bioinformatics.ca

Canadian Bioinformatics Workshops

www.bioinformatics.ca

Module 1
Introduction to R

bioinformatics.ca

Daniele Merico
Exploratory Data Analysis and Essential Statistics using R
January 24-25, 2011

UNIVERSITY OF TORONTO

Donnelly Centre
for Cellular + Biomolecular Research

Post-doctoral Fellow
Donnelly Centre
University of Toronto

http://baderlab.org/
DanieleMerico

What is R?

- **R** is a **programming language** and **software environment** for statistical computing and graphics
  - Data handling (input, output)
  - Matrix operations
  - Statistical tests
  - Graphics (e.g. exploratory statistics)
  - Highly specialized data analysis (e.g. microarrays)
- Originally developed (1991-1996) by *Robert Gentleman* and *Ross Ihaka* as the open-source version of the *S programming language* by *John Chambers* (Bell Labs)

R: Core and Packages

- **R core**
  - Language interpreter (executes R code)
  - User interface (GUI)
  - Graphics terminal
  - Suite of essential tools for statistics and graphics
- **Contributed packages**
  - Specialized data analysis (e.g. microarrays) or graphics
  - Any researcher can develop a package and submit it
  - *Bioconductor* is a project for the development of genomic data analysis packages (http://www.bioconductor.org/)

R Programming Styles

- Two modes of use:
  a. Write a short program (a script), just to analyze some data
  b. Write a longer program, which will be used over and over by you and/or other users

- In this course you will have enough exposure to R to perform (a) but not (b)

How to Use R

1. **Write R code**

   using the built-in editor or any other text editor

   – remember to save your code as a .R file!

2. **Run R code**

   using the GUI (R-Console) or a UNIX-style terminal

   – graphics will be generated by the graphics terminal as additional windows

   – you can save graphics as files using R code

   – you can save data in text format using R code

   – you can save data in the internal R representation by saving your R *workspace*

The Most Simple R Session

1. Open R
2. Write the following R code using the built-in editor

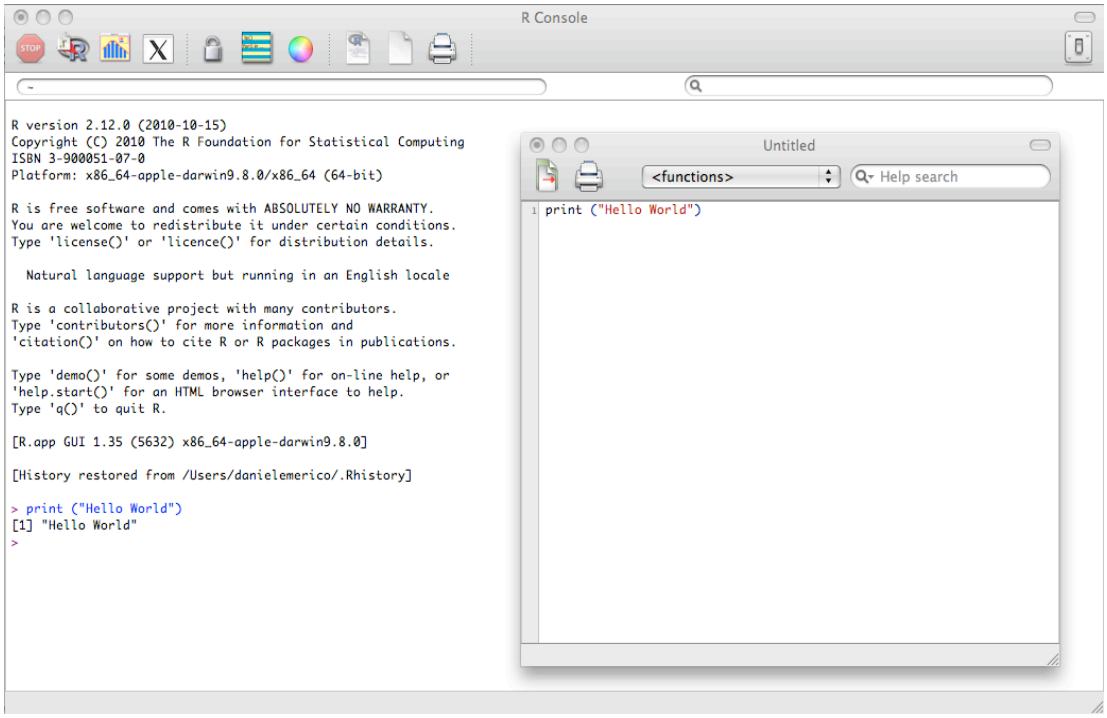
```
print ("Hello World")
```

3. Run the R code

   – Built-in editor (Win): CTRL+F8

   – Built-in editor (Mac): APPLE+RETURN

   – Any platform/editor: copy and paste into the R-console



How to Access the Help

- Exact-match help:

```
help ("keyword")          help ("if")
?"keyword"                ?"if"
```

- Multiple-match help:

```
help.search ("keyword")   help.search ("arithmetic")
??"keyword"               ??"arithmetic"
```

Objects

- In programming languages

  there are named entities called **objects**

  used to store values (i.e. numbers, text strings)

- Handling objects instead of mere values enables to build programs that work with different values in input

  (e.g. a program that computes the mean of a set of measurements)

Object Names

- Objects are identified by a textual label, the **object name**
  - Allowed characters: `a-z A-Z 0-9 _ .`
  - The first character of the name must be alphabetic
  - The name is case sensitive
- The name should suggest what is the function and content of the object
  - I will usually stick to this principle in my examples, structuring the name in two parts separated by ".", although this is not required by R

    `data.df`

    `text.ch`

Object Assignment

- The assignment operator **<-**
  is used to assign a specific value to an object

  ```
  x.n <- value        x.n <- 1
  ```

  - Other languages require to first define the *type* of an object (e.g. numeric, character), and only then assign a value
  - R is more flexible: you assign the value without previous definition, and the type is inferred automatically

Object Class

- Each variable you create belongs to a **class**

  ```
  class (object)
  ```

- The class indicates what type of value the object can assume
  - **Numeric**

    ```
    x.n <- 1 / 5
    ```

  - **Character** (i.e. string of text)

    ```
    text.ch <- "Hello World"
    ```

  - **Logical** (i.e. TRUE or FALSE)
  - **Factor**

    detailed explanation later on

Arithmetic operators

- To manipulate numeric objects,
  you can use the well-known arithmetic operators

`+` addition
`-` subtraction
`*` multiplication
`/` division
`^` power

```
x.n <- 1 + 2
x.n <- 3 ^ 2
```


Logical Values and Expressions

- You can write logical expression,
  which will output a true or false value
  - Comparison operators
  - Evaluate object properties

- You can also assign the results of such expressions
  to objects of type: *logical*

Comparisons Operators

- Greater / smaller than

```
value or object <  value or object
value or object <= value or object
value or object >  value or object
value or object >= value or object

x1.n <- 2
x2.n <- 4
x.bn <- x1.n > x2.n
x.bn                  # FALSE
```


Comparison Operators

- Equal to

```
value or object == value or object

x1.n <- 2
x2.n <- 2
x.bn <- x1.n == x2.n
x.bn            # TRUE
```

  - *always remember to use '==' and not '=' !!!*

Other logical expressions

- Class evaluation

```
is.numeric (object)
is.character (object)
```

- Special value evaluation

```
is.finite (object)
is.na (object)
```


Logical Operators

- For single values have a double symbol
- For vectors have a single symbol (operate element by element)
  - **AND** `&&` `&`
  - **OR** `||` `|`
  - **NOT** `!` `!`

```
T && T          # T
T && F          # F
F && F          # F
T || T          # T
T || F          # T
F || F          # F
!T              # F
!F              # T
```

We will do some practice on data-set sub-setting using logical conditions later on

Special Numerical Values: NA

- **NA** is a special value that is assigned when it is not possible to have an actual value
  - Example: when, in a vector, some element has not been assigned a value (see next chapter for more details)

```
x.n <- NA
is.na (x.n)                    # TRUE
```

  - Be aware of NA values, as they can "propagate" when you compute operations, producing more NA values

Special Numerical Values: Inf

- R can generate values that are not real numbers such as positive and negative infinite

```
 1 / 0          # Inf
-1 / 0          # - Inf
```

  - These values are arithmetically operated with the following results

```
Inf + 1                    # Inf
Inf / 2                    # Inf
```

Workspace

- The workspace is the collection of all the objects you have created in a specific R session
  - To list all objects in your workspace

    ```
    ls ()
    ```

  - To remove object(s) from the workspace

    ```
    rm (objects)        rm (x.n, y.n)
    ```

  - To remove all objects in the workspace

    ```
    rm (list = ls ())
    ```


Save the Workspace

- You can save all objects in your workspace, for use in another session, either using the GUI or using the commands

```
save (objects, file = filename)
save (x.n, y.n, file = "ws_xn.RData")

save.image (file = filename)
save.image (file = "ws_all.RData")
```

save.image saves all objects

Working directory

- Mind that the workspace will be saved to a file located in the current working directory
- To change the working directory use the GUI or the following command

```
setwd (path)
setwd ("C:\Users\Daniele\Documents\Data")
```

- To check what's the current working directory

```
getwd ()
```


Vectors

- A ***vector*** is an object composed of an ordered collection of elements **of the same type**

```
x.nv <- c (1943, 1940, 1942, 1940)
x.chv <- c ("George", "John", "Paul", "Ringo")

class (x.nv)          # numeric
class (x.chv)         # character
```

- `c ()` is the concatenation command, that you can use to generate an ordered collection of elements
  `c (value or object, value or object, …)`
- use `length (vector)` to count the number of vector elements

Vector Indexes

– To access a subset of the vector, use indexes: the first element is associated to index 1, etc…

```
x.chv <- c ("George", "John", "Paul", "Ringo")

x.chv[1]          # "George"
x.chv[3]          # "Paul"

x1.chv <- x.chv[1: 3]
x1.chv <- x.chv[c (1, 4)]

i <- 1
x.chv[i]
```


Vector Indexes

– The attempt to extract an element that does not exist will produce an error

```
x.chv <- c ("George", "John", "Paul", "Ringo")
x.chv[5]
```

– However, you will be able to assign a value to a position that does not exist yet

```
x.chv[5] <- "The Walrus"
```

- If, doing so, you skip positions that have no values assigned, NA values will be generated

```
x.chv[7] <- "The Eggman"
x.chv[6]     # NA
```

Vector Indexes

- Vector elements can also be accessed
  - Using textual labels associated to elements
  - Using vectors of logical values

    (only elements with a corresponding true value will be extracted)

Logical Indexes

- only elements with a corresponding **true value** will be extracted

```
x1.nv <- c (1: 4)
x1.nv[x1.nv > 2]            # 3 4

x2.nv <- c (1: 2, NA, 0, 0)
x2.nv[!is.na (x2.nv)]             # 1 2 0 0
```

- `which()` transforms logical vectors into numerical index vectors

```
which (!is.na (x2.nv))      # 1 2 3 4
```

Matrices

- Matrices and arrays can be regarded as the 2-dimensional extension of vectors

- Like for vectors,
  - Their elements must all be of the same type
  - They have names (matrices: column and row names)

Initialize a matrix

```
matrix (values, ncol, nrow, byrow)

x.mx <-
matrix (c (1: 6), ncol = 2, nrow = 3, byrow = T)
#      [,1] [,2]
# [1,]    1    2
# [2,]    3    4
# [3,]    5    6

dim (x.mx)                  # 3, 2
class (x.mx)                # matrix
```

Initialize a matrix

– Unlike for vectors, once a matrix has been initialized, it is not possible to access elements outside the defined dimensions

```
x.mx <-
matrix (c (1: 6), ncol = 2, nrow = 3, byrow = T)

x.mx[3, 3] <- 7
# Error in x.mx[3, 3] <- 7 : subscript out of bounds
```

– To add additional rows or columns to an initialized matrix, check out the *matrix concatenation operations*

Matrix Indexing

- In analogy to vectors, there are different ways to access the matrix elements
  - Numerical indexes
  - Logical values
  - Text labels (rownames, colnames)

Matrix Indexes

– A matrix element is identified by a pair of indexes

```
x.mx <-
matrix (c (1: 6), ncol = 2, nrow = 3, byrow = T)

x.mx[1, 1]
# 1

x.mx[1: 2, ]
#      [,1] [,2]
# [1,]    1    2
# [2,]    3    4
```


Rownames and Colnames

```
x.mx <-
matrix (c (1: 6), ncol = 2, nrow = 3, byrow = T)

colnames (x.mx) <- c ("c1", "c2")
rownames (x.mx) <- c ("r1", "r2", "r3")

x.mx["r1", ]
# r1 r2 r3
# 1  3  5

class (x.mx["r1", ])          # integer
dim (x.mx["r1", ])            # NULL
```

- Note that by subsetting the matrix to a single dimension the class has changed to (integer) vector, a subtype of numeric vector

Avoiding Matrix to Vector Conversion

- If, after a subsetting operation, your matrix becomes a vector, be aware that you will lose several features typical of matrices
  - **`colnames ()`** and **`rownames ()`** will not be available, only **`names ()`** will be available
  - **`dim ()`** will be NULL, only **`length ()`** will be available
- To avoid this, use the following option:

```
matrix_object [i, j, drop = F]
x33.mx <- x.mx [3, 3, drop = F]
class (x33.mx); dim (x33.mx)
```


Arithmetic Operations

- Matrix and scalar:
  - every element of the matrix is operated, using the scalar
  - Addition, subtraction, multiplication, division, …
- Matrix and vector:
  - the vector is treated as a matrix with only one row or column
  - with recycling if required
- Matrix and matrix:
  - Element by element (compatible dimensions required)
  - Matrix product (similar to dot product)

Matrix Concatenation

- Matrix concatenation enables to add a row or a column to a pre-existing matrix
  - **`rbind ()`** is used to concatenate by row and **`cbind ()`** is used to concatenate by column

```
x.mx <-
matrix (c (1: 6), ncol = 3, nrow = 2, byrow = T)

cbind (x.mx, c (7, 8))
#      [,1] [,2] [,3] [,4]
# [1,]    1    2    3    7
# [2,]    4    5    6    8
```


Data.frames

- A data frame is similar to a matrix
  but every column can have a *different type*
  (numeric, character, logical, factor)

- Statistical data are typically loaded from files as data.frames

Read.table

- Read.table is typically used to read tab-, comma- and space-separated files into data.frames

```
x.df <- read.table (
          filename,
          sep = "\t",
          header = T,
          quote  = "",
          comment.char = "",
          stringsAsFactors = F
          )
```

- **sep** is the separator character; use **"\t"** for tab
- **header** controls the presence of a column titles in the first row
- **quote** is a character vector indicating which characters are used to wrap strings that include the separator character
- **stringsAsFactors** controls automatic conversion of character vectors to factors
- **comment.char** indicates which character will be interpreted as the beginning of a comment (not read into the data.frame)

Write.table

- write.table is the "companion" of read table, it is used to write data.frames to tab/space/comma-separated text files

```
write.table (
          x.df,
          sep = "\t",
          col.names = T, row.names = F,
          quote  = F,
          filename)
```

- **col.names row.names** control whether to print the colnames and rownames; mind that the column of rownames will not have column name
- **quote** controls whether character vectors will be printed with quote characters (usually avoid this)

Data.frame: Initialization

```
x.df <- data.frame (
         c1 = c (1, 2, 3),
         c2 = c ("a", "b", "c"),
         c3 = factor (c ("f1", "f2", "f2")),
         stringsAsFactors = F
         )
```


Data.frame Indexing

- Data.frames are internally represented as lists, with the additional constraint that objects must be vectors or factors with equal length
- Indexing follows the rules for lists
  - `$` → access column and output vector
  - `[[]]` → access column and output vector
  - `[]` → access column and output data.frame

```
class (x.df)              # [1] "data.frame"
class (x.df$c1)           # [1] "numeric"
class (x.df$c2)           # [1] "character"
class (x.df$c3)           # [1] "factor"
class (x.df[, "c1"])      # [1] "data.frame"
```

Subsetting Data.frames Using Logical Conditions

- You will often have to subset to the rows of a data.frame that meet given conditions: use **`subset`**

```
subset (data.frame, logical condition)
```

```
subset (x.df, c1 > 2 | c2 == "b")
#  c1 c2 c3
#2  2  b f2
#3  3  c f2
```

  - In the logical condition you can refer to the data.frame columns just by their name (e.g. **`c1`**)

Factors

- Factors are best used when you have **categorical data**

  i.e. when you have a collection of values that belong to a discrete set, and the same value can appear multiple times
  - Categorical variable: cigarette smoking status ("Status")
  - Categorical values (*levels*): "present", "past", "never"

```
smoke.df <- data.frame (
  Individual = c ("John", "Bob", "Jack", "Al"),
  Status = factor (c ("present", rep ("past", 2),
  "never")),
  stringsAsFactors = F
  )

levels (smoke.df$Status)
```

Table

- To count how many times each level occurs in a factor

```
table (smoke.df$Status)
# never: 1, past: 2, present: 1
```

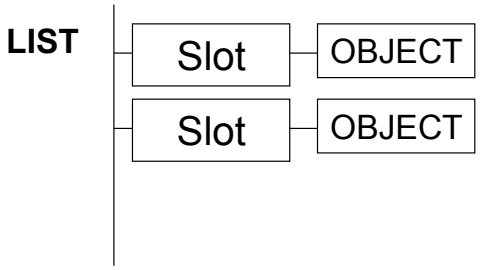
- If two factors are present, you can also cross tabulate them

```
x.df <- data.frame (
          A = c (rep ("a1", 2), rep ("a2", 2)),
          B = c (rep ("b1", 1), rep ("b2", 3))
          )

table (x.df)
#     B
# A    b1 b2
#   a1  1  1
#   a2  0  2
```


Lists

- Lists are ordered collections of objects
  - A list has a number of slots, which can be accessed by names (i.e. character labels) or by numeric indexes
  - The content of the slot can be of any class (single value, numeric or character; vector; matrix; list; etc…)



  - Several R functions for statistics output a list

List Example

```
x.ls <-
list (Name = "John", Surname = "Locke", Birth_year =
  1632)

# $Name
# [1] "John"
#
# $Surname
# [1] "Locke"
#
# $Birth_year
# [1] 1632
```


Accessing List Slots

- A slot can be accessed
  - By numerical or logical index
  - By slot name value

```
x.ls[[2]]
x.ls$Surname
x.ls[["Surname"]]
s.ch <- "Surname"; x.ls[[s.ch]]

# "Locke"
```

Lab Assignments

1. Read "Forbes_2004.txt" as a data.frame
2. Count the number of categorical values of 'category' column
3. Count the number of NA values of 'profits' column
4. Remove the rows with NA values
5. Write the subset data to a new table
6. List the object in the workspace
7. Save the object 'Forbes_nna.df' as 'Forbes_nna.RData' workspace

Solutions in the next slides, but try to figure it out yourself


```
# 1
# set working directory...
Forbes.df <- read.table (
   file = "Forbes_2004.txt",
   sep = "\t", header = T,
   stringsAsFactors = T)

# 2
table (Forbes.df$category)

# 3
notna.ix <- which (!is.na (Forbes.df$profits))
length (notna.ix) / nrow (Forbes.df)
# 0.9975

# 4
Forbes_nna.df <- Forbes.df[notna.ix, ]
```

```
# 5
write.table (
   Forbes_nna.df,
   sep = "\t",
   col.names = T, row.names = F,
   quote = F,
   file = "Forbes_2004_nna.txt"
   )

# 6
ls ()

# 7
# set working directory...
save (Forbes_nna.df, file = "Forbes_nna.RData")
```


References

- **R Tutorial**
  http://www.cyclismo.org/tutorial/R/
  http://baderlab.org/PathwayAnalysisReadings#Lectures
- **R Project Home**
  http://www.r-project.org/
- **An Introduction to R**
  http://cran.r-project.org/doc/manuals/R-intro.pdf
  The basic manual for R programming
- **R Reference Card**
  http://cran.r-project.org/doc/contrib/Short-refcard.pdf
- Peter Dalgaard. **Introductory Statistics with R**. Springer
  A guide to the use of R and fundamental statistical analysis
- **More Books**
  http://www.r-project.org/doc/bib/R-jabref.html

How to Install R on Your Own

- **Windows Users**
  - Install R
    - Download from http://cran.r-project.org/bin/windows/base/
    - Vista users: to avoid pain, install R in a subfolder of your user folder or refer to this:

      http://cran.r-project.org/bin/windows/base/rw-FAQ.html#Does-R-run-under-Windows-Vista_003f
  - Optional: install R code editor
    - Download Notepad++ http://sourceforge.net/projects/notepad-plus/files/
    - Download NppToR (interfaces Notepadd++ and R) http://sourceforge.net/projects/npptor/

How to Install R on Your Own

- **Mac Users**
  - Install R
    - http://cran.r-project.org/bin/macosx/
  - R code editor:
    - Just use the built-in one

We are on a Coffee Break & Networking Session