



R Syntax I: Data Types and Strings

Pilsung Kang

School of Industrial Management Engineering

Korea University

AGENDA

01 Handling Different Data Types

02 String Processing

Basic Instructions

- Getting Help, Using Packages, and Working Directory

Getting Help

Accessing the help files

?mean

Get help of a particular function.

help.search('weighted mean')

Search the help files for a word or phrase.

help(package = 'dplyr')

Find help for a package.

More about an object

str(iris)

Get a summary of an object's structure.

class(iris)

Find the class an object belongs to.

Using Packages

install.packages('dplyr')

Download and install a package from CRAN.

library(dplyr)

Load the package into the session, making all its functions available to use.

dplyr::select

Use a particular function from a package.

data(iris)

Load a built-in dataset into the environment.

Working Directory

getwd()

Find the current working directory (where inputs are found and outputs are sent).

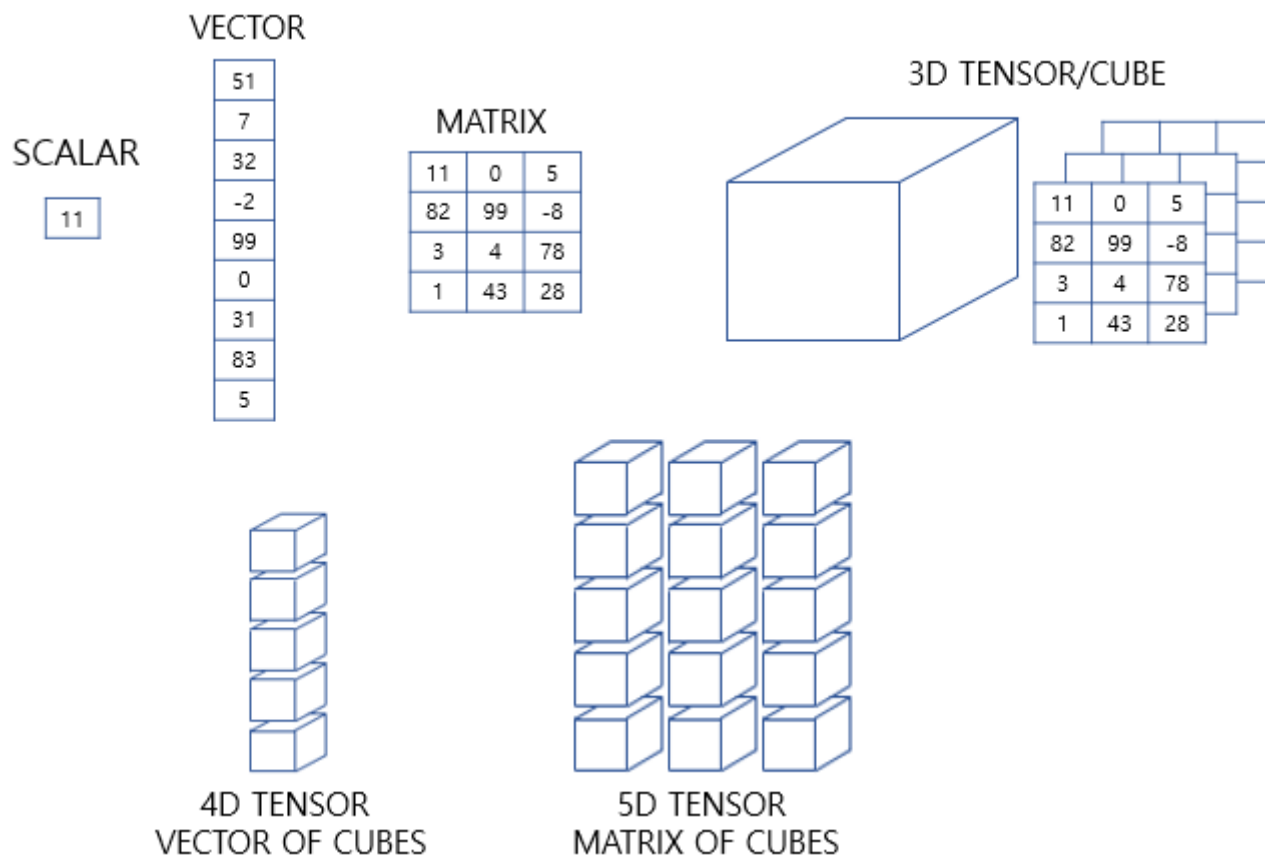
setwd('C://file/path')

Change the current working directory.

Use projects in RStudio to set the working directory to the folder you are working in.

Data Types in R

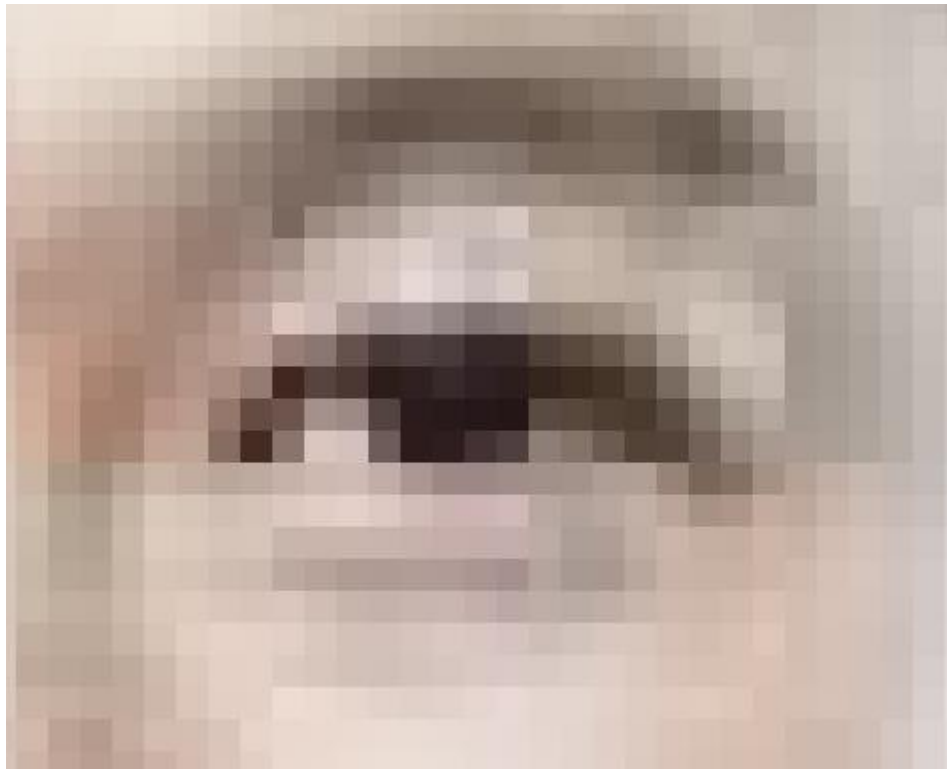
- Data Types w.r.t. dimensions



<https://wikidocs.net/37001>

Data Types in R

- Tensor in Data Analytics
 - ✓ Whose eye is it?

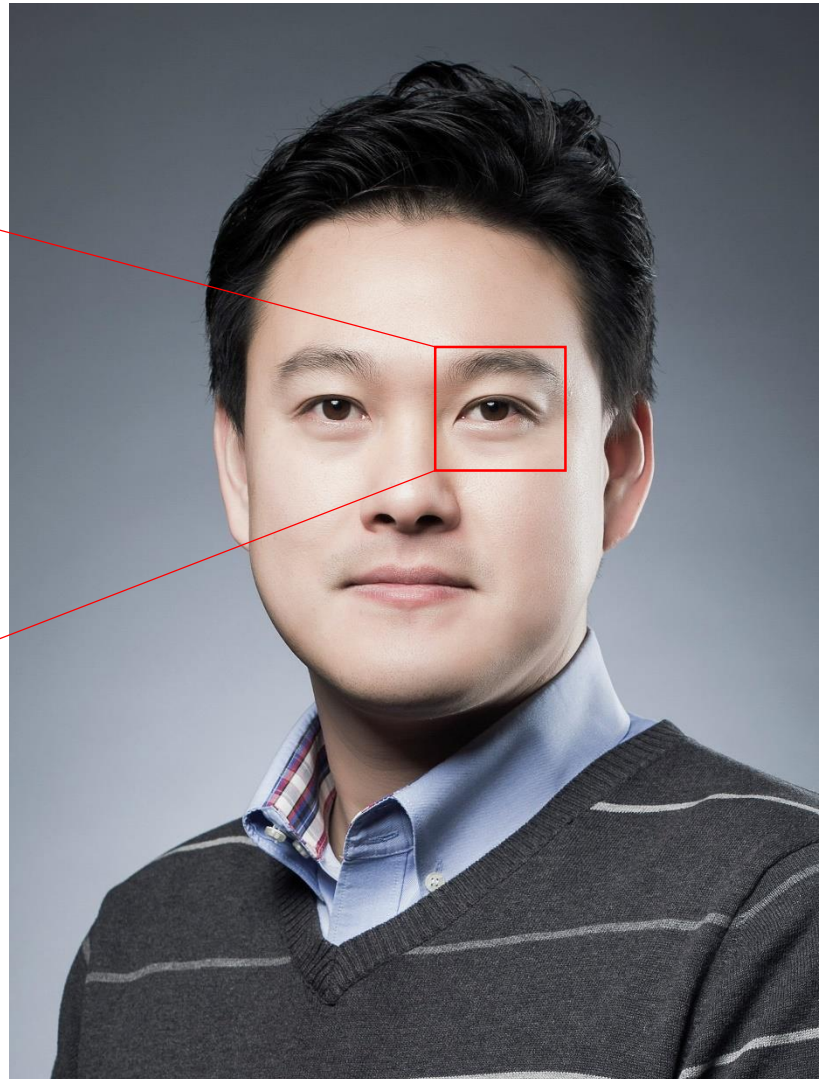
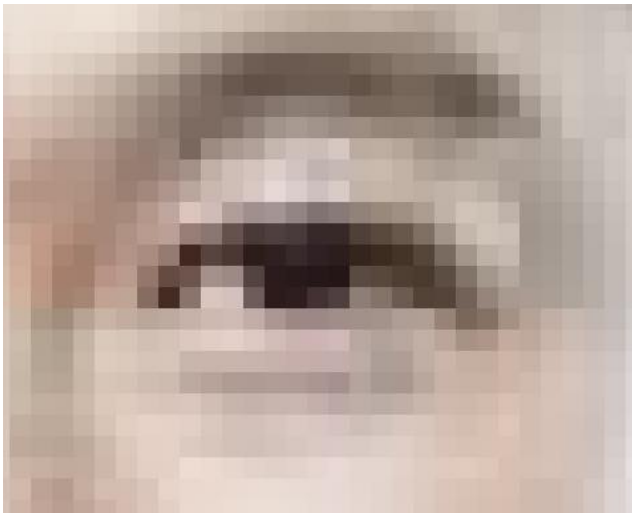


Data Types in R

- Tensor in Data Analytics

- ✓ Whose eye is it?

- It's my eye



Data Types in R

- Computers recognize an image as a 3-D Tensor: Width X Height X 3 (RGB)

1,685

2,247



```
> ps kang[1:10, 1:10, 1]
      [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]      [,8]      [,9]     [,10]
[1,] 0.2784314 0.2784314 0.2745098 0.2745098 0.2745098 0.2745098 0.2705882 0.2705882 0.2862745 0.2901961
[2,] 0.2745098 0.2745098 0.2745098 0.2745098 0.2745098 0.2745098 0.2745098 0.2745098 0.2666667 0.2705882
[3,] 0.2745098 0.2745098 0.2745098 0.2745098 0.2745098 0.2745098 0.2745098 0.2745098 0.2705882 0.2745098
[4,] 0.2705882 0.2705882 0.2745098 0.2745098 0.2745098 0.2745098 0.2745098 0.2784314 0.2784314 0.2823529
[5,] 0.2705882 0.2705882 0.2705882 0.2745098 0.2745098 0.2784314 0.2784314 0.2784314 0.2784314 0.2784314
[6,] 0.2705882 0.2705882 0.2745098 0.2745098 0.2745098 0.2745098 0.2784314 0.2784314 0.2823529 0.2784314
[7,] 0.2705882 0.2745098 0.2745098 0.2745098 0.2745098 0.2745098 0.2745098 0.2745098 0.2784314 0.2941176
[8,] 0.2745098 0.2745098 0.2745098 0.2745098 0.2745098 0.2745098 0.2745098 0.2745098 0.2901961 0.2823529
[9,] 0.2745098 0.2705882 0.2745098 0.2784314 0.2823529 0.2823529 0.2745098 0.2666667 0.2784314 0.2784314
[10,] 0.2784314 0.2784314 0.2784314 0.2823529 0.2862745 0.2862745 0.2784314 0.2745098 0.2745098 0.2745098
```

```
> ps kang[1:10, 1:10, 2]
      [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]      [,8]      [,9]     [,10]
[1,] 0.2980392 0.2980392 0.2941176 0.2941176 0.2941176 0.2941176 0.2901961 0.2901961 0.2980392 0.3019608
[2,] 0.2941176 0.2941176 0.2941176 0.2941176 0.2941176 0.2941176 0.2941176 0.2941176 0.2784314 0.2823529
[3,] 0.2941176 0.2941176 0.2941176 0.2941176 0.2941176 0.2941176 0.2941176 0.2941176 0.2823529 0.2862745
[4,] 0.2901961 0.2901961 0.2941176 0.2941176 0.2941176 0.2941176 0.2980392 0.2980392 0.2941176 0.2941176
[5,] 0.2901961 0.2901961 0.2901961 0.2941176 0.2941176 0.2980392 0.2980392 0.2980392 0.2901961 0.2901961
[6,] 0.2901961 0.2901961 0.2941176 0.2941176 0.2941176 0.2941176 0.2980392 0.2980392 0.2941176 0.2901961
[7,] 0.2901961 0.2941176 0.2941176 0.2941176 0.2941176 0.2941176 0.2941176 0.2980392 0.3058824 0.2980392
[8,] 0.2941176 0.2941176 0.2941176 0.2941176 0.2941176 0.2941176 0.2941176 0.2941176 0.3019608 0.2941176
[9,] 0.2862745 0.2823529 0.2862745 0.2901961 0.2941176 0.2941176 0.2862745 0.2784314 0.2901961 0.2901961
[10,] 0.2901961 0.2901961 0.2901961 0.2941176 0.2980392 0.2980392 0.2901961 0.2862745 0.2862745 0.2862745
```

```
> ps kang[1:10, 1:10, 3]
      [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]      [,8]      [,9]     [,10]
[1,] 0.3215686 0.3215686 0.3176471 0.3176471 0.3176471 0.3176471 0.3137255 0.3137255 0.3254902 0.3294118
[2,] 0.3176471 0.3176471 0.3176471 0.3176471 0.3176471 0.3176471 0.3176471 0.3176471 0.3058824 0.3098039
[3,] 0.3176471 0.3176471 0.3176471 0.3176471 0.3176471 0.3176471 0.3176471 0.3176471 0.3098039 0.3137255
[4,] 0.3137255 0.3137255 0.3176471 0.3176471 0.3176471 0.3176471 0.3215686 0.3215686 0.3215686 0.3215686
[5,] 0.3137255 0.3137255 0.3137255 0.3176471 0.3176471 0.3215686 0.3215686 0.3215686 0.3176471 0.3176471
[6,] 0.3137255 0.3137255 0.3176471 0.3176471 0.3176471 0.3215686 0.3215686 0.3215686 0.3215686 0.3176471
[7,] 0.3137255 0.3176471 0.3176471 0.3176471 0.3176471 0.3176471 0.3176471 0.3215686 0.3333333 0.3254902
[8,] 0.3176471 0.3176471 0.3176471 0.3176471 0.3176471 0.3176471 0.3176471 0.3176471 0.3294118 0.3215686
[9,] 0.3058824 0.3019608 0.3058824 0.3098039 0.3137255 0.3137255 0.3058824 0.2980392 0.3098039 0.3098039
[10,] 0.3098039 0.3098039 0.3098039 0.3137255 0.3176471 0.3176471 0.3098039 0.3058824 0.3058824 0.3058824
```

Data Types in R

- Variable types

- ✓ Homogeneous variables

- All elements are the same type: numeric values in this example

	Year	January	February	March	April	May	June	July	August	September	October	November	December
1	1998	0	0	2	21	47	272	391	262	251	178	47	8
2	1999	0	4	1	24	145	230	448	195	117	248	17	2
3	2000	3	9	0	28	74	281	309	341	190	169	10	8
4	2001	1	1	1	64	42	245	271	233	177	127	30	2
5	2002	2	12	2	24	87	179	107	173	80	178	18	1
6	2003	0	2	18	37	7	182	205	172	72	166	9	1
7	2004	2	1	6	54	178	202	201	193	140	97	22	0
8	2005	5	2	3	67	57	239	472	295	210	196	35	3
9	2006	0	0	26	17	152	270	356	273	168	74	67	0
10	2007	0	0	1	36	62	344	371	350	270	118	19	8
11	2008	0	15	129	33	61	172	189	262	161	106	37	2

Data Types in R

- Variable types
 - ✓ Homogeneous variables
 - Variables (Columns) are different types

	Name	Number	Position	Age	Height	Weight	College	Salary
0	Avery Bradley	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
1	Jae Crowder	99.0	SF	25.0	6-6	235.0	Marquette	6796117.0
2	John Holland	30.0	SG	27.0	6-5	205.0	Boston University	NaN
3	R.J. Hunter	28.0	SG	22.0	6-5	185.0	Georgia State	1148640.0
4	Jonas Jerebko	8.0	PF	29.0	6-10	231.0	NaN	5000000.0
5	Amir Johnson	90.0	PF	29.0	6-9	240.0	NaN	12000000.0
6	Jordan Mickey	55.0	PF	21.0	6-8	235.0	LSU	1170960.0
7	Kelly Olynyk	41.0	C	25.0	7-0	238.0	Gonzaga	2165160.0
8	Terry Rozier	12.0	PG	22.0	6-2	190.0	Louisville	1824360.0
9	Marcus Smart	36.0	PG	22.0	6-4	220.0	Oklahoma State	3431040.0

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

```
1 # Part 1-1: Data Handling (Vector) -----
2
3 # Assign values to the vector A & B
4 A <- c(1,2,3)
5 B <- c(1, "A", 0.5)
6
7 # Check the mode
8 mode(A)
9 mode(B)
10
11 # Select a subset of vector
12 A[1]
13 A[2:3]
14 A[c(2,3)]
15
16 # Assign names
17 names(A)
18 names(A) <- c("First", "Second", "Third")
19
20 # call by index or name
21 A[1]
22 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

```
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  
> |
```

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 condition
 - ✓ `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`

```
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"
```

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

```
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

```
> |
```


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

Scalar

Vector

List

Matrix

Array

Factor

Data.frame

- 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 one 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 Handling Different Data Types

02 String Processing

String 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"  
> |
```


String Processing

- Extract sub-strings
 - ✓ Use substr(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"  
> |
```

String 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"
```

String 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](#)

String Processing


- Regular expression

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

String 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>

**RDataMining.com: R and Data Mining**

Search this site

Home

News

R Package for Data Mining

Documents

Examples

- Data Exploration
- Decision Trees
- k-means Clustering
- Hierarchical Clustering
- Outlier Detection
- Time Series Analysis
- Time Series Clustering and Classification
- Association Rules
- Text Mining
- Social Network Analysis
- Parallel Computing
- Other Examples

Resources

- Online Documents
- Free Online Courses
- Talks at R Groups
- Free Datasets
- Free Data Mining Tools

Data

Books

- R and Data Mining: Examples and Case Studies
- Data Mining
- Applications with R
- Post-Mining of Association Rules

What is R

Donation & Supporters

Job News

About RDataMining

Sitemap

This website presents examples, documents and resources on Data Mining with R.

[Documents](#) on using R for data mining are available to download for non-commercial personal use, including

- R Reference card for Data Mining [\[PDF\]](#);
- R and Data Mining: Examples and Case Studies [\[PDF\]](#) [\[R code\]](#); and
- Time Series Analysis and Mining with R [\[PDF\]](#).

[Examples](#) on data mining with R are provided, including

- [data exploration](#);
- [decision trees](#);
- [k-means clustering](#) and [hierarchical clustering](#);
- [outlier detection](#);
- [time series decomposition and forecasting](#);
- [time series clustering and classification](#);
- [association rules](#);
- [text mining](#);
- [social network analysis](#);
- [parallel computing](#); and
- [other examples](#).

[Resources](#) on R and data mining are collected, including

- [free online documents, tutorials and slides](#);
- [free online courses](#);
- [free datasets](#);
- [open-source tools](#); and
- [slides of presentations at R user groups](#).

Get connected with RDataMining!

Group on LinkedIn: 2,300+ members

@RDataMining: 1,000+ followers


[RDataMining Blog](#)

Follow @RDataMining

1,035 follow

+82 Recommend this on Google


42



CFP: industry track of the 11th Australasian Data Mining Conference (AusDM 2013), submission due 15 July

3,410 Visitors

18 Mar 2013 - 14 Apr 2013



Click to see

R Reference Card for Data Mining

by Yanchang Zhao, yanchang@rdatamining.com, March 20, 2013
The latest version is available at <http://www.rdatamining.com>. Click the link also for document *R and Data Mining: Examples and Case Studies*.
The package names are in parentheses.

Association Rules & Frequent Itemsets

APRIORI Algorithm

a level-wise, breadth-first algorithm which counts transactions to find frequent itemsets

apriori() mine associations with APRIORI algorithm (*arules*)

ECLAT Algorithm

employs equivalence classes, depth-first search and set intersection instead of counting

eclat() mine frequent itemsets with the Eclat algorithm (*arules*)

Packages

arules mine frequent itemsets, maximal frequent itemsets, closed frequent itemsets and association rules. It includes two algorithms, Apriori and Eclat.

arulesViz visualizing association rules

Sequential Patterns

Functions

cspade() mining frequent sequential patterns with the cSPADE algorithm (*arulesSequences*)

seqfsub() searching for frequent subsequences (*TraMineR*)

Packages

arulesSequences add-on for *arules* to handle and mine frequent sequences

TraMineR mining, describing and visualizing sequences of states or events

Classification & Prediction

Decision Trees

ctree() conditional inference trees, recursive partitioning for continuous, censored, ordered, nominal and multivariate response variables in a conditional inference framework (*party*)

rpart() recursive partitioning and regression trees (*rpart*)

mob() model-based recursive partitioning, yielding a tree with fitted models associated with each terminal node (*party*)

Random Forest

Support Vector Machine (SVM)

svm() train a support vector machine for regression, classification or density-estimation (*e1071*)

ksvm() support vector machines (*kernelab*)

Performance Evaluation

performance() provide various measures for evaluating performance of prediction and classification models (*ROCR*)

roc() build a ROC curve (*pROC*)

auc() compute the area under the ROC curve (*pROC*)

ROC() draw a ROC curve (*DiagnosisMed*)

PRcurve() precision-recall curves (*DMwR*)

CRchart() cumulative recall charts (*DMwR*)

Packages

rpart recursive partitioning and regression trees

party recursive partitioning

randomForest classification and regression based on a forest of trees using random inputs

rpartOrdinal ordinal classification trees, deriving a classification tree when the response to be predicted is ordinal

rpart.plot plots *rpart* models with an enhanced version of *plot.rpart* in the *rpart* package

ROCR visualize the performance of scoring classifiers

pROC display and analyze ROC curves

nnet feed-forward neural networks and multinomial log-linear models

RNNNS neural networks in R using the Stuttgart Neural Network Simulator (SNNS)

neuralnet training of neural networks using backpropagation, resilient backpropagation with or without weight backtracking

Regression

Functions

lm() linear regression

glm() generalized linear regression

nls() non-linear regression

predict() predict with models

residuals() residuals, the difference between observed values and fitted values

glm.s() fit a linear model using generalized least squares (*nlme*)

gnls() fit a nonlinear model using generalized least squares (*nlme*)

Packages

nlme linear and nonlinear mixed effects models

Clustering

Partitioning based Clustering

References

- R Bloggers

✓ <http://r-bloggers.com>

R-bloggers

R news and tutorials contributed by (452) R bloggers

[Home](#) [About](#) [add your blog!](#) [Contact us](#) [RSS](#)


WELCOME!

Here you will find daily news and tutorials about R, contributed by over 450 bloggers. You can subscribe for e-mail updates:

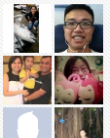
[Subscribe](#)

13100 readers
BY FEEDBURNER

And get updates to your Facebook:

 [R bloggers](#) [좋아요](#)

6,849명이 R bloggers을(를) 좋아합니다.



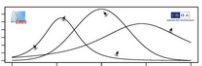
If you are an R blogger yourself you are invited to add your own R content feed to this site (**Non-English** R bloggers should add themselves - [here](#))

POPULAR SEARCHES

[hadoop](#)
[ggplot](#)
[heatmap](#)

MCMSki IV, Jan. 6-8, 2014, Chamonix (news #5)

April 15, 2013
By xi'an



More exciting news about MCMSki IV! First thing first, the 16 contributed sessions are now all-set, having gotten the stamp of approval from the scientific committee! Thanks to everyone who submitted a session proposal. (There were so many proposals that we also had to reject some, as well as every single talk proposal... Sorry people:

[Read more](#)

Bioconductor looking for Google Summer of Code Applicants

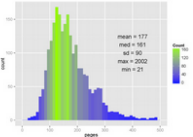
April 15, 2013
By Tal Galili

On April 8, Google announced the 177 mentoring organizations accepted for 2013's Google Summer of Code program. We're proud that Bioconductor is one of the organizations chosen. Google Summer of Code is a global program that...

[Read more](#)

How long is the average dissertation?

April 15, 2013
By beckmw




The best part about writing a dissertation is finding clever ways to procrastinate. The motivation for this blog comes from one of the more creative ways I've found to...

[Read more](#)

TOP 7 ARTICLES OF THE WEEK

1. Scatterplots
2. A quick introduction to ggplot2
3. Visualize large data sets with the bigvis package
4. Select operations on R data frames
5. Tip: Julia vs. R - introduction videos and more
6. R 3.0.0 is released! (what's new, and how to upgrade)
7. A few lists for data scientists and statisticians


SPONSORS



MANGOSOLUTIONS
data analysis that delivers

mango-solutions.com

Beginner's Guide to Generalized Additive Models with R



Alain F Zuur