

Introduction to R

- <http://www.r-project.org/>
- <http://cran.r-project.org/>

Outline

- R history, why R?
- Basic use of R and R help, install Rstudio
- How to give R commands, run R scripts
- R data structures
- Reading and writing data
- Install packages
- Graphics

R project

- R is a free software environment for *statistical computing* and *graphics* (<http://www.r-project.org>)
- ***Bioconductor*** is a software project for the analysis of genomic data (<http://www.bioconductor.org>)
 - Currently works as an expansion to R
 - <http://bioconductor.org/>

R, S and S-plus

- S: an interactive environment for data analysis developed at Bell Laboratories since 1976
 - 1988 - S2: RA Becker, JM Chambers, A Wilks
 - 1992 - S3: JM Chambers, TJ Hastie
 - 1998 - S4: JM Chambers
- Exclusively licensed by *AT&T/Lucent* to *Insightful Corporation*, Seattle WA. Product name: “S-plus”.
- Implementation languages C, Fortran.
- See:
<http://cm.bell-labs.com/cm/ms/departments/sia/S/history.html>
- R: initially written by Ross Ihaka and Robert Gentleman at Dep. of Statistics of U of Auckland, New Zealand during 1990s.

Strengths and Weaknesses

- Strengths
 - Free and Open Source
 - Strong User Community
 - Highly extensible, flexible
 - Implementation of high end statistical methods
 - Flexible graphics and intelligent defaults
- Weakness
 - Steep learning curve
 - Slow for large datasets

Features

- Highly Functional
 - Everything done through functions
 - Strict named arguments
 - Abbreviations in arguments OK (e.g. T for TRUE)
- Object Oriented
 - Everything is an object
 - Can run interactively (C should be compiled)
 - “<-” is an assignment operator
 - “a<- 7” : a gets the value 7

Getting Help in R

- From Documentation:
 - `?plot()`
 - `example("plot")`
- Documents: “Introduction to R”

Packages

- R consists of a core and packages.
- Packages like add-on toolbox
- Packages contain functions that are not available in the core.
- For example, Bioconductor code is distributed as several dozen of packages for R.
 - Software packages
 - Metadata (annotation) packages

Install packages

```
#install packages from CRAN  
>install.packages("corrplot")  
>library("corrplot")  
>corrplot(cor(mtcars), method="circle")
```

Install Bioconductor

```
>source("http://www.bioconductor.org/  
  biocLite.R")  
>biocLite()#install core bioconductor  
>biocLite("AnnotationDbi")# install  
  specific package
```

Load packages

- To use a function in a package, the package needs to be loaded in memory.
- Command for this is `library()`, for example:
`>library(gplots)`
- There are three parts in a command:
 - the command
 - brackets
 - Arguments inside brackets (these are not always present)

Running R

- Directly in the Windowing System (Console)
- **Set working directory**
 - `setwd("~/Desktop/")`
 - CMD #windows console
 - `setwd("/Users/Administrator/Desktop/")`
- Console:
 - `source("filename.R")`
 - copy and paste, per line
 - `Rscript filename.R #Linux`
 - Outputs can be diverted by using
`>sink("filename.Rout")`
`>sink() # will restore the console`

Quitting R

- Use command `q()` or menu choose File->Exit.
- R asks whether to save workspace image. If you do, all the object currently in R memory are written to a file `.Rdata`, and all command will be written a file `.Rhistory`.
- These can be loaded later, and you can continue your work from where you left it.
- Loading can be done after starting R using the menu choices File->Load Workspace and File-> Load History.

ReCap I

- R history, features, installation
- Getting help, run, quit
- Packages (library)

Break to check R, R Studio

Next:

- syntax
- data structure

R Data Structures

- Numbers, characters, logicals (TRUE/ FALSE)
- Simplest: Vectors and Matrices
- Lists: Can Contain mixed type variables
- Data Frame: Rectangular Data Set

1. Variables in R

- Numeric
 - Store floating point values
 - Missing values: NA, NaN(not a number)
 - “`is.na(xx)`” is TRUE or both NA and NaN
 - “`is.nan(xx)`” is only TRUE for NaNs
- Boolean
 - Values corresponding to True or False
- Strings
 - Sequence of characters
- Assignment: “`<-`” operator

R as a calculator

```
> 1+1
```

```
[1] 2
```

```
> 3^2
```

```
[1] 9
```

```
> log2(8)
```

```
[1] 3
```

```
> sqrt(2)
```

```
[1] 1.414214
```

```
> seq(1, 4, length=4)
```

```
[1] 1 2 3 4
```

Data Structure in R

	Linear	Rectangular
All Same Type	vector	matrix
Mixed	list	data frame

2. Data Structures: Vectors

- Most mathematical functions and operator can be applied to vectors
 - Without loops!!!
- Vector: a series of numbers or characters
 - A list of numbers, such as (1,2,3,4,5)
 - `>labs<-paste(c("X", "Y"), 1:10, sep="")`
 - `a<-c(1, 2, 3, 4, 5)`
 - `b<-c(2, 2, 2, 2, 2)`
 - `a+b`
 - Command `c` creates a vector that is assigned to object `a`

Data Structures: Vectors

Type these commands in R console

- `>rep(1,10)`
- `>seq(2,6)`
- `>seq(4,20,by=4)`
- `>x<-c(1,3,5,7)`
- `>y<-c(1,3,57)`
- `>x+y`
- `>x*4`
- `>sqrt(x)`

3. Factors

A factor is a **vector** object used to specify a discrete classification (grouping) of the components of other vectors of the same length.

- `>state <- c("tas", "sa", "qld", "nsw", "nsw", "nt", "wa", "wa", "qld", "vic", "nsw", "vic", "qld", "qld", "sa", "tas", "sa", "nt", "wa", "vic", "qld", "nsw", "nsw", "wa", "sa", "act", "nsw", "vic", "vic", "act")`
- `>statef <- factor(state)`
- `>statef`
- `>levels(statef)`
- `>incomes <- c(60, 49, 40, 61, 64, 60, 59, 54, 62, 69, 70, 42, 56, 61, 61, 61, 58, 51, 48, 65, 49, 49, 41, 48, 52, 46, 59, 46, 58, 43)`
- `>incmeans <- tapply(incomes, statef, mean)`
- #function **tapply()**, apply a function, mean() to each group of the first argument.

Factors, cont'd

```
>stdError <-function(x) sqrt(var(x)/length(x))  
>incster<-tappy(incomes, statef, stdError)  
  
>factor(cut(incomes, breaks = 35+10*(0:7))) ->  
incomef  
>table(incomef, statef)
```

Accessing vector elements, indexing

Type these commands in R console

- `>x<-c(1, 3, 5, 7)`
- `>x[1]`
- `>x[-1]` #exclude the 1st element
- `>x[1] <-100;x` #reassign value to an element
- `>x<5` #compare to get boolean vectors
- `>x[x<5]=0; x` #reassign value to elements marked as TRUE

4. Data structure: data frame

- **data frame**: represent the typical data table that researchers come up with – **like a spreadsheet**.
- It is a rectangular table with rows and columns
- data within each column has the same type (e.g. number, text, logical)
- different columns may have different types.

```
>Data
```

```
>ID   invasive      tumor_size      sex
A1      TRUE        6.5           F
A2     FALSE        7            F
A3      TRUE        4.0           M
```


4. Data structure: Data frame, Matrix

- Data frame
 - A table where columns can contain numeric and string values
 - `> d<-data.frame(a, b)`
 - `> data<-
Data.frame(gao=c(180,176), zhong=c(65,75))`
- Matrix
 - All columns must contain either numeric or string values, but these can not be combined
 - `> e<-as.matrix(d)`
 - Data frame d is converted into a matrix e
 - `> f<-as.data.frame(e)`
 - Matrix e is converted into a dataframe f

Accessing data frame

- `>Data["gao"]`
- `>Data[,1]`
- `>Data$gao`
- `>Data[, -2]` #exclude second column

5. Data structure: list

- List
 - Contains a list of objects of possibly different types.
 - `>g<-as.list(d)`
 - Converts a data frame `d` into a list `g`
- Class structures
 - Many of the Bioconductor functions create a formal class structure, such as an `AffyBatch` object.
 - They contain data in slots
 - Slots can be accessed using the `@` operator:
 - `data@cdfName`

Lists

- **vector**: an ordered collection of data of the same type.

```
> a = c(3, 5, 4)
> a[2]
[1] 5
```

- **list**: an ordered collection of data of arbitrary types.

```
> doe = list(name="john", age=28, married=F, city=c("SH", "LA"))
#caution, quote "", coding!!!
> doe$name
[1] "john"
> doe[[1]]
> doe[["name"]]
> doe$age
[1] 28
```

Data structures

check data structure

- Some command need to get, for example, a matrix, and do not accept a data frame. Data frame would give an error message.
- To check the object type:
 - `> class(d)`
- To check what fields there are in the object:
 - `> d`
 - `> str(d)`
- To check the size of the table/matrix:
 - `> dim(d)`
- To check the length of a factor or vector:
 - `> length(a)`

Data structure: names

- Some data frame related commands:
 - `>names (d)`
 - Reports column names
 - `>row.names (d)`
 - Reports row names
- These can also be used for giving the names for the data frame. For example:
 - `>row.names (d) <-c ("a", "b")`
 - `>row.names (d)`

Naming objects

- Naming objects:
 - Never use command names as object names!
 - Object names **can't start with a number**
 - Never use special characters
 - **use dot (.) instead:**
 - Better way: `good.data`
 - Object names are **case sensitive**, just like commands

Recap II

- variable: numeric, character, logic
- vector
- factor
- list
- matrix
- data frame

Read and Write tables

- The `read.table()` function

- To read an entire data frame directly, the external file will normally have a special form.
- The first line of the file should have a name for each variable in the data frame.
- Each additional line of the file has its first item a row label and the values for each variable.

	Price	Floor	Area	Rooms	Age	Cent.heat
01	52.00	111.0	830	5	6.2	no
02	54.75	128.0	710	5	7.5	no
03	57.50	101.0	1000	5	4.2	no
04	57.50	131.0	690	6	8.8	no
05	59.75	93.0	900	5	1.9	yes

...

- numeric variables and nonnumeric variables (factors)

read.table()

- `HousePrice <- read.table("houses.data", header=TRUE)`

Price	Floor	Area	Rooms	Age	Cent.heat
52.00	111.0	830	5	6.2	no
54.75	128.0	710	5	7.5	no
57.50	101.0	1000	5	4.2	no
57.50	131.0	690	6	8.8	no
59.75	93.0	900	5	1.9	yes

...

write.table()

- To write a table:
 - `write.table(dat, "dat.txt", sep="\t")`
 - Here an object `dat` is written to a file called `dat.txt`. This file should be tab-delimited (argument `sep`).
- Every R object can be stored into and restored from a file with the commands “save” and “load”.
 - `> save(x, file="x.Rdata")`
 - `> load("x.Rdata")`
- To capture what is written on the screen:
 - `> sink("output.txt")`
 - `> sink()`

Graphics

- `plot()`
- `barplot()`
- `boxplot()`
- `heatmap()`

Summary

- R installation, R studio,
- R commands, Run, Quit, Save, Help
- Packages installation and load
- R data structure, syntax
- Basic graphics

Resources

- A package specification allows the production of loadable modules for specific purposes, and several contributed packages are made available through the CRAN sites.
- CRAN and R homepage:
 - <http://www.r-project.org/>
It is R' s central homepage, giving information on the R project and everything related to it.
 - <http://cran.r-project.org/>
It acts as the download area, carrying the software itself, extension packages, PDF manuals.
- Getting help with functions and features
 - `help(solve)`
 - `?solve`