An introduction to R

Jianguo (Jeff) Xia, Assistant Professor
Institute of Parasitology, and Department of Animal Science
McGill University

June 1, 2017

http://www.xialab.ca

Objective & outline

- To get comfortable with R!

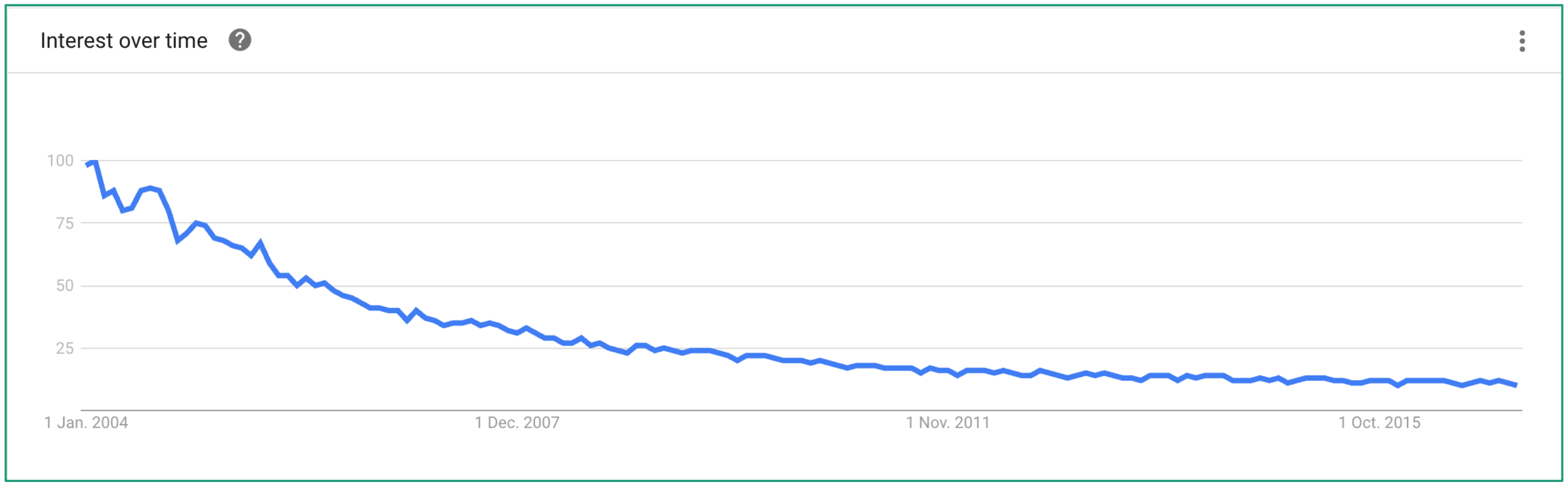
1. Why R?
2. Review basic R commands
3. Review basic R data types
    - Vector, matrix, list, data frame
4. Review basic R graphics
    - Scatter plot, box plot, histogram
5. Write a simple R script!

Survey

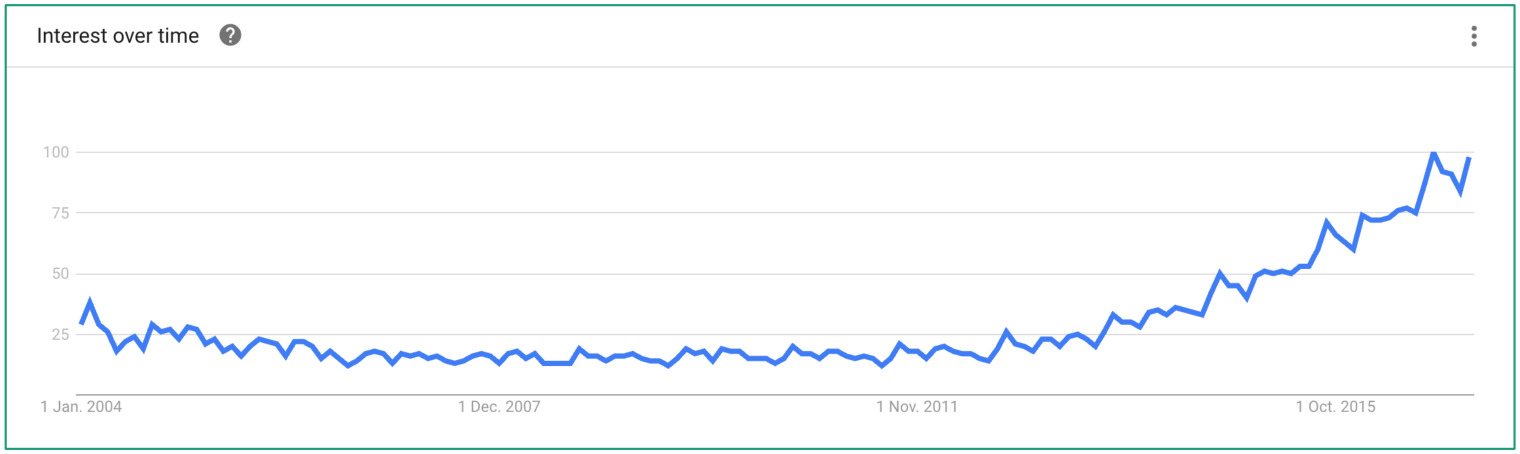
- What is your experience in R?
  a. Never used at all
  b. Used in my work but very basic
  c. Using it rather regularly in my work
- Do you have experience in any other language?
  - Java, JavaScript, C, C++, …
  - Perl, Python, Ruby, …

The Shift of Landscape (Google Trend)

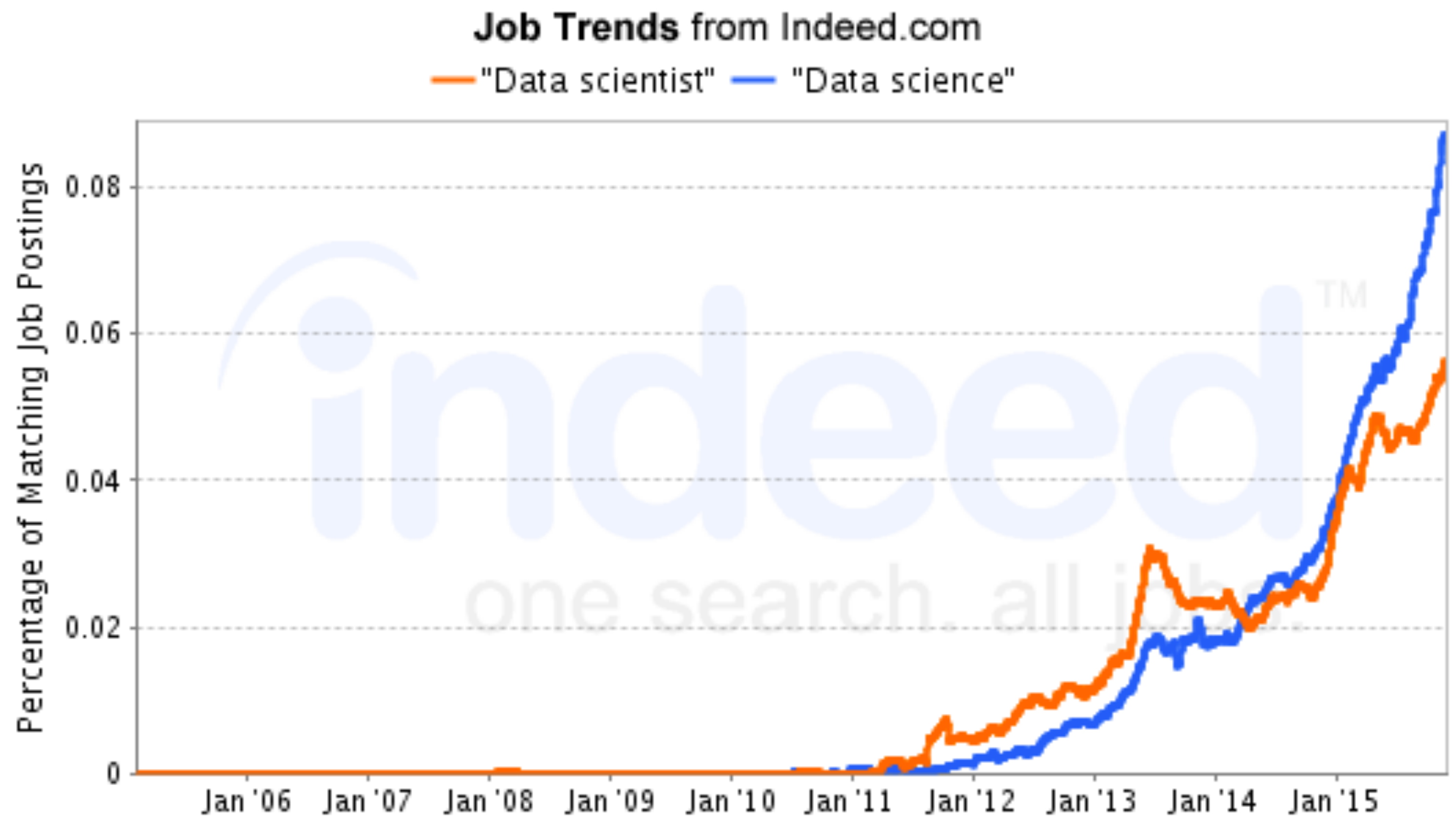
- Decline of Bioinformatics



-



Rise of Data Science



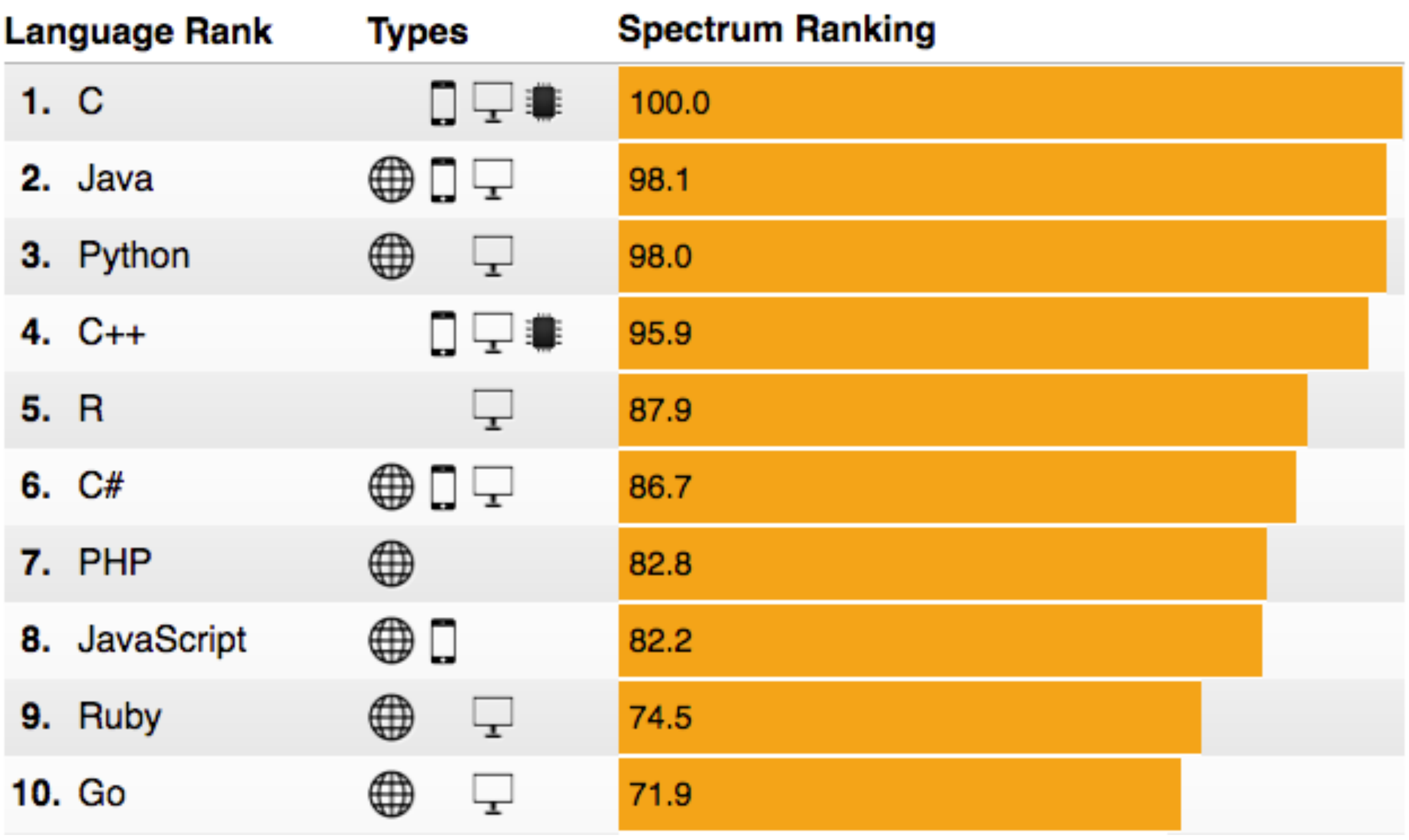
https://opensource.com/business/14/12/r-open-source-language-data-science

Embracing Data Science

- Bioinformatics is now considered part of growing the field of Data Science
- Data science is an interdisciplinary field about processes and systems to extract knowledge or insights from data in various forms, either structured or unstructured
- Big data is embraced by all big players:
  - Academics: NIH, EBI
  - Industry: Google, Amazon, IBM, Microsoft, Facebook

The 2016 Top Programming Languages

C is No. 1, but big data is still the big winner



Language Rank	Types	Spectrum Ranking
1. C	📱 🖥 ▪	100.0
2. Java	🌐 📱 🖥	98.1
3. Python	🌐 🖥	98.0
4. C++	📱 🖥 ▪	95.9
5. R	🖥	87.9
6. C#	🌐 📱 🖥	86.7
7. PHP	🌐	82.8
8. JavaScript	🌐 📱	82.2
9. Ruby	🌐 🖥	74.5
10. Go	🌐 🖥	71.9

http://spectrum.ieee.org/computing/software/the-2016-top-programming-languages

What is R

- A programming language
- A computing environment for:
  - Statistics
  - Machine learning
  - Graphics
- Open source
  - Main language in teaching statistics
  - Main statistics language at Google
  - Main language in statistical analysis of “omics” data
    - Microarray, RNAseq, metagenomics, metabolomics

Working with R

- Command line interface
- A text editor
  - Linux: nedit, gedit
  - Mac: TextMate, TextWrangler
  - Windows: Notpad ++
- R Studio
  - https://www.rstudio.com/

Syntax

- The **$** indicate terminal command line
- The **>** is for R command line
- The arrow **<-** is the assignment operator in R
  - Equal sign = is now supported but <- is preferred
- The **#** is for comments in script
  - Message for yourself or other developer
  - Ignored by R during execution

```
> weight.a <- 10
> weight.a
[1] 10
# get sum of numbers
> sum(c(2, 6, 8))
```

Interactive R command line

1. Open a terminal
2. Create a new, empty working directory
3. Navigate to the directory
4. Start R
5. Play around
   - getwd() # current position
   - dir() # show files in current directory
   - setwd(…) # move to new dir
6. Exit R: q()

```
$ mkdir  ReviewR
$ cd ReviewR
$ R
> getwd()
[1] "/Users/xia/ReviewR"
> dir()
character(0)
> print (“hello world!”)
[1] "hello world!"
> q()
```

A Powerful Calculator

- R can directly evaluate math expressions.
- Special symbols:
  - **NaN**: Not a Number
  - **Inf**: infinite number
  - **NA**: Not Available or missing value)
  - **NULL**: undefined value
  - **TRUE**: logical value true
  - **FALSE**: logical value false

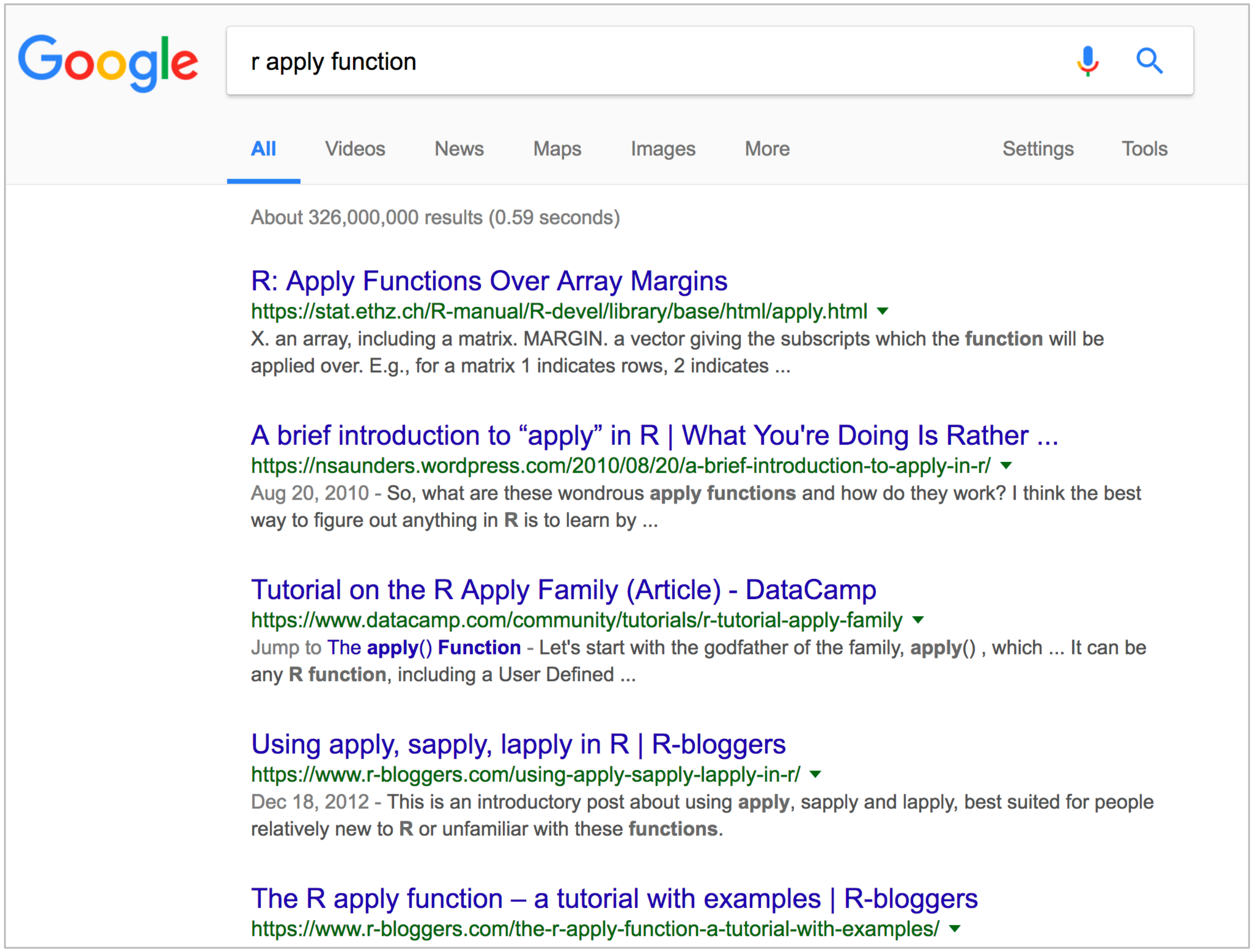
```
> 2+2
[1] 4
> 2^8
[1] 256
> exp(-2)
[1] 0.1353353
> log(-10)
[1] NaN
> 1/0
[1] Inf
```

Finding Help: ?

```
> ?apply

apply                    package:base               R Documentation

Apply Functions Over Array Margins

Description:

     Returns a vector or array or list of values obtained by applying a
     function to margins of an array or matrix.

Usage:

     apply(X, MARGIN, FUN, ...)

Arguments:

       X: an array, including a matrix.
```

Tip: type **q** to exit at any time

Finding Help: Google



About 326,000,000 results (0.59 seconds)

R: Apply Functions Over Array Margins
https://stat.ethz.ch/R-manual/R-devel/library/base/html/apply.html
X. an array, including a matrix. MARGIN. a vector giving the subscripts which the **function** will be applied over. E.g., for a matrix 1 indicates rows, 2 indicates ...

A brief introduction to "apply" in R | What You're Doing Is Rather ...
https://nsaunders.wordpress.com/2010/08/20/a-brief-introduction-to-apply-in-r/
Aug 20, 2010 - So, what are these wondrous **apply functions** and how do they work? I think the best way to figure out anything in **R** is to learn by ...

Tutorial on the R Apply Family (Article) - DataCamp
https://www.datacamp.com/community/tutorials/r-tutorial-apply-family
Jump to The **apply**() **Function** - Let's start with the godfather of the family, **apply**() , which ... It can be any **R function**, including a User Defined ...

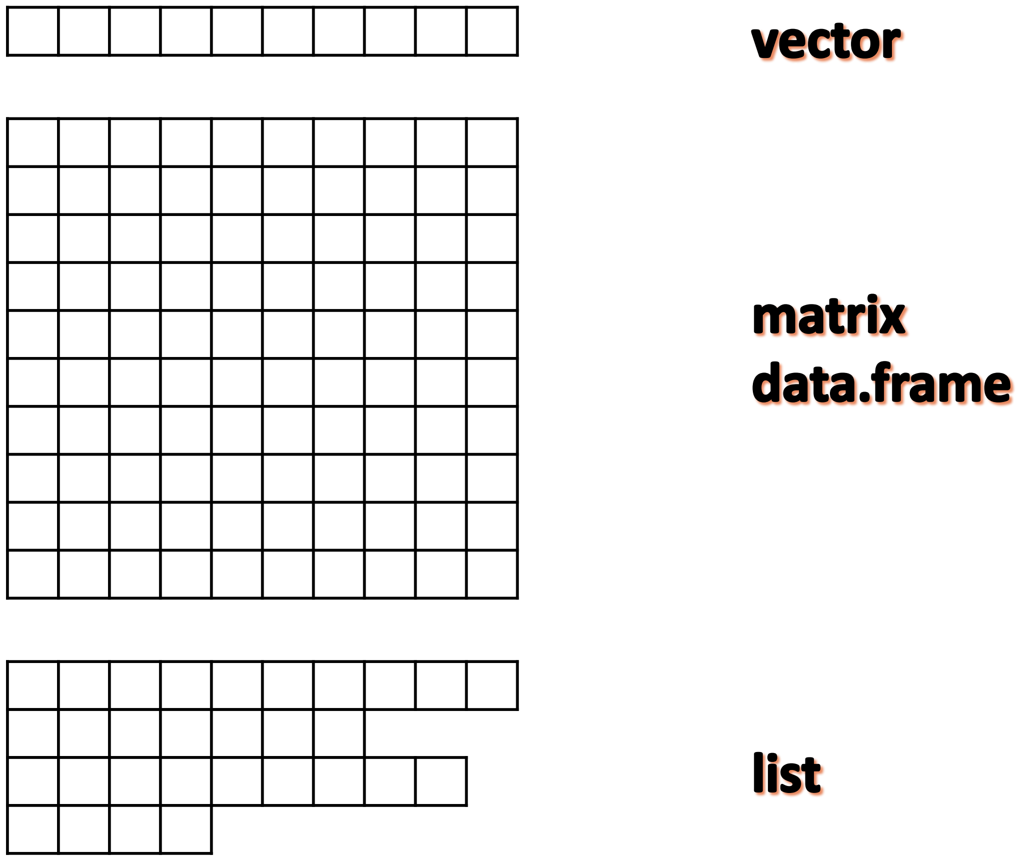
Using apply, sapply, lapply in R | R-bloggers
https://www.r-bloggers.com/using-apply-sapply-lapply-in-r/
Dec 18, 2012 - This is an introductory post about using **apply**, sapply and lapply, best suited for people relatively new to **R** or unfamiliar with these **functions**.

The R apply function – a tutorial with examples | R-bloggers
https://www.r-bloggers.com/the-r-apply-function-a-tutorial-with-examples/

Choose the ones that make sense to you!

Basic R Data Types

Visual summary of main R data types



Atomic Data Types

- In R the base type is a **vector**.
  - An indexed set of values that are all of the same type.
    - [1, 2, 3, 4, 5], [“a”, “b”, “c” …]
  - Mixing different types may cause unexpected results
- The type of the entries determines the class of the vector:
  - Integer, numeric, character, logical
- Check the class of a vector
  - class()
- Force to convert to a particular type:
  - as.integer(), as.numeric(), as.character(), as.logical()

Vector Types

```
> a <- 1:100
> class(a)
[1] "integer"
> a <- letters[1:10]
> a
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"
> class(a)
[1] "character"
> a <- rnorm(10) # built-in function, use "?rnorm" for more details
> a
 [1] -0.6768672  0.5078512  0.3473145  1.2601923 -0.1431014 -1.0052774
 [7]  0.7683541  0.7247683 -0.9169387  1.3272359
> class(a)
[1] "numeric"
> a <- a > 0
> a
 [1] FALSE  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE FALSE  TRUE
> class(a)
[1] "logical"
```

Vector Creation

- Concatenation (**c**), colon (**:**), seq, rep

```
> v1 <- c(1, 2, 3)
> v1
[1] 1 2 3
> v2 <- 1:3
> v2
[1] 1 2 3
> v3 <- seq(from=1, to=10, by=2)
> v3
[1] 1 3 5 7 9
> v4 <- rep(1, 4)
> v4
[1] 1 1 1 1
```

Vector Indexing

- Get vector elements:**[ ]**
  - By numeric index
    - ** Index start with 1, not 0
    - Use positive index to get specific element
    - Use negative to exclude elements
  - By name
    - For named vectors
  - By Boolean index (vector with logical values)
    - Same length
    - Return element with TRUE

```
> v <- c(1, 2, 3)
> v[2]
[1] 2
> v[-2]
[1] 1 3
> v[2:3]
[1] 2 3
> names(v) <- c(“a”, “b”, “c”)
> v[“b”]
b
2
```

Vector Operations

- Boolean vector, common functions, missing values,

```
> test <- rnorm(5)
> test
[1]  0.2255703  0.3354089 -0.7202021 -0.9523239 -1.5018892
> pos.inx <- test > 0
> pos.inx
[1]  TRUE  TRUE FALSE FALSE FALSE
> test[pos.inx] # using boolean index to get elements
[1] 0.2255703 0.3354089
> mean(test)
[1] -0.5226872
> sd(test)
[1] 0.7871872
> test <- c(test, NA)
> is.na(test)
[1] FALSE FALSE FALSE FALSE FALSE  TRUE
```

Factors

- A factor is a character vector augmented with information to store categorical data: **factor()**
  - Used to specify experimental design

```
> d1 <- c("M", "F", "M", "F", "F", "F”)
> d2 <- factor(d1)
> d2
[1] M F M F F F
Levels: F M
> table(d2)
F M
4 2
```

Matrix

- A matrix is table of same type: **matrix()**

```
> mat <- matrix(1:12, nrow=3, byrow=T)
> mat
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
> mat2 <- matrix(letters[1:8], nrow=2, byrow=F)
> mat2
     [,1] [,2] [,3] [,4]
[1,] "a"  "c"  "e"  "g"
[2,] "b"  "d"  "f"  "h”
> dim(mat2) # size of the matrix
[1] 2 4
```

Matrix Indexing

- Use the row and column positions to select values

```
> mat
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
> mat[2,3]  # single value
[1] 7
> mat[2,]   # row selection
[1] 5 6 7 8
> mat[,3]   # column selection
[1]  3  7 11
```

Note, the comma "**,**" matters!

Lists

- A collection of vectors or other data objects. It is a very flexible way to organize relevant information: **list()**

```
> x <- list(name="Jeff", height=180, language=c("R", "Java",
"Perl”))
> x
$name
[1] "Jeff"

$height
[1] 180

$language
[1] "R"      "Java"   "Perl”
```

List Indexing

- Elements can be accessed by name or by index [or [[

```
> x$name
[1] "Jeff”
> a <- x[1] # [ return a list
> a
$name
[1] "Jeff"
> class(a)
[1] "list”
> b <- x[[1]]  # [[ return a vector
> b
[1] "Jeff”
> class(b)
[1] "character"
```

Data Frames

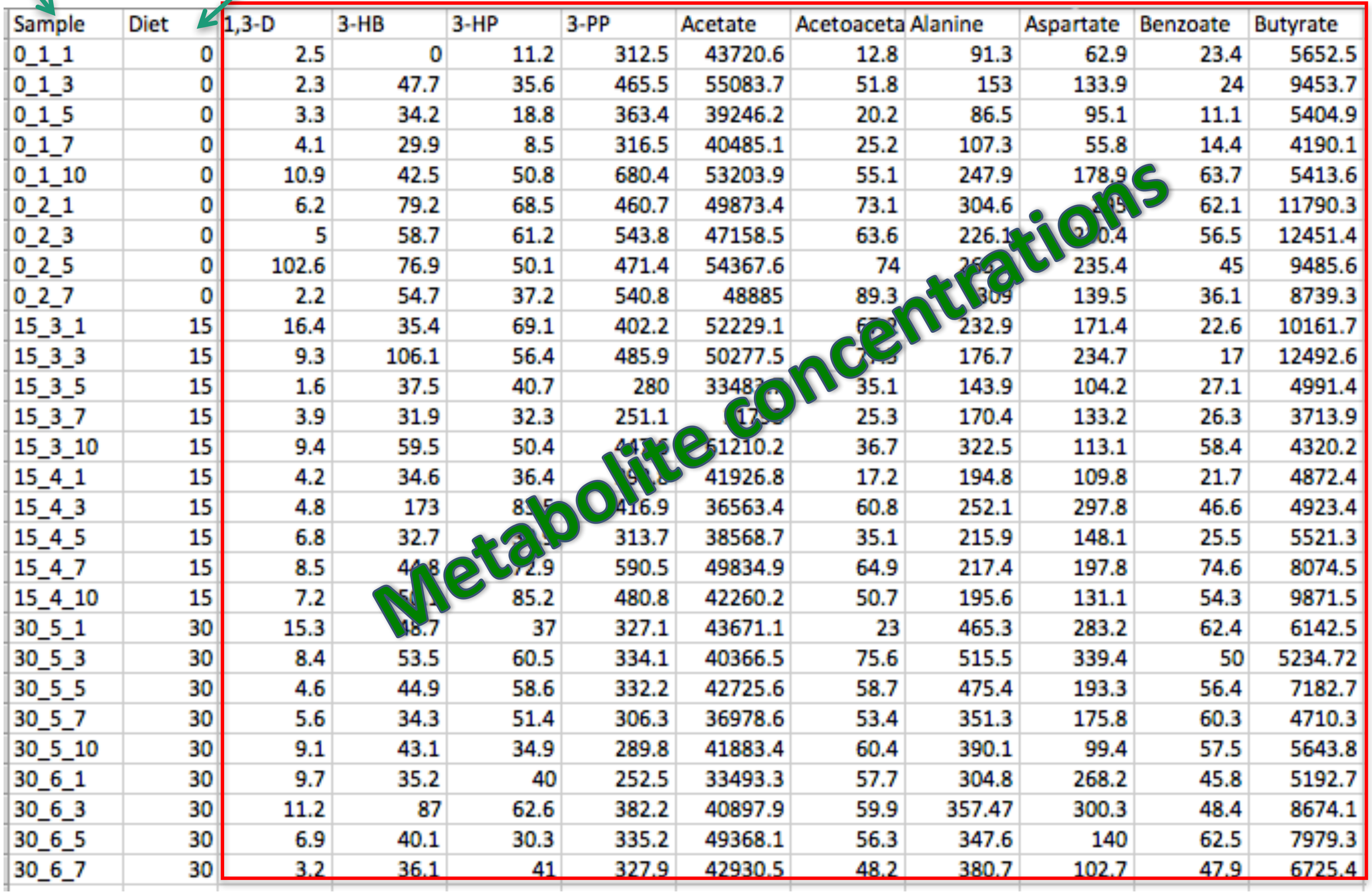
- Data frame is for storing tabular data such as Excels or CSV file: **data.frame()**
- shares many of the properties of matrices and of lists
  - Matrix-like (table) structure
  - Each column is a list of same length

```
> df <-data.frame(
    ID=c("Mike", "Tom", "Jon", "Mia"),
    age=c(12, 13, 12, 14),
    score=c(80, 85, 88, 95) );
> df
  name age score
1 Mike  12    80
2  Tom  13    85
3  Jon  12    88
4Mia  14    95
> df$age
[1] 12 13 12 14
> df[2,3]
[1] 85
```

An example data table

Sample IDs

Group Labels


Metabolite concentrations

Sample	Diet	1,3-D	3-HB	3-HP	3-PP	Acetate	Acetoaceta	Alanine	Aspartate	Benzoate	Butyrate
0_1_1	0	2.5	0	11.2	312.5	43720.6	12.8	91.3	62.9	23.4	5652.5
0_1_3	0	2.3	47.7	35.6	465.5	55083.7	51.8	153	133.9	24	9453.7
0_1_5	0	3.3	34.2	18.8	363.4	39246.2	20.2	86.5	95.1	11.1	5404.9
0_1_7	0	4.1	29.9	8.5	316.5	40485.1	25.2	107.3	55.8	14.4	4190.1
0_1_10	0	10.9	42.5	50.8	680.4	53203.9	55.1	247.9	178.9	63.7	5413.6
0_2_1	0	6.2	79.2	68.5	460.7	49873.4	73.1	304.6	[illegible]	62.1	11790.3
0_2_3	0	5	58.7	61.2	543.8	47158.5	63.6	226.1	[illegible]	56.5	12451.4
0_2_5	0	102.6	76.9	50.1	471.4	54367.6	74	[illegible]	235.4	45	9485.6
0_2_7	0	2.2	54.7	37.2	540.8	48885	89.3	309	139.5	36.1	8739.3
15_3_1	15	16.4	35.4	69.1	402.2	52229.1	[illegible]	232.9	171.4	22.6	10161.7
15_3_3	15	9.3	106.1	56.4	485.9	50277.5	[illegible]	176.7	234.7	17	12492.6
15_3_5	15	1.6	37.5	40.7	280	[illegible]	35.1	143.9	104.2	27.1	4991.4
15_3_7	15	3.9	31.9	32.3	251.1	[illegible]	25.3	170.4	133.2	26.3	3713.9
15_3_10	15	9.4	59.5	50.4	[illegible]	51210.2	36.7	322.5	113.1	58.4	4320.2
15_4_1	15	4.2	34.6	36.4	[illegible]	41926.8	17.2	194.8	109.8	21.7	4872.4
15_4_3	15	4.8	173	[illegible]	416.9	36563.4	60.8	252.1	297.8	46.6	4923.4
15_4_5	15	6.8	32.7	[illegible]	313.7	38568.7	35.1	215.9	148.1	25.5	5521.3
15_4_7	15	8.5	44.8	72.9	590.5	49834.9	64.9	217.4	197.8	74.6	8074.5
15_4_10	15	7.2	[illegible]	85.2	480.8	42260.2	50.7	195.6	131.1	54.3	9871.5
30_5_1	30	15.3	48.7	37	327.1	43671.1	23	465.3	283.2	62.4	6142.5
30_5_3	30	8.4	53.5	60.5	334.1	40366.5	75.6	515.5	339.4	50	5234.72
30_5_5	30	4.6	44.9	58.6	332.2	42725.6	58.7	475.4	193.3	56.4	7182.7
30_5_7	30	5.6	34.3	51.4	306.3	36978.6	53.4	351.3	175.8	60.3	4710.3
30_5_10	30	9.1	43.1	34.9	289.8	41883.4	60.4	390.1	99.4	57.5	5643.8
30_6_1	30	9.7	35.2	40	252.5	33493.3	57.7	304.8	268.2	45.8	5192.7
30_6_3	30	11.2	87	62.6	382.2	40897.9	59.9	357.47	300.3	48.4	8674.1
30_6_5	30	6.9	40.1	30.3	335.2	49368.1	56.3	347.6	140	62.5	7979.3
30_6_7	30	3.2	36.1	41	327.9	42930.5	48.2	380.7	102.7	47.9	6725.4

Data Frames Operation

- Download the “cow_diet.csv” example data set from the MetaboAnalyst site and save it to your current folder
  - http://www.metaboanalyst.ca/resources/data/cow_diet.csv

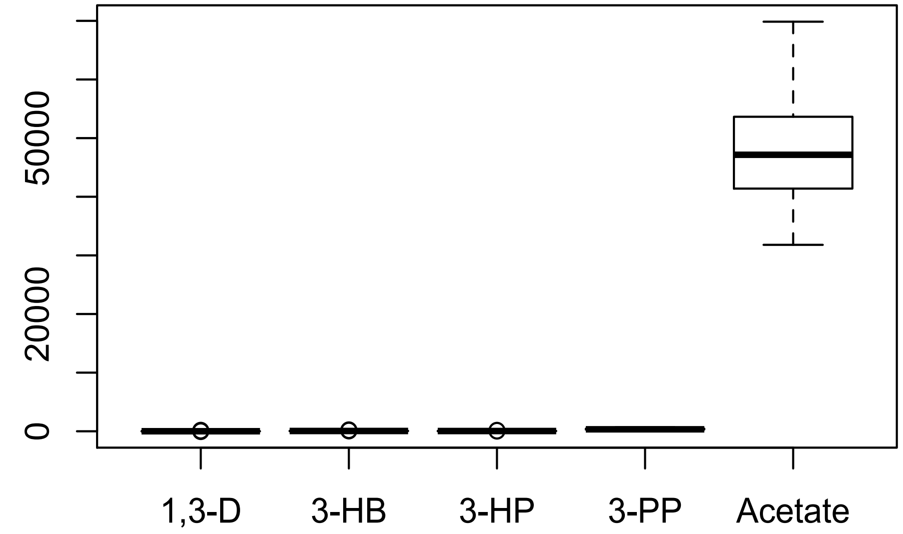
```
> cow.dat <- read.csv(”cow_diet.csv", header=T, row.names=1,
check.names=F, as.is=T)
> class(cow.dat)
[1] "data.frame"
> cow.dat[1:3, 1:4]
        Diet 1,3-D 3-HB 3-HP
0_1_1    0   2.5  0.0 11.2
0_1_3    0   2.3 47.7 35.6
0_1_5    0   3.3 34.2 18.8
```

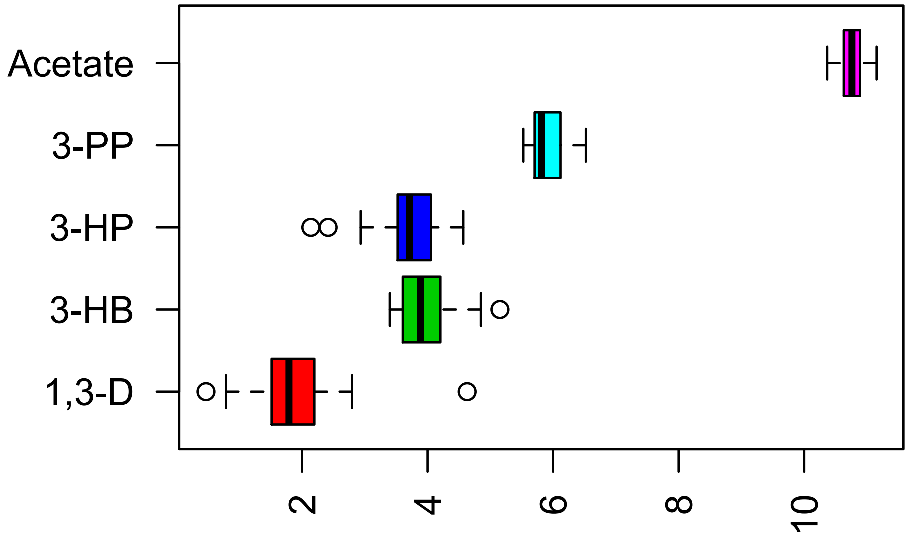
R Graphics

Box plots

- Visualize distributions of multiple variables

```
> dim(cow.dat)# get the row and column numbers of the data
[1] 39 48
> boxplot(cow.dat[,2:6])
# set the labels orthognal to axis (las=2)
> boxplot(log(cow.dat[,2:6]), las=2, horizontal=T, col=2:6)
```





Question #1

- When you take log on the data, you see the warning message:

```
> boxplot(log(cow.dat[,2:6]), las=2, horizontal=T, col=2:6)
Warning message:
In bplt(at[i], wid = width[i], stats = z$stats[, i], out = z$out[z$group ==  :
  Outlier (-Inf) in boxplot 2 is not drawn
> log(0)
[1] -Inf
```

- How to get rid of the warning?

Solution to Q1

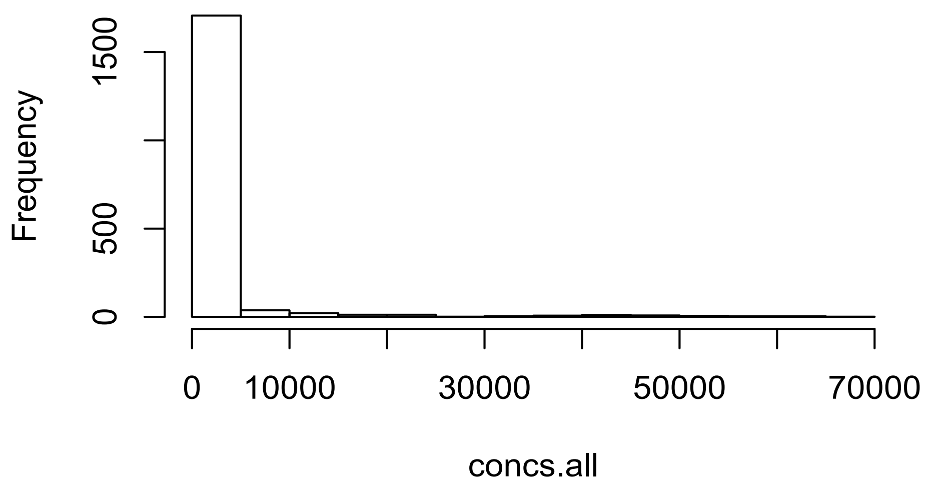
```
# make a copy of the original data, exclude group labels (1st col)
> conc.dat <- cow.dat[,-1]
# get indices for all positive values
> pos.inx <- conc.dat > 0
# get minimal concentration
> min(conc.dat[pos.inx])
[1] 1.6
# replace zero/negative with 1/5 of minimal concentration
> conc.dat[!pos.inx] <- 1.6/5;
> boxplot(log(conc.dat[,1:5]), las=2, horizontal=T, col=2:6)
```

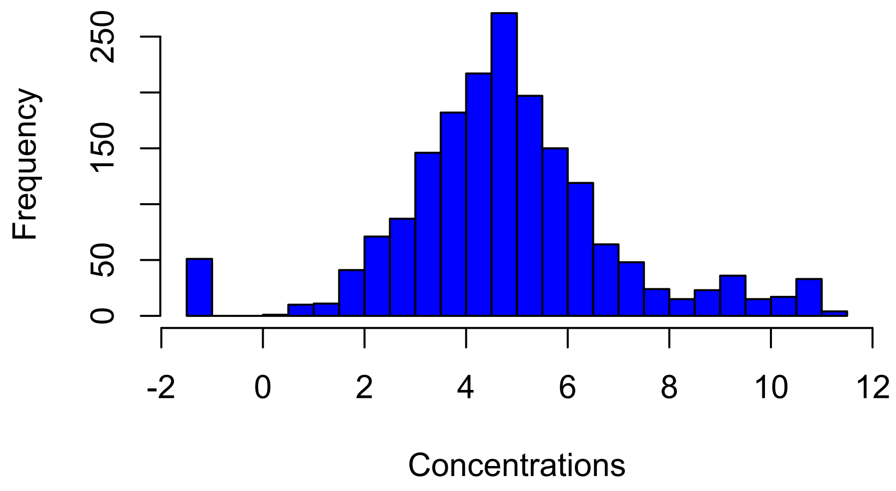
Note, *pos.inx* is a matrix of Boolean values with TRUE for positive values. Apply NOT operator (**!**) will obtain a matrix with TRUE for non-positive values

Histogram

- Visualize the distributions of numerical values

```
# convert to a numerical vector
> concs.all <- as.numeric(as.matrix(conc.dat));
> dim(concs.all)
NULL
> hist(concs.all)
> hist(concs.all, xlab="Concentrations”, main=“”)
> hist(log(concs.all), xlab="Concentrations", col="blue",
breaks=30, main=””)
```



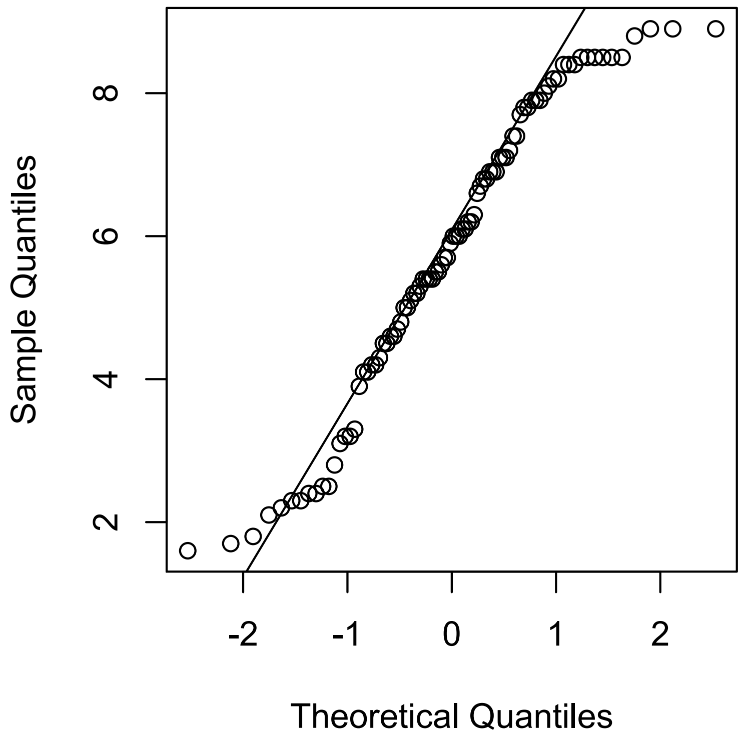


Question #2

- Based on the histogram, the log transformed data between 1.5 to 8.5 seems to be normally distributed, how do you verify that?

Solution to Q2

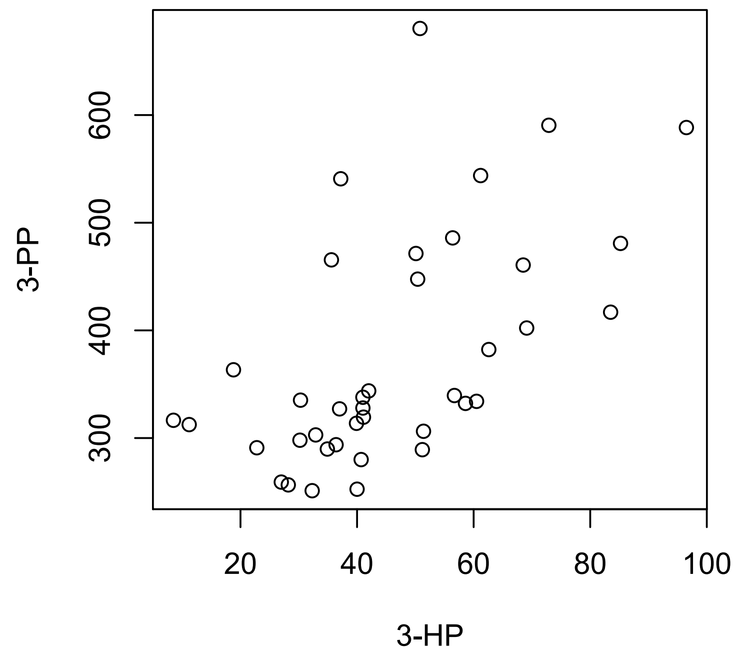
```
> norm.inx <- concs.all > 1.5 & concs.all < 8.5
> norm.dat <- concs.all[norm.inx];
> qqnorm(norm.dat)
> qqline(norm.dat);
```

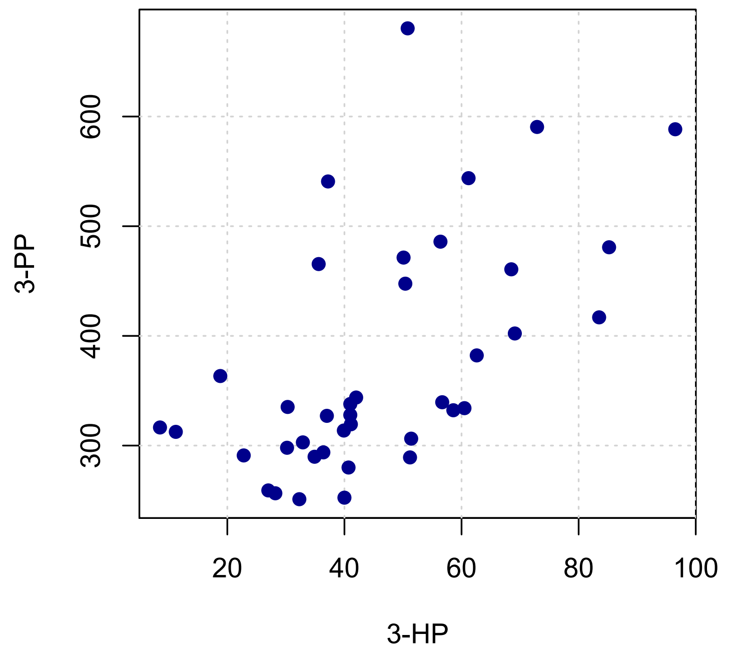


Scatter plot

- Visualize the correlations between two variables

```
> plot(cow.dat[,4:5])
> plot(cow.dat[,4:5], type='n')
> grid()
> points(cow.dat[,4:5], pch=19, col='darkblue')
```





Functions & Scripts

Built-in Functions

- Functions are a set of commands that work together to perform a given task
- Arguments are parameters you provide to the function for processing

```
> sum(c(1,2,3))
[1] 6
> mean(c(1,2,3,4,5))
[1] 3
# get average concentraion of each compound
> apply(conc.dat, 2, mean)
# get total concentration values of each sample
> apply(conc.dat, 1, sum)
```

Writing Functions: *function_name (args)*

- Functions are a set of commands that work together to perform a given task
- Let's convert our previous solution to a function

```
# Replace zero and negative values with 1/5 of min positive values
# input: a numerical matrix containing zero or negative values
# output: matrix with all positive values
ReplaceNonpositives <- function(data) {
        pos.inx <- data > 0;
        min.val <- min(data[pos.inx]);
        data[!pos.inx] <- min.val/5;
        return(data);
}
```

Write R scripts

An R script is just a plain text file with R commands & functions in it. You can prepare a script in any text editor.

1. Save the function in a text file named "ReplaceNonpositives.R" under current directory
   - You can choose other informative names
   - Multiple functions can be saved in the same file
2. To use the function:

```
> source("ReplaceNonpositives.R")
> conc.dat2 <- ReplaceNonpositives(conc.dat);
```

Libraries & Packages

- One important reason R has become so popular is the vast array of packages available at the **CRAN** and **Bioconductor** repositories
  - Install & load a CRAN package

```
> install.packages("ggplot2")
> library(ggplot2)
```

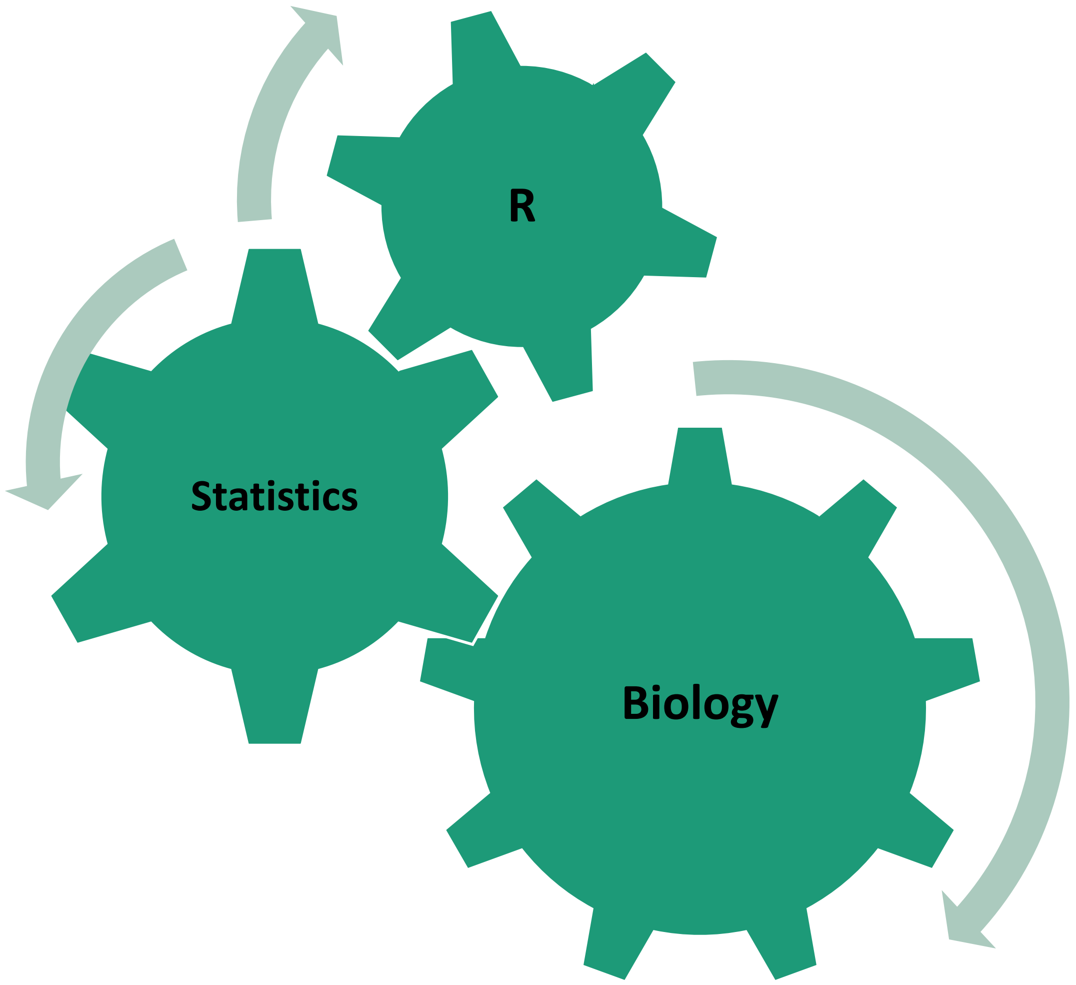
  - Install a Bioconductor package

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

Finding help

- Google your questions!
- R website http://www.r-project.org
    - Documentation
    - Mailing-list R-help
- Useful links
    - http://www.r-bloggers.com
    - http://stackoverflow.com/
    - http://www.biostars.org

Statistical Bioinformatics (BINF 531)



Sizes are proportional to the actual difficulties

Any questions?