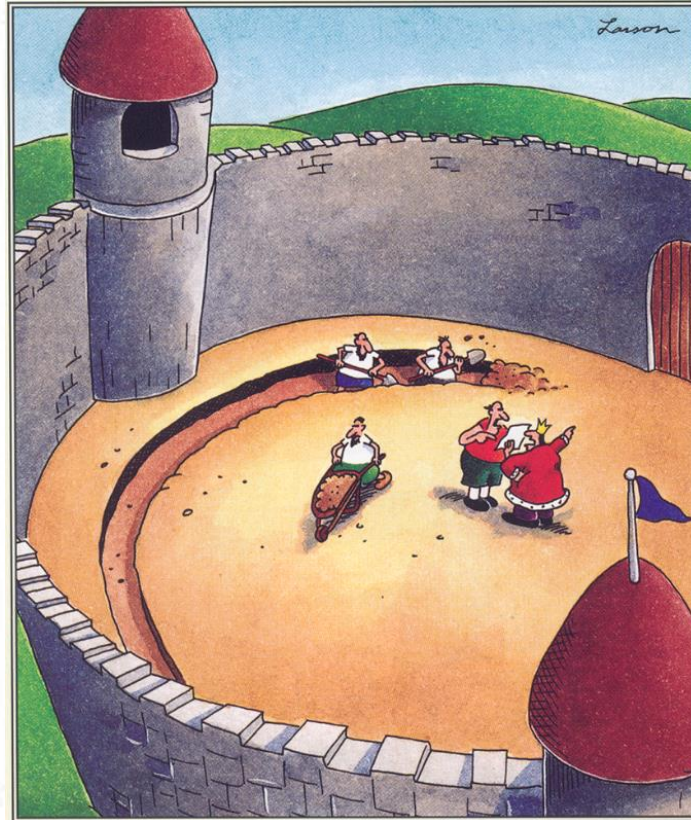


## THE FAR SIDE

By GARY LARSON



Suddenly, a heated exchange took place between the king and the moat contractor.

# DCMB BioComputing BootCamp

## Day 3, Lecture 1:

# Introduction to R

Armand Bankhead

bankhead@umich.edu

8/23/2017



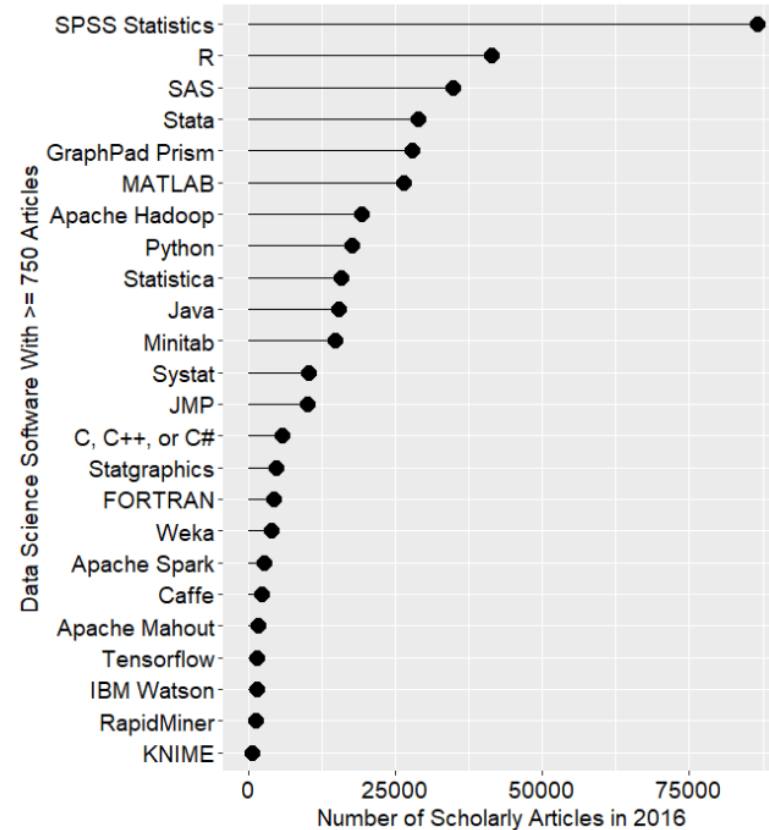
Slides Partially Sourced from Barry Grant and Hui Jiang

# Overview

1. Why R?
2. Ways to Use R
3. R as a Statistical Programming Language
4. Writing and Running R Scripts
5. Data Types
6. Data Structures
7. Vector and Matrix Operations

# Why R?

- Popular in the scientific community
- Designed to handle large datasets
- CRAN and Bioconductor open source package repositories
- Easy to automate and work with interactively
- R is a statistical computing language
- R is a concise, powerful language—much can be accomplished with few lines of code
- R can generate stunning graphics
- Well evolved function arguments (if you want to do it you likely can)
- Easy to enable others to reproduce your results
  - Good science should be reproducible!
- It's free!



# Ways to Use R

1. R interactive session: submit R functions directly to R using an R prompt
2. R Studio: integrated development environment for R
3. R command line: execute R scripts from the command line

# Ways to Use R:

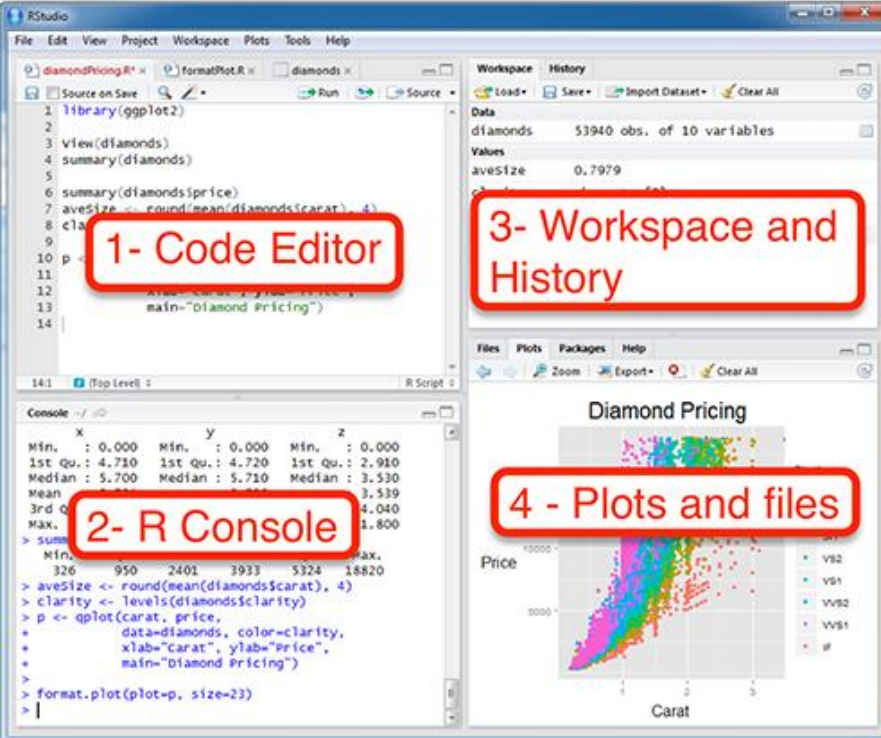
## R Interactive Session

- At the unix command line prompt type R
- Any R command should be executable from the command prompt
- Useful as an interactive environment for convenience or fast iterative development

```
bankhead@topbfx:~  
[bankhead@topbfx ~]$ R  
  
R version 3.3.3 (2017-03-06) -- "Another Canoe"  
Copyright (C) 2017 The R Foundation for Statistical Computing  
Platform: x86_64-redhat-linux-gnu (64-bit)  
  
R is free software and comes with ABSOLUTELY NO WARRANTY.  
You are welcome to redistribute it under certain conditions.  
Type 'license()' or 'licence()' for distribution details.  
  
Natural language support but running in an English locale  
  
R is a collaborative project with many contributors.  
Type 'contributors()' for more information and  
'citation()' on how to cite R or R packages in publications.  
  
Type 'demo()' for some demos, 'help()' for on-line help, or  
'help.start()' for an HTML browser interface to help.  
Type 'q()' to quit R.  
  
> print('hello world!')  
[1] "hello world!"  
> 2+2  
[1] 4  
>   
█
```

# Ways to Use R: R Studio

- Integrated development environment (IDE) provides graphical buttons
- Code completion
- Interactive debugger to debug code
- Improved window organization
- Visible history



The screenshot displays the RStudio interface with four key components highlighted by red boxes:

- 1- Code Editor:** The top-left pane shows R code for loading data, summarizing it, and creating a plot.
- 2- R Console:** The bottom-left pane shows the output of the code, including summary statistics for the 'diamonds' dataset and the execution of the plot command.
- 3- Workspace and History:** The top-right pane shows the current workspace containing the 'diamonds' dataset (53940 observations) and the 'aveSize' variable.
- 4- Plots and files:** The bottom-right pane displays a scatter plot titled 'Diamond Pricing' showing the relationship between 'Carat' (x-axis) and 'Price' (y-axis), with points colored by clarity.

# Ways to Use R:

## R Command Line

- R scripts can be executed from the command line using the Rscript command or R CMD BATCH
- Ideal for automating long running scripts and breaking long scripts up into modular pieces
  - screen and nohup

```
bankhead@topbfx:/topBfx/bankhead/projects/dcmb/bootCamp20170821
[bankhead@topbfx bootCamp20170821]$ cat helloworld.R
print('hello world!')
2 + 2
[bankhead@topbfx bootCamp20170821]$ Rscript helloworld.R
[1] "hello world!"
[1] 4
[bankhead@topbfx bootCamp20170821]$ █
```



# R as a Statistical Programming Language

R can be used to directly call statistical functions directly from an interactive session

- log2
- sqrt
- log10
- rnorm
- var
- mean
- mean
- min
- max

# Some Simple R Commands

## R Prompt!

```
> 2 + 2  
[1] 4
```

Result of the command

```
> 3^2
```

```
[1] 9
```

```
> sqrt(25)
```

```
[1] 5
```

```
> 2*(1 + 1)
```

```
[1] 4
```

```
> 2*1 + 1
```

Order of operator precedence

```
[1] 3
```

```
> exp(1)
```

```
[1] 2.718282
```

```
> log(2.718282)
```

```
[1] 1
```

Optional argument

```
> log(10, base = 10)
```

```
[1] 1
```

```
> log(10
```

Incomplete command

```
+ , base = 10)
```

```
[1] 1
```

```
> x = 1:50
```

```
> plot(x, sin(x))
```

Exercise: Try these commands

# Error Messages

- **Sometimes the commands you enter will generate errors. Common beginner examples include:**

- Incomplete brackets or quotes *e.g.*

```
> ((4+8)*20      <enter>
```

```
+
```

This returns a + here, which means you need to enter the remaining bracket - R is waiting for you to finish your input.

Press <ESC> to abandon this line if you don't want to fix it.

- Not separating arguments by commas *e.g.*

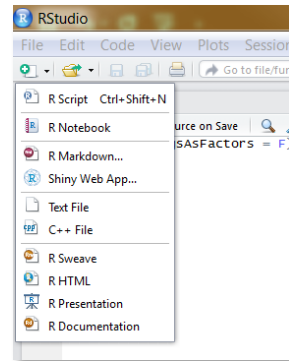
```
> plot(1:10 col="red")
```

- Typos including miss-spelling functions and using wrong type of brackets *e.g.*

```
> exp{4}
```

# Writing and Running R Scripts

- R scripts are simple text files that contain a series of R commands
- R Scripts allow us to:
  1. Execute a series of R commands
  2. Provide a written history of our work so that it is reproducible
  3. Share our work with others so that it is reproducible
- Good science is by definition reproducible!
- Comment your code! Use '#' to include comments
- How to create an R Script:
  1. RStudio Select New Icon or type Ctrl-Shift-N
  2. Edit a text file with your favorite text editor



# Writing and Running R Scripts

- R scripts can be run in several different ways:
  1. Using the source function in an interactive session  
> source('helloWorld.R')
  2. In RStudio use the ctrl-alt-r shortcut
  3. In Rstudio select Code -> Run Region -> Run All
  4. From the unix command line use Rscript command  
> Rscript helloWorld.R

Exercise: Write a helloWorld.R script and execute it using one of these approaches.

# 5 Basic Data Types in R

1. numeric: decimal values, default

```
> x = 10.5
```

```
> class(x)
```

```
[1] "numeric"
```

2. integer: integer values

```
> y = as.integer(3)
```

```
> class(y)
```

```
[1] "integer"
```

3. complex: imaginary numbers

```
> z = sqrt(as.complex(-1))
```

```
> z
```

```
[1] 0+1i
```

```
> class(z)
```

```
[1] "complex"
```

# 5 Basic Data Types in R

4. logical: TRUE or FALSE values. Also T or F

```
> x = 1
```

```
> y = 2
```

```
> z = x > y
```

```
> z
```

```
class(z)
```

```
[1] "logical"
```

5. character: a sequence of ascii character values. Characters may be surrounded by single or double quotes!

```
> x = "Joe"
```

```
> class(x)
```

```
[1] "character"
```

# R Data Structures

| Dimension | Homogeneous Data | Heterogeneous Data |
|-----------|------------------|--------------------|
| 1         | Vector           | List               |
| 2         | Matrix           | Data Frame         |
| N         | Array            | List               |



# Data Structures: Vector

- Vectors contain some number of values of the same type.
- Vectors may be created using the combine 'c' function
- Examples:
  - > days = c('mon','tues','wed','thurs','fri')
  - > myNumbers = c(1.5,3,4.5,6,7.5)
- Vectors may also be created as sequences using the ':' operator
- Examples:
  - > 1:5
  - [1] 1 2 3 4 5

# Data Structures: Vector

```
> days = c('mon','tues','wed','thurs','fri')
```

```
> myNumbers = c(1.5,3,4.5,6,7.5)
```

- Vectors can be indexed using square brackets

```
> favoriteDay = days[5]
```

```
> favoriteDays = days[2:5]
```

- Negative indexes return all but the value subtracted!

```
> favoriteDays = days[-1]
```

```
> favoriteDays
```

```
[1] "tues" "wed" "thurs" "fri"
```

# Data Structures: Vector

```
> days = c('mon','tues','wed','thurs','fri')
```

```
> myNumbers = c(1.5,3,4.5,6,7.5)
```

- Additional vector operations:

```
> sort(myNumbers)
```

```
[1] 1.5, 3, 4.5, 6, 7.5
```

```
> sort(myNumbers,decreasing=TRUE)
```

```
[1] 7.5, 6, 4.5, 3, 1.5
```

```
> rev(days)
```

```
[1] "fri" "thurs" "wed" "tues" "mon"
```

```
> length(days)
```

```
[1] 5
```

# Data Structures: Matrix

- Matrices are two dimensional data tables that contain the same data types
- A data matrix may be created several ways:
  - > `m1 = matrix(1,nrow=2,ncol=2)`
  - > `m2 = matrix(1:4,nrow=2,ncol=2)`
  - > `m3 = rbind(c(1,2),c(3,4))`
- Matrices may have row and column names
  - > `colnames(m2) = c('A','B')`
  - > `rownames(m2) = c('POS','NEG')`
  - > `m2`

|     | A | B |
|-----|---|---|
| POS | 1 | 3 |
| NEG | 2 | 4 |

# Data Structures: Matrix

```
> m4 = matrix(1:300,nrow=100,ncol=3)
```

```
> colnames(m4) = c('A','B','C')
```

```
> dim(m4)
```

```
100 3
```

- We can access one or more values of a matrix by specifying row and column values

```
> m4[1,2]
```

```
[1] 101
```

```
> m4[1:2,1:2]
```

```
  [,1] [,2]
```

```
[1,] 1  101
```

```
[2,] 2  102
```

- R has `head()` and `tail()` commands like unix!

**Exercise: Construct m4 and use `head()` and `tail()`**

# Vector and Matrix Operations

- R has a rich set of vector and matrix operators

```
> v1 = c(1,2,3)
```

```
> m1 = matrix(1:4,nrow=2,ncol=2)
```

- Simple math operations are applied to all values

```
> v1 * 2
```

```
[1] 2,4,6
```

- Standard functions are applied to each value

```
> log2(v1)
```

```
[1] 0.000000 1.000000 1.584963
```

- Linear algebra transformations are well supported
  - `t(m1)` will return the transpose of `m1`
  - `m1 * m1` will perform element-wise multiplication
  - `m1 %**% m1` will perform matrix multiplication

**Exercise: Use `m1 * m1` and `m1 %**% m1`. How do the answers differ?**

# Special Values in R

- NA values are not available or missing values
  - Often functions will specify how to treat NA values
  - `is.na()` will return TRUE/FALSE
- NaN values are not a number
  - `is.nan()` will return TRUE/FALSE
- Inf and -Inf values are computationally too large or too small
  - `is.infinite()` will return TRUE/FALSE

```
> 2 ^ 1024
```

```
[1] Inf
```

- NULL values are empty and often used to represent zero-length objects
  - `is.null()` will return TRUE/FALSE

```
> dim(c(1,2,3))
```

```
NULL
```

Exercise: Calculate  $1/0$  in R. Calculate  $\log_2(-1)$ . Calculate  $\log_2(0)$ . What does R return?

# Data Structures: List

- A list is a “generic vector” that may contain a variety of data types and data structures

```
> y = list(1, 17, 4:5, “a”)
```

- List values may be named

```
> y = list(a = 1, 17, b = 4:5, c = “a”)
```

```
$a
```

```
[1] 1
```

```
[[2]]
```

```
[1] 17
```

```
$b
```

```
[1] 4 5
```

```
$c
```

```
[1] “a”
```



# Data Structures: List

```
> y = list(a = 1, 17, b = 4:5, c = "a")
```

- A list value may be accessed using a single index

```
> y[[3]]
```

```
[1] 4 5
```

- List values may be accessed using names

```
> y$b
```

```
[1] 4 5
```

- Multiple list values may be accessed using an index but the result will be a list

```
> y[1:3]
```

```
$a
```

```
[1] 1
```

```
[[2]]
```

```
[1] 17
```

```
$b
```

```
[1] 4 5
```

# Data Structures: Data Frame

- A data frame is a special kind of list containing multiple vectors of the same length
  - Vectors may contain multiple data types
- This data structure is commonly used when reading, writing data
- Data frames may be created using the `data.frame` function:

```
> days = c('mon','tues','wed')
```

```
> myNumbers = c(1,2,3)
```

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

```
> df1 = data.frame(days,myNumbers,attend)
```

```
> df1
  days myNumbers attend
1 mon           1  TRUE
2 tues          2 FALSE
3 wed           3  TRUE
```

# Data Structures: Data Frame

- Data frames represent a powerful hybrid between a matrix and a list
  - We can use indexes to access specific columns
  - We can use '\$' column names to access individual vectors
- There are several data frame examples built into R

Exercise: Type `mtcars` at your R prompt. What columns does this data frame contain? What is the average mpg of all cars? (hint: use the `mean()` function)

# Help from within R

- Getting help for a function

```
> help("log")
```

```
> ?log
```

- Searching across packages

```
> help.search("logarithm")
```

- Finding all functions of a particular type

```
> apropos("log")
```

```
[7] "SSlogis" "as.data.frame.logical" "as.logical"  
     "as.logical.factor" "dlogis" "is.logical"
```

```
[13] "log" "log10" "log1p" "log2" "logLik" "logb"
```

```
[19] "logical" "loglin" "plogis" "print.logLik" "qlogis"  
     "rlogis"
```

# ?log

R: Logarithms and Exponentials ▾

Find in Topic

log (base)

R Documentation

## Logarithms and Exponentials

### Description What the function does in general terms

`log` computes logarithms, by default natural logarithms, `log10` computes common (i.e., base 10) logarithms, and `log2` computes binary (i.e., base 2) logarithms. The general form `log(x, base)` computes logarithms with base `base`.

`log1p(x)` computes  $\log(1+x)$  accurately also for  $|x| \ll 1$  (and less accurately when  $x$  is approximately -1).

`exp` computes the exponential function.

`expm1(x)` computes  $\exp(x) - 1$  accurately also for  $|x| \ll 1$ .

### Usage How to use the function

```
log(x, base = exp(1))
logb(x, base = exp(1))
log10(x)
log2(x)
```

```
log1p(x)
```

```
exp(x)
expm1(x)
```

### Arguments What does the function need

`x` a numeric or complex vector.  
`base` a positive or complex number: the base with respect to which logarithms are computed. Defaults to  $e = \exp(1)$ .

### Details

All except `logb` are generic functions: methods can be defined for them individually or via the [Math](#) group generic.

`log10` and `log2` are only convenience wrappers, but logs to bases 10 and 2 (whether computed via `log` or the wrappers) will be computed more efficiently and accurately where supported by the OS. Methods can be set for them individually (and otherwise methods for `log` will be used).

`logb` is a wrapper for `log` for compatibility with S. If (S3 or S4) methods are set for `log` they will be dispatched. Do not set S4 methods on `logb` itself.

All except `log` are [primitive](#) functions.

R: Logarithms and Exponentials ▾

Find in Topic

### Value What does the function return

A vector of the same length as `x` containing the transformed values. `log(0)` gives `-Inf`, and `log(x)` for negative values of `x` is `NaN`. `exp(-Inf)` is 0.

For complex inputs to the log functions, the value is a complex number with imaginary part in the range  $[-\pi, \pi]$ : which end of the range is used might be platform-specific.

### S4 methods

`exp`, `expm1`, `log`, `log10`, `log2` and `log1p` are S4 generic and are members of the [Math](#) group generic.

Note that this means that the S4 generic for `log` has a signature with only one argument, `x`, but that `base` can be passed to methods (but will not be used for method selection). On the other hand, if you only set a method for the `Math` group generic then `base` argument of `log` will be ignored for your class.

### Source

`log1p` and `expm1` may be taken from the operating system, but if not available there are based on the Fortran subroutine `dlnrel` by W. Fullerton of Los Alamos Scientific Laboratory (see <http://www.netlib.org/slatec/flnlib/dlnrel.f> and (for small `x`) a single Newton step for the solution of  $\log1p(y) = x$  respectively.

### References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole. (for `log`, `log10` and `exp`.)

Chambers, J. M. (1998) *Programming with Data. A Guide to the S Language*. Springer. (for `logb`.)

### See Also Discover other related functions

[Trig](#), [sqrt](#), [Arithmetic](#).

### Examples Sample code showing how it works

```
log(exp(3))
log10(1e7) # = 7
```

```
x <- 10^-(1+2*1:9)
cbind(x, log(1+x), log1p(x), exp(x)-1, expm1(x))
```

[Package `base` version 3.0.1 [Index](#)]

# Additional Resources

**TryR.** An excellent interactive online R tutorial for beginners.

< <http://tryr.codeschool.com/> >

**DataCamp.** Online tutorials using R in your browser.

< <https://www.datacamp.com/> >

**R for Data Science.** A new O'Reilly book that will teach you how to do data science with R, by Garrett Grolemund and Hadley Wickham.

< <http://r4ds.had.co.nz/> >

# References

- Gentleman, Robert. R Programming for Bioinformatics. CRC Press, 2009.