

Introduction to R/Bioconductor

Instructor: Mary Yang, PhD

Graduate Assistant: Dan Li and Yifan Zhang

August 2, 2017

What is R?

- R is a free, open source programming language
- R is an integrated suite of software facilities for data manipulation, simulation, calculation and graphical display
- R runs on a wide variety of platforms such as Linux, Windows, and macOS
- The R project web page <http://www.r-project.org>
<http://www.r-project.org>

Why R?

- Handles and analyzes data very effectively
 - Contains a suite of operators for calculations on arrays and matrices
- Has the graphical capabilities for very sophisticated graphs and data displays
- An elegant, object-oriented programming language
- The use of variables, loops, merging operations and data sets, as well as downloadable add-on packages to enhance functionality, make the R programming options powerful and endless
- R is one of the most widely used language in Bioinformatics

R Studio is a IDE with 4 Windows on the main screen

- Code: for coding / code checker
- Console: coding calculations and display results
- Work Space: list of objects and calculations that are created
- Five tab window
 - Files: stored and imported files
 - Plot: displayed area, graph view selection, export option
 - Packages: list of packages installed on system, option to search and install other packages
 - Help: more information about functions, arguments, user manuals, etc
 - Viewer: used to view local web content such as widgets, rCharts, and applications

The screenshot shows the RStudio interface with three main components highlighted by blue rounded rectangles:

- Code:** The top-left pane containing R code for data manipulation and plotting. The code includes functions like `order`, `consonant`, `expdown`, `explos`, `par(mfrow = c(2,2))`, `plot(as.numer ~ x$lab)`, and `boxplot(x, b.col = c("red", "green"))`.
- Console:** The bottom-left pane showing the execution of the `boxplot` command: `> boxplot(x, b.col = c("red", "green"))`.
- Workspace:** The top-right pane displaying the environment with variables `x` and `test`. The `Values` section shows the first few elements of each vector: `x` (1:20) and `test` (1:20).

Below the workspace, a boxplot is displayed with two groups (1 and 2). Group 1 has a red box with a median around 70, and Group 2 has a green box with a median around 15. The y-axis ranges from 0 to 80.

The screenshot displays the RStudio application window. The top menu bar includes File, Edit, Code, View, Plots, Session, Build, Debug, Profile, Tools, Window, and Help. The main interface is divided into several panes:

- Console:** Shows the R startup output, including the version (3.3.2) and the "Sincere Pumpkin Patch" message. The text is as follows:

```
R version 3.3.2 (2016-10-31) -- "Sincere Pumpkin Patch"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin13.4.0 (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

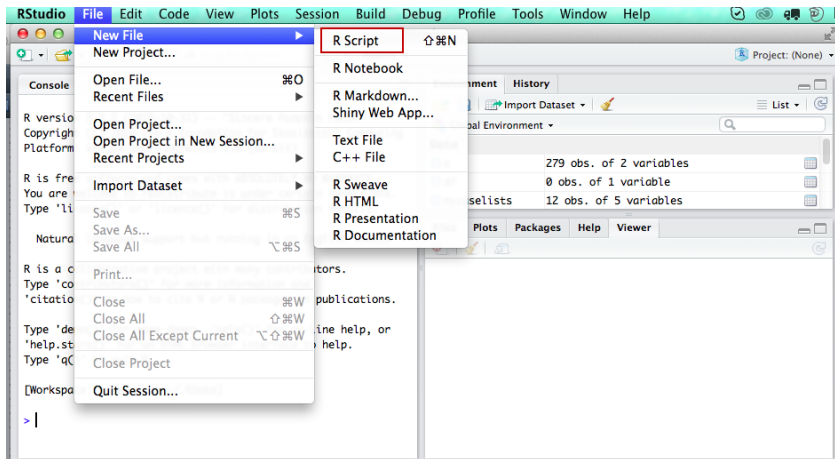
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

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.

[Workspace loaded from ~/.RData]

> |
```
- Environment:** Displays the current environment, showing the Global Environment and a list of data objects:

Data	
a	279 obs. of 2 variables
df	0 obs. of 1 variable
mycaselists	12 obs. of 5 variables
- Files, Plots, Packages, Help, Viewer:** These panes are currently empty.



The screenshot displays the RStudio environment with the following components:

- Script Editor:** Contains R code for loading the `RColorBrewer` library, importing the `VADeaths` dataset, and plotting a histogram. The histogram is titled "Set3 3 colors" and uses the `col-brewer.pal(3, "Set3")` color palette. The code includes comments for graphical parameters and breaks.
- Console:** Shows the R startup sequence, including the R version (3.1.72), locale settings, and workspace loading from `~/RData`.
- Environment Pane:** Lists the following objects:

Object	Description
a	279 obs. of 2 variables
df	0 obs. of 1 variable
mycaselists	12 obs. of 5 variables
myclinicaldata	279 obs. of 4 variables
mygeneticprofile	7 obs. of 6 variables

 The **Values** pane shows:

myconcerstudy	"hnscc_tgca_pub"
mycaselist	"hnscc_tgca_pub_3way_complete"
mycdfs	Classes 'CDFs', 'Object', atomic [1:11 NA]

Environment History

Global Environment

Data

a	279 obs. of 2 variables
df	0 obs. of 1 variable
mycaselists	12 obs. of 5 variables
myclinicaldata	279 obs. of 4 variables
mygeneticprofile	7 obs. of 6 variables
VADeaths	num [1:5, 1:4] 11.7 18.1 26.9 41 66 8.7 11.7 20.3 30.9 54...

Values

mycancerstudy	"hnscc_tsga_pub"
mycaselist	"hnscc_tsga_pub_3env_complete"

Files Plots Packages Help Viewer

Zoom Export Publish

Set3 3 colors

Frequency

VADeaths

```

13
14 #####
15 ### Lesson 1. Histogram
16 #####
17
18 #import a library named RColorBrewer
19 library(RColorBrewer)
20
21 #import a data set named VADeaths, you can type VADeaths to see the detail
22 data(VADeaths)
23
24 #set graphical parameters, two rows and 3 graphical in each row
25 par(mfrow=c(1,1))
26
27 #draw Histogram
28 # breaks : the number of cells for the histogram
29 # col : color of the bars in the graph
30 # main : title of the graph
31 hist(VADeaths,breaks=12, col=brewer.pal(3,"Set3"),main="Set3 3 colors")
32 hist(VADeaths,breaks=4, col=brewer.pal(3,"Set2"),main="Set2 3 colors")
33 hist(VADeaths,breaks=8, col=brewer.pal(3,"Set1"),main="Set1 3 colors")
34
31:1 (Untitled) R Script

```

Console

```

[Workspace loaded from ~/.RData]
> plot(c(1,2),c(2,3))
> setwd("~/Mary_Yang/Workshop")
> library(RColorBrewer)
>
> #import a data set named VADeaths, you can type VADeaths to see the detail
> data(VADeaths)
>
> #set graphical parameters, two rows and 3 graphical in each row
> par(mfrow=c(1,1))
> hist(VADeaths,breaks=12, col=brewer.pal(3,"Set3"),main="Set3 3 colors")
>

```

- R is a **case-sensitive**, interpreted language
 - For example, a and A are two different objects.
- Two ways to run R command
 - Enter commands into the R console window at the command prompt (>)
 - Create R-scripts in an editor and save them in a file (filename.R) for later re-use.

Getting Started with R

- Data management
 - Data importing.
 - Write data into files.
 - Save data as an image.
- Data process
 - Types of data.
 - Arithmetic of data.
- Functions
- Packages

Data Import

- Data can be entered from the console.
- Larger data often be read as values from external files rather than entered at the keyboard.
- Different data formats that can be imported into R and different functions to call them.
 - .csv
 - .txt
 - HTML table
 - Excel

- CSV

```
df = read.csv(file_name.csv)
df = read.csv2(file_name.csv)
```

- TXT

```
df = read.table(file_name.txt)
```

Building-in data set in R

- R also contains many datasets that are built-in to the software
- These datasets are stored as data frames. To see the list of datasets, use

```
> data()
```

- Then, a window will open and the available datasets are listed

Building-in data set in R : an example

Example: to open the dataset called *Orange*

```
> data (Orange)
```

After doing so, the data frame *Orange* is now in your workspace.

```
# To learn more about this data, type  
> ?Orange  
> Orange  
  Tree age circumference  
1  1  118  30  
2  1  484  58  
3  1  664  87  
4  1 1004 115  
5  1 1231 120  
6  1 1372 142  
7  1 1582 145
```


- CSV:

```
write.csv(data, file = "path/file_name.csv")
```

- TXT

```
write.table(data, file = "path/file_name.txt", row.names =  
  FALSE, col.names = TRUE)
```

- The argument **row.names = FALSE** makes that no row names are written to the file. Because nothing is specified about **col.names**, the default option **col.names = TRUE** is chosen and column names are written to the file.

More options

```
help (write.csv)
```

```
?write.csv
```

```
?write.table
```

The screenshot shows the RStudio interface. The script editor on the left contains R code for plotting a histogram. The console at the bottom left shows the command `?write.table` entered. The right-hand pane displays the help documentation for `write.table`, with the `Help` tab selected in the top navigation bar. The help text includes the title "Data Output", a "Description" section stating that `write.table` prints its required argument `x` to a file or connection, and a "Usage" section with the function signature: `write.table(x, file = "", append = FALSE, quote = TRUE, sep = " ", eol = "\n", na = "NA", dec = ".", row.names = TRUE, col.names = TRUE, qmethod = c("escape", "double"), fileEncoding = "")`. Below the usage is the command `write.csv(...)`.

Saving Data

Save the data into image for reuse by R

```
save(data, file = "data.RData")  
load("data.RData")
```

Data process: computing with R

- When variables are used, they need to be **initialized** with numbers.
- R can take as arguments for its functions single numbers, vectors, matrices, or data frames.
- The assignment operator is “←”. Alternatively, as of R version 1.4.0, you can use “=” as the assignment operator.

```
A = 5
```

```
A <- 5
```

Data Types

Types	Examples
Integer: Natural numbers	1, 2, 3
Numeric: Decimal values	1.5, 2.2, 3.7
Logical: Boolean values	TRUE or FALSE (T or F)
Character: Text or string values	"cat" "blue"

Data Structures

- R has a wide variety of data structures:
 - vector
 - matrix
 - data frame
 - list

	Homogeneous	Heterogeneous
1d	Vector	List
2d	Matrix	Data frame
nd	Array	

Vector

```
# numeric vector
```

```
x = c(1,2,5.3,6,-2,4)
```

```
#character vector
```

```
y = c("one", "two", "three")
```

```
#logical vector
```

```
z = c(TRUE,TRUE,TRUE,FALSE,TRUE,FALSE)
```

Vector

#replicates the values in x

```
> a = rep (1, 6)
```

```
> a
```

```
[1] 1 1 1 1 1 1
```

#Generate regular sequences.

```
> a = seq (1, 6)
```

```
> a
```

```
[1] 1 2 3 4 5 6
```

#Or

```
> a = 1:7
```

```
> a
```

```
[1] 1 2 3 4 5 6 7
```


Vector

You can refer to elements of a vector using subscripts.

```
> a  
[1] 1 2 3 4 5 6
```

```
> a[2] #Access the second element of the vector  
[1] 2
```

```
> a[c(2,4)] #Access the second and fourth elements of the vector  
[1] 2 4
```

```
> a[1:3] #Access the first three elements of the vector  
[1] 1 2 3
```

- All columns in a matrix must have the same mode(numeric, character, etc.) and the same length. The general format is

matrix(vector, nrow = r, ncol = c, byrow = FALSE, dimnames = list(char_vector_rownames, char_vector_colnames))

- *byrow = TRUE* indicates that the matrix should be filled by rows.
- *byrow = FALSE* indicates that the matrix should be filled by columns (the default).
- *dimnames* provides optional labels for the columns and rows.

Matrix

```
# generates 3 * 5 numeric matrix  
> x = matrix(1:15, nrow=3,ncol=5)
```

```
#Default byrow=FALSE
```

```
> x  
      [,1] [,2] [,3] [,4] [,5]  
[1,]    1    4    7   10   13  
[2,]    2    5    8   11   14  
[3,]    3    6    9   12   15
```

Matrix

```
# another example
> cells = c(1,26,24,68)
> rnames =c("R1", "R2")
> cnames = c("C1", "C2")
> m = matrix(cells, nrow=2, ncol=2, byrow=TRUE,dimnames=list(
  rnames, cnames))
```

```
> m
  C1 C2
R1  1 26
R2 24 68
```

Matrix

You can identify rows, columns or elements using subscripts.

```
#4th column of matrix  
> x[,4]
```

```
# 3rd row of matrix  
> x[3,]
```

```
# rows 2,3 of columns 1,2,3  
> x[2:3,1:3]
```

Data Frame

A data frame is more general than a matrix, in that different columns can have different modes (numeric, character, factor, etc.)

```
> d = c(7,9,5, 20)
> e = c("red", "white", "red", NA)
> f = c(TRUE,TRUE,TRUE,FALSE)
> mydata = data.frame(d,e,f)
#Assign columns name
> colnames (mydata) = c("ID", "Color", "Status")
```

```
> mydata
  ID Color Status
1  7   red   TRUE
2  9 white   TRUE
3  5   red   TRUE
4 20 <NA> FALSE
```

Data Frame

There are a variety of ways to access the elements of a data frame.

```
# columns 2,3 of data frame
> mydata[,2:3]
  Color Status
1  red    TRUE
2 white   TRUE
3  red    TRUE
4 <NA> FALSE
```

```
# columns ID and Color from data frame
> mydata[,c("ID", "Color")]

#Column ID
> mydata$ID
```

- A list is an ordered collection of objects (components).
- A list allows you to gather a variety of (possibly unrelated) objects under one name

```
# example of a list with 4 components a string, a numeric vector  
, a matrix, and a scalar  
> mylist <- list(name="Fred", my.numbers=a, my.matrix=x, age=5.3)
```


List

```
> mylist
```

```
$name
```

```
[1] "Fred"
```

```
$my.numbers
```

```
[1] 1 2 3 4 5 6 7
```

```
$my.matrix
```

	[,1]	[,2]	[,3]	[,4]	[,5]
[1,]	1	4	7	10	13
[2,]	2	5	8	11	14
[3,]	3	6	9	12	15

```
$age
```

```
[1] 5.3
```

Identify elements of a list using the `[[]]` convention.

```
# 2nd component of the list  
> mylist[[2]]  
[1] 1 2 3 4 5 6 7  
  
# component named my.numbers in list  
> mylist[["my.numbers"]]  
[1] 1 2 3 4 5 6 7
```

Useful Functions

number of elements or components in a vector or a list

> `length(object)`

structure of an object

> `str(object)`

class or data type of an object

> `class(object)`

list the variables in an object

> `names(object)`

dimensions of an object, such as matrix or data frame

> `dim(object)`

Useful Functions

```
# combine objects into a vector
```

```
> c(object,object,...)
```

```
# combine objects as columns
```

```
> cbind(object, object, ...)
```

```
# combine objects as rows
```

```
> rbind(object, object, ...)
```

Data Conversion

Display the data type

```
> class (m)
[1] "matrix"

> class (x)
[1] "matrix"
```

Conversion from one type of data structures to another can easily be done

```
> m1 = as.data.frame(m)
> x1 = as.vector (x)
```

If unsure about the type, you can write

```
> is.data.frame(m1)
[1] TRUE
> is.data.frame(m)
[1] FALSE

> is.vector(x)
[1] FALSE
```

Draw a Heat Map

Description

A heat map is a false color image (basically `image(t(x))`) with a dendrogram added to the left side and to the top. Typically, reordering of the rows and columns according to some set of values (row or column means) within the restrictions imposed by the dendrogram is carried out.

Usage

```
heatmap(x, Rowv = NULL, Colv = if(symm)"Rowv" else NULL,
        distfun = dist, hclustfun = hclust,
        reorderfun = function(d, w) reorder(d, w),
        add.expr, symm = FALSE, revC = identical(Colv, "Rowv"),
        scale = c("row", "column", "none"), na.rm = TRUE,
        margins = c(5, 5), ColSideColors, RowSideColors,
        cexRow = 0.2 + 1/log10(nr), cexCol = 0.2 + 1/log10(nc),
        labRow = NULL, labCol = NULL, main = NULL,
        xlab = NULL, ylab = NULL,
        keep.dendro = FALSE, verbose = getOption("verbose"), ...)
```

Arguments

x	numeric matrix of the values to be plotted.
---	---

Rowv	determines if and how the row dendrogram should be computed and reordered. Either a dendrogram or a vector of values used to reorder the row dendrogram or NA to suppress any row dendrogram (and reordering) or by default NULL, see
------	---

R function: Control structure

- R includes the usual control-flow statements (like conditional execution and looping) found in most programming languages. These include (the syntax can be found in the help file accessed by **?Control**):

```
if(cond) expr
if(cond) cons.expr else alt.expr

for(var in seq) expr
while(cond) expr
repeat expr
break
next
```


R function: Control structure

Loop: Perform the same process several times or on several variables

```
> for (i in 1:5){print (i)}  
[1] 1  
[1] 2  
[1] 3  
[1] 4  
[1] 5
```

Example: Control structure and matrix

```
> mat = matrix (sample (100, 50), nrow = 10)
> mat
```

	[,1]	[,2]	[,3]	[,4]	[,5]
[1,]	93	37	16	52	75
[2,]	87	90	21	100	46
[3,]	67	27	88	33	17
[4,]	32	91	55	56	9
[5,]	26	36	31	38	47
[6,]	59	57	62	83	18
[7,]	60	66	10	79	99
[8,]	82	40	69	70	84
[9,]	74	15	35	94	97
[10,]	5	48	65	85	6

Example: Control structure and matrix

```
> m1 = rowMeans (mat)
> m1
[1] 54.6 68.8 46.4 48.6 35.6 55.8 62.8 69.0 63.0 41.8

> m2 = apply (mat, 1, mean)
> m2
[1] 54.6 68.8 46.4 48.6 35.6 55.8 62.8 69.0 63.0 41.8
```

Example: Control structure and matrix

```
> for (i in 1:10){  
+ m3 = mean(mat[i,])  
+ print (m3)  
+ }  
[1] 54.6  
[1] 68.8  
[1] 46.4  
[1] 48.6  
[1] 35.6  
[1] 55.8  
[1] 62.8  
[1] 69  
[1] 63  
[1] 41.8
```

Example: Control structure and matrix

Tip: when performing loop on a matrix, Basic function (fastest) → apply (faster) → for loop (slow)

```
#fastest  
> m1 = rowMeans (mat)  
  
#faster  
> m2 = apply (mat, 1, mean)  
  
#slow  
> for (i in 1:10){  
+ m3 = mean(mat[i,])  
+ print (m3)  
+ }
```

Example 2: Control structure and matrix

Create a matrix

```
> height = c(6,5,5.4,6.1,6.5,5.5,5.9)
> student = c("boy", "girl", "girl", "boy", "boy", "boy", "girl" )
> mat = data.frame (height, student)
```

```
> mat
  height student
1    6.0    boy
2    5.0   girl
3    5.4   girl
4    6.1    boy
5    6.5    boy
6    5.5    boy
7    5.9   girl
```

Example 2: Control structure and matrix

```
> for (i in 1:nrow(mat)){  
+   if (mat[i,2] == "girl"){  
+     print (mat[i,])  
+   }  
+ }  
height student  
2      5      girl  
height student  
3     5.4     girl  
height student  
7     5.9     girl
```

Logical operators

- Many of these statements require the evaluation of a logical statement, and these can be expressed using logical operators:

Operator	Meaning
==	Equal to
!=	Not equal to
<, <=	Less than, less than or equal to
>, >=	Greater than, greater than or equal to
&	Logical AND
	Logical OR

User-defined function

- R users can define their own function. The general format for creating a function is

```
functionName <- function(arg1, arg2, ...) { R code }
```

- In the above, **functionName** is any allowable object name and **arg1**, **arg2**, ... are function arguments.
- As with any R function, they can be assigned default values.
- When you write a function, it is saved in your workspace as a function object.

A user-written function

The screenshot shows the RStudio interface with a script editor on the left and the Environment pane on the right. A red box highlights the 'Run' button in the toolbar, with a tooltip that says 'Run the current line or selection (R+Enter)'. Another red box highlights the function definition in the script editor, and a third red box highlights the function 'f1' in the Environment pane's 'Functions' section.

```
1- f1 = function (a, b){
2-   #This function returns the maximum of two scalars
3-   #for the statement that they are equal
4-
5-   if (is.numeric (c(a, b))){
6-     if (a < b) return (b)
7-     if (a > b) return (a)
8-     else print ("The values are equal")
9-   }
10-  else print ("Character inouts are equal")
11- }
```

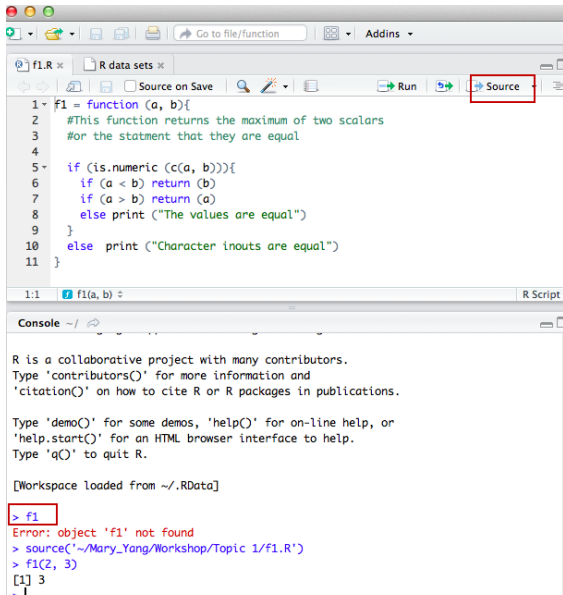
Environment

Object	Class	Attributes
Global Environment		
mydata	data.frame	4 obs. of 3 variables
mygeneticprofile	data.frame	7 obs. of 6 variables
Orange	data.frame	35 obs. of 3 variables
x	integer	int [1:3, 1:5] 1 2 3 4 5 6 7 8 9 10 ...
Values		
a	integer	int [1:7] 1 2 3 4 5 6 7
cells	numeric	num [1:4] 1 26 24 68
cnames	character	chr [1:2] "C1" "C2"
d	numeric	num [1:4] 7 9 5 20
data_df	list	List of 0
e	character	chr [1:4] "red" "white" "red" NA
f	logical	logi [1:4] TRUE TRUE TRUE FALSE
mycancerstudy	character	"hnscc_tcga_pub"
mycaselist	character	"hnscc_tcga_pub_3way_complete"
mycgds	Classes 'CGDS', 'Object' atomic	[1:1] NA
mylist	list	List of 4
rnames	character	chr [1:2] "R1" "R2"
t	list	List of 4
table	list	List of 0
tables	list	List of 0
u	character	"<DOCTYPE html>\n<html class=\"client-nojs\" lang=\"en\" dl_
url	character	"<html>\n<head><title>400 Bad Request</title></head>\n<nb_
url	character	"https://en.wikipedia.org/wiki/List_of_countries_and_depende_
Functions		
f1	function	function (a, b)

User defined function

- The function object **f1** will remain in your workspace until you remove it or quit R session.
- You can save these commands in R script (f1.R) for later-use.

Save user-written function for later-use



The screenshot shows the R Studio interface. The top toolbar has a red box around the 'Source' button. The script editor contains the following R code:

```
1- f1 = function (a, b){
2   #This function returns the maximum of two scalars
3   #or the statement that they are equal
4
5-   if (is.numeric (c(a, b))){
6     if (a < b) return (b)
7     if (a > b) return (a)
8     else print ("The values are equal")
9   }
10  else print ("Character inputs are equal")
11 }
```

The console output shows the following text:

```
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.

[Workspace loaded from ~/.RData]
> f1
Error: object 'f1' not found
> source('~/.Mary_Yang/Workshop/Topic 1/f1.R')
> f1(2, 3)
[1] 3
```

Users-written function: Another example

```
> calcul = function (a, b){
+ result = (a + b * 5) ^ 2
+ return (result)
+ }
> calcul (4, 2)
[1] 196
> calcul (0.5, 0.1)
[1] 1
> calcul ("A" , "B")
Error in b * 5 : non-numeric argument to binary operator
```

- It is recommended that you write and edit all of your R code in a script before you run it in the console.
- It creates a reproducible record of your work.
- You can save your script and then use it to rerun your entire analysis.
- Scripts are also very handy for editing and proofreading your code

- When you open an R Script (File → New File → R Script in the menu bar), RStudio creates a fourth pane above the console where you can write and edit your code.
- RStudio comes with many built-in features that make it easy to work with scripts.
 - R will run whichever line of code your cursor is on by clicking the **Run** button
 - If you have a whole section highlighted, R will run the highlighted code by clicking the **Run** button
 - Alternatively, you can run the entire script by clicking the **Source** button.
- To save a script, click the scripts pane, and then go to File → Save As in the menu bar

Built-In Function: numeric function

Function	Description
<code>abs(x)</code>	absolute value
<code><u>sqrt</u>(x)</code>	square root
<code>ceiling(x)</code>	<code>ceiling(3.475)</code> is 4
<code>floor(x)</code>	<code>floor(3.475)</code> is 3
<code><u>trunc</u>(x)</code>	<code>trunc(5.99)</code> is 5
<code>round(x, digits=n)</code>	<code>round(3.475, digits=2)</code> is 3.48
<code><u>signif</u>(x, digits=n)</code>	<code>signif(3.475, digits=2)</code> is 3.5
<code>cos(x), sin(x), tan(x)</code>	
<code>log(x)</code>	natural logarithm
<code>log10(x)</code>	common logarithm
<code>exp(x)</code>	<code>e^x</code>

Built-In Function: character function

Function	Description
<code>substr(x, start=n1, stop=n2)</code>	Extract or replace substrings in a character vector. <code>x <- "abcdef"</code> <code>substr(x, 2, 4)</code> is "bcd" <code>substr(x, 2, 4) <- "22222"</code> is "a222ef"
<code>grep(pattern, x)</code>	Search for <i>pattern</i> in <i>x</i> .
<code>gsub(pattern, replacement, x)</code>	perform replacement of matches determined by regular expression matching
<code>strsplit(x, split)</code>	Split the elements of character vector <i>x</i> at <i>split</i> . <code>strsplit("abc", "")</code> returns 3 element vector "a", "b", "c"
<code>paste(..., sep="")</code>	Concatenate strings after using <i>sep</i> string to separate them. <code>paste("x", 1:3, sep="")</code> returns <code>c("x1", "x2" "x3")</code> <code>paste("x", 1:3, sep="M")</code> returns <code>c("xM1", "xM2" "xM3")</code> <code>paste("Today is", date())</code>
<code>toupper(x)</code>	Uppercase
<code>tolower(x)</code>	Lowercase

Built-In Function: statistical function

Function	Description
<code>mean (x)</code>	mean of object x
<code><u>sd</u> (x)</code>	standard deviation of object (x)
<code>median (x)</code>	median
<code>range (x)</code>	range
<code>sum(x)</code>	sum
<code>min (x)</code>	minimum
<code>max(x)</code>	maximum

Missing data

- In R, missing values are represented by the symbol NA Test for missing values:

```
#Test for missing values
```

```
> is.na (x)
```

```
#Excluding Missing Values from Analyses
```

```
> mean (x, na.rm = TRUE)
```

```
#Create a new dataset without missing data
```

```
> newData = na.omit (myData)
```

R packages

- R started with basic packages. To see the list of all available packages on systems, you can type into the R console window

```
> library()
```

- To install a new package, you can either click on Packages-install package(s), or type commands into the console window:

```
> install.packages ("gplots") # install a package called  
  gplots  
> library (gplots)} #load gplots packages
```

- To update a package called gplots

```
> update.packages ("gplots")
```

R Package

The screenshot shows the RStudio interface with the following components:

- Environment/History pane:** Shows the 'Packages' tab with 'Install' and 'Update' buttons highlighted. Below, the 'System Library' pane lists installed packages, with 'ade4' selected.
- Console:** Shows the following R commands:

```
[32] "Top_Subnetwork_0.34_G0terms_Test.xls"
[33] "ytb.uid"
> library("biomaRt", lib.loc="/Library/Frameworks/R.framework/Versions/3.3/Resources/library")
> detach("package:biomaRt", unload=TRUE)
> install.packages("gplots")
```

The output shows the download of the 'gplots' package (499 KB).
- Install Packages dialog:** A modal dialog box is open with the following fields:
 - Install from:** Repository (CRAN)
 - Packages:** gplots
 - Install to Library:** /Library/Frameworks/R.framework/Versions/3.3/Resources/
 - Install dependencies:** checkedThe 'Install' button is highlighted.

The Workspace

- All variables or “objects” created in R are stored in what’s called the workspace
- To see what variables are in the workspace, you can use the function `ls()` to list them.
- To remove objects from the workspace use the `rm()` function:

```
# delete a object called  
> rm (X)
```

Manipulating file paths in R

Pathnames in R are written with forward slashes “/”, although in windows backslashes, “\”, are used.

```
#To set a working directory in R:
```

```
> setwd("Directory name")
```

```
#To print the current working directory:
```

```
> getwd()
```

```
#To show the files in the current working directory:
```

```
> dir()
```

```
#To create a directory for your project - good way to organize  
your files
```

```
> dir.create("R Workshop 2017")
```

Manipulating file paths in RStudio

The screenshot shows the RStudio interface with the 'Session' menu open. The menu options are: New Session, Interrupt R, Terminate R..., Restart R, Set Working Directory (highlighted), Load Workspace..., Save Workspace As..., Clear Workspace..., and Quit Session... A sub-menu for 'Set Working Directory' is also visible, containing: To Source File Location, To Files Pane Location, and Choose Directory... (highlighted).

The background shows the R script editor with the following code:

```
1 f1 = function (a, b){
2   #This function returns the maximum of
3   #for the statement that they are equal
4
5   if (is.numeric (c(a, b))){
6     if (a < b) return (b)
7     if (a > b) return (a)
8   }
9   else print ("The values are equal")
10
11 else print ("Character inouts are equal")
12 }
```

The console shows the following output:

```
library()
> detach("package:biomaRt", unload=TRUE)
> install.packages("gplots")
trying URL 'https://cran.rstudio.com/bin/macosx/mavericks/contrib/3.3/gplots_3.0.1.tgz'
Content type 'unknown' length 511025 bytes (499 KB)
downloaded 499 KB

The downloaded binary packages are in
  /var/folders/yr/959dn91x20706jth9h5xxtvc000gn/T/Rtmpk88TKI/downloaded_packages
> history
Error: object 'history' not found
> history()
> |
```

The Packages pane on the right shows a list of installed and available packages:

Package	Description	Version
affy	Analysis of Ecological Data - Exploratory and Euclidean Methods in Environmental Sciences	1.7-5
affyio	Methods for Affymetrix Oligonucleotide Arrays	1.52.0
ALL	Tools for parsing Affymetrix data files	1.44.0
ALL	A data package	1.16.0
annotate	Annotation for microarrays	1.52.1
AnnotationDbi	Annotation Database Interface	1.36.2
AnnotationForge	Code for Building Annotation Database Packages	1.16.1
ape	Analyses of Phylogenetics and Evolution	4.1
aroma.light	Light-Weight Methods for Normalization and Visualization of Microarray Data using Only Basic R Data Types	3.4.0
assertthat	Easy pre and post assertions.	0.1
BH	Boost C++ Header Files	1.62.0-1
biclust	BiCluster Algorithms	1.2.0
Biobase	Biobase: Base functions for Bioconductor	2.34.0
BiocGenerics	S4 generic functions for Bioconductor	0.20.0
BiocInstaller	Install/Update Bioconductor, CRAN, and github Packages	1.24.0
BiocParallel	Bioconductor facilities for parallel evaluation	1.8.1
biomaRt	Interface to BioMart databases (e.g. Ensembl, COSMIC, Wormbase and Gramene)	2.30.0
Biostrings	String objects representing biological sequences, and matching algorithms	2.42.1
bitops	Bitwise Operations	1.0-6

Getting Help

- R has an extensive help facility
- Apart from the Help window launched from the Help menu, it is also available from the command line prompt
- For instance

```
#explain about square root function  
> help(sqrt)  
> ?sqrt
```

- Most of the help files also include examples. You can run all of them by using the **example()**

```
#run all the examples from the matrix help file  
> example(matrix)  
  
#run all the examples from the plot help file  
> example(plot)
```

- Bioconductor is an open source and open development software project for the analysis of biomedical and genomic data.
- The project was started in the Fall of 2001 and includes developers in many countries

- Provide access to powerful statistical and graphical methods for the analysis of genomic data.
- Facilitate the integration of biological metadata (GenBank, GO, Entrez Gene, PubMed) in the analysis of experimental data.
- Allow the rapid development of extensible, interoperable, and scalable software.
- Promote high-quality documentation and reproducible research.
- Provide training in computational and statistical methods.

Bioconductor packages

- General infrastructure
 - Biobase, Biostrings, biocViews
- Annotation:
 - annotate, annaffy, biomaRt, AnnotationDbi
- Graphics/GUIs:
 - geneplotter, hexbin, limmaGUI, exploRase
- Pre-processing:
 - affy, affycomp, oligo, makecdfenv, vsn, gcrm, limma
- Differential gene expression:
 - genefilter, limma, ROC, siggenes, EBArrays, factDesign
- GSEA/Hypergeometric Testing
 - GSEABase, Category, GOstats, topGO

Bioconductor packages

- Graphs and networks:
 - graph, RBGL, Rgraphviz
- Flow Cytometry:
 - flowCore, flowViz, flowUtils
- Protein Interactions:
 - ppiData, ppiStats, ScISI, Rintact
- Sequence Data:
 - Biostrings, ShortRead, rtracklayer, IRanges, GenomicFeatures, VariantAnnotation
- Other data:

Bioconductor: Install packages

```
> source("http://bioconductor.org/biocLite.R")
> biocLite()

#Install Bioconductor package Rintact
> biocLite ("Rintact")
```