# Scaling reproducible projects

## Reproducible Computing

## @ JSM 2019

Colin Rundel

June 24, 2018

# Example - Scottish Lip Cancer Slides

# The pain points

There are a couple of code blocks that take awhile to run:

1. `get-data`
2. `neighbors`
3. `stan-fit`

What makes each of these slow?

# get-data

```
...

shape_dir = here("data/shapefiles")
dir.create(shape_dir, showWarnings = FALSE, recursive = TRUE)

base_url = "http://web1.sph.emory.edu/users/lwaller/book/ch9/"
shapefiles = c("scot.shp", "scot.dbf", "scot.shx")

for(file in shapefiles) {
   download.file(
     file.path(base_url, file),
     destfile = file.path(shape_dir, file),
     quiet = TRUE
   )
}

...
```

# neighbors

```
d = sf::st_distance(lip_cancer %>% sf::st_set_crs(NA))
class(d) = NULL

W = (d == 0.0) * 1L

m = rowSums(W)
lip_cancer$n_neighbors = m

plot(lip_cancer[,"n_neighbors"], asp=1, axes=TRUE)
```

# stan-fit

```
data = list(
  n = nrow(lip_cancer), p = 1,
  W = W, m = m,
  log_offset = log(lip_cancer$Expected),
  y = lip_cancer$Observed,
  X = matrix(1, nrow = nrow(lip_cancer), ncol=1)
)

options(mc.cores = parallel::detectCores())
full_fit = rstan::stan(model_code = car_model, data = data,
                       iter = 1000, chains = 2, verbose = FALSE)
```

# Roll your own cache

It is fairly straight forward to use R's ability to serialize objects in order to create a simple cache for slow running code.

For example, we can rewrite the `neighbors` code chunk as follows

```
if (!file.exists("dist_mat.rds")) {
  d = sf::st_distance(lip_cancer)
   saveRDS(d, "dist_mat.rds")
} else {
   d = readRDS("dist_mat.rds")
}
class(d) = NULL

W = (d == 0.0) * 1L

m = rowSums(W)
lip_cancer$n_neighbors = m

plot(lip_cancer[,"n_neighbors"], asp=1, axes=TRUE)
```

# Aside - RDS vs Rdata

Probably the most common approach for serializing and read R objects are `save` and `load` respectively.

Generally using Rdata files (via `save` and `load`) is not a best practice because they both save and restore objects and their names. This can result in objects being silently overwritten when an Rdata file is loaded and it also makes it difficult to discover exactly what objects and values are stored in an Rdata file.

`saveRDS` instead saves only a single R object's value and `readRDS` requires that the user explicitly give a name to the object when it is read in.

# Issues

- No depency tracking / invalidation
- Explicitly need to delete `rds` file to rerun code
- Quick and dirty solution that does not scale

# knitr and cached=TRUE

knitr is able to accomplish something similar by caching the results of code chunks when explicitly asked to via the `cached` chunk option.

This cacheing scheme takes into account all objects created, side-effects like plots and text output, basic environmental details like packages used, and automatic or manual specification of dependency between code chunks.

See more at Yihui's Examples for the cache feature.

# Issues

- It is important to understand under what circumstances a cached code chunk will become invalidated, see discussion here
- Constructing code chunk level dependency structures is cumbersome and can be quite brittle
- `autodep` works reasonably well but has many edge cases (e.g. does not work with `source`)
- Having to nuke the entire cache directory by hand is a semi-regular experience.