R Programming for Engineering Biology (EB501, KAIST 2023)

Haseong Kim

2023-11-10

Table of contents

1 Course introduction 1

1.1 소개 . . . 1

1.2 목차 . . . 1

1.3 데이터 분석 . . . 1

1.4 라이선스 . . . 1

2 Introduction 3

2.1 R / RStudio Introduction . . . 3

2.2 Installing R / RStudio . . . 3

2.2.1 Installing R . . . 3

2.2.2 Installing RStudio . . . 3

2.2.3 Posit Cloud . . . 4

2.3 RStudio Interface and Keyboard Shortcuts . . . 4

2.4 Starting a Project . . . 4

2.5 Getting Help . . . 4

2.6 R 패키지와 데이터셋 . . . 4

2.6.1 패키지 설치 및 로드 . . . 5

2.6.2 설치 디렉토리 확인하기 . . . 5

2.6.3 패키지 데이터 사용하기 . . . 5

3 Quarto and Markdown 7

3.1 Introduction . . . 7

3.2 Quarto의 기본 작동 원리 . . . 7

3.3 코드 입력 . . . 9

3.4 Markdown 문법 . . . 10

3.5 스타일 . . . 12

3.6 테이블 . . . 12

3.7 YAML 헤더 . . . 13

3.8 Output format . . . 13

3.9 연습문제 . . . 13

3.9.1 연습문제 1: 기본 HTML 문서 생성 . . . 13

3.9.2 연습문제 2: 슬라이드 쇼 생성 . . . 14

4 R-programming 15
4.1 What is a programming language . . . 15
4.2 Console calculator . . . 16
4.2.1 Terminology . . . 16
4.3 Data and variables . . . 17
4.3.1 Data . . . 17
4.3.2 Variables . . . 17
4.4 Object (Data structure) . . . 18
4.4.1 vector . . . 18
4.4.2 matrix . . . 22
4.4.3 data.frame . . . 23
4.4.4 list . . . 24
4.5 Functions . . . 25
4.5.1 A script in R . . . 25
4.5.2 Build a function . . . 26
4.5.3 local and global variables . . . 27
4.5.4 Vectorized functions . . . 28
4.6 Flow control . . . 29
4.6.1 if statements . . . 29
4.6.2 ifelse statements . . . 30
4.6.3 for, while, repeat . . . 30
4.6.4 Avoiding Loops . . . 31
4.6.5 dplyr::if_else . . . 32
4.6.6 dplyr::case_when . . . 32
4.7 Object Oriented Programming (Advanced) . . . 32

5 Data transform classic 33
5.1 Introduction . . . 33
5.2 Reading and writing . . . 33
5.2.1 Text file . . . 33
5.2.2 Excel file . . . 34
5.3 Classical way of the data transformation . . . 35
5.4 Subset . . . 35
5.5 Merging and Split . . . 36
5.6 Transformation . . . 36
5.7 Babies example . . . 37
5.8 Useful functions . . . 38
5.9 apply . . . 40
5.10 purrr . . . 41

6 Data transform tidyverse 43
6.1 Introduction . . . 43
6.2 Tibble object type . . . 44
6.3 Tidy data structure . . . 44
6.4 Pipe operator . . . 46
6.5 Pivoting . . . 47
6.6 Separating and uniting . . . 49
6.7 dplyr . . . 49
6.7.1 select . . . 50
6.7.2 filter . . . 51
6.7.3 arrange . . . 51
6.7.4 mutate . . . 52
6.7.5 summarise . . . 52
6.7.6 join . . . 52
6.8 Code comparison . . . 53

7 Bioconductor 55
7.1 Introduction . . . 55
7.2 Packages . . . 55
7.3 Learning and support . . . 57
7.4 OOP – Class, Object and Method . . . 57
7.5 Bioconductor의 OOP . . . 58

8 Biostrings 61
8.1 Introduction . . . 61
8.2 Working with sequences . . . 61
8.2.1 XString . . . 63
8.2.2 XStringSet . . . 64
8.2.3 XStringView . . . 66
8.3 Sequence read and write . . . 66
8.4 Sequence statistics . . . 67
8.5 Pattern matching . . . 69

Chapter 1

Course introduction

1.1 소개

카이스트 공학생물학 대학원 강의용 강의노트 (2023.11)

1.2 목차

- Chapter 1: Introduction
- Chapter 2: Quarto and Markdown

1.3 데이터 분석

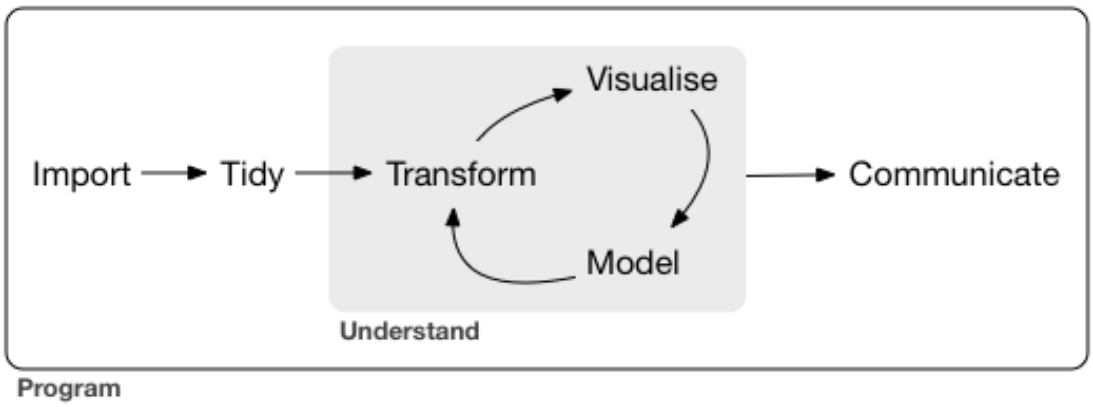


Figure 1.1: from https://r4ds.had.co.nz/

posit의 대표적 개발자인 해들리위컴은 R for Data Science의 서두에 위와 같은 그림으로 데이터과학을 설명하고 있습니다. 원본 데이터는 Tidy 형태로 R로 불러와야 하고 언제든 분석에 맞는 형태로 변형 할 수 있어야 합니다. 그리고 항상 시각화를 통해 데이터를 직접 검증해야하고 모델을 사용해서 원하는 분석을 수행합니다. 분석 후에는 공유를 통해 객관적인 검증이 필요합니다. R은 이 모든 과정을 빠르고 효율적으로 수행할 수 있는 최고의 언어입니다. 데이터과학 입장에서는 위 도구들을 둘러싸고 있는것이 프로그래밍 입니다.

1.4 라이선스

Chapter 2

Introduction

2.1 R / RStudio Introduction

R은 통계학, 생물통계학, 유전학 등의 연구 분야에서 널리 사용되는 오픈소스 프로그래밍 언어입니다. S 언어에서 파생되었고, 데이터 분석 및 가시화 패키지들이 강점입니다. 최근 인공지능의 관심이 크게 증가하면서 python 언어의 수요가 더 증가하긴 했으나 특히 중,소규모의 데이터 전처리와 가시화, 통계분석 측면에서 여전히 많은 사용자를 확보하고 있는 언어입니다.

R언어가 주목을 받고 두터운 사용자 층을 확보할 수 있게된 핵심 동력이 Rstudio 입니다. Rstudio를 개발한 Posit은 자체적으로 최고수준의 오픈소스 개발팀이 있으며 tidyverse, tidymodel, shiny 등의 데이터 분석 및 가시화 관련 주요 패키지를 개발하였고 정기적으로 conference 개최를 하면서 기술 보급의 핵심 역할을 하고 있습니다. R을 활용한 데이터 분석은 tidyverse 가 나오기 이전과 이후로 분리될 수 있을 정도로 tidyverse는 많은 변화와 효율향상을 이뤄냈습니다.

2.2 Installing R / RStudio

R 사용을 위해서는 R 언어의 코어 프로그램을 먼저 설치하고 그 다음 R 언어용 IDE(Integrated Development Environment)인 RStudio 설치가 필요합니다. Rstudio는 R 언어를 위한 오픈소스 기반 통합개발환경(IDE)으로 R 프로그래밍을 위한 편리한 기능들을 제공해 줍니다.

2.2.1 Installing R

- R 공식 웹사이트 [https://www.r-project.org/]를 방문하세요. 왼쪽 메뉴 상단에 위치한 CRAN을 클릭합니다.
- 목록에서 한국 미러 사이트 중 하나를 선택합니다.
- 'Windows용 R 다운로드'를 클릭한 다음 'base' 링크로 들어갑니다.
- 'Windows용 R x.x.x 다운로드'를 클릭하여 실행 가능한 프로그램을 다운로드합니다.
- 다운로드된 R-x.x.x-win.exe 파일을 로컬 컴퓨터에서 실행합니다 (2023년 11월 현재, R의 최신 버전은 4.3.2입니다).
- 설치 프로그램의 지시에 따라 R 언어 소프트웨어 설치를 완료합니다.

2.2.2 Installing RStudio

- Posit의 Rstudio 공식 [https://posit.co/] 웹사이트를 방문한 다음, 페이지 상단의 'Products 〉 RStudio IDE'를 클릭합니다.
- 'Open Source Edition' Free의 'Download Rstudio Desktop'을 클릭합니다.
- 'Download Rstudio'의 'Download Rstudio desktop for windows'를 클릭하여 다운로드를 시작합니다.
- 다운로드된 RStudio-x.x.x.exe 파일을 로컬 컴퓨터에서 실행합니다 (2023년 11월 현재, RStudio 데스크톱의 최신 버전은 2023.09입니다).
- 설치 가이드를 따라 설치를 완료합니다.

2.2.3 Posit Cloud

Posit Cloud는 클라우드에서 RStudio를 제공하여 사용자가 설치 및 설정 없이 브라우저에서 직접 RStudio를 사용할 수 있게 합니다.

- posit.cloud에 방문하여 사용자로 등록합니다 (Google 계정을 사용할 수 있습니다).
- 로그인하고 'Cloud Free'를 선택하여 시작합니다.
- 이 환경에서는 1GB의 RAM과 1 CPU를 무료로 제공합니다.

2.3 RStudio Interface and Keyboard Shortcuts

RStudio는 코드 편집 창, 콘솔 창, 환경 및 파일 패널을 제공합니다.

주요 키보드 단축키

- 코드 실행: Ctrl + Enter
- 콘솔 창으로 이동: Ctrl + 2
- 코드 편집 창으로 이동: Ctrl + 1
- 저장: Ctrl + S
- 코드 주석 처리: Ctrl + Shift + C

2.4 Starting a Project

'RStudio'에서 '파일 〉 새 프로젝트'로 가서 새 프로젝트를 시작할 수 있습니다.

- Hello World Example
- Create a new R file and execute the following code:

```
mystring <- "Hello ₩n world!"
cat(mystring)
print(mystring)
```

2.5 Getting Help

R은 방대한 도움말 데이터를 제공하며, 다음과 같은 명령어로 특정 함수의 도움말과 예제를 찾아볼 수 있습니다.

```
help("mean")
?mean
example("mean")
help.search("mean")
??mean
help(package="MASS")
```

RStudio 치트시트는 다양한 R 기능을 한눈에 알아볼 수 있게 만든 cheatsheet 형태의 문서를 참고할 수 있습니다. 자세한 내용은 Posit Cheatsheets를 참조하세요.

2.6 R 패키지와 데이터셋

R 패키지는 함수와 데이터셋의 묶음으로, 다른 사람들이 만들어 놓은 코드나 기능을 가져와 사용함으로써 코드 작성의 수고로움을 줄이고, 편리하고 검증된 함수(기능)를 빠르게 도입하여 사용할 수 있습니다. 예를 들어 sd() 함수는 stats 패키지에서 제공하는 함수로, 표준편차 계산을 위한 별도의 함수를 만들어서 사용할 필요가 없습니다.

```
library(UsingR)
```

2.6.1 패키지 설치 및 로드

패키지는 CRAN 또는 Bioconductor와 같은 저장소에서 구할 수 있으며, 아래와 같이 RStudio를 이용하거나 콘솔창에서 install.packages() 함수를 이용할 수 있습니다.

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

2.6.2 설치 디렉토리 확인하기

R 및 패키지의 설치 디렉토리는 다음 명령어로 확인할 수 있습니다.

```
.libPaths()
path.package()
```

2.6.3 패키지 데이터 사용하기

패키지 안에 포함된 데이터도 사용할 수 있으며 data() 함수를 이용해서 패키지 데이터를 사용자 작업공간에 복사해서 사용할 수 있습니다.

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

Chapter 3

Quarto and Markdown

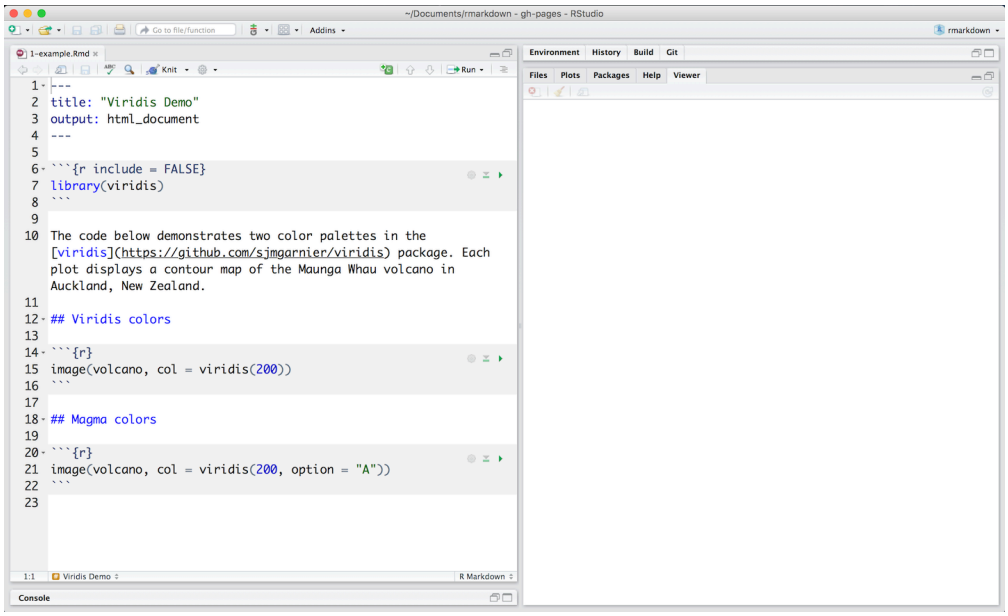
3.1 Introduction

Quarto는 데이터 과학에서 사용되는 코드와 리포트를 결합할 수 있는 통합 문서 시스템입니다. 마이크로소프트 워드나 한글과 같은 워드 프로세서에서 프로그래밍 코드와 데이터 분석을 수행할 수 있는 것처럼, Quarto를 사용하면 텍스트 기반의 마크다운 문법으로 문서를 작성하고 다양한 형식의 출력물로 변환할 수 있습니다. 이러한 출력물에는 HTML, PDF, Word 문서, 슬라이드 쇼, 책, 웹사이트 등이 포함됩니다.

Quarto에 대한 더 자세한 정보는 Quarto 공식 웹사이트에서 확인할 수 있습니다. 해당 웹사이트에는 Quarto 소개 동영상과 Quarto 공식 사이트 메뉴얼이 있으며, Quarto를 사용할 때 참고할 수 있는 cheatsheet도 제공됩니다.

3.2 Quarto의 기본 작동 원리

Quarto는 plain text 기반으로 작성되며 .qmd 라는 확장자를 갖는 파일로 저장됩니다. 다음과 같은 텍스트 파일이 Quarto 파일의 전형적인 예입니다.




```
---
title: "Viridis Demo"
output: html_document
---

```{r include = FALSE}
library(viridis)
```

The code below demonstrates two color palettes in the
[viridis](https://github.com/sjmgarnier/viridis) package. Each
plot displays a contour map of the Maunga Whau volcano in
Auckland, New Zealand.

Viridis colors

```{r}
image(volcano, col = viridis(200))
```

Magma colors

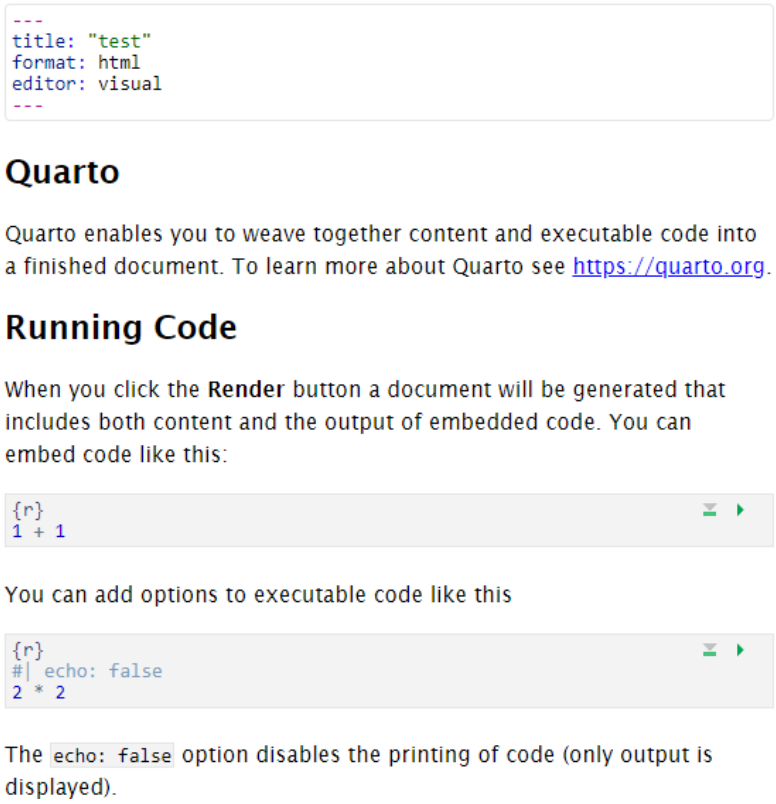
```{r}
image(volcano, col = viridis(200, option = "A"))
```
```

Quarto 문서에는 주로 세 가지 유형의 컨텐츠가 포함됩니다. 첫째, YAML 헤더는 문서의 메타데이터를 정의하며, JSON처럼 데이터를 저장하는 데 사용됩니다. 둘째, 코드 청크는 백틱 (“‘) 으로 둘러싸여 있으며, R, Python 등의 다양한 언어 코드를 실행할 수 있습니다. 셋째, 일반 텍스트와 마크다운 문법으로 작성된 내용입니다.

```
---
title: "Lecture"
format:
  html:
```

```
  toc: true
  toc-floating: true
  toc-depth: 2
  number-sections: true
---
```

이러한 Quarto 파일은 render라는 명령어로 원하는 포맷의 문서로 변환할 수 있습니다. 아래 화면은 기본 Quarto 문서 생성시 보여지는 내용이며 html 형식으로 rendering 하기 위해서는 YAML에 html 임을 명시하고 아래와 같이 render 함수를 사용하면 됩니다. 또는 Rstudio 코드 입력창 상단의 Render 버튼으로 pdf나 html 문서를 생성할 수 있습니다.



```
---
title: "test"
format: html
editor: visual
---
```

Quarto

Quarto enables you to weave together content and executable code into a finished document. To learn more about Quarto see https://quarto.org.

Running Code

When you click the **Render** button a document will be generated that includes both content and the output of embedded code. You can embed code like this:

```
{r}
1 + 1
```

You can add options to executable code like this

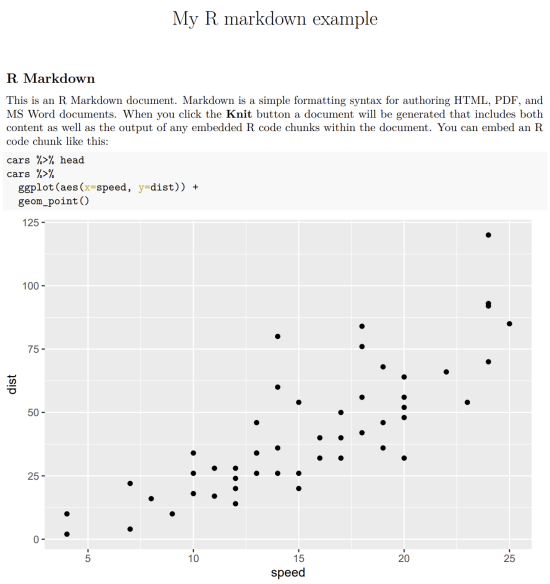
```
{r}
#| echo: false
2 * 2
```

The `echo: false` option disables the printing of code (only output is displayed).

Quarto는 내부적으로 knitr 패키지를 사용하여 R 및 기타 언어의 코드를 실행하여 .md 파일로 변환하고 .md 파일은 pandoc을 통해 HTML, PDF, Word 등 다양한 형식의 문서로 최종 변환됩니다

```
```{r}
#| eval=F

quarto::quarto_render("test.qmd", output_format = "html")
```
```


My R markdown example

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```
cars %>% head
cars %>%
  ggplot(aes(x=speed, y=dist)) +
  geom_point()
```


3.3 코드 입력

Quarto에서 코드 청크를 삽입하는 방법은 다음과 같습니다. 단축키 Ctrl + Alt + I (Windows) 또는 Cmd + Option + I (macOS)를 사용할 수 있습니다. 코드 청크의 실행과 표시 여부를 결정하는 옵션을 다음과 같이 사용할 수 있습니다:

- include: false – 코드는 실행되지만 결과와 코드를 문서에 표시하지 않습니다.
- echo: false – 코드는 실행되고 결과가 포함되지만 코드는 숨깁니다.
- eval: false – 코드를 실행하지 않고 문서에만 표시합니다.
- message: false, warning: false, error: false – 코드 실행 중 발생하는 메시지, 경고, 에러를 숨깁니다.
- fig.cap: "..." – 코드로 생성된 그림에 캡션을 추가합니다.

R 코드 청크 예시:

코드는 실행되지만 결과와 코드를 표시하지 않습니다.

````
```{r}
#| include=FALSE

summary(cars)
```
````

코드는 실행되고 결과가 포함되지만 코드는 숨깁니다.

````
```{r}
#| echo=FALSE

summary(cars)
```
````

코드를 실행하지 않고 문서에만 표시합니다.

````
```{r}
#| eval=FALSE

summary(cars)
```
````

Python 코드 청크 예시:

```
```{python}
#| eval: false
x = "hello, python in Quarto"
print(x.split(' '))
```
```

Quarto 문서에서 코드 청크와 마크다운 문법을 사용하여 내용을 서식화하고 다양한 프로그래밍 언어의 코드를 실행할 수 있습니다. Quarto에 대한 자세한 정보와 사용법은 Quarto 공식 문서에서 확인할 수 있습니다.

Quarto에서는 ‘r ‘ 을 사용해서 코드청크가 아닌 곳에 R 코드를 넣을 수 있습니다 (Inline code). 또한 R 언어 외에도 Python, SQL, Bash, Rcpp, Stan, JavaScript, CSS 등의 다양한 프로그래밍 언어에 대해서도 코드청크 기능을 사용할 수 있습니다. 그런데 이러한 언어들이 사용 가능해지기 위해서는 해당 언어들을 실행해주는 엔진이 (인터프리터) 있어야 하며 python의 경우 reticulate 라는 패키지가 이러한 기능을 담당합니다. 이 패키지를 설치할 경우 miniconda라는 가상환경 및 데이터 분석을 위한 오픈소스 패키지가 자동으로 설치됩니다.

```
```{r}
#| eval: false
library(reticulate)
```
```

위 reticulate 라이브러리를 로딩 후 아래 python 코드청크를 실행시킬 수 있습니다.

```
```{python}
#| eval: false
x = "hello, python in R"
print(x.split(' '))
```
```

3.4 Markdown 문법

마크다운은 plain text 기반의 마크업 언어로서 마크업 언어는 태그 등을 이용해서 문서의 데이터 구조를 명시하는데 이러한 태그를 사용하는 방법 체계를 마크업 언어라고 합니다. 가장 대표적으로 html 이 있습니다.

```
<html>
 <head>
  <title> Hello HTML </title>
 </head>
 <body>
 Hello markup world!
 </body>
</html>
```

마크다운도 몇 가지 태그를 이용해서 문서의 구조를 정의하고 있으며 상세한 내용은 Pandoc 마크다운 문서를 참고하시기 바랍니다. 마크다운언어의 철학은 쉽게 읽고 쓸 수 있는 문서입니다. plain text 기반으로 작성되어 쓰기 쉬우며 (아직도 사람들이 메모장 많이 사용하는 이유와 같습니다) 태그가 포함되어 있어도 읽는데 어려움이 없습니다. 위 html 언어와 아래 markdown 파일의 예들을 보시면 그 차이를 어렵지 않게 이해할 수 있습니다.

마크다운에서는 Enter를 한 번 입력해서 줄바꿈이 되지 않습니다.
 또는 문장 마지막에 공백을 두 개 입력하면 되겠습니다.

```
이 분장은 줄바꿈이<br>
됩니다
```

마크다운 테그를 몇 가지 살펴보면 먼저 # 을 붙여서 만드는 header 가 있습니다.

```
# A level-one header
## A level-two header
### A level-three header
```

```
# A level-one header {#l1-1}
## A level-two header {#l2-1}
### A level-three header {#l3-1}

# A level-one header {#l1-2}
## A level-two header {#l2-2}
### A level-three header {#l3-2}
```

인용문구를 나타내는 Block quotations이 있습니다.

> This is block quote. This paragraph has two lines

```
> This is block quote. This paragraph has two lines
```

> This is a block quote.
>
> > A block quote within a block quote.

```
> This is a block quote.
>
> > A block quote within a block quote.
```

Italic

```
*Italic*
```

Bold

```
**Bold**
```

Naver link

```
[Naver link] (https://www.naver.com/)
```

이미지를 직접 삽입하고 가운데 정렬합니다.

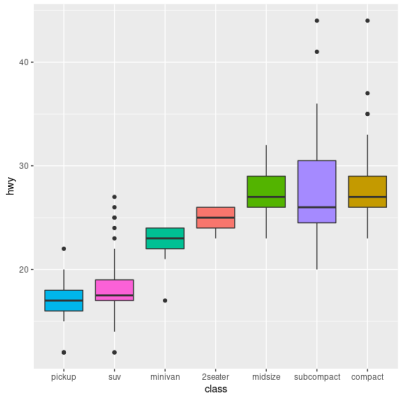


Figure 3.1: 자동차 모델에 따른 고속도로 연비 분포

```
<center>
! [자동차 모델에 따른 고속도로 연비 분포] (images/rmarkdown/000002.png) {width="200"}
</center>
```

Lists:

순서형 리스트는 숫자 해더를 사용하고 비순서형 리스트는 dash나 다른 bullet 포인트를 사용합니다.

1. 첫 번째
2. 두 번째
3. 세 번째

- 아이템 1
- 아이템 2
- 아이템 3
    - 아이템 3-1
    - 아이템 3-2

1. 첫 번째
2. 두 번째
3. 세 번째

- 아이템 1
- 아이템 2
- 아이템 3
  - 아이템 3-1
  - 아이템 3-2

참고로 소스코드 그대로 표현하기 위해서는 #| echo: fanced 를 사용하거나 두 개의 중괄호 (curly braces)를 (fanced echo 도움말) 사용하며 일반 택스트의 경우는 백틱 세 개나 (“‘) 들여쓰기를 사용합니다.

3.5 스타일

아래와 같이 YAML 해더에 css파일을 삽입하고 해당되는 class 또는 id에 해당하는 내용에 스타일을 적용할 수 있습니다.

```
.header1 {
color: red;
}
```

위와 같이 style.css 파일 작성 후 quarto파일에서 다음과 같이 사용

````
---
title: "test"
format:
  html:
    css: styles.css
---

```{r}
#| classes: header

2 + 2
```
````

3.6 테이블

kable 함수를 이용하여 Quarto 문서에 포함되는 표를 원하는 방향으로 작성할 수 있습니다. mtcars는 데이터프레임 형식의 데이터입니다.

```r
knitr::kable(
  mtcars[1:5, ],
  caption = "A knitr kable."
)
```

A knitr kable.

	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

3.7 YAML 헤더

Quarto 파일에서 YAML의 가장 중요한 기능은 output 포멧을 지정하는 것이며 title, author, date, 등을 설정할수도 있습니다.

```yaml
---
title: "R Programming"
subtitle: "Using Quarto"
format:
  html:
    css: style.css
    includes:
      in_header: header.html
      after_body: footer.html
    toc: true
    toc_float: true
    number_sections: true
mainfont: NanumGothic
---
```

3.8 Output format

Quarto의 output format 설정은 사용자가 만드는 문서의 종류와 필요에 맞게 유연하게 조정할 수 있으며, 이를 통해 다양한 형식의 고품질 문서를 생성할 수 있습니다. Quarto 문서에 대한 보다 자세한 정보는 Quarto 공식 문서에서 확인할 수 있습니다.

3.9 연습문제

3.9.1 연습문제 1: 기본 HTML 문서 생성

Quarto를 사용하여 간단한 HTML 문서를 생성합니다. 이 문서에는 자기 소개, 몇 개의 섹션, 그리고 R 코드 청크가 포함되어야 합니다.

- 새 Quarto 문서 생성

- 문서의 제목을 “My Quarto”로 설정
- YAML 헤더에 format: html 추가
- 마크다운 언어로 자기소개 섹션과 데이터 분석 섹션 추가
- “데이터 분석” 섹션에 mtcars 데이터셋의 요약 통계를 보여주는 코드 청크를 포함

  ```
  summary(mtcars)
  ```

3.9.2 연습문제 2: 슬라이드 쇼 생성

Quarto를 사용하여 간단한 슬라이드 쇼를 생성합니다. 이 슬라이드 쇼에는 여러 개의 슬라이드와 최소 하나의 인터랙티브한 R 코드 청크가 포함되어야 합니다.

- 새 Quarto 문서를 생성하고 파일 이름을 “슬라이드 쇼 연습”으로 설정
- YAML 헤더에 format: revealjs 추가
- 첫 번째 슬라이드에는 간단한 소개 및 슬라이드 쇼의 주제 작성
- 두 번째 슬라이드에는 ggplot2 사용한 mtcars 데이터셋의 mpg vs. cyl 그래프 표시

  ```
  library(ggplot2)
  ggplot(mtcars, aes(x=cyl, y=mpg)) + geom_point()
  ```

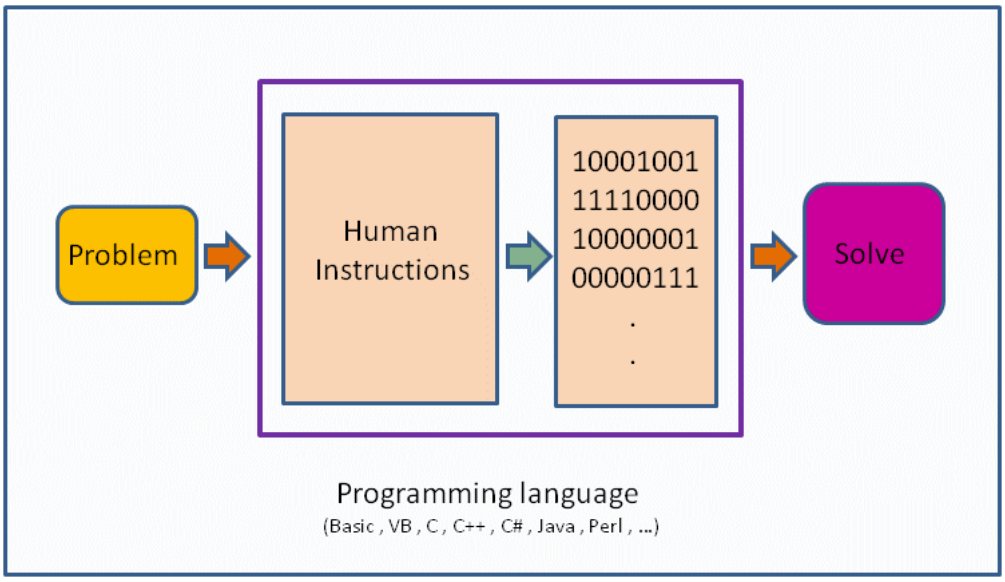
- 세 번째 슬라이드에는 결론 및 요약 포함

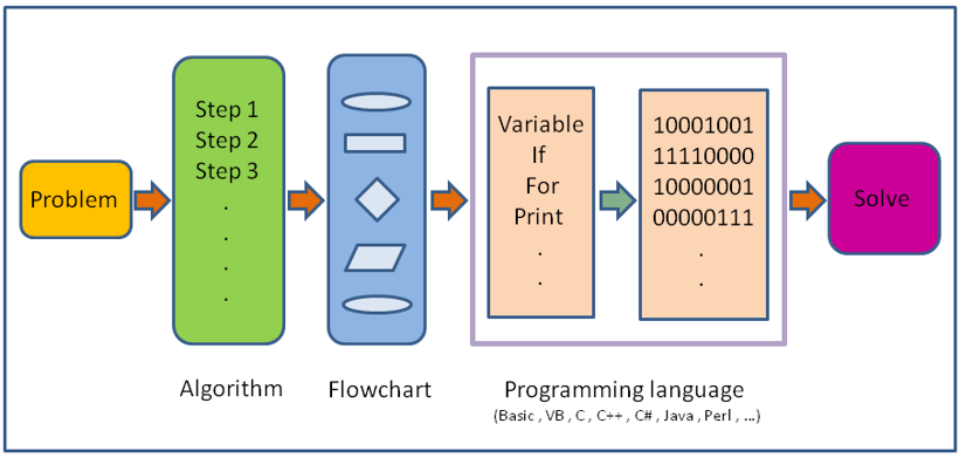
Chapter 4

R-programming

4.1 What is a programming language

프로그래밍 언어는 임의의 문제를 해결하기 위해서 만들어진 인간이 이해할 수 있는 코드를 기계가 이해할 수 있는 코드로 전환해주는 도구입니다. 즉, 인간과 기계가 소통하기 위한 언어를 말하며 프로그램은 해당 문제를 풀기위한 논리 연산의 집합으로 봅니다.





Problem
두 값 451와 224 중 높은 값을 출력하라

만약 451가 224보다 크면
452 출력
아니면
224 출력

If 451>224
print 452
else
print 224

0101000101010
0010101001010
0011110101101
1010110101010

R은 programming language로서 다른 프로그래밍 언어와 같이 몇 가지 공통적 개념을 가지고 있지만 (변수, 자료형, 함수, 조건문, 반복문) 또한 다른 언어와 차별되는 목적과 기능을 가집니다. 이번 장에서는 변수(variable, Object), 자료형(Class), 함수(Function), 조건문 및 반복문 (Flow control)에 대해서 알아보도록 합니다.

4.2 Console calculator

R은 콘솔에서 바로 계산을 수행할 수 있는 스크립트 언어입니다. 다음과 같은 연산을 콘솔에 입력하고 엔터를 누르면 결과를 바로 볼 수 있습니다. 참고로 이전에 수행한 명령은 콘솔에 커서가 있는 상태에서 위 아래 화살표를 누르면 볼 수 있고 엔터를 눌러 재사용 할 수 있습니다. ;을 사용하면 두 개의 명령을 동시에 수행할 수 있습니다.

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

Exercises

1) 다음 공식들을 계산하는 R 코드를 작성하시오

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

$$2^3 + 3^2$$

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

4.2.1 Terminology

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

4.3 Data and variables

4.3.1 Data

데이터의 의미는 사실을 나타내는 수치입니다.

- 맥도너 정보경제학 (1963)
  - 지혜 (wisdom) : 패턴화된 지식
  - 지식 (knowledge) : 가치있는 정보
  - 정보 (information) : 의미있는 데이터
  - 데이터 (data) : 단순한 사실의 나열

일반적인 데이터는 속성에 따라서 다음과 같이 분류할 수 있습니다.

1. 범주형 데이터 (Categorical Data)
   - 질적 데이터로, 숫자로 표현될 수 있지만 그 숫자들은 수치적 의미를 지니지 않습니다.
   - 명목형 (Nominal): 순서나 순위가 없는 범주를 나타냅니다. 예시: 사람 이름.
   - 순서형 (Ordinal): 순서나 순위가 있는 범주를 나타냅니다. 예시: 달리기 경기의 도착 순서.
2. 수치형 데이터 (Numerical Data)
   - 수치로 표현되며, 이 수치는 데이터의 속성을 반영합니다.
   - 구간형 (Interval): 순서가 있고, 간격이 동일하지만, 절대적인 '0'이 없는 데이터입니다. 예시: 선수들의 종점 통과 시간.
   - 비율형 (Ratio): 절대적인 '0'이 존재하며, 비율 비교가 가능한 데이터입니다. 예시: 출발시간 대비 종점 통과 시간.

이름	등수	도착	걸린시간
둘리	1	13:12	1:12
희동	5	14:30	2:30
길동	2	13:30	1:30
철수	4	14:00	2:00
영희	3	13:50	1:50

4.3.2 Variables

변수는 데이터를 저장하는 공간으로 이해할 수 있습니다.

- Assignment operator (<- OR =)
  - Valid object name <- value
  - 단축키: Alt + - (the minus sign)
- 내장 변수 Built-in variables

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

- 변수이름 작명법
  - Characters (letters), numbers, "_", "."
  - A and a are different symbols
  - Names are effectively unlimited in length

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

R에서 변수는 데이터를 담는 그릇이고 앞에서 언급한바와 같이 데이터마다 다른 특성의 타입이 있으므로 변수도 그에 맞는 타입을 갖습니다. R에서 사용하는 주요 데이터 타입은 다음과 같습니다.

1. Numeric (수치형 데이터)

- Discrete (이산형): 개별적인 값들로 이루어진 데이터. 예시: 카운트, 횟수.
- Continuous (연속형): 연속적인 값들로 이루어진 데이터. 예시: 키, 몸무게.
- Date and time: 날짜와 시간을 나타내는 데이터.

2. Factors (범주형 데이터)

- 데이터를 그룹화하는 데 사용되는 범주형 데이터
- Character 데이터를 범주형 데이터로 사용할 때 사용합니다. 예시: 데이터 식별자.

3. Character (문자형 데이터)

- 텍스트 문자열을 나타내는 데이터

4. Logical (논리형 데이터)

- 불리언 (Boolean) 값인 TRUE 또는 FALSE로 표현되는 데이터

다음은 R에서 가장 많이 이용되는 데이터 중 하나인 붓꽃 데이터이며 각각의 데이터 타입을 다음과 같이 확인할 수 있습니다.

```
data(iris)
iris
class(iris$Sepal.Length)
str(iris)
glimpse(iris)
```

4.4 Object (Data structure)

변수, 함수 등 프로그래밍에서 사용되는 모든 개체를 말합니다. 앞서 언급한 변수의 데이터 타입은 데이터 값의 타입이라고 볼 수 있고 뒤에 이야기할 vector, data.frame, list 등 데이터 타입은 변수 구조의 타입으로 볼 수 있습니다.

4.4.1 vector

vector는 R의 기본 데이터 구조입니다. numeric vector, logical vector, character vector 등 저장되는 값의 타입에 따라 크게 세가지로 나눌 수 있습니다. class() 함수를 이용해서 값의 타입을 알아낼 수 있습니다. Combine function인 c()를 활용하여 만들며 값을 순차적으로 붙여갈 수 있습니다. 다음과 같은 Univariate (단변량, Single variable)을 표현할 때 사용됩니다.

$$x_1, x_2, ..., x_n$$

```
x <- c(10.4, 5.6, 3.1, 6.4, 21.7)
class(x)
y <- c("X1", "Y2", "X3", "Y4")
```

```
class(y)
z <- c(T, F, F, T)
class(z)
```

4.4.1.1 numeric

numeric 형식의 벡터는 다음과 같은 다양한 편의 함수들을 사용해서 만들수 있습니다.

```
1:5
seq(1,5, by=1)
seq(0, 100, by=10)
seq(0, 100, length.out=11)
?seq

rep(5, times=10)
rep(1:3, times=4)
rep(1:3, each=3)
```

Exercises

1) odds라는 이름의 변수에 1부터 100까지의 홀수만을 저장하시오 (seq() 함수 사용)

인덱싱은 배열형 (vector, matrix, data.frame 등) 데이터의 일부 데이터를 참조할 때 사용하는 방법입니다. [와] 를 사용하며 위치를 나타내는 수로 참조합니다.

```
x[1]
x[1:3]
i <- 1:3
x[i]
x[c(1,2,4)]
y[3]
```

또한 해당 위치의 이름으로 참조하기도 합니다.

```
head(precip)
precip[1]
precip[2:10]
precip[c(1,3,5)]
precip[-1]
precip["Seattle Tacoma"]
precip[c("Seattle Tacoma", "Portland")]
precip[2] <- 10
```

참고로 vector 들은 다음과 같은 builtin 함수들을 사용해서 해당 변수의 attribute를 알아낼 수 있습니다. attribute에는 원소 이름, 타입, 길이 등 vector형 변수가 가질 수 있는 특성을 말합니다.

```
head(precip)
class(precip)
length(precip)
names(precip)

test_scores <- c(100, 90, 80)
names(test_scores) <- c("Alice", "Bob", "Shirley")
test_scores
```

4.4.1.2 logical

Logical 벡터는 True 또는 False를 원소로 갖는 벡터 입니다. 앞글자가 대분자로 시작하는 것을 기억하시고 T 또는 F와 같이 한 문자로 표현할 수도 있습니다. 특정 조건에 대한 판단 결과를 반환할 경우에도 논리값을 사용합니다. 이 경우 조건을 판단 후 인덱싱 방법으로 (which, any, all 등 사용) 해당 값들을 뽑아내기도 합니다. 또한 활용이 많은 sample 함수의 사용법을 익혀둡니다.

```
x <- 1:20
x > 13
temp <- x > 13
class(temp)

ages <- c(66, 57, 60, 41,  6, 85, 48, 34, 61, 12)
ages < 30
which(ages < 30)
i <- which(ages < 30)
ages[i]
any(ages < 30)
all(ages < 30)

random_number <- sample(c(1:10), 2)
```

Exercises

1) 1부터 100까지의 수를 evens이라는 이름의 변수에 저장하고 이 중 짝수만을 뽑아내서 출력하시오 (which() 함수 사용)
2) sample 함수를 사용하여 앞서 odds와 evens 변수에서 랜덤하게 1개씩의 샘플을 뽑아서 mynumbers에 저장하시오
3) 어떤 짝수가 뽑혔는지 찾아서 출력하시오 (which와 인덱싱 사용)

4.4.1.3 character

Character(문자형) 벡터의 경우 문자열을 다루는데 자주 쓰이는 paste() 함수의 사용법을 알아두면 편리합니다. paste() 함수는 서로 다른 문자열을 붙이는데 주로 사용됩니다. 참고로 문자열을 나누는 함수는 strsplit() 입니다. paste()에서 붙이는 문자 사이에 들어가는 문자를 지정하는 파라메터는 sep 이고 strsplit() 함수에서 자르는 기준이 되는 문자는split 파라미터로 지정해 줍니다 (?split 또는 ?paste 확인).

```
paste("X", "Y", "Z", sep="_")
paste(c("Four","The"), c("Score","quick"), c("and","fox"), sep="_")
paste("X", 1:5, sep="")
paste(c("X","Y"), 1:10, sep="")

x <- c("X1", "Y2", "X3", "Y4", "X5")
paste(x[1], x[2])
paste(x[1], x[2], sep="")
paste(x, collapse="_")

strsplit("XYZ", split="")
sort(c("B", "C", "A", "D"))
```

Exercises

1) m이라는 변수에 “Capital of South Korea is Seoul” 문자열을 저장하고 “Capital of South Korea”를 따로 뽑아내 m2에 저장하시오 (substr() 사용)
2) LETTERS 내장함수에서 랜덤하게 10개의 문자를 뽑아내 myletters 변수에 저장하고 이들을 연결하여 (paste 사용) 하나의 문장 (String)을 만드시오
3) myletters 변수의 문자들을 알파벳 순서대로 정렬하고 (sort 사용) 이들을 연결하여 하나의 문장 (String)을 만드시오

4.4.1.4 factor

Factor형은 범주형데이터를 저장하기 위한 object 이며 R 언어에서 특별히 만들어져 사용되고 있습니다. factor() 함수를 이용해 생성하며 생성된 객체는 다음과 같이 level이라는 범주를 나타내는 특성값을 가지고 있습니다.

예를 들어 어린이 5명이 각각 빨강, 파랑, 노랑, 빨강, 파랑 색종이를 들고 있을때 색의 종류를 나타내는 값들은 빨강, 파랑, 노랑 입니다. 다섯 명의 아이들이 어떤 색의 색종이를 들고 있는지와는 상관없이 세 가지 범주의 값을 가지는 것 입니다.

```
x <- c("Red", "Blue", "Yellow", "Red", "Blue")
y <- factor(x)
y
```

새로운 범주의 데이터를 추가할 경우 다음과 같이 해당되는 level을 먼저 추가하고 값을 저장해야 합니다.

```
levels(y)
y[1] <- "Gold"
y

levels(y) <- c(levels(y), "Gold")
levels(y)
y
y[1] <- "Gold"
y
```

factor는 기본적으로 level에 표시된 순서가 위치 (정렬) 순서입니다. 이를 바꾸기 위해서는 다음과 같이 levels 함수를 이용해서 순서를 바꿀 수 있습니다.

```
library(MASS)
str(Cars93)
x <- Cars93$Origin
plot(x)
levels(x) <- c("non-USA", "USA")
levels(x)
plot(x)
```

Exercises

UUU, UUC → Phe UUA, UUG → Leu	UCU, UCC, UCA, UCG → Ser	UAU, UAC → Tyr UAA, UAG → Stop	UGU, UGC → Cys UGA → Stop UGG → Trp
CUU, CUC, CUA, CUG → Leu	CCU, CCC, CCA, CCG → Pro	CAU, CAC → His CAA, CAG → Gln	CGU, CGC, CGA, CGG → Arg
AUU, AUC, AUA → Ile AUG → Met	ACU, ACC, ACA, ACG → Thr	AAU, AAC → Asn AAA, AAG → Lys	AGU, AGC → Ser AGA, AGG → Arg
GUU, GUC, GUA, GUG → Val	GCU, GCC, GCA, GCG → Ala	GAU, GAC → Asp GAA, GAG → Glu	GGU, GGC, GGA, GGG → Gly

1) 아미노산 Phe, Leu, Ser 를 값으로 갖는 범주형 변수 (factor)를 생성하시오

2) 각 아미노산과 해당 아미노산을 코딩하는 nucleotide triplets (codon)을 어떤 형태의 변수로 저장할 수 있을지 고민해 보시오

4.4.1.5 Missing values

특정 값이 "Not available" 이거나 "Missing value" 일 경우 벡터의 해당 원소 자리에 데이터의 이상을 알리기 위해 NA를 사용합니다. 따라서 일반적인 연산에서 NA가 포함되어 있는 경우 데이터의 불완전성을 알리기 위해 연산의 결과는 NA가 됩니다. is.na() 함수는 해당 변수에 NA 값이 있는지를 검사해주는 함수이며 R에는 이 외에도 다음과 같은 특수 값들이 사용되고 있습니다.

- NA: Not available, The value is missing
- NULL: a reserved value
- NaN: Not a number (0/0)
- Inf: (1/0)

```
hip_cost <- c(10500, 45000, 74100, NA, 83500)
sum(hip_cost)
sum(hip_cost, na.rm=TRUE)
?sum
```

4.4.1.6 Useful functions

다음은 벡터형 변수와 같이 쓰이는 유용한 함수들입니다.

```
z <- sample(1:10, 100, T)
head(z)
sort(z)
order(z)
table(z)
p <- z/sum(z)
round(p, digits=1)
```

is 함수를 사용하여 데이터 타입이 사용자가 의도한 타입과 맞는지 검사할 수 있습니다. 콘솔창에서 is.를 타이핑한 후 잠시 기다리면 다양한 is 합수를 볼 수 있습니다.

```
is.na(1)
is.numeric(1)
is.logical(TRUE)
is.data.frame("A")
is.character("A")
```

as 함수는 데이터 타입을 변환해주는 함수입니다.

```
digits <- runif(10)*10
class(digits)
digits_int <- as.integer(digits)
class(digits_int)
digits_char <- as.character(digits_int)
class(digits_char)
digits_num <- as.numeric(digits_char)
class(digits_num)
```

4.4.2 matrix

매트릭스는 2차원 행렬로 같은 형식의 데이터 값 (numberic, character, logical) 으로만 채워진 행렬을 말합니다. 메트릭스를 만드는 방법은 아래와 같으며 nrow 와 ncol 파라메터에 행과 열의 수를 넣고 각 셀에 들어갈 값은 가장 앞에 위치한 data 파라메터에 넣어 줍니다 (?matrix로 파라메터 이름 확인). 메트릭스 인덱싱은 메트릭스 안의 값을 저장하거나 참조할때 (빼올때) 사용하는 방법입니다. 메트릭스 변수이름 바로 뒤에 대괄호를 이용해서 제어를 하며 대괄호 안에 콤마로 구분된 앞쪽은 row, 뒷쪽은 column 인덱스를 나타냅니다.

```
mymat <- matrix(0, nrow=100, ncol=3) # 1
mymat[,1] <- 1:100 # 2
mymat[,2] <- seq(1,200,2) # 3
mymat[,3] <- seq(2,200,2) # 4
```

매트릭스의 row나 column에 이름이 주어져 있을 경우 이름을 따옴표(“)로 묶은 후 참조가 가능합니다. row나 column의 이름은 rownames() 또는 colnames()로 생성하거나 변경할 수 있습니다. row나 column의 개수는 nrow() 또는 ncol() 함수를 사용합니다.

```
colnames(mymat)
colnames(mymat) <- c("A", "B", "C")
colnames(mymat)
colnames(mymat)[2] <- "D"
colnames(mymat)
rownames(mymat) <- paste("No", 1:nrow(mymat), sep="")
rownames(mymat)
```

여러 row나 column을 참조할 경우 아래와 같이 combine 함수를 사용하여 묶어줘야 하며 스칼라값을 (임의의 숫자 하나) 더하거나 뺄 경우 vector / matrix 연산을 기본으로 수행합니다.

```
mymat[c(2,3,4,5),2] # 5
mymat-1 # 6
mysub <- mymat[,2] - mymat[,1] #7
sum(mysub) #8
sum(mysub^2) #8
```

Exercises

1) score 라는 변수에 1부터 100까지 중 랜덤하게 선택된 20개의 수로 10 x 2 matrix를 만드시오 (sample() 사용)
2) score의 row 이름을 문자형으로 Name1, Name2, …, Name10으로 지정하시오 (paste() 사용)
3) score의 column 이름을 문자형으로 math와 eng로 지정하시오
4) 이 matrix의 첫번째 컬럼과 두 번째 컬럼의 수를 각각 더한 후 total_score라는 변수에 저장하시오
5) total_score의의 오름차순 순서를 나타내는 인덱스 (order()함수 사용)를 o라는 변수에 저장하시오
6) score를 o순서로 재배치하고 score_ordered 변수에 저장하시오

4.4.3 data.frame

데이터프레임은 형태는 매트릭스와 같으나 컬럼 하나가 하나의 vector형 변수로서 각 변수들이 다른 모드의 값을 저장할 수 있다는 차이가 있습니다. $ 기호를 이용하여 각 구성 변수를 참조할 수 있습니다. 컬럼 한 줄이 하나의 변수 이므로 새로운 변수도 컬럼 형태로 붙여 넣을 수 있습니다. 즉, 각 row는 샘플을 나타내고 각 column은 변수를 나타내며 각 변수들이 갖는 샘플의 개수 (row의 길이, vector 의 길이)는 같아야 합니다. R 기반의 데이터 분석에서는 가장 선호되는 데이터 타입이라고 볼 수 있습니다.

```
## data.frame
ids <- 1:10
ids
idnames <- paste("Name", ids, sep="")
idnames
students <- data.frame(ids, idnames)
students
class(students$ids)
class(students$idnames)
students$idnames
str(students)
```

```
students <- data.frame(ids, idnames, stringsAsFactors = F)
class(students$idnames)
students$idnames
students[1,]
str(students)
```

데이터프레임에서는 $를 사용하여 변수 이름으로 인덱싱이 가능합니다.

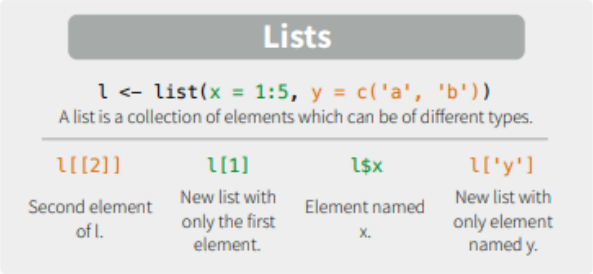
```
## data frame indexing
students$ids
students[,1]
students[,"ids"]
```

Exercises

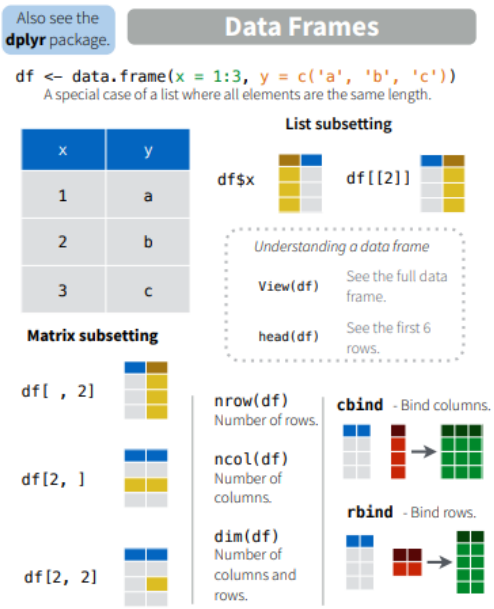
1) math라는 변수에 1부터 100까지 중 랜덤하게 선택된 10개의 수를 넣으시오
2) eng라는 변수에 1부터 100까지 중 랜덤하게 선택된 10개의 수를 넣으시오
3) students라는 변수에 문자형으로 Name1, Name2, …, Name10으로 지정하시오 (paste() 사용)
4) math와 eng라는 벡터에 저장된 값들의 이름을 students 변수에 저장된 이름으로 지정하시오
5) math와 eng 벡터를 갖는 score 라는 data.frame을 만드시오
6) math와 eng 변수를 지우시오 (rm()사용)
7) score data frame의 math와 eng를 각각 더한 후 total_score라는 변수에 저장 하시오

4.4.4 list

리스트는 변수들의 모임이라는 점에서 데이터프레임과 같으나 구성 변수들의 길이가 모두 같아야 하는 데이터프레임과는 달리 다른 길이의 변수를 모아둘 수 있는 점이 다릅니다. 즉, R언어에서 두 변수를 담을 수 있는 데이터 타입은 list와 data frame 두 종류가 있는데 list 변수 타입은 vector 형태의 여러개의 element를 가질 수 있으며 각 vector의 길이가 모두 달라도 됩니다. list의 인덱싱에서 [ ]는 리스트를 반환하고 [[ ]]는 vector element들을 반환합니다.




```
## list
parent_names <- c("Fred", "Mary")
number_of_children <- 2
child_ages <- c(4, 7, 9)
data.frame(parent_names, number_of_children, child_ages)
lst <- list(parent_names, number_of_children, child_ages)
lst[1]
lst[[1]]
class(lst[1])
class(lst[[1]])
lst[[1]][1]
lst[[1]][c(1,2)]
```



Exercises

1) 위 아미노산 예제에서 Phe, Leu, Ser 각각의 코돈을 원소로 갖는 세 개의 vector 변수들을 만들고 이를 aalist 라는 이름의 하나의 리스트 변수로 만드시오

2) aalist 리스트를 data.frame 형식의 aadf 변수로 만드시오 (데이터 구조를 바꾸어 저장 가능)

4.5 Functions

함수(Function)란 사용자가 원하는 기능을 수행하는 코드의 모음으로서 반복적으로 쉽게 사용할 수 있도록 만들어 놓은 코드 입니다.

4.5.1 A script in R

함수의 개념을 배우기 전에 스크립트를 활용한 명령어 수행을 알아보겠습니다. R 프로그래밍을 통해서 사용자가 원하는 기능을 수행하는 방법은 다음과 같이 스크립트를 만들어서 실행하는 것 입니다. 일반적으로 R을 이용한 스크립트 명령을 어떻게 실행하는지 알아보겠습니다. 다음 예제는 입력 값들의 평균을 계산해서 출력해 주는 스크립트 명령입니다. R base 패키지에서 기본으로 제공되는 mean()이라는 함수가 있지만 사용하지 않고 sum()과 length() 함수를 사용했습니다.

```
numbers <- c(0.452, 1.474, 0.22, 0.545, 1.205, 3.55)
cat("Input numbers are", numbers, "₩n")
numbers_mean <- sum(numbers)/length(numbers)
out <- paste("The average is ", numbers_mean, ".₩n", sep="")
cat(out)
```

상황에 따라 다르긴 하지만 보통 위 스크립트를 실행할 때 R 파일을 하나 만들고 source()라는 함수를 사용해서 파일 전체를 한번에 읽어들이고 실행을 시킵니다. 위 코드를 myscript.R 이라는 새로운 R 파일을 하나 만들고 저장 후 다음과 같이 실행할 수 있습니다. 참고로 위 파일은 현재 Working directory와 같은 위치에 저장해야 합니다.

```
source("myscript.R")
```

그러나 위와 같은 식으로 실행할 경우 다음 몇 가지 문제가 있습니다. 하나는 입력 값이 바뀔 때마나 파일을 열어 바뀐 값을 저장해 줄 필요가 있습니다. 결과 값에 대해서 다른 처리를 하고 싶을 경우 또한 파일을 직접 수정해 주어야 합니다. 또한 모든 변수들이 전역변수로 사용되어 코드가 복잡해질 경우 변수간 간섭이 생길 가능성이 높습니다.

4.5.2 Build a function

함수는 특정 데이터를 입력으로 받아 원하는 기능을 수행한 후 결과 데이터를 반환하는 구조를 가집니다. 함수는 일반적으로 다음과 같은 포멧으로 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)
}
```

만들어진 함수는 다음과 같이 사용할 수 있습니다. 만들어진 함수는 처음에 한 번 실행해 주어 실행중인 R session에 등록한 후 사용할 수 있습니다. 여기서 함수로 전달되는 값 pi는 argument (전달인자) 라고 합니다. 전달인자는 함수에서 정의된 매개변수의 갯수와 같은 수의 전달인자를 입력해 주어야 합니다. 참고로 parameter와 argument는 많은 사람들이 혼동하는 단어입니다. 본 예에서 my_sine함수의 괄호 안에 있는 변수 x는 parameter이고 x에 들어가는 값인 pi 나 90은 argument입니다.

```
my_sine(pi)
my_sine(90)
sin(90)
```

- Terminology
  - function name: my_sine
  - parameter: x
  - argument: pi
  - return value: y

이제 위 스크립트 (myscript.R) 에서 사용된 코드를 함수로 바꿔봅니다. numbers (전달인자)를 받는 매개변수를 x로 하고 함수 이름은 mymean 이고 평균값 (numbers_mean)을 반환하는 합수입니다.

```
numbers <- c(0.452, 1.474, 0.22, 0.545, 1.205, 3.55)

mymean <- function(x) {
 cat("Input numbers are", x, "₩n")
 numbers_mean <- sum(x)/length(x)
 out <- paste("The average is ", numbers_mean, ".₩n", sep="")
 cat(out)
 return(numbers_mean)
}

retval <- mymean(numbers)
cat(retval)
```

myscript.R이라는 파일을 열고 작성된 스크립트에 더해서 아래처럼 함수 코드를 만들 경우 source() 함수로 함수를 세션으로 읽어오고 바로 사용할 수 있습니다. 위와 같이 함수를 만들 경우 입력 값을 언제든 바꿔서 사용할 수 있고 반환값에 대한 추가적인 연산도 쉽게 수행 할 수 있습니다.

```
new_values <- c(1:10)
retval <- mymean(new_values)
retval
```

Exercises

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

Exercises

1) mysd라는 이름의 (표본) 표준편차를 구하는 함수를 myscript.R 파일에 구현하시오 (sd() 함수 사용하지 않고, 다음 표준편차 공식 이용)

$$\sigma = \sqrt{\frac{\sum(x - mean(x))^2}{length(x) - 1}}$$

코드는 아래와 같음

```{r}

mysd <- function(x) {
 numbers_sd <- sqrt(sum((x - mymean(x))^2)/(length(x)-1))
 return(numbers_sd)
}
```

2) 1부터 100까지의 값을 x에 저장하고 mysd 함수를 사용해서 표준편차를 구하시오

3) 앞서 작성한 mymean 함수와 mysd 함수를 같이 사용하여 x를 표준화 하고 z로 저장하시오. 표준화 공식은 다음과 같음

$$z = \frac{x - mean(x)}{sd(x)}$$

4) x 와 z 변수를 원소로 갖는 y라는 이름의 data.frame을 생성하시오

4.5.3 local and global variables

함수를 사용함에 따라서 함수 안에서 사용되는 변수와 함수 밖에서 사용되는 변수들의 경우를 명확히 이해할 필요가 있습니다. 다음 코드를 보면 전역변수 x, y는 지역변수 x, y와 독립적으로 사용되고 있습니다.

```
my_half <- function(x) {
 y <- x/z
 cat("local variable x:", x, "₩n")
 cat("local variable y:", y, "₩n")
 cat("global variable z:", z, "₩n")
 return(y)
}
y <- 100
x <- 20
z <- 30
cat("Global variable x:", x, "₩n")
cat("Global variable y:", y, "₩n")
cat("Global variable z:", z, "₩n")
```

```
my_half(5)

my_half <- function(x, z) {
 y <- x/z
 cat("local variable x:", x, "₩n")
 cat("local variable y:", y, "₩n")
 cat("local variable z:", z, "₩n")
 return(y)
}

my_half(5, 10)
```

log, sin등의 함수들은 Built-in function으로 같은 이름의 함수를 만들지 않도록 주의합니다.

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

4.5.4 Vectorized functions

초기에 R이 다른 프로그래밍 언어에 비해서 경쟁력을 갖는 이유 중 하나가 바로 이 벡터 연산 기능 이였습니다. vector 변수에 들어있는 각 원소들에 대해서 특정 함수나 연산을 적용하고 싶을 경우 전통 방식의 C나 Java등의 언어에서는 원소의 개수만큼 반복문을 돌면서 원하는 작업을 수행 했습니다. 그러나 R의 벡터 연산 기능은 별도의 반복문 없이 vector 안에 있는 원소들에 대한 함수 실행 또는 연산을 수행할 수 있습니다.

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

length(x)
test_scores <- c(Alice = 87, Bob = 72, James= 99)
names(test_scores)
```

Exercises

다음은 한 다이어트 프로그램의 수행 전 후의 다섯 명의 몸무게이다.

Before	78	72	78	79	105
after	67	65	79	70	93

1) 각각을 before 와 after 이름의 변수에 저장 후 몸무게 값의 변화량을 계산하여 diff 라는 변수에 저장하시오
2) diff에 저장된 값들의 합, 평균, 표준편차를 구하시오

Exercises

1) 다음 네 학생이 있으며 “John”, “James”, “Sara”, “Lilly” 각 나이는 21, 55, 23, 53 이다. ages 라는 변수를 생성하고 각 나이를 저장한 후 who라는 이름의 함수를 만들어서 50살 이상인 사람의 이름을 출력하는 함수를 만드시오.

- ages라는 변수에 나이 저장, c() 함수 이용, vector 형태 저장
- names() 함수 이용해서 각 ages 벡터의 각 요소에 이름 붙이기
- which() 함수 사용해서 나이가 50보다 큰 인덱스 찾고 해당 인덱스 값들을 idx에 저장
- ages에서 idx에 해당하는 인덱스를 갖는 값을 sel_ages에 저장
- names() 함수를 이용해서 sel_ages의 이름을 sel_names에 저장
- 위 설명을 참고해서 input이라는 파라메터를 갖고 sel_names라는 50살 이상인 사람의 이름을 반환하는 who50 이라는 이름의 함수 만들기
- who50 함수의 사용법은 who50(ages) 임

4.6 Flow control

4.6.1 if statements

R에서의 제어문의 사용은 다른 프로그래밍 언어와 거의 유사합니다. 먼저 if 는 다음과 같은 형식으로 사용되며 () 안에 특정 조건 판단을 위한 표현이 들어갑니다.

```
if(condition) {
  expr_1
}else{
  expr_2
}
```

특히 condition은 하나의 원소에 대한 조건 판단문으로 T 또는 F 값 하나만을 반환하는 문장이어야 합니다. 위 코드는 만약 condition 조건이 True 이면 expr_1를 실행하고 False이면 expr_2를 실행하라는 명령입니다. condition 안에서 사용되는 비교 연산자들은 다음과 같습니다.

! x	logical negation, NOT x
x & y	elementwise logical AND
x && y	vector logical AND
x | y	elementwise logical OR
x || y	vector logical OR
xor(x, y)	elementwise exclusive OR
<	Less than, binary
>	Greater than, binary
==	Equal to, binary
>=	Greater than or equal to, binary
<=	Less than or equal to, binary

```
x <- 2
if(x%%2 == 1) {
  cat("Odd")
}else{
  cat("Even")
}

x <- 5
if(x > 0 & x < 4) {
  print("Positive number less than four")
}

if(x > 0) print("Positive number")

x <- -5
if(x > 0) {
  print("Non-negative number")
} else if(x <= 0 & x > -5) {
  print("Negative number greater than -5")
} else {
  print("Negative number less than -5")
}

if(x > 0)
```

```
 print("Non-negative number")
else
 print("Negative number")
```

4.6.2 ifelse statements

if는 하나의 조건만 비교하는데 사용할 수 있습니다. 그러나 변수에는 여러 값이 벡터형식으로 들어가고 벡터연산을 수행할 경우의 결과도 벡터형식으로 나오지만 if문은 이들을 한 번에 처리하기 어렵습니다. ifelse는 이러한 단점을 보완하여 여러 값을 한번에 처리할 수 있습니다.

```
ifelse (condition, True일 때 리턴값, False일 때 리턴값)
```

ifelse의 경우 빠르게 원하는 값을 반환할 수 있으나 조건별로 다른 추가적인 명령의 수행은 불가능하다는 단점이 있습니다.

```
x <- c(1:10)
if(x>10){
 cat("Big")
}else{
 cat("Small")
}

ifelse(x>10, "Big", "Small")
```

Exercises

1) 다음은 median (중간값)을 구하는 공식이며 x의 길이가 (n이) 홀수일 경우와 짝수일 경우에 따라서 다른 공식이 사용된다. 다음 공식과 코드를 이용하여 mymedian 이라는 이름의 함수를 만들고 입력 값들의 중간값을 구해서 반환하는 함수를 만드시오. (%% 나머지 연산, if문 사용, 아래 중간값 코드 참고)

$$median(X) = \begin{cases} \frac{1}{2}X[\frac{n}{2}] + \frac{1}{2}X[1+\frac{n}{2}] & \text{if } n \text{ is even} \\ X[\frac{n+1}{2}] & \text{if } n \text{ is odd} \end{cases}$$

```
sorted_x <- sort(x)
# 만약 짝수이면
retval <- sort_x[n/2]/2 + sort_x[1+(n/2)]/2
# 만약 홀수이면
retval <- sort_x[(n+1)/2]
```

4.6.3 for, while, repeat

for 문은 반복적으로 특정 코드를 실행하고자 할 때 사용됩니다. 다음과 같은 형식으로 사용할 수 있습니다.

```
for(var in seq){
 expression
}
```

var는 반복을 돌 때마다 바뀌는 변수로 {} 안에서 사용되는 지역 변수 입니다. seq는 vector 형식의 변수로 반복을 돌 때마다 순차적으로 var에 저장되는 값들 입니다.

```
x <- 1:10
for(i in x){
 cat(i, "₩n")
 flush.console()
```

```
}

sum_of_i <- 0
for(i in 1:10){
 sum_of_i <- sum_of_i + i
 cat(i, " ", sum_of_i, "₩n");flush.console()
}
```

while문도 for문과 같이 반복적으로 특정 코드를 수행하고자 할 때 사용합니다. 사용하는 문법은 다음과 같으며 cond는 True 또는 False 로 반환되는 조건문을 넣고 True 일 경우 계속해서 반복하면서 expressions를 수행하며 이 반복은 cond가 False로 될 때 까지 계속됩니다.

```
while(cond){
 expression
}
```

while문을 사용할 경우 다음과 같이 indicator라 불리우는 변수를 하나 정해서 반복 할 때마다 값이 바뀌도록 해 주어야 합니다. 그렇지 않으면 무한 루프를 돌게 되는 문제가 발생합니다.

```
i <- 10
f <- 1
while(i>1){
 f <- i*f
 i <- i-1
 cat(i, f, "₩n")
}
f
factorial(10)
```

repeat 명령은 조건 없이 블럭 안에 있는 코드를 무조건 반복하라는 명령 입니다. 따라서 블럭 중간에 멈추기 위한 코드가 필요하고 이 명령이 break 입니다.

```
repeat{
 expressions
 if(cond) break
}

i <- 10
f <- 1
repeat {
 f <- i*f
 i <- i-1
 cat(i, f, "₩n")
 if(i<1) break
}
f
factorial(10)
```

4.6.4 Avoiding Loops

R에서는 가능하면 loop문을 사용하지 않는 것이 좋습니다. 이는 다른 언어들 보다 반복문이 느리게 수행된다는 이유 때문이기도 합니다. 그러나 R에서는 반복문을 수행하는 것 보다 훨씬 더 빠르게 반복문을 수행 한 것과 같은 결과를 얻을 수 있는 다양한 방법들이 제공되고 있습니다. 차차 그런 기법들에 대한 학습을 진행하도록 하겠습니다.

```
x <- 1:1E7
sum(x)
system.time(sum(x))
```

```
st <- proc.time()
total <- 0
for(i in 1:length(x)) {
  total <- total + x[i]
}
ed <- proc.time()
ed-st
```

Exercises

1) 다음 네 사람의 이름과 나이를 데이터로 갖는 users 변수를 (data.frame) 만드시오

```
user_score <- c(90, 95, 88, 70)
user_names <- c("John","James","Sara", "Lilly")
```

2) 각 사람의 점수가 80보다 작으면 이름 점수: Fail 크면 이름 점수: Pass를 출력을 하는 코드를 작성하시오. 예를 들어 John의 점수는 80보다 크므로 John 90: Pass 출력 (for, print 함수 이용)

4.6.5 dplyr::if_else

4.6.6 dplyr::case_when

4.7 Object Oriented Programming (Advanced)

OOP는 객체지향 프로그래밍 이라고 합니다. OOP를 이용해서 프로그래밍으로 풀고자 하는 문제를 좀 더 명확하게 개념을 수립하고 복잡한 코드를 명료하게 만들 수 있습니다. 그런데 R에서 OOP는 다른 언어보다는 좀 더 어려운 개념적인 이해가 필요합니다. S3, S4, 그리고 Reference class 가 있으며 S3, S4는 Generic function을 이용하며 다른 언어에서 사용하는 OOP 개념과는 다릅니다. Reference class는 다른 언어에서 사용하는 OOP 개념과 유사하며 R6 패키지를 이용해서 사용할 수 있습니다.

Chapter 5

Data transform classic

5.1 Introduction

일반적인 데이터 분석은 데이터 전처리(변환), 가시화, 모델링(통계분석)의 반복적인 수행으로 진행될 수 있습니다. R에서는 data.frame 형식의 데이터 타입이 주로 사용되며 (최근 tibble형식) 따라서 data.frame 기반의 데이터를 다루기 위한 다양한 함수를 익힐 필요가 있습니다. 이번 강의에서는 data.frame 데이터를 읽거나 쓰는 함수들과 함께 데이터 전처리를 (변환) 위한 함수들을 배워보겠습니다.

앞에서 배웠던 데이터를 저장하는 object의 종류를 먼저 간략히 정리해 봅니다.

- Vectors – 같은 타입의 데이터를 (Numeric, character, factor, …) 저장한 오브젝트로 인덱스는 [,] 사용.
- Lists – 여러개의 vector를 원소로 가질 수 있으며 각 원소 vector들은 문자나 숫자 어떤 데이터 타입도 가능하고 길이가 달라도 됨. list의 인덱싱에서 [ ]는 리스트를 반환하고 [[ ]]는 vector를 반환함.
- Matrices – 같은 타입의 데이터로 채워진 2차원 행렬이며 인덱스는 [i, j] 형태로 i는 row, j는 column 을 나타냄. 메트릭스의 생성은 matrix 명령어를 사용하며 왼쪽부터 column 값을 모두 채우고 다음 컬럼 값을 채워 나가는 것이 기본 설정임. byrow=T 를 통해 row를 먼저 채울수도 있음. row와 column 이름은 rownames와 colnames로 설정이 가능하며 rbind와 cbind로 두 행렬 또는 행렬과 백터를 연결할 수 있음 (rbind와 cbind의 경우 행렬이 커지면 컴퓨터 리소스 많이 사용함)
- data.frame – list와 matrix의 특성을 모두 갖는 오브젝트 타입으로 list와 같이 다른 타입의 vector형 변수 여러개가 컬럼에 붙어서 matrix 형태로 구성됨. 단, list와는 다르게 각 변수의 길이가 (row의 길이) 같아야 함. $ 기호로 각 변수들을 인덱싱(접근) 할 수 있고 matrix와 같이 [i,j] 형태의 인덱싱도 가능.

5.2 Reading and writing

파일에 있는 데이터를 R로 읽어들이거나 쓰는 일은 일반적인 데이터 분석 과정에서 필수적일 수 있습니다. 본 강의에서는 일반적으로 사용하는 텍스트 파일과 엑셀파일을 활용하는 방법을 알아보겠습니다.

5.2.1 Text file

편의상 데이터를 쓰는 과정을 먼저 살펴봅니다. ggplot2 예제에 있는 데이터 중 msleep 데이터는 이 데이터셋은 다양한 동물들의 수면 패턴에 대한 정보를 담고 있습니다. str 또는 dplyr::glimpse 함수로 데이터 전체적인 구조를 파악한 일부 데이터만을 이용해 추가적인 데이터를 생성한 후 별도로 파일에 저장해 보겠습니다.

```
library(tidyverse)
data(msleep)
str(msleep)
glimpse(msleep)

mydf <- data.frame(brainwt = msleep$brainwt,
        sleep_total = msleep$sleep_total)
class(mydf)
```

```
mydf <- msleep[,c("brainwt", "sleep_total")]
class(mydf)

mydf <- msleep |> dplyr::select(brainwt, sleep_total)
class(mydf)
```

마지막 tidyverse 스타일의 mydf 데이터 생성이 가장 추천하는 방법입니다. 파일에 쓰는 방법은 다음과 같습니다. 패키지에 따라서 다양한 파일 쓰기 함수들이 제공되고 있지만 위 두 파일은 utils라는 R의 기본 패키지에 들어있는 함수들로서 가장 많이 사용되는 함수들 입니다. ?write.table 등으로 도움말을 보시고 특히 함수의 전달값 (Arguments) 들을 (quote, row.names, col.names, sep) 익혀두시기 바랍니다.

```
write.table(mydf, file="table_write1.txt")
write.table(mydf, file="table_write2.txt", quote=F)
write.table(mydf, file="table_write3.txt", quote=F, row.names=F)
write.table(mydf, file="table_write4.txt", quote=F, row.names=F, sep="₩t")
write.table(mydf, file="table_write5.csv", quote=F, row.names=F, sep=",")
```

대부분의 텍스트 파일은 아래와 같이 csv 또는 txt 파일로 저장하여 메모장으로 열어 확인할 수 있으며 읽어올 때 구분자 (sep 파라메터) 나 header를 (header 파라메터) 읽을지 등을 옵션으로 지정할 수 있습니다.

```
dat <- read.csv("table_write5.csv")
head(dat)
str(dat)
glimpse(dat)
```

table_write5.csv 파일을 열어보면 다음과 같이 header와 “,”로 구분되어 있는 것을 볼 수 있습니다. read.csv 함수의 도움말을 보면 이 함수의 파라메터 head와 sep이 기본값으로 T와 ,로 되어 있는 것을 볼 수 있습니다. read.csv 외에도 read.table, read.delim 등의 함수를 이용해서 택스트 파일을 읽어올 수 있습니다.

추가로 수면 시간과 뇌 무게의 관계를 보기위해 다음과 같이 데이터를 가시화 할 수 있습니다.

```
plot(y=mydf$brainwt, x=mydf$sleep_total)
plot(y=log(mydf$brainwt), x=mydf$sleep_total)
```

NA를 제거하기 위해서 na.omit 함수를 사용합니다.

```
mydf2 <- na.omit(mydf)
mycor <- cor(log(mydf2$brainwt), mydf2$sleep_total)
fit <- lm(log(mydf2$brainwt) ~ mydf2$sleep_total)
summary(fit)
plot(y=log(mydf2$brainwt), x=mydf2$sleep_total); abline(fit); text(50, 170, round(mycor,2))
```

5.2.2 Excel file

텍스트 파일 외에 엑셀파일은 readxl 이라는 R 패키지를 활용하여 읽거나 쓸 수 있습니다. 패키지는 다음과 같은 방법으로 설치할 수 있으며 read_excel 이라는 함수를 사용해서 데이터를 읽어들일 수 있습니다. readxl은 tidyverse 패키지들 중 하나입니다.

```
# install.packages("readxl")
library(readxl)
```

실습 파일은 형광 세포를 배양하여 형광리더기를 이용해 얻어진 실제 데이터이며 plate_reader.xls 에서 다운로드 받을 수 있습니다. read_excel 함수를 이용하여 파일의 내용을 읽어오면 기본 자료형이 tibble 입니다. tibble은 최근 많이 쓰이는 R object로 data.frame과 유사하나 입력값의 type, name, rowname을 임으로 바꿀 수 없다는 점이 다릅니다.

```
dat <- read_excel("examples/plate_reader.xls", sheet=1, skip = 0, col_names=T)
```

엑셀파일에는 두 종류의 ($OD600_{nm}$, Fluorescence) 데이터가 저장되어 있습니다. 첫 번째 sheet에는 다음처럼 데이터가 저장되어 있습니다.

	A	B	C	D	E	F	G	H
1	Plate	Repeat	Well	Type	Time	595nm_kl	Time	EGFP_suli
2	1	1	B01	M	00:00:15.93	0.701	00:01:11.48	67809
3	1	1	B02	M	00:00:16.32	0.752	00:01:11.89	60025
4	1	1	B03	M	00:00:16.69	0.723	00:01:12.30	102745
5	1	1	B04	M	00:00:17.06	0.744	00:01:12.71	99979
6	1	1	B05	M	00:00:17.43	0.706	00:01:13.12	108175
7	1	1	B06	M	00:00:17.80	0.723	00:01:13.53	109575
8	1	1	B07	M	00:00:18.17	0.767	00:01:13.94	76531
9	1	1	B08	M	00:00:18.54	0.777	00:01:14.35	72137
10	1	1	B09	M	00:00:18.91	0.762	00:01:14.76	154549
11	1	1	B10	M	00:00:19.28	0.798	00:01:15.17	128498
12	1	1	B11	M	00:00:19.65	0.793	00:01:15.58	151693
13	1	1	B12	M	00:00:20.02	0.821	00:01:15.99	130526
14	1	1	C01	M	00:00:22.85	0.803	00:01:18.86	42654

프로토콜 상세 내역이 나온 세 번째 시트를 읽을 경우 sheet 옵션을 3로 설정하면 되며 skip=3으로 하고 컬럼 이름을 별도로 사용하지 않으므로 col_names=T로하여 읽을 수 있습니다.

```
dat <- read_excel("examples/plate_reader.xls", sheet=3, skip = 3, col_names=F)
```

참고로 엑셀파일로 저장하기 위해서는 tidyverse의 writexl 패키지를 사용하거나 csv 파일로 데이터를 writing 한 뒤 Excel로 해당 csv 파일을 열고 xlsx 파일로 저장할 수 있습니다.

```
library(writexl)
dat <- read_excel("examples/plate_reader.xls", sheet=1)
write_xlsx(dat, path = "examples/plate_reader.xlsx")
```

5.3 Classical way of the data transformation

아래 설명하는 subset, filter, merge, split, select 등의 함수는 임의의 데이터를 효과적으로 변환하는데 사용되는 기본 함수들입니다. 그러나 위 함수들을 개별적으로 사용하는 것 보다 tidyverse 방식의 데이터 변환 함수들이 더 많이 사용되고 있습니다. 아래 설명은 필요할 경우 참고하시면 되겠습니다.

5.4 Subset

R에서 데이터 저장은 data.frame이나 matrix 타입을 일반적으로 사용합니다. 이 데이터의 일부 열 또는 행의 데이터만을 가져와서 별도로 저장하거나 분석이 필요할 경우가 있습니다. 이 때 인덱싱을 사용해서 일부 데이터를 선택하고 사용할 수 있으며 subset 함수도 이러한 선별 기능을 제공합니다. subset은 행과 열 모두를 선별할 수 있는 함수입니다. 다음 airquality 데이터는 1973년 날짜별로 뉴욕의 공기질을 측정한 데이터 입니다. NA를 제외한 나머지 데이터만으로 새로운 데이터셋을 만들어 봅시다. is.na함수를 사용하면 해당 데이터가 NA일 경우 TRUE, NA가 아닐 경우 FALSE 를 반환해 줍니다.

```
glimpse(airquality)

is.na(airquality$Ozone)
ozone_complete1 <- airquality[!is.na(airquality$Ozone),]
glimpse(ozone_complete1)
ozone_complete2 <- filter(airquality, !is.na(Ozone))
glimpse(ozone_complete2)
```

위 ozone_complete1와 ozone_complete2는 같은 결과를 보입니다. 그러나 ozone_complete1 보다는 ozone_complete2 코드가 더 직관적이고 가독성이 높습니다. 특히 airquality$ozone 로 $를 사용하여 변수에 접근한 것이 아닌 Ozone이라는 변수 이름을 직접 사용해서 접근함으로써 코드의 간결성과 가독성을 유지할 수 있습니다. 또한 subset의 select 옵션을

이용해서 변수를 선택할 수도 있으며 &(AND)와 |(OR) 연산자를 사용해서 조건을 두 개 이상 설정할 수 있습니다. 아래 select 옵션에서 -는 해당 변수를 제외한다는 의미 입니다.

```
ozone_complete3 <- subset(airquality, !is.na(ozone), select=c(ozone, temp, month, day))
ozone_complete4 <- subset(airquality, !is.na(ozone) & !is.na(solar.r), select=c(-month, -day))
```

그러나 위 코드에서 순차적으로 수행되는 변환 과정 역시 여러 프로세스가 반복될 수록 복잡해지고 가독성이 떨어지는 것을 알 수 있습니다.

Exercises

1) airquality 데이터에서 Temp와 Ozone 변수로 이루어진 df라는 이름의 data.frame을 만드시오 (단 NA가 있는 샘플(열)은 모두 제외하시오)

5.5 Merging and Split

merge 함수는 두 개 이상의 데이터셋을 통합하는 기능을 수행하는 함수입니다. 특히 rbind나 cbind와는 다르게, 결합하는 두 데이터에 공통적이거나 한 쪽의 데이터를 기준으로 결합을 수행 합니다. ?merge를 참고하면 by, by.x, by.y, all, all.x, all.y 등의 옵션으로 이러한 설정을 수행할 수 있습니다. 간단한 예제를 통해서 이해해 보겠습니다.

10명의 사람이 있고 이 사람들의 나이와 성별을 각각 나타낸 두 데이터셋이 있습니다. 그런데 df1은 나이만을 df2는 성별 정보만을 가지고 있으며 두 정보 모두 제공된 사람은 3명 (인덱스 4,5,6) 뿐입니다. 이제 merge를 이용해서 두 데이터셋을 결합해 보겠습니다.

```
## merge
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"))

df_inner <- merge(df1, df2, by="id", all=F)
df_outer <- merge(df1, df2, by="id", all=T)
df_left_outer <- merge(df1, df2, by="id", all.x=T)
df_right_outer <- merge(df1, df2, by="id", all.y=T)
```

만약 두 데이터셋의 id가 다를 경우나 각각 다른 기준으로 결합해야 하는 경우는 by대신 by.x, by.y 옵션을 사용할 수 있습니다.

split 함수는 데이터를 특정 기준으로 나누는 역할을 하며 해당 기준은 factor 형 벡터 형태로 주어질 수 있습니다. 예를 들어 airquality 데이터의 month 변수를 기준으로 데이터를 분리해 보겠습니다.

```
str(airquality)
g <- factor(airquality$Month)
airq_split <- split(airquality, g)
class(airq_split)
str(airq_split)
```

위와 같이 airq_split은 길이가 5인 (5, 6, 7, 8, 9월) list타입이 되었고 각 요소는 서로 다른 size의 data.frame형으로 구성된 것을 확인할 수 있습니다.

5.6 Transformation

R에서 기존 가지고 있는 데이터의 변경은 새로운 변수의 추가, 삭제, 변형과 샘플의 추가, 삭제, 변형을 생각해 볼 수 있습니다. 이러한 기능은 앞에서 배운 merge, split이나 rbind, cbind, 그리고 인덱싱을 활용한 값 변경 등의 방법을 이용할 수 있습니다. 또한 가장 직관적으로 필요한 변수들을 기존 데이터셋에서 추출한 후 data.frame 명령어를 사용해서 새로운 데이터셋으로 만들어주면 될 것 입니다.

이러한 방법들 외에 within을 사용할 경우 특정 변수의 변형과 이를 반영한 새로운 데이터셋을 어렵지 않게 만들수 있습니다. with 함수의 사용 예와 함께 within 함수를 사용하여 데이터를 변형하는 예를 살펴봅니다. with나 within 함수는 R을 활용하는데 많이 사용되는 함수들은 아닙니다. 또한 이러한 기능들은 dplyr 등의 패키지에서 제공하는 경우가 많아서 필수적으로

익힐 부분은 아닙니다. 그러나 개념적인 이해를 돕기위한 좋은 도구들이며 여전히 고수준의 R 사용자들이 코드에 사용하고 있는 함수들이므로 알아두는 것이 좋습니다.

```
## without with
ozone_complete <- airquality[!is.na(airquality$Ozone),"Ozone"]
temp_complete <- airquality[!is.na(airquality$Temp),"Temp"]
print(mean(ozone_complete))
print(mean(temp_complete))

## with
with(airquality, {
 print(mean(Ozone[!is.na(Ozone)]))
 print(mean(Temp[!is.na(Temp)]))
})
```

위 with 함수에서 보는바와 같이 $를 이용한 변수 접근 대신 with함수 내에서는 ({, } 안에서) 해당 data.frame에 있는 변수 이름을 직접 접근할 수 있으며 따라서 코드의 간결함과 가독성이 향상됩니다.

within 함수는 with함수와 같이 {, } 안에서 변수의 이름만으로 해당 변수에 접근이 가능하나 입력된 데이터와 변경된 변수(들)을 반환한다는 점이 다릅니다. 아래 예는 airquality 데이터의 화씨 (Fahrenheit) 온도를 섭씨 (Celsius) 온도로 변환해서 새로운 데이터셋을 만드는 코드입니다. data.frame을 이용한 코드와 비교해 보시기 바랍니다. 데이터셋 내에서 참조할 변수들이 많아질 경우 airquality$xxx 식의 코드를 줄이는 것 만으로도 코드의 가독성과 간결성을 유지할 수 있습니다.

```
newairquality <- within(airquality, {
 celsius = round((5*(Temp-32))/9, 2)
})
head(newairquality)

## data.frame
celsius <- round((5*(airquality$Temp-32))/9, 2)
newairquality <- data.frame(airquality, celsius)
head(newairquality)
```

Exercises

1) 다음 df 의 hour, minute, second로 나누어진 값들을 초 단위로 변환하여 seconds라는 변수에 저장한 후 기존 df에 추가한 df2 데이터셋을 만드시오 (within 함수 이용)

```
df <- data.frame(hour=c(4, 7, 1, 5, 8),
                 minute=c(46, 56, 44, 37, 39),
                 second=c(19, 45, 57, 41, 27))
```

5.7 Babies example

UsingR 패키지의 babies 데이터를 이용해서 산모의 흡연 여부와 신생아 몸무게의 관계를 알아보는 분석을 수행해 보겠습니다. 본 강의를 통해 배우지 않은 내용들이 있지만 코드를 따라 가면서 참고하시기 바랍니다. 우선 UsingR 패키지를 로딩합니다. 산모의 임신 기간이 (gestation) 999로 표기된 데이터는 명백히 에러이며 이들을 NA로 처리합니다.

```
library(UsingR)
head(babies)
## a simple way to checkout the data
plot(babies$gestation)
babies$gestation[babies$gestation>900] <- NA
str(babies)
```

아래와 같이 within 함수를 사용해서 babies$ 를 반복해서 입력해주는 불편함을 줄이고 가독성을 높입니다. 똑같은 방법으로 dwt (아빠의 몸무게) 변수의 에러값들에 대해서도 NA 처리를 할 수 있습니다.

```
new_babies <- within(babies, {
 gestation[gestation==999] <- NA
 dwt[dwt==999] <- NA
})
str(new_babies)
```

smoke 변수는 흡연 여부를 나타내는 범주형 변수로 0, 1, 2, 3 값은 의미가 없습니다. 사람이 읽을 수 있는 label을 붙인 factor 형 변수로 변환하는 코드도 함께 작성해 보겠습니다.

```
str(babies$smoke)
new_babies <- within(babies, {
 gestation[gestation==999] <- NA
 dwt[dwt==999] <- NA
 smoke = factor(smoke)
 levels(smoke) = list(
  "never" = 0,
  "smoke now" = 1,
  "until current pregnancy" = 2,
  "once did, not now" = 3)
 })
str(new_babies$smoke)
```

이제 임신기간과 흡연 여부를 분석해 볼 수 있습니다. 흡연 그룹별로 기간에 차이가 있는지를 알아보는 분석은 t-test나 ANOVA를 사용할 수 있습니다.

```
fit <- lm(gestation~smoke, new_babies)
summary(fit) ## t-test 결과
anova(fit)
```

간단히 결과를 보면 summary(fit)은 3가지 t-test의 결과를 보여줍니다. never vs. smoke new 의 경우 t값이 -1.657로 피우지 않은 경우에 비해서 피우는 사람의 임신 기간이 유의하게 줄어들었음을 알 수 있습니다. 그에 비해서 현재 흡연하지 않는 경우 (never vs. until current pregnancy 또는 never vs. once did, not now) 차이가 없는 것으로 나옵니다.

이제 smoke now 인 경우 또는 나이가 25세 미만인 경우의 샘플에 대해서 newdf를 만들어 봅니다 (subset 함수 사용, id, gestation, age, wt, smoke 변수 선택). 이 후 ggplot을 이용하여 몸무게와 임신기간의 산점도를 그려보면 크게 다르진 않으나 흡연하는 여성 중 몸무게가 적게 나가는 여성에게서 짧은 임신기간을 갖는 경향을 볼 수 있습니다.

```
newdf <- subset(new_babies, (smoke=="smoke now" | smoke == "never") & age < 25, select=c(id, gestation, age, wt
# ggplot(newdf, aes(x=wt, y=gestation, color=smoke)) +
#  geom_point(size=3, alpha=0.5) +
#  facet_grid(.~smoke) +
#  theme_bw()
```

5.8 Useful functions

지금까지 배운 여러 R 프로그래밍 기법이나 함수들과 같이 R을 활용한 데이터 분석에서 자주쓰이거나 유용하게 사용되는 함수들을 소개합니다. 먼저 원소들을 비교하여 공통적 또는 유일한 원소들만을 추출해내는 함수들 입니다.

```
#match(), %in%, intersect()

x <- 1:10
y <- 5:15
match(x, y)
x %in% y
intersect(x, y)

#unique()
```

```
unique(c(x, y))
```

다음은 스트링 관련 함수들로서 서열데이터 분석 등에서 유용하게 활용되는 함수들 입니다.

```
#substr()
x <- "Factors, raw vectors, and lists, are converted"
substr(x, 1, 6)

#grep()
grep("raw", x)

#grepl()
grepl("raw", x)
if(grepl("raw", x)){
 cat("I found raw!")
}

x <- paste(LETTERS, 1:100, sep="")
grep("A", x)
x[grep("A", x)]

grepl("A", x)
r <- grepl("A", x)
if(r){
 cat("Yes, I found A")
}else{
 cat("No A")
}

#strsplit()
x <- c("Factors, raw vectors, and lists, are converted", "vectors, or for, strings with")
y <- strsplit(x, split=", ")

#unlist()
unlist(y)

y <- strsplit(x, split="")
ychar <- unlist(y)
ycount <- table(y2)
ycount_sort <- sort(ycount)
ycount_sort <- sort(ycount, decreasing = T)
ycount_top <- ycount_sort[1:5]
ycount_top_char <- names(ycount_top)

#toupper(), tolower()
toupper(ycount_top_char)
```

Exercises

built-in 데이터셋 중 state.abb 은 미국의 50개 주에대한 축약어임.

1) 이 중 문자 A 가 들어가는 주를 뽑아 x에 저장 하시오 (grep 또는 grepl 사용)

2) state.abb 중 위 x에 저장된 이름들을 빼고 y에 저장 하시오 (match() 또는 %in%사용)

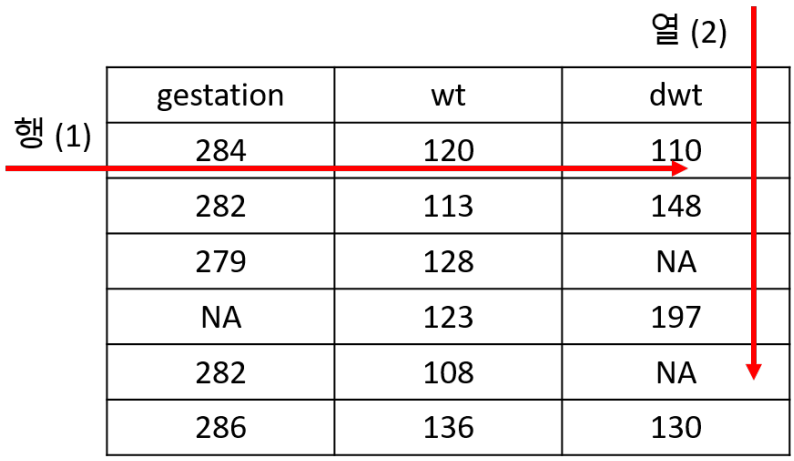
3) state.abb에 사용된 알파벳의 갯수를 구하고 가장 많이 쓰인 알파벳을 구하시오 (strsplit(), table() 등 사용)

5.9 apply

apply는 데이터를 변형하기 위한 함수라기 보다는 데이터를 다룰 때 각 원소별, 그룹별, row, 또는 column 별로 반복적으로 수행되는 작업을 효율적으로 수행할 수 있도록 해주는 함수입니다. apply 계열의 함수를 적절히 사용하면 효율성이나 편리성 뿐만 아니라 코드의 간결성 등 많은 장점이 있습니다. 쉬운 이해를 위해 colMean 함수를 예로 들면 colMean은 column 또는 row 단위로 해당하는 모든 값들에 대해 평균을 계산해주는 함수이고 apply를 사용할 경우 다음과 같이 apply 함수와 mean 함수를 이용해서 같은 기능을 수행할 수 있습니다. 아래는 babies 데이터의 clearning 된 (위에서 만들었던) new_babies 데이터에 이어서 수행되는 내용입니다.

```
library(UsingR)
head(babies)
df <- subset(babies, select=c(gestation, wt, dwt))
colMeans(df, na.rm=T)
apply(df, 2, mean, na.rm=T)
```

위와 같이 colMeans와 apply가 똑같은 결과를 보여주고 있습니다. 두 번째 인자인 margin의 값으로 (?apply참고) 여기서는 2가 사용되었으며 margin 값이 1인지 2인지에 따라서 다음과 같이 작동을 합니다.



mean외에도 다양한 함수들이 사용될 수 있으며 아래와 같이 임의의 함수를 만들어서 사용할 수 도 있습니다. 아래 코드에서는 function(x)...로 바로 함수의 정의를 넣어서 사용했으나 그 아래 mysd 함수와 같이 미리 함수 하나를 만들고 난 후 함수 이름을 이용해서 apply를 적용할 수 있습니다.

```
apply(df, 2, sd, na.rm=T)
apply(df, 2, function(x) {
 xmean <- mean(x, na.rm=T)
 return(xmean)
 })
```

apply 함수는 특히 R에서 느리게 작동하는 loop (for, while 등) 문 대신 사용되어 큰 행렬에 대해서도 빠른 계산 속도를 보여줄 수 있습니다.

```
n <- 40
m <- matrix(sample(1:100, n, replace=T), ncol=4)
mysd <- function(x) {
 xmean <- sum(x)/length(x)
 tmpdif <- x-xmean
 xvar <- sum(tmpdif^2)/(length(x)-1)
 xsd <- sqrt(xvar)
 return(xsd)
}

## for
results <- rep(0, nrow(m))
for(i in 1:nrow(m)) {
```

```
  results[i] <- mysd(m[i,])
}
print(results)
apply(m, 1, mysd)
apply(m, 1, sd)
```

apply 함수 외에도 sapply, lapply, mapply 등의 다양한 apply계열 함수가 쓰일 수 있습니다. 먼저 lapply는 matrix 형태 데이터가 아닌 list 데이터에 사용되어 각 list 원소별로 주어진 기능을 반복해서 수행하며 sapply는 lapply와 유사하나 벡터, 리스트, 데이터프레임 등에 함수를 적용할 수 있고 그 결과를 벡터 또는 행렬로 반환합니다.

```
x <- list(a=1:10, b=exp(-3:3), logic=c(T,T,F,T))
mean(x$a)
lapply(x, mean)
sapply(x, mean)

x <- data.frame(a=1:10, b=exp(-4:5))
sapply(x, mean)

x <- c(4, 9, 16)
sapply(x, sqrt)
sqrt(x)

y <- c(1:10)
sapply(y, function(x) {2*x})
y*2
```

마지막 예제에서처럼 sapply나 lapply도 임의의 함수를 만들어 적용시킬 수도 있습니다. 자세히 살펴 보면 y는 10개의 값을 갖는 벡터이고 이 벡터의 각 원소 (값에) 함수를 반복해서 적용하는 것 입니다. 함수에서 x는 각 원소의 값을 차례차례 받는 역할을 하므로 1부터 10까지 값이 함수로 들어가 2를 곱한 수가 반환됩니다. 따라서 벡터연산을 하는 y*2와 결과가 같으나 원하는 함수를 정의해서 자유롭게 사용할 수 있다는 장점이 있습니다. 리스트의 경우는 다음과 같이 사용합니다.

```
y <- list(a=1:10, b=exp(-3:3), logic=c(T,T,F,T))
myfunc <- function(x) {
  return(mean(x, na.rm=T))
}
lapply(y, myfunc)
unlist(lapply(y, myfunc))
```

즉, myfunc의 x가 list y의 각 원소들, y[[1]], y[[2]], y[[3]]를 각각 받아서 mean 연산을 수행해 줍니다. 결과로 각 list 원소들의 평균 값이 반환되며 unlist 함수는 list 형태의 반환 값을 vector 형태로 전환해 줍니다.

5.10 purrr

TODO

Exercises

다음은 앞에서 수행했던 airquality 데이터를 월별로 나눈 데이터셋임. 이 데이터셋을 이용하여 각 월별로 온도와 오존 농도의 평균값을 저장한 data.frame 형식의 데이터를 만들기 위하여 다음 단계별 과정에 적절한 코드를 작성하시오

```
## dataset
g <- factor(airquality$month)
airq_split <- split(airquality, g)
```

1) 다음 df의 ozone 평균을 구하는 ozone_func 함수를 작성하시오 (단 입력은 data.frame 형식의 오브젝트를 받고 출력은 평균값 (정수 값 하나) 출력. mean 함수 사용시 데이터에 NA가 포함되어 있을 경우 na.rm=T 옵션 적용)

```
## May data.frame
df <- airq_split[[1]]
#
# write your code here for ozone_func function
#

## Usage
ozone_func(df)
## output
# 23.61538
```

2) lapply와 ozone_func 함수를 사용하여 airq_split list 데이터의 월별 ozone 평균 값을 구하고 ozone_means에 vector 형식으로 저장하시오
3) 위 1), 2)와 같은 방법으로 temp_func 함수를 만들고 월별 temp의 평균값을 temp_means에 vector 형식으로 저장하시오.
4) 위에서 구해진 두 변수값들을 이용하여 air_means 라는 이름의 data.frame으로 저장하시오

Exercises

1) 다음 코드를 이용해서 파일을 다운로드 하고 myexp에 저장하고 데이터의 구조 및 샘플들의 이름을 확인하시오
   myexp <- read.csv("https://github.com/greendaygh/kribbr2022/raw/main/examples/gse93819_expression_valu
   header=T)

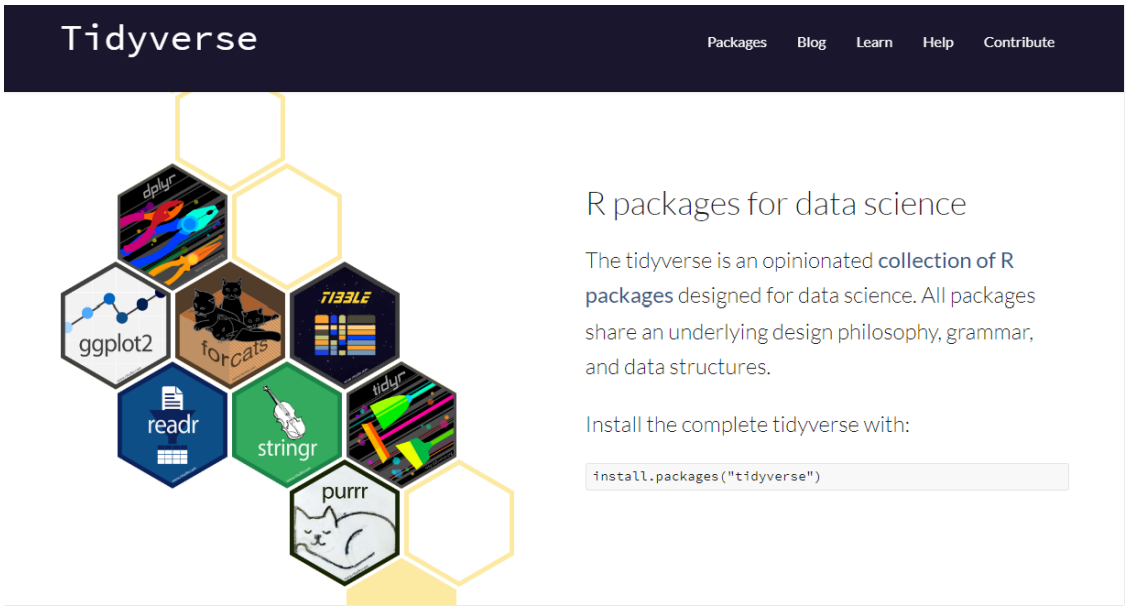
2. myexp의 1부터 10번째 샘플(컬럼) 데이터를 myexp1으로 11부터 20번째 샘플 데이터를 myexp2로 나누시오
3. myexp1의 row별 평균을 구해서 myexp1mean에 myexp2의 row별 평균을 구해서 myexp2mean에 저장하시오 (apply 이용)
4. myexp1mean과 myexp2mean을 합하여 myexpmean이라는 data.frame을 만드시오 (cbind이용, 주의필요)
5. plot을 이용하여 두 평균들의 산포도를 그리시오
6. myexpmean의 두 변수에 대한 차이를 구하여 mydiff 라는 변수에 저장하시오
7. mydiff의 값들에 대한 히스토그램 (막대그래프)을 그리시오

Chapter 6

Data transform tidyverse

6.1 Introduction

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.



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

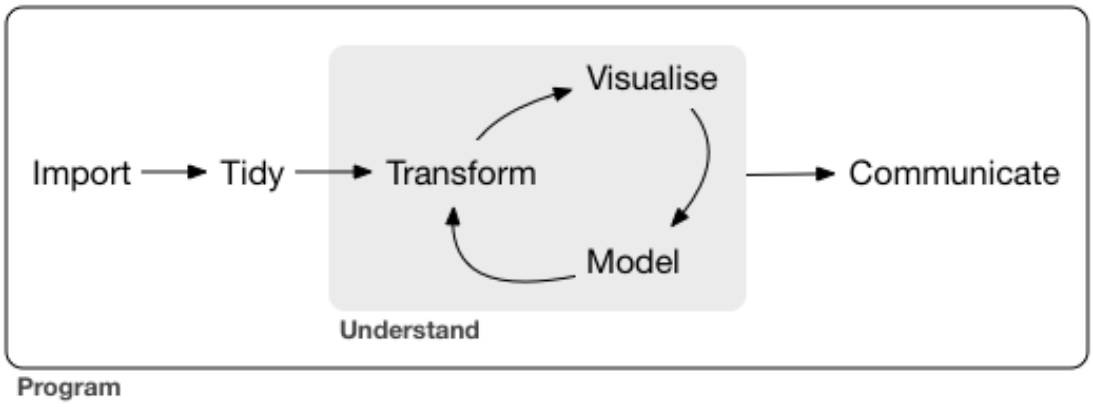


Figure 6.1: from https://r4ds.had.co.nz/

6.2 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 기본으로 사용되는 것을 참고하시기 바랍니다.

```
library(tidyverse)

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

iris
as_tibble(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)
```

tibble은 컬럼 하나가 벡터형 변수가 아닌 리스트형 변수가 될 수 있다는 것도 data.frame과 다른 점 입니다.

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

또한 다음과 같이 사용되는 변수의 (x) 참조 범위가 다릅니다.

```
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)
class(df1[, 1:2])
class(df1[, 1])

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

6.3 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

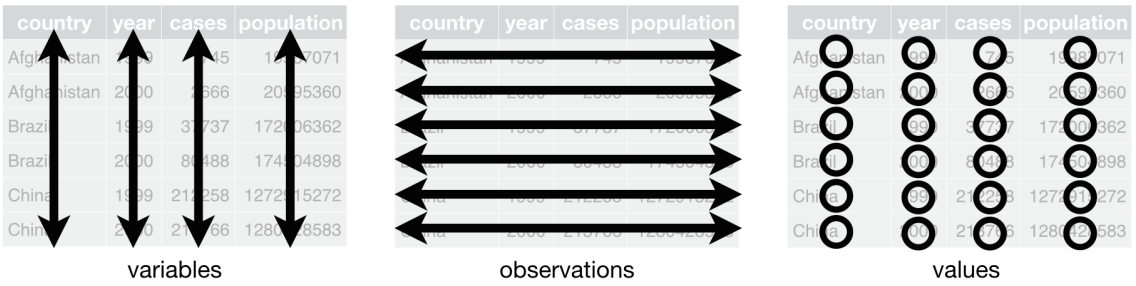


Figure 6.2: from https://r4ds.had.co.nz/

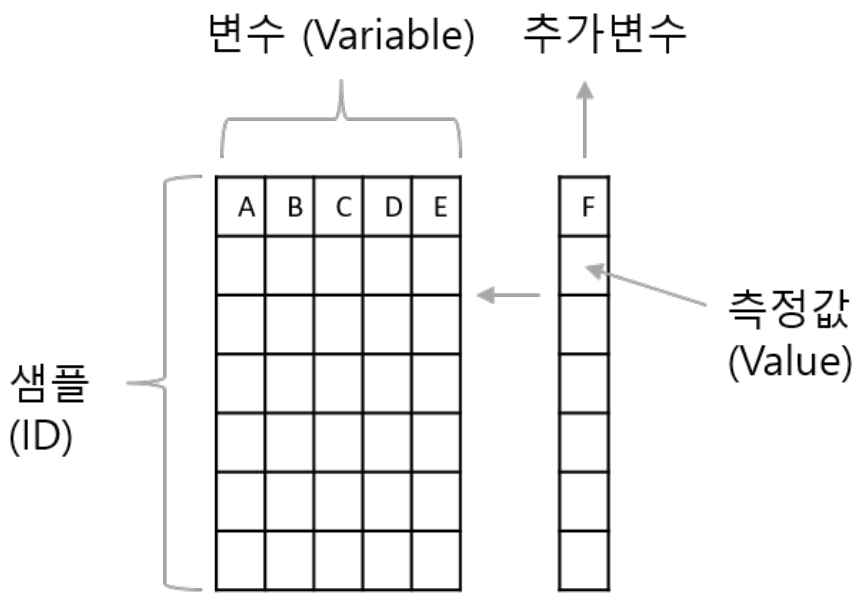
이러한 데이터 구조를 유지하는 것이 필요한 이유는 우선적으로 데이터 분석의 효율성에 있습니다. 또한 데이터를 일관성 있게 저장하고 관리하게 되면 그것을 다루는 분석 도구들을 배우고 활용하기 쉬워집니다. 특히 dplyr, ggplot2 및 tidyverse의 패키지들은 tidy 데이터 형태로 작동합니다. 그리고 변수들이 열에 배치되면서 백터연산을 지원하는 대부분의 R 함수들의 성능이 최대화 됩니다.

```
dat <- read_excel("examples/plate_reader.xls", sheet=1, skip = 0, col_names=T)

head(dat)
glimpse(dat)
```

위 데이터는 전형적인 long형 데이터 입니다. 각 변수는 하나의 컬럼에만 나타나고 각 샘플은 유일한 하나의 row를 가집니다. 만약 플레이터 한 장을 더 측정하면 아래쪽으로 동일한 컬럼에 추가 데이터가 붙게 됩니다. 그러나 임의의 데이터가 long형인지 wide 형인지 판단하는 기준은 목적에 따라서 다를 수 있습니다. 이럴 경우 특정 목저을 가지고 2차원 평면에 plot을 그릴 때 어떤 데이터를 가지고 그림을 그릴지를 고려한다면 쉽게 판단이 가능합니다.

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.



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

6.4 Pipe operator

tidyverse 패키지를 활용하기 위해서는 파이프 오퍼레이터의 이해가 필요합니다. 기존 magrittr에서 제공하던 %〉% 파이프 오퍼레이터와 함께 최근 R 4.1.0부터 도입된 네이티브 파이프 오퍼레이터 (|〉)를 사용할 수 있습니다. 작동법은 간단히 파이프 오퍼레이터의 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 합수를 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))
```

파이프 오퍼레이터는 특히 다음 설명할 dplyr의 group_by, split, filter, summary 등 행렬 편집/연산 함수를 빈번히 다양한 조합으로 쓰게되는 상황에서 더 큰 효과를 발휘할 수 있습니다. 일반적으로 파이프라인의 첫 단계 이후, 각 줄을 두 칸 들여쓰기합니다. 각 인자가 별도의 줄에 있으면 추가로 두 칸 더 들여쓰기를 합니다.

|〉는 플레이스홀더로 _를 사용하고 magrittr의 %〉% 플레이스홀더로 .을 사용합니다. magrittr의 %〉%는 R 데이터 분석에서 오랜 시간 동안 널리 사용되어 온 파이프 오퍼레이터로, 고급 기능과 유연성을 제공하지만 최근에는 네이티브 파이프 오퍼레이터 (|〉)의 사용이 권장되고 있으며 두 오퍼레이터 모두 R에서 데이터 처리와 분석을 보다 효율적이고 직관적으로 만들어주는 중요한 도구입니다.

다음 코드는 x가 paste의 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 colMeans라는 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))

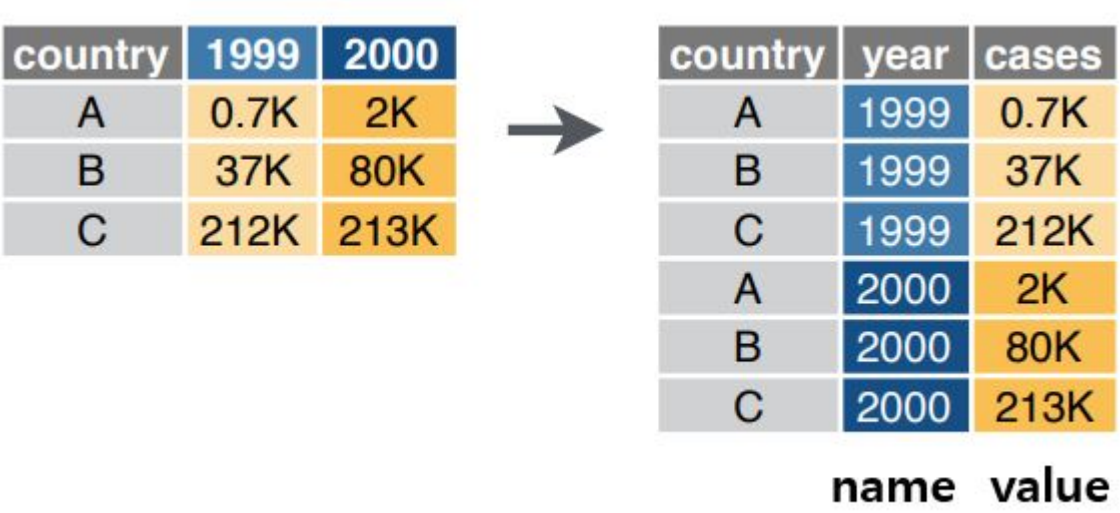
x 〈- data.frame(x=c(1:100), y=c(201:300))
x |〉
 colMeans() |〉
 sum()
```

만약 두 번째 파라미터에 입력으로 왼쪽 구문의 출력을 받아들이고 싶을 경우는 플레이스 홀더_을 사용하면 되겠습니다. round 함수는 두 개의 파라미터를 설정할 있 이으며 digits 라는 두 번째 파라미터에 값을 pipe operator로 넘겨주고 싶을 경우 아래와 같이 표현할 수 있습니다.

```
6 |〉 round(pi, digits=_)
round(pi, digits=6)
```

6.5 Pivoting

일반적으로 얻어지는 데이터의 형태는 wide형이며 이를 Long형으로 변환하기 위해서는 tidyverse 패키지에 속한 tidyr 패키지의 pivot_longer와 pivot_wider를 사용합니다. 기존에는 reshape2 패키지의 melt함수와 그 반대의 경우 dcast 함수를 사용할 수 있으나 tidyr 패키지가 널리 사용됩니다. wide형 데이터를 long형으로 변환하거나 long형을 wide형으로 변환하는 작업을 pivoting 이라고 합니다.



airquality 데이터는 wide형 데이터로도 볼 수 있고 long형 데이터로 볼 수도 있습니다. 앞에서 언급한 바와 같이 목적에 따라서 airquality 데이터가 wide형이 될 수 있고 long형도 될 수 있습니다. airquality 데이터는 특정 날짜에 airquality 지표 몇 가지에 대한 측정 값을 모아둔 데이터 입니다. 만약 Ozone과 Solar.R 만이 분석에 필요한 변수들이라면 두 개의 변수를 갖는 long형 데이터로 볼 수 도 있습니다.

```
airquality
myair2 <- airquality |>
 dplyr::select(Ozone, Solar.R, Day, Month)

myair2
ggplot(myair2, aes(x=Solar.R, y=Ozone)) +
 geom_point()

fit <- lm(myair2$Ozone~myair2$Solar.R)
summary(fit)
```

그러나 만약 각 컬럼을 airquality를 나타낼 수 있는 범주형 데이터 값으로 본다면 airquality 데이터는 wide형 데이터가 됩니다. 이 데이터를 long형으로 바꿀 경우 ID를 날짜로 하면 데이터들을 식별 할 수 있습니다. 그런데 날짜는 변수가 Month와 Day두 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

myair_long <- pivot_longer(myair, c("Ozone", "Solar.R", "Wind", "Temp"))
myair_long

myair_long <- myair |>
 pivot_longer(c("Ozone", "Solar.R", "Wind", "Temp"))
myair_long

myair_long2 <- myair |>
 pivot_longer(c(Ozone, Solar.R, Wind, Temp))
myair_long2

myair_long3 <- myair |>
 pivot_longer(!c(Month, Day))
myair_long3
```

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

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

long형 데이터를 wide형 데이터로 변환 할 수도 있습니다.

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

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 포멧에 대한 좀 더 구체적인 내용은 다음 링크를 참고하시기 바랍니다. https://www.theanalysisfactor.com/wide-and-long-data/

```
library(ggplot2)
data(msleep)
head(msleep)

ggplot(msleep, aes(x = brainwt, y = sleep_total)) +
 geom_point() +
 scale_x_log10() +
 xlab("Brain Weight (log scale)") +
 ylab("Total Sleep Time (hours)") +
 ggtitle("Relationship between Brain Weight and Total Sleep Time")
```

6.6 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 airquality데이터에서 Month와 Day 변수를 하나의 컬럼으로 병합하여 Date라는 변수로 만들어 주는 경우의 예 입니다.

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

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

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

6.7 dplyr

dplyr (https://dplyr.tidyverse.org/) 은 ggplot2을 개발한 해들리위컴이 (Hadley Wickham) 중심이 되어 만들어 졌으며 ggplot2와 함께 tidyverse의 (https://www.tidyverse.org/) 핵심 패키지 입니다. dplyr은 데이터를 다루는 크기나 분석의 속도, 편의성을 향상시켜 새롭게 만들어놓은 패키지 입니다. 기존 apply와 같은 행렬 연산 기능과 subset, split, group 와 같은 행렬 편집 기능을 더하여 만들어진 도구라고 할 수 있습니다.

dplyr의 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 dplyr은 이들을 보다 더 편리하게 사용할 수 있도록 만들어 놓은 것 입니다.

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

- select() - 변수 (columns) 선택
- filter() - 샘플 (rows) 선택
- arrange() - 샘플들의 정렬 순서 변경
- mutate() - 새로운 변수 만들기
- summarise() - 대표값 만들기
- group_by() - 그룹별로 계산 수행
- join() - 두 tibble 또는 data.frame을 병합할 때 사용
- 위 함수들과 (특히 filter, select, mutate, summarise) 조합하여 (함수 내에서) 사용할 수 있는 helper 함수들이 같이 사용될 수 있습니다 (독립적으로도 사용 가능).
  - across
  - if_any
  - if_all
  - everything
  - starts_with
  - end_with
  - contains

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 dplyr이 널리 사용되고 그 장점이 무엇인지 파악해 보도록 하겠습니다.

6.7.1 select

select() 는 주어진 데이터셋으로부터 관심있는 변수를 (column) 선택하여 보여줍니다.

```
head(iris)
iris |>
 select(Species, everything()) |>
 head(5)
iris |> select(Species, everything())
iris |> select(-Species)
```

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

```
iris |> select(starts_with('S'))
iris |> select(obs = starts_with('S'))
```

아래는 matches 함수를 사용한 방법 입니다. 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 (.)₩₩1은 하나의 문자 .가 (어떤 문자든) 한 번 더 ₩₩1 사용된 변수 이름을 말하며 이는 aavar 의 aa밖에 없으므로 aavar가 선택됩니다. grep에서 ^ 표시는 맨 처음을 나타내므로 ^S는 S로 시작하는 문자가 되겠습니다. 따라서 grep("^S", colnames(iris)) 의 경우 컬럼 이름 중 S로 시작하는 이름은 True로 그렇지 않으면 False 값을 리턴합니다.

6.7.2 filter

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()
```

filter의 ,로 구분되는 매개변수는 and 로직으로 묶인 조건입니다. 지난 강좌에서 보셨듯 R에서 and는 &, or는 |, 그리고 not은 ! 으로 사용하면 되며 filter에서 ,로 구분된 조건은 and와 같다고 보시면 되겠습니다.

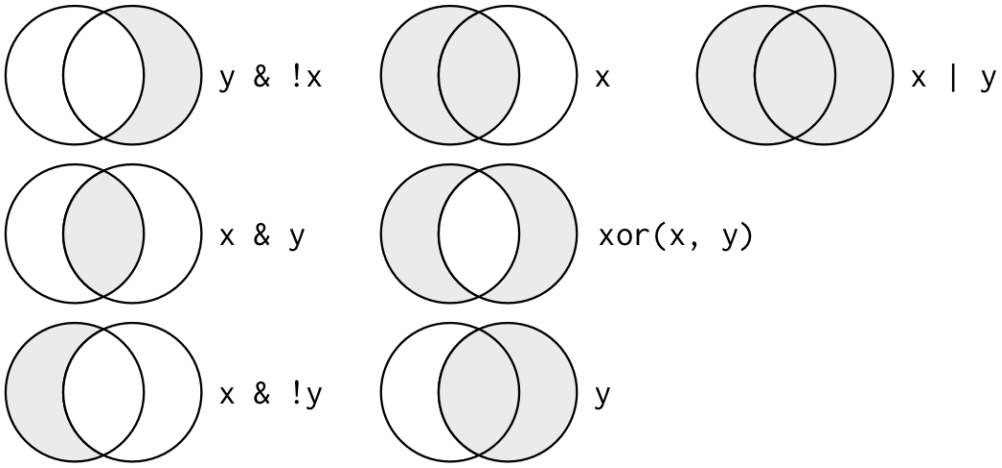


Image from (https://r4ds.had.co.nz/)

6.7.3 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)
```

6.7.4 mutate

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

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

6.7.5 summarise

summarise()는 data.frame내 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. 특히 summarise 함수는 다음과 같이 across, if_any, if_all 등의 helper 함수와 조합되어 사용이 가능합니다.

```
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))
```

6.7.6 join

join 함수는 데이터를 병합해주는 기능을 수행하는 함수 입니다. 네 가지 종류의 함수가 있으며 (left_join(), 'right_join(), 'inner_join(), 'full_join()) 기본적으로 공통되는 이름의 변수를 (key) 이용해서 공통되는 샘플끼리 자동으로 병합해 주는 기능을 수행합니다.by'에서 지정해준 파라메터의 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)
```

6.8 Code comparison

이제 split, apply, combine을 활용하여 평균을 구하는 코드와 dplyr 패키지를 사용하여 만든 코드를 비교해 보도록 하겠습니다. iris 데이터를 분석하여 품종별로 꽃받침의 길이 (Sepal.length)의 평균과 표준편차, 그리고 샘플의 수를 구해보는 코드입니다.

split은 factor형 변수인 Species를 기준으로 iris 데이터를 나누어 주는 역할을 하며 lapply는 list 형 데이터인 iris_split을 각 리스트의 각각의 원소들에 대해서 임의의 함수 function(x)... 를 수행하는 역할을 합니다. 마지막 data.frame으로 최종 경로를 combine 합니다.

```
iris_split <- split(iris, iris$Species)
iris_means <- lapply(iris_split, function(x) {mean(x$Sepal.Length)})
iris_sd <- lapply(iris_split, function(x) {sd(x$Sepal.Length)})
iris_cnt <- lapply(iris_split, function(x) {length(x$Sepal.Length)})
iris_df <- data.frame(unlist(iris_cnt), unlist(iris_means), unlist(iris_sd))
```

아래는 dplyr 패키지를 사용한 코드 입니다.

```
iris_df <- iris |>
 group_by(Species) |>
 summarise(n=n(), mean=mean(Sepal.Length), sd=sd(Sepal.Length))
```

위에서 보듯 dplyr 패키지를 사용할 경우 그 결과는 같으나 코드의 가독성과 효율성면에서 장점을 보여줍니다. iris 데이터를 받아서 Species에 명시된 그룹으로 나누고 원하는 함수를 타깃 컬럼에 대해서 적용하라는 의미 입니다. 다음은 모든 변수에 대한 평균을 구하는 코드 입니다.

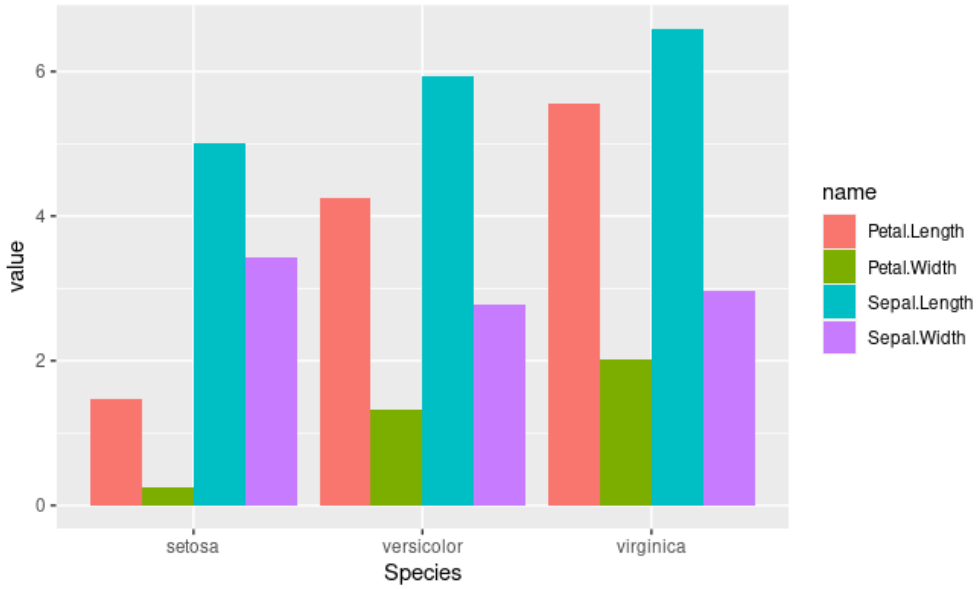
```
iris_mean_df <- iris |>
 group_by(Species) |>
 summarise(across(everything(), mean))
```

자세한 ggplot의 내용은 다음시간에 학습하겠지만 각 평균에 대한 막대그래프를 그러보겠습니다.

```
library(ggplot2)

iris_mean_df2 <- iris_mean_df |>
 pivot_longer(-Species)

ggplot(iris_mean_df2, aes(x=Species, y=value, fill=name)) +
 geom_bar(stat="identity", position="dodge")
```



Chapter 7

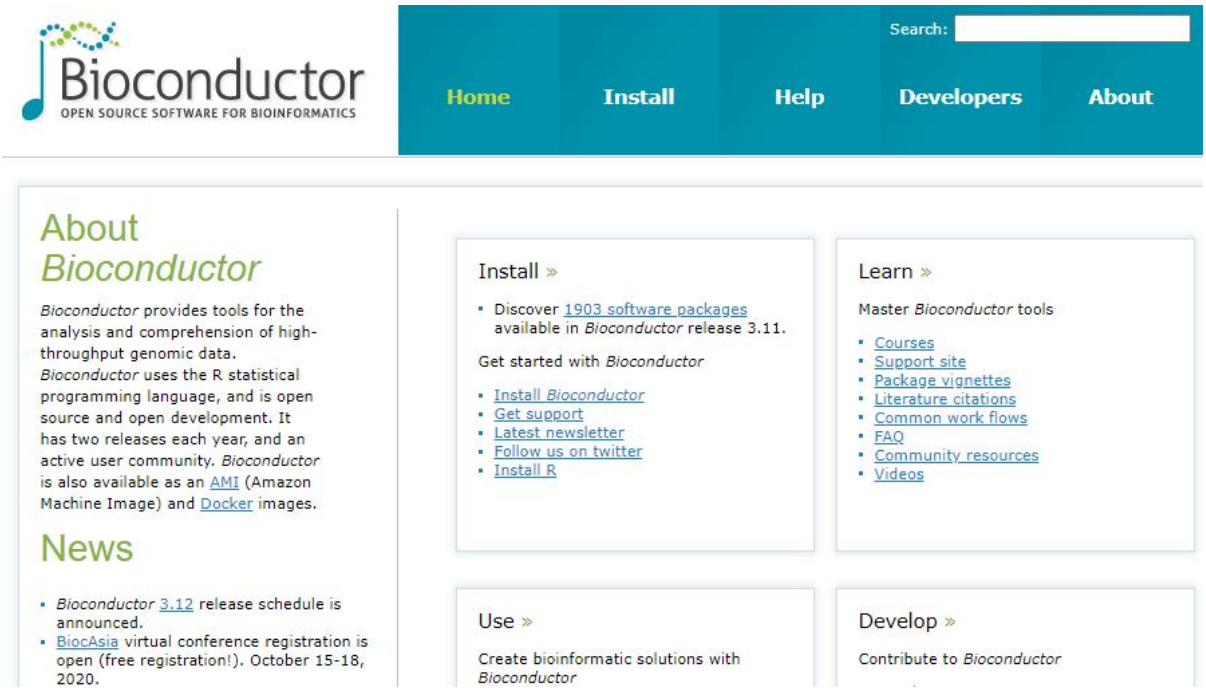
Bioconductor

7.1 Introduction

- https://www.bioconductor.org

Bioconductor는 바이오인포메틱스를 위한 R기반의 데이터, 메소드, 그리고 패키지들의 모음입니다. 2002년 microarray 데이터 분석을 위한 플랫폼으로 시작되었으며 현재 2000개 이상의 패키지로 구성되어 있습니다. R은 분산형 오픈소스이나 Bioconductor는 Full-time developer들에 의해서 유지되고 있습니다. CRAN에 배포되지 않고 CRAN에 비해 더 많은 필수 자료들 (vignettes 등)이 필요하며 높은 수준으로 quality control이 되고 있습니다. Bioconductor는 6개월마다 예정된 릴리스를 통해 모든 bioconductor 패키지가 충돌없이 조화롭게 작동하도록 유지되고 있습니다.

사용 가능한 패키지들은 이곳을 참고하시면 되겠습니다.



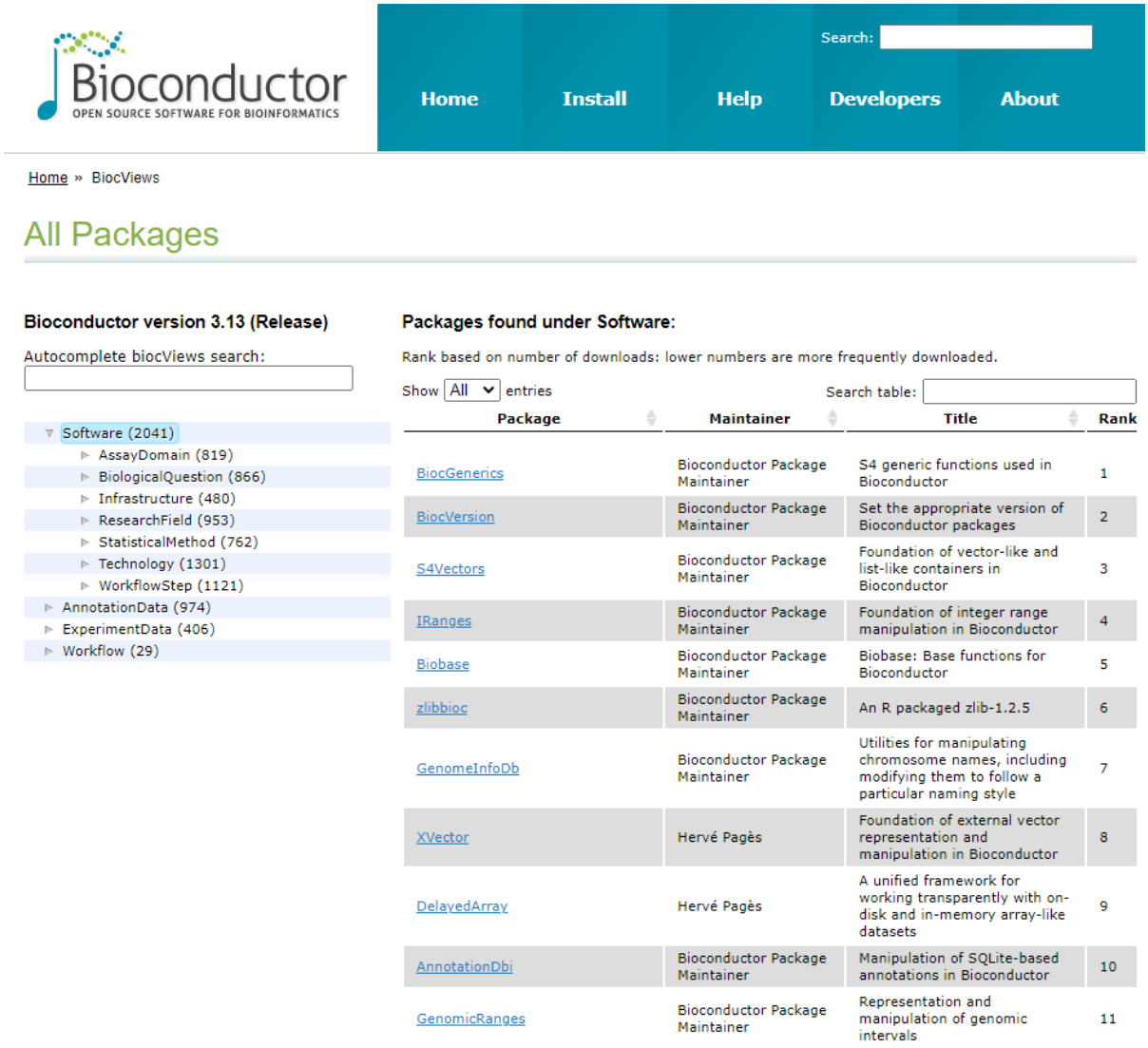
Bioconductor 코어 개발 그룹은 사용자들이 지놈스케일 데이터를 더 편리하게 다루룰 수 있도록 데이터의 구조를 개발하고 있습니다. Bioconductor의 주요 기능은 다음과 같습니다.

- 지놈스케일의 서열이나 발현등 대용량 유전자형 데이터 관리 및 통계적 분석을 위한 툴 제공
- 분자수준의 현상과 생장이나 질병 등 표현형수준의 관계를 규명하기 위한 정량 데이터 통합 및 관리

7.2 Packages

메인화면 〉〉 Use 〉〉 Software, Annotation, Experiment

- Software: 데이터 분석을 위한 알고리즘/툴 모음
- Annotation: 유전자 symbol/ID mapping, gene ontology 기반 유전자 분류, 유전체상에서 exon, transcript, gene 등의 위치, 단백질 기능 등. Annotation 〉 Packagetype 참고
- Experiment data: 검증된 실험 데이터
- Workflow: 특정 데이터 분석을 위한 프로세스 모음 RNA-seq, ChIP seq, copy number analysis, microarray methylation, classic expression analysis, flow cytometry 등



Annotation 리소스는 다음과 같이 몇 단계의 레벨로 구분할 수 있습니다.

- ChipDb: 가장 낮은 단계, Affymatrix Chip 정보
- OrgDb: 특정 생물 (Organism) 의 기능적 annotations
- TxDb/EnsDb: 전사체 정보, 위치 정보
- OrganismDb: meta-packages for OrgDb, TxDb
- BSgenome 특정 생물의 실제 염기 정보
- Others GO.db; KEGG.db
- AnnotationHub:
- biomaRt:

Bioconductor에서 제공하는 패키지를 설치하기 위해서는 BiocManager를 먼저 설치하고 해당 패키지를 설치하시기 바랍니다. BiocManager에는 available() 이라는 함수로 (특정 문자가 포함된) 사용 가능한 패키지를 검색할 수 도 있습니다. 예를 들어 IRanges라는 패키지를 설치할 경우 bioconductor 상단 오른쪽의 Search 나 software package list의 검색창에서 IRanges를 입력하여 해당 패키지를 찾고 다음과 같이 설치를 수행합니다.

```
if (!requireNamespace("BiocManager", quietly = TRUE))
  install.packages("BiocManager")

BiocManager::install("IRanges")
## .libPaths()
```

Exercises

1) OrganismDb는 meta-package의 형태로 OrgDb, TxDb, 그리고 GO.db 패키지들을 포함하는 정보를 가지고 있음. OrganismDB 중 인간의 정보를 가진 Homo.sapiens를 찾아 설치하시오

7.3 Learning and support

각 패키지는 제목, 저자, 유지관리자, 설명, 참조, 설치법 등의 정보가 포함된 landing page가 있으며 패키지 내 함수들은 상세한 설명과 예제가 제공됩니다. 예를 들어 IRanges의 landing page를 참고하세요. vignettes는 bioconductor의 중요한 특징 중 하나로 R 코드와 함께 패키지를 사용하는 방법에 대한 상세한 설명을 제공하는 문서입니다.

```
library(IRanges)

vignette(package="IRanges")
browseVignettes("IRanges")
vignette("IRangesOverview", package="IRanges")

ir1 <- IRanges(start=1:10, width=10:1)
ir1
class(ir1)
methods(class="IRanges")

example(IRanges)
?IRanges
??IRanges
```

메인페이지 >> Learn >> Support site 게시판에는 관련된 여러 QnA 들이 있어서 유사 문제에 대한 도움을 받을 수 있습니다.

7.4 OOP - Class, Object and Method

객체지향프로그래밍 (OOP)은 복잡한 문제를 프로그래밍할 때 발생되는 코드의 복잡성을 해결할 수 있는 하나의 방안으로 1990년대부터 많이 사용되었습니다.

R도 객체지향 프로그래밍 언어입니다. 그런데 R은 다른 언어들에 비해서 좀 어려운 (다른) 개념으로 사용됩니다. R에서 사용하는 Class에는 크게 base type, S3, S4, RC, 그리고 R6 등 다양한 타입이 있고 이 중 S3를 많이 사용해 왔으며 S3의 단점을 보완한 S4 형식의 class와 R6를 주로 사용합니다 (AdvancedR?). 본 강의에서는 S3 형식의 class만 다루도록 하겠습니다.

클래스를 사용하는 이유는 여러가지가 있겠지만 복잡한 개념의 데이터를 구조화하고 쉽게 관리하기 위해서 사용한다고 보면 될 것 같습니다. 여러분이 알아야할 개념은 Class와 Object 그리고 Method 입니다. 사실 R의 모든것이 Object이고 이러한 Object들의 정의가 Class 입니다.

```
df <- data.frame(x=c(1:5), y=LETTERS[1:5])
df
class(df)
```

위에서 df는 변수라고 부르지만 object이기도 합니다. df의 class는 data.frame 입니다. 클래스는 누구든 원하는 만큼 얼마든지 만들 수 있습니다.

```
class(df) <- "myclass"
df
class(df)

class(df) <- c("data.frame", "myclass")
df
class(df)
```

그런데 모든 object들이 OOP 유래는 아닙니다 base object들이 그 예입니다.

```
x <- 1:10
class(x)
attr(x, "class")

mtcars
attr(mtcars, "class")
```

method는 위와 같은 클래스들에 특화된 어떤 기능을 하는 함수라고 생각하시면 됩니다.

```
mt <- matrix(1:9, 3,3)
df <- data.frame(1:3, 4:6, 7:9)

class(mt)
class(df)
str(mt)
str(df)


diamonds <- ggplot2::diamonds

summary(diamonds$carat)
summary(diamonds$cut)

methods(class="data.frame")
```

위 summary, str 등이 generic function이라 불리는 method들 입니다. class마다 사용 가능한 method가 어떠한 정보가 있는지 알기 위해서 methods() 라는 함수를 사용합니다. R의 객체지향프로그래밍에 대한 상세한 내용은 Advanced R를 참고하세요.

Exercises

1) 다음 두 종류의 객체에 대해서 class 가 integer 일 경우 평균을 계산하고 character일 경우 비율을 계산하는 (table 함수 사용) mysummary 함수를 만드시오

```
x <- c(1:10)
y <- c("A", "G", "G", "T", "A")
```

7.5 Bioconductor의 OOP

bioconductor에서 다루는 genome 스케일의 experiment나 annotation은 대표적인 복잡한 데이터 중 하나 입니다. Bioconductor에서 OOP 개념은 다음과 같습니다.

class - 복잡한 생물학적 데이터 구조의 틀 정의
object - 특정 클래스가 특정 구현된 실체
method - 특정 클래스에 대한 기능 수행

예를 들어 앞에서 설치한 Homo.sapience의 class인 OrganismDb 살펴보면 다음과 같습니다.

```
library(Homo.sapiens)
class(Homo.sapiens)
?OrganismDb
```

> The OrganismDb class is a container for storing knowledge about existing Annotation packages and the relationships between these resources. The purpose of this object and it's associated methods is to provide a means by which users can conveniently query for data from several different annotation resources at the same time using a familiar interface.

```
homo_seq <- seqinfo(Homo.sapiens)
class(homo_seq)
?Seqinfo
```

> A Seqinfo object is a table-like object that contains basic information about a set of genomic sequences. ···

```
length(homo_seq)
seqnames(homo_seq)
```

bioconductor에는 대용량 정보가 object 형태로 구조화되어 저장되어 있으며 library() 함수로 읽어올 수 있고 다양한 함수로 해당 object의 정보를 읽어올 수 있습니다.

Exercises

1) Homo.sapiens 정보에서 상위 10개 유전자와 상위 10개 exon을 구하시오

```
genes(Homo.sapiens) [1:10]
exons(Homo.sapiens) [1:10]
```

Chapter 8

Biostrings

8.1 Introduction

High-throughput sequencing 데이터를 포함한 DNA나 Amino acid와 같은 생물학적 서열은 Bioconductor의 다양한 패키지들에 의해서 분석될 수 있으며 특히 Biostrings 패키지는 생물학적 서열을 효과적으로 활용하기 위한 핵심 도구로 활용됩니다.

8.2 Working with sequences

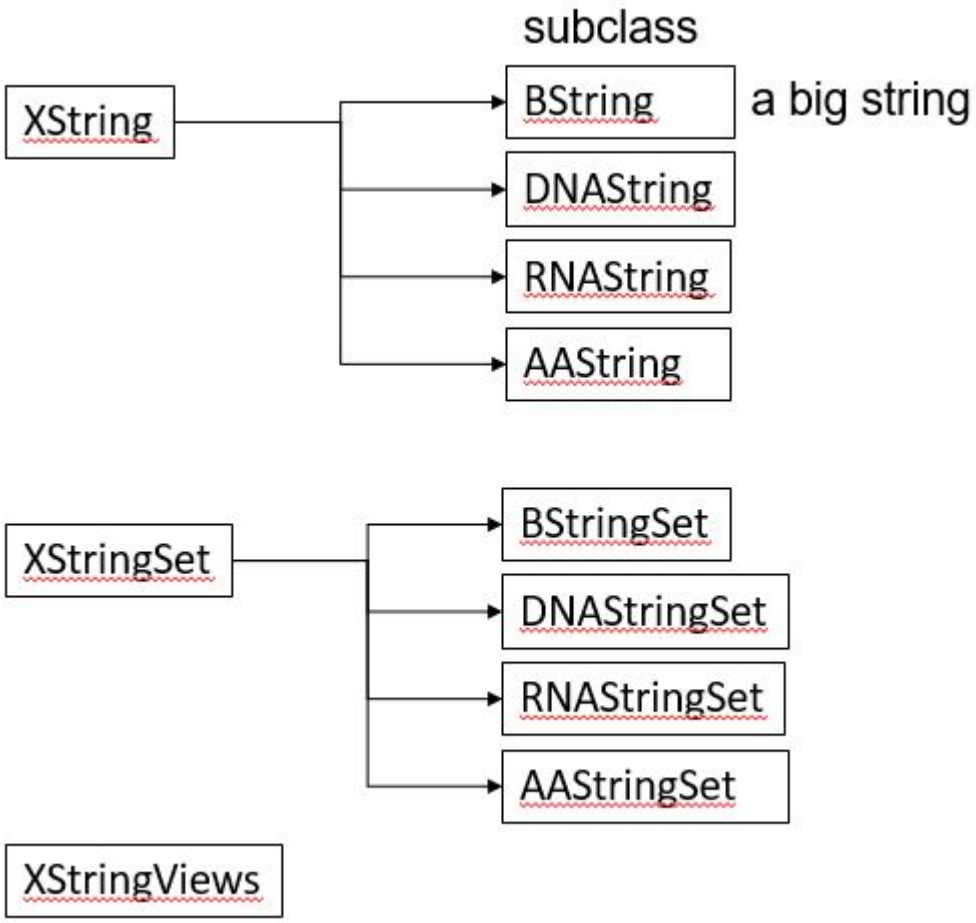
Biostrings는 DNA, RNA, amino acids와 같은 생물학적 string을 다루기 위한 다양한 함수를 제공하는 패키지 입니다. 특히 서열에서의 패턴 탐색이나 Smith-Waterman local alignments, Needleman-Wunsch global alignments 등의 서열 비교함수를 제공하여 간단한 서열 분석에 자주 활용되는 패키지 입니다 (sippl1999biological?). Biostrings 패키지의 설치 방법은 아래와 같습니다.

```
if (!requireNamespace("BiocManager", quietly = TRUE))
  install.packages("BiocManager")

BiocManager::install("Biostrings")
```

```
library(Biostrings)
```

Biostrings 패키지는 기본적으로 XString, XStringSet, XStringViews 3가지의 class를 정의하고 있습니다. XString은 DNA나 RNA, AA 등 생물학적 서열 한 가닥을 다루기위한 클래스이며 XStringSet은 여러 가닥을 다루기위한 클래스 입니다.



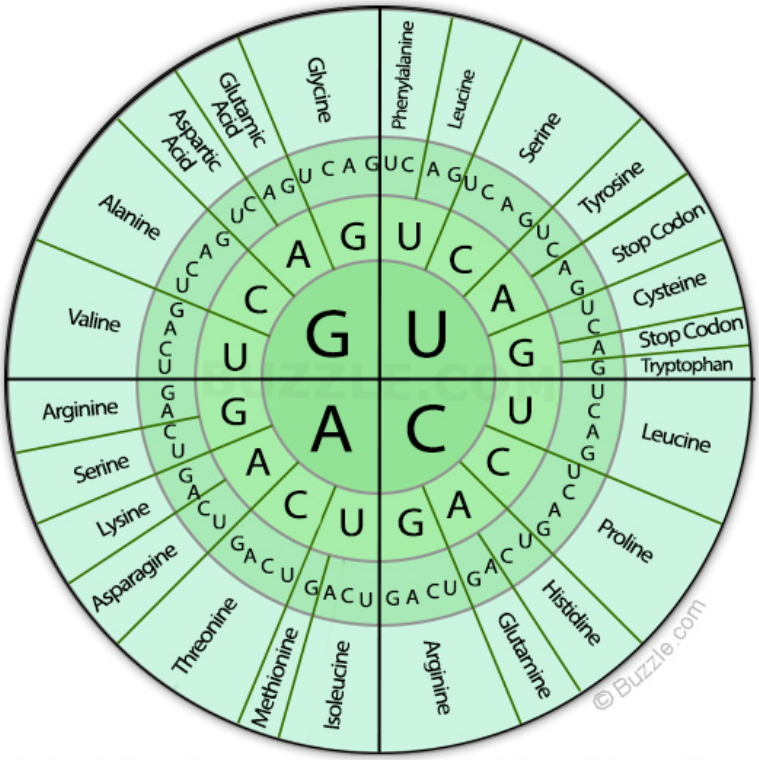
DNAString 함수를 이용해서 객체를 만들어낼 수 있으며 'A', 'C', 'G', 'T' 외에 '-' (insertion), 'N' 을 허용합니다.

```
dna1 <- DNAString("ACGT?")
dna1 <- DNAString("ACGT-N")
dna1[1]
dna1[2:3]

dna2 <- DNAStringSet(c("ACGT", "GTCA", "GCTA"))
dna2[1]
dna2[[1]]
dna2[[1]][1]
```

다음 내장변수 들은 Biostrings 패키지를 로드하면 자동으로 저장되는 변수들로 생물학적 서열을 미리 정의해 놓았습니다.
IUPAC (International Union of Pure and Applied Chemistry, 국제 순수□응용 화학 연합)

```
DNA_BASES
DNA_ALPHABET
IUPAC_CODE_MAP
GENETIC_CODE
```



To decode the codon, move from the center circle towards the periphery.

위 변수들을 이용하면 다음처럼 sample() 함수를 이용해서 랜덤하게 DNA 서열을 얻을 수 있습니다. DNA_BASES가 4개 길이를 갖는 벡터인데 이 중 10개를 뽑으려면 replace=T로 해야 합니다.

```
x0 <- sample(DNA_BASES, 10, replace = T)
x0
s1 <- "ATG"
s2 <- "CCC"
s3 <- paste(s1, s2, sep="")
s3
x1 <- paste(x0, collapse="")
x1
```

관련 함수는 Cheat sheat 참고

8.2.1 XString

XString 클래스는 DNAString과 RNAString, AAString의 subclass로 나눌 수 있습니다. DNAString class에서 length 함수는 핵산의 갯수를 (DNAStringSet 타입의 변수에서 length는 DNA 가닥의 갯수) 계산하며 핵산의 갯수는 nchar함수로 얻어낼 수 있습니다. toString은 DNAString 타입을 단순 문자열로 변환해주는 함수이며 상보서열, 역상보서열 등의 정보도 complement, reverseComplement 등을 사용하여 찾아낼 수 있습니다.

```
x0 <- paste(sample(DNA_BASES, 10, replace = T), collapse="")
x1 <- DNAString(x0)
class(x0)
class(x1)
length(x1)
toString(x1)
complement(x1)
Biostrings::complement(x1)
reverseComplement(x1)
```

DNAString의 인덱싱은 vector (string)과 같으며 DNAStringSet은 list의 인덱싱과 같습니다.

```
## indexing
x1[1]
x1[1:3]
subseq(x1, start=3, end=5)
subseq(x1, 3, 5)

## letter frequency
alphabetFrequency(x1, baseOnly=TRUE, as.prob=TRUE)
letterFrequency(x1, c("G", "C"), as.prob=TRUE)
```

Exercises

1) 개시코돈과 스탑코돈을 포함한 30개 길이를 갖는 랜덤 유전자서열을 하나 만드시오
2) AA_ALPHABET은 IUPAC에서 정의된 아미노산 서열 알파벳이 저장된 내장변수임. "M"과 "*"를 포함하는 10개 길이를 갖는 랜덤 유전자서열을 하나 만드시오

8.2.2 XStringSet

XStringSet역시 DNAStringSet, RNAStringSet, 그리고 AAStringSet으로 나눌 수 있으며 DNAStringSet class는 여러 개의 DNAString 을 모아 놓은 집합이라고 보면 됩니다. length 함수는 DNA string의 갯수이며 width 또는 nchar 함수로 각 string의 길이를 구할 수 있으며 이 외 대부분의 DNAString 에서 사용되는 함수가 동일하게 사용될 수 있습니다.

```
x0 <- c("CTC-NACCAGTAT", "TTGA", "TACCTAGAG")
x1 <- DNAStringSet(x0)
class(x0)
class(x1)
names(x1)
names(x1) <- c("A", "B", "C")
length(x1)
width(x1)
subseq(x1, 2, 4)
x1[[1]]
x1[1]


x3 <- DNAString("ATGAGTAGTTAG")
x4 <- c(x1, DNAStringSet(x3))
x4[-1]
x4
alphabetFrequency(x1, baseOnly=TRUE, as.prob=TRUE)
letterFrequency(x1, c("G", "C"), as.prob=TRUE)
rowSums(letterFrequency(x1, c("G", "C"), as.prob=TRUE))
subseq(x4, 2, 4)
```

RNA나 아미노산 역시 동일한 방식으로 적용 가능하며 c 함수를 이용해서 XStringSet으로 변환 가능합니다.

```
x1 <- paste(sample(AA_ALPHABET, 10, replace = T), collapse="")
x2 <- paste(sample(AA_ALPHABET, 10, replace=T), collapse="")

x3 <- AAString(x1)
x4 <- AAString(x2)

AAStringSet(c(x1, x2))
AAStringSet(c(x3, x4))
```

Exercises

1) 시작코돈과 종결코돈이 있는 길이 36bp 짜리 DNA (랜덤) 서열을 하나 만드시오

2) 위와 같은 랜덤서열 10개 만들어서 DNAStringSet으로 변환하시오

아래는 가장 직관적으로 생각할 수 있는 for를 이용한 방법입니다. 즉, 10개 저장소를 갖는 x0 변수를 미리 생성해 두고 for 문을 돌면서 서열을 하나씩 만들어 저장하는 방법입니다.

```
x0 <- rep("", 10)
for(i in 1:length(x0)) {
 tmp <- paste(sample(DNA_BASES, 30, replace = T), collapse="")
 x0[i] <- paste("ATG", tmp, "TAG", sep="")
}
x0
```

위 코드를 함수로 만들어 보겠습니다. random dna를 만들 때 길이만 다를뿐 같은 코드를 반복해서 사용하고 있습니다. 이럴 경우 DNA 길이를 사용자가 정해주도록 input parameter로 하고 해당 파라메터를 받아 DNA를 만들어 주는 함수를 만들어 사용하면 편리합니다.

```
data(DNA_BASES)
random_dna <- function(len) {
 tmp <- paste(sample(DNA_BASES, len, replace = T), collapse="")
 x0 <- paste("ATG", tmp, "TAG", sep="")
 return(x0)
}
random_dna(len=30)
random_dna(len=40)
```

파라메터로 넘겨진 len 값이 sample 함수의 len에 사용된 것을 참고하세요.

이제 길이 30bp짜리 10개의 서열을 반복해서 만들 때 위 함수를 앞서와 같이 for문을 이용하여 10번 반복해서 실행해 주면 같은 결과를 얻습니다. 위와 같이 함수를 만들어 두면 언제든 DNA 서열을 만들 때 재사용 할 수 있습니다.

```
x0 <- rep("", 10)
for(i in 1:length(x0)) {
 x0[i] <- random_dna(30)
}
x0
```

그런데 R에는 apply 와 같은 행렬연산 함수가 있어서 for문을 사용하지 않고 편리하게 반복문을 실행할 수 있습니다. replicate 함수는 apply와 같은 기능으로 list나 vector 변수에 대해서 사용할 수 있습니다. 즉, 다음과 같이 사용자가 원하는 함수를 반복해서 실행하고 반복 수 만큼의 길이를 갖는 결과를 반환합니다.

```
x0 <- replicate(10, random_dna(30))
x0
x1 <- DNAStringSet(x0)
x1
```

3. 위 생성한 10개 서열의 GC 비율을 계산하고 bar그래프를 그리시오

위 x0 스트링들을 XStringSet으로 바꾸고 GC 비율을 구한 후 bargraph를 그리겠습니다. gc_ratio가 G와 C의 비율값을 저장한 10x2 테이블이므로 x축에 10개의 서열과 각 서열의 GC비율을 나타내고 y축에 비율 값을 그리는 것으로 생각한 후 ggplot의 aes와 파라메터를 적절히 지정해 줍니다.

bar plot using ggplto2

```
x1 <- DNAStringSet(x0)
gc_ratio1 <- letterFrequency(x1, c("G", "C"), as.prob=TRUE)
gc_ratio2 <- rowSums(gc_ratio1)
barplot(gc_ratio2, beside=T)

names(gc_ratio2) <- paste("seq", 1:length(gc_ratio2), sep="")
```

```
barplot(gc_ratio2, beside=T)

data.frame(gc_ratio2) |>
 rownames_to_column() |>
 ggplot(aes(x=rowname, y=gc_ratio2, fill=rowname)) +
 geom_bar(stat="identity") +
 scale_y_continuous(limits = c(0, 1)) +
 scale_fill_brewer(palette = "green") +
 theme_bw()
```

8.2.3 XStringView

Biostrings의 또 다른 class인 XStringView는 XString class의 DNA, RNA, AA서열을 사용자가 원하는대로 볼 수 있는 인터페이스를 제공합니다. 사용법은 다음과 같습니다.

```
x2 <- x1 [[1]]
Views(x2, start=1, width=20)
Views(x2, start=1, end=4)
Views(x2, start=c(1,3), end=4)
Views(x2, start=c(1,3,4), width=20)
Views(x2, start=c(1,3,4), width=20)
i <- Views(x2, start=c(1,3,4), width=20)
```

다음과 같이 한 서열에 대한 여러 부분의 서열 조각도 볼 수 있으며 gaps 함수는 매개변수로 주어진 서열 view의 구간을 제외한 나머지 구간의 서열을 보여주는 함수입니다. successiveviews 함수는 처음 서열부터 매개변수 width에 주어진 갯수만큼의 서열을 보여주며 rep() 함수를 이용해서 서열의 처음부터 끝까지 보여주는 기능을 합니다.

```
v <- Views(x2, start=c(1,10), end=c(3,15))
gaps(v)

successiveViews(x2, width=20)
successiveViews(x2, width=rep(20, 2))
successiveViews(x2, width=rep(20, 3))
```

Exercises

1) 1000bp 길이의 랜덤 DNA 서열을 만들고 40bp 단위의 길이로 보는 코드를 작성하시오.

앞서 만들어둔 random_dna() 함수를 사용하면 되며 successiveViews 함수를 사용해야 하므로 DNAString으로 변환이 필요하며 서열의 길이에 따라서 rep() 를 이용하여 반복 횟수를 자동 계산합니다.

8.3 Sequence read and write

Biostrings 패키지의 readDNAStringSet이나 writeXStringSet을 사용하면 기본 DNA/RNA/AA 서열의 읽고 쓰기가 가능하며 fasta와 fastq 등의 파일타입으로 적용이 가능합니다.

```
x1 <- DNAStringSet(x0)
writeXStringSet(x1, "myfastaseq.fasta", format="fasta")

names(x1) <- "myfastaseq"
writeXStringSet(x1, "myfastaseq.fasta", format="fasta")

myseq <- readDNAStringSet("myfastaseq.fasta", format="fasta")
myseq
```

successiveViews로 나눈 여러개의 DNA 조각을 myfastaseqs.fasta에 저장하고 다시 읽을 수 있습니다.

```
myseqs <- DNAStringSet(sv)
names(myseqs) <- paste("myseqs", 1:length(myseqs), sep="")
writeXStringSet(myseqs, "myfastaseqs.fasta", format="fasta")
```

8.4 Sequence statistics

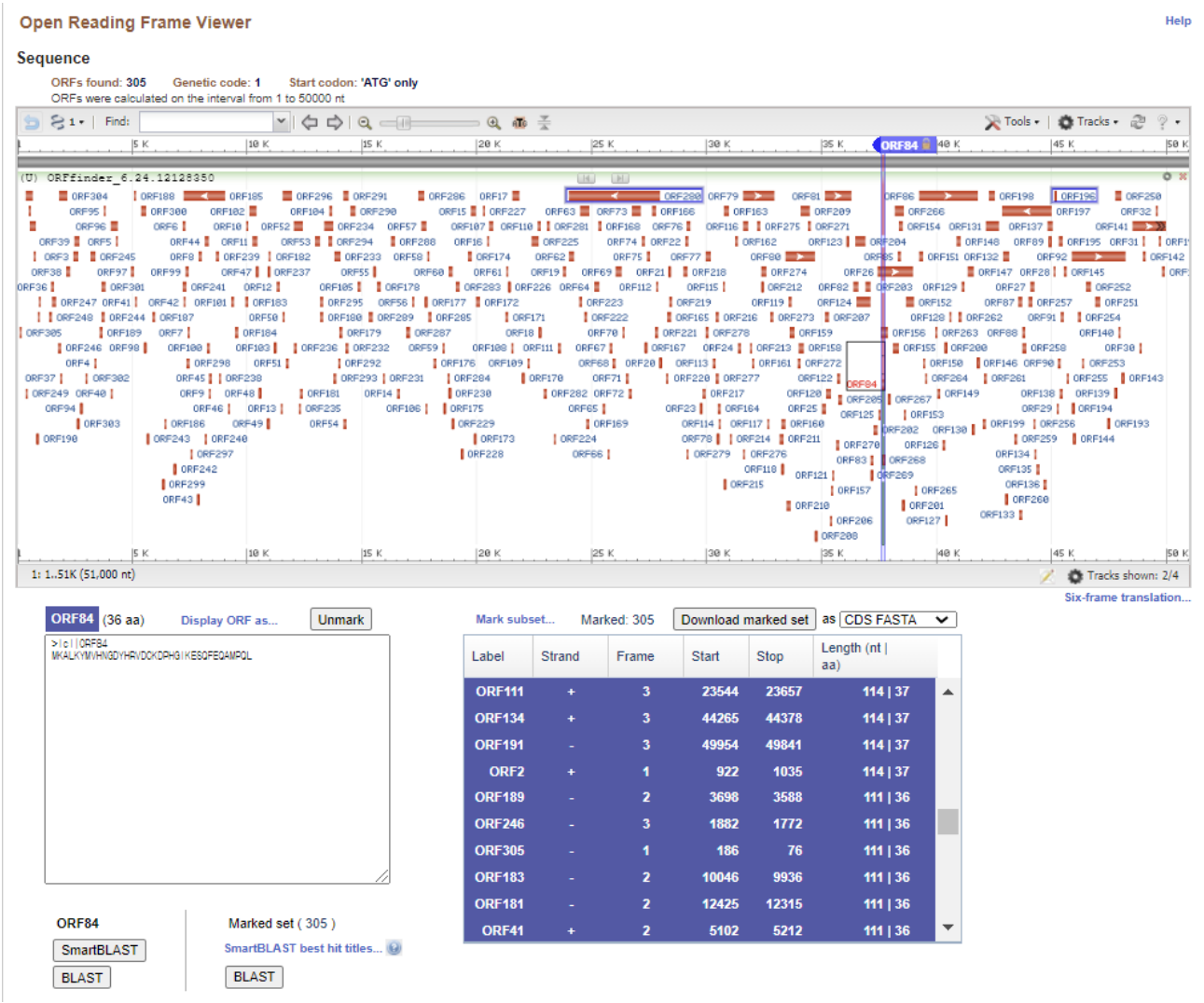
oligonucleotideFrequency 는 width와 step 이라는 옵션에 따라서 해당 서열의 모든 핵산의 수를 세어주는 합수입니다. 다음에 사용되는 yeastSEQCHR1는 Biostrings 패키지에 포함된 내장 데이터로서 yeast의 첫 번째 염색체 정보를 담고 있습니다.

```
data(yeastSEQCHR1) #Biostrings
yeast1 <- DNAString(yeastSEQCHR1)

oligonucleotideFrequency(yeast1, 3)
dinucleotideFrequency(yeast1)
trinucleotideFrequency(yeast1)

tri <- trinucleotideFrequency(yeast1, as.array=TRUE)
tri
```

아미노산 정보를 얻기 위해서 ORF를 찾아보겠습니다. yeast의 첫 번째 염색체에 대한 정보는 annotation이 되어 있지만 학습을 위해 툴을 사용하겠습니다. 이미 많은 종류의 ORF 탐색 툴이 나와있지만 본 강의에서는 NCBI에서 제공하는 orffinder를 사용하도록 하겠습니다.




```
my_ORFs <- readDNAStringSet("yeast1orf.cds")
hist(nchar(my_ORFs), br=100)
codon_usage <- trinucleotideFrequency(my_ORFs, step=3)
global_codon_usage <- trinucleotideFrequency(my_ORFs, step=3, simplify.as="collapsed")

colSums(codon_usage) == global_codon_usage
names(global_codon_usage) <- GENETIC_CODE[names(global_codon_usage)]
codonusage2 <- split(global_codon_usage, names(global_codon_usage))
global_codon_usage2 <- sapply(codonusage2, sum)
```

yeast 첫 번째 염색체에 대한 정보는 bioconductor annotationData OrgDb 또는 bioconductor annotationData TxDb 에서 찾아볼 수 있습니다.

```
#BiocManager::install("org.Sc.sgd.db")
library(org.Sc.sgd.db)
class(org.Sc.sgd.db)
?org.Sc.sgd.db
ls("package:org.Sc.sgd.db")
columns(org.Sc.sgd.db)
mykeys <- keys(org.Sc.sgd.db, keytype = "ENTREZID") [1:10]
AnnotationDbi::select(org.Sc.sgd.db,
                      keys=mykeys,
                      columns = c("ORF","DESCRIPTION"),
                      keytype="ENTREZID")
```

TxDb

```
BiocManager::install("TxDb.Scerevisiae.UCSC.sacCer3.sgdGene")
library(TxDb.Scerevisiae.UCSC.sacCer3.sgdGene)
class(TxDb.Scerevisiae.UCSC.sacCer3.sgdGene)
columns(TxDb.Scerevisiae.UCSC.sacCer3.sgdGene)
methods(class=class(TxDb.Scerevisiae.UCSC.sacCer3.sgdGene))
ygenes <- genes(TxDb.Scerevisiae.UCSC.sacCer3.sgdGene)
```

```
library(tidyverse)

mydat <- global_codon_usage2 |>
 data.frame |>
 rownames_to_column |>
 rename(codon = "rowname", freq = ".")

ggplot(mydat, aes(x=codon, y=freq)) +
 geom_bar(stat="identity")

mydat
AMINO_ACID_CODE[mydat$codon]
```

Exercises

1) AMINO_ACID_CODE를 이용해서 위 그래프의 1약자를 3약자로 변환, 라벨을 세로로 90도 회전, y축 라벨 “Frequency”, x축 라벨 “Amino acid code”, theme 옵션 “theme_bw” 등을 적용하여 다시 그림을 그리시오 (revisit ggplot2)

8.5 Pattern matching

Biostrings 패키지에는 하나의 subject 서열에 특정 pattern이 존재하는지 탐색하는 matchPattern함수를 제공합니다. 만약 여러개의 subject 서열에서 하나의 pattern을 찾을 경우에는 vmatchPattern함수를 사용하고 하나의 subject 서열에 여러개의 pattern을 찾는 경우에는 matchPDict 함수를 사용합니다.

```
length(coi)
hits <- matchPattern("ATG", yeast1, min.mismatch=0, max.mismatch=0)
hits
class(hits)
methods(class="XStringViews")
ranges(hits)

hits <- vmatchPattern("ATG", my_ORFs, min.mismatch=0, max.mismatch=0)
stack(hits)
```
