

PART 3. 데이터 분석 - 2,3장. R 프로그래밍 기초 / 데이터 마트

데이터 준전문가

ADSP, Advanced Data Analytics semi-Professional

류영표 강사

ryp1662@gmail.com



류영표

Youngpyo Ryu

동국대학교 수학과/응용수학 석사수료

現 Upstage AI X 네이버 부스트 캠프 AI tech 1~4기 멘토

前 Innovation on Quantum & CT(IQCT) 이사

前 한국파스퇴르연구소 Image Mining 인턴(Deep learning)

前 (주)셈웨어(수학컨텐츠, 데이터 분석 개발 및 연구인턴)

강의 경력

- 현대자동차 연구원 강의 (인공지능/머신러닝/딥러닝/강화학습)
- (주)모두의연구소 Aiffel 1기 퍼실리테이터(인공지능 교육)
- 인공지능 자연어처리(NLP) 기업데이터 분석 전문가 양성과정 멘토
- 공공데이터 청년 인턴 / SW공개개발자대회 멘토
- 고려대학교 선도대학 소속 30명 딥러닝 집중 강의
- 이젠 종로 아카데미(파이썬, ADSP 강사) / 강남 : ADSP
- 최적화된 도구(R/파이썬)을 활용한 애널리스트 양성과정(국비과정) 강사
- 한화, 하나금융사 교육
- 인공지능 신뢰성 확보를 위한 실무 전문가 자문위원
- 보건 · 바이오 AI활용 S/W개발 및 응용전문가 양성과정 강사
- Upstage AI X KT 융합기술원 기업교육 모델최적화 담당 조교

주요 프로젝트 및 기타사항

- 개인 맞춤형 당뇨병 예방·관리 인공지능 시스템 개발 및 고도화(안정화)
- 폐플라스틱 이미지 객체 검출 경진대회 3위
- 인공지능(AI)기반 데이터 사이언티스트 전문가 양성과정 1기 수료
- 제 1회 산업 수학 스터디 그룹 (질병에 영향을 미치는 유전자 정보 분석)
- 제 4,5회 산업 수학 스터디 그룹 (피부암, 유방암 분류)
- 빅데이터 여름학교 참석 (혼잡도를 최소화하는 새로운 노선 건설 위치의 최적화 문제)

R 이란?

- 통계 계산을 위한 프로그래밍 언어이면서 동시에 데이터 분석 소프트웨어
- AT&T의 Bell 연구소에서 만들어진 통계 프로그램 S에서 문법과 통계처리 부분을 참고하여 만들.
- 다양한 최신 통계분석과 마이닝 기능을 제공함. 세계적으로 많은 사용자들이 다양한 예제를 공유함.
- 다양한 기능을 지원하는 많은 패키지가 수시로 업데이트가 됨.

	SAS	SPSS	오픈소스 R
프로그램 비용	유료,고가	유료,고가	오픈소스
설치용량	대용량	대용량	모듈화로 간단
다양한 모듈 지원 및 비용	별도구매	별도구매	오픈소스
최근 알고리즘 및 기술반영	느림	다소 느림	매우 빠름
학습자료 입수의 편의성	유료 도서 위주	유료 도서 위주	공개 논문 및 자료 많음
질의를 위한 공개 커뮤니티	NA	NA	매우 활발

R 특징

- **오픈소스 프로그램**

: 사용자 커뮤니티에 도움 요청이 쉽다. R 사용자들이 패키지를 만들어 공유하는 CRAN(<https://cran.r-project.org/>)을 통해 최신 분석 기법이 포함된 package들을 쉽게 이용할 수 있음.

- **그래픽 및 성능**

: 프로그래밍이나 그래픽 측면 등 대부분의 주요 특징들에서 상용 프로그램과 대등하거나 월등하다.

- **시스템 데이터 저장 방식**

: 각 세션 사이마다 시스템에 데이터셋을 저장하므로 매번 데이터를 로딩할 필요가 없고 명령어 스토리도 저장 가능하다.

- **모든 운영체제**

: 윈도우, 맥, 리눅스 운영체제에서 사용 가능하다.

R 특징

- 표준 플랫폼

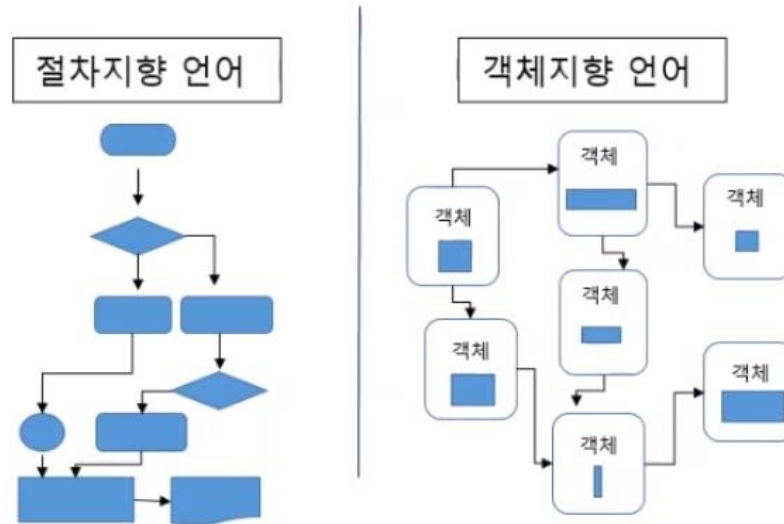
: S 통계 언어를 기반으로 구현된다.

- 객체지향 언어이며 함수형 언어

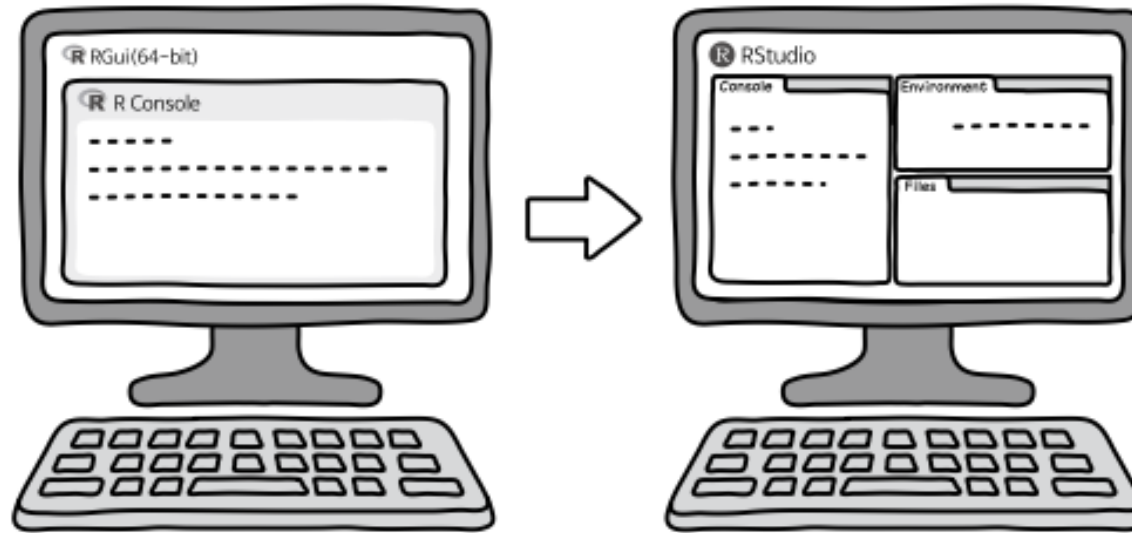
: 통계 기능뿐만 아니라 일반 프로그래밍 언어처럼 자동화하거나 새로운 함수를 생성하여 사용가능함.

- 객체 지향 언어 : 컴퓨터 프로그래밍의 한가지 기법으로 객체를 만들고 객체를 사용하는 프로그래밍 방법.

- 함수형 언어 : 자료 처리를 수학적 함수의 계산으로 취급하고 상태와 가변데이터를 멀리하는 프로그래밍



R 특징



R	RGui	R 스튜디오
프로그래밍 언어	R을 실행할 수 있는 통합 개발 환경	R을 보다 효과적이고 편리하게 사용할 수 있도록 돕는 GUI 프로그램

시각화 도구

- **강력한 시각화**

ggplot : <https://ggplot2.tidyverse.org/>

shiny : https://rstudio.github.io/shinydashboard/get_started.html

echarts4r : https://echarts4r.john-coene.com/articles/chart_types.html

- **파이썬**

plotly(make R) : <https://plotly.com/r/creating-and-updating-figures/>

plotly(make python) : <https://plotly.com/python/>

bokeh : <https://docs.bokeh.org/en/latest/docs/gallery.html>

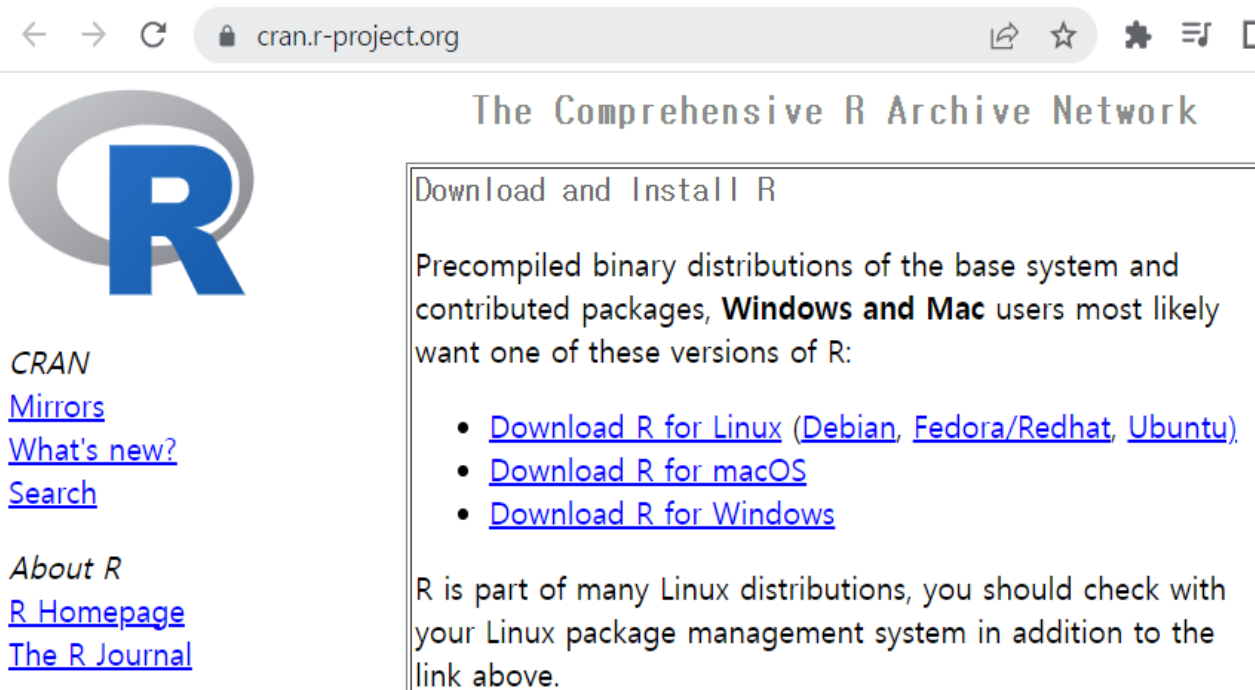
- **Tabuler**

: <https://www.tableau.com/learn/articles/best-beautiful-data-visualization-examples>

- **시각화 도구 참고** : <https://www.finereport.com/kr/10-data-visualization-tools-open-source-that-you-cannot-miss-in-2020/>

R과 Rstudio 설치

- R을 다운로드하여 설치하고 R 개발을 편하기 위한 IDE인 Rstudio를 다운하여 설치하여야 함.
- R 설치 시 주의할 점
 - : 설치시 윈도우 사용자가 계정이 한글인 경우 설치 및 실행에 오류가 발생할 수 있음.
 - : 계정이 한글인 경우 영문으로 새로운 계정을 생성해서 R을 설치해야 됨.



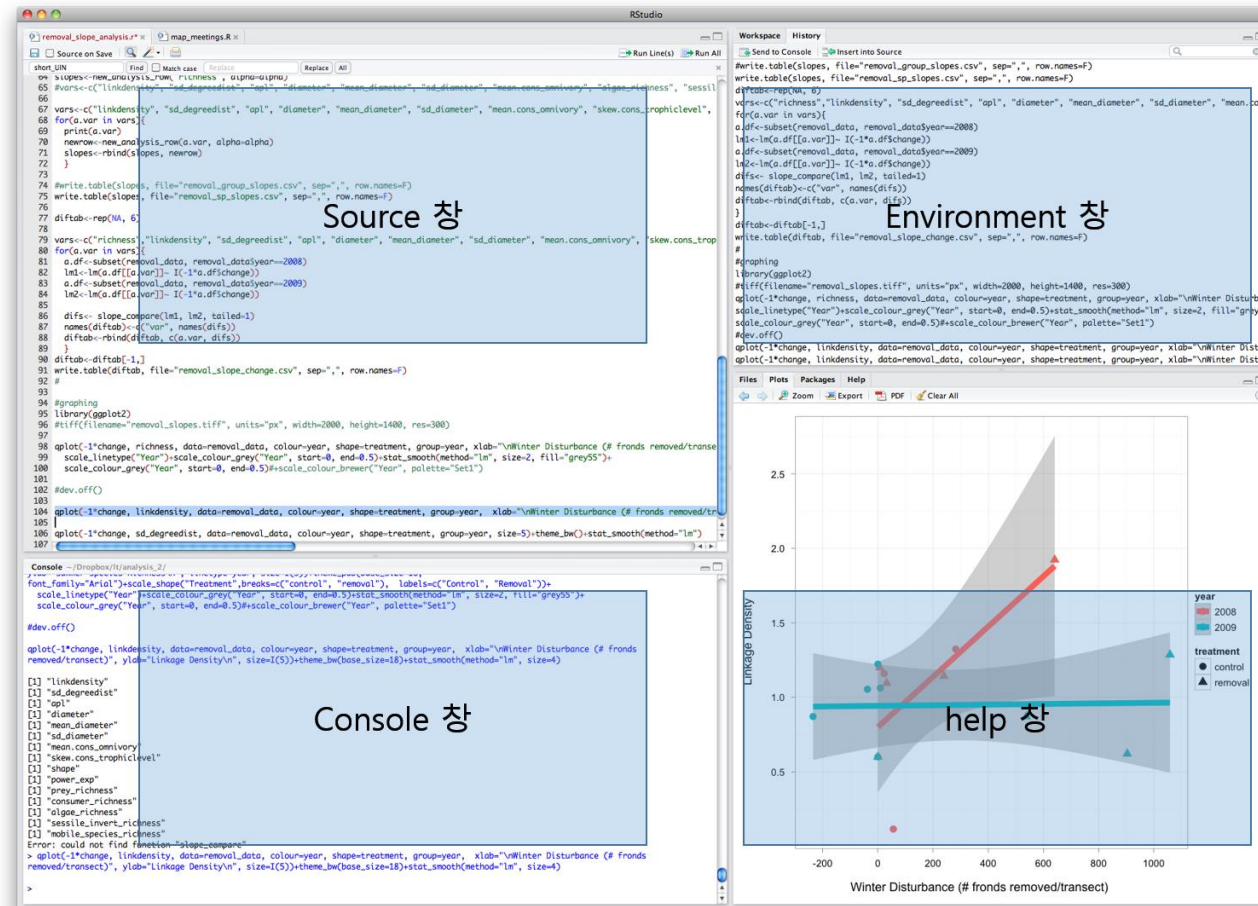
The screenshot shows the CRAN (Comprehensive R Archive Network) website. The main heading is 'The Comprehensive R Archive Network'. Below it, there's a section titled 'Download and Install R' which states: 'Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:'. It lists three download links: 'Download R for Linux (Debian, Fedora/Redhat, Ubuntu)', 'Download R for macOS', and 'Download R for Windows'. At the bottom, it mentions 'R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.' On the left side, there's a CRAN logo and links for 'CRAN Mirrors', 'What's new?', 'Search', 'About R', 'R Homepage', and 'The R Journal'.

- <https://www.rstudio.com/products/rstudio/download/>

RStudio Desktop	RStudio Desktop Pro	RStudio Server	RStudio Workbench ⓘ
Open Source License	Commercial License	Open Source License	Commercial License
Free	\$995 /year	Free	\$4,975 /year (5 Named Users)
DOWNLOAD	BUY	DOWNLOAD	BUY

R Studio

- 메모리에 어떤 변수가 어떻게 되어 있는지와 타입이 무엇인지를 볼 수 있고, 스크립트 관리와 도큐멘테이션이 편리
- 코딩을 해야하는 부담이 있으나 스크립트용 프로그래밍으로 어렵지 않게 자동화가 가능.



Why Python

TIOBE Index

May 2021 ▲	May 2020 ◆	Change ◆	Programming language ◆	Ratings ◆	Change ◆
1	1		C	13.38%	-3.68%
2	3	↑	Python	11.87%	+2.75%
3	2	↓	Java	11.74%	-4.54%
4	4		C++	7.81%	+1.69%
5	5		C#	4.41%	+0.12%
6	6		Visual Basic	4.02%	-0.16%
7	7		JavaScript	2.45%	-0.23%
8	14	↑ ↑	Assembly language	2.43%	+1.31%
9	8	↓	PHP	1.86%	-0.63%
10	9	↓	SQL	1.71%	-0.38%
11	15	↑ ↑	Ruby	1.50%	+0.48%
12	17	↑ ↑	Classic Visual Basic	1.41%	+0.53%
13	10	↓	R	1.38%	-0.46%
14	38	↑ ↑	Groovy	1.25%	+0.96%
15	13	↓	MATLAB	1.23%	+0.06%
16	12	↓ ↓	Go	1.22%	-0.05%
17	23	↑ ↑	Delphi/Object Pascal	1.21%	+0.60%
18	11	↓ ↓	Swift	1.14%	-0.65%
19	18	↓	Perl	1.04%	+0.16%
20	34	↑ ↑	Fortran	0.83%	+0.51%

PYPL Index (Worldwide)

May 2021 ▲	Change ◆	Programming language ◆	Share ◆	Trends
1		Python	29.9 %	-1.2 %
2		Java	17.72 %	-0.0 %
3		JavaScript	8.31 %	+0.4 %
4		C#	6.9 %	-0.1 %
5	↑	C/C++	6.62 %	+0.9 %
6	↓	PHP	6.15 %	+0.1 %
7		R	3.93 %	+0.0 %
8		Objective-C	2.52 %	+0.1 %
9		Swift	1.96 %	-0.2 %
10	↑	TypeScript	1.89 %	+0.0 %
11	↓	Matlab	1.71 %	-0.2 %
12		Kotlin	1.62 %	+0.1 %
13	↑	Go	1.42 %	+0.1 %
14	↓	VBA	1.33 %	-0.0 %
15	↑ ↑ ↑	Rust	1.13 %	+0.4 %
16	↓	Ruby	1.12 %	-0.1 %
17	↑ ↑ ↑ ↑ ↑ ↑ ↑	Ada	0.72 %	+0.3 %
18	↓	Visual Basic	0.7 %	-0.2 %
19	↓ ↓ ↓	Scala	0.67 %	-0.4 %
20	↓	Abap	0.61 %	+0.1 %
21	↓	Dart	0.55 %	+0.0 %
22	↑ ↑	Lua	0.49 %	+0.1 %
23	↑ ↑ ↑	Julia	0.42 %	+0.1 %
24	↓ ↓ ↓	Groovy	0.41 %	-0.0 %
25	↓ ↓ ↓	Perl	0.4 %	-0.0 %
26	↓ ↓ ↓	Cobol	0.36 %	-0.1 %
27	↑	Delphi/Pascal	0.24 %	-0.0 %
28	↓	Haskell	0.21 %	-0.1 %

Oct 2021	Oct 2020	Change	Programming Language	Ratings	Change
1	3	▲	 Python	11.27%	-0.00%
2	1	▼	 C	11.16%	-5.79%
3	2	▼	 Java	10.46%	-2.11%
4	4		 C++	7.50%	+0.57%
5	5		 C#	5.26%	+1.10%
6	6		 Visual Basic	5.24%	+1.27%
7	7		 JavaScript	2.19%	+0.05%
8	10	▲	 SQL	2.17%	+0.61%
9	8	▼	 PHP	2.10%	+0.01%
10	17	▲	 Assembly language	2.06%	+0.99%

What is programming?

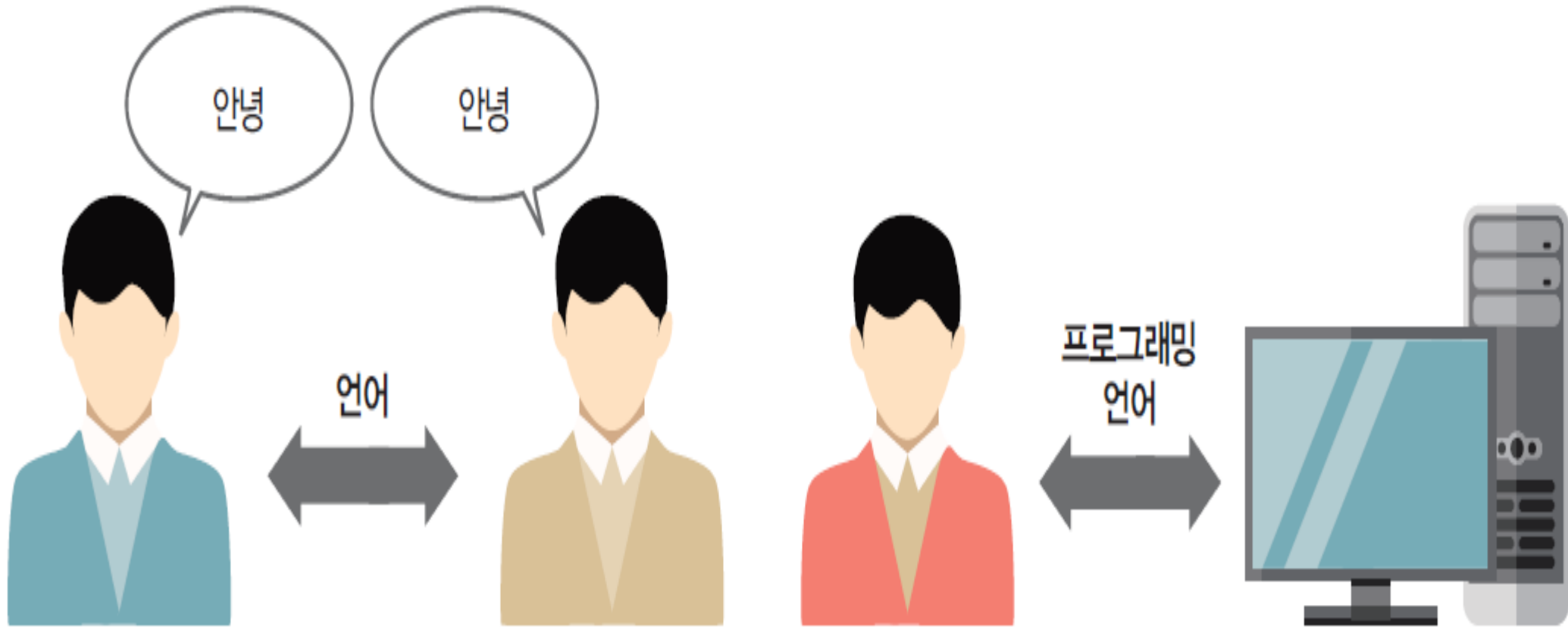


Fig. 1: 코딩교육은 왜 필요할까요?(코딩으로 준비하는 미래)



“세상의 모든 사람은 컴퓨터 프로그래밍을 배워야 합니다.
프로그래밍은 생각하는 법을 가르쳐 주기 때문입니다.”

“Everybody in this country should learn to program a computer,
because it teaches you how to think”

- 스티브 잡스 -



“프로그램은 사고력과 문제해결력을 향상시킵니다.”

“Learning to write programs stretches your mind,
and helps you think better.”

- 빌 게이츠 -



“코딩 교육은 당신의 미래일 뿐만 아니라 조국의 미래다.”

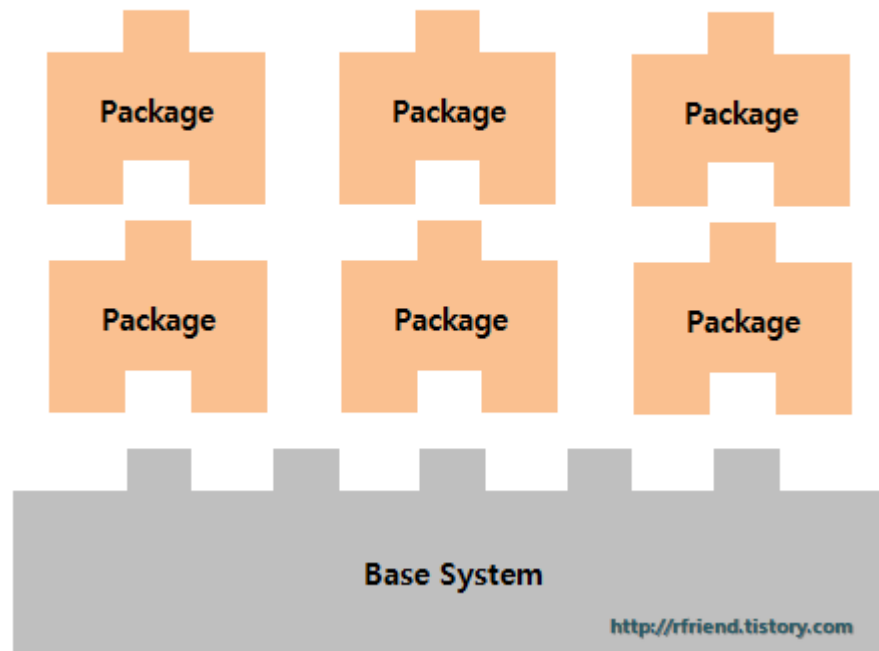
“Learning these skills isn't just important for your future,
it's important for our country's future.”

- 버락 오바마 -

R 패키지 설치하기

- 여러 패키지가 존재하는데, 이 패키지는 R의 함수들이 모여 있다고 생각하면 됨.
- 예) 선형회귀분석은 MASS라는 패키지 안에 들어 있고 `install.package("__패키지__")`를 통해 설치해주고 `library()`로 해당 패키지를 불러와서 설치한다.

【 Base System과 Package의 관계 】

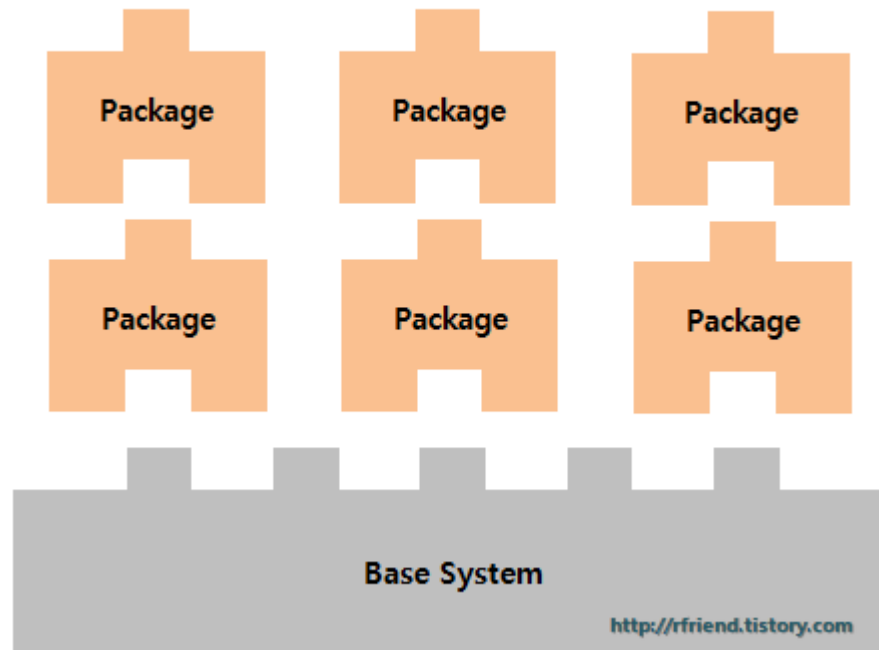


<http://rfriend.tistory.com>

R 패키지 설치하기

- Base system은 CRAN(Comprehensive R Archive Network, <https://cran.r-project.org>) site에 접속해서 다운로드해서 설치했던 R 프로그램을 말함.
- **Package** : R함수, 데이터, 컴파일된 코드 등을 모아놓는 것을 말하며, 통계분석 목적/필요에 따라 Base system에 레고 블록처럼 설치해서 사용하게 됨.

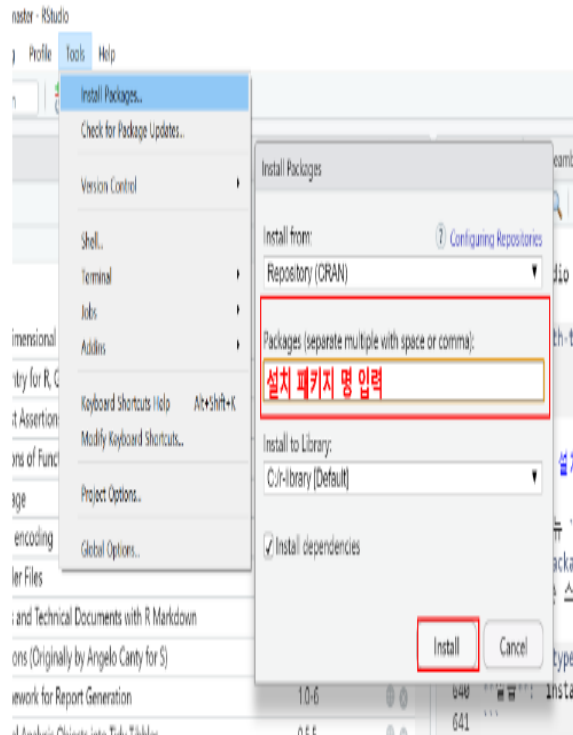
【 Base System과 Package의 관계 】



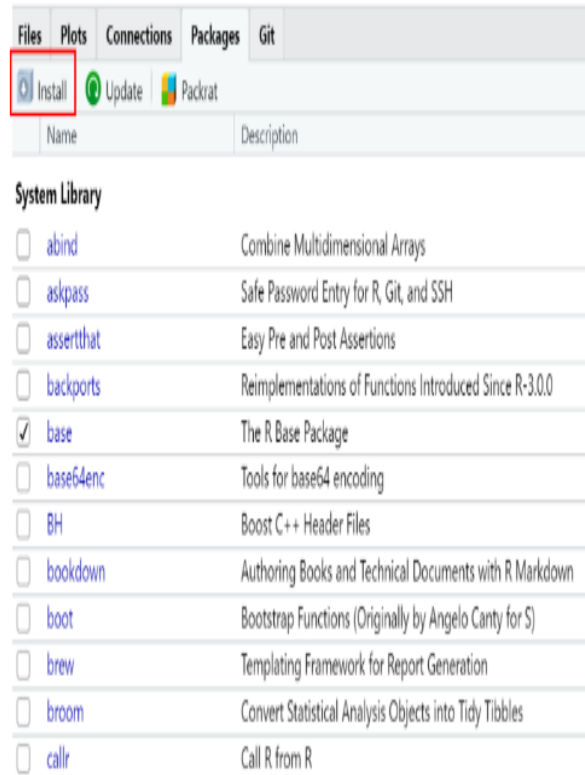
<http://rfriend.tistory.com>

R 패키지 설치하기

- RStudio 메뉴 [Tools] → [Install packages] 클릭 후 생성된 팝업 창에서 설치하고자 하는 패키지 입력 후 설치



- RStudio Packages 창에서 [Install] 버튼 누르면 위와 동일한 팝업창이 나타남(위와 동일)



- R 콘솔 또는 스크립트 창에서 `install.packages(package_name)` 함수를 사용해서 패키지 설치



실습: `install.packages()` 함수를 이용해 tidyverse 패키지 설치

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

위 명령어를 실행하면 tidyverse 패키지 뿐 아니라 연관된 패키지들이 동시에 설치됨

R 패키지 설치하기

```
# ggplot2 package를 설치하고 사용해보자
install.packages("ggplot2")
library(ggplot2)

# 문자로 구성된 vector 생성
x <- c("a", "b", "c", "a", "b", "a")

# qplot()을 이용하여 빈도 막대 그래프를 그려보자
qplot(x)

# ggplot2 package 삭제
remove.packages("ggplot2")

# 다양한 package를 설치하면 package들은 R이 설치된 경로 또는 내 문서에 설치된다.

# library 설치 경로 확인
.libPaths()

# library 설치 경로 변경
.libPaths("c:/R_workspace/R_Lecture/lib")
```

```
# R의 도움말 기능을 사용해보자

help(mean)

# 기본 함수의 파라미터를 확인해보자

args(max)

# 기본 함수의 사용예제

example(mean)





# qplot 함수의 사용예제

example(qplot)
```

R 기초 숫자 계산과 데이터 타입

```
1 install.packages('dslabs')
2 library(dslabs)
3 2+6
4 sqrt(5)
5 3*(pi/2)-1
6 x<-2+6
7 result<-sqrt(5)
8 result
```

```
> library(dslabs)
> 2+6
[1] 8
> sqrt(5)
[1] 2.236068
> 3*(pi/2)-1
[1] 3.712389
> x<-2+6
> result<-sqrt(5)
> result
[1] 2.236068
```

Environment	History	Connections	Tutorial
  Import Dataset ▾	 214 MiB ▾		
R ▾ Global Environment ▾			
Values			
result	2.23606797749979		
x	8		

```
> help(log)
```

Files Plots Packages Help Viewer Presentation

R: Logarithms and Exponentials ▾ Find in Topic

log (base) R Documentation

Logarithms and Exponentials

Description

`log` computes logarithms, by default natural logarithms, `log10` computes common (i.e., base 10) logarithms, and `log2` computes binary (i.e., base 2) logarithms. The general form `log(x, base)` computes logarithms with base `base`.

`log1p(x)` computes $\log(1+x)$ accurately also for $|x| \ll 1$.

`exp` computes the exponential function.

`expm1(x)` computes $\exp(x) - 1$ accurately also for $|x| \ll 1$.

Usage

```
log(x, base = exp(1))
logb(x, base = exp(1))
log10(x)
log2(x)

log1p(x)

exp(x)
expm1(x)
```

Notation

Column (열), Variables, Fields, Attributes

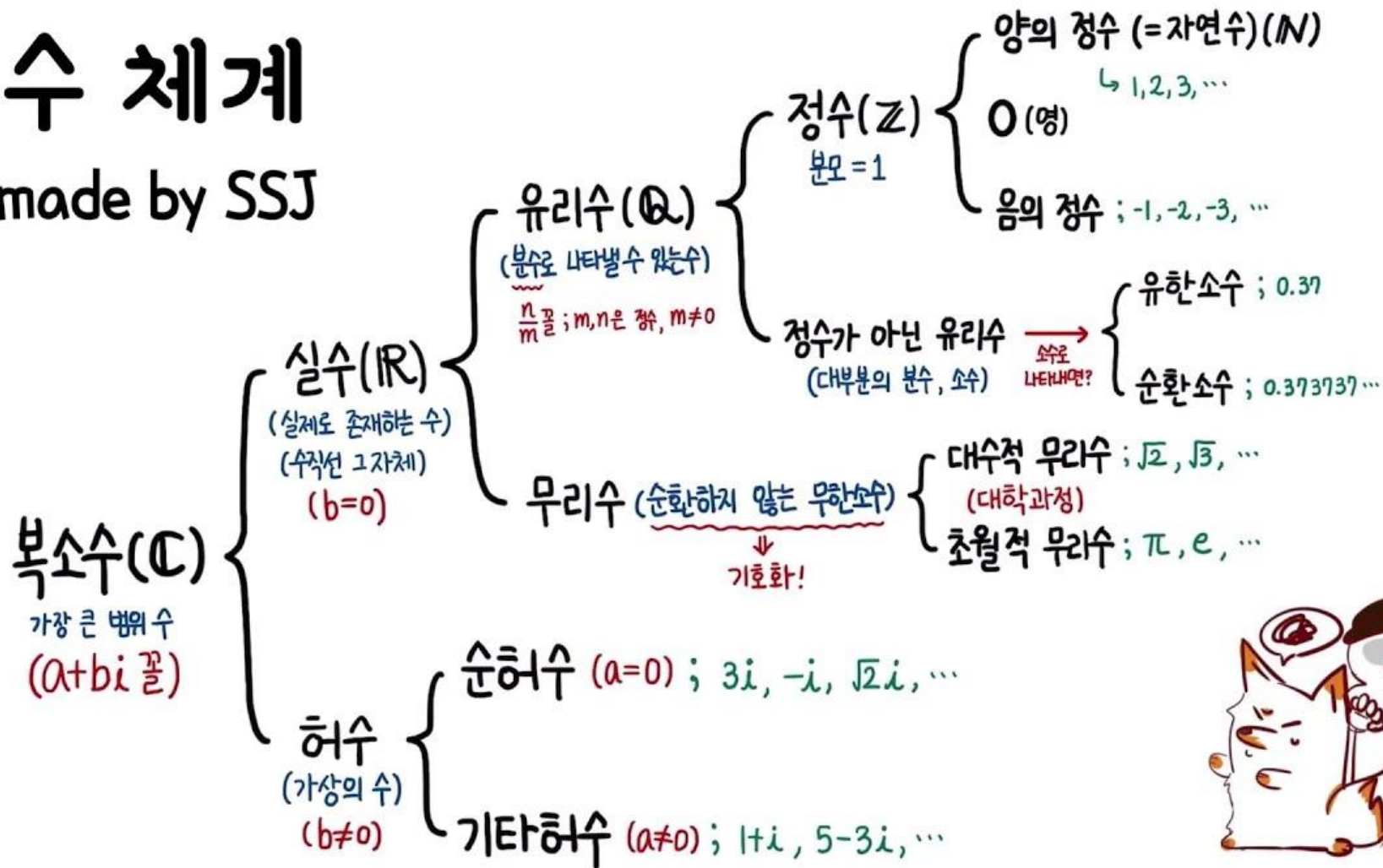
Row(행)
Observation
Records
Examples

PatientID	AdmDate	Age	Diabetes	Status
1	2015/10/15	25	Type1	Poor
2	2015/11/14	34	Type2	Improved
3	2015/10/21	28	Type1	Excellent
4	2015/10/28	52	Type1	Poor

- 통계: observations(관측치) / variables(변수)
- DB: records / fields
- Data-mining, Machine-learning: examples / attributes

수 체계

수 체계 made by SSJ



선형대수학(linear Algebra)

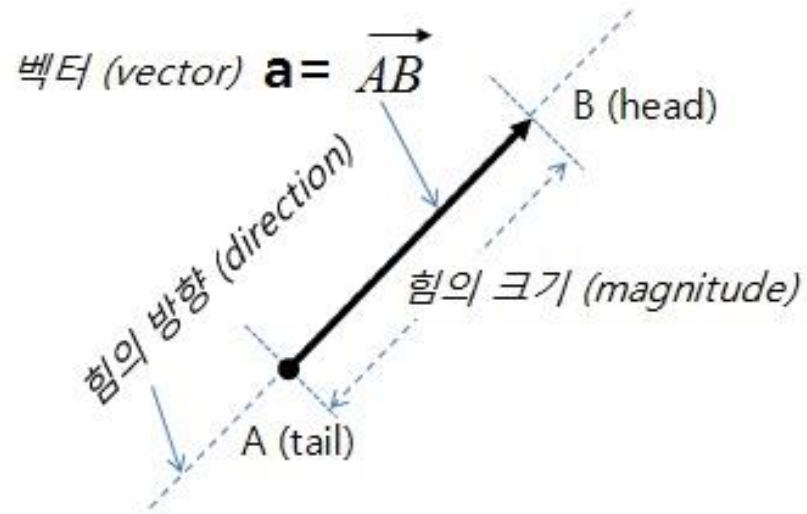
- 선형대수학은 기계공학, 통계학, 수학, 전자공학 등 다양한 학문의 기초과목이다.

수많은 학문의 토대가 되기 때문에 매우 중요

- 이름을 풀이하자면, ‘선형’과 ‘대수학’의 합성어이다. 선형이라는 것은 1차를 의미하고 대수학은 쉽게 말하면 ‘방정식’을 푸는 것.
- 다양한 분야에서 선형 방정식의 문제를 만나는 경우가 많음.
- 복잡하고 어려운 문제를 선형 방정식 문제로 쪼개고 축약 시킬 수 있는 경우가 많기 때문에 유용.

벡터(Vector)란?

- 스칼라(Scalar) : 크기만 주어지는 완전히 표시되는 양 ex)길이, 넓이, 질량,온도
- 벡터(Vector) : 크기 뿐만 아니라 방향까지 지정하지 완전히 표현할 수 없는 양



- 화살표의 길이 : 힘의 크기(magnitude) $|\overrightarrow{AB}|$
- 화살표의 방향 : 힘의 방향(direction)

벡터(Vector)란?

벡터의 표기는 아래와 같다.

$$V = \begin{bmatrix} a \\ b \end{bmatrix}$$

a와 b는 벡터의 성분(component)라고 불리며, v는 벡터라고 부름.

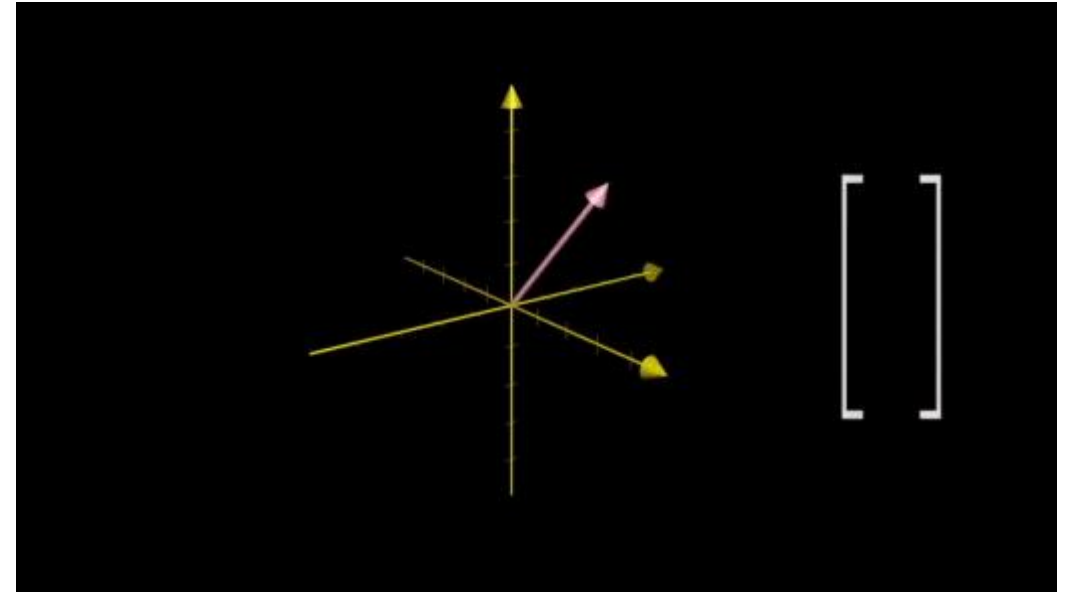
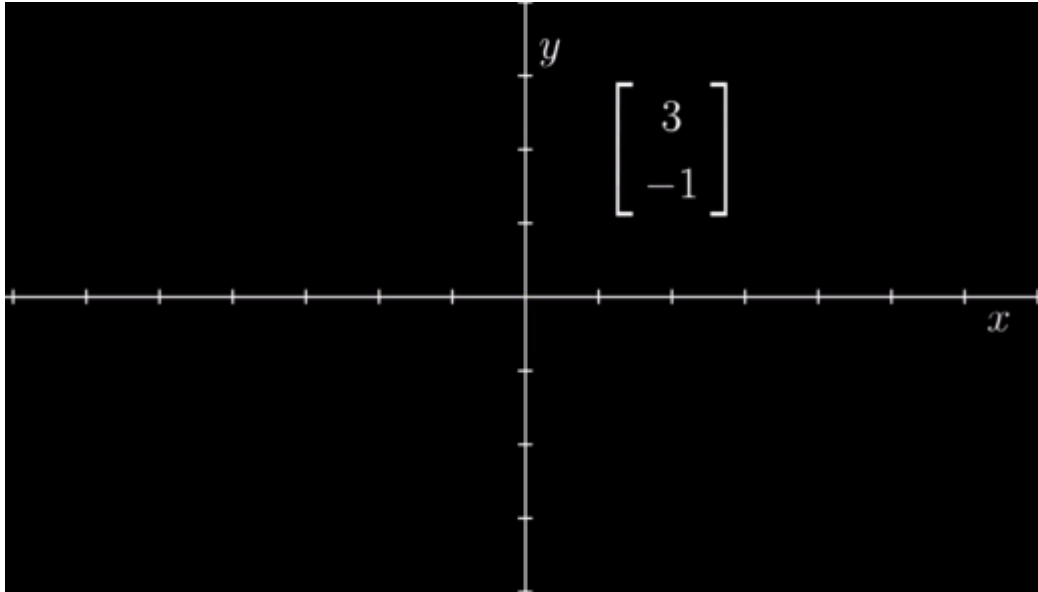
$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} : \text{열 벡터(Column Vector)}$$

$$X^T = [x_1, x_2, \dots, x_d] : \text{행 벡터(Row Vector)}$$

A diagram illustrating the relationship between column and row vectors and their dimension. A blue rectangular box labeled '벡터의 차원' (Dimension of Vector) is positioned at the bottom center. Two blue arrows originate from this box: one points upwards and to the left towards the column vector X , and the other points upwards and to the right towards the row vector X^T .

벡터의 차원

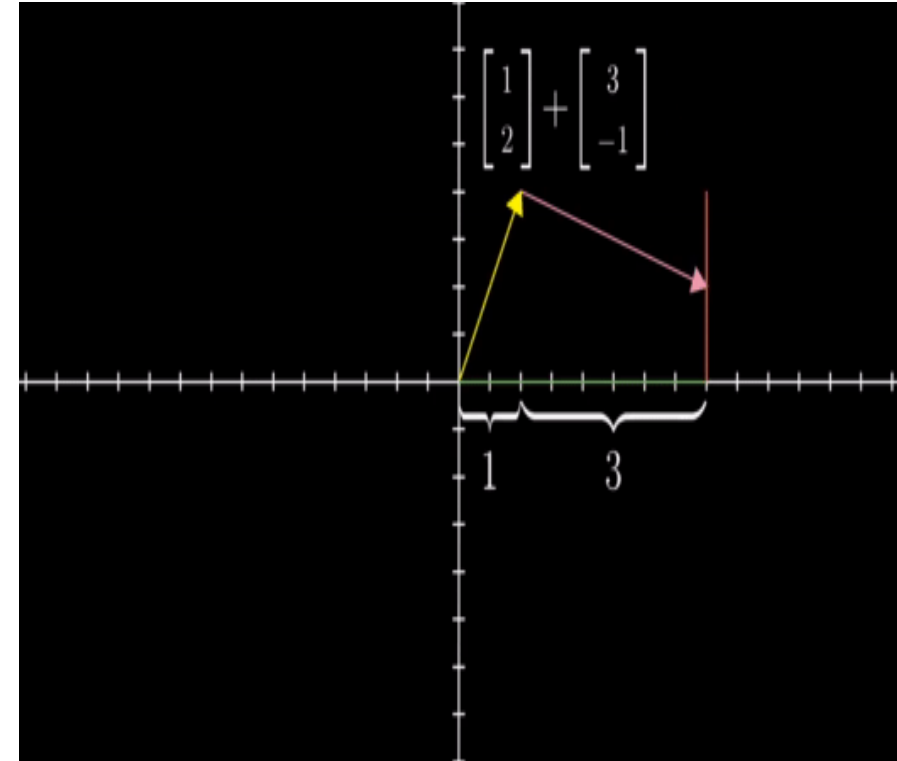
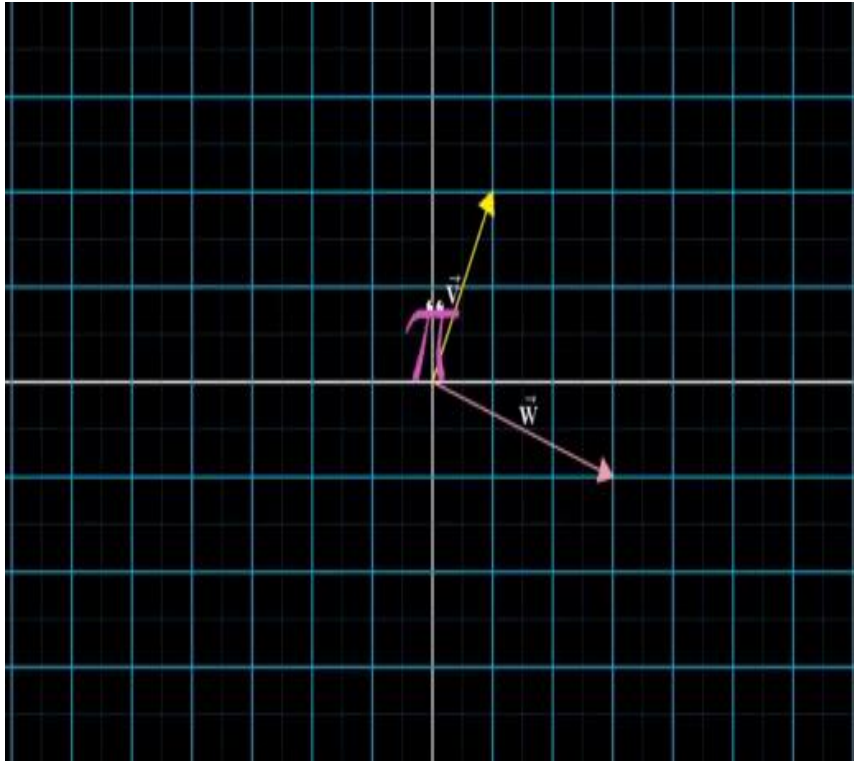
벡터(Vector)란?



평범한 좌표(1,2)와 벡터[1,2]를 구분하기 위해 관례적으로 열벡터(세로 컬럼)로 표기하는 것이 일반적.

벡터(Vector)란?

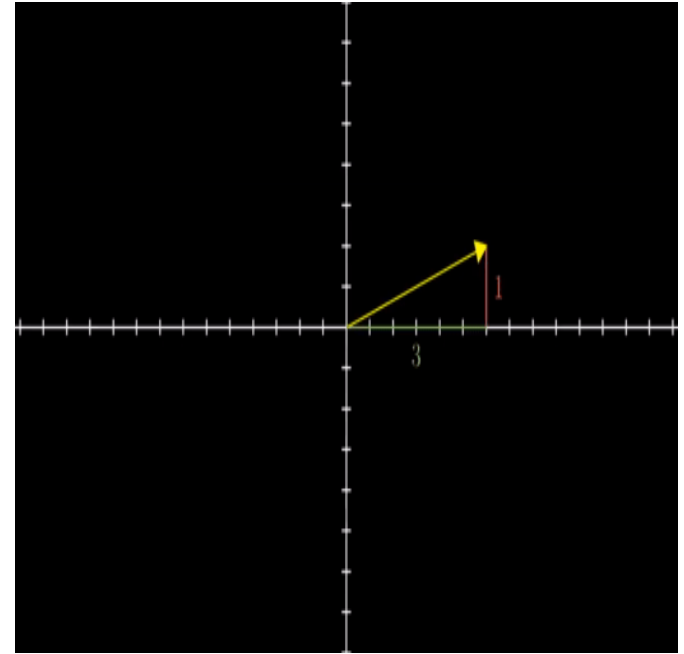
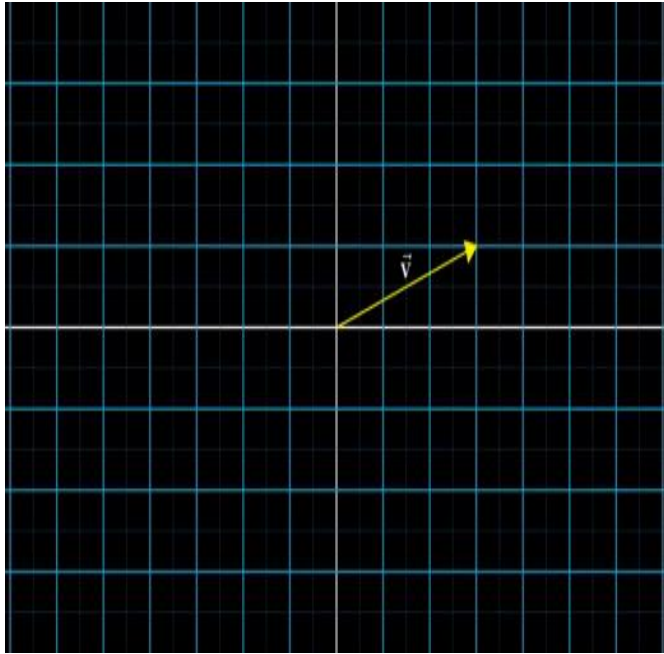
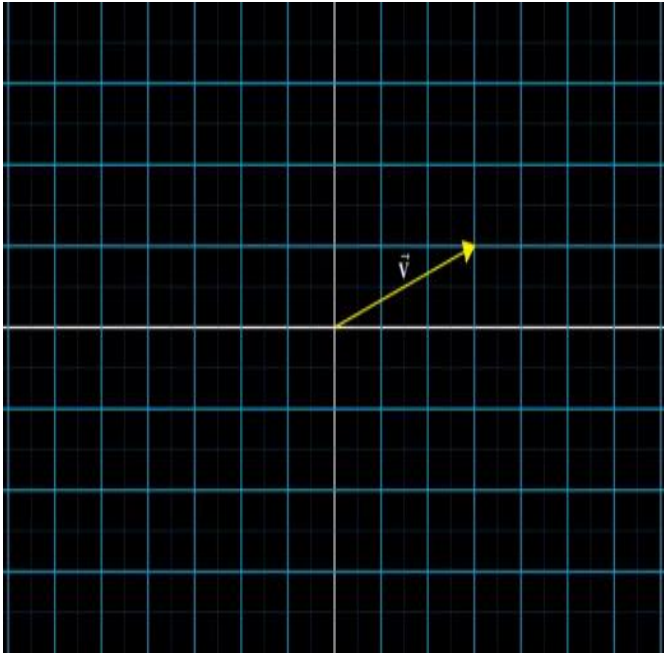
- 벡터 합) 두 벡터를 이어 붙이기.



- 두 벡터를 차례로 따라 이동한 벡터의 끝지점만 가리키는 벡터가 있다면, 그 벡터가 벡터의 합.
- 수치적으로 벡터의 합을 연산하려면, 각 차원(자리)의 값들을 더해주면 그 것이 결과 벡터로 나오게 됨.

벡터의 기본 연산

- 스칼라 곱 : 벡터를 상수배 만큼 늘리거나 곱하기



행렬(Matrix)

- 수 또는 변수 등의 일련의 개체들을 행(Row)과 열(Column)에 맞추어 직사각형 모형으로 순서 있게 배열하여 괄호 [] 로 묶은 것.
- 괄호 한의 개체와 같이 행렬을 구성하는 요소 보통 소문자 a_{ij}, b_{ij}, c_{ij} 등으로 나타냄(i행 j열의 원소)
- 행(Row) : 행렬의 가로줄
- 열(Column) : 행렬의 세로줄

$$A_{m \times n} = \begin{matrix} & \begin{matrix} 1\text{열} & 2\text{열} & & n\text{열} \end{matrix} \\ \begin{matrix} 1\text{행} \\ 2\text{행} \\ \vdots \\ m\text{행} \end{matrix} & \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \end{matrix} = (a_{ij})$$

행렬의 차원(Dimension)

- 행렬의 크기를 말함
- 행렬의 크기는 (행의 개수) (열의 개수)로 나타냄. 만약 행의 개수가 m 이고 열의 개수가 n 이면 $m \times n$ (차원) 행렬이라고 함.
- n 차 정방행렬
: 행과 열의 수가 같으면 정방행렬이라 하며, $m=n$ 이면 n 차 정방행렬

$$A_{m \times n} = \begin{matrix} & \begin{matrix} 1\text{열} & 2\text{열} & & n\text{열} \end{matrix} \\ \begin{matrix} 1\text{행} \\ 2\text{행} \\ \vdots \\ m\text{행} \end{matrix} & \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \end{matrix} = (a_{ij})$$

R 데이터 타입

- Class 함수 : 데이터 타입이 무엇인지 반환해줌.

문자형 타입 Character	숫자형 타입(계산가능)	논리형 타입 Logical	NaN, Na, Null
<ul style="list-style-type: none">• (쌍) 따옴표로 표시	<ul style="list-style-type: none">• 숫자형 Numeric<ul style="list-style-type: none">• 실수 double,• 정수 integer• 복소수 complex• 무한대 inf, -inf	<ul style="list-style-type: none">• 참, TRUE• 거짓 FALSE	<ul style="list-style-type: none">• NaN : 숫자가 아님(오류)• NA : 결측값(공간 차지 O)• NULL : 결측값(공간 차지 X)

```
1 install.packages('dslabs')
2 library(dslabs)
3 2+6
4 sqrt(5)
5 3*(pi/2)-1
6 x<-2+6
7 result<-sqrt(5)
8 result
```

```
> class(sqrt(5))
[1] "numeric"
> class(x)
[1] "numeric"
> class(result)
[1] "numeric"
```

R 데이터 타입

함수명	설명	입력내용	결과 내용
is.numeric()	수치형 여부	is.numeric(데이터)	TRUE or FALSE
is.integer()	정수형 여부	is.integer(데이터)	TRUE or FALSE
is.double()	실수형 여부	is.double(데이터)	TRUE or FALSE
is.character()	문자형 여부	is.character(데이터)	TRUE or FALSE
is.logical()	논리형 여부	is.logical(데이터)	TRUE or FALSE
is.complex()	복소수형 여부	is.complex(데이터)	TRUE or FALSE
is.null()	NULL 여부	is.null(데이터)	TRUE or FALSE
is.na()	NA 여부	is.na(데이터)	TRUE or FALSE
is.finite()	유한 수치 여부	is.finite(데이터)	TRUE or FALSE
is.infinite()	무한 수치 여부	is.infinite(데이터)	TRUE or FALSE

R 데이터 타입

함수명	설명	입력내용	결과 내용
as.numeric()	수치형으로 변환	as.numeric(데이터)	변환되거나 NA
as.integer()	정수형으로 변환	as.integer(데이터)	변환되거나 NA
as.double()	실수형으로 변환	as.double(데이터)	변환되거나 NA
as.character()	문자형으로 변환	as.character(데이터)	변환되거나 NA
as.logical()	논리형으로 변환	as.logical(데이터)	변환되거나 NA
as.complex()	복소수형으로 변환	as.complex(데이터)	변환되거나 NA

```
# Data Type 변환
```

```
var1 <- 3.1415
```

```
var2 <- 0
```

```
var3 <- "3.141592"
```

```
var4 <- "Hello"
```

```
as.character(var1)      # "3.1415"
```

```
as.double(var1)         # 3.1415
```

```
as.integer(var1)        # 3
```

```
as.numeric(var1)        # 3.1415
```

```
as.logical(var1)        # TRUE
```

```
as.logical(var2)        # FALSE
```

```
as.double(var3)         # 3.141592
```

```
as.integer(var3)        # 3
```

```
as.double(var4)         # NA
```

R 기초

변수 선언 및 사칙연산

scalar: R의 vector 자료구조의 한 유형으로
한개 값만 갖는 vector를 의미

```
a<-3
result<-a+2
result

result1<-a*3
result2<-a^2
result3<-a**2
```

print() : 1개의 데이터를 출력

```
print(result)
print(result1)
print(result2)
print(result3)
```

여러개의 데이터 출력

cat() : 출력 후 개행이 일어나지 않음. "\n"으로 개행 출력
cat('계산된 출력값은 : ', result)

```
cat('계산된 출력값들은 : ', result, result1)
print('계산된 출력값들은 : ', result, result1)
```

Environment에 있는 모든 객체 삭제

Console을 clear.

ls(), ls.str() : 변수 목록 보기.

rm(list=ls()) # Environment 객체 삭제

cat("\014")

```
> a<-3
> result<-a+2
> result
[1] 5
>
> result1<-a*3
> result2<-a^2
> result3<-a**2
>
> #print() : 1개의 데이터를 출력
> print(result)
[1] 5
> print(result1)
[1] 9
> print(result2)
[1] 9
> print(result3)
[1] 9
>
> #여러개의 데이터 출력
> #cat() : 출력 후 개행이 일어나지 않음. "\n"으로 개행 출력
> cat('계산된 출력값은 : ', result)
계산된 출력값은 : 5>
> cat('계산된 출력값들은 : ', result, result1)
계산된 출력값들은 : 5 9> print('계산된 출력값들은 : ', result, result1)
[1] "계산된 출력값들은 : "
```

R기초(1).R

R 연산자

대입 연산자	비교 연산자	산술 연산자	기타 연산자
<ul style="list-style-type: none">변수에 값을 할당	<ul style="list-style-type: none">변수, 숫자, 문자, 논리값 비교NA : 어떤 것과 비교해도 NA	<ul style="list-style-type: none">두 숫자형 타입의 계산	<ul style="list-style-type: none">논리값을 계산
<ul style="list-style-type: none"><code><-</code>, <code><<-</code>, <code>=</code>, <code>-></code>, <code>->></code>	<ul style="list-style-type: none"><code>==</code>, <code>></code>, <code>=></code>, <code><</code>, <code>=<</code><code>is.numeric</code>, <code>is.chararter</code>, <code>is.logical</code>, <code>is.na</code>, <code>is.null</code> ...	<ul style="list-style-type: none"><code>+</code>, <code>-</code>, <code>*</code>, <code>^</code>, <code>**</code>, <code>exp()</code><ul style="list-style-type: none"><code>%/%</code>(몫)<code>%%</code>(나머지)	<ul style="list-style-type: none">부정연산자 <code>!</code>: 반대되는 값AND 연산자 <code>&</code>: 둘다 참OR 연산자 <code> </code>: 둘중 하나 참

R 연산자

- 할당연산자(Assignment) : =

- Variables in C: Call-by-value

`int a = 1;`  Putting the value in a box with the variable name.

`a = 2;`  If you change the value of the variable, the box will be updated with the new value.

`int b = a;`  Assigning one variable to another makes a copy of the value and put that value in the new box.

- Names in Python: Call-by-object

`a = 1`  It tags the value with the variable name.

`a = 2`  It just changes the tag to the new value in memory
Garbage collection free the memory automaticall

`b = a`  It makes a new tag bound to the same value.

← 할당
X = 10
↑ 변수이름 ↑ 값

영문, 숫자 사용가능
대소문자 구분
숫자부터 시작할 수 없음.
특수문자 사용 불가(+, -, *, /, \$ 등)

R 연산자

```
a<-1
b<-3

result<-a/b
result
options(digits=10) #숫자를 몇 자리까지 출력할 것인가?
#default=7
result
#sprintf(): print와 유사하지만 주어진 인자들을 특정한 규칙에
#맞게 변환해 출력해주는 차이점이 존재
#%d : 부호 있는 십진법으로 나타난 정수
#%f : 십진법으로 나타낸 부동 소수점 수
#%s : 문자열
sprintf("%0.7f",a/3)
sprintf("결과값 : %f", a/3)

result1<- a %% b #몫 구하기 1/3 = 0.33333
result1

result2<- a %% b #나머지 구하기 1/3 = 0.3333
result2
```

```
> a<-1
> b<-3
>
> result<-a/b
> result
[1] 0.3333333333
> options(digits=10) #숫자를 몇 자리까지 출력할 것인가?
> #default=7
> result
[1] 0.3333333333
> #sprintf(): print와 유사하지만 주어진 인자들을 특정한 규칙에
> #맞게 변환해 출력해주는 차이점이 존재
> #%d : 부호 있는 십진법으로 나타난 정수
> #%f : 십진법으로 나타낸 부동 소수점 수
> #%s : 문자열
> sprintf("%0.7f",a/3)
[1] "0.3333333"
> sprintf("결과값 : %f", a/3)
[1] "결과값 : 0.333333"
>
> result1<- a %% b #몫 구하기 1/3 = 0.33333
> result1
[1] 0
>
> result2<- a %% b #나머지 구하기 1/3 = 0.3333
> result2
[1] 1
```

R기초(2).R

R 연산자

```
#비교 연산자 : 두 값의 비교로서 맞으면 TRUE
#맞지 않으면 FALSE를 반환
a<-100
b<-200

a==b
a!=b

a>b
a<b
!(a<=b)
#할당 연산자 : 변수에 값을 할당(저장)하는데 사용
#오른쪽에서 왼쪽으로 저장.
```

```
a=100
a<-300
```

```
b<-200
cat(a,b)
```

#논리 연산자 : 논리식을 판단하여, 참(true)과 거짓(false)를 반환.

```
# 조건에 있는 값이 scalar면 &와 &&가 동일처리
TRUE & FALSE      # FALSE
TRUE && FALSE      # FALSE
```

```
# 조건에 있는 값이 scalar면 |와 ||가 동일처리
TRUE | FALSE      # TRUE
TRUE || FALSE     # TRUE
```

```
# 조건에 있는 값이 vector이면
# &는 vector의 모든 조건에 대한 연산을 수행한 후
# 결과를 vector로 return
# &&는 vector의 첫번째 조건에 대한 연산을 수행한 후
# 결과를 scalar로 return
```

```
c(TRUE,FALSE) & c(TRUE,TRUE)    # TRUE FALSE
c(TRUE,FALSE) && c(TRUE,TRUE)    # TRUE
c(TRUE,FALSE) & c(TRUE,TRUE,FALSE) # Error
```

```
!c(TRUE,FALSE,TRUE)            # FALSE TRUE FALSE
```

```
> a==b
[1] FALSE
> a!=b
[1] TRUE
>
> a>b
[1] FALSE
> a<b
[1] TRUE
> !(a<=b)
[1] FALSE
> #할당 연산자 : 변수에 값을 할당(저장)하는데 사용
> #오른쪽에서 왼쪽으로 저장.
>
> a=100
> a<-300
>
> b<-200
> cat(a,b)
300 200>
> #논리연산자 : 논리식을 판단하여, 참(true)과 거짓(false)를 반환.
>
> # 조건에 있는 값이 scalar면 &와 &&가 동일처리
> TRUE & FALSE      # FALSE
[1] FALSE
> TRUE && FALSE      # FALSE
[1] FALSE
>
> # 조건에 있는 값이 scalar면 |와 ||가 동일처리
> TRUE | FALSE      # TRUE
[1] TRUE
> TRUE || FALSE     # TRUE
[1] TRUE
>
> # 조건에 있는 값이 vector이면
> # &는 vector의 모든 조건에 대한 연산을 수행한 후
> # 결과를 vector로 return
> # &&는 vector의 첫번째 조건에 대한 연산을 수행한 후
> # 결과를 scalar로 return
>
> c(TRUE,FALSE) & c(TRUE,TRUE)    # TRUE FALSE
[1] TRUE FALSE
> c(TRUE,FALSE) && c(TRUE,TRUE)    # TRUE
[1] TRUE
warning messages:
1: In c(TRUE, FALSE) && c(TRUE, TRUE) :
  'length(x) = 2 > 1' in coercion to 'logical(1)'
2: In c(TRUE, FALSE) && c(TRUE, TRUE) :
  'length(x) = 2 > 1' in coercion to 'logical(1)'
> c(TRUE,FALSE) & c(TRUE,TRUE,FALSE) # Error
[1] TRUE FALSE FALSE
warning message:
In c(TRUE, FALSE) & c(TRUE, TRUE, FALSE) :
  longer object length is not a multiple of shorter object length
>
> !c(TRUE,FALSE,TRUE)            # FALSE TRUE FALSE
[1] FALSE TRUE FALSE
```

R기초(3).R

R 연산자

함수명	설명	입력내용	결과 내용
abs()	절대값	abs(-3)	3
sqrt()	제곱근	sqrt(16)	16
pi	원주율	pi	3.141593
sign()	부호	sign(-3)	-1
round()	반올림	round(2.345, digits=2)	2.35
ceiling()	무조건 올림	ceiling(2.3)	3
floor()	무조건 내림	floor(2.7)	2
exp()	지수	exp(10)	22026.47
log()	자연로그	log(10)	2.302585
log10()	사용로그	log10(10)	1
log2()	로그(2)	log2(10)	3.321928
logb()	일반화 로그	logb(10, base=3)	2.095903
factorial()	계승	factorial(4)	24
choose()	조합	choose(4, 2)	6
prod()	곱	prod(1:4)	24
sin()	사인(sine)	sin(0.5)	0.4794255
cos()	코사인(cosine)	cos(0.5)	0.8775826
tan()	탄젠트(tangent)	tan(0.5)	0.5463025

R 연산자

기본함수	기능
help() or ?	함수들에 도움말
Paste(문자열1, 문자열2)	문자열을 이어붙임
seq(시작, 끝값, 간격)	수열 생성
rep(데이터, 반복횟수)	데이터를 반복
rm(변수)	대입연산자로 생성된 변수 삭제
ls()	현재 생성된 변수들의 리스트
print()	콘솔창에 값을 출력

통계함수	기능
sum()	합계
mean()	평균
median()	중앙값
var(), sd()	표본분산, 표준편차
max(), min()	최댓값, 최솟값
range()	최댓값과 최솟값(범위)
summary()	요약값

R 데이터 구조

벡터 : 하나의 스칼라값 또는 하나 이상의 스칼라 원소들을 갖는 단순 형태의 집합

- 원소는 숫자, 문자, 논리 연산자 등 → 숫자만은 숫자 벡터, 문자만은 문자 벡터
- 가장 단순 형태
- 명령어 c로 선언(c: combine (연결))
- TRUE는 1, FALSE는 0
- 대소문자 구분 → 논리 연산자는 반드시 대문자로
- '=', '<-' : 우측의 값을 좌측의 변수에 할당
- 벡터 + 벡터(문자형 벡터가 포함시 결과는 문자형 벡터)

```
x <- c(1, 10, 24) # 문자형 벡터
y <- c("사과", "바나나") # 문자형 벡터
z = c(TRUE, FALSE, TRUE) # 논리 연산자 벡터
xy <- c(x, y) # 새로운 벡터 형성, 결과는 문자형 벡터
```

R 데이터 구조

행렬 : $m \times n$ 형태의 데이터 나열

- 명령어 matrix로 선언
- 첫 번째 인수 : 들어가는 데이터들을 묶은 벡터
- ncols = columns 수, nrow = row 수
- 기본적으로 열을 우선 채우는 방향으로 / 행 우선 채우는 방향 : byrow=T
- 행렬+벡터, 벡터+벡터 = 행렬 : rbind(row, 행 추가) , cbind(column, 열 추가)
- 단, 데이터프레임을 서로 합치는 경우 결과는 데이터 프레임 형태

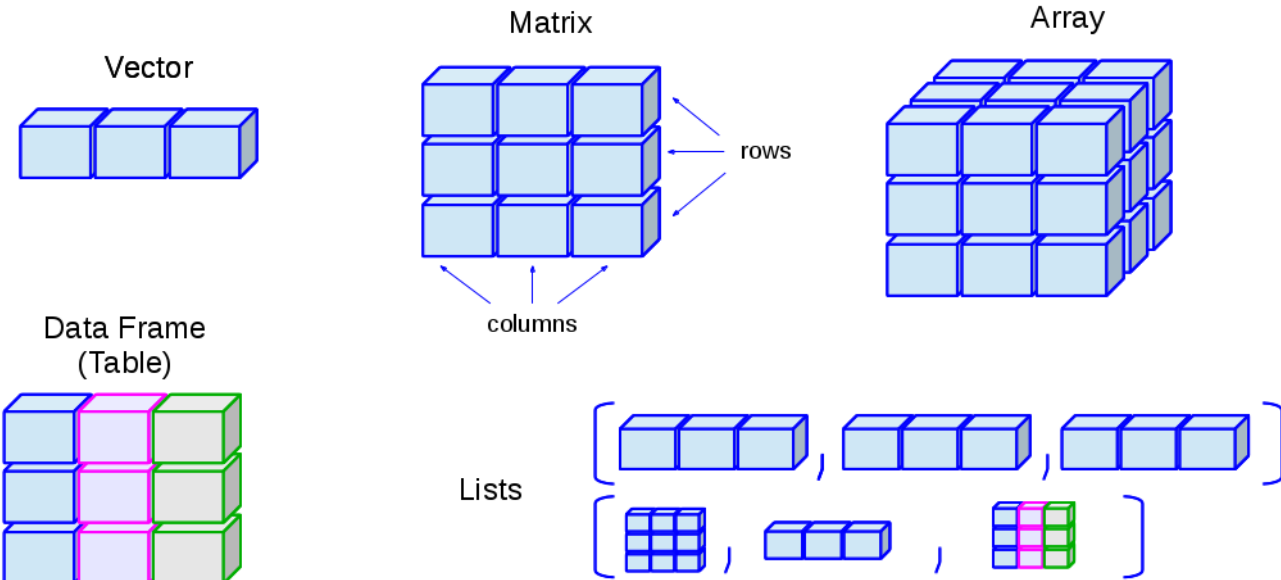
```
# 행렬 선언
```

```
mx = matrix(c(1,2,3,4,5), ncols=2) # 3행 2열 행렬
```

R 데이터 구조

데이터 프레임 : 행렬과 유사한 2차원 목록 데이터 구조

- 행렬과 달리, 각 열이 서로 다른 데이터 타입을 가질 수 있음.
- 데이터 크기가 커져도 다루기 수월
- 명령어 `data.frame` : 여러 벡터를 데이터 프레임으로 합치기
- 각 행 - 하나의 관측치, 각 열 - 하나의 변수



R 데이터 구조

```
# vector: 한가지 타입의 데이터를 한개 이상
# 저장할 수 있는 1차원 배열 형태의 데이터 타입
x <- 1 # 원소의 갯수가 1개인 vector
x <- 1:10 # 1씩 증가하는 연속적인 숫자
x

# 숫자들을 저장하는 벡터
num_vector <- c(1,3,4,10,15) # c(): combine 함수 0 매개변수들로 이루어진 벡터를 생성
num_vector[1]
num_vector[3] # R에서 인덱스는 1부터 시작
num_vector[4]

# 논리값(boolean)들을 저장하는 벡터
bool_vector <- c(TRUE, FALSE, FALSE, TRUE)
bool_vector[4]
text_vector <- c("one","two","three")
text_vector

v1 <- c(1,2,TRUE, FALSE)
v1[[2]]
v1[4]
```

R 데이터 구조

```
# vector 생성 - c()

var1 = c(10,30,77,100)
var1

var2 = c(89,56,33)
var2

var3 = c(TRUE,FALSE,FALSE,TRUE)
var3

var4 = c("홍길동", "강감찬", "유관순")
var4

var5 = c("홍길동", 100, TRUE, 3.141592)
var5

var6 = c(var1, var2)
var6

var7 = c(var1, var4)
var7
```

- Seq() : sequence의 약자로 콜론(:)의 일반형.
- 3개의 argument인 시작(from), 끝(to), 증감치(by)를 명시해서 규칙이 있는 수치형 vector를 생성할 수 있음.
- argument 이름을 사용하는 것이 readability 측면에서 좋음.

```
# vector 생성 - :

var1 = 1:5
var1

var2 = 5:0
var2

var3 = 3.3:10
var3
```

```
# vector 생성 - seq()

var1 = seq(from=1, to=5, by=1)
var1

var2 = seq(from=0.5, to=10, by=1.5)
var2

var3 = seq(from=10, to=5, by=-2)
var3
```

R 데이터 구조

- rep() : w지정하는 반복 횟수만큼 동일한 값이 복제되어 vector가 생성됨.

- Mode()함수와 is계열의 함수를 이용하면 vector의 Data Type을 알아 낼 수 있음.
- 데이터의 개수를 알아내기 위해서는 length() 함수를 이용.

```
# vector 생성 - rep() # replicate 함수
```

```
var1 = rep(1:3, times=3) # times 생략 가능
```

```
var1  
# 1 2 3 1 2 3 1 2 3
```

```
var2 = rep(1:3, each=3) # each는 각 원소가 반복할 횟수 지정
```

```
var2  
# 1 1 1 2 2 2 3 3 3
```

```
# vector의 Data Type
```

```
var1 = c(10,20,50,100)
```

```
var1
```

```
mode(var1) # numeric
```

```
is.character(var1) # FALSE
```

```
is.numeric(var1) # TRUE
```

```
is.integer(var1) # FALSE
```

```
is.double(var1) # TRUE
```

```
# vector의 개수 확인 - length()
```

```
var1 = seq(1,100,2)
```

```
var1
```

```
var2 = c(10,20,30)
```

```
var2
```

```
length(var1) # var1의 개수 - 50
```

```
length(var2) # var1의 개수 - 3
```

```
var3 = seq(1,100, length = 4) # length를 이용한 vector 생성
```

```
var3
```

R 데이터 구조

- vector 안에서 일부 데이터를 추출할 때는 대괄호([])를 이용
- Index를 이용하여 vector의 요소를 추출해서 사용할 수 있음.
- R에서는 index의 시작이 1
- Index부분에서는 위에서 설명한 c(), :, seq()등도 올 수 있음.

```
# vector 데이터 추출

var1 = c(67,90,80,50,100)
var1

var1[1]           # 67
var1[length(var1)] # 100
var1[2:4]         # 90 80 50
var1[c(1,2,5)]    # 67 90 100
var1[seq(2,4)]     # 90 80 50
var1[6]           # NA
var1[-1]          # 1번째를 제외한 나머지
                  # 90 80 50 100
var1[-(2:5)]       # 67
var1[-c(1,2,4,5)]  # 80
```

- names() 함수를 이용하면 vector의 원소 각각에 이름이 있으면 이름을 문자형 형태로 출력.
- Vector 각원소에 이름을 붙여주고 싶은 경우에는 c() 함수와 함께 사용하여 이름을 부여

```
# vector의 원소 이름
var1 = c(10,20,30)
var1

names(var1)      # NULL

names(var1) = c("국어", "영어", "수학")

names(var1)      # "국어" "영어" "수학"

var1             # 이름과 데이터 함께 출력

var1[1]          # index를 이용한 vector 원소 접근

var1["국어"]     # name을 이용한 vector 원소 접근
```


R 데이터 구조

- `rep(인수1, 인수2)` : 인수1을 인수2만큼 반복하는 숫자 벡터 생성
- `seq(인수1, 인수2)` : 인수1부터 인수2까지 1씩 증가하는 숫자 벡터 생성
 - `(=) 인수1:인수2`
 - `seq(인수1, 인수2, length=m)` : 전체 수열의 개수가 m개가 되도록 자동적으로 증가하는 수열 생성
- `rep(인수1:인수2, 3)` : 인수1부터 인수2까지 증가하는 수열이 3번 반복

```
v1 <- c(1,2,TRUE, FALSE)
v1[2]
v1[4]
v1

v2 <- c(123,"two",FALSE) # 문자열로 자동 형변환
v2
# R에서의 자동 형변환
# 유연성이 더 높은 타입으로 자동 형변환
# 유연성: 논리타입 < 정수타입 < 실수타입 < 문자열

v3 <- c(TRUE, 1, 1.1)
v3

v4 <- seq(1,5) # v4 <- 1:5
v4
v5 <- seq(1,10,2)
v5

v6 <- seq(10, 0, -2)
v6

v7 <- rep(1,10)
v7

v8 <- rep("☆☆",5)
v8

v9 <- seq(1,20)
v9[c(1,2)] # c9[1], c9[2]
v9[seq(2,10,2)] # v9[2], v9[4], v9[6], v9[8], v9[10]
```

R 데이터 구조

- 수치형 vector에 수치형 scalar를 이용하여 사칙연산을 할 수 있음.
- Vector의 길이가 동등할 경우 각 vector의 요소들끼리 연산을 수행함.
- Vector의 길이가 동등하지 않을 경우, 연산과정에서 데이터 개수가 적은 vector가 데이터의 개수가 많은 vector와 동일하게 데이터의 개수를 맞추게 됨.
- 데이터 개수가 차이나는 만큼 데이터의 개수가 늘어나게 됨.
- 새롭게 생성되는 데이터는 원래 가지고 있던 데이터를 순서대로 새롭게 생성되는 데이터에 지정하게 되는데 이러한 규칙을 **재사용 규칙(Recycling rule)**이라고 함.

vector간의 연산

```
var1 = 1:3
var2 = 4:6

var1      # 1 2 3
var2      # 4 5 6

var1 * 2   # 2 4 6
var1 + 10  # 11 12 13

var1 + var2 # 5 7 9

var3 = 1:6
var3

var1 + var3 # var1 : 1 2 3 1 2 3 (recycling rule)
             # var3 : 1 2 3 4 5 6
             # 2 4 6 5 7 9

var4 = 1:5
var4

var1 + var4 # var1 : 1 2 3 1 2 (recycling rule)
             # var4 : 1 2 3 4 5
             # 연산은 되지만 warning 발생
             # 2 4 6 5 7
```

vector의 집합연산

```
var1 = c(1,2,3,4,5)
var2 = seq(3,7)

union(var1,var2) # 합집합 : 1 2 3 4 5 6

intersect(var1,var2) # 교집합 : 3 4 5

setdiff(var1, var2) # 차집합 : 1 2
```

vector간의 비교연산

```
var1 = c("홍길동","김길동","최길동")
var2 = c("HONG","KIM","CHOI")
var3 = c("김길동","홍길동","김길동","최길동")

identical(var1,var3) # FALSE

setequal(var1,var3) # TRUE

var1 = 1:3
var2 = c(1:3)
var3 = c(1,2,3)

class(var1); class(var2); class(var3)

# data type이 다르다.
identical(var1,var2) # TRUE
identical(var1,var3) # FALSE
```

R 데이터 구조

#행렬 구조

```
m1<-matrix(c(1:6), nrow=3, ncol=2)
```

```
m1
```

```
m2<-matrix(c(1:6),nrow=3,ncol=2,byrow=TRUE)
```

```
m2
```

```
rnames<-c('행1','행2','행3')
```

```
cnames<-c('col1','col2')
```

```
r_c_names<-list(rnames,cnames)
```

```
m3<-matrix(c(1:6),nrow=3,ncol=2,byrow=FALSE,dimnames=r_c_names)
```

```
m3
```

```
m3[2,1]
```

```
m3[1,]
```

```
m3[[2,1]][1]
```

```
m3[1,][1]
```

```
> m3<-matrix(c(1:6),nrow=3,ncol=2,byrow=FALSE,dimnames=r_c_names)
```

```
> m3
```

```
      col1 col2
```

```
행1      1      4
```

```
행2      2      5
```

```
행3      3      6
```

```
> m3<-matrix(c(1:6),nrow=3,ncol=2,byrow=FALSE,dimnames=r_c_names)
```

```
> m3
```

```
      col1 col2
```

```
행1      1      4
```

```
행2      2      5
```

```
행3      3      6
```

```
> m3[2,1]
```

```
[1] 2
```

```
> m3[1,]
```

```
col1 col2
```

```
      1      4
```

```
> m3[[2,1]][1]
```

```
[1] 2
```

```
> m3[1,][1]
```

```
col1
```

```
      1
```

R 데이터 구조

```
# matrix 생성

var1 = matrix(c(1:5)) # 열을 기준으로 matrix 생성
var1                  # 5행 1열 matrix

# nrow 속성을 이용하여 지정된 행을 가지는 matrix 생성
# 열 기준으로 데이터가 채워진다.

var2 = matrix(c(1:10), nrow=2)
var2                  # 2행 5열의 matrix

# nrow 속성 사용시 만약 행과 열의 수가 일치하지 않는 경우

var3 = matrix(c(1:13), nrow=3)
var3

# matrix 생성 시 행 우선으로 데이터를 생성하는 경우

var4 = matrix(c(1:10), nrow=2, byrow=T)
var4

# vector를 대상으로 rbind()는 행 묶음으로 matrix를 생성
# vector를 대상으로 cbind()는 열 묶음으로 matrix를 생성

var5 = c(1,2,3,4)
var6 = c(5,6,7,8)

mat1 = rbind(var5, var6)
mat1

mat2 = cbind(var5, var6)
mat2

# 데이터 타입과 데이터 구조 확인
mode(mat1)           # numeric
class(mat1)           # matrix
```

```
# matrix의 원소 접근

var1 = matrix(1:21, nrow=3, ncol=7)
var1

var1[2,2]            # 2행 2열 : 5
var1[2,]             # 2행
var1[,3]             # 3열

var1[c(1,3), c(5:7)] # 1,3행 & 5~7열

length(var1)         # 모든 원소 개수 : 21
nrow(var1)           # 행 개수 : 3
ncol(var1)           # 열 개수 : 7

# matrix적용 함수 : apply()
# X : matrix, MARGIN : 1이면 행, 2면 열
# FUN : 행렬 자료구조에 적용할 함수

apply(X=var1, MARGIN=1, FUN=max) # 행단위 최대값
apply(X=var1, MARGIN=2, FUN=min) # 열단위 최소값
```

```
# array 생성

var1 = array(c(1:24), dim=c(3,2,4)) # 1~24의 데이터를 이용
                                     # 3행 2열 4면의 3차원 array 생성

var1
```

```
# matrix 연산

var1 = matrix(c(1:6), ncol=3)
var1

var2 = matrix(c(1,-1,2,-2,1,-1), ncol=3)
var2

var1*var2 # elementwise product(element단위의 곱연산)

t(var1)   # transpose matrix (전치행렬)

var3 = matrix(c(1,-1,2,-2,1,-1), ncol=2)
var3

var1 %*% var3 # matrix product (행렬곱)

# 역행렬 : matrix A가 nxn matrix일 때,
# 아래를 만족하는 nxn matrix B가 존재하면 B를 A의 역행렬이라 한다.
# AB = BA = I(단위행렬 E)
# 가우스 조던 소거법을 이용하여 계산.

var1 = matrix(c(1,2,3,3,0,1,5,4,2), ncol=3)
var1

solve(var1) # matrix inversion (역행렬)
```

R 데이터 구조_factor()

- **factor()** : 범주형 데이터를 표현하기 위한 데이터 형태. 범주형 자료로 표현되면 집단별로 통계 분석과 같은 작업이 수행 됨.
- **범주형 데이터** : 데이터가 사전에 정의된 특정 유형으로만 분류되는 경우를 의미.
Ex) 방의 크기를 “대“, “중“, ”소”로 나누어 표현하고 있을 때 특정 방의 크기를 “대”라고 명시한다면 이는 범주형 데이터.
범주형 변수가 담을 수 있는 값의 목록 (“대“, ”중“, ”소“)을 level이라고 함.
- Factor는 저장할 값 뿐만 아니라 level도 명시해야 됨. 범주형 데이터는 명목형과 순서형이 있음.



R 데이터 구조_factor()

- 입력 인자는 일반적으로 vector를 사용
- levels : 그룹으로 지정할 문자형 vector를 지정. 만약 사용하지 않으면 오름차순으로 데이터를 자체적으로 그룹지정 함.
- ordered : TRUE면 순서형, FALSE면 명목형 데이터를 의미. 기본값은 FALSE임. Level에 지정한 순서대로 값의 크기가 정해짐

```
# factor 생성

var1 = c("A", "O", "AB", "B", "A", "O", "A")
var1

var1_factor = factor(var1)

var1_factor           # factor이기 때문에
                      # data와 level이 같이 출력

nlevels(var1_factor)  # factor의 level개수
                      # 4

levels(var1_factor)   # factor의 level 목록
                      # "A" "AB" "B" "O"

is.factor(var1_factor) # factor인지를 판단

ordered(var1)          # 순서형 factor 생성

# level과 order 지정
# level에 지정이 안되면 NA로 처리
var1_factor = factor(var1,
                      levels = c("O", "A", "B"),
                      ordered = T)
var1_factor

var1_factor = factor(var1,
                      levels = c("O", "A", "B", "AB"),
                      ordered = T)
var1_factor

levels(var1_factor) = c("A형", "AB형", "B형", "O형")
levels(var1_factor)

var1_factor           # 결과를 꼭 확인해보자
```

```
# 남성과 여성의 데이터르 factor 생성 후 chart 그리기

gender = c("MAN", "WOMAN", "MAN", "MAN", "WOMAN")
gender

factor_gender = as.factor(gender)
factor_gender

table(factor_gender)  # 빈도수 구하기

plot(factor_gender)   # 빈도수로 막대그래프 생성
```

R 데이터 구조_list()

- list() : 일반적으로 통계분석의 결과를 저장할때 많이 사용하는 데이터 형태.
- list의 원소로 scalar, vector, matrix, array, data frame, factor, 또 다른 list를 가질 수 있음.
- vector와 유사한 선형 자료구조. list 하나의 메모리 영역에는 key, value가 한쌍으로 저장 됨. list는 순서가 존재. dict은 순서 존재 X

```
# list

var_scalar = 100 # scalar
var_scalar

var_vector = c(10,20,30) # vector
var_vector

var_matrix = matrix(1:4,nrow = 2,ncol = 2) # matrix
var_matrix

var_array = array(1:8, dim=c(2,2,2)) # array
var_array

var_df = data.frame(id=1:3, age=c(10,15,20)) # data frame
var_df

var_factor = factor(c("A","B","C","A","B","A")) # factor
var_factor

my_list = list(var_scalar,
               var_vector,
               var_matrix,
               var_array,
               var_df,
               var_factor)

my_list
```

```
# list 생성

myList <- list("Hong","길동",20)
myList

# list 자료구조를 vector 자료구조로 변환
myVector = unlist(myList)
myVector

# key와 value형식으로 list 생성
member = list(name=c("홍길동","김길동"),
              age=c(20,30),
              address=c("서울","수원"),
              gender=c("남자","여자"))

member

# list에서 key는 $를 이용해서 표시

member$age # 20 30

member$age[1] # 20

member[[1]] # "홍길동" "김길동"

member[["name"]] # "홍길동" "김길동"
member$name # "홍길동" "김길동"

member[2:3]
```

R 데이터 구조_dataframe()

- R에서 가장 많이 사용하는 데이터 형태로, 행과 열로 구성된 2차원 형태의 표를 지칭
- 특징
 - : 데이터베이스의 테이블 구조와 유사
 - : column 단위로 서로 다른 데이터의 저장이 가능
 - : list와 vector의 혼합형으로 column은 list, column내의 데이터는 vector 형태
- List에서 \$는 key를 의미하지만, data frame은 \$는 column을 의미.

```
# data frame 생성

# vector를 이용한 data frame 생성
no = c(1,2,3)
name = c("홍길동","최길동","김길동")
pay = c(250,150,300)

df = data.frame(NO=no,Name=name,Pay=pay)

df

# matrix를 이용한 data frame 생성
mat1 = matrix(data = c(1,"홍길동",150,
                        2,"최길동",150,
                        3,"김길동",300),
               nrow = 3,
               by=T)          # 행 우선

mat1

memp = data.frame(mat1)
memp

# 3개의 vector를 이용하여 data frame 생성
df = data.frame(x=c(1:5),
                y=seq(2,10,2),
                z=c("a","b","c","d","e"))

df

# data frame의 column을 참조하기 위해서는 $ 이용
df$x          # 1 2 3 4 5
```


R 데이터 구조_dataframe()

- str()함수는 data frame의 구조를 보여주는 함수

```
# str() 함수의 사용

df = data.frame(x=c(1:5),
                y=seq(2,10,2),
                z=c("a","b","c","d","e"))

str(df)

# 'data.frame': 5 obs. of 3 variables:
# $ x: int  1 2 3 4 5
# $ y: num  2 4 6 8 10
# $ z: Factor w/ 5 levels "a","b","c","d",..: 1 2 3 4 5

df = data.frame(x=c(1:5),
                y=seq(2,10,2),
                z=c("a","b","c","d","e"),
                stringsAsFactors = F)

df
str(df) # factor가 아닌 문자열 형태로 사용
```

```
> str(df) # factor가 아닌 문자열 형태로 사용
'data.frame': 5 obs. of 3 variables:
 $ x: int  1 2 3 4 5
 $ y: num  2 4 6 8 10
 $ z: chr  "a" "b" "c" "d" ...
```

- Summary() 함수는 data frame의 데이터를 대상으로 간단한 통계를 보여주는 함수

```
# summary() 함수의 사용

summary(df)
```

```
> summary(df)
      x      y      z
Min.   :1  Min.   : 2  Length:5
1st Qu.:2  1st Qu.: 4  Class :character
Median :3  Median : 6  Mode  :character
Mean    :3  Mean    : 6
3rd Qu.:4  3rd Qu.: 8
Max.    :5  Max.    :10
```

R 데이터 구조_dataframe()

- R에는 벡터, 행렬 또는 데이터 프레임에 임의의 함수를 적용한 결과를 얻기 위한 apply계열 함수가 있음.
- 이 함수들은 데이터 전체에 함수를 한 번에 적용하는 벡터 연산을 수행하므로 속도가 빠름.

함수	설명	특징
apply()	배열 또는 행렬에 주어진 함수를 적용한 뒤 그 결과를 벡터, 배열 또는 리스트로 반환	배열 또는 행렬에 적용
lapply()	벡터, 리스트 또는 표현식에 함수를 적용하여 그 결과를 리스트로 반환	결과가 리스트
sapply()	lapply()와 유사하지만 결과를 벡터, 행렬 또는 배열로 반환	결과가 벡터, 행렬 또는 배열
tapply()	벡터에 있는 데이터를 특정 기준에 따라 그룹으로 묶은 뒤 각 그룹마다 주어진 함수를 적용하고 그 결과를 반환	데이터를 그룹으로 묶은 뒤 함수를 적용
mapply()	sapply()의 확장된 버전으로 여러 개의 벡터 또는 리스트를 인자로 받아 함수에 각 데이터의 첫째 요소들을 적용한 결과, 둘째 요소들을 적용한 결과 등을 반환	여러 데이터를 함수의 인자로 적용

R 데이터 구조_dataframe()

```
d <- matrix(1:9, ncol = 3)
```

```
#apply example
```

```
# 행별로 합  
apply(d, MARGIN = 1, sum)  
rowSums(d)  
#MARGIN으로 행과 열을 선택
```

```
# 열별로 합  
apply(d, MARGIN = 2, sum)  
colSums(d)
```

```
#lapply() : list로 반환됨.  
d <- c(1, 2, 3)  
result <- lapply(d, function(x) {x * 2})
```

```
#결과를 vector로 바꾸고 싶다면,  
unlist(result)  
|  
x <- list(a = 1:3, b = 4:6)  
lapply(x, mean)
```

```
#sapply : 결과를 matrix, vector 등의 데이터 타입으로 반환  
lapply(iris[, 1:4], mean)
```

```
sapply(iris[, 1:4], mean)
```

```
class(sapply(iris[, 1:4], mean))
```

```
#lapply()와 다르게 sapply()는 numeric으로 결과를 반환.  
#이를 역시 dataframe으로 바꾸는 것이 가능.  
x <- sapply(iris[, 1:4], mean)  
as.data.frame(x)
```

```
#위에서 봤던 모양으로 바꿀려면 전치행렬로 바꿔야함.  
as.data.frame(t(x))
```

```
#많은 컬럼을 포함하는 데이터 프레임을 볼 때 각 컬럼의 데이터 타입을 보는 방법.  
sapply(iris, class)
```

```
str(iris)
```

```
#tapply() : 그룹별로 function을 적용하기 위해 사용.  
tapply(X = iris$Sepal.Length, INDEX = iris$Species, mean)  
#3가지 종이 존재하고 각각의 sepal length에 대한 평균을 확인할 수 있습니다.  
#이번에는 조금 더 복잡한 그룹화를 해봅시다.
```

```
m <- matrix(11:18, ncol = 2, dimnames = list(c("spring", "summer", "fall", "winter"),  
                                              c("male", "female")))
```

```
m  
#4x2의 행렬이 만들어졌습니다. 여기서 반기별 성별 셀의 합을 구할 겁니다.
```

```
#다시 말해, 봄, 여름의 남성과 여성 셀 각각의 합 그리고 가을, 겨울의 남성과 여성 셀 각각의 합을 구합니다.
```

```
#상기 행렬에서 INDEX를 지정 시, (n, m)에서 n을 먼저 나열한 뒤 m 값을 나열하면 됩니다.
```

```
#일단 행부터 묶는 것이고 spring, summer는 첫 번째 그룹이니 1을 할당하고, fall, winter는 2를 할당합니다.
```

```
#열을 묶을 때는 male을 첫 번째 그룹으로 1에 할당하고 female은 2에 할당합니다.
```

```
#이것을 코드로 표현하면 아래와 같습니다.
```

```
#mapply() : mapply()는 다수의 인자를 받아 처리하는 함수가 있고 함수에 넘겨줄 인자들이 데이터로 저장되어 있을 때, 데이터에 저장된 값들을 인자로 하여 함수를 호출합니다.
```

```
#rnorm()을 사용하여 정규분포를 따르는 난수를 발생시킨 뒤 mapply()에 적용시켜 보겠습니다.
```

```
#일단, rnorm(n = 1, mean = 0, sd = 1) / rnorm(n = 2, mean = 10, sd = 1) / rnorm(n = 3, mean = 100, sd = 1) 세 개의 값들을 호출해야 되는 상황이라고 가정해 봅시다.
```

```
#첫 번째 방법은 위의 함수 3개를 각각 호출하는 것입니다.
```

```
rnorm(n = 1, mean = 0, sd = 1)  
rnorm(n = 2, mean = 10, sd = 1)  
rnorm(n = 3, mean = 100, sd = 1)  
mapply(rnorm, c(1, 2, 3), c(0, 10, 100), c(1, 1, 1))
```

R 데이터 구조

벡터 Vector	행렬 Matrix	배열 Array	리스트 List	데이터프레임
<ul style="list-style-type: none"> 1차원 데이터 구조 같은 타입 여러 데이터 1개의 행으로 저장 c를 사용하여 묶음 	<ul style="list-style-type: none"> 2차원 구조의 벡터 같은 타입의 여러 데이터 다른 타입 -> 자동변환 <ul style="list-style-type: none"> Matrix or dim 	<ul style="list-style-type: none"> 3차원 이상의 벡터 같은 타입 여러 데이터 몇 차원의 구조인지 dim 옵션에 명시 	<ul style="list-style-type: none"> 데이터 타입, 구조와 상관없이 모든 것을 저장 가능한 자료 구조 (성분간 이질적인 특징) 	<ul style="list-style-type: none"> 2차원 구조 관계형 데이터구조 여러 개의 벡터 얼마다 다른 타입 가능
<ul style="list-style-type: none"> coma c(1,2,3) colon c(1:5) 	<ul style="list-style-type: none"> Matrix(c(1:6), nrow=2) dim(v1)<-c(2,3) 	<ul style="list-style-type: none"> a1<-array((c(1:12),dim=c(2,3,2))) 	<ul style="list-style-type: none"> L1<-list() L1[[1]]<-10 L1[[2]]<- "Test" 	<ul style="list-style-type: none"> V1<-c(1,2,3) V2<-c('A','B','C') df<-data.frame(v1,v2)

문자열 추출

- 빅데이터를 처리하기 위해서 필요한 문자열을 자르거나 교체 혹은 추출하는 작업은 항상 빈번하게 발생함.
- 문자열을 효과적으로 처리하기 위한 패키지인 **stringr** 을 설치하고 제공하는 함수를 알아보자.
- `str_length()` : 문자열 길이 반환
- `str_c()` : 문자열 연결, `str_join()`의 개선형
- `str_sub()` : 범위에 해당하는 부분 문자열 생성
- `str_split()` : 구분자를 기준으로 문자열을 분리하여 부분 문자열 생성
- `str_replace()` : 기존 문자열을 특정 문자열로 교체
- `str_extract()` : 문자열에서 특정 문자열 패턴의 첫번째 문자열 추출
- `str_extract_all()` : 문자열에서 특정 문자열 패턴의 모든 문자열 추출
- `str_locate()` : 문자열에서 특정 문자열 패턴의 첫번째 위치 찾기
- `str_locate_all()` : 문자열에서 특정 문자열 패턴의 모든 위치 찾기

문자열 추출

- 문자열 처리를 쉽고 간단하게 하기 위해서는 정규표현식 사용이 필수.
- 정규표현식은 약속된 기호들에 의해서 표현

표현식	의미
<code>^x</code>	문자열의 시작을 표현하며 x 문자로 시작됨을 의미한다.
<code>x\$</code>	문자열의 종료를 표현하며 x 문자로 종료됨을 의미한다.
<code>.x</code>	임의의 한 문자의 자리수를 표현하며 문자열이 x 로 끝난다는 것을 의미한다.
<code>x+</code>	반복을 표현하며 x 문자가 한번 이상 반복됨을 의미한다.
<code>x?</code>	존재여부를 표현하며 x 문자가 존재할 수도, 존재하지 않을 수도 있음을 의미한다.
<code>x*</code>	반복여부를 표현하며 x 문자가 0번 또는 그 이상 반복됨을 의미한다.
<code>x y</code>	or 를 표현하며 x 또는 y 문자가 존재함을 의미한다.
<code>(x)</code>	그룹을 표현하며 x 를 그룹으로 처리함을 의미한다.
<code>(x)(y)</code>	그룹들의 집합을 표현하며 앞에서 부터 순서대로 번호를 부여하여 관리하고 x, y 는 각 그룹의 데이터로 관리된다.
<code>(x)(?:y)</code>	그룹들의 집합에 대한 예외를 표현하며 그룹 집합으로 관리되지 않음을 의미한다.
<code>x{n}</code>	반복을 표현하며 x 문자가 n번 반복됨을 의미한다.
<code>x{n,}</code>	반복을 표현하며 x 문자가 n번 이상 반복됨을 의미한다.
<code>x{n,m}</code>	반복을 표현하며 x 문자가 최소 n번 이상 최대 m 번 이하로 반복됨을 의미한다.

```
# 패키지 설치와 로드

install.packages('stringr')
library("stringr")

# 문자열 길이와 위치

myStr <- "Hongkd1051Leess1002YOU25홍길동2005"

str_length(myStr)      # 31

str_locate(myStr,"홍길동")

# 부분문자열

str_sub(myStr,1,str_length(myStr)-7)

# 대소문자 변경

str_to_upper(myStr)

str_to_lower(myStr)

# 문자열 교체, 결합, 분리

myStr <- "Hongkd1051,Leess1002,YOU25,홍길동2005"
str_replace(myStr,"Hong","KIM")

str_c(myStr,"", "이순신2019")

str_split(myStr,",")

str1 <- c("홍길동","김길동","이순신","강감찬")
paste(str1,collapse=",")
```

```
myStr <- "Hongkd1051,Leess1002,YOU25,홍길동2005"

str_extract_all(myStr,"[a-z]{3}") # 영문소문자 연속 3문자 추출
str_extract_all(myStr,"[a-z]{3,}") # 영문소문자 연속 3문자 이상 추출
str_extract_all(myStr,"[a-z]{3,5}") # 영문소문자 연속 3~5문자 추출

str_extract_all(myStr,"Hong") # 해당 문자열 추출

str_extract_all(myStr,"[가-힣]{3}") # 연속된 3개의 한글 문자 추출
str_extract_all(myStr,"[0-9]{4}") # 연속된 4개의 숫자문자 추출

str_extract_all(myStr,"[^a-z]") # 영문 소문자 제외한 나머지 추출
str_extract_all(myStr,"[^가-힣]{5}") # 한글을 제외한 나머지 연속된 5개 추출
str_extract_all(myStr,"[^0-9]{3}") # 숫자를 제외한 나머지 연속된 3개 추출

myId <- "901010-1000432"

str_extract_all(myId,"[0-9]{6}-[1234]{0-9}{6}") # 주민번호 검사
str_extract_all(myId,"\\d{6}-[1234]\\d{6}")

myStr <- "Hongkd1051,Leess1002,YOU25,홍길동2005"

str_extract_all(myStr,"\\w{6}") # \w는 한글, 영문자, 숫자문자를 포함하지만 특수문자는 제외
```

외부 데이터 불러오기

csv(Comma Separated Values) 파일 불러오기

- 명령어 `read.table` : csv 파일을 데이터 프레임 형태로 불러오기
- `Header = T` : csv 파일 첫 줄을 변수명으로 지정
- `Sep = “,”` 쉼표로 구분된 데이터 파일(csv 파일)임을 지정
- 파일 경로에 \대신 \\ 이용 or / 사용가능
- 명령어 `read.csv` 사용 가능 : `sep = “,”`으로 구분자 명시 X

Txt 파일 불러오기

- 명령어 `read.table` : txt 파일을 데이터 프레임 형태로
- `Sep = ‘,’` X->txt 파일로 불러오기 가능
- 구분자나 형식 등을 잘 지켜야 함

외부 데이터 불러오기

엑셀 파일(xls/ xlsx)불러오기

- 엑셀 파일 엑셀로 실행, csv 파일로 저장, csv 파일 불러오기 방법 사용
- 패키지 설치 : RODBC

```
> library(RODBC) # 패키지 열기
> new <- odbcConnectExcel("c:\\data\\mydata") # 엑셀파일의 경로 입력 / 확장자 생략하고 작성
> youdata <- sqlFetch(new, "Sheet1") # 엑셀파일의 워크시트(sheet) 이름 입력 / 대소문자 구분
> close(new)
```


R 데이터 핸들링

데이터 이름 변경	데이터 추출	데이터 결합
<ul style="list-style-type: none">2차원 이상의 데이터 구조는 행, 열의 이름 지정 가능	<ul style="list-style-type: none">인덱싱 지원<code>[]</code> : 행, 열 이름으로 데이터 추출	<ul style="list-style-type: none"><code>rbind</code> : 행으로 결합<code>cbind</code> : 열로 결합
<ul style="list-style-type: none"><code>colnames(m1) <- c('A','B','C')</code><code>rownames(m1) <- c('r1','r2')</code>	<ul style="list-style-type: none"><code>df1[2]</code>, <code>df1['A']</code><code>df1 \$ B</code>(원하는 열의 데이터 추출)	<ul style="list-style-type: none">행의 수, 열의 수 같아야 결합 가능벡터 - 벡터 결합은 재사용 규칙 적용

R 데이터 구조

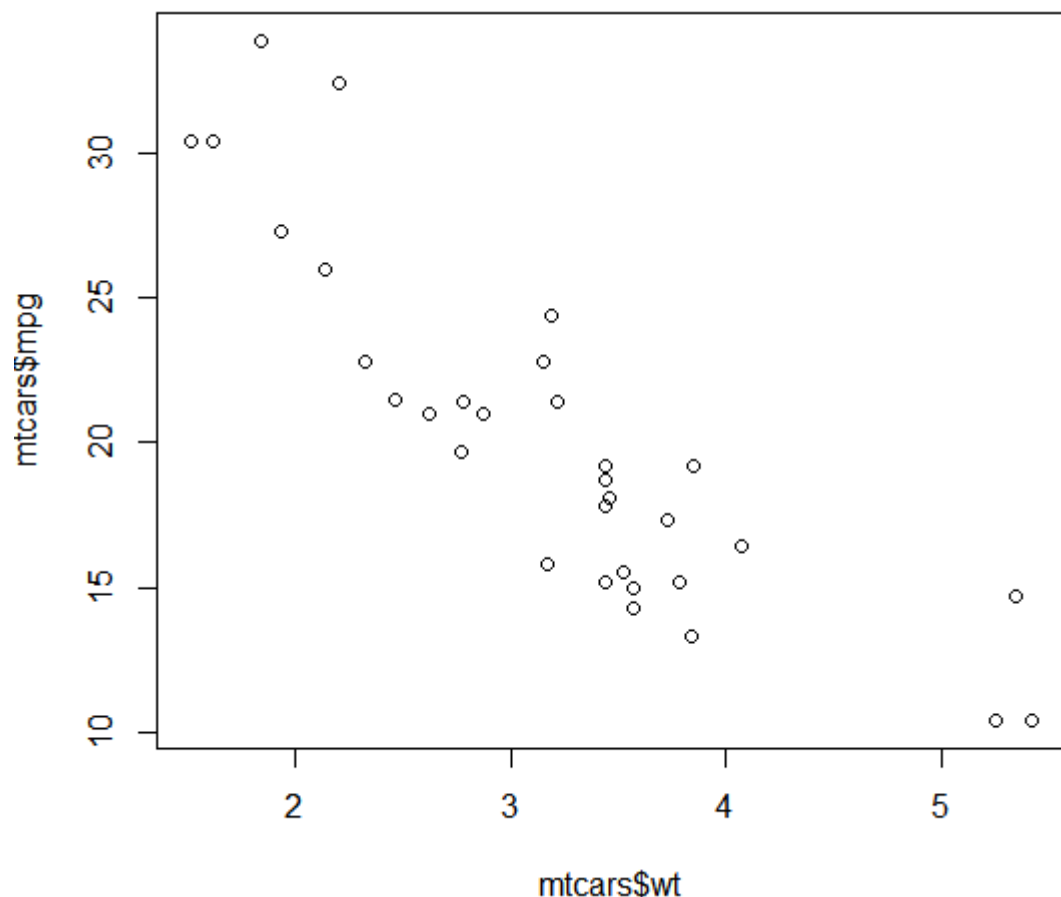
```
patient_id<-1:4
patient_id
age<-c(25,34,28,52)
age
type<-c("Type1","Type2","Type2","Type1")
type
status<-c("poor","Improved","Ecellent","poor")
status
df_patient<-data.frame(patient_id,age,type,status)
df_patient
str(df_patient)
head(df_patient,n=2)
tail(df_patient,n=2)
str(mtcars)
mtcars
summary(mtcars)

#dataframe에서 특정 컬럼(변수)의 내용만 확인
df_patient$patient_id
df_patient$age

mtcars$hp#벡터(vector)
mtcars["hp"]
mtcars[[c("mpg","cyl","wt")]]

head(mtcars,n=2)
mtcars["MazdaRX4",]

plot(mtcars$wt,mtcars$mpg)
```



제어문(반복문, 조건문(if-else), 사용자 정의함수

- 조건문 : 특정 조건을 만족할 때만 코드를 수행하는 문법
- 반복문 : 반복되는 작업을 간단하게 처리하기 위해 사용됨.

1. if 문

: 조건문에서 if문의 장점은 **문장 출력**과 다른 **명령어 수행**을 할 수 있다는 점.

: 단점은 **벡터 연산이 불가**하고 **오로지 하나의 조건에 대해서만 검사**가 가능하다는 점.

```
if ( 조건 ) {  
  (조건이 True일 때 실행될) 문장 또는 명령어  
} else {  
  (조건이 False일 때 실행될) 문장 또는 명령어  
}
```

```
> grade <- 75  
> if (grade >= 70) {  
+   print('합격')  
+ } else {  
+   print('불합격')  
+ }  
[1] "합격"
```

```
> vec1 <- c(10,20,30)  
> if (vec1 == 10) {  
+   print('인사부')  
+ } else {  
+   print('총무부')  
+ }
```

[1] "인사부"

결과값이 인사부인 이유는 첫 번째 요소인 10에 대해서만 조건 치환을 적용하기 때문

Warning message:

In if (vec1 == 10) { :

the condition has length > 1 and only the first element will be used # if문은 벡터 연산 불가

제어문(반복문, 조건문(if-else), 사용자 정의함수

- 조건문 : 특정 조건을 만족할 때만 코드를 수행하는 문법
- 반복문 : 반복되는 작업을 간단하게 처리하기 위해 사용됨.

2. if문의 else if구문

: 조건문에서 **else if**문은 if문과 동일하게 **문장 출력**과 다른 **명령어를 수행**하고, 장점은 **여러 조건에 대해서 검사**가 가능.

: 단점은 if문과 같이 **벡터 연산이 불가**하다는 점.

```
if ( 조건 1 ) {  
  ('조건1'일 때 실행될) 문장 또는 명령어  
} else if ( 조건 2 ){  
  ('조건1'이 아니고 '조건2'일 때 실행될) 문장 또는 명령어  
} else {  
  ('조건1'도, '조건2'도 아닐 때 실행될) 문장 또는 명령어  
}
```

```
> grade <- 'A'  
> if (grade == 'A') {  
+   print('합격')  
+ } else if (grade == 'B') {  
+   print('보류')  
+ } else {  
+   print('불합격')  
+ }  
[1] "합격"
```

```
> vec1 <- c(10,20,30)  
> if (vec1 == 10) {  
+   print('인사부')  
+ } else if (vec1 == 20) {  
+   print('재무부')  
+ } else {  
+   print('총무부')  
+ }
```

[1] "인사부"

결과값이 인사부인 이유는 첫 번째 요소인 10에 대해서만 조건 치환을 적용하기 때문

Warning message:

In if (vec1 == 10) { :

the condition has length > 1 and only the first element will be used # else if문은 벡터 연산 불가

제어문(반복문, 조건문(if-else), 사용자 정의함수

- 조건문 : 특정 조건을 만족할 때만 코드를 수행하는 문법
- 반복문 : 반복되는 작업을 간단하게 처리하기 위해 사용됨.

3. if else문

: 조건문에서 **ifelse문**의 장점은 if문의 한계를 해결하여 **벡터 연산(각 요소별 조건 검사)**이 가능.

: 단점은 주어진 값에 따라 yes or no를 반환해주고, **리턴값만 반환**하기 때문에 **오직 출력만 가능**하고 **조건별 명령어 수행을 불가**.

```
> vec1 <- c(10,20,30)
> ifelse (vec1 == 10, '인사부', '총무부')    # in oracle : decode(vec1,10, '인사부', '총무부')
[1] "인사부" "총무부" "총무부"
```

ifelse 는 중복사용도 가능합니다.

```
> ifelse (vec1 == 10, '인사부',
+        ifelse (vec1 == 20, '재무부', '총무부'))
[1] "인사부" "재무부" "총무부"
```

제어문(반복문, 조건문(if-else), 사용자 정의함수

- 조건문 : 특정 조건을 만족할 때만 코드를 수행하는 문법
- 반복문 : 반복되는 작업을 간단하게 처리하기 위해 사용됨.

4. for문

: for 문은 반복 횟수를 정할 수 있음.

```
for(반복변수 in 횟수) {  
  반복할 식  
}
```

```
> for (i in c(1:5)) {    # 숫자의 개수만큼 벡터의 요소를 i에 넣고 출력  
+   print(i)  
+ }  
[1] 1  
[1] 2  
[1] 3  
[1] 4  
[1] 5  
> for (i in c('a','b','c')) { # 문자의 개수만큼 벡터의 요소를 i에 넣고 출력  
+   print(i)  
+ }  
[1] "a"  
[1] "b"  
[1] "c"  
> for (i in c('a','b','c')) { # 문자의 개수만큼 반복  
+   print(10)  
+ }  
[1] 10  
[1] 10  
[1] 10
```

```
> for (i in emp$SAL) {    # emp테이블의 SAL column의 row 개수만큼 반복(반복 대상을 반복변수에 하나씩 넣으면서 반복)  
+   print(i * 0.1)        # 반복 변수를 재사용해서 반복문을 완성  
+ }  
[1] 80  
[1] 160  
[1] 125  
[1] 297.5  
[1] 125  
[1] 285  
[1] 245  
[1] 300  
[1] 500  
[1] 150  
[1] 110  
[1] 95  
[1] 300  
[1] 130  
> vec1 <- c(10,20,30)  
> for (i in vec1){        # for문과 if문의 조합  
+   if (i == 10) {  
+     print('인사부')  
+   } else {  
+     print('총무부')  
+   }  
+ }  
[1] "인사부"  
[1] "총무부"  
[1] "총무부"
```

제어문(반복문, 조건문(if-else), 사용자 정의함수

- 조건문 : 특정 조건을 만족할 때만 코드를 수행하는 문법
- 반복문 : 반복되는 작업을 간단하게 처리하기 위해 사용됨.

4. while문

: for문과는 다르게 시작 값의 정의와 증가시킬 값(반복변수 증가 구문)이 필요.

```
var <- 시작 값 정의
while(조건) {
  반복할 식
  증가 구문
}
```

```
> i <- 1
> while( i <= 5 ) { # 특정 조건이 맞을 때 까지 반복, 항상 True면 무한 루프에 빠지므로 주의
+   print(i)
+   i <- i + 1
+ }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

제어문(반복문, 조건문(if-else), 사용자 정의함수

- 조건문 : 특정 조건을 만족할 때만 코드를 수행하는 문법
- 반복문 : 반복되는 작업을 간단하게 처리하기 위해 사용됨.

5. next()

: 현재 수행 중인 반복문 블록을 중단하고, 다음 반복을 시작.

: 특정 조건에서 반복할 식을 넘어가고 싶다면, next 사용.

특정 조건이 성립할 때 뒤의 명령어를 수행하지 않고(뒤는 무시), while문의 처음으로 돌아감.

```
> i <- 1
> while(i <= 10) {
+   i <- i + 1
+   if(i %% 2 != 0) { # i가 홀수일 경우 뒤의 명령어(print(i))를 무시하고 while문의 처음(i <- i + 1)으로 돌아갑니다.
+     next
+   }
+   print(i)
+ }
[1] 2
[1] 4
[1] 6
[1] 8
[1] 10
```


제어문(반복문, 조건문(if-else), 사용자 정의함수

- 조건문 : 특정 조건을 만족할 때만 코드를 수행하는 문법
- 반복문 : 반복되는 작업을 간단하게 처리하기 위해 사용됨.

6. repeat()

: 블록 안의 문장을 반복해서 수행하다가, 특정 상황에 종료할 때 사용.

: for문, while문에 비해 잘 사용하지 않고, 특정 조건에 맞을 때 break를 사용하여 반복문을 벗어나려는 목적으로 사용.

```
repeat {  
  반복할 식  
}
```

```
> i <- 1  
> repeat {  
+ print(i)  
+ i <- i + 1  
+ if(i >= 5) { # i가 5이상일 경우 반복문 종료  
+ break  
+ }  
+}  
[1] 1  
[1] 2  
[1] 3  
[1] 4
```

제어문(반복문, 조건문(if-else), 사용자 정의함수

1. next 문장 수행 시

next는 현재 수행 중인 반복문 블록을 중단하고, 다음 반복을 시작

```
for ( i in 1:10) {  
  cmd1      # i = 1~4 까지 cmd1,2,3 계속 실행.  
  cmd2  
  if( 1 == 5 ) {    # i가 5일 때만, cmd3 생략. 다음 반복을 시작  
    next  
  }  
  cmd3  
}  
cmd4      # 반복문이 종료되면 정상 실행
```

2. break 문장 수행 시

break는 반복문을 벗어나려는 목적

```
for ( i in 1:10) {  
  cmd1  
  cmd2  
  if( 1 == 5 ) {    # i가 5가 되는 순간 반복문 종료, 반복문 내 모든 명령어 무시  
    break  
  }  
  cmd3  
}  
cmd4      # 반복문이 종료되면 정상 실행
```

3. exit 문장 수행 시

exit는 강제 프로그램 종료(엄청난 interrupt 발생 시)

exit의 숫자에 의미는 없고, 종료 코드에 따른 연속적인 작업을 수행 시 개발자가 설정

```
for ( i in 1:10) {  
  cmd1  
  cmd2  
  if( 1 == 5 ) {    # i가 5가 되는 순간 해당 프로그램 즉시 종료, 프로그램 내 모든 명령어 무시  
    exit 0  
  }  
  cmd3  
}  
cmd4      # exit를 만나면 프로그램이 종료되므로 실행되지 않음
```

제어문(반복문, 조건문(if-else), 사용자 정의함수

- 함수는 코드의 집합
- 일반적으로 패키지로 제공되는 함수를 이용하지만 사용자가 직접 필요한 코드를 작성하여 사용자 정의 함수를 만들어 사용 가능.

`함수명 <- function(매개변수) { 실행문 }` 의 형태로 선언합니다.

```
# 사용자 정의 함수

myFunc <- function(k) {
  cat("인자의 값은 :",k)
  return(k+100)
}

result = myFunc(100)
result
```

제어문(반복문, 조건문(if-else), 사용자 정의함수

주석 : 실행되지 않는 문장(#으로 표시함)

- 코드, 함수를 설명하기 위함.

반복문	조건문(if-else)	repeat
<ul style="list-style-type: none">특정 부분 코드를 반복 수행<ul style="list-style-type: none">for, while	<ul style="list-style-type: none">참, 거짓에 따라 수행여부를 결정<ul style="list-style-type: none">if() {} else if () {} else {}	<ul style="list-style-type: none">repeat 함수는 조건문과 연산을 모두 {}안에 넣어서 활용하는 구조
<ul style="list-style-type: none"><pre>for (i in 1:9){ print(i*3) }</pre>	<ul style="list-style-type: none"><pre>a<-2 if (a>3){ print('3보다 큰 수') } else if (a==3) { print('3') } else { print('3보다 작은 수') }</pre>	<ul style="list-style-type: none"><pre>k=0 repeat{ print(k) if (k>=10) break k = k+1 }</pre>

자주 사용되는 함수들

전처리 함수	기능
head().tail()	데이터 앞, 뒤 일부분
subset()	조건식에 맞는 데이터 추출
merge()	특정 열기준 데이터 2개 병합
apply()	열/행별로 주어진 함수 적용

정규분포 함수	기능
dnorm()	정규분포에서 주어진 값에서 함수 값 구함
rnorm()	정규분포에서 주어진 개수만큼 표본추출
pnorm()	정규분포에서 주어진 값보다 작을 확률값
qnorm()	정규분포에서 주어진 넓이값을 갖는 x값

자주 사용되는 함수들

표본추출 함수	기능
runif()	균일분포에서 특정개수만큼 표본추출
sample()	주어진 데이터에서 특정개수만큼 표본추출

산점도 함수	기능
plot()	주어진 데이터의 산점도를 작성
abline()	산점도에 추가 직선을 작성

function

function 이란, **사용자 정의 함수**를 정의하는 함수.

함수 정의 형식

```
변수 <- function([매개변수]){  
  함수의 실행문  
}
```

예시)

```
f1 <- function(){  
  cat("매개변수 없는 함수")  
}
```

출력값: 매개변수 없는 함수

```
f2 <- function(x){  
  cat("x의 값 = " , x, "\n")  
}
```

```
f2(10) # 실인수  
출력값: x의 값 = 10
```

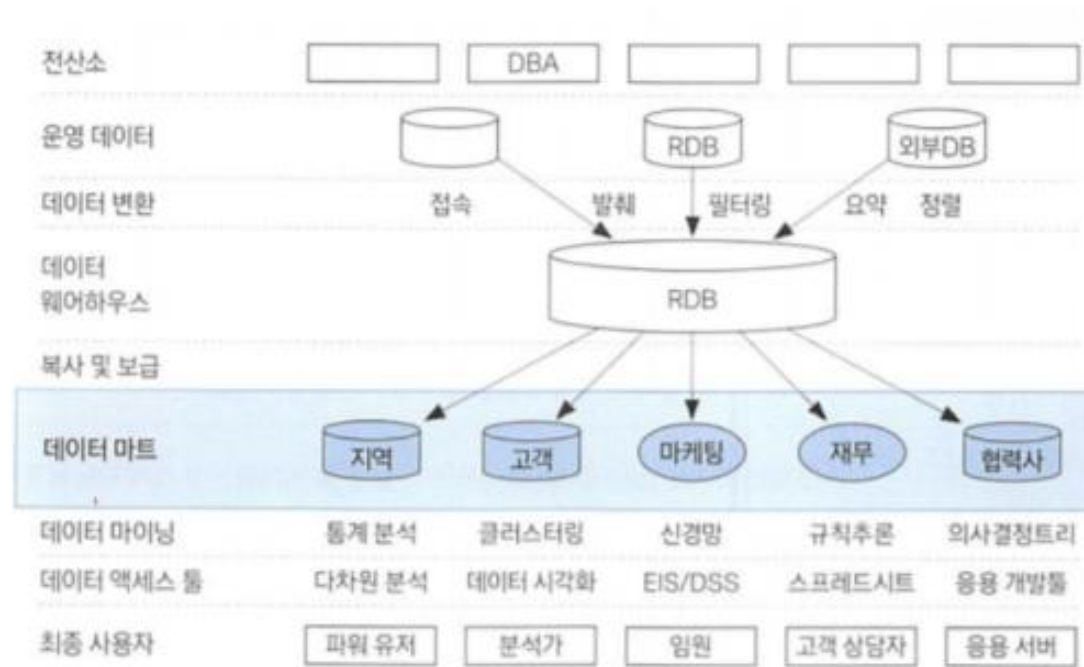
```
f2(c(1:10)) # 벡터 객체 전달  
출력값: x의 값 = 1 2 3 4 5 6 7 8 9 10
```

```
f3 <- function(x,y){  
  add <- x + y  
  return (add)  
}
```

```
add <- f3(10,30)
```

```
add  
출력값 : [1] 40
```

3장. 데이터 마트



- 데이터 웨어하우스(DW)와 사용자 사이의 중간층에 위치 한 것으로, **하나의 주제 또는 부서 중심**의 데이터 웨어하우스라고 할 수 있다.
- 대부분의 DW로부터의 **복제**, 그러나 **자체 수집 가능**, 관계형 DB나 다차원 DB 이용하여 구축한다.
- CRM 관련 업무 중 핵심
- 데이터 마트의 구축 여부에 따라 분석 효과 차이가 큼
 - > 최신 분석기법들을 사용하기에 분석가들 간 **편차가 덜하기 때문**.

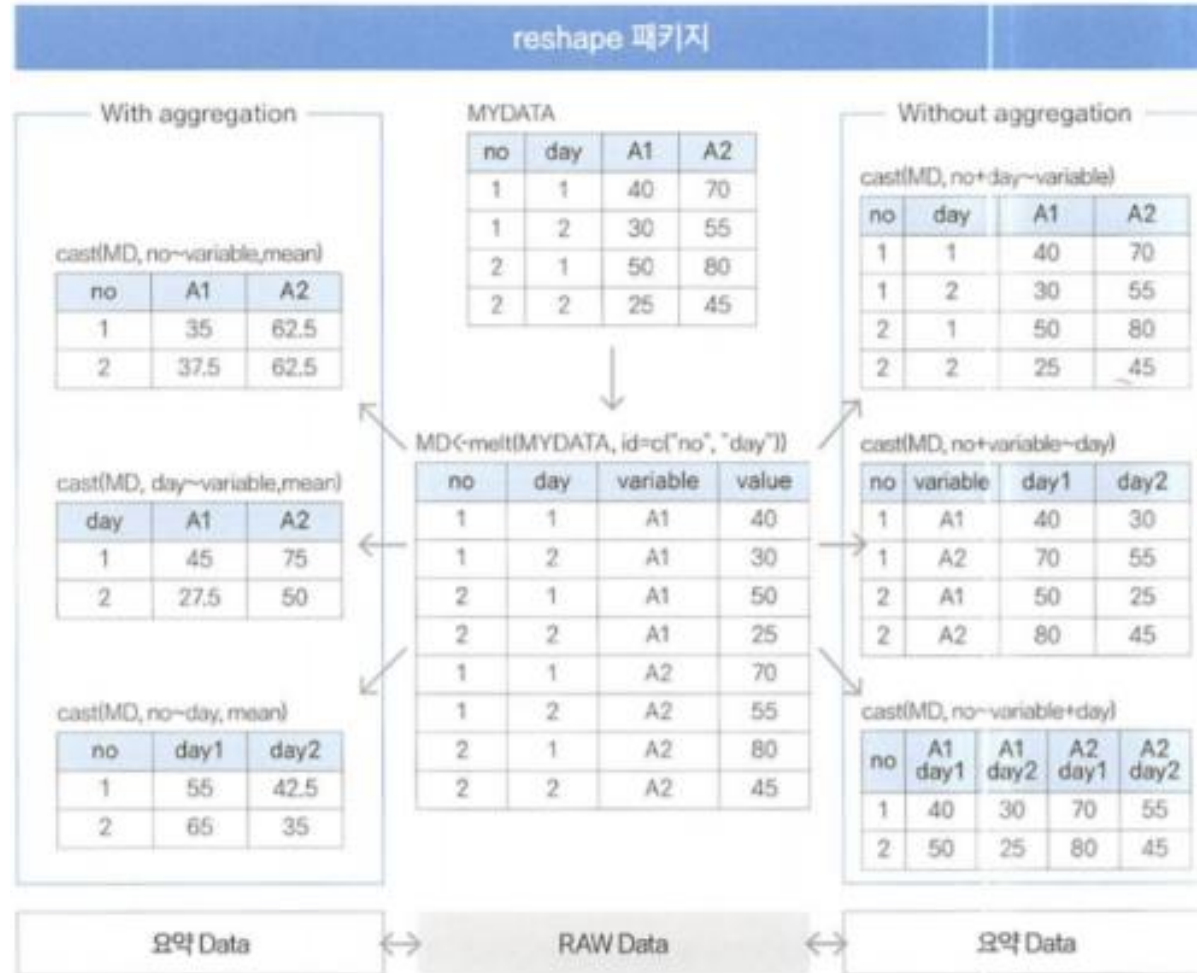
요약변수

- 수집된 정보를 분석에 맞게 **종합(aggregation)**한 변수이다.
- 많은 모델들이 **공통**으로 사용하기에 **재활용성이 높다**.
- **연속형 변수**를 범주화하여 사용해도 좋다.
- 합계, 횟수 같은 간단한 구조이므로 **자동화**하여 상황에 맞게 또는 일반적인 **자동화 프로그램**으로 구축이 가능하다.
- 요약변수만으로도 세분화하거나 **행동 예측**을 하는데 큰 도움 받을 수 있으나 **기준값(threshold value)**의 의미해석이 **애매**할 수 있어서 연속형 변수를 자동으로 타깃으로 맞춰 grouping해주면 좋다.
- 마트를 만드는데 시간과 공간의 제약이 덜한 상황이라면 **다양한 조합의 요약변수를 자동으로** 만드는 것이 적합하다.

파생변수

- 사용자(분석가)가 특정 조건을 만족하거나 특정 함수에 의해 값을 만들어 의미를 부여한 변수.
- 매우 **주관적**일 수 있으므로 논리적 타당성을 갖추어 개발해야 한다.
- 파생변수는 특정 상황에만 유의미하지 않도록 대표성을 띄게 생성해야 한다.
- 세분화, 고객행동 예측, 캠페인 **반응 예측**에 매우 잘 활용된다.
- 파생변수 자체로도 분석가능하나 이를 이용하면 데이터마이닝에 기여하는 바가 크다.
- 보다 **많은 변수**를 잘 활용하는 것이 요즘 대세이다.

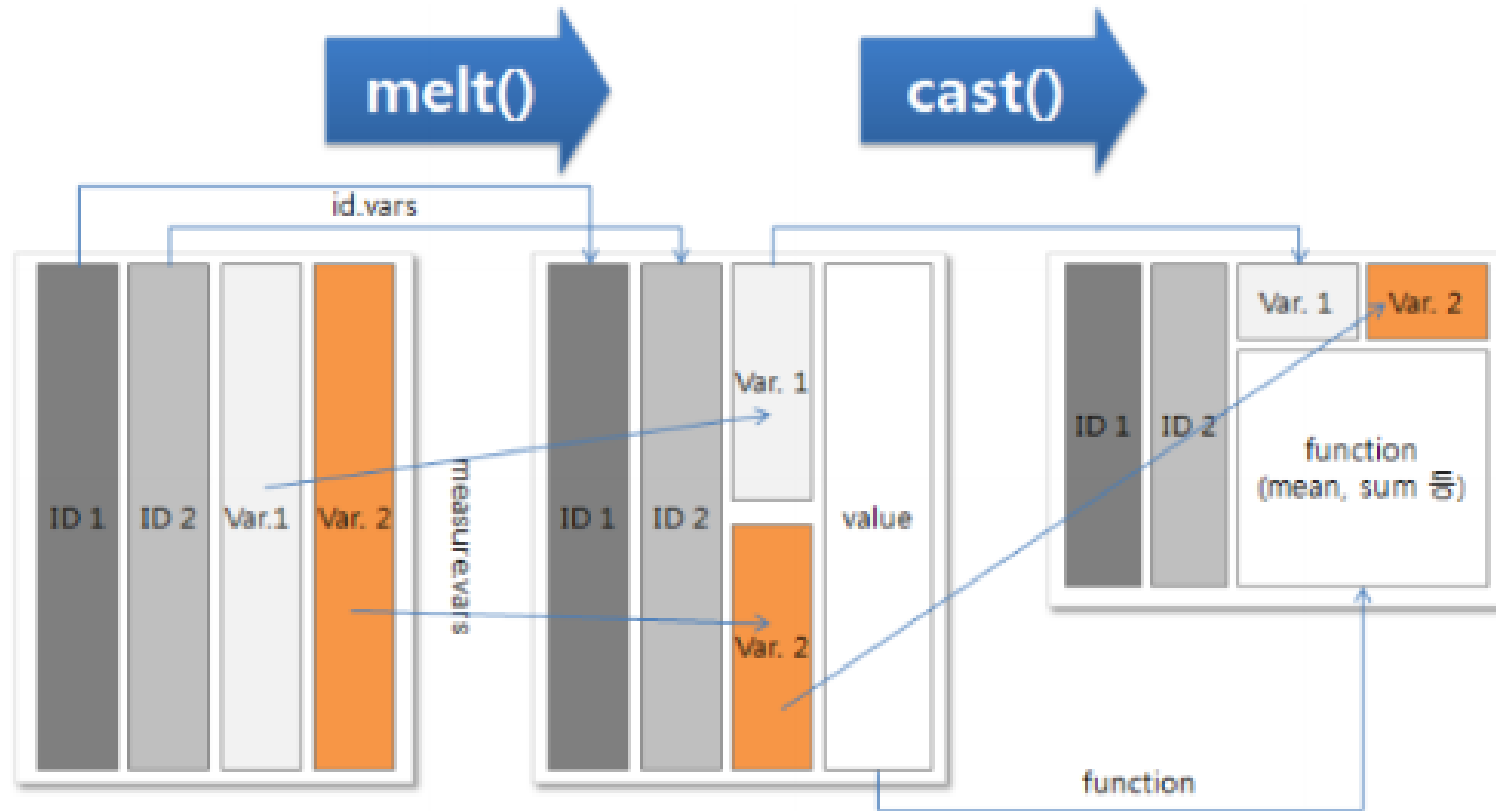
reshape 활용



- melt() : 원데이터 형태로 만드는 함수
- cast() : 요약 형태로 만드는 함수

reshape 활용

- 기존 거래 데이터 구조를 column-wise 하게 전환하는데 크게 melt와 cast단계로 구분



- 실습 : R_reshape.R

reshape 활용

- melt()

```
melt(data, id, vars, na.rm = FALSE)
```

- 데이터를 원하는 방식으로 분석하기 위해 식별자(ID), 측정변수(value) 형태로 자료를 재구성하는 단계, 녹인다고 표현을 많이 함.
- 여러 Factor 변수를 하나의 Dimension 변수와 하나의 Factor변수로 변환하는 함수.

- cast()

```
cast(data, id 변수~variable 변수, formula)
```

- melt()로 녹인 데이터들을 분석자가 원하는 여러 형태의 column으로 변환하는 함수.

sqldf를 이용한 데이터 분석

- 표준 SQL에서 사용되는 문장이 모두 가능하고 데이터 이름에 “.” 같은 특수문자가 들어간 경우 ‘ ‘ 로 묶어주면 테이블처럼 간단히 처리 가능

```
sqldf("select * from [df] limit 10 where [col] like 'char%'")
```

```
head([df])
```

```
sqldf("select * from [df] limit 6")
```

```
subset([df], grepl("qn%", [col]))
```

```
sqldf("select * from [df] where [col] like 'qn%'")
```

```
subset([df], [col] %in% c("BF","HF"))
```

```
sqldf("select * from [df] where [col] in ('BF', 'HF')")
```

```
rbind([df1], [df2])
```

```
sqldf("select * from [df1] union all select * from [df2]")
```

```
merge([df1],[df2])
```

```
sqldf("select * from [df1], [df2]")
```

```
df[order([df]$[col], decreasing=T),]
```

```
sqldf("select * from [df] order by [col] desc")
```

plyr 활용

	array	data frame	list	nothing
array	aapply	adply	alply	a_ply
data frame	dapply	ddply	dlply	d_ply
list	lapply	ldply	llply	l_ply
n replicates	raply	rdply	riply	r_ply
function arguments	maply	mdply	miply	m_ply

- 데이터를 분석하기 쉬운 형태로 **분리**하고 처리한 다음 **다시 결합**해 새로운 형태로 만들어주는 가장 필수적인 데이터 처리기능 제공
- 데이터와 **출력변수**를 동시에 **배열로 치환**하여 처리하는 패키지
- **split-apply-combine** : 데이터 분리하고 처리 후 다시 결합하는 필수적 데이터 처리기능 제공
- apply 함수와 multi-core 사용 함수 이용하면 간단하고 빠르게 처리 가능.
- ddply : plyr을 올리고 dataframe에서 dataframe으로 입출력 하는 함수.

data.table을 이용한 데이터 분석

- 큰 데이터를 탐색, 연산, 병합 하는데 아주 유용
- 기존 data frame 방식보다 월등히 빠른 속도
- 특정 column을 key 값으로 색인을 지정한 후 데이터를 처리한다.
- 빠른 Grouping과 Ordering, 짧은 문장 지원 측면에서 데이터프레임보다 유용.
- 데이터 테이블을 데이터프레임처럼 사용하면 성능이 비슷해진다.

```
> data(mtcars)
> str(mtcars$cyl)
num [1:32] 6 6 4 6 8 6 8 4 4 6 ...
> summary(mtcars$cyl)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 4.000  4.000   6.000   6.188   8.000   8.000
> table(mtcars$cyl)

 4  6  8
11  7 14
```

```
> mtcars1 <- mtcars[mtcars$cyl<7,]
> table(mtcars1$cyl)

 4  6
11  7
> str(mtcars1)
'data.frame':  18 obs. of  11 variables:
 $ mpg : num  21 21 22.8 21.4 18.1 24.4 22.8 19.2 17.8 32.4 ...
 $ cyl : num   6  6  4  6  6  4  4  6  6  4 ...
 $ disp: num  160 160 108 258 225 ...
 $ hp  : num  110 110 93 110 105 62 95 123 123 66 ...
 $ drat: num   3.9 3.9 3.85 3.08 2.76 3.69 3.92 3.92 3.92 4.08 ...
 $ wt  : num   2.62 2.88 2.32 3.21 3.46 ...
 $ qsec: num   16.5 17 18.6 19.4 20.2 ...
 $ vs  : num   0  0  1  1  1  1  1  1  1  1 ...
 $ am  : num   1  1  1  0  0  0  0  0  0  1 ...
 $ gear: num   4  4  4  3  3  4  4  4  4  4 ...
 $ carb: num   4  4  1  1  1  2  2  4  4  1 ...
```


데이터 가공_Data Exploration

- 데이터 분석을 위해 구성된 데이터의 상태를 파악한다 : head(), summary()
- summary() : 데이터 어떻게 분포돼 있는지 보여준다.

```
> summary(mtcars)
```

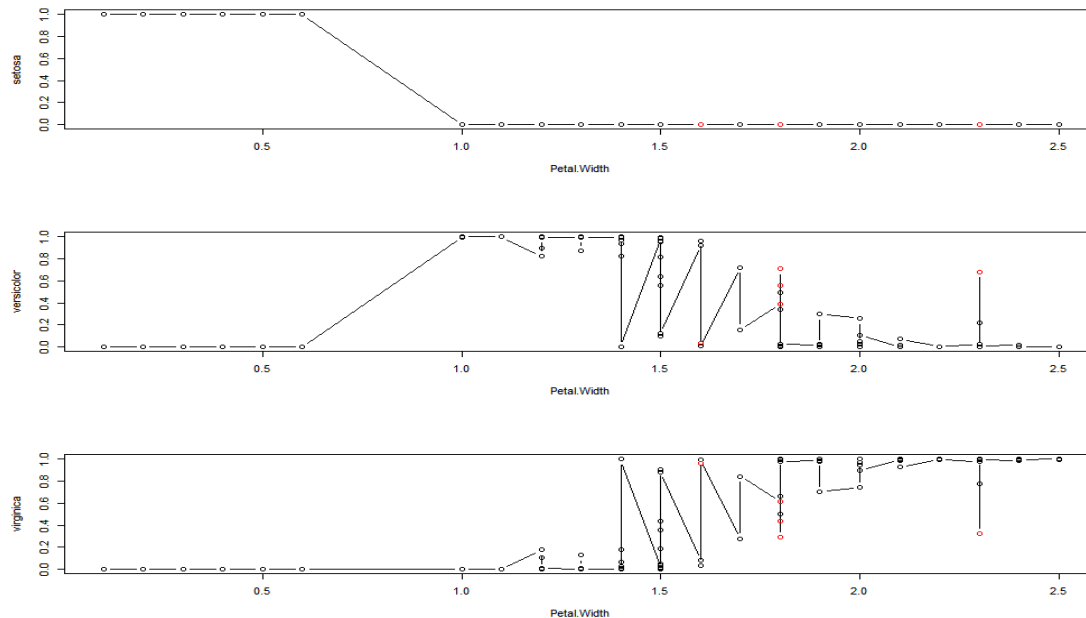
mpg	cyl	disp	hp	drat
Min. :10.40	Min. :4.000	Min. : 71.1	Min. : 52.0	Min. :2.760
1st Qu.:15.43	1st Qu.:4.000	1st Qu.:120.8	1st Qu.: 96.5	1st Qu.:3.080
Median :19.20	Median :6.000	Median :196.3	Median :123.0	Median :3.695
Mean :20.09	Mean :6.188	Mean :230.7	Mean :146.7	Mean :3.597
3rd Qu.:22.80	3rd Qu.:8.000	3rd Qu.:326.0	3rd Qu.:180.0	3rd Qu.:3.920
Max. :33.90	Max. :8.000	Max. :472.0	Max. :335.0	Max. :4.930

wt	qsec	vs	am	gear
Min. :1.513	Min. :14.50	Min. :0.0000	Min. :0.0000	Min. :3.000
1st Qu.:2.581	1st Qu.:16.89	1st Qu.:0.0000	1st Qu.:0.0000	1st Qu.:3.000
Median :3.325	Median :17.71	Median :0.0000	Median :0.0000	Median :4.000
Mean :3.217	Mean :17.85	Mean :0.4375	Mean :0.4062	Mean :3.688
3rd Qu.:3.610	3rd Qu.:18.90	3rd Qu.:1.0000	3rd Qu.:1.0000	3rd Qu.:4.000
Max. :5.424	Max. :22.90	Max. :1.0000	Max. :1.0000	Max. :5.000

carb
Min. :1.000
1st Qu.:2.000
Median :2.000
Mean :2.812
3rd Qu.:4.000
Max. :8.000

변수 중요도

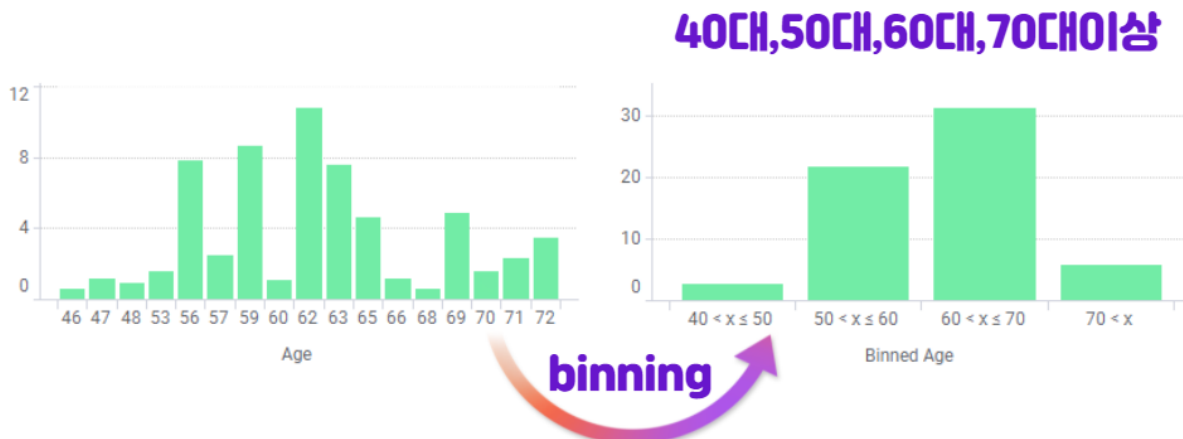
- 모델을 생성하여 사용된 변수의 중요도를 살피는 과정
- 개발 중인 모델에 준비된 데이터를 기준으로 한 번에 여러 개의 변수를 평가
- klaR 패키지 : 특정 변수 주어진 때 클래스 분류 에러율 계산, 그래프로 결과를 보여줌.
greedy.wilks() : 세분화를 위한 stepwise forward 변수 선택
→ 효율적으로 정확도를 최소한 희생하면서 초기 모델링 빨리 실행 가능
- wilk's lambda (집단 내 총분산) 활용하여 변수 중요도 정리 / plineplot()이용이 가능.
- 일반적으로 구간화 개수가 증가하면 정확도는 높아지나 속도가 느려지고 추정 오차(overestimation)가 발생 가능하다.



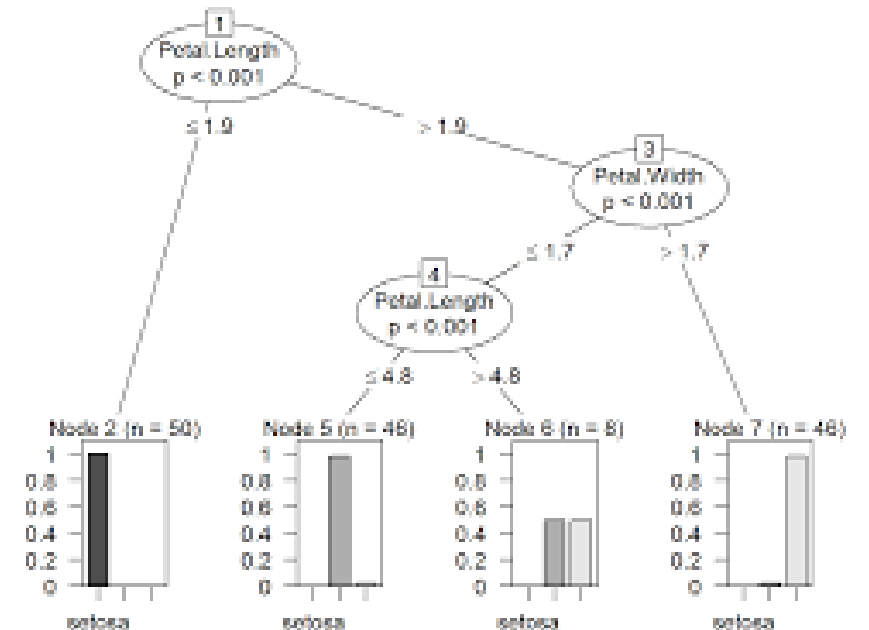
변수의 구간화

- 신형평가모형, 고객 세분화 등 시스템으로 모형을 적용하기 위해서는 각 변수들을 구간화해서 구간별로 점수를 적용할 수 있어야 한다.
- binning : 연속형 변수를 범주형 변수로 변형하는 방식
 - : 각각 동일한 개수의 레코드를 50개 이하의 구간에 데이터를 할당하여 구간들을 병합하면서 구간을 줄여나가는 방식
- 의사 결정 나무 : 세분화 또는 예측에 활용되는 의사결정 모형을 사용하여 입력 변수들을 구간화 할 수 있다.
 - : 동일한 변수를 여러분의 분리기준으로 사용 가능하기 때문에 연속변수가 반복적으로 선택될 경우, 각각의 분리 기준값으로 연속형 변수를 구간화 할 수 있다.

구간화 (Binning)



https://docs.tibco.com/pub/sfire-analyst/10.10.0/doc/html/en-US/TIB_sfire-analyst_UsersGuide/bin/bin_what_is_binning.htm



3절. 기초 분석 데이터 관리

탐색적 데이터 분석(Exploratory Data Analysis)

- 데이터의 분석에 앞서 전체적으로 데이터의 특징은 파악하고 데이터를 다양한 각도로 접근함.

```
> summary(mtcars)
```

mpg	cyl	disp	hp	drat
Min. :10.40	Min. :4.000	Min. : 71.1	Min. : 52.0	Min. :2.760
1st Qu.:15.43	1st Qu.:4.000	1st Qu.:120.8	1st Qu.: 96.5	1st Qu.:3.080
Median :19.20	Median :6.000	Median :196.3	Median :123.0	Median :3.695
Mean :20.09	Mean :6.188	Mean :230.7	Mean :146.7	Mean :3.597
3rd Qu.:22.80	3rd Qu.:8.000	3rd Qu.:326.0	3rd Qu.:180.0	3rd Qu.:3.920
Max. :33.90	Max. :8.000	Max. :472.0	Max. :335.0	Max. :4.930

wt	qsec	vs	am	gear
Min. :1.513	Min. :14.50	Min. :0.0000	Min. :0.0000	Min. :3.000
1st Qu.:2.581	1st Qu.:16.89	1st Qu.:0.0000	1st Qu.:0.0000	1st Qu.:3.000
Median :3.325	Median :17.71	Median :0.0000	Median :0.0000	Median :4.000
Mean :3.217	Mean :17.85	Mean :0.4375	Mean :0.4062	Mean :3.688
3rd Qu.:3.610	3rd Qu.:18.90	3rd Qu.:1.0000	3rd Qu.:1.0000	3rd Qu.:4.000
Max. :5.424	Max. :22.90	Max. :1.0000	Max. :1.0000	Max. :5.000

carb
Min. :1.000
1st Qu.:2.000
Median :2.000
Mean :2.812
3rd Qu.:4.000
Max. :8.000

결측값(Missing Value)

원자료					정제하기			
id	class	english	science		id	class	english	science
1	1	98	50		1	1	98	50
2	1	97	60		2	1	97	60
3	1	86	78		3	1	86	78
4	1	98	58	➔	4	1	98	58
5		80	65		7	2	90	45
6	2	89			9	3	98	15
7	2	90	45		10	3	98	45
8	2		99999		12	3	85	32
9	3	98	15					
10	3	98	45					
11	3	99999	65					
12	3	85	32					

(출처: https://rpubs.com/jmhome/R_data_processing)

- NA, 99999, '(공백)', Unknown, NotAnswer 등으로 표현 결측값 처리를 위해 시간을 많이 사용하는 것은 비효율적
- 결측값 자체가 의미 있는 경우도 있다.
- 결측값이나 이상값을 꼭 제거해야 하는 것은 아니기 때문에 분석의 목적이나 종류에 따라 적절한 판단이 필요.

결측값 처리 방법

결측값 처리 방법

단순 대치법 (Single Imputation)	다중 대치법 (Multiple Imputation)
(1) 단순 삭제 (2) 평균 대치법 비조건부 평균 대치법, 조건부 평균 대치법 (3) 단순확률 대치법 Hot-deck 방법, nearest neighbor 방법	단순 대치법을 여러 번! [1단계] 대치 (Imputation Step) [2단계] 분석 (Analysis Step) [3단계] 결합 (Combination Step)

1) 단순 대치법(Single Imputation)

: 결측값이 존재하는 레코드를 삭제 해버린다. -> 데이터수가 줄어들어 효율성이 떨어진다.

2) 평균 대치법(Mean Imputation)

: 관측 또는 실험을 통해 얻어진 데이터의 평균으로 대치한다.

: 비조건부 평균 대치법 : 관측데이터의 평균으로 대치

: 조건부 평균 대치법(regression imputation) : 회귀분석을 활용한 대치법.

결측값 처리 방법

결측값 처리 방법

단순 대치법 (Single Imputation)	다중 대치법 (Multiple Imputation)
(1) 단순 삭제 (2) 평균 대치법 비조건부 평균 대치법, 조건부 평균 대치법 (3) 단순확률 대치법 Hot-deck 방법, nearest neighbor 방법	단순 대치법을 여러 번! [1단계] 대치 (Imputation Step) [2단계] 분석 (Analysis Step) [3단계] 결합 (Combination Step)

3) 단순 확률 대치법(Single Stochastic Imputation)

: 평균대치법에서 추정량 표준 오차의 과소문제를 보완하고자 고안된 방법. Hot-deck 방법, nearest neighbor방법이 있다.

4) 다중 대치법(Multiple Imputation)

: 단순 대치법을 한번하지 않고 m번의 대치를 통해 m개의 가상적 완전 자료를 만드는 방법

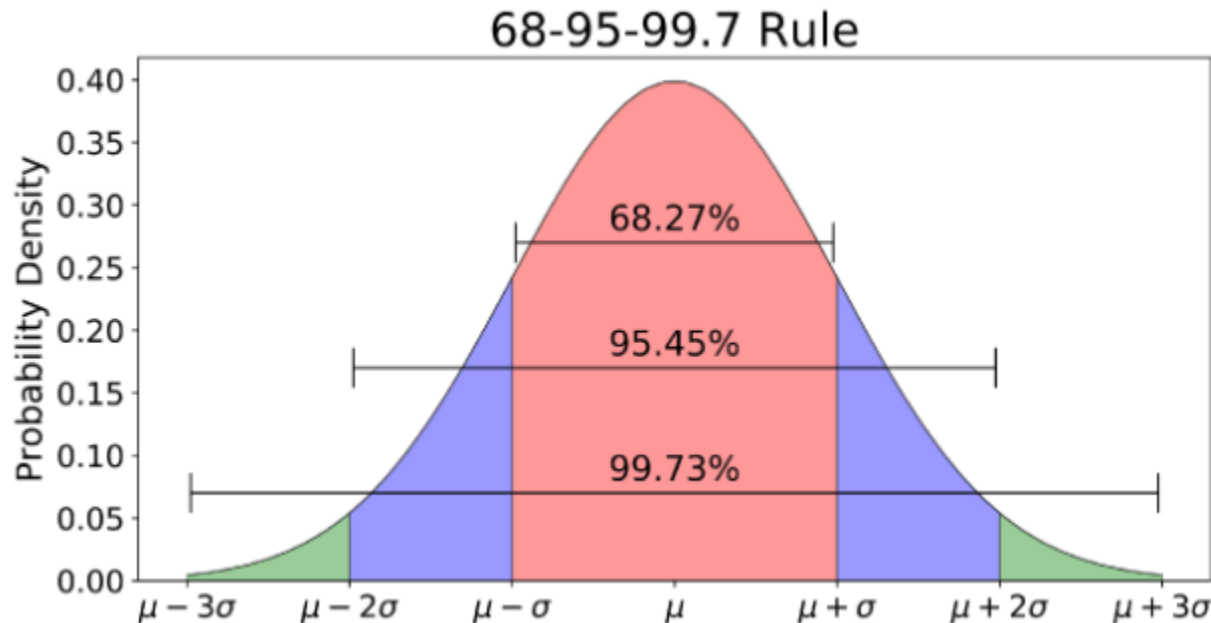
: 대치(Imputation step) → 분석(Analysis Step) → 결합(Combination Step)의 3단계를 거침.

결측값 처리 방법

함수	내용
<code>complete.cases()</code>	데이터 내 레코드에 결측값이 있으면 FALSE, 없으면 TRUE 반환
<code>is.na()</code>	결측값이 NA인지 여부를 반환
DMwR 패키지의 <code>Central Imputation()</code>	NA 값에 가운데 값(central value)로 대체 연속형 변수 - 중위값, 범주형 변수 - 최빈값
DMwR 패키지의 <code>knnImputation()</code>	NA 값을 knn 알고리즘을 사용하여 대체 k개 주변 이웃까지의 거리를 고려하여 가장 평균한 값을 사용
Amelia 패키지의 <code>amelia()</code>	randomForest 패키지의 <code>rflmpute()</code> 함수를 활용하여 NA 결측값을 대체한 후 알고리즘에 적용

이상값(Outlier)

- 의도하지 않게 **잘못** 입력된 경우(Bad data)
- 의도하지 않게 입력되었으나 분석 목적에 **부합되지 않아 제거해야** 되는 경우
- 의도하지 않은 현상이지만 분석에 포함되어야 하는 경우
- 분석에서 전처리를 어떻게 할지 결정할 때와 **부정사용방지 시스템(Fraud Detection)**에서 규칙을 발견하는데 사용.



- ESD(Extreme Studentized Deviation) : 평균에서 3 표준편차 떨어진 값(각 0.15%)
- 기하평균 - 2.5 표준편차 < data < 기하평균 + 2.5 표준편차
- $Q1 - 1.5 * IQR(Q3 - Q1) < data < Q3 + 1.5 * IQR(Q3 - Q1)$ (상자 그림의 outer fence밖에 있는값 제거)

연습문제

1. 다음 중 R의 데이터 구조 중 벡터에 대한 설명으로 적절한 것은?

- ① 벡터는 행과 열을 갖는 $m \times n$ 형태의 직사각형 데이터를 나열한 데이터 구조이다.
- ② 벡터는 하나의 스칼라 값 또는 하나 이상의 스칼라 원소들을 갖는 단순한 형태의 집합이다.
- ③ 벡터는 행렬과 유사한 2차원 목록 데이터 구조이다.
- ④ 벡터는 숫자로만 구성되어야 한다.



연습문제

2. 다음 중 연속형 변수의 경우 4분위수, 최소값, 최대값, 중앙값, 평균 등을 출력하고 범주형 변수의 경우 각 범주에 대한 빈도수를 출력하여 데이터의 분포를 파악할 수 있게 하는 함수로 적절한 것은?

- ① summary 함수
- ② ddply 함수
- ③ cast 함수
- ④ aggregate 함수



연습문제

3. R의 데이터 구조와 저장형식에 관한 설명으로 부적절한 것은?

- ① `as.numeric` 함수에 논리형 벡터를 입력하면 TRUE에 대응하는 원소는 1, FALSE에 대응하는 원소는 0인 숫자형벡터로 변형된다.
- ② 숫자형 행렬에서 우너소 중 하나를 문자형으로 변경하게 되면 해당 행렬의 모든 원소가 문자형으로 변경된다.
- ③ 데이터 프레임은 각 열 별로 서로 다른 데이터 타입을 가질 수 있다.
- ④ 행렬을 `as.vector` 함수에 입력하면 1행부터 차례로 원소를 나열하는 벡터가 생성된다.



연습문제

4. R의 데이터 구조 중 2차원 목록 데이터 구조이면서 각 열이 서로 다른 데이터 타입을 가질 수 있는 데이터 구조로 적절한 것은?

- ① 벡터
- ② 행렬
- ③ 배열
- ④ 데이터프레임



연습문제

5. R에서 제공하는 데이터 가공, 처리를 위한 패키지의 설명으로 가장 부적절한 것은?

- ① data.table 패키지는 데이터 프레임 처리함수인 ddply함수를 제공한다.
- ② reshape 패키지는 melt와 cast를 이용하여 데이터를 재구성할 수 있다.
- ③ Sqldf 패키지는 R에서 표준 SQL 명령을 실행하고 결과를 가져올 수 있다.
- ④ plyr 패키지는 데이터의 분리, 결합 등 필수적인 데이터 처리 기능을 제공한다.



연습문제

6. 다음 중 아래 R 코드의 결과로 적절한 것은?

```
s<-c("Monday", "Tuesday", "Wednesday")  
Substr(s,1,2)
```

- ① "Mo", "Tu", "We"
- ② "Monday", "Tuesday"
- ③ "Mo", "Tu"
- ④ "Monday"



연습문제

7. 아래 프로그램의 실행 결과로 다음 중 적절한 것은 무엇인가?

```
Calculate <-function(a){  
  y=1  
  for(i in 1:a) {  
    y=y*i  
  }  
  print(y)  
}  
Calculate(4)
```

① 24

② 20

③ 12

④ 6



연습문제

8. 아래 프로그램을 통해 생성된 xy에 대한 설명으로 부적절한 것은?

- `x<-c(1:5)`
- `y<-seq(10,50,10)`
- `xy<-rbind(x,y)`

- ① 2X5 행렬이다.
- ② `xy[1,]`은 x와 동일하다
- ③ `xy[, 1]`은 y와 동일하다
- ④ Matrix의 타입은 개체이다.



연습문제

9. R에서 새로운 패키지를 설치 및 사용하고자 할 때 명령어와 순서로 적절한 것은?

- ① `install.packages("패키지명")->library(패키지명)`
- ② `install.packages(패키지명) -> install.packages("패키지명")`
- ③ `library("패키지명")->install.packages("패키지명")`
- ④ `library(패키지명) -> install.packages("패키지명")`



연습문제

10. 아래 R 코드의 출력 결과는?

- `f <- function(x, a) return((x-a)^2)`
- `f(1:2,3)`



연습문제

11. 변수를 조합해 변수명을 만들고 변수들을 시간, 상품 등의 차원에 결합해 다양한 요약변수와 파생변수를 쉽게 생성하여 데이터 마트를 구성할 수 있는 패키지는 무엇인가?

- ① ETL
- ② reshape
- ③ OLAP
- ④ rattle



연습문제

12. 파생변수는 사용자가 특정 조건을 만족하거나 특정 함수에 의해 값을 만들어 의미를 부여한 변수이다. 다음 중 파생변수의 설명으로 적절한 것은?

- ① 파생변수는 매우 주관적인 변수일 수 있으므로 논리적 타당성을 갖춰야 한다.
- ② 파생변수는 많은 모델에서 공통적으로 많이 사용될 수 있다.
- ③ 파생변수는 재활용성이 높다.
- ④ 파생변수는 다양한 모델을 개발해야 하는 경우, 효율적으로 사용할 수 있다.





Thank you.

ADSP / 류영표 강사
ryp1662@gmail.com