Rbootcamp

Michael Kleinsasser

8/8/2019

Mike's Personal Introduction

- R programmer for the Department of Biostatistics
- Write and mantain R packages for faculty and students
- Consult faculty and students on writing R packages, optimization

Rbootcamp Introduction

Goals:

- R basics: syntax, common functions, etc., via Rstudio
- R functions and for loops (functions and advanced control structures)
- Basics of R on the cluster (non-interactive R using BATCH scripts)
- Simulation analysis with comment on efficiency

Materials

- All bootcamp materials online at https://github.com/umich-biostatistcs/Rbootcamp
  - Handouts for each topic with examples to work through
  - R scripts of our examples
- Go to link and download zip archive, extract

Setup

Go to Rstudio cloud to follow along:

- Enter username, etc. for free account
- Follow along by typing commands in my slides
- If you have Rstudio/R, open that
- Recommended: Install R/Rstudio for days 2, 3
- See course materials for download instructions

R Basics: Big Picture

- R is a sophisticated calculator for statistics

Chambers (2016) Extending R:

- Everything that exists in R is an object
- Everything that happens in R is a function call

Obtain a basic working knowledge of R objects and functions,

- Google the rest

R Basics: a basic schematic view

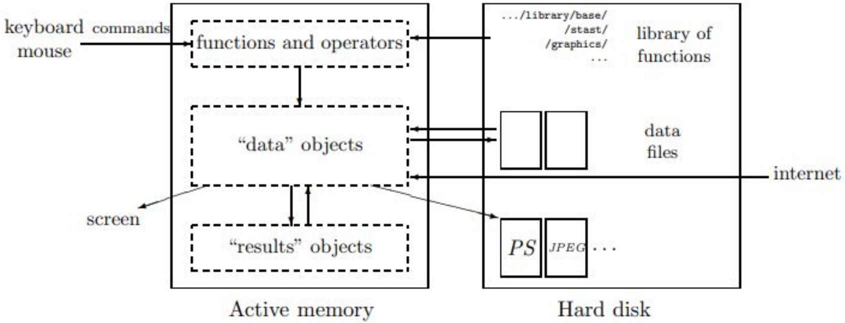


Figure 1: A schematic view of how R works.

R Basics: Goals

- Data types and functions
    - Create object having data types
    - Combine those into data structures
    - Write basic R function
- Learn parts of R most useful to statisticians
    - How do most modeling functions work in R, and
    - How to inspect structure and content of objects
- Apply knowledge to simulation
    - How do most modeling functions work in R, and
    - How to inspect structure and content of objects

Use R as a calculator

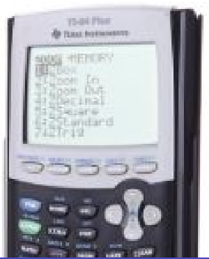
Standard operations:

```
# multiply *, divide /, add + subtract -
18056.983 - 1005.118 + 22.53
```

```
## [1] 17074.4
```

```
( (pi - 3.14) / (3.14) )  * 100
```

```
## [1] 0.05072145
```



Operators: arithmetic and logical

Operator	Description
+	addition
−	subtraction
/	division
^	exponential
%%	modulus (x mod y)
%/%	integer division

Table 1: Arithmetic Operators

Operator	Description
<	less than
<=	less than or equal
>	greater than
>=	greater than or equal
==	exactly equal to
!=	not equal to
!x	not x (x logical)
x|y	x OR y
x&y	x AND y
isTRUE(x)	is x TRUE

Table 2: Logical Operators

Operators: an example

Test if these two expressions are equivalent in R:

Operators: an example

Test if these two expressions are equivalent in R:

Convert the following expression to R code:

Create, list, delete objects

Create object with "assign" operator

- arrow then minus sign <-
- single equal sign =

```
# x gets the number 3.14
x <- 3.14
x      # print x
```

```
## [1] 3.14
```

```
# equivalently
x = 3.14
x      # print x
```

```
## [1] 3.14
```

Create, list, delete objects

Objects we create are stored in memory, e.g.:

```r
name = "Carmen"
n1 = 10
n2 = 100
m = 0.5
```

Use ls() function to list all objects in memory:

```r
ls()
```

```
## [1] "m"      "n1"     "n2"     "name" "x"
```

Notice: I created x before, it's still in memory.

Create, list, delete objects

Use ls() function to list all objects in memory:

```r
ls()
```

```
## [1] "m"    "n1"   "n2"   "name" "x"
```

The function ls.str() displays some details about objects in memory:

```r
ls.str()
```

```
## m :  num 0.5
## n1 :  num 10
## n2 :  num 100
## name :  chr "Carmen"
## x :  num 3.14
```

Create, list, delete objects

To delete objects in memory, use rm() function

```
rm(x) # delets object named x
ls()  # which objects remain in memory?
```

```
## [1] "m"    "n1"   "n2"   "name"
```

```
rm(m, n2, name) # remove multiple objects
ls()
```

```
## [1] "n1"
```

```
rm(list = ls()) # remove everything from memory
ls()
```

```
## character(0)
```

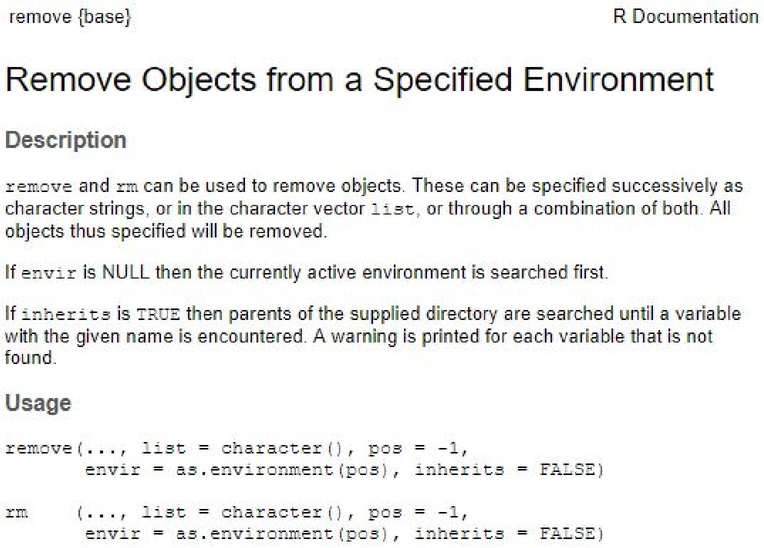
The on-line help

R has structured help pages providing “how-to”

- **Description:** what function does
- **Usage:** name with arguments and options
- **Arguments:** how each argument should be structured
- **Details:** more detailed description
- **Value:** How the output is structured/ what it contains
- **Examples:** examples of the function in use

```
?rm             # help documentation for rm function
help("rm")      # alternately
```

The on-line help



remove {base} R Documentation

Remove Objects from a Specified Environment

Description

`remove` and `rm` can be used to remove objects. These can be specified successively as character strings, or in the character vector `list`, or through a combination of both. All objects thus specified will be removed.

If `envir` is NULL then the currently active environment is searched first.

If `inherits` is `TRUE` then parents of the supplied directory are searched until a variable with the given name is encountered. A warning is printed for each variable that is not found.

Usage

```
remove(..., list = character(), pos = -1,
       envir = as.environment(pos), inherits = FALSE)

rm    (..., list = character(), pos = -1,
       envir = as.environment(pos), inherits = FALSE)
```

Other R help

Many package writers create Vignettes and READMEs

How to view vignettes?

```r
vignette(all = TRUE) # list vignettes for installed packages
vignette(all = FALSE)  # vignettes from attached packages
```

How to view READMEs?

Other help. . .

Objects in R

Objects (data, model output, functions)

- Characterized by their **names** and **content**
- attributes - specify the kind of data represented
  - e.g. mode, length

```
x = 1
mode(x)
```

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

```
length(x)
```

```
## [1] 1
```

Basic data "modes" in R

The mode is the basic type of the elements of an object

The four main modes:

- numeric, comes in two flavors: integer, numeric
- character
- logical
- complex

```r
num = 15.533; name = "Mike"; isStudent = TRUE
```

```r
mode(num); mode(name); mode(isStudent)
```

```
## [1] "numeric"

## [1] "character"

## [1] "logical"
```

Atomic vectors

Fundamental data structure in R:

- atomic vector - vector in which every element is of same mode

To create an atomic vector, use c() function:

```
c(3.145, 2.18, 9.98e3, 0.05)
```

```
## [1]    3.145    2.180 9980.000    0.050
```

Example: create empty character vector of length 3 and story your full name

```
vector(mode = "character", length = 3)
```

```
## [1] "" "" ""
```

Data types examples

3 ways to create numeric vector:

```r
# empty numeric vector
y1 <- numeric(6)
y1      # print y1
```

```
## [1] 0 0 0 0 0 0
```

```r
y2 <- vector(mode = "numeric", length = 6)
y2      # print y2
```

```
## [1] 0 0 0 0 0 0
```

```r
y3 <- c(5, 13.222, 2, 0.001, 77.4, 31.9)
y3      # print y3
```

```
## [1]  5.000 13.222  2.000  0.001 77.400 31.900
```

Objects in R: NA

NA means “Not Available” and it denotes missing data

(Insert after atomic vectors introduced)

```
c(3, 5, 9, NA, 18, 25, NA)
```

```
## [1]  3  5  9 NA 18 25 NA
```

R data types/structures

Four fundamental data types:

- character, numeric (numeric or integer), logical, complex

Combine to form data structures

- atomic vector (atomic - vector of single type)
- list
- matrix
- data.frame
- factor

We will focus on matrices and data.frames

Matrices

Matrices are the natural extension of atomic vectors into 2 dimensions

- any mode can be used, but numeric most common:

Syntax:

```
m = matrix(data, nrow, ncol, byrow, dimnames)
```

- **data** is the input vector which becomes the data elements of the matrix
- **nrow, ncol** is the number of rows/columns to be created
- **byrow** is a logical clue. If TRUE then the input vector elements are arranged by row
- **dimnames** is the names assigned to the rows and columns

Matrix examples:

Identity matrix:

```
dat = c(1,0,0,0,1,0,0,0,1)  # data
iden = matrix(data = dat, nrow = 3, byrow = T)
iden  # print matrix
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    1    0
## [3,]    0    0    1
```

Easier:

```
iden = diag(rep(1,3))
iden
```

access elements of a matrix

- single brackets used to access elements

Access individual elements:

```
# 2x5 matrix of numbers 1 to 10
P = matrix(data = 1:10, nrow = 2)
P[1,3] # row 1, column 3
```

```
## [1] 5
```

```
P[nrow(P),ncol(P)] # row 2, column 5 (bottom right position)
```

```
## [1] 10
```

Access entire rows/columns:

```
P[,3];
```

```
## [1] 5 6
```

The data.frame

The **data.frame** is the most common way to store and work with data in R

- Not surprising: they are designed for this purpose
- Most modeling functions work on data.frames

Composed of a list of equal length atomic vectors (can be of any type)

Example:

The following are data on students in the class:

- Has Master's (logical): TRUE FALSE FALSE TRUE
- GPA (numeric): 3.1 4.0 2.9 3.6
- First Name (character): Mike Dan Sara Karen

Convert to three atomic vectors of appropriate type.

data.frame examples

Create a data.frame out of the following "class" data:

- Has Master's (logical): TRUE FALSE FALSE TRUE
- GPA (numeric): 3.1 4.0 2.9 3.6
- First Name (character): Mike Dan Sara Karen

```r
# store data
has_ms <- c(TRUE, FALSE, FALSE, TRUE)
gpa <- c(3.1, 4.0, 2.9, 3.6)
name <- c("Mike", "Dan", "Sara", "Karen")
# Create data.frame
dat <- data.frame(has_MS = has_ms, GPA = gpa, Name = name)
dat    # print data.frame
```

```
##   has_MS GPA  Name
## 1   TRUE 3.1  Mike
## 2  FALSE 4.0   Dan
## 3  FALSE 2.9  Sara
## 4   TRUE 3.6 Karen
```

access elements of a data.frame

Each vector of a data.frame contains the values of a variable

Access each vector with the dollar sign $

Ex: Extract the GPA column and print it

```
dat$GPA
```

```
## [1] 3.1 4.0 2.9 3.6
```

Another example: ToothGrowth data.

```
ToothGrowth$len
```

```
##  [1]  4.2 11.5  7.3  5.8  6.4 10.0 11.2 11.2  5.2  7.0 16.5
## [15] 22.5 17.3 13.6 14.5 18.8 15.5 23.6 18.5 33.9 25.5 26.4
## [29] 23.3 29.5 15.2 21.5 17.6  9.7 14.5 10.0  8.2  9.4 16.5
## [43] 23.6 26.4 20.0 25.2 25.8 21.2 14.5 27.3 25.5 26.4 22.4
## [57] 26.4 27.3 29.4 23.0
```

data.frame

Preview head (first few rows) of data.frame:

```
head(ToothGrowth)
```

```
##    len supp dose
## 1  4.2   VC  0.5
## 2 11.5   VC  0.5
## 3  7.3   VC  0.5
## 4  5.8   VC  0.5
## 5  6.4   VC  0.5
## 6 10.0   VC  0.5
```

View tail of data.frame:

```
tail(dat)
```

View entire data.frame in new window:

```
View(ToothGrowth)
```

Inspect an object

- class() - what kind of object is it (high-level)?
- typeof() - what is the data type (low-level)?
- length() - how long is it?
- attributes() - does it have meta-data?

Inspect an object

- class() - what kind of object is it (high-level)?

```
class(ToothGrowth)
```

```
## [1] "data.frame"
```

- typeof() - what is the data type (low-level)?

```
typeof(ToothGrowth$supp)
```

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

Inspect an object

- length() - how long is it?

```
length(ToothGrowth$dose)
```

```
## [1] 60
```

- attributes() - does it have meta-data?

```
attributes(ToothGrowth)
```

```
## $names
## [1] "len"  "supp" "dose"
##
## $class
## [1] "data.frame"
##
## $row.names
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18
```

R functions

R function syntax:

```
NAME <- function(ARG1, ARG2, ARG3) {
    DO SOMETHING
    STORE RESULT
    return(RESULT)
}
```

```
pow <- function(base, expon) { # power function
  prod(rep(base, expon)) # base^(expon)
}
# Use power function
pow(5, 2)
```

```
## [1] 25
```

```
pow(10, 3)
```

```
## [1] 1000
```

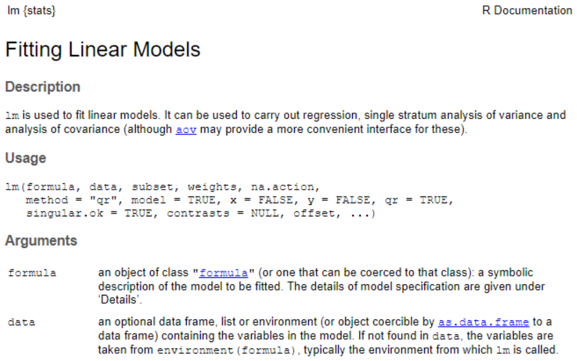
Common R functions

R has a huge collection of packages:

- 6,000+ packages for data analysis build (on CRAN alone)

Example: lm (linear models)

- Use ?lm to read help documentation



lm {stats}   R Documentation

Fitting Linear Models

Description

`lm` is used to fit linear models. It can be used to carry out regression, single stratum analysis of variance and analysis of covariance (although `aov` may provide a more convenient interface for these).

Usage

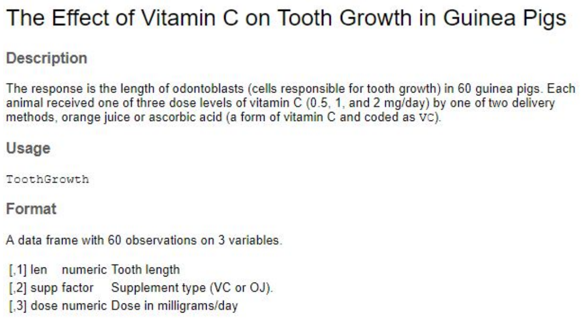
```
lm(formula, data, subset, weights, na.action,
   method = "qr", model = TRUE, x = FALSE, y = FALSE, qr = TRUE,
   singular.ok = TRUE, contrasts = NULL, offset, ...)
```

Arguments

`formula`	an object of class "`formula`" (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under 'Details'.
`data`	an optional data frame, list or environment (or object coercible by `as.data.frame` to a data frame) containing the variables in the model. If not found in `data`, the variables are taken from `environment(formula)`, typically the environment from which `lm` is called.

Fit a linear model with lm

- Use built-in data set ToothGrowth
- ?ToothGrowth for help:



The Effect of Vitamin C on Tooth Growth in Guinea Pigs

Description

The response is the length of odontoblasts (cells responsible for tooth growth) in 60 guinea pigs. Each animal received one of three dose levels of vitamin C (0.5, 1, and 2 mg/day) by one of two delivery methods, orange juice or ascorbic acid (a form of vitamin C and coded as `VC`).

Usage

```
ToothGrowth
```

Format

A data frame with 60 observations on 3 variables.

[,1]	len	numeric	Tooth length
[,2]	supp	factor	Supplement type (VC or OJ).
[,3]	dose	numeric	Dose in milligrams/day

View the data

View data in new window:

```
View(ToothGrowth)
```

Or use head to view only first 6 rows:

```
head(ToothGrowth)
```

```
##    len supp dose
## 1  4.2   VC  0.5
## 2 11.5   VC  0.5
## 3  7.3   VC  0.5
## 4  5.8   VC  0.5
## 5  6.4   VC  0.5
## 6 10.0   VC  0.5
```

How big is the data?

```
dim(ToothGrowth)
```

```
## [1] 60  3
```

Using lm() function for linear models

Call lm on the data and formula, store result “lm” object:

```r
tooth_fit = lm(formula = len ~ supp + dose,
               data = ToothGrowth)
```

Formulas in R:

```r
len ~       # Response column name, ~ for "="
  supp +    # First predictor name + for "+"
  dose      # second predictor name
```

Many R functions use the formula argument.

Getting detailed information

Basic “print” of model:

```
print(tooth_fit)     # equivalent to tooth_fit
```

```
##
## Call:
## lm(formula = len ~ supp + dose, data = ToothGrowth)
##
## Coefficients:
## (Intercept)       suppVC         dose
##       9.272       -3.700        9.764
```

Detailed summary:

```
summary(tooth_fit)
```

```
##
## Call:
## lm(formula = len ~ supp + dose, data = ToothGrowth)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -6.600 -3.700  0.373  2.116  8.800
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   9.2725     1.2824   7.231 1.31e-09 ***
## suppVC       -3.7000     1.0936  -3.383   0.0013 **
## dose          9.7636     0.8768  11.135 6.31e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

Understanding R classes

- What is this thing?

```
class(tooth_fit)
```

- What are the methods for this object?

```
methods(class = "lm")
```

- What is its structure? (i.e., what's in it)

```
str(tooth_fit)
```

Understanding R classes

- What is this thing?

```
class(tooth_fit)
```

```
## [1] "lm"
```

Understanding R classes

- What are the methods for this object?

```
methods(class = "lm")
```

```
##  [1] add1           alias          anova          case.names
##  [5] coerce         confint        cooks.distance deviance
##  [9] dfbeta         dfbetas        drop1          dummy.coef
## [13] effects        extractAIC     family         formula
## [17] hatvalues      influence      initialize     kappa
## [21] labels         logLik         model.frame    model.matrix
## [25] nobs           plot           predict        print
## [29] proj           qr             residuals      rstandard
## [33] rstudent       show           simulate       slotsFromS3
## [37] summary        variable.names vcov
## see '?methods' for accessing help and source code
```

Understanding R classes

- What is its structure? (i.e., what's in it)

```r
str(tooth_fit)
```

```
## List of 13
##  $ coefficients : Named num [1:3] 9.27 -3.7 9.76
##   ..- attr(*, "names")= chr [1:3] "(Intercept)" "suppVC" "dose"
##  $ residuals    : Named num [1:60] -6.25 1.05 -3.15 -4.65 -4.05 ...
##   ..- attr(*, "names")= chr [1:60] "1" "2" "3" "4" ...
##  $ effects      : Named num [1:60] -145.73 14.33 47.16 -3.86 -3.26 ...
##   ..- attr(*, "names")= chr [1:60] "(Intercept)" "suppVC" "dose" "" ...
##  $ rank         : int 3
##  $ fitted.values: Named num [1:60] 10.5 10.5 10.5 10.5 10.5 ...
##   ..- attr(*, "names")= chr [1:60] "1" "2" "3" "4" ...
##  $ assign       : int [1:3] 0 1 2
##  $ qr           :List of 5
##   ..$ qr   : num [1:60, 1:3] -7.746 0.129 0.129 0.129 0.129 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:60] "1" "2" "3" "4" ...
##   .. .. ..$ : chr [1:3] "(Intercept)" "suppVC" "dose"
##   .. ..- attr(*, "assign")= int [1:3] 0 1 2
##   .. ..- attr(*, "contrasts")=List of 1
##   .. .. ..$ supp: chr "contr.treatment"
##   ..$ qraux: num [1:3] 1.13 1.11 1.11
##   ..$ pivot: int [1:3] 1 2 3
##   ..$ tol  : num 1e-07
##   ..$ rank : int 3
##   ..- attr(*, "class")= chr "qr"
##  $ df.residual  : int 57
##  $ contrasts    :List of 1
##   ..$ supp: chr "contr.treatment"
##  $ xlevels      :List of 1
```

Understanding R classes

- Pull something out of the "lm" fit object:

```
tooth_fit$fitted.values        # y_hat's for the linear model
```

```
##        1        2        3        4        5        6        7        8
## 10.45429 10.45429 10.45429 10.45429 10.45429 10.45429 10.45429 10.45429
##        9       10       11       12       13       14       15       16
## 10.45429 10.45429 15.33607 15.33607 15.33607 15.33607 15.33607 15.33607
##       17       18       19       20       21       22       23       24
## 15.33607 15.33607 15.33607 15.33607 25.09964 25.09964 25.09964 25.09964
##       25       26       27       28       29       30       31       32
## 25.09964 25.09964 25.09964 25.09964 25.09964 25.09964 14.15429 14.15429
##       33       34       35       36       37       38       39       40
## 14.15429 14.15429 14.15429 14.15429 14.15429 14.15429 14.15429 14.15429
##       41       42       43       44       45       46       47       48
## 19.03607 19.03607 19.03607 19.03607 19.03607 19.03607 19.03607 19.03607
##       49       50       51       52       53       54       55       56
## 19.03607 19.03607 28.79964 28.79964 28.79964 28.79964 28.79964 28.79964
##       57       58       59       60
## 28.79964 28.79964 28.79964 28.79964
```

Extracting data from model objects

Some generic extraction methods:

```
coef(tooth_fit)             # model coefficients

coef(summary(tooth_fit))    # adds test statistics, p-values

vcov(tooth_fit)             # variance/covariance matrix
```

Note: depending on implementation, these may not be available - Check methods with “methods(object)” before attempting

Extracting data from model objects (in detail)

Extract coefficients, test stats, and p-values

```
coef(summary(tooth_fit))   # adds test statistics, p-values
```

```
##              Estimate Std. Error   t value     Pr(>|t|)
## (Intercept)  9.272500  1.2823649  7.230781 1.312335e-09
## suppVC      -3.700000  1.0936045 -3.383307 1.300662e-03
## dose         9.763571  0.8768343 11.135025 6.313519e-16
```

Predict new values

- predict() function is generic and works with many models
- Pass in a new data.frame with the same column names:

```
to_predict = data.frame(dose = 0.5, supp = "VC")
```

```
predict(tooth_fit, newdata = to_predict)
```

```
##        1
## 10.45429
```

Predict new values (example 2)

- predict() function is generic and works with many models
- Pass in a new data.frame with the same column names:

```
to_predict = data.frame(dose = seq(0,1,0.1), supp = "OJ")
```

```
predict(tooth_fit, newdata = to_predict)
```

```
##        1        2        3        4        5        6        7
##  9.27250 10.24886 11.22521 12.20157 13.17793 14.15429 15.13064 16
##        9       10       11
## 17.08336 18.05971 19.03607
```

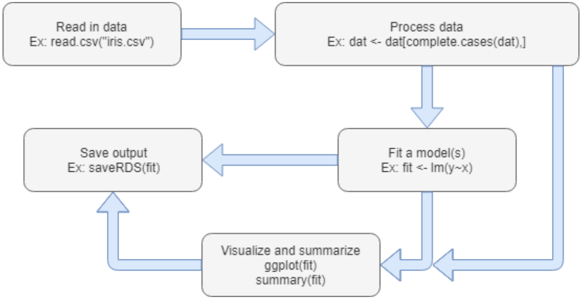
R on cluster (non-interactive R)

Cluster Computation

- Dan Barker danbarke at umich.edu
- Cluster System Administrator

R workflow for statistical analysis

- Bootcamps throughout semester on each area



Advanced control structures

Common misconception: “loops in R are slow”

Explain

Loops: often not necessary

- Many R functions are "vectorized" (vector in, vector out)
- While most other languages require loops, R does not:

```
a = c(5, 2, 4, 12, 1)
b = c(2, 0, 3, -1, 2)

a + b       # vector + vector = vector
```

```
## [1]  7  2  7 11  3
```

Takehome: When possible, operate on vectors and matrices, don't loop over each row/position index

Problem: not all functions are vectorized

- Some functions, like read.table() for reading a table of data into R, are not vectorized
- But what if we have a list of files to read in? “data1.txt”, “data2.txt”, “data3.txt”, . . . , “data50.txt”

```
   # the 50 data set names
file_names = paste0("data", 1:50, ".txt")
```

Attempt it, will cause error:

```
read.table(file_names)  # error!
```

map() from purrr package to avoid loop

- map() allows you to apply a function to each element of a vector
- faster, easier to read than a loop

```
read.table(file_names)  # error!
```

```
    # list of 50 data sets
my_dat_list = map(file_names, read.table)
    # results from reading data1.txt
my_dat_list[[1]]
    # results from reading data50.txt
my_dat_list[[50]]
```

Create your own functions for map

Generate random samples from a Normal with different variances

```r
draws = map(2:20, function(x) rnorm(25, mean=0, sd=x))
# returns a list with vectors of draws

# first vector of draws:
draws[[1]]

# last vector of draws:
draws[[19]]
```

Create your own functions for map

Now, estimate the standard error:

```
map(draws, sd)  # sd() is standard deviation in R
map_dbl(draws, sd) # numeric vector
```

For loop vs map() version

- Standard for loop:
- map():

Sometimes loops are required

Sometime you just need loops:

- Growth model, new values depend on previous values

```r
N = 20
for (i in 2:30) {
  f = rpois(1, 0.15*N[i-1])    # births
  d = rbinom(1, N[i-1], 0.1)  # deaths
  N[i] = N[i-1] + f - d
}

plot(seq_along(N), N)
```

Show the plot!

Problem: Growing object is slow

```r
N = 20
for (i in 2:30) {
  f = rpois(1, 0.15*N[i-1])    # births
  d = rbinom(1, N[i-1], 0.1)   # deaths
  N[i] = N[i-1] + f - d
}

plot(seq_along(N), N)
```

- Our growth model loop is slow because we are growing a vector at each iteration
- Solution pre-allocate vector (or any data type), then fill with loop

Efficient memory usage

Improved code:

```r
# Pre-allocate to correct size
N = vector(mode = "numeric", length = 30)
N[1] = 20
for (i in 2:30) {
  f = rpois(1, 0.15*N[i-1])   # births
  d = rbinom(1, N[i-1], 0.1)  # deaths
  N[i] = N[i-1] + f - d
}

plot(seq_along(N), N)
```

overview of loops

- Although slow at times, writing code is a trade-off between time consumed by the programmer and time gained with efficient code
- Most times, loops are fast (enough) and convenient
- Strategy: Write a loop, then if slow, re-write
- Use vectorized functions whenever possible
- Map when function is not vectorized
  - Avoids the need to track indexes
  - Does the pre-allocation for you
- If you use loops, avoid growing lists/collections/data.frames
  - pre-allocate memory

Simulation

Many built-in distribution functions in R:

They have prefix r, q, p, . . . for random, . . .

- probability density: d____
- cumulative probability density p_____
- quantile: q____
- random draw: r_____

Example: uniform unif

Simulation

distribution	function
Normal	rnorm(n, mean=0, sd=1)
exponential	rexp(n, rate=1)
uniform	runif(n, min=0, max=1)
gamma	rgamma(n, shape, scale=1)
poisson	rpois(n, lambda)
Weibull	rweibull(n, shape, scale=1)
Cauchy	rcauchy(n, location=0, scale=1)
beta	rbeta(n, shape1, shape2)
Student t	rt(n, df)
binomial	rbinom(n, size, prob)
logistic	rlogis(n, location=0, scale=1)

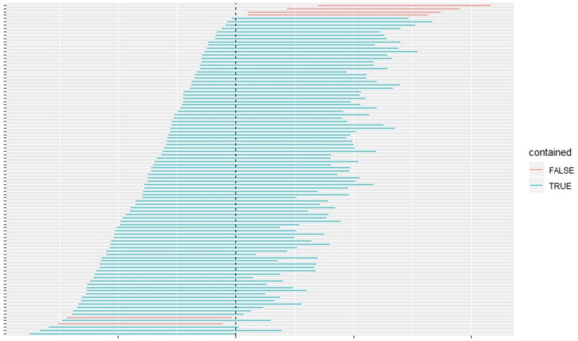
Table 3: Built-in distributions

Simulation: Confidence intervals

Open simulations.Rmd Rmarkdown document in course files

Exercise: Is the coverage rate of confidence intervals for the mean accurate?

- Given conditions similar to ours, does our method for generating confidence intervals capture the true value 95% of the time?



Simulate lm

Need to come up with some question to answer.

Bootstrap lm results

Useful R packages

- ggplot2 - Best package for plotting data
    - getting started:
- dplyr - Best package for manipulating/processing data
    - getting started:

Future bootcamps

- dplyr: date _____
- ggplot2: date _____