Lecture 1: Introduction to R (Part I)

Pilsung Kang

School of Industrial Management Engineering

Korea University

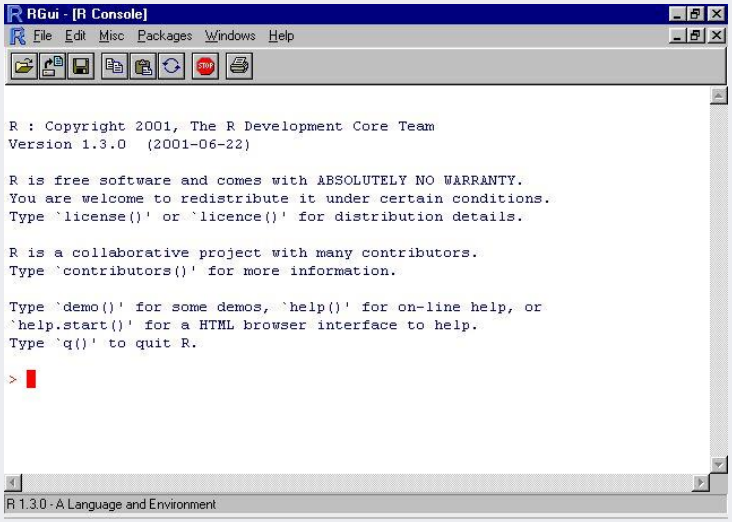
AGENDA

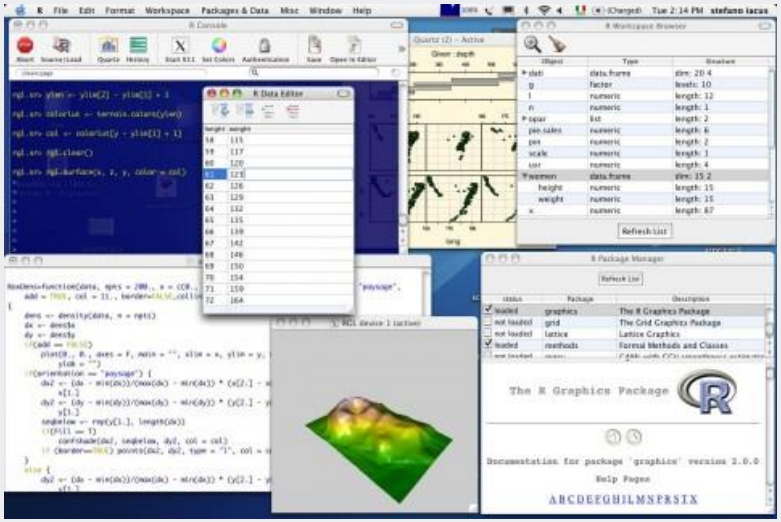
01 R Overview

02 Handling Different Data Types

03 Text Processing

History of R





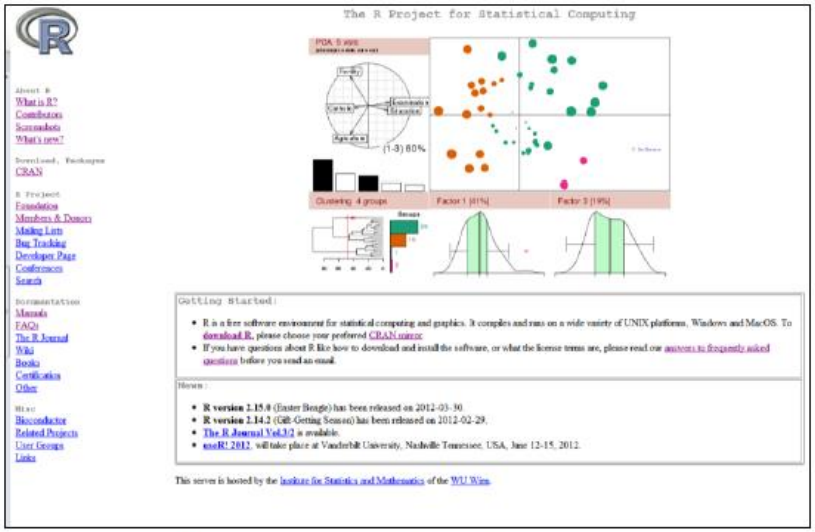
- R is an open source programming language and software environment for statistical computing and graphics
  - ✓ Created by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand, and is currently developed by the R Development Core Team
  - ✓ Highly extensible through the use of user-submitted packages for specific functions or specific areas of study

The R Language

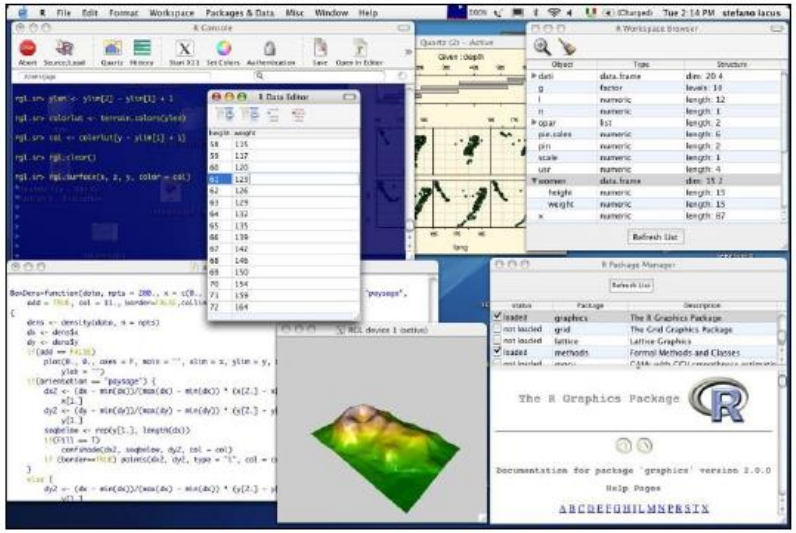
- R is an expression-based language
  - ✓ User type language expressions at the R prompt
  - ✓ These expressions are evaluated by the R interpreter
  - ✓ The computed values of the expressions are printed
- R is extensile
  - ✓ Users can implement new functionality in the form of functions
  - ✓ Developers can implement new packages of functionality that extends the base system

Why R?

- Provides a wide variety of statistical techniques
- Free software under GNU public license, highly extensible
- Independent to OS: Windows, Linux, MacOS X, etc.



〈 Homepage of R-project 〉



〈 R screenshot: Visualization 〉

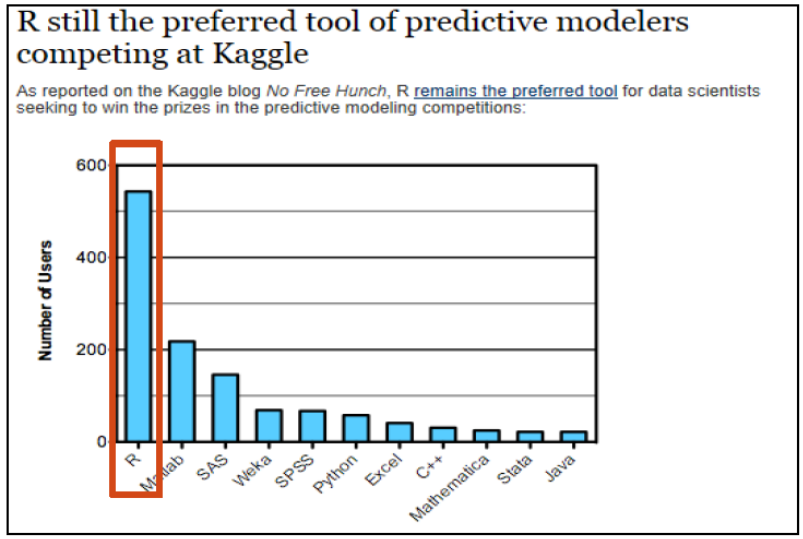
Why R?

- Data analysis with R: new trend!
  - ✓ Many researchers now use R for their statistical analysis
  - ✓ Kdnuggets & Kaggle poll results

The previous KDnuggets Poll asked **What programming languages you used for data mining / data analysis in the past 12 months?**

Here are the results, based on 570 voters:

R (257)	45%
SQL (184)	32%
Python (140)	25%
Java (139)	24%
SAS (121)	21%
MATLAB (83)	15%
C/C++ (73)	13%
Unix shell/awk/gawk/sed (59)	10%
Perl (45)	7.9%
Hadoop/Pig/Hive (35)	6.1%
Lisp (4)	0.7%
Other (70)	12.0%
None (7)	1.2%



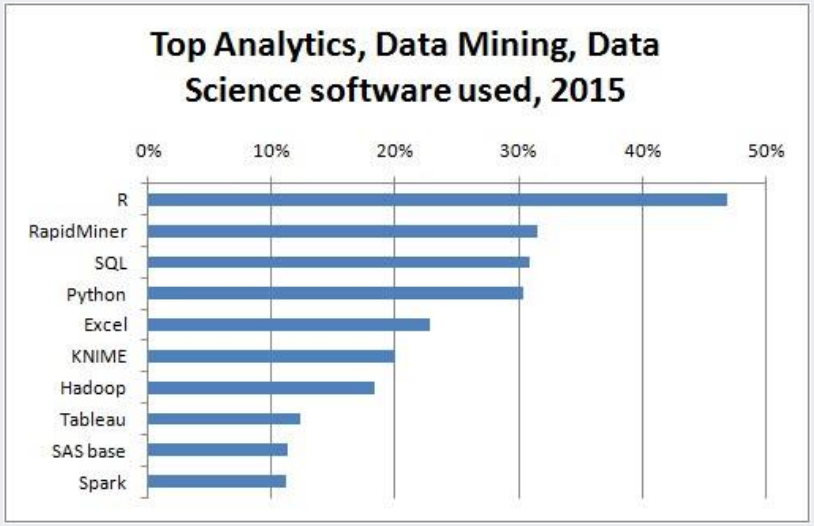
〈 R: the most widely used statistical tool 〉

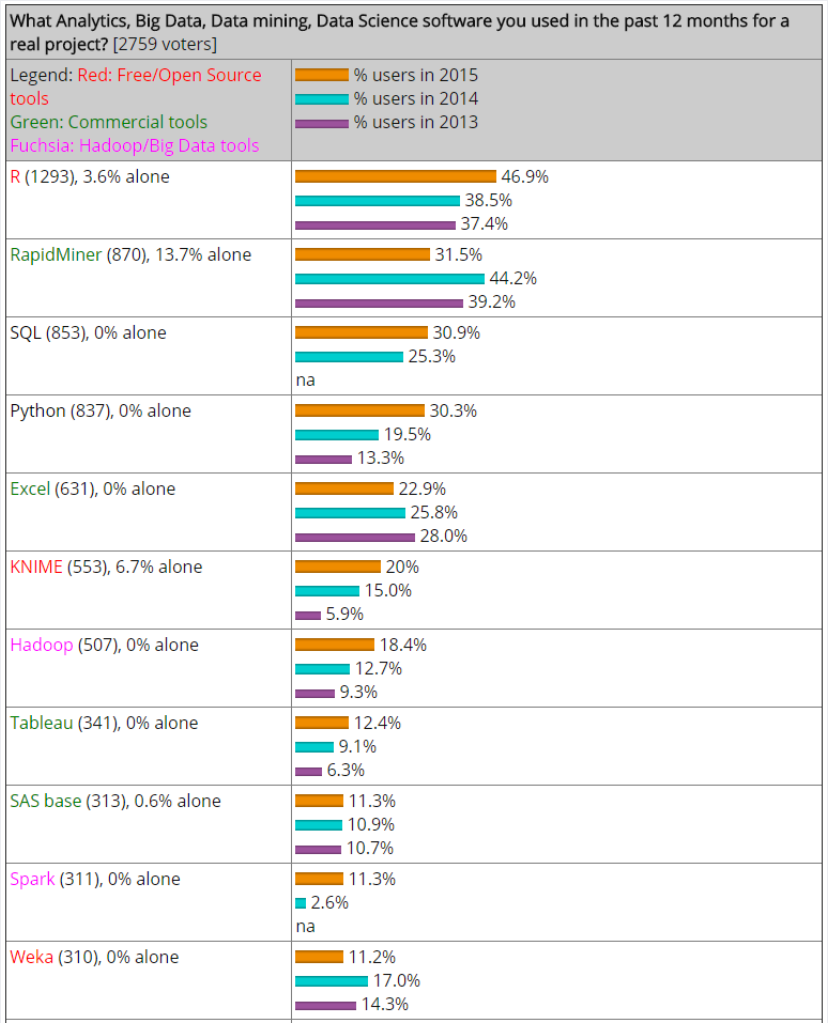
- Collaboration among users: the reason why R is loved by a number of users

useR! The R Journal

Why R?

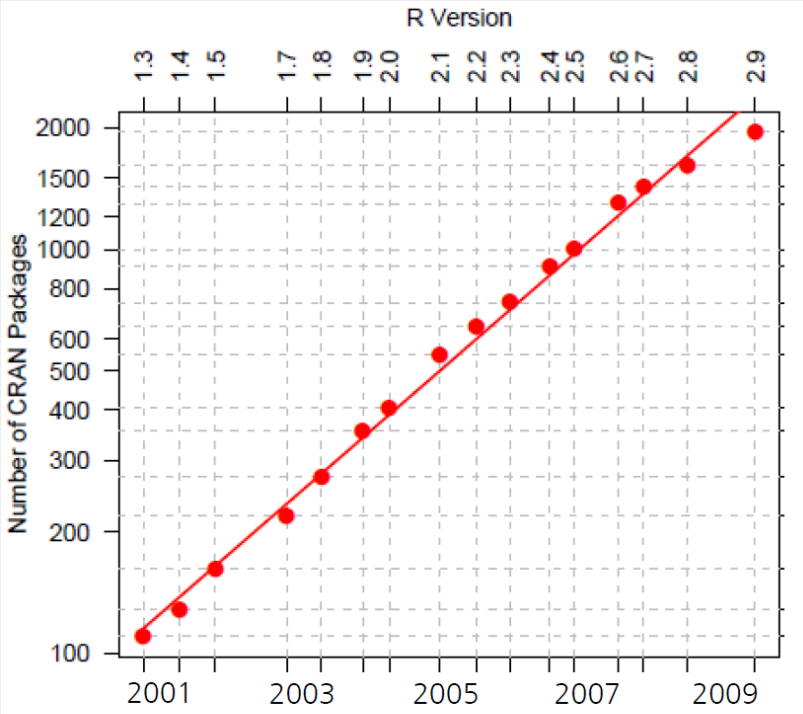
- Data analysis with R: new trend!
  - ✓ Many researchers now use R for their statistical analysis





Why R?

- R packages
  - ✓ Everyone can make, distribute, and use R packages
  - ✓ Almost every statistical analysis can be possible
  - ✓ 10,136 packages are available (2017. 02. 22)



〈 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

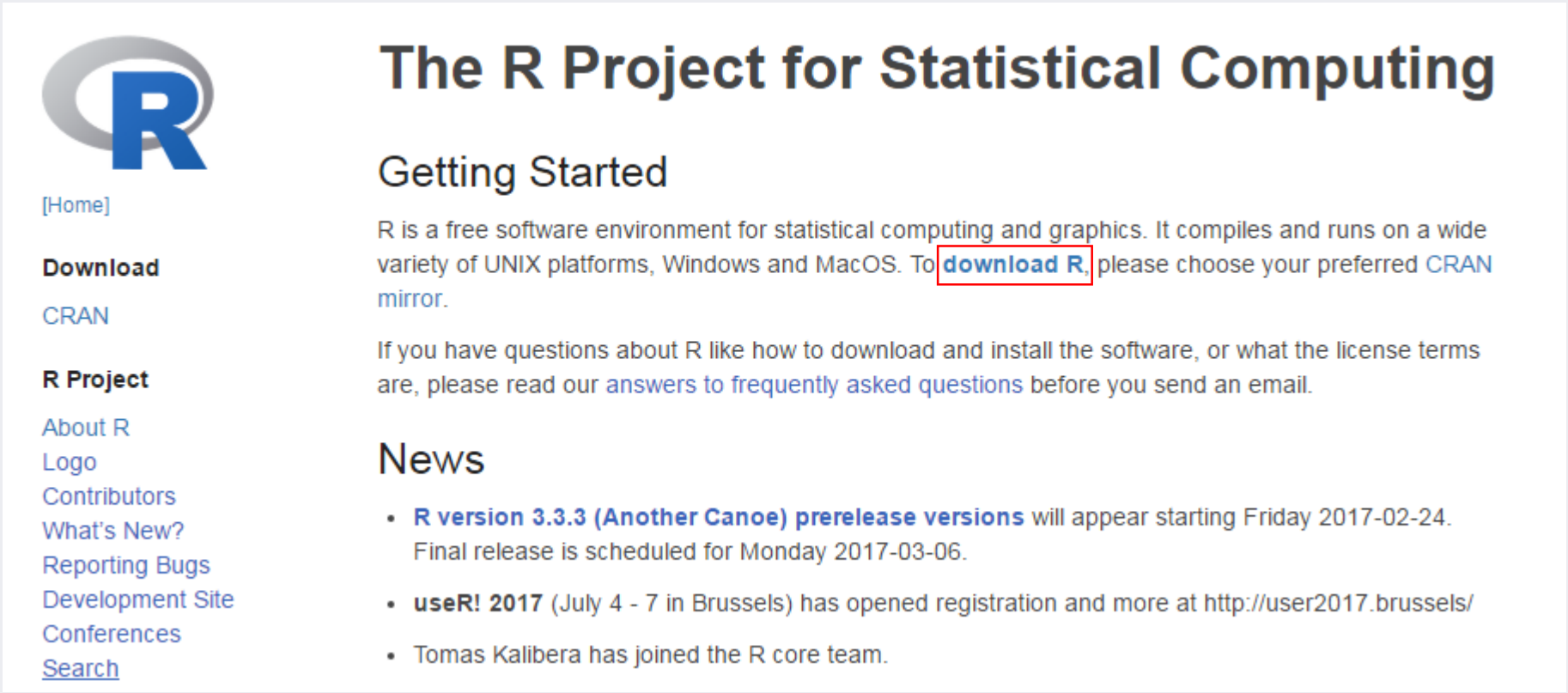
〈 Examle: packages available on CRAN 〉

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/

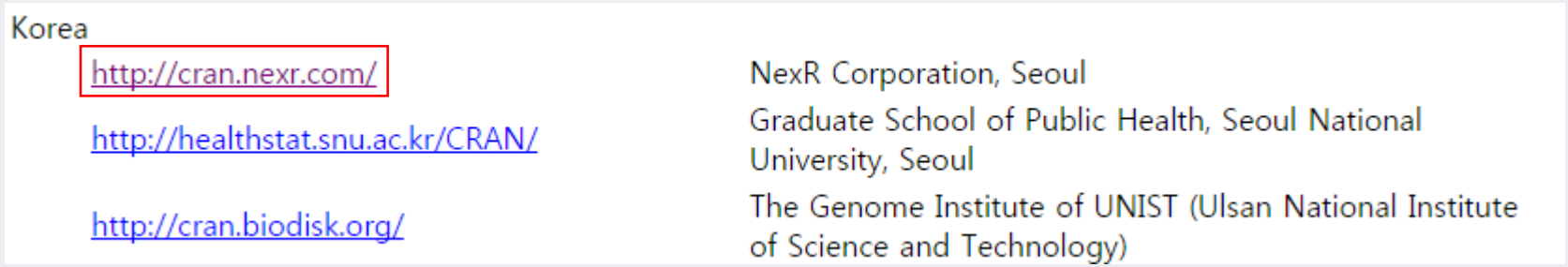
Install R

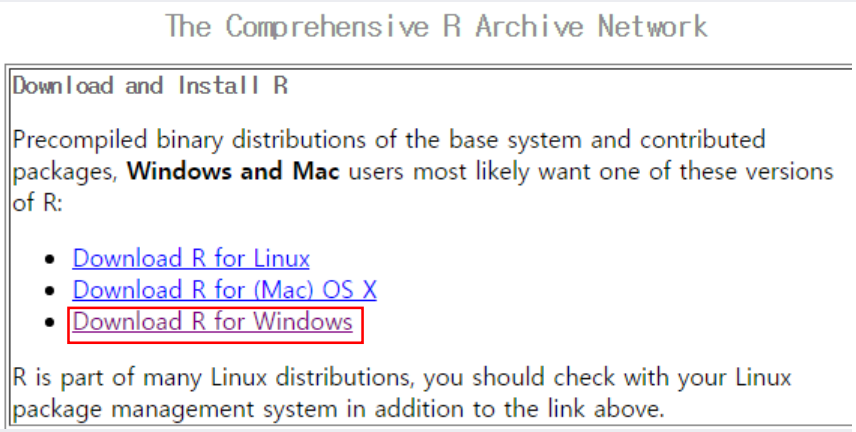
- For Windows
  - ✓ http://www.r-project.org
  - ✓ Download 'R base' file: the latest version is 3.3.2 for windows

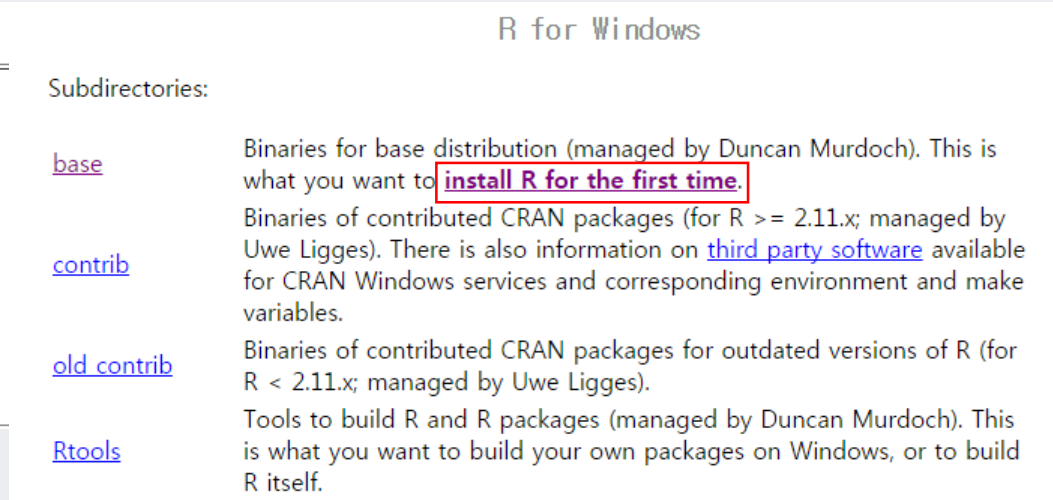


Install R

- For Windows
  - ✓ http://www.r-project.org
  - ✓ Download 'R base' file: the latest version is 3.3.2 for windows





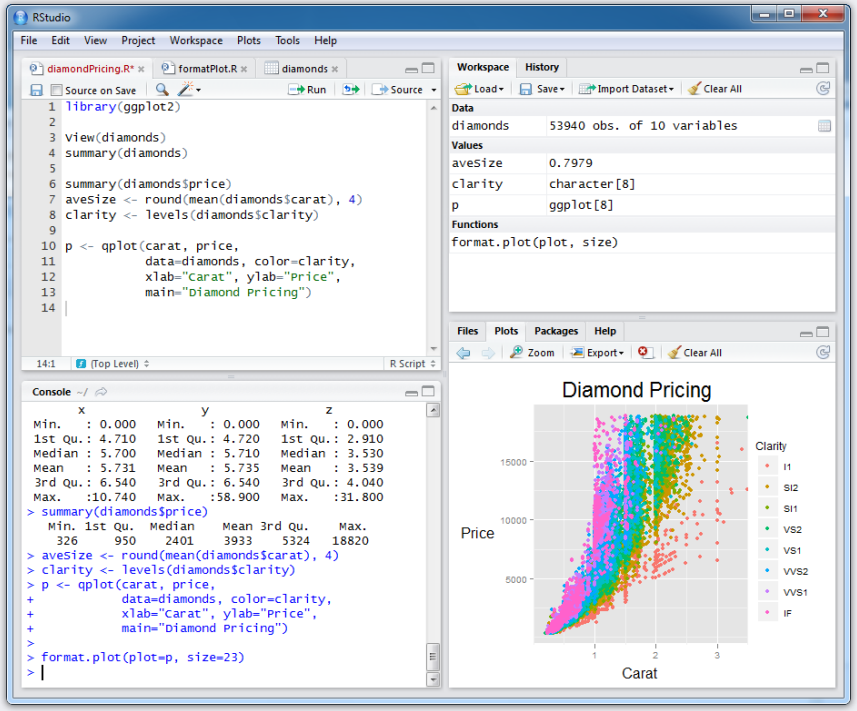


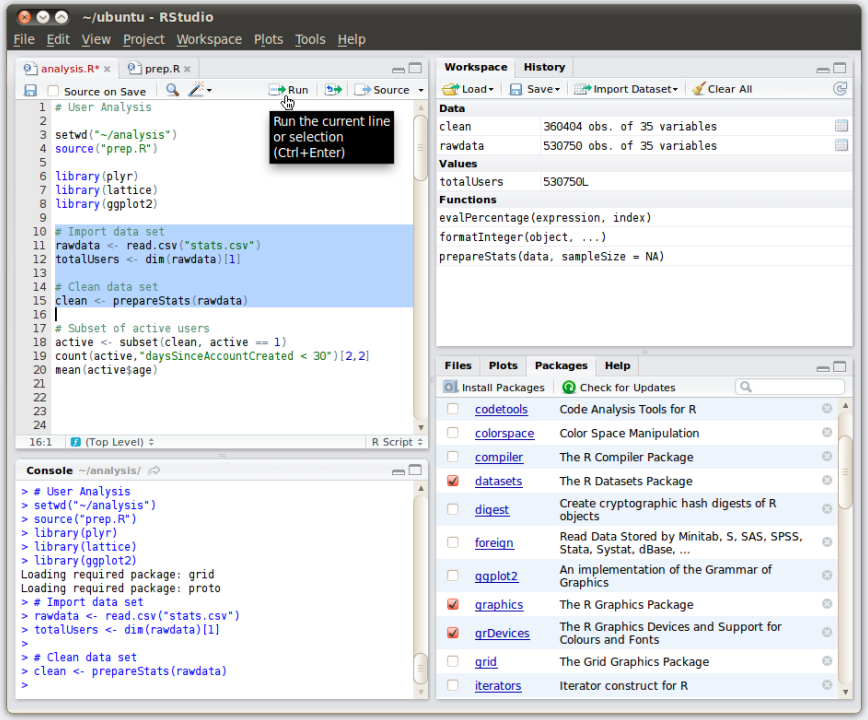
Install R Studio

- RStudio

  ✓ A free and open source integrated development environment (IDE) for R

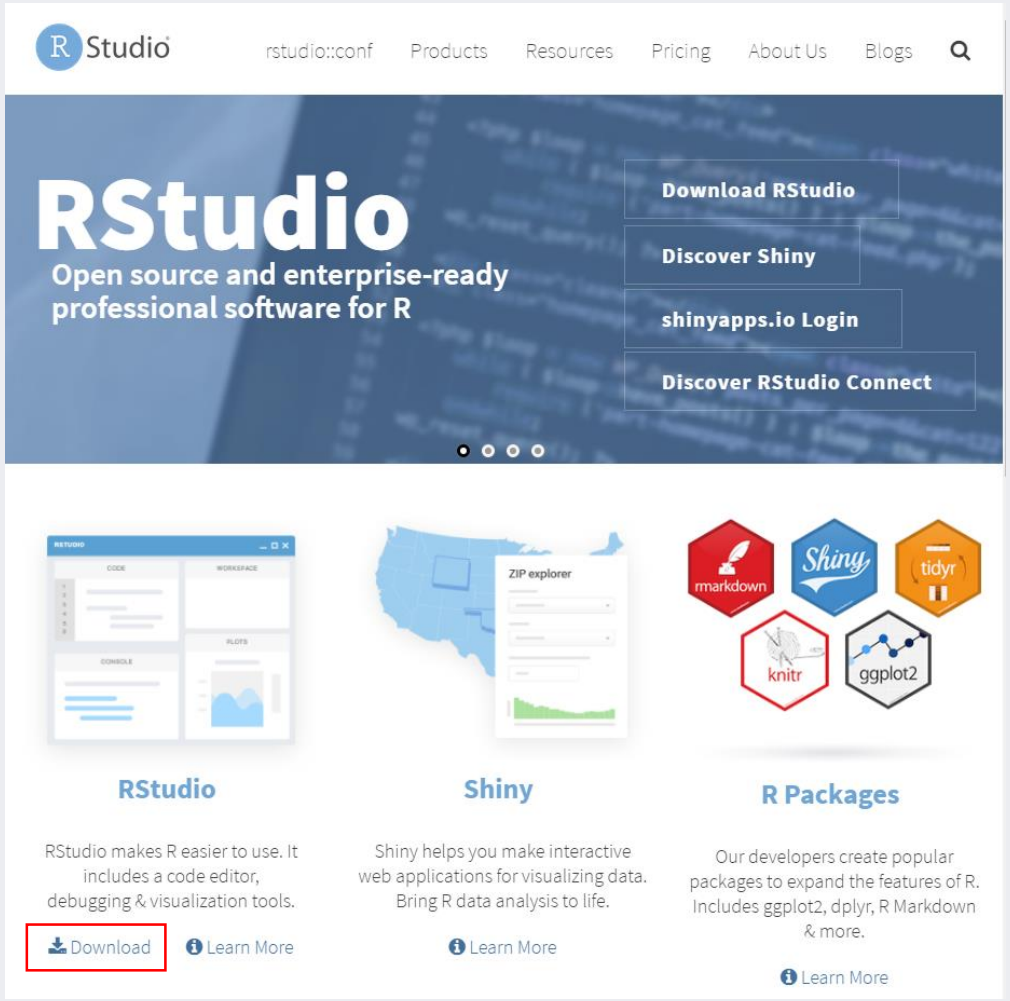
  - Available OS: Windows, Mac, or Linux
  - Can run it over the web using RStudio Server





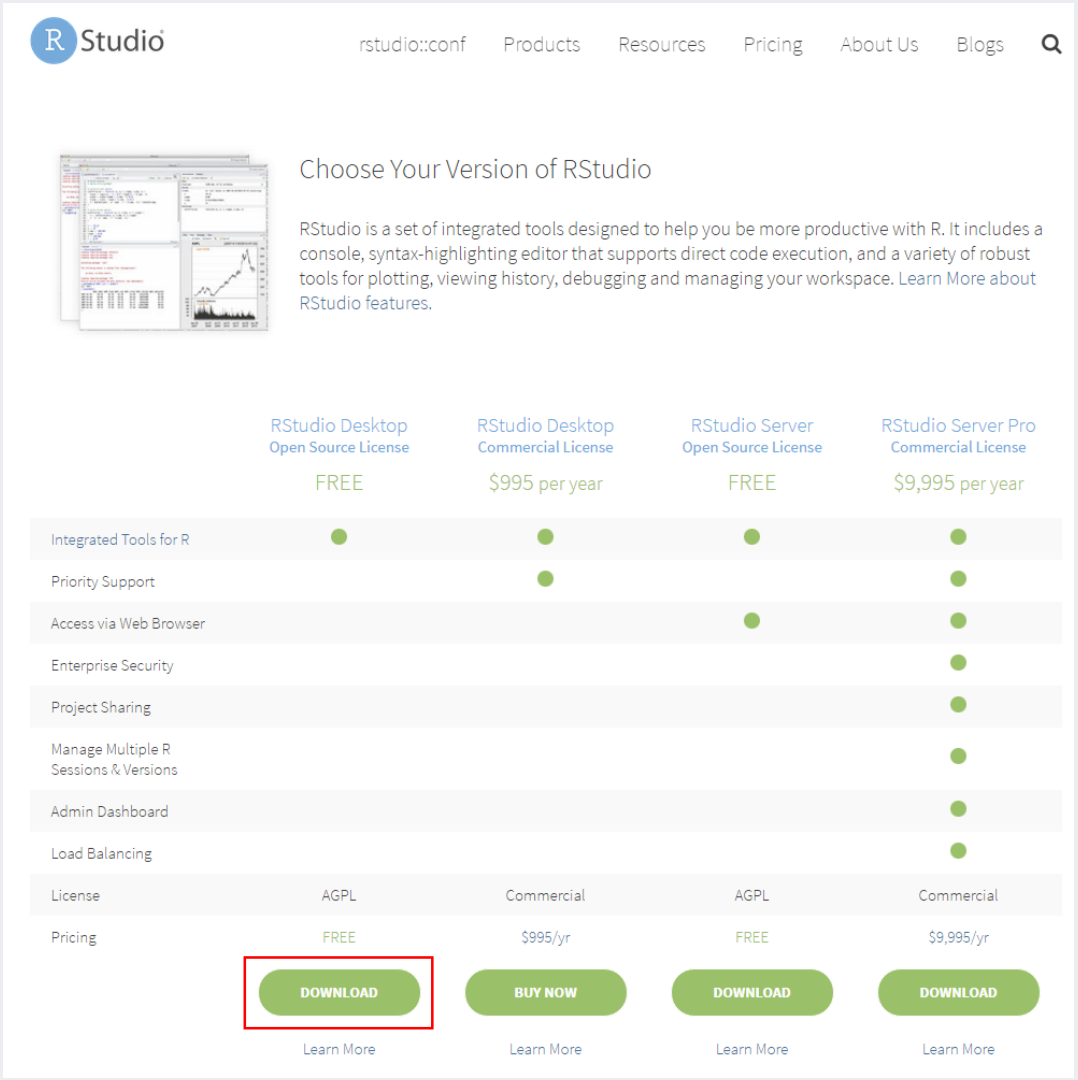
Install R Studio

- RStudio 
  - ✓ https://www.rstudio.com/



Install R Studio

- RStudio 



Install R Studio

- RStudio 

RStudio Desktop 1.0.136 — Release Notes

RStudio requires R 2.11.1+. If you don't already have R, download it here.

Installers for Supported Platforms

Installers	Size	Date	MD5
RStudio 1.0.136 - Windows Vista/7/8/10	81.9 MB	2016-12-21	93b3f307f567c33f7a4db4c114099b3e
RStudio 1.0.136 - Mac OS X 10.6+ (64-bit)	71.2 MB	2016-12-21	12d6d6ade0203a2fcef6fe3dea65c1ae
RStudio 1.0.136 - Ubuntu 12.04+/Debian 8+ (32-bit)	85.5 MB	2016-12-21	0a20fb89d8aaeb39b329a640ddadd2c5
RStudio 1.0.136 - Ubuntu 12.04+/Debian 8+ (64-bit)	92.1 MB	2016-12-21	2a73b88a12a9fbaf96251cecf8b41340
RStudio 1.0.136 - Fedora 19+/RedHat 7+/openSUSE 13.1+ (32-bit)	84.7 MB	2016-12-21	fa6179a7855bff0f939a34c169da45fd
RStudio 1.0.136 - Fedora 19+/RedHat 7+/openSUSE 13.1+ (64-bit)	85.7 MB	2016-12-21	2b3a148ded380b704e58496befb55545

Zip/Tarballs

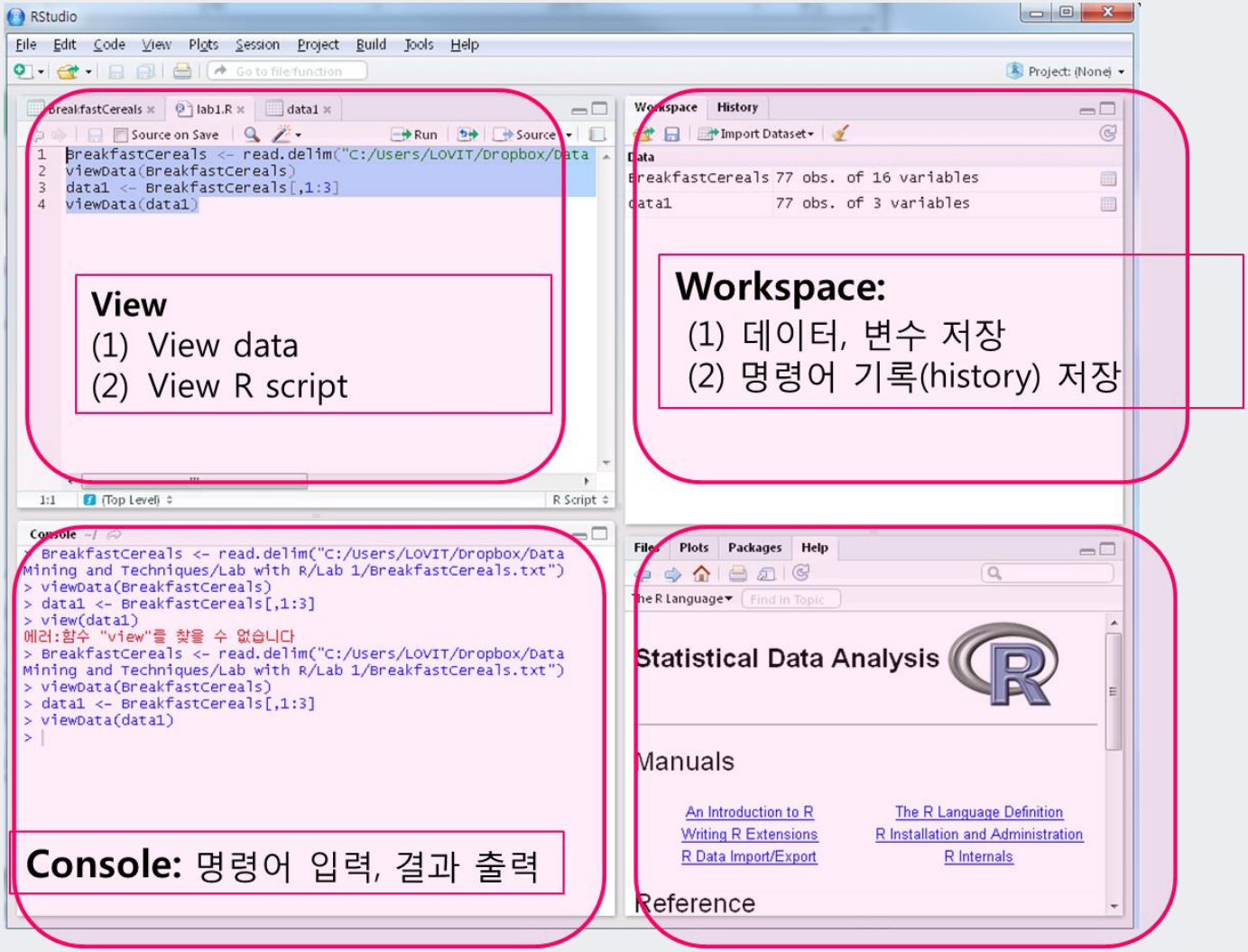
Zip/tar archives	Size	Date	MD5
RStudio 1.0.136 - Windows Vista/7/8/10	117.5 MB	2016-12-21	f415939bf5012c0ab127c7cfbc9600be
RStudio 1.0.136 - Ubuntu 12.04+/Debian 8+ (32-bit)	86.2 MB	2016-12-21	fca75f953dd425694b7fd4335bd29165
RStudio 1.0.136 - Ubuntu 12.04+/Debian 8+ (64-bit)	93.2 MB	2016-12-21	7cf0092653aa44fc76325a8f1325fb1f
RStudio 1.0.136 - Fedora 19+/RedHat 7+/openSUSE 13.1+ (32-bit)	85.4 MB	2016-12-21	30c89299d30ec03b38098e51e9bf49b8
RStudio 1.0.136 - Fedora 19+/RedHat 7+/openSUSE 13.1+ (64-bit)	86.6 MB	2016-12-21	ea2a262f650e92f568f48edc1c093902

Source Code

A tarball containing source code for RStudio v1.0.136 can be downloaded from here

R Studio

- R Studio windows



The Three Sexy Skills of Data Geeks

- Statistics (Studying)
  - ✓ The most important skill and the hardest to learn
  - ✓ Deep and rigorous discipline and actively progressing
- Data Munging (Suffering)
  - ✓ The painful process of cleaning, parsing, and proofing one's data before it's suitable for analysis
  - ✓ Real world data is messy!!
- Visualization (Storytelling)
  - ✓ exploratory data visualizations: intended to facilitate a data analyst's understanding of the data
  - ✓ communicate to a wider audience, whose goal is to visually advocate for a hypothesis

Reading list 1: A very short history of data science

AGENDA

01 R Overview

02 Handling Different Data Types

03 Text Processing

Data Types in R

- Questions
  - ✓ Q1: Are all variables homogeneous?
  - ✓ Q2: Are there more than one record?

Attribute\No. Records	1	>= 2
Homogeneous	Vector	Matrix or Array
Heterogeneous	List	Dataframe

- Dataframe makes R powerful to analyze heterogeneous multivariate data

Data Types in R

Scalar	**Vector**	List	Matrix	Array	Factor	Data.frame

- Vector
  - ✓ Vectors are homogeneous
    - All elements in a vector should be the same mode
  - ✓ Vector has an index for each element
    - A set of indices returns the corresponding sub-vector
    - Index starts from 1 (python: 0)
  - ✓ The elements of a vector can have its own name
  - ✓ Vectors in R is a column-wise vectors

```r
# Part 1-1: Data Handling (Vector) -------------

# Assign values to the vector A & B
A <- c(1,2,3)
B <- c(1, "A", 0.5)

# Check the mode
mode(A)
mode(B)

# Select a subset of vector
A[1]
A[2:3]
A[c(2,3)]

# Assign names
names(A)
names(A) <- c("First", "Second", "Third")

# call by index or name
A[1]
A["First"]
```

Handling Vectors

- Vector initiation
  - ✓ Do not have to initiate → creation and value assignment are done at the same time
    - a <- 3: create a vector named 'a' and assign the value 3 to it
- Add elements to an existing vector
  - ✓ The size of a vector is fixed when it is created
  - ✓ We have to recreate the vector if we want to add or remove some elements

```
24 # Data Handling: Vector
25 x <- c(1,2,3,4)
26 x
27 x <- c(x[1:3], 10, x[4])
28 x
29 length(x)
```

Handling Vectors

- Vector reuse
  - ✓ When R conduct an operation with two vectors, the shorter vector is reused to avoid an error

```
> 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
```

  - ✓ Column-first

```
> 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
```

Arithmetic Operators

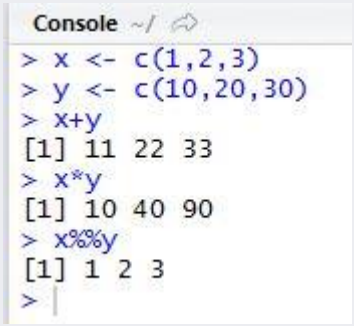
Operator	Description
+	addition
-	subtraction
*	multiplication
/	division
^ or **	exponentiation
x %% y	modulus (x mod y) 5%%2 is 1
x %/% y	integer division 5%/%2 is 2

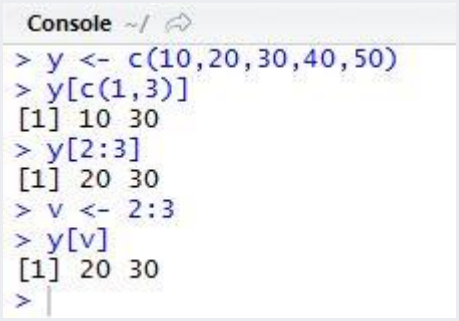
Logical Operators

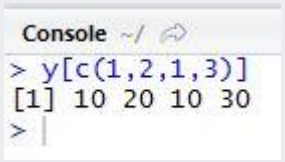
Operator	Description
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	exactly equal to
!=	not equal to
!x	Not x
x | y	x OR y
x & y	x AND y
isTRUE(x)	test if X is TRUE

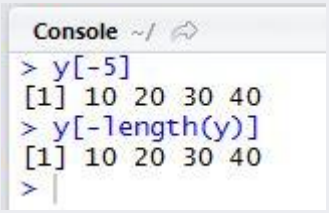
Handling Vectors

- Vector operations are element-wise
- Vector indexing
  - ✓ Extract a subset of vectors
  - ✓ Index can be used redundantly
  - ✓ A negative index is used to remove the corresponding element









Handling Vectors

- Creating vectors with operators
  - ✓ : operator: create vectors with certain range
  - ✓ seq: a generalized version of ":" operator
  - ✓ rep: repeat values

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
>
```

Handling Vectors

- Apply conditions for each element in a vector
  - ✓ any() function: return TRUE if at least one of the elements satisfies the condidition
  - ✓ all() function: return TRUE only when all elements satisfy the condition
- NA vs NULL
  - ✓ NA (Not Available): Some value exists but we cannot exactly know the value
  - ✓ NULL: Physically not exist

```
Console ~/
> x <- 1:10
> x > 8
 [1] FALSE 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
>
```

Handling Vectors

- Filtering: Extract the element that satisfy a given condition
  - ✓ Directly extract from index
  - ✓ subset(): return the values that satisfy the condition
  - ✓ which(): return the indices that satisfy the condition

```
> 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
```

Handling List

Scalar	Vector	List	Matrix	Array	Factor	Data.frame

- Lists are heterogeneous
  - ✓ Element in a list can have different modes
  - ✓ List can have other structured object such as dataframe as its element
- Elements in a list are referred by their index
- Elements in a list can have their names

```
91  # Part 1-1: Data Handling (List) -------------
92
93  # Example of a list
94  listA <- list(1, 2, "a")
95  print(listA)
96  listA[[1]]
97  listA[c(1,2)]
98  names(listA)
99  names(listA) <- c("First", "Second", "Third")
100
101 listA[["Third"]]
102 listA$Third
```

Handling List

- Creating a list
  - ✓ Use list() or vector() function
    - Element names can be assigned using tags

Console ~/

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

$salary
[1] 10000

$union
[1] TRUE

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

Console ~/

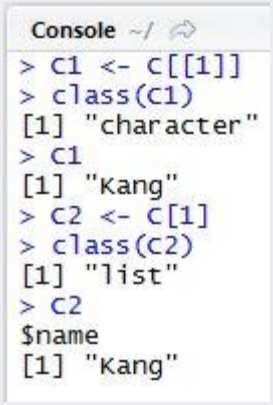
```
> B <- list("Kang", 10000, TRUE)
> B
[[1]]
[1] "Kang"

[[2]]
[1] 10000

[[3]]
[1] TRUE

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

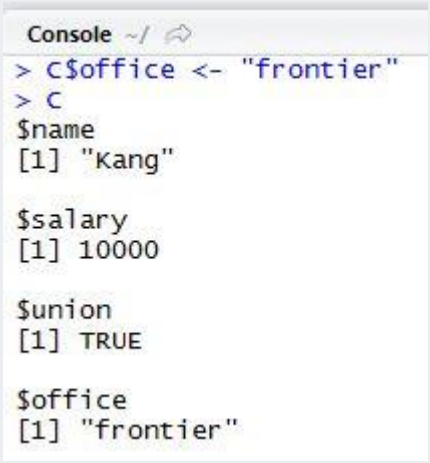
Console ~/

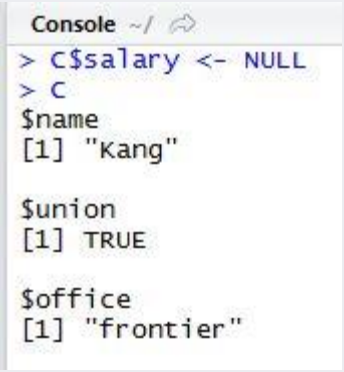
```
> C <- vector(mode="list")
> C[["name"]] <- "Kang"
> C[["salary"]] <- 10000
> C[["union"]] <- TRUE
> C
$name
[1] "Kang"

$salary
[1] 10000

$union
[1] TRUE
```

Handling List

- List operations
  - ✓ List indexing
    - Three ways of accessing the elements in a list
    - list$element_name, list[["element_name"]], list[[element's index]]
    - A list is returned if [ ] is used
  - ✓ Add/remove element in a list
    - Add: use a new name
    - Remove: use NULL







Handling List

- List operations (cont')
  - ✓ Unlist returns a vector with a single mode values

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

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


$b
[1] "Z"

$c
[1] NA

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

Handling List

- Applying functions to list
  - ✓ lapply( ) returns a list while sapply( ) returns a vector

```
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
>
```

Handling Matrix and Array

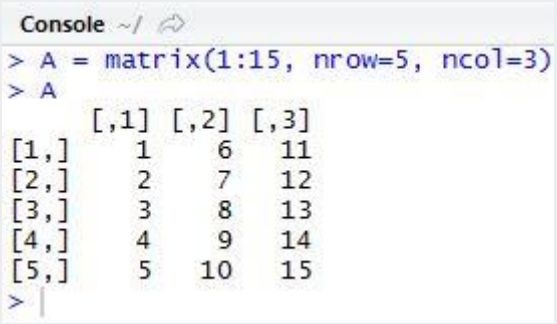
Scalar	Vector	List	**Matrix**	**Array**	Factor	Data.frame

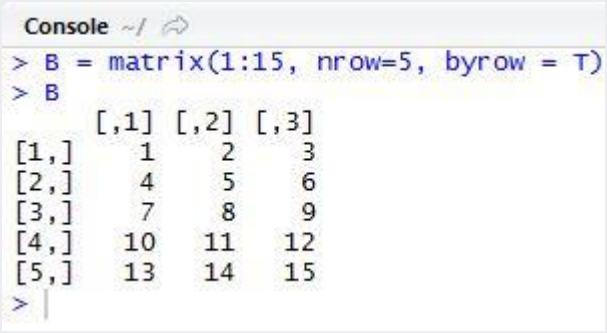
- Matrix
  - ✓ Matrix is a vector with dimensions
    - Vectors and lists can be transformed into a matrix
- Array
  - ✓ Matrix can be extended to n-dimensions
    - Indexed by multiple locations and returns subvectors

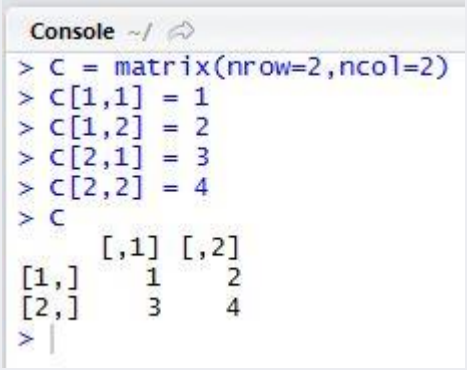
```
148 # Part 1-3: Data Handling (Matrix)
149
150 # Example of a matrix
151 A <- 1:6
152 dim(A)
153 print(A)
154
155 dim(A) <- c(2,3)
156 print(A)
157
158 B <- list(1,2,3,4,5,6)
159 print(B)
160 dim(B)
161 dim(B) <- c(2,3)
162 print(B)
163
164 D <- 1:12
165 dim(D) <- c(2,3,2)
166 print(D)
```

Handling Matrix and Array

- Features of matrix in R
  - ✓ Index begins with 1 (0 for python)
  - ✓ Column-major order
- Create a matrix: matrix( )
  - ✓ Method 1: provide all elements and assign the number of columns and rows (column first)
  - ✓ Method 2: provide all elements and assign the number of columns and rows (use row first option)
  - ✓ Method 3: Create an empty matrix and fill each element in







Handling Matrix and Array

- Matrix operations
  - ✓ Linear algebra of matrix: matrix multiplication, matrix-constant multiplication, etc.
  - ✓ Indexing and filtering

```
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
>
```

Handling Matrix and Array

- Applying functions to the rows/columns of matrix
  - ✓ Use apply( ) function family: apply( ), sapply( ), tapply( ), lapply( ), etc.
    - apply(m, dimcode, f, fargs)
      - m: matrix
      - dimcode: dimension to apply (1: row, 2: column)
      - f: function
      - fargs: arguments needed to execute f

```
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
>
```

Handling Matrix and Array

- Modifying matrix
  - ✓ rbind( ) & cbind( ): combine two matrices
  - ✓ rbind( ): combine two matrices with the same column names (top and bottom)
  - ✓ cbind( ): combine two matrices with the same row names (left and right)

```
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
>
```

Handling Matrix and Array

- Assign names for matrix columns/rows
  - ✓ Use colnames( ) and rownames( )

```
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
>
```

Handling Matrix and Array

- High dimensional array
  - ✓ Use array( ) function

```
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
```

Handling Factor

- Factor
  - ✓ Vector representation for nominal/categorical variables
    - Factor has its levels that are equivalent the number of possible values
- Usage
  - ✓ Categorical variable representation: 1 level for 1 category
  - ✓ Grouping and tagging
- Factor levels must be consistent!

```
245 # Part 1-4: Data Handling (Factor)
246
247 # Example of a factor
248 A <- c("Cho","Kim","Kang")
249 B <- as.factor(A)
250
251 print(A)
252 print(B)
253
254 mode(A)
255 mode(B)
256
257 A[1]+A[2]
258 B[1]+B[2]
```

```
> 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]) :
  요인(factors)에 대하여 의미있는 '+'가 아닙니다.
```

Handling Factor

- Factor
  - ✓ Factor in R is a vector with additional information
    - Additional information is a set of non-redundant values called level
    - The length of a factor is the number of elements, not levels
    - Possible to add a new level
    - A new value with non-existing level is considered as NA

```
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
>
```

Handling Factor

- Applying function to a factor
  - ✓ tapply( )
    - Useful to make frequency table with different categories
    - Can be used to more than two factors

```
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
>
```

Handling Factor

- Applying function to a factor
  - ✓ split( ) function
    - Used to make groups
    - can be used to more than two factors

```
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
```

Handling Dataframe

Scalar | Vector | List | Matrix | Array | **Factor** | Data.frame

- Dataframe
  - ✓ A table with rows and columns
  - ✓ Regarded as a special case of list
  - ✓ Can have different modes for different columns
  - ✓ Elements in a column must have the same modes
  - ✓ Columns can have names

```
296 # Part 1-5: Data Handling (DataFrame) ------
297
298 # Example of data frame
299 A <- c(1,2,3)
300 B <- c("a","b","c")
301 C <- data.frame(A,B)
302 C
303 C[[1]]
304 C[[2]]
305 C[1,2]
306 C$B[2]
307
308 C <- data.frame(A,B, stringsAsFactors=FALSE)
309 C
310 C[[1]]
311 C[[2]]
312 C[1,2]
313 C$B[2]
```

Handling Dataframe

- Creating and accessing Dataframe
  - ✓ Use data.frame( ) function to create a dataframe
  - ✓ Three ways to access a certain element

```
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"
>
```

Handling Dataframe

- Extracting and filtering a subset of dataframe
  - ✓ Same as matrix
- Combine dataframe
  - ✓ Same as matrix

```
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  3.7
3   4.0     4  4.0
4   2.3     0  3.3
5   2.3     1  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
>
```

Handling Dataframe

- Merge dataframes
  - ✓ If there are more than on sources of data tables in a database
  - ✓ Inner/outer/left/right joins are possible

```
> 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
```

Handling Dataframe

- Merge dataframes
  - ✓ If different data frames have different column name strategy, explicitly state the column names to use

```
firstname <- c("Alice","Jill", "Laura")
state <- c("MA", "NY", "CA")
dfC <- data.frame(firstname, state, stringsAsFactors=FALSE)
dfC

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
```

AGENDA

01 R Overview

02 Handling Different Data Types

03 Text Processing

Text Processing

- The length of a string
  - ✓ Use nchar( ) function instead of length( )
    - Space and special characters can be counted as well
- Concatenate strings
  - ✓ Use paste( ) function
    - Various spacing strategies can be used
    - Non-character values are also possible

```
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

- Extract sub-strings
  - ✓ Use substring(string, start, end) function
    - Extract the substring that begins with "start" and ends with "end"
    - If the string argument is a vector, the other options are applied to all elements

```
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

- Split text
  - ✓ Use strsplit(string, separator) function
    - A simple string or regular expression can be used as a separator
    - Ex: split the file path using "/" as a separator

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

```
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

- Regular expression
  - ✓ a sequence of characters that define a search pattern
  - ✓ this pattern is then used by string searching algorithms

```
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"
```

Regular expression in R

Text Processing

- Regular expression

POSIX	비표준	펄/Tcl	Vim	ASCII	설명
[:alnum:]				[A-Za-z0-9]	영숫자
	[:word:]	₩w	₩w	[A-Za-z0-9_]	영숫자 + "_"
		₩W	₩W	[^A-Za-z0-9_]	낱말이 아닌 문자
[:alpha:]			₩a	[A-Za-z]	알파벳 문자
[:blank:]			₩s	[₩t]	공백과 탭
		₩b	₩< ₩>	(?<=₩W)(?=₩w)|(?<=₩w)(?=₩W)	낱말 경계
[:cntrl:]				[₩x00-₩x1F₩x7F]	제어 문자
[:digit:]		₩d	₩d	[0-9]	숫자
		₩D	₩D	[^0-9]	숫자가 아닌 문자
[:graph:]				[₩x21-₩x7E]	보이는 문자
[:lower:]			₩l	[a-z]	소문자
[:print:]			₩p	[₩x20-₩x7E]	보이는 문자 및 공백 문자
[:punct:]				[][!"#$%&'()*+,./:;<=>?@₩^_`{|}~-]	구두점
[:space:]		₩s	₩_s (단순히 줄 끝에 추가)	[₩t₩r₩n₩v₩f]	공백 문자
		₩S		[^ ₩t₩r₩n₩v₩f]	공백이 아닌 모든 문자
[:upper:]			₩u	[A-Z]	대문자
[:xdigit:]			₩x	[A-Fa-f0-9]	16진수

From: wikipedia

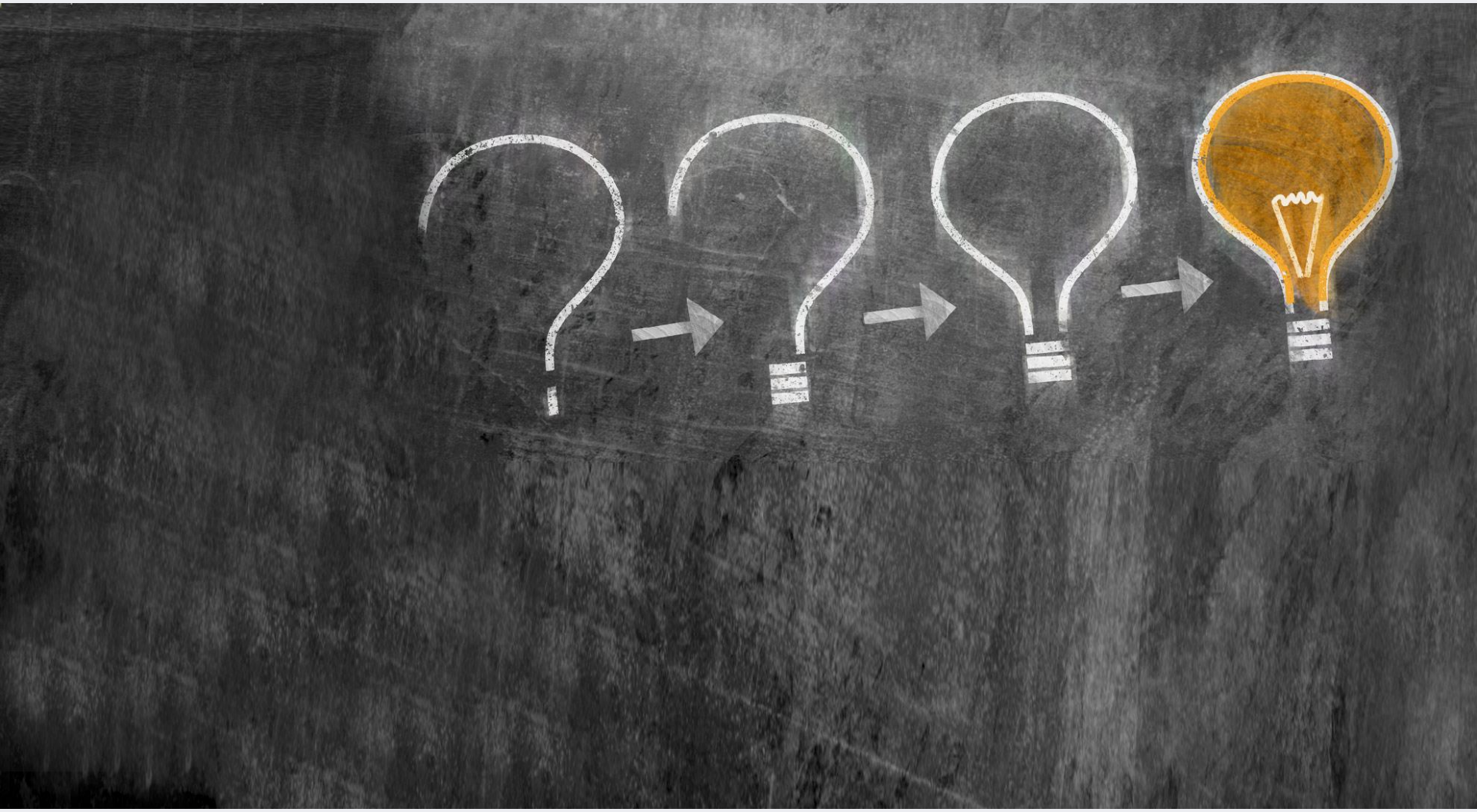
Text Processing

- Substitution
  - ✓ Use sub(old, new, string) or gsub(old, new, string) functions
  - ✓ sub( ) replaces the first substring whereas gsub( ) replaces all substrings

```
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"
```

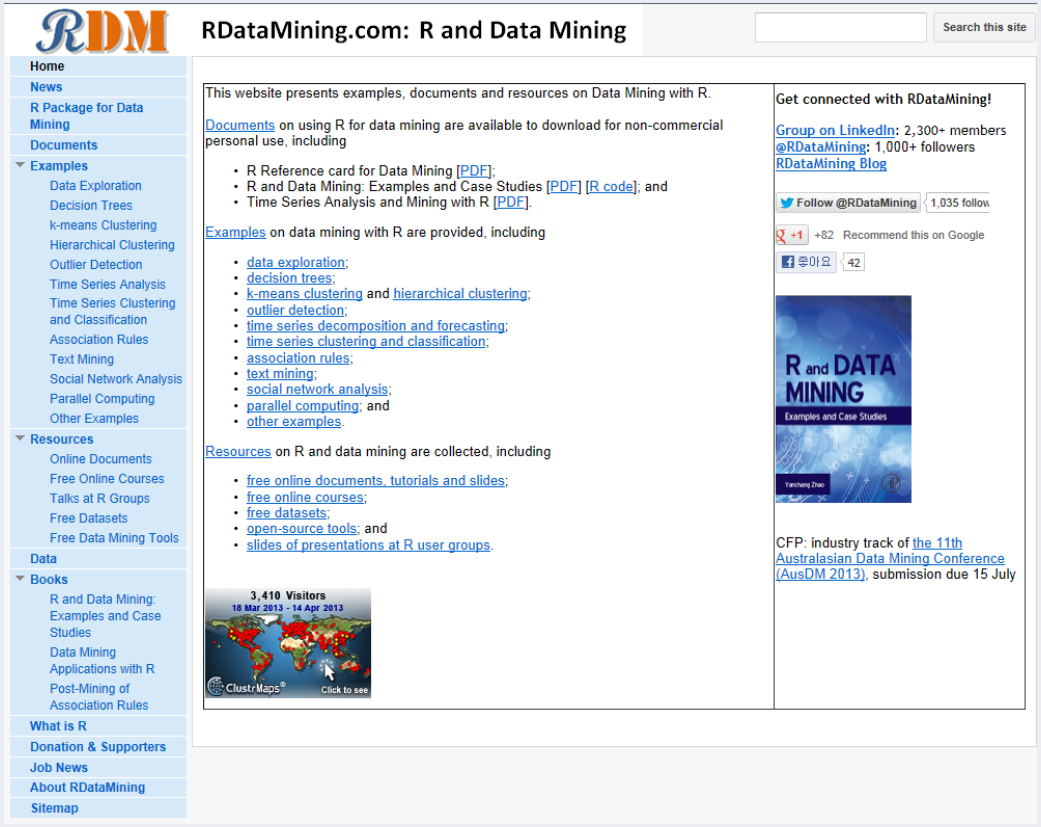
- String pattern matching
  - ✓ Use grep(pattern, x) function
    - Return the index that matches pattern

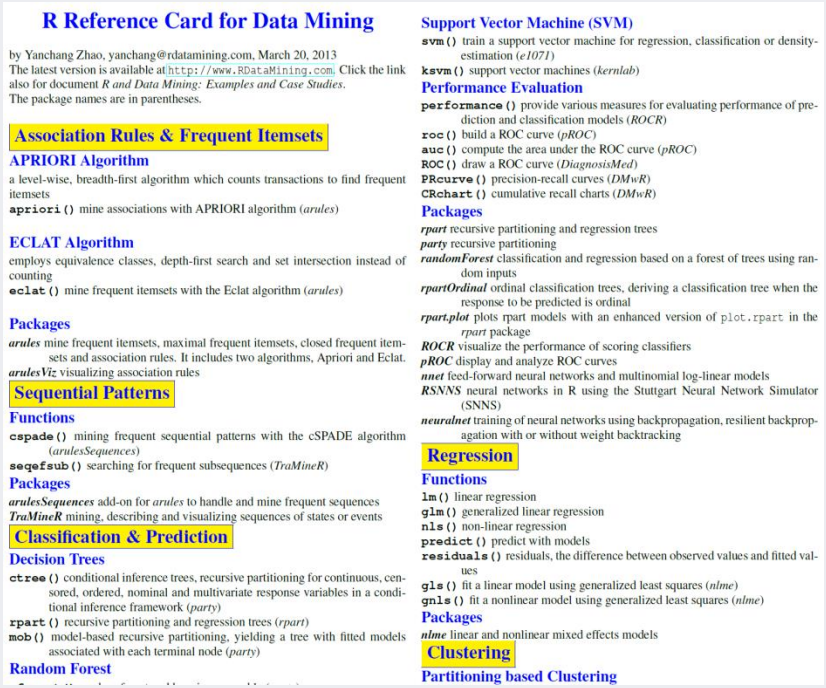
```
Console ~/
> grep("mike",path)
[1] 1 2 3
> grep("errors",path)
[1] 2
```



References

- R Datamining
  - ✓ http://www.rdatamining.com





References

- R Bloggers
  - ✓ http://r-bloggers.com