

So You Want to Write Your Own R Package?

Olivia Lau

olivia.lau@post.harvard.edu

Graduate Methods and Models Class
Harvard University

November 2, 2007

Code

Functions

Writing an R package

Sharing your R package with the world

Writing good code (1)

Remember, objects have dimensions

- Vectors: `x[4]`
What happens if `x[-(3:length(x))]`?
- Matrices: `y[,4]`
What if `y[4:ncol(y)]`?
- Arrays: `z[, ,3]`, `z[1, ,3]`
- Lists: `L[[2]]`
What about `L[-4]`?

Writing good code (2)

Choose names wisely

- For objects

```
wert <- solve(t(X) %*% X) * t(X) %*% Y  
beta <- solve(t(X) %*% X) * t(X) %*% Y
```

Why not beta.hat?

- Use names attributes *within* data objects

```
tt <- c(1, 2, 3, 4, 5)  
names(tt) <- c("a", "b", "c", "d", "e")  
tt["c"] ## = 3  
tt["row", "column"]  
tt["x", "y", "z"]  
tt[["apples"]]
```

Writing good code (3)

Loops are not always slow, but usually

```
out <- NULL
for (i in 1:1000)
  out <- cbind(out, rnorm(500))          12.2s

out <- matrix(NA, nrow=500, ncol=1000)
for (i in 1:1000)
  out[,i] <- rnorm(500)                0.16s

out <- replicate(1000, rnorm(500))     0.21s
```

How to avoid loops (1)

For matrices and arrays, use

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

```
out1 <- NULL
for (i in 1:1000)
  out1[i] <- mean(out[,i], na.rm = TRUE)

out1 <- apply(out, 2, mean, na.rm = TRUE)

x <- array(rnorm(3*10*100),
           dim = c(3, 10, 100))
out3 <- apply(x, c(1,2), mean)
```

How to avoid loops (2)

For lists, use

```
sapply(X, FUN, ...)    or  
lapply(X, FUN, ...)
```

```
out2 <- list(x1, x2, x3)  
sapply(out2, sd, na.rm = TRUE)  
lapply(out2, '%*%', beta)
```

Code \rightarrow functions?

If you can write R code, you can write functions!

```
fn <- function(x, y, ...) {  
  ## Comment your code  
  y1 <- subfn(y, ...)  
  x * y1  
}
```

Things to remember

- Pass all arguments explicitly

```
fun <- function(x) {  
  x * y  
}
```

- Declare output in the last line

```
fun <- function(x) {  
  out <- x * y  
  out  
}
```

- If you need to return two outputs, use a list

The mysterious . . .

. . . are very powerful!

Let's say you have

```
subfn <- function(x, y = 1) {  
  # if(missing(y)) y <- 1  
  x * y  
}
```

```
fn <- (x, ...) {  
  subfn(x, ...)  
}
```

What happens if the user enters

```
fn(x = 3, y = 2) ?
```

```
fn(x = 3) ?
```

Make your code easy to use

(Apply Occam's Razor early and often)

- Streamline inputs
- Use formula and data

```
D <- model.frame(formula, data = data)
X <- model.matrix(formula, data = D)
Y <- model.response(D)
```
- Specify defaults
- Simplify outputs
 - Use names on lists, arrays, matrices, vectors
 - *Never* nest lists

Now what?

- Select a working directory using `setwd()`
- Type `package.skeleton("mypackage")`
- You should have a new directory `mypackage`

```
data DESCRIPTION man
```

- Create

- An `R` directory for R functions
- A `src` directory for C/C++/Fortran code
- A `demo` directory for sourceable demos
- An empty `NAMESPACE` file

```
data DESCRIPTION demo
man NAMESPACE R src
```

DESCRIPTION file

```
Package: mypackage
Type: Package
Title: What the package does
Version: 1.0
Date: 2007-11-02
Author: Who wrote it
Maintainer: <yourfault somewhere.net>
Description: More about what it does
License: What license is it under?
```

Classes, methods, and generic functions

- Classes

- All R objects have a *class*
- Programmers can define new classes

```
class(object) <- "xxx"
```

- Generic functions

```
summary <- function (object, ...)
```

```
  UseMethod("summary")
```

- Methods

```
summary.xxx <- function (object, k, ...) {
```

```
  ## Do stuff
```

```
  class(out) <- "yyy"
```

```
  out
```

```
}
```

- (Never use `.` in a regular function name)

The NAMESPACE file

Controls interaction between R and user

- Register methods for generic functions
`S3method(summary, xxx)`
- Make functions from package available to users
`export(myfunction)`
- Make functions from other packages behave like a part of your package
`importFrom(MASS, mvrnorm)`
- Load C/C++ shared libraries
`useDynLib(mypackage)`

Sample NAMESPACE file

```
useDynLib(eiPack)

importFrom(MASS, rgamma)

export( bounds,
        eiReg,
)

S3method(summary, eiReg)
S3method(print, eiReg)
```

Sample demo file

```
## Load sample data from data directory
data(mydata)

out <- eiReg(y ~ x, data = mydata)
summary(out)
print(out)
```

Documentation

- Easy-to-use code is easy to document
- Write R-help (*.Rd) files that *you* would want to use
- If the help file is too complicated, your code is probably too complicated

Sharing your package

From the terminal prompt, from the directory that contains the directory `mypackage`,

- R CMD INSTALL mypackage
- R CMD check mypackage
- Fix all warnings and errors
- R CMD build mypackage
- Upload your package to the CRAN ftp site

- Writing good, parsimonious code is really hard
- Writing reliable functions is hard
- Writing a new R package is easy