ADF&G Biometrics Best Practices–DRAFT

Ben Williams and others

June 1, 2022

Contents

This document is to explicitly list coding best practices in *R* for biometricians and quantitative biologists within ADF&G Commercial Fisheries, with the hopes that a common framework will facilitate collaboration and code review. Much of what is in here originates from Grolemund and Wickham's excellent R for Data Science "Book" http://r4ds.had.co.nz/. This is all predicated upon the assumption that your data are human and machine readable!

Naming conventions

Generally speaking it is best to use Google's R style guide, if in doubt default to these: https://google.github.io/styleguide/Rguide.xml

File names Do use:
`lower_case_and_underscore.R`

If you are changing files or versions often then adding a date can be beneficial.
`data_2017_01_20.csv`
or add a version
`data_2017_01_20_v2.csv`

Do not use:
`DontUseMixedCases.R`
`periods.are.meh.but.ok.csv`
`dont_use.periods_and.underscore.R`

Definitely don't use a name that only you can understand
`X10_20.16_T_AND_Y410.csv`

Be consistent within and across projects. For example, all data file names contain the same information:
`datasource_briefdescription_firstyear_lastyear.csv`
Examples:
`survey_bio_1988_2016.csv` or
`fishery_cpue_1998_2016.csv`

Object names

Rename all columns on import of a dataset. Do this at the beginning of a script to make sure that files can be joined without naming conflicts.

Keep names short, but descriptive.

Use:
`year` or `Year`
`cpue` or `std.cpue`

```
No spaces
No_Upper_And_Upper_Case
NO_ALL_CAPS
```

`catch` works better than `c` and you can search for and change `catch` (plus `c` is a function command…)

Know your data types. If an object is a character or factor use a capital letter, if it is numeric (double, or integer) use a lower case name. Define data types at the beginning of a script.

For example you have the integer `year` in your data but you create figures based on the factor `Year`. If you keep the two seperate you have the ability to easily plot and run various analyses without getting errors plus:

`lm(catch~year)` is quite a bit different than `lm(catch~Year)`

Instead of writing `catch_kg` make a note at the beginning of the script that catch is in kg unless otherwise stated then use `catch`. Making notes in your code files is just good practice in general. It allows others to more easily follow your code.

Dates
Believe it or not there is an international date/time standard (ISO 8601) the format is:
yyyy-mm-dd

Use it! Dates regularly have confounding errors - clean them at the beginning of a script and make all formats consistent (because what people send you will not be).

Function names

It may be helpful to start function names with an *f* at the beginning, to clearly identify them as a function.

`f_brief_function_description`

Something like `f_ricker` or `ricker_fun` to name a Ricker spawner-recruit function. Functions named *r*, *rick*, or *sr* don't tell you what the function does - be at least slightly descriptive.

Assignments
Use `<-`, not =
Use TRUE and FALSE, not T or F (the latter can be reassigned, the former cannot).

The `<-` assignment shortcut is `Alt-`, the "pipe" `%>%` operator shortcut is `Cntrl+Shift+M` more shortcuts in Rstudio can be found by pressing `Alt+Shift+K`.

Spacing

Place spaces around all binary operators (=, +, -, `<-`, etc.). Do not place a space before a comma, but always place one after a comma. Place a space before left parenthesis, except in a function call. There should be a hard return after each pipe `%>%`.

Organization

Projects

Use them!
Don't save your work environment - you should rerun your code each time to make sure you haven't broken anything. If the analyses are lengthy or complex, then save the output - which can then be sourced. In Rstudio you can go to "Tools > Global Options> Save workspace to .Rdata on exit" change it to "Never" and your workspace will not be saved.
Don't use absolute paths no `set_wd()` or `attach` in your scripts - if data are confidential then write a note in the script of how to call the data (e.g., OceanAK), or where it is stored on a ADF&G server, or whom to contact. Why does this matter? Your `set_wd()` is not the same as mine. Use relative paths only - hence the reason for projects. **Only load packages that you are actually using.**

Project structure

If everyone uses a similar structure for projects and scripts we will be able to read and understand each other's work faster and more easily.
A project should have a number of folders:

- code - all scripts (use separate folders for r-code, admb, etc.)
- data - raw data - this gets treated as a read only file, never adjust the contents of this file - everything is done via code
- figs - this is .png etc., output
- text - if writing a document it is easier to keep it separate

sometimes there may be inclusion of a few other folders, such as:

- output - any cleaned data
- literature

This structure works well for developing R scripts, writing in markdown and can work well when writing with Sweave.

Discussions have arisen as to whether multiple year assessments are better set up within each folder or on the outside of each folder. There are caveats to both.

Example of assessment organization inside the code folder:

- code
  - 2022_forecast
  - 2023_forecast
- data
  - 2022_forecast
  - 2023_forecast

Example of assessment organization outside the code folder:

- 2022_forecast

  - code
  - data
  - figs
  - text

- 2023_forecast

  - code
  - data
  - figs
  - text

Scripts

The general structure of a script should have:

- description of what it does
- who created it and their contact e-mail
- further notes on data access, units, etc.
- list all libraries used in the analysis - have libraries you no longer use? **delete them** it will reduce conflicts.
- Typically it is best to bring in all data at the start and clean it. If the files are huge, or numerous, there may be a code chunk(s) deeper into the script to deal with it.
- fold your code!
  Use `# load ----` or `# data ----` to create breaks and make it easy to navigate within your scripts and make lengthy analyses much easier to follow.

At a minimum a script structure should look like:

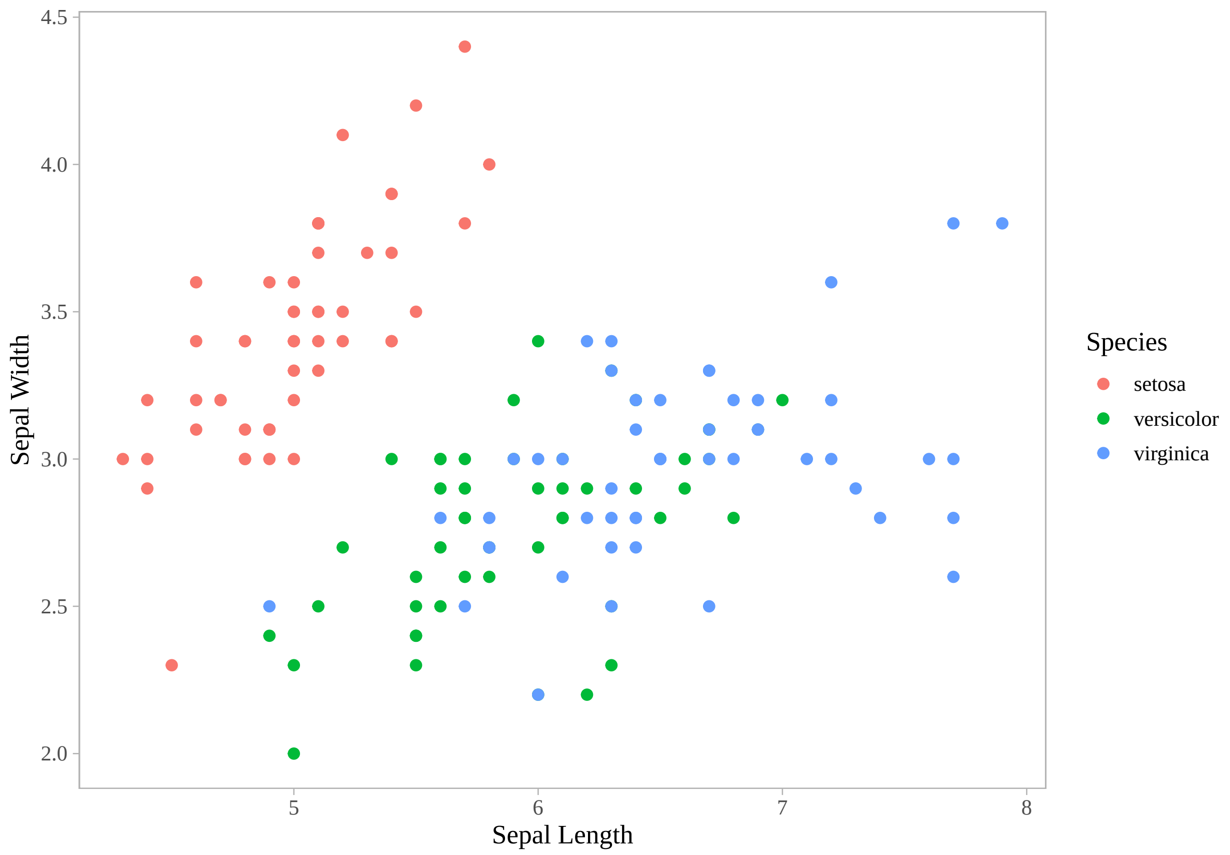
```
# notes ----

# author
# contact
# date (or last changed date)

# load ----

library()
source('code/functions.R')

# data ----
data <- read_csv('data/data_file.csv')


data %>%
    mutate(Year = factor(year)) -> data


# analysis ----
```

For example:

```r
# notes ----
# This is a demonstration of how scripts should be setup
# Author: Ben Williams
# contact: ben.williams@alaska.gov
# Last edited: 2017-7-7

# load ----
library(tidyverse)
devtools::install_github("commfish/fngr")
library(fngr)
library(extrafont)
theme_set(theme_report(base_size = 10))


# data ----
# typically this would be read_csv("data/iris.csv")
# but the iris dataset is built in

# change names
names(iris) <- c('sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'Species')
glimpse(iris)
```

```
## Rows: 150
## Columns: 5
## $ sepal_length <dbl> 5.1, 4.9, 4.7, 4.6, 5.0, 5.4, 4.6, 5.0, 4.4, 4.9, 5.4, 4.~
## $ sepal_width  <dbl> 3.5, 3.0, 3.2, 3.1, 3.6, 3.9, 3.4, 3.4, 2.9, 3.1, 3.7, 3.~
## $ petal_length <dbl> 1.4, 1.4, 1.3, 1.5, 1.4, 1.7, 1.4, 1.5, 1.4, 1.5, 1.5, 1.~
## $ petal_width  <dbl> 0.2, 0.2, 0.2, 0.2, 0.2, 0.4, 0.3, 0.2, 0.2, 0.1, 0.2, 0.~
## $ Species      <fct> setosa, setosa, setosa, setosa, setosa, setosa, setosa, s~
```

```r
# eda ----

ggplot(iris, aes(sepal_length, sepal_width, color=Species)) +
  geom_point() +
    ylab('Sepal Width') +
  xlab('Sepal Length')
```



Rmarkdown

Note: This section is under development

What is markdown?

- a very lightweight writing format originally created for conversion to HTML (a "markup" language - that is a pain to write)
- it is plain text (no proprietary software - looking at you MS Word) so can be tracked via git

- what can it be converted to?
  pdf, word, html
- How does that work? via a program called Pandoc (a "universal" document converter) - don't worry it is included with RStudio.

When to use it

-Whenever you want to rapidly communicate annotated code (informal report) -Initial report writing
-To keep notes for yourself

Structure
* YAML - 'YAML Ain't Markup Language' a human (and computer) readable data format. This is the "frontmatter" of your report that tells markdown what type of output you would like to have.
* there are a slew of options http://rmarkdown.rstudio.com/html_document_format.html - I've generally found that keeping it rather basic e.g.,

```
---
title: 'This is a title'
author: 'Me'
date: "2017_06_10"
output:pdf_document
fontsize: 11pt
csl: canjfas.csl
bibliography: bibby.bib
---
```

This has a title, author and date that will be at the head of the document. I've told it to generate a pdf and have included a bibliography (.bibtex format) for references and am using the Canadian Journal or Fisheries and Aquatic Sciences .csl (citation style language) that format the references. Here is a good site for downloading .csl styles https://github.com/citation-style-language/styles

```
sessionInfo(c("ggplot2", "FNGr"))
```

```
## R version 4.1.2 (2021-11-01)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 19044)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=English_United States.1252
## [2] LC_CTYPE=English_United States.1252
## [3] LC_MONETARY=English_United States.1252
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.1252
##
## attached base packages:
## character(0)
##
## other attached packages:
## [1] ggplot2_3.3.6   fngr_0.0.0.9000
##
## loaded via a namespace (and not attached):
##  [1] httr_1.4.3        pkgload_1.2.4     tidyr_1.2.0       jsonlite_1.8.0
##  [5] modelr_0.1.8      brio_1.1.3        assertthat_0.2.1  highr_0.9
##  [9] grDevices_4.1.2   cellranger_1.1.0  yaml_2.2.2        remotes_2.4.2
## [13] sessioninfo_1.2.2 Rttf2pt1_1.3.10   pillar_1.7.0      backports_1.4.1
## [17] glue_1.6.1        base_4.1.2        extrafontdb_1.0   digest_0.6.29
## [21] rvest_1.0.2       colorspace_2.0-3  htmltools_0.5.2   pkgconfig_2.0.3
## [25] devtools_2.4.3    broom_0.8.0       haven_2.5.0       fngr_0.0.0.9000
## [29] purrr_0.3.4       scales_1.2.0      processx_3.5.2    tzdb_0.3.0
## [33] tibble_3.1.6      farver_2.1.0      generics_0.1.2    datasets_4.1.2
## [37] usethis_2.1.6     ellipsis_0.3.2    cachem_1.0.6      withr_2.5.0
## [41] tidyverse_1.3.1   cli_3.1.1         magrittr_2.0.2    crayon_1.5.1
## [45] readxl_1.4.0      memoise_2.0.1     evaluate_0.15     ps_1.6.0
## [49] methods_4.1.2     fs_1.5.2          fansi_1.0.3       forcats_0.5.1
## [53] xml2_1.3.3        utils_4.1.2       pkgbuild_1.3.1    tools_4.1.2
## [57] prettyunits_1.1.1 hms_1.1.1         lifecycle_1.0.1   stringr_1.4.0
## [61] munsell_0.5.0     reprex_2.0.1      stats_4.1.2       callr_3.7.0
## [65] compiler_4.1.2    rlang_1.0.2       grid_4.1.2        rstudioapi_0.13
## [69] graphics_4.1.2    labeling_0.4.2    rmarkdown_2.14    testthat_3.1.2
## [73] gtable_0.3.0      DBI_1.1.2         curl_4.3.2        R6_2.5.1
## [77] lubridate_1.8.0   knitr_1.39        dplyr_1.0.8       fastmap_1.1.0
## [81] extrafont_0.18    utf8_1.2.2        rprojroot_2.0.3   readr_2.1.2
## [85] desc_1.4.1        stringi_1.7.6     vctrs_0.4.0       dbplyr_2.1.1
## [89] tidyselect_1.1.2  xfun_0.30
```