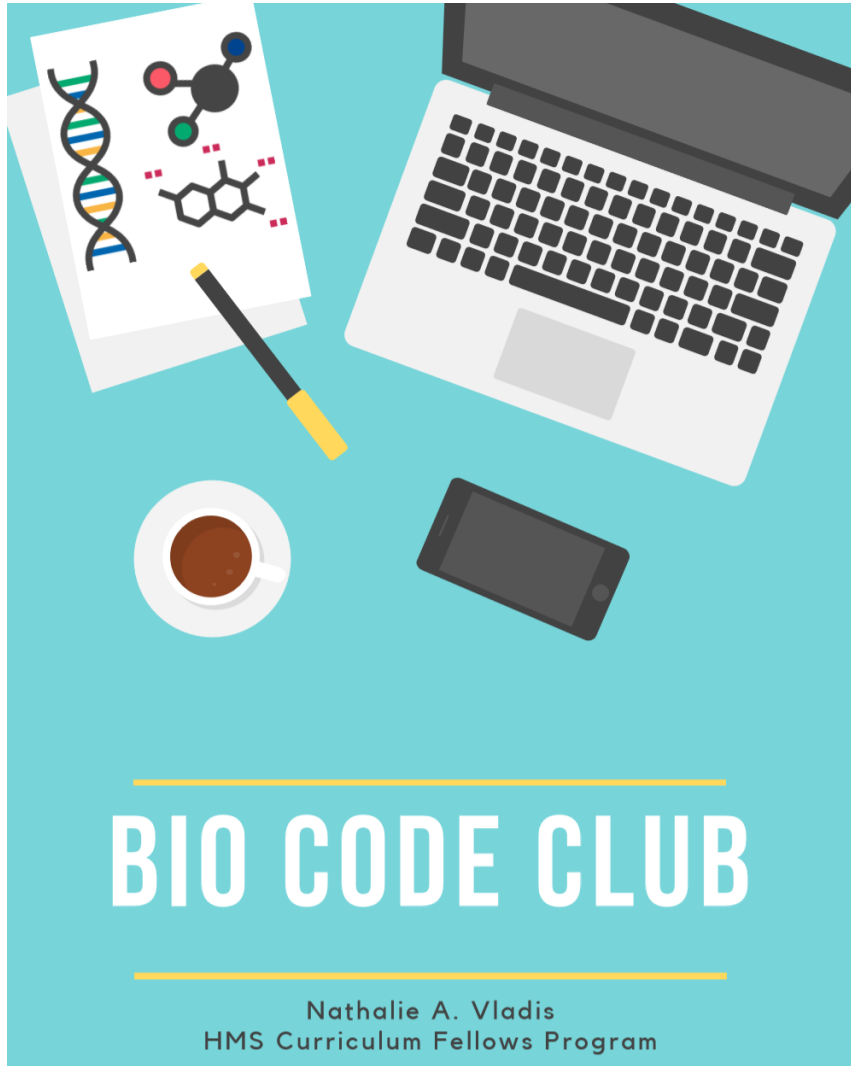


Introduction to R & Bioconductor

NATHALIE A. VLADIS (WITH MATERIALS FROM KRIS HOLTON)

QUANTITATIVE CURRICULUM FELLOW

ORGANISED BY **HMS RESEARCH COMPUTING**



**Keep an eye out for
Bio Code Club!**

**Last Wednesday of
each month during
the Summer 4-5 pm
TMEC 304**

**Once a week in
TMEC starting
September
Time & Venue TBC**

Getting to know you!

PollEv.com/nathalievlad223



R INTRO OBJECTIVES

- Familiarize ourselves with R Studio and some fundamental R commands
- Identify some key R objects that will help us store & manipulate data
- Use some popular mathematical R functions
- Discover R's potential through a class example

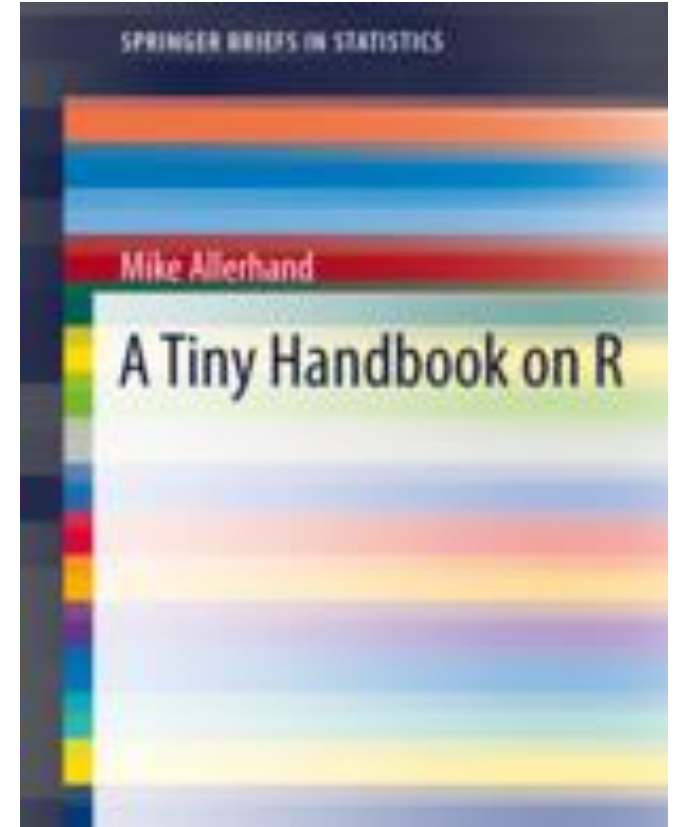


WHY R?

- Its Free!
- Open-source license (anyone can download and modify the code)
- Runs everywhere
- Huge Community and Support
- Very popular amongst biologists

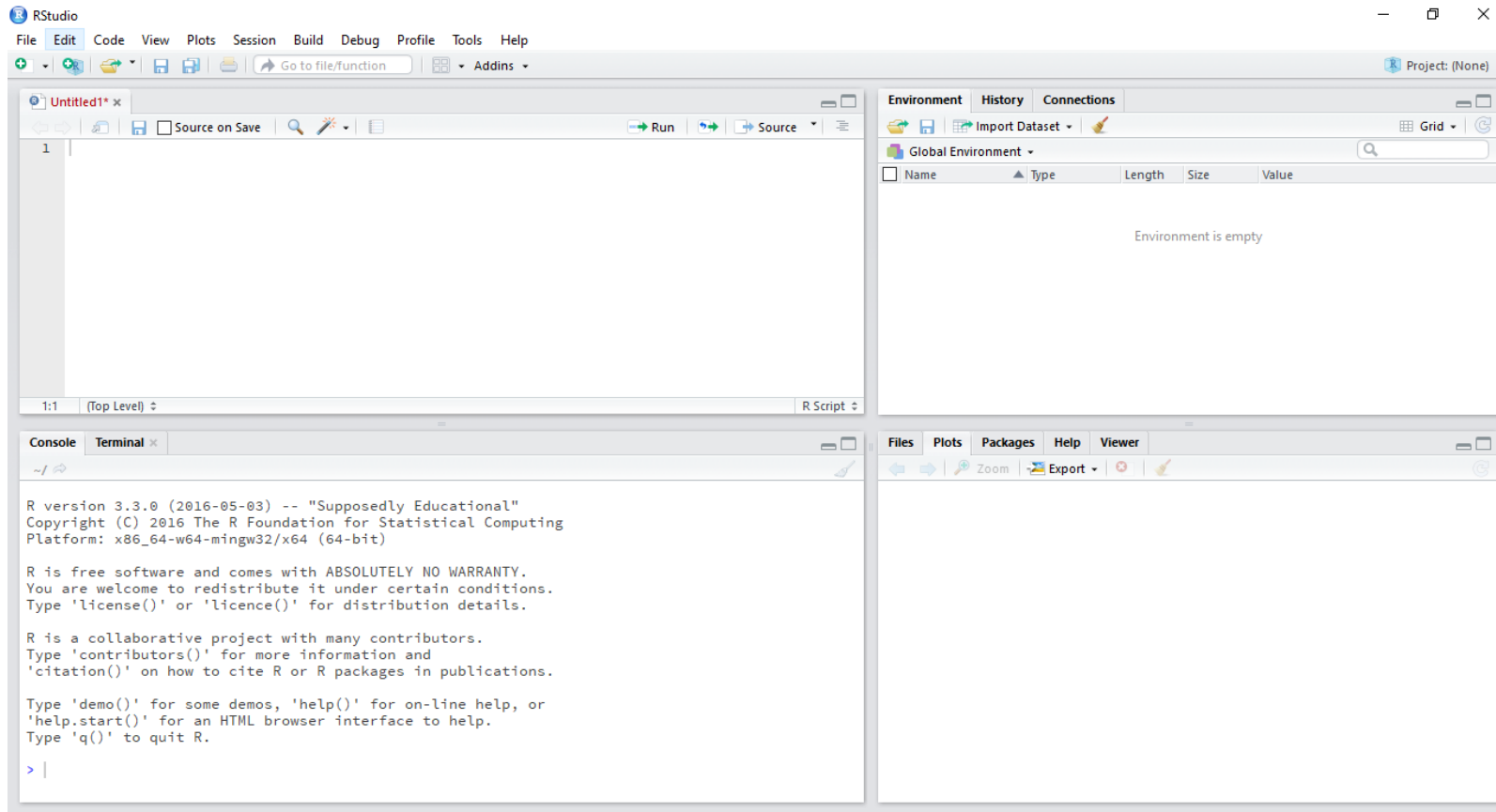
GETTING STARTED WITH R & R STUDIO



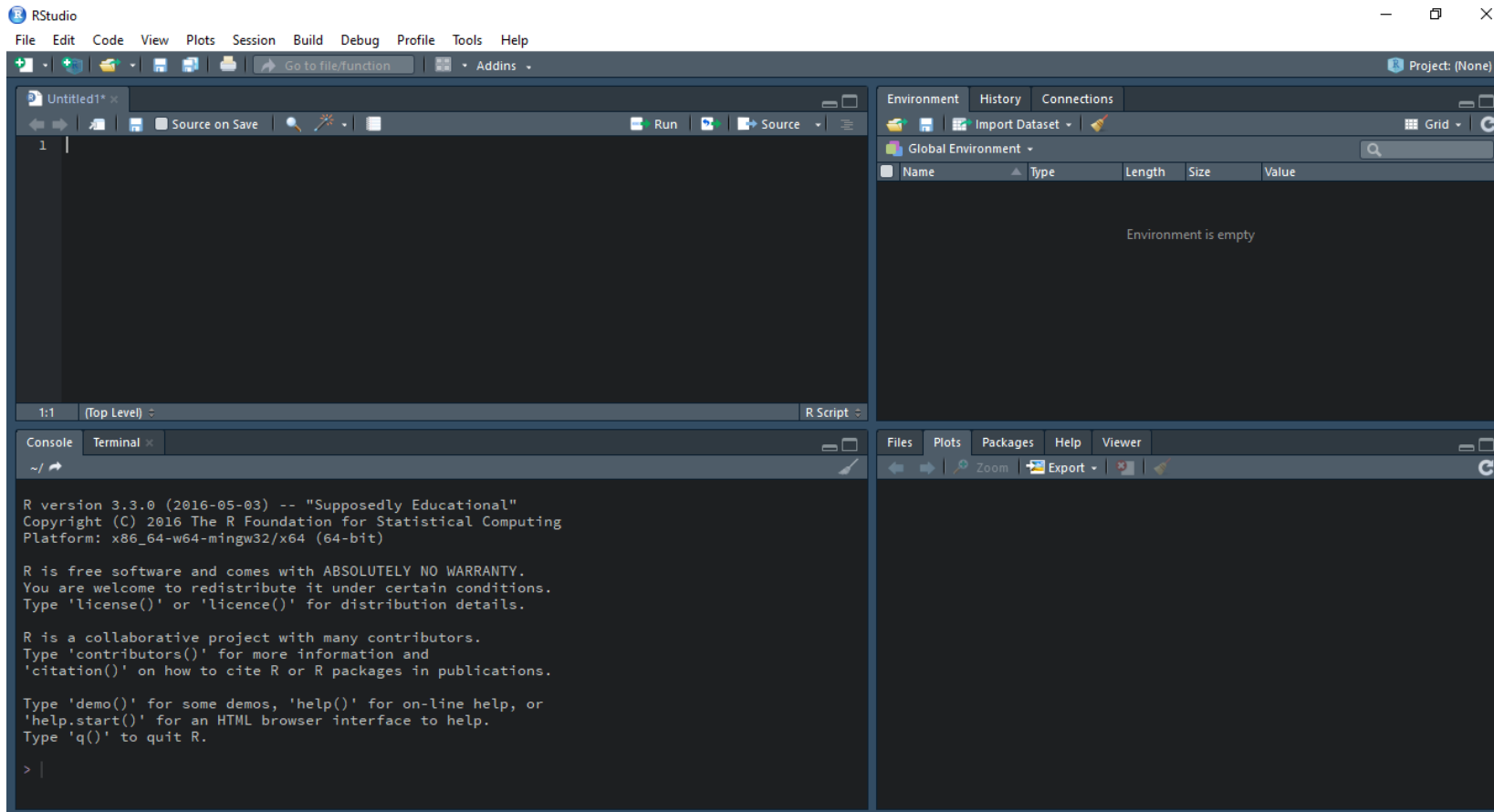


WELCOME TO R STUDIO |

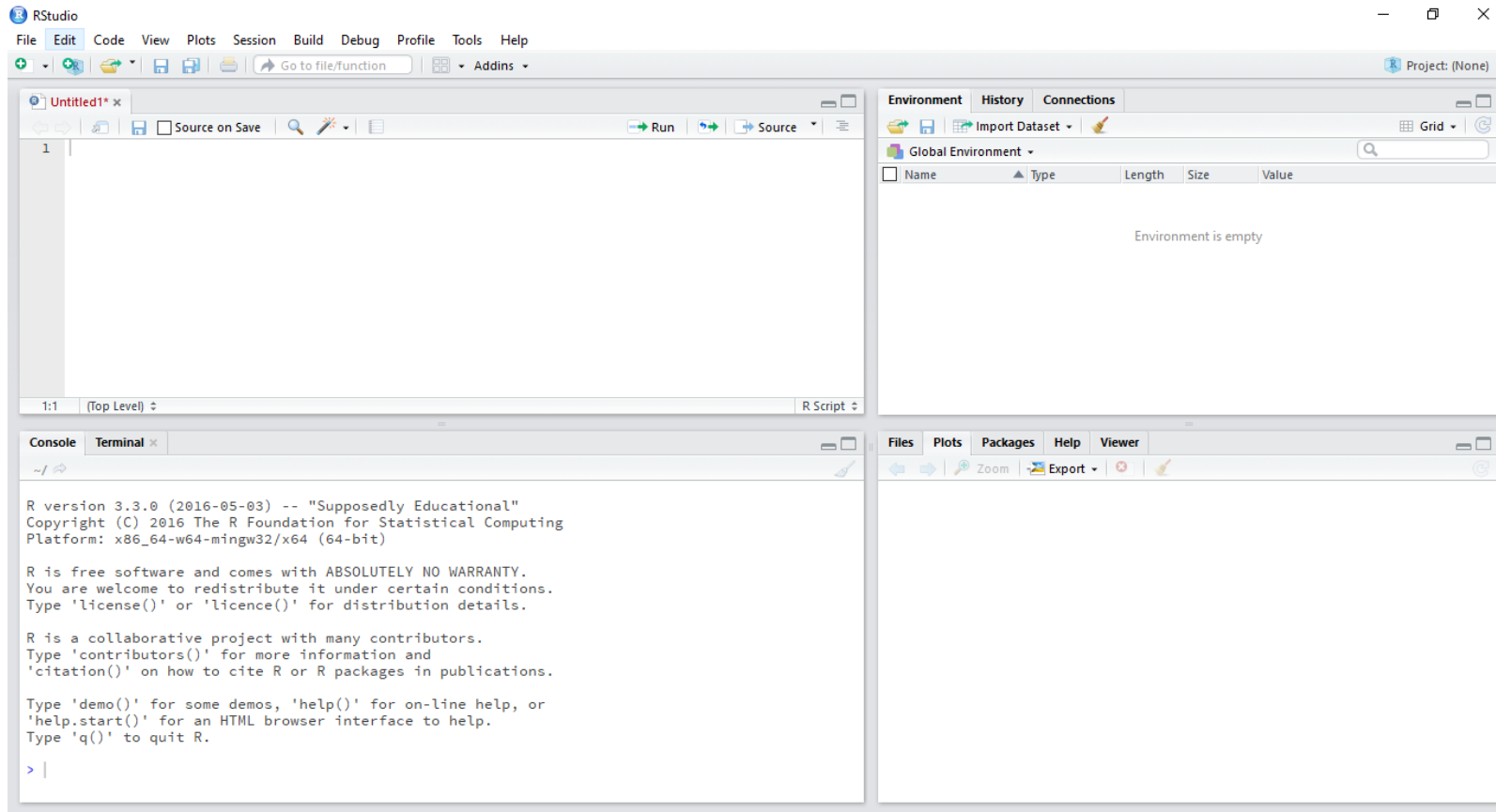
WELCOME TO R STUDIO



WELCOME TO R STUDIO!



WELCOME TO R STUDIO



SOME BASIC SYNTAX

- 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 computed

Your best friend when you write long scripts!


- You can copy-paste multiple times, this overwrites
- Often “ and ‘ are used interchangeably – Be as consistent as you can!

FINDING & READING DATA



.CSV FILES

- Stands for comma-separated values
- A delimited text file that uses a comma to separate values
- A CSV file stores tabular data (numbers and text) in plain text
- One of the most commonly used file formats for data storage in the biomedical sciences



| | A | B | C | D | E |
|---|---|--------|-----|-----|------|
| 1 | | Name | Sex | Bwt | Hwt |
| 2 | 1 | Sadie | F | 2.3 | 11.2 |
| 3 | 2 | Maggie | F | 2.4 | 6.3 |
| 4 | 3 | Luna | F | 2.4 | 8.7 |
| 5 | 4 | Ginger | F | 2.4 | 8.8 |
| | 5 | Tesla | F | 2.4 | 10.2 |
| | 6 | Bibi | F | 2.5 | 9 |



```
,Name,Sex,Bwt,Hwt,Coat,Age,  
1,Sadie,F,2.3,11.2,White,3,  
2,Maggie,F,2.4,6.3,Tabby,1,  
3,Luna,F,2.4,8.7,Black,5,F/  
4,Ginger,F,2.4,8.8,Gir  
5,Tesla,F,2.4,10.2,Tab  
6,Bibi,F,2.5,9,Calico,
```



READING DATASETS WITH READ.CSV()

- First check your working directory!

```
> read.csv("mydataset.csv")
```

```
# Read a file in the working directory
```

```
> read.csv(file.choose())
```

```
# File locator
```

Tip no 1: Do not forget to use quotation marks!

Tip no 2: Check your operating system! Syntax will differ from Mac to Windows to Linux.

INSPECTING YOUR WORKSPACE

- > `getwd()`
- > `setwd("your path")`
- > `library()` # Lists the packages installed on your computer
- > `library("package_name")` # Loads packages into your session
- > `sessionInfo()` # Lists the packages loaded into memory

```
> library("MASS")
> sessionInfo()
R version 3.5.1 (2018-07-02)
Platform: x86_64-w64-mingw32/x64 (64-bit)
Running under: windows >= 8 x64 (build 9200)

Matrix products: default

locale:
 [1] LC_COLLATE=English_United Kingdom.1252  LC_CTYPE=English_United Kingdom.1252  LC_MONETARY=English_United Kingdom.1252
 [4] LC_NUMERIC=C                            LC_TIME=English_United Kingdom.1252

attached base packages:
 [1] stats      graphics  grDevices  utils      datasets  methods    base

other attached packages:
 [1] MASS_7.3-50

loaded via a namespace (and not attached):
 [1] compiler_3.5.1 tools_3.5.1
```



PATHS

If these formats don't work for you, try:
> setwd("C:\\\\Users\\mkf8\\Downloads")

- 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")`

Console

Terminal x

~/

R version 3.3.0 (2016-05-03) -- "Supposedly Educational"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (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.

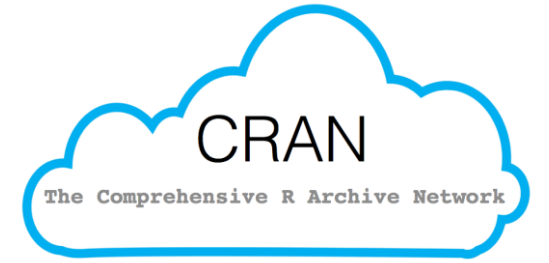
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

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]

> |

INSTALLING PACKAGES FROM CRAN



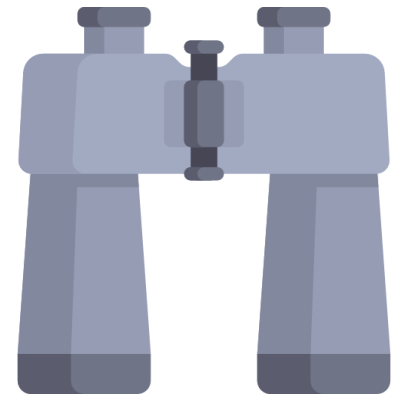
```
> install.packages() # Download and install packages  
  
> install.packages("ggplot2") # Download and install package "ggplot2"
```



FINDING FUNCTIONS WITH “APROPOS”

> `apropos("^read")` # Search for function names starting with “read”

> `apropos("\\read$")` # Search for function names ending with “read”



GETTING HELP

- > `help.start()` # Manuals and reference guides
- > `help(t.test)` # Display the help page for function 't.test'
- > `?t.test` # ... a shorthand for the same thing
- > `args(t.test)` # Displays the argument names and corresponding default values of a function



FUNCTION ARGUMENTS

- **CONSOLE INPUT:**

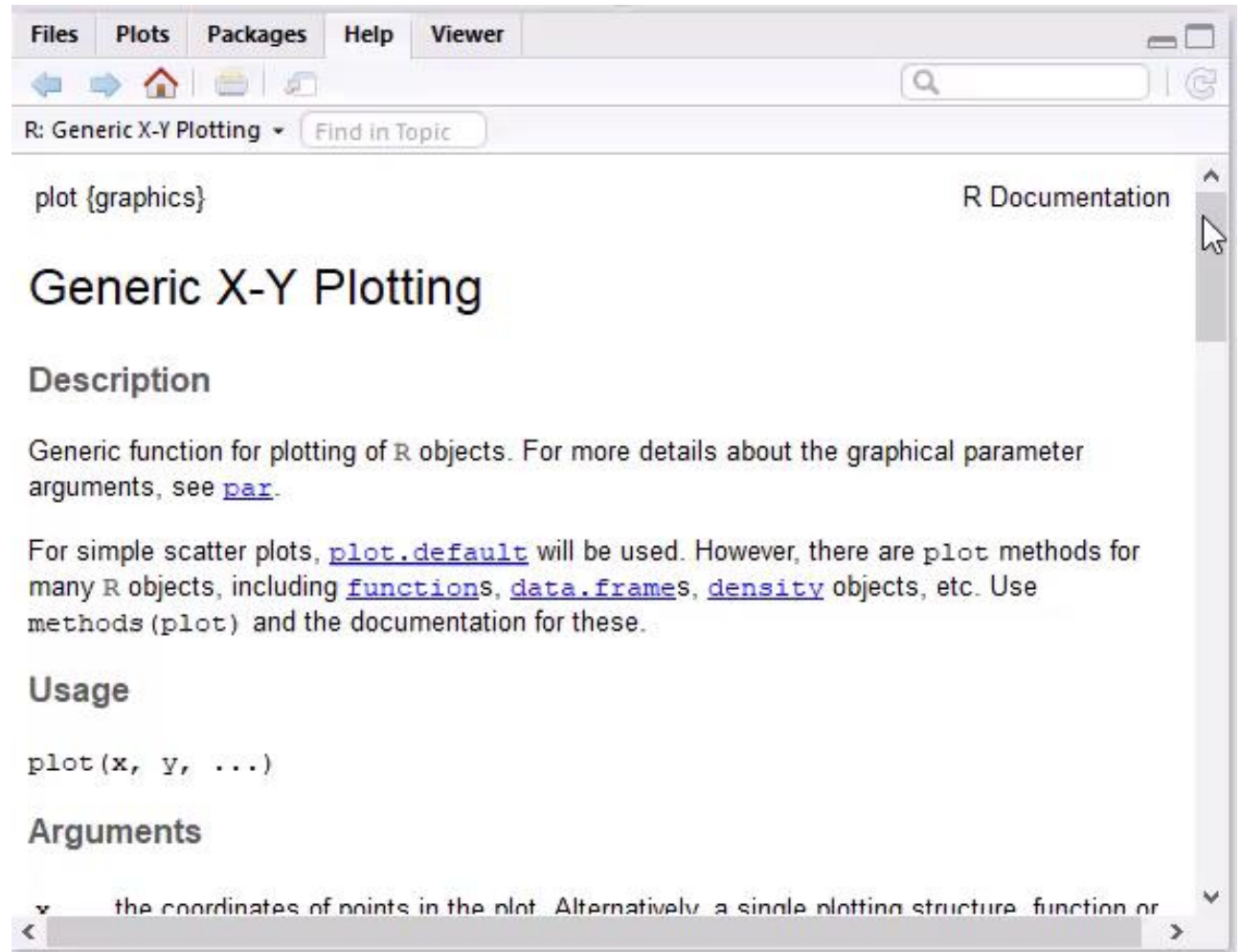
> `args(plot)`

- **CONSOLE OUTPUT:**

`function (x, y, ...)`

If you would like more information:

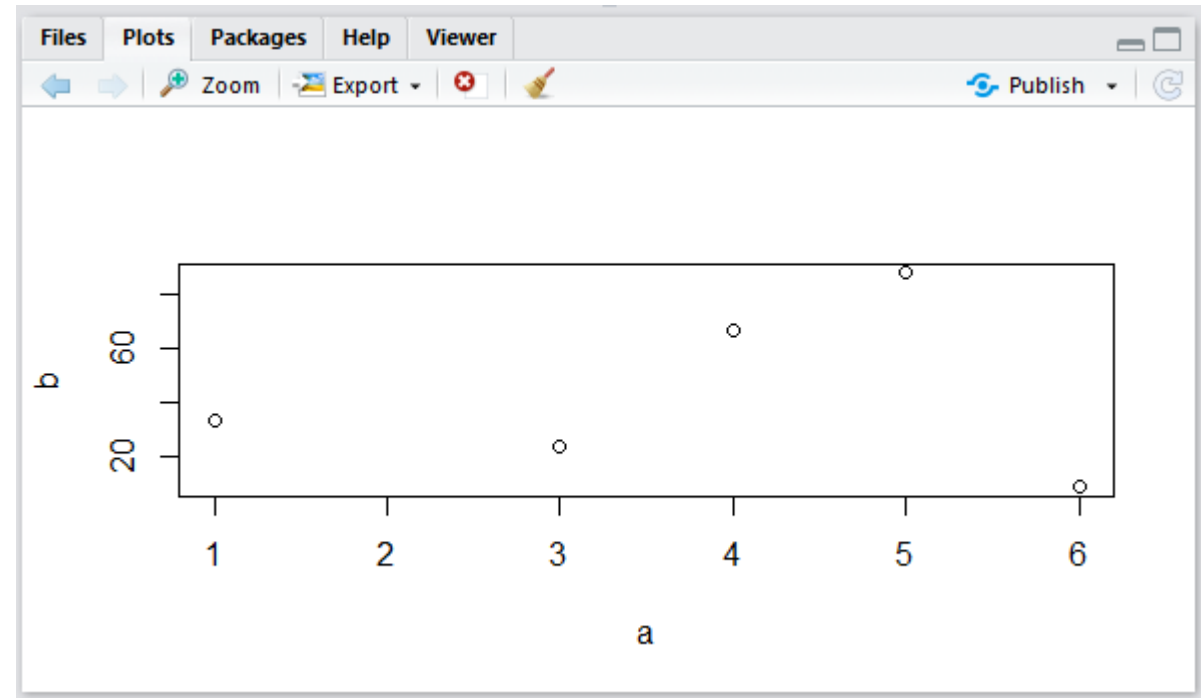
> `help(plot)`



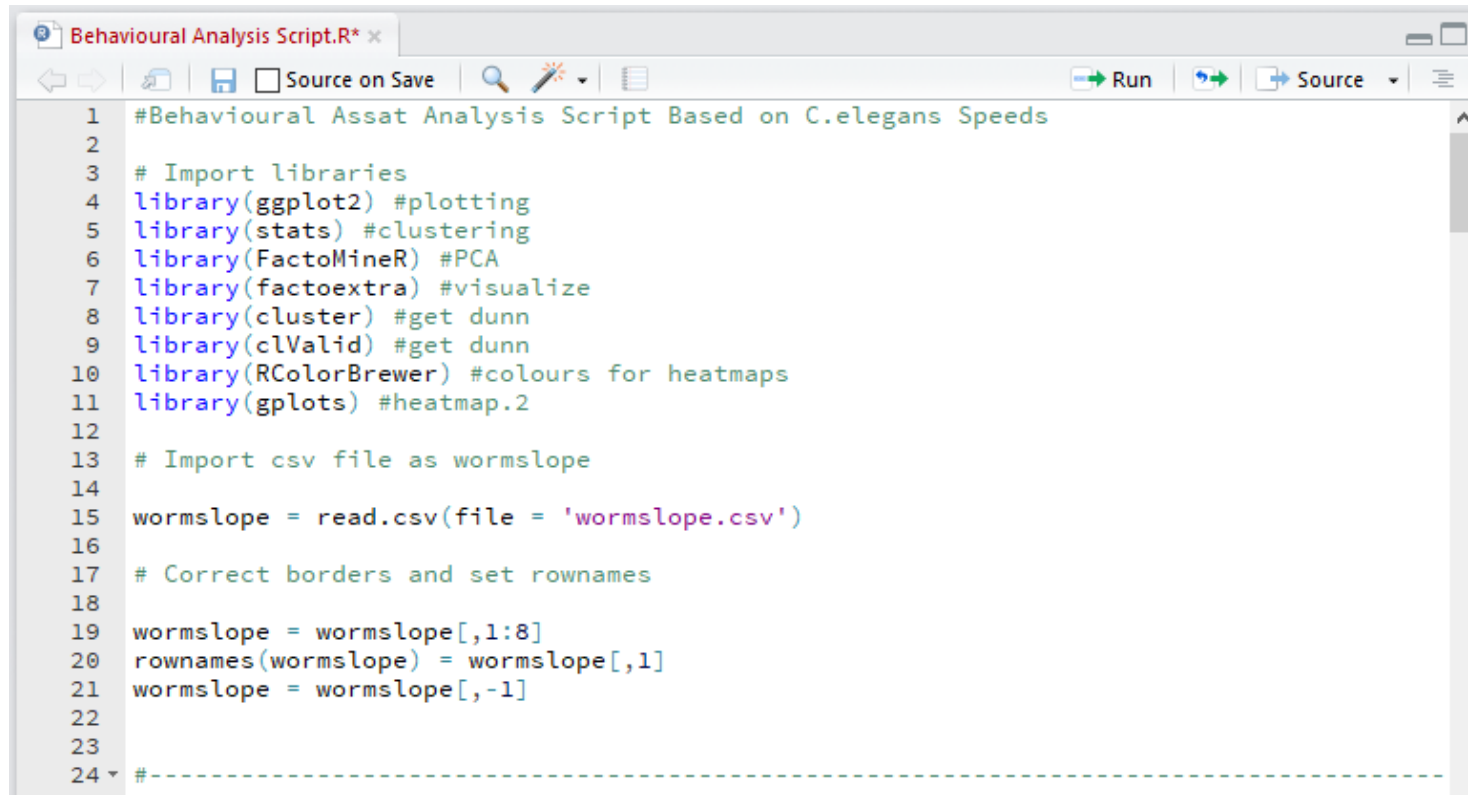
The screenshot shows the R Documentation window for the `plot` function. The window title is "R: Generic X-Y Plotting" and the page title is "plot {graphics} R Documentation". The main heading is "Generic X-Y Plotting". Below the heading is the "Description" section, which states: "Generic function for plotting of R objects. For more details about the graphical parameter arguments, see [par](#)." The "Usage" section shows the function signature: `plot(x, y, ...)`. The "Arguments" section is partially visible, starting with "x" and describing it as "the coordinates of points in the plot. Alternatively, a single plotting structure, function or".

COMMAND LINES & SCRIPTS

```
Console Terminal x
~/
>
>
> a = c(1,3,4,5,6)
> b = c(34,24,67,88,9)
> plot(a,b)
>
>
>
> 1+2
[1] 3
>
>
>
```

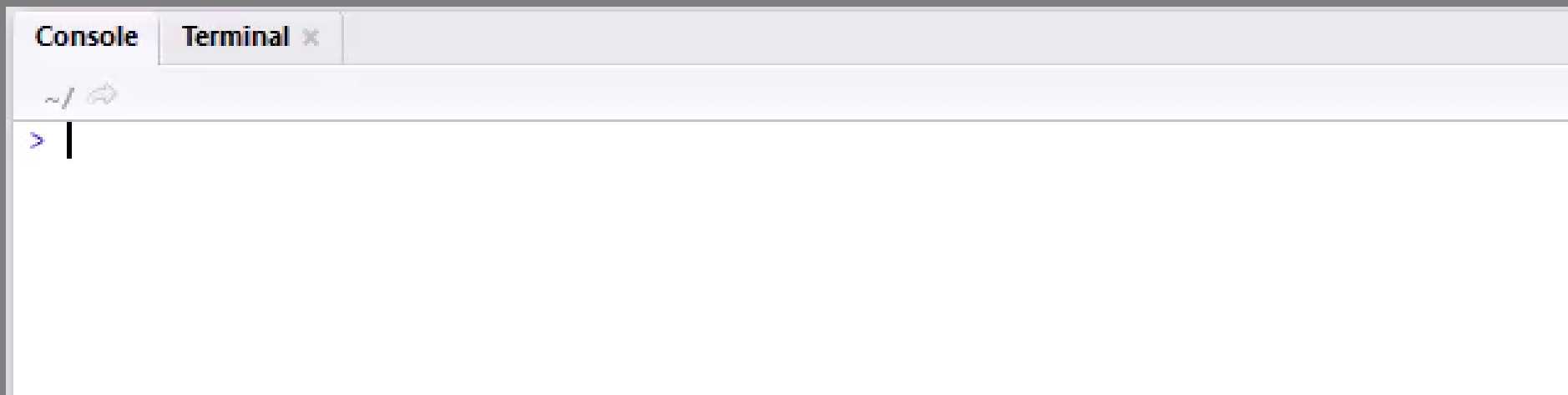


COMMAND LINES & SCRIPTS



```
Behavioural Analysis Script.R* x
Source on Save Run Source
1 #Behavioural Assat Analysis Script Based on C.elegans Speeds
2
3 # Import libraries
4 library(ggplot2) #plotting
5 library(stats) #clustering
6 library(FactoMineR) #PCA
7 library(factoextra) #visualize
8 library(cluster) #get dunn
9 library(clValid) #get dunn
10 library(RColorBrewer) #colours for heatmaps
11 library(gplots) #heatmap.2
12
13 # Import csv file as wormslope
14
15 wormslope = read.csv(file = 'wormslope.csv')
16
17 # Correct borders and set rownames
18
19 wormslope = wormslope[,1:8]
20 rownames(wormslope) = wormslope[,1]
21 wormslope = wormslope[,-1]
22
23
24 #-----
```

SAVING & CLOSING YOUR SESSION



EXPLORING R

R OBJECTS



CREATING VARIABLES IN R

- 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!**

```
> my_number = 5
```

```
> my_number
```

```
[1] 5
```

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
```

VECTOR TYPES

- numeric:

```
> mynumeric = c(3,5,7)
```

- character:

```
> mycharacter = c("bob", "nancy", "jose")
```

- logical or Boolean:

```
> mylogical = c(TRUE, FALSE, TRUE)
```

CHANGING YOUR VECTOR TYPE

- General workflow:

```
> myvector = c(3,5,7)
```

```
> myvector_char = as.character(myvector)
```

```
> myvector
```

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

- 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.
- Can be done with other types e.g. matrices
- **Use wisely**

LISTS

- Like vectors with mixed data types (numeric, character, logical)

```
> mylist = list(3, "TP53", FALSE)
```

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

```
[[2]]  
[1] " TP53 "
```

```
[[3]]  
[1] FALSE
```

Try it!
**What happens when you
unlist mylist?**

- “unlist”-ing with **unlist()** a list tries to coerce the data to an **atomic vector** of all the same type (lowest common denominator, usually a character)

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 = 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**

# 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
```



# DATAFRAMES

## AKA DF

- Very popular data structures!
- Subset of matrices allowing mixed types (numeric, character, logical)

```
> mydataframe = as.data.frame(mymatrix)
```

- You can give columns names so you can index by them

```
> names(mydataframe) = c("column1name", "column2name")
```

# 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 turn a DF into a matrix for certain operations:

```
> mymatrix = as.matrix(mydataframe)
```

Note: This turns data into all the same type

Remember: the lowest common denominator is usually character!

# ADDING & 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)

# A SELECTION OF HANDY FUNCTIONS

- > `class(object)` #gives object class
- > `mode(object)` #gives object type
- > `length(vector)` #gives length
- > `dim(object)` #gives matrix/dataframedimensions
- > `nrow(object)` #gives number of rows
- > `ncol(object)` #gives number of columns
- > `str(object)` #gives object structure

# MORE HANDY FUNCTIONS!

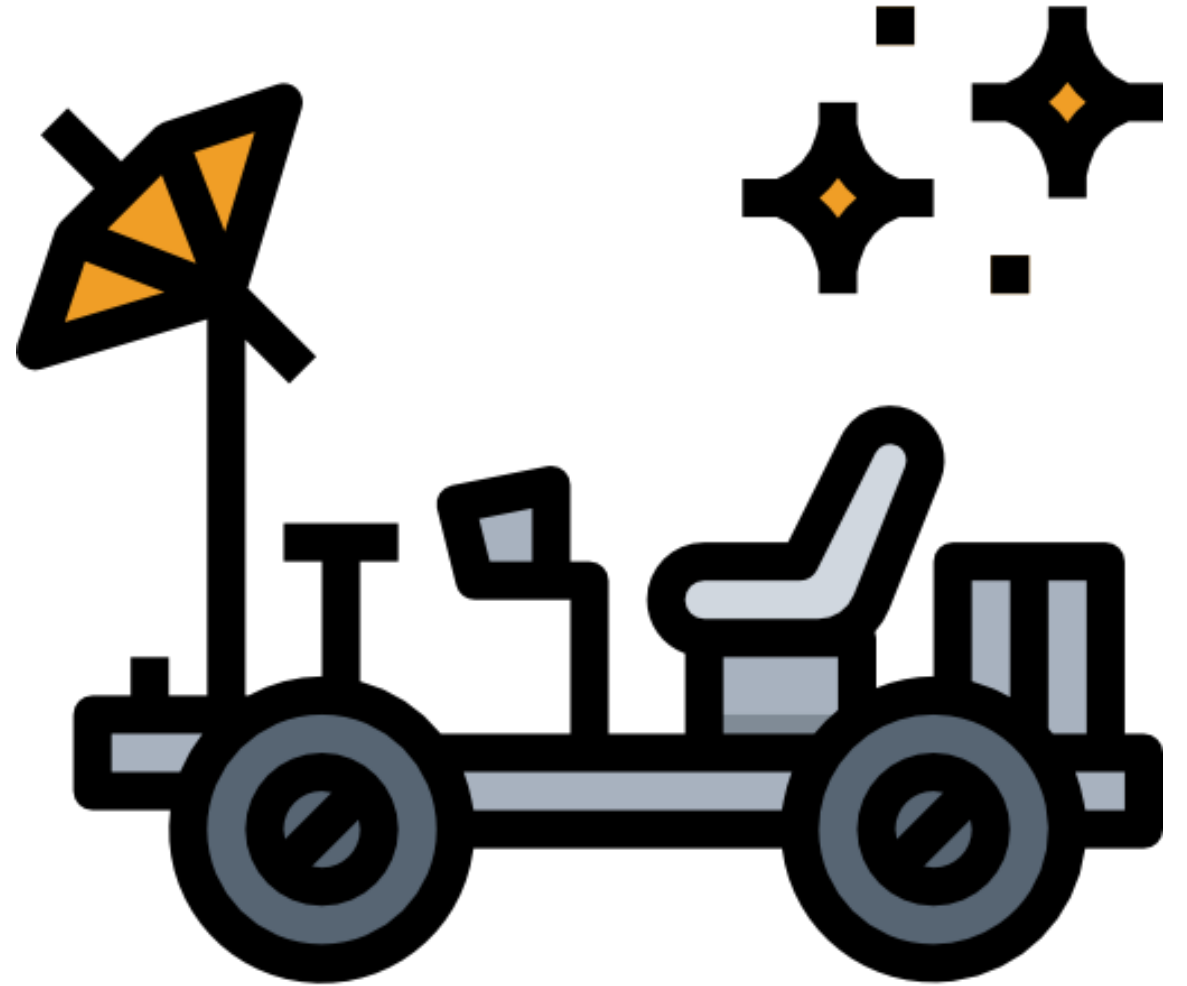
- > `head(object)`      #gives first 6 rows
- > `tail(object)`      #gives last 6 rows
- > `summary()`      #quick statistics

**Try it!**  
**How many rows did R  
return?**

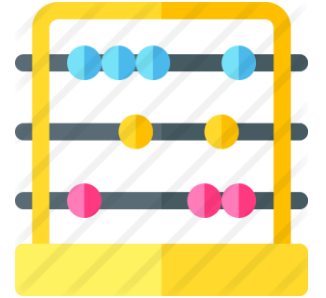
EXPLORING R

BUILT-IN MATH FUNCTIONS

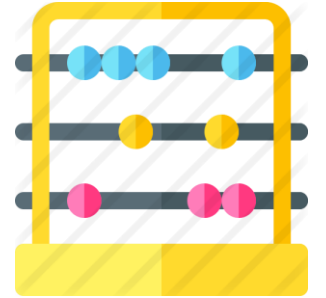
---



# R IS ESSENTIALLY A FANCY CALCULATOR AS IS ANY COMPUTER..



- > 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

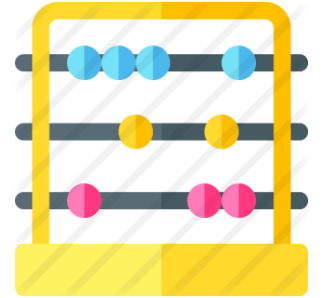


# BUT BETTER!

## R 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`

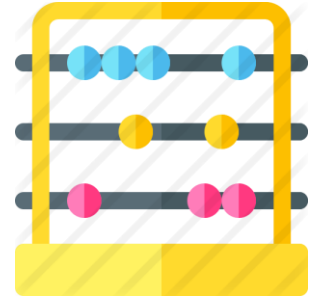




# BUT BETTER!

## MORE R 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
- > `abs(18 / -12)` #absolute value



# BUT BETTER!

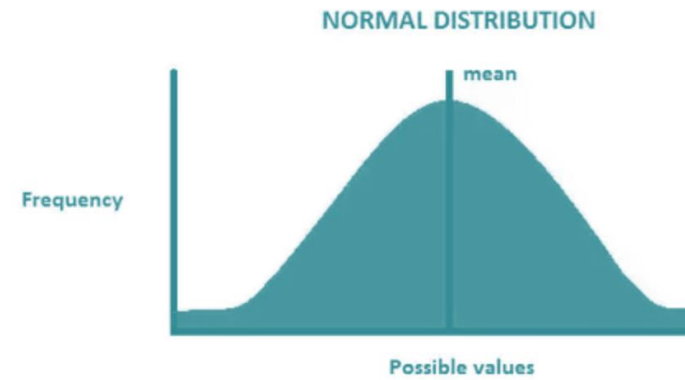
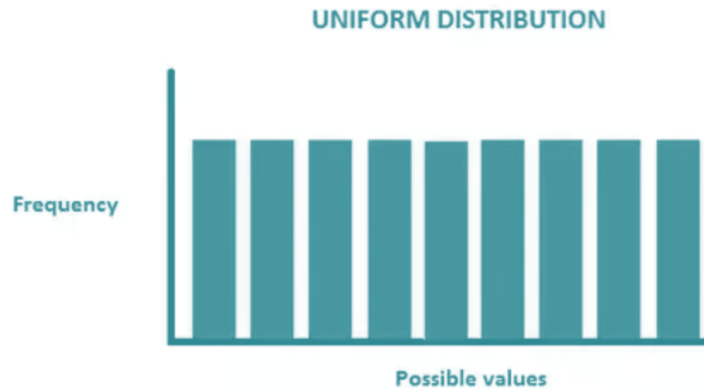
## MORE R BUILT-IN MATH FUNCTIONS!

> `runif(5)`

#number of random numbers between 0-1

> `rnorm(5)`

#random numbers from uniform normal distribution



# 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
```

# CONTROL STRUCTURES



# FOR LOOPS IN R

- Way to iterate over data

```
for (val in sequence){

statement

}
```

```
myvector <- c(2,5,3)
```

```
for (val in myvector) {
 print(val)
}
```

```
[1] 2
```

```
[1] 5
```

```
[1] 3
```

# WRITING FUNCTIONS IN R

- That's how you can pack up multiple commands into a structure you can use again and again!

## Pro Tip:

Name your functions wisely!  
Brains are unreliable machines..

```
sum.of.squares = function(x,y) {
 x^2 + y^2
}
```

```
> num_1 = 3
```

```
> num_2 = 2
```

```
> sum.of.squares(num_1, num_2)
```

```
[1] 13
```

# HANDY TRICKS

## THE APPLY FUNCTION FAMILY

- Returns an object as a result of **applying a function** to an entire data frame, matrix or list
- The **apply** functions are marginally faster than a regular for **loop**



# HANDY TRICKS

## THE APPLY FUNCTION FAMILY

```
apply (to_what, how, function)
```

About how: “1” is to apply over rows, “2” is to apply over columns

```
> mymatrix = matrix(c(1:6), nrow=3, ncol=2)
```

```
> apply(mymatrix, 1, sum)
```

```
[1] 5 7 9
```

**Your Turn:  
Try it with columns!**

```
> mymatrix
 [,1] [,2]
[1,] 1 4
[2,] 2 5
[3,] 3 6
```

# HANDY TRICKS

## VARIATIONS OF APPLY

| Function | Arguments             | Objective                                         | Input                      | Output              |
|----------|-----------------------|---------------------------------------------------|----------------------------|---------------------|
| apply    | apply(x, MARGIN, FUN) | Apply a function to the rows or columns or both   | Data frame or matrix       | vector, list, array |
| lapply   | lapply(X, FUN)        | Apply a function to all the elements of the input | List, vector or data frame | list                |
| sapply   | sapply(X FUN)         | Apply a function to all the elements of the input | List, vector or data frame | vector or matrix    |

# ONE MORE FOR THE ROAD!

## REPLICATE()

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

```
> replicate(5, rnorm(3))
```

```
 [,1] [,2] [,3] [,4] [,5]
[1,] 0.9559560 -0.1175259 -0.7622642 -1.0084890 -1.5176103
[2,] -0.7266965 -2.4495685 -0.6873605 -0.1995848 -1.3064050
[3,] 0.4646987 -1.1877134 -0.9814098 -0.6633240 0.2236935
```

```
> my_reps = replicate(5, rnorm(3))
```

**Your Turn:**  
**Sample the normal**  
**distribution 3 times then**  
**sum all of your**  
**outcomes together!**

# HANDY PACKAGES

## FOR DATA CLEANING AND MANIPULATION



# Living the R Life: An Example



# CLASS EXAMPLE

## OUR DATASET

If these formats don't work for you, try:  
> setwd("C:\\\\Users\\mkf8\\Downloads")

- 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 EXAMPLE

## QUICK STATS

- You can get some quick descriptive stats with `summary()`

> `summary(geneset)`

| TNBC1          | TNBC2          | TNBC3          | Normal1        | Normal2        |
|----------------|----------------|----------------|----------------|----------------|
| Min. : 0       | Min. : 65      | Min. : 31      | Min. : 22      | Min. : 208     |
| 1st Qu.: 7888  | 1st Qu.: 9538  | 1st Qu.: 9324  | 1st Qu.: 5074  | 1st Qu.: 7124  |
| Median : 13034 | Median : 16568 | Median : 19108 | Median : 10869 | Median : 14005 |
| Mean : 18596   | Mean : 26036   | Mean : 25646   | Mean : 14746   | Mean : 19425   |
| 3rd Qu.: 23850 | 3rd Qu.: 28194 | 3rd Qu.: 30389 | 3rd Qu.: 18866 | 3rd Qu.: 21576 |
| Max. : 103007  | Max. : 351603  | Max. : 272582  | Max. : 89837   | Max. : 212582  |

| Normal3        |
|----------------|
| Min. : 15      |
| 1st Qu.: 8944  |
| Median : 17710 |
| Mean : 25481   |
| 3rd Qu.: 32191 |
| Max. : 244692  |

### Pro Tip:

Starting with so plotting and descriptive statistics is the best way to go!

Do not dive into inferential analysis without doing some exploratory work first.



# CLASS EXAMPLE

## IMPORTING & VIEWING DATA

- Import your new dataset with headers and row names.

```
> geneset = read.csv('dataset_1.csv', header = T, row.names = 1)
```

- Can you remember which function allows us to take a peak at the first rows?

```
> head(geneset)
```

TRIPLE NEGATIVE  
BREAST CANCER

NORMAL  
SAMPLES

GENES OF  
INTEREST

|                 | 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 EXAMPLE

## TRANSPOSING DATA

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

```
> geneset_mat = as.matrix(geneset)
> geneset_mat_t = t(geneset_mat)
> head(geneset_mat_t)
```

# 't' is for 'transpose'

|         | ENSG00000008988 | ENSG00000009307 | ENSG00000019582 | ENSG00000026025 | 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           |                 |

- `as.data.frame()` will turn you data into a dataframe again!

**Your Turn:**  
**Try getting some quick stats on your newly transposed dataset!**

**What happens?**

LET'S TRY SOME PLOTS!

**YOU SEEM TRANSPARENT**

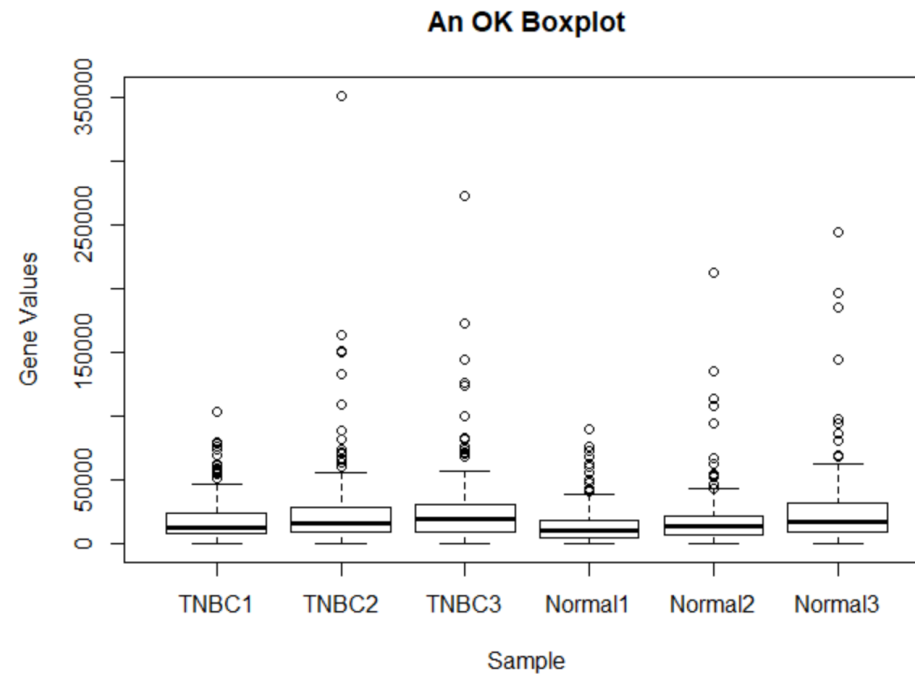


**BUT I CAN SEE YOU ARE PLOTTING  
SOMETHING IN THE BACKGROUND**

# CLASS EXAMPLE

## BOXPLOT

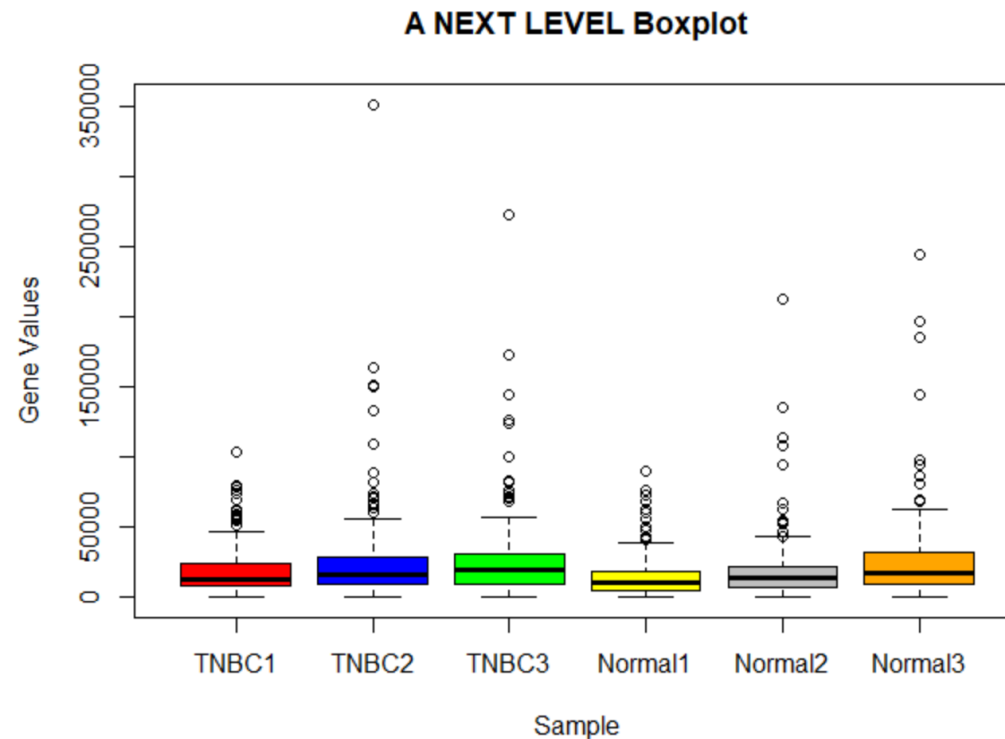
```
> boxplot(geneset, xlab = 'Sample', ylab = 'Gene Values', main = 'An OK Boxplot')
```



# CLASS EXAMPLE

## BOXPLOT

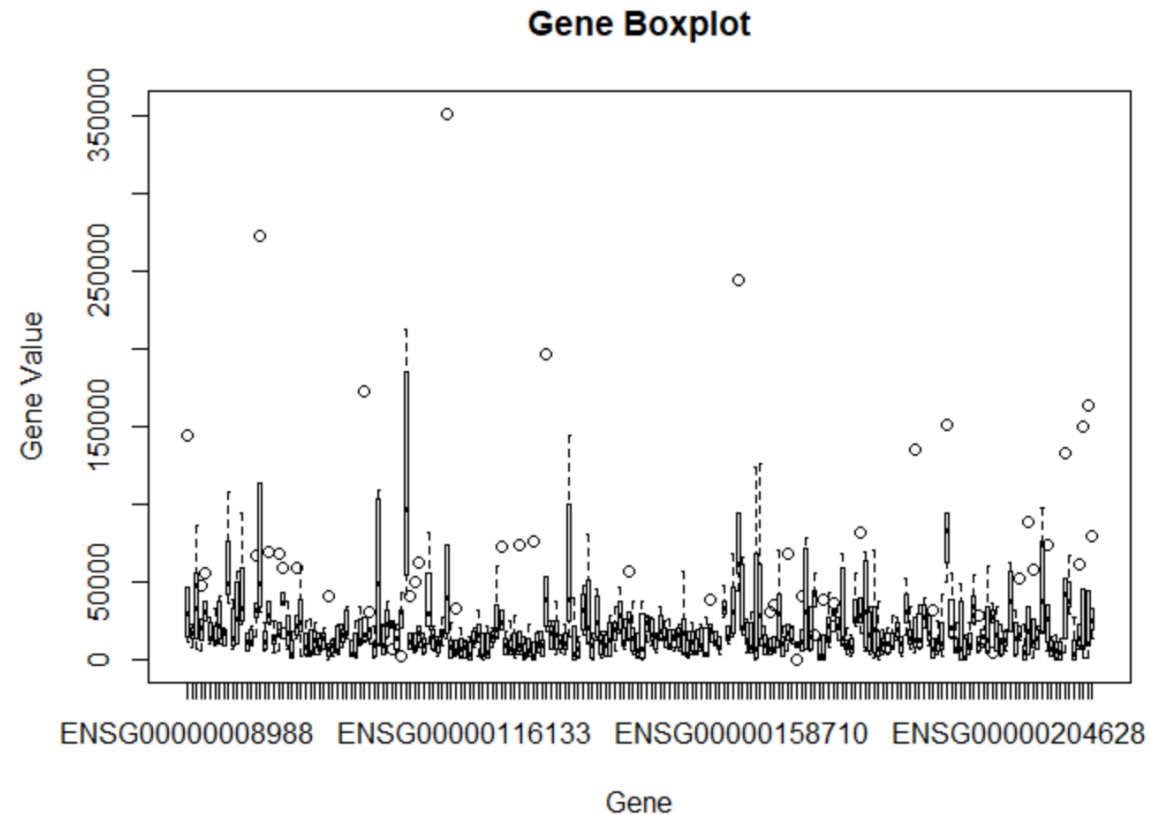
```
> boxplot(geneset, xlab = 'Sample', ylab = 'Gene Values', main = 'A NEXT LEVEL Boxplot',
col = c('red', 'blue', 'green', 'yellow', 'grey', 'orange'))
```



# CLASS EXAMPLE

## GENE BOXPLOT

```
> boxplot(geneset_mat_t, xlab = 'Gene', ylab = 'Gene Value', main = 'Gene Boxplot')
```



# CLASS EXAMPLE

## HANDY PLOT OPTIONS

There are many many  
more!

- `main = "Title"` # main 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
- `cex = N` # size of text and symbols
- `pch = N` # plot point symbol type

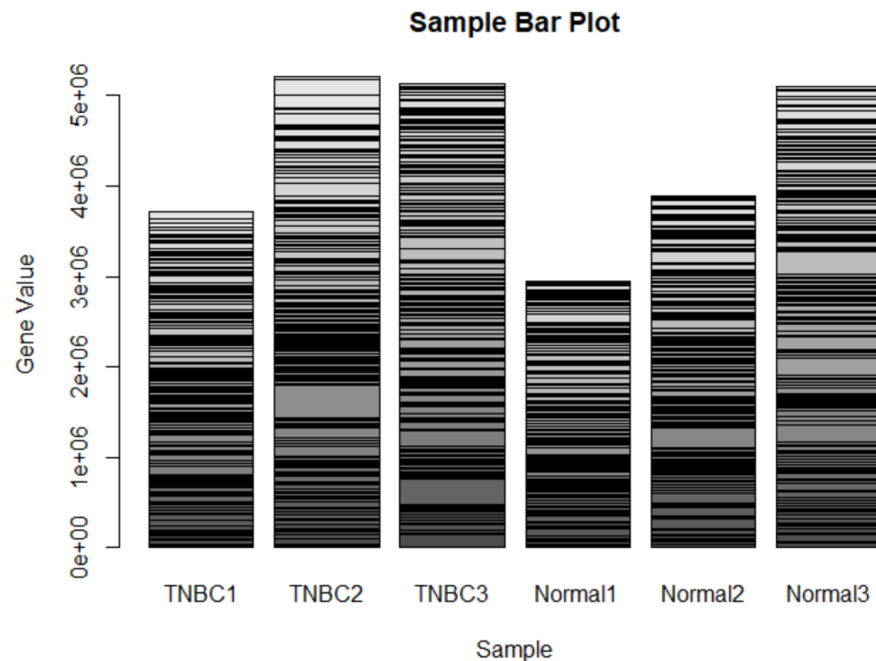
# CLASS EXAMPLE

## BARPLOTS

**Your Turn:  
Try to turn the plot blue!**

- For `barplot()` you will need **a matrix**

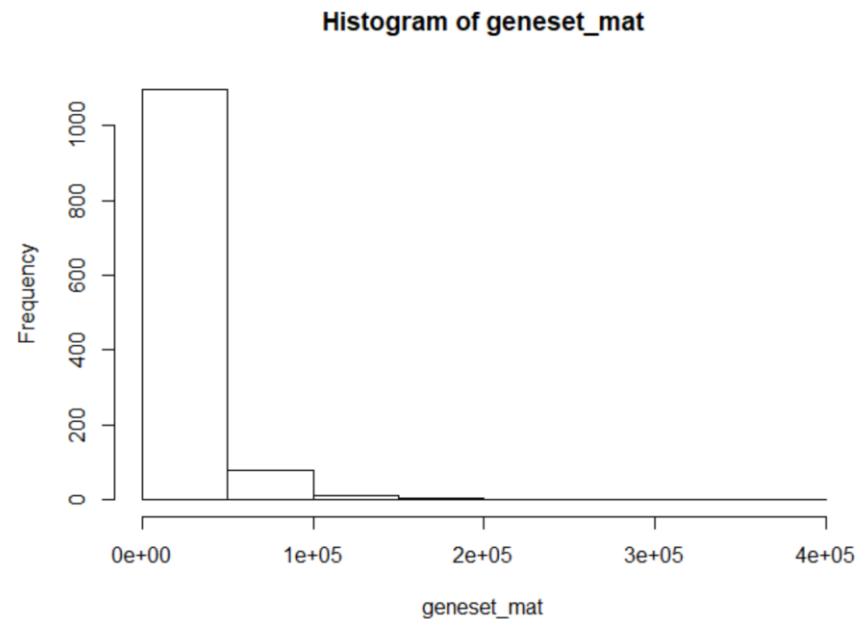
> `barplot(geneset_mat, xlab = 'Sample', ylab = 'Gene Value', main = 'Sample Bar Plot')`



# CLASS EXAMPLE

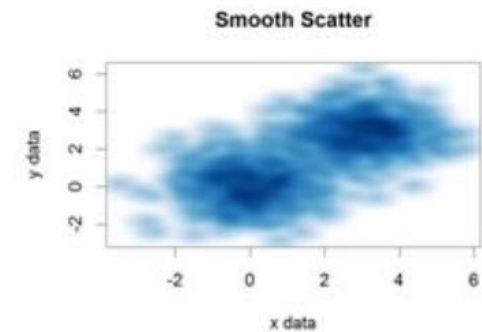
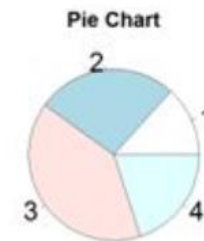
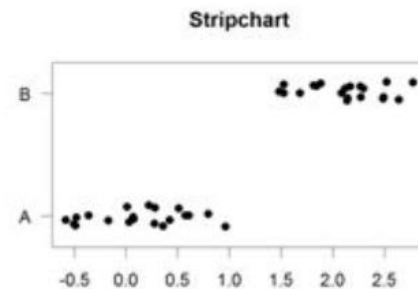
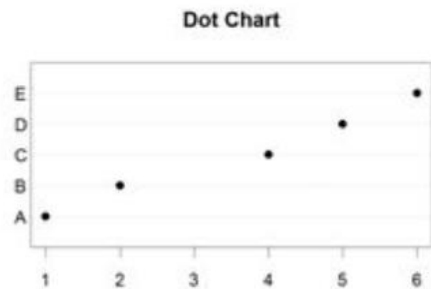
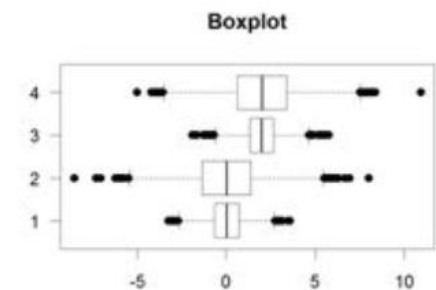
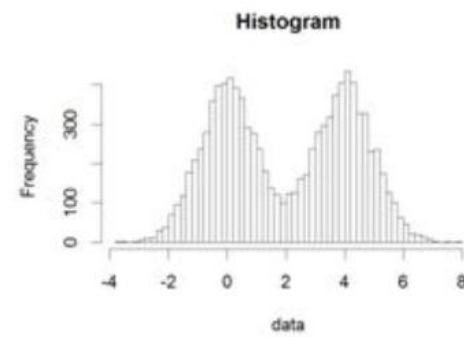
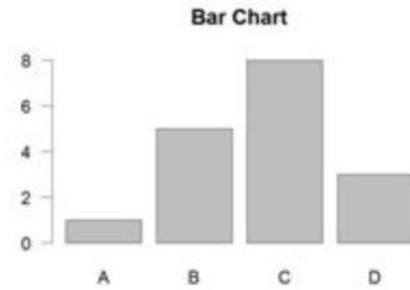
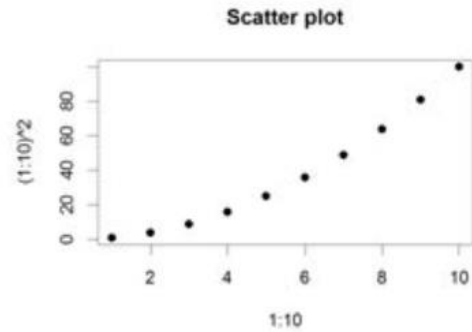
## HISTOGRAMS

- Plot a histogram of the frequency of values in our dataset  
> `hist(geneset_mat)`



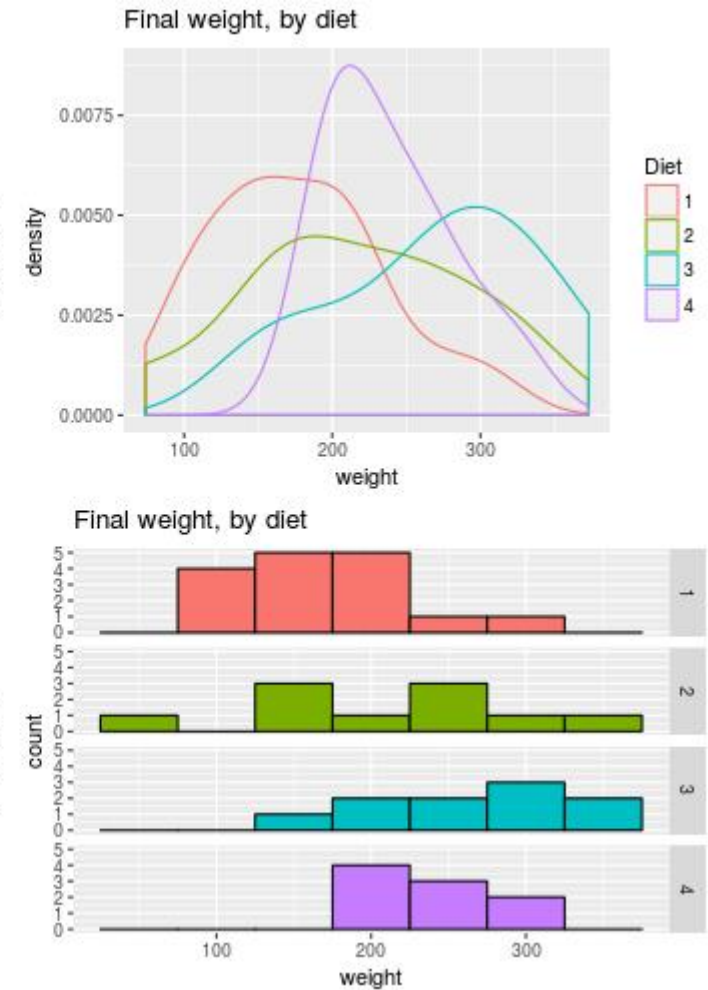
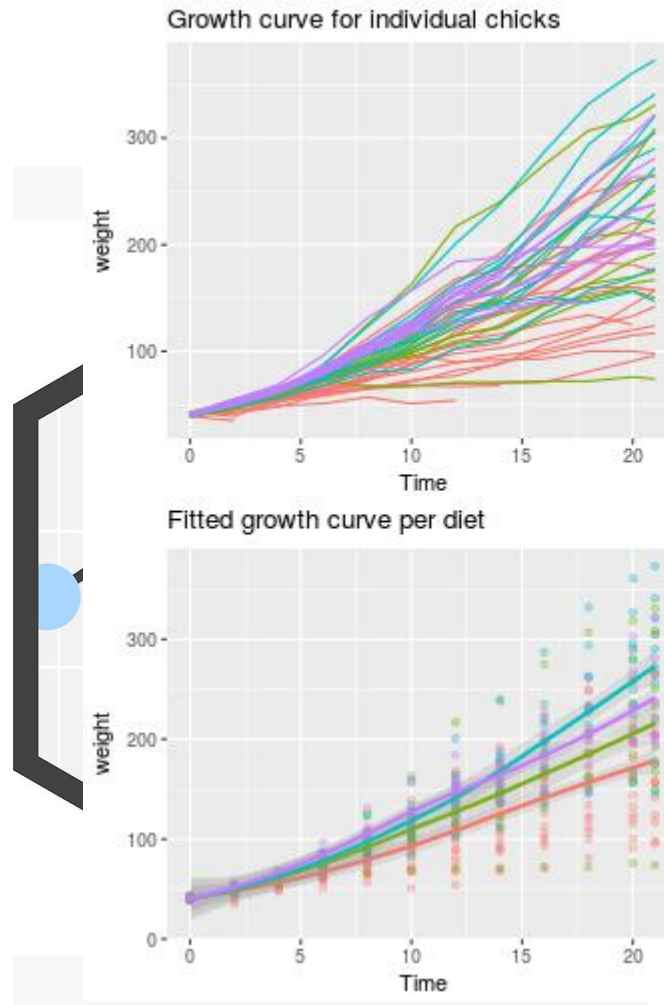


# OTHER PLOT TYPES AVAILABLE IN R



# POPULAR PLOTTING PACKAGE

## GGPLOT 2



# BIRD BONES

## CLASS ACTIVITY

- Have a look at the bird dataset.

### Content

There are 420 birds contained in this dataset. Each bird is represented by:

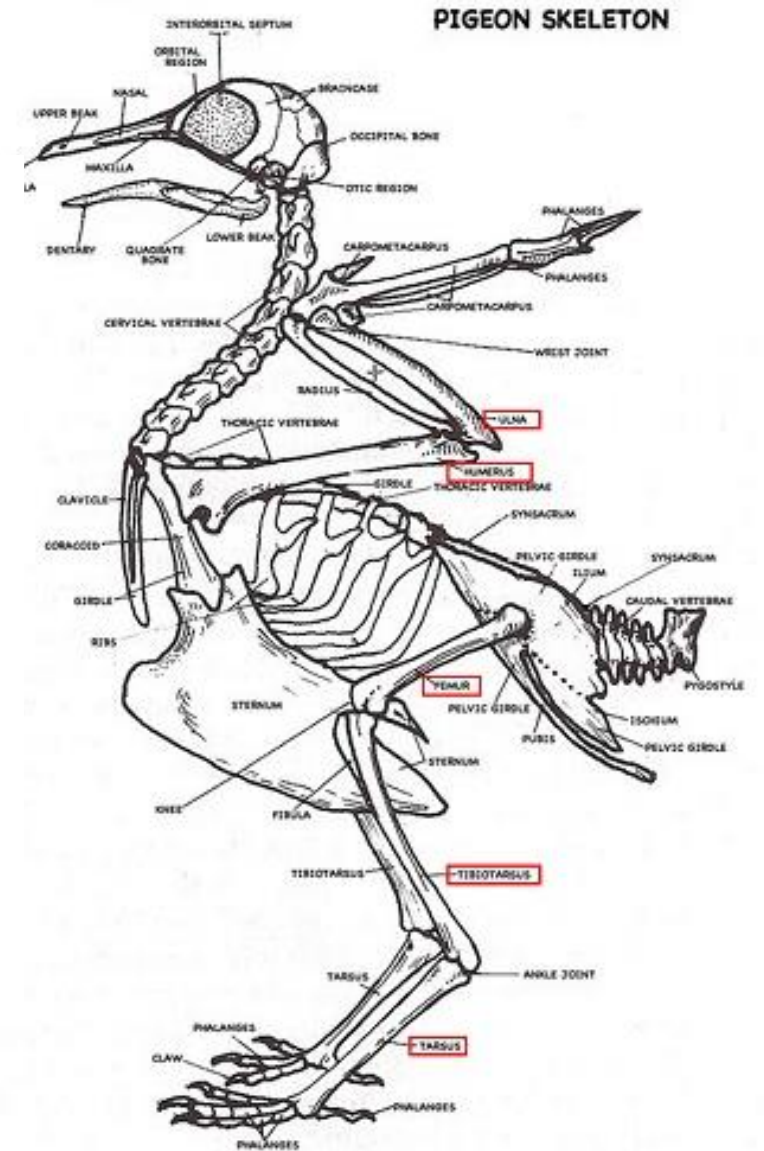
- Length and Diameter of Humerus
- Length and Diameter of Ulna
- Length and Diameter of Femur
- Length and Diameter of Tibiotarsus
- Length and Diameter of Tarsometatarsus

Each bird has a label for its ecological group:

- *SW*: Swimming Birds
- *W*: Wading Birds
- *T*: Terrestrial Birds
- *R*: Raptors
- *P*: Scansorial Birds
- *SO*: Singing Birds

### Columns

- # id Sequential id
- # huml Length of Humerus (mm)
- # humw Diameter of Humerus (mm)
- # ulnal Length of Ulna (mm)
- # ulnaw Diameter of Ulna (mm)
- # feml Length of Femur (mm)
- # femw Diameter of Femur (mm)
- # tibl Length of Tibiotarsus (mm)
- # tibw Diameter of Tibiotarsus (mm)
- # tarl Length of Tarsometatarsus (mm)
- # tarw Diameter of Tarsometatarsus (mm)
- A type Ecological Group



# BIRD BONES

## CLASS ACTIVITY

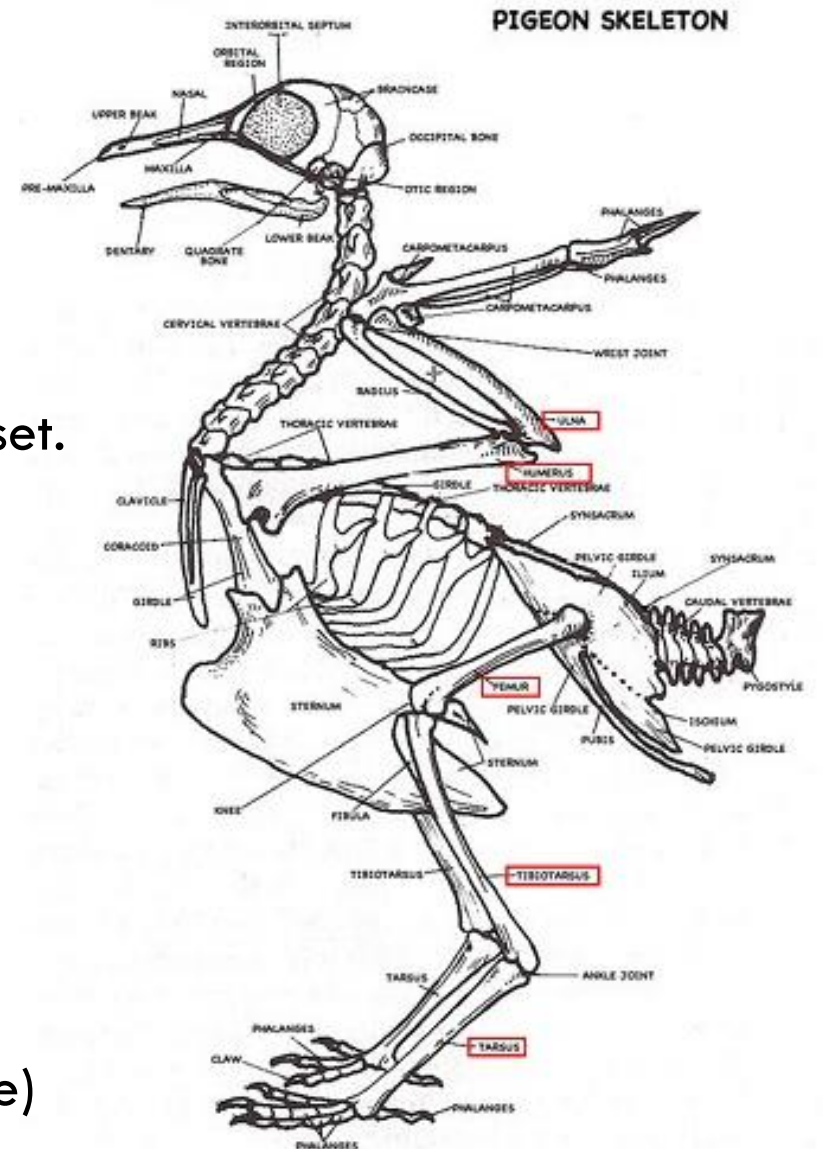
- Have a look at the bird dataset.
- Plot a histogram of huml 'Length of Humerus' from the bird dataset.
- What did you see?

**Hint:** You can use `$` to subset columns from data

What happens if you use `plot()` with 'huml' and 'humw'?

### Bonus Question

Can you make the points colors match their ecological group (type)



# ANALYSIS EXAMPLES

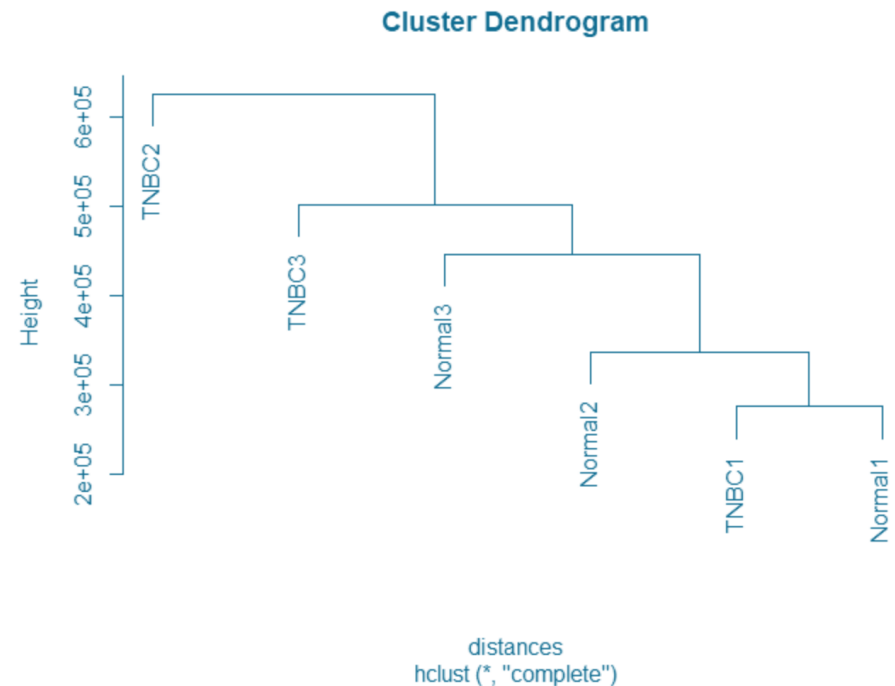


# CLASS EXAMPLE

## HCLUST

- To create an hierarchical clustering of your samples you will need to calculate the distance between every point in the matrix
- Use the transposed format!

```
> distances = dist(geneset_mat_t)
> clusters = hclust(distances)
> plot(clusters)
```



# CLASS EXAMPLE

## A SIMPLE T-TEST

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

```
welch Two Sample t-test
```

```
data: geneset[, 1:3] and geneset[, 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
```



shutterstock.com • 1290554923

```
> ?t.test
```

**Make sure you are using the correct options!**  
**One vs two sided. Paired vs independent samples**







**BIG DATA**

**SERIOUS BUSINESS**

memegenerator.net

# USING THE O2 CLUSTER

- Working in R studio Good for proofing code or for working with datasets you can store in your computer, but not a scalable solution for High Performance Computing (HPC)
- User Training has dedicated O2 sessions!

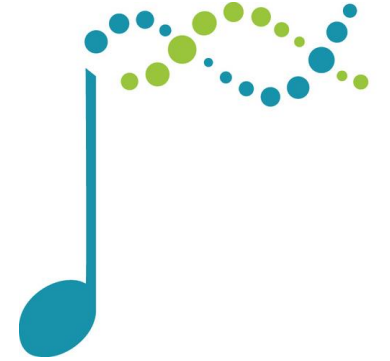


## **A short introduction**



# OUR OBJECTIVES FROM THIS INTRO

- Explain what is Bioconductor is
- Identify some handy packages
- Set up Bioconductor on your workspace
- Where to start?



# WHAT IS IT?

- Open-source, open-development software project for the analysis of genomic data
- High-quality documentation and **reproducible** research

| Statistical Analysis    | Comprehension      | High-throughput |
|-------------------------|--------------------|-----------------|
| Large data              | Biological context | Sequencing      |
| Technological artifacts | Visualization      | Microarrays     |
| Designed experiments    | Reproducibility    | Flow cytometry  |



Mariposa Symphony Orchestra

## About *Bioconductor*

Bioconductor provides tools for the analysis and comprehension of high-throughput genomic data.

Bioconductor uses the R statistical programming language, and is open source and open development. It has two releases each year, and an active user community. Bioconductor is also available as an [AMI](#) (Amazon Machine Image) and a series of [Docker](#) images.

## News

- Bioconductor [3.9](#) is available.
- Core team **job opportunities** for scientific programmer / analyst and senior programmer / analyst! contact Martin.Morgan at RoswellPark.org
- Bioconductor [F1000 Research Channel](#) available.
- Orchestrating high-throughput genomic analysis with *Bioconductor* ([abstract](#)) and other [recent literature](#).

### Install »

- Discover [1741 software packages](#) available in *Bioconductor* release 3.9.

Get started with *Bioconductor*

- [Install Bioconductor](#)
- [Get support](#)
- [Latest newsletter](#)
- [Follow us on twitter](#)
- [Install R](#)

### Learn »

Master *Bioconductor* tools

- [Courses](#)
- [Support site](#)
- [Package vignettes](#)
- [Literature citations](#)
- [Common work flows](#)
- [FAQ](#)
- [Community resources](#)
- [Videos](#)

### Use »

Create bioinformatic solutions with *Bioconductor*

- [Software](#), [Annotation](#), and [Experiment](#) packages
- [Amazon Machine Image](#)
- [Latest release announcement](#)
- [Community Slack](#) sign-up
- [Support site](#)

### Develop »

Contribute to *Bioconductor*

- [Developer resources](#)
- [Use Bioc 'devel'](#)
- 'Devel' [packages](#)
- [Package guidelines](#)
- [New package submission](#)
- [Git source control](#)
- [Build reports](#)

# USEFUL LINKS

## # Package Inventory

<https://bioconductor.org/packages>

## # Support Forum

<https://support.bioconductor.org>

Method | Open Access

## Bioconductor: open software development for computational biology and bioinformatics

Robert C Gentleman ✉, Vincent J Carey, Douglas M Bates, Ben Bolstad, Marcel Dettling, Sandrine Dudoit, Byron Ellis, Laurent Gautier, Yongchao Ge, Jeff Gentry, Kurt Hornik, Torsten Hothorn, Wolfgang Huber, Stefano Iacus, Rafael Irizarry, Friedrich Leisch, Cheng Li, Martin Maechler, Anthony J Rossini, Gunther Sawitzki, Colin Smith, Gordon Smyth, Luke Tierney, Jean YH Yang and Jianhua Zhang

*Genome Biology* 2004 5:R80

<https://doi.org/10.1186/gb-2004-5-10-r80> | © Gentleman et al.; licensee BioMed Central Ltd. 2004

Received: 19 April 2004 | Accepted: 3 August 2004 | Published: 15 September 2004

nature | methods

Perspective | Published: 29 January 2015

## Orchestrating high-throughput genomic analysis with Bioconductor

Wolfgang Huber ✉, Vincent J Carey [...] Martin Morgan

*Nature Methods* **12**, 115–121 (2015) | [Download Citation](#) ↓



Analysis  
(high-throughput,  
genomic)  
...



Annotation  
(gene models)  
...



Toy data  
to demo  
packages



Workflow

Packages that describe  
workflows involving  
multiple packages

# POPULAR R PACKAGES

- [GenomicRanges](#): 'Ranges' to describe data and annotation; `GRanges()`, `GRangesList()`
- [Biostrings](#): DNA and other sequences, `DNASTringSet()`
- [GenomicAlignments](#): Aligned reads; `GAlignments()` and friends
- [GenomicFeatures](#), [AnnotationDbi](#): annotation resources, `TxDb`, and `org` packages.
- [SummarizedExperiment](#): coordinating experimental data
- [rtracklayer](#): import Genome annotations e.g BED, WIG, GTF, etc.

# SETTING UP BIOCONDUCTOR

```
if (!requireNamespace("BiocManager", quietly = TRUE))
 install.packages("BiocManager")
BiocManager::install()
```

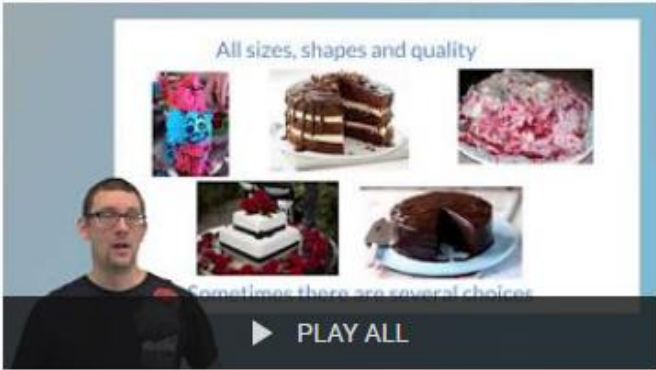
- To ensure you are getting the most up to date version of a given package use BiocManager

```
BiocManager::install(c("GenomicFeatures", "AnnotationDbi"))
```

Cool Bioconductor Feature: Packages come with vignettes. Instructions on how to use the package and workflow examples!

```
browseVignettes(package = "Biostrings")
```


# WHERE TO GET STARTED



Bioconductor for Genomic Data Science, all

37 videos • 23,159 views • Last updated on Sep 7, 2015

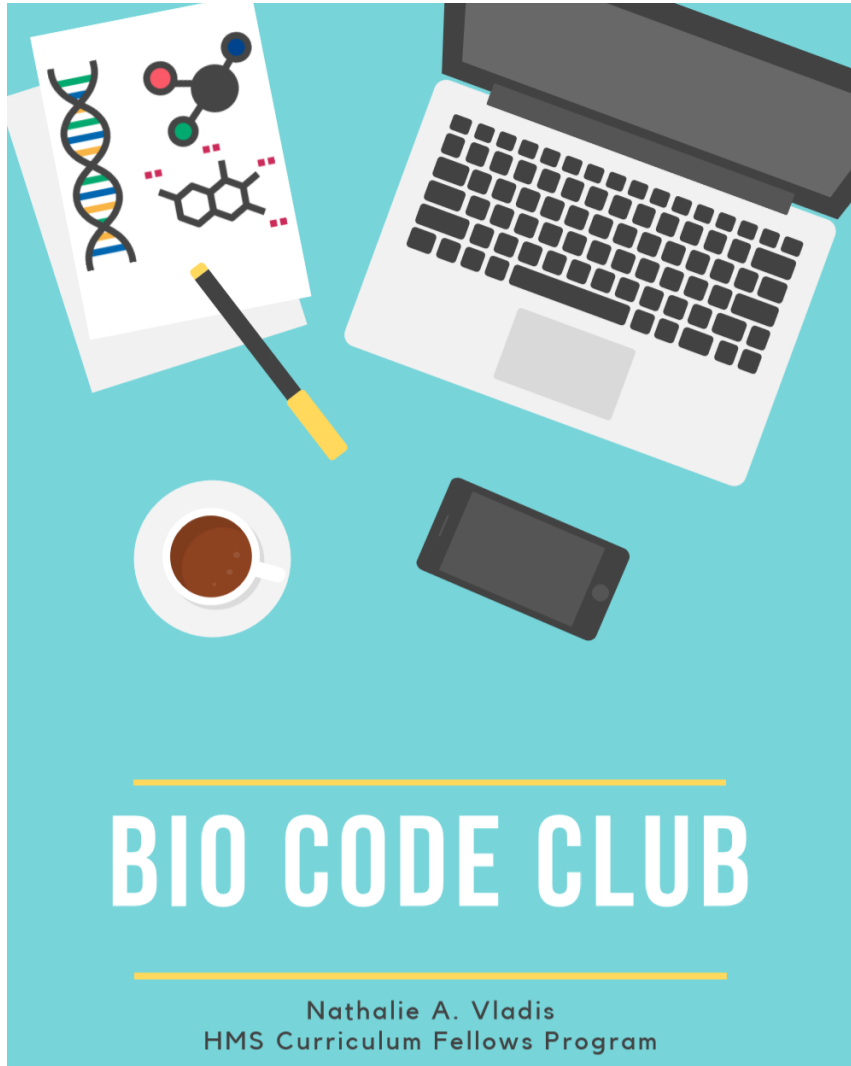
≡+ ↗ ...

 Kasper Hansen **SUBSCRIBE 564**

- Everything you need to get started, building from the ground up!
- Introduction to Key Packages
- Introduction to Popular Workflows

<https://www.youtube.com/playlist?list=PLA0uMgYDbgCKNH8Cm-68gEnw39fR5mhFa>





Keep an eye out for  
Bio Code Club!

Last Wednesday of  
each month during  
the Summer 4-5 pm  
TMEC 304

Once a week in  
TMEC starting  
September  
Time & Venue TBC

# Enjoy!

---

Need help or advice finding  
resources?

nathalie\_vladis@hms.harvard.edu

Please share your feedback about this session at:  
<https://forms.gle/Exw7Dnh2TqqYJePm6>

# APPENDIX

## Kate Holton's Materials

Here you will find info about:

- Setting up your connection to the 02 cluster Quick Start
- Importing files from other Software



R on

O<sub>2</sub>



# 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 strings 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
  - `col.names=NA` #Excel-like readability
  - `quote=FALSE` #turn off character string quoting

# 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 --x11 bash
```

(where 8G is memory requested)

- Graphics: SSH Keys!

srun: add `--x11`

sbatch: add `--x11=batch`

- Parallel/doParallel, BiocParallel, doMC libraries: run over multiple cores (-c up to 20 cores)
- Rmpi, SNOW, doMPI: 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 (“extra” recommended)  
`mfk8@login01:~$ module load gcc/6.2.0 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 (1 time, not in .bashrc)

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

```
mfk8@login01:~$ export R_LIBS_USER="~/R-version"
```

```
mfk8@login01:~$ echo 'R_LIBS_USER="~/R-version"'> $HOME/.Renvirom
```

- If you must manually download a package (not through Bioconductor/CRAN etc), put the package in the set up location (/home/mfk8/R-version)
- Accessing packages manually uploaded to your O2 R library (first time)  
> install.packages("name-of-your-package")