Style Guide

Naming things

Intro

Every project is different, but most of this will apply for general organization and style.

Coding General

Adhering to the tidyverse style guide would be enough. You can usually just Ctrl+Shft+A[1] any highlighted code to get some decent formatting (but not always!).

R Scripts

R Scripts should be broken up with headers created by Ctrl+shft+R :

```
# Header -------------------------------------------------------------------
```

They should be kept as light as possible and generally serve one function, e.g., data processing, modeling, or visualization.

R Markdown

Chunks

- All chunks should be named.
- Use 'word-word2' (i.e. kebab case) instead of snake case, as this presently causes cross-reference issues for some situations (e.g. tables).

Other

Any documentation output should prefer PDF output as it allows collaborators to open the files easily, and we can use github link to share results.

R Markdown vs. R Scripts

In general it doesn't matter, and Rmd should be preferred. Standard R scripts are probably preferable for modeling, as generally there are many models, warnings and messages are important, and it is the results you want to knit for presentation. However, Rmd is ultimately the more reproducible way to go.

[1]For Mac sub Cmd for Ctrl and Opt for Alt on shortcuts. Unfortunately, RStudio is not consistent with Ctrl/Cmd changes, so often it is still just Ctrl+...

Organization

This repo can be cloned, but you will probably want to create a different RStudio project (i.e. one not named repo-template).

In general you will have the following top level structure

- data
- data/models
- code
- code/data_prep
- code/analysis
- code/functions
- images
- manuscripts
- other_docs
- other_docs/lab_notebooks

The `data` folder should exist, but will always be added to `.gitignore` by default, as it is too risky to accidentally put something in it that would be too sensitive. But basically you can delineate standard data files saved as .csv etc., or even .RData. Model results and similar goes into `/models`, as they typically will save the data stored with them.

The `code` folder will have sub-directories pertaining to the project. These may be specific outcomes or similar. You may also want something like `code/data_preparation` or `code/models` as these scripts/Rmd are typically distinctive.

The `images` folder contains nothing but image files, preferably further organized in some meaningful fashion.

The `other_docs` folder contains pertinent non-manuscript documents, for example other articles, outlines, etc.

The `manuscripts` contains documents that are published/presented in a formal setting.

Given that Rmd files will automatically knit the html/pdf in that same directory as the .Rmd file, I would suggest leaving these associated document files in that folder, rather than some designated 'docs' folder. Otherwise, knitting is no longer automatic, which would potentially greatly slow down the process over the long haul.

Naming things

Basics

Naming things is hard, but we don't want to make it any harder than it has to be. Names should be descriptive enough to give a pretty good idea of what an object/file is or contains. That's pretty much it. We have IDEs that auto-complete, so while perhaps not ideal for coding aesthetics, a clear long name is still better than an ambiguous short one, and doesn't take any more effort to type. More directory organization may help curtail longer file names, while further compartmentalization of scripts may help avoid longer object names.

Files

No data file or object should have the name `data` in it. For files, they should be in a `data` folder that makes clear what they are. You can use `_raw` or `_0` to denote a data file in its initial state that serves as the starting point for later processing. There are other names like `analytic` that add nothing to describing the data in a way that's very helpful (all data is potentially analyzable). Data files should be fairly descriptive, possibly reflecting some filtering applied to the raw data. The accompanying ReadMe file can go a long way to clarifying similarly named files.

Likewise visualizations should be in a folder that makes clear that's what they are, so there is no need to add `plot` or similar to the file name. It is difficult to succinctly describe a visualization, and many should be more than one type, so it's probably better to focus on content. For example:

- Probably not as good: `var1_bar.png`
- Probably better: `var1_frequencies.png`

While `var1_frequencies_bar.png` might be appropriate, it generally only works for the simplest type of visualizations that aren't generally publication worthy (e.g. what would you name a faceted plot containing multiple types of different plots?). In any case, while this naming might be useful in the early stages, you still want to aim for more than just `x_scatter` or `y_line`.

There is generally little need to save files for something that isn't going to be presented.

Do not use spaces in file names.

R files R Markdown files should be named the same or very nearly the same as the document/product created.

Use `.R`, `.RData`, `.Rmd` as consistent extensions. At least for .R and .Rmd, RStudio already does this for you.

Do not use spaces in R markdown file names; this cause problems in where the document is created, whether it will be viewed in with the RStudio viewer, and likely other issues.

General objects

Generic objects are harder to name because they can be anything, but again, the more descriptive the better. In general, it's not necessary to use a `data` prefix/suffix for data objects, otherwise almost everything would have those as part of the name. However, it can be useful to use something like `model_` if you are doing analyses, because you can then collect multiple objects easily as a list by using the prefix/suffix, which then allows mapping of other functions to the model objects simultaneously (e.g. summaries). Such an approach might also work for other types of objects, including data objects.

Data

Data names generally should just describe the data contents. For example after initial import, you might just call it `medicare`, and after filtering, `medicare_65` to denote only age 65+, `medicare_65_male` for the same but only males, etc. At some point it will become too unwieldy to add more description, which might mean renaming to `mc_65_male`. As long as it's documented, that shouldn't be too big of a deal, but better would probably be to create a sub-directory, for example `data/medicare/male_65.RData`.

Column names

Column names generally adhere to similar principles as object names- meaningful, snake_case, etc. Variable transformations should be of the form `varname_transform`. For example, a scaled age goes from `age` to `age_sc` or `age_std` (standardized), a logged GDP from `gdp_per_cap` `gdp_per_cap_log`. *Always retain the original variables*, do not overwrite them with a transformation.

Categorical variables

Binary variables can often just be binary 0-1 variables. For example, instead of a factor 'sex' with 'female' and 'male' labels, one could use the variable name 'female' and 1 if female, 0 if male. What you definitely do not want is a sex variable with values of 1 and 2 and no labels, or something similarly vague. Generally factors/strings are preferred to binary/logical because that's what will have to be presented.

If collapsing a categorical you can note how many categories in the new variable name. For example, `cat_var` to `cat_var_3grp`.

Missing values

Missing values should never have a special code other than `NA` (or the default for your data system). It can only cause problems if things like `999`, `''`, `N/A`, are used, and doing so has never helped any analysis. Things like `Don't Know` types of categories almost always have to be treated as missing, so should be converted as such.

Functions

Functions should be **verbs** stating what the function does, not what it produces, or what inputs are needed, or something like that. Can you guess what the following do?

- `fit_model`
- `extract_fixed_effects`
- `add_predictions`

Functions are perhaps more easily named than other things if you adhere to this simple approach.

I find custom plotting functions to not be very useful, because they would typically need a dozen or more arguments, which can't be spread over multiple layers without adding even more arguments, and which have almost no defaults that are generalizable. Even for the exact same type of data, as an example, text changes which require multiple inputs to changed, possibly even some that were not already built into the function, thereby growing the function further. It may be the case that within a document you are repeatedly displaying the exact same information (e.g. descriptive statistics), in which instance, such a function might be viable.

It is however very useful to create a theme that will take care of various aspects that are generalizable, but even then long labels/titles etc. can easily undermine any defaults. The gist is, functions should make repetitive actions easier, not be a mere wrapper that's actually more difficult to use, or remember how to use.

Write function code as you would developing a package, and it's good to set things up in case you want to actually create a package.

Functions should be saved in a `functions/` folder. This will make it easy to create a package based on the folder if desired.

One should add appropriate documentation for a function in the function file. In R, Ctrl+shft+alt+R (from within a function) creates the roxygen template for you. Filling in just a few words goes a very long way to being able to use the function long into the future.

Unless you are writing methods for a function, there generally should only be one function per file.

Other stuff

- For R objects and more generally, lower case names.
- Names with underscores (snake_case) are generally easier to read than other options, e.g. .case or CamelCase, and dots have specific functional meaning in R, Python, etc.
- Image files should preferably be svg, but other formats may be okay for a given scenario.
- As sensitive data will often have to be housed non-locally, if at all possible, the original (sensitive) data used for a project with a repo should be in a folder with the same name as the repo. For example `some-network-drive/this-repo-name/data`.

When in doubt

Consult the tidyverse style guide for R. It's not perfect but provides some guidance, and would be useful for other languages as well.

Consult your future self and ask them what you should do to make it easier for them.