

2021 연구데이터 분석과정

생명정보학을 위한 R프로그래밍(중급과정)

한국생명공학연구원 김하성

2021-06-17

Contents

1 강의 개요	5
1.1 참고 교재	5
1.2 참고 자료	5
1.3 강의 계획	5
2 R/Rstudio basics	7
2.1 What is R / Rstudio	7
2.2 R / Rstudio Installation	7
2.3 Rstudio interface	8
2.4 R programming	9
2.5 Terminology	10
2.6 Supports	10
2.7 R packages and Dataset	11
3 Tidyverse for Data science	13
3.1 Tibble object type	13
3.2 Tidy data structure	14
3.3 Pivoting	15
3.4 Separating and uniting	16
3.5 dplyr and pipe operator	17
3.6 dplyr - Important functions	18
3.7 Airquality example	20
4 ggplot2 for data visualization	23
4.1 Basics	23
4.2 Bar graph	24
4.3 Line graph	25
4.4 Smoothing	26
4.5 Statistics and positions	27
4.6 Facets	28
4.7 Themes, Labels, and Scales	29
4.8 ggplot examples	32

Chapter 1

강의 개요

- 장소: 코빅 3층 전산교육장(1304호)
- 강사: 한국생명공학연구원 바이오합성연구센터 김하성
- 연락처: 042-860-4372, haseong@kribb.re.kr (생명연 연구동 1143)
- 강의자료: <https://greendaygh.github.io/KRIBBR2021/>

1.1 참고 교제

- Using R for Introductory Statistics by John Verzani
 - Free version of 1st Edition
 - Second edition
- R for Data Science
- Bioinformatics Data Skills by Vince Buffalo
- Introductory Statistics with R by Dalgaard
- 일반통계학 (영지문화사, 김우철 외)

1.2 참고 자료

- R 홈페이지
- Rstudio 홈페이지
- Bioconductor
- R 기본 문서들
- R ebooks
- Cheat Sheets
- RStudio Webinars
- Shiny
- Hadley github

1.3 강의 계획

- Day1 - R, Rstudio Basics, Tindyverse
- Day2 - Bioconductor, Sequence data analysis
- Day3 - High-throughput sequence data
- Day4 - Annotation

Chapter 2

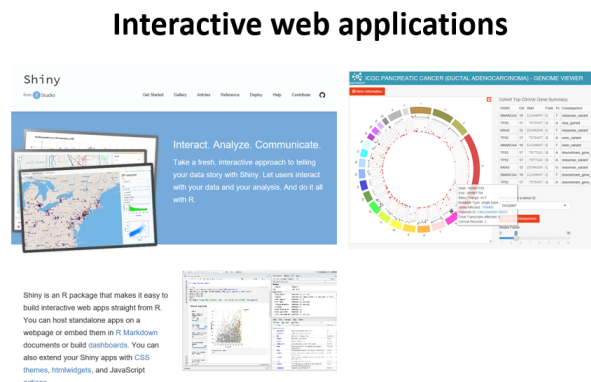
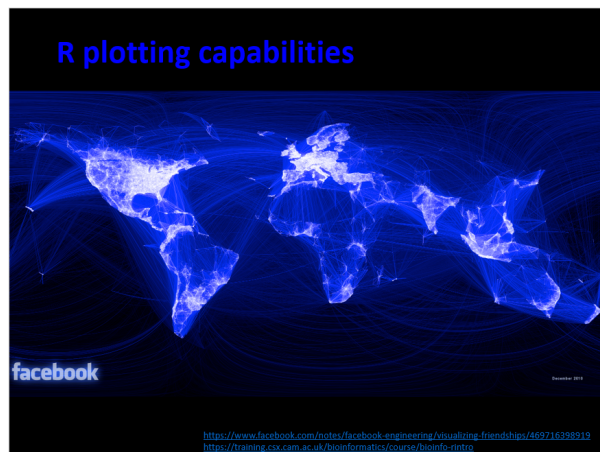
R/Rstudio basics

2.1 What is R / Rstudio



R은 통계나 생물통계, 유전학을 연구하는 사람들 사이에서 널리 사용되는 오픈소스 프로그래밍 언어입니다. Bell Lab에서 개발한 S 언어에서 유래했으며 많은 패키지가 (다른 사람들이 만들어 놓은 코드) 있어서 쉽게 가져다 사용할 수 있습니다. R은 복잡한 수식이나 통계 알고리즘을 간단히 구현하고 사용할 수 있으며 C, C++, Python 등 다른 언어들과의 병행 사용도 가능합니다.

R은 통계분석에 널리 사용되는데 이는 데이터 가시화를 위한 그래픽 기능이나 벡터 연산 등의 편리함 때문에 점점 더 많은 사람들이 사용하고 있습니다. 기존에는 느린 속도나 부족한 확장성이 다른 언어들에 비해 단점으로 지적되었으나 R 언어의 지속적인 개발과 업데이트로 이러한 단점들이 빠르게 극복되고 있습니다. R 사용을 위해서는 R 언어의 코어 프로그램을 먼저 설치하고 그 다음 R 언어용 IDE인 RStudio 설치가 필요합니다.



2.2 R / Rstudio Installation

2.2.1 R 설치

- R 사이트에 접속 후 (<https://www.r-project.org/>) 좌측 메뉴 상단에 위치한 CRAN 클릭.
- 미러 사이트 목록에서 Korea의 아무 사이트나 들어감
- Download R for Windows를 클릭 후 base 링크 들어가서
- Download R x.x.x for Windows 링크 클릭으로 실행 프로그램 다운로드 - 로컬 컴퓨터에 Download 된 R-x.x.x-win.exe 를 실행
- 설치 가이드에 따라 R 언어 소프트웨어 설치 완료

2.2.2 Rstudio 설치

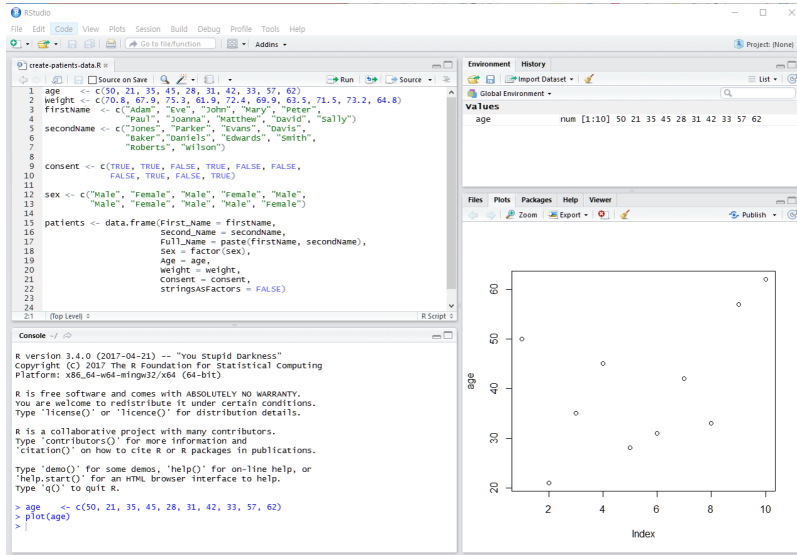
Rstudio는 R 언어를 위한 오픈소스 기반 통합개발환경(IDE)으로 R 프로그래밍을 위한 편리한 기능들을 제공해 줍니다.

- 사이트에 접속 (<https://www.rstudio.com/>), 상단의 Products > RStudio 클릭

- RStudio Desktop 선택
- Download RStudio Desktop 클릭
- RStudio Desktop Free 버전의 Download를 선택하고
- Download RStudio for Windows 클릭, 다운로드
- 로컬 컴퓨터에 다운로드된 RStudio-x.x.x.exe 실행
- 설치 가이드에 따라 설치 완료

2.3 Rstudio interface

- 좌측 상단의 공간은 코드편집창, 좌측 하단은 콘솔창이며 각 위치를 기호에 따라서 바꿀 수 있습니다.

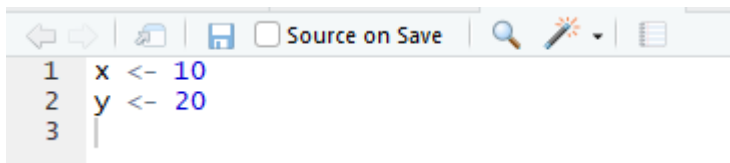


2.3.1 Keyboard shortcuts

- 참고사이트
 - <https://support.rstudio.com/hc/en-us/articles/200711853-Keyboards-Shortcuts>
 - Tools -> Keyboard shortcut Quick Reference (Alt + Shift + K)
- 코드편집창 이동 (Ctrl+1) 콘솔창 이동 (Ctrl+2)
- 한 줄 실행 (Ctrl+Enter)
- 주석처리 (Ctrl + Shift + C)
 - 또는 #으로 시작하는 라인

2.3.2 Exercise

1. 코드편집창에서 다음 입력하시오



- 단축키 Ctrl + enter로 코드 실행
- 단축키 Ctrl + 2로 커서 콘솔창으로 이동
- x값 x+y값 확인
- 단축키 Ctrl + 1로 코드편집창 이동
- 단축키 Ctrl + Shift + C 사용

```
# x <- 10
# y <- 20
```

2.3.3 Set a project

- 프로젝트를 만들어서 사용할 경우 파일이나 디렉토리, 내용 등을 쉽게 구분하여 사용 가능합니다. 아래와 같이 원하는 위치에 원하는 이름의 프로젝트를 생성하고 프로젝트를 시작할 때는 해당 디렉토리의 xxx.Rproj 파일을 클릭합니다.
- File > New Project > New Directory > New Project > “kribb-R” > Create Project
- File > New File > R Script > “day1.R”

2.4 R programming

2.4.1 Console calculator

```
2 + 2
((2 - 1)^2 + (1 - 3)^2)^(1/2)
2 + 2; 2 - 2
```

2.4.2 Exercise

다음 공식들을 계산하는 R 코드를 작성하시오

$$\sqrt{(4+3)(2+1)}$$

$$2^3 + 3^2$$

$$\frac{0.25 - 0.2}{\sqrt{0.2(1 - 0.2)/100}}$$

2.4.3 Variables and values

- 프로그래밍 언어의 공통적 개념 , , , ,
- Assignment operator (<- OR =)
 - Valid object name <- value
 - 단축키: Alt + - (the minus sign)

```
x <- 2
y <- x^2 - 2*x + 1
y
x <- "two"
some_data <- 9.8
```

- 내장 변수 Built-in variables

```
pi
```

- 변수이름 작명법
 - 문자, 숫자, “_”, “.” 등으로 구성
 - 대소문자 구분
 - 가독성, 의미있는 변수 이름
 - 길이 제한 없음

```
i_use_snake_case <- 1
otherPeopleUseCamelCase <- 2
some.people.use.periods <- 3
And_aFew.People.RENOUNCEconvention <- 4
```

- 자동 완성 기능 (Tab completion) in RStudio
- 이전 명령은 콘솔에서 위 아래 화살표

2.4.4 Exercise

- 변수 x에 1, 3, 5, 7, 9를, 변수 y에 2, 4, 6, 8, 10을 저장하는 코드를 작성하시오
- 앞서 변수 x와 y를 더한 값을 z에 저장하는 코드를 작성하시오
- 변수 x에 “hello world!” 를 저장하고 x의 값을 출력하는 코드를 작성하시오

2.4.5 Functions

함수(Function)란 사용자가 원하는 기능을 수행하는 코드의 모음으로서 반복적으로 쉽게 사용할 수 있도록 만들어 놓은 코드입니다. 특정 데이터를 입력으로 받아 원하는 기능을 수행한 후 결과 데이터를 반환하는 구조를 가집니다. 함수는 일반적으로 다음과 같은 포맷으로 구현할 수 있습니다.

```
my_function_name <- function(parameter1, parameter2, ... ){
  ##any statements
  return(object)
}
```

예를 들어 다음과 같은 my_sine 함수를 만들 수 있으며 parameter (매개변수)는 x이고 y는 반환값을 저장하는 지역변수입니다.

```
my_sine <- function(x){
  y <- sin(x)
  return(y)
}
```

- 내장 함수 (Built-in functions)

```
x <- pi
sin(x)
sqrt(x)
log(x)
log(x, 10)
x <- c(10, 20, 30)
x + x
mean(x)
sum(x)/length(x)
```

2.4.6 Exercise

1. 변수 x에 1, 3, 5, 7, 9를, 변수 y에 2, 4, 6, 8, 10을 저장하는 코드를 작성하시오
2. x와 y를 더한 값을 z에 저장하는 코드를 작성하시오
3. mysum 이라는 이름의 함수를 작성하되 두 변수를 입력으로 받아 더한 후 결과를 반환하는 코드를 작성하시오
4. mymean 이라는 이름의 함수를 작성하되 두 변수를 입력으로 받아 평균을 구한 후 결과를 반환하는 코드를 작성하시오

2.5 Terminology

- Session: R 언어 실행 환경
- Console: 명령어 입력하는 창
- Code: R 프로그래밍 변수/제어문 모음
- Object types:
 - vector: 값들의 모임 combine function c() EX: c(6, 11, 13, 31, 90, 92)
 - matrix: 2D 형태 값들의 모임
 - array: 1D, 2D, 3D, ... 형태 값들의 모임
 - factor: 범주형 데이터
 - data frame: 2D 형태 값들의 모임 (다른 타입 값 가능)
 - list:
 - function: 특정 기능 수행, [함수이름, 입력값 (arguments), 출력값 (return)] 으로 구성
- Data (value) types:
 - Integers
 - doubles/numerics
 - logicals
 - characters.
- Conditionals (조건, 제어):
 - if, ==, & (AND), | (OR) Ex: (2 + 1 == 3) & (2 + 1 == 4)
 - for, while: 반복 수

2.6 Supports

2.6.1 Help

R은 방대한 양의 도움말 데이터를 제공하며 다음과 같은 명령어로 찾아볼 수 있습니다.

```
help("mean")
?mean
example("mean")
help.search("mean")
```

```
??mean
help(package="MASS")
```

2.6.2 Cheatsheet

<https://rstudio.com/resources/cheatsheets/>

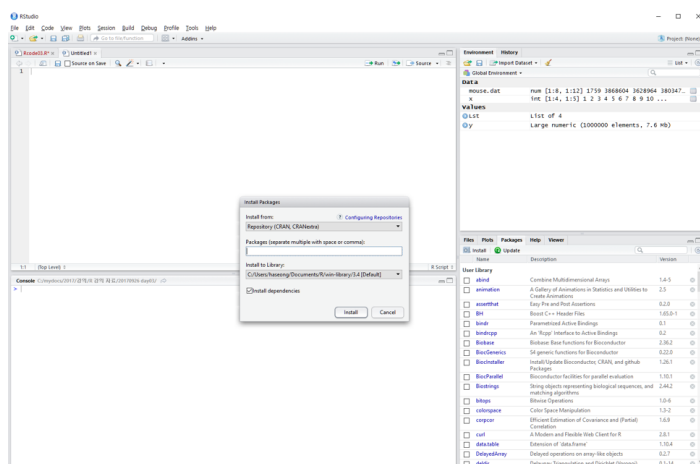
The image displays a comprehensive set of R programming cheat sheets. The sheets are organized into several categories:

- Base R Cheat Sheet:** Includes sections for Getting Help (e.g., `?mean`, `help(package="MASS")`), Using Packages (e.g., `install.packages("dplyr")`), and Working Directory (e.g., `getwd()`, `setwd()`).
- Vectors:** Covers Creating Vectors (e.g., `c(1, 2, 3)`), Vector Functions (e.g., `sort()`, `rev()`), and Selecting Vector Elements (e.g., `x[1]`, `x[1:4]`).
- Programming:** Includes For Loop, While Loop, If Statements, and Functions (e.g., `function() { ... }`).
- Reading and Writing Data:** Covers `read.table()`, `write.table()`, `read.csv()`, and `write.csv()`.
- Types:** Explains Logical, Numeric, Character, and Factor types.
- Matrices:** Covers creating and manipulating matrices (e.g., `matrix()`).
- Lists:** Explains creating and manipulating lists (e.g., `list()`).
- Data Frames:** Covers creating and manipulating data frames (e.g., `data.frame()`).
- Strings:** Includes functions for string manipulation (e.g., `paste()`, `grep()`).
- Factors:** Explains creating and manipulating factors (e.g., `factor()`).
- Statistics:** Includes functions for statistical analysis (e.g., `mean()`, `sd()`).
- Distributions:** Lists various probability distributions (e.g., `rnorm()`, `dnorm()`).
- Plotting:** Includes functions for creating plots (e.g., `plot()`, `hist()`).
- Dates:** Explains working with dates (e.g., `as.Date()`).

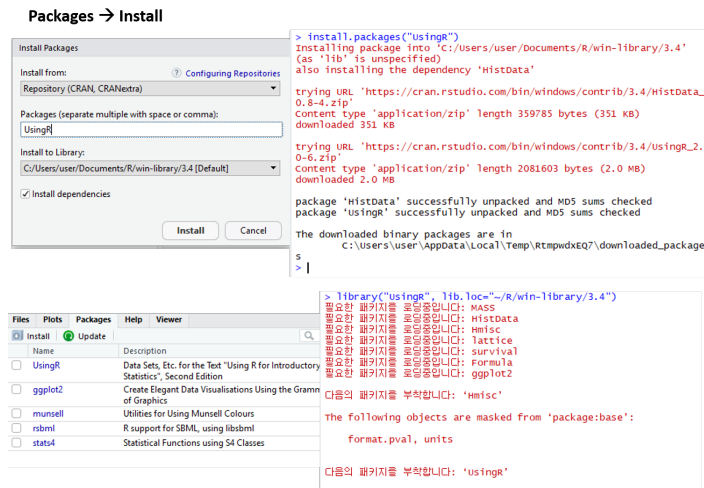
2.7 R packages and Dataset

2.7.1 R packages

- R 패키지는 함수들의 모음으로 다른 사람들이 만들어 놓은 함수를 가져와서 사용할 수 있음
- 예) `sum()` 은 base package에 있고 `sd()` 함수는 stats package에서 제공
- 패키지를 구할 수 있는 가장 대표적인 사이트
- The Comprehensive R Archive Network (CRAN) - <http://cran.r-project.org/web/views/>
- Bioconductor - <http://www.bioconductor.org/packages/release/bioc/>



- Using R package installation



```
install.packages("UsingR")
```

- UsingR package loading

```
library(UsingR)
help(package="UsingR")
```

2.7.2 Data sets

- 일반적으로 패키지 안에 관련된 데이터도 같이 저장
- data() function를 이용해서 패키지 데이터를 사용자 작업공간에 복사해서 사용 가능

```
head(rivers)
length(rivers)
class(rivers)
data(rivers)
data(package="UsingR")
library(HistData)
head(Cavendish)
str(Cavendish)
head(Cavendish$density2)
```

이 저작물은 크리에이티브 커먼즈 저작자표시-비영리-변경금지 4.0 국제 라이선스에 따라 이용할 수 있습니다.

Chapter 3

Tidyverse for Data science

tidyverse (<https://www.tidyverse.org/>)는 데이터 사이언스를 위한 R 기반의 독창적인 패키지들의 모음입니다. Rstudio의 핵심 전문가인 해들리위컴이 (Hadley Wickham) 중심이 되어 만들어 졌으며 기존의 툴보다 쉽고 효율적으로 데이터 분석을 수행할 수 있습니다.



R packages for data science

The tidyverse is an opinionated **collection of R packages** designed for data science. All packages share an underlying design philosophy, grammar, and data structures.

Install the complete tidyverse with:

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

데이터사이언스는 넓은 범위의 개념과 방법적인 정도가 있는 것은 아닙니다. 그러나 위 tidyverse의 목적은 데이터 분석을 위한 핵심이되는 고효율의 툴을 제공하는 것이며 그 철학은 다음과 같은 그림으로 요약할 수 있습니다.

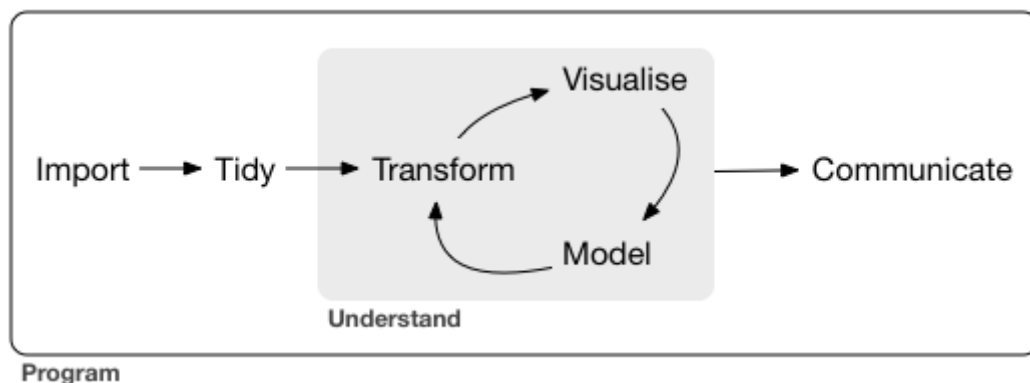


Figure 3.1: from <https://r4ds.had.co.nz/>

3.1 Tibble object type

R은 20년 이상된 비교적 오랜 역사를 가진 언어로서 `data.frame` 형태의 데이터 타입이 가장 많이 사용되고 있습니다. 그러나 당시에는 유용했던 기능이 시간이 흐르면서 몇몇 단점들이 드러나는 문제로 기존 코드를 그대로 유지한채 package 형태로 단점을 보완한 새로운 형태의 tibble 오브젝트 형식을 만들어 냈습니다. 대부분의 R 코드는 여전히 `data.frame` 형태의 데이터 타입을 사용하고 있으나 tidyverse에서는 tibble이 사용되는 것을 참고하시기 바랍니다.

```
library(tidyverse)

tb <- tibble(
  x = 1:5,
  y = 1,
  z = x ^ 2 + y
)

as_tibble(iris)
head(iris)
```

Tibble은 data.frame과 다음 몇 가지 점이 다릅니다. data.frame의 경우 타입을 변환할 때 강제로 값의 타입을 바꾸거나 내부 변수의 이름을 바꾸는 경우가 있었으나 tibble은 이를 허용하지 않습니다. 샘플들 (row) 이름을 바꿀수도 없습니다. 또한 프린팅할 때 출력물에 나오는 정보가 다르며 마지막으로 data.frame은 subset에 대한 타입이 바뀔 경우가 있었지만 tibble은 바뀌지 않습니다.

```
x <- 1:3
y <- list(1:5, 1:10, 1:20)

data.frame(x, y)
tibble(x, y)

names(data.frame(`crazy name` = 1))
names(tibble(`crazy name` = 1))

data.frame(x = 1:5, y = x ^ 2)
tibble(x = 1:5, y = x ^ 2)

df1 <- data.frame(x = 1:3, y = 3:1)
class(df1[, 1:2])
class(df1[, 1])

df2 <- tibble(x = 1:3, y = 3:1)
class(df2[, 1:2])
class(df2[, 1])
class(df2$x)
```

3.2 Tidy data structure

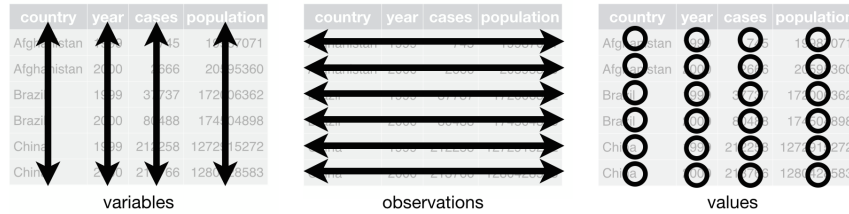
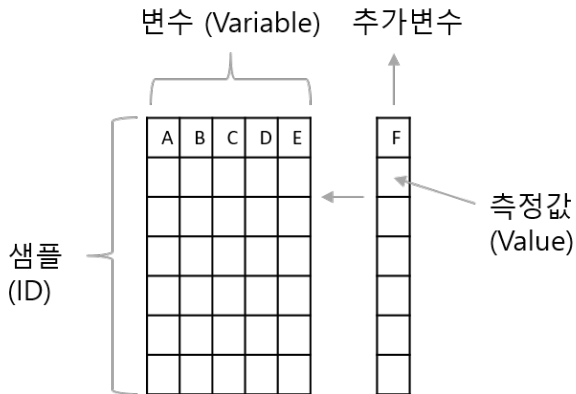
데이터의 변수와 값을 구분하는 일은 적절한 데이터 분석을 위해 필수적인 과정입니다. 특히 복잡하고 사이즈가 큰 데이터일 경우는 더욱 중요할 수 있으나 경험에 의존해서 구분을 하는 것이 대부분 입니다. Tidy data는 이러한 변수와 값의 명확한 구분과 활용을 위한 데이터 구조중 하나 입니다 (Hadley Wickham. Tidy data. The Journal of Statistical Software, vol. 59, 2014).

country	1999	2000
A	0.7K	2K
B	37K	80K
C	212K	213K

tidy data는 다음과 같은 특징이 있습니다.

- 각 변수는 해당하는 유일한 하나의 column을 가짐
- 각 샘플은 해당하는 유일한 하나의 row를 가짐
- 각 관측값은 해당하는 유일한 하나의 cell을 가짐

Tidy 데이터는 Long형 데이터로 알려져 있기도 합니다. 참고로 Wide형 데이터의 경우 샘플 데이터가 늘어날수록 row에 쌓이고 새로운 변수는 column에 쌓이는 방식으로 데이터가 확장되는 형태 입니다. 엑셀에서 볼 수 있는 일반적인 형식으로 다음 그림과 같습니다.

Figure 3.2: from <https://r4ds.had.co.nz/>

Long형 데이터의 경우 ID, variable, value 세가지 변수만 기억하면 되겠습니다. 위 wide형 데이터 경우를 보면 ID, variable, 그리고 value 이 세가지 요인이 주요 구성 요소임을 알 수 있습니다. Long형으로 변환할 경우 샘플을 참조할 수 있는 어떤 변수 (variable)도 ID가 될 수 있으며 2개 이상의 변수가 ID로 지정될 수 있습니다. 참고로 ID를 지정할 경우 해당 ID는 가능하면 중복되지 않는 값들을 갖는 변수를 사용해야 식별자로서 기능을 적절히 수행할 수 있습니다. Long형을 사용할 경우 데이터의 변수가 늘어나도 행의 수만 늘어나므로 코딩의 일관성과 변수들의 그룹을 만들어서 분석하는 등의 장점이 있습니다. 아래는 새로운 변수 F가 추가될 때 long 형 데이터에 데이터가 추가되는 경우를 나타낸 그림 입니다.

ID	variable	values
1	B	
1	C	
...	...	
2	B	
2	C	
...	...	

+

1	F	
2	F	
3	F	
4	F	
...	...	

추가변수

3.3 Pivoting

일반적으로 얻어지는 데이터의 형태는 wide형이며 이를 Long형으로 변환하기 위해서는 tidyverse 패키지에 속한 tidyr 패키지의 `pivot_longer`와 `pivot_wider`를 사용합니다. 또한 reshape2 패키지의 `melt` 함수와 그 반대의 경우 `dcast` 함수를 사용할 수도 있습니다. 본 강의에서는 tidyr 패키지를 사용합니다. wide형 데이터를 long형으로 변환하거나 long형을 wide형으로 변환하는 작업을 pivoting 이라고 합니다.

country	1999	2000	→	country	year	cases
A	0.7K	2K		A	1999	0.7K
B	37K	80K		B	1999	37K
C	212K	213K		C	1999	212K
				A	2000	2K
				B	2000	80K
				C	2000	213K

name value

airquality 데이터는 전형적인 wide형 데이터로 특정 날짜에 네 개의 변수에 해당하는 값들을 측정했습니다. 이 데이터를 long형으로 바꿀 경우 ID를 날짜로 하면 데이터들을 식별 할 수 있습니다. 그런데 날짜는 변수가 Month와 Day 두 개로 나누어져 있으므로 다음과 같이 두 변수를 식별 변수로 (ID로) 사용 합니다. 확인을 위해 상위 5개의 데이터만 가지고 형 변환을 진행해 보겠습니다.

```
airquality
```

```
myair <- airquality[1:5,]
myair_long <- pivot_longer(myair, c("Ozone", "Solar.R", "Wind", "Temp"))
myair_long
myair_long2 <- pivot_longer(myair, c(Ozone, Solar.R, Wind, Temp))
myair_long2
myair_long3 <- pivot_longer(myair, !c(Month, Day))
myair_long3
```

생성되는 long형 데이터의 변수 이름인 name과 value는 다음 파라미터를 지정하여 바꿀 수 있습니다.

```
myair_long <- pivot_longer(myair,
                           c(Ozone, Solar.R, Wind, Temp),
                           names_to = "Type",
                           values_to = "Observation")
myair_long
```

long형 데이터를 wide형 데이터로 변환 할 수도 있습니다.

```
pivot_wider(myair_long, names_from = Type, values_from = Observation)
```

3.3.1 Exercise

- 1) 다음 데이터가 long형인지 wide형인지 판단하시오
- 2) long형이면 wide형으로 wide형이면 long형으로 변환하시오

```
stocks <- tibble(
  year = c(2015, 2015, 2016, 2016),
  month = c( 1, 2, 1, 2),
  profit = c(1.88, 0.59, 0.92, 0.17)
)
```

ggplot을 이용한 그래프 작성에는 위와 같은 long형 데이터가 주로 사용됩니다. R을 이용한 데이터 가시화는 dplyr 패키지로 wide형 데이터를 편집하고 pivot_longer 함수로 long형 데이터로 변환 후 ggplot을 이용하는 방식으로 수행합니다. 두 데이터 포맷에 대한 좀 더 구체적인 내용은 다음 링크를 참고하시기 바랍니다. <https://www.theanalysisfactor.com/wide-and-long-data/>

3.4 Separating and uniting

데이터를 분석할 때 하나의 컬럼에 두 개 이상의 변수값이 저장되어 있거나 두 개의 변수를 하나의 컬럼으로 합해야 하는 경우가 종종 있습니다. 전자의 경우 separate() 함수를 사용해서 두 변수(컬럼)으로 나누어 줄 수 있으며 후자의 경우 unite() 함수를 사용하여 두 변수를 하나의 값으로 병합할 수 있습니다. 다음은 airquality 데이터에서 Month와 Day 변수를 하나의 컬럼으로 병합하여 Date라는 변수로 만들어 주는 경우의 예 입니다.

```
newairquality <- unite(airquality, Date, Month, Day, sep=".")
newairquality
```


`separate()` 함수를 사용하면 다음과 같이 해당 변수의 값을 나누어 다시 두 개의 변수(컬럼)로 나누어 줄 수 있습니다.

```
separate(newairquality, col=Date, into = c("Month", "Day"), sep = "\\.")
```

3.5 dplyr and pipe operator

`dplyr` (<https://dplyr.tidyverse.org/>) 은 `ggplot2`을 개발한 해들리위컴이 (Hadley Wickham) 중심이 되어 만들어 졌으며 `ggplot2`와 함께 `tidyverse`의 (<https://www.tidyverse.org/>) 핵심 패키지 입니다. `dplyr`은 데이터를 다루는 크기나 분석의 속도, 편의성을 향상시켜 새롭게 만들어놓은 패키지 입니다. 기존 `apply`와 같은 행렬 연산 기능과 `subset`, `split`, `group` 와 같은 행렬 편집 기능을 더하여 만들어진 도구라고 할 수 있습니다.

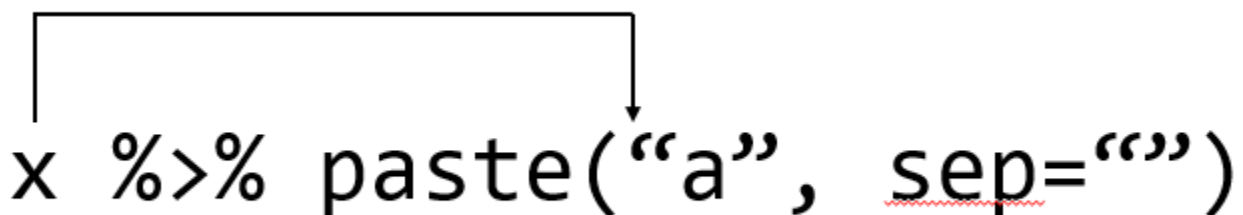
`dplyr`의 전신이라 할 수 있는 `plyr` 패키지는 다음과 같이 설명이 되어 있습니다. A set of tools for a common set of problems: you need to split up a big data structure into homogeneous pieces, apply a function to each piece and then combine all the results back together. 즉 `split-apply-combine` 세 가지 동작을 쉽게 할 수 있도록 만들어 놓은 툴 입니다. R이 다른 언어에 비해 데이터 분석에서 주목을 받는 이유로 `split`, `apply` 등의 행렬 연산 함수가 발달한 것을 내세우는데 `dplyr`은 이들을 보다 더 편리하게 사용할 수 있도록 만들어 놓은 것 입니다.

`dplyr`의 사용을 위해서는 여러 명령을 연속적으로 수행하도록 해주는 `%>%` 파이프 오퍼레이터의 이해가 필요합니다. 파이프 오퍼레이터의 작동법은 간단히 `%>%`의 왼쪽 코드의 결과를 출력으로 받아 오른쪽 코드의 입력 (첫번째 파라미터의 값)으로 받아들이는 작동을 합니다 (단축키: **Shift+Ctrl+m**). 다음 예에서 보면 `sin(pi)` 와 같은 함수의 일반적인 사용법 대신 `pi %>% sin` 처럼 사용해도 똑같은 결과를 보여줍니다. `cos(sin(pi))`와 같이 여러 함수를 중첩하여 사용할 경우와 비교해서 코드의 가독성이나 효율 측면에서 크게 향상된 방법을 제공해 줍니다.

```
library(dplyr)

pi %>% sin
sin(pi)
pi %>% sin %>% cos
cos(sin(pi))
```

특히 `%>%`는 이후 설명할 `dplyr`의 `group_by`, `split`, `filter`, `summary` 등의 행렬 편집/연산 함수를 빈번히 다양한 조합으로 쓰게되는 상황에서 더 큰 효과를 발휘할 수 있습니다.



`x %>% paste("a", sep="")`

pipe operator의 왼쪽 구문의 결과가 오른쪽 구문의 첫 번째 파라미터의 입력 값으로 처리된다고 말씀 드렸습니다. 즉, 함수에서 사용되는 파라미터가 여러개일 경우가 있으므로 기본적으로 `%>%`의 왼쪽 구문의 출력 값은 오른쪽 구문 (함수)의 첫 번째 인자의 입력값으로 들어가는 것 입니다. 이는 다음 예들을 통해서 명확히 알 수 있습니다. 먼저 `paste`함수는 그 파라미터로 ,로 구분되는 여러개의 입력 값을 가질 수 있습니다. 따라서 다음 코드는 `x`가 `paste`의 첫 번째 파라미터로 들어가게 되어 "1a", "2a", "3a", "4a", "5a"로 a 앞에 x 값들이 붙어서 출력된 것을 알 수 있습니다.

```
x <- 1:5
x %>% paste("a", sep="")
```

특정 데이터셋의 컬럼별 평균을 구하고 각 평균의 합을 구할 경우를 생각해 봅시다. R에서는 `colMeans`라는 특별한 함수를 제공하여 컬럼별로 평균을 계산해 줍니다. 그 후 `sum` 함수를 사용하여 최종 원하는 값을 얻을 수 있습니다. 이러한 코드를 `%>%` 오퍼레이터를 사용한 경우의 코드와 비교해 볼 수 있습니다.

```
x <- data.frame(x=c(1:100), y=c(201:300))
sum(colMeans(x))

x <- data.frame(x=c(1:100), y=c(201:300))
x %>% colMeans %>% sum
```

그럼 만약 두 번째 파라미터에 입력으로 왼쪽 구문의 출력을 받아들이고 싶을 경우는 `place holer .` 을 사용하면 되겠습니다. `round` 함수는 두 개의 파라미터를 설정할 수 있으며 `digits` 라는 두 번째 파라미터에 값을 pipe operator로 넘겨주고 싶을 경우 아래와 같이 표현할 수 있습니다.

```
6 %>% round(pi, digits=.)
round(pi, digits=6)
```

3.5.1 Exercise

- 1) pipe operator를 사용해서 `airquality` 데이터를 long형으로 변환하는 코드를 작성하시오 (단 `col` 파라미터에는 `Ozone`, `Solar.R`, `Wind`, `Temp` 변수를 넣음)
- 2) pipe operator를 사용해서 `airquality` 데이터의 `Month`와 `Day` 변수(컬럼)을 `Date` 변수로 병합하는 코드를 작성하시오

3.6 dplyr - Important functions

이제 `dplyr` 패키지에서 제공하는 함수를 사용해 보겠습니다. `dplyr`을 구성하는 중요한 함수는 다음과 같습니다.

- `filter()` - 샘플 (rows) 선택
- `arrange()` - 샘플들의 정렬 순서 변경
- `select()` - 변수 (columns) 선택
- `mutate()` - 새로운 변수 만들기
- `summarise()` - 대표값 만들기
- `group_by()` - 그룹별로 계산 수행
- `join()` - 두 tibble 또는 `data.frame`을 병합할 때 사용

이 함수들은 `%>%`와 함께 쓰이면서 강력한 성능을 발휘합니다. `summarise` 함수는 특정 값들의 통계 값을 계산해 주는 함수이며 그 외 함수들은 행렬 편집을 위한 함수들로 보시면 되겠습니다. 간단한 예제를 수행하면서 각각의 기능을 살펴보고 왜 `dplyr`이 널리 사용되고 그 장점이 무엇인지 파악해 보도록 하겠습니다.

`iris` 데이터는 세 종류의 `iris` 품종에 대한 꽃잎과 꽃받침의 `length`와 `width`를 측정해 놓은 데이터 입니다. `head`와 `str` 명령어를 `%>%`를 이용해서 데이터를 살펴 봅니다.

```
library(tidyverse)

iris %>% head(10)
iris %>% str
```

3.6.1 filter

먼저 아래와 같이 `filter` 함수를 사용해서 원하는 조건의 데이터 (샘플)을 골라낼 수 있습니다.

```
library(dplyr)

head(iris)
iris %>% filter(Species=="setosa")
iris %>% filter(Species=="setosa" | Species=="versicolor")
iris %>% filter(Species=="setosa" & Species=="versicolor")
iris %>%
  filter(Species=="setosa" | Species=="versicolor") %>%
  dim
```

`filter`의 ,로 구분되는 매개변수는 `and` 로직으로 묶인 조건입니다. 지난 강좌에서 보셨듯 R에서 `and`는 `&`, `or`는 `|`, 그리고 `not`은 `!` 으로 사용하면 되며 `filter`에서 ,로 구분된 조건은 `and`와 같다고 보시면 되겠습니다.

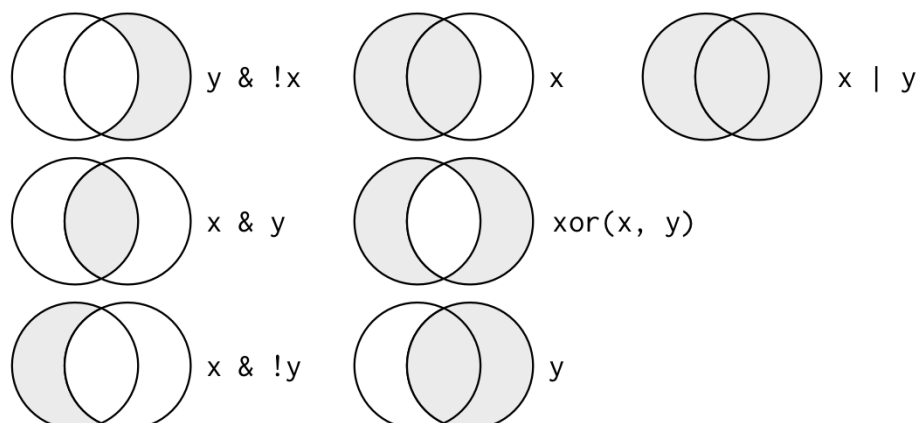


Image from (<https://r4ds.had.co.nz/>)

3.6.2 arrange

`arrange()`는 지정된 변수를 기준으로 값의 크기순서로 샘플들의 배열 순서 즉, row의 순서를 바꾸는 기능을 수행합니다. 기본으로 크기가 커지는 순서로 정렬이 진행되며 작아지는 순서를 원할 경우 `desc` 함수를 사용할 수 있습니다.

```
iris %>% arrange(Sepal.Length)
iris %>% arrange(desc(Sepal.Length))
iris %>% arrange(Sepal.Length, Sepal.Width)
```

3.6.3 select

`select()`는 주어진 데이터셋으로부터 관심있는 변수를 (column) 선택하여 보여줍니다. 다음 helper 함수들은 `select` 함수와 같이 유용하게 쓰일 수 있습니다.

`starts_with("abc")` - "abc"로 시작하는 문자열을 갖는 변수 이름 `ends_with("xyz")` - "xyz"으로 끝나는 문자열을 갖는 변수 이름 `contains("ijk")` - "ijk" 문자열을 포함하는 변수 이름 `matches("(.)\\1")` - 정규식, 반복되는 문자

```
head(iris)
iris %>% select(Species, everything()) %>% head(5)
iris %>% select(Species, everything())
iris %>% select(-Species)
iris %>% select(starts_with('S'))
iris %>% select(obs = starts_with('S'))
```

아래는 `matches` 함수를 사용한 방법입니다. 좀 더 복잡한 패턴을 적용하여 변수들을 선택할 수 있으며 `grep` 함수를 사용할 경우도 정규식 패턴을 적용할 수 있습니다.

```
iris2 <- rename(iris, aavar = Petal.Length)
select(iris2, matches("(.)\\1"))
tmp <- iris[,3:5]
colnames(iris)[grep("^S", colnames(iris))]
iris[,grep("^S", colnames(iris))]
tmp
```

아래 `(.)\\1`은 하나의 문자 .가 (어떤 문자든) 한 번 더 \\1 사용된 변수 이름을 말하며 이는 `aavar`의 `aa`밖에 없으므로 `aavar`가 선택됩니다. `grep`에서 `^`표시는 맨 처음을 나타내므로 `^S`는 S로 시작하는 문자가 되겠습니다. 따라서 `grep("^S", colnames(iris))`의 경우 컬럼 이름 중 S로 시작하는 이름은 `True`로 그렇지 않으면 `False` 값을 리턴합니다.

3.6.4 mutate

`mutate()` 함수는 새로운 변수를 추가할 수 있는 기능을 제공하며 앞에서 배웠던 `within()`과 비슷하다고 볼 수 있습니다. 아래와 같이 `mutate` 함수는 `sepal_ratio`라는 변수를 새로 만들어서 기존 `iris` 데이터들과 함께 반환해 줍니다.

```
iris2 <- iris %>% mutate(sepal_ratio = Sepal.Length/Sepal.Width)
head(iris2)
```

3.6.5 summarise

`summarise()`는 `data.frame`내 특정 변수의 값들로 하나의 요약값/대푯값을 만들어 줍니다. `summarise` 함수는 단독으로 쓰이기 보다는 `group_by()` 기능과 병행해서 쓰이는 경우에 유용하게 쓰입니다. `summarise_all()` 함수를 사용하면 모든 변수에 대해서 지정된 함수를 실행합니다.

```
iris %>% summarise(mean(Sepal.Length), m=mean(Sepal.Width))
iris %>%
  group_by(Species) %>%
  summarise(mean(Sepal.Width))

iris %>%
  group_by(Species) %>%
  summarise_all(mean)

iris %>%
  group_by(Species) %>%
  summarise(across(everything(), mean))

iris %>%
  group_by(Species) %>%
```

```
summarise_all(sd)

iris %>%
  group_by(Species) %>%
  summarise(across(everything(), sd))
```

3.6.6 join

join 함수는 데이터를 병합해주는 기능을 수행하는 함수입니다. 네 가지 종류의 함수가 있으며 (left_join(), 'right_join()', 'inner_join()', 'full_join()') (key) .by'에서 지정해준 파라미터의 값을 기준으로 기능이 수행 됩니다.

```
df1 <- data.frame(id=c(1,2,3,4,5,6), age=c(30, 41, 33, 56, 20, 17))
df2 <- data.frame(id=c(4,5,6,7,8,9), gender=c("f", "f", "m", "m", "f", "m"))

inner_join(df1, df2, by="id")
left_join(df1, df2, "id")
right_join(df1, df2, "id")
full_join(df1, df2, "id")

# vs.
cbind(df1, df2)
```

3.7 Airquality example

airquality 데이터는 뉴욕주의 몇몇 지점에서의 공기질을 측정한 데이터입니다. 데이터에서 NA를 제거하고 각 월별로 평균 오존, 자외선, 풍속, 및 온도에 대한 평균과 표준편차를 구해봅니다.

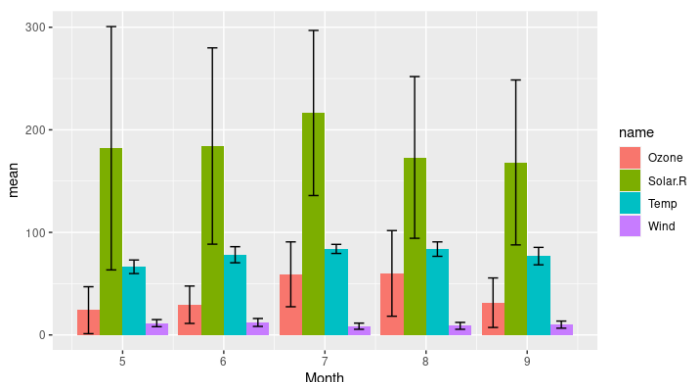
```
airmean <- airquality %>%
  filter(complete.cases(.)) %>%
  select(-Day) %>%
  group_by(Month) %>%
  summarise(across(everything(), mean)) %>%
  pivot_longer(-Month, values_to = "mean")

airsd <- airquality %>%
  filter(complete.cases(.)) %>%
  select(-Day) %>%
  group_by(Month) %>%
  summarise(across(everything(), sd)) %>%
  pivot_longer(-Month, values_to = "sd")
```

errorbar가 있는 막대그래프를 그려보겠습니다. 이를 위해서 먼저 두 테이블을 병합합니다.

```
airdata <- left_join(airmean, airds, by=c("Month", "name"))

ggplot(airdata, aes(x=Month, y=mean, fill=name)) +
  geom_bar(stat="identity", position="dodge") +
  geom_errorbar(aes(ymin=mean-sd, ymax=mean+sd), position=position_dodge(width=0.9), width=0.4)
```



ggplot2를 이용한 그래프 그리기는 다음 시간에 학습하겠습니다.

3.7.1 Exercise

1. InsectSprays 데이터는 살충제 6종에 대한 살충력을 (죽은 벌래의 마릿수) 나타내는 데이터이다. 각 살충제별로 평균과 표준편차를 구하시오
2. dplyr 패키지의 starwars 는 스타워즈 영화에 나오는 등장인물들을 분석한 데이터셋 이다. 종족에 따른 키의 평균과 표준편차를 구하시오. (NA 데이터는 제외하고 분석)

이 저작물은 크리에이티브 커먼즈 저작자표시-비영리-변경금지 4.0 국제 라이선스에 따라 이용할 수 있습니다.

Chapter 4

ggplot2 for data visualization

본 장에서는 ggplot2(<https://ggplot2.tidyverse.org/>)를 이용한 시각화에 대해서 알아봅니다. 데이터를 분석할 때 실제 데이터를 눈으로 확인하는 것은 중요합니다. 가능하면 raw 데이터를 보면서 크기 비교나 분포를 대략적으로 예측한다면 tool을 사용해서 나오는 결과를 가늠하는 척도가 될 수도 있습니다. ggplot2 는 Rstudio 개발팀의 해들리위컴이 (Hadley Wickham) 중심이 되어 만든 데이터 시각화 패키지입니다. 몇 가지 새로운 규칙을 학습해야 하지만 그 활용성이나 성능을 고려한다면 꼭 배워야할 패키지 중 하나입니다.

4.1 Basics

iris 데이터를 이용해서 간단하게 barplot을 그려봅니다. iris 데이터는 3가지 품종별 꽃잎과 꽃받침의 길이와 넓이를 측정한 데이터 입니다. 다음은 꽃잎의 길이와 넓이의 관계를 볼 수 있는 산점도 입니다.

```
library(ggplot2)
head(iris)
ggplot(data=iris) +
  geom_point(mapping=aes(x=Petal.Length, y=Petal.Width))
```

눈에 띄이는 부분은 +를 이용한 레이어들의 연결입니다. ggplot() 함수 뒤에 다양한 레이어들을 연결할 수 있고 geom_point() 함수는 지정한 위치에 산점도 레이어를 추가하는 기능을 합니다. 각 레이어들은 다음과 같은 다양한 기능을 갖는 함수들로 구성될 수 있습니다.

- 데이터 지정 (ggplot)
- 색상, 크기, x축의 값, y축의 값 등 심미적 요소 지정 (aes)
- 점, 선, 면 등 기하학적 요소 지정 (geoms)
- 그릴 통계량 지정 (stats)
- 테마, 스케일 지정 (theme)

일반적으로 ggplot을 이용하여 그래프를 그리는 순서는 다음과 같습니다.

- 어떤 그래프를 그릴지 결정
- ggplot의 데이터셋과 aesthetic 설정
- geometric 요소와 적절한 statistics를 설정한 레이어 추가
- 스케일과 테마를 설정한 레이어 추가

ggplot만을 실행할 경우 데이터와 x, y 축만 지정한 상태로 어떤 그래프 (히스토그램인지, 산포도인지 등)를 그릴지 명시되어 있지 않아서 아무것도 그리지 않은 상태의 빈 캔버스만 그려지게 되며 geom_point() 함수를 즉, 점을 그릴지 선을 그릴지 어떤 통계량을 그릴지 아니면 값 자체를 그릴지 등을 지정해 주고 나서야 비로서 그래프가 그려집니다.

```
ggplot(data=iris, mapping=aes(x=Petal.Length, y=Petal.Width))
?ggplot
ggplot(iris, aes(x=Petal.Length, y=Petal.Width))
ggplot(iris, aes(x=Petal.Length, y=Petal.Width)) + geom_point()
```

geom_point()의 도움말을 보면 다음과 같이 data, mapping, stat 등의 파라미터들이 있습니다. 이는 ggplot함수에서 설정한 data나 mapping 정보를 geom_point에서 설정 하거나 완전히 다른 데이터를 x축과 y축에 그릴 수 있다는 뜻 이기도 합니다.

```
ggplot() +
  geom_point(data=iris, mapping=aes(x=Petal.Length, y=Petal.Width))
```

그런데 위 꽃잎의 길이와 넓이는 세 가지 다른 종류의 붓꽃에 대한 정보입니다. 따라서 각 종에 따라 다른 색이나 기호를 할당하는 것도 mapping에서 설정할 수 있습니다.

```
ggplot(iris, aes(x=Petal.Length,
                 y=Petal.Width,
                 color=Species,
                 shape=Species)) +
  geom_point()

ggplot(iris, aes(x=Petal.Length, y=Petal.Width)) +
  geom_point(aes(color=Species, shape=Species))
```

위 산점도들의 `stat`은 `identity` 입니다. 즉, 따로 통계량을 계산할 필요 없이 값 그 자체를 사용하겠다는 것 입니다. 히스토그램의 경우 `geom_bar()` 함수로 막대그래프를 그릴 수 있습니다. `geom_bar`의 help페이지를 보면 `stat="count"`로 설정되어 있는 것을 알 수 있습니다. 꽃잎의 넓이에 대한 분포를 예로 구해봅시다. 히스토그램을 그릴 경우 변수 한 개의 데이터만 필요하고 y축에는 자동으로 빈도수가 들어가게 되므로 `aes`에서 x만 mapping 해 주면 됩니다.

```
ggplot(iris, aes(x=Petal.Width)) +
  geom_bar()
```

4.2 Bar graph

ggplot을 이용한 막대그래프 그리는 방법에 대해서 좀 더 알아보겠습니다. 앞서와 같이 ggplot 함수로 먼저 데이터와 aes로 x축 y축 등을 명시하고 + 오퍼레이터를 사용하여 필요한 레이어를 차례로 추가하면서 그래프를 그릴 수 있습니다. `geom_bar()` 함수의 경우 x가 연속형일 경우는 아래와 같이 히스토그램을 그려주기 어렵습니다 (위 iris 예제에서 `geom_bar()` 그래프에서는 실제 꽃받침의 width 값은 연속형이 맞으나 관측된 iris 데이터들이 같은 값들이 많은 범주형처럼 되어 있어 히스토그램 그림이 그려졌습니다) 이럴 경우 `stat`을 `bin`으로 바꿔주면 해당 범위 안에 있는 값들의 빈도수를 계산하여 히스토그램을 그릴 수 있습니다.

```
dat <- data.frame(x1=rnorm(100))
ggplot(dat, aes(x=x1)) +
  geom_bar()

ggplot(dat, aes(x=x1)) +
  geom_bar(stat="bin", bins=30)
```

x가 이산형인 경우는 `stat`을 디폴트 값인 `count`로 설정하여 해당 값들의 빈도수를 그려줄 수 있습니다. 이는 앞서 iris에서 배운 예제와 같습니다.

```
x1 <- sample(1:4, 100, replace = T)
dat <- data.frame(x=x1)
ggplot(dat, aes(x=x)) +
  geom_bar(stat="count")
```

이제 두 개의 변수가 있는 경우를 생각해 봅시다. 두 변수에 대해서 막대그래프를 그릴 경우 다음과 같이 `Error: stat_count() must not be used with a y aesthetic.` 에러가 발생할 수 있습니다.

```
x1 <- rnorm(10)
x2 <- rnorm(10)
dat <- data.frame(x1, x2)
ggplot(dat, aes(x=x1, y=x2)) +
  geom_bar()
```

이는 `geom_bar()`의 `stat`이 기본적으로 `count`로 설정되어 있으므로 생기는 에러 입니다. `stat`을 `identity`로 설정하면 x1값에 해당하는 x2값을 그려주는 막대 그래프를 그릴 수 있습니다. 참고로 이 그래프는 `geom_point`와 비슷한 정보를 보여 주게 됩니다.

```
x1 <- rnorm(10)
x2 <- rnorm(10)
dat <- data.frame(x1, x2)
ggplot(dat, aes(x=x1, y=x2)) +
  geom_bar(stat="identity")

ggplot(dat, aes(x=x1, y=x2)) +
  geom_point()
```

다음과 같이 레이어를 추가하여 두 그래프를 같은 화면에 그릴 수도 있습니다. 여기서 `col`과 `size`는 `aes`함수안에서 쓰이지 않았음을 주의하시기 바랍니다. `aes`에서는 데이터와 특정 모양, 색깔을 mapping 해주는 역할을 하고 아래와 같이 지정해 줄 경우 데이터와 상관 없이 해당 레이어의 모든 그래프에 대해서 일괄적으로 적용되게 됩니다.


```
ggplot(dat, aes(x=x1, y=x2)) +
  geom_bar(stat="identity") +
  geom_point(col="red", size=5)
```

또한 다음과 같이 다양한 레이어를 추가하여 필요한 기능을 사용할 수 있습니다. `fill=x1` 이라는 코드는 막대그래프의 색을 채울 때 `x1`에 따라서 다른 값들을 채우는 역할을 한다고 보면 되겠습니다.

```
x1 <- as.factor(1:3)
y1 <- tabulate(sample(x1, 100, replace=T))
dat <- data.frame(x1, y1)
ggplot(dat, aes(x=x1, y=y1, fill=x1)) +
  geom_bar(stat="identity") +
  guides(fill=FALSE) +
  xlab("Discrete cases") +
  ylab("Value") +
  ylim(c(0,50))+
  ggtitle("Bar graph for x:discrete and y:value")
```

4.3 Line graph

다음으로 ggplot을 이용한 line graph를 그리는 방법을 알아 봅니다. Line graph는 `geom_line`이라는 함수를 사용해서 그릴 수 있으며 `stat`의 사용법은 앞서 bar graph와 같습니다.

```
x1 <- c(12, 21, 40)
x2 <- c(33, 10, 82)
dat <- data.frame(x1, x2)
ggplot(dat, aes(x1, x2)) +
  geom_line()
```

아래와 같이 그려지는 선의 두께를 조절하거나 레이어를 추가하는 방법으로 점을 추가로 그려볼 수 있습니다. `fill`의 경우 특정 도형에 채워지는 색을 의미합니다. 도형에 대한 자세한 종류는 `?pch` 라는 도움말로 살펴보실 수 있습니다.

```
ggplot(dat, aes(x=x1, y=x2)) +
  geom_line(size=2) +
  geom_point(size=4, pch=21, fill="white") +
  guides(fill=FALSE) +
  ylim(c(0, 100)) +
  xlab("Continuous cases") + ylab("Value") +
  ggtitle("Line graph for x:continuous and y:continuous")
```

위 경우는 `x`와 `y`가 모두 연속형 데이터일 경우입니다. `x`는 이산형, `y`가 연속형일 경우 앞서와 같이 bar graph를 이용하여 그래프를 그리게 됩니다. 그런데 이런 bar의 높이에 해당하는 값들을 서로 선으로 연결하고 싶은 경우가 있습니다. 이 때는 다음과 같이 `aes`의 `group`이라는 파라미터를 설정하여 두 점 이상을 연결할 수 있습니다. 만약 `group`으로 나타낼 수 있는 변수가 없을 경우 `group=1`이라고 명시해 주고 선을 그릴 수 있으며 이 경우 모든 값들이 같은 1 그룹에 있는 것으로 간주됩니다. 1이라는 것은 하나의 예이며 어떤 숫자나 문자가 와도 괜찮습니다.

```
x1 <- as.factor(c(1:3))
y1 <- c(33, 10, 82)
dat <- data.frame(x1, y1)
str(dat)
ggplot(dat, aes(x=x1, y=y1, group=1)) +
  geom_line(stat="identity") +
  guides(fill=FALSE) +
  xlab("Discrete cases") + ylab("Value") +
  ylim(c(0,100))+
  ggtitle("Line plot for x:discrete and y:continuous")
```

위에서와 같은 방법으로 point와 bar 등을 같이 그려줄 수 있습니다.

```
ggplot(dat, aes(x=x1, y=y1, group=1)) +
  geom_bar(stat="identity", fill=x1) +
  geom_line(size=2) +
  geom_point(size=4, pch=21, fill="white") +
  guides(fill=FALSE) +
  xlab("Discrete cases") + ylab("Value") +
  ylim(c(0,100))+
  ggtitle("Line for x:discrete and y:value")
```

여기서는 fill 옵션이 geom_bar에 하나 geom_point에 하나씩 쓰였는데 geom_bar에서 사용된 fill은 bar에 채워지는 색을 x1의 값에 따라 바꾸겠다는 것을 의미하고 geom_point의 fill은 데이터에 상관 없이 모두 white로 채우라는 명령입니다. 각 geometry에 따라서 필요한 옵션이 다르므로 각각의 geom_xxx를 사용할 때 상황에 맞게 사용하시면 되겠습니다.

4.4 Smoothing

산포도는 앞서와 같이 데이터를 점으로 표현한 그래프입니다. Smoothing은 관측된 데이터를 이용하여 모형을 추정하는데 사용되는 통계적 방법이며 이를 그래프로 표현하여 추세선을 그릴 수 있습니다. 예를 들어 몸무게와 키라는 두 변수의 관계를 알아보고자 할 때 산포도를 그리고 Smoothing을 통해 점들의 평균값을 이어주는 방법으로 모형을 추정하고 추세선을 그릴 수 있습니다.

mtcars 데이터는 1974년 미국 자동차 잡지에서 추출한 데이터로서 당시 다양한 모델의 자동차에 대한 성능을 저장하고 있습니다 (?mtcars로 자세한 정보를 볼 수 있음). 이 데이터를 이용해서 연비와 마력(horsepower) 두 변수의 관계를 그래프로 그려보겠습니다. 직관적으로 생각하면 두 변수는 반비례 할 것으로 기대됩니다. ggplot을 활용해서 두 변수의 산포도를 그리고 smoothing을 수행해 보도록 하겠습니다.

```
ggplot(mtcars, aes(x=mpg, y=hp)) +
  geom_point()
```

위와 같이 mtcars는 data.frame이므로 ggplot으로 바로 받아서 x축과 y축 mapping에 필요한 변수들 이름을 직접 할당하고 geom_point함수를 이용해서 간단히 산포도를 그릴 수 있습니다. 이 산포도만으로도 mpg와 hp 두 변수간의 관계가 역함수 관계임을 알 수 있고 또한 선형이 아닌 것도 알 수 있습니다. 이제 위 그림에 geom_smooth() 함수를 이용해서 (모형) 적합 곡선 (또는 추세선)을 그려봅니다.

```
ggplot(mtcars, aes(x=mpg, y=hp)) +
  geom_point() +
  geom_smooth()
```

간단히 geom_smooth() 한 줄을 추가하여 추세선을 그렸으며 경고 메시지에서 볼 수 있듯이 알고리즘은 loess 모형을 사용했고 공식은 (formula는) y~x로, 즉, y축 변수를 반응변수로 x축 변수를 설명변수로 설정하여 그려졌습니다. 직선의 공식 $y=ax+b$ 를 생각해 보시면 무슨 의미인지 이해가 더 쉬울 듯 합니다. ?geom_smooth로 보면 알 수 있듯이 모형을 적합하는 알고리즘 옵션을 lm, glm, loess 등 다양하게 설정할 수 있으며 auto로 하게 되면 데이터의 크기나 형식에 맞춰서 방법을 자동으로 선택해서 그려주게 됩니다. se 옵션은 기본적으로 TRUE 값을 가지며 위 그림에서 볼 수 있는 선분 주위의 회색 구간으로 신뢰구간을 그려주는 옵션입니다. span 옵션은 loess 모형의 smoothing 정도를 조절할 수 있는데 이는 직접 바꿔가면서 실습을 해보면 이해에 도움이 되겠습니다.

```
ggplot(mtcars, aes(x=mpg, y=hp)) +
  geom_point() +
  geom_smooth(se=FALSE, span=0.2)
```

위와 같이 span 옵션을 작게 설정할 수록 관측된 데이터(점)에 선분(모형)이 가까이 붙게 됩니다. 이를 과대적합 (overfitting)이라고 하며 간단히 설명하면 관측된 데이터에만 너무 잘맞는 모형을 만드는 경우를 말합니다. 이럴 경우 새롭게 관측된 데이터는 모형의 예측값과 잘 맞지 않게 됩니다.

이번에는 모의 데이터를 생성해서 그래프를 그려보겠습니다. 네 개 학급에 있는 학생들의 키와 몸무게를 저장한 데이터를 만들어 봅니다. 이 경우 몇 개의 변수가 필요할지 생각해 보시기 바랍니다. 키와 몸무게 그리고 학급을 나타내는 변수 3개가 필요하며 키와 몸무게는 정수형, 그룹을 나타내는 변수는 문자형이나 factor형으로 나타내면 되겠습니다. 각 학급의 학생수는 50명으로 총 200명의 학생이 있는 것으로 하며 각 그룹별로 키나 몸무게의 차이는 없고 키가 큰 사람은 몸무게가 많이 나가는 것으로 합니다. 키와 몸무게 사이에는 다음과 같은 연관성을 만들어 줍니다. $height = weight + N(100, 10)$

```
weights <- rnorm(200, 75, 5)
heights <- weights + rnorm(200, 100, 5)
classes <- sample(c("A", "B", "C", "D"), size=length(heights), replace = T)
mydata <- data.frame(heights, weights, classes)
str(mydata)
```

이제 위 데이터를 이용해서 몸무게와 키의 산포도와 추세선을 그려보고 추가로 그룹별로 다른 색의 점으로 표현해 보겠습니다.

```
ggplot(mydata, aes(x=weights, y=heights, color=classes)) +
  geom_point() +
  geom_smooth()
```

그런데 위와 같은 코드를 실행하면 그룹마다 다른 점과 smooth 선분이 그려집니다. 우리가 원하는 그림은 단지 점만 그룹별로 다른 색으로 표현하고 추세선은 전체 학생들에 대해서 하나의 선분만 그려지길 원합니다. 이제 우리가 알아야 할 부분은 각 레이어마다 mapping을 지정할 수 있다는 것이고 이 원리를 이해한다면 다음과 같이 geom_point에서는 color를 mapping해 주고 geom_smooth에서는 지정해주지 않으면 됩니다.

```
ggplot(mydata) +
  geom_point(aes(x=weights, y=heights, color=classes)) +
  geom_smooth(aes(x=weights, y=heights))
```

그리고 중복되는 부분을 줄여줄 수도 있습니다. 즉, ggplot에서 지정하는 mapping은 하위 layer에 모두 적용이 되며 각 layer마다 다른 mapping 특성을 부여하고 싶을 경우 해당 layer의 mapping 함수 (aes)를 이용하여 설정할 수 있다는 점을 기억하시기 바랍니다.

```
ggplot(mydata, aes(x=weights, y=heights)) +
  geom_point(aes(color=classes)) +
  geom_smooth()
```

4.5 Statistics and positions

앞서 smoothing 곡선은 실제 데이터에서 관측된 값이 아닌 계산된 값을 그래프에 표현한 것 입니다. 막대그래프에서도 y축 count 값은 관측된 값이 아닌 빈도수를 계산한 값이고 boxplot의 경우도 중간값, 1,3사분위수 등 통계량을 표현해 주는 그래프 입니다. 이는 대부분 통계 분석용 소프트웨어에서 제공되는 기능으로 통계량을 가시화 해주는 역할을 합니다. ggplot2에서도 각 geom 레이어에 stat이라는 옵션을 통해 이러한 통계량을 그래프로 표현할 수 있습니다. 예를 들어 앞서 생성한 키, 몸무게 데이터에서 키의 분포를 보기 위한 히스토그램을 그리면 geom_histogram을 사용할 수 있고 이 레이어의 stat 옵션의 기본값은 "bin" 입니다 (?geom_histogram 참고).

```
library(tidyverse)

weights <- rnorm(200, 75, 5)
heights <- weights + rnorm(200, 100, 5)
classes <- sample(c("A", "B", "C", "D"), size=length(heights), replace = T)
mydata <- data.frame(heights, weights, classes)
str(mydata)

ggplot(mydata, aes(x=heights)) +
  geom_histogram()
```

경고 문구의 bins=30은 기본 stat옵션이 bin인데 bins옵션은 null로 되어 있기 때문에 경고가 발생한 것이고 30으로 강제 할당해서 그린다는 메시지 입니다. bins 옵션을 다르게 해서 테스트 해보시기 바랍니다. 또한 stat="identity"로 그래프를 그린 경우는 데이터 값을 그대로 그린다는 것도 다시 기억해 보시기 바랍니다.

```
ggplot(mydata, aes(x=heights)) +
  geom_histogram(stat="identity")

ggplot(mydata, aes(x=heights, y=weights)) +
  geom_histogram(stat="identity")
```

또 다른 예를위해 앞서 키 몸무게 데이터에 혈액형 변수를 추가해 보겠습니다. 혈액형은 위 4개 학급에 관계 없이 A, B, O, AB 네 그룹으로 나눌 수 있으며 200명의 학생들에게 랜덤하게 할당하도록 합니다.

```
bloodtype <- sample(c("A", "B", "O", "AB"), nrow(mydata), replace=T)
mynewdata <- data.frame(mydata, bloodtype)
str(mynewdata)
```

위와 같이 새로운 변수 bloodtype 이 factor형으로 추가되어 새로운 data.frame을 생성하도록 했습니다. 이제 각 학급별로 몇 명의 혈액형 타입을 갖는 학생들이 있는지를 막대그래프로 표현해 보도록 하겠습니다. 혈액형의 타입별로 다른 색으로 막대를 칠하도록 해봅시다. 막대그래프의 색은 fill옵션으로 채울수 있고 막대그래프는 geom_bar그리고 이 레이어의 stat은 기본값이 count이므로 따로 명시하지 않은채로 다음과 같이 코드를 작성할 수 있습니다.

```
ggplot(mynewdata, aes(x=classes, fill=bloodtype)) +
  geom_bar()
```

그런데 위와같이 그래프가 위로 쌓여서 보입니다. 이는 geom_bar의 position 기본값이 stack으로 되어있어서 보이는 현상입니다 (?geom_bar참고). 옆으로 나란히 막대를 위치시킨 후 크기를 비교하기 위해서 position="dodge"를 사용합니다. 또한 막대그래프에 칠해지는 색의 투명도를 alpha 옵션을 사용해 변경할 수 있습니다.

```
ggplot(mynewdata, aes(x=classes, fill=bloodtype)) +
  geom_bar(alpha=0.5, position="dodge")
```

다음과 같이 간단히 한 줄만 추가하여 위 막대그래프의 위치를 가로로 전환하거나 Coxcomb chart로 그릴수도 있습니다.

```
ggplot(mynewdata, aes(x=classes, fill=bloodtype)) +
  geom_bar(position="dodge") +
  coord_flip()
```

```
ggplot(mynewdata, aes(x=classes, fill=bloodtype)) +
  geom_bar(position="dodge") +
  coord_polar()
```

참고로 위 Coxcomb 그래프의 경우는 해석이 어렵거나 x, y축의 라벨링에 혼돈이 올수 있으니 정보 전달이 명확하도록 그래프의 옵션들을 추가하거나 용도에 맞게 사용할 필요가 있습니다.

4.6 Facets

산점도의 예에서 위와 같이 다른 색이나 모양으로 그리기 보다는 종 별로 다른 캔버스에 별도의 산점도를 그려야 할 경우가 있습니다. 이럴때 사용하는 함수가 `facet_wrap()`이나 `facet_grid()` 입니다. 보통 범주형 자료에 대해서 적용할 수 있으며 `facet_wrap()`은 하나의 변수에 대해서 그림을 나뉘고릴때 사용하고 `facet_grid()`는 두 개 변수의 조합에 의한 그래프들을 그릴 때 사용합니다. 위 붓꽃 예에서는 3가지 종을 나타내는 변수 Species를 이용하면 되겠습니다. `facet_wrap()` 함수에는 ~를 이용한 formula를 사용합니다.

```
ggplot(iris, aes(x=Petal.Length, y=Petal.Width)) +
  geom_point(aes(color=Species, shape=Species)) +
  facet_wrap(~Species, nrow=2)
```

만약 두 개의 범주형 변수에 대해서 x, y축 각각으로 나누고 싶을 때는 `facet_grid()`를 사용할 수 있습니다. iris 데이터는 하나의 범주형 변수와 네 개의 숫자형 변수로 구성되어 있습니다 (`str(iris)` 확인). 여기에 랜덤하게 0과 1을 갖는 범주형 변수 하나를 추가해 보겠습니다.

```
str(iris)
mycate <- sample(c(0,1), nrow(iris), replace=T)
myiris <- data.frame(iris, mycate)
str(myiris)
```

이제 mycate와 Species 두 범주형 변수에 대해서 facet 그래프를 그려보면 다음과 같습니다. `facet_grid()` 함수를 사용하면 되며 x와 y축의 변수는 ~를 활용한 formula를 사용합니다. 즉 ~ 왼쪽의 변수는 y축 오른쪽의 변수는 x축으로 구성되어집니다. 새로운 myiris라는 데이터를 만들었으므로 iris 대신 myiris를 사용합니다.

```
ggplot(myiris, aes(x=Petal.Length, y=Petal.Width)) +
  geom_point(aes(color=Species, shape=Species)) +
  facet_grid(Species~mycate)
```

만약 하나의 변수에 대해서 x축이나 y축 하나에만 나열하고 싶은 경우 다음처럼 . 을 사용하면 됩니다.

```
ggplot(myiris, aes(x=Petal.Length, y=Petal.Width)) +
  geom_point(aes(color=Species, shape=Species)) +
  facet_grid(.~mycate)

ggplot(myiris, aes(x=Petal.Length, y=Petal.Width)) +
  geom_point(aes(color=Species, shape=Species)) +
  facet_grid(Species~.)

ggplot(myiris, aes(x=Petal.Length, y=Petal.Width)) +
  geom_point(aes(color=Species, shape=Species)) +
  facet_grid(.~Species)
```

4.6.1 Exercise

Orange 데이터셋은 다섯 그루의 오렌지 나무에 대한 시간(age-days) 따른 성장을(circumference) 기록한 데이터임.

- 1) age와 circumference 를 각각 x와 y축으로 하는 산점도를 그리는 코드를 작성하시오 (ggplot 이용, 나무별로 다른 색 사용)
- 2) 나무별로 다른 캔버스에 age와 circumference를 x와 y축으로 하는 산점도를 그리는 코드를 작성하시오 (ggplot, facet_grid이용)
- 3) 2)에서 그려진 나무별 산점도에 다음과 같이 선분을 추가한 그래프를 그리는 코드를 작성 하시오

4.6.2 Exercise

InsectSprays는 제초제의 효능에 관한 데이터이다. 다음과 같은 plot을 그리는 코드를 작성 하시오

4.7 Themes, Labels, and Scales

Theme은 data관련 요소들 외의 것들에 대한 설정을 위해서 사용됩니다. 즉, 제목이나 라벨, 배경, 범례 등의 색, 위치, 크기, 모양 등을 설정하는데 사용합니다. 주의할 부분은 해당 텍스트 등 데이터를 변경하는 것이 아니고 보여지는 모습만을 바꿀 수 있다는 것입니다. 텍스트 설정은 labs를 사용합니다. 예제를 가지고 몇 가지 실습을 해 보겠습니다. 먼저 labs라는 명령어로 x축, y축, Title 등을 설정할 수 있습니다. 참고로 xlab(), ylab() 등의 함수도 x축, y축 라벨을 설정하는데 사용될 수 있지만 여기서는 labs만을 사용하도록 합니다.

```
ggplot(mynewdata, aes(x=classes, fill=bloodtype)) +
  geom_bar(position="dodge") +
  labs(x='Four classes',
       y='Number of students',
       title='Blood type distribution',
       subtitle = 'Blood type distribution from the 200 students',
       fill='Blood Types')
```

위 코드에서 labs에서 설정할 수 있는 옵션은 title, subtitle과 x축, y축 라벨 그리고 범례의 title까지 가능합니다. 특히 ggplot 명령에서 aes(fill=bloodtype)이 사용되었으므로 범례의 title은 fill="Blood types"로 설정해야 하며 만약 aes(color=bloodtype)으로 사용되었을 경우에는 color="Blood types"으로 설정합니다. 참고로 범례의 label을 설정하는 방법은 다음과 같이 scale_fill_discrete 함수의 labels 옵션을 사용하면 됩니다. element_blank()는 텍스트를 공백으로 설정할 때 사용합니다. 아래 나올 scale 관련 내용과 함께 이해하시면 좋습니다.

```
ggplot(mynewdata, aes(x=classes, fill=bloodtype)) +
  geom_bar(position="dodge") +
  scale_fill_discrete(name=element_blank(), labels=c("A type", "AB type", "B type", "O type"))
```

이제 본격적으로 Theme으로 그래프를 장식해 보도록 합니다. Theme 관련된 옵션들은 <https://ggplot2.tidyverse.org/reference/theme.html> 이곳을 참고하시기 바랍니다. 여기서 mapping은 그대로인채로 모양 등의 설정을 바꿔가면서 그래프의 형태를 확인하는 작업이 반복되므로 다음과 같이 myplot이라는 변수에 기본이 되는 ggplot 코드를 저장하고 이후 + 연산자를 사용해서 옵션을 바꿔가며 편리하게 코드를 재사용 할 수도 있습니다.

```
myplot <- ggplot(mynewdata, aes(x=classes, fill=bloodtype)) +
  geom_bar(position="dodge") +
  labs(x='Four classes',
       y='Number of students',
       title='Blood type distribution',
       subtitle = 'Blood type distribution from the 200 students',
       fill='Blood Types')
myplot + theme_bw()
```

위 theme_bw() 함수는 theme의 세부 사항 몇 가지를 미리 설정해 놓아서 (배경을 white 색, 눈금을 회색으로 바꾸는 등) theme 설정을 위한 일련의 과정을 한번에 수행하도록 만든 함수입니다. theme을 이용한 설정은 plot, axis, legend, panel, facet 등에 적용할 수 있으며 따라서 다음 코드와 같이 해당하는 요소를 참고할 때 . 기호로 구분된 옵션 이름을 사용합니다. 값을 지정할 때에는 element_xxx의 패턴으로 이루어진 함수를 사용합니다. 다음은 각각 plot과 panel 배경색을 바꾸는 코드입니다.

```
myplot + theme(plot.background = element_rect(fill="gray"))
```

```
myplot + theme(panel.background = element_rect(fill="gray"))
```

```
myplot +
  theme(
    panel.background = element_rect(fill="gray"),
    plot.background = element_rect(fill="gray")
  )
```

또한 축이나 라벨 텍스트의 모양도 바꿀 수 있습니다.

```
myplot +
  theme(
    axis.line = element_line(arrow = arrow(angle = 15, length = unit(.15,"inches"))),
    axis.text = element_text(face = "bold", size = 12, angle = 30),
    axis.text.x = element_text(color="blue", size=18)
  )
```

```
myplot +
  theme(
    plot.title=element_text(size=18, face = "bold", color="red", hjust=0.5),
    plot.subtitle = element_text(size=18, face = "bold", color="gray")
  )
```


위 예제 외에도 다양한 그래프를 그릴 수 있으며 모든 사용법을 외워서 사용하기 보다는 사용할 때 마다 필요한 함수와 옵션을 찾아서 사용하다 보면 점차 익숙해질 것 입니다. 가장 정확한 참고 자료는 공식 reference 페이지를 참고하면 좋으며 <https://ggplot2.tidyverse.org/reference/index.html> 이 외에도 다른 사람들이 만들어 놓은 그래프를 <https://exts.ggpilot2.tidyverse.org/> 참고해서 원하는 목적에 맞는 코드를 가져다 사용할 수 있습니다.

본 장에서 마지막으로 소개할 내용은 Scale 입니다. 앞서 어떤 데이터를 x축, y축 또는 group이나 color로 맵핑할지를 결정하는 함수가 aes였다면 scale은 어떻게 (위치, 색상, 크기, 모양 등) 맵핑할 것인가를 설정하는 방법입니다. 함수 형태는 `scale_<aesthetic>_<type>` 이며 <aesthetic>과 <type>에 해당하는 (미리 지정된) 단어를 넣어주면 되겠습니다. 예를 들어 앞서 예제에서 `fill=bloodtype`로 혈액형 데이터를 막대그래프의 색을 칠하는데 사용했다면 `scale_fill_manual` 함수로 어떤 색을 칠할지를 정해주는 방식입니다. 다음 몇 가지 예를 실습해 보고 이해해 봅시다.

```
myplot +
  scale_fill_manual(values = c("orange", "skyblue", "royalblue", "blue"))

myplot +
  scale_fill_brewer(palette="BrBG")
```

두 번째 `scale_fill_brewer`의 경우는 brewer라는 (<https://colorbrewer2.org/>) 미리 지정된 색의 조합을 가져와 사용하는 방식입니다. `?scale_fill_brewer`의 Palettes 섹션을 보시면 사용 가능한 팔레트의 이름이 나와 있으며 위 예제에서는 BrBG라는 이름의 팔레트를 사용했습니다. 아래는 viridis 라는 이름의 팔레트이며 (<https://bids.github.io/colormap/>) 이러한 팔레트는 R 뿐만 아니라 python, Matlab 등의 다른 프로그래밍 언어에서도 사용할 수 있도록 라이브러리를 제공하고 있습니다.

```
myplot +
  scale_fill_viridis_d()
```

참고로 앞서 설명한 바와 같이 `aes(fill=bloodtype)`이 사용되었으므로 `scale_fill_viridis_d`을 사용했으며 만약 `aes(color=bloodtype)`으로 사용되었을 경우에는 이에 맞는 `scale_color_viridis_d`으로 설정해야 합니다. 맵핑된 데이터가 연속형일 경우에는 (위 학급 예제의 혈액형은 4개의 혈액형으로 나뉘는 범주형 데이터임) `scale_fill_gradient`, `scale_fill_distiller` 등의 연속형 데이터에 맞는 scale 함수를 사용해야 합니다. 또한 데이터의 스케일이 log나 지수 단위일 경우에도 일 때에도 `scale_x_log10()` 등의 함수를 이용해서 x축 또는 y축의 스케일을 변경해줄 수 있습니다. 다음은 간단한 형태의 로그 분포 데이터를 생성하고 히스토그램을 그리는 코드입니다.

```
mydf <- data.frame(x=rlnorm(1000, log(10), log(2.5)))
p <- ggplot(mydf, aes(x=x)) +
  geom_histogram()
p
```

위 히스토그램의 x축을 로그 스케일로 전환하고자 할 때 다음과 같이 `scale_x_log10()` 함수를 추가하면 됩니다.

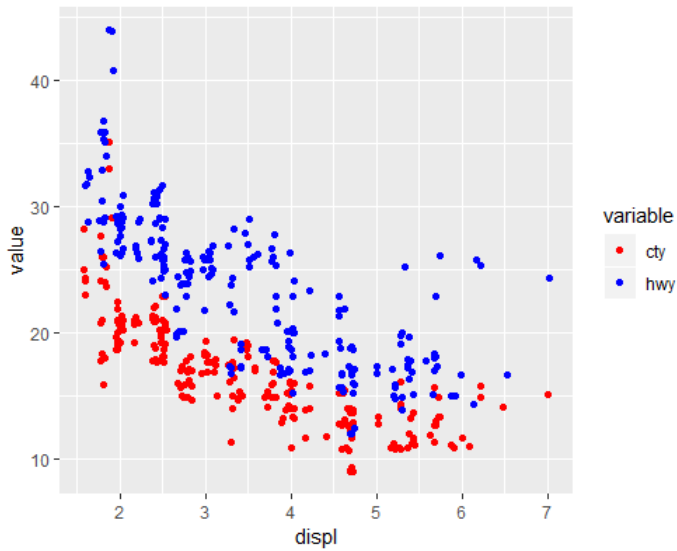
```
p + scale_x_log10()
```

4.7.1 Exercise

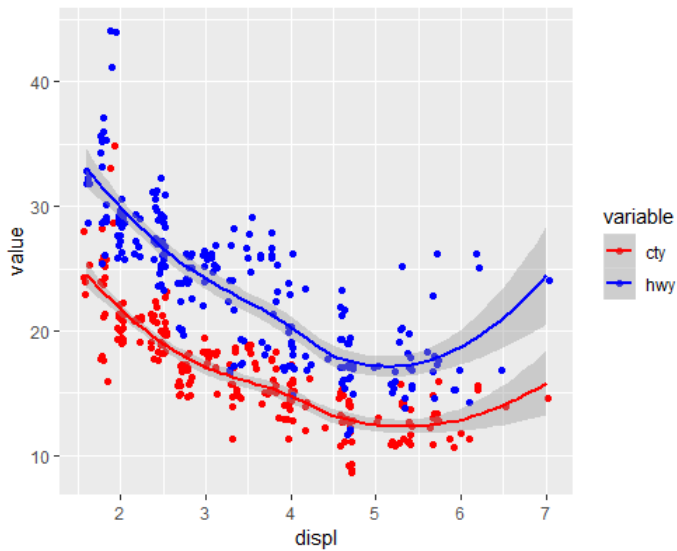
mpg 데이터셋은 38종 자동차의 연비 데이터임. 이 데이터셋을 이용하여 다음 그래프를 그리시오

- 1) 엔진 배기량과 (displ) 도심연비 (cty)를 비교하는 산포도를 그리고 어떤 연관성이 있는지 설명하시오
- 2) 위 산포도의 점들은 실제로는 한 개 이상의 데이터가 겹쳐서 표현된 경우가 많음. ggplot2에서는 이러한 문제를 극복하기 위해서 `position="jitter"` 라는 옵션을 사용할 수 있음. 이 옵션을 적용한 코드를 작성하시오.
- 3) 위 그래프에 배기량과 (displ) 고속도로연비 (hwy) 산포도를 추가하여 다음과 같이 `scale_color_manual()` 함수를 사용해서 "red"와 "blue"로 점들을 표현한 그래프를 그리시오.

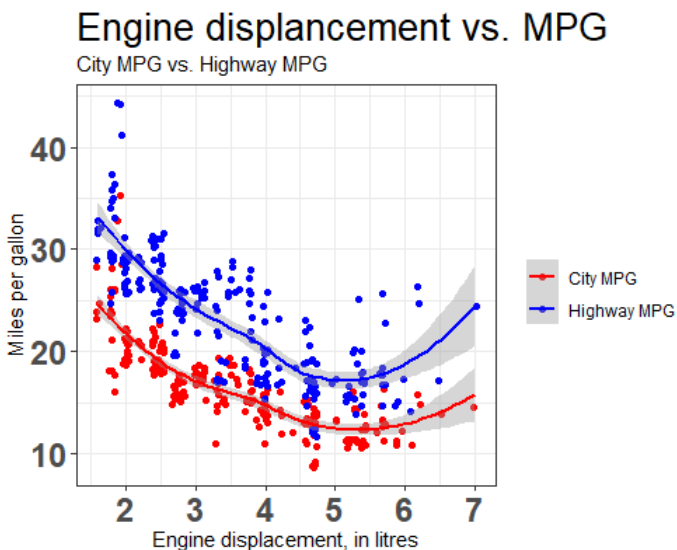
```
mydf <- data.frame(displ=mpg$displ, cty=mpg$cty, hwy=mpg$hwy) %>%
  pivot_longer(cols=c("cty", "hwy"), names_to="type")
str(mydf)
```



4) 다음과 같이 배기량과 고속도로/도심 연비의 관계를 나타내는 추세선을 추가하시오 (`geom_smooth` 이용)



5) 아래 그림과 같이 Theme을 `theme_bw()`를 사용하고 추가로 Title, subtitle, x축, y축 라벨, 그리고 범례의 Title을 변경하시오. (범례의 라벨 설정은 `scale_color_manual`에서 `labels=c("City MPG", "Highway MPG")`으로 설정, 범례의 title을 지울때는 `name=element_blank()`, Title의 텍스트 크기는 20, x축, y축의 라벨 텍스트 크기는 18로 설정)



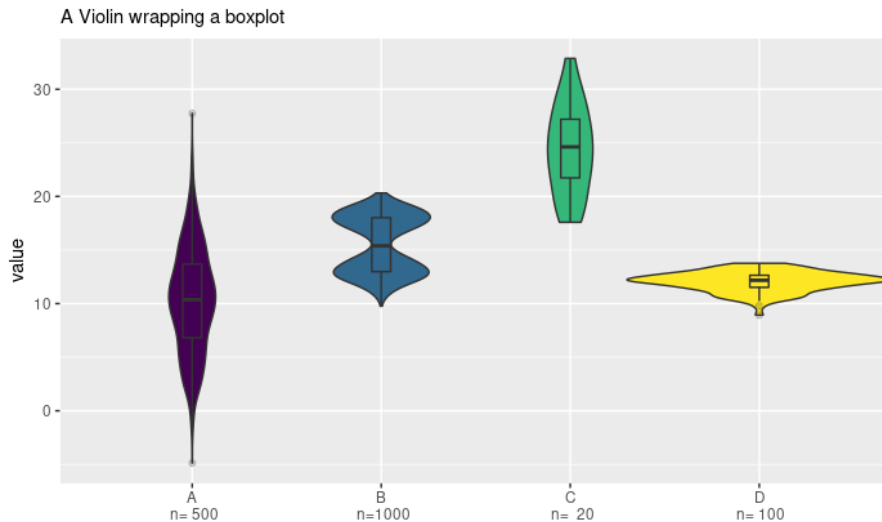
4.8 ggplot examples

인터넷에서 찾은 다음 사이트의 예제를 보면서 다양한 그래프 예제를 실행해 보겠습니다. 코드는 조금씩 변형된 부분이 있으니 참고 부탁드립니다.

- <https://www.r-graph-gallery.com/ggplot2-package.html>
- <http://r-statistics.co/Top50-Ggplot2-Visualizations-MasterList-R-Code.html>
- <https://www.datanovia.com/en/blog/ggplot-examples-best-reference/>

4.8.1 Violin plot

- https://www.r-graph-gallery.com/violin_and_boxplot_ggplot2.html



```
library(tidyverse)
library(viridis)

# create a dataset
data <- data.frame(
  name=c( rep("A",500), rep("B",500), rep("B",500), rep("C",20), rep("D", 100) ),
  value=c( rnorm(500, 10, 5), rnorm(500, 13, 1), rnorm(500, 18, 1), rnorm(20, 25, 4), rnorm(100, 12, 1) )
)

data %>% str

ggplot(data, aes(x=name, y=value, fill=name)) +
  geom_violin(width=1.4) +
  geom_boxplot(width=0.1, alpha=0.2)

# sample summary
sample_size = data %>%
  group_by(name) %>%
  summarize(num=n())

xlab <- sample_size %>%
  apply(1, function(x)paste0(x, collapse="\n n="))

apply(sample_size, 1, function(x)paste0(x, collapse="\n n="))

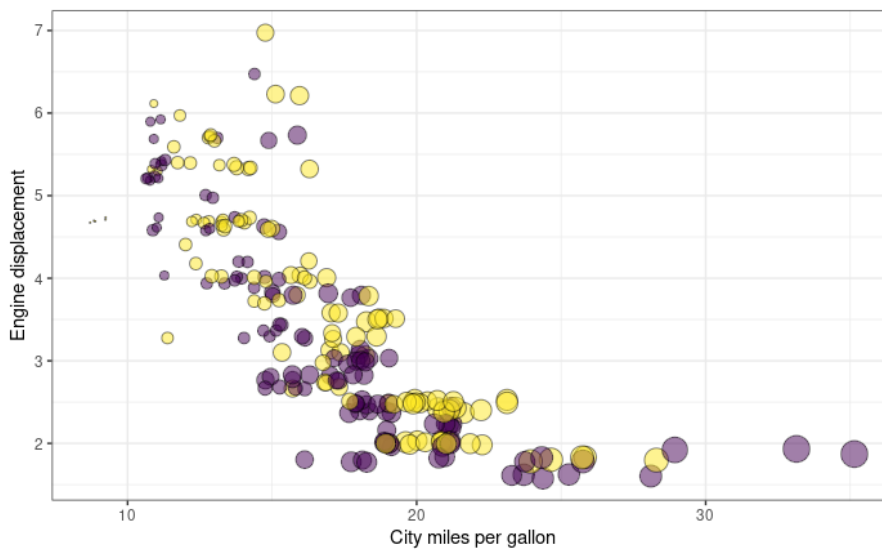
ggplot(data, aes(x=name, y=value, fill=name)) +
  geom_violin(width=1.4) +
  geom_boxplot(width=0.1, alpha=0.2) +
  scale_fill_viridis(discrete = TRUE) +
  scale_x_discrete(labels=xlab) +
  theme(
    legend.position="none",
    plot.title = element_text(size=11)
  ) +
  ggtitle("A Violin wrapping a boxplot") +
```



```
xlab("")
```

4.8.2 Bubble plot

- <https://www.r-graph-gallery.com/320-the-basis-of-bubble-plot.html>



```
mpg %>% str
```

```
# Most basic bubble plot
ggplot(mpg, aes(x=cty, y=displ, size = hwy)) +
  geom_point(alpha=0.7, position="jitter")
```

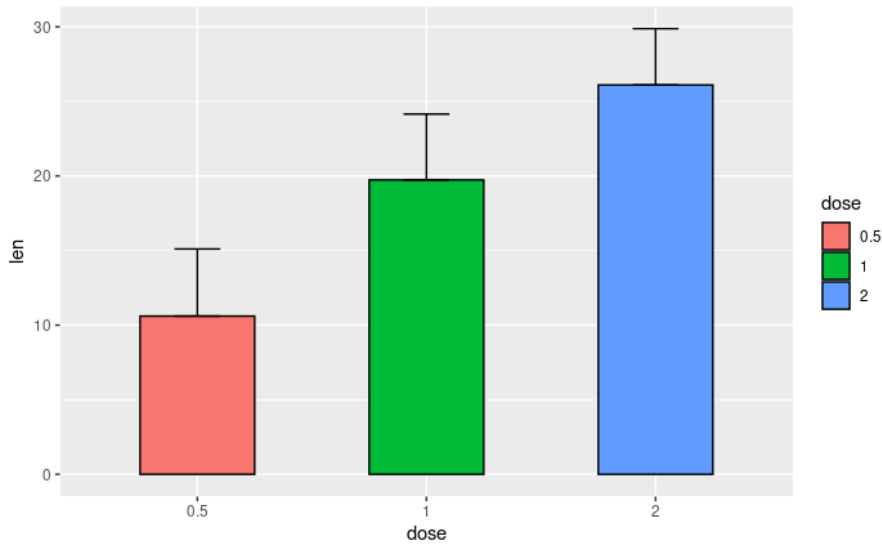
```
ggplot(mpg, aes(x=cty, y=displ, size = hwy)) +
  geom_point(alpha=0.3, position="jitter") +
  scale_size(range = c(.1, 7), name="")
```

```
ggplot(mpg, aes(x=cty, y=displ, size = hwy, color=year)) +
  geom_point(alpha=0.3, position="jitter") +
  scale_size(range = c(.1, 7), name="")
```

```
mpg %>%
  mutate(yearf = factor(year)) %>%
  ggplot(aes(x=cty, y=displ, size=hwy, color=yearf)) +
  geom_point(alpha=0.3, position="jitter") +
  scale_size(range = c(.1, 7), name="")
```

```
mpg %>%
  mutate(yearf = factor(year)) %>%
  ggplot(aes(x=cty, y=displ, size=hwy, fill=yearf)) +
  geom_point(alpha=0.5, position="jitter", shape=21) +
  scale_size(range = c(.1, 7), name="") +
  scale_fill_viridis(discrete=TRUE, guide=FALSE, option="D") +
  theme_bw() +
  ylab("Engine displacement") +
  xlab("City miles per gallon") +
  theme(legend.position = "none")
```

4.8.3 Barplot with errorbars



```
ToothGrowth %>% str
```

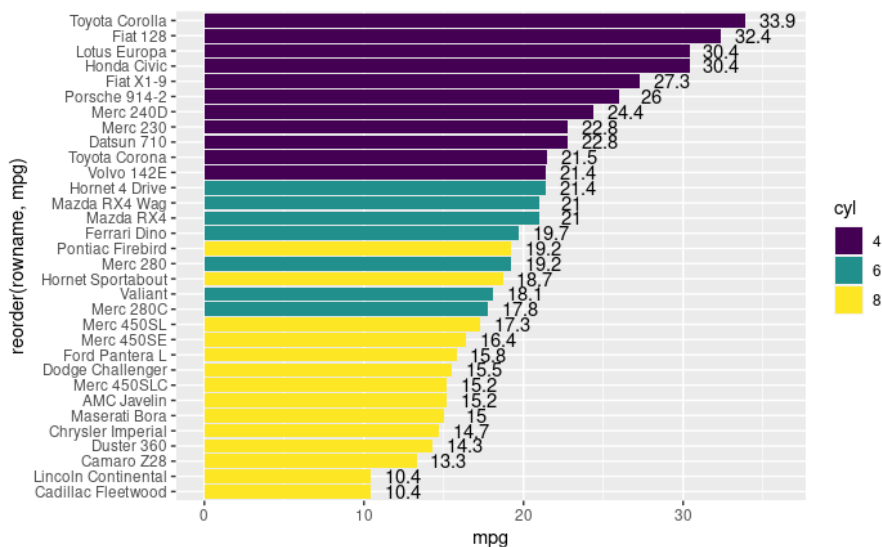
```
df <- ToothGrowth %>%
  mutate(dose = as.factor(dose))
df %>% str
```

```
## summary
```

```
df_summary <- df %>%
  group_by(dose) %>%
  summarise(sd = sd(len, na.rm = TRUE), len = mean(len))
df_summary
```

```
ggplot(df_summary, aes(x=dose, y=len, fill=dose)) +
  geom_bar(stat = "identity", color = "black", width = 0.5) +
  geom_errorbar(aes(ymin = len, ymax = len+sd), width = 0.2)
```

4.8.4 horizontal barplot



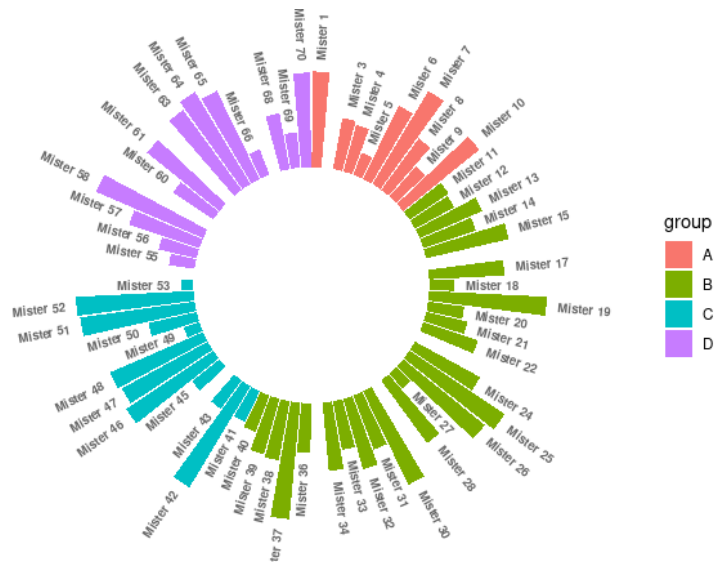
```
df <- mtcars %>%
  rownames_to_column() %>%
  as_data_frame() %>%
  mutate(cyl = as.factor(cyl)) %>%
  select(rowname, wt, mpg, cyl)
df
```

```
# change fill color by groups and add text labels
```

```
ggplot(df, aes(x = reorder(rowname, mpg), y = mpg)) +
  geom_col(aes(fill = cyl)) +
  geom_text(aes(label = mpg), nudge_y = 2) +
  coord_flip() +
  scale_fill_viridis_d()
```

4.8.5 Circular barplot

- <https://www.r-graph-gallery.com/297-circular-barplot-with-groups.html>



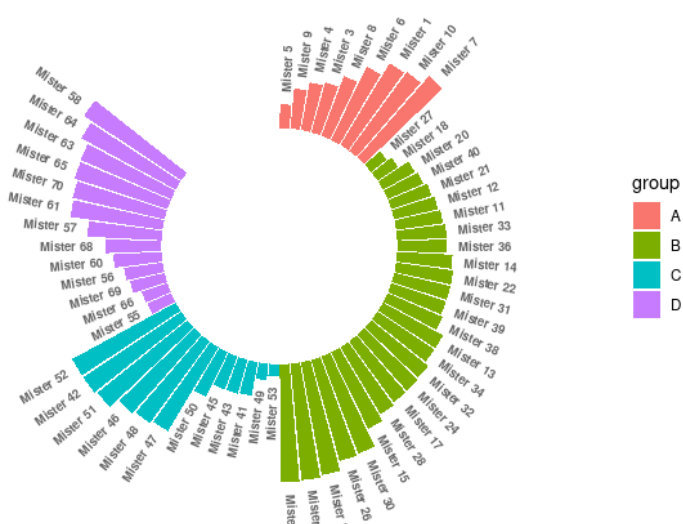
```
# Create dataset
n <- 70
data <- data.frame(
  id = seq(1, n),
  individual=paste( "Mister ", seq(1,n), sep=""),
  group=c( rep('A', 10), rep('B', 30), rep('C', 14), rep('D', n-10-30-14)) ,
  value=sample( seq(10,100), n, replace=T)
)
data %>% str

# introduce NA
empty_bar_idx <- sample(1:n, 10)
data[empty_bar_idx,c(2:4)] <- c(NA, NA, NA)

label_data <- data
number_of_bar <- nrow(label_data)
angle <- 90 - 360 * (label_data$id-0.5) /number_of_bar      # I subtract 0.5 because the letter must have
label_data$hjust <- ifelse( angle < -90, 1, 0)
label_data$angle <- ifelse(angle < -90, angle+180, angle)

data %>%
ggplot(aes(x=as.factor(id), y=value, fill=group)) +
  geom_bar(stat="identity") +
  ylim(-100,120) +
  theme_minimal() +
  theme(
    axis.text = element_blank(),
    axis.title = element_blank(),
    panel.grid = element_blank(),
    plot.margin = unit(rep(-1,4), "cm")
  ) +
  coord_polar(start = 0) +
  geom_text(data=label_data, aes(x=id, y=value+10, label=individual, hjust=hjust), color="black", fontface="bold")
```

데이터 정렬 후 plot



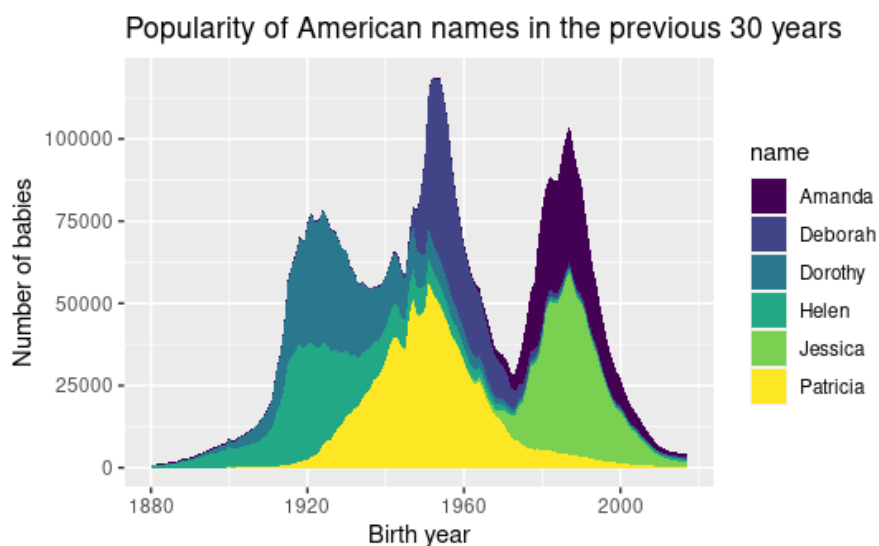
```
data2 <- data %>%
  arrange(group, value) %>%
  mutate(id2=1:n())

label_data2 <- data2
number_of_bar <- nrow(label_data2)
angle <- 90 - 360 * (label_data2$id2-0.5) / number_of_bar # I subtract 0.5 because the letter must have
label_data2$hjust <- ifelse( angle < -90, 1, 0)
label_data2$angle <- ifelse(angle < -90, angle+180, angle)

data2 %>%
  ggplot(aes(x=as.factor(id2), y=value, fill=group)) +
  geom_bar(stat="identity") +
  ylim(-100,120) +
  theme_minimal() +
  theme(
    axis.text = element_blank(),
    axis.title = element_blank(),
    panel.grid = element_blank(),
    plot.margin = unit(rep(-1,4), "cm")
  ) +
  coord_polar(start = 0) +
  geom_text(data=label_data2, aes(x=id2, y=value+10, label=individual, hjust=hjust), color="black", fontfa
```

4.8.6 Stacked area chart

- <https://www.data-to-viz.com/caveat/stacking.html>



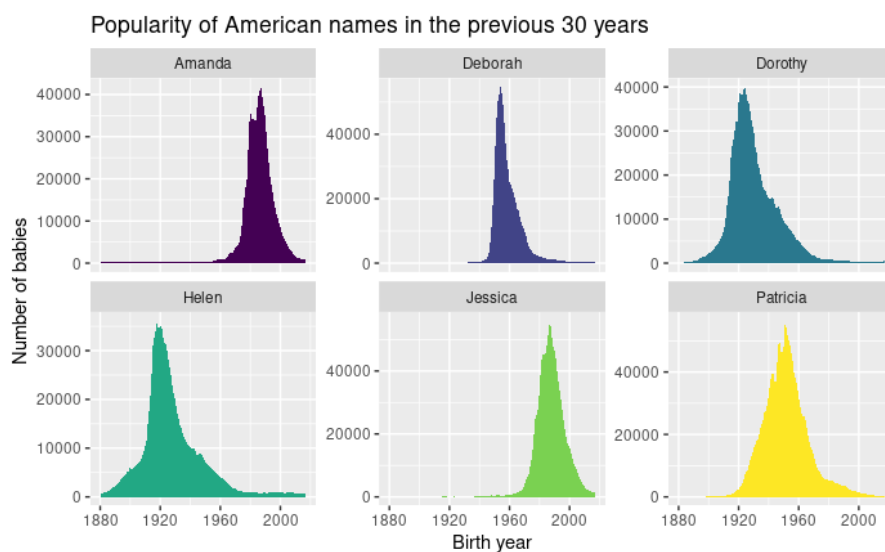
```
library(babynames)

babynames %>% str

# Load dataset from github
data <- babynames %>%
  filter(name %in% c("Amanda", "Jessica", "Patricia", "Deborah", "Dorothy", "Helen")) %>%
  filter(sex=="F")

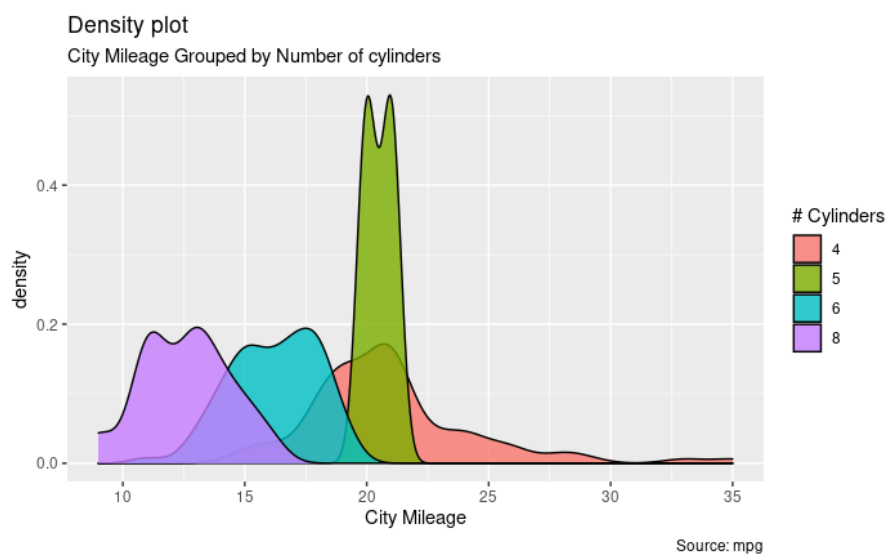
# Plot
p <- data %>%
  ggplot(aes(x=year, y=n, fill=name, text=name)) +
    geom_area( ) +
    scale_fill_viridis(discrete = TRUE) +
    ggtitle("Popularity of American names in the previous 30 years") +
    theme() +
    xlab("Birth year") +
    ylab("Number of babies")

p
```



```
p + facet_wrap(~name, scale="free_y")
```

4.8.7 Density plot



```
# Plot
g <- ggplot(mpg, aes(cty))
g + geom_density(aes(fill=factor(cyl)), alpha=0.8) +
  labs(title="Density plot",
```

```

subtitle="City Mileage Grouped by Number of cylinders",
caption="Source: mpg",
x="City Mileage",
fill="# Cylinders")

```

4.8.8 Waffle chart

- <http://r-statistics.co/Top50-Ggplot2-Visualizations-MasterList-R-Code.html#Waffle%20Chart>

```

var <- mpg$class # the categorical data

## Prep data (nothing to change here)
nrows <- 10
df <- expand.grid(y = 1:nrows, x = 1:nrows)
categ_table <- round(table(var) * ((nrows*nrows)/(length(var))))
categ_table

df$category <- factor(rep(names(categ_table), categ_table))
# NOTE: if sum(categ_table) is not 100 (i.e. nrows^2), it will need adjustment to make the sum to 100.

## Plot
df %>% str
ggplot(df, aes(x = x, y = y, fill = category)) +
  geom_tile(color = "black", size = 0.5)

```

Waffle Chart

'Class' of vehicles



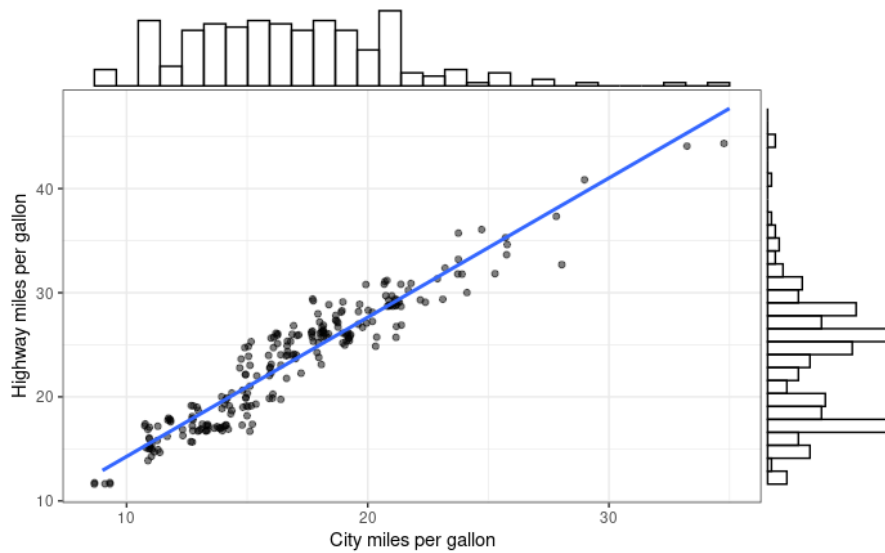
Source: mpg

```

ggplot(df, aes(x = x, y = y, fill = category)) +
  geom_tile(color = "black", size = 0.5) +
  scale_x_continuous(expand = c(0, 0)) +
  scale_y_continuous(expand = c(0, 0), trans = 'reverse') +
  scale_fill_brewer(palette = "Set3") +
  labs(title="Waffle Chart", subtitle="'Class' of vehicles",
       caption="Source: mpg") +
  theme(plot.title = element_text(size = rel(1.2)),
        axis.text = element_blank(),
        axis.title = element_blank(),
        axis.ticks = element_blank(),
        legend.title = element_blank(),
        legend.position = "right")

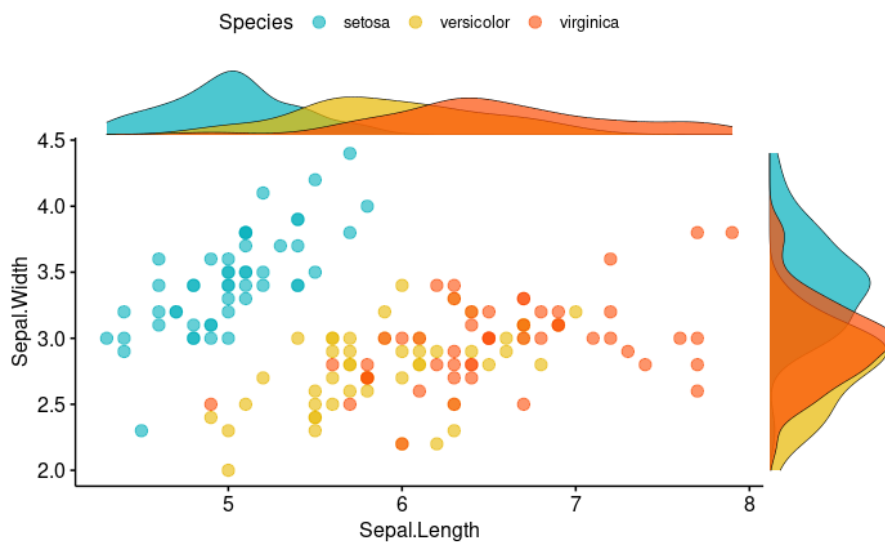
```

4.8.9 Marginal histogram



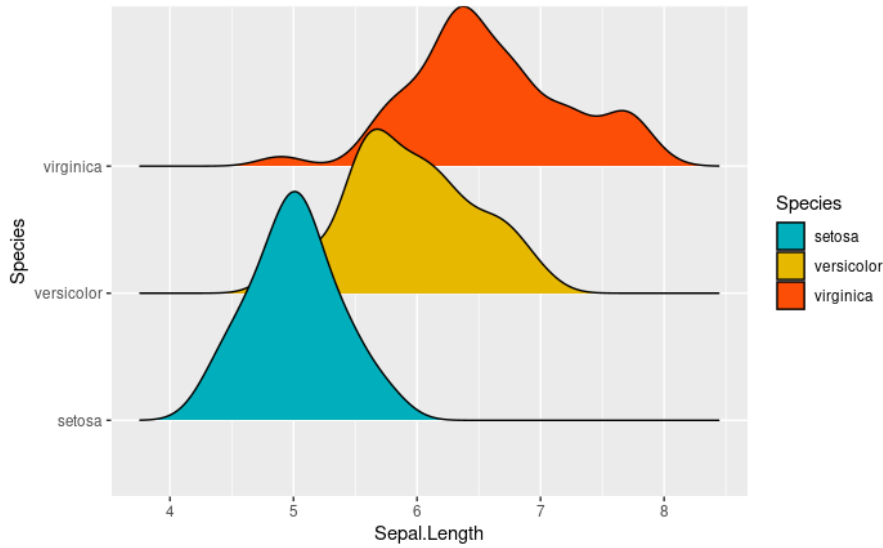
```
library(ggExtra)

# Scatterplot
p <- ggplot(mpg, aes(x=cty, y=hwy)) +
  geom_point(position="jitter", alpha=0.5) +
  geom_smooth(method="lm", se=F) +
  theme_bw() +
  theme(
    legend.position = "none"
  ) +
  xlab("City miles per gallon") +
  ylab("Highway miles per gallon")
p
ggMarginal(p, type = "histogram", fill="transparent")
ggMarginal(p, type = "density", fill="transparent")
```



```
library(ggpubr)
# Grouped Scatter plot with marginal density plots
ggscatterhist(
  iris,
  x = "Sepal.Length",
  y = "Sepal.Width",
  color = "Species",
  size = 3,
  alpha = 0.6,
  palette = c("#00AFBB", "#E7B800", "#FC4E07"),
  margin.params = list(fill = "Species", color = "black", size = 0.2)
)
```

4.8.10 Density ridgeline plots



```
library(ggribes)\nggplot(iris, aes(x = Sepal.Length, y = Species)) +\n  geom_density_ridges(aes(fill = Species)) +\n  scale_fill_manual(values = c("#00AFBB", "#E7B800", "#FC4E07"))
```

이 저작물은 크리에이티브 커먼즈 저작자표시-비영리-변경금지 4.0 국제 라이선스에 따라 이용할 수 있습니다.