

an introduction to R for epidemiologists

objects and data

Charles DiMaggio, PhD, MPH, PA-C

Departments of Anesthesiology and Epidemiology
College of Physicians and Surgeons
Columbia University
New York, NY 10032
cjd11@columbia.edu

P9489
applications of epidemiologic methods II
Spring 2013

Outline

- 1 objects
 - about objects
- 2 vector
 - logical vectors
- 3 matrix
- 4 array
- 5 list
- 6 dataframe
 - introduction to indexing data frames
- 7 data
 - getting your data into R

Outline

- 1 objects
 - about objects
- 2 vector
 - logical vectors
- 3 matrix
- 4 array
- 5 list
- 6 dataframe
 - introduction to indexing data frames
- 7 data
 - getting your data into R

5 important objects

objects are "specialized data structures"

- 1 vector - collection of like elements (numbers, characters...)
- 2 matrix - 2-dimensional vector
- 3 array - >2-dimensional vector
- 4 list - collection of *groups* of like elements any kind
- 5 dataframe - tabular data set, each row a record, each column a (like) element or variable

objects for epidemiologists

- matrix for contingency, e.g. 2x2, tables
- arrays for stratified tables
- dataframe for observations and variables
- factors for categorical variables
 - numeric representation of characters
 - read.table converts characters to factors
 - plot would return box plot not scatter
 - converted to dummy in reg

examples of R objects

```
x <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12)
y <- matrix(x, nrow = 2)
z <- array(x, dim = c(2, 3, 2))
mylist <- list(x, y, z)
```

```
names <- c("kookla", "fran", "ollie")
gender <- c("boy", "girl", "boy")
age <- c(28, 22, 34)
data <- data.frame(names, gender, age)
```

Outline

- 1 objects
 - about objects
- 2 vector
 - logical vectors
- 3 matrix
- 4 array
- 5 list
- 6 dataframe
 - introduction to indexing data frames
- 7 data
 - getting your data into R

modes

atomic vs recursive objects

mode - type of data: **numeric, character, logical, factor**

```
age <- c(34, 20); mode(age)
1t25 <- age<25
1t25
mode(1t25)
```

- **atomic** - only one mode
 - character, numeric, or logical, i.e. vectors, matrices, arrays
 - logical (1 for TRUE 0 for FALSE)
 - categorical - appear numeric but stored as factors
- **recursive** - more than one mode
 - lists, data frames, functions

atomic objects

all elements the **same**

vector: **one** dimension

```
y <- c("Tom", "Dick", "Harry") ; y #character  
x <- c(1, 2, 3, 4, 5) ; x #numeric  
z <- x<3 ; z #logical
```

matrix: **two**-dimensional vector

```
x <- c("a", "b", "c", "d")  
y <- matrix(x, 2, 2) ; y
```

array: **n**-dimensional vector

```
x <- 1:8  
y <- array(x, dim=c(2, 2, 2)) ; y
```

recursive objects

differ

list: **collections** of data

```
x <- c(1, 2, 3)
y <- c("Male", "Female", "Male")
z <- matrix(1:4, 2, 2)
xyz <- list(x, y, z)
```

dataframe: **tabular** (2-dimensional) list

```
subjno <- c(1, 2, 3, 4) ; age <- c(34, 56, 45, 23)
sex <- c("Male", "Male", "Female", "Male")
case <- c("Yes", "No", "No", "Yes")
mydat <- data.frame(subjno, age, sex, case) ; mydat
```

class (vs. mode)

a way of letting R know that an object is 'special'

- property assigned to an object that determines how generic functions operate on it
- if object does not have an explicit class assigned to it, usually assigned class same as mode
- has important implications for analysis

coercion

changing an object's mode

this is important

R will automatically *coerce* all the elements in an atomic object to a single mode (*character > numeric > logical*)

```
c("hello", 4.56, FALSE)
```

```
c(4.56, FALSE)
```

coercing objects

do it yourself

- `is.xxx / as.xxx` - to assess / coerce objects
xxx = vector, matrix, array, list, data.frame, function, character, numeric, factor, na etc...

```
is.matrix(1:3) # false
as.matrix(1:3)
is.matrix(as.matrix(1:3)) # true
# coercing factor to character
sex <- factor(c("M", "M", "M", "M", "F", "F", "F", "F"))
sex
unclass(sex) #does not coerce into character
as.character(sex) #works
```

Review

basic characteristics of R objects

- Objects - vector, matrix, array, list, dataframe
- `mode()` - "type" of object: numeric, character, factor, logical
 - vectors and matrices - atomic, one mode only
 - lists and data frames - recursive, can be of >1 mode
- `class()` - for simple vectors, same as mode
 - more complex objects, array and data frames have their own class
 - affects how printed, plotted and otherwise handled

Outline

- 1 objects
 - about objects
- 2 **vector**
 - logical vectors
- 3 matrix
- 4 array
- 5 list
- 6 dataframe
 - introduction to indexing data frames
- 7 data
 - getting your data into R

vectors are 1-dimensional strings of like elements

the basic building block of data in R

use them for quick data entry

fun with vectors

```
y<-1:5           #create a vector of consecutive integers
y+2             #scalar addition
2*y            #scalar multiplication
x<-c(1,3,2,10,5)
cumsum(x)
sort(x)
sort(x, decreasing=T) # decreasing order
```

more fun with vectors

vectorized arithmetic

```
c(1,2,3,4)/2
```

```
c(1,2,3,4)/c(4,3,2,1)
```

```
log(c(0.1,1,10,100), 10)
```

```
c(1,2,3,4) + c(4,3)
```

```
c(1,2,3,4) + c(4,3,2)
```

creating vectors

concatenation and scanning

- `c()` - the concatenation function
- `scan()` - enter vector elements interactively at the keyboard
 - no need to type `c`, parentheses, and commas
 - hit enter twice to create the vector

```
> x <- scan()  
1: 45 67 23 89 5:  
Read 4 items  
> x  
[1] 45 67 23 89
```

creating vectors

sequences

the sequence operator `:`

```
-9:8
```

`seq()` greater flexibility

```
> seq(1, 5, by = 0.5) # specify interval  
> seq(1, 5, length = 8) #specify length
```

operations on vectors

```
x <- rnorm(100) ; sum(x)
```

```
x <- rep(2, 10) ; cumsum(x)
```

```
mean(x) ; sum(x)/length(x)
```

```
var(x) #sample variance
```

```
sd(x) ; sqrt(var(x)) #sample standard deviation
```

```
x <- rnorm(100); y <- rnorm(100)
```

```
var(x, y) # covariance
```

write your own operations

$$var = s^2 = \sum(x_i - \bar{x})^2 / n - 1$$

```
sum((x-mean(x))^2)/(length(x)-1)
```

$$cov = s^2 = \sum(x_i - \bar{x})(y_i - \bar{y}) / n - 1$$

```
sum((x-mean(x))*(y-mean(y)))/(length(x)-1)
```

indexing vectors

myVector[n]

```
cousins <- c("Katie", "C J", "Lizzie", "Claire", "Paul")
cousins[1]
cousins[4]
cousins[6]
cousins[-1]
cousins[c(2,4)]
cousins[c(4,2)]
```

sorting, ordering and ranking vectors

`sort()` rearranges the same vector

```
x <- c(12, 3, 14, 3, 5, 1)
sort(x)
rev(sort(x))
```

`order()` to rearrange another vector

```
ages<- c(8, 6, 7, 4, 4)
order(ages)
cousins <- c("Katie", "C J", "Lizzie", "Claire", "Paul")
cousins[order(ages)]
```

creates an *index* of positional integers to rearrange elements of *another* vector, e.g. `cousins[c(4,5,2,3,1)]`, 4th element (Claire) in 1st position, 5th element (Paul) in 2nd position, 2nd element (CJ) in 3rd position, etc...

`rank()`, doesn't sort

```
x <- c(12, 3, 14, 3, 5, 1)
rank(x)
```

plyr::arrange to order dataframe

saves typing

- need the plyr package
- only for dataframes
- more intuitive (perhaps)

```
mynames<-c("Hippocrates", "Galen", "Graunt", "Castore",  
           "Snow", "Farr")  
myages<-c(88, 59, 63, 75, 42, 68)  
mydat<-data.frame(mynames, myages)  
mydat  
order(mydat$myages)  
mydat[order(mydat$myages),]  
library(plyr)  
arrange(mydat, myages)
```


combining and indexing vectors

- `c()` - concatenates (appends) vectors
- `cbind()`, `rbind()` - creates matrix
- `xtable()` - contingency tables

```
x<-c("red", "yellow", "orange")
```

```
x2<-rep(x, 6)
```

```
x2
```

```
x2[4]
```

myVector[n]

caution: recycling

- `cbind()` `rbind()` - will recycle data
- when performing vector or mixed vector and array arithmetic, short vectors are extended by recycling till they match size of other operands
- R may return an error message, but still complete the operation

Outline

- 1 objects
 - about objects
- 2 **vector**
 - **logical vectors**
- 3 matrix
- 4 array
- 5 list
- 6 dataframe
 - introduction to indexing data frames
- 7 data
 - getting your data into R

logical vectors

the special vector...

series of TRUEs and FALSEs (Ts and Fs)

created with relational operators:

<, >, <=, >=, ==, !=

used to index, select and subset data

about logical vectors

logical operators are the key to indexing, and indexing is the key to manipulating data

```
= <= > >= == !
```

```
x<-1:26
```

```
temp<- x > 13    #logical vector temp  
                 #same length as vector x  
                 #TRUE= 1, when condition met  
                 #FALSE = 0, when not met
```

```
sum(temp)
```

indexing is key to using R

...about indexing

```
x[-11] #exclude the 11th element
x[-(2:4)] #all but members 2 to 4
x[11:20]
x[-(11:100)]
myIndex<- x < 8 | x > 15 # OR statement (vs. &)
myIndex
x[myIndex]
```

indexing missing values

- `is.na()` - returns logical vector of NA positions
useful for replacing missing values

```
x[is.na(x)] <- 999
```

- `!is.na()` - positions that do not contain NA
- `is.nan()` - not a number/ `is.infinite()` - infinite

Outline

- 1 objects
 - about objects
- 2 vector
 - logical vectors
- 3 matrix**
- 4 array
- 5 list
- 6 dataframe
 - introduction to indexing data frames
- 7 data
 - getting your data into R

a matrix is a 2-dimensional vector

2x2 and contingency tables

```
myMatrix<-matrix(c("a","b","c","d"),2,2)
myMatrix
myMatrix2<-matrix(c("a","b","c","d"),2,2, byrow=T)
myMatrix2
colnames(myMatrix2)<-c("case", "control")
rownames(myMatrix2)<-c("exposed", "unexposed")
myMatrix2
```

myMatrix[r,c]

creating a matrix from data

`table()` - from characters

```
dig<-read.csv("http://www.columbia.edu/~cjd11/charles_dimaggio/  
DIRE/resources/R/dig.csv", header=T,  
  stringsAsFactors=F) #digitalis data  
names(dig)  
table(dig$TRTMT,dig$DEATH)
```

`xtabs()` - using formula

```
xtabs(~TRTMT+DEATH, data=dig)  
xtabs(~TRTMT+DEATH+DIABETES, data=dig) #strata
```

`ftable()` - 2-dimensional flat contingency table from multi-dimensional

```
ftable(xtabs(~TRTMT+DEATH+DIABETES, data=dig))
```

naming a matrix

rownames, colnames, dimnames

```
dat <- matrix(c(178, 79, 1411, 1486), 2, 2)
rownames(dat) <- c("Type A", "Type B") #rows only
colnames(dat) <- c("Yes", "No") #columns only
#rows and columns
dimnames(dat) <- list(c("Type A", "Type B"), c("Yes", "No"))
dimnames(dat) <- list(Behavior = c("Type A", "Type B"),
"Heart attack" = c("Yes", "No"))#rows, columns, and fields
```

transposing and reversing matrix fields

```
ugdp <- matrix(c(30, 174, 21, 184), 2, 2,  
  dimnames = list("outcome" = c("survive", "die"),  
  "treatment" = c("tolbutamide", "placebo")))  
t(ugdp) # transpose  
a <- matrix(1:30, 5, 6)  
t(a)  
ugdp[2:1,] # reverse rows  
ugdp[,2:1] # reverse columns  
ugdp[2:1,2:1] # reverse rows and columns
```

Outline

- 1 objects
 - about objects
- 2 vector
 - logical vectors
- 3 matrix
- 4 array
- 5 list
- 6 dataframe
 - introduction to indexing data frames
- 7 data
 - getting your data into R

an array is an n-dimensional vector

stratified epi tables

stratified UGDP contingency table:

outcome status vs. treatment status vs. age group

```
ugdp.age <- c(8, 98, 5, 115, 22, 76, 16, 69)
ugdp.age <- array(ugdp.age, c(2, 2, 2))
dimnames(ugdp.age) <- list(Outcome=c("Deaths", "Survivors"),
  Treatment=c("Tolbutamide", "Placebo"),
  "Age group"=c("Age<55", "Age>=55"))
ugdp.age
```

myArray[r,c,n]

Schematic 4-Dimensional Array

(From Aragon and Enanaoria. *Applied Epidemiology Using R*)

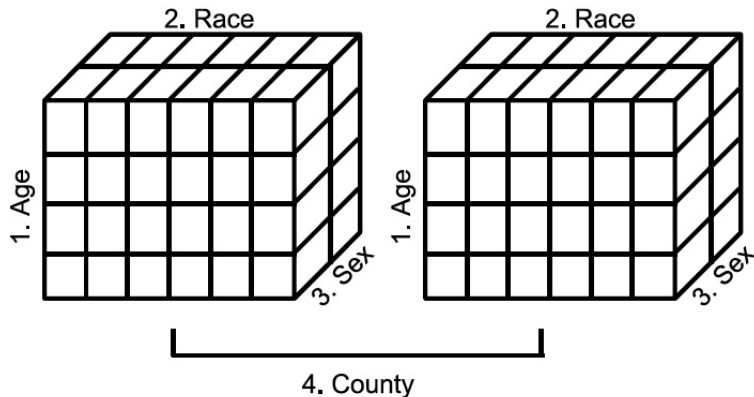


Fig. 2.2. Schematic representation of a 4-dimensional array (Year 2000 population estimates by age, race, sex, and county)

creating arrays

table() and xtab()

see above to read in dig file...

table() - from characters

```
a<-table(dig$TRTMT,dig$DEATH,dig$DIABETES)
a<-table(Treatment = dig$TRTMT, Outcome = dig$DEATH ,
Diabetes = dig$DIABETES) # need to add names
a
a[1,2,1]
```

xtabs() - using formula

```
xtabs(~TRTMT+DEATH+DIABETES, data=dig)
#no need to name
```


Outline

- 1 objects
 - about objects
- 2 vector
 - logical vectors
- 3 matrix
- 4 array
- 5 list**
- 6 dataframe
 - introduction to indexing data frames
- 7 data
 - getting your data into R

a list is a recursive collection of unlike elements

like "variables" and "observations"

- double brackets `[[...]]` index the variable
- single brackets `[...]` index the observation
- `object$name` if a named list
- often used to "store" function results
 - `str()` is your friend
- also, see discussion here

```
x <- 1:5 ; y <- matrix(c("a","c","b","d"), 2,2)
z <- c("Peter", "Paul", "Mary")
mm <- list(x, y, z) ; mm
mm[[2]][2,2]
```

`myList[[n]][r,c(,n)]`

Outline

- 1 objects
 - about objects
- 2 vector
 - logical vectors
- 3 matrix
- 4 array
- 5 list
- 6 dataframe**
 - introduction to indexing data frames
- 7 data
 - getting your data into R

dataframes

tabular epi data sets

2-dimensional tabular lists with equal-length fields

each row is a record or observation

each column is a field or variable (usually numeric vector or factors)

```
data(infert) ; str(infert)
infert[1:10, 1:6]
infert$age ; mode(infert$age) ; class(infert$age)
infert$stratum ; mode(infert$stratum) ; class(infert$stratum)
infert$education ; mode(infert$education)
class(infert$education)
```

"a list that behaves like a matrix"

data types in dataframes

the R way

data type	mode	class	e.g.
nominal	numeric	factor	infert\$education
ordinal	numeric	ordered factor	esoph\$agegp
integer	numeric	integer	infert\$stratum
ratio	numeric	numeric	infert\$age

creating data frames

① data.frame()

```
x <- data.frame(id=1:2, sex=c("M","F"))
```

② as.data.frame()

```
x <- matrix(1:6, 2, 3); as.data.frame(x)
```

③ as.table() / ftable() with as.data.frame to convert array

```
x <- array(1:8, c(2, 2, 2))
```

```
dimnames(x) <- list(Exposure=c("Y", "N"), Disease =  
  c("Y", "N"), Confounder = c("Y", "N")) # needs labels
```

```
as.data.frame(ftable(x))
```

```
as.data.frame(as.table(x))
```

④ read.table(), read.csv(), read.delim(), read.fwf()

```
dig <- read.csv("../dig.csv", header=T) str(dig)
```

(caution: default char → factor, numeric → integer)

changing dataframe column and row names

```
library(epitools)
data(oswego)
oswego2<-oswego
oswego2[1:5, 1:8]
names(oswego2)
names(oswego2)[3]
names(oswego2)[3]<-"gender"
row.names(oswego2)
row.names(oswego2)<-sample(as.character(1:75), 75, replace=F)
row.names(oswego2)
```

Outline

- 1 objects
 - about objects
- 2 vector
 - logical vectors
- 3 matrix
- 4 array
- 5 list
- 6 dataframe**
 - introduction to indexing data frames
- 7 data
 - getting your data into R

indexing data frames: births in Brooklyn

position, name, logical

```
hosp<-read.csv(file=~ /sparcsShort.csv",
stringsAsFactors=F)
head(hosp)
# manhatt = 60, brookl=59, bx=58, qns=61, si=62
mode(hosp$pdx)
myCounty<-hosp$county==59
head(myCounty)
#births = V3000
myDx<-hosp$pdx=="V3000"
head(myDx)
myObs<-hosp$county==59 & hosp$pdx=="V3000 " #note space
myVars<-c("age", "sex", "disp")
myFile<-hosp[myObs,myVars]
head(myFile)
```

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)
```

Outline

- 1 objects
 - about objects
- 2 vector
 - logical vectors
- 3 matrix
- 4 array
- 5 list
- 6 dataframe
 - introduction to indexing data frames
- 7 data**
 - getting your data into R

R comes pre-packaged with some data sets

- `data()` - list datasets
- `data(data set)` - load a dataset
- `?data` - information about a dataset, e.g., `?infert`

Outline

- 1 objects
 - about objects
- 2 vector
 - logical vectors
- 3 matrix
- 4 array
- 5 list
- 6 dataframe
 - introduction to indexing data frames
- 7 data**
 - getting your data into R

```
read.table()
```

*read.table() is how to get data into
R*

read.table() options

- *file* file name, URL or connection object *Windows double backslashes*
- *sep=* spaces, tabs, hard returns (default), or specify
- *header=TRUE* named columns, (1st row one fewer column)
- *col.names= / row.names=*
- *na.strings=* how to treat missing values
- *skip=* rows not to read in
- *nrows=* number observations to read in
- *comment.char=* if pound character is valid
- *fill=TRUE* if some observations have more variables than others (otherwise R expects all observations to be equal)
- **stringsAsFactors=FALSE** turn off auto-convert strings to factors

convenience functions based on `read.table()`

- **`read.csv()`** comma separated values
- `read.csv2` semi-colon separated values
- `read.delim` tab-delimited values
- **`read.fwf`** fixed width file, i.e. no separators, *but* `width=` takes vector of field widths
- `edit()` with data frame as an argument invokes spreadsheet interface, good for small changes

reading in a .csv file

the workhorse

```
deathZIPs<-read.csv("~/deathZIPs.csv",  
stringsAsFactors=FALSE)  
head(deathZIPs)  
str(deathZIPs)  
names(deathZIPs)
```

example of reading in a fixed width file

sparcs

```
pos <- c(-2, -5, -7, -2, -1, 6, -3, -6, -3, -1, -6,  
  -1, 3, -3, 2, 5, 1, 2, 1, -1, -1, -1, -1, -6, 6,  
  -8, -8, -8, -8, -8, -8, -8, -8, -8, -8, -8, -8, -8,  
  -8, -7, -7, -7, -7, -7, -7, -7, -7, -7, -7, -7, -7,  
  -7, -7, -7, -4, -3, -2, -3, -2, -3, -2, -3, -2, -3,  
  -2, -3, -2, -3, -6, -6, 2, -4, -4, -4, -4, -4, -4,  
  -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,  
  -1, -6, -2, -2, -2, -12, -1, -4)
```

```
names <- c("date", "age", "county", "zip", "sex",  
"race", "ethnicity", "pdx", "disp")
```

```
sparcs<-read.fwf(file="/Users/charlie/Dropbox/  
rSlidesRev/sparcs/sparcs.txt", widths=positions,  
col.names=names, stringsAsFactors=F)
```

...see sparcsReadFile.R

approaches to reading in large data sets

- optimize read.csv()
 - *colClasses=* can reduce time by half...

```
firstRows <- read.csv("~mydat.csv", ..., nrows = 5)
myClasses <- sapply(firstRows, class)
fullDat<-read.csv("~mydat.csv", ..., colClasses=myClasses)
```
 - *nrows=* *comment.char=""* (if no comments)

```
system("wc ~.mydat.csv") # get row numbers
```
- sqlite and sqldf package
 - sqldf reads (via RSQLite) large file to temporary sqlite directory
 - then uses the uses an sql= statement to read the file to R
 - since first step imports data directly into SQLite without going through R, can handle larger files than R can
 - will work if the final file is a size R can handle
- ffbase, read.table packages
- python pandas library
 - sqlite via python to R

attaching data frames

not good practice

dataframe\$variable - members of data frame (or list) (preferred)

attaching - convenient *but not recommended*

- `attach()` puts data frame in *search path*
- `detach()` - takes data frame out of the workspace path
- `search()` - see what's in your path
- `with()` - better, data frame referenced in the first argument, expression as second argument, e.g. `with(dataframe, sum(var))`

attached variables are copies, operations don't change the original (to create new variable from attached variable, first use the `$` notation, then `detach()` and `attach()` to include the new variable in the search path)

reading files from other statistical programs

foreignpackage

- `read.xport()` reads SAS export format (don't need SAS)
- `read.ssd()` reads SAS file (need SAS on your computer)
 - creates an xport file, then invokes `read.xport`
- `write.foreign()` create SPSS, SAS and Stata files

best approach is (still) to save (or export) original as .csv and use `read.csv()`

under the hood

object classes

S3 classes introduced to S version 3 in 1992

- known as S3 or old-style classes, not formally defined
- usually lists with attributes, just grew over time e.g. dataframes, factors
- functions create classes: `lm()` for linear model creates objects of the class `lm`
- `getClass()` gives summary

S4 or new-style classes (since 1998)

- formally defined "slots" for components, faster to read and write, unlike old - style user can't redefine attributes
- test with `isS4()`, look inside with `slots()`
- e.g., `sp` package
 - foundation "spatial" class has 2 slots: bounding box (`bbox()`) and `proj4string()`
 - `SpatialPoints` class extends spatial class by adding slot for matrix of points
 - `SpatialPointsDataFrame()` class associates row of data with points (inherits all objects from `SpatialPoints` and `Spatial` classes upon which it is based)

Credit where credit is due...

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