R 프로그래밍 #2

2019. 03. 13

한국생명공학연구원
김하성

https://github.com/greendaygh/Rprog2019

In the last class

- R, RStudio installation
- RStudio interface
- Keyboard short cuts

```
# make directory C:\Rprog\02
setwd(" C:/Rprog/02 ")
```

```
> gene1_expression <- 2
> gene1_expression <- c(2, 4, 5, 6, 9, 10)
> gene1_expression [c(1,2)]
```

It’s slash
not backslash

- variable, value
- variable type
- vector, vectorized function
- Help

In the last class

- Define a function

```
my_sine <- function(x){
            y <- sin(x)
            return(y)
}
```

- Load once (Ctrl + Enter)
- Use

```
> my_sine(pi)
```

- This returns the sine of pi
    - one parameter: x
    - one argument: pi

Exercise 1

There are four persons named “John”, “James”, “Sara”, “Lilly” and their ages are 21, 55, 23, 53. Let’s build a function that prints every name who’s age is above 50.

- Create a variable named ‘ages’ to save the values
- Use ‘names’ function to assign the names of the values
- Build a function named ‘who’ with one parameter named ‘input’
    - Use ‘which(input>50)’ to find indexes of the hit names
    - Use ‘names’ to extract the names of the values
    - Store the hit names in a variable named ‘greater_then_fifty’
    - Return the variable ‘greater_then_fifty’
- Use the function
    - Pass ‘ages’ to the parameter of the function

Object - Vector

- Basic data structure in R
  - Numeric vector
  - Logical vector
  - Character vector
- Use ‘class’ function

Numeric vector

- Convenient functions for generating structured data

```
x <- c(10.4, 5.6, 3.1, 6.4, 21.7)
class(x)

1:5
1:length(ages)
seq(1,5, by=1)
seq(0, 100, by=10)
seq(0, 100, length.out=11)
?seq

rep(5, times=10)
rep(1:3, times=4)

length(x)
test_scores <- c(Alice = 87, Bob = 72, James= 99)
names(test_scores)
```

Logical vector

```
is.na(1)
is.numeric(1)
is.logical(TRUE)

x > 13
temp <- x > 13
class(temp)

which(ages < 30)
any(ages < 30)
all(ages < 30)
```

Character vector

```
x <- c("X1", "Y2", "X3", "Y4", "X5")

paste("X", 1:5, sep="")
paste("X", "Y", "Z", sep="_")
paste(c("Four","The"), c("Score","quick"), c("and","fox"), sep="_")
paste(c("X","Y"), 1:10, sep="")

?sample
sample(1:10, 3)
sample(x, 3)
sample(x, 20)
sample(x, 20, replacement=T)
```

NA, NULL, NaN, Inf

- NA: Not available, The value is missing
- NULL: a reserved value
- NaN: Not a number (0/0)
- Inf: (1/0)

```
hip_cost <- c(10500, 45000, 74100, NA, 83500)
sum(hip_cost)
sum(hip_cost, na.rm=TRUE)
?sum
```

Some useful functions

```
z <- sample(1:10, 100, T)
head(z)
sort(z)
order(z)
table(z)
p <- z/sum(z)
round(p, digits=1)
digits <- as.character(z)
n <- as.numeric(digits)
d <- as.integer(digits)
```

Exercise 2

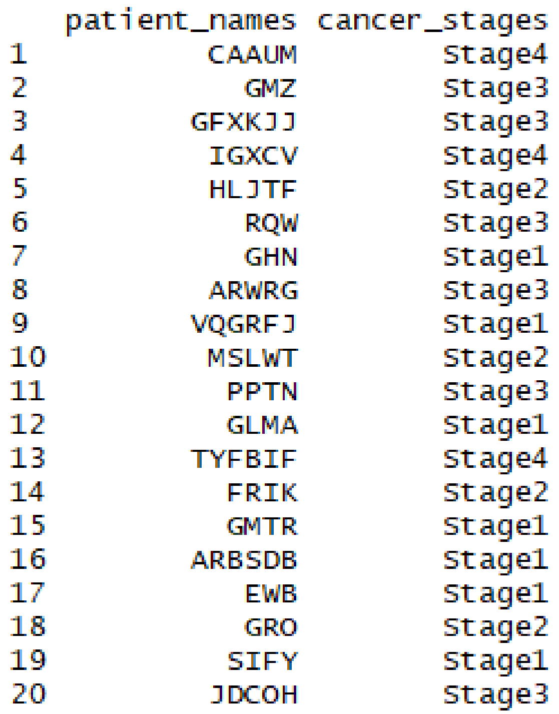
- Use sample() function to generate 100 values ranging between 1 to 100 with replacement, and save the values to ‘ages’
- Use paste() function to generate “X_1”, “X_2” …, “X_100” character vectors and use names() function to assign the names on the values in ages variable.
- From the ‘who’ function of exercise 1, use one more argument named “cri” which can be used as a criterion of the age 50 in the exercise 1.
- Execute who(ages, 20) and who(ages, 30)

* Can you find the people whose age is in between 20 and 30

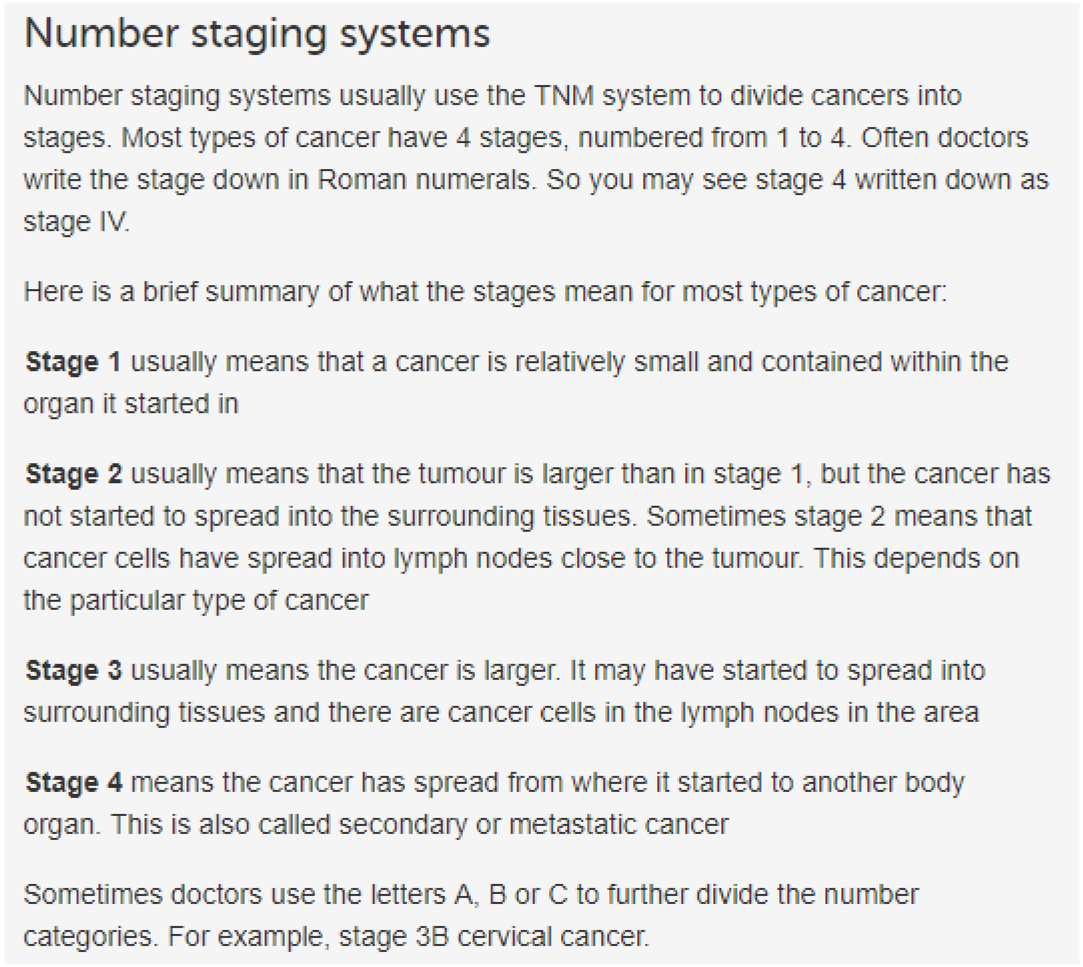
Categorical data

Positive numbers: 1,2,3,5,8,9,…

Years: 1964, 1965, 1966, 1967, …



```
   patient_names cancer_stages
1          CAAUM        Stage4
2            GMZ        Stage3
3         GFXKJJ        Stage3
4          IGXCV        Stage4
5          HLJTF        Stage2
6            RQW        Stage3
7            GHN        Stage1
8          ARWRG        Stage3
9         VQGRFJ        Stage1
10         MSLWT        Stage2
11          PPTN        Stage3
12          GLMA        Stage1
13        TYFBIF        Stage4
14          FRIK        Stage2
15          GMTR        Stage1
16        ARBSDB        Stage1
17           EWB        Stage1
18           GRO        Stage2
19          SIFY        Stage1
20         JDCOH        Stage3
```



Number staging systems

Number staging systems usually use the TNM system to divide cancers into stages. Most types of cancer have 4 stages, numbered from 1 to 4. Often doctors write the stage down in Roman numerals. So you may see stage 4 written down as stage IV.

Here is a brief summary of what the stages mean for most types of cancer:

Stage 1 usually means that a cancer is relatively small and contained within the organ it started in

Stage 2 usually means that the tumour is larger than in stage 1, but the cancer has not started to spread into the surrounding tissues. Sometimes stage 2 means that cancer cells have spread into lymph nodes close to the tumour. This depends on the particular type of cancer

Stage 3 usually means the cancer is larger. It may have started to spread into surrounding tissues and there are cancer cells in the lymph nodes in the area

Stage 4 means the cancer has spread from where it started to another body organ. This is also called secondary or metastatic cancer

Sometimes doctors use the letters A, B or C to further divide the number categories. For example, stage 3B cervical cancer.

https://www.cancerresearchuk.org/

Object - factor

- Made from a character vector
- factor function
- levels are a list of all possible categories
- Default is the collection of unique values

```
n <- 20
x <- sample(1:4, n, replace=T)
cancer_stages <- paste("Stage", x, sep="")
class(cancer_stages)
cancer_stages <- factor(cancer_stages)
class(cancer_stages)
levels(cancer_stages)
cancer_stages[1]
cancer_stages[1] <- "stage5"
levels(cancer_stages) <- c(levels(cancer_stages), "stage5")
cancer_stages[1] <- "stage5"
cancer_stages
```

Exercise 3

1. Use sample() function to generate 10 random numbers ranging from 0 to 100
2. Save the numbers to a matrix variable named “group1”
3. Repeat step1 four times and save each group of the numbers to variables named “group2”, “group3”, “group4”, and “group5”
4. use cbind() or rbind() to generate matrix and save it to “ages”

Object - Arrays and matrices

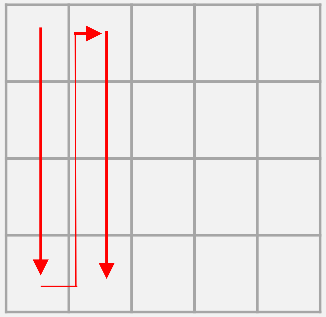
An array is a multiply subscripted collection of data entries
A matrix is a 2-dimensional array

```
x <- c(1:10)
class(x)
dim(x)

x <- array(1:10, dim=c(10))
class(x)
dim(x)

x <- array(1:20, dim=c(4, 5))
class(x)
dim(x)

matrix(1:20, nrow=4, ncol=5)
```



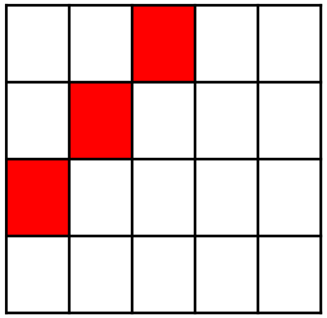
Index matrix

```
x <- matrix(1:20, nrow=4, ncol=5)
```

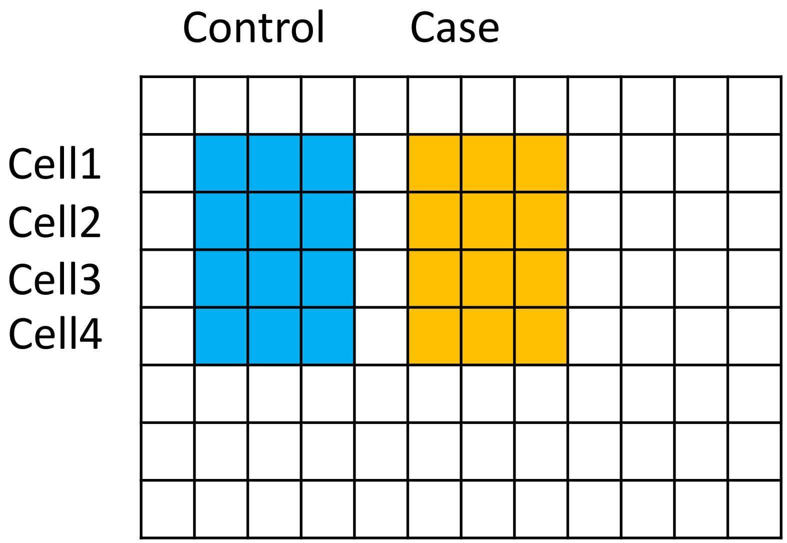
Element in 1st row and 2nd column

```
x[1,2]
x[c(1,2)]
```

Ex: Extract elements x[1,3], x[2,2], and x[3,1] , and Replace these entries in the array x by zeros



Exercise 4 – 96 well plate read



Data generation

- Create a matrix with the 12 by 8 with 0
- Generate 12 random numbers for control
- Generate 12 random numbers for case
- Generate a set of index for control/case
- Assign the control/case values to the index values of the matrix
- Write the data on a file

Write data

```
myval<-matrix(0, nrow=8, ncol=12)
myval

control_values <- sample(1:100, 12)
row_idx <- rep(2:5, 3)
col_idx <- rep(c(2, 3, 4), each=4)
control_idx <- matrix(c(row.idx, col.idx), ncol=2)
myval[control_idx] <- control_values

?write.table
write.table(myval, file="my96well.csv", quote=F, row.names=F,
col.names=F, sep=",")
```

Read data

```
myval <- read.table(file=" my96well.csv ", sep=",")

row_idx <- rep(2:5, 3)
col_idx <- rep(c(2, 3, 4), each=4)
control_idx <- matrix(c(row_idx, col_idx), ncol=2)

m <- myval[control_idx]
```

The apply function

- For matrices, vectorized functions are applied to each element

```
# a common pattern to generate matrix
rowSums(m)
colSums(m)
apply(m, 1, mean)
apply(m, 2, mean)
sapply(m, sum)

?sweep
sweep(m, 1, 10)
sweep(m, 1, 10, "+")
sweep(m, 1, 10, "/")
```

Object - Lists

An R list is an object consisting of an ordered collection of objects known as its components

A list could consist of a numeric vector, a logical value, a matrix, a complex vector, a character array, a function, and so on

```
Lst <- list(name="Fred", wife="Mary", no.children=3, child.ages=c(4,7,9))
Lst
```

Components are always *numbered* and may always be referred to as such

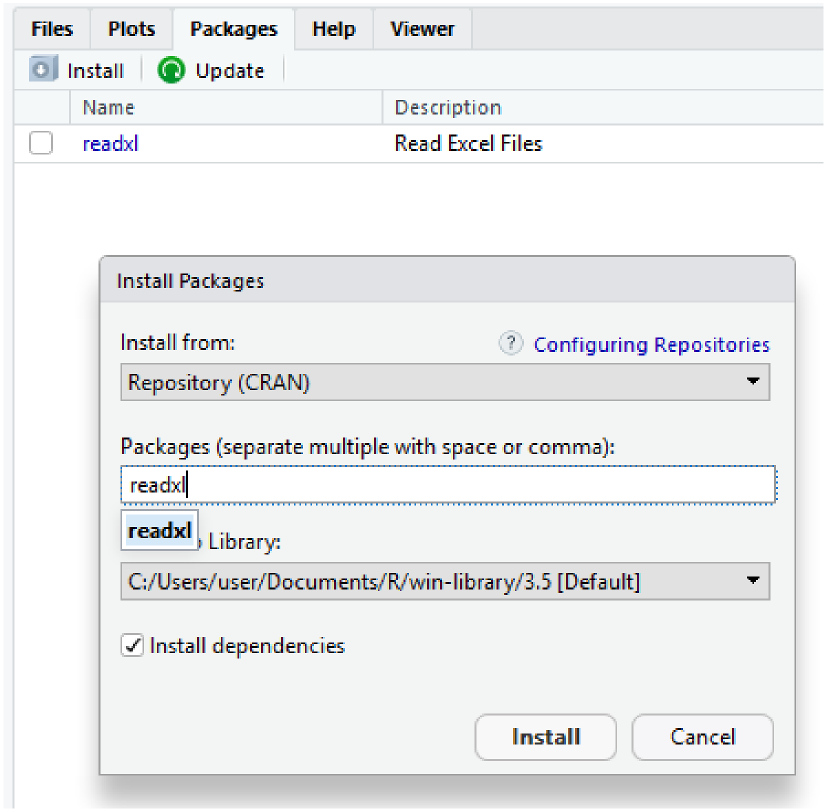
```
Lst[[1]]
Lst[[2]]
Lst[[4]][1]
Lst$child.ages
Lst$child.ages[1]
names(Lst)
names(Lst)[1] <- "my.name"
Lst
Lst[[5]] <- matrix(1:20, c(4,5))
names(Lst)[5] <- "my.matrix"
```

Data frames

A data frame is a list with class "data.frame".
The components must be vectors (numeric, character, or logical), factors, numeric matrices, lists, or other data frames with the *same length*

```
L3 <- LETTERS[1:3]
fac <- sample(L3, 10, replace = TRUE)
z <- data.frame(x = 1, y = 1:10, fac = fac)
z
class(z)
class(z[,1])
class(z[,2])
class(z[,3])
z$x
z$y
```

Installing packages (CRAN)



Name	Description
readxl	Read Excel Files

Install Packages

Install from: Repository (CRAN)

Configuring Repositories

Packages (separate multiple with space or comma): readxl

Install to Library: C:/Users/user/Documents/R/win-library/3.5 [Default]

✓ Install dependencies

readxl

```
install.packages(readxl)
```

Read excel file

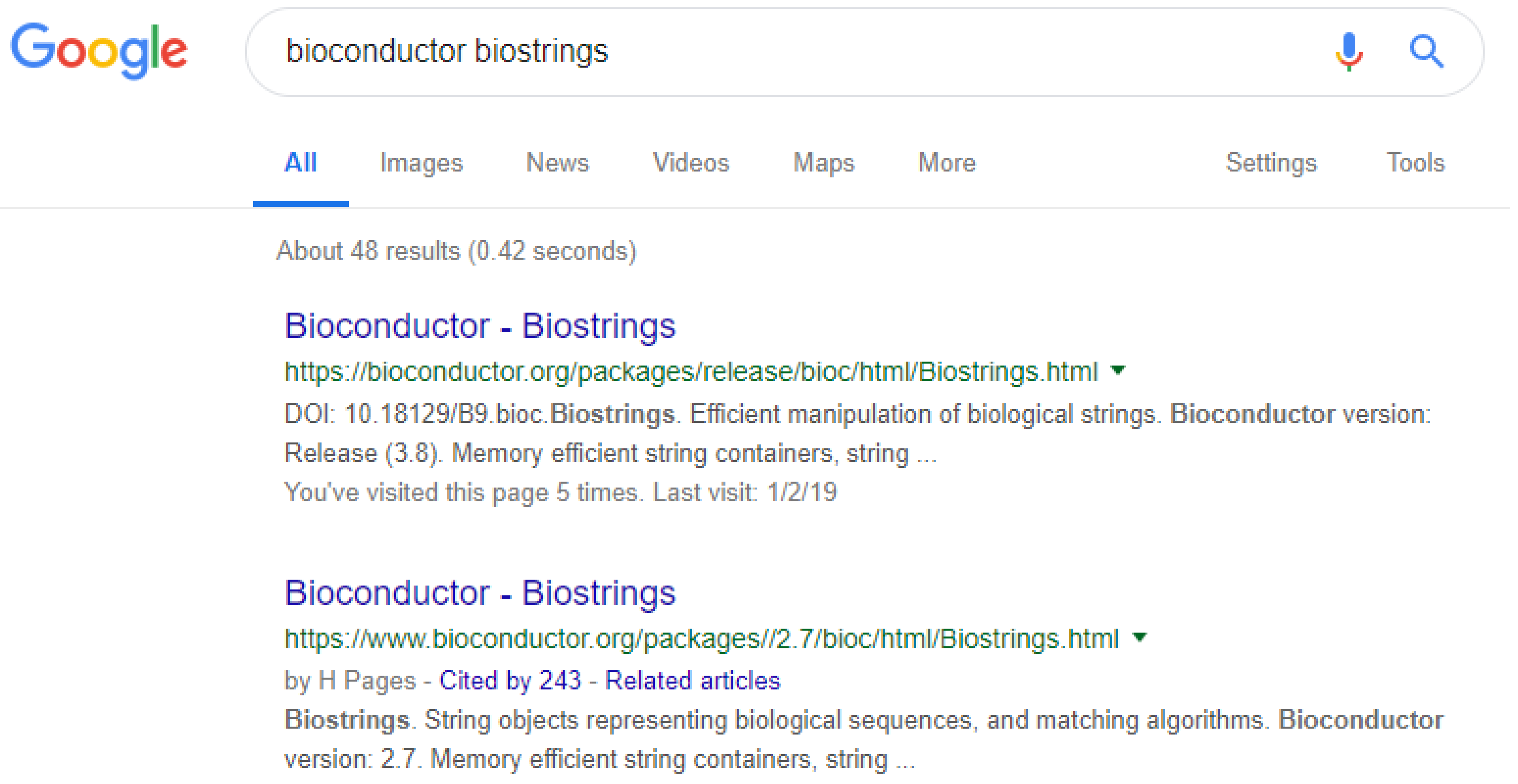
- Install readxl package
- Download Rprog04-fl.xls

  from https://github.com/greendaygh/Rprog2019

Copy to the working directory

```
library(readxl)
# download
f <-  " Rprog04-fl.xls "
fl <- read_excel(f, sheet = 2, skip=2)
```

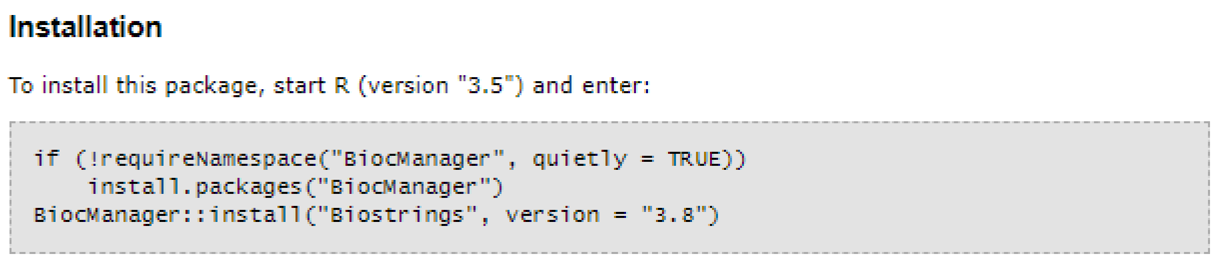
Installing packages (Bioconductor)



About 48 results (0.42 seconds)

Bioconductor - Biostrings
https://bioconductor.org/packages/release/bioc/html/Biostrings.html
DOI: 10.18129/B9.bioc.**Biostrings**. Efficient manipulation of biological strings. **Bioconductor** version: Release (3.8). Memory efficient string containers, string ...
You've visited this page 5 times. Last visit: 1/2/19

Bioconductor - Biostrings
https://www.bioconductor.org/packages//2.7/bioc/html/Biostrings.html
by H Pages - Cited by 243 - Related articles
Biostrings. String objects representing biological sequences, and matching algorithms. **Bioconductor** version: 2.7. Memory efficient string containers, string ...



Installation

To install this package, start R (version "3.5") and enter:

```
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
BiocManager::install("Biostrings", version = "3.8")
```

Next

- R basics II
    - Matrix, Data.frame, List
    - if, if else, for, break
    - Data manipulation