Building R packages

Stat 580: Statistical Computing

• Theme: Black - White

• Printable version

References

- "Writing R Extensions", by R Core Team.
- "R packages", by Hadley Wickham.

Before the hard work

- Why building a package?
 - own convenience (e.g. especially useful for R's C interface)
 - for everybody
 - will it be an important package?
 - what is the target group of users?
 - do some marketing (R CRAN)

Public packages

- Planning for a public package:
 - what is the good subset of functionality you should provide for the initial version?
 - o stable?
 - more functionality in the future?
 - are they going to affect the structure and functions in earlier versions?
 - design of complex output object
 - what should be included in the object
 - utility functions of the object
 - for example, lm() returns an object of class lm
 - S3 classes and methods can be useful

• Using S3 classes allows simpler and more intuitive function names:

```
matstat <- function(X){</pre>
  cmean <- colMeans(X)</pre>
 rmean <- rowMeans(X)</pre>
  return(list(cmean=cmean, rmean=rmean, nr=nrow(X), nc=ncol(X)))
plotmatstat <- function(obj, type=0){</pre>
  if (type==0){
    plot(obj$rmean, xlab="rows", ylab="row means")
  } else if (type==1){
    plot(obj$cmean, xlab="cols", ylab="col means")
  } else {
    stop("type not equal to 0 or 1")
a <- matrix(runif(10*20), nr=10)</pre>
res <- matstat(a)</pre>
plotmatstat(res)
```

• Using S3 classes allows simpler and more intuitive function name:

```
matstat <- function(X){</pre>
  cmean <- colMeans(X)</pre>
 rmean <- rowMeans(X)</pre>
  out <- list(cmean=cmean, rmean=rmean, nr=nrow(X), nc=ncol(X))</pre>
  class(out) <- "mstat"</pre>
  return(out)
plot.mstat <- function(obj, type=0){</pre>
  if (type==0){
    plot(obj$rmean, xlab="rows", ylab="row means")
  } else if (type==1){
    plot(obj$cmean, xlab="cols", ylab="col means")
  } else {
    stop("type not equal to 0 or 1")
a <- matrix(runif(10*20), nr=10)</pre>
res <- matstat(a)</pre>
plot(res)
```

- example of common methods: print, summary, plot, predict
 - create generic function via UseMethod()
 - use methods to list all available methods for an generic function, or all methods for a class
 - o methods(plot), methods(class="mstat")

```
print.mstat <- function(x){
  cat("dims:", x$nr, x$nc, "\n")
}</pre>
```

- common strategy:
 - at the end of the function, create a list
 - assign a class via class()
 - write corresponding methods

- assign an existing class to a new object manually can create problems, since the structure may not match
 - creates errors during the applications of some methods

Example

```
ascomplex <- function(x, y){</pre>
 out <- list(real=x, img=y)</pre>
 class(out) <- "com"</pre>
 return(out)
print.com <- function(x) {</pre>
  cat(paste(c(x$real, "+", x$img, "i\n"), collapse=""))
`+.com` <- function(x, y) {
 r <- x$real + y$real
 c <- x$img + y$img
 return(ascomplex(r, c))
`-.com` <- function(x, y) {
 r <- x$real - y$real
 c <- x$img - y$img
 return(ascomplex(r, c))
`*.com` <- function(x, y){
 r <- x$real * y$real - x$img * y$img
 c <- x$real * y$real + x$img * y$real
 return(ascomplex(r, c))
```

Building R packages

- 1. Create a template package via package.skeleton()
 - supply a name: package.skeleton("mypackage")
 - this creates a folder with correct structure
- 2. Put your codes into the right folder
- 3. Edit the package files (DESCRIPTION, NAMESPACE)
- 4. Edit help files (we won't cover this, check "writing R Extensions")
- 5. Build, check and install the package

package.skeleton()

- create a folder with
 - "data": contains .rda files of each data object
 - "DESCRIPTION": general information
 - "man": help files
 - "NAMESPACE": contain "exports" and "imports" information
 - "R": contains .R files
 - "Read-and-delete-me": check and delete

DESCRIPTION

- update the information
- license: popular choices include GPL-3 or GPL (>= 2)
- if the package depends on other packages:
 - add: "Imports" or "Depends:" followed by a list of the packages separated by comma
 - "Imports": load these packages
 - "Depends": attach these packages

NAMESPACE

- to export:
 - export(): export functions (including S3 and S4 generics).
 - exportPattern(): export all functions that match a pattern.
 - exportClasses(), exportMethods(): export S4 classes and methods.
 - S3method(): export S3 methods.
- to import:
 - import(): import all functions from a package.
 - importFrom(): import selected functions (including S4 generics).
 - importClassesFrom(), importMethodsFrom(): import S4 classes and methods.
 - useDynLib(): import a function from C.

Build, check, and install

in command line:

build

```
R CMD build mypackage
```

- generate a .tar.gz file ("tarball")
- install

```
R CMD install mypackage
```

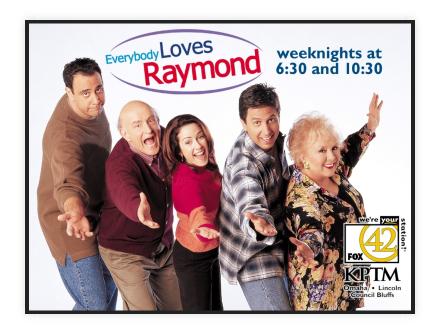
- either the folder or the tarball file
- check

```
R CMD check mypackage
```

need checking before submitting to CRAN

The end

As an old saying goes,



I hope you enjoy this course!

I am looking forward to your presentations! Thanks!