# Intro to R/Bioconductor

## HMS Research Computing

Fall 2017

Kris Holton

# What can you do with R?

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

# Course Objectives

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

# Notation

- mfk8@compute-a:~$

  O2 bash (your console/terminal)

- >

  In R (either personal computer or in O2 running R)

- Blue content: try it out!

# Bioconductor


- “Bioconductor provides tools for the analysis and comprehension of high-throughput genomic data.”
- Fast, easy way to access packages.
- Source Bioconductor: tells R to look at Bioconductor repository for packages
- Top of script:

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

# Installing packages through Bioconductor

- Available packages:

  http://www.bioconductor.org/packages/release/BiocViews.html#___Software

- Easy install of new packages:

  > biocLite("nameofpackage")
  At the top of every R session using the package, it must be called via:

  > library("nameofpackage")

- Example:
- > biocLite("biomaRt")

  > library("biomaRt")

# Installing packages through CRAN

- CRAN is the *Comprehensive R Archive Network*
- First time install of a package:

  > install.packages("name-of-package")

  (asks you to select a mirror: pick something from or near the country you're in)
- At the top of every R session using the package, the package must be called via:
  > library("nameofpackage")

# Installing packages through Github

- First, install devtools

> install.packages('devtools')

> library(devtools)

- Then, install_github repo/packages

> install_github("repo/package")

# R documentation

- General R help on a function

  > ?name_of_function

  > help(name_of_function)

- Learn how to use your package (vignettes open in your web browser)
  > browseVignettes(package = "name_of_package")

- > ?t.test

# Setting your R “working directory”

- Simplifies paths
- What is your current WD?

  > getwd()
- Setting your WD:

  > setwd(“path”)
- 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:

- To “print” in R
  Just type a variable or object’s name, R will display as much as it can
- “Commenting” in R

  # means what appears afterwards is not passed as an argument
- Keep a file of your commands – anything from Sublime to Notepad++ to Google Drive notepad: have syntax highlighting
- You can copy-paste multiple times, this overwrites.
- Watch quotation marks (often introduced by Microsoft Word/Powerpoint): R gets fussy with certain smart quotes/fonts

# Data Objects

# How R Thinks: Variables

- Assign variables with a <- (traditional) or = (new way)
- 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”: put these together as a vector

```
> myvector <- c(3,5,7)

> myvector

[1] 3 5 7
```

# Vectors Types

- atomic: all the same data type
- numeric:

  > mynumeric <- c(3,5,7)

- character:

  > mycharacter <- c(“bob”, “nancy”, “jose”)

  #note quotation marks “ “ (single or double fine, try to be consistent)

- logical:

  > mylogical <- c(TRUE, FALSE, TRUE)

# Changing Your Vector Type

- General workflow:

  > variable <- as.type(variable)

  > myvector <- as.character(myvector)

  > myvector

  [1] “3”, “5”, “7” #see how these are now in “ “ like a character vector
- Applicable to changing between other types
- Where this comes in handy: when R says you are trying to do an operation on your variable that is one type of vector, when it has to be another type. Use wisely.

# Data Type: Lists

- Like vectors with mixed data types (numeric, character, logical)
- mylist <- c(3, “TP53”, FALSE)
- “unlist”-ing a list tries to coerce the data to an atomic vector of all the same type (lowest common denominator, usually a character)

# Data Type: Factors

- Makes a vector nominal (able to be ordered by integers)
- Create a variable “gender” with 2 "male" entries and 4 "female" entries

  > gender <- c(rep("male", 2), rep("female", 4))

  > gender <- factor(gender) # stores gender as 2 2’s and 4 1’s and associates

  > gender

  [1] male male female female female female

  Levels: female male
- Now 1=female, 2=male internally (alphabetically)
- R now treats gender as a nominal variable

# Data Type: Matrices

- Data must be all the same type (numeric, character, logical)
- Columns must have the same length
- Creation:

```
> mymatrix <- matrix(c(1,2,3,4,5,6), 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 (numeric, character, logical): lists!
- > 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 (no repeats!)

  > row.names(mydataframe)<-mydataframe[,1]

# Dataframes: Indexing/Converting

Can use matrix or $ notation

> mydataframe$column1name #works on column1

> mydataframe[,1] #works on column1

> mydataframe[“rowname1”,] #works on rowname1

> mydataframe[1,] #works on row 1

> mydataframe[-1,] #excludes row 1

- To make a data frame into a matrix for certain operations:

  > mymatrix <- as.matrix(mydataframe) #turns data into all the same type (lowest common denominator is usually character)

# Dataframes: indexing shortcut

- “attach” makes names of lists/dataframes accessible without $

```
> attach(mydataframe)
```

- “detach” when finished

```
> detach(mydataframe)
```

# Adding and joining rows/columns

- “rbind” to add a row or another df/matrix to a pre-existing dataframe/maxtrix

  > 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

> dim(object) #gives matrix/dataframe dimensions

> nrow(object) #gives number of rows

> ncol(object) #gives number of columns

> str(object) #gives object structure

> head(object) #gives first 6 rows

> tail(object) #gives last 6 rows

> summary() #quick statistics

# Missing Values

- Placeholders
- NA: Not Available
- NaN: Not a Number
- is.na(x) is a logical test for NA/NaN
- is.nan(x) is a 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
> factorial(8) #factorial
> choose(12, 8) #combinations (binomial coefficients)
> 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 2

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

- Repeat

```
> 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
> test1 >= 2
[1] FALSE  TRUE  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
- R sometimes breaks with complicated “for” “if” “else” loops
- > 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) #what output is seen outside of function
}
> myfunction(five,three)
[1] 8
> myanswer <- myfunction(five, three); myanswer
[1] 8
```

# Lambda-like functions

- “Anonymous” in-place functions
- Call function on some variable, and use that variable in the calculations

```
> (function (x) x^2)
> plot(function (x) x^2)
```

# Apply

- Returns an object based on applying a function to a dataframe or matrix or list
- Format: apply (to_what, how, function)
- “1” is to apply over rows, “2” is to apply over columns

```
> mymatrix <- apply(mymatrix, 1, function)

> mymatrix <- apply(mymatrix, 2, function)

> apply(mymatrix, 1, sum) #row sums

> apply(mymatrix, 2, sum)  #column sums
```

# Variations on apply

- sapply: (to_what, function) returns a vector

```
> sapply(1:3, function(x) x^2)
[1] 1 4 9
```

- lapply: (to_what, function): returns a list

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

[[2]]
[1] 4

[[3]]
[1] 9
```

# More applies

- mapply (multivariate): pass more arguments

```
> mapply(function, arguments, MoreArgs=X)
```

- by: factors

```
> by(data, factor, function)
```

- replicate: repeat

```
> replicate(repetitions, function(data))
```

- Also tapply, eapply, rapply, vapply
- Useful packages: plyr, dplyr

# Importing Data

# Importing Data: text file

- You can specify how your data is separated (comma separated: “,”  tab: “\t” space: ” “), and if the first row is a “header” row containing the column names)

- mydata <- read.table(file=”PathToFile/filename.csv”, header=TRUE, sep=”,”)
- add “row.names=1 to make column 1 the rownames (only if these are unique identifiers!)
- stringAsFactors=FALSE  converts all to characters

# Importing Data from MS Excel

- Read in the first worksheet from the workbook myexcel.xlsx
- First row contains variable (column) names

  > library(xlsx) #install the first time from CRAN

  > mydata <- read.xlsx("c:/myexcel.xlsx", 1)
- Read in the worksheet named mysheet

  > mydata <- read.xlsx("c:/myexcel.xlsx", sheetName = "mysheet")

# Importing Data from SPSS

- In SPSS: save SPSS dataset in transport format

  get file='c:\mydata.sav'.

  export outfile='c:\mydata.por'.

- in R

  > library(Hmisc) #install the first time from CRAN

  > mydata <- spss.get("c:/mydata.por", use.value.labels=TRUE)

  # last option converts value labels to R factors

# Importing Data from SAS

- In SAS: save SAS dataset in transport format

  libname out xport 'c:/mydata.xpt';

  data out.mydata;

  set sasuser.mydata;

  run;

- In R

  > library(Hmisc) #install the first time from CRAN

  > mydata <- sasxport.get("c:/mydata.xpt")

# character variables are converted to R factors

# Importing Data from STATA

- In R: input Systat file

  > library(foreign) #install the first time from CRAN

  > mydata <- read.systat("c:/mydata.dta")

# Exporting Data

- Easy way to export a variable (vector, dataframe, matrix, etc):

  > write.table(nameofvariable, file="path/nameoffile.tsv", sep="\t") #sep="," or " " etc

- Add

  row.names=FALSE #turn off row names

  col.names=FALSE #turn off column names

  quote=FALSE #turn off character string quoting

# Saving your workspace

- Save and pick up where you leave off – saves variables

  > save.image()

  > save(object list, file=”mysaves.RData”)
- Loading workspaces

  > load(“mysaves.RData”)

# Class Example

# Class Example

- Download class data and R script to a folder from http://hmsrc.me/rclassfiles

Set your working directory to the folder where your data is

- > setwd("pathtofolder/note/forward/slashes")
- A Mac example:

  > setwd("/Users/mfk8/Downloads")
- A Windows example (note forward slashes):

  > setwd("C:/Users/mfk8/Downloads")

# Class Sample – Import Data

- Import Rcoursetestdata1.csv as data frame, with headers and row names

  > mydf <- read.table("Rcoursetestdata1.csv", header=TRUE, row.names=1, sep=",")

  > head(mydf)

| | TNBC1 | TNBC2 | TNBC3 | Normal1 | Normal2 | Normal3 |
|---|---|---|---|---|---|---|
| ENSG00000008988 | 15258 | 15077 | 144720 | 12095 | 43544 | 46883 |
| ENSG00000009307 | 14660 | 20767 | 8678 | 13774 | 23030 | 18917 |
| ENSG00000019582 | 50866 | 55775 | 15089 | 6696 | 13754 | 86319 |
| ENSG00000026025 | 21174 | 47966 | 26682 | 6068 | 21126 | 12728 |
| ENSG00000034510 | 25645 | 31574 | 56403 | 29590 | 25216 | 37199 |
| ENSG00000044574 | 23910 | 27200 | 13757 | 13364 | 10852 | 12378 |

# Class Sample - Continued

- Get basic statistics on mydf

  > summary(mydf)

```
TNBC1          TNBC2          TNBC3          Normal1
Min.   :    0  Min.   :   65  Min.   :   31  Min.   :   22
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  Mean   :14746
3rd Qu.: 23850  3rd Qu.: 28194  3rd Qu.: 30389  3rd Qu.:18866
Max.   :103007  Max.   :351603  Max.   :272582  Max.   :89837
   Normal2         Normal3
Min.   :  208   Min.   :   15
1st Qu.:  7124  1st Qu.:  8944
Median : 14005  Median : 17710
Mean   : 19425  Mean   : 25481
3rd Qu.: 21576  3rd Qu.: 32191
Max.   :212582  Max.   :244692
```

# Class Sample – Transposing Data

- Need your data to read the other way? Turn it into a matrix, and transpose!

  > mymatrix <- as.matrix(mydf)

  > myTmatrix<- t(mymatrix) #t = transpose

  > myTdf <- as.data.frame(myTmatrix) #makes a data frame again

  > myTdf

| | ENSG00000008988 | ENSG00000009307 | ENSG00000019582 | ENSG00000026025 |
|---|---|---|---|---|
| TNBC1 | 15258 | 14660 | 50866 | 21174 |
| TNBC2 | 15077 | 20767 | 55775 | 47966 |
| TNBC3 | 144720 | 8678 | 15089 | 26682 |
| Normal1 | 12095 | 13774 | 6696 | 6068 |
| Normal2 | 43544 | 23030 | 13754 | 21126 |
| Normal3 | 46883 | 18917 | 86319 | 12728 |

# Class Example – Data Ops Cont'd

- Get basic statistics on transposed data frame

  > summary(myTdf)

```
   ENSG00000008988  ENSG00000009307 ENSG00000019582 ENSG00000026025
Min.   : 12095   Min.   : 8678   Min.   : 6696   Min.   : 6068
1st Qu.: 15122   1st Qu.:13996   1st Qu.:14088   1st Qu.:14828
Median : 29401   Median :16789   Median :32978   Median :21150
Mean   : 46263   Mean   :16638   Mean   :38083   Mean   :22624
3rd Qu.: 46048   3rd Qu.:20305   3rd Qu.:54548   3rd Qu.:25305
Max.   :144720   Max.   :23030   Max.   :86319   Max.   :47966
```

# Class Example: Simple t-test

- Do a t-test on TNBC and Normal from mydf

  > t.test(mydf[,1:3], mydf[,4:6])

```
 Welch Two Sample t-test
data:  mydf[, 1:3] and mydf[, 4:6]
t = 2.3053, df = 1168.9, p-value = 0.02132
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
  527.5122 6556.6411
sample estimates:
mean of x mean of y
 23426.08  19884.01
```

# Class Example: Simple Wilcoxon

- wilcox.test(mymatrix[,1:3], mymatrix[,4:6])

Wilcoxon rank sum test with continuity correction

data:  mymatrix[, 1:3] and mymatrix[, 4:6]

W = 199820, p-value = 0.0009604

alternative hypothesis: true location shift is not equal to 0

# Class Example: Linear Modelling

- model<-lm(y ~ x1 + x2 …, data=data)

```
> mymodel  <- lm(TNBC1 ~ Normal1, data=mydf)
> anova(mymodel)
Analysis of Variance Table


Response: TNBC1
           Df     Sum Sq    Mean Sq F value    Pr(>F)
Normal1     1 5.4597e+09 5459744483  20.215 1.177e-05 ***
Residuals 198 5.3477e+10  270084874
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

# Class Sample: Practical Plotting

# Saving plots

- Use “File->”save as” or,
- Open up a plot file (png, pdf, jpeg, tiff, bmp)

  > png(file=“nameyourfile.png”)
- Make your plot

  > plot()
- Turn plot off

  > dev.off()

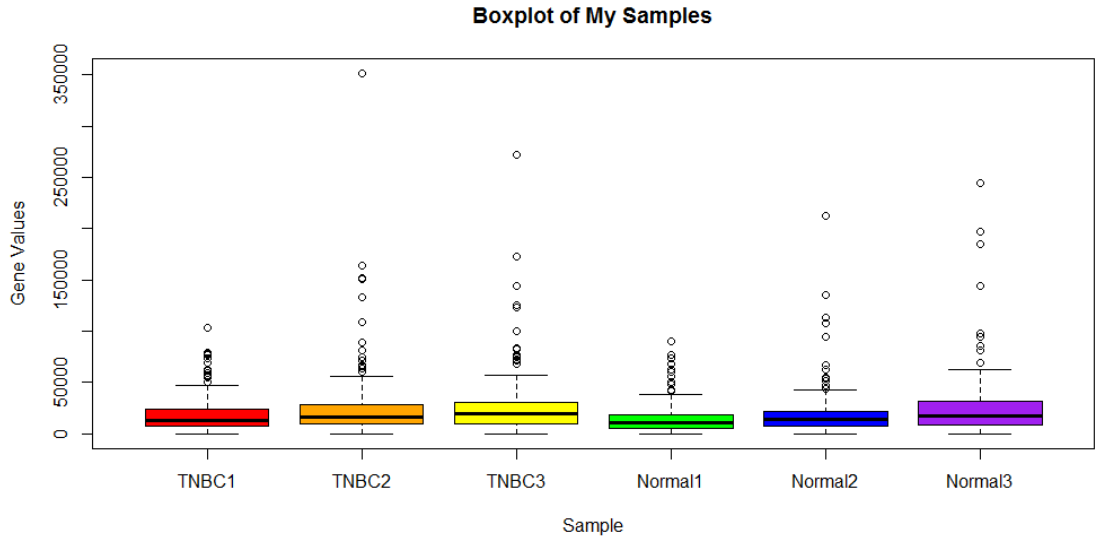
# Plotting Options

- main = “Title” #title
- xlab= “x label” #x-axis label
- ylab=“y label” #y-axis label
- xlim(N,N) #x-axis start, stop
- ylim(N,N) #y-axis start, stop
- col =c(“color1”, “color2”) #vector with colors
- lty = c(N, N) #line type
- lwd= c(N, N) #line width
- cex = N #size of text and symbols
- pch = N #plot point symbol type
- par(mfrow(x,y)) #multiple figures in one plot

# Class Example – Boxplot of Samples

- Boxplot of Samples

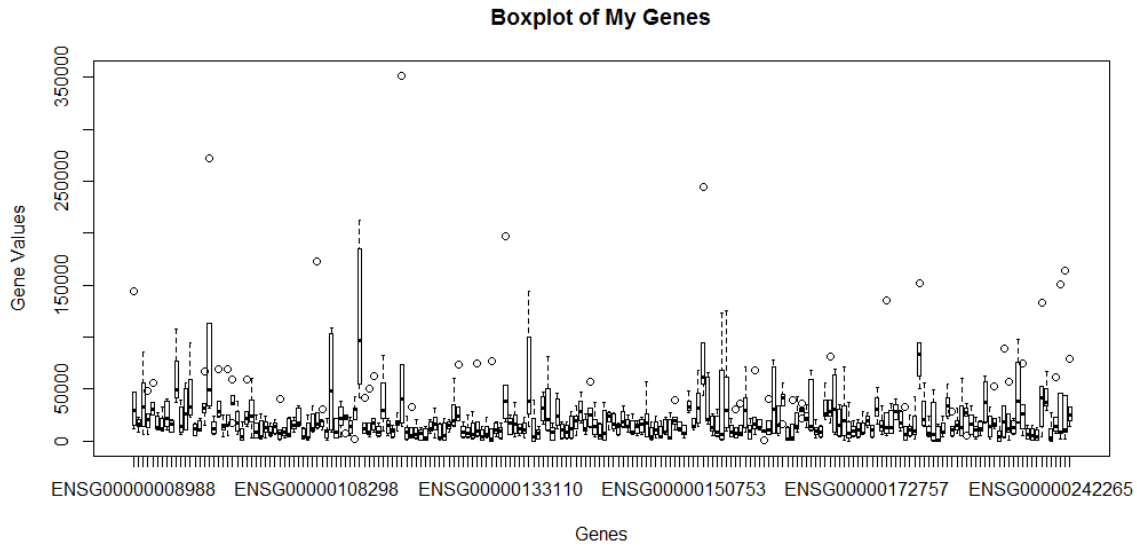
  > boxplot(mydf, main="Boxplot of My Samples", xlab="Sample", ylab="Gene Values")



# Class Example – Boxplot of Genes

- Boxplot of Genes

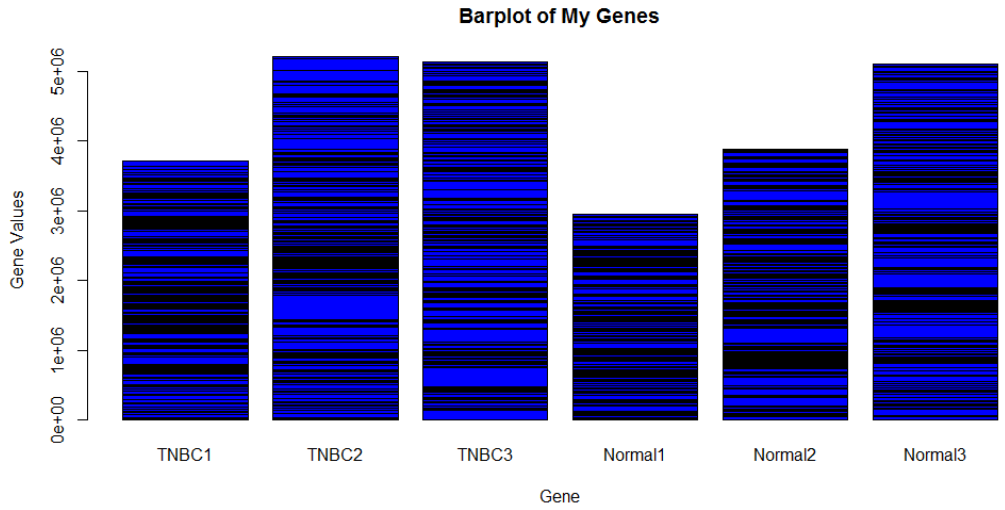
  > boxplot(myTdf, main="Boxplot of My Genes", xlab="Gene", ylab="Gene Values")



# Class Example – Barplot of Genes

- Barplot of Genes #barplot needs a matrix

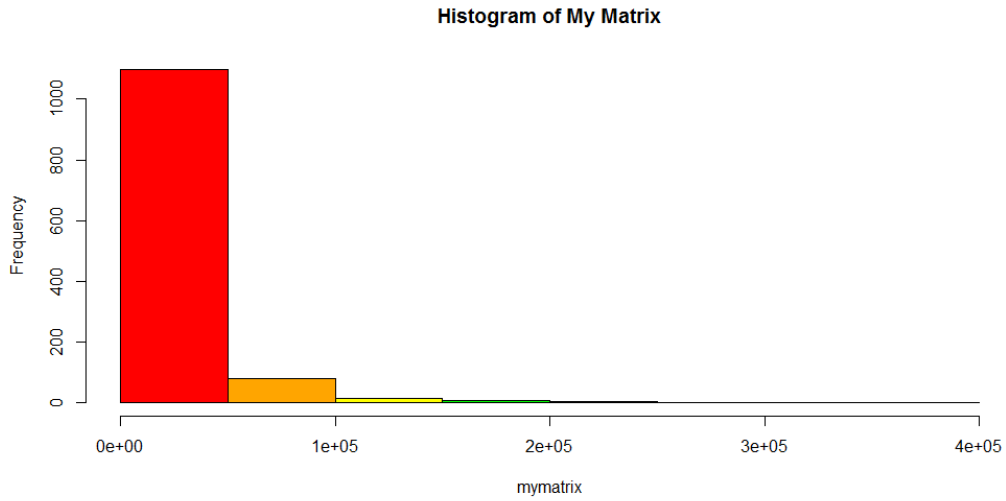
  > barplot(myTmatrix, main="Barplot of My Genes", xlab="Gene", ylab="Sample Values")



# Class Example – Histogram of Values

- Plot a histogram of the frequency of values in mymatrix

  >hist(mymatrix)



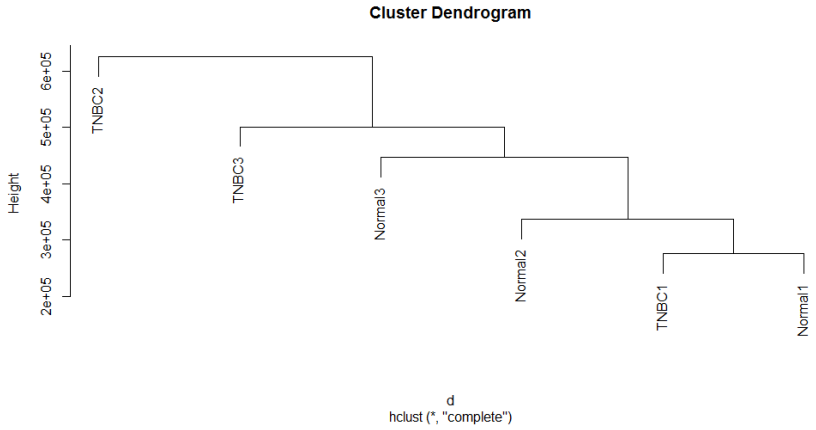
# Class Example - HCL

- Do hierarchical clustering of samples

  > d <- dist(myTmatrix) #takes matrix as input, calculates distance

  > hc <- hclust(d) #performs HCL on distance matrix

  > plot(hc)



# Class Sample – Line Graphs

- #start a line plot with ENSG00000008988 "b" means both points and lines

  > plot(myTdf$ENSG00000008988, type="b", col="green", ylim=c(10000,150000), main="Gene Values Over Samples", xlab="Sample", ylab="Gene Values")

- #add a new line for ENSG00000009307 "lines" adds a line to the current plot

  > lines(myTdf$ENSG00000009307, type="b", col = "blue")

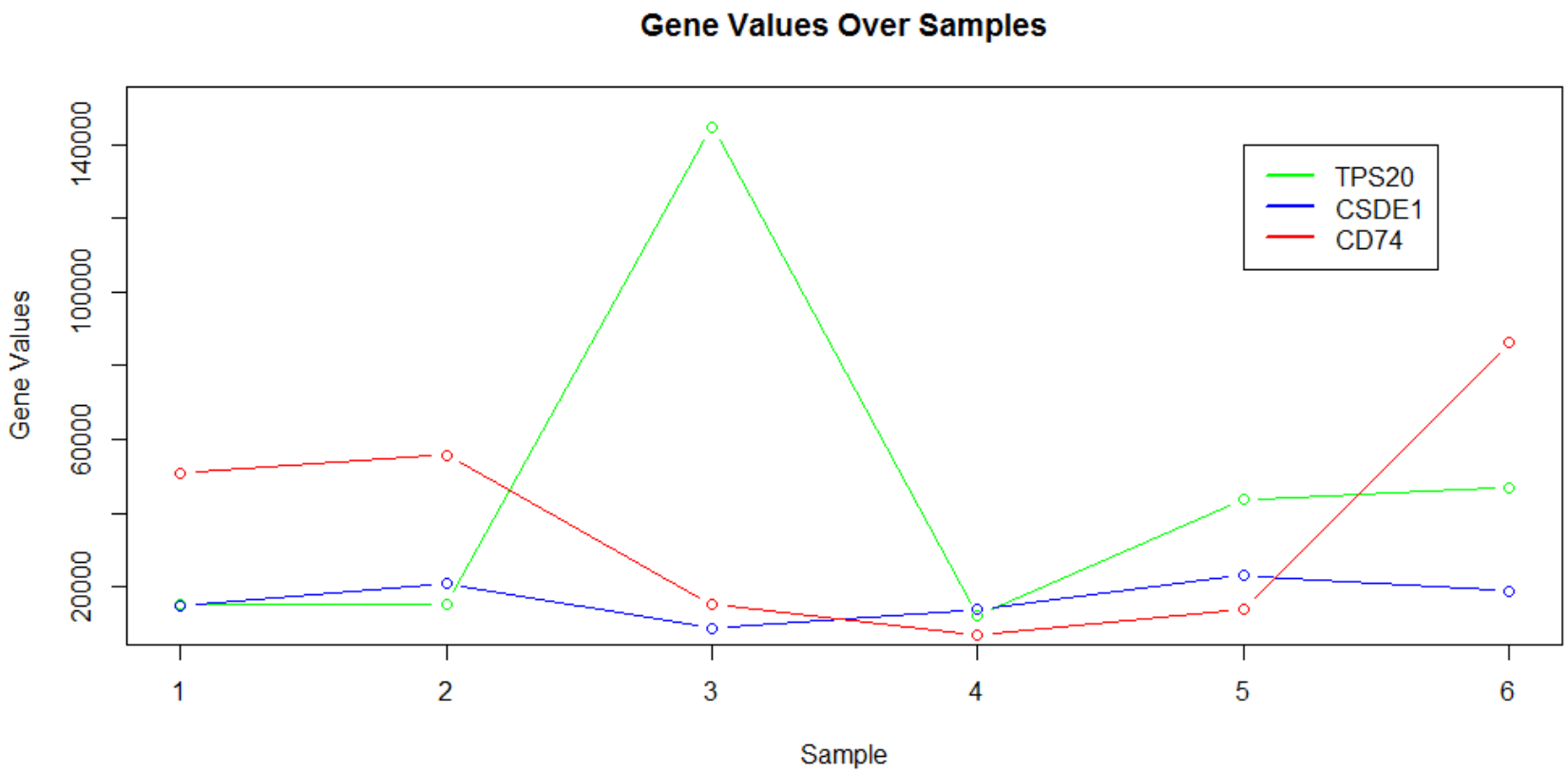
- #add a new line for ENSG00000019582

  > lines(myTdf$ENSG00000019582, type="b", col="red")

# Class Sample – Line Graphs Legend

```
> legend(5, 140000, #positions x, y
c("TPS20", "CSDE1", "CD74"), #line names
lty=c(1,1), #specifies lines
lwd=c(2.5,2.5), #specifies line width
col=c("green", "blue", "red") #add colors
) #ends legend
```

# Class Sample – Line Graphs



# Class Example – Count Data

- Import count data

```
> mytable<-read.table("Rcoursetestdata2.csv", header=T, row.names=1, sep=",")
> mytable
```

| | neversmoke | smoke | pastsmoke | gender |
|---|---|---|---|---|
| 1 | 0 | 1 | 0 | male |
| 2 | 0 | 1 | 0 | female |
| 3 | 0 | 0 | 0 | male |
| 4 | 1 | 0 | 0 | female |
| 5 | 0 | 0 | 1 | male |
| 6 | 0 | 1 | 0 | female |
| 7 | 0 | 0 | 1 | male |
| 8 | 0 | 0 | 1 | female |
| 9 | 1 | 0 | 0 | male |
| 10 | 0 | 0 | 1 | female |

# Class Example – Count Data Cont'd

- Get summary statistic on mytable

> summary(mytable)

```
> summary(mytable)
  neversmoke      smoke       pastsmoke      gender
 Min.   :0.0   Min.   :0.00   Min.   :0.0   female:5
 1st Qu.:0.0   1st Qu.:0.00   1st Qu.:0.0   male  :5
 Median :0.0   Median :0.00   Median :0.0
 Mean   :0.2   Mean   :0.30   Mean   :0.4
 3rd Qu.:0.0   3rd Qu.:0.75   3rd Qu.:1.0
 Max.   :1.0   Max.   :1.00   Max.   :1.0
>
```

# Class Example – Tables Cont'd

- Take a subset of a table to do statistics on

```
> genderVsmoke <- table(mytable$gender, mytable$smoke)
> genderVsmoke
        0 1
female 3 2
male   4 1
```

# Class Example – Tables Cont'd

- summary(genderVsmoke) does a Chi Square Test of Independence

  > summary(genderVsmoke)

Number of cases in table: 10

Number of factors: 2

Test for independence of all factors:

Chisq = 0.4762, df = 1, p-value = 0.4902

Chi-squared approximation may be incorrect

# Class Example – Tables Cont'd

- Get the frequencies of a table via prop.table

  > propgenderVsmoke <- prop.table(genderVsmoke)

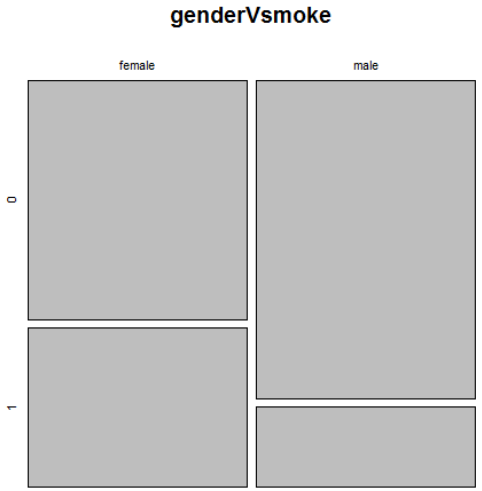
  > propgenderVsmoke

  | | 0 | 1 |
  |---|---|---|
  | female | 0.3 | 0.2 |
  | male | 0.4 | 0.1 |

# Class Example – Plotting Tables

- Do a mosaicplot of table genderVsmoke

  > mosaicplot(genderVsmoke)



# Bonus Plot: Heatmap

- Try installing packages “gplots” and “RColorBrewer”

  > install.packages(“gplots”, “RColorBrewer”)
- Call the libraries

  > library(“gplots”)

  > library(“RColorBrewer”)
- Check out the code!

R on 

# 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 installed, in console

  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 and O2

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

```
mfk8@login01:~$ srun --pty –p interactive –t 0-12:00 --mem 8G bash
```

(where 8G is memory requested)

- Graphics:

srun: add `--x11`

sbatch: add `--x11=batch`

- parallel and BiocParallel libraries: run over multiple cores (-n up to 20 cores)
- Rmpi: run R scripts over multiple nodes (>20 cores)

# R Versions

- mfk8@login01:~$ 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@login01:~$ module load R/version
- Unloading R

  module unload R/version
- Starting R from an interactive (not login!)

  mfk8@compute-a:~$ R

# Managing your R packages on O2

- It is best to manage your own R packages to work with the version of R you select. In doing so, there are no disruptions to your workflow.
- Setting up your O2 R library:

  mfk8@login01:~$ mkdir -p ~/R-version

  mfk8@balcony:~$ export R_LIBS_USER="~/R-version"

  mfk8@balcony:~$ echo 'R_LIBS_USER="~/R-version"'> $HOME/.Renviron
- If you must manually download a package (not through Bioconductor/CRAN etc), put the package in the set up location (/home/user123/R-version)
- Accessing packages manually uploaded to your O2 R library (first time)

  > install.packages("name-of-your-package",lib="~/R-version")

# Useful R packages

- “RColorBrewer” – define colors and pallets
- “gplots” and “ggplots2” – great for plotting
- “genefilter” – useful to apply filters over matrices
- “edgeR” and “DESeq2” – RNAseq differential analysis alternatives using count data as input
- “cummeRbund” – front end to extract, manage, and plot cuffDiff
- “biomaRt” – cross-annotate samples

# RStudio


- Feature-rich GUI for R, works on top of version of R installed
- RMarkdown, Rshinyapps
- Not optimized for multithreading
- Good for proofing code, but not a scalable solution for HPC
- Not currently available as an O2, but under consideration

# Contact Information

- HMS Research Computing
- rchelp@hms.harvard.edu
- Office Hours Wed 1-3p 500 Gordon Hall
- http://rc.hms.harvard.edu/