Tips and tricks from an R veteran

Jonathan Rougier (age 52.7)

School of Mathematics
University of Bristol

Bristol R Meetup, June 2019

A dinosaur speaks!

A dinosaur speaks!

- **Languages:** ~~Spectrum Basic~~, ~~Z80 assembler~~, ~~Pascal~~, C, ~~C++~~, Bourne Shell, ~~Java~~, R, ~~Python~~, ~~html & xml~~
- **Operating systems:** ~~PDP-11~~, ~~DOS~~, ~~Windows~~, ~~Solaris~~, ~~MacOS~~, Ubuntu GNU/Linux
- **Computers:** ~~DEC Minicomputers~~, ~~386 and 486 machines~~, GNU/Linux servers & supercomputers, ~~Mac laptops~~, 2nd hand ex-Windows laptops running Ubuntu GNU/Linux
- **Version control:** ~~CVS~~, ~~Subversion~~, ~~git~~, rsync
- **Literate programming:** LaTeX, ~~noweb~~, ~~Sweave~~, knitr, Rmarkdown
- **Core software**: OpenBLAS and R, vim, Texlive for LaTeX, Firefox, LibreOffice, evince, gimp

A dinosaur speaks!

- Starting using R in 1997, ver 0.49.

  Was present in Vienna in 2000, when R went to ver 1.0.

  Less active from about 2003.

  My R contemporaries: R & R (of course), Luke Tierney, Brian Ripley, Martin Mächler, Peter Dalgaard, Thomas Lumley, Dirk Eddelbuettel, Kurt Hornik, Fredrich Leisch, Uwe Ligges, Martyn Plummer, Duncan Temple Lang.

A dinosaur speaks!

- Starting using R in 1997, ver 0.49.

  Was present in Vienna in 2000, when R went to ver 1.0.

  Less active from about 2003.

  My R contemporaries: R & R (of course), Luke Tierney, Brian Ripley, Martin Mächler, Peter Dalgaard, Thomas Lumley, Dirk Eddelbuettel, Kurt Hornik, Fredrich Leisch, Uwe Ligges, Martyn Plummer, Duncan Temple Lang.

- Author of some early core functions, `outer` (%o%), `kronecker` (%x%), `aperm`, and the popular `sprintf`.

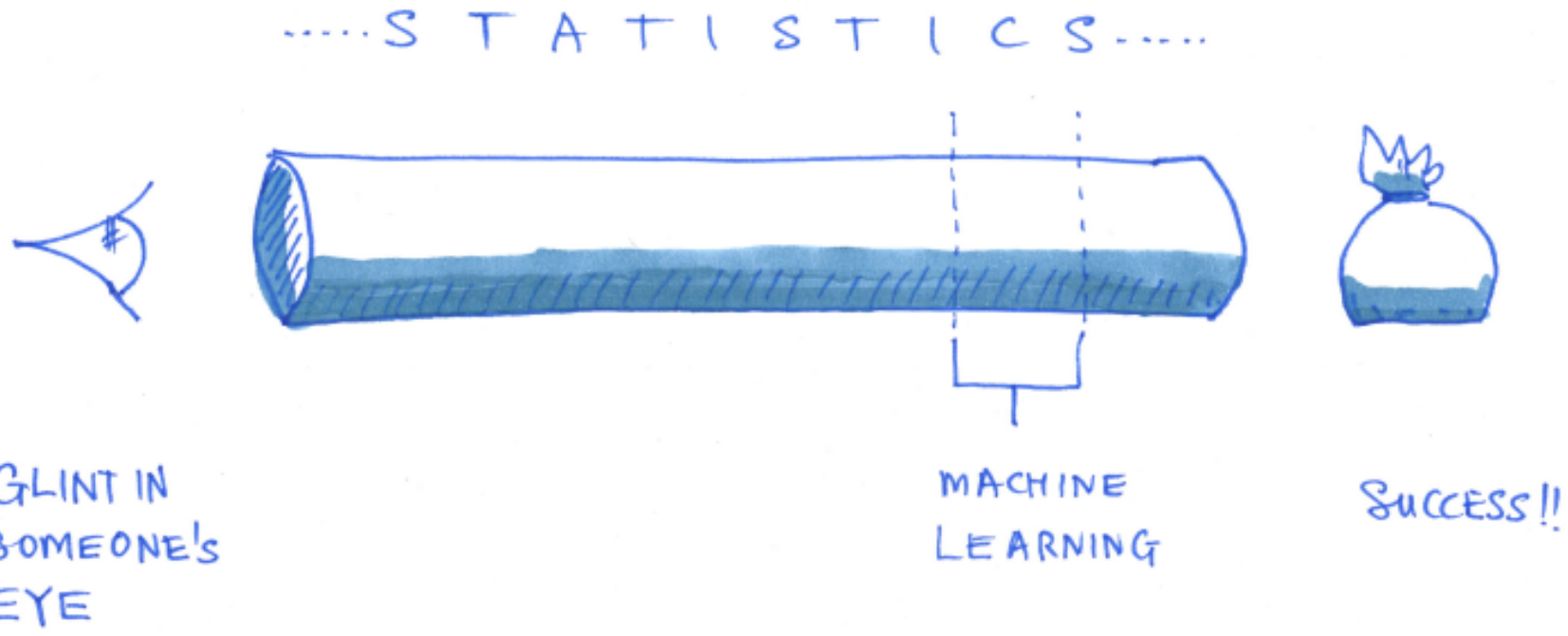
- Most of my CRAN packages have been superseded, but `Oarray` still going (maintained by Robin Hankin), and `tensor`.

A dinosaur speaks!

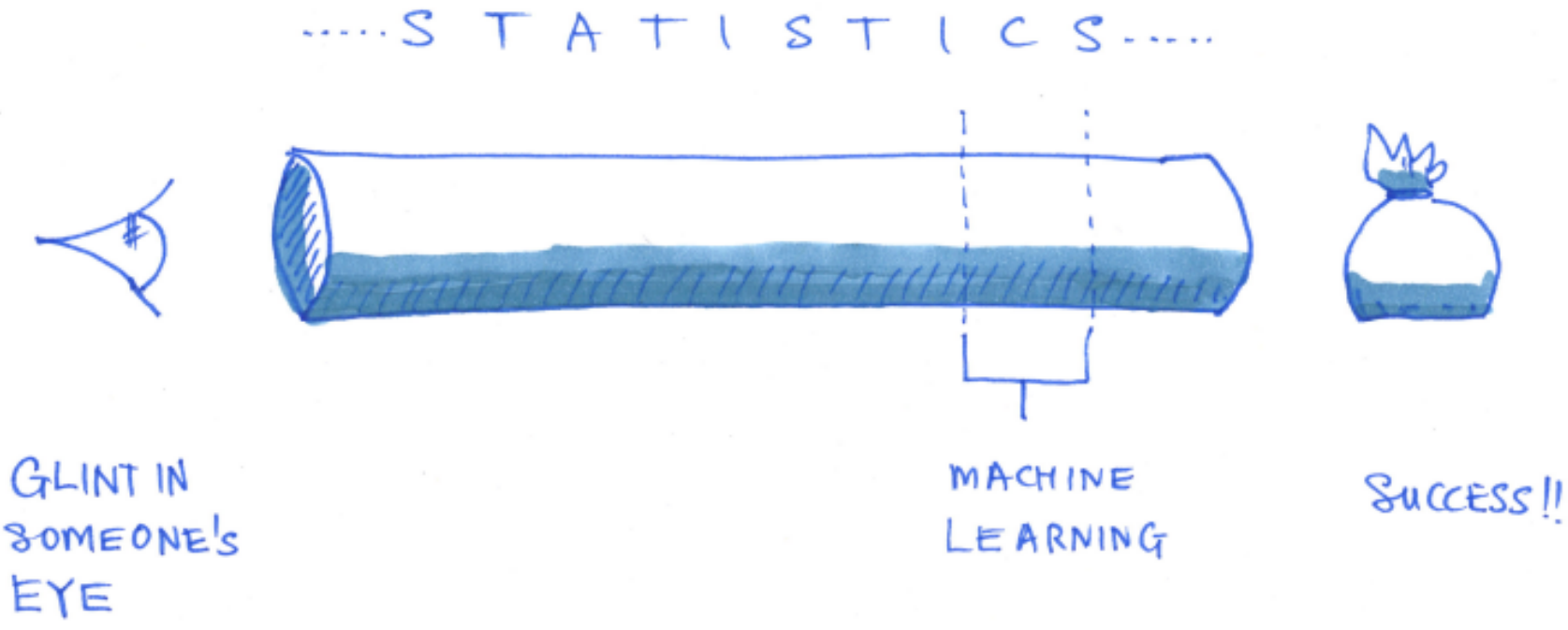
My `.Rprofile`, embedded in my `/.bashrc`

```
echo "
options(repos = \"http://www.stats.bris.ac.uk/R/\")

## this for R markdown

ksource <- function(x, output = tempfile(), ...) {
  source(knitr::purl(x, output = output), ...)
}

require(rmarkdown)

## always need this

require(data.table)

" > ~/.Rprofile
```

Tip 1: Statistics *is* the pipeline



Tip 1: Statistics *is* the pipeline



- Scoping
- Audit of resources
- Audit of data sources
- Data wrangling
- Visualisation
- Algorithm design
- Pilot testing
- Profiling/snagging

Tip 2. Lots of defensive programming

I use `stopifnot`'s at the top of each function, but write little additional functions to make the output easier to understand.

Tip 2. Lots of defensive programming

I use `stopifnot`'s at the top of each function, but write little additional functions to make the output easier to understand.

```
is.scalar <- function(x)
  is.numeric(x) && length(x) == 1L

is.pos.int <- function(x)
  is.scalar(x) && x >= 1 && x == round(x)

skel <- function(x, y, nreps = 100L) {

  n <- length(x)
  stopifnot(length(y) == n, is.pos.int(nreps))

  # etc etc
}
```

(I'm also a bit fussy about types . . .)

Tip 3: Simulated inputs/datasets

Repeatedly use simulated inputs/datasets to test code.

Tip 3: Simulated inputs/datasets

Repeatedly use simulated inputs/datasets to test code.

My typical workflow

R running in Terminal:

```
> source("foo.R") # specified seed
> # or knit2pdf("foo.Rnw")
> # or ksource("foo.Rmd")
> source("foo.R") # different seed
> source("foo.R") # different seed
> # etc etc
```

And '`vim foo.R`' in another Terminal.

Tip 3: Simulated inputs/datasets

Repeatedly use simulated inputs/datasets to test code.

I put this snippet at the top of the source file:

```
## set test

test <- if (exists("test")) test else getOption("test", FALSE)

## control random seed

production <- !exists("production")
if (production) {
  message("Setting seed to 1001")
  set.seed(1001)
} else {
  message("Using random seed")
  set.seed(NULL)
  my.Random.seed <- .Random.seed
}
```

Tip 3: Simulated inputs/datasets

Repeatedly use simulated inputs/datasets to test code.

Deeper in the source file, tests are run locally with

```
if (isTRUE(test)) {
  ## tests go here, using
  ## random inputs and datasets
}
```

Reproducible production graphics will be generated with

```
if (isTRUE(production)) {
  ## draw awesome graphic and
  ## write as PNG file
}
```

Tip 4: Environments are your friend

I use environments in at least three different ways.

Tip 4: Environments are your friend

I use environments in at least three different ways.

1. To keep the function environment uncluttered:

```
skel <- function(x, foo = NULL) {

  if (is.null(foo)) local({
    bar <- seq_along(x)
    nut <- x * 2^bar
    foo <<- sum(nut)
  })

  # etc etc
}
```

Tip 4: Environments are your friend

I use environments in at least three different ways.

2. To bind immutable variables into functions:

```
loglik <- with(
  list(y = yobs, n = length(yobs)),
  function(theta) {
    mu <- theta[1L]; sig2 <- theta[2L]
    if (is.na(sig2) || sig2 <= 0) {
      NA
    } else {
      -(n / 2) * log(sig2) - (y - mu)^2 / (2 * sig2)
    }
  })
```

Tip 4: Environments are your friend

I use environments in at least three different ways.

3. To hold mutable variables across function calls:

```
makeSkel <- function(scale = 1) {

  sumdd <- scale; ndd <- 1L

  function(x, learn = FALSE) {

    # stuff: uses x and scale, creates dd and xnew

    if (isTRUE(learn)) {
      sumdd <<- sumdd + dd; ndd <<- ndd + 1L
      scale <<- sumdd / ndd
    }
    xnew
  }
}
```

Tip 4: Environments are your friend

I use environments in at least three different ways.

3. To hold mutable variables across function calls:

```
## 'classic' adaptive MCMC (eg slice sampler)

skel <- makeSkel(scale = 10)
nsteps <- 100L
spinup <- as.integer(0.1 * nsteps)

x <- 0; X <- rep(NA_real_, nsteps)

for (i in (-spinup):nsteps) {
  x <- skel(x, learn = i < 0L)
  if (i > 0L) X[i] <- x
}
```

Summary

1. Statistics *is* the pipeline
2. Lots of defensive programming
3. Simulated inputs/datasets
4. Environments are your friend