Five Favourite Functions in R

Cole Beck

October 4, 2012

1 Introduction

This is a quick tutorial on five of my favourite R functions. They often come in handy and should definitely save you valuable time.

2 capture.output

Sometimes a function prints output that you would like to collect. Usually you can save the return value of the function call and collect the data in a variable. Occasionally you'll have to dig within the attributes of the returned value to find what you want. There may even be a different function you should call to grab this data. But on the rare occasion that none of these techniques work, then what? Maybe you're familiar with the sink function. It takes all output and sends it to a connection/file. You could use this to create a file with your output, and then scan it to read it back into R. Sink's sister function, capture.output, however, may be the function you desire. It simply saves your output to a character string.

```
library(lme4)

## Loading required package:  Matrix

## Loading required package:  lattice

## Attaching package:  'lme4'

## The following object(s) are masked from 'package:stats':
##
## AIC, BIC

fm1 <- lmer(Reaction ~ Days + (Days | Subject), sleepstudy)
fm1

## Linear mixed model fit by REML
## Formula: Reaction ~ Days + (Days | Subject)
##    Data: sleepstudy
##   AIC  BIC logLik deviance REMLdev
##  1756 1775   -872     1752    1744
## Random effects:
##  Groups   Name        Variance Std.Dev. Corr
##  Subject  (Intercept) 612.1    24.74
##           Days         35.1     5.92    0.066
##  Residual             654.9    25.59
## Number of obs: 180, groups: Subject, 18
##
## Fixed effects:
```

```
##             Estimate Std. Error t value
## (Intercept)   251.41       6.82    36.8
## Days           10.47       1.55     6.8
##
## Correlation of Fixed Effects:
##      (Intr)
## Days -0.138

str(fm1)

## Formal class 'mer' [package "lme4"] with 34 slots
##   ..@ env     :<environment: 0x5ca4750>
##   ..@ nlmodel : language I(x)
##   ..@ frame   :'data.frame': 180 obs. of  3 variables:
##   .. ..$ Reaction: num [1:180] 250 259 251 321 357 ...
##   .. ..$ Days    : num [1:180] 0 1 2 3 4 5 6 7 8 9 ...
##   .. ..$ Subject : Factor w/ 18 levels "308","309","310",..: 1 1 1 1 1 1 1 1 1 1 ...
##   .. ..- attr(*, "terms")=Classes 'terms', 'formula' length 3 Reaction ~ Days
##   .. .. .. .. ..- attr(*, "variables")= language list(Reaction, Days)
##   .. .. .. .. ..- attr(*, "factors")= int [1:2, 1] 0 1
##   .. .. .. .. .. ..- attr(*, "dimnames")=List of 2
##   .. .. .. .. .. .. ..$ : chr [1:2] "Reaction" "Days"
##   .. .. .. .. .. .. ..$ : chr "Days"
##   .. .. .. ..- attr(*, "term.labels")= chr "Days"
##   .. .. .. ..- attr(*, "order")= int 1
##   .. .. .. ..- attr(*, "intercept")= int 1
##   .. .. .. ..- attr(*, "response")= int 1
##   .. .. .. ..- attr(*, ".Environment")=<environment: 0x2d046a8>
##   .. .. .. ..- attr(*, "predvars")= language list(Reaction, Days)
##   .. .. .. ..- attr(*, "dataClasses")= Named chr [1:2] "numeric" "numeric"
##   .. .. .. .. ..- attr(*, "names")= chr [1:2] "Reaction" "Days"
##   ..@ call    : language lmer(formula = Reaction ~ Days + (Days | Subject), data = sleepstudy)
##   ..@ flist   :'data.frame': 180 obs. of  1 variable:
##   .. ..$ Subject: Factor w/ 18 levels "308","309","310",..: 1 1 1 1 1 1 1 1 1 1 ...
##   .. ..- attr(*, "assign")= int 1
##   ..@ X       : num [1:180, 1:2] 1 1 1 1 1 1 1 1 1 1 ...
##   .. ..- attr(*, "assign")= int [1:2] 0 1
##   ..@ Xst     :Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
##   .. .. ..@ i       : int(0)
##   .. .. ..@ p       : int 0
##   .. .. ..@ Dim     : int [1:2] 0 0
##   .. .. ..@ Dimnames:List of 2
##   .. .. .. ..$ : NULL
##   .. .. .. ..$ : NULL
##   .. .. ..@ x       : num(0)
##   .. .. ..@ factors : list()
##   ..@ Zt      :Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
##   .. .. ..@ i       : int [1:360] 0 18 0 18 0 18 0 18 0 18 ...
##   .. .. ..@ p       : int [1:181] 0 2 4 6 8 10 12 14 16 18 ...
##   .. .. ..@ Dim     : int [1:2] 36 180
##   .. .. ..@ Dimnames:List of 2
##   .. .. .. ..$ : NULL
##   .. .. .. ..$ : NULL
##   .. .. ..@ x       : num [1:360] 1 0 1 1 1 2 1 3 1 4 ...
```

```
##   .. .. ..@ factors : list()
##   ..@ pWt     : num(0)
##   ..@ offset  : num(0)
##   ..@ y       : num [1:180] 250 259 251 321 357 ...
##   ..@ Gp      : int [1:2] 0 36
##   ..@ dims    : Named int [1:18] 1 180 2 36 1 3 1 1 2 5 ...
##   .. ..- attr(*, "names")= chr [1:18] "nt" "n" "p" "q" ...
##   ..@ ST      :List of 1
##   .. ..$ : num [1:2, 1:2] 0.9667 0.0157 0 0.2309
##   .. .. ..- attr(*, "dimnames")=List of 2
##   .. .. .. ..$ : chr [1:2] "(Intercept)" "Days"
##   .. .. .. ..$ : chr [1:2] "(Intercept)" "Days"
##   ..@ V       : num[0 , 0 ]
##   ..@ A       :Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
##   .. .. ..@ i       : int [1:360] 0 18 0 18 0 18 0 18 0 18 ...
##   .. .. ..@ p       : int [1:181] 0 2 4 6 8 10 12 14 16 18 ...
##   .. .. ..@ Dim     : int [1:2] 36 180
##   .. .. ..@ Dimnames:List of 2
##   .. .. .. ..$ : chr [1:36] "308" "309" "310" "330" ...
##   .. .. .. ..$ : NULL
##   .. .. ..@ x       : num [1:360] 0.967 0 0.982 0.231 0.997 ...
##   .. .. ..@ factors : list()
##   ..@ Cm      :Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
##   .. .. ..@ i       : int(0)
##   .. .. ..@ p       : int 0
##   .. .. ..@ Dim     : int [1:2] 0 0
##   .. .. ..@ Dimnames:List of 2
##   .. .. .. ..$ : NULL
##   .. .. .. ..$ : NULL
##   .. .. ..@ x       : num(0)
##   .. .. ..@ factors : list()
##   ..@ Cx      : num(0)
##   ..@ L       :Formal class 'dCHMsimpl' [package "Matrix"] with 10 slots
##   .. .. ..@ x       : num [1:54] 3.42 3.22 2.41 3.42 3.22 ...
##   .. .. ..@ p       : int [1:37] 0 2 3 5 6 8 9 11 12 14 ...
##   .. .. ..@ i       : int [1:54] 0 1 1 2 3 3 4 5 5 6 ...
##   .. .. ..@ nz      : int [1:36] 2 1 2 1 2 1 2 1 2 1 ...
##   .. .. ..@ nxt     : int [1:38] 1 2 3 4 5 6 7 8 9 10 ...
##   .. .. ..@ prv     : int [1:38] 37 0 1 2 3 4 5 6 7 8 ...
##   .. .. ..@ colcount: int [1:36] 2 1 2 1 2 1 2 1 2 1 ...
##   .. .. ..@ perm    : int [1:36] 0 18 1 19 2 20 3 21 4 22 ...
##   .. .. ..@ type    : int [1:4] 2 1 0 1
##   .. .. ..@ Dim     : int [1:2] 36 36
##   ..@ deviance: Named num [1:13] 1751.99 1743.63 75.96 8.28 25.45 ...
##   .. ..- attr(*, "names")= chr [1:13] "ML" "REML" "ldL2" "ldRX2" ...
##   ..@ fixef   : Named num [1:2] 251.4 10.5
##   .. ..- attr(*, "names")= chr [1:2] "(Intercept)" "Days"
##   ..@ ranef   : num [1:36] 2.26 -40.4 -38.96 23.69 22.26 ...
##   ..@ u       : num [1:36] 2.33 39.69 -41.79 -34.59 -40.3 ...
##   ..@ eta     : num [1:180] 254 273 293 313 332 ...
##   ..@ mu      : num [1:180] 254 273 293 313 332 ...
##   ..@ muEta   : num(0)
##   ..@ var     : num(0)
```

```
##   ..@ resid   : num [1:180] -4.1 -14.62 -42.19 8.78 24.52 ...
##   ..@ sqrtXWt : num[0 , 1]
##   ..@ sqrtrWt : num(0)
##   ..@ RZX     : num [1:36, 1:2] 3.022 0.269 3.022 0.269 3.022 ...
##   ..@ RX      : num [1:2, 1:2] 3.79 0 2.3 16.56
##   ..@ ghx     : num(0)
##   ..@ ghw     : num(0)

# you may be unfamiliar with S4 classes
selectMethod("print", "mer")

## function (x, ...)
## {
##     .local <- function (x, digits = max(3, getOption("digits") -
##         3), correlation = TRUE, symbolic.cor = FALSE, signif.stars = getOption("show.signif.stars"),
##         ...)
##     {
##         so <- summary(x)
##         REML <- so@dims[["REML"]]
##         llik <- so@logLik
##         dev <- so@deviance
##         dims <- x@dims
##         cat(so@methTitle, "\n")
##         if (!is.null(x@call$formula))
##             cat("Formula:", deparse(x@call$formula), "\n")
##         if (!is.null(x@call$data))
##             cat("   Data:", deparse(x@call$data), "\n")
##         if (!is.null(x@call$subset))
##             cat(" Subset:", deparse(x@call$subset), "\n")
##         print(so@AICtab, digits = digits)
##         cat("Random effects:\n")
##         print(so@REmat, quote = FALSE, digits = digits, ...)
##         ngrps <- so@ngrps
##         cat(sprintf("Number of obs: %d, groups: ", dims[["n"]]))
##         cat(paste(paste(names(ngrps), ngrps, sep = ", "), collapse = "; "))
##         cat("\n")
##         if (is.na(so@sigma))
##             cat("\nEstimated scale (compare to 1):", sqrt(exp(so@deviance[["lr2"]])/so@dims[["n"]]),
##                 "\n")
##         if (nrow(so@coefs) > 0) {
##             cat("\nFixed effects:\n")
##             printCoefmat(so@coefs, zap.ind = 3, digits = digits,
##                 signif.stars = signif.stars)
##             if (correlation) {
##                 corF <- so@vcov@factors$correlation
##                 if (!is.null(corF)) {
##                   p <- ncol(corF)
##                   if (p > 1) {
##                     rn <- rownames(so@coefs)
##                     rns <- abbreviate(rn, minlength = 11)
##                     cat("\nCorrelation of Fixed Effects:\n")
##                     if (is.logical(symbolic.cor) && symbolic.cor) {
##                       corf <- as(corF, "matrix")
##                       dimnames(corf) <- list(rns, abbreviate(rn,
```

```
##                         minlength = 1, strict = TRUE))
##                       print(symnum(corf))
##                     }
##                     else {
##                       corf <- matrix(format(round(corF@x, 3),
##                         nsmall = 3), ncol = p, dimnames = list(rns,
##                         abbreviate(rn, minlength = 6)))
##                       corf[!lower.tri(corf)] <- ""
##                       print(corf[-1, -p, drop = FALSE], quote = FALSE)
##                     }
##                   }
##                 }
##             }
##         }
##         invisible(x)
##     }
##     .local(x, ...)
## }
## <environment: namespace:lme4>
## attr(,"target")

## An object of class "signature"

##     x
## "mer"
## attr(,"defined")

## An object of class "signature"

##     x
## "mer"
## attr(,"generic")
## [1] "print"
## attr(,"generic")attr(,"package")
## [1] "base"
## attr(,"class")
## [1] "MethodDefinition"
## attr(,"class")attr(,"package")
## [1] "methods"

fm2 <- summary(fm1)
fm2@REmat

##  Groups     Name          Variance Std.Dev. Corr
##  "Subject"  "(Intercept)" "612.1"  "24.74"  ""      ""
##  ""         "Days"        " 35.1"  " 5.92"  "0.066" ""
##  "Residual" ""            "654.9"  "25.59"  ""      ""

fm2@coefs

##             Estimate Std. Error t value
## (Intercept)   251.41      6.825  36.838
## Days           10.47      1.546   6.772

cat(paste(capture.output(fm1)[7:10], collapse = "\n"), "\n")
```

```
##  Groups   Name        Variance Std.Dev. Corr
##  Subject  (Intercept) 612.1    24.74
##           Days         35.1     5.92    0.066
##  Residual             654.9    25.59


# commented out b/c knitr doesn't like sink

# sink('obnoxiousway.txt')
args(sample)

## function (x, size, replace = FALSE, prob = NULL)
## NULL

# sink()

# scan('obnoxiousway.txt', what='character', sep='\n')[1]
capture.output(args(sample))[1]

## [1] "function (x, size, replace = FALSE, prob = NULL) "
```

3 sprintf

The paste function comes up a little short at times - I have two problems with it. First, if I'm mixing character strings and variables, there may lots of extraneous typing. Second, it would be nice if it could format my output. The function I want is the sprintf function. It's easy to specify my character string and there are many formatting options I can apply to my data.

```
vowels <- c("A", "E", "I", "O", "U")
paste("I'd like to buy an ", vowels[1], ", ", vowels[2], ", ", vowels[3], ", ", vowels[4],
    ", or ", vowels[5], ".", sep = "")

## [1] "I'd like to buy an A, E, I, O, or U."

# %s represents a character string
sprintf("I'd like to buy an %s, %s, %s, %s, or %s", vowels[1], vowels[2], vowels[3], vowels[4],
    vowels[5])

## [1] "I'd like to buy an A, E, I, O, or U"

vals <- rnorm(5, sd = 25)
for (i in vals) print(i)

## [1] -18.44
## [1] 59.15
## [1] -21.07
## [1] -39.07
## [1] 14.2

# %f represents double precision value + asks that the sign (+/-) be printed m.n gives
# the field width (m=8) and precision (n=4)
for (i in sprintf("%+8.4f", vals)) print(i)
```

```
## [1] "-18.4440"
## [1] "+59.1524"
## [1] "-21.0673"
## [1] "-39.0671"
## [1] "+14.2003"
```

There are many other types you could specify

dioxX	integer value; 'o' for octal, 'xX' for hexadecimal
eE	double precision value, in exponential notation
gG	double precision value, context will choose "f" or "e" form
aA	double precision value, in binary notation
%	a literal '%'

4 expand.grid + combn

Have you ever needed to create a data.frame of all the combinations of something? You can acheive this with for loops. If you're choosing elements, you must make a slight alteration to your iterators. This can end up being very inefficient, especially if you're growing your data.frame one row at a time. Fortunately expand.grid and combn can handle both tasks.

```
vowels <- c("A", "E", "I", "O", "U")
twovowels <- matrix(nrow = 0, ncol = 2)
for (i in vowels) {
    for (j in vowels) {
        twovowels <- rbind(twovowels, c(j, i))
    }
}
data.frame(twovowels)
```

```
##    X1 X2
## 1   A  A
## 2   E  A
## 3   I  A
## 4   O  A
## 5   U  A
## 6   A  E
## 7   E  E
## 8   I  E
## 9   O  E
## 10  U  E
## 11  A  I
## 12  E  I
## 13  I  I
## 14  O  I
## 15  U  I
## 16  A  O
## 17  E  O
## 18  I  O
## 19  O  O
## 20  U  O
## 21  A  U
## 22  E  U
## 23  I  U
## 24  O  U
```

```
## 25  U  U
```

```
expand.grid(vowels, vowels)
```

```
##    Var1 Var2
## 1     A    A
## 2     E    A
## 3     I    A
## 4     O    A
## 5     U    A
## 6     A    E
## 7     E    E
## 8     I    E
## 9     O    E
## 10    U    E
## 11    A    I
## 12    E    I
## 13    I    I
## 14    O    I
## 15    U    I
## 16    A    O
## 17    E    O
## 18    I    O
## 19    O    O
## 20    U    O
## 21    A    U
## 22    E    U
## 23    I    U
## 24    O    U
## 25    U    U
```

```
t(combn(vowels, 2))
```

```
##       [,1] [,2]
##  [1,] "A"  "E"
##  [2,] "A"  "I"
##  [3,] "A"  "O"
##  [4,] "A"  "U"
##  [5,] "E"  "I"
##  [6,] "E"  "O"
##  [7,] "E"  "U"
##  [8,] "I"  "O"
##  [9,] "I"  "U"
## [10,] "O"  "U"
```

5 apply

The set of apply functions can be very useful in speeding up your code. A basic "apply" iterates over the rows or columns of a matrix, passing this vector as the values to a given function call. The important thing to remember is that MARGIN=1 applies to rows and MARGIN=2 applies to columns.

```
# function to count NAs in each column
nmissing <- function(x) apply(x, MARGIN = 2, FUN = function(i) sum(is.na(i)))
```

```r
x <- sample(100)
x[sample(100, 10)] <- NA
x <- matrix(x, nrow = 25)
nmissing(x)
```

```
## [1] 2 6 1 1
```

```r

# function to indicate repeated rows
repeated.bad <- function(x) {
    nx <- nrow(x)
    y <- rep(FALSE, nx - 1)
    if (nx > 1) {
        for (i in seq(1, nx - 1)) {
            if (all(x[i, ] == x[i + 1, ]))
                y[i] <- TRUE
        }
    }
    y
}
repeated <- function(x) {
    x <- apply(x, MARGIN = 1, paste, collapse = "")
    nx <- length(x)
    y <- rep(FALSE, nx - 1)
    if (nx > 1) {
        for (i in seq(1, nx - 1)) {
            if (x[i] == x[i + 1])
                y[i] <- TRUE
        }
    }
    y
}
set.seed(2)
vowels <- c("A", "E", "I", "O", "U")
x <- data.frame(X1 = sample(vowels, 1000, replace = TRUE), X2 = sample(vowels, 1000, replace = TRUE))
x1 <- duplicated(x)
system.time(x2 <- repeated.bad(x))
```

```
##    user  system elapsed
##   0.444   0.000   0.443
```

```r
system.time(x3 <- repeated(x))
```

```
##    user  system elapsed
##   0.008   0.000   0.008
```

```r
head(x1, 20)
```

```
##  [1] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
## [15]  TRUE  TRUE  TRUE FALSE  TRUE FALSE
```

```r
head(x3, 20)
```

```
##  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [15] FALSE  TRUE FALSE FALSE FALSE FALSE
```

```r
# splitting data by ID
set.seed(68)
size <- 10000
x <- data.frame(id = sample(1:20, size, replace = TRUE), X1 = rnorm(size), X2 = rpois(size,
    1000), X3 = rbinom(size, 1, prob = 0.5))

a1 <- aggregate(x, by = list(x[, 1]), quantile, 0.9)
# aggregate is basically an apply of a tapply
a2 <- apply(x, MARGIN = 2, tapply, x[, 1], quantile, 0.9)

ids <- sort(unique(x[, 1]))
res <- vector("list", 20)
for (i in seq_along(ids)) {
    res[[ids[i]]] <- apply(x[x[, 1] == ids[i], ], MARGIN = 2, quantile, 0.9)
}
```

6 do.call

Similar to the apply functions, do.call can be utilized to run a given function multiple times. If there's one thing you should learn about do.call, it should be how to use it to combine lists of data.frames.

```r
# combine list of data.frames into one data.frame
do.call("rbind", res)
```

```
##       id    X1   X2 X3
##  [1,]  1 1.287 1040  1
##  [2,]  2 1.291 1040  1
##  [3,]  3 1.222 1038  1
##  [4,]  4 1.240 1040  1
##  [5,]  5 1.161 1042  1
##  [6,]  6 1.293 1034  1
##  [7,]  7 1.264 1040  1
##  [8,]  8 1.279 1037  1
##  [9,]  9 1.258 1043  1
## [10,] 10 1.269 1038  1
## [11,] 11 1.212 1042  1
## [12,] 12 1.345 1040  1
## [13,] 13 1.112 1039  1
## [14,] 14 1.354 1041  1
## [15,] 15 1.285 1040  1
## [16,] 16 1.230 1037  1
## [17,] 17 1.236 1042  1
## [18,] 18 1.106 1040  1
## [19,] 19 1.349 1039  1
## [20,] 20 1.316 1038  1
```

```r
# increase the size of a matrix
m <- matrix(rnorm(25), nrow = 5)
# grow three more rows and columns
newvals <- rep(0, 3)
m <- do.call("cbind", c(list(m), newvals))
m
```

```
##        [,1]    [,2]     [,3]    [,4]    [,5] [,6] [,7] [,8]
## [1,] 0.5461 -0.8542  1.20939 -1.5030  0.3487    0    0    0
## [2,] 0.5112  0.1923 -0.09633  1.6866  0.2494    0    0    0
## [3,] 0.7192  0.2614  1.22432 -1.2128  1.9240    0    0    0
## [4,] 1.1429  0.5608  1.49441 -2.6082 -0.3268    0    0    0
## [5,] 0.6726  2.1773  1.30773  0.7242 -0.6439    0    0    0

m <- do.call("rbind", c(list(m), newvals))
m

##        [,1]    [,2]     [,3]    [,4]    [,5] [,6] [,7] [,8]
## [1,] 0.5461 -0.8542  1.20939 -1.5030  0.3487    0    0    0
## [2,] 0.5112  0.1923 -0.09633  1.6866  0.2494    0    0    0
## [3,] 0.7192  0.2614  1.22432 -1.2128  1.9240    0    0    0
## [4,] 1.1429  0.5608  1.49441 -2.6082 -0.3268    0    0    0
## [5,] 0.6726  2.1773  1.30773  0.7242 -0.6439    0    0    0
## [6,] 0.0000  0.0000  0.00000  0.0000  0.0000    0    0    0
## [7,] 0.0000  0.0000  0.00000  0.0000  0.0000    0    0    0
## [8,] 0.0000  0.0000  0.00000  0.0000  0.0000    0    0    0
```

7 Bonus

Buy five, get one free - today's bonus section will cover the tryCatch function. It is useful in handling errors, but I'll show a simple example of how it can be used to bypass problems.

```
set.seed(35)
results <- vector("list", 5)
for (i in seq(5)) {
    x <- sample(1:2, 25, replace = TRUE)
    x <- matrix(x, ncol = 5)
    # x has a good chance to be singular, which will cause solve to fail
    if (!is.null(tryCatch(res <- solve(x), error = function(e) {
    })))
        results[[i]] <- res
}
results

## [[1]]
##            [,1] [,2] [,3] [,4] [,5]
## [1,]  7.401e-17   -1  1.5    0   -1
## [2,]  7.401e-17    0 -0.5    0    1
## [3,] -1.000e+00    0  1.5    0   -1
## [4,]  0.000e+00    1 -1.5    1    0
## [5,]  1.000e+00    0 -0.5   -1    1
##
## [[2]]
## NULL
##
## [[3]]
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    1  0.0   -1   -1
## [2,]   -1    0 -0.5    1    1
## [3,]    0   -1  0.0    0    1
```

```
## [4,]   -1    0  1.0    0    0
## [5,]    1    0  0.0    0   -1
##
## [[4]]
##         [,1] [,2]    [,3] [,4] [,5]
## [1,]  0.6667    0  0.6667   -1    0
## [2,] -1.6667    1 -0.6667    1    0
## [3,]  0.6667   -1  0.6667    0    0
## [4,] -0.3333    0 -0.3333    0    1
## [5,]  1.0000    0  0.0000    0   -1
##
## [[5]]
## NULL
##
```