Technical Reference

Addison Larson

2/11/2019

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. 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.

Project file structure

The easiest way to run this script is to download the entire repository from Github (link). The script uses relative file paths but it looks for subfolders in the following structure:

Insert file structure here.

Script components (to be fixed later)

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

Script outputs

1. `ipd.csv`, the primary output
   - Estimated count of residents for each indicator (`CntEst`)
   - Margin of error (MOE) associated with the count (`CntMOE`)
   - Estimated percentage of residents for each indicator (`PctEst`)
   - MOE associated with the percentage (`PctMOE`)
   - Rank (percentile) of the percentage as compared to the study area, ranging from 0 to 1 (`Rank`)
   - IPD score of the percentage, ranging from 0 to 4 (`Score`)
   - Text explanation and equivalent of the IPD score (`Class`)
   - Overall IPD score (`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

1a. 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(tidycensus); library(tidyverse); library(here); library(summarytools)
```

1b. 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"
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
```

1c. Year

The data download year.

```
ipd_year <- 2017
```

1d. Functions

Load custom functions.

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 in other scripts.

```
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)
}
```

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. **Note** that all minima are coerced to equal zero and that if the first bin break is negative (this happens when our data have a large spread), then it is coerced to equal 0.001.

```
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)
}
```

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))
}
```

Summarize data

The requested summary tables of IPD data call for more than `base::summary` exports. `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)
}
```

Squared difference for variance replicates

```
sqdiff_fun <- function(v, e) (v - e) ^ 2
```

2. 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, 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.

2a. Download variance replicates from Census website

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

```
nj_url <- paste0("https://www2.census.gov/programs-surveys/acs/replicate_estimates/",
                 ipd_year, "/data/5-year/140/B02001_34.csv.gz")
pa_url <- paste0("https://www2.census.gov/programs-surveys/acs/replicate_estimates/",
                 ipd_year, "/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))
```

2b. Combine and format downloads

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:"))
```

3. Variance replicate table processing

3a. 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. Additional guidance on computing variance replicates is available at this (link).

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

3b. 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
```

4. ACS estimates download

4a. Fields

Fields for downloads from the ACS API were discussed in Section 1b.

4b. 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 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.

```
s_counts <- get_acs(geography = "tract", state = c(34,42),
                    output = "wide", year = ipd_year,
                    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", year = ipd_year,
                    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)
```

4c. 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 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", year = ipd_year,
                   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", year = ipd_year,
                    variables = c(F_P = female_percent)) %>%
  rename(F_PE = F_P, F_PM = DP05_0003PM) %>%
  select(-NAME)
```

4d. Combine and format downloads

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.

```
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) %>%
  mutate_if(is.numeric, funs(. / 100))
```

Exception 4.1

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)
}
```

Exception 4.2

Half-standard deviations serve as the classification bins for IPD scores, and the overall number of 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`:

```
head(dl_counts)
```

```
## # A tibble: 6 x 37
##   GEOID   LI_UE LI_UM LI_CE LI_CM  F_UE  F_UM  F_CE  F_CM  D_UE  D_UM  D_CE
##   <chr>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 340210~  2144   354  1495   342  2164   354   932   170  2144   354   283
## 2 340210~  1160   265   509   201  1194   271   638   158  1194   271    79
## 3 340210~  1603   280  1329   262  1603   280   807   148  1603   280   147
## 4 340210~  4513   495  1928   507  4513   495  2327   344  4513   495   449
## 5 340210~  2591    19   245    81  2591    19  1378    93  2591    19   249
## 6 340210~  5584   469   778   320  5584   469  2787   304  5584   469   328
## # ... with 25 more variables: D_CM <dbl>, OA_CE <dbl>, OA_CM <dbl>,
## #   LEP_UE <dbl>, LEP_UM <dbl>, LEP_CE <dbl>, LEP_CM <dbl>, OA_UE <dbl>,
## #   OA_UM <dbl>, EM_UE <dbl>, EM_UM <dbl>, EM_CE <dbl>, EM_CM <dbl>,
## #   Y_CE <dbl>, Y_CM <dbl>, FB_UE <dbl>, FB_UM <dbl>, FB_CE <dbl>,
## #   FB_CM <dbl>, RM_UE <dbl>, RM_UM <dbl>, RM_CM <dbl>, Y_UE <dbl>,
## #   Y_UM <dbl>, RM_CE <dbl>
```

```
head(dl_percs)
```

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

5. Calculations

For all nine indicators, Section 4 computes:

a. Percentages and their MOEs
b. Rank (Percentile)
c. IPD Score and Classification
d. Total 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 1,379 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()))
```

5a. Percentages and percentage 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$uni_est[,c])
  pct_matrix <- cbind(pct_matrix, pct)
  moe <- NULL
  for (r in 1:length(comp$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)
}
```

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, 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")
```

Exception 5.2

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] <- 0.1
```

Exception 5.3

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)
```

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>
```

5. 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")
```

Here are the first few lines of `percentile`:

```
head(percentile)
```

```
## # A tibble: 6 x 9
##   D_Rank EM_Rank F_Rank FB_Rank LEP_Rank LI_Rank OA_Rank RM_Rank Y_Rank
##    <dbl>   <dbl>  <dbl>   <dbl>    <dbl>   <dbl>   <dbl>   <dbl>  <dbl>
## 1   0.61    1     0.02     0.99     1       0.95    0.33    0.69  0.76
## 2   0.1     0.92  0.7      0.8      0.91    0.78    0.14    0.84  0.98
## 3   0.28    0.97  0.33     0.94     0.97    0.99    0.05    0.83  0.97
## 4   0.34    0.96  0.5      0.96     0.94    0.76    0.14    0.75  0.87
## 5   0.31    0.39  0.69     0.56     0.45    0.16    0.91    0.18  0.570
## 6   0.07    0.7   0.290    0.95     0.7     0.3     0.74    0.62  0.3
```

5c. 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	$x < -1.5stdev$
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	$x \geq 1.5stdev$

Calculation

```
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)
}
```

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>
```

Compute total IPD score across all nine indicators

Calculation

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

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

Result

Here are the first few lines of `score`:

```
head(score)
```

```
## # A tibble: 6 x 10
##   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 2 more variables: Y_Score <dbl>, IPD_Score <dbl>
```

6. 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(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.

```
ipd <- ipd %>% 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")))
```

Tack unwanted columns back onto the final 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, .)))
```

6. Summary Tables

```
# Replace -99999 with NA for our purposes
ipd_summary <- ipd
ipd_summary[ipd_summary == -99999] <- NA
# Count of tracts that fall in each bin
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]]$var <- names(export_counts)[i]
}
export_counts <- map_dfr(export_counts, `[`, c("var", "x", "freq"))
# Format export
colnames(export_counts) <- c("Variable", "Classification", "Count")
export_counts$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)) %>%
  mutate_at(vars(Variable), funs(case_when(. == "D_Class" ~ "Disabled",
                                           . == "EM_Class" ~ "Ethnic Minority",
                                           . == "F_Class" ~ "Female",
                                           . == "FB_Class" ~ "Foreign-Born",
                                           . == "LEP_Class" ~ "Limited English Proficiency",
                                           . == "LI_Class" ~ "Low-Income",
                                           . == "OA_Class" ~ "Older Adults",
                                           . == "RM_Class" ~ "Racial Minority",
                                           . == "Y_Class" ~ "Youth")))
# Bin break points
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())
# Minimum, median, mean, standard deviation, maximum
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())
# Population-weighted county means for each indicator
export_means <- dl_counts %>% select(GEOID, ends_with("UE"), ends_with("CE")) %>%
  select(GEOID, sort(current_vars())) %>%
  mutate(County = str_sub(GEOID, 3, 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) %>%
  mutate_at(vars(County), funs(case_when(. == "005" ~ "Burlington",
                                         . == "007" ~ "Camden",
                                         . == "015" ~ "Gloucester",
                                         . == "021" ~ "Mercer",
                                         . == "017" ~ "Bucks",
                                         . == "029" ~ "Chester",
                                         . == "045" ~ "Delaware",
                                         . == "091" ~ "Montgomery",
                                         . == "101" ~ "Philadelphia")))
```

7. Export

```
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"))
```