# ProjectTemplate Demo

Ivan Hanigan

May 11, 2012

## Outline

## The Compendium concept

My goal is to develop data analysis projects along the lines of the Compendium concept of Gentleman and Temple Lang (2007) [1]. Compendia are dynamic documents containing text, code and data. Transformations are applied to the compendium to view its various aspects.

- Code Extraction (Tangle): source code
- Export (Weave): LaTeX, HTML, etc
- Code Evaluation

I'm also following the orgmode technique of Schulte et al (2012) [2]

## The R code that produced this report

I support the philosophy of Reproducible Research http://www.sciencemag.org/content/334/6060/1226.full, and where possible I provide data and code in the statistical software R that will allow analyses to be reproduced. This document is prepared automatically from the associated Emacs Orgmode file. If you do not have access to the Orgmode file please contact me.

## ProjectTemplate

This is a simple demo of the R package *ProjectTemplate* http://projecttemplate.net/ which is aimed at standardising the structure and general development of data analysis projects in R. A primary aim is to allow analysts to quickly get a project loaded up and ready to:

- reproduce or
- create new data analyses.

## Why?

It has been recognised on the R blogosphere that it

- is "meant to handle very complex research projects" (http://bryer.org/2012/maker-an-r-package-for-managing-document-building-and-versioning)
  and
- is considered as being amongst the best approaches to the workflow for doing data analysis with R (http://blog.revolutionanalytics.com/2010/10/a-workflow-for-r.html)

## The Reichian load, clean, func, do approach

The already mentioned blog post
http://blog.revolutionanalytics.com/2010/10/a-workflow-for-r.html
also links to another'best' approach, the:

- *Reichian load, clean, func, do* approach
  http://stackoverflow.com/a/1434424.

By Josh Reich. I've also followed to prepare this demo using the tutorial and data from the package website
http://projecttemplate.net/getting_started.html

## The Peng NMMAPSlite approach

The other approach I followed was that of Roger Peng from Johns Hopkins and his NMMAPSlite R package [3]. Especially the function

```
readCity(name, collapseAge = FALSE, asDataFrame = TRUE)
```

Arguments

- name character, abbreviated name of a city
- collapseAge logical, should age categories be collapsed?
- asDataFrame logical, should a data frame be returned?)

Description: Provides remote access to daily mortality, weather, and air pollution data from the National Morbidity, Mortality, and Air Pollution Study for 108 U.S. cities (1987–2000); data are obtained from the Internet-based Health and Air Pollution Surveillance System (iHAPSS)

## Init the project

First we want to initialise the project directory.

```
####
# init
require('ProjectTemplate')
create.project('analysis',minimal=TRUE)
```

## dir()

```
####
# init dir
dir('analysis')
```

```
cache
config
data
munge
README
src
```

## The reports directory

I've added the reports directory manually and asked the package author if this is generic enough to be in the defaults for

```
minimal = TRUE
```

I believe it may be as the *Getting Started* guidebook states:

> *'It's meant to contain the sort of written descriptions of the results of your analyses that you'd **publish in a scientific paper.***
> *With that report written . . . , we've gone through **the simplest sort of analysis you might run with ProjectTemplate**.*

```
####
# init reports
dir.create('analysis/reports')
```

## Do the analysis

```
####
# this is the start of the analysis,
# assumes the init.r file has been run
if(file.exists('analysis')) setwd('analysis')
Sys.Date()
# keep a track of the dates the analysis is rerun
getwd()
# may want to keep a reference of the directory
# the project is in so we can track the history
```

## Get the projecttemplate tutorial data

Get the data from http://projecttemplate.net/letters.csv.bz2 (I downloaded on 13-4-2012) Put it in the data directory for auto loading.

```
####
# analysis get tutorial data
download.file('http://projecttemplate.net/letters.csv.bz2',
  destfile = 'data/letters.csv.bz2', mode = 'wb')
```

## Tools

Edit the *config/global.dcf* file to make sure that the load_libraries setting is turned on

## Load the analysis data

```
####
# analysis load
require(ProjectTemplate)
load.project()
```

## check the analysis data

```
tail(letters)
```

```
zyryan      z  y
zythem      z  y
zythia      z  y
zythum      z  y
zyzomys     z  y
zyzzogeton  z  y
```

## Develop munge code

Edit the *munge/01-A.R* script so that it contains the following two lines of code:

```
# For our current analysis, we're interested in the total
# number of occurrences of each letter in the first and
# second letter positions and not in the words themselves.
# compute aggregates
first.letter.counts <- ddply(letters, c('FirstLetter'),
  nrow)
second.letter.counts <- ddply(letters, c('SecondLetter'),
  nrow)
```

Now if we run with

```
load.project()
```

all munging will happen automatically. However. . .

## To munge or not to munge?

As you'll see on the website, once the data munging is completed and outputs cached, load.project() will keep recomputing work over and over. The author suggests we manually edit our configuration file.

```
 # edit the config file and turn munge on
 # load.project()
 # edit the config file and turn munge off
 # or my preference
 source('munge/01-A.r')
# which can be included in our first analysis script
# but subsequent analysis scripts can just call load.projec
# without touching the config file
```

## Cache

Once munging is complete we cache the results

```
cache('first.letter.counts')
cache('second.letter.counts')
```

## Plot first and second letter counts

Produce some simple density plots to see the shape of the first and second letter counts.

- Create *src/generate_plots.R*. Use the src directory to store any analyses that you run.
- The convention is that every analysis script starts with load.project() and then goes on to do something original with the data.

## Do generate plots

Write the first analysis script into a file in **src**

```
require('ProjectTemplate')
load.project()
plot1 <- ggplot(first.letter.counts, aes(x = V1)) +
  geom_density()
ggsave(file.path('reports', 'plot1.pdf'))

plot2 <- ggplot(second.letter.counts, aes(x = V1)) +
  geom_density()
ggsave(file.path('reports', 'plot2.pdf'))
dev.off()
```

And now run it (I do this from a main ‘overview’ script).

```
source('src/generate_plots.r')
```

## First letter

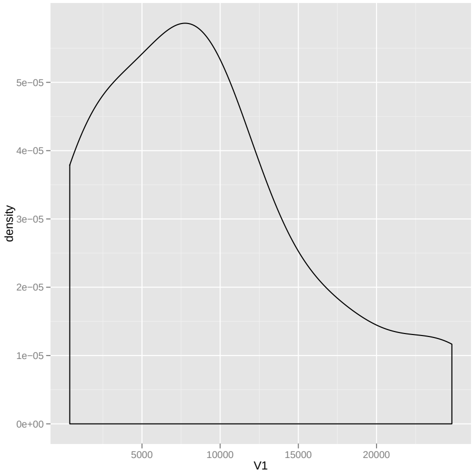

Figure : plot1.pdf

## Second letter

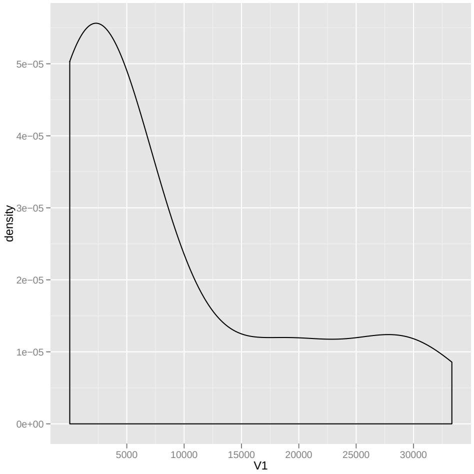

Figure : plot2.pdf

## Report results

We see that both the first and second letter distributions are very skewed. To make a note of this for posterity, we can write up our discovery in a text file that we store in the reports directory.

```
\documentclass[a4paper]{article}
\title{Letters analysis}
\author{Ivan Hanigan}
\begin{document}
\maketitle
blah blah blah
\end{document}
```

## Produce final report

```
# now run LaTeX on the file in reports/letters.tex
```

## Personalised project management directories

```
####
# init additional directories for project management
analysisTemplate()

dir()
```

admin
analysis
data
document
init.r
metadata
ProjectTemplateDemo.org
references
tools
versions

## Navigating using other code editors

Emacs is not for everyone.

*a great operating system, lacking only a decent editor*

http://upsilon.cc/žack/blog/posts/2008/10/from_Vim_to_Emacs_-_part_1/

## RStudio

Let's take a look at the project using RStudio.

## NMMAPSlite

To fit a poisson GAM following the general apprach from the NMMAPS paradigm.

$$\begin{aligned} log(O_i) &= s(temp_i) + s(dewpoint_i) \\ &+ s(Time, df = 7 \times YearsOfData, fixed = T) \\ &+ offset(log(Pop_i)) \end{aligned}$$

Where:
$O_i$ = Counts of health outcome on $day_i$
$s(temp_i) + s(dewpoint_i)$ = Non-parametric smooths of temperature and dewpoint on $day_i$
$s(Time, df = 7 \times YearsOfData, fixed = T)$ = parametric smooth of time (days) to adjust for time varying confounders
$Pop_i$ = interpolated population on $day_i$

## NMMAPSlite readCity()

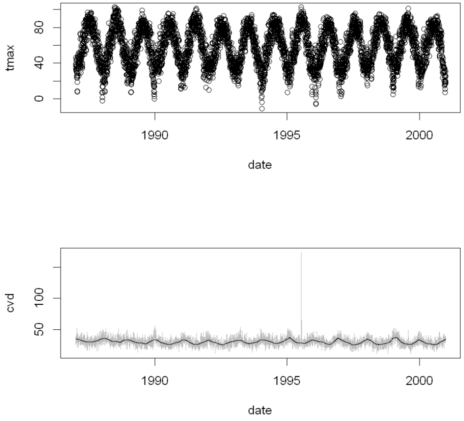

Figure : NMMAPSliteEgGAM-qc.png

# NMMAPSlite GAM

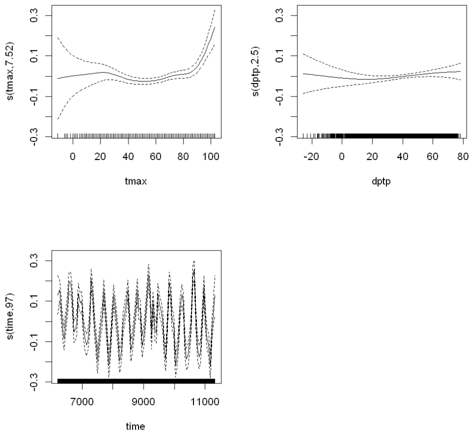

Figure : NMMAPSliteEgGAM-exposureResponse.png

## NMMAPSlite ad hoc extension

So now if a user wants to do the analysis differently with the same data they can.

```
require(ProjectTemplate)
load.project()
ls()
# fit without seasonal cycle (df = 1)
fit <- gam(cvd ~ s(tmax) + s(dptp) +
  s(time, k= numYears/numYears, fx=T),
  data = df, family = poisson)
png('reports/NMMAPSliteEgGAM-exposureResponse-extension.png
par(mfrow=c(2,2))
plot(fit)
dev.off()
```

## NMMAPSlite GAM extension

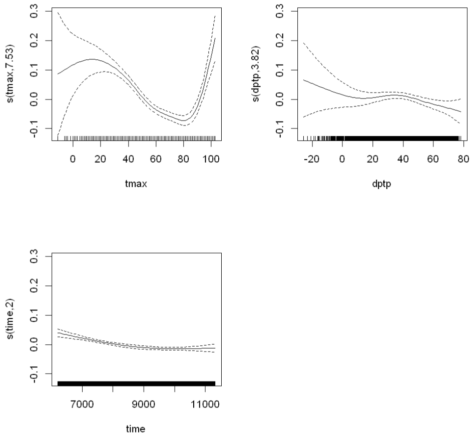

Figure : NMMAPSliteEgGAM-exposureResponse-extension.png

## Conclusions

The Emacs Orgmode file is the compendium from which the whole analysis can be re-created. The upshot is that once I have developed the project's main *.org* file and completed the analysis I can send it (and only it) to another analyst and if they run it (using Emacs) it should get the project to exactly the same state that it was in when I left it, ready for reproduction or extension.

---

THANKS for listening!
To see a copy of the org file for this demo go to
https://github.com/ivanhanigan/ProjectTemplateDemo

## References

- Robert Gentleman and Duncan Temple Lang.
  Statistical Analyses and Reproducible Research.
  *Journal of Computational and Graphical Statistics*, 16(1):1–23, March 2007.
- E Schulte, D Davison, T Dye, and C Dominik.
  A Multi-Language Computing Environment for Literate Programming and Reproducible Research.
  *Journal of Statistical Software*, 46(3), 2012.
- RD Peng and LJ Welty.
  The NMMAPSdata Package.
  *R News*, 4(2):10–14, 2004.

## System State

Note down the state of the computer at the time of the successful run (note that this doesn't export to the LaTeX file using exports results).

```
sessionInfo()
```