nla2007_census_data

Bryan Milstead

18 December 2017

Todo

- compare values to previous NLA / MRB1 census estimates (same buffer)

Introduction

This document outlines the processing steps involved in calculating estimates of the number of people living near the lakes from the 2007 National Lake assessment (NLA). Lake polygons were supplied by the 2007 NLA and population estimates were extracted from the 2000 and 2010 US census data. The 2000 data were processed at the block group level and the 2010 data were processed at both the block group and the more detailed block levels. The block level took considerably more time (order of magnitude) than the block group with very little improvement in accuracy (see discussion below)

The [get_census] function from the package [goatscape] (currently under development) were used to download the census data and linework for a preliminary buffer of 5000m around each lake. The spatial data for the 5k buffer was saved and used to extract estimates for additional smaller buffers.

Processing steps

- NLA lakes read from the NLA shapefile and converted to [sf] objects
- Lakes were then transformed to the Albers projection [sf::st_transform]
- Lakes were extracted individually from the NLA shapefile and buffered to 5000m [st::st_buffer] and the area of the original lake removed ([sf::st_difference])
- The [get_census] function (see code below) was used to:
  - determine the state(s) and county(s) of the lake
  - download the census data by year for the block or block groups by state and county ([censusapi] package)
  - download the TIGER linework ([tigris] package)
  - geoprocessing functions in package [sf] used to intersect the lake buffer and tiger linework
  - population estimated for each intersection polygon as: estimated_population = (total population in block/blockgroup) * (the area of the interection) / (area of the block/blockgroup)
  - the estimated population for the buffer (estimated_population) was the sum of the estimated population for all intersection polygon.
  - population density (est_pop_per_km2) is equal to the estimated_population/buffer area

```
#' get_census
#'
#' This function takes an input landscape, a user-defined land area as an
#' [sf] polygon and returns the total population for that area for a given USA decadal census.
#' The input polygon must be projected and must overlap a USA census area
#' Data are returned for years 2000, or 2010; The default is 2010
#' For 2000 only the census block group data are available.  For 2010 the block level information
#' is available but can take a long time to download for larger polygons.  By default the function
#' downloads the block group level for 2010 but you can request the more detailed block data.
#'
#' @param landscape A spatial polygon of class "sf" (see package [sf]). Note: the object must be projec
```

```r
#' @param year Which census year do you want to use? In format YYYY.  Currently the package supports ce
#' @param spatial keep the spatial data? Default = FALSE
#' @param level for 2000 level == "block_group" for 2010 choose level = "block" or level = "block_group
#' @param api_key Your Census API key. Obtain one at http://api.census.gov/data/key_signup.html
#' @param delete_zips Important:  if set to TRUE all zip files in the working directory will be deleted
#'
#' @export
#' @examples
#' get_census()
get_census <- function(landscape, year = 2010, spatial = FALSE, level = "block_group",
                       api_key = Sys.getenv("CENSUS_API_KEY"), delete_zips = FALSE) {

  # check for census api key (modified from tidycensus::get_decennial)
  if (Sys.getenv("CENSUS_API_KEY") != "") {
    api_key <- Sys.getenv("CENSUS_API_KEY")
  } else if (is.null(api_key)) {
    stop("A Census API key is required.  Obtain one at http://api.census.gov/data/key_signup.html.")
  }

  # check that level = "block_group" for year == 2000
  if (year == 2000 & level == "block") stop("Block level data not available for 2000 use: level = 'block

  # check that landscape is an sf object
  if (class(landscape)[1] != "sf") stop("The landscape input must be an [sf] object")

  # check that the landscape is projected
  if (is.na(sf::st_crs(landscape)$epsg) | is.na(sf::st_crs(landscape)$proj4string)) stop("The landscape

  # check year year == 2000 or year == 2010
  # load the tiger spatial data for counties
  # reproject county to Albers
  if (year == 2000) {
    if (!exists("county2000low")) {
      cty <- tigris::counties(cb = TRUE, year = 2000)
      cty <- sf::st_transform(sf::st_as_sf(cty), "+init=ESRI:102008")
      assign("county2000low", cty, envir = globalenv())
    }
    county <- county2000low
  } else if (year == 2010) {
    if (!exists("county2010low")) {
      cty <- tigris::counties(cb = TRUE, year = 2010)
      cty <- sf::st_transform(sf::st_as_sf(cty), "+init=ESRI:102008")
      assign("county2010low", cty, envir = globalenv())
    }
    county <- county2010low
  } else {
    stop("Year must == 2000 or 2010")
  }

  # save input crs for output
  input_crs <- sf::st_crs(landscape)

  # reproject input landscape to albers (even if already in albers: just as fast)
```

```r
landscape <- sf::st_transform(landscape, "+init=ESRI:102008") # reproject to North America Albers Equ

# get counties that landscape intersects
fips <- as.data.frame(dplyr::slice(county, unlist(sf::st_intersects(landscape, county))))

# check that fips has data.  If nrow == 0 then input polygon does not overlap tiger files (i.e. not i
if (nrow(fips) == 0) stop("The input polygon does not overlap USA Census boundaries or projection inf

# rename fields
fips <- dplyr::select(fips, state = STATE, county = COUNTY)

# create fips string for get_census
fips$regionin <- paste("state:", fips$state, "+county:", fips$county, sep = "")

# set up censusapi and tigris calls
if (year == 2000) {
  name <- "sf3"
  vars <- "P001001"
}

if (year == 2010) {
  name <- "sf1"
  vars <- "P0010001"
}

if (level == "block") region <- "block:*"

if (level == "block_group") region <- "block group:*"

# get the census data for total pop
pop <- list()
for (i in c(1:nrow(fips))) {
  pop[[i]] <- censusapi::getCensus(name = name, vintage = year, vars = vars, region = region, regionin
}

# rbind the pop data
tot_pop <- pop[[1]]
if (length(pop) > 1) for (i in c(2:length(pop))) tot_pop <- rbind(tot_pop, pop[[i]])
names(tot_pop)[5] <- "tot_pop"

# get the census blocks/block_groups for state(s) and county(s)-each year is slightly different
# year = 2000
if (year == 2000) {
  blk <- list()
  for (i in c(1:nrow(fips))) {
    blk[[i]] <- sf::st_as_sf(tigris::block_groups(
      state = as.numeric(fips$state[i]),
      county = as.numeric(fips$county[i]), year = year
    ))
    blk[[i]] <- dplyr::select(
      blk[[i]], state = STATEFP, county = COUNTYFP, tract = TRACTCE00, block.group = BLKGRPCE00,
      area_land = ALAND00, area_water = AWATER00, geometry
    )
  }
}

# year == 2010 & level == 'block_group'
if (year == 2010 & level == "block_group") {
  blk <- list()
  for (i in c(1:nrow(fips))) {
    blk[[i]] <- sf::st_as_sf(tigris::block_groups(
      state = as.numeric(fips$state[i]),
      county = as.numeric(fips$county[i]), year = year
    ))
    blk[[i]] <- dplyr::select(
      blk[[i]], state = STATEFP, county = COUNTYFP, tract = TRACTCE10, block.group = BLKGRPCE10,
      area_land = ALAND10, area_water = AWATER10, geometry
    )
  }
}

# year == 2010 & level == 'block'
if (year == 2010 & level == "block") {
  blk <- list()
  for (i in c(1:nrow(fips))) {
    blk[[i]] <- sf::st_as_sf(tigris::blocks(
      state = as.numeric(fips$state[i]),
      county = as.numeric(fips$county[i]), year = year
    ))
    blk[[i]] <- dplyr::select(
      blk[[i]], state = STATEFP, county = COUNTYFP, tract = TRACTCE10, block = BLOCKCE10,
      area_land = ALAND10, area_water = AWATER10, geometry
    )
  }
}

# rbind the blk
blocks <- blk[[1]]
if (length(blk) > 1) for (i in c(2:length(blk))) blocks <- rbind(blocks, blk[[i]])

# convert area_land and area_water to numeric
blocks$area_land <- as.numeric(blocks$area_land)
blocks$area_water <- as.numeric(blocks$area_water)

# join blocks and tot_pop
blocks <- merge(blocks, tot_pop, by = names(blocks)[1:4], all.x = TRUE)

# reproject blocks
blocks <- sf::st_transform(blocks, "+init=ESRI:102008")

# calc area of blocks-add to attributes
blocks$blk_area <- as.numeric(sf::st_area(blocks))

# create intersection of landscape and blocks
int <- sf::st_intersection(landscape, blocks)
```

```
  # calc area of intersection polygons
  int$int_area <- as.numeric(sf::st_area(int))

  # calc proportion of original block included in int polygons
  int$pro_blk <- round(int$int_area / int$blk_area, 4)

  # calc proportional population by block/block_group
  int$pro_pop <- round(int$tot_pop * int$pro_blk, 2)

  # calc proportion of original landscape included in intersection
  # note: landscapes that cross international borders or have missing data will have pro_ls < 1
  int$pro_ls <- round(sum(int$int_area) / as.numeric(sf::st_area(landscape)), 2)

  # add warning if int$pro_ls < 1
  if (round(sum(int$int_area) / as.numeric(sf::st_area(landscape)), 2) != 1) warning("Input landscape a

  # prepare output list
  out <- list()

  # if spatial = TRUE transform intersection to input_crs and add
  if (spatial) {
    out$census_data <- sf::st_transform(int, input_crs)
  } else {
    out$census_data <- as.data.frame(int)
  }

  # calc estimated pop for the input landscape
  out$est_population <- round(sum(int$pro_pop), 2)

  # calc input area in m2
  out$input_area_km2 <- round(as.numeric(sf::st_area(landscape) / 1000000), 2)

  # calc pop density
  out$est_pop_per_km2 <- round(out$est_population / out$input_area_km2, 1)

  # percent overlap between input landscape and census area
  out$percent_overlap <- round(sum(int$int_area) / as.numeric(sf::st_area(landscape)) * 100, 1)

  # add overlap warning
  if (out$percent_overlap != 100) out$warning <- "percent_overlap not equal 100%; input landscape parti

  # remove the zip files containing the census linework from the working directory
  junk <- dir(pattern = ".zip") # select the  zip files in the working directory:
  if (delete_zips == TRUE) file.remove(junk) # send them to oblivion

  # return the output
  return(out)
}
```

- the code used to download and process the data for the 5k buffer is shown below: repeating this could take many days.

```
library(sf)
library(goatscape) # install('C:/bryan/rscripts/goatscape')
```

```r
library(tigris)
library(tidyverse)
library(magrittr)
options(tigris_use_cache = TRUE)

# #preallocate lists for output
# grp2000<-vector(mode = "list", length = 1159)
# blk2010<-vector(mode = "list", length = 1159)
#
# #create df to track output
# samp<-data.frame(row=1:1159)
# samp$nla_id<-lakes$SITEID
# samp$pop2000<-NA
# samp$pop2010<-NA
# samp$time<-NA


# get nla lakes
lakes <- st_read("L:/Public/Milstead_Lakes/NLA_2007/GIS/gis_final/National_LakePoly.shp")

# convert to albers
lakes <- st_transform(lakes, "+init=ESRI:102008")

# load saved data (if continuing)
load(here::here("data/pop.rda"))

# get census data
todo <- which(is.na(samp$pop2000))
for (i in todo) {
  # for(i in c(668:1159)){
  # record start time
  start <- Sys.time()

  # select lake and buffer
  buf <- NA # remove old output
  buf <- tryCatch(st_difference(st_buffer(lakes[i, ], 5000), st_geometry(lakes[i, ])), error = function

  # get census 2000
  grp2000[[i]] <- tryCatch(
    get_census(buf, year = 2000, spatial = TRUE, level = "block_group", delete_zips = TRUE),
    error = function(e) NA
  )
  samp$pop2000[i] <- tryCatch(grp2000[[i]]$est_population, error = function(e) NA)

  # get census 2010
  blk2010[[i]] <- tryCatch(
    get_census(buf, year = 2010, spatial = TRUE, level = "block", delete_zips = TRUE),
    error = function(e) NA
  )
  samp$pop2010[i] <- tryCatch(blk2010[[i]]$est_population, error = function(e) NA)

  # calc processing time
  samp$time[i] <- difftime(Sys.time(), start, units = "min")
  # add counter and save
  print(i)
  print(Sys.time())
  plot(
    blk2010[[i]]$census_data[17],
    main = paste("i=", i, "; ", blk2010[[i]]$census_data$SITEID[1], "; pop=", round(samp$pop2010[i]), "
  )
  if (i %in% seq(9, 1159, 25)) save(grp2000, blk2010, samp, file = here::here("data/pop.rda"))
}

# save(grp2000, samp, file = here::here('data/grp2000.rda'))
# save(blk2010, samp, file = here::here('data/blk2010.rda'))


################

# lets do the block group data for 2010 also

# #preallocate lists for output
grp2010 <- vector(mode = "list", length = 1159)
# samp1<-samp
samp$time1 <- NA
samp$pop2010bg <- NA

# get census data
todo <- which(is.na(samp$pop2010bg))
for (i in todo) {
  # record start time
  start <- Sys.time()

  # select lake and buffer
  buf <- NA # remove old output
  buf <- tryCatch(st_difference(st_buffer(lakes[i, ], 5000), st_geometry(lakes[i, ])), error = function

  # get census 2010
  grp2010[[i]] <- tryCatch(
    get_census(buf, year = 2010, spatial = TRUE, level = "block_group", delete_zips = TRUE),
    error = function(e) NA
  )
  samp$pop2010bg[i] <- tryCatch(grp2010[[i]]$est_population, error = function(e) NA)

  # calc processing time
  samp$time1[i] <- difftime(Sys.time(), start, units = "min")

  # add counter and save
  print(i)
  print(Sys.time())
  plot(
    grp2010[[i]]$census_data[17],
    main = paste("i=", i, "; ", grp2010[[i]]$census_data$SITEID[1], "; pop=", round(samp$pop2010bg[i]),
  )
  if (i %in% seq(9, 1159, 25)) save(grp2010, samp, file = here::here("data/grp2010.rda"))
}
```

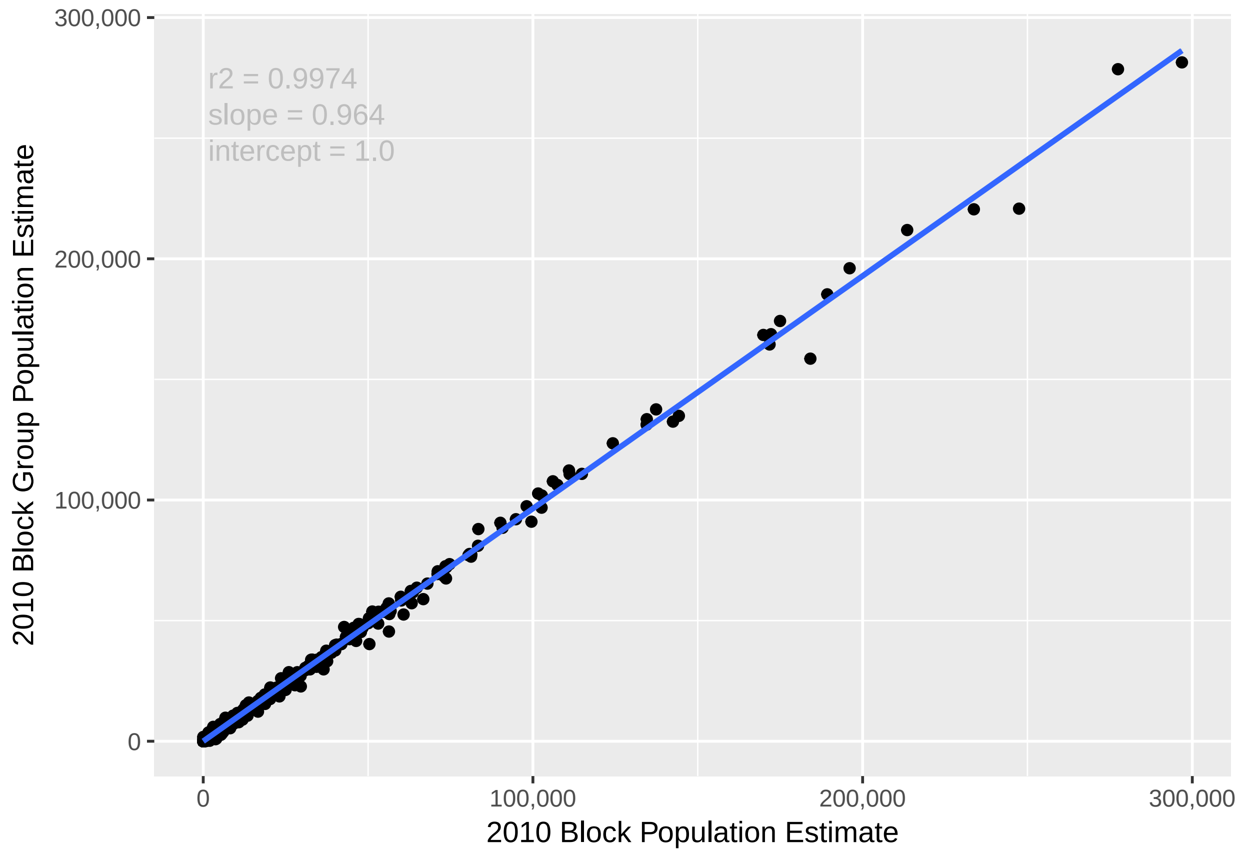
Analysis

- The output from above is too large for github so it will be moved to the directory "./data/too_big"
- Some QA/QC
  - Check that data were returned for all lakes in the same order: TRUE
  - to keep track of the processing a df named 'samp' was created with the population totals; recreate this from the census data. All TRUE
- Compare 2010 estimates from block and block group levels

```
##
## Attaching package: 'scales'

## The following object is masked from 'package:purrr':
##
##     discard

## The following object is masked from 'package:readr':
##
##     col_factor
```



waterbodydatabase has MRB1 lake population at 300, 1000, 2500