

2016 MACHINE LEARNING CAMP

Introduction to R

강필성

고려대학교 산업경영공학부

pilsung_kang@korea.ac.kr

Table of Contents

I	R Introduction
II	형태에 따른 데이터 처리
III	텍스트 데이터 처리
IV	조건문과 반복문
IV	함수

History of



```

RGui [R Console]
File Edit Misc Packages Windows Help

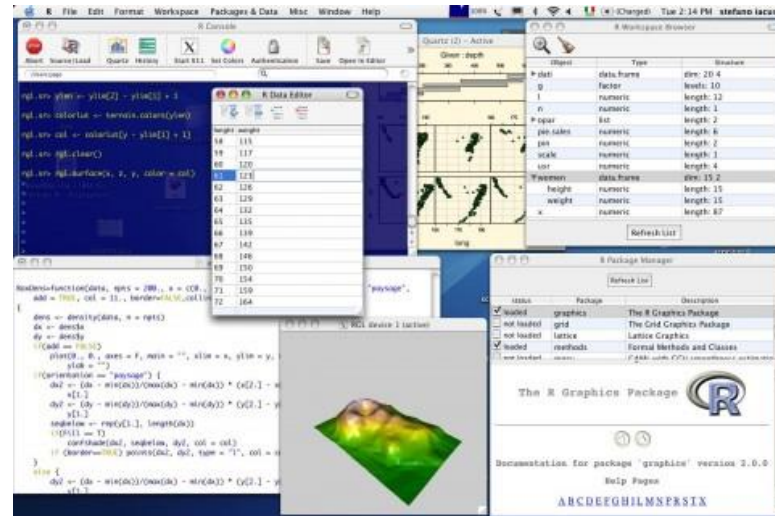
R : Copyright 2001, The R Development Core Team
Version 1.3.0 (2001-06-22)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.


R is a collaborative project with many contributors.
Type 'contributors()' for more information.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for a HTML browser interface to help.
Type 'q()' to quit R.

>
  
```



- ❖ R은 뉴질랜드 오클랜드(Auckland) 대학의 로스 이하카(Ross Ihaka)와 로버트 젠틀맨(Robert Gentleman) 교수의 주도로 개발
- ❖ S-PLUS의 기초적 프로그램을 작성했던 Chambers 등 많은 통계학 및 컴퓨터 관련 학자들이 R 개발 핵심 팀으로써 R을 만드는데 참가
- ❖ 1997년 8월 R언어를 구체화하기 위한 국제적인 논의가 진행되었으며, 2000년 2월에 버전 1.0.0이 등장해서 현재에 이르고 있음 (3.0.0)
- ❖ 많은 통계학 및 계량분석자들이 R을 보다 잘 이용할 수 있도록 응용 소프트웨어를 개발하고 있으며 이를 R에 덧붙여서 쓰면 매우 유용

- 

About R
What is R?
Contributors
Screenshots
What's new?


Download, Environments
CRAN

R Project
Foundation
Members & Donors
Making R
Bug Tracking
Developer Page
Conferences
Search

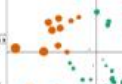
Documentation
Manuals
FAQs
The R Journal
Wiki
Books
Certification
Other

Help
Bioconductor
Related Projects
User Groups
Links


The R Project for Statistical Computing




PCA: 2 plots



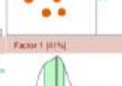
Clustering: 4 groups




Density



Factor 1 (10%)



Factor 2 (10%)



Density

Getting Started:

 - R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS. To [download R](#), please choose your preferred [CRAN mirror](#).
 - If you have questions about R like how to download and install the software, or what the license terms are, please read our [answers to frequently asked questions](#) before you send an email.

News:

 - R version 2.15.0 (Easter Henge)** has been released on 2012-03-30.
 - R version 2.14.2 (Gad-Getting Serious)** has been released on 2012-02-29.
 - The R Journal** Vol.3/2 is available.
 - useR! 2012**, will take place at Vanderbilt University, Nashville Tennessee, USA, June 12-15, 2012.

This server is hosted by the [Institute for Statistics and Mathematics](#) of the [WU Wien](#)

[illegible]

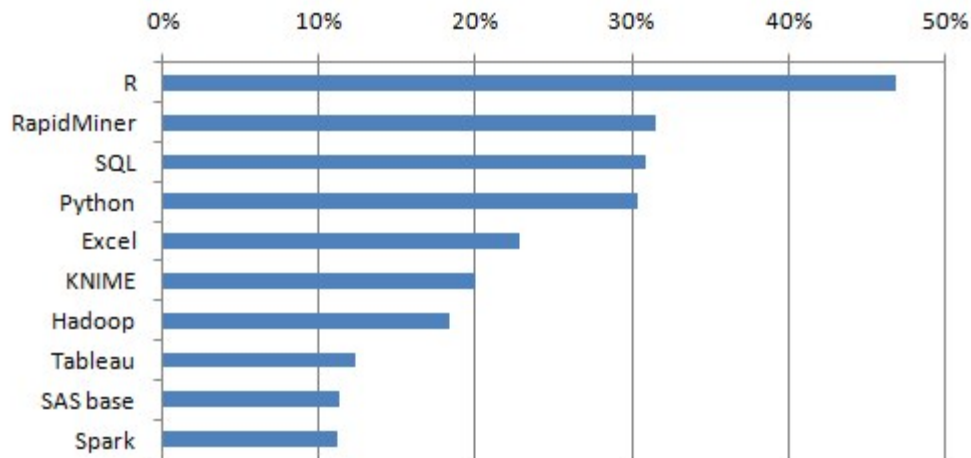
4

Why R?

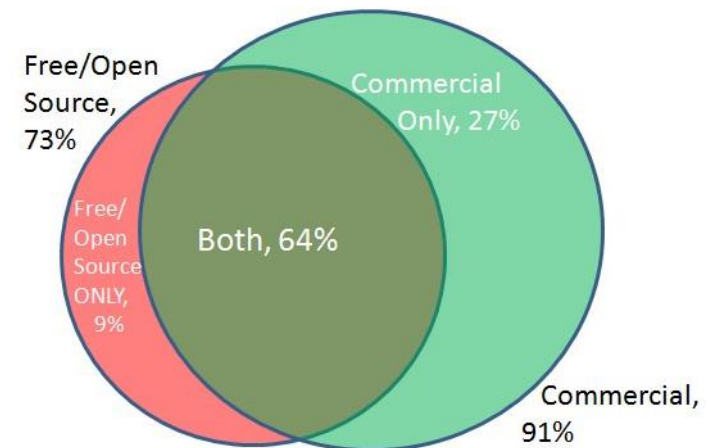
❖ R을 이용한 데이터 분석: 새로운 트렌드! (파이썬도 포함해서...)

- 다양한 분야의 연구자들이 데이터 분석을 위해 R을 사용하고 있음
- Kdnuggets & Kaggle poll results

Top Analytics, Data Mining, Data Science software used, 2015



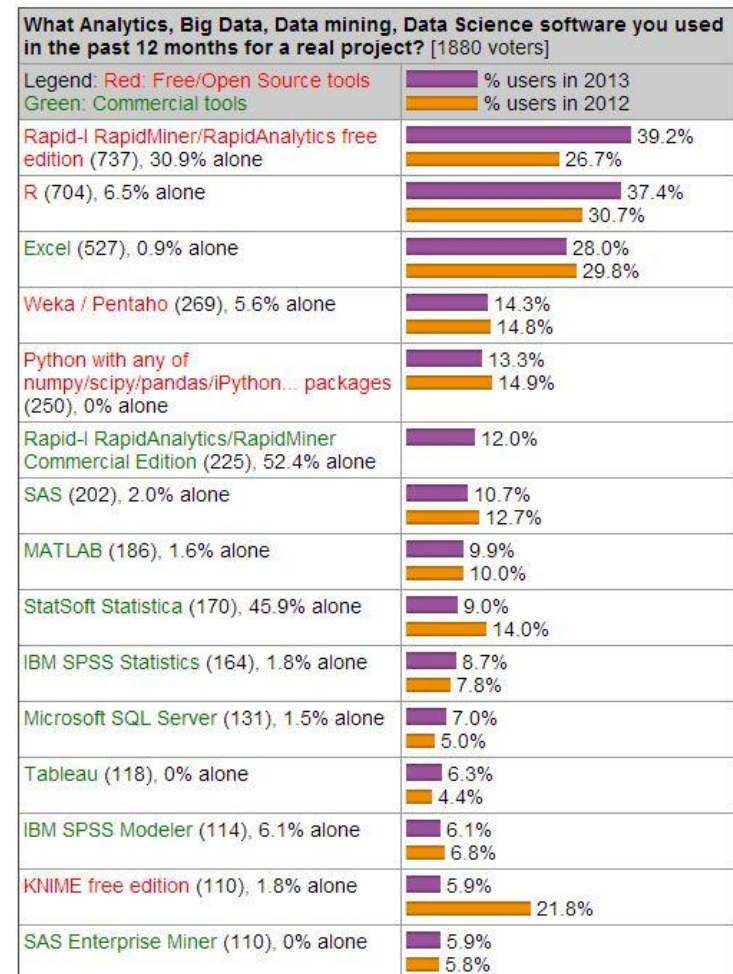
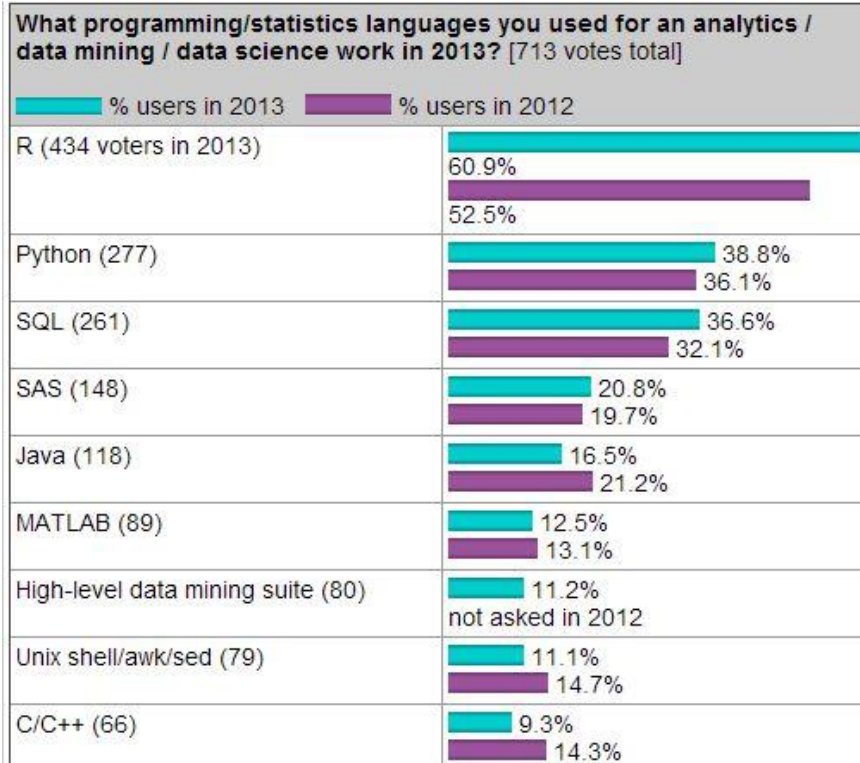
Analytics, Data Mining, Data Science Software Usage, 2015



Why R?

❖ R을 이용한 데이터 분석: 새로운 트렌드!

- 다양한 분야의 연구자들이 데이터 분석을 위해 R을 사용하고 있음



R이냐 Python이냐?

Your primary programming language for Analytics, Data Mining, Data Science tasks: [512 voters]

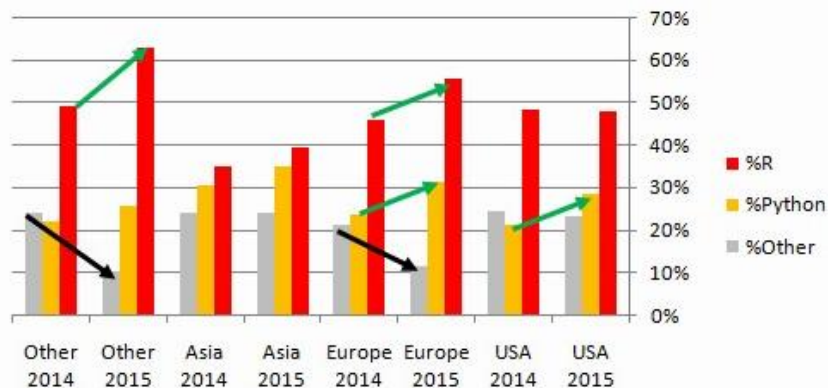
2015 primary programming language:

R (and its packages) (263)	<div style="width: 51%;"></div> 51% (of 2015 votes)
Python (including scikit-learn and other libraries) (151)	<div style="width: 29%;"></div> 29%
Other (Java, MATLAB, SAS, Scala, etc) (89)	<div style="width: 17%;"></div> 17%
none (9)	<div style="width: 1.8%;"></div> 1.8%

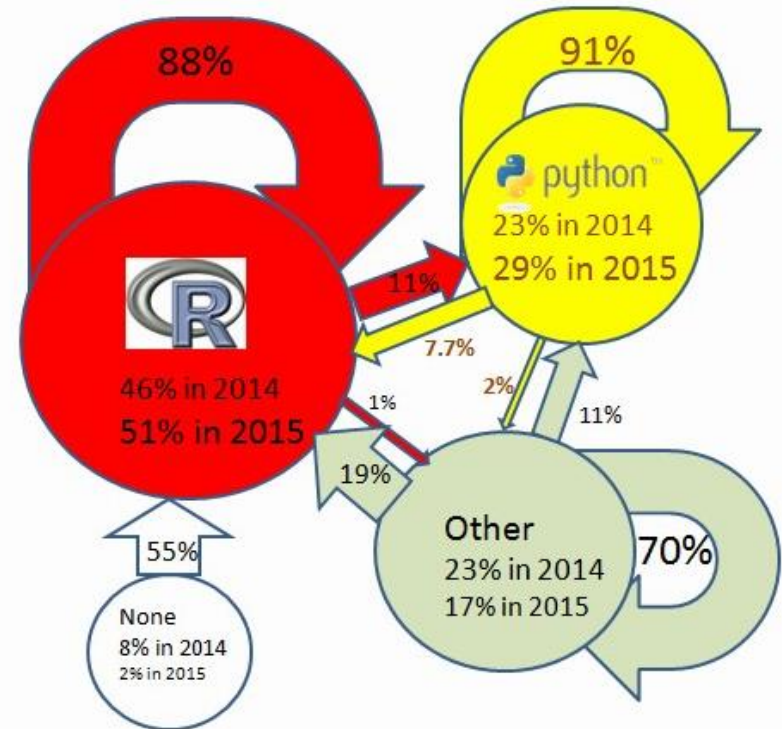
2014 primary programming language:

R (and its packages) (237)	<div style="width: 46%;"></div> 46% (of 2014 votes)
Python (including scikit-learn and other libraries) (117)	<div style="width: 23%;"></div> 23%
Other (Java, MATLAB, SAS, Scala, etc) (118)	<div style="width: 23%;"></div> 23%
none (40)	<div style="width: 7.8%;"></div> 7.8%

R & Python change by region, 2014-2015

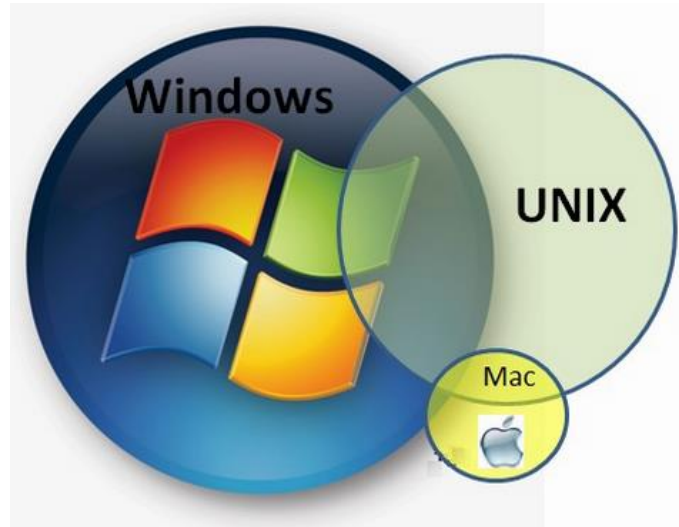
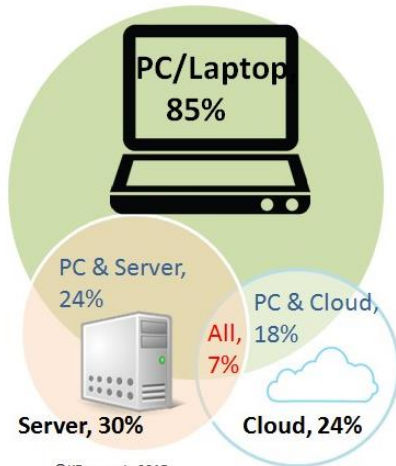


Primary Analytics, Data Mining, Data Science Languages in 2014 vs 2015

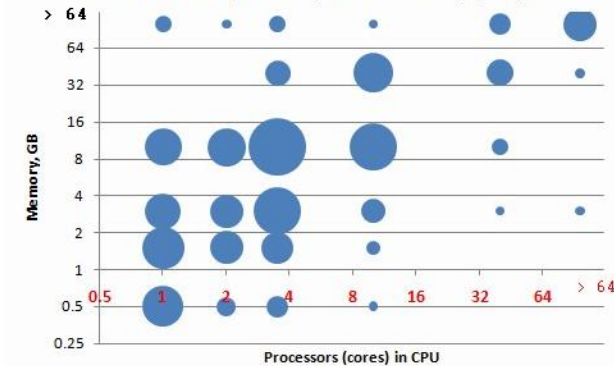


데이터 크기와 연산 장치 스펙

Platforms for Analytics/Data Mining



Analytics, Data Mining Computing:
Processors (cores) vs Memory (GB)



How many processors/cores your typical data mining job actually uses?

1 core (69)	24%
2 cores (45)	16%
3-4 cores (86)	30%
5-16 cores (51)	18%
17-64 cores (17)	6.0%
> 64 cores (14)	5.0%

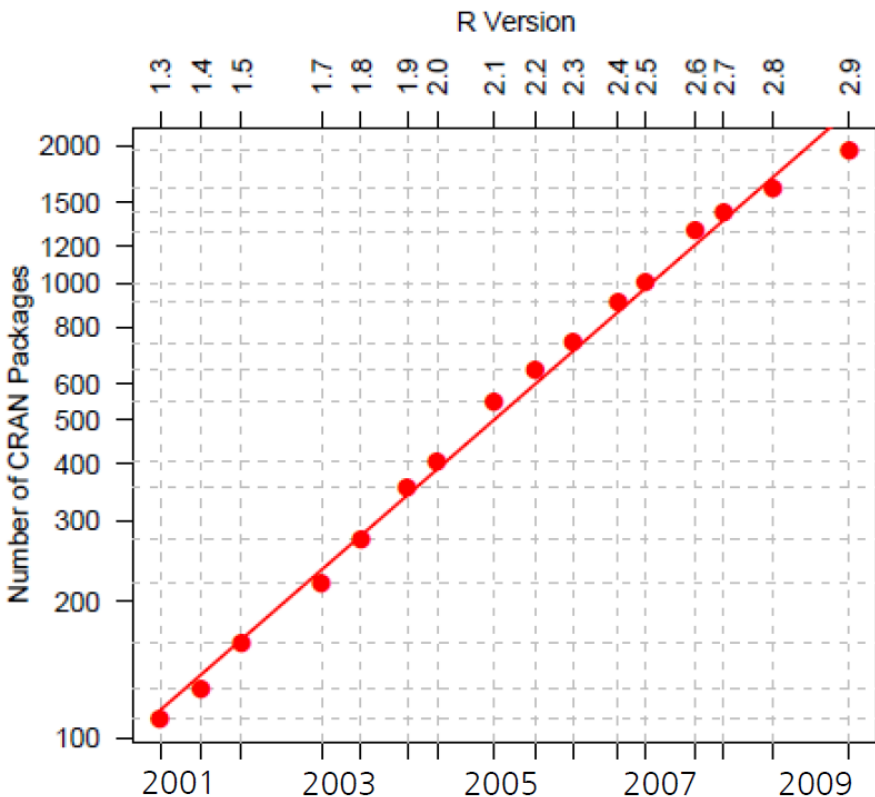
How much memory your typical data mining job actually uses:

< 1 GB (28)	10%
1-2 GB (44)	16%
2-4 GB (58)	21%
5-16 GB (94)	33%
17 - 64 GB (33)	12%
> 64 GB (25)	9%

Why R?

❖ R packages

- 누구나 R 패키지를 생성하고 배포할 수 있음
- (거의) 모든 통계 분석 방법론이 이미 패키지로 개발되어 배포 중임

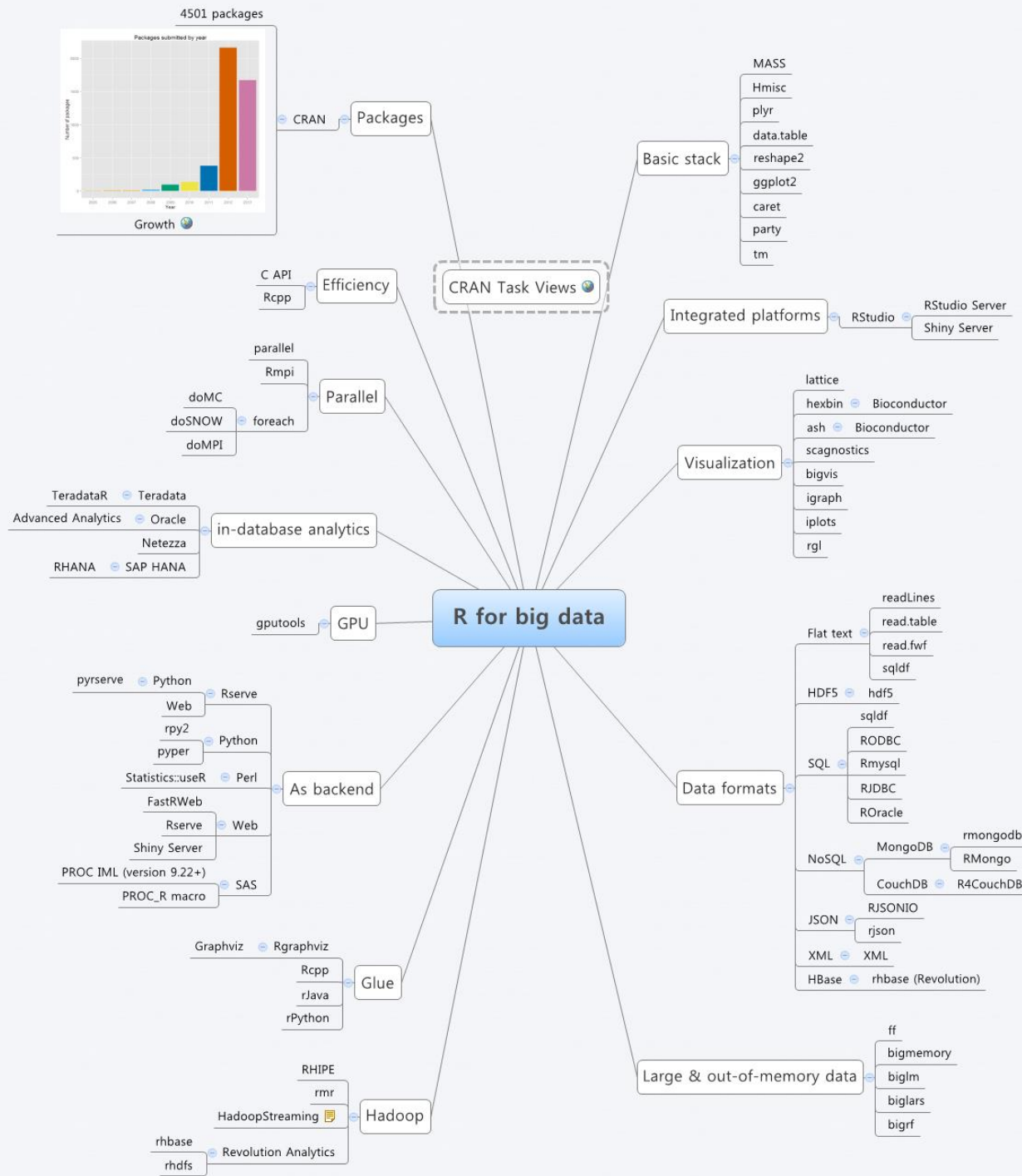


< The number of packages and R version on CRAN >

Bayesian	Bayesian Inference
ChemPhys	Chemometrics and Computational Physics
ClinicalTrials	Clinical Trial Design, Monitoring, and Analysis
Cluster	Cluster Analysis & Finite Mixture Models
Distributions	Probability Distributions
Econometrics	Computational Econometrics
Environmetrics	Analysis of Ecological and Environmental Data
ExperimentalDesign	Design of Experiments (DoE) & Analysis of Experimental Data
Finance	Empirical Finance
Genetics	Statistical Genetics
Graphics	Graphic Displays & Dynamic Graphics & Graphic Devices & Visualization
HighPerformanceComputing	High-Performance and Parallel Computing with R
MachineLearning	Machine Learning & Statistical Learning
MedicalImaging	Medical Image Analysis
Multivariate	Multivariate Statistics
NaturalLanguageProcessing	Natural Language Processing
OfficialStatistics	Official Statistics & Survey Methodology
Optimization	Optimization and Mathematical Programming
Pharmacokinetics	Analysis of Pharmacokinetic Data
Phylogenetics	Phylogenetics, Especially Comparative Methods
Psychometrics	Psychometric Models and Methods
ReproducibleResearch	Reproducible Research
Robust	Robust Statistical Methods
SocialSciences	Statistics for the Social Sciences
Spatial	Analysis of Spatial Data
Survival	Survival Analysis
TimeSeries	Time Series Analysis

< Example: packages available on CRAN >

R for Big Data

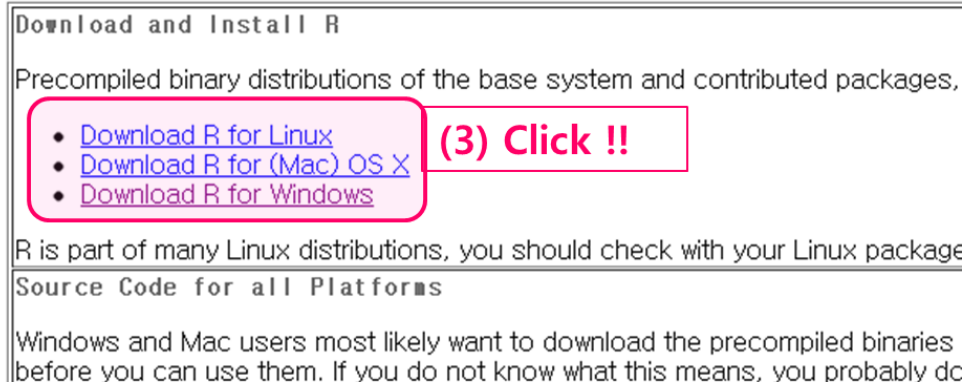
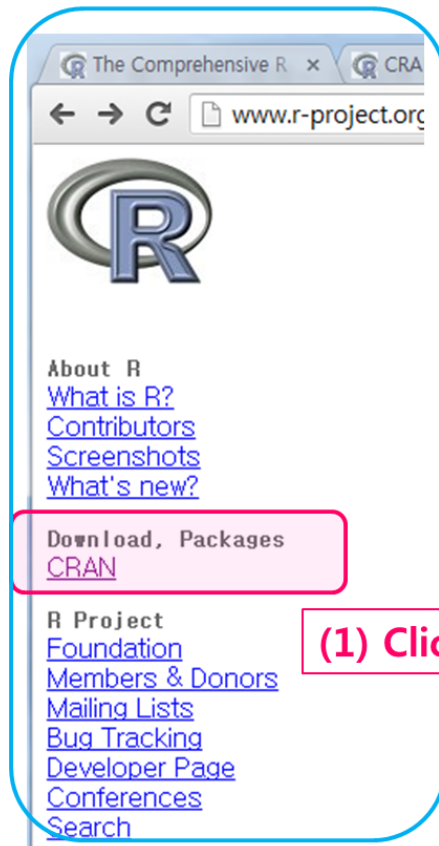


<http://www.r-bloggers.com/stepping-up-to-big-data-with-r-and-python-a-mind-map-of-all-the-packages-you-will-ever-need/>

Install R

❖ For Windows

- <http://www.r-project.org>
- Download 'R base' file: the latest version is 3.1.3 for windows

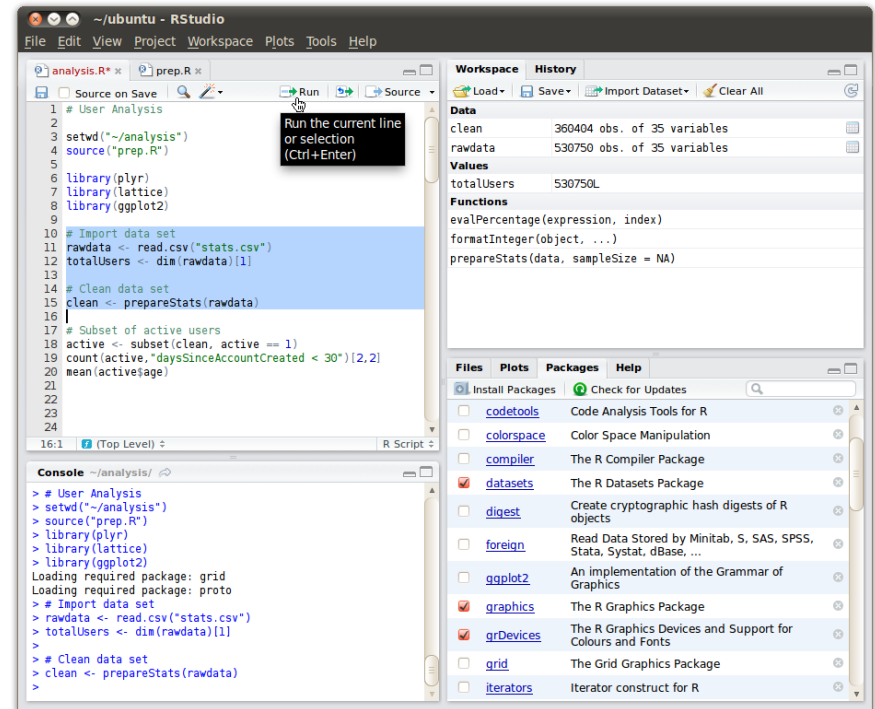
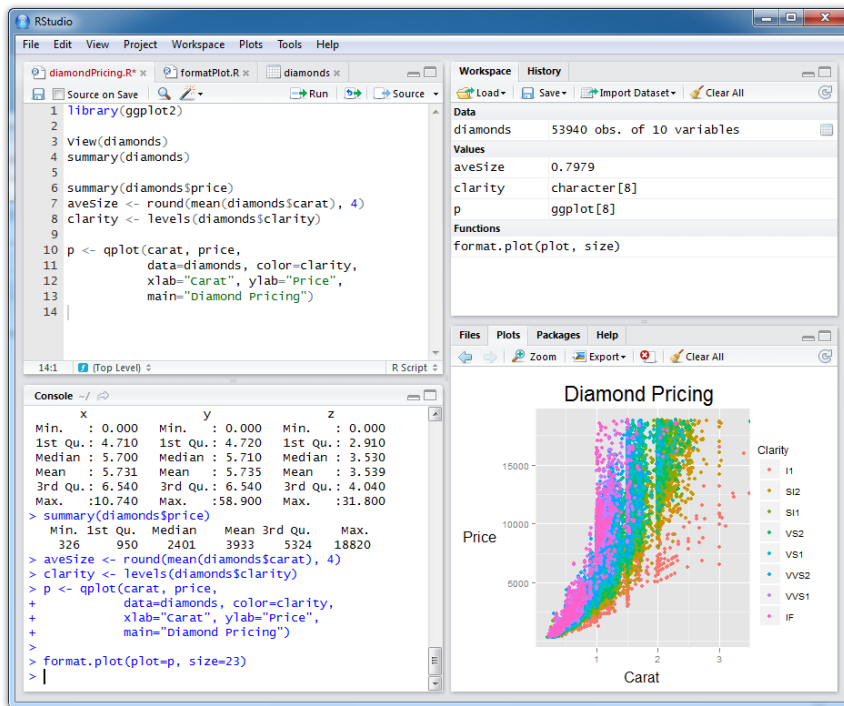


Install RStudio



■ R 언어의 대표적인 통합 개발 환경 (IDE) 소프트웨어 (오픈소스 기반 무료)

- ✓ Available OS: Windows, Mac, or Linux
- ✓ 로컬 및 서버 환경에서 실행 가능



Install RStudio

RStudio

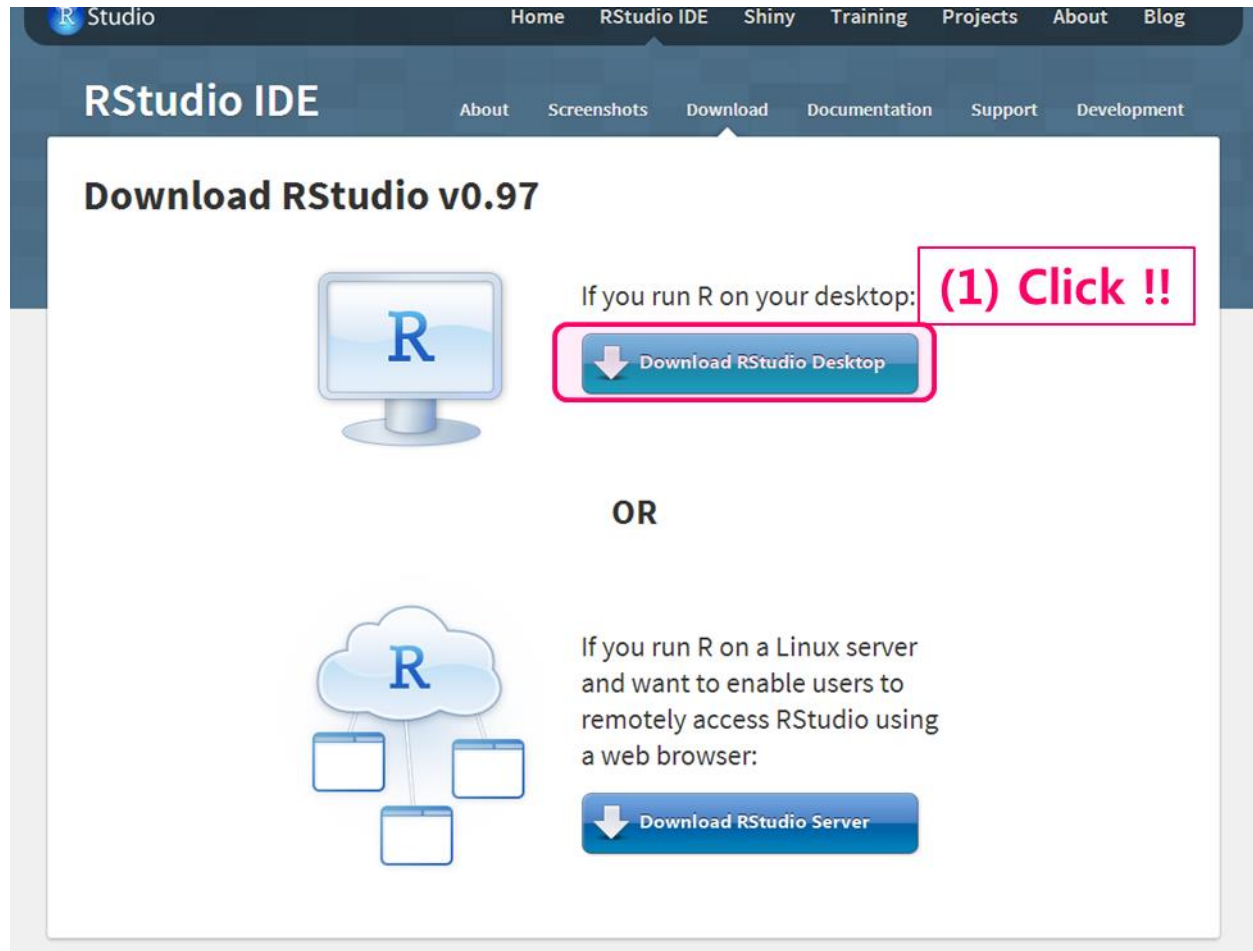
The screenshot shows the RStudio environment with four callout boxes highlighting key features:

- View:**
 - (1) View data
 - (2) View R script
- Workspace:**
 - (1) 데이터, 변수 저장
 - (2) 명령어 기록(history) 저장
- Console:** 명령어 입력, 결과 출력
- Help:** Statistical Data Analysis, Manuals, Reference

Install RStudio

❖ RStudio homepage

- <http://www.rstudio.com>



R warming up #0: Data structure

❖ 정형 데이터: Structured data

- 행렬/테이블 형태의 데이터로서 각 행을 관측치를 의미하며 각 열은 변수(속성)을 의미함
- 관계형 DB 적재 용이

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa
8	5.0	3.4	1.5	0.2	setosa
9	4.4	2.9	1.4	0.2	setosa
10	4.9	3.1	1.5	0.1	setosa
11	5.4	3.7	1.5	0.2	setosa
12	4.8	3.4	1.6	0.2	setosa
13	4.8	3.0	1.4	0.1	setosa
14	4.3	3.0	1.1	0.1	setosa
15	5.8	4.0	1.2	0.2	setosa
16	5.7	4.4	1.5	0.4	setosa
17	5.4	3.9	1.3	0.4	setosa
18	5.1	3.5	1.4	0.3	setosa
19	5.7	3.8	1.7	0.3	setosa
20	5.1	3.8	1.5	0.3	setosa
21	5.4	3.4	1.7	0.2	setosa
22	5.1	3.7	1.5	0.4	setosa
23	4.6	3.6	1.0	0.2	setosa
24	5.1	3.3	1.7	0.5	setosa
25	4.8	3.4	1.9	0.2	setosa
26	5.0	3.0	1.6	0.2	setosa
27	5.0	3.4	1.6	0.4	setosa
28	5.2	3.5	1.5	0.2	setosa
29	5.2	3.4	1.4	0.2	setosa

	A	B	C	D	E	F	G	H	I	J
1		강봉균	강운태	고흥길	권영세	김무성	김부겸	김성순	김성조	김영선
2	강봉균	0	0.384961	0.788574	0.597168	0.73252	0.850977	0.749609	0.700586	0.669434
3	강운태	0.384961	0	0.330371	0.31543	0.308105	0.384961	0.362891	0.33125	0.330273
4	고흥길	0.788574	0.330371	0	0.687695	0.788477	0.743262	0.668457	0.783301	0.677344
5	권영세	0.597168	0.31543	0.687695	0	0.652148	0.587207	0.619434	0.674902	0.645313
6	김무성	0.73252	0.308105	0.788477	0.652148	0	0.687793	0.605371	0.750098	0.636914
7	김부겸	0.850977	0.384961	0.743262	0.587207	0.687793	0	0.820508	0.681445	0.651855
8	김성순	0.749609	0.362891	0.668457	0.619434	0.605371	0.820508	0	0.641211	0.642676
9	김성조	0.700586	0.33125	0.783301	0.674902	0.750098	0.681445	0.641211	0	0.678418
10	김영선	0.669434	0.330273	0.677344	0.645313	0.636914	0.651855	0.642676	0.678418	0
11	김영환	0.576367	0.145117	0.504492	0.416309	0.505176	0.573242	0.574316	0.463184	0.458398
12	김일준	0.049414	0.047559	0.036914	0.042578	0.043652	0.040137	0.04668	0.038672	0.04248
13	김충조	0.876563	0.388672	0.755371	0.554688	0.708203	0.886719	0.7625	0.699316	0.622754
14	김학송	0.79209	0.338672	0.856836	0.638281	0.797461	0.740723	0.665137	0.77041	0.683008
15	김형오	0.539551	0.27334	0.497461	0.589844	0.548633	0.51416	0.584668	0.553809	0.591016
16	김효석	0.852051	0.37832	0.760742	0.579492	0.700586	0.844043	0.792871	0.700488	0.64082
17	남경필	0.623633	0.303906	0.734863	0.712207	0.685645	0.607227	0.605078	0.695313	0.635059
18	박근혜	0.689648	0.299414	0.743457	0.705371	0.716504	0.701758	0.706836	0.677344	0.677051
19	박병석	0.856445	0.381348	0.771777	0.572363	0.719629	0.857422	0.769336	0.69043	0.656348
20	박상천	0.763574	0.389258	0.666797	0.602148	0.647656	0.811035	0.814355	0.627832	0.609766
21	박종근	0.674902	0.30791	0.746875	0.673242	0.723633	0.636816	0.634668	0.706543	0.633789
22	박종희	0.168359	0.175391	0.190137	0.184082	0.191016	0.159375	0.13125	0.190137	0.17793
23	박주선	0.774902	0.376758	0.686328	0.557813	0.690234	0.789355	0.77373	0.677832	0.611328
24	박진	0.687988	0.313965	0.804883	0.686133	0.731641	0.66377	0.665918	0.738184	0.668945

Install RStudio

❖ RStudio homepage

- <http://www.rstudio.com>

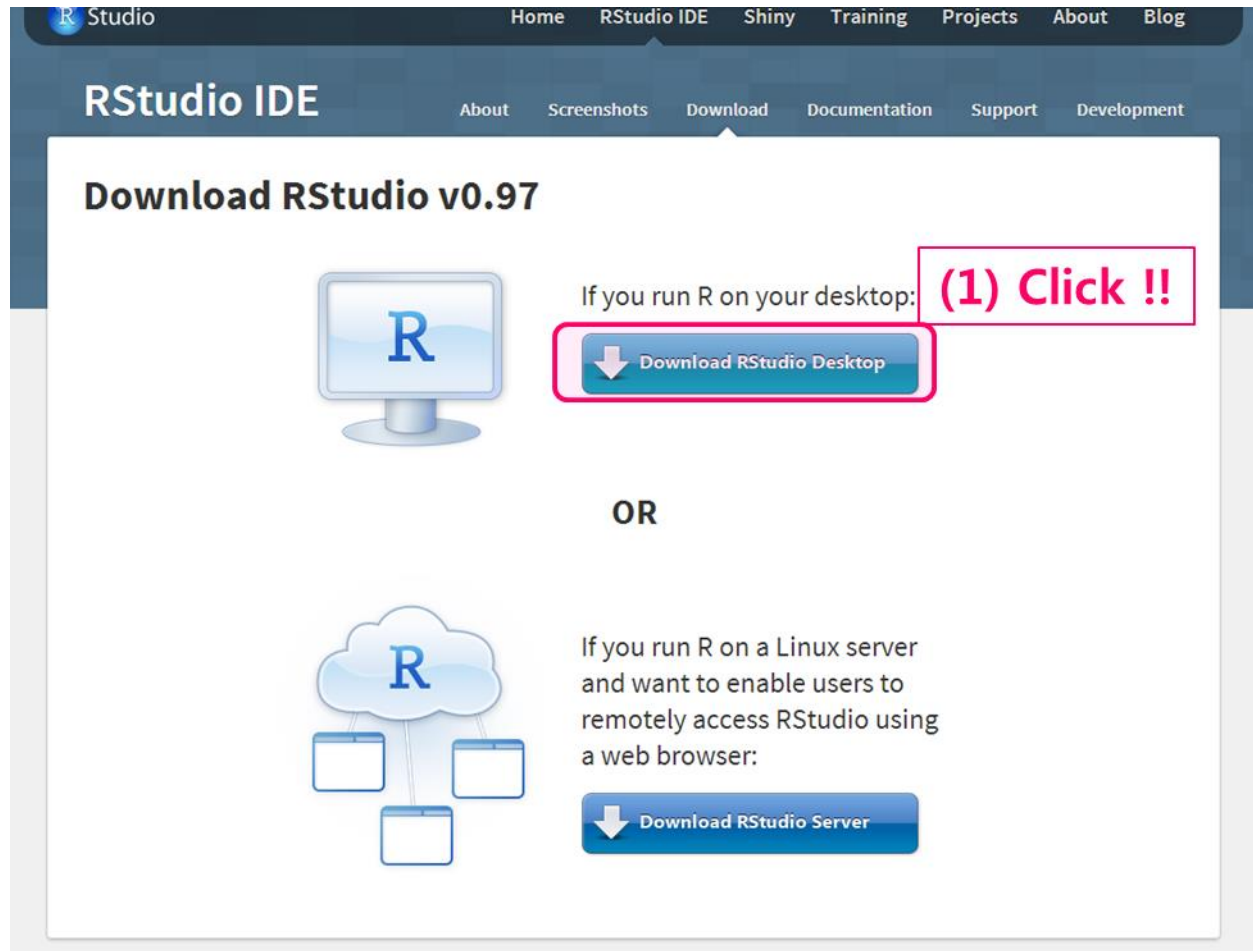


Table of Contents

I	R Introduction
II	형태에 따른 데이터 처리
III	텍스트 데이터 처리
IV	조건문과 반복문
IV	함수

Data Type in R

Scalar

Vector

List

Matrix

Array

Factor

Data.frame

❖ 벡터 (Vector)

- "벡터들은 동질적이다"
 - ✓ 한 벡터의 모든 원소는 같은 자료형(mode)을 가지고 있어야 한다
- "벡터는 위치로 인덱스된다"
 - ✓ 여러 개의 위치로 인덱스될 수 있으며, 이 때 하위 벡터(subvector)를 반환한다
 - ✓ 인덱스는 1부터 시작
- "벡터 원소들은 이름을 가질 수 있다"

R에서 벡터는 기준은 열(column)이다!

```

Introduction to R.R x
← → Source on Save 🔍 📌
1 # Assign values to the vector A & B
2 A <- c(1,2,3)
3 B <- c(1, "A", 0.5)
4
5 # Check the mode
6 mode(A)
7 mode(B)
8
9 # Select a subset of vector
10 A[1]
11 A[2:3]
12 A[c(2,3)]
13
14 # Assign names
15 names(A)
16 names(A) <- c("First", "Second", "Third")
17
18 # call by index or name
19 A[1]
20 A["First"]
  
```

Data Type in R

Scalar

Vector

List

Matrix

Array

Factor

Data.frame

❖ 리스트 (List)

- "리스트는 이질적이다"
 - ✓ 리스트의 원소들은 서로 다른 모드가 가능하다
 - ✓ 리스트나 데이터 프레임과 같이 구조화된 다른 객체들도 포함할 수 있다
- "리스트는 위치로 인덱스된다"
 - ✓ 단일 원소를 참조할 때는 대괄호 두 개
 - ✓ 복수 원소들을 참조할 때는 대괄호 하나
- "리스트의 원소들은 이름을 가질 수 있다"

```

22 # Example of a list
23 listA <- list(1, 2, "a")
24 print(listA)
25 listA[[1]]
26 listA[c(1,2)]
27 names(listA)
28 names(listA) <- c("First", "Second", "Third")
29
30 listA[["Third"]]
31 listA$Third

```


Data Type in R

Scalar

Vector

List

Matrix

Array

Factor

Data.frame

❖ 행렬 (Matrix)

- "행렬은 차원을 가진 벡터이다"
 - ✓ 벡터와 리스트를 행렬로 변환할 수 있다(dim 명령어 사용)

❖ 배열 (Array)

- "행렬은 2차원을 넘어 n차원으로 확장될 수 있다"
 - ✓ 여러 개의 위치로 인덱스될 수 있으며, 이 때 하위 벡터(subvector)를 반환한다

```

33 # Example of a matrix
34 A <- 1:6
35 dim(A)
36 print(A)
37
38 dim(A) <- c(2,3)
39 print(A)
40
41 B <- list(1,2,3,4,5,6)
42 print(B)
43 dim(B)
44 dim(B) <- c(2,3)
45 print(B)
46
47 D <- 1:12
48 dim(D) <- c(2,3,2)
49 print(D)

```

Data Type in R

Scalar

Vector

List

Matrix

Array

Factor

Data.frame

❖ 요인 (Factor)

- "열거형 값들로 이루어진 벡터"
 - ✓ 벡터에 있는 고유한 값(수준, level)을 읽어내어 정보로 함께 저장
- 요인의 사용처
 - ✓ 범주형 변수(Categorical variable) 표현: 한 요인으로 하나의 카테고리를 표현
 - ✓ 집단분류(grouping): 데이터 항목의 집단에 따라 label을 붙이거나 태깅

나중에 데이터 간의 요인 수준이 맞지 않아 문제가 발생할 수 있음!

```

51 # Example of a factor
52 A <- c("Cho", "Kim", "Kang")
53 B <- as.factor(A)
54
55 print(A)
56 print(B)
57
58 mode(A)
59 mode(B)
60
61 A[1]+A[2]
62 B[1]+B[2]

```

```

> mode(A)
[1] "character"
> mode(B)
[1] "numeric"
> A[1]+A[2]
Error in A[1] + A[2] : non-numeric argument to binary operator
> B[1]+B[2]
[1] NA
Warning message:
In Ops.factor(B[1], B[2]) : + not meaningful for factors

```

Data Type in R

Scalar

Vector

List

Matrix

Array

Factor

Data Frame

❖ 데이터 프레임 (Data Frame)

- 열과 행이 있는 테이블(사각형)로 된 데이터 구조
 - ✓ 행렬이라기보다 리스트에 가까움
 - ✓ (일반적으로) 벡터와 요인을 열(column)로 갖는다
 - ✓ 모든 열의 길이는 동일해야 한다
 - ✓ 모든 열은 이름을 가질 수도 있다

```
64 # Example of data frame
65 A <- c(1,2,3)
66 B <- c("a","b","c")
67 C <- data.frame(A,B)
68 C
69 C[[1]]
70 C[[2]]
71 C[1,2]
72 C$B[2]
73
74 C <- data.frame(A,B, stringsAsFactors=FALSE)
75 C
76 C[[1]]
77 C[[2]]
78 C[1,2]
79 C$B[2]
```

Data Handling: Vector

❖ 벡터의 선언

- C와는 달리 (파이썬이나 펄 등의 스크립트 언어와 마찬가지로) 따로 변수를 선언할 필요는 없음 → 생성과 동시에 값 할당
 - ✓ `a <- 3`: a라는 변수를 만들고 3의 값을 할당

❖ 벡터에 원소 추가

- R에서 벡터의 크기는 처음 만들어질 때 정해지므로 원소를 추가하거나 삭제하고 싶을 경우 새로이 벡터를 할당해야 함

```
> x <- c(1,2,3,4)
> x
[1] 1 2 3 4
> x <- c(x[1:3], 10, x[4])
> x
[1] 1 2 3 10 4
> |
```

❖ 벡터의 길이 파악

- `length()` 함수 이용

Data Handling: Vector

❖ 벡터의 재사용

- 두 벡터를 사용하는 연산을 할 때 각각의 길이가 다를 경우 짧은 쪽을 재사용하거나 반복 사용하여 긴 쪽에 맞춤

```
> c(1,2,4) + c(10,11,12,13,14)
[1] 11 13 16 14 16
Warning message:
In c(1, 2, 4) + c(10, 11, 12, 13, 14) :
  longer object length is not a multiple of shorter object length
> |
```

```
> x <- matrix(1:6, nrow=3, ncol=2)
> x
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
> x + c(1:2)
     [,1] [,2]
[1,]    2    6
[2,]    4    6
[3,]    4    8
> |
```

R 연산자

Arithmetic Operators

Operator	Description
+	addition
-	subtraction
*	multiplication
/	division
^ or **	exponentiation
x %% y	modulus (x mod y) 5%%2 is 1
x %/% y	integer division 5%/%2 is 2

Logical Operators

Operator	Description
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	exactly equal to
!=	not equal to
!x	Not x
x y	x OR y
x & y	x AND y
isTRUE(x)	test if X is TRUE

Data Handling: Vector

❖ 일반 벡터 연산

- 기본적으로 R에서 벡터 연산은 원소끼리 이루어짐

❖ 벡터 인덱싱

- 벡터의 특정 위치에 있는 원소들을 뽑아내 다른 벡터를 생성
- 인덱스는 중복 사용 가능
- 음수 인덱스는 해당 위치의 원소를 제거하는데 사용

```
Console ~/
> x <- c(1,2,3)
> y <- c(10,20,30)
> x+y
[1] 11 22 33
> x*y
[1] 10 40 90
> x%%y
[1] 1 2 3
> |
```

```
Console ~/
> y <- c(10,20,30,40,50)
> y[c(1,3)]
[1] 10 30
> y[2:3]
[1] 20 30
> v <- 2:3
> y[v]
[1] 20 30
> |
```

```
Console ~/
> y[c(1,2,1,3)]
[1] 10 20 10 30
> |
```

```
Console ~/
> y[-5]
[1] 10 20 30 40
> y[-length(y)]
[1] 10 20 30 40
> |
```


Data Handling: Vector

❖ 연산자로 벡터 생성하기

- : 연산자 – 일정 범위의 숫자로 이루어진 벡터 생성
- 연산자의 순서 주의
- seq 함수 - : 연산자의 일반화된 형태
- rep 함수 – 특정 값을 벡터 단위로 반복
 - ✓ each 옵션 – 특정 값을 원소 단위로 반복

Operator Syntax and Precedence

Description

Outlines R syntax and gives the precedence of operators.

Details

The following unary and binary operators are defined. They are listed in precedence groups, from highest to lowest.

:: :::	access variables in a namespace
\$ @	component / slot extraction
[[:	indexing
^	exponentiation (right to left)
- +	unary minus and plus
:	sequence operator
%any%	special operators (including %*% and %/%)
* /	multiply, divide
+ -	(binary) add, subtract
< > <= >= == !=	ordering and comparison
!	negation
& &&	and
	or
~	as in formulae
-> ->>	rightwards assignment
<- <<-	assignment (right to left)
=	assignment (right to left)
?	help (unary and binary)

Console ~/ ↻

```
> x <- 1:5
> y <- 5:1
> z <- 2
> 1:z-1
[1] 0 1
> 1:(z-1)
[1] 1
> |
```

Console ~/ ↻

```
> seq(from=12,to=30,by=3)
[1] 12 15 18 21 24 27 30
> seq(from=12,to=30,by=4)
[1] 12 16 20 24 28
> seq(from=1.1,to=2,length=10)
[1] 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2.0
> |
```

Console ~/ ↻

```
> rep(10,5)
[1] 10 10 10 10 10
> rep(c(10,20,30),3)
[1] 10 20 30 10 20 30 10 20 30
> rep(1:3,3)
[1] 1 2 3 1 2 3 1 2 3
> rep(c(10,20,30),each=3)
[1] 10 10 10 20 20 20 30 30 30
> |
```

Data Handling: Vector

❖ 벡터 원소에 각각 조건문 적용하기

- `any()` 함수: 벡터의 일부 인수가 TRUE이면 TRUE 반환
- `all()` 함수: 벡터의 모든 인수가 TRUE이면 TRUE 반환

❖ NA와 NULL

- NA (Not Available) – 물리적으로 존재하나 정확히 알 수 없는 값
- NULL – 물리적으로 존재하지 않는 값

Console ~/ ↻

```
> x <- 1:10
> x > 8
[1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE TRUE
> any(x > 8)
[1] TRUE
> any(x > 20)
[1] FALSE
> all(x > 8)
[1] FALSE
> all(x > 0)
[1] TRUE
> |
```



Console ~/ ↻

```
> x <- c(1,2,NA,4,5)
> y <- c(1,2,NULL,4,5)
> mean(x)
[1] NA
> mean(x, na.rm = TRUE)
[1] 3
> mean(y)
[1] 3
> |
```

Data Handling: Vector

❖ 필터링: 특정한 조건을 만족하는 원소들을 추출

- 인덱스에서 직접 추출
- `subset()` 함수 이용
- `which()` 함수: 해당 조건을 만족하는 원소들의 위치를 반환

```
Console ~/    
> x <- c(1,2,NA,4,5)  
> x[x>2]  
[1] NA 4 5  
> subset(x, x>2)  
[1] 4 5  
> x <- c(10,20,NA,40,50)  
> x[x>20]  
[1] NA 40 50  
> subset(x, x>20)  
[1] 40 50  
> which(x>20)  
[1] 4 5  
> |
```

Data Handling: Matrix & Array

❖ R 행렬의 특징

- 행과 열은 1부터 시작 (위쪽 끝 원소는 [1,1]로 표현)
- 내부 저장 공간은 "열 우선 배열(column-major order)"

❖ 행렬 생성자: matrix()

- 방식 1: 원소의 값을 모두 주고 행과 열의 수를 지정 (열 우선)
- 방식 2: 원소의 값을 모두 주고 행 지정 옵션 사용 (행 우선)
- 방식 3: 빈 행렬 생성 뒤 각 원소의 값을 입력

```
Console ~/
> A = matrix(1:15, nrow=5, ncol=3)
> A
```

	[,1]	[,2]	[,3]
[1,]	1	6	11
[2,]	2	7	12
[3,]	3	8	13
[4,]	4	9	14
[5,]	5	10	15

```
Console ~/
> B = matrix(1:15, nrow=5, byrow = T)
> B
```

	[,1]	[,2]	[,3]
[1,]	1	2	3
[2,]	4	5	6
[3,]	7	8	9
[4,]	10	11	12
[5,]	13	14	15

```
Console ~/
> C = matrix(nrow=2, ncol=2)
> C[1,1] = 1
> C[1,2] = 2
> C[2,1] = 3
> C[2,2] = 4
> C
```

	[,1]	[,2]
[1,]	1	2
[2,]	3	4

Data Handling: Matrix & Array

❖ 행렬의 연산

- 행렬의 선형대수 연산: 행렬간 곱, 행렬-상수 곱, 행렬 합 등
- 행렬 인덱싱
- 행렬 필터링

```

Console ~/
> A = matrix(1:4, nrow=2, ncol=2)
> B = matrix(seq(from=2,to=8,by=2), nrow=2, ncol=2)
> A
  [,1] [,2]
[1,]  1   3
[2,]  2   4
> B
  [,1] [,2]
[1,]  2   6
[2,]  4   8
> A*B # 행렬 원소간 곱셈
  [,1] [,2]
[1,]  2  18
[2,]  8  32
> A %% B # 행렬간 곱셈
  [,1] [,2]
[1,] 14  30
[2,] 20  44
> A*3 # 행렬*상수
  [,1] [,2]
[1,]  3   9
[2,]  6  12
> A+B # 행렬간 합
  [,1] [,2]
[1,]  3   9
[2,]  6  12
>

```

```

Console ~/
> C = matrix(1:15, nrow=5, ncol=3)
> C
  [,1] [,2] [,3]
[1,]  1   6  11
[2,]  2   7  12
[3,]  3   8  13
[4,]  4   9  14
[5,]  5  10  15
> C[3,2]
[1]  8
> C[2,]
[1]  2  7 12
> C[,3]
[1] 11 12 13 14 15
> C[2:4,2:3]
  [,1] [,2]
[1,]  7  12
[2,]  8  13
[3,]  9  14
> C[-1,]
  [,1] [,2] [,3]
[1,]  2   7  12
[2,]  3   8  13
[3,]  4   9  14
[4,]  5  10  15
> C[1,] <- c(10, 11, 12)
> C
  [,1] [,2] [,3]
[1,] 10  11  12
[2,]  2   7  12
[3,]  3   8  13
[4,]  4   9  14
[5,]  5  10  15
>

```


Data Handling: Matrix & Array

❖ 행렬의 행과 열에 함수 적용

- *apply() 함수: apply(), sapply(), tapply(), lapply() 등

```
apply(m, dimcode, f, fargs)
```

- m: 행렬
- dimcode: 차원수 (1:행적용, 2:열적용)
- f: 적용 함수
- fargs: f에 필요한 인수 집합 (선택사항)

```
Console ~/   
> A <- matrix(c(1:6), nrow=3, ncol=2)  
> apply(A,1,mean)  
[1] 2.5 3.5 4.5  
> apply(A,2,mean)  
[1] 2 5  
> |
```


Data Handling: Matrix & Array

❖ 행렬의 행과 열 추가/변경/제거

- `rbind()`, `cbind()` 함수 사용
- `rbind()`: 두 행렬을 열 기준으로 결합(상하결합)
- `cbind()`: 두 행렬을 행 기준으로 결합(좌우결합)

```

Console ~/
> A <- matrix(c(1:6), nrow=3, ncol=2)
> B <- matrix(c(11:16), nrow=3, ncol=2)
> A
      [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
> B
      [,1] [,2]
[1,]   11   14
[2,]   12   15
[3,]   13   16
> |

```

```

Console ~/
> rbind(A,B)
      [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
[4,]   11   14
[5,]   12   15
[6,]   13   16
> cbind(A,B)
      [,1] [,2] [,3] [,4]
[1,]    1    4   11   14
[2,]    2    5   12   15
[3,]    3    6   13   16
> cbind(A[,1],B[,2])
      [,1] [,2]
[1,]    1   14
[2,]    2   15
[3,]    3   16
> |

```

Data Handling: Matrix & Array

❖ 의도하지 않은 차원 축소 피하기

- drop 인수 사용하기
- as.matrix() 함수 사용하기

```

Console ~/
> A <- matrix(c(1:6), nrow=3, ncol=2)
> A
      [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
> B <- A[1,]
> B
[1] 1 4
> attributes(A)
$dim
[1] 3 2

> attributes(B)
NULL
> C <- A[1,,drop=FALSE]
> C
      [,1] [,2]
[1,]    1    4
> attributes(C)
$dim
[1] 1 2


> D <- as.matrix(A[1,])
> D
      [,1]
[1,]    1
[2,]    4
> attributes(D)
$dim
[1] 2 1

```

Data Handling: Matrix & Array

❖ 행렬의 행과 열에 이름 붙이기

- `colnames()`, `rownames()` 함수 사용

```
Console ~/ 
> A <- matrix(c(1:6), nrow=3, ncol=2)
> colnames(A)
NULL
> rownames(A)
NULL
> colnames(A) <- c("1st", "2nd")
> colnames(A)
[1] "1st" "2nd"
> rownames(A) <- c("First", "Second", "Third")
> rownames(A)
[1] "First" "Second" "Third"
> A[, "1st", drop=FALSE]
      1st
First   1
Second  2
Third   3
> |
```

Data Handling: Matrix & Array

❖ 고차원 배열

- `array()` 함수 사용

```

Console ~/
> A <- matrix(c(1:15), nrow=5, ncol=3)
> B <- matrix(c(11:25), nrow=5, ncol=3)
> A
      [,1] [,2] [,3]
[1,]    1    6   11
[2,]    2    7   12
[3,]    3    8   13
[4,]    4    9   14
[5,]    5   10   15
> B
      [,1] [,2] [,3]
[1,]   11   16   21
[2,]   12   17   22
[3,]   13   18   23
[4,]   14   19   24
[5,]   15   20   25
> C <- array(data=c(A,B),dim=c(3,2,2))
> C
, , 1
      [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6

, , 2
      [,1] [,2]
[1,]    7   10
[2,]    8   11
[3,]    9   12

```

Data Handling: List

❖ 리스트 생성하기

- list() 함수 또는 vector() 함수 사용
 - ✓ Tag를 이용하여 이름을 지정할 수 있음

```
Console ~/ ↵
> A <- list(name="Kang", salary = 10000, union = TRUE)
> A
$name
[1] "Kang"

$salary
[1] 10000

$union
[1] TRUE

> A$name
[1] "Kang"
> |
```

```
Console ~/ ↵
> B <- list("Kang", 10000, TRUE)
> B
[[1]]
[1] "Kang"

[[2]]
[1] 10000

[[3]]
[1] TRUE

> B[[1]]
[1] "Kang"
> |
```

```
Console ~/ ↵
> C <- vector(mode="list")
> C[["name"]] <- "Kang"
> C[["salary"]] <- 10000
> C[["union"]] <- TRUE
> C
$name
[1] "Kang"

$salary
[1] 10000

$union
[1] TRUE
```

Data Handling: List

❖ 리스트 연산

- 리스트 인덱싱
 - ✓ 리스트의 구성 요소는 세 가지 방법으로 접근 가능
 - ✓ \$이름, 대괄호 두 개(`[[]]`)
 - ✓ 대괄호를 하나만 사용할 경우, 결과값은 리스트로 반환됨
- 리스트에 원소 추가/삭제
 - ✓ 추가는 새로운 이름을 이용
 - ✓ 제거는 NULL 이용

```
Console ~/
> C$name
[1] "Kang"
> C[["name"]]
[1] "Kang"
> C[[1]]
[1] "Kang"
> |
```

```
Console ~/
> C1 <- C[[1]]
> class(C1)
[1] "character"
> C1
[1] "Kang"
> C2 <- C[1]
> class(C2)
[1] "list"
> C2
$name
[1] "Kang"
```

```
Console ~/
> C$office <- "frontier"
> C
$name
[1] "Kang"

$salary
[1] 10000

$union
[1] TRUE

$office
[1] "frontier"
```

```
Console ~/
> C$salary <- NULL
> C
$name
[1] "Kang"

$union
[1] TRUE

$office
[1] "frontier"
```

Data Handling: List

❖ 리스트 연산

■ 리스트 해제

- ✓ `unlist()` 함수 사용, 리스트 이름을 제거하고 싶을 경우 `use.names=FALSE` 옵션 추가

```

Console ~/
> tmp1ist <- list(a = list(1:5, c("a","b","c")), b = "Z", c = NA)
> tmp1ist
$a
$a[[1]]
[1] 1 2 3 4 5

$a[[2]]
[1] "a" "b" "c"

$b
[1] "Z"

$c
[1] NA


> unlist(tmp1ist)
 a1  a2  a3  a4  a5  a6  a7  a8  b  c
"1" "2" "3" "4" "5" "a" "b" "c" "Z" NA
> unlist(tmp1ist, use.names = FALSE)
[1] "1" "2" "3" "4" "5" "a" "b" "c" "Z" NA
> |

```


Data Handling: List

❖ 리스트에 함수적용

- `lapply()` 함수와 `sapply()` 함수 이용
 - ✓ 리스트의 각 요소에 적용하고 결과값을 리스트로 반환(`lapply`)하거나 벡터로 반환(`sapply`)

```
Console ~/   
> A <- list(1:3,25:29)  
> A  
[[1]]  
[1] 1 2 3  
  
[[2]]  
[1] 25 26 27 28 29  
  
> lapply(A,median)  
[[1]]  
[1] 2  
  
[[2]]  
[1] 27  
  
> sapply(A,median)  
[1] 2 27  
> |
```

Data Handling: Data Frame

❖ 데이터 프레임 생성 및 접근

- `data.frame()` 함수를 사용하여 데이터프레임 생성
- 데이터프레임에 접근하는 세 가지 방식
 - ✓ `DF$name`, `DF[[]]`, `DF[,]`

```

Console ~/
> kids <- c("Jack", "Jill")
> ages <- c(12,10)
> d <- data.frame(kids, ages, stringsAsFactors=FALSE)
> d
  kids ages
1 Jack   12
2 Jill   10
> |

```

```

Console ~/
> d[[1]]
[1] "Jack" "Jill"
> class(d[[1]])
[1] "character"
> d$kids
[1] "Jack" "Jill"
> class(d$kids)
[1] "character"
> d[,1]
[1] "Jack" "Jill"
> class(d[,1])
[1] "character"
> d[1]
  kids
1 Jack
2 Jill
> class(d[1])
[1] "data.frame"
> |

```

Data Handling: Data Frame

❖ 부분 데이터 프레임 추출 및 필터링

- Matrix 사용법과 동일

❖ 데이터 프레임 결합

- `rbind()`: 열을 기준으로 상하결합
- `cbind()`: 행을 기준으로 좌우결합

```

Console ~/
> Exam
  Exam1 Exam2 Quiz
1  2.0   3.3  4.0
2  3.3   2.0  3.7
3  4.0   4.0  4.0
4  2.3   0.0  3.3
5  2.3   1.0  3.3
6  3.3   3.7  4.0
> Exam[2:5,]
  Exam1 Exam2 Quiz
2  3.3   2.0  3.7
3  4.0   4.0  4.0
4  2.3   0.0  3.3
5  2.3   1.0  3.3
> Exam[2:5,2]
[1] 2 4 0 1
> Exam[2:5,2, drop=FALSE]
  Exam2
2      2
3      4
4      0
5      1
>

```

```

Console ~/
> Exam[Exam$Exam1 > 3,]
  Exam1 Exam2 Quiz
2  3.3   2.0  3.7
3  4.0   4.0  4.0
6  3.3   3.7  4.0
> rbind(d,list("Laura",19))
  kids ages
1 Jack  12
2 Jill  10
3 Laura 19
>

```

Data Handling: Data Frame

❖ 데이터 프레임 결합: Merge

- 여러 개의 관계형 데이터베이스를 하나의 테이블로 결합하여 사용할 경우
- Inner/Outer/Left/Right join 모두 가능

```
> merge(dFA, dfB) # default: inner join
  kids ages state
1  Jill   10    NY
2 Laura   19    CA
> merge(dFA, dfB, all = TRUE) # outer join
  kids ages state
1 Alice   NA    MA
2 Jack    12   <NA>
3 Jill    10    NY
4 Laura   19    CA
> merge(dFA, dfB, all.x = TRUE) # left join
  kids ages state
1 Jack    12   <NA>
2 Jill    10    NY
3 Laura   19    CA
> merge(dFA, dfB, all.y = TRUE) # right join
  kids ages state
1 Alice   NA    MA
2 Jill    10    NY
3 Laura   19    CA
```

Data Handling: Data Frame

❖ 데이터 프레임 결합: Merge

- 결합 기준이 되는 변수 이름이 다를 경우, 해당 변수명을 각각 지정

```

316 firstname <- c("Alice", "Jill", "Laura")
317 state <- c("MA", "NY", "CA")
318 dfC <- data.frame(firstname, state, stringsAsFactors=FALSE)
319 dfC
320
321 merge(dfA, dfC, by.x="kids", by.y="firstname")

```

```

> dfC <- data.frame(firstname, state, stringsAsFactors=FALSE)
> dfC
  firstname state
1    Alice    MA
2     Jill    NY
3    Laura    CA
> merge(dfA, dfC, by.x="kids", by.y="firstname")
  kids ages state
1  Jill   10    NY
2 Laura   19    CA

```

Data Handling: Factor

❖ 팩터(Factor)

- R에서의 팩터는 벡터에 추가정보가 더해진 것임
 - ✓ 추가 정보는 벡터의 값 가운데 겹치지 않는 값의 기록으로 이루어져 있으며 이를 레벨(level)이라고 함
 - ✓ 팩터의 길이는 레벨의 수가 아닌 데이터의 길이로 결정됨
 - ✓ 새 레벨을 추가 가능
 - ✓ 기존에 존재하지 않는 레벨을 추가할 경우 NA 처리

```

Console ~/
> x <- c(5,12,13,12)
> xf <- factor(x)
> xf
[1] 5 12 13 12
Levels: 5 12 13
> str(xf)
Factor w/ 3 levels "5","12","13": 1 2 3 2
> unclass(xf)
[1] 1 2 3 2
attr(,"levels")
[1] "5" "12" "13"
> length(xf)
[1] 4
>

```

```

Console ~/
> xff <- factor(x, levels=c(5,12,13,88))
> xff
[1] 5 12 13 12
Levels: 5 12 13 88
> xff[2] <- 88
> xff
[1] 5 88 13 12
Levels: 5 12 13 88
> xff[2] <- 20
warning message:
In `[<-,factor`(`*tmp*`, 2, value = 20) :
  invalid factor level, NA generated
> xff
[1] 5 <NA> 13 12
Levels: 5 12 13 88
>

```

Data Handling: Factor

❖ 팩터에 함수 적용하기

- `tapply()` 함수
 - ✓ 카테고리 변수에 의한 빈도표 등을 이용할 때 유용
 - ✓ 두 개 이상의 팩터에 대해서도 적용 가능

```

Console ~/
> ages <- c(25,26,55,37,21,42)
> affils <- c("R","D","D","R","U","D")
> tapply(ages, affils, mean)
  D  R  U
41 31 21
> |

```

```

Console ~/
> gender <- c("M", "M", "F", "M", "F", "F")
> age <- c(47, 59, 21, 32, 33, 24)
> income <- c(55000, 88000, 32450, 76500, 123000, 45650)
> tmp <- data.frame(gender, age, income)
> tmp$over25 <- ifelse(tmp$age>25,1,0)
> tmp
  gender age income over25
1      M  47  55000       1
2      M  59  88000       1
3      F  21  32450       0
4      M  32  76500       1
5      F  33 123000       1
6      F  24  45650       0
> tapply(tmp$income, list(tmp$gender, tmp$over25), mean)
      0      1
F 39050 123000.00
M      NA  73166.67
> |

```


Data Handling: Factor

❖ 팩터에 함수 적용하기

- `split()` 함수
 - ✓ 그룹을 만드는데 사용
 - ✓ 두 개 이상의 팩터에 대해서도 적용 가능


```
Console ~/   
> split(tmp$income, list(tmp$gender, tmp$over25))  
$F.0  
[1] 32450 45650  
  
$M.0  
numeric(0)  
  
$F.1  
[1] 123000  
  
$M.1  
[1] 55000 88000 76500
```

Table of Contents

I	R Introduction
II	형태에 따른 데이터 처리
III	텍스트 데이터 처리
IV	조건문과 반복문
IV	함수

Text Processing

❖ 문자열의 길이 알아내기

- `nchar()` 함수 사용 (not `length()`)
 - ✓ 공백(space) 및 특수문자도 하나의 문자로 취급

❖ 문자열 연결하기

- `paste()` 함수 사용
 - ✓ 기본적으로 문자열 쌍들 사이에 빈칸 삽입, 옵션을 통해 조절 가능
 - ✓ 문자가 아닌 인자도 문자로 변환 시도, 벡터의 경우 모든 조합 생성

```

Console ~/
> s <- "welcome to Data Science!"
> length(s)
[1] 1
> nchar(s)
[1] 24
> s1 <- "My name is"
> s2 <- "Pilsung Kang"
> paste(s1, s2)
[1] "My name is Pilsung Kang"
> paste(s1, s2, sep="-")
[1] "My name is-Pilsung Kang"
> paste(s1, s2, sep="")
[1] "My name isPilsung Kang"
>

```

```


Console ~/
> paste("The value of log10 is", log(10))
[1] "The value of log10 is 2.30258509299405"
> s1 <- c("My name is", "Your name is")
> s2 <- c("Pilsung")
> s3 <- c("Pilsung", "Younho", "Hakyeon")
> paste(s1, s2)
[1] "My name is Pilsung" "Your name is Pilsung"
> paste(s1, s3)
[1] "My name is Pilsung" "Your name is Younho" "My name is Hakyeon"
> stooges <- c("Dongmin", "Sangkyum", "Junhong")
> paste(stooges, "loves", "R.")
[1] "Dongmin loves R." "Sangkyum loves R." "Junhong loves R."
> paste(stooges, "loves", "R", collapse = ", and ")
[1] "Dongmin loves R, and Sangkyum loves R, and Junhong loves R"
>

```

Text Processing

❖ 하위 문자열 추출하기

- substring(string, start, end) 사용
 - ✓ Start에서 시작하고 end에서 끝나는 하위 문자열 추출
 - ✓ string 인자가 문자열 벡터인 경우, 모든 문자열에 적용한 뒤 하위 문자열로 벡터를 반환
 - ✓ 모든 인자가 벡터일 경우, 서로 대응하는 벡터로 취급

```
Console ~/   
> substr("Data science", 1, 4)  
[1] "Data"  
> substr("Data science", 6, 10)  
[1] "scien"  
> stooges <- c("Dongmin", "Sangkyum", "Junhong")  
> substr(stooges, 1, 3)  
[1] "Don" "San" "Jun"  
> cities <- c("New York, NY", "Los Angeles, CA", "Peoria, IL")  
> substr(cities, nchar(cities)-1, nchar(cities))  
[1] "NY" "CA" "IL"  
> |
```

Text Processing

❖ 구분자로 문자열 분할하기

- `strsplit`(문자열, 구분자) 함수 사용
 - ✓ "구분자"는 간단한 문자열이나 정규 표현식으로 사용
 - ✓ 예시 1: 파일 경로를 슬래시(/)로 구분하기

```

Console ~/
> path <- "c:/home/mike/data/trials.csv"
> strsplit(path, "/")
[[1]]
[1] "c:"      "home"    "mike"    "data"    "trials.csv"

```

- ✓ 문자열이 "문자열의 벡터"일 경우 각 문자열에 적용 후 리스트 반환

```

Console ~/
> path <- c("c:/home/mike/data/trials.csv",
+ "c:/home/mike/data/errors.txt",
+ "c:/home/mike/data/report.doc")
> strsplit(path, "/")
[[1]]
[1] "c:"      "home"    "mike"    "data"    "trials.csv"

[[2]]
[1] "c:"      "home"    "mike"    "data"    "errors.txt"

[[3]]
[1] "c:"      "home"    "mike"    "data"    "report.doc"

```

Text Processing

❖ 정규 표현식

- 문자열 관련 다양한 클래스를 축약하여 정의한 것
 - ✓ 특정 순서의 문자열/특정 문자들의 부분집합/미리 정의된 문자열 집합 등을 표현
 - ✓ 실제 사람이 눈으로 보는 문자열과 컴퓨터에 기록된 문자열이 다를 경우 정확한 추출을 위해 사용

```

Console ~/
> strsplit(path, "om")
[[1]]
[1] "C:/h" "e/mike/data/trials1.csv"

[[2]]
[1] "C:/h" "e/mike/data/errors2.txt"

[[3]]
[1] "C:/h" "e/mike/data/report3.doc"

> strsplit(path, "[hm]")
[[1]]
[1] "C:/" "o" "e/" "ike/data/trials1.csv"

[[2]]
[1] "C:/" "o" "e/" "ike/data/errors2.txt"

[[3]]
[1] "C:/" "o" "e/" "ike/data/report3.doc"

> strsplit(path, "i.e")
[[1]]
[1] "C:/home/m" "/data/trials1.csv"

[[2]]
[1] "C:/home/m" "/data/errors2.txt"

[[3]]
[1] "C:/home/m" "/data/report3.doc"

```

```

Console ~/
> strsplit(path, "\\.")
[[1]]
[1] "C:/home/mike/data/trials1" "csv"

[[2]]
[1] "C:/home/mike/data/errors2" "txt"

[[3]]
[1] "C:/home/mike/data/report3" "doc"

> strsplit(path, "r{2}")
[[1]]
[1] "C:/home/mike/data/trials1.csv"

[[2]]
[1] "C:/home/mike/data/e" "ors2.txt"

[[3]]
[1] "C:/home/mike/data/report3.doc"

> strsplit(path, "[[:digit:]]")
[[1]]
[1] "C:/home/mike/data/trials" ".csv"

[[2]]
[1] "C:/home/mike/data/errors" ".txt"

[[3]]
[1] "C:/home/mike/data/report" ".doc"

```

Text Processing

❖ 하위 문자열 대체하기

- `sub(old, new, string)` 또는 `gsub(old, new, string)` 함수 사용
 - ✓ `sub()`는 첫 번째 하위문자열만 대체, `gsub`는 모든 문자열 대체

```

Console ~/
> tmpstring <- "kim is stupid and kang is stupid too"
> sub("stupid", "smart", tmpstring)
[1] "kim is smart and kang is stupid too"
> gsub("stupid", "smart", tmpstring)
[1] "kim is smart and kang is smart too"

```

❖ 문자열에서 주어진 패턴 일치 여부 확인하기

- `grep(pattern, x)` 함수 사용
 - ✓ `pattern`이 있는 `x`의 인덱스를 반환

```

Console ~/
> grep("mike", path)
[1] 1 2 3
> grep("errors", path)
[1] 2

```


Table of Contents

I	R에서 사용되는 데이터의 형태 (Data Type)
II	형태에 따른 데이터 처리
III	텍스트 데이터 처리
IV	조건문과 반복문
IV	함수

조건문과 반복문

❖ if-else 조건문

```
if (condition) {
  statement 1
} else {
  statement 2
}
```

- condition: 조건 (단순 논리값부터 복잡한 함수까지 모두 가능)
- statement 1: 조건을 만족할 경우 실행
- statement 2: 조건을 만족하지 않을 경우 실행

Console ~/ ↗	Console ~/ ↗	Console ~/ ↗
<pre>> r <- 1 > if (r==4) { + printf("The value of r is 4") + } else { + print("The value of r is not 4") + } [1] "The value of r is not 4"</pre>	<pre>> carbon <- c(10, 12, 15, 19, 20) > if (mean(carbon) > median(carbon)) { + print ("Mean > Median") + } else { + print ("Median <= Mean") + } [1] "Mean > Median"</pre>	<pre>> # Caution! > if (mean(carbon) > median(carbon)) { + print ("Mean > Median") + } [1] "Mean > Median" > else { Error: unexpected 'else' in "else" > print ("Median <= Mean") [1] "Median <= Mean" > } Error: unexpected '}' in "}"</pre>

조건문과 반복문

❖ ifelse 조건문

■ 벡터화된 조건문

```
ifelse (condition, result1, result2)
```

- condition: Boolean vector (단순 논리값부터 복잡한 함수까지 모두 가능)
- result 1: condition이 TRUE이면 실행
- statement 2: condition이 False이면 실행

Console ~/ ↗

```
> x <- 1:10
> y <- ifelse(x%%2 == 0, "even", "odd")
> y
[1] "odd" "even" "odd" "even" "odd" "even" "odd" "even" "odd" "even"
```

조건문과 반복문

❖ for 를 이용한 반복문

```
for (i in x) {
  statement
}
```

- i: 반복문의 index
- x: 반복문의 index가 실행되는 구성요소
- statement: 반복문에서 실행하는 명령

```
Console ~/ ↩
> # 반복문
> n <- c(5,10,15)
> for (i in n) {
+   print(i^2)
+ }
[1] 25
[1] 100
[1] 225
```

❖ while/repeat/break를 이용한 반복문

```
while (condition) {
  statement
}
```

- condition의 조건을 만족하지 않을 때까지 statement를 실행

```
Console ~/ ↩
> i <- 1
> while (i <= 10) {
+   i <- i+4
+   print(i)
+ }
[1] 5
[1] 9
[1] 13
```

```
repeat {
  statement
  condition break
}
```

- statement를 반복실행, condition 만족 시 중단

```
Console ~/ ↩
> i <- 1
> repeat {
+   i <- i+4
+   print(i)
+   if (i > 10) break
+ }
[1] 5
[1] 9
[1] 13
```

Table of Contents

I	R에서 사용되는 데이터의 형태 (Data Type)
II	형태에 따른 데이터 처리
III	텍스트 데이터 처리
IV	조건문과 반복문
IV	함수

함수: 개요

❖ 왜 함수를 사용하는가?

- An incidental advantage of putting code into functions is that **the workspace is not then cluttered with objects that are local to the function.**

```
all()          # returns TRUE if all values are TRUE
any()          # returns TRUE if any values are TRUE
args()         # information on the arguments to a function
cat()          # prints multiple objects, one after the other
cumprod()      # cumulative product
cumsum()       # cumulative sum
diff()         # form vector of first differences
               # N. B. diff(x) has one less element than x
history()      # displays previous commands used
is.factor()    # returns TRUE if the argument is a factor
is.na()        # returns TRUE if the argument is an NA
               # NB also is.logical(), is.matrix(), etc.
length()       # number of elements in a vector or of a list
ls()           # list names of objects in the workspace
```

함수: 개요

❖ 왜 함수를 사용하는가? (cont')

- An incidental advantage of putting code into functions is that **the workspace is not then cluttered with objects that are local to the function.**

```
mean()      # mean of the elements of a vector
median()    # median of the elements of a vector
order()     # x[order(x)] sorts x (by default, NAs are last)
print()     # prints a single R object
range()     # minimum and maximum value elements of vector
sort()      # sort elements into order, by default omitting NAs
rev()       # reverse the order of vector elements
str()       # information on an R object
unique()    # form the vector of distinct values
which()     # locates 'TRUE' indices of logical vectors
which.max() # locates (first) maximum of a numeric vector
which.min() # locates (first) minimum of a numeric vector
with()      # do computation using columns of specified data frame
```


함수: 구문

❖ 함수 작성하기

```
function_name <- function(arguments) {
  statement 1;
  statement 2;
  ...
  return(object)
}
```

- `function_name`: 해당 함수의 호출 시 사용하는 이름
- `arguments`: 함수 실행을 위해 받아들이는 인자
- `statement 1, 2, ...`: 함수 내부에서 실행하는 연산
- `object`: 함수 실행을 통해 반환되는 값 (`return`을 명시하지 않을 경우 가장 마지막 값을 반환)

Console ~/ ↗

```
> # User written functions
> mean.and.sd1 <- function(x) {
+   av <- mean(x)
+   sdev <- sd(x)
+   c(mean=av, SD=sdev)
+ }
> distance <- c(148, 182, 173, 166, 109, 141, 166)
> mean.and.sd1(distance)
      mean      SD
155.00000 24.68468
```

Console ~/ ↗

```
> mean.and.sd2 <- function(x) {
+   av <- mean(x)
+   sdev <- sd(x)
+   c(mean=av, SD=sdev)
+   return(av)
+ }
> distance <- c(148, 182, 173, 166, 109, 141, 166)
> mean.and.sd2(distance)
[1] 155
```

함수: 구문

❖ 함수 작성하기

- Input arguments에는 다양한 형태의 데이터가 사용될 수 있음
- Default argument를 정의하는 것도 가능
 - ✓ Default argument는 함수를 호출하면서 argument를 지정하지 않을 경우 실행됨
 - ✓ function_name() 의 형태

```

Console ~/
> mean.and.sd3 <- function(x = rnorm(10)) {
+   av <- mean(x)
+   sdev <- sd(x)
+   c(mean=av, SD=sdev)
+ }
> mean.and.sd3()
      mean      SD
-0.5742746  1.4007627
> mean.and.sd3(distance)
      mean      SD
155.00000  24.68468
  
```

함수: 구문


❖ 함수의 인수 (arguments)

- 함수의 인수는 각각의 이름을 가짐
- 함수 안에서 각 인수에 접근할 때는 이름으로 접근
- 세 가지 방식으로 인수 지정 가능 (우선순위 순)

✓ **Exact name**

✓ Partially matching names

✓ Argument order

```
Console ~/   
> # Function arguments  
> addTheLog <- function(first, second) {first + log(second)}  
> # Exact names  
> addTheLog(second=exp(4),first=1)  
[1] 5  
> # Partially matching names  
> addTheLog(s=exp(4),first=1)  
[1] 5  
> # Argument order  
> addTheLog(1,exp(4))  
[1] 5
```

함수: 구문

❖ 함수의 반환값

- 모든 R객체가 함수의 반환값이 될 수 있음
 - ✓ return() 함수 사용
 - ✓ return() 함수를 사용하지 않을 경우, 가장 마지막에 수행된 값이 반환됨

Console ~/ ↗

```
> oddcount <- function(x) {
+ k <- 0
+ print(sprintf("odd number calculator"))
+ for (n in 1:x) {
+ if (n %% 2 == 1) {
+ cat(sprintf("%d is an odd number. \n", n))
+ k <- k+1
+ }
+ }
+ return(k)
+ }
> oddcount(10)
[1] "odd number calculator"
1 is an odd number.
3 is an odd number.
5 is an odd number.
7 is an odd number.
9 is an odd number.
[1] 5
```

Console ~/ ↗

```
> oddcount <- function(x) {
+ k <- 0
+ print(sprintf("odd number calculator"))
+ for (n in 1:x) {
+ if (n %% 2 == 1) {
+ cat(sprintf("%d is an odd number. \n", n))
+ k <- k+1
+ }
+ }
+ k
+ }
> oddcount(10)
[1] "odd number calculator"
1 is an odd number.
3 is an odd number.
5 is an odd number.
7 is an odd number.
9 is an odd number.
[1] 5
```

함수: 구문

❖ 함수의 반환값

- 모든 R객체가 함수의 반환값이 될 수 있음
 - ✓ return() 함수 사용
 - ✓ return() 함수를 사용하지 않을 경우, 가장 마지막에 수행된 값이 반환됨

```

Console ~/
> # Return the result without either return() or explicit designation
> oddcount <- function(x) {
+ k <- 0
+ print(sprintf("odd number calculator"))
+ for (n in 1:x) {
+ if (n %% 2 == 1) {
+ cat(sprintf("%d is an odd number. \n", n))
+ k <- k+1
+ }
+ }
+ }
> oddcount(10)
[1] "odd number calculator"
1 is an odd number.
3 is an odd number.
5 is an odd number.
7 is an odd number.
9 is an odd number.

```

← for 문의 마지막 실행에서 NULL을 반환

함수: 특징

❖ 함수는 객체다!

- R함수는 다른 객체가 사용되는 방식으로 동일하게 사용된다
 - ✓ 객체의 화면 출력 (일부 C로 직접 작성된 내장함수는 불가능), arguments 확인
 - ✓ 다른 함수의 인수로 활용

```


Console ~/ ↗
> abline
function (a = NULL, b = NULL, h = NULL, v = NULL, reg = NULL,
  coef = NULL, untf = FALSE, ...)
{
  int_abline <- function(a, b, h, v, untf, col = par("col"),
    lty = par("lty"), lwd = par("lwd"), ...) .External.graphics(C_abline,
    a, b, h, v, untf, col, lty, lwd, ...)
  if (!is.null(reg)) {
    if (!is.null(a))
      warning("'a' is overridden by 'reg'")
    a <- reg
  }
  if (is.object(a) || is.list(a)) {
    p <- length(coefa <- as.vector(coef(a)))
    if (p > 2)
      warning(gettextf("only using the first two of %d regression coefficients",
        p), domain = NA)
    islm <- inherits(a, "lm")
    noInt <- if (isl)
      !as.logical(attr(stats::terms(a), "intercept"))
    else p == 1
    if (noInt) {
      a <- 0
      b <- coefa[1L]
    }
  }
}

```

함수: 특징

❖ 함수는 객체다!

- R함수는 다른 객체가 사용되는 방식으로 동일하게 사용된다
 - ✓ 객체의 화면 출력 (일부 C로 직접 작성된 내장함수는 불가능), arguments 확인
 - ✓ 다른 함수의 인수로 활용

```
Console ~/   
> args(abline)  
function (a = NULL, b = NULL, h = NULL, v = NULL, reg = NULL,  
          coef = NULL, untf = FALSE, ...)  
NULL  
.
```

함수: 특징

❖ 함수는 객체다!

- R함수는 다른 객체가 사용되는 방식으로 동일하게 사용된다
 - ✓ 객체의 화면 출력 (일부 C로 직접 작성된 내장함수는 불가능)
 - ✓ 다른 함수의 인수로 활용

```

Console ~/
> sort
function (x, decreasing = FALSE, ...)
{
  if (!is.logical(decreasing) || length(decreasing) != 1L)
    stop("'decreasing' must be a length-1 logical vector.\nDid you intend to set 'partial'?")
  useMethod("sort")
}
<bytecode: 0x09b6e014>
<environment: namespace:base>

```

```

Console ~/
> f1 <- function(a,b) return(a+b)
> f2 <- function(a,b) return(a-b)
> g <- function(h, x, y) h(x,y)
> g(f1,5,2)
[1] 7
> g(f2,5,2)
[1] 3

```


함수: 특징

❖ Anonymous Function

- R에서는 이름이 없는 함수를 생성하는 것도 가능함 (anonymous function)
- Anonymous function은 일반적으로 arguments의 형태로 다른 함수에 전달됨

```

Console ~/
> apply.to.three <- function(f) {f(3)}
> apply.to.three(function(x) {x*7})
[1] 21
> a <- c(1,2,3,4,5)
> sapply(a,function(x) {x+1})
[1] 2 3 4 5 6

```

- ✓ R이 `apply.to.three(function(x) {x*7})`을 interpret할 때, `function(x) {x*7}`이라는 anonymous function을 argument로 넘김
- ✓ 이후 `f(3)`이 evaluate되면서 `3*7`을 반환함
- R 환경에서 함수를 보다 유연하게 사용할 수 있도록 도와줌

함수: 특징

❖ 함수의 지역변수

- 함수는 지역변수가 아닌 변수를 바꾸지 않음
- 함수의 코드 내에서 지역변수가 아닌 변수들을 읽어올 수는 있지만 이 변수들에게 값을 쓸 수 있는 권한은 없음

```

Console ~/
> w <- 12
> f <- function (y) {
+ d <- 8
+ w <- w+1
+ y <- y-2
+ print(w)
+ h <- function() return(d*(w+y))
+ return(h())
+ }
> t <- 4
> f(t)
[1] 13
[1] 120
> w
[1] 12
> t
[1] 4

```

Note) 지역변수 w는 값이 바뀌기 전까지는 광역변수와 실제로 같은 메모리 위치를 참조하며, 값이 바뀌는 경우 새 메모리 위치가 할당됨

함수 예제 1: 단일 함수

❖ 예제 1: 1이 연달아 나오는 부분 찾기

- Q: 1과 0으로 이루어진 벡터에서 1이 k번 반복되어 나타나는 모든 부분을 찾아 그 첫번째 인덱스 셋을 반환

```

Console ~/
> # Function example 1
> findrepeats <- function(x, k) {
+   n <- length(x)
+   repeats <- NULL
+   for (i in 1:(n-k+1)) {
+     if(all(x[i:(i+k-1)] == 1)) repeats <- c(repeats, i)
+   }
+   return(repeats)
+ }
>
> vec <- c(0,1,1,0,0,1,1,1,0,1,1)
> findrepeats(vec,2)
[1] 2 6 7 10
> findrepeats(vec,3)
[1] 6
> findrepeats(vec,4)
NULL

```

함수 예제 2: 두 개 이상의 함수 연결

❖ 예제 2: Kendall's tau 계산하기

- Raw Data: 매 시간별 측정된 기온과 기압

Time	10:00	11:00	12:00	13:00	14:00
Temperature	10	15	13	17	20
Pressure	900	920	890	940	920

- To do
 - ✓ 각 측정 지표가 매시간 증가(+)하는지 감소(-)하는지 판단
 - ✓ 전체 측정 기간 중에서 두 측정 지표가
동시에 증가하거나 동시에 감소하는 횟수를 반환

```

Console ~/
> # Example 2: Kendall's tau
> findud <- function(v) {
+   vud <- v[-1] - v[-length(v)]
+   return(ifelse(vud > 0, 1, -1))
+ }
>
> udcorr <- function(x,y) {
+   ud <- lapply(list(x,y), findud)
+   return(mean(ud[[1]] == ud[[2]]))
+ }
>
> temp <- c(10, 15, 13, 17, 20)
> pressure <- c(900, 920, 890, 940, 920)
>
> udcorr(temp, pressure)
[1] 0.75
  
```

Q & A

