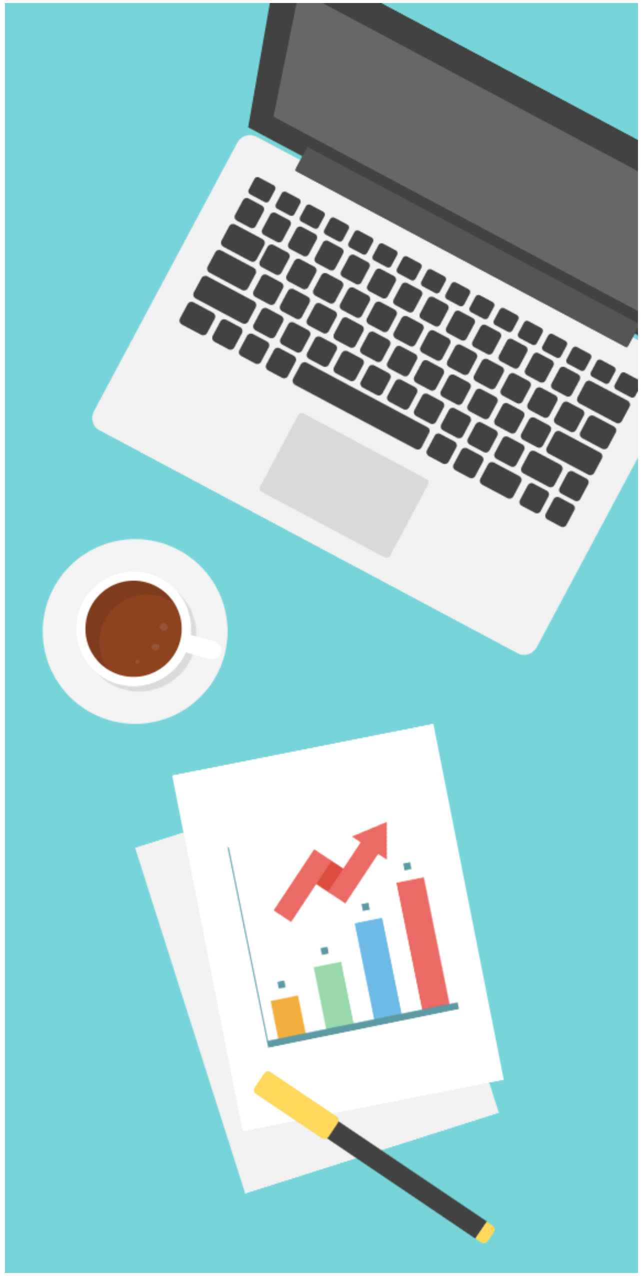


Intro to R Programming

& RStudio

Nathalie Vladis, PhD

HMS Curriculum Fellow in Quantitative Skills

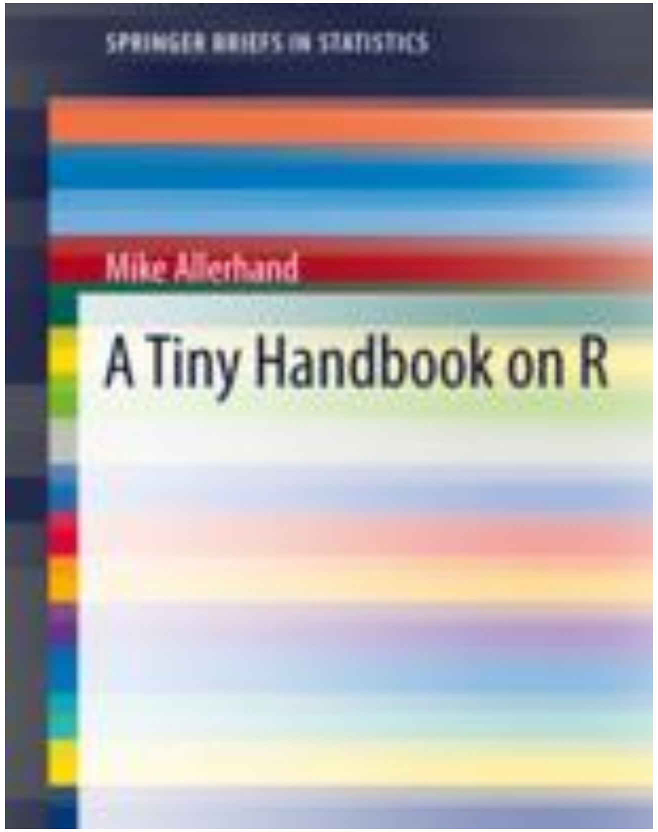
R INTRO OBJECTIVES

- 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



SPRINGER BRIEFS IN STATISTICS

Mike Allerhand

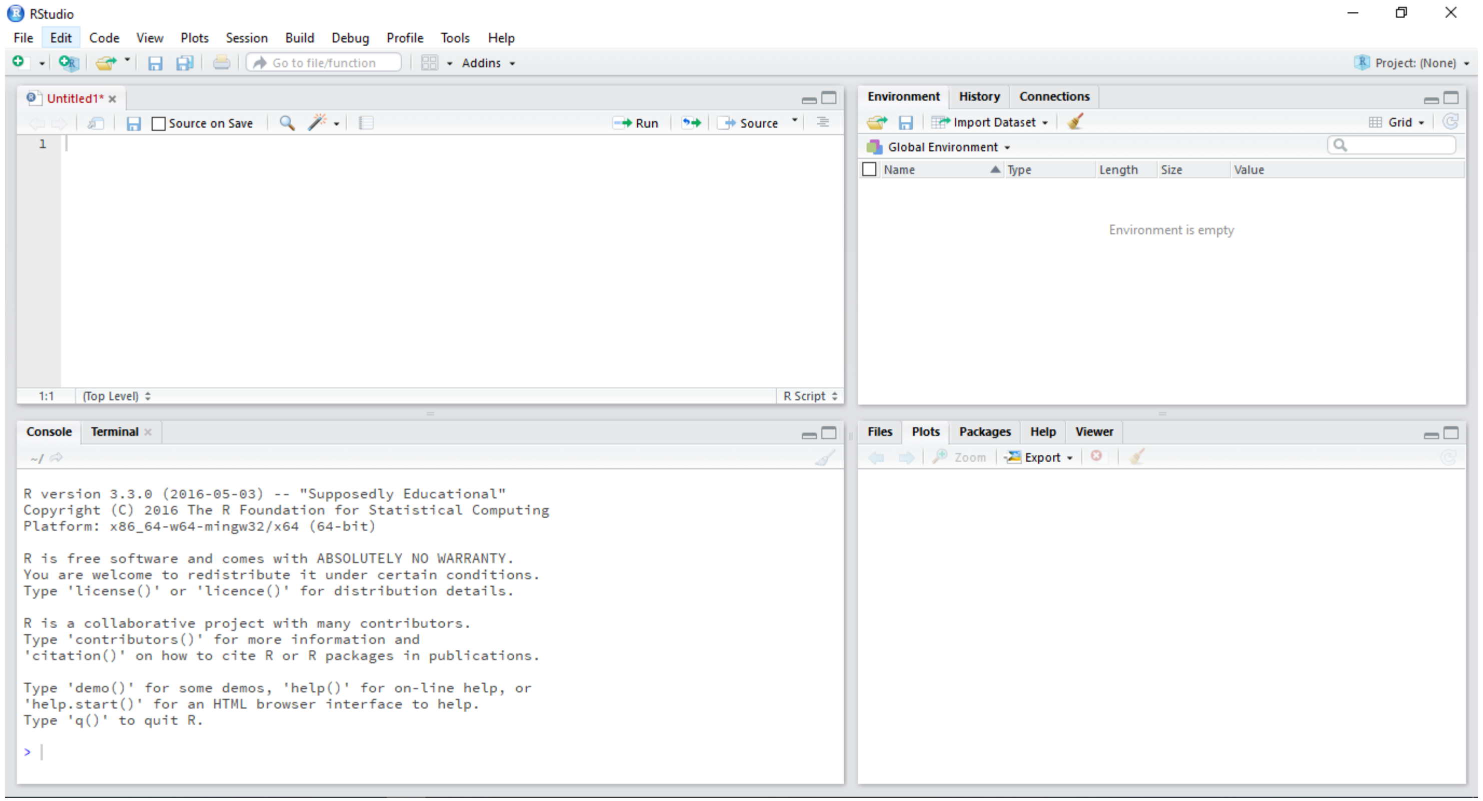
A Tiny Handbook on R

WELCOME TO R STUDIO

WELCOME TO RSTUDIO!
A QUICK TOUR

Syntax Editor

Workspace Management



```
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.

> |
```

Console

Plotting Area & Viewer

SOME BASIC SYNTAX

Try it in the RStudio!

- 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

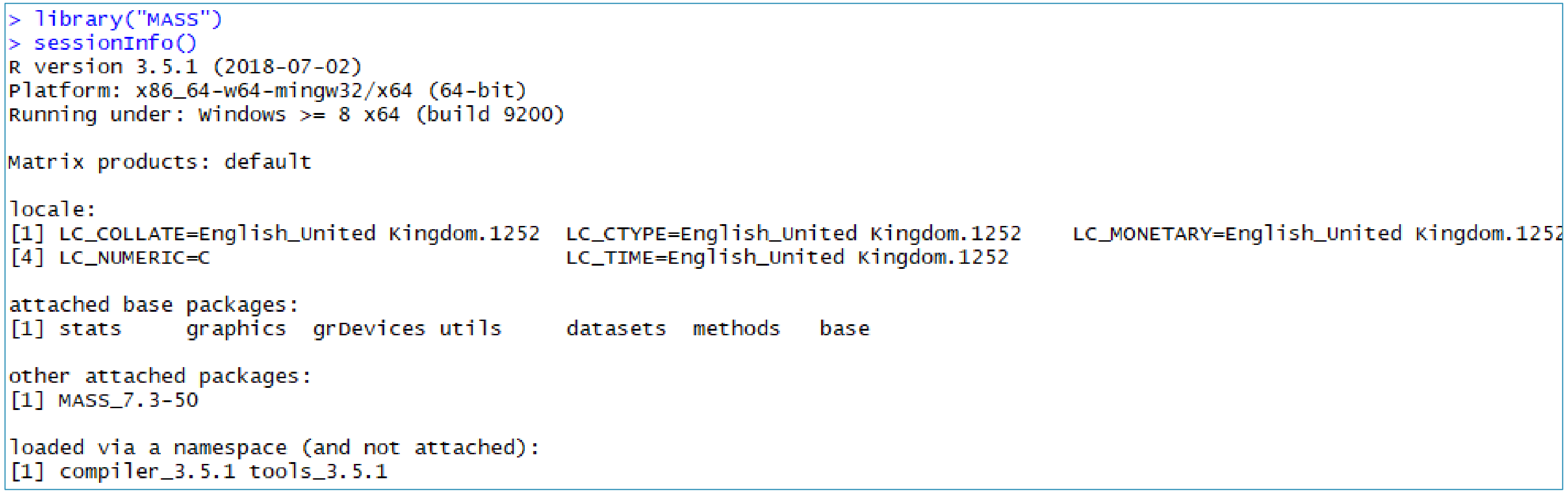
# Your **best friend** when you write long scripts! (Use Often!)

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

INSPECTING YOUR WORKSPACE

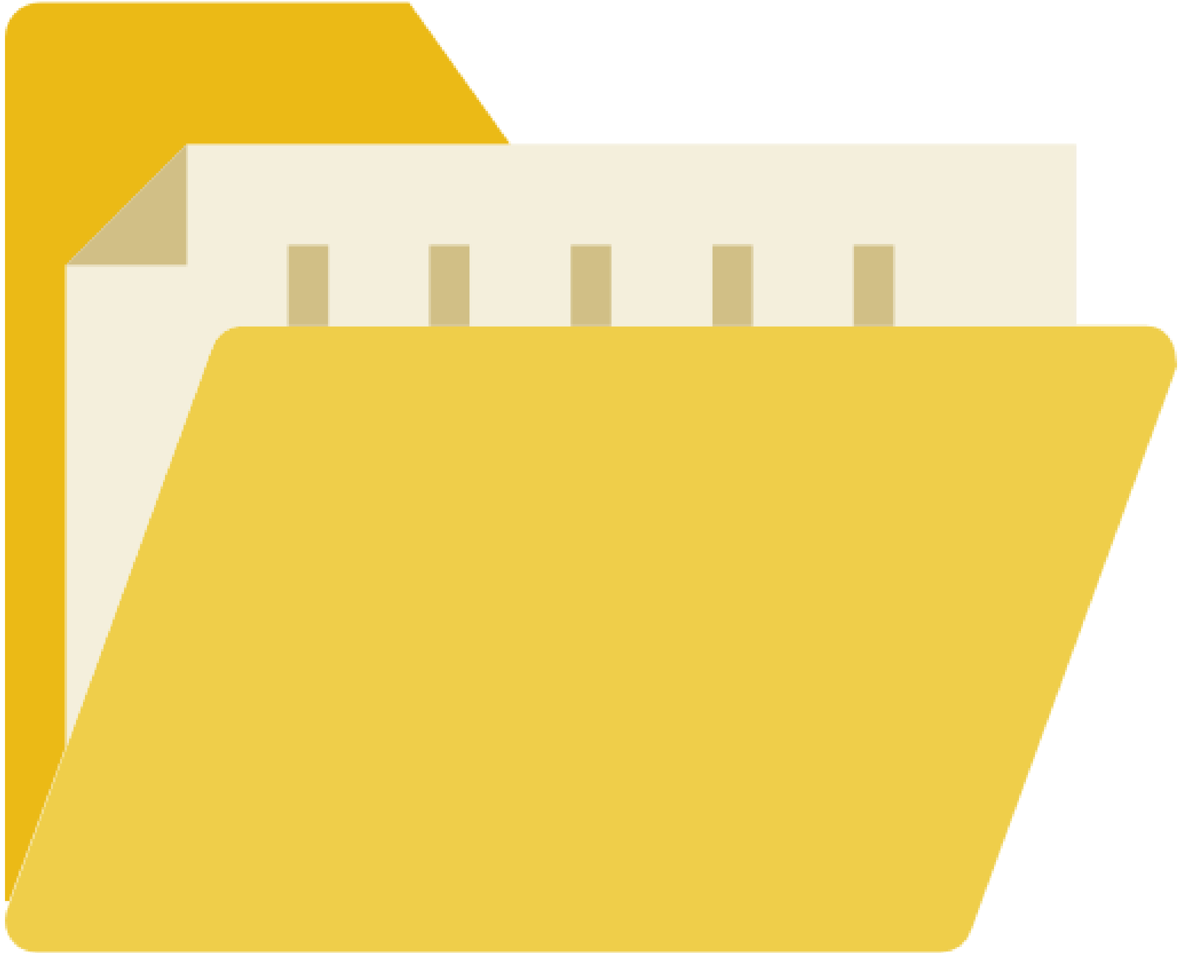
Try it in the RStudio!

```
> 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
```



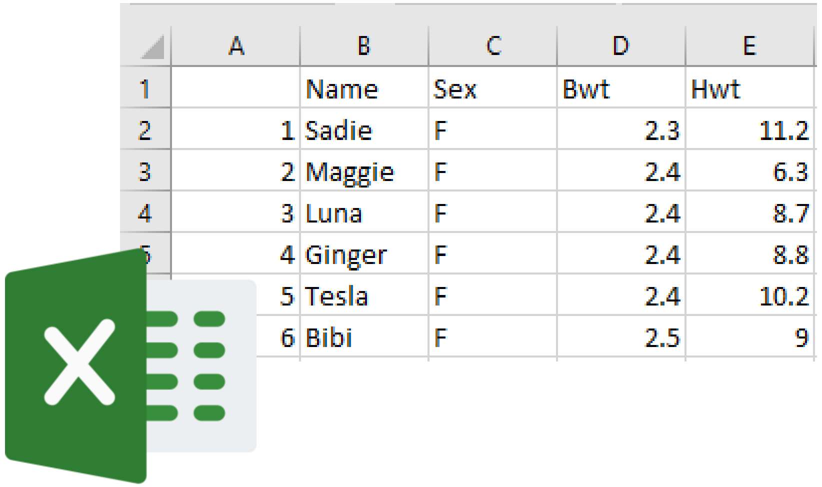
```
> library("MASS")
> sessionInfo()
R version 3.5.1 (2018-07-02)
Platform: x86_64-w64-mingw32/x64 (64-bit)
Running under: Windows >= 8 x64 (build 9200)

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
```

FINDING & READING DATA

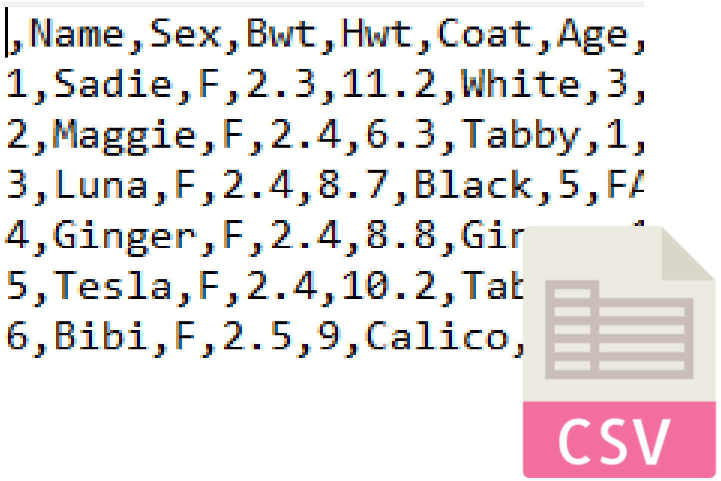


.CSV FILES

- 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,FA
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!

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

```
> read.csv(“mydataset.csv”)          # Read a file in the working directory

> read.csv(file.choose())            # File locator
```

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

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

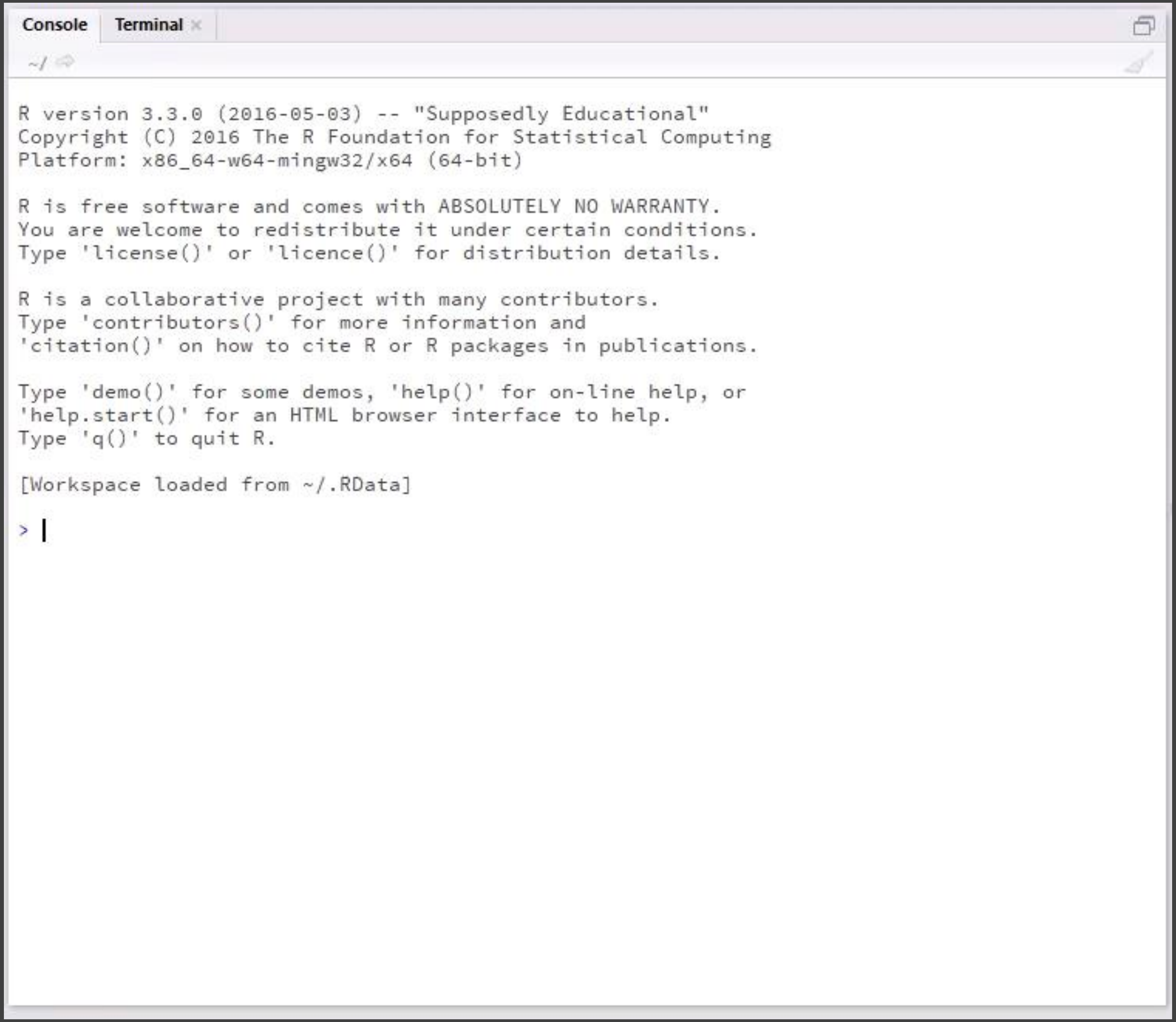
Set your working directory to the folder where your data is

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

  ```
  > setwd(“/Users/mfk8/Downloads”)
  ```

- A Windows example (note forward slashes):

  ```
  > setwd(“C:/Users/mfk8/Downloads”)
  ```



```
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

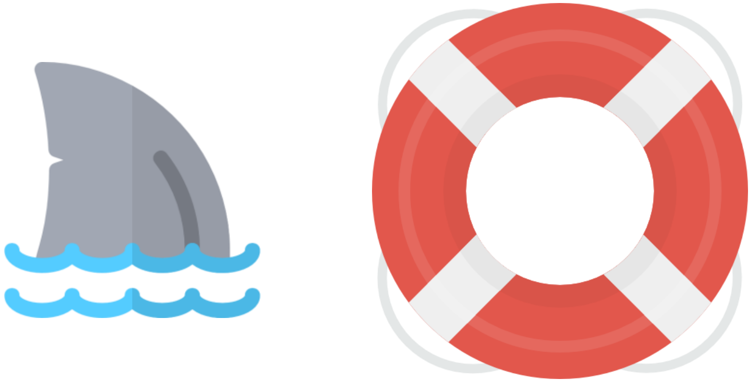
CRAN
The Comprehensive R Archive Network

```
> install.packages()            # Download and install packages

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

Try it!

GETTING HELP

```
> help.start()     # Manuals and reference guides

> help(t.test)     # Dispay 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:**

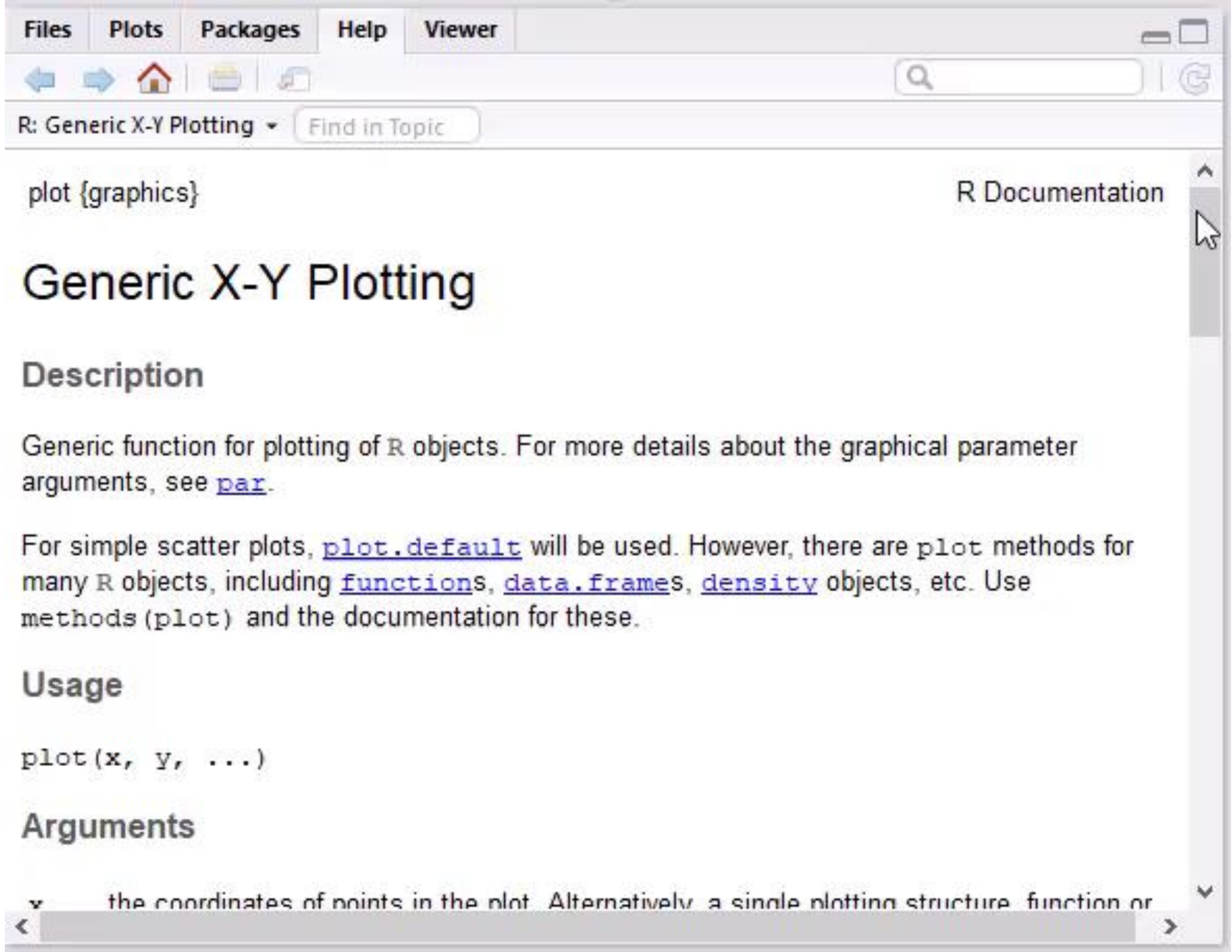
```
> args(plot)
```

- **CONSOLE OUTPUT:**

function (x, y, …)

If you would like more information:

```
> help(plot)
```



plot {graphics} R Documentation

Generic X-Y Plotting

Description

Generic function for plotting of R objects. For more details about the graphical parameter arguments, see par.

For simple scatter plots, plot.default will be used. However, there are plot methods for many R objects, including functions, data.frames, density objects, etc. Use methods(plot) and the documentation for these.

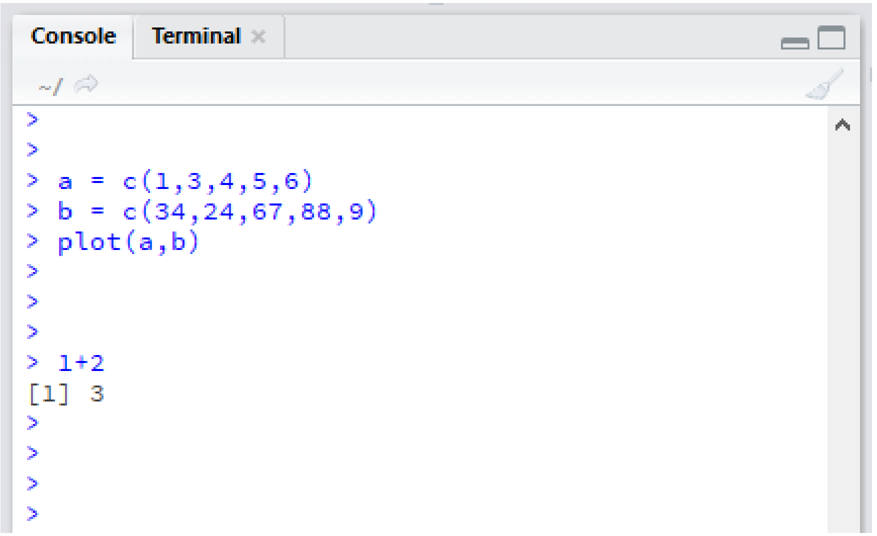
Usage

```
plot(x, y, ...)
```

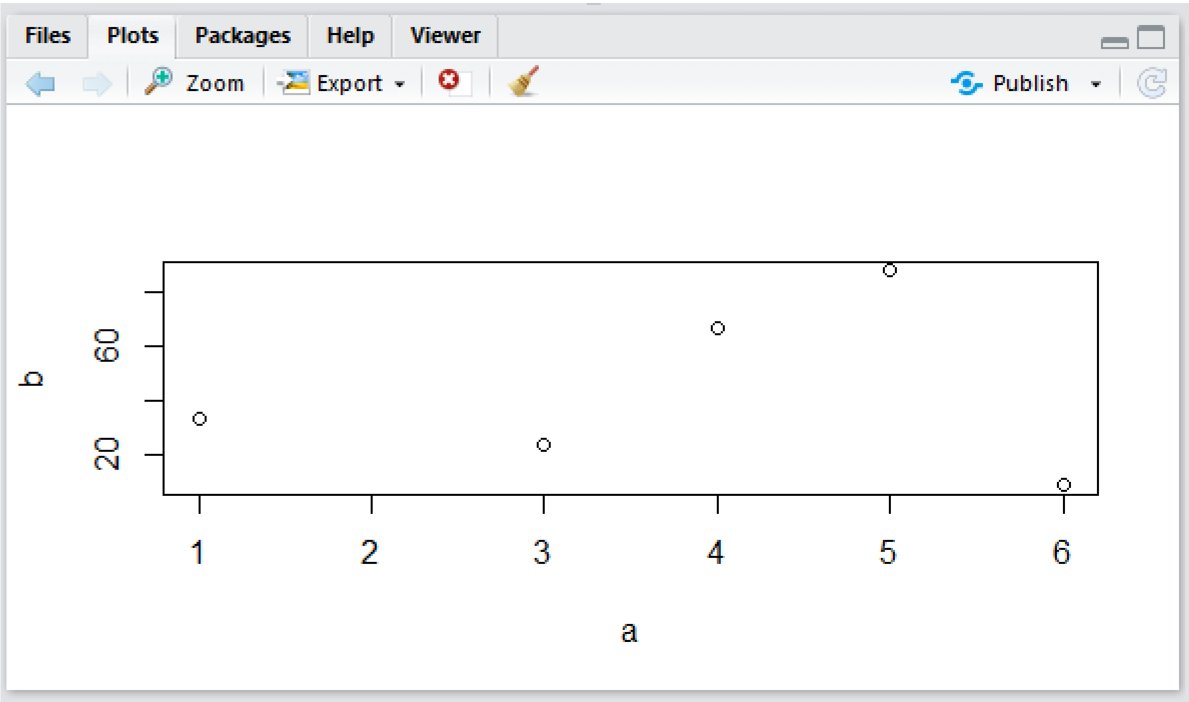
Arguments

x the coordinates of points in the plot. Alternatively, a single plotting structure, function or

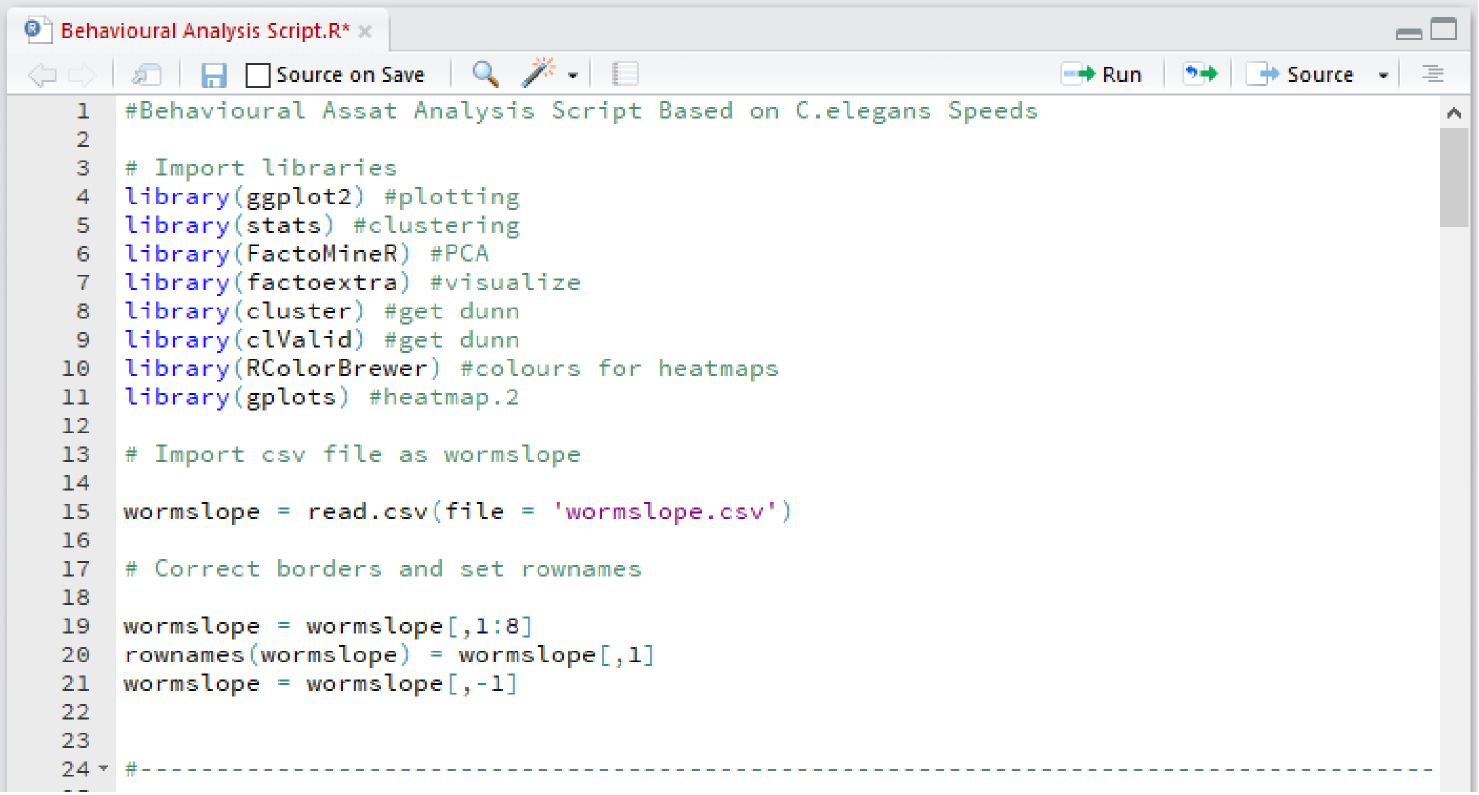
COMMAND LINES & SCRIPTS



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

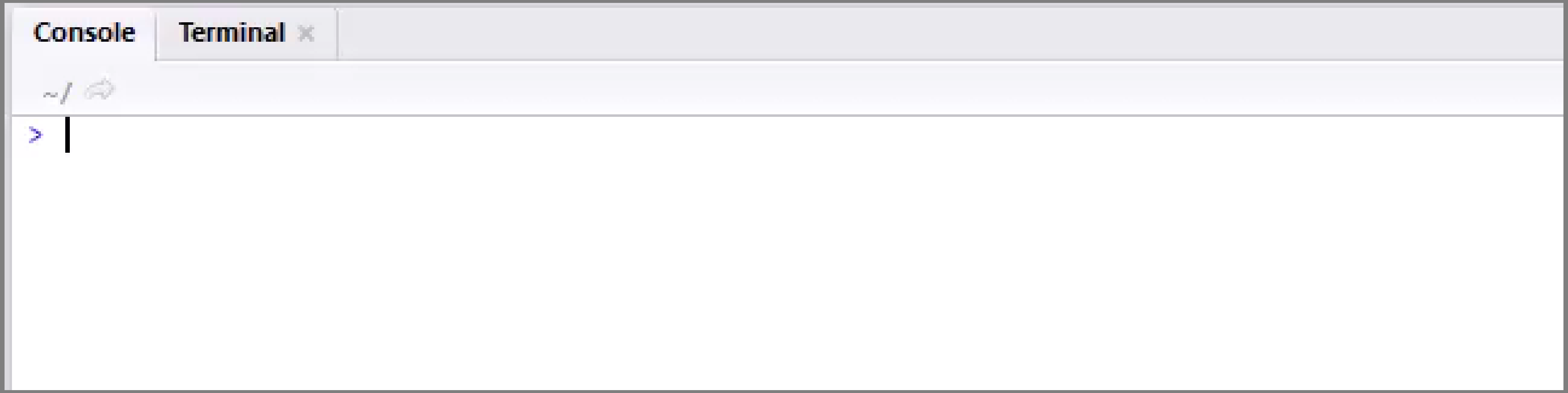


COMMAND LINES & SCRIPTS



```r
#Behavioural Assat Analysis Script Based on C.elegans Speeds

# Import libraries
library(ggplot2) #plotting
library(stats) #clustering
library(FactoMineR) #PCA
library(factoextra) #visualize
library(cluster) #get dunn
library(clValid) #get dunn
library(RColorBrewer) #colours for heatmaps
library(gplots) #heatmap.2

# Import csv file as wormslope

wormslope = read.csv(file = 'wormslope.csv')

# Correct borders and set rownames

wormslope = wormslope[,1:8]
rownames(wormslope) = wormslope[,1]
wormslope = wormslope[,-1]


#-------------------------------------------------------------------------------------
```

SAVING & CLOSING YOUR SESSION



```
> 
```

EXPLORING R

R OBJECTS

CREATING VARIABLES IN R

- Assign variables with a <- (traditional) or = (more modern 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
```

Run the code!

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”
```

Run the code !

- 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)

Notice how rep() creates repeats with minimal effort!

- Create a variable “gender” with 2 "male" entries and 4 "female" entries

```
> gender = c(rep("male", 2), rep("female", 4))
> gender_factor = factor(gender)
> gender_factor
[1] male male female female female female Levels: female male
```

Now 1=female, 2=male **internally** (alphabetically)

R now will treat ‘gender’ as a **nominal variable with 2 levels**

MATRICES

Run the code!

- 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
```

Remember: the lowest common denominator is usually character!

- To turn a DF into a matrix for certain operations:

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

Note: This turns data into all the same type

ADDING & 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)
```

A SELECTION OF HANDY FUNCTIONS

Try it!
Explore bird_data with these functions.

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

MORE HANDY FUNCTIONS!

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

Try it!
If you enter head(bird_data), how many rows does 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
```

Try it!

Practice .

Remember to encapsulate the vector in c().
Example: new_vec = c(1,2,3,4)

BUT BETTER!

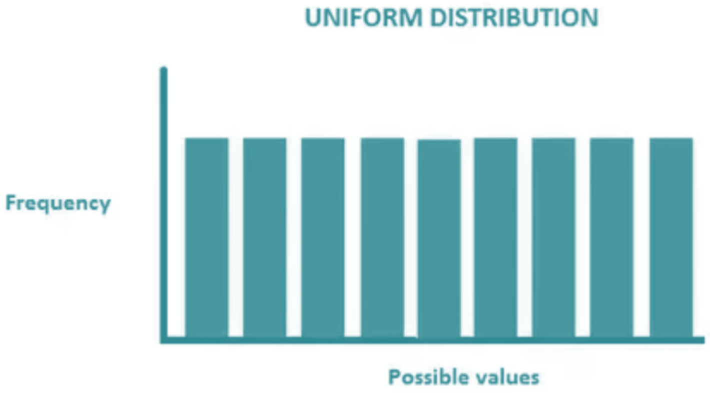
MORE R BUILT-IN MATH FUNCTIONS!

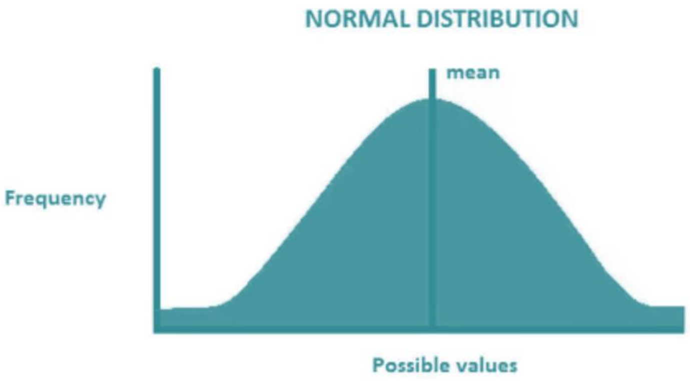
```
> log(10)                     #natural log (base e)
> exp(2.302585)               #antilog (e raised to power)
> log10(100)                  #log base 10
> sqrt(88)                    #square root
> factorial(8)                #factorial
> choose(12, 8)               #combinations (binomial coefficients)
> round(log(10), digits=3)    #round to specified digits
> abs(18 / -12)               #absolute value
```

BUT BETTER!

MORE R BUILT-IN MATH FUNCTIONS!

```
> runif(5)          #random numbers from uniform distribution
> rnorm(5)          #random numbers from normal distribution
```





SERIES SHORTCUTS

- Series: colon or "seq"

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

- Repeat

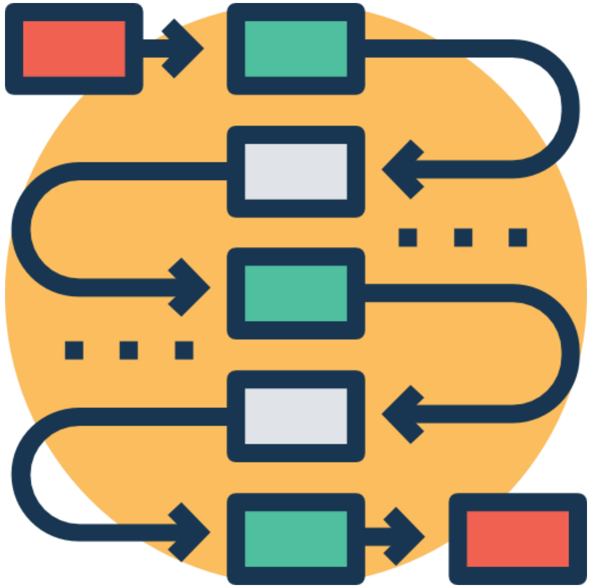
```
> rep(what, times)
> rep(10, 3)
```

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
```

Try it !

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..

```
multiplier = function(x,y) {
  x * y
}


> num_1 = 3
> num_2 = 2

> multiplier(num_1, num_2)
```

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)
```

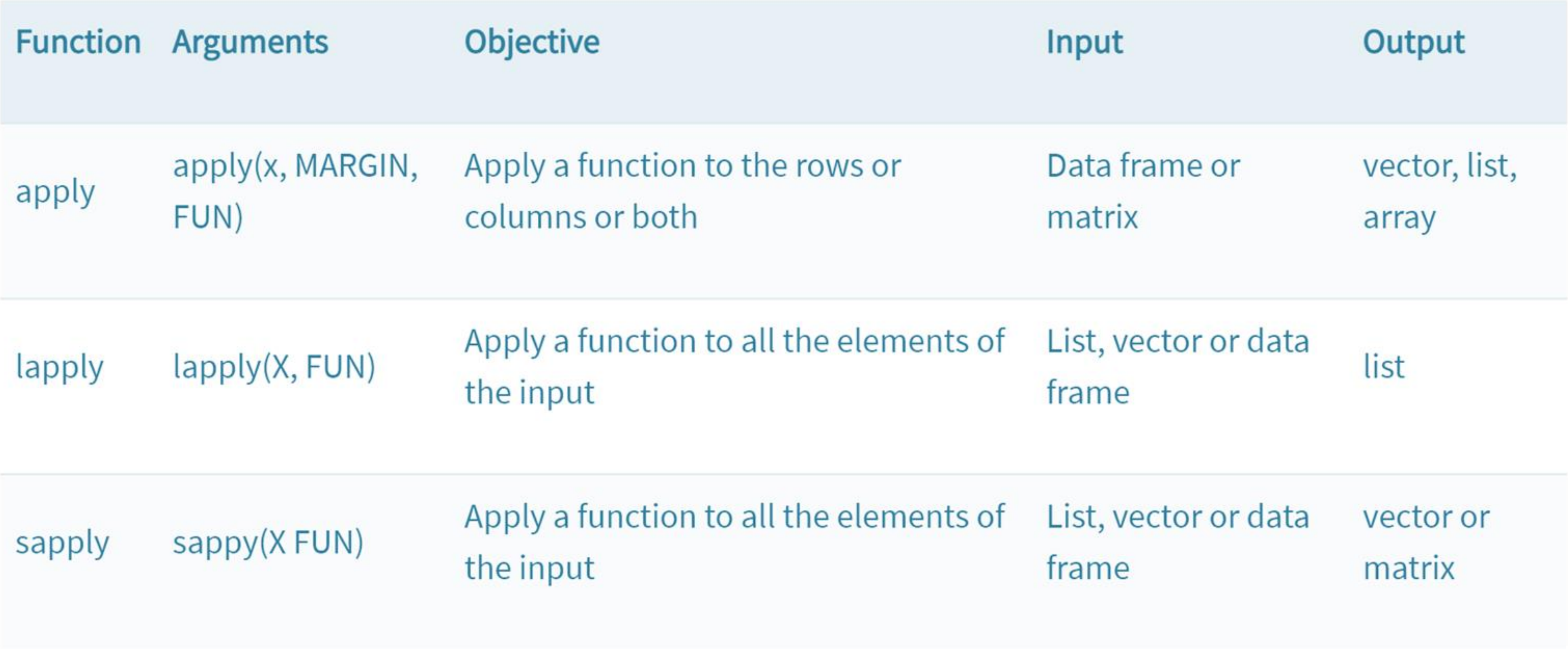
About how: “1” is to apply over rows, “2” is to apply over columns

```
> mymatrix = matrix(c(1:6), nrow=3, ncol=2)
> apply(mymatrix,1,sum)
[1] 5 7 9
```

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

Your Turn:
Try it with columns!

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	sappy(X FUN)	Apply a function to all the elements of the input	List, vector or data frame	vector or matrix

Table taken from https://www.guru99.com/r-apply-sapply-tapply.html

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

Tidyverse
dplyr
ggplot2
TIBBLE
readr
tidyr
purrr

Living the R Life:
An Example

CLASS EXAMPLE

OUR DATASET

If these formats don’t work for you, try:

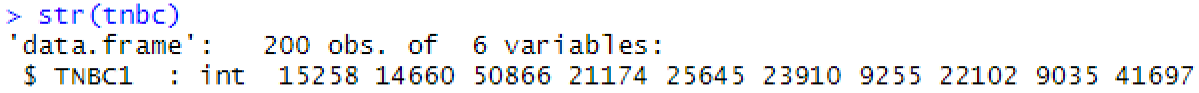
```
> setwd(“C:\\Users\\mkf8\\Downloads”)
```

- Import your new dataset with headers and row names.

```
> tnbc = read.csv(‘tnbc.csv', header = T, row.names = 1)
```

CLASS EXAMPLE
IMPORTING & VIEWING DATA

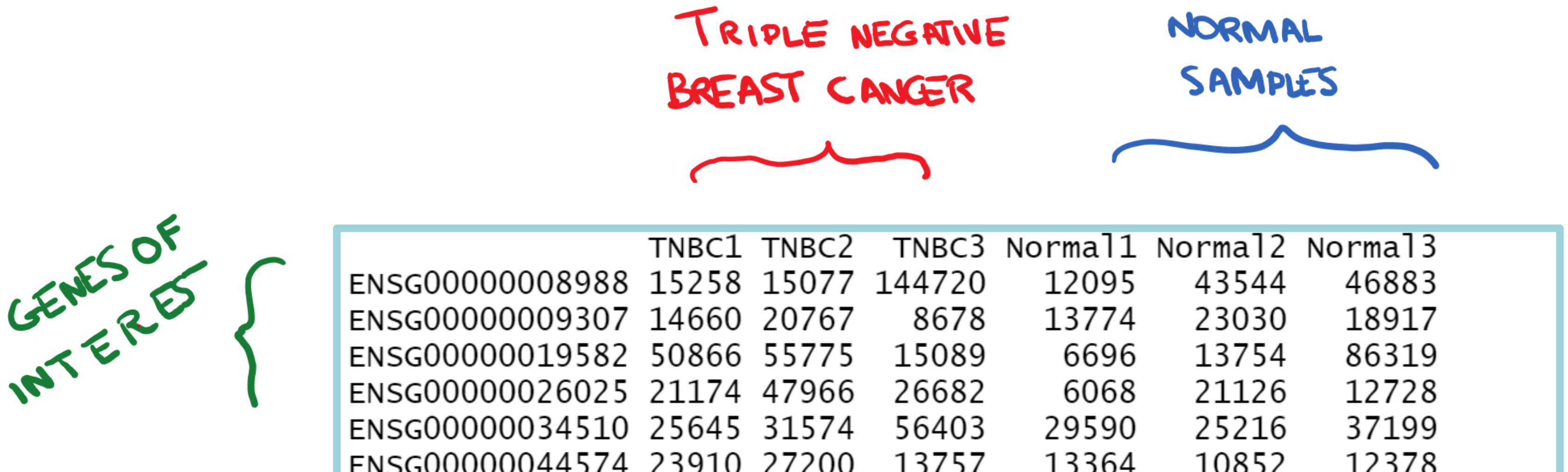
- Obtain structure just like you did with bird_data.



```
> str(tnbc)
'data.frame':	200 obs. of  6 variables:
 $ TNBC1  : int  15258 14660 50866 21174 25645 23910 9255 22102 9035 41697
```

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

> head(tnbc)

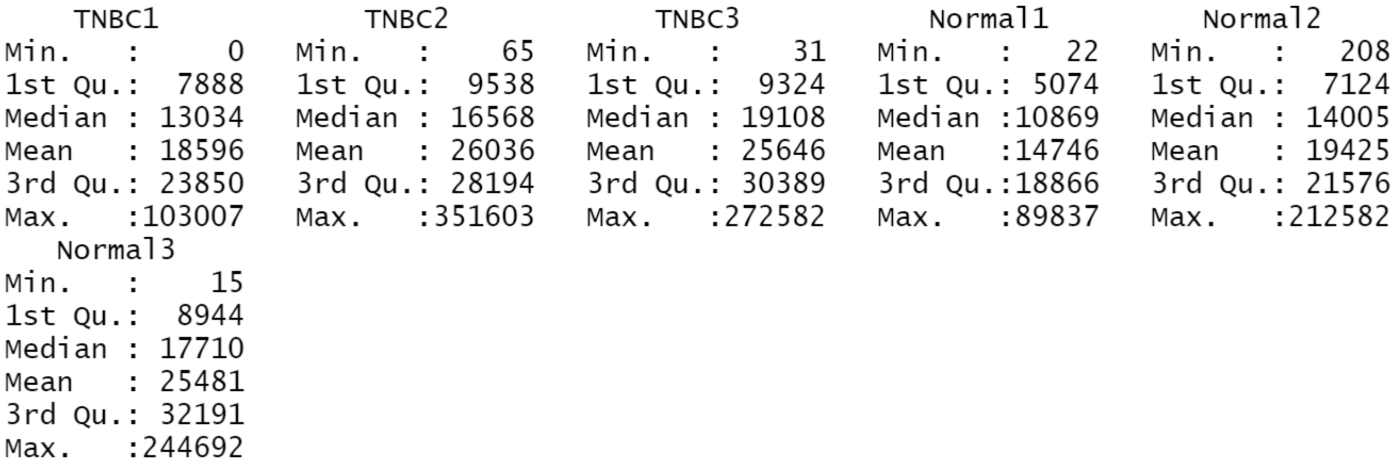


```
                TNBC1 TNBC2  TNBC3 Normal1 Normal2 Normal3
ENSG00000008988 15258 15077 144720   12095   43544   46883
ENSG00000009307 14660 20767   8678   13774   23030   18917
ENSG00000019582 50866 55775  15089    6696   13754   86319
ENSG00000026025 21174 47966  26682    6068   21126   12728
ENSG00000034510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

CLASS EXAMPLE
QUICK STATS

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

> summary(tnbc)



```
     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 some plotting and descriptive statistics is the best way to go!

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

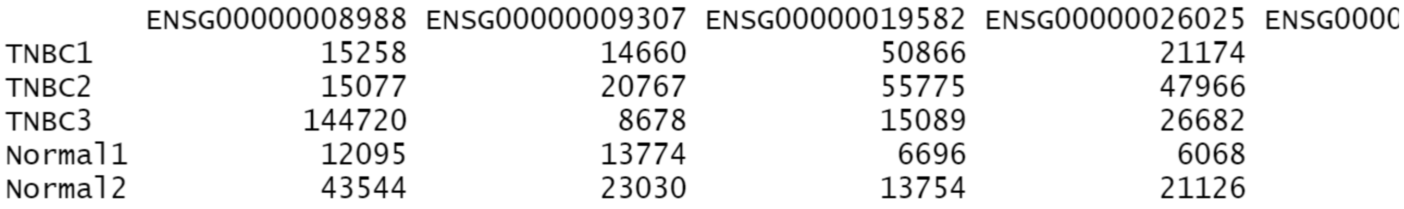
CLASS EXAMPLE
TRANSPOSING DATA

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

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

What happens?

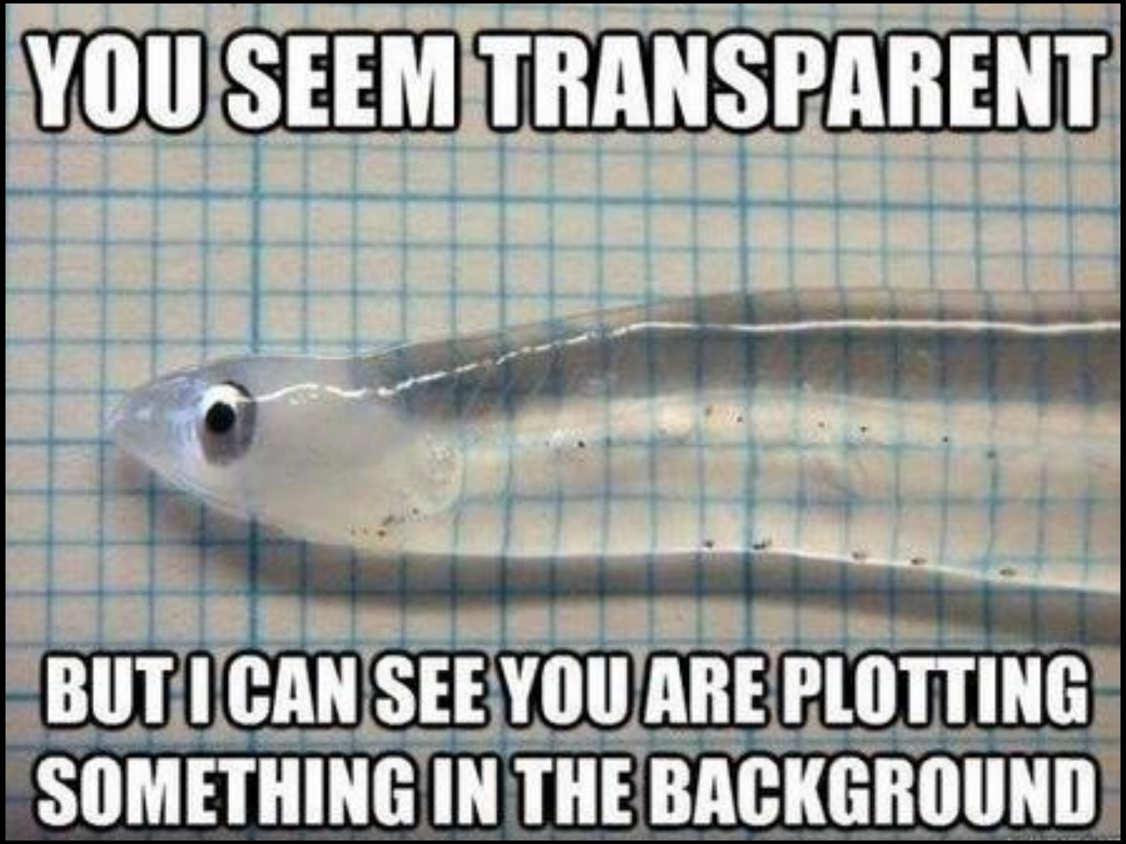
```
> tnbc_mat = as.matrix(tnbc)
> tnbc_mat_t = t(tnbc_mat)                    # 't' is for 'transpose'
> head(tnbc_mat_t)
```



```
        ENSG00000008988 ENSG00000009307 ENSG00000019582 ENSG00000026025 ENSG0000C
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
```

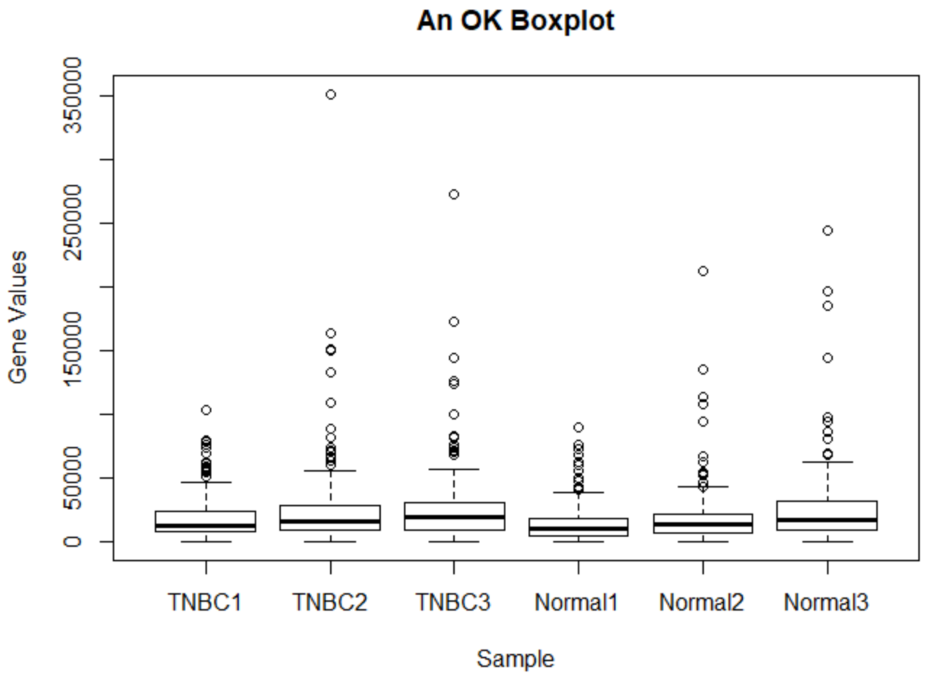
- as.data.frame() will turn you data into a dataframe again!

LET'S TRY SOME PLOTS!



CLASS EXAMPLE
BOXPLOT

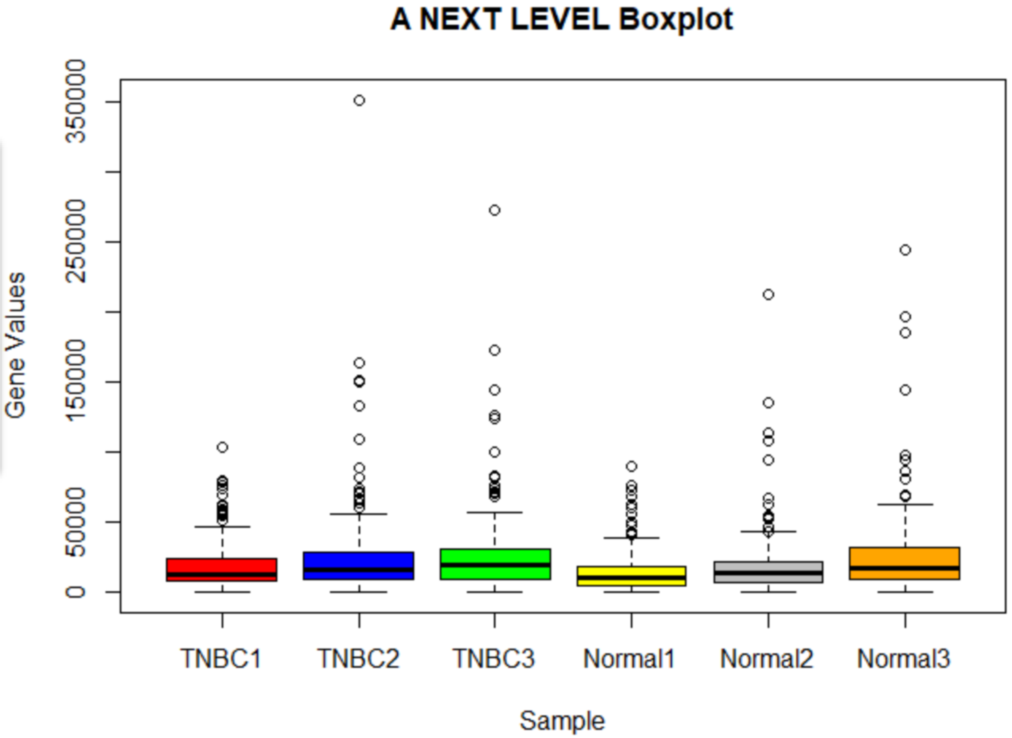
```
> boxplot(geneset, xlab = 'Sample', ylab = 'Gene Values', main = 'An OK Boxplot')
```



CLASS EXAMPLE
BOXPLOT

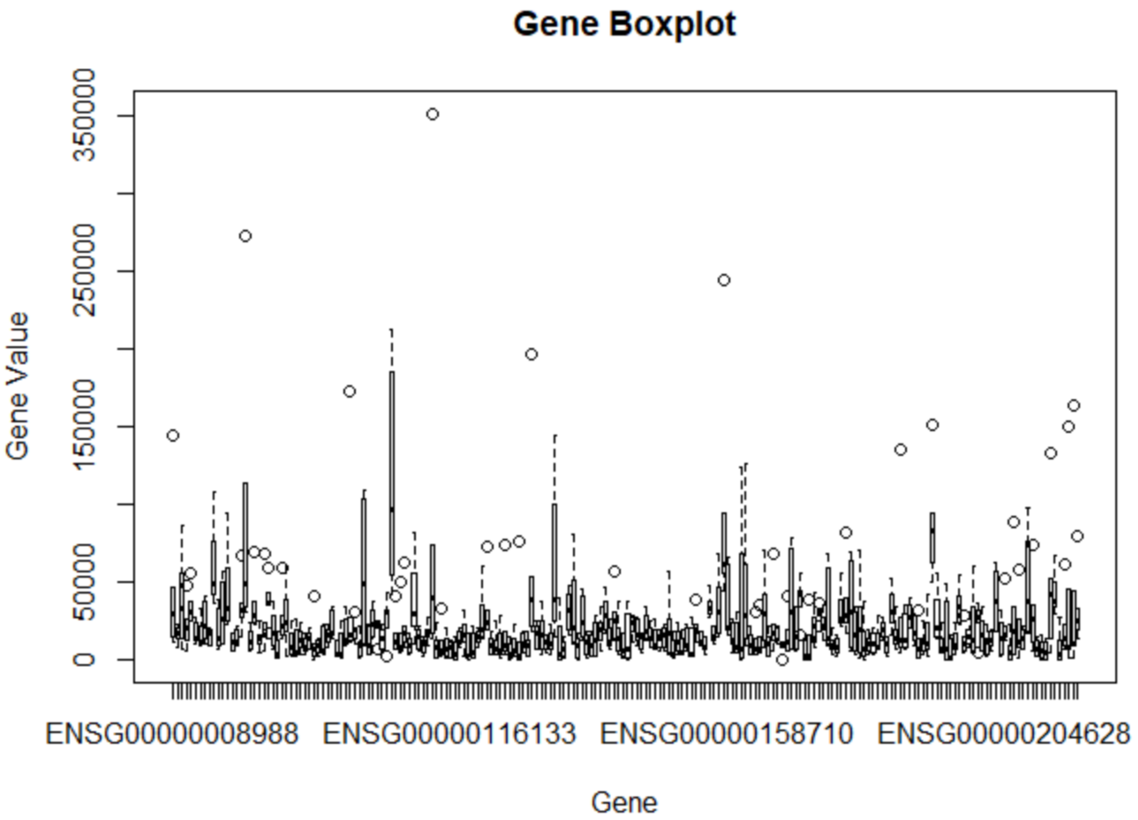
```
> boxplot(geneset, xlab = 'Sample', ylab = 'Gene Values', main = 'A NEXT LEVEL Boxplot',
col = c('red', 'blue', 'green', 'yellow', 'grey', 'orange'))
```

Can you propose a way to turn all TNBC mice in one color and all control in another?



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!

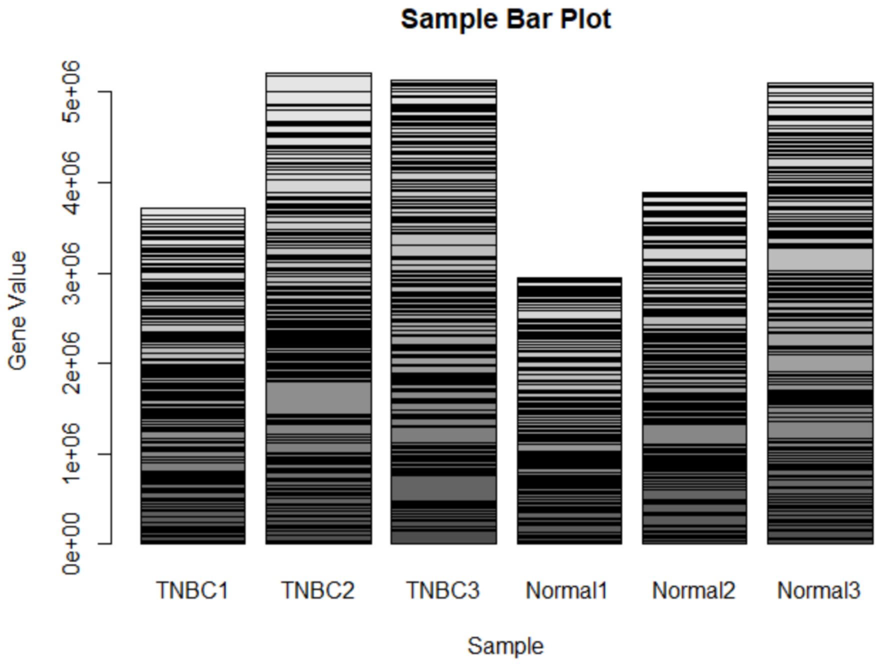
• main = “Title”	# main title
• xlab= “x label”	# x-axis label
• ylab=“y label”	# y-axis label
• xlim(N,N)	# x-axis start, stop
• ylim(N,N)	# y-axis start, stop
• col =c(“color1”, “color2”)	# vector with colors
• cex= N	# size of text and symbols
• pch= N	# plot point symbol type

CLASS EXAMPLE
BARPLOTS

Your Turn:
Try to turn the plot blue!

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

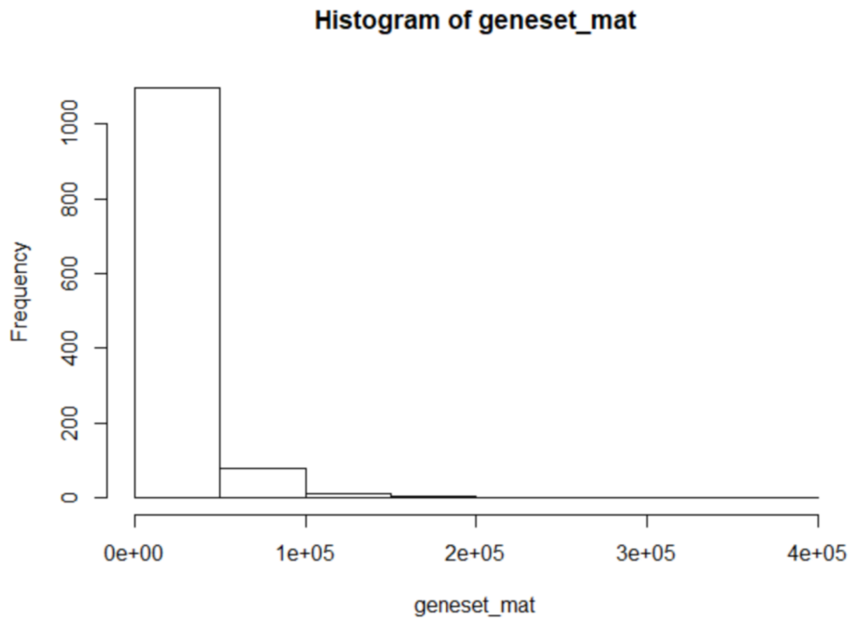
```
> barplot(geneset_mat, xlab = 'Sample', ylab = 'Gene Value', main = 'Sample Bar Plot')
```



CLASS EXAMPLE
HISTOGRAMS

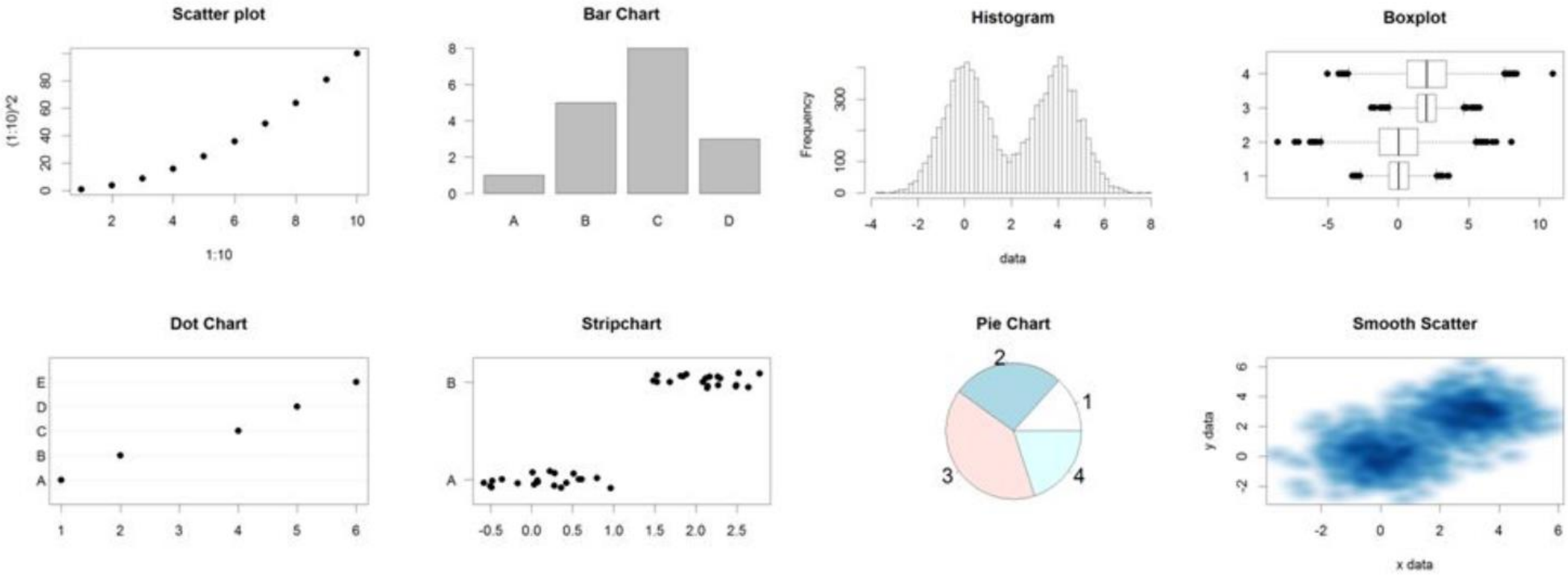
- Plot a histogram of the frequency of values in our dataset

```
> hist(geneset_mat)
```



OTHER PLOT TYPES AVAILABLE IN R

Anything is possible!



Scatter plot

Bar Chart

Histogram

Boxplot

Dot Chart

Stripchart

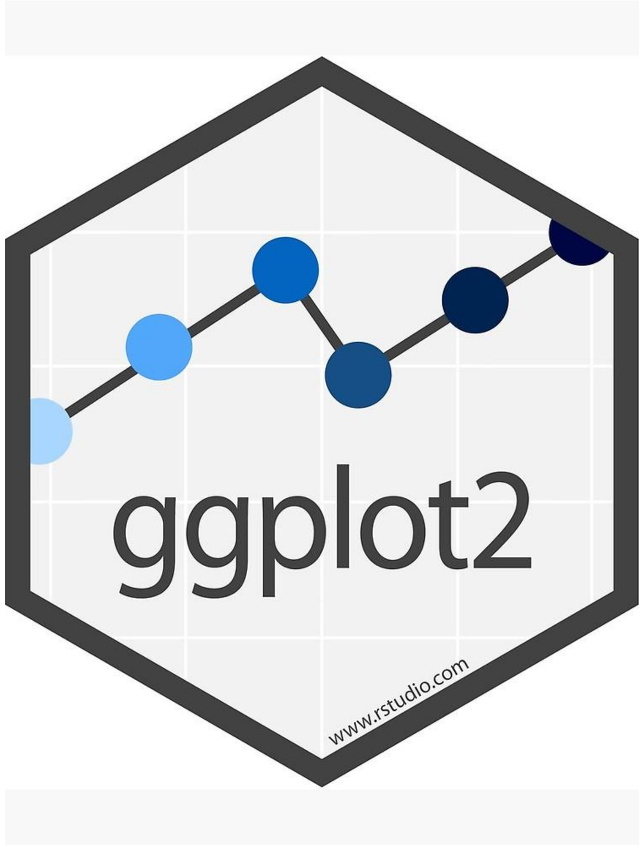
Pie Chart

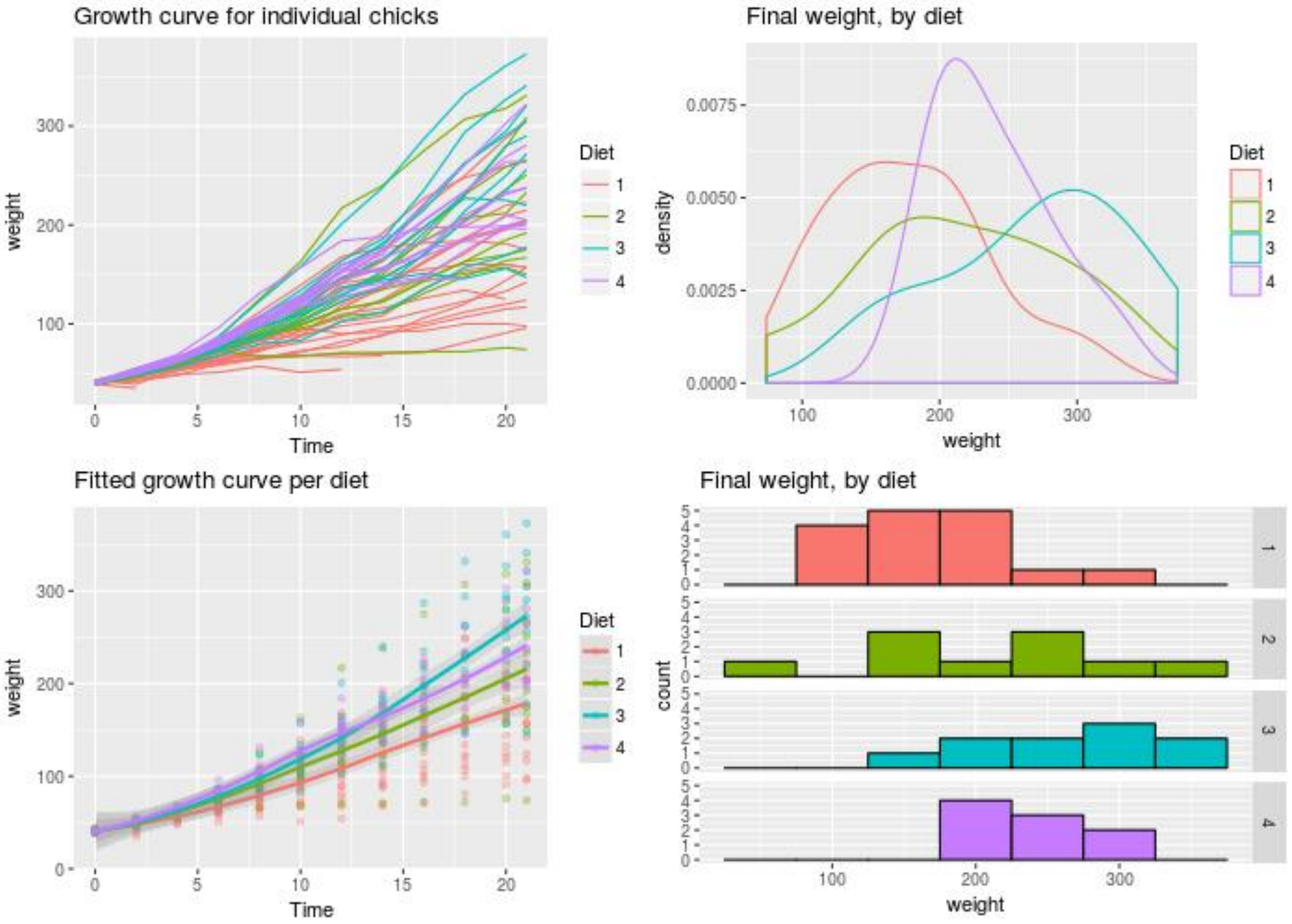
Smooth Scatter

Slide from Simon Andrews (Babraham Bioinformatics)

POPULAR PLOTTING PACKAGE

GGPLOT 2





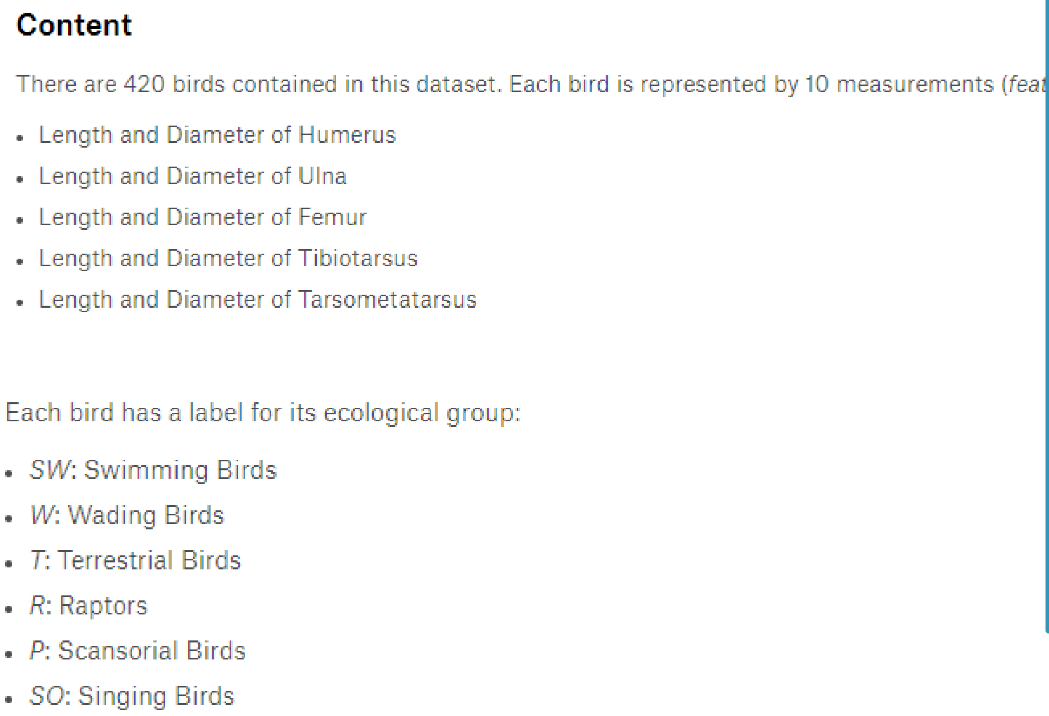
From http://www.cookbook-r.com

CLASS ACTIVITY

BIRD BONES

BIRD BONES
CLASS ACTIVITY

- Have a look at the bird dataset.



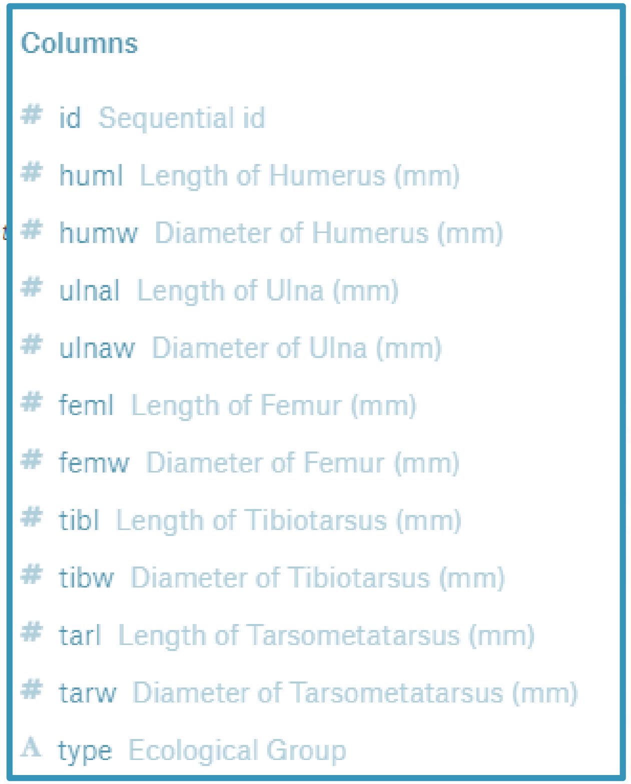
Content

There are 420 birds contained in this dataset. Each bird is represented by 10 measurements (feat

- 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

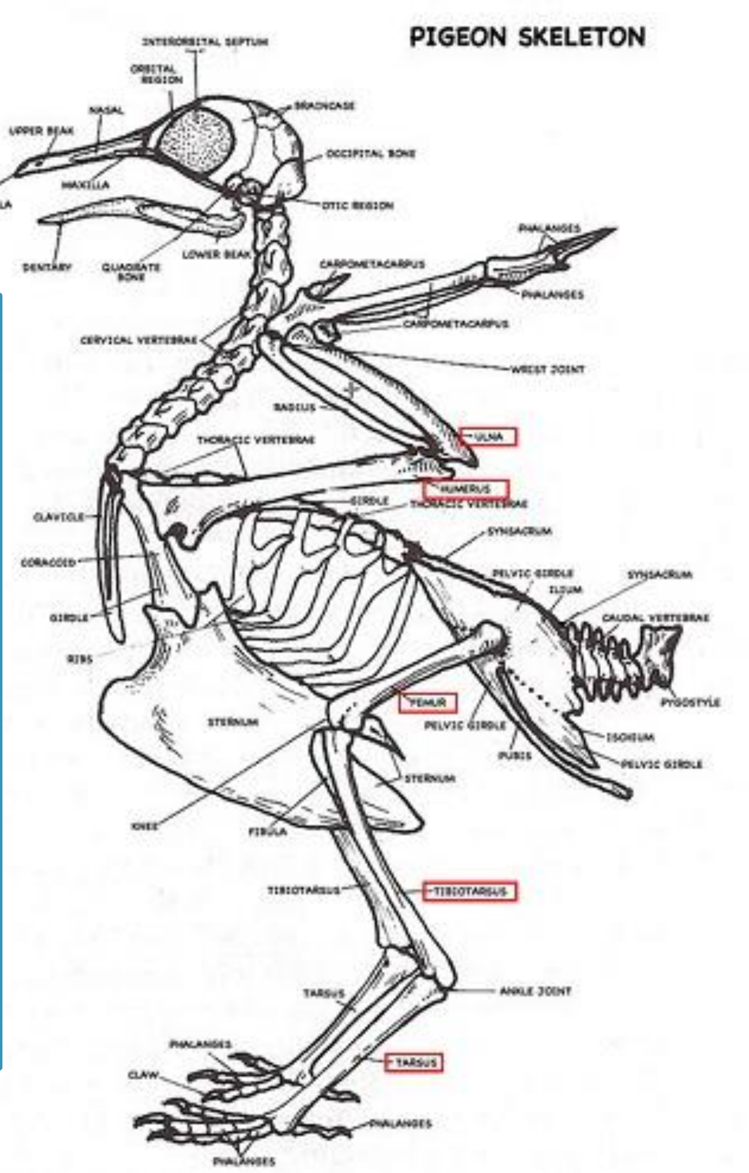
Each bird has a label for its ecological group:

- *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



https://www.kaggle.com/zhangjuefei/birds-bones-and-living-habits

BIRD BONES
CLASS ACTIVITY

1) Have a look at the bird dataset.

2) Plot a histogram of huml 'Length of Humerus' from the bird dataset.

3) What did you see?

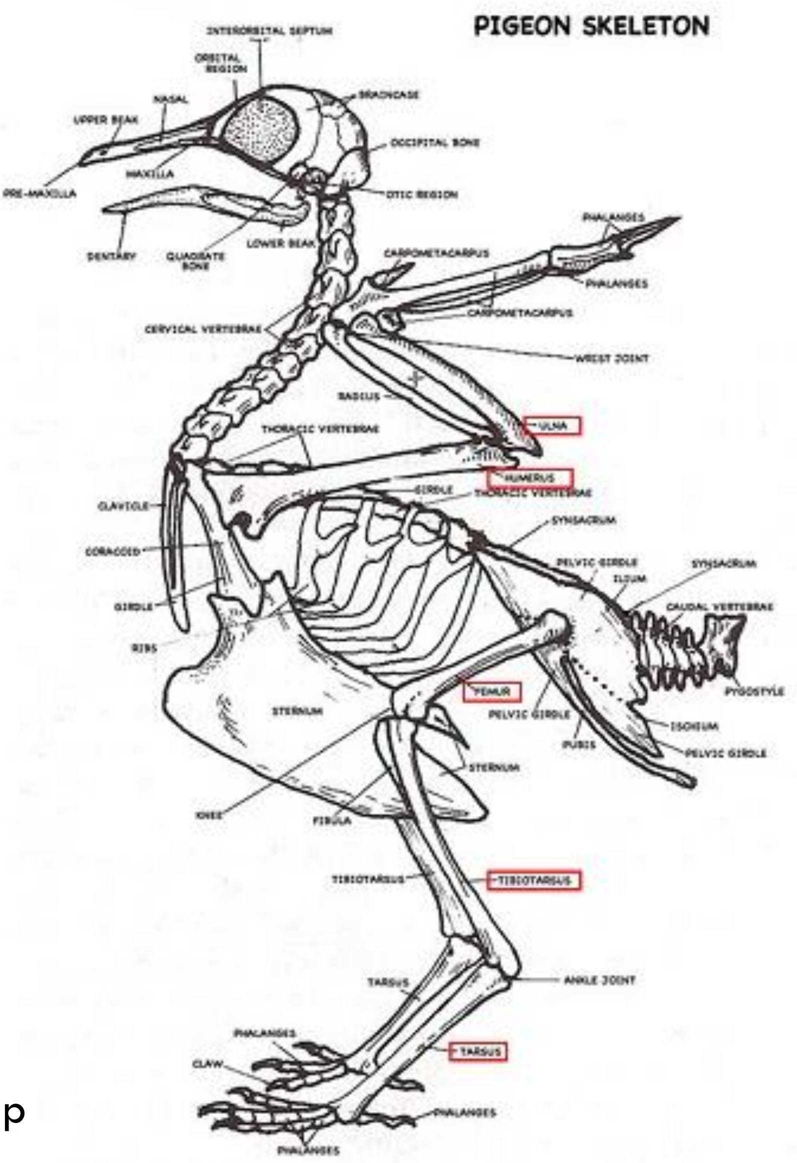
Hint: You can use $ to subset columns from dataframes

4) What happens if you use plot() with 'huml' and 'feml'?

5) Let's do something crazy: plot() the entire dataset! What do you see?

Bonus Question

6) In your original plot (4), can you make the points colors match their ecological group (column: 'type')



LET'S SUM IT UP!

WHAT DID WE LEARN IN TODAY'S LESSON?

- Intro to R objects
- How to do basic math in R
- Handle dataframes
- Basic Plotting in R

Thank you very much!