Technical Reference

Outline

1. About
    a. Getting started
    b. Output abbreviations
    c. Project structure
2. Setup
    a. Dependencies
    b. Fields
    c. Year
    d. States
    e. Counties
    f. Census API key
    g. Functions
        1. Override `base` and `stats` function defaults
        2. Create custom half-standard deviation breaks
        3. *Exception*
        4. Move column or vector of columns to last position
        5. Summarize data
3. Variance replicate table download
    a. Download variance replicates from Census website
    b. Combine and format downloads
4. Variance replicate table processing
    a. Compute racial minority count MOE
    b. Save results
5. ACS estimates download
    a. Fields
    b. Download counts and universes from Census API
        1. *Exception*
    c. Download percentages from Census API
    d. Format downloads
        1. *Exception*
        2. *Exception*
        3. *Exception*
6. ACS estimates calculations
    a. Percentages and percentage MOEs
        1. Calculation
        2. Result
        3. *Exception*
        4. *Exception*
    b. Percentile
        1. Calculation
        2. Result
    c. IPD score and classification
        1. Calculation
        2. Result
    d. Composite IPD score
        1. Calculation
        2. Result

7. ACS estimates cleaning
8. Summary tables
    a. Counts by indicator
    b. Breaks by indicator
    c. Summary by indicator
    d. County means by indicator
9. Export
    a. Append to TIGER/LINE file
    b. Export files

1. 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. IPD analysis assists both DVRPC and outside organizations in equity work by identifying populations of interest, including Youth, Older Adults, Female, Racial Minority, Ethnic Minority, Foreign-Born, Limited English Proficiency, Disabled, and Low-Income populations at the census tract level in DVRPC's nine-county region.

There are many ways of identifying these populations of interest. This document discusses DVRPC's process, which is automated in an `R` script.

1a. Getting started

For guidance on software prerequisites and how to run this script, see `getting_started.pdf`.

1b. Output abbreviations

Components of field names that you'll see in `outputs`.

Component	Equivalent
D	Disabled
EM	Ethnic Minority
F	Female
FB	Foreign-Born
LEP	Limited English Proficiency
LI	Low-Income
OA	Older Adults
RM	Racial Minority
Y	Youth
CntEst	Count Estimate
CntMOE	Count MOE
PctEst	Percentage Estimate
PctMOE	Percentage MOE
Pctile	Percentile
Score	Score
Class	Classification

Abbreviations of field names that you'll see in `outputs` *not* comprised of the above components.

Abbreviation	Equivalent
GEOID	Census Tract Identifier
STATEFP	State FIPS Code
COUNTYFP	County FIPS Code
NAME	Census Tract FIPS Code
IPD_Score	Composite Score
U_TPopEst	Total Population Estimate
U_TPopMOE	Total Population MOE
U_Pop6Est	Population 6+ Estimate
U_Pop6MOE	Population 6+ MOE
U_PPovEst	Poverty Status Population Estimate

Abbreviation	Equivalent
U_PPovMOE	Poverty Status Population MOE
U_PNICEst	Non-Institutional Civilian Population Estimate
U_PNICMOE	Non-Institutional Civilian Population MOE

1c. Project structure

This script uses relative file paths based off the location of `ipd_2017.Rproj`. As long as you download the entire repository, the script should have no trouble locating the correct subfolders. The project is structured as follows:

```
ipd_2017
|--ipd_2017.Rproj
|-- script.R
|-- documentation
|   |-- discussion.pdf
|   |-- getting_started.pdf
|   |-- script_reference.pdf
|   |-- script_reference.Rmd
|   |-- variables.csv
|--| outputs
|   |-- breaks_by_indicator.csv
|   |-- counts_by_indicator.csv
|   |-- ipd.csv
|   |-- ipd.dbf
|   |-- ipd.prj
|   |-- ipd.shp
|   |-- ipd.shx
|   |-- mean_by_county.csv
|   |-- summary_by_indicator.csv
```

2. Setup

2a. Dependencies

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

```
library(plyr); library(here); library(sf); library(summarytools);
library(tidycensus); library(tidyverse); library(tigris)
```

2b. 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 4, will be appended later.

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

```
disabled_universe                     <- "S1810_C01_001"
disabled_count                        <- "S1810_C02_001"
disabled_percent                      <- "S1810_C03_001"
ethnic_minority_universe              <- "B03002_001"
ethnic_minority_count                 <- "B03002_012"
ethnic_minority_percent               <- NA
female_universe                       <- "S0101_C01_001"
female_count                          <- "S0101_C05_001"
female_percent                        <- "DP05_0003PE"
foreign_born_universe                 <- "B05012_001"
foreign_born_count                    <- "B05012_003"
foreign_born_percent                  <- NA
limited_english_proficiency_universe <- "S1601_C01_001"
limited_english_proficiency_count     <- "S1601_C05_001"
limited_english_proficiency_percent   <- "S1601_C06_001"
low_income_universe                   <- "S1701_C01_001"
low_income_count                      <- "S1701_C01_042"
low_income_percent                    <- NA
older_adults_universe                 <- "S0101_C01_001"
older_adults_count                    <- "S0101_C01_030"
older_adults_percent                  <- "S0101_C02_030"
racial_minority_universe              <- "B02001_001"
racial_minority_count                 <- "B02001_002"
racial_minority_percent               <- NA
youth_universe                        <- "B03002_001"
youth_count                           <- "B09001_001"
youth_percent                         <- NA
```

2c. Year

The data download year.

```
ipd_year <- 2017
```

2d. States

The data download state or states. Be sure to use the two-character text abbreviation.

```
ipd_states <- c("NJ", "PA")
```

2e. Counties

The counties in your study area. Note that these are characters of length 5 concatenating the 2-digit state FIPS and the 3-digit county FIPS.

```
ipd_counties <- c("34005", "34007", "34015", "34021",
                  "42017", "42029", "42045", "42091", "42101")
```

2f. Census API Key

Placeholder if you have never installed an API key before. If this is you, check out `getting_started.pdf` for where to get a Census API key and how to install it.

```
# Census API Key
# census_api_key("YOUR API KEY GOES HERE", install = TRUE)
```

2g. Functions

Load custom functions.

2g.i. Override `base` and `stats` function defaults

A time-saver so that it's not required to call `na.rm = TRUE` every time these functions are called.

```
min <- function(i, ..., na.rm = TRUE) {
  base::min(i, ..., na.rm = na.rm)
}
mean <- function(i, ..., na.rm = TRUE) {
  base::mean(i, ..., na.rm = na.rm)
}
sd <- function(i, ..., na.rm = TRUE) {
  stats::sd(i, ..., na.rm = na.rm)
}
max <- function(i, ..., na.rm = TRUE) {
  base::max(i, ..., na.rm = na.rm)
}
```

2g.ii. Create custom half-standard deviation breaks

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.

2g.iii. *Exception*

All minima are coerced to equal zero. If the first bin break is negative, as happens when the data has a large spread and therefore a large standard deviation, then this bin break is coerced to equal 0.001. In these cases, only estimates of 0 percent will be placed in the bottom bin.

```
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]] <- 0
    half_st_dev_breaks[[2]] <- ifelse(half_st_dev_breaks[[2]] < 0,
                                      0.001,
                                      half_st_dev_breaks[[2]])
    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)
}
```

2g.iv. Move column or vector of columns to last position

The requested schema for IPD data export renames and places all relevant universes in the final columns of the dataset. `move_last` moves a column or vector of columns to the last position in a tibble or data frame.

```
move_last <- function(df, last_col) {
  match(c(setdiff(names(df), last_col), last_col), names(df))
}
```

2g.v. Summarize data

`description` tailors the exports from `summarytools::descr` to create summary tables with the requested fields. $\frac{stdev}{2}$ is returned after *stdev*.

```
description <- function(i) {
  des <- as.numeric(descr(i, na.rm = TRUE,
                          stats = c("min", "med", "mean", "sd", "max")))
  des <- c(des[1:4], des[4] / 2, des[5])
  return(des)
}
```

3. Variance replicate table download

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, using the approximation formula can artificially deflate the derived MOE. 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 5. Additional guidance on computing variance replicates is available at this (link).

3a. Download variance replicates from Census website

Download, unzip, and read variance replicate tables for Table B02001. Results are combined into a single table called `var_rep`.

```
ipd_states_numeric <- fips_codes %>%
  filter(state %in% ipd_states) %>%
  select(state_code) %>% distinct(.) %>% pull(.)
var_rep <- NULL
for (i in 1:length(ipd_states)){
  url <- paste0("https://www2.census.gov/programs-surveys/acs/replicate_estimates/",
                ipd_year,
                "/data/5-year/140/B02001_",
                ipd_states_numeric[i],
                ".csv.gz")
  temp <- tempfile()
  download.file(url, temp)
  var_rep_i <- read_csv(gzfile(temp))
  var_rep <- rbind(var_rep, var_rep_i)
}
```

3b. Combine and format downloads

Subset `var_rep` for the study area defined in `ipd_counties` and extract the necessary subgroups.

```
var_rep <- var_rep %>%
  mutate_at(vars(GEOID), funs(str_sub(., 8, 18))) %>%
  filter(str_sub(GEOID, 1, 5) %in% ipd_counties) %>%
  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:"))
```

4. Variance replicate table processing

4a. Compute racial minority count MOE

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) %>%
  summarize_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.

```
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)
```

4b. Save results

Save the racial minority MOE.

```
rm_moe <- cbind(id, moe) %>%
  as_tibble(.) %>%
  rename(GEOID = id, RM_CntMOE = moe)
```

Here are the first few lines of `rm_moe`:

```
head(rm_moe)
```

```
## # A tibble: 6 x 2
##   GEOID       RM_CntMOE
##   <chr>       <chr>
## 1 34005700102 204
## 2 34005700103 246
## 3 34005700104 224
## 4 34005700200 105
## 5 34005700303 316
## 6 34005700304 115
```

5. ACS estimates download

5a. Fields

Fields for downloads from the ACS API were discussed in Section 2b.

5b. Download counts and universes from Census API

Download counts and percentages for each of IPD's nine indicators. Note that the download is for all census tracts in `ipd_states`.

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 or more different types of tables will result in failure. For this reason, the counts and universe fields supplied by the user in Section 2b are evaluated for their contents and split into three batches: `s_counts` for Subject Tables, `d_counts` for Detailed Tables, and `dp_counts` for Data Profiles.

The chunk below zips the user-defined calls from the API with the preferred script abbreviations into a tibble called `counts_calls` and separates the calls into three batches.

```
counts <- c(disabled_count, disabled_universe,
            ethnic_minority_count, ethnic_minority_universe,
            female_count, female_universe,
            foreign_born_count, foreign_born_universe,
            limited_english_proficiency_count, limited_english_proficiency_universe,
            low_income_count, low_income_universe,
            older_adults_count, older_adults_universe,
            racial_minority_count, racial_minority_universe,
            youth_count, youth_universe)
counts_ids <- c("D_C", "D_U", "EM_C", "EM_U", "F_C", "F_U",
                "FB_C", "FB_U", "LEP_C", "LEP_U", "LI_C", "LI_U",
                "OA_C", "OA_U", "RM_C", "RM_U", "Y_C", "Y_U")
counts_calls <- tibble(id = counts_ids, api = counts) %>%
  drop_na(.)
s_calls <- counts_calls %>%
  filter(str_sub(api, 1, 1) == "S")
d_calls <- counts_calls %>%
  filter(str_sub(api, 1, 1) == "B")
dp_calls <- counts_calls %>%
  filter(str_sub(api, 1, 1) == "D")
```

API calls are made separately for ACS Subject Tables, Detailed Tables, and Data Profiles and appended to `dl_counts`. Sometimes there are no requests for a certain type of table; in these situations, the script bypasses a download attempt. Then, information from `counts_calls` is used to rename the downloads to the appropriate abbreviation.

```
dl_counts <- NULL
if(length(s_calls$id > 0)){
  s_counts <- get_acs(geography = "tract",
                      state = ipd_states,
                      output = "wide",
                      year = ipd_year,
                      variables = s_calls$api) %>%
    select(-NAME)
  dl_counts <- bind_cols(dl_counts, s_counts)
}
if(length(d_calls$id > 0)){
  d_counts <- get_acs(geography = "tract",
                      state = ipd_states,
                      output = "wide",
                      year = ipd_year,
                      variables = d_calls$api) %>%
    select(-NAME)
  dl_counts <- left_join(dl_counts, d_counts)
}
if(length(dp_calls$id > 0)){
  dp_counts <- get_acs(geography = "tract",
                       state = ipd_states,
                       output = "wide",
```

```
                       year = ipd_year,
                       variables = dp_calls$api) %>%
    select(-NAME)
  dl_counts <- left_join(dl_counts, dp_counts)
}
counts_calls$api <- str_replace(counts_calls$api, "E$", "")
for(i in 1:length(counts_calls$id)){
  names(dl_counts) <- str_replace(names(dl_counts),
                                  counts_calls$api[i],
                                  counts_calls$id[i])
}
```

5b.i. *Exception*

The API does not allow redundant downloads, so universes for Older Adults and Youth are duplicated after download. `duplicate_cols` identifies duplicate API calls, and `combined_rows` serves as a crosswalk to duplicate and rename fields.

```
duplicate_cols <- counts_calls %>%
  group_by(api) %>%
  filter(n()>1) %>%
  summarize(orig = id[1],
            duplicator = id[2])
e_paste <- function(i) paste0(i, "E")
m_paste <- function(i) paste0(i, "M")
e_rows <- apply(duplicate_cols, 2, e_paste)
m_rows <- apply(duplicate_cols, 2, m_paste)
combined_rows <- as_tibble(rbind(e_rows, m_rows)) %>%
  mutate_all(as.character)
for(i in 1:length(combined_rows$api)){
  dl_counts[combined_rows$duplicator[i]] <- dl_counts[combined_rows$orig[i]]
}
```

5c. Download percentages from Census API

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 procedure is identical to that described in Section 5b.

```
percs <- c(disabled_percent,
           ethnic_minority_percent,
           female_percent,
           foreign_born_percent,
           limited_english_proficiency_percent,
           low_income_percent,
           older_adults_percent,
           racial_minority_percent,
           youth_percent)
percs_ids <- c("D_P", "EM_P", "F_P", "FB_P", "LEP_P",
               "LI_P", "OA_P", "RM_P", "Y_P")
percs_calls <- tibble(id = percs_ids, api = percs) %>%
  drop_na(.)
s_calls <- percs_calls %>%
```

```
  filter(str_sub(api, 1, 1) == "S")
d_calls <- percs_calls %>%
  filter(str_sub(api, 1, 1) == "B")
dp_calls <- percs_calls %>%
  filter(str_sub(api, 1, 1) == "D")
dl_percs <- NULL
if(length(s_calls$id > 0)){
  s_percs <- get_acs(geography = "tract",
                     state = ipd_states,
                     output = "wide",
                     year = ipd_year,
                     variables = s_calls$api) %>%
    select(-NAME)
  dl_percs <- bind_cols(dl_percs, s_percs)
}
if(length(d_calls$id > 0)){
  d_percs <- get_acs(geography = "tract",
                     state = ipd_states,
                     output = "wide",
                     year = ipd_year,
                     variables = d_calls$api) %>%
    select(-NAME)
  dl_percs <- left_join(dl_percs, d_percs)
}
if(length(dp_calls$id > 0)){
  dp_percs <- get_acs(geography = "tract",
                      state = ipd_states,
                      output = "wide",
                      year = ipd_year,
                      variables = dp_calls$api) %>%
    select(-NAME)
  dl_percs <- left_join(dl_percs, dp_percs)
}
percs_calls$api <- str_replace(percs_calls$api, "PE", "")
names(dl_percs) <- str_replace(names(dl_percs), "PE", "E")
names(dl_percs) <- str_replace(names(dl_percs), "PM", "M")
for(i in 1:length(percs_calls$id)){
  names(dl_percs) <- str_replace(names(dl_percs),
                                 percs_calls$api[i],
                                 percs_calls$id[i])
}
```

5d. Format downloads

Subset `dl_counts` and `dl_percs` for DVRPC's nine-county region. Percentages should be in the range 0 to 1, so all values in `dl_percs` are divided by 100.

```
dl_counts <- dl_counts %>%
  filter(str_sub(GEOID, 1, 5) %in% ipd_counties)
dl_percs <- dl_percs %>%
  filter(str_sub(GEOID, 1, 5) %in% ipd_counties) %>%
  mutate_if(is.numeric, funs(. / 100))
```

5d.i. *Exception*

Note that variable B02001_002 ("Estimate; Total: - White alone") was downloaded as the count for Racial Minority. Compute B02001_001 (Universe) − B02001_002 ("Estimate; Total: - White alone") and substitute for RM_CE.

```
dl_counts <- dl_counts %>% mutate(x = RM_UE - RM_CE) %>%
  select(-RM_CE) %>%
  rename(RM_CE = x)
```

5d.ii. *Exception*

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

```
if(exists("rm_moe")){
  dl_counts <- dl_counts %>%
    select(-RM_CM) %>%
    left_join(., rm_moe) %>%
    rename(RM_CM = RM_CntMOE) %>%
    mutate_at(vars(RM_CM), as.numeric)
}
```

5d.iii. *Exception*

Half-standard deviations serve as the classification bins for IPD scores, and including zero-population tracts affects computed standard deviation values. Start by removing the 10 census tracts with zero population.

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

Here are the first few lines of dl_counts and dl_percs. Notice the naming convention:

- UE = universe estimate
- UM = universe MOE
- CE = count estimate
- CM = count MOE
- PE = percentage estimate
- PM = percentage MOE

We use these strings to select columns, so consistency is key.

```
head(dl_counts)
```

```
## # A tibble: 6 x 37
##   GEOID D_CE  D_CM  D_UE  D_UM  F_CE  F_CM  F_UE  F_UM LEP_CE LEP_CM
##   <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl>  <dbl>
## 1 3402~   283   115  2144   354   932   170  2164   354    900    266
## 2 3402~    79    47  1194   271   638   158  1194   271    197     92
## 3 3402~   147    80  1603   280   807   148  1603   280    439    228
## 4 3402~   449   133  4513   495  2327   344  4513   495    948    245
```

```
## 5 3402~   249    57  2591    19  1378    93  2591    19     78     58
## 6 3402~   328   125  5584   469  2787   304  5584   469    329    175
## # ... with 26 more variables: LEP_UE <dbl>, LEP_UM <dbl>, LI_CE <dbl>,
## #   LI_CM <dbl>, LI_UE <dbl>, LI_UM <dbl>, OA_CE <dbl>, OA_CM <dbl>,
## #   EM_CE <dbl>, EM_CM <dbl>, EM_UE <dbl>, EM_UM <dbl>, FB_CE <dbl>,
## #   FB_CM <dbl>, FB_UE <dbl>, FB_UM <dbl>, RM_UE <dbl>, RM_UM <dbl>,
## #   Y_CE <dbl>, Y_CM <dbl>, Y_UE <dbl>, OA_UE <dbl>, Y_UM <dbl>,
## #   OA_UM <dbl>, RM_CE <dbl>, RM_CM <dbl>
```

```r
head(dl_percs)
```

```
## # A tibble: 6 x 9
##   GEOID        D_PE   D_PM LEP_PE LEP_PM OA_PE OA_PM  F_PE  F_PM
##   <chr>       <dbl>  <dbl>  <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 34021000800 0.132 0.05    0.458  0.102 0.121 0.047 0.431 0.061
## 2 34021001600 0.066 0.039   0.179  0.083 0.09  0.047 0.534 0.057
## 3 34021001900 0.092 0.049   0.297  0.125 0.064 0.038 0.503 0.075
## 4 34021002601 0.099 0.0280  0.235  0.047 0.092 0.02  0.516 0.047
## 5 34021003903 0.096 0.022   0.031  0.023 0.23  0.024 0.532 0.035
## 6 34021004204 0.059 0.023   0.06   0.032 0.18  0.031 0.499 0.033
```

6. ACS estimates calculations

For all nine indicators, this section computes:

a. Percentages and percentage MOEs
b. Percentile
c. IPD score and classification
d. Composite IPD score

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 1,369 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. `select(sort(current_vars()))` organizes columns alphabetically.

```r
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()))
```

6a. Percentages and percentage MOEs

6a.i. 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$uni_est[,c])
  pct_matrix <- cbind(pct_matrix, pct)
  moe <- NULL
  for (r in 1:length(comp$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]))
    moe <- append(moe, moe_indiv)
  }
  pct_moe_matrix <- cbind(pct_moe_matrix, moe)
}
```

6a.ii. Result

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

```
pct <- as_tibble(pct_matrix) %>% mutate_all(round, 3)
names(pct) <- str_replace(names(comp$uni_est), "_UE", "_PctEst")
pct_moe <- as_tibble(pct_moe_matrix) %>% mutate_all(round, 3)
names(pct_moe) <- str_replace(names(comp$uni_est), "_UE", "_PctMOE")
```

6a.iii. ***Exception***

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] <- 0.1
```

6a.iv. ***Exception***

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)
```

Here are the first few lines of `pct` and `pct_moe`:

```
head(pct)
```

```
## # A tibble: 6 x 9
##   D_PctEst EM_PctEst F_PctEst FB_PctEst LEP_PctEst LI_PctEst OA_PctEst
##      <dbl>     <dbl>    <dbl>     <dbl>      <dbl>     <dbl>     <dbl>
## 1    0.132     0.77     0.431     0.425      0.458     0.697     0.121
## 2    0.066     0.262    0.534     0.16       0.179     0.439     0.09
## 3    0.092     0.457    0.503     0.273      0.297     0.829     0.064
## 4    0.099     0.411    0.516     0.309      0.235     0.427     0.092
## 5    0.096     0.036    0.532     0.094      0.031     0.095     0.23
## 6    0.059     0.08     0.499     0.28       0.06      0.139     0.18
## # ... with 2 more variables: RM_PctEst <dbl>, Y_PctEst <dbl>
```

```
head(pct_moe)
```

```
## # A tibble: 6 x 9
##   D_PctMOE EM_PctMOE F_PctMOE FB_PctMOE LEP_PctMOE LI_PctMOE OA_PctMOE
##      <dbl>     <dbl>    <dbl>     <dbl>      <dbl>     <dbl>     <dbl>
## 1   0.05       0.077    0.061     0.119      0.102     0.11      0.047
## 2   0.039      0.157    0.057     0.067      0.083     0.141     0.047
## 3   0.049      0.196    0.075     0.141      0.125     0.076     0.038
## 4   0.0280     0.091    0.047     0.07       0.047     0.102     0.02
## 5   0.022      0.025    0.035     0.031      0.023     0.031     0.024
## 6   0.023      0.053    0.033     0.063      0.032     0.056     0.031
## # ... with 2 more variables: RM_PctMOE <dbl>, Y_PctMOE <dbl>
```

6b. Percentile

6b.i. Calculation

Add percentiles (an additional "spreadsheet tab") to `comp`, making sure to first sort column names alphabetically. Compute the empirical cumulative distribution function for each of the nine indicator variables. The ECDF can range from 0 to 1, where 1 indicates the largest observed percentage.

```
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)
}
```

6b.ii. Result

`percentile` stores the 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", "_Pctile")
```

Here are the first few lines of `percentile`:

```
head(percentile)
```

```
## # A tibble: 6 x 9
##   D_Pctile EM_Pctile F_Pctile FB_Pctile LEP_Pctile LI_Pctile OA_Pctile
##      <dbl>     <dbl>    <dbl>     <dbl>      <dbl>     <dbl>     <dbl>
```

```
## 1      0.61       1        0.02        0.99        1          0.95       0.33
## 2      0.1        0.92     0.7         0.8         0.91       0.78       0.14
## 3      0.28       0.97     0.33        0.94        0.97       0.99       0.05
## 4      0.34       0.96     0.5         0.96        0.94       0.76       0.14
## 5      0.31       0.39     0.69        0.56        0.45       0.16       0.91
## 6      0.07       0.7      0.290       0.95        0.7        0.3        0.74
## # ... with 2 more variables: RM_Pctile <dbl>, Y_Pctile <dbl>
```

6c. 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 $< -1.5stdev$
1	Below Average	$-1.5stdev \leq$ x $< -0.5stdev$
2	Average	$-0.5stdev \leq$ x $< 0.5stdev$
3	Above Average	$0.5stdev \leq$ x $< 1.5stdev$
4	Well Above Average	x $\geq 1.5stdev$

6c.i. Calculation

Note that we divide *rounded* `PctEst` columns by *unrounded* half-standard deviation breaks.

```
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 <- case_when(p < breaks[2] ~ 0,
                     p >= breaks[2] & p < breaks[3] ~ 1,
                     p >= breaks[3] & p < breaks[4] ~ 2,
                     p >= breaks[4] & p < breaks[5] ~ 3,
                     p >= breaks[5] ~ 4)
  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)
}
```

6c.ii. 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")
```

Here are the first few lines of `score` and `class`:

```
head(score)
```

```
## # A tibble: 6 x 9
##   D_Score EM_Score F_Score FB_Score LEP_Score LI_Score OA_Score RM_Score
##     <dbl>    <dbl>   <dbl>    <dbl>     <dbl>    <dbl>    <dbl>    <dbl>
## 1       2        4       0        4         4        4        2        2
## 2       1        3       2        3         3        3        1        3
## 3       1        4       2        4         4        4        1        3
## 4       1        4       2        4         4        3        1        3
## 5       1        2       2        2         2        1        3        1
## 6       1        2       2        4         2        1        2        2
## # ... with 1 more variable: Y_Score <dbl>
```

```
head(class)
```

```
## # A tibble: 6 x 9
##   D_Class EM_Class F_Class FB_Class LEP_Class LI_Class OA_Class RM_Class
##   <chr>   <chr>    <chr>   <chr>    <chr>     <chr>    <chr>    <chr>
## 1 Average Well Ab~ Well B~ Well Ab~ Well Abo~ Well Ab~ Average  Average
## 2 Below ~ Above A~ Average Above A~ Above Av~ Above A~ Below A~ Above A~
## 3 Below ~ Well Ab~ Average Well Ab~ Well Abo~ Well Ab~ Below A~ Above A~
## 4 Below ~ Well Ab~ Average Well Ab~ Well Abo~ Above A~ Below A~ Above A~
## 5 Below ~ Average  Average Average  Average   Below A~ Above A~ Below A~
## 6 Below ~ Average  Average Well Ab~ Average   Below A~ Average  Average
## # ... with 1 more variable: Y_Class <chr>
```

6d. Composite IPD score

6d.i. Calculation

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

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

6d.ii. Result

Here are the first few records of the composite IPD score that has been appended to `score`:

```
head(score$IPD_Score)
```

```
## [1] 25 23 27 25 16 18
```

7. ACS estimates 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.

Transform percentage estimates and their MOEs into values ranging from 0 to 100 instead of 0 to 1 and round to the tenths place.

```
pct <- pct %>%
  mutate_all(funs(. * 100)) %>%
```

```
  mutate_all(round, 1)
pct_moe <- pct_moe %>%
  mutate_all(funs(. * 100)) %>%
  mutate_all(round, 1)
```

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

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

Rename columns.

```
names(ipd) <- str_replace(names(ipd), "_CE", "_CntEst")
names(ipd) <- str_replace(names(ipd), "_CM", "_CntMOE")
ipd <- ipd %>% mutate(STATEFP = str_sub(GEOID, 1, 2),
                      COUNTYFP = str_sub(GEOID, 3, 5),
                      NAME = str_sub(GEOID, 6, 11),
                      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` and FIPS codes first, the following variables in alphabetical order, and the total IPD score and universes at the end.

```
ipd <- ipd %>% select(GEOID, STATEFP, COUNTYFP, NAME, 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")))
```

At the beginning of processing, we removed 10 census tracts from processing because their populations were equal to zero. Tack these back on to the dataset.

```
slicer <- enframe(slicer, name = NULL, value = "GEOID")
ipd <- plyr::rbind.fill(ipd, slicer)
```

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

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

8. Summary Tables

This section generates a handful of other deliverables, including:

a. Counts by indicator
b. Breaks by indicator

c. Summary by indicator
d. County means by indicator

Replace `-99999` with `NA` for numeric columns to avoid distorting summary statistics.

```
ipd_summary <- ipd
ipd_summary[ipd_summary == -99999] <- NA
```

8a. Counts by indicator

The number of census tracts that fall in each bin. Count census tracts by indicator and bin. Reorder factor levels so that "Well Below Average" appears before "Below Average," and the like.

```
counts <- ipd_summary %>% select(ends_with("Class"))
export_counts <- apply(counts, 2, function(i) plyr::count(i))
for(i in 1:length(export_counts)){
  export_counts[[i]]$var <- names(export_counts)[i]
}
export_counts <- map_dfr(export_counts, `[`, c("var", "x", "freq"))
colnames(export_counts) <- c("Variable", "Classification", "Count")
export_counts$Classification <- factor(export_counts$Classification,
                                       levels = c("Well Below Average",
                                                  "Below Average",
                                                  "Average",
                                                  "Above Average",
                                                  "Well Above Average",
                                                  "NoData"))
export_counts <- arrange(export_counts, Variable, Classification)
export_counts <- export_counts %>%
  spread(Classification, Count) %>%
  mutate_all(funs(replace_na(., 0))) %>%
  mutate(TOTAL = rowSums(.[2:7], na.rm = TRUE))
```

8b. Breaks by indicator

The bin breaks for each indicator. Select the percentage estimate columns, divide by 100 so that they range from 0 to 1, apply the `st_dev_breaks` function to all percentage values, and export the results.

```
breaks <- ipd_summary %>% select(ends_with("PctEst")) %>% mutate_all(funs(. / 100))
export_breaks <- round(mapply(st_dev_breaks, x = breaks, i = 5, na.rm = TRUE), digits = 3)
export_breaks <- as_tibble(export_breaks) %>%
  mutate(Class = c("Min", "1", "2", "3", "4", "Max")) %>%
  select(Class, current_vars())
```

8c. Summary by indicator

Summary statistics of each indicator. Round results to two decimal places.

```
pcts <- ipd_summary %>% select(ends_with("PctEst"))
summary_data <- apply(pcts, 2, description)
export_summary <- as_tibble(summary_data) %>%
  mutate_all(round, 2) %>%
```

```
  mutate(Statistic = c("Minimum", "Median", "Mean", "SD", "Half-SD", "Maximum")) %>%
  select(Statistic, current_vars())
```

8d. County means by indicator

Population-weighted means by county and indicator. For the most accurate statistics, aggregate all counts back to the county level and compute percentages. Counties are noted in the same format as the user supplied to `ipd_counties`.

```
export_means <- dl_counts %>% select(GEOID, ends_with("UE"), ends_with("CE")) %>%
  select(GEOID, sort(current_vars())) %>%
  mutate(County = str_sub(GEOID, 1, 5)) %>%
  select(-GEOID) %>%
  group_by(County) %>%
  summarize(D_PctEst = sum(D_CE) / sum(D_UE),
            EM_PctEst = sum(EM_CE) / sum(EM_UE),
            F_PctEst = sum(F_CE) / sum(F_UE),
            FB_PctEst = sum(FB_CE) / sum(FB_UE),
            LEP_PctEst = sum(LEP_CE) / sum(LEP_UE),
            LI_PctEst = sum(LI_CE) / sum(LI_UE),
            OA_PctEst = sum(OA_CE) / sum(OA_UE),
            RM_PctEst = sum(RM_CE) / sum(RM_UE),
            Y_PctEst = sum(Y_CE) / sum(Y_UE)) %>%
  mutate_if(is.numeric, funs(. * 100)) %>%
  mutate_if(is.numeric, round, 1)
```

9. Export

9a. Append to TIGER/LINE file

Using the arguments supplied in `ipd_county`, download the relevant census tracts and append `ipd` to them. Note that if you are running this program for a lot (perhaps dozens) of counties, then it is wise to uncomment `cb = TRUE` and download generalized shapefiles instead of the most detailed ones.

```
options(tigris_use_cache = TRUE, tigris_class = "sf")
st <- str_sub(ipd_counties, 1, 2)
cty <- str_sub(ipd_counties, 3, 5)
trct <- map2(st, cty, ~{tracts(state = .x,
                               county = .y,
                               #cb = TRUE,
                               year = ipd_year)}) %>%
  rbind_tigris() %>%
  st_transform(., 26918) %>%
  select(GEOID) %>%
  left_join(., ipd)
```

9b. Export files

Results are saved in `outputs`.

```r
st_write(trct, here("outputs", "ipd.shp"), delete_dsn = TRUE, quiet = TRUE)
write_csv(ipd, here("outputs", "ipd.csv"))
write_csv(export_counts, here("outputs", "counts_by_indicator.csv"))
write_csv(export_breaks, here("outputs", "breaks_by_indicator.csv"))
write_csv(export_summary, here("outputs", "summary_by_indicator.csv"))
write_csv(export_means, here("outputs", "mean_by_county.csv"))
```