aes(hypertension, n, fill = stroke))
```



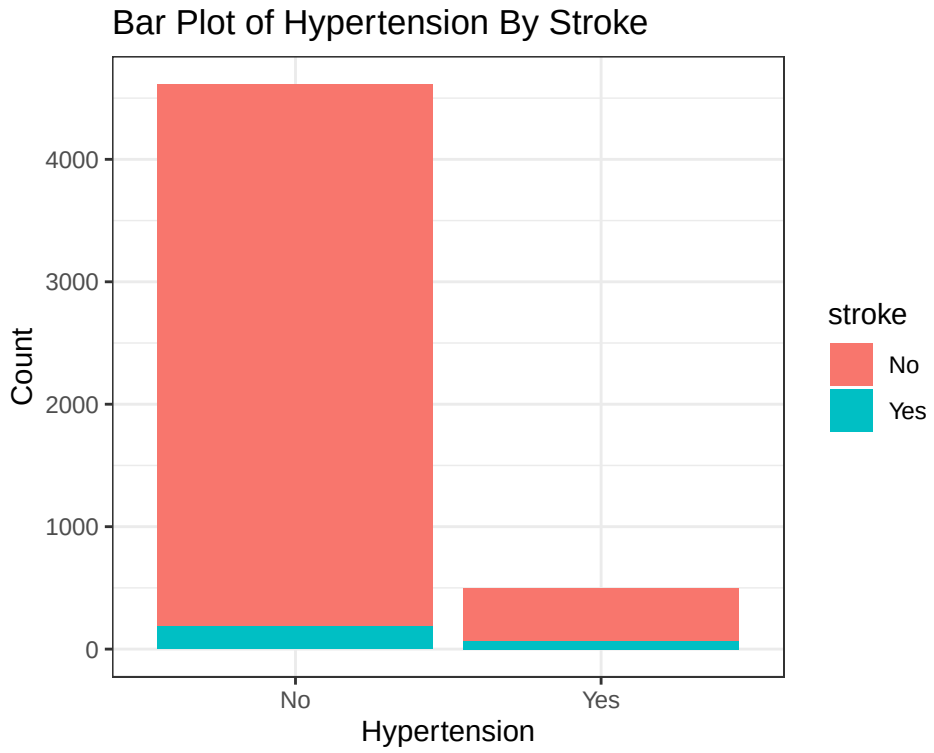
이제 여기에 레이블(labels)을 추가해 보자. 레이블 추가하는 `labs()` 함수를 주로 사용한다. 이것 역시도 하나의 레이어를 만든다. 이 함수 안에서 `title`, `x`, `y` 인자를 사용하여 레이블을 지정한다.

```
ggplot(stroke_df) +
    geom_bar(aes(x = hypertension, fill = stroke)) +
    labs(
        title = "Bar Plot of Hypertension By Stroke",
        x = "Hypertension",
        y = "Count"
    )
```



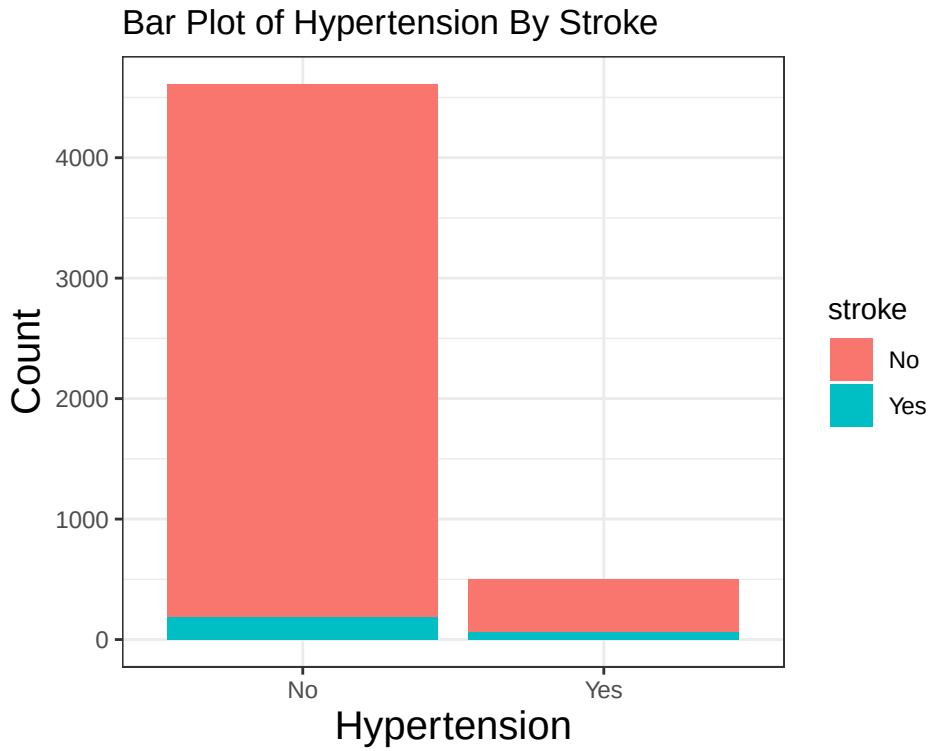
다음은 테마(theme)을 적용해 보자. 이것 역시도 하나의 레이러이다. `theme_bw()`과 같은 함수는 이미 정해진 테마를 지정할 때 사용한다.

```
ggplot(stroke_df) +
    geom_bar(aes(x = hypertension, fill = stroke)) +
    labs(
        title = "Bar Plot of Hypertension By Stroke",
        x = "Hypertension",
        y = "Count"
    ) +
    theme_bw()
```



이렇게 생성된 그래프에서 `theme()` 함수를 사용하여 개별 요소들을 하나씩 자신의 취향에 맞게 조정할 수 있다.

```
ggplot(stroke_df) +
    geom_bar(aes(hypertension, fill = stroke)) +
    labs(
        title = "Bar Plot of Hypertension By Stroke",
        x = "Hypertension",
        y = "Count"
    ) +
    theme_bw() +
    theme(
        axis.title = element_text(size = 15)
    )
```



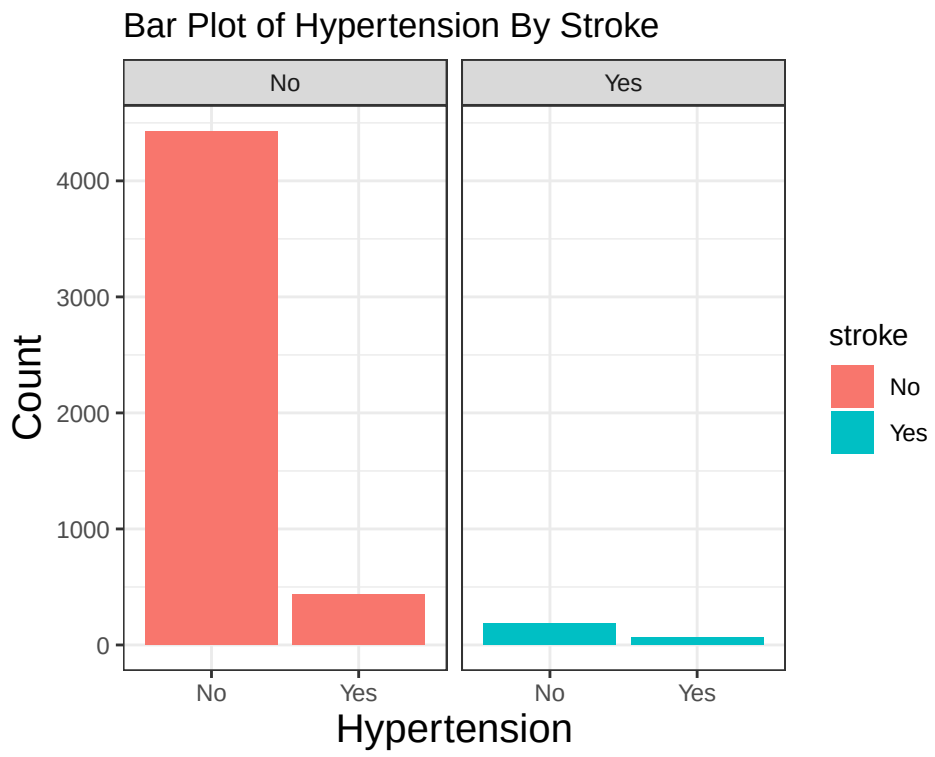
여기까지 진행하여 어느 정도 원하는 그래프를 얻었다. 이 과정에서 우리는 데이터를 매핑하고, 지옴을 추가하고, 레이블을 추가하고, 테마를 적용하는 등의 작업을 하였다. 이 과정에서 우리는 데이터와 그래프의 개별 요소들을 하나씩 조정하여 원하는 그래프를 만들었다.

지금까지 우리는 스케일(scales)을 명시적으로 다루지는 않았다. 그렇지만 하나의 매핑에 대응하는 디폴트 스케일들이 사용되고 있다. 예를 들어 `x = hypertension`에 대응하는 스케일은 범주형 변수이므로 범주형 스케일이 사용되고 있는데, `scale_x_discrete()` 함수를 사용하여 스케일을 직접 조정해 볼 수 있다. 그러나 이 부분은 뒤에서 더 자세하게 설명하고자 한다.

이제 패시팅(faceting)만 추가로 적용해 보자. 이것은 데이터를 작은 데이터셋으로 나누어 각각의 데이터셋에 대해 그래프를 만들고, 이것들을 모아서 배치하는 것을 말한다. 그룹별 차이 등을 확인하는 데 유용하다.

```
ggplot(stroke_df) +
    geom_bar(aes(hypertension, fill = stroke)) +
    labs(
        title = "Bar Plot of Hypertension By Stroke",
        x = "Hypertension",
```

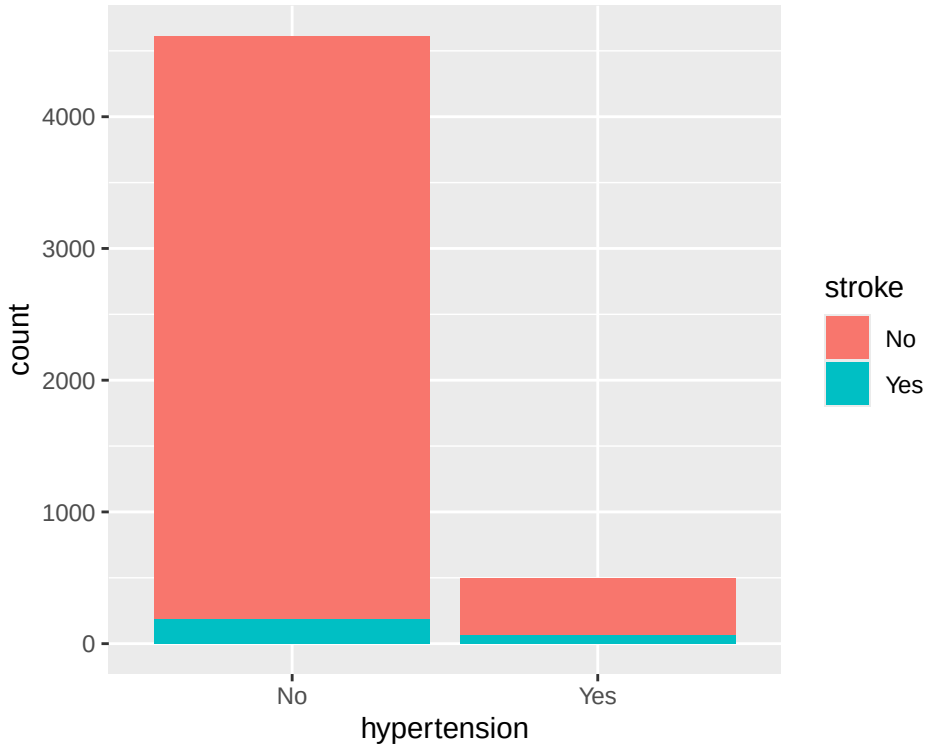
```
    y = "Count"
) +
theme_bw() +
theme(
    axis.title = element_text(size = 15)
) +
facet_wrap(~stroke)
```



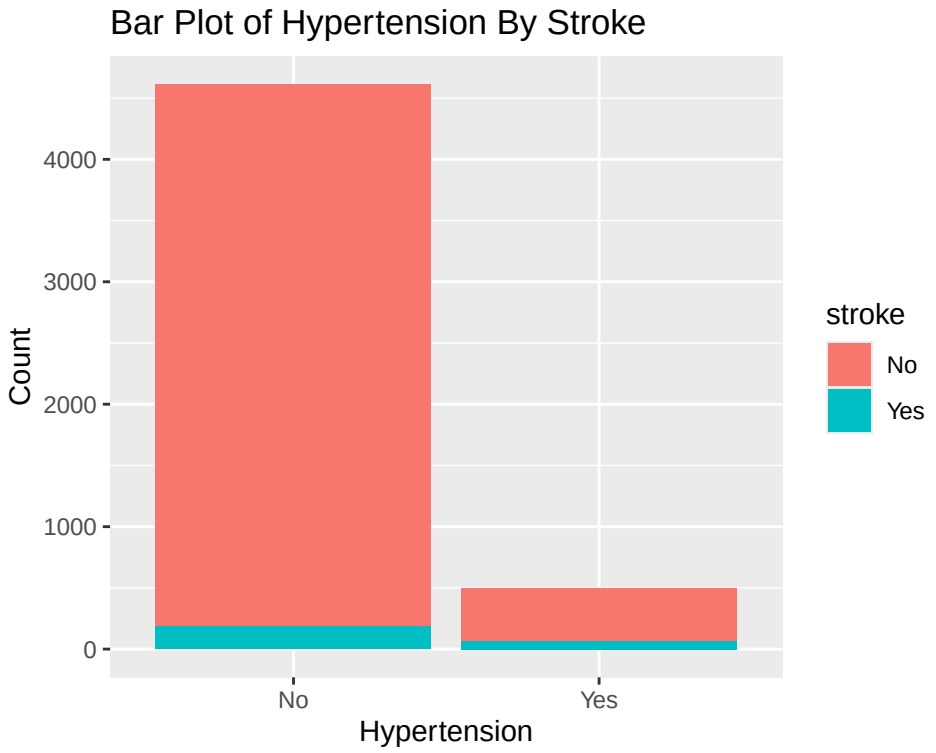
20.3 ggplot2 패키지를 사용할 때 꼭 기억하고 있어야 하는 것들

- ggplot2는 레이어를 + 연산자를 사용하여 추가하고, + 연산자로 추가된 레이어는 일반적인 R 데이터와 같이 변수에 할당할 수 있다. 이렇게 할당된 변수는 일반적인 R 객체처럼 그 이름으로 출력할 수 있다.
- ggplot2로 코딩할 때는 변수의 이름에 따옴표를 붙이지 않고, 그 이름 그대로 사용한다(비표준 평가).

```r
# 변수에 할당
p <- ggplot(stroke_df) +
    geom_bar(aes(hypertension, fill = stroke))
p
```

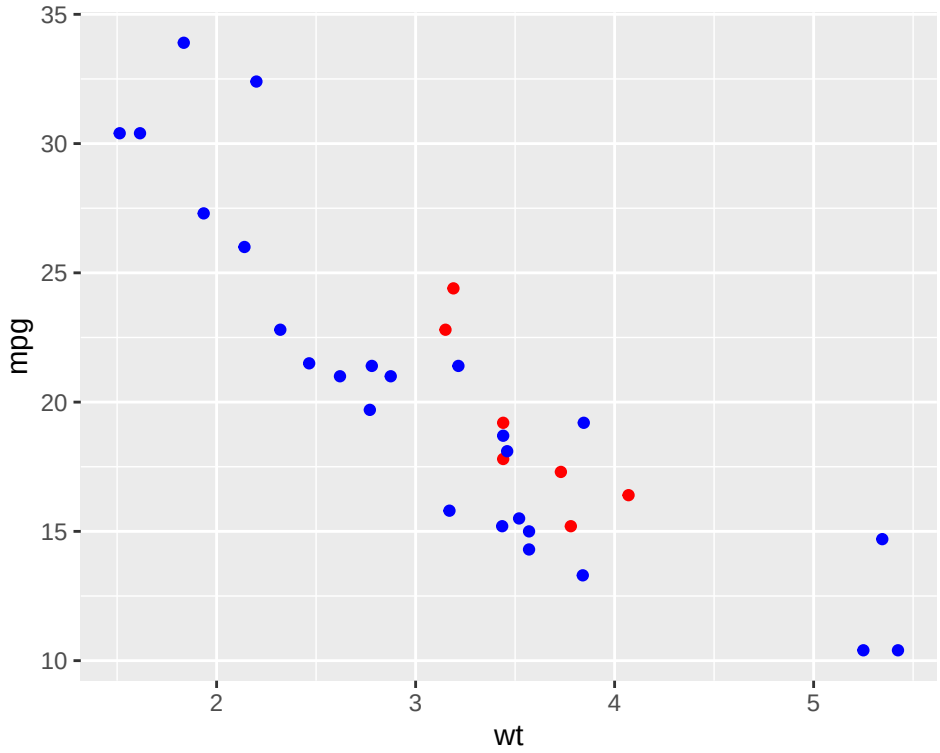



```r
# 변수를 업데이트
p <- p +
    labs(
        title = "Bar Plot of Hypertension By Stroke",
        x = "Hypertension",
        y = "Count"
    )
# 마지막 변수를 출력
p
```



- ggplot2의 레이어는 독립적으로(independently) 존재하여, 자체의 데이터, 매핑, 지옴, 통계 변환을 가질 수 있다. 따라서 다른 데이터를 종합하여 하나의 그래프를 만들 수 있다.

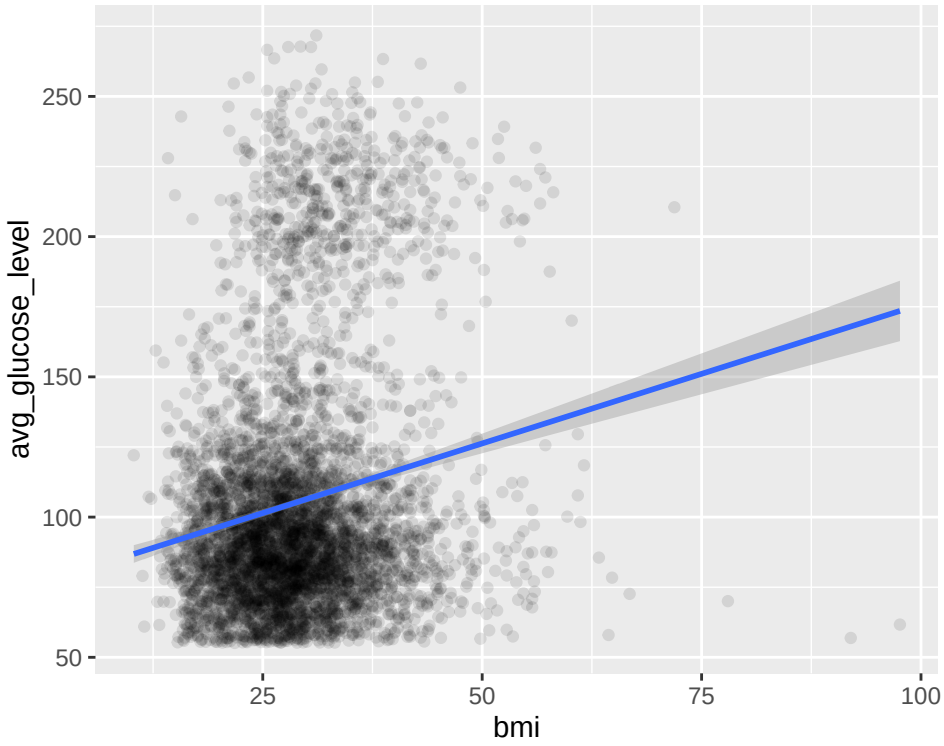
```
library(stringr)

ggplot(mtcars, aes(x = wt, y = mpg)) +
    geom_point(
        data = mtcars[str_detect(rownames(mtcars), "Merc"), ],
        aes(x = wt, y = mpg),
        color = "red"
    ) +
    geom_point(
        data = mtcars[!str_detect(rownames(mtcars), "Merc"), ],
        aes(x = wt, y = mpg),
        color = "blue"
    )
```



- 하지만 위와 같이 사용하는 경우는 흔하지 않다. 더 흔한 패턴은 `ggplot()` 함수에 데이터를 전달하여, 그 이하에 뒤따르는 모든 레이어가 자체의 데이터셋을 지정하지 않는 경우에는 `ggplot()` 함수가 전달된 데이터셋을 사용한다. 데이터 뿐만 아니라 매핑도 마찬가지이다. 그런데 어느 함수에 어떻게 지정하는지에 따라서 결과가 달라질 수 있다. 조금 시행착오를 하고 나면 분명히 이해가 될 것이다.

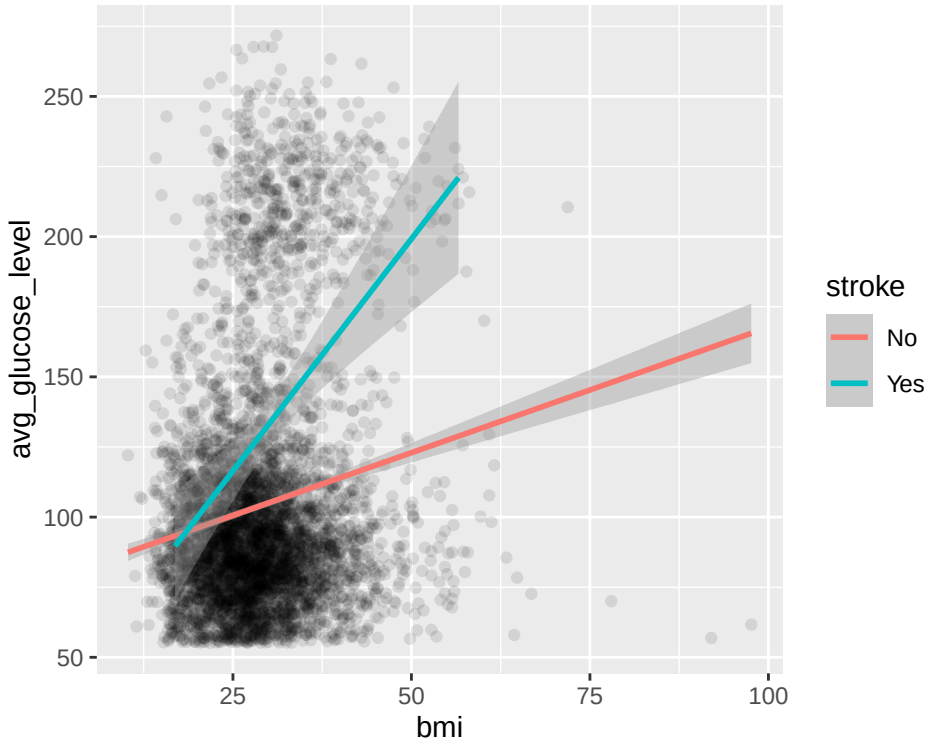
다음 경우는 데이터와 매핑이 `ggplot()` 함수에 사용되었고, 뒤따르는 `geom_point()` 함수와 `geom_smooth()` 함수는 자체의 데이터셋과 매핑을 지정하지 않았다. 따라서 `ggplot()` 함수에 전달된 데이터셋과 매핑을 사용하여 그래프를 만든다.

```
ggplot(stroke_df, aes(x = bmi, y = avg_glucose_level)) +
    geom_point(alpha = 0.1) +
    geom_smooth(method = "lm")
```



다음 경우는 `geom_point()` 함수는 데이터셋과 매핑을 지정하지 않아서 `ggplot()` 함수에 전달된 데이터셋과 매핑을 사용하여 그래프를 만든다. 반면, `geom_smooth()` 함수는 자체의 매핑을 사용하여 그래프를 만든다.

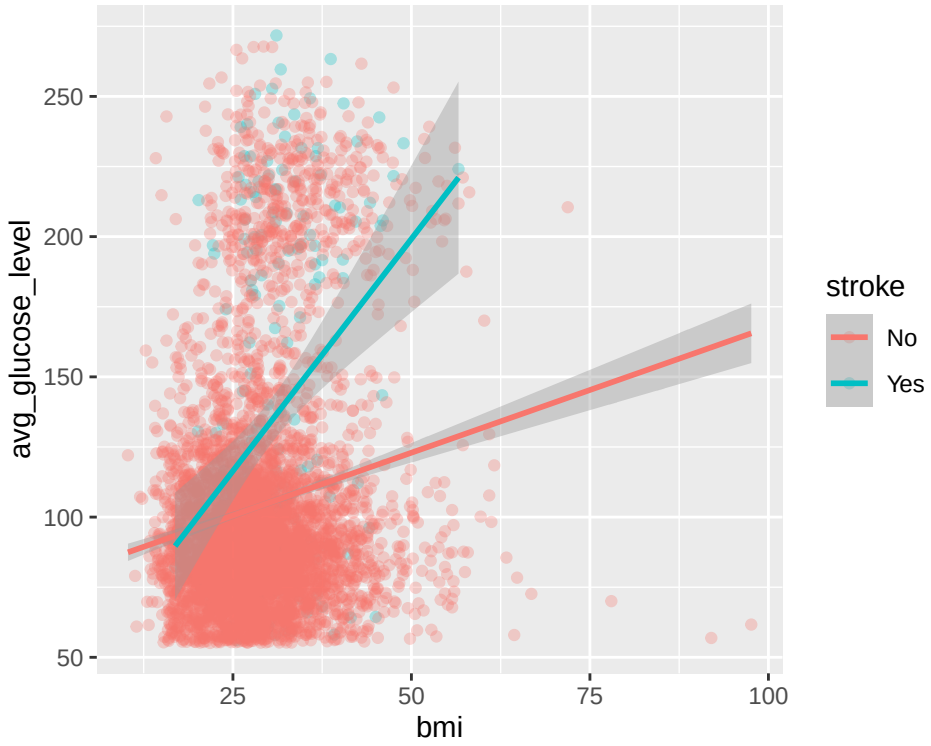
```
ggplot(stroke_df, aes(x = bmi, y = avg_glucose_level)) +
    geom_point(alpha = 0.1) +
    geom_smooth(aes(color = stroke), method = "lm")
```



- 그룹핑이나 패시팅은 모두 데이터를 분리하여 그래프를 만든다는 점에서는 유사한데, 경우에 따라 어떤 것이 더 효율적일 수 있다.

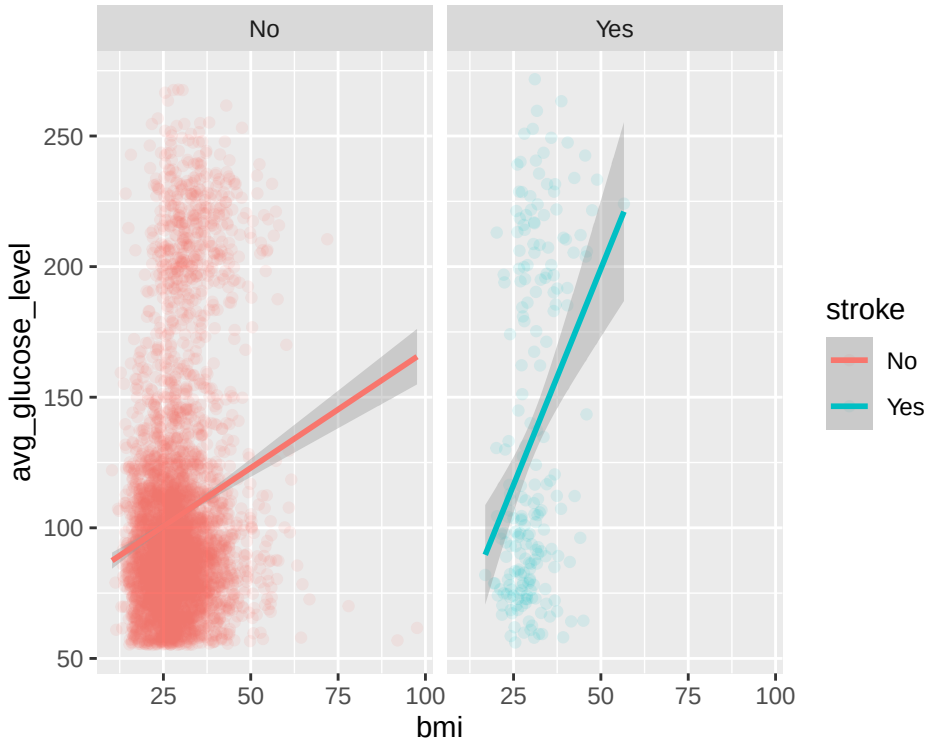
다음은 색에 의한 그룹핑을 사용한 경우이다. 이 경우에는 오버플롯팅으로 눈에 잘 들어오지 않는다.

```
ggplot(stroke_df, aes(x = bmi, y = avg_glucose_level, color = stroke)) +
    geom_point(alpha = 0.3) +
    geom_smooth(method = "lm")
```



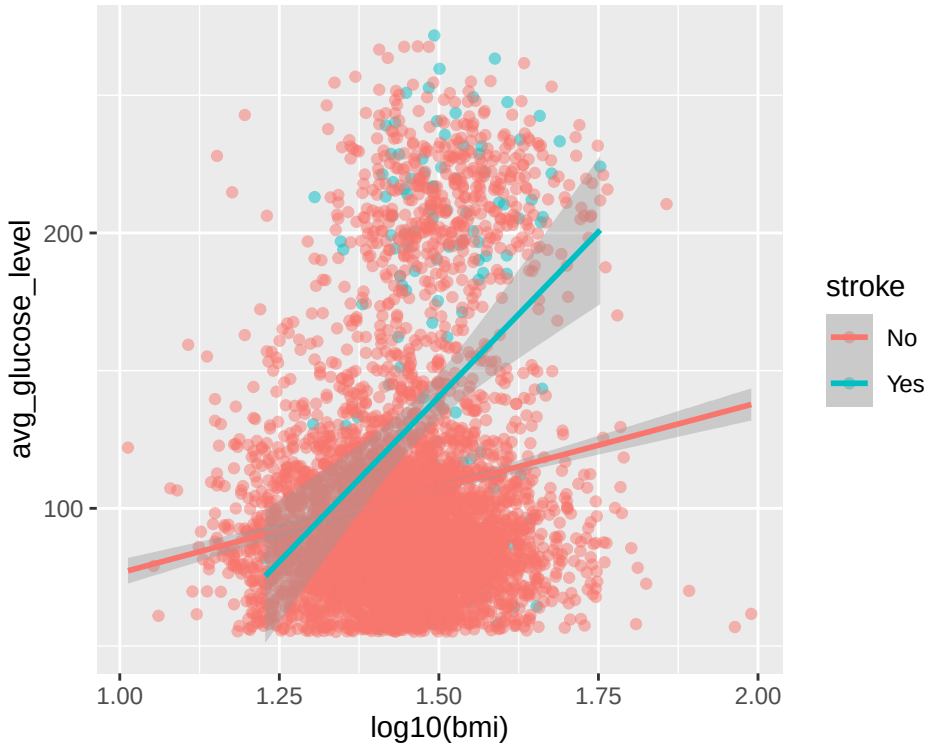
다음은 패시팅을 적용하여 분리한 것이다. 이 그래프를 보면 뇌졸중이 있는 그룹에서 `bmi` 대 `avg_glucose_level` 사이의 상관관계가 더 크다는 것을 보여준다.

```
ggplot(stroke_df, aes(x = bmi, y = avg_glucose_level, color = stroke)) +
    geom_point(alpha = 0.1) +
    geom_smooth(method = "lm") +
    facet_wrap(~stroke)
```



- ggplot2 패키지의 함수 안에서 일반 수학 연산이 모두 가능하다.

```
ggplot(stroke_df, aes(x = log10(bmi), y = avg_glucose_level, color = stroke)) +
    geom_point(alpha = 0.5) +
    geom_smooth(method = "lm")
```



- `ggplot2` 패키지는 tidy 데이터프레임을 사용한다. 따라서 `dplyr`, `tidyr` 패키지 등을 사용한 데이터 전처리가 필요한 경우들이 있다.
- 데이터 프로세스와 테마 프로세스의 분리: `ggplot2` 패키지는 데이터 관련된 프로세스와 제목/레이블/테마 등 주변 시각적 요소와 관련된 프로세스를 분리하여 생각하는 것이 중요하다.
    - 보통 데이터 관련된 프로세스를 먼저 진행하고, 그 다음 제목/레이블/테마 등을 진행하는 과정을 밟는다.
- RStudio 그래프 뷰어는 **한글**을 지원하지 않는다. 이런 경우에는 `showtext` 패키지를 사용하여 한글을 사용할 수 있다.
- 완성된 그래픽 객체는 `ggsave()` 함수를 사용하여 다양한 크기, 포맷, 해상도 등으로 저장할 수 있다.

21 스케일(scales)과 Color Scale

여기선 ggplot2 패키지에서의 스케일(scales)의 개념을 소개하고, 그 가운데 실제로 그래프를 만들 때 중요한 색상 스케일에 대해서 설명하고자 한다.

21.1 스케일과 스케일의 역할

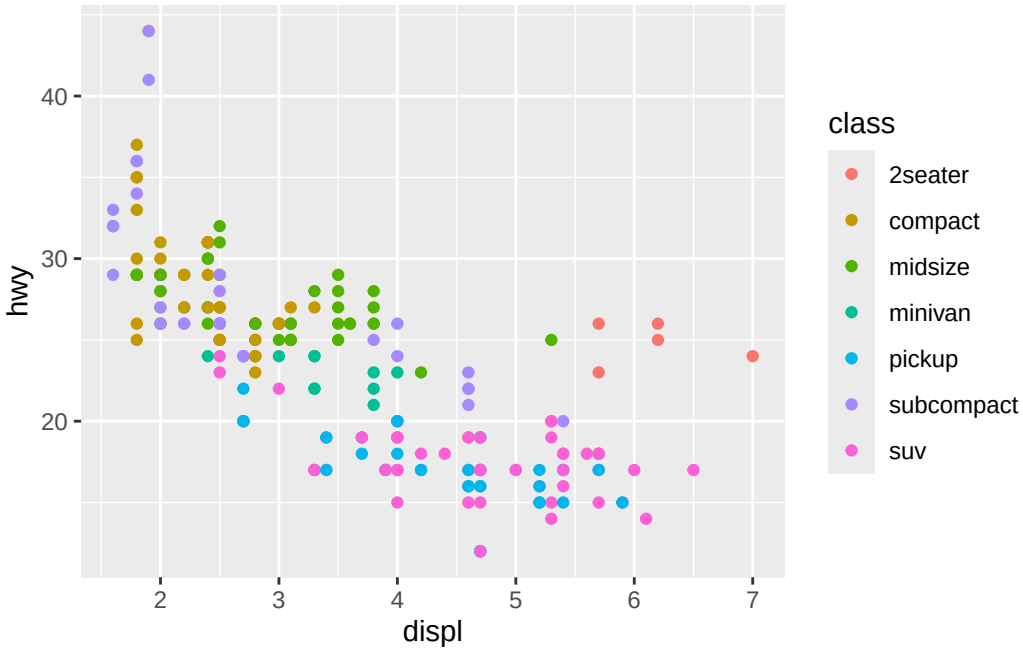
ggplot2 패키지에서 스케일은 데이터를 매핑하는 함수이다. ggplot2 패키지의 용어로 말하면 데이터를 aesthetic 속성에 매팽하는 데 사용되는 함수이다. 따라서 각각의 매핑에는 거기에 해당되는 스케일이 사용된다. 언뜻 보기에 스케일을 사용하지 않는 것처럼 보이는 경우에도 디폴트 스케일이 사용된다. 다음 예시를 보자.

```
library(ggplot2)
library(dplyr)
glimpse(mpg)
```

```
Rows: 234
Columns: 11
$ manufacturer <chr> "audi", "audi", "audi", "audi", "audi", "audi", "audi", "~
$ model        <chr> "a4", "a4", "a4", "a4", "a4", "a4", "a4", "a4 quattro", "~
$ displ        <dbl> 1.8, 1.8, 2.0, 2.0, 2.8, 2.8, 3.1, 1.8, 1.8, 2.0, 2.0, 2.~
$ year         <int> 1999, 1999, 2008, 2008, 1999, 1999, 2008, 1999, 1999, 200~
$ cyl          <int> 4, 4, 4, 4, 6, 6, 6, 4, 4, 4, 4, 6, 6, 6, 6, 6, 6, 8, 8, ~
$ trans        <chr> "auto(l5)", "manual(m5)", "manual(m6)", "auto(av)", "auto~
$ drv          <chr> "f", "f", "f", "f", "f", "f", "f", "4", "4", "4", "4", "4~
$ cty          <int> 18, 21, 20, 21, 16, 18, 18, 18, 16, 20, 19, 15, 17, 17, 1~
$ hwy          <int> 29, 29, 31, 30, 26, 26, 27, 26, 25, 28, 27, 25, 25, 25, 2~
$ fl           <chr> "p", "p", "p", "p", "p", "p", "p", "p", "p", "p", "p", "p~
```

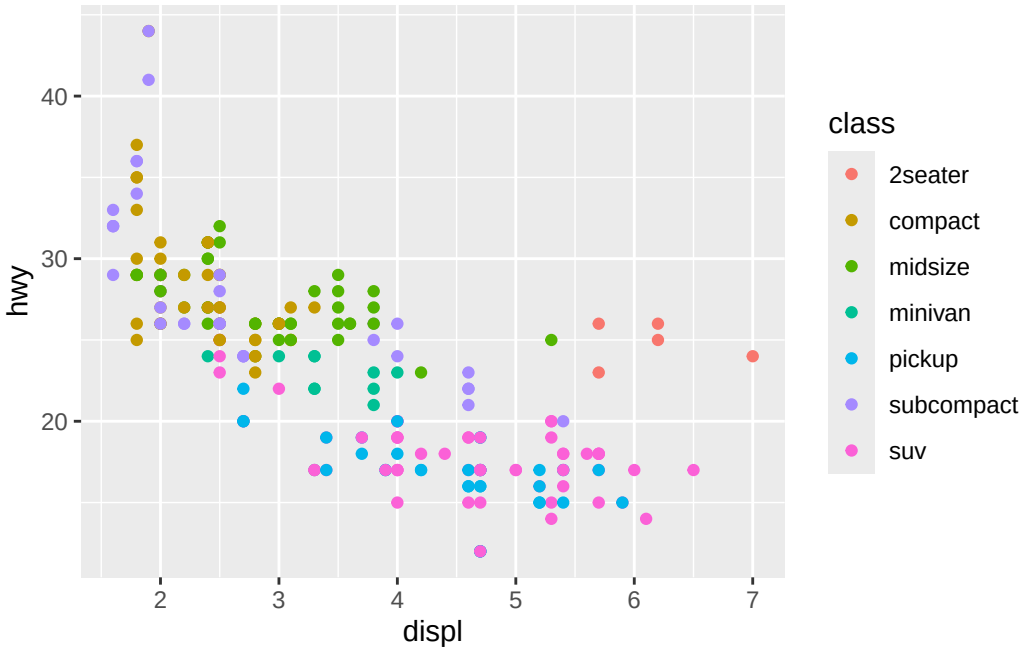
```
$ class        <chr> "compact", "compact", "compact", "compact", "compact", "c~
```

```
ggplot(mpg, aes(x = displ, y = hwy, color = class)) +
    geom_point()
```



위 그래프에는 `x`, `y`, `color` aesthetic이 사용되고 있고, 각각의 매핑에는 디폴트 스케일이 사용된다. 이것을 디폴트를 풀어서 보면 다음과 같다.

```
ggplot(mpg, aes(x = displ, y = hwy, color = class)) +
    geom_point() +
    scale_x_continuous() +
    scale_y_continuous() +
    scale_color_discrete()
```



- `x` aesthetic에 대한 매핑: `scale_x_continuous()`
- `y` aesthetic에 대한 매핑: `scale_y_continuous()`
- `color` aesthetic에 대한 매핑: `scale_color_discrete()`

스케일 함수들의 이름은 위와 같이 `scale_<aesthetic>_<type>()` 형식으로 되어 있다. 여기서 `<aesthetic>`은 스케일이 적용되는 aesthetic 속성의 이름이고, `<type>`은 스케일의 타입이다. 이 `type`은 데이터의 타입에 따라 다르다. 예를 들어, `x`와 `y`는 수치형 데이터이고, `color`는 범주형 데이터이다. 따라서 `x`와 `y`에 대한 스케일은 `continuous`이고, `color`에 대한 스케일은 `discrete`이다. 즉, 변수의 타입에 따라 스케일의 타입이 결정된다.

이 스케일 함수들의 도움말을 찾아 보면 표 21.1와 같은 인자들이 사용된다는 것을 알 수 있다.

- `name` 인자: 축의 레이블을 지정할 수 있고, 레젠드의 제목을 지정할 수 있다.
- `breaks` 인자: 축의 눈금을 지정할 수 있고, 레젠드의 키를 지정할 수 있다.
- `labels` 인자: 축의 눈금 레이블을 지정할 수 있고, 레젠드의 키 레이블을 지정할 수 있다.

표 21.1: 스케일 함수의 인자 이름과 의미

스케일 함수 인자	축	레전드
`name`	Label	Title
`breaks`	Ticks & grid line	Key

표 21.1: 스케일 함수의 인자 이름과 의미

스케일 함수 인자	축	레전드
`labels`	Tick label	Key label

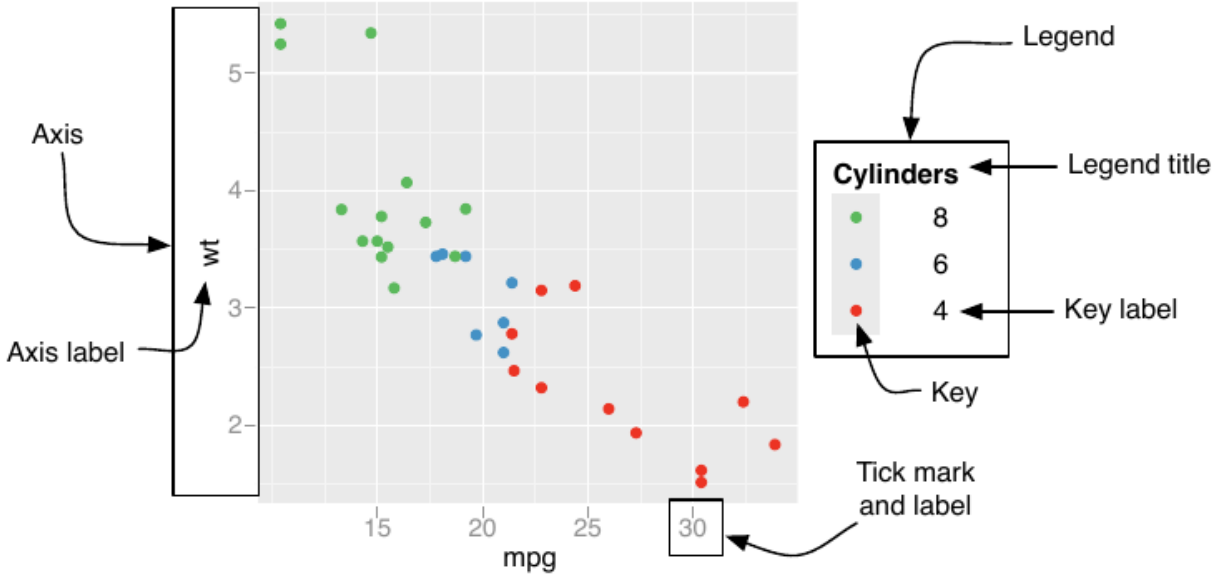
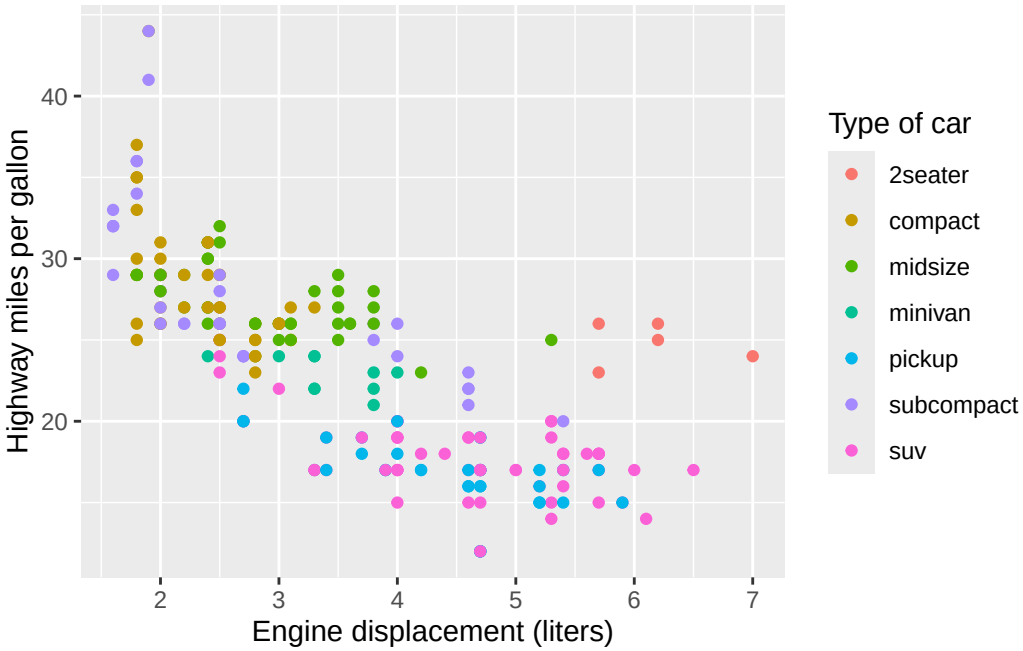


그림 21.1: 스케일: 축과 레전드

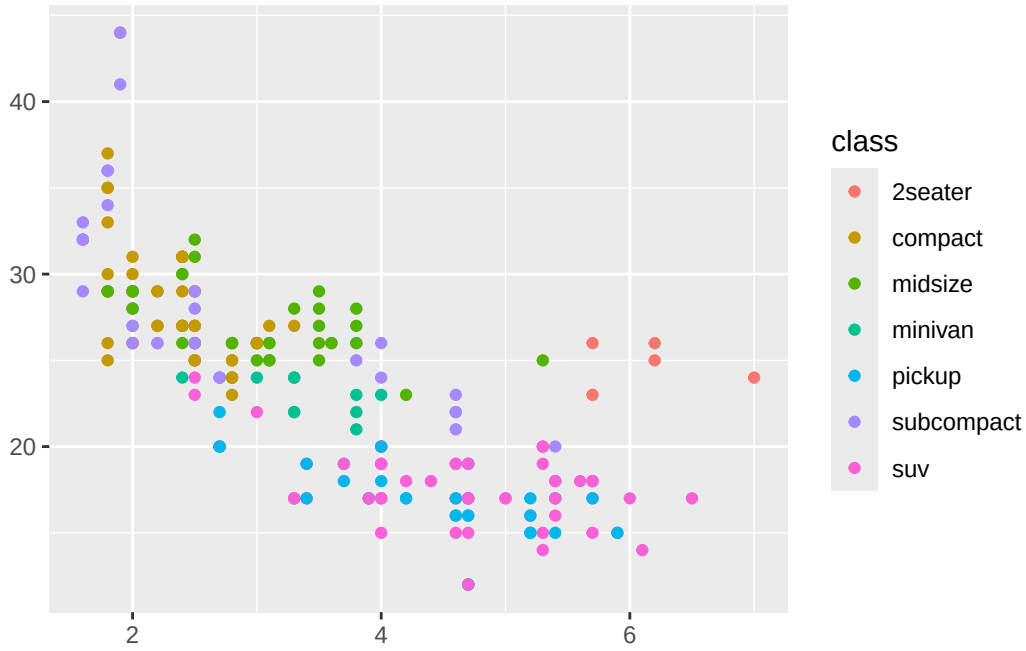
이 내용을 위 그래프에 적용해 보자.

```
ggplot(mpg, aes(x = displ, y = hwy, color = class)) +
    geom_point() +
    scale_x_continuous(name = "Engine displacement (liters)") +
    scale_y_continuous(name = "Highway miles per gallon") +
    scale_color_discrete(name = "Type of car")
```



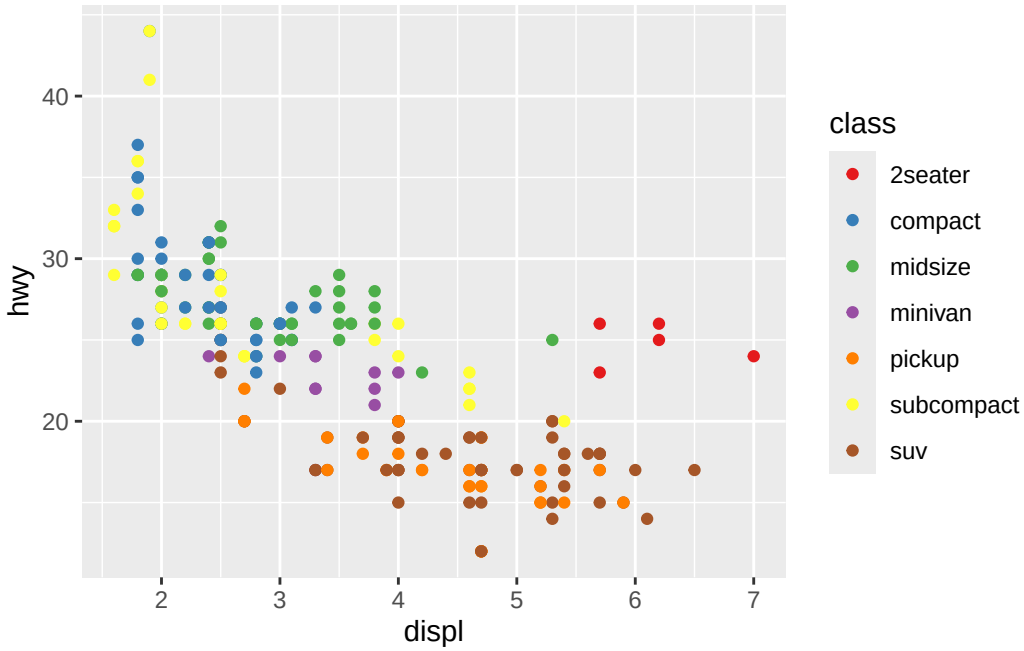
축의 틱(ticks)을 변경할 수도 있고, `NULL`을 `name` 인자에 주면 레이블이 사라진다.

```
ggplot(mpg, aes(x = displ, y = hwy, color = class)) +
    geom_point() +
    scale_x_continuous(name = NULL, breaks = seq(2, 8, by = 2)) +
    scale_y_continuous(name = NULL, breaks = seq(10, 50, by = 10))
```



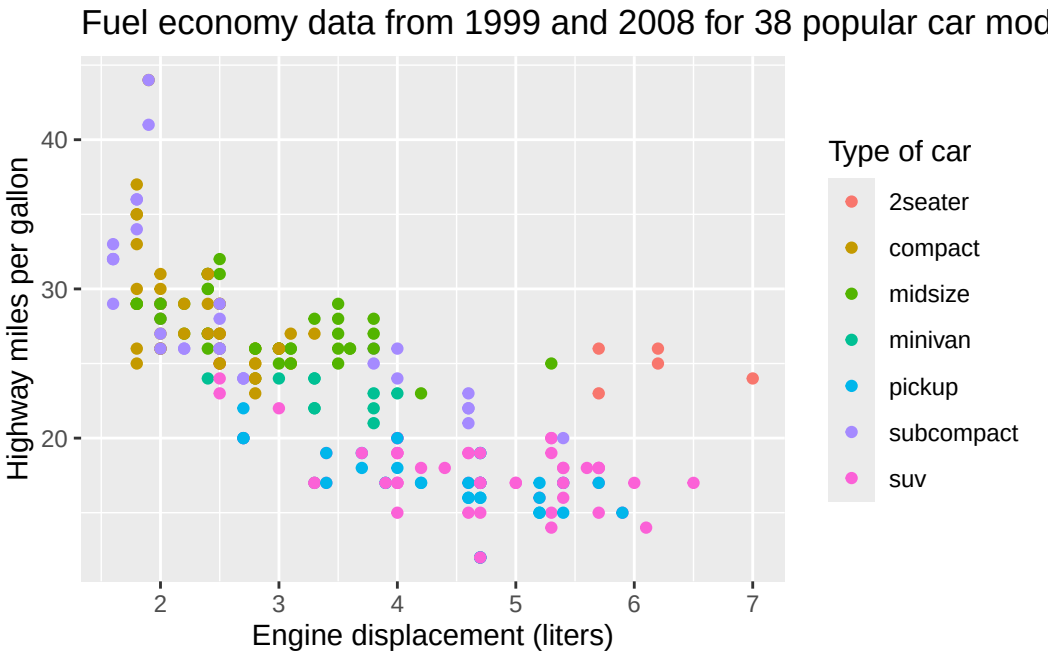
이제 디폴트 스케일이 아닌 다른 스케일 함수를 지정해 보자. `color` aesthetic에 `scale_color_brewer()` 함수로 새로운 색상 팔레트를 지정해 보았다.

```
ggplot(mpg, aes(x = displ, y = hwy, color = class)) +
    geom_point() +
    scale_x_continuous() +
    scale_y_continuous() +
    scale_color_brewer(palette = "Set1")
```



이와 같이 스케일은 데이터를 매핑하고, 그 결과로 축과 레젠드를 만들어 낸다. 따라서 축과 레전드에 관련된 것들은 스케일 함수를 통해서 조절할 수 있다. 그렇지만, 레이블 관련된 작업은 아주 흔하기 때문에 레이블을 바꾸기 위해서 스케일 함수를 사용하는 것은 비효율적이다. 그래서 이런 작업을 쉽게 하기 위해서 `labs()` 함수를 제공한다. 다음은 `labs()` 함수를 사용한 예시이다.

```
ggplot(mpg, aes(x = displ, y = hwy, color = class)) +
    geom_point() +
    labs(
        title = "Fuel economy data from 1999 and 2008 for 38 popular car models",
        x = "Engine displacement (liters)",
        y = "Highway miles per gallon",
        color = "Type of car"
    )
```



21.2 색상 스케일(color scale)

그래프에서 색상은 매우 중요한 역할을 한다. 그런데 색상을 잘 선택하는 것은 쉬운 일이 아니다. 여기서는 나처럼 색에 대한 감이 떨어지는 사람들을 위해 그래프에서 색상을 잘 선택하는 방법을 정리하고자 한다.

클라우스 윌케(Claus Wilke)의 책에 따르면 그래프에서 색은 다음 3가지 가운데 하나의 역할을 하는데 사용된다.

1. 데이터 그룹간의 차이를 드러내고자 할 때(Qualitative, Diversing)
2. 데이터의 값들을 표현하고자 할 때(Sequential)
3. 어떤 내용을 강조하고자 할 때(Accent)

`ggplot2` 패키지로 이런 개념을 살리고자 할 때 우리가 어떤 색상 팔레트를 사용할 것인가라는 문제로 귀결된다.

21.2.1 색상 팔레트(Color Palette)

색상 팔레트(color palette)란 화가의 팔레트처럼 조화로운 색들을 모은 것을 말한다. 그리고 이런 팔레트는 색의 역할에 따라 역시 3가지로 나눌 수 있다.

1. Qualitative
2. Diverging
3. Sequential

그래서 팔레트를 선택하는 방법은 다음과 같이 정리할 수 있다.

1. 어떤 목적인가?
2. 몇 개의 색을 사용할 것인가?
3. 그리고 어떤 색을 중심으로 사용할 것인가?
4. 어디에 쓸 것인가? 즉. 지도를 표시하는 데 사용할지, 선 그래프에 사용할지, 히트맵 등에 사용할지 등을 결정한다.
5. 좀 더 나아가면, 색각 이상 사람들도 잘 볼 수 있게 하려면 어떤 색을 선택해야 하는지 등을 고려한다.

지도를 표현할 때 많이 사용되는 COLORBREWER 2.0 사이트를 방문하여 연습해 보면 좋겠다. 실제로 뒤에서 설명할 ggplot2 패키지 내에서 색상 팔레트 생성 방법에 여기에 기초해서 만들어 졌다.

colorspace 패키지는 색상 팔레트의 생성과 선택을 돕는 패키지로, 색 관련한 가장 포괄적인 기능들을 제공한다. 그래도 그 이전에 추가 패키지를 사용하지 않고 ggplot2 패키지 내에서 색상 팔레트를 생성하는 방법을 소개하고자 한다.

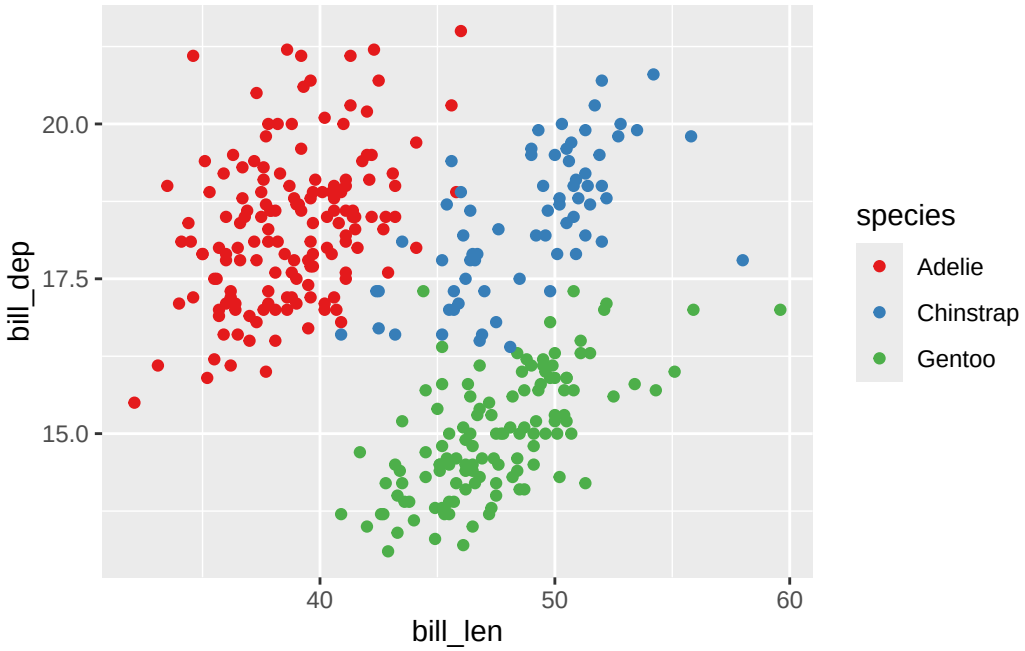
21.2.2 ggplot2 패키지 내에서 색상 팔레트 생성

ggplot2 패키지에서 팔레트 사용하는 방법은 ColorBrewer 기반 스케일를 참고한다

- `scale_color_brewer()` 함수: `color` aesthetic에 사용되는 색상 팔레트
- `scale_fill_brewer()` 함수: `fill` aesthetic에 사용되는 색상 팔레트

```
ggplot(penguins) +
    geom_point(aes(x = bill_len, y = bill_dep, color = species)) +
    scale_color_brewer(palette = "Set1")
```

```
Warning: Removed 2 rows containing missing values or values outside the scale range
(`geom_point()`).
```



21.2.3 colorspace 패키지

colorspace 패키지는 아주 광범위한 기능을 제공하는 패키지인데 여기선 팔레트를 선택하는 방법에 초점을 맞추어 설명하고자 한다.

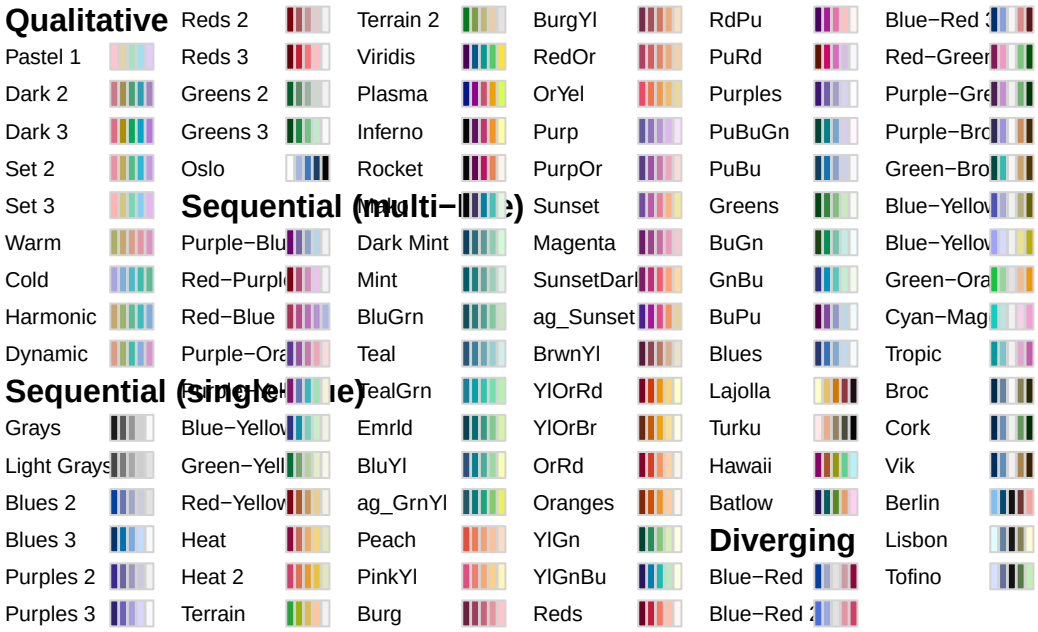
- `hcl_palettes()` 함수는 팔레트의 속성과 색상을 확인하는 데 사용된다. `plot = TRUE`가 없으면 이름을 출력하고, 있으면 이것을 색으로 디스플레이한다.

```
library(colorspace)
hcl_palettes()
```

```
HCL palettes

Type:  Qualitative
Names: Pastel 1, Dark 2, Dark 3, Set 2, Set 3, Warm, Cold, Harmonic, Dynamic

Type:  Sequential (single-hue)
Names: Grays, Light Grays, Blues 2, Blues 3, Purples 2, Purples 3, Reds 2,
       Reds 3, Greens 2, Greens 3, Oslo
```

```
Type:  Sequential (multi-hue)
Names: Purple-Blue, Red-Purple, Red-Blue, Purple-Orange, Purple-Yellow,
       Blue-Yellow, Green-Yellow, Red-Yellow, Heat, Heat 2, Terrain,
       Terrain 2, Viridis, Plasma, Inferno, Rocket, Mako, Dark Mint,
       Mint, BluGrn, Teal, TealGrn, Emrld, BluYl, ag_GrnYl, Peach,
       PinkYl, Burg, BurgYl, RedOr, OrYel, Purp, PurpOr, Sunset,
       Magenta, SunsetDark, ag_Sunset, BrwnYl, YlOrRd, YlOrBr, OrRd,
       Oranges, YlGn, YlGnBu, Reds, RdPu, PuRd, Purples, PuBuGn, PuBu,
       Greens, BuGn, GnBu, BuPu, Blues, Lajolla, Turku, Hawaii, Batlow

Type:  Diverging
Names: Blue-Red, Blue-Red 2, Blue-Red 3, Red-Green, Purple-Green,
       Purple-Brown, Green-Brown, Blue-Yellow 2, Blue-Yellow 3,
       Green-Orange, Cyan-Magenta, Tropic, Broc, Cork, Vik, Berlin,
       Lisbon, Tofino
```

```
hcl_palettes(plot = TRUE)
```



- `qualitative_hcl()`, `sequential_hcl()`, `diverging_hcl()` 함수는 각각 퀄리티브, 시퀀셜, 다이버징 팔레트를 생성하는 데 사용된다. `n`은 색의 개수이고, `palette`는 위에서 본 팔레트 가운데 선택하여 입력한다.

```
qualitative_hcl(n = 5, palette = "Dark 3")
```

```
[1] "#E16A86" "#AA9000" "#00AA5A" "#00A6CA" "#B675E0"
```

```
sequential_hcl(n = 5, palette = "Blues 3")
```

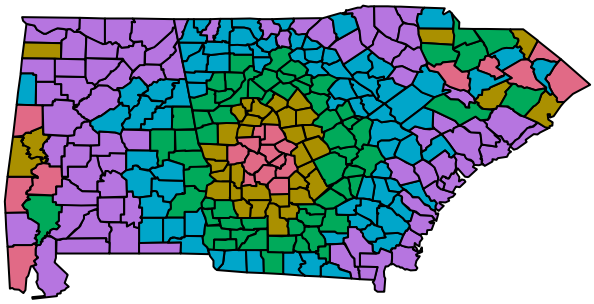
```
[1] "#00366C" "#0072B4" "#79ABE2" "#C3DBFD" "#F9F9F9"
```

```
diverging_hcl(n = 5, palette = "Blue-Red 3")
```

```
[1] "#002F70" "#879FDB" "#F6F6F6" "#DA8A8B" "#5F1415"
```

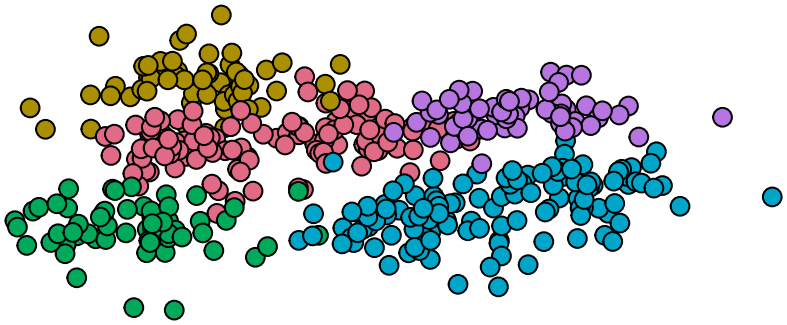
- `demoplot()` 함수에 첫 번째 인자에서 앞에서 만든 팔레트를 주고, `type` 인자에 “map”, “heatmap”, “scatter”, “spine”, “bar”, “pie”, “perspective”, “mosaic”, “lines” 중 하나를 선택하여 입력하여, 해당 팔레트를 사용하여 간단한 플롯을 만들어 볼 수 있는 함수이다.

```
q5 <- qualitative_hcl(n = 5, palette = "Dark 3")
demoplot(q5, type = "map")
```



```
demoplot(q5, type = "heatmap")
```

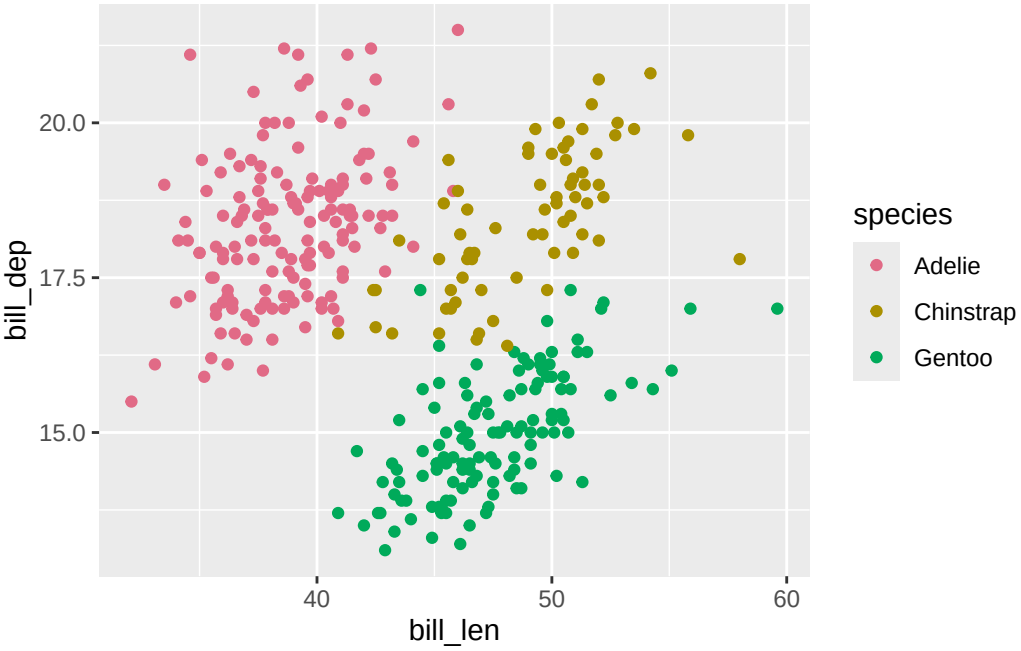
```
demoplot(q5, type = "scatter")
```



- 만약 앞에서 만든 q5 팔레트를 사용하고 싶다면 다음과 같이 ggplot2 패키지의 scale_color_manual() 함수를 사용하면 된다.

```
ggplot(penguins) +
    geom_point(aes(x = bill_len, y = bill_dep, color = species)) +
    scale_color_manual(values = q5)
```

```
Warning: Removed 2 rows containing missing values or values outside the scale range
(`geom_point()`).
```



- `hcl_wizard()` 함수는 `colorspace` 패키지의 모든 기능을 시각적으로 확인하고, 색 팔레트 값을 변환하는 Shiny 앱을 실행한다.

```
pl <- hcl_wizard()
```

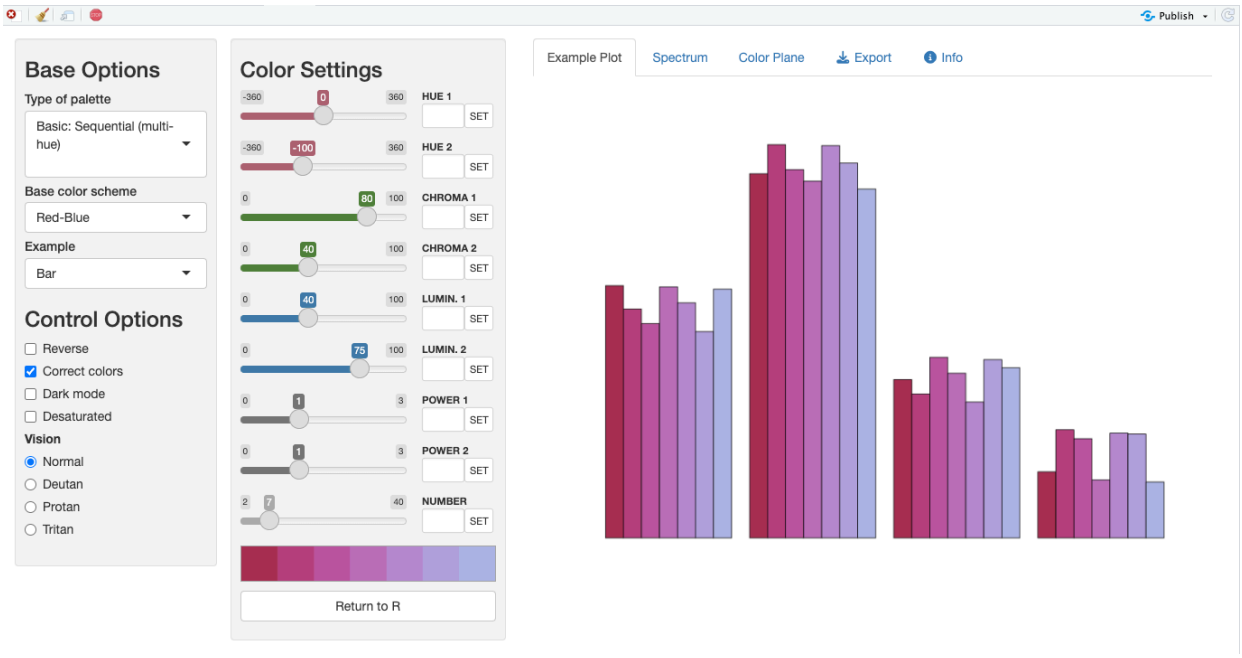


그림 21.2: `hcl_wizard()`으로 실행한 Shiny 앱

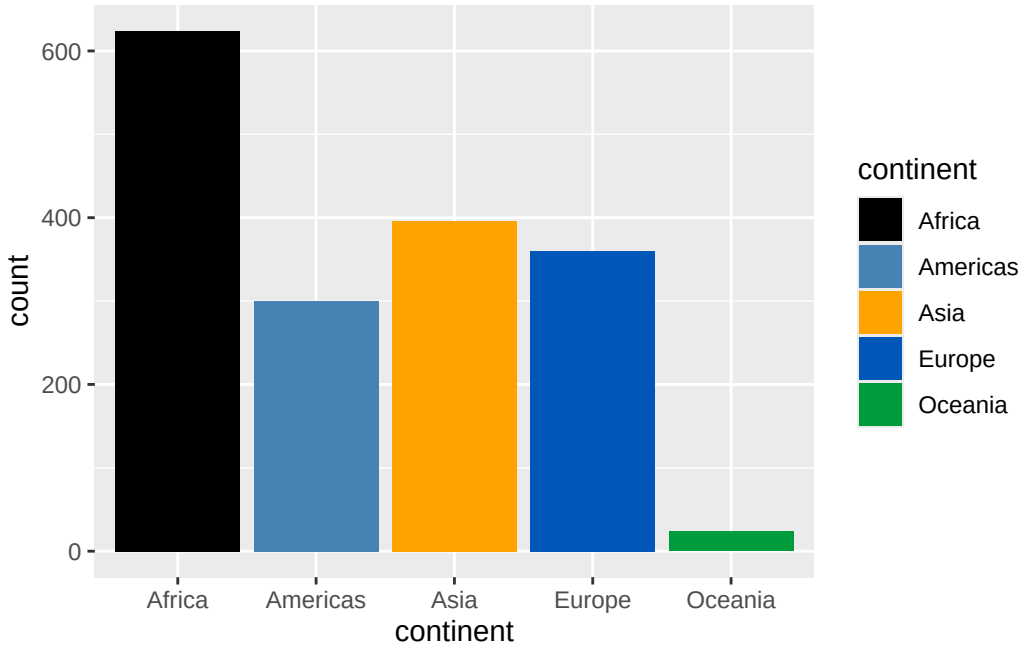
만약 위와 같이 실행했다가 Shiny 앱을 종료하고 나면 `pl`이라는 함수를가 생성된다. 여기에 `pl(5)` 같이 색의 개수를 주면 해당 팔레트에 기반한 색상을 반환한다.

21.3 매뉴얼로 자신의 원하는 임의의 색상 사용하기

`ggplot2` 패키지는 거의 대부분의 매핑이라는 **데이터의 값에 따라서** 뭔가를 하도록 하게 되어 있다. 그런데 가끔은 사용자의 임의의 색상을 사용하고 싶을 때가 있다.

이런 경우는 Create your own discrete scale 도움말 페이지를 참고한다. 특히n `scale_*_manual()` 스케일 함수들의 `values` 인자에 주목한다. 이름을 가진 벡터를 사용하면 해당 변수에 임의의 색상을 지정할 수 있다.

```r
# 오륜기 색을 사용
library(dplyr)
library(gapminder)
gapminder %>%
    mutate(continent = continent) %>%
    ggplot() +
    geom_bar(aes(x = continent, fill = continent)) +
    scale_fill_manual(
        values = c(
            "Africa" = "#000000", "Americas" = "steelblue",
            "Asia" = "#FFA300", "Europe" = "#0057B8",
            "Oceania" = "#009B3A"
        )
    )
```



21.4 정리

- ggplot2 패키지에서 팔레트 사용하는 방법은 ColorBrewer 기반 스케일를 참고한다

- `colorspace` 패키지는 색상 팔레트의 생성과 선택을 돕는 패키지로, 색 관련한 가장 포괄적인 기능들을 제공한다.

22 ggplot2 레이어(layers)와 통계적 변환(stats)

```
library(ggplot2)
library(tidyverse)
```

이 장에서는 ggplot2 패키지의 ggplot2의 레이어(layer)에 대해서 설명하고, Statistical Transformation 개념을 이해하고 활용하는 방법을 소개한다.

22.1 ggplot2의 레이어(layer)

ggplot2는 레이어를 + 연산자를 사용하여 하나씩 쌓아올리면서 그래프를 만든다.

ggplot2에서 레이어는 다음 5가지 구성요소를 가지고 있다. 이 5가지 요소가 모두 갖춰어져야 하나의 레이어가 완성된다.

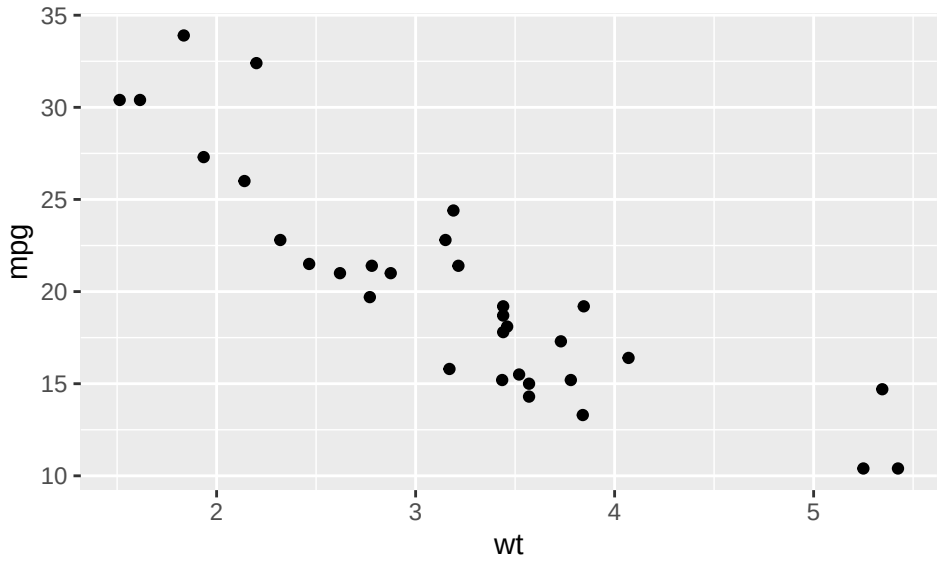
1. 매핑(mapping)
2. 데이터(data)
3. 지옴(geom)
4. 통계학적 변환(stat)
5. 위치(position)

layer()라는 함수를 가지고 이 5가지 요소를 모두 지정하여 레이어를 만들 수 있다. 그러나 이런 방법으로 작업하기는 시간과 노력이 많이 소요된다. 대신 ggplot2는 보통 다음 2가지 방법으로 레이어를 만든다.

1. geom_*() 함수
2. stat_*() 함수

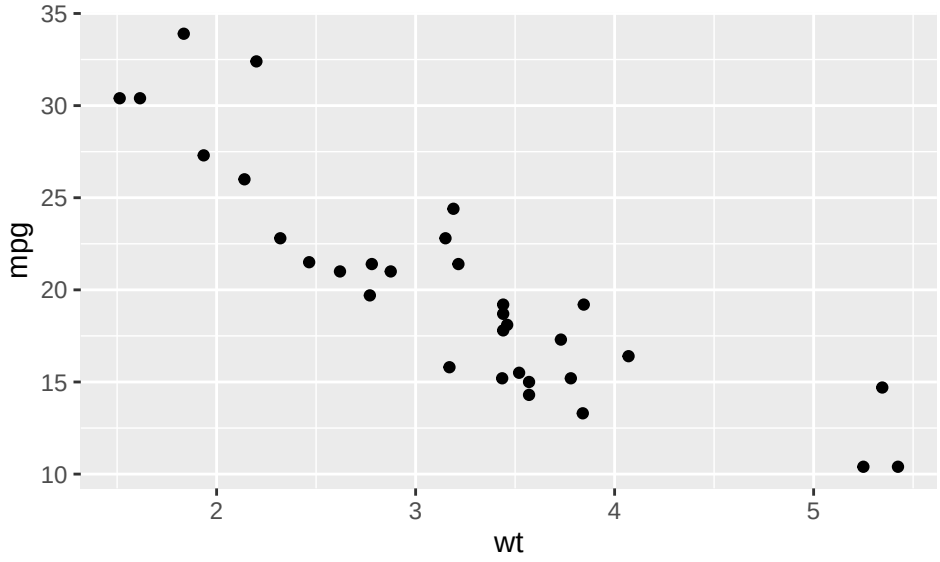
geom_*() 함수는 레이어의 구성 요소 가운데 기하학적 요소인 지옴에서 시작한다.

```
ggplot(mtcars) +
    geom_point(aes(wt, mpg))
```



반면 `stat_*()` 함수는 통계학적 변환(stat)에서 시작한다.

```
ggplot(mtcars) +
    stat_identity(aes(wt, mpg))
```



두 함수가 시작하는 지점은 다르지만 결과는 같다. 5가지가 다 충족하지 않는 것 같지만 빠진 부분은 ggplot2가 설정해 놓은 디폴트를 사용하기 때문에 보이지 않는 것일 뿐이다.

```
ggplot(mtcars) +
    geom_point(aes(wt, mpg))
```

위 코드에서 geom_point()가 생성하는 레이어의 데이터는 ggplot()이라는 모 함수의 것을 그대로 사용하고 있다(data). 매핑은 보이는 바와 같이 aes(wt, mpg)로 x, y position aesthetics에 매핑되고 있다(mapping). 당연히 point 지옴이 사용되고 있다(geom). geom_point()는 stat = "identity"(항등 함수)라는 통계적 변환을 사용한다(stat). 그리고 position = "identity"라는 위치(position) 요소를 디폴트로 사용한다. 따라서 이런 5가지 요소가 갖추어지기 때문에 산점도가 완성된다.

다음 코드도 마찬가지이다. 다만 stat에서 시작하고, stat_identity() 함수의 디폴트인 point 지옴이 사용된다. 나머지도 위와 같은 개념이 적용된다.

```
ggplot(mtcars) +
    stat_identity(aes(wt, mpg))
```

이 장에서 주로 설명할 stat_*() 모양의 함수들을 통계적 변환에서 시작하여, 하나의 레이어를 완성해 나간다. 함수 자체로 레이어를 완성하든 아니면 앞의 ggplot() 함수의 것을 가지고 오든 레이어가 완성되려면 앞에서 설명한 5가지 요소가 다 갖춰야 한다는 점을 주의할 필요가 있다.

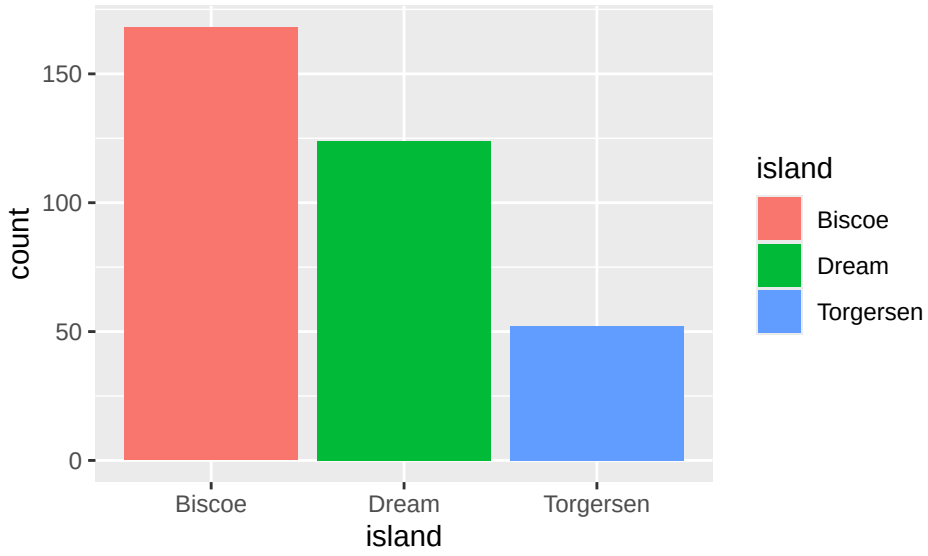
22.2 Statistical Transformation의 역할 이해

보통 처음 ggplot2() 패키지를 배울 때는 기존 통계나 수학 지식을 바탕으로 히스토그램을 만들 때는 geom_histogram(), 박스 플롯을 만들 때는 geom_boxplot(), 산점도(scatter plot)을 만들 때는 geom_point() 등으로 접근하는 것이 직관적이기 때문에 이 방법으로 접근한다. 이 경우에도 보이지는 않지만 다음과 같이 statistical transformation이 사용된다.

- geom_point(): stat = "identity"
- geom_bar(): stat = "count"
- geom_histogram(): stat = "bin"

예를 들어, 막대 그래프를 만들 때 우리는 다음과 같이 시작한다.

```
penguins |>
    ggplot(aes(x = island)) +
    geom_bar(aes(fill = island))
```



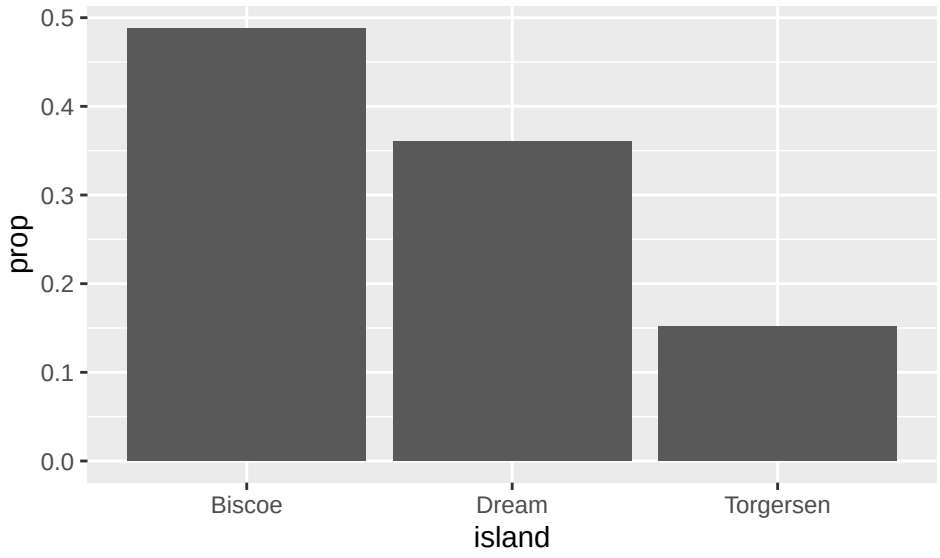
이 경우 우리는 x aesthetic만 `island` 변수에 매핑시켰는데도 막대 그래프가 완성된다. 그것은 내부에서 `stat = "count"`에 의해서 각 섬(island)의 값들을 카운트하고 이 값을 막대의 높이 매핑하기 대문이다. `dplyr` 패키지로 이것을 계산해 보면 다음과 같을 것이다. 내부에서 이와 비슷한 계산을 해 주는 것인 statistical transformation이다.

```
penguins |>
    group_by(island) |>
    count(name = "count")
```

```
# A tibble: 3 x 2
# Groups:   island [3]
  island    count
  <fct>     <int>
1 Biscoe      168
2 Dream       124
3 Torgersen    52
```

만약 "count"가 아니라 비율(portion) 값으로 그래프를 만들려면 geom_bar()의 매핑에서 이것을 선택하면 된다. after_scale() 함수의 사용법은 뒤에서 설명한다. group = 1은 전체 그룹에 대하여 island의 카운트를 고려하라는 뜻이다. 지옴(geom)에 따라 어떤 statistical transformation을 사용할 수 있는지는 해당 지옴의 도움말을 보면 computed variables 항목을 보면 알 수 있다.

```
penguins |>
    ggplot(aes(x = island)) +
    geom_bar(aes(y = after_stat(prop), group = 1))
```



이것은 다음 값들을 폴롯팅한 결과이다.

```
penguins |>
    group_by(island) |>
    summarize(
        count = n(),
        prop = count / nrow(penguins)
    )
```

```
# A tibble: 3 x 3
  island    count  prop
  <fct>     <int> <dbl>
1 Biscoe      168 0.488
```

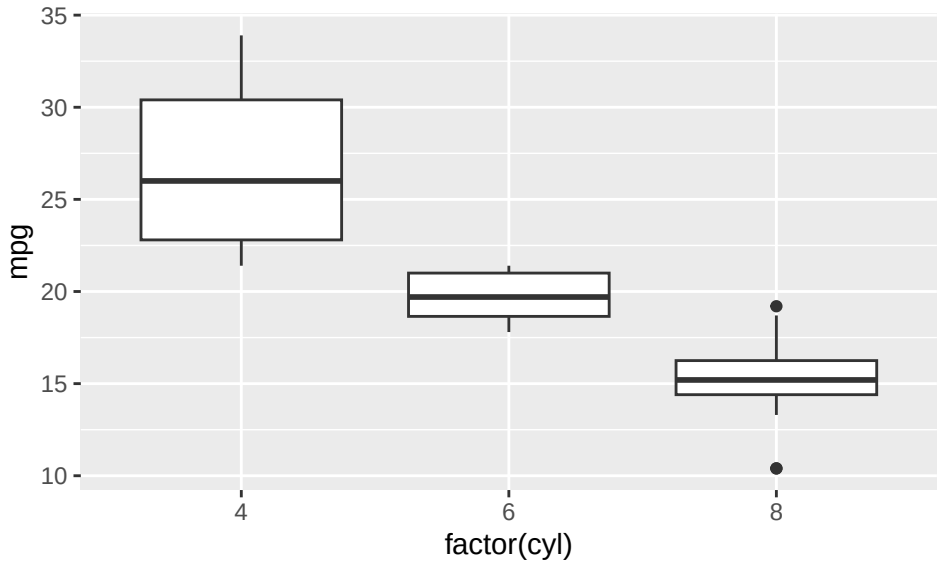
```
2 Dream       124 0.360
3 Torgersen    52 0.151
```

ggplot2 패키지로 그래프를 만들다 보면 어떤 경우에는 어떻게 매핑해야 하는지 헷갈릴 수 있다. 그런 경우에는 dplyr 패키지 등을 사용하여 데이터프레임으로 데이터를 명확히 하고, ggplot2 코드는 가급적 간단하게 하는 것도 한 가지 방법일 수 있다.

22.3 통계 써머리 데이터를 가지고 그래프를 만들기

어떤 경우에는 기하학적 객체(지옴)가 아니라 statistical summary 등이 우선 머리에 떠오를 때가 있다. 다음 예를 보자.

```
ggplot(mtcars, aes(x = factor(cyl), y = mpg)) +
    geom_boxplot()
```

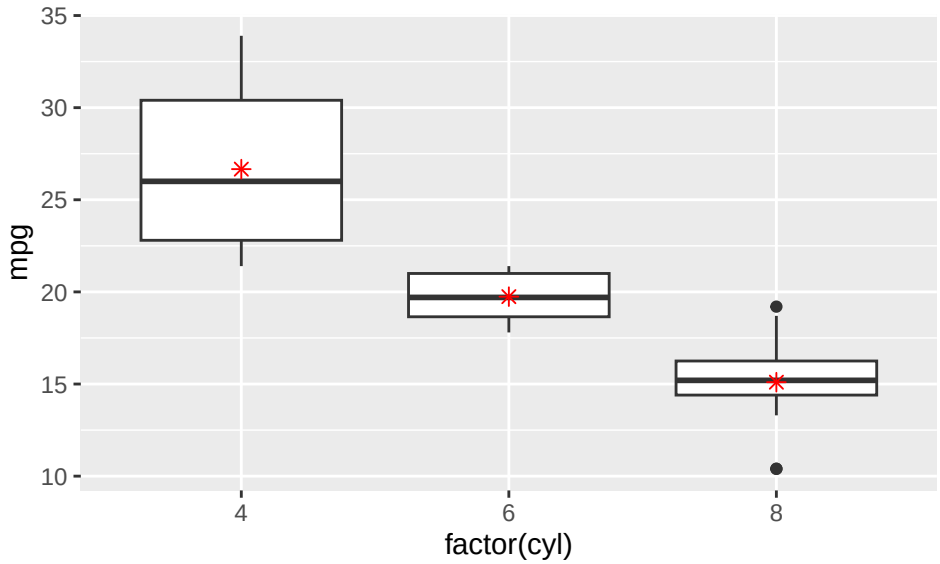


박스 플롯의 가운데 가로 막대는 중앙값(median)을 표시한다. 만약 우리가 평균값을 해당 위치에 점으로 표시하고 싶을 수 있다. 필요한 정보를 생각해 보자.

1. 일단 cyl 별로 평균값이 필요하겠다. 이 값은 점의 y 위치가 될 것이다.
2. 점의 x 위치는 위의 x 축의 위치가 같다.

이럴 때 유용하게 사용되는 함수가 `stat_summary()`이다. 이 함수는 다음과 같이 사용한다.

```
ggplot(mtcars, aes(x = factor(cyl), y = mpg)) +
    geom_boxplot() +
    stat_summary(fun = "mean", geom = "point", shape = 8, color = "red", size = 2)
```



위 코드를 쪼개서 생각해 보자.

- `fun = "mean"`은 `mean()` 함수를 사용하라는 뜻이다. `stat_summary()` 함수는 주어진 `x`에 대하여 `y`의 통계량을 계산한다. 즉 `cyl` 팩터가 `4`인 경우에는 이것의 평균값을 `y`의 위치로 매핑한다. `x` 매핑을 `ggplot()`에서 정의하였기 때문에 이것을 가지고 온다. 따라서 `x`, `y` 위치가 결정된다.
- `geom = "point"`를 통해서 점 geom을 선택했다. 이 점의 색깔과 형태를 (매핑이 아닌) 셋팅으로 지정했다.

다음은 `cyl` 그룹별로 카운팅한 값을 일정한 위치에 표시해 본 예이다. 설명을 위해서 약간 꼬아 놓은 예로, `stat_summary()`를 사용하면서 이것은 `text` 지옴으로 표시하고자 하였다. `text` 지옴의 `x`, `y`, `label` aesthetic이 필요하기 때문에 `label`을 만들기 위해 미리 데이터프레임에 계산을 했다.

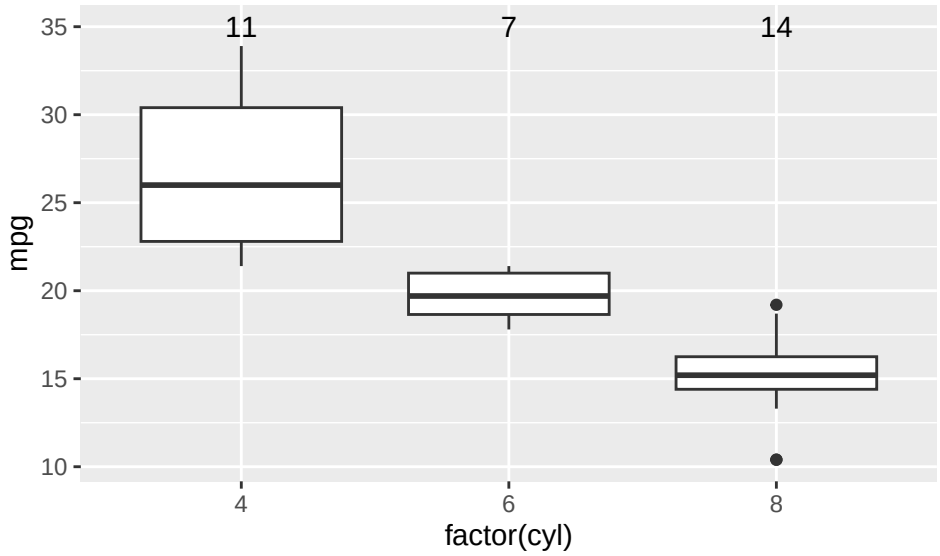
```
df <- mtcars |>
    group_by(cyl) |>
    mutate(cyl_count = n()) |>
    ungroup()
```

```
df
```

```
# A tibble: 32 x 12
     mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb cyl_count
   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>     <int>
 1  21       6  160    110  3.9   2.62  16.5     0     1     4     4         7
 2  21       6  160    110  3.9   2.88  17.0     0     1     4     4         7
 3  22.8     4  108     93  3.85  2.32  18.6     1     1     4     1        11
 4  21.4     6  258    110  3.08  3.22  19.4     1     0     3     1         7
 5  18.7     8  360    175  3.15  3.44  17.0     0     0     3     2        14
 6  18.1     6  225    105  2.76  3.46  20.2     1     0     3     1         7
 7  14.3     8  360    245  3.21  3.57  15.8     0     0     3     4        14
 8  24.4     4  147.    62  3.69  3.19  20       1     0     4     2        11
 9  22.8     4  141.    95  3.92  3.15  22.9     1     0     4     2        11
10  19.2     6  168.   123  3.92  3.44  18.3     1     0     4     4         7
# i 22 more rows
```

```
df |>
    ggplot(aes(factor(cyl), y = mpg, label = cyl_count)) +
    geom_boxplot() +
    stat_summary(aes(y = 35), geom = "text")
```

```
No summary function supplied, defaulting to `mean_se()`
```

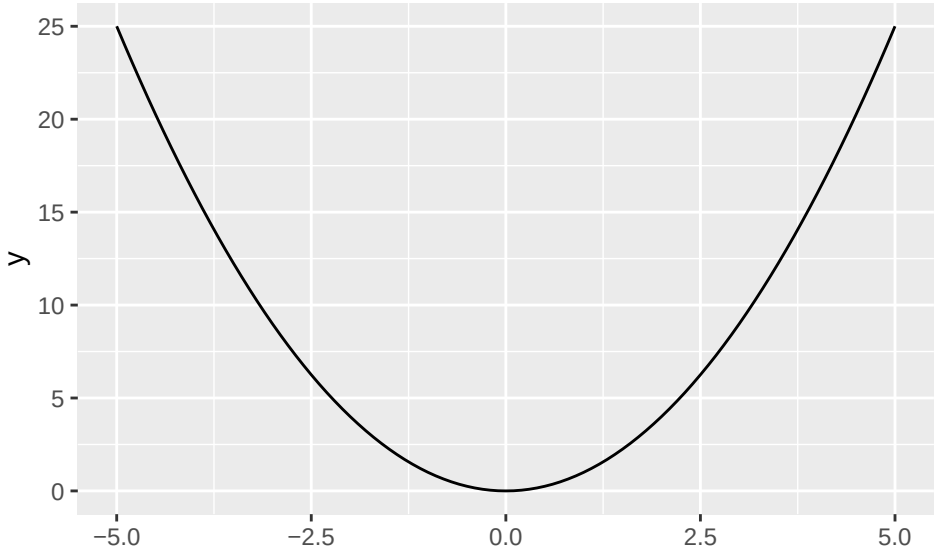


이 경우에는 `stat_summary()` 함수는 `text` 지옴을 사용했고, `x`, `y`, `label` 지옴을 사용하여 하나의 레이어를 완성했다.

22.4 `stat_function()` 함수로 수학 함수를 그래프로 그리기

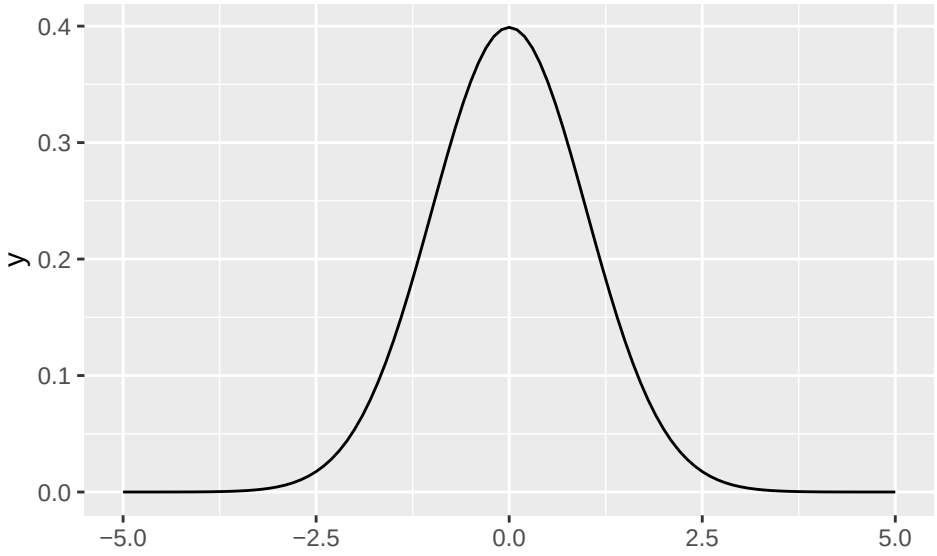
`stat_function()` 함수는 우리가 알고 있는 수학 함수들을 만들 때 편리하다. `fun` 인자에는 수학 함수를 R 언어 형태로 기술한다. 여기서는 anonymous function으로 지정했다.

```
ggplot() +
    stat_function(fun = \(x) x^2, xlim = c(-5, 5))
```

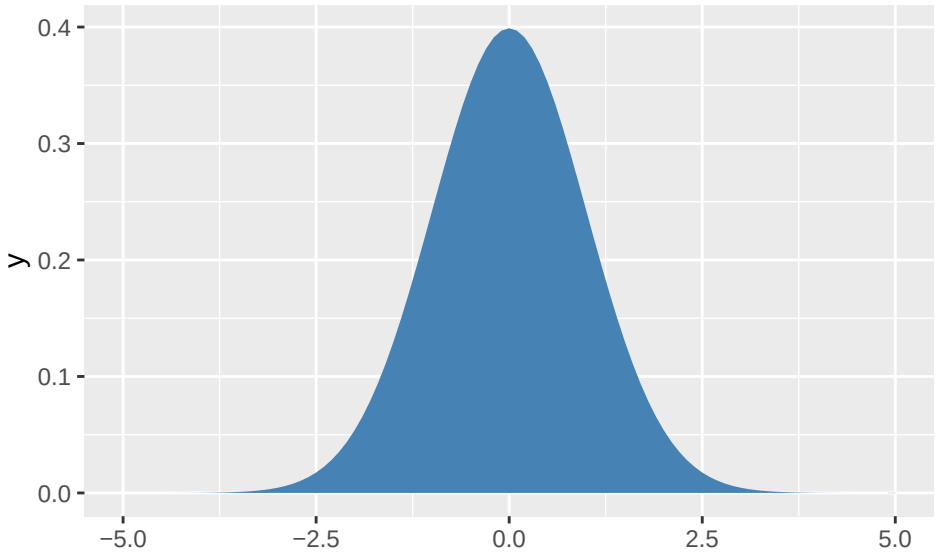


이 함수로 통계 분포 함수를 쉽게 그릴 수 있다.

```
# 표준 정규 분포
ggplot() +
    stat_function(fun = dnorm, xlim = c(-5, 5))
```




```
# 곡선 아래 면적으로 채움
ggplot() +
    stat_function(fun = dnorm, geom = "polygon", xlim = c(-5, 5), fill = "steelblue")
```



22.5 Stat을 사용하는 대표적인 경우: 불확실성에 대한 시각화

아마도 우리가 `stat`을 사용해야 하는 가장 흔한 경우는 불확실성에 대한 시각화일 것이다. 이런 목적을 위해 `ggplot2`는 다음과 같은 지옴 함수를 지원하다.

- `geom_errorbar()`, `geom_linerange()`, `geom_crossbar()`, `geom_pointrange()`
- `geom_smooth()`

먼저 `geom_errorbar()`를 보자. 에러(error)라고 하는 것은 통계적 분석에서 예측된 값과 실제 값의 차이를 의미하지만, 어떤 에러를 사용해야 한다는 명확한 원칙은 없다. 불확실성에 대한 시각화에 대하여 다음 자료를 읽어 보길 권한다.

- Visualizing Uncertainty

`geom_errorbar()`는 `x`, `ymin`, `ymax`라는 aesthetic이 필요한 레이어이다. 따라서 이 조건을 맞추어야 한다.

에러 값으로 사용할 SEM(Standard Error of the mean)을 아래 공식에 따라 준비한다.

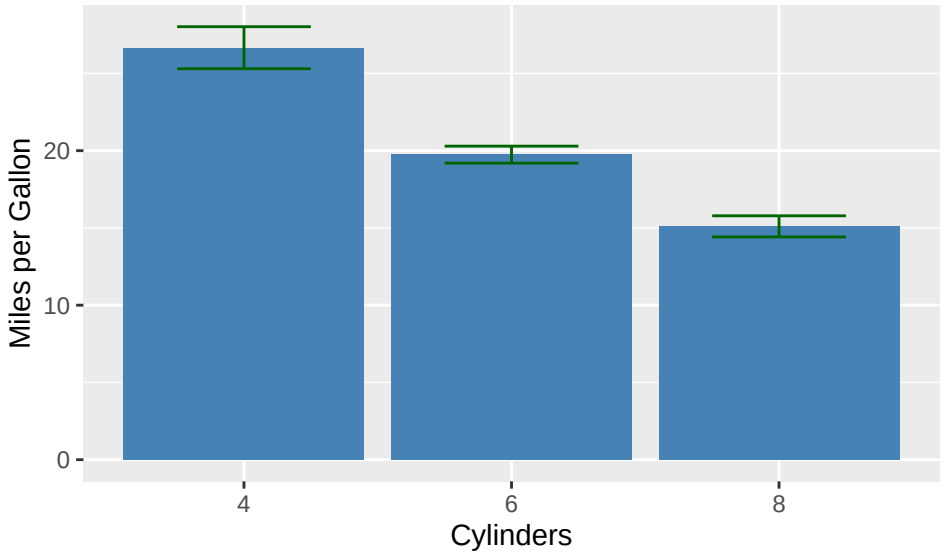
$$SEM = \frac{s}{\sqrt{n}}$$

```
df <- mtcars |>
    group_by(cyl) |>
    summarise(
        mean_mpg = mean(mpg, na.rm = TRUE),
        se_mpg = sd(mpg, na.rm = TRUE) / sqrt(n())
    )
df
```

```
# A tibble: 3 x 3
    cyl mean_mpg se_mpg
  <dbl>    <dbl>  <dbl>
1     4     26.7  1.36
2     6     19.7  0.549
3     8     15.1  0.684
```

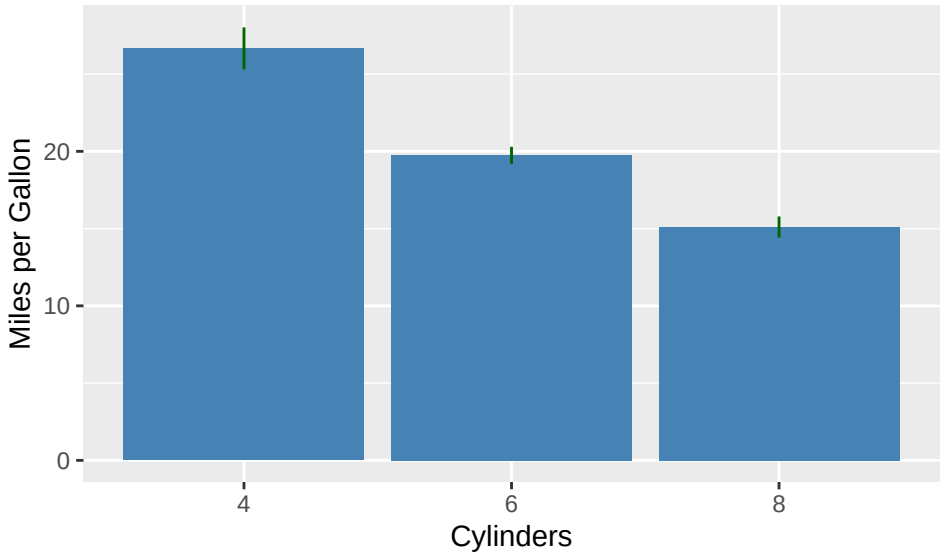
이 데이터프레임을 막대 그래프를 만들고, `geom_errorbar()`를 사용하여 에러 바를 추가한다.

```
ggplot(df, aes(factor(cyl), mean_mpg)) +
    geom_bar(stat = "identity", fill = "steelblue") +
    geom_errorbar(
        aes(ymin = mean_mpg - se_mpg, ymax = mean_mpg + se_mpg),
        width = 0.5,
        color = "darkgreen"
    ) +
    labs(x = "Cylinders", y = "Miles per Gallon")
```



다음은 `geom_linerange()`를 사용한 경우이다.

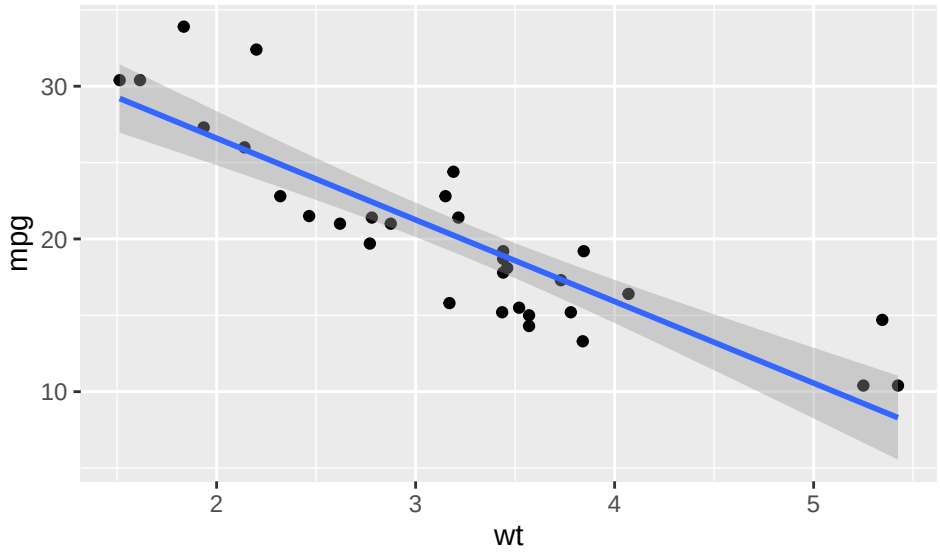
```
ggplot(df, aes(factor(cyl), mean_mpg)) +
    geom_bar(stat = "identity", fill = "steelblue") +
    geom_linerange(
        aes(ymin = mean_mpg - se_mpg, ymax = mean_mpg + se_mpg),
        color = "darkgreen"
    ) +
    labs(x = "Cylinders", y = "Miles per Gallon")
```



회귀 곡선을 그리는 경우에는 `geom_smooth()` 함수를 사용한다. 이 함수는 데이터가 1000개 미만인 경우에는 `loess` 방법을 사용하고, 1000개 이상인 경우에는 `lm` 방법을 사용한다. 95% 신뢰 구간을 표시하는데, 이것을 가리려면 `se = FALSE`를 지정한다.

```
ggplot(mtcars, aes(wt, mpg)) +
    geom_point() +
    geom_smooth(method = "lm")
```

```
`geom_smooth()` using formula = 'y ~ x'
```



23 팩터의 레벨에 따른 순서 조정(forcats 패키지를 중심으로)

5 장에서도 간단히 설명했지만, R에서는 팩터(factor)는 보통 범주형(categorical) 변수를 표현하는 특수한 종류의 벡터이다. 이런 팩터는 통계 분석에서 아주 중요한 역할을 한다.

ggplot2를 사용하여 그래프를 만들 때 가장 흔히 부딪히는 문제 중 하나는 팩터형(factor) 변수의 순서를 정하는 것이다. ggplot2에서는 팩터형(factor) 변수에 대한 지옴을 배치하는 순서는 팩터의 레벨(level)에 의해 결정되는 데 가끔 이것을 수정하고 싶을 때가 생긴다.

이 장에서는 팩터에 대하여 더 자세히 알아보고, 팩터를 다루는 데 아주 큰 도움을 주는 forcats 패키지를 설명하고, ggplot2에서 어떻게 또 활용되어 우리가 원하는 그래프를 얻을 수 있는지 설명한다.

23.1 팩터

팩터는 벡터의 한 종류로, 벡터의 각 요소가 취할 수 있는 값이 제한되어 있는 경우에 사용한다. 문자열

23.1.1 문자열 벡터와 팩터가 다른 점

예를 들어, 성별이라는 정보를 다음과 같이 정리했다. 모두 "female"이다.

```
gender <- c("female", "female", "female")
```

달(month)이라는 정보를 다음과 같이 정리했다.

```
months <- c("Jan", "Feb", "Feb", "Jan", "Mar", "Mar", "Mar", "Aug", "Sep", "Oct", "Apr", "Dec")
```

위 두 벡터는 아직 문자열 벡터이지 팩터가 아니다. 그래서 이것을 팩터가 바꾸지 않은 채로 분석하면 여러 가지 문제가 생길 수 있다. (Contigency) table을 만들면 다음과 같다.

```
table(gender)
```

```
gender
female
     3
```

```
table(months)
```

```
months
Apr Aug Dec Feb Jan Mar Oct Sep
  1   1   1   2   2   3   1   1
```

정렬을 해 보자.

```
sort(gender)
```

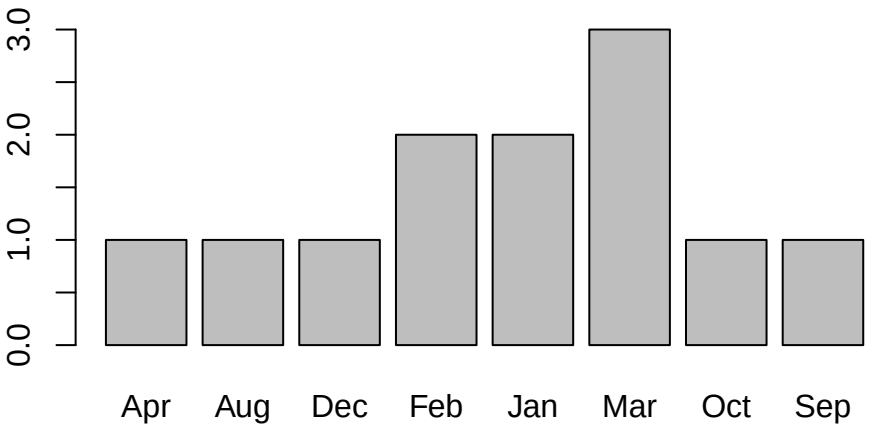
```
[1] "female" "female" "female"
```

```
sort(months)
```

```
 [1] "Apr" "Aug" "Dec" "Feb" "Feb" "Jan" "Jan" "Mar" "Mar" "Mar" "Oct" "Sep"
```

months 벡터를 가지고 막대그래프를 만들어 보자.

```
barplot(table(months))
```



`table()` 함수, `sort()` 함수, `barplot()` 함수 실행 결과를 보면 알파벳 순서대로 출력되는 것을 확인할 수 있다.

팩터는 **각 요소가 취할 수 있는 값이 제한되어 있는** 경우를 표현한다고 했다. 이 말을 성별 데이터를 모은 벡터에는 `"male"`과 `"female"`만 존재한다(제3의 성을 무시하면). 달 데이터는 "Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec" 중 하나만 존재할 것이다. 이것을 팩터의 **레벨(levels)**이라고 한다.

실제로 팩터는 각 레벨에 대하여 번호를 붙이고 내부에서 이 번호를 사용하여 데이터를 저장한다. 그래서 원래 문자열을 저장하는 것보다 메모리를 적게 사용한다. 그래서 "male"과 "female"을 레벨로 가지는 팩터는 "male"들은 1이고 "female"들은 2로 저장되고, 달을 레벨로 가지는 팩터는 "Jan"들은 1이고 "Feb"들은 2 등으로 저장한다.

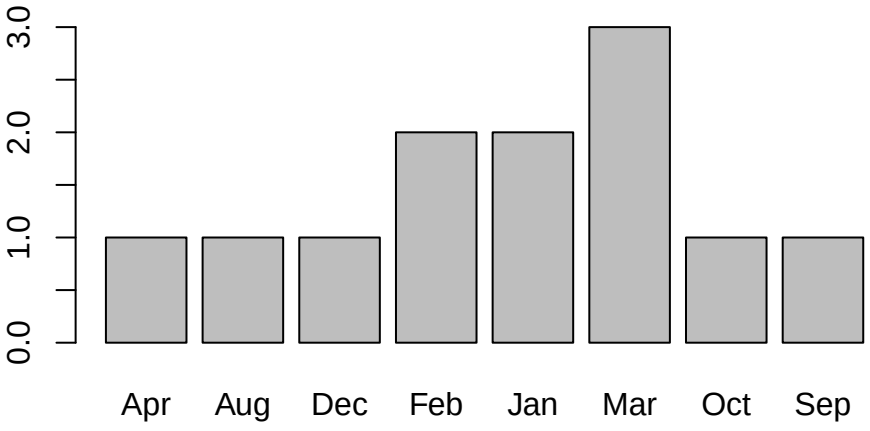
그리고 팩터를 출력했을 때는 숫자가 아니라 문자열로 출력된다. 이것은 팩터의 레이블(labels)이라고 한다.

문자열을 팩터로 변환하는 `factor()` 함수를 사용하여 레벨과 레이블을 지정할 수 있다. 그렇지만 `factor()` 함수를 디폴트 옵션으로 그대로 사용했을 때 그냥 원하는 팩터가 되는 것은 아니다.

```
mfactor <- factor(months)
mfactor
```

```
 [1] Jan Feb Feb Jan Mar Mar Mar Aug Sep Oct Apr Dec
Levels: Apr Aug Dec Feb Jan Mar Oct Sep
```

```
barplot(table(mfactor))
```



위 결과를 보면 레벨이 생성되기는 하지만, 레벨의 문자열 알파벳 순서대로 따라가는 것을 볼 수 있다.
이 문제를 해결하려면 추가 타이핑을 하는 수고가 필요하다.

```
gender_factor <- factor(gender,
    levels = c("male", "female"),
    labels = c("남자", "여자")
)
gender_factor
```

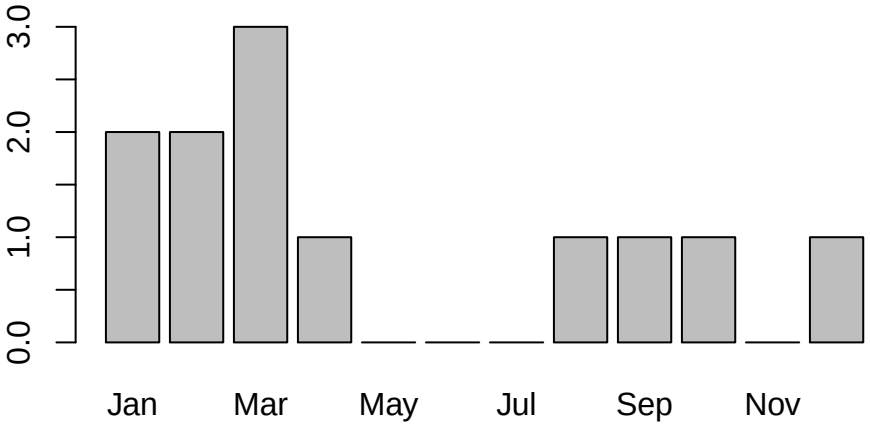
```
[1] 여자 여자 여자
Levels: 남자 여자
```

```
months_factor <- factor(months,
    levels = c("Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "De
    labels = c("Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "De
)
months_factor
```

```
 [1] Jan Feb Feb Jan Mar Mar Mar Aug Sep Oct Apr Dec
Levels: Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
```

이렇게 만들어진 `months_factor`를 사용하여 막대그래프를 만들면 다음과 같다.

```
barplot(table(months_factor))
```



결과를 보면 알파벳 순서대로가 아니라 레벨에 지정된 순서대로 출력되는 것을 확인할 수 있다.

정확한 팩터 데이터를 사용하는 문제는 순서형 팩터 데이터를 가지고 회귀분석을 할 때도 중요하다. 어찌되었든, 단순히 문자열 벡터를 팩터로 변환하는 것을 넘어서, 해당 데이터의 의도와 목적에 맞게 팩터를 만들어 사용하는 것이 중요하다.

23.2 ggplot2 그래프도 같은 로직을 따른다.

ggplot2 그래프도 베이스 R의 팩터 처리 로직을 그대로 따른다. 다음 사례를 보자.

```
library(ggplot2)
library(dplyr)
glimpse(penguins)
```

```
Rows: 344
Columns: 8
$ species     <fct> Adelie, Adelie, Adelie, Adelie, Adelie, Adelie, Adelie, Ad~
$ island      <fct> Torgersen, Torgersen, Torgersen, Torgersen, Torgersen, Tor~
$ bill_len    <dbl> 39.1, 39.5, 40.3, NA, 36.7, 39.3, 38.9, 39.2, 34.1, 42.0, ~
$ bill_dep    <dbl> 18.7, 17.4, 18.0, NA, 19.3, 20.6, 17.8, 19.6, 18.1, 20.2, ~
$ flipper_len <int> 181, 186, 195, NA, 193, 190, 181, 195, 193, 190, 186, 180,~
$ body_mass   <int> 3750, 3800, 3250, NA, 3450, 3650, 3625, 4675, 3475, 4250, ~
$ sex         <fct> male, female, female, NA, female, male, female, male, NA, ~
$ year        <int> 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007~
```

팩터의 레벨을 확인해 보자.

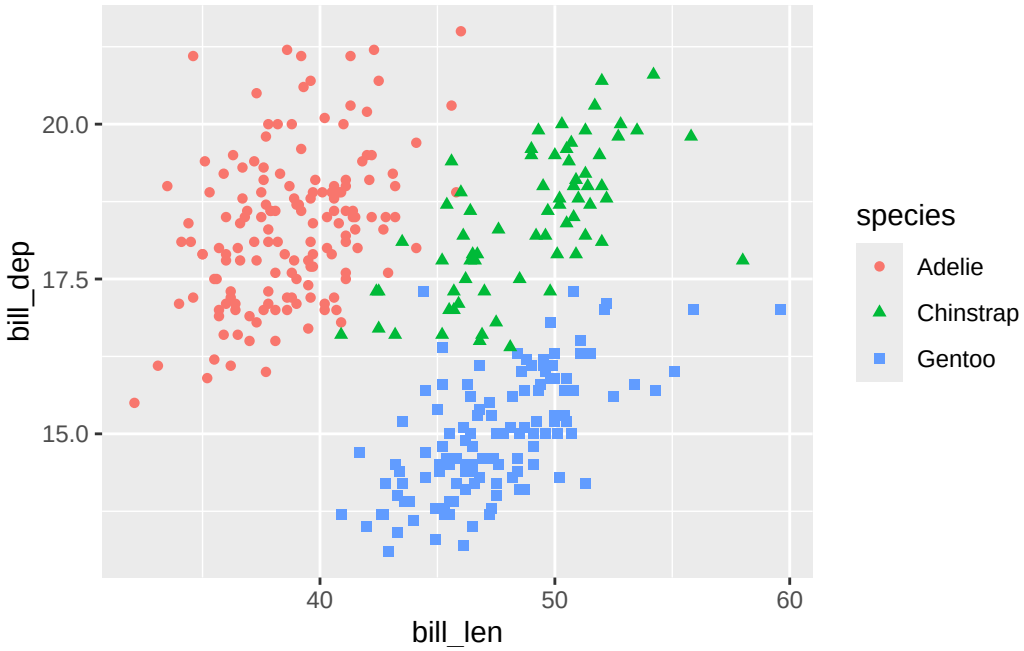
```
levels(penguins$species)
```

```
[1] "Adelie"    "Chinstrap" "Gentoo"
```

ggplot2 그래프를 하나 만들어 보자. 오른쪽 레전의 순서에 주목하자.

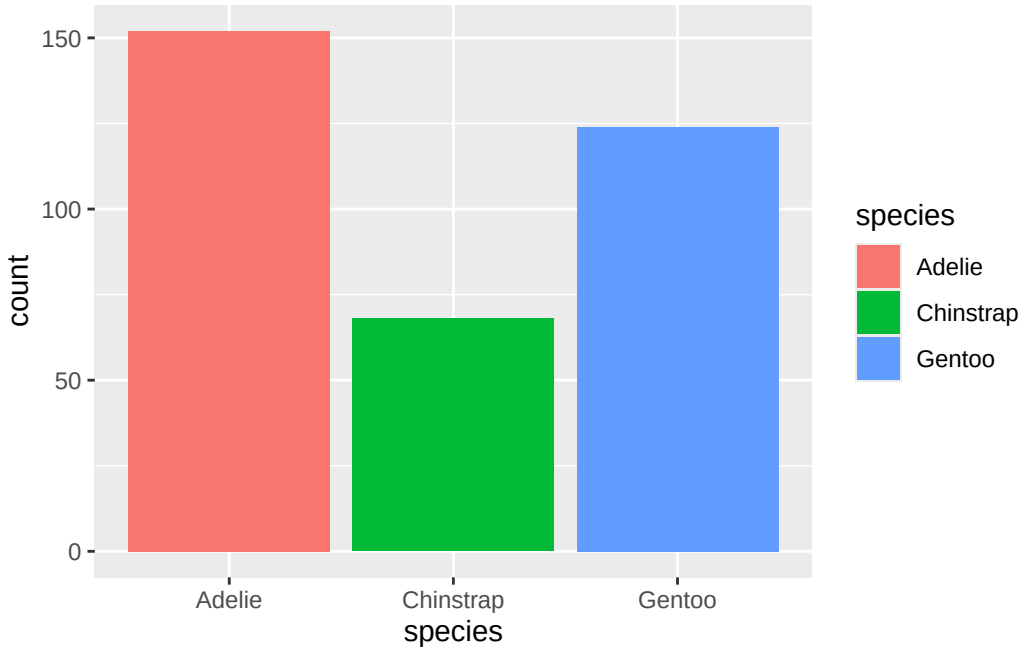
```
penguins %>%
    ggplot() +
    geom_point(aes(x = bill_len, y = bill_dep, color = species, shape = species))
```

```
Warning: Removed 2 rows containing missing values or values outside the scale range
(`geom_point()`).
```



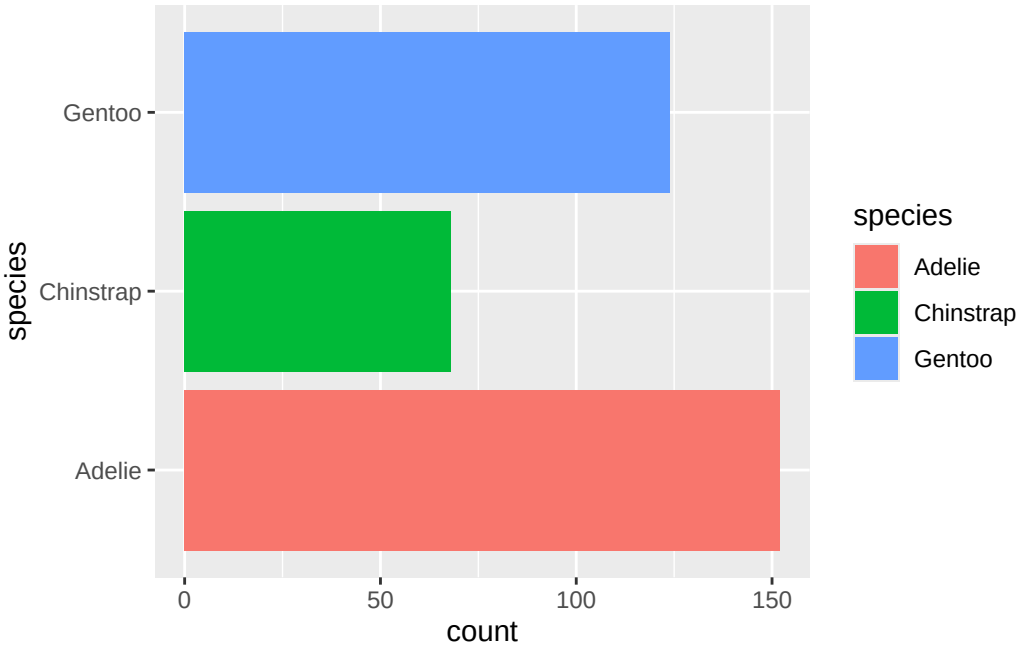
막대그래프를 만들어 보자. 역시 레벨 순서대로 표시된다.

```
penguins %>%
    ggplot() +
    geom_bar(aes(x = species, fill = species))
```



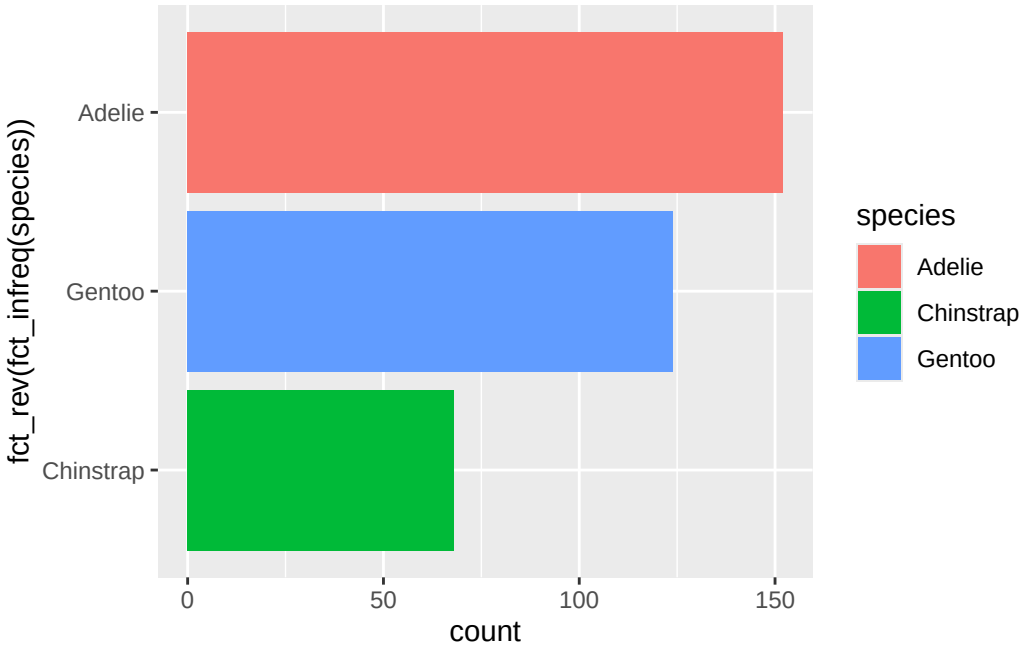
가로 막대그래프를 만들어 보자. 이 경우에는 아래에서 윗쪽으로 레벨의 순서에 따라 막대가 배치된다.

```
penguins %>%
    ggplot() +
    geom_bar(aes(x = species, fill = species)) +
    coord_flip()
```



카테고리의 레벨이 많은 경우에는 카운트를 정렬해 주는 것이 좋을 것이다. 이렇게 하려면 팩터의 레벨을 조정해 주어야 한다. 이럴 때 큰 도움을 주는 것이 `forcats` 패키지이다. 다음 절에서 자세히 설명한다.

```
library(forcats)
penguins %>%
    ggplot() +
    geom_bar(aes(x = fct_rev(fct_infreq(species)), fill = species)) +
    coord_flip()
```



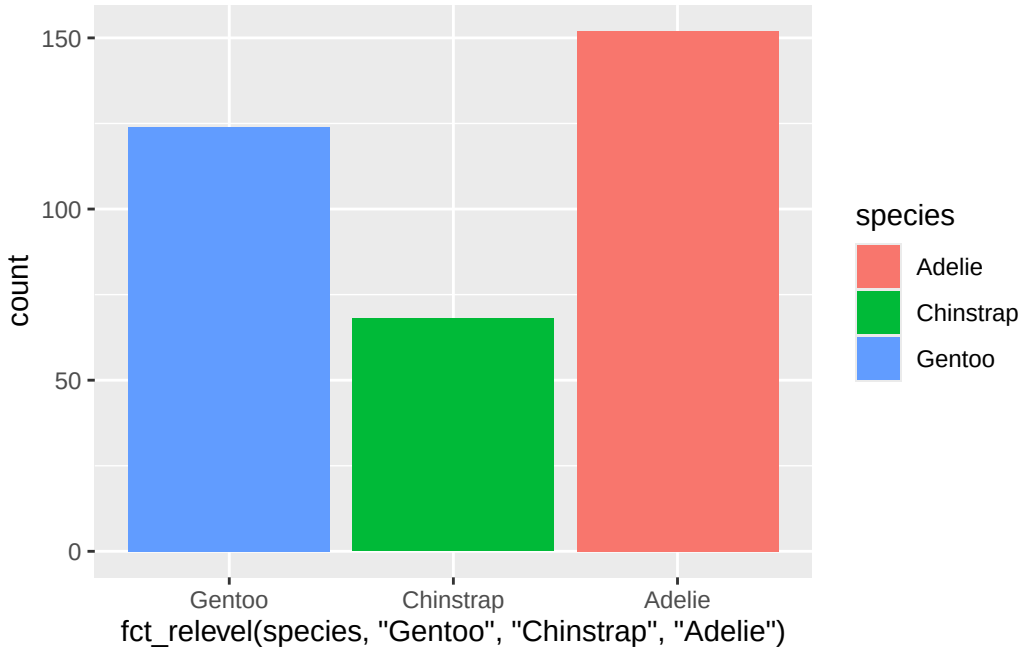
23.3 `forcats` 패키지로 팩터/그래프 조정

`forcats` 패키지는 팩터를 다루는 다양한 함수를 제공한다.

```
library(forcats)
```

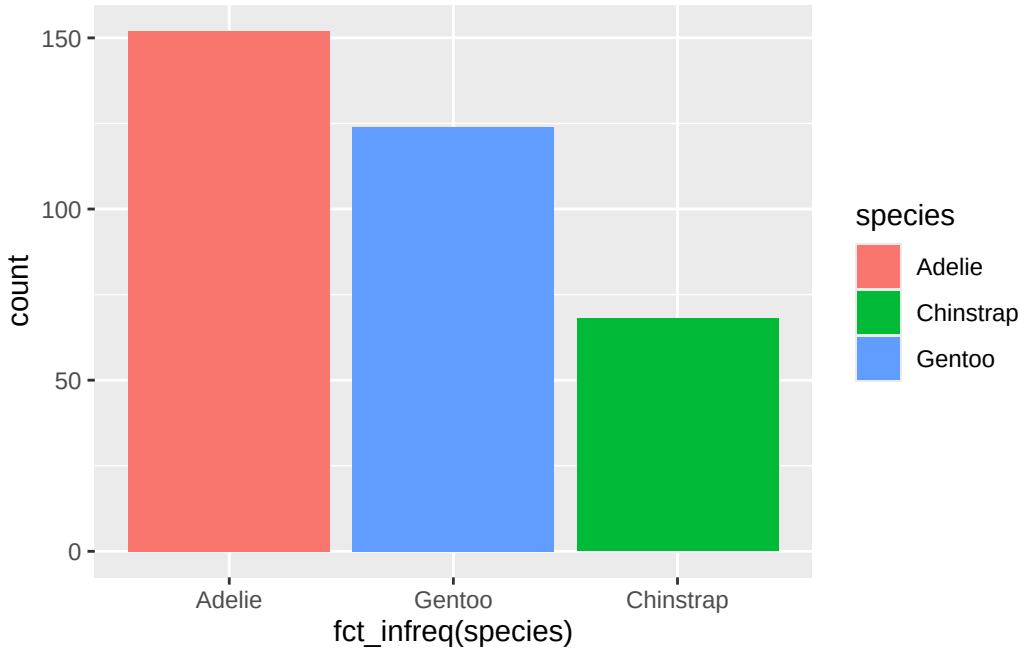
- `forcats` 패키지 함수 가운데 `fct_relevel()` 함수는 사용자가 지정한 순서대로 레벨이 정렬된 새로운 팩터를 만들어준다. 따라서 이 함수를 사용하면 순서를 사용자가 원하는 대로 정할 수 있다.

```
penguins %>%
    ggplot() +
    geom_bar(aes(x = fct_relevel(species, "Gentoo", "Chinstrap", "Adelie"), fill = species))
```



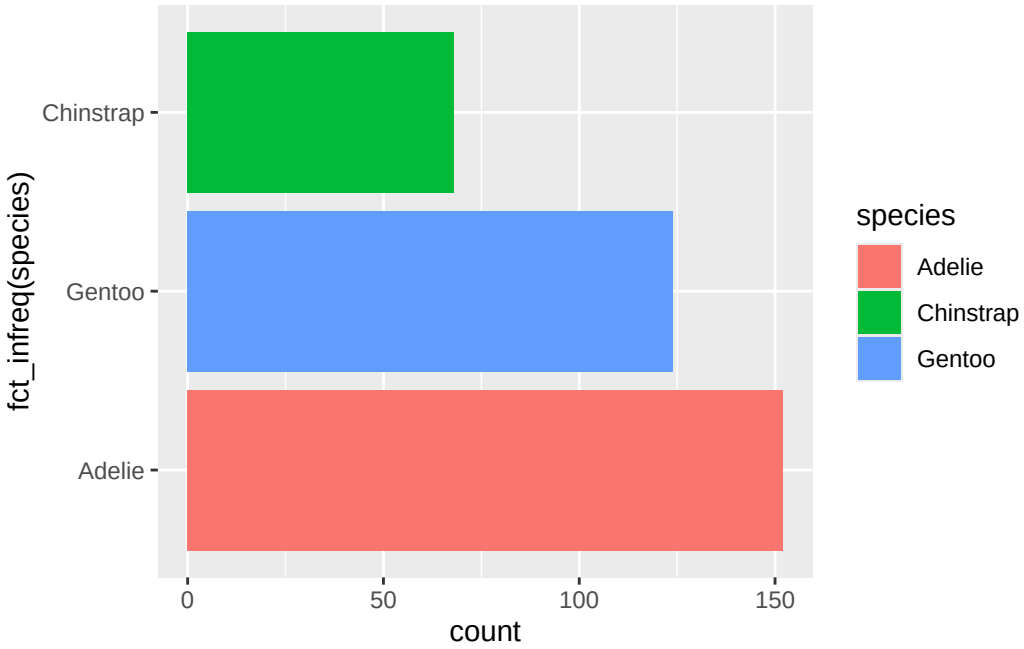
- `fct_infreq()` 함수는 레벨의 빈도수에 따라 정렬한다. 따라서 이 함수를 사용하면 빈도수가 높은 레벨이 앞에 오게 된다.

```
penguins %>%
    ggplot() +
    geom_bar(aes(x = fct_infreq(species), fill = species))
```



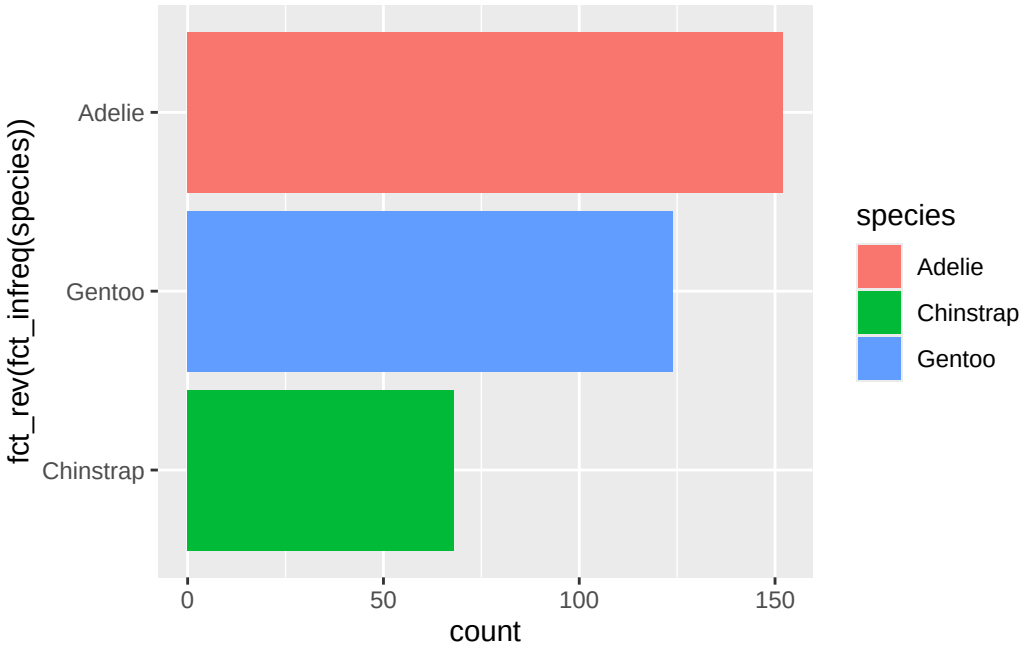
수평 막대그래프에서는 레벨의 빈도가 가장 높은 것이 아래에 있고, 빈도가 가장 낮은 것이 위에 놓인다.

```
penguins %>%
    ggplot() +
    geom_bar(aes(x = fct_infreq(species), fill = species)) +
    coord_flip()
```



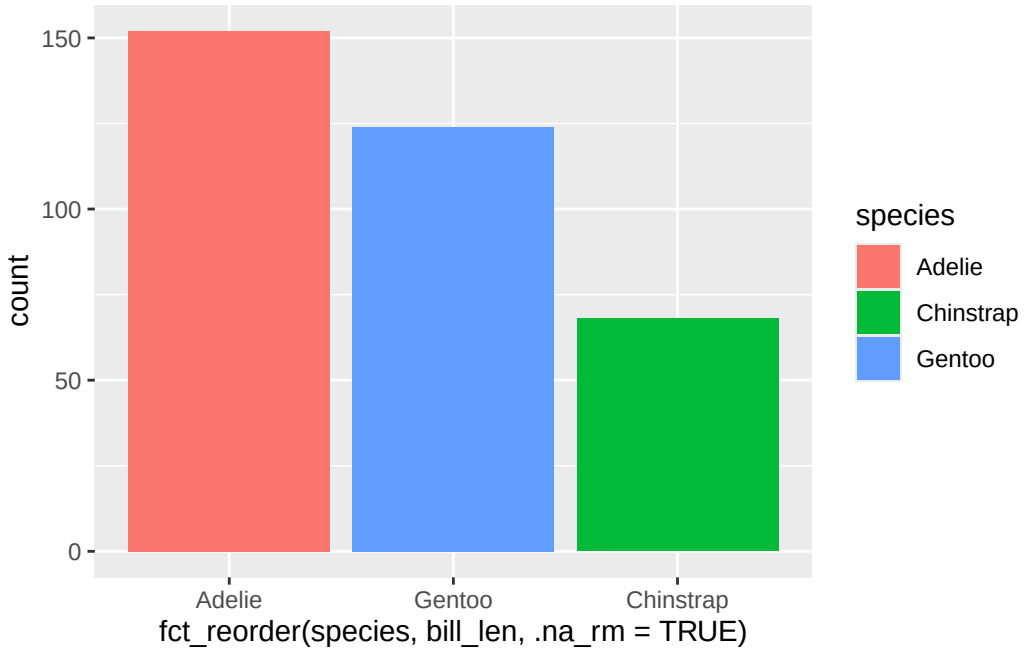
- `fct_rev()` 함수는 레벨의 순서를 반대로 뒤집는다. 따라서 위의 수평 막대그래프를 다음과 같이 빈도가 높은 것이 위로 가게 하려면 다음과 같이 하면 된다.

```
penguins %>%
    ggplot() +
    geom_bar(aes(x = fct_rev(fct_infreq(species)), fill = species)) +
    coord_flip()
```



- `fct_reorder()` 함수는 특정 변수에 따라 레벨을 정렬하는데, 드폴트는 중앙값(median)을 기준으로 한다. 예를 들어, 다음과 같이 레벨을 정렬할 수 있다.

```
penguins %>%
    ggplot() +
    geom_bar(aes(
        x = fct_reorder(species, bill_len, .na_rm = TRUE),
        fill = species
    ))
```



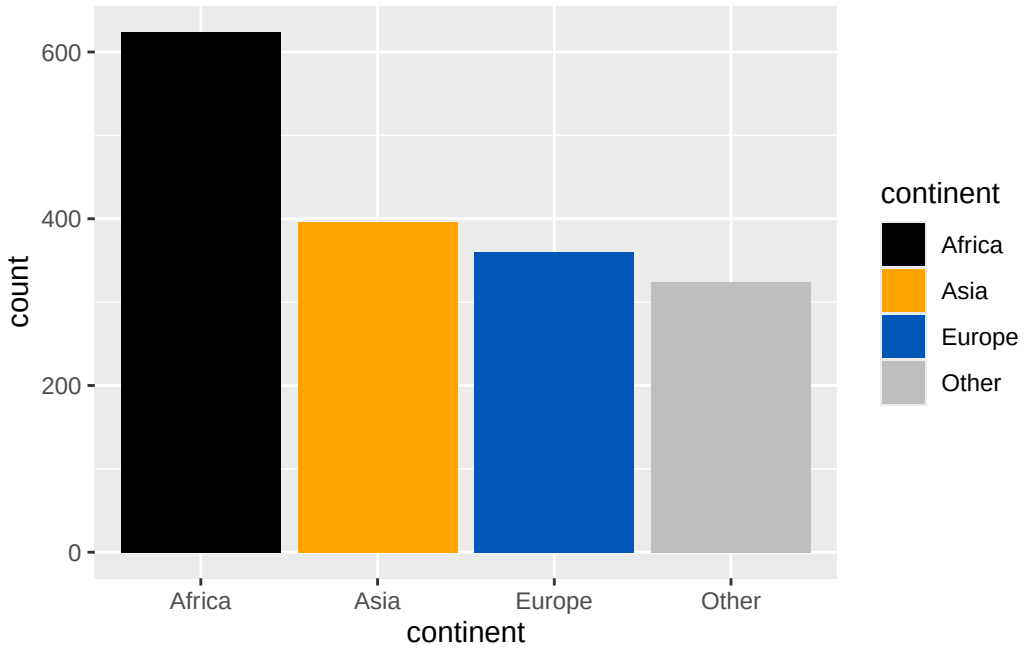
- `fct_lump_*()` 함수는 여러 기준에 따라서 빈도가 낮은 레벨을 묶어서 "기타"로 묶는다.
    - `fct_lump_n()`: n개의 가장 흔한 레벨을 제외하고 나머지를 묶음
    - `fct_lump_prop()`: 어떤 비율(proportion) 이하의 것들을 기타로 묶음
    - `fct_lump_min()`: 어떤 빈도수 이하의 것들을 기타로 묶음
    - `fct_lump_freq()`: 어떤 빈도수 이하의 것들을 기타로 묶음

```
library(gapminder)
glimpse(gapminder)
```

```
Rows: 1,704
Columns: 6
$ country   <fct> "Afghanistan", "Afghanistan", "Afghanistan", "Afghanistan", ~
$ continent <fct> Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, ~
$ year      <int> 1952, 1957, 1962, 1967, 1972, 1977, 1982, 1987, 1992, 1997, ~
$ lifeExp   <dbl> 28.801, 30.332, 31.997, 34.020, 36.088, 38.438, 39.854, 40.8~
$ pop       <int> 8425333, 9240934, 10267083, 11537966, 13079460, 14880372, 12~
$ gdpPercap <dbl> 779.4453, 820.8530, 853.1007, 836.1971, 739.9811, 786.1134, ~
```

이 데이터에서 대륙별 데이터를 묶어서 기타로 묶어 보자.

```
# 오륜기 색을 사용
gapminder %>%
    mutate(continent = fct_lump_n(continent, n = 3)) %>%
    ggplot() +
    geom_bar(aes(x = continent, fill = continent)) +
    scale_fill_manual(
        values = c(
            "Africa" = "#000000", "Americas" = "steelblue",
            "Asia" = "#FFA300", "Europe" = "#0057B8",
            "Oceania" = "#009B3A", "Other" = "grey"
        )
    )
```



23.4 정리하기

24 그래프 테마(theme) 커스터마이징

ggplot2 패키지로 그래프를 만들 때, 데이터와 연관된 프로세스가 끝나면 테마 등 주변의 요소들을 조정하는 작업을 하게 된다. 여기선 이 주변 요소들을 원하는 바대로 조정하는 작업을 설명한다. ggplot2 패키지 자체에서 제공하는 기능도 있고, 여러 목적으로 다양한 확장 패키지들도 많이 개발되어 있어 적절한 곳에서 소개하고자 한다.

24.1 (한글 포함) 글자

ggplot2 패키지로 그래프를 만들어 논문을 투고한다고 생각해 보자. 그러면 저널 출판사에서 그래프와 관계된 문자의 크기, 폰트 등을 지정해 줄 것이고 그대로 그래프를 만들어야 할 것이다. 이 문제를 해결하는 방법을 설명하고자 한다.

24.1.1 showtext 패키지를 사용하여 한글을 포함한 폰트 패밀리 지정

showtext 패키지는 한글을 포함한 폰트를 지정하는 데 사용되는 패키지이다. 과정은 다음과 같다.

1. showtext 패키지를 설치하고 로드한다.
2. 폰트를 선택한다.
    - 현재 컴퓨터에 있는 폰트 사용: font_add() 함수를 사용하여 현재 컴퓨터에 설치된 폰트 파일을 지정하고, 현재 R 세션에서 이 폰트에 대한 이름을 붙인다.
    - 구글 폰트 사용: font_add_google() 함수를 사용하여 인터넷에 있는 구글 폰트에서 사용할 폰트를 선택하고, 현재 R 세션에서 이 폰트에 대한 이름을 붙인다.
3. showtext_auto() 함수를 사용하여 현재 R 세션에 이 폰트를 사용하도록 지시한다.
4. ggplot2 패키지(또는 base R 그래프 함수로) 그래프를 만들 때 이 폰트를 사용한다.

```
library(ggplot2)
library(showtext)
```

```
Loading required package: sysfonts
```

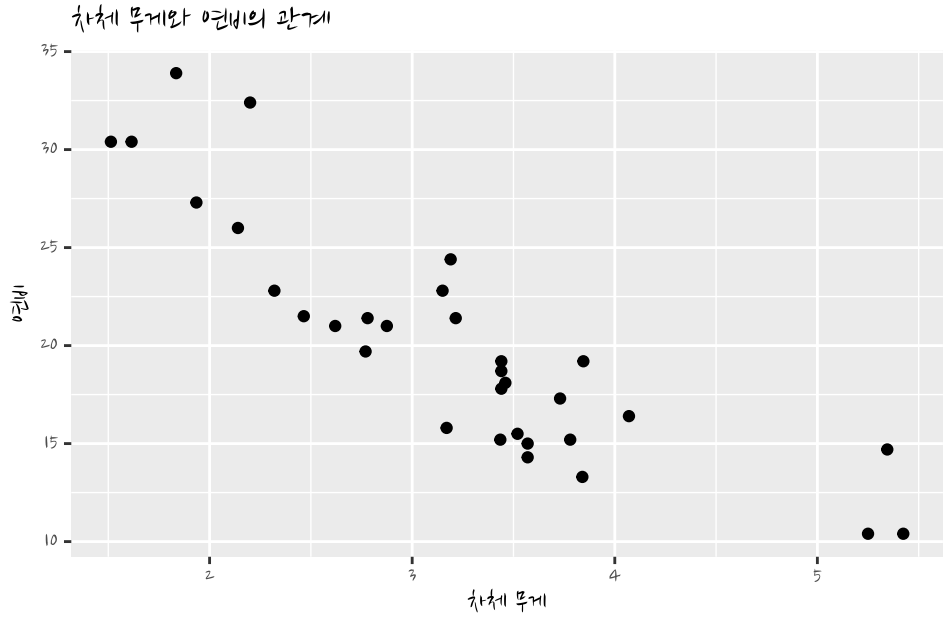
```
Loading required package: showtextdb
```

다음은 구글 폰트에서 사용할 폰트(Nanum Brush Script)를 선택하고, 현재 R 세션에서 이 폰트에 대한 이름을 붙인 예이다. 두 번째 인자에 준 이름으로 이후 R 코드에서 이 폰트를 지목하여 사용하게 된다. `showtext_auto()` 함수는 현재 R 세션에 이 폰트를 사용하도록 지시한다.

```
font_add_google("Nanum Brush Script", "my-nbs")
showtext_auto()
```

이제 이 폰트를 사용하여 그래프를 만든다. `theme()` 함수에서 `text` 요소에서 `family` 인자에서 앞에서 지정한 폰트 이름을 지정함으로써 이 그래프의 텍스트에 사용할 폰트가 정해진다.

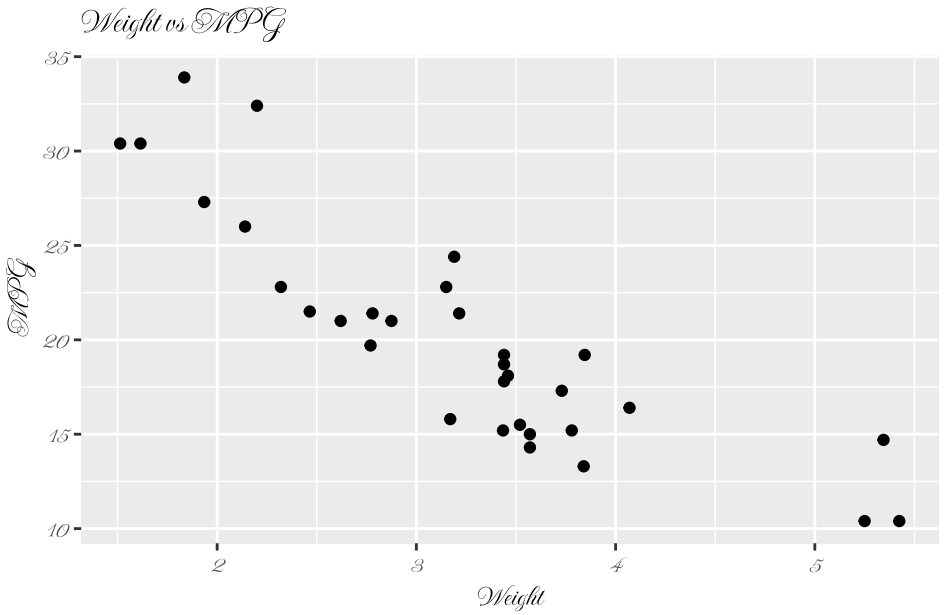
```
ggplot(mtcars, aes(x = wt, y = mpg)) +
    geom_point() +
    labs(
        title = "차체 무게와 연비의 관계",
        x = "차체 무게",
        y = "연비"
    ) +
    theme(text = element_text(family = "my-nbs"))
```



만약 구글 폰트의 하나인 “Imperial Script”를 사용하고 싶다면 다음과 같이 할 수 있을 것이다.

```
font_add_google("Imperial Script", "my-imperial")
showtext_auto()
```

```
ggplot(mtcars, aes(x = wt, y = mpg)) +
    geom_point() +
    labs(
        title = "Weight vs MPG",
        x = "Weight",
        y = "MPG"
    ) +
    theme(text = element_text(family = "my-imperial"))
```



24.1.2 theme() 함수 안에서 텍스트 요소와 그 값을 지정

앞에서 theme() 함수 안에서 text 요소에 대한 폰트 패밀리를 지정하였다. 이 외에도 theme() 함수 안에서 다양한 텍스트 요소에 대한 폰트 패밀리를 지정할 수 있다.

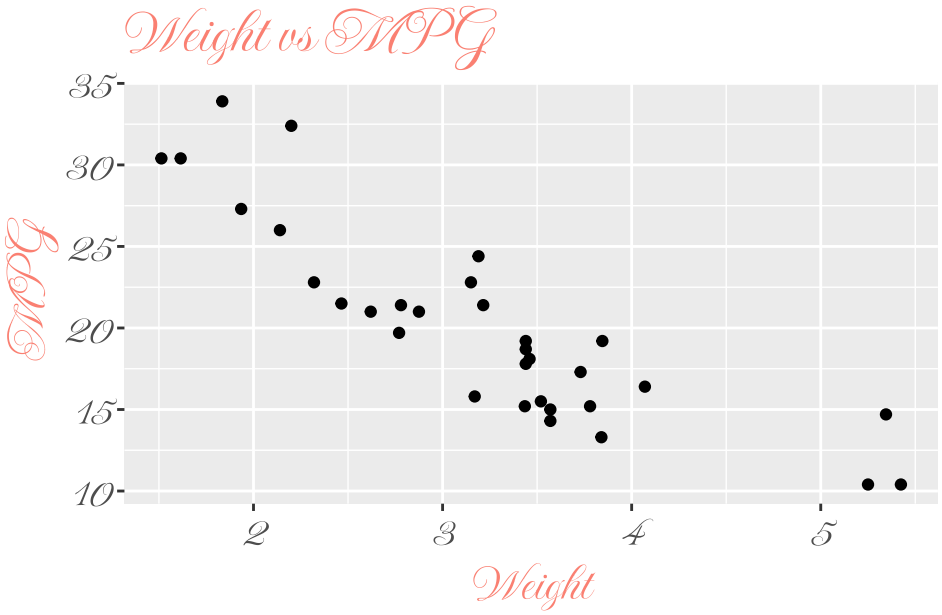
폰트 뿐만 아니라 그래프의 테마와 관련된 것들은 모두 theme() 함수 안에서 지정할 수 있다. 대체로 다음과 같은 문법을 따른다.

```
theme(요소이름 = element_*(속성 = 값))
```

- 요소 이름은 그래프 요소의 이름이다. 예를 들어, text 요소는 텍스트 요소이다. 이들 요소 이름들은 위계 구조를 가지고 있어 예를 들어 text는 전체 텍스트를 의미하고, axis.text는 axis의 텍스트를 의미한다.
- 해당 요소의 값은 element_*() 함수를 사용한다. 예를 들어, text 요소의 값은 element_text() 함수를 사용한다. 이 안에 해당 요소의 속성을 지정한다. 어떤 속성들을 지정할 수 있는지는 ?element_text와 같이 도움말에 상세히 나와 있다.

위에서 만들어 보았던 그래프를 예시로 폰트, 크기, 페이스, 색상을 element_text() 함수한 예이다.

```r
ggplot(mtcars, aes(x = wt, y = mpg)) +
    geom_point() +
    labs(
        title = "Weight vs MPG",
        x = "Weight",
        y = "MPG"
    ) +
    theme(
        text = element_text(
            family = "my-imperial",
            size = 20,
            face = "bold",
            color = "salmon"
        )
    )
```

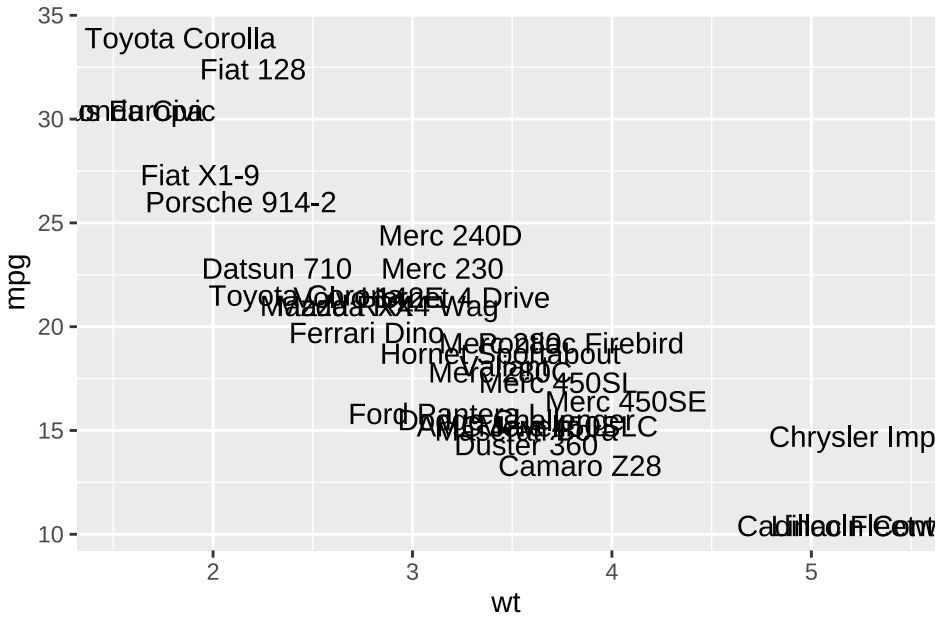


24.1.3 그래프에 텍스트 삽입

24.1.3.1 `geom_text()`와 `geom_label()` 함수

`ggplot2` 패키지의 `geom_text()`와 `geom_label()` 함수는 그래프에 텍스트를 삽입하는 데 사용된다. 이 함수는 `label` 매핑을 사용할 수 있고, 여기에 값(변수)를 매핑한다.

```
ggplot(mtcars, aes(x = wt, y = mpg)) +
    geom_text(
        aes(label = rownames(mtcars))
    )
```

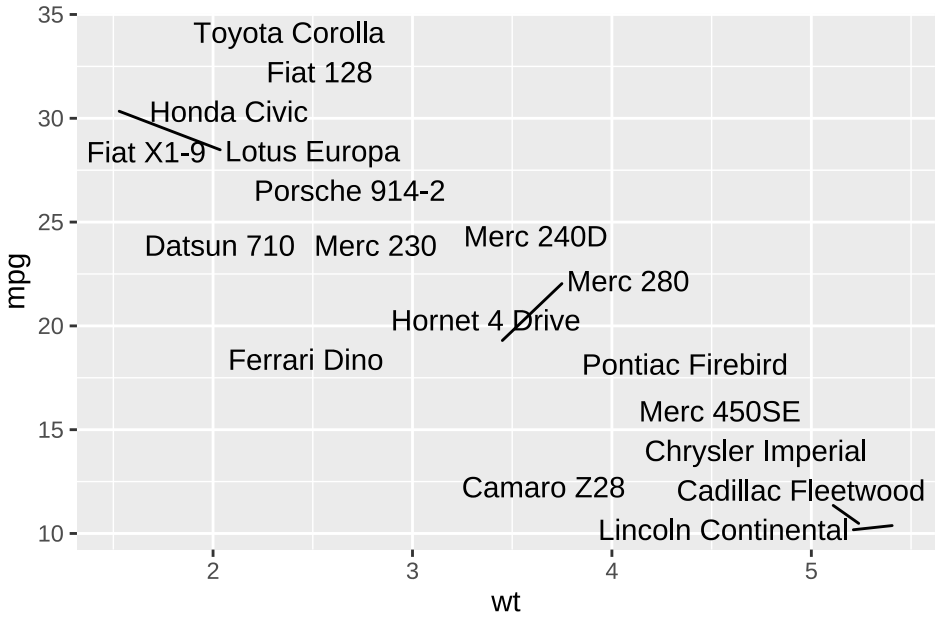


텍스트들이 겹쳐 있는 경우 텍스트들이 겹치는 것을 방지하기 위해 `ggrepel` 패키지의 `geom_text_repel()` 함수를 사용할 수 있다.

```
library(ggrepel)
ggplot(mtcars, aes(x = wt, y = mpg)) +
    geom_text_repel(
        aes(label = rownames(mtcars))
    )
```

```
Warning: ggrepel: 14 unlabeled data points (too many overlaps). Consider
```

increasing max.overlaps

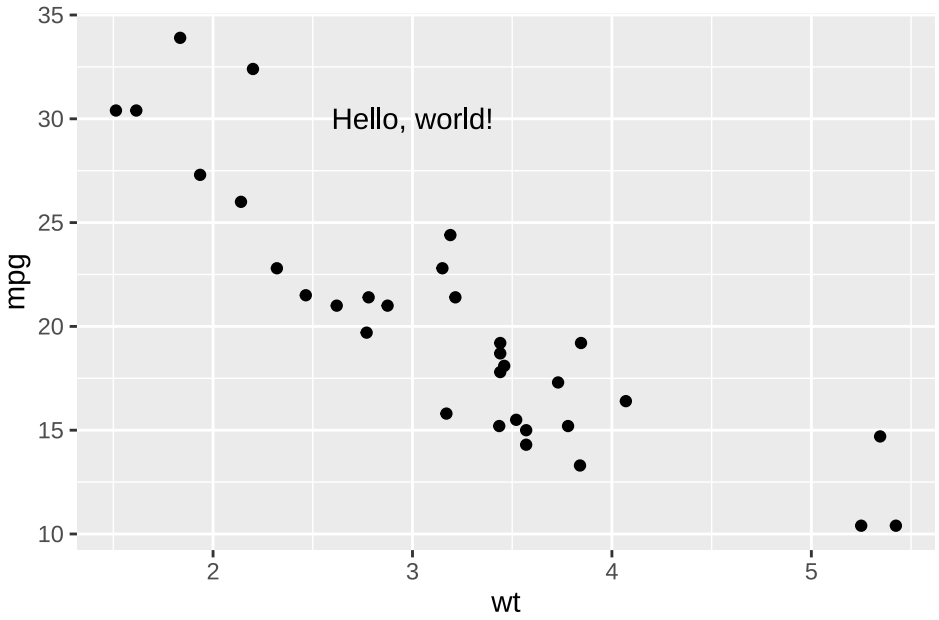


24.1.3.2 annotate() 주석 추가 함수

annotate() 함수는 텍스트를 포함한 여러 형태의 주석을 간단하게 추가할 수 있는 기능을 제공한다. 앞의 geom_text()와 geom_label() 함수는 데이터 프로세스를 거치기 때문에 다소 번잡해 보일 수 있다. 그렇지만 annotate() 함수는 이런 번잡함이 없어서 편리하다.

첫 번째 인자는 "text"인 것은 이것이 텍스트 형태의 주석이라는 것을 말한다. 다른 형태의 주석은 ?annotate 도움말을 참고한다. 위치는 축의 값으로 결정된다.

```
ggplot(mtcars, aes(x = wt, y = mpg)) +
    geom_point() +
    annotate("text",
        x = 3, y = 30,
        label = "Hello, world!"
    )
```



24.1.3.3 ggtext 패키지

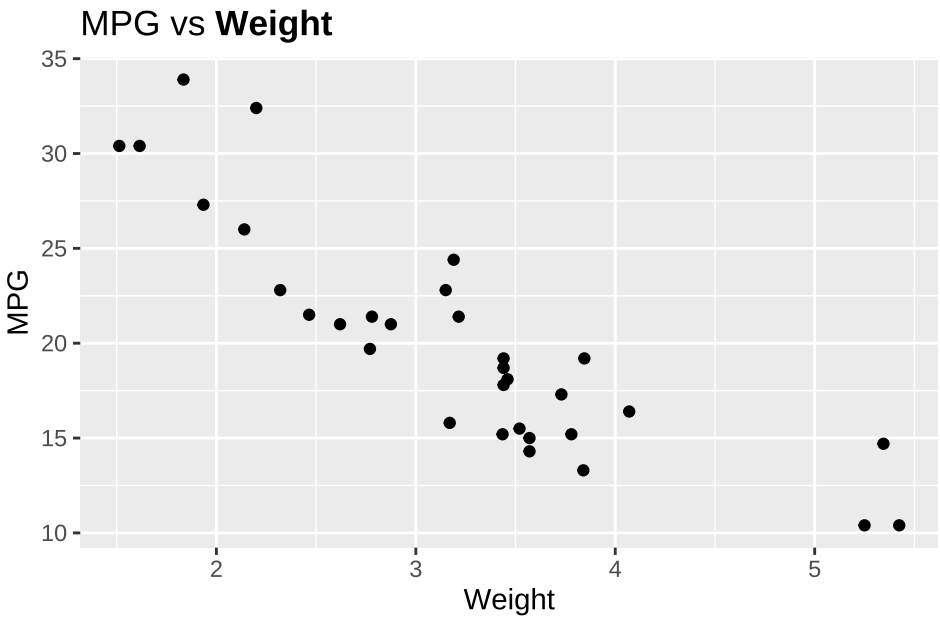
ggtext 패키지는 그래프를 텍스트를 넣을 수도 있고, 텍스트를 Markdown/HTML 형식으로 작성하여 그래프에 삽입할 수 있게 해 준다. 다음과 같은 핵심 함수를 가지고 있다.

- element_markdown() 함수는 theme() 함수에서 텍스트 요소를 Markdown/HTML 형식으로 작성할 수 있게 해 준다.
- geom_richtext() 함수는 텍스트를 Markdown/HTML 형식으로 작성하여 그래프에 삽입할 수 있게 해 준다.
- geom_textbox() 함수는 텍스트를 가진 박스를 그래프에 삽입할 수 있다.

다음은 element_markdown() 함수를 theme() 함수에서 사용하기 때문에 plot.title에 마크다운 문법으로 제목을 지정해 줄 수 있다.

```
library(ggtext)

ggplot(mtcars, aes(x = wt, y = mpg)) +
    geom_point() +
    labs(
        title = "MPG vs **Weight**",
        x = "Weight",
```

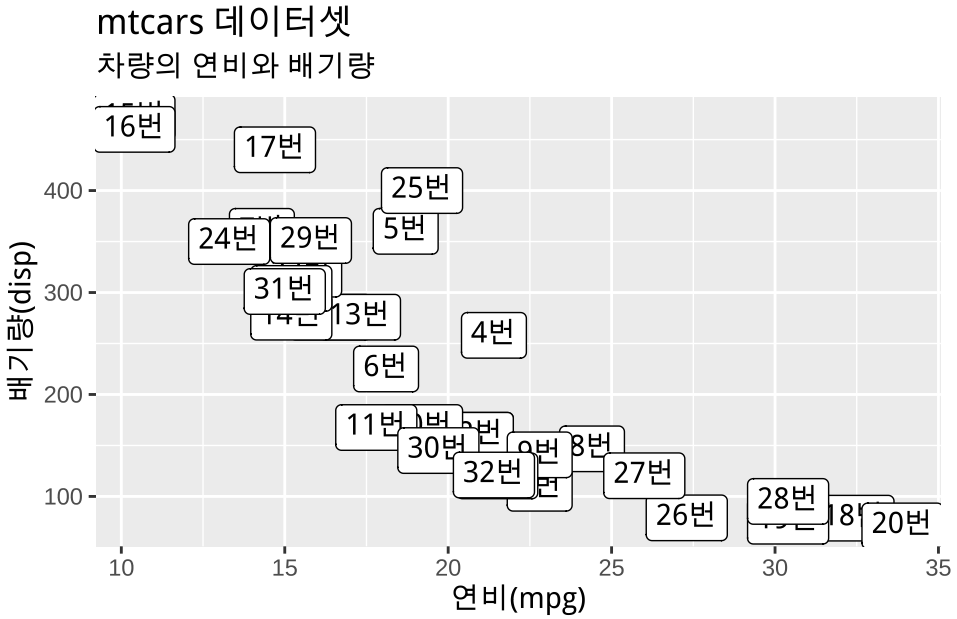
```
        y = "MPG"
    ) +
    theme(
        plot.title = element_markdown()
    )
```



`geom_richtext()` 함수는 데이터로 들어가 있는 텍스트(Markdown/HTML 형식으로 작성 가능)를 그래프에 삽입할 수 있게 해 준다. `geom_richtext()` 지옴에서 핵심적으로 사용되는 매핑이 `label`이다. 다음은 일부러 `mtcars` 데이터셋의 행 번호를 가지고 텍스트 데이터를 만들고 이것을 `label`이라는 새로운 열로 만들고, 이것을 `geom_richtext()` 지옴에서 사용하는 예이다.

```
library(dplyr)
library(stringr)
df <- mtcars |>
    mutate(
        label = str_glue("{row_number()}번")
    )

ggplot(df, aes(x = mpg, y = disp)) +
    geom_richtext(aes(label = label)) +
```

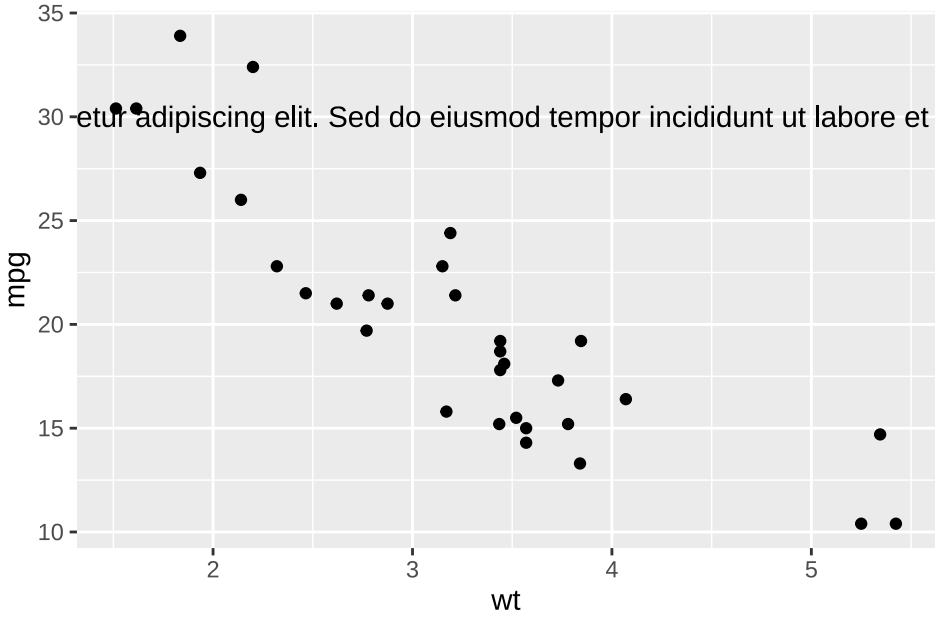
```
    labs(
        title = "mtcars 데이터셋",
        subtitle = "차량의 연비와 배기량",
        x = "연비(mpg)",
        y = "배기량(disp)"
    )
```



`geom_textbox()` 함수는 줄바꿈이 되는 텍스트 상자를 그래프에 삽입할 수 있게 해 준다.

다음은 단순한 `annotate()` 함수를 사용한 예이다.

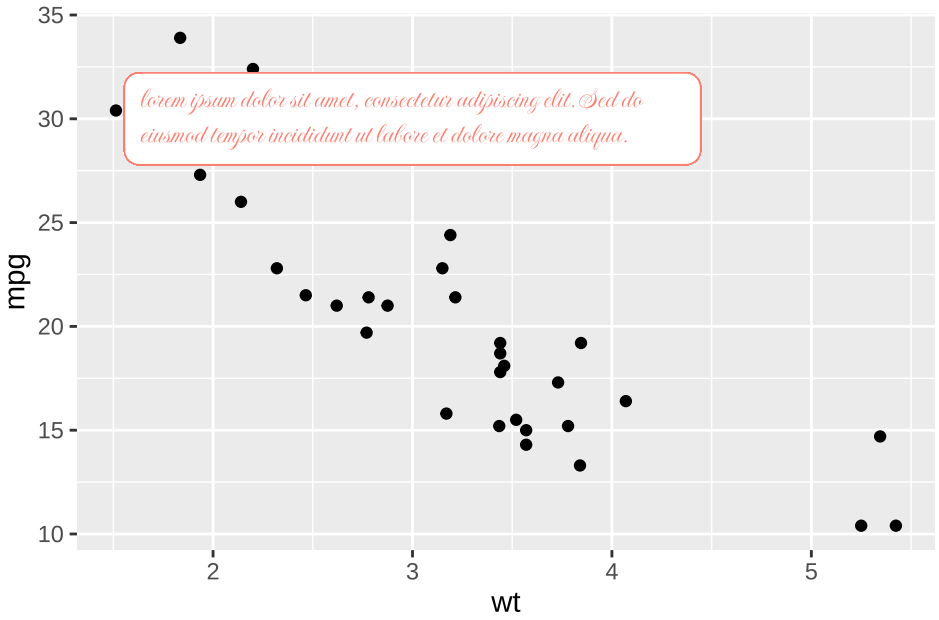
```
ggplot(mtcars, aes(x = wt, y = mpg)) +
    geom_point() +
    annotate("text",
        x = 3, y = 30,
        label = "lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed do eiusmod tempor
    )
```



다음은 `geom_textbox()` 함수를 사용한 예이다. 경고문이 뜨기는 하지만 무시해도 된다. 물론 앞의 `geom_richtext()` 함수와 비슷하게 Markdown/HTML 형식으로 작성할 수 있다.

```
ggplot(mtcars, aes(x = wt, y = mpg)) +
    geom_point() +
    geom_textbox(
        aes(label = "lorem ipsum dolor **sit amet**, consectetur adipiscing elit. Sed do eiusmo
        x = 3, y = 30,
        width = unit(3, "inch"),
        family = "my-imperial",
        color = "salmon"
    )
```

```
Warning in geom_textbox(aes(label = "lorem ipsum dolor **sit amet**, consectetur adipiscing elit.
i Please consider using `annotate()` or provide this layer with data containing
  a single row.
```



24.1.4 `theme()` 함수의 텍스트 관련 인자들(with AI)

앞에서도 소개했지만 여기서 `theme()` 함수에서 텍스트 요소를 커스터마이징하는 주요 인자들을 정리해 보면 표 24.1 표와 같다.

표 24.1: 텍스트를 제어하기 위한 `theme()` 함수의 인자들

요소	설명	주요 속성	예시
`text`	모든 텍스트의 기본 스타일	`family`, `size`, `face`, `color`, `angle`, `hjust`, `vjust`	`element_text(family = "Arial", size = 12)`
`plot.title`	그래프 제목	`size`, `face`, `color`, `hjust`, `margin`	`element_text(size = 16, face = "bold", hjust = 0.5)`
`plot.subtitle`	그래프 부제목	`size`, `face`, `color`, `hjust`, `margin`	`element_text(size = 14, color = "gray50")`

요소	설명	주요 속성	예시
plot.caption	그래프 캡션	size, face, color, hjust	element_text(size = 10, hjust = 1)
axis.title	축 제목	size, face, color, angle	element_text(size = 12, angle = 0)
axis.title.x	x축 제목	size, face, color, angle, margin	element_text(size = 12, margin = margin(t = 10))
axis.title.y	y축 제목	size, face, color, angle, margin	element_text(size = 12, angle = 90)
axis.text	축 텍스트	size, face, color, angle	element_text(size = 10, color = "black")
axis.text.x	x축 텍스트	size, face, color, angle, hjust, vjust	element_text(angle = 45, hjust = 1)
axis.text.y	y축 텍스트	size, face, color, angle, hjust, vjust	element_text(hjust = 1)
legend.title	범례 제목	size, face, color, hjust	element_text(size = 12, face = "bold")
legend.text	범례 텍스트	size, face, color, hjust	element_text(size = 10)
strip.text	facet 텍스트	size, face, color, margin	element_text(size = 12, face = "bold")
strip.text.x	가로 facet 텍스트	size, face, color, margin	element_text(size = 12)

strip.text.y	세로 facet 텍스트	size, face, color, margin, angle	element_text(size = 12, angle = 0)

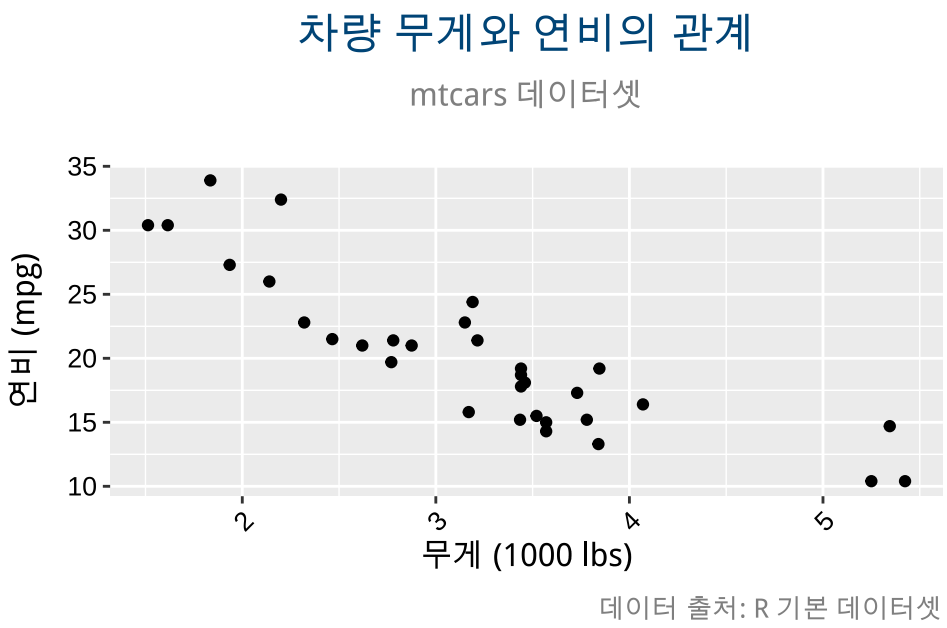
각 텍스트 요소는 element_text() 함수를 사용하여 스타일을 지정할 수 있으며, 다음과 같은 주요 속성들을 사용할 수 있다:

- family: 폰트 패밀리 (예: “Arial”, “Times New Roman”)
- size: 텍스트 크기 (숫자 또는 rel() 함수 사용)
- face: 폰트 스타일 (“plain”, “bold”, “italic”, “bold.italic”)
- color: 텍스트 색상
- angle: 텍스트 회전 각도 (0-360)
- hjust: 수평 정렬 (0-1, 0=왼쪽, 1=오른쪽)
- vjust: 수직 정렬 (0-1, 0=아래, 1=위)
- margin: 여백 (margin() 함수 사용)

구체적인 사용법은 아래 코드를 참고한다.

```
# 예시: 텍스트 테마 적용하기
ggplot(mtcars, aes(x = wt, y = mpg)) +
    geom_point() +
    labs(
        title = "차량 무게와 연비의 관계",
        subtitle = "mtcars 데이터셋",
        caption = "데이터 출처: R 기본 데이터셋",
        x = "무게 (1000 lbs)",
        y = "연비 (mpg)"
    ) +
    theme(
        # 제목 관련
        plot.title = element_text(
            size = 16,
            face = "bold",
            color = "#004476",
            hjust = 0.5,
```

```
        margin = margin(b = 10)
    ),
    plot.subtitle = element_text(
        size = 12,
        color = "gray50",
        hjust = 0.5,
        margin = margin(b = 20)
    ),
    plot.caption = element_text(
        size = 10,
        color = "gray50",
        hjust = 1,
        margin = margin(t = 10)
    ),
    # 축 제목
    axis.title = element_text(
        size = 12,
        face = "bold"
    ),
    axis.title.y = element_text(
        angle = 90,
        margin = margin(r = 10)
    ),
    # 축 텍스트
    axis.text = element_text(
        size = 10,
        color = "black"
    ),
    axis.text.x = element_text(
        angle = 45,
        hjust = 1
    )
)
```



24.2 그래프(plot) 수준 커스터마이징(with AI)

그래프 전체적인 모양을 제어하는 `theme()` 함수의 인자들은 표 24.2 표와 같다. 앞에서 우리는 텍스트와 관계된 내용들은 이미 한번 설명했어서, 일부는 중복된다.

한가지 주의할 점은 `plot.background` 요소는 그래프 전체의 배경(전체 캔버스)을 조정하는 것이고, `panel.background` 요소는 그래프 패널의 배경을 조정하는 요소인데 이것에 대해서는 뒤에서 다시 설명한다.

표 24.2: 그래프 수준의 요소를 제어하기 위한 `theme()` 함수의 인자들

요소	설명	주요 속성	예시
`plot.background`	그래프 전체 배경	`fill`, `colour`, `size`, `linetype`	`element_rect(fill = "white", colour = "black")`
`plot.margin`	그래프 여백	`margin()` 함수 사용	`margin(t = 10, r = 10, b = 10, l = 10)`

요소	설명	주요 속성	예시
`plot.title`	그래프 제목	`size`, `face`, `colour`, `hjust`, `margin`	`element_text(size = 16, face = "bold")`
`plot.subtitle`	그래프 부제목	`size`, `face`, `colour`, `hjust`, `margin`	`element_text(size = 14, colour = "gray50")`
`plot.caption`	그래프 캡션	`size`, `face`, `colour`, `hjust`	`element_text(size = 10, hjust = 1)`
`plot.tag`	그래프 태그	`size`, `face`, `colour`, `hjust`	`element_text(size = 12, face = "bold")`
`plot.tag.position`	태그 위치	“topleft”, “top”, “topright”, “left”, “right”, “bottomleft”, “bottom”, “bottomright”	`"topleft"`

여기서 `element_rect()` 함수는 그래프의 사각형 모양의 것들을 조절하는 데 사용된다. 주요 속성들은 다음과 같다:

- `element_rect()` 함수의 속성:
  - `fill`: 배경색
  - `colour`: 테두리 색상
  - `size`: 테두리 두께
  - `linetype`: 테두리 선 스타일 (“solid”, “dashed”, “dotted” 등)
- `margin()` 함수는 도움말을 참고한다.

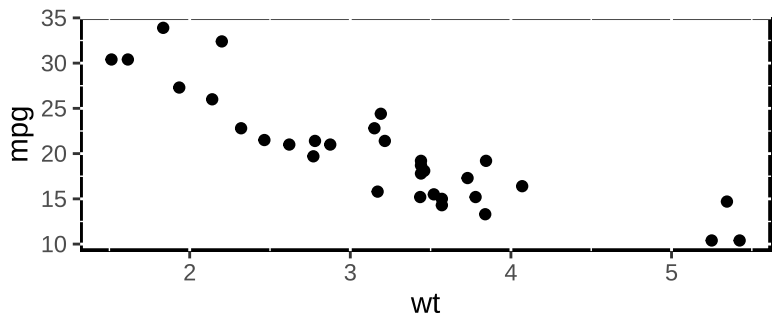
구체적인 사용법은 아래 코드를 참고한다.

```
# 예시: 그래프 수준 테마 적용하기
ggplot(mtcars, aes(x = wt, y = mpg)) +
    geom_point() +
    labs(
        title = "차량 무게와 연비의 관계",
```

```
    subtitle = "mtcars 데이터셋",
    caption = "데이터 출처: R 기본 데이터셋",
    tag = "그림 1"
) +
theme(
    # 그래프 배경
    plot.background = element_rect(
        fill = "white",
        colour = "black",
        linewidth = 1
    ),
    # 플롯팅 영역
    panel.background = element_rect(
        fill = "white",
        colour = "black",
        linewidth = 1
    ),
    # 그래프 여백
    plot.margin = margin(
        t = 20, # 위쪽
        r = 20, # 오른쪽
        b = 20, # 아래쪽
        l = 20 # 왼쪽
    ),
    # 제목
    plot.title = element_text(
        size = 16,
        face = "bold",
        hjust = 0.5,
        margin = margin(b = 10)
    ),
    # 부제목
    plot.subtitle = element_text(
```

```
            size = 12,
            colour = "gray50",
            hjust = 0.5,
            margin = margin(b = 20)
        ),
        # 캡션
        plot.caption = element_text(
            size = 10,
            colour = "gray50",
            hjust = 1,
            margin = margin(t = 10)
        ),
        # 태그
        plot.tag = element_text(
            size = 12,
            face = "bold",
            colour = "#004476"
        ),
        plot.tag.position = "topleft"
    )
```

그림 1



24.3 레전드 다루기(with AI)

범례(legend)를 제어하는 `theme()` 함수의 인자들은 표 24.3 표와 같다.

표 24.3: 범례를 제어하기 위한 `theme()` 함수의 인자들

요소	설명	주요 속성	예시
`legend.position`	범례 위치	“none”, “left”, “right”, “bottom”, “top”, 또는 좌표값	`"bottom"` 또는 `c(0.95, 0.95)`
`legend.justification`	범례 정렬 기준점	“left”, “right”, “center”, “top”, “bottom” 또는 좌표값	`c("right", "top")`
`legend.direction`	범례 항목 배치 방향	“horizontal”, “vertical”	`"horizontal"`
`legend.box`	여러 범례의 배치	“horizontal”, “vertical”	`"horizontal"`
`legend.box.just`	여러 범례의 정렬	“left”, “right”, “center”, “top”, “bottom”	`"left"`
`legend.box.margin`	여러 범례 사이 여백	`margin()` 함수 사용	`margin(6, 6, 6, 6)`

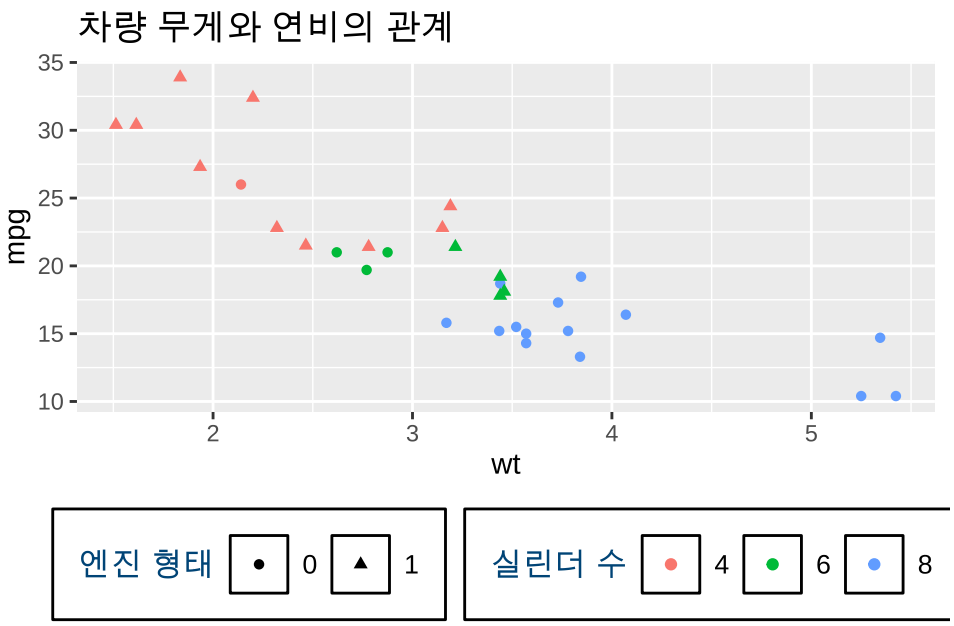
요소	설명	주요 속성	예시
legend.box.background	여러 범례 배경	element_rect() 함수 사용	element_rect(fill = "white")
legend.margin	범례 여백	margin() 함수 사용	margin(6, 6, 6, 6)
legend.background	범례 배경	element_rect() 함수 사용	element_rect(fill = "white")
legend.key	범례 키 배경	element_rect() 함수 사용	element_rect(fill = "white")
legend.key.size	범례 키 크기	unit() 함수 사용	unit(1.5, "lines")
legend.key.height	범례 키 높이	unit() 함수 사용	unit(1.5, "lines")
legend.key.width	범례 키 너비	unit() 함수 사용	unit(1.5, "lines")
legend.text	범례 텍스트	element_text() 함수 사용	element_text(size = 10)
legend.title	범례 제목	element_text() 함수 사용	element_text(face = "bold")
legend.title.align	범례 제목 정렬	0-1 사이 값	0.5
legend.text.align	범례 텍스트 정렬	0-1 사이 값	0
legend.spacing	범례 항목 간격	unit() 함수 사용	unit(0.5, "lines")
legend.spacing.x	범례 항목 가로 간격	unit() 함수 사용	unit(0.5, "lines")
legend.spacing.y	범례 항목 세로 간격	unit() 함수 사용	unit(0.5, "lines")

구체적인 사용법은 아래 코드를 참고한다.

```
# 예시: 범례 테마 적용하기
ggplot(mtcars, aes(x = wt, y = mpg, color = factor(cyl), shape = factor(vs))) +
    geom_point() +
```

```
    labs(
        title = "차량 무게와 연비의 관계",
        color = "실린더 수",
        shape = "엔진 형태"
    ) +
    theme(
        # 범례 위치
        legend.position = "bottom",
        legend.direction = "horizontal",

        # 범례 배경
        legend.background = element_rect(
            fill = "white",
            colour = "black",
            linewidth = 0.5
        ),

        # 범례 키
        legend.key = element_rect(
            fill = "white",
            colour = "black",
            linewidth = 0.5
        ),
        legend.key.size = unit(1.5, "lines"),

        # 범례 텍스트
        legend.text = element_text(
            size = 10,
            colour = "black"
        ),

        # 범례 제목
        legend.title = element_text(
```

```
            size = 12,
            face = "bold",
            colour = "#004476"
        ),

        # 범례 여백
        legend.margin = margin(
            t = 10,
            r = 10,
            b = 10,
            l = 10
        ),

        # 범례 항목 간격
        legend.spacing = unit(0.5, "lines")
    )
```



24.4 축(axis) 다루기(with AI)

축(axis)을 제어하는 `theme()` 함수의 인자들은 표 24.4 표와 같다.

표 24.4: 축을 제어하기 위한 `theme()` 함수의 인자들

요소	설명	주요 속성	예시
`axis.line`	축 선	`colour`, `size`, `linetype`	`element_line(colour = "black", size = 1)`
`axis.line.x`	x축 선	`colour`, `size`, `linetype`	`element_line(colour = "black", size = 1)`
`axis.line.y`	y축 선	`colour`, `size`, `linetype`	`element_line(colour = "black", size = 1)`
`axis.text`	축 텍스트	`size`, `face`, `colour`, `angle`, `hjust`, `vjust`	`element_text(size = 10, colour = "black")`
`axis.text.x`	x축 텍스트	`size`, `face`, `colour`, `angle`, `hjust`, `vjust`	`element_text(angle = 45, hjust = 1)`
`axis.text.y`	y축 텍스트	`size`, `face`, `colour`, `angle`, `hjust`, `vjust`	`element_text(hjust = 1)`
`axis.title`	축 제목	`size`, `face`, `colour`, `angle`, `hjust`, `vjust`	`element_text(size = 12, face = "bold")`
`axis.title.x`	x축 제목	`size`, `face`, `colour`, `angle`, `hjust`, `vjust`	`element_text(size = 12, hjust = 0.5)`
`axis.title.y`	y축 제목	`size`, `face`, `colour`, `angle`, `hjust`, `vjust`	`element_text(size = 12, angle = 90)`

요소	설명	주요 속성	예시
axis.ticks	축 눈금	colour, size, linetype	element_line(colour = "black", size = 0.5)
axis.ticks.x	x축 눈금	colour, size, linetype	element_line(colour = "black", size = 0.5)
axis.ticks.y	y축 눈금	colour, size, linetype	element_line(colour = "black", size = 0.5)
axis.ticks.length	축 눈금 길이	unit() 함수 사용	unit(0.25, "cm")
axis.ticks.length.x	x축 눈금 길이	unit() 함수 사용	unit(0.25, "cm")
axis.ticks.length.y	y축 눈금 길이	unit() 함수 사용	unit(0.25, "cm")

앞에서 설명하지 않았던 element_line() 함수는 축 선의 모양을 조정하는 데 사용된다. 주요 속성들은 다음과 같다:

- element_line() 함수의 속성:
  - colour: 선 색상
  - size: 선 두께
  - linetype: 선 스타일 (“solid”, “dashed”, “dotted” 등)

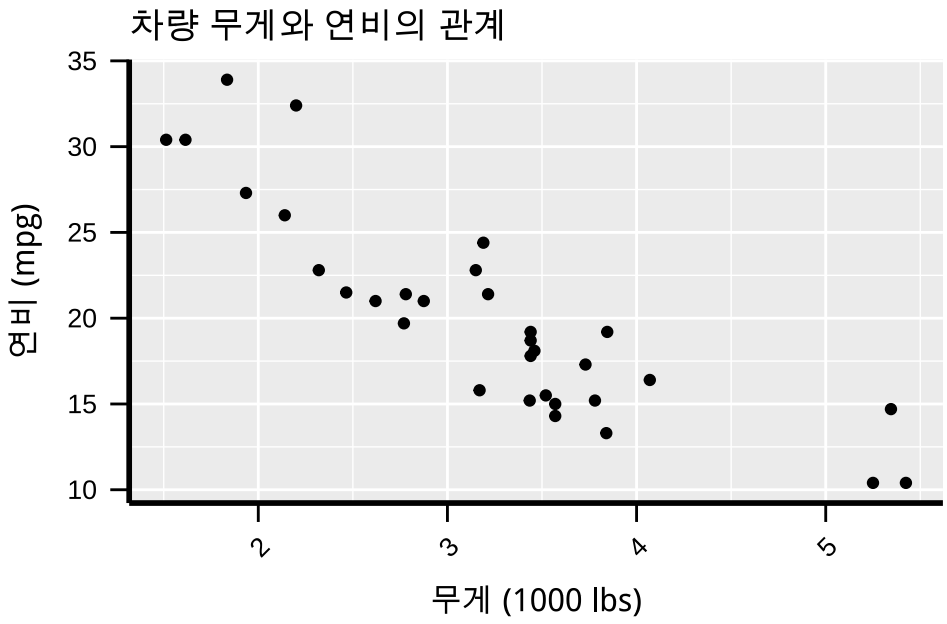
구체적인 사용법은 아래 코드를 참고한다.

```
# 예시: 축 테마 적용하기
ggplot(mtcars, aes(x = wt, y = mpg)) +
    geom_point() +
    labs(
        title = "차량 무게와 연비의 관계",
        x = "무게 (1000 lbs)",
        y = "연비 (mpg)"
```

```
) +
theme(
    # 축 선
    axis.line = element_line(
        colour = "black",
        size = 1
    ),

    # 축 텍스트
    axis.text = element_text(
        size = 10,
        colour = "black"
    ),
    axis.text.x = element_text(
        angle = 45,
        hjust = 1,
        margin = margin(t = 5)
    ),
    axis.text.y = element_text(
        hjust = 1,
        margin = margin(r = 5)
    ),

    # 축 제목
    axis.title = element_text(
        size = 12,
        face = "bold"
    ),
    axis.title.x = element_text(
        margin = margin(t = 10)
    ),
    axis.title.y = element_text(
        angle = 90,
```

```
            margin = margin(r = 10)
        ),

        # 축 눈금
        axis.ticks = element_line(
            colour = "black",
            size = 0.5
        ),

        # 축 눈금 길이
        axis.ticks.length = unit(0.25, "cm")
    )
```

```
Warning: The `size` argument of `element_line()` is deprecated as of ggplot2 3.4.0.
i Please use the `linewidth` argument instead.
```



24.5 패널(panel): 플롯팅 영역 제어(with AI)

앞에서도 설명했지만 패널이란 그래프 플롯팅 영역을 말한다. 앞에서 본 plot과 헷갈리지 말아야 한다.

패널(panel)을 제어하는 theme() 함수의 인자들은 표 24.5 표와 같다.

표 24.5: 패널을 제어하기 위한 theme() 함수의 인자들

요소	설명	주요 속성	예시
panel.background	패널 배경	fill, colour, size, linetype	element_rect(fill = "white", colour = "black")
panel.border	패널 테두리	fill, colour, size, linetype	element_rect(fill = NA, colour = "black")
panel.grid	패널 격자	colour, size, linetype	element_line(colour = "gray90")
panel.grid.major	주요 격자	colour, size, linetype	element_line(colour = "gray90", size = 0.5)
panel.grid.minor	보조 격자	colour, size, linetype	element_line(colour = "gray95", size = 0.25)
panel.grid.major.x	x축 주요 격자	colour, size, linetype	element_line(colour = "gray90")
panel.grid.major.y	y축 주요 격자	colour, size, linetype	element_line(colour = "gray90")
panel.grid.minor.x	x축 보조 격자	colour, size, linetype	element_line(colour = "gray95")
panel.grid.minor.y	y축 보조 격자	colour, size, linetype	element_line(colour = "gray95")
panel.spacing	패널 간격	unit() 함수 사용	unit(1, "lines")

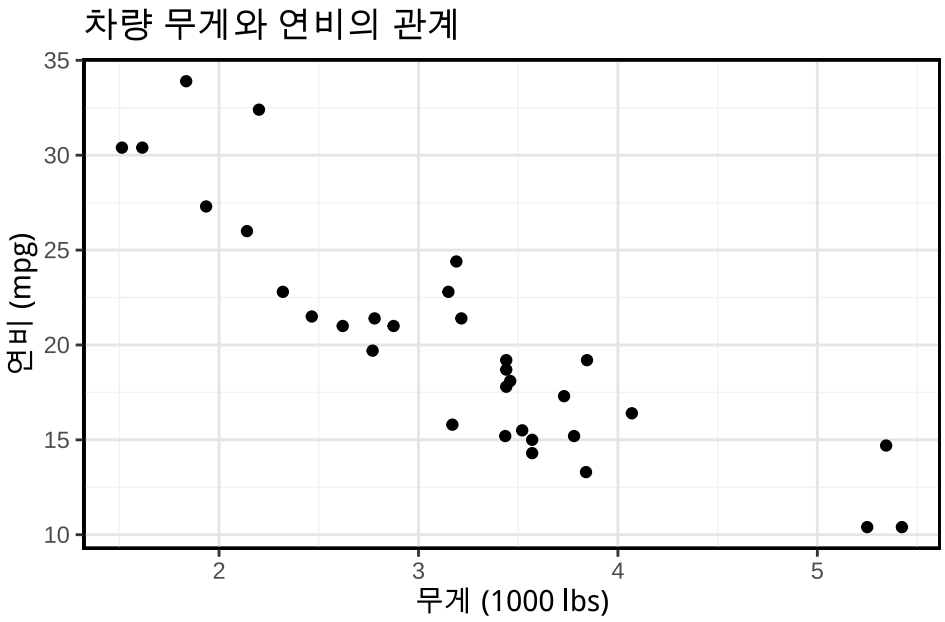
요소	설명	주요 속성	예시
panel.spacing.x	가로 패널 간격	unit() 함수 사용	unit(1, "lines")
panel.spacing.y	세로 패널 간격	unit() 함수 사용	unit(1, "lines")
panel.ontop	격자를 데이터 위에 표시	TRUE/FALSE	TRUE

구체적인 사용법은 아래 코드를 참고한다.

```
# 예시: 패널 테마 적용하기
ggplot(mtcars, aes(x = wt, y = mpg)) +
    geom_point() +
    labs(
        title = "차량 무게와 연비의 관계",
        x = "무게 (1000 lbs)",
        y = "연비 (mpg)"
    ) +
    theme(
        # 패널 배경
        panel.background = element_rect(
            fill = "white",
            colour = "black",
            size = 1
        ),

        # 패널 테두리
        panel.border = element_rect(
            fill = NA,
            colour = "black",
            size = 1
        ),
```

```
        # 주요 격자
        panel.grid.major = element_line(
            colour = "gray90",
            size = 0.5
        ),

        # 보조 격자
        panel.grid.minor = element_line(
            colour = "gray95",
            size = 0.25
        ),

        # x축 주요 격자
        panel.grid.major.x = element_line(
            colour = "gray90",
            size = 0.5
        ),

        # y축 주요 격자
        panel.grid.major.y = element_line(
            colour = "gray90",
            size = 0.5
        ),

        # 격자를 데이터 위에 표시
        panel.ontop = FALSE
    )
```

```
Warning: The `size` argument of `element_rect()` is deprecated as of ggplot2 3.4.0.
i Please use the `linewidth` argument instead.
```



24.6 패시팅(facet) 제어(with AI)

패시팅(facet)을 제어하는 `theme()` 함수의 인자들은 표 24.6 표와 같다.

표 24.6: 패시팅을 제어하기 위한 `theme()` 함수의 인자들

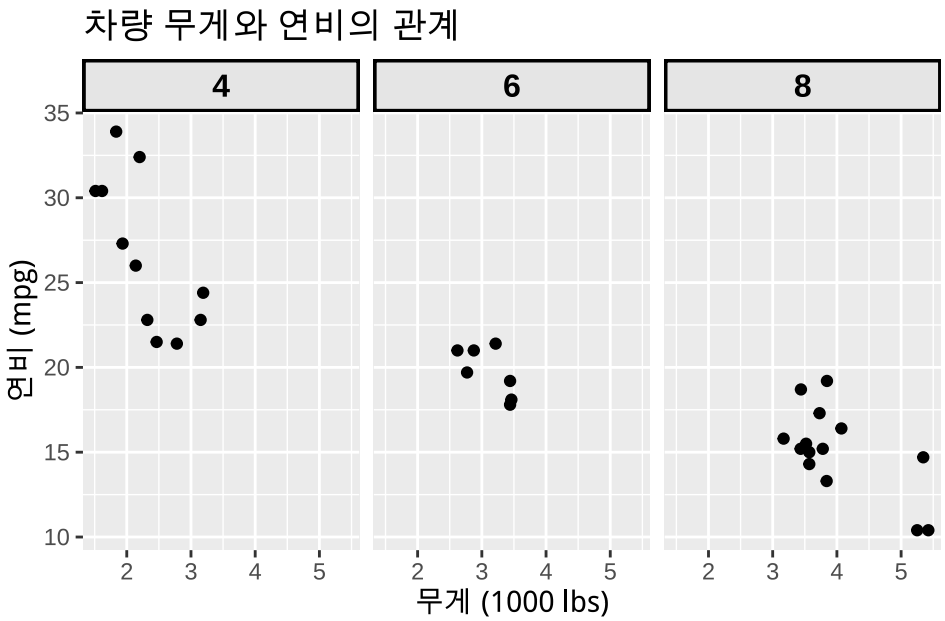
요소	설명	주요 속성	예시
`strip.background`	패시트 배경	`fill`, `colour`, `size`, `linetype`	`element_rect(fill = "gray90", colour = "black")`
`strip.background.x`	가로 패시트 배경	`fill`, `colour`, `size`, `linetype`	`element_rect(fill = "gray90")`
`strip.background.y`	세로 패시트 배경	`fill`, `colour`, `size`, `linetype`	`element_rect(fill = "gray90")`
`strip.text`	패시트 텍스트	`size`, `face`, `colour`, `angle`, `hjust`, `vjust`	`element_text(size = 12, face = "bold")`
`strip.text.x`	가로 패시트 텍스트	`size`, `face`, `colour`, `angle`, `hjust`, `vjust`	`element_text(size = 12)`

요소	설명	주요 속성	예시
strip.text.y	세로 패시트 텍스트	size, face, colour, angle, hjust, vjust	element_text(size = 12, angle = 0)
strip.placement	패시트 위치	“inside”, “outside”	"outside"
strip.switch.pad.grid	그리드 패시트 간격	unit() 함수 사용	unit(0.5, "lines")
strip.switch.pad.wrap	래핑 패시트 간격	unit() 함수 사용	unit(0.5, "lines")

구체적인 사용법은 아래 코드를 참고한다.

```
# 예시: 패시팅 테마 적용하기
ggplot(mtcars, aes(x = wt, y = mpg)) +
    geom_point() +
    facet_wrap(~cyl) +
    labs(
        title = "차량 무게와 연비의 관계",
        x = "무게 (1000 lbs)",
        y = "연비 (mpg)"
    ) +
    theme(
        # 패시트 배경
        strip.background = element_rect(
            fill = "gray90",
            colour = "black",
            size = 1
        ),

        # 패시트 텍스트
        strip.text = element_text(
            size = 12,
            face = "bold",
```

```
            colour = "black"
        ),

        # 패시트 위치
        strip.placement = "outside",

        # 패시트 간격
        strip.switch.pad.wrap = unit(0.5, "lines")
    )
```



25 주요 통계 그래프 모음 (작성중)

이 장에서는 주요 통계 그래프를 만드는 방법을 설명한다.

25.1 분포(distirbution)을 보는 그래프

여기서에는 어떤 (연속형) 변수의 분포(distribution)을 보는 그래프들을 설명하고자 한다. 분포를 보는 그래프에는 다음과 같은 것들이 있다. 이들은 하나의 변수가 어떤 분포를 가지는지를 시각화한다. 서로 다른 변수의 분포를 비교하기 위해서 하나의 그래프 안에 여러 분포을 시각화하기도 한다.

- 히스토그램 (histogram)
- 밀도 플롯(density plot): 커널 밀도 추정 (kernel density estimation)
- 박스 플롯 (box plot)
- 바이올린 플롯 (violin plot)
- 릿지 플롯(ridge plot)

```
library(ggplot2)
library(dplyr)
df <- readRDS("data/stroke_df.rds")
glimpse(df)
```

```
Rows: 5,110
Columns: 12
$ id                <chr> "9046", "51676", "31112", "60182", "1665", "56669", ~
$ gender            <fct> Male, Female, Male, Female, Female, Male, Male, Fema~
$ age               <dbl> 67, 61, 80, 49, 79, 81, 74, 69, 59, 78, 81, 61, 54, ~
$ hypertension      <fct> No, No, No, No, Yes, No, Yes, No, No, No, Yes, No, N~
$ heart_disease     <fct> Yes, No, Yes, No, No, No, Yes, No, No, No, No, Yes, ~
```

```
$ ever_married      <fct> Yes, Yes, Yes, Yes, Yes, Yes, Yes, No, Yes, Yes, Yes~
$ work_type         <fct> Private, Self-employed, Private, Private, Self-emplo~
$ residence_type    <fct> Urban, Rural, Rural, Urban, Rural, Urban, Rural, Urb~
$ avg_glucose_level <dbl> 228.69, 202.21, 105.92, 171.23, 174.12, 186.21, 70.0~
$ bmi               <dbl> 36.6, NA, 32.5, 34.4, 24.0, 29.0, 27.4, 22.8, NA, 24~
$ smoking_status    <fct> formerly smoked, never smoked, never smoked, smokes,~
$ stroke            <fct> Yes, Yes, Yes, Yes, Yes, Yes, Yes, Yes, Yes, Yes, Ye~
```

```
ggplot(df, aes(x = age)) +
  geom_histogram(binwidth = 5, fill = "steelblue", color = "white") +
  labs(title = "Histogram of Age", x = "Age", y = "Count") +
  theme_minimal()
```

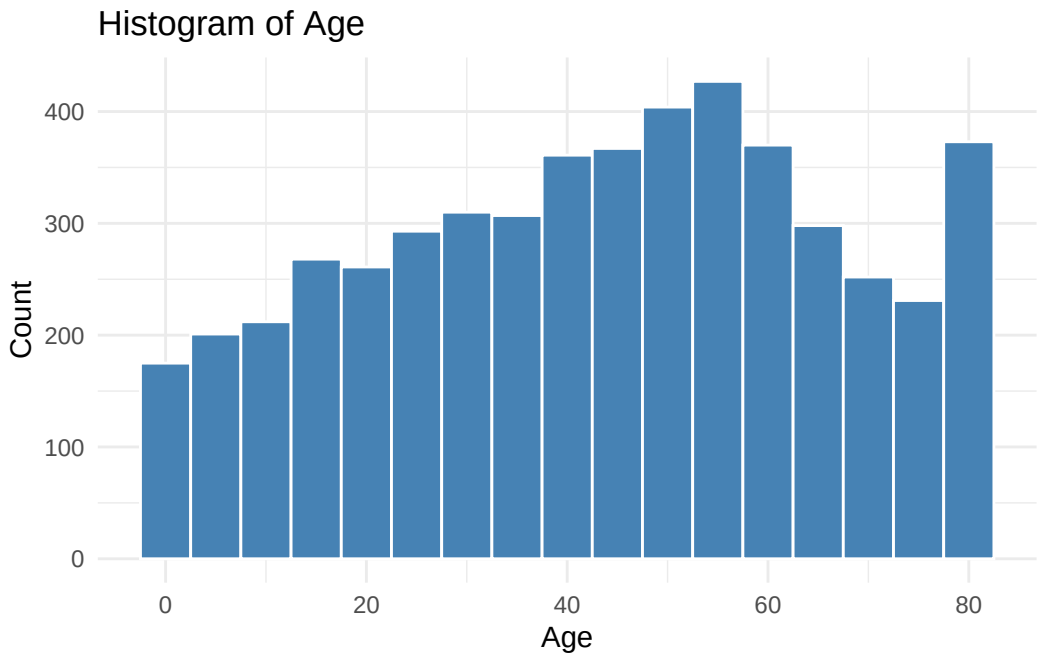
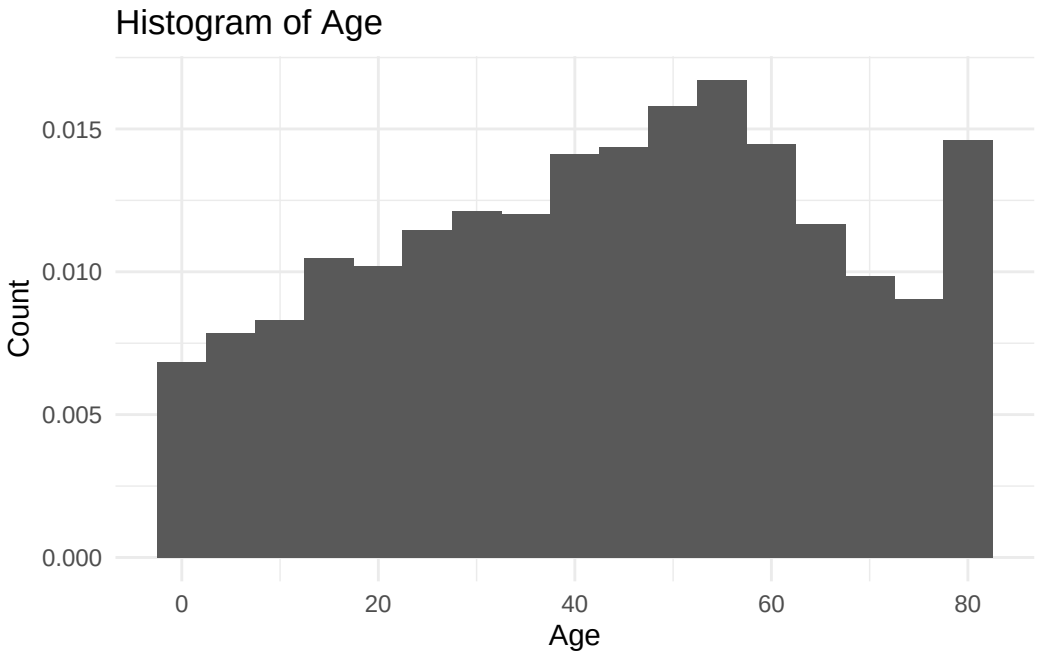
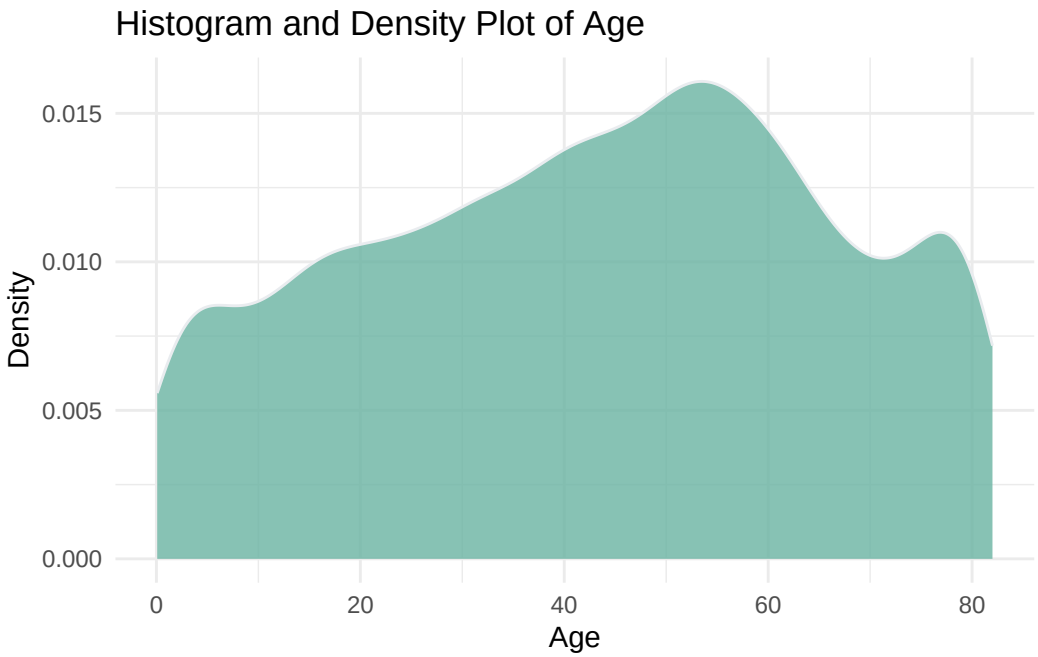


그림 25.1: 히스토그램

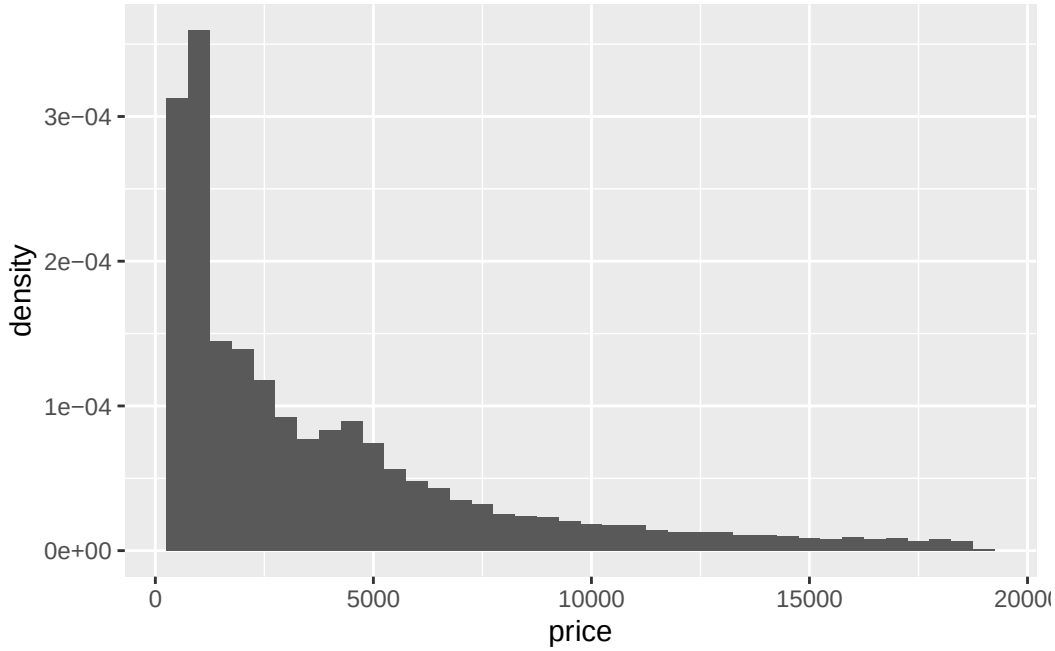
```
ggplot(df, aes(x = age)) +
  geom_histogram(aes(y = after_stat(density)), binwidth=5) +
  labs(title = "Histogram of Age", x = "Age", y = "Count") +
  theme_minimal()
```




```r
ggplot(df, aes(x = age)) +
  geom_density(fill="#69b3a2", color="#e9ecef", alpha=0.8) +
  labs(title = "Histogram and Density Plot of Age", x = "Age", y = "Density") +
  theme_minimal()
```



```r
ggplot(diamonds, aes(price)) +
  geom_histogram(aes(y = after_stat(density)), binwidth = 500)
```



Part IV

IV. R 통계 분석

26 R로 통계를 하려면 꼭 알아야 하는 R 포뮬러 (Formula)

R 언어에서는 통계적 모델(statistical model)을 표현하기 위해 **포뮬러(formula)**를 사용한다. R 언어의 관점에서는 이것 역시 벡터나 데이터프레임, 리스트와 같은 객체의 한 종류이다.

포뮬러는 모델의 독립변수와 종속변수 간의 관계를 나타내는 상징적인 수식이다. 여기서 **상징적 수식**이라고 하는 것은 보통의 수학적 표현이 아니라는 뜻이다. R 언어에서 특별한 역할을 하는 표현식의 일종이다.

R에서 많은 통계 분석 함수나 그래픽 함수들이 이 포뮬러를 사용하기 때문에 잘 이해할 필요가 있고, 실제로도 많은 데이터 분석 작업에서 포뮬러를 사용한다.

26.1 R 포뮬러로 모델 정의하기

```
f <- y ~ x + b
class(f)
```

```
[1] "formula"
```

~ 기호를 중심으로 왼쪽은 종속 변수(dependent variable), 오른쪽은 독립 변수(independent variable)를 표시한다. 대부분 이런 변수들은 데이터프레임의 열 이름을 사용한다. 이 `f`라는 객체는 `formula` 클래스의 객체이다.

포뮬러를 정의할 때는 표 26.1에 있는 기호들을 사용한다.

표 26.1: R 포뮬러 기호 정의

기호	역할	예시	통계적의미
~	포뮬러의 좌변과 우변을 구분	y ~ x	regress y on x
+	포뮬러에 변수 추가함	y ~ x + z	regress y on x and z
.	모든 변수를 사용	y ~ .	regress y on all other variables in a data frame
-	포뮬러에서 변수를 뺌	y ~ . - x	regress y on all other variables except x
1	Y 절편 포함	y ~ x - 1	regress y on x without an intercept
:	상호작용	y ~ x + z + x:z	regress y on x, z, and the product x times z
*	factor crossing	y ~ x * z	regress y on x, z, and the product x times z
^	고차원 상호작용	y ~ (x + z + w)^3	regress y on x, z, w, all two-way interactions, and the three-way interactions
I()	안에 수식을 진짜 수식으로 사용	y ~ x + I(x^2)	regress y on x and x squared

- `*` 기호는 factor crossing을 의미한다. `a*b`는 `a + b + a:b`와 같다.
- `^` 기호는 고차원 상호작용을 의미한다. `(a+b+c)^2`는 `a + b + c + a:b + a:c + b:c`와 같다.
- `I()` 기호는 안에 있는 것이 진짜 수식으로 사용된다는 것을 의미한다.

26.2 예제로 살펴 보기

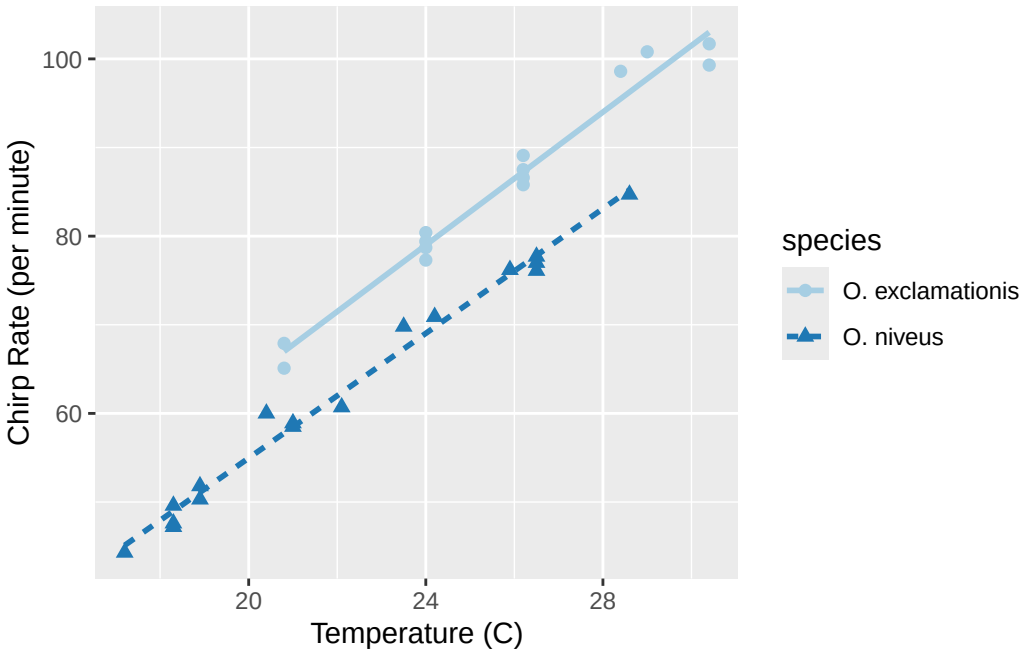
다음 자료는 Tidymodeling with R 책에서 인용하였다.

귀뛰라미(crickets) 데이터는 온도와 귀뛰라미의 초당 소리 횟수 간의 관계를 나타내는 데이터이고, 2개의 아종별에 대해 조사했다.

```
library(tidyverse)

data(crickets, package = "modeldata")
names(crickets)
```

```
[1] "species" "temp"    "rate"
```

```
# Plot the temperature on the x-axis, the chirp rate on the y-axis. The plot
# elements will be colored differently for each species:
ggplot(
    crickets,
    aes(x = temp, y = rate, color = species, pch = species, lty = species)
) +
    # Plot points for each data point and color by species
    geom_point(size = 2) +
    # Show a simple linear model fit created separately for each species:
    geom_smooth(method = lm, se = FALSE, alpha = 0.5) +
    scale_color_brewer(palette = "Paired") +
    labs(x = "Temperature (C)", y = "Chirp Rate (per minute)")
```



위 그래프를 보면 온도와 귀뛰라미의 초당 소리 횟수 간의 선형 관계가 있고, 각 아종이 서로 다르다는 것을 볼 수 있다.

먼저 `lm()` 함수를 사용하는데, 이 함수는 R 포뮬러와 데이터프레임을 인자로 받는다. 먼저 소리 횟수와 온도와 관계를 R 포뮬러로 다음과 같이 표현할 수 있다.

```
rate ~ temp
```

종속 변수 rate가 독립 변수 temp에 의해 설명된다는 것을 의미한다.

만약 아종에 대한 변수를 독립 변수로 추가하려면 다음과 같은 포뮬러를 사용할 수 있다.

```
rate ~ temp + species
```

이 경우 species는 범주형 변수이다. 이 포뮬러는 온도와 아종에 대한 영향을 동시에 고려한다는 것을 의미한다. 이런 범주형 변수는 모델 fitting 과정에서 숫자형으로 변환된 다음 계산된다.

temp와 species이 상호작용(interaction)을 하는 경우 다음과 같은 포뮬러를 사용할 수 있다. 변수들 간의 상호작용은 콜론(:)으로 표현한다.

```
rate ~ temp  + species + temp:species
```

위 포뮬러는 다음과 같은 단축형으로도 표현할 수 있는데, 모두 같은 의미이다.

```
rate ~ temp * species
rate ~ (temp + species)^2
```

이제 모델 fitting을 해보자.

```
interaction_fit <- lm(rate ~ (temp + species)^2, data = crickets)
```

이렇게 해서 만들어진 interaction_fit 객체는 여러 정보를 담고 있다.

```
interaction_fit
```

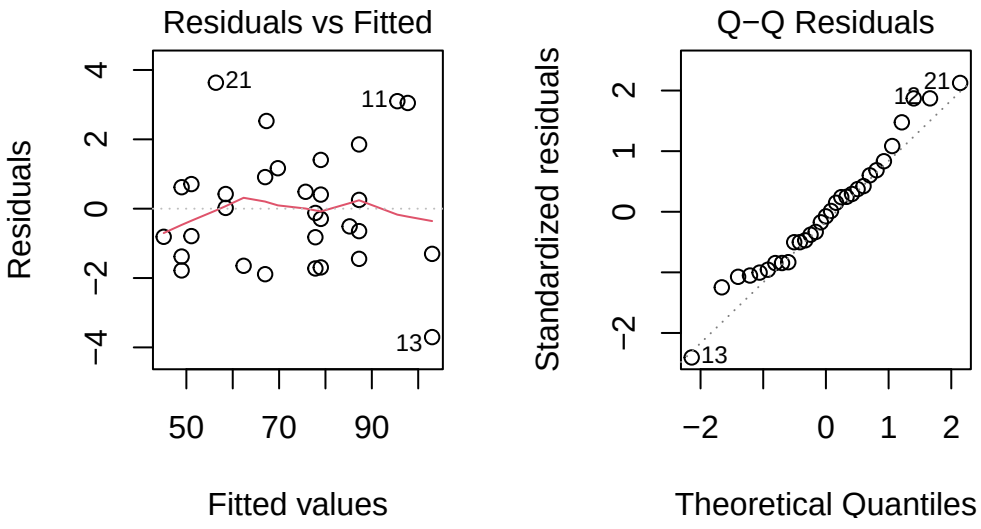
```
Call:
lm(formula = rate ~ (temp + species)^2, data = crickets)

Coefficients:
          (Intercept)                  temp       speciesO. niveus
              -11.041                 3.751                -4.348
temp:speciesO. niveus
               -0.234
```

```r
summary(interaction_fit)
```

```
Call:
lm(formula = rate ~ (temp + species)^2, data = crickets)

Residuals:
    Min      1Q  Median      3Q     Max
-3.7031 -1.3417 -0.1235  0.8100  3.6330

Coefficients:
                     Estimate Std. Error t value Pr(>|t|)
(Intercept)          -11.0408     4.1515  -2.659    0.013 *
temp                   3.7514     0.1601  23.429   <2e-16 ***
speciesO. niveus      -4.3484     4.9617  -0.876    0.389
temp:speciesO. niveus -0.2340     0.2009  -1.165    0.254
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.775 on 27 degrees of freedom
Multiple R-squared:  0.9901,    Adjusted R-squared:  0.989
F-statistic: 898.9 on 3 and 27 DF,  p-value: < 2.2e-16
```

이제 모델을 시각화해보자.

```r
# Place two plots next to one another:
par(mfrow = c(1, 2))

# Show residuals versus predicted values:
plot(interaction_fit, which = 1)

# A normal quantile plot on the residuals:
plot(interaction_fit, which = 2)
```



인터랙션 term이 필요한지를 확인하기 위해 인터랙션 term이 없는 모델을 보자.

```
main_effects_fit <- lm(rate ~ temp + species, data = crickets)
```

인터랙션이 있는 모델과 없는 모델의 차이를 `anova()` 함수를 사용하여 비교할 수 있다.

```
anova(main_effects_fit, interaction_fit)
```

```
Analysis of Variance Table

Model 1: rate ~ temp + species
Model 2: rate ~ (temp + species)^2
  Res.Df    RSS Df Sum of Sq     F Pr(>F)
1     28 89.350
2     27 85.074  1    4.2758 1.357 0.2542
```

P-값이 0.25로 인터랙션 term이 필요하지 않다는 귀무 가설을 기각할 수 없다.

```
summary(main_effects_fit)
```

```
Call:
lm(formula = rate ~ temp + species, data = crickets)
```

```
Residuals:
    Min      1Q  Median      3Q     Max
-3.0128 -1.1296 -0.3912  0.9650  3.7800

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)       -7.21091    2.55094  -2.827  0.00858 **
temp               3.60275    0.09729  37.032  < 2e-16 ***
speciesO. niveus -10.06529    0.73526 -13.689 6.27e-14 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.786 on 28 degrees of freedom
Multiple R-squared:  0.9896,    Adjusted R-squared:  0.9888
F-statistic:  1331 on 2 and 28 DF,  p-value: < 2.2e-16
```

이렇게 만들어진 모델을 사용하여 예측(prediction)을 할 수 있다.

```r
new_values <- data.frame(species = "O. exclamationis", temp = 15:20)
predict(main_effects_fit, new_values)
```

```
       1        2        3        4        5        6
46.83039 50.43314 54.03589 57.63865 61.24140 64.84415
```

- aov()와 anova() 함수의 차이

26.3 R 포뮬러가 하는 일

- 모델에 사용되는 변수(열) 정의
- 팩터 형 변수 열을 fitting 할 때 적절한 형태로 변환
- 변수(열)의 역할 정의: 종속 변수 vs 독립 변수

26.4 참고 자료

- R Formula Tutorial

27 통계 분석 결과의 활용도를 높여주는 broom 패키지

전통적인 base R에서 통계 분석 결과(통계적 모델)를 활용하는 방법은 `summary()` 함수를 사용하는 것이다. 하지만 이 방법은 결과를 데이터프레임으로 변환할 수 없고, 결과를 정리하기도 어렵다. 다음 예를 보자.

27.1 전통적 통계 분석과 활용법

26 장에서 보았던 귀뚜라미(crickets) 데이터셋을 사용하여 귀뚜라미의 초당 소리 횟수(`rate`)와 온도(`temp`), 아종(`species`) 간의 관계를 분석한 예이다.

```
library(tidyverse)
data(crickets, package = "modeldata")
interaction_fit <- lm(rate ~ temp * species, data = crickets)
```

`lm()` 함수를 사용하여 선형 회귀 모델을 만들었다. 이 모델은 온도와 아종이 초당 소리 횟수에 미치는 영향을 분석한다. 이렇게 만들어진 모델 객체를 `interaction_fit` 변수에 저장했다. 보통 이 객체를 출력해 보거나, `summary()` 함수를 사용하여 결과를 확인한다.

```
attributes(interaction_fit)
```

```
$names
 [1] "coefficients"  "residuals"     "effects"       "rank"
 [5] "fitted.values" "assign"        "qr"            "df.residual"
 [9] "contrasts"     "xlevels"       "call"          "terms"
[13] "model"
```

```
$class
[1] "lm"
```

이 가운데 잔차(residuals)와 회귀 계수(coefficients)를 확인해 보자.‘

```
interaction_fit$residuals
```

```
          1           2           3           4           5           6
 0.91074559 -1.88925441 -1.69388557 -0.29388557  0.40611443  1.40611443
          7           8           9          10          11          12
-1.44706949 -0.64706949  0.25293051  1.85293051  3.09974659  3.04887825
         13          14          15          16          17          18
-3.70314789 -1.30314789 -0.81108570 -1.78029355 -1.38029355  0.61970645
         19          20          21          22          23          24
-0.79077057  0.70922943  3.63303691  0.02255989  0.42255989 -1.64664796
         25          26          27          28          29          30
 2.52890568  1.16668250  0.48699763 -1.72347938 -0.82347938 -0.12347938
         31
-0.51014892
```

```
interaction_fit$coefficients
```

```
         (Intercept)                 temp      speciesO. niveus
         -11.0408481            3.7514472            -4.3484072
temp:speciesO. niveus
          -0.2339856
```

또는 이 객체를 summary() 함수를 사용하여 요약할 수 있다.

```
result_summary <- summary(interaction_fit)
result_summary
```

```
Call:
```

```
lm(formula = rate ~ temp * species, data = crickets)

Residuals:
    Min      1Q  Median      3Q     Max
-3.7031 -1.3417 -0.1235  0.8100  3.6330

Coefficients:
                     Estimate Std. Error t value Pr(>|t|)
(Intercept)          -11.0408     4.1515  -2.659    0.013 *
temp                   3.7514     0.1601  23.429   <2e-16 ***
speciesO. niveus      -4.3484     4.9617  -0.876    0.389
temp:speciesO. niveus -0.2340     0.2009  -1.165    0.254
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.775 on 27 degrees of freedom
Multiple R-squared:  0.9901,    Adjusted R-squared:  0.989
F-statistic: 898.9 on 3 and 27 DF,  p-value: < 2.2e-16
```

이렇게 만들어진 객체에 대해 여러 속성(attributes)을 확인할 수 있다. 예를 들어 회귀 계수, 결정 계수, 조정된 결정 계수, F-통계량 등을 확인할 수 있다.

```
result_summary$coefficients
```

```
                       Estimate Std. Error    t value     Pr(>|t|)
(Intercept)          -11.0408481  4.1514800 -2.6594969 1.300079e-02
temp                   3.7514472  0.1601220 23.4286850 1.780831e-19
speciesO. niveus      -4.3484072  4.9616805 -0.8763981 3.885447e-01
temp:speciesO. niveus -0.2339856  0.2008622 -1.1649059 2.542464e-01
```

```
result_summary$r.squared
```

```
[1] 0.9900871
```

```
result_summary$adj.r.squared
```

```
[1] 0.9889857
```

```
result_summary$fstatistic
```

```
   value    numdf    dendf 
898.9095   3.0000  27.0000 
```

베이스 R에서는 `t.test()`, `aov()`, `lm()` 등 통계 분석 함수를 사용하여 모델을 만들고, 그 결과를 `summary()` 함수를 사용하여 요약하는 것이 일반적이다. 그런데 이렇게 하는 경우는 다음 단계의 분석이나 시각화 등 후속 작업을 하기에 불편한다. 이런 불편을 해소하기 위해 `broom` 패키지가 만들어졌다.

27.2 broom 패키지 소개

`broom` 패키지는 통계 분석 결과를 깔끔하게 정리하여 데이터프레임 형태로 변환해 주는 패키지이다. 이 패키지의 주 함수는 다음 3가지이다.

1. `broom::tidy()`: 통계 분석 결과를 데이터프레임 형태로 변환한다.
2. `broom::glance()`: 통계 분석 결과의 요약 정보를 데이터프레임 형태로 변환한다.
3. `broom::augment()`: 통계 분석 결과를 원본 데이터에 추가하여 데이터프레임 형태로 변환한다.

```
library(broom)
```

`broom`이 지원하는 통계 모델은 `lm()`, `glm()`, `nls()`, `lmer()`, `glmer()`, `gam()` 등 다양한 모델을 지원한다. 패키지 비니에트 "Available methods"를 참고하면 어떤 모델이 지원되는지 확인할 수 있다.

27.3 broom 패키지로 통계 분석 결과 정리하기

`broom` 패키지를 사용하여 통계 분석 결과를 정리해 보자. 먼저 `tidy()` 함수를 사용하여 회귀 계수와 잔차를 데이터프레임 형태로 변환한다.

```
tidy_result <- tidy(interaction_fit)
tidy_result
```

```
# A tibble: 4 x 5
  term                  estimate std.error statistic  p.value
  <chr>                    <dbl>     <dbl>     <dbl>    <dbl>
1 (Intercept)            -11.0       4.15     -2.66  1.30e- 2
2 temp                     3.75      0.160    23.4   1.78e-19
3 speciesO. niveus        -4.35      4.96     -0.876 3.89e- 1
4 temp:speciesO. niveus   -0.234     0.201    -1.16  2.54e- 1
```

`tidy()` 함수는 통계 분석 결과를 데이터프레임 형태로 변환하여, 각 회귀 계수와 그에 대한 통계적 검정 결과를 포함한다. 이 데이터프레임은 `term`, `estimate`, `std.error`, `statistic`, `p.value` 등의 열을 포함한다.

이제 `glance()` 함수를 사용하여 모델의 요약 정보를 데이터프레임 형태로 변환하자.

```
glance_result <- glance(interaction_fit)
glance_result
```

```
# A tibble: 1 x 12
  r.squared adj.r.squared sigma statistic  p.value    df logLik   AIC   BIC
      <dbl>         <dbl> <dbl>     <dbl>    <dbl> <dbl>  <dbl> <dbl> <dbl>
1     0.990         0.989  1.78      899. 3.77e-27     3  -59.6  129.  136.
# i 3 more variables: deviance <dbl>, df.residual <int>, nobs <int>
```

`glance()` 함수는 모델의 요약 정보를 데이터프레임 형태로 변환하여, 결정 계수, 조정된 결정 계수, F-통계량, p-value 등의 정보를 포함한다. 이 데이터프레임은 모델의 전반적인 성능을 평가하는 데 유용하다.

이제 `augment()` 함수를 사용하여 원본 데이터에 통계 분석 결과를 추가한다.

```
augment_result <- augment(interaction_fit)
augment_result
```

```
# A tibble: 31 x 9
    rate  temp species          .fitted .resid   .hat .sigma  .cooksd .std.resid
   <dbl> <dbl> <fct>              <dbl>  <dbl>  <dbl>  <dbl>    <dbl>      <dbl>
 1  67.9  20.8 O. exclamationis    67.0  0.911 0.271    1.80 0.0336         0.601
 2  65.1  20.8 O. exclamationis    67.0 -1.89  0.271    1.76 0.145         -1.25
 3  77.3  24   O. exclamationis    79.0 -1.69  0.0966   1.77 0.0269        -1.00
 4  78.7  24   O. exclamationis    79.0 -0.294 0.0966   1.81 0.000811      -0.174
 5  79.4  24   O. exclamationis    79.0  0.406 0.0966   1.81 0.00155        0.241
 6  80.4  24   O. exclamationis    79.0  1.41  0.0966   1.79 0.0186         0.833
 7  85.8  26.2 O. exclamationis    87.2 -1.45  0.0730   1.78 0.0141        -0.847
 8  86.6  26.2 O. exclamationis    87.2 -0.647 0.0730   1.80 0.00282       -0.379
 9  87.5  26.2 O. exclamationis    87.2  0.253 0.0730   1.81 0.000431       0.148
10  89.1  26.2 O. exclamationis    87.2  1.85  0.0730   1.77 0.0232         1.08
# i 21 more rows
```

`augment()` 함수는 원본 데이터에 통계 분석 결과를 추가하여, 각 관측치에 대한 예측값, 잔차, 표준화된 잔차 등을 포함하는 데이터프레임을 생성한다. 이 데이터프레임은 모델의 적합도를 평가하고, 이상치를 식별하는 데 유용하다.

28 P-value의 의미 (with `infer` package)

가설 검정(hypothesis testing) 또는 `Null Hypotheisis Significance Test`에서 p-값을 보고 어떤 결정을 내린다. 이 장에서는 p-값의 의미와 해석에 대해 알아보려고 한다. 이런 p-값은 통계학에서 매우 중요한 것인데, 개념을 오인하기 쉽기로도 악명이 높다. 어떤 책에 나온 퀴즈를 번역한 것이다(Lazic 2016). 어느 것이 정답일까?

> ℹ 통계학 퀴즈 p-값 0.05가 당신에게 말하는 것은 …
>
> a. 귀무가설(H_0)이 참일 확률 (즉, 효과가 없을 가능성이 5%)
> b. 대립가설(H_1)이 참일 확률이 95% (그래서 효과가 실제할 확률이 높다.)
> c. 재현될 확률이 95% 이상이다.
> d. 결과가 거짓 양성일 확률이 5%이다.
> e. 효과의 크기에 관한 어떤 것; 낮은 p-값은 큰 효과를 시사
> f. 귀무가설이 참이라고 가정했을 때, 이때 얻어지는 결과보다 같거나 또는 더 극단적인 결과를 얻을 확률
> g. 위 목적들의 일부 조합 (어떤 내용인지 명시하세요).

이 장에서는 `infer` 패키지를 사용하여 p-값을 계산하는 방법을 알아보고, p-값의 의미와 해석에 대해 설명한다. `infer` 패키지는 R에서 통계적 가설 검정을 수행하는 데 유용한 도구를 제공한다. 이 패키지를 사용하면 p-값을 쉽게 계산하고, 이를 통해 귀무가설을 검정할 수 있다. 이 파트의 후반분에서 "가설 검정" 방법론이 비교적 일반적인 원리에 따른다는 것을 이해하게 되면 `infer` 패키지를 더 잘 사용할 수 있을 것이다.

다만, 여기서는 p-값의 의미와 해석에 중점을 두고 설명한다.

28.1 t-검정

28.1.1 전통적인 방법: 마치 매뉴얼에 따라서 하는 것처럼

29 장에서 t-검정(t-test)에 대해 설명했다. t-검정은 두 집단의 평균을 비교하는 통계적 방법이다. 이때 p-값은 두 집단의 평균 차이가 우연에 의한 것인지, 아니면 실제로 차이가 있는지를 판단하는 데 사용된다.

29 장에서 설명하는 바와 같이 방의 길이를 측정한 `roomwidth` 데이터셋을 사용한다.

```
library(HSAUR2)
library(tidyverse)
data("roomwidth", package = "HSAUR2")
glimpse(roomwidth)
```

```
Rows: 113
Columns: 2
$ unit  <fct> metres, metres, metres, metres, metres, metres, metres, metres, ~
$ width <dbl> 8, 9, 10, 10, 10, 10, 10, 10, 11, 11, 11, 11, 12, 12, 13, 13, 13~
```

이 데이터셋은 방의 너비를 피트(feet)와 미터(meters) 단위로 측정한 것이다. 이 데이터를 사용하여 두 집단(피트와 미터)의 평균 차이를 검정하려고 하는데, 먼저 두 집단의 평균을 비교하기 위해 데이터를 변환한다. 1미터는 3.28피트이므로, 방의 너비를 하나의 단위로 맞추어 `converted`라는 새로운 변수를 만든다.

```
df <- roomwidth |>
    mutate(
        converted = ifelse(
            unit == "feet", 1 * width, 3.28 * width
        )
    )
```

전통적인 방법으로 t-검정을 수행하려면 `t.test()` 함수를 사용하고 p-값을 확인한다. 이때 귀무가설은 두 집단의 평균이 동일하다는 것이다. 즉, p-값이 0.05보다 작으면 귀무가설을 기각하고, 두 집단의 평균이 통계적으로 유의미하게 다르다고 판단한다.

```r
t_test_result <- t.test(
    converted ~ unit,
    data = df,
    var.equal = TRUE
)
t_test_result
```

```
    Two Sample t-test

data:  converted by unit
t = -2.6147, df = 111, p-value = 0.01017
alternative hypothesis: true difference in means between group feet and group metres is not equal
95 percent confidence interval:
 -15.572734  -2.145052
sample estimates:
  mean in group feet mean in group metres
            43.69565             52.55455
```

계산된 p-값이 0.0102이므로, 0.05보다 작다. 따라서 귀무가설을 기각하고, 두 집단의 평균이 통계적으로 유의미하게 다르다고 결론 내린다.

28.1.2 `infer` 패키지: 가설 검정 방법론에 따라서

`infer` 패키지를 사용하여 p-값을 계산하는 방법을 보자.

먼저 우리는 우리가 관찰한 데이터에서 통계량을 계산한다.

```r
library(infer)
observed_stat <- df |>
    specify(converted ~ unit) |>
    calculate(stat = "t", order = c("feet", "metres"))
observed_stat
```

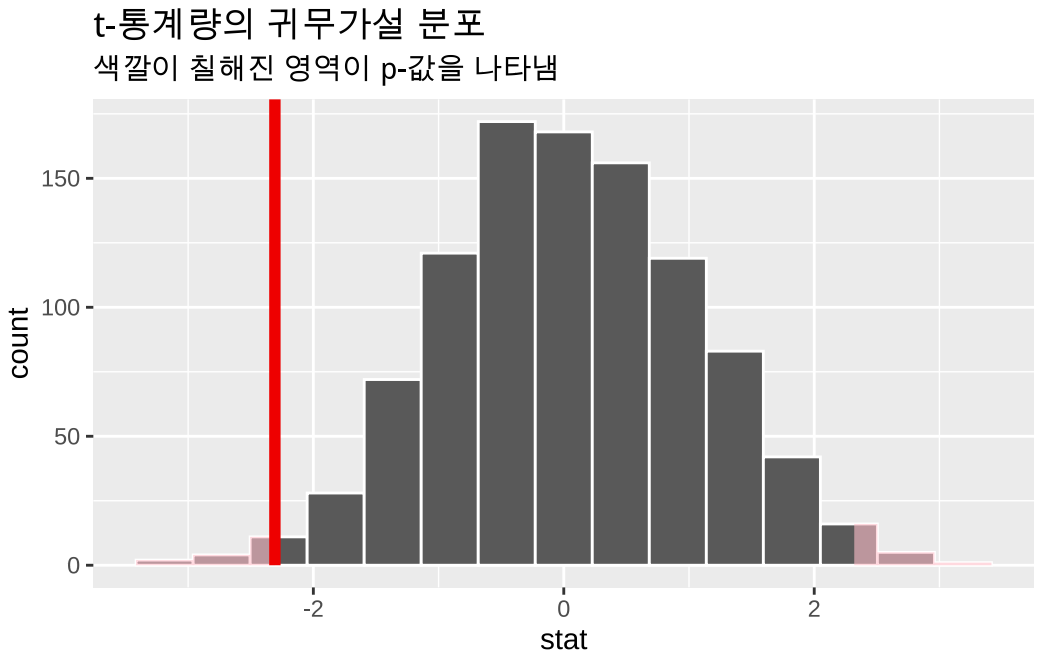
```
Response: converted (numeric)
Explanatory: unit (factor)
# A tibble: 1 x 1
   stat
  <dbl>
1 -2.31
```

다음은 두 집단의 평균이 동일하다는 귀무가설이 참이라고 가정하고, 이때의 분포를 시뮬레이션한다. 이때 `hypothesize()` 함수를 사용하여 귀무가설을 설정하고, `generate()` 함수를 사용하여 시뮬레이션을 수행하고(이 경우에는 1000번의 반복을 수행한다), `calculate()` 함수를 사용하여 각 반복에서 t-통계량을 계산한다. 이 통계량이 분포가 귀무가설이 참일 때의 분포를 나타낸다.

```
null_dist <- df |>
    specify(converted ~ unit) |>
    hypothesize(null = "independence") |>
    generate(reps = 1000, type = "permute") |>
    calculate(stat = "t", order = c("feet", "metres"))
```

`infer` 패키지의 `visualize()` 함수를 사용하여 이 분포를 시각화하고, `shade_p_value()` 함수를 사용하여 관찰된 통계량에 해당하는 p-값을 음영 처리한다.

```
library(showtext)
showtext_auto()
visualize(null_dist) +
    shade_p_value(obs_stat = observed_stat, direction = "two-sided") +
    labs(
        title = "t-통계량의 귀무가설 분포",
        subtitle = "색깔이 칠해진 영역이 p-값을 나타냄"
    )
```



이 그래프에서 음영 처리된 영역이 p-값을 나타낸다. p-값은 관찰된 t-통계량보다 같거나 더 극단적인 t-통계량을 얻을 확률이다.

이제 p-값을 계산해 보자. `infer` 패키지의 `get_p_value()` 함수를 사용하여 p-값을 계산할 수 있다. 이 함수는 관찰된 통계량과 귀무가설 분포를 인자로 받아 p-값을 반환한다.

```
p_value <- null_dist |>
    get_p_value(obs_stat = observed_stat, direction = "two-sided")
p_value
```

```
# A tibble: 1 x 1
  p_value
    <dbl>
1   0.022
```

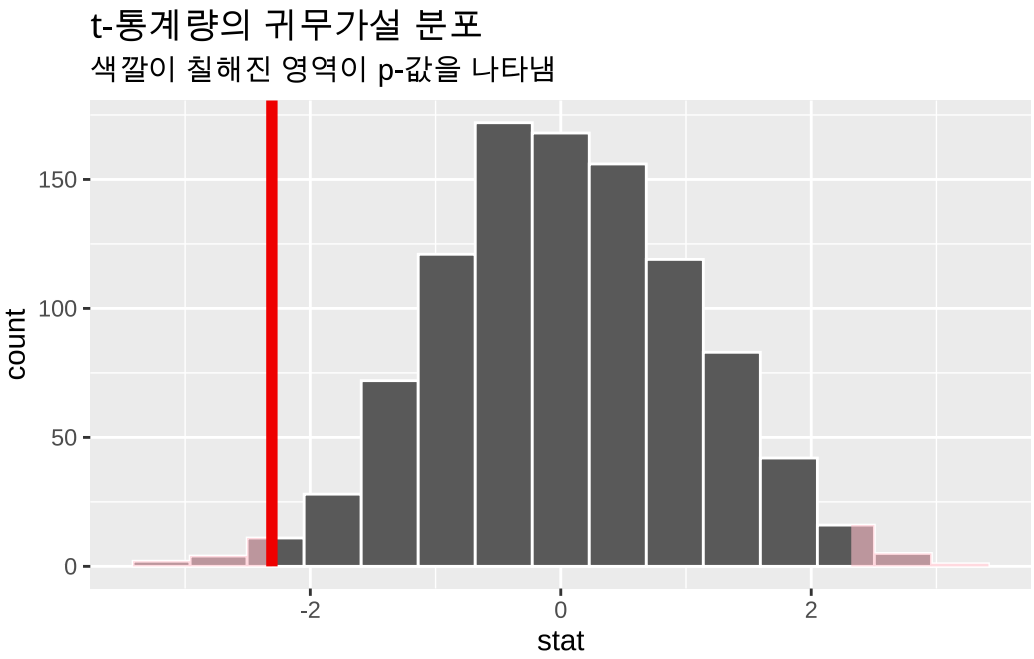
여기서 계산한 p-값은 전통적인 방법으로 계산한 p-값과 거의 동일하다. 전통적인 방법은 수학적인 공식에 따른 것이고, `infer` 패키지는 시뮬레이션을 통해 p-값을 계산하기 때문에 약간은 차이가 있을 수 있다. 그러나 두 방법 모두 p-값의 의미는 동일하다.

28.2 p-값의 의미와 해석

위 `infer` 패키지를 사용한 방법에서 p-값은 어떻게 계산되고 있는지 다시 살펴보자.

먼저 p-값이 어디에서 계산되고 있는가?

```
visualize(null_dist) +
    shade_p_value(obs_stat = observed_stat, direction = "two-sided") +
    labs(
        title = "t-통계량의 귀무가설 분포",
        subtitle = "색깔이 칠해진 영역이 p-값을 나타냄"
    )
```

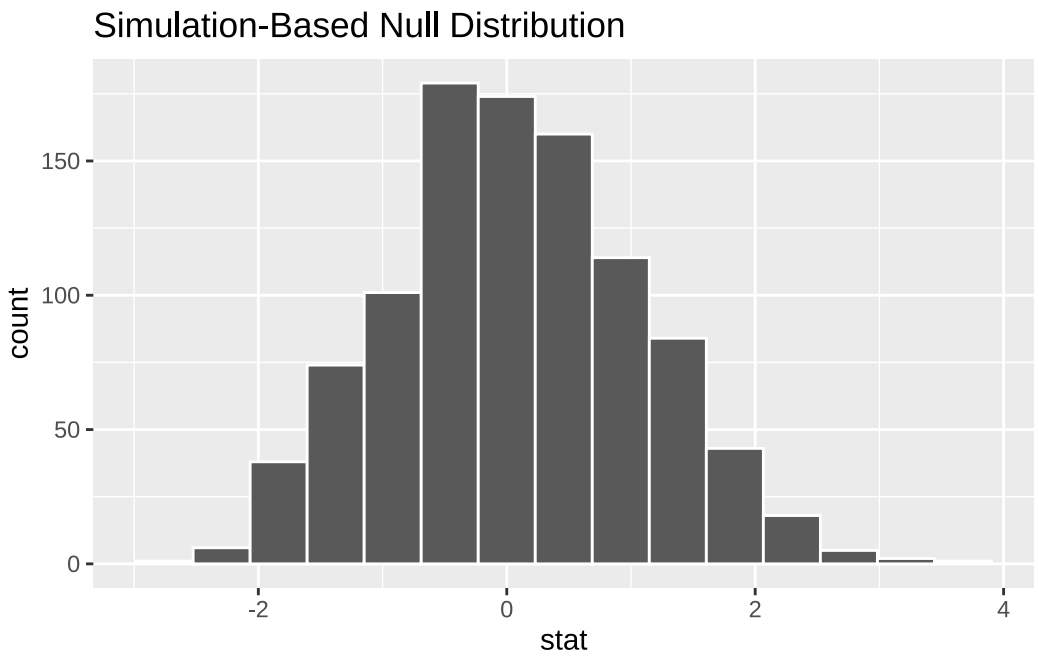


- 일단 p-값은 어떤 확률 분포 곡선의 아래 면적을 나타내는 **확률(probability)** 값이다. 그 면적은 관찰된 통계량보다 같거나 더 극단적인 통계량을 얻을 확률을 나타낸다.
- 그리고 p-값이 결정되는 것은 우리가 관찰할 데이터에서 얻어진 통계량(observed statistic)에 따라서 달라지는 값이다. 강조하고자 하는 것은 우리가 설정한 가설의 참과 거짓을 확률이 아니라는 것이다. 가설의 확률이 아니라 관찰된 통계량(데이터)의 확률인 것이다.

그리고 우리가 p-값을 어떤 확률 분포에서 계산하고 있는가?

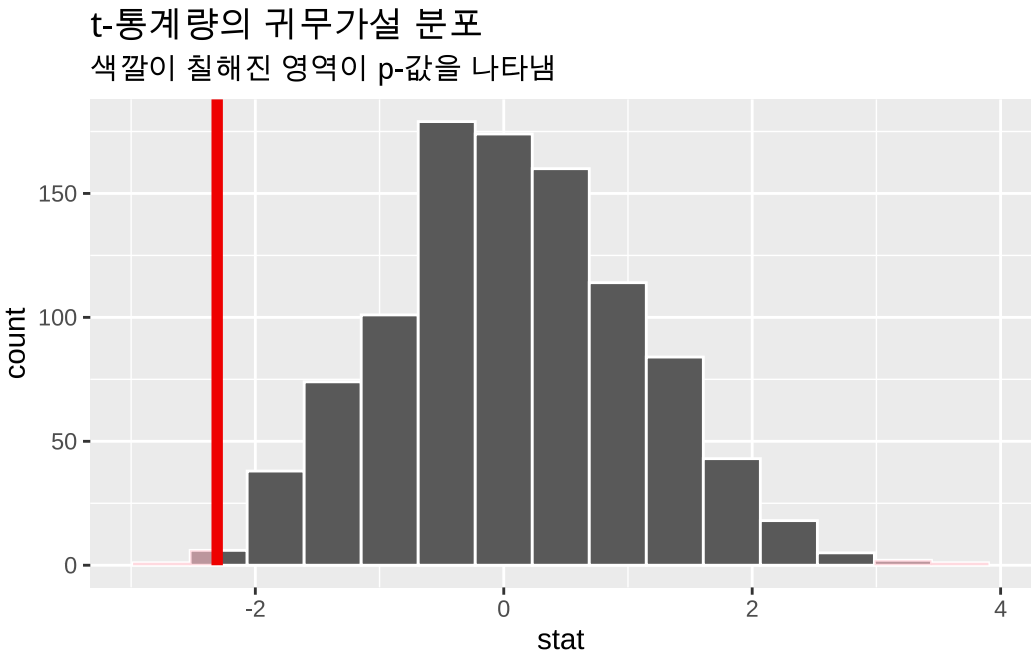
- 다음에서 보는 바와 같이 p-값이 얻어지는 확률 분포는 **귀무가설 참이라고 가정했을 때** 얻어지는 통계량의 분포이다.

```
null_dist <- df |>
    specify(converted ~ unit) |>
    hypothesize(null = "independence") |>
    generate(reps = 1000, type = "permute") |>
    calculate(stat = "t", order = c("feet", "metres"))
visualize(null_dist)
```



이제 p-값의 의미를 알았으니 문제를 다시 살펴보자. 정답이 `f`임을 이해할 수 있을 것이다(모르겠으면 다시 위 내용을 읽어보자).

```
visualize(null_dist) +
    shade_p_value(obs_stat = observed_stat, direction = "two-sided") +
    labs(
        title = "t-통계량의 귀무가설 분포",
        subtitle = "색깔이 칠해진 영역이 p-값을 나타냄"
    )
```



통계학 퀴즈 p-값 0.05가 당신에게 말하는 것은 …

a. 귀무가설(H_0)이 참일 확률 (즉, 효과가 없을 가능성이 5%)
b. 대립가설(H_1)이 참일 확률이 95% (그래서 효과가 실제할 확률이 높다.)
c. 재현될 확률이 95% 이상이다.
d. 결과가 거짓 양성일 확률이 5%이다.
e. 효과의 크기에 관한 어떤 것; 낮은 p-값은 큰 효과를 시사
f. 귀무가설이 참이라고 가정했을 때, 이때 얻어지는 결과보다 같거나 또는 더 극단적인 결과를 얻을 확률
g. 위 목적들의 일부 조합 (어떤 내용인지 명시하세요).

즉, p-값은 귀무가설이 참이라고 가정했을 때, 관찰된 통계량보다 같거나 더 극단적인 통계량을 얻을 확률이다. 따라서 p-값이 작다는 것은 관찰된 통계량이 귀무가설 하에서 발생하기 어려운 값이라는 것을 의미한다.

28.3 정리

아마도 논문 등을 쓸 때는 전통적인 방법으로 p-값을 계산할 것이다. 그렇더라도 p-값의 개념이 흔들릴 때는 `infer` 패키지를 떠올릴 필요가 있다. 그럼 헷갈리지 않을 것이다.

29 두 연속 변수의 평균 비교: t-test

29.1 독립 2-표본 t-test (independent two-sample t-test)

독립 표본 t-test는 두 개의 독립적인 표본의 평균을 비교하는 통계적 방법이다. 이 방법은 두 집단의 **평균**이 통계적으로 유의미하게 다른지를 검정하는 데 사용된다. 예를 들어, 두 가지 다른 치료 방법의 효과를 비교하거나, 두 개의 다른 그룹의 성적을 비교할 때 사용할 수 있다.

독립 표본 t-test의 가설은 다음과 같다:

- **귀무가설(H0)**: 두 표본의 평균이 동일하다. 즉, $H_0 : \mu_1 = \mu_2$.
- **대립가설(H1)**: 두 표본의 평균이 다르다. 즉, $H_1 : \mu_1 \neq \mu_2$.

독립 표본 t-test를 수행하기 위해서는 다음과 같은 조건을 만족해야 한다:

1. 두 표본은 독립적이어야 한다.
2. 각 표본은 정규분포를 따라야 한다(normality).
3. 두 표본의 분산이 동일해야 한다(등분산, equality of variance).
4. 각 표본의 관측치는 서로 독립적(independent)이어야 한다.

> 조건에 대한 부연 설명
>
> - **독립성**: 두 표본은 서로 영향을 주지 않아야 한다. 즉, 한 표본의 관측치가 다른 표본의 관측치에 영향을 미치지 않아야 하고, 같은 표본 안에서 관측치 역시 서로 독립적이어야 한다.
> - **정규성**: 각 표본의 데이터는 정규분포를 따라야 한다. 이는 표본의 크기가 충분히 크면 중심극한정리에 의해 완화될 수 있다.
> - **등분산성**: 두 표본의 분산이 동일해야 한다. 이는 `F-test`나 `Levene's test`를 사용하여 검정할 수 있다.

이와 같은 조건을 만족할 때, 독립 표본 `t-test`의 통계량은 다음과 같이 계산할 수 있다. `t-`통계량(`t-statistic`)은 두 표본의 평균이 차이를 표준화한 값이다.

$$t = \frac{\bar{y_1} - \bar{y_2}}{s\sqrt{\frac{1}{n_1} + \frac{1}{n_2}}}$$

여기서 s는 **pooled standard deviation**(두 집단을 하나로 묶어서 계산한 표준 편차)으로 아래와 같이 계산된다.

$$s = \sqrt{\frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{n_1 + n_2 - 2}}$$

- $\bar{y_1}$과 $\bar{y_2}$는 각각 두 표본의 평균이다.
- s_1과 s_2는 각각 두 표본의 표준편차이다.
- n_1과 n_2는 각각 두 표본의 크기이다.

귀무가설이 참이라고 하면 이 `t`-통계량은 자유도 $df = n_1 + n_2 - 2$를 가진 t-분포를 따른다.

따라서, t-분포의 누적분포함수(CDF)를 사용하여 p-value를 계산할 수 있다.

만약 두 표본의 분산이 동일하지 않다면, **Welch's t-test**를 사용해야 한다. **Welch's t-test**는 등분산성을 가정하지 않고, 각 표본의 분산을 따로 계산하여 `t`-통계량을 계산한다. Welch's t-test의 `t`-통계량은 다음과 같이 계산된다.

$$t = \frac{\bar{y_1} - \bar{y_2}}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}$$

이때, 자유도는 다음과 같이 계산된다.

$$df = \frac{\left(\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}\right)^2}{\frac{\left(\frac{s_1^2}{n_1}\right)^2}{n_1 - 1} + \frac{\left(\frac{s_2^2}{n_2}\right)^2}{n_2 - 1}}$$

따라서 알아둘 내용은

1. 독립 표본 t-test는 두 개의 독립적인 표본의 평균을 비교하는 통계적 방법으로 표본의 정규성을 가정한다.
    - 정규성은 `Shapiro-Wilk test`를 사용하여 검정할 수 있다.
    - 정규성을 만족하지 않는 경우, 비모수적 방법인 `Wilcoxon Mann-Whitney randk sum test`를 사용할 수 있다.
2. 독립 표본 t-통계량을 계산할 때, 등분산을 가정하는 경우와 가정하지 않는 경우가 있다.

    - 등분산을 가정하는 경우 `t-`통계량과 자유도를 계산할 때 `pooled standard deviation`을 사용한다(독립 표본 t-test).
    - 등분산을 가정하지 않는 경우 `t-`통계량과 자유도를 계산할 때 각 표본의 표준편차를 따로 사용한다(Welch's t-test)
    - 등분산은 아래에서 설명하는 바와 같이 `F-test`나 `Levene's test`를 사용하여 검정할 수 있다.
3. 독립 표본이 아닌 경우, 서로 대응되는 두 표본인 경우(paired samples)에는 **대응 표본 t-test**를 사용한다. 다음 절에서 다룬다.
    - 정규성을 만족하지 않는 경우, 비모수적 방법인 `Wilcoxon signed-rank test`를 사용할 수 있다.

29.2 대응 표본 t-test (paired t-test)

대응 표본 t-test는 두 개의 관련된 표본(예: 같은 집단의 전후 측정값)의 평균을 비교하는 통계적 방법이다. 이 방법은 두 표본이 서로 관련되어 있을 때 사용되며, 예를 들어 같은 사람의 치료 전후의 혈압을 비교할 때 사용할 수 있다. 대응 표본 t-test의 가설은 다음과 같다:

- **귀무가설(H0)**: 두 표본의 평균이 동일하다. 즉, $H_0 : \mu_d = 0$.
- **대립가설(H1)**: 두 표본의 평균이 다르다. 즉, $H_1 : \mu_d \neq 0$.

대응 표본 t-test를 수행하기 위해서는 다음과 같은 조건을 만족해야 한다:

1. 두 표본은 관련되어 있어야 한다.
2. 각 표본의 차이는 정규분포를 따라야 한다(normality).
3. 각 표본의 관측치는 서로 독립적(independent)이어야 한다.

이와 같은 조건을 만족할 때, 대응 표본 t-test의 통계량은 다음과 같이 계산할 수 있다. `t-`통계량(`t-statistic`)은 두 표본의 평균 차이를 표준화한 값이다.

$$t = \frac{\bar{d}}{s_d/\sqrt{n}}$$

여기서 $\bar{d}$는 두 표본의 차이의 평균, s_d는 두 표본의 차이의 표준편차, n은 표본의 크기이다. **귀무가설이 참이라고 하면** 이 `t-`통계량은 자유도 $df = n - 1$을 가진 t-분포를 따른다.

29.3 독립 2-표본 t-test 예제: roomwidth

이 예제에서는 두 개의 다른 단위(피트와 미터)로 측정된 방의 너비를 비교한다. HSAUR2 패키지의 roomwidth 데이터를 사용하여, 두 표본의 평균이 통계적으로 유의미하게 다른지를 검정한다.

```
library(HSAUR2)
library(tidyverse)
data("roomwidth", package = "HSAUR2")
glimpse(roomwidth)
```

```
Rows: 113
Columns: 2
$ unit  <fct> metres, metres, metres, metres, metres, metres, metres, metres, ~
$ width <dbl> 8, 9, 10, 10, 10, 10, 10, 10, 11, 11, 11, 11, 12, 12, 13, 13, 13~
```

1미터가 3.28피트이므로, 측정된 방의 너비로 하나의 단위로 맞추어 converted라는 새로운 변수를 만든다.

```
df <- roomwidth |>
    mutate(
        converted = ifelse(
            unit == "feet", 1 * width, 3.28 * width
        )
    )
```

데이터를 요약해 보면 다음과 같다.

```
df |>
    group_by(unit) |>
    summarise(
        mean = mean(converted),
        sd = sd(converted),
        n = n()
    )
```

```
# A tibble: 2 x 4
  unit    mean    sd     n
  <fct>  <dbl> <dbl> <int>
1 feet    43.7  12.5    69
2 metres  52.6  23.4    44
```

박스 플롯을 만들어 보면 다음과 같다.

```
df |>
    ggplot(aes(x = unit, y = converted)) +
    geom_boxplot() +
    labs(
        x = "Unit",
        y = "Converted Width (meters)",
        title = "Boxplot of Room Widths by Unit"
    ) +
    theme_minimal()
```

q-q plot을 사용하여 두 표본의 정규성을 확인할 수 있다.

```
df |>
    ggplot(aes(sample = converted, color = unit)) +
    stat_qq() +
    stat_qq_line() +
    labs(
        title = "Q-Q Plot of Room Widths by Unit",
        x = "Theoretical Quantiles",
        y = "Sample Quantiles"
    ) +
    theme_minimal()
```

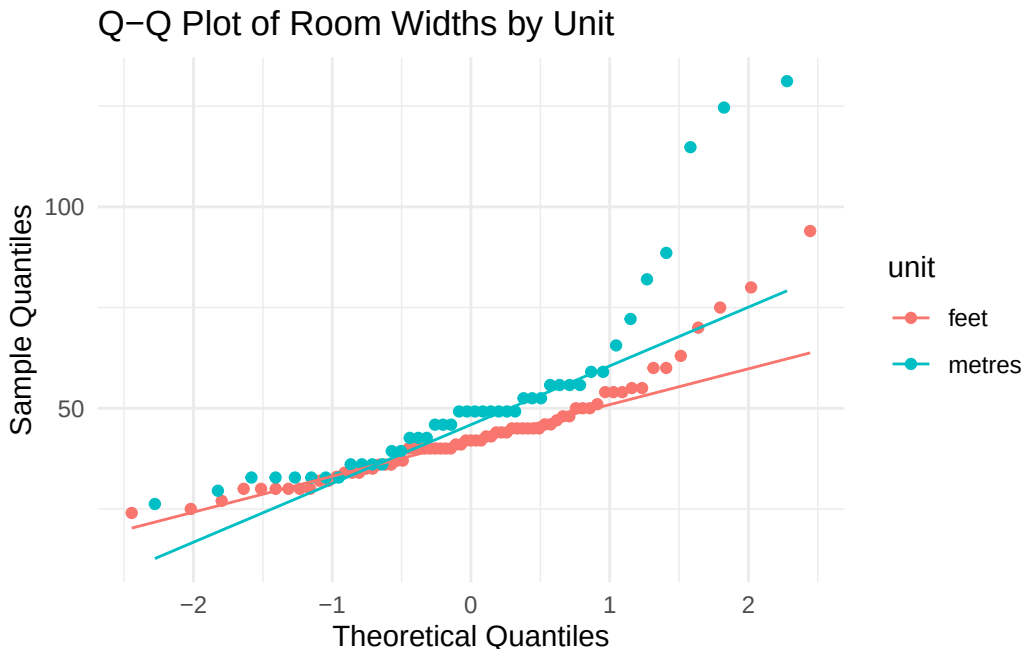


그림 29.1: 정규성 가정이 만족되지 않을 수 있음을 시사한다.

이제 가설 검정을 수행해 볼 차례이다. 먼저 가정들을 확인해야 한다.

`Shapiro-Wilk test`를 사용하여 두 표본의 정규성을 검정할 수 있다.

```
shapiro_test_result <- df |>
    group_by(unit) |>
    summarise(
```

```
        shapiro_p_value = shapiro.test(converted)$p.value
    )
shapiro_test_result
```

```
# A tibble: 2 x 2
  unit   shapiro_p_value
  <fct>            <dbl>
1 feet        0.0000131
2 metres      0.000000566
```

Shapiro-Wilk test은 표본이 정규분포를 따른다는 귀무가설을 검정한다. 따라서 이 경우 p-value가 0.05보다 작아서 귀무가설을 기각하고, 두 표본 모두 정규성을 만족하지 않는다고 결론지을 수 있다.

다음은 등분산을 검정하는 방법이다.

Levene's test를 사용하여 두 표본의 등분산성을 검정할 수 있다.

```
library(car)
levene_test_result <- leveneTest(converted ~ unit, data = df)
levene_test_result
```

또는 F-test를 사용하여 두 표본의 등분산성을 검정할 수 있다.

```
f_test_result <- var.test(converted ~ unit, data = df)
f_test_result
```

Levene's test와 F-test의 p-value가 0.05보다 작아서 두 표본의 분산이 동일하다고 가정할 수 없다.

따라서 이 경우에는 표본이 정규성과 등분산성을 만족하지 않으므로, 비모수적 방법인 Wilcoxon Mann-Whitney rank sum test를 사용하는 것이 바람직하다. R에서는 wilcox.test를 사용하여 두 표본의 평균이 통계적으로 유의미하게 다른지 검정할 수 있다.

```
wilcox_test_result <- wilcox.test(
    converted ~ unit,
    data = df,
    conf.int = TRUE
```

```
)
wilcox_test_result
```

```
    Wilcoxon rank sum test with continuity correction

data:  converted by unit
W = 1145, p-value = 0.02815
alternative hypothesis: true location shift is not equal to 0
95 percent confidence interval:
 -9.3599953 -0.8000423
sample estimates:
difference in location
             -5.278598
```

29.3.1 만약 두 표본이 정규성을 만족했다면…

두 표본이 정규성을 만족했다면 다음 2가지 방법으로 독립 표본 t-test를 수행할 수 있다.

1. 등분산을 가정하지 않는 경우: Welch's t-test를 수행할 수 있다(`default`).
2. 등분산을 가정하는 경우: 독립 표본 t-test를 수행할 수 있다(`var.equal = TRUE`).

등분산을 가정하지 않는 경우 **Welch's t-test**를 수행할 수 있다. R에서 `t.test()` 함수에 `var.equal = FALSE` (default)옵션을 사용하여 Welch's t-test를 수행할 수 있다.

```
t_test_result_welch <- t.test(
    converted ~ unit,
    data = df,
    var.equal = FALSE
)
t_test_result_welch
```

```
    Welch Two Sample t-test
```

```
data:  converted by unit
t = -2.3071, df = 58.788, p-value = 0.02459
alternative hypothesis: true difference in means between group feet and group metres is not equal
95 percent confidence interval:
 -16.54308  -1.17471
sample estimates:
  mean in group feet mean in group metres
             43.69565              52.55455
```

등분산을 가정할 수 있는 경우에는 독립 표본 t-test를 수행할 수 있다. R에서 `t.test()` 함수에 `var.equal = TRUE` 옵션을 사용하여 독립 표본 t-test를 수행할 수 있다.

```
t_test_result <- t.test(
    converted ~ unit,
    data = df,
    var.equal = TRUE
)
t_test_result
```

```
    Two Sample t-test

data:  converted by unit
t = -2.6147, df = 111, p-value = 0.01017
alternative hypothesis: true difference in means between group feet and group metres is not equal
95 percent confidence interval:
 -15.572734  -2.145052
sample estimates:
  mean in group feet mean in group metres
             43.69565              52.55455
```

29.4 대응 표본 t-test(paired t-test) 예제: waves 데이터셋

HSAUR2 패키지의 waves 데이터셋을 사용하여 대응 표본 t-test를 수행한다. 이 데이터셋은 조력 발전 시뮬레이션 데이터라고 했는데, 각 행의 한 사람이라고 봐도 무방하다. 한 사람에 대해 두 가지 방법을 적용한 결과라 생각하면 된다.

```
data("waves", package = "HSAUR2")
head(waves, n = 10)
```

```
   method1 method2
1     2.23    1.82
2     2.55    2.42
3     7.99    8.26
4     4.09    3.46
5     9.62    9.77
6     1.59    1.40
7     8.98    8.88
8     0.82    0.87
9    10.83   11.20
10    1.54    1.33
```

이 경우는 한 사람에 대해 두 가지 방법을 적용한 결과이므로, 대응 표본 t-test를 수행할 수 있다.

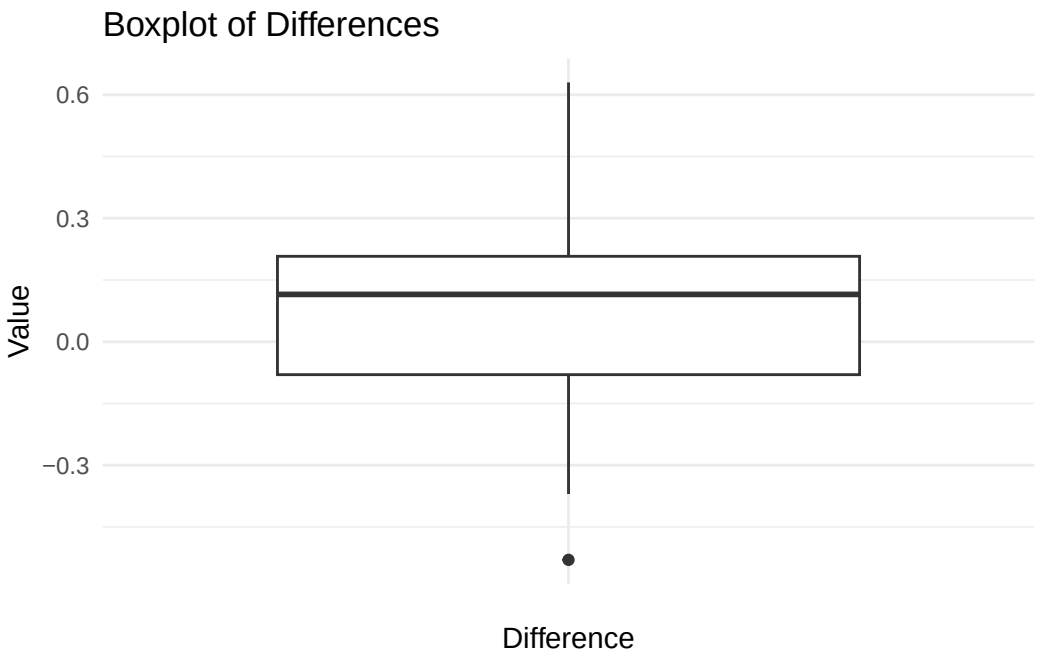
값의 차이를 구하자.

```
waves <- waves |>
    mutate(difference = method1 - method2)
```

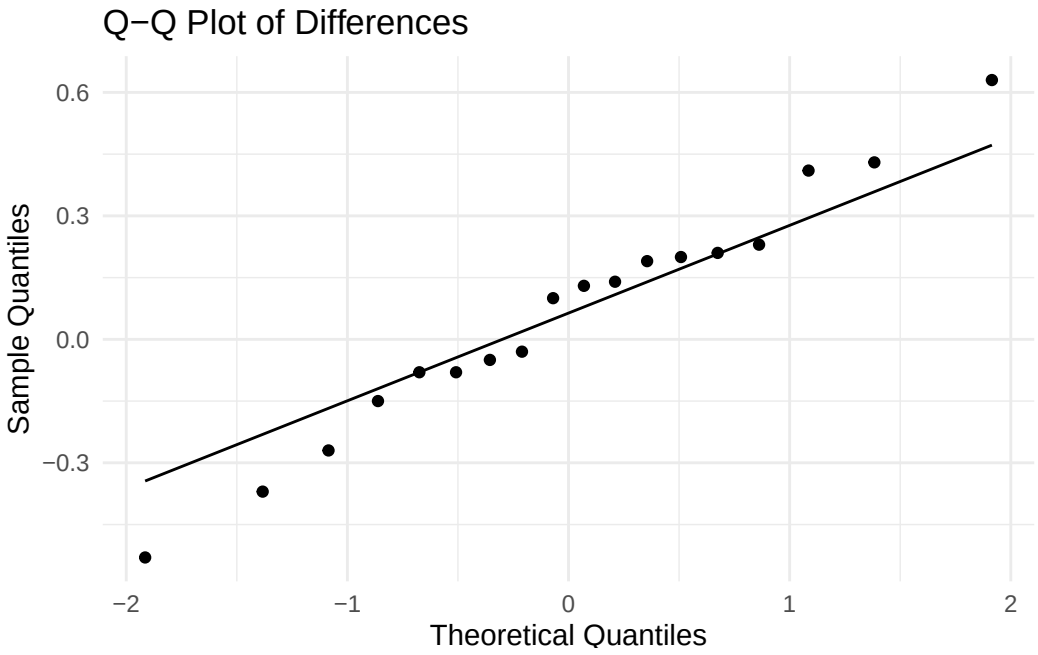
박스 플롯과 q-q plot을 사용하여 정규성을 확인할 수 있다.

```
waves |>
    ggplot(aes(x = "", y = difference)) +
    geom_boxplot() +
    labs(
        x = "Difference",
        y = "Value",
```

```
        title = "Boxplot of Differences"
    ) +
    theme_minimal()
```




```
waves |>
    ggplot(aes(sample = difference)) +
    stat_qq() +
    stat_qq_line() +
    labs(
        title = "Q-Q Plot of Differences",
        x = "Theoretical Quantiles",
        y = "Sample Quantiles"
    ) +
    theme_minimal()
```



Shapiro-Wilk test를 사용하여 정규성을 검정할 수 있다.

```
shapiro_test_result <- shapiro.test(waves$difference)
shapiro_test_result
```

```
    Shapiro-Wilk normality test

data:  waves$difference
W = 0.98346, p-value = 0.9785
```

Shapiro-Wilk test의 p-value가 0.05보다 크므로, 귀무가설을 기각하지 못하고, 데이터가 정규분포를 따른다고 결론지을 수 있다. 따라서 대응 표본 t-test를 수행할 수 있다. R에서 `t.test()` 함수를 사용하여 대응 표본 t-test를 수행할 수 있다.

```
t_test_result <- t.test(
    waves$method1,
    waves$method2,
    paired = TRUE
)
```

```
t_test_result
```

```
    Paired t-test

data:  waves$method1 and waves$method2
t = 0.90193, df = 17, p-value = 0.3797
alternative hypothesis: true mean difference is not equal to 0
95 percent confidence interval:
 -0.08258476  0.20591810
sample estimates:
mean difference
     0.06166667
```

또는 다음과 같이 `difference` 변수를 사용하여 대응 표본 t-test를 수행할 수 있다.

```
t_test_result <- t.test(
    waves$difference,
    mu = 0, # 귀무가설에서 차이가 0이라고 가정
    alternative = "two.sided" # 양측 검정
)
t_test_result
```

```
    One Sample t-test

data:  waves$difference
t = 0.90193, df = 17, p-value = 0.3797
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 -0.08258476  0.20591810
sample estimates:
 mean of x
0.06166667
```

29.4.1 만약 정규성을 만족하지 않는다면…

정규성을 만족하지 않는 경우, 비모수적 방법인 `Wilcoxon signed-rank test`를 사용할 수 있다. R에서 `wilcox.test()` 함수를 사용하여 대응 표본의 순위 합 검정을 수행할 수 있다.

```
wilcox_test_result <- wilcox.test(
    waves$method1,
    waves$method2,
    paired = TRUE
)
```

```
Warning in wilcox.test.default(waves$method1, waves$method2, paired = TRUE):
cannot compute exact p-value with ties
```

```
wilcox_test_result
```

```

    Wilcoxon signed rank test with continuity correction

data:  waves$method1 and waves$method2
V = 109, p-value = 0.3165
alternative hypothesis: true location shift is not equal to 0
```

또는 다음과 같이 `difference` 변수를 사용하여 대응 표본의 순위 합 검정을 수행할 수 있다.

```
wilcox_test_result <- wilcox.test(
    waves$difference,
    mu = 0, # 귀무가설에서 차이가 0이라고 가정
    alternative = "two.sided" # 양측 검정
)
```

```
Warning in wilcox.test.default(waves$difference, mu = 0, alternative =
"two.sided"): cannot compute exact p-value with ties
```

```
wilcox_test_result
```

```
    Wilcoxon signed rank test with continuity correction

data:  waves$difference
V = 109, p-value = 0.3165
alternative hypothesis: true location is not equal to 0
```

30 두 연속 변수의 상관관계: correlation

```
library(tidyverse)
```

상관(Correlation)은 두 연속 변수 간의 선형 관계를 측정하는 방법이다. 상두 연속 변수의 상관관계는 보통 Pearson 상관계수(Pearson correlation coefficient)를 사용하여 측정하는데, 이는 두 변수 간의 선형 관계의 강도와 방향을 나타낸다. Pearson 상관계수는 -1에서 1 사이의 값을 가지며, 0은 두 변수 간에 선형 관계가 없음을 의미한다. 양의 값은 두 변수가 함께 증가함을 나타내고, 음의 값은 한 변수가 증가할 때 다른 변수가 감소함을 나타낸다.

보통 다음 수식에 의해서 계산된다:

$$r = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2 \sum(y_i - \bar{y})^2}} \tag{30.1}$$

위에서 x_i와 y_i는 각각 두 변수의 관측치, $\bar{x}$와 $\bar{y}$는 각각 두 변수의 평균이다.

이것은 두 변수의 공분산(covariance)을 각 변수의 표준편차의 곱으로 나눈, 정규화된 값을 나타낸다.

30.1 편차(deviation), 공분산(covariance), 상관계수(correlation)에 대한 이해

상관관계를 어떤 한 변수가 이렇게 바뀌는 데 다른 변수가 어떻게 바뀌는지를 나타내는 지표이다. 상관관계를 이해하기 위해서는 먼저 편차(deviation), 공분산(covariance), 그리고 상관계수(correlation)에 대해 알아야 한다.

편차는 각 관측치가 평균으로부터 얼마나 떨어져 있는지를 나타내는 값이다. 편차는 다음과 같이 계산된다:

$$d_i = x_i - \bar{x}$$

간단한 R 코드로 계산해 보자. 먼저, 두 변수 `x`와 `y`를 생성하고, 각각의 편차를 계산해 보자.

```r
set.seed(42)
x <- sample(1:10, 10, replace = TRUE)
y <- x + round(rnorm(10, mean = 0, sd = 2), 0)
```

위 코드는 임의의 수를 만들기 위한 것으로 크게 관심을 가질 필요는 없다. 다음과 같은 두 변수 x와 y가 있다고 가정하자.

```r
x
```

```
 [1]  1  5  1  9 10  4  2 10  1  8
```

```r
y
```

```
 [1]  4  5  5  9 13  9 -1  9  1  9
```

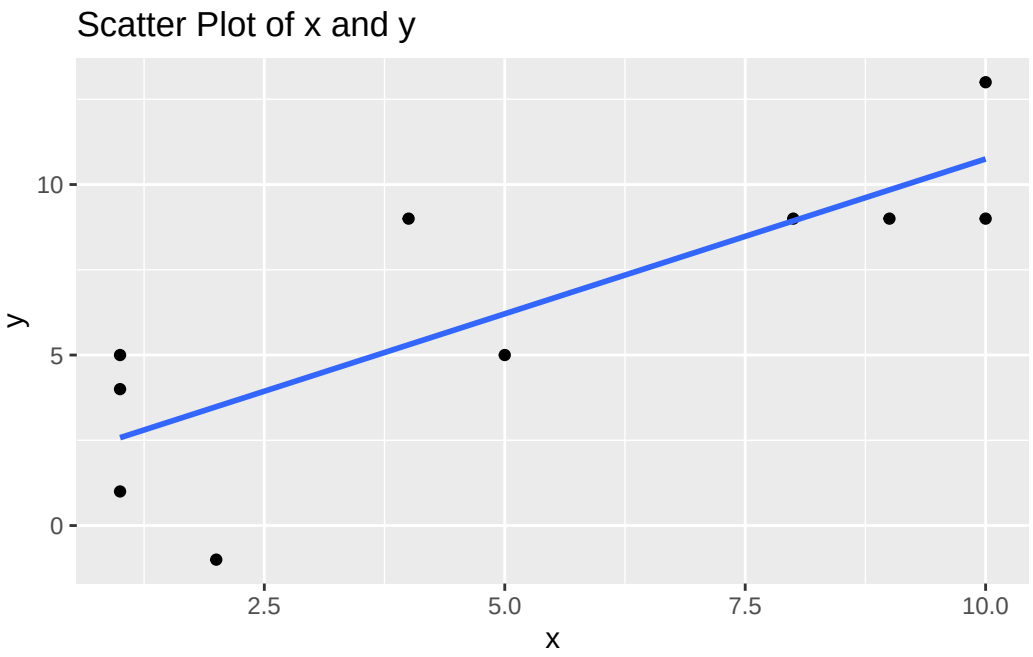
두 연속 변수의 산점도를 그려보자.

```r
dx <- data.frame(x)
dy <- data.frame(y)
df <- cbind(dx, dy)
df
```

```
    x  y
1   1  4
2   5  5
3   1  5
4   9  9
5  10 13
6   4  9
7   2 -1
8  10  9
9   1  1
10  8  9
```

```
df |>
    ggplot(aes(x = x, y = y)) +
    geom_point() +
    geom_smooth(method = "lm", se = FALSE) +
    labs(
        title = "Scatter Plot of x and y",
        x = "x",
        y = "y"
    )
```

```
`geom_smooth()` using formula = 'y ~ x'
```



편차(deviation)을 계산해 보자. 편차는 각 관측치가 평균으로부터 얼마나 떨어져 있는지를 나타낸다.

```
deviation_x <- x - mean(x)
deviation_y <- y - mean(y)
```

```
deviation_x
```

```
 [1] -4.1 -0.1 -4.1  3.9  4.9 -1.1 -3.1  4.9 -4.1  2.9
```

```
deviation_y
```

```
 [1] -2.3 -1.3 -1.3  2.7  6.7  2.7 -7.3  2.7 -5.3  2.7
```

공분산(covariance)은 두 변수 간의 편차의 곱의 평균이다.

```
covariance <- sum(deviation_x * deviation_y) / (length(x) - 1)
covariance
```

```
[1] 13.41111
```

`cov()` R 함수를 사용하여 계산할 수도 있다.

```
cov(x, y)
```

```
[1] 13.41111
```

즉 `deviation_x`와 `deviation_y`의 요소끼리 서로 곱한 후, 그 평균을 구한 것이다.

이런 공분산을 통해서 변수 `x`와 `y`가 얼마나 함께 변하는지를 알 수 있다. 공분산이 양수이면 두 변수가 함께 증가하거나 감소하는 경향이 있음을 나타내고, 음수이면 한 변수가 증가할 때 다른 변수가 감소하는 경향이 있음을 나타낸다.

즉, 공분산은 두 변수 간의 편차의 곱의 평균을 나타내며, 두 변수 간의 관계를 측정하는 데 사용된다. 공분산은 다음과 같이 계산된다:

$$cov(X, Y) = \frac{1}{n-1} \sum (x_i - \bar{x})(y_i - \bar{y})$$

위 예에서 `covariance`는 `12.0`이었다. 이 값이 양수이기 때문에 `x`와 `y`가 함께 증가하는 것을 알았다. 그런데 공분산의 단점이 있다. 바로 공분산의 값이 두 변수의 측정 단위에 의존한다는 것이다. 즉, 두 변수의 단위가 다르면 공분산의 값도 달라진다. 예를 들어, `x`가 센티미터 단위이고, `y`가 미터 단위라면, 공분산의 값은 두 변수의 단위에 따라 달라질 것이다. 따라서, 공분산의 값을 해석하기가 어렵다. 그래서 공분산을 표준화한 상관계수(correlation coefficient)를 사용한다. 이 값을 -1에서 1‘ 사이의 값으로 정규화하여, 두 변수 간의 관계를 더 쉽게 해석할 수 있다.

상관계수는 공분산을 각 변수의 표준편차의 곱으로 나눈 값이다. 상관계수는 다음과 같이 계산된다:

$$r = \frac{cov(X,Y)}{\sigma_X \sigma_Y} = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2}\sqrt{\sum(y_i - \bar{y})^2}}$$

R에서는 `cor()` 함수를 사용하여 상관계수를 계산할 수 있다. 먼저 위에서 계산한 `covariance`를 사용하여 상관계수를 계산해 보자.

```
correlation <- covariance / (sd(x) * sd(y))
correlation
```

```
[1] 0.8173169
```

```
cor(x, y)
```

```
[1] 0.8173169
```

30.2 두 연속 변수와의 상관관계 예제: `water` 데이터셋

이번에는 두 연속 변수 간의 상관관계를 살펴보자. `HSAUR2` 패키지의 `water` 데이터셋을 사용하여, 물의 경도(hardness)와 사망률(mortality) 간의 상관관계를 분석한다.

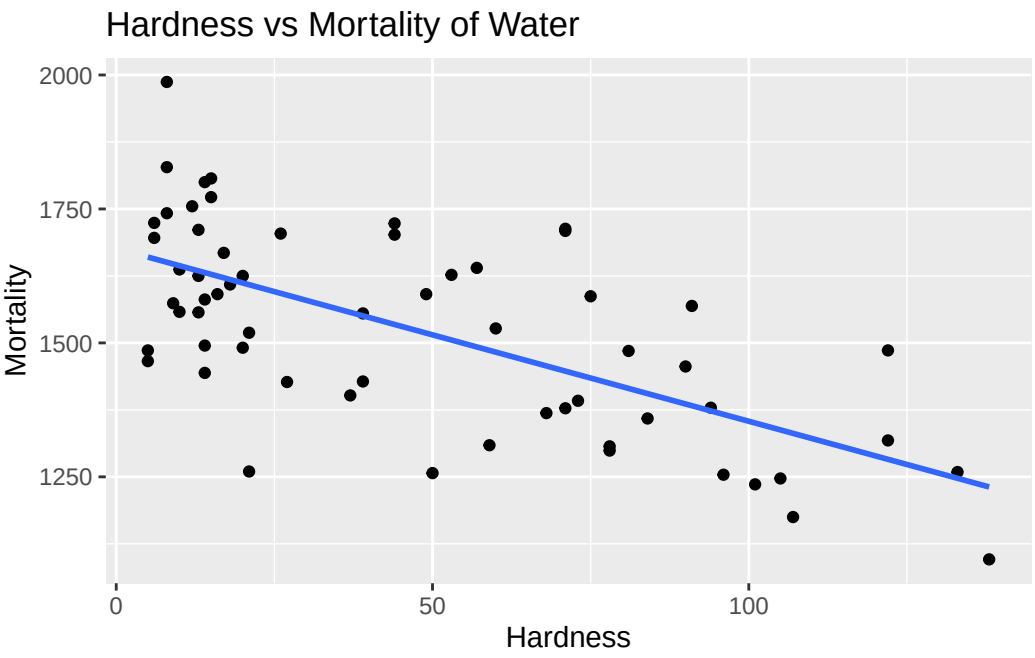
```
library(tidyverse)
library(HSAUR2)
data("water", package = "HSAUR2")
head(water)
```

```
  location       town mortality hardness
1    South       Bath      1247      105
2    North Birkenhead      1668       17
3    South Birmingham      1466        5
4    North  Blackburn      1800       14
5    North  Blackpool      1609       18
6    North     Bolton      1558       10
```

30.2.1 산점도와 회귀곡선

```
water |>
    ggplot(aes(x = hardness, y = mortality)) +
    geom_point() +
    geom_smooth(method = "lm", se = FALSE) +
    labs(
        title = "Hardness vs Mortality of Water",
        x = "Hardness",
        y = "Mortality"
    )
```

```
`geom_smooth()` using formula = 'y ~ x'
```



30.2.2 피어슨 상관계수

Hardness와 Mortality의 피어슨 상관계수를 계산해 보자.

```
cor(water$hardness, water$mortality)
```

```
[1] -0.6548486
```

30.3 상관관계에 대한 가설 검정

상관관계에 대한 가설 검정을 수행해 보자. 귀무가설은 두 변수 간에 상관관계가 없다는 것이다.

상관계수의 유의성을 검정하기 위해 `cor.test()` 함수를 사용할 수 있다.

사용되는 통계량은 다음과 같이 정의된다:

$$t = \frac{r\sqrt{n-2}}{\sqrt{1-r^2}}$$

여기서 r은 샘플 상관계수, n은 관측치의 수이다.

이 통계량은 상관계수가 0이고, bivatiate 정규분포를 따른다면, 이 통계량은 자유도 $df = n - 2$를 가지는 t-분포를 따른다.

```
cor.test(water$hardness, water$mortality)
```

```
    Pearson's product-moment correlation

data:  water$hardness and water$mortality
t = -6.6555, df = 59, p-value = 1.033e-08
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.7783208 -0.4826129
sample estimates:
       cor
-0.6548486
```

30.4 두 연속 변수와의 순위 상관관계: 스피어만 상관계수

스피어만(Spearman) 상관계수는 두 변수 간의 **순위(rank)** 상관관계를 측정하는 방법이다. 이는 두 변수의 값이 아닌 순위를 기반으로 계산되며, 비모수적(non-parametric) 방법으로, 데이터가 정규분포를 따르지 않을 때 유용하다. 스피어만 상관계수는 다음과 같이 계산된다:

$$\rho = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)}$$

위에서 d_i는 두 변수의 순위 차이, n은 관측치의 수이다.

매뉴얼로 위 x와 y의 스피어만 상관계수를 계산해 보자. 먼저 두 변수의 순위를 계산한다.

```
rank_x <- rank(x)
rank_y <- rank(y)
```

```
rank_x
```

```
[1] 2.0 6.0 2.0 8.0 9.5 5.0 4.0 9.5 2.0 7.0
```

```
rank_y
```

```
[1]  3.0  4.5  4.5  7.5 10.0  7.5  1.0  7.5  2.0  7.5
```

```
d <- rank_x - rank_y
d
```

```
[1] -1.0  1.5 -2.5  0.5 -0.5 -2.5  3.0  2.0  0.0 -0.5
```

```
d_squared <- d^2
d_squared
```

```
[1] 1.00 2.25 6.25 0.25 0.25 6.25 9.00 4.00 0.00 0.25
```

```r
n <- length(x)
spearman_correlation <- 1 - (6 * sum(d_squared)) / (n * (n^2 - 1))
spearman_correlation
```

```
[1] 0.8212121
```

스피어만 상관계수는 `cor()` 함수의 `method = "spearnman"` 인자를 사용하여 게산할 수 있다.

```r
cor(x, y, method = "spearman")
```

```
[1] 0.8122502
```

이 값에 대한 가설 검정을 수행할 수 있다. 귀무가설은 두 변수 간에 순위 상관관계가 없다는 것이다. `cor.test()` 함수의 `method = "spearman"` 인자를 사용하여 스피어만 상관계수에 대한 가설 검정을 수행할 수 있다.

```r
cor.test(x, y, method = "spearman")
```

```
Warning in cor.test.default(x, y, method = "spearman"): Cannot compute exact
p-value with ties


    Spearman's rank correlation rho

data:  x and y
S = 30.979, p-value = 0.004305
alternative hypothesis: true rho is not equal to 0
sample estimates:
      rho
0.8122502
```

30.5 켄달의 타우(Kendall's Tau)

켄달의 타우(Kendall's Tau)는 두 변수 간의 **순위 상관관계**를 측정하는 또 다른 방법이다. 이 방법은 두 변수의 순위 간의 일치와 불일치를 기반으로 계산되는데 일치(concordant)와 불일치(discordant) 쌍의 수를 사용한다. 보통 일치되는 순위가 많은 경우 사용된다.

R에서 켄달의 타우를 계산하려면 `cor()` 함수의 `method = "kendall"` 인자를 사용한다. 먼저, 위에서 생성한 `x`와 `y`의 켄달의 타우를 계산해 보자.

```
cor(x, y, method = "kendall")
```

```
[1] 0.658703
```

30.6 `infer` 패키지를 사용한 상관관계 분석

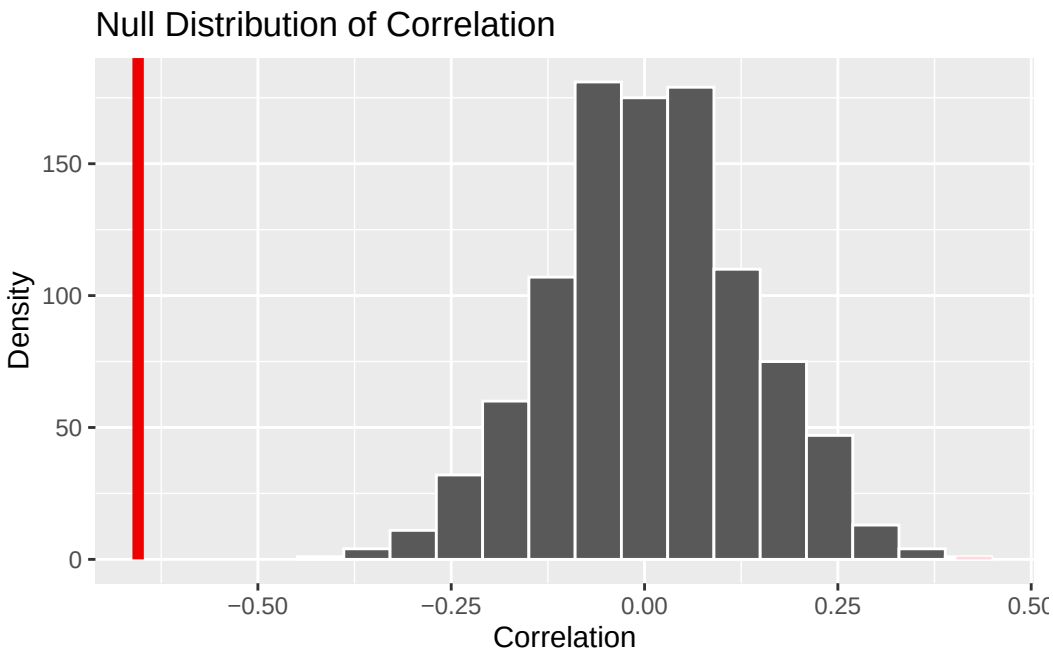
```
library(infer)
```

```
obs_hat <- water |>
    specify(mortality ~ hardness) |>
    calculate(stat = "correlation")
obs_hat
```

```
Response: mortality (numeric)
Explanatory: hardness (numeric)
# A tibble: 1 x 1
    stat
   <dbl>
1 -0.655
```

```
null_dist <- water |>
    specify(mortality ~ hardness) |>
    hypothesize(null = "independence") |>
    generate(reps = 1000, type = "permute") |>
    calculate(stat = "correlation")
```

```r
visualize(null_dist) +
    shade_p_value(obs_stat = obs_hat, direction = "two-sided") +
    labs(
        title = "Null Distribution of Correlation",
        x = "Correlation",
        y = "Density"
    )
```




```r
null_dist |>
    get_p_value(obs_stat = obs_hat, direction = "two-sided")
```

```
Warning: Please be cautious in reporting a p-value of 0. This result is an approximation
based on the number of `reps` chosen in the `generate()` step.
i See `get_p_value()` (`?infer::get_p_value()`) for more information.

# A tibble: 1 x 1
  p_value
    <dbl>
1       0
```

31 분산 분석의 기초: 제곱합(sum of squares)을 중심으로

(**book?**){lazic2016experimental, 이 장에서는 분산 분석(ANOVA)에서 제곱합(sum of squares)에 대해 설명한다. 제곱합은 분산 분석에서 매우 중요한 개념이다. 다음은 (Lazic 2016) 책에서 인용한 예이다.

```
library(tidyverse)
library(showtext)
font_add_google("Nanum Gothic", "nanumgothic")
showtext_auto()
```

간단한 데이터셋을 만들어 보자. x라는 팩터 변수가 있고, y라는 결과 변수(종속 변수)가 있다.

```
y <- c(6, 2, 3, 1, 4, 8, 7, 9)
x <- factor(rep(c("A", "B"), each = 4))
df <- tibble(y, x)
df
```

```
# A tibble: 8 x 2
      y x
  <dbl> <fct>
1     6 A
2     2 A
3     3 A
4     1 A
5     4 B
6     8 B
7     7 B
```

```
8     9 B
```

31.1 전체 제곱합(total sum of squares)

값들의 변동(variation)을 보기 위해서, 모든 관측치에 대해 전체 평균까지의 거리를 계산한 그래프를 그려보자.

```
df <- df %>%
    mutate(id = row_number())
df
```

```
# A tibble: 8 x 3
      y x        id
  <dbl> <fct> <int>
1     6 A         1
2     2 A         2
3     3 A         3
4     1 A         4
5     4 B         5
6     8 B         6
7     7 B         7
8     9 B         8
```

```
ggplot(df, aes(id, y)) +
    geom_point() +
    geom_hline(yintercept = mean(y), color = "red") +
    geom_segment(aes(x = id, xend = id, y = y, yend = mean(y)), color = "steelblue") +
    scale_x_continuous(breaks = 1:8, labels = x) +
    labs(
        x = "group x",
        title = "각 관측치에서 평균까지의 거리"
    ) +
    theme_minimal()
```

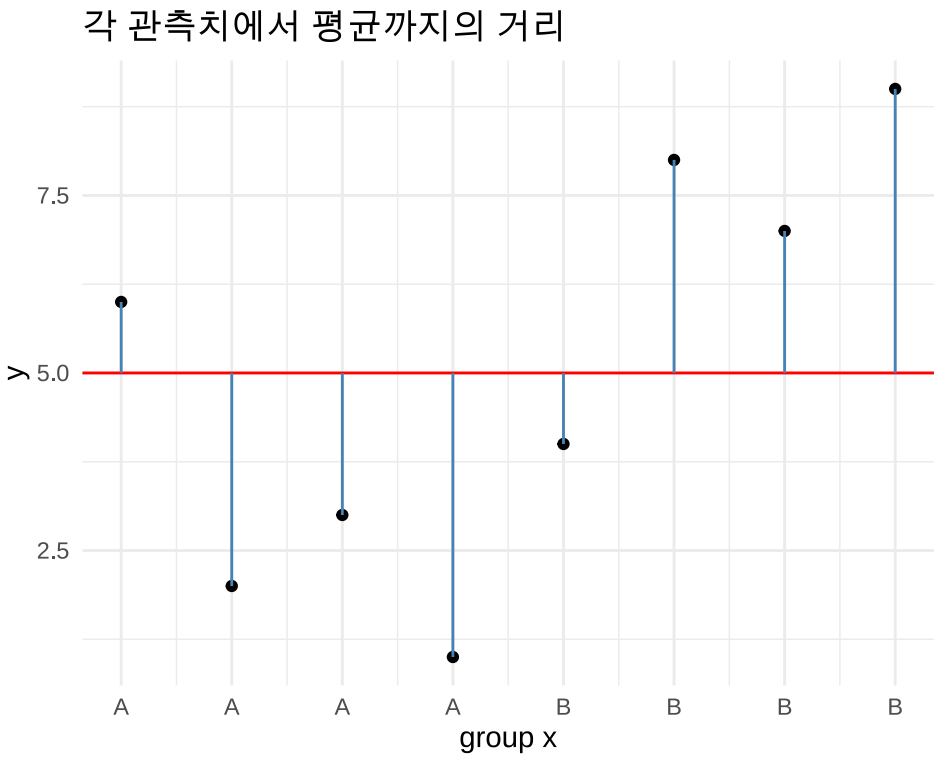


그림 31.1: 각 관측치에서 평균(빨간 선)까지의 거리(파란 선)

그래프에서 평균까지의 거리는 파란색 선으로 표시했다. 수식으로 표현하면 다음과 같다.

$$y_i - \bar{y}$$

이 값을 통계에서는 편차(deviation)라고 부른다. 편차는 관측치와 평균의 차이를 의미한다. 이런 편차의 합은 항상 0이 된다.

```
sum(df$y - mean(df$y))
```

```
[1] 0
```

통계학에서는 이 값을 사용하기 위해서 제곱을 취한다. 대표적인 경우가 분산(variance)이다. 분산은 편차의 제곱을 평균한 값이다. 수식으로 보면 다음과 같다.

분산은 다음과 같이 계산한다.

$$\frac{1}{n-1}\sum_{i=1}^{n}(y_i - \bar{y})^2$$

```
dev_y_squared <- (df$y - mean(df$y))^2
dev_y_squared
```

```
[1]  1  9  4 16  1  9  4 16
```

```
sum(dev_y_squared) / (length(df$y) - 1)
```

```
[1] 8.571429
```

`var()` 함수를 사용하면 분산을 바로 계산할 수 있다.

```
var(df$y)
```

```
[1] 8.571429
```

여기서 우리는 제곱합(sum of squares)을 알아 보려고 한다. 제곱합은 편차의 제곱을 모두 더한 값이다. 수식으로 표현하면 다음과 같은데 분산과 다른 점은 뭔가로 나누지 않았다는 점이다. 위 그래프에서 파란색 선의 길이를 제곱하여 더하면 제곱합이 된다.

$$\sum_{i=1}^{n}(y_i - \bar{y})^2$$

```
sum(dev_y_squared)
```

```
[1] 60
```

31.2 그룹별 평균을 고려한 제곱합(Residual Sum of Squares)

위 그래프는 x 요인을 고려하지 않고, 전체 그룹에서의 데이터의 변동을 보여주고 있다. 이제 그룹별로 데이터의 변동을 보기 위해서 **그룹별로 평균**을 계산하고, 평균까지의 거리를 계산해 보자.

```r
ddf <- df %>%
    group_by(x) %>%
    mutate(mean_y = mean(y))
ddf
```

```
# A tibble: 8 x 4
# Groups:   x [2]
      y x        id mean_y
  <dbl> <fct> <int>  <dbl>
1     6 A         1      3
2     2 A         2      3
3     3 A         3      3
4     1 A         4      3
5     4 B         5      7
6     8 B         6      7
7     7 B         7      7
8     9 B         8      7
```

그룹의 평균에 대해서 각 관측치까지의 거리를 그려보자.

```r
ggplot(ddf, aes(id, y)) +
    geom_point() +
    geom_segment(aes(x = id, xend = id, y = y, yend = mean_y), color = "steelblue") +
    annotate("segment", x = 1, xend = 4, y = 3, yend = 3, color = "red") +
    annotate("segment", x = 5, xend = 8, y = 7, yend = 7, color = "orange") +
    scale_x_continuous(breaks = 1:8, labels = x) +
    labs(x = "group x", title = "각 관측치에서 평균까지의 거리") +
    theme_minimal()
```

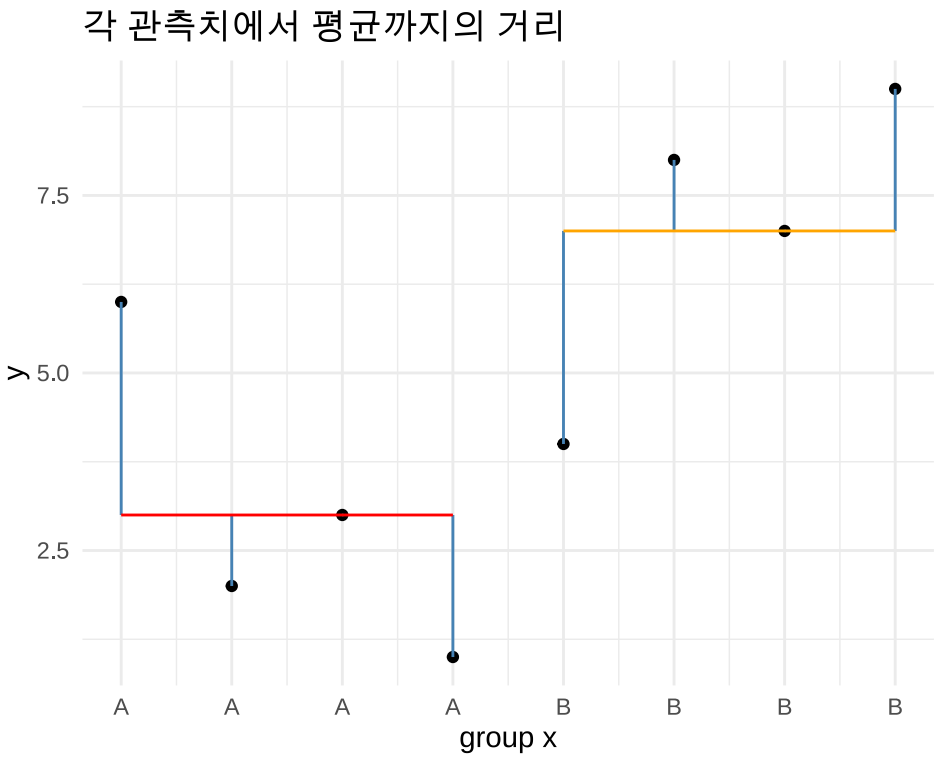


그림 31.2: 각 관측치에서 그룹별 평균까지의 거리(파란 선)

그룰별 평균까지의 거리를 제곱한 다음, 그 값들을 모두 더하면 잔차 제곱합(residual sum of squares)이 된다. 이 의미는 그룹별 평균을 고려한 후에도 남아있는 데이터의 변동을 의미한다.

부연하면, 상상해 보자. 그룹 A의 y값이 모두 3이고 그룹 B의 y값이 모두 7이라고 하자. 그러면 그룹별 평균을 고려한 후에는 남아있는 데이터의 변동은 없을 것이다(모두 빨간 평균 선에 놓일 것이다). 즉, 잔차 제곱합은 0이 된다.

```
ddf %>%
    group_by(x) %>%
    summarise(
        sum_y_squared = sum((y - mean_y)^2)
    )
```

```
# A tibble: 2 x 2
  x     sum_y_squared
  <fct>         <dbl>
```

```
1 A                 14
2 B                 14
```

31.3 그룹별 제곱합(sum of squares for each group)

전체 제곱합은 60이었고, 그룹을 고려한 잔차 제곱합은 28이었다. 그 차이를 그룹별 제곱합(sum of squares for each group)이라고 한다.

그래서 일반적으로 다음과 같은 관계를 가진다.

$$\text{Total Sum of Squares} = \text{Residual Sum of Squares} + \text{Sum of Squares for each group}$$

다음과 같이도 표현할 수 있다. 즉, 그룹별 제곱합은 예측 변수의 효과를 반영한다. 잔차 제곱합은 예측 변수의 효과를 반영하지 못한 부분을 의미한다.

$$\text{Total} = \text{Predictor(s)} + \text{Residual}$$

31.4 전체 데이터 변동에 기여하는 정도

다음과 같은 공식을 염두에 두면서 전체 데이터 변동성에 각 요소가 얼마 만큼 기여하는지 생각해 보자.

$$\text{Total} = \text{Predictor(s)} + \text{Residual}$$

그림 31.3은 생각하기 편하게 만든 간단한 그림이다.

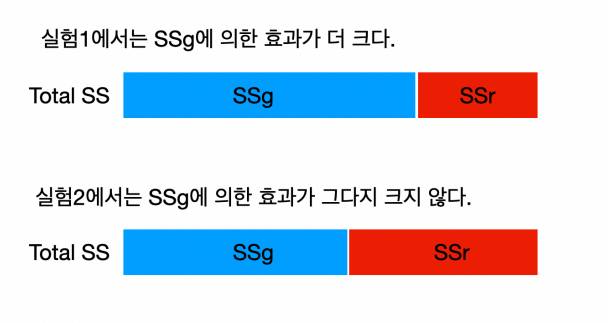


그림 31.3: 전체 변동에 기여하는 정도를 생각하자!

실험1에서는 잔차보다 그룹별 제곱합이 더 큰 기여를 하고 있다. 실험2에서는 잔차보다 그룹별 제곱합이 더 작은 기여를 하고 있다. 이 기여도를 비율로 표현하고 싶다.

그런데 우리가 어떤 것을 서로 비교하고 싶다면 그것들의 단위를 맞출 필요가 있다. 이 경우에는 각 제곱합에서 자유도(degrees of freedom)를 나누어 주어야 한다. 그 나눈 값을 평균 제곱합(mean sum of squares)이라고 한다.

$$\text{Mean Sum of Squares} = \frac{\text{Sum of Squares}}{\text{Degrees of Freedom}}$$

그래서 그룹별 평균 제곱합과 잔차 평균 제곱합의 비율을 가지고 데이터 변동성에 대한 기여도를 평가한다. 그것이 F-통계량(F-statistic)이다.

$$F = \frac{\text{Mean Sum of Squares for each group}}{\text{Mean Sum of Squares for residual}}$$

F-통계량은 그룹별 평균 제곱합과 잔차 평균 제곱합의 비율을 가지고 데이터 변동성에 대한 기여도를 평가한다. 그리고 F-통계량은 항상 두 개의 자유도를 가지고 계산하게 된다.

31.5 R 함수: `aov()`

`aov()` 함수를 사용하면 분산 분석을 할 수 있다. 다음과 같이 사용하면 된다.

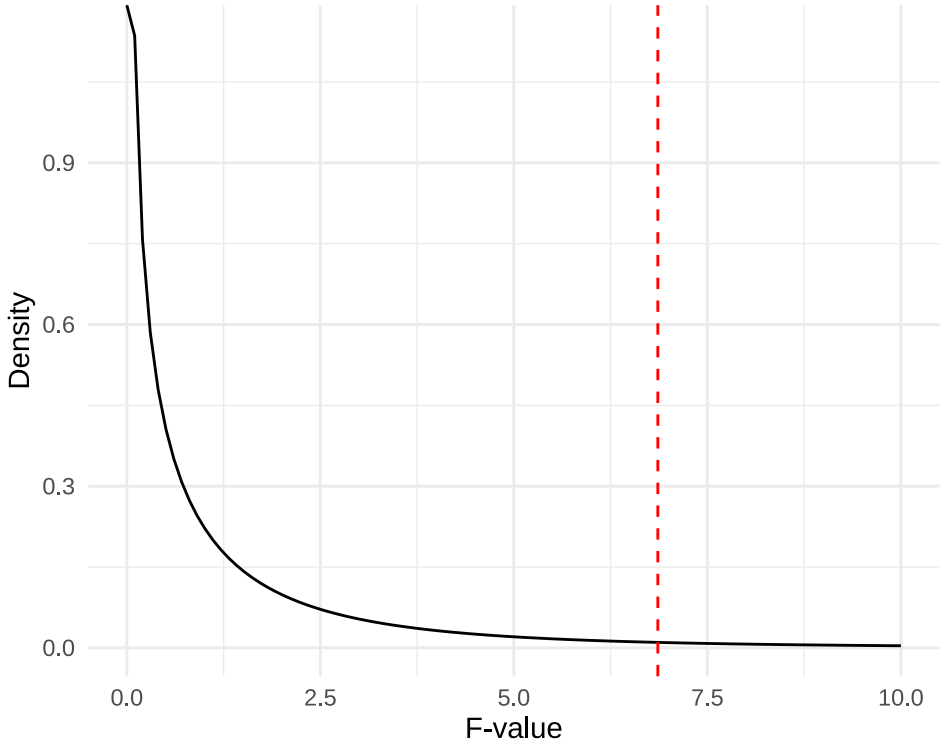
```
summary(aov(y ~ x, data = ddf))
```

```
            Df Sum Sq Mean Sq F value Pr(>F)
x            1     32   32.00   6.857 0.0397 *
Residuals    6     28    4.67
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

자유도 `df1 =1, df2 = 6`인 F-분포 곡선을 그려보자. 그리고 우리 데이터셋의 F-통계량인 `6.86`을 표시해 보자. 해당 경계를 시작으로 오른쪽 Area Under the Curve(AUC)를 계산하면 p-값이 된다.

```
x <- seq(0, 10, length.out = 100)
y <- df(x, df1 = 1, df2 = 6)
```

```r
ggplot(data.frame(x, y), aes(x, y)) +
    geom_line() +
    labs(x = "F-value", y = "Density") +
    geom_vline(xintercept = 6.86, color = "red", linetype = "dashed") +
    theme_minimal()
```



F-분포는 `pf()` 함수를 사용하여 p-값을 계산할 수 있다.

```r
pf(6.86, df1 = 1, df = 6, lower.tail = FALSE)
```

```
[1] 0.0396308
```

31.6 정리

이 장에서는 분산 분석에서 중요하게 다뤄지는 제곱합(sum of squares) 등의 개념을 설명했다. 이 개념들과 그 전개 논리를 이해하면 분산 분석에 대해 잘 이해할 수 있을 것이다. 다음 장에서는 분산 분석을 실제로 해보자.

- 전체 제곱합(total sum of squares)
- 그룹별 제곱합(sum of squares for each group)
- 잔차 제곱합(residual sum of squares)
- 평균 제곱합(mean sum of squares)
- F-통계량(F-statistic)
- 분산 분석(ANOVA)

32 분산 분석(ANOVA)

33 명목 변수 분석의 시작: 분할표(contingency table)

이 장에서는 명목 변수(counting data)의 분석에서 가장 기본이 되는 분할표(contingency table)에 대해 알아본다. 분할표는 명목 변수의 기술 통계(descriptive statistics)의 기본이다.

분할표는 명목 변수가 취할 수 있는 값들에 대한 빈도(frequency)와 비율(proportion)을 계산하여 만든다. 빈도는 명목 변수의 각 범주(category)에 대해 관측된 횟수를 나타내고, 비율은 명목 변수의 각 범주에 대해 관측된 횟수를 전체 관측치의 수로 나눈 값을 나타낸다.

33.1 R 코드로 분할표 만들기

먼저 사용할 뇌졸중 데이터셋을 불러온다.

```
library(tidyverse)
stroke_df <- readRDS("data/stroke_df.rds")
glimpse(stroke_df)
```

```
Rows: 5,110
Columns: 12
$ id                <chr> "9046", "51676", "31112", "60182", "1665", "56669", ~
$ gender            <fct> Male, Female, Male, Female, Female, Male, Male, Fema~
$ age               <dbl> 67, 61, 80, 49, 79, 81, 74, 69, 59, 78, 81, 61, 54, ~
$ hypertension      <fct> No, No, No, No, Yes, No, Yes, No, No, No, Yes, No, N~
$ heart_disease     <fct> Yes, No, Yes, No, No, No, Yes, No, No, No, No, Yes, ~
$ ever_married      <fct> Yes, Yes, Yes, Yes, Yes, Yes, Yes, No, Yes, Yes, Yes~
$ work_type         <fct> Private, Self-employed, Private, Private, Self-emplo~
$ residence_type    <fct> Urban, Rural, Rural, Urban, Rural, Urban, Rural, Urb~
```

```
$ avg_glucose_level <dbl> 228.69, 202.21, 105.92, 171.23, 174.12, 186.21, 70.0~
$ bmi               <dbl> 36.6, NA, 32.5, 34.4, 24.0, 29.0, 27.4, 22.8, NA, 24~
$ smoking_status    <fct> formerly smoked, never smoked, never smoked, smokes,~
$ stroke            <fct> Yes, Yes, Yes, Yes, Yes, Yes, Yes, Yes, Yes, Yes, Ye~
```

33.2 table()과 관련된 함수

명목 변수의 빈도와 비율을 계산하는 함수로 table()이 있다. table() 함수는 팩터 변수의 각 레벨(level)에 대해 관측된 횟수를 계산한다.

```
table(stroke_df$stroke)
```

```
  No  Yes
4861  249
```

table() 함수에 2개의 팩터 변수를 입력하면 두 변수의 모든 조합에 대한 빈도를 계산한다. table() 함수에 먼저 입력한 변수가 행으로 출력되고, 두 번째 변수가 열로 출력된다.

```
table(stroke_df$stroke, stroke_df$hypertension)
```

```
       No  Yes
  No  4429  432
  Yes  183   66
```

이런 분할표에서 행/열에 대한 집계를 함께 계산하고 싶을 수 있다. 이런 경우에는 addmargins() 함수를 사용한다. 이 함수 안에 위에서 만든 분할표를 입력하면 행/열에 대한 집계를 함께 계산한다.

```
addmargins(table(stroke_df$stroke, stroke_df$hypertension))
```

```
       No  Yes  Sum
```

```
  No   4429  432 4861
  Yes   183   66  249
  Sum  4612  498 5110
```

만약, 행들에 걸친 마진만 계산하고 싶다면 `margin = 1`을 사용하고(열 마진), 열들에 걸친 대한 마진만 계산하고 싶다면 `margin = 2`를 사용한다(행 마진).

```
addmargins(table(stroke_df$stroke, stroke_df$hypertension), margin = 1)
```

```
       No  Yes
  No   4429  432
  Yes   183   66
  Sum  4612  498
```

```
addmargins(table(stroke_df$stroke, stroke_df$hypertension), margin = 2)
```

```
       No  Yes  Sum
  No   4429  432 4861
  Yes   183   66  249
```

우리는 뇌졸중이 있는 환자군에서 고혈압 환자의 비율과 없는 환자군에서 고혈압 환자의 비율을 비교하고 싶다. 이런 경우에는 `prop.table()` 함수를 사용한다.

```
prop.table(table(stroke_df$stroke, stroke_df$hypertension), margin = 1)
```

```
            No       Yes
  No   0.9111294 0.0888706
  Yes  0.7349398 0.2650602
```

만약, 고혈압이 있는 환자군에서의 뇌졸중이 있는 환자의 비율을 계산하고 싶다면 `margin = 2`를 사용한다.

```r
prop.table(table(stroke_df$stroke, stroke_df$hypertension), margin = 2)
```

```
            No       Yes
  No  0.9603209 0.8674699
  Yes 0.0396791 0.1325301
```

R xtabs() 함수도 분할표를 만든다. 이 함수에서는 R 포뮬러(formula)를 사용하여 분할표를 만든다.

```r
xtabs(~ stroke + hypertension, stroke_df)
```

```
      hypertension
stroke   No  Yes
   No  4429  432
   Yes  183   66
```

3개 이상의 변수에 대한 분할표를 만들 때에는 xtabs() 함수를 사용할 수 있다.

```r
tb <- xtabs(~ stroke + hypertension + smoking_status, stroke_df)
tb
```

```
, , smoking_status = formerly smoked

      hypertension
stroke   No  Yes
   No   714  101
   Yes   51   19

, , smoking_status = never smoked

      hypertension
stroke   No  Yes
   No  1602  200
   Yes   58   32
```

```
, , smoking_status = smokes

      hypertension
stroke   No  Yes
   No   664   83
   Yes   31   11

, , smoking_status = Unknown

      hypertension
stroke   No  Yes
   No  1449   48
   Yes   43    4
```

이런 고차원의 분할표를 좀 더 읽기 쉽게 만들고 싶다면 `ftable()` 함수를 사용한다. 여기서 `f`는 `flatten`을 의미한다.

```
ftable(tb)
```

```
                    smoking_status formerly smoked never smoked smokes Unknown
stroke hypertension
No     No                                     714          1602    664    1449
       Yes                                    101           200     83      48
Yes    No                                      51            58     31      43
       Yes                                     19            32     11       4
```

고차원 분할표를 데이터프레임으로 변환하고 싶다면 `as.data.frame()` 함수를 사용한다.

```
df_tbl <- as.data.frame(tb)
df_tbl
```

```
   stroke hypertension  smoking_status Freq
1      No           No formerly smoked  714
```

```
2     Yes           No formerly smoked   51
3      No          Yes formerly smoked  101
4     Yes          Yes formerly smoked   19
5      No           No    never smoked 1602
6     Yes           No    never smoked   58
7      No          Yes    never smoked  200
8     Yes          Yes    never smoked   32
9      No           No          smokes  664
10    Yes           No          smokes   31
11     No          Yes          smokes   83
12    Yes          Yes          smokes   11
13     No           No         Unknown 1449
14    Yes           No         Unknown   43
15     No          Yes         Unknown   48
16    Yes          Yes         Unknown    4
```

위 경우를 보면 팩터들의 조합에 따른 빈도를 계산하고 그 값을 `Freq` 변수에 저장하고 있다.

만약 이와 같은 데이터프레임을 가지고 시작할 때, 이것을 분할표로 변환하고 싶으면 `xtabs()` 함수를 사용하는데, 팩터의 조합에 따른 빈도 값을 포뮬러의 좌변에 둔다.

```
xtabs(Freq ~ stroke + hypertension + smoking_status, df_tbl)
```

```
, , smoking_status = formerly smoked

      hypertension
stroke   No  Yes
   No   714  101
   Yes   51   19

, , smoking_status = never smoked

      hypertension
stroke   No  Yes
   No  1602  200
```

```
   Yes    58    32

, , smoking_status = smokes

      hypertension
stroke   No  Yes
   No   664   83
   Yes   31   11

, , smoking_status = Unknown

      hypertension
stroke   No  Yes
   No  1449   48
   Yes   43    4
```

33.3 분할표를 그래프로 그리기

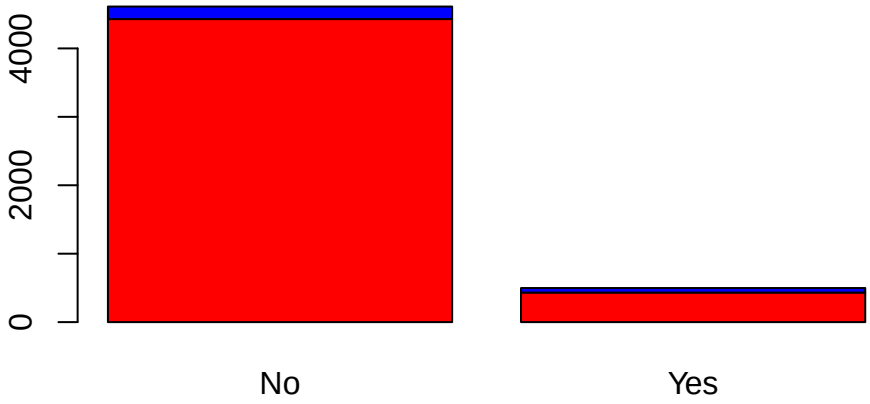
명목형 변수를 시각화하는 하는 대표적인 그래프는 막대 그래프(bar plot)이다. 이외에도 모자이크 그래프(mosaic plot)도 있고, 파이 차트(pie chart)도 있다.

```
tbl <- xtabs(~ stroke + hypertension, stroke_df)
tbl
```

```
      hypertension
stroke   No  Yes
   No  4429  432
   Yes  183   66
```

베이스 R의 `barplot()` 함수를 사용하여 분할표를 그래프로 그릴 수 있다. 이 함수에는 분할표를 넣어서 사용한다.

```
barplot(tbl, col = c("red", "blue"))
```

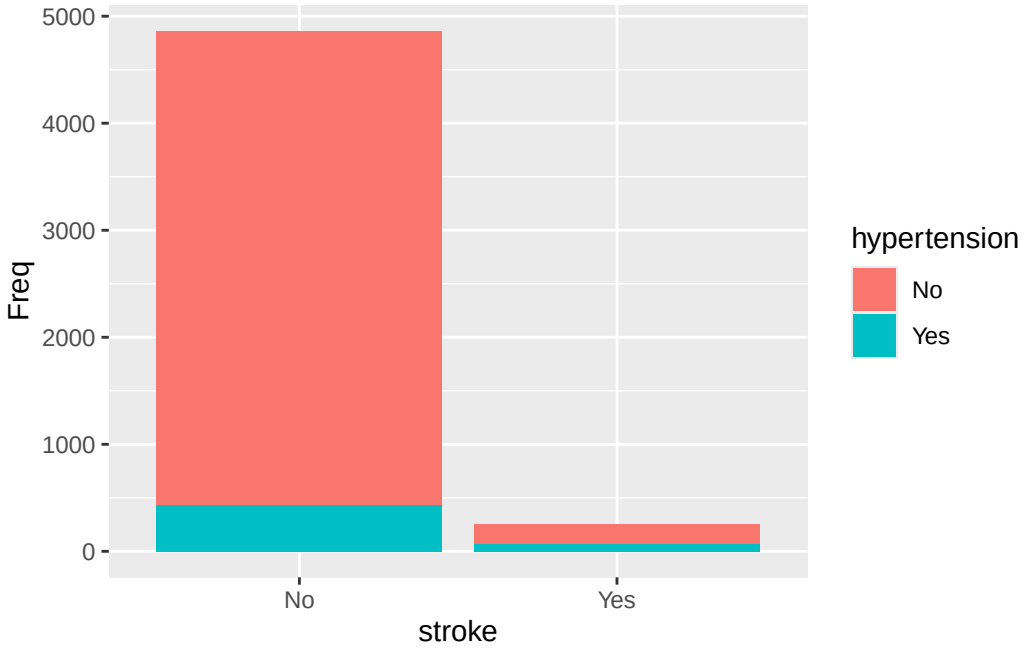


ggplot2 패키지는 데이터프레임을 사용하기 때문에 이것을 데이터프레임으로 변환하여 사용한다.

```
df_tbl2 <- as.data.frame(tbl)
df_tbl2
```

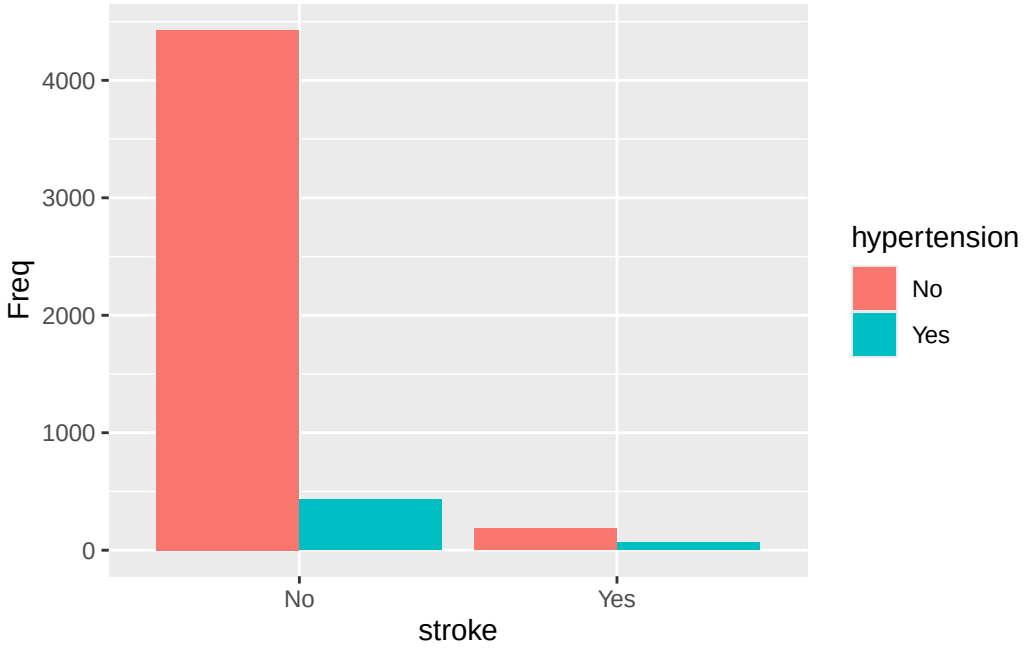
```
  stroke hypertension Freq
1     No           No 4429
2    Yes           No  183
3     No          Yes  432
4    Yes          Yes   66
```

```
ggplot(df_tbl2, aes(x = stroke, y = Freq, fill = hypertension)) +
    geom_col()
```



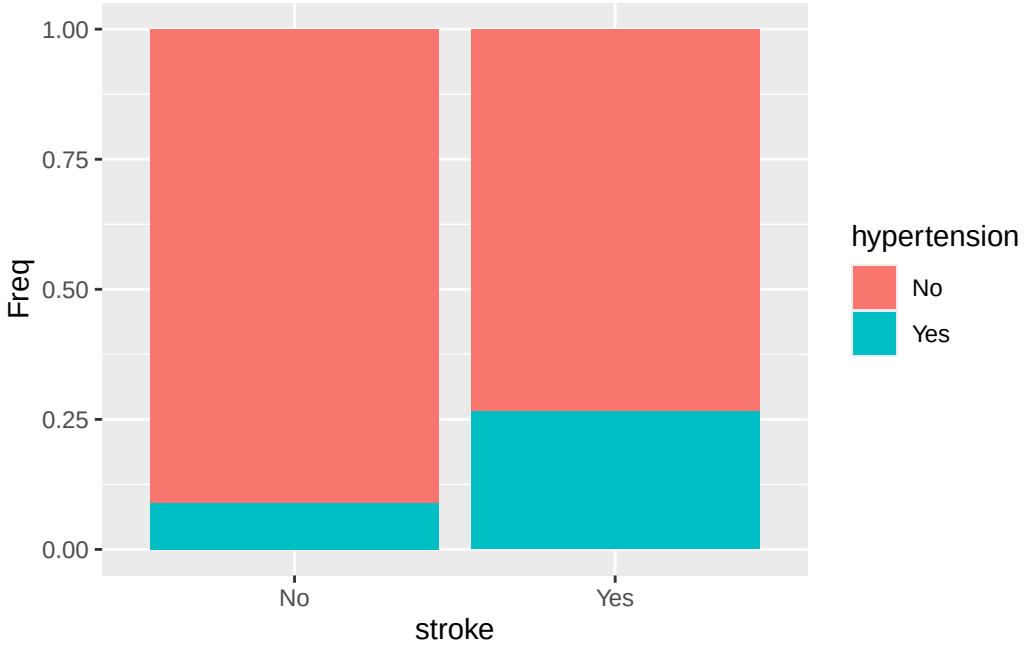
막대로 “stack” 형태가 아니라 분리하고 싶다면 `geom_col()` 함수에 `position = "dodge"` 옵션을 추가한다.

```
ggplot(df_tbl2, aes(x = stroke, y = Freq, fill = hypertension)) +
    geom_col(position = "dodge")
```



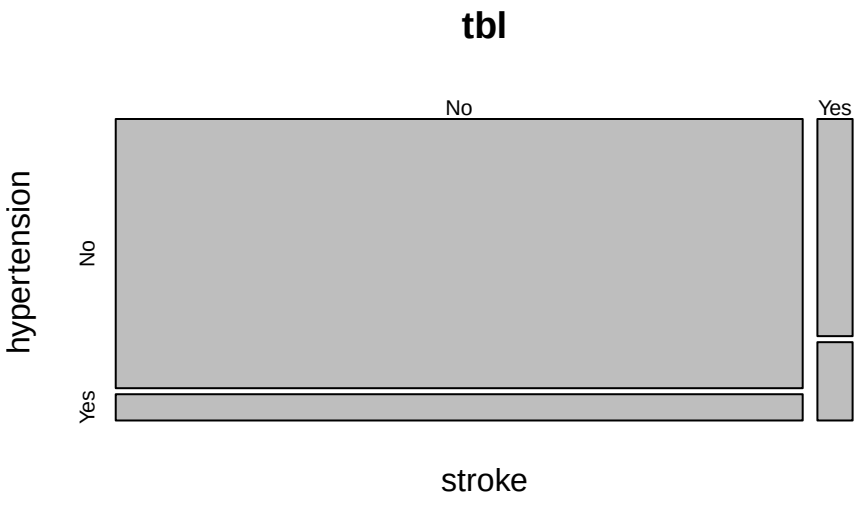
비율을 그래프로 그리고 싶다면 `geom_col()` 함수에 `position = "fill"` 옵션을 추가한다.

```
ggplot(df_tbl2, aes(x = stroke, y = Freq, fill = hypertension)) +
    geom_col(position = "fill")
```



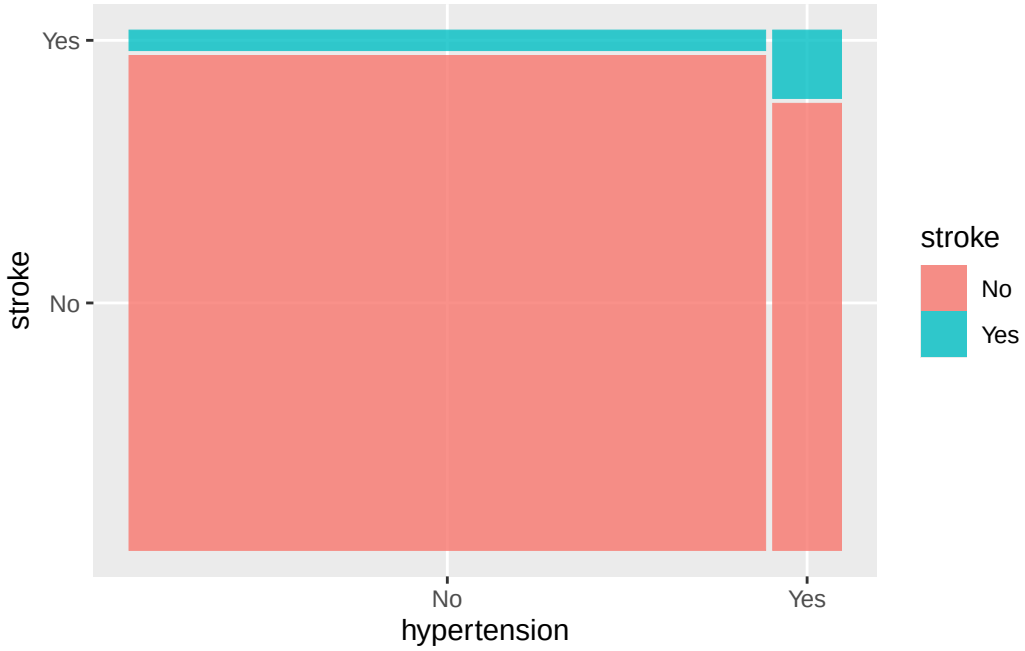
베이스 R의 `mosaicplot()` 함수를 사용하여 분할표를 그래프로 그릴 수 있다. 이 함수에는 분할표를 넣어서 사용한다.

```
mosaicplot(tbl)
```



ggmosaic 패키지를 사용하여 분할표를 그래프로 그릴 수 있다. 이 패키지는 데이터프레임을 사용하기 때문에 원래의 데이터프레임을 가지고 시작한다.

```
library(ggmosaic)
ggplot(stroke_df) +
    geom_mosaic(aes(x = product(stroke, hypertension), fill = stroke))
```



34 분할표 분석: 두 명목 변수의 독립성을 검정-카이제곱(Chi-Squared) 검정

이 장에서는 두 명목형 변수의 독립성을 검정하는 카이제곱(Chi-Squared) 검정에 대해 알아본다. 카이제곱 검정은 두 명목형 변수의 분포가 서로 독립적인지 검정하는 방법이다. 예를 들어, 뇌졸중(stroke)과 고혈압(hypertension)의 관계를 살펴보자.

```
library(tidyverse)
stroke_df <- readRDS("data/stroke_df.rds")
glimpse(stroke_df)
```

```
Rows: 5,110
Columns: 12
$ id                <chr> "9046", "51676", "31112", "60182", "1665", "56669", ~
$ gender            <fct> Male, Female, Male, Female, Female, Male, Male, Fema~
$ age               <dbl> 67, 61, 80, 49, 79, 81, 74, 69, 59, 78, 81, 61, 54, ~
$ hypertension      <fct> No, No, No, No, Yes, No, Yes, No, No, No, Yes, No, N~
$ heart_disease     <fct> Yes, No, Yes, No, No, No, Yes, No, No, No, No, Yes, ~
$ ever_married      <fct> Yes, Yes, Yes, Yes, Yes, Yes, Yes, No, Yes, Yes, Yes~
$ work_type         <fct> Private, Self-employed, Private, Private, Self-emplo~
$ residence_type    <fct> Urban, Rural, Rural, Urban, Rural, Urban, Rural, Urb~
$ avg_glucose_level <dbl> 228.69, 202.21, 105.92, 171.23, 174.12, 186.21, 70.0~
$ bmi               <dbl> 36.6, NA, 32.5, 34.4, 24.0, 29.0, 27.4, 22.8, NA, 24~
$ smoking_status    <fct> formerly smoked, never smoked, never smoked, smokes,~
$ stroke            <fct> Yes, Yes, Yes, Yes, Yes, Yes, Yes, Yes, Yes, Yes, Ye~
```

34.1 분할표 만들기

뇌졸중과 고혈압의 관계를 살펴보기 위해, 먼저 두 변수의 분할표를 만들어보자. 카이제곱 검정은 뒤에서 보겠지만 이런 분할표로 `chisq.test()` 함수를 사용하여 바로 검정할 수 있는데, 어떤 원리로 검정이 되는지 이해하기 위해서 분할표로 시작하자.

```
tbl <- xtabs(~ stroke + hypertension, stroke_df)
tbl
```

```
      hypertension
stroke   No  Yes
   No  4429  432
   Yes  183   66
```

`addmargins()` 함수를 사용하여 분할표에 마진을 추가할 수 있다. 마진은 각 행과 열의 합계를 나타낸다.

```
tbl_with_margins <- addmargins(tbl)
tbl_with_margins
```

```
      hypertension
stroke   No  Yes  Sum
   No  4429  432 4861
   Yes  183   66  249
   Sum 4612  498 5110
```

위 분할표에서 뇌졸중 확률은 다음과 같이 계산할 수 있다. 이런 확률은 마진을 보면서 계산한다.

$$
\begin{aligned}
P(\text{stroke+}) &= \frac{249}{5110} = 0.0487 \\
P(\text{stroke-}) &= \frac{4861}{5110} = 0.9513 \\
P(\text{hypertension+}) &= \frac{498}{5110} = 0.0975 \\
P(\text{hypertension-}) &= \frac{4612}{5110} = 0.9025
\end{aligned}
$$

34.2 기대값 계산하기

카이제곱 검정은 관측된 값과 기대값을 비교하여 두 변수의 독립성을 검정한다. 기대값은 각 칸에 대해 계산할 수 있다. 뇌졸중이 없으면서, 고혈압이 없는 사람들은 얼마나 기대할 수 있는가? 확률론에서 **두 사건이 독립이라면**(서로 영향을 주지 않으면, independent) 다음과 같은 공식이 성립한다.

$$P(A \cap B) = P(A) \times P(B)$$

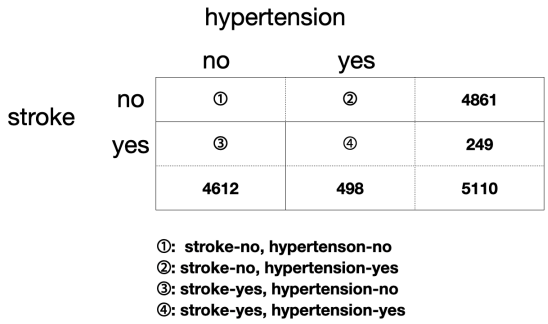


그림 34.1: 두 사건이 독립인 경우, 두 사건이 동시에 발생할 확률은 각 사건이 발생할 확률의 곱이다. 이것을 바탕으로 각 칸에 기대되는 값을 계산할 수 있다.

위 공식을 사용하여 각 칸에 대한 기대값을 계산할 수 있다.

- 뇌졸중이 없으면서, 고혈압이 없는 사람들은 다음과 같이 계산할 수 있다.

$$\begin{aligned} P(\text{stroke-, hypertension-}) &= P(\text{stroke-}) \times P(\text{hypertension-}) \\ &= 0.9513 \times 0.9025 = 0.8586 \end{aligned}$$

  이 확률값에 대해서, 전체 샘플이 5110개 이므로, 기대되는 값은 다음과 같다.

$$\text{기대값} = 0.8586 \times 5110 = 4387.446$$

- 같은 방법으로 뇌졸중이 없고 고혈압이 있는 경우에는 다음과 같이 계산된다.

$$\begin{aligned} P(\text{stroke-, hypertension+}) &= P(\text{stroke-}) \times P(\text{hypertension+}) \\ &= 0.9513 \times 0.0975 \\ &= 0.09275 \end{aligned}$$

이 확률값에 대해서, 전체 샘플이 5110개 이므로, 기대되는 값은 다음과 같다.

$$\text{기대값} = 0.09275 \times 5110 = 473.9525$$

이같은 방법으로 모든 칸에 대해 기대값을 계산하면 다음과 같은 결과를 얻는다.

```
chisq.test(tbl)$expected
```

```
      hypertension
stroke         No        Yes
   No   4387.2665 473.73346
   Yes   224.7335  24.26654
```

34.3 카이제곱 통계량 계산하기

카이제곱 검정은 관측값과 기대값의 차이를 이용하여 두 변수의 독립성을 검정한다. 두 명목형 변수에 대한 카이제곱 통계량(chisquared statistic)은 다음과 같이 계산된다.

$$\chi^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i}$$

여기서 O_i는 관측값, E_i는 기대값이다.

이 경우는 카이제곱 통계량은 다음과 같이 계산된다.

$$(4429 - 4387.27)^2/4387.27 + (432 - 473.73)^2/473.73 + \\ (183 - 224.73)^2/224.73 + (498 - 24.26)^2/24.26$$

이 값은 다음과 같다.

```
chisq.test(tbl)$statistic
```

```
X-squared
 81.60537
```

이 값은 관측된 개수와 기대되는 개수의 차이를 제곱하여 기대되는 개수로 나눈 값을 모두 더한 값이다.

34.4 카이제곱 검정 수행하기

카이제곱 검정은 `chisq.test()` 함수를 사용하여 수행할 수 있다. 이 함수는 관측값과 기대값을 비교하여 두 변수의 독립성을 검정한다. 이 카이제곱 통계량은 자유도가 `(r-1)(c-1)`인 카이제곱 분포를 따른다. 여기서 `r`은 행의 개수, `c`는 열의 개수이다.

```
chisq.test(tbl)
```

```
    Pearson's Chi-squared test with Yates' continuity correction

data:  tbl
X-squared = 81.605, df = 1, p-value < 2.2e-16
```

`chisq.test()` 함수는 카이제곱 통계량, 자유도, p-값을 반환한다. p-값은 두 변수의 독립성을 검정하는 데 사용된다. 일반적으로 p-값이 0.05보다 작으면 두 변수는 독립적이지 않다고 판단한다.

카이제곱 검정의 예시로, 뇌졸중과 고혈압의 관계를 살펴보았다. 이 예시에서는 두 변수의 독립성을 검정하기 위해 카이제곱 검정을 수행하였다. 결과적으로 p-값이 0.05보다 작아서, 뇌졸중과 고혈압은 독립적이지 않다고 판단할 수 있다.

34.5 카이제곱 검정의 가정

카이제곱 검정을 수행하기 전에 몇 가지 가정을 확인해야 한다.

1. **독립성**: 각 관측값은 서로 독립적이어야 한다. 즉, 한 관측값이 다른 관측값에 영향을 미치지 않아야 한다.
2. **표본 크기**: 각 셀의 기대값이 5 이상이어야 한다. 만약 기대값이 5 미만인 셀이 있다면, 카이제곱 검정의 결과가 신뢰할 수 없을 수 있다. 이 경우, Fisher의 정확 검정(Fisher's Exact Test)을 고려할 수 있다.
3. **명목형 변수**: 카이제곱 검정은 명목형 변수에 대해서만 적용할 수 있다. 순서형 변수에 대해서는 다른 검정 방법을 사용해야 한다.

35 infer 패키지로 가설 검정하기

`tidymodels` 패키지에 포함된 `infer` 패키지는 가설 검정(hypothesis testing)을 위한 표준적이고 일반적인 프레임워크를 제공한다. 개별 통계 함수나 특정 패키지를 사용하는 것보다는 좀 더 일반적이고 일관된 접근법을 취할 수 있고, 무엇보다 가설 검정의 의미를 이해하면서 가설 검정을 할 수 있다는 장점을 가지고 있다.

35.1 가설 검정 방법론

가설 검정의 일반적인 방법은 다음과 같다.

1. 관찰된 데이터를 바탕으로 통계량을 계산한다.
2. 귀무 가설을 설정한다.
3. 귀무 가설을 바탕으로 귀무 분포를 생성한다.
4. 관찰된 통계량이 귀무 분포에서 나올 확률(p-value)을 계산한다.
5. p-value가 임계값(critical value)보다 작으면 귀무 가설을 기각한다.

- 여기서 **통계량(statistic)**은 데이터의 특정 특성을 수치로 나타낸 것을 의미한다. 예를 들어, 평균, 비(proportion), 중앙값, 표준편차, 상관계수 등이 통계량에 해당한다.
- **귀무 가설(null hypothesis)**은 "아무런 차이가 없다"는 것을 의미한다. 예를 들어, 두 그룹의 평균이 같다고 가정하는 것을 말한다.
- **귀무 분포(null distribution)**는 귀무 가설을 바탕으로 생성된 확률 분포(probability distribution)를 의미한다.
    - 어떤 통계량이 어떤 확률 분포를 따르는지 알면, 그 통계량이 관찰될 확률을 계산할 수 있다.

35.2 관찰된 통계량이 귀무 분포에서 나올 확률이란?

분포와 관련된 R 함수

R에는 알려진 분포들에 대한 여러 함수들을 제공하고, 다음과 같은 접두사를 사용한다. 베이스 R에서 제공되는 분포는 `?distributions`를 참고하자. 정규분포(`norm`), t 분포(`t`), F 분포(`f`) 등 다양하게 준비되어 있다.

- `d`: 확률 밀도(density) 함수, 확률 밀도 함수는 확률 변수가 특정 값을 가질 확률을 나타내는 함수이다.
- `p`: 누적 분포(cumulative distribution) 함수, 누적 분포 함수는 확률 변수가 특정 값 이하일 확률을 나타내는 함수이다.
- `q`: 분위수(quantile) 함수, 분위수 함수는 확률 변수가 특정 값 이하일 확률이 특정 값일 때, 그 값을 찾는 함수이다.
- `r`: 랜덤 샘플(random sample) 생성 함수, 랜덤 샘플 생성 함수는 확률 변수가 특정 분포를 따를 때, 그 분포에서 랜덤하게 샘플을 생성하는 함수이다.

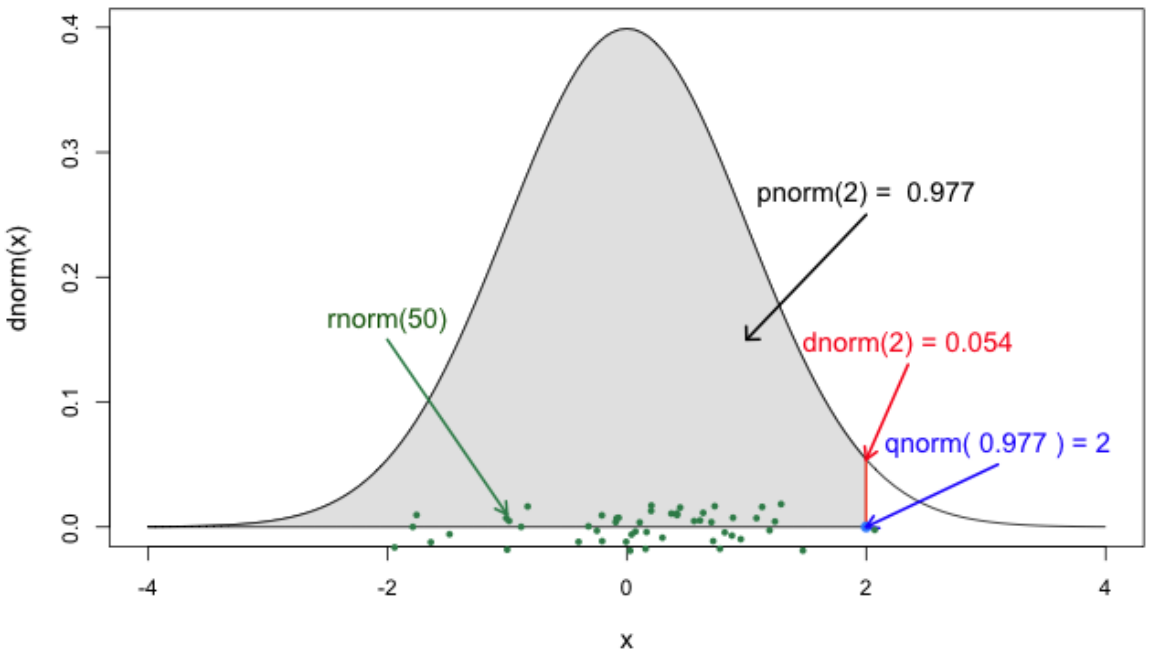


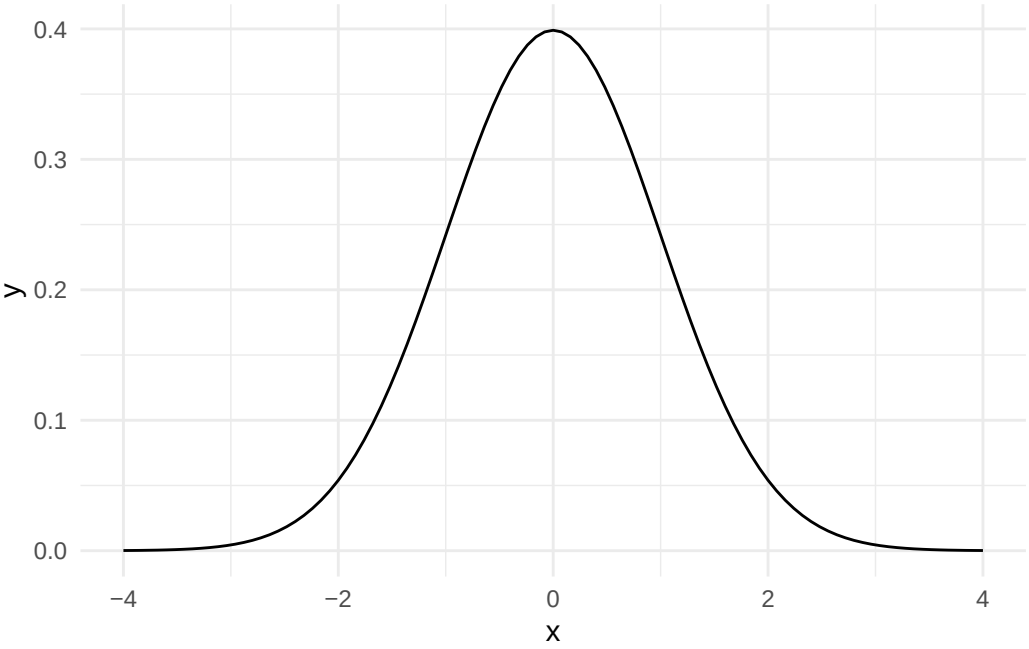
그림 35.1: 정규분포와 관련된 R 함수

확률 변수 `x`가 표준정규분포를 따른다고 해 보자. 이 `x`는 우리가 구하려고 하는 어떤 통계량이라고

생각하자.

평균이 0이고 표준편차가 1인 정규분포를 ggplot2 패키지를 사용해 시각화해 보자.

```
library(ggplot2)
library(gghighlight)
library(tidyverse)

ggplot(data.frame(x = c(-4, 4)), aes(x)) +
    stat_function(fun = dnorm, args = list(mean = 0, sd = 1)) +
    theme_minimal()
```



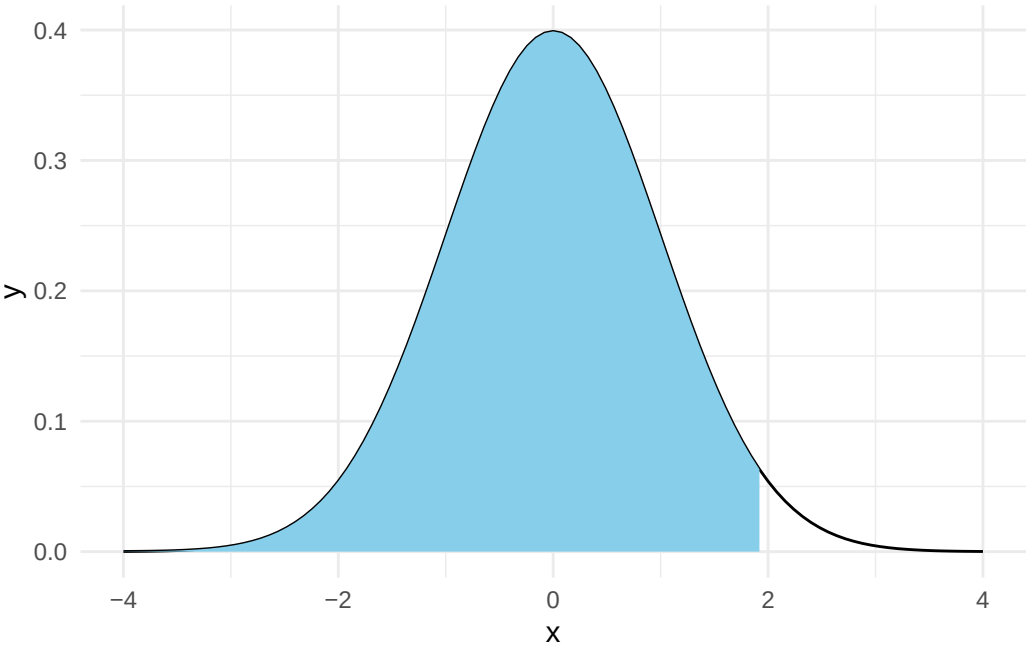
x <= 2일 때의 확률은 다음과 같이 계산할 수 있다.

```
pnorm(2, mean = 0, sd = 1)
```

```
[1] 0.9772499
```

이것을 시각화해 보자.

```
ggplot(data = tibble(x = -4:4), mapping = aes(x = x)) +
    stat_function(fun = dnorm) +
    stat_function(
        fun = function(x) ifelse(x < 2, dnorm(x), NA),
        geom = "area",
        fill = "sky blue"
    ) +
    theme_minimal()
```



x >= 2일 때의 확률은 다음과 같이 계산할 수 있다. 이것이 p-value와 관련되어 있다.

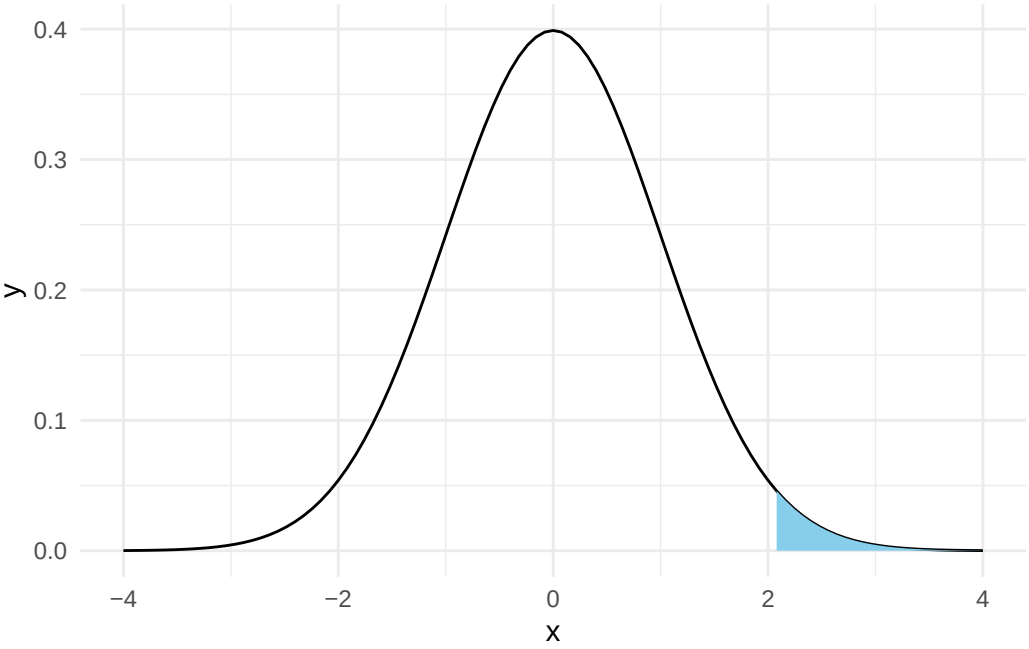
```
1 - pnorm(2, mean = 0, sd = 1)
```

```
[1] 0.02275013
```

이것을 시각화해 보자.

```
ggplot(data = tibble(x = -4:4), mapping = aes(x = x)) +
    stat_function(fun = dnorm) +
    stat_function(
```

```
        fun = function(x) ifelse(x > 2, dnorm(x), NA),
        geom = "area",
        fill = "sky blue"
    ) +
    theme_minimal()
```



`infer` 패키지에서는

- 귀무 가설을 사용하여 귀무 분포를 생성하고,
- `X = 2`와 같이 관찰된 통계량이 귀무 분포에서 나올 확률을 계산한다.
- 이와 관련된 확률이 `p-value`이다.

35.3 `infer` 패키지의 기본 개념

`infer` 패키지는 모델 추정 및 모델 검정을 위한 패키지로, 4개의 핵심 동사(함수)를 기반으로 가설 검정에 대한 일반적인 프레임워크를 제공한다. 이 4개 핵심 동사를 통해 모델을 설정하고, 귀무 가설을 설정하고, 귀무 분포를 생성하고, 통계량을 계산한다.

- `specify()` : 변수들의 관계를 통해 모델을 설정
- `hypothesize()` : 귀무 가설을 설정

- `generate()` : 귀무 가설을 바탕으로 (샘플링 또는 이론에 근거하여) 귀무 분포를 생성
- `calculate()` : 관찰된 현재의 데이터를 바탕으로 통계량을 계산함.

이들 동사와 더불어 다음과 같은 보조 함수를 사용한다.

- `visualize()` : 분포 시각화
    - `shade_p_value()` : p-value 영역 시각화
    - `shade_confidence_interval()` : 신뢰 구간 시각화
- `get_p_value()` : p-value 계산
- `get_confidence_interval()` : 신뢰 구간 계산

이것은 ModernDive 책에서 언급하는 바와 같이 Aleln Downey의 가설 검정 프레임워크와 맥을 같이 한다.

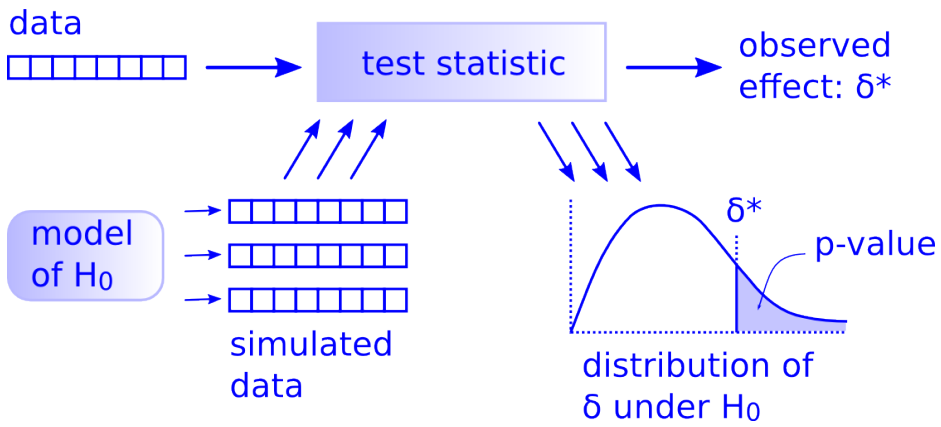


그림 35.2: Allen Downey의 가설 검정 프레임워크

실제로 `infer` 패키지를 사용하려면 이런 프레임워크를 이해한 다음, 여러 상황에 따라 어떤 옵션이나 조건을 사용할지 결정하게 된다.

35.4 `gss` 데이터셋

`infer` 패키지에는 미국에서 시행된 설문 데이터인 `gss` 데이터셋이 포함되어 있다. 이 데이터셋을 사용하여 `infer` 패키지 사용법을 익히자. 변수의 의미는 `?gss`를 참고하자.

```
library(infer)
data(gss)
glimpse(gss)
```

```
Rows: 500
Columns: 11
$ year    <dbl> 2014, 1994, 1998, 1996, 1994, 1996, 1990, 2016, 2000, 1998, 20~
$ age     <dbl> 36, 34, 24, 42, 31, 32, 48, 36, 30, 33, 21, 30, 38, 49, 25, 56~
$ sex     <fct> male, female, male, male, male, female, female, female, female~
$ college <fct> degree, no degree, degree, no degree, degree, no degree, no de~
$ partyid <fct> ind, rep, ind, ind, rep, rep, dem, ind, rep, dem, dem, ind, de~
$ hompop  <dbl> 3, 4, 1, 4, 2, 4, 2, 1, 5, 2, 4, 3, 4, 4, 2, 2, 3, 2, 1, 2, 5,~
$ hours   <dbl> 50, 31, 40, 40, 40, 53, 32, 20, 40, 40, 23, 52, 38, 72, 48, 40~
$ income  <ord> $25000 or more, $20000 - 24999, $25000 or more, $25000 or more~
$ class   <fct> middle class, working class, working class, working class, mid~
$ finrela <fct> below average, below average, below average, above average, ab~
$ weight  <dbl> 0.8960034, 1.0825000, 0.5501000, 1.0864000, 1.0825000, 1.08640~
```

35.5 1-Sample t-test

- 통계량: 평균 (지난 주 근무한 시간, hours)
- 귀무 가설: 평균이 40이다.
- 대립 가설: 평균이 40이 아니다.

```
obs_mean_hours <- gss %>%
    specify(response = hours) %>%
    calculate(stat = "mean")
obs_mean_hours
```

```
Response: hours (numeric)
# A tibble: 1 x 1
   stat
  <dbl>
```

```
1  41.4
```

귀무 가설을 바탕으로 귀무 분포를 생성한다. bootstrap 방법을 사용하여 귀무 분포를 생성한다.

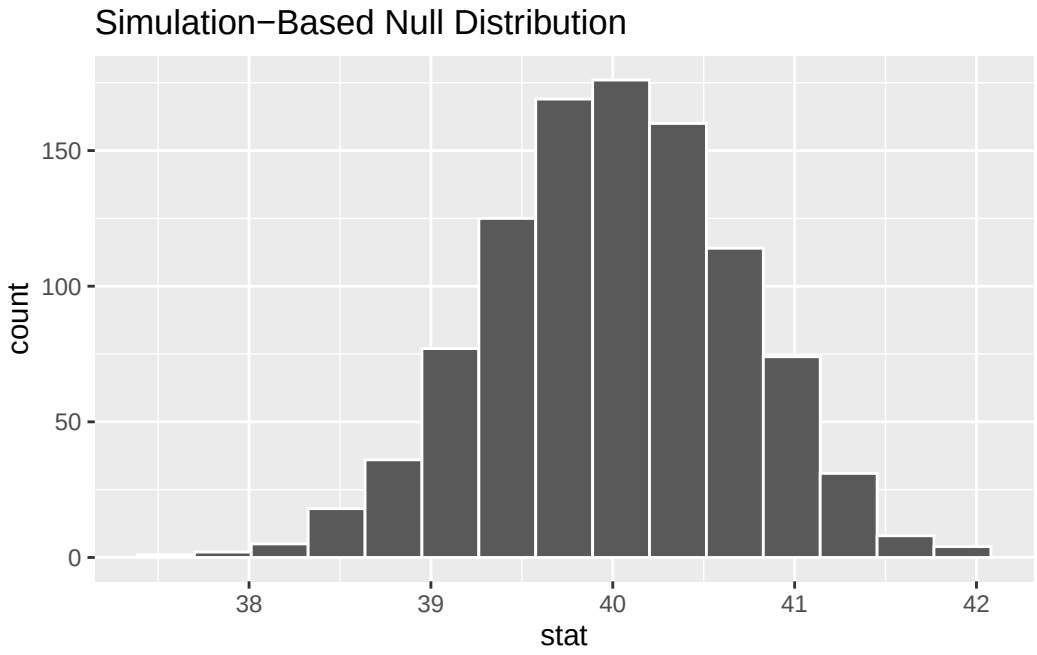
```
null_dist_hours <- gss %>%
    specify(response = hours) %>%
    hypothesize(null = "point", mu = 40) %>%
    generate(reps = 1000, type = "bootstrap") %>%
    calculate(stat = "mean")
```

위 코드의 의미는 다음과 같다.

- `specify(response = hours)`: 지난 주 근무한 시간을 응답 변수로 지정한다.
- `hypothesize(null = "point", mu = 40)`: 귀무 가설을 설정한다. 귀무 가설은 평균이 40이다. 구간이 아닌 포인트 가설이다.
- `generate(reps = 1000, type = "bootstrap")`: 귀무 가설을 바탕으로 귀무 분포를 생성한다. bootstrap 방법을 사용하여 귀무 분포를 생성한다. 1000번 반복하여 샘플을 확보한다.
- `calculate(stat = "mean")`: 샘플에 대한 평균을 구한다.

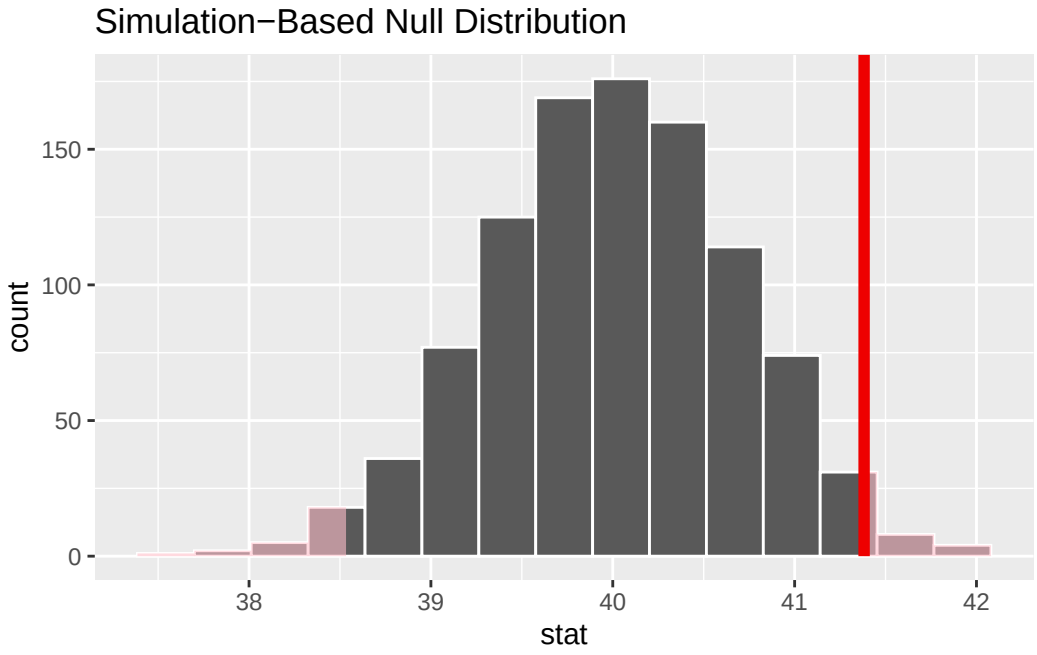
귀무 분포를 시각화해 보자. `visualize()` 함수는 귀무 분포를 시각화한다.

```
null_dist_hours %>%
    visualize()
```



이제 앞에서 관찰된 평균이 이 분포에 어떻게 위치하는지 확인해 보자.

```
null_dist_hours %>%
    visualize() +
    shade_p_value(obs_mean_hours, direction = "two-sided")
```



이제 이 분포에서 관찰된 평균이 나올 확률을 계산해 보자.

```
null_dist_hours %>%
    get_p_value(obs_mean_hours, direction = "two-sided")
```

```
# A tibble: 1 x 1
  p_value
    <dbl>
1   0.032
```

```
t_test(gss, response = hours, mu = 40)
```

```
# A tibble: 1 x 7
  statistic  t_df p_value alternative estimate lower_ci upper_ci
      <dbl> <dbl>   <dbl> <chr>          <dbl>    <dbl>    <dbl>
1      2.09   499  0.0376 two.sided       41.4     40.1     42.7
```

```
# calculate the observed statistic
observed_statistic <- gss %>%
    specify(response = hours) %>%
    hypothesize(null = "point", mu = 40) %>%
    calculate(stat = "t") %>%
    dplyr::pull()
observed_statistic
```

```
       t
2.085191
```

```
pt(unname(observed_statistic), df = nrow(gss) - 1, lower.tail = FALSE) * 2
```

```
[1] 0.03755959
```

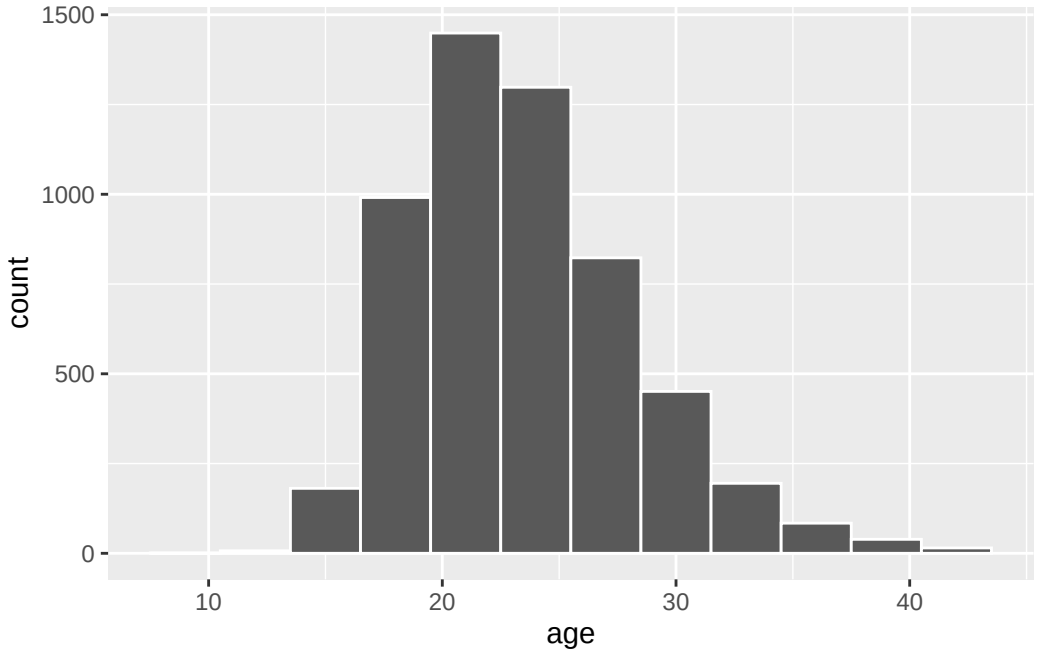
```r
library(dplyr)
library(knitr)
library(readr)

ageAtMar <- read_csv("https://ismayc.github.io/teaching/sample_problems/ageAtMar.csv")
```

```
Rows: 5534 Columns: 1
-- Column specification --------------------------------------------------------
Delimiter: ","
dbl (1): age

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
glimpse(ageAtMar)
```

```
Rows: 5,534
Columns: 1
$ age <dbl> 32, 25, 24, 26, 32, 29, 23, 23, 29, 27, 23, 21, 29, 40, 22, 20, 31~
```

```r
ageAtMar %>% ggplot(aes(x = age)) +
    geom_histogram(binwidth = 3, color = "white")
```



관찰된 나이의 평균은 다음과 같다.

```
obs_mean_age <- ageAtMar %>%
    specify(response = age) %>%
    calculate(stat = "mean")
obs_mean_age
```

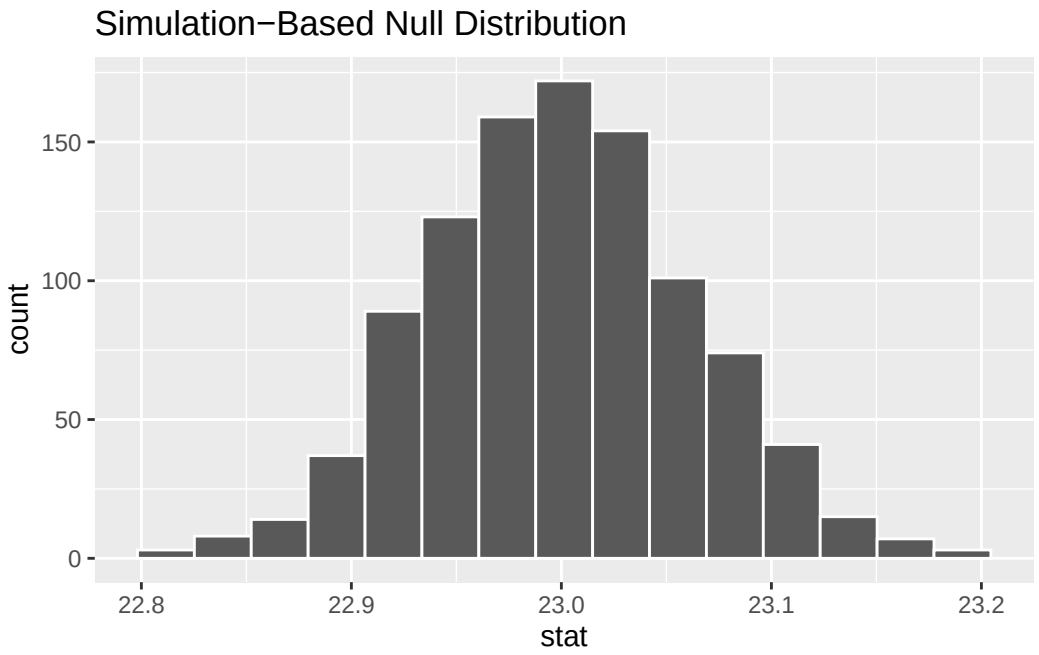
```
Response: age (numeric)
# A tibble: 1 x 1
   stat
  <dbl>
1  23.4
```

귀무 가설을 바탕으로 귀무 분포를 생성한다. `bootstrap` 방법을 사용하여 귀무 분포를 생성한다.

```
null_dist_age <- ageAtMar %>%
    specify(response = age) %>%
    hypothesize(null = "point", mu = 23) %>%
    generate(reps = 1000, type = "bootstrap") %>%
    calculate(stat = "mean")
```

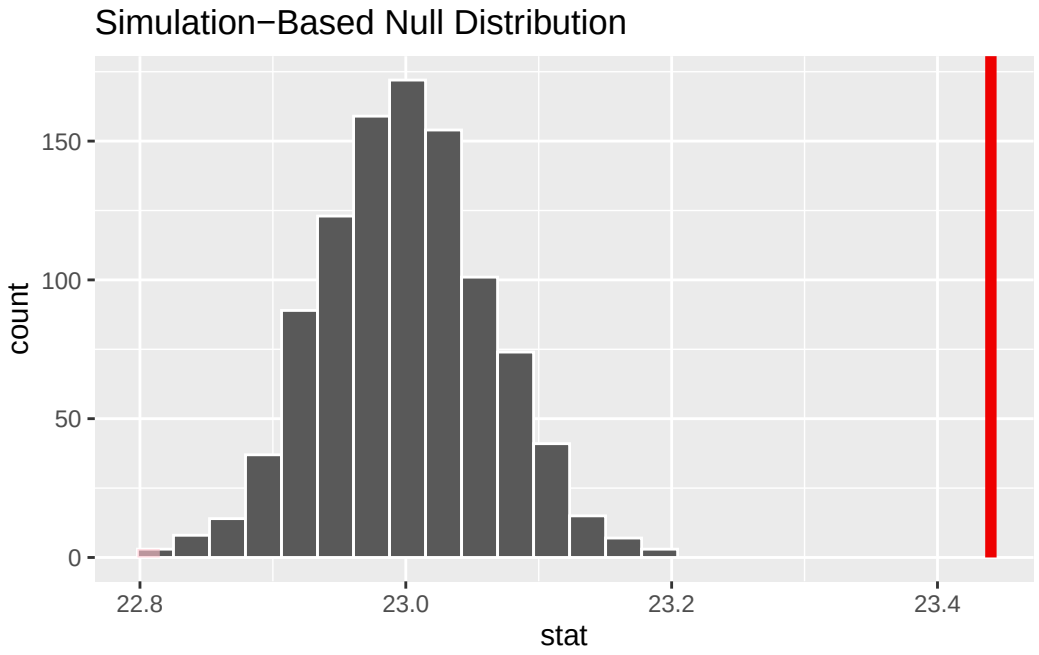
귀무 분포를 시각화해 보자.

```
null_dist_age %>%
    visualize()
```



이제 앞에서 관찰된 평균이 이 분포에 어떻게 위치하는지 확인해 보자.

```
null_dist_age %>%
    visualize() +
    shade_p_value(obs_mean_age, direction = "two-sided")
```



이제 이 분포에서 관찰된 평균이 나올 확률을 계산해 보자.

```
null_dist_age %>%
    get_p_value(obs_mean_age, direction = "two-sided")
```

```
Warning: Please be cautious in reporting a p-value of 0. This result is an approximation
based on the number of `reps` chosen in the `generate()` step.
i See `get_p_value()` (`?infer::get_p_value()`) for more information.

# A tibble: 1 x 1
  p_value
    <dbl>
1       0
```

신뢰 구간을 계산해 보자.

```
null_dist_age %>%
    get_confidence_interval(level = 0.95)
```

```
# A tibble: 1 x 2
```

```
  lower_ci upper_ci
     <dbl>    <dbl>
1     22.9     23.1
```

t_test()를 시행해 보자.

```
t_test(ageAtMar, response = age, mu = 23)
```

```
# A tibble: 1 x 7
  statistic  t_df  p_value alternative estimate lower_ci upper_ci
      <dbl> <dbl>    <dbl> <chr>          <dbl>    <dbl>    <dbl>
1      6.94  5533 4.50e-12 two.sided       23.4     23.3     23.6
```

```
# calculate the observed statistic
observed_statistic <- ageAtMar %>%
    specify(response = age) %>%
    hypothesize(null = "point", mu = 23) %>%
    calculate(stat = "t") %>%
    dplyr::pull()
observed_statistic
```

```
       t
6.935697
```

```
pt(unname(observed_statistic), df = nrow(ageAtMar) - 1, lower.tail = FALSE) * 2
```

```
[1] 4.504324e-12
```

참고문헌

Lazic, Stanley E. 2016. Experimental Design for Laboratory Biologists. Cambridge University Press.

Wickham, Hadley. 2014. “Tidy Data.” The Journal of Statistical Software 59. http://www.jstatsoft.org/v59/i10/.