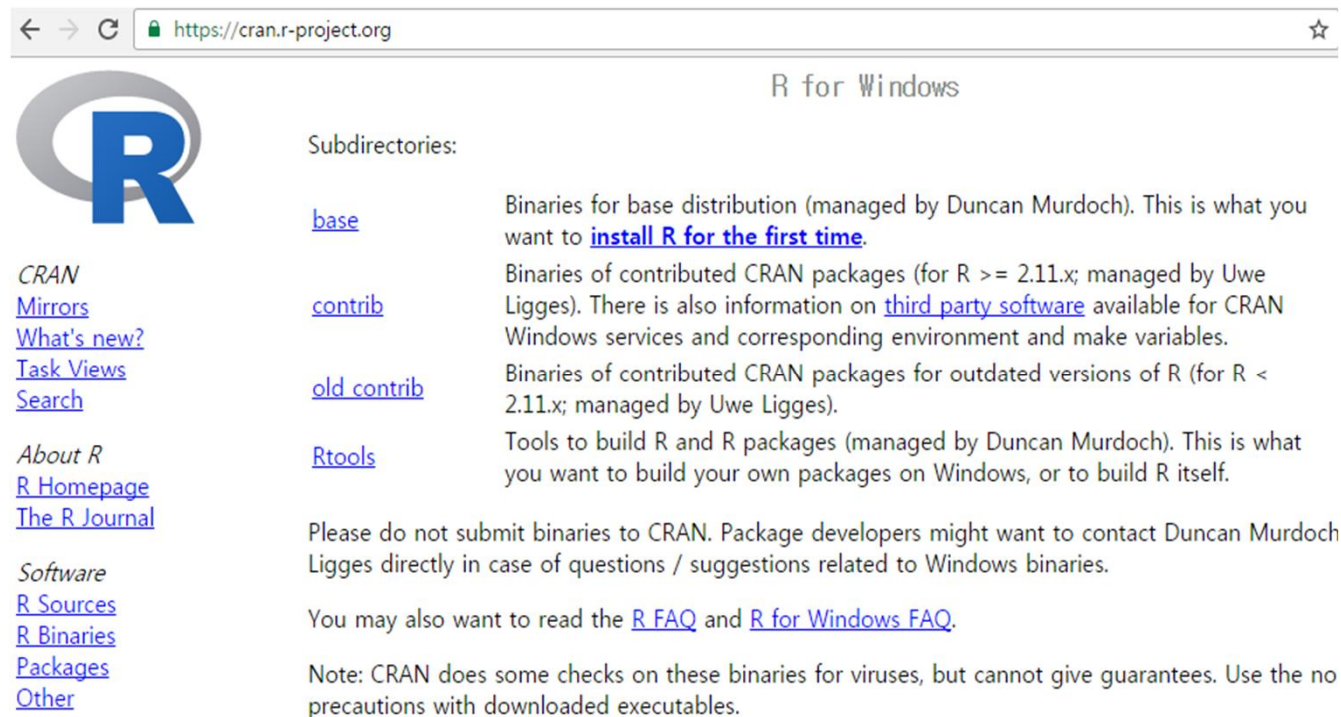


R 데이터 분석

(교육기간 : 2018년 3월 19일 ~ 3월22일)



The screenshot shows the CRAN R for Windows website. The browser address bar displays <https://cran.r-project.org>. The page features the R logo on the left, a list of subdirectories in the center, and a sidebar with various links on the right.

Subdirectories:

- [base](#): Binaries for base distribution (managed by Duncan Murdoch). This is what you want to [install R for the first time](#).
- [contrib](#): Binaries of contributed CRAN packages (for R >= 2.11.x; managed by Uwe Ligges). There is also information on [third party software](#) available for CRAN Windows services and corresponding environment and make variables.
- [old contrib](#): Binaries of contributed CRAN packages for outdated versions of R (for R < 2.11.x; managed by Uwe Ligges).
- [Rtools](#): Tools to build R and R packages (managed by Duncan Murdoch). This is what you want to build your own packages on Windows, or to build R itself.

Please do not submit binaries to CRAN. Package developers might want to contact Duncan Murdoch Ligges directly in case of questions / suggestions related to Windows binaries.

You may also want to read the [R FAQ](#) and [R for Windows FAQ](#).

Note: CRAN does some checks on these binaries for viruses, but cannot give guarantees. Use the no precautions with downloaded executables.

CRAN
[Mirrors](#)
[What's new?](#)
[Task Views](#)
[Search](#)

About R
[R Homepage](#)
[The R Journal](#)

Software
[R Sources](#)
[R Binaries](#)
[Packages](#)
[Other](#)

학습모듈(Learning Object) 및 목차

| 시간 | 1일차 | | 2일차 | | 3일차 | | 4일차 | | 5일차 | | 6일차 | | 비고 |
|------------|---|---|-------------------------------|---|--|---|--|---|---|---|---|---|----|
| 1교시 | 데이터 분석 개요 R의 개요와 역사 R의 설치 및 R의 기본 사용법 R data structure의 이해 | | 데이터 분석의 기초 기초 통계학 | | 데이터 시각화 개요 R을 활용한 머신 러닝 – Introduction | | 분류분석 – Classification CART Tree Model, Random Forest의 이해와 실습 | | 텍스트 마이닝 – Text Mining Corpus, Wordcloud, Tweet분석, 텍스트 마이닝의 이해와 실습 | | 신경망 모형 Neural Network Model Neural Network Model의 이해와 실습 | | |
| 2교시 | | | | | | | | | | | | | |
| 3교시 | | | | | | | | | | | | | |
| 4교시 | | | | | | | | | | | | | |
| 5교시 | R 프로그래밍 기초 R의 주요 package를 사용하여 데이터 처리하기 | | 기술통계학 추정과 검정 탐색적 데이터 분석 | | 예측분석 - Predictive Analysis Regression의 이해와 실습 | | 군집분석 - Clustering K-means의 이해와 실습 연관분석 – Association Analysis Association Rule, 장바구니 분석의 이해와 실습 | | 사회 연결망 분석 – Social Network Analysis 사회 연결망 분석의 이해와 실습 | | 딥러닝 Deep Learning의 개요와 이해 | | |
| 6교시 | | | | | | | | | | | | | |
| 7교시 | | | | | | | | | | | | | |
| 8교시 | | | | | | | | | | | | | |
| Total Time | | 8 | | 8 | | 8 | | 8 | | 8 | | 8 | 48 |



Session 1

데이터 분석 개요
R의 개요와 역사
R의 설치 및 R의 기본 사용법
R data structure의 이해



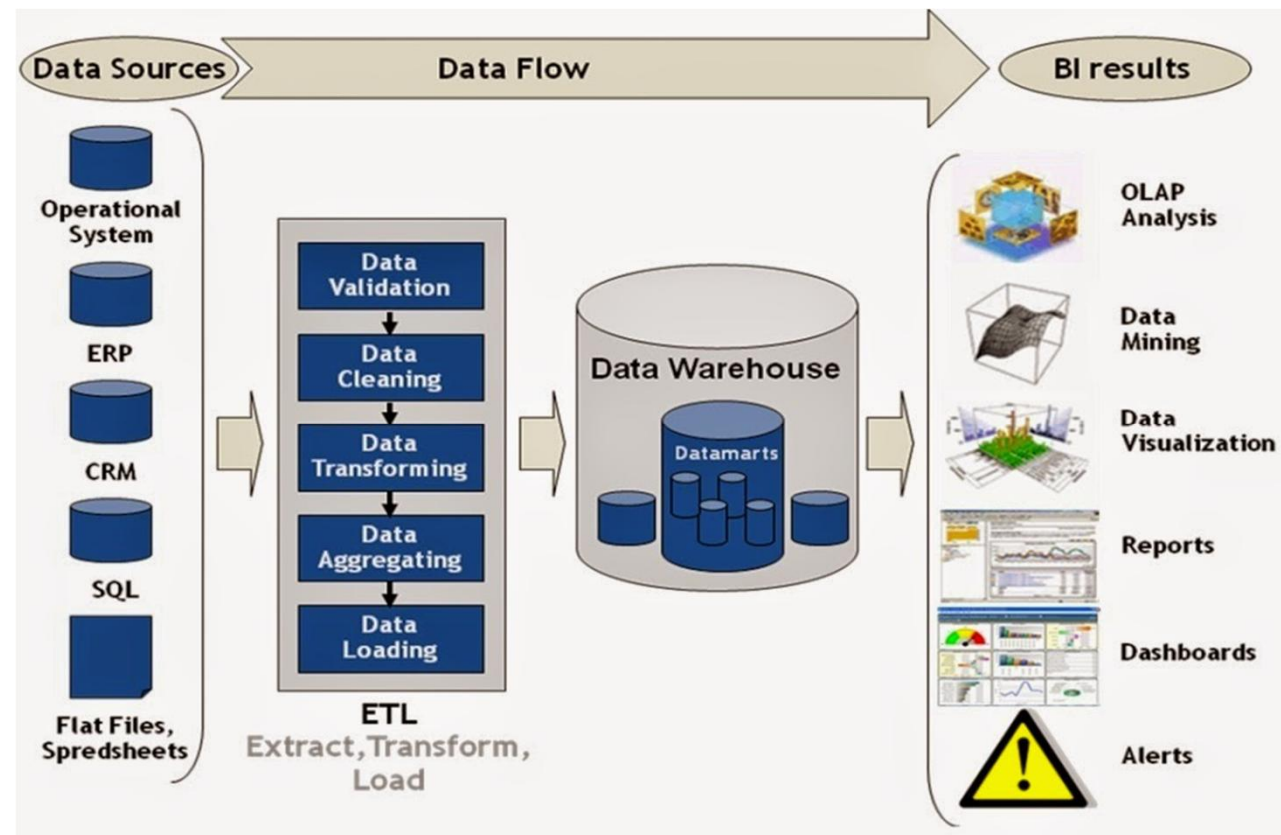
데이터 분석 프로세스

- ✓ 분석 프로세스는 요건 정의 후 이를 기반으로 분석용 데이터를 만들면서 모델링을 수행하고, 최종검증 및 테스트를 통해 모델성과에 대한 자체 평가 및 개선과정을 수행하고, 최종적으로 운영시스템에 적용하여 운영 상태로 전환하게 된다.
 - 요건 정의 – 분석 요건을 구체적으로 도출, 선별, 결정
 - ❖ 분석요건 도출 -> 수행방안 설계 -> 요건 확정
 - ❖ 단위업무(수행 준거)
 - 모델링 – 요건 정의에 의해 도출된 구체적인 분석기법을 적용하여 모델을 개발
 - ❖ 모델링 마트 설계 및 구축 -> 탐색적 분석 및 유의 변수 도출 -> 모델링 -> 모델링 성능 평가
 - 검증 및 테스트 – 분석용 데이터를 Training용/Test용으로 분리, 분석용 데이터를 이용한 자체 검증과 신규 데이터에 모델을 적용하여 결과를 도출한다
 - ❖ 운영상황에 실제 테스트 모델링 -> 비즈니스적 영향도 평가
 - 적용 – 채택된 분석 모형을 실 운영 환경에 적용하고 지속적으로 조정 및 개선하는 업무
 - 운영시스템에 적용 및 자동화 -> 주기적 리모델링
 - 분석 결과를 업무 프로세스에 완전히 통합시켜 실제 일/주/월 단위로 운영하는 것
 - 분석 시스템과 연계 운영 가능, 별도 code로 분리되어 Legacy에 별도 운영 가능

데이터 처리 기술 이해

✓ ETL(Extraction, Transporation, Loading)

- 다양한 운영계 시스템(Legacy)에서 데이터를 ODS(operation data store)로 복사한 후에 DW로 탑재하고, 주제 별로 DM(data mart)로 구성해서 사용자가 쉽게 분석을 할 수 있도록 하는 구조



데이터 분석 프로세스

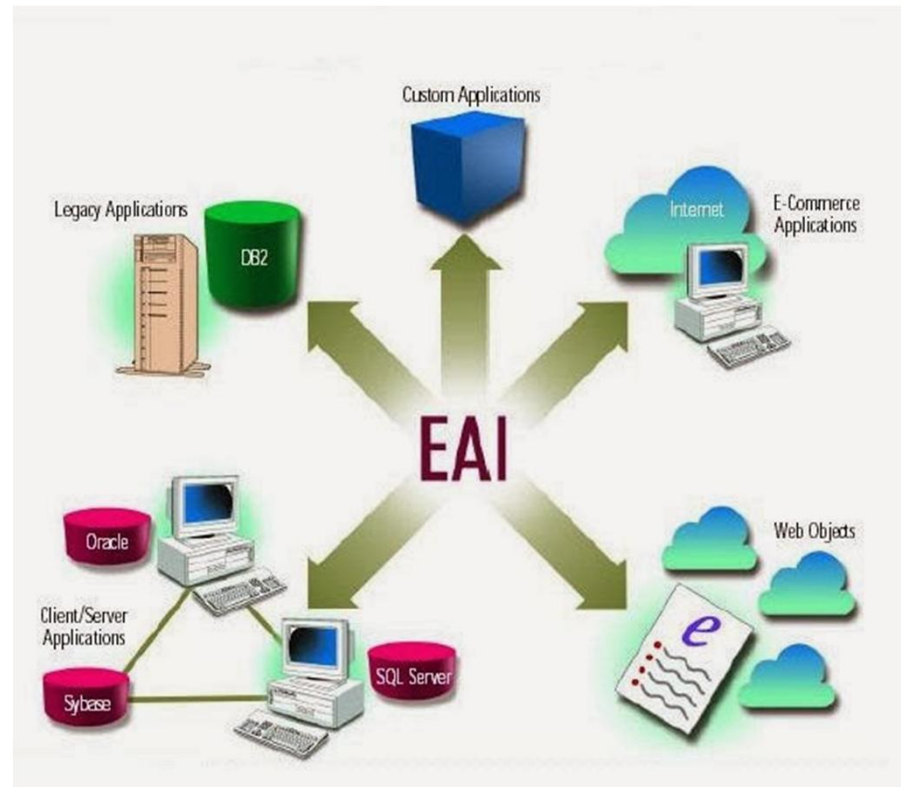
✓ CDC(Change Data Capture)

- 데이터 복제 및 로딩에 드는 시간 및 자원을 감소시켜주는 방안으로, 소스 데이터베이스의 변경을 인식해 실시간에 가깝게 처리하여 대량 데이터의 로딩을 없애준다.
- Time Stamp on Rows - 가장 단순한 방법으로 변경시점에 대한 시간정보를 기록.관리해 해당 시점 이후 데이터를 가져옴
- Version Numbers on Rows - 버전 정보관리를 통해 변경을 인식
- Status on Rows - 변경여부 정보를 이용
- Time/Version/Status on Rows - 변경시점과 버전정보 및 변경여부를 결합해서 활용
- Triggers on Tables - trigger를 등록해서 다수의 시스템에 갱신하도록 하는 방법, 복잡한 것이 단점
- Event Programming - 어플리케이션을 이용한 방식으로 부하증가와 복잡성이 단점
- Log Scanner on Database - DBMS의 Log를 이용한 방법으로 이기종 간 처리에 적합

데이터 분석 프로세스

✓ EAI(Enterprise Application Integration)

- 기업내.외부 간의 정보 연계를 위해 사용되는 기술로 통합적 관리가 가능한 시스템
- Mediation(Intra-communication) – 약속된 시간에 전달하는 방식
- Federation(Inter-communication) – 일괄적으로 요청들을 수집해서 데이터를 전달하는 방식



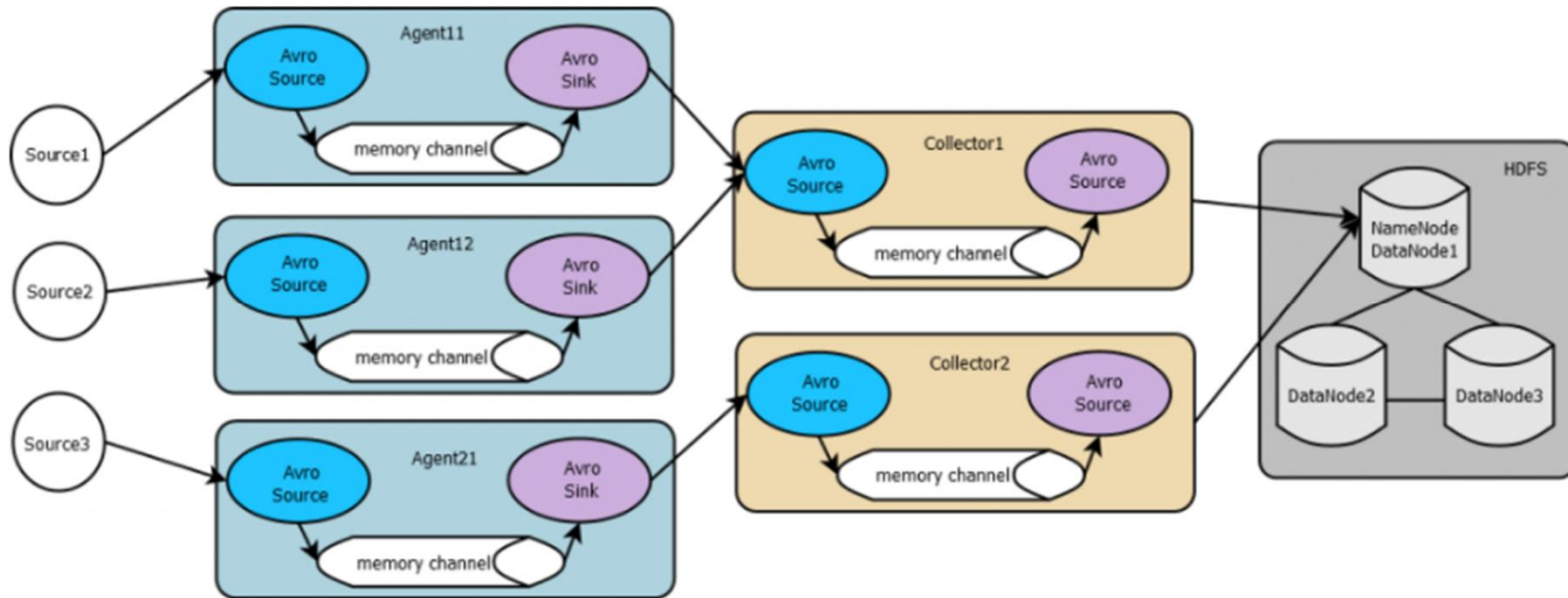
데이터 분석 프로세스

✓ 빅데이터 시대 – 데이터 연계 및 통합기법

| Batch | 비동기식 실시간 통합 | 동기식 실시간 통합 |
|------------|-------------|------------------|
| 비실시간 통합 | 근접실시간 통합 | 실시간 통합 |
| 대용량 | 중간용량 | 데이터 가능 처리시간에만 작업 |
| Batch | CDC | 웹서비스, SOA |
| 데이터 재처리 허용 | 데이터 재처리 허용 | 데이터 재처리 불가 |

데이터 분석 프로세스

- ✓ 대용량 비정형 데이터 분산 병렬처리, 데이터 연동, 대용량 질의 기술 지원
 - ✓ 성능 확장은 최소 5대 클러스터에서부터 하둡기반 2만대의 서버들을 단일 클러스트로 구성할 수 있을 정도로 선형적인 확장성
 - ✓ 서버가 분리된 상태에서 3중 복제가 되어 고장에 대해 대응
 - ✓ 맵과 리듀스 기능을 통해 key-value 기준으로 map을 해서 처리하고 결과를 merge하는 리듀스를 통해 데이터 크기에 상관없이 비즈니스 로직에 집중



데이터 분석 프로세스

✓ 분산 데이터 저장 기술

| 구분 | HDFS | Lustre |
|---------------------------------|---|--|
| Architecture | 중앙집중형 분산 파일 시스템 | 중앙집중형 분산파일 시스템으로 데이터와 메타데이터의 복제가 제공되지 않고 대신 MDS(Metadata Servers)라는 공유 저장소에 저장 |
| Naming | 네임스페이스와 메타데이터가 네임노드에 의해 관리되고 데이터노드에 저장된 파일에 대한 매핑을 수행 | MDS를 통해 지원 |
| API and client access | C를 이용한 일부 API, Java | 별도 도구를 지원 |
| Cache consistency | 일회쓰기 및 반복 읽기모델을 지원하여 수정이 불가능하고 동시 쓰기 허용이 안됨 | DLM(Distributed Lock Manager)를 사용하여 동시 읽기 저장 지원 |
| Replication and synchronization | 배치 정책에 의해 block이 여러 데이터노드에 복제 및 분산됨 | 별도 독립 소프트웨어로 지원 |
| Load balancing | 총 자원대비 활용도를 지정하여 관리하며 균형형을 위해 다른 곳으로 이동시킴 | 간단한 도구를 지원하여 새로운 곳에 저장 |

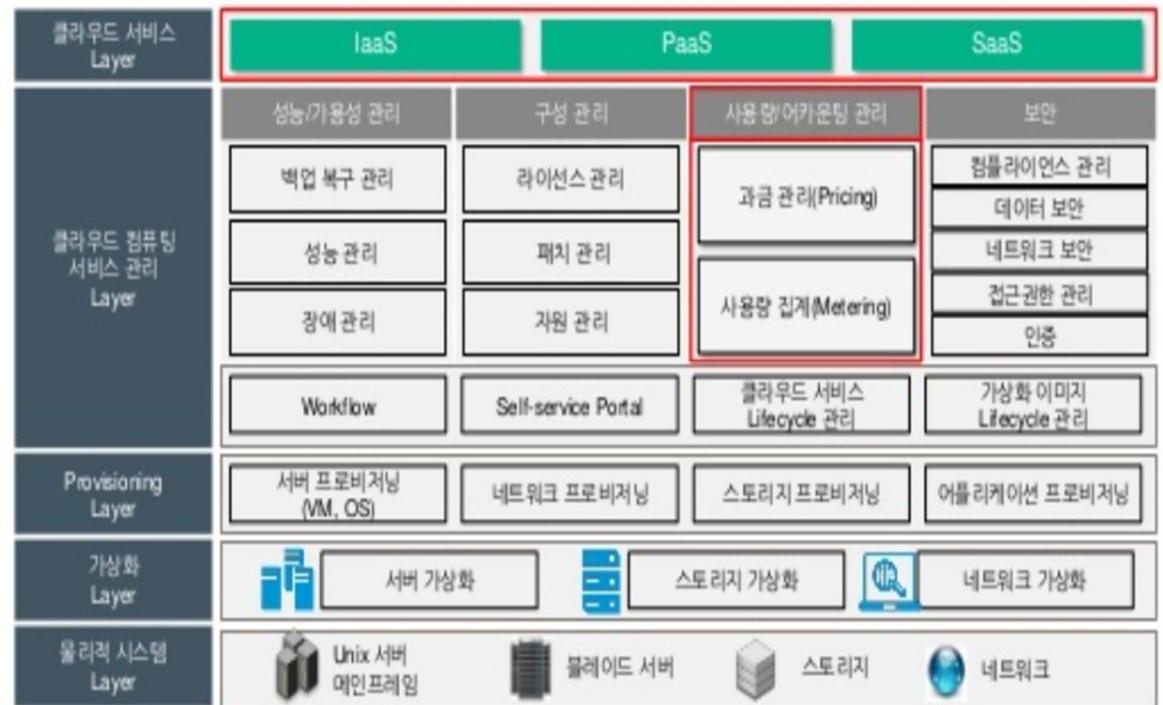
데이터 분석 프로세스

✓ 분산 데이터 저장 기술

| 구분 | HDFS | Lustre |
|------------------------------|-----------------------------------|----------------------------------|
| Fault detection | 서버들 간에 완벽하게 연결되고 커뮤니케이션되어 이상 시 대응 | 서버 커뮤니케이션에서보다 클라이언트 요청 시 감지하여 대응 |
| Chunk based | 지원 | 지원 안함 |
| User DB for storing metadata | 지원 | 지원 안함 |
| File modification | 지원안함 | 지원 안함 |

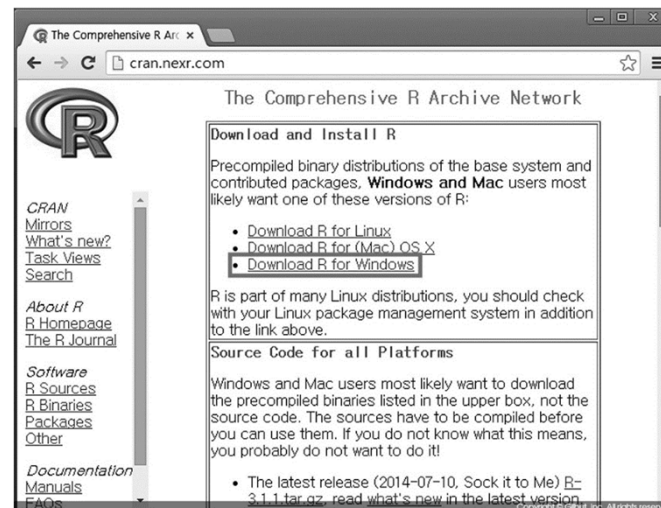
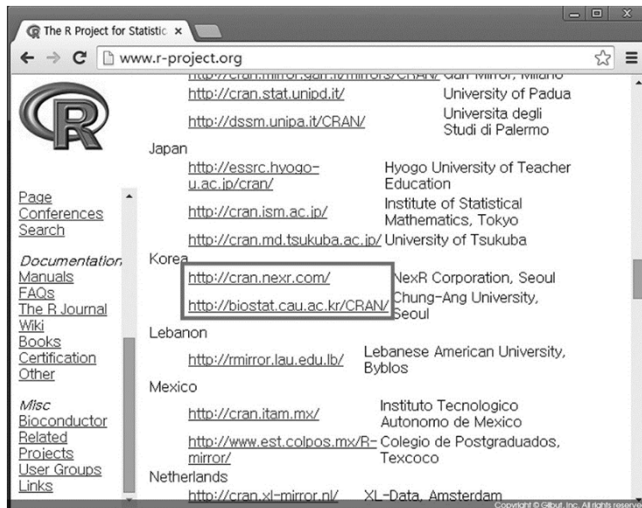
데이터 분석 프로세스

- ✓ 클라우드 컴퓨팅 및 가상화
 - 시스템 관리 편의성 제공
 - 서버 사이징의 유연성 및 정확성
 - QA, lab 환경 제공
 - 서버통합으로 데이터 센터 공간효율 증대
 - 즉각적인 서버 제공
 - 오래된 응용프로그램의 수명 연장
 - 응용프로그램 분리를 용이하게 함
 - 예측하지 못한 장애로부터 보호하여 서비스 시간 증대
 - 특정 하드웨어로의 종속성을 제거
 - 에너지 사용 감소
 - 클라우드로의 이동을 편리하게 함



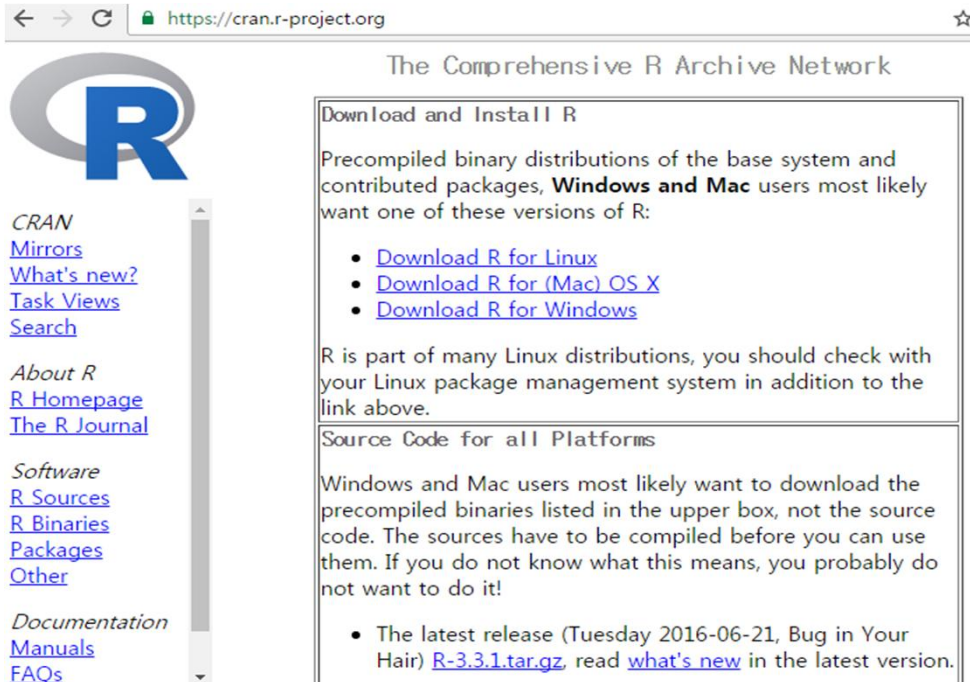
R

- ✓ R은 데이터 분석을 위한 통계 및 그래픽스를 지원하는 자유 소프트웨어 환경이다
- ✓ R은 벨 연구소에서 만들어진 통계 분석 언어인 S에 두고 있다
- ✓ R은 현재 데이터 분석을 위한 도구로 많은 인기를 누리고 있다.
- ✓ R은 컴퓨터 언어이자 다양한 패키지의 집합이다
- ✓ R 안에서 대부분의 데이터 분석을 해낼 수 있다는 장점이 있다.
- ✓ R은 통계, 기계 학습, 금융, 생물정보학, 그래픽스에 이르는 다양한 통계 패키지를 갖추고 있으며 이 모든 것이 무료로 제공된다.
- ✓ R의 다양한 패키지는 CRAN(<http://cran.r-project.org/web/views/>)을 통해 한곳에서 살펴볼 수 있다.
- ✓ R은 공개 소프트웨어로 <http://www.r-project.org/>에서 다운로드해서 설치할 수 있다



R 설치

- ✓ <http://cran.r-project.org>에 접속해서 r최신버전을 다운로드 받습니다
- ✓ iInstall R for the first time 항목을 클릭합니다.



The Comprehensive R Archive Network

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- [Download R for Linux](#)
- [Download R for \(Mac\) OS X](#)
- [Download R for Windows](#)

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

Source Code for all Platforms

Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!

- The latest release (Tuesday 2016-06-21, Bug in Your Hair) [R-3.3.1.tar.gz](#), read [what's new](#) in the latest version.

CRAN

- [Mirrors](#)
- [What's new?](#)
- [Task Views](#)
- [Search](#)

About R

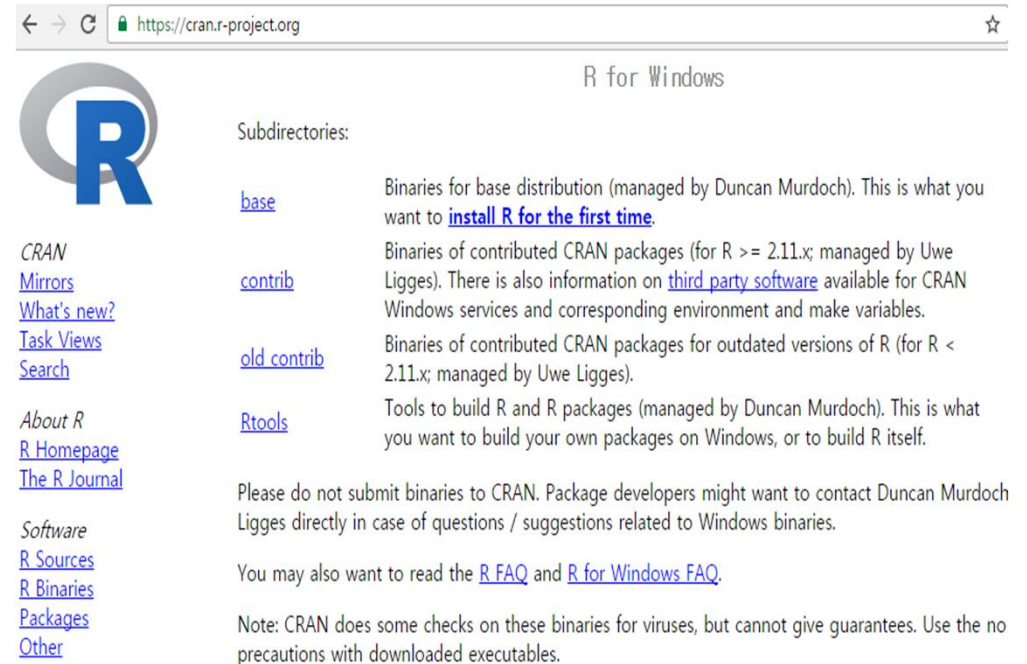
- [R Homepage](#)
- [The R Journal](#)

Software

- [R Sources](#)
- [R Binaries](#)
- [Packages](#)
- [Other](#)

Documentation

- [Manuals](#)
- [FAQs](#)



R for Windows

Subdirectories:

- [base](#): Binaries for base distribution (managed by Duncan Murdoch). This is what you want to [install R for the first time](#).
- [contrib](#): Binaries of contributed CRAN packages (for R >= 2.11.x; managed by Uwe Ligges). There is also information on [third party software](#) available for CRAN Windows services and corresponding environment and make variables.
- [old contrib](#): Binaries of contributed CRAN packages for outdated versions of R (for R < 2.11.x; managed by Uwe Ligges).
- [Rtools](#): Tools to build R and R packages (managed by Duncan Murdoch). This is what you want to build your own packages on Windows, or to build R itself.

Please do not submit binaries to CRAN. Package developers might want to contact Duncan Murdoch Ligges directly in case of questions / suggestions related to Windows binaries.

You may also want to read the [R FAQ](#) and [R for Windows FAQ](#).

Note: CRAN does some checks on these binaries for viruses, but cannot give guarantees. Use the no precautions with downloaded executables.

CRAN

- [Mirrors](#)
- [What's new?](#)
- [Task Views](#)
- [Search](#)

About R

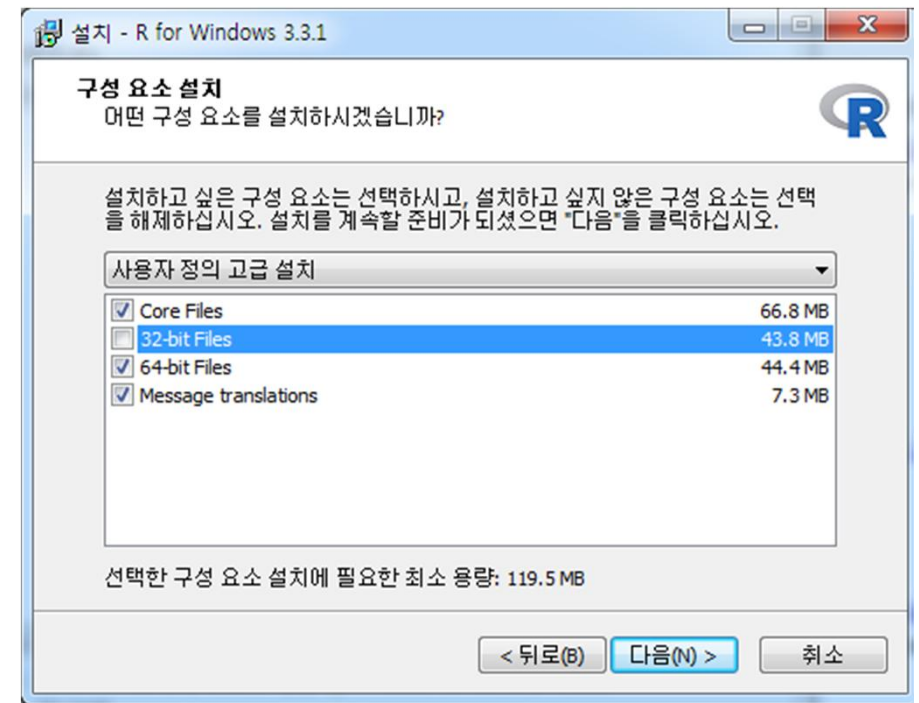
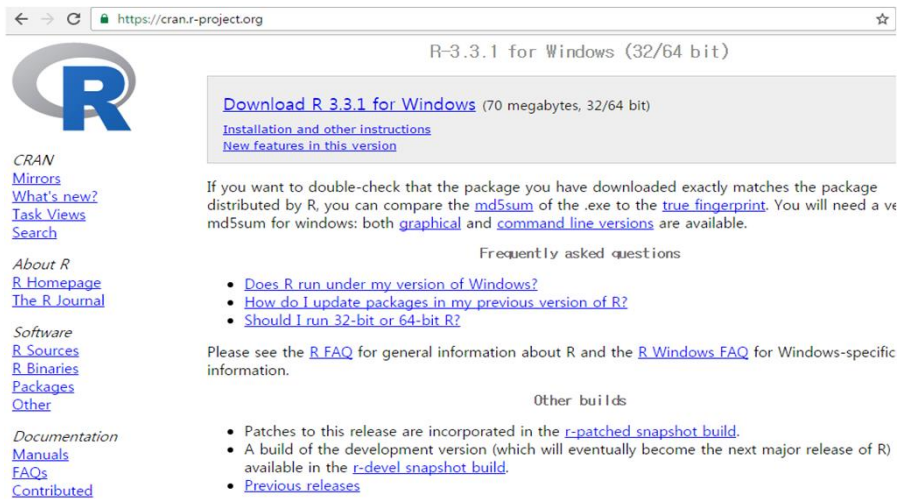
- [R Homepage](#)
- [The R Journal](#)

Software

- [R Sources](#)
- [R Binaries](#)
- [Packages](#)
- [Other](#)

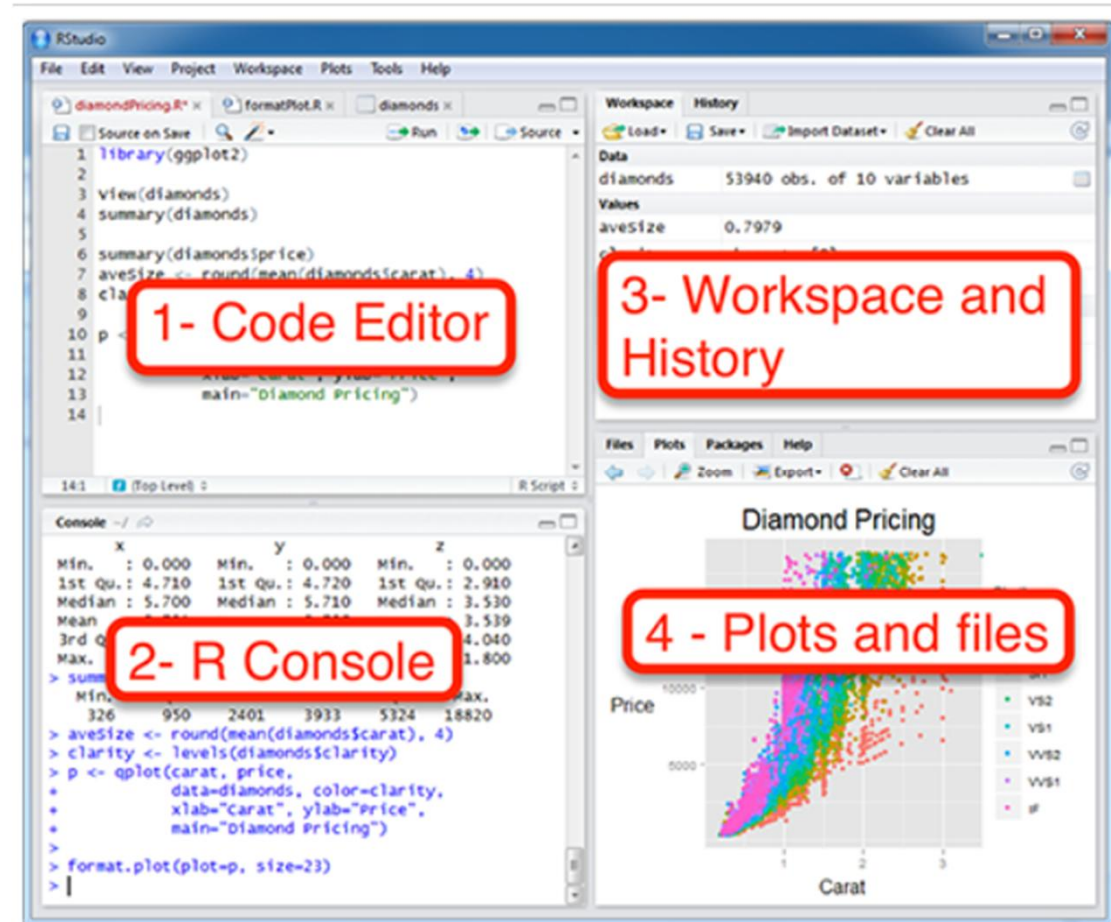
R 설치

✓ Download R3.3.1 for Windows를 클릭합니다



R Studio

- ✓ 데이터 분석을 지원하는 IDE
- ✓ Source – R 명령어 입력하는 창, Ctrl+Enter 실행
- ✓ Console – Source창에서 실행한 명령문이 실행
- ✓ Environment – 할당된 변수와 데이터가 나타나는 곳
- ✓ Packages – 패키지 설치 현황과 help 등을 볼 수 있다.

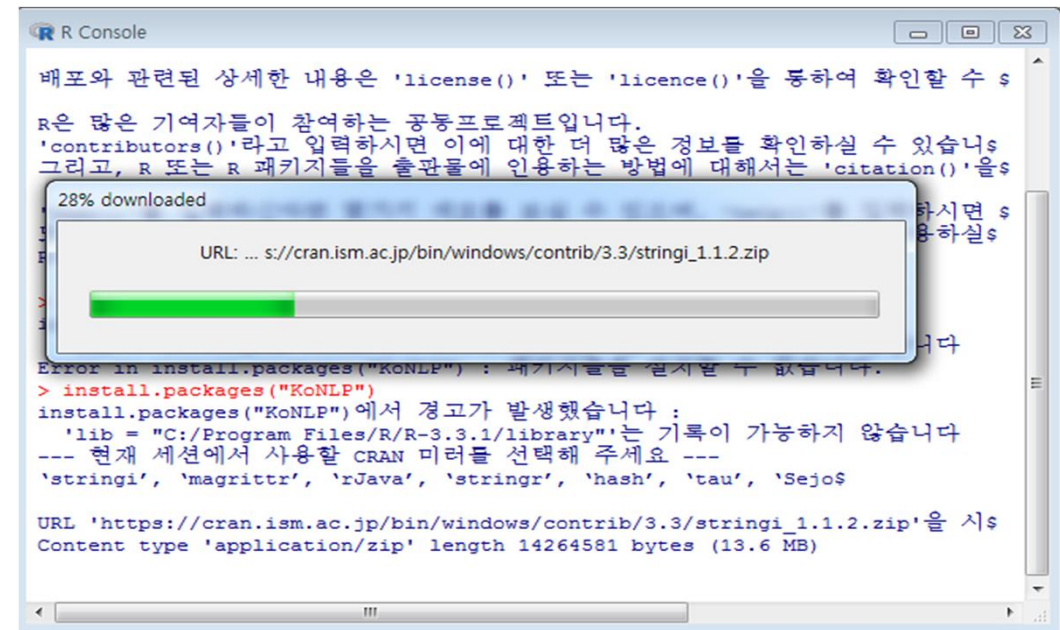


R 특성

- ✓ R은 메모리에서 모든 것을 처리
- ✓ Single thread를 사용
- ✓ 오픈 소스이므로 새로운 알고리즘이 빠르게 적용, 시험 된다
- ✓ 언어에 가까운 문장 형식을 사용해 자동화가 용이
- ✓ R은 입력된 명령을 1줄씩 바로 검사하고 처리하는 인터프리터 형태의 언어
- ✓ >는 명령 프롬프트
- ✓ R은 대소문자를 다른 문자로 구별
- ✓ 이전 작업의 history기능은 방향키를 사용
- ✓ R종료는 q()를 사용
- ✓ 스크립트창을 열어서 코드를 입력하신 후 완성되면 해당 부분을 선택하고 CTRL+R을 눌러 실행하면 편리
- ✓ # 기호는 주석

R 패키지 설치와 사용

- 패키지 설치하기 – `install.package("패키지명")`
- 패키지 사용하기 – `library(패키지명)` / `require(패키지명)` 명령
- 패키지 업데이트 – `update.packages("패키지명")`
- 패키지 삭제하기 – `remove.packages("패키지명")`
- 설치된 패키지 정보 확인 – `library(help=패키지명)`
- CRAN 사이트에서 추가로 설치할 수 있는 패키지가 어떤 것들이 있는지 확인 - `available.packages()`



R 기본

- ✓ `help` 또는 `?` : 도움말 시스템을 호출한다.
- ✓ `help.search` 또는 `??` : 주어진 문자열이 포함된 문서를 검색한다.
- ✓ `help.start` : R 시스템 전반에 대한 도움말을 담고 있는 HTML 페이지를 보여준다.
- ✓ `example` : `topic`의 도움말 페이지에 있는 Examples 섹션을 실행한다.

```
help(  
  topic # 도움말을 찾고자 하는 대상  
)  
  
help.search(  
  pattern # 찾고자 하는 문자열  
)  
  
example(  
  topic # 예제를 실행하고자 하는 topic  
)  
  
help.start()
```

```
> ?print  
> ??print  
> example(print)
```

R 기본

- ✓ RStudio는 오픈 소스 AGPL 라이선스로 제공 - 개인적인 용도로 사용할 경우 무료로 사용할 수 있다
- ✓ 제품을 배포하지 않고 내부적으로 사용할 경우 별다른 제약이 없는 GPL과 달리 AGPL은 네트워크 서비스(예를 들면, 클라우드 서비스)로 제품을 제공하더라도 소스 코드를 제공해야 할 의무를 진다.
- ✓ 장시간 분석을 수행 또는 반복적인 작업을 Rscript를 사용해 코드를 .R 파일에 저장하고 배치로 실행할 수 있다.
- ✓ source() 함수를 사용해 다른 함수를 R 스크립트에서 불러들여 실행할 수도 있다.

```
> print("hello")  
> seq(1:100)  
> # 주석입니다
```

```
> source("data_load.R")
```

R 자료형

- ❑ 숫자형(Numeric) – integer, double
- ❑ 산술 연산자 +, -, *, / , %%, ^, **
- ❑ Character (문자형) – “a”, “abc”
- ❑ NA (Not Applicable)
- ❑ NULL
- ❑ Logical – True(t), False(F)
- ❑ 날짜와 시간 : Sys.Date(), Sys.time(), date(), as.Date()
- ❑ Object – Factor, Vector, Scalar, Metrix, Array
- ❑ Collection Object – List , Data Frame

R 자료형

- ✓ 변수
 - R의 변수명은 알파벳, 숫자, _(언더스코어), .(마침표)로 구성되며, -(하이픈)은 사용할 수 없다.
 - 첫 글자는 알파벳 또는 .으로 시작해야 한다.
 - 첫 글자가 .으로 시작한다면 . 뒤에는 숫자가 올 수 없다.
 - 예약어는 사용할 수 없습니다.
- ✓ 변수값 할당
 - 할당 연산자 중 =는 명령의 최상위 수준에서만 사용할 수 있는 반면 <-는 어느 곳에서도 사용할 수 있다. 따라서 함수 호출과 동시에 변수에 값을 할당하는 목적으로는 <-만 사용할 수 있다.

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

R 함수 호출

- ✓ R의 함수 인자는 위치 또는 이름으로 지정할 수 있다.

```
foo(a, b, c=1, d=2)
```

foo 함수는 a, b, c, d라는 4개의 인자를 받을 수 있으며 c와 d에는 기본값이 지정되어 있으므로 생략이 가능하다. 만약 c 또는 d를 인자로 전달하지 않으면 각각 기본값인 c=1, d=2로 지정된다.

```
# 함수 호출시 인수 전달 예
```

```
foo(3, 4)
```

```
foo(3, 4, 1)
```

```
foo(3, 4, 1, 2)
```

```
foo(a=3, b=4, d=5)
```

```
foo(d=5, a=3, b=4)
```

```
foo(3, 4, d=5)
```

R 자료형

- ✓ 작업 디렉터리 설정

```
> setwd("디렉터리명")
```

- ✓ print() 함수는 1번에 1가지만 출력, cat() 함수는 숫자와 문자 여러 개를 한번에 출력
- ✓ 여러 개의 명령을 연속적으로 실행하고 싶을 경우에 세미콜론(;)을 사용

```
> print(3, 4)
> cat(1, 'a', 2, 'b')
> 1+2 ; 3*2 ; 4/2
```

- ✓ 산술연산자

- / 나누기 (실수 가능)
- %/% 정수 나누기
- %% 나머지 구하기
- ^, ** 승수 구하기

- ✓ 데이터 타입 확인할 때는 class 함수 사용

```
> class('1')
> class(1)
```


R 자료형

✓ NA – Not Applicable

값이 있어도 정해진 범위 안에 있는 값이 아니라서 사용할 수 없는 경우

```
> sum(1, NA, 2) # NA를 더하므로 결과가 NA로 출력됩니다.  
> sum(1, NULL, 2) #NULL값을 제외하고 나머지 값만 더해서 결과 출력  
> sum(1, 2, NA, na.rm=T) # NA값을 제거하고 올바른 계산을 수행
```

`is.na` : NA 값이 저장되어 있는지를 판단한다.

```
is.na(  
  x # R의 데이터 객체  
)
```

NA가 저장되어 있으면 TRUE, 그렇지 않으면 FALSE를 반환한다.

```
> four <- NA  
> is.na(four)
```

R 자료형

✓ NULL

- NULL은 NULL 객체를 뜻하며, 변수가 초기화되지 않았을 때 사용한다.

`is.null` : 변수에 NULL이 저장되어 있는지를 판단한다.

```
is.null(  
  x # R의 데이터 객체  
)
```

반환 값은 NULL이 저장되어 있으면 TRUE, 그렇지 않으면 FALSE다.

```
> x <- NULL  
> is.null(x)  
> is.null(1)  
> is.null(NA)  
> is.null(NULL)
```

```
is_even <- NULL  
if (a 가 짝수면) {  
  is_even <- TRUE  
} else {  
  is_even <- FALSE  
}
```

- ✓ NA는 결측치, 즉 값이 빠져 있는 경우를 뜻한다
- ✓ NULL은 프로그래밍의 편의를 위해 미정(undefined) 값을 표현하는 데 사용하는 개념이다.

R 자료형

✓ 문자열

```
> a <- "hello"
> print(a)

> a <- 'hello'
> print(a)
```

✓ 진리값

- TRUE, T는 모두 참 값을 의미
- FALSE, F는 거짓 값을 의미
- 진릿값에는 &(AND), |(OR), !(NOT) 연산자를 사용할 수 있다.
- TRUE와 FALSE는 예약어reserved word고 T, F는 각각 TRUE와 FALSE로 초기화된 전역 변수다.

```
> TRUE & TRUE
> TRUE & FALSE
> TRUE | TRUE
> TRUE | FALSE
> !TRUE
> !FALSE
> T <- FALSE
> TRUE <- FALSE
> c(TRUE, TRUE) & c(TRUE, FALSE)
> c(TRUE, TRUE) && c(TRUE, FALSE)
```

R 자료형

✓ 날짜와 시간

```
> Sys.Date() # 날짜만 보여주는 함수
> Sys.time() # 날짜와 시간을 보여주는 함수
> date()      # 미국식 날짜와 시간을 출력하는 함수
> as.Date('2017-12-01') # 문자형태의 날짜를 날짜타입으로 변환해주는 함수
> as.Date('2017/07/04')
> as.Date('04-07-2017') #오류
> as.Date('2017-12-01', format='%d-%m-%Y')
> as.Date(10, origin='2017-12-01') #주어진 날짜 기준으로 10일후의 날짜
> as.Date(-10, origin='2017-12-01') #주어진 날짜 기준으로 10일 이전 날짜
```

✓ 날짜 format

- %d 일자를 숫자로 인식
- %m 월 을 숫자로 인식
- %b 월을 영어 약어로 인식
- %B 월을 전체 이름으로 인식
- %y 년도를 숫자 두 자리로 인식
- %Y 년도를 숫자 네 자리로 인식

R 자료형

✓ 날짜 산술 연산

```
as.Date("2017-07-31")-as.Date("2017-07-04")
```

- POSIXt : 날짜를 년, 월, 일 로 표시하는 리스트형 클래스
- POSIXct : 날짜를 연속적인 데이터로 인식해서 1970년을 기준으로 초 단위로 계산
- POSIXct 은 R고급 과정에서 회귀분석 등을 할 때 주로 POSIXct를 많이 사용

```
as.Date("2017-07-04 20:00:00 ") - as.Date("2017-07-04 18:30")  
as.POSIXct("2017-07-04 20:00:00 ") - as.POSIXct("2017-07-04 18:30")
```

✓ lubridate 패키지로 날짜와 시간 제어하기

```
install.packages("lubridate")  
library(lubridate)  
date<-now() #현재 날짜와 시간 넣기  
date  
year(date) #년도만 출력  
month(date,label=T) #월을 영문으로 출력  
month(date,label=F) #월을 숫자로 출력  
day(date)
```

R 자료형

✓ lubridate 패키지로 날짜와 시간 제어하기

```
wday(date, label=T) # 요일을 영문으로 출력
wday(date, label=F) # 요일을 가중치 숫자로 출력 , 일요일 1 시작
date<-date-days(2) #2일전 날짜 출력
date
month(date)<-2 #2개월 더한 날짜 출력
date
date+years(1) #1년 추가
date+months(1) #1개월 추가
date+hours(1) #1시간 추가
date+minutes(1) #1분 추가
date+seconds(1) #1초 추가
date<-hm("22:30") ; date #시간 분 지정
date<-hms("22:30:15") ; date #시간 분 초 지정
```

R 자료형

```
> objects( )  # 생성한 모든 변수 확인  
> rm(list=ls())  # 모든 변수들을 삭제  
> rm(var3)  # 변수 삭제
```

Hadoop R 연동

✓ Factor 형

- 여러 번 중복으로 나오는 데이터들을 각 값으로 모아서 대표 값을 출력해 주는 형태
- stringsAsFactors=FALSE 옵션은 대표값으로 정리하지 않고 중복되는 상태 그대로 사용하게 해 줍니다.
- 범주형Categorical 데이터(자료)를 표현하기 위한 데이터 타입
- 범주형 데이터 - 데이터가 사전에 정해진 특정 유형으로만 분류되는 경우
- 범주형 데이터는 또 다시 명목형Nominal과 순서형Ordinal으로 구분
- 명목형 데이터는 값들 간에 크기 비교가 불가능한 경우
- 순서형 데이터는 대, 중, 소와 같이 값에 순서를 둘 수 있는 경우

factor : 팩터 값을 생성한다.

```
factor(  
  X,          # 팩터로 표현하고자 하는 값(주로 문자열 벡터로 지정)  
  levels,     # 값의 레벨  
  ordered    # TRUE면 순서형, FALSE면 명목형 데이터를 뜻한다. 기본값은 FALSE>다.  
)
```

반환 값은 팩터형 데이터 값이다.

Hadoop R 연동

✓ Factor 형

nlevels : 팩터에서 레벨의 개수를 반환한다.

```
nlevels(  
  x # 팩터 값  
)
```

반환 값은 팩터 값의 레벨 개수다.

ordered : 순서형 팩터를 생성한다.

```
ordered(  
  x # 팩터로 표현하고자 하는 값(주로 문자열 벡터로 지정)  
)
```

levels : 팩터에서 레벨의 목록을 반환한다.

```
levels(  
  x # 팩터 값  
)
```

반환 값은 팩터에서 레벨의 목록이다.

is.ordered : 순서형 팩터인지를 판단한다.

```
is.ordered(  
  x # R 객체  
)
```

반환 값은 x가 순서형 팩터면 TRUE, 그렇지 않으면 FALSE다.

Hadoop R 연동

✓ Factor 형

```
> sex <- factor("m", c("m", "f"))
> sex
> nlevels(sex)
> levels(sex)
> levels(sex)[1]
> levels(sex)[2]
> levels(sex) <- c("male", "female") #팩터 변수에서 레벨 값을 직접 수정
> sex
> factor(c("m", "m", "f"), c("m", "f"))
> ordered("a", c("a", "b", "c"))
```

R 자료형

- ✓ 벡터 - 동일한 형태의 데이터를 모아서 함께 저장
1차원 배열과 비슷한 개념, 특정 항목의 요소를 사용하려면 벡터명[색인]
벡터 자체를 연산 할 수 있습니다.
각 벡터의 요소에 names() 함수를 사용해서 이름 지정할 수 있습니다.
seq(), rep() 함수를 사용해서 벡터에 연속적인 데이터 할당 할 수 있습니다.
length() 함수는 벡터의 길이를 리턴합니다.
%in%는 벡터에 특정 문자의 포함 여부를 리턴합니다.

c : 주어진 값들을 모아 벡터를 생성한다.

names : 객체의 이름을 반환한다.

names<- : 객체에 이름을 저장한다.²

```
c(  
  ... # 벡터로 모을 R 객체들  
)
```

```
names(  
  x # 이름을 얻어올 R 객체  
)
```

```
names(  
  x # 이름을 저장할 R 객체  
) <- value # 저장할 이름
```

반환 값은 벡터다.

반환 값은 x와 같은 길이의 문자열 벡터 또는 NULL이다.

R 자료형

✓ 벡터 데이터 접근

| 문법 | 의미 |
|----------------------------|---|
| <code>x[n]</code> | 벡터 <code>x</code> 의 <code>n</code> 번째 요소. <code>n</code> 은 숫자 또는 셀의 이름을 뜻하는 문자열 |
| <code>x[-n]</code> | 벡터 <code>x</code> 에서 <code>n</code> 번째 요소를 제외한 나머지. <code>n</code> 은 숫자 또는 셀의 이름을 뜻하는 문자열 |
| <code>x[idx_vector]</code> | 벡터 <code>x</code> 로부터 <code>idx_vector</code> 에 지정된 요소를 얻어옴. 이때 <code>idx_vector</code> 는 색인을 표현하는 숫자 벡터 또는 셀의 이름을 표현하는 문자열 벡터 |
| <code>x[start:end]</code> | 벡터 <code>x</code> 의 <code>start</code> 부터 <code>end</code> 까지의 값을 반환함. 반환 값은 <code>start</code> 위치의 값과 <code>end</code> 위치의 값을 모두 포함함 |

`length` : 객체의 길이를 반환한다.

```
length(  
  x # R 객체. 팩터, 배열, 리스트를 지정한다.  
)
```

반환 값은 객체의 길이이다.

`NROW` : 배열의 행 또는 열의 수를 반환한다.

```
NROW(  
  x # 벡터, 배열 또는 데이터 프레임  
)
```

반환 값은 행의 수다.

R 자료형

✓ 벡터

```
> (x <- c(2014+4, 2014-4, 2014+4, 2014/4)) #함수 내에서 연산
> x1 <- c(14, 18, 22)
> x2 <- c(26, 20, 34)
> (x3 <- c(x1, x2))
# n:m 표현식, 1씩 증가하거나 1씩 감소하는 단순한 수열일 경우 사용
> z<- 14:20
# from과 to로 처음과 끝을 지정하고, by로 증감 단위를 지정하거나, length.out을 등분개수를 지정해 수열 생성 seq()
> seq(from=14, to=30, by=4)
> seq(from=14, to=30, length.out=5)
# rep(반복할 내용, 반복수)
> rep(7:8, times=3)
> rep(7:8 each=3)
#조건을 이용한 원소 선택
> z [ z > 3 ]
> z [ z %% 3 == 0]
# index를 부여하고 원소 선택
> (z <- letters[1:20])
> names(z) <- 1:20
```

R 자료형

✓ 벡터 연산

- 벡터는 값을 하나씩 접근해 해당 값을 사용한 계산을 수행하거나, 벡터 전체에 대해 연산을 한 번에 수행하거나, 벡터를 집합처럼 취급해 집합 연산(합집합, 교집합, 차집합)을 계산할 수 있다.

`identical` : 객체가 동일한지를 판단한다.

```
identical(  
  x, # R 객체  
  y  # R 객체  
)
```

반환 값은 x와 y가 동일하면 TRUE, 그렇지 않으면 FALSE다.

`union` : 합집합을 구한다.

```
union(  
  x, # 벡터  
  y  # 벡터  
)
```

반환 값은 x와 y의 합집합이다.

`intersect` : 교집합을 구한다.

```
intersect(  
  x, # 벡터  
  y  # 벡터  
)
```

반환 값은 x와 y의 교집합이다.

R 자료형

✓ 벡터 연산

setdiff : 차집합을 구한다.

```
setdiff(  
  x, # 벡터  
  y  # 벡터  
)
```

반환 값은 x와 y의 차집합이다.

setequal : x와 y가 같은 집합인지 판단한다.

```
setequal(  
  x, # 벡터  
  y  # 벡터  
)
```

반환 값은 x와 y가 같은 집합인지 여부다.

```
> vec 1 <- c(1, 2, 3, 4, 5)  
> vec1[-3]  
> vec1 [3]  
> vec 2 <- c(10, 9, 8, 7, 6)  
> append(vec1, vec2, after=3)  
> vec1+vec2  
#벡터 연산 - 데이터형이 다른 경우 union을 사용  
> var3<-c('3','4','5')  
> var1+var3  
> union(var1, var3)  
> var1<-c(1,2,3)  
> setdiff(var1, var2)  
> setdiff(var2, var1)  
> intersect(var1, var2)  
> names(vec1) <- c('key1', 'key2', 'key3', 'key4', 'key5')  
> var1 <- seq(2, -2)  
> var2 <- seq(1, 10, 2)  
> var3 <- rep(1:3, 2)  
> length(var2)      #벡터의 길이  
> nrow(var2)        #행렬의 행을 리턴  
> 3 %in% var2       #벡터에 특정 문자의 포함 여부 찾기
```

R 자료형

✓ 벡터 연산

| 연산자 | 의미 |
|--------------|--|
| value %in% x | 벡터 x에 value가 저장되어 있는지 판단함 |
| x + n | 벡터 x의 모든 요소에 n을 더한 벡터를 구함. 마찬가지로 *, /, -, == 등의 연산자를 적용 가능함 |

```
> x <- c("a", "b", "c")
> x[c(1, 2)]
> x[c(1, 3)]
> x <- c(1, 3, 4)
> names(x) <- c("kim", "seo", "park")
> x
> x[c("seo", "park")]
> x <- c("a", "b", "c")
> length(x)
> nrow(x) # nrow()는 행렬만 가능
> NROW(x) # NROW()는 벡터와 행렬 모두 사용 가능
> identical(c(1, 2, 3), c(1, 2, 3))
> identical(c(1, 2, 3), c(1, 2, 100))
> c(1, 2, 3) != c(1, 2, 100)
> c(1, 2, 3) == c(1, 2, 100)
> setequal(c("a", "b", "c"), c("a", "d"))
> setequal(c("a", "b", "c"), c("a", "b", "c", "c"))
```


R 자료형

✓ 벡터

`seq` : 시퀀스를 생성한다.

```
seq(  
  from, # 시작 값  
  to,   # 끝 값  
  by    # 증가치  
)
```

from부터 to까지의 값을 by 간격으로 저장한 숫자 벡터를 반환한다.

`seq_along` : 주어진 객체의 길이만큼 시퀀스를 생성한다.

```
seq_along(  
  along.with # 이 인자 길이만큼 시퀀스를 생성한다.  
)
```

반환 값은 `along.with`의 길이가 N일 때, 1부터 N까지의 숫자를 저장한 벡터다.

```
> 3:7  
> 7:3  
> x <- c(2, 4, 6, 8, 10)  
> 1:NROW(x)  
> seq_along(x)
```

R 자료형

✓ 벡터

`rep` : 주어진 값을 반복한다.

```
rep(  
  x,      # 반복할 값이 저장된 벡터  
  times,  # 전체 벡터의 반복 횟수  
  each    # 개별 값의 반복 횟수  
)
```

반환 값은 반복된 값이 저장된 `x`와 같은 타입의 객체다.

```
> rep(1:2, times=5)  
> rep(1:2, each=5)  
> rep(1:2, each=5, times=2)
```

R 자료형

- ✓ Matrix - 벡터를 여러 개 합친 형태, 2차원으로 데이터를 저장합니다.
동일한 데이터 유형만 저장
rbind()로 행을 추가할 수 있다
cbind()로 컬럼을 추가할 수 있다.
컬럼 이름을 지정, 조회하려면 colnames() 사용
행이름 지정, 조회하려면 rownames() 사용

matrix : 행렬을 생성한다.

```
matrix(  
  data,          # 행렬을 생성할 데이터 벡터  
  nrow,          # 행의 수  
  ncol,          # 열의 수  
  byrow=FALSE,   # TRUE로 설정하면 행우선, FALSE일 경우 열 우선으로 데이터를 채운다.  
  dimnames=NULL  # 행렬의 각 차원에 부여할 이름  
)
```

반환 값은 행렬이다.

R 자료형

✓ Matrix

`dimnames` : 객체의 각 차원에 대한 이름을 가져온다.

```
dimnames(  
  x # R 객체  
)
```

반환 값은 객체 x의 각 차원에 대한 이름이다.

`dimnames<-` : 객체의 차원에 이름을 설정한다.

```
dimnames(  
  x # R 객체  
) <- value # 차원에 부여할 이름
```

```
# matrix(벡터 또는 숫자, 행수, 열수)  
# dim(행렬) - 행렬의 행, 열 수를 반환  
> mat1 <- matrix(c(1:10), nrow=2, byrow=T)  
> z <- c(11, 21, 31, 12, 22, 32)  
> dim(z)  
# dig(행렬) - godfuff의 대각선에 있는 값을 반환  
> dig(z)  
# 행렬의 곱셈은 %*%를 사용  
> mat1[, 1]  
> mat1[1, ]  
> mat1 <- rbind(mat1, c(14,15,16,17)) #길이가 다르면 오류  
> mat1 <- rbind(mat1, c(11, 12, 13, 14, 15))  
> mat1 <- cbind(mat1, c(100, 101, 102))  
> colnames(mat1) <- c('key1', 'key2', 'key3', 'key4', 'key5', 'key6')  
> mat[c(1, 2), c(2)] # 1, 2행 2열 원소  
> mat[c(1, 2), c(2)] <- 2 #특정 원소 대체  
> mat + 2000 #원소 전체에 연산 적용  
> matrix(1:9, nrow=3,  
+       dimnames=list(c("r1", "r2", "r3"), c("c1", "c2", "c3")))  
> (x <- matrix(1:9, ncol=3))  
> dimnames(x) <- list(c("r1", "r2", "r3"), c("c1", "c2", "c3"))  
> x
```

R 자료형

✓ Matrix

`rownames` : 행렬의 행 이름을 가져온다.

```
rownames(  
  x # 2차원 이상의 행렬과 유사한 객체  
)
```

반환 값은 행 이름이다.

`colnames` : 행렬의 열 이름을 가져온다.

```
colnames(  
  x # 2차원 이상의 행렬과 유사한 객체  
)
```

반환 값은 열 이름이다.

`rownames<-` : 행렬의 행 이름을 설정한다.

```
rownames(  
  x # 2차원 이상의 행렬과 유사한 객체  
) <- value # NULL 또는 x와 같은 길이의 문자열 벡터
```

`colnames<-` : 행렬의 열 이름을 설정한다.

```
colnames(  
  x # 2차원 이상의 행렬과 유사한 객체  
) <- value # NULL 또는 x와 같은 길이의 문자열 벡터
```

R 자료형

✓ Matrix

| 연산자 | 의미 |
|-------------|---|
| $A + x$ | 행렬 A의 모든 값에 스칼라 x를 더한다. 이외에도 -, *, / 연산자를 사용할 수 있다. |
| $A + B$ | 행렬 A와 행렬 B의 합을 구한다. 행렬 간의 차는 - 연산자를 사용한다. |
| $A \%*\% B$ | 행렬 A와 행렬 B의 곱을 구한다. |

t : 행렬 또는 데이터 프레임의 전치 행렬을 구한다.

```
t(  
  x # 행렬 또는 데이터 프레임  
)
```

반환 값은 x의 전치 행렬이다.

solve : 수식 $a \%*\% x = b$ 에서 x를 구한다.

```
solve(  
  a, # 행렬  
  b  # 벡터 또는 행렬  
)
```

반환 값은 $a \%*\% x = b$ 에서 x다. b를 지정하지 않으면 a의 역행렬을 구한다.

R 자료형

✓ Matrix

nrow : 배열의 행의 수를 구한다.

```
nrow(  
  x # 벡터, 배열 또는 데이터 프레임  
)
```

반환 값은 x의 행의 수다.

dim : 객체의 차원 수를 구한다.

```
dim(  
  x # 행렬, 배열 또는 데이터 프레임  
)
```

반환 값은 x의 차원 수다.

ncol : 배열의 열의 수를 구한다.

```
ncol(  
  x # 벡터, 배열 또는 데이터 프레임  
)
```

반환 값은 x의 열의 수다.

dim<- : 객체의 차원 수를 지정한다.

```
dim(  
  x # 행렬, 배열 또는 데이터 프레임  
) <- value # 객체의 차원
```

```
> x <- matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9),  
nrow=3)  
> x * 2  
> x / 2  
> x - x  
> x + x  
> x %*% x  
> t(x)  
> (x <- matrix(c(1, 2, 3, 4), ncol=2))  
> solve(x)  
> (x <- matrix(c(1, 2, 3, 4, 5, 6), ncol=3))  
> nrow(x)  
> ncol(x)  
> dim(x)  
> dim(x) <- c(3, 2)  
> x
```

R 자료형

- ✓ Array - Matrix를 여러 층으로 쌓아 둔 형태, 3차원으로 데이터를 저장합니다.
동일한 데이터 유형만 저장

array : 배열을 생성한다.

```
array(  
  data=NA,          # 데이터를 저장한 벡터  
  dim=length(data), # 배열의 차원. 이 값을 지정하지 않으면 1차원 배열이 생성된다.  
  dimnames=NULL     # 차원의 이름  
)
```

반환 값은 배열이다.

```
> array1 <- array(c(1:12), dim=c(4, 3))  
> array1 <- array(c(1:12), dim=c(2, 2, 3))  
> array2[1,1,3] # 데이터 조회  
> array(1:12, dim=c(3, 4))  
> (x <- array(1:12, dim=c(2, 2, 3)))  
> x[, , 3]
```


R 자료형

- ✓ list - '(키, 값)' 형태의 데이터를 담는 연관 배열 Associative Array
다른 유형의 데이터를 저장 할 수 있다

list : 리스트 객체를 생성한다.

```
list(  
  key1=value1,  
  key2=value2,  
  ...  
)
```

반환 값은 key1에 value1, key2에 value2 등을 저장한 리스트다.

| 문법 | 의미 |
|--------|-------------------------|
| x\$key | 리스트 x에서 키 값 key에 해당하는 값 |
| x[n] | 리스트 x에서 n번째 데이터의 서브리스트 |
| x | 리스트 x에서 n번째 저장된 값 |

R 자료형

✓ list

```
> list1 <- list(name = 'park', address='seoul', tel='010-1234-5678', age=31)
> list1
> list1[1:2]
#특정 key만 조회하고 싶을 경우는 변수이름$key 형식으로 조회
> list1$birth <- '1988-10-23'
#list에 새로운 요소 추가생성
> list1$birth <- '1919-04-13'
> list1
#하나의 key에 두개 이상의 값을 추가 가능
> list1$name <-c('Korea', '대한민국')
> list1$name
#특정 요소를 삭제하려면 NULL을 할당하여 삭제할 수 있습니다.
➤ list1$birth<-NULL
> (x <- list(name="foo", height=70))
> (x <- list(name="foo", height=c(1, 3, 5)))
> list (a=list(val=c(1, 2, 3)), b=list(val=c(1, 2, 3, 4)))
> x[1]
```

- ✓ unlist - list를 character 벡터로 변환 , index가 없는 것은 그대로 유지되지만, index가 있는 것은 index 뒤에 순서대로 1, 2, 3, ...이 붙음

R 자료형

- ✓ data.frame - 엑셀의 표, 데이터베이스의 테이블 형태의 타입
다른 유형의 데이터를 저장 할 수 있다
ncol(dataframe)은 dataframe의 열의 개수를 리턴
nrow(dataframe)은 dataframe의 행의 개수를 리턴
names(dataframe) 은 dataframe의 열 이름을 출력
rownames(dataframe) 은 dataframe의 행 이름을 출력
colnames(dataframe) 은 dataframe의 열 이름을 출력

data.frame : 데이터 프레임을 생성한다.

str : 임의의 R 객체의 내부 구조(structure)를 보인다.

```
data.frame(  
  # value 또는 tag=value로 표현된 데이터 값. '...'은 가변 인자를 의미한다.  
  ...,  
  # 주어진 문자열을 팩터로 저장할 것인지 또는 문자열로 저장할 것인지를 지정하는 인자.  
  # 기본값은 보통 TRUE다. 따라서 이 인자를 지정하지 않으면 문자열은 팩터로 저장된다.  
  stringsAsFactors=default.stringsAsFactors()  
)
```

```
str(  
  object # 구조를 살펴볼 R 객체  
)
```

반환 값은 데이터 프레임이다.

R 자료형

✓ data.frame

| 문법 | 의미 |
|-----------------|--|
| d\$colname | 데이터 프레임 d에서 컬럼 이름이 colname인 데이터를 접근한다. |
| d\$colname <- y | 데이터 프레임 d에서 컬럼 이름이 colname인 컬럼에 데이터 y를 저장한다. 만약 colname이 d에 없는 새로운 이름이라면 새로운 컬럼이 추가된다. |

```
> (d <- data.frame(x=c(1, 2, 3, 4, 5), y=c(2, 4, 6, 8, 10), z=c('M', 'F', 'M', 'F', 'M')))  
> str(d)  
> d$x  
> d$x <- 6:10  
> d  
> d$w <- c("A", "B", "C", "D", "E")  
> d  
> str(d)  
> (x <- data.frame(1:3))  
> colnames(x) <- c('val')  
> x  
> rownames(x) <- c('a', 'b', 'c')  
> x
```

R 자료형

✓ data.frame

| 문법 | 의미 |
|---------------------------------|--|
| <code>d[m, n, drop=TRUE]</code> | 데이터 프레임 d의 m행 n 컬럼에 저장된 데이터. m과 n을 벡터로 지정하여 다수의 행과 컬럼을 동시에 가져올 수 있으며 m, n에는 색인뿐만 아니라 행 이름이나 컬럼 이름을 지정할 수 있다. 만약 m, n 중 하나를 생략하면 모든 행 또는 컬럼의 데이터를 의미한다. <code>d[, n]</code> 과 같이 행을 지정하지 않고 특정 컬럼만 가져올 경우 반환 값은 데이터 프레임이 아니라 해당 컬럼의 데이터 타입이 된다. 이러한 형 변환을 원하지 않으면 <code>drop=FALSE</code> 를 지정하여 데이터 프레임을 반환하도록 할 수 있다.. |

```
> sales1 <- matrix(c(1, 'Apple', 500, 5, 2, 'Peach', 200, 2, 3, 'Banana', 100, 4, 4, 'Grape', 50, 7) , nrow=4, byrow=T)
> df1 <- data.frame(sales1)
> names(df1) <- c('NO', 'NAME', 'PRICE', 'QTY')
> df1 <- rbind(data.frame(NO=5, NAME='mongo', PRICE=700, QTY=10))
#행렬로 데이터 프레임 생성
> sales2<-matrix(c(1,' Apple', 500, 5,
> 2, 'Peach', 200, 2,
> 3, 'Banana', 100, 4,
> 4, 'Grape', 50, 7), nrow=4, byrow=T)
> df1<-data.frame(sales2)
> names(df1) <- c('NO','NAME', 'PRICE', 'QTY')
#matrix를 사용하여 생성하는 경우 모든 컬럼의 데이터형이 동일하게 자동으로 변형
```

R 자료형

✓ data.frame

```
#dataframe에서 원하는 데이터만 조회- 변수$라벨명
sales
sales$NAME
sales[1,3]
sales[1,]
sales[,3]
sales[c(1,2)]
sales[,c(1,2)]
sales[,c(1:3)]
#특정 조건에 맞는 데이터만 조회 -subset()
subset(sales, qty<5)
subset(sales, price==200)
subset(sales, name==Apple)
subset(sales, name=='Apple')
#데이터 프레임 합치기 - rbind(), cbind() 행과 열의 개수가 같아야만 가능
#행과 열의 개수가 다른 경우 merge() 가능
no<-c(1,2,3)
name<-c('Apple','Peach','Banana')
price<-c(500,200, 100)
df1<-data.frame(NO=no, NAME=name, PRICE=price)
```

R 자료형

✓ data.frame

```
no<-c(10,20,30)
name<-c('train','car', 'airplane')
price<-c(1000,2000, 3000)
df2<-data.frame(NO=no, NAME=name, PRICE=price)
df3<-cbind(df1, df2)
df4<-rbind(df1, df2)

df1 <- data.frame(name=c('apple', 'banana', 'cherry'), price=c(300, 200, 100))
df2 <- data.frame(name=c('apple', 'cherry', 'berry'), qty=c(10, 20, 30))

merge(df1, df2)      #df1기준으로 df2와 공통으로 있는 name 컬럼 데이터를 출력함
merge(df1, df2, all=T)  # 데이터가 없는 것도 모두 나오게 all=T 옵션 지정함

#데이터 프레임에 행과 열 추가하기
new<-data.frame(name="mango", price=400)
df1<-rbind(df1, new)
df1<-rbind(df1, data.frame(name='berry', price=500))
df1<-cbind(df1, data.frame(qty=c(10, 20, 30, 40, 50)))
```

R 자료형

✓ data.frame

```
#데이터 프레임에서 특정 컬럼만 골라내서 새로운 형태 만들기
no<-c(1,2,3,4,5)
name<-c('나영석','정재승','김영하','유시민','유희열')
address<-c("서울", "대전", "포항", "광주","제주")
tel<-c(1111,2222,3333,4444,5555)
hobby<-c('독서','미술','영화감상','등산','수영')
member<-data.frame(NO=no, NAME=name, ADDRESS=address, TEL=tel, HOBBY=hobby)

member2<-subset(member, select=c(NO, NAME, TEL))

member3<-subset(member, select=-TEL)

colnames(member3)<-c("번호", "이름", "주소", "취미")
```


R 자료형

✓ 유틸함수

head : 객체의 처음 부분을 반환한다.

```
head(  
  x,      # 객체  
  n=6L    # 반환할 결과 값의 크기  
)
```

반환 값은 x의 앞부분을 n만큼 잘라낸 데이터다.

tail : 객체의 뒷부분을 반환한다.

```
tail(  
  x,      # 객체  
  n=6L    # 반환할 결과 값의 크기  
)
```

반환 값은 x의 뒷부분을 n만큼 잘라낸 데이터다.

View : 데이터 뷰어를 호출한다.

```
View(  
  x,      # 데이터 프레임으로 강제 형 변환한 뒤 뷰어로 볼 데이터  
  title   # 뷰어 윈도우의 제목  
)
```

```
> d <- data.frame(x=1:1000)  
> d  
> head(d)  
> tail(d)  
> View(d)
```

R 자료형

✓ 타입 판별

| 함수 | 의미 |
|------------------|-------------------------|
| class(x) | 객체 x의 클래스 |
| str(x) | 객체 x의 내부 구조 |
| is.factor(x) | 주어진 객체 x가 팩터인가 |
| is.numeric(x) | 주어진 객체 x가 숫자를 저장한 벡터인가 |
| is.character(x) | 주어진 객체 x가 문자열을 저장한 벡터인가 |
| is.matrix(x) | 주어진 객체 x가 행렬인가 |
| is.array(x) | 주어진 객체 x가 배열인가 |
| is.data.frame(x) | 주어진 객체 x가 데이터 프레임인가 |

```
> class(c(1, 2))
> class(matrix(c(1, 2)))
> class(data.frame(x=c(1, 2), y=c(3, 4)))
#class( )는 문자열로 데이터 타입을 반환
#벡터에 저장된 데이터 타입에 class의 리턴 값
#은 logical, character, factor 등이 될 수 있다
> str(c(1, 2))
> str(matrix(c(1,2)))
> str(list(c(1,2)))
> str(data.frame(x=c(1,2)))
> is.factor(factor(c("m", "f")))
> is.numeric(1:5)
> is.character(c("a", "b"))
> is.data.frame(data.frame(x=1:5))
```

R 자료형

✓ 타입 변환

| 함수 | 의미 |
|------------------|---------------------------|
| as.factor(x) | 주어진 객체 x를 팩터로 변환 |
| as.numeric(x) | 주어진 객체 x를 숫자를 저장한 벡터로 변환 |
| as.character(x) | 주어진 객체 x를 문자열을 저장한 벡터로 변환 |
| as.matrix(x) | 주어진 객체 x를 행렬로 변환 |
| as.array(x) | 주어진 객체 x를 배열로 변환 |
| as.data.frame(x) | 주어진 객체 x를 데이터 프레임으로 변환 |

```
> x <- c("a", "b", "c")
> as.factor(x)
> as.character(as.factor(x))
> x <- matrix(1:9, ncol=3)
> as.data.frame(x)
> (x <- data.frame(matrix(c(1, 2, 3, 4), ncol=2)))
> data.frame(list(x=c(1, 2), y=c(3, 4)))
> as.factor(c("m", "f"))
> factor(c("m", "f"), levels=c("m", "f"))
```

R 유용한 패키지

❑ stringr 패키지

```
install.packages("stringr")
library(stringr)
#str_detect() - 주어진 데이터들에서 특정 문자가 있는지를 검사해 TRUE/FALSE를 출력
fruits <- c('apple', 'Apple', 'banana', 'pineapple')
str_detect(fruits, 'A')
str_detect(fruits, '^a')
str_detect(fruits, 'a$')
str_detect(fruits, '^[aA]')
str_detect(fruits, '[aA]')
#str_count() 주어진 단어에서 해당 글자가 몇번 나오는지 알려주는 함수
str_count(fruits, "a")
str_count(fruits, "p")
str_count(fruits, "e")
str_count(fruits, c("a", "b", "p", "p"))
#str_c() - paste()처럼 주어진 문자열을 합쳐서 출력하는 함수
str_c('apple', 'banana')
str_c('fruits:', fruits)
str_c(fruits, "name is ", fruits)
str_c(fruits, collapse="")
str_c(fruits, collapse="-")
```

R 유용한 패키지

❑ stringr 패키지

```
#str_dup() - 주어진 문자열을 주어진 횟수만큼 반복해서 출력하는 함수
str_dup(fruits, 3)
#str_length() - 주어진 문자열의 길이를 출력하는 함수
str_length('apple')
str_length(fruits)
#str_locate() - 주어진 문자열에서 특정 문자가 처음 나오는 위치와 마지막 나오는 위치를 알려줌
str_locate('apple', 'a')
str_locate(fruits, 'a')
#str_replace() - 주어진 문자열에서 변경전 문자를 변경 후 문자로 바꾸는 함수
str_replace('apple', 'p','*')
str_replace('apple', 'p','**')
str_replace_all('apple', 'p','*')
#str_split() - 주어진 데이터셋을 지정된 기호로 분리를 하는 함수
fruits <- str_c('apple', '/', 'orange', '/', 'banana')
str_split(fruits, '/')
#str_sub() - 주어진 문자열에서 지정된 길이 만큼의 문자를 잘라내는 함수
str_sub(fruits, start=1, end=3)
str_sub(fruits, start=6, end=9)
str_sub(fruits, start=-5) #-는 뒤에서부터 시작합니다.
#str_trim() - 문자열의 가장 앞과 가장 뒤에서 공백이 있을 경우 제거해 주는 함수
str_trim(" apple banana berry ")
```

R 프로그래밍

✓ R 프로그래밍 특성

- 다른 프로그래밍 언어에서는 for 문을 사용해 데이터를 한 행씩 읽어들이고 처리하고, R에서는 전체 데이터를 한 번에 다루는 벡터 연산을 더 자주 사용한다.
- R에는 초기화되지 않은 변수에 저장하는 NULL 외에도 관측 또는 기록되지 않은 데이터를 표시하기 위한 NA가 있다. 코드에서 NA를 제대로 구분해서 처리하지 않으면 원하는 결과를 얻지 못할 수도 있다.
- R 객체는 그 값을 수정할 수 없다. 문법적으로 객체의 데이터를 수정하고 있다고 보이는 코드에서도 실제로는 해당 부분의 값이 수정된 새로운 객체가 생성된다.

R 프로그래밍

✓ if

```
if (cond) {  
  cond가 참일 때 실행할 문장  
} else {  
  cond가 거짓일 때 실행할 문장  
}
```

`ifelse` : 주어진 test 값에 따라 yes 또는 no 값을 반환한다.

```
ifelse(  
  test, # 참, 거짓을 저장한 객체  
  yes, # test가 참일 때 선택할 값  
  no # test가 거짓일 때 선택할 값  
)
```

test에 다수의 TRUE, FALSE가 저장되어 있을 때 TRUE에 대해서는 yes 값을, FALSE에 대해서는 no 값을 선택하여 반환한다.

```
> x <- c(1, 2, 3, 4, 5)  
> ifelse(x %% 2 == 0, "even", "odd")
```

R 프로그래밍

✓ 반복문

```
for (i in data) {  
  i를 사용한 문장  
}  
  
while (cond) {  
  조건이 참일 때 수행할 문장  
}  
  
repeat {  
  반복해서 수행할 문장  
}
```

```
> for (i in 1:10) {  
+   print(i)  
+ }
```

- 반복문 내 블록에서는 break, next 문을 사용해 반복의 수행을 조정할 수 있다.
- break : 반복문을 종료한다.
- next : 현재 수행 중인 반복문 블록의 수행을 중단하고 다음 반복을 시작한다.

```
> i <- 1  
> while (i <= 10) {  
+   print(i)  
+   i <- i + 1  
+ }
```

```
> i <- 1  
> repeat {  
+   print(i)  
+   if (i >= 10) {  
+     break  
+   }  
+   i <- i + 1  
+ }
```

```
> i <- 0  
> while (i <= 9) {  
+   i <- i + 1  
+   if (i %% 2 != 0) {  
+     next # print()를 실행하지 않고 while 문의 처음으로 감  
+   }  
+   print(i)  
+ }
```


R 프로그래밍 요소

- ✓ 제어문, 반복문 연습 문제
 - while 문 이용 1~10사이의 홀수 출력
 - for 문 이용 1~10까지의 누적합 출력

R 프로그래밍

✓ NA 처리

| 함수 | 의미 |
|-------------------------|--|
| na.fail(object, ...) | object에 NA가 포함되어 있으면 실패한다. |
| na.omit(object, ...) | object에 NA가 포함되어 있으면 이를 제외한다. |
| na.exclude(object, ...) | object에 NA가 포함되어 있으면 이를 제외한다는 점에서 na.omit과 동일하다. 그러나 naresid, napredict를 사용하는 함수에서 NA로 제외한 행을 결과에 다시 추가한다는 점이 다르다. |
| na.pass(object, ...) | object에 NA가 포함되어 있더라도 통과시킨다. |

```
> NA & TRUE
> NA + 1
> (x <- data.frame(a=c(1, 2, 3), b=c("a", NA, "c"), c=c("a", "b", NA)))
> na.fail(x)    # NA가 포함되어 있으므로 실패
> na.omit(x)    # NA가 포함된 행을 제외
> na.exclude(x) # NA가 포함된 행을 제외
> na.pass(x)    # NA의 여부에 상관없이 통과
```

R 프로그래밍

✓ 가변 길이 인자함수

```
> f <- function(...) {  
+   args <- list(...)  
+   for (a in args) {  
+     print(a)  
+   }  
+ }  
> f('3', '4')  
  
> g <- function(z, ...) {  
+   print(z)  
+   f(...)  
+ }  
> g(1, 2, 3)
```

✓ 중첩 함수

- 중첩 함수를 사용하면 함수 안에서 반복되는 동작을 한 함수로 만들고 이를 호출하여 코드를 간결하게 표현할 수 있다
- 내부 함수가 외부 함수에 정의된 변수를 접근할 수 있어 클로저closure로 사용할 수 있다.

```
> f <- function(x, y) {  
+   print(x)  
+   g <- function(y) {  
+     print(y)  
+   }  
+   g(y)  
+ }  
> f(1, 2)
```

```
> f <- function(x1) {  
+   return(function(x2) {  
+     return(x1 + x2)  
+   })  
+ }  
> g <- f(1)  
> g(2) # x1 = 1, x2 = 2  
> g2 <- f(2)  
> g2(2) # x1 = 2, x2 = 2
```

R 프로그래밍

✓ 스코프

- 코드에 기술한 이름(예를 들면, 변수명)이 어디에서 사용 가능한지를 정하는 규칙
- R에서는 문법적 스코프lexical scope(정적 스코프static scope)를 사용
- 문법적 스코프는 변수가 정의된 블록 내부에서만 변수를 접근할 수 있는 규칙
- 콘솔에서 변수를 선언하면 모든 곳에서 사용 가능한 전역 변수가 된다.
- 전역 변수는 현재 실행 중인 R 세션 동안 유효하다.
- 변수는 내부 블록에서만 접근할 수 있으므로 함수 내부에 정의한 이름은 함수 바깥에서 접근할 수 없다

```
> n <- 1
> f <- function() {
+   print(n)
+ }
> f()
> n <- 2
> f()
> n <- 100
> f <- function() {
+   n <- 1
+   print(n)
+ }
> f()
```

```
> rm(list = ls())
> f <- function() {
+   print(n)
+ }
> f()
> rm(list = ls())
> f <- function() {
+   n <- 1
+ }
> f()
```

```
> n
> n <- 100
> f <- function(n) {
+   print(n)
+ }
> f(1)
> f <- function(x) {
+   a <- 2
+   g <- function(y) {
+     print(y + a)
+   }
+   g(x)
+ }
> f(1)
```

```
> a <- 100
> f <- function(x) {
+   g <- function(y) {
+     print(y + a)
+   }
+   g(x)
+ }
> f(1)
```

R 프로그래밍

✓ 스코프

- 함수 내부에서 전역 변수와 같은 이름의 지역 변수를 사용하면, 함수 내부의 지역 변수가 우선한다.
- 변수는 내부 블록에서만 접근할 수 있으므로 함수 내부에 정의한 이름은 함수 바깥에서 접근할 수 없다.
- 함수 내에서 이름은 함수 안의 변수들로부터 먼저 찾는다.
- 함수 인자의 변수명 역시 전역 변수보다 우선한다.
- 내부 함수에서 외부함수 안의 변수 또는 전역 변수에 값을 할당하려면 <<-를 사용해야 한다.

```
> f <- function() {  
+   a <- 1  
+   g <- function() {  
+     a <- 2  
+     print(a)  
+   }  
+   g()  
+   print(a)  
+ }  
> f()
```

```
> b <- 0  
> f <- function() {  
+   a <- 1  
+   g <- function() {  
+     a <<- 2  
+     b <<- 2  
+     print(a)  
+     print(b)  
+   }  
+   g()  
+   print(a)  
+   print(b)  
+ }  
> f()
```

R 함수 정의

✓ 함수 연습 문제

- 입력된 숫자가 양수이든 음수이든 양수로 출력하기
- 입력된 숫자가 양수이면 2배로 만들어서 출력하고 숫자가 0보다 작거나 같으면 0으로 만들어서 결과를 출력하기
- 입력된 숫자가 0보다 크면 2배의 값을 출력하고 0일 경우 0을 출력하고, 0보다 작을 경우 -2배의 값을 출력

R 프로그래밍

- ✓ 객체의 불변성
 - 수정 불가

```
> a <- list()
> a$b <- c(1, 2, 3)
# 리스트 a를 복사한 새로운 객체 a'을 만들고, 이 a'에 필드 b를 추가하고 해당 필드에 c(1, 2, 3)을 채워넣은 다음,
변수명 a가 a'을 가리키도록 하는 것이다.
```

tracemem : 객체의 복사를 추적한다.

```
tracemem(
  x # 추적할 R 객체
)
```

untracemem : 객체 복사 추적을 중단한다.

```
untracemem(
  x # 추적을 중단할 R 객체
)
```

```
> a <- list()
> tracemem(a)
> a$b <- c(1, 2, 3)
> untracemem(a)
```

R 프로그래밍

✓ 객체의 불변성

- 객체가 불변이라 벡터 기반 연산을 사용하는 것이 for 등의 반복문을 사용하는 것보다 효율적
- for 문 안에서 벡터의 인자를 하나씩 바꾸는 코드는 벡터의 i번째 값을 1씩 증가시킬 때마다 i번째 값이 수정된 벡터를 매번 새로 만들어 벡터에 할당한다.

```
> v <- 1:1000
> for (i in 1:1000) {
+ v[i] <- v[i] + 1 # i 번째 값을 바꿀 때마다 새로운 벡터가 생성된다!
+ }

# v 안에 있는 전체 값을 1만큼 증가시킨 객체를 한 개 만든 다음, 이를 v에 할당한다.
> v <- 1:1000
> v <- v + 1

> rm(list = ls()) # 메모리에 있는 객체들을 삭제
> gc()           # 가비지 컬렉션(Garbage Collection)을 수행해 사용하지 않는 메모리를 해제
> v <- 1:99999999
> for (i in 1:99999999) {
+   for (j in 1:99999999) {
+     v[j] <- v[j] + 1
+   }
+ }
```


R 프로그래밍

✓ 모듈 패턴

- 모듈이란 외부에서 접근할 수 없는 데이터와 그 데이터를 제어하기 위한 함수로 구성된 구조물
- 패턴이란 정형화된 코딩 기법

✓ 모듈 패턴의 장점

- 데이터를 외부에서 직접 접근할 수 없게 되어 내부 구현이 숨겨진다.
- 사전에 정의된 함수로만 데이터를 다룰 수 있게 되어 데이터의 내부 구조를 잘 모르는 사용자가 데이터를 잘못 건드려서 손상시키는 일을 막을 수 있다.
- 모듈의 사용자는 내부 구조는 건드릴 수 없고 외부로 노출된 함수만 불러 쓰고 있는 상태가 되므로, 모듈의 제작자는 해당 함수가 이전과 같은 결과를 내놓기만 한다면 함수의 내부 구조나 데이터 구조를 마음대로 바꿀 수 있다.

R 프로그래밍

- ✓ 큐Queue는 먼저 들어온 데이터를 먼저 처리FIFO, First In First Out하는 데 사용하는 자료 구조
 - Enqueue : 줄의 맨 뒤에 데이터를 추가한다.
 - Dequeue : 줄의 맨 앞에 있는 데이터를 가져온다. 가져온 데이터는 줄에서 빠진다.
 - Size : 줄의 길이, 즉 자료 구조 내에 저장된 데이터의 수를 반환한다.

```
> q <- c()
> q_size <- 0
> enqueue <- function(data) {
+ q <- c(q, data)
+ q_size <- q_size + 1
+ }
> dequeue <- function() {
+ first <- q[1]
+ q <- q[-1]
+ q_size <- q_size - 1
+ return(first)
+ }
> size <- function() {
+ return(q_size)
+ }
```

```
> enqueue(1)
> enqueue(3)
> enqueue(5)
> print(size())
> print(dequeue())
> print(dequeue())
> print(dequeue())
> print(size())
q라는 변수가 전역으로 선언되어 있으므로 이 함수를 거
치지 않고 외부에서 데이터를 조작해버릴 수 있다.
> enqueue(1)
> q <- c(q, 5) # 전역 변수 q를 조작
> print(size())
> dequeue()
> dequeue()
> size()
```

R 프로그래밍

✓ 큐Queue 모듈

```
> queue <- function() {  
+   q <- c()  
+   q_size <- 0  
+  
+   enqueue <- function(data) {  
+     q <- c(q, data)  
+     q_size <- q_size + 1  
+   }  
+  
+   dequeue <- function() {  
+     first <- q[1]  
+     q <- q[-1]  
+     q_size <- q_size - 1  
+     return(first)  
+   }  
+  
+   size <- function() {  
+     return(q_size)  
+   }  
+  
+   return(list(enqueue=enqueue, dequeue=dequeue, size=size))  
+ }
```

```
> q <- queue()  
> q$enqueue(1)  
> q$enqueue(3)  
> q$size()  
> q$dequeue()  
> q$dequeue()  
> q$size()  
> q <- queue()  
> r <- queue()  
> q$enqueue(1)  
> r$size()  
> r$enqueue(3)  
> q$dequeue()  
> r$dequeue()  
> q$size()  
> r$size()
```

R 프로그래밍

- ✓ R에서는 주로 벡터 기반으로 데이터를 처리한다.
- ✓ 벡터 기반 처리는 개별 요소를 for 루프 등으로 하나씩 처리하는 방식보다 빠르게 수행될 뿐만 아니라 손쉽게 병렬화가 가능하다
- ✓ 데이터 처리 및 기계 학습 기법의 예제로 사용할 아이리스(붓꽃)iris 데이터 셋

| 컬럼명 | 의미 | 데이터 타입 |
|--------------|--|--------|
| Species | 붓꽃의 종. setosa, versicolor, virginica 세 가지 값 중 하나 | Factor |
| Sepal.Width | Sepal.Width | Number |
| Sepal.Length | 꽃받침의 길이 | Number |
| Petal.Width | 꽃잎의 너비 | Number |
| Petal.Length | 꽃잎의 길이 | Number |

```
> head(iris)
> str(iris)
```

R 프로그래밍

- ✓ datasets 패키지에 있는 데이터 셋은 R에 기본적으로 포함된 데이터들
- ✓ 데이터의 목록은 `library(help=datasets)` 명령으로 확인

| 데이터 셋 | 의미 |
|--------------|--|
| AirPassenger | 1949년부터 1960년까지의 항공기 승객 수 |
| airquality | 973년 5월부터 9월까지의 뉴욕 대기 오염 정도에 대한 기록 |
| cars | 자동차의 주행 속도에 따른 제동 거리 |
| mtcars | 1974년 미국 모터 트렌드 매거진에 실린 32개 자동차에 대해 연료 효율을 비롯한 10여 가지 특징을 기록 |
| Titanic | 타이타닉 호의 생존자 정보를 호실(1등실, 2등실, 3등실), 성별, 나이, 생존 여부로 정리 |
| InsectSprays | 6종류의 살충제를 사용했을 때 각 살충제에 대해 살아남은 벌레의 수 |
| Orange | 오렌지 나무의 종류, 연령, 둘레 |
| swiss | 1888년경 프랑스어를 사용하는 스위스 내 47개 주의 출산율과 사회 경제적 지표(농업 종사자 비율, 군 입대 시험 성적, 교육 등) |

R 프로그래밍

- ✓ 데이터를 사용할 때는 'data(데이터 셋 이름)' 명령을 사용한다

```
> data(mtcars)  
> head(mtcars)
```

R 기본 함수 - 파일 읽기/쓰기

- ❑ `list.files()` - 파일 이름 확인하기
- ❑ `list.files(recursive=T)` - 하위 디렉터리 내용까지 전부 출력
- ❑ `list.files(all.files=T)` - 숨김 파일까지 전부 출력
- ❑ `scan()` - 텍스트 파일 읽어서 배열에 저장하기
- ❑ `readline()` - 한 줄 읽어 들이기
- ❑ `readLines()` - 파일 읽어 들어서 배열에 담기
- ❑ `read.table()` - 일반 텍스트 형태의 파일을 읽어서 데이터 프레임에 담기
- ❑ `read.csv()` - csv 파일 읽어서 데이터 프레임에 저장
- ❑ `read.csv.sql()` - googleVis 패키지를 이용하여 원하는 데이터를 SQL 쿼리로 불러오기
- ❑ `write.table()` - 파일에 저장
- ❑ `WriteXSL()` - WriteXLS 패키지를 설치하고, excel 파일 형식으로 저장
- ❑ `write.xml()` - XML 패키지를 설치하고, xml형식으로 저장

R 기본 함수 - 파일 읽기/쓰기

`read.csv` : CSV 파일을 데이터 프레임으로 읽어들이는다.

```
read.csv(  
  file,          # 파일명  
  header=FALSE, # 파일의 첫 행을 헤더로 처리할 것인지 여부  
  # 데이터에 결측치가 포함되어 있을 경우 R의 NA에 대응시킬 값을 지정한다.  
  # 기본값은 "NA"로, "NA"로 저장된 문자열들은 R의 NA로 저장된다.  
  na.strings="NA",  
  # 문자열을 팩터로 저장할지 또는 문자열로 저장할지 여부를 지정하는 데 사용한다. 별다른  
  # 설정을 하지 않았다면 기본값은 보통 TRUE다.  
  stringsAsFactors=default.stringsAsFactors()  
)
```

반환 값은 데이터 프레임이다.

`write.csv` : 데이터 프레임을 CSV로 저장한다.

```
write.csv(  
  x,          # 파일에 저장할 데이터 프레임 또는 행렬  
  file="",    # 데이터를 저장할 파일명  
  row.names=TRUE # TRUE면 행 이름을 CSV 파일에 포함하여 저장한다.  
)
```


R 기본 함수 - 파일 읽기/쓰기

a.csv 파일

```
id,name,score  
1,"Mr. Foo",95  
2,"Ms. Bar",97  
3,"Mr. Baz",92
```

```
> (x <- read.csv("a.csv"))  
> str(x)  
)
```

b.csv 파일

```
1,"Mr. Foo",95  
2,"Ms. Bar",97  
3,"Mr. Baz",92
```

```
> (x <- read.csv("b.csv"))  
> names(x) <- c("id", "name", "score")  
> x  
> str(x)  
> x$name <- as.character(x$name)  
> str(x)  
# 문자열을 팩터가 아니라 문자열 타입으로 읽도록  
stringsAsFactors=FALSE를 지정  
> x <- read.csv("a.csv", stringsAsFactors=FALSE)  
> str(x)
```

R 기본 함수 - 파일 읽기/쓰기

c.csv 파일

id,name,score

1,"Mr. Foo",95

2,"Ms. Bar",NIL

3,"Mr. Baz",92

```
> (x <- read.csv("c.csv"))
```

```
> str(x)
```

#데이터를 read.csv()로 읽어들이면 NIL이 문자열이므로 팩터로 저장된다. 그 결과 score 컬럼 전체가 팩터가 된다.

na.strings 인자 의 기본값은 "NA"로, "NA"라는 문자열이 주어진다면 이를 R이 인식하는 NA로 바꿔준다.

na.strings에 지정하는 값은 벡터므로 여러 문자열을 벡터로 지정하면 벡터 내 모든 문자열이 NA로 저장된다.

```
> x <- read.csv("c.csv", na.strings=c("NIL"))
```

```
> str(x)
```

```
> is.na(x$score)
```

```
> write.csv(x, "d.csv", row.names=FALSE)
```

R 기본 함수 - 파일 읽기/쓰기

- ✓ 바이너리 파일로 R 객체를 저장하고 불러들이는 함수 - `save()`, `load()`

`save` : 메모리에 있는 객체를 파일에 저장한다.

```
save(  
  ...,          # 저장할 객체의 이름  
  list=character(), # 저장할 객체의 이름을 벡터로 지정할 경우 ... 대신 사용  
  file          # 파일명  
)
```

`load` : 파일로부터 객체를 메모리로 읽어들인다.

```
load(  
  file # 파일명  
)
```

반환 값은 파일에서 읽어들인 객체의 이름들을 저장한 벡터다.

R 기본 함수 - 파일 읽기/쓰기

- ✓ 바이너리 파일로 R 객체를 저장하고 불러들이는 함수 - `save()`, `load()`

```
> x <- 1:5
> y <- 6:10
> save(x, y, file="xy.RData")

> rm(list=ls()) # 메모리에 있는 객체를 모두 삭제
> a <- 1:5
> b <- 6:10
> c <- 11:15
> save(list=ls(), file="abc.RData")

> rm(list=ls())
> ls()
> load("abc.RData")
> ls()
> a
> b
> c
```

R 기본 함수 – 파일 읽기/쓰기

- ✓ RODBC와 XLConnect 패키지를 설치하여 xls파일로부터 데이터를 읽어 데이터 프레임에 저장할 수 있다.
 - ✓ XML 패키지를 이용하여 html페이지에서 표 형식의 데이터를 읽어 올 수 있다
 - ✓ 64-bit Oracle Data Access Components(ODAC)를 설치해야 Oracle DBMS에 접속해서 데이터를 가져올 수 있다.
 - ✓ RODBC 패키지를 설치하고 mysql에 접속해서 데이터를 가져올 수 있다.
-
- ✓ 텍스트 파일을 읽어서 배열에 저장하기 – scan()

```
#setwd("c://workspaceR") #데이터가 있는 작업용 디렉토리 설정
getwd()
scan1 <- scan("./lab01/data/data1.txt") # 텍스트 파일읽어서 저장
# 실수와 문자는 호출할 때 what이라는 옵션을 사용해야 한다.
scan1 <- scan("./lab01/data/data1.txt", what="")
scan2 <- scan("./lab01/data/data2.txt", what="")
```

R 기본 함수 - 파일 읽기/쓰기

- ✓ scan()함수로 텍스트 파일 읽어서 배열에 저장하기

```
# 사용자로부터 숫자 입력받기
input1<-scan()
input1
# 사용자로부터 문자 입력받기
input2<-scan(what="")
```

- ✓ readline() - 한줄 단위로 읽어들이는 함수

```
input3 <- readline()
input3
input4 <- readline("Are you OK?")
Input4

#파일에서 행단위로 읽어 들여서 배열에 담기
input5 <- readLines("./data/scan_4.txt")
input5
```

R 기본 함수 - 파일 읽기/쓰기

✓ read.table()

```
#텍스트 형태의 파일을 읽어서 데이터 프레임에 담기
fruits<-read.table("./data/fruits.txt")
fruits
fruits<-read.table("./data/fruits.txt", header=T)
fruits
fruits2<-read.table("./data/fruits_2.txt", header=T) #주석은 자동 제외
fruits2
fruits2<-read.table("./data/fruits_2.txt", skip=2) #건너 뛴 라인 수를 지정
fruits2
fruits2<-read.table("./data/fruits_2.txt", nrow=2) #출력할 줄 수를 지정
fruits2
fruits3<-read.table("./data/fruits.txt", header=T, nrow=2) #주석은 자동 제외
fruits3
fruits3<-read.table("./data/fruits.txt", header=T, skip=2, nrow=2) #주석은 자동 제외
fruits3
```

R 기본 함수 - 파일 읽기/쓰기

- ✓ xls(microsoft office excel)파일로부터 데이터 프레임에 저장하기
- ✓ 32bit면 RODBC 패키지를 설치하고, 64bit이면 XLConnect 패키지 설치

```
install.packages("RODBC")
library(RODBC)
excel = odbcConnectExcel("fruits_6.xls", readOnly=F)
data1 = sqlTables(excel)
data1

data2 = sqlQuery(excel, "select * from [Sheet1$]", as.is=T)
data2

install.packages("XLConnect")
library(XLConnect)
excel = loadWorkbook("./lab01/data/fruits_6.xls", create=F)
data1 = readWorksheet(excel, sheet="sheet1", startRow=1, endRow=8, startCol=1, endCol=5)
data1
```


R 기본 함수 - 파일 읽기/쓰기

✓ readHTMLTable()

```
#html 페이지에서 표 형식의 데이터 가져오기
install.packages("XML")
library(XML)
pop<-"https://en.wikipedia.org/wiki/World_population"
pop
pop_table <- readHTMLTable(pop) #페이지로부터 table을 읽어 들임
length(pop_table)
pop_table <-readHTMLTable(pop, which=4) # 4번째 테이블을 읽어 들임
pop_table
pop_table[, c(2,3)]
```

R 기본 함수 - 파일 읽기/쓰기

✓ #excel 파일 형식으로 저장하기

```
#excel 파일 형식으로 저장하기
install.packages("WriteXLS")
library(WriteXLS)
name<-c("Apple", "Banana", "Peach")
price<-c(300, 200, 100)
item<-data.frame(NAME=name, PRICE=price)
item
#Perl이 설치되어 있어야 합니다.
http://www.activestate.com/activeperl/downloads에서 ActivePer 윈도우 버전에 맞게 다운로드
WriteXLS(item, "./lab01/output/item.xls")
```

R 기본 함수 - 파일 읽기/쓰기

- ✓ mysql 로 접속해서 데이터를 가져옵니다.

```
install.packages("RMySQL")
install.packages("dbConnect")
library(RMySQL)
library(dbConnect)
mysqlconnection = dbConnect(MySQL(), user = 'root', password = 'mysql5',
                             dbname = 'sakila', host = 'localhost')
# List the tables available in this database.
dbListTables(mysqlconnection)
# Query the "actor" tables to get all the rows.
## my changes
con = dbConnect(MySQL(), user = 'root', password = 'dlavorxmfdls',
                 dbname = 'sakila', host = 'localhost')

dbListTables(con)
myQuery <- "select * from actor;"
dbGetQuery(con, myQuery)
result <- dbGetQuery(con, myQuery)
View(result)
dbDisconnect(mysqlconnection)
result = dbSendQuery(mysqlconnection, "select * from actor")
# Store the result in a R data frame object. n = 5 is used to fetch first 5 rows.
data.frame = fetch(result, n = 5)
```

R 기본 함수 - 파일 읽기/쓰기

✓ XML 형식으로 저장하기

```
#XML 형식으로 저장하기
install.packages("XML")
library(XML)
install.packages("kulife")
library(kulife)
name<-c("Apple", "Banana", "Peach")
price<-c(300, 200, 100)
item<-data.frame(NAME=name, PRICE=price)
item
write.xml(item, "./lab01/output/item.xls")
```

R 기본 함수 - 파일 읽기/쓰기

✓ XML 형식으로 저장하기

```
library(XML)
library("methods")
result <- xmlParse(file = "xmlfile.xml")
result

# Extract the nodes
rootnode <- xmlRoot(result)
rootnode
rootnode[1]
rootnode[2]
rootnode[[1]][[2]]

# Find number of nodes in the root.
rootsize <- xmlSize(rootnode)
rootsize

# Convert the input xml file to a data frame.
xmldataframe <- xmlToDataFrame("xmlfile.xml")
print(xmldataframe)
```

R 기본 함수 - 파일 읽기/쓰기

✓ Mongodb 연동

```
install.packages("mongolite")
library(mongolite)
install.packages("ggplot2"); library(ggplot2); diamonds
m <- mongo(collection = "diamonds")
# Insert test data
data(diamonds, package="ggplot2")
m$insert(diamonds)
# Check records
m$count()
nrow(diamonds)
# Perform a query and retrieve data
out <- m$find('{"cut" : "Premium", "price" : { "$lt" : 1000 } }')
# Compare
nrow(out)
nrow(subset(diamonds, cut == "Premium" & price < 1000))
# Cross-table
tbl <- m$mapreduce(
  map = "function(){emit({cut:this.cut, color:this.color}, 1)}",
  reduce = "function(id, counts){return Array.sum(counts)}"
)
```

R 기본 함수 - 파일 읽기/쓰기

✓ MongoDB 연동

```
# Same as:
data.frame(with(diamonds, table(cut, color)))
# Stream jsonlines into a connection
tmp <- tempfile()
m$export(file(tmp))
# Stream it back in R
library(jsonlite)
mydata <- stream_in(file(tmp))
# Or into mongo
m2 <- mongo("diamonds2")
m2$count()
m2$import(file(tmp))
m2$count()
# Remove the collection
m$drop()
m2$drop()
```

R 유용한 패키지

- ❑ sqldf() 패키지 : 데이터를 조회할 때 RDBMS에서 사용하는 SQL 문장을 활용하게 해주는 패키지

```
install.packages("sqldf")
library(sqldf)
library(googleVis)
Fruits
sqldf('select * from Fruits') #Fruits의 모든 데이터 가져오기
sqldf('select * from Fruits where Fruit=₩'Apples₩' ') #Fruit값이 Apples인 값들의 모든 정보 조회하기
sqldf('select * from Fruits limit 3 ') # 출력되는 행 수 제어하기
sqldf('select * from Fruits order by Year ') # 정렬 출력
sqldf('select * from Fruits order by Year desc') # 역 정렬 출력
sqldf('select sum(Sales) from Fruits ') # 판매량 합 구하기
sqldf('select max(Sales) from Fruits ') # 판매량 최대 구하기
sqldf('select min(Sales) from Fruits ') # 판매량 최소 구하기
sqldf('select avg(Sales) from Fruits ') # 판매량평균 구하기
sqldf('select Fruit, sum(Sales) from Fruits group by fruit ') # 그룹핑
sqldf('select Fruit, avg(Sales) from Fruits group by fruit ') # 그룹핑
sqldf('select count(*) from Fruits ') # 그룹핑
sqldf('select * from Fruits
      where Sales > (select sales from Fruits where expenses =78)')
sqldf('select * from Fruits where sales IN ( select Sales From Fruits Where Sales > 95) ')
```


R 유용한 패키지

- ❑ sqldf 패키지 : 데이터를 조회할 때 RDBMS에서 사용하는 SQL 문장을 활용하게 해주는 패키지

```
#조인 사용하기
name<-c('Sandra Bullock', 'Jodie Foster', 'Meg Ryan', 'Demi Moore')
studno<-c(100, 200, 300, 400)
profno<-c(1000, 2000, 3000, 5000)
student<-data.frame(STUDNO=studno, NAME=name, PROFNO=profno)
name<-c('Winona Ryder', 'Michelle Pfeiffer', 'Julia Roberts', 'James Dean')
profno<-c(1000, 2000, 3000, 4000)
professor<-data.frame( PROFNO=profno,NAME=name)
sqldf('select s.name student_name, p.name prof_name from student s, professor p where s.profno = p.profno')
sqldf('select s.name student_name, p.name prof_name from student s join professor p on s.profno = p.profno')
sqldf('select s.name student_name, p.name prof_name from student s left join professor p using (profno)')
sqldf('select s.name student_name, p.name prof_name from student s right join professor p using (profno)')

empno<-c(10, 20, 30, 40, 50)
ename<-c("JAMES", "MILLER", "FORD", "STEVE", "ALLEN")
pempno <- c(NA, 10, 20, 20, 30)
emp<-data.frame(empno=empno, ename=ename, pempno=pempno)
sqldf('select a.ename "사원이름", b.ename "상사이름" from emp a, emp b where a.pempno = b.empno')
sqldf('select a.ename "사원이름", b.ename "상사이름" from emp a join emp b on a.pempno = b.empno')
```

R 유용한 패키지

- ❑ sqldf 패키지 : 데이터를 조회할 때 RDBMS에서 사용하는 SQL 문장을 활용하게 해주는 패키지

```
library(googleVis)
sqldf("update Fruits set Profit=50 where Fruit='Apples' and Year=2008" )
sqldf('select * from Fruits')

sqldf("delete from Fruits where Fruit='Apples' Year=2008" )
sqldf('select * from Fruits')

var1 <- matrix("가", "라", "사"))
var1
ca<-c("가", "나","다", "라" , "마", "바", "사")
lv<-c(3,7,11, 31, 49, 78, 43)
id<-c(3233, 3789, 4939, 2336, 4555, 7888, 9999)
data<-data.frame(CA=ca.LV=lv.ID=id)
data
var2<-as.data.frame(var1)
sqldf("select * from var2")
sqldf("select id from data where ca in (select * from var2)")
```

R 기본 함수

✓ apply 계열 함수

- 벡터, 행렬 또는 데이터 프레임에 임의의 함수를 적용한 결과를 얻기 위한 함수
- 데이터 전체에 함수를 한 번에 적용하는 벡터 연산을 수행하므로 속도가 빠르다.

| 함수 | 설명 | 다른 함수와 비교했을 때의 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() | lapply와 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() | sapply의 확장된 버전으로, 여러 개의 벡터 또는 리스트를 인자로 받아 함수에 각 데이터의 첫째 요소들을 적용한 결과, 둘째 요소들을 적용한 결과, 셋째 요소들을 적용한 결과 등을 반환 | 여러 데이터를 함수의 인자로 적용 |

R 기본 함수

`apply` : 배열 또는 행렬에 함수 FUN을 MARGIN 방향으로 적용하여 결과를 벡터, 배열 또는 리스트로 반환한다.

```
apply(  
  X,          # 배열 또는 행렬  
  MARGIN,     # 함수를 적용하는 방향. 1은 행 방향, 2는 열 방향  
             # c(1, 2)는 행과 열 방향 모두를 의미  
  FUN        # 적용할 함수  
)
```

반환 값은 FUN이 길이 1인 벡터들을 반환한 경우 벡터, 1보다 큰 벡터들을 반환한 경우 행렬, 서로 다른 길이의 벡터를 반환한 경우 리스트다.

```
> d <- matrix(1:9, ncol=3)  
> d  
> apply(d, 1, sum)  
> apply(d, 2, sum)  
  
> head(iris)  
> apply(iris[, 1:4], 2, sum)
```

R 기본 함수

`rowSums` : 숫자 배열 또는 데이터 프레임에서 행의 합을 구한다.

```
rowSums(  
  x,                # 배열 또는 숫자를 저장한 데이터 프레임  
  na.rm=FALSE,     # NA를 제외할지 여부  
)
```

반환 값은 행 방향에 저장된 값의 합이다.

`rowMeans` : 숫자 배열 또는 데이터 프레임에서 행의 평균을 구한다.

```
rowMeans(  
  x,                # 배열 또는 숫자를 저장한 데이터 프레임  
  na.rm=FALSE,     # NA를 제외할지 여부  
)
```

반환 값은 행 방향에 저장된 값의 평균이다.

```
> colSums(iris[, 1:4])
```

R 기본 함수

`lapply` : 벡터, 리스트, 표현식, 데이터 프레임 등에 함수를 적용하고 그 결과를 리스트로 반환한다.

```
lapply(  
  X,      # 벡터, 리스트, 표현식 또는 데이터 프레임  
  FUN,    # 적용할 함수  
  ...     # 추가 인자. 이 인자들은 FUN에 전달된다.  
)
```

반환 값은 X와 같은 길이의 리스트다.

```
> (result <- lapply(1:3, function(x)  
  { x*2 })))  
> result1  
> unlist(result)
```

`unlist` : 리스트 구조를 벡터로 변환한다.

```
unlist(  
  x,                # R 객체. 보통 리스트 또는 벡터  
  recursive=FALSE, # x에 포함된 리스트 역시 재귀적으로 변환할지 여부  
  use.names=TRUE   # 리스트 내 값의 이름을 보존할지 여부  
)
```

반환 값은 벡터다.

R 기본 함수

`do.call` : 함수를 리스트로 주어진 인자에 적용하여 결과를 반환한다.

```
do.call(  
  what, # 호출할 함수  
  args, # 함수에 전달할 인자의 리스트  
)
```

반환 값은 함수 호출 결과다.

```
> (x <- list(a=1:3, b=4:6))  
> lapply(iris[, 1:4], mean)  
#data.frame에도 lapply() 함수 적용  
> colMeans(iris[, 1:4])
```

R 기본 함수

데이터 프레임을 처리한 결과를 리스트로 얻은 뒤 해당 리스트를 다시 데이터 프레임으로 변환할 필요가 있을 수 있다.

1. `unlist()`를 통해 리스트를 벡터로 변환한다.
2. `matrix()`를 사용해 벡터를 행렬로 변환한다.
3. `as.data.frame()`을 사용해 행렬을 데이터 프레임으로 변환한다.
4. `names()`를 사용해 리스트로부터 변수명을 얻어와 데이터 프레임의 각 컬럼에 이름을 부여한다.

```
> d <- as.data.frame(matrix(unlist(lapply(iris[, 1:4], mean))), ncol=4, byrow=TRUE))
> names(d) <- names(iris[, 1:4])
> d
> data.frame(do.call(cbind, lapply(iris[, 1:4], mean)))
> x <- list(data.frame(name="foo", value=1),
+           data.frame(name="bar", value=2))
> unlist(x)
> x <- list(data.frame(name="foo", value=1),
+           data.frame(name="bar", value=2))
> do.call(rbind, x)
```


R 기본 함수

`sapply` : 벡터, 리스트, 표현식, 데이터 프레임 등에 함수를 적용하고 그 결과를 벡터 또는 행렬로 반환한다.

```
sapply(  
  x,      # 벡터, 리스트, 표현식 또는 데이터 프레임  
  FUN,    # 적용할 함수  
  ...,    # 추가 인자. 이 인자들은 FUN에 전달된다.  
)
```

반환 값은 FUN의 결과가 길이 1인 벡터들이면 벡터, 길이가 1보다 큰 벡터들이면 행렬이다.

```
> lapply(iris[, 1:4], mean)  
> sapply(iris[, 1:4], mean)  
> class(sapply(iris[, 1:4], mean)) # "numeric"은 숫자를 저장한 벡터를 의미함  
> x <- sapply(iris[, 1:4], mean)  
> as.data.frame(x)  
> as.data.frame(t(x))  
> sapply(iris, class)  
> y <- sapply(iris[, 1:4], function(x) { x > 3 })  
> class(y)  
> head(y)
```

R 기본 함수

`tapply` : 벡터 등에 저장된 데이터를 주어진 기준에 따라 그룹으로 묶은 뒤 각 그룹에 함수를 적용하고 그 결과를 반환한다.

```
tapply(  
  X,      # 벡터  
  INDEX,  # 데이터를 그룹으로 묶을 색인. 팩터를 지정해야 하며 팩터가 아닌 타입이 지정되면  
          # 팩터로 형 변환된다.  
  FUN,    # 각 그룹마다 적용할 함수  
  ...,    # 추가 인자. 이 인자들은 FUN에 전달된다.  
)
```

반환 값은 배열이다.

```
> tapply(1:10, rep(1, 10), sum)  
> tapply(1:10, 1:10 %% 2 == 1, sum)  
> tapply(iris$Sepal.Length, iris$Species, mean)  
> m <- matrix(1:8,  
              ncol=2,  
              dimnames=list(c("spring", "summer", "fall", "winter"),  
                            c("male", "female")))  
> m  
> tapply(m, list(c(1, 1, 2, 2, 1, 1, 2, 2),  
                  c(1, 1, 1, 1, 2, 2, 2, 2)), sum)
```

R 기본 함수

`mapply` : 함수에 리스트 또는 벡터로 주어진 인자를 적용한 결과를 반환한다.

```
mapply(  
  FUN, # 실행할 함수  
  ..., # 적용할 인자  
)
```

...에 주어진 여러 데이터가 있을 때 FUN에 이들 데이터 각각의 첫째 요소를 인자로 전달하여 실행한 결과, 각각의 둘째 요소를 인자로 전달하여 실행한 결과 등을 반환한다.

| 함수 | 설명 |
|-------------------------------------|--|
| <code>rnorm(n, mean=0, sd=1)</code> | 평균이 n , 표준 편차가 sd 인 정규 분포를 따르는 난수 n 개 발생 |
| <code>runif(n, min=0, max=1)</code> | 최솟값이 min , 최댓값이 max 인 균등 분포를 따르는 난수 n 개 발생 |
| <code>rpois(n, lambda)</code> | 람다 값이 $lambda$ 인 포아송 분포를 따르는 난수 n 개 발생 |
| <code>rexp(n, rate=1)</code> | 람다가 $rate$ 인 지수 분포를 따르는 난수 n 개 발생 |

R 기본 함수

```
> rnorm(10, 0, 1)
> mapply(rnorm,
+       c(1, 2, 3),    # n
+       c(0, 10, 100), # mean
+       c(1, 1, 1))    # sd
> mapply(mean, iris[, 1:4])
```

R 기본 함수

- ✓ 데이터를 그룹으로 묶은 후 함수 호출

| 함수 | 특징 |
|--------------------|---|
| doBy::summaryBy() | 데이터 프레임을 컬럼 값에 따라 그룹으로 묶은 후 요약 값 계산 |
| doBy::orderBy() | 지정된 컬럼 값에 따라 데이터 프레임을 정렬 |
| doBy::sampleBy() | 데이터 프레임을 특정 컬럼 값에 따라 그룹으로 묶은 후 각 그룹에서 샘플(sample) 추출 |

`base::summary` : 다양한 모델링 함수의 결과에 대한 요약 결과를 반환한다.

```
summary(  
  object # 요약할 객체  
)
```

반환 값은 요약 결과며, 데이터 타입은 object 타입에 따라 다르다.

R 기본 함수

- ✓ 데이터를 그룹으로 묶은 후 함수 호출

`doBy::summaryBy` : 포물러에 따라 데이터를 그룹으로 묶고 요약한 결과를 반환한다.

```
doBy::summaryBy(  
  formula,          # 요약을 수행할 포물러  
  data=parent.frame() # 포물러를 적용할 데이터  
)
```

반환 값은 데이터 프레임이다.

```
> install.packages("doBy")  
> library(doBy)  
> summary(iris)  
> quantile(iris$Sepal.Length)  
> quantile(iris$Sepal.Length, seq(0, 1, by=0.1))  
> summaryBy(Sepal.Width + Sepal.Length ~ Species, iris)
```

R 기본 함수

✓ 포물러

- $Y_1 + Y_2 + \dots + Y_n \sim X_1 + X_2 + \dots + X_m$
- 포물러의 정확한 의미는 사용하는 함수에 따라 다르나 일반적으로 "(Y₁, Y₂, ..., Y_n)의 순서쌍을 (X₁, X₂, ..., X_m)의 순서쌍으로 모델링한다"

| 연산자 | 예 | 의미 |
|-----|------------------|---|
| + | $Y \sim X1 + X2$ | Y를 X1, X2로 모델링. 상수항은 암시적으로 허용된다. 따라서 선형 회귀에 이 포물러를 사용하면 $Y = a * X1 + b * X2 + c$ 를 의미한다. |
| - | $Y \sim X1 - X2$ | Y를 X1로 모델링하되 X2는 제외한다. 특히 선형 회귀에서 $Y \sim X1 + X2 - 1$ 은 Y를 X1과 X2로 모델링하되 상수항은 제외한다는 의미다. 즉, $Y = a * X1 + b * X2$ 를 의미한다. |
| | $Y \sim X1 X2$ | X2의 값에 따라 데이터를 그룹으로 묶은 후 각 그룹별로 $Y \sim X1$ 을 적용한다. |
| : | $Y \sim X1:X2$ | Y를 X1과 X2의 상호 작용(interaction)에 따라 모델링한다. 상호 작용은 $Y = a * X1 * X2 + b$ 와 같이 X1과 X2가 동시에 Y 값에 영향을 주는 상황을 말한다. 특히 영향을 주는 방식이 $Y = a * X1 + b * X2 + c$ 와 같은 합의 형태와는 구분된다 |
| * | $Y \sim X1 * X2$ | $Y \sim X1 + X2 + X1:X2$ 의 축약형 표현이다 |

R 기본 함수

`base::order` : 데이터를 정렬하기 위한 순서를 반환한다.

```
order(  
  ..., # 정렬할 데이터  
  # na.last는 NA 값을 정렬한 결과의 어디에 둘 것인지를 제어한다. 기본값인 na.last=TRUE는  
  # NA 값을 정렬한 결과의 마지막에 둔다. na.last=FALSE는 정렬한 값의 처음에 둔다.  
  # na.last=NA는 NA 값을 정렬 결과에서 제외한다.  
  na.last=TRUE,  
  decreasing=FALSE # 내림차순 여부  
)
```

반환 값은 원 데이터에 지정하면 정렬된 결과가 나오도록 하는 색인이다.

```
> order(iris$Sepal.Width)  
> iris[order(iris$Sepal.Width),]  
> iris[order(iris$Sepal.Length, iris$Sepal.Width), ]
```


R 기본 함수

`doBy::orderBy` : 포물러에 따라 데이터를 정렬한다.

```
doBy::orderBy(  
  formula, # 정렬할 기준을 지정한 포물러  
            # ~의 좌측은 무시하며, ~ 우측에 나열한 이름에 따라 데이터가 정렬된다.  
  data,    # 정렬할 데이터  
)
```

반환 값은 `order()`와 동일하다.

```
> orderBy(~ Sepal.Width, iris)  
> orderBy(~ Species + Sepal.Width, iris)
```

R 기본 함수

base::sample : 샘플링을 수행한다.

```
sample(  
  x,          # 샘플을 뽑을 데이터 벡터. 만약 길이 1인 숫자 n이 지정되면 1:n에서 샘플이 선택된  
  da.  
  size,       # 샘플의 크기  
  replace=FALSE, # 복원 추출 여부  
  # 데이터가 뽑힐 가중치. 예를 들어, x=c(1, 2, 3)에서 2개의 샘플을 뽑되 각 샘플이 뽑힐 확률을  
  # 50%, 20%, 30%로 하고자 한다면 size=2, prob=c(5, 2, 3)을 지정한다. 이 인자의 이름은  
  # prob이지만 prob에 지정한 값의 합이 1일 필요는 없다.  
  prob  
)
```

반환 값은 샘플을 저장한 길이 size인 벡터다.

```
> sample(1:10, 5)  
> sample(1:10, 5, replace=TRUE)  
> sample(1:10, 10)  
> iris[sample(NROW(iris), NROW(iris)),]
```

R 기본 함수

doBy::sampleBy : 포물러에 따라 데이터를 그룹으로 묶은 후 샘플을 추출한다.

```
doBy::sampleBy(  
  formula,          # ~ 우측에 나열한 이름에 따라 데이터가 그룹으로 묶인다.  
  frac=0.1,         # 추출할 샘플의 비율이며 기본값은 10%  
  replace=FALSE,    # 복원 추출 여부  
  data=parent.frame(), # 데이터를 추출할 데이터 프레임  
  systematic=FALSE   # 계통 추출(Systematic Sampling)7을 사용할지 여부  
)
```

반환 값은 데이터 프레임이다.

```
> sampleBy(~ Species, frac=0.1, data=iris)
```

R 기본 함수

```
> split(iris, iris$Species)
> lapply(split(iris$Sepal.Length, iris$Species), mean)
> subset(iris, Species == "setosa")
> subset(iris, Species == "setosa" & Sepal.Length > 5.0)
> subset(iris, select=c(Sepal.Length, Species))
> subset(iris, select=-c(Sepal.Length, Species))
> iris[, !names(iris) %in% c("Sepal.Length", "Species")]
> x <- data.frame(name=c("a", "b", "c"), math=c(1, 2, 3))
> y <- data.frame(name=c("c", "b", "a"), english=c(4, 5, 6))
> merge(x, y)
> x <- data.frame(name=c("a", "b", "c"), math=c(1, 2, 3))
> y <- data.frame(name=c("c", "b", "a"), english=c(4, 5, 6))
> cbind(x, y)
> x <- data.frame(name=c("a", "b", "c"),
+               math=c(1, 2, 3))
> y <- data.frame(name=c("a", "b", "d"),
+               english=c(4, 5, 6))
> merge(x, y)
> merge(x, y, all=TRUE)
```

```
> x <- c(20, 11, 33, 50, 47)
> sort(x)
> sort(x, decreasing=TRUE)
> x
> x <- c(20, 11, 33, 50, 47)
> order(x)
> x <- c(20, 11, 33, 50, 47)
> order(x, decreasing=TRUE)
> iris[order(iris$Sepal.Length), ]
> iris[order(iris$Sepal.Length,
iris$Petal.Length), ]
```

R 기본 함수

✓ 조건에 맞는 데이터의 색인 찾기

`which` : 조건이 참인 색인을 반환한다.

```
which(  
  x # 논릿값 벡터 또는 배열  
)
```

반환 값은 논릿값이 참인 색인이다.

`which.max` : 최댓값의 위치를 반환한다.

```
which.max(  
  x # 숫자 벡터  
)
```

반환 값은 최댓값이 저장된 색인이다.

`which.min` : 최솟값의 위치를 반환한다.

```
which.min(  
  x # 숫자 벡터  
)
```

반환 값은 최솟값이 저장된 색인이다.

```
> subset(iris, Species == "setosa")  
> iris[iris$Species == "setosa", ]  
> which(iris$Species == "setosa")  
> which.min(iris$Sepal.Length)  
> which.max(iris$Sepal.Length)
```

R 기본 함수

- ✓ aggregate()-데이터프레임에 대해 특정 값을 기준으로 그룹핑한 후 합계, 함수 적용하여 결과 생성

```
#aggregate(계산될컬럼~기준될컬럼, 데이터, 함수)
install.packages("googleVis")
library(googleVis)
Fruits
aggregate(Sales~Year, Fruits, sum) #년도 별 Sales된 수량 sum
aggregate(Sales~Fruit, Fruits, sum)
aggregate(Sales~Fruit, Fruits, max)
aggregate(Sales~Fruit+Year, Fruits, sum) #' + 추가조건'형태로 추가
```

- ✓ apply() - Matrix에 대해 apply(대상, 행/열, 적용함수)

```
mat1<-matrix(c(1,2,3, 4,5,6), nrow=2, byrow=T)
mat1
apply(mat1, 1, sum) #행의 합계
apply(mat1, 2, sum) #열의 합계
apply(mat1[,c(2,3)], 2, max) #2열, 3열 최대값 구하기
```

R 기본 함수

- ✓ 데이터 분석에서 "Split-Apply-Combine"이라는 전략을 구현한 것
- ✓ 데이터셋을 나누어서 각 조각들을 만들고, 각각의 조각에 특정 함수를 적용하고, 결과를 합쳐서 제공한다
- ✓ lapply - 입력으로 리스트를 받아서 결과로 리스트를 제공

```
list1 <- list(Fruits$Sales)
list1
list2 <- list(Fruits$Profit)
lapply(c(list1, list2), max)
sapply(c(list1, list2), max)
lapply(Fruits[, c(4, 5)], max)
sapply(Fruits[, c(4, 5)], max)
```

- ✓ sapply - simplify, 결과를 벡터 형태로 단순화해서 리턴

```
Fruits
tapply(Sales, Fruits, sum)
attach(Fruits) #데이터프레임에서 각 컬럼 이름을 변수명처럼 처리해서 데이터를 쉽게
관리하게 하는 기능
tapply(Sales, Fruits, sum) #과일 이름별로 Sales 합계 구하기
tapply(Sales, Year, sum) #년도별 Sales 합계 구하기
```

R 기본 함수

- ✓ sweep - 벡터, matrix, array, dataframe으로 구성된 데이터들에 동일한 기준을 적용시켜 차이 나는 부분을 요약 보여주는 함수

```
mat1  
a<-c(1,1,1)  
sweep(mat1, 2, a) #mat1의 각 행을 a의 요소값을 뺀 나머지를 출력
```


R 기본 함수

- ❑ aggregate() - 데이터 프레임 상대로 주어진 함수 값 구하기
- ❑ apply() - lapply(), sapply(), tapply(), mapply()
- ❑ cor() - 상관함수
- ❑ cumsum() - 설정된 지점까지의 누적합
- ❑ cumprom() - 설정된 지점까지의 누적곱
- ❑ diff() - 차이 나는 부부를 찾아냄
- ❑ length() - 요소갯수를 구해서 출력함
- ❑ max() - 최대값을 출력함
- ❑ min() - 최소값을 출력함
- ❑ mean() - 평균값을 출력함
- ❑ median() - 가운데값을 출력함
- ❑ order() - 각 요소의 원래 위치
- ❑ prod() - 누적곱을 출력함
- ❑ range() - 범위값을 출력함
- ❑ rank() - 각 요소의 순위를 출력함

```
vec1<-c(1,2,3,4,5)
vec2<-c('a','b','c','d','e')
max(vec1)
max(vec2)
mean(vec1)
mean(vec2)
min(vec1)
sd(vec1)
sum(vec1)
var(vec1)
sort(Fruits$Sales)
sort(Fruits$Sales, decreasing=T)
```

R 기본 함수

- ❑ `rev()` - 요소의 역순을 출력함
- ❑ `sd()` - 표준편차를 출력함
- ❑ `sort()` - 정렬결과를 출력함
- ❑ `sum()` - 총합을 출력함
- ❑ `summary()` - 요약통계량을 출력함
- ❑ `tapply()` - 벡터에서 주어진 함수연산을 수행함
- ❑ `var()` - 분산값을 출력함

R 수학 관련 함수

- ☐ `sin(x)`
- ☐ `cos(x)`
- ☐ `tan(x)`
- ☐ `sinh(x)`
- ☐ `cosh(x)`
- ☐ `tanh(x)`
- ☐ `asin(x)`
- ☐ `acos(x)`
- ☐ `atan(x)`
- ☐ `asinh(x)`
- ☐ `acosh(x)`
- ☐ `atanh(x)`
- ☐ `log(x)`
- ☐ `log10(x)`
- ☐ `log2(x)`

R 정규표현식과 함수

□ 정규표현식 - 많은 문자들 중에서 원하는 문자들만 효율적으로 찾아주는 표현식

\\Wd Digit, 0, 1, 2,...9
\\WD 숫자가 아닌것
\\Ws 공백
\\WS 공백이 아닌것
\\Ww 단어
\\WW 단어가 아닌것
\\Wt Tab
\\Wn New line(줄바꿈)
^ 시작되는 글자
\$ 마지막 글자
\\ Escape character (탈출문자),
| 두개 이상의 조건을 동시에 지정
. 엔터를 제외한 모든 문자
[ab] a 또는 b
[^ab] a와 b를 제외한 모든 문자
[0-9] 모든 숫자
[A-Z] 영어와 대문자
[a-z] 영어와 소문자
[A-z] 모든 영문자(대소문자전부)

i+ i가 최소 1회 나오는 경우
i* i가 최소 0회 나오는 경우
i? i가 최소 0회에서 최대 1회만 나오는 경우
i{n} i가 연속적으로 n회 나오는 경우
i[n1, n2] i가 n1회에서 n2회 나오는 경우
i{n, } i가 n회 이상 나오는 경우
[:alnum:] 문자와 숫자가 나오는 경우
[:alpha:] 문자가 나오는 경우
[:blank:] 공백이 있는 경우
[:cntrl:] 제어 문자가 있는 경우
[:digit:] Digit, 0, 1, 2,...9
[:graph:] Graphical characters
[:lower:] 소문자가 있는 경우
[:print:] 숫자, 문자, 특수문자, 공백문자
[:punct:] 특수 문자
[:space:] 공백문자
[:upper:] 대문자가 있는 경우
[:xdigit:] 16진수가 있는 경우

R 기본 함수

- ✓ 특정 패턴만 찾아내서 위치 출력

```
char <- c('apple', 'Apple', 'orange', 'APPLE', 'banana', 'grape')
grep('apple', char)
grep('orange', char)
grep('pp', char)
grep('pp', char, value=T)
grep('^A', char, value=T)
grep('e$', char, value=T)

char2 <- c('apple', 'Apple2', 'orange', 'APPLE3', 'banana', 'grape')
grep('[1-9]', char2, value=T)
grep('[:upper:]', char2, value=T)

#입력된 배열이나 문자열의 길이 리턴
nchar(char)
nchar('Republic of Korea')
nchar('대한민국')
```

R 기본 함수

□ 문자열 연결, 부분 추출 함수

#문자열 연결

```
paste('apple', 'orange', 'banana', 'grape')  
paste('apple', 'orange', 'banana', 'grape', sep=") #공백을 지움  
paste('apple', 'orange', 'banana', 'grape', sep='/')  
paste('I', 'W'm', 'Hungry') #특수문자가 있을 경우 escape character를 사용
```

#문자열 부분추출 substr(문자열, 시작위치, 끝위치)
substr('I W'm Hungry', 5,8)

#특정 글자를 기준으로 분리 strsplit(문자열, split="기준문자");
strsplit('2017/07/06', split='/')

#특정 패턴을 찾기 regexpr(패턴, 텍스트)
grep('-', '010-4018-4589') #위치를 못찾음
regexpr('-', '010-4018-4589')

R 시각화 차트 관련 함수

- ❑ `plot()`
- ❑ `barplot()` – 막대 그래프
- ❑ `hist()` – 히스토그램 그래프
- ❑ `pie()` – 파이 모양의 차트
- ❑ `pie3D()` – 3D Pie 차트
- ❑ `boxplot()` – 상자 차트
- ❑ `igraph()` – 관계도 그리기
- ❑ `treemap()` – 나무맵
- ❑ `radarchart()` – 레이더차트

R – Chart 시각화

✓ 분포도, 꺾은선 그래프

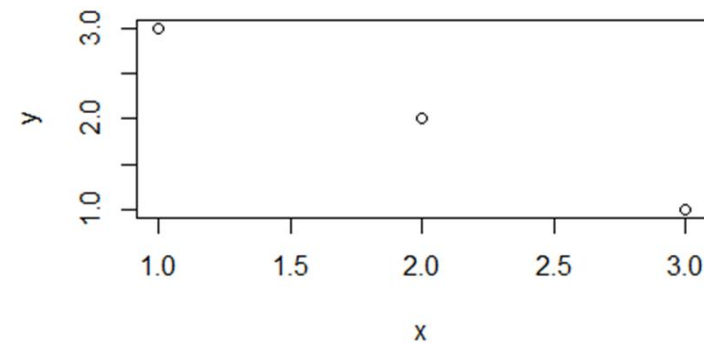
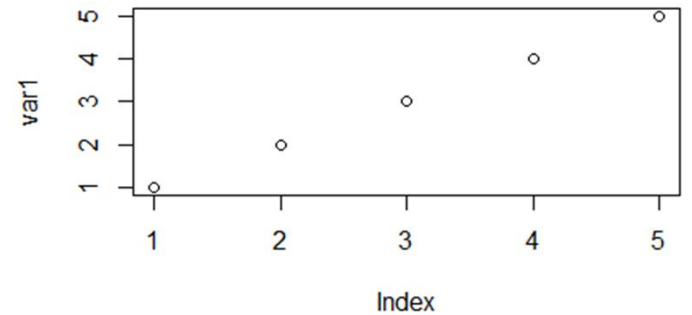
- 데카르트 방식 : `plot(x, y)`
- 수식기반 방식 : `plot(y~x)`

`plot(y축 데이터, 옵션)`
`plot(x축 데이터, y축 데이터, 옵션)`

```
#기본값으로 그래프 그리기  
var1<-c(1,2,3,4,5)  
plot(var1) #y축으로 사용됨
```

```
var2<-c(2,2,2)  
plot(var2)
```

```
x<-1:3  
y<-3:1  
plot(x, y)
```



R – Chart 시각화

✓ plot함수의 옵션

| 옵션 | 설명 |
|-----------------|--------------------------|
| main | 제목 설정 |
| sub | 서브 제목 |
| ann=F | x, y 축 제목을 지정하지 않습니다. |
| xlab="" ylab="" | x, y축에 사용할 문자열 지정 |
| tmag=2 | 제목 등에 사용되는 문자의 확대율 지정 |
| axes=F | x, y 축을 표시하지 않습니다. |
| axis | x,y 축을 사용자의 지정값으로 표시합니다. |

R – Chart 시각화

✓ plot함수 그래프 타입 옵션

| 그래프 타입 | 설명 |
|----------|---------------------------|
| type="p" | 점 모양 그래프(기본값) |
| type="l" | 선 모양 그래프(꺼은선 그래프) |
| type="b" | 점과 선 모양 그래프 |
| type="c" | "b"에서 점을 생략한 모양 |
| type="o" | 점과 선을 중첩해서 그린 그래프 |
| type="h" | 각 점에서 x축까지의 수직선 그래프 |
| type="s" | 왼쪽값을 기초로 계단모양으로 연결한 그래프 |
| type="S" | 오른쪽 값을 기초로 계단모양으로 연결한 그래프 |
| type="n" | 축만 그리고 그래프는 그리지 않습니다. |

R - Chart

- ✓ x, y축의 최대 한계값 수동 지정 - xlim, ylim

```
# x, y축의 최대 한계값 지정하기
plot(x, y, xlim=c(1,10), ylim=c(1, 5))
# 그래프 제목 지정해서 출력하기
plot(x, y, xlim=c(1,10), ylim=c(1, 5) , xlab="X축 값", ylab="Y축 값", main="Plot Test")

# 현재 창의 내용을 모두 지우고 새로 그리고 싶으면 plot.new()
# 새로운 창에서 그래프를 다시 그리고 싶으면 dev.new()

v1<-c(100, 130, 120, 160, 150)
plot(v1, type='o', col='red', ylim=c(0, 200), axes=FALSE, ann=FALSE)

axis(1, at=1:5, lab=c("MON", "TUE", "WED", "THU", "FRI"))
axis(2, ylim=c(0, 200))

title(main="FRUIT", col.main="red", font.main=4)
title(xlab="DAY", col.lab="black")
title(ylab="PRICE", col.lab="blue")
```

R – Chart 시각화

✓ plot함수 그래프 타입 옵션

| 선 모양 선택 | 설명 |
|-----------------------|---------|
| lty=0, lty="blank" | 투명선 |
| lty=1, lty="solid" | 실선 |
| lty=2, lty="dashed" | 대쉬선 |
| lty=3, lty="dotted" | 점선 |
| lty=4, lty="dotdash" | 점선과 대쉬선 |
| lty=5, lty="longdash" | 긴 대쉬선 |
| lty=6, lty="twodash" | 2개의 대쉬선 |

R – Chart 시각화

✓ Plot함수 색, 기호 옵션

| 색, 기호 | 설명 |
|-------------------|--|
| col=1, col="blue" | 기호의 색지정 1-검정, 2-빨강, 3-초록, 4-파랑, 5-연파랑, 6-보라, 7-노랑, 80-회색 |
| pch=0, pch="문자" | 점의 모양을 지정 |
| bg="blue" | 그래프의 배경색 지정 |
| lwd="숫자" | |
| cex="숫자" | |
| var() | 점이나 문자를 그릴 때 점이나 문자의 굵기를 지정 |

R - Chart

- ✓ 그래프의 배치 조정(mfrow)
- ✓ 여러 개의 그래프를 중첩으로 그리기
- ✓ 범례 추가하기

```
par (mfrwo=c(행의 개수, 열의 개수))  
par (new-T)  
legend(x 축 위치 , y 축 위치, cex=글자크기, col=색상, pch=크기, lty=선모양)
```

```
# 그래프의 배치 조정 : mfrow  
par(mfrow=c(1,3))  
plot(v1, type="o")  
plot(v1, type="s")  
plot(v1, type="l")  
  
par(mfrow=c(1,3))  
pie(v1)  
plot(v1, type="o")  
barplot(v1)
```

R - Chart

```
#여러 개의 그래프를 중첩으로 그리기 par(new=T)
#그래프가 새로 그려질 때마다 y축 제목과 ylim 값이 새롭게 적용됩니다.
par(mfrow=c(1,1))
v1<-c(1,2,3,4,5)
v2<-c(5,4,3,2,1)
v3<-c(3,4,5,6,7)
plot(v1, type="s", col="red", ylim=c(1,5))
par(new=T)
plot(v2, type="o", col="blue", ylim=c(1,5))
par(new=T)
plot(v3, type="l", col="green" )

#lines() 함수
plot(v1, type="s", col="red", ylim=c(1,10))
lines(v2, type="o", col="blue", ylim=c(1,5))
lines(v3, type="l", col="green" , ylim=c(1, 15))

#범례 추가하기
plot(v1, type="s", col="red", ylim=c(1,10))
lines(v2, type="o", col="blue", ylim=c(1,5))
lines(v3, type="l", col="green" , ylim=c(1, 15))
legend(4, 9, c("v1", "v2", "v3"), cex=0.9, col=c("red", "blue", "green"), lty=1)
```

R – Chart 시각화

✓ barplot함수

| 옵션 | 설명 |
|---------------------|----------------------------------|
| angle, density, col | 막대를 칠하는 선분의 각도, 선분의 수, 선분의 색을 지정 |
| legend | 오른쪽 상단에 범례를 표현 |
| name | 각 막대의 라벨을 정하는 문자열 벡터를 지정 |
| width | 막대의 상대적인 폭을 벡터로 지정 |
| space | 각 막대 사이의 간격을 지정 |
| beside | TRUE를 지정하면 각각의 값마다 막대를 그립니다. |
| horiz | TRUE를 지정하면 막대를 옆으로 눕혀서 그립니다. |

R - Chart

✓ barplot함수

```
#bar그래프 그리기
x<-c(1,2,3,4,5)
barplot(x)

#가로로 출력
x<-c(1,2,3,4,5)
barplot(x, horiz=T)
#그룹으로 묶어서 출력하고 색깔지정하기
x<-matrix(c(5,4,3,2), 2, 2)
barplot(x, beside=T, names=c(5,3), col=c("green", "yellow"))
#하나의 막대 그래프로 출력하기
barplot(x, names=c(5,3), col=c("green", "yellow"), ylim=c(0, 12))

#그룹으로 묶어서 가로로 출력하기
par(matrix=c(1,0,5,1,0,5))
barplot(x, names=c(5,3), beside=T,col=c("green", "yellow"), horiz=T)

#그룹으로 묶어서 가로로 하나의 막대 그래프로 출력하기
barplot(x, names=c(5,3), col=c("green", "yellow"), horiz=T, ylim=c(0, 12))
```

R - Chart

✓ barplot함수

```
#그룹으로 묶어서 가로로 하나의 막대 그래프로 출력하기
barplot(x, names=c(5,3), col=c("green", "yellow"), horiz=T, ylim=c(0, 12))
#여러 막대 그래프를 그룹으로 묶어서 출력하기
v1<-c(100, 120, 140, 160, 180)
v2<-c(120, 130, 150, 140, 170)
v3<-c(140, 170, 120, 110, 160)

qty<-data.frame(BANANA=v1, CHERRY=v2, ORANGE=v3)
qty
barplot(as.matrix(qty), main="Fruits's Sales QTY", beside=T,
col=rainbow(nrow(qty)), ylim=c(0, 400))
legend(14, 400, c("MON", "TUE", "WED", "THU", "FRI"), cex=0.8,
fill=rainbow(nrow(qty)))

#하나의 막대 그래프에 여러 가지 내용을 하나로 출력하기
barplot(t(qty),ylim=c(0, 900), main="Fruits's Sales QTY",
col=rainbow(length(qty)), space=0.1, cex.axis=0.8, las=1, names.arg=c("MON",
"TUE", "WED", "THU", "FRI"), cex=0.8)
legend(0.2, 800, names(qty), cex=0.7, fill=rainbow(length(qty)))
```

R - Chart

✓ barplot함수

```
#조건을 주고 그래프 그리기
peach<-c(180, 200, 250, 198, 170)
colors<-c()
for(i in 1:length(peach)) {
  if(peach[i]>=200) {
    colors<-c(colors, "red")
  }else if(peach[i]>=180) {
    colors <- c(colors, "yellow")
  }else {
    colors <- c(colors, "green")
  }
}
barplot(peach, main="Peach Sales QTY", names.arg=c("MON", "TUE", "WED", "THU", "FRI"), col=colors)
```

R – Chart 시각화

✓ pie함수

| 옵션 | 설명 |
|---------------------|------------------------------------|
| angle, density, col | Pie부분을 구성하는 각도, 수, 색상을 지정 |
| legend | 각 pie 부분의 이름을 지정하는 문자벡터를 지정 |
| radius | 원형의 크기를 지정 |
| clockwise | 시계방향(T)으로 회전할지 반 시계방향(F)으로 회전할지 지정 |
| init, angle | 시작되는 지점의 각도 지정 |

```
#히스토그램 그래프 그리기
height <- c(182, 175, 167, 172, 163, 178, 181, 166, 159, 155)
hist(height, main="histogram of height")

par(mfrow=c(1,2), oma=c(2,2,0.1,0.1))
h<-c(1,1,2,3,3,3)
hist(h)
plot(h, main="Plot")
```

R - Chart

✓ pie 함수

```
#pie 차트 그리기
par(mfrow=c(1,1), oma=c(0.5,0.5,0.1,0.1))
p1<-c(10,20,30,40)
pie(p1, radius=1)
#시작 각도를 90도로 지정하기
pie(p1, radius=1, init.angle=90)
#색깔과 label명을 지정하기
pie(p1, radius=1, init.angle=90, col=rainbow(length(p1)), label=c("Week 1", "Week 2", "Week 3", "Week4"))
#수치 값을 함께 출력하기
pct <- round(p1/sum(p1) * 100, 1)
lab <-paste(pct, "%")
pie(p1, radius=1, init.angle=90, col=rainbow(length(p1)), label=lab)
legend(1,1,1, c("Week 1", "Week 2", "Week 3", "Week4"), cex=0.5, fill=rainbow(length(p1)))

pct <- round(p1/sum(p1) * 100, 1)
lab1<- c("Week 1", "Week 2", "Week 3", "Week4")
lab2<- paste(lab1, "\n", pct, "%")
pie(p1, radius=1, init.angle=90, col=rainbow(length(p1)), label=lab2)
```

R - Chart

✓ pie3 함수

```
#pie3D()
install.packages("plotrix")
library(plotrix)
p1<-c(10, 20, 30, 40, 50)
f_pay<-round(p1/sum(p1) * 100, 1)
f_label<-paste(f_pay, "%")
pie3D(p1, main="3D Pie Chart", col=rainbow(length(p1)), cex=0.5, labels=f_label, explode=0.05)
legend(0.5, 1, c("Week 1", "Week 2", "Week 3", "Week4"), cex=0.6, fill=rainbow(length(p1)))
```

R – Chart 시각화

✓ boxplot 함수

| 옵션 | 설명 |
|------------|--------------------------------|
| col | 박스 내부의 색깔을 지정 |
| name | 각 막대의 라벨을 지정할 문자 벡터를 지정 |
| range | 박스의 끝에서 수염까지의 길이를 지정, 기본값은 1.5 |
| notch | True로 저장할 경우 상자의 허리 부분을 가늘게 표시 |
| horizontal | True로 지정하면 상자를 수평으로 그립니다. |
| width | 박스의 폭을 지정 |

```
#상자 차트 boxplot() : 최대값, 최소값, 평균값등을 비교할 때 유용
v1<-c(10, 12, 15, 11, 20)
v2<-c(5, 7, 15, 8, 9)
v3 <- c(11, 20, 15, 18, 13)
boxplot(v1, v2, v3)
boxplot(v1, v2, v3, col=c("blue", "yellow", "pink"), names=c("Blue", "Yellow", "Pink"),
horizontal=T)
```

R – Chart 시각화

✓ 관계도 igraph 함수

| 옵션 | 설명 |
|-------------------|-----------------------------------|
| edge.color | 선의 색상 |
| edge.width | 선의 폭 |
| edge.arrow.size | 화살의 크기 |
| edge.arrow.width | 화살의 폭 |
| edge.arrow.mode | 화살의 머리 유형 |
| edge.lty | 선 유형 solid, dashed, dotted |
| edge.label.family | 레이블 형 1:일반, 2:볼드, 3:이탤릭, 4.볼드 이탤릭 |
| edge.label.font | 레이블 글꼴 serif, sans, mono |
| edge.label.color | 선 레이블 색상 |

R – Chart 시각화

✓ 관계도 igraph 함수

| 옵션 | 설명 |
|---------------------|-----------|
| vertex.size | 점의 크기 지정 |
| vertex.frame.color | 점 윤곽의 색 |
| vertex.label | 점 레이블 |
| vertex.label.font | 폰트 |
| vertex.label.dist | 점 중심과의 거리 |
| vertex.label.color | 점 레이블 색상 |
| vertex.color | 점의 색 지정 |
| vertex.shape | 점의 형태 |
| vertex.label.family | 점 레이블 종류 |
| vertex.label.cex | 점 레이블 크기 |
| vertex.label.degree | 점 레이블 방향 |

R - Chart

✓ 관계도 igraph 함수

```
#관계도 igraph()
install.packages("igraph")
library(igraph)
g1<-graph(c(1,2,2,3,2,4,1,4,5,5,3,6))
plot(g1)

str(g1)
savePlot("./netwok_1.png", type="png")

name <- c('세종대왕', '일지매 부장', '김유신 과장','손흥민 대리', '노정호 대리', '이순신 부장', '유관순 과장', '신사임
당 대리', '강감찬 부장', '광개토 과장', '정몽주 대리')
pemp <- c('세종대왕', '세종대왕','일지매 부장', '김유신 과장', '김유신 과장', '세종대왕','이순신 부장', '유관순 과장',
'세종대왕', '강감찬 부장', '광개토 과장' )
emp <-data.frame(이름=name, 상사이름= pemp)
emp

g<-graph.data.frame(emp, directed=T)
plot(g, layout=layout.fruchterman.reingold, vertex.size=8, edge.arrow.size=0.5)
str(g)
```

R - Chart

✓ Treemap

- 어떤 기준 값을 지정해서 그 값을 가진 데이터들끼리 모아서 면적으로 보여주는 역할

```
#treemap 활용
install.packages("treemap")
library(treemap)
total<-read.csv("./lab01/data/학생시험결과_전체점수.csv", header=T, sep=",")
total
#같은 점수를 가진 이름들을 기준으로 treemap생성
treemap(total, vSize="점수", index=c("점수", "이름"))
#같은 조끼리 그룹핑해서 이름 출력
treemap(total, vSize="점수", index=c("조", "이름"))

treemap(total, vSize="점수", index=c("점수", "조","이름"))
```

R - Chart

✓ Stars()

- 나이팅게일 차트

```
#내장함수 stars()를 사용 비교 분석
total<-read.table("./lab01/data/학생별전체성적_new.txt", header=T, sep=",")
total
row.names(total)<-total$이름
stars(total, flip.labels=FALSE, draw.segment=FALSE, frame.plot=TRUE, full=TRUE, main="학생별 과목별 성적분석-
Star Chart")
lab<-names(total)
value<-table(lab)
value
pie(value, labels=lab, radius=0.1, cex=0.6, col=NA)
stars(total, flip.labels=FALSE, draw.segment=TRUE, frame.plot=TRUE, full=TRUE, main="학생별 과목별 성적분석-
Star Chart")
stars(total, flip.labels=FALSE, draw.segment=TRUE, frame.plot=TRUE, full=FALSE, main="학생별 과목별 성적분석-
Star Chart")
```

R – Chart 시각화

✓ 그래프에 추가적인 선이나 설명 넣기

| 옵션 | 설명 |
|--------|--------------------------------|
| 점 | points() |
| 직선 | lines() , segments(), abline() |
| 격자 | grid() |
| 화살표 | arrows() |
| 직사각형 | rect() |
| 문자 | text(), mtext(), title() |
| 테두리와 축 | box(), axis() |
| 범례 | legend() |
| 다각형 | polygon() |

R - Chart

- ✓ `radarchart()`
 - spider 차트, 레이더 차트

R 함수

✓ ggplot2

- qplot과 그래프 작성을 문법체계하에 layer의 개념으로 속성을 추가하여 모든 편집을 다할 수 있다.

| qplot 옵션 | 설명 |
|------------|---------------------------------------|
| X, y | x축과 y축에 해당하는 변수를 저장함 |
| data | dataframe을 지정하여 축과 색, 크기 등을 지정할 수 있음 |
| color | 변수의 레벨을 이용하여 색상을 지정 |
| shape | 변수의 값을 이용하여 심볼 모양을 사각형, 원 등 다양하게 차별화 |
| size | 심볼의 크기를 변수를 이용하여 지정 |
| alpha | 투명도를 표시하는 값으로 0에서 1의 값으로 1이 가장 진한 상태임 |
| geom | 그래프 유형을 선정, 라인 바 등 |
| facet | 조건에 따른 격자형으로 그래프 작성 |
| xlim, ylim | 각각 2개의 값으로 최소, 최대값을 각 축에 설정 |
| xlab, ylab | 각각 축에 대한 라벨을 설정하는데 변수내용을 추가할 수 있음 |
| main, sub | 메인과 서브 타이틀로 그래프에 대한 설명을 추가함 |

R - Chart

✓ ggplot2

- plot()함수의 확장 버전, 다양한 형태로 데이터를 표현

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

#ggplot(dataframe, aes(x=x축 데이터, y=y축 데이터)) + geom 함수...
#aesthetic mapping 미적 매핑 정도 번역
#데이터를 렌더링 geom_point() :

korean <- read.table("./lab01/data/학생별국어성적_new.txt", header=T, sep=",")
korean
ggplot(korean, aes(x=이름, y=점수)) + geom_point()

# stat : 주어진 데이터에서 geom에 필요한 데이터를 생성합니다.
# stat_bin : 데이터를 갖는 data frame을 출력합니다.
# count : 각 항목의 빈도 수
# density : 각 항목의 밀도 수
# ncount : count와 같으나 값의 범위가 (0, 1)로 스케일링
# ndensity : density와 같으나 값의 범위가 (0,1)로 스케일링
```


R - Chart

✓ ggplot2

- plot()함수의 확장 버전, 다양한 형태로 데이터를 표현

```
# bar plot 표현하기
ggplot(korean, aes(x=이름, y=점수)) + geom_bar(stat="identity")
#색상과 테두리 넣기
ggplot(korean, aes(x=이름, y=점수)) + geom_bar(stat="identity", fill="green", colour="red")
#각도 조절, 색상 지정
gg<-ggplot(korean, aes(x=이름, y=점수)) + geom_bar(stat="identity", fill="green", colour="red")
gg+theme(axis.text.x=element_text(angle=45, hjust=1, vjust=1, color="blue", size=8))

# 바 그래프에 여러 과목이 동시에 출력하기
score_kem <- read.csv("./lab01/data/학생별과목별성적_국영수_new.csv", header=T)
score_kem
library(plyr)
sort_kem <- arrange(score_kem, 이름, 과목)
sort_kem
sort_kem2 <- ddply(score_kem, "이름", transform, 누적합계=cumsum(점수))
sort_kem2
sort_kem3 <- ddply(sort_kem2, "이름", transform, 누적합계=cumsum(점수), label=cumsum(점수)-0.5*점수)
sort_kem3
```

R - Chart

✓ ggplot2

- plot()함수의 확장 버전, 다양한 형태로 데이터를 표현

```
ggplot(sort_kem3, aes(x=이름, y=점수, fill=과목))+geom_bar(stat="identity") + geom_text(aes(y=label,
label=paste(점수, "점")), colour="black", size=4)
```

```
gg2 <- ggplot(sort_kem3, aes(x=이름, y=점수, fill=과목)) + geom_bar(stat="identity") + geom_text(aes(y=label,
label=paste(점수, '점')), colour="black" , size=4) + guides(fill=guide_legend(reverse=T))
gg2 + theme(axis.text.x=element_text(angle=45, hjust=1, vjust=1, color="black", size=8))
```

```
#클리블랜드 점 그래프 geom_segment()
```

```
score<-read.table("./lab01/data/학생별전체성적_new.txt", header=T, sep=",")
```

```
score
```

```
score[, c("이름", "영어")]
```

```
ggplot(score, aes(x=영어, y=reorder(이름, 영어))) + geom_point(size=6) +
theme_bw()+theme(panel.grid.major.x=element_blank(), panel.grid.minor.x=element_blank(),
panel.grid.major.y=element_line(color="red", linetype="dashed"))
```

```
ggplot(score, aes(x=영어, y=reorder(이름, 영어))) + geom_segment(aes(yend=이름), xend=0,
color="blue")geom_point(size=6, color="green") + theme_bw()+theme(panel.grid.major.x=element_blank())
```

R - Chart

- ✓ ggplot2
 - plot()함수의 확장 버전, 다양한 형태로 데이터를 표현

```
install.packages("gridExtra")
library(gridExtra)
v_mt <- mtcars
v_mt
graph1 <- ggplot(v_mt, aes(x=hp, y=mpg))
graph1 + geom_point()
graph2 <- graph1 + geom_point(colour="blue") # 색상 변경하기
graph2
graph3 <- graph2 + geom_point(aes(colour=factor(am))) #종류별로 다른 색상 지정하기
graph3
graph4 <- graph1 + geom_point(size=7) #크기 지정하기
graph4
graph5 <- graph1 + geom_point(aes(size=wt)) #값 별로 다른 크기 지정하기
graph5
graph6 <- graph1 + geom_point(aes(shape=factor(am),size=wt)) #종류별로 크기와 모양 지정하기
graph6
```

R - Chart

✓ ggplot2

- plot()함수의 확장 버전, 다양한 형태로 데이터를 표현

```
graph7 <- graph1 + geom_point(aes(shape=factor(am), color=factor(am),size=wt))+  
scale_color_manual(values=c("red", "green")) #종류별로 크기와 모양 색상 지정하기  
graph7
```

```
graph8 <- graph1 + geom_point(color="red")+geom_line() #선추가하기  
graph8
```

```
graph9 <- graph1 + geom_point(color="blue")+labs(x="마력", y="연비(mile/gallon)") #x축과 y축 이름 바꾸기  
graph9
```

```
#geom_line()
```

```
three <- read.csv("./lab01/data/학생별과목별성적_3기_3명.csv", header=T)
```

```
three
```

```
sort_score <- arrange(three, 이름, 과목)
```

```
sort_score
```

```
ggplot(sort_score, aes(x=과목, y=점수, color=이름, group=이름)) + geom_line()
```

```
ggplot(sort_score, aes(x=과목, y=점수, color=이름, group=이름)) + geom_line() + geom_point() # 점 추가
```

R - Chart

✓ ggplot2

- plot()함수의 확장 버전, 다양한 형태로 데이터를 표현

```
ggplot(sort_score, aes(x=과목, y=점수, color=이름, group=이름, fill=이름)) + geom_line() + geom_point(size=6, shape=22)
```

```
# 선그래프의 선 아래 색으로 채우는 함수 : geom_area()
```

```
dis <- read.csv("./lab01/data/1군전염병발병현황_년도별.csv", stringsAsFactors=F)
```

```
dis
```

```
ggplot(dis, aes(x=년도별, y=장티푸스, group=1)) + geom_line()
```

```
ggplot(dis, aes(x=년도별, y=장티푸스, group=1)) + geom_area()
```

```
ggplot(dis, aes(x=년도별, y=장티푸스, group=1)) + geom_area(color="red", fill="cyan", alpha=0.4)
```

```
ggplot(dis, aes(x=년도별, y=장티푸스, group=1)) + geom_area(fill="cyan", alpha=0.4)
```

회귀분석

✓ 회귀분석

- 하나 혹은 그 이상의 독립변수들이 종속변수에 미치는 영향을 추정하는 통계기법
- 변수들 사이의 인과관계를 밝히고 모형을 적합하여 관심있는 변수를 예측하거나 추론하기 위한 분석 방법
- 영향을 받는 변수(Y) – 반응변수, 종속변수, 결과변수
- 영향을 주는 변수(X) – 설명변수, 독립변수, 예측변수

✓ 단순회귀분석

- 한 개의 종속변수와 한 개의 독립변수 간의 관계를 직선으로 표현하여 분석하는 방법
- 잔차 – 예측 값과 관측 값의 차이
- 회귀분석은 잔차를 최소화 할 수 있도록 절편과 기울기를 구함

✓ 회귀분석의 장점

- 결과를 통해 유효한 정보를 획득할 수 있고, 필요 없는 변수 선택을 통해 모델의 안전성을 높일 수 있음

✓ 회귀분석의 단점

- 사전에 결측치 처리 및 변수 간의 교호작용의 유무 및 비선형 여부를 파악

회귀분석

✓ 회귀분석의 절차

| 구분 | 활동 | 설명 | 비고 |
|---------|-------------------------|-----------------------------|------------------------------|
| 회귀모형설정 | 주요 설명변수, 종속변수 파악 | 경험이론 등을 토대로 인과 관계 모형 설정 | 비선형 관계인 경우 다항회귀, 비선형 회귀를 고려 |
| 선형성검토 | 산점도 작성을 통한 상관관계 파악 | 종속변수 vs 설명변수 설명변수 간 | 선형관계 여부 파악 이상치 판단 |
| 회귀식추정 | 모형의 회귀계수 및 유의성 확인 | 회귀변수 모델의 적정성 검토 | T-검정, 분산분석,F-검정 |
| 변수선택 | 중요 설명변수 선택 | Backward, Forward, Stepwise | 유의한 설명변수로 회귀모형 적합 |
| 다중공선성검토 | 설명변수 간 높은 상관관계를 파악하여 해소 | VIF, Condition Index, 상관계수 | 문제변수제거, PCA 능형회귀, 주성분회귀수행 |
| 적합된모형검토 | 적합된 회귀모형에 대한 검토 | 오차의 가정 체크 이상치 진단 등 | |
| 최종모형 | 설명변수의 설명력 파악 | 결정계수, 조정된 결정계수, | |

회귀분석

✓ 회귀모형에 대한 검토 방법

| 항목 | 설명 | 도구 |
|------|---|----|
| 선형관계 | 검토결과 - 포물선 3차식 형태의 비선형 관계 파악 해결방안 - 다항회귀고려(표준화) | |
| 등분산성 | 검토결과-나팔모양 해결방안-종속변수 log변환, 가중최소자승법사용 | |
| 정규성 | 검토결과-잔차의 정규성 파괴 해결방법-종속변수 log변환 | |
| 독립성 | 검토결과 - 잔차의 양의 자기상관(0에 가까울수록) 음의 자기상관(4에 가까울수록) 해결방법-설명변수 차분 | |
| 이상치 | 검토결과- 표준잔차 >2, 이상치 스튜던트잔차 >2, 영향치 | |

회귀분석

- ✓ 회귀분석의 가정
 - 선형성 – 독립변수의 변화에 따라 종속변수도 일정크기로 변화
 - 독립성 – 잔차와 독립변인의 값이 관련되어 있지 않음
 - 등분산성 – 독립변인의 모든 값에 대해 오차들의 분산이 일정
 - 비상관성 – 관측치들의 잔차끼리 상관이 없어야 함
 - 정상성 – 잔차 향이 정규분포를 이루어야 함
- ✓ 회귀분석의 가정조건을 검증해보는 가장 쉬운 방법은 각 가정조건에 해당하는 line graph를 그려보는 것이다.
- ✓ 회귀분석의 체크포인트 요약
 - 모형이 통계적으로 유의미한가? – F통계량 확인
 - 회귀계수들이 유의미한가? – 계수에 대한 p-value, t 통계량, 신뢰구간 확인
 - 모형이 얼마나 설명력을 지니는가? – 결정계수, 수정된 결정계수
 - 모형이 데이터를 잘 적합하고 있는가? – Residual plot, 회기진단
 - 데이터가 전제하고 있는 가정을 만족시키는가? – 회귀진단
 - 선형성 – 독립변수의 변화에 따라 종속변수도 일정크기로 변화

기초 통계

✓ 통계학

- 관심의 대상에 대한 불확실한 특정 사실이 data나 information을 수집/정리/요약하고 과학적인 판단을 할 수 있는 방법을 연구하는 학문
- 자료를 효율적으로 수집
- 자료를 명확하게 정리
- 자료에서 현재 상태를 파악하거나 가설 검증, 예측

✓ 통계학 유형

- 기술통계학(Descriptive Statistics) - 수집된 자료를 그래프나 표, 몇가지 수치 자료를 정리하고 요약하여 자료의 특성을 나타내는 분석 방법
- 추론통계학(Inference Statistics) - 실제로 관측된 자료를 이용하여 모집단의 특성에 대하여 추측하는 분석 방법
- 모수통계학(Parametric Statistics) - 통상 모집단에 분포를 가정하고 분석하는 방법
- 비모수통계학(Non Parametric Statistics) - 모집단의 가정이 없이 분석하는 방법

기초 통계

✓ 기술통계

- 대푯값(평균, 중위수, 최빈수 등), 산포(분포/왜도/첨도 등)를 제시하거나 도식화하는 절차로 구성

✓ 추론통계

- 표본의 정보로 모집단을 추측하기 위한 절차로 구성
- 모수통계, 비모수통계로 구성

✓ 모집단과 표본

- 빅데이터에서는 모집단이 전체 데이터를 이용해서 분석하는 것을 기본적으로 가정하나 이를 지원하는 시스템의 성능이 이를 받쳐주지 못할 때는 표본 데이터를 이용해서 분석해야 한다.
- 모집단(Population) : 통계적인 관심의 대상이 되는 모든 개체에 대한 전체 집합, 수집 가능한 관측 값을 가지게 된다.
- 모수(Parameter) : 모집단의 특성치
- 표본(Sample) : 모집단에서 추출된 관측 값의 부분 집합
- 통계량(Statistic) : 표본의 특성치

기초 통계

✓ 모수적 통계 분석 기법

- 중심극한정리에 의해 본래의 분포에 상관없이 무작위 추출을 한 집단의 평균은 정규분포를 갖는다것에 기반하여 집단 간의 비교를 평균을 통해 실시한다

| 구분 | 설명 | 기법 |
|----------------|--------------------------|--|
| 빈도분석 | 변수의 분포특성과 중심치 특성을 파악 | 사분위수 산포도 평균 분산 범위 |
| 상관분석 | 연속형 두 변수 간의 관계를 상관계수로 분석 | 단순상관분석(두 변수 간) 다중상관분석(한 변수와 두 변수 이상 변수간 상관) 부분상관분석(다른 변수 상관관계를 통제하고 순수한 두 변수 간 상관) |
| 표본평균검증 | 두 집단 간의 평균 값을 비교 | 단일 표본 T-test(한 모집단에서 추출된 표본 검정) 독립표본 T-test(독립 두 모집단 추출된 두 표본 간의 평균 차이 검정) Paired T-test(동일 모집단에서 추출된 두 표본 간의 평균 차 검정) |
| 세 집단 이상의 평균 검정 | 세 집단 이상의 평균 값을 비교 | 단일변량분산분석(ANOVA) 다변량분산분석(MANOVA) |
| 회귀분석 | 인과관계 분석 및 예측 | 선형회귀 – 단순회귀, 중회귀, 다항회귀 비선형회귀 – 곡선회귀, 로짓프로빗 로지스틱회귀분석 |

기초 통계

✓ 비모수적 통계 분석 기법

- 정규성 검정에서 정규분포를 따른다는 가정이 충족되지 못한 경우, 모집단에 대한 가정이 필요 없으며, 질적자료나 양적 자료 중 명목척도나 순서 척도로 측정된 자료를 분석하는데 활용한다.

| 구분 | 설명 |
|------------|--|
| 적합도 검정 | 단일 표본 카이스퀘어 검정 단일표본 K-S 검정 이항분포 검정 |
| 변수 간 상관 분석 | 스피어만 순위 상관분석 켄달의 상관분석 카이 제곱 분석 |

기초 통계

✓ 표분추출 분류

| 구분 | 설명 | |
|---------------------------------------|---|--|
| 단순랜덤샘플링 (Random Sampling) | 복원, 비복원 추출 | sample(1:100, 5) sample(1:100, 5, replace=T) sample(1:100, 5, replace=T, prob=1:100) |
| 계통샘플링 (Systemtric Sampling) | 임의 위치에서 K번째 항목 추출 | |
| 층화샘플링 (Stratified Random Sampling) | 명확하게 다른 data를 중첩 없이 분할하여 샘플링 | |
| 집락추출(Cluster Sampling) | 군집을 구분하고 군집별로 단순 랜덤 샘플링한 후 모든 자료를 활용하는 방법 | |

기초 통계

✓ 자료의 종류

- 질적 변수(Qualitative Variable) - 범주형 변수 (Categorical Variable)
- 양적변수(Quantitative Variable) - 연속형 변수(Continuous Variable), 이산형 변수(Discrete Variable)

✓ 자료의 형태에 따른 분류

- 명목척도 자료 - 단지 구분하기 위한 부호로 표현된 자료(성별, 혈액형, 주거형태, 학과)
- 서열척도 자료 - 관측되는 개체들에 순서를 갖는 자료 (학년, 선호도, 신용등급)
- 구간척도 자료 - 관측 값의 순서와 순서사이의 간격이 의미가 있는 자료(성적, 온도 등)
- 비율척도 자료 - 간격(차이)에 대한 비율이 의미를 가지는 자료(연령, 시간, 거리)

기초 통계

✓ 질적자료 요약 기법

| 기법 | 설명 | 특징 | 비교 |
|-------|--------------------------------------|---|-----------|
| 도수분포표 | 범주형 건수 및 비율 파악 | 도수 빈도, 상대도수, 누적도수, 누적 상대도수 | 1개의 변수 요약 |
| 막대그래프 | 범주별 도수 및 상대도수를 직사각형 막대 모양으로 그린 그래프 | 막대의 높이가 상대도수에 비례 | |
| 원그래프 | 원을 범주별 상대도수에 비례하여 중심각을 나눈 그래프 | 비율파악 및 비교 범주의 상대도수와 360을 곱하여 원조각의 각을 계산 | |
| 분할표 | 범주형 변수 두 개의 관측 값의 패턴을 파악하기 위해 작성하는 표 | 두 변수의 범주를 행열에 배치하고 범주들이 교차하는 셀에 도수 표시 | 2개 변수 요약 |

기초 통계

✓ 양적자료 요약 기법

| 기법 | 설명 | 특징 |
|---------|-------------------------------------|---|
| 히스토그램 | 도수 분표를 표로 나타낸 도표 | 도수 빈도, 상대도수, 누적도수, 누적 상대도수 |
| 줄기 잎 그림 | 자료의 분포를 시각적으로 확인 하는 도표 | 막대의 높이가 상대도수에 비례 |
| 선 그래프 | x축의 연속 변수의 변화에 따른 y축의 연속변수를 표현하는 도표 | 비율파악 및 비교 범주의 상대도수와 360을 곱하여 원조각의 각을 계산 |
| 산점도 | 두 연속형 변수에 대한 관측값의 패턴을 파악하기 위한 도표 | 두 변수의 범주를 행열에 배치하고 범주들이 교차하는 셀에 도수 표시 |