an introduction to R for epidemiologists

the basics

Charles DiMaggio, PhD, MPH, PA-C

Professor of Surgery and and Population Health
New York University School of Medicine
Bellevue Hospital
Division of Trauma and Surgical Critical Care
462 First Avenue, New York, NY 10016

Spring 2017

- http://www.injuryepi.org/
- Charles.DiMaggio@nyumc.org

Outline

1. functions for epidemiologists
   - marginals - apply()
   - stratified analysis - tapply(), by(), aggregate()
   - summary statistics - sweep()
   - cross tabulations - table()

2. indexing to manipulate data
   - position
   - logical
   - indexing matrices and arrays
   - indexing lists and data frames

Outline

What is a function?

an R function is an object (like most everything in R) that "does something"

- returns information
- processes data
- transforms objects
- conducts analyses
- creates other functions!

basic form:

```
myResults<-functionName(object, arguments)
```

Where "arguments" are a set of parameters or information the function needs to , well...function

basic functions

return information about objects

three very useful functions:

- **str()** - structure or summary **(good place to start)**
- **head()** - displays first 6 lines of a data frame, (equivalent to data[1:6,])
- **names()** - vector of *list/dataframe* names

Other helpful functions

- mode() ; class()
- nrow() ; ncol() - helpful if need *n* in equation
- rownames() ; colnames() - vector of *matrix* row/column names
- row.names() - vector of *dataframe* row names

Outline

the apply function

- functions like *sum()* and *mean()* work on vectors
- *apply()* to use a vector-based function on the *margins* or *dimensions* of a matrix or array
- convenient way to get marginal values
- *apply(object, dimension, function)*
  - you can specify more than one dimension or margin

try this

```
x<-matrix(c(10,20,30,40),2,2,byrow=T)
x
dimnames(x) <- list(c("e", "E"), c("d", "D"))
x
r.tot<-apply(x, 1, sum) #row totals
r.tot
x2<-cbind(x, Tot = r.tot) #add row margin totals
x2
c.tot<-apply(x2,2,sum)
c.tot
x.tot <- rbind(x2, Tot = c.tot)
x.tot
```

know your dimensions

```
x3<-array(c(1:12), c(2,3,2))
apply(x3, 1, sum)
```

"keep" the first dimension, "disolve" the other two dimensions:
sum(x3[1,,]) sum(x3[2,,]) sum(x3[3,,])
1+3+5+7+9+11 = 36
2+4+6+8+10+12= 42

```
apply(x3, c(1,2), sum)
```

"keep" the first two dimensions, "disolve the third: sum(x3[1,1,]) , x3[1,2,], x3[1,3,], x3[2,1,], x3[2,2,], x3[2,3,]

1+7=8	3+9=12	5+11=16
2+8=10	4+10=14	6+12=18

```
apply(x3, c(1,2,3), sum)  #try this
```

convenience functions based on apply()

```
x<-matrix(c(10,20,30,40),2,2,byrow=T)
rowSums(x)
colSums(x)
rowMeans(x)
colMeans(x)
addmargins(x)
```

there's an apply for that

lapply()- applies function to each component of list returns a list

```
x <- list(1:5, 6:10, 11:15); lapply(x, mean)
```

sapply()- like lapply() but simplifies results to vector

```
sapply(x, mean)
```

mapply() - like sapply(), but applies to each member of list in order

```
y <- list(16:20, 21:25, 26:30)
mapply(sum, x, y)
```

recap

if you think you need a loop, use apply

- functions like apply considered more computationally efficient than loops
- apply() for marginals
- R often returns lists as results of other operations, lapply() and sapply() can be helpful
  - lapply() returns another list, sapply() will try to simplify results to a vector or matrix
  - lapply() useful with dataframes where you can use it to get info like class() on each column
  - mapply() - takes an input matrix and returns results in form of data frame

consider the "**reshape**" or "**plyr**" packages

Outline

tapply() and by() to group values

stratified analyses

- apply a function to groups of values in a vector defined by a grouping or index factor
- any function (even user created) can be applied to strata of a vector
- tapply() returns an array, by() returns a list (class "by")

```
patients<-data.frame(patient=1:100, age=rnorm(100,mean=30,
 sd=10), gender=sample(c("M","F"),100, replace=T),
Tx=sample(c("Rx","placebo"),100, replace=T))

tapply(patients$age, patients$gender, mean)
by(patients$age, patients$gender, mean)

tapply(patients$age,list(patients$gender, patients$Tx), mean)
  tapply(patients$age, patients[,c(3,4)], mean)
  tapply(patients$age, patients[,c("gender", "Tx")], mean)
```

e.g. age and gender stratified population-based rates

tapply your own function

- tapply() and by() will accept user-created functions
- e.g. population-based age and gender-stratified hospital complication rates...

```
hospDat<-data.frame(hospital=sample(1:20,100, replace=T),
complications=round(rnorm(100,mean=30,
 sd=10)), gender=sample(c("M","F"),100, replace=T),
ageCat=sample(c("young","adult", "older"),100, replace=T))
head(hospDat)
(tapply(hospDat$complications, list(hospDat$gender,
hospDat$ageCat), sum))/29000*10000
myfx<-function(x){sum(x)/29000*10000}
tapply(hospDat$complications, list(hospDat$gender,
hospDat$ageCat), myfx)
```

aggregate()

more than one way to skin a cat

aggregate(num.var ~ cat.var1 * cat.var2, data=, FUN=)

BMI by sex and ethnicity

```
df<-data.frame(age=round(rnorm(100, 35,5)),
weight=round(rnorm(100,160,15)),hgt=round(rnorm(100,60,6)),
sex=sample(c("M", "F"), 100, replace=T),
clinic=sample(c("a", "b","c"),100, replace=T),
eth=sample(c("B", "W"), 100, replace=T))

aggregate((weight/(hgt^2))*703 ~ sex*eth, data=df, FUN=mean)
```

alternate syntax, weight and height by sex and ethnicity

```
aggregate(df[c("weight", "hgt")], by=list(gender=df$sex,
ethnicity=df$eth), FUN=mean)
```

Outline

sweep()

operates on rows or columns of a matrix by
specifying a statistic (often derived from matrix itself using apply())
and a mathematical operation with which to "sweep" that statistic

takes 4 arguments:

1. a *data* object
2. *dimension* (like apply)
3. *statistic* to sweep across that dimension
4. mathematical *operation* to perform

using sweep()

convert vector values to proportions

```
v <- c(1, 2, 3, 4, 5)
sum.v <- sum(v) # Step 1: statistic = sum
prop.v <- v/sum.v #Step 2: operation = division
```

convert matrix values to proportions

```
m<-matrix(round(rnorm(16, 20, 15)),2,2)
dimnames(m)<-list(behavior=c("type A", "type B"),
                  mi=c("yes", "no"))
m.rtot<-apply(m, 1, sum) # Step 1: statistic = apply sum
m.rdist<-sweep(m, 1, m.rtot, "/") #Step 2: operation = sweep "/"
sweep(m, 1, apply(m,1,sum), "/") # combine steps into one line
```

prop.table()

optimized convenience function based on apply and sweep

e.g. prop.table(m,1)

Outline

table(), prop.table(), ftable()

table() returns a frequency table

```
(t1<-table(df$sex))
(t2<-table(df$sex,df$eth))
(t3<-table(df$sex, df$eth, df$clinic))
```

table() plus prop.table())

```
prop.table(t1)
prop.table(t2)
prop.table(t3)
prop.table(t2, margin=1)
```

ftable() "flattens out" multi-dimensional tables

```
(t4<-ftable(df$sex, df$eth, df$clinic))
prop.table(t4)
```

about table()

- applied to a factor returns a frequency table of the factor *levels*
- default excludes missing values, override with "exclude=NULL"
- prop.table() based on sum of all cells, "margin=" 1 for rows, 2 for columns
- *xtabs()* returns similar results as table(), takes arguments differently, e.g xtabs(region+income)
- *CrossTable()* function in "gmodels" package gives SAS PROC FREQ-like tables

Outline

indexing is the key to working with R data

INDEXING IS THE KEY TO WORKING WITH R DATA

- position
- logical vector
- name

see indexing example and exercises document

indexing vector elements

locating the element

```
x <- c(chol = 234, sbp = 148, dbp = 78, age = 54)
x[1] # by position
x[x<100]# by logical
x["sbp"] # by name
```

replacing the element

```
x[1] <- 250 #by position
x[x<100] <- NA # by logical
x["sbp"] <- 150 # by name
```

Outline

indexing by position

including and excluding elements

```
x<-letters
x[11] #only the 11th element
x[-11] #exclude the 11th element
x[11:20] #members 11 to 20
x[-(11:26)] # all but members 11 to 20
x[-(11:100)] # careful...
```

Outline

logical operators

```
==  - IS (equivalent to)
! - NOT
& - AND
| - OR (if either or both comparison elements are TRUE)
xor - EITHER  (element-wise exclusive or operator, if either,
                but not both, comparison elements TRUE)
&&  || - control flow in "if" functions,  only the first
                element of logical is used.
```

which()

returns integer vector from Boolean operation

```
age <- c(8, NA, 7, 4)
which(age<5 | age>=8)
```

indexing with logical vectors

1. create a logical vector
2. use the logical vector to index data

```
myNames<-c("dopey" , "grumpy" , "doc" , "happy" , "bashful" ,
"sneezy" , "sleepy" )
myAges<-c(142, 240, 232, 333, 132, 134, 127)
myGenders<-c("m" , "m" , "f" , "f" , "f" , "m" , "m")

(young <- myAges < 150) #create logical vector ages
myNames[young] #index name vector using logical vector ages
myNames[!young] # old
male<- myGenders == "m" #logical vector males
myNames[male] #index names using logical vector males

myNames[young & male]
myNames[young | male]
```

using indexing to categorize data

indexing plus assignment

```
# simulate vector with 1000 age values
age <- sample(0:100, 1000, replace = TRUE)
mean(age) ;  sd(age)
agecat <- age # make copy
#replace elements agecat with strings for q category
agecat[age<15] <- "<15" # creating character vector
agecat[age>=15 & age<25] <- "15-24"
agecat[age>=25 & age<45] <- "25-44"
agecat[age>=45 & age<65] <- "45-64"
agecat[age>=65] <- "65+"
table(agecat) # get freqs
```

Outline

indexing a matrix

```
m<-matrix(round(rnorm(16,50,5)),2,2)
dimnames(m)<-list(behavior=c("type A", "type B"),
MI=c("yes", "no"))
```

1. by position

   ```
   m[1, ]; m[1, , drop = FALSE]; m[1,2]
   ```

2. by name

   ```
   m["type A",]
   m[, "no"]
   ```

3. by logical

   ```
   m[, 2] < 45 # logical vector
   m[,2][m[, 2] < 49] # index second column by criterion
   ```

more matrix indexing

```
m[m[,1]<50,] # return all rows meeting criterion m[,1]<50
```

note extra comma after 3, tells R to return all the rows in x for which the 1st column is <3

```
m2<-matrix(round(rnorm(81,50,5)),3,3)
lower.tri(m2)
upper.tri(m2)
```

indexing arrays

```
a<-array(sample(10:70,8, rep=T),c(2,2,2))
dimnames(a)<-list(exposure=c("e", "E"), disease=c("d", "D"),
confounder=c("c", "C"))
```

unexposed, diseased, confounder negative

1. by position

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

2. by name

   ```
   a["e","D","c"]
   ```

3. by logical

   ```
   a==48
   a<40
   z<-a<40
   aa<-a[z]
   aa
   ```

Outline

indexing lists

1. by position [[ ]] (bin) [ ] (contents)

```
l<- list(1:5, matrix(1:4,2,2),
    c("John Snow", "William Farr"))
l[[1]]
l[[2]][2,1]
l[[3]][2]
```

2. logical

```
char <- sapply(l, is.character)
char
epi.folk<-l[char]
epi.folk
```

indexing lists by name

results of conditional logistic model

```
# matched c-c conditional logistic abortion infertility
data(infert)
library(survival) # package with clogit()

mod1 <- clogit(case ~ spontaneous + induced +
               strata(stratum), + data = infert)
mod1 # default results (7x risk c spont AB, 4x induced)

str(mod1) ; names (mod1) #structure, names
mod1$coeff # name to index result (list element)

summod1<-summary(mod1) #more detailed results
names(summod1) #detailed list components
```

indexing data frames

name, logical

```
sparcs<-read.csv(file="../sparcsShort.csv", stringsAsFactors=F)
```

index rows

```
brooklyn<-sparcs[sparcs$county=="59",]
nyc<- sparcs$county=="58"| sparcs$county=="59"|
sparcs$county=="60"| sparcs$county=="61"| sparcs$county=="62"
nyc.sparcs<-sparcs[nyc,]
```

index columns

```
dxs<-sparcs[,"pdx"]
vars<-c("date", "pdx", "disp")
my.vars<-sparcs[,vars]
```

index rows and columns

```
sparcs2<-sparcs[nyc,vars]
```

subset()

alternative to indexing

args: dataframe, Boolean logical vector, variables to incl/excl

variables to include

```
brooklyn.sparcs<-subset(sparcs, county=="59",
select=c(date, pdx,disp))
```

range of variables

```
sparcs5<-subset(sparcs, subset= nyc,
select=c(county, pdx,disp))
```

excluding rows

```
sparcs5<-subset(sparcs, subset= nyc,
select=-c(county, pdx,disp))
```

replacing data frame elements

indexing plus assignment

data(infert)

1. position

```
infert[1:4, 1:2]
infert[1:4, 2] <- c(NA, 45, NA, 23)
infert[1:4, 1:2]
```

2. name

```
names(infert)
infert[1:4, c("education", "age")]
infert[1:4, c("age")] <- c(NA, 45, NA, 23)
infert[1:4, c("education", "age")]
```

3. logical

```
table(infert$parity)
# change values of 5 or 6 to missing
infert$parity[infert$parity==5 | infert$parity==6] <- NA
table(infert$parity)
```

Credit where credit is due...

- Tomas Aragon, MD, DrPH
  - Applied Epidemiology Using R
  - http://www.epitools.net/
- John Fox, PhD
  - An Introduction to Statistical Computing in R
  - http://socserv.mcmaster.ca/jfox/Courses/UCLA/index.html
- Bill Venebles, PhD
  - An Introduction to R
  - cran.r-project.org/doc/manuals/R-intro.pdf
- Phil Spector, PhD
  - Data Manipulation with R