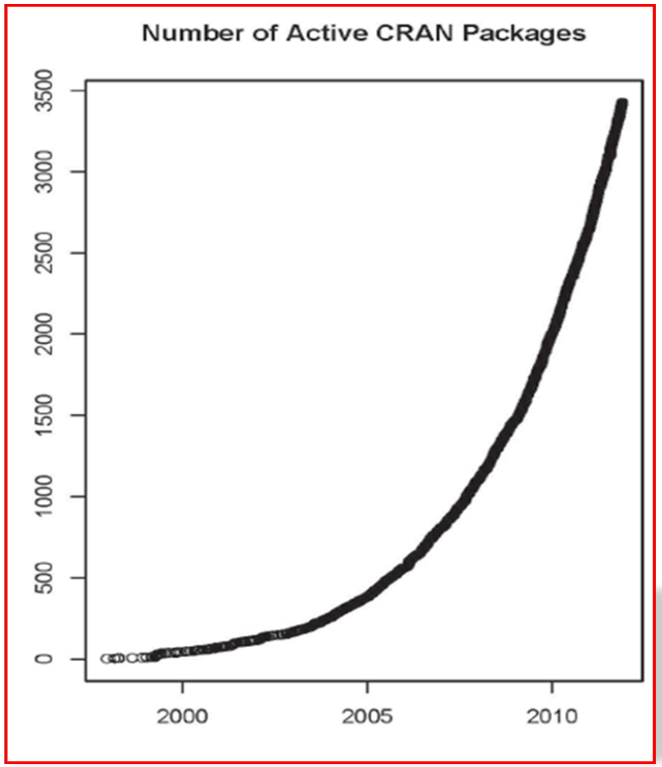
Introduction to R statistical environment

R Nano Course Series

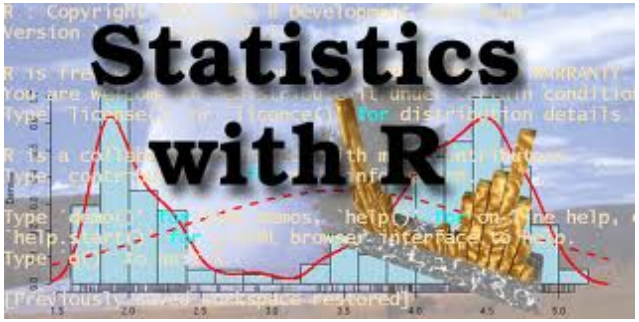
Aishwarya Gogate
Computational Biologist I
Green Center for Reproductive Biology Sciences

History of R

- R is a free software environment for statistical computing and graphics
- R language built as a dialect of the S statistical language (S-plus) developed at Bell Laboratories in mid 70's.
- 2000: R version 1.0.0 was released.
- Quickly became popular for bioinformatics, microarray analysis
- New version released every 6 months
- Now -versions for Windows (32 and 64bit), UNIX/Linux, MacOS, and RStudio (GUI version)



What is R?

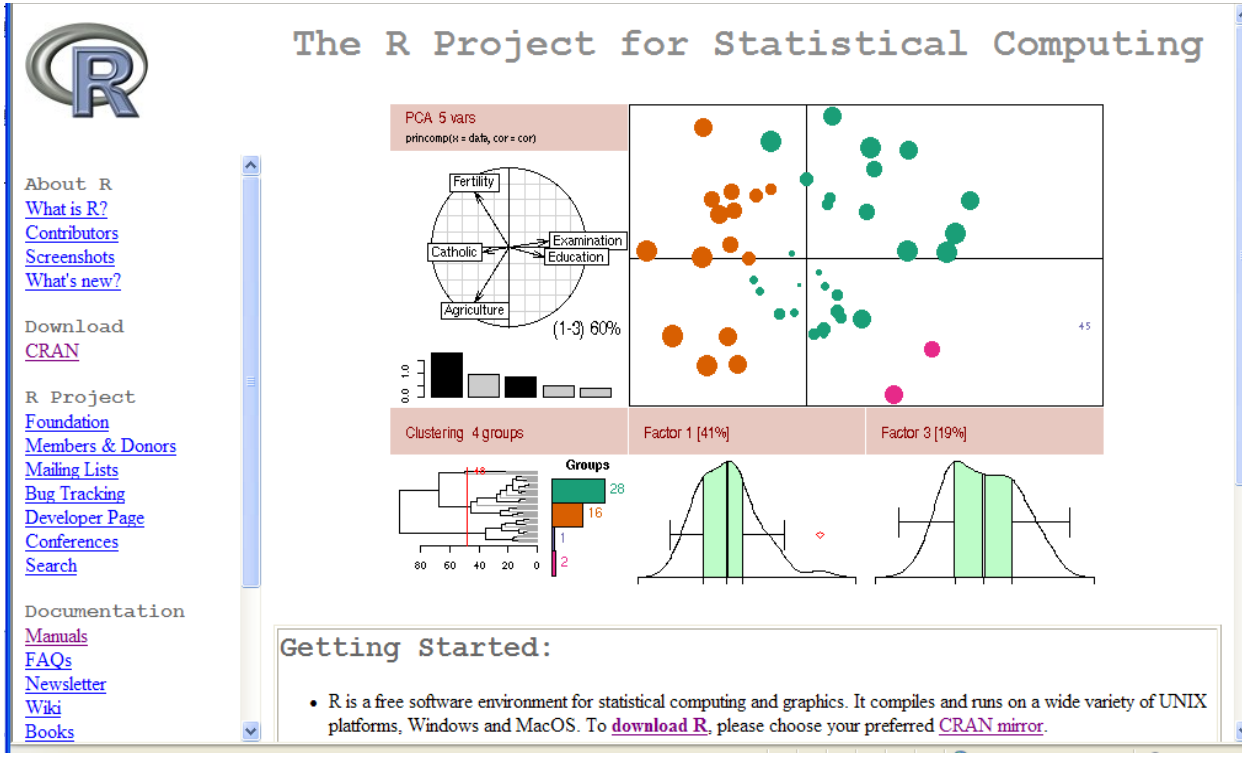


- Object oriented statistical language
- Suite of operators for calculations on arrays and matrices
- Sophisticated graphical facilities for display or output files
- Active R community - R-help and R-devel mailing lists
- ~25 base, or standard, packages
- Thousands of contributed packages in repositories:
  - CRAN: http://CRAN.R-project.org
  - Bioconductor: www.bioconductor.org
  - Many more packages available on personal websites

What I plan to cover in this section

- Introduction to R
- R language – the basics
- Strengths of R
- Working with R objects/ data types
- Simple math functions using different data types
- Using R functions on vectors and matrices
- Subsetting vectors, data frames and matrices
- R packages, Introduction to Bioconductor
- Installing a package (DESeq2)
- Importing a file - Input/output with R – text files (csv, tab-delim, etc)
- Moving between R and Excel

R Website
www.r-project.org



Current version is 3.3.2 (Sincere Pumpkin Patch), released 2016-10-31

Welcome to RStudio

Software, education, and services for the R community

Powerful IDE for R

RStudio IDE is a powerful and productive user interface for R. It's free and open source, and works great on Windows, Mac, and Linux.

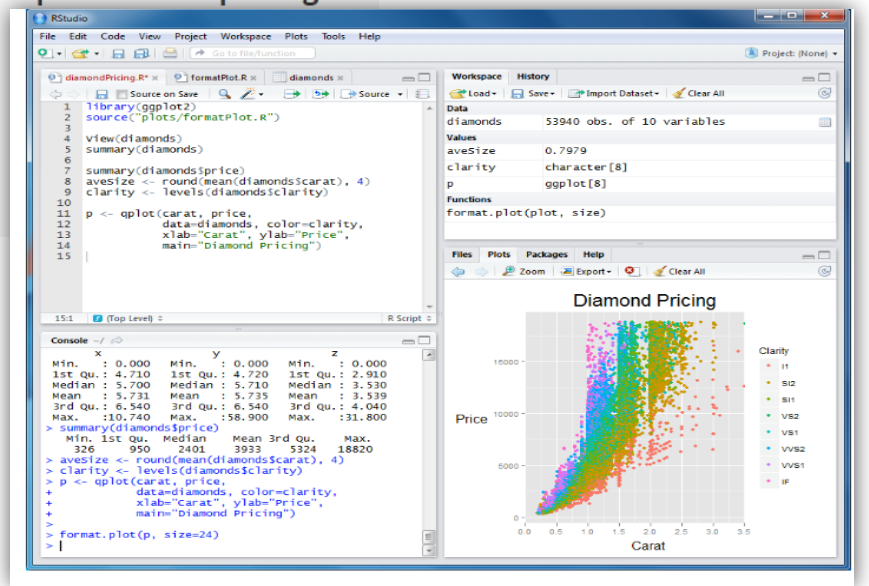
Download now | Learn more

R training and education

We've got hands-on courses for beginners and even R experts. Customize an on-site training or enroll in one of our public workshops.

Request on-site | View courses

Open source R packages



IDE= integrated development environment

Easy to use Interface : Your code, code execution, data that is read into Rstudio and output window (showing plot) - **All in one screen!**

http://www.rstudio.com/

- www.bioconductor.org
- A group of R packages aimed at high-throughput genomic data analysis and genomic annotations (originally microarrays, but now SNP data, RNA-seq, ChIP-seq, other sequencing, mass spec, flow cytometry, etc)
- Open source and Open development
- Each Bioconductor package has a "vignette" for documentation
- Easy to download Bioconductor packages within R:

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

  biocLite() # installs a default set of Bioconductor packages to get started

  biocLite("package.name")
- Packages are grouped as
  - Software (Microarray, visualization, statistics, etc.)
  - AnnotationData (organized by organism,microarray)
  - ExperimentData

– Current version is 3.4 which works with R version 3.3.1 (users with older versions – update installations)

R tutorials

- Under "Manuals" on R website – several in depth tutorials; some basic, some advanced
- Basic introductions to several specific topics in R: http://www.cyclismo.org/tutorial/R/
- Various forums available which discuss ranges of errors that users encounter – **When in doubt, Just Google and get the syntax!**
- Many R books available:
  - General purpose R: e.g., *R Cookbook (2011)*, *R in a Nutshell (2010)*
  - Specific topics*: e.g., Introductory statistics in R,*
    - *Applied Statistical Genetics with R,*
    - *The art of R programming (software design),*
    - *R Graphics Cookbook*
    - *Data Mining with R: Learning with Case Studies*

Getting help in R – Important!

- From the command line:
  - ?function_name (Example: `?mean`)
  - Press `q` to get back to R command line
  - ??keyword (Example: `??mean`)
  - help(function, package=PKG) for a function in a specific package
  - Google it
  - Package-specific help files

Strengths and Weaknesses

- Strengths
  - Strong in statistics
  - Good with matrices
  - Good graphics
  - Access to a wealth of contributed statistical/ computational/ mathematics/ bioinformatics methods
  - Widely used- relatively easy to create and distribute your own R package
- Weaknesses
  - Fairly memory-intensive
  - Not real good for parsing or file handling

Working in R

- Can work interactively (line by line)
- In Batch mode (run a whole file with code at once)
  - Linux Command Line:
    `Rscript filename.r [possible additional arguments]`
    `nohup Rscript filename.r &` (to continue running in background even if you logout)
  - "filename" refers to the R file with the code to run.
- In linux, type `R` in terminal window and hit Enter to launch R
  - Should see some introductory text
  - In Windows, would open the R program with interface
- **Ctrl-C** will stop an R command without exiting R (Ex: to get out of an infinite loop)

Starting off in R

- When working on your own, it's good practice to work with an R script (example, my_R_script.R)
  - Copy code to the console as needed, so all your code can be saved
  - In Linux, you can use any text editor
  - In Windows in R, you have an R script within the program. Simply highlight the code to run and click the *run* button.
  - In Windows connected to a server for R, *Notepad++* is free and supports color coding for programming R.
- Can copy/paste multiple lines of code at a time into the console

Creating variables in R

- The entities R operates on are technically known as *objects.*
- To create variables (objects) in R, use either <- or =
- Examples: `x <- 1` `x = 1`
- This is called making an assignment
- Give objects meaningful names
  - Object names CANNOT start with a number
  - Object names CAN have "." and numbers within them
- Similar to Python, if you try to access an object that hasn't been assigned, R will complain
  - `> cloud`
  - `Error: object "cloud" not found`

Trying some simple R code

- R can be used like a **calculator**

```
> (1+2+3+4+5)/5  # mean of 5 numbers
[1] 3
```

Refers to the index of the first entry printed

```
>  (1+3)*5
[1] 20

>  7/6
[1] 1.166667

>  (7/6)*6
[1] 7
```

Notice that for display, it rounded to a certain number of significant digits. But the true answer is actually calculated and stored.

R Functions

- A function performs some calculation and provides an output
- May take 0, 1 or more parameters as input, separated by commas. Parameter values can be specified directly in the functional call, objects in your workspace, or even calls to another function

```
function_name(param1=x, param2=y)
```

- Can assign the output to an object

```
result <- function_name(param1=x, param2=y)
```

- Often, variable/object names containing data are the first parameter(s) passed to functions – additional options follow

R Functions

- Syntax for the `mean()` function:

  ```
  mean(x, trim=0, na.rm=FALSE)
  ```

- As long as you provide the parameters in the *correct order*, you don't need to type in parameter names
- Example:

  `result <- mean(x=B)` is the same as

  `result <- mean(B)` where B is a numeric vector

- Many functions have default parameters that you don't need to specify in the function call, as long as you want to use the defaults.
- Other parameters are required to be specified

Some Functions do not need any arguments

- To call a function without input parameters, you still need parentheses.

Examples:

- `ls()` # lists all the objects in your workspace
- `quit()` # quits R
- `library()` # lists all installed R packages
- `getwd()` # get your working directory path
- `colors()` # lists all available color names

Trying some more code

- `a <- 2` # assign the number 2 to an object called *a*
- `a` # view the value of the object *a*
- `b <- 3`
- `a+b` # calculate and print the result of a+b
- `a^b` # calculate & print the result of a to the b^{th} power
- Use the function c(), which stands for "combine" to create a vector object called a.vec:
- `a.vec <- c(2,4,6)`
- `a.vec` # View the value of a.vec
- `length(a.vec)` # Get the length of the vector
- `a.vec[1:2]` # Get the first 2 entries of the vector

Creating a vector

```
> b.vec<- c(3,5,7)
```

Add each entry in `a.vec` to each entry in `b.vec`

```
> a.vec + b.vec
[1] 5   9   13
```

Multiple each entry in `a.vec` to each entry in `b.vec`

```
> a.vec*b.vec
[1] 6   20   42
```

R is case sensitive

```
> A.vec
Error: object 'A.vec' not found
```

Creating a matrix

Create a 3 X 2 matrix:

- `a.mat <- matrix(data=c(1,2,3,4,5,6), nrow=3, ncol=2)`
- `a.mat`

Calculate the square of each element in a.mat (Note that this does NOT do traditional matrix multiplication.)

- `a.mat^2`

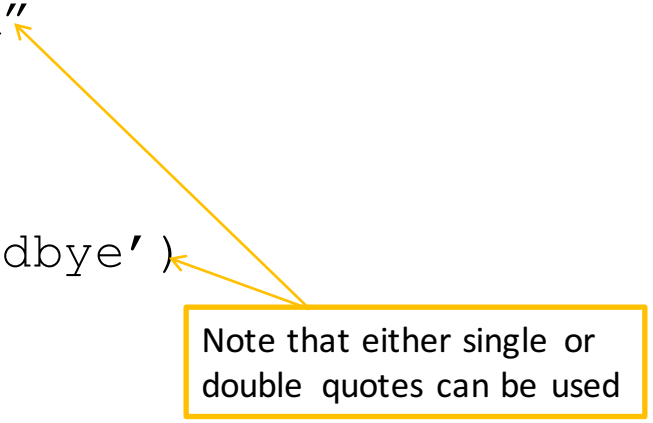
Calculate the log-base 2 of each element in a.mat

- `log2(a.mat)`
- `a.mat[1:2,1:2]` # display the first 2 rows and first 2 columns of the matrix a.mat
- `a.mat[1:2,1]` # display the first 2 rows and first column
- `a.mat[1:2,]` ← **Q: What does this do?**

Trying some more code

Objects can be of types other than numeric, for example characters (strings) or vectors of strings

```
> message <- "Hello world"
> message
[1] "Hello world"
> c.vec <- c('Hello','Goodbye')
> c.vec
[1] "Hello" "Goodbye"
```



Q: What should we type to get just "Goodbye"?

Useful functions for working in R

Objects you create are stored in your R workspace (working directory)

1. `ls()` # lists the objects in your workspace
2. `rm(object1,object2)` # removes object1 & object2 from your workspace
3. `rm(list=ls())` # Completely clears your workspace
4. `save(object1,object2,file="C:/myRobjects.RData")` # saves object1 and object2 to a Rdata file.
5. `save.image("path/my_workspace.RData")` # saves entire workspace
6. `load("C:/myRobjects.Rdata")` # load saved Rdata from a file

Useful functions for working in R

7. `getwd()` # Find where your current working directory is
8. `setwd('C:/workingDirectory')` # Change where your working directory is
9. `quit()` # quit R
10. `library()` # lists all packages available to load
11. `library(package)` or `require(package)` # load installed R package

R mathematical functions

- `x<- 2.1`
- Common mathematical functions
  - `exp(x)` # e to a power
  - `log(x, base=exp(1))` # natural log, or any base
  - `log2(x)` # log base 2
  - `log10(x)` # log base 10
  - `sqrt(x)` # square root
  - `abs(x)` # absolute value
  - `round(x)` # round to the nearest integer
  - `factorial(x)` # factorial of x Try: `factorial(5)`
  - `choose(n, k)` # "n choose k" Try: `choose(5,3)`
  - `cos(), sin(), tan()` # cosine, sine, & tangent

Types of objects

- `x <- c(0,1,0,1)`
- Each *object* has an assigned data-type.
- The most common atomic modes are:
  - logical (TRUE or FALSE) (0 or 1 is also recognized)
    - `logical(length=3)` # *Create* a logical variable of length 3
    - `is.logical(x)` # *Test* whether an object is data-type logical
    - `as.logical(x)` # *Convert* an object to a logical
  - character
    - `character(length), is.character(x), as.character(x)`
  - numeric
    - `numeric(length), is.numeric(x), as.numeric(x)`
  - integer
    - `integer(length), is.integer(x), as.integer(x)`

Types of objects

- Vector (an ordered set of objects) - The objects in a vector could be logical, integer, character, numeric, or even vectors or matrices themselves as in lists.
  - `vector(length=1)` # ***Create*** a logical variable of length 1
  - `is.vector(x)` # ***Test*** whether an object is a vector
  - `as.vector(x)` # ***Convert*** an object to a vector
- Factor - Factors are used to describe items that can have a finite number of values (gender, social class, etc.). Each possible value is called a *level.*
  - `factor(x,…), is.factor(x), as.factor(x)`
- Can also do these 3 functions for:
  - vector, matrix, list, data.frame, function (`function` works somewhat differently), environment

Working with vectors

Three ways to create a vector:

- `my.vec<- c('h','e','l','l','o')`
- `my.vec1 <- seq(from=1,to=10,by=2)`
- `my.vec2 <- 1:4`

Can add to a vector using the c() function:

- `my.vec2<- c(my.vec2, NA)`

Subset a vector:

- `my.vec1[my.vec1<6]`
- `subset(my.vec1,my.vec1<6)` # alternative way

Just for fun (graphics preview):

- `plot(my.vec2,my.vec1)`

Functions to learn about an object

- `length(my.vec)` # length of a vector or list
- `nchar('abcdefg')` # size of a character string
- `dim(a.mat)` # dimensions of a matrix or data frame

  `[1] 3 2` # 1st value is number of rows, 2nd value is number of columns
- `mode(my.vec)` # storage mode, i.e. logical, numeric, character, …
- `typeof(my.vec)` # logical, integer, double, complex, character, etc.

  Similar
- `attributes(my.vec)` # useful for learning about unknown complex objects
  - `Res<-t.test(rnorm(20,1),rnorm(20,2))`
  - `Res` # Display formatted results of t-test
  - `attributes(Res)` # provides names of list values
  - `Res$p.value`

Working with matrices

- `my.mat1<-matrix(1:20,nrow=5,ncol=4)`

Did the matrix get filled in by rows or by columns?

We can use the same method for subsetting that we used for vectors for matrices.

- `my.mat1[!is.na(my.vec2),]`

What does the above line of code do?

A useful way to create a matrix (or data frame) is to combine multiple vectors as columns using the function cbind(), which stands for "column bind".

- `my.mat2<- cbind(1:5,seq(1,10,2))`

cbind() can also be used to combine a matrix or data frame with additional vectors.

Data Frames (Tables)

- A *data frame* object in R is like a table of data. Different columns can be different types of objects (e.g. character and numeric)
- When we read in data from a text file into R, it will be read in as a *data frame* object.
- Like matrices, data frames can have column and row names
- Like matrices, you can easily access or display any subset of rows or columns

Working with data frames

A *data frame* is a tightly coupled collection of variables which share many of the properties of matrices and of lists, and used as the fundamental data structure in most R code.

- `numbers <- 1:4`
- `letters<-c('a','b','c','d')`
- `grp <- as.factor(c(1,0,0,1))`
- `mydata <- data.frame(cbind(numbers, letters, grp))`
- `mydata`
- `mydata[1:2,]` # Commonly used to view top rows of a table

```
  numbers letters grp
1       1       a   1
2       2       b   0
```

- `mode(mydata)`

```
[1] "list"
```

- `?mode` # See for list of possible values

Working with data frames

- `colnames(mydata)` # Get the column headers of the data frame
- `rownames(mydata)` # Get the row names (named by default) of the data frame
- There are 3 different ways to access a column
  - `mydata$letters`
  - `mydata[,"letters"]`
  - `mydata[,2]`
- Display the row(s) where the 2nd column = 'b'

```
mydata[mydata[,2]=='b',]
```

Installing a package

```
> source("http://bioconductor.org/biocLite.R")
> biocLite("DESeq2") # Package for DE analysis of RNA-seq data
> install.packages("ggplot2") # Used to create plots in R
```

Importing a file

- To read in a file

```
x <- read.table(file="/path/to/file/file.csv", sep="\t", header=T)
```

- To check that the file has been properly imported you can :

```
head(x)  # Displays first few rows of the imported file
```

Moving between R and Excel

- Often times there is a confusion about how to open the output files created in R
- The files may be in formats such as .txt, .csv, etc.
- All you have to do : **Right-click -> Open with -> Microsoft Excel**
- Or you can also : **Copy & paste the contents of the .txt file into Excel**

Thank You!

- After the short break…

1. Please have Rstudio open on your computer
2. Open the file **R_Intro_Workshop_code.r** which has been provided to you online