Courses to work through:

* Introduction to R
* Intermediate R
* Introduction to writing Functions in R
* Pipes in R tutorial for Beginners
* Writing Efficient R Code
* Optimizing R Code with Rcpp
* Joining Data with data.table in R
* Exploratory data analysis in R
* Supervised Learning in R: Classification
* Introduction to Machine Learning
* Data Vizualisation in R
* Visualizing Geospatial Data in R
* Network Analysis in R
* Hierarchical and mixed effect models in R
* Generalized Linear Models in R
* Nonlinear Modeling in R with GAMs
* Survival Analysis in R
* Spatial Analysis with the sf and raste packages
* Spatial Statistics in R

# 1. Introduction to writing funcitons in R

args(function) #shows the arguments a function needs.

#first function

# Your functions, from previous steps

toss\_coin <- function(n\_flips, p\_head) {

coin\_sides <- c("head", "tail")

weights <- c(p\_head, 1-p\_head)

sample(coin\_sides, n\_flips, replace = TRUE, prob = weights)

}

# Call your function e.g. for 10 coin tosses with a probability of head of 80%

toss\_coin(10, 0.8)

Normal glm which is considered a bad function (because it is old)

glm(

n\_visits ~ gender + income + travel,

data = snake\_river\_visits,

family = poisson)

# write function to improve it (include verb in name, data first and then formula)

run\_poisson\_regression <- function(data, formula) {

glm(formula, data, family = poisson)

}

model <- snake\_river\_visits %>%

run\_poisson\_regression(n\_visits ~ gender + income + travel)

library(assertive) #contains a lot of variable checks for functions such as assert\_is\_numeric()

# Thus we can write functions, with error messages if a user provides a wrong input: Error, Meaningful error message and last try to fix it if possible: Final example function

# Update the function definition to fix the na.rm argument

calc\_harmonic\_mean <- function(x, na.rm = FALSE) {

assert\_is\_numeric(x)

if(any(is\_non\_positive(x), na.rm = TRUE)) {

stop("x contains non-positive values, so the harmonic mean makes no sense.")

}

# Use the first value of na.rm, and coerce to logical

na.rm <- coerce\_to(use\_first(na.rm), target\_class = "logical")

x %>%

get\_reciprocal() %>%

mean(na.rm = na.rm) %>%

get\_reciprocal()

}

# See what happens when you pass it malformed na.rm

calc\_harmonic\_mean(std\_and\_poor500$pe\_ratio, na.rm = 1:5)

Getting multiple outputs of a function: Usually only one return.. but this can be avoided by creating a list or using the package zealot which will unpack it. Another version would be to use attributes. (load broom and zealot)

# From previous step

groom\_model <- function(model) {

list(

model = glance(model),

coefficients = tidy(model),

observations = augment(model)

)

}

# Call groom\_model on model, assigning to 3 variables

c(mdl, cff, obs) %<-% groom\_model(model)

# See these individual variables

mdl; cff; obs

#the syntax for assigning attributes is as follows: attr(object, "attribute\_name") <- attribute\_value

pipeable\_plot <- function(data, formula) {

plot(formula, data)

# Add a "formula" attribute to data

attr(data, "formula") <- formula

invisible(data)

}

# From previous exercise

plt\_dist\_vs\_speed <- cars %>%

pipeable\_plot(dist ~ speed)

# like this we can inclue metadata in the return value by storing it as an attribute

ENVIRONMENTS

#an environment is a variable (kind of list) that is used to store other variables. There are several levels of environments (parent environments).

# Add capitals, national\_parks, & population to a named list

rsa\_lst <- list(

capitals = capitals,

national\_parks = national\_parks,

population = population

)

# List the structure of each element of rsa\_lst

ls.str(rsa\_lst)

# Convert the list to an environment

rsa\_env <- list2env(rsa\_lst)

# List the structure of each variable

ls.str(rsa\_env)

# Find the parent environment of rsa\_env

parent <- parent.env(rsa\_env)

# Print its name

environmentName(parent)

# Does population exist in rsa\_env? (does a variable exist in an environment?

exists("population", envir = rsa\_env)

# Does population exist in rsa\_env, ignoring inheritance?

exists("population", envir = rsa\_env, inherits = FALSE)