Sim Study Template

Phoebe Meyerson

Fall 2023

This document contains a template to help with setting up your simulation studies.

Load packages

```
library(here)
```

```
## here() starts at /Users/phoebemeyerson/Desktop/SMA/thesis
```

```
library(mapview)
library(mase)
library(sf)
```

```
## Linking to GEOS 3.11.0, GDAL 3.5.3, PROJ 9.1.0; sf_use_s2() is TRUE
```

```
library(sfdep)
library(purrr)
library(survey)
```

```
## Loading required package: grid
```

```
## Loading required package: Matrix
```

```
## Loading required package: survival
```

```
##
## Attaching package: 'survey'
```

```
## The following object is masked from 'package:graphics':
##
##     dotchart
```

```
library(SUMMER)
```

```
## SUMMER version 1.3.0
##   See latest changes with 'news(package = 'SUMMER')'
```

```
library(INLA)
```

```
## Loading required package: sp

## This is INLA_23.11.01 built 2023-11-01 19:16:01 UTC.
##  - See www.r-inla.org/contact-us for how to get help.
##  - List available models/likelihoods/etc with inla.list.models()
##  - Use inla.doc(<NAME>) to access documentation
```

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.3     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.4     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.0

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x tidyr::expand() masks Matrix::expand()
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## x tidyr::pack()   masks Matrix::pack()
## x tidyr::unpack() masks Matrix::unpack()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

```
library(Matrix)
library(matrixcalc)
```

Data

```
# Truth
truth <- readRDS(here("data/simdata.rds")) %>%
  mutate(tnt = case_when((tnt == 2) ~ 0,
                         (tnt == 1) ~ 1)) %>%
  rename(tcc = tcc16)

# Load data
dat_all_reps <- readRDS(here("data/sample.rds")) %>%
  mutate(tnt = case_when( # refactor tnt as indicator variable
    tnt == 1 ~ 1,
    tnt == 2 ~ 0
  )) %>%
  mutate(
    DOMAIN = SUBSECTION,
    PROVINCE = "M333"
  ) %>%
  rename(tcc = tcc16)
```

```r
# Set up data so that easy to access samples for each iteration of simulation
dat_list <- dat_all_reps %>%
  group_by(rep) %>%
  group_split()

# set up xpop as needed for different SAE packages
xpop <- read_csv(here("data/m333.csv"), show_col_types = FALSE) %>%
  rename(SUBSECTION = MAP_UNIT_S)

# load ecomap shapefile
ecomap <- readRDS(here("data/ecomap.rds")) %>%
  filter(PROVINCE == "M333")
```

```r
# PG's functions

# PG's code for Horvitz-Thompson
get_direct <- function(formula, by, sample_des, CI = 0.95) {
  res <- svyby(formula, by, design = sample_des, svymean, na.rm = T)
  out_dat <- data.frame(
    region = as.vector(res[[all.vars(by)[1]]]),
    est = as.vector(model.matrix(formula, res)[, 2]),
    var = res$se ^ 2) %>%
    mutate(
    lower = est + qnorm((1-CI)/2) * res$se,
    upper = est + qnorm(1 - (1-CI)/2) * res$se,
    method = "HT")
  return(out_dat)
}

# My code for PS
# get_ps <- function(data, sample_des, strata_var, CI = 0.95) {
#
# }


# PG's code for GREG
get_greg <- function(working_fit, formula, by,
                     pop_dat, sample_des, CI = 0.95) {
  pop_unit_ests <- as.vector(predict(working_fit, pop_dat,
                                     type = "response"))
  area_ests <-
    aggregate(pop_unit_ests,
              list(region = as.vector(pop_dat[[all.vars(by)[1]]])),
              mean)
  colnames(area_ests)[2] <- "working_est"
  sample_des$variables$res <-
    sample_des$variables[[all.vars(formula)[1]]] -
    as.vector(predict(working_fit, sample_des$variables, type = "response"))
  sample_des$variables$region <-
    as.vector(sample_des$variables[[all.vars(by)[1]]])
  res_ht <- svyby(~res, ~region, sample_des, svymean)
  out_dat <- left_join(area_ests, res_ht, by = "region")
  out_dat$est = out_dat$working_est + out_dat$res
```

```
  out_dat$var = out_dat$se ^ 2
  out_dat$method = "GREG"
  out_dat$lower = out_dat$est + qnorm((1-CI)/2) * out_dat$se
  out_dat$upper = out_dat$est + qnorm(1 - (1-CI)/2) * out_dat$se
  out_dat <- dplyr::select(out_dat, region, est, var, lower, upper, method)
  return(out_dat)
}

# PG's code for SMA
get_bym2_sdir <- function(direct_est,
                                adj_mat,
                                pc_u = 5, # play around with these numbers
                                pc_alpha = 0.01,
                                pc_u_phi = 0.5,
                                pc_alpha_phi = 2/3,
                                CI = .95) {
  hyperpc_bym_int <- list(
    prec = list(prior = "pc.prec", param = c(pc_u , pc_alpha)),
    phi  = list(prior = 'pc', param = c(pc_u_phi , pc_alpha_phi))
  )
  sd_dat <- direct_est %>%
    mutate(est = ifelse(est != 0 & est != 1 & var > 1e-5, est, NA)) %>%
    mutate(prec = 1 / var,
           region = match(region, rownames(adj_mat)))
  sd_fit <-
    INLA::inla(est ~ f(region, model = "bym2",
                       graph = adj_mat,
                       hyper = hyperpc_bym_int,
                       scale.model = TRUE),
               family = "gaussian", data = sd_dat,
               scale = sd_dat$prec,
               control.family =
                 list(hyper = list(prec = list(initial= log(1), fixed= TRUE))),
               control.predictor = list(compute = TRUE),
               control.compute=list(config = TRUE))

  sd_fit_sample <-
    inla.posterior.sample(n = 1000, sd_fit,
                          list(region = 1:nrow(adj_mat), "(Intercept)" = 1))
  sd_est_mat <-
    do.call(cbind, lapply(sd_fit_sample,
                          function(x) x$latent[1:nrow(adj_mat)] +
                            x$latent[nrow(adj_mat) + 1]))
  out_dat <- data.frame(region = rownames(adj_mat),
                        est = rowMeans(sd_est_mat),
                        median = apply(sd_est_mat, 1,
                                       function(x) median(x, na.rm = T)), # get rid of ?
                        var = apply(sd_est_mat, 1, var),
                        lower = apply(sd_est_mat, 1,
                                      function(x) quantile(x, (1-CI)/2)),
                        upper = apply(sd_est_mat, 1,
                                      function(x) quantile(x, 1-(1-CI)/2)),
                        method = paste0("bymS", direct_est$method[1]))
```

```r
  return(out_dat)
}
```

```r
# USING ONE SAMPLE
svy_dat <- dat_list[[1]]

# add column of weights to svy_dat
# LATER: figure out how to do this quickly for every sample
counts.N <- truth %>%
  group_by(SUBSECTION) %>%
  count()

weights.n <- svy_dat %>%
  group_by(SUBSECTION) %>%
  count() %>%
  ungroup() %>%
  mutate(weights = counts.N$n / n) %>%
  dplyr::select(SUBSECTION, weights)

svy_dat <- left_join(svy_dat, weights.n, by = "SUBSECTION")

# making sample_des
sample_des <- svydesign(id = ~1,
                        data = svy_dat,
                        weights = ~weights)

# HT
HT_est <- get_direct(~DRYBIO, ~SUBSECTION, sample_des)

# working_fit
working_fit <- svyglm(DRYBIO ~ tcc + tnt, sample_des)

# pop_dat
pop_dat <- truth %>%
  dplyr::select(SUBSECTION, DRYBIO, tcc, tnt)

pop_dat_split <- pop_dat %>%
  group_by(SUBSECTION) %>%
  group_split()

# GREG
GREG_est <- data.frame(region = c(),
                       est = c(),
                       var = c(),
                       lower = c(),
                       upper = c(),
                       method = c())
for (i in 1:23) {
  GREG_i <- get_greg(working_fit, ~DRYBIO, ~SUBSECTION,
                     pop_dat_split[[i]], sample_des, CI = 0.95)
  GREG_est<- rbind(GREG_est, GREG_i)
}
```

```r
mGREG_est <- get_greg(working_fit, ~DRYBIO, ~SUBSECTION, pop_dat, sample_des, CI = 0.95)

## NOTE: mGREG and GREG estimates are the same... fishy
```

```r
# setting up
PS_est <- data.frame(region = c(),
                     est = c(),
                     var = c(),
                     lower = c(),
                     upper = c(),
                     method = c())

domains <- unique(xpop$SUBSECTION)
D <- length(domains)

# fitting PS using mase
for (i in 1:D) {
  xpop_d <- filter(xpop, SUBSECTION == domains[i])
  samp_d <- filter(svy_dat, SUBSECTION == domains[i])
  xpop_d_ps <- xpop_d %>%
    rename(tnt0 = tnt.2, tnt1 = tnt.1) %>%
    dplyr::select(tnt0, tnt1) %>%
    pivot_longer(everything(), names_to = "tnt", values_to = "prop") %>%
    mutate(tnt = parse_number(tnt))

  PS <- postStrat(y = samp_d$DRYBIO, N = xpop_d$npixels,
                  xsample = samp_d$tnt, xpop = xpop_d_ps,
                  datatype = "means",
                  var_est = TRUE,
                  var_method = "SRSunconditional")
  PS_est_i <- data.frame(
    region = domains[i],
    est = PS$pop_mean,
    var = PS$pop_mean_var,
    lower = PS$pop_mean - 1.96*sqrt(PS$pop_mean_var),
    upper = PS$pop_mean + 1.96*sqrt(PS$pop_mean_var),
    method = "PS"
  )
  PS_est <- rbind(PS_est, PS_est_i)
}
```

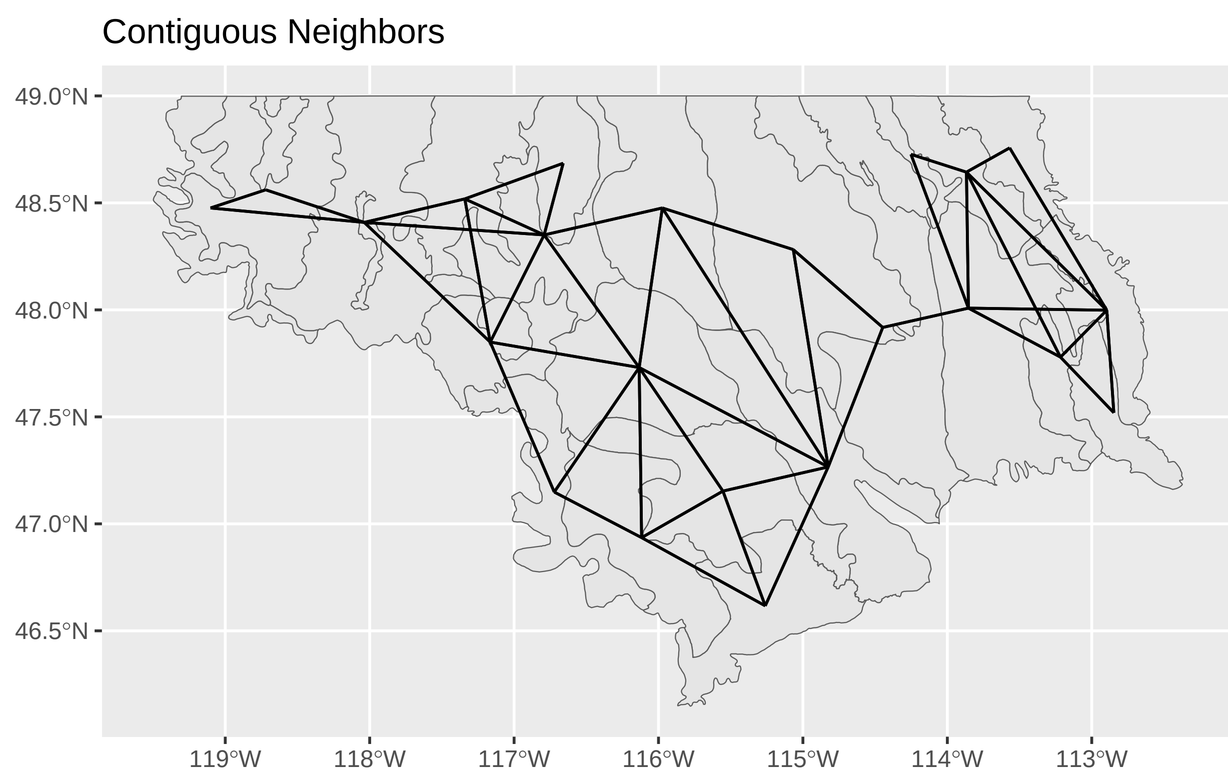
```
## Assuming simple random sampling
## Assuming simple random sampling
## Assuming simple random sampling
## Assuming simple random sampling
## Assuming simple random sampling
## Assuming simple random sampling
## Assuming simple random sampling
## Assuming simple random sampling
## Assuming simple random sampling
## Assuming simple random sampling
## Assuming simple random sampling
## Assuming simple random sampling
```

```
## Assuming simple random sampling
## Assuming simple random sampling
## Assuming simple random sampling
## Assuming simple random sampling
## Assuming simple random sampling
## Assuming simple random sampling
## Assuming simple random sampling
## Assuming simple random sampling
## Assuming simple random sampling
## Assuming simple random sampling
## Assuming simple random sampling
```

Figuring out adjacency matrices

```r
### FROM LATTICE PROCESSES TUTORIAL 141
# using ecomap data
# mapview(ecomap)

# contiguous neighbors
m333_contig <- st_contiguity(ecomap)
m333_contig_sf <- st_as_edges(st_centroid(ecomap$geometry), nb = m333_contig)
ggplot() + geom_sf(data = ecomap) +
  geom_sf(data = m333_contig_sf) + ggtitle("Contiguous Neighbors")
```



```r
m333_adj <- wt_as_matrix(m333_contig, st_weights(m333_contig, style = "B"))
rownames(m333_adj) <- GREG_est$region

# trying out SMA
SMA_est <- get_bym2_sdir(GREG_est, m333_adj)
```

```
## Warning in inla.model.properties.generic(inla.trim.family(model), mm[names(mm) == : Model 'bym2' in s
##   Use this model with extra care!!! Further warnings are disabled.
```

```r
# ALL ESTIMATES
# truth
true_est <- truth %>%
  group_by(SUBSECTION) %>%
  summarize(truth = mean(DRYBIO))
# HT
HT_est
```

```
##    region      est       var     lower    upper method
## 1  M333Aa 36.63821 10.055910 30.422957 42.85346     HT
## 2  M333Ab 40.53566 18.364593 32.136449 48.93488     HT
## 3  M333Ac 36.14279 11.968798 29.362106 42.92347     HT
## 4  M333Ad 49.22270  7.597315 43.820407 54.62499     HT
## 5  M333Ae 31.33578  2.651934 28.144025 34.52753     HT
## 6  M333Ag 36.32860  3.848562 32.483591 40.17361     HT
## 7  M333Ah 11.56166  3.267875  8.018583 15.10474     HT
## 8  M333Ai 13.64890  3.247709 10.116767 17.18103     HT
## 9  M333Ba 51.18909  6.058138 46.364977 56.01321     HT
## 10 M333Bb 38.77212  2.632757 35.591923 41.95231     HT
## 11 M333Bc 18.33333  2.758759 15.077930 21.58874     HT
## 12 M333Ca 26.82098  9.502565 20.779153 32.86281     HT
## 13 M333Cb 36.28635  2.868123 32.967042 39.60565     HT
## 14 M333Cc 37.37491 33.037614 26.109356 48.64046     HT
## 15 M333Ce 27.78642  9.279032 21.816075 33.75676     HT
## 16 M333Cf 23.80772  8.237663 18.182363 29.43308     HT
## 17 M333Cg 15.03524  4.222602 11.007711 19.06276     HT
## 18 M333Ch 19.82480  6.646974 14.771679 24.87793     HT
## 19 M333Da 68.88083 11.608299 62.203045 75.55861     HT
## 20 M333Db 55.90808  9.492561 49.869433 61.94673     HT
## 21 M333Dc 41.42926  5.000649 37.046364 45.81216     HT
## 22 M333Dd 53.05138 15.785491 45.264251 60.83850     HT
## 23 M333De 55.72630 11.249812 49.152440 62.30017     HT
```

```r
# PS
PS_est
```

```
##    region      est       var     lower    upper method
## 1  M333Aa 34.77092  8.743130 28.975442 40.56641     PS
## 2  M333Ab 40.96769 17.265677 32.823496 49.11187     PS
## 3  M333Ac 36.99064 10.901646 30.519186 43.46210     PS
## 4  M333Ad 49.60430  7.376948 44.280836 54.92777     PS
## 5  M333Ae 31.48679  2.137496 28.621238 34.35235     PS
## 6  M333Ag 35.57435  3.615519 31.847507 39.30120     PS
```

```
## 7  M333Ah 12.08356  2.915981  8.736616 15.43050     PS
## 8  M333Ai 14.02586  2.345010 11.024427 17.02729     PS
## 9  M333Ba 49.47426  5.530668 44.864859 54.08367     PS
## 10 M333Bb 38.69476  2.479213 35.608640 41.78088     PS
## 11 M333Bc 16.15615  1.488280 13.765045 18.54725     PS
## 12 M333Ca 25.34087  8.151466 19.744916 30.93682     PS
## 13 M333Cb 36.56100  2.627798 33.383745 39.73825     PS
## 14 M333Cc 37.11899 33.698977 25.741024 48.49695     PS
## 15 M333Ce 28.35411  8.586680 22.610711 34.09750     PS
## 16 M333Cf 20.79167  6.050684 15.970437 25.61291     PS
## 17 M333Cg 15.86447  3.824543 12.031412 19.69754     PS
## 18 M333Ch 20.60768  6.340583 15.672298 25.54306     PS
## 19 M333Da 67.38454 11.043741 60.871047 73.89804     PS
## 20 M333Db 56.27944  9.361445 50.282532 62.27635     PS
## 21 M333Dc 41.21328  4.656444 36.983840 45.44273     PS
## 22 M333Dd 54.00179 15.594993 46.261654 61.74193     PS
## 23 M333De 56.88004 10.518140 50.523426 63.23665     PS
```

```
# GREG
GREG_est
```

```
##    region      est       var     lower    upper method
## 1  M333Aa 35.37502  6.123104 30.525108 40.22493   GREG
## 2  M333Ab 43.27051 12.327370 36.389007 50.15201   GREG
## 3  M333Ac 39.69848  7.288227 34.407219 44.98974   GREG
## 4  M333Ad 49.53384  4.802812 45.238513 53.82916   GREG
## 5  M333Ae 29.24076  1.505550 26.835870 31.64565   GREG
## 6  M333Ag 36.84156  2.818073 33.551344 40.13177   GREG
## 7  M333Ah 13.80800  1.767387 11.202368 16.41364   GREG
## 8  M333Ai 12.53917  2.568208  9.398204 15.68014   GREG
## 9  M333Ba 49.56533  3.656188 45.817650 53.31301   GREG
## 10 M333Bb 36.83746  1.624571 34.339311 39.33560   GREG
## 11 M333Bc 17.75329  1.181825 15.622579 19.88400   GREG
## 12 M333Ca 26.10522  5.300583 21.592800 30.61765   GREG
## 13 M333Cb 36.74821  1.717534 34.179582 39.31683   GREG
## 14 M333Cc 34.35416 20.369493 25.508340 43.19998   GREG
## 15 M333Ce 27.11231  5.302575 22.599040 31.62558   GREG
## 16 M333Cf 19.60945  2.622926 16.435196 22.78370   GREG
## 17 M333Cg 14.84707  2.912415 11.502239 18.19191   GREG
## 18 M333Ch 22.03307  3.636281 18.295608 25.77053   GREG
## 19 M333Da 67.60443  8.189154 61.995662 73.21320   GREG
## 20 M333Db 57.82573  6.771147 52.725619 62.92583   GREG
## 21 M333Dc 41.09433  3.129402 37.627128 44.56153   GREG
## 22 M333Dd 56.31880 10.061166 50.101924 62.53568   GREG
## 23 M333De 56.19302  7.574608 50.798804 61.58723   GREG
```

```
#SMA
SMA_est
```

```
##    region      est   median       var     lower    upper   method
## 1  M333Aa 35.42676 35.48580  6.009605 30.376326 40.17210 bymSGREG
## 2  M333Ab 42.80217 42.86466 10.825671 36.106296 49.35853 bymSGREG
## 3  M333Ac 39.54996 39.57475  7.601116 34.174963 44.96416 bymSGREG
```

```
## 4  M333Ad 49.09619 48.98073  4.687402 44.995937 53.16843 bymSGREG
## 5  M333Ae 29.34909 29.33759  1.543513 26.885540 31.71898 bymSGREG
## 6  M333Ag 36.77444 36.76708  2.977821 33.507221 40.41001 bymSGREG
## 7  M333Ah 14.02724 13.99790  1.756850 11.418493 16.68423 bymSGREG
## 8  M333Ai 12.96947 12.98680  2.483861  9.828941 16.13329 bymSGREG
## 9  M333Ba 49.27121 49.32348  3.329775 45.792692 52.92253 bymSGREG
## 10 M333Bb 36.88531 36.89080  1.653158 34.278818 39.51396 bymSGREG
## 11 M333Bc 17.92474 17.94966  1.138115 15.852046 20.05720 bymSGREG
## 12 M333Ca 26.57881 26.59733  5.617921 22.063838 31.21742 bymSGREG
## 13 M333Cb 36.67712 36.68170  1.893519 33.893498 39.37760 bymSGREG
## 14 M333Cc 34.66997 34.61968 17.868327 26.893503 42.83639 bymSGREG
## 15 M333Ce 27.34083 27.35062  5.258684 22.866309 32.10630 bymSGREG
## 16 M333Cf 19.86645 19.90299  2.696168 16.645789 23.02759 bymSGREG
## 17 M333Cg 15.24812 15.25370  2.888492 11.909378 18.45610 bymSGREG
## 18 M333Ch 22.41168 22.32594  3.614210 18.926239 26.29318 bymSGREG
## 19 M333Da 65.89493 65.94534  7.980697 60.635380 71.48723 bymSGREG
## 20 M333Db 56.78948 56.80773  6.836756 51.792432 61.64635 bymSGREG
## 21 M333Dc 41.01916 41.02652  3.180820 37.604070 44.50132 bymSGREG
## 22 M333Dd 55.10211 55.13071  9.327656 48.996332 60.93522 bymSGREG
## 23 M333De 55.17968 55.10200  7.876775 49.810759 60.67153 bymSGREG
```

```
# SIM STUDY PSEUDOCODE
# get it working for a loop of 20
# store everything in a really long dataset (est, SE, upper, lower, method, rep)
  # each rep gets me 23 x 4 rows
# want to know: confidence interval coverage rates, empirical MSE (compare to SE^2),
  # percent relative bias of estimate and PRB of MSE estimator
  # do this by method and subsection
# make lots of boxplots w 23 points for each estimator

# after this, do some subsampling, consider fire scenario
  # look into how to make a new mini polygon within a subsection
  # play around with model selection, SMA hyperparameters, adjacency matrix construction
```

KM Summer Simulation Code: Ignore For Now

Create container(s) for storing simulation output

```
# store <- list()
#
# # Number of monte carlo samples
# store$B <- 10 # Set larger once you get your sim working
#
# # Number of domains
# store$domains <- unique(xpop$MAP_UNIT_S)
# store$D <- length(store$domains)
#
# # Number of estimators
# # (Will just do 2 for the template)
# store$n_est <- 2
```

```
#
# # Estimates
# store$estimates <- array(rep(NA, store$B*store$D*store$n_est),
#                          c(store$B, store$D, store$n_est))
#
# # Estimated SEs
# store$ses <- array(rep(NA, store$B*store$D*store$n_est),
#                          c(store$B, store$D, store$n_est))
#
# # CIs
# store$ci_lb <- array(rep(NA, store$B*store$D*store$n_est),
#                          c(store$B, store$D, store$n_est))
#
# store$ci_ub <- array(rep(NA, store$B*store$D*store$n_est),
#                          c(store$B, store$D, store$n_est))
```

Run simulation

```
#
# for(i in 1:store$B){
#   # Select/set sample
#   samp <- dat_list[[i]]
#
#   for(d in 1:store$D){
#
#     # For direct estimators, filter down to just the domain of interest
#     samp_d <- filter(samp, SUBSECTION == store$domains[d])
#     xpop_d <- filter(xpop, MAP_UNIT_S == store$domains[d])
#
#     # Fit estimators
#
#     # HT
#     HT <- horvitzThompson(y = samp_d$DRYBIO, N = xpop_d$npixels,
#                          var_est = TRUE)
#     # PS
#     xpop_d_ps <- xpop_d %>%
#       dplyr::select(tnt.1, tnt.2) %>%
#       pivot_longer(everything(), names_to = "tnt", values_to = "prop") %>%
#       mutate(tnt = parse_number(tnt)*10)
#     PS <- postStrat(y = samp_d$DRYBIO, N = xpop_d$npixels,
#                     xsample = samp_d$tnt, xpop = xpop_d_ps,
#                     datatype = "means",
#                     var_est = TRUE,
#                     var_method = "SRSunconditional")
#
#   # Store estimates, their SEs, and CIs
#     store$estimates[i, d, 1] <- HT$pop_mean
#     store$estimates[i, d, 2] <- PS$pop_mean
#
#     store$ses[i, d, 1] <- sqrt(HT$pop_mean_var)
#     store$ses[i, d, 2] <- sqrt(PS$pop_mean_var)
```

```
#
#     store$ci_lb[i, d, 1] <- HT$pop_mean - 1.96*sqrt(HT$pop_mean_var)
#     store$ci_lb[i, d, 2] <- PS$pop_mean - 1.96*sqrt(PS$pop_mean_var)
#
#     store$ci_ub[i, d, 1] <- HT$pop_mean + 1.96*sqrt(HT$pop_mean_var)
#     store$ci_ub[i, d, 2] <- PS$pop_mean + 1.96*sqrt(PS$pop_mean_var)
#   }
# }
```

Compute performance metrics

```
# Percent relative bias of estimators

# Percent relative bias of SEs

# Confidence interval coverage
```

Discussion

Create some graphs and tables of your results. Write 1-2 paragraphs to summarize your results with a focus on the over-arching goal of selecting the *best* SAE for FIA.