HOPE CODE CONVENTION

# PROJECT STRUCTURE

* We want to make a separate folder structure
* File organization
  + Default project structure
    - Data
      * Input
        + Folder\_example\_1
        + Folder\_example\_2
        + Spatial data
      * Processed
      * Temp
    - renv
    - Outputs
      * Data
      * Figures
      * Tables
    - R
      * Functions
        + Data\_handling
        + Age\_depth\_models

count.ages.R

* + - * 00\_Config\_file.R
      * 00\_Master.R
      * 01\_[folder name]
        + Run\_01.R
        + 01-01\_Folder

Run\_01\_01.R

Script\_1 (“Download\_data.R”)

Script\_2 (“Attach\_Denv.R”)

Script\_3

* + - * 02\_[folder name]
        + Run\_02.R
        + 02-01\_Folder

Run\_02-01.R

Script\_1 (“blabla.R”)

Script\_2 (“blabla.R”)

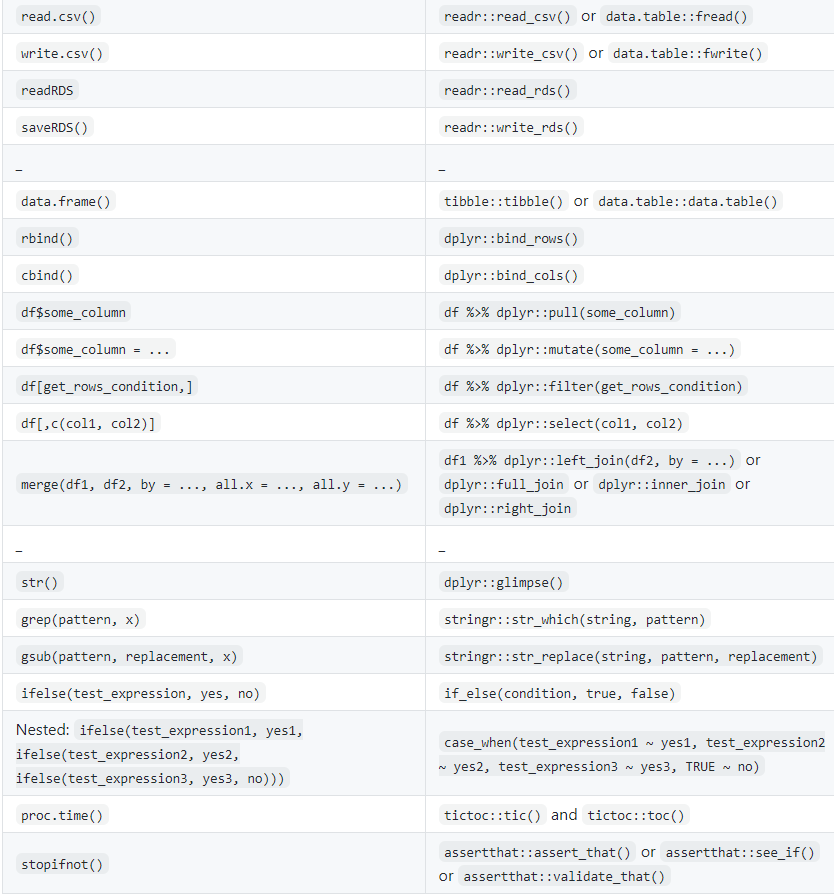
Script\_3

* + - gitignore
    - R project file
* Temp file
  + Save temporary data files
  + No links between Temp files and scripts on Github
  + Include in gitignore
* **Filenames**
  + Underscore
  + Only the first letter capitalized
  + Dates in file names
    - Do not have them
      * Exception is:
        + Processed data
        + Output data
    - Create using Sys.date() “yyyy-mm-dd” example ”data\_xx\_2021-01-11”
    - Create only if there is a change in the data
* Renv package to make sure that the libraries
* The packages are sorted in the order of their first use in the script.

# CODING

## Coding style:

TIDYVERSE: As much as possible



## Code structure

1. Script headers
   1. Decide on naming, rough date, objectives, etc
   2. default one

#----------------------------------------------------------#

#

#

# Project name

#

# Script name

# - continue

#

# Authors

# Year

#

#----------------------------------------------------------#

# purpose of the script

1. File structure
   1. The script is always partitioned into clearly readable chunks
   2. A brief title preceded by a ‘#’ is written between the two dashed lines each preceded by ‘#’
   3. Depending on the level in the document outline, different lengths of chunk titles

H1 #----------------------------------------------------------#

H2 #--------------------------------------------------#

H2 #----------------------------------------#

* 1. Double line space is maintained between two consecutive chunks. (check)
  2. All the chunk titles are written in lower-case letters and are separated from the chunk by single line space. (check)
  3. Number the chunks
     1. Make a default one

1. Inline comment: [space]#[space]comment
2. Single-line comments: #[space]comment
3. Multi-line comments:
   1. Follow the 80 character limit
   2. The next line of the comment would start with double space
4. Comments on a function -> see *function documentation?*

#### Errors:

* 1. Use **i** (contextual information) first and then **x** elements
  2. Provide bullet list error description
  3. If the cause of the problem is clear, use “must”
  4. If you cannot state what was expected, use “can’t”
  5. Avoid giving opinions
  6. Add below the location of the function where the error occurs

## NAMING

### Code Naming

1. Distinguish between objects and functions
   1. Naming objects:
      1. underscore\_separated (“snake\_style”)
      2. Use nouns
   2. Naming functions:
      1. Period.separated
      2. Use verbs
      3. Star with a period for internal functions? STUDY!
2. Preference of abbreviated / descriptive names?
   1. Help our future self!
   2. Preference for explanatory names

### Column/Variable Naming in tibbles

1. No use of periods
2. Always use lowercase and underscore

## TRUE/FALSE:

1. Always use TRUE and FALSE, instead of T and F

## Syntax

1. Spaces
   1. Always space after a comma
   2. No spaces inside or outside parentheses for regular function calls
   3. Infix operators (==, +, -, <-, ~, etc.) should always be surrounded by spaces
   4. Not surrounded by spaces: Operators with high precedence: ::, :::, $, @, [, [[, ^, unary -, unary +, and :.
2. Brackets
   1. Normal ( )
      1. Not on separate first and last line
      2. Before the bracket, always space *unless it’s a function*
      3. No space after/before brackets
   2. Square [ ]
      1. Space instead of missing value
      2. Never space before or after the bracket
   3. Curly { }
      1. Not used for chunks of code
      2. Use only for function
         1. { should the last character on a line and should never be on its own
         2. } should be the first character on a line
      3. Always new brackets after **else** unless followed by **if**
3. Line length: 80 characters rule
4. Indentation:
   1. Each parameter/variable in function on its own line
   2. For paste functions, keep it in one line (except if longer than 80 char)
5. Assignment
   1. Always **<-**, not **=** or **->** (ALT + “-”)
   2. Multi-line assignment: The assigned object on a separate line
   3. Single-line assignment: everything on one line
   4. Automatic line (devtools::install\_github("seasmith/AlignAssign"))
6. No use of semi-columns
7. Pipes (%>%) (CTRL + SHIFT + m )
   1. space before
   2. new line after

## Functions

1. Naming (see above)
2. Each function in a separate script
   1. Script name should be the same as the function
   2. There should be only the function inside the script
3. Always state the variables in the function call
4. Return
   1. Always **return** in the end
5. Map
   1. Use tidle (~) for change in map default values in the function
   2. pmap
      1. Use ..1 , ..2 for calling data from the list
6. Libraries
   1. Always use “::”
   2. Libraries are loaded in the config. File
   3. Package libraries dependencies are stated in the DESCRIPTION file

### Function documentation

Each function should have documentation in the beginning of the function (see example)

#### EXAMPLE 1

.calculate.ksss <- function(centroids, statistical\_input, time\_column = "schoolyr", location\_column = "school\_dist", value\_column = "absences\_ill", population\_column = "student\_days", k\_nearest\_neighbors = 5, nsim = 9999, heat\_map = TRUE, heat\_map\_title = NULL, heat\_map\_caption = NULL) {

#' **@title** Calculates the population-based Kulldorff Spatial Scan Statistic

#' **@param centroids** an SPDF from which the centroids of each school catchment area can be drawn, along with a uniquely identifying location\_column (such as an ID or School-District combo)

#' **@param statistical\_input** a tibble of at least 4 columns containing a location, time, value, and population size, ordered by (location\_column, time\_column)

#' **@param time\_column** an integer or string column on which values can be clustered temporally

#' **@param location\_column** an integer or string column on which values can be clustered spatially (must be a unique key)

#' **@param value\_column** an integer or string column containing the count data for the given space-time

#' **@param population\_column** an integer column containing the size of the population for the given space-time

#' **@param heat\_map** a boolean variable dictating whether to plot a heat\_map based on how likely a school catchment is to be part of a cluster

#' **@param heat\_map\_title** a string used as the title for a heat\_map if one is drawn

#' **@param heat\_map\_caption** a string used as the caption for a heat\_map if one is drawn

#' **@return** a list containing the results of running KSSS and a tibble of all clusters

#' **@description** This is a description of the function (can be several sentences)

...

Some code here

...

}