Basic R

J. Alexander Branham

October 2017

Intro to R

Welcome

- `https://jabranham.com/learn-r`
- pdf available
- html too for easy copy/paste

What is R

R is ”GNU S”, a freely available language and environment for statistical computing and graphics which provides a wide variety of statistical and graphical techniques: linear and nonlinear modelling, statistical tests, time series analysis, classification, clustering, etc

Why use R?

- Free & open source
- Available on nearly every platform
- Extensible (CRAN)
    - We’ll be using base R in this workshop
- Documentation & community
- Graphics
- Nerd cred

How to use R?

if you have installed Rstudio, then open it else open R

Rstudio

- File > open new R script
- Top left: editor
- Bottom left: R console
- Tip: nearly all your commands should be typed in an R script, then sent to the console for evaluation
    - exceptions: `install.packages()`, help queries

Data Types

TYPES OF DATA

- What are basic data types?
- logical, numeric, character
    - also complex and raw, but we'll ignore those

LOGICAL

- Statement of truth-y-ness

```
c(TRUE, FALSE, TRUE, FALSE, TRUE, TRUE, TRUE)
```

```
[1]  TRUE FALSE  TRUE FALSE  TRUE  TRUE  TRUE
```

```
3 != 4
```

```
[1] TRUE
```

```
"Democat" %in% c("Democrat", "Independent", "Republican")
```

```
[1] FALSE
```

LOGICAL STATEMENTS POP QUIZ

```
## Statements of truthyness
7 == 2
3 != 7
7 >= 2
2 >= 7
## And/or
2 == 2 & 2 > 3
2 == 2 | 2 > 3
```

LOGICAL STATEMENTS, ANSWERS

```
## Statements of truthyness
7 == 2 # FALSE
3 != 7 # TRUE
7 >= 2 # TRUE
2 >= 7 # FALSE
## And/or
2 == 2 & 2 > 3 # FALSE
2 == 2 | 2 > 3 # TRUE
```

NUMERIC

- numeric is umbrella term for ”double” and ”integers”
- stores numbers:

```
(x <- c(1, exp(1), pi, 10384.287459))
```

```
[1]     1.000000     2.718282     3.141593 10384.287459
```

```
is.numeric(x)
```

```
[1] TRUE
```

CHARACTER

- character type represents letters/words:

```
(myname <- c("My name is Alex"))

[1] "My name is Alex"

(myname2 <- c("My", "name", "is", "Alex"))

[1] "My"   "name" "is"   "Alex"

length(myname)

[1] 1

length(myname2)
```

COERCION

- vectors can have only one data type:

```
(x <- c("My name", 3 == 4, 7.27))
```

```
[1] "My name" "FALSE"   "7.27"
```

```
class(x)
```

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

COERCION, CONTINUED

- anything can be coerced to a character
- logicals can be coerced to numeric
    - TRUE is 1, FALSE is 0

ATOMIC VECTORS

- all the above are called atomic vectors
- useful to remember this when R yells at you

LISTS

- sometimes we need to store more than one type of data
- we can do this with a list

LISTS, CONTINUTED

```
list(c(1.82, 1940, 93.20, 192.917),
     c("Beyonce", "Lady Gaga", "Pink"),
     c(TRUE, FALSE, TRUE, TRUE, FALSE, TRUE, FALSE, FALSE))

[[1]]
[1]    1.820 1940.000   93.200  192.917

[[2]]
[1] "Beyonce"   "Lady Gaga" "Pink"

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

NAMES

- we can name elements of vectors:

```
list(n = 10,
     dv = c(1, 3, 5, 7, 9, 19, 92, 4, 10, 4))
```

```
$n
[1] 10

$dv
 [1]  1  3  5  7  9 19 92  4 10  4
```

DIMENSIONS

- all the vectors we've worked with so far have been single-dimension
- but we often work with two dimensional data
    - rows are observations
    - columns are variables

MATRIX

```
matrix(c(1, 2, 3, 4, 5, 6), nrow = 2)
```

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

DATA.FRAME

- matrix where columns can be different types:

```
data.frame(x = 1:3,
           y = c("a", "b", "c"),
           z = c(TRUE, FALSE, TRUE))
```

```
  x y     z
1 1 a  TRUE
2 2 b FALSE
3 3 c  TRUE
```

DATA STRUCTURES

	homogeneous	heterogeneous
1d	atomic vector	list
2d	matrix	data.frame
nd	array	

Subsetting

Subsetting

- oftentimes, we are interested in subsetting
- how to refer to a specific column or row?

[

- the [function is how we subset

```
x <- 1:10
```

[for positive numbers

```
x[c(1, 7)]
```

```
[1] 1 7
```

[for negative numbers

```
x[-c(1, 7)]
```

```
[1]  2  3  4  5  6  8  9 10
```

[for logical statements

```
x[x > 3]
```

```
[1]  4  5  6  7  8  9 10
```

[for 2D data

```
(dat <- data.frame(x = 1:3,
                  y =  c("a", "b", "c"),
                  z = c(TRUE, FALSE, TRUE)))
```

```
  x y     z
1 1 a  TRUE
2 2 b FALSE
3 3 c  TRUE
```

[for specific elements

```
dat[1, 3]
```

```
[1] TRUE
```

[for whole rows/columns

```
dat[, 1]
```

```
[1] 1 2 3
```

```
dat[3, ]
```

```
  x y    z
3 3 c TRUE
```

[

When you have a list, [always returns a list:

```
mylist <- list(x = 1:10, y = pi, z = c(TRUE, FALSE))
mylist[2]
```

```
$y
[1] 3.141593
```

[[for elements of lists

`[[` will return the actual element:

```
mylist[[2]]
```

```
[1] 3.141593
```

Subsetting by name

- we can subset by name so we don’t have to remember/figure out positions

```
dat[, c("x", "y")]
```

```
  x y
1 1 a
2 2 b
3 3 c
```

Subsetting by name - $

This is common so $ provides a quicker way:

```
dat[["x"]]
```

```
[1] 1 2 3
```

```
dat$x
```

```
[1] 1 2 3
```

Distributions

Distributions in R

- R has functions dealing with probability distributions built in
- They share common prefixes depending on what you want:

What you want	prefix
cdf	p
quantile (inverse cdf)	q
random draw	r
density	d

Common distributions

R's name	name
norm	normal
unif	uniform
t	t
binom	binomial
weibull	weibull
beta	beta
hyper	hypergeometric
nbinom	negative binomial
gamma	gamma

Conditionals

CONDITIONALS

- conditional statements:
- If (this one thing [condition]), then (do this other thing), else (do this different other thing)
- In R, need to consider whether (condition) is of length 1 or > 1
- Let's start when (condition) is length one

IF, THEN, ELSE

```
x <- 3
if (x == 7) {
  print("x is 7")
} else {
  print("x is not 7")
}

[1] "x is not 7"
```

IF, THEN, ELSE WITH LOGICALS

```
x <- TRUE
if (x) {
  print("That's true")
} else {
  print("That's false")
}

[1] "That's true"
```

CONDITIONS WITH LENGTH > 1

- remember: if, else only works if condition is of length one

```
x <- 1:10
if (x > 5){
  TRUE
} else {
  FALSE
}
```

```
[1] FALSE
Warning message:
In if (x > 5) { :
  the condition has length > 1 and only the first element will be
```

IFELSE, CONTINUTED

```
x <- 1:10
ifelse(x > 5, TRUE, FALSE)
```

```
[1] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
```

Writing Functions

FUNCTIONS

- You can write a function in R quite easily
- Let’s say we want to write a function to find the mean

CUSTOM MEAN FUNCTION

```
my_mean <- function(x){
  sum(x) / length(x)
}
my_mean(0:10)
```

```
[1] 5
```

MISSING VALUES IN CUSTOM MEAN FUNCTION

```
my_mean <- function(x, na.rm = FALSE){
  if (na.rm) {
    x <- x[!is.na(x)]
  }
  sum(x) / length(x)
}
x <- c(1, NA, 3)
my_mean(x, na.rm = TRUE)
```

```
[1] 2
```

Using Loops

LOOPS

- computers are much better than humans at doing repetitive tasks quickly & without error

LOOPS

- computers are much better than humans at doing repetitive tasks quickly & without error

- loops are a common way of doing something similar multiple times
- we'll talk about for, which loops a prescribed number of times

LOOPS

- computers are much better than humans at doing repetitive tasks quickly & without error

- loops are a common way of doing something similar multiple times
- we'll talk about for, which loops a prescribed number of times

- R has while and repeat loops as well, which loop until a logical check fails (returns FALSE)

FOR

pseudo-code structure of `for`:

```
output <- vector("numeric", length = 72) ## pre-allocate output!
for (something in somevector){           ## defined sequence
   do stuff, referring to each element of ##body
   somevector sequentially with the
   placeholder something
}
```

FOR LOOPS, EXAMPLE

```
x <- 6:10
for (i in x) {
  print(i)
}
```

```
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

FOR LOOPS, EXAMPLE 2

```
x <- 6:10
y <- vector(length = length(x))
for (i in seq_along(x)) {
  if (i > 1){
    y[i] <- x[i] + y[i - 1]
  } else {
    y[i] <- x[i]
  }
}
## what is y?
```

for loops, example 2 answer

```
y
```

```
[1]  6 13 21 30 40
```

FOR LOOPS, EXAMPLE 3

```r
means <- vector()

for (i in names(mtcars)) {
  means[[i]] <- mean(mtcars[[i]])
}

## What will means be?
```

FOR LOOPS, EXAMPLE 3 ANSWER

```
means
```

```
      mpg        cyl       disp         hp       drat         wt
20.090625   6.187500 230.721875 146.687500   3.596563   3.217250
       vs         am       gear       carb
 0.437500   0.406250   3.687500   2.812500
```

FOR LOOP QUIZ

```
## what does this code do?
x <- list.files(pattern = "*.csv")
data <- vector("list", length = length(x))
for (i in x) {
  data[[i]] <- read.csv(i)
}
```

NOTES ON LOOPING

- You may see a lot of advice online against loops
- They used to be slow in R, not the case anymore
- So long as you’re smart (pre-allocate output length!)

The apply Family

THE APPLY FAMILY

- The apply family of functions make our life easier by applying functions over ”stuff”
- Like a pre-built loop
- apply, lapply, sapply, vapply, mapply, rapply, tapply
- We’ll look at apply and lapply

APPLY

```
apply(X, MARGIN, FUN)
```

APPLY EXAMPLE

```
apply(mtcars, 2, mean)
```

```
      mpg        cyl       disp         hp       drat         wt
20.090625   6.187500 230.721875 146.687500   3.596563   3.217250
       vs         am       gear       carb
 0.437500   0.406250   3.687500   2.812500
```

APPLY YOUR OWN FUNCTIONS

Note that we can apply our own functions!

```
apply(mtcars, 2, my_mean, na.rm = TRUE)
```

```
      mpg        cyl       disp         hp       drat         wt
20.090625   6.187500 230.721875 146.687500   3.596563   3.217250
       vs         am       gear       carb
 0.437500   0.406250   3.687500   2.812500
```

LAPPLY

```
lapply(X, FUN) # always returns a list
```

LAPPLY EXAMPLE

```r
lapply(mtcars, mean)
```

```
$mpg
[1] 20.09062

$cyl
[1] 6.1875

$disp
[1] 230.7219

$hp
[1] 146.6875
```

SAPPLY

- `lapply` always returns a list
- `sapply` will simplify this (e.g. to a numeric vector) if it can

```
sapply(mtcars, mean)
```

```
      mpg        cyl       disp         hp       drat         wt
20.090625   6.187500 230.721875 146.687500   3.596563   3.217250
       vs         am       gear       carb
 0.437500   0.406250   3.687500   2.812500
```

Putting it all together

FUNCTION

Create a function to represent this

$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ \frac{1}{3} & \text{for } 0 \leq x < 1 \\ \frac{2}{3} & \text{for } 1 \leq x < 2 \\ 0 & \text{for } 2 < x \end{cases}$$

FUNCTION, ANSWER

```
myfun <- function(x){
  ifelse(x >= 0 & x < 1, 1 / 3,
         ifelse(x >= 1 & x < 2, 2 / 3, 0))
}
```

REJECTION SAMPLER

using the function from the last slide, construct a function that will return a vector of samples using rejection sampling. Make it take one argument *n* the number of samples.

REJECTION SAMPLER, ANSWER

```
myreject <- function(n){
  x <- runif(n, 0, 2)
  y <- runif(n, 0, 2 / 3)
  reject <- y > myfun(x)
  x[!reject]
}
```

REJECTION SAMPLING WITH MULTIPLE N'S

- We want to test the effect of varying n on our rejection sampler.
- Calculate the mean of the samples from our rejection sampler varying n from 1 to 1,000

REJECTION SAMPLING WITH MULTIPLE N, ANSWERS

```
ns <- seq(1, 1000)
means <- sapply(ns, function(n){mean(myreject(n))})
## plot(means)
## summary(means)
```