Technical Reference

Addison Larson

2/8/2019

About

DVRPC's IPD analysis identifies populations of interest under Title VI of the Civil Rights Act and the Executive Order on Environmental Justice (#12898) using 2013-2017 American Community Survey (ACS) five-year estimates from the U.S. Census Bureau. The aim of IPD analysis is to assist both DVRPC and other outside organizations in their equity work by identifying populations of interest. Populations of interest include Youth, Older Adults, Female, Racial Minority, Ethnic Minority, Foreign-Born, Limited English Proficiency, Disabled, and Low-Income populations at the census tract level for the DVRPC region.

There are many ways of identifying these populations of interest. This document discusses DVRPC's process, which is automated in an `R` script. This document, and its corresponding script, have six major components:

1. Setup
2. Variance Replicate Table Download and Processing
3. ACS Estimates Download
4. Calculations
5. Cleaning
6. Summary Tables

In the end, the script generates a number of outputs, including:

1. `ipd.csv`, the primary output.
    - Estimated count of residents for each indicator (`CntEst`)
    - Margin of error (MOE) associated with the estimated count (`CntMOE`)
    - Estimated percentage of residents out of the total population for each indicator (`PctEst`)
    - MOE associated with the estimated percentage (`PctMOE`)
    - Rank (percentile) of the estimated percentage as compared to the study area, ranging from 0 to 1 (`Rank`)
    - IPD score of the estimated percentage, ranging from 0 to 4 (`Score`)
    - Text explanation of the IPD score (`Class`)
    - Total IPD score, created by summing up all component IPD scores (`IPD_Score`)
    - Universes and their MOEs used in analysis
2. `counts_by_indicator.csv`, a summary of the number of tracts that recieve each IPD score (0-4) for each indicator.
3. `breaks_by_indicator.csv`, a summary of the break values used for each indicator.
4. `export_summary.csv`, the minimum, median, mean, standard deviation, and maximum value of each indicator.
5. `mean_by_county.csv`, a summary of each indicator aggregated to the county level for DVRPC's nine-county region.

1. Setup

Dependencies

Packages required to run this script. If you don't have the packages, you'll get the warning `there is no package called <name of package>`, in which case you'll need to install it before proceeding.

```r
library(tidycensus); library(tidyverse); library(here); library(summarytools)
```

Functions

Load custom functions from `functions.R`. For more on the functions, see `functions_reference.Rmd`.

```r
source("functions.R")
```

2. Variance Replicate Table Download and Processing

This will feel out of order, but it's necessary — the racial minority indicator is created by summing up several subgroups in ACS Table B03002. This means that the MOE for the count has to be computed. While the ACS has issued guidance on computing the MOE by aggregating subgroups, in this instance, the large number of subgroups yields questionable MOEs. Variance replicate tables are used instead to compute a more accurate MOE. This single column is substituted in for the racial minority count MOE in Section 3.

Download, unzip, and read variance replicate tables for Table B03002.

```
nj_url <- "https://www2.census.gov/programs-surveys/acs/
replicate_estimates/2017/data/5-year/140/B02001_34.csv.gz"
pa_url <- "https://www2.census.gov/programs-surveys/acs/
replicate_estimates/2017/data/5-year/140/B02001_42.csv.gz"
nj_temp <- tempfile()
download.file(nj_url, nj_temp)
nj_var_rep <- read_csv(gzfile(nj_temp))
pa_temp <- tempfile()
download.file(pa_url, pa_temp)
pa_var_rep <- read_csv(gzfile(pa_temp))
```

Combine the two states' variance replicate tables into one and subset for DVRPC region. Only extract the necessary subgroups.

```
keep_cty <- c("34005", "34007", "34015", "34021",
              "42017", "42029", "42045", "42091", "42101")
var_rep <- bind_rows(nj_var_rep, pa_var_rep) %>%
  mutate_at(vars(GEOID), funs(str_sub(., 8, 18))) %>%
  filter(str_sub(GEOID, 1, 5) %in% keep_cty) %>%
  select(-TBLID, -NAME, -ORDER, -moe, -CME, -SE) %>%
  filter(TITLE %in% c("Black or African American alone",
                      "American Indian and Alaska Native alone",
                      "Asian alone",
                      "Native Hawaiian and Other Pacific Islander alone",
                      "Some other race alone",
                      "Two or more races:"))
```

Add up the racial minority counts into a single count per census tract for the estimate and 80 variance replicates. Separate the resulting tibble into the estimates and the variance replicates.

```
num <- var_rep %>%
  group_by(GEOID) %>%
  summarise_if(is.numeric, funs(sum)) %>%
  select(-GEOID)
estim <- num %>% select(estimate)
individual_replicate <- num %>% select(-estimate)
```

Compute the variance replicate for the count. GEOIDs are stored as `id` to be re-appended to the MOEs after they are calculated. Additional guidance on computing variance replicates is available at this (link).

```
id <- var_rep %>% select(GEOID) %>% distinct(.) %>% pull(.)
sqdiff_fun <- function(v, e) (v-e)^2
sqdiff <- mapply(sqdiff_fun, individual_replicate, estim)
sum_sqdiff <- rowSums(sqdiff)
variance <- 0.05 * sum_sqdiff
moe <- round(sqrt(variance) * 1.645, 0)
```

Save the racial minority MOE.

```r
export_moe <- cbind(id, moe) %>%
  as_tibble(.) %>%
  rename(GEOID = id, RM_CntMOE = moe) %>%
  write_csv(., here("outputs", "rm_moe.csv"))
```

3. ACS Estimates Download

Fields

The base information we need for IPD analysis are universes, counts, and percentages for nine indicators at the census tract level. For each indicator, the table below shows the indicator name, its abbreviation used in the script, its universe, its count, and its percentage field if applicable. Because the schemata of ACS tables can change with every update, these field names are applicable *only* to 2013-2017 ACS 5-Year Estimates.

Some percentage fields are empty. This is okay: we will compute the percentages when they are not directly available from the ACS.

Note that variable B02001_002 ("Estimate; Total: - White alone") is listed as the count for Racial Minority. This is a mathematical shortcut: otherwise, we would need to add several subfields to compute the same estimate. The desired count is B02001_001 (Universe) − B02001_002 ("Estimate; Total: - White alone"). The subtraction is computed after download, making a correct estimate and an incorrect MOE. The correct MOE for the count, as calculated in Section 2, will be appended later in Section 3.

Indicator	Abbreviation	Universe	Count	Percentage
Youth	Y	B03002_001	B09001_001	N/A
Older Adults	OA	S0101_C01_001	S0101_C01_030	S0101_C02_030
Female	F	S0101_C01_001	S0101_C05_001	DP05_0003PE
Racial Minority	RM	B02001_001	B02001_002	N/A
Ethnic Minority	EM	B03002_001	B03002_012	N/A
Foreign-Born	FB	B05012_001	B05012_003	N/A
Limited English Proficiency	LEP	S1601_C01_001	S1601_C05_001	S1601_C06_001
Disabled	D	S1810_C01_001	S1810_C02_001	S1810_C03_001
Low-Income	LI	S1701_C01_001	S1701_C01_042	N/A

While it's quicker to embed the names of the desired columns into the code, fields are explicitly spelled out in this script. This is a purposeful design choice. The user should check that the field names point to the correct API request with every IPD update. The best way to check the field names is to visit Census Developers (link) and select the corresponding API.

```
youth_universe                      <- "B03002_001"
youth_count                         <- "B09001_001"
youth_percent                       <- NULL
older_adults_universe               <- "S0101_C01_001"
older_adults_count                  <- "S0101_C01_030"
older_adults_percent                <- "S0101_C02_030"
female_universe                     <- "S0101_C01_001"
female_count                        <- "S0101_C05_001"
female_percent                      <- "DP05_0003PE"
racial_minority_universe            <- "B02001_001"
racial_minority_count               <- "B02001_002"
racial_minority_percent             <- NULL
ethnic_minority_universe            <- "B03002_001"
ethnic_minority_count               <- "B03002_012"
ethnic_minority_percent             <- NULL
foreign_born_universe               <- "B05012_001"
foreign_born_count                  <- "B05012_003"
foreign_born_percent                <- NULL
limited_english_proficiency_universe <- "S1601_C01_001"
limited_english_proficiency_count    <- "S1601_C05_001"
```

```r
limited_english_proficiency_percent  <- "S1601_C06_001"
disabled_universe                    <- "S1810_C01_001"
disabled_count                       <- "S1810_C02_001"
disabled_percent                     <- "S1810_C03_001"
low_income_universe                  <- "S1701_C01_001"
low_income_count                     <- "S1701_C01_042"
low_income_percent                   <- NULL
```

API Calls for counts (universes and counts)

Download counts and percentages for each of IPD's nine indicators. Note that the download is for all census tracts in New Jersey and Pennsylvania (`state = c(34,42)`).

Input data for IPD comes from ACS Subject Tables, Detailed Tables, and Data Profiles. While one can request all the fields for Subject Tables in one batch, mixing requests for two different types (e.g. Subject Tables and Detailed Tables) will result in failure. For this reason, counts are downloaded in two batches: `s_counts` for Subject Tables, and `d_counts` for Detailed Tables.

Note two exceptions embedded in processing:

1. The API does not allow redundant downloads, so universes for Older Adults and Youth are duplicated after download.
2. This API call downloads variable B02001_002 ("Estimate; Total: - White alone") as the count, when the desired variable for Racial Minority is B02001_001 − B02001_002. The subtraction is computed at the bottom of this code chunk.

```r
s_counts <- get_acs(geography = "tract", state = c(34,42), output = "wide",
                    variables = c(LI_U = low_income_universe,
                                  LI_C = low_income_count,
                                  F_U = female_universe,
                                  F_C = female_count,
                                  D_U = disabled_universe,
                                  D_C = disabled_count,
                                  # OA_U = older_adults_universe, # Redundant download
                                  OA_C = older_adults_count,
                                  LEP_U = limited_english_proficiency_universe,
                                  LEP_C = limited_english_proficiency_count)) %>%
  select(-NAME) %>%
  mutate(OA_UE = F_UE, OA_UM = F_UM)
d_counts <- get_acs(geography = "tract", state = c(34,42), output = "wide",
                    variables = c(EM_U = ethnic_minority_universe,
                                  EM_C = ethnic_minority_count,
                                  # Y_U = youth_universe, # Redundant download
                                  Y_C = youth_count,
                                  FB_U = foreign_born_universe,
                                  FB_C = foreign_born_count,
                                  RM_U = racial_minority_universe,
                                  RM_C = racial_minority_count)) %>%
  mutate(Y_UE = EM_UE, Y_UM = EM_UM, x = RM_UE - RM_CE) %>%
  select(-NAME, -RM_CE) %>%
  rename(RM_CE = x)
```

API Calls for percentages

Download percentage tables that are available for four of IPD's nine indicators. We will compute percentages and their associated MOEs for the rest of the dataset later.

The API request limitations are similar to those above. Percentages are downloaded in two batches: `s_percs` for Subject Tables, and `dp_percs` for Data Profiles.

```
s_percs <- get_acs(geography = "tract", state = c(34,42), output = "wide",
                   variables = c(D_P = disabled_percent,
                                 OA_P = older_adults_percent,
                                 LEP_P = limited_english_proficiency_percent)) %>%
  select(-NAME)
dp_percs <- get_acs(geography = "tract", state = c(34,42), output = "wide",
                    variables = c(F_P = female_percent)) %>%
  rename(F_PE = F_P, F_PM = DP05_0003PM) %>%
  select(-NAME)
```

Combine downloads into merged files

Merge downloads into two separate data frames: `dl_counts` for counts and universes, and `dl_percs` for available percentages. Subset data frames for DVRPC's nine-county region.

```
keep_cty <- c("34005", "34007", "34015", "34021",
              "42017", "42029", "42045", "42091", "42101")
dl_counts <- left_join(s_counts, d_counts) %>%
  filter(str_sub(GEOID, 1, 5) %in% keep_cty)
dl_percs <- left_join(s_percs, dp_percs) %>%
  filter(str_sub(GEOID, 1, 5) %in% keep_cty)
```

4. Calculations

For all nine indicators, Section 4 computes:

1. Percentages and their MOEs
2. Rank (Percentile)
3. IPD Score and Classification
4. Total IPD Score

Exception 1

Before computing percentages and percentage MOEs, import the count MOE for the Racial Minority variable computed from variance replicates. If `rm_moe.csv` exists, then this chunk will substitute the correct count MOE in `dl_counts`; if not, this chunk will do nothing.

```
if(TRUE %in% (list.files(here("outputs")) == "rm_moe.csv")){
  rm_moe <- read_csv(here("outputs", "rm_moe.csv")) %>%
    mutate_at(vars(GEOID), as.character)
  dl_counts <- dl_counts %>% left_join(., rm_moe) %>%
    select(-RM_CM) %>%
    rename(RM_CM = RM_CntMOE)
}
```

```
## Parsed with column specification:
## cols(
##   GEOID = col_double(),
##   RM_CntMOE = col_double()
## )
```

```
## Joining, by = "GEOID"
```

Split `dl_counts` into a list named `comp` for processing and sort column names for consistency. The name of the list, `comp`, is a nod to the "component parts" of `dl_counts`. The structure of `comp` is similar to a four-tab Excel spreadsheet: for example, `comp` is the name of the `.xlsx` file, `uni_est` is a tab for universe estimates, and `uni_est` has nine columns and 1,379 rows, where the column is the IPD indicator and the row is the census tract observation.

The order of columns is important because processing is based on vector position. We want to make sure that the first column of every tab corresponds to the Disabled indicator, the second to Ethnic Minority, et cetera.

```
comp <- list()
comp$uni_est <- dl_counts %>% select(ends_with("UE")) %>% select(sort(current_vars()))
comp$uni_moe <- dl_counts %>% select(ends_with("UM")) %>% select(sort(current_vars()))
comp$count_est <- dl_counts %>% select(ends_with("CE")) %>% select(sort(current_vars()))
comp$count_moe <- dl_counts %>% select(ends_with("CM")) %>% select(sort(current_vars()))
```

Percentages and their MOEs

Calculation

MOEs of the percentage values are obtained using the `tidycensus` function `moe_prop`. This chunk mentions `r` and `c` several times: continuing the spreadsheet analogy, think of `r` as the row number and `c` as the column number for a given spreadsheet tab.

```
pct_matrix <- NULL
pct_moe_matrix <- NULL
for (c in 1:length(comp$uni_est)){
  pct <- unlist(comp$count_est[,c] / comp$uni_est[,c]) * 100
  pct_matrix <- cbind(pct_matrix, pct)
  moe <- NULL
  for (r in 1:length(comp$uni_est$LI_UE)){
    moe_indiv <- as.numeric(moe_prop(comp$count_est[r,c],
                                     comp$uni_est[r,c],
                                     comp$count_moe[r,c],
                                     comp$uni_moe[r,c])) * 100
    moe <- append(moe, moe_indiv)
  }
  pct_moe_matrix <- cbind(pct_moe_matrix, moe)
}
```

Result

pct and pct_moe stores the percentages and associated MOEs for the nine indicator variables. Results are rounded to the tenths place.

```
pct <- as_tibble(pct_matrix) %>% mutate_all(round, 1)
names(pct) <- str_replace(names(comp$uni_est), "_UE", "_PctEst")
pct_moe <- as_tibble(pct_moe_matrix) %>% mutate_all(round, 1)
names(pct_moe) <- str_replace(names(comp$uni_est), "_UE", "_PctMOE")
```

Exception 2

If estimated percentage == 0 & MOE == 0, then make MOE = 0.1.

Matrix math: Only overwrite MOE where pct_matrix + pct_moe_matrix == 0.

```
overwrite_locations <- which(pct_matrix + pct_moe_matrix == 0, arr.ind = TRUE)
pct_moe[overwrite_locations] <- 0
```

Exception 3

Substitute percentages and associated MOEs when available from American FactFinder. This applies to the Older Adults, Female, Limited English Proficiency, and Disabled variables.

```
pct <- pct %>% mutate(D_PctEst = dl_percs$D_PE,
                      OA_PctEst = dl_percs$OA_PE,
                      LEP_PctEst = dl_percs$LEP_PE,
                      F_PctEst = dl_percs$F_PE)
pct_moe <- pct_moe %>% mutate(D_PctMOE = dl_percs$D_PM,
                              OA_PctMOE = dl_percs$OA_PM,
                              LEP_PctMOE = dl_percs$LEP_PM,
                              F_PctMOE = dl_percs$F_PM)
```

Rank (Percentile)

Calculation

Add percentages (an additional spreadsheet tab) to `comp`, making sure to first sort column names for consistency. Compute the empirical cumulative distribution function for each of the nine indicator variables.

```
comp$pct_est <- pct %>% select(sort(current_vars()))

percentile_matrix <- NULL
for (c in 1:length(comp$uni_est)){
  p <- unlist(comp$pct_est[,c])
  rank <- ecdf(p)(p)
  percentile_matrix <- cbind(percentile_matrix, rank)
}
```

Result

`percentile` stores the rank (percentile) for the nine indicator variables. Results are rounded to the hundredths place.

```
percentile <- as_tibble(percentile_matrix) %>% mutate_all(round, 2)
names(percentile) <- str_replace(names(comp$uni_est), "_UE", "_Rank")
```

Compute IPD score and classification

Each observation is assigned an IPD score for each indicator. The IPD score for an individual indicator can range from 0 to 4, which corresponds to the following:

IPD Score	IPD Classification	Standard Deviations
0	Well Below Average	x $< -1.5 stdev$
1	Below Average	$-1.5 stdev \leq$ x $< -0.5 stdev$
2	Average	$-0.5 stdev \leq$ x $< 0.5 stdev$
3	Above Average	$0.5 stdev \leq$ x $< 1.5 stdev$
4	Well Above Average	x $\geq 1.5 stdev$

Note an exception buried in processing:

1. If the estimated percentage for an observation == 0 and this observation also falls in Bin 1, move to Bin 0. Locations where this exception apply are stored in `overwrite_locations`.

```
score_matrix <- NULL
class_matrix <- NULL
for (c in 1:length(comp$uni_est)){
  p <- unlist(comp$pct_est[,c])
  breaks <- st_dev_breaks(p, 5, na.rm = TRUE)
  score <- (cut(p, labels = FALSE, breaks = breaks,
                include.lowest = TRUE, right = FALSE)) - 1
  overwrite_locations <- which(score == 1 & p == 0)
  score[overwrite_locations] <- 0
  class <- case_when(score == 0 ~ "Well Below Average",
                     score == 1 ~ "Below Average",
                     score == 2 ~ "Average",
                     score == 3 ~ "Above Average",
                     score == 4 ~ "Well Above Average")
  score_matrix <- cbind(score_matrix, score)
  class_matrix <- cbind(class_matrix, class)
}
```

Result

`score` and `class` store the IPD scores and associated descriptions for the nine indicator variables.

```
score <- as_tibble(score_matrix)
names(score) <- str_replace(names(comp$uni_est), "_UE", "_Score")
class <- as_tibble(class_matrix)
names(class) <- str_replace(names(comp$uni_est), "_UE", "_Class")
```

Compute total IPD score across all nine indicators

Sum the IPD scores for the nine indicator variables to determine the overall IPD score.

```
score <- score %>% mutate(IPD_Score = rowSums(.))
```

5. Cleaning

There is a specific output format for `ipd.csv`, including column names, column order, flags for missing data, and census tracts with insufficient data. This section ensures conformity with the output formatting.

Merge the percentage estimates, their MOEs, rank (percentile), score, and class tibbles into a single data frame called `df`.

```r
df <- bind_cols(dl_counts, pct) %>%
  bind_cols(., pct_moe) %>%
  bind_cols(., percentile) %>%
  bind_cols(., score) %>%
  bind_cols(., class)
```

Rename columns.

```r
names(df) <- str_replace(names(df), "_CE", "_CntEst")
names(df) <- str_replace(names(df), "_CM", "_CntMOE")
df <- df %>% mutate(U_TPopEst = F_UE, U_TPopMOE = F_UM, U_Pop6Est = LEP_UE,
                    U_Pop6MOE = LEP_UM, U_PPovEst = LI_UE, U_PPovMOE = LI_UM,
                    U_PNICEst = D_UE, U_PNICMOE = D_UM) %>%
  select(-ends_with("UE"), -ends_with("UM"))
```

Reorder columns, with `GEOID` first, the following variables in alphabetical order, and the total IPD score and universes at the end.

```r
df <- df %>% select(GEOID, sort(current_vars())) %>%
  select(move_last(., c("IPD_Score", "U_TPopEst", "U_TPopMOE", "U_Pop6Est",
                        "U_Pop6MOE", "U_PPovEst", "U_PPovMOE", "U_PNICEst", "U_PNICMOE")))
```

Replace `NA` values with `NoData` if character and `-99999` if numeric.

```r
df <- df %>% mutate_if(is.character, funs(ifelse(is.na(.), "NoData", .))) %>%
  mutate_if(is.numeric, funs(ifelse(is.na(.), -99999, .)))
```

Slice unwanted tracts because of low population.

```r
slicer <- c("42045980000", "42017980000", "42101980800",
            "42101980300", "42101980500", "42101980400",
            "42101980900", "42101980700", "42101980600",
            "42101005000")
df <- df %>% filter(!(GEOID %in% slicer))
```

Export result.

```r
write_csv(df, here("outputs", "ipd.csv"))
```

6. Summary Tables

(Forthcoming)