Gathering data

Chris Hoover et al

4/23/2021

Get data for seed populations

The first decision to be made is which household and person characteristics are to be included in the synthetic population. `Rsynthpops` uses functions from `tidycensus` to get PUMS data. `tidycensus` also ships with a dataset `pums_variables` where users can inspect potential variables to include in the PUMS download. Make sure to get and/or supply your census api key in order to use the `tidycensus` download functions. Below, we see the first few variables available for the 1-year 2018 PUMS data.

We'll generate data for San Francisco, CA as an example for this exercise.

```
state_use  = "CA"
fips_use   = "06075" # San Francisco county fips code
year_use   = 2019
survey_use = "acs5"
```

```
#census_api_key(my_key, install = TRUE) # Replace `my_key` with your census api key

pums_vars <- pums_variables %>%
  filter(year == year_use, survey == survey_use) %>%
  # The distinct call shows one row per variable but can be removed to show all levels of all variables
  distinct(var_code, var_label, data_type, level)

  head(pums_vars, n = 10)
```

```
## # A tibble: 10 x 4
##    var_code var_label                                          data_type level
##    <chr>    <chr>                                              <chr>     <chr>
##  1 RT       Record Type                                        chr       <NA>
##  2 SERIALNO Housing unit/GQ person serial number               chr       <NA>
##  3 DIVISION Division code based on 2010 Census definitions     chr       <NA>
##  4 PUMA     Public use microdata area code (PUMA) based on 201~ chr       <NA>
##  5 REGION   Region code based on 2010 Census definitions       chr       <NA>
##  6 ST       State code based on 2010 Census definitions        chr       <NA>
##  7 ADJHSG   Adjustment factor for housing dollar amounts (6 im~ chr       housi~
##  8 ADJINC   Adjustment factor for income and earnings dollar a~ chr       housi~
##  9 WGTP     Housing Unit Weight                                num       housi~
## 10 NP       Number of persons associated with this housing rec~ num       housi~
```

The `get_pums()` function from `tidycensus` automatically retains the `SERIALNO`, `SPORDER`, `WGTP`, `PWGTP`, and `ST` variables for the state or states from which data is requested, but anything else you want to include should be identified from the `var_code` variable in `pums_variables` and passed in the `variables` argument to `get_pums()`. We'll build on the `tidycensus` vignette on downloading and using PUMS data to build the person and household level target and seed data that will be used to generate the backbone of our synthetic population. **Note**: this might take a few minutes depending on the population size you're working with.

Once data has been downloaded for the entire state, it may need to be further filtered to your target geography. This can be done by identifying the PUMAs (Public Use Microdata Areas) that fall within your target geography, here the county of San Francisco. Luckily, a similar FIPS naming convention applies for PUMAs and counties, so filtering by PUMAs that contain the three digit FIPS county code, 075 for San Francisco, will parse the full state into the target county. For sub-county level target geographies such as census tracts, `Rsynthpops` ships with a dataframe `cts_to_pumas` that contains a lookup table relating PUMAs to state, county, and census tract FIPs codes.

```
SF_pums <- CA_pums %>%
  filter(grepl("075", PUMA))
```

Get data for target populations

Census data is reported in aggregate at the census block, census block group, census tract, county, and state levels. PUMS data is reported for individuals within PUMA areas that are somewhere between a census tract and a county in size and population. The target data to be fed into `ipu` thus provides a target for the synthetic population generator to replicate, while also maintaining correlations between variables found in the individual records of the seed datasets.

The first step for target data is to download aggregate census data with characteristics matching the seed data. This is perhaps the most tedious exercise of the entire process as we need to find the variables in the aggregate census dataset that match the variables we've chose in our seed datasets, then match the levels of each variable from the seed data to the aggregate data. We can use the `load_variables()` function from `tidycensus` to get a full list of available variables from the same survey we use to get the PUMS data. Most variables can likely be found in the default dataset called from `load_variables`, but some may be found in the `survey/profile` or `survey/subject` datasets (where `survey` should be replaced by the data survey being used, e.g. "acs5"). A number of lookup tables that relate acs variable names to puma variables to aid matching acs to puma categories are shipped with `Rsynthpops` and their derivation can be found in `Analysis/Generate_Lookups`

```
acs_vars<-load_variables(year_use, survey_use, cache=FALSE)
head(acs_vars)
```

```
## # A tibble: 6 x 3
##   name       label                                  concept
##   <chr>      <chr>                                  <chr>
## 1 B01001_001 Estimate!!Total:                       SEX BY AGE
## 2 B01001_002 Estimate!!Total:!!Male:                SEX BY AGE
## 3 B01001_003 Estimate!!Total:!!Male:!!Under 5 years SEX BY AGE
## 4 B01001_004 Estimate!!Total:!!Male:!!5 to 9 years  SEX BY AGE
## 5 B01001_005 Estimate!!Total:!!Male:!!10 to 14 years SEX BY AGE
## 6 B01001_006 Estimate!!Total:!!Male:!!15 to 17 years SEX BY AGE
```

```
acs_vars_subject<-load_variables(year_use, paste0(survey_use, "/subject"), cache=FALSE)
acs_vars_profile<-load_variables(year_use, paste0(survey_use, "/profile"), cache=FALSE)
```

```
# Age by sex -------------
acs_age_sex <- get_acs(
  geography = "tract",
  table     = "B01001",
  year      = year_use,
  state     = state_use,
  survey    = survey_use
) %>%
  mutate(county_fips = substr(GEOID,1,5))
```

```r
# Race ----------------------
acs_race <- get_acs(
  geography = "tract",
  table     = "B03002",
  year      = year_use,
  state     = state_use,
  survey    = survey_use
) %>%
  mutate(county_fips = substr(GEOID,1,5))

# School grade ----------------
acs_grade <- get_acs(
  geography = "tract",
  table     = "B14007",
  year      = year_use,
  state     = state_use,
  survey    = survey_use
) %>%
  mutate(county_fips = substr(GEOID,1,5))

# household size -----------------------
acs_hhsize_type<- get_acs(
  geography = "tract",
  variables = c(fam_2     ="B11016_003",
                fam_3     ="B11016_004",
                fam_4     ="B11016_005",
                fam_5     ="B11016_006",
                fam_6     ="B11016_007",
                fam_7plus ="B11016_008",
                non_1     ="B11016_010",
                non_2     ="B11016_011",
                non_3     ="B11016_012",
                non_4     ="B11016_013",
                non_5     ="B11016_014",
                non_6     ="B11016_015",
                non_7plus ="B11016_016"),
  state     = state_use,
  year      = year_use,
  survey    = survey_use
) %>%
  mutate(county_fips = substr(GEOID,1,5))

# household income ---------------
acs_hh_income <- get_acs(
  geography = "tract",
  table     = "B19001",
  year      = year_use,
  state     = state_use,
  survey    = survey_use
) %>%
  mutate(county_fips = substr(GEOID,1,5))

# Occupation --------------
```

```r
acs_occup <- get_acs(
  geography = "tract",
  table     = "S2401",
  year      = year_use,
  state     = state_use,
  survey    = survey_use
) %>%
  mutate(county_fips = substr(GEOID,1,5))
```

```r
save.image(file = here::here("Tutorial/data/downloaded_data.Rdata"))
```