Introduction to R
Data import/export; Basic operations

2007 R 統計軟體研習會
蔡政安

Associate Professor
Department of Public Health & Biostatistics Center
China Medical University

What is R?

- R is an open-source software package especially suitable for professional data analysis and graphical display. R (http://cran.at.r-project.org)
- Flexible and highly customizable by users .
- Provide a great variety of packages for statistical analysis in finance, environment, and different life science areas.
- Functions and results of analysis are stored as objects.

Disadvantages of R

- R is not efficient in handling large data sets.
- Slow computation for a large number of do-loops, compared to C/C++, Fortran etc.
- Self-Learning is not so convenient compared to “point-and-click” statistics software.
- No warrantee and informal support.
- Needed to upgrade R version to install some newly developed packages.

Where do I get R?

- There are versions for Unix, Windows, and Mac OS. All of them are free
- Download website: http://cran.csie.ntu.edu.tw/, and follow the instructions.
- Installation of R software and packages.

  install.packages("e1071")

  install.packages("graphics")

  install.packages("ade4")

Getting Help in R

- **library()** #lists all available libraries on system
- **help(command)** #getting help for one command, e.g. help(heatmap)
- **help.search("topic")** #searches help system for packages associated with the topic, e.g. help.search("distribution")
- **help.start()** #starts local HTML interface
- **q()** #quits R console

Basic Usage of R

- The general R command syntax uses the assignment operator “<-” (or “=“) to assign data to object.

  object <- function (arguments)

- **source(“myscript.R”)**, #command to execute an R script named as myscript.R.
- **objects()** or **ls()** , # list the names of all objects
- **rm(data1)**, #Remove the object named data1 from the current environment
- **data1 <- edit(data.frame())** # Starts empty GUI spreadsheet editor for manual data entry.

Basic Usage of R

- **class(object)** #displays the object type.
- **str(object)** #displays the internal type and structure of an R object.
- **attributes(object)** #Returns an object's attribute list.
- **dir()** # Reads content of current working directory.
- **getwd()** # Returns current working directory.
- **setwd("/home/user")** # Changes current working directory to user specified directory.

Data Import

- **read.delim("clipboard", header=T)** # Command to copy&paste tables from Excel or other programs into R. If the 'header' argument is set to FALSE, then the first line of the data set will not be used as column titles.
- **scan("my_file")** # reads vector/array into vector from file or keyboard.
- **my_frame <- read.delim(“c://Affymetrix/affy1.txt", na.strings = "", fill=TRUE, header=T, sep="\t")** # The function read.delim() is often more flexible for importing tables with empty fields and long character strings (e.g. gene descriptions).
  - It supports data import on the web.
  - Different coding of missing values (na.strings=“NA” or “ ”).
  - Data columns can be separated by TAB, comma, or semicolon (sep=“”).

Data Import

- There are some alternatives for reading data as followings.
- **my_frame <- read.table(file="path", header=TRUE, sep="\t")** #Reads in table with info on column headers and field separators.

  data<-read.table("http://www.cmu.edu.tw/example.txt", header=TRUE)
- **my_frame <- read.csv(file="path", header=TRUE)** # reads .csv file with comma separated value.
- You can skip lines, read a limited number of lines, different decimal separator, and more importing options.
- The foreign package can read files from Stata, SAS, and SPSS.

Data Export

- **write.table(iris, "clipboard", sep="\t", col.names=NA, quote=F)** # Command to copy&paste from R into Excel or other programs. It writes the data of an R data frame object into the clipbroard from where it can be pasted into other applications.
- **write.table(dataframe, file="file path", sep="\t", col.names = NA)** # Writes data frame to a tab-delimited text file. The argument 'col.names = NA' makes sure that the titles align with columns when row/index names are exported (default).
- **write(x, file="file path")** # Writes matrix data to a file.
- **sink("My_R_Output")** # redirects all subsequent R output to a file 'My_R_Output' without showing it in the R console anymore.

  sink() # restores normal R output behavior.

Data Types

- Numeric data: 1, 2, 3

  x <- c(1, 2, 3); x; is.numeric(x); as.character(x) # Creates a numeric vector, checks for the data type and converts it into a character vector.

- Character data: "a", "b" , "c"

  x <- c("1", "2", "3"); x; is.character(x); as.numeric(x) # Creates a character vector, checks for the data type and converts it into a numeric vector.

- Complex data: 1, b, 3
- Logical data: TRUE, FALSE, TRUE

  1:10 < 5 # Returns TRUE where x is < 5.

Object Types

- vectors: ordered collection of numeric, character, complex and logical values.
- factors: special type vectors with grouping information of its components
- data frames: two dimensional structures with different data types
- matrices: two dimensional structures with data of same type
- arrays: multidimensional arrays of vectors
- lists: general form of vectors with different types of elements
- functions: piece of code

Subsetting Syntax

- **my_object[index]** # Subsetting of one dimensional objects, like vectors and factors. Returns elements with positions in index
- **my_object[row.index, col.index]** # Subsetting of two dimensional objects, like matrices and data frames.
- **my_object[row.index, col.index, dim]** # Subsetting of three dimensional objects, like arrays.
- **dim(my_object)** # Returns the numbers of row and column
- **my_logical <- (my_object > 10)** # Generates a logical vector as example.

  my_object[my_logical] # Returns the elements where my_logical contains TRUE values.
- **my_object$Name1** # Returns the ‘Name1' column in the my_object data frame.

Basic Operators

- Comparison operators
  - equal: ==
  - not equal: **!=**
  - greater/less than: > / <
  - greater/less than or equal: >= <= Example: **1 == 1** # Returns TRUE.

- Logical operators
  - AND: **&**

x <- 1:10; y <- 10:1 # Creates the sample vectors 'x' and 'y'.
x > y & x > 5 # Returns TRUE where both comparisons return TRUE.

  - OR: |

x == y | x != y # Returns TRUE where at least one comparison returns TRUE.

  - NOT: !

!x > y # The '!' sign returns the negation (opposite) of a logical vector.

Basic Operators (Cont'd)

- Calculations
  - Four basic arithmetic functions: addition, subtraction, multiplication and division

    1 + 1; 1 - 1; 1 * 1; 1 / 1 # Returns the results of these calculations.

- Calculations on vectors

  x <- 1:20; sum(x); mean(x), sd(x); sqrt(x) ;rank(x);sort(x) # Calculates for the vector x its sum, mean, standard deviation and square root etc.

  x <- 1:20; y <- 1:20; x + y # Calculates the sum for each element in the vectors x and y.

Vectors Manipulation

- **x <- 1:10; as.character(x)** # The function 'as.character' changes the data mode from numeric to character.
- **as.numeric(character)** # The function 'as.numeric' changes the data mode from character to numeric.
- **d <- as.integer(x); d** # Transforms numeric data into integers.
- **x <- 1:100; sample(x, 5)** # Selects a random sample of size of 5 from a vector.
- **x <- as.integer(runif(100, min=1, max=5)); sort(x); rev(sort(x)); order(x); x[order(x)]** # Generates random integers from 1 to 4. The sort() function sorts the items by size. The rev() function reverses the order. The order() function returns the sorted indices. The latter function is most flexible for sorting vectors, data frames, lists and other objects.
- **x <- rep(1:10, times=2); x; unique(x)** # The unique() function removes the duplicated entries in a vector.

- **paste(LETTERS[1:8], 1:12, sep="")** # The command 'paste' merges vectors after converting to characters. **x <- paste(rep("A", times=12), 1:12, sep="")**

Matrices and Arrays Manipulation

- **x <- matrix(1:30, 3, 10, byrow = T)** # Lays out vector (1:30) in 3 by 10 matrix.
- **dim(x) <- c(3,5,2)** # transforms above matrix into multidimensional array.
- **x <- array(1:25, dim=c(5,5))** # creates 5 by 5 array ('x') and fills it with values 1-25.
- **y <- c(x)** # writes array into vector.
- **array[rows,columns]** # Syntax to access columns and rows of matrices and two-dimensional arrays.
- **as.matrix(iris)** # Convert the object into a matrix.
- **x <- array(c(1:5,5:1),dim=c(5,2))** # Creates 5 by 2 index array (x) and fills it with the values 1-5, 5-1.
- **array[rows,columns,dimensions]** # Syntax to access columns, rows and dimensions in arrays with more than two dimensions.

  x <- array(1:250, dim=c(10,5,5)); x[2:5,3,]

Matrices and Arrays Manipulation

- Appending arrays

  cbind(array1,array2) # Appends columns of arrays with same number of rows.
  rbind(array1,array2) # Appends rows of arrays with same number of columns.

- Calculations between arrays

  Z <- array(1:12, dim=c(12,8)); X <- array(12:1, dim=c(12,8)) # Creates arrays Z and X with same dimension.

  newarray <- Z/X # Divides Z/X elements and assigns result to array ‘newarray'.

  t(my_array) # Transposes 'my_array‘.

- Information about arrays

  nrow(X); ncol(X) # returns number of rows and columns of array 'X'.

- Outer product of two arrays X and Y is the array A with dimension c(dim(X), dim(Y)) .

  X%o%Y; outer(X,Y, FUN=“*”)

Matrix Multiplication

- A*B: the matrix of element by element products for the same size of matrices, A and B.
- A%*%B: the matrix product.
- diag(A), gives the vector of main diagonal entries of A.
- eigen(A) calculates the eigenvalues and eigenvectors of matrix A. The result of this function is a list of two components named values and vectors.
- svd(A) computes the singular-value decomposition of a rectangular matrix A. The result of svd(A) is actually a list of three components named d, u and v, where A=UDV’.

Data Frame Objects

- **my_frame[rows, columns]** # Generic syntax to access columns and rows in data frames.
- **dim(my_frame)** # Gives dimensions of data frame.
- **length(my_frame); length(my_frame$y1)** # Provides number of columns or rows of data frame, respectively.
- **colnames(my_frame); rownames(my_frame)** # Gives column and row names of data frame.
- **row.names(my_frame)** # Prints row names or indexing column of data frame.
- **my_frame[order(my_frame[,2], decreasing=TRUE),]** # Sorts the rows of a data frame by the specified columns, here the 2nd column; for increasing order use 'decreasing=FALSE'. In addition to the 'order()' function, there are: 'sort(x)' for vectors, 'rev(x)' for vector in decreasing order and 'sort.list(x)' for vector sequences.

Data Frame Manipulations

- **data<-data.frame(y1=rnorm(12),y2=rnorm(12),y3=rnorm(12),y4=rnorm(12))** # Creates data frame with vectors 1-12 and 12-1.
- **rownames(data) <- month.name[1:12]** # Assigns row (index) names. These indices need to be unique.
- **names(data) <- c("y4", "y3", "y2", "y1")** # Assigns new column titles.
- **names(data)[c(1,2)] <- c("y3", "y4")** # Changes titles of specific columns.
  new_data <- data.frame(IND=row.names(data), data) # Generates new column with title "IND" containing the row names.
- **new_data[,2:5]; new_data[,-1]** # Different possibilities to remove column(s) from a data frame.

Lists Objects

- Lists are ordered objects with collections of different modes (e.g. numeric vector, array, etc.). A name can be assigned to each list component.
- **my_list <- list(name="Fred", wife="Mary", no.children=3, child.ages=c(4,7,9))** # Example of how to create a list.
- **attributes(my_list); names(my_list)** # Prints attributes and names of all list components.
- my_list[2]; my_list[[2]] # Prints component 2 of list; [[..]] is operator to call single element, while '[..]' is general subscripting operator.
- my_list$wife # Named components in lists can also be called with their name. Command returns same component as 'my_list[[2]]'.
- my_list[[4]][2] # Returns from the fourth list component the second entry.
- length(my_list[[4]]) # Prints the length of the fourth list component.
- my_list$wife <- 1:12 # Replaces the content of the existing component with new content.
- my_list$wife <- NULL # Deletes list component 'wife'.

Some useful functions on R

- **apply(iris[,1:3], 1, mean)** # Calculates the mean values for the columns 1-3 in the sample data frame 'iris'. With the argument setting '1', row-wise iterations are performed and with '2' column-wise iterations.
- **tapply(iris[,4], iris$Species, mean)** # Calculates the mean values for the 4th column based on the grouping information in the 'Species' column in the 'iris' data frame.
- **sapply(x, sqrt)** # Calculates the square root for each element in the vector x. Generates the same result as 'sqrt(x)'.
- **lapply(x, fun)** #Returns a list of the same length as X, each element of which is the result of applying FUN to the corresponding element of X.

```
x <- list(a = 1:10, beta = exp(-3:3), logic = c(TRUE,FALSE,FALSE,TRUE))
lapply(x,mean); lapply(x, quantile, probs = 1:3/4)
```

Some useful functions on R

- **unique()** #function makes vector entries unique:
  unique(iris$Sepal.Length); length(unique(iris$Sepal.Length))

- **table()** #counts the frequencies in a vector. It is the most basic "cluster function":
  my_counts <- table(iris$Sepal.Length) # Counts identical entries in the Sepal.Length column of the iris data set and aligns the counts with the original data frame.

- **%in%** # Returns the intersect between two vectors. In a subsetting context with '**[ ]**', it can be used to intersect matrices, data frames and lists:
  1:10 %in% c(1,3,5,9)
  sstr <- c("c","ab","B","bba","c","@","bla","a","Ba","%")
  sstr[sstr %in% c(letters,LETTERS)]

- **merge()** #Joins data frames based on a common key column:
  frame1 <- iris[sample(1:length(iris[,1]), 30),]; my_result <- merge(frame1, iris, by.x = 0, by.y = 0, all = TRUE); dim(my_result) # Merges two data frames (tables) by common field entries, here row names (by.x=0). To obtain only the common rows, change 'all = TRUE' to 'all = FALSE'. To merge on specific columns, refer to them by their position numbers or their column names (e.g. "Species").

Demo

- data(airquality)
  airquality$Month
  airquality[airquality$Month==5,]
  oz <- airquality[airquality$Month==5,]$Ozone
  mean(oz)
  mean(oz, na.rm=TRUE)
  attach(airquality)
  mean(Ozone, na.rm=TRUE)
  tapply(Ozone, Month, mean, na.rm=TRUE)
  detach()

- airquality[airquality$Month==5 & airquality$Ozone>20,]
  subset(airquality,Month==5 & Ozone>20)

Free Documentation

The R project website http://www.r-project.org has several user contributed documents. These are located at http://cran.R-project.org/other-docs.html.

- StatsRus at http://lark.cc.ukans.edu/ pauljohn/R/statsRus.html is a well done compilation of R tips and tricks.
- “Using R for Data Analysis and Graphics” by John Maindonald.
- “R for Beginners / R pour les debutants” by Emmanuel Paradis.
- “Kickstarting R” compiled by Jim Lemon, is a nice, a short introduction in English.
- “Notes on the use of R for psychology experiments and questionnaires” by Jonathan Baron and Yuelin Li is useful for students in the social sciences and offers a nice, quick overview of the data extraction and statistical features of R.