The mechanics of `1_download.R`

Addison Larson

1/25/2019

Setup

Dependencies

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

Functions

1. For a given vector of numbers `x` and a number of bins `i`, `st_dev_breaks` computes the bin breaks starting at $-0.5 \cdot stdev$ and $0.5 \cdot stdev$. For the purposes of IPD analysis, `i = 5`, and `st_dev_breaks` calculates the minimum, $-1.5 \cdot stdev$, $-0.5 \cdot stdev$, $0.5 \cdot stdev$, $1.5 \cdot stdev$, and maximum values. These values are later used to slice the vector into five bins.
2. `move_last` moves a column or vector of columns to the last position in a tibble or data frame.

```r
st_dev_breaks <- function(x, i, na.rm = TRUE){
  half_st_dev_count <- c(-1 * rev(seq(1, i, by = 2)),
                         seq(1, i, by = 2))
  if((i %% 2) == 1) {
    half_st_dev_breaks <- sapply(half_st_dev_count,
                                 function(i)(0.5 * i * sd(x)) + mean(x))
    half_st_dev_breaks[[1]] <- ifelse(min(x) < half_st_dev_breaks[[1]],
                                      min(x), half_st_dev_breaks[[1]])
    half_st_dev_breaks[[i + 1]] <- ifelse(max(x) > half_st_dev_breaks[[i + 1]],
                                          max(x), half_st_dev_breaks[[i + 1]])
  } else {
    half_st_dev_breaks <- NA
  }
  return(half_st_dev_breaks)
}
move_last <- function(df, last_col) {
  match(c(setdiff(names(df), last_col), last_col), names(df))
}
```

Downloads

API Calls for counts (universes and estimates)

Download all necessary tables for count data for IPD's nine indicators. Below is a brief overview of the indicators, abbreviations used in data processing, and ACS 5-Year Estimate source tables. *Warning*: The ACS sometimes changes its data schemata for tables. When working with updated ACS estimates, revisit each field name in the Census API and make sure you are downloading the fields you want.

Indicator	Abbreviation	Table	Universe	Estimate
Youth	Y	B09001	B03002_001	B09001_001
Older Adults	OA	DP05	S0101_C01_001	DP05_0029E
Female	F	S0101	S0101_C01_001	S0101_C05_001
Racial Minority	RM	B02001	B02001_001	B02001_002
Ethnic Minority	EM	B03002	B03002_001	B03002_012
Foreign-Born	FB	B05012	B05012_001	B05012_003
Limited English Proficiency	LEP	S1601	S1601_C01_001	S1601_C05_001
Disabled	D	S1810	S1810_C01_001	S1810_C02_001
Low-Income	LI	S1710	S1701_C01_001	S1701_C01_042

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, making a correct estimate and an incorrect MOE.

```
s_counts <- get_acs(geography = "tract", state = c(34,42), output = "wide",
                    variables = c(LI_U = "S1701_C01_001",
                                  LI_C = "S1701_C01_042",
                                  F_U = "S0101_C01_001",
                                  F_C = "S0101_C05_001",
                                  D_U = "S1810_C01_001",
                                  D_C = "S1810_C02_001",
                                  # OA_U = "S0101_C01_001", # Redundant download
                                  LEP_U = "S1601_C01_001",
                                  LEP_C = "S1601_C05_001")) %>%
  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 = "B03002_001",
                                  EM_C = "B03002_012",
                                  # Y_U = "B03002_001", # Redundant download
                                  Y_C = "B09001_001",
                                  FB_U = "B05012_001",
                                  FB_C = "B05012_003",
                                  RM_U = "B02001_001",
                                  RM_C = "B02001_002")) %>%
  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 = "DP05_0029E")) %>%
  rename(OA_CM = DP05_0029M) %>% select(-NAME)
```

API Calls for percentages (universes and estimates)

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

Indicator	Abbreviation	Table	Percentage
Older Adults	OA	S0101	S0101_C01_030
Female	F	DP05	DP05_0003PE
Limited English Proficiency	LEP	S1601	S1601_C06_001
Disabled	D	S1810	S1810_C03_001

```
s_percs <- get_acs(geography = "tract", state = c(34,42), output = "wide",
                   variables = c(D_P = "S1810_C03_001",
                                 OA_P = "S0101_C01_030",
                                 LEP_P = "S1601_C06_001")) %>% select(-NAME)
dp_percs <- get_acs(geography = "tract", state = c(34,42), output = "wide",
                    variables = c(F_P = "DP05_0003PE")) %>%
  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 order of columns will be important because processing is based on vector position. The name of the list, `comp`, is a nod to the "component parts" of `dl_counts`. 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[[1]] <- dl_counts %>% select(ends_with("UE")) %>% select(sort(current_vars()))
comp[[2]] <- dl_counts %>% select(ends_with("UM")) %>% select(sort(current_vars()))
comp[[3]] <- dl_counts %>% select(ends_with("CE")) %>% select(sort(current_vars()))
comp[[4]] <- 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`.

```
pct_matrix <- NULL
pct_moe_matrix <- NULL
for (m in 1:length(comp[[1]])){
  pct <- unlist(comp[[3]][,m] / comp[[1]][,m]) * 100
  pct_matrix <- cbind(pct_matrix, pct)
  moe <- NULL
  for (l in 1:length(comp[[1]]$LI_UE)){
    moe_indiv <- as.numeric(moe_prop(comp[[3]][l,m],
                                     comp[[1]][l,m],
                                     comp[[4]][l,m],
                                     comp[[2]][l,m])) * 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.

```
pct <- as_tibble(pct_matrix) %>% mutate_all(round, 1)
names(pct) <- str_replace(names(comp[[1]]), "_UE", "_PctEst")
pct_moe <- as_tibble(pct_moe_matrix) %>% mutate_all(round, 1)
names(pct_moe) <- str_replace(names(comp[[1]]), "_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.

```
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.

```
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 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[[5]] <- pct %>% select(sort(current_vars()))
percentile_matrix <- NULL
for (m in 1:length(comp[[1]])){
  p <- unlist(comp[[5]][,m])
  rank <- ecdf(p)(p)
  percentile_matrix <- cbind(percentile_matrix, rank)
}
```

Result

`percentile` stores the percentile rank for the nine indicator variables.

```
percentile <- as_tibble(percentile_matrix) %>% mutate_all(round, 2)
names(percentile) <- str_replace(names(comp[[1]]), "_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 (m in 1:length(comp[[1]])){
  p <- unlist(comp[[5]][,m])
  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[[1]]), "_UE", "_Score")
class <- as_tibble(class_matrix)
names(class) <- str_replace(names(comp[[1]]), "_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"))
```