Woods Hole

Manipulating data in R

Anders Nielsen

Outline

- Defining and manipulation vectors and matrices
- Reading data from text files
- Writing data to text files
- Working with binary files
- Merging
- Group summaries
- Subsetting

Editor and the function `source()`

- If you write programs spanning more than a few lines it is convenient to write them in an editor.
- Under windows it is simplest to use the build in editor in R or notepad, but they are lagging some neat features (parentheses matching, syntax highlighting, block clipping, ...), so eventually you may experience editor–envy and want to choose a better editor. There are many options (emacs, vi, WinEdt, Rstudio, UltraEdit, TinnR, ...)
- A frequently used approach is to write your code in and editor and then paste blocks into R to run it.
- Once the script is complete, the file is saved, and we can run it all by typing:

  ```
  > source("C:/programdir/script.R")
  ```

- Lines starting with "`#`" are ignored by R and can be used to insert comments in the script.
- Try it quickly.

Defining vectors and matrices

Defining vectors

- Integers from 9 to 17

```
> x<-9:17
> x
[1]  9 10 11 12 13 14 15 16 17
```

- A sequence of 11 numbers from 0 to 1

```
> y<-seq(0,1,length=11)
> y
 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

- The same number or the same vector several times

```
> z<-rep(1:2, 5)
> z
 [1] 1 2 1 2 1 2 1 2 1 2
```

- Combine numbers, vectors or both into a new vector

```
> xz10<-c(x,z,10)
> xz10
 [1]  9 10 11 12 13 14 15 16 17  1  2  1  2  1  2  1  2  1  2 10
```

Define matrices

- Combine rows into a matrix

```
> A<-rbind(1:3, c(1,1,2))
> A
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    1    1    2
```

- Or columns

```
> B<-cbind(1:3, c(1,1,2))
> B
     [,1] [,2]
[1,]    1    1
[2,]    2    1
[3,]    3    2
```

- Define a matrix from one long vector

```
> C<-matrix(c(1,0,0,1,1,0,1,1,1), nrow=3, ncol=3)
> C
     [,1] [,2] [,3]
[1,]    1    1    1
[2,]    0    1    1
[3,]    0    0    1
```

Can also be done by rows by adding "`, byrow=TRUE`" before lats parenthesis

Index and logical index

- Important for optimal use of **R**
- Example: Define a vector with integers from (-5) to 5 and extract the numbers with absolute value less than 3.

```
> x<- (-5):5
> x
 [1] -5 -4 -3 -2 -1  0  1  2  3  4  5
```

  - by their index in the vector

```
> x[4:8]
[1] -2 -1  0  1  2
```

  - by negative selection (set a minus in front of the indices we don't want)

```
> x[-c(1:3,9:11)]
[1] -2 -1  0  1  2
```

  - A logical vector can be defined by

```
> index<-abs(x)<3
> index
 [1] FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE
```

  - Now this vector can be used to extract the wanted numbers

```
> x[index]
[1] -2 -1  0  1  2
```

- This also works for matrices

```
> A<-matrix((-4):5, nrow=2, ncol=5)
> A
     [,1] [,2] [,3] [,4] [,5]
[1,]   -4   -2    0    2    4
[2,]   -3   -1    1    3    5
> A[A<0]
[1] -4 -3 -2 -1
```

- And for assignments

```
> A[A<0]<-0
> A
     [,1] [,2] [,3] [,4] [,5]
[1,]    0    0    0    2    4
[2,]    0    0    1    3    5
```

- Matrix rows can be selected by

```
> A[2,]
[1] 0 0 1 3 5
```

- and similarly for columns

```
> A[,c(2,4)]
     [,1] [,2]
[1,]    0    2
[2,]    0    3
```

Exercise 2.1

a) Define the matrix A and the vector y by:

$$A = \begin{pmatrix} 6 & 8 & 3 \\ 2 & 2 & 3 \\ 5 & 7 & 3 \end{pmatrix} \quad \text{and} \quad y = \begin{pmatrix} 4 \\ 3 \\ 5 \end{pmatrix}$$

Read a little about `solve()` in the documentation and solve the linear system $Ax = y$

b) Explain what goes on here:

```
> x<-matrix(1:12,4)
> x
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
> x[cbind(c(1,3,2),c(3,3,2))]
[1]  9 11  6
```

Reading data from text files

Read data from file

- Frequently data is collected in white space separated columns, where the first line indicate the variable name:

```
x1  x2    x3
 2 0.3 0.01
 2 1.0 0.11
 3 2.1 0.04
 3 2.2 0.02
 1 0.1 0.10
 1 0.2 0.06
```

- The function `read.table()` is designed to read this format

```
> mydat <- read.table("c:/datadir/filename.dat", header = TRUE)
```

- The data frame `mydat` now contains

```
> mydat
  x1  x2   x3
1  2 0.3 0.01
2  2 1.0 0.11
3  3 2.1 0.04
4  3 2.2 0.02
5  1 0.1 0.10
6  1 0.2 0.06
```

- The individual variables can be extracted via the $ operator, e.g:

```
> mydat$x3
[1] 0.01 0.11 0.04 0.02 0.10 0.06
```

- If the data frame is attached (`attach()`), then the variables can be accessed directly (remember to `detach()` when the data is no longer needed)

```
> attach(mydat)
> x.sum<-x1+x2
> x.sum
[1] 2.3 3.0 5.1 5.2 1.1 1.2
> detach(mydat)
```

- In a data frame rows, columns, and elements can be accessed like in a matrix.

```
> mydat[c(1,2), ]
  x1  x2   x3
1  2 0.3 0.01
2  2 1.0 0.11
```


- The `read.table()` function has a lot of optional arguments:

```
> args(read.table)
function (file, header = FALSE, sep = "", quote = "\"'", dec = ".",
    numerals = c("allow.loss", "warn.loss", "no.loss"), row.names,
    col.names, as.is = !stringsAsFactors, na.strings = "NA",
    colClasses = NA, nrows = -1, skip = 0, check.names = TRUE,
    fill = !blank.lines.skip, strip.white = FALSE, blank.lines.skip = TRU
    comment.char = "#", allowEscapes = FALSE, flush = FALSE,
    stringsAsFactors = default.stringsAsFactors(), fileEncoding = "",
    encoding = "unknown", text, skipNul = FALSE)
NULL
```

- Some of the important ones are:
  - `header` Is the first line variable names or not?
  - `sep` What character is used to separate the columns?
  - `dec` What character is used as decimal separator?
  - `nrows` How many rows do we want to read?
  - `na.strings` What string represent a missing value?
  - `skip` How many lines to skip before start reading?
  - `comment.char` What char in the beginning of a line should indicate that the line should be skipped?
  - `text` To be used if we wish to read from a string of input instead of from a file.

- Consider for instance the data file:

```
This file
has a bit of text

and an empty line
before the data
a b c
1 2 3
4 5 6
and then some more text at the end
```

```
> dat<-read.table("testdat1.dat", header=TRUE, skip=5, nrow=2)
> dat
  a b c
1 1 2 3
2 4 5 6
```

- Or the file:

```
A B C
1 2 3
4 3,2 2
1 5 .
; below this line is the extended data
5 4 6
```

```
> dat<-read.table("testdat2.dat", header=TRUE, na.strings=".",
+                 comment.char=";", dec=",")
> dat
  A   B  C
1 1 2.0  3
2 4 3.2  2
3 1 5.0 NA
4 5 4.0  6
```

- Other functions which are useful for reading data frames from files are:
  - `read.fwf()` fixed width format
  - `read.csv()` comma separate
- Additional arguments are similar to those of `read.table()`

Reading from more complicated files

- `scan()` Can be a little tricky to use, but is very flexible.
- Its simplest use is:

```
4.141593 5.141593 6.141593 7.141593 8.141593
```

```
> vec<-scan("scantest.txt")
> vec
[1] 4.141593 5.141593 6.141593 7.141593 8.141593
```

- `readLines()` Reads entire lines.

```
A B C
1.324654 2.324654 3.324654 4.324654 5.324654
How many roads
```

```
> vec<-readLines("readlinestest.txt")
> vec
[1] "A B C"
[2] "1.324654 2.324654 3.324654 4.324654 5.324654"
[3] "How many roads"
> as.numeric(strsplit(vec[2]," ")[[1]])
[1] 1.324654 2.324654 3.324654 4.324654 5.324654
```

File connections

- File connections can open a file for reading different sections in different ways. Consider:

```
> f1<-file("readlinestest.txt", open="r")
> scan(f1,what="",nlines=1)
[1] "A" "B" "C"
> scan(f1,what=double(),nlines=1)
[1] 1.324654 2.324654 3.324654 4.324654 5.324654
> scan(f1,what="",nlines=1)
[1] "How"   "many"  "roads"
> close(f1)
```

Exercise 2.2

a) Read the following data file into an R data frame:

```
Data created for R-course
result 1970-1980 result 1981-1990 result 1991-2000
              13                77                96
              15                97                91
              23                67                66
```

b) Read the following data into an R-list:

```
Data created for R-course
A
 1 2 3
B
8 8 6
7 5 5
5 7 6
9 6 9
```

Writing data to text files

Basic writing functions

cat()

```
> cat("Test file for cat\n",round(rnorm(5),3),"\n", file="cattest.txt")
```

```
Test file for cat
 -1.77 0.028 -0.348 2.04 -0.986
```

writeLines()

```
> lin<-c("Count down", paste(rev(1:10), collapse="-"), "Go")
> writeLines(lin, con="writelinestest.txt")
```

```
Count down
10-9-8-7-6-5-4-3-2-1
Go
```

write.table()

```
> mat<-matrix(round(rnorm(12),8), ncol=3)
> write.table(mat, file="writetabletest.txt", row.names=FALSE,
+             col.names=FALSE, sep=", ")
```

```
-0.9618385, -0.5525868, 0.96874699
0.48703314, 0.00209819, 0.1168733
1.5786884, -0.19872487, -1.22946711
1.20843349, -0.59592035, 1.94360476
```

In addition a few special ones

sink()

```
> sink("sinktest.txt")
>   x<-1:5
>   y<-1:3
>   outer(x,y)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    2    4    6
[3,]    3    6    9
[4,]    4    8   12
[5,]    5   10   15
> sink()
```

output not shown

dump()

```
> x<-1:3
> y<-rpois(10, 4)
> dump(c("x","y"), file="dumptest.txt")
```

```
x <-
1:3
y <-
c(3L, 4L, 3L, 2L, 6L, 5L, 4L, 2L, 4L, 5L)
```

dput()

```
> lis<-list(x=1:5, y=3, z=c("a","b","c"))
> dput(lis, file="dputtest.txt")
```

```
structure(list(x = 1:5, y = 3, z = c("a", "b", "c")), .Names = c("x",
"y", "z"))
```

Using file connections

```
> fo<-file("testout.txt", open="w")
>   cat("Header of file\n\n", file=fo)
>   mat<-matrix(round(rnorm(12),8), ncol=3)
>   write.table(mat, file=fo, row.names=FALSE, col.names=FALSE)
> close(fo)
```

```
Header of file

-0.33844101 0.16780281 0.38669457
0.84155108 0.65716514 0.17559155
-1.06555382 0.67072751 -0.0812511
-0.75221001 -1.08574223 0.65309674
```

Using append=TRUE

```
>   cat("Header of file\n\n", file="testappend.txt")
>   mat<-matrix(round(rnorm(12),8), ncol=3)
>   write.table(mat, file="testappend.txt", row.names=FALSE,
+               col.names=FALSE, append=TRUE)
```

```
Header of file

-0.26134779 0.86400299 -2.94553714
-2.29265841 -0.65174258 0.47716661
0.07957215 -2.62986158 0.52977277
0.52673329 -0.38651126 1.81745344
```

Exercise 2.3

a) Make R write the following data file:

```
"a";"b"
"A";1
"B";2
"C";3
"D";4
"E";5
```

b) Write the following data file

```
TITLE extra line
2 3 5 7

11 13 17
One more line
```

Working with binary files

Writing a binary file

- Binary files take up less space than ordinary text files, so if we have a large amounts of data to store it is preferable.

```
> N<-10000
> x<-rnorm(N)
> sum(x)
[1] -43.02195
> f1<-file("binfile.bin", open="wb")
>   writeBin(as.integer(N),f1)
>   writeBin(x,f1)
> close(f1)
```

- Notice I wrapped N to ensure it became an integer.

Reading a binary file

- To read it we have to know what is in there.

```
> rm(N,x)  # just making sure it is not there already
> f1<-file("binfile.bin", open="rb")
>   N<-readBin(f1,integer(),1)
>   x<-readBin(f1,numeric(),N)
> close(f1)
> sum(x)
[1] -43.02195
```

- Binary files created in this way works also outside **R**.

Using save and load

- R also has its own internal binary format
- To save data and functions to it use e.g:

```
> x<-rnorm(3)
> lis<-list(y=1:5, z="lalala", fun=function()cat("ha-ha-ha\n"))
> save(x,lis, file="test1.RData")
```

- To read back into R simple use:

```
> rm(list=ls())
> load(file="test1.RData")
> ls()
[1] "lis" "x"
```

- This format is much simpler to use, but only works within **R**.

Exercise 2.4

a) Save a vector of numbers, a text string and two integers to general binary file.

b) Read the objects saved under a) back into **R**.

c) Repeat a) and b) using `save()` and `load()`

Merging data

Merging and such

- It is simple to add a column to a data frame:

```
>   dat<-data.frame(x=LETTERS[1:3], y=1:3)
>   dat$z<-dat$y^2
>   dat$name<-c("Cat", "Vic", "Osc")
>   dat
```

```
  x y z name
1 A 1 1  Cat
2 B 2 4  Vic
3 C 3 9  Osc
```

- If we have two data sets:

```
>   dat1
  name age
1  Cat   9
2  Vic   7
3  Osc   4
>   dat2
  names gender
1   Cat Female
2   Vic   Male
3   Osc   Male
```

- Then we can merge that information into one data set by:

```
> dat<-merge(dat1,dat2, by.x="name", by.y="names")
> dat
  name age gender
1  Cat   9 Female
2  Osc   4   Male
3  Vic   7   Male
```


Exercise 2.5

a) Make a suitable third data frame and merge it with this one.

Group summaries

Apply function within a group

- Consider the following data frame

```
> dat
```

	gender	height
1	Male	10
2	Male	5
3	Male	12
4	Male	10
5	Male	2
6	Female	7
7	Female	6
8	Female	12
9	Female	9
10	Female	4

- A couple of ways to calculate group means:

```
> tapply(dat$height, dat$gender, mean)
  Male Female
   7.8    7.6
> aggregate(height~gender, data=dat, mean)
  gender height
1   Male    7.8
2 Female    7.6
> by(dat$height, dat$gender, mean)
dat$gender: Male
[1] 7.8
------------------------------------------------
dat$gender: Female
[1] 7.6
```

- Now consider:

```
> dat2
   gender     tmt height
1    Male  active     10
2    Male placebo      5
3    Male  active     12
4    Male placebo     10
5    Male  active      2
6  Female placebo      7
7  Female  active      6
8  Female placebo     12
9  Female  active      9
10 Female placebo      4
> tapply(dat2$height, list(dat2$gender,dat2$tmt), mean)
       active  placebo
Male      8.0 7.500000
Female    7.5 7.666667
> aggregate(height~gender+tmt, data=dat2, mean)
  gender     tmt   height
1   Male  active 8.000000
```

```
2 Female  active 7.500000
3   Male placebo 7.500000
4 Female placebo 7.666667
> by(dat2$height, list(dat2$gender, dat2$tmt), mean)
: Male
: active
[1] 8
------------------------------------------------------------
: Female
: active
[1] 7.5
------------------------------------------------------------
: Male
: placebo
[1] 7.5
------------------------------------------------------------
: Female
: placebo
[1] 7.666667
```

Exercise 2.6

a) Try calculating something other than the mean (e.g. median, standard deviation, or the sum)

b) What would we do if we wanted to divide into groups defined by something, which is not a factor? Hint: study the `cut()` function.

Subsets of data

The subset function

- All the logical indexing stuff also applies to data frames, so e.g:

```
> datA<-airquality[airquality$Temp>80,c("Ozone","Temp")]
```

- But a neat function is built in for making subsets of data

```
> datA<-subset(airquality, Temp > 80, select = c(Ozone, Temp))
> datB<-subset(airquality, Day == 1, select = -Temp)
> datC<-subset(airquality, select = Ozone:Wind)
```

Exercise 2.7

a) In the data set:

```
> dat2
```

	gender	tmt	height
1	Male	active	10
2	Male	placebo	5
3	Male	active	12
4	Male	placebo	10
5	Male	active	2
6	Female	placebo	7
7	Female	active	6
8	Female	placebo	12
9	Female	active	9
10	Female	placebo	4

Make a sub-set of the active males with height greater than 10. Hint Remember '&' means logical and.

Exercise 2.X

a) Use the data set 'CO2' which is built into **R**make a sub data frame, which contains only data of type 'Mississippi'

b) Find the average uptake per concentration (conc) for each combination of plant and treatment.

Appendix: Database

```
# RODBC Example
# import 2 tables (Crime and Punishment) from a DBMS
# into R data frames (and call them crimedat and pundat)

library(RODBC)
myconn <-odbcConnect("mydsn", uid="Rob", pwd="aardvark")
crimedat <- sqlFetch(myconn, Crime)
pundat <- sqlQuery(myconn, "select * from Punishment")
close(myconn)
```

http://www.statmethods.net/input/dbinterface.html