Best Practices Guidelines for Projects 2 and 3

A goal of science is to produce work that is reproducible and replicable. Data analysis and coding is not exempt from these ideals! One should be able to successfully replicate all the results from code that you create and understand what the code is doing.

The following Best Practices Guidelines exist to help you in these efforts. These guidelines are derived from the two papers that we read at the beginning of Unit 3. Following these guidelines are a requirement for Projects 2 and 3. Treat this list as a checklist before turning in these projects!

Organization

Standard organization is a key feature of replicability. Where code, data, documentation, and results are placed is important for not just keeping track of your code, but it also helps orient users to what it is using and what it is producing. This helps clearly demarcate what is *data* and what are *results* or calculations that are being performed on the data. Additionally, clearly establishing where documentation and code live helps users navigate your code quickly and efficiently.

All elements of project are in a single folder and follows standard organization.

- A subfolder labeled **src** that contains all source code (aka scripts). Any functions you write or data cleaning scripts should exist as files in this folder.
- A subfolder labeled **doc** that contains documentation. This includes any files that have explanation or present code with explanation or text. This includes RMD files (even though they also contain code), any pictures or figures that may be used in documentation but are not produced by your code, and any other documentation of the project *excluding* the README file (which belongs in the main directory).
- A subfolder labeled **data** that contains all raw data files. This includes your CSV file that should be unaltered, or the same as how you downloaded it, and any other data files required to run your code.
- A subfolder labeled **results** that contains all files produced by the code, including data visualization files and any saved text files of calculations.

Additionally, there are also other subfolders that we won't use in class including the **lib** or library folder which contains compiled code in different languages and **bin** or binary folder which contains compiled, executable binary files.

Documentation

All projects should contain basic documentation on the purpose of a project and how to use the code therein. Without this information, your code is useless to outsiders! There are a few basic elements of documentation that you'll need to consider for projects:

- **README file**: which sits in the main directory and describes the project, the organization of the project, and how to run any files that produce results and their locations. README files are always plain text, nothing fancy like PDFs, so they can be read by any means (including command line). They always contain instructions about how to run the code successfully.
- **A Project Rmd file**: which reproduces the visualization(s) in the doc/ folder. This can also contain a lot of documentation about how specific code is run and what it is doing, which forms the basis of literate programming.
- **Any other documentation**: including files containing notes, project files, literature associated with the project, and articles produced by the project are typically stored in the doc/ folder.
- Other documentation describing individual functions in the project, including a description of inputs and outputs. This is typically located in the code itself (in the script or RMD file) as commented code blocks.

Documentation is a frequently overlooked but critical part of any successful project! Remember, even if you don't intend for someone else to use your code, future you will probably use the code and may not remember much at all! Do Future You a favor and document your code!!

Style

All project code should follow style guidelines in http://adv-r.had.co.nz/Style.html. The most important are:

- Use `<-` and not `=` to assign value to objects. Use `=` inside functions with arguments.
- File names and object names should be lowercase and use a period or underscore `_` to separate words within a name.
- File names and object names should be short and meaningful.
- If file names need to be run in a sequence, prefix them with numbers (e.g. 0, 1, 2, . . .).
- Space should be used to separate all operators (`=`,`+`, `-`, `<-`) and after commas.
- Lines should be about 80 characters long, and longer lines should be broken up with indentation for readability.
- Indentation should be used within control elements like loops, functions, and conditionals. Indentation should be used for code within these elements to clearly delineate where the start and end of code within curly braces `{ }`, with the opening curly brace remaining on the line that defines the control element and the closing curly brace on a line on its own.

Incorrect example:

```
for(i in 1:10){ x <- x+1
print(x)}
```

Correct example:

```
for(i in 1:10){
  x <- x +1
  print(x)
}
```

Version Control (Optional for independent projects)

- All project elements are tracked by git in a single folder (including raw data files).
- The project exists as a single repository on Github.
- Each member of the team updates the code via commits that are pushed to Github.
- A release is created and submitted as the final version.