The mechanics of `1_download.R`

Addison Larson

1/25/2019

Setup

Dependencies

Packages required to run this script.

```r
require(tidycensus); require(tidyverse); require(here)
```

Functions

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

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

Fields

The base information we need for IPD analysis are universes, counts, and percentages for nine indicators at the census tract level. The table below shows the name of each indicator; its abbreviation used in scripts; and its universe, count, and percentage field, if available. 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. Technically, the desired count is B02001_001 − B02001_002, aka the universe minus the downloaded count. This is computed after download, making a correct estimate and an incorrect MOE.

Indicator	Abbreviation	Universe	Count	Percentage
Youth	Y	B03002_001	B09001_001	N/A
Older Adults	OA	S0101_C01_001	DP05_0029E	S0101_C01_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 at the top of this script. This is by design: because of the annual changes in ACS data schemata, 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                   <- "DP05_0029E"
older_adults_percent                 <- "S0101_C01_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"
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
```

Downloads

API Calls for counts (universes and counts)

Download counts and percentages for each of IPD's nine indicators.

Input data for IPD comes from several sources, including 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 three batches: `s_counts` for Subject Tables, `d_counts` for Detailed Tables, and `dp_counts` for Data Profiles.

Note two exceptions embedded in processing:

1. The API does not allow redundant downloads, so universes for Youth and Limited English Proficiency 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 technically B02001_001 − B02001_002, aka the universe minus the estimate. This is computed after download and at the bottom of this code chunk: the result is a correct estimate and an incorrect MOE.

```
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
                                  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)
dp_counts <- get_acs(geography = "tract", state = c(34,42), output = "wide",
                     variables = c(OA_CE = older_adults_count)) %>%
  rename(OA_CM = DP05_0029M) %>% select(-NAME)
```

API Calls for percentages (universes and estimates)

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_perce` 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) %>%
  left_join(., dp_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)
```

Calculations

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 will be 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. For all nine indicators:

1. Compute percentages and associated MOEs
2. Compute percentile
3. Compute IPD score and classification
4. Compute total IPD score across all nine indicators

```
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()))
```

Compute percentages and associated 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")
```

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

Exception 1

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

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

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

Exception 2

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

```r
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)
```

Exception 3

To be added. Use variance replicates to compute `RM_CntMOE` and `RM_PctMOE`.

Compute 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.

```r
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 percentile rank 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	$s \leq -1.5$
1	Below Average	$-1.5 < s \leq -0.5$
2	Average	$-0.5 < s \leq 0.5$
3	Above Average	$0.5 < s \leq 1.5$
4	Well Above Average	$s > 1.5$

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 = TRUE)) - 1
  overwrite_locations <- which(score == 0 & 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(.))
```

Cleaning

Merge all information into a single data frame called `df`.

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

Rename columns.

```
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.

```
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.

```
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.

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

Export result.

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