

review of lm()

Wilkinson-Rogers notation

result ~ predictor

result ~ nuisance + predictor

result must be numeric

predictors can be numeric or factor

huge difference!

aov() is a wrapper for lm()

more anova-like summary

calls lm() on each 'stratum'

different behaviour with ordered factors

other model fitting possibilities

glm : generalised linear modelling

allows different error distributions (binomial, poisson..)

lme : linear mixed effects (lm estimates fixed effects)

nlme

repetitive operations: scripts

you can save a series of commands in a text file

run it whenever you like with source()

practical problem: the environment may change

data available may be different

commands in file may clobber your variables

need ways of managing data going in and coming back

apply()

key to processing data by row or column

apply(data,1,mean) # row means

apply(data,2,mean) # column means

min,max, SD, which.min, summary ...

but what if it's a bit more complicated?

introduction to functions

a data type called 'function'

has 'arguments' (list of data to go in)

has a 'body' (some code)

the function gets a name by assignment

you run it like this : name()

writing your own functions

many of the R functions we've used were written in R

you see the source if you forget the ()

many functions assign variables

why doesn't your environment fill up?

why don't your variables get clobbered?

variable scope and environments

there is not just one environment

each function has its own environment

variables assigned in a function are local

their 'scope' is limited to the function

called 'lexical scope' because defined by simple text rule

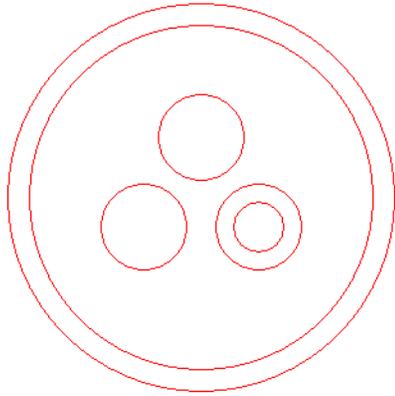


you can see the enclosing environment (inheritance)

almost always better to pass in arguments

last thing evaluated in a function is the return value

can be any data type



arguments: data to be processed

either they are required or they have a default value

the first one is usually the data

subsequent arguments are often options

identified by name or order

```
myfirstfunction <- function (x){x-1}
```

name of function: myfirstfunction

arguments: requires one piece of data

not necessarily called x

return value: decremented x

```
x<-100;myfirstfunction(x)
```

x is still 100

myfirstfunction returns 99

```

review of lm()
  Wilkinson-Rogers notation
    result ~ predictor
    result ~ nuisance + predictor
  result must be numeric
  predictors can be numeric or factor
    huge difference!
  aov() is a wrapper for lm()
    more anova-like summary
    calls lm() on each 'stratum'
    different behaviour with ordered factors
other model fitting possibilities
  glm : generalised linear modelling
    allows different error distributions (binomial, poisson..)
  lme : linear mixed effects (lm estimates fixed effects)
  nlme
repetitive operations: scripts
  you can save a series of commands in a text file
  run it whenever you like with source()
  practical problem: the environment may change
    data available may be different
    commands in file may clobber your variables
  need ways of managing data going in and coming back
apply()
  key to processing data by row or column
  apply(data,1,mean) # row means
  apply(data,2,mean) # column means
  min,max, SD, which.min, summary ...
  but what if it's a bit more complicated?
introduction to functions
  a data type called 'function'
    has 'arguments' (list of data to go in)
    has a 'body' (some code)
  the function gets a name by assignment
  you run it like this : name()
writing your own functions
  many of the R functions we've used were written in R
  you see the source if you forget the ()
  many functions assign variables
    why doesn't your environment fill up?
    why don't your variables get clobbered?
variable scope and environments
  there is not just one environment
  each function has its own environment
  variables assigned in a function are local
    their 'scope' is limited to the function
    called 'lexical scope' because defined by simple text rule
*onion.html
*envir.html
you can see the enclosing environment (inheritance)
  almost always better to pass in arguments
  last thing evaluated in a function is the return value
  can be any data type
arguments: data to be processed
  either they are required or they have a default value
  the first one is usually the data
  subsequent arguments are often options
  identified by name or order
myfirstfunction <- function (x){x-1}
  name of function: myfirstfunction

```

```

arguments: requires one piece of data
  not necessarily called x
return value: decremented x
x<-100;myfirstfunction(x)
  x is still 100
myfirstfunction returns 99

```