Intro to R and Bioconductor

HMS RESEARCH COMPUTING



What can you do with R?

- R is a statistical language
- R is free!
- Bioconductor thousands of packages for your workflow
- Heavy duty statistics!
- Make nice plots!



Course Objectives

- Learn to run R on O2
- Gain familiarity with R language
- Learn to import/export data
- Simple Statistics/Plotting



Notation

```
[wgr4@compute-a-16-162 ~]$
                                [wgr4@compute-a-16-162 ~]$
                                                           nodule load gcc/6.2.0
O2 Bash
                                [wgr4@compute-a-16-162 ~]$
                                                           nodule load R/4.0.1
                                 [wgr4@compute-a-16-162 ~]$
                                R version 4.0.1 (2020-06-06) -- "See Things Now"
                                Copyright (C) 2020 The R Foundation for Statistical Computing
                                Platform: x86_64-pc-linux-gnu (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.
```



Notation

```
[wgr4@compute-a-16-162 ~]$
[wgr4@compute-a-16-162 ~]$ module load gcc/6.2.0
[wgr4@compute-a-16-162 ~]$ module load R/4.0.1
[wgr4@compute-a-16-162 ~]$ R
R version 4.0.1 (2020-06-06) -- "See Things Now"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (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.
```

Notation

Blue content: try it out!



R on O2

- Open a high-memory R session better than a desktop!
- Log in to O2 with X11 enabled (important for graphics)
- Mac (XQuartz must be installed)
 - ssh -XY user123@o2.hms.harvard.edu
- Linux
 - ssh -XY user123@o2.hms.harvard.edu
- Windows: MobaXterm has X11 client built-in
 - ssh -XY user123@o2.hms.harvard.edu



- SLURM is how we interact with the cluster
- Simple interactive session:

mfk8@login01:~\$ srun --pty -p interactive -t 0-2:00 --mem 8G --x11 -c 1

SLURM is how we interact with the cluster

• Simple interactive session:

mfk8@login01:~\$ srun --pty -p interactive -t 0-2:00 --mem 8G --x11 -c 1

Walltime

(DD-HH:MM)

SLURM is how we interact with the cluster

• Simple interactive session:

mfk8@login01:~\$ srun --pty -p interactive -t 0-2:00 --mem 8G --x11 -c 1

Memory
(units:K|M|G|T)

- SLURM is how we interact with the cluster
- Simple interactive session:

```
mfk8@login01:~$ srun --pty -p interactive -t 0-2:00 --mem 8G --x11 -c 1
```

- SLURM is how we interact with the cluster
- Simple interactive session:

```
mfk8@login01:~$ srun --pty -p interactive -t 0-2:00 --mem 8G --x11 -c 1
```

- SLURM is how we interact with the cluster
- Simple interactive session:

mfk8@login01:~\$ srun --pty -p interactive -t 0-2:00 --mem 8G --x11 -c 1

mfk8@compute-a :~\$

R Versions on O2

mfk8@compute-a:~\$ module spider R

- Why does it matter what version of R you run?
- Downstream packages may only work with certain versions of R.
- How to load a version of R

mfk8@compute-a:~\$ module load gcc/6.2.0 R/version

How to unload a version R

mfk8@compute-a:~\$ module load gcc/6.2.0 R/version

Starting R from an interactive (not login!)

mfk8@compute-a:~\$ R



Managing your R packages on O2

- An R Personal Library is required on O2
- You must create an R Personal Library per version.
- Let's set up an R Personal Library in 3 simple steps!
 - R/4.0.1 (most current version on O2)

Managing your R packages on O2

- Let's set up an R Personal Library in 3 simple steps!
 - R/4.0.1 (most current version on O2)
- 1) Create an R Personal Library directory

```
mfk8@compute-a:~$ mkdir -p ~/R-4.0.1
```

2) Export the R_LIBS_USER variable

```
mfk8@compute-a:~$ export R_LIBS_USER="~/R-4.0.1"
```

• 3) Create .Renviron file

mfk8@compute-a:~\$ echo 'R_LIBS_USER="~/R-4.0.1""> \$HOME/.Renviron

Loading the latest R version on O2

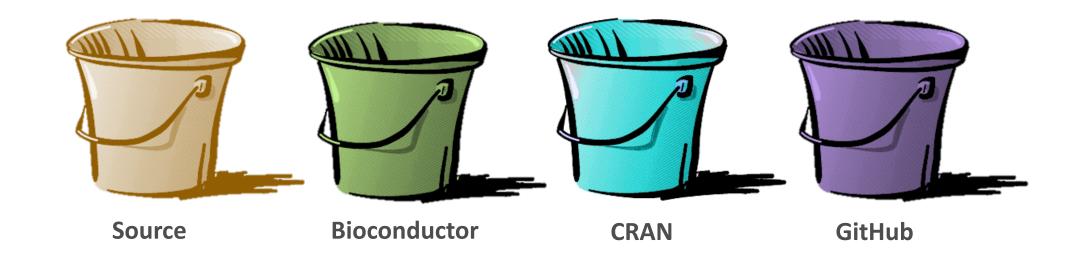
Starting R from an interactive (not login!)

mfk8@compute-a:~\$ module load gcc/6.2.0 R/4.0.1

mfk8@compute-a:~\$ R

Installing Packages

Installing R packages from four general "bins"





Installing packages from Source

- If you must manually download a package, you can place the package inside the R Personal Library directory (e.g., ~/R-4.0.1)
- Accessing packages manually uploaded to your O2 R library (first time)
 - > install.packages("name-of-your-package")



Source

Installing packages from Bioconductor

- "Bioconductor provides tools for the analysis and comprehension of high-throughput genomic data."
- 1) Install BiocManager (only once)
 - > install.packages("BiocManager")
- 2) Install a package from Bioconductor using BiocManager
 - > BiocManager::install("PackageName")



Bioconductor

Installing packages through CRAN

- CRAN is the Comprehensive R Archive Network
- 1) First time install of a package:
 - > install.packages("PackageName")
- · 2) Select a mirror from or near the country you're in



Installing packages through Github

- First, you must install the R package "devtools"
 - > install.packages("devtools")
- Second, load the installed package
 - > library("devtools")
- Finally, use the install_github function
 - > install_github("repo/package")



Loading an R package

- After installing a package, you must use the library function to load it
 - > library("PackageName")



Exercise: Install and load the biomaRt package from Bioconductor

Hint: the BiocManager and library functions are required!

Exercise: Install and load the biomaRt package from Bioconductor

- Hint: the BiocManager and library functions are required!
 - > BiocManager::install("biomaRt")
 - > library("biomaRt")

R documentation

General R help on a function

```
?name_of_function
help(name_of_function)
```

For example:

> ?t.test

Note: press "q" to exit the help view

Setting your R "working directory"

- Prints current working directory:
 - > getwd()
- Setting your WD:
 - > setwd("path")



Setting your R "working directory"

- An O2 example:
 - > setwd("/home/mfk8/MyDataDirectory")

- A Mac example:
 - > setwd("/Users/mfk8/MyDataDirectory")

- A Windows example (note forward slashes):
 - > setwd("C:/Users/mfk8/My Documents/MyDataDirectory")

R Basics

- Keep a file of your commands
 any text editor works
- For example, Notepad++ have syntax highlighting
- Type a variable or object's name to "print" the value(s)
- A line starting with "#" is not passed as an an argument

```
MyRscript.r X

1  # Here, I am using Notepad++,
2  # which highlights commented lines in green
3  # But you proably noticed at this point
4  x <- 1
5  x
6
7  # My function
8  = myfunc <- function(x) {
2  <- x+1
return(z)
}
11
12
13</pre>
```

Data Objects

Variables in R

- Assign variables with a "<-"
- A variable can be overwritten so be careful with naming
- Names can be UPPER/lowercase/./_ mixes, but can't start with a number

```
> myX <- 5
```

> myX

[1] 5

Data Type: Vectors

- Basic way to store data
- "c" stands for concatenate
- "c" can be used to create a vector
- For example:
 - > myvector <- c(3,5,7)
 - > myvector

[1] 3 5 7

Vectors Types

- Numeric:
 - > mynumeric <- c(3,5,7)
- Character (single or double quotes are fine, but be consistent):
 - > mycharacter <- c("bob", "nancy", "jose")
- Logical:
 - > mylogical <- c(TRUE, FALSE, TRUE)

Changing Your Vector Type

- General workflow:
 - > variable <- as.type(variable)
- For example:
 - > myvector <- as.character(myvector)
 - > myvector

```
[1] "3", "5", "7"
```

Changing Your Vector Type

- General workflow:
 - > variable <- as.type(variable)
- For example:
 - > myvector <- as.character(myvector)
 - > myvector

Numbers are now in "quotes" like a character vector!

Changing Your Vector Type

- General workflow:
 - > variable <- as.type(variable)
- For example:
 - > myvector <- as.character(myvector)
 - > myvector

> class(myvector)

Data Type: Lists

- Like vectors with mixed data types (numeric, character, logical)
 - > mylist <- c(3, "TP53", FALSE)
 - > mylist

[1] "3" "TP53" "FALSE"



Data Type: Factors

- Makes a vector nominal (able to be ordered by integers)
- Factors represent an efficient method to store character values
- Create a variable "gender" with 2 "male" entries and 4 "female" entries
 - > gender <- c("male", "male", "female", "female", "female", "female")
- Convert vector as a factor
 - > gender <- factor(gender)
 - > gender
 - [1] male male female female female
 - Levels: female male



Data Type: Matrices

- Data must be all the same type (numeric, character, logical)
- Columns must have the same length
- Creation:
 - > mymatrix <- matrix(c(1:6), nrow=3, ncol=2)
- Indexed by [row,column]

Data Type: Matrices

- Data must be all the same type (numeric, character, logical)
- Columns must have the same length
- Creation:
 - > mymatrix <- matrix(c(1:6), nrow=3, ncol=2)
- Indexed by [row,column]
 - > mymatrix[1,1] #returns item in row 1, column 1
 - > mymatrix[1,] #returns all of row 1
 - > mymatrix[,1] #returns all of column 1

Data Type: Data Frames

- Subset of matrices allowing mixed types
 - > mydataframe <- as.data.frame(mymatrix)
- You can give columns names so you can index by them
 - > names(mydataframe) <- c("column1name", "column2name")
- You can use unique identifiers as rownames
 - > row.names(mydataframe) <- mydataframe[,1]

Dataframes: Indexing/Converting

- matrix or \$ notation are acceptable
- For example (column):
 - > mydataframe\$column1name

-OR-

> mydataframe[,1]

Dataframes: Indexing/Converting

- matrix or \$ notation are acceptable
- For example (row):
 - > mydataframe["rowname1",]

-OR-

> mydataframe[1,]

- To make a data frame into a matrix:
 - > mymatrix <- as.matrix(mydataframe)

Adding and joining rows/columns

- "rbind" to add a row or another df/matrix to a pre-existing dataframe/matrix
 - > mymatrix <- rbind(mymatrix, newrow)
 - > mymatrix <- rbind(mymatrix, matrixtwo)
- "cbind" to add a column or another df/matrix to a pre- existing dataframe/matrix
 - > mymatrix <- cbind(mymatrix, newcol)
 - > mymatrix <- cbind(mymatrix, matrixtwo)

Useful functions:

- > class(object) #gives object class
- > mode(object) #gives object type
- > length(vector) #gives length
- > head(object) #gives first 6 rows
- > tail(object) #gives last 6 rows
- > summary() #quick statistics

- > nrow(object) #gives number of rows
- > ncol(object) #gives number of columns
- > str(object) #gives object structure
- > dim(object) #gives matrix/df dimensions



Missing Values

- NA: Not Available
- NaN: Not a Number
- To identify missing values:
 - > is.na(x) # logical test for NA/NaN
 - > is.nan(x) #logical test for only NaN
 - > x[!is.na(x)] #subsets and excludes NAs

Doing Math

Simple Arithmetic

- > 18 + 22 #addition
- > 18 12 #subtraction
- > 18 * 2 #multiplication
- > 18 / 2 #division
- > 18 %/% 4 #integer part of quotient
- > 18 %% 4 #modulo (remainder)
- > 18 ^ 2 #exponent



Built-in math functions

- > log(10) #natural log (base e)
- > exp(2.302585) #antilog (e raised to power)
- > log10(100) #log base 10
- > sqrt(88) #square root
- > round(log(10), digits=3) #round to specified digits
- > runif(5) #number of random numbers between 0-1
- > rnorm(5) #random numbers from uniform normal distribution
- > abs(18 / -12) #absolute value



Built-in math functions

- > max(object) # max
- > min(object) #min
- > sum(object) #sum
- > mean(object) #mean
- > median(object) #median

- > range(object) #range
- > var(object) #variance
- > sd(object) #standard deviation
- > length(object) #number of values



Series Shortcuts

Series (colon or "seq"):

```
10:1
-OR-Format: seq(from, to, by)> seq(1, 10, 2) # gives odd numbers
```

Repeat

Format: rep(what, times)

> rep(10, 10)

Logical Operations: < > =

- Test of condition: returns logical TRUE/FALSE
 - > test1<- c(1,2,3)
 - > test1 > 2
 - [1] FALSE FALSE TRUE
 - > which(test1 >= 2)
 - [1] 2 3
 - > test1[test1 >=2] #subsetting data based on equality condition
 - > any(test1 >=5) #FALSE
 - > all(test1 >=5) #FALSE



Control Structures



"for" loops

Way to iterate over data

```
> myvector <- c(1,3,5)
> j <- NULL #initialize it, good practice
> newvector <- NULL #initialize it, good practice
> for (i in myvector) { j <- i + 20
    newvector <- c(newvector, j)
}
> newvector
```

Functions

Way to pack up commands into a repeatable format

```
> five <- 5
> three <- 3
> myfunction <- function(x,y) { z <- x + y
    return(z)
}
> myfunction(five,three)
[1] 8
```

Apply

- Returns an object based on applying a function to a dataframe or matrix or list
- Format: apply (to_what, how, function)
- Where how accepts a "1" to apply the function over rows or "2" to apply over columns
- For example:
 - > apply(mymatrix, 1, sum) #row sums
 - > apply(mymatrix, 2, sum) #column sums

Variations on apply

- Fomat: lapply(to_what, function)
 #returns a list
- For example

```
> lapply(1:3, function(x) x^2)
```

[[1]]

[1] 1

[[2]]

[1] 4

[[3]]

[1] 9

- Format: sapply(to_what, function)
 #returns a vector
- For example:
- > sapply(1:3, function(x) x^2)
 [1] 1 4 9



Importing Data



Importing Data: text file

Format: mydata <- read.table(file="PathToFile/filename.csv", header=TRUE, sep=",")

- You can specify how your data is separated (e.g., comma: ","; tab: "\t"; space: "")
- Add "header=TRUE" to make the first row a header containing the column names
- Add "row.names=1" to make column 1 the row names (must be unique identifiers!)





Importing Data from MS Excel

- The "xlsx" package is required
 - > install.packages("xlsx")
 - > library ("xlsx")
- Read in the first worksheet from the workbook myexcel.xlsx
 - Format: mydata <- read.xlsx("FileName.xlsx", sheetIndex=1)
- Read in the worksheet named mysheet
 - Format: mydata <- read.xlsx("FileName.xlsx", sheetName = "mysheet")

Exporting Data

Easy way to export a variable:

Format: write.table(x="ObjectName", file="FileName.tsv", sep="\t")

More options:

row.names=FALSE #turn off row names
col.names=FALSE #turn off column names
quote=FALSE #turn off character string quoting

Saving and Loading your workspace

Save and pick up where you leave off – saves variables

Format: save.image(file="FileName.RData")

-OR-

Format: save(object list, file="FileName.RData")

Load workspace

Format: Ioad(File="FileName.RData")

Class Example



Class Example

- Download class data and R script to a folder from:
 github.com/hmsrc/user-training/tree/master/RBioconductorCourse
- Set your working directory to the folder where your data is located
- O2 cluster and Mac example:
 - setwd("path/to/folder")
- Windows example:
 - setwd("C:/path/to/folder")

<u>Class Sample – Import Data</u>

- Import Rcoursetestdata1.csv as data frame, with headers and row names
 - > mydf <- read.table("Rcoursetestdata1.csv", header=TRUE, row.names=1, sep=",")
 - > head(mydf)

```
head(mydf)
                              TNBC3 Normal1 Normal2 Normal3
ENSG00000008988 15258 15077 144720
                                              43544
                                      12095
                                                       46883
ENSG00000009307 14660 20767
                               8678
                                      13774
                                              23030
                                                       18917
ENSG00000019582 50866 55775
                                                       86319
                              15089
                                       6696
                                              13754
ENSG00000026025 21174 47966
                                              21126
                                                       12728
                              26682
                                       6068
ENSG00000034510 25645 31574
                              56403
                                      29590
                                              25216
                                                       37199
ENSG00000044574 23910 27200
                             13757
                                      13364
                                              10852
                                                       12378
```

Class Sample - Continued

- Get basic statistics on mydf
 - > summary(mydf)

```
summary(mydf)
   TNBC1
                    TNBC2
                                     TNBC3
                                                    Normal1
                Min. :
                                Min.
                                                 Min.
1st Qu.: 7888
                1st Qu.: 9538
                                1st Qu.: 9324
                                                 1st Qu.: 5074
Median : 13034
                Median : 16568
                                 Median : 19108
                                                 Median :10869
Mean : 18596
                Mean : 26036
                                 Mean : 25646
                                                        :14746
                                                 Mean
3rd Qu.: 23850
                3rd Qu.: 28194
                                 3rd Qu.: 30389
                                                 3rd Qu.:18866
      :103007
                       :351603
                                 Max.
                                        :272582
                                                 Max.
                                                        :89837
  Normal2
                   Normal3
                Min. :
Min.
     : 208
1st Qu.: 7124
                1st Qu.: 8944
Median : 14005
                Median : 17710
Mean : 19425
                Mean : 25481
                3rd Qu.: 32191
3rd Qu.: 21576
      :212582
                Max.
                       : 244692
```

Class Sample – Transposing Data

- Need your data to read the other way? Turn it into a matrix, and transpose!
- For example:
 - > mymatrix <- as.matrix(mydf)
 - > myTmatrix<- t(mymatrix) #t = transpose
 - > myTdf <- as.data.frame(myTmatrix) #makes a data frame again
 - > myTdf

Class Example: Summary statistics

- Get basic statistics on transposed data frame
 - > summary(myTdf)

Class Example: Simple t-test

- Do a t-test on TNBC and Normal from mydf
 - > t.test(mydf[,1:3], mydf[,4:6])

Class Example: Simple Wilcoxon

- Do a Wilcoxon test on TNBC and Normal from mydf
 - > wilcox.test(mymatrix[,1:3], mymatrix[,4:6])

Useful R packages

- "RColorBrewer" and "Viridis" define colors and pallets
- "ggplot2" great for plotting
- "genefilter" useful to apply filters over matrices
- "plyr" and "dplyr" advanced matrix/df operations
- "edgeR" and "DESeq2" RNAseq differential analysis alternatives using count data as input
- "biomaRt" cross-annotate samples

Plotting



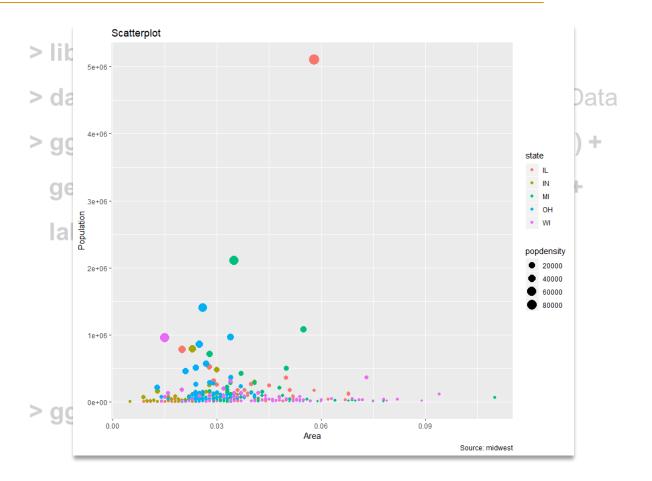
Plotting: short example

- "ggplot2" is an R package for building graphs
- Three general components
 - Data set
 - Coordinate system
 - Geoms

```
> library ("ggplot2") # Load R package
> data("midwest", package = "ggplot2") # Load Data
> gg <- ggplot(midwest, aes(x=area, y=poptotal)) +
 geom_point(aes(col=state, size=popdensity)) +
 labs(y="Population",
   x="Area".
    title="Scatterplot",
    caption = "Source: midwest")
> gg
```

Plotting: short example

- "ggplot2" is an R package for building graphs
- Three general components
 - Data set
 - Coordinate system
 - Geoms
- For more information:
 - Chan Bioinformatics Core https://hbctraining.github.io/Intro-to-Rflipped/lessons/11_ggplot2.html





RStudio

- Feature-rich GUI for R, works on top of version of R installed
- RMarkdown
- Not optimized for multithreading
- Not offered through HMS-RC on O2, but under consideration as a standalone service via Open OnDemand

Contact information

Email: rchelp@hms.harvard.edu

Website: http://rc.hms.harvard.edu

Wiki: https://wiki.rc.hms.harvard.edu/display/O2/O2

Twitter: @hms_rc

Q Location: Gordon Hall 500, 5th Floor, 25 Shattuck Street

https://rc.hms.harvard.edu/office-hours/ for Zoom web conferencing during remote work

Soffice hours: Wednesdays 1-3p for pressing needs, but appointments encouraged.

Please fill out the survey

- Accessible through the Harvard Training Portal
- https://trainingportal.harvard.edu/
- Click on "Me" then "Intro to O2"
- Scroll to "Evaluations" and click on the survey
- We appreciate any feedback or comments!