ProjectTemplate and R Workflow

Dr Jeromy Anglim
School of Psychology, Deakin University

Context

- Use case
  - You have one or more datasets
  - You need to prepare the data
  - You need to explore the data and generate insights probably in the form of tables, figures, and text
- I.e., not package development where the aim is to develop a set of reusable functions

Symptoms of a bad R workflow

- Loading a saved rdata file to return to analyses
- library statements spread throughout the code;
  - i.e., no easy way to see at a glance which packages are required
  - unclear whether a package will be loaded in time for when it is needed
- Diagnosing an obscure error when you try to run your code on a system with a different stringAsFactors setting
- It's unclear how to get back up and running with analyses
- When starting a new project, you need to create a pile of setup folders; loose or approximate standards force you to think about where to put things, and make it difficult to return to projects in the future
- Data manipulations are interspersed with analyses. Thus, derived variables or datasets may not exist or may be in the wrong form when a given analysis is performed.

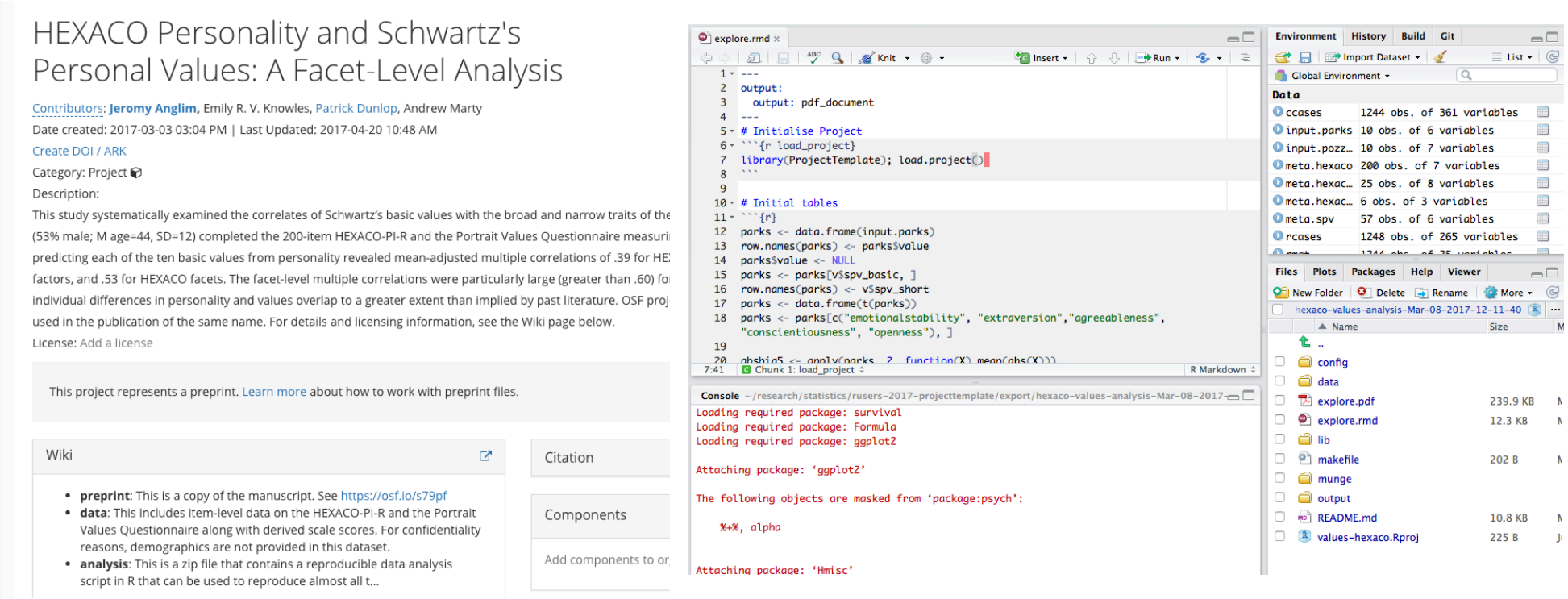
Aims of this talk

- Aims
  - Introduce ProjectTemplate
  - Introduce the idea of customising ProjectTemplate
  - Show how to use this approach
  - Explain why this overcomes many issues related to data analysis workflow in R

- Benefits
  - All the benefits of ProjectTemplate
  - And you are up and running in a new project in a few clicks

Initial Demo:
Example project for journal article

- Files are available at: https://osf.io/wkc5u/
- Video review of materials: https://www.youtube.com/watch?v=GKtjr-lxHYM



ProjectTemplate

- http://projecttemplate.net/

ProjectTemplate

Introduction
Installing
Getting Started
Mastering ProjectTemplate
Configuring
Updating
Architecture
Supported File Formats
Changes
Mailing List
Contributing
Building Packages
ProjectTemplate on GitHub
ProjectTemplate on CRAN

ProjectTemplate is a system for automating the thoughtless parts of a data analysis project:

- Organizing the files in your project.
- Loading all the R packages you'll use.
- Loading all of your data sets into memory.
- Munging and preprocessing your data into a form that's suitable for analysis.

In addition to automating the drudge work of analyzing data, ProjectTemplate hopes to promote better coding and analysis practices by:

- Curating the best R packages.
- Providing simple tools for keeping a log of your work
- Providing template code for:
  - Data diagnostics
  - Data munging
  - Code profiling
  - Unit testing

To learn how to use ProjectTemplate, we suggest going through the ProjectTemplate tutorial.

Styled with skeleton. Design ideas taken from the plyr and rApache websites. Powered by Jekyll and GitHub Pages.

Creator

- John Myles White

Commit Team

- Kirill Müller
- Kenton White

Code Contributors

- People who supplied pull requests
- Diego Valle-Jones
- Patrick Schalk
- Noah Lorang
- Jeffrey Breen
- Aleksandar Blagotic

Idea Contributors and Inspiration

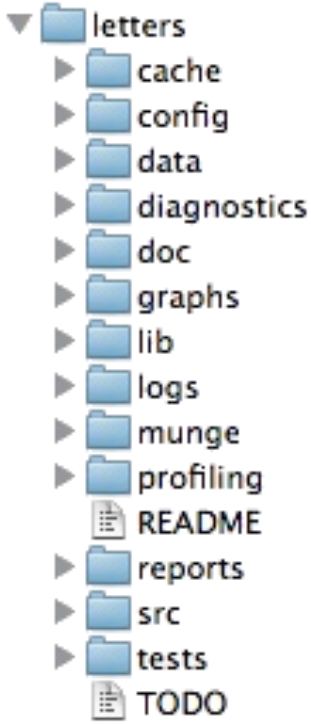
- Hadley Wickham
- Mike Dewar
- Jamie Olson
- The Ruby on Rails Developers

ProjectTemplate

- Why use ProjectTemplate?
  - Encourages good workflow
    - (1) Configure, (2) load packages, (3) load support files, (4) load data, (5) manipulate data, (6) analyse data
- More
  - Systematic place to store files and settings
  - Standardises configuration and package loading settings
  - Automatically load r-script files
  - Automatically load data files stored in data directory
  - Automate running initial data manipulation code
- Installation
  - `install.project("ProjectTemplate", dep = TRUE)`

Summary of Creating a ProjectTemplate Project

- Create the folder structure
  - `library('ProjectTemplate')`
    `create.project('myproject')`
- Review `config/global.dcf`
  - Choose settings
  - Specify packages to load
- Add data for auto-loading to `data` directory
- Add any additional R support functions to the `lib` directory
- Load the project
  - `library('ProjectTemplate');`
    `load.project()`
- Write any initial data manipulation code and place in the munge directory
- Create data analysis files (e.g., r-scripts, RMarkdown, Sweave Files) in home or reports directory
  - Include the load project commands above at the top of each such file



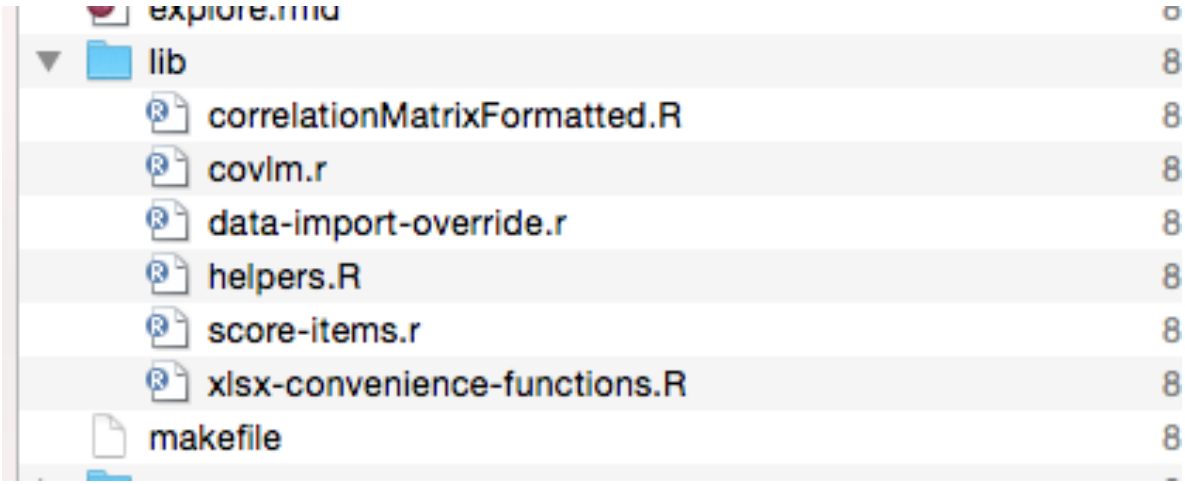
config directory

- Configuration settings are stored in `config/global.dcf`
- `data_loading, munging, load_libraries:` indicate which aspects of ProjectTemplate should run
  - munging: FALSE is useful when debugging the data manipulation process
- `libraries:` Specify which packages you want to load
- `as_factors:` Specifies whether by default strings should be imported as factors

```
data_loading: TRUE
cache_loading: FALSE
munging: TRUE
logging: FALSE
load_libraries: TRUE
libraries: personalityfacets, gdata, gtools, xtable, car, psych, GPArotation,
           nFactors, vegan, digest, bootstrap, MASS, boot, QuantPsyc
as_factors: FALSE
data_tables: FALSE
version: 0.6
recursive_loading: FALSE
attach_internal_libraries: TRUE
```

lib directory

- Code is automatically run
- Thus, you don't have to add and run: `source("lib/myfile.r")` each time you add a new script
- Typical use case:
  - you have little functions that you have from another project or you have developed some custom function for the project and you want them to be accessible (i.e., a full R package would be overkill).



data directory

- Place files in the data folder
- Data is automatically loaded based on the file extension.
  - Benefit: no need to remember function names for data import
- Name of file is generally derived from file name
  - e.g., "mydata.csv" is imported as object "mydata"
  - Excel files with multiple worksheets take the format "filename.sheetname"
  - Data in Rdata format keeps original names
  - You can override defaults by putting a function in the lib directory
- Full list of supported file formats: http://projecttemplate.net/file_formats.html

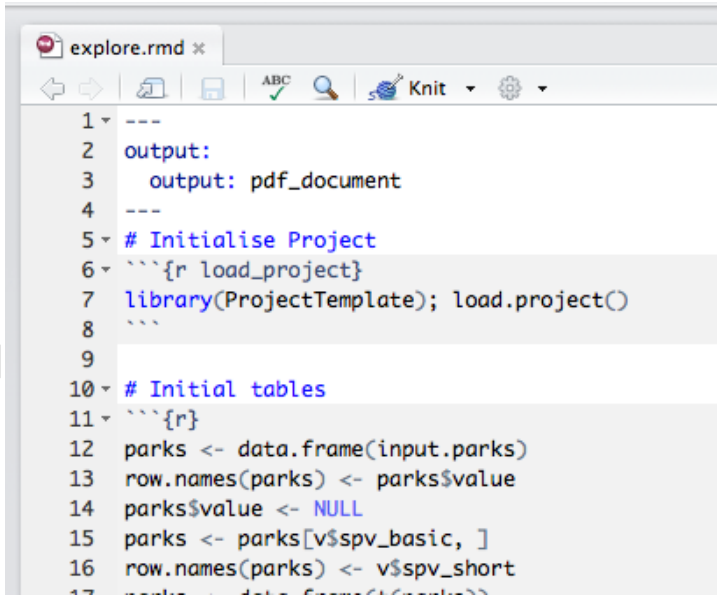
- **.csv**: CSV files
- **.csv.bz2**: CSV
- **.csv.zip**: CSV
- **.csv.gz**: CSV f
- **.csv2**: CSV file
- **.csv2.bz2**: CS
- **.csv2.zip**: CS
- **.csv2.gz**: CS
- **.tsv**: CSV files
- **.tsv.bz2**: CSV
- **.tsv.zip**: CSV
- **.tsv.gz**: CSV f
- **.tab**: CSV files
- **.tab.bz2**: CSV
- **.tab.zip**: CSV
- **.tab.gz**: CSV f
- **.wsv**: CSV file
- **.wsv.bz2**: CS bzip2.
- **.wsv.zip**: CSV
- **.wsv.gz**: CSV
- **.dat**: CSV files
- **.dat.bz2**: CSV bzip2.
- **.dat.zip**:
- **.dat.gz**:
- **.txt**: CSV
- **.txt.bz2**: bzip2.
- **.txt.zip**:
- **.txt.gz**: C
- **.RData**:
- **.rda**: .RD
- **.R**: R sou
- **.r**: R sour
- **.url**: A DC
- **.sql**: A DC
- **.xls**: XLS
- **.xlsx**: XLS
- **.sav**: Bina
- **.dta**: Bina
- **.arff**: Wek
- **.dbf**: DBF
- **.rec**: EPII
- **.mtp**: MT
- **.m**: Octav
- **.sys**: Syst
- **.syd**: Sys
- **.sas**: SAS
- **.xport**: S
- **.xpt**: SAS
- **.db**: A SC
- **.file**: A DC
- **.mp3**: MF
- **.ppm**: PP

munge directory

- Munge files are automatically run after data import
  - Typical names start with numbers to have control over sequencing "01-munge.r", "02-munge.r"
- Common tasks
  - create derived variables:
    - composites of other variables
    - collapse categories
    - convert from character to numeric
    - create factors
    - etc.
  - created aggregated datasets
  - merge datasets
  - remove cases
  - reshape data
  - etc.
- Important point: If you find you are creating derived variables in your analysis script, move this code to the munge files

Running ProjectTemplate

- You should place the following as the first command in an analysis script
  - `library("ProjectTemplate"); load.project()`
- What happens when you run it:
  - Configuration file is loaded: options are set; packages are loaded (and installed from CRAN needed)
  - Scripts in the `lib` file are sourced
  - Data in the `data` folder is loaded into R
  - Data manipulations specified in the `munge` folder are run
- The benefits
  - Thus, after running a single command you are now ready to analyse your data, or perform new analyses.
  - All data import and manipulation steps are reproducible



Customise your own version of ProjectTemplate

- Once you start using ProjectTemplate, you may find many customisations that you always need to make to a new project
  - Packages that you always use
  - Settings that you prefer over the defaults (e.g., `as_factors: FALSE`)
  - Particular ways that you generate analysis scripts
  - Integration with RStudio project structure
  - Only a few folders in ProjectTemplate are truly necessary (i.e., lib, config, data, munge); can be cleaner to remove unnecessary folders
  - Create new folders you routinely use
- Thus, make a customised version of ProjectTemplate that matches your specific needs
- Save this customised version to a special folder on your computer or put it on Dropbox, github, etc.
- To create a new project
  - Make a copy of your customised folder structure
  - Rename the project
  - You only need to complete the project specific customisations

My Customised ProjectTemplate

- Basic description
  - http://jeromyanglim.blogspot.com.au/2014/05/customising-projecttemplate-in-r.html
- Overview of files
  - https://github.com/jeromyanglim/AnglimModifiedProjectTemplate
- Zip file of Template
  - https://github.com/jeromyanglim/AnglimModifiedProjectTemplate/archive/master.zip

My customisations

- Modified config to include preferred packages (e.g., ggplot2, psych, etc.)
- added to config: as_factors: FALSE
- added rstudio project file so that project can be opened with one click in rstudio
- added initial rmarkdown file for performing analyses
  - includes code to load ProjectTemplate
- added output folder as a default space to output any derived files (tables, graphs, derived data)
- added the following to munge file to make it easy to debug munge code
  - # library(ProjectTemplate); load.project(list(munging=FALSE))
- added raw-data folder; standardised place to do very low level data transformations
- readme.md is modified to explain to others how to run code

- My conventions evolve as project needs evolve or new tricks arise
  - considering an export folder and script designed to export for open science
  - makefile to run rmarkdown files

Customised ProjectTemplate Workflow

- Setup ProjectTemplate Folder Structure
  - Copy the zip file
    - I have mine stored on github and bookmarked
    - https://github.com/jeromyanglim/AnglimModifiedProjectTemplate/archive/master.zip
  - Rename the folder and the RStudio Project file
- Add script files
  - Functions that get created during the project or functions that need to be imported get put in .r script files in the `lib` folder (e.g., "myfuntions.r")
- Data
  - Ensure that raw data is roughly in the right format
  - Place data files in `data` folder with the names you want the data.frames to have in R (e.g., mydata.csv becomes `mydata` in R)
- Data manipulation
  - Before analysing data, it is usually necessary to clean the data, create new variables, merge data, and so on.
  - This all goes in scripts in the munge folder.
  - Run `library("ProjectTemplate"); load.project()` to load the data and then write any data manipulation code.

Customised ProjectTemplate Workflow

- Analyses
  - Store analyses (i.e., code to generate summary statistics, models, tables, figures, etc.) in Rmarkdown files
  - You need a code chunk before any analysis that loads the project with the following code
    - `library("ProjectTemplate"); load.project()`
  - It can be useful to have multiple RMarkdown files: e.g., for exploratory analyses, final analyses and so on.
  - Alternatively, just put the rmarkdown file in the working directory

Example of Creating a Project

- If you want to do workshop, go to: https://github.com/jeromyanglim/leuven2016rworkshop
- https://github.com/jeromyanglim/leuven2016rworkshop/tree/master/project-examples/exercise-project-template

  - Go to "exercise-project-template/raw-materials" unzip the Customised version of ProjectTemplate
  - Give the folder and the rstudio project file an appropriate name
  - Put cas.sav into the `data` folder (California Schools Data)
  - Open the Rstudio project file in RStudio
  - Open "reports/explore.rmd" and run `library(ProjectTemplate); load.project()`
  - Add a few basic analyses of `cas` to the next R code chunk
  - Go to "munge/01-munge.R" and add a new variable to cas (e.g., create a variable called performance which is the sum of cas$math and cas$english
  - Return to "reports/explore.rmd" and add another code chunk. Create a histogram of cas$performance.
  - Now imagine that you are exiting RStudio and then returning again. i.e., Quit RStudio and then reload the Rstudio Project file
  - Open "reports/explore.rmd" and run `library(ProjectTemplate); load.project()`
  - You should see that your histogram code for cas$performance still runs

Conclusion

- A few draw backs pertain mostly to collaboration and sharing
  - It does introduce some alternative conventions
    - (e.g., option specification, package loading, data loading)
    - it helps to have a readme that explains how it all works
    - A little bit of a startup cost if you typically just have a five line script and a data file.
  - Sometimes you don't want to rename data files
  - It creates one more dependency
- But many benefits
  - ProjectTemplate is a great tool if you regularly perform data analysis projects
  - Standardisation is very helpful to your future self.
    - One click and you're back up and running with your analysis.
  - It's a great framework for reproducible research.
  - The true power comes from customising ProjectTemplate to your specific workflow. It is very flexible.

Thank You

Questions