R Basic

http://cbb.sjtu.edu.cn/~jingli/courses/2017fall/bi372/

Dept of Bioinformatics & Biostatistics, SJTU

Background

- Statistical software
  - SAS,
  - SPSS, Stata, Minitab
  - Excel
  - R
- Why should you use R?
  - Not only R is free, but it's also open-source
  - It runs on a variety of platforms including Windows, Unix and MacOS
  - R allows you to integrate with other languages (C/C++, Java, Python)
  - Bioconductor (for omics data)

History of R

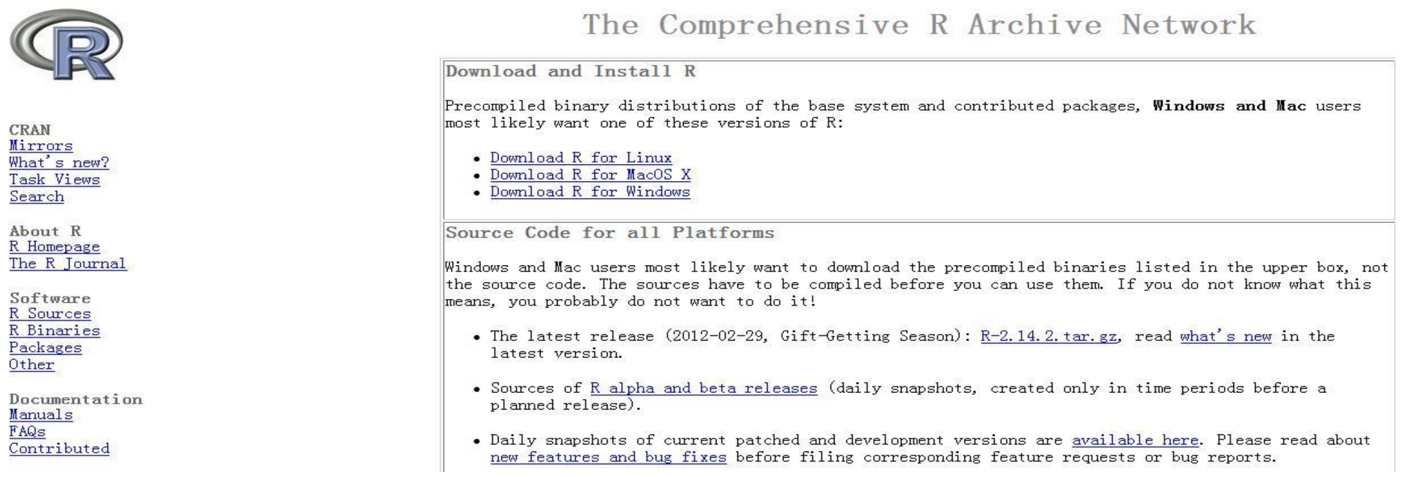
- S programming language (Scheme, Steele and Sussman, MIT Lab, 1970)
- Use S to develop statistical and graphical tools (Chambers and Allan, AT&T, 1980)
- S-PLUS is a commercial implementation of the S (Statistical

Sciences, Inc., 1988)

- R is an implementation of S (Ross Ihaka and Robert Gentleman, 1995)
- The Comprehensive R Archive Network, CRAN (1977)

R software

- Home page: http://www.r-project.org
- BioConductor: http://www.bioconductor.org
- For Linux/OS X/Windows


The Comprehensive R Archive Network

CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

Documentation
Manuals
FAQs
Contributed

Download and Install R

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

- Download R for Linux
- Download R for MacOS X
- Download R for Windows

Source Code for all Platforms

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

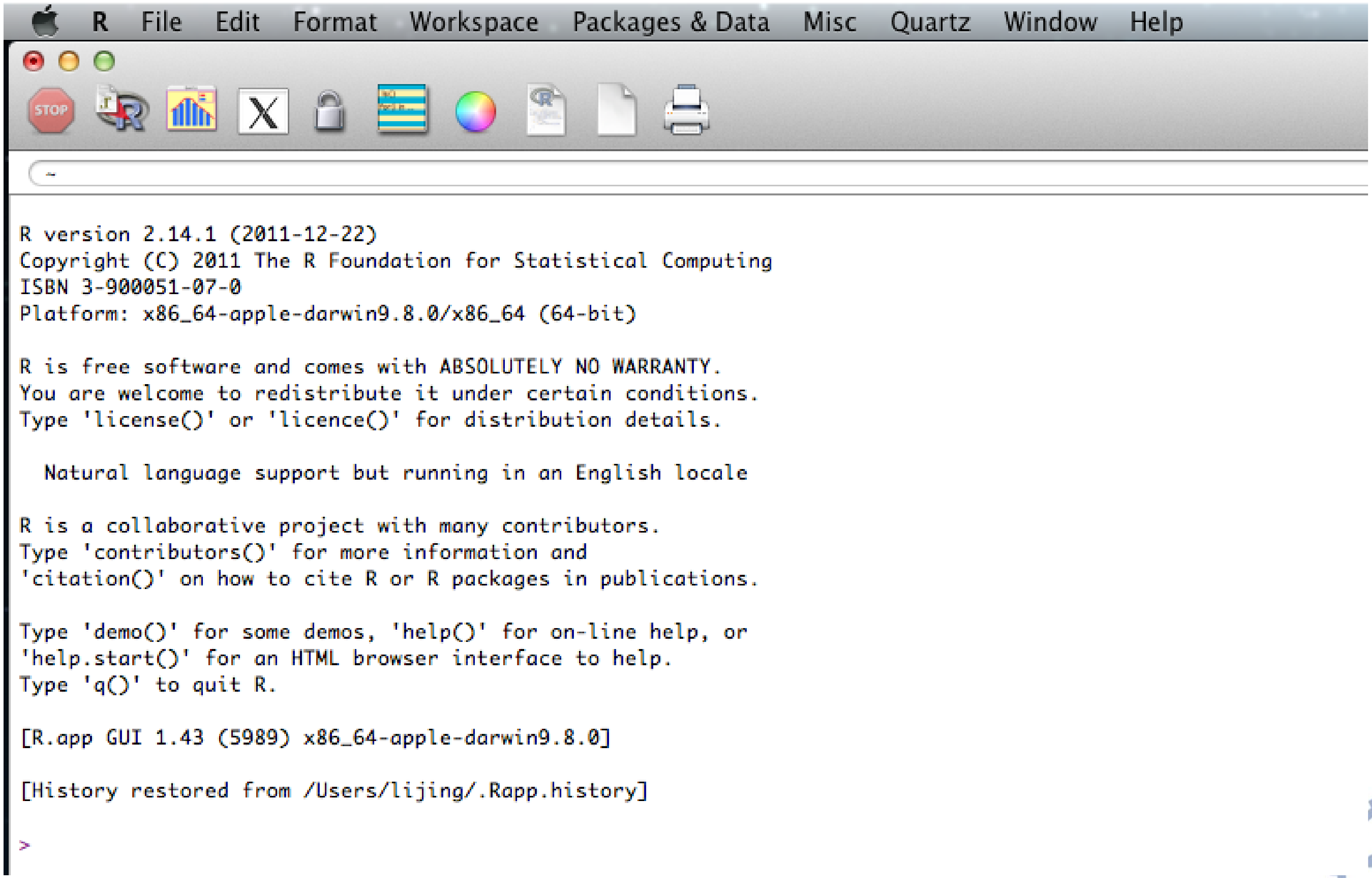
- The latest release (2012-02-29, Gift-Getting Season): R-2.14.2.tar.gz, read what's new in the latest version.
- Sources of R alpha and beta releases (daily snapshots, created only in time periods before a planned release).
- Daily snapshots of current patched and development versions are available here. Please read about new features and bug fixes before filing corresponding feature requests or bug reports.

Install R

Installation for Windows

1. go the the R website (http://www.R-project.org)
2. click on Download R on the left and choose a mirror site geographically near to you
3. choose Download R for Windows and click on base
4. click on Download R 2.15.x for Windows and save it (an .exe file) on your computer
5. double-click on this to run the installation

R GUI



```
R version 2.14.1 (2011-12-22)
Copyright (C) 2011 The R Foundation for Statistical Computing
ISBN 3-900051-07-0
Platform: x86_64-apple-darwin9.8.0/x86_64 (64-bit)

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.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

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

[R.app GUI 1.43 (5989) x86_64-apple-darwin9.8.0]

[History restored from /Users/lijing/.Rapp.history]

>
```

R Environment

- Code Editor for R

  basic code editors provided by R

  Tinn-R (http://www.sciviews.org/Tinn-R/)

  RStudio (http://rstudio.org/)

R Introduction

You can enter commands one at a time at the command prompt (>) or run a set of commands from a source file. There is a wide variety of data types, including vectors (numerical, character, logical), matrices, dataframes, and lists.

To quit R, use

```
>q()
```

R Introduction

A key skill to using **R** effectively is learning how to use the built-in help system. Other sections describe the working environment, inputting programs and outputting results, installing new functionality through packages and etc.

A fundamental design feature of **R** is that the output from most functions can be used as input to other functions. This is described in reusing results.

R Introduction

- These objects can then be used in other calculations. To print the object just enter the name of the object. There are some restrictions when giving an object a name:
  - Object names cannot contain `strange' symbols like !, +, -, #.
  - A dot (.) and an underscore ( ) are allowed, also a name starting with a dot.
  - Object names can contain a number but cannot start with a number.
  - R is case sensitive, X and x are two different objects, as well as temp and temP.

An example

```
> x <- c(1:10)
> x[(x>8) | (x<5)]
> # yields 1 2 3 4 9 10
> # How it works
> x <- c(1:10)
> X
>1 2 3 4 5 6 7 8 9 10
> x > 8
> F F F F F F F F T T
> x < 5
> T T T T F F F F F F
> x > 8 | x < 5
> T T T T F F F F T T
> x[c(T,T,T,T,F,F,F,F,T,T)]
> 1 2 3 4 9 10
```

R Introduction

```
> x = sin(9)/75
> y = log(x) + x^2
> x
[1] 0.005494913
> y
[1] -5.203902
> m <- matrix(c(1,2,4,1), ncol=2)
> m
> [,1] [,2]
[1,] 1 4
[2,] 2 1
```

R Workspace

Objects that you create during an R session are hold in memory, the collection of objects that you currently have is called the workspace. This workspace is not saved on disk unless you tell R to do so.

This means that your objects are lost when you close R and not save the objects, or worse when R or your system crashes on you during a session.

R Workspace

When you close the RGui or the R console window, the system will ask if you want to save the workspace image. If you select to save the workspace image then all the objects in your current R session are saved in a file .RData.

This is a binary file located in the working directory of R, which is by default the installation directory of R.

R Workspace

```
getwd() # print the current working directory

ls()  # list the objects in the current workspace
setwd(mydirectory)   # change to mydirectory
setwd("c:/docs/mydir")
```

R Help

Once R is installed, there is a comprehensive built-in help system. At the program's command prompt you can use any of the following:

help.start() # general help
help(*foo*) # help about function *foo*
?*foo* # same thing
apropos("*foo*") # list all function containing string foo
example(*foo*) # show an example of function *foo*

R Datasets

R comes with a number of sample datasets that you can experiment with. Type

```
> data( )
```

to see the available datasets. The results will depend on which packages you have loaded. Type

```
>help(datasetname)
```

for details on a sample dataset.

R Packages

- R packages are collections of R functions, data, and compiled code in a well-defined format. The directory where packages are stored is called the library.

  >library() # to see which packages are installed

- When you start R not all of the downloaded packages are attached, only seven packages are attached to the system by default. You can use the function search to see a list of packages that are currently attached to the system.

  > search() # to see which packages are currently loaded

R Packages

- To attach another package to the system you can use the menu or the library function.

Via the menu: select the `Packages' menu and select `Load package...', a list of available packages on your system will be displayed. Select one and click `OK', the package is now attached to your current R session.

Via the library function:

```
>library(MASS)
>help(package="MASS")
> shoes
$A
[1] 13.2 8.2 10.9 14.3 10.7 6.6 9.5 10.8 8.8 13.3
$B
[1] 14.0 8.8 11.2 14.2 11.8 6.4 9.8 11.3 9.3 13.6
```

R Packages

```
> .libPaths() # to get the library location
> install.packages() # to install package
> update.packages() # to update package
```

For example, we want to install a new package vioplot

1. install.packages()
2. select a CRAN Mirror
3. library()

Source Codes

you can have input come from a script file (a file containing R commands) and direct output to a variety of destinations.

The source( ) function runs a script in the current session. If the filename does not include a path, the file is taken from the current working directory.

```
# input a script
source("myfile")
```

Data input &output

- Importing Data
- Keyboard Input
- Database Input
- Exporting Data
- Viewing Data
- Data Type

From A Comma Delimited Text File

```
# first row contains variable names, comma is separator
# assign the variable id to row names
# note the / instead of \ on mswindows systems

mydata <- read.table("c:/mydata.csv", header=TRUE, sep=",", row.names="id")
```

```
x<-scan()  get data from pasteborad
```

From Excel

The best way to read an Excel file is to export it to a comma delimited file and import it using the method above.

On windows systems you can use the **RODBC** package to access Excel files. The first row should contain variable/column names.

```
# first row contains variable names
# we will read in workSheet mysheet
>library(RODBC)
>channel <- odbcConnectExcel("c:/myexel.xls")
>mydata <- sqlFetch(channel, "mysheet")
>odbcClose(channel)
```

Keyboard Input

You can also use **R**'s built in spreadsheet to enter the data interactively, as in the following example.

```
# enter data using editor
>mydata <- data.frame(age=numeric(0),
gender=character(0), weight=numeric(0))
>mydata <- edit(mydata)
```

Output

The sink( ) function defines the direction of the output.

```
# direct output to a file
sink("myfile", append=FALSE, split=FALSE)

# return output to the terminal
>sink()
```

Output

The append option controls whether output overwrites or adds to a file.

The split option determines if output is also sent to the screen as well as the output file.

Here are some examples of the sink() function.

```
# output overwrites existing file. no output to terminal.
>sink("myfile.txt", append=TRUE, split=TRUE)
```

Exporting Data

To A Tab Delimited Text File

```
>write.table(mydata, "c:/mydata.txt", sep="\t")
```

To an Excel Spreadsheet

```
>library(xlsReadWrite)
>write.xls(mydata, "c:/mydata.xls")
```

Viewing Data

There are a number of functions for listing the contents of an object or dataset.

```
# list objects in the working environment
ls()
# list the variables in mydata
names(mydata)
# list the structure of mydata
str(mydata)
# list levels of factor v1 in mydata
levels(mydata$v1)
# dimensions of an object
dim(object)
```

Object in R

- List the objects in current session:

```
> ls() # or objects()
> rm(x)
> rm(list=ls())
> q() # or quit() to exit
Save the current images? yes? no? cancel?
> save(x, file="x.RData")
> load(file="x.RData")
```

Object in R

- Type of object
    - Vector, Matrix
    - Factor
    - List
    - Dataframe

- Class of an object
    - Numeric
    - String/Character
    - Logical
    - Function

Vectorized Arithmetic

– We can do little statistics with a single number!
– we need a way to store a sequence/list of numbers
– One can simply concatenate elements with c function

```
> weight <- c(60,72,75,90,95,72)
> weight
[1] 60 72 75 90 95 72
> weight[1]
[1] 60
> height <- c(175, 180,163,156,171,149)
> bmi <- weight/height^2
```

Vectors

We have 3 types of vectors: numeric, logical, character

```
# Numeric vectors
> numVec <- c(1,5,8)
> x
[1] 1 5 8
#logical vectors
> logVec <- c(TRUE, TRUE, FALSE, TRUE)
> logVec
[1] TRUE TRUE FALSE TRUE
# Character vectors
> charVec <- c(“Hello”, “my”,”name”,”is”,”Ricky”)
> charVec
[1] “Hello”   “my”   “name”   “is”   “Ricky”
```

Missing and Special values

- In R, missing data are denoted by NA
- NaN – Not a number
- -Inf, Inf

- R has provided different ways to deal with missing data, like omitting, imputing, etc.

```
> weight <- c(60,72,75,90,NA,72)
> mean(weight)
[1] NA
> mean(weight, na.rm=TRUE)
[1] 73.8
```

Matrices and arrays

- A matrix is a 2-D array of numbers
- Matrices can be used to perform statistical operations (linear algebra).
- Matrices can be used to store tables

```
> X <- 1:12
[1] 1 2 3 4 5 6 7 8 9 10 11 12
> length(X)
[1] 12
> dim(X)
[1] NULL
> dim(X) <- c(3,4)
> X
        [,1]   [,2]   [,3]   [,4]
[1,]    1       4      7      10
[2,]          2       5      8      11
[3,]          3       6      9      12
> X <- matrix(1:12, nrow=3, byrow=FALSE)
> rownames(X) <- c("A", "B", "C")
> X
        [,1]   [,2]   [,3]   [,4]
A       1       4      7      10
B       2       5      8      11
C       3       6      9      12
> colnames(X) <- c('1','2','x','y')
> X
```

Matrices and Arrays

– Matrices can also be formed by "glueing" rows or columns using *rbind* or *cbind* functions.

```
> x1 <- 1:4; x2 <- 5:8
> y1 <- c(3,9)
> myMatrix <- rbind(x1, x2)
> myMatrix
      [,1] [,2] [,3] [,4]
x1    1    2    3    4
x2    5    6    7    8
> myNewMatrix <- cbind(myMatrix, y1)
> myNewMatrix
                         y1
x1    1    2    3    4    3
x2    5    6    7    8    9
```

Factors

- It is common to have *categorical data* in statistical data analysis (e.g. Male/Female).
- In R such variables are referred to as *factors*
- A factor has a set of *levels*

```
> pain <- c(0,3,3,2,2,1)
> fpain <- as.factor(c(0,3,2,2,1))
> levels(fpain) <- c("none", "mild", "medium", "severe")
> is.factor(fpain)
[1] TRUE
> is.vector(fpain)
[1] FALSE
```

Lists

– Lists can be used to combine objects of possibly different kinds/sizes into a large composite object

– The components of the list are named according to the arguments used

– Named components can be accessed with the $ sign

```
> x <- c(31,32,40)
> y <- as.factor(c(“F”, “M”, “M”)
> z <- c(“London”, “New York”,
“Shanghai”)
> Persons <- list(age=x, gender=y, loc=z)
> Persons
$age
[1] 31 32 40

$gender
[1] F M M

$loc
[1] “London” “New York”  “Shanghai”

> Persons$age
[1] 31 32 40
```

Data.frame

- DFs are a list of vectors and/or factors of the same length that are related “across”
- Each row comes from a unique object (e.g., a person, experiment, etc.)
- Each column is of the same data type
- More storage-efficient and indexing-efficient than simple lists

```
> MyDataFrame <- data.frame(age=c(31,32,40), sex=y)
> MyDataFrame
> MyDataFrame$age
[1] 31 32 40
> is.vector(MyDataFrame$age)
[1] TRUE
> is.vector(MyDataFrame$sex)
[1] FALSE
```

Names

- Names of an R object can be accessed and/or modified with ‘names’ function (method)
- Names can be used for indexing
- So remember to give explicit names to variables

```
> x <- 1:3
> names(x)
NULL
> names(x) <- c(‘a’, ‘b’, ‘c’)
> persons <- data.frame(age=c(31,32,34), sex=y)
> names(persons)
[1] “age” “sex”
> names(persons) <- c(“age”, “gender”)
> names(persons)[1] <- “Age”
```

Indexing

– Indexing is a great way to directly access elements of interest, for vector, list, matrix, array, and data.frame

```
# Indexing a vector
pain <- c(0,3,2,2,1)
pain[1]
pain[1:2]
pain[c(1,3)]
pain[-5]

\# Indexing a matrix
MyMatrix[1,2]
MyMatrix[1,]
MyMatrix[,1]
MyMatrix[,-2]
```

```
# Indexing a list
MyList[3]
MyList[[3]]
MyList[[3]][1]

# Indexing a
data.frame
MyDataFrame[1,]
MyDataFrame[2,]
```

Functions and arguments

- Many of the R tasks are done using *function calls,* like *log(x), plot(weight, height)*
- If you do want to get help for a function e.g. *plot()*, just type *?plot*
- Most function arguments have sensible default and can thus be omitted, e.g., *plot(weight, height, col=1)*

- If you do NOT specify the names of the argument, the order is very important

R programming

– R is a true programming language.

```
# if statement
x <- -2
if (x >0) {
      print(x)
}
else if (x==0) {
       print(0)
}
else {
       print(-x)
}
```

```
# for-loops
n <- 1e6
x <- rnorm(n,10,1)
y <- x^2
y <- rep(0,n)
for (i in 1:n) {
     y[i] <- sqrt(x[i])
}
# while-loops
count <- 1
while (count<=n) {
       y[count] <- sqrt(x[count])
       count <- count + 1
}
```

Creating your own functions

– As with other programming languages, you can create your own functions

```
testFunc <- function(yourName, myName="Yahoo", number=0)
{
        if (number == 0) {
       return(yourName)
        } else {
       return(myName)
        }
}
testFunc("Google");
testFunc("Baidu", "Facebook", 1)
testFunc(number=1, myName="Twitter",
yourName="Microsoft")
```

Useful Functions

```
length(object) # number of elements or components
str(object)    # structure of an object
class(object)  # class or type of an object
names(object)  # names
c(object,object,...)  # combine objects into a vector
cbind(object, object, ...) # combine objects as columns
rbind(object, object, ...) # combine objects as rows
ls()       # list current objects
rm(object) # delete an object
newobject <- edit(object) # edit copy and save a
newobject
fix(object)               # edit in place
```

Exercise (1)

1. Collect the heights (in cm) and the shoes lengths (in cm) of your classmate (10 students). Save data as .txt file.
2. Using R, load the data. Add the information of one more student to the data frame.
3. Create a vector called ratio (height/shoes length) and add to the data frame.
4. Save the data frame as new.csv

Exercise (2)

```
ndata<- rnorm(10000, mean=200, sd=10)
```

# Generate numbers from a normal distribution

1. Randomly sample 10 numbers from ndata, and calculate the mean. Repeat 10 times

   # sample(); mean()

2. Randomly sample 100 numbers from ndata, and calculate the mean. Repeat 10 times

3. Randomly sample 1000 numbers from ndata, and calculate the mean. Repeat 10 times

4. Perform the same analyses to another dataset :

```
kdata<-rnorm(10000, mean=200, sd=50)
```

5. Save ndata and kdata as .txt files.