R Programming

Dr Heather Turner
Freelance/Department of Statistics, University of Warwick, UK

2 October 2018

Using Functions From Other Packages

In our own functions (outside of packages), it is possible to use `library`

```
scale_rows <- function(X){
    library(matrixStats)
    X <- X - rowMeans(X)
    X/rowSds(X)
}
```

But this loads the entire package, potentially leading to clashes with functions from other packages. It is better to use the **import** package:

```
scale_rows <- function(X){
    import::from(matrixStats, rowSds)
    X <- X - rowMeans(X)
    X/rowSds(X)
}
scale_rows(matrix(1:12, nrow = 3))
```

Custom ggplot

ggplot2, like **dplyr** and other tidyverse packages, uses *non-standard evaluation*, that is, it refers to variable names as if they were objects in the current environment

```
ggplot(mtcars, aes(x = mpg, y= disp)) +
    geom_point()
```

To emulate this, we have to use tools from **rlang**: `enquo` then `!!`

```
ggscatter <- function(data, x, y){
    import::from(rlang, enquo, `!!`)
    import::from(ggplot2, ggplot, aes, geom_point)
    nse_x <- enquo(x)
    nse_y <- enquo(y)
    ggplot(data, aes(x = !! nse_x, y = !! nse_y)) +
        geom_point()
}
ggscatter(mtcars, x = mpg, y = disp)
```

Externalizing Function Code

It is a good idea to separate function code from analysis code.

Put related functions together and source as required

```
source("modelFunctions.R")
source("plotFunctions.R")
```

The **import** package enables only necessary, top-level functions to be imported to the global workspace:

```
import::here(poissonModel, quasiPoissonModel, .from = "modelFunctions.R")
```

In either case, `import::from` commands can be put outside the function body to make the code easier to read.

Documenting Functions

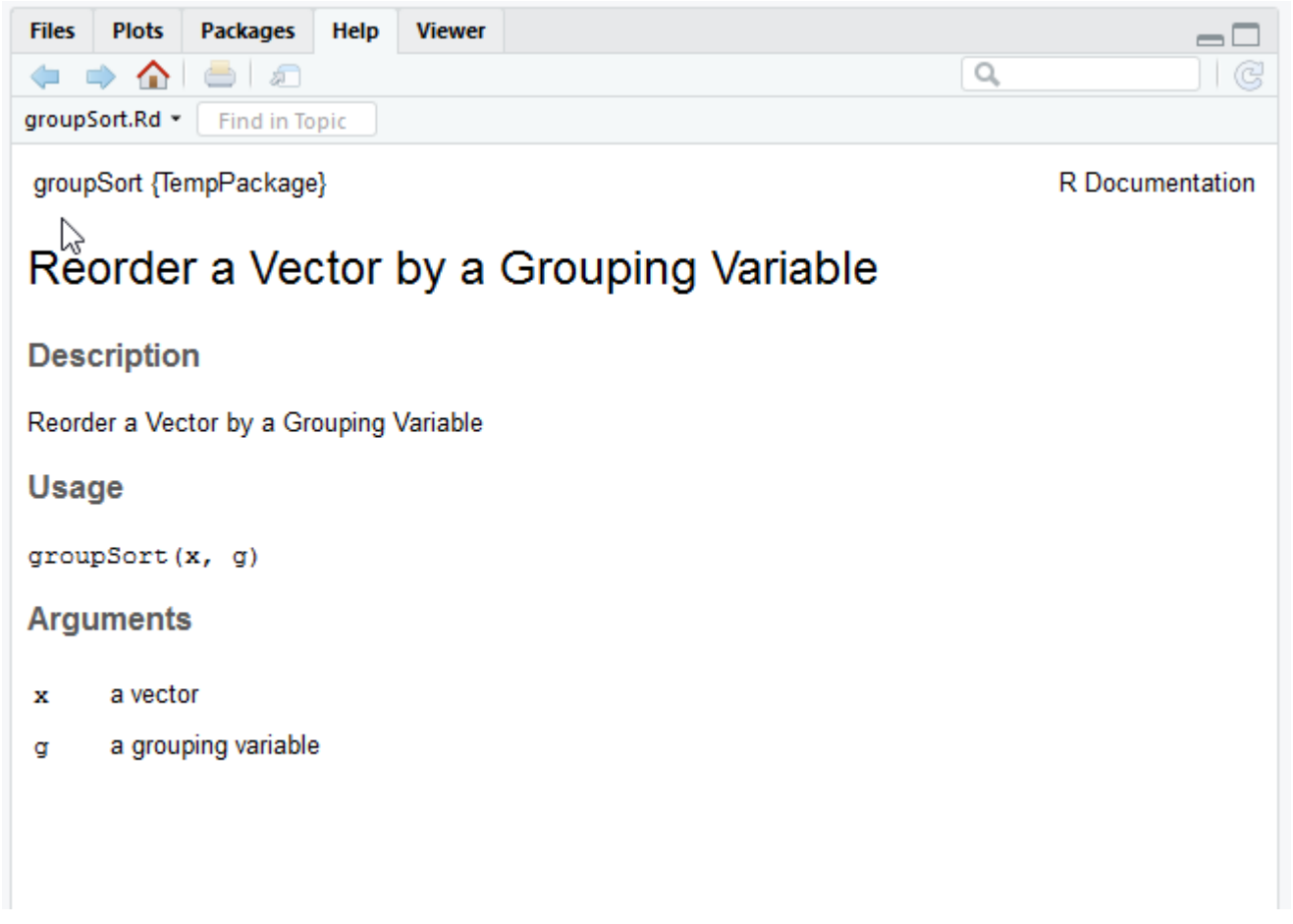
Comments help to record what a function does

```
# reorder x by grouping variable g
groupSort <- function(x, g) {
    ord <- order(g) #indices for ascending order of g
    x[ord]
}
```

The **docstring** package enables *roxygen* comments to be turned into a help file

```
library(docstring)
groupSort <- function(x, g) {
    #' Reorder a Vector by a Grouping Variable
    #'
    #' @param x a vector
    #' @param g a grouping variable
    ord <- order(g) #indices for ascending order of g
    x[ord]
}
```

```
?groupSort
```



For fuller documentation, see the **docstring** vignette.

Validation

When developing a function, we will want to validate its output.

A simple approach is to try different inputs

```
log_2 <- function(x){
    log(x, 2)
}
log_2(2^2)
```

```
# [1] 2
```

```
log_2(2^0)
```

```
# [1] 0
```

Doing this each time we change the function becomes tedious to check and error-prone as we miss important tests.

Unit testing

The **testthat** packages allows us to create a test suite:

```
context("log_2 works correctly")

test_that("log_2 returns log to base 2", {
    expect_equal(log_2(2^3), 3)
    expect_equal(log_2(2^0), 0)
})

test_that("negative values give error", {
    expect_error(log_2(2^-1))
})
```

Running Tests

If we save the tests in a file, e.g. `tests.R`, we can use `test_file()` to run and check all tests:

```
library(testthat)
test_file("tests.R")
```

```
# v | OK F W S | Context
# x |  2 1     | log_2 works correctly
# ----------------------------------------------------------------------
# tests.R:9: failure: negative values give error
# `log_2(2^-1)` did not throw an error.
# ----------------------------------------------------------------------
#
# == Results =============================================================
# OK:       2
# Failed:   1
# Warnings: 0
# Skipped:  0
```

Sanity Checks

To avoid mistakes, you may want to add some basic sanity checks

```
logit <- function(p){
    stopifnot(p > 0 & p < 1)
    log(p/(1 - p))
}
logit(2)
```

```
# Error in logit(2): p > 0 & p < 1 is not TRUE
```

```
logit(0.5)
```

```
# [1] 0
```

Error Messages

Often the R messages can be quite obscure

```
zap <- function(x) if (max(x) < 1e7) 0 else x
x <- c(1, 2, NA)
zap(x)
# Error in if (max(x) < 1e+07) 0 else x: missing value where TRUE/FALSE needed
```

More helpful error message can be implemented using `stop`

```
zap <- function(x) {
    if (any(is.na(x))) stop("missing values in x\nare",
                            " not allowed")
    if (max(x) < 1e7) 0 else x
}
zap(x)
# Error in zap(x): missing values in x
# are not allowed
```

Warning Messages

Warning messages should be given using `warning()`

```
safe_log2 <- function(x) {
    if (any(x == 0)) {
        x[x == 0] <- 0.1
        warning("zeros replaced by 0.1")
    }
    log(x, 2)
}
safe_log2(0:1)
```

```
# Warning in safe_log2(0:1): zeros replaced by 0.1
# [1] -3.322  0.000
```

Other messages can be printed using `message()`.

Suppressing Warnings

If a warning is expected, you may wish to suppress it

```
log(c(3, -1))
# Warning in log(c(3, -1)): NaNs produced
# [1] 1.099   NaN
x <- suppressWarnings(log(c(3, -1)))
```

All warnings will be suppressed however!

Similarly `suppressMessages()` will suppress messages.

Writing an R Package

If using functions across many projects, or you want to share your functions with the wider world, it's best to put those functions in a package.

A package is built from the package source, which is a directory of the function code, tests, etc organised with a particular structure.

The **usethis** package helps to create the right structure and add components the the package, e.g. with `create_package()` and `use_tests()`.

The **devtools** package helps to develop the package, e.g. with `load_all()` to load the functions as if the package were installed and `document()` to create helpfiles from the roxygen comments.

Package vs Stand-alone Function

	Package	Standalone function
Function code	with related function code in R/	anywhere in any .R file
roxygen comments	above function definition	in function body
Imports	roxygen comments	in function .R (import::from)
Exports	roxygen comments	in analysis .R (import::here)
testthat tests	in tests/testthat/	in separate .R file
Long-form docs	.Rmd in vignettes/	-
Shared data	file in data/, roxygen in R/	-
Package metadata	DESCRIPTION file	-
Package intro	README.md	-
Package news	NEWS.md	-